Firmware Upgrade and Testing of ALICE BusyBox

Rikard Bølgen

Master Thesis

Department of Physics and Technology

University of Bergen
02 June 2009

"Particle physics is the unbelievable in pursuit of the unimaginable. To pinpoint the smallest fragments of the universe you have to build the biggest machine in the world. To recreate the first millionths of a second of creation you have to focus energy on an awesome scale."

The Guardian

Abstract

A Large Ion Collider Experiment (ALICE) is a general purpose heavy–ion experiment at CERN. It is a detector designed to study the physics of strongly interacting matter and quark-gluon plasma in nucleus–nucleus collisions produced in the Large Hadron Collider (LHC). The detector is built-up of sub-detectors which are studying different aspects of the heavy-ion experiment. ALICE is an international collaboration where the Department of Physics and Technology at the University of Bergen is among the 31 countries and 109 institutes participating.

Huge amounts of data from the collisions in the LHC is produced every second pushing the limit of the state-of-the-art Data Acquisition systems (DAQ) in ALICE. The ALICE Online system reduces the overall data rate to fit the event building bandwidth and storage capability of the DAQ system. The Timing, Trigger and Control (TTC) is a part of the Online System and has a Central Trigger Processor (CTP) designed to select events containing potential interesting physics, scaled down to fit the restrictions imposed by the bandwidth of the DAQ system.

The Time Projection Chamber (TPC), Photon Spectrometer (PHOS), Forward Multiplicity Detector (FMD) and the Electro Magnetic Calorimeter are sub-detectors which utilize a BusyBox to tell the TTC when it is ready to take event data. The Front-end electronics (Fee) of these four detectors can only buffer 4 or 8 events, and the BusyBox keeps track of the number of used buffers. If the buffers are full, the BusyBox asserts a *busy* signal to the TTC system which prevents new triggers from being issued.

This thesis describes the functionality of the BusyBox, along with development, testing and upgrades of the firmware.

The BusyBox is a Field Programmable Gate Array (FPGA) based system containing a Detector Control System (DCS) board running a light weight version of Linux with an Ethernet interface and an LVDS interface to the DAQ and CTP system. The firmware is written in VHDL and has a DCS bus module interface where internal control and status register can be read or written to.

Acknowledgments

The work for this thesis has been carried out at the Department of Physics and Technology at the University of Bergen from fall 2008 to spring 2009. It has been an interesting and enriching time where much has been learned, and there are some people who deserve my gratitude.

First and foremost I want to thank to my teaching supervisor, Kjetil Ullaland, for his assistance, support and guidance concerning this thesis. Thanks to Dominik Fehlker for guidance and help from start to end of my master thesis. His insight in the ALICE project has been indispensible for me. Thanks to Håvard Helstrup for helping out with installing and debugging the DATE system. Johan Alme, Magne Munkejord, Dieter Röhrich, Csaba Soos and Lijiau Liu deserve credits for helping me with problems related to the BusyBox.

In addition to those mentioned I want to thank my fellow students Jone Tveiten, Lars Roger Solem, Christian Erstad, Anja Kohfeldt and Yngve Skogseide at room 446 for creating a good working environment and for time well spent.

Finally, I wish to thank my parents.

Contents

Chapter 1

ALICE - A Large Ion Collider Experiment

This chapter gives a brief overview of the ALICE experiment and the different scientific instruments at the Large Hadron Collider (LHC), before ALICE is discussed in more detail. This includes the ALICE physics, sub-detectors, online system and data flow.

1.1 Introduction

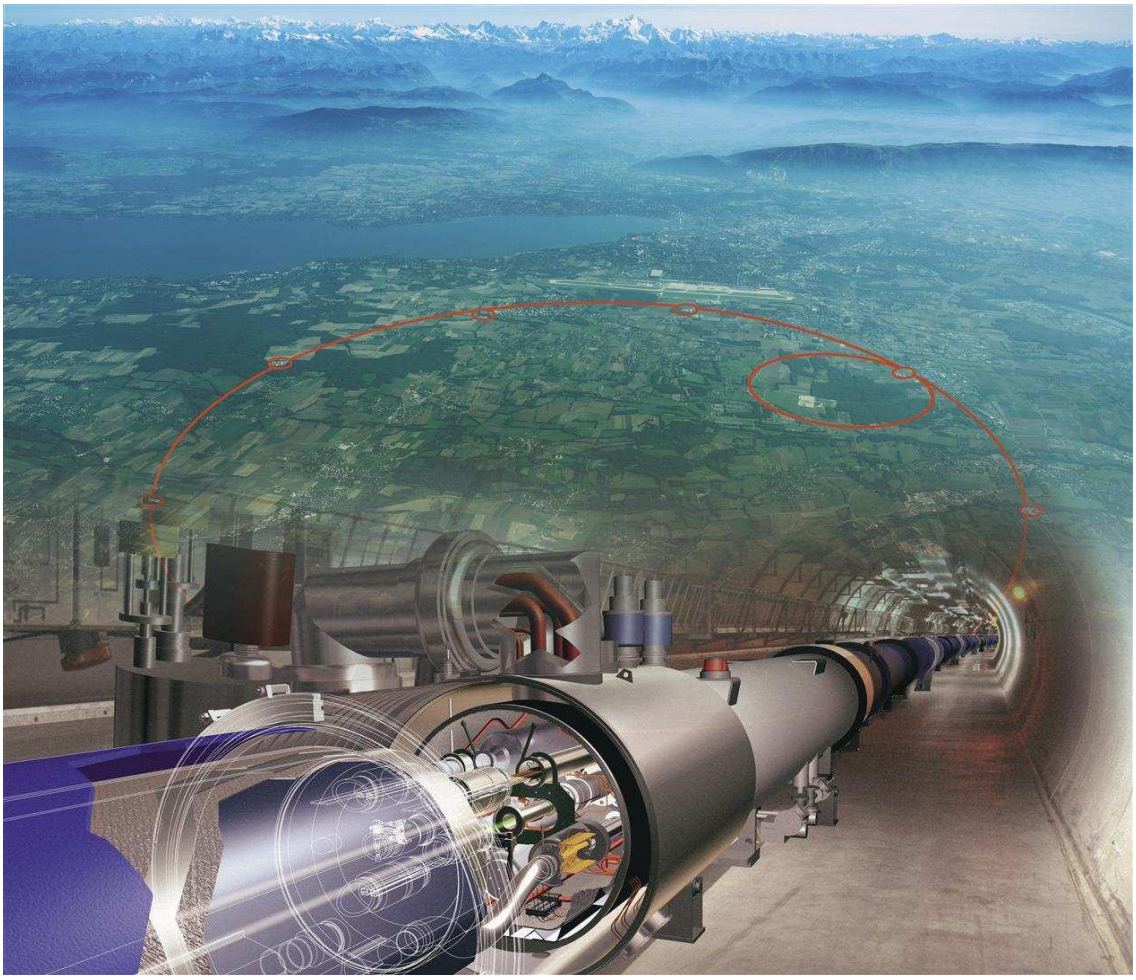


Figure 1-1: Illustration of the LHC on the border between Switzerland and France, from [1].

On the border between France and Switzerland near Geneva lies the biggest scientific instrument in the world, the Large Hadron Collider [1] (LHC). Situated 100 meters underground and with a circumference of 27 km, it will be used by particle physicists to study the smallest known particles, how the universe evolved and how it works today.

CERN[1] (The European Organization for Nuclear Research) is the largest particle physics laboratory in the world, and its main function is to provide the particle accelerators and other infrastructure needed for high energy physics. The Large Hadron Collider (LHC) is what most of the activities at CERN are directed to.

LHC can create the conditions just a fraction of a second after the Big Bang by colliding subatomic particles called hadrons. In opposite directions, either protons or lead ions will be accelerated to 99.999999% of the speed of light inside the accelerator. The beam pipes are kept at ultra high vacuum surrounded by superconducting electromagnets cooled down to minus 271˚C. The electromagnetic field pushes the hadrons around and thousands of magnets are used to direct and focus the beams before the particles collide.

Particles collide at four points, where four main experiments are located:

- *ALICE (A Large Ion Collider Experiment)*: Designed to look for Quark Gluon Plasma (QGP).
- *ATLAS (A Toroidal LHC ApparatuS)*: Is mainly looking for the Higgs boson.
- *CMS (Compact Muon Solenoid)*: Investigating physics in the TeV domain.
- *LHCb (LHC beauty)*: Investigating physics related to the bottom quark.

There are six large detectors at these four points. Each of them will study particle collisions under a different view, and with different technology.

ALICE [1] (A Large Ion Collider Experiment) is one of them and is optimized to study heavy ion collisions which will generate quark-gluon plasma, a state of matter wherein quark and gluons are deconfined. ALICE consists of sub detectors in an onion–like structure around the collision point.

The Inner Tracking System (ITS), Time Projection Chamber (TPC), Transition Radiation Detector (TRD), Time of Flight (TOF), Photon Spectrometer (PHOS), High Momentum Particle Identification Detector (HMPID), Muon Spectrometer, Forward Multiplicity Detector (FMD), and the Electro-magnetic Calorimeter (EM-Cal) is the ALICE central barrel detectors[1].

The particles circulate in well defined bunches with a bunch spacing of 25 ns. This corresponds to a frequency of 40 MHz and a particle will make 11 245 circuits every second. Each beam has 3565 bunches and a bunch can contain 100 billion particles, but not all bunches will contain particles. During a proton run, 2808 of the 3565 bunches contains particles and only 608 particles during a lead run. The average crossing rate for lead is 6.8 MHz and 31.6 MHz for protons, but the maximum collision rate for Pb-Pb is 8 kHz and 200 kHz for p-p in ALICE [2].

[1] CERN is an acronym for Conseil Européen pour la Recherche Nucléaire. (http://cern.ch)

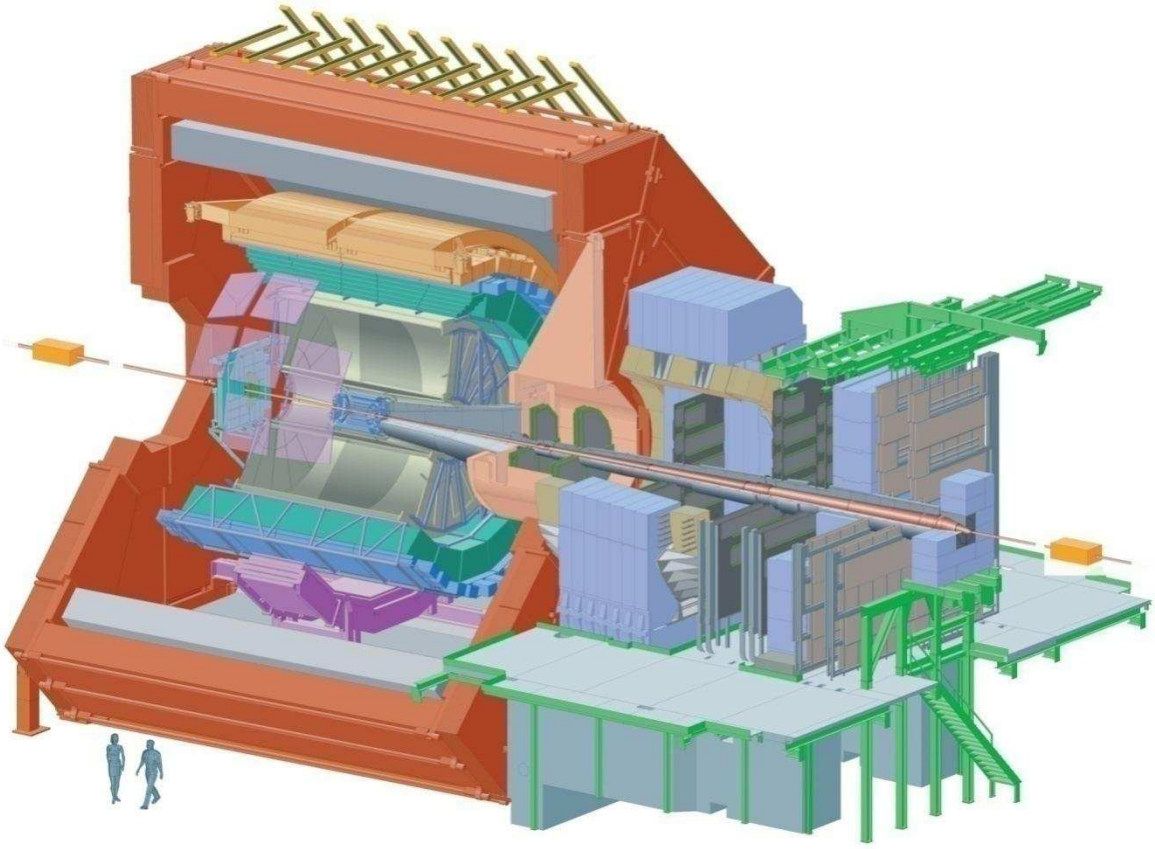


Figure 1-2: The ALICE detector. From

Particle collisions are called events. The collision energy for protons is up to 14 TeV and up to 1150 TeV for lead [1]. These energies have never been reached in a laboratory before, but in absolute terms it is not very high energies compared to everyday life. Clapping your hands will produce more collision energy. It is the energy concentration which makes particle collision so special.

When a collision occurs, a shower of particles will spread in every direction. Some will decay faster than others, hence the onion-like structure to detect them before they do.

In simple terms, the different detectors have different ways of detecting different things. But what is common for every detector, is that it converts what it detected to an electrical signal. This signal is then digitized and stored. But how are events detected? In addition to the main detectors in ALICE there are several sub detectors used to determine if an event has occurred. Not all collisions are of interest, only the good ones that are head on and generate a certain amount of energy. Based on information from the sub-detectors, a Central Trigger Processor (CTP) sends triggers to the sub-detectors Front-end electronics (Fee). The Fee starts buffering data upon a positive trigger, and sends it to the Data Acquisition system (DAQ). The Busy Box system is designed to monitor and verify the transfer of event data, and to prevent overflow in the Fee buffers.

1.2 A Large Ion Collider Experiment

A Large Ion Collider Experiment (ALICE) is a collaboration consisting of 32 countries with 109 institutes and more than 1000 members [1]. Together, physicist and engineers designed a general purpose heavy-ion experiment to study physics of strongly interacting matter and quark-gluon plasma in nucleus collisions at the LHC. Not only heavy systems will be studied, but also lower-mass ion collisions.

Figure 1-3: Member counties 2008. From [1]

To understand the phenomenon studied in ALICE, physicists need many different detectors to observe a collision from different views in order to get the whole picture. Hence, ALICE is constructed in an onion-like manner with its sub-detectors surrounding the collision point. The whole thing is wrapped in a huge magnet.

Triggers are very important in ALICE. The data readout starts when a central trigger system distributes triggers to the sub-detectors readout electronics. Events contain so much data that it is impossible to handle the data without state-of-the-art technology. Not all of the 18 detectors are of interest in this thesis since the Busy Box only utilizes TPC, PHOS, EMCal and FMD.

1.3 ALICE Physics

Physicists do not have a complete theory of how elementary particles and their fundamental interactions work. This theory is called The Standard Model and describes all particles that make up visible matter in the Universe, and three of the four fundamental interactions.

An atom consists of electrons orbiting the nucleus with protons and neutrons. Protons and neutrons are composite particles named hadrons. The hadrons again consist of quarks which are engaged in the strong interaction named gluons. Quarks are never found alone in nature and are always bound together in hadrons.

Quarks are about one hundredth of the mass of a proton or a neutron. Is the mechanism that confines quarks responsible for generating most of the mass of ordinary matter? By colliding heavy ions in LHC, quarks and gluons are no longer confined. They undergo a phase transition and Quark Gluon Plasma (QGP) is formed. The plasma is 100 000 times hotter than the sun, but expands and cools down after about 10^{-23} s and regroup to form ordinary matter. Only traces of the QGP can be detected. QGP is part of the effort to consolidate the grand theory of particle physics and ALICE will study the properties of QGP.

1.4 The ALICE Sub-Detectors

After an event, a wealth of information needs to be processed. The data readout in ALICE is based on triggers. The triggers are distributed by a central trigger processor system, and the detector electronics start to buffer and read out data upon receiving a trigger. Subsequently, data is sent to computer farms and analyzed before it is permanently stored.

TPC, PHOS, FMD and EMCal are the sub detectors which will be utilized for the Busy Box, and they will be discussed in the next sections.

1.4.1 Time Projection Chamber

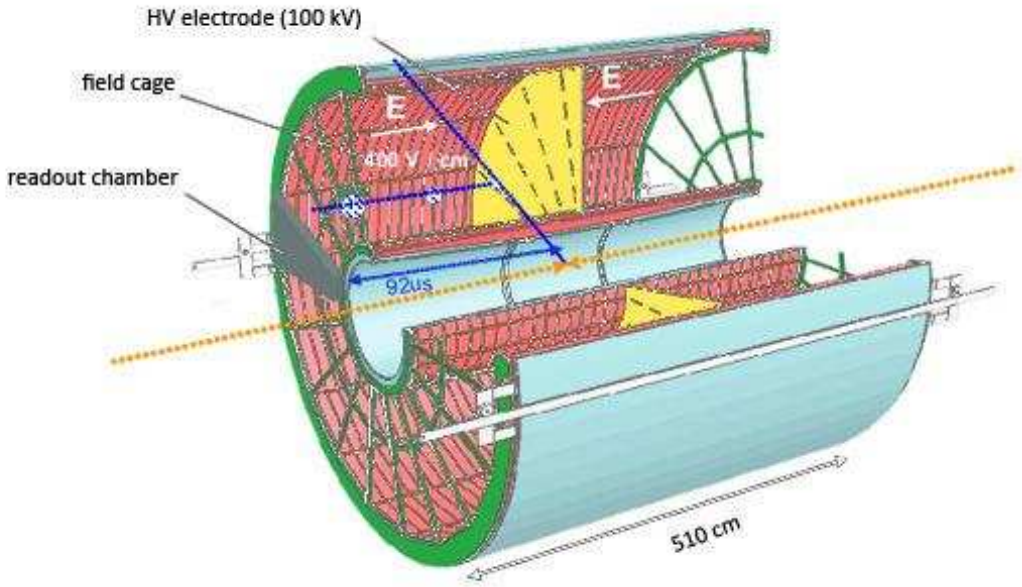


Figure 1-4: Layout of the ALICE TPC. From [3].

The TPC's [3] function is to provide trace finding, charge particle momentum, particle identification and two-track separation. It is basically a barrel filled with a compressed gas mixture with a high voltage (HV) electrode at its axial center. This creates two symmetric regions with high electrostatic fields in both drift directions. After an event, particles will leave a trace in the form of ionized gas and hit the readout chamber at a constant velocity. Depending on the electrical charge and the momentum of the particle, the trace will be bent stronger or weaker in either direction. About 560 000 electronic channels in 36 sectors, 18 on each end-cap of the chamber, detect the electric charges.

1.4.2 PHOton Spectrometer

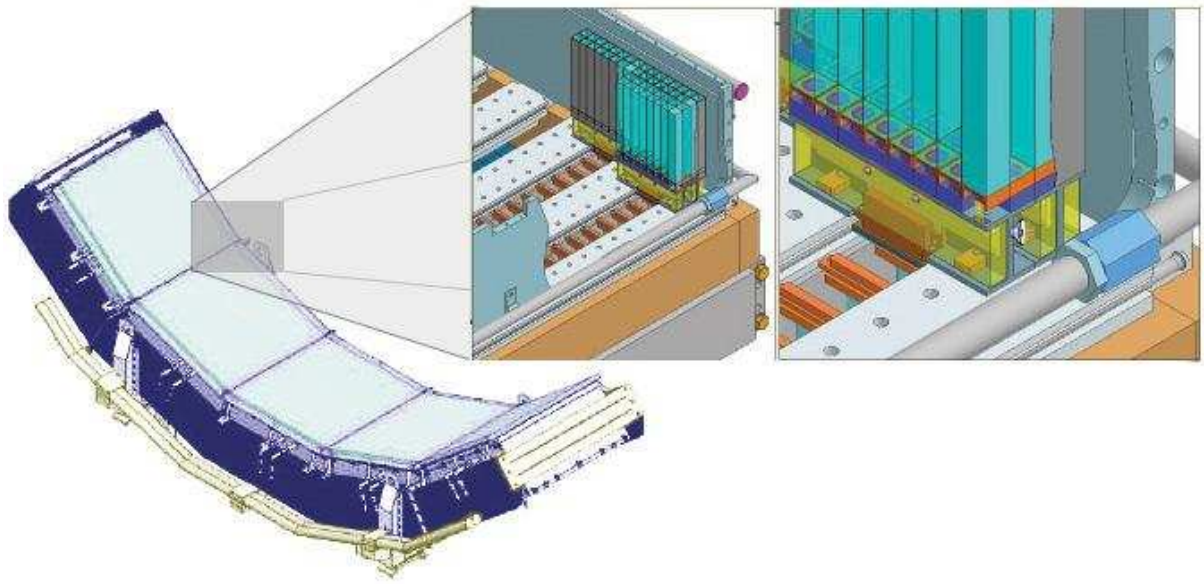


Figure 1-5: The PHOS detector; to the left is the lead tungstate crystals. From [4].

PHOS [4] is a high resolution electromagnetic calorimeter consisting of 17 920 detection channels based on lead tungstate crystals (PWO). PHOS detect collision temperatures when high energy photons hit the crystal and make them glow. This light can be detected by photodiodes.

1.4.3 The Forward Multiplicity Detector

FMD [5] will study total particle production, elliptic flow and multiplicity fractions at forward angles relative to the LHC beam line. It is a silicone strip detector with 51 200 strips arranged in 5 rings, and consist of 3 sub-detectors. The sensors are arranged in one or two rings around the beam-line. In general this detector utilizes the small band-gap in semiconductors and with an applied external electrical field, excited electrons due to charged particles traversing the crystal, will drift towards the anode. Holes drift towards the cathode and current flows.

1.4.4 ElectroMagnetic Calorimeter

EMCal [6] has much of the same functions as PHOS, but it will study the properties of matter at high densities and temperatures over an wider area. The study involves measuring photons and particles called Jets.

1.5 ALICE Online System

Timing, Trigger and Control (TTC), Detector Control System (DCS), Data Acquisition (DAQ) and High Level Trigger (HLT) controls the operation of the detectors and are named “Online Systems”. The Experiment Control System (ECS) is the top level control of the ALICE experiment. The online systems can operate independently of the ECS and have little communication between them.

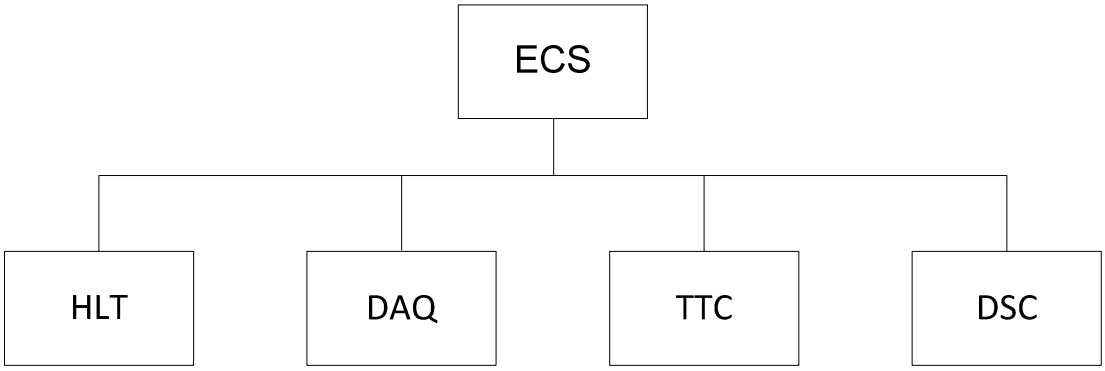


Figure 1-6: The different online systems in ALICE.

1.5.1 Timing, Trigger and Control System

All the experiments at LHC need a common timing, trigger, and control system in order to detect, synchronize, and control the data readout of events. The LHC use a precise bunch clock (40 MHz) and timing signals in order to control the detector synchronization and data readout. The ALICE TTC system [7-9] has a Central Trigger Processor (CTP) that generates different levels of hardware triggers based on input from the fast detectors. The triggers help filtering events that, for some reason, are not interesting. The CTP use Local Trigger Units (LTUs) as an interface to distribute the triggers to each sub-detector. Through optical fibers the LTUs sends trigger messages on two channels in addition to the global LHC clock to the sub-detectors Fee. The LTU receives an important feedback from the Fee, the busy signal. If the buffers on the Fee are full, the busy signal is sent to the LTU and then to the CTP, which blocks trigger from being distributed to the Fee. TPC, PHOS, FMD and EMCal utilize the Busy Box in addition to the Fee busy signal.

1.5.2 Data Acquisition System

Alice DAQ system [1]is responsible for moving data from the sub-detectors to a permanent storage facility. The DAQ system decides whether to collect or to discard the data from a collision. The Fee receives the triggers via the LTUs and the optical broadcast system. If a positive trigger is received, the data is transferred to the Local Data Concentrator/Front-End Processor (LDC/FEP). There, the data is checked, processed and assembled into sub events

before it is sent to the Global Data Collector (GDC) computers, which do the event building. Finally, the data is temporarily stored in the Global Data Storage Servers (GDS) before it is shipped and archived in the CERN's computing center where they become available for the offline analysis.

1.5.3 High Level Trigger

The ALICE DAQ has an archiving rate constraint of 1 GB/s and the detectors can produce data in the hundreds of GB/s. The purpose of the High Level trigger system (HLT) is to reduce the overall data rate from the sub-detectors by event selection (trigger on event), selection of Region of Interest (RoI) and data compression [9].

1.5.4 Detector Control System

The Detector Control System (DCS) is in charge of ALICE's experimental equipment. Everything from control of the cooling system, the ventilation system, the magnetic fields and other support system as well as the configuration and monitoring of the Fee in the ALICE's sub-detectors. The DCS tasks are distributed over many embedded computer devices and PCs in a heterogeneous system, allowing independent operations and a scalable design. The Fee DCS is detached from the data flow and only controls, configures and monitors the Fee. The configuration task includes uploading configuration data to the Field Devices, like the BusyBox.

1.6 About This Work

The Department of Physics and Technology at the University of Bergen is involved in several research projects at CERN. The microelectronic group at the university is responsible for the development of electronics needed in some of those projects. Development of the BusyBox started in 2005 with the design of the breadboard and 19" rack case by Anders Rossebø and Bjørn Pommeresche. Magne Munkejord developed the firmware for the FPGA.

My role in this project has been to document, upgrade and test the BusyBox. The work for this thesis started in early august 2008 after spending some weeks at CERN. In the first stage of the project I acquired the necessary software and project files to begin experimenting with the firmware. My focus became to learn all aspects surrounding the BusyBox. I started out studying the Virtex-4 FPGA, and refresh my knowledge about VHDL, by learning VHDL code written for the BusyBox, along with experimenting with the BusyBox. Much time was spent on learning the VHDL code written for the BusyBox, and in late November to late January, I wrote the User Guide for the BusyBox.

A fix to the Trigger Receiver module (developed by Johan Alme) was requested by Luciano Musa (leader of the ALTRO Design Team and Coordinator of the ALICE TPC Fee), based on a change in the trigger specifications from the ALICE trigger group. I had to verify that the

new change complied with the BusyBox, and studies showed that the firmware had to be modified. Firmware simulations were conducted both on the Trigger Receiver module and the BusyBox before the firmware was compiled. Along with Dominik Fehlker, we did tests in the microelectronics lab in Bergen on the hardware. This was a full scale test of a real system with Front-end electronics (Fee), CTP emulator and a Local Data Concentrator (LDC) with D-RORCs.

I also wrote a proposal for an upgrade of the BusyBox communication protocol, see Appendix B, and updated the BusyBox section on the Bergen Wiki Webpage.

This is still an ongoing project, and even though the BusyBox is commissioned, new changes will come in the future, and upgrades to the firmware are then necessary.

1.6.1 Content

This thesis describes the BusyBox and how it works. The hardware and the software used for testing and generating firmware are described. It examines the testing methods for firmware and VHDL, and with what needs to be done in the future.

Chapter 2: Gives an overview of the BusyBox system and setup. Triggers and trigger handling is discussed along with adaptation made for the BusyBox.

Chapter 3: Explains the BusyBox communication protocol. How the physical layer works, message formats and transmission types.

Chapter 4: Describes the firmware functionality, structure and modification done to the BusyBox. Information about VHDL and firmware is also given.

Chapter 5: Gives some explanation of how testing and simulations are performed, and discusses the verification plan and how testing both with firmware and hardware are carried out, before the test results are outlined.

Chapter 6: Gives a summary, outlook and conclusion of the work that has been done.

Appendix A: Abbreviations.

Appendix B: Proposal for an upgrade of the BusyBox communication protocol.

Appendix C: Test Setup

Appendix D: User Guide for the BusyBox.

Chapter 2

System Overview

Triggers are essential in ALICE to select events of interest and reduce the overall data rate to match the storage capacity of the Data Acquisition System (DAQ). This chapter describes the DAQ system, the reception and handling of triggers, and trigger information in the trigger receiver logic. How data is read and types of technology the BusyBox has adopted is discussed at the end of the chapter.

2.1 Introduction

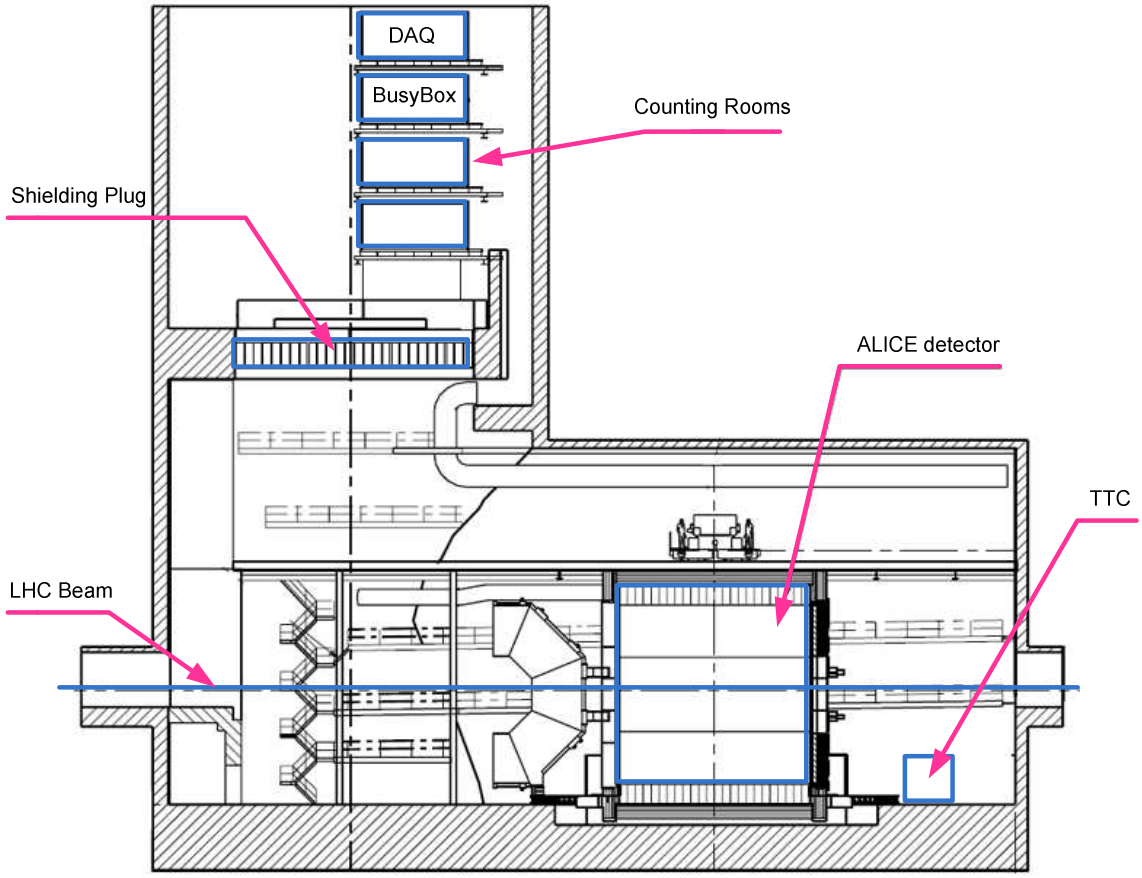


Figure 2-1: General layout of underground structure at Point 2, from [10].

The ALICE experiment needs a system to minimize the overall data rate from all its sub-detectors. The maximum permanent storage rate is 1.25 Gb/s, while one sub-detectors like the Time Projection Chamber (TPC) can produce a stunning 710 Gb/s of data from a p-p event.

To manage this task the ALICE experiment utilizes a trigger based data readout system with triggers generated from the Timing, Trigger and Control (TTC) and High Level Trigger (HLT) system discussed in chapter 1.5.

Figure 2-1 shows an overview of the underground structure at Point 2 where the ALICE experiment is located. The BusyBox together with some of the DAQ system, are located in the counting rooms which are shielded from radiation by a shielding plug. Just outside the ALICE detector is the trigger system with the Central Trigger Processor (CTP) and Local Trigger Units (LTU) where the *busy* signal is routed to from the BusyBox.

2.2 ALICE Data Flow

Figure 2-2 shows the principle of how data flows and triggers are distributed in the ALICE DAQ system. The TTC system (CTP and LTU) distributes triggers to the detectors, and the Front-end electronics (Fee) starts to buffer data if it receives a L0 trigger, or in the case of TPC an L1a trigger. If a trigger sequence ends with an L2a trigger the event data are shipped to the DAQ-ReadOut Receiver Cards (D-RORC) where the data fragments are merged by the Local Data Concentrators (LDC). A copy of the data from the D-RORCs are also sent to the HLT system and processed before it is sent back to the D-RORC and LDC. The LDC ships the data fragments to the Global Data Concentrators (GDC) for a complete merge of the data before it is sent to a permanent storage. The data then contain region of interest information, trigger information, event summary data and the compressed data. Due to dense cabling the TPC, PHOS, EMCal and FMD use a BusyBox to tell the TTC system when it can take data, while in the other sub-detectors this is done by the Fee itself.

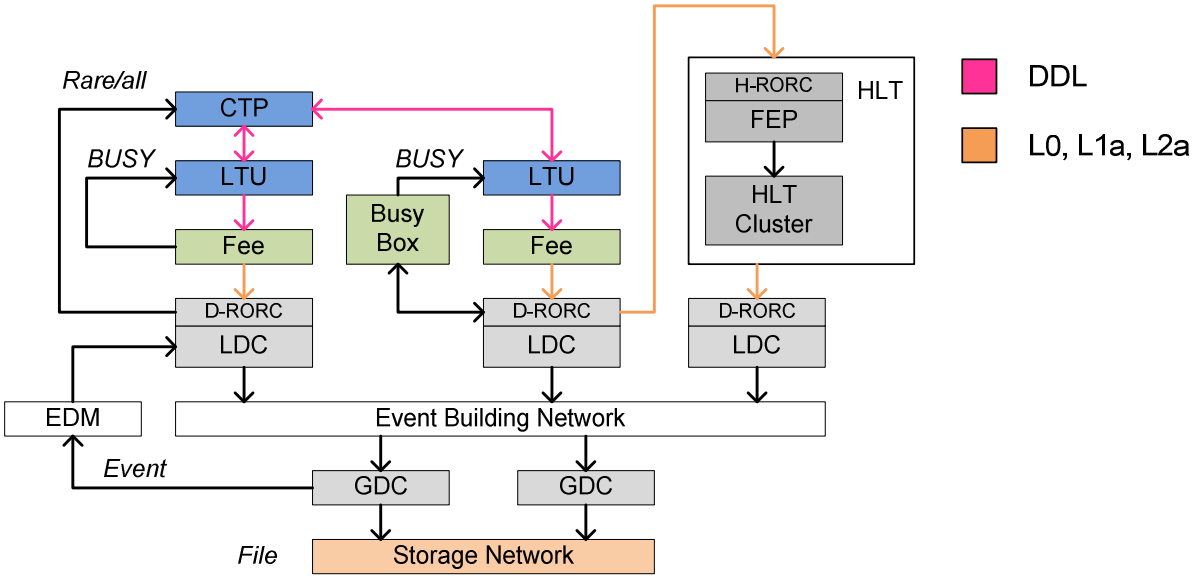


Figure 2-2: Overview of the ALICE Data Acquisition System, from [1]

The Time Projection Chamber (TPC), PHOton Spectrometer (PHOS), Forward Multiplicity Calorimeter (FMD) and ElectroMagnetic Calorimeter's (EMCal) Front-End Electronics (Fee) have a common design, and share most of the same components. Each detector has a set of Front-End Card (FEC) connected to a Readout Control unit (RCU). The RCU is linked to the D-RORC in the counting rooms. The Busy Box is also situated in the counting rooms, one for each of the four sub-detectors. The Busy Box communicates with the DRORCs and the LTU.

2.2.1 Front-end cards

The FECs [11] sample, digitize, process and buffer signals from an event. Key components are the PASA and ALTRO chips. The charge collected on the detectors pads is amplified and shaped by the PASA chips. The pulses produced are then sent to the ALTRO chip which is a mixed signal ASIC with an integrated 10 bit ADC and digital filters. The signal is sampled at a configurable rate of 2.5 to 10 MHz, filtered and buffered into memory. Depending on the sampling rate, 4 or 8, events can be buffered.

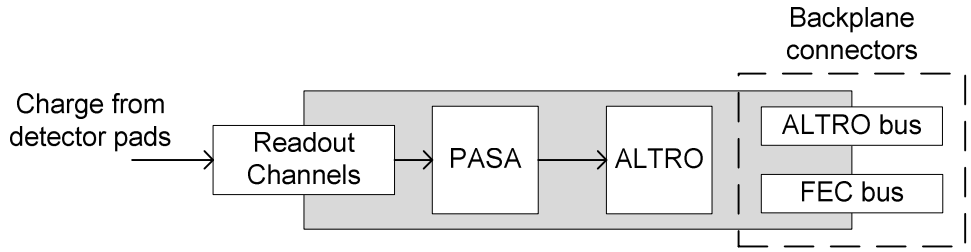


Figure 2-3: Overview of Front-end card. Up to 128 channels can be supported by one board.

Every FEC has two backplane connectors, a control node (FEC bus) and a readout node (ALTRO bus). The nodes are connected to a PCB backplane. Up to 14 FECs can be connected to each backplane and two branches of backplanes are connected to a Readout Control Unit (RCU).

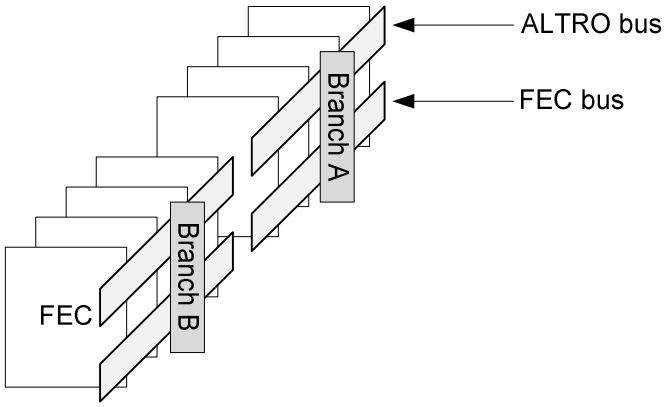


Figure 2-4: Front-end cards connected to the PCB backplanes divided into Branch A and Branch B.

2.2.2 Readout Control Unit

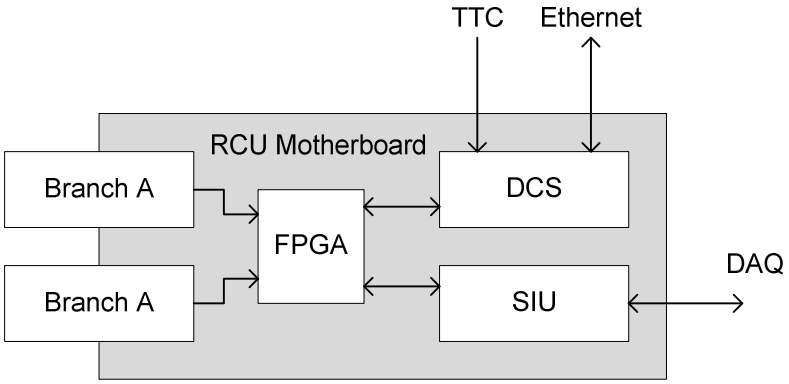


Figure 2-5: Overview of the Readout Control Unit.

The Readout Control Unit (RCU) includes a DCS board and a Source Interface Unit (SIU) on a motherboard. It has two tasks. 1) Move data from the FECs to the SIU and 2) control the FECs and sub systems on the RCU motherboard. The number of FECs connected to the RCU varies between the four detectors (TPC, PHOS, FMD and EMCal). The FECs and RCU are named Front end electronics or just Fee.

DCS board

The DCS board is an embedded computer measuring 10 cm x 15 cm and runs on a tailor made version of Linux. It has a TTCrx chip to decode TTC signals from the LTU and an Ethernet interface. This makes the DCS board very versatile and is therefore used by other systems like the BusyBox.

Detector Data Link

A Source Interface Unit (SIU) and a Destination Interface Unit (DIU) linked together with two optical fibres are the parts that make up the Detector Data Link (DDL) system. The link has a transfer rate of up to 200 Mb/s in both directions and is used for transferring event data from the FECs to the DAQ system. The SIU board is mounted on the RCU and the DIU is mounted on the Readout Receiver Cards (RORCs) in the DAQ system. Configuration data can also be sent on the DDL for configuring the Fee. This can only be done when the DDL is not used for sending event data and the Ethernet interface on the DCS board is used instead.

Figure 2-6: Overview of the Detector Data Link system used by the RCU to send event data to the RORCs.

2.2.3 Read-Out Receiver Cards and the DAQ system

Event data are shipped from the RCUs to special PCI cards named DAQ-ReadOut Receiver Cards, or D-RORCs. A D-RORC has two DDL links, one link to the RCUs and another link to the HLT. Data received on from the RCU is copied and transferred to the HLT via the SIU. The Busy IF interface is used as an LVDS link to the BusyBox and the event ID from a recent event is sent on request to the BusyBox from the D-RORC. Up to six D-RORCs can be installed in an ordinary PC called a Local Data Concentrator (LDC). The DAQ system consists of several LDCs which will merge the event fragments from one detector into a sub-event and forward it to the Global Data Concentrator (GDC). All the sub-events are then merged into a complete event and sent to CERNs storage facility for permanent storage and later analysis [1].

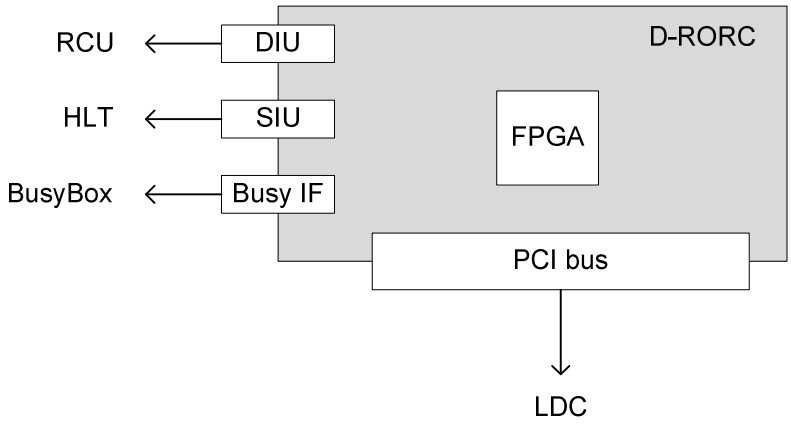


Figure 2-7: Overview of the DAQ-ReadOut Receiver Card.

2.2.4 Trigger system

The ALICE trigger system [12] is situated in the experiment hall and has a centralized layout. The heart of the trigger system is the Central Trigger Processor (CTP) and is responsible for generating three hierarchical hardware triggers – L0, L1a and L2a. This is done before an event is accepted, transmitted to the DAQ system, and copied to the HLT for further software evaluation.

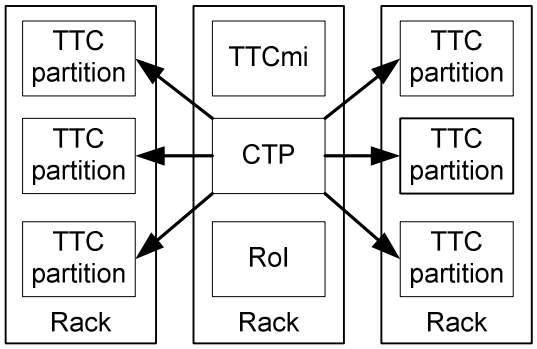


Figure 2-8: Overview of the ALICE Trigger system.

Some detectors are vulnerable some time before and after a collision due to drifting particles and for this reason the CTP will issue a L2 Reject trigger message if too much activity has been recorded before and after a collision. This is the so called Past-future Protection which is a procedure that selects events with no pile-up of long drifting ionized particles at a programmable time interval, or with a number of pile-up interactions up to a programmable limit. The Past-future Protection is preformed at all three trigger levels.

The trigger traffic is sent from the TTC system via an optical fiber network to all the sub-detectors Front-end electronic. Information sent by the TTC system is divided into three categories. 1) L1 Accept triggers, 2) Serial B messages and 3) the bunchcrossing frequency (BC0). L1 Accept contains the time critical L0 and L1 triggers, Serial B contains whole trigger messages like the L1 message, RoI (Region of Interest message), L2a and L2r messages. The event ID is a part of the L2a trigger message.

2.3 Processing of Trigger Information

Trigger information is sent to the TTCrx ASIC chip on RCU's DCS boards. The TTCrx chip decodes the optical information to electrical signal and routes it on two dedicated lines to the FPGA on the RCU's motherboard along with the BC0 clock. Two lines, named Channel A and Channel B, are used for distributing triggers and messages to the FPGA from the DCS board.

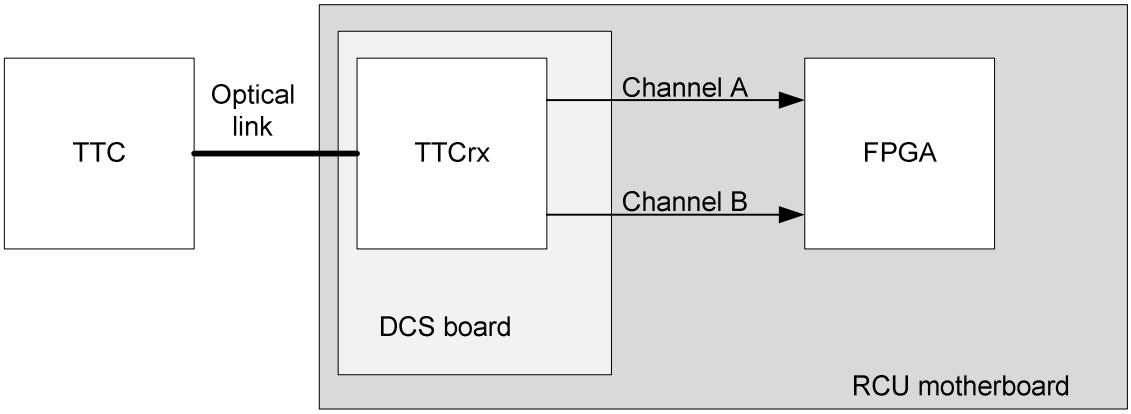


Figure 2-9: Overview of the TTC signal distribution to the RCU FPGA. The rest of the main components are hidden for illustration purposes.

2.3.1 Triggers

The RCU FPGA has a trigger receiver logic which is able to decode the information sent from the TTCrx serial B line. This decoding is done by a firmware module called the Trigger Receiver Module, and it can detect transmission errors, connection errors and generates the CDH header with the event ID among others.

Legal Trigger Sequences for TPC, PHOS, FMD* and EMCal
L0 - (L1r)
L0 – L1a – L1a message – L2a message
L0 – L1a – L1a message – L2r message
Pre-pulse*
Pre_pulse – L0*
Pre_pulse – L0 – L1a –L1a message – L2a message*
Pre_pulse – L0 – L1a –L1a message – L2r message*

Table 2-1: Legal trigger sequence for TPC, PHOS, FMD and EMCal. Sequences marked with star are only valid for FMD.

The triggers and messages from TTC system must arrive at specified latencies, and the trigger receiver logic controls that the sequence and timing is correct.

Trigger	Latency with respect of BC0
L0	1.2 µs
L1a	6.5 µs
L2a/L2r messages	88 µs

Table 2-2: Legal timing of the trigger sequences. BC0 is the bunchcrossing leading to the collision.

2.3.2 Trigger Reciver logic overview

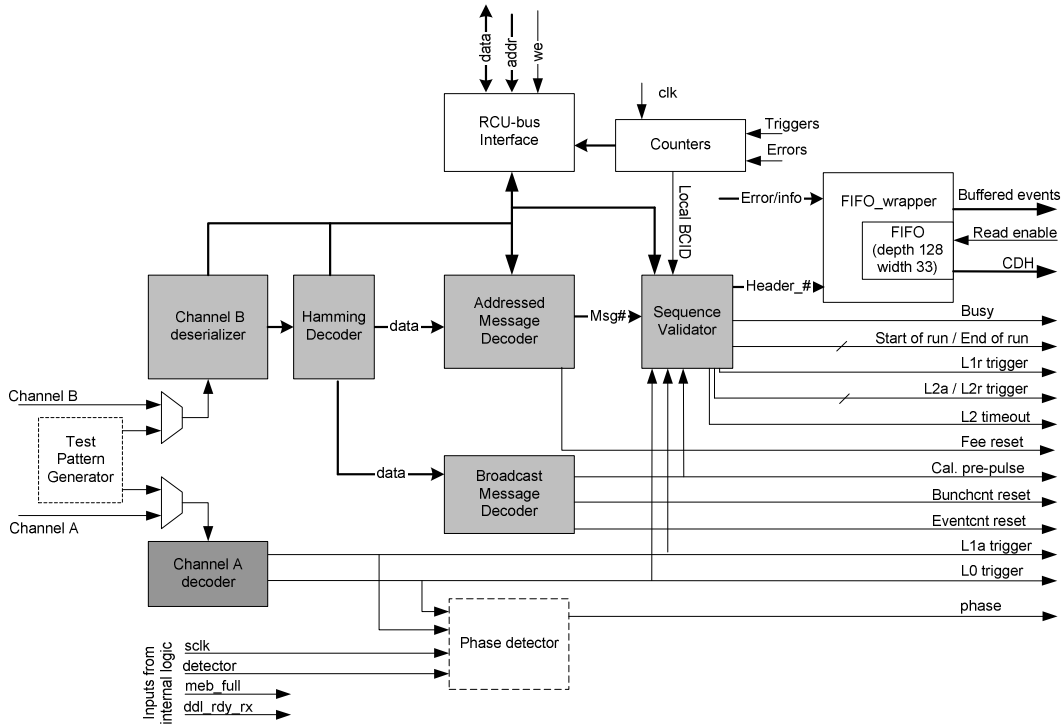


Figure 2-10: Schematic overview of the Trigger Receiver module, from [9].

The Trigger Receiver Module decodes the trigger information submitted from the TTC system. Channel A is decoded into the L0 and L1a triggers. Channel B is first deserialized and then hamming encoded before the data is split into broadcast- and address messages. The sequence validator verifies that the sequence of triggers and messages are correct and will output control signals regarding the actual event. When a sequence is received and validated, the data is formatted into the CDH version 2 format and stored in a FIFO. The CDH message format and structure of the FIFO is shown in Figure 2-11.

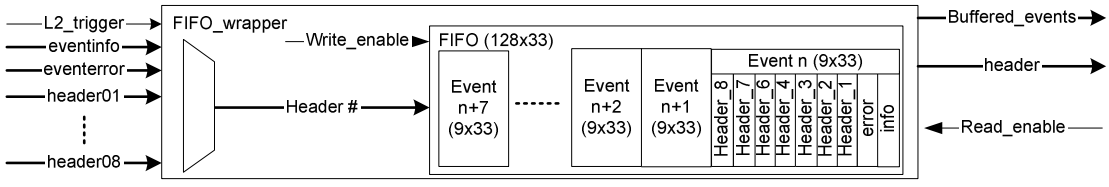


Figure 2-11: Structure of the CDH FIFO, from [9]

2.3.3 Trigger decoding

In addition to channel A and channel B, the system clock is also received by the DCS board from the TTC system. The system clock is the clock that is directly synchronous to the BC frequency in LHC (40 MHz).

Channel A

L0 and L1a triggers are received by Channel A from the TTCrx chip on the DCS board and distributed with LVDS signalling to the RCU FPGA. A L0 trigger is defined to have a length of 1 clock period and an L1a is defined to have length of 2 clock periods synchronous with the system clock (BC0).

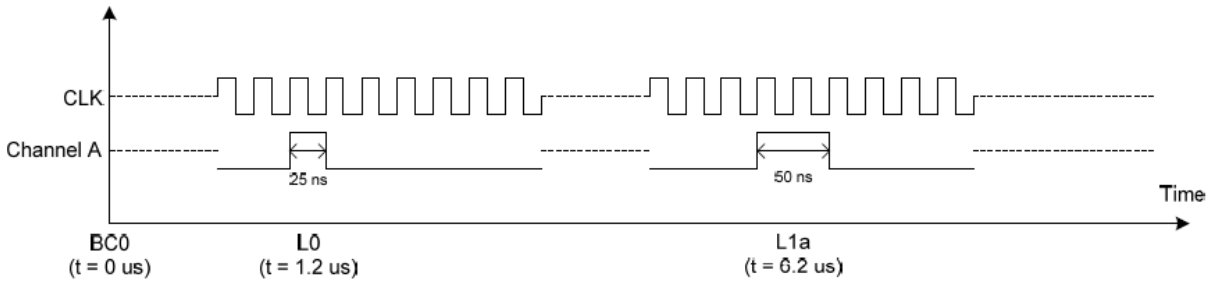


Figure 2-12: L0 and L1a trigger distribution on channel A. The clock is the system clock in LHC, from [9].

Channel B

Trigger messages are received by channel B and can be divided into individually addressed messages and broadcast messages. An important feature of the broadcast message is the bunch count reset, which resets the local bunch counter that is used for event identification. The address messages include information on the bunch crossing ID and the orbit ID that gives the unique event ID important for the BusyBox.

2.4 Data Readout

After a trigger sequence has been validated, the CDH is generated and stored in the FIFO, see section 2.3.2. The FIFO will set a flag to inform the data readout logic that data is available and ready to be read out.

Event Info	Parity | X"A956" | "0000" | Event Information(11:0)
Event Error	Parity | "0000" | Event Errors(27:0)
Header 01	Parity | "0000" | version(3:0) | "000" | CIT | RoC(3:0) | ESR | L1SwC | "00" | BCID(11:0)
Header 02	Parity | X"00" | OrbitID(23:0)
Header 03	Parity | X"00" | L2Class[47:24]
Header 04	Parity | X"0" | DAQ_error/status(15:0) | Local_BCID(11:0)

Header 05	Parity | L2Class[31:0]
Header 06	Parity | RoIdata[3:0] | X"00" | "00" | L2Class[49:32]
Header 07	Parity | RoIdata[35:4]

Table 2-3: Data stored in the CDH FIFO, from [9].

An include payload flag tells the readout logic whether data should be sent together with the header to the DAQ system or not. If this flag is zero the CDH head is transmitted without any payload, and the Fee buffer is dumped if it contained any data. If this flag is 1 then a full readout process is started. The payload flag is set when an L2a, L2r or a L2 timeout has been issued by the Sequence Validator module. If orphan messages arrives, i.e. no triggers are detected only messages, the include payload flag will not be, but the CDH header is sent.

2.5 Adaptions for the BusyBox

The BusyBox takes advantage of technology already in use in the ALICE project. The DCS board and the Trigger Receiver firmware module from the RCU are adopted by the BusyBox.

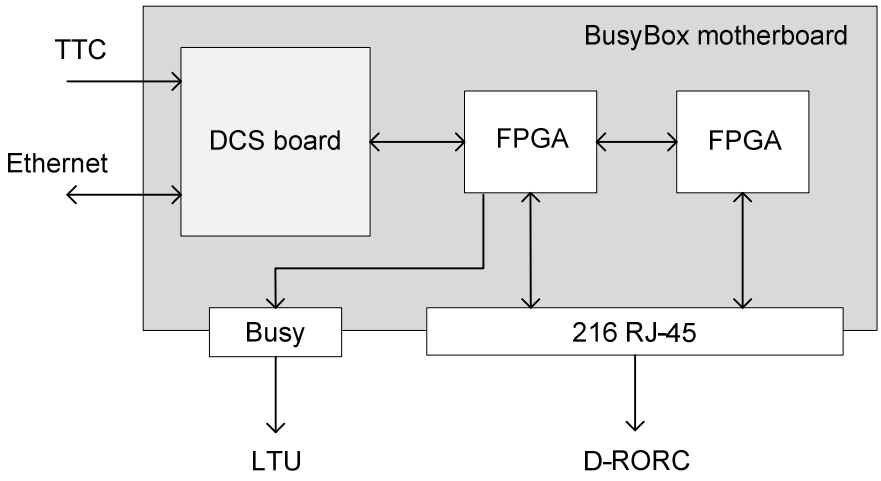


Figure 2-13: Overview of the BusyBox main components.

The Trigger Receiver module from the RCU needs to be fitted with a wrapper to communicate with the rest of the BusyBox firmware. The bus wrapper translates the definition of the bus from internal RCU type to Trigger Busy Logic type, see Figure 2-14. The main difference is that the width of the data bus in the Trigger Busy Logic is 16 bits, while it is 32 bits for the RCU.

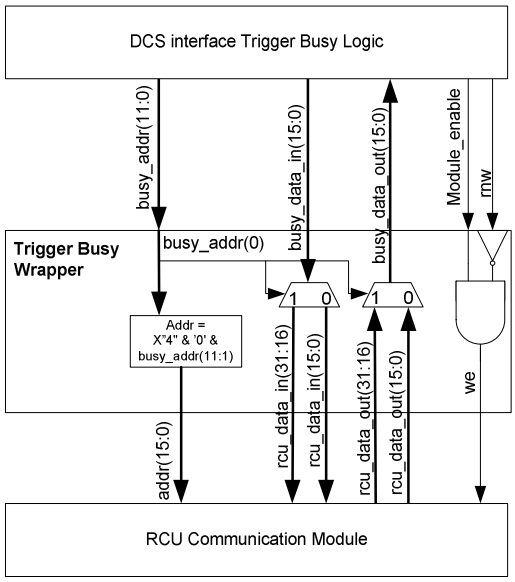


Figure 2-14: Wrapper that translates the BusyBox bus protocol to the RCU bus protocol, from [9].

Chapter 3

The BusyBox Communication Protocol

This Chapter describes the communication protocol which is used by the BusyBox and the D-RORCs to send commands and receive event IDs.

3.1 Introduction

Besides decoding trigger information the BusyBox must also be able to communicate with the D-RORCs. The communication is necessary in the sense that the BusyBox needs to know when to set the *busy* signal to the TTC system, and this will be discussed in more detail in chapter 4.

The BusyBox communication protocol was developed by Magne Munkejord as part of his master thesis. His work included investigation of serial communication protocols, implementation and testing. A robust serial communication protocol with the D-RORCs was then achieved.

The protocol defines the mechanical, electrical and functional characteristics of a serial data bus. It feature an LVDS coupled network interface, NRZ encoding, RS-232 like message format and full duplex command/response protocol. The communication link has a data rate of 40 Mbps.

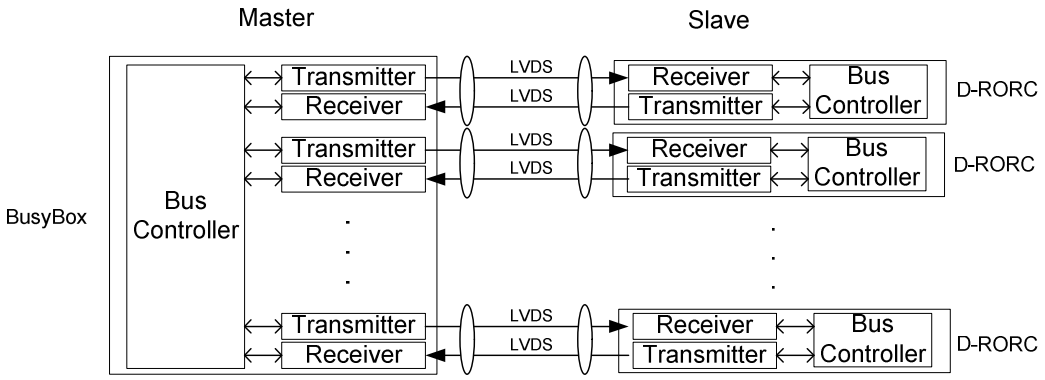


Figure 3-1: BusyBox - D-RORC bus structure.

3.2 Physical Layer

The communication between BusyBox and the D-RORCs are done with LVDS and the transmission lines are twisted pair cables with RJ-45 connectors. The TP cables are not longer than 15 m, thus providing good signal integrity.

3.2.1 LVDS

LVDS (Low-Voltage Differential Signalling) is an electrical signalling system that can run at very high speed over inexpensive twisted pair copper cables. LVDS is a differential signalling system, and transmits two differential voltages which are compared by the receiver. LVDS uses this difference in voltage between the two wires to encode the information.

The Virtex-4 FPGA is configured with the LVDS I/O standard specified as LVDS_25[2] for the output and the input I/O block.

3.2.2 Twisted Pair and RJ-45

Twisted pair cabling is a form of wiring in which two conductors, the forward and return conductor of a single circuit, are twisted together for the purpose of cancelling out electromagnetic interference from external sources. The RJ-45 is a standard eight wire connector.

Standard straight Cat-5 twisted pair cables with RJ-45 connectors are used in the BusyBox – D-RORC communication lines and the connection scheme is shown in Figure 3-2. The wiring scheme is the same as used for 10/100 BASE-T Ethernet.

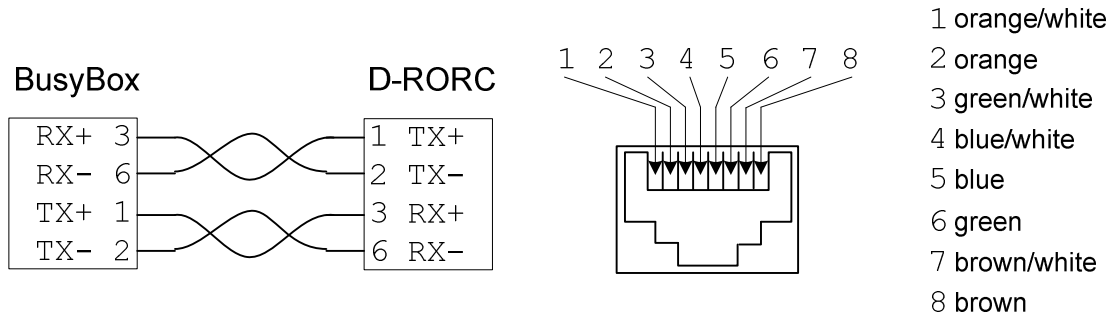


Figure 3-2: RJ-45 pin connection for BusyBox and D-RORC. Connector is shown to the right. Straight through cables are used.

[2] DIFF_TERM is enabled to set the internal differential resistor.

3.3 Message Formats

The communication on the bus consists of two types of messages: 1) Message sent from the BusyBox, and 2) Message sent from the D-RORCs. All words sent are 16 bit long with an RS232-like message format: 2 start bits, 16 data bits, 1 parity bit and 1 stop bit. The BusyBox message is also Hamming encoded.

BusyBox message

The BusyBox message has two 4 bit words: The *Command type* word and the *Request ID* word. The remaining 8 LSB bit of the message are unused, see Table 3-1.

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

Table 3-1: Bit map for BusyBox message.

Command type The command type word is used to command the D-RORCs to transmit event ID or to do error handling in relation to debugging, see Table 3-2.

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.

Table 3-2: Command types.

Request ID The request ID word is generated by the BusyBox to control the event ID queue in the D-RORCs.

D-RORC message

The D-RORC message is 48 bit long with 4 words: Request ID, Bunchcount ID, Orbit ID and D-RORC ID, see Table 3-3. The message is divided into three 16 data bits before it is sent.

47 – 44	43 – 32	31 – 8	7 – 0
Request ID	Bunchcount ID	Orbit ID	D-RORC ID

Table 3-3: Bit map for D-RORC message.

Request ID See *BusyBox message*.

Bunchcount ID The bunchcount ID is the number of the bunch that is involved in the collision.

Orbit ID The orbit ID is the number of times all bunches has circulated since the start of the run.

D-RORC ID The D-RORC ID is the unique ID given to each D-RORC.

3.4 Transmission

Both the D-RORC and BusyBox run on the same nominal bunch cross frequency, but do not share the same clock source. This is defined as a plesiochronous system and refers to the fact that this system runs in a state where different parts of the system are almost, but not quite perfectly synchronized. The sender and receiver operate at the same nominal frequency, but might have a slight frequency mismatch, which leads to a drifting phase.

The communication between BusyBox and D-RORC use NRZ line coding. A NRZ (non-return-to-zero) code is a binary code in which 1's are represented by one significant condition and 0's are represented by some other significant condition, with no other neutral or rest condition. NRZ is not inherently a self-synchronous code, and needs some kind of synchronisation technique to avoid bit slip.

The BusyBox has two clock domains, clock A and clock B. Clock A is 200 MHz and is derived from clock B, 40 MHz, which is the nominal BC frequency in LHC. Clock A is used for serial communication with the D-RORCs.

Messages sent from the D-RORCs are 48 bit long, and commands sent from the BusyBox are 16 bit long. To avoid that the two communication devices get out of synch due to the system being plesiochronous, long bit streams are avoided by dividing the D-RORC messages into three 16 bit messages before they are sent to the BusyBox. In addition to this, each bit is cycled 5 times with respect to clock A, giving a 40 Mbps rate. At the receiving end, the bit stream is sampled into a shift register long enough to hold a complete message. Then, the message is run through majority gates to determine the logic values of the capture data.

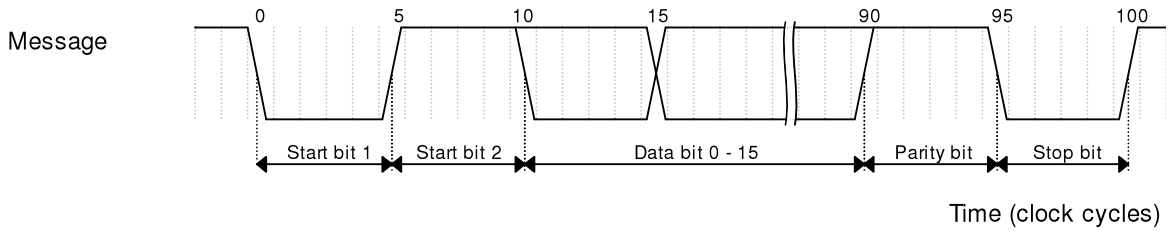


Figure 3-3: Message format.

Chapter 4

Design and Functionality of the BusyBox

The BusyBox system is FPGA based and written in VHDL. This chapter will first give a short introduction to VHDL and firmware and the remainder of the chapter will discuss the functionality of the BusyBox. A detail description of the firmware modules are given in Appendix D.

4.1 Introduction

Every detector has to tell the trigger system when it is ready to take a new event, and the BusyBox does exactly this. To know when the detectors are ready the BusyBox[3] keeps track of free buffers in the Fee of the TPC, PHOS, FMD and EMCal. It will inform the TTC system if buffers are full and no more data can be accepted, i.e. the detector are not ready to take a new event, by issuing a *busy* signal to the LTU. The LTU forwards the *busy* to the CTP that mask any incoming triggers from being distributed to the Fee. For some sub-detectors the *busy* are set by the Fee itself, but due to dense cabling the TPC, PHOS, FMD and EMCal need to use the BusyBox in order to keep track of free buffers.

The BusyBox can keep track of free buffers in the Fee by comparing the current event ID in the D-RORCs with the event ID it recently received from the LTU. If these two event IDs match, it implies that this particular event has been read out from the Fee to the D-RORCs. Hence, the occupied buffer is now freed.

Every time an L0 trigger (L1a trigger for TPC) is issued, the BusyBox increments a *buffer count* register and decrement the same register if the event IDs from the D-RORCs compared with the one in the BusyBox match. Depending on how big the buffer size is in the Fee, see section 2.2.1, the *busy* is asserted to the LTU when the *buffer count* register is full.

In addition to set the *busy* when buffers are full, there are three other *busy* conditions: 1) During the reception of triggers, 2) Problem with the connection to the LTU or issuing of a global reset from CTP, and 3) After an L1 trigger the *busy* is set for a given time interval. The last condition is a past future protection for the TPC detector, and the fact that electrons starting from the central plane in the TPC, take about 88 μs to reach the end plates where they are detected by the Fee. The purpose of this time interval is to make sure that pile-ups corrupting the data are avoided.

[3] There is one BusyBox for each of the four sub-detectors.

4.2 Design

The main components of the BusyBox are the FPGA(s), the DCS board, *busy* outputs and RJ-45 connectors. With some minor adaptation the BusyBox takes advantage of reuse of existing technology in ALICE.

The DCS board is an embedded computer running Linux, and was originally designed for use in the TRD detector. Due to its versatility, it is now adopted by the BusyBox. For communication between the DCS board and the FPGA(s) a modified version of the RCU-bus is used. To decode trigger information from the LTU the BusyBox's FPGA(s) also use the RCU Trigger Receiver logic described in chapter 2.

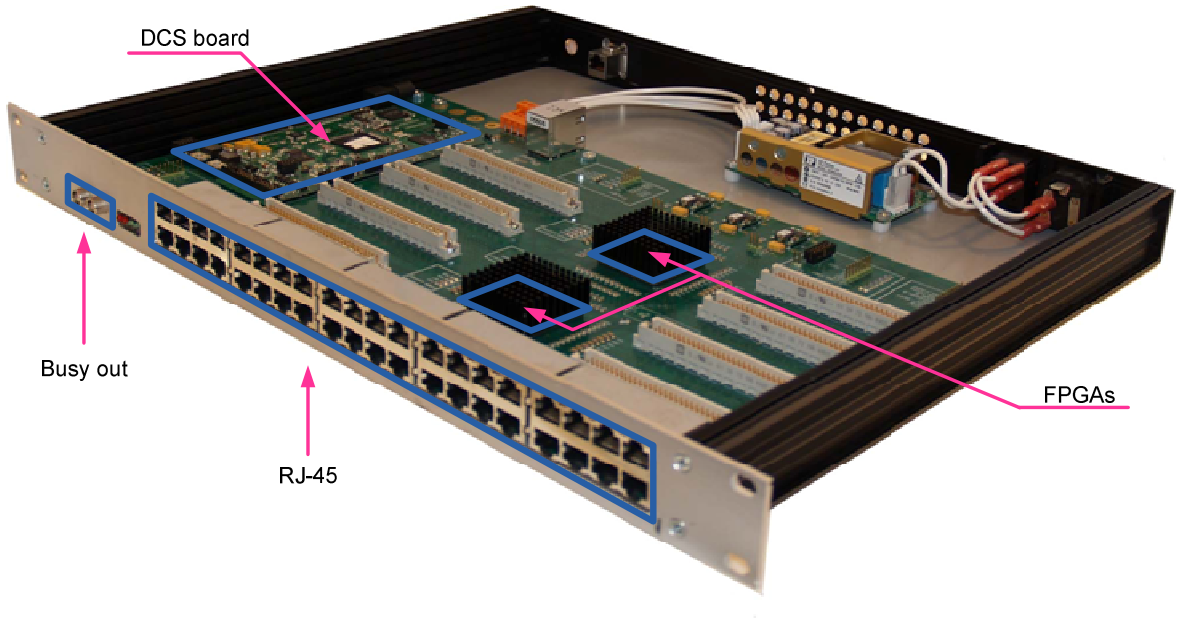


Figure 4-1: Picture of BusyBox (TPC version with two FPGAs).

The BusyBox design is implemented in the Virtex-4 (XC4VLX) FPGA from Xilinx with the FF1148 package of the type ball grid array. This FPGA has 640 general purpose I/Os which can be configured with the LVDS 2.5 standard. To communicate with one D-RORC, the BusyBox needs to configure 4 I/Os with LVDS. The TPC detector utilizes 216 D-RORCs, which means the BusyBox needs to configure 864 I/Os in total with LVDS. Thus, two FPGAs are implemented for the TPC configuration. The other detectors use less than 120 D-RORCs, and they have only one FPGA.

4.2.1 FPGA, Firmware and VHDL

FPGA

FPGA (Field-Programmable Gate Array) is a semiconductor device that can be configured after it is manufactured. A hardware description language is normally used to specify how the chip will work. FPGAs contain arrays of programmable logic blocks and reconfigurable interconnects that allow the blocks to be wired together and build a digital system.

It has become increasingly more popular than their fixed ASIC counterparts. Advantages include shorter time to marked, ability to re-program in the field to fix bugs, and lower non-recurring engineering costs.

VHDL

VHDL (Very-High-Speed-Integrated-Circuit Hardware Description Language) is a HDL language originally developed for the US Department of Defence and is now an IEEE standard. VHDL is a data flow language, and can describe concurrent systems. It is used as a design-entry language for FPGAs in electronic design automation of digital circuits. The key advantages of VHDL when used for system design, is that it allows the behaviour of the system to be described and verified before it is synthesized into real hardware.

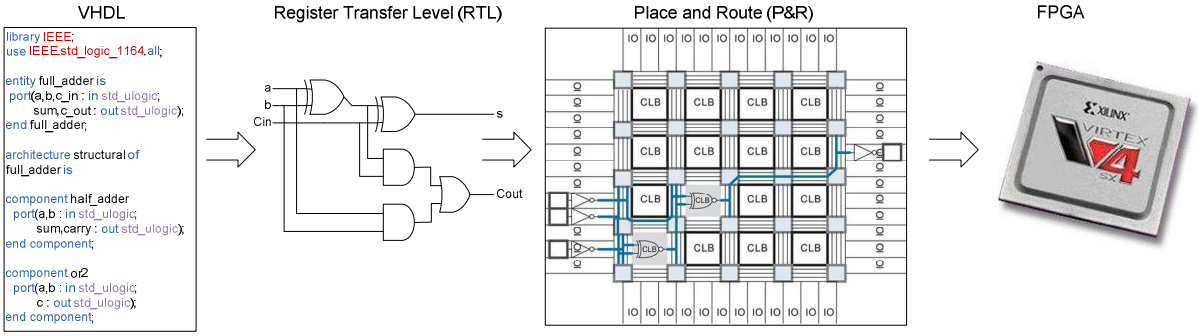


Figure 4-2: Illustration shows the basic operations from VHDL code to firmware on an FPGA.

The VHDL code is synthesized to a netlist before the code is implemented as a digital system on an FPGA. A Xilinx FPGA has an array of CLBs (Complex Logic Blocks) and each block has an LUT (LookUp Table), a D-flipflop and a 2-to-1 multiplexer. The LUT is like a small RAM with typically 4 inputs, and can implement any logical gate with up to 4 inputs. The gates and registers in the netlist are translated to the LUTs. The location of the instantiated CLBs are mapped before they are placed and connected together. Finally, a bit-file is generated and the design can be programmed into the FPGA. Figure 4-2 illustrates the basic operations discussed in this section.

4.2.2 Development and Code Organization

The VHDL code for the BusyBox is written in Xilinx's ISE and tested in Mentor Graphic's QuestaSim.

The code is organized with a top module connecting eight sub-modules together. All the source files are listed and explained in detail in Appendix D. TCL scripts sets up the project for both ISE and QuestaSim.

A SVN repository has been used as a version control system for the BusyBox project to track changes during firmware development. If errors were done, it was possible to revert to previous working versions.

4.3 BusyBox Firmware Functionality

The BusyBox firmware can be divided into three functionality categories: 1) Trigger interpretation, 2) Event verification, and 3) Busy control. The Trigger Receiver module is responsible for trigger interpretation and L2a, L2r or L2 timeout triggers are recognized from valid trigger sequences. The data from the recently finished trigger sequence is stored in the CDH FIFO, see Figure 2-11.

The FIFO has a data available signal to the Event ID Verification module that is flagged when the Trigger Receiver has generated the CDH header. The Event ID Verification module, Transmitter module and Receiver module, are included in the event verification process. The Event ID Verification module reads out the CDH header and extracts the orbit ID and bunchcount ID, named event ID. The Controller, a sub-module in the Event ID Verification module, will command the Transmitter module to send a request to the D-RORCs for their event ID.

A D-RORC will reply to the request with the latest event ID it has received from the RCU, and send this to the Receiver module in the BusyBox. The time it takes for the D-RORCs to read out an event from the Fee varies, and the Transmitter module will keep sending request to the D-RORCs until all have answered. If all received event IDs from the D-RORCs matches the event ID extracted by the Trigger Receiver module, all event data has been read out from the Fee buffers to the D-RORCs. Hence, the occupied buffer is now freed.

The busy control is done by the Busy Controller module, and this is just an OR function of four processes. The busy parameters are: TTCrx ready from the TTC system, the *busy* from the Trigger Receiver module, calculation of free buffers in the Fee and the timeout counter for post future protection. The calculation of free buffers in the Fee is based on housekeeping triggers and valid events. The counter increments 1 for an L0 trigger (L1a trigger for TPC), decrements 1 for L2a/L2r triggers and verified events. So if the counter is incremented to 4 or 8 (depending on how many multi event buffers are configured in the ALTRO chip) the Busy Controller will assert the *busy* signal to the TTC system.

The DCS Bus Arbiter and Address Decoder module handles communication between the BusyBox and DCS card, the RX Memory module stores D-RORC messages and the Control and Status module has information about the registers and control signals available in the BusyBox.

4.4 Firmware Structure

The overall structure of the firmware is best described with help of a structure chart. It shows the relationship between the main firmware modules. The different modules are represented by rectangles, and arrow lines between them represent data flow. Thick lines are buses and thin lines are single signals. The structure chart for the BusyBox firmware is shown in Figure 4-3.

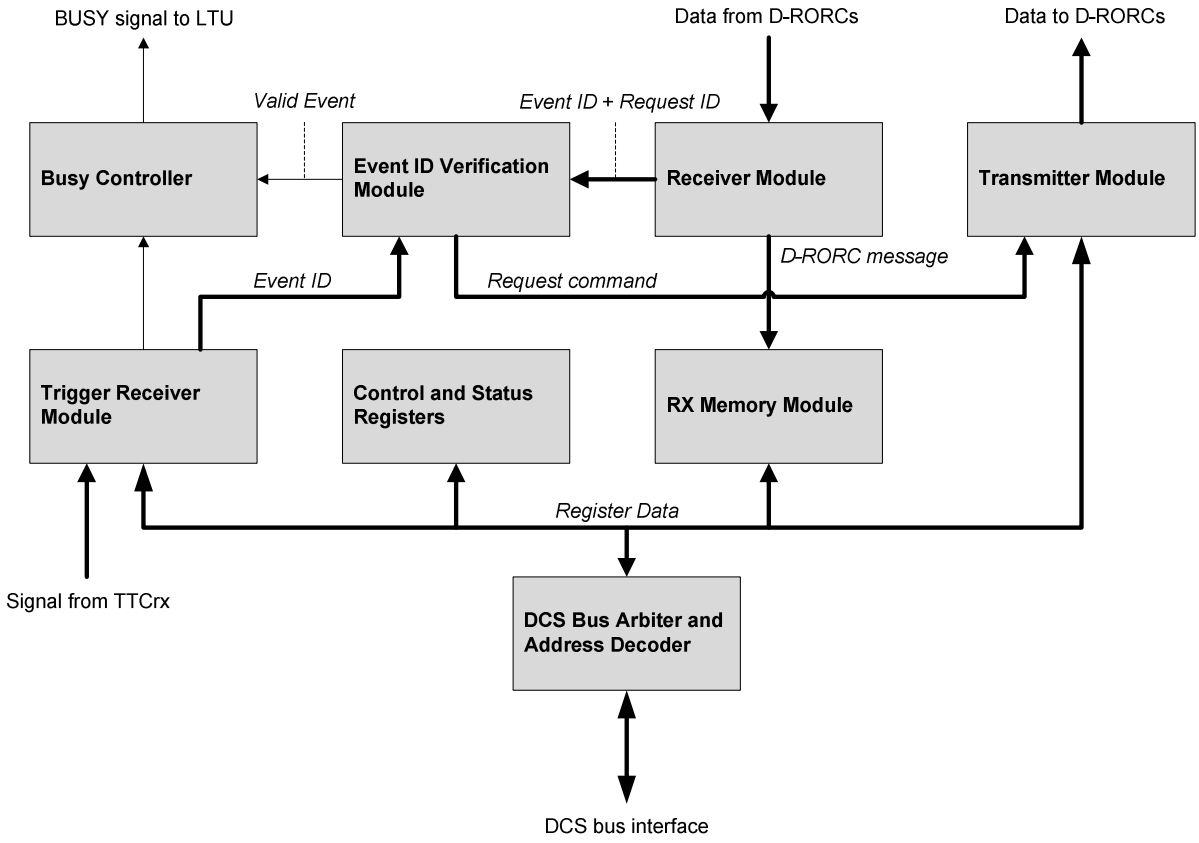


Figure 4-3: Overall structure of firmware modules in BusyBox.

4.4.1 Trigger Receiver module

As mentioned earlier, the Trigger Receiver module in the RCU decodes the trigger information. It is also used in the BusyBox for the same purpose. The Trigger Receiver module needs the adoptions discussed in section 2.5 before it can be used in the BusyBox.

The Trigger Receiver module asserts a local *busy* signal and trigger signal to the Busy Controller module during the reception of a valid trigger sequence. A valid trigger sequence ends with an L2 Accept, L2 Reject or an L2 timeout trigger and a CDH header is generated in the module accordingly.

The CDH header is stored in a FIFO in the Trigger Receiver module, and contains the event ID used by the Event ID Verification module to verify that Fee buffers have been read out to the D-RORCs. The CDH FIFO has counters linked to the Event ID module which is incremented when a CDH header enters the FIFO. The counters are used by the Event ID Verification module to extract the event ID.

4.4.2 Busy Controller module

This module decides when to assert the *busy* signal to the LTU based on triggers from the Trigger Receiver module and event information from the Event ID Verification module. A TTCrx ready signal is added to the BusyBox since each sub-detector should report busy if there is a physical problem with the connection to the LTU, or if the CTP is issuing a global reset.

By incrimination or decrimination of the *buffer count* register, based on triggers and event verification signals, the Busy Controller module keeps track of the Fee buffers without having direct contact with the Fee itself. The *buffer count* register is incremented after a L0 trigger (L1 for TPC) and decremented when an event is validated, L2 Reject trigger or L2 timeout is generated, see Figure 4-4.

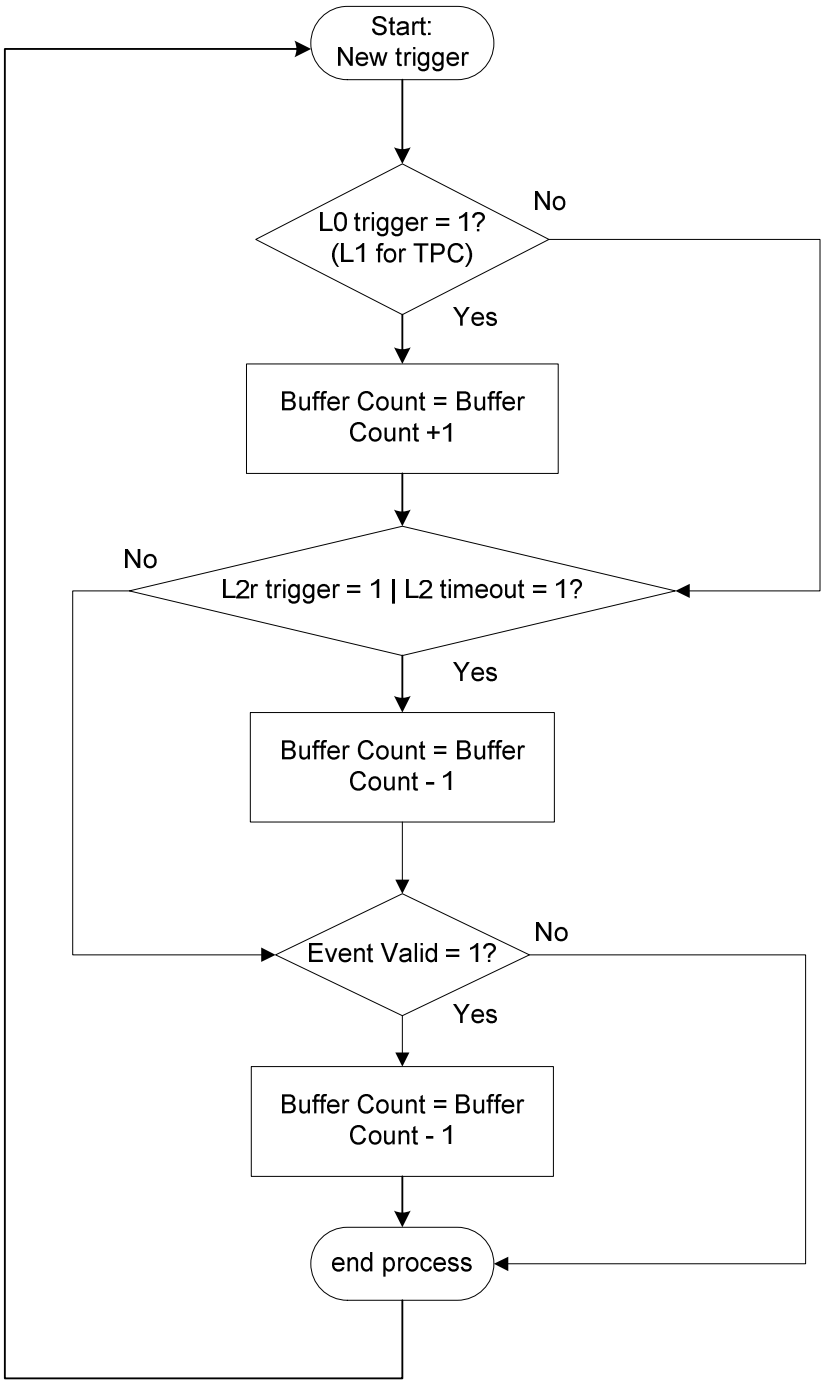


Figure 4-4: Flowchart of Fee buffer count process.

4.4.3 DCS bus Address and Arbiter module

The DCS bus arbiter and address decoder module, uses an asynchronous 16 bit data/address handshake protocol for communication between the FPGA and DCS board. This protocol is used to read and write registers in the BusyBox firmware. The MSB of the 16 bits DCS bus address selects which FPGA to communicate with. Then each firmware module can be

accessed with the next three bits and the remaining bits are used to target specific sub-module registers.

4.4.4 Control and Status module

This module has information about register and control signals in the BusyBox available for the Fee Server using the DCS bus arbiter and address decoder module as an interface. One register in particular, the CHEN register (CHannel ENable), is important and can be used to enable and disable the individual serial communication channels with the D-RORCs.

The Trigger Receiver module has its own control and status register which mostly holds information concerning trigger configurations and messages errors.

4.4.5 Event ID Verification module

The Event ID Verification module is the control centre of the BusyBox, and as the name implies, it verifies the event IDs received from the D-RORCs up against the one it receives from the LTU. Once the Trigger Receiver has received and validated a trigger sequence it generates the CDH header and places it in the CDH FIFO ready for the Event ID Verification module to read it. The FIFO in the Trigger Receiver is constantly monitored by the Event ID Verification module and the second the CDH header arrives it is read out. The event ID is extracted and put in the Event ID Queue FIFO.

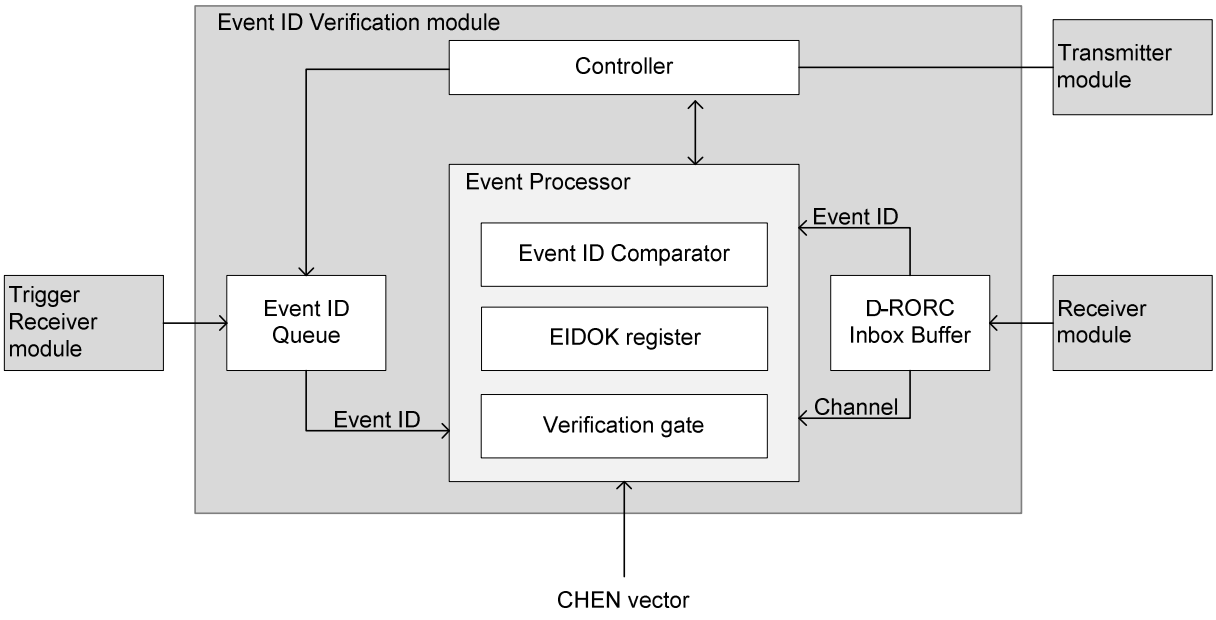


Figure 4-5: Overview of the Event ID Verification module.

The Controller module, see Figure 4-5, will send requests for the event ID it extracted to all D-RORCs enabled in the CHannel ENable (CHEN) register. The module keeps sending request until all enabled channels have replied with the correct event ID.

The Event Processor increments a Request ID register to synchronize messages in the Inbox Buffer. Each time a new event ID is requested, the request ID is sent along with the command type word to the D-RORCs, see Table 3-2. They will remember this ID, and new event IDs are not sent to the BusyBox before the Request ID has changed.

A flag is raised to the Busy Controller module when the event ID has been verified, indicating a valid event (event has been read out from Fee to all active D-RORCs).

4.4.6 Transmitter module

The Transmitter module sends commands to the D-RORCs. By default it request event IDs from the D-RORCs, but the DCS board can send other commands meant for debugging purposes as well. The Transmitter module is the arbiter between the Event ID Verification module and the DCS board to avoid communication conflicts. The CHEN vector is used to select which serial channels to transmit the request on.

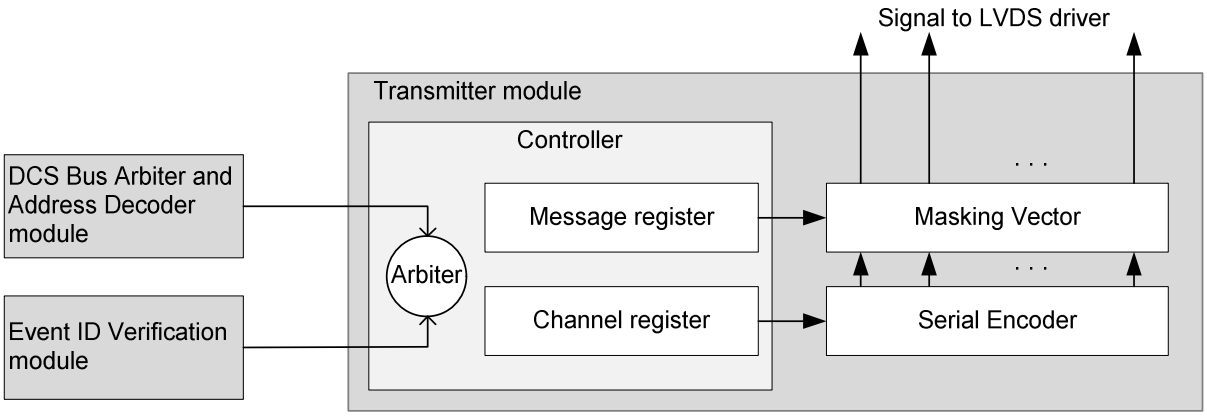


Figure 4-6: Overview of the Transmitter module.

4.4.7 Receiver module

Serial data from the D-RORCs are handled by the Receiver module and up to 120 single channels can be implemented in one FPGA. The serial receiver channels are arranged in a multiplexer architecture with up to 16 serial channels connected to a *branch controller,* and up to eight *branch controllers* connected to a single *backbone controller,* see Figure 4-7. The number of serial channels and branch controllers are set with VHDL generics before the firmware is compiled.

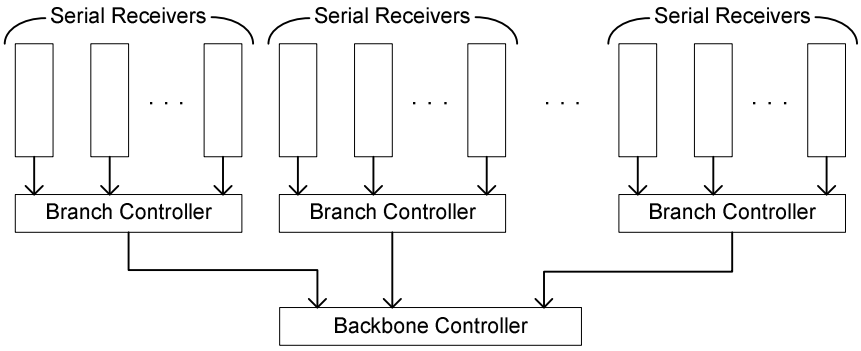


Figure 4-7: Multiplexer structure of Receiver module.

The reply from the D-RORCs cannot be predicted, and therefore the *branch controller* scans through the serial channels for messages that have been decoded, concatenated and checked for parity errors. The *backbone controller* scans through all the *branch controllers* and reads out the message buffered in each *branch controller,* and puts them in the D-RORC inbox FIFO in the Event ID Verification module ready for verification.

The complete message received from the D-RORCs is forwarded to the RX Memory module where up to 1024 messages can be stored.

4.4.8 RX Memory

The BusyBox can store up to 1024 D-RORC messages from the Receiver module in the RX Memory module. Four BRAM modules are instantiated in the FPGA, and can be accessed from both clock domains. The data words from the Receiver module are 56 bit, and are written into memory at the address given by a 10 bit counter. The DCS bus is limited to read 16 bit at a time, and needs four read operations to get the whole word from memory. The RX Memory module can also be written to by the DCS card for testing and verification purposes.

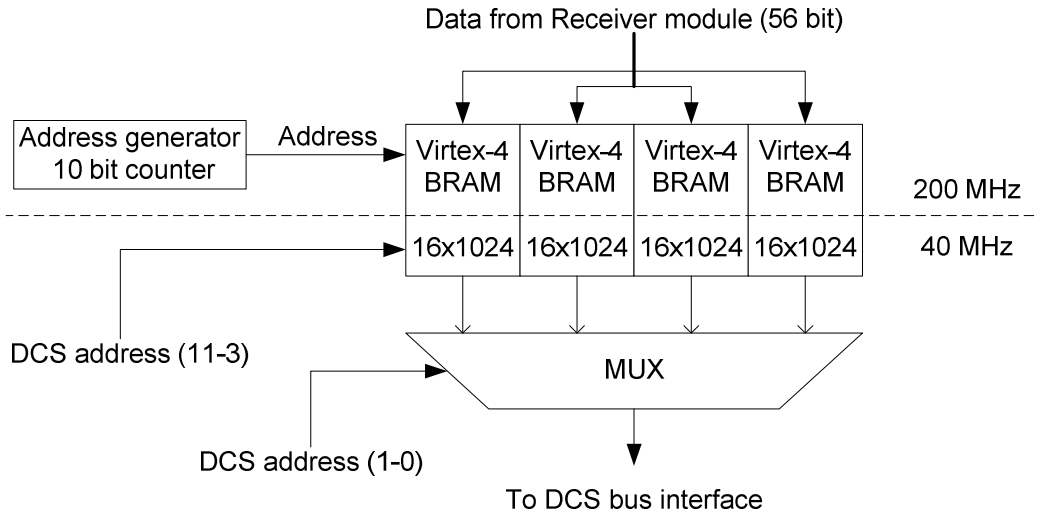


Figure 4-8: Overview of RX Memory module.

4.5 Firmware Upgrades

Johan Alme modified the Trigger Receiver module according to a change in the sequence validation specification requested by the ALICE trigger group. If only orphan messages arrive, they will kick off a new sequence starting. Earlier this was only done when a L0 or L1a trigger was detected. The CTP never issues trigger messages without L0 or L1a triggers, but errors can occur. The fix was done in the Sequence Validator module, a sub-module of the Trigger Receiver module, see Figure 2-10. Investigations of the BusyBox firmware revealed that it affected the Busy Controller logic.

Three scenarios are possible. 1) Orphan messages arrives the RCU, 2) Orphan messages arrives the BusyBox, or 3) They both receive orphan messages. In any of the three cases, the RCU ships the CDH header without event data to the D-RORCs. The BusyBox verification process is not affected by a trigger sequence with orphan messages. The event valid flag is set to the Busy Controller after the validation process described in section 4.4.5 has finished. The Busy Controller logic on the other hand, which is in charge of counting free buffers, relies on a correct trigger sequence to determine the right numbers of free buffers. So, if no L0/L1a triggers are sent from the Trigger Receiver module to the Busy Controller module, it will get the buffer count wrong. It will not increment the buffer count register when orphan messages arrive, instead the buffer count is decremented when the event valid flag is set, giving an erroneous value in the register. This is only the case with scenario two and three.

The Sequence Validator flags an *include payload* on a register if a trigger sequence ends with an L2 Accept trigger. It is not set if a trigger sequence with orphan messages ends with an L2 Accept trigger. Thus, this payload register is used as a signal in the Busy Controller logic as part of the other measures taken to prevent erroneous values in the buffer count register.

The solution to the problem is to include a 1 x 16 FIFO in the Busy Controller logic, see Figure 4-9. The *include payload* signal is written to the FIFO when the L2 Accept trigger is set, and the payload signal is read out when the *event valid* flag is set. The *buffer count* register is only incremented if the *include payload* flag is set.

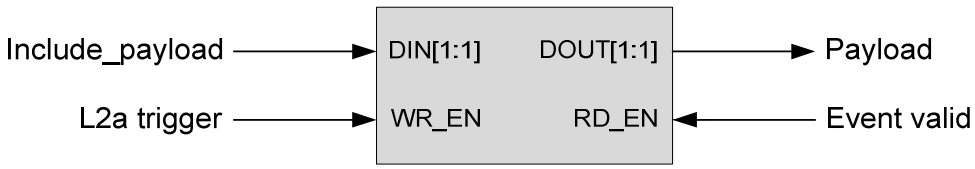


Figure 4-9: Payload FIFO.

The *payload* signal is available one clock cycle after the *event valid* was flagged. A shift register is added to shift the *event valid* signal one time before the *payload* signal is evaluated by the buffer count logic.

The original design did not implement a firmware version register which made it very difficult to distinguish between different versions of the BusyBox firmware, i.e. reverting to a working version if the new firmware had errors. So, a firmware version register is added to address 0x2015 starting at version 1.01.

Chapter 5

Verification and Testing

This chapter discusses the testing done and results achieved. First an introduction to verification and firmware simulation before the test setup and results are discussed.

5.1 The Verification Plan

5.1.1 Introduction

In a hardware design project, around 70 % of the project development cycle is devoted to design verification [13]. The consequences for a commercial company of an informal verification process can result in huge economic losses or a delayed product shipment. The development of the BusyBox is done by students with little or none experience with hardware design and verification, which has resulted in ad hoc approaches and lack of specification documents. This conduct can be justified by the fact that the BusyBox project is not a commercial product and has little non recurring engineering cost.

There exists no specification plan for the firmware design of the BusyBox that can be used in a verification plan. Thus, a template specification that describes the design as, e.g.; “The same thing we did before, but with missing triggers and with these additional features”, is used instead. It has been a part of this work to work out a BusyBox specification. This is included in Appendix D.

5.1.2 Verification strategies

The verification strategy has two approaches: 1) Firmware simulation with testbench, and 2) Hardware test with documentation of operation. The BusyBox is not flawless, and bugs reported by the users at CERN are sudden loss of firmware configuration and spurious D-RORC communication.

5.2 Firmware Simulation

5.2.1 Introduction

To simulate a design, an additional code called a testbench is required to mimic the environment in which the design will reside. The simulator lets the designer see all values of the internal signals, as well as the external signal in the design on a time scale, by supplying waveforms to the design. This is an indispensible feature in finding and backtracking bugs

and errors in the design. The restrictions to the coding style of the testbench are far fewer than for the design being manufactured.

The testbench for the BusyBox is written in VHDL, but may be written in Verilog, *e*, Open Vera or it could also include external data files or C routines. The term “testbench” usually refers to simulation code used to create a predetermined input sequence to a design, then optionally to observe the response.

The design under test (DUT) is encapsulated in the testbench like a harness with a completely closed system: no inputs or outputs going in or out, see Figure 5-1. The verification challenge is to determine what input pattern to supply the design and what to expect from the output given a properly working design.

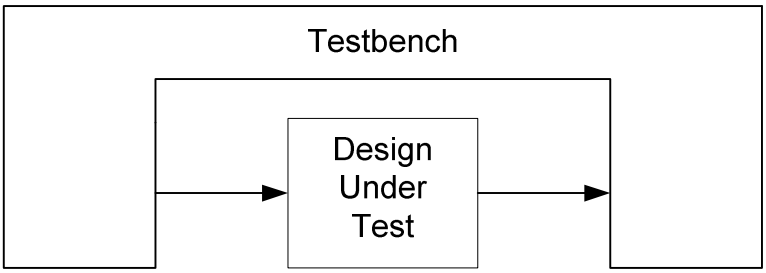


Figure 5-1: Generic structure of a testbench and design under test.

The testbench is not synthesizable and provides the DUT with input signal and clock cycle (test vectors) similar to the stimuli expected in the real system. The simulator will, when executed, keep track of all signals and their interactions in time, and the outputs can be monitored in a waveform viewer.

5.2.2 Testbench for the Trigger Receiver Module

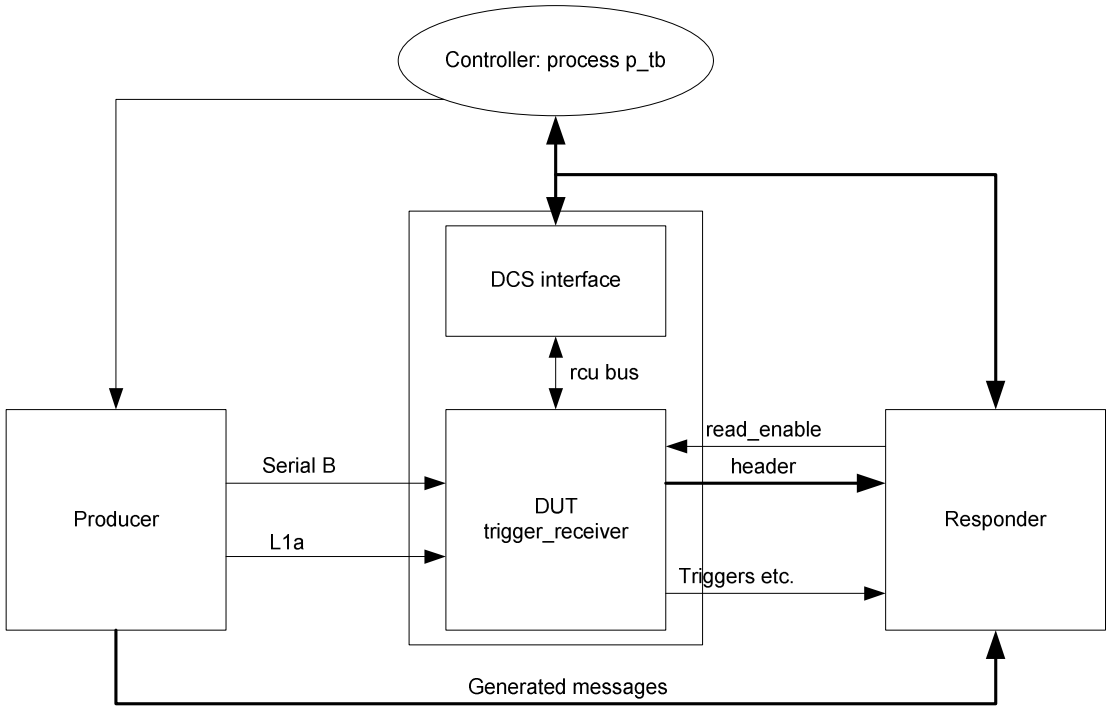


Figure 5-2: Testbench for Trigger Receiver Module, from [14]

A sketch of the testbench for the Trigger Receiver Module can be seen in Figure 5-2, where the Trigger Receiver Module is the Device Under Test (DUT). The DCS interface is added so that communication with the internal RCU bus structure of the trigger receiver can be used to forward functions and procedures to and from the DUT. Besides the DUT, there is a Producer and a Responder module. The Producer module emulates the L1 Accept and Serial B lines. The Responder module emulates the internal RCU logic interface, and receives the generated stimulus from the Producer module. The Responder module can then verify the content that is produced by the Trigger Receiver Module. The Controller (process p_tb) implements different test cases by using defined procedures in a testbench package.

5.2.3 Testbench for the BusyBox

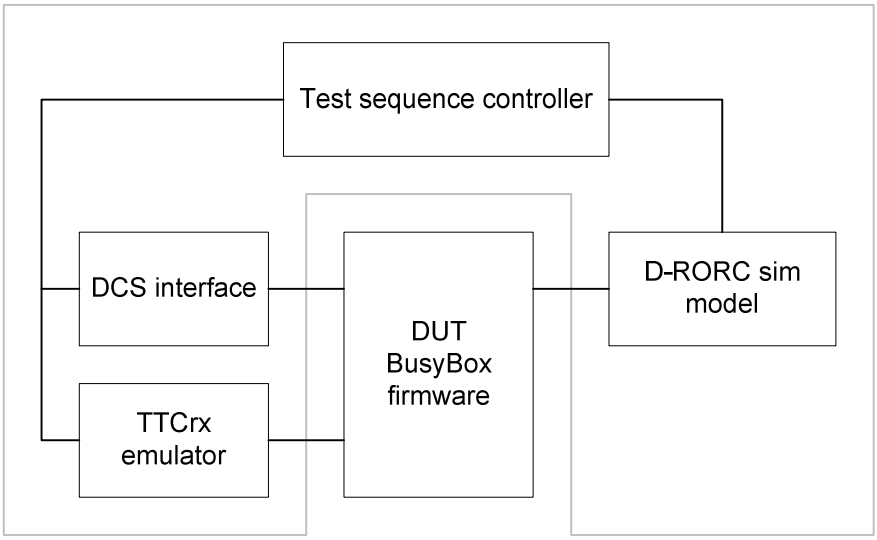


Figure 5-3: Testbench for the BusyBox.

The testbench for the BusyBox incorporates much of the same features described in the Trigger Receiver testbench. It has a controller which can initiate trigger sequences and read control and status registers through the DCS interface. The D-RORC sim model has the same serial receiver and transmitter that is implemented in the D-RORC and behaves much in the same manner as the real D-RORC does. It can handle communication with all 120 channels on the BusyBox.

5.3 Test Cases

5.3.1 Introduction

In order to verify that all the requirements of an application are met, test cases are used to test the DUT. A test case is used instead of a direct testbench approach, where individual features are verified using individual testbenches. The test case approach does a number of tests on the DUT where the tests are divided into groups or *test cases*. Figure 5-3 illustrates the idea of a test case approach. The implemented test cases are described in the following sub-sections.

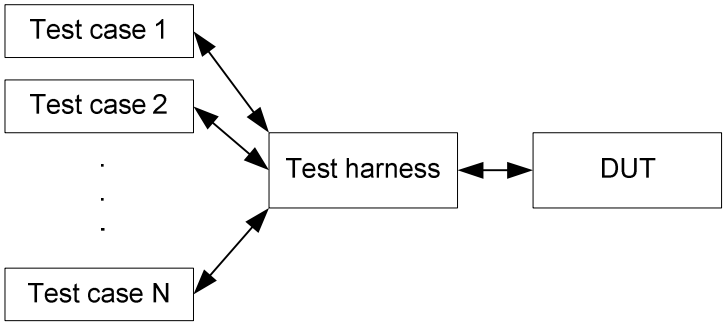


Figure 5-4: Typical verification with the separation of the test harness and the test cases, from [13].

5.3.2 Test cases for the Trigger Receiver Module

The testbench for the Trigger Receiver module has over 30 different test cases to test the firmware. These tests include running all types of trigger sequences, bit error tests, hamming error tests, just to mention some.

The Sequence Validation module of the Trigger Receiver has been updated to handle orphan messages and a new test case has been added to verify that it work. It will test how the module handles a possible error in a trigger sequence where only orphan messages arrive with no L0 and L1a triggers.

5.3.3 Test cases for the BusyBox

In addition to the test performed on the Trigger Receiver module a set of test cases are added to test the BusyBox. These tests are run on the BusyBox testbench and cover the functions of the firmware.

All legal trigger sequences, see Table 5-1, without pre-pulse, are tested along with orphan messages. It must be verified that the BusyBox can count the Fee buffers and assert the *busy* when it is supposed to. An important feature is the event validation process. It must be verified that the BusyBox can extract the event ID from the CDH FIFO in the Trigger Receiver module, request the D-RORC to reply on all types of command, see Table 3-2, and assert the event valid flag if an event has been verified.

5.4 Evaluation of Firmware Simulations

The Trigger Receiver testbench had to be modified to incorporate a trigger sequence with just orphan messages. The Producer module, see Figure 5-2, is responsible for sending triggers and messages and was modified to only send L1a and L2a messages. When the changes were done it could be verified that the Sequence Validator sub-module of the Trigger Receiver

behaved according to specifications. This means that the *include payload* flag is not set when only orphan messages are detected.

A direct testbench approach was used instead of the test cases method to verify the new changes made to the firmware in the Busy Controller module. This was done before the whole BusyBox firmware was simulated using the testbench discussed in section 5.2.3.

5.5 Hardware Test

5.5.1 Introduction

Given the fact that the BusyBox is in use and working, but with a few reported errors, a hardware test is conducted to test and document what is working. So, when changes are done and new errors are revealed under testing, the hardware test documentation can be used as a golden reference. The documentation can then tell if the hardware passed this test in a previous working version, hence, the new fix has to be reviewed and changed.

5.5.2 BusyBox interface to the DAQ System

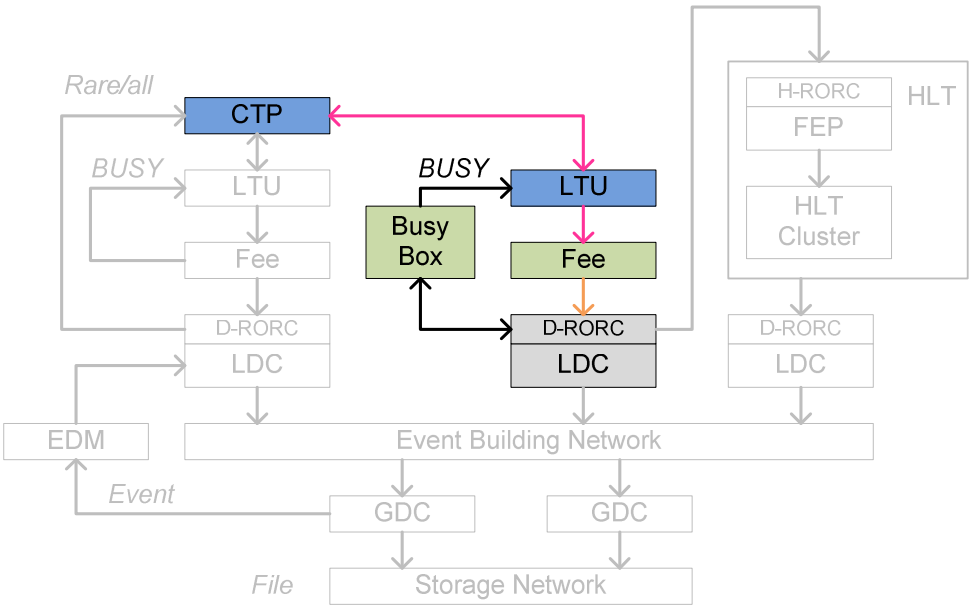


Figure 5-5: Overall architecture of the Data Acquisition System with the BusyBox interface highlighted, from [1].

Figure 5-5 shows the architecture of the DAQ system in ALICE where the BusyBox interface is highlighted. In order to test the BusyBox, a TTC system with a CTP and LTU is required, along with Front-end electronics (Fee) and a Local Data Concentrator (PC) with D-RORCs and DATE installed.

Local Trigger Crate

All the detectors in ALICE receive their trigger signals from the ALICE Trigger System through a standard interface called a TTC partition in the Local Trigger Crate. The TTC partition consists of a Local Trigger Unit (LTU), and two modules (TTCvi, TTCex), to generate TTC signals to the detectors. These modules are fitted into a standard full width 6U VME crate together with a Single Board Computer (SBC). The SBC configures the modules, oversees the readout of data and controls the detector in stand-alone mode.

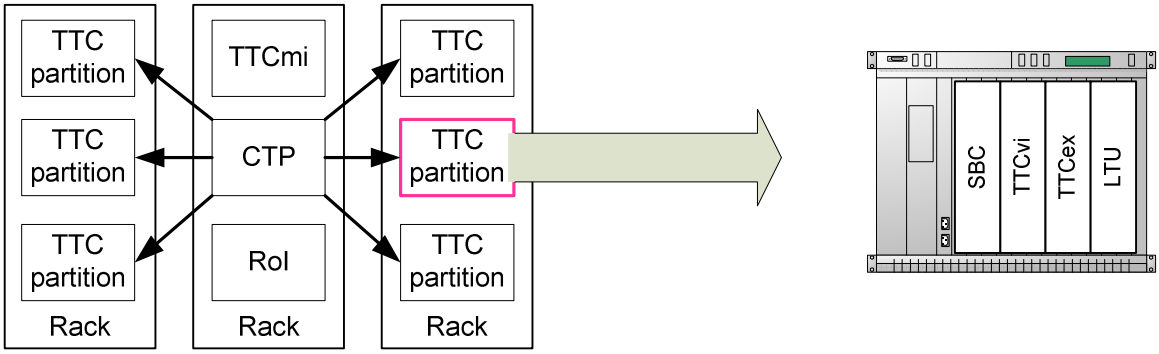


Figure 5-6: Overview of ALICE Trigger System and one LTC holding the TTC partition.

In stand-alone mode [12], the LTU can fully emulate the CTP protocol, and can be operated at a remote site without the CTP rack cluster shown in Figure 5-6. It can emulate all existing trigger sequences, see Table 5-1. This is all done with the LTU software running on the SBC.

Sequence name	Sequence structure
L0	L0
L2 accept	L0 – L1 – L2a
L2 reject	L0 – L1 – L2r

Table 5-1: List of emulation sequences.

Front- end electronic

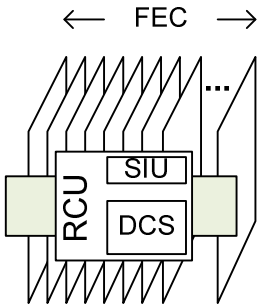


Figure 5-7: Overview of the Fee setup.

The Front-end electronics consists of four main components; Front end cards (FEC), PCB backplane, Readout Control Unit (RCU), Detector Control System (DCS) and Source Interface Unit (SIU). The FECs are connected to the PCB backplane, which is connected to the RCU motherboard that holds the DCS board and SIU.

Local Data Concentrator

The Local Data Concentrator (LDC), is a PC running Linux with D-RORCs connected to its PCI bus. DATE (ALICE Data Acquisition and Test Environment) is the software framework of the ALICE DAQ and is a distributed process-oriented system. It is design to run on UNIX/Linux platforms connected to an IP network and sharing common file systems such as NFS. The standard UNIX system tool available for process synchronisation and data transmission are also used. The DATE system performs different function [1]:

- The LDC (Local Data Concentrator) collects event fragments transferred by the DLL's into its main memory and reassembles these fragments into sub-events. It can also do local data recording in standalone mode.
- The GDC (Global Data Collector) collects and puts together all the sub-events concerning the same physics event, builds the full events and archives them to the mass storage system.

5.6 Test setup and Implementation

The hardware tests are conducted in the microelectronics lab at the University of Bergen. The lab is fitted with one TTC partition, an LDC with three D-RORCs and an RCU with Front-end cards which are used in the test setup, see Figure 5-8.

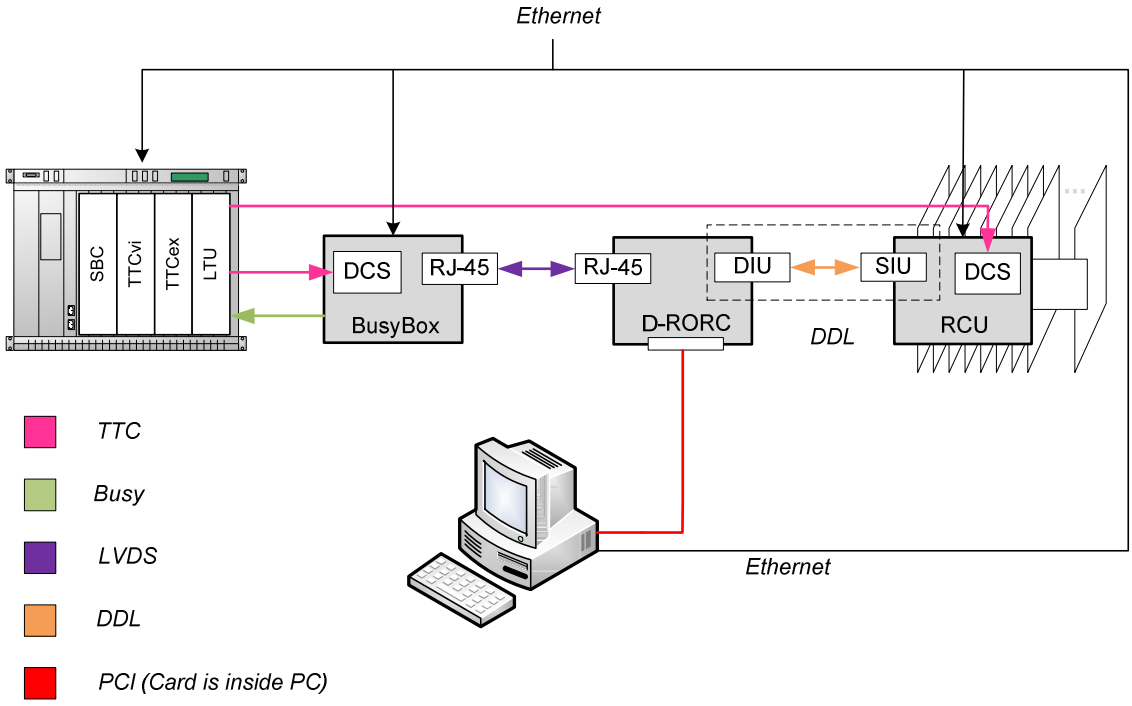


Figure 5-8: Complete test setup for BusyBox.

5.6.1 LDC

The LDC uses the Scientific Linux CERN SLC release 4.7 with Red Hat distribution and is compatible with the PCI-X bus. Three D-RORCs are installed, but only one is used (s/n 3006), see Figure 5-9. The RCU must have a branch with some Front-end cards connected to the PCB backplane in order to work in the test-setup.

DRORC s/n	3006
RORC firmware version	v2.15
RORC software version	v5.3.5
RORC driver version	v5.3.5
DLL-RORC version	v2.1.1.0 (2.125 GB/s)
PCI bus mode	PCI-X (100 MHz)
RORC minor	2
RORC channel	0
RORC revision	4
PC	drorc.ift.uib.no

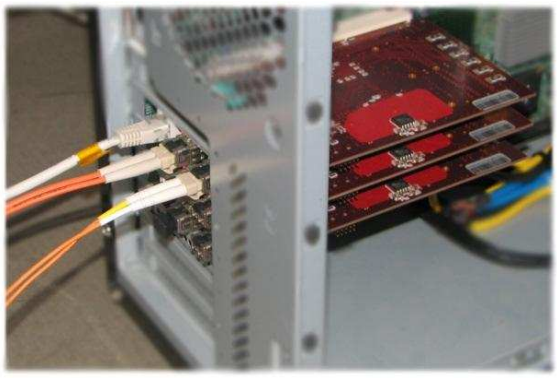


Figure 5-9: LDC with three D-RORCs.

DATE is the software framework for the ALICE DAQ and consists of a set of software packages. Every data acquisition is controlled by a *runControl* process, a component of the *runControl* system. This system consists of several other processes that must be present when

running. The *runControl* process can only take commands from one operator at a time and the **Lock**, see Figure 5-10, allows assigning the mastership of the *runControl* process to the *runControl Human Interface*. This has to be done to have full control of the data acquisition, since DATE allows concurrent, independent data acquisition controlled by multiple independent *runControl processes*.

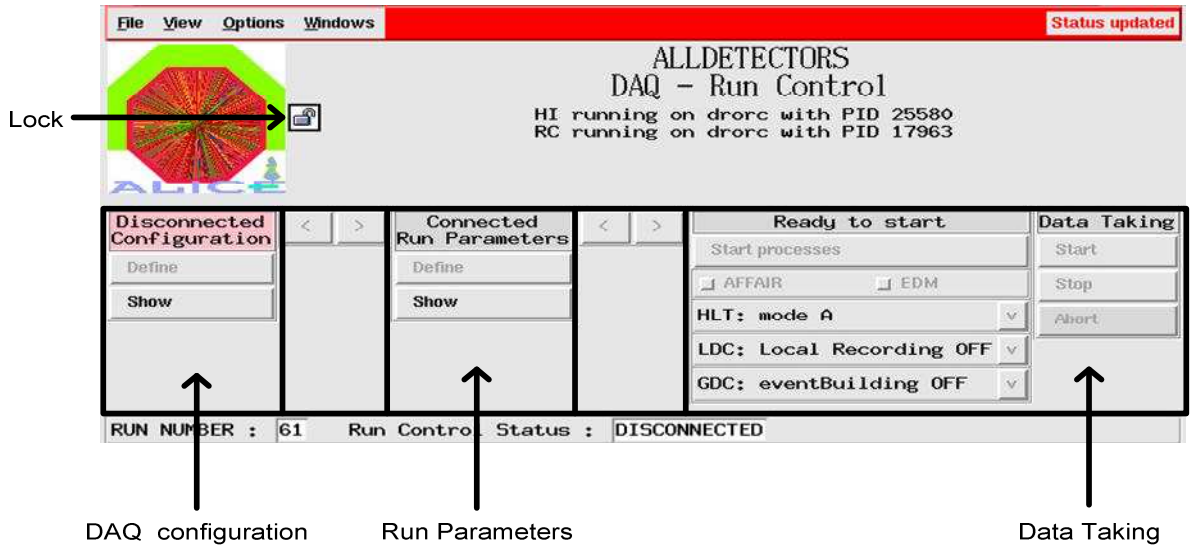


Figure 5-10: The main runControl window.

Figure 5-10 also shows the three phases; DAQ configuration, Run Parameters and Data taking to move through before the LDC is ready to take data. Put another way, this is where you tell DATE to do local data recording in standalone mode as an LDC, and set different run parameters, e.g. max event size and so on.

5.6.2 BusyBox

The BusyBox is programmed via the SelectMap interface using a programming script on the DCS board. Another script, *bbinit.sh.new2*, sets all the appropriate register values needed to use the BusyBox in the test-setup.

DCS board version	2.84UiB
DCS board number	dcs0055
BusyBox firmware version	1.0
bbinit script	bbinit.sh.new2
Channel	0
Mounted system (/mnt)	kjekspc7.ift.uib.no/ nfs_export/dcscard

Table 5-2: BusyBox test information.

5.6.3 Fee

The RCU firmware is programmed and configured by running *S30rcu-config.sh* and *S40configure.sh,* which are scripts on the DCS board. The RCU is then configured from the LDC via the DDL link using an *altro-loopback* program. Next, the RCU registers are set to receive CTP triggers and the correct L1 trigger latency.

DCS board version	2.8UiB
DCS board number	dcs0031
RCU version	v1.4
RCU power	4.3 V
Fee power	3.3 V
L1 Latency	0x4006 - 0x104
Trigger Conf.Reg.	0x5102 - 0x1000
altro-loopback version	0.9

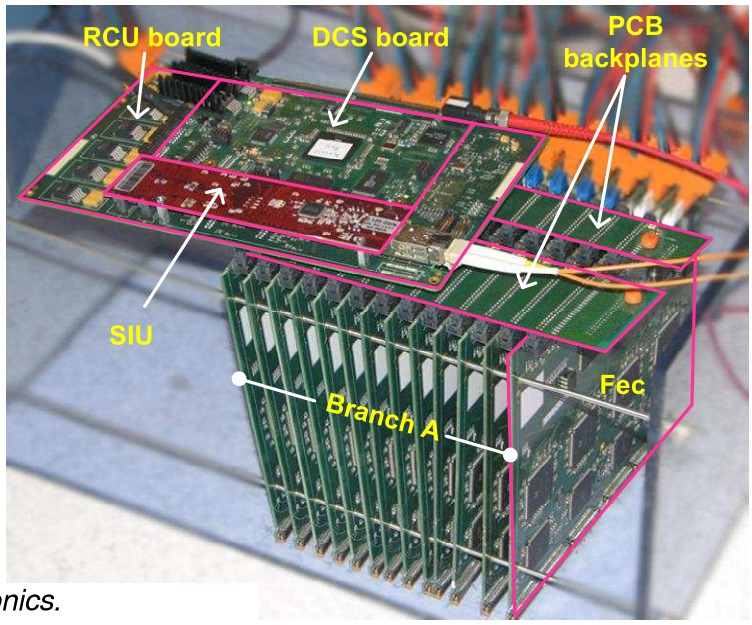


Figure 5-11: Front-end electronics.

5.6.4 LTU

The TTC partition is used in standalone mode to issue L2a trigger sequences with the CTP emulator program running on the SBC module. Lemo cables connect the LTU and TTCex module together, and fibre cables are connected from the TTCex module to the DCS board on the BusyBox and RCU.

LTU firmware version	b0
LTU software version	1.5.0
Lemo wiring for the TTC partition (b0) LTU => TTCex	
CLK2	BC
Orbit	B1
L1	A1

Figure 5-12: TTC partition

5.6.5 Testing

It is important to configure the test hardware correct or else it will not work. All the configurations are done from the LDC. The RCU has to be programmed and configured as described in section 5.5.3. After the BusyBox has been programmed and initialized, *runControl* can be started as DATE user. The CTP emulator is loaded with the L2a trigger sequence and the test setup is ready to go. There are a handful of scripts on the LDC, RCU and BusyBox which are helpful during tests, and are used to read or write several registers at once. The *runControl* Data Taking must be stopped and restarted whenever the BusyBox is reset, or else the D-RORC event FIFO is not reset and will get out of synch with the BusyBox.

Since the lab only has enough Front-end cards to one RCU, the tests are preformed with one channel on the BusyBox. Testing the BusyBox involves issuing triggers with the CTP emulator and check if the event validation takes place, by reading the appropriate registers in the BusyBox firmware. The LVDS link between the BusyBox and D-RORC is monitored with an oscilloscope using probes connected to the TP cable link. This can tell if the BusyBox sends request commands to the D-RORC and if the D-RORC replies on these requests. The busy output is also monitored when determining the upper trigger rate.

Initial tests of the BusyBox shows that it handles all legal trigger sequences and asserts the *busy* signal if the fee buffers are full. The upper trigger rate is determined to be 9 kHz, which is above the expected trigger rate of 1 kHz from p-p collisions.

5.7 Communication Tests

The BusyBox word has a 4 bit Command type and Request ID, and each is encoded with Hamming (8:4). The Hamming encoded bits are concatenated to a 16 bit data word before start, parity and stop bits are added to form a complete message. The BusyBox has four commands it can send to the D-RORC(s), see Table 3-2. The MSB of the data word is transmitted first, and the Hamming decoding is preformed on bit 0 and down to bit 7, and to bit 8 down to 15. The result has to be interpreted from right to left, when measuring with the oscilloscope, to give the correct result when reading the BusyBox messages.

The D-RORC sends three 16 bit words which are concatenated to one 48 bit message by the BusyBox, see Table 3-3 for a bit map of the D-RORC message. These words are not Hamming encoded and have only parity check. The messages are sent with one high bit period between them which correspond to 25 ns. The MSB of the each D-RORC data word is transmitted first.

An oscilloscope is used to monitor the communication between the BusyBox and D-RORC during communication tests. Probes are connected to the transmitting and receiving LVDS lines to measure the differential signalling to and from the BusyBox.

First, an L2a trigger sequence is issued using the CTP emulator, and the oscilloscope triggers on the BusyBox transmission. Figure 5-13 shows parts of a transmitted and received message. The time it takes the D-RORC to answer varies, but during test the delta time between the two signals is about 250 ns ± 30 ns.

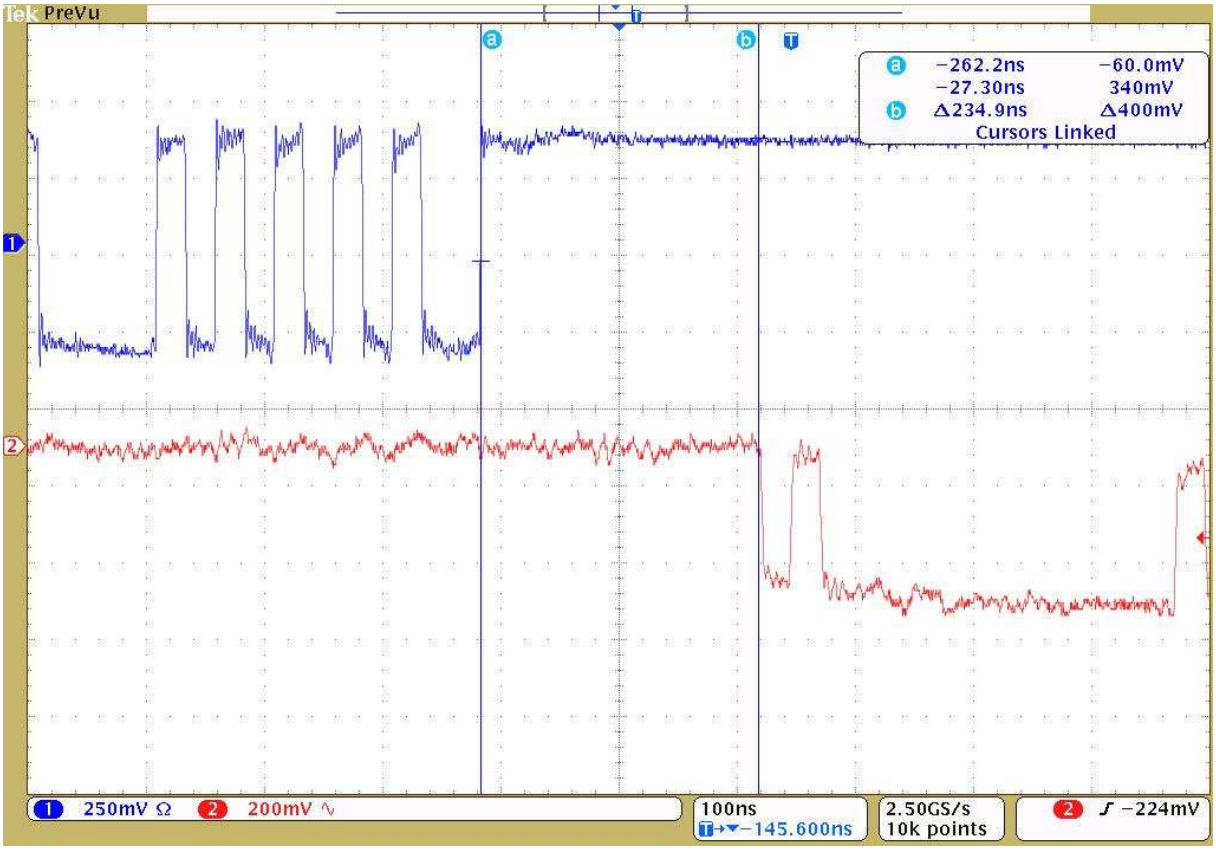
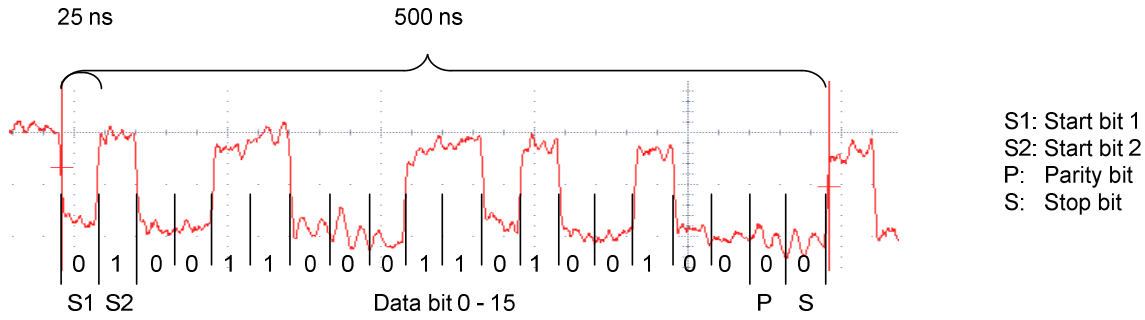


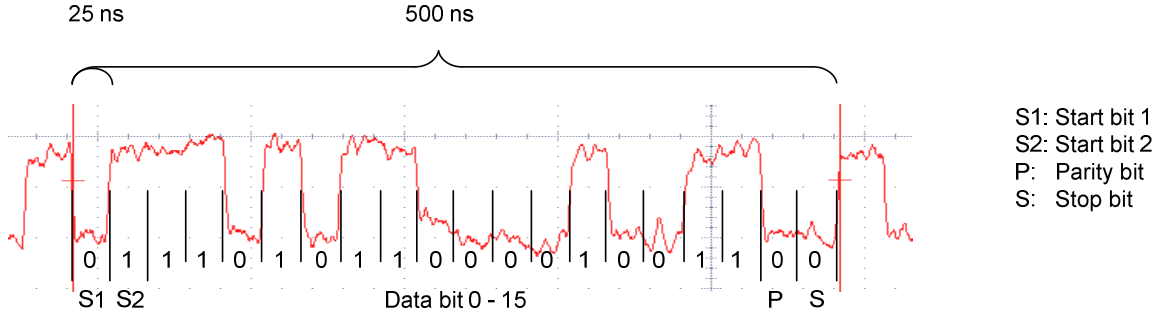
Figure 5-13: Screenshot of an LVDS transmission between BusyBox (blue) and D-RORC (red).

The event ID from the L2a trigger sequence in this example is 0x58CC86B94. The BusyBox will transmit the Request event ID command and Request ID (0x2) to the D-RORC. The D-RORC will answer and transmit the Request ID it received from the BusyBox (0x2), Bunchcount ID and Orbit ID from the Fee, and the D-RORC ID (0x7). The response from the D-RORC is shown in Figure 5-14, 5-15 and 5-16. From these figures we can see that expected answer complies with the transmitted message.



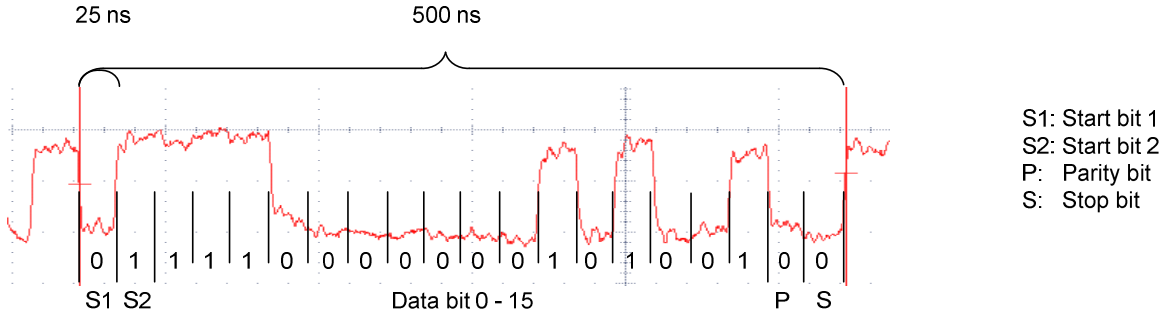
Word(1) 0 -15: 001100011010 0100 (Reverse order)
Request ID: 0010 -> 0x2
Bunchcount ID: 010110001100 -> 0x58C

Figure 5-14: First part of the LVDS message sent from D-RORC.



Word(2) 0 -15: 1101011000010011 (Reverse order)
Orbit ID: 1100100001101011 -> 0xC86B

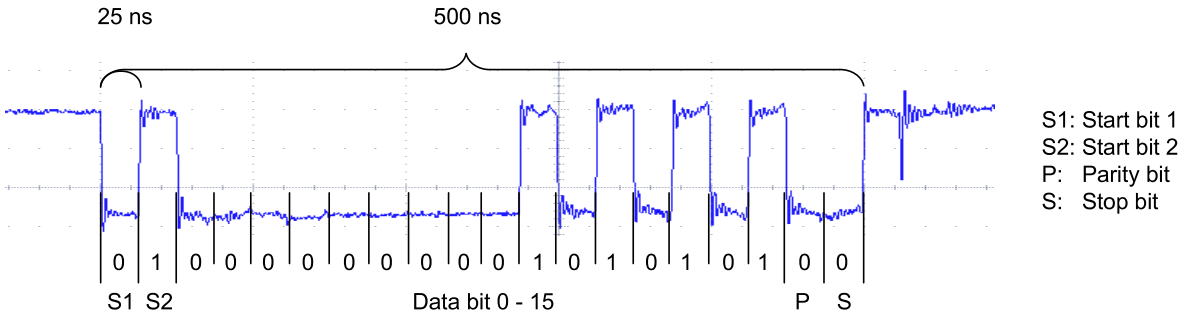
Figure 5-15: Second part of the LVDS message sent from D-RORC.



Word 0 -15: 11100000 00101001 (Reverse order)
Orbit ID: 10010100 -> 0x94
D-RORC ID: 11100000 -> 0x7

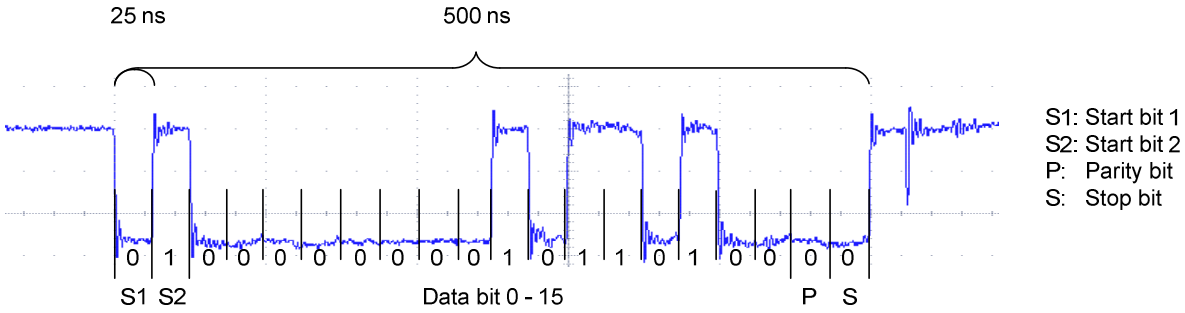
Figure 5-16: Third part of the LVDS message sent from D-RORC.

The four BusyBox Command types are documented in Figure 5-17, 5-18, 5-19 and 5-20. See Table 3-2 for description of the Command types.



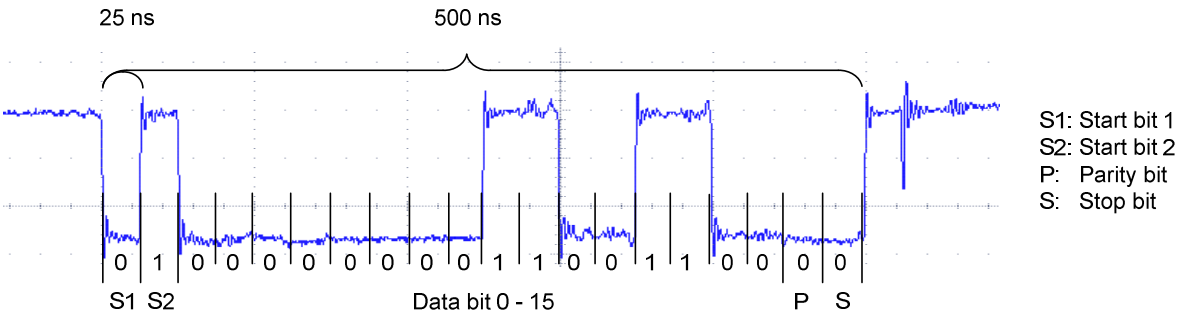
Word 0 -15: 00000000 01010101 (Reverse order)
Command type: 0100
Request ID: 0

Figure 5-17: Overview of the Request event ID (0100) command transmitted from the BusyBox. Red digits are data and black digits are parity from the Hamming (8:4) encoding.



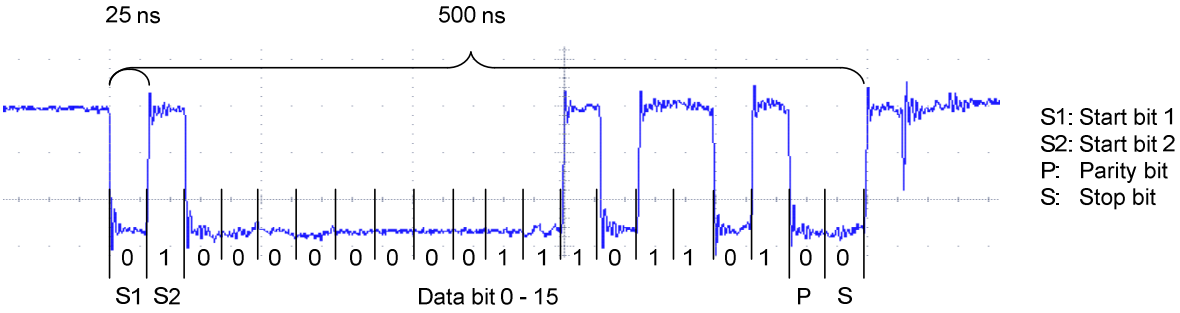
Word 0 -15: 00000000 10110100 (Reverse order)
Command type: 0101
Request ID: 0

Figure 5-18: Overview of the Resend last message (0101) command transmitted from the BusyBox. Red digits are data and black digits are parity from the Hamming (8:4) encoding.



Word 0 -15: 00000000 11001100 (Reverse order)
Command type: 0110
Request ID: 0

Figure 5-19: Overview of the Force pop event ID (0110) command transmitted from the BusyBox. Red digits are data and black digits are parity from the Hamming (8:4) encoding.



Word 0 -15: 00000000 11101101 (Reverse order)
Command type: 0111
Request ID: 0

Figure 5-20: Overview of the Force request ID (0111) command transmitted from the BusyBox. Red digits are data and black digits are parity from the Hamming (8:4) encoding.

5.8 Evaluation of Hardware Tests

All the basic tests conducted so far, revealed no errors. The BusyBox handles all legal trigger sequences, see Table 2-1, i.e. the Trigger Receiver module decodes triggers and the Busy Controller module keeps track of free buffers. The *busy* is asserted if the Fee buffers are full. After an L2a trigger sequence the Event Validation module reads out the event ID from the Trigger Receiver, and send a request to the D-RORC. The reply from the D-RORC is validated and the event valid flag is set. The LVDS communication has been documented, and it can be confirmed that the D-RORC messages are written to the RX Memory module.

Chapter 6

Summary, Outlook and Conclusion

This chapter discus the work covered by this thesis and some of the experiences that has been gained.

6.1.1 Summary

The result of this master thesis is a specification document for the BusyBox, a working hardware test setup and a new firmware version for the BusyBox. The specification document describes the BusyBox firmware and related hardware in detail, along with a BusyBox user guide. The test setup has the necessary equipment to test the basic functionalities of the BusyBox. Beyond that, a firmware version which incorporates a new trigger specification has been developed and simulated with VHDL testbenches. I have also updated the IFT Nuclear Physics group's wiki page, which is chosen as a location for a knowledge base for the BusyBox.

Prior to my work with the BusyBox, former master student Magne Munkejord, developed the firmware and a communication protocol for the BusyBox. It was assembled and put into service at CERN in 2008. Later, some problems were discovered and the most important problem was that the BusyBox went into saturation when the trigger rate passed 500 Hz. The expected TPC event rate for p – p collisions is 1 kHz, i.e. the BusyBox became the bottle neck and halted further triggers during tests.

Magne Munkejord has done a great job developing the firmware in such a short time, but at the expense of documentation and production testing. The topic of this thesis has primarily been to document and debug the BusyBox firmware. So, I started writing a combined specification document and user guide, and during this phase the saturation problem was solved. More bugs were reported due to inadequate testing; loss of configuration in the BusyBox firmware and spurious D-RORC communication.

The BusyBox had only been tested with one sector of the TPC (6 RCUs) in the RCU lab at CERN, and the loss of configurations occurred when all sectors of the TPC (216 RCUs) where in use. When both A and C sides of the TPC are used, the loads on the BusyBox's FPGAs are at the highest. The FPGAs have to be programmed on power up, and either the voltage regulators or power supply cannot provide enough power to the FPGAs, hence the configuration is lost. We could not verify this since the C side is not active due to maintenance.

See Appendix B for a description of the problem and the proposal we wrote for an enhanced start up procedure for the BusyBox. The problem was first handled by a work-around to take data at CERN before a solution was found. Csaba Soos, who design the D-RORC firmware,

fixed the problem which was related to a reset issue concerning the D-RORCs firmware. The firmware works in two clock domains. During a reset before a test run in ALICE, the faster clock would miss the assertion of the reset signal by the slower clock in the D-RORC firmware, resulting in de-synchronization between some D-RORCs and the BusyBox.

A test setup was installed in the microelectronics lab here in Bergen to debug and test the BusyBox. This included a PC with DATE, D-RORCs, Front-end electronics and a TTC partition. The test setup was also used to test and document the basic firmware functionalities of the BusyBox. Installing the test setup proved to be more difficult than expected and required a lot of expertise, not only on the different Front-end electronics, but also on the DATE software and the TTC partition.

Johan Alme updated the Trigger Receiver module for the RCU and BusyBox, and we therefore had to test that it worked according to the specifications. The update was requested by the ALICE trigger group based on a possible error that can occur in during a trigger sequence, which could result in orphan trigger messages. Studies of the BusyBox firmware showed that this would affect the counting of free Fee buffers in the Busy Controller module. I upgraded the BusyBox firmware so it can handle orphan trigger messages.

6.1.2 Outlook

The BusyBox power supply must be tested when the C side of the TPC is up and running. This will show if inadequate power supply is the cause of configuration loss in the FPGAs.

The new update I made to the firmware was tested and simulated with a VHDL testbench develop by Magne Munkejord. It was very difficult to interpret the results from these tests, and I started to design a new testbench which is more similar to the Trigger Receiver testbench. It must be considered to continue the development of this new testbench. The basic functionalities of the new firmware must also be tested.

6.1.3 Conclusion

The BusyBox is the Achilles' heel of the ALICE DAQ system, which relies on a BusyBox that is working without a hitch. It is a fairly complicated system which resides in an even more complex system. Thus, it is important to have a BusyBox that is well documented, easy to understand and test if bugs are discovered.

For the first time here at IFT in Bergen, we have a full test setup for the BusyBox system, and initial test shows that the BusyBox is stable and works reliable. The first runs with the LHC beam will give designers of the ALICE system information on error situations that can be expected, and changes may occur to the BusyBox design.

I am confident that the work we have done in this thesis will ease the process of future firmware upgrades and testing of the BusyBox.

Appendix A

A.1 List of Abbrevations

ADC	Analog to Digital Converter
ALICE	A Large Hadron Collider
ALTRO	ALICE TPC ReadOut
ASIC	Application-Specific Integrated Circuit
BC	Bunch Crossing
BRAM	Block Random Access Memory
CERN	Conseil Européen pour la Recherche Nucléaire
CHD	Common Data Header
CHEN	CHannel ENable
CLB	Complex Logic Block
CTP	Central Trigger Processor
DAQ	Data Acquisition system
DCM	Digital Clock Manager
DCS	Detector Control System
DDL	Detector Data Link
D-RORC	Data ReadOut Receiver Card
EIDOK	EventID OK
EMCal	Electromagnetic Calorimeter
FEE	Front End Electronic
FIFO	First In First Out
FMD	Forward Multiplicity Detector
FSM	Finite State Machine
FPGA	Field Programmable Gate Array
HLT	High Level Trigger
JTAG	Joint Test Action Group

LED	Light Emitting Diode
LHC	Large Hadron Collider
LSB	Least Significant Bit
LTU	Local Trigger Unit
LVDS	Low Voltage Differential Signaling
MSB	Most Significant Bit
NRZ	Non Return to Zero
PASA	Preamplifier and Shaper ASIC
PCB	Printed Circuit Board
PHOS	PHOton Spectrometer
PISO	Parallel In Serial Out
QGP	Quark Gluon Plasma
RCU	Readout Control Unit
RORC	Read Out Receiver Card
TCP/IP	Transmission Control Protocol/Internet Protocol
TOF	Time Of Flight
TP	Twisted Pair
TPC	Time Projection Chamber
TRD	Transition Radiation Detector
TTC	Timing, Trigger and Control
UiB	University of Bergen
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Appendix B

Proposal for an enhanced start-up procedure for the BusyBox

Problem

In some cases, the DRORC and the BusyBox get out of sync. Issuing the reset on the DRORC and then on the BusyBox, breaks the communication on some of the channels, thus we get permanently busy.

Description of problem

After upgrading the DRORC firmware on *Friday the 13th 2009* at P2, each time when a run started one or more readout channels got stuck, and the busy was asserted by the BusyBox. Blocked channels were always randomly placed amongst the selected ones, and the possibility of broken cables can therefore be excluded.

BB/DRORC communication

The BusyBox and DRORCs have queues with event IDs. After the BusyBox receives an event ID from TTC, it sends a request to all DRORCs and they answer with the first event ID in their queue. If the event ID from the DRORCs matches the one from the BusyBox, it implies that the event has been read-out to DAQ, and that the event data buffers occupied in corresponding Fee cards are freed. To keep all the queues synchronized (BusyBox and DRORC), the BusyBox generates a Request ID each time it pops a new ID from the queue of event IDs received from TTC. The Request ID is included in the requests that are sent to the DRORCs. The DRORC will remember the Request ID from the last request sent from BusyBox to DRORCs and compare this with the Request ID in the present request. If the IDs match it implies that the BusyBox has not popped an event ID and neither should the DRORC. If the IDs does not match it implies that the BusyBox is now requesting the next event ID.

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
Query Event ID	1000	Request an Event ID, if any, from the D-RORC, but the D-RORC should not pop from its local queue
Reset Event ID	1100	Command the D-RORC to discard all events from its local queue

Table 6-1: BusyBox commands. Proposed new commands are shown in red.

Initialization / Synchronization at start of a new run

If the DRORCs for some reason are not synchronized with the BusyBox, for example if a DRORC has not been reset properly before a run is started we will get into trouble. The proposed enhancement of the BusyBox-DRORC communication introduces a query and a reset command in addition to the request-ID command. It is a topic for discussion if we want to query the DRORC before an eventual reset is issued or if the BusyBox should just send a reset/init command at the start of each run.

Normal case

After normal initialization of the BusyBox and the DRORCs, the Request ID, Bunch Count ID and Orbit ID are 0 (TBC). The BusyBox will start by sending a broadcast command, “1000”, to the DRORCs, and expect in response that all DRORCs return Request ID, Bunch Count ID and Orbit ID with value 0. The BusyBox then marks all channels as OK and is ready for operation.

Abnormal scenario 1

After the initialization of the BusyBox and the DRORCs, the Request ID, Bunch Count ID and Orbit ID are unequal to 0. The BusyBox will start by sending a broadcast command, “1000”, to the DRORCs, and the response is not all 0. The BusyBox will then notify the DCS and a decision can be made on how to proceed.

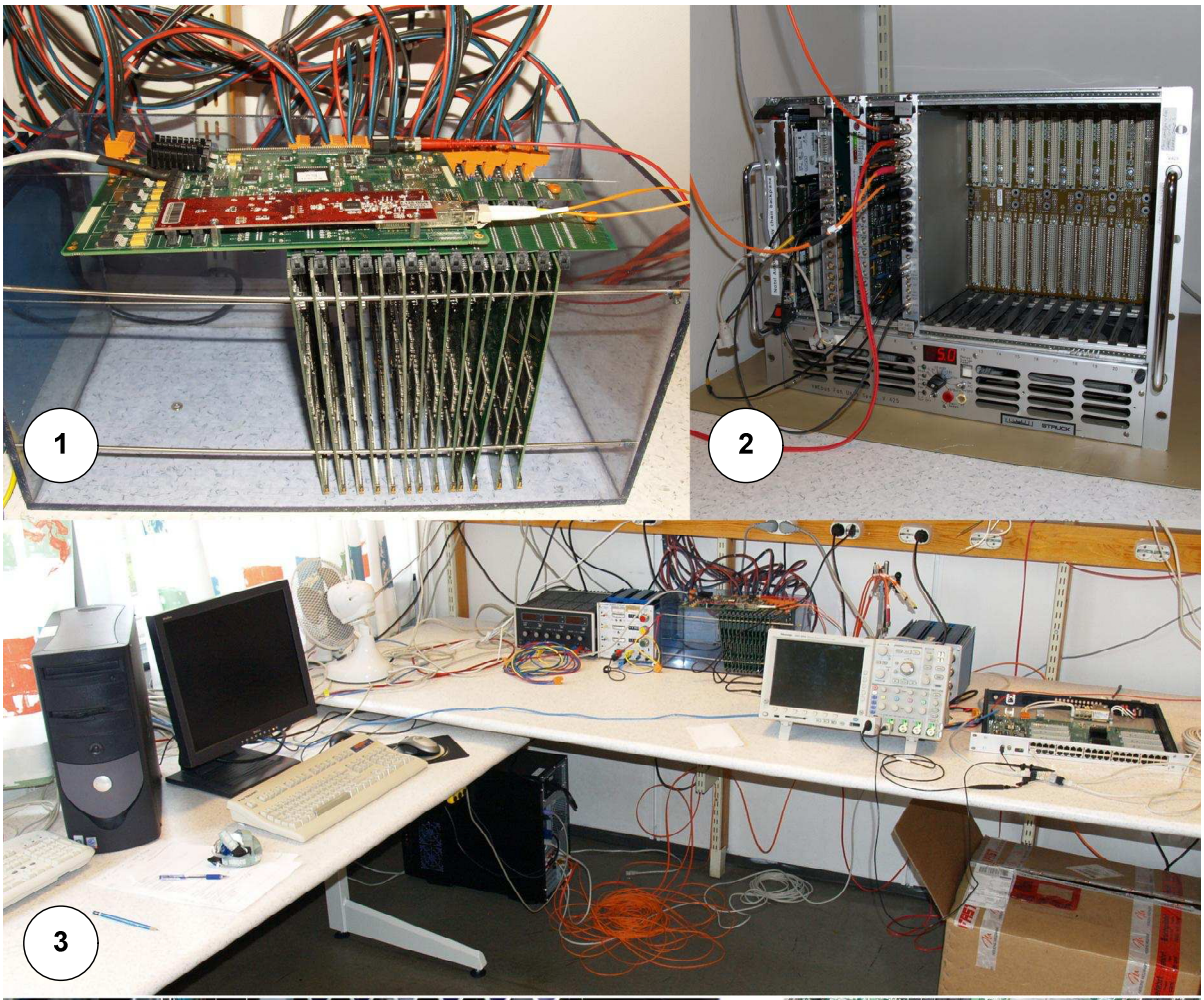
Abnormal scenario 2

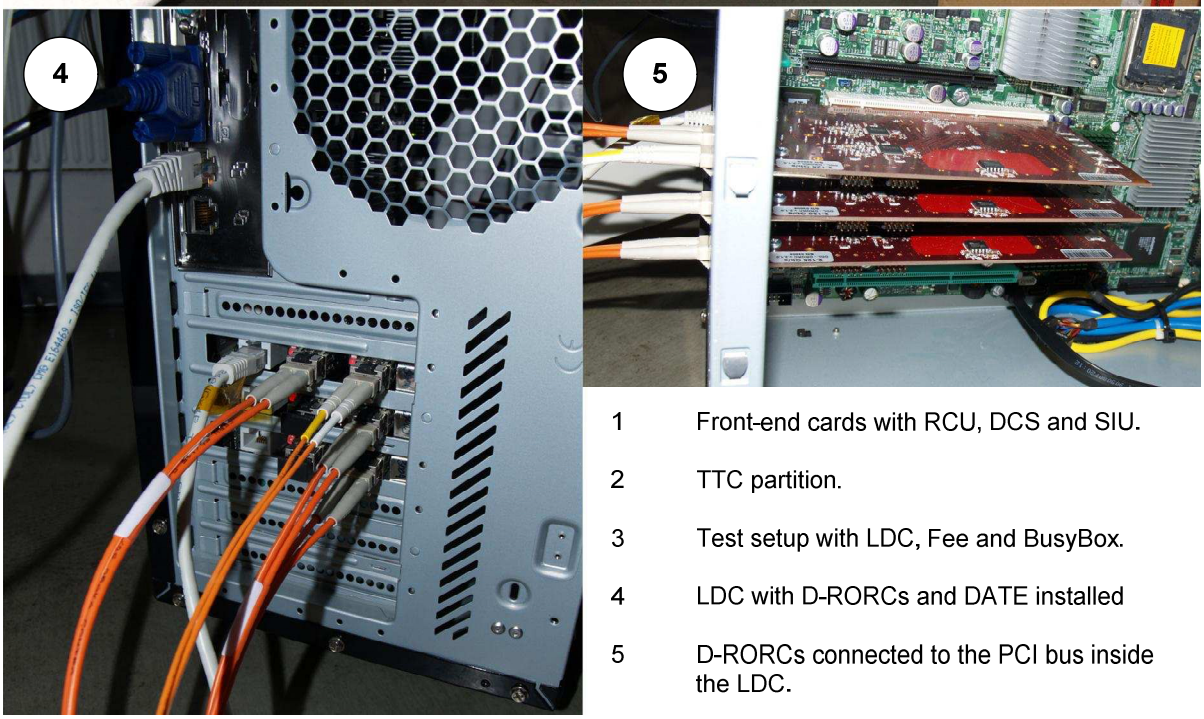
As in scenario 1, the response from the DRORCs is not all 0. The BusyBox can then proceed with issuing a “Reset Event ID” command. The expected response is again that all DRORCs return Request ID, Bunch Count ID and Orbit ID with value 0. If any results differ from 0 the

BusyBox will notify the DCS and a decision can be made on how to proceed. Else, the BusyBox will mark all channels as OK and is ready for operation.

Appendix C

Test Setup





1 Front-end cards with RCU, DCS and SIU.

2 TTC partition.

3 Test setup with LDC, Fee and BusyBox.

4 LDC with D-RORCs and DATE installed

5 D-RORCs connected to the PCI bus inside the LDC.

Reference Documents

1. CERN Public web site. *European Organization for Nuclear Research,* http://public.web.cern.ch, (2008).

2. Collaboaration, A., Journal of Physics G: Nuclear and Particle Physics. *ALICE: Physics Performance Report, Vomume II,* CERN, Switzerland, (2006).

3. ALICE TPC web site. *ALICE Time Projection Chamber (TPC)* http://aliceinfo.cern.ch/TPC/index.html, (2008).

4. Hans Muller's web site. *Alice PHOS Electronics,* http://hmuller.web.cern.ch/hmuller/, (2008).

5. Christensen, C.H., *ALICE Forward Multiplicity Detector,* PhD Thesis. Niels Bohr Institute, University of Copenhagen, Denmark, (2007).

6. ALICE EMCal web site. *Electro Magnetic Calorimeter (EMCal)* http://rhic30.physics.wayne.edu/, (2008).

7. Rossebø, A., *Busy-logikk for ALICE TPC,* MSc Thesis. Department of Physics and Technology, University of Bergen, Norway, (2006).

8. Munkejord, M., *Development of the ALICE Busy Box,* MSc Thesis. Department of Physics and Technology, University of Bergen, Norway, (2007).

9. Alme, J., *Firmware Development and Integration for ALICE TPC and PHOS Front-end Electronics,* PhD Thesis. Department of Physics and Technology, University of Bergen, Norway, (2008).

10. *ALICE, Technical Proposal for A Large Ion Collider Experiment at the CERN LHC,* 198. http://doc.cern.ch//archive/electronic/other/generic/public/cer-000214817.pdf, (1995).

11. L. Musa, J.B., N. Bialas, R. Bramm, R. Campagnolo, C. Engster, F. Formenti, U. Bonnes, R. Esteve Bosch, U. Frankenfeld, P. Glassel, C. Gonzalez, H.-A. Gustafsson, A. Jimenez, A. Junique, J. Lien, V. Lindenstruth, B. Mota, P. Braun-Munzinger, H. Oeschler, L. Osterman, R. Renfordt, G. Rüschmann, D. Rohrich, H.-R. Schmidt, J. Stachel, A.-K. Soltveit, K. Ullaland, IEEE Xplore. *The ALICE TPC Front End Electronics,* http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01352697, (2003).

12. D. Evans, S.F., G.T. Jones, P. Jovanovic, A. Jusko, L. Králik, R. Lietava, L. Šándor, J. Urbán, O. Villalobos-Baillie, *The ALICE Central Trigger System,* Article. (2008).

13. Bergeron, J., Springer. *Writing Testbenches: Functional Verification of HDL Models, Second Edition,* (2003).

14. Alme, J., *TTC receiver requirement specification_v1.2.doc,* Digital module requirement specification. (2008).

15. Microelectronics group, *VHDL Guidelines and Coding Rules,* University of Bergen Department of Physics and Technology, (2006), internal note.