Development and commissioning of data acquisition systems for the KASCADE-Grande experiment

DIPLOMARBEIT
zur Erlangung des akademischen Grades
Diplom-Physiker

dem Fachbereich Physik der
Universität Siegen

vorgelegt von
Sven Over

Juni 2004

Contents

Chapter 1

Introduction

1.1 Astroparticle physics and cosmic rays

High energy particles from the cosmos are constantly hitting the earth's atmosphere. Besides photons and neutrinos, this includes charged particles which are commonly referred to as "cosmic rays". Ionized nuclei like protons and alpha particles constitute about 98% of the particles, about 2% are electrons. Of the nuclei, protons make up the the largest fraction ($\approx 87\%$), followed by α-particles ($\approx 12\%$). Nuclei of heavier elements constitute only the remaining 1% [Lon92].

The ionizing effect of cosmic rays was already observed by Henry Coulomb as early as 1785. He discovered that a charged metal sphere, though isolated, could not retain its charge. After the discovery of radioactivity by Becquerel in 1896, it was thought that the ionizing radiation of radioactive materials in the earth was responsible for the observed discharge [Rao98].

The phenomenon was examined in many different experiments. Wulf developed special electrometers to measure ionization, which he used for observations at different altitudes. He found that the ionization at the top of the Eiffel Tower was significantly lower than on ground level, but the decrease was not as high as expected when assuming that γ-rays—at that time the most penetrating radiation known—were responsible for the ionization. In 1912 Victor Hess undertook a series of balloon flights in order to measure the degree of ionization at even higher altitudes. On April 17th 1912 he performed his first flight in Lower Austria on the day of a partial solar eclipse. He reached a height of $2,000$ m and found first evidence that the intensity of the ionizing radiation increases at high altitude. He also remarked, that the ionization rate did not drop during the solar eclipse and that therefore the sun could not be the source of the radiation if it came from the cosmos. Hess undertook more flights to examine the rate of ionization at moderate altitudes of several hundred metres. He found that the ionization decreased with height up to about $1,000$ m above sea level, due to the decrease of ionization caused by ground radioactivity. On August 7th 1912 Hess undertook his seventh flight and reached an altitude of $5,350$ m. At $3,600$ m he measured ionization rates clearly higher than at ground level, and the rate kept on increasing with altitude. He concluded that there must be radiation with a high capability for penetrating the atmosphere coming from above [Hes12].

Since the days of Victor Hess, cosmic rays were extensively studied. Both the muon and the pion were first observed in cosmic ray experiments, employing cloud chambers and nuclear emulsions [Gru00]. The highest energies observed so far are beyond 10^{20} eV: The Fly's Eye experiment measured one event with a cosmic ray primary particle energy of approximately $3.2 \cdot 10^{20}$ eV in 1991 [Nag00].

The energy spectrum of cosmic rays is usually expressed in differential form as the flux I (number of incoming particles per area, per solid angle, per second) per energy interval, as a

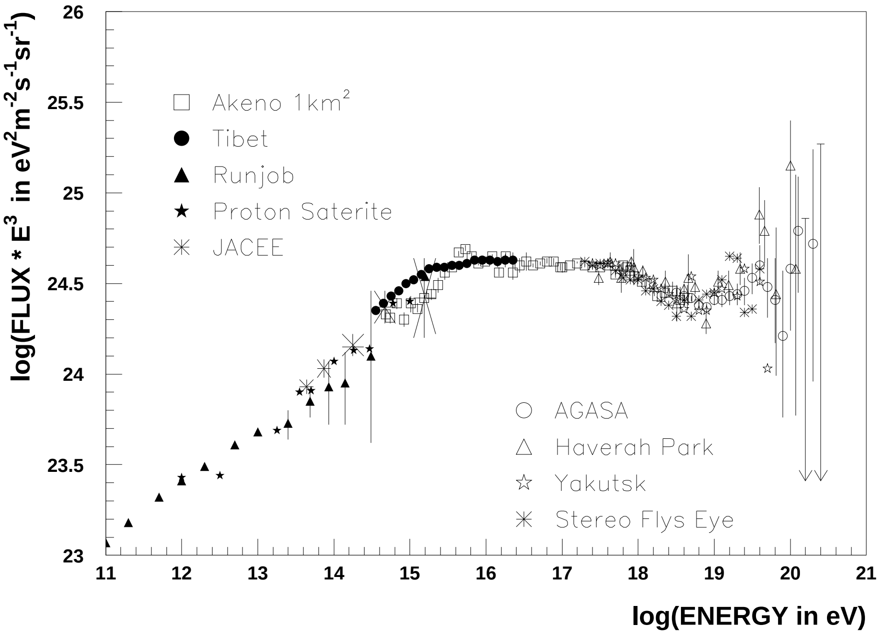


Figure 1.1: Differential energy spectrum of primary cosmic rays: Below 10^{16} eV, results are shown from direct measurements (JACEE, RUNJOB and Proton Satellite). Data above 10^{18} eV are from four experiments: AGASA, Haverah Park, Yakutsk and Fly's Eye [Nag00].

function of the energy E. It was found, that the energy spectrum follows a power law over many orders of magnitude: $\mathrm{d}I/\,\mathrm{d}E \propto E^{-\gamma}$. Figure 1.1 shows the energy spectrum above 10^{11} eV as measured by various experiments. For better visualization, the flux is multiplied with E^3. The value of the spectral index γ is approximately 2.7 up to energies of $5 \cdot 10^{15}$ eV. At this energy, the spectrum shows a kink, commonly referred to as "the knee", and γ changes its value to 3.1. A second kink ("the ankle") appears at an energy of $3 \cdot 10^{18}$, and the spectral index changes its value back to 2.7 [Bie01].

The reasons for the changes of the spectral index have not been determined yet. Recent measurements suggest, that the knee in the total energy spectrum is caused by a kink in the energy spectrum of light elements. The position of the knee even seems to be different for hydrogen and helium [Ulr04]. Different models exist that try to explain how cosmic ray particles are accelerated to high energies and how they propagate through the galaxy. In the theory of Fermi acceleration, the cosmic ray particles are treated as a gas that is compressed between magnetic shock fronts. The maximum energy which can be reached in this process is proportional to the charge of the particle [Bie01].

The knee could be an evidence for a maximum energy of the acceleration mechanism, or for a threshold energy at which particles can escape from the galaxy. The precise examination of the energy spectrum and the chemical composition in the knee region and above is vital in order to evaluate different models. The knee at $5 \cdot 10^{15}$ eV for light elements might be followed by another decrease of the spectral index at higher energies for heavy elements such as iron, since the underlying mechanism might depend on the magnetic rigidity or the mass of the particles.

Direct measurements of cosmic rays by means of balloon and satellite experiments are only practicable for energies up to about 10^{14} eV, as the sensitive area of such experiments is very limited. For higher energies, the particle flux is too low to gain sufficient statistics. Ground based experiments can occupy a large area and can be run for a long time, but they are unable to measure cosmic ray particles directly, because the particles interact with the atoms of the earth's atmosphere. Through inelastic scattering, secondary particles are created which again interact with the atmosphere. The cascade of particles created in this process is called an "extensive air shower".

1.2 Extensive air showers

When primary cosmic ray particles hit the earth's atmosphere, they start a cascade of nuclear interactions that produce an avalanche of secondary particles. The mass depth of the earth's atmosphere amounts to approximately 1,000 $\mathrm{g/cm^2}$. Comparing this number with the radiation lengths of photons and electrons of 37.1 $\mathrm{g/cm^2}$, and with the atmospheric attenuation lengths of hadrons (120 $\mathrm{g/cm^2}$ for nucleons, 160 $\mathrm{g/cm^2}$ for pions, 180 $\mathrm{g/cm^2}$ for kaons), it is obvious that almost no primary cosmic ray particle reaches sea level. The penetration depths of the avalanche of secondary particles depends on the primary energy. The cascade processes induced by primary particles with energies up to 10^{13} eV die out before reaching sea level in the majority of cases, whereas the extended air showers of primary particles with energies above 10^{16} eV typically contain millions of secondary particles that reach ground level.

Extensive air showers consist of three components: the electromagnetic, the hadronic and the muonic component. The primary particle, being a nucleus, establishes the hadronic component with the first interaction, which typically takes place in an altitude between 15 and 20 km above sea level. The hadronic component consists of baryons, mesons and core fragments. Most of the particles created in hadronic interactions are charged and neutral pions. Almost 100% of the charged pions decay to muons and neutrinos and thereby feed the muonic component of the air shower. Neutral pions decay in 98.8% of the cases to two photons, which initiate the electromagnetic component. Due to their high energy, the longitudinal momentum of the hadrons is much bigger than their transverse momentum. Therefore the hadrons stay relatively close to the shower axis, which is defined as the prolongation of the original track of the primary particle.

Highly energetic photons from the decay of neutral pions initiate electromagnetic cascades of pair production and bremsstrahlung processes. These processes take place alternatingly and cause an exponential increase of electrons and photons with time, as long as the mean electron energy does not fall below a critical energy (84.2 MeV for electrons in air), at which the energy loss due to ionization begins to dominate against the radiation loss.

The muons produced by decays of charged pions do almost not interact with the atmosphere, as muons barely lose energy in bremsstrahlung processes due to their larger mass. [Gai90, Rao98, Gru00]

The measurement of extensive air showers can make use of a variety of techniques: relativistic particles create Cherenkov photons in the atmosphere, or they excite atoms which then emit fluorescence light. Particles reaching ground level can be detected by means of scintillators, Cherenkov detectors (e.g. employing water tanks), calorimeters or wire chambers.

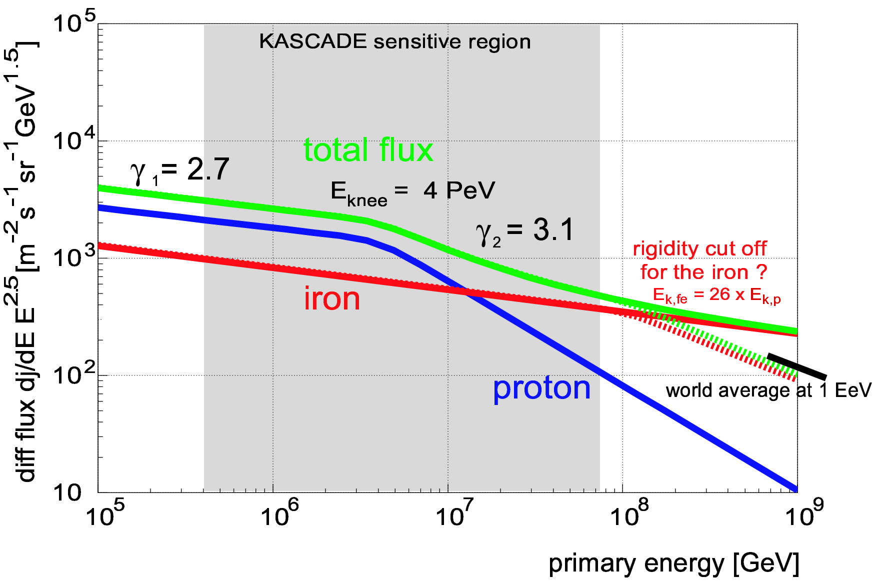


Figure 1.2: Cosmic ray spectrum around the energy range of KASCADE: The extension of KASCADE to KASCADE-Grande will provide measurements of primary energies up to 10^{18} eV [Hau03].

1.3 Aims of the KASCADE-Grande experiment

KASCADE-Grande is a ground based multi-detector experiment, aimed for the measurement of the energy spectrum and chemical composition of cosmic rays up to energies of 10^{18} eV. It is an extension of the KASCADE experiment, which is in operation since 1996.

The major goal is the observation of the "iron knee". Figure 1.2 illustrates the energy spectrum around the range that was already covered by KASCADE. Recent analysis of KASCADE data suggests a charge dependence of the knee position for different primary particle elements [Rot03]. A further verification of this dependency would be the observation of the iron knee at a primary particle energy of about $E = Z_{\text{Fe}} \cdot E^{H}_{\text{knee}} \approx 10^{17}$ eV, which demands measuring energy spectrum and chemical composition of cosmic rays up to energies well above 10^{17} eV.

Figure 1.3 displays an air shower event, taken jointly by the KASCADE and the Grande arrays. While KASCADE offers fine grained information on electron and muon numbers within the area of the KASCADE array, the Grande array increases the sensitive area from 0.04 km^2 to about 0.5 km^2. In three years of data taking, KASCADE-Grande will collect about 25,000 events with primary energies above 10^{17} eV, and about 250 above 10^{18} eV [Ber01, Nav04]. Analysis of the acquired data will allow for statements on the energy spectrum as well as the chemical composition in this energy range and therefore will give evidence if a second knee exists and whether this is due to a decrease of the flux of heavier elements such as iron.

1.4 Aims of the Grande FADC system

A new data acquisition system for the Grande array is currently being developed. It will provide full pulse shape information of the detector signals by means of flash ADCs (FADCs). This will make examinations of the time development of the energy deposits in the detector stations

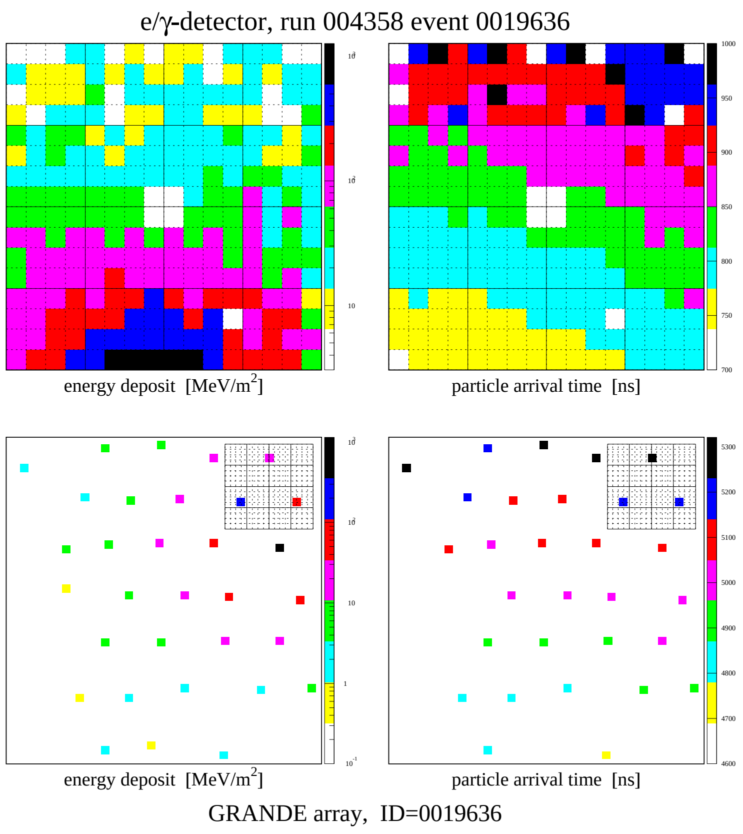


Figure 1.3: KASCADE-Grande event display from joint data taking of the KASCADE (top) and Grande (bottom) arrays. Energy deposits are shown on the left hand side, relative arrival times on the right hand side [Gla03].

possible. Designed as a dead time free system, it can be used for the search for intercorrelated shower events, even if these follow shortly after each other, as they might be caused by the decay of hypothetic exotic particles. By applying fit algorithms to the pulse shapes obtained, the effective time resolution is kept below one nanosecond.

The overall aim for the new data acquisition system is to improve the quality of the data recorded with the Grande array significantly.

1.5 Scope of this thesis

This thesis describes the data acquisition environment of the Grande array of the KASCADE-Grande experiment. In the first part the currently running data acquisition system is reviewed and the online software, which has been developed as a part of this thesis, is discussed in detail. The second part covers the new FADC based data acquisition system, which is currently under development. The overall design is presented and concepts of the data processing are discussed. The hardware related part of the new online software, which was developed as well as a part of this thesis, is documented.

Therefore, the structure of this thesis is as follows.

- In chapter 2, the KASCADE-Grande experiment and the Grande detector setup are described.
- Chapter 3 covers the current data acquisition hardware. The different components are explained and the trigger logic is discussed.
- Chapter 4 documents the online software used for data taking in detail.

- In chapter 5, some data obtained with the previously described system are presented in order to demonstrate its successful operation.
- In chapter 6, the design of the new FADC based data acquisition system is introduced.
- Chapter 7 deals with the tasks and concepts of the software for the FADC system. It also contains the documentation of the part which has already been developed.
- In Chapter 8, a summary is given.

Chapter 2

The KASCADE-Grande experiment

KASCADE-Grande is a cosmic ray air shower experiment, located on site of the Forschungszentrum Karlsruhe, at 49° northern latitude, 8° eastern longitude, 110 meters above sea level. It incorporates the KASCADE (**Ka**rlsruhe **S**hower **C**ore and **A**rray **De**tector) experiment which is in operation and taking data since 1996. KASCADE is a multi-detector setup, comprising an array of 252 scintillator detector stations on an area of $200 \times 200\,\mathrm{m}^2$, a central detector and a muon tracking detector. It was designed to investigate cosmic rays with primary particle energies in the PeV range. By the end of the year 2002 it has collected a data set of $8 \cdot 10^8$ showers [Ant03].

In order to extend the energy range of the experiment up to primary particle energies of $10^{18}\,\mathrm{eV}$, KASCADE was extended by the Grande and Piccolo arrays to form the KASCADE-Grande experiment. The Grande array consists of 37 scintillator detector stations from the former EAS-TOP experiment. EAS-TOP had been in operation between 1989 and 2000 at Campo Imperatore on Mount Gran Sasso at $2,000\,\mathrm{m}$ above sea level [Cas03]. The Piccolo array is located between the centres of the KASCADE array and the Grande array and provides a fast common trigger for both, KASCADE and Grande.

The main aim of KASCADE-Grande is the measurement of the energy spectrum and the chemical composition of cosmic rays in the range from 10^{16} up to $10^{18}\,\mathrm{eV}$ [Ber01].

2.1 The components of KASCADE-Grande

Table 2.1 lists the components of KASCADE-Grande. The layout of KASCADE-Grande is shown in figure 2.1.

2.1.1 KASCADE

The former KASCADE experiment, which is now part of KASCADE-Grande, consists of the following components [Ant03].

- The scintillator station detector array, comprising 252 stations on a grid with 13 m spacing, organized in 16 clusters. The stations include unshielded liquid scintillator detectors for measuring the electromagnetic components of showers. The stations in the outer twelve clusters are additionally equipped with plastic scintillator detectors, placed below 10 cm of lead and 4 cm of iron shielding, dedicated to the measurement of the muonic component.

Detector	Particle	Total area (m^2)	Threshold
KASCADE:			
Array, liquid scintillators	e/γ	490	5 MeV
Array, plastic scintillators	μ	622	230 MeV
Muon tracking detector, streamer tubes	μ	128×4 layers	800 MeV
central detector:			
Calorimeter, liquid ionization chambers	h	304×8 layers	50 GeV
Trigger layer, plastic scintillators	μ	208	490 MeV
Top cluster, plastic scintillators	e/γ	23	5 MeV
Top layer, liquid ionization chambers	e/γ	304	5 MeV
Multi-wire proportional chambers	μ	129×2 layers	2.4 GeV
Limited streamer tubes	μ	250	2.4 GeV
Grande	e/μ	370	3 MeV
Piccolo	e/μ	80	5 MeV

Table 2.1: Detector components of KASCADE-Grande, their total sensitive areas and thresholds for vertical particles [Ant03, Kam03].

- The central detector, containing (from top to bottom):
  - The top cluster, made out of 25 scintillation counters, serving as a trigger source for small showers and filling in the gap of the four missing array detector stations at the location of the central calorimeter.
  - The hadron sampling calorimeter with nine layers of warm-liquid ionization chambers between iron and concrete absorbers, featuring a total of 44,000 readout channels.
  - The trigger layer, consisting of 456 scintillators installed in the third gap (counted from top) of the sampling calorimeter, used as a fast trigger source and for measuring arrival times.
  - Two layers of multi wire proportional chambers, installed below the hadron calorimeter, measuring muons with energies exceeding a threshold energy of 2.4 GeV (for vertical muons).
  - The limited streamer tubes, installed below the multi wire proportional chambers in order to improve the reconstruction quality of muon tracks in case of high muon densities.
- The muon tracking detector, located in a 48 m long tunnel north of the central detector building. It consists of three layers of limited streamer tubes with a vertical spacing of 82 cm, and vertical chambers at the sides of the tunnel.

With this multi-detector setup, KASCADE is able to measure a set of different observables for every shower. Data of different components allow for consistency checks and cross calibration. A schematic layout of KASCADE is given in figure 2.3.

2.1.2 Grande

The Grande array comprises 37 scintillator detector stations from the former EAS-TOP experiment, placed on a hexagonal grid with an average spacing of 137 m. Each station is equipped

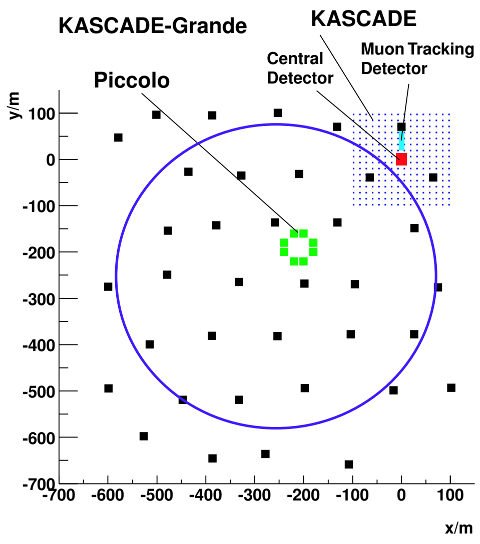


Figure 2.1: Schematic layout of the KASCADE-Grande experiment [Hau03].

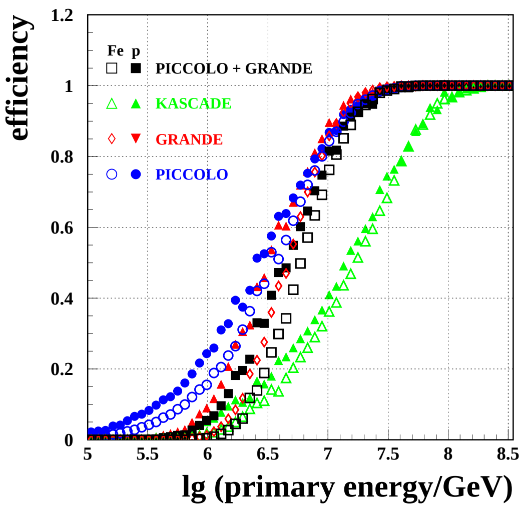


Figure 2.2: Simulated trigger efficiency for showers inside the area of the circle in figure 2.1 [Hau03].

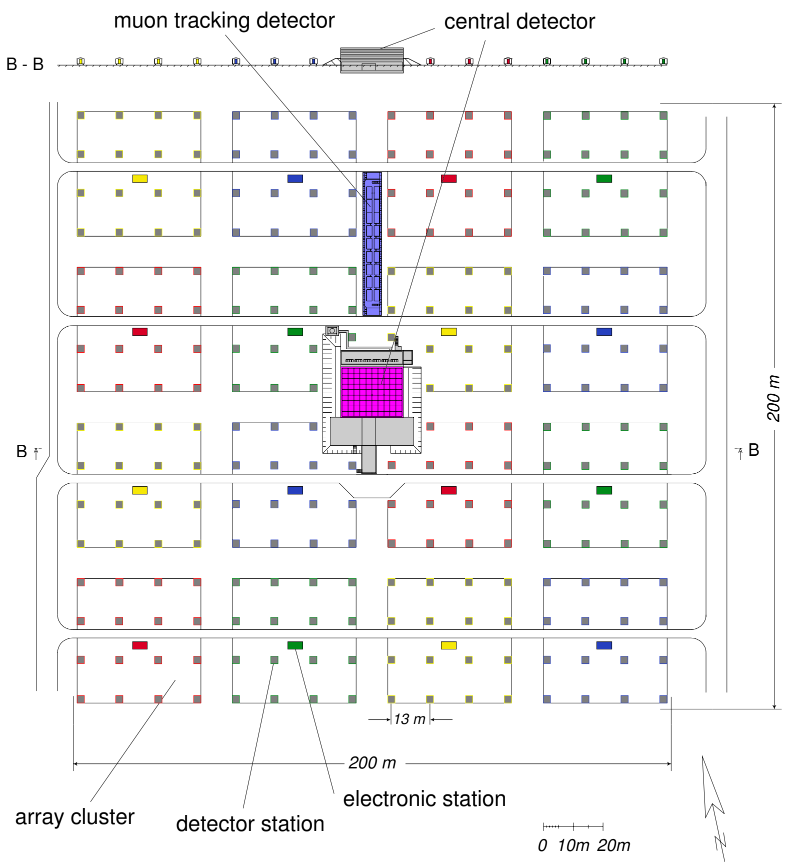


Figure 2.3: Schematic layout of the KASCADE part [Ant03].

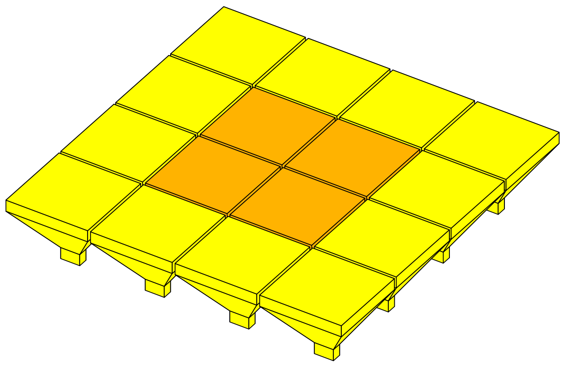


Figure 2.4: Sketch of a Grande scintillator station. The inner four modules are equipped with two photo multiplier tubes (high gain and low gain channel). The remaining twelve modules have only a high gain channel photo multiplier.

with 10 m^2 of plastic scintillator, subdivided into 16 individual detector units and arranged in a 4×4 matrix. Every unit contains a NE102A plastic scintillator with an area of 80×80 cm^2 and 4 cm thickness. The scintillator is mounted in a metal case with an upside down pyramidal shape. The inside of the case is painted white in order to provide for diffuse reflection of scintillator light pulses. Below the scintillator block, a photo multiplier of type XP3462 is mounted at the tip of the pyramid. It is operated in high gain mode at a voltage of 1,500 to 2,050 V and will be referred to as “Pb”. For measuring high particle densities, the central four scintillator pyramides are equipped with a second photo multiplier tube (“Pb1”) of the same type, operated in low gain mode at a lower voltage of 1,200 to 1,600 V. The lower voltage results in signal amplitudes of about 10% of the high gain amplitude. A minimum ionizing muon vertically penetrating the scintillator deposits an energy of 8.2 MeV. Taking into account the angular distribution, the mean energy deposit per minimum ionizing particle amounts to 10.3 MeV [Agl93]. The dynamic range achieved in units of a minimum ionizing particles per 10 m^2 is 0.3 to 750 for the high gain channel, and 12 to 30,000 for the low gain channel [Ber01].

2.1.3 Piccolo

The Piccolo array consists of eight detector stations, equipped with 10 m^2 of plastic scintillator each, located near the centre of KASCADE-Grande. Its main purpose is to provide a fast common trigger to both , KASCADE and Grande, for a joint measurement of air showers. Figure 2.2 shows the simulated trigger efficiency for different trigger sources, depending on the primary energy. The Piccolo trigger condition used in this simulation is coincidence in at least two Piccolo detector stations, involving at least four of 48 electronic channels [Hau03]. With this, Piccolo is 100% efficient for energies above approximately $5 \cdot 10^{16}$ eV.

Chapter 3

The Grande data acquisition system

In this chapter, the hardware components of the Grande data acquisition (DAQ) system and their interaction will be described. The first part covers the electronic setup in the detector stations. The second part deals with the central DAQ system: the trigger logic and the measurement devices.

3.1 The Grande detector stations

Each of the 37 Grande detector stations is equipped with approximately $10\,\mathrm{m}^2$ of plastic scintillator and 20 photo multiplier tubes (see section 2.1.2). The individual output signals of the 16 Pb (high gain) and 4 Pb1 (low gain) photo multipliers are added up to one high gain and one low gain signal by means of analogue signal mixers. One output of the high gain signal mixer is connected to a discriminator, which has the threshold set such that it is sensitive to a typical signal of a single particle penetrating one of the scintillators. The generated signal is transmitted to the DAQ station and further referred to as "Logic Run Signal".

The mixed high and low gain signals are input to a shaping amplifier[1]. This device integrates the incoming signals for $8\,\mu$s and shapes pulses with an amplitude proportional to the integrated charge of the signals. The mixed high gain signal is split to two output signals with different amplifications, hence referred to as "Pb" and "Pb*10" signals. As the name suggests, the Pb*10 is amplified by a factor of ten, compared to Pb. For the low gain channel there is only one output, from now on referred to as the "Pb1" signal of the detector station.

The use of a signal shaper is necessary since the signals are transmitted over approx. 700 m of coaxial cable, which were laid into existing cable trays and are subject to interference noise introduced by other cables. The original mixed photo multiplier signals have a typical width of a few nanoseconds. Without signal shaping these signals cannot be distinguished from noise in the DAQ station. The signal shaper generates pulses with a risetime of 8 microseconds [CAE03].

An overview of the signal conditioning is given in figure 3.1.

The high voltage supplies for all 37 detector stations are located in the central DAQ station. There are two high voltage cables from the DAQ station to each detector station: one for the high gain tubes and one for the low gain tubes. In order to get comparable signal responses by all photo multipliers, the high voltage has to be separately adjusted for each photo multiplier. Therefore each detector station is equipped with a programmable high voltage divider, which provides individual voltages to every photo multiplier tube. The voltage divider can be accessed

[1]CAEN N442. It was especially developed for KASCADE-Grande by CAEN in collaboration with INFN Turin and Forschungszentrum Karlsruhe.

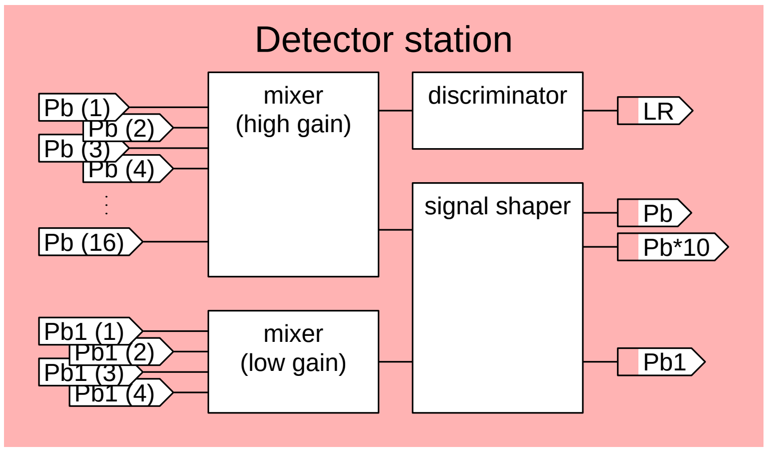


Figure 3.1: Overview schematic of the signal conditioning in a Grande detector station.

via a multiport processor in the detector station that is connected to the DAQ station via glass fibres.

3.2 The central DAQ station

A simplified overview of the hardware components in the DAQ station is given in figure 3.2. The shaped detector signals and logic run signals of all 37 detector stations are transmitted to the central Grande DAQ station. In addition, the global KASCADE-Grande timestamp signals (1 Hz and 5 MHz) as well as the global KASCADE-Grande trigger (hence referred to as “external trigger”) are available. All these are input to the data acquisition hardware, which consists of the following components:

- The coincidence logic, which generates trigger signals from the incoming logic run signals whenever they fulfil certain trigger conditions,
- the single particle trigger logic, which allows to select one of the incoming logic run signals to be the single particle trigger,
- the real time clock module, which creates a timestamp for the generated trigger signals, and also serves as control component, since it is equipped with logic outputs that can be set by the online software,
- scalers, that count the logic run signals (effectively the single station events) of all 37 detector stations,
- pattern units, which store the trigger source of the event,
- a TDC[2], which measures the relative times between logic run signals during one air shower event,
- peak-sensing ADCs[3] that digitize the amplitude of the three shaped photomultiplier signals (Pb*10, Pb, Pb1) from the detector stations,

[2]Time-to-digital converter
[3]Analogue-to-digital converter

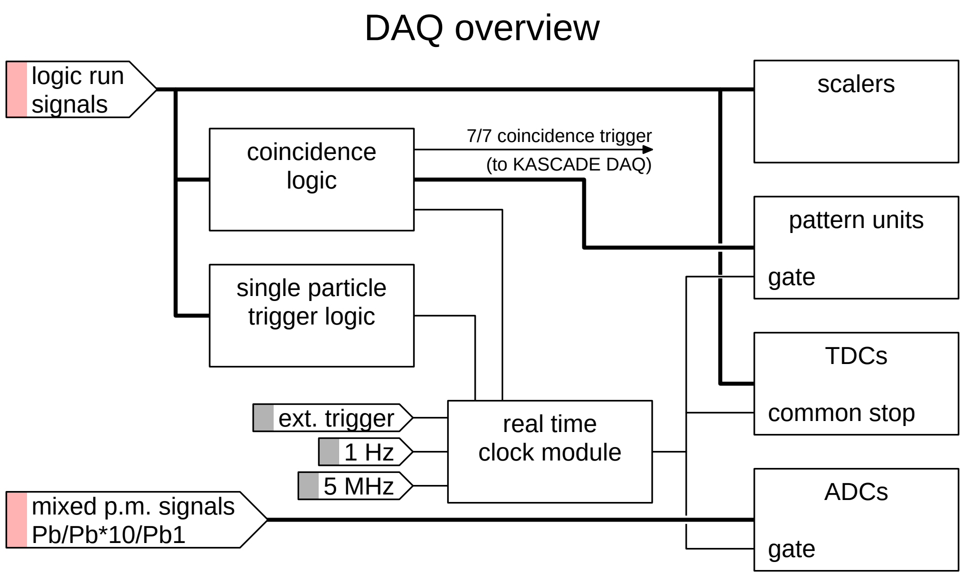


Figure 3.2: Simplified overview schematic of the Grande DAQ station.

- the DAQ PC, executing the online software, which controls and reads out all other devices.

All hardware components that are programmable or can be read out, are mounted in CAMAC or VME crates. The data acquisition PC can access these devices via the CAMAC and VME busses by means of special hardware. The next chapter will cover the online software running on the DAQ PC in detail. Section 4.5 will focus on the hardware access by the DAQ PC.

The operation of the Grande data acquisition system is event driven. As soon as a trigger signal initiates the acquisition of an event, the measurement devices are activated. The online software on the DAQ PC acquires the data from the devices and prepares the devices for the next event. Until this process is completed, a veto signal suppresses the recording of another air shower event.

The term *"event"* is used in two different contexts: A *single station event* is the detection of at least one particle in a single detector station, marked by the logic run signal. An *air shower event* incorporates the measurement of a number of detector stations, that detected particles in coincidence.

The acquisition of an air shower event can be initiated by two different trigger sources:

- The coincidence logic, which issues a trigger signal whenever the logic run signals fulfil defined trigger conditions,
- the central KASCADE-Grande trigger. Thereby, Grande is triggered externally by other components of KASCADE-Grande, as the Piccolo or the KASCADE array.

Apart from these two trigger sources, a measurement cycle can also be initiated by the single particle trigger: In order to obtain energy spectra of detector stations for calibration purposes, the acquisition of regular air shower data is interleaved with taking uncorrelated single station events. Therefore the logic run signal of a single station, which is selected by the online software by means of a set of programmable CAMAC discriminators[4], can act as the single particle

[4]LeCroy 4413

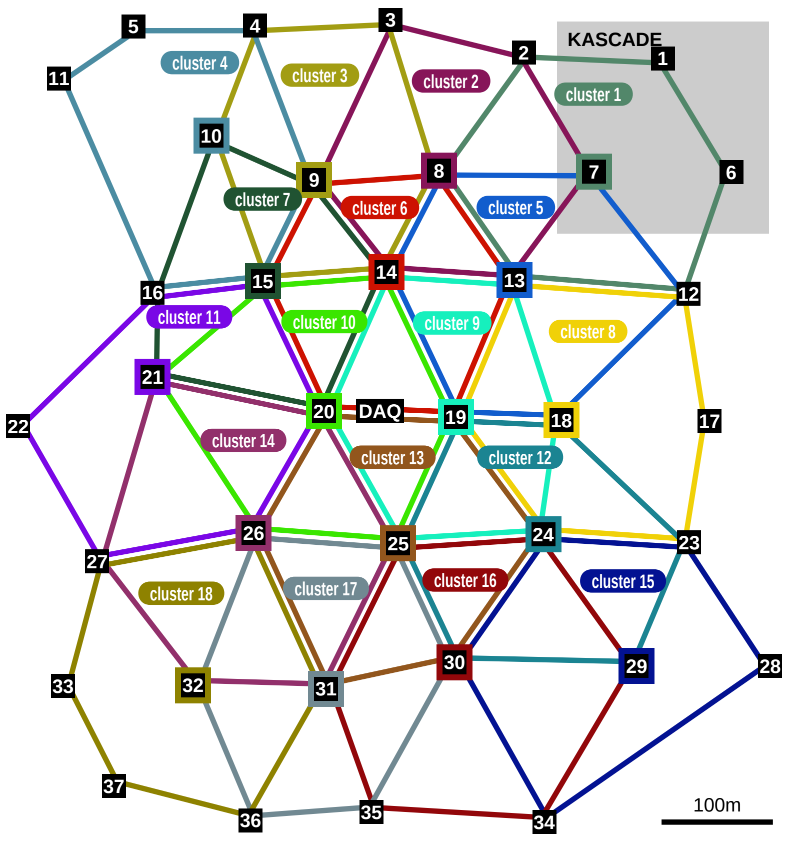


Figure 3.3: Arrangement of trigger clusters.

trigger. The online software will perform only a partial read-out in this case, in order to release the veto signal quickly and therefore to reduce the dead time to a minimum.

3.2.1 The coincidence logic

The Grande array is organised in 18 trigger units, so-called clusters. Each cluster comprises seven detector stations and has a hexagonal shape. The trigger clusters are overlapping. Every station is part of up to seven trigger clusters. The arrangement of the trigger clusters is shown in figure 3.3.

For a cluster, there are two levels of coincidence. The first level is the four-out-of-seven coincidence (hence called 4/7 coincidence). A schematic overview of the coincidence logic is given in figure 3.4. A 4/7 coincidence demands that there is a coincidence of the logic run signals of the central station of the cluster with at least three adjacent peripheral stations. The second level is the seven-out-of-seven (7/7) coincidence, which demands coincidence of the logic run signals of all stations. Technically, the detection of these two kinds of coincidences is provided by one programmable coincidence unit[5] for each cluster. These devices have eight inputs and six outputs. At run start they are programmed using CAMAC commands by defining the six output states for each of the $2^8 = 256$ possible combinations of input states. The first four outputs are programmed to report 4/7 coincidences, the remaining two to report 7/7 coincidences. The logical OR of the 4/7 coincidence signals of all 18 clusters forms the global 4/7 coincidence signal. The global 7/7 coincidence signal is generated by a logical OR of the 18 7/7 coincidence signals.

[5]CAEN C85

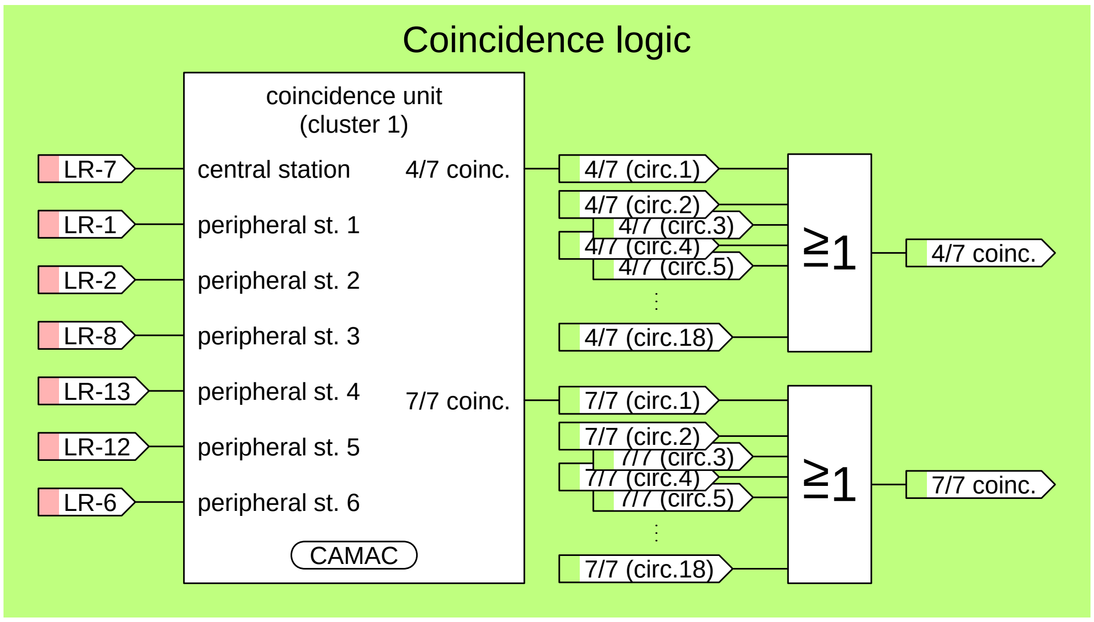


Figure 3.4: The coincidence logic: for each cluster, one programmable coincidence unit detects 4/7 and 7/7 coincidences. The figure shows the coincidence logic for cluster 1 as an example. (LR: logic run signal)

3.2.2 The trigger and veto logic

A schematic of the trigger logic is shown in figure 3.5. The central component is the real time clock module, a custom-made VME module, formerly developed for the KASCADE experiment. It has inputs for the 1 Hz and 5 MHz timestamp signals as well as the EVENT signal. The EVENT signal is formed as logical OR of the trigger sources. Unlike the 4/7 coincidence signal, the one particle trigger signal as well as the external trigger signal can be disabled by the real time clock module using two of the programmable outputs. An incoming EVENT signal makes the real time clock module store the current values of the 1 Hz and 5 MHz clock counters, and the BUSY output is immediately switched to HIGH. The latter serves as veto signal, in combination (logical OR) with one of the programmable outputs of the real time clock module. This has technical reasons, as the online software has finer control over the programmable output than over the BUSY output.

As a result, an EVENT signal occurs whenever the veto signal is not HIGH and either

- (at least) one 4/7 coincidence is detected, or
- an external trigger signal is received and its acceptance is not disabled, or
- a single particle trigger was generated, and its acceptance is not disabled.

The so-generated EVENT signal is not only input to the real time clock module, but triggers gate generators for the measurement devices as well. The acceptance of the external trigger signal is configured at run start and remains unchanged during the whole run. The acceptance of the single particle trigger is managed by the online software: After taking a limited number of single particle events (three with the current configuration) following an air shower event, the acquisition of single particle events is stopped until the next air shower event.

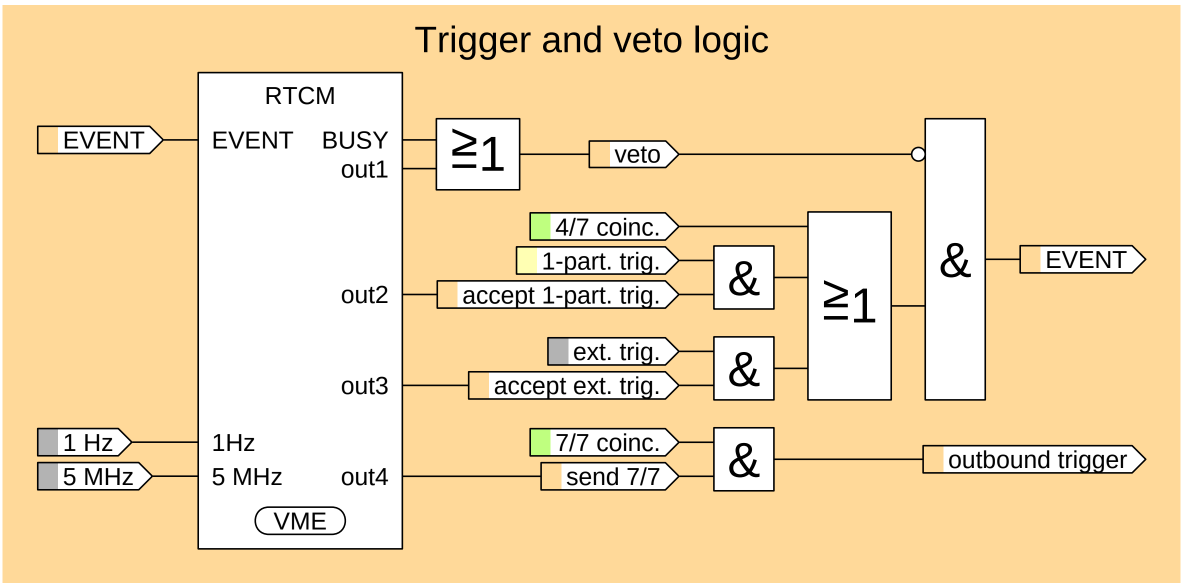


Figure 3.5: The Grande trigger logic.

3.2.3 TDC and ADC measurements

The observables of main interest are the relative times between the particle detections in the different stations and the energy deposits. These are recorded by one 128 channel TDC[6] and a set of four 32 channel peak-sensing ADC modules[7].

The TDC receives the logic run signals of all detector stations, and for every EVENT signal it generates one data package, containing the measured times of the logic run signals relative to the EVENT signal with a resolution of 0.8 ns. It can record multiple hits per channel. All hits in a time window around the EVENT signal, which is configured by the online software to $\pm 5\,\mu$s, are taken into account. The obtained results are subject to the time calibration, which corrects for unequal cable lengths. The development and operation of a time calibration system are described in [Stü03], which also lists the acquired time correction offsets.

The four peak-sensing ADC modules digitize the maximum amplitudes of the received signals for the Pb*10, Pb and Pb1 channels of each detector station with a resolution of 12 bits. The Pb*10 signals of the first 32 detector stations are connected to the first ADC module. The second and third ADC module are connected to the Pb and Pb1 signals for the same stations. All three channels of the detector stations 33–37 are connected to the fourth ADC module. A gate generator is started by the EVENT signal, and provides a $10\,\mu$s wide gate signal for the ADCs. The ADCs generate one data package containing the measured values for each EVENT signal. The correlation of ADC channel numbers of the Pb*10 channel with energy deposits is given by the single particle spectra (see section 5.2). The calibration of the Pb and Pb1 channels is performed by correlating their values with the Pb*10 channels in the overlapping region, and extrapolating to higher energies (see section 5.3).

[6]CAEN V767
[7]CAEN V785

3.2.4 Trigger identification

In order to identify the trigger source that caused a read-out cycle, two 16 channel VME pattern units[8] are installed. Connected to them are the 4/7 coincidence signals of the 18 trigger clusters. In addition, the ORed 7/7 coincidence signal and the external trigger signal are input to a pattern unit, as well as the single particle trigger (after conjunction with the enable signal from the real time clock module). The latter allows the online software to detect whether the read-out cycle was triggered by a single particle event or an air shower trigger. For a single particle trigger, only a partial read-out is performed, which only features acquisition of the Pb*10 channel ADC value of the given detector station.

[8]CAEN V259N

Chapter 4

The Grande data acquisition software

In this chapter, the tasks and design of the Grande data acquisition online software will be discussed. It consists of three major parts: First, the initialization of the experiment, which is performed at run start. Second, the event read-out, which is the most time-critical part as it contributes to the experiment's dead time. And third, the communication with the central KASCADE-Grande data acquisition system (DAQ), where stability is especially important, since communication errors can cause the Grande array to be excluded from data taking until the end of the ongoing run. At the end of the chapter, two more tasks of the online software will be discussed. These are the recording of single particle spectra, which is vital for the calibration of the Grande array, and data quality checks, including measurements of single station hit rates and dead time.

4.1 Aims of the software design

The Grande online software is the successor of the EAS-TOP online software, which performed many of the same tasks as the Grande online software for the EAS-TOP experiment. It had been implemented in FORTRAN on the VMS operating system[1]. Computers running VMS can safely be described as legacy hardware. Replacement of damaged or faulty hardware is difficult. Accordingly, one purpose of the reimplementation of the online software was to make use of current consumer PC hardware, which is easy to obtain and replace for a foreseeable time. Another reason which made the migration to new computer hardware necessary was the update of most measurement components from CAMAC devices to VME devices, and access to the VME bus system from the old hardware was not possible. Nevertheless the VMS computer is still in operation in the central Grande DAQ station, since some special programs sporadically used for calibration have not been migrated yet. The reimplementation of these programs is currently in progress.

As the platform for the new Grande online software the Linux operating system was chosen. It offers the necessary facilities both to access the measurement hardware and for the network communication with the central KASCADE-Grande DAQ. Moreover it is well documented, and sufficient expertise existed already.

The design of the new software took three major aims into account:

- stability,

  (A typical KASCADE-Grande run lasts about four days, as run start is usually performed twice a week. A crash of the Grande online software causes the central DAQ software to

[1]running on a DEC MicroVAX II

exclude the Grande array from data taking. It demands manual intervention from the shift crew to reinclude Grande. Therefore it is highly desirable that the Grande online software is stable enough to run steadily for days and weeks.)

- speed,

  (The speed of the read-out impacts directly on the dead time. The read-out is started immediately once an air shower event has been triggered. A veto signal that prevents detection of additional events is set until the read-out is completed. Taking into account that the online software not only has to acquire the data from the hardware components, but also has to send it to a number of consumers (typically the local hard disk and the central KASCADE-Grande DAQ), it is obvious that attention has to be paid to optimization in order to keep the dead time as low as possible.)

- correctness of data.

  (The data constituting one shower event is collected from different measurement devices such as ADCs, pattern units and a TDC. Care has to be taken that the data read from the different devices correspond to the same shower event. Recording of single particle spectra, which will be discussed later, can interfere with shower events and cause ambiguities in the data. To ensure that only correct data are taken, events which may suffer such ambiguities are discarded.)

4.2 Tasks of the online software

This section briefly introduces the different tasks performed by the Grande online software. A more detailed description is given in the following sections.

The Grande online software is started by the shift crew at run start. Its first task is to set up the experiment, i.e. to prepare the hardware components for data taking. This includes setting the appropriate operation modes for the measurement devices, as well as supplying trigger patterns to the coincidence units and setting programmable outputs which determine if the external trigger is to be accepted and whether the Grande 7/7 coincidence trigger (see section 3.2.1) is to be sent to the central KASCADE-Grande trigger distributor.

After initialization all measurement devices are in a clean state. In this state, the hardware can record exactly one event without any intervention by the software. Once an event has taken place, a veto signal is set by the hardware which prevents additional events from being accepted. The online software constantly polls the real time clock module to check if an event has taken place. In case of an event, the online software has to read the data from the different components, to bring the components back to the clean state and to reset the veto signal.

In addition to shower events, uncorrelated events of single detector stations have to be gathered in order to obtain energy spectra. To achieve this, the online software selects one of the 37 detector stations. The logic run signal of the selected station serves as a single particle trigger, which is handled by the hardware in the same way as an air shower trigger. The software can distinguish the events which were triggered by the selected hut from shower events by checking an input channel of a pattern unit. In case of a single particle event, only a partial readout is performed, i.e. the data that contribute to the energy spectrum are read, all other data are cleared. This is to speed up the read-out and therefore reduce dead time. After a defined number of such single particle events for one station have been gathered and histogrammed, a dedicated spectrum event is sent to the consumers which contains the recorded energy spectrum, and the taking of single particle spectra is continued with the next detector station.

Scalers are counting the logic run signals of each detector station. Additionally, a logical AND of the 5 MHz clock signal and the veto signal is counted, in order to estimate the dead time. These values are retrieved once per second and a scaler event is formed containing this information.

The collected data have to be sent to consumers. A consumer can either be a disk consumer, which means that the data are written to a file, or a network consumer, which is another process that connected to the Grande online software by means of a network connection. Consumers can select which kinds of events they receive. Typically there is one disk consumer which dumps all available data to disk, whereas the central KASCADE-Grande DAQ only gets shower events with certain trigger conditions.

At run stop, the sending of Grande 7/7 coincidence triggers to the KASCADE-Grande trigger distributor is switched off. Apart from that no special actions have to be performed, since the clearing of the measurement devices is done at run start.

4.3 Overall design

The Grande data acquisition software is implemented as one executable: `grandedaq`. It is written in the programming language C. The source code is divided into several C files. Additionally there are compiler macros to ease the use of some functions which are defined in header files. Table 4.1 lists the source code and header files of `grandedaq` and their purpose.

The global configuration of the data acquisition system is specified in the file `experiment.h`. Many changes in the configuration such as cabling, arrangement of trigger clusters and addresses of VME and CAMAC devices can be reflected in the online software by changing the corresponding entries in `experiment.h`. It also contains the settings of data acquisition parameters such as the number of single particle events acquired after every shower event. The complete `experiment.h` file is given in appendix A as a reference of the experiment's configuration at the time of writing this thesis.

In order to disentangle the two tasks, hardware readout and communication with the consumers, `grandedaq` can make use of multi-threaded programming. That means that the `grandedaq` process forks into two processes: the main thread takes care of running the experiment, another thread (hence referred to as the dispatcher thread) is responsible for distributing the data. On operating system level, these two processes look like two independent programs. Slow network communication therefore can only delay the thread that distributes the data, not the one that reads out the measurement devices. This is to prevent temporary network problems from causing long periods of dead time. The number of events that `grandedaq` can queue for transfer to the consumers is only limited by the amount of available memory. The use of multi-threaded programming is a compile time option, i.e. `grandedaq` can be compiled either to use threads or not. If it is disabled, `grandedaq` will perform all tasks within the same process. In that case it transfers the event data right after it was generated. From now on, it is assumed that the multi-thread option is enabled.

The overall workflow of `grandedaq` — as implemented in `main.c` — is as follows.

1. The PCI-VME link device is initialized, which is necessary to get access to the VME bus.
2. The data acquisition hardware is initialized by the function `init_experiment`, which is implemented in `experiment.c` (see section 4.8).
3. The data dispatcher is initialized: files on hard disk to write data to are opened and network ports on which `grandedaq` waits for connections from consumers are registered. This is described in more detail in section 4.10.

Name	Purpose
`main.c`	main event loop
`grandedaq.h`	global definitions
`experiment.c`	experiment initialization and read-out routines
`experiment.h`	configuration of the data acquisition system
`dispatcher.c` `dispatcher.h`	data transfer management
`cmessage.h`	definitions for data exchange with central KASCADE-Grande DAQ
`debt.c` `debt.h`	helper functions for data exchange between threads *(only used if `grandedaq` is compiled with thread support)*
`log.c` `log.h`	creation of log messages
`vme.h`	helper macros for accessing VME and CAMAC devices
`caenc85.c` `caenc85.h`	support for CAEN C85 programmable coincidence unit
`caenc236.h`	support for CAEN C236 ‘status A’ module
`caenc257.c` `caenc257.h`	support for CAEN C257 16 channel scaler
`caenv259n.c` `caenv259n.h`	support for CAEN V259N pattern unit
`caenv767.c` `caenv767.h`	support for CAEN V767 128 channel TDC
`caenv785.c` `caenv785.h`	support for CAEN V785 32 channel peak sensing ADC
`lecroy4413.h`	support for LeCroy 4413 programmable discriminator
`gpsmodule.c` `gpsmodule.h`	support for GPS clock module
`rtcm.c` `rtcm.h`	support for real time clock module

Table 4.1: Source code files of `grandedaq`: support for the VME and CAMAC devices is provided by dedicated source files.

4. The dispatcher thread is launched. This thread works independently and therefore has its own workflow which is described next.

5. A signal handler is installed. This function gets called whenever the `grandedaq` process receives a user interruption signal (`SIGINT`) or a termination request (`SIGTERM`). Without a signal handler installed, the `grandedaq` process would be immediately terminated when such signals are received. The signal handler function only sets one variable to denote the reception of a signal and then returns to the normal program execution.

6. The main event loop is entered. It constantly polls the real time clock module via VME for an event. In case of an event, the `readout_experiment` function is called, which is implemented in `experiment.c`. While polling, once every second a read-out of the scalers is performed (function `readout_scaler` in `experiment.c`, see section 4.12).

   The event loop is continuously executed as long as no reception of a signal is flagged by the signal handler.

7. The function `shutdown_experiment` (implemented in `experiment.c`) is called. The dispatcher thread is terminated and the main program calls the function `dispatcher_flush` (in `dispatcher.c`), which sends to the consumers all data which were queued for transmission but have not been transmitted yet.

The workflow of the dispatcher thread (implemented as function `dispatcher_thread` in `dispatcher.c`) only consists of one loop, which alternately performs the two following tasks.

1. It checks for data queued for transmission by the main thread. Every event is added to individual queues of all consumers that shall receive the event. (This is implemented as function `dispatcher_queue` in `dispatcher.c`.) Section 4.7 describes how the data is passed safely from the main thread to the dispatcher thread.

2. The function `dispatcher_select` (implemented in `dispatcher.c`) is called. It goes through the list of consumers and sends data to those which have got queued event data and are ready to receive data. It also accepts new network connections and sets them up as new consumers. Network consumers that have closed the connection are removed from the list of consumers.

The dispatcher thread is discussed more in detail in section 4.10.

The rest of this chapter is conceived as a reference of the Grande data acquisition software, which was developed as part of this thesis. Therefore the description of concepts and data structures is very detailed.

4.4 Command line options

The `grandedaq` program accepts a number of command line options in order to configure run time choices.

- `--exttrigger`

  Accept trigger signals from the central KASCADE-Grande trigger distributor.

- `--sendtrigger`

  Forward Grande 7/7 coincidences to the central KASCADE-Grande trigger distributor.

- `--vme-device` *`device`*

  Use *`device`* as PCI-VME interface card device file name. This defaults to `/dev/vmemm_1` and only needs to be changed if more than one such interface is present.

- `--plufile` *`file`*

  Read settings of the programmable logical units from *`file`*. This is discussed with the initialization of these devices in section 4.8.

- `--comment` *`text`*

  This places *`text`* as a comment in a log message at the beginning of the run's data file.

- `--port` *`port-number[%event-type-selection]`*

  This makes `grandedaq` listen on TCP port *`port-number`* for network connections from consumers. If the *`port-number`* is followed by a '`%`' character and a number, this number selects what kind of events are sent to this network consumer. This is discussed in detail in section 4.10.

- `--file` *`file-name[#events-per-file][%event-type-selection]`*

  Set up a disk consumer which writes data to *`file-name`*. If *`file-name`* is followed by a '`#`' character and a number, a new file will be opened after this many events. In this case, an eight-digit decimal number is suffixed to the output filename, which is incremented for every new file. If *`file-name`* is followed by a '`%`' character and a number, this number is an event type selection of the same type as for the `--port` option.

4.5 Access to the hardware components

All hardware components that can be read out or programmed are attached to either the VME or the CAMAC bus. The VME bus can be accessed from the PC by means of a PCI-VME interface[2]. CAMAC devices are accessed via the VME bus using a CAMAC branch driver[3].

The acronym "VME" stands for "VersaModule Eurocard". It is an international standard (as documented by IEEE and ANSI) which specifies both the mechanical dimensions of VME boards and the electronics of the bus system [Pet97]. VME boards are mounted in crates, in which they are interconnected via the backplane. The PCI-VME interface consists of such a VME module and a PCI card in the PC connected to it. VME devices are accessed by logical addresses they occupy within the VME address space. Via the PCI-VME interface the PC can access this address space. The "Computer Automated Measurement And Control" system (CAMAC) uses geographical addresses, which means that the module slots (called "stations") in CAMAC crates are numbered, as are the crates in one CAMAC branch. Modules are accessed by their branch, crate and station number. All crates within the CAMAC branch are connected via the branch highway to the CAMAC branch driver. In this setup, the CAMAC branch driver is a VME module which maps CAMAC instructions to the VME address space. This way both VME and CAMAC devices can be accessed by the online software.

[2]W-IE-NE-R Plein & Baus PCI-VME

[3]CES CBD 8210

The PCI-VME interface comes with the necessary drivers and a programming library to run it on Linux. `grandedaq` uses this library, but through functions and macros which are defined in `vme.h`. These ease the access to VME and CAMAC devices. They also add a layer of abstraction which makes migration to a different method of VME access easier if it becomes necessary in the future.

Instead of issuing VME and CAMAC commands directly in the experiment read-out function, helper functions for the different hardware components are used. The support for the hardware components used is implemented in dedicated source files as listed in table 4.1. The functions provided for the different devices vary as the devices have different purposes. Only those functions were implemented that are actually used in `grandedaq`.

4.6 Event data structures

The data produced by `grandedaq` are organized as events. In this context ‘event’ does not only stand for an air shower event, but can be any of these four types of events:

- shower event: containing the data acquired by the read-out of a regular air shower event,
- scaler event: containing the values of the scaler modules, obtained once every second,
- spectrum event: containing an energy spectrum histogram of one detector station,
- log message event: containing a text message, e.g. a warning that an air shower event was discarded.

To issue an event, memory for the event data is allocated and a pointer to this memory is given to the ringbuffer. The dispatcher thread retrieves this pointer from the ringbuffer and takes care of sending the event to consumers. The event data memory is freed by the dispatcher thread as soon as it is no longer needed. (See section 4.10.) The general structure of the event data — valid for all four types — is as follows:

1. a `grandedaq`-internal book keeping data structure (`struct event`), which will not be sent to the consumers. Its contents will be discussed in section 4.10,
2. one data structure (`struct CMessage`) containing meta data as the data package size and some words with fixed values (“magic words”). This data structure is used in all network transfers between the central KASCADE-Grande DAQ and the data acquisition programs of subsystems,
3. an event type dependent data structure. This is the `GRANDEHEAD` structure for shower events, `struct scalerevent` for scaler event, `struct spectrumevent` for spectrum events and `struct logevent` for log message events. The contents of these structures are listed in table 4.2.

   The first six member variables of the particular data structures are common for all four types:

   - `magic_word`: this has a fixed value, symbolizing the characters ‘`GREH`’ (“Grande event header”),
   - `event_class`: this is to distinguish between the four types of events and can hold one of the values ‘`GREV`’ (“Grande event”), ‘`GRSE`’ (“Grande scaler event”), ‘`GSPE`’ (“Grande spectrum event”) and ‘`GRLM`’ (“Grande log message”),

GRANDEHEAD	
int magic_word	magic word: 'GREH' = Grande Event Header
int event_class	'GREV' = Grande Event
int run_number	unique Grande internal event ID
int sent_event_number	sequential per-consumer event counter
int rtcm_seconds	seconds counter from real time clock module
int rtcm_microtime	5 MHz time label from real time clock module
int trigger_pattern	bit vector specifying which trigger conditions are fulfilled
int _7_7_pattern	bit vector indicating trigger clusters with 7/7 coincidence
int _4_7_pattern	bit vector indicating trigger clusters with 4/7 coincidence
int hit_pattern_1	hit bits for detector stations 1–32
int hit_pattern_2	hit bits for detector stations 33–37
int length_of_data	number of data words (int) following this header
struct scalerevent	
int magic_word	magic word: 'GREH' = Grande Event Header
int event_class	'GRSE' = Grande Scaler Event
int internal_event_number	unique Grande internal event ID
int sent_event_number	sequential per-consumer event counter
int rtcm_seconds	seconds counter from real time clock module
int rtcm_microtime	5 MHz time label from real time clock module
unsigned long long fivemhz	5 MHz signals counted
unsigned long long deadtime	5 MHz signals counted while veto was on
int scaler_channels	number of scaler channels in this event
int scaler_data[]	scaler values
struct spectrumevent	
int magic_word	magic word: 'GREH' = Grande Event Header
int event_class	'GSPE' = Grande Spectrum Event
int internal_event_number	unique Grande internal event ID
int sent_event_number	sequential per-consumer event counter
int rtcm_seconds	seconds counter from real time clock module
int rtcm_microtime	5 MHz time label from real time clock module
int station_number	number of detector station
int spectrum_channels	number of different ADC channel numbers
int spectrum_data[]	spectrum data
struct logevent	
int magic_word	magic word: 'GREH' = Grande Event Header
int event_class	'GRLM' = Grande Log Message
int internal_event_number	unique Grande internal event ID
int sent_event_number	sequential per-consumer event counter
int rtcm_seconds	seconds counter from real time clock module
int rtcm_microtime	5 MHz time label from real time clock module
int length	length of message string
char string[]	the log message string

Table 4.2: Event data structures

- `internal_event_number` (for historical reasons, this has the name `run_number` in the `GRANDEHEAD` structure): the event number, counted globally for all consumers and all types of events by `grandedaq`,
- `sent_event_number`: an event number which is counted for each consumer separately. It is guaranteed that the first event received by a consumer has the number zero, and that further values of `sent_event_number` increment continuously,
- `rtcm_seconds` and `rtcm_microtime`: values of the 1 Hz and 5 MHz counters of the real time clock. For shower events, this specifies the time of the trigger signal. In spectrum events the time of the last single particle trigger that contributed to the spectrum is taken. For scaler events, it is the time as read out from the real time clock module's time registers before the scaler read-out is performed. Log message events include time values here in case they were read out in the operation that caused the log message, otherwise both values are set to zero.

The remaining contents depend on the type of event and are discussed in the corresponding sections,

4. for shower events, the data read from ADCs and TDCs (these are not part of the `GRANDEHEAD` data structure),
5. one magic word of type `int` (value $1,234,567,890$), marking the end of the data package.

4.7 Data exchange between threads

This section describes how the data is exchanged between the main thread which runs the experiment and produces the event data, and the dispatcher thread which takes care of the data transmission to the consumers.

Although threads appear to be different processes on operating system level, they share the same memory. Memory allocated by one thread can be used by the other thread. Global variables set by one thread can be read and modified by the other thread, whereas local variables (variables that are defined in the scope of a function) are private. Memory that is allocated by the main thread to hold event data can be read by the dispatcher thread in order to send it to the consumers. Special care has to be taken whenever variables shall be modified by both threads. For example the number of queued events is a variable that is incremented by the main thread when new event data was produced, whereas the dispatcher thread decreases it after processing an event. Due to the fact that the two threads are working independently it may happen that the incrementation (which comprises reading the variable and writing back the new value) and the decrementation (which comprises the same two steps) interfere. In general, race conditions can occur whenever both threads modify the same variable.

To have a clean way of transferring the data from one thread to the other, a collection of helper functions was created with names beginning with `debt_` which stands for "data exchange between threads". Data exchange in this case means that one process adds entries to the back of a queue while another process retrieves and removes entries from the front of the queue. In this context 'entries' means pointers to event data structures as described in section 4.6. The `debt` functions realize this with ring buffers. A ring buffer has a finite, fixed size and therefore can only store a fixed maximum number of entries. This number can be chosen large enough to not run into problems with ring buffer overflows. A data structure named `debt_ringbuffer` is defined in `debt.h`. To circumvent the possibility of race conditions, each member variable of this data structure is written to by only one thread. Even when one thread is manipulating the

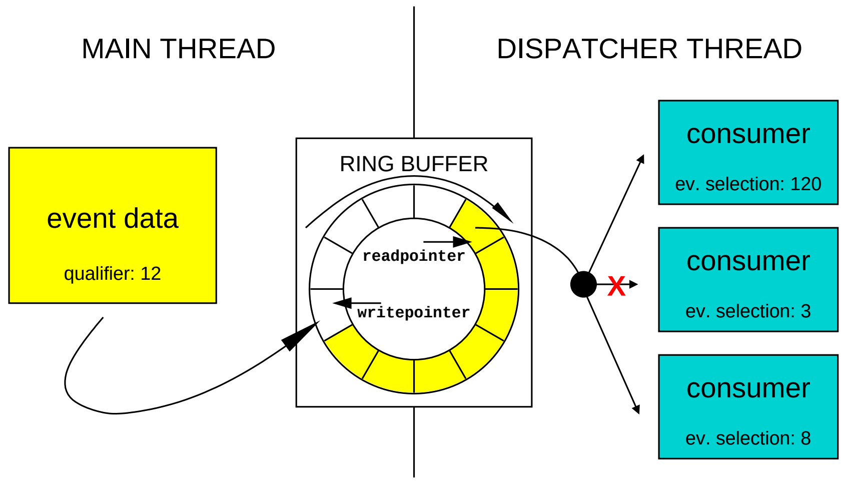


Figure 4.1: Data exchange between threads: The main thread adds a pointer to a data event structure in the ring buffer. The new entry is written to the position pointed to `writepointer`, before `writepointer` is incremented. The dispatcher thread retrieves a pointer to a data event structure from the position pointed to by `readpointer` (after which `readpointer` is incremented) and appends it to the transmission queues of the consumers that are configured to receive this event. The selection mechanism is described in section 4.10.

variables within that data structure, the order in which this is done ensures that all variables bear valid values for the other thread at any time. The `debt_ringbuffer` structure consists of the following variables:

- `int size`: depth of the ringbuffer in number of entries (set at initialization time),
- `void **entries`: pointer to an array of the entries, which are of type `void *` (set at initialization time),
- `int readpointer`: position to read the next entry from (modified by the receiving thread),
- `int writepointer`: position to write the next entry to (modified by the sending thread),
- `sem_t sem`: unused[4],
- `sem_t *setsem`: unused[5].

`debt.c` provides the following functions:

- `debt_ringbuffer_init`: returns a pointer to a newly created `debt_ringbuffer` data structure. The size is given as an argument to this function.

  After initialization of a ringbuffer, both `readpointer` and `writepointer` are zero,

[4]This is only used for the function `debt_ringbuffer_read_blocking` which not only checks but waits for a new entry in the ringbuffer. This was implemented in `debt.c` but became obsolete during the development, as `grandedaq` uses only the non-blocking function `debt_ringbuffer_read`.

[5]This is only used for "ringbuffer sets". This feature was implemented in `debt.c` but became obsolete during the development of `grandedaq`.

- `debt_ringbuffer_write`: adds a new entry to the ringbuffer. Return zero on success and a non-zero value in case the entry can not be added because the ring buffer is full.

  To add an entry to the queue, it is written to the position pointed to by `writepointer`, which is incremented afterwards. The ring buffer is full if adding an entry—and therefore incrementing `writepointer`—leads to equal values of `readpointer` and `writepointer`,

- `debt_ringbuffer_read`: retrieves and removes one entry from the ring buffer. Returns the `NULL` value if the ring buffer is empty.

  To retrieve an entry from the queue, it is taken from the position pointed to by `readpointer`. To remove it from the queue, `readpointer` is incremented.

In case the `readpointer` or the `writepointer` reaches the value of `size` it is set to zero again. As long as `readpointer` and `writepointer` have different values, there is at least one entry in the ring buffer. The number of entries currently in the ring buffer is `writepointer` − `readpointer` if this number is equal to or greater than zero, and `writepointer` − `readpointer` + `size` otherwise.

Since adding to and removing from the ringbuffer change different variables within the `debt_ringbuffer` structure, simultaneous adding and removing cannot lead to inconsistencies.

During the initialization of `grandedaq`, a ring buffer which stores pointers to event data structures queued for transmission is created. The size is set to 1,000. Due to the implementation of the dispatcher thread, this ringbuffer is emptied at least every two seconds (see section 4.10) Since the number of events per second is in the order of 10, the fixed size of the ring buffer should not be a limitation. As a matter of fact, `grandedaq` so far never reported a ring buffer overflow.

4.8 Initialization of the experiment

This section describes which actions are carried out in order to set up the Grande data acquisition hardware at the beginning of a run. The initialization is performed by the function `init_experiment` (implemented in `experiment.c`), which is called once when `grandedaq` is started. It consists of the following operations.

- Initialization of VME bus and CAMAC crates: a global reset command is issued for the VME bus and each CAMAC crate.

- Initialization of the real time clock module: A reset command is issued, and the programmable outputs are initialized. The veto output is set to HIGH. According to the command line options with which `grandedaq` was started, the programmable outputs, that define if Grande accepts external triggers and if Grande 7/7 coincidences are sent to the central KASCADE-Grande trigger distributor, are set.

- Setting the absolute time in the real time clock module: First the program waits for one signal at the 1 Hz input of the real time clock module. Then the seconds counter is set to the absolute time in UNIX time format (seconds since midnight January 1st, 1970) as retrieved from the GPS clock module. Afterwards both the GPS clock module and the real time clock module are read out, and the absolute time is compared to the one previously received. The whole procedure is repeated if (and as long as) these numbers do not match.

- Prepare the real time clock module for data taking: Pending interrupts and latched timestamps are cleared. The “enable event” bit of the real time clock module control register is set. Hence, the real time clock module is ready to start data taking as soon as the veto is switched off.

- Program coincidence units: If no special command line option was given to `grandedaq` (see section 4.4), the coincidence units are programmed according to the trigger logic as described in chapter 3. Alternatively, the programming of the coincidence units can be read from a file. This file has to contain 256 bytes for each trigger cluster which represent the $2^8 = 256$ possible combinations of input states. The least significant bit of the byte number corresponds to the state of the first input, the most significant bit corresponds to the state of the eighth input. The values are bit vectors: the six least significant bits of each byte define the output states, the remaining two bits are ignored. Though not all output lines of the coincidence units are used, by convention the first four outputs should report 4/7 coincidences and the remaining two outputs should report 7/7 coincidences. Therefore the only byte codes used are 0 (no coincidence), $2^0 + \cdots + 2^3 = 15$ (4/7 coincidence) and $2^0 + \cdots + 2^5 = 63$ (7/7 coincidence).

- Initialize TDCs: The `caenv767_init` function is called for each TDC. This function performs a software reset and sets the time window for recording logic run signals to $\pm 5\,\mu$s around the trigger signal.

  Although in the current experimental setup there is only one TDC module, the number of TDCs is defined in `experiment.h`, and `grandedaq` is written such that it can obtain the TDC data from any number of TDC modules.

- Initialize ADCs: For every ADC, the function `caenv785_init` is called, which issues a software reset command, clears all data and releases interrupts. The operation mode of the ADC is set such that for every gate signal exactly one ADC event is written to the ADC’s internal memory, containing entries for all ADC channels (overflow and zero suppression turned off, all thresholds set to zero).

- Initialize pattern units: For every pattern unit, the function `caenv259n_init` is called which performs a reset of the module.

- Initialize scalers: For every scaler, the function `caenc257_init` is called which issues a reset command.

- Initialize discriminators: The programmable discriminators used for selecting one logic run signal as single particle trigger are set up. The threshold for every channel is set to 100 mV and all channels are masked out, which means that no detector station can emit the single particle trigger. Whether single particle spectra are to be taken can be configured in `experiment.h`. Then, the first station is unmasked, thus providing the single particle trigger.

- Reset veto signal: At the end of this initialization process, the veto signal is finally released. This starts data taking.

After returning from `init_experiment`, `grandedaq` enters its main event loop, in which event and scaler read-outs are performed.

4.9 Event read-out

The event read-out is performed by the function `readout_experiment` (implemented in `experiment.c`) which is called by the `main` function once the real time clock module reports the reception of an event signal. In order to reduce dead time, all hardware read-outs are performed first and the veto signal is reset before the event data structure is filled, as this requires some computations. Each hardware component is brought to a clean state after read-out. A variable `discard` is defined inside `readout_experiment` which is set to zero at the beginning and is set to a non-zero value during the read-out, in case a component reveals a problem with the event currently being processed. Most of these problems are caused by coincidences of single particle events and shower events. This issue is described more in detail in section 4.11.

The hardware readout itself comprises the following operations.

- The programmable output no. 1 of the real time clock module is set to HIGH. This ensures an active veto signal until this programmable output is reset, independent from resetting the real time clock module's BUSY state.
- The data-ready flag of the (first) TDC is checked. In case that the TDC has not completed the data acquisition for this event by this time, a warning is printed and the check is repeated until the TDC reports data ready. This ensures that the following steps are not executed before the measurement devices finished data acquisition.
- The pattern units are read out. These provide information on which trigger clusters signalled a 4/7 coincidence, whether a 7/7 coincidence was detected by a coincidence unit, whether an external trigger was received and whether the read-out cycle was caused by a single particle trigger. The latter case will be discussed in section 4.11.
- The TDCs and ADCs are read out. In case a device reports that no data is available, the read-out is attempted up to 100 more times (this number is defined as `MAXLOOP` in `experiment.h`), before the `discard` variable is set to one. Since the availability of data was checked with the TDC at the beginning of the read-out procedure, this is a redundant check.

  If any of the devices reports that there is more data available, i.e. if it stored a second data package, `discard` is set to one, since it cannot be verified which of the data packages belongs to the shower event currently processed, and the remaining data in the device is cleared.
- The event timestamp is read from the real time clock module. BUSY and EVENT flags are reset.

At this point all data have been obtained from the hardware. The veto is switched off by resetting the corresponding programmable output of the real time clock module. From this moment, the DAQ hardware is able to record the next event. If the `discard` variable has been set to a non-zero value, the event is discarded and an corresponding log message is printed. In this case the program returns from the `readout_experiment` function without having sent an event to the consumers. Otherwise an event data package is built according to the structure described in section 4.6. An air shower event contains the `GRANDEHEAD` structure. The members of this structure are listed in table 4.2. The first four member variables (`magic_word`, `event_class`, `run_number`, `sent_event_number`) are common to all types of event data structure and have been discussed in section 4.6. Additionally, the `GRANDEHEAD` structure has the following members:

- `rtcm_seconds` and `rtcm_microtime`: The timestamp of the event, as read from the real time clock module, which gives the time of the trigger signal that caused the recording of this event,
- `trigger_pattern`: a bit vector indicating the trigger sources:
  - Bit 0 (value 1): set if the pattern unit reports an external trigger,
  - Bit 1 (value 2): set if the pattern unit reports a 7/7 coincidence,
  - Bit 2 (value 4): set if the pattern unit reports a 4/7 coincidence in any of the 18 trigger clusters.
- `hit_pattern_1`: A bit vector, indicating which of the detector stations 1–32 have TDC hits in the data. The least significant bit stands for station 1,
- `hit_pattern_2`: accordingly for stations 33–37,
- `_4_7_pattern`: A bit vector indicating which trigger clusters have a 4/7 coincidence. This is read from the pattern units,
- `_7_7_pattern`: A bit vector, indicating which trigger clusters have a 7/7 coincidence. This information is generated by `grandedaq` itself, since it is not available from any hardware device. The pattern units report only the 4/7 coincidences separately for the trigger clusters. For 7/7 coincidences there is only one pattern unit channel which reports if there is such a coincidence in at least one cluster.

  If the pattern unit does not report a 7/7 coincidence at all, `_7_7_pattern` is set to zero. Otherwise `grandedaq` checks for clusters with a 4/7 coincidence in which all seven stations have TDC hits. The time relationship of the TDC hits is not examined. Therefore this variable reports 7/7 coincidences for all trigger clusters that have logic run signals within the TDC gate (which is the time of the trigger signal $\pm 5\,\mu$s) in all seven stations, even if there was no real coincidence, i.e. the times differ too much to originate from one air shower,
- `length_of_data`: number of four-byte data words following this structure.

The data block following this header structure contains the data read from the ADCs and TDCs. For each of the 37 detector stations, data with the following format is added:

- One word of type `int`, containing the ADC value for the Pb*10 channel in the 16 least significant bits, and the ADC value for the Pb channel in the 16 most significant bits,
- one word of type `int`, containing the ADC value for the Pb1 channel in the 16 least significant bits, and the number of TDC hits for this station in the 16 most significant bits,
- the corresponding number of words of `int`, containing the TDC hit values.

Appended to this data block—like to the data of all event types—is one word of type `int` with a fixed (“magic”) value to mark the end of this event’s data. This word is also counted in the length of the data block as stored in `length_of_data` within the `GRANDEHEAD` structure.

A pointer to the event data is given to the ring buffer, so the dispatcher thread will transmit it to all consumers that are configured to receive shower events with the given trigger condition.

4.10 Communication with the central KASCADE-Grande DAQ

This section describes the communication with network consumers such as—but not restricted to—the central KASCADE-Grande DAQ, and also the management of disk consumers.

For network communication, TCP (transport control protocol) is used. TCP is a connection based protocol which is widely used. For example, most standard internet services (as World Wide Web or the standard e-mail protocol SMTP) use TCP for their network communication. Being a connection based protocol, it distinguishes between servers (processes that listen for connections) and clients (processes that initiate connections). In this sense, `grandedaq` is a network server, i.e. it waits for other processes to connect to it. According to the command line options with which `grandedaq` was started, it listens on zero or more TCP ports for connections. Once a process connects to one of `grandedaq`'s TCP ports, it receives data on this connection. Depending on the event qualifier mask of the port it connects to, it may receive only a certain subset of the data produced by `grandedaq`. The consumer can change its selection of events for an established connection and can also request a limited number of events. This feature is foreseen for data quality monitoring programs and is not used by the central KASCADE-Grande event builder.

The communication with network consumers is implemented in `dispatcher.c`. Network connections on Linux are closely related to files on disk: the same functions are used to read from or write to them, and they both are identified by "file descriptors". Therefore disk consumers are managed by the same functions in `grandedaq` as network consumers once they are set up.

The dispatcher code maintains a list of consumers (entries are of type `struct consumer`) and a list of open TCP ports at which it waits for connections (`struct dispatcherport`). The lists are initialized by the function `dispatcher_init` which is called once at program start. Afterwards, for every "`--file`" command line option that was given to `grandedaq` one entry is added to the consumer list by means of the function `dispatcher_diskconsumer`. This function sets up the consumer data structure for the new disk consumer, opens the file the data will be written to and adds the consumer to the global list of consumers. Disk consumers are only opened at the start of `grandedaq`, not during the run. They are closed in case of an error occurs while writing to the file, such as lack of disk space. For every "`--port`" command line option, the function `dispatcher_listenport` is called, which registers the TCP port with the given port number and adds a corresponding entry to the list of dispatcher ports. At this point, the function `dispatcher_thread` is launched as a separate thread and an corresponding message is printed on screen.

bit number	value	meaning
0	1	shower event with external trigger
1	2	shower event with 7/7 coincidence
2	4	shower event with 4/7 coincidence
3	8	Grande shower event (‘`GREV`’)
4	16	Grande scaler event (‘`GRSE`’)
5	32	Grande spectrum event (‘`GSPE`’)
6	64	Grande log message (‘`GRLM`’)

Table 4.3: Event qualifier bits: The qualifier of an event is calculated by adding up the values of the applicable features.

After initialization, the main thread of `grandedaq` enters the main event loop. Within this loop, the read-out of the measurement devices triggered by an air shower as well as the regular read-out of the scalers and the creation of single particle spectra are producing data events. The latter two are discussed in the following two sections. From now on the term “event” is used to refer to data packages that contain an event data structure. The dispatcher thread receives the data events via the ring buffer. As described in section 4.6, a book keeping data structure, which will not be transmitted to the consumers, is placed at the beginning of the data passed to the dispatcher thread. This structure contains the size of the subsequent event data, and also a bit vector that describes type and features of this event. The value of this event qualifier is the sum of the values given in table 4.3 for features that apply to this event. The value of this event qualifier allows to distinguish between different types of events (such as shower event, scaler event and so forth), and also between different trigger conditions in case of shower events. E.g. a shower event without a 7/7 coincidence or an external trigger has an event qualifier of $4+8=12$, a shower event with a 7/7 coincidence and an external trigger has $1+2+4+8=15$, whereas a scaler event has 16.

To select which data a consumer receives, each consumer has a numeric attribute formerly referred to as “event type selection”. For disk consumers, this event type selection is specified directly on the command line. Network consumers inherit it from the network port, for which it is also defined on the command line (see section 4.4). The event type selection mask is calculated in the same way as the event qualifier. By matching the event qualifier with the event type selection mask, it is determined which data events are sent to which consumers. If the bitwise AND of the event’s qualifier and the consumer’s event selection is non-zero, the event is sent to the corresponding consumer. Examples are given below.

- To set up a consumer which receives all events, an event type selection of $8+16+32+64=120$ is sufficient, since any event is of one of the four event types and therefore has one of these four bits set.
- The central KASCADE-Grande DAQ connects to a port which has an event selection number of $3=1+2$. Any event that has either a 7/7 coincidence or an external trigger has at least one common bit set. The KASCADE-Grande DAQ receives neither scaler events, spectrum events, log events nor shower events that only have 4/7 coincidences in order to limit the event rate for the joint data taking.

The function `dispatcher_queue` inserts an event to the individual transfer queues for every consumer which has an event selection number that qualifies for receiving that event. After calling `dispatcher_queue` for every event in the ringbuffer, the dispatcher thread executes the function `dispatcher_select` (named after the Linux system call `select`), which is the main function for communication with the consumers. The following tasks are performed by `dispatcher_select`.

1. By issueing the system call `select`, it waits for
   - a new connection request on an open network port,
   - data to be read from a network consumer, or (whatever happens first)
   - a consumer (disk or network) for which event data is queued and which is ready to receive data.

   If neither of the conditions are fulfilled within two seconds, the function `dispatcher_select` quits. The dispatcher thread then empties the ring buffer (and calls `dispatcher_queue` for every event) and executes `dispatcher_select` again.

2. For every connection request, the connection is established, and a new network consumer is created. The event selection number is set to the one of the port the connection was received on. The transfer queue of this consumer is initially empty.

3. Data is read from every network consumer for which the `select` call reported data available. This can either be an end-of-file notification, which means that the peer closed the network connection (in this case the consumer is deleted), or data that the opposite site sent to `grandedaq`. The latter case is only used to make it possible for network consumers to set their event type selection mask after the connection has already been established. To do this, the client process has to send three words of type `int`, the first being a magic number (representing the letters '`GCRQ`' which stand for '`grandedaq` configuration request'), the second being the new event type selection mask, and the third being the number of events to be received. If the latter number is zero (which is the default for network consumers) the number of events is not limited, otherwise only the requested number of events is sent to this consumer. After this number of events is reached, more events will only be sent if the consumer requests them by sending another configuration request.

4. Data is sent to consumers for which data is available and that are reported to be ready to receive data. Ready to receive data means that the operating system has finished the last write operation to this file or network connection. In case of a write error, caused by a broken network connection or a lack of disk space, the consumer is closed and deleted.

4.11 Recording of single particle spectra

For calibration purposes, the recording of single particle spectra is vital. A spectrum in this context is a histogram assigning a number of events to each ADC count value. The data acquisition software can select a station by means of the programmable discriminators. The logic run signal of this station serves as single particle trigger. One programmable output of the real time clock module determines whether the single particle trigger is suppressed or triggers the data taking and the read-out of single particle events.

In the data acquisition configuration file `experiment.h`, three macro declarations exist that define `grandedaq`'s behaviour while taking single particle spectra. The constant `SINGLEMUONPU` gives the number of the pattern unit channel to which the single particle trigger is connected. By setting it to zero, the taking of single particle spectra is completly disabled. The concept is that after every shower event a certain number (defined by `SINGLEMUONSPEREVENT` which is currently set to three) of single particle events of the same station is taken. After a fixed number of shower events (`SPECTRAEVENTS`, currently set to $5,000$) has been recorded, the next station is selected. The reason for counting the shower events in between while taking the spectrum instead of counting the single particle events themselves is that a station could be out of order, and `grandedaq` would never continue with the next station as it still tries to complete the spectrum of the malfunctioning station. Two global variables keep track of the status of the ongoing spectrum taking:

- `singlemuons` (type `int`) stores how many single particle events have been taken since the last shower event,

- `spectrumevents` (type `int`) is the number of shower events recorded since the beginning of the recording of the currently taken spectrum.

Like a shower event, a single particle event leads to a call of the `readout_experiment` function. After the read-out of the pattern units has been completed, it is checked whether the single particle trigger caused this read-out. If no single particle trigger was observed, the full read-out of the shower event is continued, and the variable `spectrumevents` is increased while `singlemuons` is set to zero. If `spectrumevents` reached the value of `SPECTRAEVENTS`, a spectrum event is build from the collected data, and the next station is selected as single particle trigger.

If the pattern units simultaneously report a single particle trigger and a shower trigger (a 4/7 coincidence or an external trigger), the complete event is discarded, because it cannot be decided if the read-out was originally caused by the single particle or by the shower trigger. It may occur that the second event happened so late that it was only recorded partially. The only safe conclusion is to discard both the single particle and the shower data.

If the pattern unit reports solely a single particle event, the variable `singlemuons` is increased, and in case it reached the value of `SINGLEMUONSPEREVENTS` no more single particle events need to be taken before the next shower event. In order to suppress the single muon trigger, the appropriate programmable output of the real time clock module is switched off. The function `readout_singlemuon` is called to perform the partial read-out of the measurement hardware: only the ADC of the selected station is read-out, all other components are cleared.

Spectrum events contain the data structure `struct spectrumevent`. The spectrum data in it are zero suppressed, i.e. ADC channel numbers that were not observed are excluded. Apart from the member variables discussed in section 4.6, the following variables are filled:

- `station_number`: number of the selected detector station,
- `spectrum_data[]`: array of `int` values, containing the ADC channel number in their 16 most significant bits, and the number of measurements of that ADC channel number in their 16 least significant bits,
- `spectrum_channels`: size (in entries) of the `spectrum_data` array.

4.12 Recording of scaler events

The running experiment can suffer from malfunctions like failing or hot detector stations. To monitor the performance of the detectors, scalers that are counting the logic run signals of the 37 stations, are read out once per second. Since the time between two scaler read-outs can differ from one second due to on-going shower or single particle event readouts, the measuring time is monitored using the same scalers by counting the 5 MHz timestamp signals. In addition, the logical AND of this 5 MHz signal and the veto signal is counted in order to measure the dead time. A scaler event is created at every read-out. In addition to the variables common to all event types, the scaler event structure (`struct scalerevent`) contains the following members:

- `fivemhz`: number of 5 MHz counts during this scaler measurement period,
- `deadtime`: dead time in units of 5 MHz,
- `scaler_channels`: number of scaler channel, i.e. number of detector stations (37),
- `scaler_data[]`: scaler values.

4.13 Summary

The Grande data acquisition software “`grandedaq`” is used for running the Grande array since July 2003. Since then, crashes have happened only in three cases, probably due to low memory. Since the summer of 2003, the Grande array is participating in regular runs of KASCADE-Grande. In operation, `grandedaq` has proven to reach the aim of stability. The next chapter will focus on the aims of speed (in terms of little dead time) and data integrity.

Chapter 5

Examination of Grande data

In order to demonstrate the successful operation of the data acquisition software, data obtained with that software is presented in this chapter. As this is raw and uncalibrated data, no physics conclusions should be drawn at this stage.

The data examined were taken as Grande run #211 in 2004 from March 15th, 12:30 until March 18th, 13:31. The central KASCADE-Grande data acquisition system included these data in its runs 4787–4792.

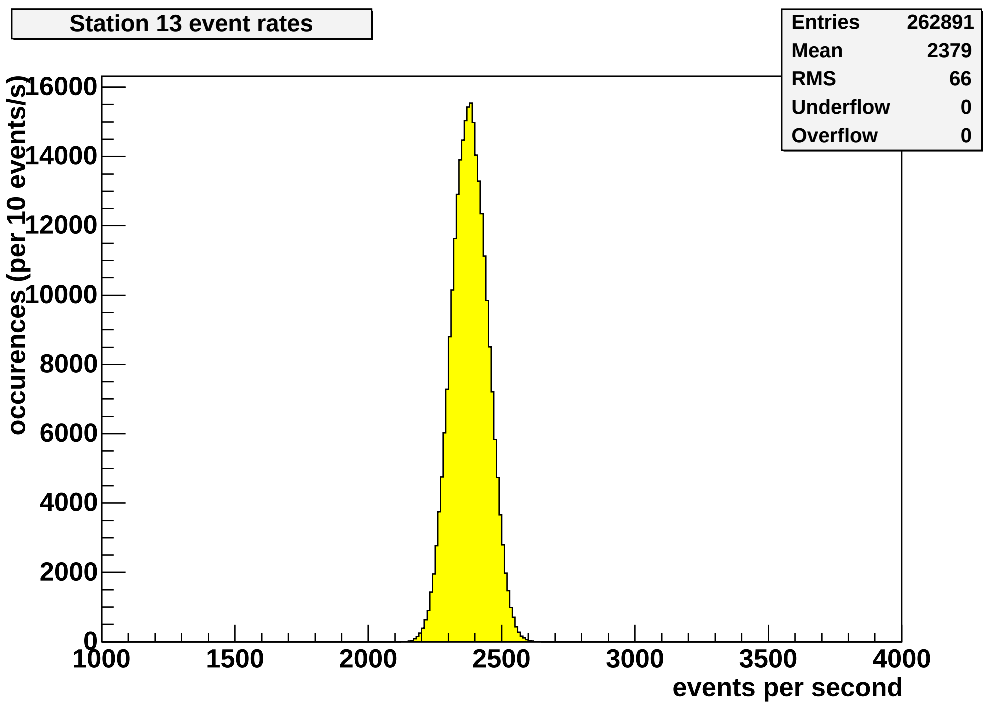
5.1 Single station event rates

Once every second, the event rates of the 37 detector stations are read from the scalers. The scaler measurements run continuously, and are not interrupted during the read-out of air shower events. Figure 5.1(a) shows a histogram of the results for detector station 13 as an example. The average number of events per second for this station is 2379, with a root mean square of 66. The full width at half maximum of this distribution is 160 events per second.

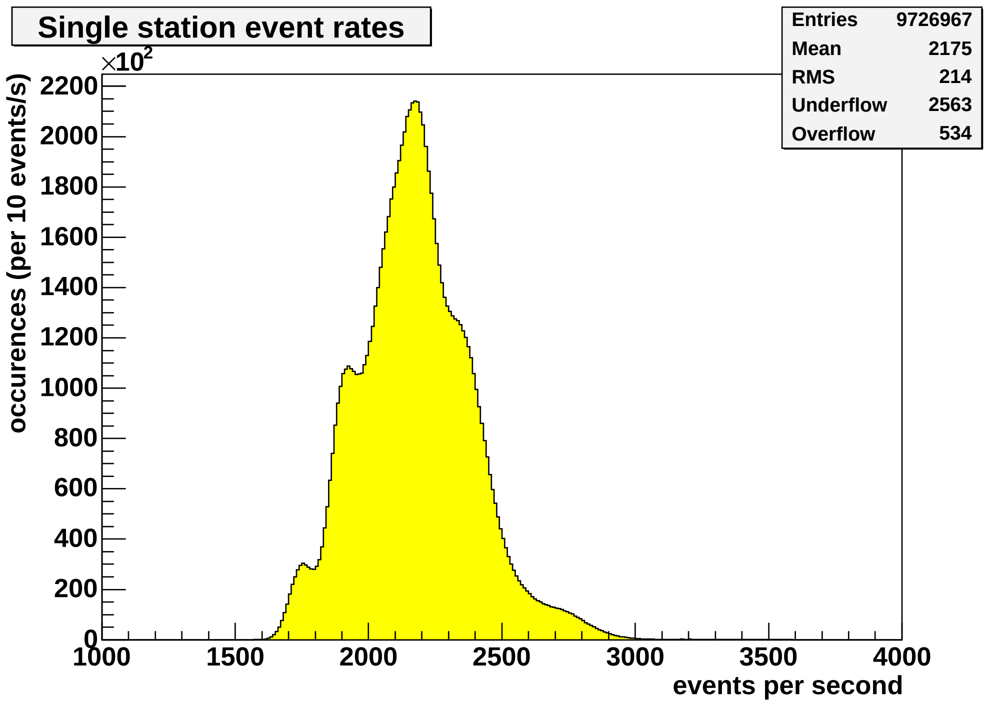
The recorded numbers of events per second differ for specific detector stations, due to different settings of high voltage of the photo multipliers and discriminator thresholds, as these have to be adjusted one by one during calibration in order to optimize the signal to noise relation. Figure 5.1(b) shows the superposition of the scaler measurements of all 37 detector stations, in order to give an impression of the range of different single event rates. The structure clearly shows an overlay of peaks at different positions for the different detector stations. For estimating data and event rates for the FADC system in chapter 6, the mean value from this histogram is needed, which amounts to 2175 ± 214 events per second.

5.2 Single particle spectra

The recording of air shower events is interleaved with the taking of single particle events. After each shower event, an ADC measurement can be started by the logic run signal of a given detector station. As the logic run signal denotes the detection of at least one particle in a detector station, the histogram filled with the so-obtained ADC values reflects the energy spectrum of the single particle events. Due to the high rate of particle detections in a single station in the order of 2,500 per second, as shown in the last section, as opposed to the rate of air shower events, which is in the order of a few per second, the vast majority of these single particle events is caused by uncorrelated particles. The Grande data acquisition software even discards single

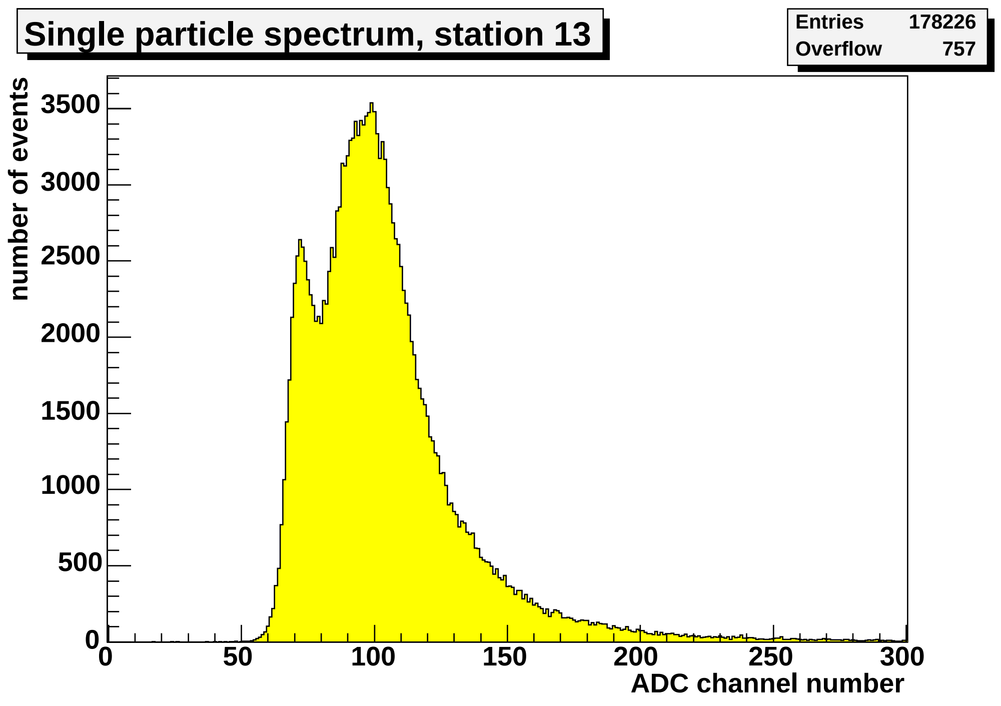


(a)

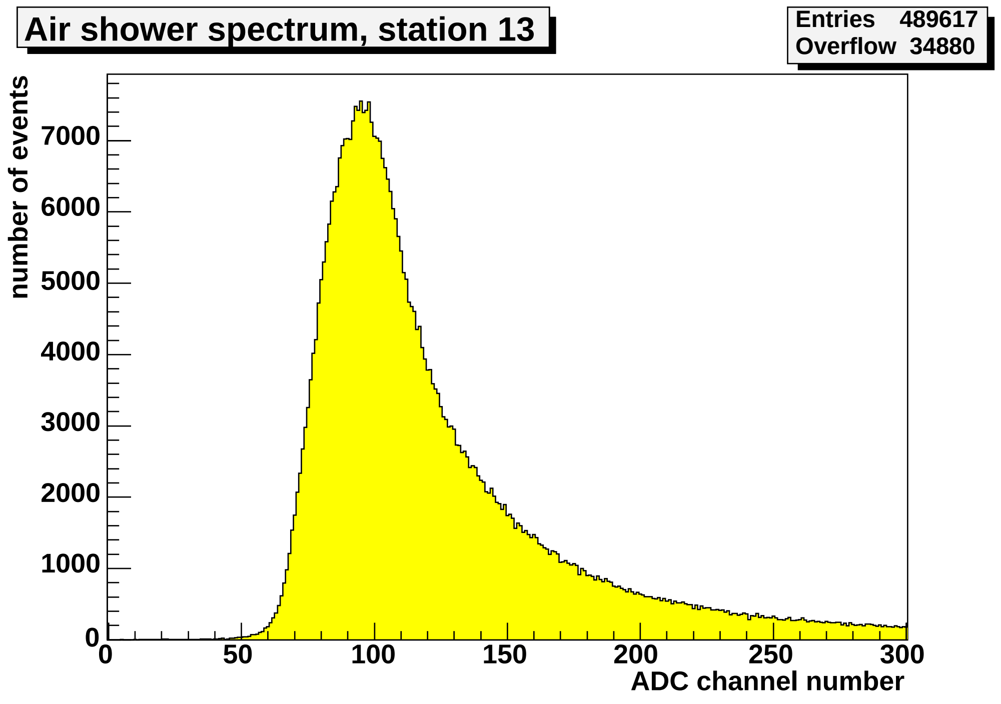


(b)

Figure 5.1: Single station event rates, as measured by scalers. Histogram (a) shows the number of events per second for detector station 13. Histogram (b) is filled with the single event rates of all 37 detector stations.



(a)



(b)

Figure 5.2: (a) shows the single particle spectrum for detector station 13. (b) shows the spectrum of the same detector station for air shower events. Included are all air shower events (independent from trigger source) during which detector station 13 signalled a particle detection.

station measurements that coincide with air shower events to avoid taking of ambiguous data (see section 4.11).

The taking of single particle spectra is necessary for calibration purposes. The spectra have a characteristic shape, resulting from the angular distribution of minimum ionizing muons penetrating the scintillator. Their mean energy deposition is a known value for the detector setup, which amounts to 10.3 MeV [Agl93] and can be related to an ADC channel in the spectrum.

Figure 5.2(a) shows the single particle spectrum of station 13. It shows the characteristic single muon peak, and also a shoulder towards the left hand side resulting from events that were triggered by noise. The cut-off from the discriminator threshold can be seen at the lower edge of the spectrum. This cut-off is not perfectly sharp, because of noise pick up during signal transmission, as the discriminator is located in the detector station whereas the ADC measurement is performed in the central DAQ station, after approximately 700 m of cable.

Figure 5.2(b) shows the spectrum of ADC values obtained in air shower events, for the same detector station. Included are all air shower events for which detector station 13 signalled a particle detection, independent from the trigger source. It lacks the noise contribution and shows a significantly larger fraction of events at higher energies, due to the fact that these measurements were performed when a certain number of detector stations detected particles in coincidence and thus the trigger conditions were fulfilled. By selecting only the ADC measurements that have taken place during air shower events, the fraction of single station events involving multiple particles is increased. Additionally, noise events are discriminated against real particle events.

5.3 Correlation between differently amplified ADC channels

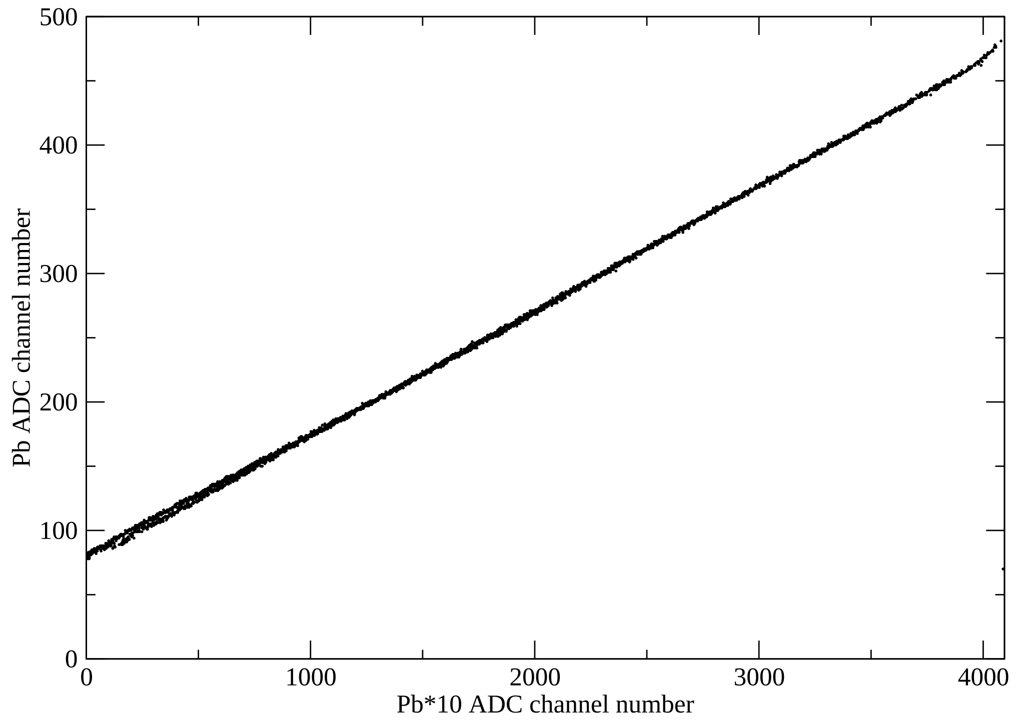
The plots in figure 5.3 show the correlation between the three ADC channels for detector station 1. They contain the ADC values from all air shower events in the examined run, in which station 1 signalled a particle detection by means of the logic run signal. Overflow values are not shown.

The signals Pb*10 and Pb are both formed by the signal shaper in the detector station from the same input signal (the mixed high gain photomultiplier signals), but amplified differently by a factor of ten. The Pb1 signal is shaped from low gain photomultiplier signals, which do not only have a smaller amplitude due to the lower voltage at which the photomultipliers are run, but also cover only one fourth of the detector area of the high gain photomultipliers. Therefore figure 5.3(a) shows a strict correlation between Pb*10 and Pb, whereas the correlation between the Pb and Pb1 channels—as presented in figure 5.3(b)—is less accurate, as the signals are generated from different photomultipliers.

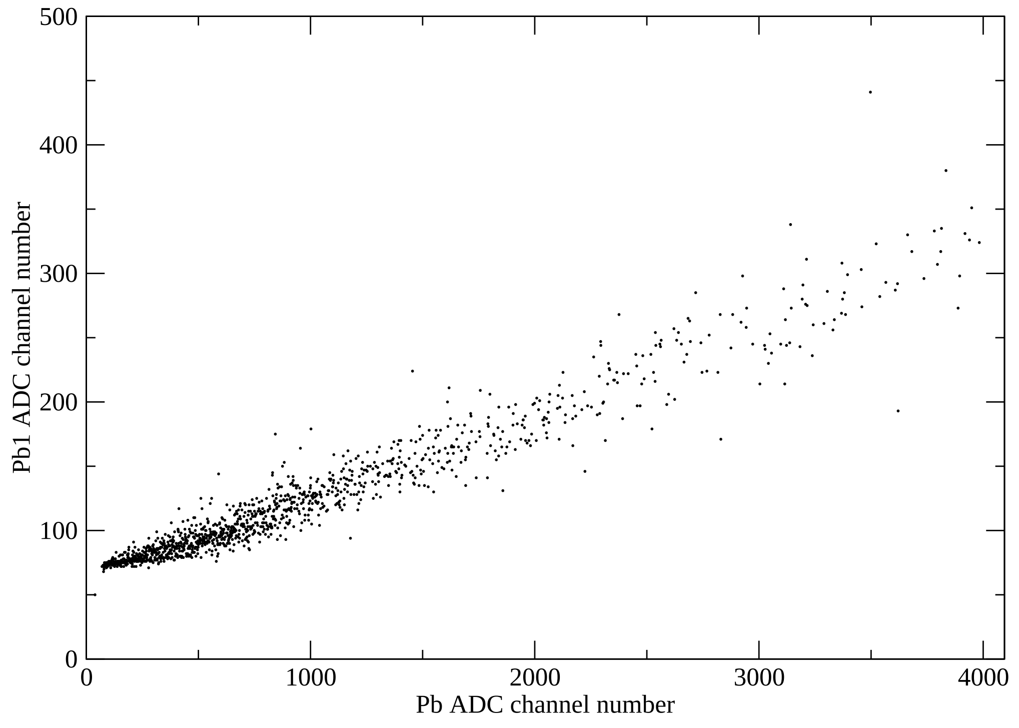
The three channels are read from different ADC modules. The plots presented prove, that the read-out procedure succeeds to obtain the ADC values belonging to the same air shower event. If this was not the case, the plots would show arbitrary combinations of ADC channel values, as the energy deposits in different air shower events are not correlated.

5.4 Station multiplicity

The plots in figure 5.4 show air shower event rates depending on the number of participating detector stations, for different trigger conditions. Figure 5.4(a) includes all events taken, whereas figure 5.4(b) and figure 5.4(c) include only those with a 4/7 or 7/7 coincidence (for definitions see section 3.2.1). In general, the event rate decreases with higher station multiplicities. Events with a small number of stations are suppressed by the trigger conditions, which aim to select

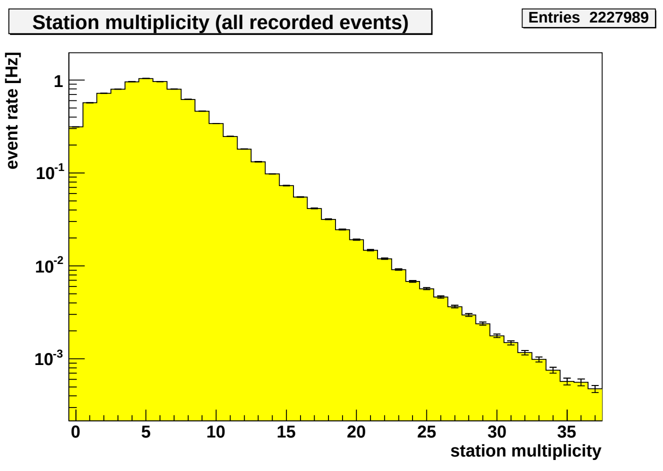


(a)

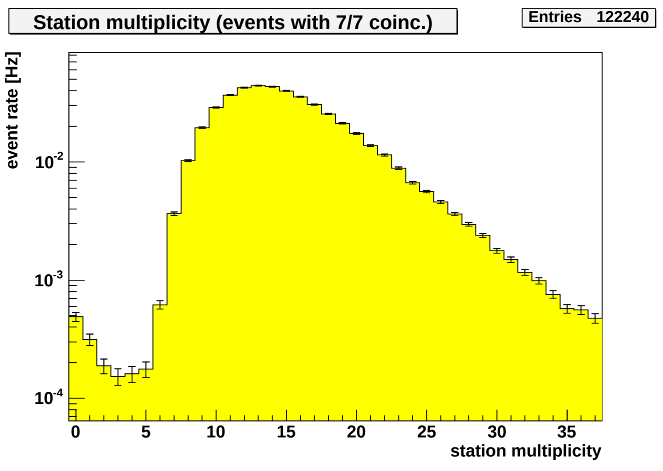


(b)

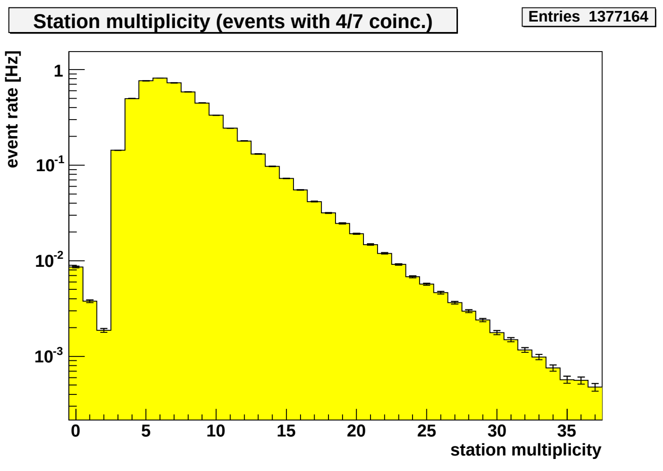
Figure 5.3: Correlation between differently amplified ADC channels: The correlation between the Pb*10 and Pb (both high gain) channels of detector station 1 is presented in (a). The comparison of the channels Pb (high gain) and Pb1 (low gain) of the same station is shown in (b).



(a)



(b)



(c)

Figure 5.4: Station multiplicity: (a) shows the event rate versus the number of participating detector stations for all events. (b) and (c) only include events with a 4/7 and a 7/7 coincidence respectively.

only showers of a certain minimum size. The trigger cut-off for the different trigger conditions can be seen at the left edge of the histograms. The first plot contains all events taken, including those which were externally triggered by other components of KASCADE-Grande. This class even includes events with no stations participating. This should not be the case for events with the 4/7 or 7/7 coincidence flag set, as these trigger conditions demand a minimum number of four or seven detector stations[1]. It was found that the coincidence units in the DAQ station emit electronic glitches, which lead to the taking of events although the trigger conditions were not properly fulfilled. These glitches often occur in coincidence with logic run signals, therefore a fraction of these mistriggered events have non-zero station numbers. The rate of clearly misidentified 7/7 coincidences—because less than six stations were involved—in the examined run was $(1.31 \pm 0.07) \cdot 10^{-3}\,\mathrm{s}^{-1}$. The rate of alleged 4/7 coincidences (with less than three stations involved) was $(1.37 \pm 0.02) \cdot 10^{-2}\,\mathrm{s}^{-1}$.

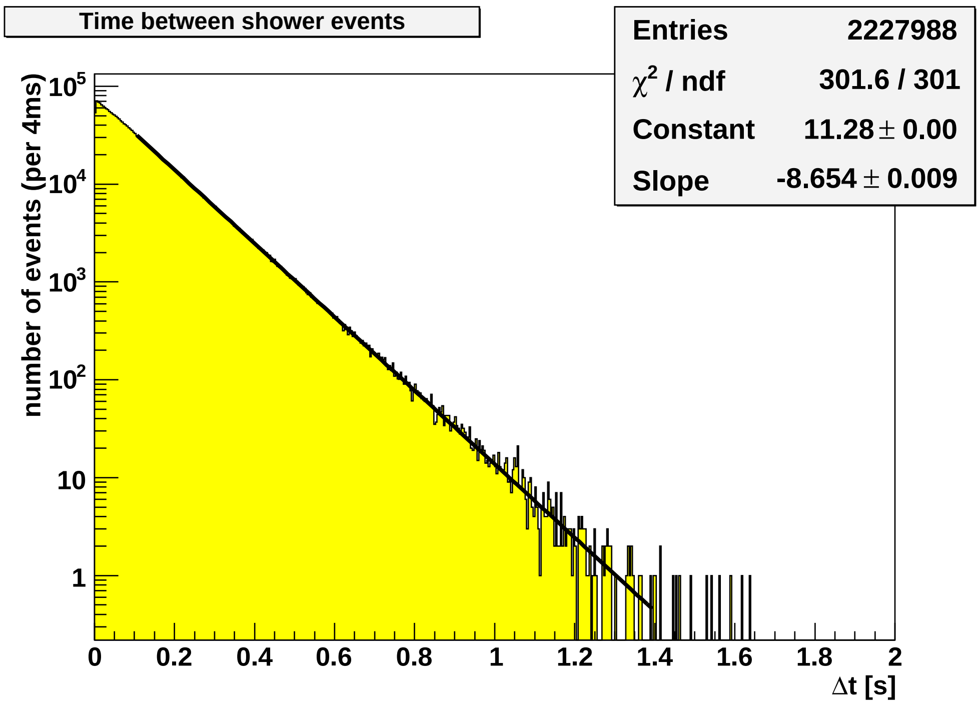
5.5 Time between shower events

Figure 5.5 shows the distribution of time differences between consecutive air shower events. Since air shower events are randomly distributed in time, the distribution has an exponential shape. As shown in figure 5.5(b), time differences less than $500\,\mu$s were not observed, which is due to the read-out time during which a veto signal is set that suppresses the detection of another air shower event. Additional dead time is caused by the read-outs of single station events, which take place three times after each shower event, triggered by a single station. As these do not happen at fixed times, they do not lead to gaps in the time distribution, but cause the measured distribution to contain less events for small time differences.

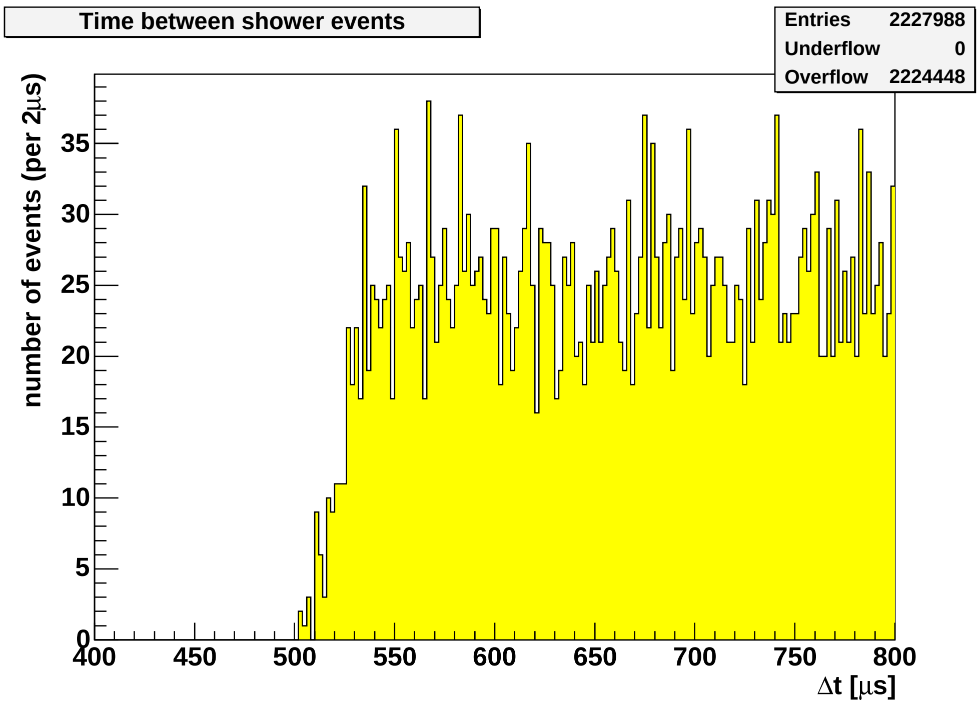
In order to check, how well the measured distribution agrees with an exponential behaviour, a fit was performed. The first 100 ms were excluded because of the bias due to the single particle events. At a rate of 2,500 events per second, the probabilty that the three single particle events are taken within 100 ms is practically 100%. The slope parameter—which corresponds to the event rate per second—was found to be $(8.654 \pm 0.009)\,\mathrm{s}^{-1}$. The fitted exponential function represents the data very well, according to the value of 1.00 for $\chi^2/n.d.f.$.

The total dead time in this run, as measured by the scalers, amounts to approximately 8.2 per mill. This corresponds to an average dead time of approximately $970\,\mu$s per recorded air shower event, which includes taking of the air shower event itself, as well as taking of up to three single particle events afterwards.

[1]Trigger cluster 15 only comprises six detector stations. A coincidence of all stations of this cluster also creates a 7/7 coincidence trigger signal. Therefore 7/7 coincidence events with only six participating stations can occur. In the examined run, this trigger cluster was still programmed to regard the missing detector station as constantly reporting particle detections, so that a 4/7 coincidence could be caused by only three stations. This was changed afterwards: Now this cluster also demands a full 4/7 coincidence.



(a)



(b)

Figure 5.5: Time between shower events: the time difference between two consecutive air shower events is shown. (a) presents the exponential relationship. Figure (b) is a zoom of the above distribution, showing dead time effects.

Chapter 6

The Grande FADC system

In addition to the installed data acquisition system, a FADC (flash analogue to digital converter) based system is currently being developed. In this chapter, the motivation and conceptual design will be discussed. The components will be introduced, and the data flow will be explained. All components store data temporarily in their memory. The consequence of the limited size of these memories, i.e. the expected rate of data loss, will be discussed at the end of this chapter.

6.1 Motivation

The Grande data acquisition system, as discussed in the previous chapters, is based on a global trigger. Once the coincidence logic finds evidence for an air shower event in the 37 logic run signals, which signal the detection of at least one particle in a detector station, a trigger for the whole Grande data acquisition system is generated. It starts the recording of this air shower event by the measurement devices and their read-out. Until the read-out is finished, no further event can be recorded. This causes a dead time of typically $500\,\mu$s after each air shower event.

Another cause of dead time is the taking of single particle spectra, which are vital for the calibration of the detectors. There is a trade off between high statistics and low dead time. On the one hand it is desirable to gain energy spectra with a sufficient number of entries in a short time, many times per day, in order to have a valid energy calibration for any time of the day while e.g. the temperature is changing. On the other hand, every recorded single particle event leads to an additional dead time of typically $250\,\mu$s. With the current configuration of the DAQ software, three single particle events are taken after every shower event. The recording of a spectrum for one detector lasts 5,000 air shower events, and typically takes 10 minutes. Since the spectra are recorded for only one station at a time, the collection of energy spectra for all 37 stations takes approximately 6 hours.

The data recorded for every air shower event contains for every detector station a triple of ADC values with different gains and amplifications, and zero or more TDC hits. The three numbers from the ADC translate into the deposited energy. The TDC hits—usually not more than one per station—give information on when the signal amplitude in the detector station exceeded the discriminator threshold, thus it denotes the arrival time of the first particle of the shower in that detector station. No information about the signal's time development can be obtained from these data. Resolving sub-showers that follow shortly after a first shower is impossible, as there is no time resolved information on the energy deposit. Apart from the total energy deposit and the arrival time of the first particle, the original photomultiplier pulses contain more information which is very valuable for the shower reconstruction.

The generation of trigger signals is accomplished by means of one programmable coincidence unit for each of the 18 trigger clusters. While it is possible to set arbitrary trigger conditions in the single coincidence units, the setup is still restricted to to the architecture of the 18 trigger clusters containing seven detector stations each.

While the current data acquisition system has proven to work and is already taking data, the given points leave space for improvements by a new data acquisition system. This leads to the following aims for the design of the FADC based system:

- Provide additional information on the photomultiplier pulses, such as their time development,
- improvement of the overall time and energy resolution,
- achieve a minimal dead time, or even dead time freeness,
- provide flexible setting of trigger conditions,
- allow for fast and continuous retrieval of energy spectra.

6.2 Design of the FADC system

To reach the previously mentioned aims, a completely new data acquistion system is currently under development. The new system is based on digitizer boards in all detector stations that continuously process the mixed photomultiplier signals and transmit zero suppressed data streams via an optical link to the central DAQ station. The components that make up this system are:

- 37 “**K**ASCADE-**G**rande **E**lectro**m**agnetic Detector **D**igitizer Boards” (KGEMD), which will be installed in the detector stations,
- five “**K**ASCADE-**G**rande **E**lectro**m**agnetic Detector **S**torage Boards” (KGEMS), to be installed in the Grande DAQ station,
- five first level PCs, located in the Grande DAQ station,
- five “**K**ASCADE-**G**rande **E**lectro**m**agnetic Detector **P**CI Interface Cards” (KGEMP), built into the first level PCs as interface to the KGEMS boards,
- one master PC, also located in the Grande DAQ station,
- one “**K**ASCADE-**G**rande **E**lectro**m**agnetic Detector **T**rigger Receiver Card” (KGEMT), built into the master PC.

In this new concept, there is no central trigger signal that starts data recording. The hardware in the detector stations is completely free running and self triggering. Once a particle is detected, the KGEMD digitizer board generates a data packet which contains the digitized pulse shapes of the mixed photomultiplier signals of one microsecond length. These data packets are transmitted via glass fibres to the central Grande DAQ station, where they are received by the KGEMS receiver boards and written to the memory of the first level PCs by means of the KGEMP interface cards. Due to the high data rates, it is impossible to feed the complete data of all 37 detector stations into one PC. Instead, a set of first level PCs will be used. Each first level PC handles the complete data of a subset of up to eight detector stations. Since uncorrelated particles make up the largest amount of the digitized pulses, the selection of single station events that take place

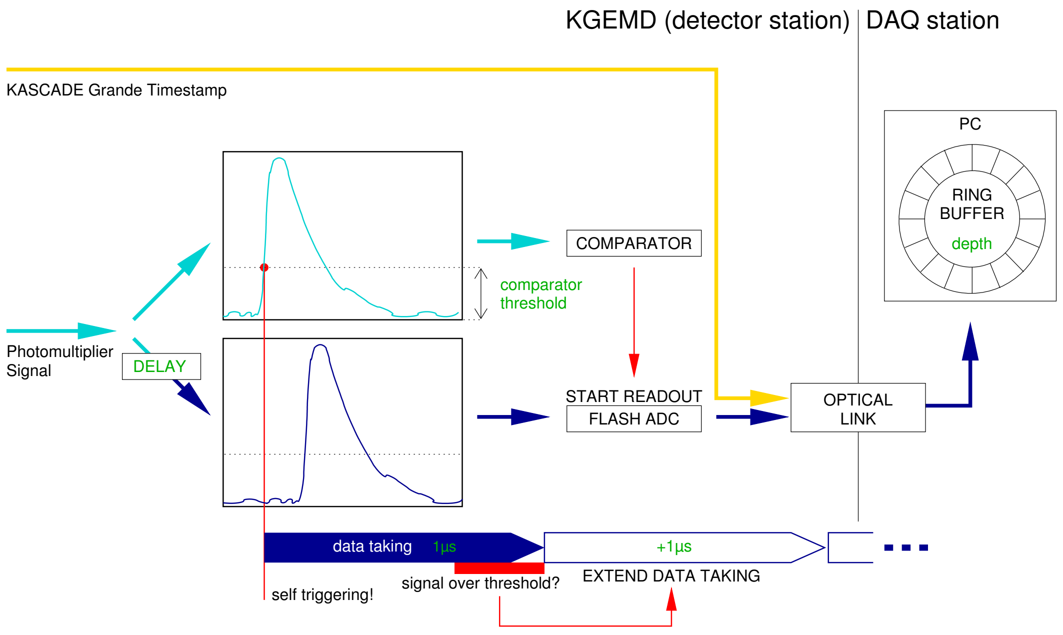


Figure 6.1: Schematic overview of the FADC concept (I): The digitizer board KGEMD starts the read-out of the FADC data whenever the incoming high gain signal exceeds a certain threshold. Due to an internal delay in the FADC, the data read from the FADC starts before the threshold was crossed. One digitization period lasts 1.008 μs. If the signal exceeds threshold again during the last part of this period, a second data taking period starts immediately after the first. The low gain signal is digitized in parallel. The digitized data are transmitted together with timestamp information via an optical link to the central Grande DAQ station.

in coincidence, and thus are regarded to form an air shower event, is the main task of the data processing in the DAQ station. Since none of the PCs has complete information about the whole Grande array, one additional PC—referred to as master PC—is needed, which receives information from all first level PCs on the time periods during which data was recorded by the digitizer boards. The master PC performs the event finding, and requests the full digitized data for air shower events from the first level PCs, which is then written to storage.

In the following sections, the components of the FADC system are discussed in detail.

6.3 The digitizer board (KGEMD)

The KGEMD board in each detector station digitizes two input signals with an effective sampling rate of 250 MHz. The input signals are the same mixed high and low gain signals that are fed into the signal shaper module of the existing DAQ hardware (see section 3.1). The FADC chip used[1] has a resolution of 12 bits and a sampling rate of 62.5 MHz. Unlike conventional ADCs, which usually integrate the input current for a certain time and measure the accumulated charge, flash ADCs contain one voltage comparator for every ADC channel. For a 12 bit flash ADC, this involves 4,095 voltage comparators. The responses of all comparators are encoded to a binary number. This way, flash ADCs are much faster and reach higher digitization frequencies than conventional ADCs.

[1]Analog Devices, AD9238

The effective sampling rate of 250 MHz is achieved by cascading four FADCs for each input signal. The input signal is split and distributed to all four FADCs, while the 62.5 MHz clock signal is shifted such that every 4 ns (this number corresponds to 250 MHz) one of the FADCs is sampling the input signal. The cascading of four FADCs to digitize one signal makes intercalibration of the resulting pulse shapes necessary. As the components used have finite tolerances, there may be an offset of amplitude between the four FADCs, as well as a slightly different amplification constant. While this effect is minimized by using high precision components on the digitizer boards, it is still necessary to apply offsets and correction factors in order to obtain comparable measurements by the four FADCs. These constants can be determined by digitizing a pulse with a known shape, e.g. a sine wave, and adjusting the intercalibration such that the obtained data agrees best with the shape of the input function.

The data are transferred via an optical link to the DAQ station. The link driver is also clocked with 62.5 MHz and transmits one 16-bit word with every clock cycle, resulting in a total transmission bandwidth of 1 Gbit/s. A continuous transfer of all data of the eight FADCs alone—not including control data such as timing information—would require a transmission speed of $8 \times 12\,\text{bit} \times 62.5\,\text{MHz} = 6\,\text{Gbit/s}$. Therefore it is necessary to suppress the transmission of zero data, which form the largest part of the data: While the FADCs digitize the signals continuously, the read-out is not started until the high gain signal exceeds a threshold. Internal data pipelines in the FADC chips provide a delayed read-out, so that the data recorded include samples digitized before the threshold was hit, which allows for the determination of the pedestal on a per-event basis. Independent from the shape of the signal, the digitization period lasts 63 cycles of the 62.5 MHz clock, which is slightly longer than one microsecond. Thus the size of a data packet produced by KGEMD is fixed and amounts to 512 words (16 bit), including meta data (packet header and timestamp). The structure is given in appendix B. The data taking is seamlessly prolonged for another microsecond in case the high gain signal exceeds the threshold again during the last 200 nanoseconds of the first period. A conceptual overview of the KGEMD board is shown in figure 6.1.

The timestamp, which is part of every data packet, consists of the values of an 1 Hz, a 5 MHz and a 62.5 MHz counter. The first two are incremented with every cycle of the common KASCADE-Grande 1 Hz and 5 MHz timestamp signals, which the digitizer boards receive via an optical link. The 62.5 MHz counter is incremented by the internal clock of the KGEMD board. The central 1 Hz signal resets the 5 MHz and 62.5 MHz counters to zero.

To ensure dead time freeness, a new event can be recorded even while the data transmission process of a previous event is still in progress. For this purpose, the digitization and the optical transmission are implemented as independent processes in the design of the KGEMD. The data of each FADC are written to one FIFO during digitization from which it is read during transmission. The timestamp values for the buffered events are stored in a FIFO that is implemented inside the central programmable logic chip of the digitizer card. This FIFO can store ten timestamps and therefore limits the number of queued events, since the FADC data FIFOs holds up to 32 events. The consequences of this limitation are discussed in section 6.9. The control logic of the KGEMD board is implemented in an FPGA[2] (field programmable gate array).

First test measurements with the KGEMD digitizer board have already been carried out on site of the Forschungszentrum Karlsruhe. Figure 6.2 shows one digitized photomultiplier pulse. The x-axis labels are sample numbers, which correspond to the time in units of 4 ns. The y-axis of each plot covers the complete range of ADC channels from 0 to 4,095. The upper two plots show raw data as received from digitizer boards, whereas the lower two plots show the same data after applying the intercalibration constants.

[2]Altera, FLEX 10K 30RC

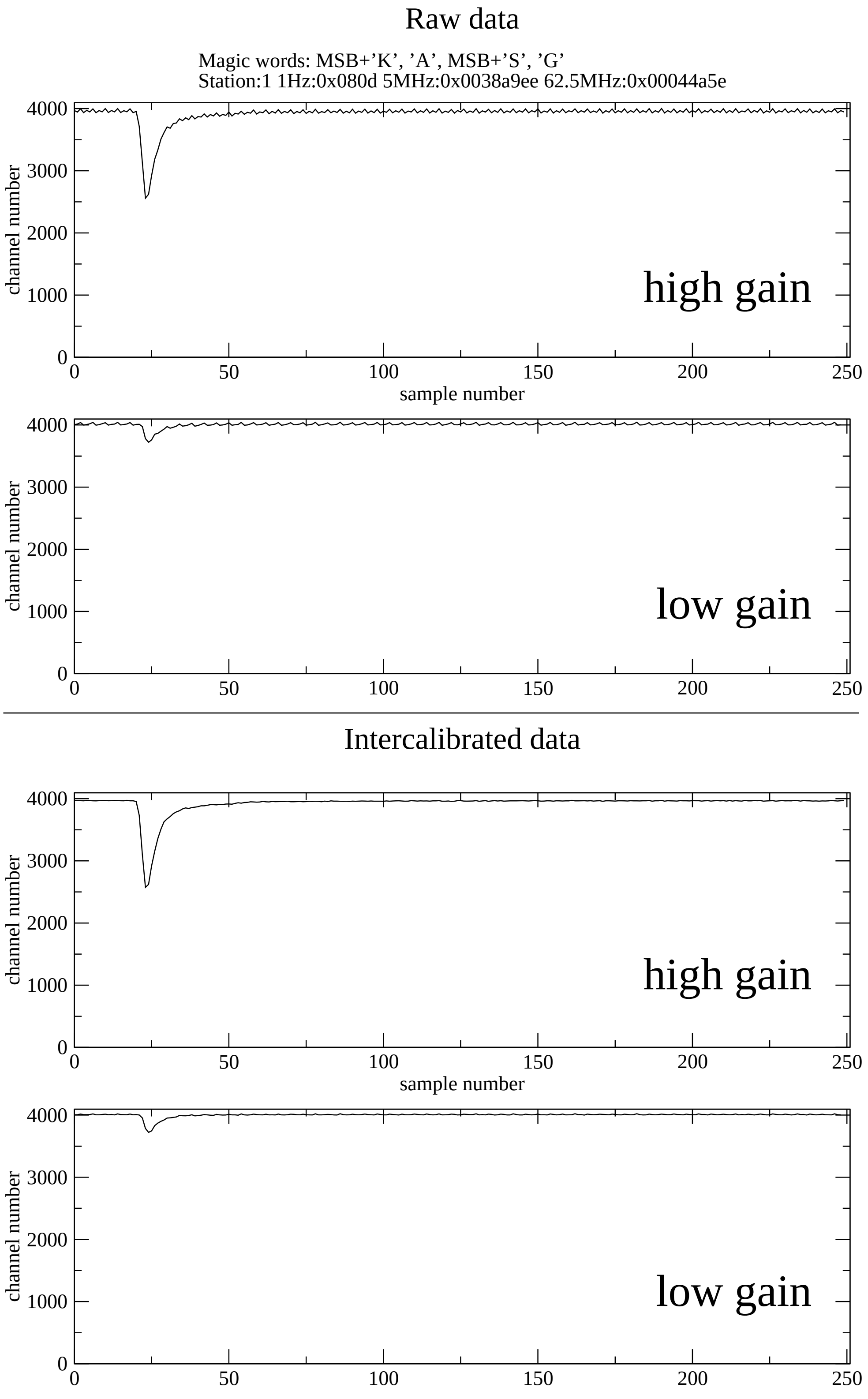


Figure 6.2: FADC sample pulse, taken from test measurements at the Forschungszentrum Karlsruhe. The upper two plots show the raw data received from the KGEMD board, the lower two present the intercalibrated pulse shapes [Zim04].

6.4 The storage board (KGEMS)

The central Grande DAQ station will be equipped with five VME storage boards (KGEMS), each of which multiplexes the data streams from up to eight detector stations into one common stream. On the input side, the KGEMS board is equipped with eight optical link receivers. The data received from each link is written to a FIFO which acts as a buffer for up to 32 data packets. The multiplexing is performed by an FPGA which assembles the output data packet-by-packet from the data in the input FIFOs. The resulting data stream is buffered in another FIFO, which can store 128 data packets, and sent to the first level PCs via a 32 bit wide LVDS (low voltage differential signalling) cable connection. An overview of the components in the Grande DAQ station, including the KGEMS storage boards, the first level PCs with their KGEMP interface cards, and the master PC with the KGEMT trigger receiver card, is shown in figure 6.3.

The KGEMS board can also be accessed via the VME bus for test purposes. Data can be injected to either the input or output FIFOs, and the read-out can be performed using the VME bus instead of the cable connection to the first level PCs. The read-out via VME only serves for test purposes, as the data rates during regular experiment operation are much to high to transfer the data streams over the VME bus. Additionally, control commands can be sent to the KGEMS board via the VME bus, such as clearing the data buffers or masking out one optical link from the data stream in case a detector station is "hot", i.e. its event rate is beyond reasonable limits.

The KGEMS board does not modify the data it receives and multiplexes. In particular, it does not add supplementary meta data. All necessary information is already contained in the data sent by the digitizer boards. E.g. the station number is sent by each KGEMD as a fixed word in the header and therefore the storage board does not need to add information on the link number it received a data packet from.

6.5 The first level PCs and the PCI interface card (KGEMP)

The five first level PCs together receive the complete data from all 37 detector stations. They act as ring buffers, as each first level PC writes the data into a dedicated part of its memory which is periodically overwritten with the latest data. In the current design, each first level PC will be equipped with 1 GB of memory, of which 128 MB are used by the operating system and the remaining 896 MB constitute the ring buffer. The data is received by means of a custom-made PCI interface card (KGEMP) which is connected to a KGEMS storage board (see figure 6.4). The KGEMP card is based on a commercial prototyping PCI interface card[3], which has been extended by a custom-made interface to the KGEMS board. It writes the data directly into the ring buffer memory, which is then accessed by the online software running on the PC. The task of this PC is to collect the timestamp information from all received data packets and to send a list of those, together with the number of the detector station and the address of the corresponding data in the ring buffer for each event, to the master PC. Additionally, the first level PCs integrate all received pulse shapes in order to obtain energy spectra. More information on the concepts of the online software of the FADC system will be given in the next chapter.

6.6 The master PC

The master PC receives all timestamps from the five first level PCs and therefore has the complete information on the times that events took place in the single detector stations. The master

[3]HK Meßsysteme GmbH, PCI-Proto Lab

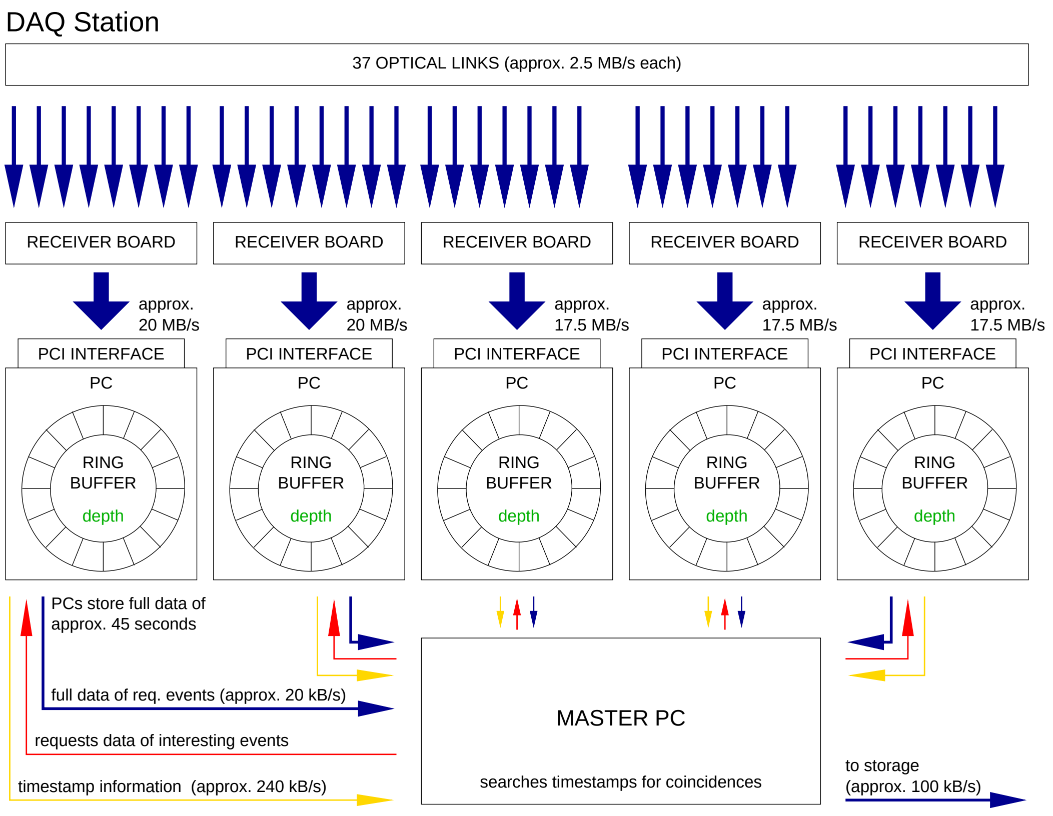

DAQ Station
37 OPTICAL LINKS (approx. 2.5 MB/s each)
RECEIVER BOARD
approx. 20 MB/s
approx. 17.5 MB/s
PCI INTERFACE
PC
RING BUFFER
depth
PCs store full data of approx. 45 seconds
full data of req. events (approx. 20 kB/s)
requests data of interesting events
timestamp information (approx. 240 kB/s)
MASTER PC
searches timestamps for coincidences
to storage (approx. 100 kB/s)

Figure 6.3: Overview schematic of the FADC concept (II): Five storage boards receive the data of the digitizer boards located in the 37 detector stations and transfer the resulting data streams to five first level PCs. These send timestamp information to a master PC which in turn requests the complete data of the interesting events. The shown data rates are explained in section 6.8.

PC scans these timestamp data for coincidences. The defintion of a coincidence in this context, which is given by the required number of participating stations, the necessary geometrical conditions, and the time interval within which the single station events have to take place, can be configured by the software. Once these requirements are fulfilled for a set of timestamps, the master PC requests the corresponding full data packets from the first level PCs. In addition to the single station events that fulfilled the software trigger conditions, it is possible to include all data within a certain time window around the time of the detected air shower event. The event building will be discussed in more detail in section 7.1.3.

6.7 The trigger receiver card (KGEMT)

In order to accept trigger signals from the central KASCADE-Grande trigger distributor, the master PC is equipped with a custom made trigger receiver PCI card (KGEMT)[4]. It has electric inputs for the common KASCADE-Grande 1 Hz and 5 MHz clock signals, and for the central trigger signal. To set the internal 1 Hz counter of the KGEMT card to the correct absolute time, this card is equipped with a Global Positioning System (GPS) receiver. The master PC reads

[4]Developed by Zentrum für Sensorsysteme (ZESS), Siegen, in collaboration with the experimental particle physics working group at the University of Siegen.

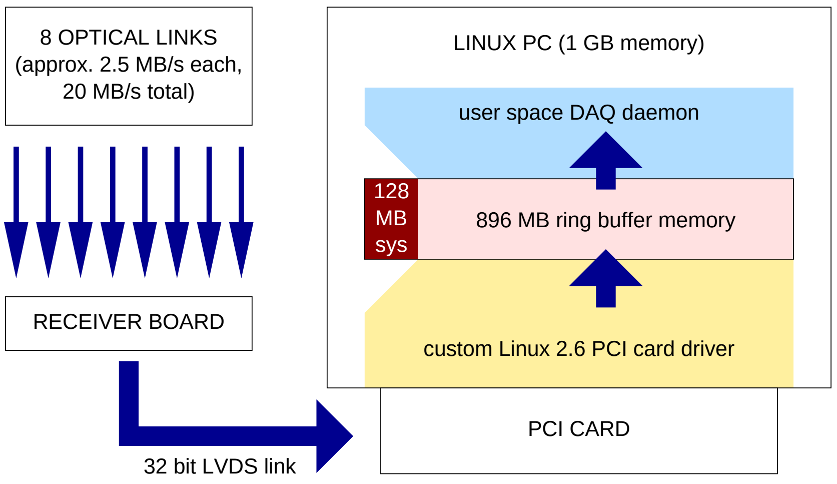


Figure 6.4: Implementation of the ring buffer on the first level PC

the clock counter values on which external trigger signals were received from the KGEMT card. The reception of an external trigger is treated like the fulfilment of the event finding conditions. An “external trigger” flag is attached to these air shower events which includes the time of the trigger reception.

6.8 Expected data rates

Considering the data taken by the currently running DAQ environment, as shown in chapter 5, the event and data rates for the FADC system are estimated. These estimates are based on typical measured values, since the exact numbers are influenced by many aspects, such as the threshold settings on the digitizer boards and the trigger conditions in the event building software, which are not yet fixed.

The single event rate per station, averaged over all stations, was measured to be $(2175 \pm 214)\,\mathrm{s}^{-1}$ (see 5.1). The error (RMS) reflects the spread of the mean values of the different detector stations.Including a safety margin of more than the RMS value, a number of 2,500 s^{-1} as design event rate is assumed for the calculation of the data and event rates. Since for each single station event a data package with a size of $1,024$ bytes is generated, this results in an output data rate of $2,500\,\mathrm{kB/s}$ for each digitizer board[5]. Each first level PC is connected to one storage board which receives the data of seven or eight detector station. The input data rate of one first level PC therefore amounts to approximately $20\,\mathrm{MB/s}$ ($17.5\,\mathrm{MB/s}$ for a storage board with only seven links connected).

The data transmitted from the first level PCs to the master PC consist of two parts: First, a list of timestamps of all events is received. For 20,000 events per seconds this makes up a data rate of approximately $240\,\mathrm{kB/s}$ for each first level PC, assuming a length of 12 bytes for each record consisting of the absolute time in seconds, the $62.5\,\mathrm{MHz}$ timestamp value, the station number and the address pointing to the data package in the ring buffer memory. Second, the complete data packages of single station events that took place in coincidence, as requested by the master PC. The data rate for the second part is determined by the rate of air shower events

[5]Unlike for SI-units, the prefix ‘k’ in the context of the unit byte (‘B’) stands for the number $2^{10} = 1,024$. Accordingly, one megabyte (‘MB’) is $2^{20}\,\mathrm{B} = 1,024\,\mathrm{kB}$, and one gigabyte (‘GB’) is $2^{30}\,\mathrm{B} = 1,024\,\mathrm{MB}$.

and their station multiplicity. The total rate of single station events requested by the master PC, R_{total}, can be calculated according to

$$R_{\text{total}} = \sum_{k=1}^{37} k \cdot R_k$$

with R_k being the rate of air shower events with k participating detector stations. To estimate this number, the event rates R_k measured by the current Grande setup can be used. These are shown in figure 5.4(a) and result in a total event rate R_{total} of approximately $51\,\mathrm{s}^{-1}$ for the run examined in chapter 5. This value, while already dead time corrected, still reflects the trigger conditions realized in the Grande DAQ hardware. Adding a safety margin to this number, and taking into account that the event recognition conditions can be chosen less restrictive than the current hardware trigger conditions, a design data rate of 100 kB/s seems reasonable.

6.9 Impact of finite memory sizes

The data rates previously calculated should be understood as averaged in time. The data sources do not uniformly produce data with that rate. Instead, the transmissions are intermittent, with peak rates much higher than the average rates. The digitizer board, for example, has an output data rate of 2,500 kB/s, but the transmission of the data generated during one average second, lasts only slightly longer than 20 ms. The storage board, which receives these data streams, is equipped with FIFOs which act as data buffers in order to derandomize the data flow. The digitizer board itself contains FIFOs to buffer the data, as the transmission of a data packet lasts longer than its generation takes. The PCI interface also has a data buffer, which is needed due to the fact that the input data stream is not synchronized with the availability of the PCI bus to the interface card. Accordingly, it has to buffer data until they can be transferred to the PC's memory. Since the arrival times of particles in the detector stations are randomly distributed over time, the digitizer board may already generate the next data packet while the transmission of the last one is not yet finished. Accordingly, the FIFOs in the other components may need to buffer more than only one event at a time. If data were generated or received while the target buffer was full, these data would be lost, which would practically result in dead time. In this section, the utilization of the data buffers on the KGEMD, KGEMS and KGEMP boards is examined by means of a computer simulation of the data acquisition.

A computer program was developed which simulates the data transport through the components of the FADC system in an idealized way. The simulation is accomplished in steps of one second, divided into bins of one nanosecond, for each of which the following steps are performed:

1. For each of the 37 KGEMD objects, 2,500 events are randomly distributed over the period of one second. Imitating the behaviour of the real KGEMD board, the transmission is started immediately as an event takes place and lasts for $8.192\,\mu$s. In case an event occurs while the transmission of another event is still in progress, its transmission is delayed until all previous events are processed. If an event is followed by the next event within the digitization time of one microsecond, it is either ignored (if occured within the first 800 ns of the digitization period) or appended to the end of the currently running digitization (in the last 200 ns). This imitates the prolongation of the digitization to a second event, as performed by the KGEMD board.

2. The events transmitted by the KGEMD boards are filled into 37 KGEMS input buffers. Each of the five KGEMS boards includes seven or eight of these inputs.

3. The events in the input buffers of one KGEMS board are multiplexed to its output buffer. The procedure imitates the function of the KGEMS board: first it is checked, which of the eight input buffers contains at least one event. Subsequently these buffers are emptied by transferring the data to the output buffer of the board. The list of buffers containing data is processed one by one. Buffers which receive an event during this procedure are ignored until the processing is finished and a new list of data containing buffers is built. The data transfers in this step are clocked with the same frequency as the optical data reception, therefore the transfer time is again $8.192\,\mu$s per event.

4. The data is transferred from the KGEMS output buffer to the KGEMP board at a speed of 20 million words (32 bit) per second, resulting in a transfer time of $12.8\,\mu$s per event.

5. The KGEMP interface card transfers the data from its input buffer to the memory of the PC. A data transfer rate of $40\,\mathrm{MB/s}$ is assumed, which leads to a transmission time of $25.6\,\mu$s per event.

The state of all objects at the end of the one second simulation period is stored and used as initial state for the next second. Thus, a continuous simulation of arbitrary length can be performed. The simulation only reflects an idealized situation. Neither cable delays nor hardware latencies are taken into account.

This data transmission chain contains four intermediate buffers: the KGEMD buffer, one buffer on the input side and one on the output side of the KGEMS board, and one more on the KGEMP board. The simulation program keeps track of the usage of all of these buffers—not only for one but for all 37 detector stations—and produces histograms, showing the fractions of time during which a given number of events is in that buffer. Figure 6.5 shows the histograms of the data transmission chain of one detector station for one day of simulated data taking. As presented in figure 6.5(a), during the simulated day of data taking, the buffer on the KGEMD board did not hold more than 4 events at any time. This simulation can make no further statement about the rate of higher fill levels, apart from not having them observed in a simulation of 24 hours. By extrapolating the values in the histogram, one can roughly estimate the probability of a full buffer. For the extrapolation from simulated time fractions (equated with probabilities) to the probability of a full buffer, an exponential relation between buffer fill level and its probability was assumed, as suggested by the shape of the histograms. By multiplying the so-obtained probability with the event rate and with one year, one gets the estimated number of events that take place during one year while the buffer is full, thus the number of lost events per year. This is done in table 6.1 for all four components of the data transmission chain. The resulting numbers are only a very rough estimate and may diverge from the real rates by orders of magnitude. Nevertheless, these numbers, being less than one event per year by many orders of magnitude, strongly suggest that the limited size of the buffers will never cause data loss.

In order to examine the effect of a "hot" detector station, i.e. a station which has an abnormally high event rate, another day of data taking was simulated. This time, detector station 1 generated 25,000 events per second, which is ten times the normal event rate. All other stations generated the normal rate of events. Figure 6.6 shows the histograms of this simulation. Unlike for the first simulation, this time data was lost. More than 700,000 events could not be transferred to KGEMP interface card, because its buffer was full. This leads to the necessity to exclude detector stations from data taking that show such malfunctions. The KGEMS boards provide this possibility by means of control commands that can be issued over the VME bus. By masking out the input channels belonging to the malfunctioning huts, the normal output data rate is retained and loss of data from the other detector station connected to the same storage board, is prevented.

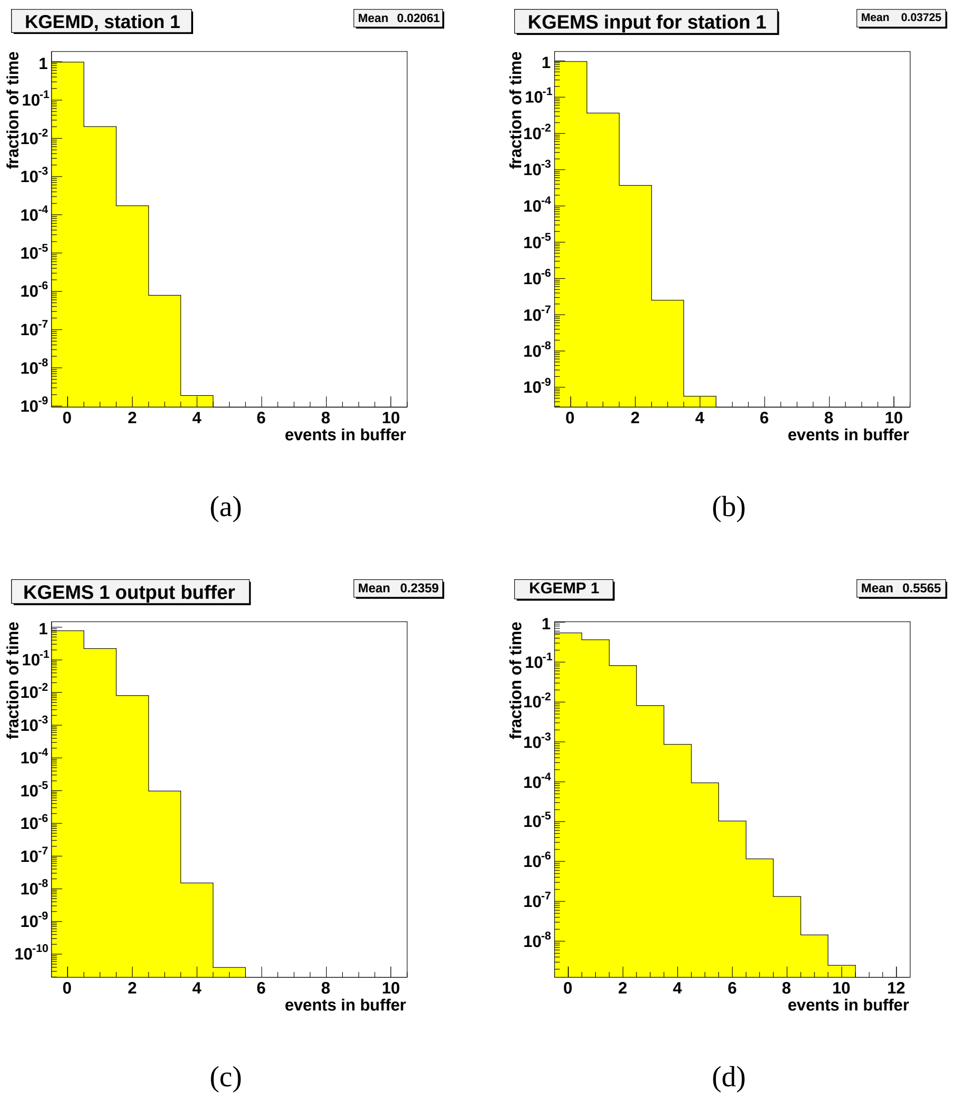


Figure 6.5: Buffer fill levels for one day of simulated data taking: The histograms for the data transmission chain of detector station 1 are shown. The KGEMS output buffer and the KGEMP buffer contain the multiplexed data streams of a full set of eight detector stations.

Component	Buffer size	$\log p_0$	$\log p_i$	$\log p_{\text{full}}$	Event rate	Lost events
	S [events]	(from Monte Carlo)		(extrapol.)	$[\text{s}^{-1}]$	$[\text{y}^{-1}]$
KGEMD	10	0	-8.7 $(i=4)$	-22	2,500	$1\cdot 10^{-11}$
KGEMS input	32	0	-9.2 $(i=4)$	-74	2,500	$1\cdot 10^{-63}$
KGEMS output	128	-0.1	-10.4 $(i=5)$	-263	20,000	$2\cdot 10^{-252}$
KGEMP	32	-0.3	-8.6 $(i=10)$	-27	20,000	$1\cdot 10^{-15}$

Table 6.1: Event loss rates: p_j denotes the probability that the buffer holds j events. The variable i represents the highest fill level observed.

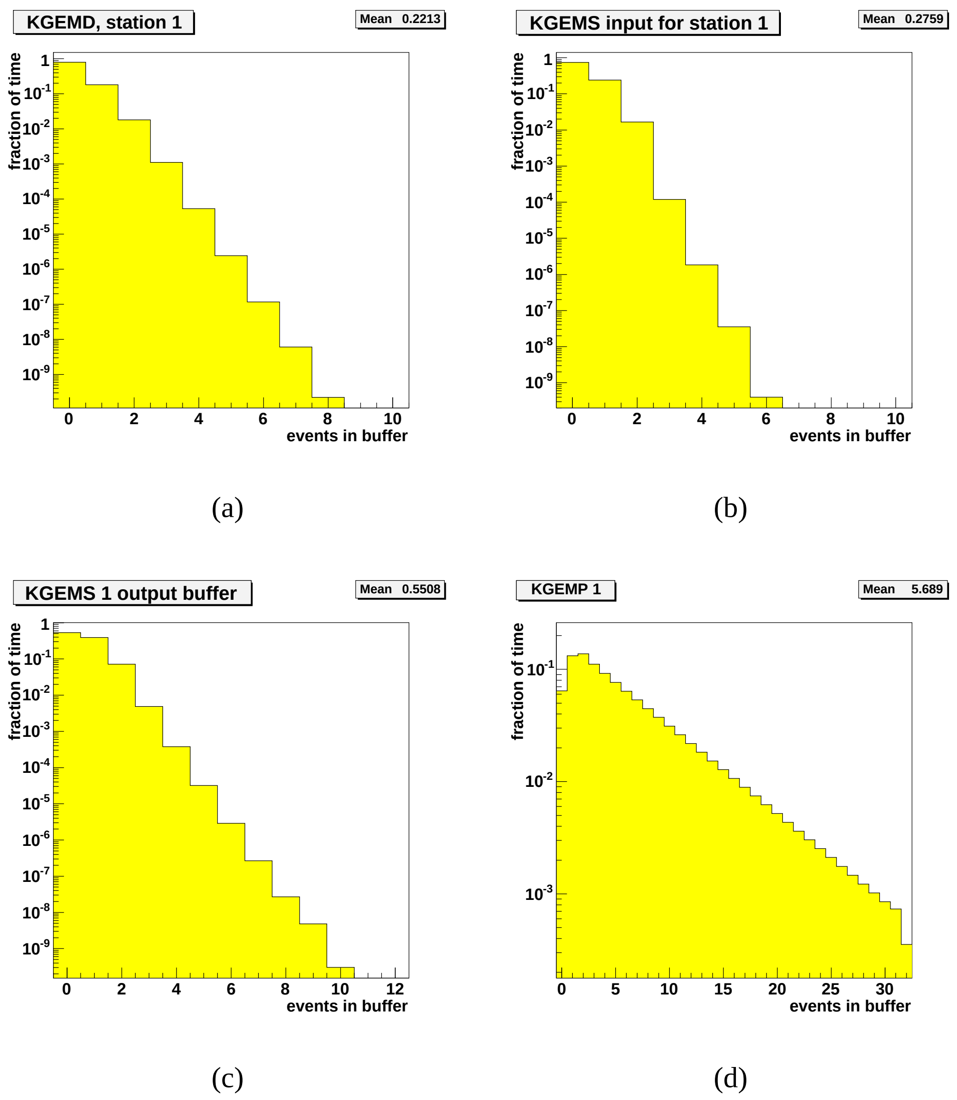


Figure 6.6: Buffer fill levels for a hot detector station: Again, one day of data taking was simulated. The histograms for the data transmission chain of detector station 1 are shown. A malfunction of this station is simulated, which leads to an event rate ten times as high as normal. The event rates of all other detector stations remain normal.

6.10 Summary

At the beginning of this chapter, five aims for the development of a new read-out system for the Grande array were presented. To summarize the concepts of the FADC system presented here, it will be discussed to what extent these aims have been met.

"Provide additional information on the photomultiplier pulses, such as their time development": The FADC system digitizes the mixed photomultiplier pulses in all detector stations. These shapes provide differentiated information on the time structure of the energy deposition.

"Improvement of the overall time and energy resolution": The sampling rate of 250 MHz corresponds to a time resolution of 4 ns. It can be significantly improved by applying fitting methods to the obtained pulse shapes. The resulting time and energy resolution is currently being investigated [And04].

"Achieve a minimal dead time, or even dead time freeness": The FADC system is continuously running and self triggering. The photomultiplier pulses are digitized continuously, and are recorded for periods of approximately one microsecond whenever the signal exceeds a threshold. After one period of data recording, the next one can follow immediately and seamlessly. The generated data streams therefore have full time coverage. Accordingly, the FADC system is free of dead time.

"Provide flexible setting of trigger conditions": The FADC system is not based on a global hardware trigger signal. The event selection rules, that determine which single station events are considered to belong to air shower events, are implemented in the data acquisition software. Changing these algorithms does not require changes to the hardware or manual intervention.

"Allow for fast and continuous retrieval of energy spectra": All single station events—not only those being part of air shower events—are available to the data acquisition software, which fills histograms with the energy deposits, calculated by integrating the pulse shapes. Creating energy spectra takes place for all detector stations at the same time. Making use of all 2,500 events per second and station, obtaining an energy spectrum for calibration purposes with as much statistics as the ones created by the current DAQ system, is a matter of seconds.

While the initially presented aims are all reached, there is one drawback of the FADC system, compared to the current DAQ system. As the event selection is performed in the software by analyzing the received data streams, no electronic event signal is generated at the time the event takes place. This renders the FADC system unable to provide a hardware trigger signal to the central KASCADE-Grande trigger distributor, in order to trigger other components of KASCADE-Grande. The current Grande DAQ environment triggers KASCADE-Grande with its 7/7 coincidence signal. However, this restriction does not impede the combined data taking of the Grande array with the rest of KASCADE-Grande. The central KASCADE-Grande event builder receives the event data of the different components of KASCADE-Grande, and combines them to one joint event if they coincide in time. Since the FADC DAQ system accepts the central KASCADE-Grande trigger, Grande will provide data not only for events that fulfil the internal Grande trigger conditions, but also for events triggered by other components of KASCADE-Grande.

Chapter 7

The Grande FADC data acquisition software

7.1 Concepts of the DAQ software

The data acquisition software for the new FADC system consists of two parts: The first level PC and the master PC online software. It has not been developed yet, but certain aspects of its design arise from the concepts of the FADC system. These aspects will be discussed in this section. The hardware related part of the first level PC online software, namely the driver for the KGEMP PCI interface card, was developed as part of this thesis and will be described in detail in the next section.

The online software has to process more than 90,000 single station events every second. The vast majority of those are uncorrelated events. As shown in the last chapter, the number of single station events that are part of air shower events which fulfil the trigger conditions of the running Grande setup, is in the order of 50–60 per second. Therefore, one task of the data acquisition software is the selection of events with interesting pulse shape information—those of air shower events with a number of participating detector stations that allows for shower reconstruction—from the bulk of 90,000 received data packets per second. Further tasks are listed below.

- Determination of the absolute time: From the timestamp information that the digitizer board sends with every data packet, the absolute time of the event must be determined.

- Acquisition of energy spectra: For each detector station, a histogram has to be filled with all single event energies for calibration purposes.

These tasks can be classified in two categories: First, tasks that involve processing the complete set of data of a detector station, and second, tasks that only work on a selected subset of data. Tasks of the first category can only be carried out on the first level PCs, which buffer the full data streams of seven or eight detector stations. The acquisiton of energy spectra and the adjustment of the received timestamps to match the absolute time are examples for actions in the first category. The second category of tasks run on the master PC, which receives pulse shape data only on request from the five first level PCs, but gets complete lists of timestamps for all events from all first level PCs. This category includes the event finding and event building. In the following subsections, the three given tasks are outlined. In the future, more online data processing might be implemented, e.g. correction of the pulse shape data for low frequency noise. Future improvements may also include additional online reconstruction of shower events.

7.1.1 Absolute time determination

The timestamps sent by the digitizer board with the pulse shape data, do not include an absolute time. They consist of the values of the 1 Hz, 5 MHz and 62.5 MHz counters. From the central Grande DAQ station, clock signals are sent via optical fibres to all detector stations, which increment the 1 Hz and 5 MHz counters on the digitizer board. The 62.5 MHz counter is incremented by an internal clock. Both the 5 MHz and the 62.5 MHz counters are reset with every 1 Hz clock pulse. The timestamp values sent by the digitizer board have to be corrected in two aspects: First, the delay of the clock signals from the clock signal distributor in the DAQ station to the particular digitizer board must be added. Second, an offset must be added to the value to the 1 Hz counter. This is because the digitizer boards do not receive clock values every second, but only a clock signal. The digitizer board starts with a counter value of zero at an arbitrary time. Furthermore, due to the limited width of the counter of 15 bits, it rolls over to zero after 2^{15} seconds (approx. 9 hours), which has to be corrected for.

The online software on the first level PCs has to determine the relation between the 1 Hz counter values of every digitizer board and the absolute time. This can be done by clearing the buffers on the storage board and examining the timestamp of the first event received afterwards. Taking into account the running time of the optical signals (not more than $3\,\mu$s in each direction) and the theoretical possibility that the digitizer board had to transmit a full buffer of events, before the event examined was sent (which takes $10 \times 8.192\,\mu$s plus $8.192\,\mu$s for the transmission of the event itself), an upper limit for the age of the timestamp received at the time of the reset command can be estimated. This leads to the conclusion, that the reset command must not be issued within the first $100\,\mu$s of a second (with respect to the 1 Hz signal) in order to ensure that the event received was not generated in the previous second. All events received until the end of the same absolute second are assigned the timestamp of that second, which determines the offset between the 1 Hz values of the timestamps and the absolute time.

The online software running on the first level PCs has to transform the timestamps of all events received to absolute times accordingly.

7.1.2 Acquisition of energy spectra

The energy deposited in a single station event is calculated by integrating its pulse shape. The result is a number in arbitrary units of amplitude multiplied by time. To relate these numbers with energies, the spectrum of the integration results is needed. The spectrum has a characteristic shape, as seen in section 5.2, from which the integration result corresponding to the most probable particle energy can be extracted, which is a property of the detector setup. The online software continuously has to integrate all pulse shapes received and fill one histogram per detector station with the corresponding integration results.

Different methods of numerical integration of typical pulse shapes are being investigated with respect to their accuracy and computation time [And04]. According to this, the trapezium method will be used, in which adjacent data points are connected by straight lines, so that any pair of adjacent points and their projection to the x-axis form a trapeze.

7.1.3 Event building

The online software running on the master PC receives lists of timestamps of all single station events from the five first level PCs. The search for air shower events within the set of single station events is based on the coincidences in time of those single station events being part of air shower events. The rate of 2,500 events per second in a single detector station is mostly

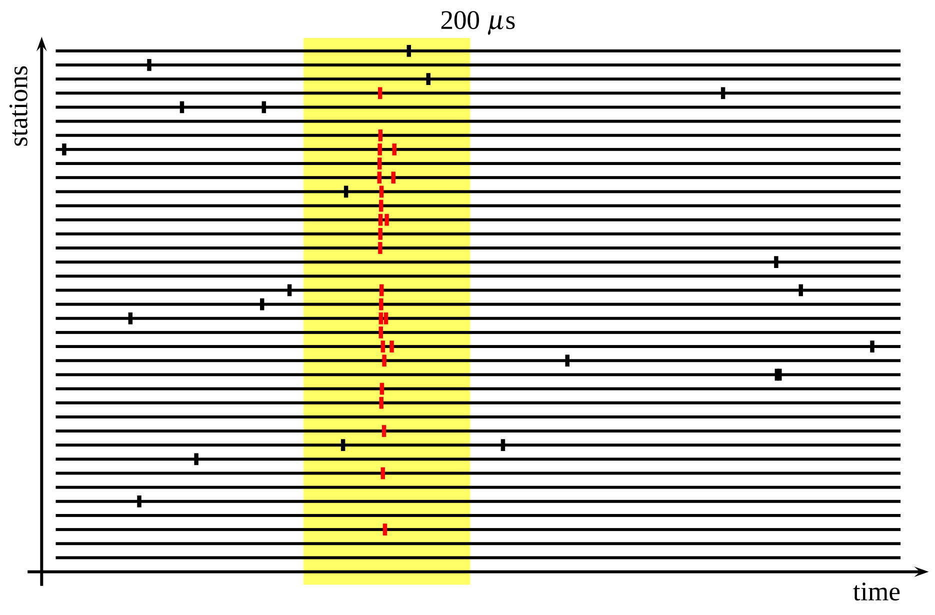


Figure 7.1: Illustration of the event finding: Displayed are 37 horizontal lines, each representing one detector station. A time span of 200 μs is counted along the x-axis. The black markers stand for digitization periods of one microsecond, caused by the background of 2,500 uncorrelated events per station and second. The red markers represent the measured relative times of one air shower event taken from Grande data, in which 21 of the 37 detector stations were hit. The shaded area marks a time window of $\pm 20\,\mu$s around the air shower event detection time.

made up of uncorrelated events, since the number of interesting air shower events is in the order of a few per second only. These uncorrelated events constitute the background for the detection of air shower events. While any of the detector stations is hit by an uncorrelated particle on average every 400 μs (this number corresponds to the rate of 2,500 s^{-1}), an air shower takes place within 3 μs, due to the geometry of the Grande array. This ratio is illustrated in figure 7.1. It shows the randomly distributed background events in the 37 detector stations for a time range of 200 μs. One air shower event—taken from Grande data—is added to the figure. The online software can find such occurences by scanning the timestamp list for a given number of timestamps within a defined time span. This very simple selection algorithm lacks any geometric constraints such as the trigger cluster structure of the running DAQ system. In order to evaluate whether this selection condition is sufficient to separate air shower events from the background of uncorrelated particle detections, a calculation of the rate at which this condition is fulfilled accidentally by background events has been performed.

The following simplified model can serve to calculate an upper limit for the rate of random coincidences: The background events are distributed equally in time. The probability distribution for the time between two events is an exponentially falling curve, with the event rate as parameter. The 37 detector stations are not considered separately, instead a total event rate of $37 \times 2,500$ events per second is assumed. By multiplying the probability, that $N-1$ single station events follow within a given time window after any event, with the total event rate, the rate at which $N-1$ events follow an initial event within that time window is obtained. This number serves as an upper limit for the rate of random coincidences of at least N stations. For two reasons, this overestimates the rate of alleged air shower events fulfilling the trigger conditions: First, since the detector stations are not accounted for separately, successive events in

N	2	3	4	5	6	7	8	9
$R_N\left[\mathrm{s}^{-1}\right]$	$2.2\cdot10^{4}$	$2.9\cdot10^{3}$	$2.6\cdot10^{2}$	$1.8\cdot10^{1}$	$1.0\cdot10^{0}$	$4.6\cdot10^{-2}$	$1.8\cdot10^{-3}$	$6.3\cdot10^{-5}$

Table 7.1: Upper limits for random coincidence rates: The rate of an N-fold coincidences of uncorrelated events (event rate 92,500 s^{-1}) within a time window of $3\,\mu$s is not higher than R_N.

one detector station are considered a coincidence, although the event builder would not. Second, coincidences of more than N single station events are counted more than once: for a coincidence of $N+1$ events within the time window, both the first and the second event fulfil the condition, that at least $N-1$ events follow within the given time span. While this would still be considered one air shower event by the event builder, the rates calculated with this model reflect that these are counted twice. Accordingly, $N+j$-fold coincidences are counted $j+1$ times. Thus, the model does not correctly implement the given event builder logic, but as the calculated rates are definitely too high, they form a valid upper limit.

For a random N-fold coincidence within a time window of T, the obtained upper limit is

$$R_N = \frac{r^N}{(N-2)!}\cdot\int_0^T t^{N-2}\exp(-rt)\,\mathrm{d}t$$

with r being the single event rate in the whole Grande array. For the derivation of this formula, see appendix C.

For an assumed event rate of $r = 37\times 2,500\,\mathrm{s}^{-1} = 92,500\,\mathrm{s}^{-1}$ and a time window width of $T = 3\,\mu$s, the random coincidence rates are given in table 7.1. In addition, the requirement of a sevenfold coincidence suffices to have less than one random coincidence in 20 seconds. The online software might be extended by more event selection checks, resulting in more restrictive trigger conditions, in order to better select only interesting events in terms of energy. In order to cut the rate of recorded events caused by random coincidences to an negligible number, this simple event selection scheme is adequate.

Additionally, the master PC receives the global KASCADE-Grande trigger signal. The trigger receiver card (KGEMT) provides the online software with timestamps of the usual 1 Hz and 5 MHz format. The external trigger is handled by the online software like a real coincidence at the given time.

Once a coincidence is found or an external trigger is received, the master PC requests the corresponding data packets from the first level PCs. The single station events to be obtained are selected by means of a time window around the detection time (coincidence or trigger time). This time window can be significantly larger than the $3\,\mu$s used so far. As an example, in figure 7.1, a time window of $\pm 20\,\mu$s is marked. Keeping the full information on all single station events in this window generates an additional overhead caused by background events of approx. $92,500\,\mathrm{s}^{-1}\times 40\,\mu\mathrm{s}\approx 4$ data packets per air shower event on average.

The parameters of the event builder, including the number of stations needed for accepting an air shower event, and the widths of the time windows both for coincidence detection and for data recording, are not yet fixed. These will be adjusted as the FADC system will be commissioned. Since these parameters are set in the online software alone, no hardware changes are necessary to adjust either parameters or even the algorithm of event selection.

The recorded data of selected air shower events is written to mass storage, and can also be sent to the central KASCADE-Grande DAQ for combined data taking with other components of KASCADE-Grande.

7.2 The KGEMP Linux driver

This section describes the functionality of the KGEMP driver software, which was developed as a part of this thesis. Since the KGEMP PCI interface card is a custom development, software to support this hardware device had to be created. As it involves direct access to hardware, this cannot be accomplished as a simple program running on the first level PCs, but the underlying operating system has to be extended in functionality to support the KGEMP card. The actual online software then uses these extensions to acquire the data via the KGEMP interface.

The Linux operating system distinguishes between two domains of execution of machine code: kernel space and user space. Only the operating system itself runs in kernel space and therefore has full access to hardware and memory. Regular programs run in user space and can only access the hardware through functions provided by the operating system. The use of these system calls can be subject to safety restrictions, for example such that programs started by one user cannot access the files of other users on the hard disk. Another important aspect of this architecture is the memory management. A user space program can only access memory that it has reserved. Whenever it requests some memory space, the operating system (running in kernel space) assigns a part of the PC's memory to that process. While this concept of adding the kernel space layer between the computer hardware and the software programs eventually running on it, greatly improves the simplicity of software development and security when many programs are running simultaneously, it also necessitates certain tasks to be performed by the KGEMP driver software listed below.

- Allocation of ring buffer memory: The biggest part of the PC's memory is to be used as ring buffer memory. The driver is responsible for allocating this memory. In contrast to memory allocated by user space programs, the ring buffer must be a continuous piece of memory and must not be swapped to hard disk in case of low memory.
- Driving the KGEMP card: This includes initialization at the begin of operation and the management of the data transfers from the KGEMP card into the PC's memory.
- Provide an interface to user space programs: The online software running on the first level PCs needs to gain access to the ring buffer memory. Furthermore, some control commands have to be available, such as a reset of the KGEMP card.

The major focus of the development is low CPU usage. As the online software on the first level PC has to perform CPU intensive tasks like the integration of approximately 20,000 pulse shapes every second, it is vital that the CPU time spent on the data reception itself is kept as low as possible.

The KGEMP driver is written in C and implemented in one single file, `kgemp.c`. The header file `kgemp.h` contains definitions of data structures and constant numbers, and is included by both the driver source code and the user space program sources that use this driver. The driver was developed for use with Linux 2.6, which is at the time of writing, the current stable version of the Linux kernel.

7.2.1 Memory allocation

The KGEMP driver needs to get hold of the ring buffer memory. The ring buffer has to be a continuous section of the physical memory. During the normal operation of a Linux PC, the memory gets more and more fragmented, as programs allocate and free memory time again. To circumvent the problem of finding continuous memory, a more direct approach to occupying the memory needed was chosen. When starting the PC, a command line option can be passed to the Linux operating system kernel, which specifies the size of the memory of the PC. Instead of giving the full available memory size here, a much smaller memory size is pretended. By specifying 128 MB instead of the actually available 1024 MB, an amount of 896 MB remains untouched by the operating system. Only the first 128 MB of the memory are managed by the Linux kernel, and all memory allocations will be served from this part of the memory. The KGEMP driver is designed to make use of the additional memory, which is present but not known to the rest of the operating system. It automatically probes how much additional memory is available, by writing and reading back data patterns to the memory area following the officially available memory. This area is called "high memory" in the Linux terminology. The obtained memory is per definition continuous. While this method works reliably, it is a violation of the operating system's philosophy. The presence of more than one driver occupying memory in this way would render all these drivers unusable. There is no management of the high memory on operating system side. The KGEMP driver only works in this respect, because it is the only driver that uses the high memory. The fact that the KGEMP card is only used in dedicated data acquisition machines, legitimates this approach, as it drastically simplifies the development of the KGEMP driver.

7.2.2 Hardware access

The code base of the Linux operating system kernel offers a framework to access PCI hardware devices. This includes helper functions to read from and write to PCI configuration registers, and a list of installed PCI devices maintained by the operating system. At startup, the KGEMP driver checks for PCI devices that match the KGEMP identification numbers. The driver can support more than one KGEMP card. The maximum number is defined as `MAX_KGEMP_DEVICES` in `kgemp.h` and currently set to four. If more than one KGEMP device is installed, the available high memory is partitioned equally between the devices. While the design of the KGEMP driver was intentionally chosen not to be bound to supporting only one KGEMP card, this feature will not be used in the actual data acquisition environment. The limiting factor is the bandwidth of the PCI bus and the computing capacity of the PC. Distributing the input data of one first level PC over a number of PCI interfaces would not improve the rate, since all interfaces were attached to the same PCI bus and the data is handled by the same CPU.

For each KGEMP card found (normally only one), its control registers are made available to the driver by claiming the corresponding address ranges. An internal data structure is created for every device, containing the obtained addresses to the card's control registers and the ring buffer's write position counter. The data transfer into the ring buffer memory is performed by the KGEMP card independently. To start the transmission, the driver only has to write the amount of data to be transferred and the physical destination address into the appropriate control registers. After completion, the KGEMP card issues an interrupt which causes the KGEMP driver's interrupt handling routine to be called, which then sets up the next transfer by updating the destination address and again setting the amount of data to be transferred. The latter is always set to the value of `KGEMP_DEFCHUNKSIZE` as defined in `kgemp.h`. The smaller this value is, the more often the interrupt handler has to be called to re-initiate the data transfer. As this is time consuming, too low values can negatively impact on the data transfer rate. The larger the value, the less accurate the control over the transfer is, as the ring buffer position pointer is also updated by the interrupt handler. `KGEMP_DEFCHUNKSIZE` is currently set to 64 kB, which turned out to be a practical value during tests. With the expected data rate of approximately 20 MB/s, this means that the data transfer is performed in periods of 3.2 ms on average.

7.2.3 The interface to user space programs

The functionality discussed so far ensures that the data transmitted by the KGEMS storage board to the KGEMP interface card will be continuously written into the ring buffer area of the first level PC's memory. To make these data available for user space programs, the KGEMP driver provides an interface based on a pseudo file. This means that it creates a file-like object that programs can access like a file on a hard disk. It has a fixed size equal to the size of the ring buffer memory. Programs can read from and write to this pseudo file and thereby obtain data from or write data to the ring buffer. These accesses are not synchronized with the transfer of newly received data. Read accesses at certain positions of the pseudo file will return the data in the ring buffer at that position, regardless of whether and when data were written to this position. For synchronizing the read-out with the filling of the ring buffer, and for the general control over the KGEMP device, additional system calls are available through the KGEMP driver. Userspace programs call them via the `ioctl` (input/output control) function, which is a standard Linux interface for issueing special commands for pseudo files. The commands provided are:

- `KGEMP_IOGETPOS`: Returns the current position up to which data was written to the ring buffer, and the number of times the ring buffer was already completely filled,
- `KGEMP_IOSETPOS`: Sets the position pointer to a value supplied by the calling program,
- `KGEMP_IORESET`: Resets the KGEMP device. This includes stopping the current data transfer and resetting the write position pointer back to zero,
- `KGEMP_IOSTART`: Starts the data transfer into the ring buffer. This initiates the first data transmission period, which will then be renewed continuously by the interrupt handler after completion,
- `KGEMP_IOSTOP`: Instructs the interrupt handler function not to set up the next data transmission when the currently ongoing transmission is completed,
- `KGEMP_IOABORT`: Instantly aborts the ongoing data transmission. No more data transmissions are performed before the next call to `KGEMP_IOSTART`.

These functionalities allow for the implementation of the first level PC online software: After resetting the KGEMP card and starting the data transmission, the online software can regularly check the ring buffer write counter and—in case it has advanced—read data from the ring buffer pseudo file. However, this procedure suffers from two drawbacks, that were eliminated by implementing additional features for the KGEMP driver.

First, when waiting for new data, the online software constantly has to poll the ring buffer write position. As long as it has not advanced, no new data are available. Therefore the `KGEMP_IOGETPOS` function keeps getting called. In order to circumvent this waste of CPU time, an additional control command was implemented. The `KGEMP_IOWAIT` function expects a write pointer position as argument and puts the calling process into sleep state until the given position is reached. Thus, the online software process can wait for new data to be available without consuming CPU time at all. Simultaneously, background processes can run, e.g. for calculating energy integrals of the pulse shapes.

Second, due to the design of the driver interface as pseudo file, the online software uses the same system call to obtain data from the ring buffer as for reading data from a file. This system call copies the requested data into a memory area specified and previously reserved by the calling program. While this is the natural approach for real files, as the data is read from hard disk to the specified buffer memory, in the case of the KGEMP driver it leads to a superfluous copying cycle, as the data is copied from memory to memory. It is desirable to access the data directly at its original location in the ring buffer. Therefore the memory mapping technique was implemented to the KGEMP driver and another system call for the KGEMP pseudo file is provided, which makes the whole ring buffer memory accessible for the user space program. Calling the `mmap` function returns a pointer, which the online software can use to directly access the ring buffer. This direct access does not involve calls to read or write functions, but works exactly like accesses to ordinary memory.

7.2.4 Summary

The KGEMP driver, developed as a part of this thesis, manages the direct transfer of the data received from the KGEMS storage board by the KGEMP interface card to the memory of the PC. As the transfer is performed by the KGEMP card itself, CPU time is only consumed for the regular setup of transmissions. Additionally, the KGEMP driver facilitates direct access to the received data by the online software on the first level PCs, which is still to be developed. Waiting for new data takes place with no CPU consumption at all, since it does not involve constant polling. Rather the waiting process sleeps until it is reactivated by the driver at the time new data are received. All these features minimize the CPU time needed for running the data transfer into the PC. Therefore a maximum of computing capacity is available for the tasks of the online software. In tests, a data transfer rate of more than 40 MB/s was established and working for more than two days, causing no measurable CPU usage [Sch04].

Chapter 8

Summary and outlook

In this thesis, the data acquisition system of the Grande array, which is part of the KASCADE-Grande experiment, has been presented. The hardware used has been described, and the online software running the Grande array has been discussed in detail.

The `grandedaq` program was developed as the first part of this thesis. It drives and reads out the hardware components in the central Grande DAQ station. From the obtained data, it builds data streams which can contain—apart from the air shower event data—calibration data (the single event spectra, retrieved from dedicated taking of uncorrelated single station events) and monitoring data (the single event rates as obtained from scaler measurements). Programs running on remote machines can connect to the `grandedaq` program via the network and receive selected types of data. The central KASCADE-Grande event building software retrieves data of air shower events fulfilling certain trigger conditions. The joint data taking of the Grande array as part of the KASCADE-Grande experiment is realized this way. The complete Grande data stream is also written to hard disk. In order to prevent that network communication problems impact on the data taking, the `grandedaq` program forks into two independently running processes, one of which takes care of running the read-out routines, the other one being responsible for data distribution. The regular data taking with the Grande array has started in summer 2003. Since then, the `grandedaq` program has proven to be stable. No evidence has been found that the online software produces incorrect data. The total dead time amounts to less than one per cent.

In the second part of this thesis, a new flash ADC (FADC) based data acquistion system, which is currently under development, has been introduced: It consists of custom-made digitizer, storage and PCI interface cards, and a farm of six PCs. It is dead time free and provides full pulse shape information on the photomultiplier signals in an air shower event. All detector stations work in a self-triggering mode—there is no global trigger signal. The full data of all stations is available in the memories of the PCs. As the search for real air shower events is performed by online software, the event selection conditions can easily be changed. Energy spectra are created from the full set of single station events before event selection.

All hardware components developed in the framework of the FADC system are equipped with FIFO components in order to buffer and derandomize data streams. The impact of the finite sizes of these buffers has been examined. It has been shown, that for the event and data rates expected in regular operation, all buffers are large enough to prevent any data loss. Malfunctions of detector stations leading to extremely high event rates may lead to data loss. By disabling the reception of data from the malfunctioning detector stations, it can be prevented that “hot” stations affect the data taking of other stations.

The concepts of the online software for the FADC system, which still has to be developed, have been introduced. An event selection algorithm has been presented and an upper limit for the rate of random coincidences fulfilling the selection conditions was calculated. By demanding hits from at least seven detector stations within three microseconds, a background event rate of less than one event in 20 seconds is achieved. The hardware related part of the online software for the new FADC system has been developed as part of this thesis. It implements the data transfer from the storage boards into the memory. Special care has been taken, that the transfer itself and the access to the data by the online software consume as little CPU time as possible.

At the time of writing of this thesis, the design of the hardware for the FADC system has been finished and the hardware is being tested or in production. Installation at the site of the Forschungszentrum Karlsruhe will begin in summer 2004. Online software for the operation of the FADC system will be developed. The concepts presented in this thesis may be extended by additional features, e.g. by the correction of the pulse shape data for low frequency noise, or the automatic adjustment of intercalibration constants. Moreover, reconstruction algorithms have to be developed which make use of the additional information provided by the FADC system.

Appendix A

The `experiment.h` configuration file

The file `experiment.h` is the central configuration file for the online software of the current data acquisition system.

```
#ifndef _experiment_h_
#define _experiment_h_

#ifdef USE_THREADS
#include <debt.h>
#endif

#define NHUTS 37                // number of huts
#define NCIRCLES 18             // number of circles

/* RTCM Real Time Clock Module */

#define RTCMBASE 0xfe0000
#define RTCMOUTVETO 0
#define RTCMOUTSINGLEMUON 1
#define RTCMOUTEXTTRIGGER 2
#define RTCMOUTSENDTRIGGER 3

/* CAMAC CRATES
 * CBDBASE - base address of CBD8210 CAMAC branch driver
 * CAMACCRATES - values (branch*100+crate) of CAMAC crates
 *   needed for initialization of crate controllers
 */

#define CBDBASE ((unsigned long) 0x00800000)
#define CAMACCRATES { 1,2,3 }

/* now define circles
 * First number of each circle is the central hut.
 * The other numbers are the surrounding huts, starting
 * with the up right corner, going counter-clockwise.
 * This is the order in which they are plugged into the
 * PLUs (magic boxes).
 * A hut number of zero means "no hut present".
 */
```

```
#define CIRCLES { \
{  7, 1, 2, 8,13,12, 6 }, \
{  8, 2, 3, 9,14,13, 7 }, \
{  9, 3, 4,10,15,14, 8 }, \
{ 10, 4, 5,11,16,15, 9 }, \
{ 13, 7, 8,14,19,18,12 }, \
{ 14, 8, 9,15,20,19,13 }, \
{ 15, 9,10,16,21,20,14 }, \
{ 18,12,13,19,24,23,17 }, \
{ 19,13,14,20,25,24,18 }, \
{ 20,14,15,21,26,25,19 }, \
{ 21,15,16,22,27,26,20 }, \
{ 24,18,19,25,30,29,23 }, \
{ 25,19,20,26,31,30,24 }, \
{ 26,20,21,27,32,31,25 }, \
{ 29,23,24,30,34, 0,28 }, \
{ 30,24,25,31,35,34,29 }, \
{ 31,25,26,32,36,35,30 }, \
{ 32,26,27,33,37,36,31 } }

/* ADCs (CAEN V785):
 * NADC - number of ADCs
 * ADCBASE - base addresses of ADCs
 * ADCCHANNELS - for every hut, an array of the three ADC channel numbers
 *    (ADC 1 has channels 1-32, ADC 2 has channels 33-64 ...)
 */

#define NADC 4
#define ADCBASE { 0x11110000, 0x22220000, 0x33330000, 0x44440000 }
                                                 // ADC base addresses
#define ADCCHANNELS { \
{  1,33,65 }, \
{  2,34,66 }, \
{  3,35,67 }, \
{  4,36,68 }, \
{  5,37,69 }, \
{  6,38,70 }, \
{  7,39,71 }, \
{  8,40,72 }, \
{  9,41,73 }, \
{ 10,42,74 }, \
{ 11,43,75 }, \
{ 12,44,76 }, \
{ 13,45,77 }, \
{ 14,46,78 }, \
{ 15,47,79 }, \
{ 16,48,80 }, \
{ 17,49,81 }, \
{ 18,50,82 }, \
{ 19,51,83 }, \
{ 20,52,84 }, \
{ 21,53,85 }, \
{ 22,54,86 }, \
{ 23,55,87 }, \
{ 24,56,88 }, \
{ 25,57,89 }, \
{ 26,58,90 }, \
{ 27,59,91 }, \
{ 28,60,92 }, \
{ 29,61,93 }, \
{ 30,62,94 }, \
{ 31,63,95 }, \
{ 32,64,96 }, \
{  97,102,107 }, \
{ 112,103,108 }, \
{  99,104,109 }, \
{ 100,105,110 }, \
{ 101,106,111 }}

/* TDCs (CAEN V767):
 * NTDC - number of TDCs
 * TDCBASE - base addresses of TDCs
 * TDCCHANNELS - for every hut, the TDC channel number (first channel is 1)
 */

#define NTDC 1                  // number of CAEN-V767 TDCs
#define TDCBASE { 0x0bc50000 }
#define TDCCHANNELS { \
 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16, \
17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37 };

/* Pattern Units (CAEN V259N):
 * NPU - number of pattern units
 * PUBASE - base addresses of the PUs
 */

#define NPU  2                  // number of pattern-units
#define PUBASE  { 0xaaaa00, 0xbbbb00 }

#define EXTTRIGGERPU 19    // pattern unit channel for external trigger:
                    // channels 1-16 are on PU#1, channels 17-32 on PU#2
#define SINGLEMUONPU 32    // pattern unit channel indicating
                           // single muon measurement
#define SEVENOUTOF7PU 20   // pattern unit channel indicating
                           // 7/7 coincidence

/* Scalers (CAEN C257)
 * NSCALER - number of scalers
 * SCALERBCN - CAMAC BCN addresses of scaler
 * SCALERCHANNELS - for every hut, the scaler channel number
 * SCALER5MHZ_LOW - 24 least significant bits of 5MHz counter double channel
 * SCALER5MHZ_HIGH - dto. MSB
 * SCALERDEADTIME_LOW, SCALERDEADTIME_HIGH: dead time counter (5MHz && VETO)
 */

#define NSCALER 3               // number of scalers
#define SCALERB 0
#define SCALERC 3
#define SCALERN (int[]){ 3,5,7 }
#define SCALERCHANNELS { \
 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16, \
17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37 };
```

```
#define SCALER5MHZ_LOW 47
#define SCALER5MHZ_HIGH 48
#define SCALERDEADTIME_LOW 45
#define SCALERDEADTIME_HIGH 46

#define SCALEREVENTSECONDS 1

/* Station health indicators.
   Station is ill if its scaler values is below STATIONILL_LOW
   or above STATIONILL_HIGH counts per second. */
// #define STATIONILL_LOW 1000
// #define STATIONILL_HIGH 6000

/* PLUs a.k.a. "magic boxes" (CAEN C85)
 * NPLU - number of PLUs (should be equal to number of circles)
 * PLUBCN - CAMAC BCN addresses of PLUs
 */

#define NPLU 18 // number of PLUs (magic boxes). Should be equal to NCIRCLES
#define PLUBCN { \
CBD8210_BCN(0,1,3), \
CBD8210_BCN(0,1,5), \
CBD8210_BCN(0,1,7), \
CBD8210_BCN(0,1,9), \
CBD8210_BCN(0,1,11), \
CBD8210_BCN(0,1,13), \
CBD8210_BCN(0,1,15), \
CBD8210_BCN(0,1,17), \
CBD8210_BCN(0,1,19), \
CBD8210_BCN(0,2,3), \
CBD8210_BCN(0,2,5), \
CBD8210_BCN(0,2,7), \
CBD8210_BCN(0,2,9), \
CBD8210_BCN(0,2,11), \
CBD8210_BCN(0,2,13), \
CBD8210_BCN(0,2,15), \
CBD8210_BCN(0,2,17), \
CBD8210_BCN(0,2,19) }

/* DISCRIMINATORs (LECROY 4413)
 * NDISCRIMINATOR - number of discriminators
 * DISCRIMINATORBCN - CAMAC BCN addresses of the discriminators
 */

#define NDISCRIMINATOR 3
#define DISCRIMINATORBCN { \
CBD8210_BCN(0,1,21), \
CBD8210_BCN(0,2,21), \
CBD8210_BCN(0,2,22) }

/* STATUS A (CAEN C236)
 * STATUSABCN - CAMAC BCN address
 * STATUSABASE - convenience macro: CBDBASE|STATUSABCN
 */
```

```
//#define STATUSABCN (CBD8210_BCN(0,2,20))
//#define STATUSABASE (CBDBASE|STATUSABCN)

/* GPS Camac Module
 * GPSB, CPSC, GPSN: Camac BSN of GPS module
 */

#define GPSB 0
#define GPSC 3
#define GPSN 20

/*
 */

// MAXLOOP: if ADC/TDC has no data, try this many times
#define MAXLOOP 100

// SPECTRAEVENTS: take spectrum during this many shower events
#define SPECTRAEVENTS 5000
// SINGLEMOUNSPEREVENT:
// take this many single muon events after one shower event
#define SINGLEMUONSPEREVENT 3

int init_experiment(int A24, int A32, int enableexttrigger,
                    int sendtrigger, const char *mboxfile);
void shutdown_experiment(int A24);

void readout_experiment(int A24, int A32);

struct event_scaler *readout_scaler(int A24, int A32, int rtcm_seconds,
                                    int rtcm_microtime);
#if ( SINGLEMUONPU > 0 )
void readout_singlemuon(int A24, int A32);
struct event_spectrum *build_spectrumevent(int rtcm_seconds,
                                           int rtcm_microtime);
#endif

#endif
```

Appendix B

Structure of the data sent by the KGEMD board

The data packets sent by the KGEMD digitizer board have the following structure:

- Two “magic words” with fixed values to mark the beginning of a data packages,
- one word containing the station number,
- 252 words containing the data of the high gain channel,
- 252 words with the low gain data,
- one word giving the value of the 1 Hz counter,
- two words containing the 5 MHz counter value,
- two words containing the 62.5 MHz counter value.

In this context, a “word” is 16 bit wide. The data packet consists of 512 words, thus has a total size of $1,024$ bytes.

Appendix C

Derivation of random coincidence rates

In this appendix, the rate of single station events that are followed by $N-1$ more events within a given time window, is calculated. It serves as an upper limit for the rate of random N-fold coincidences.

N	Order of coincidence
T	Width of time window (3 μs)
r	Total single event rate (92,500 s^{-1})
$p_i(t)$	Probability density for the i-th next event in $[t; t + \mathrm{d}t]$

The single events are randomly distributed in time. The probability density distribution for the time between two events therefore has an exponential shape. As the function has to be normalized and the expected value has to be $1/r$, it is determined to be

$$p_1(t)\ \mathrm{d}t = r\exp(-rt)\ \mathrm{d}t.$$

The function $p_i(t)\ \mathrm{d}t$ gives the probability that the i-th next event takes place within the time interval $[t; t + \mathrm{d}t]$. In this case the $i-1$-st event has to take place at a time t', with $0 < t' < t$. The i-th event itself then occurs at the time $t - t'$ after the $i-1$-st. This leads to

$$p_i(t)\ \mathrm{d}t = \int_{t'=0}^{t} p_{i-1}(t')\ \mathrm{d}t' p_1(t-t')\ \mathrm{d}t.$$

The solution to this iterative formula is the function

$$p_i(t)\ \mathrm{d}t = \frac{r^i t^{i-1}}{(i-1)!}\exp(-rt)\ \mathrm{d}t$$

as proven by complete induction:

- Check for $i = 1$:

$$p_1(t)\ \mathrm{d}t = \frac{r^1 t^{1-1}}{(1-1)!}\exp(-rt)\ \mathrm{d}t = r\exp(-rt)\ \mathrm{d}t.$$

- Correctness for $i-1 \rightsquigarrow$ correctness for i:

$$\begin{aligned}
p_i(t)\,\mathrm{d}t &= \int_{t'=0}^{t} p_{i-1}(t')\,\mathrm{d}t' p_1(t-t')\,\mathrm{d}t \\
&= \int_{t'=0}^{t} \frac{r^{i-1}t'^{i-2}}{(i-2)!} \exp(-rt')\,\mathrm{d}t' r \exp(-r(t-t'))\,\mathrm{d}t \\
&= \frac{r^i}{(i-2)!} \exp(-rt) \int_{t'=0}^{t} t'^{i-2}\,\mathrm{d}t'\,\mathrm{d}t \\
&= \frac{r^i}{(i-2)!} \exp(-rt) \frac{1}{i-1} t^{i-1}\,\mathrm{d}t \\
&= \frac{r^i t^{i-1}}{(i-1)!} \exp(-rt)\,\mathrm{d}t.
\end{aligned}$$

The rate of events followed by $N-1$ more events within the time window, is simply the total event rate multiplied by the probability that an event is followed by $N-1$ additional events within that time window. The latter is obtained by integrating p_{N-1} from 0 up to T:

$$R_N(T;r) = r\int_0^T p_{N-1}(t)\,\mathrm{d}t = \frac{r^N}{(N-2)!}\int_0^T t^{N-2}\exp(-rt)\,\mathrm{d}t.$$

List of Figures

List of Tables

List of Acronyms

Acronym	**Explanation**
ADC	Analogue-to-digital converter
CAMAC	Computer Automated Measurement And Control
CPU	Central processing unit *(of a computer)*
DAQ	Data acquisition
FADC	Flash analogue-to-digital converter
FIFO	First-in first-out
FPGA	Field programmable gate array
KASCADE	Karlsruhe Shower Core And Array Detector
KGEMD	KASCADE-Grande Electromagnetic Detector Digitizer Board
KGEMP	KASCADE-Grande Electromagnetic Detector PCI Interface Card
KGEMS	KASCADE-Grande Electromagnetic Detector Storage Board
KGEMT	KASCADE-Grande Electromagnetic Detector Trigger Receiver Card
LVDS	Low voltage differential signal
m.i.p.	Minimum ionizing particle
n.d.f.	Number of degrees of freedom
PU	Pattern unit
RAM	Random access memory
RMS	Root mean square
RTCM	Real time clock module
TDC	Time-to-digital converter
VME	Versa Module Eurocard

Bibliography

[Agl93] M. Aglietta et al., *UHE cosmic ray event reconstruction by the electromagnetic detector of EAS-TOP*, Nucl. Instrum. Meth. **A336** (1993), 310–321.

[And04] V. Andrei, diploma thesis, University of Siegen, 2004, in preparation.

[Ant03] T. Antoni et al., *The Cosmic ray experiment KASCADE*, Nucl. Instrum. Meth. **A513** (2003), 490–510.

[Ber01] M. Bertaina et al., *KASCADE-Grande: A conclusive experiment on the knee*, Proc. 27th International Cosmic Ray Conference (ICRC 2001) (K.-H. Kampert, G. Heinzelmann, and C. Spiering, eds.), vol. 2, 2001, pp. 792–795.

[Bie01] P. Biermann and G. Sigl, *Introduction to cosmic rays*, Lect. Notes Phys. **576** (2001), 1–26, astro-ph/0202425.

[CAE03] CAEN, Viareggio, Italy, *Technical Information Manual, Mod. N442*, Revision n. 0 ed., 2003, preliminary.

[Cas03] A. Castellina, *Cosmic rays and high energy physics: The EAS-TOP data*, Nucl. Phys. Proc. Suppl. **122** (2003), 243–246.

[Gai90] see e.g. references in: T. K. Gaisser, *Cosmic Rays and Particle Physics*, Cambridge Univ. Press, 1990.

[Gla03] R. Glasstetter, 2003, personal communication.

[Gru00] see e.g. references in: C. Grupen, *Astroteilchenphysik*, Vieweg, Braunschweig/Wiesbaden, 2000.

[Hau03] A. Haungs et al., *The KASCADE-Grande experiment*, Proc. 28th International Cosmic Ray Conference (ICRC 2003) (T. Kajita, Y. Asaoka, A. Kawachi, Y. Matsubara, and M. Sasaki, eds.), vol. 2, 2003, pp. 985–988.

[Hes12] V. F. Hess, *Über Beobachtungen der durchdringenden Strahlung bei sieben Freiballonfahrten*, Physik. Zeitschr. **13** (1912), 1084–1091.

[Kam03] K.-H. Kampert et al., *Status of the KASCADE-Grande experiment*, Nucl. Phys. Proc. Suppl. **122** (2003), 422–426, astro-ph/0212347.

[Lon92] M. S. Longair, *High energy astrophysics, Volume 1, 2nd edition*, Cambridge Univ. Press, 1992.

[Nag00] M. Nagano and A. A. Watson, *Observations and implications of the ultrahigh-energy cosmic rays*, Rev. Mod. Phys. **72** (2000), 689–732.

[Nav04] G. Navarra et al., *KASCADE-Grande: A large acceptance, high-resolution cosmic-ray detector up to* 10^{18} eV, Nucl. Instrum. Meth. **A518** (2004), 207–209.

[Pet97] W. Peterson, *The VMEbus Handbook*, 4th ed., VMEbus International Trade Association, 1997.

[Rao98] see e.g. references in: M. V. S. Rao and B. V. Sreekantan, *Extensive Air Showers*, World Scientific Publishing, 1998.

[Rot03] M. Roth et al., *Energy spectrum and elemental composition in the PeV region*, Proc. 28th International Cosmic Ray Conference (ICRC 2003) (T. Kajita, Y. Asaoka, A. Kawachi, Y. Matsubara, and M. Sasaki, eds.), vol. 1, 2003, pp. 139–142.

[Sch04] B. Schöfer, 2004, personal communication.

[Stü03] M. Stümpert, *Entwicklung eines Zeitkalibrationssystems für KASCADE-Grande*, Diplomarbeit, Interner Bericht KASCADE-Grande 2003-02.

[Ulr04] H. Ulrich, *Untersuchungen zum primären Energiespektrum der kosmischen Strahlung im PeV-Bereich mit dem KASCADE-Experiment*, FZKA-6952.

[Zim04] D. Zimmermann, 2004, personal communication.

Acknowledgement

I want to express my gratitude to all who made this thesis possible. In the first place this is of course Prof. Dr. Peter Buchholz, who introduced me to the field of particle and astroparticle physics. I can say without doubt, that his introductory lectures on particle physics were the best lectures I attended during the course of my studies. I also want to thank him for the effort he put into making my stay as a DESY summer student in 2001 possible. Finally, I want to thank him for the interesting and exciting work in the KASCADE-Grande experiment and in his working group.

I want to thank Prof. Dr. Claus Grupen for revising this thesis. I am glad that I could attend his famous after-dinner speeches on several occasions. I hope there will be more.

I want to thank our entire working group for the good atmosphere. Thanks to all my colleagues for the good team work, for the interesting and funny discussions we had during lunch, and for the parties at the university. Special thanks to Dr. Wolfgang Walkowiak for guiding me through some minor and major problems related to my thesis, and through the bicycle paths of the Siegerland. I thank Ulrich Husemann for giving so valuable comments on this thesis in its early stages. He also was always a quick solution to all my ROOT problems. Thanks to Dr. Wolfgang Walkowiak and Marc Brüggemann for proof-reading. Thanks to Marc Brüggemann, Dirk Zimmermann, Victor Andrei, Roxana Lixandru and Yury Kolotaev for the good time we had working on the same project, in Siegen and in Karlsruhe.

I want to thank all the helpful people in Karlsruhe: Dr. Jürgen Wochele, who answered all my questions about the Grande array, and also assisted me with many organizational issues; Dr. Harald Schieler and Andreas Weindl, who helped me during the development of the Grande online software. Thanks to Dr. Giancarlo Trinchero, Dr. Andrea Chiavassa, Ralph Glasstetter, Dr. Holger Ulrich, and many others, not only for discussions about physics and the experiment, but also for showing me some nice places in and around Karlsruhe.

Thanks again to Ulrich Husemann and Dirk Zimmermann for our regular visits to the cinema, which pretty fast turned out to be not that regular. We should keep it up.

I say thanks to Dr. Matthias Böcker (also known as “alten Wemser”) for many enjoyable discussions on politics.

I want to say a big “thank you” to Tomaida-Roxana Lixandru for many discussions unrelated to physics, for sharing some of my musical preferences, and for being a good friend in difficult times. Her presence definitely had a positive influence on my English skills.

Thanks to all those who made my studies in Dortmund a good time: Daniel Weber, Andreas Jung, Jens Damerau, Aleksandar Rakic, Daniel Dobos, and many more. Special thanks to Andreas Jung and Jörg Seehafer for building my bed, and of course to Daniel Dobos for two years of cocktail parties, barbecues, internet, and glue on the floor.

And thanks to my flatmates in Siegen (current and former) for standing me: Claudia, Meike, Simone, Antje, René, Anke.

Finally, I thank my mother Marianne Over, for giving me support—financially and in general—during my studies, and during all my life.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate und Ergebnisse Anderer kenntlich gemacht habe.

Sven Over