Institut für Kernphysik Frankfurt
Fachbereich Physik

Firmware Development for the TRD Data Processing Board Prototype

Master's Thesis

David Schmidt

November 18, 2019

Contents

Chapter 1

The Compressed Baryonic Matter Experiment

1.1 The Experiment

The investigation of nuclear matter and the associated phase diagram has been the focus of many experiments in high energy physics. The phase diagram of nuclear matter is depicted in Fig. 1.1. The goal of many of the experiments is to scan different regions of the phase diagram. For example, the experiments at the LHC (Large Hadron Collider) examine the region of very low baryon chemical potential μ_B and extremely high temperature T. In contrast, the Compressed Baryonic Matter (CBM) experiment at FAIR/GSI (Facility for Antiproton and Ion Research at GSI Helmholtzzentrum für Schwerionenforschung) plans to explore the region of the phase diagram of nuclear matter at high μ_B and moderate temperatures. Such conditions are for example expected within neutron stars. The SIS100 (SchwerIonen-Synchrotron)

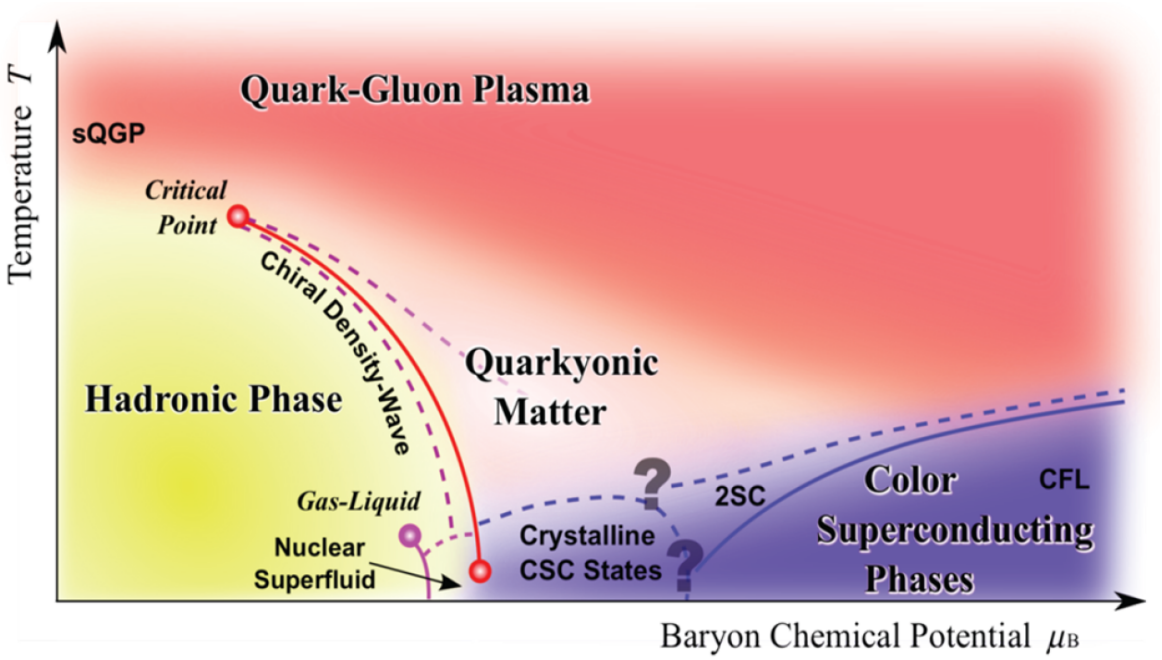

Figure 1.1: Depiction of the phase diagram of nuclear matter as predicted by Fukushima and Hatsuda [16].

accelerator at FAIR is very well suited to generate the required very high net-baryon densities [14]. The high luminosity of the accelerator allows the measurement of particles with very low production cross sections, such as multi-strange (anti-)hyperons, thermal dileptons and charm particles [9]. The CBM experiment will measure these particles with unprecedented precision and statistics. To achieve the targeted precision, efficient background suppression and very high interaction rates are essential, which poses a significant challenge to the data taking to keep the data storage budget realistic and achievable. Events contributing to the desired observables have to be selected and filtered through online track reconstruction. As the data taking in CBM is done without a hierarchical trigger system, this requires self-triggered readout electronics, high-bandwidth data processing and high-performance algorithms to keep data storage space manageable.

Aside from the highly performant readout chain the experiment needs a multitude of radiation hard detectors to accurately track and identify the particles. Figure 1.2 shows the experimental setup of CBM for the SIS100. The CBM experiment employs a modular structure to accommodate the different physics cases. There are three different setups: The muon setup, the electron setup and the hadron setup.

In closest proximity to the target is the Micro-Vertex Detector (MVD). It consists of 4 layers of Monolithic Active Pixel Sensors (MAPSs). The layers are located 5 cm to 20 cm downstream of the target. The MVD will be used to identify open charm particles through the position of their decay vertex [15].

The next detector is the Silicon Tracking System (STS) 30 cm to 100 cm downstream of the target. It will provide accurate track and momentum measurements of charged particles with a momentum resolution of about $\Delta p/p = 1.5\,\%$ [5]. The MVD and the STS are positioned

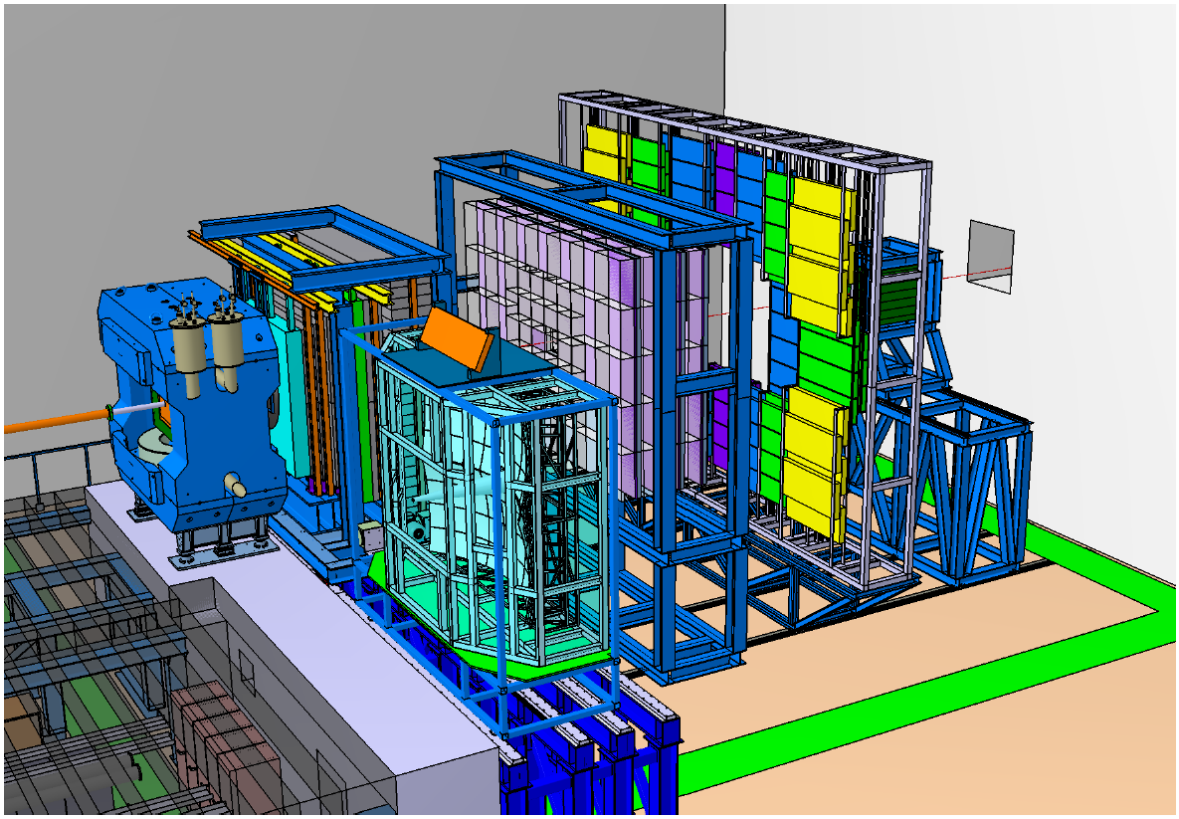

Figure 1.2: 3D model of the CBM experimental setup [9].

inside the magnetic field of the superconducting dipole magnet. The magnet will deliver an integrated magnetic field of 1 Tm [3].

The STS is followed by the Ring-Imaging Cherenkov Detector (RICH) or the Muon Chamber System (MUCH). The RICH provides electron identification capabilities through the measurement of Cherenkov radiation. It is built with a 1.7 m long gas radiator and two arrays of mirrors and photo-multipliers. It is placed 1.6 m downstream. The RICH is used in the electron setup [4].

The MUCH replaces the RICH in the muon setup. It tracks muons through a hadron absorber system. The tracking is provided by several layers of Gas Electron Multiplier (GEM) detectors, as well as Resistive Plate Chambers (RPCs). The detector system is built as compact as possible to reduce the contribution from the decay of mesons into muons [7].

The next detector along the beam-line is the Transition Radiation Detector (TRD) located 4.1 m to 5.9 m downstream of the target. The TRD provides particle identification and tracking information [9].

The Time Of Flight (TOF) detector is positioned about 6 m downstream. It consists of an array of Multi-gap Resistive Plate Chambers (MRPCs) and will be used to identify hadrons via their time-of-flight. The required time resolution of the TOF detector is in the region of 80 ps [6].

To measure direct photons and photons from the decay of neutral mesons a Electromagnetic Calorimeter (ECAL) is utilized. The ECAL is a modular *shashilk* type calorimeter consisting of 140 layers of lead and scintillator sheets. The modules can be placed at variable distances from the target [15].

The Projectile Spectator Detector (PSD) will measure the centrality and the reaction plane orientation of the collision through the number of non-interacting nucleons from a projectile nucleus in a collision. The PSD is a modular lead-scintillator calorimeter comprised of 44 modules with 60 alternating layers of lead and scintillator [8].

1.2 The CBM Transition Radiation Detector

As the subject of this thesis is the development of the prototype Data Processing Board (DPB) firmware for the TRD Front-End Electronics (FEE), the TRD detector will receive a more detailed description in this section.

Figure 1.3 depicts the working principle of the CBM-TRD. The detector is based on a Multi-Wire Proportional Chamber (MWPC) as the Read-Out Chamber (ROC) and a radiator to produce Transition Radiation (TR) photons. Sufficiently fast particles will produce TR-photons inside the radiator. In the given particle momentum ranges only electrons reach high enough γ-factors to produce TR-photons, any other charged particle will be too massive. The TR-photons deposit additional energy in the ROC gas volume, filled with a xenon gas mixture, which has a high absorption cross-section for the TR-photons in the given energy range [22].

Aside from the TR-photons, passing particles generate charge via primary ionization processes in the gas according to their specific energy loss $\langle dE/dx \rangle$, which is collected inside the ROC. The specific energy loss further improves the particle separation between pions and electrons. Hence, electrons that do not produce TR-photons can still be identified. Finally, the measurement of the specific energy loss of light nuclei can be used for their identification.

The TRD will be used in all three configurations of the CBM experiment. In the electron setup, the TRD and the RICH provide the electron identification to accurately measure low

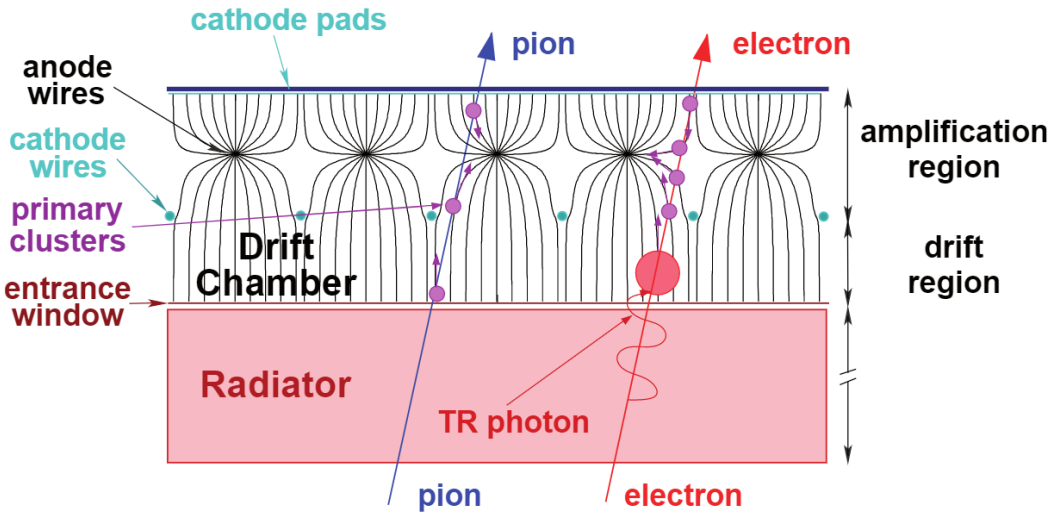

Figure 1.3: Schematic drawing of the working principle of the CBM-TRD. Adopted from [9], originally from [25].

and intermediate dielectrons. The STS will provide the tracking information and the MVD is used to reject γ-conversions produced in the target.

In the muon setup the TRD acts as the last tracking detector. In this setup a carbon absorber is placed behind the STS followed by the four layers of the MUCH detector with a iron absorber after each layer. The TRD is behind the last absorber layer, thus only muons will reach the TRD.

The hadron setup only consists of the MVD, STS, TRD and the TOF. Neither the RICH nor the MUCH will be used. The MVD and the STS will be responsible for tracking and secondary vertex detection. TRD and TOF will provide the hadron identification. The TRD in this setup supports the TOF measurement through the $\langle dE/dx \rangle$ information.

1.3 The CBM Readout Chain

The high interaction rate of the CBM experiment requires a high performance Data Acquisition (DAQ) process with a high data throughput. The expected event rate for the experiment is up to 10 MHz for a beam intensity of 10^9 ions/s at a target interaction rate of 1 %. The upper limit for the total data rate is set by the hadron setup at an average interaction rate of 5×10^6/s with an average event size of 48.7 kByte. Including an estimated dark-rate of about 10 % leads to an upper limit for the total data rate of 270 GByte/s [13]. Archiving the data at this rate is beyond the scope of the readout chain, therefore online event selection has to reduce the archiving rate by a factor of about 100.

Achieving this reduction requires efficient and fast algorithms in hardware and software to reject background events. The high-performance computing cluster in the GSI GreenIT cube will be responsible for the bulk of the online event selection task. The most performance intensive task in the event reconstruction is the track finding, which will be based on a Cellular Automaton and a Kalman Filter. To help the online event selection some feature extraction can be done inside an FPGA (Field-Programmable Gate Array) during the data acquisition. Figure 1.4 shows the schematic view of the CBM readout concept. At the lowest level of the readout chain are the Front-End Boards (FEBs), which carry the readout ASICs (Application-Specific Integrated Circuits). The FEBs connect via e-links to the Common

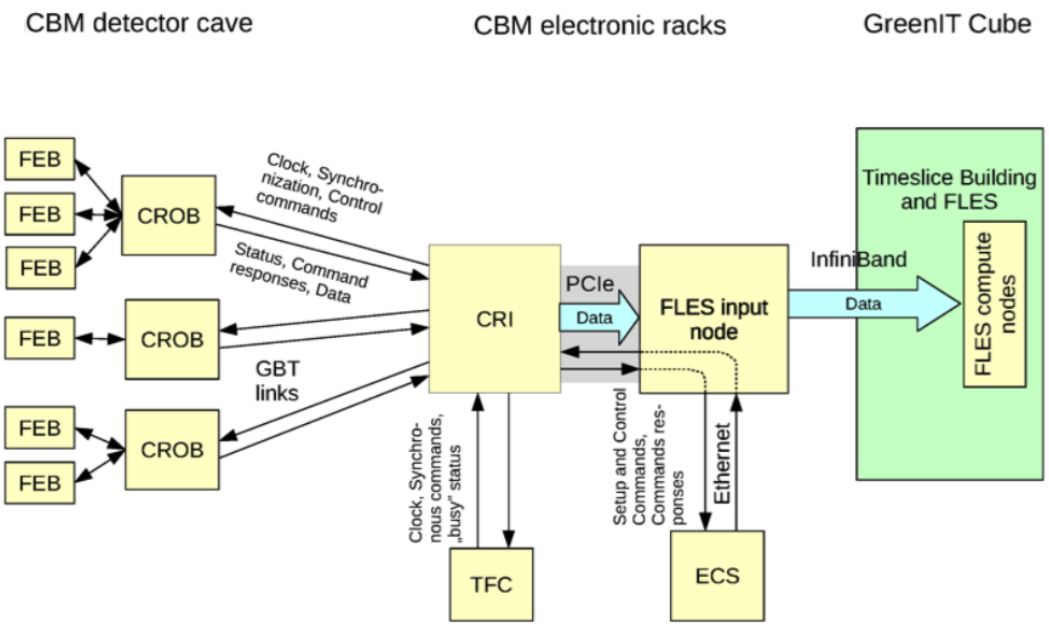

Figure 1.4: Schematic depiction of the CBM readout chain. Taken from [9].

Read-Out Board (CROB) based on the GBTx ASIC developed at CERN [24]. The CROB aggregates multiple e-links into optical GBT links and transmits the ASIC data out of the CBM detector cave to the Common Read-Out Interface (CRI) boards in the CBM electronics racks. Additionally, the CROB provides the downlinks from the CRI boards to the ASICs to deliver the proper clock, initialize the link synchronization and transmit control commands. The CRI is forseen to be based on a Xilinx Virtex UltraScale FPGA. The FPGA stores the detector data inside timeslices and transmits them to the First-Level Event Selection (FLES) input nodes. From the input nodes the data will be transported to the FLES compute node in the GreenIT Cube over InfiniBand [10].

1.3.1 The First-Level Event Selector

The free-running environment of the CBM experiment with self-triggered detectors delivers no inherent event separation. Therefore, instead of event building the First-Level Event Selection (FLES) performs timeslice building. A timeslice is the aggregation of the data from all the connected input links in a given time period. The timeslices are then distributed to the compute nodes. In order to allow for efficient timeslice building in the FLES the data is partitioned into µSlices in a previous processing stage. µSlices are a container with a globally defined structure. The stored data inside the container is required to be from a defined time frame, which is shared between all subsystems. Furthermore, the data content of a µSlice has to be self-contained [10].

Each µSlice is constructed with a global descriptor followed by a block of subsystem specific data. The 32 byte descriptor contains the start time of the µSlice, which allows the timeslice building to combine the µSlices of all subsystem into one processing interval.

Table 1.1 lists the content of the µSlice descriptor. For a more detailed description of the data fields see [18].

The µSlices are built during the data processing stage inside the FPGA and transmitted to the FLES through the First-Level Interface Module (FLIM). The packaging of the data

Table 1.1: Description of the data fields in the µSlice descriptor. Table from [18]

name	size	description
`hdr_id`	8 bit	Descriptor format identifier. Fixed to 0xDD.
`hdr_ver`	8 bit	Descriptor format version. Set to 0x01 for this version.
`eq_id`	16 bit	Equipment identifier. Specifies the FLES input link.
`flags`	16 bit	Status and error flags.
`sys_id`	8 bit	Subsystem identifier. Specifier of the CBM subsystem that has generated this µSlice (0x40 for the TRD).
`sys_ver`	8 bit	Subsystem format version. This specifies the format of the microslice data content.
`idx`	64 bit	Start time of the µSlice in ns since global time zero.
`crc`	32 bit	CRC32-C checksum of the data content.
`size`	32 bit	Content size. This is the size (in bytes) of the µSlice data content.
`offset`	64 bit	Offset in data buffer.

into µSlices is done via the `tlast` and `tkeep` signals. The `tlast` flag indicates that the current data word is the last one in the µSlice. By asserting `tlast`, the current µSlice is finished and a new one begins as soon as the first word is accepted. The `tkeep` signal indicates the valid bytes of the last word. The `tkeep` flag is only valid if `tlast` is asserted.

Handshaking between the user logic and the FLIM is done via the `tvalid` and `tready` signals. Any data that has to be transmitted to the FLIM must be marked with the `tvalid` flag set. The FLIM accepts the data by asserting `tready`. If `tvalid` and `tready` are asserted in the same clock cycle the data is transferred to the FLIM.

1.3.2 The Prototype TRD Readout Chain

Figure 1.5 illustrates the prototype TRD readout chain. The readout chain at the current development stage is based on the SPADIC 2.2 (Self-triggered Pulse Amplification and Digitization asIC) as the readout ASIC connected directly to the Data Processing Board (DPB). The DPB is an AFCK (AMC-FMC Carrier Kintex) board. The AFCK is connected to the FLES Interface Board (FLIB). The FLIB acts as the FLES input node, where the µSlices

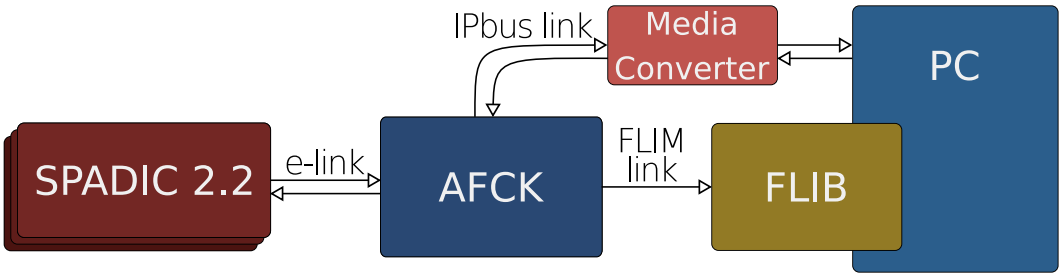

Figure 1.5: Illustration of the prototype TRD readout chain.

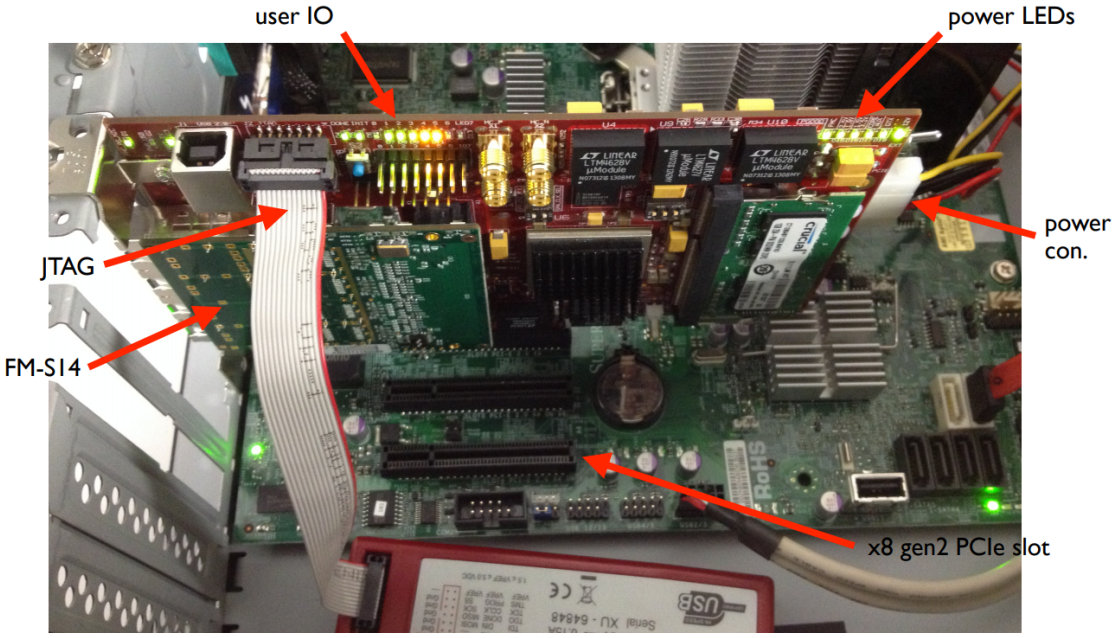

Figure 1.6: Photo of the FLIB connected to a x8 gen 2 PCIe slot. Taken from [17].

from the DPB are aggregated into timeslices. The FLIB is located inside the DAQ-computer and is connected via PCI express (PCIe). Figure 1.6 shows a photo of the FLIB. The control and configuration of the DPB and the SPADIC is done via IPbus, which requires an ethernet connection. The ethernet connection is established with an optical cable from the AFCK to the media converter and from there a standard ethernet cable connects to the DAQ-PC.

AFCK

The AFCK was developed by Grzegorz Kasprowicz from the Warsaw University of Technology. It is the successor of the AFC (AMC FMC Carrier) with a Xilinx Kintex-7 FPGA instead of a Xilinx Artix-7 FPGA. The board features a very flexible clock circuit, which allows the connection of any clock source to any clock input. Detailed information on the specifications can be found at [20].

The card features two FPGA Mezzanine Card (FMC) High-Pin Count (HPC) connectors to provide flexible connectivity to the FPGA. In the prototype TRD readout the two connected FMCs are the gDPB FMC, which features three KEL40 connectors to provide connectivity to the SPADICs as well as an SFP connector over which an ethernet connection is established (via the media converter). Aside from the SFP connector it is possible to connect to the ethernet via a modified SATA-to-SFP cable or via a µTCA crate.

The ethernet connection is necessary to communicate with the DPB over IPbus. The IPbus protocol is used in the prototype readout chain but will be replaced by a communication protocol over the PCIe bus in the final experiment.

The other FMC is the nDPB FMC which provides the FLIM link to the FLIB input node. Figure 1.7 shows the AFCK with the two connected FMCs.

IPbus Protocol

The IPbus protocol was developed at CERN to provide a flexible ethernet-based control system for IP-aware hardware [23]. The IPbus suite provides a extensible software interface

to control end-user hardware, which has the IPbus firmware integrated. The software provides hardware access through the Hardware Access Library (HAL) with a C++/Python API. The connection to the hardware is established via a connection file that contains the IP address of the device. Writing and reading of registers in the hardware is done via an address table. Each register has a unique address composed of the node address and the mask. Each node consists of 32 bits, and the mask specifies which of the 32 bits are associated with the register. Each register can be assigned a read, write or read-write permission. Furthermore, it is possible to directly read and write memory blocks from the IPbus software. In the FPGA the memory blocks are implemented as FIFOs.

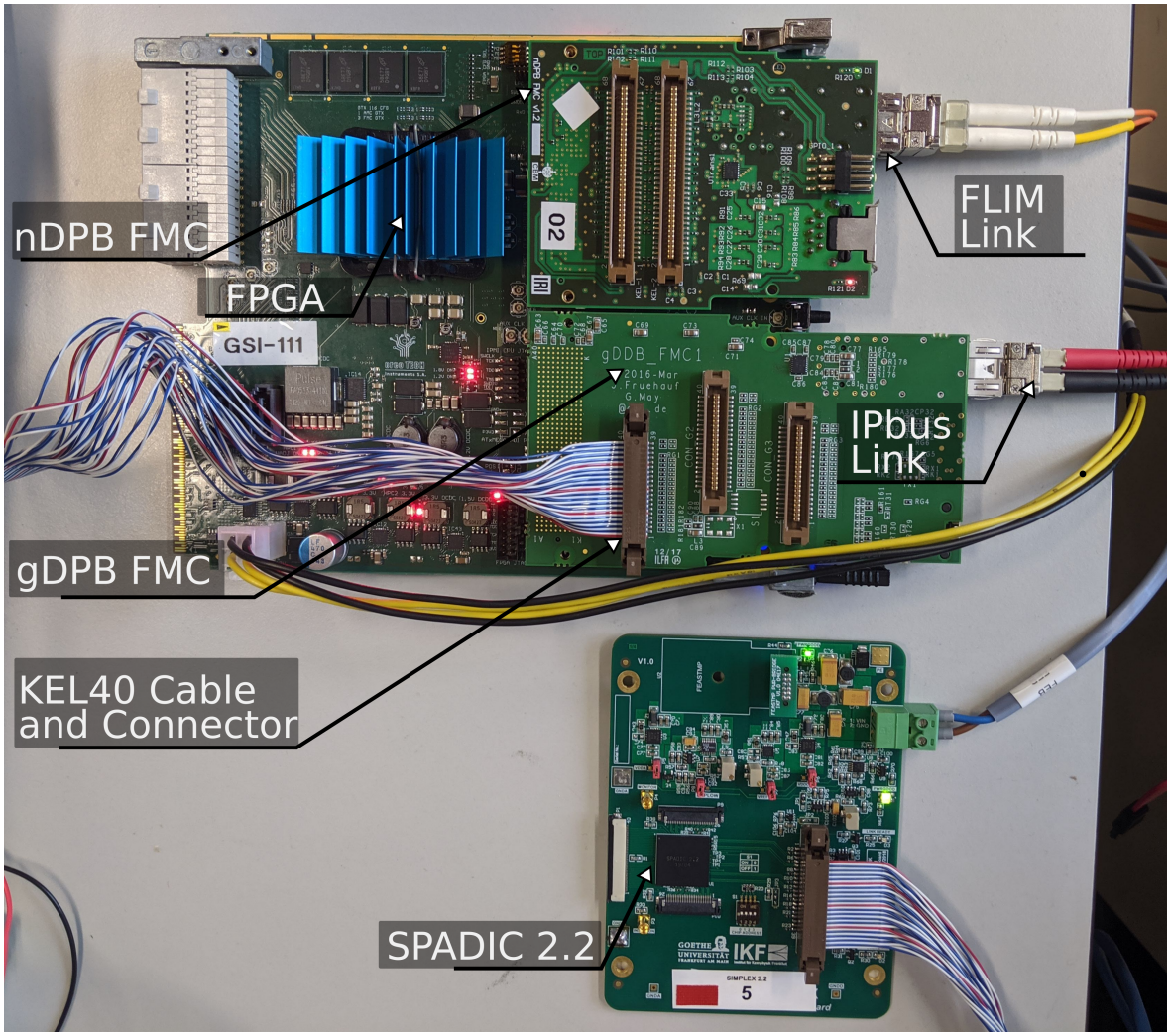

Figure 1.7: Photo of the AFCK with the gDPB FMC and the nDPB FMC connected to a SPADIC 2.2.

Chapter 2

The SPADIC 2.2

2.1 SPADIC 2.2 Specifications

The SPADIC (**S**elf-triggered **P**ulse **A**mplification and **D**igitization as**IC**) is the readout chip for the CBM TRD. Earlier versions of the SPADIC were developed by Tim Armbruster from the University of Heidelberg in the working group of Peter Fischer [11]. The work on the SPADIC was later continued by Michael Krieger from the same working group. Michael Krieger also developed the current version 2.2 of the SPADIC. Below is a short summary of the SPADIC specification and features, for a more detailed description see [2, 11, 21].

The SPADIC 2.2 was specifically designed to run in the free-running environment of the CBM experiment and is matching in particular the TRD characteristic. However, due to its high flexibility the chip is also an interesting option for other readout applications. The ASIC provides an oscilloscope-like behavior, in the sense that it samples the signal generated in the detector over up to 32 time samples. The sampling rate is 16 MHz, which amounts to 62.5 ns per time bin. Each sample has an effective ADC resolution of 8 bits, which was found to be a good trade-off between performance and costs. A good amplitude resolution is of great importance for the energy resolution and the spatial resolution of the TRD.

Figure 2.1 depicts a conceptional block diagram of the SPADIC. Each SPADIC features 32 channels, which are internally split into two 16-channel groups. Each channel amplifies the incoming signals in a Charge Sensitive Amplifier (CSA) and samples the amplified output in a continuously running ADC. Additionally, a Digital Signal Processor (DSP) can be programmed to pre-process the signal to improve ion tail cancellation or to discriminate multi-hits. A hit detection logic checks if the digitized signal satisfies the hit detection conditions, e.g. a steep slope and a value above a programmed threshold. If a hit has been detected the digitized signal is stored for the message building.

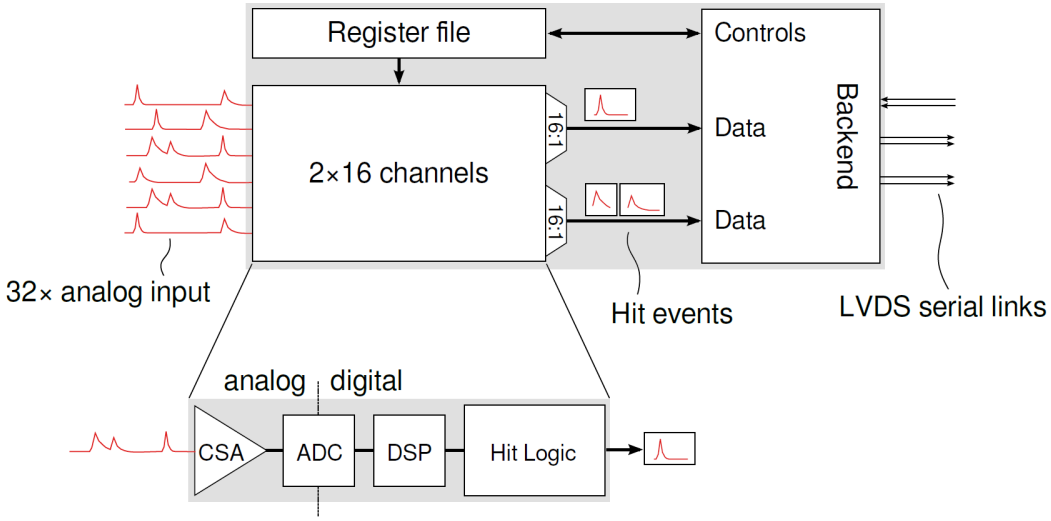

Figure 2.1: Simplified block diagram of the SPADIC [9]. The backend block is based on the SMX e-link interface.

The message building can be configured to build the hit message with a specified set of samples, which can be chosen from the 32 available samples. This allows to decrease the information in each hit in favor of more total hits transmitted, which is necessary for the high data rates planned for the CBM experiment. The hit messages receive a timestamp for event reconstruction and some other meta-data and are then serialized in two e-links compatible with the GBTx protocol, with one e-link for each 16-channel group.

The messages in each e-link are ordered in time. The time ordering is done by a special time-ordering FIFO, depicted in Fig. 2.2. The ordering ensures that all hit message of one epoch (wraparound of the local timestamp) are framed between two epoch markers to allow unambiguous time reconstruction. To achieve this, it is vital that the ordering logic correctly recovers from a FIFO overflow.

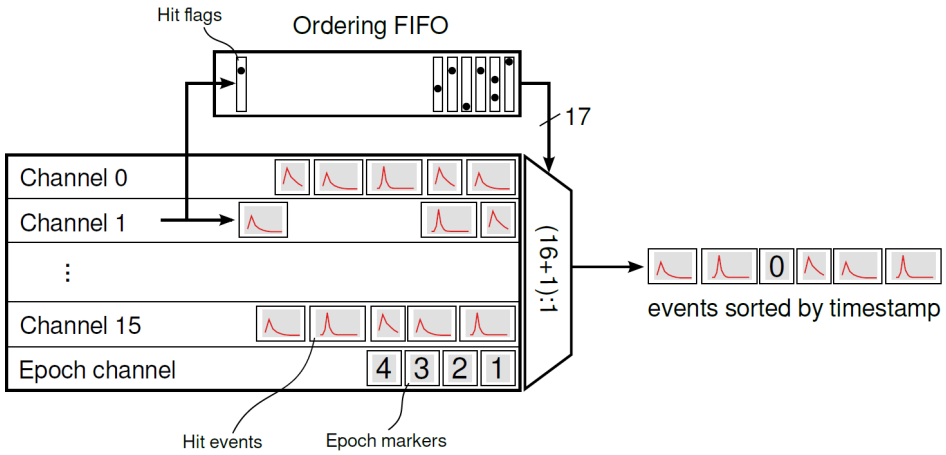

Figure 2.2: Principle of the SPADIC time-ordering FIFO. The time-ordering FIFO is implemented for each 16-channel group. Taken from [9].

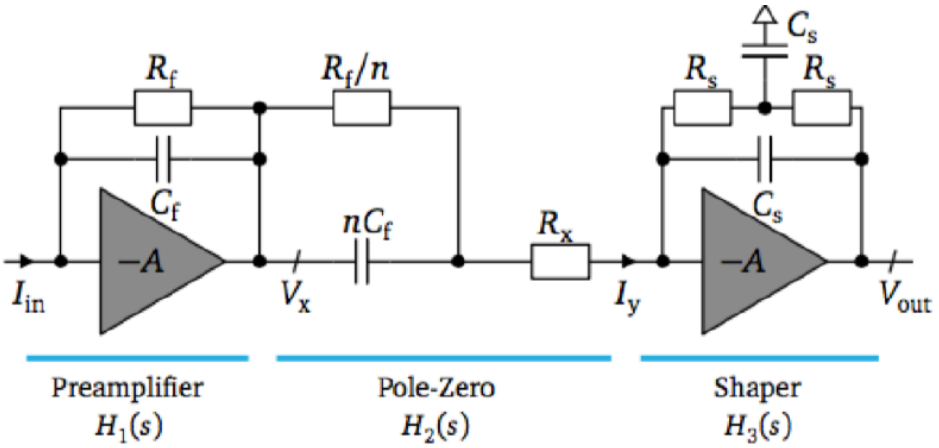

Figure 2.3: Schematic of the SPADIC analog front end, with the CSA, the pole zero correction and the shaping stage [9].

The principle of the analog front end of the chip is shown in Fig. 2.3. It contains the CSA, a pole-zero correction and a signal shaper. The resulting pulse shape can be described with

$$f(t) = A \cdot \frac{t}{\tau} \cdot \exp\left(-\frac{t}{\tau}\right) \quad (\text{for } t \geq 0),$$

with the peaking time τ. The peaking time was decreased from 240 ns in SPADIC 2.0 to 120 ns in SPADIC 2.2 to better match the detector signal collection time and attain the full load information in fewer samples. Additionally, the faster shaping time improves the signal reconstruction capabilities for multi-hits. However, a switchable second shaping stage was implemented to regain the 240 ns peaking time. The resulting pulse shape becomes, given $\tau = 120\,\text{ns}$,

$$g(t) = A \cdot \left(\frac{t}{\tau}\right)^2 \cdot \exp\left(-\frac{t}{\tau}\right) \quad (\text{for } t \geq 0).$$

The ADC is a custom pipelined ADC based on current memory cells. It features 8 pipeline stages, which produce a 9 bit digitized signal. However, the effective resolution is only roughly 8 bits due to differential nonlinearities.

Furthermore, each channel can be triggered in three different ways: Through a forced external trigger, a self-trigger that satisfies the hit condition or a force trigger an adjacent channel via the Forced Neighbor Readout (FNR). The FNR is an important feature to increase the spatial resolution of the TRD without recording a lot of noise hits. A physical hit in the TRD can induce charges in multiple pads. The combined charge information improves the positional resolution significantly. However, the signal amplitude in the pads neighboring the pad where the hit occurred can be too small to generate a self-trigger. The neighbor logic allows the recording of these signals without lowering the thresholds.

The neighbor logic allows a flexible configuration in accordance to the pad plane design of the TRD. The trigger information can be passed on through the channel groups as well as from one chip to another.

2.2 SMX Communication Protocol

The SPADIC communication protocol is adopted from the STS-MUCH-XYTER (SMX) ASIC [19]. The communication is based on 60 bit downlink frames and 30 bit uplink frames. Figure

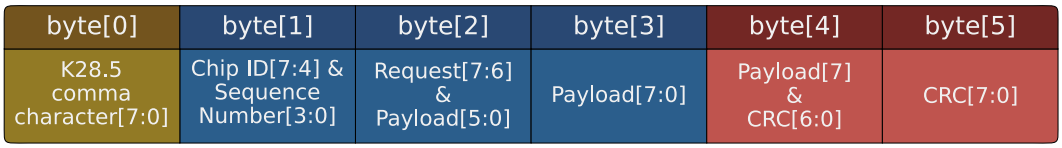

Figure 2.4: Structure of a regular downlink control frame. The downlink frame consists of 6 bytes before 8b10b encoding. The first byte is always a K28.5 comma character to properly synchronize the command. Followed by 3 bytes and 1 bit, which are determined by the user, and finalized by 15 bits of CRC (Cyclic Redundancy Check) for error detection.

2.4 shows the structure of a regular downlink control frame. The downlink frames consist of a K28.5 comma-character and 5 data characters. The K28.x comma-characters are control symbols used for the synchronization of a bit-stream in the 8b10b encoding. Each character is 1 byte long before the 8b10b encoding and 10 bit after. The chip address in byte⟨1⟩ allows to communicate with a specific ASIC, which has the corresponding address on the FEB. The chip address supports values from 0 to 7, for up to eight chips on an FEB and value 15 to broadcast a command to each chip connected on the downlink. The request type in byte⟨2⟩ specifies the operation which the ASIC should perform. The payload is split between byte⟨2⟩ and byte⟨3⟩ and one bit of byte⟨4⟩.

The request type no_op with code 0x0 marks the no operation request. To write a register on the ASIC, the request type WRaddr with code 0x1 has to be sent. The payload is the address of the register. This frame has to be followed by a WRdata request with code 0x2 with the data to be written into the register in its payload. The sequence number of the WRdata frame has to be larger by one compared to the sequence number in the WRaddr frame, otherwise it will not be accepted. The WRaddr stays valid for consecutive WRdata requests but will be overwritten by an RDdata request. To read a register, the request type RDdata with code 0x3 carrying the register address in the payload has to be sent.

The structure of the 30 bit uplink response frames is shown in Fig. 2.5. The response for WRaddr and WRdata requests is the standard acknowledge frame with the prefix `100`. It contains the status of the acknowledgement, the 4 bit sequence number of the corresponding control request, a Config Parity (CP) bit, a 4 bit status value and 4 CRC bits.

The acknowledge status can be:

- `01`: acknowledged,
- `10`: not acknowledged,
- `11`: alert → check the 4 bit status value.

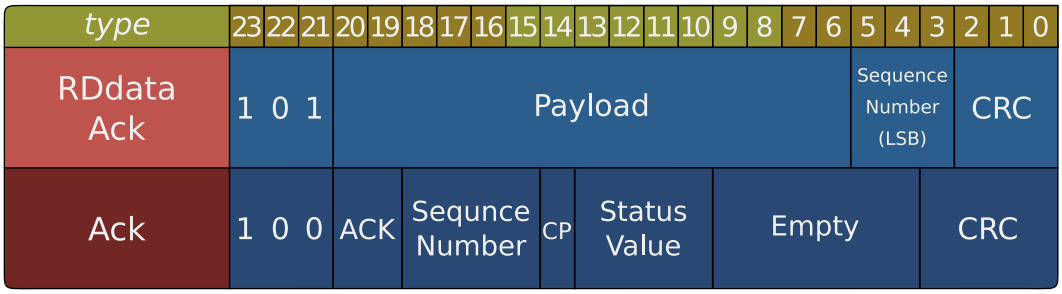

Figure 2.5: Structure of the response uplink frames. There are two types of response frames. One for the read data requests (RDdata Ack) that includes the register content for the requested register address. The other for the write address and write data requests.

And the status values for an alert acknowledge are:

- bit ⟨1⟩: Sync alert → downlink comma character arrive out-of-sync,
- bit ⟨2⟩: Incorrect sequence number in downlink,
- bit ⟨0⟩,⟨3⟩: not used in the SPADIC.

The RDdata request acknowledge with the prefix `101` contains the register content in the payload and the 3 least significant bits of the sequence number. The sequence number of the corresponding control frame is returned in the response frame to be able to match the acknowledge status to the original command.

Besides the regular uplink and downlink frames there are three additional special frames which are used during the link initialization: The Start-Of-Synchronization (SOS) frame, the End-Of-Synchronization (EOS), and the K28.1 comma-character. The SOS and EOS frames use a DC-balanced code with more than 5 consecutive 1's/0's, which is not allowed in 8b10b encoding but allows for an easy detection of the frames even without established synchronization. The SOS frame consists of 10 consecutive 1's followed by 10 consecutive 0's and the EOS frame consists of 6 1's, 4 0's, 4 1's and 6 0's:

SOS: `1111_1111_1100_0000_0000`,
EOS: `1111_1100_0011_1100_0000`.

Link Initialization Sequence

To get a stable link synchronization the DPB has to set the proper downlink clock phase and uplink data delays. To calculate the downlink clock phase, the DPB sends an SOS frame to each ASIC on the FEB. When the ASIC detects the SOS frame it responds with an SOS frame to the DPB. When all the ASICs have responded with an SOS frame the DPB switches to sending K28.1 comma characters. If the ASICs detect the K28.1 comma character, they will respond with a K28.1 comma character, otherwise they will send an SOS frame. Now the DPB monotonously changes the clock phase to find the clock edges. Once the clock edges have been found, the DPB fixes the clock phase to the center between two clock edges between which correct transmission was assured. This step is repeated to find the correct data delays. With the delays set, the EOS frame is sent to finish the synchronization procedure.

2.3 The SPADIC Message Protocol

Figure 2.6 shows the structure of the SPADIC hit frames. A usual hit message consists of a Start-Of-Message (SOM) frame, followed by several Raw DAta (RDA) frames and is closed by a End-Of-Message (EOM) frame.

The SOM frame starts with the prefix `001` and contains the 4 bit channel ID, the 8 bit hit timestamp, the multi-hit flag, the 2 bit hit type and the first 6 bits of the ADC values. The channel ID identifies the channel where the hit occurred. The hit timestamps are the lower 8 bits of the full 14 bit timestamp and marks the time of the hit. The multi-hit flag indicates wether the current hit message was triggered while the previous one was being built. The hit type is used to distinguish between the different trigger types.

The trigger types are:

- `00`: external trigger,
- `01`: self-trigger,
- `10`: neighbor-trigger,
- `11`: self- and neighbor-trigger.

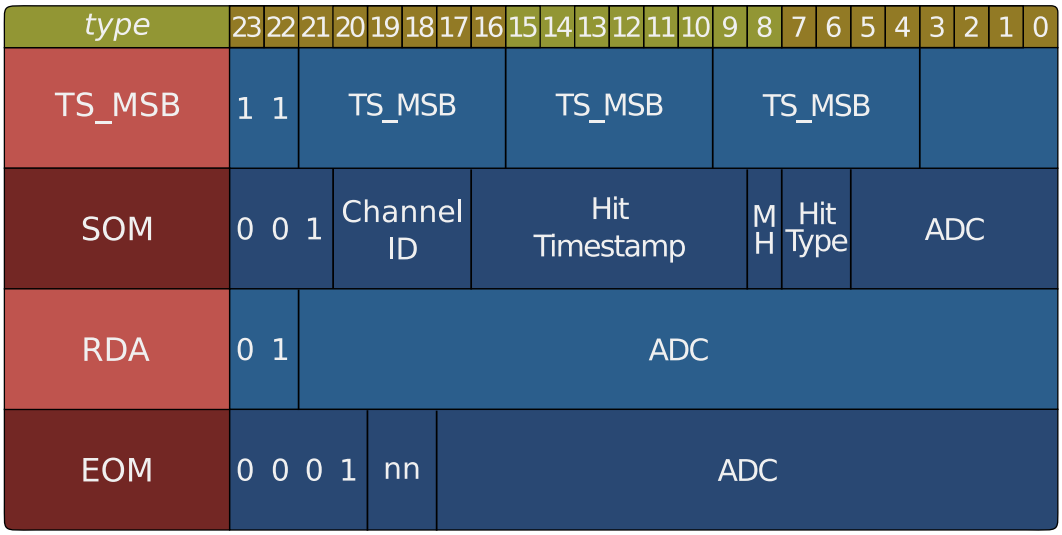

Figure 2.6: Structure of the SPADIC hit frames.

The RDA frames have the prefix `01` and contain 22 bits reserved for ADC values.
The EOM frame prefix is `0001`, followed by the 2 bit samples indicator and 18 bits reserved for ADC values. The samples indicator is modulo 4 of the number of samples and in combination with the total words in a hit message can be used to calculate the number of samples in the hit message (see Tab. 3.1).

The TimeStamp Most Significant Bits (TS_MSB) frame transmit the 6 most significant bits of the full 14 bit timestamp. The TS_MSB frame has the prefix `11`, then 3 times the TS_MSB timestamp for error correction and 4 unused bits. The TS_MSB frame acts as an epoch marker and is sent whenever the hit timestamp overflows.

Figure 2.7 displays the error frames for different exception types. The exception that is expected the most is the Buffer Overflow Message (BOM) that occurs when the buffer of a channel is full. It contains 14 bits for the number of lost hits due to the buffer overflow and the channel ID where the lost hits occurred.

The MeSsage Build (MSB) error is raised during an error in the message building logic. It contains 13 unused bits and the channel ID.

If any of the FIFO output buffers in the SPADIC become full while a message is being build, the BUffer Full (BUF) error is transmitted. It contains the channel ID, 11 unused bits and the 2 bit buffer full status code. The status codes are:

- `01`: channel buffer full,
- `11`: channel buffer full and multi-hit,
- `10`: multi-hit; ordering FIFO full.

The UNUsed Request (UNU) and the MISsing Request (MIS) exceptions signal an error in the channel switch of the SPADIC. While the UNU exception is specific to a channel and contains the channel ID, the MIS exception is a more general error and not specific to a channel.

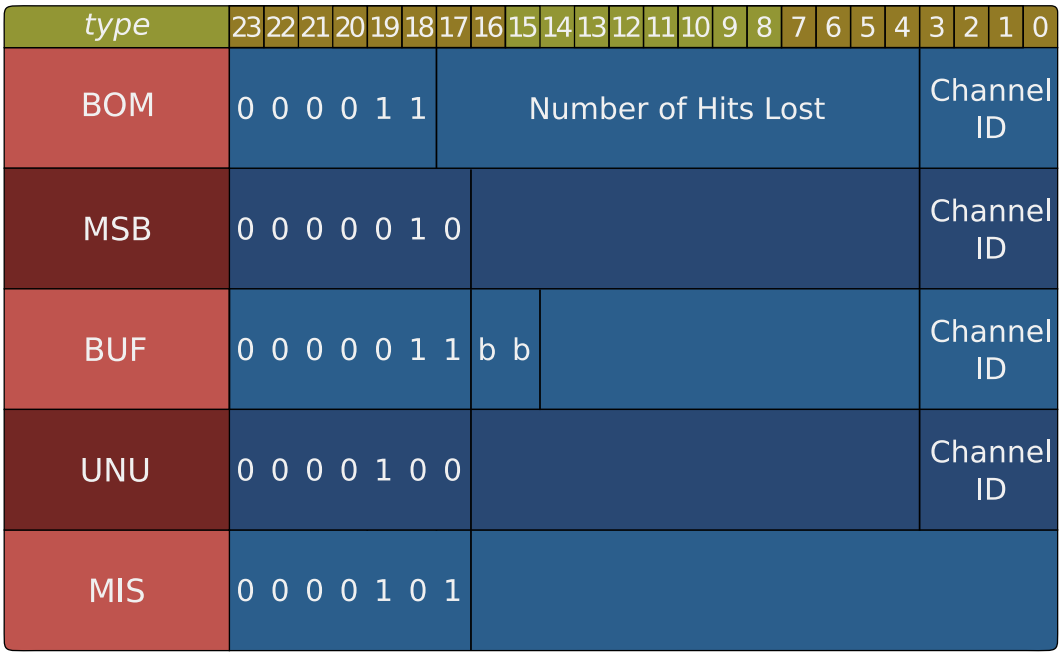

Figure 2.7: Structure of the SPADIC error frames.

2.4 SPADIC Configuration

Digital Registers

The SPADIC is configured through a number of digital registers. With these the user can e.g. reset the SPADIC, enable the data readout or set the thresholds. For example, to enable the data readout the following two frames have to be sent immediately after another:

Type	Chip Address	Sequence Nr.	Request	Payload
Address Frame	0001	0001	01	00 0000 0000 0010
Value Frame	0001	0010	10	00 0000 0000 0001

Sending these two frames would address the SPADIC on the FEB with chip address 1 and enable the data readout which has the register address 2 and write the value 1 to it. Table 2.1 shows a list of a few important configuration registers. For a complete list of the digital register file see [12].

2.4.1 Analog Shift-Register

The analog part of the chip can be configured through a 584 bit deep shift register. This allows, for example, to shift the baseline value up or down. A complete list of the possible configuration with the shift register can be found in [12]. Every time an analog value has to be changed the entire shift register has to be rewritten. The procedure for this is as follows:

- Write the value 0x1242 into the digital register with address 0x4000 to ready the state machine in the SPADIC to write data into the shift register.
- Continuously write the shift register data in 15 bit chunks into the digital register with address 0x4002. The first 15 bit chunk will be shifted to the position 569 to 583 in the shift register and so on.

Unfortunately, a bug in the SPADIC 2.2 prevents the shift register from being read.

Table 2.1: List of important digital registers.

Name	Address	Size	Description
`readoutEnabled`	2	1	Enables the data transfer from the SPADIC to the AFCK.
`compDiffMode`	3	1	Enables the differential trigger logic.
`hitWindowLength`	4	6	Defines the time frame for the message building. Values from 0 to 32.
`selectMask_(0-2)`	5-7	32	The next three registers define the selection mask for the first half of the chip. The selection mask allows to only read out specific ADC samples.
`selectMask2_(0-2)`	8-10	32	Same as above, but for multi-hits.
`useAverageBaseline`	11	1	Enables the calculation of the average baseline between hits. The calculated value is then transmitted in the first ADC sample. The precision of the calculation is 11 bits. During transmission the two most significant bits are discarded (These should always be reconstructible). Multi-hits do not carry the averaged baseline sample.
`neighborSelectMatrixA_(0-38)`	19 - 57	576	Registers to configure the forced neighbor read out for the first chip half; mirrored for the second chip half.
`triggerHoldoff`	116	6	This register allows the configuration of a retrigger protection. The value defines the time-frame (in timebins) in which no retrigger can occur.
`threshold1_(0-19)`	117 - 136	288	Set the first threshold on a per channel basis.
`threshold2_(0-19)`	137 - 156	288	Set the second threshold on a per channel basis.
`CMD_trigger`	242	2	Externally trigger the SPADIC. `01` for a forced trigger, `10` for a in-chip injected pulse, and `11` to simultaneously inject a pulse and trigger.

Chapter 3

The Data Processing Board Firmware

3.1 Common Firmware Blocks

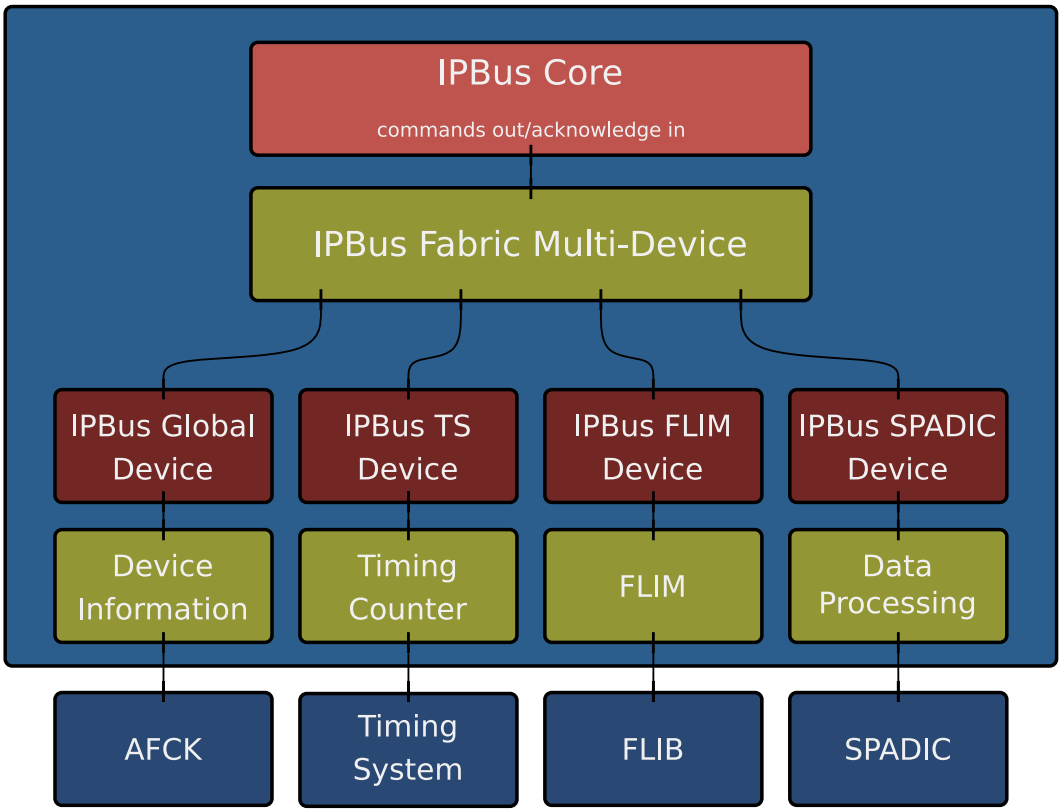

Figure 3.1: IPbus command distribution network. The IPbus command are sent to the IPbus core. The IPbus Fabric Multi-Device then distributes the commands to the corresponding slaves.

The CBM experiment consists of several subsystems with different requirements in the readout chain. However, they all are supposed to finally share the CBM main clock, and therefore the same Timing System (TS), and the FLIB readout. Further on, they use the IPbus protocol to communicate with the AFCK FPGA board in the prototype setup, which will later be replaced by a PCIe communication bus. To minimize the amount of redundant work, the DPB firmware uses a modular block design which shares the basic building blocks between the different subsystems.

The IPbus communication is initialized in the IPbus core. The IPbus core generates a request to assign an IP address to the AFCK MAC address. Once the network connection is established, the IPbus commands are distributed to the IPbus slaves in the IPbus Fabric Multi-Device block. Figure 3.1 shows the IPbus network inside the FPGA. The blocks which are shared between the DPBs of the different subsystems are the global device, the timing system device and the FLIM device.

The IPbus global device allows the configuration and monitoring of the device information, e. g. setting the device ID for identification of the specific AFCK in the readout chain. Furthermore, the global device interfaces the AFCK components through the I2C protocol. This enables the configuration of the Si570 oscillator frequency via IPbus, to provide the system clock for the AFCK, or initialize a reset of the AFCK.

The IPbus TS device interfaces the CBM TS which delivers a common 40 MHz clock for all AFCKs in each subsystem. Additionally, the IPbus TS provides the frame counter for time deterministic commands. The 6 byte downlink frame takes sixty 160 MHz clock cycles after the 8b10b encoding to be fully transmitted to the SPADIC, which corresponds to fifteen 40 MHz clock cycles. Therefore, the frame counter has to be increased every fifteen 40 MHz clock cycles. To send time deterministic commands, this frame counter can be read out over IPbus and then instruct the AFCK to send a command at a given frame number a few seconds later, to account for the IPbus latency.

The IPbus FLIM device enables the configuration of the µSlices. The µSlices are then built in the FLIM block and sent to the FLIB readout node.

3.2 IPbus SPADIC Device

Everything which is specific to the SPADIC readout chip is done inside the IPbus SPADIC device. This includes the interfacing to the chip as well as the processing of the data from the SPADIC. Figure 3.2 shows the block diagram of the IPbus SPADIC device. As the SPADIC employs the same communication protocol as the SMX most of the IPbus block could be reused from the STS-DPB firmware, and aside from the data processing, only minor modifications were made wherever necessary.

Inside the IPbus SPADIC core the IPbus commands are distributed to one common IPbus register and to a interface IPbus register for each connected SPADIC, as is illustrated in Fig. 3.3. The common register handles global configurations and status registers as well as two FIFOs for raw data and processed data allowing easier debugging, while the interface registers contain the command register to configure the ASIC and the status registers to monitor the ASIC. The common control and status registers are listed below:

- Control registers:
  - **clk_delay** Sets the clock delay that is applied to the outgoing SPADIC clock.
  - **clk_out_sel** Selects the outgoing clock for which the delay will be applied.
  - **reset_fifo** Resets the FIFO that stores the µSlice data before it is further processed by the FLIM.
  - **rst_dataproc** Synchronously resets the entire data processing chain.
  - **ipbus_fifo_used** Marks whether the µSlice data is processed by the FLIM and sent to the FLIB or the IPbus FIFO for inspecting the processed data.
  - **rawdata_elink_sel** Sets the elink from which the data is stored inside the raw data FIFO.
  - **reset_raw_fifo** Resets the FIFO that stores the raw unprocessed data.

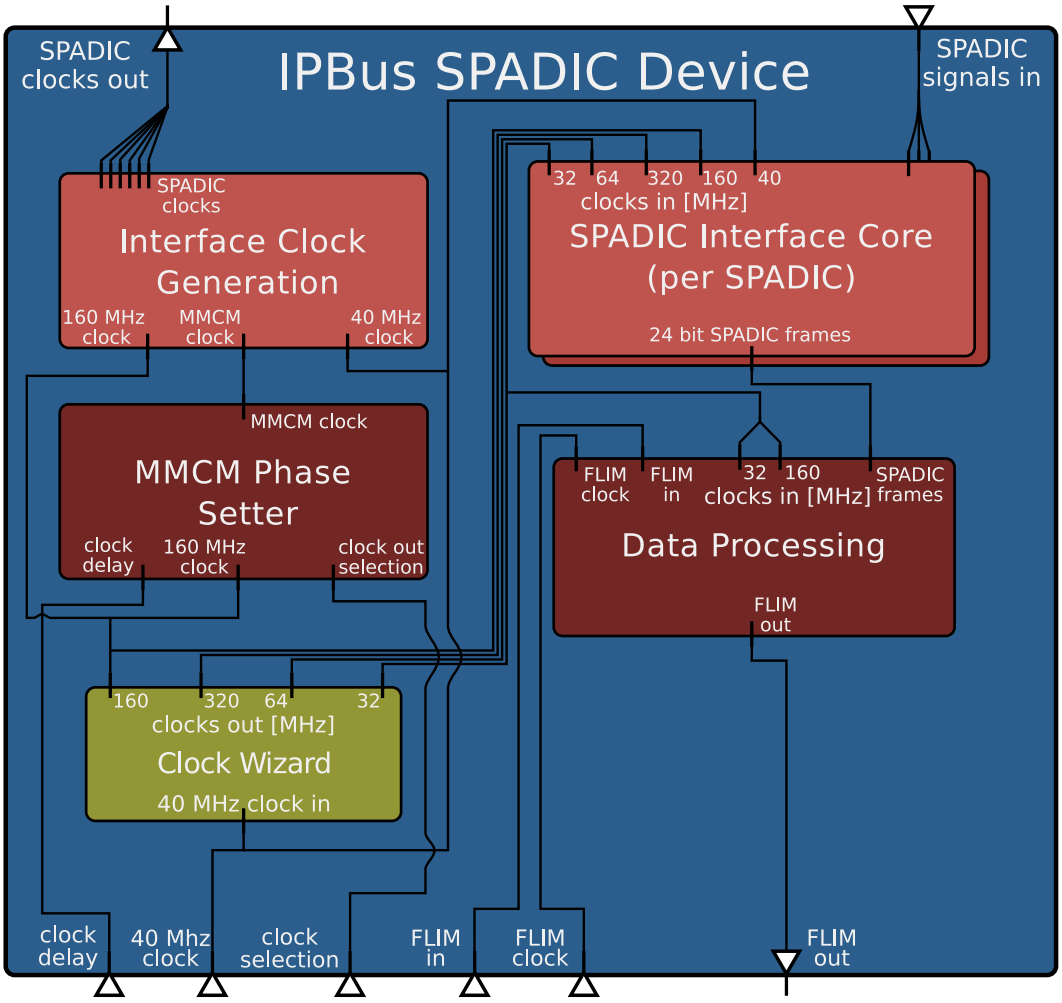

Figure 3.2: Block diagram of the IPbus SPADIC device. Inside the IPbus SPADIC device the necessary clocks for the design are generated and phase locked. This includes the clocks for the connected SPADICs. Additionally, the IPbus SPADIC device contains the interface cores which directly interface the SPADICs. The data processing block then processes the data from the SPADICs.

- Status registers:
  - **iface_pll_locked** Monitors the status of the Phase-Locked-Loop (PLL) for the outgoing SPADIC interface clocks.
  - **clk_delay_ready** Marks if the clock delay was successfully set in the Mixed-Mode Clock Manager (MMCM) phase setter.
  - **tfc_pps_position** Monitors the Pulse-per-second signal from the CBM TS.
  - **tfc_fr_per_counter** Monitors the TS frame number and the TS period.

The clocks which are necessary to drive the design are generated from the 40 MHz main clock in the SPADIC Clock Wizard. The generated clock frequencies are 320 MHz, 160 MHz, 64 MHz and 32 MHz. The SPADIC is driven by a 160 MHz clock and the data is transmitted at Double-Data-Rate (DDR). Therefore, a 320 MHz clock is required to process the incoming SPADIC signals. As the chip signals are transmitted 8b10b encoded, one 8 bit frame is completed every 10th 320 MHz clock cycle, which leads to a 32 MHz clock to process the

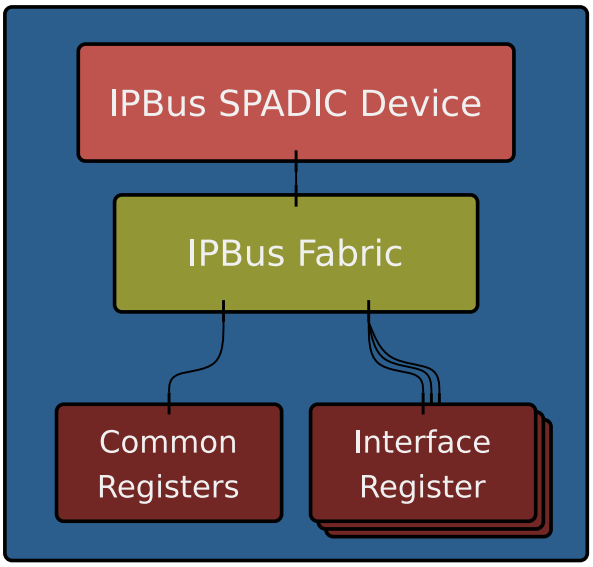

Figure 3.3: IPbus command distribution inside the IPbus SPADIC device, to one common register and to as many interface registers as connected SPADICs.

aggregated SPADIC message frames. As each SPADIC is connected by 2 e-links a 64 MHz clock is used to aggregate the incoming links together for further processing. The SPADIC interface clock generator in combination with the MMCM phase setter provide the connected SPADIC interfaces with the phase adjusted and phase locked clock. The phases are adjusted according to the delays calculated during the initial link synchronization.

3.2.1 SPADIC Interface Core

The data signals from and to the SPADICs are handled in the SPADIC interface core. The control and status registers generated for each interface also directly connect to the interface core as can be seen in the block diagram in Fig. 3.4.
The following control and registers are generated for each connected interface:

- Control registers:
  - **spadic_rst_n** Resets the SPADIC interface.
  - **din_delays** Set the delays calculated during link initialization for the data lines.
  - **seq_det_ins** Tells the special sequence decoders to clear the detected sequence to help eliminate metastability.
  - **enc_mode** Specifies the encoding mode for the encoders. The modes are **SOS**, **K28_1**, **EOS** and **FRAME**.
  - **dwnl_cmds[0:4]** Write a SPADIC command to one of the 5 command slots.
  - **dwnl_td_cmds[0:1]** Write a time-deterministic command to one of the 2 time-deterministic command slots.
  - **dwnl_td_fnums[0:1]** Specify the frame number at which the time-deterministic command will be executed.

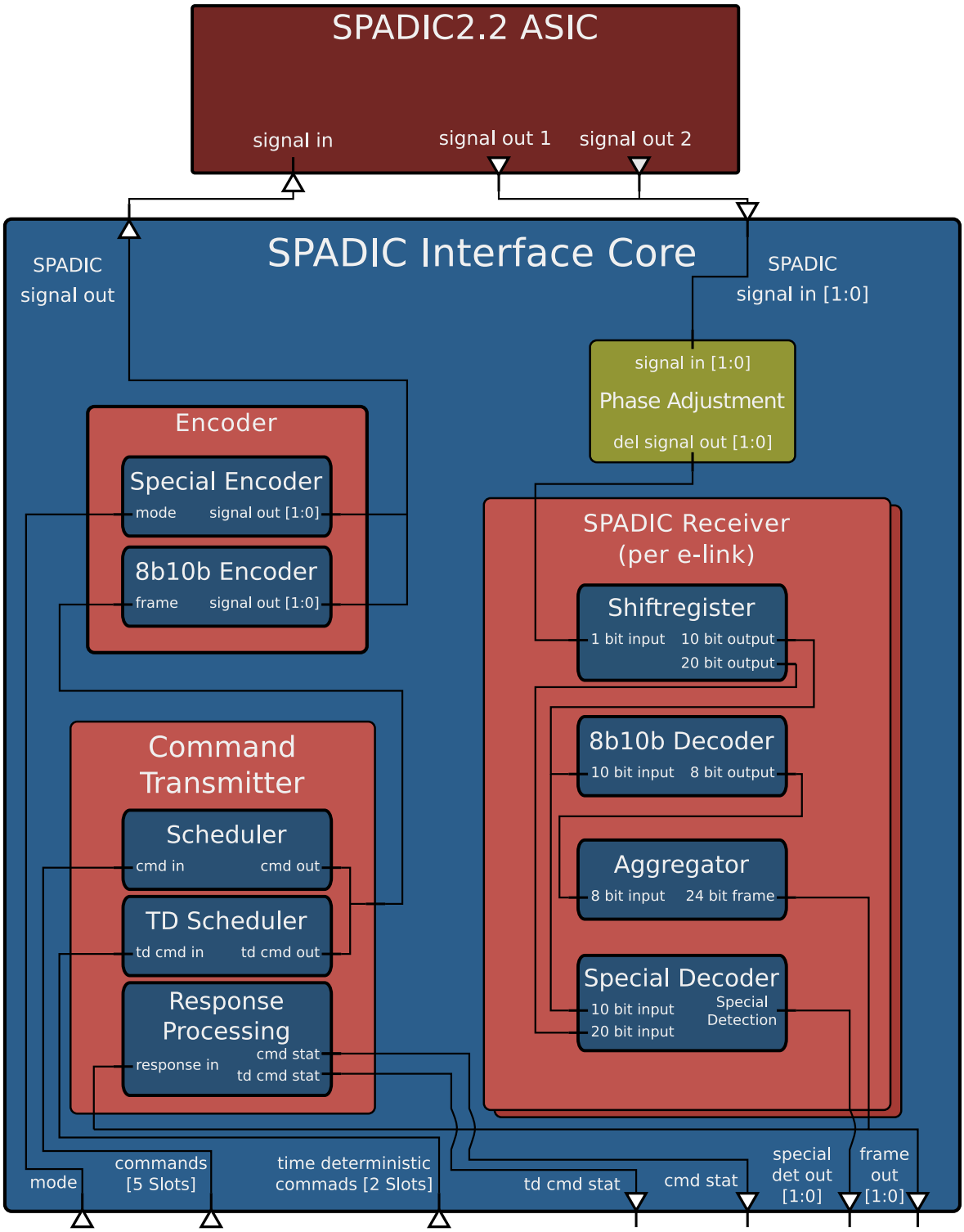

Figure 3.4: SPADIC Interface Core block diagram.

- Status registers:
  - **din_delays_locked** Reports whether the delay of the data lines is locked in.
  - **seq_det_out** Monitors the current detected special sequence.

- **cmd_stats[0:4]** Registers the responses from the SPADIC to previously sent commands.
- **cmd_td_stats[0:1]** Registers the responses from the SPADIC to previously sent time-deterministic commands.

To complete the link synchronization sequence according to Section 2.2 the encoding mode has to be set with respect to the sequence order. First, the SOS mode is being used to start transmitting the SOS sequence, followed by the K28_1 mode and finally the EOS mode. During this sequence the SPADIC responds accordingly. The sequences are detected by the Special Decoder inside the SPADIC receiver. After the link initialization sequence the encoding mode is set to FRAME mode for regular frame transmission. To decode the data, the incoming signals are aggregated in a shift register with a 10 bit vector output and a 20 bit vector output. The 20 bit output is connected to the Special Decoder while the 10 bit output is connected to the 8b10b decoder. The output of the Special Decoder is the detected sequence which can then be readout from the **seq_det_out** status register.

The outgoing command frames come in through the **dwnl_cmds**/**dwnl_td_cmds** command registers. The commands are then scheduled in the command transmitter. Once they are ready to be executed they are 8b10b encoded and sent to the SPADIC. On the incoming side the 8b10b decoded data is aggregated to the full 24 bit message frames in a shift register. The 24 bit frames are marked as valid as soon as three 8 bit frames have been collected. If the message frame is a response to a command, it is processed in the command transmitter and the output is written to the **cmd_stat**/**td_cmd_stat** status register. The TS_MSB frames and hit frames are then further processed in the data processing block.

3.3 Data Processing

The FLIM accepts 64 bit frames, therefore, the 24 bit frame format is not optimal and has to be processed to increase the efficiency of the FLIM link usage and, in the current development stage, reduce the amount of storage needed to save the detector data. To achieve this, the raw data is processed in the data processing block. The block diagram for the data processing is illustrated in Fig. 3.5.

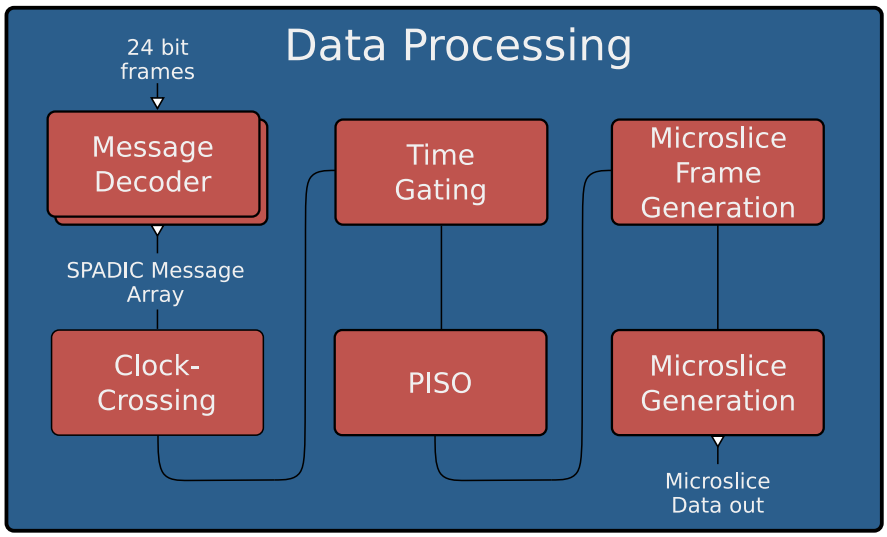

Figure 3.5: Block diagramm for the SPADIC data processing.

3.3.1 SPADIC Message Decoder

First the raw SPADIC messages are extracted in the SPADIC message decoder. For each connected e-link a message decoder entity is generated. A schematic flowchart for the decoding process is demonstrated in Fig. 3.6.

The frames are identified by their respective prefix. This identification is done regardless of the validity of the frame. This means as soon as the first 8 bit of the frame are ready the frame type is determined. This allows for the message decoder to know if there are any messages in the message building buffer without waiting for the frame to be complete, which is useful for the error handling in the time gating block.

Once the frame is marked as valid, the incoming words are decoded into their respective record types to ease the decoding process. With the frame records ready, the SPADIC message building can begin. The frames are aggregated in a SPADIC message record type, see Code Block 3.1. If it is an SOM word, the SPADIC message is filled with the meta data and the first 6 ADC bits of the first sample are stored in the most significant bits of the 288 bit ADC vector. If it is an RDA word the ADC vector is filled with the ADC values and the number of RDA words is counted to calculate the number of samples later. If it is an EOM word, the remaining ADC values are stored and the number of samples is calculated through a Look-Up-Table (LUT) of the number of RDA words and the 2 bit samples indicator. Table 3.1 shows the LUT for the number of samples calculation.

The TS_MSB words and the error messages are forwarded in the `misc_message` and the special flag is set.

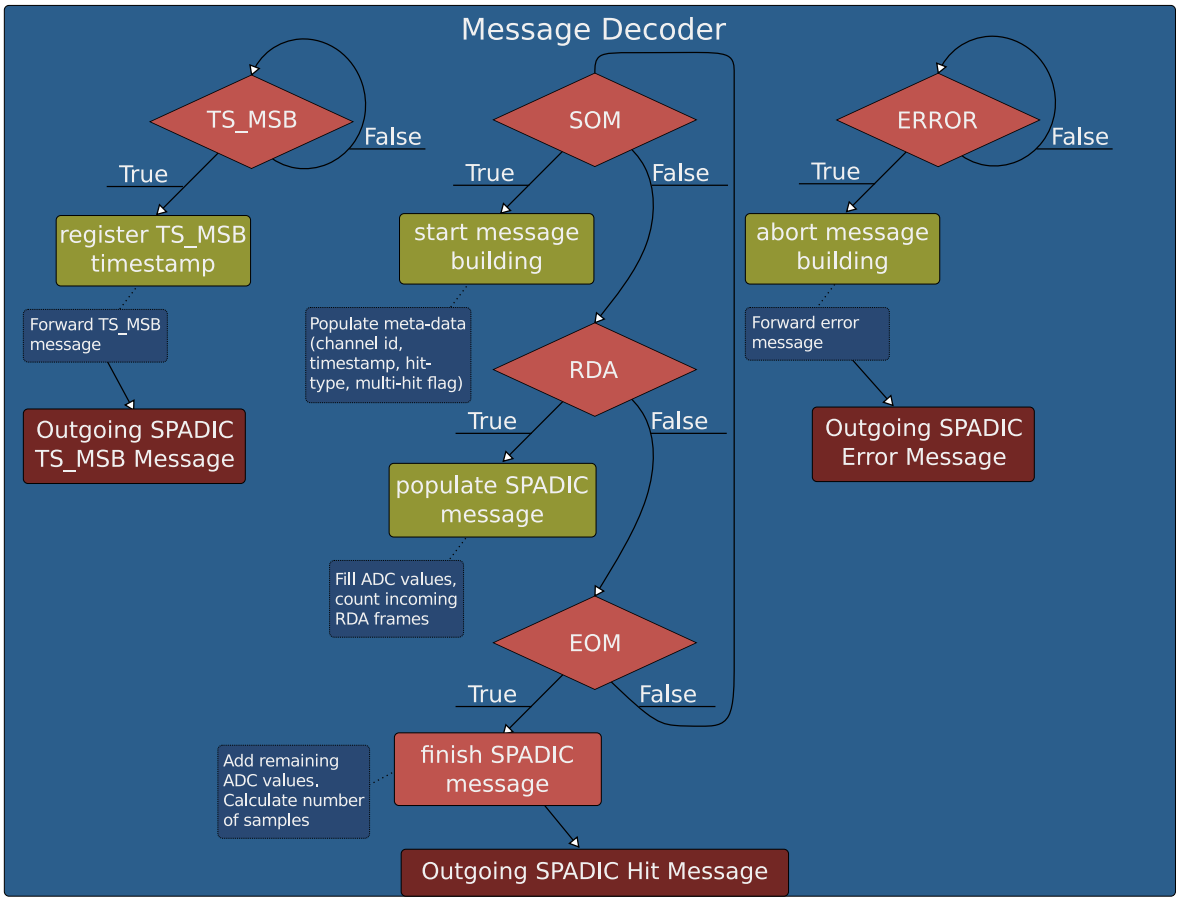

Figure 3.6: Flowchart for the SPADIC message decoder.

The Message Decoder runs in the 32 MHz clock domain, as this is the clock domain in which the 24 bit frames are received. However, the rest of the data processing design runs in the 160 MHz clock domain. Therefore, the decoded SPADIC messages have to cross the clock domain. This can introduce meta-stability issues, which can be eliminated by double flopping the data. To double-flop the data, the original SPADIC message is written into a register (first flip-flop), which in turn is written into another register (second flip-flop), as can be seen in the Code Block 3.2. The Code Block 3.2 ensures that the signal is stable after crossing the clock domain and is only valid for one 160 MHz clock cycle .After that, the decoded SPADIC messages are in the 160 MHz clock domain and are ready to be processed by the rest of the design.

```
type spadic_message_rt is record
    timestamp      : unsigned(13 downto 0);
    elink_id       : std_logic_vector(5 downto 0);
    channel_id     : std_logic_vector(3 downto 0);
    hit_type       : std_logic_vector(1 downto 0);
    multihit       : std_logic_vector(0 downto 0);
    samples        : std_logic_vector(4 downto 0);
    adc            : std_logic_vector(287 downto 0);
    special        : std_logic;
    misc_message   : std_logic_vector(20 downto 0);
    wr             : std_logic;
end record spadic_message_rt;
```

Code Block 3.1: Record type for the decoded SPADIC message. The 14 bit timestamp is composed of the 6 bit TS_MSB timestamp and the 8 bit hit timestamp. The special flag marks if the SPADIC message is a TS_MSB or error message. The misc_message then contains either the TS_MSB timestamp or the SPADIC error message. The wr flag indicates whether the SPADIC message is valid.

```
per_elink_cc_32_160: for i in 0 to elink_channels_g-1 generate
begin
  cc_32_160 : process(clk_160_i)
  begin
    if rising_edge(clk_160_i) then
      if rst_c160_i = '1' then
        spadic_messages_c160_i(i) <= spadic_message_zero_c;
        spadic_messages_r1(i) <= spadic_message_zero_c;
        spadic_messages_r2(i) <= spadic_message_zero_c;
        spadic_messages_r3(i) <= spadic_message_zero_c;
      else
        -- double-flob the data to eliminate metastability.
        spadic_messages_r1(i) <= spadic_messages_c32_i(i);
        spadic_messages_r2(i) <= spadic_messages_r1(i);
        spadic_messages_r3(i) <= spadic_messages_r2(i);
      end if;
      -- make sure, that the message is only valid for one clock
      -- cycle.
      if (spadic_messages_r2(i).wr = '1' and
          spadic_messages_r3(i).wr = '0')
      then
        spadic_messages_c160_i(i) <= spadic_messages_r2(i);
      else
        spadic_messages_c160_i(i) <= spadic_message_zero_c;
      end if;
    end if;
  end process;
end generate per_elink_cc_32_160;
```

Code Block 3.2: Clock crossing code. It eliminates meta-stability issues and ensures that the SPADIC message is only valid for one 160 MHz clock cycle. The code block is generated for each connected e-link.

Table 3.1: LUT of the Samples Indicator and the number of RDA words for the number of samples in the SPADIC message.

RDA Words	Samples Indicator	Nr. of Samples
0	1	1
0	2	2
1	3	3
1	0	4
1	1	5
2	2	6
2	3	7
3	0	8
3	1	9
3	2	10
4	3	11
4	0	12
5	1	13
5	2	14
6	3	15
6	0	16
6	1	17
7	2	18
7	3	19
8	0	20
8	1	21
8	2	22
9	3	23
9	0	24
10	1	25
10	2	26
10	3	27
11	0	28
11	1	29
12	2	30
12	3	31
12	0	32

3.3.2 Time Gating

One intrinsic design goal of the µSlice readout concept is that the µSlices are self-contained in time. The µSlices have a specific length in time and each SPADIC hit message which falls into that time frame must be included in the µSlice. The µSlice length in the current implementation is 128 µs.

The required time consistency is achieved by time-gating the incoming SPADIC hit messages. The time-gating procedure makes use of the fact that the raw SPADIC messages are sorted within time in each e-link, but not between e-links. Figure 3.7 illustrates the process for two e-links. The data flow is gated through FIFOs for each e-link. The FIFOs are continuously read out until the TS_MSB word arrives that marks the end of the µSlice time frame, which is the TS_MSB word with the timestamp `xxx000`. The three zeros in the least significant bits cycle every 128 µs. The `x` signifies a do-not-care. When this TS_MSB word for a specific e-link arrives, the readout from the corresponding FIFO is stopped.

After the TS_MSB word of each e-link has been received, the last TS_MSB word is forwarded and the FIFOs resume continuous transmission. If however, one e-link is not sending the TS_MSB word correctly, a check is made whether there are still messages in the message building buffer. If not, the TS_MSB word can be assumed to have been missed and the TS_MSB word is marked as received for that e-link.

To minimize the footprint of the gating FIFOs they are implemented as a shift register with a depth of 32. This depth is more than enough to store the data until all TS_MSB words have arrived. To ease the serialization of the e-link data in the next step the FIFOs

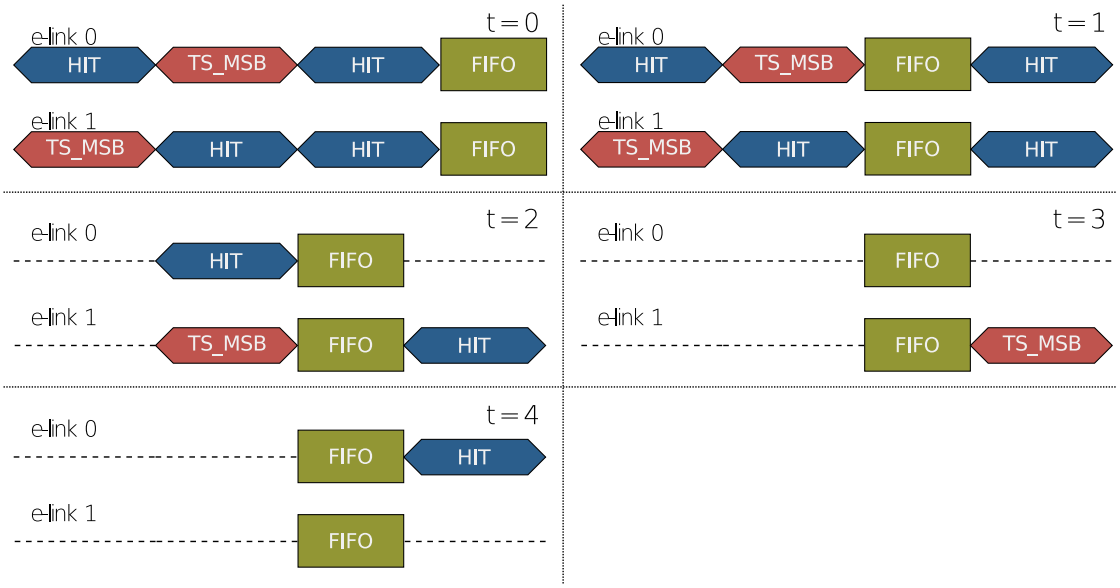

Figure 3.7: Schematic view for the time-gating procedure with 2 connected e-links. In the top left of the picture it shows the incoming data at $t = 0$. In the top right at $t = 1$ no TS_MSB word has arrived yet, so the data is passed through. At $t = 2$ e-link 0 detects the TS_MSB word that marks the end of a µSlice and the reading of the FIFO is stopped. At $t = 3$ e-link 1 detects the TS_MSB word that marks the end of a µSlice. As e-link 1 is the last e-link to receive the TS_MSB word it forwards the TS_MSB word. The reading of the e-link 0 FIFO is still stopped and the incoming hit message is buffered. At $t = 4$ the continuous readout can be resumed and the buffered hit message is read out.

are read out every n-th clock cycle, with n being the number of connected e-links. The time between two valid SPADIC messages on the same e-link is

$$n_{cc} + n_w \times 3 \times 32\,\text{MHz clock cycles},$$

with n_{cc} being the number of 32 MHz clock cycles since the last valid SPADIC message and n_w being the number of words in the raw SPADIC messages. The factor 3 is due to the 8b10b decoding of the 24 bit frames. Therefore, the minimum time between two SPADIC hit messages is $6 \times 32\,\text{MHz}$ clock cycles, as two words are the minimum for a raw SPADIC hit messages. Six 32 MHz clock cycles correspond to thirty 160 MHz clock cycles, therefore the FIFOs will not fill up if less than 30 e-links are connected. If more than 30 e-links are connected, the SPADIC retrigger protection (see Section 2.4) can be set to make sure the FIFOs will not fill up during very high data rates.

Serialization

After the time-gating process the e-link data is ready to be serialized for the µSlice generation. This is done in a standard Parallel-In-Serial-Out (PISO) shift register. The PISO module consists of $n + 1$ registers, with n being the number of connected e-links. If valid e-link data reaches the PISO module, the shift register is filled up to the n-th entry. The data is then continuously shifted up by one. The output port of the shift register is the $1 + n$-th entry of the shift register. Figure 3.8 shows the concept of the PISO module for 6 connected e-links.

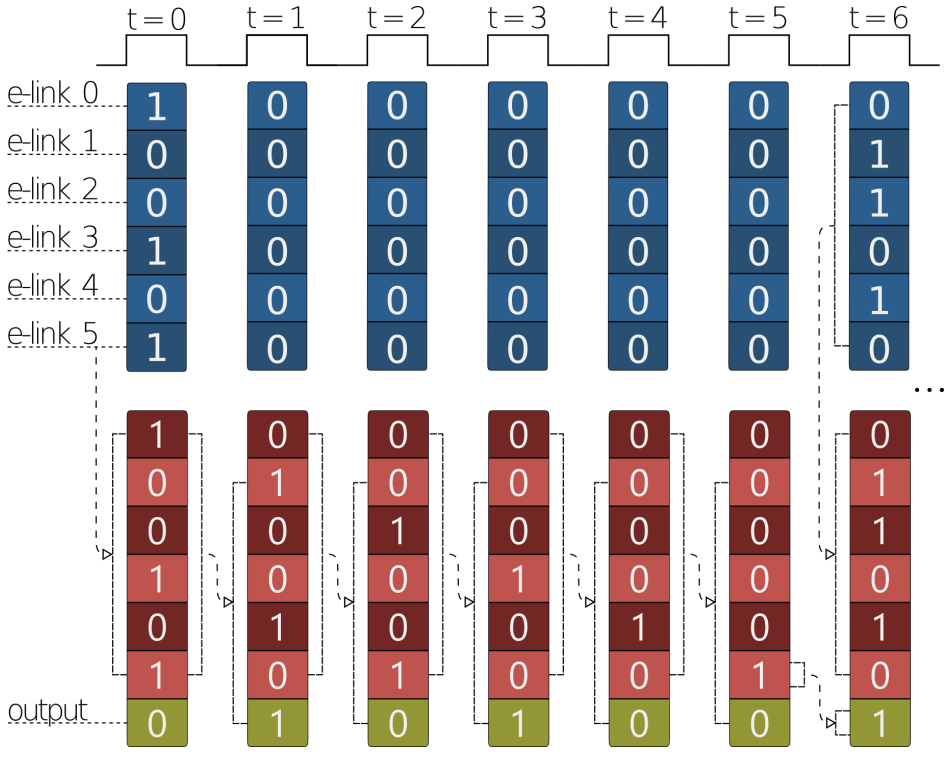

Figure 3.8: Concept of the PISO shift register for 6 connected e-links. An entry with a 1 marks a valid SPADIC message. In the first clock cylce the e-link data is pushed into the shift register. As the time gating FIFOs are only read out every 6th clock cycle, the next 5 clock cycles have no valid incoming SPADIC messages. During that time the PISO module continuously shifts the data down by one to the output port of the shift register. In the 6th clock cycle the PISO module has serialized the data and is ready for new data.

3.3.3 The µSlice Message Format

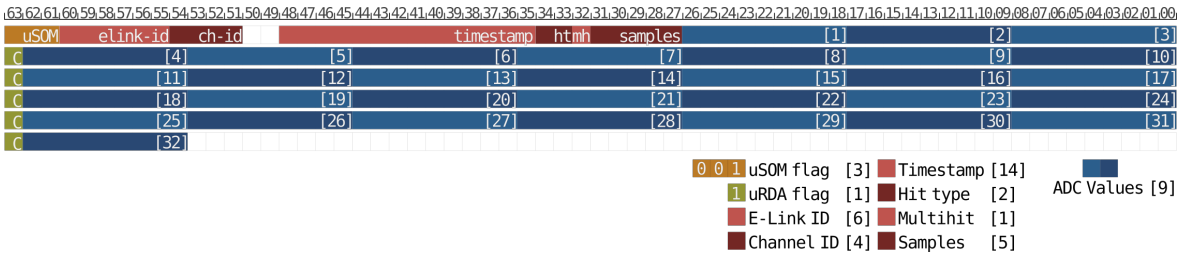

Figure 3.9: Data format for the 64 bit µSlice frames.

To achieve efficient 64 bit frames for the FLIM, the data has to be reformatted and tightly packed into the 64 bit payload. Figure 3.9 shows the chosen data format. There are two frame types in the µSlice-Data format, the µ-Start-Of-Message (µSOM) frame and the µ-Raw-DAta (µRDA) frame. This new data format has been worked out in the course of this thesis.

The µSOM word starts with the prefix `001` in the most significant bits. The prefix is followed by the 6 bit e-link ID and the 4 bit channel ID, which uniquely identify the TRD pad being read out in connection with the AFCK equipment ID. After that two unused bits follow. Then the hit specific meta-data is accumulated:

- The full 14 bit timestamp: 6 bit TS_MSB timestamp plus 8 bit hit timestamp,
- the 2 bit hit-type,
- the 1 bit multi-hit flag,
- 5 bit for the number of samples in the message.
- The rest of the frame is then filled with three 9 bit ADC samples.

The µRDA frame begins with a `1` as the prefix and is then continuously filled with the ADC values. This leads to six 64 bit frames for a full 32 ADC samples hit message. This compares to 14 frames needed if no processing would be done, and amounts to an efficiency increase of a factor of 2.3.

For non-data word the µ-Miscellaneous-Message (µMISC) message is created. The format for a miscellaneous SPADIC message, e. g. an error message or a forwarded TS_MSB message, is shown in Fig. 3.10. The µMISC starts with the `0001` µMISC message prefix, followed by the e-link ID. The next valid bit is the TS_MSB flag in bit 20, which marks if the miscellaneous message in the next 20 bits is a TS_MSB timestamp or an error message. If it is a forwarded TS_MSB word the TS_MSB timestamp is stored in the 6 least significant bits of the miscellaneous message. If it is an forwarded error, the error message is stored in the full 20 bits of the miscellaneous message.

These formats have to be generated from the serialized SPADIC messages. The frames are consecutively stored in a $5 \times 64 + 10 = 330$ bit long µSlice-Message vector, starting with the first frame in the 64 most significant bits and so on. The 330 bit µSlice vector will then be stored in a FIFO. From this FIFO the µSlice vector is read out, and the 64 bit µSlice frames are generated in a serialized data stream. If the µSlice vector starts with the µMISC message prefix just one µSlice frame is generated and the next message can be read out in the next clock cycle. However, if the µSlice vector is filled with a hit message, the number of samples is extracted from the meta data, and the necessary frames n_f for a full message are looked up. The readout from the µSlice vector FIFO is stopped for n_f clock cycles, which is the time needed to serialize the µSlice message. The serialized µSlice frames are then again stored in a FIFO until they are ready to be read out by the µSlice generation block. The µSlice frame

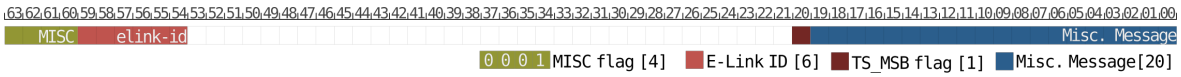

Figure 3.10: Miscellaneous-Message format for the 64 bit µSlice frame

FIFO is written to with the 160 MHz data processing clock and read from with the FLIM clock to cross the clock domains.

3.3.4 µSlice Generation

The µSlice generation block is based on the STS-DPB firmware design. It directly interfaces the IPbus FLIM device for the µSlice configuration, e.g. the length of a µSlice. From this configuration the µSlice frames are built in accordance to the FLES requirements, as described Section 1.3.

At the initialization stage or after a reset the state machine responsible for the µSlice generation is instructed to start a new µSlice. Every µSlice is started with a µEPOCH frame, which consists of the µEPOCH flag `01`, followed by the 30 bit µTS_MSB. The µTS_MSB is generated from a 53 bit epoch counter which increments with every new µSlice. Finally, the 32 least significant bits are the µSlice time in ns, that can be used to crosscheck with the µSlice time in the µSlice descriptor. The µEPOCH frame is illustrated in Fig. 3.11

After that the µSlice is marked as started and the µSlice data is read from the µSlice frame FIFO. The time of incoming µSOM words and forwarded TS_MSB words is continuously extracted and stored in a 64 bit long timestamp which consist of the µEPOCH counter in the most significant bits and the 11 bit SPADIC timestamp in the least significant bits. This timestamp is compared with the current µSlice range. If it is not in the current time frame, the state machine is notified that the current µSlice has to be ended. This is done with an empty frame, which has the `tlast` flag set. Additionally, the µSlice index is incremented and the µSlice time frame is updated. Then the state machine is instructed to start a new µSlice with a new µEPOCH frame. However, as there is a delay between the start of a µSlice and the time the state machine is ready for a data word, the incoming data is buffered in an array of registers to avoid creating a latch, which could create timing problems in the FPGA. But, when the state machine is ready for data, the data is continuously read from the µSlice frame FIFO and the SPADIC hit and error messages are buffered as valid data in the µSlice FIFO. The µSlice FIFO is read out every time it has valid data and the FLIM is ready for data. The FLIM then constructs the µSlice and transmits it to the connected FLIB node.

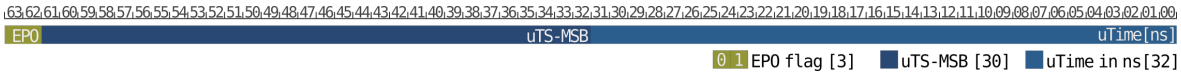

Figure 3.11: The format of the µEPOCH frame.

Chapter 4

The IPbus Control Software

4.1 The Software Structure

Along with the firmware, significant work on the control software for the SPADIC was done. Due to the bug, which prevents the analog shift register from being read, a lot of the previous software could not be used without a significant rework. However, as the control software was overly complicated, much of the old software was discarded and a more maintainable structure was implemented.

The software is written in Python.

The folder structure is as follows:

```
/ipbus_sw/
  -/cfg/
    - contain the IPbus address tables of the different devices
  -/python_lib/
    - Global control libraries that are shared between the subsystems
      - /global_dev_control.py
      - /ipbus_i2c_control.py
      - /flim_dev_control.py
    - Control libraries specific to the SPADIC
      - /spadic_control.py
      - /spadic_calibration.py
  -/run/spadicDPB/
    - Contains the SPADIC command line interface and some
       debugging scripts.
      -spadic_cli.py
    -/cfg/
      - Contains config files for the analog and digital part of the
         SPADIC
    -/functions/
      - Contains the functions for the command line interface
```

As in the firmware implementation, the global control libraries to access the configuration of the AFCK and the FLIM can be reused. Additionally, many functions in the SPADIC control library are based on the STS-XYTER control functions. The SPADIC control library provides the functions to reset the SPADIC, initialize the link, write and read the digital registers and the analog shift-register (write only) and save the current configuration. Moreover, functions to access the IPbus FIFOs containing the raw data and the µSlice data are available. Finally, access to the time deterministic commands is implemented in the control library as well.

The SPADIC calibration library implements the decoding of the IPbus FIFO data. The decoded data is used to determine the baseline and adjust it to the desired value. Furthermore, the IPbus data is suitable to investigate the noise in a beam time environment.

4.2 The SPADIC Software

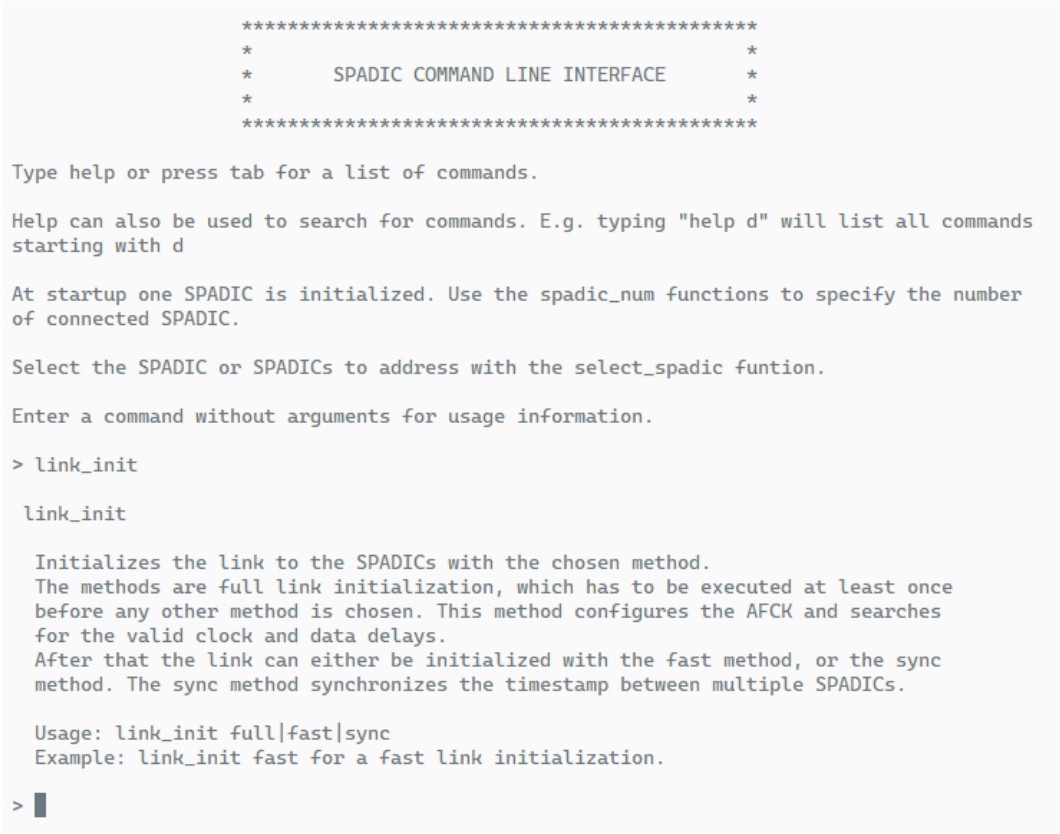

Figure 4.1: Snapshot of the SPADIC Command Line Interface.

An entirely new Command Line Interface (CLI) was developed to ease the initialization and configuration process of the SPADICs. The SPADIC CLI is designed to be usable with minimal instructions from an expert, so that inexperienced users can operate it during a beam time or their own measurements. The CLI features auto-completion, a command history and extensive usage information. Figure 4.1 shows a snapshot of the CLI.

At startup, the CLI assumes one SPADIC is connected to the AFCK. The number of SPADICs can be increased with the `spadic_num [# of connected SPADICs]` command; counting starts from 1. In the next step, the to be addressed SPADICs have to be selected with the `select_spadic [SPADIC position(s)]` command. The arguments for the commands can be `select_spadic -1` to address all connected SPADICs, `select_spadic 0` to only address the SPADIC at position 0 or `select_spadic 1,2` to select the SPADICs at position 1 and 2.

The link to the selected SPADICs can then be initialized via the `link_init [method]` function. If the link is initialized for the first time, the full link initialization procedure, as described in Section 2.2, has to executed: `link_init full`. After that the clock and data

delays are saved in a file and the link can be initialized with the `link_init fast` method. To synchronize the front ends in time the `link_init sync` command is utilized.

Once the link is established the SPADICs are programmed with a default digital and analog configuration and the SPADIC is ready to be configured to the users needs. As a first step the ADC baseline should be calibrated. This is done with the `calibrate_baseline [ADC value]` function. The function works by externally triggering the SPADIC, decoding the data from the raw data FIFO and calculating the mean baseline of each channel and the standard deviation of the means to the desired baseline. The baseline is then adjusted in small steps towards the desired baseline. These two steps are repeated until the standard deviation to the desired baseline is equal or smaller than 1.

The thresholds can be set via the `set_thresholds [threshold1 [threshold2]]`. If only one value is given both threshold will be set to this value.

With the `disable_channels [channels to disable]` command, undesired channels can be turned off. Before the command is used, only a single SPADIC should be selected. The arguments accepts channel numbers separated by commas or ranges separated by a colon. Mixing ranges and single channels, like `disable_channels 0,1,3:10`, is possible as well. Range definitions are inclusive. The disabled channels are not persistent between commands (for a single SPADIC), therefore all channels have to be disabled in a single command. To enable all channels `disable_channels off` can be executed.

Aside from the SPADIC CLI, some useful scripts were developed to investigate the noise level during the DESY beam-time campaign in August 2019. One of the scripts continuously calculates the standard deviation σ of the samples of each channel, which is used as a probe for the general noise level. The result are drawn to the terminal, as can be seen in Fig. 4.2. To calculate the standard deviation the SPADIC is externally triggered 100 times.

Another script synchronously triggers all the connected SPADICs and plots the signal shape side by side to uncover correlations in the noise between the front ends.

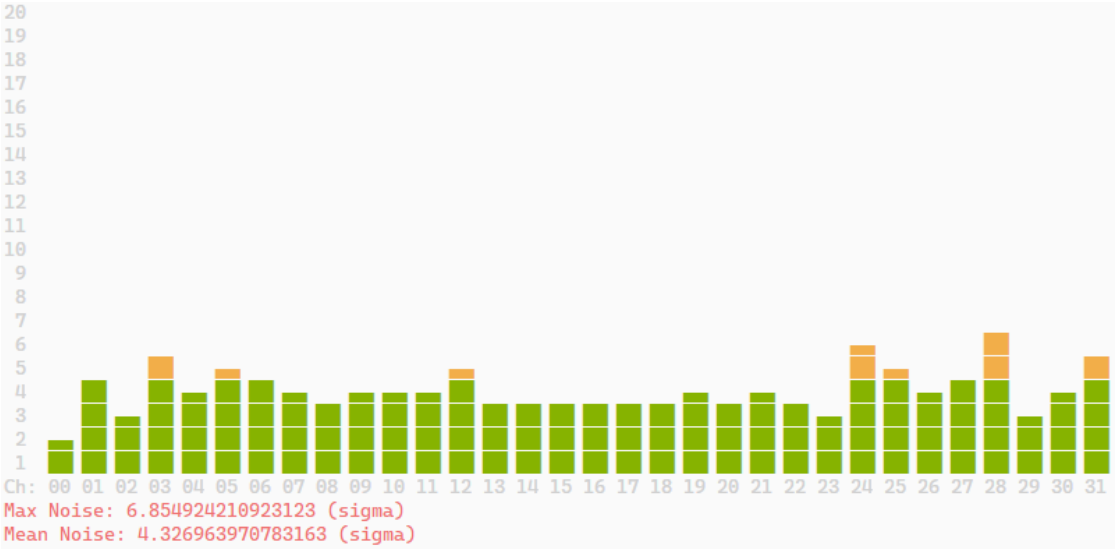

Figure 4.2: Snapshot of the output of the noise script. The output plots the standard deviation to the ADC mean of each channel.

To ensure data transport consistency two small scripts were developed, which continuously fill histogramms depicting the signal shape. One with the data from the raw data FIFO and one with the data from the µSlice FIFO. The histogramms can then be compared during all stages of the data taking process. Figure 4.3 shows a comparison of the histogramms from the raw data FIFO, the µSlice FIFO and the ROOT online monitor with a signal from a ^{55}Fe source. The signals are from the same channel but recorded at slightly different timeframes.

For a continuously updated documentation of the DAQ software see [1].

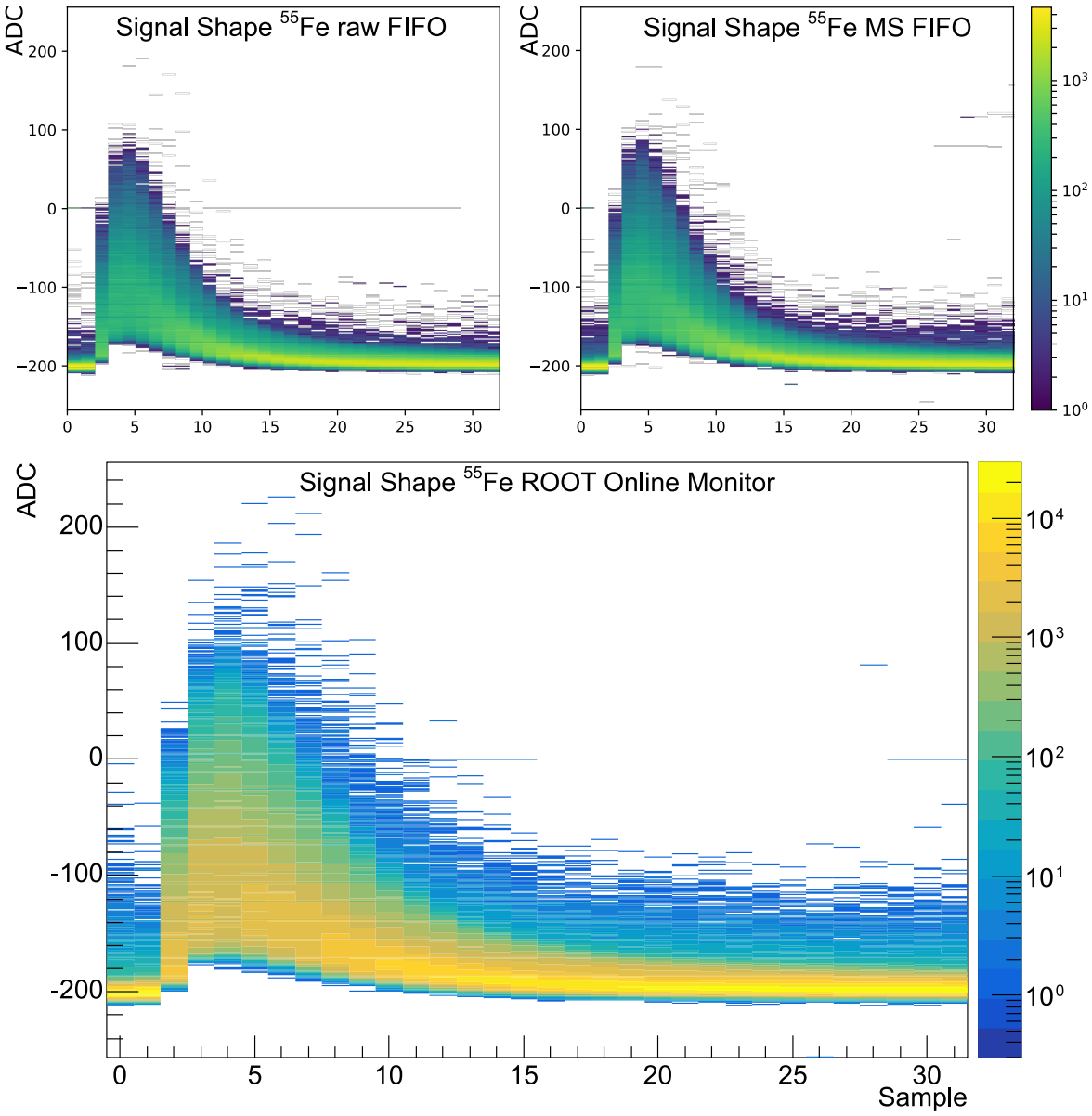

Figure 4.3: Comparison of the data output at all three processing stages of the readout chain. Top left is from the raw data FIFO, top right from the µSlice FIFO and the bottom is from the ROOT online monitor.

Chapter 5

The Measurement Setup

5.1 In the Laboratory

To be able to test the firmware on the hardware, a measurement setup was put together in the laboratory. At the beginning of the development this was just a table setup as can be seen in Fig. 1.7. With this setup it was possible to check the basic functionality of the design, like the link initialization or the IPbus communication. Furthermore, the general debugging of the data processing was done with this setup through the usage of the Xilinx Integrated Logic Analyzer (ILA) and the output of the IPbus data FIFOs. At a later stage a SPADIC 2.2 was connected to a small TRD chamber in preparation for the beam-time at DESY (Deutsches Elektronen SYnchrotron). Figure 5.1 shows the SPADIC connected to the chamber. In this setup it was possible to record the signals of an ^{55}Fe source, which allowed to test the functionality of the data processing as well as the SPADIC in real working conditions. In both setups the link connection was stable for long time periods (several days), and the data processing worked as expected. Due to time constraints no rigorous testing, aside from the basic functionality of the SPADIC, was done. However, the message transmission of the SPADIC showed correct behavior and no indications for missing messages were observed, e.g. during clusterisation of hit messages.

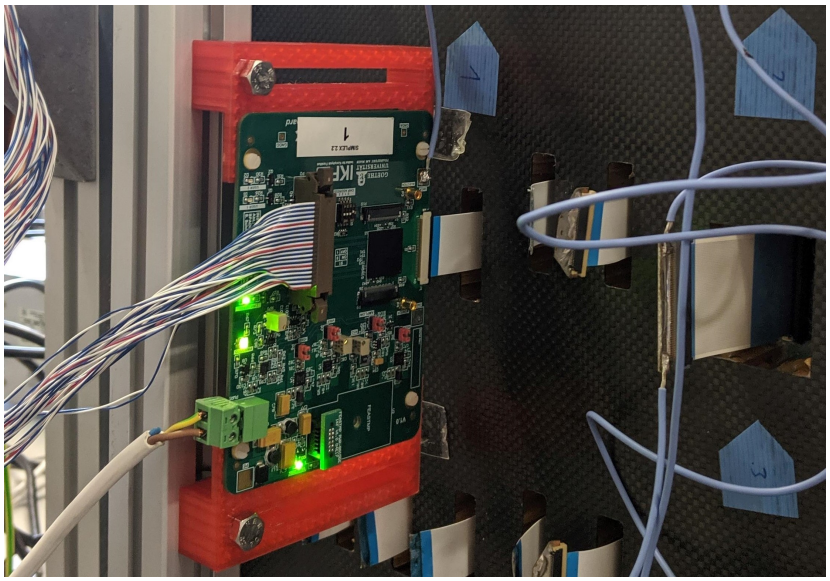

Figure 5.1: A SPADIC 2.2. connected to a small TRD chamber in the laboratory.

5.2 Beam-time at DESY

In August 2019 the CBM-TRD team used the DESY accelerator to measure the radiator performance at different radiator thickness levels. The accelerator provides an electron beam with momenta between $1 - 5\,\mathrm{GeV}/c$. The test setup consisted of two large TRD prototypes. Unfortunately, only two of the five produced SPADIC FEBs were functional. Therefore, only one MWPC equipped with both functional chips was used, as an additional ^{55}Fe source has been used to provide a continuous gain measurement. The two SPADIC 2.2 FEBs were connected to a single AFCK located between the two chambers. The AFCK was connected by optical fiber to the FLIB inside the DAQ-PC located inside the experimental area. The first FEB was used to measure the electron and TR-photon signals. The second FEB measured the signals from the ^{55}Fe source.

The experimental setup is depicted in Fig. 5.2. The operated chamber was closest to the electron beam. On the bottom left, the ^{55}Fe source is mounted close to the entrance window of the chamber. The radiators were installed to the right of the source. Figure 5.3 shows the backside of the operational TRD with two connected SPADICs.

Before the radiator performance measurements were done, the detectors and the readout were conditioned with an 80/20 Ar/CO_2 gas mixture. During this test the detector setup had a noise level in the range of about 30 ADC values around the mean ($1\,\sigma$). A lot of work was done to bring the noise down to acceptable levels. Cables were wrapped around ferrite beads (as can be seen in Fig. 5.3), low voltage cables were shortened and the gas supply tubes, which had contact to the ground, were isolated from the chambers. The left picture of Fig. 5.4 shows the signal shape of the ^{55}Fe source after the noise treatment. However, there is still a noticeable noise signal in the iron pulse shape. Therefore, a signal selection condition has

Figure 5.2: The two TRD chambers in the experimental setup. The chamber in the front was used in the measurements.

been applied to remove the noise from the signal as can be seen at the right side of Fig. 5.4. The selection condition is

$$\frac{\text{sample}_0 + \text{sample}_1 + \text{sample}_3}{3} + 15 < \text{sample}_4 \quad \textbf{and} \quad \text{sample}_1 + 20 < \text{sample}_5.$$

The values 15 and 20 were chosen with respect to the gas gain in Xe/CO_2 at a High-Voltage (HV) of 1800/500. From this point on only pulse shapes matching the selection condition will be shown.

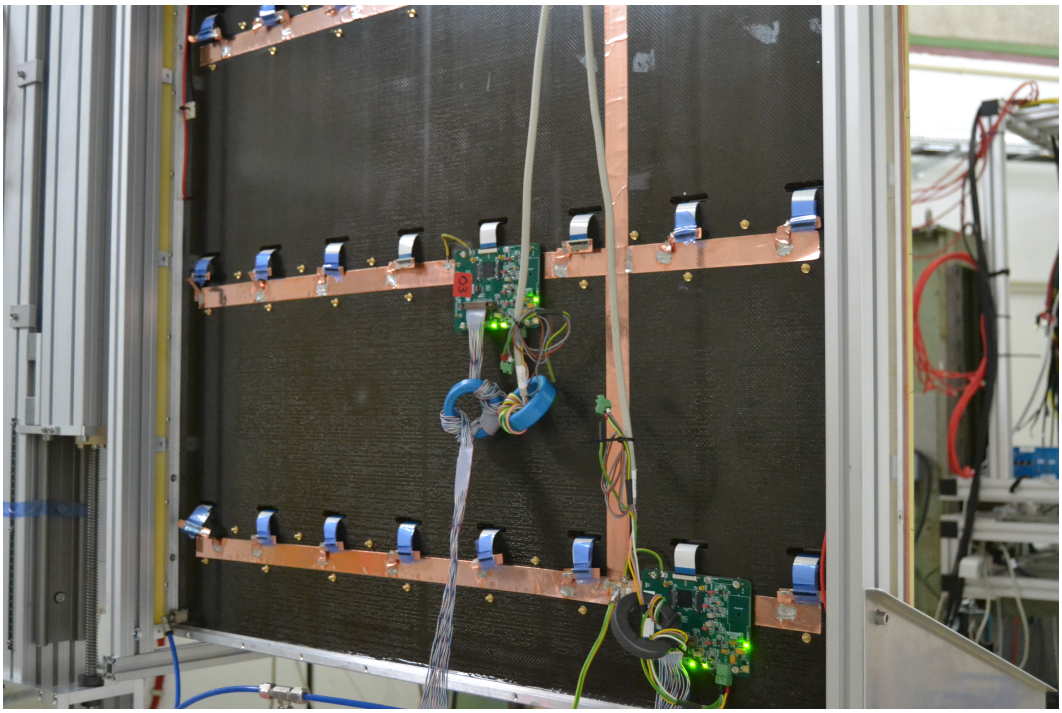

Figure 5.3: Backside of the first TRD chamber, equipped with two SPADIC 2.2 FEBs.

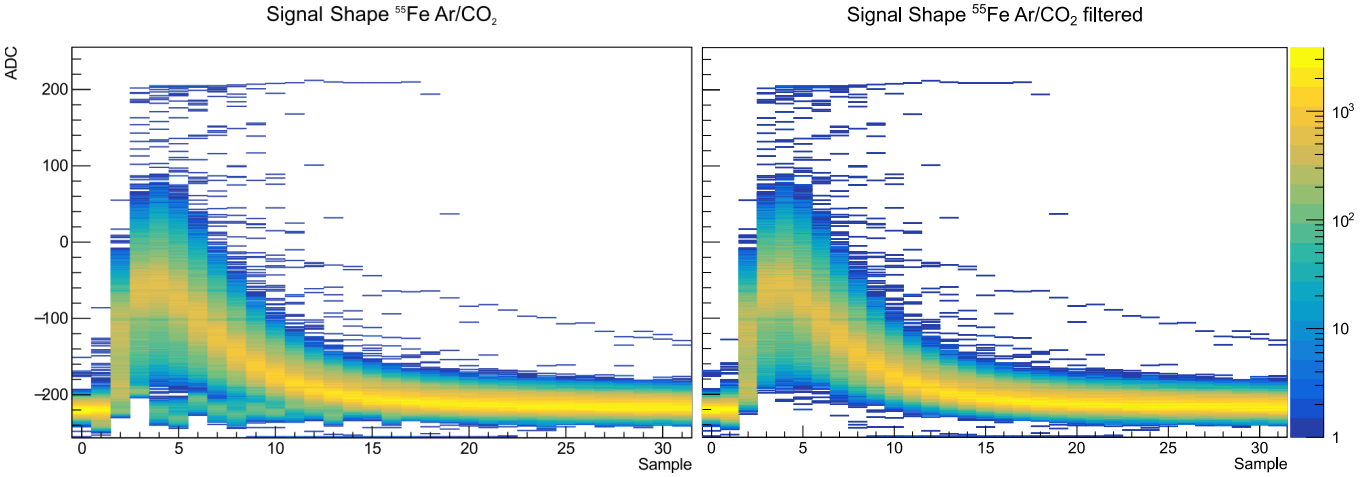

Figure 5.4: Left: Raw ^{55}Fe signal shape in a Ar/CO_2 gas mixture. Right: Filtered ^{55}Fe signal shape in a Ar/CO_2 gas mixture.

Figure 5.5 depicts the distribution of the maximal ADC values (MaxADC). The distribution shows the characteristic peaks expected from a ^{55}Fe source in argon gas, i.e. the argon escape peak at about −130 ADC value and the 5.9 keV-peak at around −50 ADC value. The peak at around −190 ADC value is a noise peak, here no selection on the shape level is applied. Note however, that the MaxADC distribution only corresponds to the ^{55}Fe spectrum as a first approximation. To fully reconstruct the ^{55}Fe spectrum a cluster reconstruction algorithm is required.

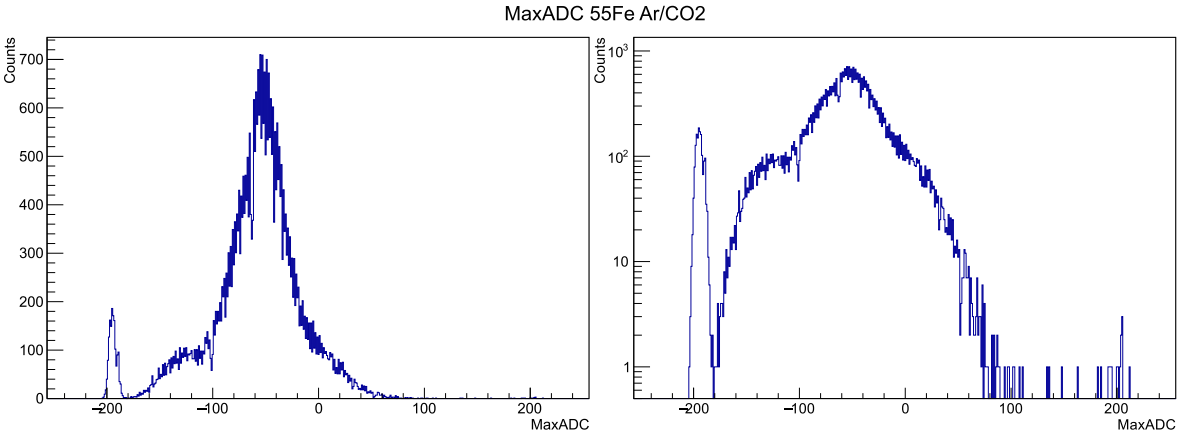

Figure 5.5: MaxADC distribution of the ^{55}Fe signal in Ar/CO_2

Radiator Perfomance Measurements

After the noise level was under control the radiator measurements were performed. The measurements were done at beam energies of 2 and 3 GeV/c and radiator thicknesses of 0, 100, 150, 200, 250 and 300 mm plus the 15 mm radiator at the entrance window. Additionally, a true zero measurement without the 15 mm radiators was performed. For the measurement the MWPC was filled with a 85/15 Xe/CO_2 gas mixture. Figure 5.6 shows the signal shapes of the 3 GeV/c electron beam at radiator thicknesses of 0, 115 and 315 mm. In comparison with the signal shape of the ^{55}Fe source, the range of accepted ADC values is larger, and in particular for the measurements with a radiator the signal shape is distributed more evenly over a larger ADC range, as the produced TR-photons deposit energies in a broader spectrum. This is very well illustrated in Fig. 5.7. With increasing radiator thicknesses the MaxADC distribution gets flatter over the entire ADC range, which is compatible with more TR per electron being registered in the detector.

All in all the TRD DESY-2019 campaign was a success. The planned measurements were done and the quality of the data is acceptable despite the noise problems. The readout was stable and the developed data processing performed as expected.

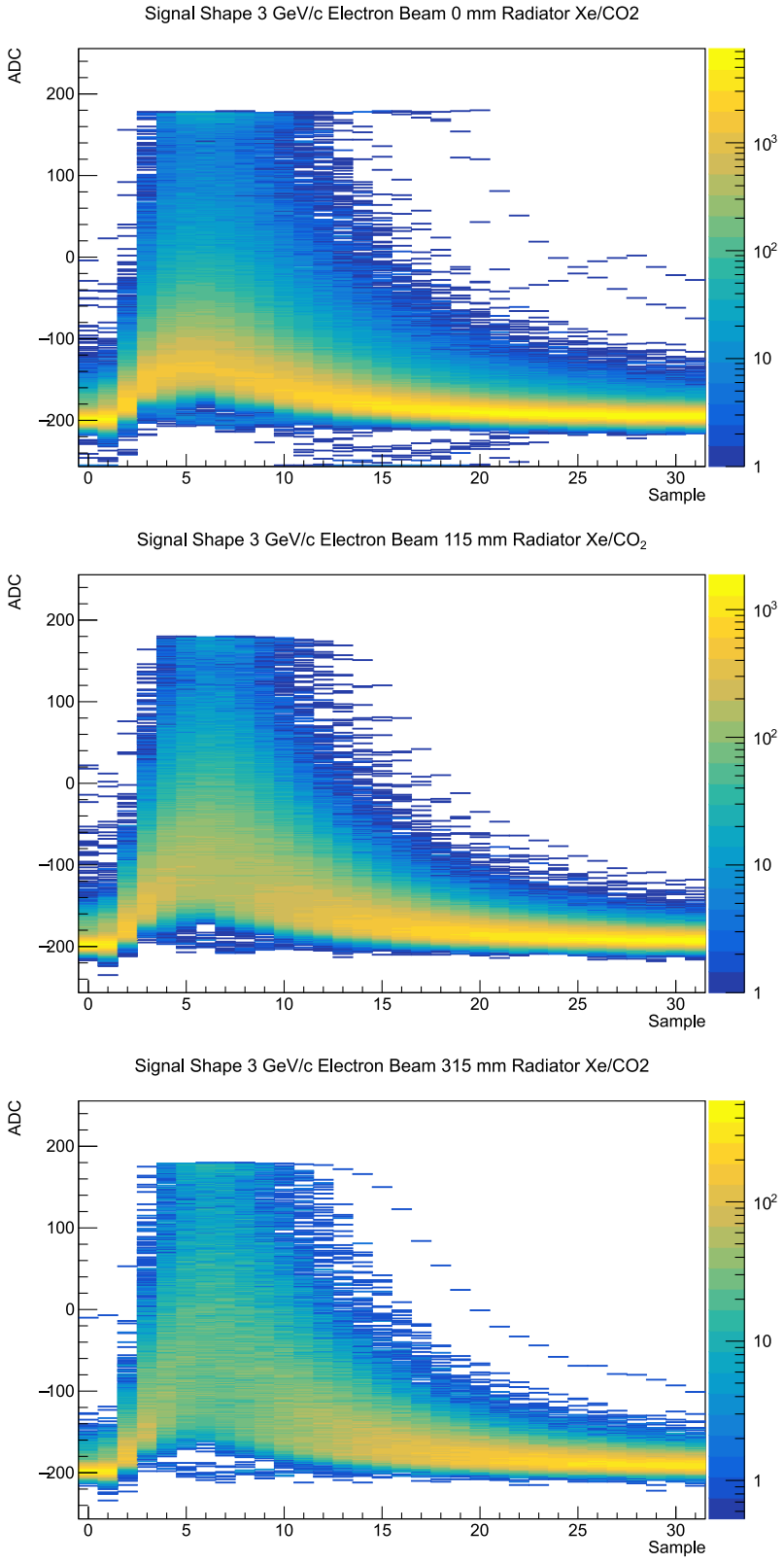

Figure 5.6: From top to bottom: Signal shapes without a radiator, with a 115 mm radiator and a 315 mm radiator. All at a beam energy of 3 GeV/c and a 85/15 Xe/CO_2 gas mixture.

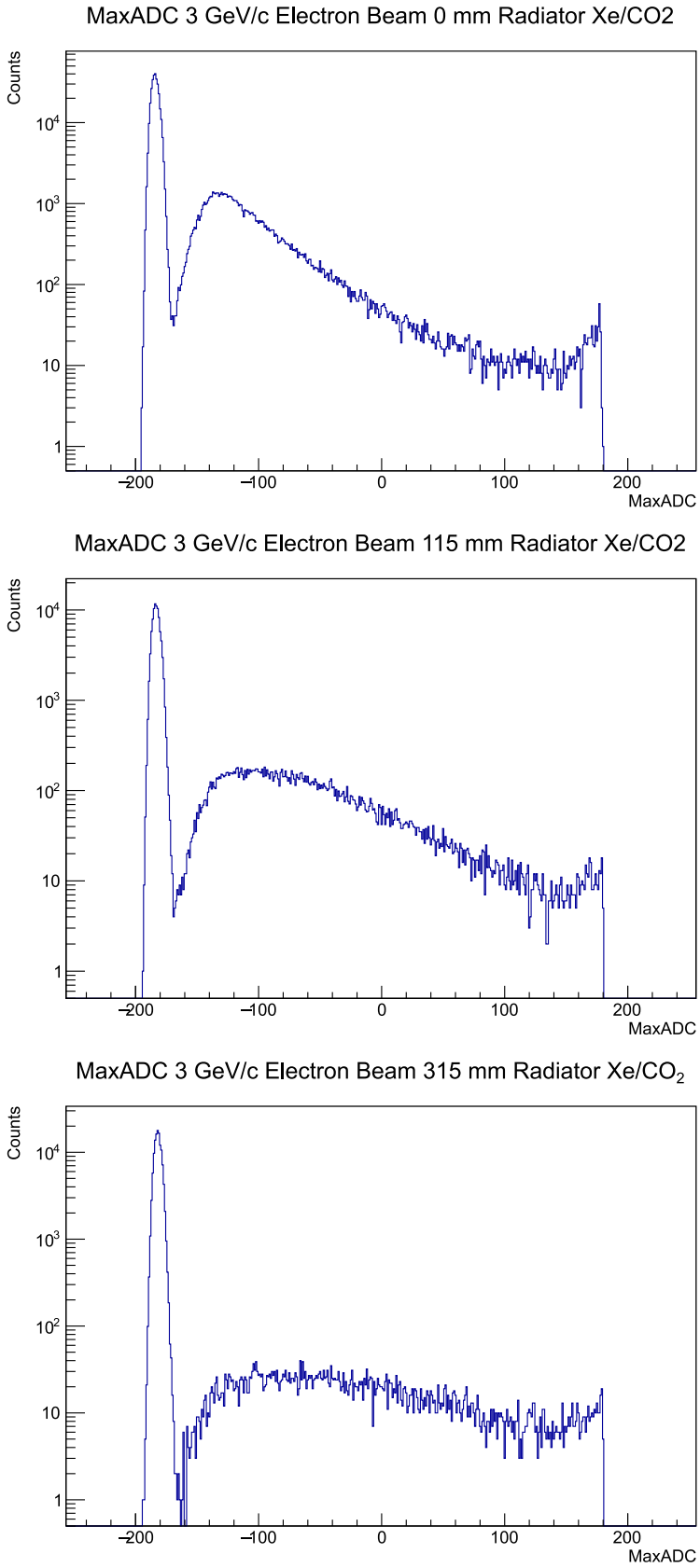

Figure 5.7: From top to bottom: MaxADC distribution without a radiator, with a 115 mm radiator and a 315 mm radiator. All at a beam energy of 3 GeV/c and a 85/15 Xe/CO_2 gas mixture.

Chapter 6

Conclusion and Outlook

6.1 Conclusion

The development of firmware for the DPB took a big step forward with the implementation of the data processing, which is now conform with the µSlice readout requirements of the CBM data acquisition scheme. The firmware developed in this work performed as desired in the laboratory as well as in the DESY-2019 test beam campaign. The required changes for the new SPADIC version 2.2 were implemented in the firmware and software. Additionally, the software framework was restructured to be more maintainable and easier to operate.

6.2 Outlook

The next step in the development of the TRD readout is the implementation of the CROB in the readout chain. The CROB is equipped with three GBTx ASICs, each allowing up to 14 uplinks and 2 downlinks. The GBTx ASIC provides transparent transport of the e-link frames from the SPADIC to the DPB and vice versa through optical connections. Additionally, the chips deliver a stable clock to the front-ends. The GBTx was specifically designed to run in an environment with very high levels of radiation and magnetic field [24].

Following the CROB implementation, the DPB firmware has to be ported to the CRI prototype. The CRI combines the FLIB and the DPB into one board. The access to the board is established over the PCIe bus, therefore the addressing software has to be rewritten as well.

After the CRI firmware is operational, work on feature extraction inside the FPGA can begin. The feature extraction can be divided into two stages. In the first stage, the relevant information from a single hit message is extracted, e. g. the channel ID, refined time information and the charge information. In the second stage a cluster finding algorithm can be employed to find hits that are most likely from a single event. From the corresponding charge distribution on the pads, the spatial position can be reconstructed in accordance to the pad response function [26]. The implementation of both feature extraction stages inside the FPGA would provide a significant reduction in the data rate on the FLIM link, while the implementation will be probed and designed also concerning FPGA capacity and reconstruction precision.

List of Acronyms

AFCK AMC FMC Carrier Kinter

ADC Analog to Digital Converter

AMC Advanced Mezzanine Card

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

BOM Buffer Overflow Message

BUF BUffer Full

CBM Compressed Baryonic Matter

CLI Command Line Interface

CP Config Parity

CRC Cyclic Redundancy Check

CRI Common Read-Out Interface

CROB Common Read-Out Board

CSA Charge Sensitive Amplifier

DAQ Data Acquisition

DDR Double-Data-Rate

DESY Deutsches Elektronen SYnchrotron

DPB Data Processing Board

DSP Digital Signal Processor

ECAL Electromagnetic Calorimeter

EOM End-Of-Message

EOS End-Of-Synchronization

FAIR Facility for Antiproton and Ion Research

FEB Front-End Board

FEE Front-End Electronics

FIFO FIFOFirst-In-First-Out

FLES First-Level Event Selection

FLIB FLES Interface Board

FLIM First-Level Interface Module

FMC FPGA Mezzanine Card

FNR Forced Neighbor Readout

FPGA Field-Programmable Gate Array

GBT GigaBit Transceiver

GEM Gas Electron Multiplier

GSI Helmholtzzentrum für Schwerionenforschung

HAL Hardware Access Library

HPC High-Pin Count

HV High-Voltage

ILA Integrated Logic Analyzer

LHC Large Hadron Collider

LUT Look-Up-Table

MAPS Monolithic Active Pixel Sensor

MIS MISsing Request

MMCM Mixed-Mode Clock Manager

MRPC Multi-gap Resistive Plate Chamber

MSB MeSsage Build

MUCH Muon Chamber System

MVD Micro-Vertex Detector

MWPC Multi-Wire Proportional Chamber

PCIe PCI express

PISO Parallel-In-Serial-Out

PLL Phase-Locked-Loop

PSD Projectile Spectator Detector

RDA Raw DAta

RICH Ring-Imaging Cherenkov Detector

ROC Read-Out Chamber

RPC Resistive Plate Chamber

SIS SchwerIonenSynchrotron

SMX STS-MUCH-XYTER

SOM Start-Of-Message

SOS Start-Of-Synchronization

SPADIC Self-triggered Pulse Amplification and Digitization asIC

STS Silicon Tracking System

TOF Time Of Flight

TR Transition Radiation

TRD Transition Radiation Detector

TS Timing System

TS_MSB TimeStamp Most Significant Bits

UNU UNUsed Request

µMISC µ-Miscellaneous-Message

µRDA µ-Raw-DAta

µSOM µ-Start-Of-Message

Bibliography

[1] The CBM-TRD software guide. URL http://praisig.gitpages.cbm.gsi.de/cbmtrd-software-guide/daq_software/DAQ_SOFTWARE.html.

[2] The SPADIC project website. URL http://spadic.uni-hd.de.

[3] Technical Design Report for the CBM Superconducting Dipole Magnet. Technical report, Darmstadt, 2013. URL http://repository.gsi.de/record/109025.

[4] Technical Design Report for the CBM Ring Imaging Cherenkov Detector. Technical report, 2013. URL http://repository.gsi.de/record/65526.

[5] Technical Design Report for the CBM Silicon Tracking System (STS). Technical report, Darmstadt, 2013. URL http://repository.gsi.de/record/54798.

[6] Technical Design Report for the CBM Time-of-Flight System (TOF). Technical report, Darmstadt, 2014. URL http://repository.gsi.de/record/109024.

[7] Technical Design Report for the CBM : Muon Chambers (MuCh). Technical report, Darmstadt, 2015. URL http://repository.gsi.de/record/161297.

[8] Technical Design Report for the CBM Projectile Spectator Detector (PSD). Technical report, Darmstadt, 2015. URL http://repository.gsi.de/record/109059.

[9] Technical Design Report for the CBM Transition Radiation Detector (TRD). Technical report, Darmstadt, 2018. URL http://repository.gsi.de/record/217478.

[10] Technical Design Report for the CBM Online Systems Part I Data Acquisition. Technical report, Darmstadt, In development. URL https://git.cbm.gsi.de/doc/tdr-online.

[11] T. Armbruster. SPADIC a Self-Triggered Detector Readout ASIC with Multi-Channel Amplification and Digitization. May 2013.

[12] P. Fischer. SPADIC 2.2 chip manual, work-in-progress, 2019.

[13] V. Friese. Estimate of the CBM computing requirements for operation at SIS-100. *CBM Computing Nodes 2018. CBM-CN-18001*, 2018. URL https://indico.gsi.de/event/7449/contribution/0.

[14] V. Friese and the CBM Collaboration. Challenges in QCD matter physics –The scientific programme of the Compressed Baryonic Matter experiment at FAIR. *The European Physical Journal A*, 53(3), Mar 2017. ISSN 1434-601X. doi:10.1140/epja/i2017-12248-y.

[15] B. Friman, C. Hohne, J. Knoll, S. Leupold, J. Randrup, R. Rapp, and P. Senger. The CBM physics book: Compressed baryonic matter in laboratory experiments. *Lect. Notes Phys.*, 814:pp.1–980, 2011. doi:10.1007/978-3-642-13293-3.

[16] K. Fukushima and T. Hatsuda. The phase diagram of dense QCD. *Reports on Progress in Physics*, 74(1):014001, Dec 2010. ISSN 1361-6633. doi:10.1088/0034-4885/74/1/014001. URL http://dx.doi.org/10.1088/0034-4885/74/1/014001.

[17] D. Hutter. FLIB Installation Guide. 2014. URL https://github.com/cbm-fles/flesnet/wiki/.

[18] D. Hutter and J. de Cuveland. The FLES Detector Input Interface. Feb 2016.

[19] K. Kasinski, W. Zablotny, J. Lehnert, R. Szczygiel, W. Zubrzycka, A. Rodriguez-Rodriguez, and C. J. Schmidt. SMX2 and SMX2.1 Manual v2.0, 2018.

[20] G. Kasprowicz. Project website of the AFCK at the OHWR (open hardware repository). URL https://www.ohwr.org/projects/afck.

[21] M. Krieger. SPADIC design review, talk given at GSI, May 2016.

[22] P. Kähler and F. Roether. The transition radiation detector in the CBM experiment at FAIR. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, page 162727, 2019. ISSN 0168-9002. doi:https://doi.org/10.1016/j.nima.2019.162727.

[23] C. G. Larrea, K. Harder, D. Newbold, D. Sankey, A. Rose, A. Thea, and T. Williams. IPbus: a flexible ethernet-based control system for xTCA hardware. *Journal of Instrumentation*, 10(02):C02019–C02019, Feb 2015. doi:10.1088/1748-0221/10/02/c02019.

[24] P. Moreira, J. Christiansen, and K. Wyllie. GBTX manual, 2018.

[25] P. Reichelt. Simulationsstudien zur Entwicklung des Übergangsstrahlungszählers für das CBM-Experiment, 2011.

[26] F. Roether. Data reduction by feature extraction. Technical Report CBM Progress Report 2018, Darmstadt, 2019. URL http://repository.gsi.de/record/220128.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderen fremden Texten entnommen wurden, sind von mir als solche kenntlich gemacht worden. Ferner erkläre ich, dass die Arbeit nicht - auch nicht auszugsweise - für eine andere Prüfung verwendet wurde.

Frankfurt, den 18.11.2019 David Schmidt