Department of physics and technology

Development of the ALICE Busy Box

Masters thesis

by

Magne Munkejord

University of Bergen
October 2007

Abstract

The European Organisation for Nuclear Research (CERN) is building a large circular particle accelerator called the Large Hadron Collider (LHC). The LHC will be able to accelerate and contain two beams of particles, travelling in opposite directions around the accelerator ring.

At four points around the ring the beams will intersect and the accelerated particles will collide. Four large detector experiments are being constructed around these points to observe the collisions.

The ALICE experiment is designed to study the effects of large hadrons colliding at very high energies. Collisions will occur with a average rate of approximately 8 kHz. Only a subset of these collisions provide interesting data and a sophisticated trigger system is employed to select which events should be recorded.

The Front End Electronics (FEE) of the detector sub systems produces large amounts of data for each event that is recorded. The event data is transfered out of the detector and up to the Data Acquisition (DAQ) system over high speed optical fiber links.

The Busy Box is a FPGA based device that monitors and verifies the transfer of event data from the FEE to the DAQ system of the ALICE experiment at CERN. By looking at which events have been triggered for readout and which of these events have arrived at the DAQ the Busy Box keeps track of the number of used buffers in the FEE. The Busy Box will assert a busy signal to let the trigger system know if the buffers run full and thereby prevent additional triggers and a possible overflow and data corruption in the buffers of the FEE. This feature makes it possible to run the system at its maximum capacity concerning the readout bandwidth from the FEE to the DAQ system where the busy generation will reduce the trigger rate whenever the bandwidth is insufficient.

The Busy Box will be utilized to four of the sub detector systems of ALICE; PHOS, EMCal, FMD and the Time Projection Chamber (TPC). For the TPC sub detector system the Busy Box needs to communicate with over 200 units in the DAQ system to verify the transfer of each event. The process of event transfer verification must execute significantly faster than the maximum readout rate of the detector which could reach a kHz.

Acknowledgements

The work presented in this thesis has been carried out with the help and support of many people. Most of them are employees or students at the Department of Physics and Technology at the University of Bergen where I spent my time as a master student.

First of all I would like to thank my supervisor Kjetil Ullaland for his guidance and support during this work. Dieter Röhrich deserves credit for sharing his wisdom and knowledge with me.

I want to thank Johan Alme and Ketil Røed for discussions and feedback on my work, and for their great help whenever I got stuck. Are Stangeland has been a great co-student and travel companion on our trip to CERN. And from CERN I would like to thank Csaba Soós for his company and for his work with integrating and testing of the Busy Box.

Big thanks to my friends and co-students Yngve Skogseide, Edvard Fosdahl, Knut Solvåg, Olav Torheim, Thomas Gundersen and Anders Rossebø for times well spent at room 446 — the microelectronics lab.

Finally I would like to thank my parents, Kari and Einar, other family and friends for motivation and support.

Contents

Chapter 1

Introduction

1.1 European Organisation for Nuclear Research (CERN)

CERN is the European organization for nuclear research. It was founded in 1954 after World War II as a cooperation between 12 European countries to promote nuclear research. Today there are 20 nations that have status as members while several other nations and organizations have observer status. The organization makes it possible to build large experiments that would be too expensive for one nation alone.

CERN's facilities are located at the border between France and Switzerland, about an hours drive from Geneva. About 3000 full time employees runs and develop the facilities while some 6500 researchers and scientists spends time there to research.

1.2 The Large Hadron Collider (LHC)

The LHC [1] is a large particle accelerator currently under construction at CERN's facilities. When it is finished it will be the largest and most powerful human built particle accelerator in the world. LHC is circular accelerator with a circumference of approximately 27 km. It is built in a tunnel that is 100 to 150 meters underground. The accelerator will be able to accelerate protons and heavy ions up to 99,9 $\%$ of the speed of light. Thus the particles will do approximately 11000 round trips in the accelerator per second. Figure 1.1 shows an illustration of the construction.

The particles are accelerated in several steps in other accelerators before bunches of particles are injected into the LHC where they will be accelerated up to their maximum velocity. The charged particles are accelerated by strong electric fields. The fields are generated by shooting electromagnetic waves into resonance cavities which oscillates at 40.08 MHz. This way particles of the same charge can be accelerated in both directions as long as they are in the electric field at the right time. What is important to note is that most of the bunches will be empty. Hence the actual rate at which bunches passes each other is lower than the nominal bunchcrossing rate of 40.08 MHz.

Two beams made up of particle bunches are accelerated in opposite directions around the accelerator ring. The two beams will travel in separate vacuum pipes so that there will not be

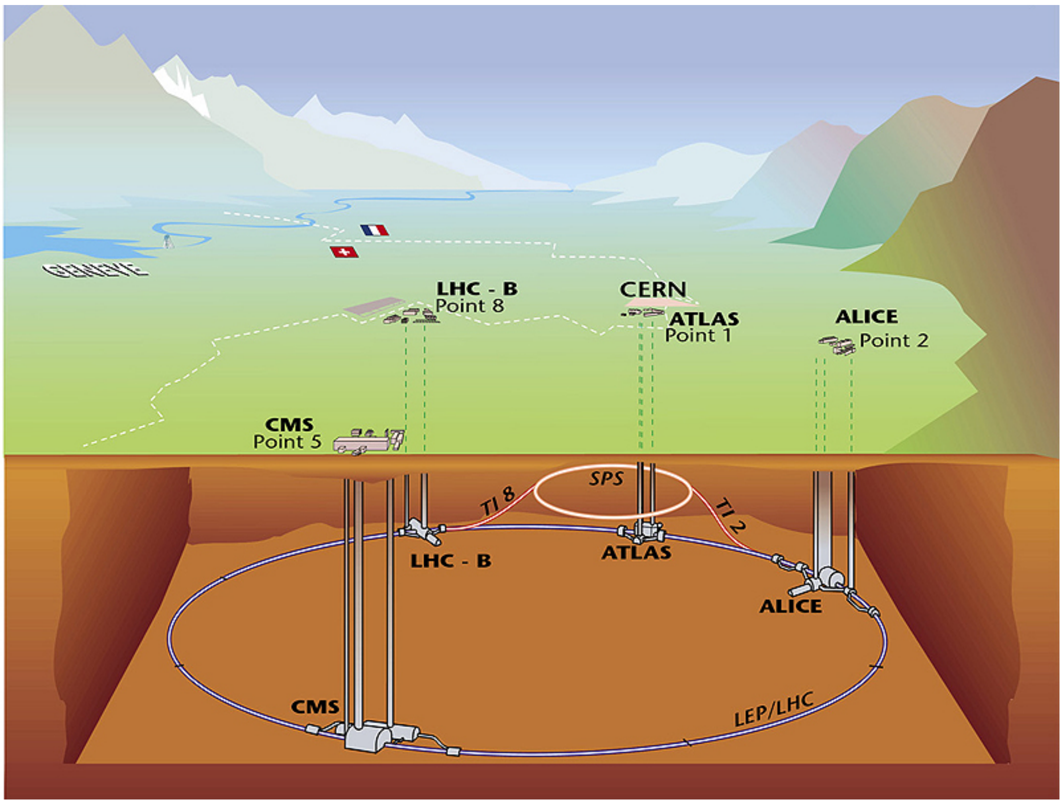

Figure 1.1: Illustration of the LHC tunnel under CERN facilities on the border between Switzerland and France [1].

any other particles to collide with. The particles are kept in their orbit by strong magnetic fields that are generated by supercooled and superconducting magnets.

At four points around the accelerator ring the beams intersect and when bunches of particles crosses each others paths some of them might collide. In other words there is only a statistical probability that a bunchcrossing will result in a collision. As the particles collide the number of particles in each bunch decreases and thus the probability for collisions which ultimately reduces the collision rate. Since it is impossible to refill the bunches, the accelerator must be emptied when the collision rate is to low. A typical LHC run will last about 6-7 hours before the system must be restarted.

The reason for building the LHC is to be able to recreate physical processes that are known to happen out in the universe. Large detector systems are required to observe these events and therefor it is necessary to recreate them in a laboratory to be able to study them. Even though the energies in the collisions in the LHC will be the highest ever in man-built accelerators they are relatively low compared to the energies of particles observed in the cosmic radiation from far out in the universe. These particles have been created by processes like supernovas and the creations of black holes. It is expected that the results from research involving the LHC will illuminate these processes and other mysteries like the dark matter, anti matter and super symmetry.

Researchers hope that the energy density in the LHC collisions will be high enough to create Quark-Gluon-Plasma (QGP). Nucleons are made of quarks that are bound together be the strong force which is carried by particles called gluons. QGP is believed to be a state of matter where the quarks and gluons move freely. It is believed that all matter existed in this state 10 to 30 μs after the *Big Bang* [2]. By studying matter in this state scientist will be able to put existing theories to the test and to better understand what matter is and how the elementary forces interact.

The QGP will quickly expand and as the energy density decreases the quarks in the plasma will recombine into new particles that are spread out in all directions. These new secondary particles can be detected and will bring a lot of information about what happened in the collision. Very large and advanced detector systems are required to obtain the desired information and six different detector experiments are built around the points where the LHC beams intersect.

1.3 Collision rates in the LHC

The rate at which collisions occur inside the detector depends on many factors. As described earlier the particles travel in bunches. For a collision to occur two bunches must cross paths in the detector. The rate at which this happens depends of many bunches have been injected into the LHC. This number is higher for proton-proton mode than Pb-Pb mode. The probability that some particles in two crossing bunches collide depends among other things on how many particles are in the bunches. After a while the number of particles in each bunch will be reduced as particles collide.

Another factor is the focus of the beams. Since the particles in the bunches have the same electric charge they will repel each other. The keep the bunches together the beams focused several places around the LHC. At the places where the beams intersect it is possible to adjust the focus of the beams so the collision rate to some extent can be controlled.

1.4 A Large Ion Collider Experiment (ALICE)

As mentioned, the ALICE detector experiment is optimized to study heavy ion collisions (Pb-Pb[1]) but also proton-proton collisions will be studied. The main goal for the ALICE experiment is to prove the creation of and study the QGP.

When heavy ions collide in ALICE it is expected that tens of thousands of new particles will be created and pass through the detector system. The ALICE detector system must be able to track and identify most of them.

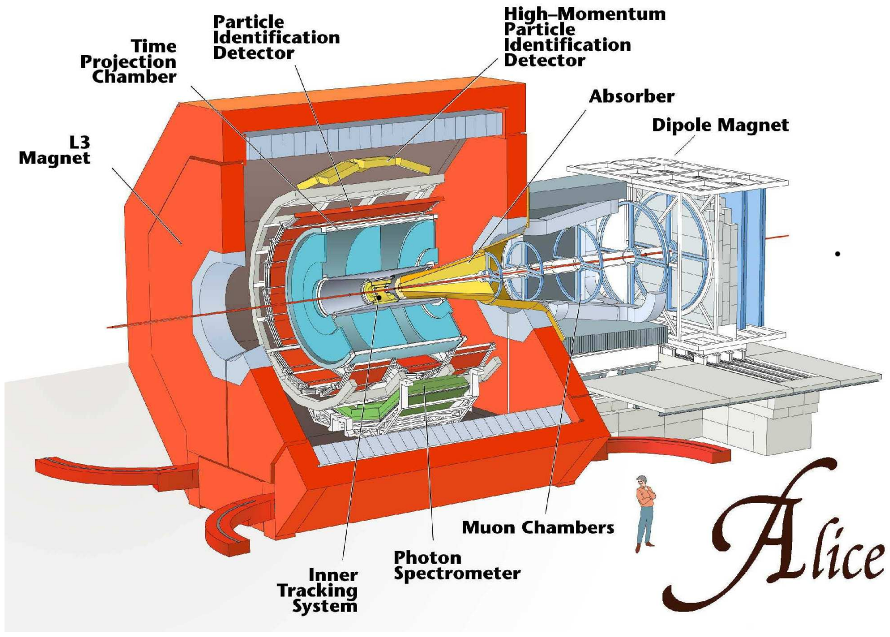

Figure 1.2: Illustrasjon av ALICE

The illustration in figure 1.2 shows an overview of ALICE. The beams are represented by a thin line through the detector and the particles will collide in the middle of the detector. ALICE consists of many sub detector systems. All the sub systems that are built around the collision point are called the central barrel detectors. In addition there are several sub detector systems in front and back of the central barrel detectors. These are used to determine trigger conditions.

The ALICE central barrel detectors [3] :

Inner Tracking System (ITS) Six layers of silicon wrapped around the beam pipe. It is capable of tracking particles with low momentum that does not reach out to the other detectors.

Time Projection Chamber (TPC) The main tracking detector of ALICE. Can reconstruct a 3D image of the trajectories of charged particles that passes through it.

Transition Radiation Detector (TRD) Can detect electrons and positrons.

[1]Pb is the symbol for lead in the periodic table of elements.

Photon Spectrometer (PHOS) Based on lead tungstate scintillator crystals. Can detect photons.

High Momentum Particle Identification Detector (HMPID) Can detect particles with high momentum (fast moving particles).

Time Of Flight (TOF) Detects particles as they pass and measure the time-of-flight from the origin of the collision.

A large solenoid magnet (L3 magnet) is built around the central barrel detectors. This magnet will create a uniform homogeneous magnetic field parallel to the beams. When a charged particle travel through this field its trajectory will be curved depending on the particles charge and momentum.

1.5 Time Projection Chamber (TPC)

The TPC [4] is constructed as two coaxial cylinders around the collision point (the ITS will be placed inside the TPC). An illustration of the TPC structure is shown in figure 1.3. The volume between the cylinders is filled with gas[2]. At the axial center the chamber is divided in two by a thin aluminized Mylar foil which acts as an electrode. The electrical potential of electrode is approximately 100 kV lower than the end caps of the chamber which creates electrostatic fields from the end caps towards the center electrode. When a charged particle travels through the chamber it will ionize the gas molecules in its path. The result is a trail of freed electrons and gas ions. The electric field makes the ionized gas molecules to drift towards the center electrode and the freed electrons towards the end caps of the chamber. As the electrons reach the end caps they can be detected as electrical charges. The time and position where the charges are detected on the end caps makes it possible to reconstruct a 3D image of all the particle trajectories in the chamber.

The charge of a single electron is to small to be detected without being lost in noise. To amplify the charges Multi-Wire Proportional Chambers (MWPCs) are used. Each end cap of the chamber is segmented in 18 sectors and each sector consists a inner and outer readout chamber (MWPC). See figure 1.3. Thin metal strings are stretched out in parallel so that they make up a plane. There are three such planes in each readout chamber. By applying different voltages to the wire planes strong electric fields are created. These fields are much stronger than the field in the rest of the chamber. When the drifting electrons enter readout chambers they are accelerated so that they gain enough energy to ionize gas molecules. The process is repeated for the electrons that are freed and a avalanche of electrons will eventually hit the end cap. The reason for using three wire planes is to be able to close the readout chambers by applying a voltage to the gating grid (first wire plane from the center of the chamber) so that the incoming drifting electrons are absorbed and will not reach the end caps.

[2]The gas mixture is normally 90% Neon and 10% CO_2 but can be altered to adjust the properties of the detector. Recently it has been proposed to add 5% N_2 [4].

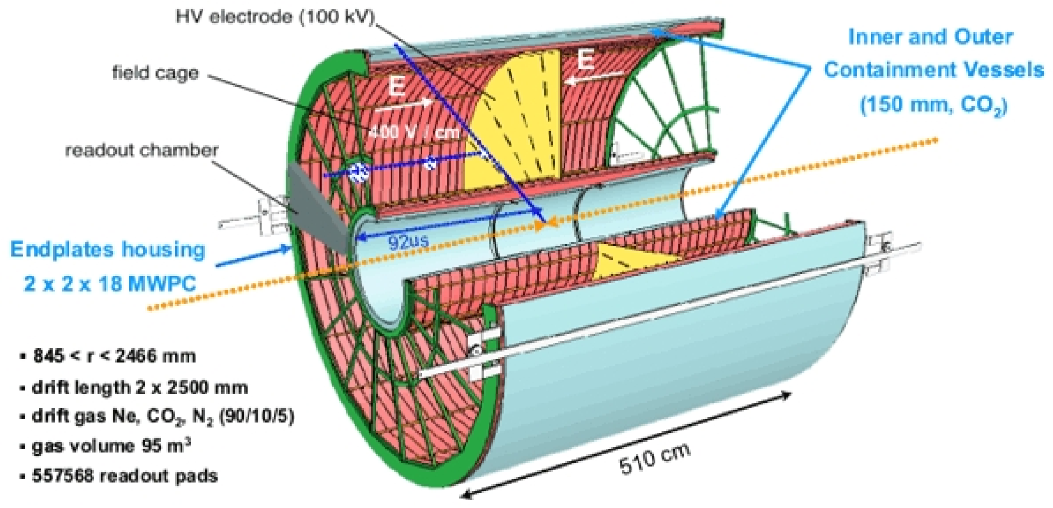

Figure 1.3: Illustratioin of the Time Projection Chamber [4].

To obtain to position where the electrons hit the end caps of the chamber, each end cap is divided into pads. In total the TPC has approximately 560000 pads where charges can be detected. Each pad or channel is sampled and readout by electronics that will be described in the next chapter.

A lot of information can be obtained about the particles that has travelled through the TPC by studying their tracks. When charged particles moves in a magnetic field they will experience a force perpendicular to their direction of movement. In addition the particles will loose speed as they ionize gas molecules. These two effects will cause the trajectories of the particles to be bent and with a sharper curve as the particle looses speed. By analyzing the tracks it is possible to determine momentum of the particles by looking at how much energy they loose per unit of travelled distance.

Chapter 2

Data acquisition for ALICE TPC

2.1 Introduction

Data acquisition is the process of sampling the signals from the detector to obtain the desired information about the physical processes that are being observed. The ALICE data acquisition system acquires huge amounts of data for later analysis. Data are captured from the detectors at rates of hundreds of GB/s while the maximum rate of the permanent storage is 1.2 GB/s. One of the challenges of the data acquisition system is to filter and select which data should be kept and stored.

There are three main components in play for the ALICE data acquisition:

Front End Electronics (FEE) Includes all electronics that sits directly connected to the detectors. It samples and digitizes the detector signals, performs some signal processing, buffer and push data to the DAQ-system.

Data Acquisition (DAQ) system This system receives the data streams from the FEE. The captured data is transmitted in optical fiber channels from the detector and up to the DAQ receivers which are located in the counting rooms.

Trigger system Initiates the data capture in the FEE makes sure that only the most interesting events are readout and pushed to the DAQ system. All events that are readout will be labeled by the trigger system. It is crucial for the coordination of the sub detector systems of ALICE and controlling the data flow.

The schematic diagram in figure 2.1 gives an overview of the system but it includes only the readout chain of the TPC detector. The readout chains of the other sub detector systems that are planned to utilize a Busy Box are identical. The Busy Box is a system whose purpose is to verify the data transfer from the FEE to the DAQ system and to prevent overflow in the buffers of the FEE. The purpose and functionality of the Busy Box and the other components will be elaborated in the following sections.

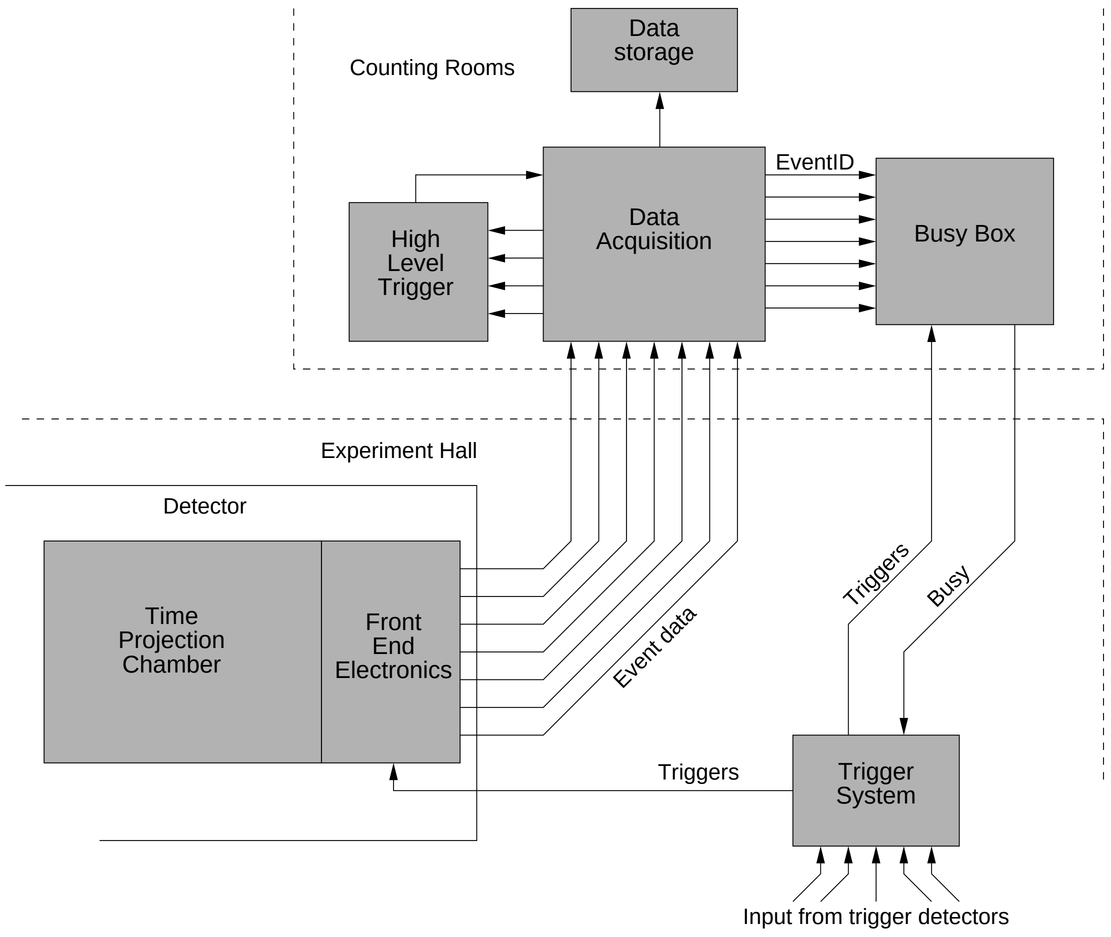

Figure 2.1: Illustration of the data flow and the trigger system for the ALICE TPC.

The counting rooms, where some of the components are located, are compartment crates that are placed in the vertical tunnel down to the experiment hall of ALICE. The counting rooms will not be exposed to ionizing radiation from the collisons as all the components that are in the experiment hall will be. The FEE which is located inside the L3 magnet must also deal with the magnetic field. No materials wih magnetic properties can be used inside the L3 magnet.

2.2 Front End Electronics (FEE) of TPC

The data acquisition starts at the FEE. The two endcaps of the TPC contains 557568 pads in total. When the trigger system confirms a collision in the detector by issuing a Level 1 trigger, the analog signals from these pads are sampled and digitized by the FEE. To capture a complete event the FEE of the TPC must sample the detector signals from the pads during the entire drift time of the TPC which is approximately 100 μs.

Each endcap is divided into 18 sectors and each sector is divided into 6 patches as shown in

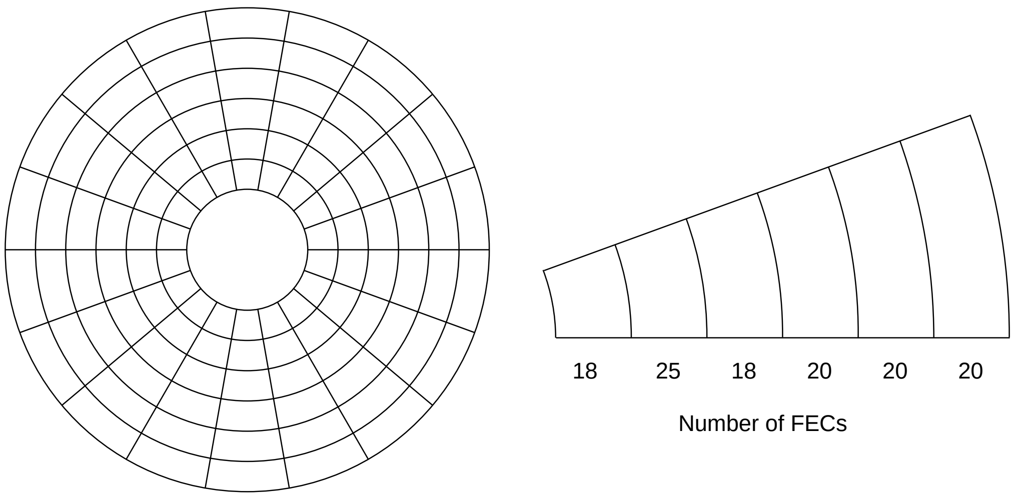

Figure 2.2: The end cap of the TPC divided into sectors and patches.

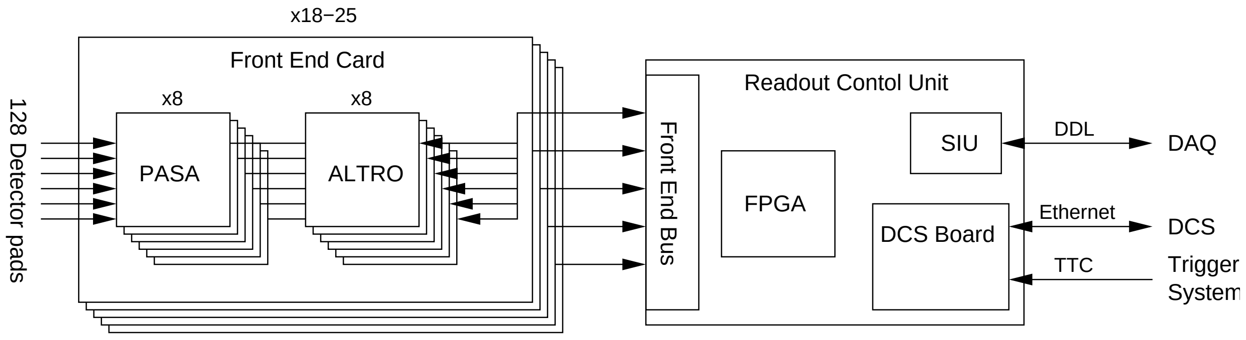

Figure 2.3: Schematic overview of the components of the Front End Electronics of the Time Projection Chamber.

figure 2.2. Figure 2.3 shows an overview of the components of the FEE that are applied to each patch. In total the TPC has 216 patches.

2.2.1 Front End Card (FEC)

The FEC is a circuit board that hosts a number of Integrated Circuits (ICs) to sample, digitize, process and buffer signals from the pads. The number of FECs used for each patch is shown in figure 2.2. As can be seen in the figure the density of FECs, and thus pads, varies and the pad density is highest in the innermost patches of the sector. This is because the track density will be higher in this region and a higher resolution is thus required to be able to reconstruct the event properly in this region.

There are two different Application Specific Integrated Circuits (ASICs) involved in the data chain. The Preamplifier and Shaper ASIC (PASA) is an analog ASIC that amplifies and shapes the raw signals from the pads before they are sampled by the ALICE TPC ReadOut (ALTRO) chip [5]. The ALTRO chip is an mixed signal ASIC with integrated 10 bit Analog to Digital Converters (ADCs) and will sample the signals at 5 or 10 MHz rate. The samples are run run through a pipeline where signal processing is performed in five steps before the data are stored in on-chip memory. The signal processing includes baseline reduction, 3rd order digital filtering and zero suppression. The ALTRO can buffer up to 4 or 8 events depending on the sample rate but independent of the data reduction. Even if the signal processing reduces the event data size to a tenth of the original size the event will still occupy one buffer slot [6].

Each PASA and ALTRO chip can operate on 16 channels independently. Each FEC hosts eight pairs of these chips and is able to capture signals from 128 channels and in total 4356 FECs are used for the ALICE TPC.

2.2.2 Readout Control Unit (RCU)

As the name implies the RCU is responsible for controlling the operation of the FECs and the readout from the FEC buffers to the DAQ system. There is one RCU for each patch on the TPC, resulting in 216 RCU boards. The RCU is based on FPGA technology and interfaces with the FECs, DAQ, Detector Control System (DCS) and the trigger system.

The RCU will decode the triggers received via the TTC system and forward the triggers to control the FECs. As outlined earlier, the FEC of the TPC should initiate data capture upon reception of a Level 1 trigger and the current buffer should be flagged for transmission to the DAQ system if the sequence ends with a Level 2 Accept message. The RCU will continuously, event-by-event, readout the flagged buffers from the ALTROs and transfer them to the DAQ system via the DDL system. For each event that is transfered to the DAQ system the RCU prepends a Common Data Header (CDH). The CDH contains information about the corresponding trigger sequence, including the event ID.

In addition to controlling the data capture and the readout to DAQ, the RCU is also responsible for configuring and monitoring the FEE. The configuration data can be sent to the RCU in one of two ways. Either via the DCS system or via the DDL system.

Access to the FECs is provided by a bus, named the Front End Bus. The Bus is composed of 40 bi-directional lines and 8 control lines [5]. The data are sent to DAQ with a Source Interface Unit (SIU) optical data adapter, described in section 2.3. The SIU is mounted on the RCU board as an add-on card. The RCU will continuously read the buffers of the ALTROs and transmit the content to DAQ.

2.3 Detector Data Link (DDL)

The DDL utilizes optical fibers to transfer data at high rates from the FEE to the DAQ system. One link contains two fiber channels and can transfer 200 $\mathrm{MB/s}$ in both directions simultaneously. ALICE uses almost 400 DDLs where 216 out of these are used for the TPC [7]. The DDL system can also be used to transfer configuration data to the FEE.

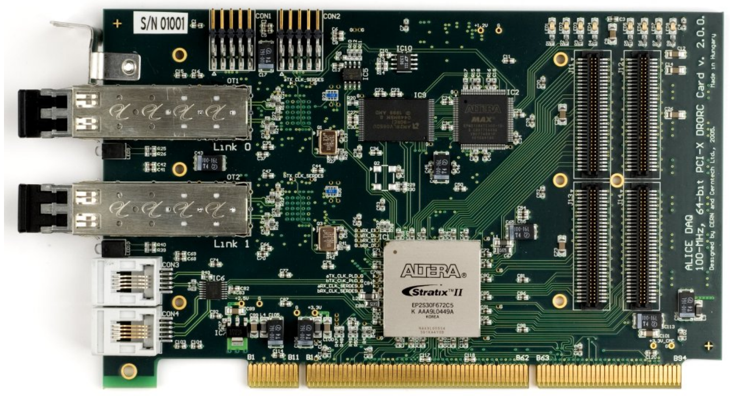

Figure 2.4: Picture of the D-RORC with two optical transceivers (SIU/DIU) and two LVDS ports.

The DDL system is composed of three parts; the optical fibers, a SIU and a Destination Interface Unit (DIU) [7]. The SIU units are mounted on the RCU boards and the DIU units are mounted on special receiver cards for the DAQ.

2.4 Data Acquisition system

On the DAQ side, special Read Out Receiver Cards (RORCs) have been designed to receive the data from DDL. The RORC includes one or two DIUs and is designed as PC card with a Peripheral Component Interconnect (PCI) interface. Up to six of these cards can be installed in a regular computer called a Local Data Concentrator (LDC). The use of commercial hardware is very desirable because it reduces the total costs of the system considerably. The RORC utilizes its own Direct Memory Access (DMA) controller to write data directly to the main memory of its host computer with a minimum load on the Central Processing Unit (CPU) [8].

The RORC exists in two different versions, the pRORC and the DAQ Read Out Receiver Card (D-RORC). The pRORC has only one DIU while the D-RORC have two. Figure 2.4 shows a picture of the D-RORC. When receiving data from a sub detector whose data is to be analyzed by the High Level Trigger (HLT), the extra DIU on the D-RORC will be used as a SIU and the incoming data will be copied and transferred to the HLT [9].

Computers with RORCs are called LDCs. They will merge event data fragments from the D-RORCs to sub-events. Through a network of computers the sub events are merged until they reach the Global Data Concentrator (GDC) where the event data will be complete. From here the data will be transferred to CERN's storage facility for permanent storage and later analysis [10].

The data transfer from the RCU to the D-RORC will go on as long as there is event data in the buffers. However the capacity of the event building network may get saturated. In this case the RCU will see the DDL link as busy and the transmission must wait even if there are more data to be shipped [11].

Table 2.1: Latencies associated with different trigger levels in the CTP.

Signal Status	L0 (μs)	L1 (μs)	L2 (μs)
Last trigger input at CTP	0.8	6.1	87.6
Trigger output at CTP	0.9	6.2	87.7
Trigger output at detector	1.2	6.5	88.0

2.5 Trigger system

The trigger system will make sure that the FEE only captures data when interesting events occurs in the detector. By issuing different triggers the system can start, abort and verify data capturing of events in the FEE. The sub detector systems of ALICE are arranged in to six clusters. The trigger system can be configured to only trigger data capture in only a subset of these clusters [9]. The CTP is the source of all the hardware triggers in ALICE. It is placed in the same racks as the Local Trigger Units (LTUs) for the detectors in the experiment hall. The LTUs will forward the triggers from the CTP to the FEE of their detector systems.

2.5.1 Central Trigger Processor (CTP)

The CTP is located in the experiment hall, outside of the L3 magnet. Several sub detector systems feeds the CTP with the required information to make decisions about which triggers should be issued to the FEE. The CTP uses three levels of triggers to control the FEE : Level 0, Level 1 and Level 2 [9].

2.5.2 Trigger sequences

The triggers are issued in sequences. For a sequence to be valid it must follow the rules outlined in the following paragraphs.

The Level 0 trigger is the fastest of the triggers (see table 2.1) and must arrive at the detectors 1.2 μs after the time of impact. The Level 0 trigger only indicates a collision in ALICE. Some detectors will start to capture data right away while others wait for the Level 1 trigger. For the TPC for example, it will take some time before the drifting electrons reach the endcaps and therefore the FEE will initiate data capture on a Level 1 trigger.

The Level 1 trigger must reach the FEE 6.5 μs after time of impact and confirms the collision that the Level 0 first indicated. The Level 1 is based on more physical parameters about the collision, for example the multiplicity of the collision. If the requirements for the Level 1 trigger is not met in the CTP, the Level 1 will not be issued. This results in a timeout for the Level 1 trigger in the detectors and the trigger sequence is aborted and hence data capture will be aborted in the detectors.

If a Level 1 trigger is issued the sequence must end with a Level 2 trigger message to be valid. The Level 2 trigger message is the last in the trigger sequence and will tell if the past-future protection of any of the detectors has been violated. The criteria for the past-future protection is programmable in the CTP, but if it is violated a Level 2 trigger message with a Reject flag will be issued. This causes the FEE to abort data capture and to overwrite the data on the next trigger sequence. Otherwise the Level 2 trigger message will be issued with an Accept flag 88 μs after the time of impact, which is about the same as the drift time for the TPC. When the FEE receives a Level 2 Accept trigger message, it will finish the data capture and mark the event data for transfer to the DAQ backend.

2.5.3 Past-future protection

The collisions at the interaction point and also unwanted interactions between the accelerated particles and gas molecules occur randomly in time. For some detectors this causes complications and this is especially the case for the TPC. The long drift time of the TPC makes it vulnerable some time before and after the collision. Any charged particle traveling through the chamber will leave a track and tracks from particles that do not originate from the collision we want to record will contaminate the event. For this reason the CTP will not trigger on collisions if there has been activity in the past and it will issue a Level 2 Reject trigger message if too much activity is detected after (future) the collision. If collisions occur after the initial triggered collision it is referred to as pile ups in the detector. The algorithms that analyze the event data is able handle a certain number of pile ups and this a programmable parameter in the CTP.

2.5.4 Timing, Trigger and Control (TTC)

The TTC is a system of optical fibres and electronics used to distribute triggers and a reference clock signal to all detectors at low latencies. The source of the optical signal is the LTU and from there it will be distributed to all of the FEE through a network of optical fibers and splitters.

Two trigger channels and the global LHC clock are distributed by the TTC system. The channels are modulated together with the clock in the optical signal. Channel A is used to broadcast trigger pulses for Level 0 and Level 1 while channel B is used to broadcast trigger messages for Level 1 and Level 2 triggers. The LHC global clock is synchronized to the nominal bunchcrossing rate of the LHC and runs at 40.08 MHz. This clock is used as the reference clock for all the digital electronics in the ALICE experiment.

2.5.5 Event ID

When the trigger system issues a trigger sequence the captured event data must be given an ID so that event fragments from all sub detectors can be identified and combined to a complete event. The event ID will be generated by the trigger system and broadcasted along with the triggers as part of the Level 2 trigger messages on the TTC system.

The event ID is 36 bits in length where the first 24 bits are the orbit ID and the 12 remaining bits are the bunchcrossing ID. The bunchcrossing ID is a count that is incremented at the nominal bunchcrossing rate of the LHC. It is reset when the bunches have made a full round trip in the accelerator ring. When the bunchcrossing count is reset the orbit count is incremented. When two bunches cross each other at the interaction point they will have to travel the same length

Table 2.2: Number of D-RORC cards per detector.

Detector	# D-RORCs	Panel height
TPC	216	5U
PHOS	20	1U
EMCal	24	1U
FMD	3	1U

before they meet again at the same point. Thus the bunchcrossing ID will be the same each time they meet but the orbit ID will be incremented.

2.6 Busy generation

Even with the high capacity of the DAQ system, the FEE of some detectors will sometimes produce data faster than can be received. This will eventually lead to a situation where the buffers of the FEE are full when new triggers are issued by the CTP. Hence the CTP needs to know in advance when the buffers are full and the whole detector system must be considered to be busy.

The topic of this work is the Busy Box, which is a device that is able to determine the busy state of a detector system by communicating with the trigger and DAQ systems. It will verify that each event that has been triggered is successfully transfered to the DAQ system and thus it is able to keep track of how many events are unaccounted for. The number of unaccounted events will then translate to the number of used buffers in the FEE and lets the Busy Box determine the busy state of the detector system.

The Busy Box is a independent system that will be placed in the counting rooms near the LDCs for the detector system it is monitoring. This ensures short communication cables to the DAQ system and easy access if problems should arise. Also, by placing it in the counting rooms rather than the experiment hall, the device does not need to be radiation tolerant. Using a dedicated device for the busy generation gives better modularity for the overall system so that the development of the different systems can be done more independently.

To avoid being a bottleneck for the readout rate of the detector systems the Busy Box system must be able to verify events at a higher rate then the maximum readout rate of the detectors. These rates are not always clear but a verification rate of at least 1 kHz should be well within the requirements for the detector systems where the Busy Box will do the busy generation. The Busy Box will be used for the TPC, PHOS, EMCal and FMB sub detector systems. Table 2.2 lists the detectors and how many D-RORCs are used in the DAQ system for each detector.

2.7 About this work

This report describes the work that has done in developing the ALICE Busy Box which is a component in the data acquisition system of ALICE. The development of the Busy Box includes the work of several students and employees of the microelectronics group and the nuclear physics group at the Institute of Physics and Technology at the University of Bergen. Also personnel at CERN have contributed with specification requirements and integration into the ALICE data acquisition systems.

At the time I was included in the project the overall structure and functionality had been outlined. The circuit board had been designed and produced and the components was currently being mounted. My first involvement in the project was to cooperate with the board designer, Anders Rossebø, in the early stages of planning the firmware implementation of the required functionality. Anders was at this time at the very end of his masters thesis and when he graduated he was hired to finish the hardware by integrating the boards into rack cases and including power supply units.

Meanwhile, I was given the responsibility of the further firmware development. This work can be summarized in three phases.

- Testing the newly produced hardware and debug programming interfaces. Some minor design flaws were identified on the board that affected the functionality.
- Development of a robust serial communication with the D-RORCs. This work has included investigations of serial communication protocols, implementation and testing, and debugging with hardware test setups, both at CERN and in Bergen. This phase required several iterations before a satisfactory solution was achieved.
- Designing and implementing the complete firmware. At first this included an interface for the software on the DCS board to communicate with all the D-RORCs over the serial links. Later on, all the basic operation has been implemented in firmware to meet the timing requirements of the system. This work has also included defining the basic procedures both on the D-RORC and the Busy Box that are required for the Busy Box to set the busy signal correctly.

The operation of the firmware will be controlled and monitored by software running on the DCS board. During this work I have cooperated closely with Are Stangeland which is responsible for developing the software for the Busy Box. Are has also contributed a great deal in clarifying and defining the system requirements.

I have done all of the firmware development for the Busy Box except for the Trigger Receiver Module which has been designed by Johan Alme (see figure 5.1).

During the development I have had three trips to CERN. On my first trip I spent two weeks working on the cabling on the TPC. The other trips have been for for testing and integration of the Busy Box.

Prior to the Busy Box project I have been involved in two other projects. The first was the Cosmic Ray Telescope where I was responsible for the firmware for a FPGA in the readout electronics. The second was the TriggerOR board for the PHOS detector of ALICE. This board and firmware for the FPGA on it was designed by four students from Bergen University College. However the board had not been produced when they were finished. My task was to simulate and verify the firmware, test the hardware and eventually integrate the board at CERN. However the PHOS project was delayed and thus the integration of the TriggerOR was also delayed, so I switched to the Busy Box project in December 2006.

In addition to the work presented in this report I have also written an article about the Busy Box in close cooperation with Are Stangeland. The article was published in the FPGAworld 2007 proceedings and I was present at FPGAworld conference and did a presentation based on the article.

Chapter 3

Busy Box

This chapter describes the hardware. It includes some discussions why this hardware was chosen and give an overview of how the system is intended to work. A lot of this information is taken from [12].

3.1 Busy Box hardware

The first step in designing the Busy Box is to define the requirement specifications and make sure that all aspects of the design are covered. An analysis of the system described in chapter 2 reveals the following requirements:

- The Busy Box needs an interface to the TTC system so that it is able to receive the same trigger information as the FEE. This implies having an optical receiver and electronics to decode the channels integrated in the system.

- A reliable two-way communication link with all of the D-RORCs of the detector system it is serving is required to retrieve the event IDs from the D-RORCs. For the TPC detector system, which is the largest concerning number of DDLs, this implies at least 216 communication links. The D-RORC has been outfitted with LVDS serial transceivers and RJ-45 ports for communications with the Busy Box and thus the Busy Box needs at least 216 RJ-45 ports and LVDS I/O drivers. The Busy Box will be placed in the counting rooms next to the LDC for the corresponding detector and thus the maximum cable length will not exceed 15 meters.

- The Busy Box needs a line to the LTU to provide it with the busy-signal. The LTU is located in the experiment hall while the Busy Box is in the counting rooms 50 to 100 meters above.

- An interface to the DCS is required so that the operation of the Busy Box can be monitored, controlled and configured by the DCS system.

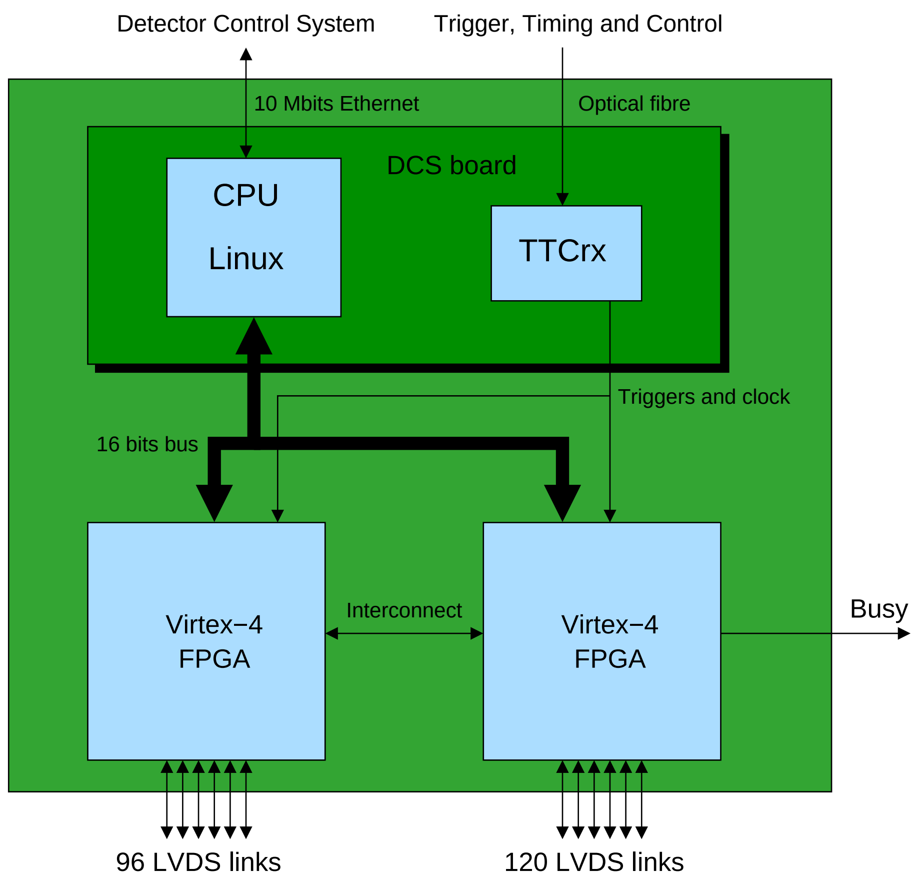

Figure 3.1: Busy Box system overview.

- Sufficient processing power is required to compare and keep track of all the event IDs received from the D-RORCs and the trigger system within the time window of a trigger sequence. This processing must be fast enough so that the Busy Box does not become the bottleneck of the entire system.

3.2 System overview

Some components of the design are predetermined by the systems it has to interface with. The design is also influenced by the experience gained with the development of the RCU board.

As with the RCU, the system is based on FPGAs to implement the required digital electronics and it has a DCS addon board. To obtain the high number of I/O drivers required two FPGAs with LVDS capable I/O pads are included. The FPGAs will also provide all the logic resources required to implement serial transceivers, decoding of the signals from the TTCrx and any possible processing required. The DCS board brings many other features to the system. It includes a miniature computer with Linux OS, an optical receiver and a TTCrx chip.

Figure 3.1 shows a simplified block diagram of the Busy Box system. The signals from the

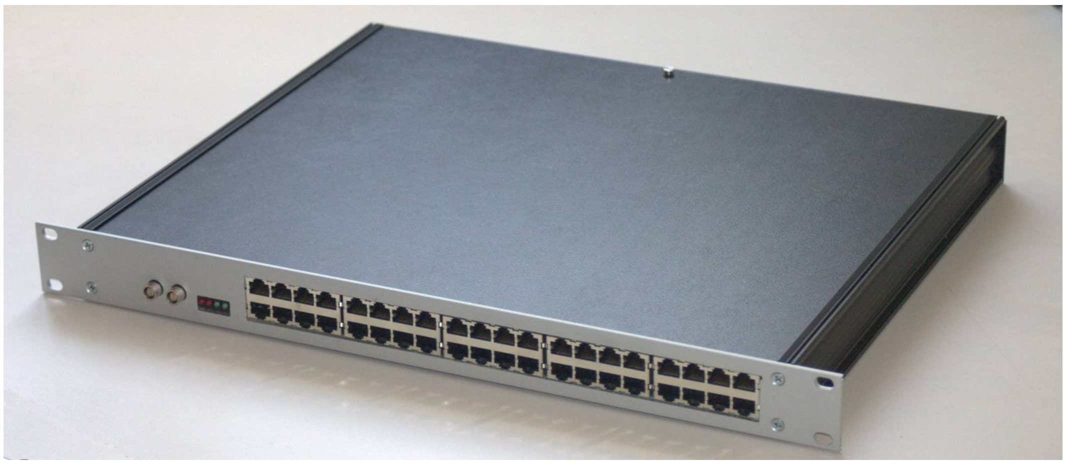

Figure 3.2: Picture of the Busy Box in a one unit 19” rack case. (Photo: C. Soos)

TTCrx chip are connected to the FPGA where logic to decode the serial channel A and B can be realized. The clock provided by the TTC system will be used as clock source for all digital electronics, including the FPGAs.

The two FPGAs will work independently and the FPGA that drives the busy signal will get the busy state from the other FPGA via dedicated interconnect signals between the FPGAs.

The serial communication with all the D-RORCs will be handled by the programmable logic in the FPGAs. Since some of the detectors where the Busy Box is will be used utilize only a few DDLs the second FPGA can be left out when mounting components to the circuit board. For this reason the number of LVDS I/Os on the first FPGA have been maximized within the constraints of the I/O bank configurations on the FPGA. More details on this is given in section 3.5.1. As a result the first FPGA is connected to 120 of the RJ-45 ports connected while the second FPGA is connected to the 96 remaining.

The picture in figure 3.2 shows a Busy Box built in one unit rack case. From right to left of the front panel are LEMO connectors for the BUSY-signal, four Light Emitting Diode (LED) indicators and the 40 RJ-45 connectors. The picture in figure 3.3 shows a Busy Box with the lid off so the internal components are visible. A large PCB is needed to host all the components for the system. The board has been designed so that one of the two FPGA can be left unmounted to increase the scalability of the system. The DCS board is mounted on top of the Busy board to the left. The white connectors on each side of the FPGAs are used to connect add-on boards called mezzanine cards with additional RJ-45 connectors. To fit enough RJ-45 connectors in the front panel of the box, four mezzanine cards must be connected, making the box five units in height of standard 19” racks. The Busy Box will be placed in racks in the counting rooms near the LDCs.

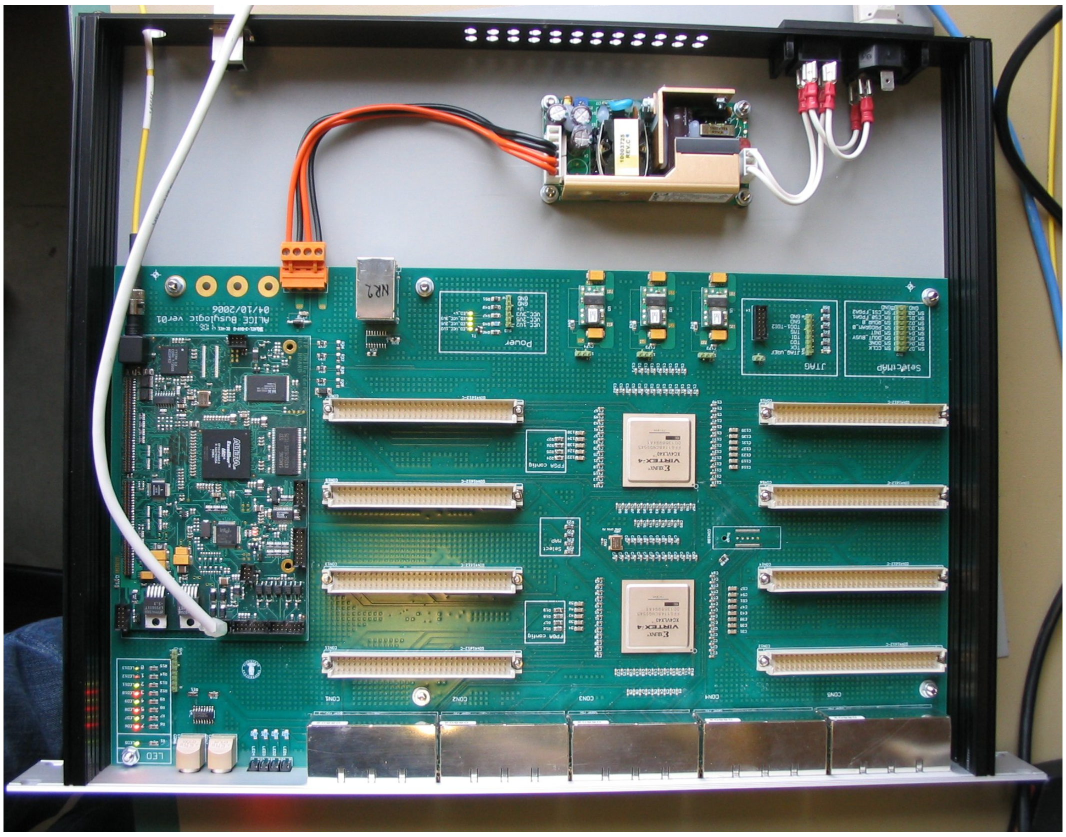

Figure 3.3: Picture of the Busy Box with the lid off to show the internal components. (Photo: A. Rossebø)

3.3 DCS board

The DCS board is one of the add-on boards used on the RCU and was developed for the FEE of ALICE to be part of the DCS system. It is mainly composed of an Altera EPXA1 FPGA (includes a 32 bit hard core ARM processor), 8 MB of flash ROM, 32 MB Synchronous Dynamic RAM (SDRAM) and an Ethernet transceiver. With these components it is able to run a lightweight version of Linux Operating System (OS) and implement the TCP/IP network protocol. Each DCS board runs a FEE server that interfaces via a parallel data bus with the system it is integrated in and provides services to connected clients such as housekeeping data about the system.

The DCS board on the RCU also has an interface to the programming logic in the FPGA and is thus able to load configuration data into the FPGA. This is feature that is very useful for the RCU boards as they will be physically unaccessible and in an radiation hazardous environment once the LHC is operational.

Another important feature of the DCS board is that it has a TTCrx chip and an optical receiver mounted. The optical receiver will convert the optical signal received from the LTU into an electrical signal. The TTCrx chip will demodulate the this signal and output the three electrical signals; the LHC bunchcrossing clock, channel A and channel B.

The DCS board was included in the design because it brings a lot of features that are required and some that will come in handy for the Busy Box. It will be mounted on top of the Busy Box board with two 70 pin connectors. The connections are used for signals and for distributing power from the Busy Box board to the DCS board.

3.3.1 Trigger message decoding

The signals from the TTCrx chip are routed to both the FPGAs on the Busy Box board through the board-to-board connectors. A firmware module has been designed to decode these signals and make all trigger information available in the Common Data Header (CDH)-format [13].

The serial channels A and B are synchronous to the BC clock so no clock recovery is required. Channel A will contain the Level 0 and Level 1 trigger pulses while channel B will contain complete trigger messages. A trigger message may be a broadcast message or an addressed message. Broadcast messages include calibration pre-pulse, event count reset and bunch count reset. Addressed messages include Level 1 Accept message, Level 2 Accept message, Level 2 Reject message and possibly a Region of Interest message if it is supported by the LTU. All trigger messages are hamming coded for better transmission error tolerance.

3.4 LVDS driver for BUSY signal

The Busy Box is placed in the counting rooms high above the experiment hall where the rest of the trigger system resides. The cable from the Busy Box to the LTU might be as long as 100 meters. To guarantee the required signal integrity all the way down to the experiment hall the BUSY-signal will be transmitted in a specific type of coaxial cable and driven by an LVDS driver

(SN65LVDM31) which both are certified by CERN [12] [14]. The cable is connected to the Busy Box and the LTU with standard LEMO connectors.

3.5 Virtex-4 FPGA

The FPGA that was found to be most appropriate for this application at the time of design was the Virtex-4 LX-40 in the ff1148 package from Xilinx. Of the 1148 pins 640 are user programmable I/Os that support many I/O standards including LVDS 2.5 which will be used to communicate with the D-RORCs.

The FPGA is produced on a 90 nm copper CMOS process and the core voltage is 1.2 V. This makes it possible to implement designs that run on clock speeds up to 500 MHz. The logic resources are arranged in a 128x36 array of Configurable Logic Blocks (CLBs). Each CLB includes four slices where each slice contains two Logic Cells/LookUp Tables (LUTs). In addition the FPGA includes 64 XtremeDSP slices with fast 18x18 multipliers. For memory the FPGA includes 96 dualport block RAMs. Each block RAM can store up to 18 Kbits depending on the configuration.

To provide flexible clocking and synchronization Virtex-4 includes 8 Digital Clock Managers (DCMs) and 4 Phase Matched Clock Dividers (PMCDs). The DCMs can do clock deskew, phase shifting and frequency synthesis.

3.5.1 Virtex-4 I/O banks

The I/O pins are distributed over 13 banks where there are constraints for which I/O standards are supported on the same bank depending on the bank supply voltage. One of the I/O banks is reserved for the programming interfaces. The I/O banks that will be used to communicate with the D-RORCs must support LVDS 2.5 and hence the supply voltage of these are 2.5 V. These banks can then not include single ended signals which, for this system, require 3.3 V supply voltage.

3.6 Virtex-4 Programming Interfaces

Virtex-4 supports different programming interfaces that allows the user to load his/her design in to the device. For the Busy Box system the FPGAs will primarily be programmed from the DCS board. This can be done through the SelectMAP interfaces of the FPGAs. The JTAG interface is also made available. Both interfaces is discussed in the following sections.

3.6.1 SelectMAP interface

The SelectMAP is an interface to access the configuration memory of the Xilinx FPGAs [15]. Virtex-4 supports SelectMAP interfaces with 32 or 8 bit parallel or serial data transmission and the FPGA can be set to be master or slave. The master modes can be used to let the FPGAs load the contents of a Programmable Read Only Memory (PROM) into its configuration memory upon the power on cycle. In slave mode the FPGAs will wait for another device to drive the clock signal and load data into the configuration memory. It is possible to let the SelectMAP data lines become user I/Os after configuration but then the device must be reset before the SelectMAP interface is available again. To make the SelectMAP interfaces available even after the device has been programmed it was decided to use dedicated lines for SelectMAP and the data bus.

To make room for the SelectMAP data lines in the board-to-board connectors the system bus was reduced to 16 data lines (the RCU uses 32). The SelectMAP interface with 8 data lines can then be used. Since the FPGAs will be programmed from the DCS board they are set to slave mode.

The FPGA will sample the logic value of three mode pins during the power up cycle to determine which interface is used. The mode pins for the FPGAs are pulled to logic high or low by the Busy Box board and the configuration can be altered by mounting resistors between different nodes on the board. For the selected interface mode (SelectMAP 8 bit slave mode) pins M2 and M1 are pulled to logic high and M0 is pulled to logic low.

Linux kernel device drivers have been developed so that the SelectMAP interface accessible as a device in the Linux OS [16]. The programming bit file generated by the Xilinx tools can then redirected to the device driver which will take care of writing the configuration data to the device.

3.6.2 JTAG interface

The FPGAs can also be programmed via a JTAG interface. JTAG is an acronym for Joint Test Action Group after the committee that was responsible for developing the boundary scan technology, which for that reason is often referred to as Joint Test Action Group (JTAG). It is standardized in the Institute of Electrical and Electronics Engineers (IEEE) 1149 standard. Its initial purpose was to test both Printed Circuit Boards (PCBs) and chips at board level. Later on it has proven useful other functions as well. The JTAG interface for Virtex-4 can be used to load the configuration data to the device and access internal logic for many other purposes, for example the use of integrated logic analyzers.

Figure 3.4 shows the connection diagram for the two Virtex-4 FPGAs on the busy board. When only one FPGA is mounted on the board, the jumper needs to be applied to bypass the missing FPGA for the JTAG chain to be working. The JTAG interface can be activated at any time and will override the mode selected by the mode pins.

To access the FPGAs a JTAG programming device is connected to the JTAG connector on the board. The programming device is often data adapter between the JTAG and other standard interfaces for example the parallel or USB port of a regular computer.

3.7 Busy Box circuit board

The circuit board for the Busy Box system hosts all the components and provides interconnect and power distribution. This also includes signal lanes between the FPGAs and all the RJ-45

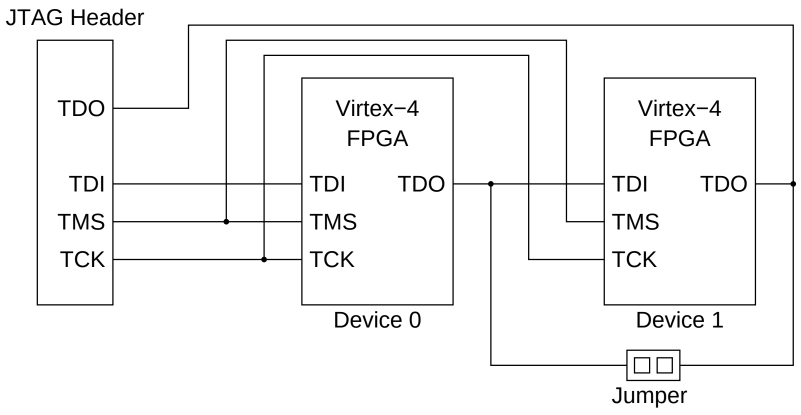

Figure 3.4: JTAG connection scheme for the Virtex-4 devices in the Busy Box

connectors and the connectors for the RJ-45 extension cards. See the picture in figure 3.3.

3.7.1 Power supply

The board receives power from an external 5 Vpower supply. Three voltage regulators of type PTH05000W from Texas Instruments are used to generate three different supply voltages:

1.2 V This is the core voltage of the FPGAs

2.5 V This voltage is used to supply some of the auxiliary components in the FPGAs and it is used as supply voltage to the I/O banks where the LVDS standard is used.

3.3 V This voltage supplies the I/O banks that contains the pins for the configuration interface for the FPGAs and to the banks that are used for single-ended signals.

The voltage regulators can be enabled and disabled individually with jumpers. If no jumpers are applied the regulators are always on, by connecting two of the three pins the regulators can be set to always off and to be controlled by signals from the DCS board. See [12]

Chapter 4

Serial Communication with the D-RORCs

4.1 Physical Layer

The physical layer of the communication channel between the Busy box and the D-RORCs is done with LVDS in Twisted Pair (TP) cables with RJ-45 connectors. The TP cables will not exceed 15 meters in length and provide good signal integrity by cancelling out electromagnetic interference from external sources and crosstalk from neighboring wires. This makes the TP cables ideal for LVDS. Figure 4.1 shows a standard LVDS application. The transmitter drives 3.5 mA through the *current loop*. At the receiver end, the signal is terminated by a 100 Ω resistor which results in a voltage swing of 350 mV. The LVDS receiver will sense the voltage drop over this resistor and drive a single-ended signal with corresponding valid logic levels. The common mode voltage for the LVDS lines is 1.25 V.

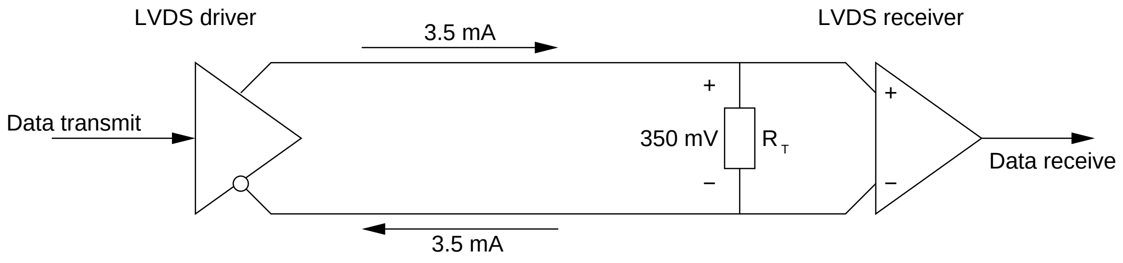

Figure 4.1: Standard point to point LVDS Application.

The Busy Box uses the built-in IO Blocks in the Virtex-4 FPGA for LVDS I/O. Input and output buffers are instantiated in the source code with with I/O standard specified as LVDS_25. For the input buffers the DIFF_TERM attribute is set to true which enables the internal differential resistor in the IO Block [17]. Figure 4.2 shows how the LVDS pairs are connected in the RJ-45 connectors. With this connection scheme standard CAT-5 TP cables can be used [12].

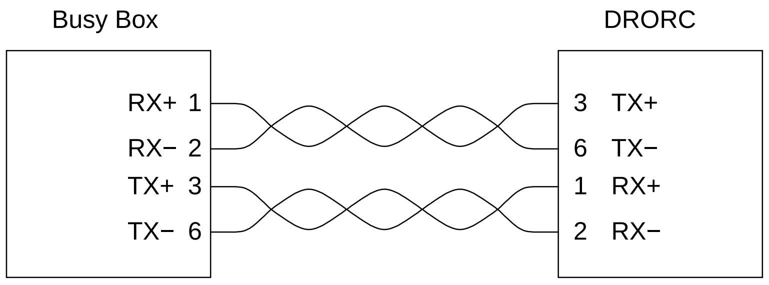

Figure 4.2: RJ-45 pin connection scheme for Busy Box and D-RORC

4.2 Serial bit-encoding schemes

To reliably transmit data over a single signal line the data bits must be encoded in the signal in a way such the receiver can identify each bit in the stream unambiguously. Bit-encoding schemes is a trade-off between reliability, efficiency and implementation cost/complexity.

One of the challenges is to synchronize the receiver with the incoming bit-stream. Two devices that run on the same nominal clock frequency but do not share the same clock source are defined as *plesiochronous* see for example [18]. The performance of clock sources, for example oscillation crystals, may vary with attributes such as temperature and process variations. For this reason the phase alignment will drift over time as one clock will be slightly faster or slower than the other.

One way to synchronize the receiver logic with the bit-stream is to encode both clock and data in the signal. For example Manchester coding, where a high-to-low transition indicates a logic 1 and a low-to-high transition indicates a logic 0. The clock can then be recovered by using a Digital Phase Locked Loop (PLL) to lock on to these transitions. The drawbacks are that this requires twice the bandwidth compared to Non-Return-to-Zero (NRZ) encoding.

The NRZ coding scheme does not include clock information and hence the nominal bit-rate must be known and fixed. Each bit will be transmitted by holding the line high or low for a bit-period. If the transmitted data contains a lot of consecutive zeros or ones, there will not be any transitions in the signal. If the phase alignment of two communicating devices exceeds 180 degrees in either direction during this time, it is impossible to know how many bit-periods have passed. Therefore, NRZ coding is not suitable for raw bit-streams.

One approach to use NRZ is to split the data into words of fixed length and transmit it in frames. A frame can consist of the data word proceeded by a start-bit and ended by a stop-bit. The incoming signal is oversampled so that rising and falling edges can be detected by comparing two consecutive samples. Typically, the receiver will detect the first edge of the start-bit and from then on, by counting samples, select the samples that are believed to be in the middle of a bit-period to determine the value of this bit. The length of the transmitted word must be relatively short so that phase error will not be given time to accumulate. The phase error tolerance is typically a function of the frame-length. This method synchronizes at the start of each frame while the Manchester-coding synchronize on each bit.

An identical technique was implemented and tested on the Busy Box hardware. The frame included a 32 bit data word, two start-bits, one parity-bit and one stop-bit, in total 36 bits. The communication logic ran on 200 MHz and a bit-period was 4 clock cycles resulting in a 50 MHz bit-rate. A problem with this method became apparent when testing it in hardware; Unconnected ports have floating inputs that picks up all kinds of electromagnetic noise. The receiver triggered on this noise and produced a lot of garbage data. All incoming messages are stored in the same memory in the Busy Box. When the memory is full, the oldest messages will automatically be overwritten when new messages arrive. The garbage messages that are produced will quickly fill the memory and make the communication unusable. Making a stricter start condition that is less probable to occur in noise reduces the garbage data produced, but at some point real data transmissions also gets rejected. The efforts to find a better start condition did not lead to satisfactory compromise between the amount of garbage data produced and data loss.

To improve the performance of the serial receiver the samples are shifted into a shift register long enough to contain a complete frame. This makes it possible to evaluate both start and stop conditions before capturing data. In addition, the logic value of a bit is determined by running all samples in the bit-period through a majority gate. The idea is that bad samples will be outvoted. When using majority gates, it is desirable to to use an odd number of samples as inputs so that no value (one or zero) has to be given precedence. Implementation of this serial receiver is discussed in next chapter.

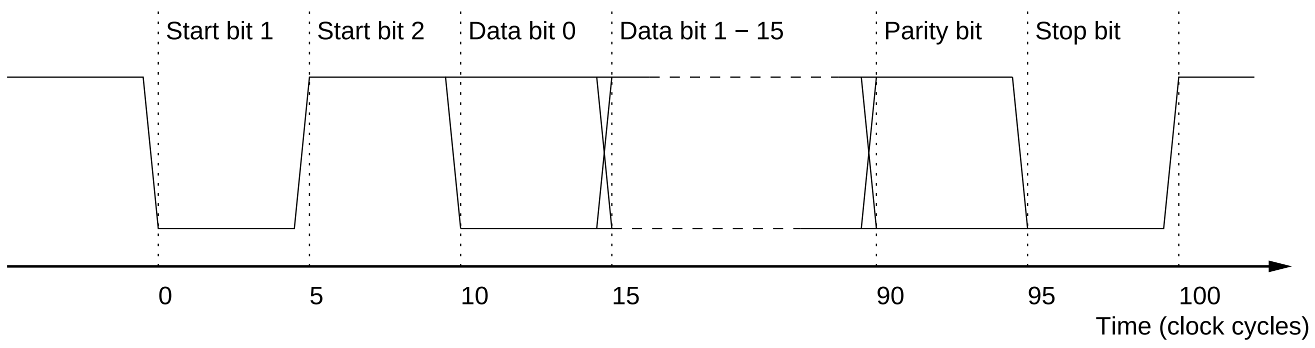

Figure 4.3: Illustration of serial data transmission

Figure 4.3 show an illustration of the bit encoding. The properties are summarized below.

- Serial data transmission and NRZ encoding.
- 5 clock-cycles/periods per bit. (40 MHz baudrate/bitrate) [1]
- Majority function used on all samples in a bit to determine the value of this bit.
- 2 start bits, 16 data bits, 1 parity bit and 1 stop per transmitted word.
- Start bit 1 is low, start bit 2 is high, stop bit is low.
- Idle lines should be pulled logic high by transmitter.

[1]Number of clock cycles per bit is to be decided later on when more tests have been performed.

4.3 Alternative solutions

Other solutions for serial data transmission have been investigated. The current implementation of the serial receiver uses too much logic resources and an alternative is desired. A interesting serial receiver is the digital phase follower. It is discussed in several articles around the internet and an implementation of it in Xilinx FPGAs in presented in [19].

In the implementation of the digital phase follower the logic of the serial receiver runs on the same nominal clock frequency as the bit stream. To obtain oversampling of the signal a second clock which is phase shifted 90 degrees to the original clock. The signal will be sampled by registers in both clock domain and by also sampling the signal with registers that triggers on the falling edges of the clocks 4x oversampling is achieved.

If there is a transition in the serial signal the receiver will see where it is in the four samples and compare it to where the transition was last time. By doing this the receiver is able to follow the phase of the incoming bit stream relative to the system clock. If the phase of the incoming bit stream is shifted more than 180 degrees in either direction the receiver will compensate by producing either none or two bits of data this clock cycle.

The digital phase follower requires transitions to happen often to be able to follow the phase correctly. This is not guaranteed with a raw bit stream. A technique that guarantees a transition density is the 8b/10b encoding.

The 8b/10b encoding maps 256 unique 8 bit symbols to a subset of 10 bit symbols. By only using a subset of the 10 bit symbols each of these symbols will consist of either 5 ones and 5 zeros, 6 ones and 4 zeros, or 4 ones and 6 zeros. All of the utilized 10 bit symbols have a disparity of 0 or $\pm$ 2. The disparity is the number of ones minus the number of zeros in a symbol. The 8 bit symbols that are mapped to 10 symbols with a disparity of + 2 are also mapped to the inverted 10 bit symbol with a disparity of - 2. The encoder will always pick the 10 symbol that maintains the running disparity to $\pm$ 1. If the running disparity is + 1 then selecting a 10 bit symbol with a disparity of - 2 changes the running disparity to - 1 and so on. A 10 bit symbol with a disparity of 0 will not change the running disparity. This way the DC balance of the signal is maintained and it also guarantees that there will never be more than 5 consecutive zeros or ones which guarantees the transition density.

If the 10b/8b bit encoding was implemented in the Busy Box it would be easy to detect unconnected ports by checking the running disparity or valid 10 bit symbols. However the encoders and decoders must be implemented on both sides and there is not available resources to implement a decoder for each channel in the Busy Box. A possible solution is to implement several decoders so that the undecoded messages can be gathered and decoded at one central point. At the moment this is considered not to be worth the effort but it is a possible improvement of the communication.

4.4 Basic operation

When the D-RORCs has received an event data fragment, the event ID is extracted from the Common Data Header (CDH) and stored in a FIFO queue. To verify that all data fragments from an event has been successfully transfered to the DAQ-system, the Busy Box must make sure that all D-RORCs have this event ID in its queue. The Busy Box will maintain a queue of event IDs received from the TTC link. If there are no errors then the event IDs in the queues in the D-RORCs and in the Busy Box will match. Hence a principal procedure for event verification can be defined:

1. Start out with the first event ID in the queue in th Busy Box.
2. Compare this event ID with the event ID in each D-RORC.
3. When the event ID has been matched in all D-RORCs the event has been verified.
4. Pop one event ID from the queues in the Busy Box and in all the D-RORCs and start over again.

For this procedure to work it is critical that the queues are kept synchronized and that they contain the same event IDs in the same order. The latter criteria is obtained by making sure that the Busy Box and the FEE have the same way of decoding and interpreting the trigger messages issued on the TTC-system. If the queue in one of the D-RORCs is one element ahead or behind the queue in the Busy Box the event ID comparison will always result in a mismatch and some high level error handling will be required to resolve the situation. A strategy for keeping the queues synchronized even if transmission errors occur will be discussed in section 4.5.

The comparison of event IDs can be done either in the Busy Box or in each individual D-RORC. Both cases were evaluated and the conclusion was that it is better to do the comparison in the Busy Box for several reasons. First, it was considered a good idea to keep all the processing in one device. Especially in the development phase it will make debugging a lot easier. Also, it makes sense if the firmware should need an update in the future then it may only be necessary to update the Busy Boxes and not all the D-RORCs. Second, if an error situation occurs all the information that is needed to resolve it is kept in the Busy Box. Consequently the protocol is based on that the Busy Box receives event IDs from the D-RORCs, does the matching and decides when the D-RORCs should pop one element in the their queues.

The time from the Level 2 Accept trigger is issued and until the event ID is available in the queues of the D-RORCs will vary and will be different for the individual D-RORCs as the sizes of the data fragments vary. Also, there might be data fragments from previous triggers that has to be transfered first. As a result the Busy Box will have to send several requests before all D-RORCs have been verified.

Figure 4.4 shows diagram of the communication between the LTU, Busy Box, two D-RORCs and the RCU boards of the FEE. The process of event verification is started when the Busy Box receives a trigger sequence from the LTU that ends with a Level 2 Accept trigger. The same trigger sequence will be received by the RCU boards as shown on figure 4.4. The RCU boards will command the FECs to start sampling data when the Level 1 trigger is received[2]. At T1 the Level 2 Accept trigger is issued which means that the event has been accepted and the RCUs

[2]This is true for the TPC. Other sub detectors will start data sampling at other criteria.

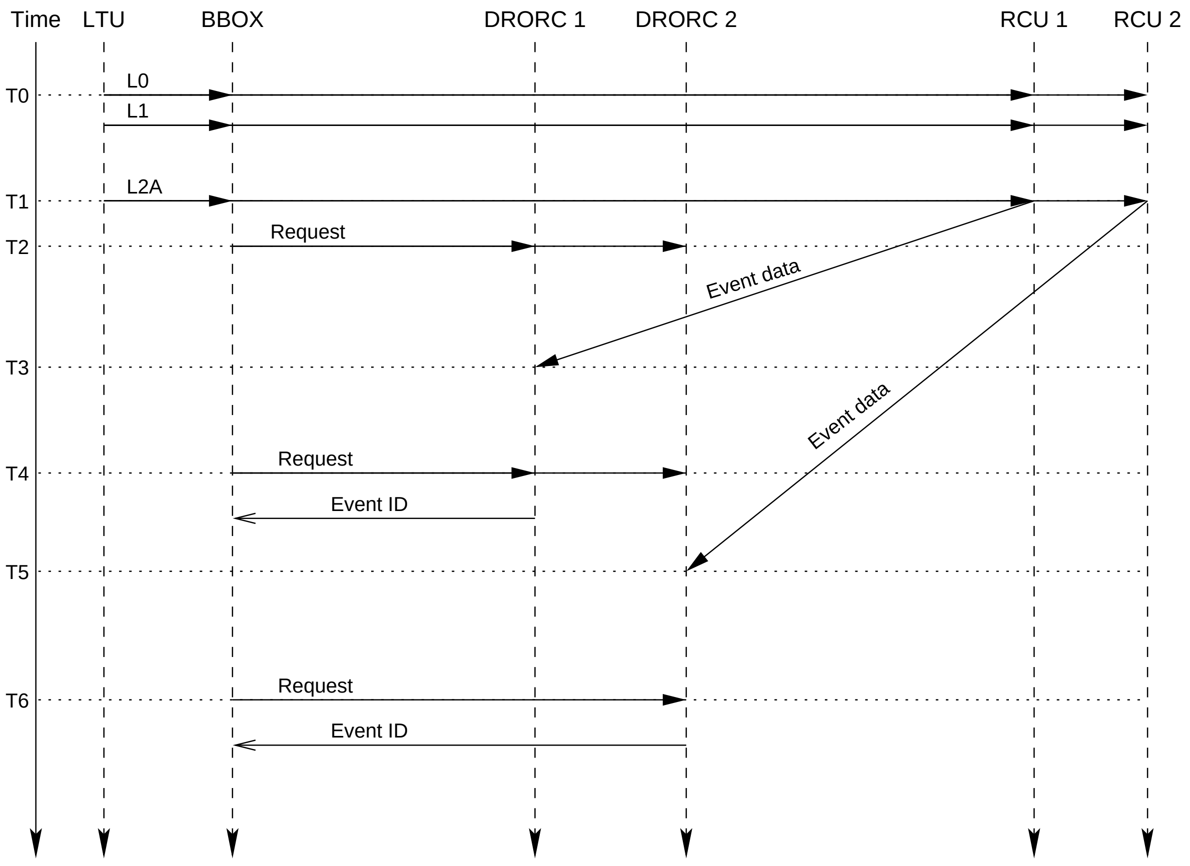

Figure 4.4: Message sequence diagram for an event verification with two sets of D-RORC and RCU boards.

will mark the current buffer for transmission to the D-RORCs. The Busy Box will also see the Level 2 Accept trigger and start requesting event IDs at T2. Since none of the D-RORCs have received this event yet they don't reply. At T3 RCU 1 has finished transferring the event data fragment and D-RORC 1 has extracted the event ID and put it in its local queue. The Busy Box will periodically send new requests by a programmable time interval. At T4 the Busy Box sends a new request and D-RORC 1 will reply with the event ID. At T5 the other event data fragment has been transferred to D-RORC 2. At T6 the Busy sends requests to D-RORCs that it has not yet received a correct event ID from. D-RORC 2 replies and the event have been verified if the event IDs match and these are the only two DDL channels monitored by the Busy Box.

If a new trigger sequence is issued before an event has been verified the event IDs from the new trigger sequence will stack up in the queues and be verified in turn. The D-RORCs can be programmed to pop an event ID from their queue whenever they reply to a request from the Busy Box. Since the Busy Box will not send new request to D-RORCs that have replied with a correct event ID before the next event ID is requested, the queues will stay synchronized. However this requires 100 % integrity for the communication and that the event IDs always match. If a D-RORC replies to a request from the Busy Box and the event ID does not match or the message is lost in transmission, the Busy Box will send a new request. Since the correct event ID has already been thrown away (popped from the queue), the D-RORC will never reply with a correct event ID. The result is that the D-RORC will continue to pop event IDs as the Busy Box keeps sending requests. This will continue until the Busy Box asserts busy-signal because the buffers of the FEEs are believed to be full. The next section will discuss a strategy to prevent this error situation even if transmission errors occur.

4.5 Transmission error handling

Testing and experience suggests that it is likely that transmission errors will occur during a LHC run. A LHC run may last for several hours and the planned average rate of of events for ALICE is 200 Hz. This quickly sums up to millions of events per run. Taking into account the number of DDL links and the number of messages that has to be transmitted for the verification of one event for one DDL link, the total number of messages transmitted between the D-RORC and Busy Box can reach a billion.

There are several well documented approaches to making a system more tolerant to transmission errors. However the LVDS communication system of the Busy Box is somewhat unique. Even though there exists a two-way, full duplex communication channel to each D-RORCs, the receiving and transmitting parts of each channel has been implemented in two different sub modules. Transmission error handling protocols like Automatic Repeat reQuest or similar are based on acknowledgements from the receiver to the transmitter that a frame has been successfully received and a timeout in the transmitter after sending a frame before resending if an acknowledgement has not been received. Such an protocol would be hard to implement on the Busy Box because the receivers do not have a direct way for sending an acknowledgement to the transmitter. The implementation of the transmitter module in the Busy Box is not very efficient when sending messages to individual D-RORCs as is explained in section 5.5. Hence the protocol implemented should not include lots of individual messaging to the D-RORCs.

An approach that is considered suitable for the Busy Box is to let the top level protocol be tolerant to lost messages. The communication between the two devices is very simple with one type of request and one type of reply. Requests can be sent multiple times until a correct event ID is received. The challenge is to keep the event ID queues synchronized. The protocol described in section 4.4 would recover if the request from the Busy Box is lost. A new request will be sent to the D-RORC at the next iteration. More severe consequences will follow if the D-RORCs reply is lost and the D-RORCs throws away the event ID. Then the event ID queues will come out of sync as described earlier.

To keep all the queues synchronized the Busy Box generates a request ID each time it pops a new event ID from the queue of event IDs received from the TTC. The request ID is included in the requests that are sent to the D-RORCs. The D-RORCs will remember the request ID from the last request and compare this with the request ID in the new request. If the IDs match it implies that the Busy Box has not popped an event ID and neither should the D-RORC do. If the ID does not match it implies that the Busy Box has verified the event and is now requesting the next event ID. For extra security the request ID generation can be deterministic so that the next ID is known. The D-RORC can then check if the newly received request ID matches the ID that is next in line. This will prevent requests with an corrupt request ID causing the D-RORC to pop an event ID incorrectly. However this is not likely to happen since corrupt messages should be detected and discarded by the parity check.

To summarize, the transmission error handling for the system is based on two abilities.

- Corrupt messages will be detected and discarded.
- The system will recover if messages are lost.

Figure 4.5 shows the decision flow for the D-RORC. As can be seen in the figure, there are three possible outcomes when a request from the Busy Box is received.

No reply. If the Busy Box requests a new event ID but none is available yet there is no need to send anything as a new request will be sent automatically. The request ID register should not be updated since the D-RORC have not popped an event ID.

Resend last message. If the received request ID matches the currently stored request ID it implies that this event ID has already been sent to Busy Box but for some reason it is re-requested. In any case it is safe to retransmit the last sent message.

Pop queue and send new event ID. If the Busy Box has generated a new request ID it implies that the Busy Box has moved on to a new event ID. The old event ID should be popped and the new should be transmitted to the Busy Box.

Figure 4.6 shows the sequence diagram for the Busy Box. This includes the basic operation that is implemented in firmware. The procedure outlined by this diagram lets the Busy Box perform the process of event transfer verification. By including the request ID in the process it should be possible to recover if any messages are lost in transmission.

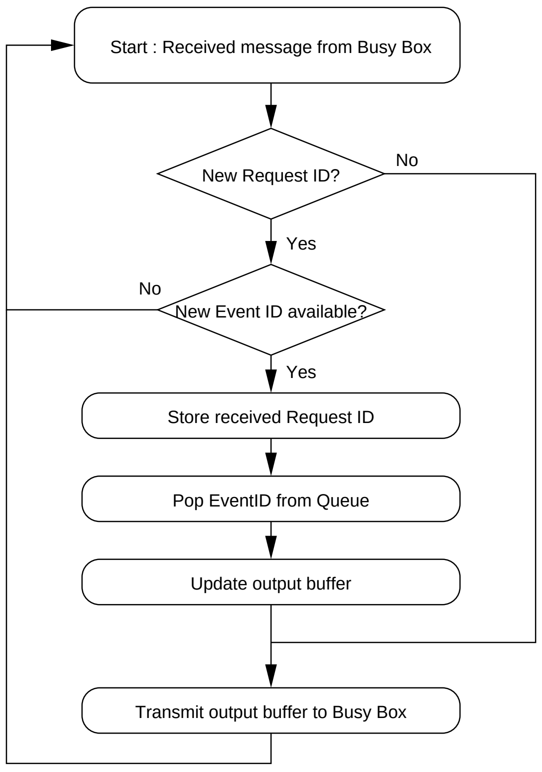
Start : Received message from Busy Box
New Request ID?
No
Yes
No
New Event ID available?
Yes
Store received Request ID
Pop EventID from Queue
Update output buffer
Transmit output buffer to Busy Box

Figure 4.5: Sequence diagram for the D-RORC

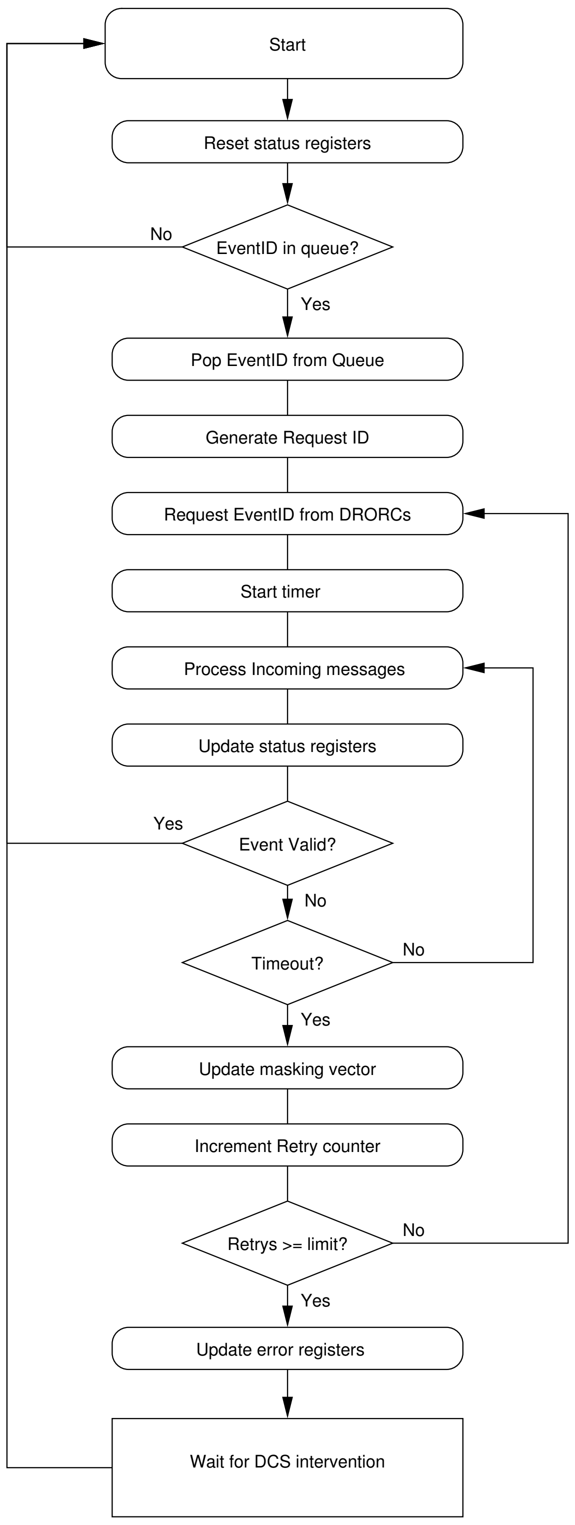

Figure 4.6: Sequence diagram for the Busy Box

4.6 Messages

Messages from the D-RORC must contain a 36 bit event ID. This already requires three 16 bit words. The message is 48 bits in length and will be transmitted as three 16 bit words that will be concatenated at the receiver. In addition to the event ID a message contains the D-RORC ID and a Request ID. The D-RORC ID is register that is set in each D-RORC to give it an unique ID. The Request ID is an ID generated by the Busy Box that is used to control the event ID queues in the D-RORCs. The bit-mapping for the D-RORC messages are defined in table 4.1.

Table 4.1: Bitmapping for D-RORC messages

47 - 44	43 - 32	31 - 8	7 - 0
Request ID	Bunch Count ID	Orbit ID	D-RORC ID

In general the Busy Box requests and the D-RORCs reply. Requests from the Busy Box is a single 16 bit word. The bit mapping is defined in table 4.2.

Table 4.2: Bit-mapping for Busy Box messages

15 - 12	11 - 8	7 - 0
Command type	Request ID	Unused

Table 4.3 lists the different commands or requests currently defined. Under normal operation the Busy Box will send the Request Event ID to the D-RORCs. Since the time taken to push a data fragment from the FEE to a D-RORC varies, the Busy Box will start requesting event IDs before it is available in the D-RORC. If there is no new Event IDs available the D-RORC should not reply anything as the Busy Box will keep sending out new requests until it has received valid replies from all D-RORCs.

The remaining three commands are meant for debugging and error handling. *Resend last message* commands the D-RORC to retransmit whatever message it last transmitted to the Busy Box. *Force pop Event ID* will force the D-RORC to pop one event ID from it queue without any checking of request ID.

Table 4.3: Busy Box commands

Command type	Bit Code	Description
Request Event ID	0100	Request an Event ID from the D-RORC.
Resend last message	0101	Command the D-RORC to re-transmit the last message sent.
Force pop Event ID	0110	Command the D-RORC to pop one Event ID from its local queue.
Force Request ID	0111	Command the D-RORC to store the attached Request ID.

Chapter 5

Implementation

5.1 Introduction

All firmware modules have been written in the VHSIC Hardware Definition Language (VHDL). The blockRAM modules and also the FIFO memory modules that are used in the design have been generated with the Xilinx coregen tool.

5.2 Overview

Figure 5.1 gives an overview of the top level module of the firmware. Another module is built around this top level module that instantiates Virtex-4 primitives such as I/O buffers and a DCM for clock deskew and synthesis. The DCM will provide synchronous 40 and 200 MHz clocks derived from the BC clock received from the TTCrx chip on the DCS board.

Since the FPGA does not support internal tristate drivers a separate module (named DCS Bus Arbiter and Address Decoder in the figure) has been designed to create a bridge from the DCS bus interface to internal sub modules. The module is responsible for the asynchronous handshaking and keeping the data signals at high impedance at all times except for when data has been requested from the DCS board.

The receiver module contains up to 120 serial receivers depending on a global generic that can be set at the top level of the design. Serial data will be decoded into 48 bit words with an additional 8 bit to indicate the channel number which it was received on. This data will be moved to both the RX memory module and to the event verification module. The RX memory module includes four Block RAMs and can store 1024 of the latest received messages from the D-RORCs. It provides a FIFO-like interface to receive incoming messages and a Random Access Memory (RAM) interface to the DCS side.

The transmitter module only contains one serial encoder that will transmit on all channels to all D-RORCs that are not masked away by a masking vector. The module accepts requests from both the DCS bus and the event verification module.

The Trigger Receiver module decodes the signals from the TTCrx chip and generates trigger messages and pulses. The pulses will be used by the Busy Generator to count used buffers and

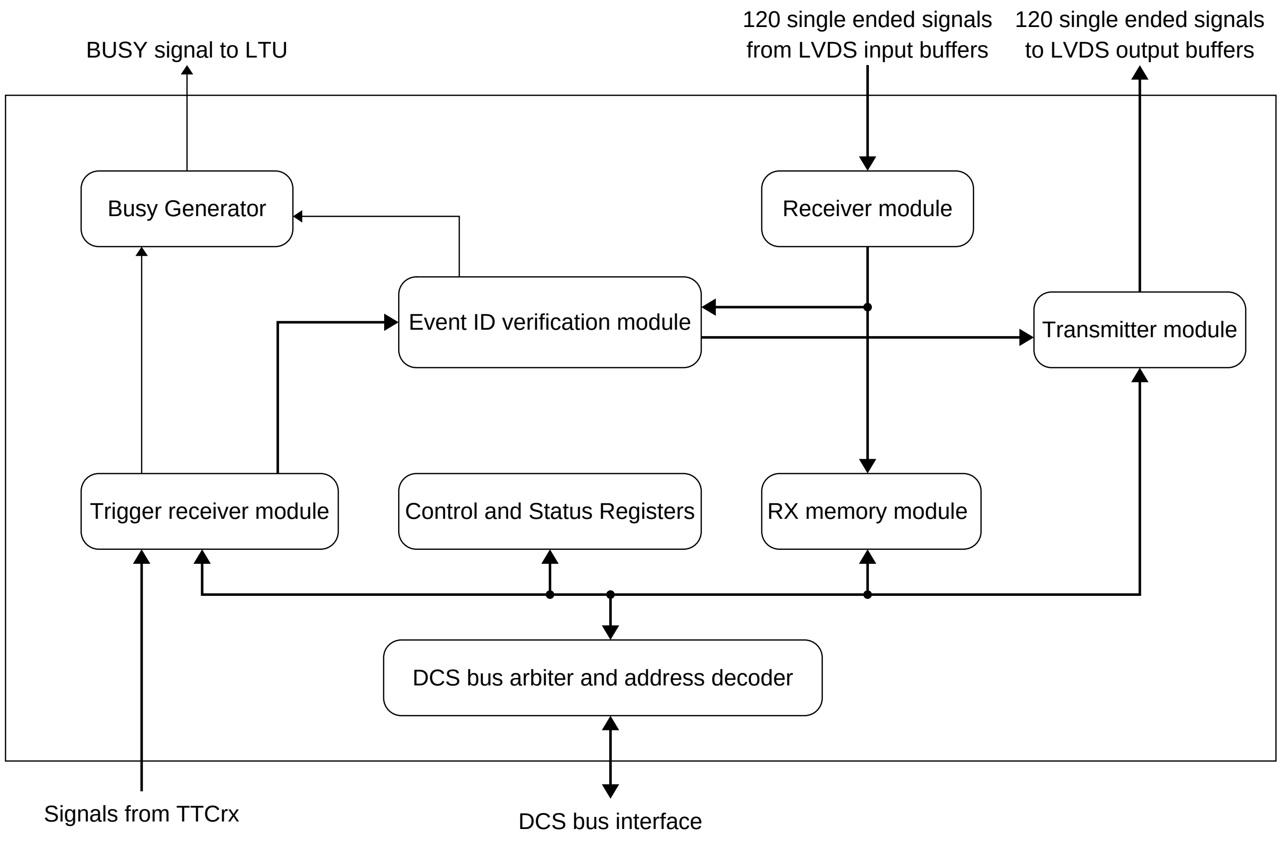
BUSY signal to LTU
120 single ended signals from LVDS input buffers
120 single ended signals to LVDS output buffers
Busy Generator
Receiver module
Event ID verification module
Transmitter module
Trigger receiver module
Control and Status Registers
RX memory module
DCS bus arbiter and address decoder
Signals from TTCrx
DCS bus interface

Figure 5.1: Overview of Busy Box firmware modules.

set the BUSY signal while the trigger messages will be used by the event ID verification module.

The event ID verification module will extract the 36 bit Event ID from the trigger message produced by the Trigger Receiver module and store it in a local queue for *verification*. A state machine will start the verification process by sending requests to all D-RORC via the transmitter module. The incoming replies are buffered and processed. Status registers for the current Event ID are updated as the D-RORC messages are being processed. A large logic gate on the status registers will indicate when the Event ID has been verified. The state machine will then generate a pulse and return to its initial state waiting for new Event IDs to enter the queue. The pulse created is read by the Busy Generator which will decrease its counter of used buffers in the FEE every clock this signal is asserted.

Finally, the Control and Status Registers module provides an interface from the DCS bus module to internal status and control signals. These signals are not shown on the figure but connects to most of the other modules. One important register which is held inside this module is the CHannel ENable (CHEN) register. One bit for each channel can be set to enable or disable that channel. This will disable the serial receiver and also exclude this channel in the verification process.

5.3 DCS bus arbiter and address decoder

The parallel DCS bus between the DCS board and FPGAs on the busy board are based on an asynchronous handshake protocol. The DCS board acts as the bus master while the FPGAs acts as slaves. The bus, from now on referred to as the DCS bus, has 16 bits data and 16 bits addresses in addition to control lines. It is the same interface as is used between the DCS and RCU boards except for that the RCU bus have 32 data lines. Figure 5.2 illustrates a read and a write action.

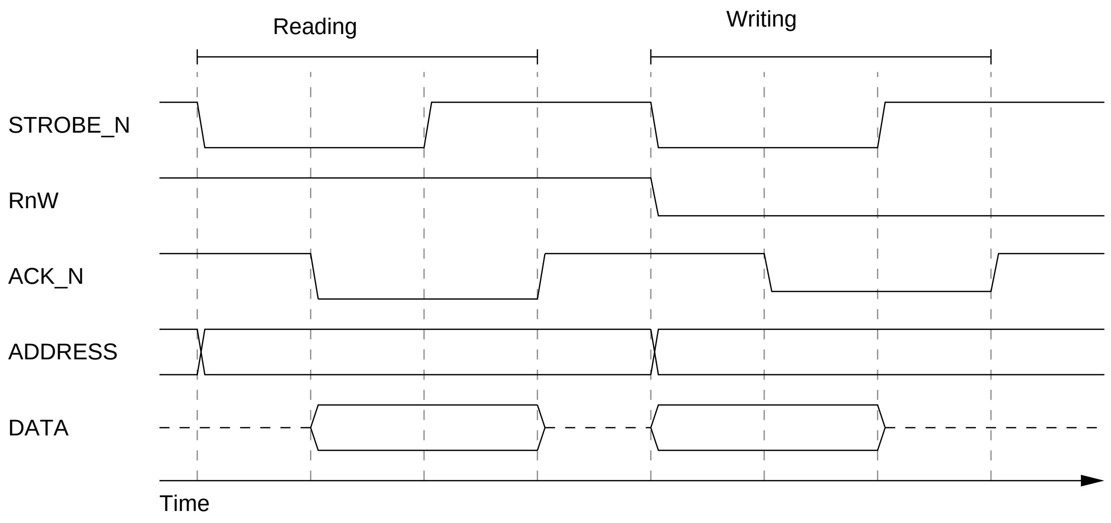

Figure 5.2: Read and write transactions on the DCS bus

The asynchronous handshake is done with the STROBE_N from the DCS board and the ACK_N from the busy board. These signals are sampled through 3 registers on both sides before they are evaluated to avoid metastability in the internal logic. This delay ensures that all other signals will have time to stabilize before they are sampled. The RnW signal indicates if the bus master(the DCS board) wants read the contents of a register or to write the content of the data lines to that register address.

The data lines are connected to in/out capable ports with tristate drivers. When reading one of the FPGAs on the busy board will drive the data lines, and when writing the DCS board drives them.

Table 5.1: Bit-mapping of DCS bus address

15	14 - 12	11 - 0
FPGA address	Module address	Sub module address

Since there are two FPGAs on the DCS bus, the MSB of the address is used to select which FPGA to communicate with. The next three bits are used to address a specific module in that FPGA and the remaining are used to address registers or memory locations of that module.

If the address given by the DCS board does not point to any of the internal modules, the request will be ignored. Otherwise an edge detector will detect a falling edge on the DCS_strobe_n

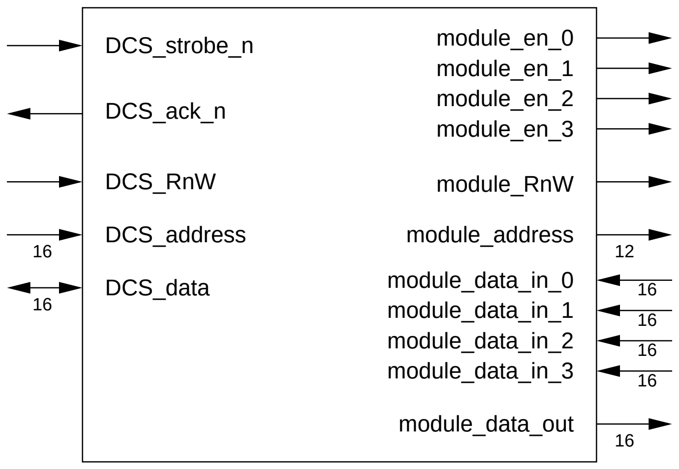

Figure 5.3: Overview of the interface of DCS bus arbiter and address decoder.

and generate a module enable signal that will be sent to the addressed module and the DCS_ack_n will be pulled low next clock cycle to acknowledge the request. The addressed module is then responsible for handling the request within the next clock cycle.

The bi-directional data lines are split into data out and data in signals for each connected module. The three module address bits is used as a select signals for the multiplexer that redirects the module enable signal and for the multiplexers that select the correct data signals from the internal modules. The 12 bits sub module address, the data and the RnW signals are passed on to the modules without any modifications.

5.4 Receiver module

5.4.1 Serial Receiver

The serial receivers of the Busy Box receives 3 x 16 bit words. It is built up by a serial receiver for a 16 bit word (excluding the frame and parity) and an outer state machine that makes sure that three consecutive words are received in a short restricted interval.

The incoming serial signal will be sampled at 200 MHz, first through some registers to make sure that the signal has stabilised at a valid logic level then into the shift register as shown in figure 5.4. In the figure a configuration with three samples per bit is illustrated, but the latest testing has been done with 5 cycles per bit. It is desirable to use three samples per bit to reduce the logic resources used but this has not been prioritised and will have to be done at a later stage in the development.

The capture condition is that start bit 1 and 2, and stop bit is resolved to their correct values. Logic calculates the parity of the data bits and compare it with the received parity bit at all times. When the capture condition is satisfied a data available flag is raised. The state machine will then copy the data word to a buffer and start a countdown timer for the next word. If the timer expires the data will be discarded and the next word received will be regarded as the first in the sequence

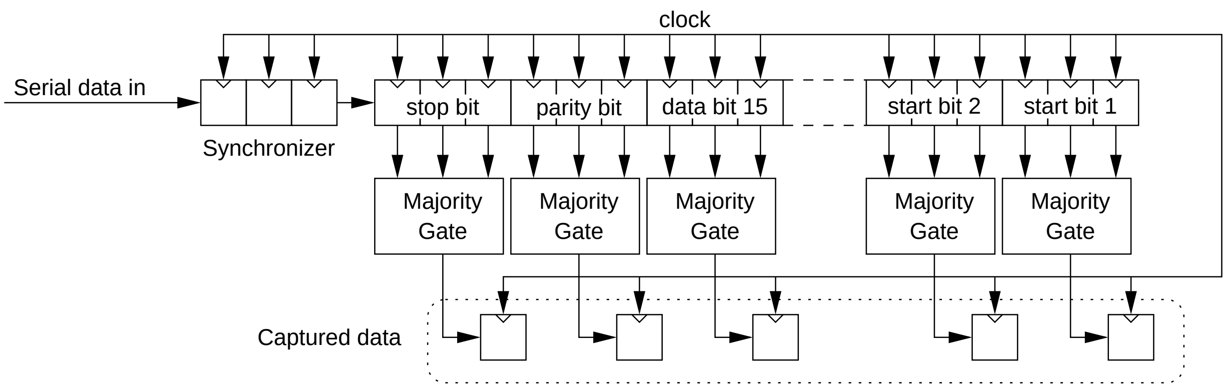

Figure 5.4: Illustration of internal architecture of the implementation of a serial decoder.

of three. If three words are received within the allowed time slot and none of them contained parity errors, the outer state machine will raise its data available flag to be read be the collector multiplexer tree described in the next section.

5.4.2 Multiplexer tree architecture

At maximum there will be 120 serial receivers on one FPGA. The messages must be copied from the buffers of each receiver to a memory fast enough so that the buffers never gets overwritten. The time to receive one message can be calculated by multiplying number of bits in a frame *times* number of cycles per bit *times* number of frames to complete a message. For 3 cycles per bit the calculation becomes 20 * 3 * 3 = 180 clock cycles and for 5 cycles per bit it becomes 20 * 5 * 3 = 300 clock cycles. In any case it is sufficiently fast enough to check one receiver at a time for new data. Every receiver will then be checked every 120 clock cycle and hence it will never be able to receive more than one message before being checked.

A straight forward solution is a large multiplexer controlled by a counter, all running at 200 MHz. However, the multiplexer has to be implemented by logic blocks in the FPGA. As the width of a multiplexer grow, so does the depth and propagation delay. At 200 MHz the maximum width of a multiplexer is about 30 inputs for the Virtex-4 LX FPGA. The way multiplexers are implemented in Xilinx FPGAs are discussed in [20]. To make the structure fast the multiplexer was split into two levels. The concept is illustrated in figure 5.5. At the bottom node, the backbone controller may have up to eight branch controllers and each branch are connected to 16 receivers. The branch controllers will cycle through the connected receivers and check for data available flags. If a data available flag is found, the message is copied to a buffer and the branch controller will hold until the backbone controller has verified that it has read the message. The backbone controller will spit out messages as they are received by toggling a write enable flag.

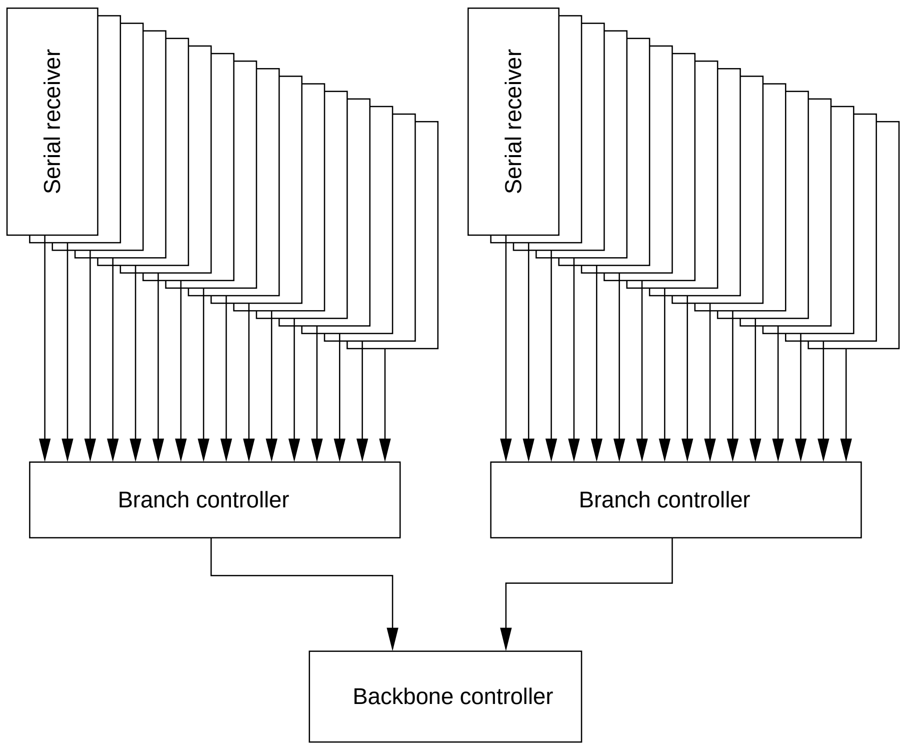

Figure 5.5: Concept for data collector architecture exemplified with 32 serial receivers and two branch controllers connected to the backbone controller. The backbone controller can handle up to eight branch controllers

Resource usage

The VHDL source code is designed to be scalable. The number of serial receivers instantiated by the source code is controlled by setting two generic inputs (number of channels and number of branches) at the top level entity. When the design is synthesized with all 120 channels, the resource usage is first reported to be 130 % of available resources. The tools are able to squeeze the design into the FPGA by violating timing constraints. Consequently, the architecture has to be optimised before the design can be synthesized with all 120 serial receivers.

The implementation described above is not optimal concerning resource usage. The branch controllers will be idle most of the time, waiting for the backbone controller. This idle time can be used more efficiently if the messages were moved in a semi-serial fashion rather than in full parallel. The 48 bit message can be split into three 16 bit words that are consecutively transfered serially. This will take 16 clock cycles instead of one. The benefit is that only three sets of multiplexers are needed instead of 48. This will reduce the logic usage considerably for this module.

To reduce logic resources, the existing architecture should be modified to use the semi-serial technique outlined. The branch controllers must be rebuilt to use only 3 sets of multiplexers and connect to only eight serial receivers. An extra buffer level at the bottom is needed so that the next message can be transfered while waiting for the backbone controller. The backbone controller must be modified so that it can serve up to 16 branch controllers instead of 8.

5.5 Transmitter module

The transmitter module consists of a 16 bit serial encoder and controller logic built around it. The output of the serial encoder is connected to all LVDS outputs but a masking vector can be set to disable individual channels. The module can be controlled both from the DCS interface and the Event ID verification module.

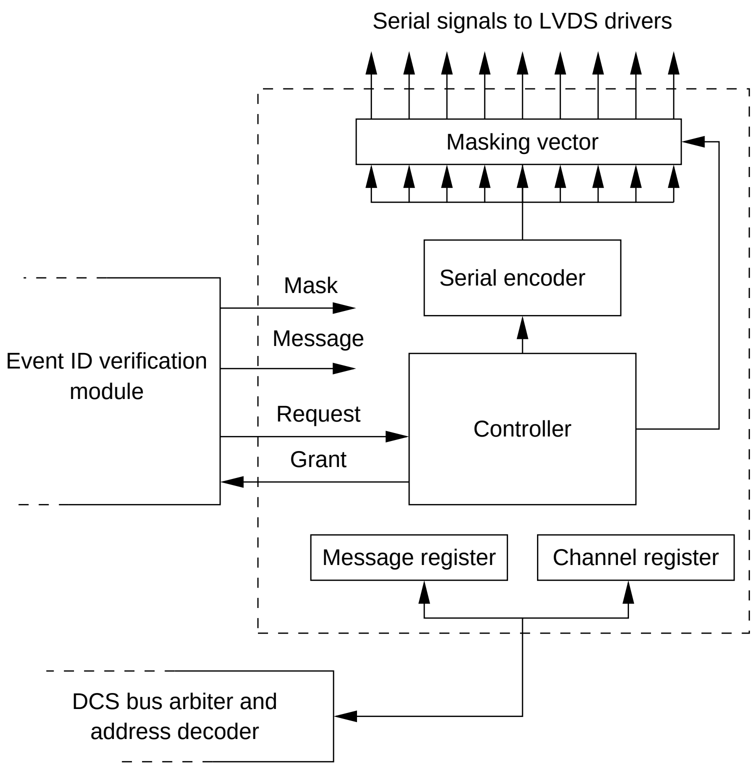
Serial signals to LVDS drivers
Masking vector
Serial encoder
Mask
Message
Event ID verification module
Request
Grant
Controller
Message register
Channel register
DCS bus arbiter and address decoder

Figure 5.6: Conceptual illustration of the transmitter module.

5.5.1 Serial encoder

This module is a state machine and a Parallel-In-Serial-Out (PISO) module. When initiated the state machine will load data in the PISO and transmit the start bits. The output of the PISO will then be directed to the output of the module and the PISO is shifted at the bit-rate. Finally the parity and stop bit are added to complete the frame.

5.5.2 Transmitter controller

The controller is responsible for initiating the serial encoder and setting the masking vector. Since the module has interfaces to both the Event ID verification module and the DCS bus module, it must negotiate with these modules to avoid conflicts and make sure that all messages are transmitted.

Two 16 bit registers can be accessed from the DCS bus as can be seen on figure 5.6. One contains the data or message to transmit, the other register specifies the channel that will be unmasked. All other channels will be masked so that the message will only be transmitted to the channel specified by this register. If the value in the channel register does not specify an existing channel[1], the message will be broadcasted to all channels. When data is written to the message register, a pending flag register will be set. The controller will see this flag when it is in a idle state. Data from the message register will be loaded into the serial encoder and the masking vector will be created based on the channel number in the channel register. The pending flag will be removed when the controller executes this procedure. The DCS board is controlled by software under the Linux OS that runs on a 40 MHz microprocessor. For this reason it is considered safe to assume that the transmitter module will work faster than the DCS board and a queue has not been implemented.

The event ID verification module must wait until the transmitter module is ready to handle the request. This handshake is done with the *request* and *grant* signals that are shown in figure 5.6. The controller will grant the request if there is no pending flag. When granted, data from the event ID verification module is loaded into the transmitter and a masking vector is copied to the masking registers. This lets the Event ID verification module set the masking vector directly and thereby transmitting to any subset of all channels.

5.6 RX Memory module

This module stores messages received by the Receiver module and makes them available via the DCS bus. For memory, four of the Block RAMs of the Virtex-4 are instantiated as 16x1024 blocks in dual port mode. Dual port RAM can be accessed from two asynchronous clock domains independently. On the receiver side, the four Block RAMs are combined so that 64 bits are written at once. When the Receiver module has received a message it will put the message and the channel number on the output and toggle a write enable signal. The message is 48 bits and the channel number is eight bits, in total 56 bits. This word, with eight zeros appended will be written to memory at the address given by a 10 bit counter. The counter will be incremented each time a word is written and wrap around to zero when it reaches 1023.

The DCS bus is only 16 bits wide so the DCS board must do four read operations to get one message. The DCS bus bridge module passes on the twelve Least Significant Bits (LSBs) of the address to the internal modules. The ten Most Significant Bits (MSBs) will be used as address to the block RAMs and the two LSBs is used to select which of the four block RAMs to access.

[1]Channels are numbered from zero up to the number of channels specified minus one. For example will 0x00FF always result in a broadcast since there will never be a channel number 255.

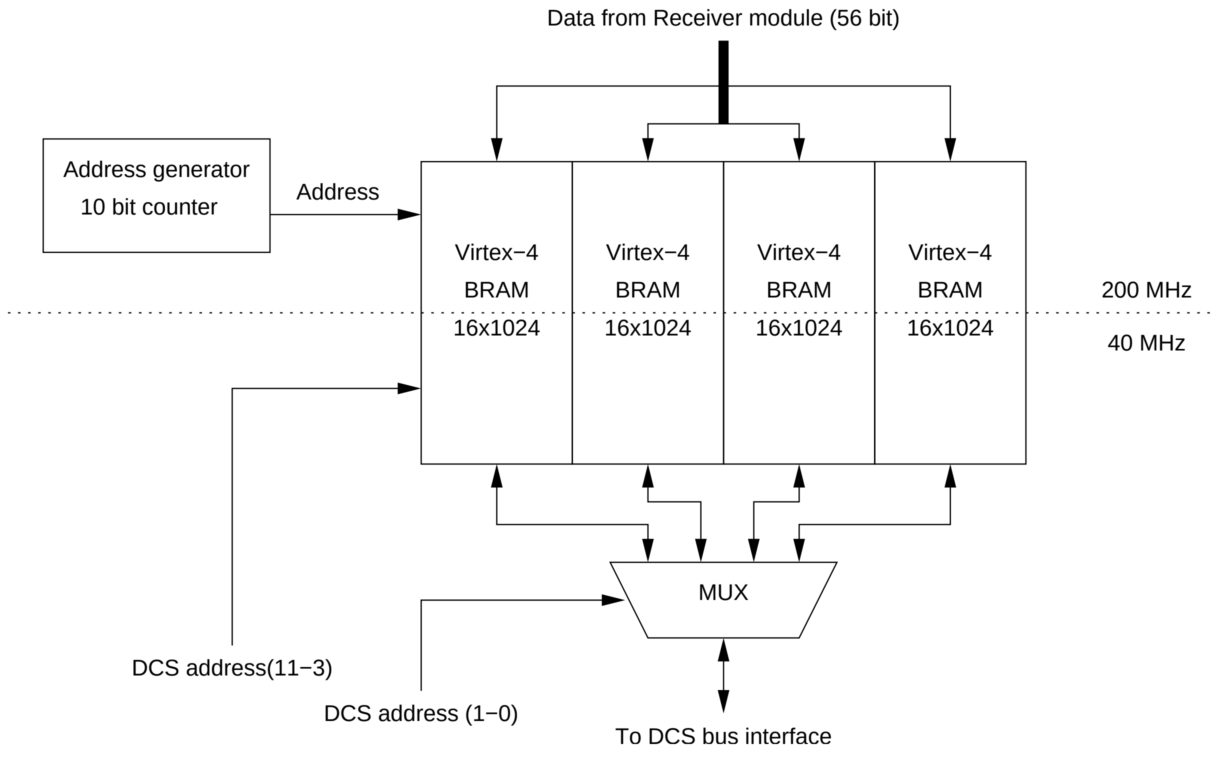

Figure 5.7: Illustration of the architecture for the RX memory module.

The DCS bus have both read and write access to this memory area. The write mode is only used for testing and verification.

Table 5.2 shows how the bits are arranged when 4 16 bits words are read out.

Table 5.2: Format of D-RORC answer in RX register

Word	15..12	11..8	7..0
1	Header [3..0]	BCID [11..0]	
2	Orbit ID MSB [23..8]		
3	Orbit ID LSB [7..0]		D-RORC ID [7..0]
4	Chan Num [7..0]		Unused

5.7 Trigger receiver module

The Trigger receiver module [13] decodes channel A and channel B signals from the TTCrx chip on the DCS board. For each trigger sequence it will generate event information in the CDH format and store it as nine words in an event First-In-First-Out (FIFO). Also, individual signals will be asserted when triggers are received, like Level 0, Level 1 and Level 2 Reject/Accept triggers.

The Hardware Definition Language (HDL) source code for the Trigger receiver module is the same that is used for the RCU firmware. A bus wrapper has been built around the module so that it interfaces correctly with the DCS bus bridge module. The bus on the RCU board is 32 bits wide so the wrapper uses an extra address bit to select the 16 most significant or the 16 least significant bits.

5.8 Event ID verification module

The event ID verification module is a composition of sub modules as shown in figure 5.8. Together they are able to perform the event verification process described in section 4.4.

Whenever the trigger receiver module has received a trigger sequence it will be read out by the event verification module and checked for a Level 2 Accept flag. If the trigger message contains a Level 2 Accept flag it implies that this event should be transfered to the D-RORCs. The event ID will then be extracted and stored in the event ID queue for verification.

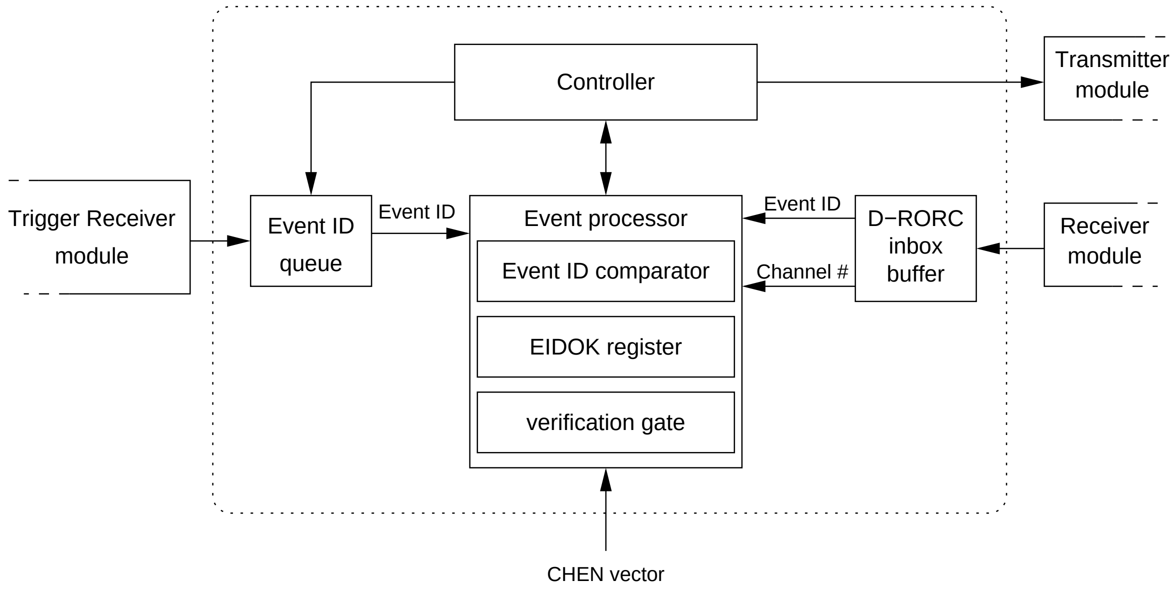

Figure 5.8: Overview of the event ID verification module.

The receiver module operates in the 200 MHz domain while the internal logic of the verification module runs in the 40 MHz domain. The D-RORC inbox buffer is a FIFO that buffers data from the receiver module and makes them available in the 40 MHz domain. When all the channels receive data simultaneously, the receiver module will produce data at a higher rate than the verification module can process. Hence there is a risk that the buffer will overflow such that data will be lost. This is however not likely to happen if the request interval/timeout register in the controller logic is set reasonably. The FIFO is able to buffer 512 words from the receiver module. Each word contains the D-RORC message followed by the channel number which it was received on.

The block labeled *event processor* in figure 5.8 will continuously compare the event ID on the output of the event ID queue and the D-RORC inbox buffer. If the IDs match, the channel number of the D-RORC message will be used to address a register and this register will be set to indicate that the D-RORC on this channel has received the current event. There exists one register for each channel and they are called the Event ID OK (EIDOK) registers. A large logic gate will evaluate the contents of these registers and also the CHEN registers. When all channels are either disabled in the CHEN register or checked in the EIDOK register the verification gate will assert an event verified signal.

The controller is a state machine that monitors and controls the process. Initially the controller sits in a wait state where a synchronous reset signal that resets EIDOK registers is asserted. It will negotiate with the transmitter module to transmit requests to the D-RORCs as described in section 5.5. When the transmit request is granted, the FSM will go to a wait state.

5.9 Busy generator

The busy generation is logical OR between two independent processes. It is enough that one the processes indicate a busy status. One of the processes is a countdown timer that is started every time a Level 0 trigger is detected. If there has been a collision in the detector, then the busy-signal should be raised to block any new trigger sequences for a specific period. This has to do with the drift time of the TPC. The countdown time can be set with a register in the Control and status registers module.

The other process is the calculation of free FEE buffers. If the number of used buffer is greater than or equal to the number of available buffers, the busy-signal is raised. The number of used buffers are counted by some simple principles:

1. The count is incremented by one when the Level 1 trigger signal from the Trigger receiver module is asserted. The FEE of the TPC will start sampling data on the Level 1 trigger. For other sub detectors this may be different and consequently the module must be modified for these sub detectors.

2. The count should be decremented by one when a Level 2 Reject trigger is asserted. This implies that the FEE should abort the data taking and mark the buffer as free again.

3. The count should be decremented by one when the Event verification module asserts the event valid signal. This implies that the event has been successfully transfered to the DAQ-system and hence the buffers that the event occupied is now free.

The logic also handles the cases where two or more of these signals are asserted at once. A Level 1 trigger and Level 2 Reject trigger will never happen at the same time so this is not handled. Condition 1 and 3 will cancel each other out. If condition 2 and 3 are present at once the buffer count must be decremented by two.

5.10 Control and status registers

This module provides access to many of the control signals and registers of the Busy Box firmware. A list with description follows.

RX memory pointer This read only register returns the address where the next received message will be written to in the Receiver memory module.

Number of event IDs This read only register returns the number of event IDs that are stored in queue for verification. The value does not include the event ID that is being processed.

Current event ID This is a 36 bit word that must be read out in three operations. It returns the event ID that is currently being processed by the Event verification module.

Most recently received event ID Returns the event ID that was last received from the Trigger receiver module.

Level 0 timeout A read/write register that specifies (in clock cycles in the 40 MHz domain) how long the busy-signal should be asserted after receiving a Level 0 trigger.

Number of FEE buffers Read/write register that specifies how many events that can be stored in the buffers in the FEE before they will overflow.

Halt Event verification This a single register that, when set to one, will cause the FSM in the Event verification module to halt in fixed state. The FSM will continue if a zero is written to the register.

Force verify Writing a one to this register creates pulse to the Event ID verification module and forces it to move on to the next event ID in the queue even if it has not been verified that all D-RORCs have received data for this event. The FSM in the verification module will only see the created pulse if has been halted and is in the halt state.

Request Timeout A 12 bit read/write register that specifies how many clock cycles the Busy Box firmware should wait before sending a new request to the D-RORCs.

CHannel ENable registers These registers are contained in the module and indicates if a channel is enabled or disabled. The eight LSBs of the address specifies the channel number. When writing, the LSB of the data are stored in the CHEN register.

Chapter 6

Testing and verification

6.1 Simulation

The HDL source code describes the behaviour of an idealized digital electronic device. The code includes both algorithmic and structural descriptions. A HDL design may consist of many submodules which in turn includes other submodules and so on. The complexity makes it necessary to take advantage of computer assisted verification tools so that the description of the system can be verified before progressing the next stage of implementation.

A simulator is a software tool that can simulate how a described system will behave in the time domain. The simulator needs to keep track of all signals and their interactions in time. The values of signals are changed by processes defined in the HDL code. Each process defines its output signals as a function of the input signals and each such process has a sensitivity list. The sensitivity list is a list of signals that the process is sensitive to. The simulator uses discrete values to represent time and preserve the rules of causality. When a signal in the sensitivity list of a process is updated the process is evaluated by the simulator and may result in a change in the process' output signals. Since the change in the output signals is an effect caused by the change in the input signals the update of the output signals must happen some time *after* the cause. The simulator emulates this by scheduling signal updates to happen at some time value in the future. The time delay from cause to effect can be defined by the HDL code of the process. Otherwise the signal update will be scheduled for the next delta delay which is an artificial time unit that only serves the purpose of emulating causality. In effect the simulator will never progress in real time units, only iterations when delta delays are used.

An iteration is all the operations the simulator does at one time cycle before moving on to the time where the next signal update is scheduled. To summarize; an iteration includes updating the signal changes that are scheduled for this time cycle and evaluating all processes that have one or more of these signals in their sensitivity list and finally schedule the updates of the changed signals. The simulator usually parses the code, analyze all processes and pre-compile binary program code to speed up simulation times.

A simulator is an indispensable tool when writing HDL source code. It lets the designer see all the internal signal values of a design at all times. This crucial to find and trace back bugs and

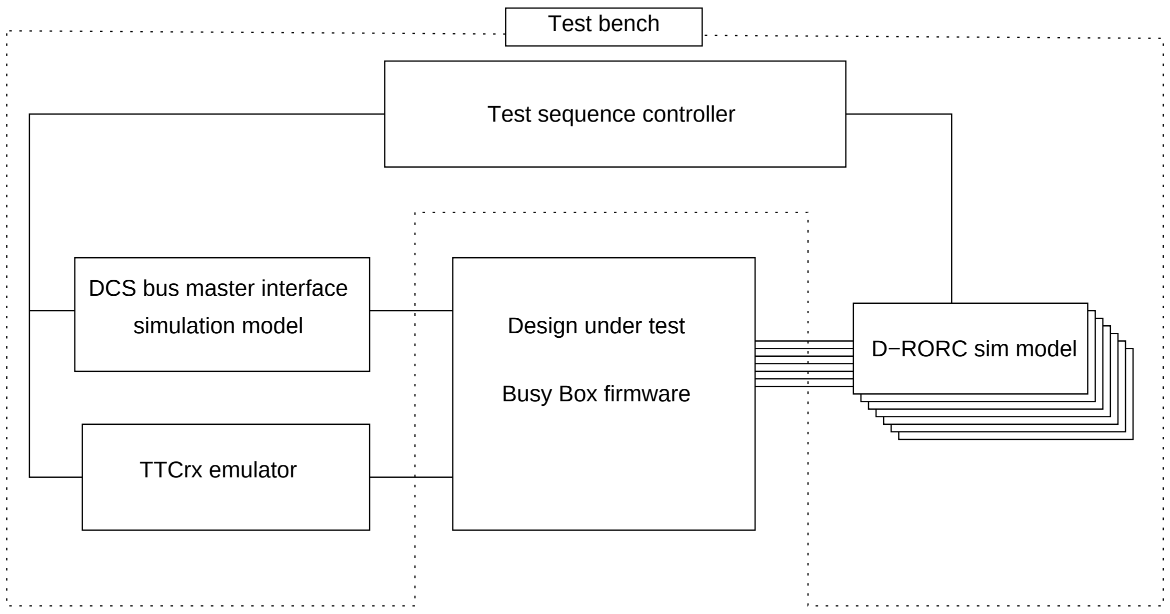

Figure 6.1: Schematic overview of the testbench for the Busy Box firmware.

errors that that appears at the outputs of the design.

6.1.1 Simulating the Busy Box firmware

To be able to simulate a design it is necessary to create a simulation environment around the design that interacts with it in the same way that is expected in the real system setup. For the case of the Busy Box, the design will be realized in a FPGA that is connected to other devices such as the DCS board and D-RORC. These other devices needs to be emulated by accurate simulation models so that the performance of the design under test can be evaluated. Simulation models and the simulation environment is written in VHDL as well, but the code doesn't need to be synthesizable. Such a simulation environment is called a testbench. The testbench will generate all stimuli and may also automatically verify that the design responds as expected.

Figure 6.1.1 shows a conceptual illustration of a testbench for the VHDL source code for the Busy Box firmware. The test sequence controller is a set of sequential statements that uses emulation modules or subprograms to manipulate the Busy Box design. The DCS bus master emulation is simply two VHDL procedures that, when called, performs bus transactions according to given parameters and specifications. For example a write transaction can be performed by calling a DCS write procedure with data and address given as parameters.

At the time of this writing the TTCrx emulation has not yet been developed. It can be implemented as subprograms or as an entity that are controlled with signals from the main test sequence controller. It must be able to emulate the serial signals for channel A and channel B and provide an easy way to generate trigger sequences.

The D-RORC emulators are entities with the same serial receiver and transmitter that is implemented in the D-RORC. The D-RORC can be programmed to reply in the same as the D-RORC will but it will receive event IDs from the test sequence controller.

With these tools available it is easy to write a test sequence that emulates different scenarios that are to be simulated. For example trigger sequences can be sent to the Busy Box through the TTCrx emulator. Later on in the simulation the event IDs that were sent with the trigger sequences can be given to the D-RORC emulators. The Busy Box firmware design will try to verify the event IDs by communicating with the D-RORC emulators. The entire process can be inspected by looking the output of the simulator, usually in a wave-viewer software.

This testbench does not do any error checking and will not detect or alert about any errors automatically. The purpose is only to create an simulation environment to be used by the developer to be able to inspect the resulting behaviour of the source code.

6.2 Serial LVDS communication tests

The serial communication with the D-RORCs is the foundation of the system and a prerequisite for further testing of dependent features. Consequently, it has been a high priority to develop and verify a reliable communication link. Testing the serial communication in hardware is necessary to get an impression of how well the communication channels performs.

6.2.1 Loopback cable test

The first tests involved just the Busy Box. In the early stages of development the serial protocol was based on the work presented in [21]. It consisted of frames with 2 start bits, 32 bits of payload, one parity bit and finally a stop bit. In total a 36 bit frame. The firmware included both transmitter and receiver and the communication logic ran on 200 MHz. One bit period was 4 clock cycles resulting in a 50 MHz bit rate. The goal of the test was to be able to transmit a message through a loopback-cable. In other words, to drive a serial signal into a cable and decode the signal at the other end of the cable with the same device. With this test setup the transmitter and receiver run on the same clock which means that there is no accumulation of phase error. This will not be the case when communicating with the D-RORCs.

The firmware included a DCS interface with access to memories and registers in the FPGA that provided an interface to access the serial communications via DCS board. The RCU shell is a flexible software program on the DCS board that provides a command line interface to perform read and write operations on the system bus to the Busy Box FPGAs.

The first thing these tests revealed was that unconnected ports produced a lot of garbage data. This was discussed in section 4.2. When registers to disable/enable individual channels were implemented this test setup was fairly successful. It was planned to develop a software that transfered large data chunks over the link get some statistics about the transmission loss. However this was never achieved before the system was brought to CERN for testing with the D-RORC.

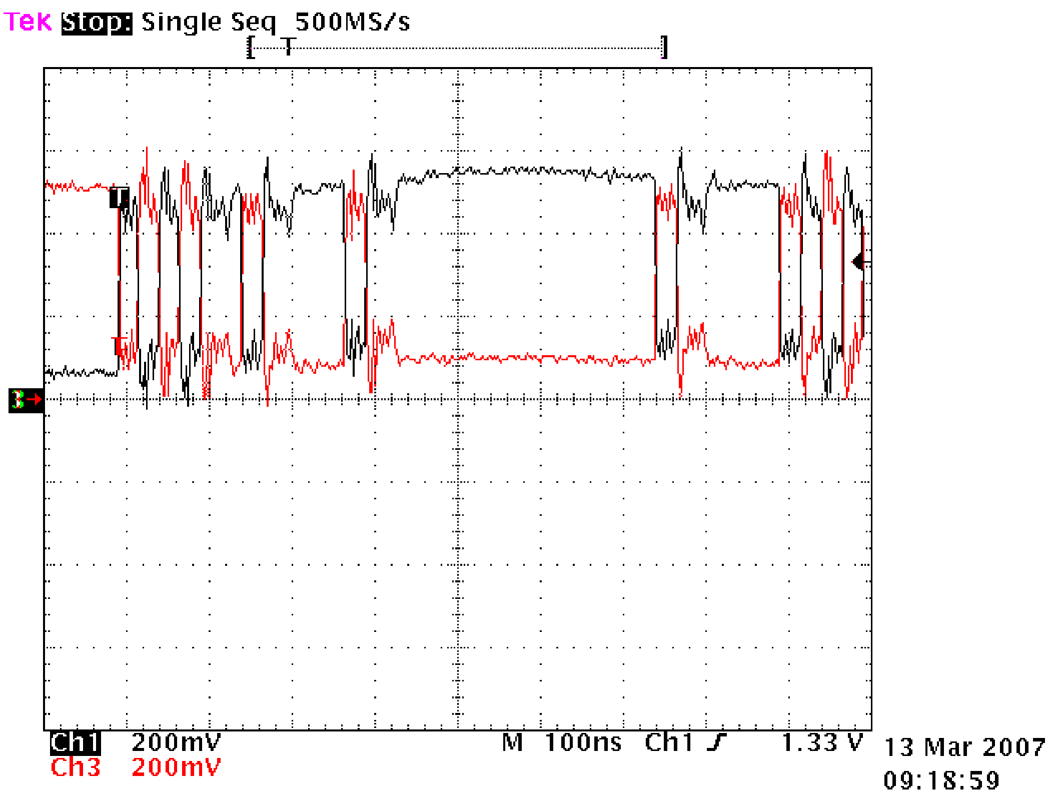

Figure 6.2: Measurement on the transmission lines of the LVDS-based communication between a D-RORC and a Busy Box. Ground is referenced to the groundplane in the PCB of the receiving device, in this case the Busy Box.

6.2.2 First integration tests at CERN

The system was brought to CERN to test the communication with the D-RORCs. After some debugging and the implementation of the improved serial receiver discussed in section 4.2 a successful "proof of concept" test was achieved during this stay. This included a system setup with a LTU in emulation mode, a DDL Data Generator (DDG), D-RORC and the Busy Box. The DDG is device developed by the DAQ team at CERN to emulate the FEE. It is capable of receiving triggers and generate event data and a CDH in the proper format.

The "proof of concept" test included issuing a trigger sequence which was received by the DDG and the Busy Box. The DDG generated event data and a CDH and transmitted this to the D-RORC. A small program was able request the Event ID from the D-RORC and match it with the one which was received by the Busy Box. Figure 6.2 shows the serial differential signal as seen on the oscilloscope.

During the stay at CERN the system was discussed with the developer of the D-RORC firmware. It was agreed that the messages from the D-RORCs should be 48 bits so that an event ID of 36 bit could be transmitted in a single message. The messages from the Busy Box to the D-RORC should be only 16 bit since hardly any data will flow in this direction.

6.2.3 Test of communications with another device as D-RORC replacement

After the stay at CERN the discussed changes were implemented along with other new features such as the event transfer verification module. The new communication modules are the ones that are currently being used in the design. To be able to test the new communication in hardware before the next tests at CERN another device capable of implementing the modules that were intended for the D-RORC was used. This device was the TriggerOR board which has the same Virtex-4 FPGA as the Busy Box system. The TriggerOR board also uses an DCS board and is very similar to the Busy Box system. This made it easy to develop a firmware design with a serial transmitter and receiver and a bus interface to the DCS board. The serial communication could then be controlled and verified with the RCU shell on both boards.

After some debugging and tweaking the new communication modules was sent to CERN so that they could be implemented in the D-RORC firmware prior to the next integration tests.

6.3 More integration tests at CERN

During this stay a test setup identical to the previous but two sets of DDGs and D-RORCs was used. With the new communication modules and new firmware features the Busy Box was able to verify the event transfer from the DDGs to the D-RORCs at rates up to 1800 Hz. This rate is much higher than the maximum expected readout rate for the TPC detector system which is about 200 Hz.

6.3.1 Integration test with a complete TPC sector

Personnel at CERN have tested the Busy Box in a test setup that is very realistic. A complete TPC sector with all the FEE was set up in a laboratory. The setup is illustrated in figure 6.3. A problem with these tests are that the firmware for the RCU boards is an old version that was not designed for the new trigger sequences. As a result the Level 0 trigger had to be disabled. Also some of the RCUs did not produce a correct CDH and had to be excluded from the test setup.

With three of the RCUs and corresponding D-RORCs disabled this test setup achieved short runs with 400 Hz readout rate before the Busy Box asserts the busy signal because an event can not be verified. The problem is being investigated but there has not been discovered any misbehaviour in the Busy Box.

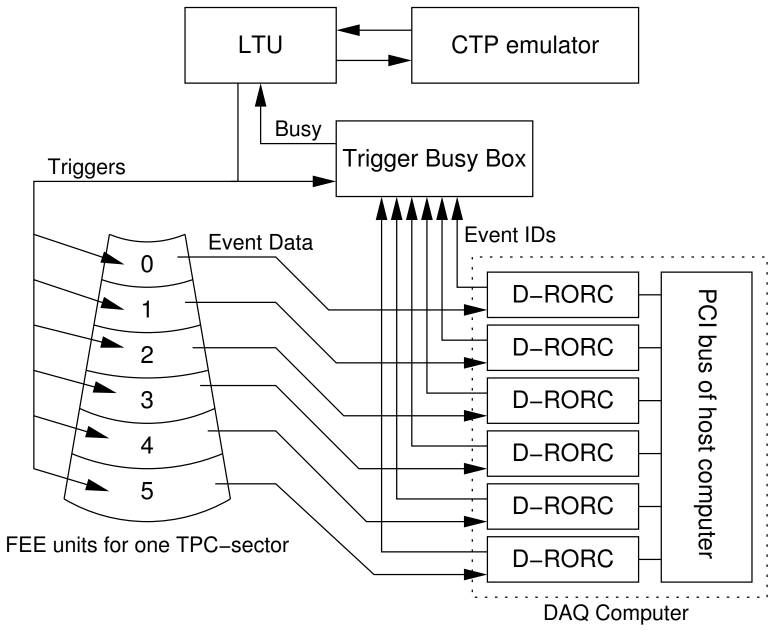

Figure 6.3: Schematic illustration of the test setup at the RCU lab at CERN.

Chapter 7

Discussion and conclusion

7.1 Firmware completion

There is still some work to be done with the firmware development. First of all the architecture must be optimized to use less logic resources so that all 120 serial receivers can be implemented in the first FPGA. There are several possible strategies that can be applied to achieve this goal. One is to modify the architecture of the multiplexer structure as described in section 5.4.2. If this is not sufficient the choice or implementation of serial receiver must be evaluated. The current implementation uses a lot of resources. An interesting strategy to reduce the resource usage is to review the asynchronous resets. All synchronous processes in the VHDL source code uses a standard template with asynchronous resets of all registers. Since all the registers in the FPGAs will in a known state when it is programmed the asynchronous reset is not really necessary. By removing the asynchronous reset and reviewing the code of the serial receiver the resource usage can be reduced considerably. An article [22] published on the Xilinx website claims that designs can become 50% smaller by getting the priorities of the registers right.

The firmware must be built in two versions, one for FPGA1 and one for FPGA2, where the busy signal from FPGA2 is routed to FPGA1 which will coordinate the busy generation.

Furthermore, the request ID feature must be implemented both in the Busy Box and the D-RORC. Currently the system will hang in a busy state even with single transmission error over the LVDS link.

Finally the registers available to the software running on the DCS board must be evaluated. The software designer is working on the requirement specifications.

7.2 Identifying and exploring sources for error conditions

There are many sources of errors and all possible scenarios must be identified and explored. Some sources of errors that have not yet been covered:

Invalid trigger sequences If there is an error in the trigger system or transmission error of the triggers this will most likely lead to an event that can not be verified by the Busy Box.

How these scenarios will develop is at this time somewhat uncertain and depends among other things on how the firmware of the RCUs will react. At present time a new version of the RCU firmware is under development and so the final behaviour of this firmware is not yet available.

Missing event ID in the D-RORCs This can be caused by errors in the RCU or D-RORC or a transmission error of the event data. This will cause the Busy Box to hang in the busy state because the event transfer can not be verified. A possible solution is to report the Event ID to DCS system and try to skip the event.

Event ID mismatch If the event ID from one or more D-RORCs keeps resulting in a mismatch with the event ID form the trigger system (the Busy Box should keep requesting event IDs if the don't match) this is most likely not a transmission error between the D-RORC and Busy Box. The software can check if the event IDs received from these D-RORCs have been issued by the trigger system at all and try to resolve the situation by skipping some events and reporting it to the DCS.

Non-responding D-RORC This could be a malfunction in the D-RORC itself or the communication link. The error condition must be reported to the DCS so that it can be investigated by personnel and cannot be resolved by the Busy Box.

There are probably many more error scenarios but only thorough analysis of the complete system and testing can determine the probability that such scenarios will occur. It may not be worth the effort to implement error handling features that only solves rare problems.

7.3 Software for the DCS board

There is a lot of work to be done with the software that will run on the DCS board before the Busy Box system can be fully integrated with the rest of the data acquisition system.

One of the first things that the software must take care of is to program the FPGAs upon start up of the system. Then the software must set the correct parameters in the configuration registers, check that the system is ready to go and report to the DCS system when the Busy Box is operational.

The Busy Box must also be fully integrated in the DCS system. A software framework for this has already been developed for the RCU board. This software must be adapted to the Busy Box hardware and functionality. This includes adapting the low level functions that access the hardware registers. All DCS boards will run a FEE server which clients can connect to through the network interface and gain access to various services. The Busy Box will most likely require some special services for the FEE server that must be developed.

Another task for the software is the high level error handling. This will in the first stage include monitoring the operation of the firmware and detect possible error conditions, especially if the event transfer verification has stopped. Procedures to quickly identify the source of the problem must be developed. How the different error conditions should be handled must be discussed with parties of the other systems.

7.4 Testing and integration of the finalized system

So far the Busy Box has not been tested with more than six channels. When the firmware is ready to operate on 216 channels this must be tested with the TPC detector. Any problems and issues that is revealed during these tests must be fixed before the system is ready to be integrated.

7.5 Conclusion

The development of the firmware for the Busy Box is not yet completed. With the results from the tests at CERN and the improvements outlined in this thesis, I am confident that the Busy Box will meet requirements with a little more work.

From the test results we have verified that the communication is robust but we need to be able to recover quickly if message is lost or corrupted. This will be achieved by implementing the request ID feature in the Busy Box and the D-RORCs. The tests have also convinced us that the operation of the Busy Box is fast enough. The biggest delay in the procedure is the polling for event IDs from the D-RORCs. Since this is mostly parallel operations the timing should not be affected when adding more channels to the system.

The tasks of the Busy Box will be implemented with simple and robust solutions that ensures stable and reliable operation. The work to finish and include the features discussed in this thesis will continue.

Appendix A

Firmware for readout electronics of a muon detector

This appendix describes the Cosmic Ray Telescope (CRT) and the implementation of the firmware for the digital part of the readout electronics. This was the first project I was involved in and my first introduction to firmware development. The information about cosmic radiation has been obtained from Wikipedia. Information about the detector has been obtained from conversations with Lars G. Johansen.

A.1 Introduction

The Cosmic Ray Telescope (CRT) is a scintillator based muon detector built at the University of Bergen (UiB). It is designed to detect muons that originate from high energy particles from the cosmic radiation of the Universe. The detector includes 16 channels where the scintillator of each channel are placed strategically to be able to obtain spatial information about the detected muons. Muon hits will be registered in time and space and transfered to a desktop computer for analysis and storage.

The detector is built to be used in exercises for undergraduate students. Such exercises give students an excellent opportunity to gain first hand experience of experimental particle physics.

A.2 Cosmic rays

Cosmic rays are particles from space that hit the Earth's atmosphere. They are actually not rays but rather single particles that travel in space at high speeds. The sources of these particles can be many different physical processes from anywhere in the universe. Examples of sources for cosmic rays are the sun in our own solar system, super novas, rotating neutron stars or black holes. However the sources and origins of the most energetic particles still remains unknown.

The magnetic field of the local galaxy will bend the orbits of the particles and cause them to move in a circular or spiral paths. Thus the cosmic rays do not provide much information about the direction to the source from which they originate.

A.2.1 Air showers and muons

When cosmic rays enter the Earth's atmosphere, the high energy particles will collide and interact with the gas molecules in the atmosphere. In the collision the original nucleus will disintegrate and new particles will be generated. The new particles will collide with other gas molecules and the process repeats itself, creating a cascade of particles, and the result is called an air shower. The number of new particles created depends on the energy of the original particle. A high energy cosmic ray striking the Earth's atmosphere may result in billions of particles in an air shower. The reactions that occur in the collisions will produce many kaons and pions. These unstable mesons will quickly decay into muons or neutrinos. The muons do not interact strongly with matter and hence they will continue in their path down towards the Earth's surface. In fact, the muons are also unstable particles with a mean lifetime of about 2.2 μs before they decay into several lower energy particles. By the theory of classic mechanics they would not reach the surface before decaying. But the high speeds at which they travel makes them subject to the relativistic effect of time dilation. The muons will experience time slower than observers on the Earth. This allows the muons to reach the surface and even penetrate several hundred meters into the ground.

A.2.2 Muon detection

Muons are ionizing radiation that can be detected with scintillating materials. When the muons pass through a scintillating material, they will generate a short flash of light by an effect called Compton scattering. The Compton scattering process will convert the some of the energy of the incoming ionizing radiation into photons with a material dependent wavelength, typically in the UV-region. Thus the number of photons produced, or the *intensity* of the flash, will be proportional to the energy of the initial particle. The flash can be converted to an electrical pulse by using some sort of photo sensitive transducer, for example a PhotoMultiplier Tube (PMT).

A.3 The Cosmic Ray Telescope detector system

The illustration in figure A.1 gives an overview of how the detector is composed. 16 sets of scintillators, PMTs and signal shapers provide 16 channels. The signal shapers convert the signals from the PMTs into signals with square pulses of fixed length. A Complex Programmable Logic Device (CPLD) will sample the outputs of the signal shapers and add a time stamp to the samples. The data are then sent to a converter that transmits it through a standard Universal Serial Bus (USB)-cable to a desktop computer. The computer will be used for data analysis and storage.

A.3.1 Scintillator

The utilized scintillators are made of organic plastic material that are doped in several levels so that light of short wavelengths will be shifted in several steps to a violet light in the visible

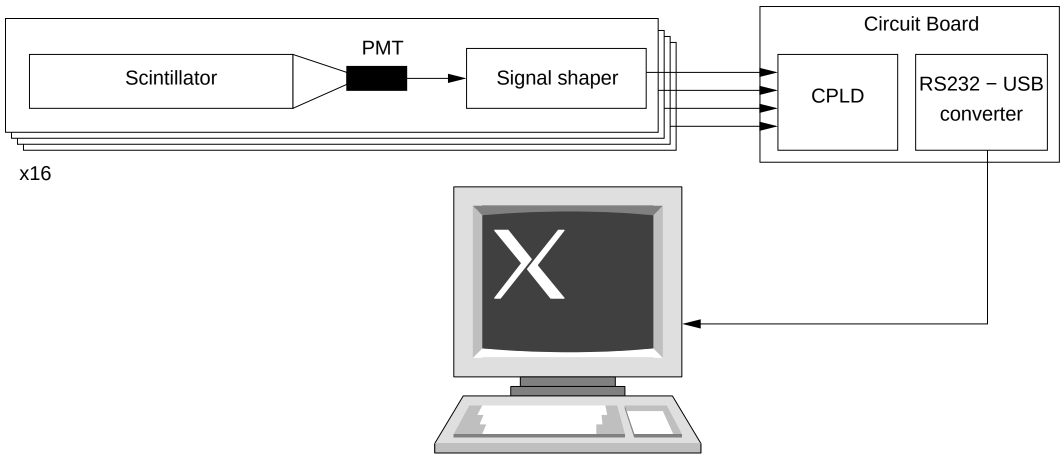

Figure A.1: Overview of the main components of the muon detector.

spectra. In effect, the scintillators converts high energy photons or radiation to photons of lower energy. This type of scintillator have been chosen because the violet light they emit is in the range where the PMTs are most sensitive.

The detector is built with 16 separate scintillators placed as shown in figure A.2. Each scintillator is 100 cm long, 25 cm wide and 2 cm thick. The idea is to create two horizontal planes with a 4 x 4 grid placed vertically above each other. This makes it possible to obtain spatial information about the passing muons. A muon that strikes through one of the planes will most likely be detected in two channels in that plane which will indicate a square in the grid.

The scintillators are individually wrapped in reflective mylar and coated in black materials to keep light from entering. Any light or radiation that is not caused by muons is considered as noise and must be prevented from affecting the scintillators. Cones made of nonscintillating plexiglass leads the light from the scintillators into the PMTs.

A.3.2 PhotoMultiplier Tube

A PMT is a vacuum tube usually made of glass. Inside the tube is a photocathode, an electron multiplier consisting of several dynodes, and at the end an anode. A high voltage source produces an electric field from the photocathode to the anode at the other end of the tube. The dynodes are charged with intermediate voltages to create a potential between each dynode.

Incoming photons will hit the photocathode, and by the photoelectric effect electrons are released from the cathode. The electric field will accelerate the freed electrons towards the first dynode of the electron multiplier. When the accelerated electrons hit the dynode, a greater number of electrons will be realised and accelerated towards the next dynode. The process is repeated for each dynode in the tube and the result is a great increase in the number of electrons. At the end of the tube the produced electrons will deposit a negative charge on the anode. This charge can be read out as a negative current pulse.

The PMT utilized in the CRT are a Russian produced model named FEU-84. The electron

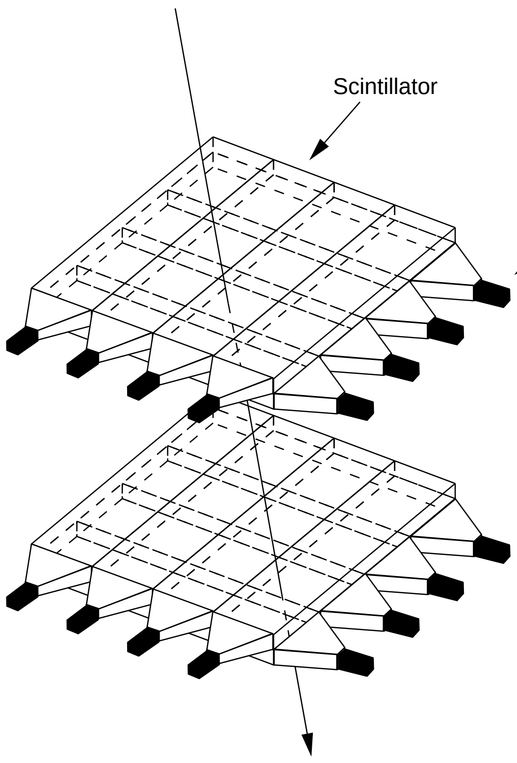

Figure A.2: Illustration of the muon detector setup.

multiplier contains ten dynodes. An external high voltage power source supplies the voltages applied to the electrodes. The power supply generates a differential voltage potential from 0 down to -2 kV from a +5 V input supply voltage. A circuit of resistors and capacitor generates the intermediate voltages for the dynodes. A dedicated output with a voltage signal ranging from 0 to 2.048 V can be used to monitor the voltage applied across the photocathode and to anode. The negative pulse at the anode will range from a few tenths of mV to several V depending on the amplification of the tube.

A.3.3 Asynchronous trigger circuits

Analog electronics detects the negative pulses from the PMTs and creates a digital pulse of fixed length. The output signal from the PMT is amplified and shaped in four stages before triggering a monostable latch. The output of the latch will be driven to either +5 V or 0 V which are valid logic high and low values, respectively. If the latch is triggered it will produce a logic high voltage on its output for about 200 ns before returning to logic zero voltage. The length of the pulse is determined by the discharge-time of a RC-circuit. Thus all information about the intensity of the flash has been stripped away and consequently the individual channel will only provide information about whether a muon has passed through the scintillator or not.

A.3.4 Digital readout electronics

A CPLD will sample the asynchronous output pulses from the trigger circuits at a rate of 10 MHz. Hence the sample period is 100 ns and a pulse from the monostable latch will normally be sampled twice at a logic high value. However, the muons will hit randomly in time and jitter noise will be introduced by the electronics. This implies that the length of the pulse may deviate slightly from the nominal 200 ns and the pulses will arrive randomly in time and phase. This may cause a pulse to appear as one or three high samples as well as the nominal two.

Whenever there is a hit in one or more of the channels the readout electronics will capture the values of all channels and transmit them together with a timestamp to the computer. The communication link with the computer is provided by USB standard interface. The CPLD will interface with an on-board USB converter with standard RS-232 protocol. The converter is connected to the computer with a USB cable and is able to translate and forward data in both directions.

Complex Programmable Logic Device

The digital electronics is provided by an MAX II CPLD produced by Actel. The specific device is EPM1270T144C with speed grade -5. It resides in a Thin Quad Flat Pack (TQFP) with 144 pins where 212 are user programmable I/O pins. The programmable logic configuration is based on Static Random Access Memory (SRAM) and thus it is volatile, but the device has an integrated non-volatile FLASH memory to contain the users configuration data. The CPLD will program itself from the internal FLASH memory as part of the power-up cycle and hence it is not necessary to reprogram the device manually after a power cycle. The configuration data is loaded into the CPLD with the JTAG interface. [23]

A.3.5 Data analysis and presentation

The data stream from the detector is received by a standard desktop computer. The USB interface of the computer stores the received data in a small buffer that has to be read out frequently by software on the computer to prevent overflow in the buffer.

LABVIEW is a software package by National Instruments that provides an easy programming interface to build data acquisition applications on a standard computer. The package provides modules that gives the user easy access to many of the computers communication ports. For the CRT application, LABVIEW even provides a module that interfaces with RS232-to-USB converter, completely hiding away the USB and other underlaying protocols and hardware. This was another good reason for utilizing the converter in the readout electronics.

A prototype DAQ application has been designed and tested with LABVIEW. The application is able to receive data from the detector and also send a control byte to the digital readout electronics. The received data can be written to ASCII text files so that data can be exported to other workstations for analysis. The firmware, including the control byte, is presented and discussed in the next section.

In future exercises, students will be able to easily design and program their own data acquisition and analysis applications with LABVIEW. For more advanced analysis the ROOT framework will be used. ROOT is C++ based framework developed at CERN specifically designed for particle physics analysis [24].

A.4 Firmware implementation

A.4.1 Overview

The CPLD will receive 16 channels where pulses of 200 ns in length, will arrive at random in time and phase. A logic high indicates a hit in the scintillator for the corresponding channel. The inputs are sampled synchronously at 10 MHz which is also the system clock frequency. If there is a hit in one channel then the samples for all channels, a 16 bit word, will be captured. For each 16 bit word that is captured a 30 bit time stamp should be appended. The time stamp is generated by a counter driven by the system clock.

All the captured data will be transferred to a computer. USB has become widely supported by desktop computers and provides faster data transfer rates than the traditional serial port. For this reason and because there was a RS232-to-USB converter available, the USB interface was chosen as data transfer media. The converter makes it easy to implement a USB interface as the RS232-protocol is simple and straightforward.

The firmware implements a RS-232 UART for communication with the computer via the converter. The transmitter will be used to push data whenever there are words in the FIFO. The receiver is used to receive control bytes from the computer. By toggling some bits in the control byte several commands can be sent to the firmware.

Bit 1	CLK_RST	Setting this bit high will reset the counter that generates the timestamp.
Bit 2	ACQ_START	Setting this bit high enables the data acquisition.
Bit 3	ACQ_STOP	Setting this bit high disables the data acquisition.
Bit 4	FIFO_OUT_CLEAR	Setting this bit high will reset/clear the FIFO/buffer.
Bit 5	CAL_TRIG	Setting this bit high will generate a calibration trigger.

Bits that are zero do not do anything. If both ACQ_START and ACQ_STOP bits are high in the same control byte the data acquisition will be disabled. The CAL_TRIG command will trigger a readout even if the data acquisition is disabled.

A.4.2 Implementation

In figure A.3 the 16 channels enter the firmware as the signal named `data_in`. In the `input_ctrl` module the signal is run through two synchronizer registers so that all values have valid logic values before they are evaluated by any logic. The synchronized data is routed to the 16 lowest bits of the FIFO. A 16 bits OR gate will generate a trigger if there is a hit in one of the channels, the `fifo_full` is low and the `enable` signal from the `myon_ctrl` module is high. The `trig` will cause the write request of the FIFO to be asserted as can be seen in the figure.

The `l_fifo_full` is a latched version of `fifo_full`. When the `fifo_full` is asserted the signal is latched until a new word has been written to the FIFO. When the new word is written the MSB will contain a one to indicate the FIFO full warning. This is to let the user who receives the data know that the FIFO has been full and that one or more words most likely have been lost.

The counter increments each clock cycle and will count up to $2^{30} \approx 1.07$ billion before starting over again. At 10 MHz clock this equals to about 107 seconds. The 30 bit word `time_stamp` will be written to the FIFO whenever `smpl_data` is written. The counter can be reset by sending a control byte with the CLK_RST bit high to the `myon_ctrl` module.

The `myon_ctrl` receives the control byte and generates control signals accordingly as can be seen in figure A.3. The control byte is received from the `rx_uart` module. This module is a serial receiver that decodes the frame received from the RS232-to-USB converter. The serial data signal from the converter enters the firmware as the signal named `rxsd` in the figure. When a byte has been successfully received the `rx_data_av` signal is asserted and the data is available on the `rx_data` output.

The `tx_uart` module is a serial transmitter that generates a serial signal `txsd` that is output to the RS232-to-USB converter. The module will load the 8 bit signal named `tx_data` in the figure and start the transmission of the byte when `tx_load` is asserted. While transmitting the module is busy and will not load new data. To indicate this state the module raises `tx_busy`.

The word from the FIFO is 48 bits in length and so it has to be transmitted as six bytes by the `tx_uart` module. The `myon_roc` is a readout controller that pops a word from the FIFO and feeds slices of 8 bits of the 48 bits at a time to the serial transmitter. The module is a state machine that controls a MUX to select the different slices. The FIFO will assert `fifo_empty` when there are no words to pop. If it is not empty the `read_req` signal can be asserted and a new word will be available on the output of the FIFO the next clock cycle.

A.5 Testing

A.5.1 Simulation - Formal verification

ModelSim is a software tool that can simulate a VHDL design. The simulator will move forward in time and simulate all signals and interactions. All of the VHDL source code has been simulated with ModelSim. Both individual testing of modules and the entire design. A module or entity has inputs and outputs. The testing procedure involves setting the input signals and monitoring the outputs, or in other words, creating stimuli and verify the response. This procedure is usually automated by a testbench, that can be written in VHDL or other programming languages supported by the simulator.

Testbenches for all modules have been written in VHDL that automatically generate stimuli and verifies the response. A testbench for the UART modules connects the serial out of the transmitter to the serial in of the receiver and verifies that a byte is correctly transferred.

Testing the complete design is more complex. Many features can only be tested indirectly but still the same principles apply. An input generator creates pulses of fixed length and period. The generator will create a pulse for one channel at a time and increase the channel number for each pulse so that a pattern is created. The testbench will also send some control bytes with a transmitter UART module so that the corresponding response can be observed. The serial output from the design is decoded by receiver UART and written to log file called `sim_myon_out.txt`.

The following is an excerpt from the log file created by the testbench.

```
1 00000000_00000000_00000000_00000000_10011101_11111110_10101111_
2 00000000_00000100_00000000_00000001_00011001_01000100_10101111_
3 00000000_00000100_00000000_00000001_00011001_01001000_10101111_
4 00000000_00001000_00000000_00000001_10100101_11100100_10101111_
5 00000000_00001000_00000000_00000001_10100101_11101000_10101111_
6 00000000_00010000_00000000_00000010_00110010_10000100_10101111_
7 00000000_00010000_00000000_00000010_00110010_10001000_10101111_
```

The first two columns are the bits indicating hits in the channels. The next 30 bits are the timestamp and the two last bits in column 6 is the two status-bits, calibration trigger and fifo full warning. The seventh column is a *separator* or *end-of-word* byte that is sent at the end of each transmitted word from the FIFO. If the lines above are compared with the input of the FIFO in figure A.3 one can see that each line is word in the FIFO but listed from LSB to MSB.

At line 1 of the excerpt the data indicates no hits in any of the channels, however bit 7 in the sixth column is one which indicates a calibration trigger was present when the data was captured. This is consistent with the expected result since the testbench sends a control byte with the CAL_TRIG bit high.

Line 2 and 3 indicates a hit in one of the channels and hence the data has been captured and transmitted correctly. One can see the timestamp has increased and that it increases with only one from line 2 to line 3. This is consistent with the fact that a pulse of 200 ns in length will be detected in two consecutive clock cycles.

Further, in line 4 and 5, and lines 6 and 7 one can see similar hits in other channels where the timestamp has increased. Looking at a larger excerpt from the log file one can clearly recognize the pattern in the channel numbers that have hits and that it is consistent with the input generated by the input generator.

The testbench for the muon firmware design is not fully automatic in the sense that output has to be inspected manually to discover any errors and deviations from the expected result. This is only a formal verification of the source code for the design.

A.5.2 Synthesis and post-synthesis simulation

The firmware was synthesized with the Precision software from Mentor Graphics. Precision produces VHDL code which describes the netlist of the synthesized design and a file with timing information. These design files has been simulated with the origianl testbench in ModelSim with timing information. The result is compared with the result from the simulation of the VHDL source code. This verifies that the design has been synthesized correctly and that the timing constraints are met.

A.5.3 Testing in real system

The firmware design has been tested in hardware. It has been verified that one is able to command the firmware properly and that signals will be captured and transferred to the computer.

The complete readout chain with scintillators and PMTs have been tested with three channels. There are still some problems with the hardware of the rest of the channels.

During testing of the communication link with the computer the baudrate/bitrate fo the RS-232 interface was experimented with. The baudrate was finally set to 115 k which should be plentyfull as the expected trigger rate is a few Hertz.

A.6 Conclusion

With this work I have managed to successfully develop and integrate a firmware design. All modules written by me in VHDL except for the `tx_uart` and `rx_uart` modules by Ketil Røed and the FIFO which was generated by the Quartus Megafunction wizard.

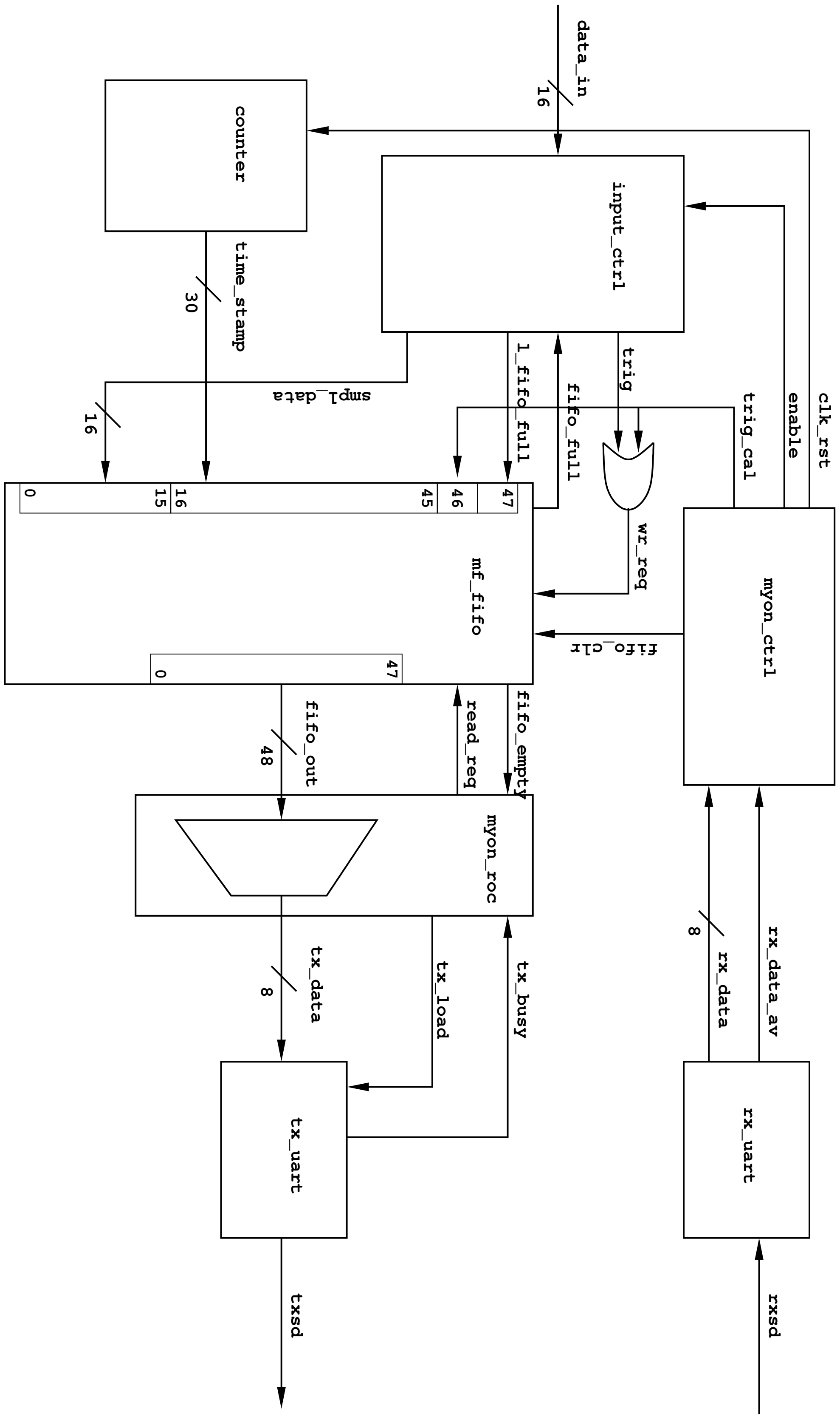

Figure A.3: Overview of architecture of the muon firmware.

Appendix B

FPGAworld 2007 article

The article in this appendix was published in the proceedings of the FPGAworld conference.
ISSN 1404-3041 ISRN MDH-MRTC-215/2007-1-SE

The conference was held in Lund and Stockholm in Sweden, on the 11th and 13th of September, respectively. I was present on the conference in Stockholm to give a presentation based on the article.

Busy Generation in a large Trigger Based Data Acquisition System

M. Munkejord* A. Stangeland* J. Alme* W. Rauch† M. Richter*

A. Rossebø* D. Röhrich* C. Soos‡ K. Ullaland*

Abstract

This paper gives an overview of a specific trigger and data acquisition system used in experimental nuclear physics, and describes one of its many components, which generates the busy signal. It is a FPGA based device that continuously keeps track of the number of issued triggers and computes the number of free buffers in the Front End Electronics.

I Introduction

The Large Hadron Collider at CERN accelerates two separate, circular beams of nuclei. The two beams move in opposite directions and at four points they intersect, allowing for collisions. ALICE (A Large Ion Collider Experiment) [1] is placed at one of these points and comprises several detectors. Recording and transfer of event data will be controlled through trigger signals, which are based on inputs from fast detectors. The Time Projection Chamber (TPC) [2] is one of the main tracking detectors in ALICE. It has approximately 560000 channels and generates data at a rate of up to 25 GB/s. For Lead-Lead collisions the maximum interaction rate will be about 8 kHz. In proton-proton collisions it will be higher, about 200 kHz in ALICE. Not all interactions will be recorded and kept for later analysis, and it is the trigger system that controls which. On average the collision rates will therefore be somewhat higher than the transfer rate to the Data Acquisition System (DAQ). For this reason the detector Front End Electronics (FEE) [3] has some buffer memory. To prevent overflow in the FEE buffers, a mechanism to halt the issuing of new triggers is required. This is what is referred to as busy generation and will be provided by a dedicated device called the Busy Box. The Busy Box will be used in several of the detectors of ALICE and it is the subject of this paper.

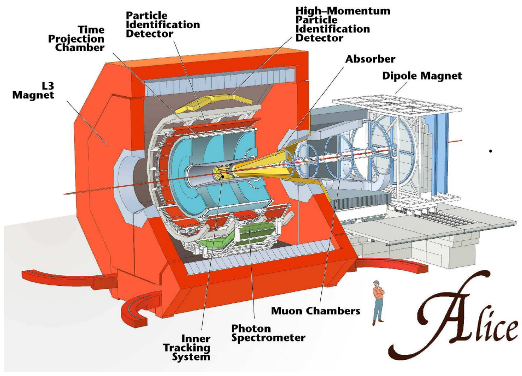

Figure 1: Illustration of ALICE

II The Trigger System

ALICE has one Central Trigger Processor (CTP) [7]. It receives information from all sub detectors and makes decisions on what triggers to issue. All

[*]Department of Physics and Technology, University of Bergen, Norway

[†]University of Applied Sciences, Frankfurt, Germany

[‡]CERN, European Organization for Nuclear Research, Geneva, Switzerland

triggers are forwarded to the Local Trigger Units which distribute them to the FEE over an optical fiber channel. The global system clock will be distributed over the same fiber. This clock signal drives all of the digital electronics in the detector and runs at the nominal bunch crossing rate of 40.08 MHz. The clock is also used as reference when creating Event IDs for collisions. Event IDs will be distributed with the triggers and makes it possible to compare data from different sub detectors when analyzing events. Event IDs also play an important role in the busy handling, as will be explained later.

The hardware trigger system for ALICE has three levels - Level 0, Level 1 and Level 2, and they are issued in sequence. A trigger sequence is started by a Level 0 trigger, which will be issued once a collision has been detected. Some time after that a Level 1 trigger will be issued if the collision satisfies certain conditions. If not the Level 1 trigger is suppressed, the trigger sequence is aborted and any data recorded so far discarded. Provided a Level 1 trigger was issued, a Level 2 trigger will be issued. The Level 2 trigger will indicate whether the event was accepted or not. If the event was accepted the FEE will mark the data in its buffers for transmission to the DAQ system. The DAQ system will receive the event data whenever there is capacity available. If a Level 2 Reject trigger is issued then FEE will overwrite its buffer when new triggers are received.

The TPC is constructed like a barrel filled with gas (see figure 1). When particles from a collision travels through the TPC, they will ionize the gas in their path leaving a trail of ionized atoms. Electric fields will cause the freed electrons to drift towards the ends of the barrel where they can be detected. To fully record an event the TPC requires about 90 μs. This makes the TPC a slow detector and new collisions can occur while there still are drifting electrons from a previous collision. However, during analysis one is able to distinguish up to a certain number of events, the number depending amongst other things on the quality of the reconstruction algorithms. If too many collisions occur after a trigger has been issued, the CTP will issue a Level 2 Reject trigger and the data will be discarded as explained earlier. This feature is called the past-future protection and is meant to discard data from events that can not be analyzed.

III Busy Handling

The task of the Busy Box is to let the trigger system know when the detector is busy and can not handle new trigger sequences. As long as the busy signal is asserted, the CTP will not issue additional trigger sequences. The generation of the busy signal is a logical OR between two separate processes inside the Busy Box. One is a simple timer started whenever a Level 0 trigger is received. In the case of the TPC the timer is set to approximately $90\mu s$, which is the time it takes to record one event. The other process will flag busy when all buffers on the FEE are occupied.

If a Level 2 Accept is issued for an event, the FEE will tag the data with the Event ID and push it over optical fibre links to DAQ computers. These are regular PCs with special data adapters called D-RORCs (DAQ-Read Out Receiver Card) connected to a PCI bus. Instead of communicating directly with the FEE to find the number of buffers in use, the Busy Box queries the D-RORCs. Once a D-RORC has received the data for an event from the FEE, it extracts the Event ID and transmits it upon request to the Busy Box over LVDS lines. The Busy Box also extracts the Event ID from the Level 2 Accept trigger, but stores it in a local queue. Once an Event ID enters the queue, the Busy Box will start polling the D-RORCs and compare the Event ID from the trigger with that from every D-RORC. If all the Event IDs match it can be safely assumed that all the corresponding buffers are freed, and the used buffers counter will be decremented. In this way the number of free FEE buffers can be calculated indirectly.

Traditionally the FEE in the detectors has generated its own busy signals. For the TPC alone, however, there are more than 4000 Front-End Cards but only 216 D-RORC cards. Communicating with the D-RORCs therefore significantly reduces the need for connections. Also, the D-

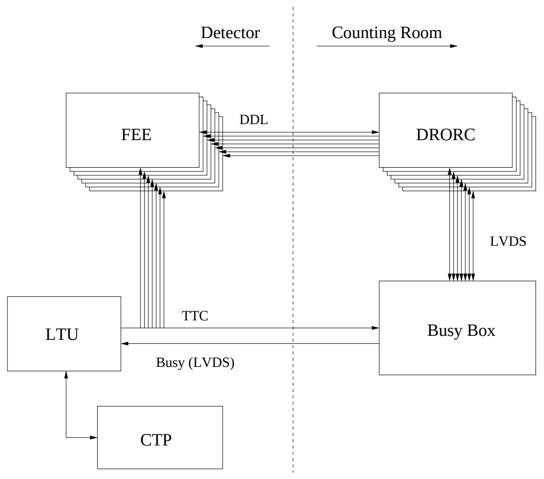

Figure 2: Illustration of Busy Box concept

RORC cards are placed in a counting room, away from the radiation environment close to the detector. By also placing the Busy Box in the same counting room easy access is assured.

IV Busy Box

In the case of the TPC, the Busy Box needs to communicate with the 216 D-RORCs over 15 meters TP (Twisted Pair) cables with RJ-45 connectors. Many of the other sub detectors have fewer data links and hence, fewer D-RORCs. For this reason the Busy Box is made modular (see figure 3). The motherboard has 40 ports for RJ-45 connectors. If more ports are needed, mezzanine cards with 48 ports can be attached with ribbon cables. The boards/cards are built in standard 19" rack cases up to five units in height. The logic resources are provided by one or two Virtex-4 FPGAs, depending on the number of ports required. The Virtex-4 FPGA in the ff1148 package was chosen because it has many IO pins, supports LVDS and supports programming by SelectMAP [6].

Attached to the motherboard is a DCS card. The DCS card is part of the DCS (Detector Control System) which monitors, configures and controls most of ALICE. The DCS card is mainly composed of an Altera EPXA1 (containing a 32 bit ARM processor), 8 MB Flash ROM, 32 MB SDRAM and an Ethernet transceiver. With these components it is able to run a lightweight version of Linux. Device drivers for Linux have been developed so that programming the FPGA with SelectMAP from a remote location is possible. This feature, although very handy for the Busy Box, was initially developed for the FEE which resides inside the detector and is unreachable once the accelerator has been started. The DCS card also has a 16 bit wide bus interface to both FPGAs, allowing software to access memory mapped registers inside the FPGAs. The DCS board provides connectivity to the trigger system and the Detector Control System.

The main requirements for the firmware are to provide communication with the D-RORCs and an interface to the DCS bus and triggers. In addition it will do most of the work of processing the incoming messages from the D-RORCs. It is essential to implement as many of the low-level functions in firmware as possible since it is faster than the software. There will be two versions of the motherboard, with one or two FPGAs. The first FPGA is connected to the first 120 of the RJ45 ports and the second to the 96 remaining. Since the number of ports will vary for different Busy Boxes, the firmware is designed to be scalable at compile time (by generics) to include any number of ports from 1 to 120. The two FPGAs will operate in parallel, with some simple logic in the first FPGA to coordinate the busy-signal.

Every received message from the D-RORCs will be stored in memory that is available to the

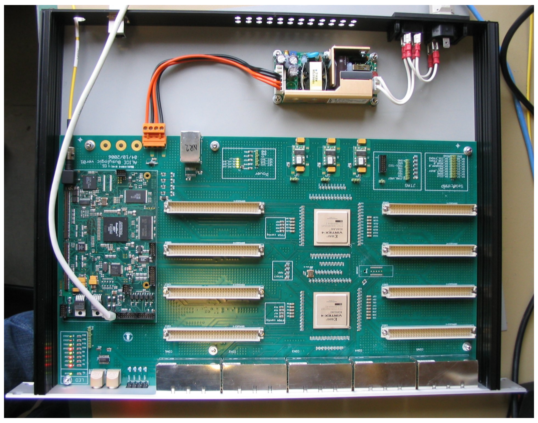

Figure 3: Picture of the inside of the Busy Box

software. The firmware also provides registers for transmitting messages to any or all of the connected D-RORCs. This allows software to communicate with the D-RORCs directly so that higher level error handling can be done in software. It is also very useful for debugging in the development phase.

Messages from the D-RORCs will also be pushed into a FIFO queue for processing by the firmware. Internal status registers for each D-RORC will be updated as messages are processed. The information in these registers will be used to determine when all D-RORCs have received data for the current Event ID or, if not, determine the next appropriate action.

As described earlier, the Busy Box will request the Event IDs from the D-RORCs. It will receive messages from all active D-RORCs containing the requested Event ID or a message saying that the Event ID has not been received yet. The Busy Box will wait until it has received messages from all D-RORCs or until a programmable timeout runs out and then re-request from those that had not received the Event ID. The firmware will retry this procedure a few times before it sets appropriate error registers and allows software to resolve the error or report it to the DCS.

The communication logic on both sides (Busy Box and D-RORC) runs on 200 MHz. Dedicated hardware inside the Virtex-4 called Digital Clock Managers are used to generate this clock in the Busy Box. The D-RORCs are referenced to the clocks of their host computer. This means that the two devices do not share clock source and clock skew and jitter noise is to be expected. A protocol that includes a bit clock in the encoded signal is desirable but due to the large number of receivers that have to be implemented into a single FPGA, Non-Return-to-Zero encoding is used. Currently, the receivers utilizes 5x oversampling which gives a bit rate of 40 Mbps. The receivers will push samples into a shift register long enough to contain samples for a complete word. When the receiver sees valid start and stop bits in the samples, it will use majority gates to determine the value of each bit and store the resulting bits in an output buffer. Parity checks are also implemented to maintain data integrity. A message from a D-RORC to the Busy Box is 48 bits. To make the protocol more tolerant of jitter and keep the receivers small, the 48 bits are transmitted as 3 times 16 bit words (with a very short timeout between the words). This allows the receiver to resynchronize to the bit stream more often. It also reduces the probability that noise from floating inputs produce garbage data by accident because it is less likely that this noise will produce three valid words consecutively.

V VERIFICATION

The design has been tested in simulations with the QuestaSim software. For this purpose testbenches has been written in VHDL that emulates the devices that the Trigger Busy Box firmware will interface with. For some of the emulated devices, a dedicated VHDL entity has been written, others are emulated by VHDL procedures that drives the signals of the interface. A main test sequence process calls procedures that controls the emulators to interact with Busy Box firmware. The main test sequence can easily be modified to simulate specific scenarios. The testbench does not automatically verify the result but gives the opportunity to study the functional operation of the design in operation.

The first priority of the hardware tests was to verify a reliable communication between the Busy Box and the D-RORC. Several test setups have been used in the different stages of development. The first was a *loopback* test where the Busy Box transmitted messages to itself through a TP cable. By using the DCS board to access registers of the FPGA, messages can be sent, and the received messages can be read out and verified by software.

After some modifications to the firmware the Busy Box was brought to CERN for testing with the D-RORC. These tests were concluded with a "proof-of-concept" test where software running on the DCS board controlled the communications of the Busy Box. The test included retrieval of an event ID form the D-RORC and successfully comparing it with the event ID which was sent from the LTU in emulator mode.

Further test of the communication has been performed with another FPGA based device were firmware have been developed specifically to emulate the D-RORC in the absence of the real D-RORC and the remaining components of a real test setup.

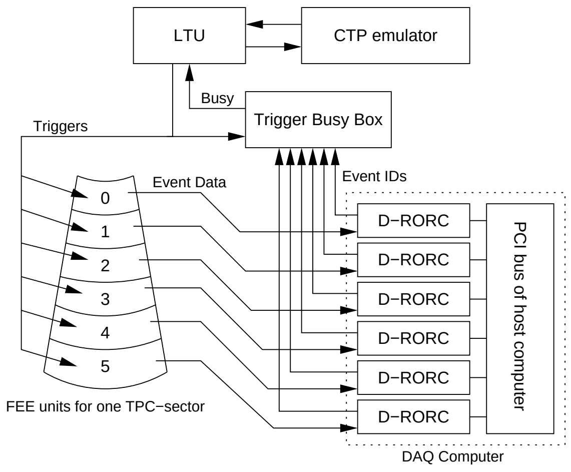

Figure 4: Illustration of test setup.

Integration tests at CERN

Recently tests have been performed with the current Busy Box design at CERN in a real test environment, including real components and several channels. The setup is illustrated in figure 4. On the detector side four complete FEE-units have been used simultaneously (the TPC has a total of 216 FEE-units). The FEE sampled floating inputs instead of real detector signals to simulate data, and on the trigger side a real LTU has been used. The LTU receives the BUSY-output from the Busy Box and passes it on to the CTP. Trigger inputs to the LTU will come from the CTP in the final setup, but so far a CTP emulator has been used instead. The CTP emulator will issue triggers at a variable rate, as is expected in the real system under normal operation. In the test the Busy Box verifies that all event data has been successfully transfered to the D-RORCs by comparing the Event IDs from the D-RORCs with the Event IDs from the trigger system.

VI Conclusion

The modular design of the Busy Box and its scalability makes it possible to use it with several ALICE sub-detectors. This also allows independent testing of different functionalities and makes it easy to add new or modify existing ones. Both during development and integration of the Busy Box in a detector system, the combination of software and firmware gives flexibility. So far laboratory tests have been performed to verify basic functionality, and error handling will be added. Further commissioning tests using more channels will be performed in the near future.

References

[1] ALICE Collaboration, *Technical Proposal For A Large Ion Collider Experiment at the CERN LHC.* CERN/LHCC 1995-71, 1995.

[2] ALICE Collaboration, *Technical Design Report of the Time Projection Chamber*, CERN/LHCC 2000-001, ALICE TDR 7, 7 January 2000. ISBN 92-9083-155-3 https://edms.cern.ch/file/398930/1/ALICE-DOC-2003-011.pdf

[3] L. Musa et al., *The ALICE TPC Front End Electronics*, in proc. of the IEEE Nuclear Science Symposium, Portland, October 2003.

[4] ALICE Collaboration, *Technical Design Report of the Photon Spectrometer (PHOS)* CERN/LHCC 99-4, ALICE TDR 2, 5 March 1999. ISBN 92-9083-138-3 https://edms.cern.ch/file/398934/1/Cover-Contents.pdf

[5] Rossebø Anders, *BUSY-logikk for ALICE TPC*, Master thesis, University of Bergen, 2006.

[6] Xilinx Inc., *Virtex-4 User Guide v.1.5*, January 2006.

[7] D. Evans, S. Fedor, G. T. Jones, P. Jovanović, A. Jusko, I. Králik, R. Lietava, L. Šándor, J. Urbán and O. Villalobos Baillie for the ALICE collaboration. http://lhc-workshop-2005.web.cern.ch/lhc-workshop-2005/ParallelSessionB/51-OrlandoVillalobosBaillie.pdf

[8] ALICE collaboration, *Technical Design Report of the Trigger, Data Acquisition, High-Level Trigger and Control System*, CERN-LHCC-2003-062, ALICE TDR 010, CERN, 2004. ISBN 92-9083-217-7. https://edms.cern.ch/document/456354/

[9] Wiki-page of the Experimental Nuclear Physics group at the Department of Physics and Technology at the Univerity of Bergen: http://web.ift.uib.no/˜kjeks/wiki/

[10] J. Alme, *TTC receiver requirement specification v1.1*, University of Bergen, 02.03.2007.

Appendix C

Abbreviations

ADC Analog to Digital Converter

ALICE A Large Ion Collider Experiment

ALTRO ALICE TPC ReadOut

ARQ Automatic Repeat reQuest

ASIC Application Specific Integrated Circuit

BC Bunch Crossing

CDH Common Data Header

CERN European Organisation for Nuclear Research

CHEN CHannel ENable

CLB Configurable Logic Block

CPLD Complex Programmable Logic Device

CPU Central Processing Unit

CRT Cosmic Ray Telescope

CTP Central Trigger Processor

D-RORC DAQ Read Out Receiver Card

DAQ Data Acquisition

DCM Digital Clock Manager

DCS Detector Control System

DDG DDL Data Generator

DDL Detector Data Link

DIU Destination Interface Unit

DMA Direct Memory Access

EDA Electronic Design Automation

EIDOK Event ID OK

FEC Front End Card

FEE Front End Electronics

FIFO First-In-First-Out

FPGA Field Programmable Gate Array

FSM Finite State Machine

GDC Global Data Concentrator

HDL Hardware Definition Language

HLT High Level Trigger

HMPID High Momentum Particle Identification Detector

IC Integrated Circuit

IEEE Institute of Electrical and Electronics Engineers

IRQ Interrupt Request

ITS Inner Tracking System

JTAG Joint Test Action Group

LAN Local Area Network

LDC Local Data Concentrator

LED Light Emitting Diode

LHC Large Hadron Collider

LSB Least Significant Bit

LTU Local Trigger Unit

LUT LookUp Table

LVDS Low Voltage Differential Signaling

MSB Most Significant Bit

MWPC Multi-Wire Proportional Chamber

NRZ Non-Return-to-Zero

OS Operating System

PASA Preamplifier and Shaper ASIC

PCB Printed Circuit Board

PCI Peripheral Component Interconnect

PC Personal Computer

PHOS Photon Spectrometer

PISO Parallel-In-Serial-Out

PLL Phase Locked Loop

PMCD Phase Matched Clock Divider

PMT PhotoMultiplier Tube

PROM Programmable Read Only Memory

QGP Quark-Gluon-Plasma

RAM Random Access Memory

RCU Readout Control Unit

RORC Read Out Receiver Card

SDRAM Synchronous Dynamic RAM

SIU Source Interface Unit

SRAM Static Random Access Memory

TCP/IP Transmission Control Protocol/Internet Protocol

TOF Time Of Flight

TPC Time Projection Chamber

TP Twisted Pair

TQFP Thin Quad Flat Pack

TRD Transition Radiation Detector

TTC Timing, Trigger and Control

USB Universal Serial Bus

UiB University of Bergen

VHDL VHSIC Hardware Definition Language

VHSIC Very High Speed Integrated Circuits

Bibliography

[1] CERN. CERN Public web pages. http://public.web.cern.ch.

[2] Håvard Helstrup. Generell informasjon om kjernefysikk. http://web.ift.uib.no/∼kjeks/mer.html.

[3] ALICE Collaboration. ALICE public web pages. http://aliceinfo.cern.ch/Public/panorama/ALICE_EYES/INSIDE_ALICE/index2.html.

[4] ALICE TPC. ALICE TPC web pages. http://aliceinfo.cern.ch/TPC/index.html.

[5] Jørgen A. Lien. *The Readout Control Unit of the ALICE TPC*. PhD thesis, University of Bergen, Bergen, Norway, 2004.

[6] CERN. *ALICE TPC Readout Chip User Guide*, DRAFT 0.2 edition, June 2002. UserManual_draft_02.pdf.

[7] Introduction to DDL. http://ph-dep-aid.web.cern.ch/ph-dep-aid/.

[8] Introduction to RORC. http://ph-dep-aid.web.cern.ch/ph-dep-aid/.

[9] The ALICE Collaboration. Alice technical design report. Technical report, CERN, 2004. ISBN 92-9083-217-7.

[10] Olav Torheim. Implementasjon av interrupt-styrd DMA-overføring på HLT-RORC. Master's thesis, University of Bergen, September 2005.

[11] Csaba Soos. Mail correspondence.

[12] Anders Rossebø. Busy-logikk for alice tpc. Master's thesis, University of Bergen, June 2006.

[13] Johan Alme. TTC receiver requirement specification, 2007.

[14] Luciano Musa. ALICE TPC trigger system questionnaire, February 2007.

[15] Xilinx Inc. *Virtex-4 Configuration Guide*, v1.7 edition, July 2007. UG071.pdf.

[16] Jon Kristian Bernhardsen, Jan Martin Langeland, and Stian Skjerveggen. Hovedprosjekt - selectMAP kernel module, 2006.

[17] Xilinx Inc. *Virtex-4 User Guide*, v2.2 edition, April 2007. UG070.pdf.

[18] Neil H. E. Weste and David Harris. *CMOS VLSI Design*, chapter 7.7. Pearson Education, Inc., third edition, 2005. ISBN 0-321-26977-2.

[19] Nick Sawyer. Data recovery. Xilinx application note, 2005. http://www.xilinx.com/.

[20] Ken Chapman. Multiplexer selection. Xilinx TechXclusives, January 2004. http://www.xilinx.com/.

[21] Wolfgang Rauch. Busy generation for the ALICE DAQ, 2006.

[22] Ken Chapman. Get your priorities right - make your designs up to 50% smaller. Xilinx TechXclusives, July 2004. http://www.xilinx.com/.

[23] Altera Corporation. *MAX II Device Handbook*, MII5V1-2.1 edition, December 2006. max2_mii5v1.pdf.

[24] The root system homepage. http://root.cern.ch/.