PHYSICS

Master Thesis

Integration and design for the ALICE ITS readout chain

By: Gitle Mikkelsen

Supervisor: Johan Alme

June 1, 2018

Abstract

ALICE and its Inner Tracking System detector in the LHC at CERN will undergo a major upgrade during its second long shutdown taking place in 2019 and 2020. New silicon sensors called ALPIDE will be utilized, which requires a new readout electronics system due to different interfaces and higher demands regarding bandwidth and latency. The readout chain consists of staves containing the sensors, a layer of readout unit boards gathering data from and controlling the sensor staves, and a layer of common readout units multiplexing and compressing data from the readout units before forwarding it to the O^2 data center system.

As part of the ALICE collaboration, The University of Bergen is in charge of the development of several components in this readout chain. All development sites for the readout electronics should have the readout chain in place so that design and integration tasks can be done locally. As part of the work in this thesis, the ITS readout chain is integrated and tested. The working readout chain is then used to develop various control communication interfaces along the chain, such as an I^2C interface for an auxiliary FPGA on the readout unit, and a high-speed interface for uploading data to the flash memory on the readout unit.

Acknowledgments

I would like to thank my supervisor *Associate Professor Johan Alme* for all the help and guidance needed to write this thesis. I would also like to thank the various people whom I have worked with, whose work I am building on, or who provided help at some point in the thesis, including *Magnus Ersland, Attiq Ur Rehman*, *Magnus Rentsch Ersdal, Simon Voigt Nesbø, Matthias Bonora, Matteo Lupi, Joachim Schambach, Arild Velure, Ola Slettevoll Grøttvik, Jozsef Imrek, Shiming Yuan, Piero Giubilato, Filippo Costa, Hartmut Hillemanns, Tomas Vanat* and *Krzysztof Marek Sielewicz*.

I would also like to thank *Professor Kjetil Ullaland* for leading the Microelectronics class of '16 and providing guidance and education through the last two years, as well as my fellow students of this class. I am also grateful to *Professor Dieter Röhrich* for heading the ALICE collaboration team at the University of Bergen and giving me the opportunity to work with this.

Finally, I would like to thank my family for the continuous love and support.

Nomenclature

ALICE	A Large Ion Collider Experiment (One of four main experiments in the LHC)
ALPIDE	Alice Pixel Detector (Sensor IC used in the ITS)
CERN	The European Organization for Nuclear Research
CRU	Common Readout Unit (Component of the ITS readout chain)
DAQ	Data Acquisition
DCS	Detector Control System
DMA	Direct Memory Access
FIFO	First In First Out queue
FLP	First-Level Processing (First level of O2 facility)
FPGA	Field-Programmable Gate Array (Type of reconfigurable IC)
FSM	Finite State Machine
GBT	GigaBit Link (A high-bandwidth, radiation hard, optical link)
GBT-SCA	GBT Slow Control Access IC
GBTx	IC that implements the GBT protocol
I^2C	Two-wire protocol for serial communication
IC	Integrated Circuit
ITS	Inner Tracking System (Central detector of ALICE)
LHC	Large Hadron Collider (Particle Accelerator)
LTU	Local Trigger Unit
O^2	Computing facilty of ALICE
PA3	ProAsic 3 (Flash-based, auxiliary FPGA on the RU)

RU	Readout Unit (Component of the ITS readout electronics)
RUv0-CRU	RUv0 with CRU emulation firmware
RUv1	Version 1 of the Readout Unit
SEU	Single Event Upset (A type of radiation effect where a memory bit changes value unexpectedly)
SWT	Single Word Transaction (Custom GBT frame that does not contain experiment data)
TPC	Time Projection Chamber (A detector in ALICE)
TTC	Timing and Trigger Control
UART	Universal Asynchronous Receiver-Transmitter
WP10	Work Package 10 (ITS readout electronics development project)

Contents

1 Background

1.1 CERN, ALICE and ITS

The European Organization for Nuclear Research (CERN) is a scientific research organization located on the border between Switzerland and France near the city of Geneva[1]. CERN is most known for its particle accelerator complex, consisting of a succession of machines that accelerate particles in order to increase their energies. Each machine boosts the energy of a beam of particles, before passing it on to the next. The last step in this chain is the Large Hadron Collider (LHC), the largest particle accelerator in the world, capable of pushing the beams to energies of 6.5 TeV, at 99.9999991% the speed of light. It consists of a ring-shaped tunnel, with a circumference of 27 km, buried between 50 and 175 meters below ground. This tunnel can be seen in figure 1. Inside the tunnel is two beam pipes surrounded by various electromagnets and radiofrequency cavities which accelerate and guide the beam of particles. The two beams travel in opposite directions, and they are eventually made to collide by crossing their paths. By studying how the particles behave during these high-energy collisions, one can learn about fundamental physics.
Distributed along the LHC ring, there are four major experiments, which record data from the particle collisions. One of these experiments is A Large Ion Collider Experiment (ALICE). ALICE studies the properties of quark-gluon plasma, a state of matter formed under extreme temperature and density, which the LHC can produce during the particle collisions. Such matter was likely what the universe consisted of in the moments right after the Big Bang. ALICE is an international collaboration of around 174 institutes in around 42 countries. One of these is the University of Bergen.
The Inner Tracking System (ITS) is the detector at the heart of ALICE[2]. It consists of seven barrel-shaped layers of sensors wrapped around the beam pipe where the particle collisions take place. Its main task is to locate the primary vertex of collisions with high accuracy. Figure 2 shows ALICE with ITS highlighted.

1.2 ITS upgrade

During the second long shutdown of the LHC, scheduled to take place from 2019 to 2020, ALICE, including the ITS, will undergo a major upgrade to improve its resolution and data rate for higher accuracy. The current sensor system of the ITS will be

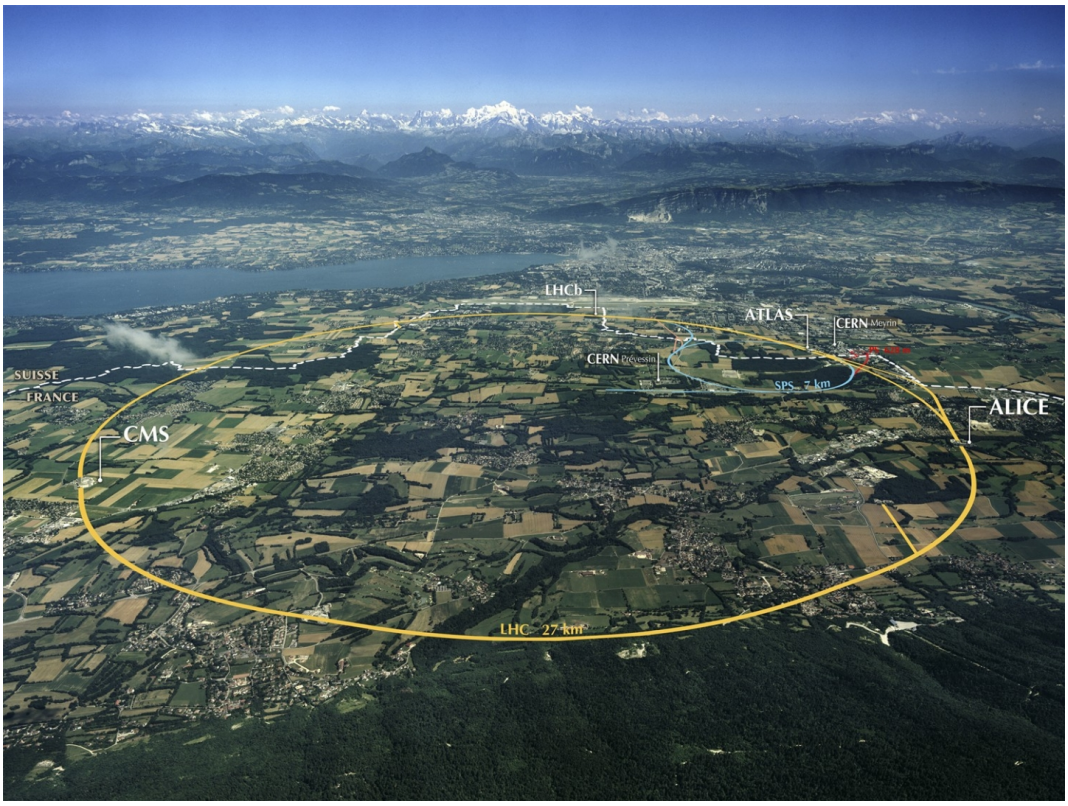

Figure 1: LHC seen from the air

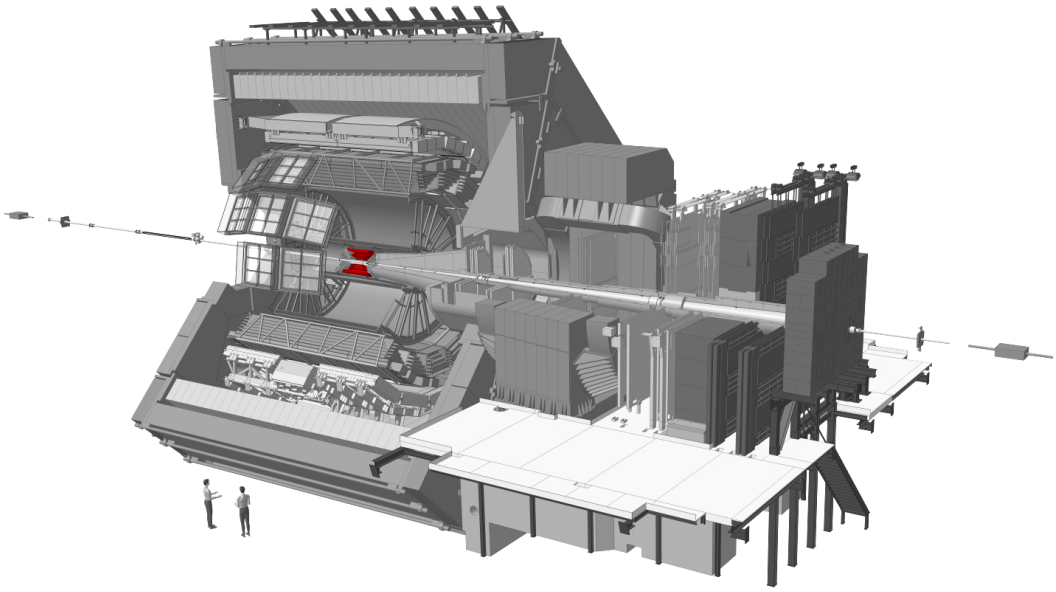

Figure 2: Overview of ALICE, with ITS highlighted in red[3]

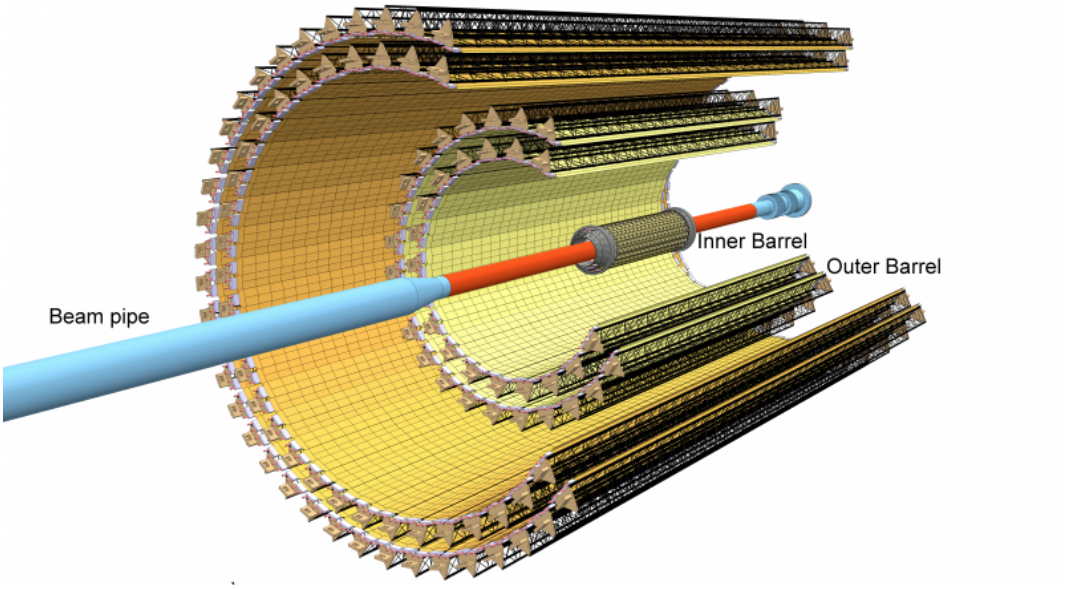

Figure 3: The layers of sensor staves in the ITS after its upgrade

replaced by silicon pixel ALPIDE (Alice Pixel Detector) chips, developed by CERN specifically for the ITS upgrade[4]. These sensor chips will be mounted on staves, and the staves will be arranged concentrically around the beam pipe, separated into an inner barrel consisting of three layers, and an outer barrel consisting of four layers, as shown in figure 3. The radius of the sensor barrels will vary from 22 mm in the innermost layer, to 400 mm in the outermost layer[5].

The ALPIDE sensors use entirely different interfaces for control and data readout than the existing sensor systems. In addition, the data bandwidth is greatly increased. For these reasons, the readout electronics is to be upgraded as well. The readout chain will be redesigned to use a layer of Common Readout Units (CRU) to combine and multiplex data from the detector before forwarding it to the data computing system for processing and storage. This computing system is also being upgraded during the second long shutdown. The new computing system is called O^2[6]. Each CRU is connected to several Readout Units (RU), which again are connected to the actual sensor staves.

1.3 About this thesis

1.3.1 Objective

The University of Bergen is part of the ALICE collaboration, and is responsible for part of the development of the new ITS readout electronics. It will therefore be necessary to set up a readout chain in Bergen for development and testing purposes. The setup should be as similar as possible to the ones used in other collaborating development sites such as CERN and The University of Texas at Austin, but it will not be possible to exactly replicate these setups since mass production of several components of the readout electronics has not started and existing supply is not enough for distributing the same components to all development teams. The software and firmware of the setup must therefore be adapted to the available hardware. The setup must be tested to ensure functionality.
One of the university's responsibilities is designing firmware for an auxiliary FPGA (PA3) on the readout unit, a circuit board which is an important part of the readout electronics. In this thesis various interfaces between this FPGA and the top level software is designed and tested using the local readout chain setup. These interfaces include access to the register bus, and a high-speed interface for uploading data to a flash memory chip on the readout unit.

1.3.2 Structure

The readout electronic chain and its most relevant components and protocols are introduced in more detail in chapter 2. In chapter 3, the local setup of the readout chain is integrated and tested in various configurations. In chapter 4 an interface for accessing the register bus on the RU PA3 FPGA from top level software is implemented and tested. In chapter 5, a higher-speed interface for writing data to the flash memory on the RU is implemented and tested. In chapter 6, participation in a radiation testing campaign for the RU and its components is discussed. Finally, the thesis is summed up and the results discussed in chapter 7.
Appendixes include more detailed information about setting up the CRU and readout chain, structured as a manual, as well as more details about testing the CRU, and finally a manual for using the software for the PA3 developed in chapter 4 and 5.

2 The upgraded ITS readout electronics chain

2.1 Structure

In total the upgraded ITS will consist of 24 120 ALPIDE detector chips, which results in a detection area of 10 m² with more than 12.5 billion pixels[7]. Each stave is connected to its own readout unit, resulting in a total of 192 readout units as there will be 192 sensor staves in the ITS.
The readout units are electronic boards that are tasked with controlling and gathering data from the ALPIDE sensor chips on the staves. They will also forward trigger information from the ALICE trigger system to the sensor chips. The readout units are to be mounted approximately 4 meters away from the sensor barrels. The beam collisions will produce a lot of radiation in the area around ALICE, therefore the readout units will require radiation hardening in order to operate properly. In the next layer of the chain are common readout units, which will combine and multiplex data from multiple readout units before forwarding it to the data computing facility (O^2) for processing and storage. The CRUs are mounted in computers located in an intermediary computer room, called the counting room, away from the radiated ALICE cavern and does therefore not require radiation hardening like the readout units do. These computers are reachable from the main Detector Control System (DCS) over the network. DCS controls the readout chain by sending commands and monitoring the system. Experiment data is forwarded from the FLP node to the O^2 computing system for processing and storage. A block diagram overview of the readout chain is shown in figure 4.

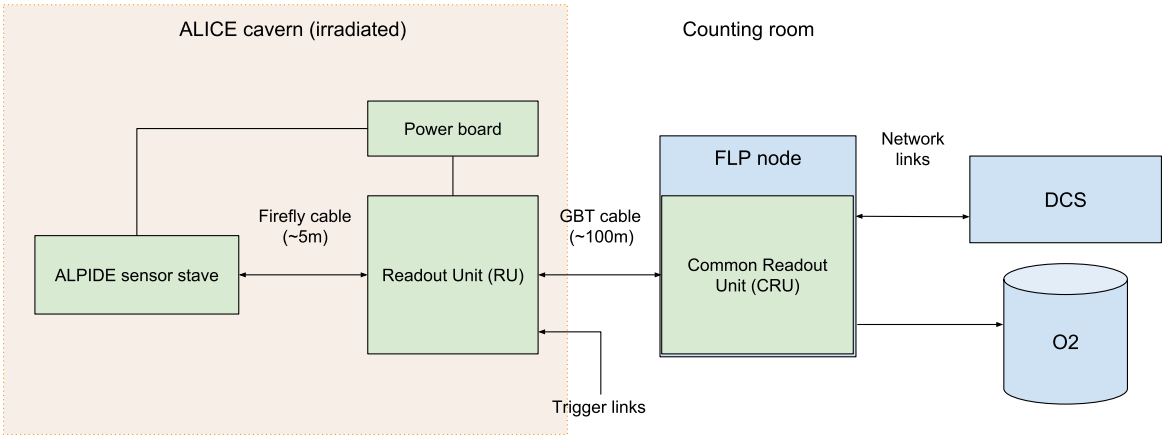

Figure 4: Block diagram of the ITS readout chain after the upgrade

2.2 Readout Unit (RU)

The readout unit is a circuit board with several components and ports. It is located between the sensor staves and the common readout units in the readout chain.
The heart of the board is the main FPGA, a Xilinx Ultrascale, which runs the main firmware that manages the readout process and stream of data. In figure 5, the Ultrascale is hidden under the heatsink. There is also secondary flash-based FPGA, a Microsemi ProAsic3 A3PE600L, hereafter referred to as PA3. In figure 5, the PA3 and flash memory chip are the two black ICs in the upper left corner of the board, as marked.
The readout unit includes high-speed Firefly ports for electrically connecting to its designated ALPIDE chip stave. It also contains three GBTx chips and slots for suitable optical transceivers like VTRx's for connecting to a CRU using the GBT link. Up to three GBT uplinks (from the CRU to the RU) can be used for data, although in normal operation (50 kHz Pb-Pb collisions) only one uplink is needed. One of the GBT links will also be used for control, and one will be used for trigger information. The GBTx chip used for control is connected to a GBT-SCA ASIC, which can be used for various slow control tasks. The main communication with the PA3 will use an I^2C master module on the GBT-SCA. This interface is discussed in chapter 4.
The readout unit hardware, firmware and software development is organized as a project called Work Package 10 (WP10), assigned to a team of engineers and students. The University of Bergen is in charge of designing the firmware for the PA3.

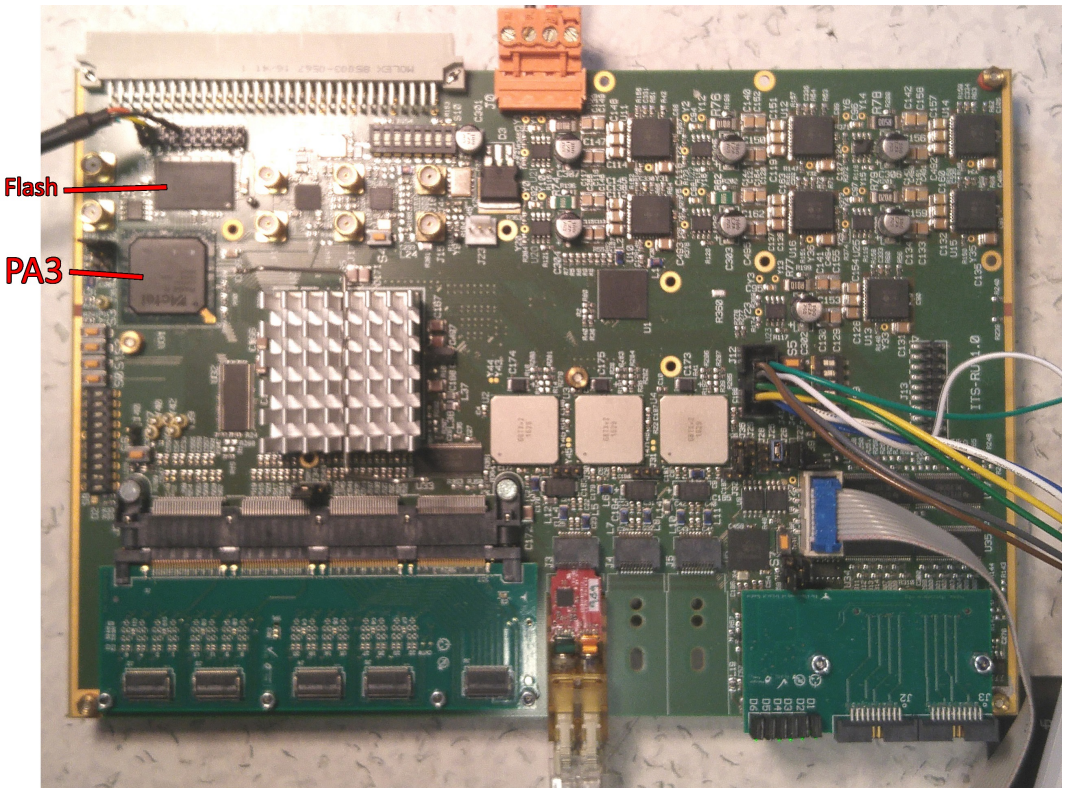

Figure 5: The Readout Unit board, version 1

2.2.1 PA3

The PA3 auxiliary FPGA on the RU handles configuration of the Ultrascale, using data from a flash memory chip which it can write to and read from. In addition to initial configuration at power-up, the PA3 will also continuously re-configure the Ultrascale during operation, by overwriting configuration memory without actually resetting or pausing the Ultrascale operation. This technique is known as scrubbing, and it should ensure a reliable operation of the main FPGA as any unwanted error in the configuration caused by single event upsets (SEU) will be cleaned within seconds, without having to shut down the detector and lose valuable experiment time[8]. The programming and scrubbing of the Ultrascale happens through the selectmap protocol, one of several interfaces on the Ultrascale that can be used to access its configuration memory.

The firmware on the PA3 consists of a flash interface with accompanying read and write controllers, which together handle access to the external flash memory chip. The write controller is designed in chapter 5 of this thesis. There is also the config-

uration controller, which controls the selectmap interface to the Ultrascale for configuration and scrubbing. The register interface is implemented as a Wishbone bus, accessible by an I²C module. A block diagram of the full firmware as of version A200 can be seen in figure 6.

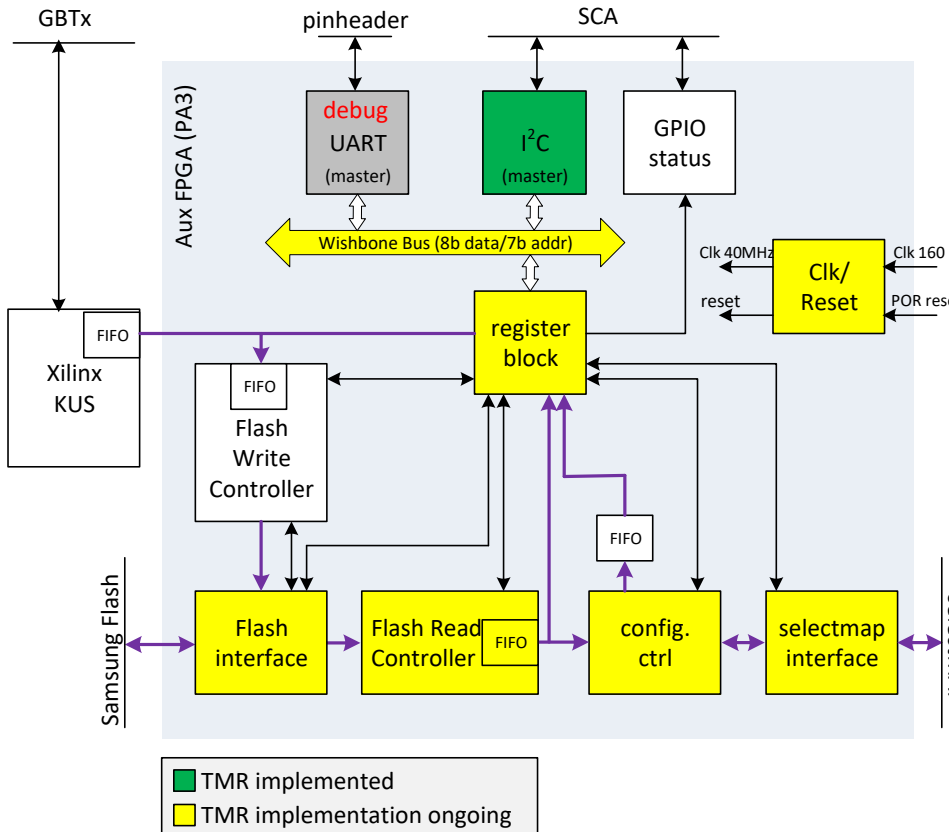

Figure 6: Block diagram of the firmware on the PA3 FPGA.

2.3 Common Readout Unit (CRU)

The CRUs act as an interface between the Front Electronics (RUs with ALPIDE sensors), the detector control system, the data computing facilities, and the trigger networks[9]. The CRU mainly consists of an FPGA and multiple data links, specifically optical GBT links connected to RUs, TTC-PON carrying trigger and timing information, and a PCI-express interface for communicating with its host computer which is part of the O² First Level Processing (FLP) system. The CRU will organize (tag and multiplex) and compress (discard useless frames etc.) the data to reduce the bandwidth requirement for the final readout chain steps.

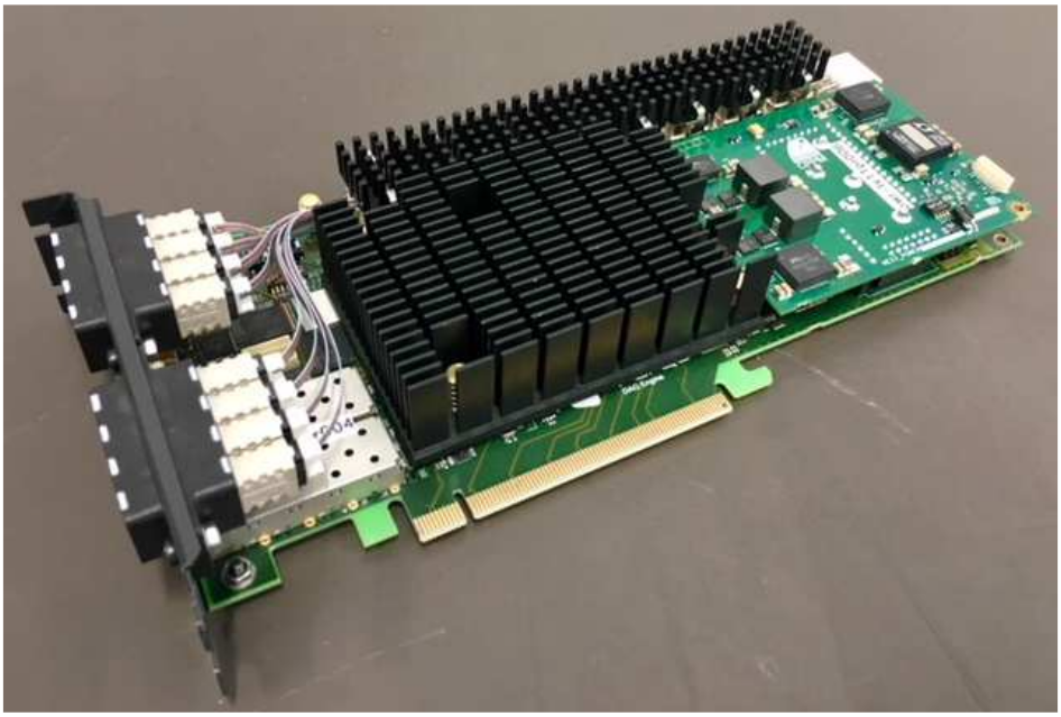

Figure 7: Version 2 of the PCIe40 DAQ board CRU implementation[10]

The final CRUs are implemented on the PCIe40 DAQ board that are developed by the LHCb team, a different experiment of the LHC. This board can be seen in figure 7. The FPGA on this board is an Intel Arria 10 GX (10AX115S3F45E2SG). For development and testing purposes, a version of the CRU firmware has also been implemented on an Arria 10 GX development kit board, and individual teams such as the ITS WP10 team have implemented CRU emulators on other hardware such as prototype readout unit boards, because the final CRU boards have very limited availability during the development phase.
The common modules of the firmware for the CRU is developed by the ALICE DAQ group. The firmware is modular and its base will include the necessary modules for the interfaces common to most detectors in ALICE. The CRUs will not only process data from ITS, but also other detectors, such as the Time Projection Chamber (TPC). Each of these systems have their own requirements for the CRU and GBT links. Therefore the system must be able to accommodate a variety of specifications and functionalities. For this purpose, a user logic module at the heart of the firmware is left to be customized by detector teams if they need additional features or specialized behavior. A typical user logic module would forward trigger data from the trig-

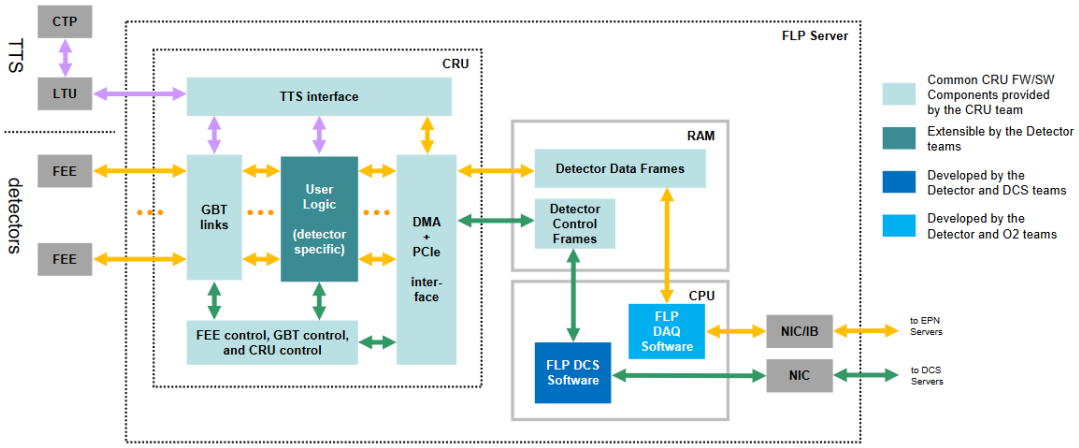

Figure 8: Block diagram of the FLP node with the CRU and its firmware[10]

ger network, specifically Local Trigger Units (LTU), to the RUs, and forward readout data from the RUs to the PCI-e Direct Memory Access (DMA) interface. The data may be divided and packaged based on triggers, or it may forward the data as-is. For the ITS detector, the CRU does not process triggers due to especially strict latency requirements. Instead, the LTUs are directly connected to the RUs, as can be seen in figure 4.

Each CRU will have 24 GBT links available[11]. Since each RU uses 3 GBT links, this means that each CRU can be connected to a maximum of 8 RUs. Each FLP node will house one CRU. A block diagram of the FLP node with CRU can be seen in figure 8.

2.4 GBT Link

The GigaBit Transceiver (GBT) architecture is an optical serial data link developed at CERN, designed for use in the LHC, which requires high bandwidth as well as radiation hardening[12]. It is frame based, with one 120-bit frame transmitted continuously at an interval of 25 ns. This results in a raw serial line rate of 4.8 Gb/s. 25 nm corresponds to the LHC bunch crossing interval. The bunch crossing interval is the time between bunches of particles crossing each other in the LHC. In other words it is the time between potential collisions.

The GBT link is implemented in the ITS readout chain in two ways. A radiation hardened ASIC called GBTx is used on the readout unit. This ASIC can can accept data in parallel as input, serialize and encode the data, and output it to a laser transmitter, and opposite for the downlink. The laser transmitter used is a custom unit

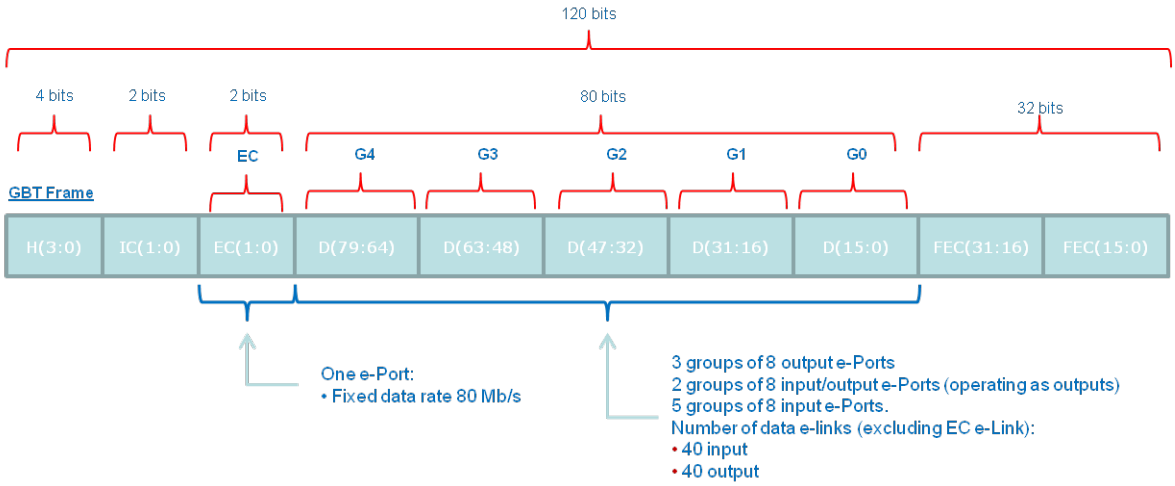

Figure 9: GBT standard frame structure

also designed at CERN to be radiation hardened. On the common readout unit, the GBT link controller is implemented as a module on the FPGA.

The GBT protocol specifies three different frame modes; the standard GBT frame mode, the wide frame mode, and the 8B/10B frame mode.

ITS will use the standard GBT frame, illustrated in figure 9. This frame starts with a 4 bit long header. The header can be either 0b0101, which signals that the frame contains valid data, or 0b0110, which signals the opposite, for example if the transmitter is idle, or the frame contains non-data information, such as Single Word Transactions (SWT). Then follow 4 bits for slow control information, of which the first 2 bits are for Internal Control (IC), strictly reserved for control of the GBTx ASIC. The last 2 bits for slow control is for External Control (EC). Next follows the main data payload of 80 bits. The data and EC fields are not pre-assigned and can be used for different purposes such as Data Acquisition (DAQ), Timing and Trigger Control (TTC) or experiment control, depending on requirements. The last 32 bits are used for error correction. This leaves 84-bits per frame, or 3.36 Gb/s, of usable bandwidth, of which 3.2 Gb/s is dedicated to data.

Before transmitting the frame, the data, EC and IC fields are fed through a scrambling algorithm which DC balances them. Then, a Reed-Solomon encoder generates the 32 error correcting bits based on the scrambled data in addition to the header. The receiver does the opposite; first decoding and checking the error correction bits, then de-scrambling the data before the IC, EC and data fields can be read. This is illustrated in figure 10.

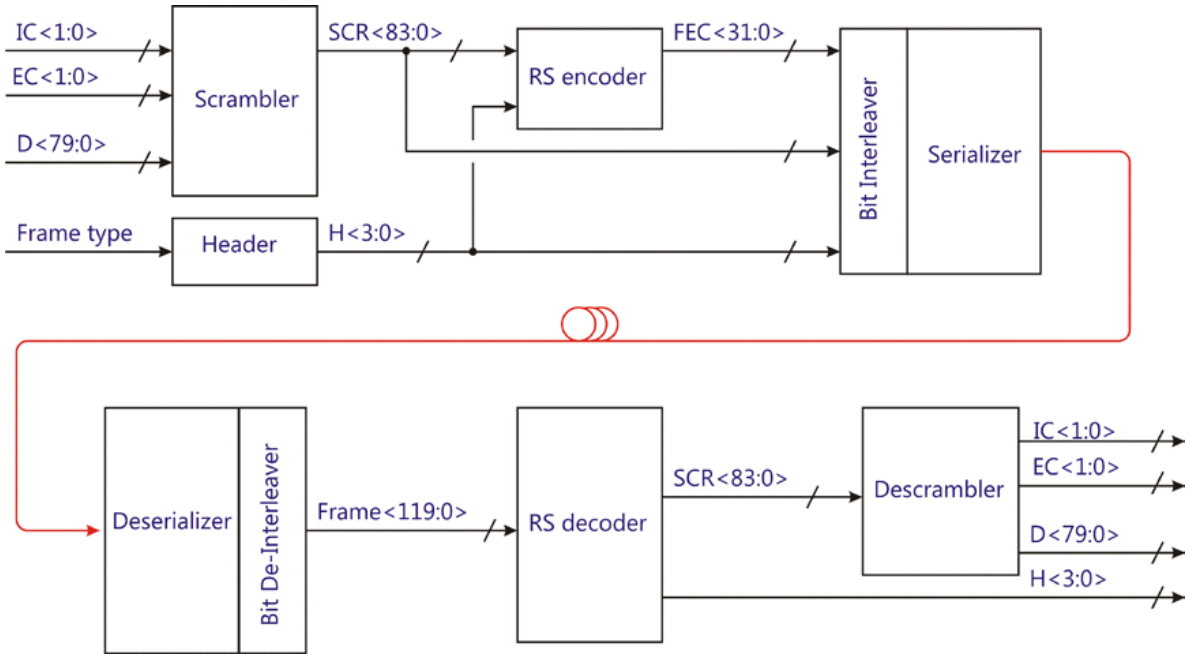

Figure 10: GBT link block diagram

The 4-bit header is used to track frames and synchronize the receiver to the transmitter. The header is not affected by the scrambling so that it can be easily detected. When a GBT receiver is powered up, it enters a frame-lock acquisition mode in which it searches for valid headers. Once a configurable amount of frames with valid headers have been detected in succession, it considers the link established and enters frame-tracking mode. In this mode, it receives data and operates normally, while keeping track of invalid headers. Once a configurable amount of frames in succession is found to be invalid, it considers the synchronization lost and re-enters the acquisition mode. Typically multiple invalid frames are needed to trigger this, so that occasional random single event upsets aren't enough to cause the link to fall out of synchronization.

The data field (80-bit) of the GBT frame is used to transmit the data. GBT frames are differentiated into control frames and data frames, with the header specifying data valid for the latter only. Control frames start with a 4 bit identifying header. Four headers are defined: IDLE, SOP (Start Of Packet), EOP (End Of Packet) and SWT. IDLE frames contain no information. SOP and EOP, as the names suggest, mark the start and end of packets of data from the detectors. They contain various metadata relating to the packets such as length and tags.

Single Word Transactions frames can contain arbitrary data used for special control or data transfers. In the GBT downlink, this will normally be the only type of GBT frames. In the uplink, SWT frames may only be sent in between data frames, in other words between EOP and SOP control frames. In the ITS readout electronics, SWT frames are for example used to access the register bus on the readout unit main FPGA.

2.4.1 Slow Control

Part of the GBT link is the slow control system. The 2 bytes in the EC field of the GBT frame payload is forwarded to a dedicated ASIC for slow control called GBT-SCA. This chip is part of the readout unit board as mentioned. On the CRU main FPGA, the SCA communication is implemented as part of the GBT VHDL module. The GBT-SCA ASIC contains several communication modules, including a range of GPIO, ADC and DAC pins, as well as I²C, SPI and JTAG masters[13]. These modules are connected to various components on the board such as the FPGAs. An I²C module will be used for communication with the readout unit PA3 auxiliary FPGA in an interface that is discussed in chapter 4.
Communication with the GBT-SCA is done using the High Level Data Link Control (HDLC) serial protocol. This protocol is command based. Rather than reading and writing directly to registers, transactions specify a command ID, a transaction ID and data if the command requires it. Command IDs determine what the GBT-SCA chip will do, for example writing or reading registers or executing operations. Every command transaction returns a package with the same transaction ID. The return packet contain status info and returned data if there is any.
The IC slow control field is used for accessing the GBTx registers, for configuration and monitoring. This field can also control the laser transceivers through a master communication module on the GBTx chip, accessible through its registers.

2.5 ALF/FRED (DCS interface)

The readout process is monitored and controlled by the ALICE Detector Control System (DCS). The DCS system accesses the readout chain through the FLP node and CRU over a network link. One of the protocols considered for the communication between the CRU and DCS is called ALF (On the CRU side) and FRED (On

the DCS side). This protocol is based on Distributed Information Management System (DIM). DIM is a communication system for distributed/mixed environments, originally developed for one of the experiments of the Large Electron–Positron Collider, an earlier particle accelerator at CERN[14]. It provides a network transparent inter-process communication layer.
The CRU host computer runs a DIM server, which acts as a bridge between the DIM network and the CRU driver, allowing DCS to communicate with the CRU from the control center without physical access to the CRU host computer.

3 Readout chain setup and integration

3.1 Variations of the setup

Several different implementations of the ITS readout chain has been set up and tested. Special focus is given to the CRU, as well as control functionality, as the setup in Bergen will at first be mostly used for development of the CRU and the PA3 auxiliary FPGA on the RUv1, rather than data readout related activities.

3.1.1 Arria 10 DK CRU and VLDB

This implementation of the readout chain consists of the Arria 10 Development Kit CRU, and a VLDB (The Versatile Link Demo Board) in place of a readout unit. As introduced in chapter 2.3, the Arria 10 DK runs the same firmware as the final PCIe40-based CRU. The VLDB is a demonstration and development board for the GBT link system. It can be seen in figure 11. The board contains a GBTx chip with one GBT link and all e-links exposed on mini-HDMI ports, a GBT-SCA chip with various modules such as I²C, GPIO, ADC, DAC and SPI exposed to pins, and custom FEASTMP rad-hard DC-DC converters developed at CERN. It is used to test the main functionality of the CRU and GBT link before it is replaced by a readout unit in the setup. The setup is illustrated in figure 12.

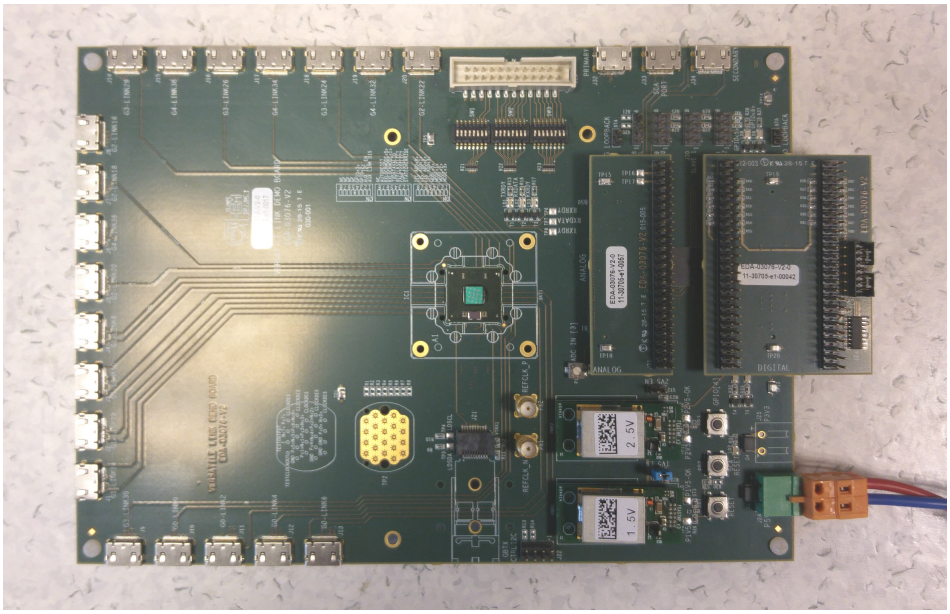

Figure 11: The VLDB board

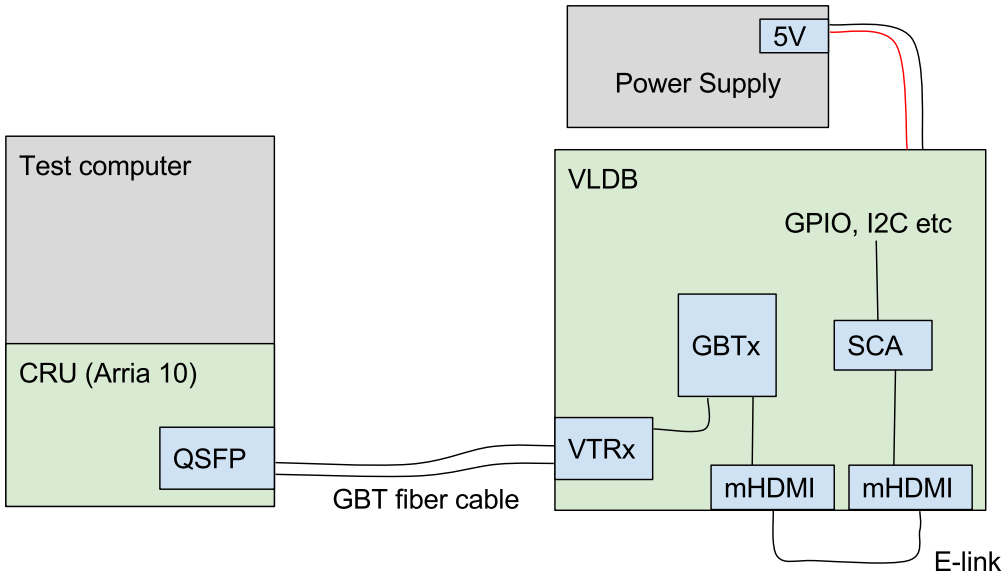

Figure 12: Block diagram of hardware setup with the Arria 10 DK CRU and the VLDB

3.1.2 Arria 10 DK CRU and RUv1

This setup replaced the VLDB with a readout unit, as can be seen in figure 13. The CRU to RU link is based on the same components and protocol (GBT) as the CRU to VLDB link, so in theory the integration procedure should be similar. Version 1 of the readout unit (RUv1 for short) is used. This board is described in chapter 2.2. This is the setup that most accurately reflects the actual readout chain that will be used in operation in ALICE, with one exception: It lacks the ALPIDE sensor chip as software for controlling the sensor interface module on the readout unit is not ready for the Arria10 DK CRU implementation as of this thesis's completion.

3.1.3 RUv0-CRU and RUv1

In this setup the Arria 10 DK CRU is replaced with a readout unit version 0 running CRU emulation firmware. An ALPIDE sensor slave will also be connected to the RU in this setup. The setup is illustrated in figure 14.
During the development phase, very few units of either the final PCIe40 board or the Arria 10 DK board is available. There are not enough boards to provide all the detector teams with their own CRU for development and testing purposes. For this

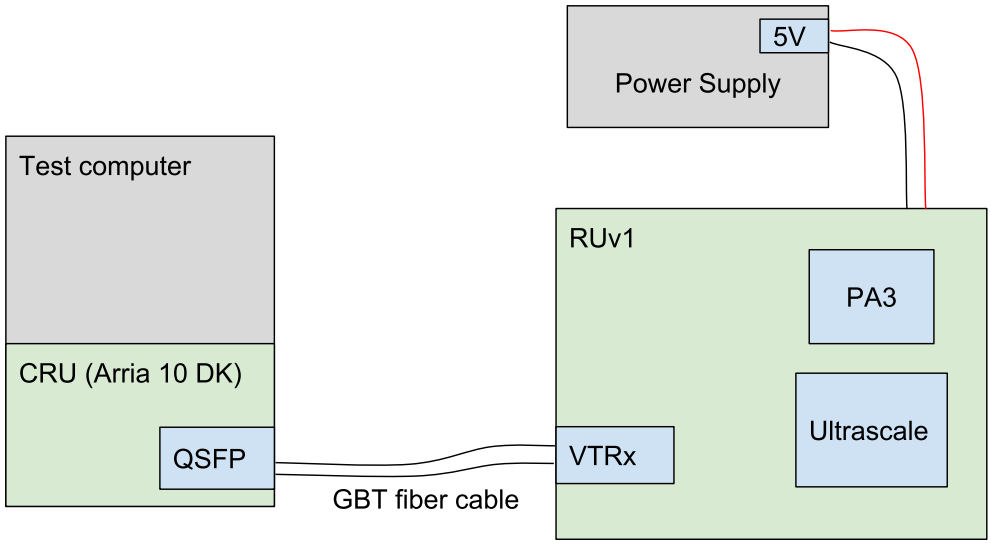

Figure 13: Block diagram of hardware setup with the Arria 10 DK CRU and the RUv1

reason, the ITS team uses a different board to emulate the CRU, namely an older version (v0a) of the Readout Unit, RUv0a for short, seen in figure 15. When used as the CRU emulator, it is referred to as RUv0-CRU.
This is the readout chain setup that most accurately reflects the setup of other WP10 development groups, which are used for development of the Ultrascale FPGA on the RU, as well as during radiation testing campaigns for the readout unit. Therefore it is necessary to replicate this setup in Bergen, in addition to the Arria 10 DK-based setups. The RUv0-CRU firmware is developed by one of the WP10 teams that also is in charge of the firmware for the main FPGA on the RUv1.
The RUv0-CRU board contains a Xilinx Kintex-7 XC7K325T FPGA, and is connected to a test computer using USB (interface is provided by a Cypress FX3 chip). The board is also built to accommodate a daughterboard containing a GBTx and GBT-SCA chip, as well as a Firefly connectors for ALPIDE sensors, but these will not be used in the CRU emulation configuration. The board will emulate the CRU by implementing the GBT-FPGA module on its main FPGA, which can send and receive data using an optical transceiver module plugged into an SFP slot on the board. The board can then be connected to a normal readout unit to control it and receive readout data, like the actual CRU. There are drawbacks to this emulation, such as significantly worse bandwidth over USB compared to PCI-express, and only one

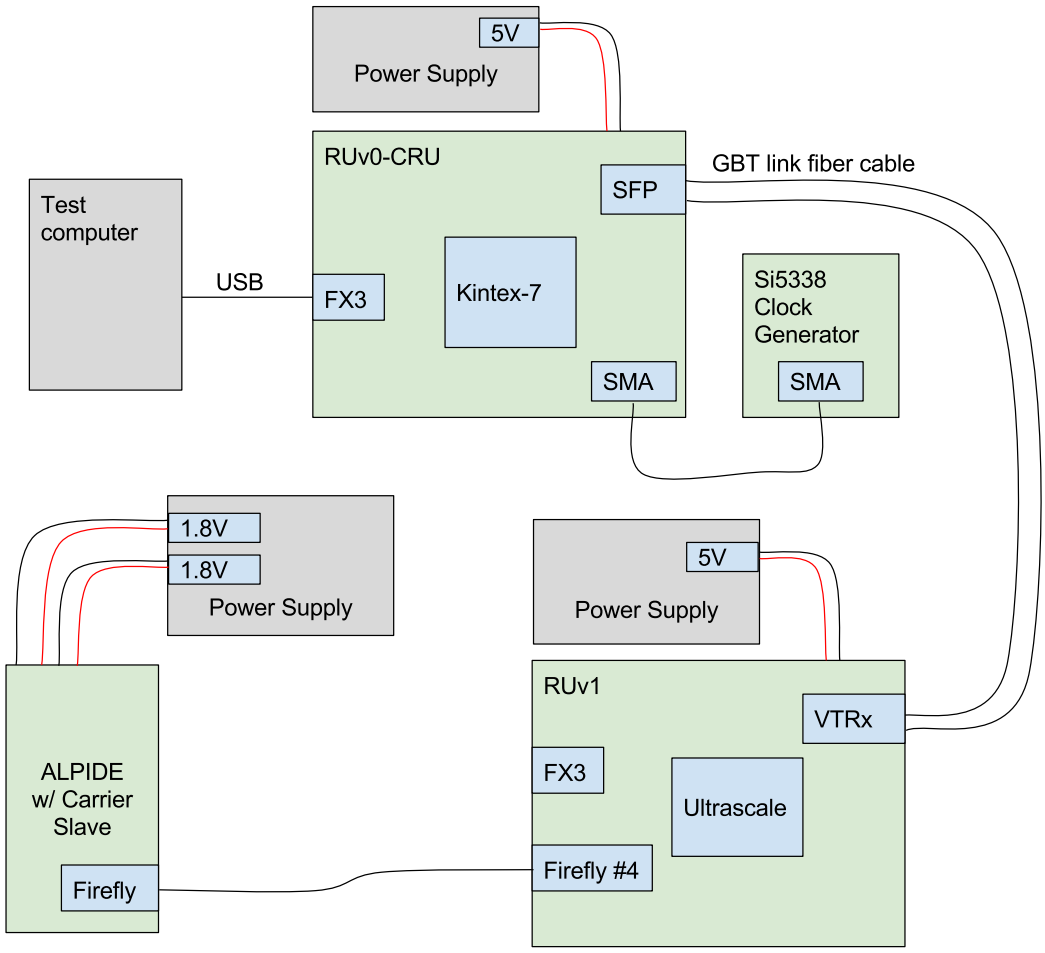

Figure 14: Block diagram of hardware setup with the RUv0-CRU, RUv1 and ALPIDE chip.

GBT channel is available, however until more CRU boards are available it is the only option.

This setup differs from the Ultrascale firmware development team's setup in two ways. First, the setup in Bergen only use one ALPIDE sensor, while the Ultrascale team uses a whole stave as will be used during operation in ALICE. Secondly, the Ultrascale team powers the setup with programmable power supplies that can be adjusted and powered on remotely. Because of this, some of their software scripts automate this process and this code will need to be removed if the scripts are to be used for the Bergen setup.

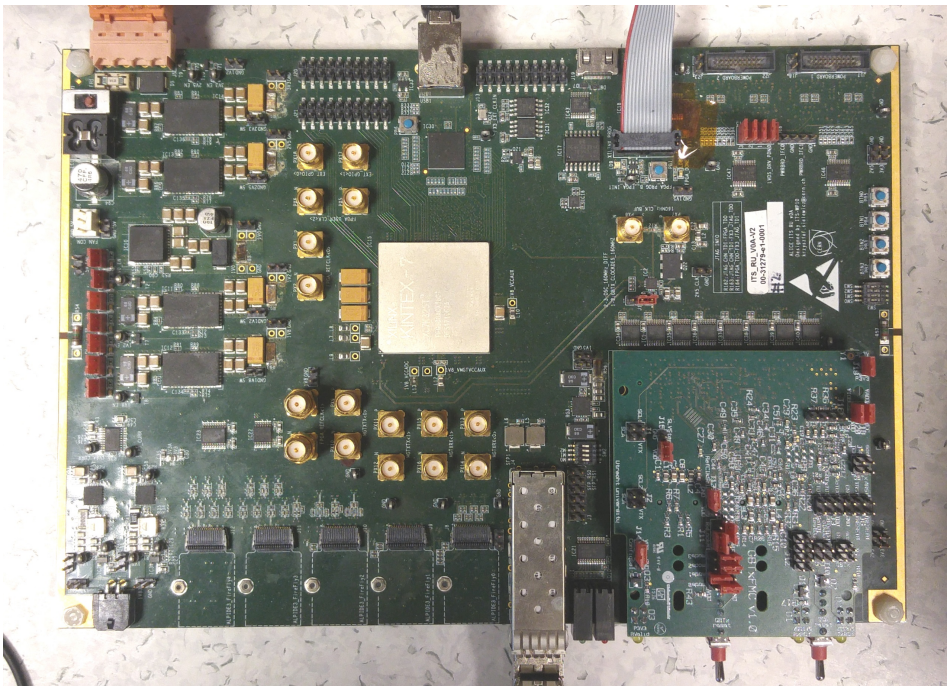

Figure 15: Version 0 of the readout unit board, used as a CRU emulator

3.2 Hardware and software setup

3.2.1 Arria 10 DK CRU and VLDB

The Arria 10 DK CRU is set up first as it is used with both the VLDB and RUv1. The CRU is a PCI-express card and must thus be mounted in a host computer. The computer chosen is a Supermicro server with an Intel Xeon E5-1650 CPU with 12 threads, 8 GB RAM, and an SSD, already present in the lab. This falls short of the CRU development team's recommended specifications, specifically the RAM capacity is somewhat low. The amount of RAM is also less than the recommended specs from Intel for synthesis for the Arria 10 FPGA used in the CRU. However, since the CRU in this lab setup will likely not be used under maximum stress with every available GBT channel, the computer hardware is judged to be acceptable. Also, the machine has several free RAM slots, so if any problems are encountered, the capacity can easily be upgraded.

The CRU host computer is set up with CentOS 7.3, as recommended by CERN. The computer needs a series of drivers and programs to communicate with the CRU. Intel Quartus Prime Standard Edition is installed to program the FPGA on the CRU, and also for building and developing the firmware. The CRU also uses onboard

clock generators on the Arria 10 SDK board. These must be configured to supply the firmware with the correct clocks. The Arria 10 GX Development Kit software package contains utilities for configuring the clock generators. It is downloaded and the clocks were successfully configured by launching the ClockController tool. All clocks are set to 240 MHz. The drivers, tools and toolchain for the CRU and ALICE development is also installed, most importantly the ReadOut Card (ROC) drivers and software, necessary for the host computer to control and communicate with the CRU. These are provided by the ALICE O2 group.
The FPGA is programmed by using the onboard USB Blaster module on the Arria 10 DK card. A USB cable is connected from the PC itself, to this port at the back of the CRU. The ALICE CRU team release pre-compiled firmware bitstreams for every CRU release, for this setup the latest such release is used rather than building the firmware manually. It is necessary to calibrate the CRU before use, and allocate memory hugepages for the ROC software. Hugepages are a dedicated portion area of memory reserved in the Linux kernel for a specific purpose[15], in this case DMA over PCI-e from the CRU. Scripts for calibration and hugepage allocation is distributed together with the firmware releases. The CRU's FPGA and clock chips are volatile, which means they lose their configuration and have to be re-programmed and re-calibrated every power cycle.
The VLDB is integrated and connected to the CRU to complete the first of the three readout chain setups.
The VLDB needs a 5V input voltage, this is supplied from a TTi QL355TP bench power supply. A mini-HDMI cable is used to connect the GBTx chip to the GBT-SCA chip using the exposed slow-control-dedicated e-ports, as illustrated in figure 12. A VTRx optical transceiver module is attached to the VLDB and it is connected to the CRU GBT channel 0 (fiber cables marked A1 and A8) from the CRU transceiver. The GBTx on the VLDB is configured using the USB-I2C dongle included with the VLDB, and the GBTx Programmer Java application[16].

3.2.2 Arria 10 DK CRU and RUv1

The CRU is kept as for the previous setup with the VLDB, but the VLDB is switched for a readout unit version 1.
The RUv1 is powered by 5V using a TTi QPX1200 power supply. A VTRx optical transceiver was mounted on the RUv1 in the first slot corresponding to GBTx chip

0, and it was connected to the CRU GBT channel 0 (fiber cables marked A1 and A8) from the CRU transceiver.
The same USB-I2C adapter as used in the VLDB tests will be used to configure the GBTx chip on the RUv1. This is connected to the pin header J12. RUv1 has an error in layout so the I2C dongle wire needs to have its two rows switched when connecting it. To make the connection, individual jumper wires are therefore used instead of the included cable. Once configured, it is possible to fuse the GBTx so that it no longer has to be configured at power up. However, it was decided against doing this to this RUv1 in the case the fused configuration would need later modifications, as it is a non-reversible action.
The auxiliary PA3 FPGA on the RUv1 must be programmed manually. A Microsemi FlashPro 3 programmer is used. A method of programming the PA3 remotely over the GBT link using the JTAG module on the GBT-SCA is also being developed.
The main FPGA on the RUv1 can be programmed by the auxiliary non-volatile PA3 FPGA if the appropriate firmware is stored in the flash memory chip. on the board. This can be done automatically on boot-up if a certain DIP switch on the board is set. Programming can also be triggered by writing the "start configuration" command to the PA3 configuration controller module control register. This is only possible if the PA3 as well as the GBTx chip has already been configured.

3.2.3 RUv0-CRU and RUv1

The Ultrascale FPGA on the RUv0a board is programmed with a Xilinx DLC10 USB programmer with the firmware for the RUv0-CRU[17].
The RUv0-CRU firmware requires an external 120MHz clock signal on the FPGA_REFCLK<1> SMA connectors. This is provided by a Silicon Labs Si5338-EVB Clock Generator board. It was configured to output a differential 120MHz signal on output 0, and the two clock signals with opposite polarity was connected to the SMA connectors on the RUv0a. The FX3 USB interface chip on the board is programmed to establish USB communication. This was done using the CyUSB software from Cypress with the slfifo_uart FX3 firmware also included in RUv0-CRU repository.
A GBT link compatible (850nm laser wavelength) transceiver was inserted into slot J4 on the RUv0-CRU. The RUv1 is set up as with the previous Arria 10-based readout chain, and connected to the transceiver on the RUv0-CRU with a fiber-optic cable.
A firefly cable is connected between the ALPIDE chip with carrier slave adapter

board, and the RUv1's fourth firefly port, the one physically closest to the VTRx transceiver. A power supply with 12V output powers the RUv1 board, which draws around 2.5A when running. A second power supply with two channels, both outputting 1.8V powers the ALPIDE. One channel powers the analog and the other powers the digital circuit.

3.3 Test descriptions

3.3.1 CRU register access

This is first and most elementary test performed. The register bus on the CRU is accessed by reading a known register such as the firmware version register, as well as writing to then reading back a writable register. This is done using the ROC software for the Arria 10 DK CRU, and with software developed by the RUv1 Ultrascale WP10 for the RUv0-CRU[18]. This test checks whether the CRU is configured correctly and can be reached from the test computer, a prerequisite for all other tests.

3.3.2 CRU data benchmark

This test measures the possible data rate between the CRU and the test computer. This is only tested on the Arria 10 DK CRU, as the data rate with the RUv0-CRU is insignificant in comparison due to its USB interface rather than PCI-express like the Arria 10 DK. The test is performed with a dedicated DMA benchmarking script included in the ROC software.

3.3.3 GBT loopback

This test checks whether the GBT link between the CRU and RU is working. The test consists of configuring the GBTx on the readout unit with a loopback configuration which makes it return all GBT data as-is to the CRU. The CRU can then output a data pattern. The incoming GBT stream is checked for validity. These tests have already been designed for the Arria 10 DK CRU and are distributed by the CRU development team[19].
When both boards are configured, the status register of the GBTx was read using the GBTx programmer application. It showed "Idle" meaning that the GBT link is running and synchronized. Now the loopback test can be performed. The loopback test

software is included in the Python script gbt.py included with the CRU firmware releases. First the GBT data stream needs to be initialized using the command parameter "init". A second parameter indicates the link numbers. We write 0-3 to include all four GBT links on the CRU. Then, the GBT link error counters are reset with the command "cntrst". Now the error counters can be displayed using the command "cntstat". With link 0 receiving valid data from the loopback, it is expected to see no errors for this link but a high amount of errors for the other links.
For the RUv0-CRU, this test is not performed.

3.3.4 SCA access

In this test the GBT-SCA chip on the readout unit is accessed. First simply reading and writing registers is attempted, then various modules such as GPIO and I²C is used. This test will determine whether the GBT-SCA chip and communication with it works as excepted.
Python scripts with functions for controlling some of the modules on the GBT-SCA is available from the developers of both the Arria 10 DK CRU and the RUv0-CRU, and the tests are based on these scripts.
First, it is attempted to establish a connection. This function sends a reset then connect packet to the GBT-SCA, and reads the response to verify success.
Next, the GPIO module of the GBT-SCA chip is tested. On the VLDB, the GPIO pins are exposed on the board to various interfaces, such as switches and LEDs. These can be used to check whether the operations worked. However, on the RU, these pins are not exposed and the test results are therefore not easily verifiable at this point. However it will later become possible to check the result using the FPGAs on the RU which is connected to some of the GBT-SCA's GPIO pins.
Before the GPIO module can be used, it must be enabled. This is done by setting a byte in a control register of the GBT-SCA. The GPIO module must be configured by settings registers, to enable or disable, and set direction (input/output) of GPIO pins. Then, commands to write or read the GPIO pin value registers can be executed.
The I²C module is also tested, as it will be used later for setting up a communication interface with the readout unit's PA3 FPGA. The I²C module is enabled in the same way as the GPIO module, by setting its control bit in the control register to high. The I²C module must also be configured with the baud rate. It is set to 1MHz (max) for this test. An Then a write transaction command is sent, with the chip ID and data as

the SCA command package payload.
For the VLDB, the chip ID was set to 0b1111000. An oscilloscope was connected to the I²C data and clock pins to monitor the result.
For the RUv1, as with the GPIO test, the I²C pins are not exposed and the result cannot easily be checked at this point. The I²C module is instead tested later using the PA3 FPGA as an I²C slave.

3.3.5 DCS interface (ALF/FRED)

In this test, a dummy FRED client is used to connect with an ALF server running on the CRU host computer to access the previously tested ROC low level functions over the network rather than locally.
An ALF server is included with the ROC software already installed on the computer. The DIM protocol also needs a DNS server running somewhere on the network. This server application is downloaded from the official DIM web page, and for testing purposes it is run on the same host as ALF and FRED application. A FRED sample client is also included with the ROC software. This program sends various commands such as communicating with the GBT-SCA chip to the server. This program is used to test the ALF/FRED functionality.

3.3.6 Readout from the ALPIDE chip

This test attempts to start a readout process and receive sensor data from the ALPIDE chip connected to the RUv1. This test is only performed on the RUv0-CRU + RUv1 setup, because the Arria 10 DK CRU lacks software for interfacing with the ALPIDE controller on the RUv1 Ultrascale FPGA. Software for the RUv0-CRU-based setup is available from the Ultrascale firmware development team. A series of scripts need to be run, which will initialize the Ultrascale on the RUv1 and RUv0-CRU and start a readout test. The script will perform the test and report the result.
Some of these scripts must be modified to work with this setup, as they are made for the Ultrascale team's setup with a full sensor stave with 8 ALPIDE chips. Some places the number of chips can be selected as a parameter to the functions, but other places it is hard coded. These lines of code are rewritten for the one chip setup. The ALPIDE chip availabe in Bergen has the chip ID 0, this is used as a parameter for the functions that need them, with the exception of setup_readout and test_readout, which takes a parameter of a list of transceiver IDs. These are the opposite of the

chip ID order, so the transceiver ID passed to these functions is 8.

3.4 Test results and discussion

3.4.1 CRU register access

Reading and writing to the register bus of both the Arria 10 DK CRU and RUv0-CRU is successful.

3.4.2 CRU data benchmark

The DMA benchmark completed successfully with no errors with the Arria 10 DK CRU. The achieved data rate with default settings was 28.44 Gbps, or 3.56 GBps. This is not ideal, in theory a speed of over 6 GB/s should be possible.
In order to improve the data rate, the parameters of the benchmark script is tweaked. First, error-checking was disabled. This improved the speed to 4.47 GBps. It was also attempted to use a different hugepage configuration. Instead of using 128*2 MiB hugepages, a single 1 GiB hugepage was allocated and used, by modifying the ROC software configuration files. A further increase in data rate was observed, at 4.67 GBps. This is still significantly lower than the theoretical result, but it is enough for development purposes.
If necessary, it is likely that the data rate can be improved by upgrading the host computer to reduce performance bottlenecks such as RAM, storage or CPU speed. It may also be further improvable by tweaking the configuration of the ROC software.

3.4.3 GBT loopback

During loopback no errors were seen on link 0 and a high amount of errors on other links. It is concluded that the GBT link is working.

3.4.4 SCA access

Reading and writing to registers on the GBT-SCA was successful on both the VLDB and RUv1.
It was attempted to set all GPIO pins connected to LEDs on the VLDB to high, and then low. It was observed that the LEDs turned on and off as expected. After setting the direction to input, it was attempted to read the GPIO values of pins connected

to switches, when the switches were enabled and then disabled. The values of these bits were read back as high and then low, as expected. It is therefore concluded that the communication with the GPIO module works as expected.
When the I²C write transaction was executed, the expected data could be observed on the oscilloscope, namely the chip ID being read out. However, due to lack of an I²C slave to assert an ACK signal, full functionality of transactions could not be tested at this time. Nevertheless, control communication with the I²C module of the GBT-SCA is concluded to work.
Communication with the I²C module on the RUv1 was also successful. For this board, the full I²C functionality was thoroughly tested later in preparation for the work in chapter 4.

3.4.5 DCS interface (ALF/FRED)

The DNS server and ALF server is started, and the ALF client is run. Activity corresponding to the commands executed by the ALF client is observed on the ALF server console, so the communication between client and server is working. The behavior and returned data from the server's operations is also as expected. The ALF/FRED interface is therefore concluded to be working.

3.4.6 Readout from the ALPIDE chip

Communication with the ALPIDE chip was working, however unexpected behavior during readout was observed. The event counter of the ALPIDE controller reports a value in the order of thousands even though only one trigger is sent. This may be caused by an incorrect configuration in the ALPIDE chip, such as triggering continuously rather than only when receiving a trigger signal.
Due to preparations for irradiation test campaigns, it was prioritized to work on essential control component tests rather than readout from the ALPIDE. Therefore, this problem was not resolved.

3.4.7 Test result summary

All performed tests on the readout chain variations were successful, with the exception of readout from the ALPIDE. This is not critical, as the ALPIDE is not needed for the ongoing development in Bergen. Results are summarized in table 1.

	A10 CRU + VLDB	A10 CRU + RUv1	RUv0-CRU + RUv1
CRU register access	Passed	Passed	Passed
CRU data benchmark	Passed, 4.67 GBps	Passed, 4.67 GBps	N/A
GBT loopback	Passed	Passed	N/A
SCA access	Passed	Passed	Passed
SCA GPIO module	Passed	N/A	N/A
SCA I^2C module	Passed (no slave)	Passed	Passed
ALF/FRED	Passed	Passed	N/A
ALPIDE readout	N/A	N/A	Partial failure

Table 1: ITS readout chain test results

4 Communication with the RU Auxillary FPGA

4.1 Background

At the early stage of the PA3 development process in Bergen, communication with the PA3 Wishbone bus was only accomplished with an USB to UART cable directly connected to the PA3 from a computer. This is only a temporary solution, as an electrical UART interface will not be available on the readout unit during operation in ALICE. It is therefore necessary to develop an interface for communicating with the PA3 Wishbone bus over the GBT link via the CRU.

4.2 I²C interface

The PA3 communication interface is implemented with an I²C connection between the PA3 and the GBT-SCA chip. I²C is a protocol for inter-chip communication that uses two bidirectional lines: SCL (clock) and SCA (data)[20]. Simplified, an I²C transaction consists of a 7-bit slave address (chip ID) followed by a R/W bit and an ACK bit response, then data grouped into bytes, each also followed by an ACK bit.
On the readout unit, the PA3 is connected to two of the GBT-SCA's I²C master modules, channel 0 and 5. Both channels will not be needed, as the PA3 will only have one bus, and it will not be possible to increase the data rate by using both channels as the GBT-SCA chip can not process commands in parallel. I²C master channel 0 on the GBT-SCA is chosen for the interface. The full path of accessing the PA3 Wishbone bus from the Arria 10 CRU host computer can be seen in figure 18.
Initially, on PA3 firmware version A112 and earlier, the PA3 Wishbone bus had a data width of 32 bits and address width of 16 bit. The 7-bit I²C slave address of the PA3 was set to 0b0011010. This meant the I²C protocol worked as follows:
To write to the Wishbone bus, a 6-byte I²C write is transmitted. The first two bytes of data contain the 16-bit Wishbone address. The last four bytes contain the 32-bit data value.
To read from the Wishbone bus, a 2-byte I²C write is transmitted, containing the Wishbone address. Then, a 4-byte I²C read command is transmitted, and the returned 4 bytes is the read data value.
However, it was eventually decided to change the Wishbone bus to a data width of 8 bits and an address width of 7 bits as of version A200. This will in theory lead to a

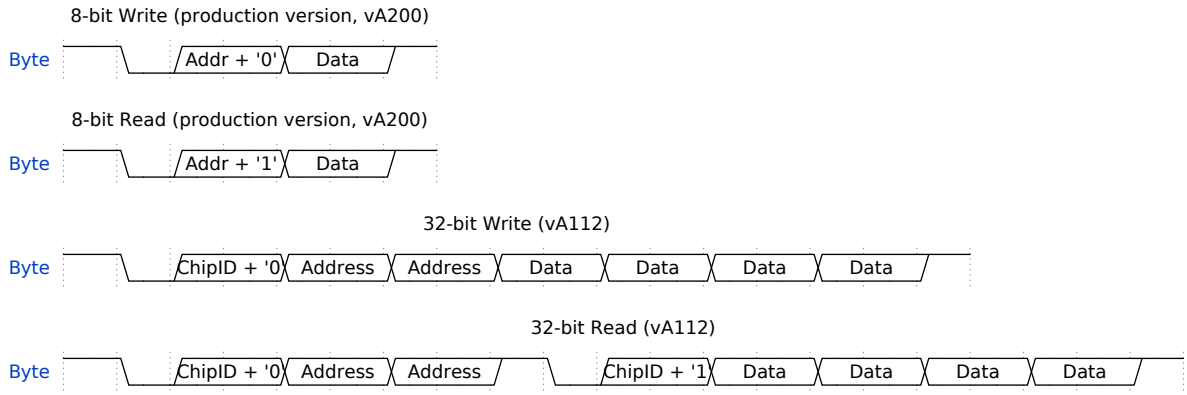

Figure 16: Comparison between the I²C transactions for the new 8-bit and the old 32-bit PA3 bus

significant increase in speed, because both read and write transactions could use the GBT-SCA's single byte transaction command rather than multi-byte transactions. To initiate a multi-byte transaction in the GBT-SCA I²C master, multiple HDLC commands need to be issued to the GBT-SCA: Setting the control register to update the number of bytes in the transaction, setting/getting the data registers, and sending the transaction, at a minimum. For Wishbone read operations, this sequence even needs to be performed twice as a read operation involves both a multi-byte write and a multi-byte read I²C transaction. In contrast, to initiate a single byte transaction only one command is needed, for either read and write operations. Additionally, the I²C transactions themselves are shorter, as they only contain two bytes, as can be seen in figure 16.

To be able to use such single byte transactions, the I²C slave address field needs to be used as the Wishbone address, hence the 7 bit address width. This is not technically correct use of the I²C protocol, because the register bus address is not a slave address which this 7-bit field is meant for. Some values for this field is also reserved, such as 0b11110XX which indicates that the I²C transaction use a 10-bit addressing scheme instead. However, in this case it is judged to be acceptable, due to the significant advantages and the fact that the slave address is not needed as there is only one slave connected to the I²C master. All PA3 Wishbone registers can fit into a 7-bit address space, even if excluding all reserved addresses, although this is not strictly necessary because the I²C module on the PA3 is not designed to differentiate these special addresses from others.

Since communication with the GBT-SCA chip on the RUv1 board has already been

successfully tested with both the Arria 10 CRU- and RUv0-CRU-based readout chains in chapter 3, implementing the software for communicating with the PA3 via I²C is relatively straight-forward.
On the PA3 FPGA firmware, the I²C controller module is implemented by Arild Velure[1]. This chapter will present the software for the FLP node (CRU host computer) to perform the communication.

4.3 Software for the Arria 10 CRU

4.3.1 Design

During development, several software implementations were written, for the early 32-bit Wishbone bus and for the final 8-bit bus, and with different low level libraries including the O² ReadoutCard software[21] in C++.
The final software presented here is based on the CERN Git repository CRU-ITS[22], an early version of a control interface to the Arria 10 CRU modules that is needed by the ITS WP10 team such as SCA and SWT, written in Python. Functions for controlling the SCA I²C master modules were added to this repository in the SCA interface class including initial configuration, and reading and writing single- and multi-byte transactions.
An abstraction class for the PA3 was created. This instantiates the SCA interface class and implements read and write functions that calls the SCA single-byte with only address (and data for write) parameters for an easy to use high-level interface for accessing the PA3 Wishbone bus. Some helper functions such as register dumping for debugging and monitoring were also implemented. Pseudocode for the class structure with its essential function can be seen in figure 17.

4.3.2 Testing

The software was tested on the PA3 firmware version A200 with the 8-bit Wishbone bus, and the Arria 10 DK CRU with firmware v2018-01-16. It was able to read and write the Wishbone bus without any issues.
A benchmark function was created to measure the speed of the bus access. The function repeated a read or write operation one million times, and used the difference in time between the start and the end divided by a million to get an estimate of the

[1] arild.velure@cern.ch

```
class PA3():
        init():
                sca.init()
                sca.i2cEn(0)
        read(addr):
                return (sca.i2cRd_7b(0, addr) >> 16) & 0xff
        write(addr, data):
                sca.i2cWr_7b(0, addr, data)
```

Figure 17: Pseudocode for implemented functions for accessing the RUv1 flash memory using the Arria 10 CRU

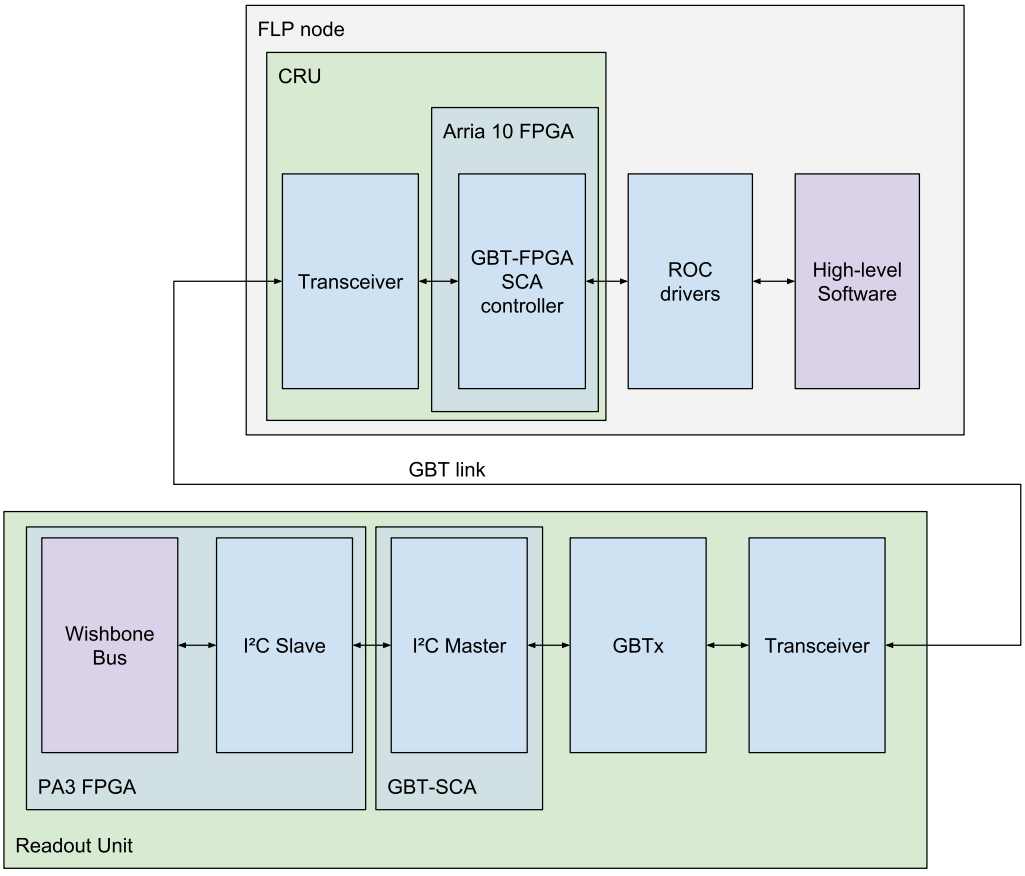

Figure 18: Communication chain for the PA3 I²C interface

time spent for each transaction. The result was an access time of 39 µs for both read and write transactions. No errors in read data were detected throughout the million transactions, suggesting a reliable operation.

4.3.3 C++ ALF/FRED implementation

A simple library with functions to read and write from the PA3 bus was also implemented in C++ based on the demonstration FRED client mentioned during chapter 3.3.5. This could be used to interface the PA3 with DCS. The functions are based on the style used for the UART library used to communicate with the PA3 during early development. Both the read and write function takes two arguments: An uint8_t for address, and for the write function an uint8_t for data. For the read function, the second argument is a pointer to an uint8_t which is filled with the returned data. The code for this library can be found on UoB's git server[2].

4.4 Software for the RUv0-CRU

4.4.1 Design

The WP10 team in charge of the RUv1 Ultrascale FPGA has written basic test software for the RUv0-CRU including SCA communication code. This test code is located in the CERN Gitlab repository RUv1_Test[18]. The software for I²C communication is based on these libraries, however during development there was many changes to both the I²C protocol such as the aforementioned change from 32-bit to 8-bit PA3 Wishbone bus, as well as upgrades of the SCA controller module on the RUv0-CRU firmware, therefore the functions for I²C communication had to be rewritten and updated several times. The final implementation consists of PA3 register read and write functions as well as helper functions such as register dumping for debugging and monitoring implemented in an existing class for SCA-related functionality, written in Python. Pseudocode for the PA3 related functions in this class can be seen in figure 19.
These functions and other PA3 related variables such as register address constants have also been gathered in a dedicated abstraction class for the PA3, similar to the Arria 10 DK software implementation discussed above. This class depends on the SCA interface class containing the actual PA3 Wishbone read and write functions,

[2]https://gitlab.uib.no/gmi001/alf2bus

```
class Sca():
        initialize():
                [..]
                self.enable_channel([..])
        read_pa3_register(addr):
                return (sca.i2cRd_7b(0, addr) >> 16) & 0xff
        write_pa3_register(addr, data):
                sca.i2cWr_7b(0, addr, data)
```

Figure 19: Pseudocode for implemented functions for accessing the RUv1 flash memory using the RUv0-CRU

which it instantiates at initialization. The SCA chip's I²C module is also enabled during this class' instantiation.

4.4.2 Testing

The software was tested on the PA3 firmware version A200 with the 8-bit Wishbone bus, and the RUv0-CRU with firmware v2018-01-16. It was able to read and write the Wishbone bus successfully.
A benchmark function was created to measure the speed of the bus access, as with the Arria 10 version of the software. The results were an access time of 46 000 µs for both read and write transactions. This is far slower than the results with the Arria 10 DK CRU, likely because of bottlenecks in the USB to RUv0-CRU interface. But this is not seen as a prioritized problem, as the RUv0-CRU will not be used in production.
No errors in read data were detected throughout the million transactions, suggesting a reliable operation.

4.5 Discussion

4.5.1 Results

In this chapter access to the readout unit's PA3 FPGA's Wishbone bus using the GBT-SCA's I²C master module has been presented. Demonstration software running on the test computer interfacing with both the Arria 10 DK CRU emulator and the RUv0-CRU has been developed and tested successfully.

	UART	I²C	
		Arria 10 DK CRU	RUv0-CRU
8-bit PA3 bus	?	39 µs	46 000 µs
32-bit PA3 bus	19 µs	180 µs (R) / 130 µs (W)	?

Table 2: Comparison of PA3 bus access time

4.5.2 Performance

The I²C communication has a large amount of overhead on most layers of communication, such as software to CRU, HDLC protocol from the CRU to the GBT-SCA, and configuration and commands the GBT-SCA before it sends an I²C transaction. With the RUv0 as a CRU emulator, the chain is especially slow as it appears to be bottle-necked by the USB communication.
The choice of adapting single-byte transactions using the I²C slave address as Wishbone address greatly increased the performance of the bus access. As mentioned, a bus access transaction takes approximately 39 µs with an Arria 10 DK CRU. For reference, this should in theory result in uploading a 25 MB bitfile to the flash memory taking approximately 16 minutes if transferred by I²C.
The benchmark test was repeated for the previous scheme of multi-byte transactions with 32-bit data and 16-bit addresses. The result was 180 µs for read operations and 130 µs for write operations. This means that the single-byte transacting scheme increased the speed by approximately 4.7x and 3.4x for read and write operations respectively. This is as mentioned mainly due to the reduced number of commands needed to send to the GBT-SCA chip to initiate single-byte transactions compared to multi-byte transactions.
The performance results is summarized in table 2.

4.5.3 DCS implementation

It will eventually be necessary to integrate the software using the ALICE Detector Control System's software libraries and frontend for use in operation. A proof of concept with PA3 bus read and write functions written in C++ has been written. This library uses the O² ReadoutCard library[21] for low level functions for communicating with the SCA chip through the Arria 10 CRU. DCS will likely only need access to a few select registers such as status registers for modules critical to the PA3's operation. This might include the configuration controller, to check whether

or not scrubbing of the Ultrascale is ongoing without errors. If the error is easily fixable, such as by resetting the PA3, this should also be exposed to the DCS interface so that the DCS shift leader can perform this task without having to wait for experts to arrive. However, some registers should not be exposed to the DCS control panel like this. An example would be the clock configuration register, which could cause the PA3 to lose clock depending on configuration of the RUv1 in terms of available clock sources. In the case of error that can not be resolved easily, a detector expert could access the rest of the registers through specialized low level tools, such as the Python software discussed in this chapter.

5 High-speed interface between Ultrascale and PA3 flash controller

5.1 Background

Due to the volatile nature of SRAM-based FPGAs, firmware for the Ultrascale Ultrascale on the Readout Unit need to be stored in a flash memory chip on the board. The PA3 auxiliary FPGA is responsible for programming the Ultrascale FPGA using the content of this flash, and also for writing to the flash. It is possible to do this over Wishbone access such as UART or I²C, these methods are, however, slower than desired for such a task, with the Arria 10 CRU I²C implementation using 15 minutes or more to transfer one 25 MB firmware bitfile. Therefore, a new higher-speed interface for writing to the flash memory of the readout unit should be designed and implemented. This interface will be discussed in this chapter.

5.2 Description of the solution

It is decided to use a FIFO on the Ultrascale FPGA on the readout unit, which can be directly read by the PA3. Writing to the Ultrascale FIFO is done through its Wishbone bus, which has significantly faster access speed and bandwidth than the GBT-SCA's I²C module, because it uses the main 80-bit data payload field of the GBT protocol in a so-called Single Word Transaction (SWT), rather than the EC field which only occupies 2 bits for each GBT frame, as described in section 2.4. In addition, communication with the GBT-SCA has extra overhead as the path between CRU and GBT-SCA is more complex than the path between the CRU and the Ultrascale FIFO. The team in charge of the Ultrascale firmware designed a custom protocol using such SWTs to access the Wishbone bus on the Readout Unit from the Common Readout Unit. Simplified, this protocol writes the address and value of the desired Wishbone register in the GBT frame data field. These values are doubled to provide extra protection against SEUs.

A new write controller on the PA3 that takes data directly from this FIFO and loads it into the flash interface buffer is needed. This write controller module would have to act as a bridge between incoming data to the flash from the Ultrascale FIFO, and the flash interface. To initiate a page write to the flash, some control signals must first be written to the flash interface module, namely the address of the page to the

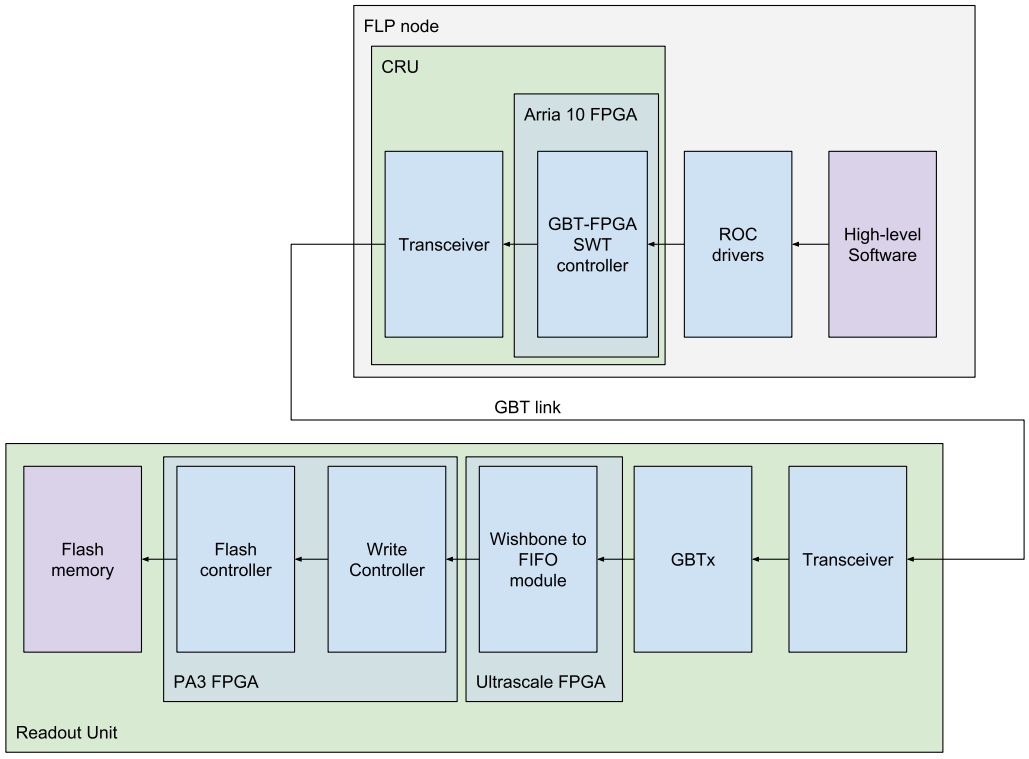

Figure 20: Chain for uploading data to the flash using the Ultrascale FIFO interface

written to, and the size of the page. As of now, the page size signal is set to a constant 4096 bytes, but this would need to be dynamic in the future to for example be able to read and write the spare section of flash pages. The spare section is an extra section of flash pages which can, for example, be used to hold ECC information. When these configuration signals are set, a write page command can be send to the flash interface by setting the command signal to the page write command value and pulsing an execute signal. The flash interface will then read from the write controller FIFO until it had read a full page, which it will then write to the flash memory.
During a redesign of the PA3 firmware it has been decided that the existing 4096 byte page buffer on the Wishbone bus, as well as a page buffer in the flash interface module, should be removed. Instead, the flash interface will read data from a single FIFO queue. It was therefore decided to combine the writing from the bus and the Ultrascale FIFO into a single module called the write controller, which will control the flash interface and expose it to a FIFO containing the data to be written to the flash. This design will reduce the overall complexity and footprint of the PA3 firmware, especially the amount of memory cells needed. This path for uploading

data is illustrated in figure 20.

5.3 Design for buffered flash interface

Prior to the specifications of the PA3 firmware redesign with the new memory-less flash interface was ready, work began on a temporary solution to interfacing the Ultrascale FIFO with the flash on the PA3 on the firmware version A112. A few designs were considered, such as a module that only copies data from the Ultrascale FIFO into the page buffer with a command, and lets all other control communication, such as initiating flash page writes and block erasing, happen over the I²C with the existing flash interface. However, it was observed that this method was quite slow, because of bottlenecks in the I²C communication. With this method, several I²C transactions would have to occur for every page written to the flash, such as setting the page address and triggering the commands for data transfer from the FIFO and writing to the flash, and this appeared to cause relatively large delays between each page write, especially if the RUv0-CRU was used as a CRU emulator.
A new module is designed to only take one set of Wishbone commands and configurations before each task of writing to the flash. The starting page address must be set, then the module would read data from the Ultrascale FIFO and send it to the flash page buffer in order. A byte counter would keep track of the page buffer position, and when the whole page buffer was filled, the module automatically initiated a flash page write command to the flash interface, and incremented its page address register, and started over. After the desired number of pages had been written, a stop command would send the module back to its idle state.
This high level of automation has some disadvantages, such as being more prone to bugs. For example, if for some reason a byte was duplicated or lost when sent to the Ultrascale FIFO, the byte counter would become off by one and the flash page write commands would not happen at the correct point. It also introduces more corner cases during operation, which will need to be tested thoroughly. However, this style of module was still deemed more favorable due to the higher speed and ease of interfacing with it on a software level.
This module was finished and tested and with the RUv0-CRU it was able to successfully write an RUv1 bitfile of 24.1 MB to the flash in about 5 minutes and 30 seconds, of which 21 seconds were used to erase blocks, which was done over the flash interface and not through this module. This is faster than the I²C and UART interface,

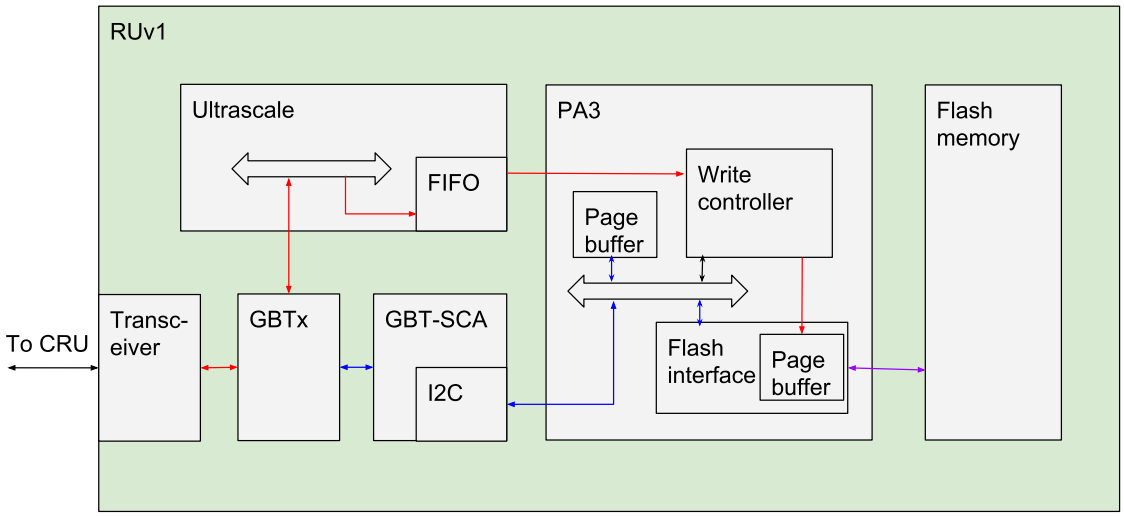

Figure 21: Block diagram of the RUv1 with the high-speed (red) and low-speed (blue) paths for writing to the flash, with the old PA3 firmware using page buffers

but not as fast as desirable.
A block diagram of this write controller and the data path through the RUv1 can be seen in figure 21.

5.4 Write controller design for new memory-less flash interface

When the temporary solution for the old firmware was finished, work began on the write controller module for the redesigned PA3 firmware with the memory-less flash interface, version A200. The module is meant to be an intermediary to abstract away the multiple ways of writing to the flash, to a single interface that the flash controller can read from. The flash interface requires a FIFO that it can read data from, as well as a control interface that takes a flash address, commands and outputs status signals.
Writing from the Wishbone bus to the flash requires buffering the data in a FIFO that the flash controller can read from. At first, it was planned to place this FIFO in front of the write controller, and then use the write controller as a multiplexer to forward either the bus buffer FIFO or the Ultrascale FIFO to the flash controller. However, it was discovered multiple problems with this approach. For one, the Ultrascale FIFO's output is delayed with one clock cycle. This means that either the bus FIFO needs

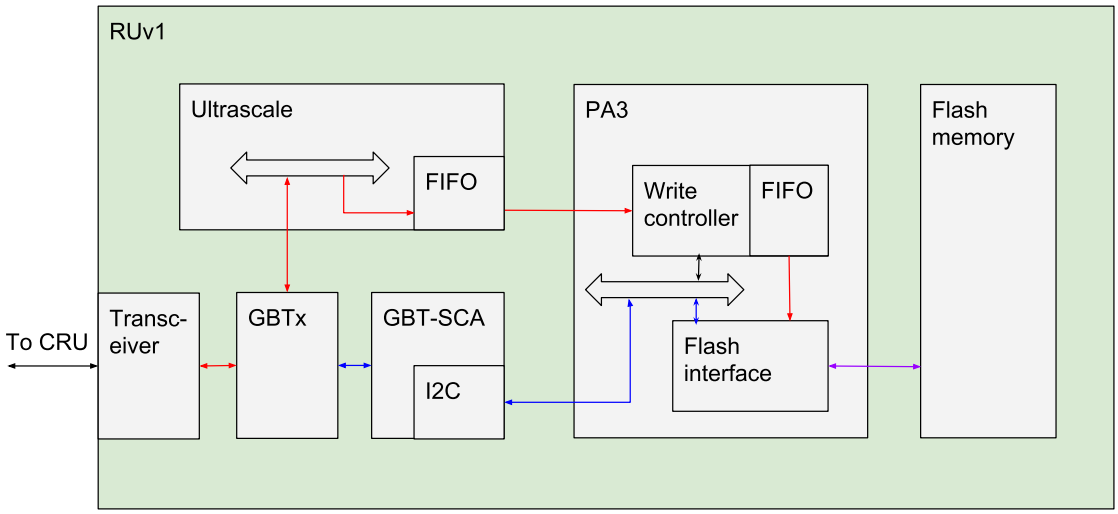

Figure 22: Block diagram of the RUv1 with the high-speed (red) and low-speed (blue) paths for writing to the flash, with the new PA3 firmware using a memory-less flash interface module

to also be delayed to match the Ultrascale FIFO, and then the flash interface would need to handle this delay.

Instead, it was decided to use a buffer flash FIFO behind the write controller module, and connect this single FIFO to the flash controller. This buffer FIFO would have no output delay so it would be very simple to interface with for the flash controller. The write controller module would write to this buffer FIFO, either directly from the Wishbone bus using a register, or copy data from the Ultrascale FIFO into the buffer FIFO, accounting internally for the clock cycle delay by using a state machine.

Otherwise, the module would work mostly like the previous design. A start command would initiate the write operation, and the module would start writing to the buffer FIFO as data comes in, automatically keeping track of the page number and initiating page writes to the flash controller. This would continue until a stop command is received.

A block diagram of this write controller and the data path through the RUv1 can be seen in figure 22.

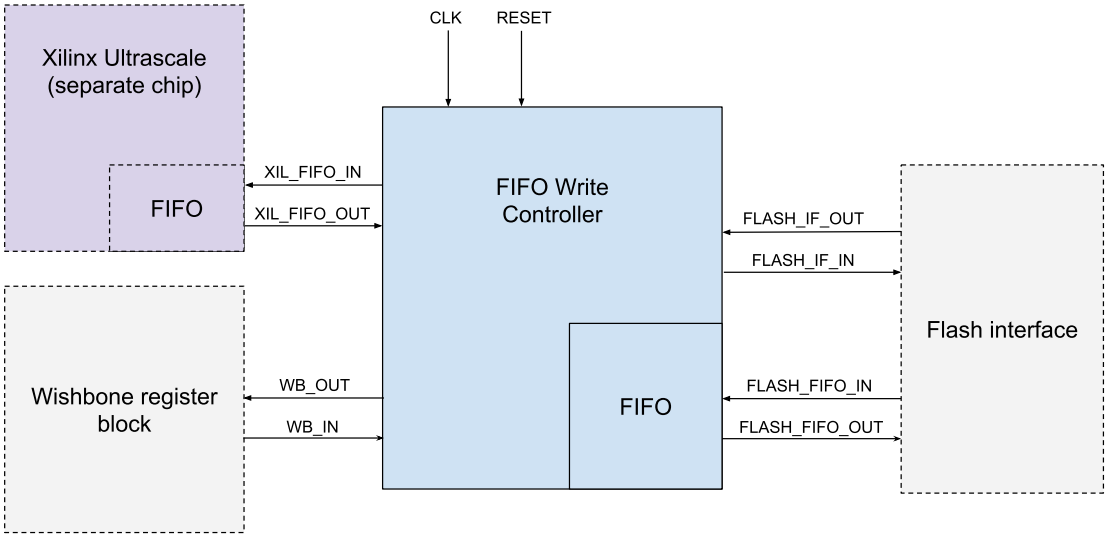

Figure 23: Detailed block diagram of the write controller and connections to interfaced modules.

5.4.1 Bus interface

The module is controlled and monitored through various signals accessible through the Wishbone bus on the PA3.
The input control signals are listed and described in table 3. In the VHDL module and block diagram (figure 23), these signals are grouped under the name WB_IN.

Signal name	Width in bits	Description
command	7	Command signal (see table 4 for commands).
execute	1	Executes the command in the command signal if pulsed.
addr	24	Flash page address from which to start writing.
fifo_data	8	Data to write to the flash in bus write mode.
fifo_we	1	Write enable signal for the data in fifo_data.

Table 3: List of input signals for the FIFO Write Controller module

Command name	Description	Value
WRITE_XIL	Starts writing to the flash with the Ultrascale Ultrascale FIFO as data input.	"0000001"
WRITE_BUS	Starts writing to the flash with the PA3 Wishbone bus (fifo_data / fifo_we) as data input.	"0000010"
STOP	Stops writing to the flash and return to idle mode.	"0000100"
CLEAR_ERRORS	Clears error signals (see table 5).	"0001000"

Table 4: List of commands for the FIFO Write Controller module

The output status signals are listed and described in table 5. In the VHDL module and block diagram (figure 23), these signals are grouped under the name WB_OUT. They provide all necessary information for operation, monitoring and simple debugging. The first bit (idle) is usable as a ready signal, as the module can accept new commands (except STOP) when and only when the main state machine is in the idle state. There are only 8 bits of output signals, so they can all fit inside a single WB register if necessary.

Signal name	Width in bits	Description
idle	1	High if the module is idle and ready to receive a command.
activeInput	1	This signal is high if the input is the Ultrascale FIFO, low if the input is the WB bus (or if the module is idle).
flashFifoEmpty	1	High if the flash FIFO is empty.
flashFifoFull	1	High if the flash FIFO is full.
xilFifoEmpty	1	High if the Ultrascale FIFO is empty.
stopping	1	High if the stop signal is set and the module will cancel operation shortly.
errorWrongCommand	1	High if the module received an invalid command.
errorFifoWrite	1	High if data is written to the FIFO interface on the WB bus when the module cannot accept it.

Table 5: Overview of the status signal outputs for the FIFO Write Controller module

5.4.2 Operation

The design is based on three processes. The main process controls the primary states and command interface of the module using a finite state machine (FSM). The state machine starts in the idle state. Here it waits for a command from the Wishbone control register. Once a command has been received, it latches the start page address signal, and proceeds to the next state called AwaitFirstData. This state polls two signals. One is the stop signal. If this is high, the module returns to the idle state. The other is the flash FIFO empty signal. If this signal is low, it means that the user has started writing data to the FIFO, and therefore a page write command is sent to the flash interface once it is ready. The flash interface will then start to read out the data from the flash FIFO. The next state waits until the whole page has been read by the flash interface and the page write operation is complete by polling for a trx_done signal from the flash interface. It then returns to the AwaitFirstData state. This behavior is represented as a flow diagram in figure 24.

The next processes handles data input from the Ultrascale FIFO. The Ultrascale FIFO readout process polls for the FIFO empty signal from the Ultrascale FIFO and sets the read enable signal when there is data in the FIFO. It then waits for a clock cycle until the data is available on the data signal, and sets the flash FIFO write enable signal while forwarding the data from the Ultrascale. This means that the maximum data rate when reading from the Ultrascale FIFO is 1 byte every 3 clock cycles. This could be made faster in several ways, such as dropping the wait cycle after reading data once and immediately read the next byte. The disadvantage by this method is that the empty signal will be one cycle delayed, and thus it is possible to attempt reading a value from the Ultrascale FIFO when it is in fact empty. Another method is to redesign the Ultrascale FIFO and its link to the PA3 to remove the extra clock cycle of delay. However such optimization is deemed unnecessary because the link is already fast enough. 1 byte every 3 clock cycles at 40 MHz will mean a whole 24 MB bitfile can be written in $24MB/(40MHz/3) = 1.8$ s. The actual speed is bottlenecked elsewhere anyway, and is therefore significantly slower. This is tested and confirmed in the next section.

Fetching data from the Wishbone bus is more straight forward. No process is needed, only forwarding of the write enable and data signals to the integrated FIFO. However the write enable and data from the Wishbone bus and Ultrascale is multiplexed with the activeInput signal to make sure only one source is able to write to the FIFO

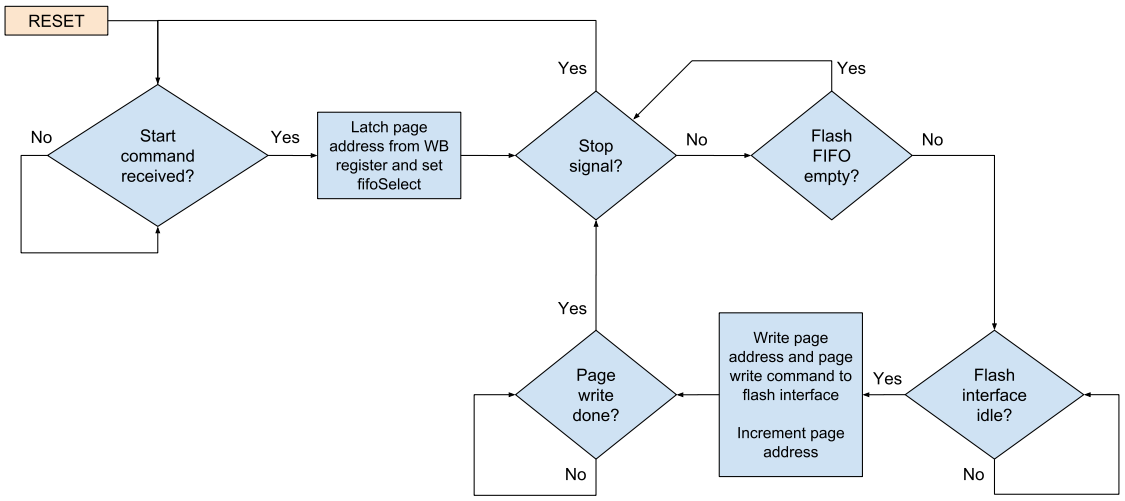

Figure 24: Flow diagram of the FIFO Write Controller VHDL module for the RU Auxillary FPGA

at a time.

The last process that handles the stop signal. This process simply watches for the stop command, and sets the stop signal high if it is triggered. It also clears the stop signal when the state machine is idle, or during resets.

The reason for separating the behavior logic into separate processes instead of using one FSM for everything was to reduce complexity. The stop command can be accepted during any state, so by separating it into its own process, the main FSM does not need to check for this command in every one of its state cases. Similarly, to allow the WB bus or Ultrascale FIFO to continuously write data to the flash FIFO, even during page write operations by the flash interface, readout logic would have to be implemented in every state in the main FSM, and it would also have to contain two separate branches of states due to the difference in readout for the WB bus and Ultrascale FIFO data sources.

In the main FSM another method for keeping track of page write cycles, rather than polling for the flash interface trx_done signal, was considered. A counter could increment every time data was written to the flash FIFO and execute the flash interface page write command for a new page when the counter reached the page size. But it was decided against this due to increased complexity of the flash writer module. There is limited available space in the PA3, and additional redundant logic is unde-

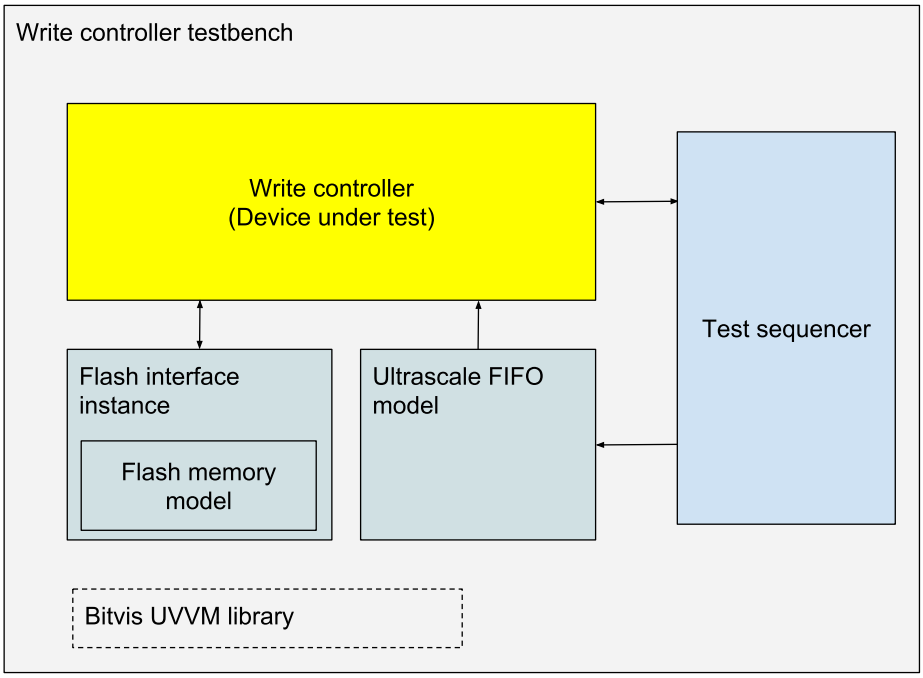

Figure 25: Block diagram of the testbench of the PA3 write controller module

sirable.

5.5 Simulation

Before testing on hardware, the module was simulated on logic level using Modelsim. A testbench VHDL entity is created, and various instances and dummy models are created, such as the write controller itself, the flash module, and a simple Ultrascale FIFO model. The testbench uses the Bitvis Universal VHDL Verification Methodology (UVVM) library for utility and logging functions. A block diagram of the testbench can be seen in figure 25.
Five tests were implemented in the testbench. The first checks all outputs for default values after a reset. Two tests checks writing one page and multiple pages respectively using the Ultrascale FIFO interface. The last two tests checks writing one page and multiple pages respectively using the Wishbone interface. All tests completed with no errors.

5.6 Software

In chapter 4.3, a Python class for PA3-related functions for the Arria 10 CRU was introduced. This class is extended with functions for uploading data to the flash using the write controller module. Two main functions are implemented, one for uploading data through the Ultrascale FIFO, and one through the Wishbone bus over I²C in case the Ultrascale FIFO is not available, such as when the Ultrascale is not configured with its firmware. Both functions will take a starting block number and a bitfile path as parameters, with two optional parameters of a block number and path of a scrubbing bitfile as well. The functions work by first looping through the amount of blocks required by the data and issuing commands to the flash interface to erase these blocks. Then, the starting block number is written and a write operation is started, and the bitfile data is fed to the write controller through the appropriate route until it is finished. This is repeated for the scrubbing file if its parameters are set. Finally, the configuration page containing the address and size of the bitfiles are assembled and written to page 0 of the flash using the same method as the bitfiles themselves.
In addition, two simpler functions are implemented to write a single page. Finally, a function for reading a page is also implemented. This uses the I²C Wishbone access since the Ultrascale FIFO data path is unidirectional. The write controller is also not used. Instead a read page command is issued to the flash interface directly, which starts writing data to a read FIFO which can be accessed on the Wishbone bus byte by byte.
Since the Ultrascale FIFO needs an SWT interface to receive data, an SWT software class already present in the CRU-ITS Gitlab repository[22] is instantiated and used in the PA3 class.
In summary, the top level flash functions are implemented in the PA3 class can be seen in figure 26.

5.7 Testing

The module for PA3 firmware version A112 with the buffered flash interface was fully tested by writing bitfiles to the flash several times, and verifying the flash content afterwards. The PA3 configuration controller was also able to successfully program the Ultrascale with the uploaded bitfile. The speed of uploading a 24.1 MB

```
class PA3():
    [...]
    flashWriteBitfile(file, blockAddr, [scrubFile, scrubBlockAddr])
    flashWriteBitfileI2C(file, blockAddr, [scrubFile, scrubBlockAddr])
    flashWritePage(addr, data)
    flashWritePageI2C(addr, data)
    flashReadPage(addr)
```

Figure 26: Implemented functions for accessing the RUv1 flash memory using the Arria 10 CRU

bitfile is approximately 33 seconds, significantly faster than other methods.
The module for the new PA3 firmware version A200 with the memory-less flash interface module was also tested successfully. The testing was done by using the software presented in section 5.6 with the Arria 10 DK CRU to upload a bitfile to the flash using the high-speed data path via the Ultrascale FIFO, and then triggering the PA3 to configure the Ultrascale with the newly uploaded bitfile. The Ultrascale was successfully configured using this method, suggesting that the data uploaded to the flash was error-free. A few pages from the flash was also read back and inspected manually for mismatches, but none were detected. The speed of the data upload was measured, and it is approximately the same as with firmware version A112, as expected.
This test was repeated for the data path via Wishbone access over I²C rather than the Ultrascale FIFO, and this too resulted in a successful configuration of the Ultrascale. Uploading the bitfile this way took about 15 minutes, which is faster than with the PA3 version A112, as expected.

5.8 Discussion

5.8.1 Result

An interface for writing to the flash memory on the PA3 has been implemented and tested successfully. The interface uses a FIFO interface on the Ultrascale FPGA which can be written to via the Wishbone bus, and read to from a write controller module on the PA3 FPGA.

	UART	I²C		Ultrascale FIFO	
		RUv0-CRU	Arria 10	RUv0-CRU	Arria 10
PA3 vA112 (32-bit)	8 min	Hours	50 min	5-6 min	33 s
PA3 vA200 (8-bit)	N/A	Hours	15 min	5-6 min	33 s

Table 6: Comparison of approximate time needed to upload an Ultrascale bitfile (24.1 MB) to the flash memory on the RUv1

5.8.2 Performance

The speed of the interface depends on the specific CRU implementation used. With the PCI-e Arria 10 board it takes 33±0.5 seconds to write an Ultrascale bitfile of 24.1MB size to the flash memory. This is significantly better than the existing interfaces using the I²C master on the GBT-SCA chip, or the direct UART interface to the PA3. With the RUv0-CRU connected to the host computer via USB2, it takes just over 5 minutes to write the same bitfile, in addition to 21 seconds of erasing the flash. This is still faster than using the I²C interface, but comparable to the debug UART interface. Curiously, with USB3 connectivity, the uploading takes longer than USB2, at a little over 6 minutes. This is despite USB3 having a higher bandwidth and packet frequency. It may be possible to optimize the USB communication module in the RUv0-CRU or the FX3 chip to reduce these numbers, however this is not regarded as a priority due to the fact that the RUv0-CRU will not be used as the CRU in operation in ALICE.

Even with the Arria 10 DK CRU, the performance bottleneck is not in the PA3 firmware and the write controller module. This was determined by inspecting the empty signal of the FIFO queue in the write controller using an oscilloscope. It is empty the majority of the time. The empty signal only goes low (data is written to the FIFO) for two clock cycles approximately every 12 clock cycles. In other words the data is immediately read by the flash interface when available in the FIFO. The write controller read from the FIFO on the Ultrascale once every three clock cycles when data is available. We can therefore conclude that the bottleneck lies before the FIFO on the Ultrascale.

The PCI-e, GBT and Ultrascale-to-PA3 FIFO links' raw bandwidth is significantly higher than the demonstrated speed. This likely means that the bottleneck lies in either the SWT controller in the CRU firmware, the SWT to Wishbone controller in the Ultrascale firmware, or the software code execution speed. As mentioned, the

Wishbone bus access protocol built on GBT SWT frames use one frame per Wishbone access, by doubling the address and data values in the 80-bit frame payload. This provides extra reliability, but it is not fast. Since this protocol is a likely suspect for the bottleneck, it might be worth redesigning this protocol to have the possibility of including several Wishbone transactions in the same GBT frame. A similar technique, called prefetching, is employed in the USB communication with the RUv0-CRU, where several Wishbone accesses can be queued and included in the same USB transfer. However, the USB frame frequency is much lower than that of the GBT link, so it is not certain if any time will be gained in this way.
Another possibility for speeding up the SWT Wishbone access protocol would be to have a special frame type which only contained data values which would be written, in order, to the same address as the last normal Wishbone transaction. In addition to an 8-bit header with a flag indicating this type of frame, a single GBT frame of this type could fit 7 Wishbone write transactions, potentially speeding up write operations by 7 times. However this is likely to move the bottleneck elsewhere, such as the Ultrascale-PA3 FIFO interface or the write controller. The Wishbone implementation on the Ultrascale might also be a larger bottleneck than the SWT protocol, as it has relatively slow overhead such as arbitration and triplication. Therefore it might require a redesign for SWT protocol optimizations to be worthwhile.

5.8.3 Reliability

The write controller is not the highest priority for reliability on the PA3, because it will likely not be used during ALICE run periods. The flash will only be written to during technical stops when the beam is off and radiation and downtime is not of concern. Therefore this module will not be triplicated, to save space on the FPGA. However, certain signals that might trigger activity in other critical modules, such as the flash interface command and execute signal, might need TMR anyway.
Nevertheless, the module has been designed with reasonable reliability in mind. The firmware is no more complex than it needs to be, which reduces the risk of corner case bugs. As described in section 5.4.2, the fastest data path, from the Ultrascale FIFO, has been conservatively designed to use three clock cycles every byte read, to make sure all signals such as empty and data is sampled accurately. Some error signals have been implemented which can and is be checked by software during flash write operations to make sure no bytes have been missed or otherwise errors

have occurred during the write process.

6 Irradiation testing

6.1 Background

Since the readout unit will be operating in an irradiation environment in the ALICE detector cavern, it needs to be tested for radiation robustness during development. Radiation can cause several unwanted effects in electronics. For example, so called Single Event Upsets (SEUs) are caused by a high-energy particle or photon striking the electronics, and is completely random in nature. Such an event can for example induce a current in a node on the FPGA, causing a bit in a register to change its value. This is called a bitflip, and can lead to failure if the value of this register is critical for the operation of the FPGA. Another radiation effect that can cause failures are dose effects, which are not entirely random like SEUs, but is likely to happen once a total dose of radiation has been deposited in the electronics and could permanently render the affected component unusable. However, the components used on the RUv1 is resilient enough to dose effects, and the ALICE run times short enough, that the random SEUs are of more concern.

The goal of the irradiation testing is to gain data on how frequently readout electronics, specifically the RUv1 board, would fail under radiation, as well as how these failures manifest themselves and how to best mitigate them. One can estimate the probability of failure by measuring the number of failure events over the irradiation time period. Since the average radiation flux both in the irradiation test, and during operation in ALICE, is known, one can calculate an estimate for the probability of a failure during an ALICE run by simply dividing the probability estimate for the test with the ratio of the flux during the test and the flux in the RUv1 positions in the ALICE cavern. This is because the probability of a SEU, and thus failures SEUs might cause, scales linearly with increased flux.

The main SRAM-based Ultrascale FPGA on the readout unit is more susceptible to SEUs than the auxiliary flash-based PA3. For this reason, the irradiation testing will place heavier focus on the Ultrascale, and it needs a higher level of protection in its firmware than the PA3. However, since the work in this thesis is centered around the PA3 and the communication links, this will be discussed in detail as well.

6.2 Test setup

The procedures at the different irradiation campaigns for the ITS readout electronics are usually approximately the same. At arrival to the beam testing facility, the RUv1 board is mounted to a stand which can be moved in two axes remotely, as shown in figure 27. This is used to adjust the position of the board in front of the beam so that it exactly hits the desired component on the board. On some setups, a plate of lead is mounted in front of the RUv1, with coverable holes over the components, so that only the components under the open holes is irradiated and the rest of the board is shielded. For neutron beam testing, there is usually no such plate, as it would not absorb much of the neutron radiation anyway. All the communication and power cables that are needed for the RUv1 is attached and stretched out of the radiation area, as the board is not physically accessible during the beam test due to radiation. In a safe distance from the beam, the rest of the system was set up. This includes an ALPIDE sensor stave, an RUv0-CRU, and various power supplies and other equipment needed to support the readout electronics. The RUv0-CRU is connected to a laptop, which is remotely connected to for controlling the test setup. The team is located in the control room away from the irradiation environment. The whole setup is illustrated in figure 28.

A notable difference between the setup in the Bergen lab and the beam test setup was that all cards are powered by programmable Hameg power supplies. This allows for remote controllable power cycling, which is necessary during the test to reset the system after failures or latch-ups in the FPGA. This usually happens a few times each testing campaign.

The test procedure is automated by a Python script. Briefly explained, the script powers up, configures and initialized the system and starts a readout session, recording data from the ALPIDE stave. Periodically, preferably continuously, the Ultrascale is scrubbed by the PA3 to restore its configuration memory. Eventually, either the script will finish normally after a specified amount of time, or the radiation will cause errors. The errors can either be in the data being recorded in which case it won't be detected until later when the data is analyzed, or it can cause a critical failure of the test sequence such as being unable to communicate with the board. If that happens, it is logged and the system is reset until fixed and the test sequence restarted.

All relevant data from the test, such as readout data, logs with events and register

Figure 27: Mount with an RUv1, on the Prague beam test. Front shielding plate with a hole above the Ultrascale FPGA to be radiated can be seen.

dumps, are kept for analysis.

6.3 Ultrascale beam test in Prague

At the start of December 2017, the RUv1 were to be tested in radiation for three days at a cyclotron facility, Řež, outside of Prague in the Czech Republic[3]. The participants were the author, as well as five other members of the ITS WP10 team[4].
The focus of the testing was the Ultrascale main FPGA, and the test routine was to run a script described above while the Ultrascale was radiated. A second test was also planned for the flash chip on the RUv1, to calculate an estimate on how many bitflips in the flash one must expect per period of time. A known pattern would be written to the flash, then it would get irradiated with a known luminosity. Then the content of the flash would be read back and the amount of bitflips would be counted. During this beam test, it was planned to use and test the PA3 scrubbing feature.

[3]https://www.ujv.cz/en
[4]Matteo Lupi, Matthias Bonara, Tomas Vanat, Joachim Schamback, and Magnus Ersland.

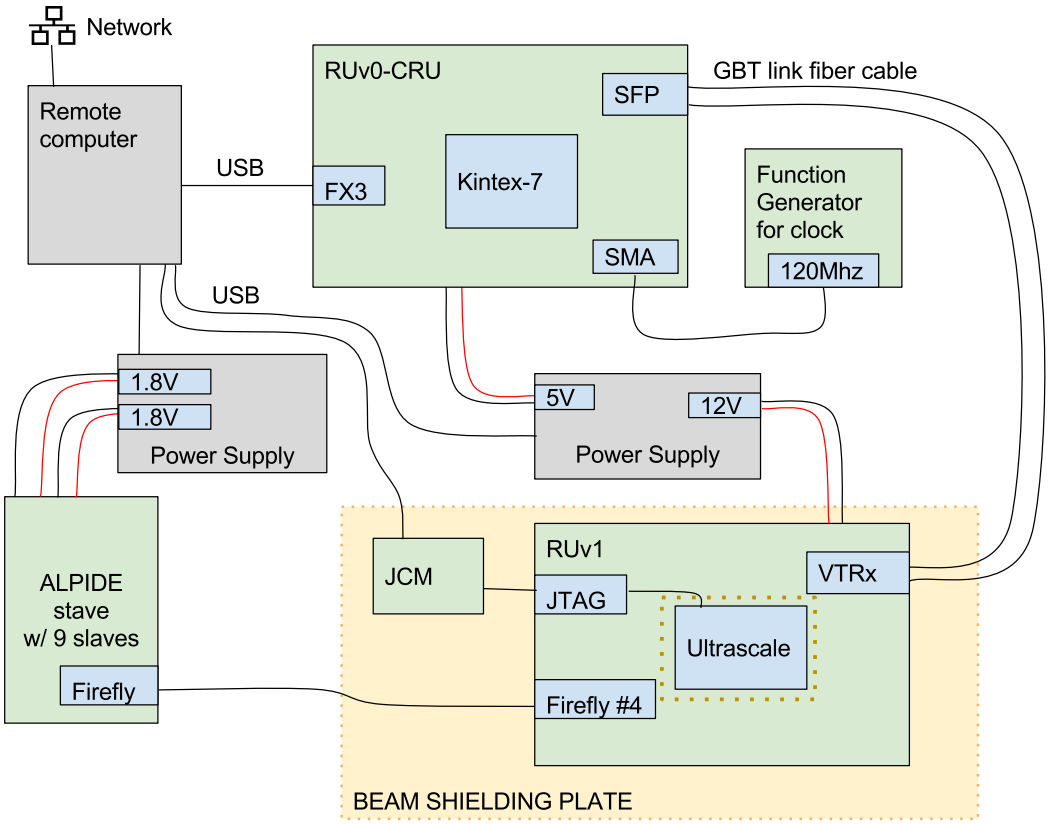

Figure 28: Block diagram of typical setup used for beam tests of the readout unit.

However, as a backup scrubbing solution, a JTAG Configuration Manager (JCM) is also mounted behind the shielding plate near the RUv1. The JCM is a tool that can program and scrub the FPGA over JTAG[23]. It can also read back the FPGA configuration memory, which can be used to inspect exactly where the configuration file was corrupted in the case of a single event upset or a failure in scrubbing.

6.3.1 PA3 I²C communication problem when prefetching

No problems with the I²C communication directly attributed to the radiation was observed.
However, a few other problems were discovered under operation, including the I²C communication with the PA3 failing when the communication with the CRU was prefetched for efficiency. It was discovered that this was because of insufficient waiting time between commands. The SCA chip with the I²C master does not have a

FIFO queue interface, so each I²C transaction must be finished before a new one can be started. But with the prefetching of commands to the CRU, there was no time for this. Due to how the prefetching works with queuing commands and sending multiple of them in the same USB packets, it is also not feasible to implement a normal busy polling, to halt operation until the SCA chip is ready. Therefore, a static delay period was added between operations instead. The delay is implemented in the RUv0-CRU firmware, so it is therefore deterministic and accurate, so once the appropriate delay period is found, it should work the same way every time. Another benefit is that the software controlling the USB communication does not slow down. All the commands to the CRU is sent and then cached on the board, ready to be executed once the delay ends. After this delay was implemented and the delay period found by trial and error, PA3 I²C communication worked even with prefetching of the commands. This was preliminary tested by creating a script that runs the PA3 register dumping function, described in the PA3 I²C communication chapter, over and over again in a loop for many minutes, equivalent to tens of thousands of I²C read operations, without observing any errors.

6.4 PA3 beam test in Oxford

In March 2018, a three day long beam test took place at ChipIR at STFC (Science and Technology Facility Council) Rutherford Appleton Laboratory near Oxford, England[5]. ChipIR is a facility at the ISIS Neutron and Muon Source Target Station 2 that provides a test beam of neutrons of an energy spectrum typical for atmospheric environments, around 200 to 800 MeV[24].

The focus of the test was the auxiliary PA3 FPGA on the readout unit. The participants were the author, as well as four other members of the ITS WP10 team[6]. The ITS WP10 team only had parasitic access to the beam for the first two days. This means we were able to place the readout unit in the beam and run tests, but we had no control over the configuration or downtime of the beam. As neutrons are not easily absorbed, the readout unit has near full flux even if it is placed behind other components being tested. On the last day of the test, 12 hours were dedicated to testing the readout unit where the WP10 has control of the facility.

[5]https://www.isis.stfc.ac.uk/Pages/ChipIR.aspx

[6]Matteo Lupi, Matthias Bonara, Hartmut Hillemanns, and Magnus Ersland.

6.5 Results

From the results of the beam test at Prague and others, it is estimated that the Ultrascale will experience one single event upset in its configuration memory ever 40 seconds during operation in ALICE, with an average flux of $1*10^{3}$ $cm^{-2}s^{-1}$, considering the cross-section of the configuration RAM of $2.55*10^{-15}$ $cm^{2}bit^{-1}$ and the amount of bits in the configuration RAM of $1.52*10^{8}$[25].
The results of the flash memory on the Prague test showed that the SEU cross section differs significantly between flipping a bit from 0 to 1, and 1 to 0. Bits of 0 are the most vulnerable, the cross section for 0 to flip to 1 is estimated to be 10^{-16} cm^{2}/bit, and for 1 to 0, the estimate is 10^{-21} cm^{2}/bit.
The PA3 proved to be relatively reliable in the radiation environment. Some SEUs were observed, particularly in the a CRC checking module that verified data being written to the Ultrascale during scrubbing. However these bitflips are not critical, as even if it is caused by an erroneous configuration, the next scrub cycle would repair the bit again within a couple of seconds. Only one time did the system become unresponsive and had to be reset by cycling the power. It is not confirmed whether this was because of a critical failure in the PA3 or another component. Other times when the Ultrascale became unresponsive, the PA3 Wishbone bus was still available and the system could be restored by resetting the PA3 using the Wishbone bus.

6.6 Conclusion and mitigation of radiation effects

6.6.1 Ultrascale

The two most important techniques for mitigation the effects of radiation in the main FPGA is Triple Modular Redundancy (TMR) for logic, Error-Correcting Code (ECC) for memory, and scrubbing. TMR solves the immediate effects of upsets by triplication the logic of the FPGA, and using a majority voting scheme to select the correct signal values if an error occurs. For example, if a logic block is responsible for calculating the CRC code for some data, the calculation is done three times in parallel. Suppose that one of the logic blocks experience an error caused by a single event upset, and the resulting CRC code is wrong. The majority voter then disregards this erroneous code and instead outputs the correct code from the two other logic blocks since this result represents the majority. Some critical modules of the Ultrascale FPGA, such as the clock networks and Mixed-Mode Clock Man-

agers (MMCMs) and the Dynamic Reconfiguration Port (DRP) cannot be wrapped by TMR protection, however due to low cross sections they have been shown to not be statistically affected by SEUs in 10 hours of running in ALICE, which is a typical mode of operation[25].
TMR can be broken by accumulating radiation effects that causes more than one of the three TMR blocks of a module to fail. Therefore the configuration of the FPGA needs to be regularly repaired to prevent this. This is the purpose of scrubbing. As explained in the background chapter, the PA3 scrubs the Ultrascale by overwriting its configuration memory with the data from a scrubbing bitfile store in the flash memory. This process does not disrupt the operation of the Ultrascale, as for example a normal reconfiguration would. A scrubbing cycle takes around 4 seconds, but this could possibly be optimized in the future.

6.6.2 Flash memory

The RUv1 will use three techniques to mitigate the effects of radiation. As mentioned, after radiation testing of the flash it was observed that the likelyhood of a bit 0 flipping to 1 due to an SEU is far more likely than a 1 flipping to 0. Configuration bitfiles are predominantly made up of 0's, with a ratio of approximately 20:1 for a scrubbing file and 50:1 for a programming file for the Ultrascale[26]. For this reason, to minimize the number of SEUs in the flash memory, stored bitfiles will be flipped in value, so the predominant content are 1's instead. When reading the file out from the flash and into the configuration of the Ultrascale, the content will be flipped back to their original values.
The second mitigation measure is adding hamming encoding to the bitstream. The bitstream will be divided into 128 byte blocks, and ECCs for the blocks will be inserted after each one before writing to flash. Then when writing the flash content to the Ultrascale configuration memory, the ECC will be calculated again and its integrity will be verified.
The third measure is to make use of the dual-bank feature of the flash memory chip to duplicate the written file. The flash memory chip on the RUv1 (Samsung K9WBG08U1M) contains two identical banks of memory, accessible using chip enable signals. If both chip enable signals are active, one can write to both banks at the same time with no speed penalty. This will be done when writing bitfiles to the flash. If the content of the two banks are different when reading back, we know the

content has been corrupted.
This gives a probability of a fatal error in the flash memory as the probability of a double bitflip in the same ECC encoded block in both banks of the flash. This probability can be calculated using the estimations for the cross section observed in the Prague beam test. The result is a probability P = $1*10^{-40}$ of a fatal error (double bitflip in the same ECC block) in any of the 192 readout units in ALICE, over the course of a 10 hour run[26]. This is an acceptable result.

6.6.3 PA3

The PA3 proved more reliable than the Ultrascale during the Oxford beam test. This is to be expected as the PA3 is a flash-memory based FPGA with larger flash cells than the flash memory. The larger structure implies that more charge must be deposited in the flash cell to be able to flip its value. Additionally, the number of flash cells are significantly lower than for the flash memory. No confirmed unrecoverable failures in the configuration memory were detected.
Some non-critical SEUs in the design space registers were detected in the PA3. A CRC module was integrated in the PA3 firmware that continuously calculated a CRC for the bitfile data during scrubbing. Mismatches in this CRC was observed numerous times, but it did not cause failures. The mismatches were likely caused by temporary transients that were latched by a register, or bitflips in the SRAM buffer memory cells. This does not affect the configuration memory of the PA3.
The final PA3 firmware will feature some protection against SEUs using TMR, like the Ultrascale. Not all modules will be wrapped in TMR due to limited resources on the PA3, only the most critical components that is used under operation such as the Wishbone bus, flash controller and configuration controller. This can be seen in figure 6. This, in combination with the generally low susceptibility for SEU should make the PA3 robust enough for operation in ALICE. This TMR protection was not present in the firmware version A112 used in this beam test, it will be implemented as of version A200.

7 Summary and conclusion

After the second long shutdown of the Large Hadron Collider at CERN, the Inner Tracking System in ALICE will be upgraded and a new readout system is needed. The University of Bergen is part of the collaboration developing this readout system. In this thesis, an up-to-date readout electronic chain has been set up and tested in the microelectronics lab at the University of Bergen to be used in this development effort. The readout chain setup consists of a readout unit version 1, and a common readout unit. Two implementations of the CRU have been used and tested. The first is an Arria 10 development card, mounted in a computer modeling an FLP node. The second is a readout unit board version 0, with CRU emulation firmware. The former is closer to the final hardware that will be used in the ALICE upgrade, but the latter was also needed to match the hardware used by other development groups that lacks Arria 10 implementations of the CRU.

The readout chain was tested to work reliably in all cases, except triggering of an ALPIDE sensor slave attached to the readout unit. The ALPIDE was not prioritized, as this thesis and the team at Bergen is more focused on other aspects of the readout chain, such as the PA3 auxiliary FPGA on the readout unit.

When the readout chain was set up and tested, it was used to develop various communication protocols along the chain, most importantly wishbone bus access on the PA3 from the CRU software via the I²C master on the GBT-SCA. Software for this purpose was written in both Python based on the central WP10 development group's software for the CRU, and in C++ based on the ReadoutCard library from the O² data center software group. During the work on this thesis, the PA3 firmware was also moved to a different wishbone bus configuration with single byte data and address widths. The software for I²C communication was adapted for this new format as well, and significant performance improvements were measured.

A path for uploading data to the flash memory chip on the PA3 with higher speed than the I²C interface was also implemented and tested. It uses a FIFO interface between a write controller module on the PA3 and the Ultrascale FPGA which is filled by Ultrascale bus accesses that uses the data field of the GBT protocol rather than the SCA EC field, resulting in higher bandwidth and far lower overhead. This data path was measured to be the fastest method of writing to the flash, with a bitfile upload using only about 30 seconds. It is also likely possible to reduce this time even further by eliminating bottlenecks, as much optimization on the Ultrascale and CRU

side has not been attempted.
The author participated in two irradiation testing campaigns, where the readout unit was tested for reliability in an irradiated environment. The results from these tests show that with sufficient protection through TMR of the critical modules, the components on the RUv1 is sufficiently radiation hard to work with an acceptable probability of failure in ALICE.
Potential future work related to the work in this thesis includes fully integrating the PA3 communication interfaces with the DCS, and integrating and testing the final version of the CRU based on the PCIe40 DAQ board. This should be relatively similar to the process with the Arria 10 DK CRU.

References

[1] LHC Guide. Mar 2017. URL https://cds.cern.ch/record/2255762.

[2] B Abelev et al (The ALICE Collaboration). Technical design report for the upgrade of the ALICE inner tracking system. *Journal of Physics G: Nuclear and Particle Physics*, 41(8):087002, 2014. URL http://stacks.iop.org/0954-3899/41/i=8/a=087002.

[3] Antonin Maire and David Dobrigkeit Chinellato. ALICE sub-detectors highlighted (LHC runs 1+2 // runs 3+4). General Photo, May 2017. URL https://cds.cern.ch/record/2302924.

[4] Gianluca Aglieri Rinella. The ALPIDE pixel sensor chip for the upgrade of the ALICE inner tracking system. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 845:583 – 587, 2017. ISSN 0168-9002. doi: https://doi.org/10.1016/j.nima.2016.05.016. URL http://www.sciencedirect.com/science/article/pii/S0168900216303825. Proceedings of the Vienna Conference on Instrumentation 2016.

[5] J. Schambach and M.J. Rossewij et al. Alice inner tracking system readout electronics prototype testing with the CERN "Giga Bit Transceiver". *Journal of Instrumentation*, 11(12):C12074, 2016. URL http://stacks.iop.org/1748-0221/11/i=12/a=C12074.

[6] The Alice collaboration and more. O2 : A novel combined online and offline computing system for the ALICE experiment after 2018. *Journal of Physics: Conference Series*, 513(1):012037, 2014. URL http://stacks.iop.org/1742-6596/513/i=1/a=012037.

[7] A. Szczepankiewicz. Readout of the upgraded ALICE-ITS. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 824:465 – 469, 2016. ISSN 0168-9002. doi: https://doi.org/10.1016/j.nima.2015.10.056. URL http://www.sciencedirect.com/science/article/pii/S0168900215012681. Frontier Detectors for Frontier Physics: Proceedings of the 13th Pisa Meeting on Advanced Detectors.

[8] Brendan Bridgford, Carl Carmichael, and Chen Wei Tseng. Single-event upset mitigation selection guide. *Xilinx Application Note, XAPP987 (v1. 0)*, 2008.

[9] J. Mitra, S.A. Khan, S. Mukherjee, and R. Paul. Common readout unit (CRU) - a new readout architecture for the ALICE experiment. *Journal of Instrumentation*, 11(03):C03021, 2016. URL http://stacks.iop.org/1748-0221/11/i=03/a=C03021.

[10] CRU team and Jean-Pierre et al. Introduction to the CRU project. Production Readiness Review 2018-04-27, 2018. URL https://indico.cern.ch/event/722231/contributions/2980408/attachments/1641026/2628648/Introduction_to_the_CRU_Project.pdf.

[11] Joachim Schambach and Piero Giubilato. ITS readout electronics. ALICE ITS Plenary presentation 2018-03-01, 2018. URL https://indico.cern.ch/event/708563/contributions/2909236/attachments/1609584/2555432/20110301_WP10_ITS_Plenary.pdf.

[12] P Moreira and et al. Ballabriga. The GBT Project. 2009. URL http://cds.cern.ch/record/1235836.

[13] A. Caratelli and S. Bonacini et al. The GBT-SCA, a radiation tolerant ASIC for detector control and monitoring applications in HEP experiments. *Journal of Instrumentation*, 10(03):C03034, 2015. URL http://stacks.iop.org/1748-0221/10/i=03/a=C03034.

[14] C. Gaspar and M. Donszelmann. DIM: A distributed information management system for the DELPHI experiment at CERN. In *Conference Record*, pages 156–158, 1993.

[15] Linux kernel documentation: Hugetlbpage.txt. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/vm/hugetlbpage.txt?id=HEAD. Accessed: 2018-05-09.

[16] GBTx java programmer URL. https://espace.cern.ch/GBT-Project/VLDB/Control/Forms/AllItems.aspx?RootFolder=/GBT-Project/VLDB/Control/Java%20programmer%20(GBTx%20programming%20tool). Accessed: 2018-03-13.

[17] CERN gitlab repository RUv0-CRU. https://gitlab.cern.ch/alice-its-wp10-firmware/RUv0_CRU, . Accessed: 2018-03-15.

[18] CERN gitlab repository RUv1-Test. https://gitlab.cern.ch/alice-its-wp10-firmware/RUv1_Test, . Accessed: 2018-03-15.

[19] CERN gitlab repository cru-tests. https://gitlab.cern.ch/alice-cru/cru-tests, . Accessed: 2018-03-13.

[20] Philips Semiconductors. The I2C-bus specification. *Philips Semiconductors*, 9397 (750):00954, 2000.

[21] Github repository: ALICE O2 readoutcard. https://github.com/AliceO2Group/ReadoutCard. Accessed: 2018-04-17.

[22] CERN gitlab repository CRU-ITS. https://gitlab.cern.ch/alice-its-wp10-firmware/CRU_ITS, . Accessed: 2018-04-04.

[23] A. Gruwell, P. Zabriskie, and M. Wirthlin. High-speed FPGA configuration and testing through JTAG. In *2016 IEEE AUTOTESTCON*, pages 1–8, Sept 2016. doi: 10.1109/AUTEST.2016.7589601.

[24] C. D. Frost, S. Ansell, and G. Gorini. A new dedicated neutron facility for accelerated see testing at the isis facility. In *2009 IEEE International Reliability Physics Symposium*, pages 952–955, April 2009. doi: 10.1109/IRPS.2009.5173387.

[25] Matteo Lupi. Mitigation of radiation effects. ALICE ITS Plenary presentation 2018-03-01, 2018. URL https://indico.cern.ch/event/708563/contributions/2915990/attachments/1609411/2555687/201803_ITS_PLENARY.pdf.

[26] Arild Velure. Remote programming and scrubbing. ALICE ITS Plenary presentation 2018-03-01, 2018. URL https://indico.cern.ch/event/708563/contributions/2915988/attachments/1609361/2555400/Plenary_28-02-18.pdf.

A Instructions for setting up Arria 10 DK CRU in host computer

A.1 Hardware

Mount the Arria 10 DK in a free PCI-e slot in the host computer. Insert a QSFP plug with 8 optical fiber cables (4 bidirectional channels, 850nm) into the appropriate port on the CRU. Connect a USB cable from the host computer to the micro-USD slot in the back panel of the CRU (integrated USB Blaster programmer).

A.2 Software

The computer should run CentOS 7.3, as recommended by CERN. The computer needs a series of drivers and programs to work with the CRU. First, install Intel Quartus Prime to program the FPGA on the CRU, or to build the firmware if needed. The CRU uses onboard clock generators on the Arria 10 Development card. These must be configured. Install the Arria 10 SDK[7] which contains utilities for configuring these clocks. Launch the ClockController.sh tool with root which is part of the Board Test System included with the SDK in folder examples/board_test_system. Set all clocks to 240 MHz.
Next, the drivers, tools and toolchain for the CRU and ALICE development is installed. This is provided by the CERN ALICE O2 group. Both the FLP Prototype software[8] and additional ReadoutCard software components[9] must be installed. Step by step instructions and commands for doing this can be found in the links in the footnotes.

A.3 Firmware

The CERN CRU team release pre-compiled firmware bitstreams for every CRU release, these can be programmed directly to the CRU using the Quartus programming tools. The bitfiles are found on CERN's internal wiki page "ALICE Web > CRU >

[7] https://www.altera.com/content/dam/altera-www/global/en_US/support/boards-kits/arria10/FPGA/arria10GX_10ax115sf45_fpga_v15.1.2p2.zip
[8] https://alice-o2.web.cern.ch/node/157
[9] https://alice-o2.web.cern.ch/node/161

"CruHwFwSwDev"[10].
To be able to program the Arria 10 you might need to set up USB Blaster drivers. These drivers can be found in the Intel Quartus installation folder. Do these commands (adjusting the file path if a different Quartus version is installed):

```
cd /opt/intelFPGA_pro/17.1/qprogrammer/drivers/wdrvr/linux64
./setup_inst_dir
./configure
./configure.wd
```

Then the firmware bitfile can be programmed to the Arria 10 by running this command as root in the same folder as the bitfile:

```
quartus_pgm --cable=1 --mode=JTAG --operation="p;XXX"
```

Replace XXX with the name of the bitfile, for example cru-preint9-pcie40.sof.

A.4 Initialization

The CRU must be initialized before use. After programming the Arria 10 on the CRU, first run the transceiver calibration script like this, as root:

```
system-console -cli --script=standalone-startup.tcl
```

The script is found in the subfolder utils of the firmware releases.
Now reboot the host computer (using the restart button or via the OS, so that the PC doesn't lose power).
Then the card is ready for control tasks such as communicating with the GBT-SCA. But to use data readout related functionality you must also allocate hugepages. This is done by running the script roc-setup-hugetlbfs as root, which is part of the ROC driver tools.

A.5 Reconfiguration after power cycles

The CRU's FPGA and clock chips are volatile, which means they lose their configuration and have to be reprogrammed every power cycle. In addition, it is necessary to re-calibrate the CRU, and allocate memory hugepages for the ROC software. In other words, the instructions in the two last sections must be repeated for every power cycle.

[10]https://twiki.cern.ch/twiki/bin/viewauth/ALICE/CruHwFwSwDev

A.6 Connecting to RUv1

To connect to an RUv1, connect one of the fiber cables to a transceiver in the slot next to the Firefly ports on the RUv1. The transceiver must used 850 nm wavelength, such as the VTRx. The fiber cables marked A8 and A1 is the default GBT channel 0. If the GBTx is not fused, it must be configured. This is done with the GBTx Java Programmer application[11] and the USB-I2C dongle. Connect the dongle to the CRU host PC, and to the RUv1 (but on RUv1.0 the two rows on the header is reversed so a manual cable must be constructed). Java runtime must also be installed before the programmer tool can be used. Now run the programmer tool with this command as root:

```
java -jar programmerv2.20180116.jar
```

Click "import i.." button. Choose RUv1_Test/modules/gbt/software/GBTx_configs/GBTx0_Config.xml (part of the RUv1_Test repository[12]. Click "Write GBTX" button. To verify, click "Read GBTX" button, and state should say "Idle, 18'h".
Now you can reach the RUv1 using their relevant software tools (see last appendix chapter for PA3 software instructions), if the FPGAs are programmed.

A.7 Tips

With some software such as ROC and Quartus tools, they might need both root access and path variables to be able to work, so a custom sudo alias, "mysudo" that carried over the path variables to the sudo command was created, by adding the following line to the .bashrc file:

```
alias mysudo='sudo env "PATH=$PATH"
"LD_LIBRARY_PATH=$LD_LIBRARY_PATH" "QUARTUS_ROOTDIR=$QUARTUS_ROOTDIR"'
```

Now, software could be run from the terminal using the commands "mysudo [program]".
A lot of the commands and steps in this chapter are taken from the CRU getting started guide which can be found on the CruHwFwSwDev twiki page linked in a footnote earlier. This guide also contains some more useful information.
A dedicated email for CRU support also exists, namely alice-cru-fw-support@cern.ch.

[11] https://espace.cern.ch/GBT-Project/VLDB/Control/Forms/AllItems.aspx?RootFolder=/GBT-Project/VLDB/Control/Java%20programmer%20(GBTx%20programming%20tool
[12] https://gitlab.cern.ch/alice-its-wp10-firmware/RUv1_Test

B More details on testing of CRU

B.1 Card detection

One of the ROC software tools is roc-list-cards. This program lists all ROC cards in the system, including the CRU if it is detected. After configuring the software as described in the chapter "Setup", the program detected the CRU successfully and gave the following output:

```
[gitlemikkelsen@iftnc041239 py]$ mysudo roc-list-cards
infoLoggerD not available, falling back to stdout logging
============================================================
#  Type   PCI Addr   Vendor ID   Device ID   Serial  FW Version
------------------------------------------------------------
0   CRU    05:00.0    0x1172      0xe001      0       n/a
============================================================
```

The line warning about infoLoggerD was disregarded as it is acceptable to only view the result in the terminal for these tests. This line, if present, will be omitted in the following documentation.
However, the program also attempts to open up a DMA channel to the card and read some information such as the FW Version. This fails in the above command because the memory for the DMA has not yet been allocated. The memory is allocated by setting up hugepages designated to the ROC software. There is a script for setting up the hugepages called roc-setup-hugetlbfs. This is run as root:

```
[gitlemikkelsen@iftnc041239 CRU]$ mysudo roc-setup-hugetlbfs
File '/etc/flpprototype.d/hugepages-2MiB.conf' not found, allocating
   default amount of 2 MiB hugepages: 128
File '/etc/flpprototype.d/hugepages-1GiB.conf' not found, allocating
   default amount of 1 GiB hugepages: 0
Adding 'pda' group
Creating hugetlbfs mounts
Setting permissions on hugeltbfs mounts

Hugepages:
      Size  Minimum  Current  Maximum  Default
   2097152      128      128      128        *
1073741824        0        0        0
[..]
```

It is observed that a couple of configuration files can be used to customize the number of hugepages in specific sizes, but since these files have not been created, the default settings of 128*2MiB hugepages are used. Now that the hugepages are set up, the roc-list-cards program is run again:

```
[gitlemikkelsen@iftnc041239 CRU]$ mysudo roc-list-cards
2017-11-27 12:33:40.165836      [pci=05:00.0 serial=0 channel=0] Acquiring
    DMA channel lock
2017-11-27 12:33:40.166685      [pci=05:00.0 serial=0 channel=0] Acquired
   DMA channel lock
2017-11-27 12:33:40.198609      [pci=05:00.0 serial=0 channel=0]
    Initializing memory-mapped DMA buffer
2017-11-27 12:33:40.199429      [pci=05:00.0 serial=0 channel=0] Scatter-
    gather list size: 1
2017-11-27 12:33:40.202227      [pci=05:00.0 serial=0 channel=0] Buffer is
    hugepage-backed
2017-11-27 12:33:40.202317      Enabling link(s): 0
2017-11-27 12:33:40.203434      [pci=05:00.0 serial=0 channel=0] Releasing
    DMA channel lock
================================================================================
#   Type   PCI Addr   Vendor ID   Device ID   Serial   FW Version
    ------------------------------------------------------------------------------

0   CRU    05:00.0    0x1172      0xe001      0        20171016-180400-6
    cce25a6
================================================================================
```

Now, the program aquires a DMA link with the CRU successfully, and reads the FW version which consists of the date and part of its git commit hash. The output above was from a test performed on firmware released 2017-10-16, but the procedure and result is the same as on the firmware and ROC software used when the CRU first arrived in September.

B.2 Register access

Next it was attempted to read and write registers using the roc-reg-* programs. These implement low level functions for directly communicating with the CRU over the PCI-e interface. They are meant for development and debug purposes only, the underlying code will be used under the hood of higher level software such as the ALF/FRED network communication tools.

For read register access test, it will be attempted to read the version and error info of the board support package (BSP) module in the firmware. This module has the base address of 0x00200000, and the first four registers (each 32-bit) of this address space is dedicated to the version and error info. The fourth register (0x0020000C) should be a constant value representing the ASCII string "vinf".
The roc-reg-read program is used to read the register. This takes three parameters: "id", which is the PCI ID of the CRU, displayed in the output of the roc-list-cards program. As seen in the previous test, the id of our CRU is 05:00.0. The second paramter is "channel", for access to the avalon bus, the channel to be used is 2. The third parameter is "address", the register address. The program is run:

```
[gitlemikkelsen@localhost ~]$ mysudo roc-reg-read --id=5:0.0 --channel=2
   --address=0x0020000c
0x76696e66
```

The program ran successfully and output the value "0x76696E66". If one convert each byte to ASCII, the result is "vinf", as expected.
Next, writing to a register is tested. A writable register that will be used later is the data register of the SCA submodule of the GBT link #0 module. This Avalon slave has a base address of 0x04224000, and the write data register is located at address 0x04224020. The roc-reg-write program will be used, which has the same parameters as roc-reg-read, in addition to the "value" with the 32-bit data to be written. A test value of 0xDEADBEEF is attempted written and then read back.

```
[gitlemikkelsen@localhost ~]$ mysudo roc-reg-write --id=5:0.0 --channel=2
    --address=0x04224020 --value=0xdeadbeef
0xdeadbeef
[gitlemikkelsen@localhost ~]$ mysudo roc-reg-read --id=5:0.0 --channel=2
   --address=0x04224020
0xdeadbeef
```

The register was successfully written to.

B.3 DMA benchmark

Next the included PCI-e DMA benchmark tool was tested. This reads a stream of data from the CRU and measures the achieved bandwidth. This is done using the ROC program roc-bench-dma. Before this program can be used, hugepages must be allocated using roc-setup-hugetlbfs.

This program has numerous parameters to configure variables such as memory buffer sizes. For the tests, a data size of 10GiB is used with the parameter “bytes=10Gi”. Like with the register access programs, the CRU PCI-e ID need to be speficied with the parameter “id”. All data links also need to be enabled with the parameter “links='0-31'”. For the first test, the buffer is adjusted to fit the default hugepage configuration of 128*2MiB hugepages with the parameters “buffer-size=256Mi” and “superpage-size=2Mi”. This produced the following output:

```
[gitlemikkelsen@localhost ~]$ mysudo roc-bench-dma --id=5:0.0 --bytes=10
    Gi --buffer-size=256Mi --superpage-size=2Mi --links="0-31"
2017-11-28 13:00:37.242639      DMA channel: 0
2017-11-28 13:00:37.242746      IOMMU enabled
2017-11-28 13:00:37.243515      Using buffer file path: /var/lib/hugetlbfs
    /global/pagesize-2MB/roc-bench-dma_id=5:0.0_chan=0_pages
2017-11-28 13:00:37.243669      Buffer size: 268435456
2017-11-28 13:00:37.243698      Superpage size: 2097152
2017-11-28 13:00:37.243738      Superpages in buffer: 128
2017-11-28 13:00:37.243769      Page size: 8192
2017-11-28 13:00:37.243794      Page limit: 1310720
2017-11-28 13:00:37.243818      Pages per superpage: 256
2017-11-28 13:00:37.243842      Generator data size: <internal default>
2017-11-28 13:00:37.333497      [pci=05:00.0 serial=0 channel=0] Acquiring
     DMA channel lock
2017-11-28 13:00:37.334241      [pci=05:00.0 serial=0 channel=0] Acquired
    DMA channel lock
2017-11-28 13:00:37.362082      [pci=05:00.0 serial=0 channel=0]
    Initializing memory-mapped DMA buffer
2017-11-28 13:00:37.405941      [pci=05:00.0 serial=0 channel=0] Scatter-
    gather list size: 1
2017-11-28 13:00:37.407798      [pci=05:00.0 serial=0 channel=0] Buffer is
     hugepage-backed
2017-11-28 13:00:37.407874      Enabling link(s): 0 1 2 3 4 5 6 7 8 9 10
    11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
2017-11-28 13:00:37.407907      Card type: CRU
2017-11-28 13:00:37.407966      Firmware info: 20171016-180400-6cce25a6
2017-11-28 13:00:37.407980      Starting benchmark
2017-11-28 13:00:37.407997      [pci=05:00.0 serial=0 channel=0] Starting
         DMA
Time         Pushed          Read            Errors          C
00:00:03     1310720         1306372         0               43.6
2017-11-28 13:00:40.648724      Popped 0 excess pages
```

```
2017-11-28 13:00:40.648767      [pci=05:00.0 serial=0 channel=0] Stopping
        DMA
Seconds         3.0203
Pages           1310720
Bytes           1.07374e+10
GB              10.7374
GB/s            3.55508
Gb/s            28.4406
Errors          0
2017-11-28 13:00:40.648887      Benchmark complete
2017-11-28 13:00:40.649848      [pci=05:00.0 serial=0 channel=0] Releasing
        DMA channel lock
```

The benchmark completed successfully with no errors. The achieved data rate was 28.44 Gbps, or 3.56 GBps. This is not ideal, in theory a speed of well over 6 GB/s should be possible.

In order to improve the data rate, it was attempted to tweak the parameters of the benchmark. First, errorchecking was disabled with the parameter “no-errorcheck”. This improved the speed to 4.47 GBps:

```
[gitlemikkelsen@localhost ~]$ mysudo roc-bench-dma --id=5:0.0 --bytes=10
    Gi --links="0-31" --no-errorcheck --superpage-size=2Mi --buffer-size
    =256Mi
[...]
Time        Pushed          Read            Errors          C
00:00:02    1310208         1310208         n/a             44.3
[...]
Seconds         2.40026
Pages           1310720
Bytes           1.07374e+10
GB              10.7374
GB/s            4.47345
Gb/s            35.7876
Errors          n/a
[...]
```

Then, it was attempted to use a different hugepage configuration. Instead of using 128*2MiB hugepages, a single 1GiB hugepage was allocated and used. This first required creating a config file /etc/flpprototype.d/hugepages-1GiB.conf with the content “1”. Now, roc-setup-hugetlbfs was run again, and this set up the 1GiB

hugepage. Now, roc-bench-dma was run again with the following parameters removed: “buffer-size” and “superpage-size”. These will be set to their default values which is 1Gi and 1Mi respectively, as can be observed in the program output:

```
[gitlemikkelsen@localhost ~]$ mysudo roc-bench-dma --id=5:0.0 --bytes=10
    Gi --links="0-31" --no-errorcheck
[...]
2017-11-28 13:05:30.888955      Using buffer file path: /var/lib/hugetlbfs
    /global/pagesize-1GB/roc-bench-dma_id=5:0.0_chan=0_pages
2017-11-28 13:05:30.889117      Buffer size: 1073741824
2017-11-28 13:05:30.889148      Superpage size: 1048576
2017-11-28 13:05:30.889171      Superpages in buffer: 1024
[...]
Time        Pushed          Read            Errors          C
00:00:02    1307648         1307648         n/a             44.3
[...]
Seconds      2.30026
Pages        1310720
Bytes        1.07374e+10
GB           10.7374
GB/s         4.66791
Gb/s         37.3433
Errors       n/a
[...]
```

A further increase in data rate was observed, at 4.67 GBps.

C Instructions for using the PA3 I2C communication and write controller module

C.1 Ready made software tools

The git repository CRU_ITS in branch GM_dev[13] contains python scripts for interfacing with the PA3 using the Arria 10 CRU. Specifically software/py/PA3.py, with software/py/pa3.py being a Python Fire wrapper to run from command line like this: python pa3.py [function] [arguments]. This software can be adapted to the RUv0-CRU by changing all the referenced functions for SCA and SWT classes to the appropriate classes made for the RUv0-CRU tools in the git repo RUv1-Test[14].

C.2 I2C communication

Before I2C communcation can be done, the GBT-SCA chip must be initiated and its I2C module must be enabled. It is enabled by sending a command to channel 0 with cmd 0x02 and data 0xff000000. This sets the whole control register containing the I2C modules to 1, effectively enabling all of them. Then the specific channel used for the PA3 I2C (#3) is configured to be 1 MHz and otherwise default by writing a command to channel 3 with cmd 0x30 and data 0x0b000000. This is all done in the PA3.py function init().
An I2C write transaction can be initiated by sending a command to the GBT-SCA chip channel 3, cmd 0x82 and data 0xYYZZ0000 where YY is 7-bit address and ZZ is 8-bit data. This is done in PA3.py function write(addr, data).
An I2C read transaction can be initiated by sending a command to channel 3, cmd 0x86, data 0xYY000000 where YY is 7-bit address, then reading the response from the GBT-SCA chip, of which bit 23 down to 16 will be the read value of the register. This is done in PA3.py function read(addr).
Function usage summary is as follows:

```
python pa3.py init
python pa3.py read ADDRESS
python pa3.py write ADDRESS VALUE
```

[13] https://gitlab.cern.ch/alice-its-wp10-firmware/CRU_ITS/tree/GM_dev
[14] https://gitlab.cern.ch/alice-its-wp10-firmware/RUv1_Test

C.3 Writing data to flash

To write data to flash, first erase the blocks that will contain the data, by writing the address of the flash block to registers 38, 37, 36 where bit 0 of register 36 is the least significant bit, and bit 7 of register 38 is the most significant bit (although the flash address can't get big enough for bit 7 here to be used). Then send erase command to flash interface by writing data 0xA0 to register 32. Now wait until flash is ready by polling register 33 until bits 5 down to 3 are all 0. Repeat this for all blocks.
Then start a write operation by writing the address of the flash page to start writing to to register 38, 37 and 36 again, then write data 0b10000001 (if you want to use Ultrascale FIFO path) or 0b10000010 (if you want to use I2C path) to register 51 (write ctrl command register).
Now start writing data to the selected path. To write data to the Ultrascale FIFO, write two and two bytes to address 0 of the module WB2FIFO on the RUv1 (see SWT software tools for implementation). To write data using I2C, send one byte each to register 49.
Continue writing all data until finished. Data must be a multiple of 4096, otherwise fill up with 0's until a multiple of 4096 is reached.
Then stop the write operation by writing 0b10000100 to register 51.
To be safe, you can read the status register (51), if no errors occurred and the write operation finished, it should have value 0. If it doesn't, refer to table 5 and the bitmapping of register 52 to find out what is wrong.
Functions for writing single pages or whole bitfile including configuration page, either via I2C or Ultrascale FIFO, are found in PA3.py as functions flashWrite*().
Function usage summary is as follows:

```
python pa3.py flashReadPage PAGE_ADDRESS [--hexify]
python pa3.py flashWriteBitfile BITFILE [PAGE_ADDR=0x100]
        [SCRUBFILE] [SCRUBFILE_ADDR]
python pa3.py flashWriteBitfileI2C BITFILE [PAGE_ADDR=0x100]
        [SCRUBFILE] [SCRUBFILE_ADDR]
python pa3.py test-flash-access
And more... (see PA3.py)
```

C.4 Example

Below is a practical example of software usage. First the Arria 10 DK CRU and an RUv1 need to be set up and configured as explained in appendix A. Then, the following series of commands will initiate communication, upload a bitfile and scrubbing file (via I2C) to the flash and then program the Ultrascale with that bitfile. This must be done from a bash terminal in the software/py folder of the CRU_ITS repository. It is also assumed that an RUv1 bitfile exists with the path ~/RUv1.bit, as well as its scrubbing bitfile at ~/RUv1_scrub.bit.

```
python pa3.py init
python pa3.py flashWriteBitfileI2C ~/RUv1.bit 0x100 ~/RUv1_scrub.bit 0x200
python pa3.py write 8 0x81
```