Faculty of Physics and Astronomy

University of Heidelberg

Diploma thesis
in Physics

submitted by
Rainer Schwemmer
born in Sulzbach-Rosenberg

March 2007

Commissioning of the LHCb Outer Tracker Front-end electronics

This diploma thesis has been carried out by Rainer Schwemmer at the Physical Institute under the supervision of Prof. Dr. Ulrich Uwer

Kurzfassung

Das Physikalische Institut der Universität Heidelberg ist maßgeblich an der Entwicklung und Produktion des äußeren Spurkammersystems des LHCb Experiments beteiligt. Dies beinhaltet den Entwurf und die Produktion der Ausleseelektronik.

Zur Überprüfung dieser Elektronik wurde ein automatischer Teststand entwickelt. Teil dieser Entwicklung war das eingliedern der Slow Control in die bereits existierende FPGA Firmware. Weiterhin wurde die nötige Software für die FPGA Kommunikation programmiert.

Als nächster Schritt wurde ein Steuerungsprogramm für das äußere Spurkammersystem mit PVSS entwickelt. Dieses Programm war eines der ersten, mit dem Zweck einen kompletten Subdetektor zu steuern. Diese wegweisende Entwicklung führte zu wesentlichen Verbesserungen des allgemeinen LHCb Softwareframeworks.

Das Programm erlaubte zum ersten mal die Steuerung eines erheblichen Teils der Ausleseelektronik. Es bereitete den Weg für die Überprüfung des Systemverhaltens mit der endgültigen Hardware. Neben anderen Untersuchungen erlaubten die angesammelten Daten eine erste Analyse der Zeitkoordination der Hardware.

Abstract

The Heidelberg Institute of Physics is involved in the development and production of the LHCb Outer Tracker system. This included the design and production of readout electronic components.

In order to verify these components an automatic test system was developed. This included the integration of the slow control into an existing FPGA firmware. Furthermore the software to communicate with the FPGA was implemented.

As a next step a PVSS control system for the Outer Tracker was developed. It was one of the first systems with the purpose of controlling a whole LHCB sub detector. This pioneering work led to a number of improvements in the design of the common LHCb software framework.

The control system allowed operating a significant fraction of the readout electronics for the first time. It provided the opportunity to test system performance with the final experiment hardware. Among other studies, the recorded data allowed preliminary analysis of the time alignment procedure.

Contents

Chapter 1

Introduction

The field of High Energy Physics deals with the analysis of the smallest known entities in the universe. One goal is to comprehend the interaction between matter and radiation.

The standard model of particle physics is the accepted explanation of microscopic interactions. According to the model, all physical processes can be attributed to twelve elementary particles and their anti particles, as well as the so called force carriers which propagate interactions between particles.

Particles come in two general types: Fermions and Bosons. Fermions are subdivided into three generations, each containing a group of two quarks, one lepton and its associated neutrino. Fermions are the basis for everything tangible. Their half numbered spin prohibits them to ever take the same quantum mechanical state even without interaction through force carriers (Pauli principle).

Bosons are the propagators of the three forces: Strong, Electromagnetic and Weak. Their associated particles are the Gluons, Photons and W and Z Bosons. The gravitational force plays a special role. While it also is a kind of interaction, its influence on the sub atomic scale is so small that it is negligible for particle interactions.

Figure 1.1 shows an overview of the known elementary particles. The mass of corresponding particles between two generations is increasing with each generation, with the heavier quarks reaching masses of up to $172\,\mathrm{GeV/c^2}$.

While the standard model describes the sub atomic world in great detail and verifiability, it is a model and not a theory, because it only describes how the particles interact. It does not give any answers to why the particle's properties like mass, charge or flavour are the way they are or have the properties they have.

1.1 LHC

Due to the heuristic character of the standard model, it only describes processes that have been observed before. It is the purpose of experimental machines like the Large Hadron Collider (LHC) to expand the horizon of the standard model

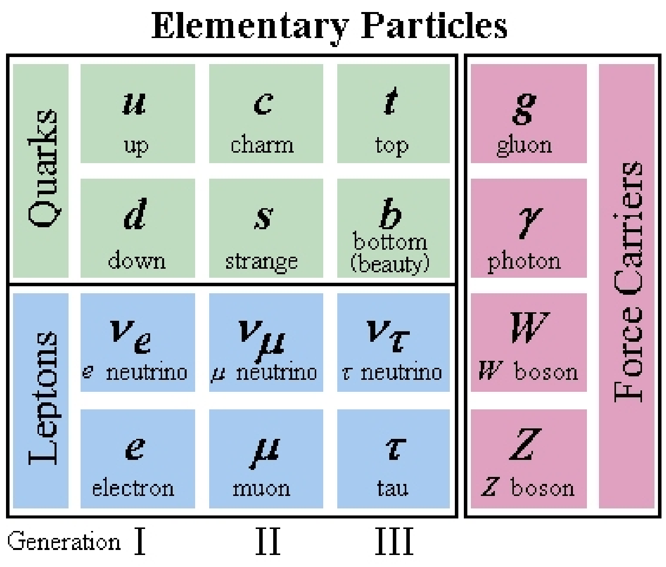


Figure 1.1: *The standard model particles*

and give access to new physical processes that are expected due to observations made in nature.

LHC is a ring particle accelerator at the Conseil Européen de la Recherche Nucléaire (CERN) at the Swiss-France border near Geneva. LHC is built underground and is the successor of LEP, an electron positron collider.

Within its 27 km circumfering ring, two anti parallel proton beams are accelerated to energies of 7 TeV each. The protons are brought to collision at four interaction points. During collision, the protons released their energy as showers of different particles described by the formula: $E = m \cdot c^2$. The high energies are necessary to create large amounts of the more massive quarks and bosons.

Large particle detectors are built at each of the four interaction points to investigate different aspects of these collisions.

- ATLAS and CMS are multipurpose detectors and are built to give access to all of the available physical activities.
- LHCb is interested in rare B-meson decays to study CP violation and give answers to the question about the ratio of matter and antimatter in the universe.
- The ALICE experiment focuses on the detection of gluon plasmas in heavy ion collisions. The LHC can be filled with lead and other ions instead of protons for this experiment.
- TOTEM shares an interaction point with CMS and will measure the total cross section as well as scattering and diffractive processes of proton-proton interactions at low deflection angles.

- LHCf will study the collision products at very small scattering angles. It is ultimately trying to improve the models of showers originating from cosmic particles colliding with the earth's atmosphere. It shares an interaction point with the ATLAS experiment.

1.2 LHCb

The main focus of the LHC beauty Experiment (LHCb) is to do precision measurements of CP violating processes and rare decays in b-quark systems. [1] The high energies of LHC give access to a full spectrum of b-quark particles. The gluon fusion process responsible for the b-particle creation will eject them predominantly in forward and backward direction, along the proton beams. To take advantage of this process, the LHCb detector is designed as a single arm spectrometer as seen in figure 1.2.

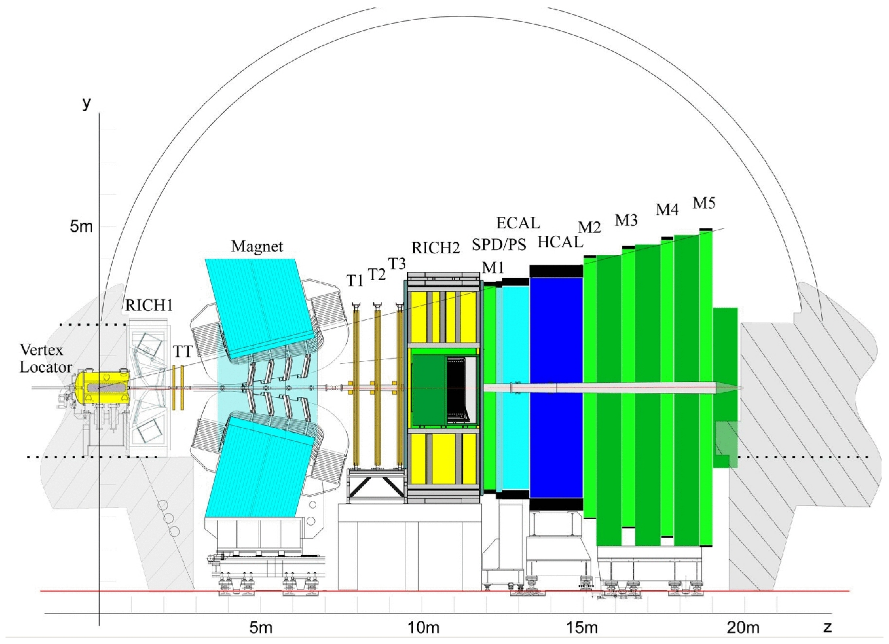


Figure 1.2: *Side view of the LHCb detector*

The detector is built from several sub components which all fulfil specific tasks. The Vertex Locator (VELO) is used to determine the exact point of interaction and any secondary decay vertices. It requires the best spatial resolution and is built from silicon strip detectors.

Particles and their energies are identified by the Ring Imaging CHerenkov (RICH) system, Electromagnetic and Hadronic Calorimeters (ECAL and HCAL) and the Muon System.

A magnet is used to deflect the trajectory of charged particles and to determine their momentum with the help of the tracking system. The tracking system consists of four sub systems. The VELO and the Trigger Tracker (TT), which are located near the interaction point and provide a first track segment before the magnet. The Inner and Outer Trackers (IT and OT) are positioned behind the magnet and are used to reconstruct the deflected track, performing the momentum measurement.

1.3 LHCb Outer Tracker

The LHCb Outer Tracker is a detector built from drift chambers. The chambers are made of so called straws that are 2.5 m long and have a diameter of 5 mm. Each straw contains a 25μm conducting wire, which is clamped inside the straws and kept at a potential of 1550 V. The straws themselves are coated with conducting material on the inside to provide an anti pole for the wire.

If a charged particle passes through a straw, it will ionise the atoms of a special counting gas[1] inside the straw. The electrons produced during the ionisation process will start to drift towards the wire in the middle of the straw. The primary electrons will be accelerated towards the wire by the electric field. The strong electric field around the wire will cause them to produce an avalanche of secondary electrons.

The charge deposited on the wire produces an electrical signal which is amplified, shaped and discriminated. The time between the hit of the ionising particle and the signal on the wire is proportional to the distance between hit and wire. This allows for spatial resolutions of up to 200 μm.

The Outer Tracker contains approximately 55.000 of these straws. They are incorporated into so called modules. Each module is five meters long, 34 cm wide and contains 128 straws. The straws are divided in the middle and can be read out individually on both ends. This separation is done to lower the hit occupancy for each straw.

The modules are combined to six by five meter wide layers. Four layers are grouped into one tracking station. The modules in one station are arranged in a way that puts the wires inside the front and back layers in a vertical position and the wires inside the two middle layers $\pm 5^\circ$ tilted to the vertical position. This geometry allows for best resolution along the bending axis of the magnet but also medium resolution perpendicular to it. The vertical layers are called X-layers, while the tilted layers are U- and V-layers.

The modules are supported by C-Frames. The frames are made from aluminium and house the necessary power and electronics infrastructure for the modules. Two adjacent frames leave a hole around the beam pipe and carry two of the aforementioned layers of modules on their front and back side respectively.

To allow a unique addressing of modules inside the detector, each layer is subdivided into four quadrants which contain, the upper or lower half of one

[1]The proposed gas is $ArCO_2$

layer on a C-Frame. This subdivision is illustrated in figure 1.3.

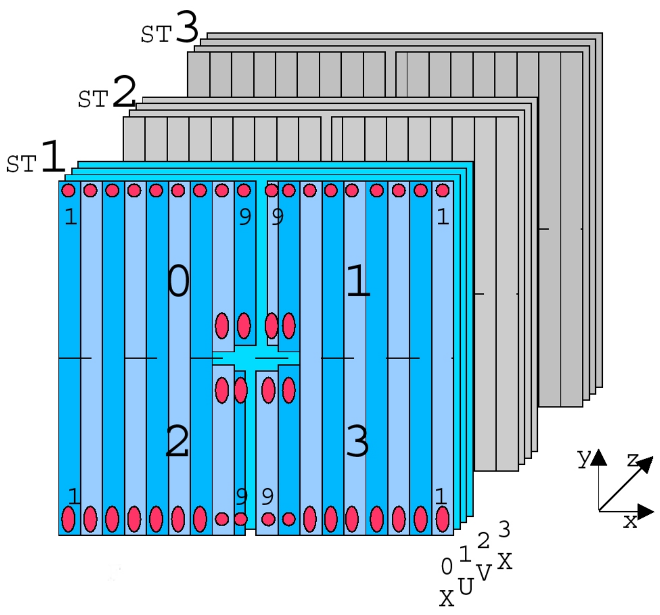


Figure 1.3: *Outer Tracker detector layout. The detector is subdivided into three stations containing four layers each. The layers themselves are subdivided into quadrants, dividing the modules at the point where the straws inside the modules are subdivided.*

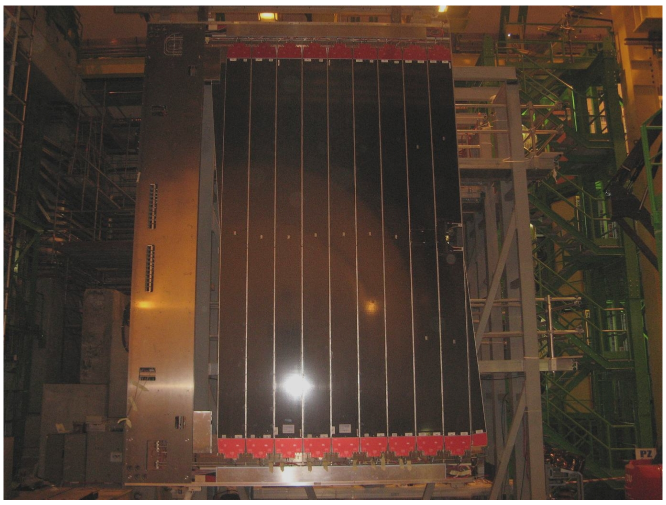


Figure 1.4: *A C-Frame on its way to the detector. It is the first C-Frame, fully equipped with Front-end Boxes.*

Chapter 2

Readout and Control electronics

Since the readout and control components of LHCb, especially of the Outer Tracker, are playing an integral part in this thesis, a short overview over the control and Data Acquisition (DAQ) systems shall be given here.

2.1 LHCb DAQ and Slow Control System

2.1.1 Data Acquisition and Fast Control

The proton beams within the LHC ring are not continuous but are grouped in packets or *bunches*. At the interaction point, these bunches will collide at a constant rate of 40 MHz. Multiplying this rate with the total number of data channels from all sub detectors and one byte per channel yields a data rate of approximately 40 TB/s. A filtering scheme has to be used to reduce this rate into the range where the data stream can be saved to hard drives. This filter network is depicted in figure 2.1.

The idea is to do a coarse analysis of the data at high rates and *trigger* the readout of an *event* only if it is interesting. This trigger decision is done on two levels of the readout chain.

The highest instance of these triggers (L0) are distributed by the so called Timing and Fast Control (TFC) system. It also supplies all front-end electronics with the 40 MHz clock signal that is synchronous to the collision frequency of the beam.

The main component of this system is the Readout Supervisor (ODIN) [19]. It takes the L0 decisions from the trigger logic and broadcasts it to the front-end electronics via an optical fibre network.

Each front-end component is equipped with a small buffer to store the data for 4 μs until the L0 decision arrives.

If an event is accepted by the L0 trigger, it is forwarded to the Trigger ELectronics and Level 1 (TELL1) [3] boards via optical data fibres. Originally, a second trigger was foreseen at this level to reduce the 1.1 MHz event rate of the

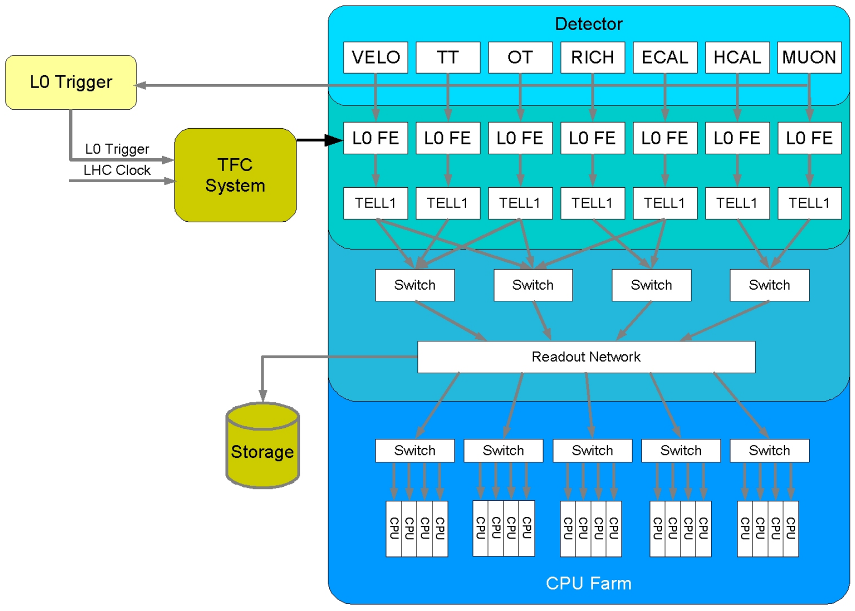


Figure 2.1: *The LHCb event readout chain.*

triggered L0 data stream to 40 kHz. With the arrival of faster networking technology, this level was made obsolete. The TELL1 is now primarily used to receive the optical data stream from the front-ends and to combine it to network packets that are sent to the readout farm.

The event is forwarded to a high speed computer network and processed by higher level triggers that are completely implemented in software. These high level triggers will decide if an event contains one of the interesting B-decays and will save the data to hard drives.

At this point the data rate will have been reduced from 40 TB/s to a more manageable level of 34 MB/s.

2.1.2 Slow Control system

While the Fast Control (TFC) system controls the trigger process and keeps the front-end electronics in sync with the LHC beam by providing an electronic clock signal that is synchronous to the bunches in the beam, the Slow Control has to manage the operating parameters of the front-end electronics.

Slow Control requirements

To allow for different operating modes and compensation of production variances of the detector components, each front-end component can be adjusted with individual parameters. These parameters are often only configured during start up of the devices and don't have to be touched afterwards.

At the same time, each component will also offer status information and counter variables that are essential for the correct operation of the devices, but don't have to be read at the full 40 MHz of the detector.

The Slow Control system will do all these tasks and provide an interface for the control of the experiment and its sub systems to the human operator(s).

Slow Control hierarchy

The LHCb slow control or Experiment Control System (ECS) is implemented as a linear hierarchy structure. The top entity in this hierarchy has complete control over all sub detectors. The sub detectors will report their status to this entity and receive commands from it.

Each sub detector has its own sub systems that report to and receive commands form their respective sub detector control entities on the second level. This hierarchy is continued until the device level is reached with the leaves of the hierarchy tree.

The flow of commands in the hierarchy is from top to bottom, while the flow of process information from the hardware is from bottom to top.

LHCb appoints the upper levels of this hierarchy. Each of the sub detectors is responsible for the creation of their respective control hierarchies and has control over shape and partitioning of their hierarchies as long as certain guidelines are adhered to.

2.2 Outer Tracker readout and Slow Control

The Outer Tracker readout and control system can be grouped into functional units that build the structure of the entire sub detector. Figure 2.2 shows an image of these units. Two times nine Front-end Boxes are connected to one Distribution Box [4] and two TELL1 boards.

The Distribution Box converts the optical signal of the TFC system into electrical signals and forwards them to the readout electronics. It is also responsible for the distribution of the Slow Control. The connection between Distribution Box and front-end electronics is established by SCSI cables. Each C-Frame contains two Distribution Boxes. They control the upper and lower rows of Front-end Boxes on the frame respectively.

The TELL1 board receives the L0 event data as optical signal and converts it back into electrical signals.

2.2.1 Outer Tracker DAQ components

The readout or L0 electronics are realised as so called Front-end Boxes that are mounted onto both ends of a wire chamber module. Amplifier Shaper Discriminator with Baseline Restoration (ASDBLR) boards are used to amplify and

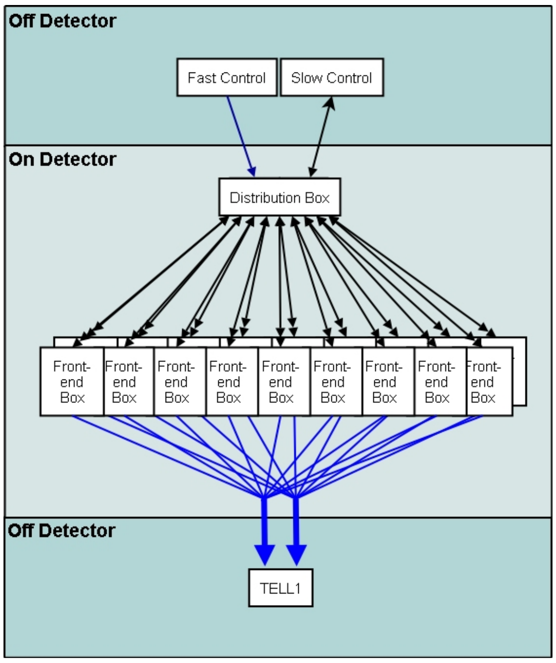


Figure 2.2: *Outer Tracker readout and control unit*
Blue connections denote optical connections, black are electrical.

discriminate the wire signals before the drift time measurement is performed. The ASDBLR measures the voltage across a capacitor that is charged by the electrons deposited on the wire for this. If the voltage rises higher than a configurable threshold value, the ASDBLR will notify the OTIS chip.

The Outer-Tracker Time Information System (OTIS) Time to Digital Converter (TDC) chip compares the time the ASDBLR signal arrives at with the clock signal from the TFC system, which is synchronous to the proton collisions. The chip calculates the electron drift time from this information and outputs it as digital signal. The OTIS subdivides the 25 ns clock signal into 64 time bins and has an absolute drift time resolution of under 390 ps[1].

The digitised drift time information of four OTIS boards is combined by a Giga bit Optical Link (GOL) [6] chip and sent to the off-detector TELL1 board via optical fibres. The GOL chip is mounted to the GOL/AUX board. This board was designed and produced at the University of Heidelberg [7]. It combines the drift time signals from the four OTIS boards and provides the Slow and Fast Control distribution for all chips inside the Front-end Box.

An overview of all front-end components is shown in figure 2.3 and 2.4.

Eight preamplifier, four OTIS and one GOL board are connected together and assembled to one Front-end Box. Each box can process the data of 128 straw tubes.

[1]The LHCb specification requires this measurement to have a resolution of at least 1 ns

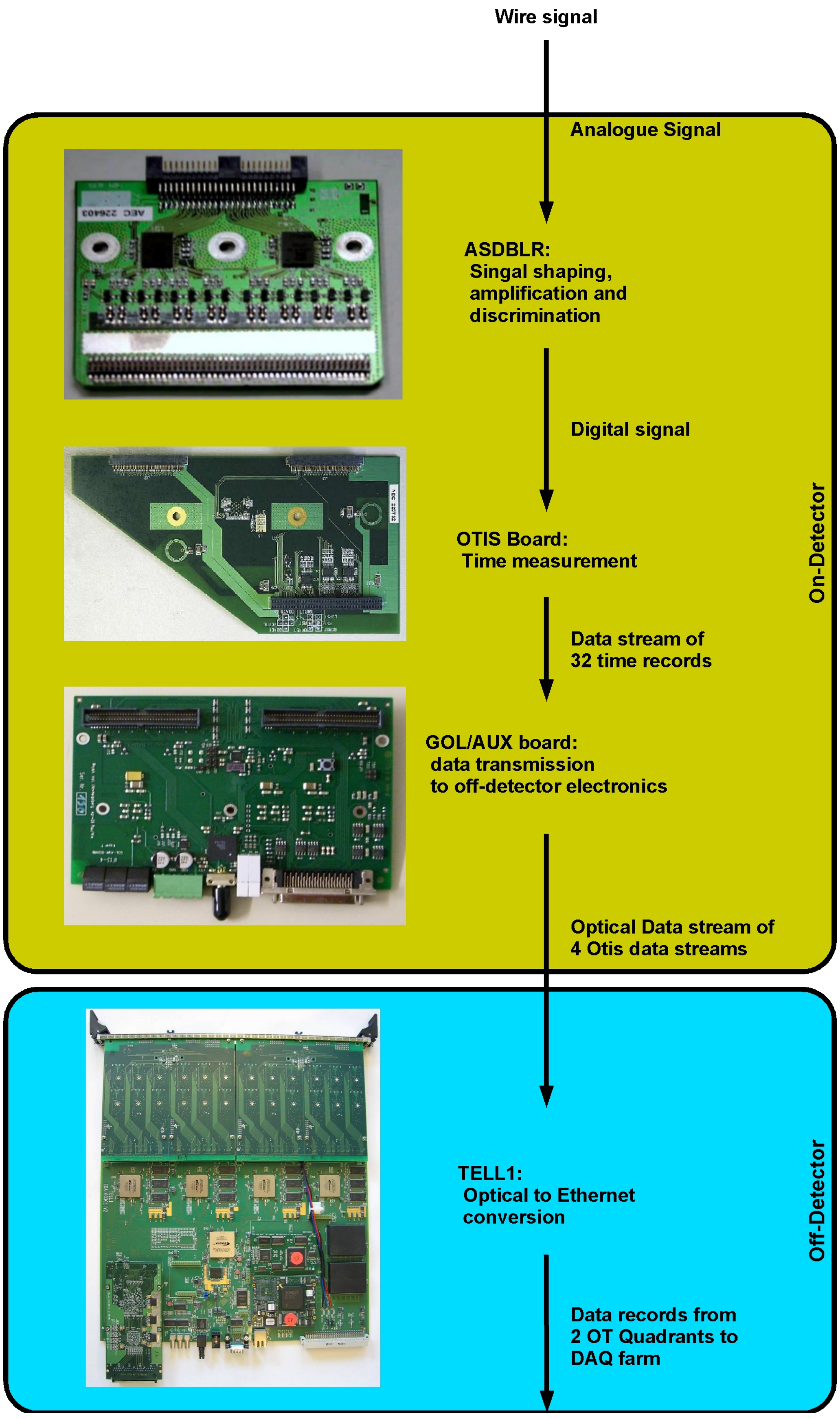


Figure 2.3: *Outer Tracker readout chain components*

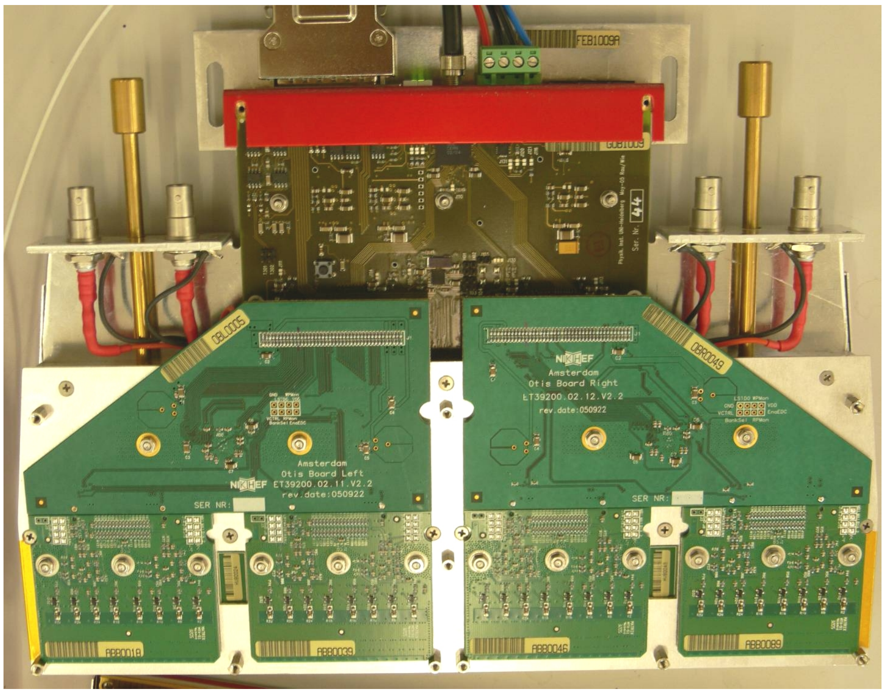


Figure 2.4: *Fully equipped Outer Tracker Front-end Box containing ASDBLR, TDC and GOL/AUX boards.*

Since the boxes are directly connected to the detector, they are within the radiation zone and all components have to be radiation tolerant. This is achieved by utilising the 0.25μm process for the chips and redundant logic circuits that control each other.

2.2.2 Outer Tracker Slow Control distribution

The distribution of the Outer Tracker Slow Control is done in parallel with the Fast Control. To protect the computers hosting the Control Systems from radiation, they are set up behind a radiation shield besides the detector. Fast and Slow control are transported to the detector as optical respectively electrical signals and have to bridge a total distance of approximately 70 m.

After the control signals, coming from the ECS, reach the Distribution Box, they are translated into an OT specific hardware communications protocol.

From the Distribution Box the signals are sent directly to the Front-end Boxes by the same SCSI cables that are also transporting the TFC signals. Inside the Front-end Boxes, the Slow Control is distributed further to the four OTIS chips and the GOL chip.

As shown in figure 2.2 each Distribution Box within one control unit is connected to 18 Front-end Boxes. The 18 boxes are again subdivided into two independent domains which are responsible for the control of one quadrant.

The Distribution Box itself also has components that have to be monitored and adjusted. It is also connected to the Slow Control system. It is however not part of the Front-end Box control domains but has it's own.

Chapter 3

Test setup for the mass production test of the OT readout electronics

The GOL/AUX board was developed and built by the Institute for Physics at the University of Heidelberg. It connects the four OTIS boards to the GOL chip and distributes Slow and Fast Control inside the Front-end Box.

Each board had to be tested after assembly. A setup was built to perform these tests. At the same time, a copy of this test setup is used for the test of the OTIS boards at NIKHEF in Amsterdam.

It was decided to enhance an existing test setup which was used for the mass production test of the OTIS chip and already contained DAQ and TFC emulation capabilities. [8][9].

3.1 Setup for the GOL/AUX board test

Figure 3.1 shows a schematic of the setup that had to be developed. It consists of a Windows PC, four reference OTIS boards and the GOL/AUX board which has to be tested. Inside the PC is an Altera Stratix-EP1S25 Field Programmable Gate Array (FPGA) which is mounted on a PCI card. The FPGA is used as the DAQ, Fast and Slow Control for the connected front-end boards.

The test had to verify the following functionalities of the GOL/AUX board:

- slow control distribution
- fast control distribution
- reset distribution
- power distribution and regulation

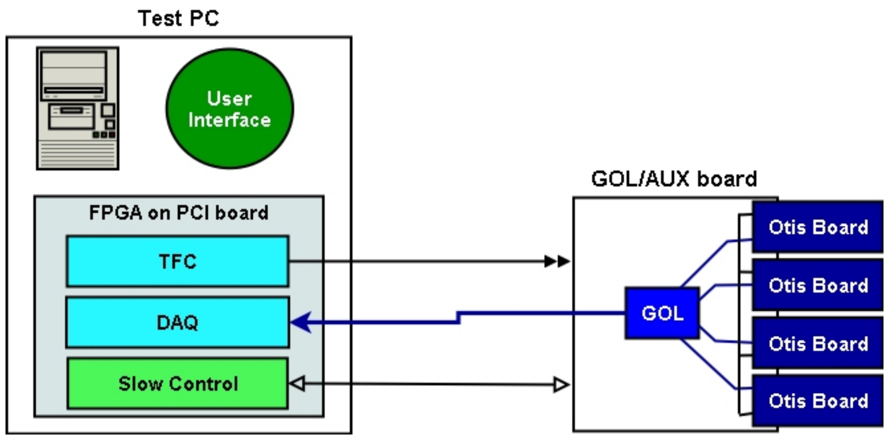


Figure 3.1: *GOL/AUX board test setup schematic*

- functionality of the GOL serialser
- quality of the optical signal transmission.

At the same time, the test had to be user friendly because it was planned to employ student assistants to perform it. It also had to be portable since a duplicate of it had to be sent to Amsterdam for the OTIS board test.

As mentioned earlier, a similar setup already existed. The old setup used a second PC as Slow Control interface while the TFC and DAQ were already implemented on the FPGA. This made the setup complicated and hard to transport.

It was the purpose of this part of the project to integrate the slow control into the FPGA firmware to allow better portability and automation of the test.

3.2 Slow control for the front-end electronics

3.2.1 Specifications for the test

The Slow Control for the OT electronics uses an I^2C bus as it's inter chip communication medium. The front-end chips are grouped into several I^2C control domains. Each domain contains nine Front-end Boxes connected to two independant Distribution I^2C buses. Each chip has a unique I^2C address inside these domains which identifies its position within the domain and on the detector.

For the mass production test, the slow control had to be able to access and modify all configuration registers of the following two chips:

OTIS Chip: This chip occupies only a single I^2C address. The chip has 56 registers that contain one byte of configuration data each. In order to address these registers the OTIS chip uses an internal register pointer that determines which register is currently visible to the I^2C bus.

After each read or write request, the register pointer is incremented by one. This mechanism allows to program or read multiple consecutive registers during one I^{2}C access.

GOL Chip: The GOL chip has 5 configuration registers which are also selected by a register pointer. Contrary to the OTIS Chip it occupies two consecutive I^{2}C addresses. The lower of the two addresses is used exclusively to change the register pointer, while the upper address represents the selected configuration register.

The chip features no automatic incrementation of the register pointer, hence every register has to be selected previously to accessing it.

3.2.2 Differential I^{2}C connection

To get the I^{2}C signal from the Distribution Box to the Front-end Boxes it has to travel approximately six meters of cable. I^{2}C was originally not designed for long range transmission but rather interconnecting devices on a single printed circuit board.

Another problem that had to be considered are grounding potential differences and ground loops occurring between two electronic components at these distances.

The solution was to use Low Voltage Differential Signals LVDS as means of transmission between Distribution Box and front-ends. A differential signal will also cause less noise and signal power loss during transmission.

The test setup had to support this modified I^{2}C version.

3.3 Implementation of the I^{2}C system

Migrating the Slow Control to the FPGA firmware of the test setup was now a matter of creating a PCI to differential I^{2}C interface FPGA program and creating a software for the test PC that would be able to access this interface.

3.3.1 I^{2}C standard

The Inter-Integrated Circuits (I^{2}C) bus is a serial data bus developed by Philips Semiconductors during the 1980s [10]. Communication on this bus is achieved by two bidirectional signal wires. The SDA line is used for the transmission of data while the SCL line is as data clock for the receiving party. The bus is made bidirectional by utilising an open drain circuit design. This architecture allows for multiple outputs to be connected to the same wire.

The communication protocol is based on the master-slave paradigm, where the master is the device that is currently in control of the clock line. Slaves are all other devices that are not in control of the clock line. Each device is allowed to act as master as well as slave. There can also be multiple devices on one bus trying to become a master. Care has to be taken that only one device is in control of the bus at any time.

Arbitration logic is used to resolve collisions between two devices that are trying to become the master. If a device tries to put the SDA line into a logic high state and the line stays low, it knows that there is another device trying to send. It will then wait until the current transfer is completed before making a new attempt.

The advantage of this logic is that no data distortion takes place if a collision occurs. As long as both devices are sending identical bits the data stays coherent. If the data bits differ, the device whose (1) bit was overwritten by a (0) will recognise this and retreat from the bus immediately. The data of the winner stays intact.

Selection of a particular device on the bus is done via a seven bit address pattern which has to be transmitted at the beginning of each read or write cycle. To distinguish between read and write cycles, an eighth bit is appended to the seven bit address.

Data can be transferred as an arbitrarily long stream of eight bit blocks during one cycle. Each eight bit block has to be acknowledged by a ninth bit sent by the receiving device.

The I^{2}C protocol defines a number of states which the I^{2}C bus can be in:

- Idle: Serial Data (SDA) and Serial Clock Line (SCL) are at logic high.

- Write: The current master is writing data to one or more slaves.

- Read: The current master is reading from one slave.

Transition between these states are actuated by the master by invoking certain conditions:

- Start Condition: Marks the transition from Idle to Read or Write state. The Read or Write states are selected by the eighth bit during target I^{2}C address selection. The target I^{2}C address has to be written to the bus, before any further reading or writing can be done.

- Repeated Start Condition: This condition takes the I^{2}C bus from Read to Write state or vice versa. It can also switch from Read to Read or Write to Write state but for different I^{2}C addresses. Again, the target I^{2}C address has to be written to the bus before any further read or write operation can take place.

- Stop Condition: The master releases the I^{2}C bus.

3.3.2 Special considerations toward differential I^{2}C

The differential I^{2}C method used for the front-end electronics needs some special considerations. LVDS drivers and receivers only allow unidirectional signal lines. The bidirectional I^{2}C lines had to be split up into unidirectional lines before they are fed to the LVDS drivers for transmission.

Since the OT front-end chips are all designed as I^{2}C slaves while the only I^{2}C master per bus is located inside the Distribution Box, the SCL signal is only sent into one direction and does not have to be separated. However, data needs to be transmitted in both directions and needs special treatment.

Figure 3.2 shows a schematic of the required modifications. The split SDA line can be connected together at the slave side after LVDS receival. On the master side, the lines have to stay separated to avoid deadlocks by signal feedback from the slave side.

This separation of the SDA line requires a modified kind of I^{2}C master with separated I/O ports.

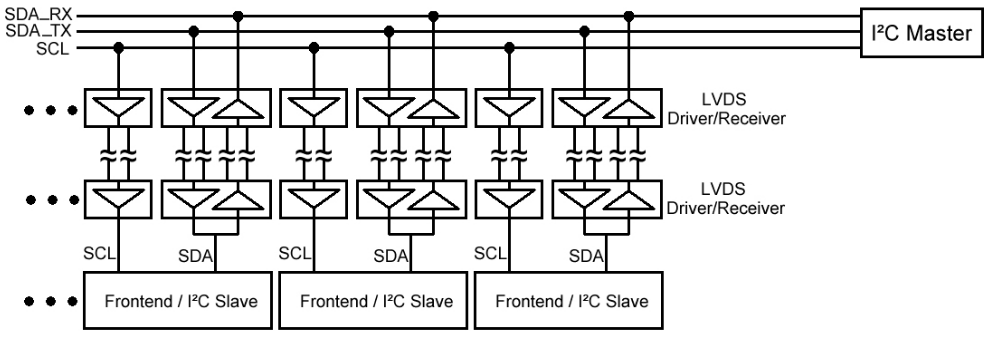


Figure 3.2: *Differential I^{2}C distribution scheme for the final experiment setup. The I^{2}C signals for the front-ends are transmitted as LVDS signals through six meters of SCSI cables. Each I^{2}C master controls nine Front-end Boxes*

3.3.3 I^{2}C interface design

Figure 3.3 shows a schematic of the old FPGA program. The sub systems like TFC and DAQ were connected to a command bus that broadcasts 32 bit wide data words from the PCI bus.

Data from the sub systems is transported to the PCI bus via independent 64 bit wide data buses.

The I^{2}C interface had to be integrated into this structure and receive it's data via the 32 bit commands from the PCI bus and forward the received I^{2}C data as 64 bit words to the PCI bus.

Figure 3.4 shows a block diagram of the created I^{2}C interface. The interface is subdivided into three functional blocks.

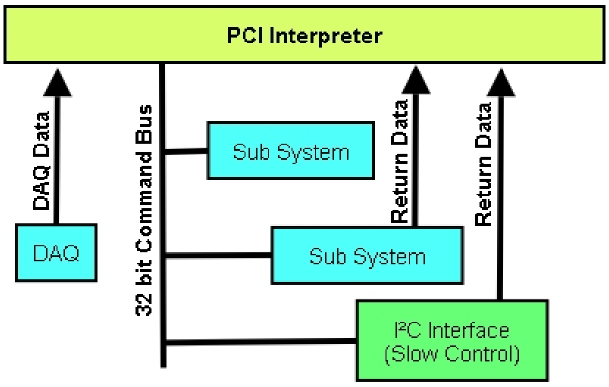


Figure 3.3: *FPGA sub system communications infrastructure.*

The Input Control block receives the 32 bit commands from the PCI bus and extracts the target I^{2}C address from the command. It assigns a unique *Ticket* ID to each I^{2}C access and prepares the received 32 bit commands for execution by the I^{2}C Control block. It also keeps track of the register numbers that are accessed and stores them in a FIFO buffer.

The I^{2}C Control block processes the prepared I^{2}C statements from the Input Control and passes them to an open source I^{2}C-Master core that handles the actual I^{2}C I/O[1] operations. The output of the core is analysed for any errors which are immediately reported back to the PCI interface. If a read command was sent, the received I^{2}C data is sent to the Output Control.

The Output Control block merges the received I^{2}C data with the Ticket ID, I^{2}C address and register number information memorised and provided by the Input Control. These additional information bits are put into the output data to allow exact identification of the origin of a specific I^{2}C datum. The data is then parallelised and stored inside a 64 bit wide Output FIFO until the PCI interpreter is ready to pick up the data.

[1]Input/Output

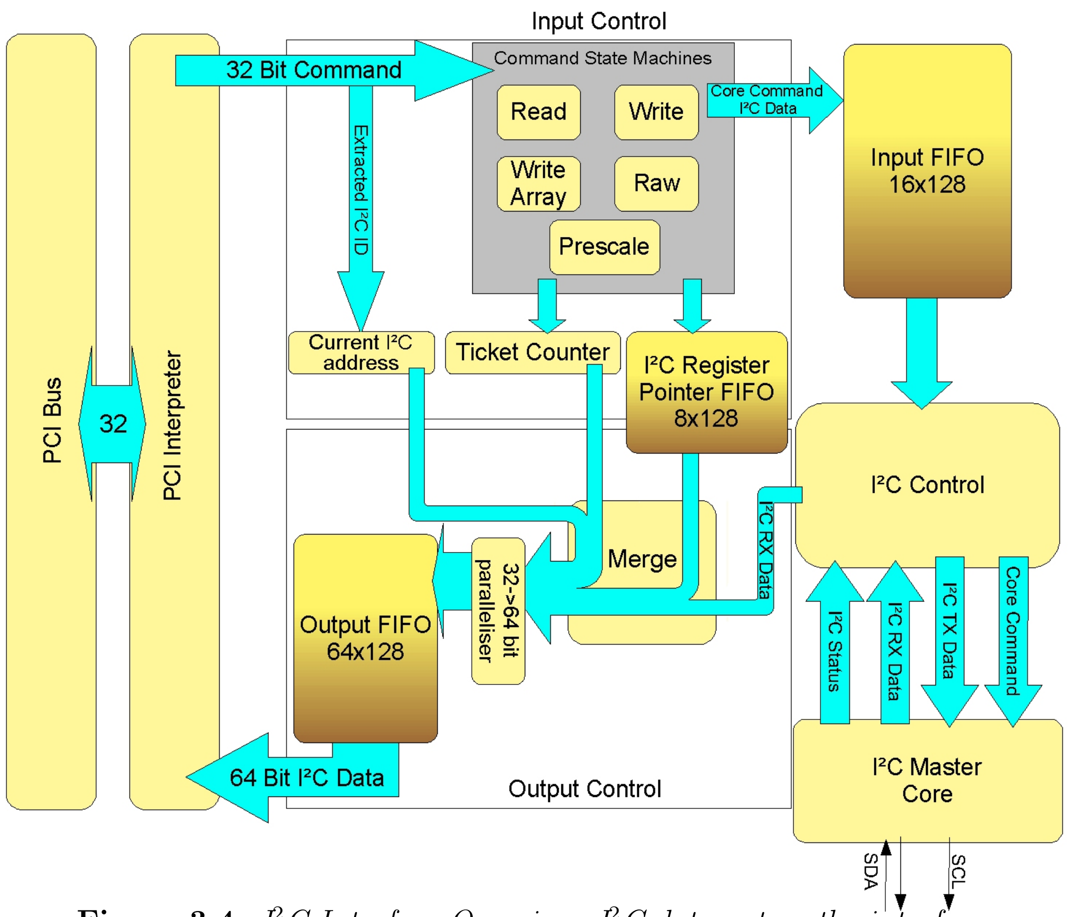


Figure 3.4: *I²C Interface Overview. I²C data enters the interface as 32 bit command. The command is translated into I²C-Master Core commands within the Input Control block. The core commands are stored inside the Input FIFO until the command is ready for execution. Execution is supervised by the I²C Control block. Received data is merged with their ticket, register and I²C address identifiers and stored inside the Output FIFO until they are collected by the PCI Interpreter*

3.3.4 I^{2}C-Master Core

The I^{2}C-Master [11] core is an open source package developed for a library of open source FPGA programs. All packages in this library use the *Wishbone* [12] interface as their interconnect architecture. A major decision factor for using this core was that it supported the necessary splitting of the SDA channel. The I^{2}C-Master core translates 8 bit parallel data words into serialised I^{2}C data and handles all the I^{2}C protocol specific tasks. Figure 3.5 shows an image of the component's interface. It is controlled by five internal registers (Table 3.1), that can be accesses by a single I/O port pair (wb_dat_i and wb_dat_o). A specific register can be selected by a third port (wb_addr_i).

I^{2}C transfers are performed by first writing a control command to the Command register. After that, the data to be transmitted can be written to the Transmit register. If the command was a read request, the received data can be read from the Receive Register.

Each received or transmitted 8 bit word has to be accompanied by a core command. Since the exact execution time of a particular command can not be determined, an interrupt port (wb_inta_o) signals the completion of a command.

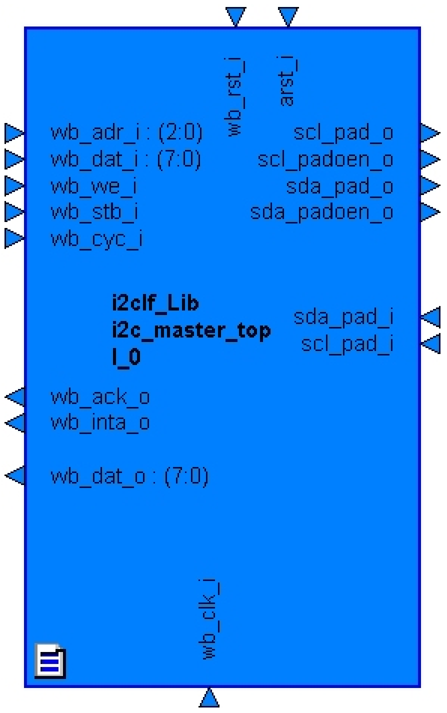


Figure 3.5: *I^{2}C-Master Core component.*

Name	**wb_addr_i**	**purpose**
PRERlo	0x00	Clock Scale low word
PRERhi	0x01	clock Scale high word
CTR	0x02	Activate/Deactivate Core
TXR/RXR	0x03	Transmit/Receive Register
CR/SR	0x04	Command/Status Register

Table 3.1: *I^{2}C Master Core registers [11]. The mode of the last two registers depends on read or write access*

3.3.5 Input Control

The first element in the communications chain from PCI Interpreter to I^{2}C bus is the Input Control subsystem. It is responsible for receiving data from the command infrastructure of the superior FPGA program.

This infrastructure was originally not designed to handle the transfer of large amounts of data. The infrastructure delivers data only in discrete 32 bit wide words and does not provide data streams. A data format had to be devised to allow I^{2}C write cycles that consist of more than the 32 bits for a single command.

The data format subdivides the 32 bit commands into 3 sections. The four Most Significant Bit (MSB) of the command are the magic pattern of the I^{2}C interface. This pattern can be configured externally and the interface will only respond to commands containing this pattern. The next four bits identify one of seven possible sub commands:

1. **Read**: reads up to 125 bytes of data from a specific I^{2}C address and register
2. **Write**: writes one byte to a specific I^{2}C address and register
3. **Write Array**: writes up to 125 bytes to a specific I^{2}C address and register
4. **Prescale**: sets the clock ratio between the FPGA program and I^{2}C SCL
5. **Raw**: allows direct control of the I^{2}C-Master core
6. **E**nables/Disables the output of I^{2}C data compatible to the LVDS transmission method
7. **E**nables/Disables an I^{2}C compliant tri-state output

The rest of the 32 bit command contains the arguments to these sub commands.

Each sub command has its own state machine that processes the sub command and its parameters. The output of these machines are 8 bit command/data pairs that are understood by the I^{2}C-Master core and are collected inside a 16 bit wide and 128 bit deep FIFO buffer until they are ready for execution.

For read cycles the Input Control extracts the target I^{2}C address from the incoming commands. It stores the address together with a ticket number and the list of register numbers that are going to be read from the target I^{2}C address. The ticket number is generated by a counter that is incremented for each I^{2}C transfer. This procedure is used to supply each received I^{2}C byte with exact information about its source.

3.3.6 I^{2}C Control Block

This block controls the data flow between Input and Output control and the I^{2}C-Master Core. As soon as the Input Control has finished processing the current command sequence, the state machine within this block starts processing the accumulated core commands. Figure 3.7 shows a flowchart diagram of this state machine.

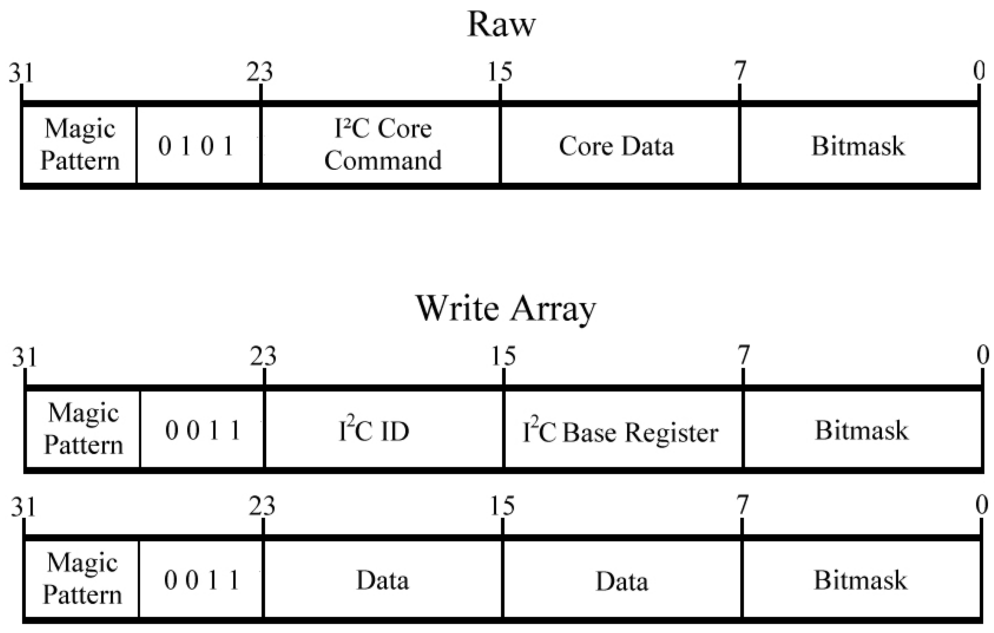


Figure 3.6: *Raw and Write Array command structures.*

After a core command and its data byte has been fetched from the Input FIFO it is inspected if it is a read or write command. It is then forwarded to the I^{2}C-Master. The machine waits for the interrupt signal from the master and verifies that the request was successful. If an error was encountered, the error status is written to the status register and the process is aborted. If the core command contained a read request, the received data is forwarded to the output control. As long as there are more entries available from the Input FIFO, the I^{2}C Control block inhibits the acceptance of new commands to the input control.

3.3.7 Output Control

This sub program receives the I^{2}C data from the Control Block. It processes and buffers it until it is ready for pick up by the PCI Interpreter. The transmission to the interpreter has to be done in 64 bit wide data words. It was decided to embed each received I^{2}C data byte into a 32 bit word, which contains additional information about its origin. This was done because it could not be assured, that data sent to the PCI Interpreter would appear in PCI memory in the same order. Two of these 32 bit blocks are combined into a 64 bit word. If there is an uneven amount of received I^{2}C bytes, only bits [31:0] of the last 64 bit word are used.

The I^{2}C Address, Ticket and Register number that is stored as additional information within the 32 bit structure are supplied by the Input Control. The Input Control collects and calculates these from the commands it receives and stores them until the current I^{2}C transfer is completed. Address and Ticket

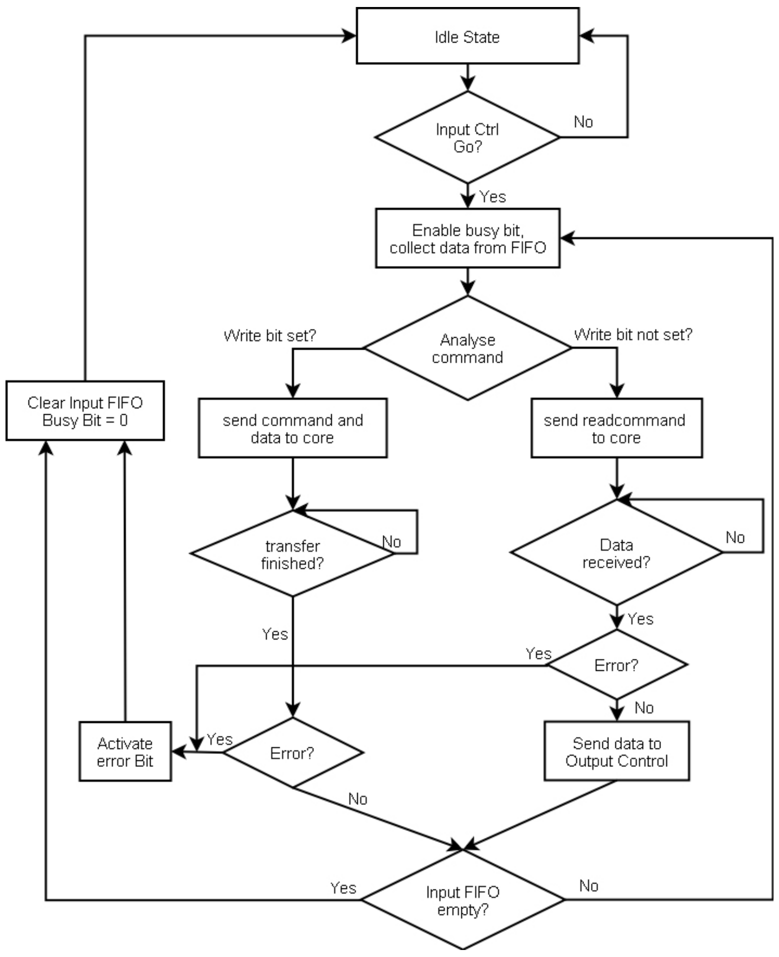


Figure 3.7: *I^{2}C Control flowchart*

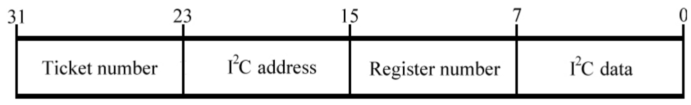


Figure 3.8: *Output Control data format. Each I^{2}C byte is tagged with its source I^{2}C address and register number together with a serial number to identify the transfer it belongs to.*

number are stored in two registers, while the Register numbers are stored inside the Register Pointer FIFO. A state machine merges the I^{2}C bytes with their I^{2}C Address, Ticket and Register number and forwards them to the paralleliser.

After a successful transfer, the Output Control sends an interrupt to the PCI Interpreter, signalling that new data is ready for pick up.

3.3.8 Implementation tools

The design of the FPGA program was done with the Integrated Development Environment (IDE) *HDL-Designer 2005.1* of Mentor Graphics. The core of this IDE is a program[2], which allows the creation of virtual circuit diagrams in an object modelling environment. An FPGA program is developed by placing and connecting logic elements on the diagram. Figure 3.9 shows the virtual circuit of the I^2C interface within the object modelling environment.

After the circuit is completed, the program is translated it into VHDL code. It can then be sent to a second program [3] within the IDE for simulation purposes. During simulation, the states of all logic elements are calculated for incremental time intervals. External signals can be applied to the inputs of the virtual circuit. The resulting states at each node in the circuit will be calculated and can be monitored.

After the circuit shows the expected behaviour, the VHDL code is compiled into a form that is optimised for the FPGA (Stratix-EP1S25) by an external application[4] and uploaded to the chip.

[2]FPGA-Advantage ™
[3]ModelSim SE 6.0d ™
[4]Quartus 5.0 ™

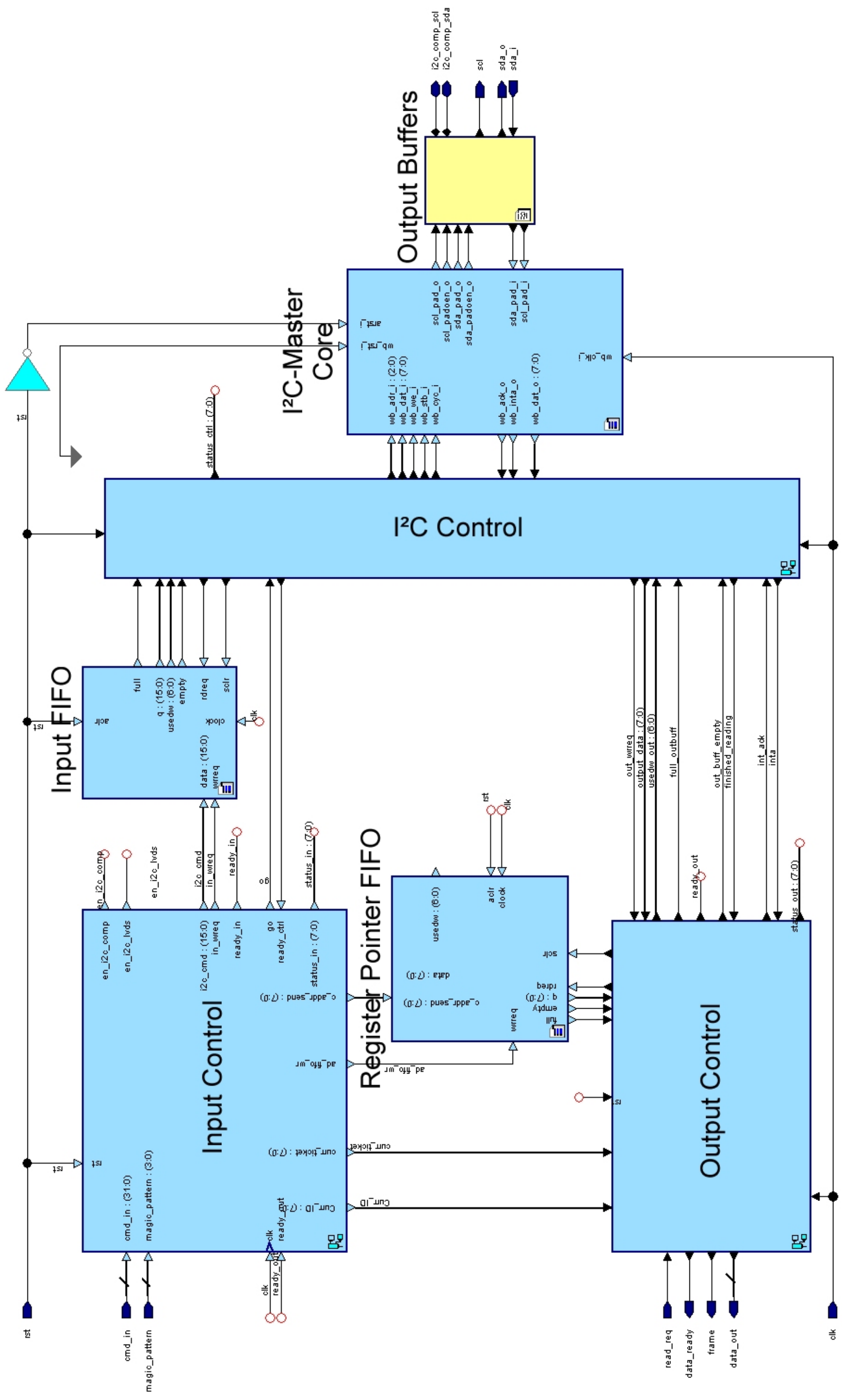


Figure 3.9: *Top level of the I²C Interface virtual circuit diagram. The program is subdivided into logical block elements that are interconnected with signal lines, carrying information between the blocks.*

3.3.9 Simulation and test

Simulation plays an important role during development of FPGA software. Figure 3.11 and 3.10 shows a comparison between a simulation and an oscilloscope recording of the simulated transfer. The example chosen is a transfer of three data words to the I^{2}C address 0x5C. The simulation and the measurement are in perfect agreement.

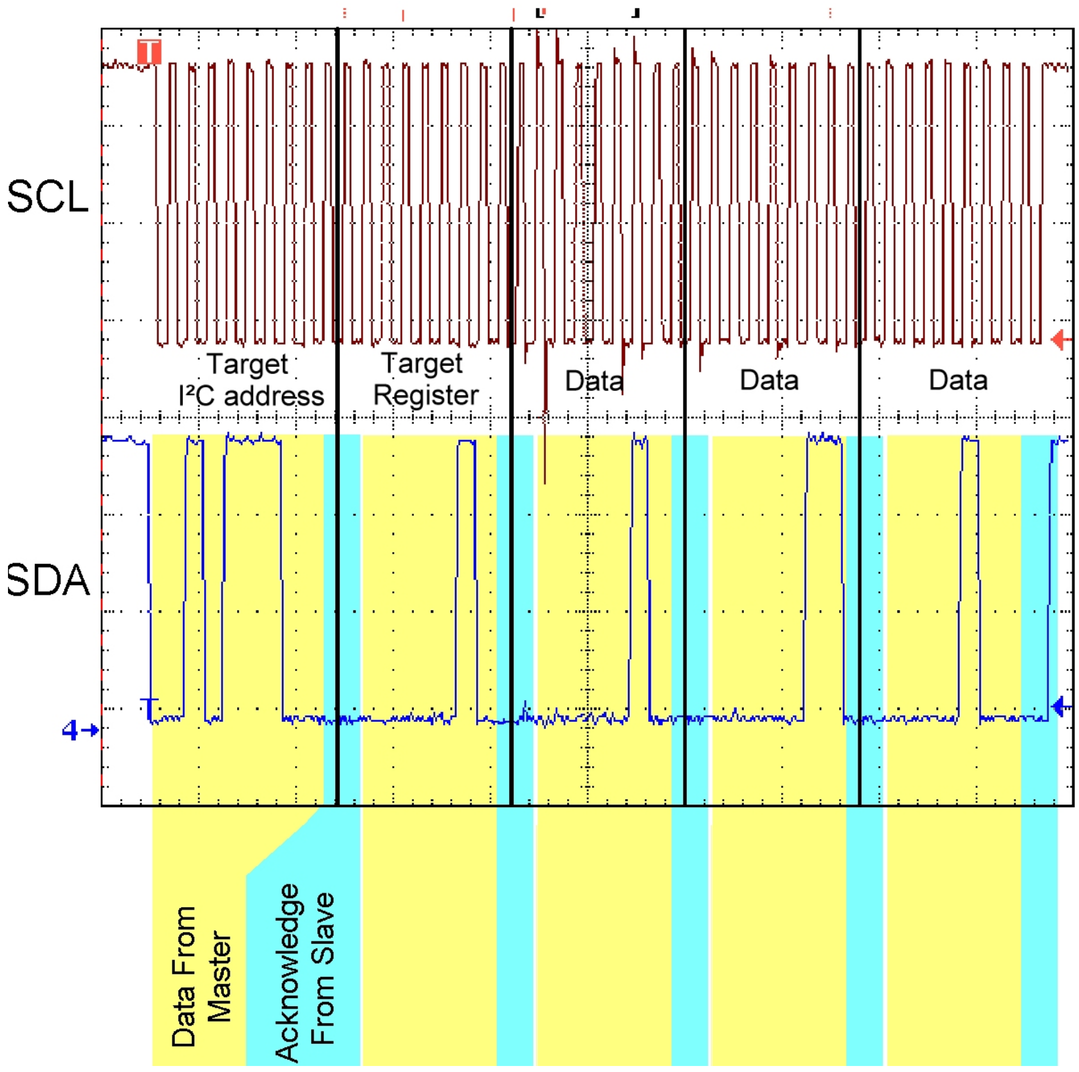


Figure 3.10: *Oscilloscope recording of the Write Array example from figure 3.11. The small pauses between the individual write requests are not visible here, because the simulation was done at a higher SCL speed. The upper part shows the clock signal generated by the master. The lower shows the data signal.*

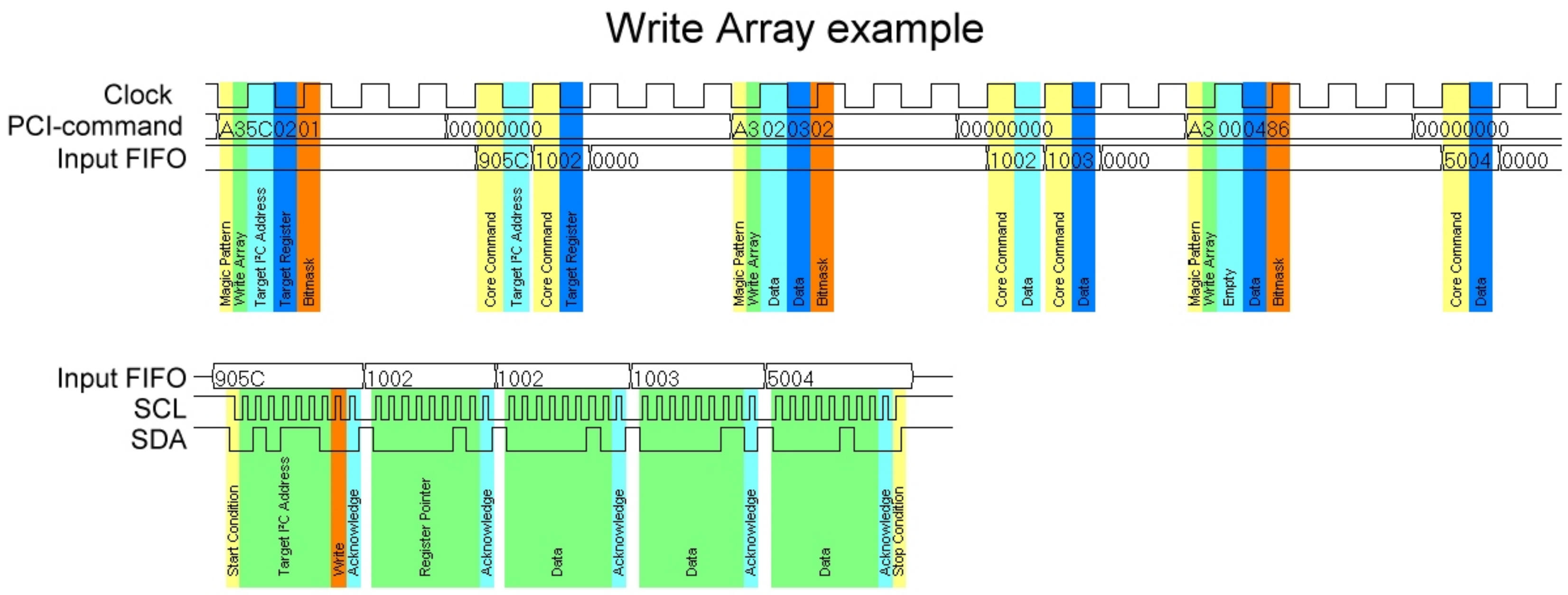


Figure 3.11: *Simulation of a Write Array Command to an OTIS Chip. The upper part shows the transaction from the PCI-command bus to the I²C interface. Each command received is translated into I²C core commands and added to the input FIFO.*
The lower part shows the I²C bit stream created by the master core. After the target address and the register pointer are set, three data words are transmitted. Each 8 bit word sent by the master is acknowledged by the slave with an acknowledge bit.

3.3.10 Interface software

To access the FPGA-I^{2}C interface with the test software, the old FPGA driver had to be enhanced with an I^{2}C interface module.

The driver was implemented in C++ and consists of a general I^{2}C interface class. A GOL and OTIS class was added to hide the I^{2}C implementation and allow access to the registers of the chips without any knowledge about I^{2}C. Figure 3.12 shows an UML diagram of the classes and their relation.

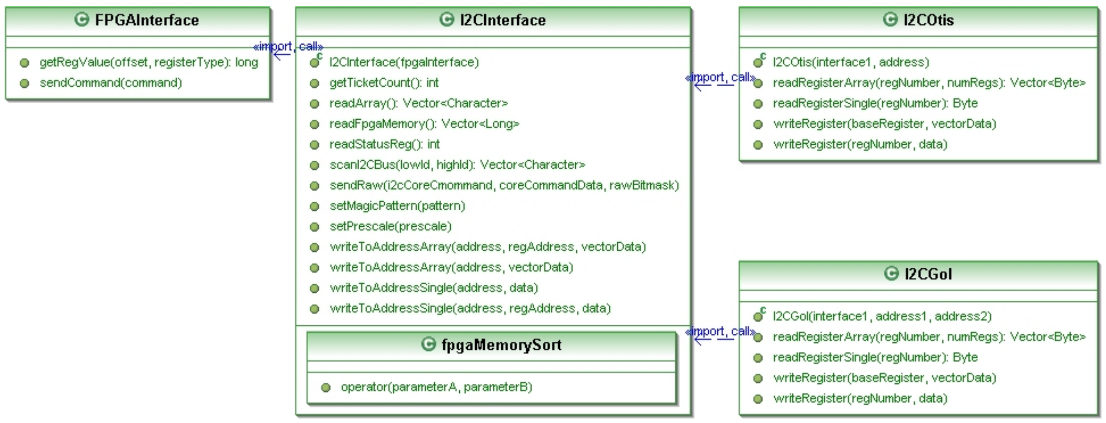

FPGAInterface
getRegValue(offset, registerType): long
sendCommand(command)
«import, call»
I2CInterface
I2CInterface(fpgaInterface)
getTicketCount(): int
readArray(): Vector<Character>
readFpgaMemory(): Vector<Long>
readStatusReg(): int
scanI2CBus(lowId, highId): Vector<Character>
sendRaw(i2cCoreCmommand, coreCommandData, rawBitmask)
setMagicPattern(pattern)
setPrescale(prescale)
writeToAddressArray(address, regAddress, vectorData)
writeToAddressArray(address, vectorData)
writeToAddressSingle(address, data)
writeToAddressSingle(address, regAddress, data)
fpgaMemorySort
operator(parameterA, parameterB)
«import, call»
I2COtis
I2COtis(interface1, address)
readRegisterArray(regNumber, numRegs): Vector<Byte>
readRegisterSingle(regNumber): Byte
writeRegister(baseRegister, vectorData)
writeRegister(regNumber, data)
«import, call»
I2CGol
I2CGol(interface1, address1, address2)
readRegisterArray(regNumber, numRegs): Vector<Byte>
readRegisterSingle(regNumber): Byte
writeRegister(baseRegister, vectorData)
writeRegister(regNumber, data)

Figure 3.12: *UML Diagram of the I^{2}C C++ Class library. The FPGAInterface class has more methods which have been skipped here for clarity. The fpgaMemorySort class is a helper class that allows sorting of the 32 bit data structure shown in figure 3.8 by register pointer number.*

The *I2CInterface* class connects to the FPGA through the already available *FPGAInterface* class. It writes and reads data to and from the FPGA by invoking the *sendCommand()* and *getRegValue()* methods.

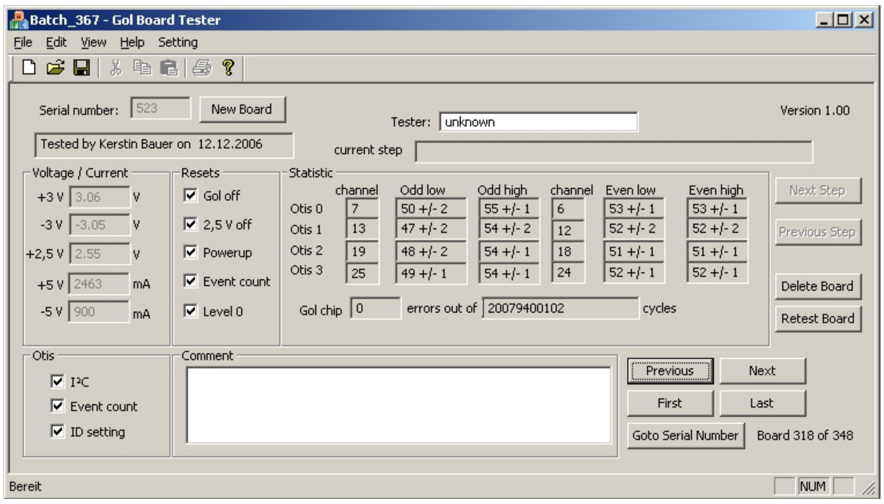
The *I2CInterface* decomposes I^{2}C read and write requests into 32 bit commands and sends them to the I^{2}C interface on the FPGA. Received data is sorted by I^{2}C register addresses and filtered by ticket number and requested I^{2}C address.

3.4 GOL/AUX test using the I^{2}C interface

The GOL/AUX board test was performed during the fall of 2006. It helped identify several boards with manufacturing errors. 87% boards passed the test so far and another 8% are expected to pass after minor fixes.

It took three months to test 466 boards. The test of one board took approximately 15 minutes.

Figure 3.13 shows a screen shot of the program's Graphical User Interface (GUI). The program performed all necessary tests with minimal user interaction.



Serial number: 523

Tester: unknown

Version 1.00

Tested by Kerstin Bauer on 12.12.2006

current step

Voltage / Current

+3 V	3.06	V
-3 V	-3.05	V
+2,5 V	2.55	V
+5 V	2463	mA
-5 V	900	mA

Resets

- ✓ Gol off
- ✓ 2,5 V off
- ✓ Powerup
- ✓ Event count
- ✓ Level 0

Statistic

	channel	Odd low	Odd high	channel	Even low	Even high
Otis 0	7	50 +/- 2	55 +/- 1	6	53 +/- 1	53 +/- 1
Otis 1	13	47 +/- 2	54 +/- 2	12	52 +/- 2	52 +/- 2
Otis 2	19	48 +/- 2	54 +/- 1	18	51 +/- 1	51 +/- 1
Otis 3	25	49 +/- 1	54 +/- 1	24	52 +/- 1	52 +/- 1

Gol chip 0 errors out of 20079400102 cycles

Otis

- ✓ I²C
- ✓ Event count
- ✓ ID setting

Comment

Board 318 of 348

Figure 3.13: *User interface of the GOL/AUX test program*

3.5 Determining the influence of I^{2}C traffic

The question arose if the process of reading the I^{2}C-registers of the OTIS chips would cause an unacceptable amount of noise. As a consequence, crosstalk between the I^{2}C bus and the OTIS chip could lead to an increase in occupancy.

There are two possibilities of how I^{2}C crosstalk noise can enter the drift time measurement system.

1. The drift chambers can pick up the I^{2}C signal from the service cables running from the Distribution Box to the Front-end Boxes.

2. The I^{2}C signal is picked up directly by the electronics components in the Front-end Box from the signal traces on the boards or from the inside of the components themselves.

The first mechanism is very unlikely. The interference would not only have to overcome the shielding of the SCSI cables and the shielding created by the grounded aluminium C-Frames, in which the cables are held. The signal is also sent as differential signal through twisted pair cables, which already reduces stray radiation by a large amount.

The second scenario is more likely and a test was performed to measure the actual noise contribution of I^{2}C accesses to the Front-end Boxes during data acquisition. The setup for this test consisted of:

- 6 Front-end Boxes
- 1 Distribution Box
- 1 TELL1-Board

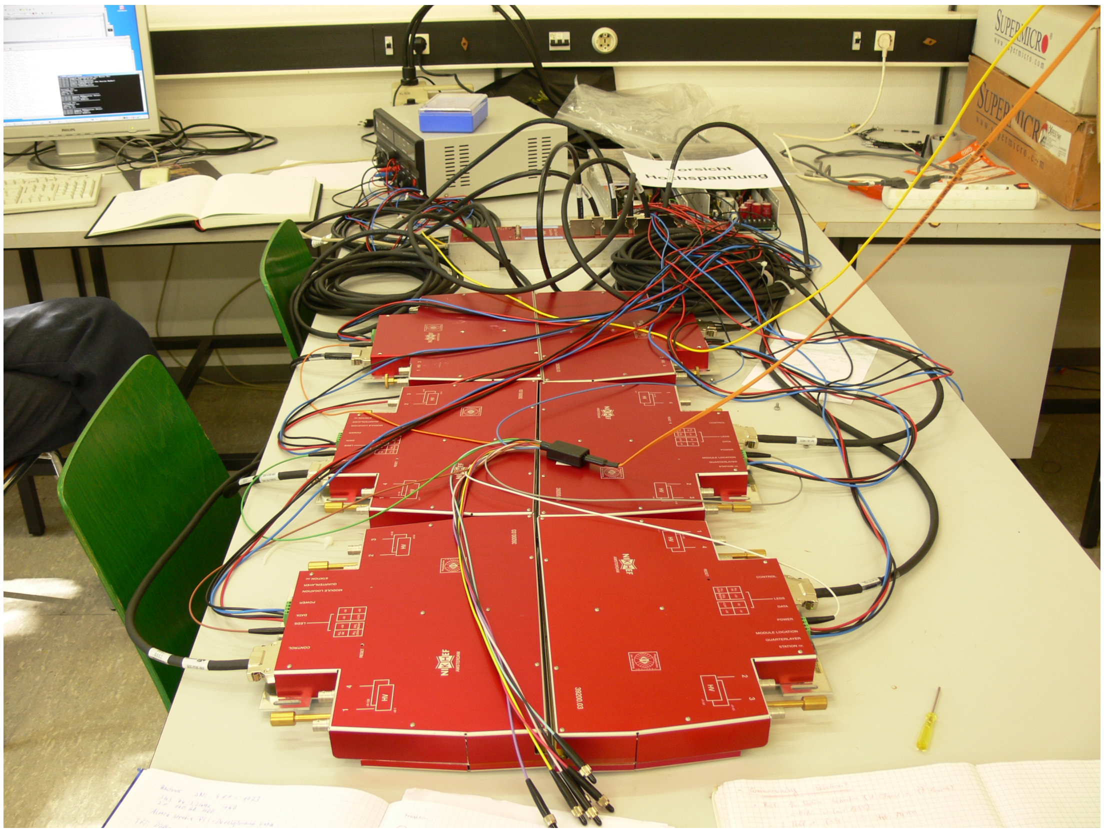


Figure 3.14: *The lab setup in Dortmund consisting of two prototype Front-end Boxes, four final boxes and one Distribution Box. Only the four final boxes were taken into consideration for the noise scan. The prototypes were only used as additional load for the PVSS control system test*

- 1 ODIN-Board

The Front-end Boxes were controlled, using the PVSS system introduced in chapter 4. A similar program was used as control for the ODIN Readout Supervisor [2] which was provided by the LHCb online group. Data acquisition was done on a Linux farm node PC.

This was also the first setup containing all the final hardware components of the Outer Tracker readout and control chain.

Figure 3.14 shows a picture of the laboratory setup in Dortmund which was used for this test. There were no wire chambers connected to the boxes. In fact, connecting chambers to the boxes might have increased the noise unrealistically because the control cables were not shielded by a C-Frame.

The test was performed by repeating the following steps for different ASDBLR threshold voltages between 500 and 900 mV in 50 mV intervals:

1. All ASDBLR boards are programmed with a common threshold value.

2. 10.000 random events are recorded for the threshold value set in step 1.

3. The number of “hits” for each OTIS chip is determined.

The test was performed twice; once with I^{2}C access to the chips and once without.

Since the Slow Control system only allows a minimal polling interval of 1 second, it had to be modified to run in an endless loop. This modification made it possible to read from all subscribed devices in a continuous mode without any pauses between the I^{2}C accesses.

The I^{2}C speed was set to 128 kbit/s. The random trigger rate was approximately 110 Hz.

The results for the four Front-end Boxes can be seen in figure 3.15. In this plot the number of hits on all 32 channels of the four OTIS-Chips were summed up and plotted against the programmed threshold voltage.

While there is a clear difference visible at lower voltages, it is diminishing at higher levels and vanishes completely at the foreseen working point of 750 mV. The increase in occupancy at lower thresholds is approximately a factor of four.

For the final experiment the Front-end Boxes will be polled in intervals of minutes rather than milliseconds. This will greatly reduce the observed I^{2}C crosstalk effect.

In conclusion, I^{2}C accesses can crosstalk to the drift time measurement system, but the effect is completely gone at the foreseen working point.

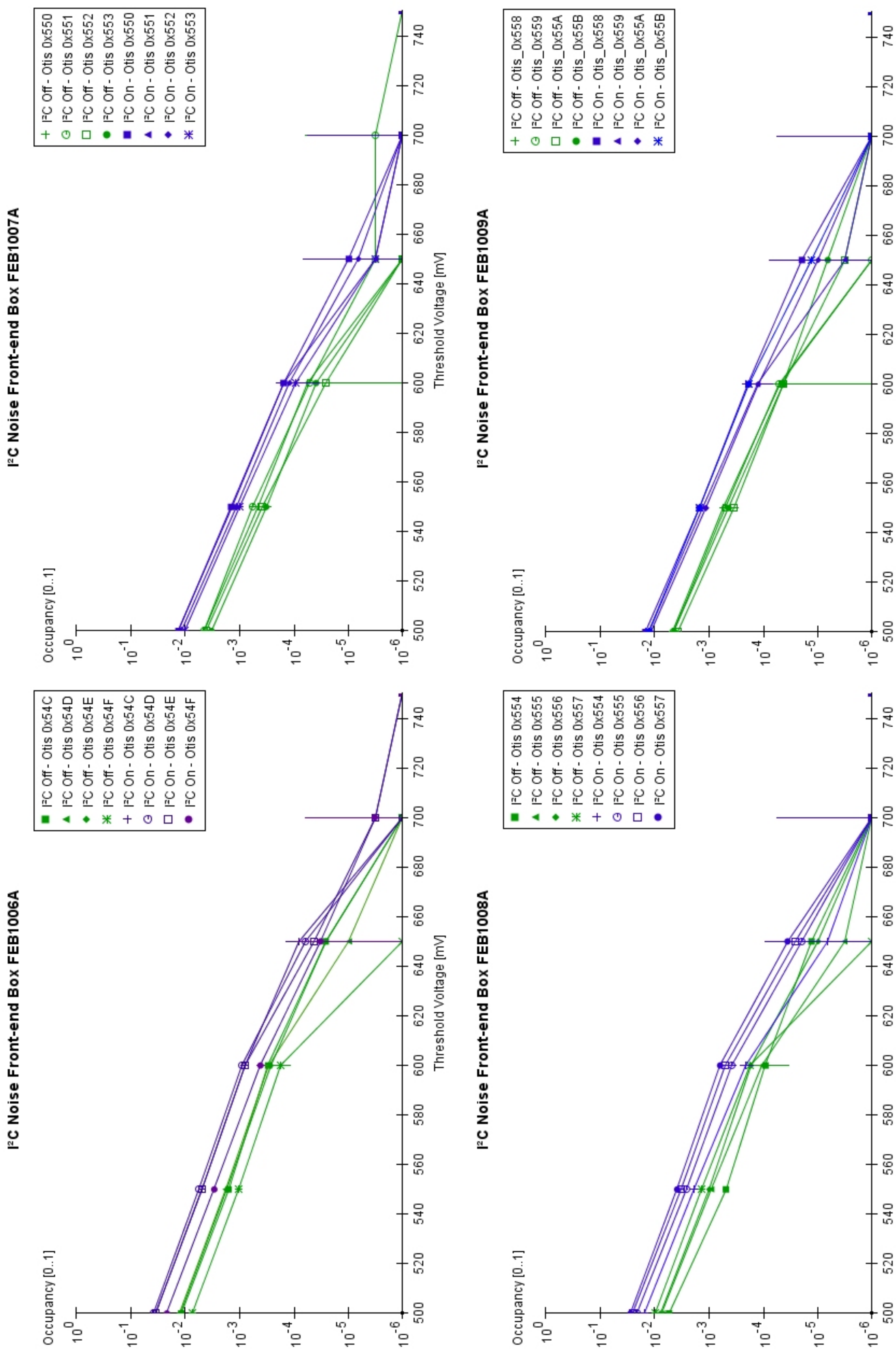


Figure 3.15: *Comparison of the noise levels of the four Front-end Boxes with and without I^2C.*
Blue: with I^2C access
Green: without I^2C

Chapter 4

First implementation of a Slow Control system for the LHCb Outer Tracker

The hardware and systems for each of the LHC experiments are highly specialised and individual. It was decided to allow each experiment, and subsequently their sub-projects, to design their own control systems. It was decided to use a commercial Supervisory Control and Data Acquisition (SCADA) product for the development of these systems. CERN performed a survey of available products and selected PVSS as a common base for all experiments. A general introduction to PVSS and its philosophy is given in appendix A.

In addition, certain guidelines and tools have been developed by CERN and LHCb to ensure interoperability between these individual developments.

4.0.1 JCOP framework

The Joint COntrols Project (JCOP) framework is the main set of rules which all developers of sub detector control systems have to regard. It also provides a collection of standardised PVSS panels and programs that can be used to solve common implementation problems. After all sub systems are integrated into the framework, it will form the control hierarchy of the experiment. [17] The hierarchy is designed as a tree structure. The root of the tree is the highest level control instance while the leaves are the devices that are being controlled. A cut-out of the LHCb tree can be seen in figure 4.1.

4.0.2 Finite State Machine tool

The Finite State Machine is a tool within the JCOP framework that is used to model and operate this tree. The upper part of a hierarchy is defined by JCOP. The sub detectors are responsible for implementing the leaves of this super tree.

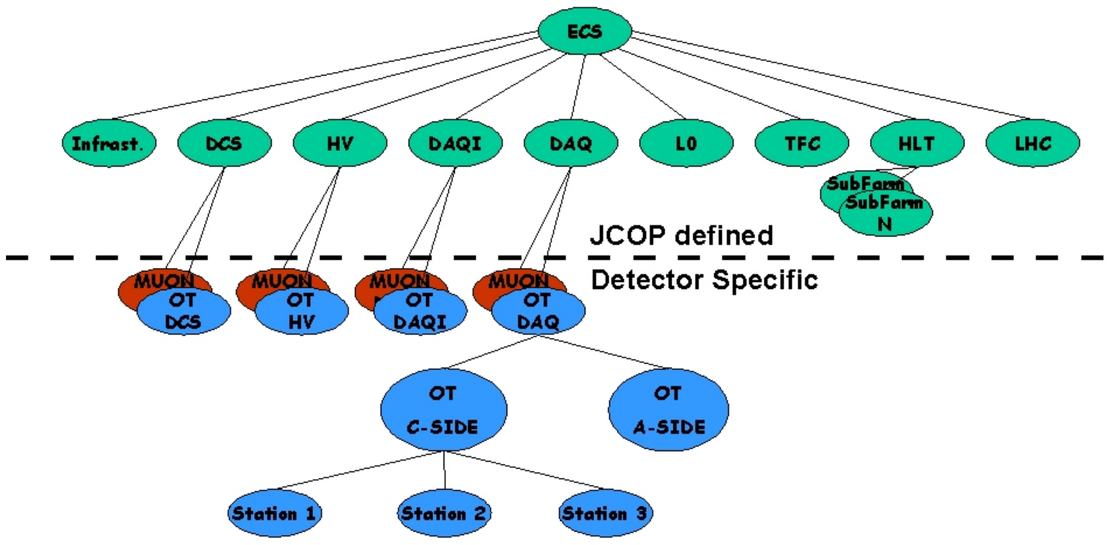


Figure 4.1: *The LHCb state machine hierarchy. The green nodes are defined by JCOP. Each detector is responsible to build its own hierarchy at the leaves of this hierarchy.*

The nodes within the tree are FSM objects. There are three kinds of objects:

Device Units (DU): Device Units are the leaves of the tree and each Device Unit is associated with a device or a group of devices that are combined to a logical device.

Logical Units (LU): Logical Units are nodes within the tree that are used to group Device Units or other Logical Units into logical groups.

Control Units (CU): If a Device Unit or Logical Unit and its sub tree can be separated from the main hierarchy and controlled on its own it is called a Control Unit.

Each FSM object is a small state machine with configurable states. The states are calculated from the states of either the children, if it is a Logical Unit, or the associated device, if it is a Device Unit. If a state transition takes place in one of the nodes, the transition is reported to the parent node which will recalculate its state depending on the new child state.

The FSM states are defined rather coarse. A device is merely described in terms of working, not working or error states rather than a detailed analysis of what it is exactly doing. This simplification is done because the detector operators won't be detector experts and only need to know if something is working or not. It is the duty of a detector expert to find and fix errors that can not be resolved by the state machines logic.

The JCOP framework defines a number of standard states for the higher levels of the detector hierarchy. They are shown in figure 4.2. The sub detectors are responsible to translate the states of their own trees to the standard states at the transition node from sub detector to the superior hierarchy. The translation is done within a Logical Unit.

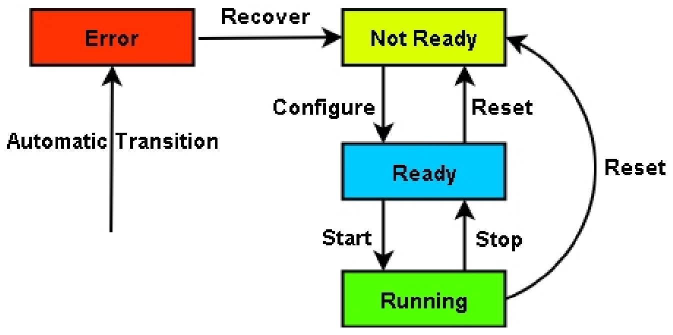


Figure 4.2: *Standard FSM states*

FSM objects also have to accept commands which can be processed by the node and are, by default, also passed on to their children. Commands are kept simple too and are only used to signal starts, stops or configuration requests. It is the duty of the FSM nodes of the sub detectors to know what to do if they receive a start, stop, etc. command.

JCOP defines a number of standard commands that all sub detector hierarchies have to be able to understand. They initiate the transitions between states seen in figure 4.2. If a transition really occurs depends on the outcome of the command forwarded to the child nodes and ultimately by the response of the hardware to that command.

All FSM objects are described by a special programming language and are running in an interpreter program outside of PVSS. A state machine manager is used to connect PVSS to the FSM program. The FSM objects are exported as virtual devices to PVSS, which acts as the control system for these objects.

4.0.3 Distributed Information Management System

Distributed Information Management System (DIM) is a an inter process messaging protocol. It is used as the primary interface between PVSS and the drivers for the detector hardware. It can be regarded as the communications link between the data servers and the hardware drivers in the generic SCADA model. It is also used as the interface between PVSS, the FSM and other software programs.

DIM, like most communications systems, is based on the client/server paradigm [18]. A DIM server offers named data services, which can be accessed via TCP/IP. A DIM Name Server (DNS) is used to administer the names and server addresses of these services.

A DIM client can access the name server and browse the list of available services. It can subscribe to services on a server and will be notified by the server when the service is updated with new values. A client can also send messages (commands) to motivate the server to perform some action.

Figure 4.3 shows how servers, clients and name server interact.

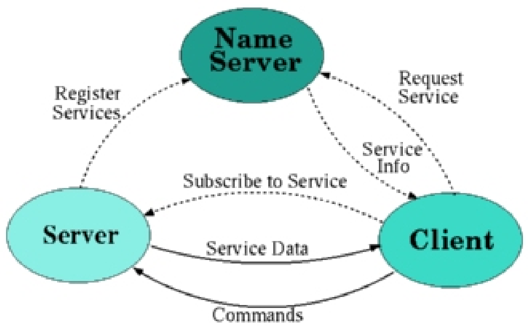


Figure 4.3: *DIM messaging design.*

The DIM protocol is available for C, C++ and other programming languages and has been used to implement communication between PVSS and the commissioning tools for the Mini DAQ system introduced in chapter 5.

A DIM-PVSS manager allows the direct mapping of DIM services to PVSS data points. If a DIM service is updated, the corresponding PVSS Data Point will be updated. If the Data Point is changed by PVSS, it will be forwarded to the DIM server as DIM command.

4.0.4 Configuration Database (Config DB)

The Configuration Database is a relational database within the JCOP framework tool used for the configuration of hardware and software devices. It is supposed to save static as well as dynamic configuration data. Static data contains everything that is necessary to rebuild the FSM hardware tree and to map PVSS Data Points to their respective devices. Dynamic data contains the information necessary to configure the hardware itself.

Dynamic device configuration information is saved to the database as so called recipes. A recipe contains the information necessary to configure all devices within the recipe for a certain task. For example, the recipe "Cosmics" would contain the configuration data for a data acquisition run with cosmic particles.

The Configuration DB keeps data in two locations. Dynamic and Static configuration data is saved in an external Oracle^{TM} database. This database is used as backup and to keep track of different versions of configuration properties.

The second storage location is a cache implemented in PVSS itself. The cache only hosts dynamic configuration data and is the preferred data source because it can be accessed much faster than the Oracle^{TM} database.

At the time of this thesis only the configuration cache was available for testing.

4.1 SPECS

The Serial Protocol for the Experiment Control System (SPECS) [5] is a serial, master-slave bus protocol similar to I^{2}C but without acknowledgements for better throughput. Like the LVDS I^{2}C protocol introduced in chapter 3 it is split up into receiving and transmitting clock and data lines.

SPECS data is transmitted as short messages or frames. Each frame has strong redundancy mechanisms to detect data corruption. This is necessary because the hardware components used for the SPECS transmission will be exposed to radiation.

Its main purpose is the picka-back transport of other protocols from the control system to the detector hardware. This implementation was chosen because the detector consists of many components of different origin. These components use different hardware communication systems like I^{2}C, JTAG, Parallel Bus, etc. Moreover, there are several different flavours of implementation for certain protocols. The SPECS hardware transmits all these different communication types to the detector in a unified way. At the detector a rad hard *mezzanine* board decodes the SPECS messages and transmits them to the detector components via their respective communication standards.

The hardware components of SPECS consist of a *master* card which is a PCI card connected to an ordinary PC. The *mezzanine* is the slave and is connected to the master via twisted pair CAT 5 network cables and to the detector components via individual, sub detector specific connections. The typical distance between SPECS master and slave will be 70 m in the experiment.

4.2 Control scheme of the Outer Tracker

The first implementation of a control system for the Outer Tracker was one of the main goals of this thesis. This assignment contained the following tasks:

- Determining a control hierarchy with sensible partitions and implementing them with the FSM tool.
- Defining state machine objects and incorporating them into the FSM hierarchy.
- Creating graphical user interfaces and control routines for the utilised hardware
- Tests and benchmarks for increasing system sizes.

The Outer Tracker Control Scheme is using PVSS as Slow Control and SPECS as the communications link between the hardware and the control system. The detector is subdivided into two physically distinct halves on both sides of the beam pipe. Each of these halves is controlled by an individual PVSS system consisting of twelve Distribution Boxes or 1100 hardware items of GOL, OTIS and Distribution Box devices. The two systems can be operated from a higher level control instance in the hierarchy. Since these two systems are independent of each other and running in parallel it is enough to develop a control system for one half of the detector. The other half can be controlled by a copy.

4.3 FSM Hierarchy and Objects

4.3.1 Control Hierarchy

The Outer Tracker addressing scheme divides the sub detector into Stations, Layers and Quadrants, as seen in figure 1.3. This division was adapted almost completely as control hierarchy, which is shown in figure 4.4.

The hierarchy contains Command Units for the two sides of the detector, for single Tracking Stations and Distribution Boxes. The Distribution Box level replaces the Layer level of the physical partitioning. This had to be done because one Distribution Box controls two layers, one on the front and one on the back side of a C-Frame, and certain control functions of the box affect both layers at once. The Distribution Box level was also chosen as the last Control Unit level because of this reason.

The further partitioning contains the two Quadrants on the front and back side of a C-Frame that is associated with a Distribution Box and at the bottom the Front-end Boxes and the front-end electronics devices themselves.

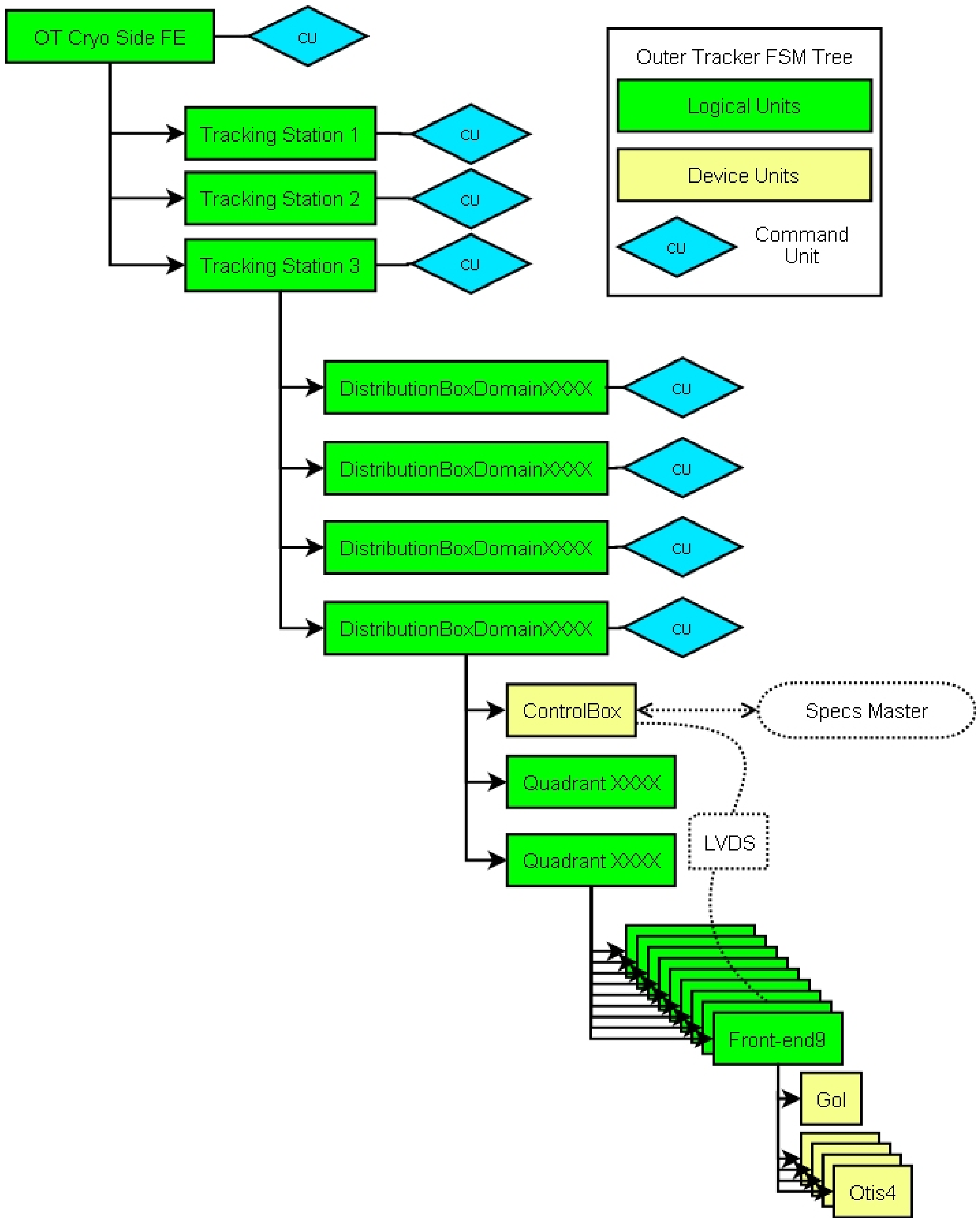


Figure 4.4: *Outer Tracker FSM hierarchy. The hierarchy mirrors the hardware hierarchy. An exception are the DistributionBox-Domains. They are inserted because certain commands affect all front-end boxes connected to one distribution box.*
Each command Unit can be separated from the tree and controlled on its own. The Distribution Box level is the minimum Control Unit because of the aforementioned shared commands.
Only one instance of the sub tree is expanded on each level for better clarity.

4.3.2 Control FSM Objects

As explained earlier, each node within the FSM tree is a state machine object that represents a summary of the state of its children and defines commands that are used to control it. A state logic and commands had to be defined for the different device types. The states should reflect the general condition of a device rather than a detailed description. They are only required to give an idea if the device is operational or not.

States

Figure 4.5 shows a diagram of the state machine logic implemented for all chips. It has been decided to use the same states for all types of chips, because they all behave the same from a working/not working point of view.

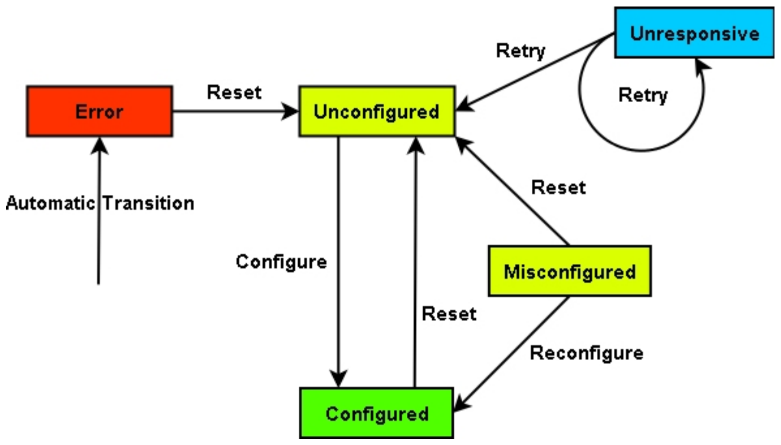


Figure 4.5: *Outer Tracker front-end electronics FSM states*

The different states perform the following tasks:

Unconfigured: This is the default state, after a reset has occurred or the devices have been powered up. It signals, that the device has been found on the I^2C bus and that it is responsive to I^2C read/write requests. The configuration registers are in an unknown state.

Configured: This state signals that the device is configured and that data acquisition can take place. This state corresponds to the *Running* state of the global control hierarchy.

Misconfigured: If a device has been programmed with a certain configuration, but the actual configuration on the chip is different, it will migrate into this state. The user has the option to issue a *Reconfigure* command. If the reconfigure fails too, the device will go into error.

Unresponsive: The device could not be reached via the I^2C bus. I.e. it is not responding to I^2C requests. This state is explicitly not an error because errors should be recoverable by a *Recover* or *Reset* command. If the device is not connected to the system it can not be reset.

Error: The device is responsive, but an error has been detected. This state depends on the chips status register or if it was repeatedly misconfigured.

The state of a device is saved as a single integer variable inside the device's PVSS data point. The states correspond to different values of this variable. This was mainly done to allow easier debugging of problems. The state variable is recalculated every time the SPECS server's monitoring algorithm updates the properties of a device in PVSS.

Commands

The device state machines support the following commands:

Configure: This command is used to initialise all devices into a certain detector run mode. The argument of the command is a string denoting the configuration name that should be used. When the command is issued, each Device Unit fetches the recipe for the particular run mode from the configuration database. The devices are then programmed with the configuration data.

After the configuration has been written, each device is read again to ensure that the configuration has been accepted. If the configuration is correct, the device will migrate into the *Configured* state.

Reset: This command is not directly supported by the OTIS and GOL chips. A reset signal can only be sent by the Distribution Box and will affect all 18 connected Front-end Boxes and the Distribution Box itself. The command is only accepted by the Distribution Box Device Unit. It performs a power up reset and sets the states of all connected Front-end Boxes and the Distribution Box to *Unconfigured*.

Recover: The *Recover* command is used if an error is detected. It should try to perform an automated recovery of the error condition, i.e. acknowledge and reset error status bits etc. However, most of these operations require a L0 reset signal which can not be issued by the Outer Tracker system itself but has to be issued by the TFC system. Until this issue is resolved, the *Recover* command will behave like the *Reset* command.

Reconfigure: This command can be issued to a device that is in a *Misconfigured* state. It will try to program the device a second time. If the command is unsuccessful, the device will go into *Error*.

4.4 Graphical User Interfaces

Graphical User Interfaces (GUI) are the interaction point between user and hardware. The design philosophy behind the GUI created for the Outer Tracker control was to have PVSS panels on every level of the FSM hierarchy that would allow detailed control at the lowest level of the tree and coarse control at the upper levels. In return, the upper levels of the hierarchy would be able to control many devices at the same time.

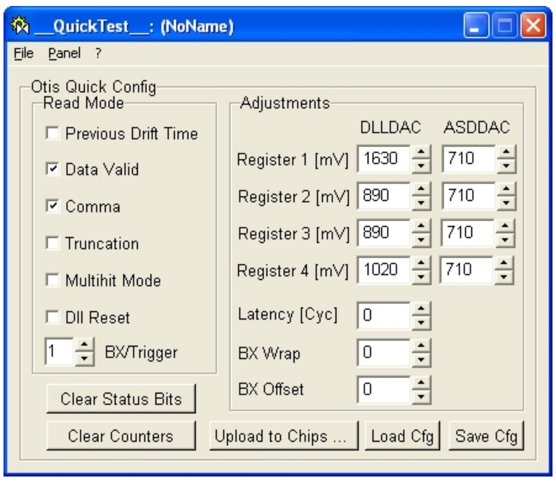
Another important aspect taken into account was that certain settings should not be immediately uploaded to the hardware as soon as a change is detected on the panel's widgets. I.e. certain panels can affect a large number of devices simultaneously. To prevent accidental misconfiguration, the user always has to press an *Upload* button before those changes become active.

To be more flexible in panel design, all panels have been broken down into sub panels which fulfil a certain, specific purpose and can be included on other panels. This modularisation allows the quick development of new panels that can control the Outer Tracker from different perspectives.

Panels are categorised into different kinds:

- **Single Target Panels:** These panels are designed to affect only one device type, but allow control of this one device in great detail.
- **Multi Target Panels:** These panels can control a whole array of devices of the same type, but allow control of common functions only. These panels require a list of compatible devices during their instantiation. All changes on the panel are then propagated to the whole device list.
- **Composite Panels:** Composite panels are built up from single and multi target panels. They are used to control collections of devices that belong to a logical unit like a C-Frame or everything connected to a Distribution Box.

An example of a single target panel for the OTIS chip is given in figure 4.7. Figure 4.6 shows the multi target version. Figure 4.8 finally shows a composite panel for one front-end box consisting of the multi target OTIS config, OTIS status panels and GOL panels.



Otis Quick Config

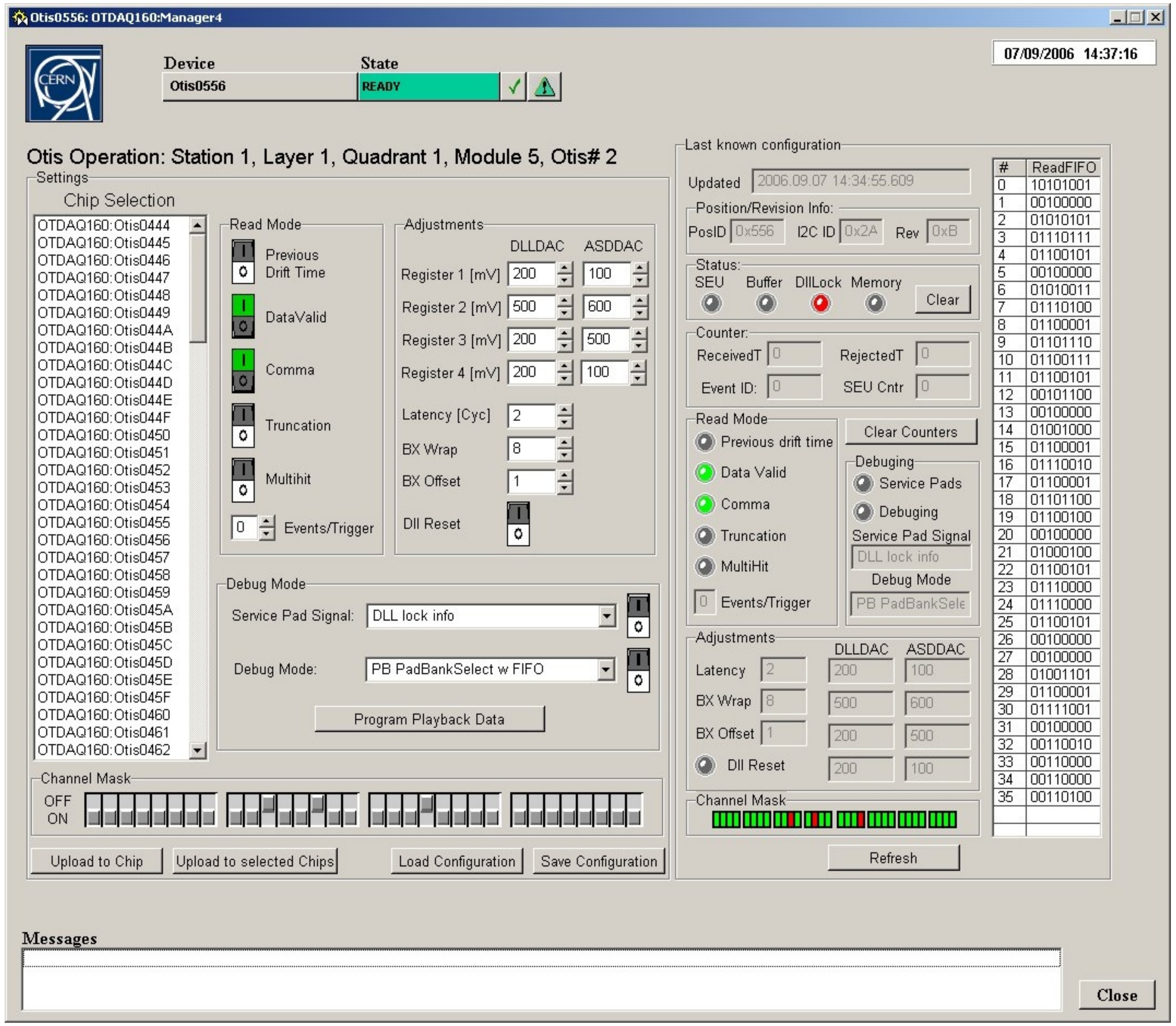
Read Mode

☐ Previous Drift Time
☑ Data Valid
☑ Comma
☐ Truncation
☐ Multihit Mode
☐ Dll Reset
1 BX/Trigger

Adjustments

	DLLDAC	ASDDAC
Register 1 [mV]	1630	710
Register 2 [mV]	890	710
Register 3 [mV]	890	710
Register 4 [mV]	1020	710
Latency [Cyc]	0	
BX Wrap	0	
BX Offset	0	

Figure 4.6: *Example of a multi target panel. When instantiating a multi target panel a list of compatible devices can be passed to it. All settings on that panel will then be applied to all devices on that list.*



Otis0556: OTDAQ160:Manager4

Device: Otis0556
State: READY
07/09/2006 14:37:16

Otis Operation: Station 1, Layer 1, Quadrant 1, Module 5, Otis# 2

Settings

Chip Selection

OTDAQ160:Otis0444
OTDAQ160:Otis0445
OTDAQ160:Otis0446
OTDAQ160:Otis0447
OTDAQ160:Otis0448
OTDAQ160:Otis0449
OTDAQ160:Otis044A
OTDAQ160:Otis044B
OTDAQ160:Otis044C
OTDAQ160:Otis044D
OTDAQ160:Otis044E
OTDAQ160:Otis044F
OTDAQ160:Otis0450
OTDAQ160:Otis0451
OTDAQ160:Otis0452
OTDAQ160:Otis0453
OTDAQ160:Otis0454
OTDAQ160:Otis0455
OTDAQ160:Otis0456
OTDAQ160:Otis0457
OTDAQ160:Otis0458
OTDAQ160:Otis0459
OTDAQ160:Otis045A
OTDAQ160:Otis045B
OTDAQ160:Otis045C
OTDAQ160:Otis045D
OTDAQ160:Otis045E
OTDAQ160:Otis045F
OTDAQ160:Otis0460
OTDAQ160:Otis0461
OTDAQ160:Otis0462

Read Mode
- Previous Drift Time
- DataValid
- Comma
- Truncation
- Multihit
- 0 Events/Trigger

Adjustments

	DLLDAC	ASDDAC
Register 1 [mV]	200	100
Register 2 [mV]	500	600
Register 3 [mV]	200	500
Register 4 [mV]	200	100

Latency [Cyc]: 2
BX Wrap: 8
BX Offset: 1
Dll Reset

Debug Mode
Service Pad Signal: DLL lock info
Debug Mode: PB PadBankSelect w FIFO
Program Playback Data

Channel Mask
OFF
ON

Upload to Chip | Upload to selected Chips | Load Configuration | Save Configuration

Last known configuration
Updated: 2006.09.07 14:34:55.609
Position/Revision Info:
PosID: 0x556 I2C ID: 0x2A Rev: 0xB
Status:
SEU Buffer DllLock Memory
Clear
Counter:
ReceivedT: 0 RejectedT: 0
Event ID: 0 SEU Cntr: 0
Read Mode
- Previous drift time
- Data Valid
- Comma
- Truncation
- MultiHit
- 0 Events/Trigger

Clear Counters
Debuging
- Service Pads
- Debuging
Service Pad Signal: DLL lock info
Debug Mode: PB PadBankSele

Adjustments

		DLLDAC	ASDDAC
Latency	2	200	100
BX Wrap	8	500	600
BX Offset	1	200	500
Dll Reset		200	100

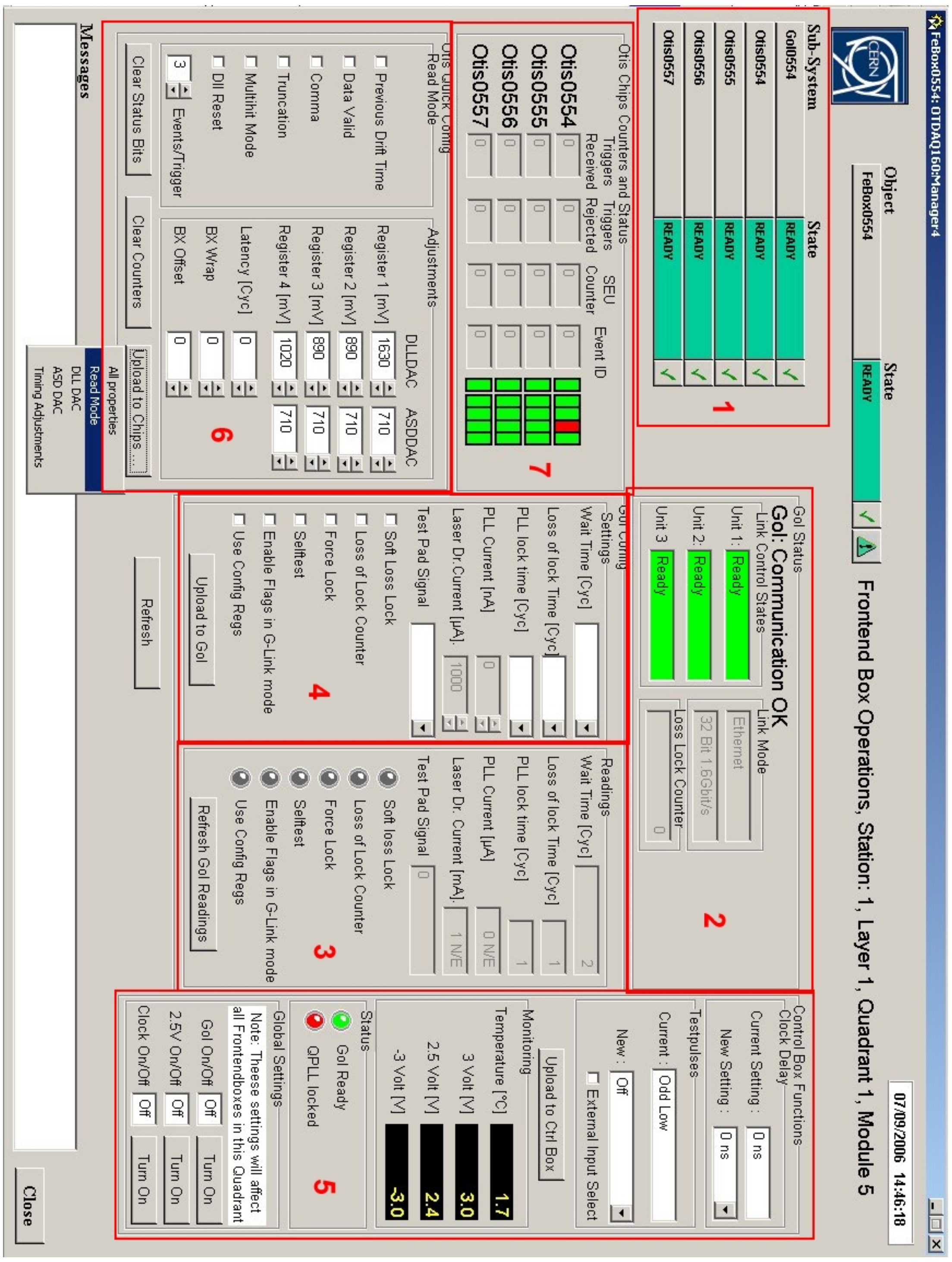
Channel Mask
Refresh

#	ReadFIFO
0	10101001
1	00100000
2	01010101
3	01110111
4	01100101
5	00100000
6	01010011
7	01110100
8	01100001
9	01101110
10	01100111
11	01100101
12	00101100
13	00100000
14	01001000
15	01100001
16	01110010
17	01100001
18	01101100
19	01100100
20	00100000
21	01000100
22	01100101
23	01110000
24	01110000
25	01100101
26	00100000
27	00100000
28	01001101
29	01100001
30	01111001
31	00100000
32	00110010
33	00110000
34	00110000
35	00110100

Messages

Close

Figure 4.7: *Detailed configuration panel for the OTIS chip. The panel is subdivided into* readings *on the right and* writings *on the left. The user enters the necessary settings on the left, uploads them to the chip and verifies the settings on the right.*
A list of all available OTIS devices was inserted on the left to allow for quick configuration of an arbitrary selection of chips.


FeBox0554: OTDAQ16Ö:Manager4

07/09/2006 14:46:18

Frontend Box Operations, Station: 1, Layer 1, Quadrant 1, Module 5

Object: FeBox0554 — State: READY

Sub-System	State
Gol0554	READY ✓
Otis0554	READY ✓
Otis0555	READY ✓
Otis0556	READY ✓
Otis0557	READY ✓

Gol Status: Gol: Communication OK

Link Control States: Unit 1: Ready; Unit 2: Ready; Unit 3: Ready

Link Mode: Ethernet; 32 Bit 1.6Gbit/s

Loss Lock Counter: 0

Gol Config — Settings: Wait Time [Cyc]; Loss of lock Time [Cyc]; PLL lock time [Cyc]; PLL Current [nA]: 0; Laser Dr.Current [μA]: 1000; Test Pad Signal

Soft Loss Lock; Loss of Lock Counter; Force Lock; Selftest; Enable Flags in G-Link mode; Use Config Regs

Readings: Wait Time [Cyc]: 2; Loss of lock Time [Cyc]: 1; PLL lock time [Cyc]: 1; PLL Current [μA]: 0 N/E; Laser Dr. Current [mA]: 1 N/E; Test Pad Signal: 0

Soft loss Lock; Loss of Lock Counter; Force Lock; Selftest; Enable Flags in G-Link mode; Use Config Regs

Control Box Functions — Clock Delay: Current Setting: 0 ns; New Setting: 0 ns

Testpulses: Current: Odd Low; New: Off; External Input Select

Monitoring: Temperature [°C]: 1.7; 3 Volt [V]: 3.0; 2.5 Volt [V]: 2.4; -3 Volt [V]: -3.0

Status: Gol Ready; QPLL locked

Global Settings: Note: Theese settings will affect all Frontendboxes in this Quadrant

Gol On/Off: Off; 2.5V On/Off: Off; Clock On/Off: Off

Otis Chips Counters and Status

	Triggers Received	Triggers Rejected	SEU Counter	Event ID
Otis0554	0	0	0	0
Otis0555	0	0	0	0
Otis0556	0	0	0	0
Otis0557	0	0	0	0

Otis Quick Config — Read Mode: Previous Drift Time; Data Valid; Comma; Truncation; Multihit Mode; Dll Reset; 3 Events/Trigger

Adjustments	DLLDAC	ASDDAC
Register 1 [mV]	1630	710
Register 2 [mV]	890	710
Register 3 [mV]	890	710
Register 4 [mV]	1020	710
Latency [Cyc]	0	
BX Wrap	0	
BX Offset	0	

Messages

Figure 4.8: *Control panel for one front-end box. The panel is completley modularised and built up from reusable components.*
1: State machine summary of child devices.
2: GOL single target status component.
3: GOL single target configuration readout.
4: GOL multi target configuration.
5: Distribution box function panel.
6: Multi target OTIS configuration, as seen in figure 4.6.
7: 4 x OTIS status panels.

4.5 Performance measurements

To allow smooth operation of the detector, the control system should be able to perform tasks like detector configuration or transitions between run modes within acceptable time frames. JCOP suggests "several minutes" as upper limit for a reasonable start up time.

To evaluate the time it will take to fully configure the Outer Tracker front-end electronics, several run time tests were done with increasing numbers of FSM objects. Since there was not enough hardware available to actually test a complete half of the detector at that time, all device units were mapped to one physical Front-end and Distribution Box.

Since the control system is split up into two parallel systems it is enough to do performance measurements for one half of the detector only.

To see if the system could actually handle the enormous amount of Control Units, the first test was done with only the Device Units of two tracking stations installed. *One* half station was configured repeatedly to get an average time per station.

After it was determined that the system would run stable, the third tracking station was also installed and the configuration of *one* tracking station was repeated.

The results of the two measurements are summarised in table 4.1.

Run Number	time [s]
1	42
2	45
3	42
4	42
5	44
6	39
7	42
8	48
Average	43.0 ± 2.7

(a) Two half stations installed

Run Number	time [s]
1	62
2	63
3	60
4	61
5	68
6	60
7	67
8	63
Average	63.0 ± 3.0

(b) Three half stations installed

Table 4.1: *Configuration time for one half station depending on the number of installed half stations. Note that the configuration time increases by $\frac{3}{2}$ from two to three half stations.*

From these results the following conclusions can be drawn:

1. The total configuration time for the front-end electronics will be at least

$$T_{conf} = 3 \cdot 63.0\, s = 189\, s$$

2. The size of the system influences the configuration time per element of the system

3. The increase in configuration time by a factor of $\frac{3}{2}$ per element when increasing the system size by $\frac{3}{2}$ seems to hint at a proportional relation between the number of elements in the system and the configuration time per element. This would lead to an $O(n^2)$ behaviour for total configuration time.

While the 189 seconds of configuration can still be considered within the "several minutes" frame suggested by JCOP, the fact that the configuration time seems to increase quadratically with the number of devices might be considered problematic. At the same time it would also offer the possibility of substantial improvement in configuration time if the number of elements could be reduced.

Since two measurement points are not enough to decide if the behaviour is really quadratic, further analysis with finer granularity was performed and the process that contributed most to the configuration time was searched.

The main contributor to the total run time was found to be a function that searches the configuration data for a certain chip within the recipe database. Figure 4.9 shows a performance analysis of this function for the retrieval of the configuration data for one chip depending on the total number of devices in the recipe. For each of the measurement points the function was called 10.000 times and its mean value and RMS were determined. With the fully installed number of chips, this function takes up approximately

$$\eta_{getRecipe} = \frac{140\,s}{189\,s} \hat{=} 77.8\%$$

of the total configuration time.

During configuration, this function is performing a linear search on the data array containing the configuration data for each Device Unit in the system. With every added Device Unit the configuration time per device increases by $O(n)$ causing the total run time to increase by $O(n^2)$.

A possible strategy for improving performance in this case is to split the data array into several smaller arrays. While the total amount of data stays constant, the total run time for each array decreases by a factor of

$$t_{part} = (\frac{1}{n_{parts}})^2 \cdot t_{unparted}$$

with a netto gain in speed of

$$t_{total} = (\frac{n_{parts}}{n_{parts}^2}) \cdot t_{unparted} = \frac{1}{n_{parts}} \cdot t_{unparted}$$

The Configuration Database offers the possibility to define more than one recipe cache per system. By creating a separate recipe cache per Distribution

Box, the size of the configuration data array per recipe can be reduced by a factor of 12.

Performing the run time test for three stations again resulted in a total run time of

$$t_{total} \approx 70\,s$$

which is not even close to the expected performance increase of a factor of 12, pointing to another, different bottleneck. It was found to be related to the fact that each Device Unit spawns a pseudo thread within PVSS. The synchronisation overhead of the enormous amount of threads causes this new bottleneck.

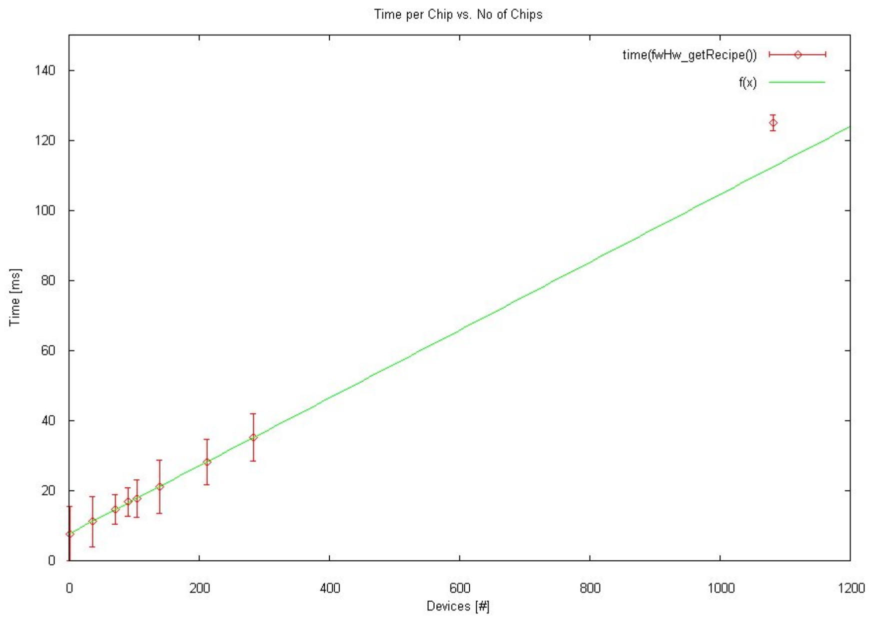


Figure 4.9: *Run time of the configuration data searching function depending on the number of devices inside a recipe. The time per device increases linearly with the number of devices. Since all devices have to be searched for a complete configuration, the total run time increases quadraticaly.*

4.6 Conclusion

An improvement, which was not completely tested by the time of this writing, was to reduce the number of Device Units by combining all the chips within one Front-end Box into one Device Unit. This modification would reduce the number of Device Units by a factor of five and speed up the configuration time by reducing the number of configuration threads and recipe array entries.

Furthermore, it was proposed to redesign the way devices are configured. Instead of having each device look for its configuration in a list, the list should be processed entry by entry. During a full configuration each entry has to be uploaded to a device anyway, making a search unnecessary. This suggestion was being worked on by the creators of the JCOP framework at the time of this writing.

A lot of pioneering work has been done with the development of the Outer Tracker control system since it was one of the first sub detectors in LHCb to start development on a control system. A lot of feedback has been exchanged with the creators of the JCOP framework and bugs and design issues were identified and worked out. While the current control system was already a success during commissioning tests, the final system can be expected to perform significantly better.

Chapter 5

Commissioning results of the Outer Tracker readout electronics

During the commissioning phase of the detector, all components of the Outer Tracker electronics are tested again to evaluate their behaviour outside the laboratory and under the aspects of being integrated into a large system.

A number of tests were performed at CERN with $\frac{1}{12}$th of the Outer Tracker front-end electronics to verify lab results and to study the behaviour of a large system.

These tests included but where not limited to:

- Verifying the functionality of the readout chain from front-end electronics to DAQ farm.
- Determining the noise levels of the Front-end Boxes when connected to the modules.
- Testing the calibration/test pulse system
- Comparing the OTIS's time properties to measurements done in the lab.
- Looking for any general effects that can be explained by running a large system instead of individual components.

5.1 The Mini-DAQ system

To test the electronics on a medium scale, a setup at CERN, denoted the Mini-DAQ system, was built. The Mini-DAQ contains all components of the final Outer Tracker read-out and control hierarchy.

The Front-end Boxes were mounted to their final positions on one of the C-Frames and connected to the wire chambers and the C-Frame's TFC, Readout and Slow Control infrastructure.

The Run Control and DAQ for this system was provided by the LHCb Online Group in terms of a Commissioning Rack (CRack). The CRack is a mobile, miniaturised version of the final LHCb online system. The components and their function are listed in table 5.1.

The Slow Control system introduced in chapter 4 was installed on the Windows PC, together with the Fast Control system for the ODIN board [2].

Component	Function
2 x TELL1 Board	Acquisition of data from the front-ends
ODIN	Fast Control distribution and Readout Supervisor
Windows PC	Control PC running multiple PVSS systems for Slow and Fast Control
Linux PC	Control PC running the DIM DNS node and the boot server for all other components
4 x Linux PC	Farm nodes used for data acquisition
Linux PC	Control PC for the four farm nodes. It contains one hard drive which is shared among the node PCs as network share.

Table 5.1: *Commissioning Rack components*

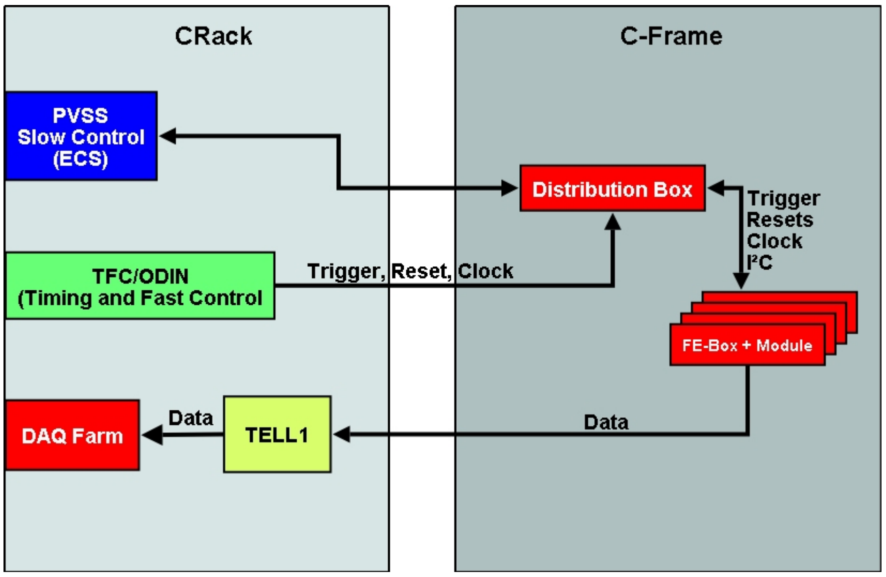


Figure 5.1: *Schematic of the Mini-DAQ setup*

The functional scheme of the setup is displayed in figure 5.1 and resembles the final composition of components.

The Front-end Boxes are connected to the fast and slow control system via the Distribution Box. They are connected to the Distribution Box with SCSI cables. The Distribution Box receives the Fast Control signals from the ODIN through an optical link and the Slow Control signals via CAT5 SPECS cable.

The drift time data is sent to the TELL1 board via an optical fibre. The TELL1s are connected to the DAQ computers by gigabit Ethernet cables.

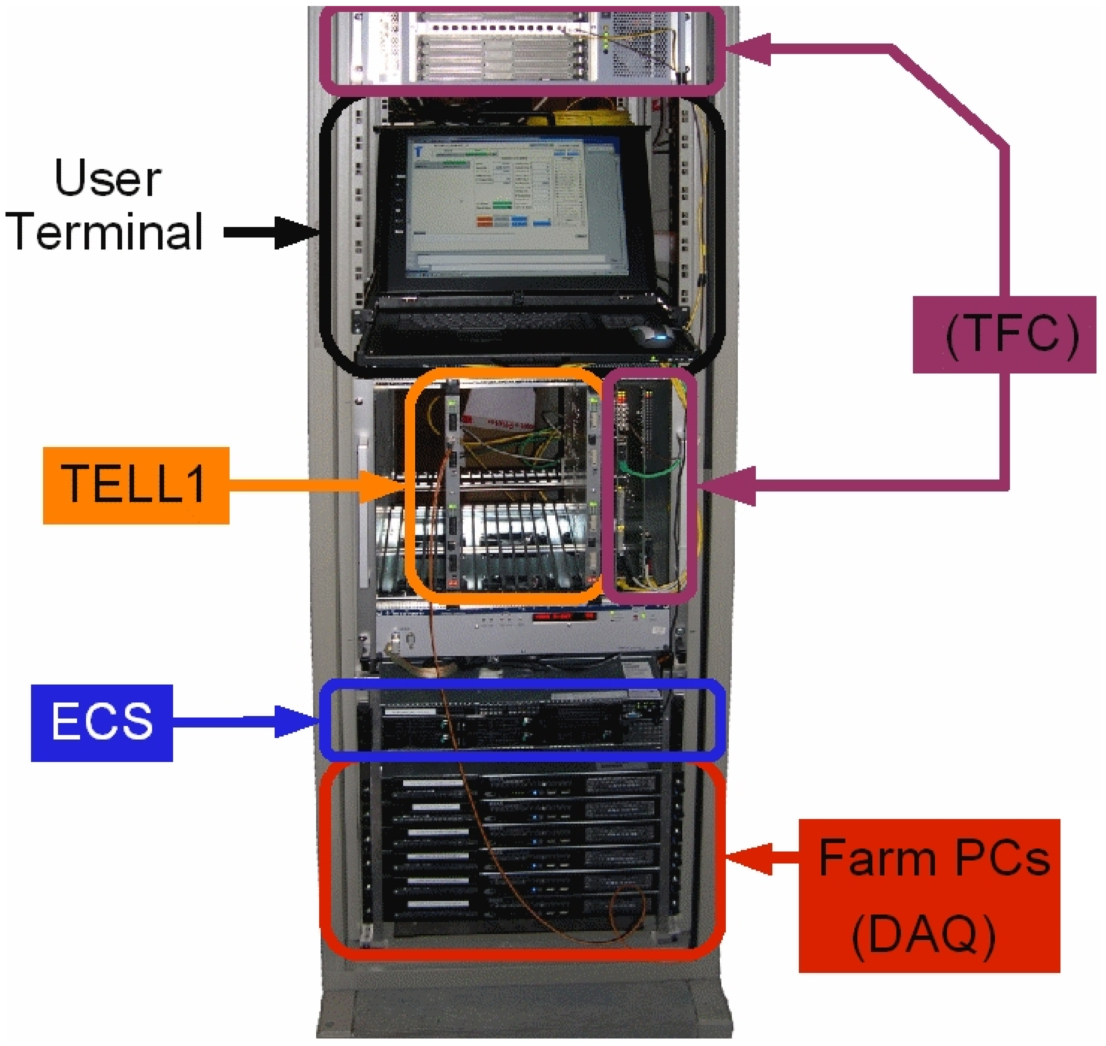


Figure 5.2: *Commissioning Rack provided by the LHCb Online group.*

5.2 Test of the Mini-DAQ system

To test the functionality of the Mini-DAQ setup, especially the data acquisition aspect, several tests of the data transfer process from front-end electronics to the DAQ nodes of the CRack were performed. One particular test of the data distribution system is presented here as an example.

The trigger rates for the test setups presented in this document are all in the order of a few hundred Hertz up to one kHz. This rate will be increased to 1.1 MHz, once the experiment is fully operational. To be able to cope with the enormous rate at which data is generated from all detector components, LHCb had to develop a concept to distribute the load of data acquisition to a computer farm.

In case of the Outer Tracker, each event contains a four bit number (EventID) which is sent along with the event data. The TELL1 board receives this data and combines the event fragments from all connected OTIS chips into a so called Multi Event Package (MEP). These MEPs are sent to their destination computer via a gigabit Ethernet link. The MEP is encapsulated into a common IP frame and sent to the IP address of the target DAQ computer.

MEPs can contain the data of multiple events at the same time to reduce network header overhead. For this test the packing density was set to one event per MEP.

It is the responsibility of the TFC system to make the TELL1 board aware of the destination each MEP should be sent to. To do this, the ODIN Readout Supervisor keeps an internal list of possible destination MAC addresses and can be programmed to assign MEPs by a round robin policy[1].

To verify the functionality of this distribution policy, the following test was carried out.

Three Front-end Boxes were connected to the TELL1 board and configured with default values. The ODIN was programmed with the IP addresses of the four farm nodes of the CRack and destination assignment was enabled. The round robin scheduler was configured to assign only one MEP to a particular farm node before switching to the next node.

If the system works properly, the EventID between any two consecutive events should always increase by four on a particular node. Since 16 values are possible for the EventID and there are four nodes, each node should always receive events with the same four IDs. Figure 5.3 shows histograms of the EventID distribution on all four nodes. As expected, each node saw the same four IDs, for the most part. The small bars besides the main peaks hint to something unexpected.

Inspecting the ID difference between two consecutive events in figure 5.4 reveals that there are several occurrences where two consecutive events would not

[1]Round robin is a scheduling policy, where a limited resource (MEPs) is assigned to clients (DAQ nodes) in portions of equal size. MEPs are limited here because there is always only one MEP available for distribution.

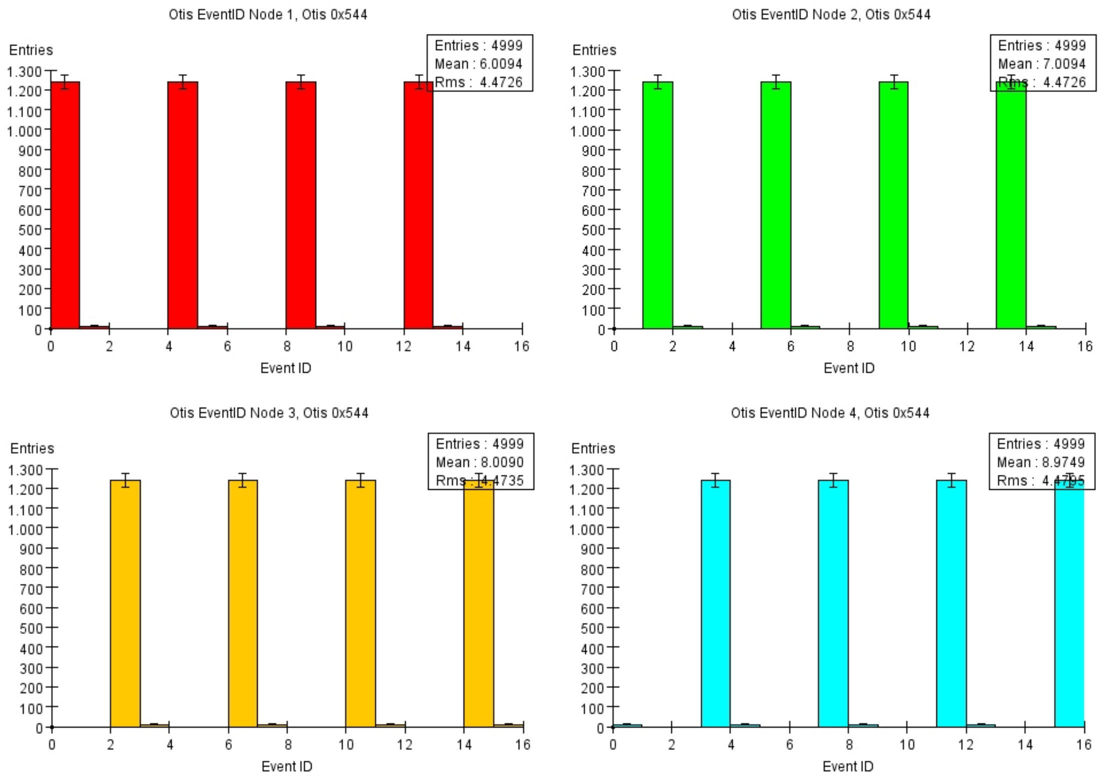


Figure 5.3: *Example EventID distribution for Node1-Node4 for OTIS chip 0x544. The EventID increases by one for each node. The small side peaks are caused by the skipping of Node3. The shift happened very early during data acquisition. The small bars are the original counting sequence.*

have a spacing of four. This can be traced back to two distinct effects:

All the multiple of four entries can be explained by dropped MEPs. The program used for the DAQ could not keep up with the rate the MEPs arrived at the Ethernet port and had to discard those packets. This behaviour was expected, because the DAQ program was a prototype and did a lot of analysis on the received data at run time.

Part of this data is debug and error information, causing a lot of network overhead. This overhead is another source of dropped packets related to the networking hardware. Since the MEPs are sent as Ethernet packets, the switch forwarding the MEPs to the farm nodes is allowed to discard those packets if it can't keep up with the rate at which network packets arrive. In case of the LHCb DAQ network protocol a discarded packet is lost permanently because acknowledgements for MEPs were sacrificed for higher throughput rates. This problem is prevented in the final setup by utilising network switches with 80 MB of packet cache per channel.

The occurrences of differences of three and seven are harder to explain. Table

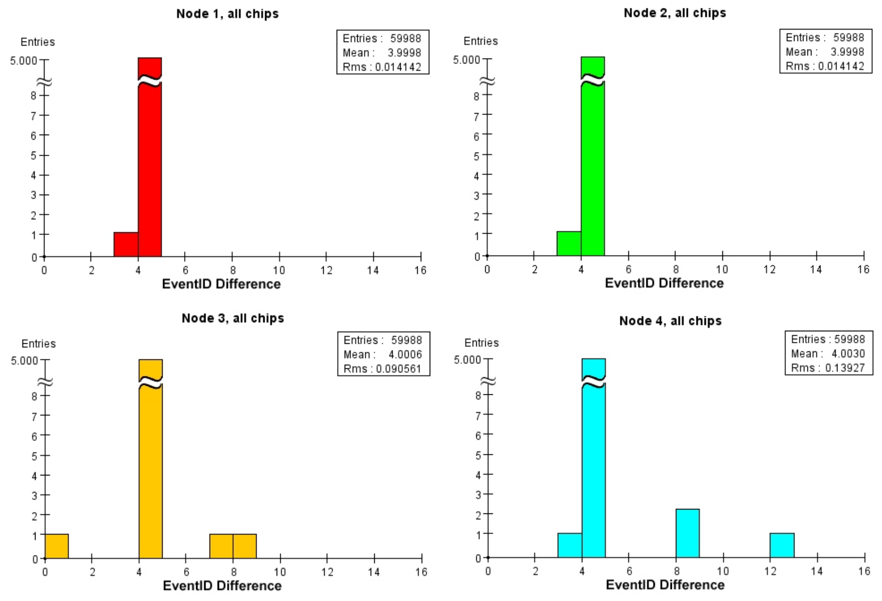


Figure 5.4: *EventID difference for all DAQ nodes and OTIS chips*

5.2 shows an excerpt of the data stream at the point in question. It seems that the ODIN skipped node three in its distribution cycle at one point. This is causing the jump of seven events on that node and the shift of three on all the other nodes. It is proposed that the ODIN will take DAQ node occupancy into account when assigning events later. This mechanism might have already been implemented but not documented in the utilised ODIN firmware.

This example shows that the transport from the front-end electronics to the farm is working as expected. The encountered effects are either noncritical or known and expected.

Node 1	Node 2	Node 3	Node 4
0xD	0xE	0xF	0x0
0x1	0x2	0x3	0x4
0x5	0x6	$\rightarrow$	0x7
0x8	0x9	0xA	0xB
0xC	0xD	0xE	0xF
...	...	...	...

Table 5.2: *Event Number discrepancy. The assignment of data to node three is being skipped causing all other nodes to see an event difference of three for the next four events. After that, the difference is back to four again. Data was not lost but rather rescheduled to a different node.*

5.3 Noise Measurements

The time measurement performed by the OTIS chip determines the drift time of electrons inside the straw tubes relatively to the Bunch Crossing signal of LHC. Each electron cascade caused by a passing particle contains a certain charge which is deposited on the wire and fed into the ASDBLR chip for pre-amplification. Since the system ultimately measures the voltages across a capacitance, the following measurements are using units of voltages rather than charges. Equation 5.1 is an empirical formula that can be used to convert from one system to the other [24].

$$C_{thr}[fC] = e^{(-1.25+0.0033*V_{thr}[mV])} \tag{5.1}$$

Since the straw tubes are acting as a large capacitance and antenna, they are a big source for noise. The following measurement was done to determine the noise level when the Front-End boxes are connected to the modules and assembled into a large system. The measurement is performed similar to the I^2C measurement explained in chapter 3.

The sensitivity of each ASDBLR chip can be adjusted by an external, analog signal provided and configured on the OTIS chip. It denotes the threshold voltage/charge at which the ASDBLR discriminates the input pulses.

This threshold was varied between 500 mV and 900 mV in steps of 25 mV. 10.000 events were recorded for each setting. The number of registered "hits" for one OTIS were summed up and plotted against the configured threshold voltage. The red data points in figure 5.5 show the result of this test for one Front-end Box. The foreseen working point for the modules is around the 750 mV setting. At this point the occupancy through noise has already dropped under

$$\eta_{noise} \leq 0.1\%$$

This is a very good result, taking into account that the final grounding scheme was not applied in this test. As comparison, the I^2C noise measured in Dortmund was also inserted into the diagram (blue) to emphasise the noise that is caused by attaching the Front-end Boxes to the modules.

5.4 Test pulse system

The test pulse system is supposed to be a tool for testing and monitoring the Outer Tracker electronics [22]. Among others, it provides the following functionalities:

- Being a tool for testing the functionality of the complete read out chain and all electronics components on the chain.
- Providing a stable phase with respect to the Bunch Crossing signal to allow measurements of the time resolution of the channels.

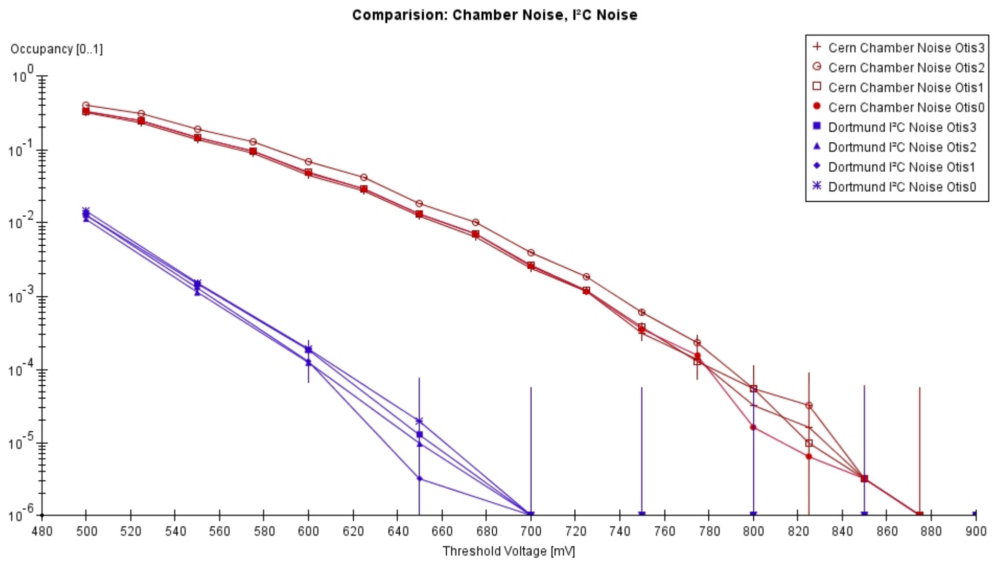


Figure 5.5: *Wire chamber noise measured at CERN (red). The noise levels is under 0.1% at the foreseen working point of 750 mV. The I^2C noise measurement has been inserted to allow a qualitative comparison between the boxes in the laboratory and connected to the wire chambers on the C-Frame.*

- Determining the sensitivity of the individual channels on the detector.

The system allows the injection of charge pulses into the pre-amplifier ASD-BLR chips. The pulses simulate the signals from electron charges deposited on the module wires.

Two kinds of pulses are available:

- Test pulse High: This is a 14 fC pulse with low jitter. It was designed to test the timing alignment and channel separation.

- Test pulse Low: This is a 7 fC pulse. It was designed to check the ASDBLR behaviour at the foreseen working point.

Each test pulse can be sent to either odd or even channels. This separation is done to detect channels which are causing crosstalk to neighbouring channels. Each of the following tests has been done for even and odd test pulse signals. The results are shown for odd test pulses only for brevity.

Pulse distribution can be coupled to the TFC system's calibration pulse [19]. It allows the generation of pulses which are phase locked to the Bunch Crossing signal of LHC. The relative phase to the bunch signal can also be adjusted by the TFC system.

5.4.1 Channel sensitivity divergence

Ideally, microchips coming out of mass production should behave identically and have identical properties. This is never the case due to inhomogeneities in the materials used, position of the chip on the wafer and other effects. It is especially problematic for analog circuits.

The ASDBLR chip is an analog to digital converter and suffers from these effects. Each input channel of the ASDBLR has a slightly different sensitivity. During production of the ASDBLR boards only chips with a maximum difference of 50 mV between each channel have been used. This value results from the requirement of the Outer Tracker to have a charge resolution of at least 0.3 fC.

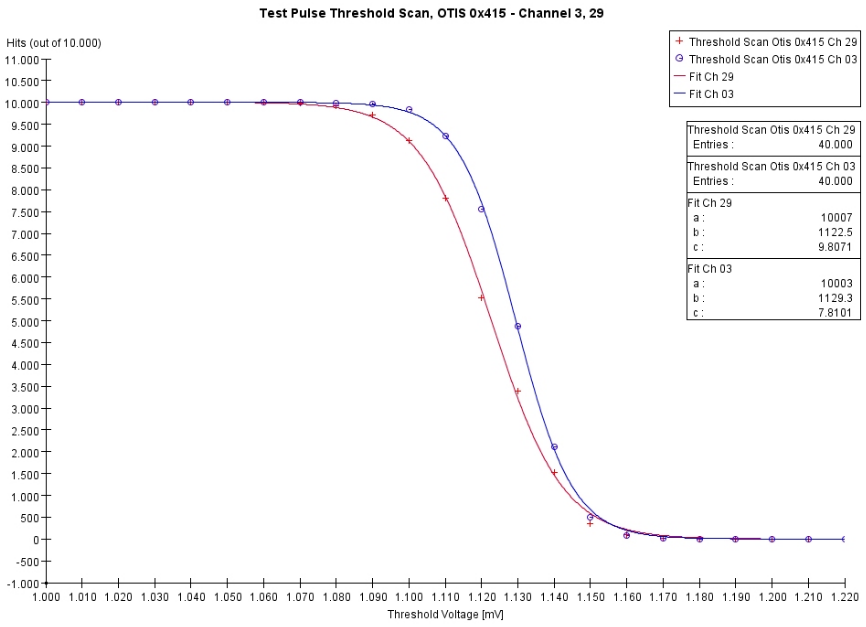


Figure 5.6: *Two sample test pulse threshold distributions. An Ideal distribution would be a step function. Noise is softening the step. Each of these distributions (over 1200) is fitted with a Fermi function to determine* $V_{Thr}^{50\%}$

A threshold scan of pulses with known charge can be used to measure the relative sensitivity of all channels on one ASDBLR board. These measurements have been carried out before for the pre selection of ASDBLR chips [23] [24] [25]. Later, they were repeated with the Mini-DAQ setup to verify the correct function of the boards in a large system.

The threshold scan is performed by activating the test pulses for all Front-end Boxes and varying the ASDBLR threshold voltage around the expected test pulse voltage. 10.000 random trigger events are recorded for each threshold. The result is a distribution similar to figure 5.6 for each channel. When the threshold value reaches the level of the test pulse, the number of “hits” starts to diminish.

Under ideal conditions this distribution would be a step function. Electrical noise causes a softening of the edge due to noise increasing or decreasing the height of the test pulse signal. The Fermi function:

$$f(x) = A \cdot \frac{1}{e^{\left(\frac{x-B}{C}\right)} + 1} \tag{5.2}$$

was used to fit the distribution, which is an adequate choice here as demonstrated in figure 5.6.

Parameter B denotes the point at which the sensitivity of the channel equals 50%. This point is considered as the real test pulse height and is used to determine the sensitivity of each channel. The threshold voltage difference for two channels on a chip should be limited to 50 mV due to the selection done on the chips.

A histogram for the maximal difference between any two channels on one chip is shown in figure 5.7 for high and low test pulse levels. Most chips are within the expected margin and the distribution is similar to the pre selection distribution from [25].

Figure 5.8 shows the same data as deviation from the chip wide average for each channel, for high and low test pulse levels. The distributions are also well within the ±25 mV limit.

Figure 5.9 shows the absolute value of $V_{Thr}^{50\%}$ for high and low test pulse levels. It demonstrates the necessity to distinguish between chip wide variances and detector wide variances. During calibration of the detector this measurement can be used to find individual threshold settings for each ASDBLR chip to normalise the sensitivity of all chips across the entire Outer Tracker.

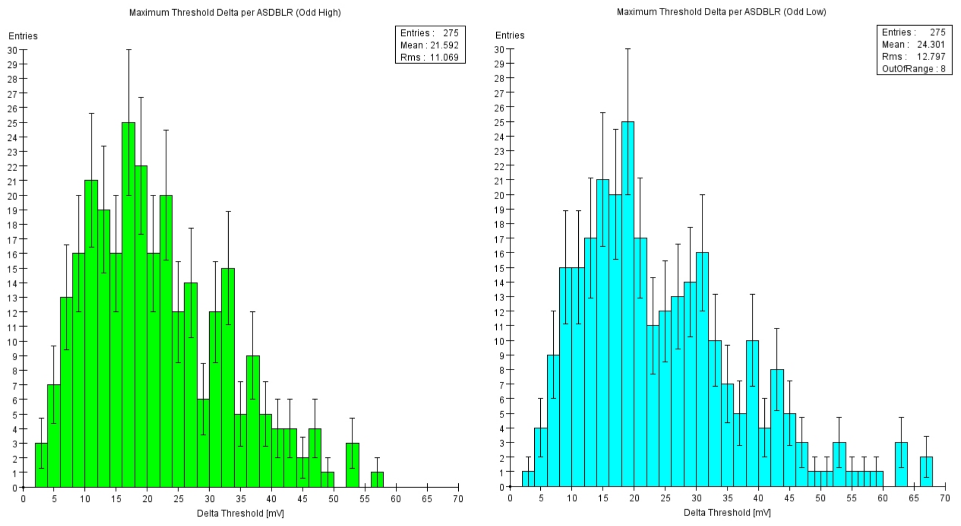


Figure 5.7: *Maximum difference between two channels within one ASDBLR chip. Each entry corresponds to one ASDBLR chip.*
Green: Odd High test pulse
Blue: Odd Low test pulse

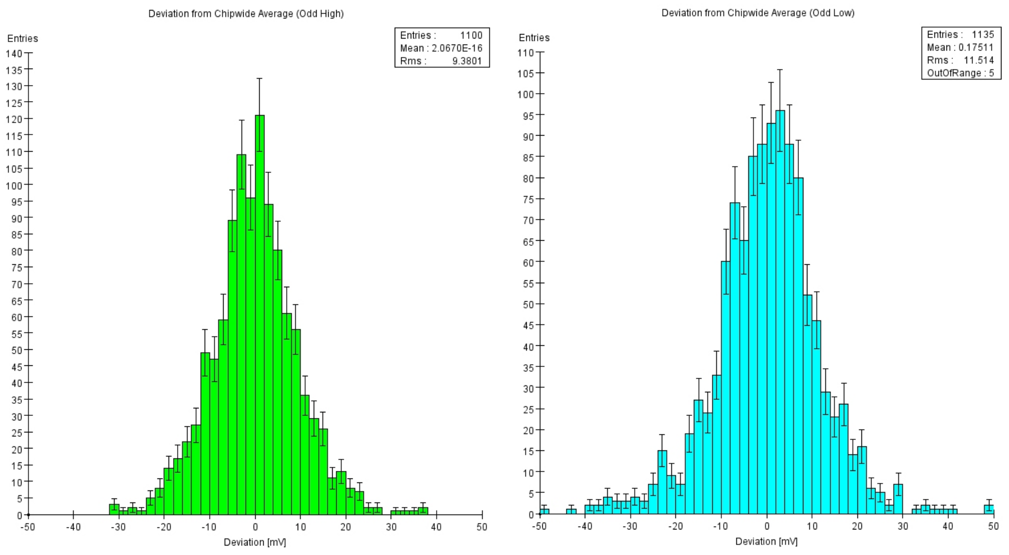


Figure 5.8: *Channel deviation from the average threshold value per ASDBLR chip. Each entry corresponds to one channel. Well above 90% fulfil the ±25 mV specification.*
Green: Odd High test pulse
Blue: Odd Low test pulse

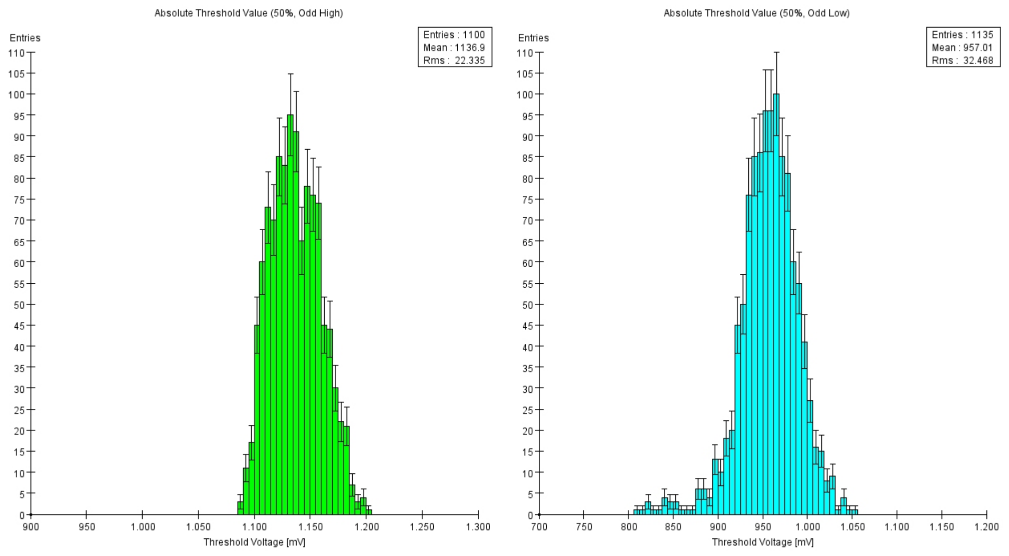


Figure 5.9: *Absolute threshold for each channel. Each entry corresponds to one channel. The different number of channels for high and low test pulses is a result from cutting off unresponsive channels.*
Green: Odd High test pulse
Blue: Odd Low test pulse

5.4.2 ASDBLR Asymmetry

Another property of the front-end electronics can be observed within the test pulse threshold scan data. It was originally discovered when conducting threshold scans of pure noise as seen in figure 5.10. It shows the sum of all hit masks of all 18 Front-end Boxes for pure noise, recorded at a threshold of 500 mV.

Each OTIS chip is connected to two ASDBLR boards, which are equipped with two ASDBLR chips each. Each ASDBLR has eight input channels which are connected to the detector straws.

Looking at figure 5.10, an asymmetric tendency between the right and left chip on the ASDBLR boards can be observed. There are always groups of eight (one ASDBLR chip) channels which receive more or less hits.

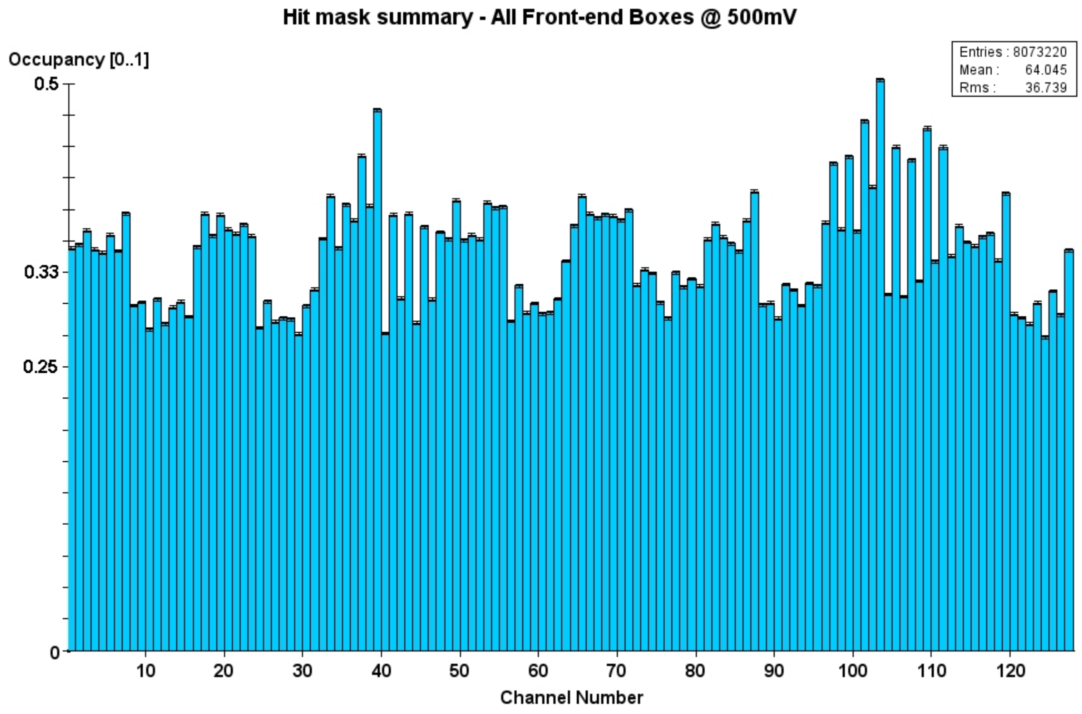


Figure 5.10: *ASDBLR asymmetry between right and left ASDBLR boards for a pure noise measurement at a threshold voltage of 500 mV. The total number of Front-end Boxes for this test was 18. All boxes were connected to the straw tube modules.*

Summing up all left and right ASDBLR chips results in figure 5.10 and mapping them to a channel interval of [0..7], a clear left/right asymmetry and also an even/odd channel asymmetry can be observed. The even/odd asymmetry was already known and expected.

Since a threshold value of 500 mV is very low[2] and not realistic for the exper-

[2]The foreseen working point is approximately 750 mV

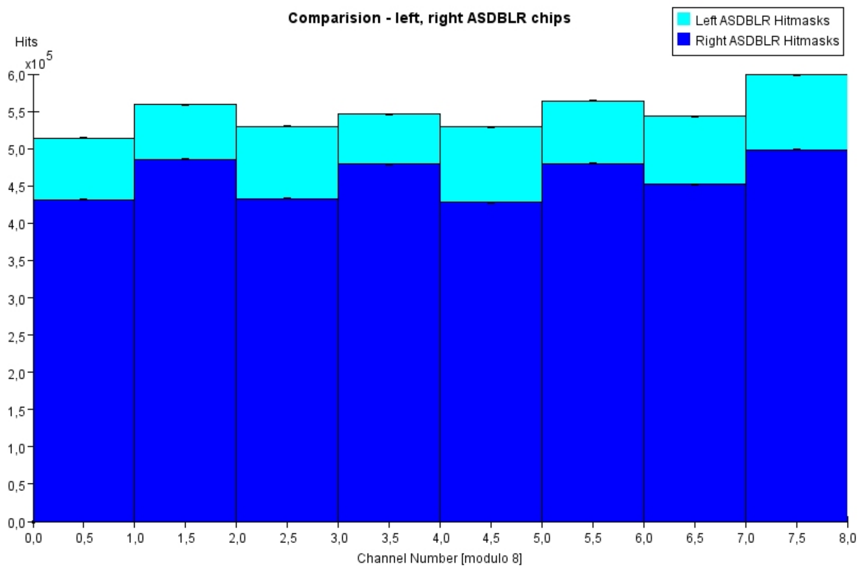


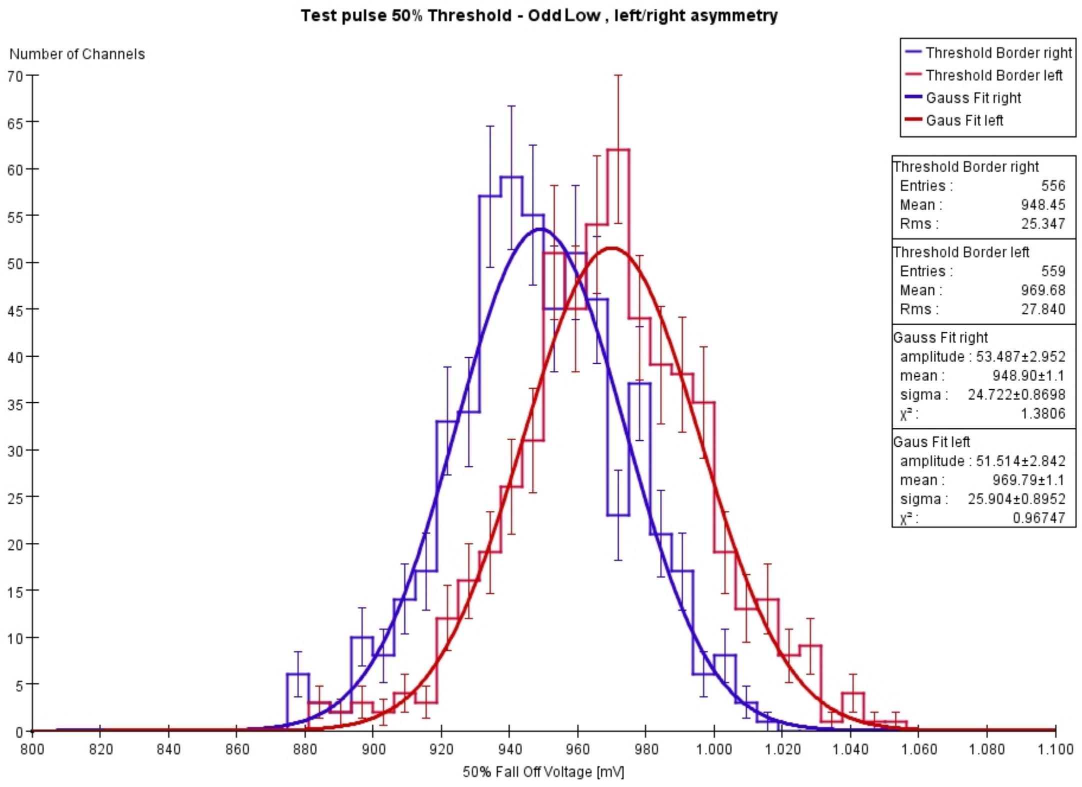
Figure 5.11: *Channel hit mask from fig. 5.10 sorted by left and right ASDBLR chips and mapped to channels 0-7. The maximum possible occupancy of one channel is 10.000 Hits or 0,1 in units of the shown Y-Axis. The data shown is the sum of 144 ASDBLR chips for each side (light blue = left chips; dark blue = right chips)*

iment, the noise measurement alone does not have any significance. To enforce this finding for realistic thresholds, the test pulse data presented in section 5.4.1 has been analysed for this effect.

If the two ASDBLR chips on an ASDBLR board really have different sensitivities, sorting the data in figure 5.9 by left and right ASDBLR chips should yield two slightly separated distributions. The mean for the right channels should be lower than for left channels because left channels have higher sensitivity and need a higher threshold setting to diminish the test pulse “hit” rate to 50%.

This analysis of the histogram data for figure 5.9 has been done and the results are shown in figure 5.12. As suspected, right channels have a tendency for lower threshold values than left channels. This is true for low (upper histogram) as well as for high (lower histogram) level test pulses.

This effect will have to be taken into consideration during calibration of the detector electronics. Different threshold voltages can be assigned to each individual ASDBLR chip.

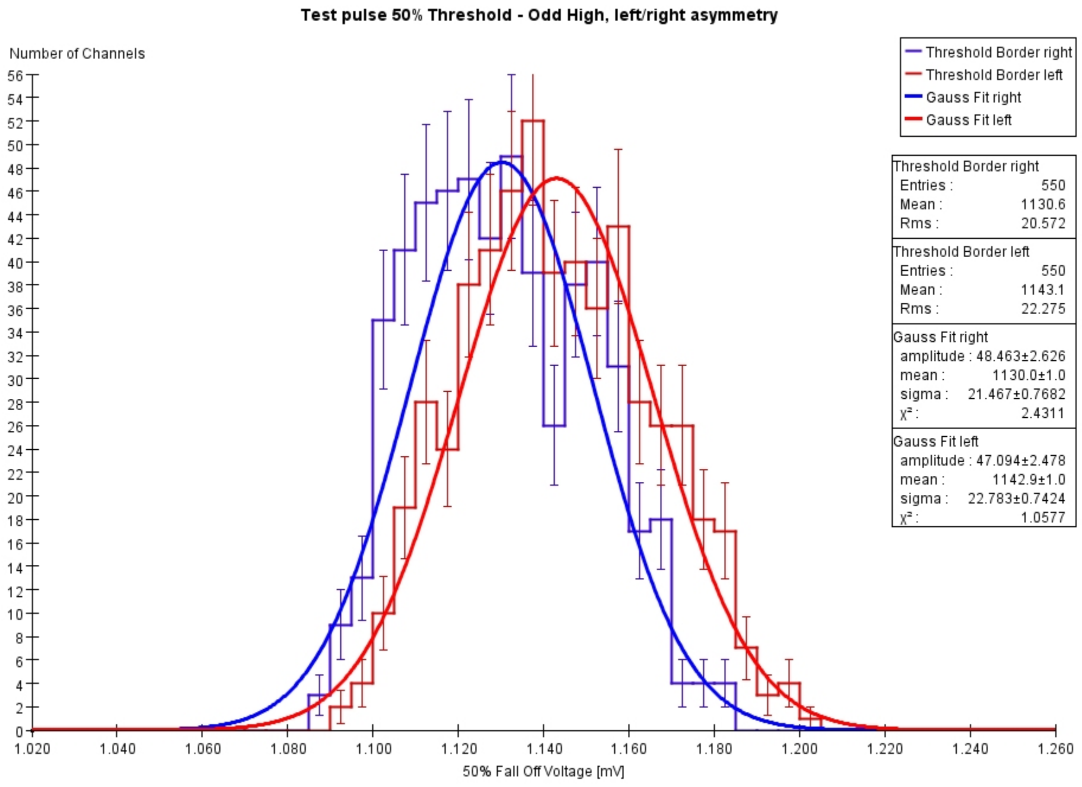


Threshold Border right	
Entries :	556
Mean :	948.45
Rms :	25.347

Threshold Border left	
Entries :	559
Mean :	969.68
Rms :	27.840

Gauss Fit right	
amplitude :	53.487±2.952
mean :	948.90±1.1
sigma :	24.722±0.8698
χ^2 :	1.3806

Gaus Fit left	
amplitude :	51.514±2.842
mean :	969.79±1.1
sigma :	25.904±0.8952
χ^2 :	0.96747



Threshold Border right	
Entries :	550
Mean :	1130.6
Rms :	20.572

Threshold Border left	
Entries :	550
Mean :	1143.1
Rms :	22.275

Gauss Fit right	
amplitude :	48.463±2.626
mean :	1130.0±1.0
sigma :	21.467±0.7682
χ^2 :	2.4311

Gauss Fit left	
amplitude :	47.094±2.478
mean :	1142.9±1.0
sigma :	22.783±0.7424
χ^2 :	1.0577

Figure 5.12: *Absolute values for $V_{Thr}^{50\%}$ for odd low/high test pulses. The channels have been sorted by left (red) and right (blue) ASD-BLR chips.*
Upper histogram: Odd Low test pulse
Lower histogram: Odd High test pulse

5.5 OTIS timing properties

5.5.1 OTIS linearity

The OTIS chip's function within the LHCb experiment is to measure the drift times of electron clusters within the Outer Tracker straw tubes in respect to the rising edge of LHC bunch crossing signal. The bunch crossing signal is synchronous to the collisions of the proton bunches within the beam which phase locks it to the transition of particles through the detector.

The linearity of this mapping procedure was tested with the test pulse system. The test pulse signal can be shifted in 240 intervals with a step size of 104.17 ps, relatively to the 40 MHz bunch clock. This measurement was done in the laboratory before and was repeated here with the complete electronics to study any system effects.

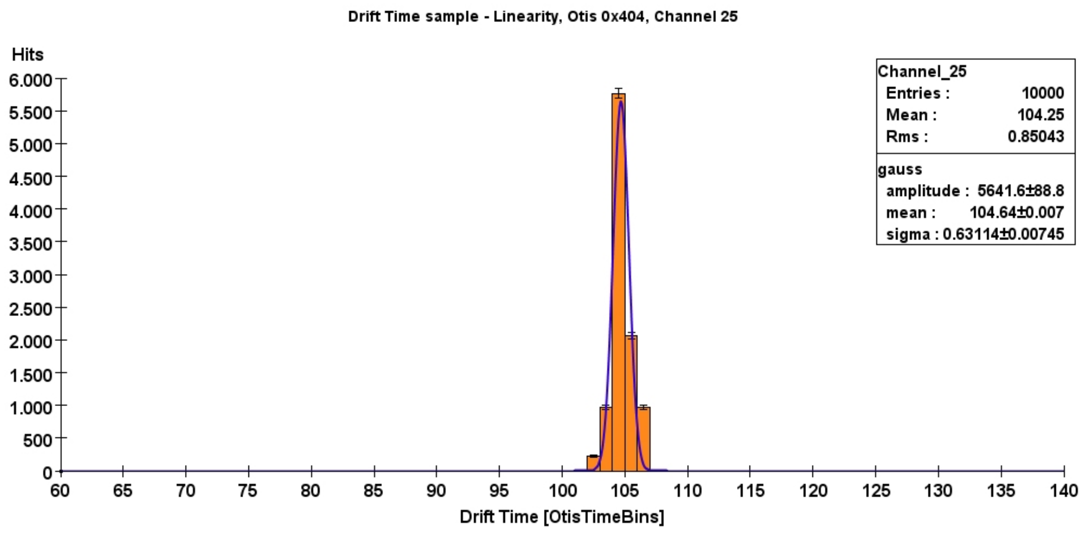


Figure 5.13: *Sample test pulse time measurement for the linearity measurement. A total of approximately 277.000 of these histograms were fitted with a Gaussian to obtain channel plots like figure 5.14. The X-Axis has been expanded to make the fit visible.*

The test was done by recording 10.000 events for each of the 240 intervals. Each recorded data set is similar to figure 5.13 and shows a very narrow peak around the configured test pulse delay. The peak is then fitted with a Gaussian and the mean is plotted against the configured test pulse delay as shown in figure 5.14. The resulting distribution should be a straight line with a slope of $\frac{64}{240}$ which is the ratio of the resolution of the OTIS chip and the test pulse system.

Closer examination (figure 5.16) reveals small deviations from a straight line in form of a step-like structure. The steps begin and end at the borders of an OTIS time bin. This structure is caused by two effects.

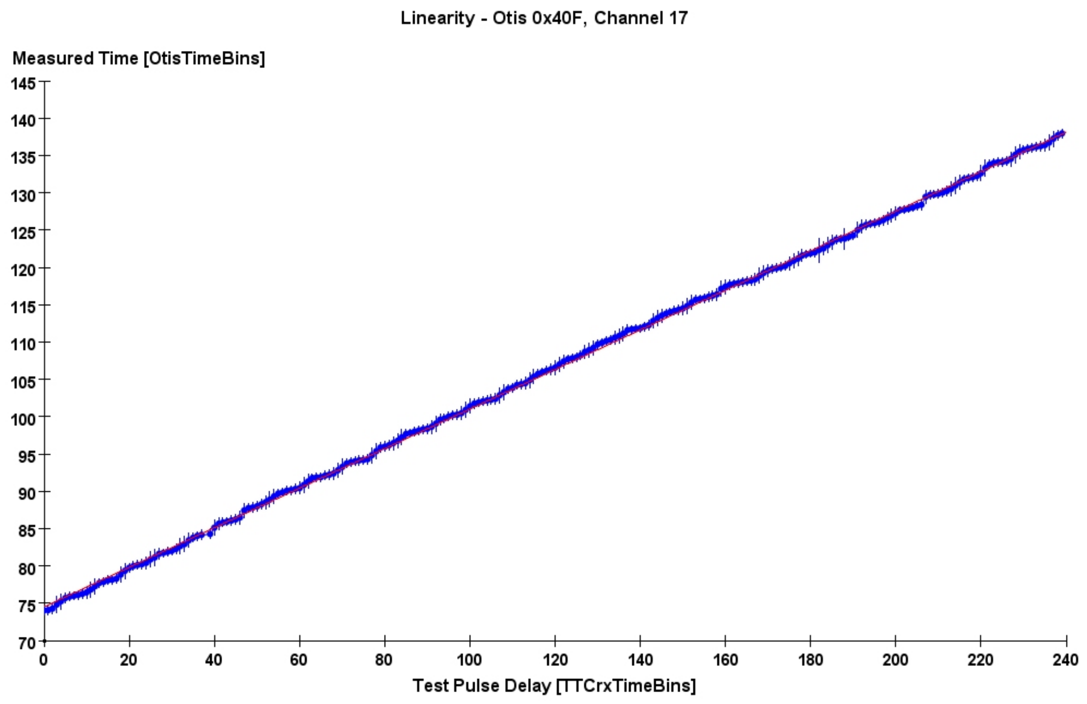


Figure 5.14: *Example linearity plot for OTIS 0x422, Channel 17. There is no visible deviation from the optimal behaviour.*

The first results from the fact that the test pulse delay has a higher resolution than the OTIS time measurement system. Whenever the delay is moved across the border between two OTIS time bins, the measured mean of the pulse suddenly shifts from one time bin to the next.

The second effect is caused by the layout of the OTIS time bins themselves. The bins sizes are alternating between a short and long bins. This also causes a step like effect because the long time bin can contain more delay settings than the short bin, making the slope of short bins steeper than long bins.

By combining two consecutive OTIS time bins, the long-short effect can be overcome. This has been done for the plot in figure 5.15. The distribution is much smoother now, showing that the long-short bin effect is dominating in figure 5.14.

This effect is well known and documented [20][9][8]. To correct it, the TELL1 board will have a look-up table for each channel with corrections for the measured drift times.

In closing, the linearity of all OTIS chips could be verified. The results obtained from the measurements are consistent with previous measurements performed on the OTIS chips in the lab.

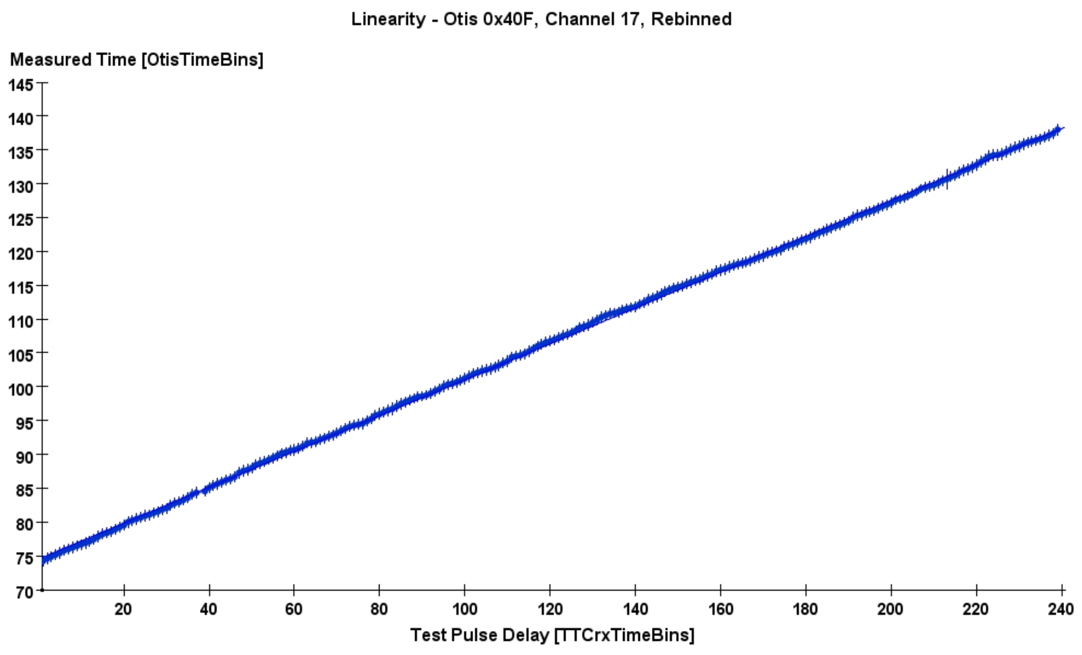


Figure 5.15: *Example linearity plot for OTIS 0x422, Channel 17 after merging two consecutive OTIS time bins. The step structure from figure 5.14 has almost completely vanished, leaving only the binning effect of the higher resolution test pulse delay.*

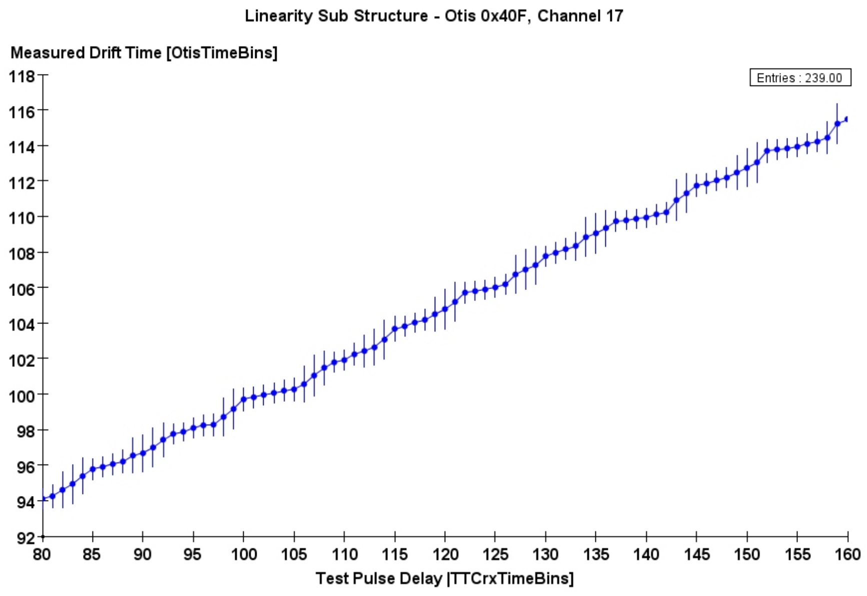


Figure 5.16: *Higher resolution plot of the step structure in figure 5.14*

5.5.2 Long-term stability

To validate the stability of the distribution system, a long term test was performed on the setup. This test was supposed to find any clock drifts or other abnormalities that are only visible after a long time span. The idea was to acquire test pulse events over a very long time and check if the measured OTIS time or the shape of the pulse changes.

The usual test pulse setup was modified by injecting the test pulse into the Distribution Box as external signal instead of TFC calibration commands. The reason for this change was to have an unbiased reference signal. The external test pulse signal is only forwarded by the Distribution Box FPGA without any refurbishing while the TFC calibration command is passing at least one synchronisation circuit for the phase shifted clock from the TTC receiver chip (TTCrx).

The test pulse signal was obtained by connecting the ODIN Orbit[3] signal output to the external test pulse input of the Distribution Box. The signal was passed through a NIM discriminator module to convert the ODIN output signal (ECL) to Distribution Box logic signal levels (NIM). A schematic of the setup can be seen in figure 5.17.

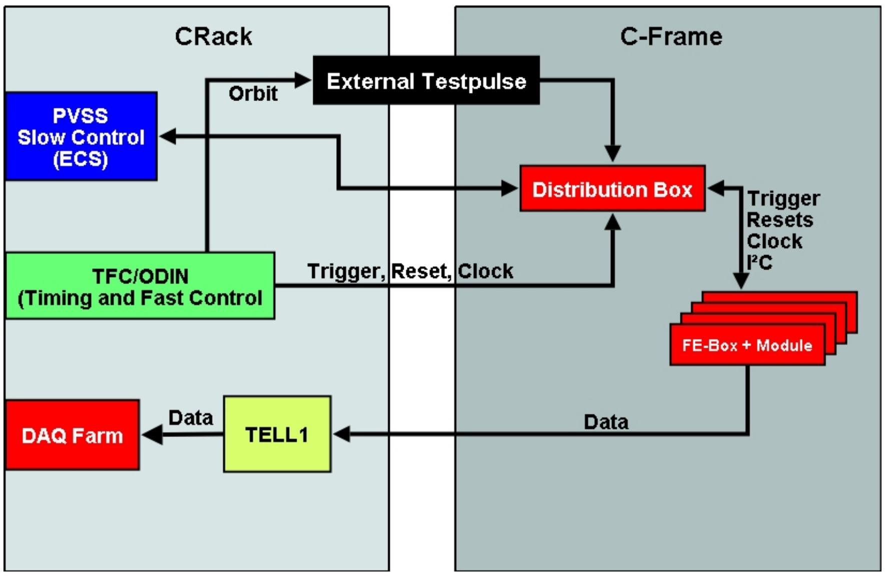


Figure 5.17: *Rewired CRack setup for the stability Measurement. The Orbit signal is extracted from the ODIN board and fed as external test pulse into the Distribution Box.*

The data acquisition was done by an automated script. It was programmed to acquire 10.000 Events every 10 minutes. Odd High was chosen as test pulse type, because high level test pulses have less noise and jitter. After all properties

[3]The Orbit signal corresponds to the time a proton bunch needs to orbit around LHC.

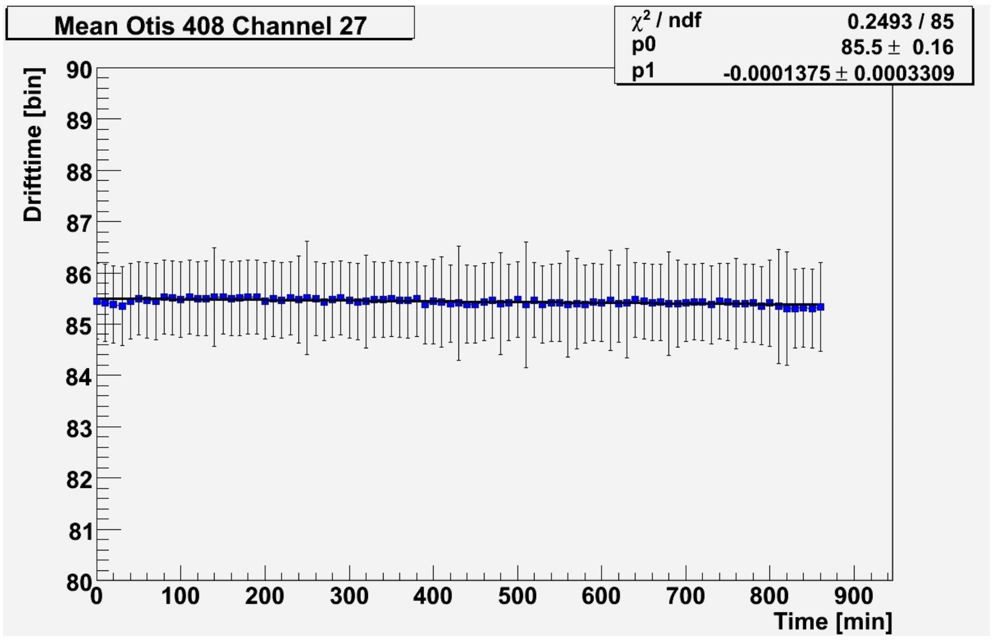


Figure 5.18: *Example stability plot for OTIS 0x408, channel 27. Each entry corresponds to 10.000 events taken in 10 minute intervals. The error bars are RMS values rather than the sigma of Gauss fits.*

were configured, the program was started and left running for approximately 14 hours.

Figure 5.18 shows an example plot of the data gathered for one OTIS channel. For each of the 86 data samples (10.000 events) the average measured time was determined. The resulting values were plotted versus elapsed time of the test.

The measured times are very stable over the 14 h time span. The drift time RMS stays stable during the whole measurement. This means that the clock jitter stays steady too. A fitted first order polynomial shows that there are also no tendencies for a long term drift of the signal.

Figure 5.19 shows a histogram of the fitted polynomial slopes for all channels. The slopes of all channels are consistent with zero. They are in fact all within the same histogram bin at zero.

The histogram has a resolution of 200 bins from -1 to 1. Since the test was conducted over a time span of 14 hours the range for possible drifts can be limited to:

$$maximumDrift \leq 0.3\ \frac{ps}{h} \tag{5.3}$$

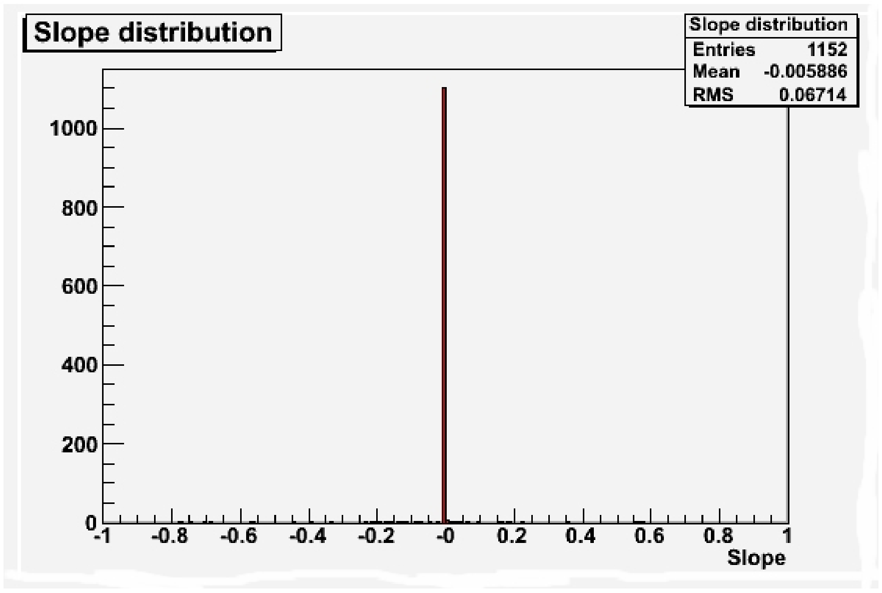


Figure 5.19: *Histogram of all odd channels slopes for the long term stability test. All slopes values are within the same bin which restricts any long term drifts to under 0.3 ps per hour.*

5.5.3 Test pulse and TDC resolution

The resolution of a time measurement for the test pulse signal depends on the influence of two parameters:

- The jitter introduced by the test pulse and TFC distribution system (σ^2_{TP}).
- The resolution of the TDC system inside the OTIS chip (σ^2_{TDC}).

The overall error for the measured time can be calculated as:

$$\sigma^2_{tot} = \sigma^2_{TP} + \sigma^2_{TDC} \tag{5.4}$$

Since the test pulse and TFC signals are split up in the Distribution Box and then sent to the Front-end Boxes they have a strong correlation when arriving. If the signal jitters by an arbitrary time Δt on one channel, it will also jitter by Δt on all other channels.

This strong correlation can be used to disentangle equation 5.4 and determine the values for σ^2_{TP} and σ^2_{TDC}.

If t_1 and t_2 are the measured times of test pulse "hits" on two different channels, their respective errors are calculated by:

$$\sigma^2_{t_1} = \sigma^2_{TP} + \sigma^2_{TDC_1} \tag{5.5}$$

$$\sigma^2_{t_2} = \sigma^2_{TP} + \sigma^2_{TDC_2} \tag{5.6}$$

Adding and subtracting the measured times of the two channels and taking the correlation into account, the total error for the sum and difference is:

$$\sigma(t_1+t_2)^2 = (\sigma_{TP}+\sigma_{TP})^2 + \sigma^2_{TDC_1} + \sigma^2_{TDC_2} \tag{5.7}$$

$$\sigma(t_1-t_2)^2 = (\sigma_{TP}-\sigma_{TP})^2 + \sigma^2_{TDC_1} + \sigma^2_{TDC_2} \tag{5.8}$$

Subtracting 5.8 from 5.7 yields:

$$\sigma(t_1+t_2)^2 - \sigma(t_1-t_2)^2 = 4\cdot\sigma^2_{TP} \tag{5.9}$$

Inserting 5.9 in 5.5 or 5.6 allows the determination of $\sigma^2_{TDC_1}$ and $\sigma^2_{TDC_2}$ respectively. $\sigma^2_{t_1}$ and $\sigma^2_{t_2}$ can be obtained directly by histograming a large enough number of measurements for t_1 and t_2 while $\sigma(t_1+t_2)^2$ and $\sigma(t_1-t_2)^2$ can be obtained by histograming the sum and difference of the measured times of two channels.

The data gathered from the linearity test from chapter 5.5.1 represents a good sample of measured test pulse times for all time bins of the OTIS chip and can be used to determine σ^2_{TP} and σ^2_{TDC}.

The first step to do this is to determine σ^2_{TP} with equation 5.9. Figure 5.20 shows the two histograms for $t_1 - t_2$ (top) and $t_1 + t_2$ (bottom) gathered for one record of the linearity measurement (10.000 events) for channel 3 and 1 of OTIS 0x509. From the Gauss fits one can determine the variances of the two distributions and calculate an approximate value for σ_{TP} of:

$$\sigma_{TP} = \sqrt{\frac{1.25^2 - 0.96^2}{4}} \approx 0,4[OtisTimeBins]$$

To gather more statistics this whole procedure was repeated for all possible channel combinations of all 240 records of the linearity test for OTIS 0x509 and histogramed in figure 5.21. Fitting a Gauss function yields a total error of

$$\sigma_{TP} = 0.32 \pm 0.07[OtisTimeBins] \mathrel{\hat{=}} 0.13 \pm 0.03\,ns$$

for the correlated uncertainty introduced to test pulse and TFC by the distribution mechanism.

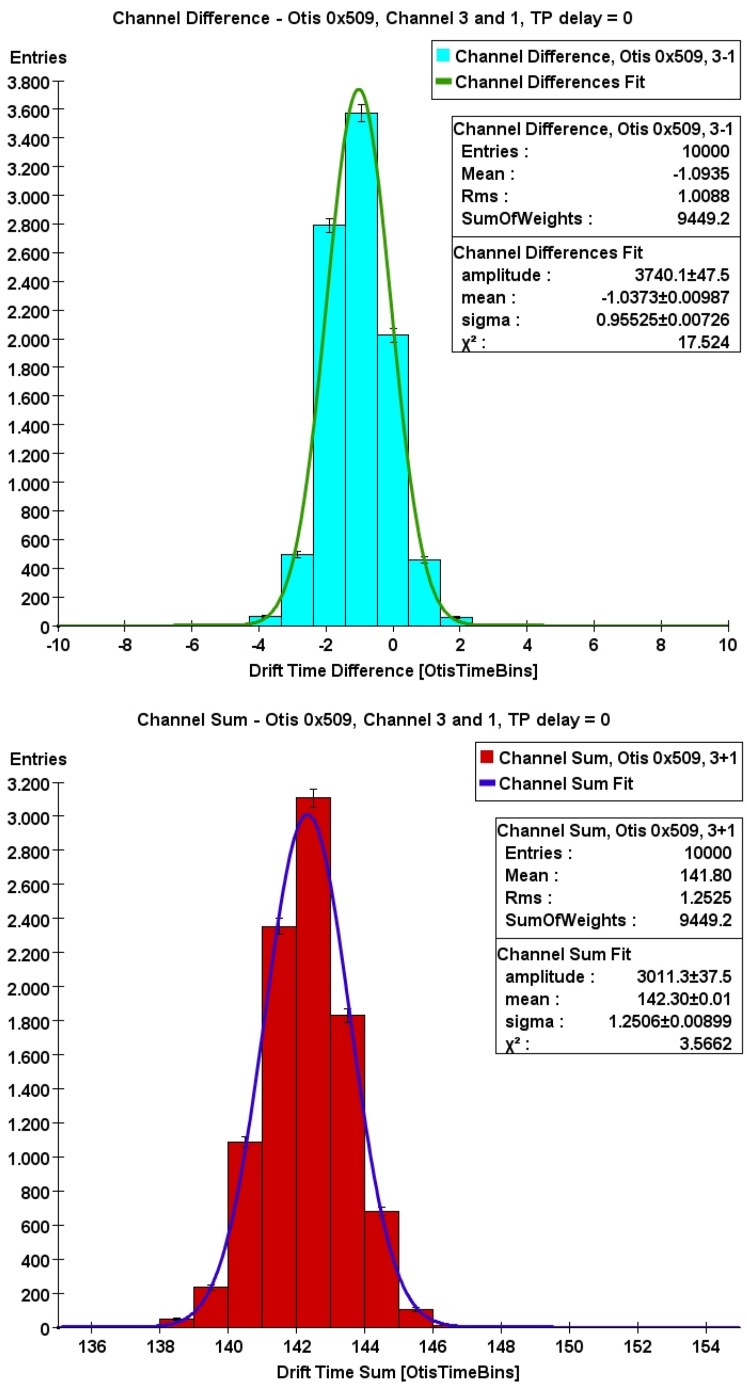


Figure 5.20: *Example for the difference and sum histograms. These histograms are created for each possible channel combination of OTIS 0x509 (only odd channels).*

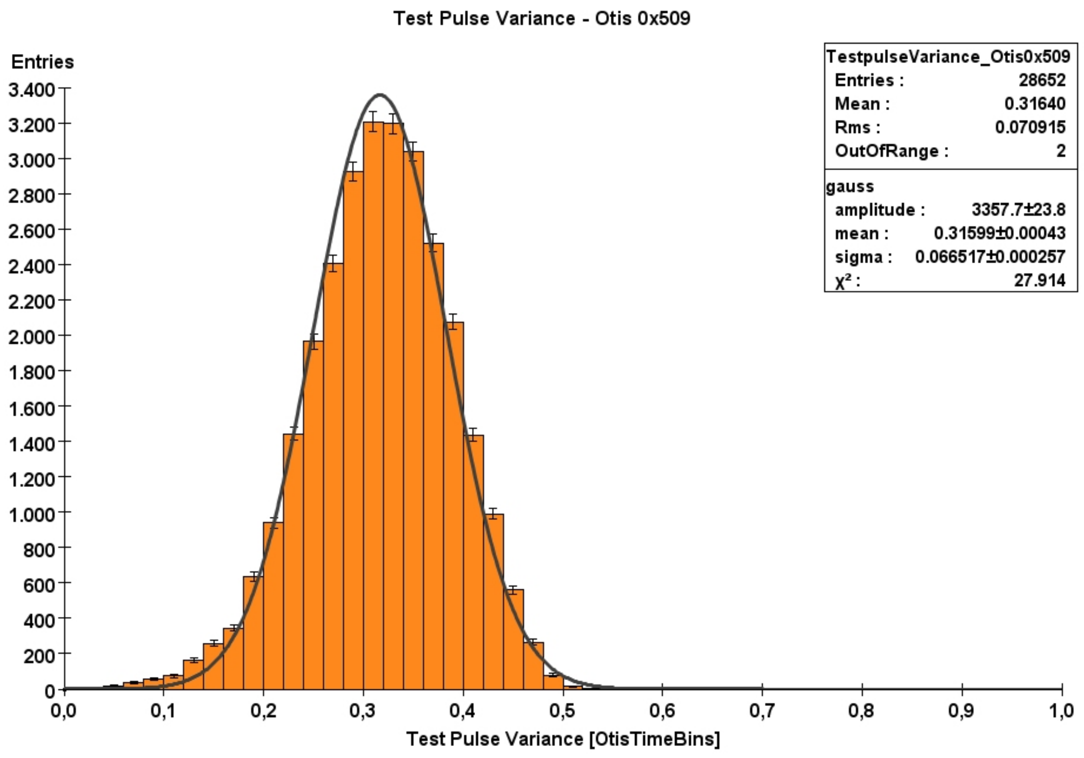


Figure 5.21: *Combined distribution system uncertainty for Otis 0x509. Each entry corresponds to one channel combination for each of the 240 linearity test records.*

Chapter 6

Conclusion

Summary

This thesis describes the testing and commissioning of parts of the Outer Tracker Front-end Electronics.

In the first part, a test bench for the GOL/AUX board was developed. The setup was used to verify the functionality of the boards after their production. The development was based on an earlier setup for the test of the OTIS chip. It included the implementation of a PCI to I^2C interface on an Altera Stratix FPGA and the creation of software that allows the control of GOL and OTIS chips. The implementation accounts the specific needs of a differential I^2C connection.

In the second part of the thesis the first implementation of a Slow Control system for the Outer Tracker front-end electronics based on PVSS was performed. The project combined the following tasks:

- PVSS panels that allow control of every function of the Outer Tracker electronics.
- Implementation of a control hierarchy within the LHCb Finite State Machine and controls hierarchy.
- Testing the performance of the created control system especially in respect to total configuration time of the system.
- Implementation of automated measurement tools for the commissioning phase.

During the final stages of the project, a data acquisition setup was built at CERN. It was the first setup consisting completely of the final readout components. The setup, denoted Mini-DAQ, contained up to $\frac{1}{12}$th of the final electronics. It was used to study the dynamics of a medium sized system of Outer Tracker components. In particular, these studies covered:

- The succesfull validation of the front-end to DAQ farm readout path.
- The behaviour of noise in the final system. Especially the validation of acceptable noise levels at the foreseen module working point.
- A first study using the test pulse system to time align all electronic components on a large scale.
- Evaluating the test pulse system as a mean for normalising the sensitivity of all ASDBLR chips across the detector.
- The verification of previous studies concerning the timing properties of OTIS chips and their behaviour and stability in the large system.

Outlook

The developed PVSS control system was one of the first systems written for the purpose of controlling a whole LHCb sub detector. The pioneering work on this project, especially the performance tests of the Finite State Machine and the SPECS system in general, led to a number of improvements in the design of these software components.

While the control system is only a first iteration and not completely ready to be used by the end-user yet, it offers a solid basis for future developments. Especially the modular panel design will help in expanding and customising it to the required needs.

Appendix A

PVSS

A.1 Supervisory Control and Data Acquisition Systems

A.1.1 SCADA tasks

With the increase of complexity and scale in today's industrial processes and work flows, the need for systems allowing easy surveillance and control of theses processes arises. As the acronym already suggests, SCADA systems are a type of software that is capable of:

- collecting data from hardware
- exerting a *limited* amount of control over the hardware
- do the above from and to remote locations

This sets SCADA systems apart from similar concepts like telemetry which only allow monitoring. "Limited" means that personnel will still have to investigate critical failures at the remote site, should they occur.

Another task of SCADA systems, which has emerged over the past years, is to hide the complexity of the system they control from the operator. The operator should be concerned with managing the work flow, not the hardware.

A.1.2 SCADA architecture

Figure A.1 shows an example of a typical SCADA architecture as described in [15].

Since SCADA applications are supposed to exert remote control they necessarily have to be divided into at least two components, or layers in software terms. These two layers form the end points in a client-server model and are called the *client* and *server* layers.

On the server side, Remote Terminal Units (RTU)s are connected to the hardware and act as data servers for the state information of the hardware. Additionally they allow the input of control commands from the local side.

On the client side, *operator interfaces* are used as interfaces to the human controller. The operator can enter commands for the hardware on the client side and also monitor the state data generated by the hardware, which is relayed by the RTUs.

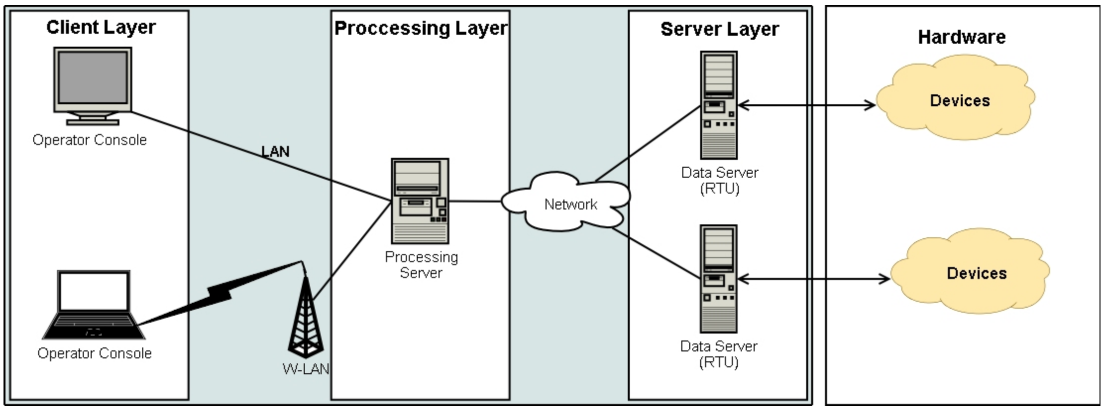


Figure A.1: *Typical architecture of a SCADA application. Client, processing and server layer are separated by communication channels. The SCADA system itself is independent of the hardware it controls. It merely connects to the hardware and is used to relay, filter and manage the information gathered from it.*

A third (processing) layer between clients and servers is inserted to manage the data flow between operator and hardware. It is responsible for filtering information, performing automated or scheduled tasks and do logging and general data storage.

Another advantage of having a processing layer is the ability to keep the data displayed on the different operator interfaces synchronous. It is easier to collect all data in one central location and then forward it to anyone who might be interested instead of connecting directly to the hardware and manage synchronisation there.

A.2 PVSS as implementation of a SCADA system

PVSS[1] is a commercial software package chosen by CERN to be the recommended SCADA system to manage the LHC accelerator itself as well as the associated experiments.

[1]Process visualisation and control system

It was chosen over other products or self made software, because of its genericity and scalability. According to the IT-CO-BE group at CERN [16] its strengths that make it interesting for High Energy Physics (HEP) are:

- It can run in a distributed manner with any of its components running in a distributed manner
- It is possible to integrate distributed systems
- It has multi-platform support (Linux and Windows)
- It is device oriented with a flexible data point concept
- It has advanced scripting capabilities
- It has a flexible Application Programming Interface allowing access to all features of PVSS from an external application

To get a better understanding for the control system developed for this project, a short introduction to PVSS is given here.

A.2.1 PVSS Manager Concept

In PVSS, work is delegated to specialised programs that are specific for certain tasks. These programs are called managers.

PVSS provides several prefabricated managers (e.g. user interface, archivers, drivers, data acquisition, etc.). Furthermore, the manufacturer[2] supplies the user with the Application Programming Interface that was used for the development of the provided managers. It allows the user of PVSS to develop additional managers that have full access to the functionality of PVSS.

Managers are able to communicate with each other via the TCP/IP protocol. This method allows for interesting distribution concepts of a PVSS system which will be shown in section A.2.6.

Figure A.2 shows a schematic of a typical PVSS system. The system is subdivided into four layers.

User Interface Layer: This layer is responsible for interaction with the user. It contains managers that display data or are accepting user input.

Processing Layer: The processing layer allows to run automated scripts that can handle non critical errors or other tasks that do not require user interaction. The Application Programming Interface (API) manager depicted here is a different, more lightweight interface to PVSS. It allows limited access to PVSS features for the trade off of simplicity.

[2]ETM professional, Austria

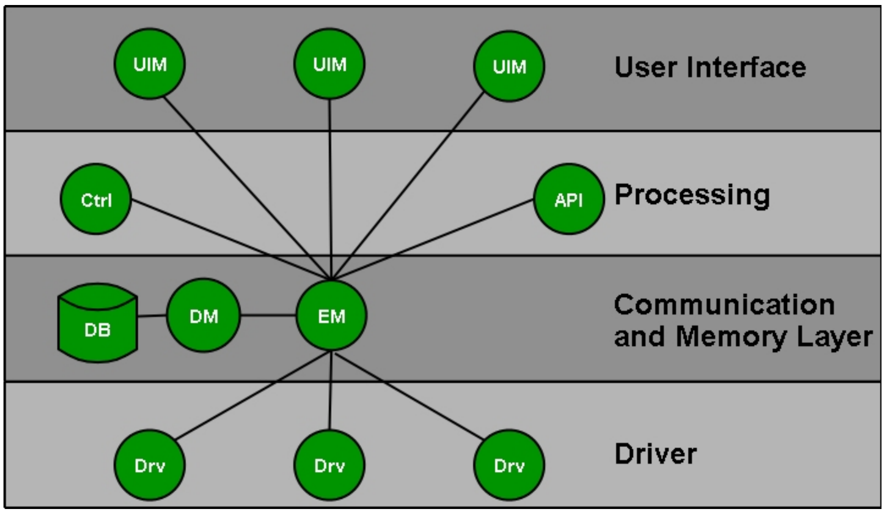


Figure A.2: *Structure of a PVSS system.*

Communication and Memory Layer: This layer is the heart of a PVSS system. The Event and Database manager are the bare minimum components that are required for a operable PVSS system.

Driver Layer: The hardware communication layer. PVSS comes with a rich assortment of industry standard drivers like OPC[3], CAN[4] bus, PROFIBUS etc. Drivers are the equivalent to data servers in the SCADA architecture.

The second and third layer correspond to the processing layer of the generic SCADA application.

A.2.2 Event Manager (EM)

As mentioned earlier, the Event Manager, together with the Database Manager is the central component of each PVSS system. All other managers are connected to the EM and all communication between managers is handled by the EM.

As the name of this manager already suggests, PVSS is an event driven system. This means that, contrary to procedural systems, PVSS will only react to state changes imposed from the outside instead of acting on its own. State changes can be caused by user input, the change of state in some kind of hardware, or a timer that is set to trigger at a specific time. The last possibility, while available, should be used cautiously because excessive use would contradict the purpose of the event driven architecture, which is an integral part of PVSS's scalability scheme.

[3]Object Linking and Embedding for Process Control
[4]Controller Area Network

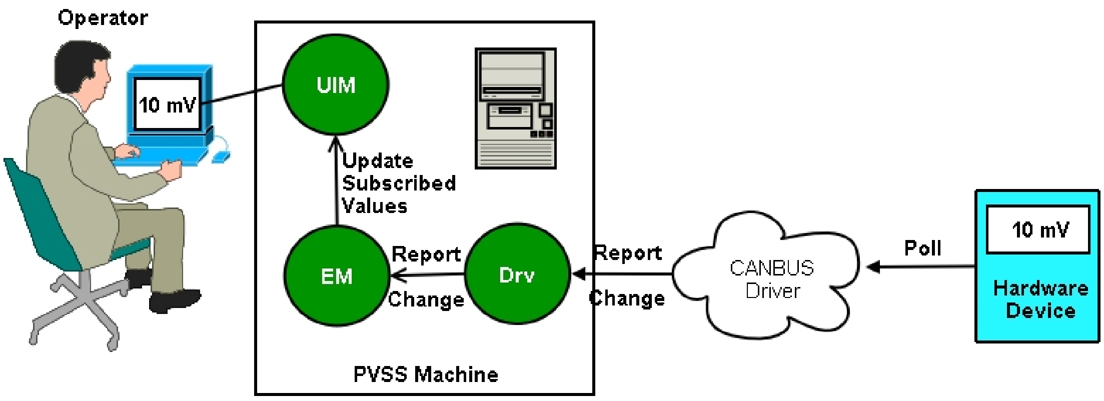


Figure A.3: *Example of a simple PVSS system with only a user interface and a volt meter connected.*

To further understanding of this concept, and because it is one of the core aspects of PVSS, a short example of is shown in figure A.3. An operator is interested in the state of a certain piece of equipment. A device to measure voltages in this case. The Driver Manager that connects the voltmeter to PVSS, registers itself with the Event Manager. The driver (not PVSS itself) polls the voltmeter for changes of its state in regular intervals. If a *significant* change is detected, the driver will forward this change to the EM. The EM will then notify all parties that are *subscribed* to the state of the voltmeter about the change and tell them the new value. In our example, the operator is the only interested client so the Event Manager will only notify the User Interface Manager of the operator.

A consequence of this is, that a PVSS system can only have one Event Manager. This is of course a problem because it introduces a single point of failure to the system. The solution to this problem is to have two PVSS systems that are running in parallel as redundant system.

A.2.3 Database Manager (DM)

The Database Manager is another integral component of PVSS. Its main purpose is to keep an image of the current process in its memory. Additionally all value changes are written to a high speed file database. This database only contains the current values of all monitored variables.

Archiving of process variables is also done by the Database Manager to so called archive files. Variables can be provided with attributes that tell the Archive Manager if a variable should be archived or not. This behaviour, again, shows the philosophy of PVSS of only doing work when necessary.

If a piece of hardware, for example the voltmeter from our previous example, changes its state, the Event Manager will automatically pass this information to the Database Manager which will store the new information. If a manager becomes interested in the value of the voltmeter it can ask the Event Manager for the current value. The Event Manager in turn will ask the Database Manager for the last known value of the device and will pass it on to the interested manager.

A.2.4 Control manager (CM)

The Control Manager is the part of PVSS where a great part of its expandability comes from. A special, ANSI-C compatible programming language called *Control* was integrated into the PVSS. Programs written in this language can be run within an instance of the Control Manager and have full access to all process variables.

Control has been used to write most of the higher level functionality of PVSS. It offers the ability to aggregate commonly used functions in libraries, which can be shared between CMs. Functions can be registered with the Event Manager and are called when a chosen datum changes its value. This feature is again one of the core functionalities of PVSS, because it is the point where data and data handling logic are joined.

Another feature of Control is the ability to import functions from Windows Dynamic Link Libraries or Linux Shared Object Libraries into PVSS. This allows for the creation of light weight drivers, which do not need access to the full functionality of the PVSS API. This feature was used to access the C++ library presented in chapter 3 to gain access to the front-end hardware, when the final communications hardware was not yet available.

A.2.5 User Interface Manager (UIM)

The User Interface Manager is a generic shell from which custom made user interfaces, so called PVSS panels, can be displayed. These panels can be created within a graphical editor and are stored as ASCII text files. This allows them to be easily ported from one operating system platform to another.

The developer can place widgets, representing process information, on the panels and connect these widgets to functions written in Control. This connection allows the panels to access process data. A command to a widget on the panel is sent to the function(s) connected to it. After being processed by the function, it is forwarded as event to the Event Manager, which forwards it to any manager that might be interested in it. Similarly, events happening anywhere within the system can be subscribed by call back functions within the panel which can update the widgets on the panel.

A.2.6 Distribution of a PVSS system

PVSS distinguishes between two different kinds of distribution: Scattered Systems and Distributed Systems.

Scattered Systems

A Scattered System is still only a single PVSS system, i.e. one Event Manager. As mentioned earlier, communication between managers is realised by TCP/IP, which means they do not have to run all on the same computer. Managers do not even have to run on the same kind of operating system. As long as a manager program can be compiled and the core PVSS Application Programming Interface is available for a certain platform, a manager can run on a completely different platform than the rest of the system.

Figure A.4 shows an example for a Scattered System.

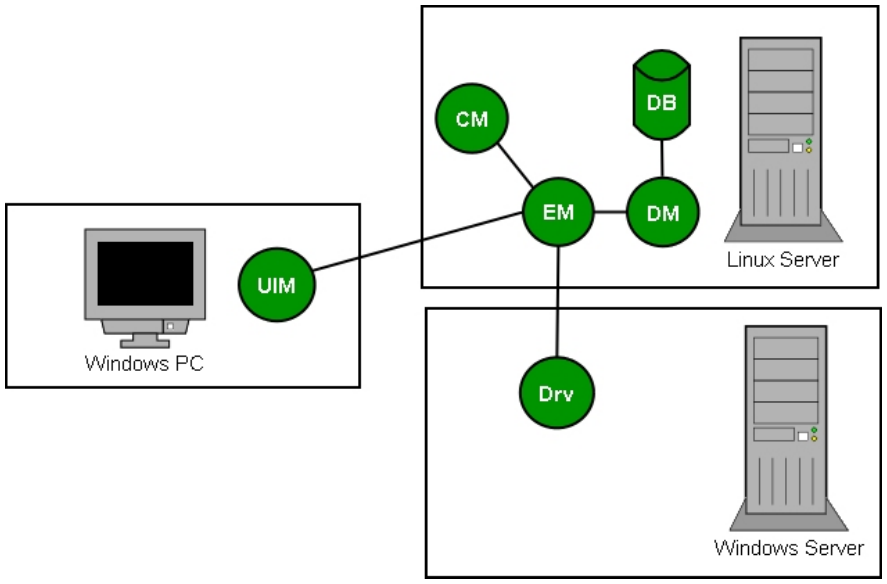


Figure A.4: *It has been determined, that the Linux implementation of PVSS performs better and faster, than its windows counterpart. Certain drivers, like OPC, or GUI options, like ActiveX, are only available on windows platforms though. The solution is to run the core components of the PVSS system (EM, DM, CM) on a Linux server, while running the OPC driver and User Interface Manager on Windows machines.*

Distributed (single) System

While a scattered system consists only of one PVSS system, a distributed system is the combination of multiple PVSS systems into a larger unit. Special *Distribution Managers* are used to build point to point connections between two systems via TCP/IP. The Distribution Managers will forward subscription requests to process variables between the two systems. If one of the subscribed process variable changes, the Event Manager will notify the Distribution Manager which will forward the change to the other PVSS system.

The important information here is that only subscribed data is exchanged between the two systems. This can be used to share load and also allows a kind of encapsulation of information. Figure A.5 shows the ECS scheme for the Outer Tracker front-end electronics. Two independent PVSS systems are used to manage the approximately 1100 electronic devices on each side of the detector. A third system is used as a supervisory system for the two systems connected to the hardware. If a hardware component on side *A* changes its state, only the Event Manager on side *A* will be notified of the change by the drivers.

If the superior system is interested in that particular component, the Distribution Manager will forward the information to the upper system. On the other hand, *A* and *C* side do not know anything about each other because they are not directly connected. The Event Manager on side *A* doesn't need to concern itself with events on the *C*-side, splitting the load on the two systems in half.

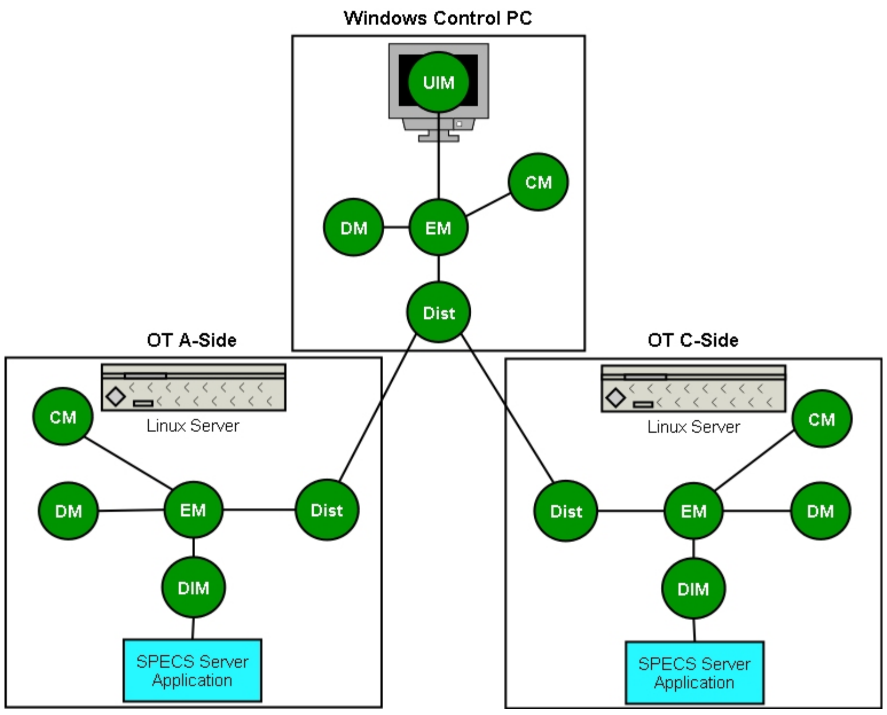


Figure A.5: *Outer Tracker electronics control scheme. The detector is divided into Cryo (C)-Side and A-Side. Each side controls three SPECS-Master PCI cards which are connected to approximately 1100 devices per side. The System has been disconnected from the rest of the LHCb hierarchy and is connected to a Windows control PC instead.*

A.2.7 Data representation in PVSS

So far only the data processing logic has been presented. This section will deal with the representation of data objects in PVSS.

PVSS organises its data in so called *data points* (DPs). A data point is a concrete instance of a *data point type* (DPT). Data point types and data points are very similar to Classes and their instances in Object Oriented Programming.

The reason for this structure is to allow the grouping of process variables into logical units that belong together. Process variables are saved in so called Data Point Elements (DPE)s. Data Point Elements can be *floats*, *integers*, *strings* and other primitive data structures known from common programming languages.

PVSS also allows the use of complete Data Point Types as Data Point Elements. This can be used to not only group variables together, but also arrange them in hierarchies for better clarity[5].

To revisit the volt meter example from above, a more generalised instrument could be used to measure voltage, current and phase shift. This instrument type would be represented by a Data Point Type “Multi Meter” that comprises the data point elements: voltage, current and phase shift. Each instrument in the setup, which should be monitored, would then be assigned to a Data Point instance of the instrument’s Data Point Type.

Finally, each Data Point and Data Point Element can be assigned a so called *config*. Configs are used as switches to:

- assign a certain DP to a concrete hardware device via a driver.
- signal that a certain value should be archived and how it should be archived.
- connect them to other variables. I.e. if a variable changes, a function calculates and updates the value of a different variable.
- set alert limits for variables.
- etc.

[5]This principle is similar to the concept of “composition” in Object Oriented Design.

Appendix B

Acronyms

API	Application Programming Interface
ASDBLR	Amplifier Shaper Discriminator with Baseline Restoration
CERN	Conseil Européen de la Recherche Nucléaire
CRack	Commissioning Rack
DAQ	Data Acquisition
DIM	Distributed Information Management System
ECS	Experiment Control System
FIFO	First-In-First-Out
FSM	Finite State Machine
FPGA	Field Programmable Gate Array
GOL	Giga bit Optical Link
HEP	High Energy Physics
JCOP	Joint COntrols Project
LEP	Large Electron-Positron collider
LHC	Large Hadron Collider
LHCb	LHC beauty Experiment
LVDS	Low Voltage Differential Signal
MEP	Multi Event Package

MSB	Most Significant Bit
NIM	Nuclear Instrumentation Module
ODIN	Readout Supervisor
OTIS	Outer-Tracker Time Information System
PCI	Peripheral Component Interconnect
PVSS	Prozess Visualisierungs und Steuerungs System
RMS	Root Mean Square
SCADA	Supervisory Control and Data Acquisition
SCL	Serial Clock Line
SCSI	Small Computer System Interface
SDA	Serial Data
SPECS	Serial Protocol for the Experiment Control System
TDC	Time to Digital Converter
TELL1	Trigger ELectronics and Level 1
TFC	Timing and Fast Control
TTCrx	TTC receiver chip

Bibliography

[1] *"'The LHCb Experiment"'*, K.A.Gerorge, on behalf of the LHCb Collaboration,
Czechoslovak Journal of Physics, Vol.53 (2003),Suppl.A

[2] Richard Jacobsson,
"Controlling Electronics Boards with PVSS, ICALEPCS 2005"

[3] Guido Haefeli, Aurelio Bay, Federica Legger, Laurent Locatelli, Jorgen Christiansen, Dirk Wiedner.
"Specification for a common read out board for LHCb",
Version 3.0, LHCb 2003-007 IPHE 2003-02 September 2, 2003

[4] Albert Zwart,
"Control-Box for the Outer Tracker Detector",
Version 2

[5] Dominique Breton, Daniel Charlet,
"SPECS: the Serial Protocol for the Experiment Control System of LHCb"
Version 2.0, LHCb DAQ 2003-004

[6] P. Moreira, T. Toifl, A. Kluge, G. Cervelli, A. Marchioro, and J. Christiansen
"GOL Reference Manual, Gigabit Optical Link Transmitter manual",
CERN - EP/MIC, Geneva Switzerland March 2001 Version 0.1.

[7] Dirk Wieder,
"Aufbau der Ausleseelektronik für das äußere Spurkammersystem des LHCb-Detektors"

[8] Jan Knopf,
*"Aufbau eines Auslesesystems
für die Äußeren Spurkammern
des LHCb-Detektors"*,
Physikalisches Institut Heidelberg 2004

[9] Ralf Muckerheide, *"Entwicklung eines Serientests für den TDC-Auslesechip der LHCb Spurkammern"*

[10] Philips Semiconducters,
“The I^{2}C-bus specification”,
version 2.1, January 2000

[11] Richard Herveille,
“I^{2}C-Master Core Specification Rev 0.9, www.opencores.org”

[12] Richard Herveille,
“WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores”

[13] Altera Corporation.
“Stratix PCI Developement Board Data Sheet”,
Version 2.0, September 2003

[14] Altera Corporation.
“PCI Compiler Users Guide”

[15] Stuart A. Boyer,
“SCADA: Supervisory Control and Data Acquisition”

[16] http://itcobe.web.cern.ch/itcobe/Services/Pvss/whatAreScadaAndPvss.html

[17] JCOP Architeture Group,
“JCOP Glossary”

[18] http://dim.web.cern.ch/dim/dim_intro.html

[19] Z. Guzik and Richard Jacobsson,
“LHCb Readout Supervisor ’ODIN’ - Technical reference”

[20] Harald Deppe, Uwe Stange , Ulrich Trunk, Ulrich Uwer
Physikalisches Institut University at Heidelberg
“The OTIS Reference Manual”,
Version 1.1 , 02.02.2004.

[21] Jorgen Christiansen et al.,
*“TTCrx Reference Manual,
A Timing, Trigger and Control Receiver ASIC for LHC Detectors”*

[22] U. Uwer, A. Zwart
“Test Pulse System for the LHCb Outer Tracker Detector”

[23] Study of the Global performance of an LHCb OT Front-End Electronics Prototype, LHCB-2004-120

[24] Noise Studies with the LHCb Outer Tracker ASDBLR Board, LHCB-2004-117

[25] Signal Output Uniformity of the ASDBLR - Nikhef March 31, 2005

Danksagung

An dieser Stelle möchte Ich meinen Dank gegenüber den Personen zum Ausdruck bringen, die zum gelingen dieser Arbeit beigetragen haben.

An erster Stelle gebührt mein ausdrücklicher Dank Herrn Professor Ulrich Uwer für die Möglichkeit, diese Arbeit durchzuführen, sowie für die gute Betreuung und geleisteten Hilfestellungen.

Ganz besonderer Dank gilt Jan Knopf und Dirk Wiedner, die mir bei allen Fragen und Problemen mit Soft- und Hardware immer kompetente Hilfe geleistet haben und daß sie mir die zahlreichen 17 Bit Erlebnisse die Ich im Laufe der Zeit verursacht habe nicht übel genommen haben.

Desweiteren möchte Ich mich bei Clara Gaspar für die Unterstützung in Sachen PVSS und SPECS bedanken. Möge sie mir die vielen Bugreports und Feature Requests verzeihen.

Weiterhin möchte Ich mich bei allen Mitgliedern der HE Gruppe für die gute Laune und die zahlreichen Süßigkeiten bedanken.

Mein Dank gilt auch den Kollegen vom NIKHEF in Amsterdam, am CERN und an der Universität Dortmund.

Bedanken möchte Ich mich auch bei den Korrekturlesern, Gregor Seidel, Manuel Schiller, Joshua Fisher, Marc Deissenroth und Johannes Albrecht

Ganz besonderer Dank gebührt Kerstin Bauer, die mir vor allem in der letzten Phase der Arbeit beigestanden und mich immer unterstützt hat, vielen Dank.

Zu guter Letzt möchte Ich meinen Eltern Sonja und Hans Schwemmer danken, die immer an mich geglaubt und mir dieses Studium ermöglicht haben.

Heidelberg den 20.3.2007

Erklärung

Ich versichere, dass ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 20. März 2007