# Contributions to noise in the data readout for Trigger Tracker in the LHCb Experiment

**Ueli Bieler**

ETH Zürich

**Diploma Thesis**

February 2007

carried out at the

**Physics Institute of the University of Zürich**

under supervision of

**Prof. Dr. Ulrich Straumann**
**Dr. Achim Vollhardt**
**Dr. Johannes Gassner**

## Abstract

This thesis reports the analysis of contributions to noise in the data readout for Trigger Tracker in the LHCb experiment. Measurements have shown that some specific data channels have more noise than the others. This additional contributions to noise cannot be explained by basic electronic noise principles of the detector but by noise sources in the readout chain. The focus is on the channels near the header. Because of a crosstalk effect in the readout electronics the pseudo-digital header affects the close-by analog data channels. Therefore the correlation between the header and the data channels is studied precisely by self-made analysis tools in order to develop an algorithm that cancels the crosstalk contribution to noise. Thanks the algorithm the noise can be reduced efficiently.

# Contents

# Chapter 1

# Introduction

## 1.1 LHCb experiment

The Large Hadron Collider (LHC) at CERN will collide protons at a center-of-mass energy of 14 TeV with a frequency of 40 MHz. It will be the world's most intense source of b-hadrons, producing approximately $10^{12}$ $b\bar{b}$ pairs per year at the LHCb operational luminosity of $2 \cdot 10^{32}$ $cm^{-2}s^{-1}$. The LHCb experiment will benefit from the high b quark production cross section at the LHC in order to study CP violation and other rare phenomena with highest possible precision and in a wide range of B meson decays. The detector provides good particle identification, vertexing and has an efficient and flexible trigger.

Within the Standard Model of particle physics, CP violation can be generated by a single phase in the complex 3x3 Cabbibo-Kobayashi-Maskawa quark mixing matrix (CKM matrix). The existence of CP-violating processes is required in order to generate a matter-antimatter asymmetry as it is observed in the universe today. An improved understanding of the sources of CP violation therefore provides an important link between particle physics and comology.

The level of CP violation that is required to explain the observed matter-antimatter asymmetry in the universe is several orders of magnitude larger than that predicted within the Standard Model. This hints at the need for additional sources of CP violation and therefore for physics beyond the Standard Model. B-mesons are the favored candidates that might unveil new physics and the sources of CP violation. The smallness of the relevant elements in the CKM matrix suppresses the standard tree level decay of b-mesons. This phenomena makes so called penguin and box diagrams accessible to experimental measurements at high statistics. Such diagrams including internal loops represent a very sensitive system for indirect searches for deviations from the Standard Model and for "new physics".

## 1.2 LHCb detector

Key requirements on the LHCb detector are a high reconstruction efficiency of the order of $95\%$ for trajectories of charged particles, good $\pi/K$ separation capability from a few GeV up to approximately 100 GeV, a very good proper time resolution of approximately 40 fs, and highly selective

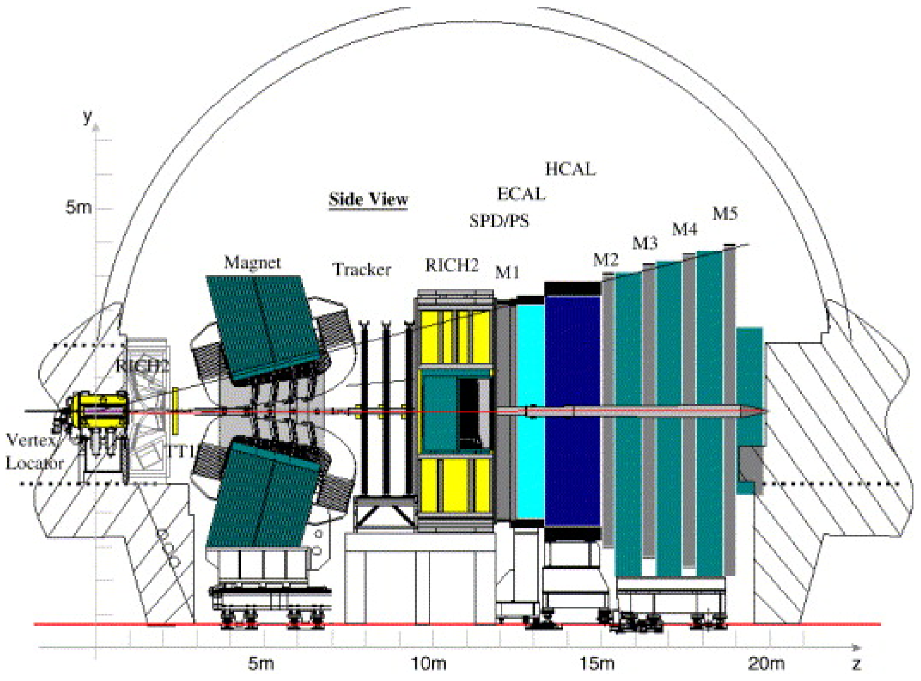


Figure 1.1: Sideview of the LHCb detector

and efficient triggers not only for decay channels involving leptons in the final state but also for purely hadronic final states. Since both b-hadrons are preferentially produced in the same direction and are forward-boosted along the beam-pipe, the detector is not required to have full $4\Pi$ solid-angle coverage. LHCb takes advantage of this by using a wedge-shaped single-arm detector. A side view of the LHCb detector is shown in figure 1.1.

The only detector component to provide coverage upstream of the interaction point is the silicon vertex detector (VELO) that is built around the proton interaction region. It is used to measure the particle trajectories close to the interaction point in order to precisely separate primary and secondary vertices. Directly after the vertex detector, a RICH-1 (a Ring imaging Cherenkov detector) is located. It is used for particle identification of low-momentum tracks.

The main tracking system is placed before and after the dipole magnet and consists of four planar tracking stations: The Trigger Tracker (TT) and the stations T1, T2 and T3. It is used to reconstruct the trajectories of charged particles and to measure their momenta. Following the tracking system is RICH-2. It is used to perform particle identification of high momentum tracks. The information from both RICH detectors has to be correlated with the tracking and momentum information from the main tracking system to provide particle identification.

The electromagnetic and hadronic calorimeters provide measurement of the energy of electrons, photons and hadrons. These measurements are used at trigger level to identify the particles with high transversal moment. The muon system is used to trigger on muons in the events.

In the high-rate hadronic environment of the LHC, triggering is one of the major challenges for the experiment. The LHCb trigger is designed to distinguish events containing B mesons from minimum-bias events through the existence of secondary vertices and the presence of particles with a large momentum.

Informations about the detector design can be found in the LHCb Technical Design Reports [1] and [2].

## 1.3 Silicon detectors

### 1.3.1 Vertex Locator

The vertex locator (VELO) measures the particle trajectories close to the interaction region. The high resolution of the coordinate measurements of the track allows the reconstruction and separation of the primary interaction vertex from the secondary decay vertices of B mesons. To realize this high precision the detector must be placed as close as possible to the interaction point. This is achieved by placing the whole vertex detector inside a secondary vacuum.

The vertex detector provides information at several stages in the trigger. Two dedicated silicon stations, placed at the opposite side of the interaction region as the LHCb spectrometer, are used in the Level-0- trigger to reject events with multiple proton-proton interactions. The remaining VELO detectors are used in the Level-1 trigger to select B events by detecting displaced secondary vertices. The higher trigger levels (2 and 3) use the full vertex detector information to reconstruct and precisely measure a full decay chain.

### 1.3.2 Silicon Tracker

The Silicon Tracker is a large-surface silicon microstrip detector that constitutes an important part of the LHCb tracking system. It uses single-sided silicon strip detectors with a strip pitch of approximately 200 $\mu$m, produced from 6” wafers and arranged into up to 38 cm long readout strips.

The Silicon Tracker consists of two parts: the so-called Trigger Tracker (TT) station is located in between RICH1 and the LHCb dipole magnet. It covers a rectangular area of approximately 140 cm in width and 120 cm in height. Its four detection layers add up to a total sensitive area of approximately 7 $m^2$. The Inner Tracker (IT) covers a cross-shaped area around the LHC beam pipe in tracking stations T1-T3, in between the LHCb dipole magnet and RICH2. The cross extends over approximately 120 cm in width and 40 cm in height. Twelve detection layers in the Inner Tracker add up to a sensitive area of approximately 4.2 $m^2$.

Readout electronics and infrastructure such as High Voltage and Low Voltage distribution and cooling system are a common development for Inner Tracker and Trigger Tracker and have been described in the Inner Tracker Technical Design Report [2].

The Trigger Tracker (TT) fulfills a two-fold purpose. Firstly, it will be used in the Level-1 trigger to assign transverse-momentum information to large-impact parameter tracks. Secondly, it will be used in the offline analysis to reconstruct the trajectories of long-lived neutral particels that

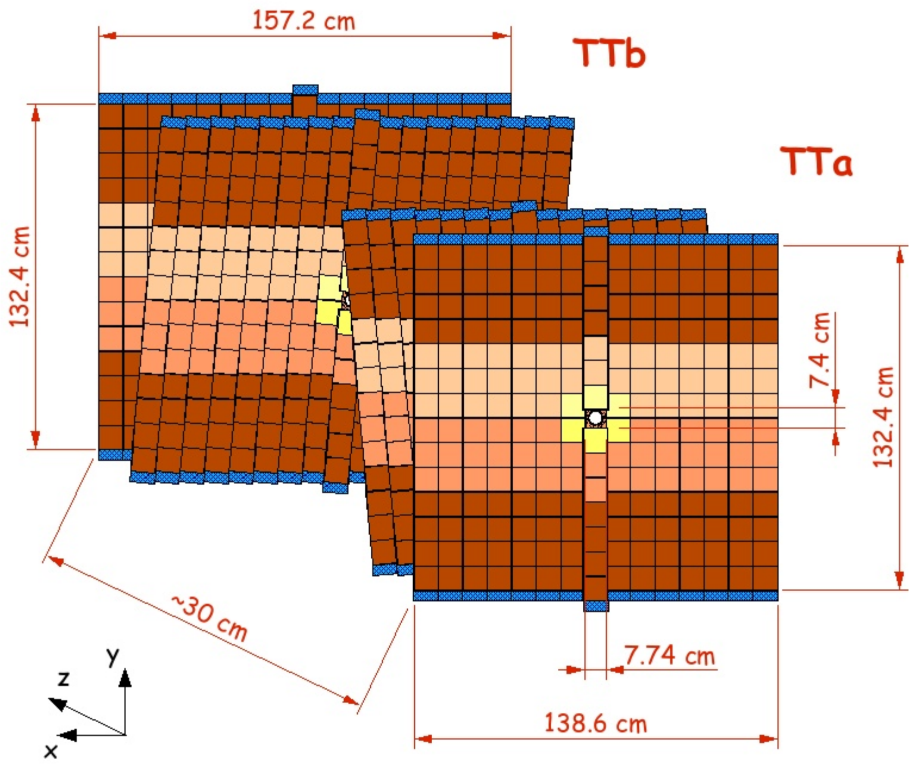


Figure 1.2: Layout of the TT station

decay outside of the fiducial volume of the Vertex Locator and of low-momentum particles that are bent out of the acceptance of the experiment before reaching tracking station T1-T3 [1].

The Trigger Tracker consists of two pairs of 160 cm wide and 130 cm high detection layers(TTa and TTb) with a gap of 23 cm between the pairs. This allows a full 3-dimensional track reconstruction. A layer consists of detector modules constructed from seven silicon sensors. Electronically, each module is split into several readout sectors. There are three regions on a layer: cryo, beam-pipe and access which are numbered 1-3. The center region corresponds to the three innermost columns of the detector where each detector module is segmented into a four, a two and a one sensor readout sector.

The TT station is housed in a single large box (detector box) surrounding the beam pipe. The box which provides electrical and thermal insulation is split in halves that can be retracted for installation and maintenance and for the bake-out of the beam pipe. A key parameter for the operation of the silicon sensors is their temperature. To reduce radiation induced leakage currents and the associated noise to an acceptable level, the sensors have to be operated at a temperature of around 5 °C. The temperature and the humidity inside the box is measured. This permits to control the proper operation of the cooling system and to calculate the dewpoint of the box.

While the silicon sensors and the hybrids are located in the detector box, the digitizing and

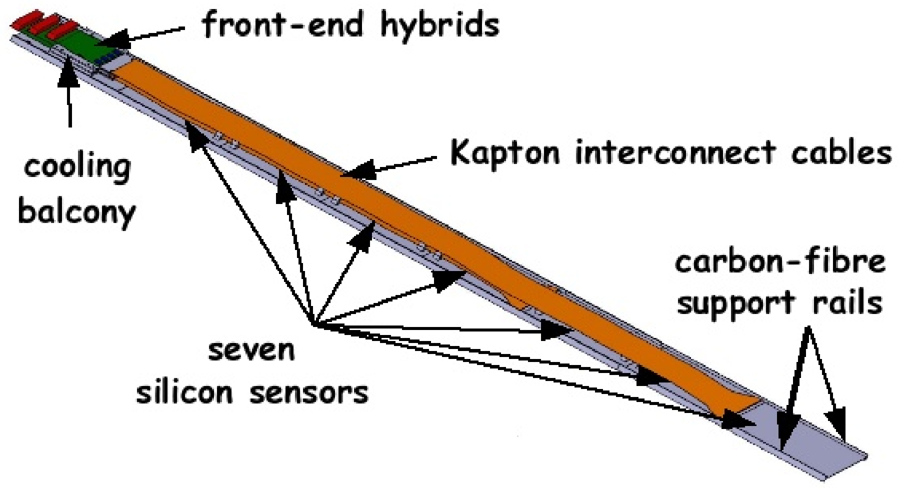


Figure 1.3: View of a half-module for the TT station

control electronics is located in the service boxes, out of the detector acceptance. This reduces the amount of dead material and hence multiple scattering and generation of secondary particles. The geometry description for the LHCb Trigger Tracker can be found at [3].

## 1.4 Silicon Sensors

The silicon sensor uses a reversely biased pn-junction to detect traversing particles. The number of electron-hole pairs depends on the energy of the particle. The relevant signal is the total free charge generated by the particle entering the detector device. Under the influence of an electric field, electrons as well as holes travel to the electrodes, where they give rise to a short pulse whose height is proportional to the observed charge and that can be measured from the electronics attached to the device.

The energy required for production of electron-hole-pairs is very low compared to the energy required for production of paired ions in a gas detector. Consequently, in semiconductor detectors the statistical variation of the pulse height is smaller and the energy resolution is higher. As the electrons travel fast, the time resolution is also very good. Compared with gas ionization detectors, the density of a semiconductor detector is very high, and charged particles of high energy can give off their energy in a semiconductor of relatively small dimensions.

The used silicon sensors are single sided, $p^+$-n, AC-coupled, polysilicon biased silicon strip detectors. The size of each sensor is 94.4x94.6 mm, has 512 strips and a thickness of 500 $\mu m$. A sketch of half-module is shown in figure 1.3. The modules are segmented in half-modules with a 4-3 sensor layout and around the beam pipe with a 4-2-1 readout segmentation.

The sensors are connected to the staggered front end hybrids that provides electrical and mechanical support for the front end chips. All of them are carrying a pitch adapter, four beetle readout chips with 128 channels per chip, blocking capacitors and resistors. Four sensors are directly connected to the lowest readout hybrid and three sensors are connected by the inter-connect cable to the hybrid above. Around the beam pipe four sensors are connected to th lowest hybrid, two sensors to the second one and one sensor to the hybrid on top (4-2-1 readout segmentation). Two half-modules lengthwise assembled form a module and 17 modules together give a layer [4].

## 1.5 Data acquisition

### 1.5.1 The readout chain

The detector modules are equipped with Beetle front-end chips, which sample the detector data at the LHC bunch crossing frequency of 40 MHz and store the analog data for the latency of the Level-0 trigger. On reception of a trigger accept, the data are transmitted via low-mass copper cables to a so-called Service Box, that is located close to the detector in an area of significantly reduced radiation load. The cable length of about 5 metres is a compromise between low radiation and short transmission up to the analogue-digital conversion to minimize any loss of signal quality. The Service Box contains the necessary electronic components for supplying the readout hybrids with power and timing signals. Here, the data are digitized, multiplexed and prepared for optical transmission to the LHCb counting room. The sketch of the readout scheme is shown in figure 1.4. Details can be found in the Technical Design Report [1].

### 1.5.2 Beetle chip

The Beetle 1.3 [5] is a mired-signal front-end chip. It has been developed for the Vertex Locator (VELO) and the Silicon Tracker (ST) data acquisition system of LHCb. Its goal is to collect the charges from 128 channels of a detector at a rate of 40 MHz. It incorporates fast preamplifiers and active shapers, followed by an analog pipeline in which data are stored during the fixed latency of the LHCb Level-0 trigger accept.

Due to the heavy radiation in the LHCb cavern, this chip is made with radiation-hard technology. After collection the 128 analogue levels corresponding to an event are stored in one of the 187 pipelines. In the presence of a Level-0 trigger accept, the data are multiplexed and transmitted via four analog output ports to the Level-1 Off-detector electronics called TELL1 [6].

There are three tasks:

- Amplification and storage of 128 detector channels (25 ns samples).
- Readout of a complete event within 900 ns over analogue links, each analogue link carries the information of 32 detector channels.
- In addition to the detector data an header is transmitted on the analogue links.

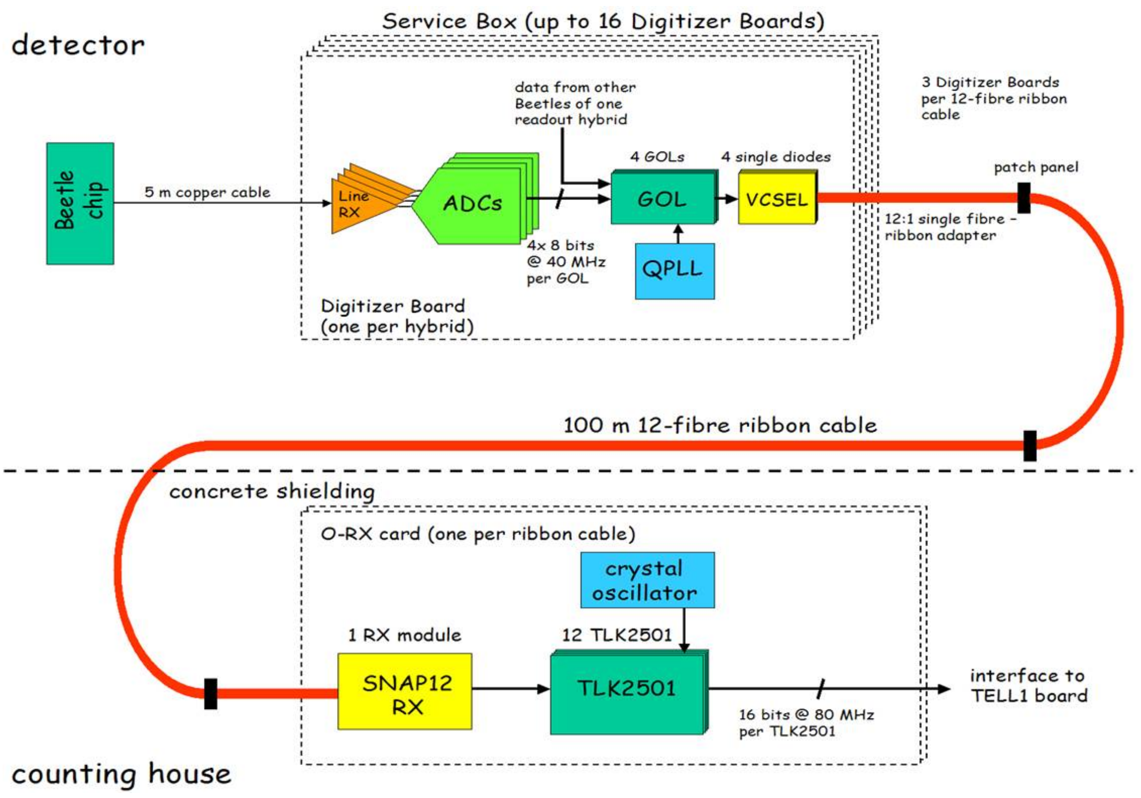

detector
Service Box (up to 16 Digitizer Boards)
Beetle chip
5 m copper cable
Line RX
ADCs
data from other Beetles of one readout hybrid
4 GOLs
GOL
4 single diodes
VCSEL
QPLL
4x 8 bits @ 40 MHz per GOL
Digitizer Board (one per hybrid)
3 Digitizer Boards per 12-fibre ribbon cable
patch panel
12:1 single fibre - ribbon adapter
100 m 12-fibre ribbon cable
concrete shielding
O-RX card (one per ribbon cable)
crystal oscillator
1 RX module
12 TLK2501
SNAP12 RX
TLK2501
16 bits @ 80 MHz per TLK2501
interface to TELL1 board
counting house

Figure 1.4: Sketch of the readout scheme

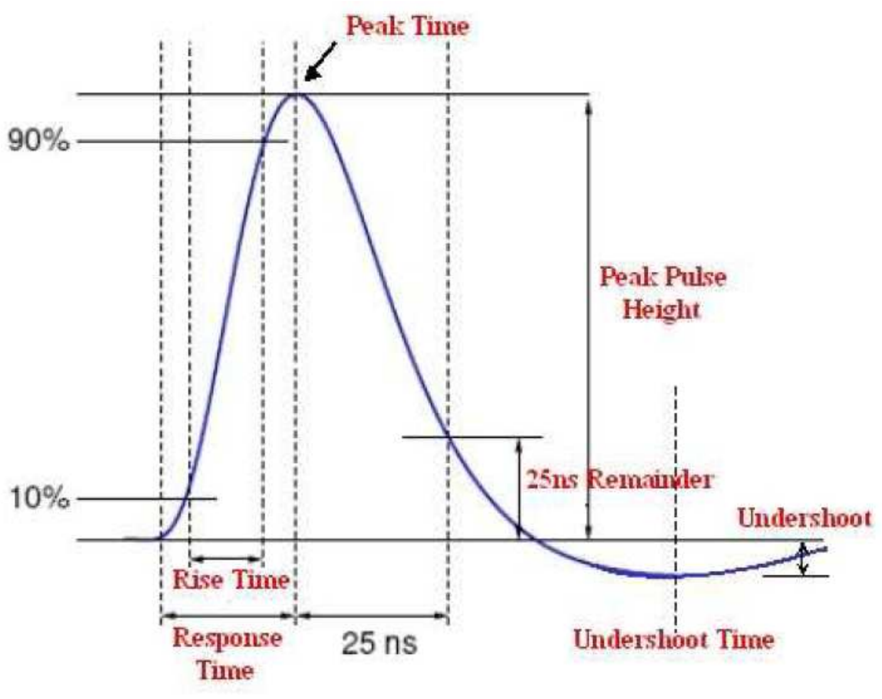


Figure 1.5: The typical analogue pulse of a readout channel.

In order to characterize the events, 16 information pseudo-digital bits are inserted before the analogue signal data stream. A part of this 'header' is the Pipeline Column Number (PCN) which represents the position of the event data in the pipeline. Eight bit are therefore necessary to define the PCN. The remaining 8 bits of the header gives information about the Beetle settings and status. Recapitulating the Beetle 1.3 has tree tasks:

Two modes of data transmission can be selected:

1. 16+128=144 data samples are transmitted over a single port.
2. The 144 data samples are transmitted over four ports. Each port carries 4+32=36 samples. In this way the duration of an event readout is four time less.

The Beetle chip provides an analog as well as a binary output mode. A differential current is transmitted in each case. The front-end output signal (figure 1.5) is a semi-Gaussian pulse which can be characterized by the peaking time, peaking voltage and the remainder, which is the ratio between the signal voltage 25 ns after the peak and the peaking voltage.

### 1.5.3 Service Box

The Service Box includes the line receivers for the Beetle analogue data [1], the 8-bit ADCs [2], the radiation-hard Gigabit Optical Link serializers (GOL) [3] and the VCSEL diodes [4] for coupling the gigabit data stream into an optical fibre [5]. The latter components are all located on the so called 'Digitizer Board'. The GOL chip is a radiation-hard gigabit serializer developed at CERN [7] and is used to control the VCSEL diode and further transmit the digitized and multiplexed data via optical fibers that will be employed to bridge the approximately 100 m to the LHCb counting room. Additional elements of the Service Box are the Control Card and the backplane of the Service Box. While the backplane only provides the power supply and timing distribution for the Digitizer Boards and the Beetle readout hybrids, the Control Card has to provide the timing signals and the correct phase relation between them. An optical readout link connects the 'Service Box' to the Tell1 board. Its purpose is to combine the incoming data from different sources and perform basic data processing like hit finding and clustering. An integrated FIFO memory temporarily stores the event data until receipt of a Level1-accept signal, which releases the data for transmission over a gigabit ethernet interface to the DAQ Processor farm.

### 1.5.4 TELL1 readout board

The TELL1 [6] board will stand in a radiation free area at about 100 m from the detector and receives analog data form the frontend electronics that is situated in the radiation area. It is used for event synchronization, buffering during the trigger latency, pre-processing including common mode correction and zero suppression. For the data acquisition, the board is interfacing to standard Gigabit Ethernet network equipment providing up to four Gigabit Ethernet links. TELL1 accepts 24 optical links running at 1.6 Gbit/s and provides for the analogue option 64 8-bit ADC channels sampling at 40MHz. The digitized data is sent to the so called 'Preprocessor-FPGA', where the zero suppression for the L1 trigger and the data storage during L1 latency is performed. The L1 accepted data is read out of the L1 buffer and sent for further processing to the 'SyncLink-FPGA'.

### 1.5.5 Trigger System

The detector produces a total data rate of 40 TByte/s and thus exceeds capabilities of present network architectures and storage media. To reduce this rate, the trigger has the task to select, out of these millions of events, the most interesting 200 per second. LHCb implements a selective four stage trigger system that analyzes the sub-detectors in different trigger stages. Events with B-mesons can be distinguished from other inelastic pp interactions by the presence of secondary vertices and particles with high transverse momentum $p_T$.

The pipeline comprises the following stages:

[1] AD8129 datasheet, www.analog.com

[2] TSA0801 datasheet, www.st.com

[3] GOL Homepage, http://proj-gol.web.cern.ch/proj-gol/

[4] VCSEL datasheet, www.ulm-photonics.com

[5] MTP 12-fibre ribbon cable, www.fonetworks.com

- The Level-0 comprises three high $p_T$ triggers, which operate on muons, electrons, and hadrons. Additionally a pile-up veto is issued, which suppresses events with more than one pp interaction. The Level-0 operates on the bunch-crossing frequency of 40.08 MHz. The maximum Level-0 output rate is limited by the Level-0 derandomizer and accounts to 1.11 MHz. The latency of the Level-0 is set to be 4.0 $\mu$s.

- The Level-1 is based on the VELO, TT, T1, T2, T3 and the summary information of L0. It selects events, which have a secondary vertex. The Level 1 operates at mean Level-0 output rate of 1 MHz and using VELO, L0 and TT data reduces the event rate to 40 kHz. Then using T1, T2, T3 data Level-1 trigger achieves another suppression factor of 8 with the latency of about 50 ms. The trigger algorithm is implemented on a commodity CPU farm.

The general architecture of each level is the following: an input stage followed by a buffer defined by the trigger latency, an interface to receive the trigger decision, and an output stage with a buffer to derandomize the data transmission to the next trigger level. The derandomization is needed to adjust to the input bandwidth of the subsequent level.

# Chapter 2

# Definition of the project

In this chapter the initial situation and the the goals of the project are described.

## 2.1 Noise studies

Measurements with silicon ladders in the laser teststand showed that the ADC values of some channels have an exceptional large standard deviation. These channels are found systematically close to the so-called header. The header characterizes the event and is composed of 16 pseudo-digital bits (PDB) that are inserted to the analog signal data stream from the Beetle (beetle channels). This leads to the assumption that the header influences the nearby beetle channels, in particular the channels 0,32, 64 and 96. A challenge of the project is to analyse the correlation between header and beetle channels and to evaluate the noise contribution due to the header.

The reason of the correlation seems to be the so-called crosstalk. This term means the undesired capacitive, inductive, or conductive coupling from one circuit or channel to another. Usually the coupling is capacitive, and to the nearest neighbor, but other forms of coupling and effects on signal further away are sometimes important, especially in analog designs.

A goal of the project is also the localization of the noise source. Crosstalk can occur at different positions of the read-out chain. Depending on the location of the noise source it's possible to make a hardware correction or not. Otherwise the data have to be corrected with software.

## 2.2 Software corrections

The main goal of the project is the development of an algorithm that cancels the crosstalk contribution due the header. Then the algorithm has to be inserted in the software that process the raw data.

# Chapter 3

# Noise studies

In this chapter the different noise sources are studied. The channel noise has contributions from the detector and from readout electronics. The measured noise is the quadratic sum of all contributing terms. In particular the noise in channels next to the header is very high and has to be estimated.

## 3.1 Noise of silicon strip detectors

### 3.1.1 Basic electronic noise principles

The noise for silicon strip detectors connected to the Beetle preamplifiers consists of thermal noise and shot noise. Shot noise in electronic devices consists of random fluctuations of the electric current in an electrical conductor, which are caused by the fact that the current is carried by discrete charges. This occurs not only in p-n junctions but also in any conductor. The variations are well approximated by a Gaussian distribution and are independent of the frequency (white noise). Shot noise is to be distinguished from current fluctuations in equilibrium, which happen without any applied voltage and without any average current flowing. These equilibrium current fluctuations are known as Johnson noise or thermal noise and are also independent of the frequency.

In the LHCb experiment there are used charge-sensitive amplifiers in order to measure particles with a high rate. They produce a short pulse whose height is proportional to the observed charge in the detector (The shaping time of the readout amplifier is comparable with the 25 ns that will be between the bunch collisions at the LHC). Noise is generated in the conducting channels of the input field effect transistors (FET) and in all steps of the signal amplification. But due to the amplification the signal becomes larger and the noise to signal ratio smaller. A prediction of the thermal noise and the shot noise of the Beetle amplifier used with long silicon strip detectors can be found in [8] and [9]. The noise performance for different strip length was studied in [10].

The channel noise has contributions from the detector (such as the leakage current) and from the read-out electronics.The term coming from the readoutchip is a constant dependent on the chip, plus a term dependent on the capacitance at the input of the amplifier. The measured noise is the quadratic sum of all contributing terms. It is Gaussian distributed around zero. It is uncorrelated between events, and cannot be eliminated. Some channels have a very high noise, which may be due to defects in the read-out chip or to detector faults.

The voltage of all channels of a group may be shifted by a positive or negative amplitude which is nearly the same for all channels of the same strip length on a chip, i.e., it is strongly correlated for groups of channels, and therefore can be calculated and corrected. This shift may also contain a slope, but this can be also corrected. This Common Mode (CM) effect is uncorrelated between events, and obeys a Gaussian distribution. One important contribution to this CM comes from the pickup of external highfrequency signals by the detector and the chips.

### 3.1.2 Noise performance due to radiation damage

An important parameter to characterize silicon-strip detectors is the signal-to-noise ratio (S/N). The signal scales linearly with the detector thickness, while the internal serial noise of the Beetle front-end amplifier increases with the capacitive load and with shorter shaping times. Due to radiation damage an enhancement of the strip capacitance could occur [11]. The estimates show that even at room temperature a sufficient signal-to-noise ratio, higher than 11, can be expected for the hottest regions around the beam pipe. Simulations of the expected radiation dose for the Silicon Tracker can be found in [12]. The impact of this radiation damage on the performance of the Silicon Tracker is studied in [13].

## 3.2 Noise from readout chain

There are different contributions to the common and random noise of the system. The measured ADC value contains various contributions due the read out electronics: For example there are effects of the pipeline non-uniformity. Though the front-end Beetle has been optimized for noise performance, it contributes also to the noise.

### 3.2.1 Pipline non-uniformity

The Beetle contains a pipeline composed of 128 x 187 cells which can store up to 187 events waiting for a L0 trigger. If we look at a single channel, the 187 positions in which the sample can be stored can have slightly different capacitance values. This might imply different values of pedestals for the same channel as a function of the position in the pipeline. On top of that, we can have some state machine dependent effects. The position in the pipeline is encoded in the header of the analogue data as the Pipeline Column Number PCN. The pedestals are in practice averaged over the 187 pipelines positions by an algorithm [14]. The fluctuation of the pedestal values in a fixed channel as a function of the PCN bring an additional kind of noise to the analog signal. The noise caused by the pipeline non-uniformity has been studied in detail in [15].

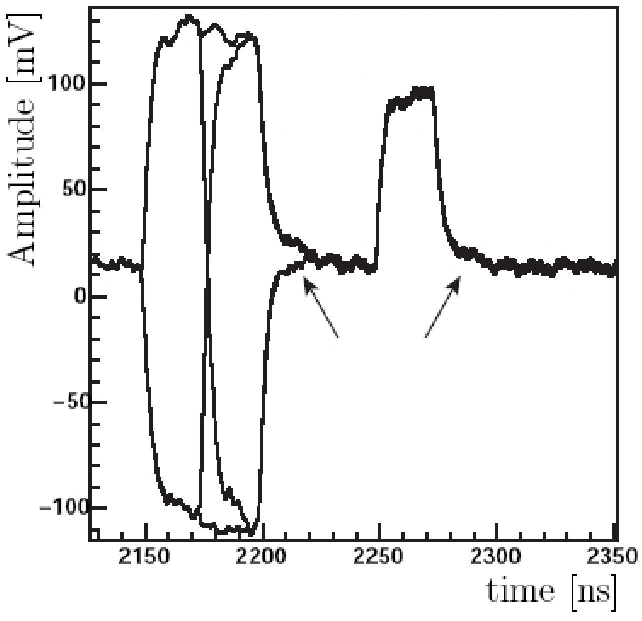


Figure 3.1: Analog signal measured at the Beetle output with a 2 GHz oscilloscope. A charge is injected in the third channel.

### 3.2.2 Digitizer board

### 3.2.3 Crosstalk

An internal crosstalk effect is already present at the level of the Beetle. This was shown for example in [15] by the LHCb group at EPFL in Lausanne. The analog signal was measured directly at the output of the Beetle with a oscilloscope and a tail was seen following each signal, both after the header and after a test signal, that was produced by a injected charge. This is shown in figure 3.1. The signal shape depends on the frequency compensation network, the Beetle internal characteristic and on the line. The measured crosstalk depends on the ADC sampling phase which determines at which point the signal shape is sampled. The main question is, how much one channel affects another and if crosstalk only occurs from one channel to his neighbor or also to the second and third channels. If the amplitude of a pulse propagates in several next neighbors, the correction will become more complicated.

In [15] the crosstalk was evaluated. But in contrast to the Trigger Tracker the Vertex Locater applies a cooper cable from the Service Box to the LHCb counting house. Then the 100 m transmission line contributes a lot to crosstalk noise. Therefore tests have been made do select the best cable type [15]. Among other things the cable have to be well protected against crosstalk.

In the Trigger Tracker especially the high noise in the channels near the header plays an important role. The following chapters will focus on the crosstalk effect in the channels next to the header.

# Chapter 4

# Noise measurement data

The results of measurements in the laser teststand on prototype ladders and bare hybrids are showed in this chapter.

## 4.1 Laser teststand

### 4.1.1 Setup

The tests were made in the laser teststand at the Physics Institute of the University of Zürich. The test-layers or -hybrids are mounted in a freezer that provides stable temperature inside the box. The documentation of the set-up can be found on TT Construction Web [1].

The readout mode of the chips used was 128 analog channels multiplexed onto four ports. The output signals from the Beetle chips are connected via a 50 cm long Kapton flex cable to a PCB board. On this board the signals are multiplexed and transfered via a 5 m long twisted pair CAT6-cable out of the box. In the other direction the differential control signals (clock, trigger, reset) are transmitted to the Beetle hybrid. The PCB also contains connectors for low voltage which are supplied from power supplies outside the freezer. The low voltage is needed for operation of the Beetle chip.

Outside the box the CAT6 cable is connected to an interface which de-multiplexes and amplifies the signals of the four Beetle chips for the following digitization stage. This is carried out by 8-bit FADCs that are located on an ODE prototype board [16]. The ODE board is read out via the VME bus to a PC running a LabVIEW program.

To process and analyse the data a modified version of the software developed for the ST test-beam was used. Details on the procedure used for determining pedestals and common mode noise can be found in [17].

There can be selected two modes of data transmission. The data can be transmitted over a single port or over four ports. In this measurements the four port mode was used. Then the 16 header bits are distributed to the four ports.

[1] see http://lhcb.physik.unizh.ch/tt/laser/lasersetupdoc.php

### 4.1.2 Noise measurement results of a detector module

The first measurement was made with a silicon ladder as shown in figure 1.3. The module TT62M consists of seven silicon sensors that are electronically organized into two readout sectors. The 512 silicon micro-strips are prorated to the four readout Beetles.

The readout hybrid was fixed on a water cooled copper block (water temperature of 10°C). Bias voltage was applied (250 V, current limit 250 $\mu$A). The measurement was made only for Beetle 1 of TT62M. The result is shown in figure 4.1.

The raw noise for each channel is calculated for each channel as the RMS of the ADC distribution of the raw data. The plot in figure 4.1 shows the raw noise (above) and the average ADC value for each channel. The additional contribution to noise in the channels near the header disappears in the noise of the silicon strip detectors. Therefore the following measurements will be done with a bare hybrid.

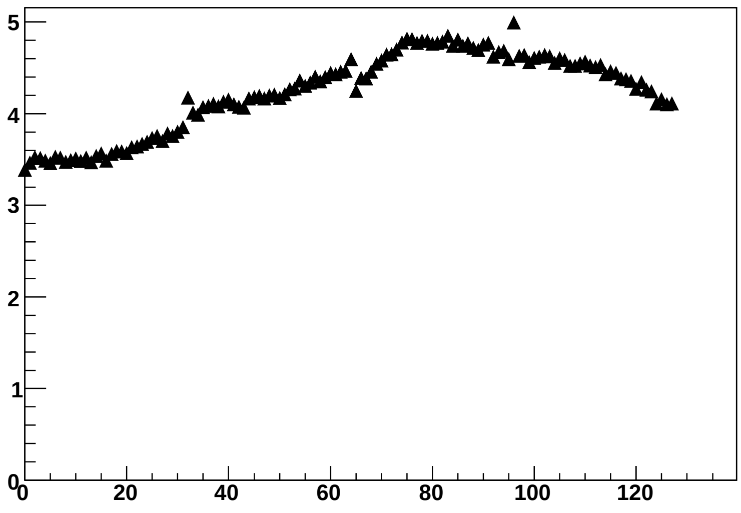
### 4.1.3 Noise measurement results of a bare hybrid

There were made four measurements with the Beetle 1 of a bare hybrid, room temperature, without bias voltage. In figure 4.2 the result of one of the four measurements is shown. The number of saved events was around 1000. The plots of the four measurements looks very similar. This indicates a high stability of the noise. While the average of raw noise of a whole detector module is up to 5 ADC counts, those of a bare hybrid is between one and two ADC counts.

Four measurements were made to compare the raw noise of the four beetles that are placed on the bare hybrid L11. The conditions were the same as in the measurements before. The figures 4.2, 4.3, 4.4 and 4.5 shows the raw noise of the Bettles 1, 2, 3 and 4. It's interesting that the plots of Beetle 1 and Beetle 4 are similar, but they are different from the plots of Beetle 2 and Beetle 3.

## 4.2 Burn-in station

The measurement results of the Burn-in station are different from the results of the laser teststand. The noise of the channels 0, 32, 64 and 96 cannot be observed because the header is intentionally frozen for this reason. But there was observed a lot of noise unsteady distributed to the different channels. The noise source was anywhere on the digitizer board, that wasn't used in the laser teststand. The result of noise measurements on different spots was, that the input of the line receiver is clean, but the input of the ADC is noisy (see figure 1.4). The solution was to attach a capacitance that acts as a low-pass filter. Figure 4.6 shows the performance of such a low-pass filter. Because all digitizer boards were already produced, the capacitance had to be interposed belated in front of the ADC input.



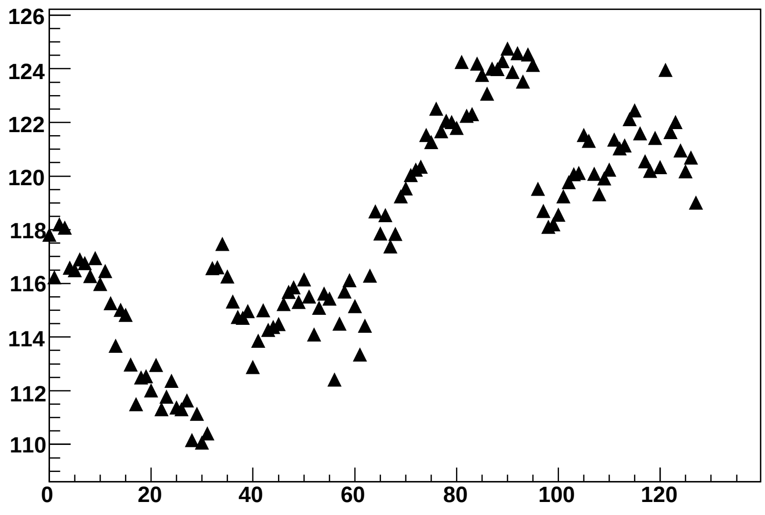


Figure 4.1: The raw noise and the average ADC values for each channel of the ladder TT62M

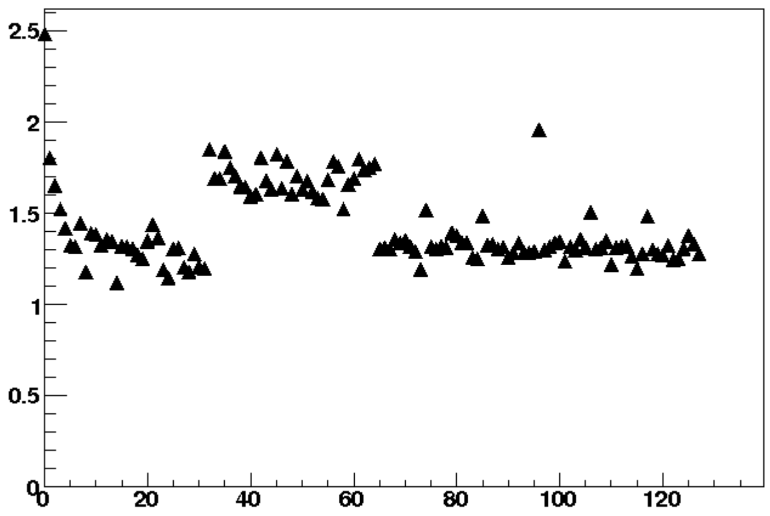


Figure 4.2: Raw noise of the bare hybrid L11, Beetle 1

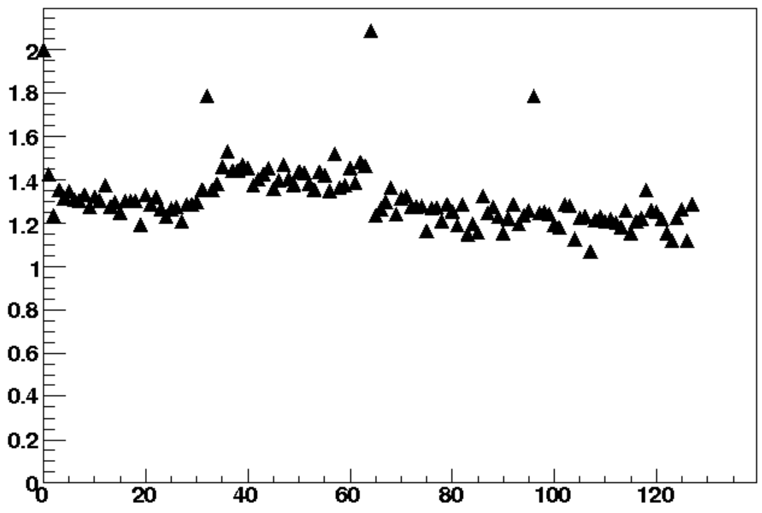


Figure 4.3: Raw noise of the bare hybrid L11, Beetle 2

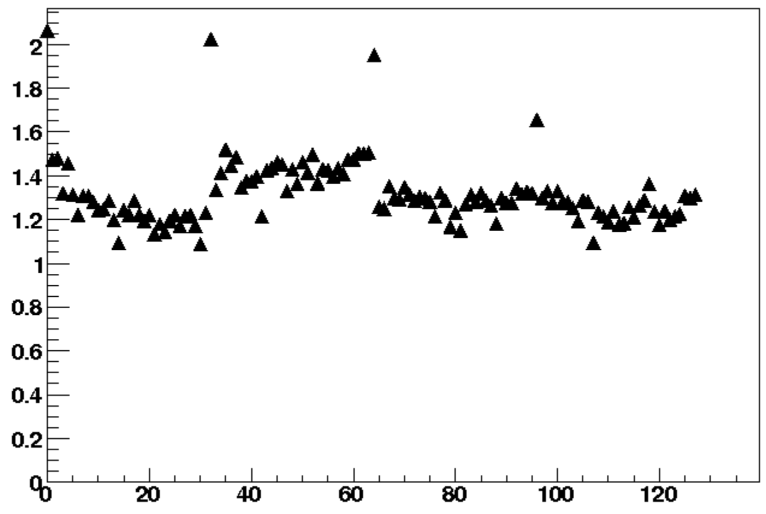


Figure 4.4: Raw noise of the bare hybrid L11, Beetle 3

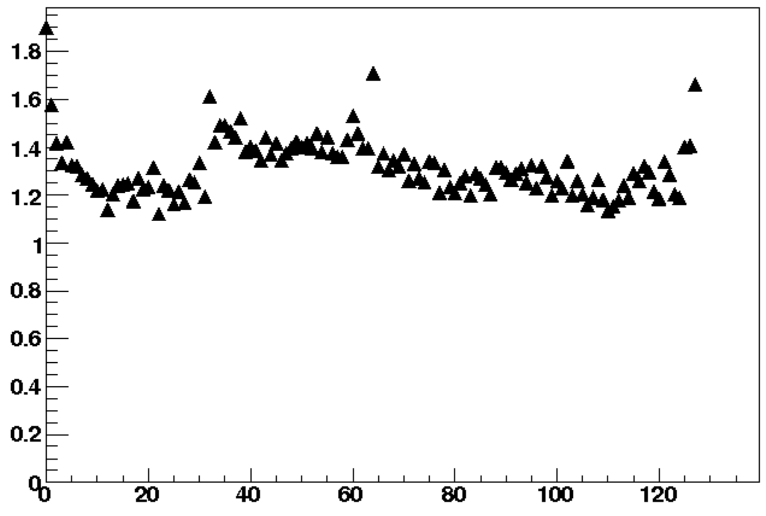


Figure 4.5: Raw noise of the bare hybrid L11, Beetle 4

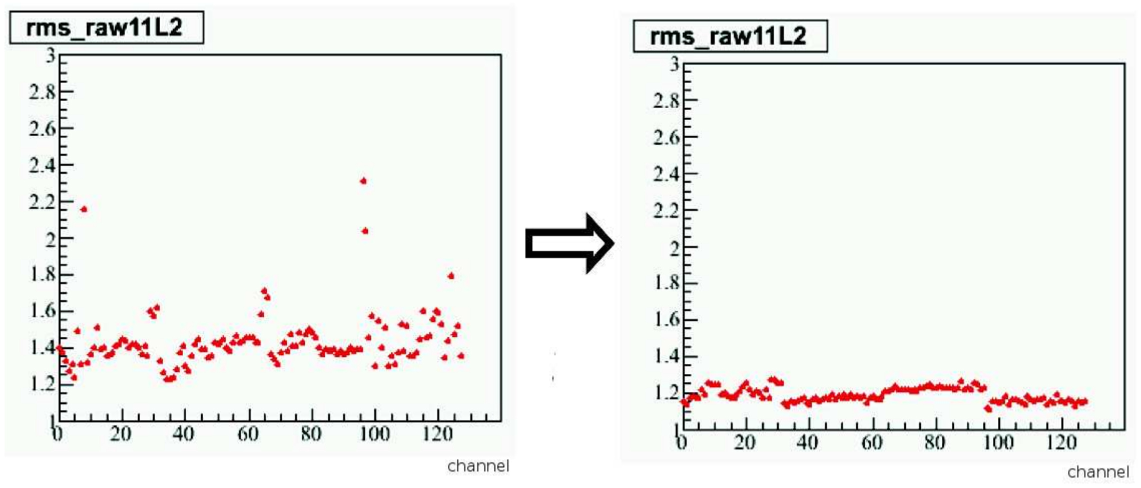


Figure 4.6: A low-pass filter at the ADC input reduce the unsteady noise of the channels 0-127 observably.

# Chapter 5

# Software

A code documentation for the laser teststand set-up can be found on the web [1] including some informations about the raw data processing, a laser analysis package and root macros for analysis. In this chapter only the programs are described that were relevant to study the noise of raw data, especially due to crosstalk.

## 5.1 Raw data processing software

The voltage output of the readout chip for each channel is digitised by the ADC. The term ”raw data” is used for these digitised voltages. These ADC values are stored in a labview file for later analysis. The so-called raw data processing step converts the labview output files to a root Tree. During this step corrections for pedestal and a linear common mode correction in 32 channels is applied. The script ’ProceedPedestalFile’ starts the main program pedana (pedestal analyses). There is also a program that is called newana (new analyses) and that processes files using pre-calculated pedestal runs.

### 5.1.1 Definition of the term ’pedestal’

The measured ADC values contains various contributions: the actual signal from a particle traversing the detector (if present), a DC offset of the channel, called ”pedestal”, the channel noise and a random shift of all voltages on a group of channels called ”common mode” (CM). The signal from a particle comes from its energy loss in the sensor, and is described by a Landau distribution. The pedestal corresponds to a DC offset of the channel output voltage, and has contributions from the chip and from the detector in case that any DC current flows into the amplifier. It has to be calculated for every channel, because significant differences can be found between them. The pedestals are calculated as the mean value of the ADC counts for each channel, excluding the particle hits. Pedestal levels might vary with time as a result of temperature variations or changes in the power supply voltages. Therefore, the pedestal values are continuously updated to allow a correct signal calculation.

[1] http://lhcb.physik.unizh.ch/tt/laser/codedoc.php

### 5.1.2 Script ’proceedPedestalFile’

The script ’proceedPedestalFile’ can be found at

- /disk/panther2/lhcb/progs/

It builds the data file number and starts the pedestal program (pedana). The correct data path has to be entered in the script. The name of the labview output file is the the argument of ’proceed-PedestalFile’. That means that you have to enter in the terminal ’proceedPedestalFile 20070130’ to process the labview file ’20060130.dat’.

### 5.1.3 Main program ’pedana’

The main program pedana analyses the laserteststand measurements and can be found at

- /disk/panther2/lhcb/progs/root2

The program includes different functions that creates runs from raw data, calculates pedestals, writes pedestals, etc. The functions are defined in ’FunctionDef.cpp’. The names of the defined functions are self-explanatory: ’writePedestalFile’, ’readPedestalFile’, ’calculatePedestals’ and ’createRunFromRawData’. The whole information of an event is stored in a variable of the struct ’Event’ that is defined in ’StructDef.h’. The struct ’Event’ includes the integers ’evtnumber’ and ’flag’ and the vectors ’header’, ’data’ and ’channel’. There are 16 header values and 128 beetle, data and channel values - one for each detector strip of a beetle. These constants are defined in ’ConstantDef.h’. The output tree contains 128 raw ADC values, 128 pedestal/common-mode corrected values and 16 header values. The program ’pedana’ was designed by M. Needham in march 2004:

```
//
#include "FunctionDef.h"
#include "jobParameters.h"
#include "constantsDef.h"
#include "StructDef.h"
#include "RunProcessor.h"
#include "TROOT.h"

#include <iostream>

int main(int argc, char *argv[]){

  // jobOptions from argc...
  jobParameters params(argc,argv);

  // create run from raw Data
  std::cout << "creating run from raw buffer" << std::endl;
  run dataRun = run();
  FunctionDef::createRunFromRawData(params.nEventsToRead(), dataRun);

  // calculate pedestals
  std::cout << "calculating pedestals " << std::endl;
  std::vector<float> newPed = FunctionDef::calculatePedestals(dataRun);

  // write pedestal file
  std::cout << "write pedestals " << std::endl;
  FunctionDef::writePedestalFile(params.pedFileName(),newPed);

  // read pedestals
  std::cout << "read pedestals"  << std::endl;
  std::vector<float> pedestals =
          FunctionDef::readPedestalFile(params.pedFileName());

  // new event process object.....
  RunProcessor* rProc = new RunProcessor(params.rootFileName(),
          pedestals,params.outlierCut(), params.iter());

  // run over the data
  std::cout << "create run " << std::endl;
  rProc->execute(dataRun);

  rProc->finalize();
  delete rProc;

  return 0;
}
```

## 5.2 Correction of crosstalk effects

The noise measurements have shown that a correlation between the header and the adjacent channels exists. It seems to be a crosstalk effect that is different from the crosstalk effect that occurs beetween all adjacent channels and it is also larger. In all probability it is being caused by a bug in the Beetle. The effect is especially important because the ADC value of a the pseudo-digital header bit varies much more than the beetle channels. The difference between true and false is approximately 65 ADC. The beetle channels varies only within an interval of 10 ADC counts. Therefore a main goal was the development of an algorithm that cancels the crosstalk contribution due the header.

### 5.2.1 Correction term

We assumed that the error due the header is linear to the difference between the ADC value of the header bit and its close-by analog channel. Furthermore we assumed that the most significant effect is between adjacent channels. So we could reduce the scale of variables. For the correction of channel i we applied the following term:

$$C_i = C_i^X - k \cdot (H_i - C_i^X) \qquad \text{for i= 0, 32, 64 ,96} \tag{5.1}$$

$C_i^X$ is the measured and $C_i$ is the corrected ADC value of the analog channel. k is the so-called crosstalk factor. $H_i$ is the pseudo-digital header bit that is placed in front of the channel i. The term has to be applied to each event separately what can be done by an algorithm that is inserted in the software that process the raw data. In the case of channel 0 the header bit is the 'least significant bit' (LSB) of pipeline column number. In figure 5.1 is inscribed which header bit belongs to the channels 32, 64 and 96.

### 5.2.2 Beetle output mode

The Beetle readout chip provides different readout modes. Figure 5.1 shows the assignment of the header bits and analog input channels to the output channels in analog readout mode that has been used in the laser teststand measurements. 32 analog channels are multiplexed onto 4 ports with up to 40 MHz. The data transmission is synchronous to the rising edge of the readout clock and takes 900 ns per trigger. The meaning of the various header bits is described in figure 5.2.

### 5.2.3 Analog data acquisition

The Beetle programming and read-out is steered by a PC in Windows XP mode. The labVIEW vi 'ODE readout PCI 4port' is responsible for the data acquisition of the Beetle signals. The raw data can be saved to a LabVIEW file [2]. The approach for data acquisition is described on the TT Construction Web [3]. Now the labVIEW file can be processed by executing the script 'proceed-PedestalFile'. The header information is still available so that an analyse of the contribution to noise in channels near the header can be made.

### 5.2.4 Code of the algorithm

The algorithm is inserted in the function 'createRunFromRawData' that can be found in the sub-program 'FunctionDef'. There are nested program loops, the first loop runs from one event to the next, the second from one channel to the next. This logarithm considers only the crosstalk that occurs between the last header bit and its adjacent analog Beetle channel. After modifying any subprogram of 'pedana' it can be made with gmake. The Makefile can be found on /data/lhcb/progs/root2.

[2]the data files were copied to the data directory /disk/panther2/lhcb/data/

[3]http://lhcb.physik.unizh.ch/tt/laser/lasersetupdoc.php

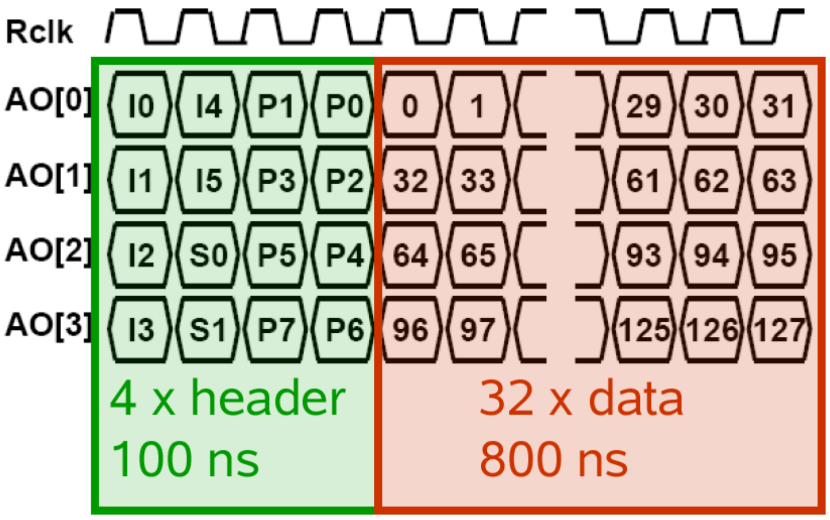
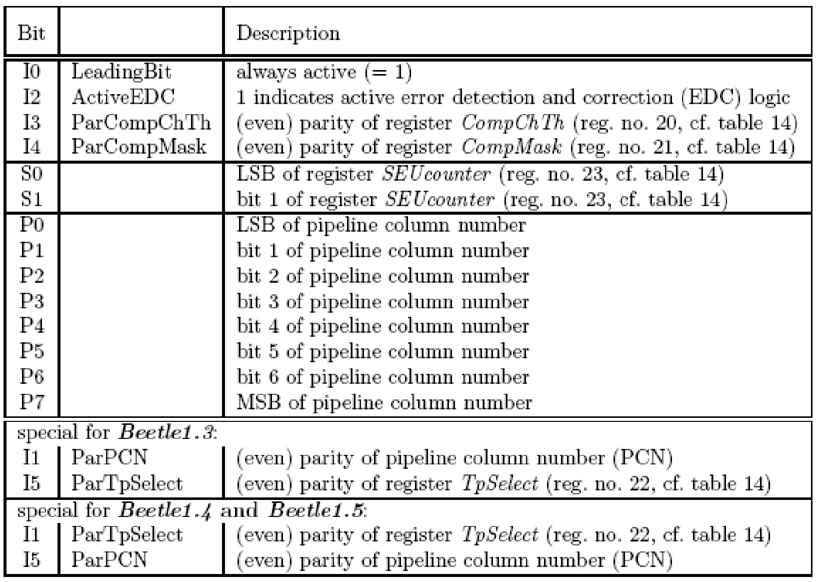


Figure 5.1: The analog readout output mode on four ports [5].



| Bit | | Description |
|---|---|---|
| I0 | LeadingBit | always active (= 1) |
| I2 | ActiveEDC | 1 indicates active error detection and correction (EDC) logic |
| I3 | ParCompChTh | (even) parity of register *CompChTh* (reg. no. 20, cf. table 14) |
| I4 | ParCompMask | (even) parity of register *CompMask* (reg. no. 21, cf. table 14) |
| S0 | | LSB of register *SEUcounter* (reg. no. 23, cf. table 14) |
| S1 | | bit 1 of register *SEUcounter* (reg. no. 23, cf. table 14) |
| P0 | | LSB of pipeline column number |
| P1 | | bit 1 of pipeline column number |
| P2 | | bit 2 of pipeline column number |
| P3 | | bit 3 of pipeline column number |
| P4 | | bit 4 of pipeline column number |
| P5 | | bit 5 of pipeline column number |
| P6 | | bit 6 of pipeline column number |
| P7 | | MSB of pipeline column number |
| special for ***Beetle1.3***: | | |
| I1 | ParPCN | (even) parity of pipeline column number (PCN) |
| I5 | ParTpSelect | (even) parity of register *TpSelect* (reg. no. 22, cf. table 14) |
| special for ***Beetle1.4*** and ***Beetle1.5***: | | |
| I1 | ParTpSelect | (even) parity of register *TpSelect* (reg. no. 22, cf. table 14) |
| I5 | ParPCN | (even) parity of pipeline column number (PCN) |

Figure 5.2: Beetle data formats and definitions of the header bits [5].

```
bool FunctionDef::createRunFromRawData(int nEventsToRead,
                                       run& dataRun){

  // read in the raw data and create a run structure
  dataRun.reserve(6000);
  int counter = 0;
  long nevt;
  bool stop = false;

  // Begin loop over all events
  while((counter < nEventsToRead)&&(stop == false)) {
    Event evt;
    if( fread(&nevt, sizeof(short),  1, stdin) <  1 ) break;

    // Begin loop over ports
    for (int iPort = 0; iPort <  NPORT ; ++iPort){

      if( fread(&(evt.header[iPort* nHeaderInPort]),sizeof(char),
                nHeaderInPort, stdin) <  nHeaderInPort  ) {
        stop = true;
        break;
      }

      if( fread(&(evt.beetle[iPort*nStripsInPort]),sizeof(char),
                nStripsInPort, stdin) < nStripsInPort ){
        stop = true;
        break;
      }
    }    // End loop over ports

    // Marker for new event
    std::cout<< 9999 << std::endl;

    // Event number
    std::cout<< (nevt/256) << std::endl;

    //ADC value of the LSB
    std::cout<< int(evt.header[3]) << std::endl;

    //ADC value of bit 1 of PCN
    std::cout << int(evt.header[2]) << std::endl;

    // Output of PCN (decimal)
    int pcn;
```

```
pcn =
  bool(evt.header[14]>100) *128+
  bool(evt.header[15]>100) *64+
  bool(evt.header[10]>100) *32+
  bool(evt.header[11]>100) *16+
  bool(evt.header[6]>100) *8+
  bool(evt.header[7]>100) *4+
  bool(evt.header[2]>100) *2+
  bool(evt.header[3]>100) *1;

std::cout << pcn << std::endl;


// Algorithm that cancels the crosstalk due to header
// for  the channels 0, 32, 64 and 96

float konst0= 0.0512;     // crosstalk factors for each port
float konst1= 0.0512;
float konst2= 0.0512;
float konst3= 0.0512;

// Begin loop over all channels
for (int j=0; j<128; j++){
  if (j==0){
    evt.beetle[j] = float(evt.beetle[j]) +
        konst0 * (float(evt.header[3])-float(evt.beetle[j]));
        std::cout << int(evt.beetle[j]) << std::endl;
  }

  else if (j==32){
        evt.beetle[j] = float(evt.beetle[j]) +
        konst1 * (float(evt.header[7])-float(evt.beetle[j]));
        std::cout << int(evt.beetle[j]) << std::endl;
  }

  else if (j==64){
        evt.beetle[j] = float(evt.beetle[j]) +
        konst2 * (float(evt.header[11])-float(evt.beetle[j]));
        std::cout << int(evt.beetle[j]) << std::endl;
  }

  else if (j==96){
        evt.beetle[j] = float(evt.beetle[j]) +
        konst3 * (float(evt.header[15])-float(evt.beetle[j]));
```

```
            std::cout << int(evt.beetle[j]) << std::endl;
      }

      else std::cout << int(evt.beetle[j]) << std::endl;
    }  // End loop over all channels, end of algorithm

    evt.evtnumber = nevt;
    evt.flag      = 1;
    dataRun.push_back(evt);
    ++counter;
  }  // End loop over all events

  std::cout<<NPORT<< " port mode, read in " <<counter<<std::endl;

  return true;
}
```

### 5.2.5 Approximation of the crosstalk factor

The crosstalk factors have to be evaluated precisely for the different Beetles and also for the channels 0, 32, 64 and 96 seperately. They can be found by minimizing the difference between the mean of the selected channel in the case of LSB is false and true respectively. For this reaseon the function 'createRunFromRawData' in the subprogram 'FunctionDef' was modified. There is a loop over crosstalk factors between -0.1 and 0.1. The program with the modification of the function 'createRunFromRawData' is saved as 'FunctionDefMeanComp'. In order to start the program, it has to be saved as "FunctionDef.C" and compiled. After using the algorithm the original function can be found in 'FunctionDefOriginal.C'. At the beginnig the channel number and the header number that belongs to it have to be entered manually. The number of the header bit corresponds to the position in figure 5.1. Actually it is possible to observe any channel in subject to any header bit. But in fact there is only a significant correlation between the header and its next neighbours.

The code of the modified function in 'FunctionDefMeanComp' is:

```
bool FunctionDef::createRunFromRawData(int nEventsToRead,
                                       run& dataRun){

  // enter channel number and beetle manually!
  int channelnumber=0;       // 0, 32, 64, 96
  int beetlenumber=3;        // 3, 7, 11, 15

  // declarations for the mean calculations
  float adcsum0[20000];
  float adcsum1[20000];
float sum0[20000];
float sum1[20000];
float konst[20000];

// set zero all components
for (int kk=0; kk<20000; ++kk){
  adcsum0[kk]=0; adcsum1[kk]=0;
  sum0[kk]=0;sum1[kk]=0; konst[kk]=0;
  konst[kk]=float(kk)/10000-1;
}

// read in the raw data and create a run structure
dataRun.reserve(6000);
int counter = 0;
long nevt;
bool stop = false;


// loop over all events--------------------------------------
while((counter < nEventsToRead)&&(stop == false)){
  Event evt;

  if( fread(&nevt, sizeof(short), 1, stdin) < 1 ) break;
  for (int iPort = 0; iPort <  NPORT ; ++iPort){

    if(fread(&(evt.header[iPort*nHeaderInPort]),sizeof(char),
             nHeaderInPort, stdin) <  nHeaderInPort  ) {
      stop = true;
      break;
    }

    if(fread(&(evt.beetle[iPort*nStripsInPort]),sizeof(char),
             nStripsInPort, stdin) < nStripsInPort ){
      stop = true;
      break;
    }
  }

  float lsb[20000];
  float ch0[20000];

  for (int j=0; j<20000; ++j){
    ch0[j]=float(evt.beetle[channelnumber]);
    lsb[j]=float(evt.header[beetlenumber]);
    }

    // correction by different crosstalk factors
    for (int k=0; k<20000;++k){

      ch0[k] -= konst[k] * (lsb[k]-ch0[k]);  // correction term

      if (lsb[k] < 100){
        adcsum0[k] += ch0[k];             // summarize (case 1)
        sum0[k] += 1;
      }

      else{
        adcsum1[k] += ch0[k];             // summarize (case 2)
        sum1[k] += 1;
      }
    }

    evt.evtnumber = nevt;
    evt.flag      = 1;
    dataRun.push_back(evt);
    ++counter;
  } // end loop over all events---------------------------------

  // output of the XT factors and the differences between means
  for (int k=0; k<20000; ++k){
    std::cout<< konst[k] <<  std::endl;
    std::cout<<adcsum0[k]/sum0[k]-adcsum1[k]/sum1[k]<<std::endl;
  }
  return true;
}
```

## 5.3 Creating Histograms with ROOT

The plots of the analysis tools are produced with root. ROOT supports differnet histogram types. For example TH1F is a 1-D histogram with one float per channel. An overview of the histogram classes can be found in [18]. Histograms are created with constructors that have the following form:

```
TH1F *h1 = new TH1F("h1","h1 title",100,0,4.4);
```

The parameters of the TH1F constructor are the name of the histogram, the title, the number of bins, the x minimum, and x maximum. A histogram is typically filled with statements like:

```
h1->Fill(x);
h1->Fill(x,w);
h2->Fill(x,y);
```

The Fill method computes the bin number corresponding to the given x or (x,y) argument and increments this bin by one or by the given wight w. The basic whiteboard on which an object is drawn is called a canvas:

```
TCanvas* tPlot = new TCanvas("name","title",1000,800);
```

The parameters are the name of the canvas, the title, the length and the height. The histogram is drawn with a statement like:

```
h1->Draw()
```

A lot of information about using ROOT can be found in [18]. The making of the plots is left out in all program code in this documentation because of straightforwardness.

# Chapter 6

# Crosstalk analysis package

Several tools have newly developed to analyse the data and study the effect of the algorithm. Some of them are presented in this section. In this chapter the algorithm isn't applied yet. The analysis of the corrected data is to be found in the next chapter.

## 6.1 Distribution of Pipeline Column Number

The first challenge was the correct reading of the data files. The channels of the header had to be assigned to the 8 bits of the binary Pipline Colum Number (PCN). Then the PCN was converted to the decimal system. The distribution of the PCN should be uniformly distributed, because the PCN stands for a point of time that is random. Thanks to the PCN the accurate moment of a signal can be discovered. Figure 6.1 was created by the ROOT program 'PCNdistribution.C' and shows the distribution of the Pipline Column Number (PCN) of 10'000 events. The number of events has to be enterd in the code manually. The data acquisition seems to be correct. The 187 piplines are apparently uniformly distributed.

The header bits are pseudo-digital. That means that the value of the bit is an analog signal, but the ADC values are accumulated in two areas that stands for true or false respectively. The cut between the two cases can be easly done because the two areas are well separated.

The pseudo-analog least significant bit (LSB) of PCN is placed ahead of channel 0 (see figure 5.1) and the two peaks of its ADC distribution have more or less the same number of events as expected. Figure 6.2 was created by the ROOT program 'LSBdistribution.C'. It's interesting that there are two maximums per peak. This can be explained by the influence of the pseudo-digital bit 1 of PCN. Bit 1 is placed ahead of LSB. Depending on whether bit 1 is true or false the ADC value of LSB changes for two or three ADC.

```
{
gStyle->SetOptStat(111 0 );

std::ifstream bla("bla");  // open data file
//......plot for distribution of pipline colum number
TH1F *h = new TH1F("h","Distribution of Pipeline Colum Number"
                  ,200,0,200);

//......definition of the canvas
TCanvas* tPlot = new TCanvas("  ","  ",1000,500);

//......number of events
Int_t n =10000;

//......loop over events
for(Int_t i=1; i<=n; i++)
{
  int delimiter=0;
  bla >> delimiter;       // read delimiter (marker)

  int nevt=0;
  bla >> nevt;            // read event number

  int lsb=0;
  bla >> lsb;             // read lsb
  h2->Fill(lsb);          // fill histogram 2

  int pcn=-3;
  bla >> pcn;             // read pipline colum number
  h->Fill(pcn);           // fill histogram

  //......loop over channels
  for (int j=0; j<128; j++){
    int adc=0;
    bla >> adc;           // read adc values
  }
}

h->Draw();                // draw plot
}
```

In figure 6.3 the ADC value distribution of channel 0 is showed. The plot is drawed by the ROOT program 'CH0distribution.C'. In contrast to the distribution of LSB the ADC values of the 128 analog data channels varies between 110 and 120. Therefore the crosstalk effect from the header to channel 0 is more significant than the crosstalk from one analog data channel to another.

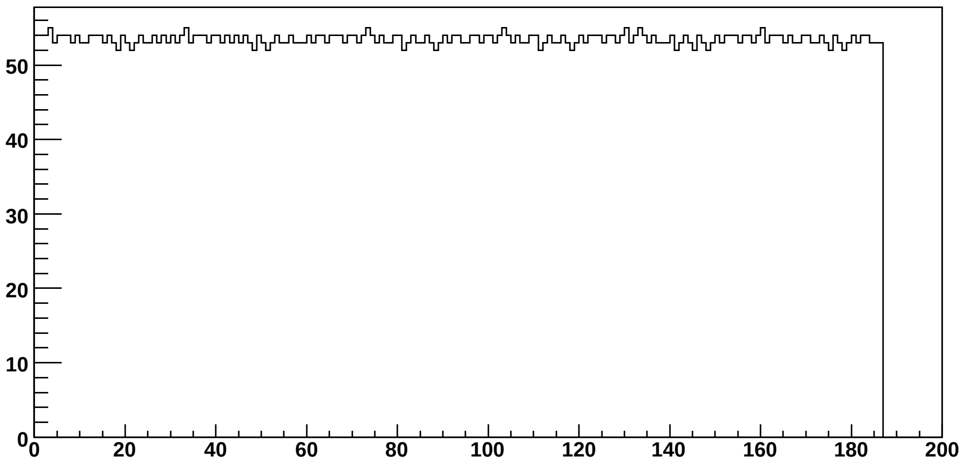


Figure 6.1: The Pipline Column Number is uniformly distributed as expected

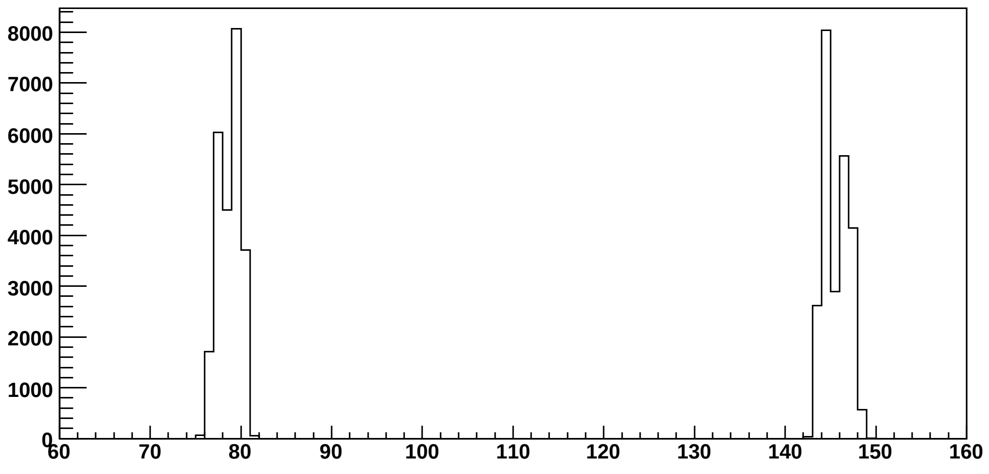


Figure 6.2: The difference between true and false of the pseudo-digital LSB is about 65 ADC.

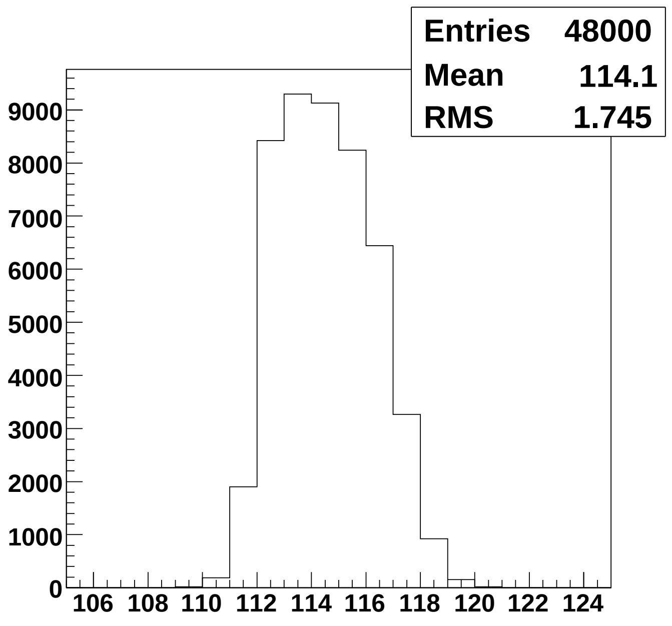


Figure 6.3: The ADC value of channel 0 stays between 110 and 120

## 6.2 Distribution of ADC

The ROOT program 'ADCdistribution.C' shows the distribution of all 128 channels and 48'000 events (Figure 6.4). The splitting up can be explained: The 128 analog data channels are subdivided into four parts. The analog readout takes place on four different ports. This causes different baselines. If we compare the distribution for the channels 0 to 31 and 32 to 63 seperately the splitting up disappears (Figure 6.5).

The average ADC value of each channel sepaprately has been calculated by the ROOT program 'ADCmeans.C'. Figure 6.6 shows that the means are different from channel to channel and especially from port to port. The code of the program 'ADCmeans.C' is:

```
{
gStyle->SetOptStat(111 0 );     // style standard

std::ifstream bla("bla");       // open data file

TH1F *h = new TH1F("h","Mean value for each channel",128,0,128);

TCanvas* tPlot = new TCanvas("  ","  ",1000,500);

//......definitions and zero-setting of all vector components
Int_t n = 48000;                // number of saved events
```

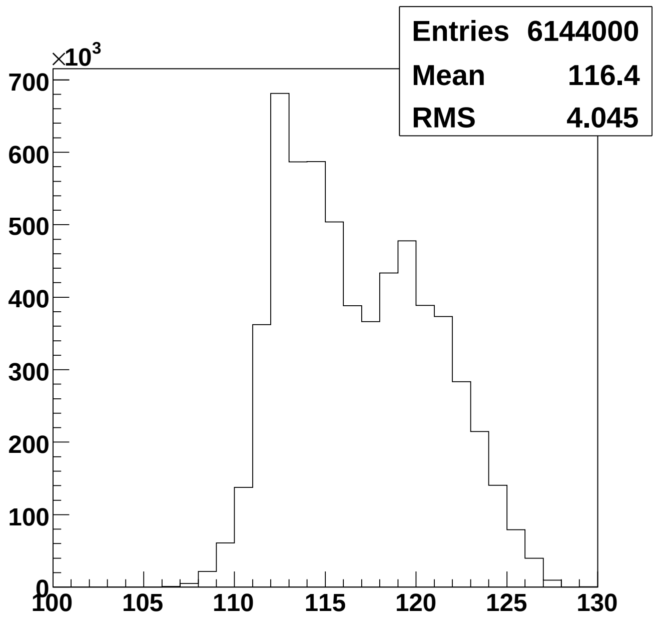


Figure 6.4: The ADC distribution over all channels and all events

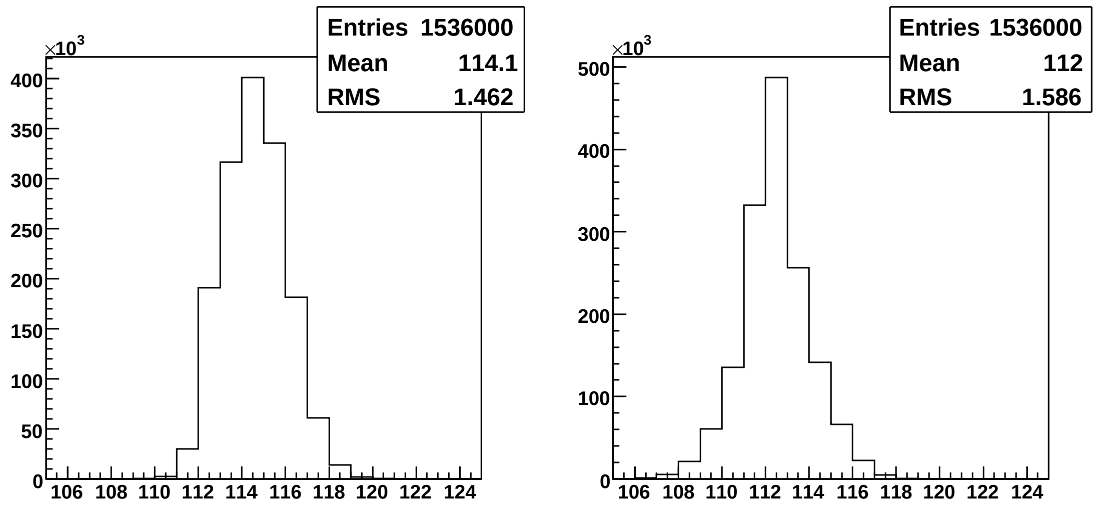


Figure 6.5: Comparision of the ADC distributions of the channels 0-31 and 32-63 seperately.

```
float ch0x[n];                    // ADC values of Channel 0
float lsby[n];                    // ADC values of lsb

float sumar[128];
for(int i=0; i<128; ++i){
  sumar[i]=0;
}

//......loop over events
for(Int_t i=1; i<=n; i++){

  int delimiter=0;
  bla >> delimiter;               // read delimiter (marker)

  int nevt=0;
  bla >> nevt;                    // read event number

  int lsb=0;
  bla >> lsb;                     // read lsb

  int pcn=-3;
  bla >> pcn;                     // read pcn

  //......loop over channels
  for (int j=0; j<128; j++){
    int adc=0;
    bla >> adc;                   // read adc of data channels
    sumar[j] += float(adc);       // add adc value
  }

} //......end loop over events

for(int i=0;i<128;++i){           // fill histogram
  h->Fill(i,sumar[i]/n);
}

h->Draw();                        // draw histogram
}
```

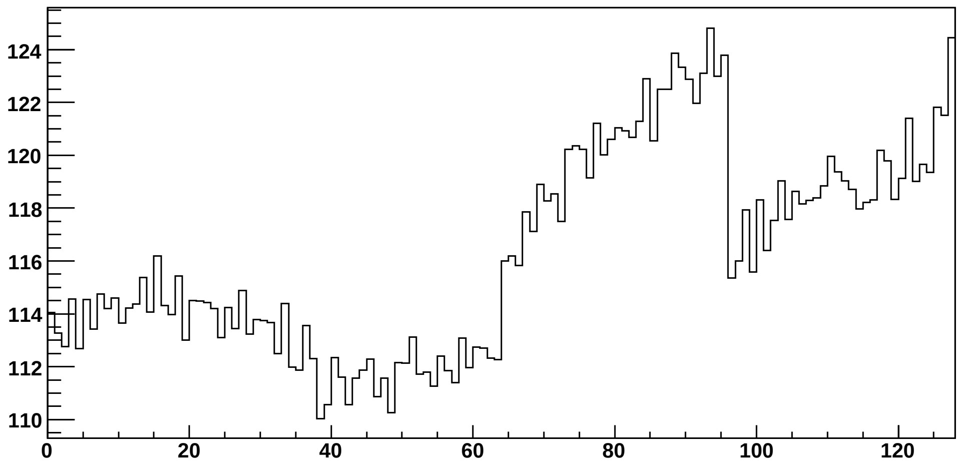


Figure 6.6: The average ADC value for each channel separately

## 6.3 The influence of the header to Channel 0

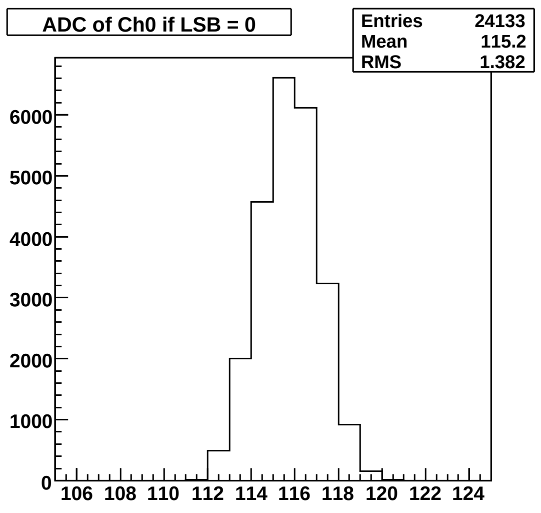
The contribution to noise due the header can be studied by comparing the average ADC values of channel 0 in the case of LSB is true and false separately (Figure 6.7). Depending on the content of the LSB the average ADC value of channel 0 changes 2 or 3 ADC counts. This is about five percent of the difference between the ADC values of the LSB and channel 0. Contrary to first expectations the correlation between the two neigbours was negative. That means, that a high ADC value doesn't lift but reduce its adjacent ADC value. This effect could may be understood by surveying a typical analog pulse (see figure 1.5). A so-called undershooting effect occurs.

The analysis has been made with the following program code:

```
{
gStyle->SetOptStat(111 0 );

std::ifstream bla("bla");

TH1F *h0 = new TH1F("h0"," ADC of Ch0 if LSB = 0 ",20,105,125);
TH1F *h1 = new TH1F("h1"," ADC of Ch0 if LSB = 1 ",20,105,125);

TCanvas* tPlot = new TCanvas("  ","  ",1000,500);

Int_t n =48000;
int ch0x[n];
int lsby[n];
//......loop over all events
for(Int_t i=1; i<=n; i++){

  int delimiter=0;
  int nevt=0;
  int lsb=0;
  int pcn=-3;

  bla >> delimiter;    // read "header data"
  bla >> nevt;
  bla >> lsb;
  bla >> pcn;

  //......loop over all channels
  for (int j=0; j<128; j++){
    int adc=0;
    bla >> adc;

    if (j==0){
        if (bool(lsb>100) ==1){
          h1->Fill(adc);
        }
        else
          h0->Fill(adc);
      }
  } //......end loop channels
} //......end loop events

tPlot->Divide(2,1);

tPlot->cd(1);
h0->Draw();

tPlot->cd(2);
h1->Draw();

}
```

## 6.4 The influence of the header to the channels 1, 2 and 3

Now the effect to the channels 1, 2 and 3 is studied. Figure 6.8 shows like figure 6.7 the average ADC value in dependence of the LSB, but for the channels 1,2, and 3 that aren't directly adjacent



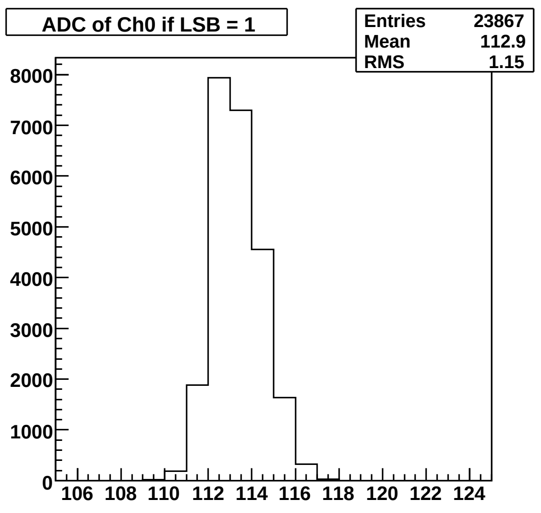


Figure 6.7: The average ADC value of channel 0 is different in case of LSB is true and false

to the pseudo-digital LSB. The program code is saved as 'CHifLSB.C'. In contrast to channel 0 the crosstalk factors are positiv here and as expected the crosstalk factor becomes smaller with the distance. In channel 3 the influence of the header is no more apparent. The crosstalk effects for the channels 2 and 3 are about the half of the effect in channel 0.

## 6.5 Calculation of raw noise for each channel

The principal object of the project was the reduction of noise. In order to affirm the logarithm a tool is needed that pinpoints the noise. Figure 6.9 shows the raw noise of beetle 4 of the bare hybrid L11. There were saved 48'000 Events. The program code is:

```
{
  gStyle->SetOptStat(111 0 );

  std::ifstream bee4("bee4");

  TH1F *h = new TH1F("h","Noise of Channels",128,0,128);

  TCanvas* tPlot = new TCanvas("  ","  ",1000,500);

  Int_t n =4800;               // number of events

  //......initialize vectors and arrays
  float sumar[128][n];
```

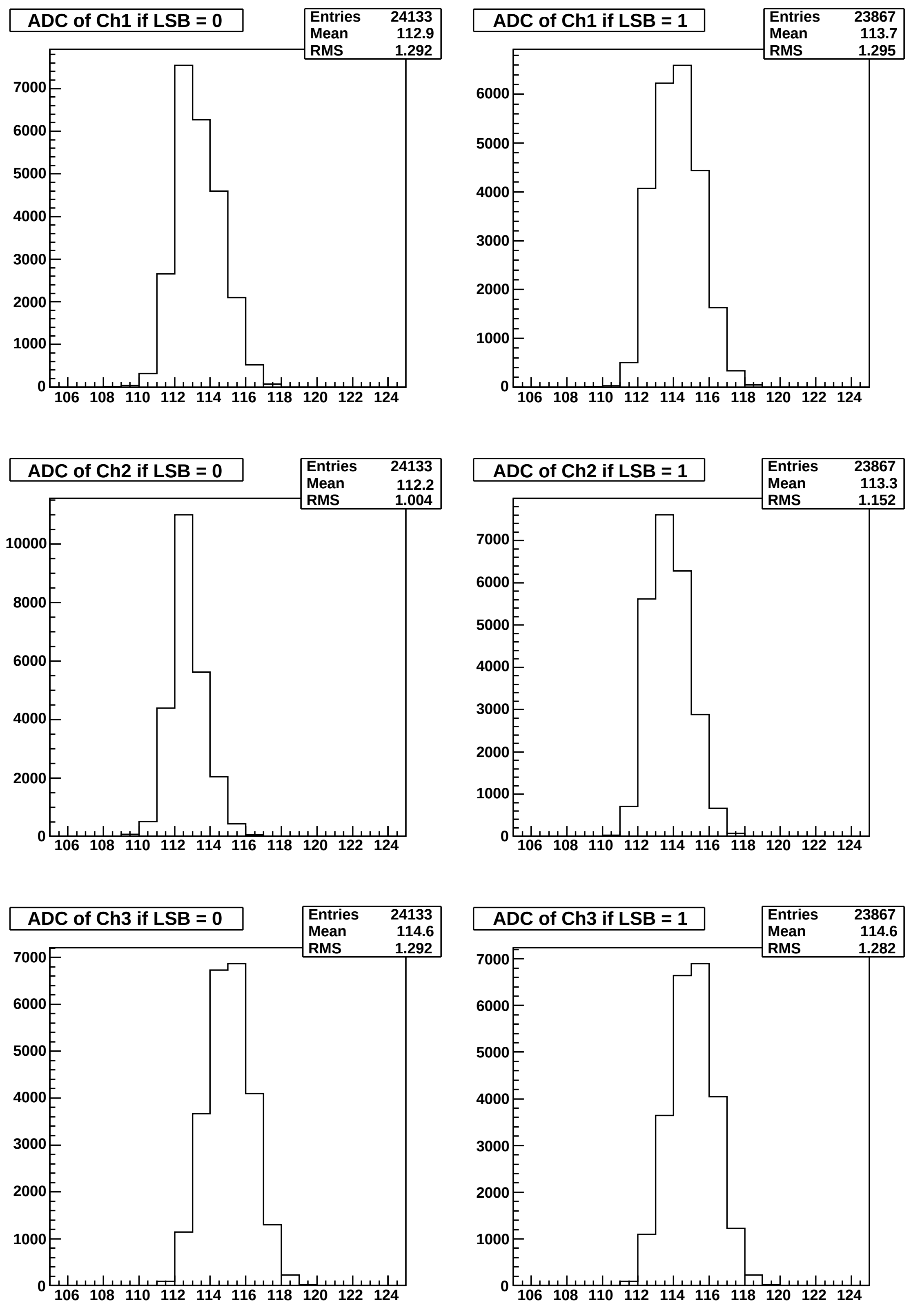


Figure 6.8: The crosstalk effect of LSB to the channels 1, 2 and 3.

```
float summe[128];
float mittel[128];
float noise[128];
float diff[128][n];
float sumdiff[128];


//......set zero all components
for(int j=0; j<128; ++j){
  summe[j]=0;
  mittel[j]=0;
  noise[j]=0;
  sumdiff[j]=0;
  for(int i=0; i<n; ++i)
    {
      sumar[j][i]=0;
      diff[j][i]=0;
    }
}

//......loop over events
for(Int_t i=0; i<n; ++i){

  int delimiter=0;            // read delimiter from data file
  bee4 >> delimiter;

  int nevt=0;                 // read event number
  bee4 >> nevt;

  int lsb=0;                  // read LSB
  bee4 >> lsb;

  int pcn=-3;                 // read PCN
  bee4 >> pcn;


  //......loop over all channels
  for (int j=0; j<128; j++){

    int adc=0;
    bee4 >> adc;              // read ADC value

    sumar[j][i] = float(adc); // fill array with all ADC values
    } //......end loop over channels
  } //......end loop over events


  //......calculate mean for each channel
  float summe[128];
  float mittel[128];

  for (int j=0; j<128; ++j){

    for (int i=0; i<n; ++i){
      summe[j] += sumar[j][i];
    }
    mittel[j]= summe[j]/ float(n);
  }


  //......calculate noise for each channel
  for (int j=0; j<128; ++j)
    {
      for (int i=0; i<n; ++i){
        sumdiff[j] += (sumar[j][i]-mittel[j])
                     *(sumar[j][i]-mittel[j]);
      }
      noise[j] = sqrt(sumdiff[j]/n);
    }


  //......fill histogram
  for(int j=0;j<128;++j){
       h->Fill(j,noise[j]);
  }

  //......draw histogram
  h->Draw();
}
```

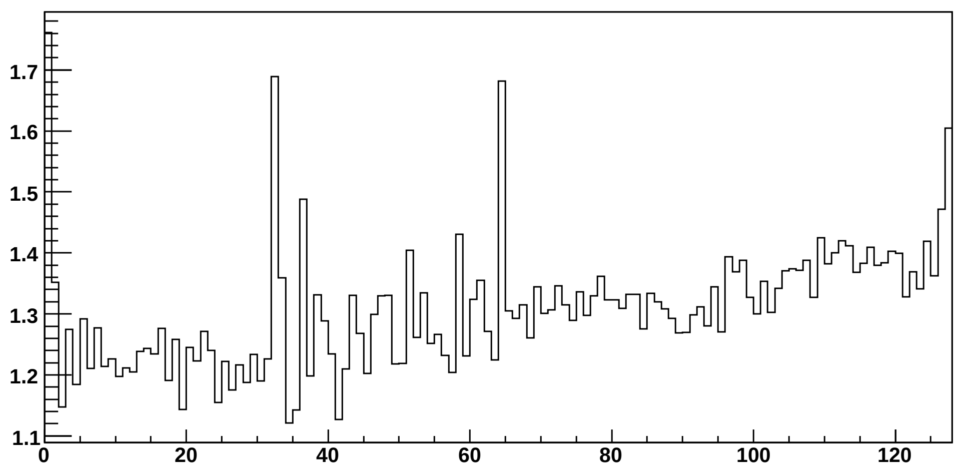


Figure 6.9: Raw Noise of the fourth beetle of the bare hybrid L11

# Chapter 7

# Results

In this chapter the corrected data are tested by the analysis tools that are described in the chapter before. The main goal was the pinpointing of the crosstalk factors. Here the results are presented. Then the noise of the data before and after the appliance of the algorithm is compared in order to see if the algorithm serves the purpose.

## 7.1 Crosstalk factors

The crosstalk factors have been approximated using the function that is defined in "Function-DefMeanComb". This function calculates the average ADC value of channel 0 in case of LSB is true and false seperately and outputs the difference. The same procedure can be done with other channels. In figure 7.2 is shown the dependency of the difference on the crosstalk factor. The minimum is very similar for the beetles one to three, only beetle 4 has a different value. The comparison of the results for the different channels (0, 32, 64 and 96) shows that the crosstalk factors are also not so different. For one channel, beetle and measurement the crosstalk factor can be pinpointed very precisely ($< 10^{-4}$), but the goal is to find a common crosstalk factor by accepting a maximal difference of the two means. This maximal difference should be small compared with the rms-value of the ADC distribution in case of LSB is true and false (see 7.1). The rms-value is between 1 and 1.5 for each beetle and channel. Therefore a maximal difference of 0.1 could be accepted without any problems. This difference would ask for a crosstalk factor precision of $\pm 0.015$ what can be estimated from figure 7.3 and the 15 analog plots for the different channels and Beetles.

In table 7.1 are listed the calculated crosstalk factors for the channels 0, 32, 64 and 96. Because the factors varies from $-1.2$ to $-4.9$ it isn't possible to find a common correlation factor that fulfills the above condition. But we have only to ignore Beetle 4 to diminish the variation to 1.4. Then the condition is very well fulfilled. The question arises if the Beetle 4 is an exception or if it must be expected that the Beetles behaviour is so different.

The mean difference of evaluated precisely for the different Beetles and also for the channels 0, 32, 64 and 96 seperately. They can be found by minimizing the difference between the mean of

Table 7.1: The crosstalk factors for the channels 0, 32, 64 and 96 of the four Beetle of the bare hybrid L11

| | Beetle 1 | Beetle 2 |
|---|---|---|
| Channel 0 | $(-4.6 \pm 0.1) \cdot 10^{-2}$ | $(-4.6 \pm 0.1) \cdot 10^{-2}$ |
| Channel 32 | $(-3.7 \pm 0.1) \cdot 10^{-2}$ | $(-4.2 \pm 0.1) \cdot 10^{-2}$ |
| Channel 64 | $(-3.5 \pm 0.1) \cdot 10^{-2}$ | $(-4.9 \pm 0.1) \cdot 10^{-2}$ |
| Channel 96 | $(-4.3 \pm 0.1) \cdot 10^{-2}$ | $(-4.3 \pm 0.1) \cdot 10^{-2}$ |
| | Beetle 3 | Beetle 4 |
| Channel 0 | $(-4.7 \pm 0.1) \cdot 10^{-2}$ | $(-3.4 \pm 0.1) \cdot 10^{-2}$ |
| Channel 32 | $(-4.1 \pm 0.1) \cdot 10^{-2}$ | $(-3.3 \pm 0.1) \cdot 10^{-2}$ |
| Channel 64 | $(-4.5 \pm 0.1) \cdot 10^{-2}$ | $(-3.1 \pm 0.1) \cdot 10^{-2}$ |
| Channel 96 | $(-3.9 \pm 0.1) \cdot 10^{-2}$ | $(-1.2 \pm 0.1) \cdot 10^{-2}$ |

Table 7.2: Second order: The crosstalk factors for the channels 1, 33, 65, 97 of the four Beetle of the bare hybrid L11

| | Beetle 1 | Beetle 2 |
|---|---|---|
| Channel 1 | $(1.60 \pm 0.01) \cdot 10^{-2}$ | $(1.29 \pm 0.01) \cdot 10^{-2}$ |
| Channel 33 | $(-0.27 \pm 0.01) \cdot 10^{-2}$ | $(-0.18 \pm 0.01) \cdot 10^{-2}$ |
| Channel 65 | $(1.02 \pm 0.01) \cdot 10^{-2}$ | $(1.15 \pm 0.01) \cdot 10^{-2}$ |
| Channel 97 | $(0.97 \pm 0.01) \cdot 10^{-2}$ | $(0.86 \pm 0.01) \cdot 10^{-2}$ |
| | Beetle 3 | Beetle 4 |
| Channel 1 | $(1.38 \pm 0.01) \cdot 10^{-2}$ | $(1.24 \pm 0.01) \cdot 10^{-2}$ |
| Channel 33 | $(-0.18 \pm 0.01) \cdot 10^{-2}$ | $(-0.17 \pm 0.01) \cdot 10^{-2}$ |
| Channel 65 | $(0.69 \pm 0.01) \cdot 10^{-2}$ | $(0.86 \pm 0.01) \cdot 10^{-2}$ |
| Channel 97 | $(0.96 \pm 0.01) \cdot 10^{-2}$ | $(0.30 \pm 0.01) \cdot 10^{-2}$ |

Table 7.3: Third order: The crosstalk factors for the channels 2, 34, 66, 98 of the four Beetle of the bare hybrid L11

| | Beetle 1 | Beetle 2 |
|---|---|---|
| Channel 2 | $(2.4 \pm 0.1) \cdot 10^{-2}$ | $(1.9 \pm 0.1) \cdot 10^{-2}$ |
| Channel 34 | $(0.0 \pm 0.1) \cdot 10^{-2}$ | $(-0.0 \pm 0.1) \cdot 10^{-2}$ |
| Channel 66 | $(0.3 \pm 0.1) \cdot 10^{-2}$ | $(-0.1 \pm 0.1) \cdot 10^{-2}$ |
| Channel 98 | $(-0.3 \pm 0.1) \cdot 10^{-2}$ | $(-0.3 \pm 0.1) \cdot 10^{-2}$ |
| | Beetle 3 | Beetle 4 |
| Channel 2 | $(2.0 \pm 0.1) \cdot 10^{-2}$ | $(1.7 \pm 0.1) \cdot 10^{-2}$ |
| Channel 34 | $(-0.3 \pm 0.1) \cdot 10^{-2}$ | $(-0.3 \pm 0.1) \cdot 10^{-2}$ |
| Channel 66 | $(-0.2 \pm 0.1) \cdot 10^{-2}$ | $(0.4 \pm 0.1) \cdot 10^{-2}$ |
| Channel 98 | $(-0.7 \pm 0.1) \cdot 10^{-2}$ | $(0.1 \pm 0.1) \cdot 10^{-2}$ |

Table 7.4: Fourth order: The crosstalk factors for the channels 3, 35, 67, 99 of the four Beetle of the bare hybrid L11

| | Beetle 1 | Beetle 2 |
|---|---|---|
| Channel 3 | $(0.4 \pm 0.1) \cdot 10^{-2}$ | $(0.4 \pm 0.1) \cdot 10^{-2}$ |
| Channel 35 | $(-0.4 \pm 0.1) \cdot 10^{-2}$ | $(0.3 \pm 0.1) \cdot 10^{-2}$ |
| Channel 67 | $(-0.4 \pm 0.1) \cdot 10^{-2}$ | $(-0.5 \pm 0.1) \cdot 10^{-2}$ |
| Channel 99 | $(-0.3 \pm 0.1) \cdot 10^{-2}$ | $(-0.4 \pm 0.1) \cdot 10^{-2}$ |
| | Beetle 3 | Beetle 4 |
| Channel 3 | $(0.7 \pm 0.1) \cdot 10^{-2}$ | $(-0.1 \pm 0.1) \cdot 10^{-2}$ |
| Channel 35 | $(0.4 \pm 0.1) \cdot 10^{-2}$ | $(0.2 \pm 0.1) \cdot 10^{-2}$ |
| Channel 67 | $(-0.2 \pm 0.1) \cdot 10^{-2}$ | $(-0.6 \pm 0.1) \cdot 10^{-2}$ |
| Channel 99 | $(-0.8 \pm 0.1) \cdot 10^{-2}$ | $(-0.5 \pm 0.1) \cdot 10^{-2}$ |

the selected channel in the case of LSB is false and true respectively. For this reaseon the function 'createRunFromRawData' in the subprogram 'FunctionDef' was modified. There is a loop over crosstalk factors between -1 and 1.

In this example the crosstalk factor for channel 0 was manually approximated. In figure **??** is shown the squared difference of the means for different crosstalk factors (normed) for channel 0. The crosstalk factor depends on the beetle, but they are not so different. The same procedure can be done for the channels 32, 64 and 96. Then the expression '$j == 0$' and the component of the vector 'event.header' has to be changed. The measured crosstalk factors are shown in 7.1.

## 7.2 Effect of the algorithm

The following plots compare the noise of the four Beetles before and after the appliance of the algorithm. There were used the crosstalk factors in table 7.1. It's evidently that the algorithm fulfils its purpose. The additional noise of the channels 0, 32, 64 and 96 is canceled. The second and third order crosstalk is neglected. Therefore the the noise of channel 1 isn't reduced.

## 7.3 Open questions

The main goal was to find one crosstalk factor that can reduce the noise of all channels that have significant contributions to noise due to the header. The results shows that this could be possible. The crosstalk factor is simlilar for the majority of the channels. But there was one of four Beetle that was different. The question arises if this is a exception or the normal case. The solution could

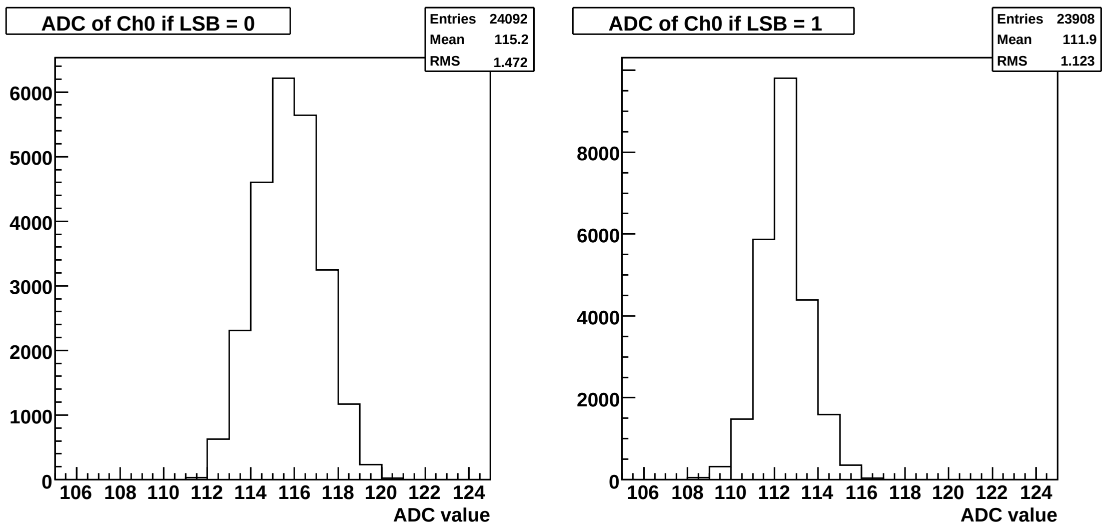


Figure 7.1: Distribution of ADC value of channel 0 in case of LSB=0 and LSB=1 without algorithm.

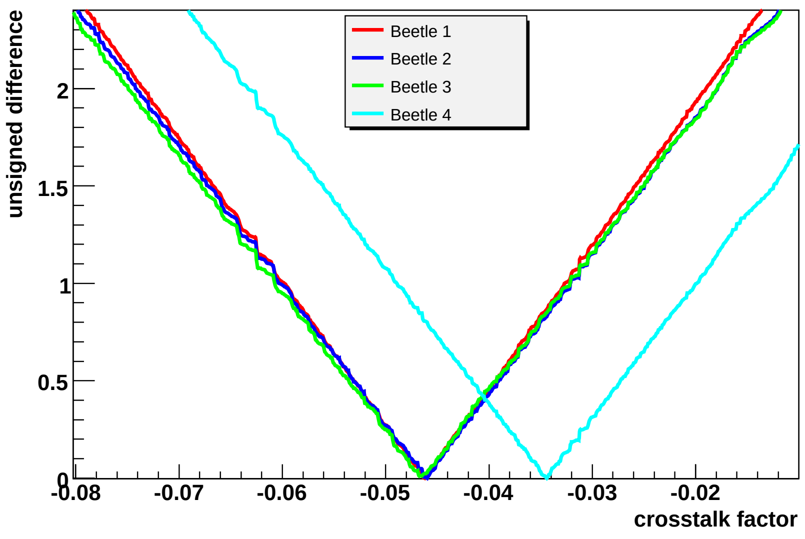


Figure 7.2: The difference of the means in case of LSB=1 and LSB=0 against the crosstalk factor (channel 0 of each Beetle).

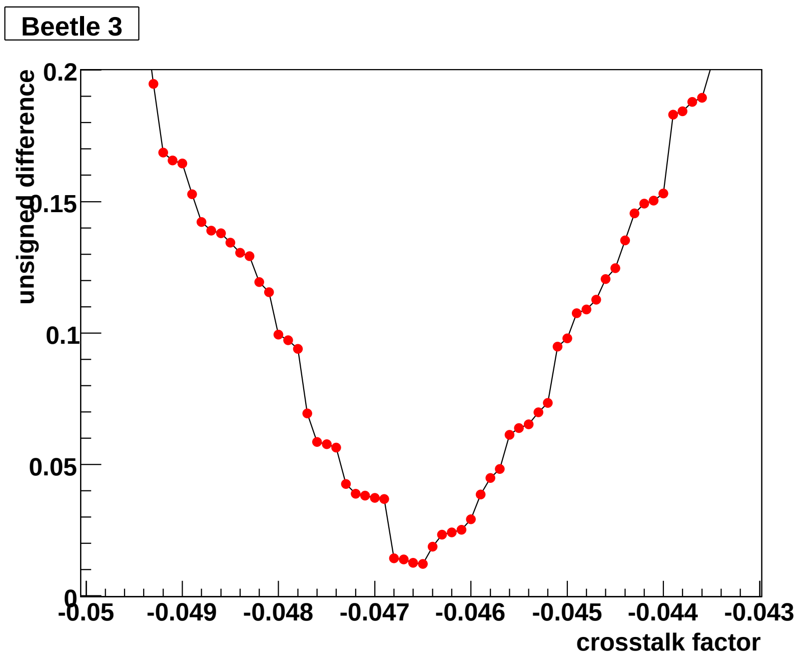


Figure 7.3: The difference between the means in case of LSB=1 and LSB=0 against crosstalk factor for channel 0 of beetle 3.

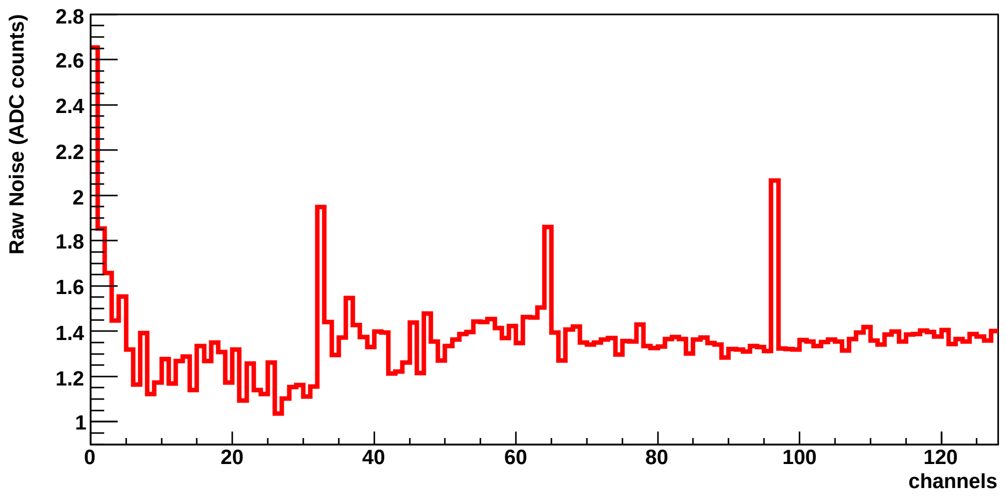


Figure 7.4: Raw Noise of Beetle 1 of the bare hybrid L11 without algorithm.

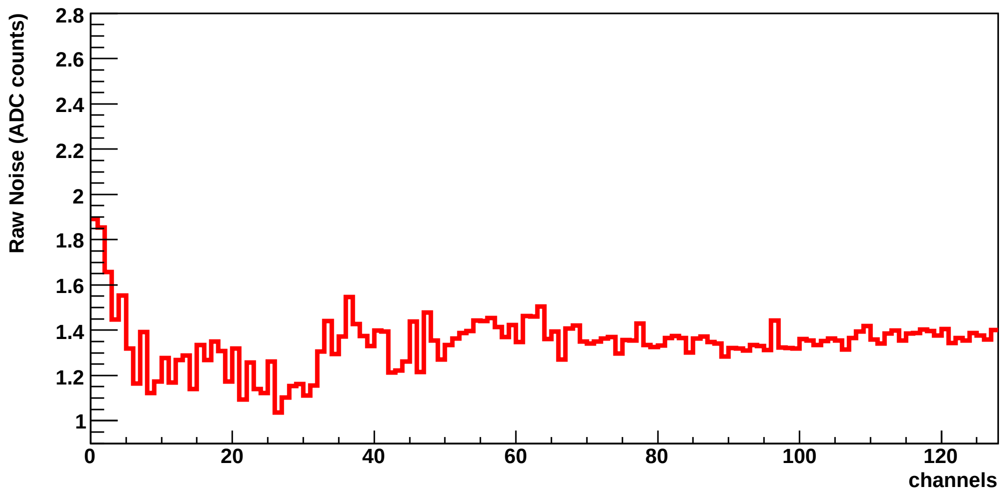


Figure 7.5: Raw Noise of Beetle 1 of the bare hybrid L11 with algorithm (only first order correction).

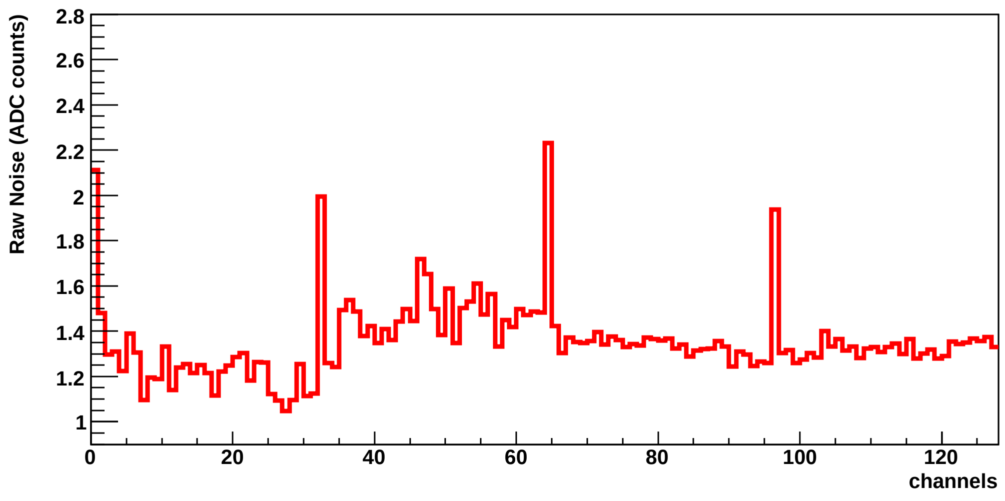


Figure 7.6: Raw Noise of Beetle 2 of the bare hybrid L11 without algorithm.

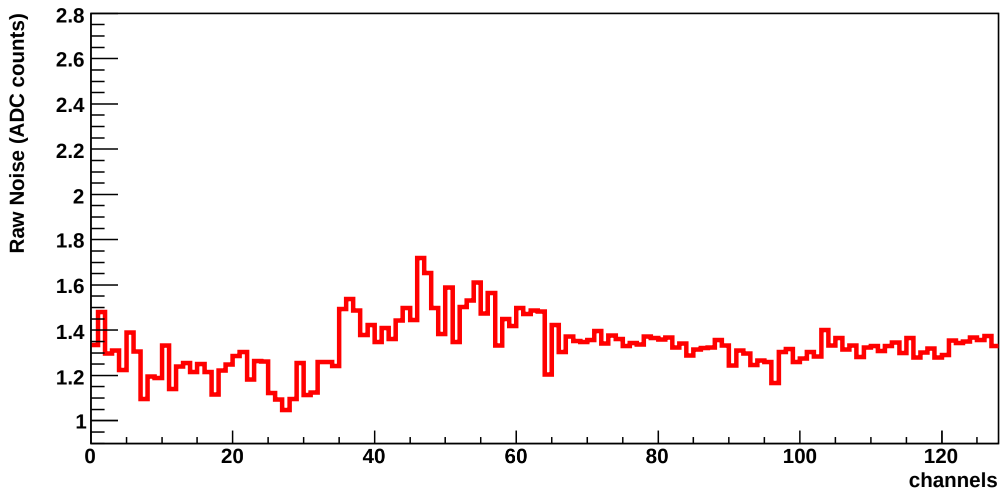


Figure 7.7: Raw Noise of Beetle 2 of the bare hybrid L11 with algorithm (only first order correction).

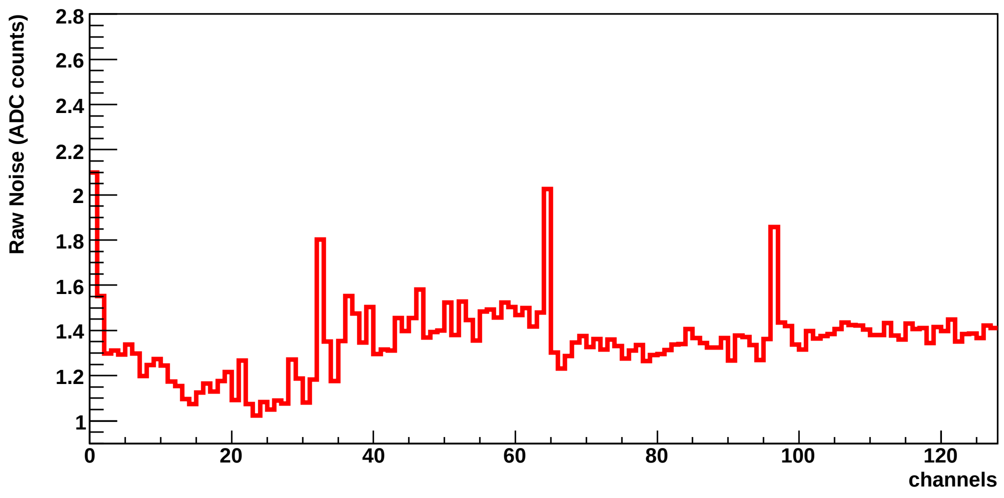


Figure 7.8: Raw Noise of Beetle 3 of the bare hybrid L11 without algorithm.

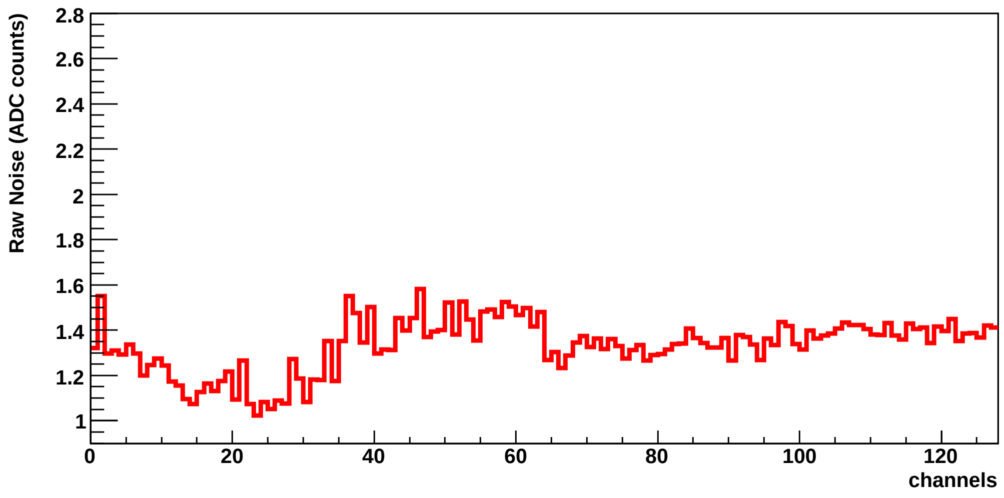


Figure 7.9: Raw Noise of Beetle 3 of the bare hybrid L11 with algorithm (only first order correction).

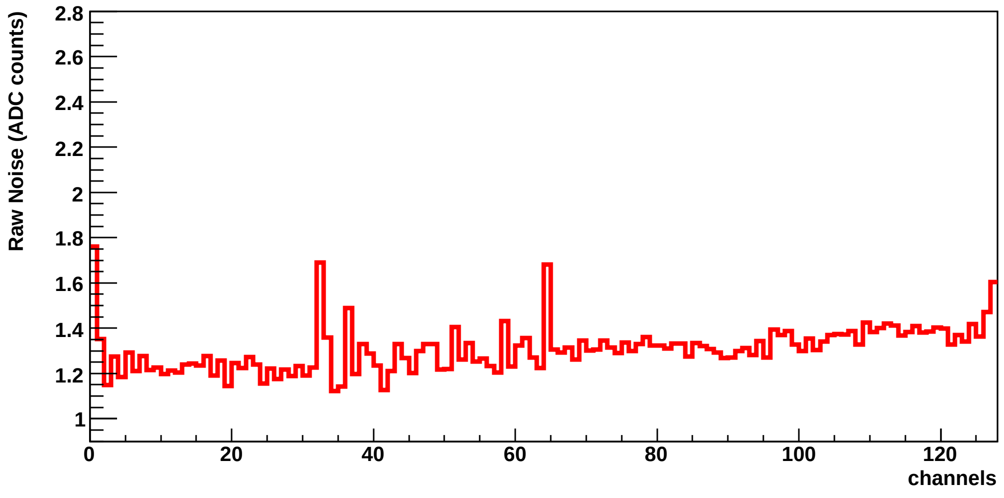


Figure 7.10: Raw Noise of Beetle 4 of the bare hybrid L11 without algorithm.

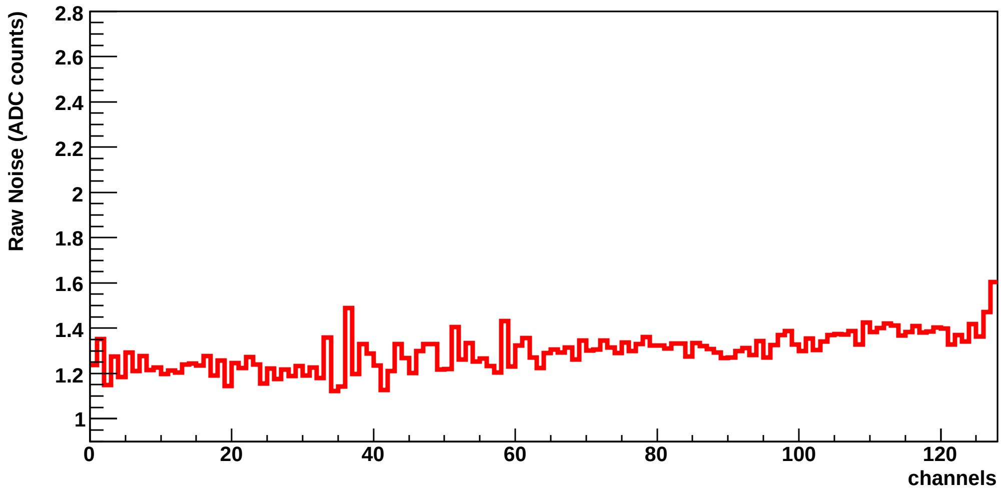


Figure 7.11: Raw Noise of Beetle 4 of the bare hybrid L11 with algorithm (only first order correction).

be found by the measurement of other hybrids. Furthermore it would be interesting to interchange the cables, because they could generate different conditions. In addition the function of the timing has to be studied that probably influences the crosstalk factor.

The second and third order crosstalk can be neglected mostly. But there are also counter-examples. Sometimes the second and third order crosstalk are significantly involved to the noise, especially in the case of channel 0. There the noise is about the half of the first order crosstalk effect. It's not clear yet why this happens only in the channel 1 and 2 of each beetle, but cannot be observed in the channels 33, 34, 65, 66, 97 and 98.

# Chapter 8

# Summary

Noise measurements of a bare hybrid in the laserteststand have shown that the channels next to the header have much more noise than the others, and that a correlation between the header and the adjacent channels exists. This additional contributions to noise cannot be explained by basic electronic noise principles of the detector but by noise sources in the readout chain. It seems that the noise source is caused by a bug in the readout Beetle that produces a crosstalk effect. In order to analyse the crosstalk effect due the header several tools have been programmed. The influence of the pseudo-digital header bit to the analog data channels is significant because the ADC value of a header bit is very different in case of true and false respectively. While the ADC values of the analog data channels varies only about 10 ADC counts, the difference between true and false is approximately 70 ADC counts.

The effect of the crosstalk can be corrected by an algorithm, that is applied to the raw data, before the signals are digitized and corrections are applied. The algorithm is integrated in the C++ program ”pedana” that converts the labview output files to a root tree. The main challenge of the algorithm is to correct the channels that are adjacent to the header by canceling the effect of the crosstalk. Therefore a fraction of the difference between the ADC values of the header and the analog data channels was subtracted from data channel. The accurate magnitude of the fraction can be expressed by the so-called crosstalk factor. The approximation of the crosstalk factor was done by averaging over the ADC values in the case of the last header bit is true and false separately. The crosstalk factor that minimized the difference between the two means best was used in the algorithm.

One result of the approximation was that the crosstalk factors are not so different. Three of four Beetles have almost the same crosstalk factor. Therefore it seems to be possible to use the same crosstalk factor for each channel that is adjacent to the header. But there are more measurements necessary to decide if the Beetles behavior is constant enough. The discrepancy of the fourth Beetle could be caused by different cables.

The second result was that the crosstalk factor can be negative. This was actually the case for channel 0 and all other channels next to the header. That means that a high ADC value doesn’t lift but reduce its adjacent ADC value. This seems to be an undershoot effect and reminds to a typical analog pulse.

A third result was that there exists a second and third order crosstalk. For example the channel 1 and 2 are also influenced by the LSB. In contrast to the channels next to the header they have positive crosstalk factors, but normally the effect is much smaller than in the channels 0, 32, 64 and 96. In the algorithm the second and third order crosstalk factors have been neglected.

The algorithm has been applied to the data of four different Beetles. The result was that the noise was reduced severely.

# Acknowledgment

First of all, I would like to thank Prof. Dr. Ulrich Straumann for his help and for giving me the opportunity to write my diploma thesis at the Physics Institute of the University of Zürich. As next I would like to thank Dr. Achim Vollhardt and Dr. Johannnes Gassner for their very friendly support and for critical reading the manuscript. I thank Angela Büchner, Christoph Salzmann, Dr. Olaf Steinkamp, Dima Volyanskyy and Andreas Wenger for their assistance and for the pleasent working atmosphere.

# List of Figures

# List of Tables

# Bibliography

[1] CERN/LHCb. Reoptimized Detector Design and Performance. Technical report, LHCb TDR 9, 2003.

[2] CERN/LHCb. Lhcb Inner Tracker Technical Design Report. Technical report, LHCb TDR 008, 2002.

[3] M. Needham *et al.* Update geometry description for the LHCb Trigger Tracker. *LHCb note*, 032, 2006.

[4] A. Buechler. Thermal and Mechanical Characterization of the TT Detector for the LHCb Experiment. *Master Thesis*, 2007.

[5] S. Löchner and M. Schmelling. The Beetle Reference Manual - chip version 1.3, 1.4, 1.5. Technical report, LHCb Electronics, 2005.

[6] Guido Haefeli *et al.* TELL1: Specification for a common read out board for LHCb. *LHCb note*, 007, 2003.

[7] P. Moreira *et al.* A Radiation Tolerant Gigabit Serializer for LHC Data Transmission. *http://proj-gol.web.cern.ch*, 2001.

[8] S. Koestner and U. Straumann. Noise considerations of the Beetle amplifier used with long silicon strip detectors. *LHCb note*, 029, 2005.

[9] U. Straumann. Noise predictions for the TT production ladders. *LHCb note*, 029 addendum, 2005.

[10] C. L. Gomez. Signal and Noise Performance of a Prototype Silicon Microstrip Detector for the LHCb Silicon Tracker. -, 2004.

[11] S. Braibant *et al.* Investigations of design parameters for radiation hard silicon microstrip detectors. *Nucl. Instr. and Meth.*, A 485, 2002.

[12] L. Shekhtman G. Corti. Radiation background in the LHCb experiment. *LHCb note*, 083, 2003.

[13] M. Siegler *et al.* Expected Paritcle Fluences and Performance of the LHCb Trigger Tracker. *LHCb note*, 070, 2004.

[14] Laurent Locatellii *et al.* Tests on the VeLo analogue transmission line with the TELL1 prototype RB3. *LHCb note*, 086:19, 2004.

[15] Laurent Locatellii *et al.* Tests on the VeLo analogue transmission line with the TELL1 prototype RB3. *LHCb note*, 086, 2004.

[16] Y. Ermoline. Vertex Detector Electronics: ODE Pre-Prototype. *LHCb note*, 057, 2001.

[17] M. Agari *et al.* Test Beam Results of Multi-Geometry Prototype. *LHCb note*, 058, 2002.

[18] R. Brun, F. Rademakers, P. Canal, I. Antechva, D. Buskulic, O. Couet, A. Gheata, and M. Gheta. *ROOT Users Guide 4.04*. CERN, Geneva, 2005.