,

Savonia University of Applied Sciences

Building and developing the ARC system

Thesis

Mikko Nykänen

Automation Technology

Accepted __ . __ . ______ ______________________________

Preface

I want to thank my supervisors, senior teacher Harri Heikura (Savonia University of Applied Sciences) and Rauno Lauhakangas (Helsinki Institute of Physics) for this great opportunity to work in Cern. Also I want to thank Leszek Ropelewski in whose lab I was working during six months period for this opportunity. My stay in Cern gave me a good view how to do some independent work and what is reality in physics research.

This thesis was made in CERN in summer 2009. The opportunity to make this work surfaced because they had all components but no manpower to build the system and this seemed to be a good subject to do a thesis.

In Varkaus 8.3.2010

Mikko Nykänen

SAVONIA UNIVERSITY OF APPLIED SCIENCES, BUSINESS AND ENGINEERING, VARKAUS

Degree Programme

Automation Technology

Author

Mikko Nykänen

Title of Project

Building and developing the ARC system

Type of Project	Date	Pages
Thesis	8.3.2010	52+12
Academic Supervisor	Company Supervisor	Company
Harri Heikura	Rauno Lauhakangas	Cern

Abstract

ARC system is an electronics test system for testing silicon detectors in LHC experiment. The first objective in thesis was to construct and develop the ARC system. The ARC system collects measurements data from detectors and controls the electronics. The ARC system consisted of two parts: the ARC electronics and ARCS software. The second part of the thesis was to develop a program, which was able to analyse data collected with the ARC system. The goal in the thesis was to make system that collects data from detector and after that it is analysed with the analysing program.

First part of the thesis was to make an ARC setup, which consists of electronic and software. When the ARC setup was ready, attention focused to the analysing program. In ROOT environment a simple analysing code were made. This code goes through the measurement data and prints the results to histogram. The end result was a histogram where the user is capable of reading the collected measurements.

When the ARC setup and analysing the program were finished, few tests were made. Using a strontium source measurements were taken from Triple GEM chamber. The measurements were put into a Pulse Height Spectrum histogram. This was done in ROOT environment by using macros. The histogram is considered to be the interpretation of signals that were collected during DAQ.

The ARC system was working and it was collecting data from detector like it should. After DAQ the data can be analysed in ROOT environment. The analysis concentrated on Landau distribution of the measurements, but the desired curve was not completely achieved during the measurements. The ARC setup works, but the software (ARCS) was not intended solely for the acquisition of data. With a better LabView program for the ARC setup, more efficient DAQ could be done. During the work a system was successfully built. This system collects data from a detector and the results of these measurements were later put to graphs. The user manual was also made for the appliance.

Keywords

ARC, Electronics, Measurements, Tests, Microchips

Confidentiality

No

SAVONIA-AMMATTIKORKEAKOULU, VARKAUDEN YKSIKKÖ

Koulutusohjelma

Automaatiotekniikan koulutusohjelma

Tekijä

Mikko Nykänen

Työn nimi

ARC – järjestelmän rakentaminen ja kehittäminen

Työn laji	Päiväys	Sivumäärä
Opinnäytetyö	8.3.2010	52+12
Työn valvoja	Yrityksen yhdyshenkilö	Yritys
Harri Heikura	Rauno Lauhakangas	Cern

Tiivistelmä

ARC -järjestelmä on testausjärjestelmä, jolla testataan *pii* -ilmaisimia. Työn ensimmäiseksi tavoitteeksi asetettiin ARC -järjestelmän rakentaminen ja kehittäminen. ARC -järjestelmästä piti kehittää tiedonsiirto ja -keruulaitteisto, jolla kerätään mittaustietoja GEM -ilmaisimista. Järjestelmä koostui kahdesta *osasta*: elektroniikasta eli ARC:sista ja ARCS -ohjelmistosta. Toisena osana työssä oli kehittää ohjelma, jolla pystyttäisiin käsittelemään kerättyä tietoa nopeasti.

Ensiksi koottiin ARC -järjestelmä johon kuului elektroniset komponentit ja ohjelmisto. Elektroniikan eli järjestelmän ollessa kunnossa huomio siirtyi analysointiohjelmaan. ROOT -ympäristössä tehtiin yksinkertainen ohjelma, joka käsittelee kerättyjä mittaustietoja ja tulostaa näiden perusteella valmiita kuvioita. Lopputuloksena oli histogrammi, josta pystytään lukemaan järjestelmällä kerättyjen mittaustulosten kirjo.

ARC -järjestelmän ja analysointiohjelman ollessa valmis, otettiin strontium lähteen avulla mittauksia Triple GEM -ilmaisimesta, joka on uuden tyypin GEM -ilmaisintyyppi. Mittaustuloksista tehtiin Pulssin Korkeuskirjo – histogrammi ROOT -ympäristössä käyttäen tehtyjä makroja. Kuvaajaa tulkitsemalla katsottiin, millaisia signaaleja ilmaisimesta luettiin.

ARC -järjestelmä saatiin toimintaan ja kerättyjä mittaustuloksia käsiteltiin tehdyllä ohjelmalla. Aikaiseksi saatiin jonkin kaltainen Landaun -jakauma mittaustuloksissa, mutta täysin haluttua kuvaajaa ei saatu aikaiseksi mittauksien aikana. ARC -elektroniikka toimii, mutta ARCS -ohjelmistoa ei ole tarkoitettu pelkästään mittaustulosten hankintaan. Paremmalla LabView -ohjelmalla ARC -järjestelmästä voisi tehdä laboratorioon tiedonkeruujärjestelmän. Työssä onnistuttiin rakentamaan laitteisto, joka kerää tietoa ilmaisimesta ja näistä mittaustuloksista saatiin myöhemmin luettavia kuvaajia. Käyttäjälle tehtiin myös käyttöohjeet laitteistoon.

Avainsanat

Elektroniikka, Mittaus, Testaus, Mikropiiri

Luottamuksellisuus

Ei

Table of content

Abbreviations

GEM	Gas Electron Multiplier is a chamber that amplifies electrons. Basic idea is that there is plain with holes, electric field around the plain, electric field is much stronger inside the holes than around. When electron goes through this hole it shoots many electrons out the other side. By putting these layers in series, one electron can multiply up to 8000 electrons. This is way can the charge be read that is much larger than elementary charge 1.6×10−19 coulombs.
APV25-S1	An analogue pipeline ASIC intended for read-out of silicon strip detectors.
ASIC	An application-specific integrated circuit. The circuit is made only for specific purpose like DAQ.
DAQ	Data acquisition is measured values from physical world is converted in digital numeric values.
ROOT	ROOT is OOP framework that analyse large amounts of data in efficient way. ROOT has set of methods how data can be analysed and user only has to know the commands how to operate.
OOP	Object-Oriented Programming uses object – data structure in data fields and methods between interactions of objects.
ARC & ARCS	APV Read-Out Controller and APV Read-Out Controller Software
CM	Common Mode
CMS	Compact Myon Solenoid
Cern	European Organization for Nuclear Research
FE	Front-End
LHC	Large Hadron Collider
Hybrid	Used to descripe board that has APV25-S1 microchip
LV	Low Voltage
DLL	Direct Link-library
ASCII	American Standard Code for Information Interchange
PHS	Pulse Height Spectrum
GUI	Graphical User Interface
FIR Filter	Finite Impulse Response filter
FADC	Fast Analoqye Digital Converter
HD	Hard Disk

1 Introduction

New inventions in instrumentation have lead to a new founding's in physics. Classical physics research relied on human observation and measurements done by human. This changed after the invention of electricity and first instruments that could see or record atomic scale events. This opened new chapter in physics. With new equipment it was possible to see and measure things that were impossible with the old ways. And it was common that theory not proven was proved later on when the necessary technology was developed. New founding's in physics or new instrument technology is awarded e.g. by Nobel price in some occasion.

Nowadays electronics make it possible to record large amount of data in a day. This was not possible when human eye was the source of collecting information. Modern instrument technology needs also fast and reliable data acquisition. DAQ collects data from detector and saves it to file for analysis. This chain of devices needs to be reliable and easy to use. Modern silicon trackers and detectors need also modern DAQ, which increases the overall performance. The goal of working in Cern was to install one DAQ system and learn how to use it. This thesis goes through this project.

In this thesis we go through electronics and software that are used in this work. That is the theory part were we get basic knowledge about tools. Next is the assembly part. In this part the ARC electronics and control PC are put together and necessary software are installed. The third part is about analysing program for the data that is collected with ARCS. The fourth section, is about making tests, making adjustments to setup and also making some modifications to the analysing program. In the end the results of this work are discussed and evaluated.

2 Cern

Cern is European Organization for Nuclear Research and it was founded in 1957. Twenty member states give the funds to Cern. The Cern`s mission is to research particles and forces that are occurring in the world. Pursuit of knowledge and understanding the fundamental building blocks in the world needs constant research. Because of this many inventions have been invented in Cern, the best example of these is the World Wide Web. The organization facilities are located in French-Switzerland border in Geneva region in Switzerland. Cern has 2500 staff members to its payroll. They are scientists, secretaries, even to plumbers who run the daily things of Cern. Every year approximately 8000 scientists come Cern to make research or participate meetings. 850 universities all over the world make cooperation with Cern. [1]

2.1 LHC

LHC is particle accelerator that accelerates particles and collides particles. With LHC scientists try to solve what the matter is and how the universe was born. One of the main goals in LHC experiment is to find Higgs boson. Scientists hope that this boson explains how the particles have a mass and how the gravity force is transmitted between particles. [2]

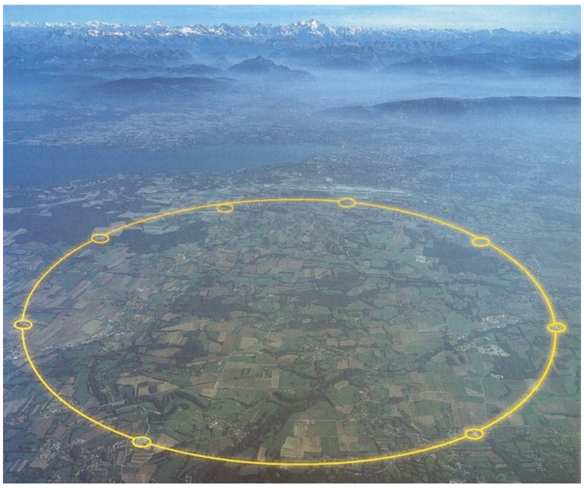


Figure 1[3]. LHC is the biggest man made instrument in the world. LHC is 27 km long accelerator that is under ground at depth of 100 m. LHC is built under French and Switzerland border near Geneva.

3 Goals

The goal of this work is to build and develop an ARC system. The idea is to be able to get data from detectors using ARC electronics. From acquired data the plots are drawn. The main focus is in histograms in these plots. Making a code to do this is one of the key points in analysing the data. After the ARC setup is running and operational, we need to make documentation. This documentation includes how to make this setup and also how to operate this setup. When setup is ready, users can collect data from the detector and process the data to see what is gotten from read-out.

The first thing is to get electronics working. This means the ARC system is built so that the user can collect data with it. This is done by assembling the ARC setup and installing all necessary software in control PC. At the same time we need to get some experience in using this setup, because it will be crucial to teach how to use this ARC system to users. When this all has been achieved the analysing code will be made.

The analysis of the data is done with code made for this purpose. We need a histogram that shows the signals that are coming from detector. So the final form of the data collected from detector is a plot. With this plot the user can read all the collected data and make decisions if we are getting what we are supposed to get. In this plot there should be some fits like *landau* so the user can see if the acquired data is in line with the theory.

Documentation helps the user to understand the ARC system and the same time it tells how to use this ARC setup to collect data.

4 Equipment and software operations

The ARC system consists of two main building blocks: hardware and software. The goal of this chapter is to make the reader familiar with used electronics and software.

4.1 ARC – APV Read-out Controller system

This chapter introduces used ARC electronics, the specifications for control PCs, APV25-S1 microchip and the ARC setup structure. ARC or APV Read-out Controller was made for quality assurance in testing FE hybrids and detector modules. The purpose was to make a flexible system, with low cost, and a compact test system for all main participants in module production of CMS tracker. Because of this all silicon detector modules in CMS can be tested in ARC system.

ARC system was developed in RWTH Aachè University [4] in Germany. ARC system is available for anybody interested. Electronic parts cost money, but ARCS software [5] is free and the source code is open. The user can do modifications to the ARCS program, but those modifications are made with the user's own risk.

4.1.1 ARC electronics

First we go through the hardware and how to build a setup. After we have got acquainted with the electronics, then the software, ARCS, that is running this setup is discussed.

Control PC

Control PC is the interface for the user to use ARC system. The hardware requirements for the control PC computer are the following:

Processor 500MHz
Memory 128mb
One ISA Bus, obligatory
OS (Windows 95, 98, NT, 2000 or XP)

These are the minimum values, but these values can also be higher. But there has to be one ISA Bus in the motherboard for the PCMIO data interface card. [6]

PCMIO

PCMIO is IO card, which, in IBM XT slot, works as interface between ISA bus and the ARC board. PCMIO is an ISA bus-extender module which has address range of 64 words. PCMIO card handles 16 data lines, 6 address lines and 3 control lines from ISA bus, buffers and transfers the data via a 50 pin flat cable. [7]

PCMIO card can be used in memory, or like in this case, in accessing to external IO. But to be able to use this card in Windows the GiveIO – DLL [6] was needed to be installed in PC. The reason for this is that Windows restricts the IO access rights. When ARCS program starts, it seeks from the system root, where the GiveIO.SYS library is installed. In ARC#1 and ARC#2 control PCs these places are *e:/WINNT/system32/drivers/giveio.sys*. If the GiveIO library is not found, the program gives error message and crashes. Installation of this library is crucial to get the IO working.

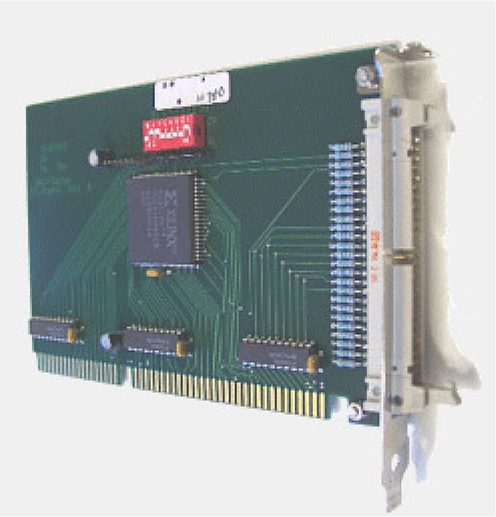
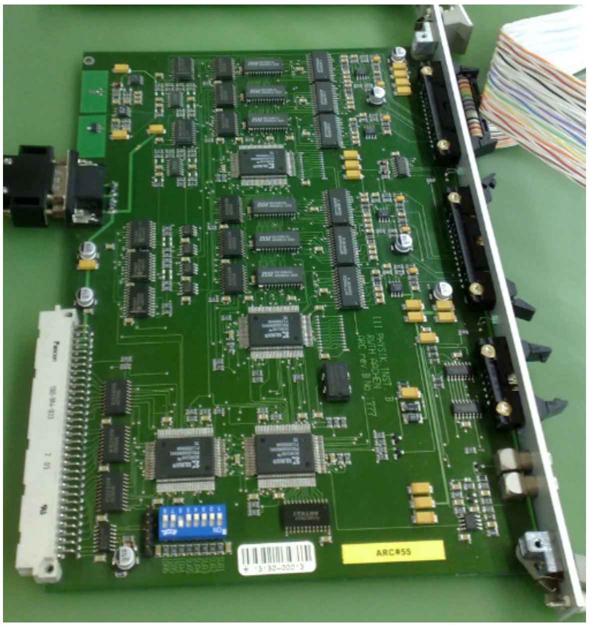


Figure 2. PCMIO is interface between ARC and the computer. The card is connected to ISA bus in the computer. The switch structure (red) gives the operation mode to the board. The 50 pin connector on the right connects the board to ARC board.

ARC board

The ARC board is the central part in ARC setup. In the board there are two hybrid ports, one trigger port and two additional ports. The control PC and ARC board are connected via a 50 pin flat cable. The ARC board is connected to a *FE adapter* via a 26 pin twisted pair cable. The board has switch structure which defines its address and in the same setup there can not be two ARC boards with the same address. The board is powered by an external power source with RS32 connection. [6]



The ARC board was used as a single-board-setup so this setup was easy to mount on top of the table. For test purposes it was a fast and good way to use this board, but for longer usage the setup must be mounted better.

Figure 3. ARC board.

ARC board power source

ARC board needs ±5 V operational voltage. ARC Board, without any hybrids connected to FE adapter, uses 150 mA in negative voltage side and 1.5 A in positive voltage side. One connected hybrid in FE adapter increases the positive side current need around 0.5 A. The setup with two hybrids and two FE adapters connected to ARC board uses 2.5 A current in positive side. This increase does not happen in negative side. This negative/ positive usage voltage needs to be grounded in same place otherwise the board does not work. For transferring this usage voltage we have a cable with one end RS32 connection and the other end with three banana cords. These cords are connected to power sources plus, minus and ground.

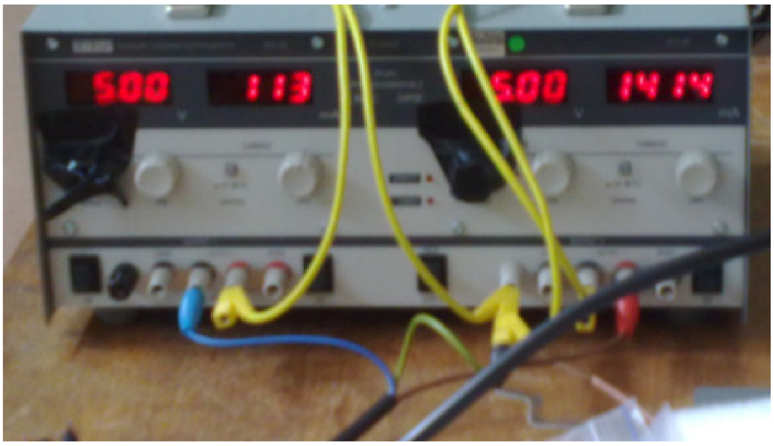


Figure 4. An ARC board power source. The yellow cord connects both sides to same ground. Yellow-green cable in RS32 cable is connected to ground. This connection puts the power source and the ARC board to the same ground. The left side (blue cable) is the negative and the right side (brown cable) is positive side. In the figure the difference in currents can be seen when the ARC board is connected to power source.

FE adapter and totem_apv_adapter_test_card

FE adapter is an interface between the ARC board and FE electronics. FE electronics means all the devices placed near collision point in the LHC experiment. The setup needs also an adapter for APVs, this enables the setup to read APV25-S1. This adapter is called as totem_apv_adapter_test_card. These two adapters are connected directly to each other. The purpose of these two adapters is to make electronics compatible with different electronics. Hybrid, detector end, and totem_apv_adapter_test_card are connected with a twisted pair cable.

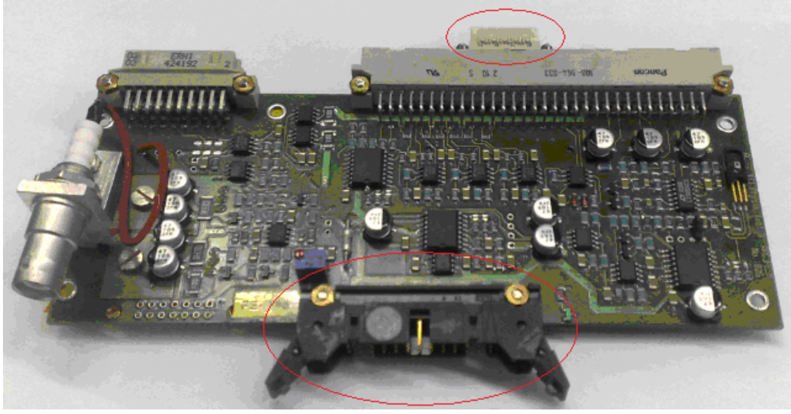


Figure 5. An FE adapter is the second important part in the ARC system. The black connector is connected to the ARC board via a 25 pin twisted pair cable. The totem_apv_adapter_test_card is under this board. Only two connections are used in this board and those are marked with red circles.

Hybrid

Hybrid in this setup means the device that is controlled with this ARC system. In this setting hybrid is a small board that has one APV25-S1 microchip. The chip reads 128 channels of analogue data and processes it in analogue from. This means that the chip makes 25 ns record what the signal level is in the channel. The data is then transferred to an ARC board in analogue format. The hybrid is connected to a detector/ chamber reading strips. The APV chip reads data of the strips when the trigger comes from an ARC board to an APV25-S1 chip.

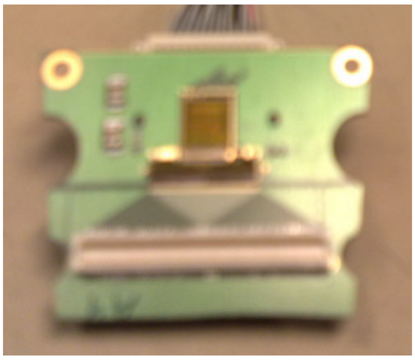


Figure 6. A hybrid that is connected to 128 channels connector.

APV25-S1

APV25-S1 is an analogue pipeline ASIC (application-specific integrated circuit [9]), made for read-out in silicon strips detectors in the CMS tracker. The purpose of this chips architecture is to process data in analogue format before transmitting the data in analogue form to the DAQ. [10]

One chip handles 128 channels of silicon strips. Each channel in chip has a preamplifier and shaper that drives 192 column analogue memories which keep records of the most recent hits in strips that the chip has recorded. The samples are written in memory at LHC 40 MHz frequency. The data accessing mechanism allows that the data can be marked and queued. This data is then requested and processed with the FIR filter. The FIR filter uses switched capacitor network which deconvolves the shaping function of the preamplifier and shaper stages and makes a pulse shape confined to one 25 ns period. This shaped data is kept in another memory, before it can be read out through an analogue multiplexer. [10]

Modes and APV registers

The APV25-S1 operates in different modes. These modes determine how the chip's behaves or what kind of signal the chip should get. To use the most of the chips functions the user needs to know what kind of effects these settings have and how to change them. The APV register is a list of operation voltages or settings read from an APV25-S1 chip. These settings are listed in APV25-S1 User Guide 2.2. [10]

4.1.2 ARC structure

This chapter gives overall view of the setup so we get some insight about distances and how the setup is build. It is good to know how far or near the electronics can be in the setup.

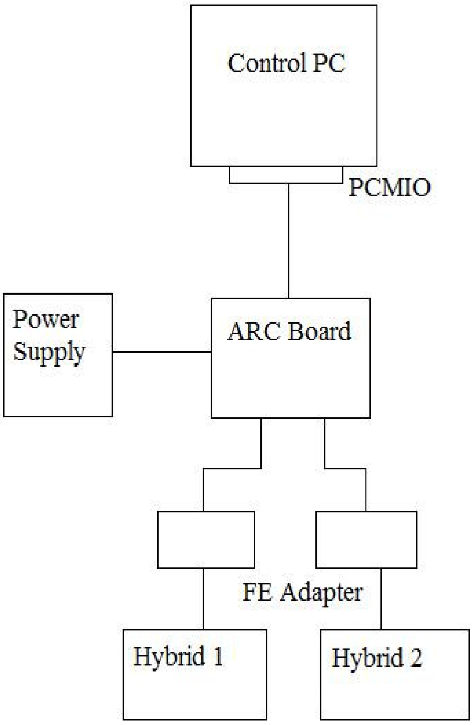


Figure 7. ARC system. Distance between the PCMIO and the ARC board is 1.5 m. The cable is a 50 pin flat cable, but maximum length of this cable was not determined. The power supply and the ARC board have distance of 1 m, but this can be further away because the power source is a possible noise source. The cable between these two units has connector RS32 and one (three banana cord) end. These banana cords must be connected to ±5 V and ground. The ARC board and FE adapter can be about 0.5-1 m apart from each other. The cable is a coloured twisted pair cable. The FE adapter and an APV adapter are together in close contact but the distance to hybrids 1 and 2 is about 30-40 cm.

4.1.3 How ARC works

The user uses the control PC and gives commands in LabView program. These commands are converted in the program to commands to electronics, using GiveIO library. Those commands are transferred via a PCMIO card to the ARC board. The ARC board looks for where the user wants to read the data and accesses to that address. The ARC board sends its request via a FE adapter to selected APV25-S1 address. APV25-S1 makes its DAQ and collects the data. The data is then transferred in analogue format back to the ARC board. The ARC board digitalises data from APVs in 8 bit digital format and the data is given values between 0-255. This digitalized data is then transferred to a control PC via a 50 pin cable. ARCS program reads this data and makes its calculations based on this digitalized data. ARCS program can handle this data flow from 6 APVs from one hybrid.

ARC board functions

The ARC board has three main function block parts: clock and trigger, data sampling and I2C communication. Figure 8 on page 11 is the function block diagram of the ARC board.

The board's clock is a local crystal oscillator which works in 40 MHz or the clock can be given via an extension port to the board, but the clock can not be faster than 40 MHz. An external (NIM) or software asynchronous trigger is connected to the system clock and synchronised with the system clock. NIM is a negative voltage signal that defines logical 1 and 0. When voltage is -1 V it means logical 1 and when its 0 V it means logical 0. [8] When triggered, the board sends trigger patterns to all FE hybrids. This starts the wanted procedures. There are three different types of trigger patterns RESET, CAL REQUEST and EVENT TRIGGER. RESET cleans the registers in APVs and writes the current register values from ARCS to APV registers. CAL REQUEST or calibration request calculates how much the electronics is wrong and makes the correction automatically. EVENT TRIGGER is the mark to the FE board and APVs to start the event. This means the electronics record next 25 ns period on channels. A trigger signal is distributed via separate trigger link or with missing clock pulses. [9]

Data sampling consists of two groups of three eight bit FADCs from two hybrids. The data is sampled from hybrids with 20 or 40 MHz clock. The FADC clock phase can be tuned in ten steps of 1.7 ns up to 17 ns in ARCS. This gives the user more accurate means to find the best latency settings. The basic APV register value in latency can be adjusted by steps of 25 ns. Each channel has 8 Kbyte RAM, where the digitalised data is stored. The sample record length is defined by the stop address. Sampling starts at the address 0 and stops at the defined stop address. This stop address can be set to any value between 0 and 8 Kbyte. Before the data is read, memory address counter sets start address to 0 and after one read cycle the address counter is increased automatically until the set value is achieved. [9]

I2C communication has two functions: *Slow control* and *LV control*. In slow control one I2C controller handles one hybrid operation. The hybrid connected to an FE adapter is powered from the ARC board. *LV controls* the LV regulators in the FE adapter. This gives the right operation voltages to the hybrids. Voltage levels in hybrids are measured via separate I2C link. [9]

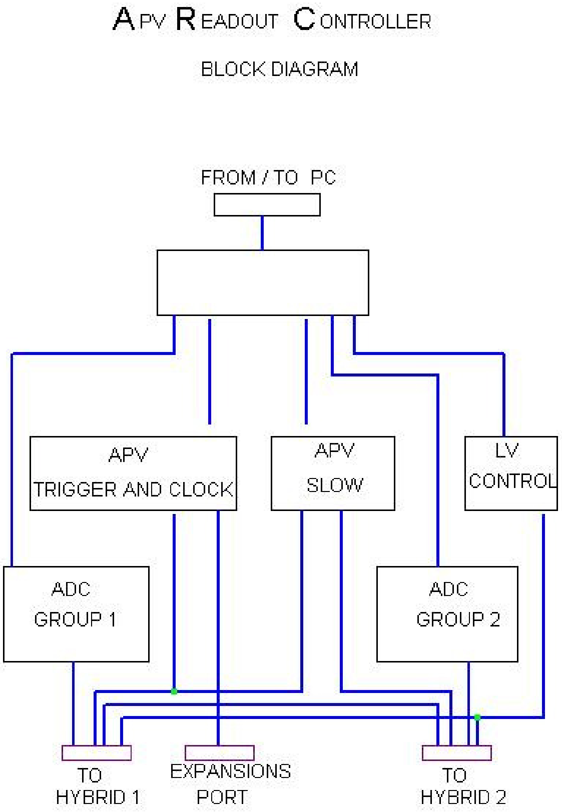


Figure 8. The ARC board function block diagram. [11] This figure shows how the board works. The top most rectangle, where five links goes, is connection to the PC. All information from board is send to the PC when it is requested. The expansion port can give external system clock to the board or give a separate trigger pulse to hybrids. ADC groups deliver digitalized data to the PC. Slow control and LV control monitor and operate the FE adapters and hybrids.

4.2 ARCS – APV Read-Out Controller Software

ARCS is software application made for DAQ and for test purpose of ARC system. ARCS is made by embedding C/C++ codes as dynamic link libraries (DLL) into the LabView program. LabView program works as user-friendly graphical user interface (GUI) which controls operations done by these C/C++ libraries. The purpose of this program is to give the user easy to use test platform in testing FE electronics and APV25-S1 chips functionality. [11]

4.2.1 ARCS program structure

The ARCS program has three phases: initialisation, testing and analysing. When program starts it will run diagnosis to look for devices connected to PC's PCMIO card. If the program finds an ARC board, it searches for devices connected to the ARC board. This phase will list all devices (hybrids now on), give them an address and ask the user to give those hybrids an individual name. Testing phase uses these addresses to communicate with hybrids. Depending on the user commands, the ARCS program makes contact to these addresses and makes preferred tests. The test results can be saved and some tests can be observed when making the test. The test results are saved into root file which has the same name as the hybrid. The analysis can be done during testing or after. This does not affect the DAQ process. The analysis gives the test data as graphical presentations where the user can see the results of all the hybrids with few mouse clicks.

Initialization phase searches hybrids connected to the ARC board and asks the user to name those hybrids. If there is more than one ARC board, HV DEPP board or LEP16 connected, ARCS program checks the addresses. The addresses in the boards have to be different and this is done by selecting different address in the board switch structures. After initialization phase, if there are no errors, the program starts to read data from found hybrids with a random trigger in *monitoring* window.

Monitoring gives full control over hybrids and boards listed on the program. In this window the user sees what voltages are feeding to hybrids, and what voltages are read at the hybrid end. The controls over DAQ and settings are in the *Display* tap. The display shows the list of all boards and hybrids connected to ARCS. The user can choose which board and hybrids are currently in use from this tap. In *Data* tap the user can take raw data from currently read hybrids. Another possibility is to open *XML Parser* and start to analyse recent or previously taken data. *Errors* tap records errors occurring during the DAQ because of a bad header or wrong operation voltage. *APV & Boards* tap controls the devices connected to the ARC board. The user can power on/off the hybrids or adjust the ADC (analogy-digital conversion) clock to finetune DAQ. By selecting *Trigger* button the user can make adjustments to trigger, e.g. the users can change the internal software trigger to external trigger. DAQ starts when the ARC board gets a trigger signal to its trigger port. *Reassign* resets current settings in the hybrid and starts it up. This has to be done every time as the power is down on hybrid. *I2C scan* searches APV25-S1 from hybrid currently in use. *APV register* opens one of the hybrid APV25-S1 chip register and displays it. This is the most important setting that can be altered in the ARCS program. More about this chip can be found at *User Guide 2.2* [13]. *MUX & PLL* controls the impedance level of a coming signal. Selecting more resistors parallel makes low signals more visible to input. *Additional Controller* tap is usable if there are DEPP or LEP16 boards connected to ARC; otherwise there is no use for this. *Monitoring* is meant for the user who knows what he is doing.

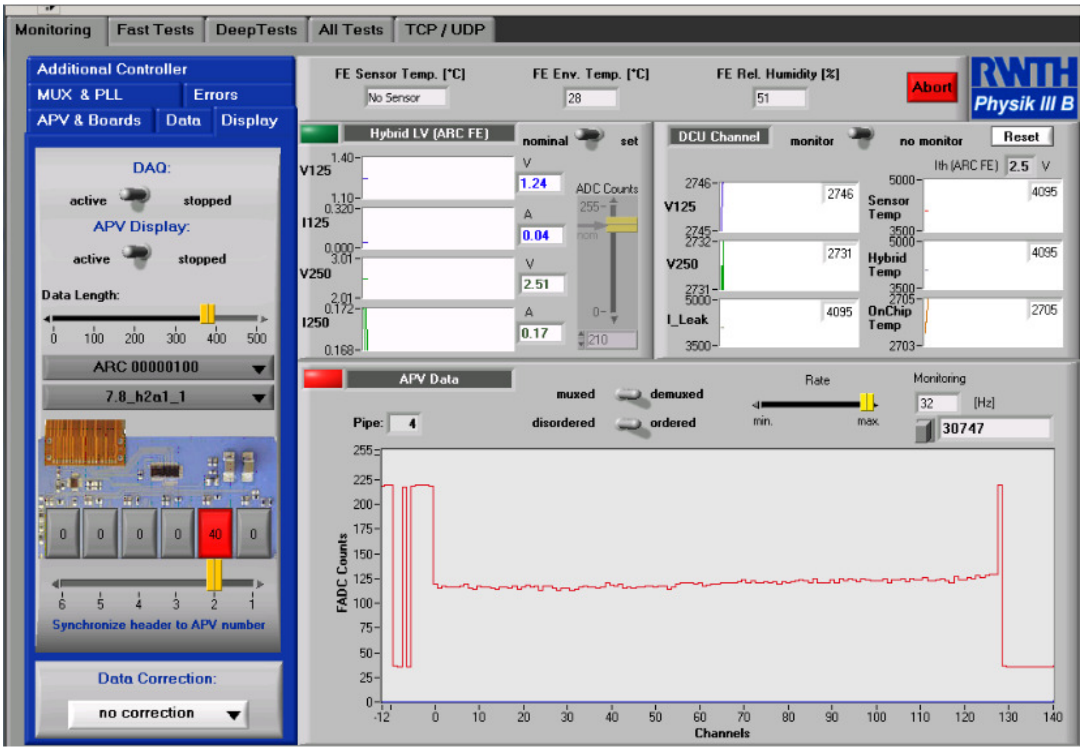


Figure 9. *Monitoring* is meant for the experienced user. The user can make changes in electronics and get data directly from electronics. Most information the user gets from the *APV Data* panel. This panel shows values read from APV channels at monitoring speed which in this figure is 32 Hz. The user sees if there are signal in channels.

Fast test is meant to check the electronics quality. There are nine tests like *Self test on the ARC system* or *I2C address scan & communication.* By selecting tests and running them, the program makes grades how good electronics are. The electronics are graded A, B or failed. The results of the test depend on settings selected during the initialization phase. The mismatch with electronics and settings give false grading. *Fast test* was meant to be a standard test, which check FE electronics and APV25-S1 chips quality.

Deep tests is tool for experts. The user defines the parameters and runs tests. Unlike *Fast test,* the user gets the measurements data in graphs and he makes decision if everything is ok. There are eight tests in deep test: *Pedestal & Noise & CM, Pipeline, Pulse shape, LED, Pinhole, Gain, Backplane* and *IV Curve. LED, Pinhole and IV Curve* tests were not available with the used setup.

Pedestal & Noise & CM test checks the noise and the pedestal levels. The ARCS collects data during a test and calculates pedestal values in every channel, with current setting. The results are drawn to a graphical presentation. With mathematical algorithms the program also calculates the noise of the channel. After the test the program gives every channel a colour depending on how good the channel is.

Orange = Pinhole, channel does not have enough voltage.
Green and light green = some minor problems on channel.
White = channel is working.
Red = channel is too noisy.

The program draws noise and common mode subtracted noise to the same plot. The program makes a histogram of these CM values to show how large CM is in every APV25-S1 chip and polarity. The common mode is difference of the measurement and real value.

Pipeline test tests every 192 pipelines of the APV25-S1 chips. The aim of this test is to find faults in the memory storage. The test reads every cell of the pipeline *n* times. This *n value* is defined by the user, but normally the value is 100. From this data the program draws pedestal graphs of every APV25-S1. After this it is up to the user to read the graph and say: “This cell is broken and these are working correctly”. Noise in the cell tells the cell’s ability to recover from changes. Calibration height draws a graph which shows how much channel should be calibrated so that channels would have equal pedestal. The user can check these values in every APV chips.

Pulse shape test gives test pulse to reading pads in every APV and reads the values through the DAQ chain. From the read data the program reconstructs the pulse in time domain and draws what kind of signal was given. The user can invert the pulse or make different strength pulses. This is good test to see if the APV chip can capture possible pulse at 25 ns resolution.

Gain test measures how much the signal attenuates during DAQ. During the test the program sends a test voltage to read-out end and reads the value with the DAQ chain. The test draws three plots: *Gain slope*, *Gain offset* and *Chi^2*. *Gain slope* plot shows all 128 channels gain values in function of the channel. *Gain offset* plot shows how much this *gain slope* value is in ADC count value. Chi^2 value measures how close the measured and theoretical value are. Lower *Chi^2 value* means that the channel is good.

Backplane test gives a backplane test pulse to the read-out pads and reads it afterwards. This test makes two plots. The first test measures the total difference sum of the sent pulse and read pulse in the events in channel. The second test measures the integral of between the measured and the theoretical value. If the integral value in the channel is negative, it means that measured values during the test have been lower than should be. This indicates that in some extend this is a bad channel. If the channel is optimal the integral value is near or equal to zero. The total difference and integral plots are shown as a function of the channel.

All tests make it easier to do the same test many times without worrying that some tests were not taken. The user selects a test he wants to do and runs the sequence. Test parameters are taken automatically from every test page. For example, the user wants to take 5000 events worth of raw data. The user opens Monitoring/ Data and puts in *# of events* box value 5000. Now when the user uses all tests, it takes 5000 events. Tests are selected by clicking a mouse in the box and after that pressing *Start All Chosen test* button. After the test starts no other function in program are not usable, but Abort button.

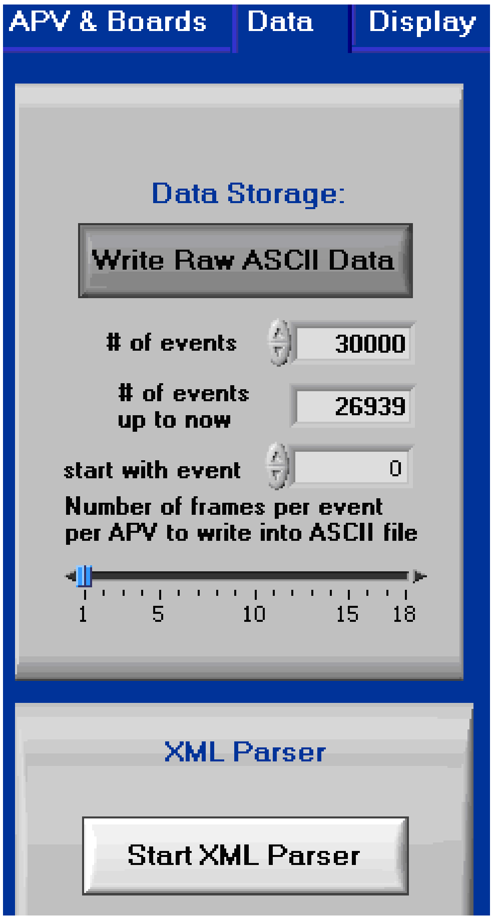
TCP/ UDP tap makes it possible to get environment data out of the ARC and also make it possible to control the cooling in the setup.

4.2.2 Collecting the raw data

Taking the raw data is one of the main tools in this thesis. During initialization phase the user is asked to name the found hybrids. So assuming that ARCS found two hybrids and the user gives them names “h1run100” and “h2run100”, these are the names that the program recognizes these two hybrids now with. The first step, in taking the raw data from hybrids, is to select the hybrid where DAQ takes place. The user selects from *Monitoring/ Display*, where two hybrid names “h1run100” and “h2run100” are listed, the preferred hybrid. Basically this tells which hybrid port is used in the ARC board.

After the user decides that the data is taken e.g. from hybrid “h1run100”, he goes to *Monitoring/ Data.* Before taking the data the user has to decide few parameters for DAQ. The first thing is to set event number, 20000 events is enough for testing. The starting point when DAQ starts does not need to be at event zero. The user can set it start later. The third setting is how many consecutive events program records per one trigger from every APV25-S1 chip in hybrid. This can be set 1 to 18 frames per event. When triggered, the program records next 1-18 frames of data. If the read-out is 20/40 MHz (50/25 ns pulse time) and take 18 frames in event, it means that the last piece of information is recorded 900/450 ns after the trigger. This enables the user to get data at a long period of time. This helps the user to define the latency or pulse shape easily.

DAQ starts when the user clicks the “Write Raw Ascii Data” button. This writes data from the APVs to ascii file. Ascii file is named after the hybrid “Raw_Data_h1run100”. Every time the user takes raw data from the same hybrid, its data is stored to this file, after the previous data. The APV settings are written to this file. Data is stored in format: Header (3 logical one), Address (8 bit), error bit (logical 1/0), 128 channels data (0 – 255) and tick mark, which marks that the data from this APV is completed. Recording data was easy. After that starts the analysis. In ARCS program there was not ready tool to analyse these files, so to analyse these files we needed to do an own code for the analysis.



# of events 30000

# of events up to now 26939

start with event 0

Number of frames per event per APV to write into ASCII file

1 5 10 15 18

Figure 10. This was the most important view in the ARCS program during this work. Using this panel the data was collected from detector.

5 Building the ARC system

The project started when I went to Helsinki Institute of Physics to learn more about this job. The first thing was to get to know about the physics part of the project. Although this was not necessary, it gave some clarification what was ahead. Another thing was to study working system so I could easily build same type system in Cern. Three days that I was in HIP I studied the ARC and also theory behind GEM detectors. This time was spent on studying ARC system and ARC board power source. I took pictures from ready ARC system that I could duplicate it in Cern. Studying ARCS software was mainly done independently. With Rauno Lauhakangas and Harri Heikura the bachelor thesis work was decided to be about building and developing the ARC systems.

In Cern we went through the goals with Leszek Ropelewski and decided that system should be in operation in two months. In the beginning it seemed it would be easily achieved in this time frame. The basic idea was to provide DAQ that could collect data about next generation GEM detectors.

The work in this thesis can be divided in two parts: hardware and software. These are explained separately, but some parts in work in hardware and software affected to each other. An exception is the ARCS software and other programs that were needed for running ARC system. These are included to hardware part of the thesis. The main part in the software discussion is the code and the macros, that were made for the analysing the data gained from detector.

5.1 Hardware

This chapter describes making the ARC system, building the control PCs, inspection of electronics and the overall progress in work in aspect of hardware. This chapter handles all modifications made to the ARC system and tries to explain why these modifications were made. These modifications include also the changes in software that were installed to control PC.

5.1.1 Control PC

Making the ARC system in Cern started by making the list of things that existed and what was missing. The missing list was short. There was missing two cables, between totem_apv_adapter_test_cad and hybrid, and also there were not enough APV25-S1 chips. Other things that were missing were Windows 2000 OS and a few memory combs. After these shortages were identified a request was send to HIP to get these missing parts. After these missing parts arrived to Cern the making of control PC started.

Control PCs crucial specification was ISA bus. This bus is not found in newer PCs so we had to go through older PCs to find a motherboard with ISA bus. Windows 2000 was installed to this PC. Next issue was memory shortage in the PC. By salvaging same type motherboards for right memory combs and using few memory combs got from HIP, the memory was upgraded. This upgrading was done by changing the memory combs in the motherboard and checking the memory capacity in BIOS. After finding the good combination in memory the hardware part was done in control PC. Now it was left to install all necessary programs to this computer.

By following the manufactures instructions [6] *GiveIO*, *Xerces 2.2.0* and *ROOT version 3.05/02* were installed to the computer. *GiveIO* is Dynamic-Link Library which makes it possible to access Windows PCs IO ports. *ROOT version 3.05/02* was needed for making root files in Windows OS. *Xerces 2.2.0* is needed for making XML files. *Root to Postscript Converter* program converts, like the name suggests, the *ARCS* program root files to pictures where the data is drawn and written. The *LabVIEW 6.0.2 Run-Time Engine* was installed to PC, because ARCS program, LabView application, or Root to Postscript Converter does not work without this program. Run-Time Engine was set to start in PCs start up. The last thing was to “install” the *ARCS 7.2 bug fixed* to the computer. This was done by unzipping the program folder to this computer. At this point the control PC was ready and next it was time to assemble the ARC system.

The setup that was build in Leszek Ropelewski`s lab in Cern. The control PCs were named as ARC#1 and ARC#2.

ARC#1 hardware configuration:	Processor 850 MHz
	Memory 576 Mb/133 MHz
	OS Windows 2000, service pack 4

ARC#2 hardware configuration: Processor 850 MHz
Memory 448 Mb/100 MHz
HD cloned with Ultimate Boot CD

With *Ultimate Boot CD* (UBCD) [13] OS of the ARC#1 was cloned to ARC#2. This was done to make sure that we have two identical computers running. These two PCs were same model so they should be compatible together.

In ARC#1 it was installed *ARCS 7.2 bug fixed*, because there was some problems with newer version of *ARCS 8.1*. ARCS is LabView program and to use it we needed to install also *LabVIEW 6.0.2 Run-Time Engine* [5] to get ARCS software working. ARCS also needs *ROOT Version 3.05/02* [5] and *Xerces 2.2.0* [5] programs to work. Instructions how to install these programs were found on the download site. To get some figures of the results out the ARCS program, the *Root to Postscript Converter* [5] was installed to the computer. This program reads the root files ARCS makes and makes plots in postscript format. To read those postscripts files the *Rampant Logic Postscript Viewer* program [14] was installed. This updates the *Adobe Reader* so it can open and read the postscripts created. General updating PC drivers e.g. DirectX and installing some other useful programs to PC were done, too.

5.1.2 ARC setup

It was decided to build the ARC system in one board table setup. All began by installing the PCMIO card in the PCs ISA bus. The next step was connecting all cables according to instructions in all parts of the system. The connections between the control PC, the ARC board and the FE adapter board succeeded without problems. There were minor problems with connection with hybrid and totem_apv_adapter_test_card in the adapter card end. From the beginning there was not a correct connection cable available, so we had to use self-made cable. Few times this connection was made in the wrong way. To prevent this, markings were made by pen to indicate which side must be up, when connected. At this point we had all the electronics parts and connections made, except the ARC board power source and its connections.

5.1.3 ARC board power supply

The ARC board needed its own power supply. This proved to be the hardest part in the installation. After the ARC system was built, a suitable laboratory power source was looked for to give operational voltages to the ARC board. The first attempt to do this proven to be a mistake and voltages in the ARC board were twice as high as the correct voltages. The problem was that the ARCS program was not able to found the ARC board at all. It was clear, that the board was not getting correct voltages. Because of this the building instructions given in HIP was read the second time and the solution was found.

The problem in the power supply was that positive and negative voltages were not grounded at the same point. This made the power supply to give more positive voltage to the ARC board than it should have. The ARC board was not working correctly. By grounding the power supply's end positive and negative side to same point, the problem was solved. This was done because in the ARC board there is only one ground, where the negative and positive voltages are connected. This ground need to be connected to the power supply's ground.

The ARC system was now functional and ready to use. There were not major concerns to alter the setup. With the ARC setup the user was able to collect data from one hybrid/ APV25-S1 chip. The next step was to use the setup and start to experiment the ARC setup and the ARCS software.

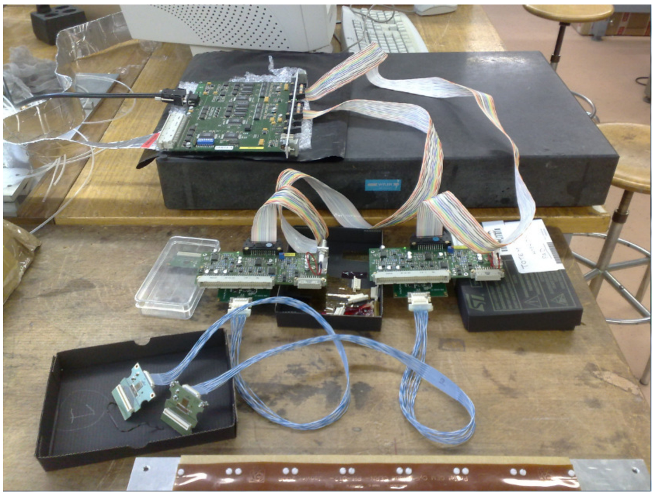


Figure 11. The first ARC setup. The setup looked like this almost the whole time. The only changes were made to the hybrid end. Later a detector was installed to this setup.

5.1.4 APV25-S1 test

When the working setup was ready, it was used as testing tool for finding the best two APV25-S1 chips. This was done because chips were handled roughly and the quality of chips was questionable. The tests were made to determine which two APV25-S1 chip would be the best to use in further testing. This eliminated some extend influence of the bad chip during tests. Also the FE adapter and the ARC board functionality were tested during this test period.

The tests consist of two parts: visual and functionality test. The visual tests were done with a microscope. Things that were looked for were broken bonds or other mechanical flaws. Functionality tests were done with the ARCS program. During the test a logbook was kept how well the chips behaved during the test. The test was made with *Fast Tests* and *Deep Tests* in the ARCS program. The test was done to found chips that caused the least errors during the testing.

The chip testing was made subjectively, not objectively. The results were graded based on the tester was observations during testing. So if it felt that the tested chip was good, it got good grading. The results on these tests were decided with a gut feeling, not scientifically. The tests were made in certain pattern. When an error occurred during the test, the chip in question got a minus point. The two APV25-S1 chips that had caused the least errors during the testing were selected.

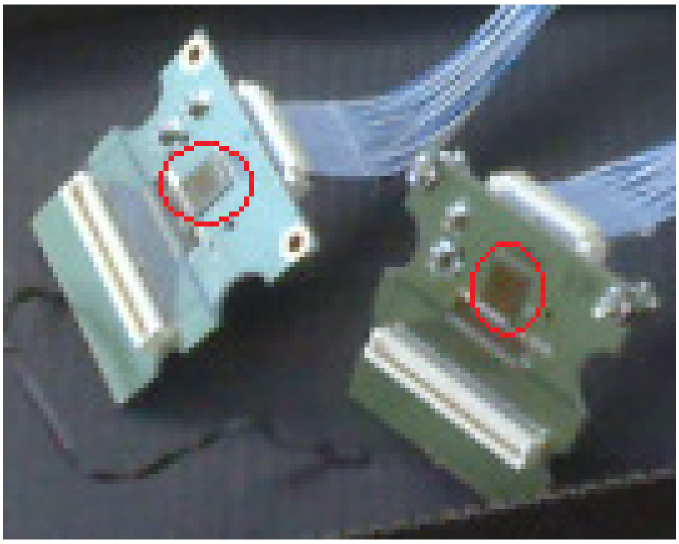


Figure 12. Two APV25-S1 chips marked in hybrids with red circles. The best chips were determined, but this was based on a good guess than on the hard data.

5.1.5 Postscripts files

The ARCS program makes three kinds of files. Those are XML, Root or raw ascii files. The user can read the XML and the raw ascii files with standard software in the PC. To read the root files, the *Root to Postscript Converter, a* LabView application, *was* installed on the PC. The converter reads the root files and generates pictures from the calculated values and saves in postscript figures. Then there was a problem, how to read these postscripts files, because this file type is not normally used. The problem was solved by installing *Rampant Logic Postscript Viewer* extension to Adobe Reader. Now the Adobe Reader was able to open the PS files.

5.1.6 Setups and testing

The ARC system was ready to use, but testing was needed to be done. The tests were concentrated on finding the good latency settings or trying to acquire data from the detector/ chamber and make the pulse height spectrum. For this purpose, dual timers, amplifiers, discriminators and chamber/detector were installed to the ARC setup. In these the tests the setup was altered a few times. The hardware setup modifications outside the ARC system are explained on ***5.3 Tests*** chapter.

5.1.7 Noise shielding

The ARC setup was grounded to the same spot, to minimize the noise. The reading pads, chamber, HV connection to the chamber, unused strips and aluminium shield covering the chamber were grounded in star configuration. There were not visible changes in noise, when the external noise sources were shielded. The noise in the signal was not external, it come from the ARC electronics itself. All electronics were not covered with aluminium, as it should have. This could have solved the situation. The PC, the ARC board and its power source, the FE adapter and the chamber were not grounded at the same potential. This can be also source of the excessive noise.

Because there were not benefits on covering the chamber with aluminium, the aluminium cover was removed from the setup. This was done, because the chamber has HV parts. If the aluminium cover and the chamber's HV parts would connect, that could have lead to 4.0 kV voltage trap. This is high enough to harm the user, who is currently using the setup. For safety reasons the shield was removed.

Too much noise in DAQ or the test beam experiments is a problem. Sometimes trying to get rid of it, it is hard to get all excess noise off. Noise comes from many places. Lights, cables, power sources, electric fields and other devices near measurement, give extra noise to the measurements and this is not wanted. In laboratory conditions it is easy to separate noise from electronics, but in the test areas there are too many noise sources to work with. In DAQ too much noise kills the good signal. This happens because the noise part is cut from the data and at the same time, this cuts parts of the signal. The less noise in the signal the better, after this we could hope good results.

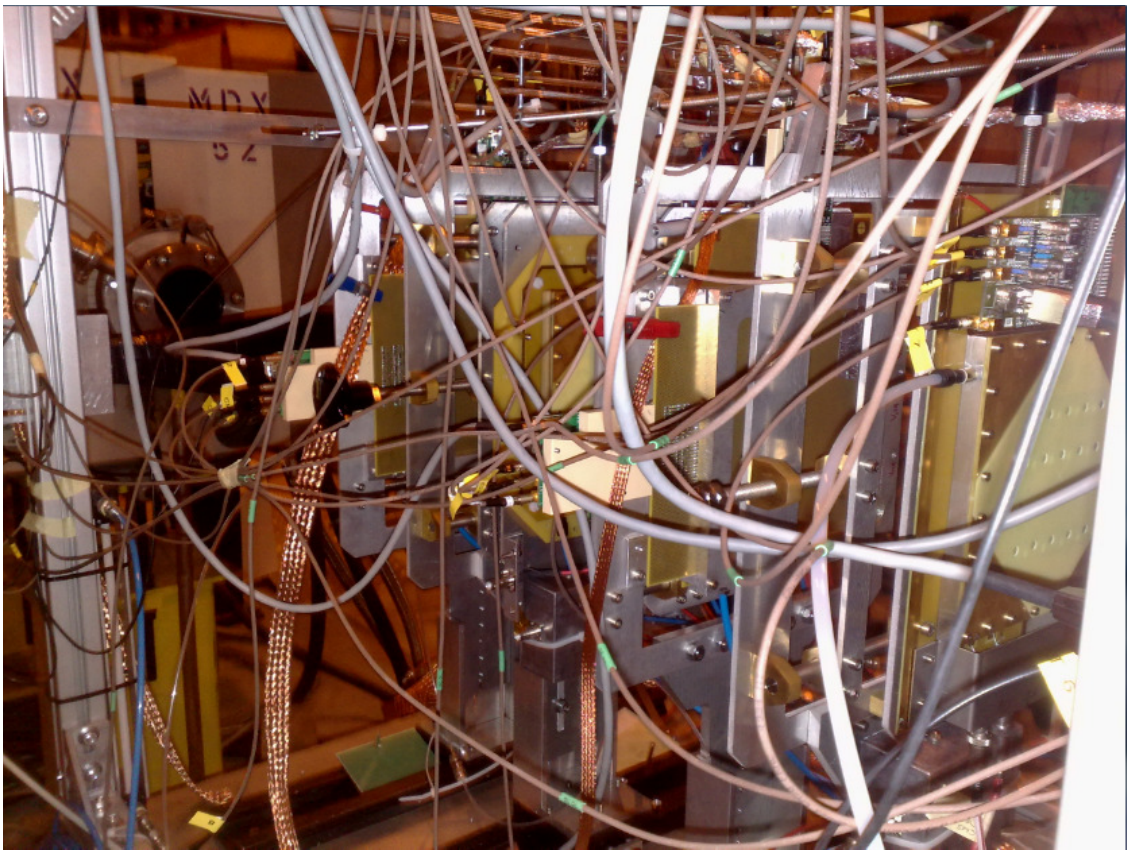


Figure 13. The Test beam 2. When taking measurements the reality is something like this. There are many problems to be solved, before the solution for too much noise is solved.

5.1.8 Two APV25-S1 chips reading

When the ARC board setup was ready, it came obvious this was not enough for the need of the user. It came clear that the current setup was needed to be upgraded, so the user could read two APV25-S1s at the same time. This was one of the main points at start of this work, but was left behind the other urgent things. The point is that the signal is divided with two string planes. So to able to reconstruct the signal it is needed to collect both set of string. It was not clear until the doing this thesis, that nobody had not done similar thing before with this system. It was decided to make two identical ARC setups. Both ARC setups read one APV25-S1 chip at the same time. The trigger signal was synchronized and given to the both board setups at the same time. After DAQ the data were taken from these two control PC ARC#1 and ARC#2. The raw data from the ascii file were taken and analysed, in keeping mind, that two data set make one signal.

Making a copy of the existing setup was easy, because all the necessary parts for this were found in Cern and HIP. The second setup was a clone of the first setup. The ARC#1 control PC hard disk was cloned using UBCD. This program clones PC system to another HD tract by tract. This way two identical control PC were made. At the same time the ARC#1 PC power was switched to another, because it was not working properly. The first try in cloning was not working, because UBCD made some automatic fixing process during cloning. In the next cloning this option was switched off and the cloning was a success. Also during this period it was noticed that because the system folder in PC was *e:/* the ARCS program was not working correctly. But the problem was fixed with another HD and naming this as *c:/*.

When making the another ARC setup, were the ARC#1 and ARC#2 control PC memory capacity noticed to be too small. The memory was upgraded to ARC#1 *576 mb/133 MHz* and ARC#2 *448 mb/100 MHz.* To be on the safe side, unlike the first memory upgrade, PCs were tested with memory test program. The program runs a set of random memory tests and kept records if faults in memory were found. The program ran the test over 17 hours (seven times) in the ARC#1 and 26 hours (nine times) in the ARC#2. No faults were found in the PC memories during the tests.

Figure 14. The small setup got bigger. The reading two x – y – strips in detector lead to duplicating the ARC setup. In this case we can't say this is any more a small test setup. This takes too much space.

5.1.9 Introducing trigger for the two ARC system

After making the second ARC system it was tried to give the same trigger signal to the two ARC systems, but this encountered some problems. Even though both ARC systems should almost be identical, few modifications after cloning, ARC#2 setup got more triggers than the ARC#1 setup. The trigger rate was twice as it was in the ARC#1.

To solve the problem the electronic configurations were changed in the ARC setups. First the FE boards and the hybrids were changed to one another. This change did not improve the trigger rates and ARC#2 got still twice more triggers. Then the ARC boards were changed head-on, so the ARC#2 PC got the ARC#1 PCs whole electronics from the ARC board to the APV25-S1 chip. But again the problem stayed on the ARC#2 PC DAQ. Next the PCMIO cards were examined and the cards had the same settings in the board. Next step was to look if the cause of this problem lied on the software. The ARCS and the GiveIO were reinstalled, but the problem was still there. The PCMIO cards were not changed, but rest of the electronics were changed from setup ARC#1 to ARC#2. The reason for the different trigger rate was not found. The problem may lie in the PCMIO card the settings are different in these systems.

Because we could not provide the same triggers to record the events at the same time, the idea of using two ARC setups was abandoned. This was a major setback, because without possibility to read two set of strips at the same time, it rendered this setup useless. The user of this ARC setup needs to get the data from both planes to get the full signal, because the electron that goes through the detector leaves voltage that is divided in two sets of strips. To collect the one charges or events data, the data from the both sets of strips had to be put together.

5.1.10 The current ARC Setup

The setbacks in reading the two set of strips forced us to make the one ARC board setup working. There were now two ARC systems in the laboratory, but the ARC#2 system reliability was a question. With the current setup the user can collect raw data with ARC#1 and it works for sure. The rest ARCS program functions are not usable or not needed in use it was intended. Those functions were meant to be used in quality checks. With this current setup the user can collect the data with the ARCS setup and then analyse it. For the analysis a code was made to look for right things in the data.

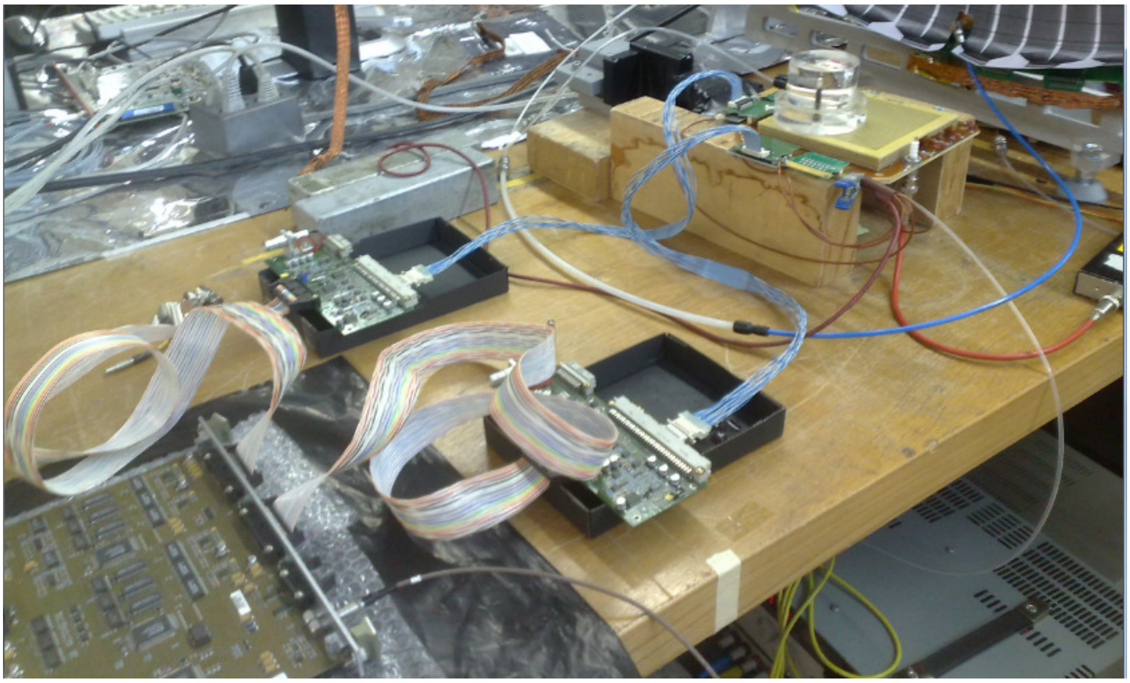


Figure 15. One ARC board collects the data from Triple-GEM. In the ARCS the user selects the switch hybrid used in the DAQ.

5.2 Analysing code & macro

The thesis did not only consist of putting the electronics together, but also included programming. The software was made to analyse the raw data which was collected with the ARC electronics and the ARCS program. The purpose was to analyse the data that was collected and display it in plots. The software development started from the assumption that the data was collected by using the ARCS program's internal function of taking the raw data. Two options were given: LabView approach or making the code with a familiar programming language. I thought that making a LabView program inside the ARCS would be really hard, because of my inexperience in the LabView programming. I decided to make the analysing code using the C# programming language. The reason for this selection was my earlier knowledge of C#.

5.2.1 C#

The goal in the C# was to make a program that reads the data from the ascii file and analyse it. Then the data would be presented as plots in the screen. The program should draw the pulse height spectrum (PHS), the noise distribution in channels and plot Gauss and Landau fits. These were the main points what the program should do. This is discussed in general level and not in detailed manner. Examples of the code are presented in the appendices for comparison to the ROOT approach.

Trying to make an analysing code with basic C# skill and make all things from scratch was really like inventing a square wheel. The main issues are discussed that were needed to be solved, before analysing the data with this code.

Reading

The program starts by reading the ascii file with lineader function. The LineReader reads a one line at time and reads the whole document in side the loop. The nullexception in the code prevents the program from crashing when the linereader reads nothing. Help for this reading method and other things in C# programming were got from SAMS TEACH YOURSELF C# in 21 DAYS [15]. The program makes some filtering and cutting strings during the reading. This data is then put into the arrays and table. Examples how this was done are found in the *Appendix 1: Linereader*.

Saving data to arrays

The data was saved to the arrays before it could be processed. The first attempt in saving data to the arrays ended badly and a code with over 400 variables was the result. The total length of the main code was over 4000 lines. After rethinking, the number of variables dropped to 20-30. These were arrays and tables in the program. This change dropped number of the analysing program's main lines to 400. This number of code lines was only the main code at certain point time. The total number of C# code was much larger.

The major error was to make every channel its own array. This was done because we did not know the number of events we were processing at time. That excluded the first choice of using table to save the data, because it needs specified size at beginning of the code. After noticing that the arrays were not cutting for this a major shift was made. The data was read and saved in one table, which size was 100 000*28. This is a large table and if the number of read events is less than 100 000 there is not error calculations. From this table the channel data is read during the calculations.

Looping

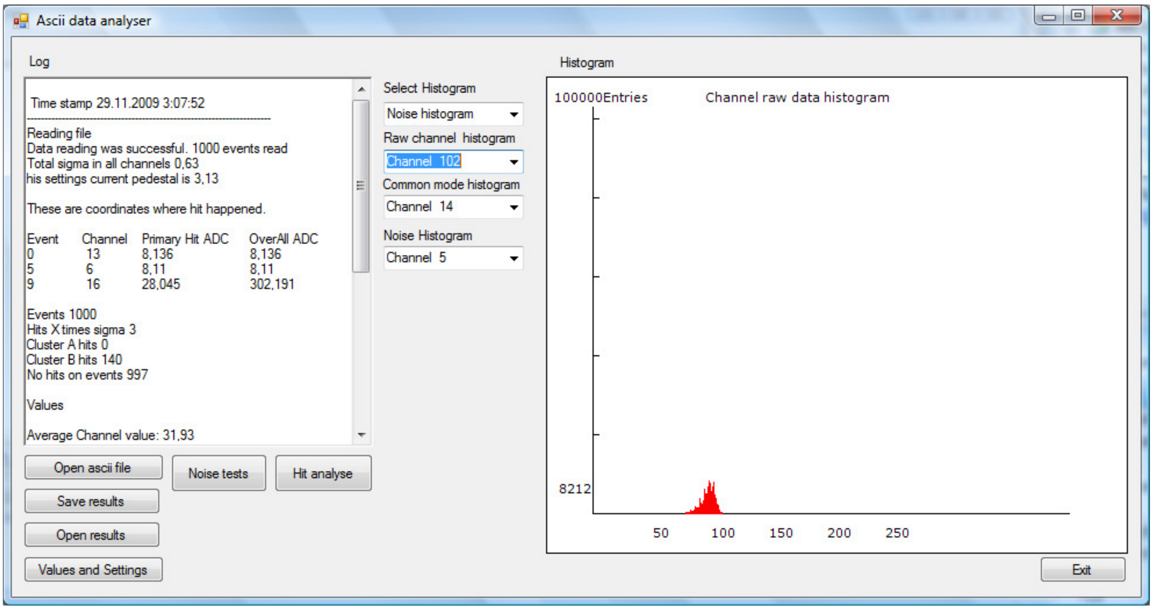
The data was written to array and out by using the loop structures. The data in one event is written from all the channels to the table. The program saves all the 128 channels data in a row of the table. The program fills the table until the LineReader reaches to the end of the file. When data is stored in the table the program uses that data to calculate different values. When calculating noise or pedestal values the program takes the table and runs it channel by channel. This means that if the table holds 10 000 events of data, program runs 10 000 events and calculates the necessary values. After this the calculated values are saved in the table which size is *X**128 channels, where the *X* is number of events. When the data is needed it will then be printed out from the table, like it is tried in the *Ascii data analyser* GUI. *Appendix 2: Looping* has a code example of this.

Graphical User Interface

The principle of the GUI was to be easy to use. The visual part of the program has two windows: settings and general view. In the settings the user can make changes to how many sigma we consider to noise in the data to be and the program makes its calculation based on this value. The settings window should give numerical values of the data after it has been analysed. The general view gives the option to open the wanted data file and analyse it. At the same time it should make a log file of the calculated values. These log files were dated and saved to the same file. Not finished function was to try look hits (particles crossed the chamber) in the chamber. This gives information how many times the value of the level that the user defines in settings is achieved in chamber. This also gives event number, channel number and even some cluster size. But this was unfinished and does not give correct values. In pedestal runs the user can calculate noise and the pedestal but not much else.

Visual presentation

After calculating the noise and the pedestal values to the arrays these values can be then drawn into general window. This is done by selecting channel from the list and its data is shown in plot in the screen. Making GUI was difficult and making the fits in the graphs was more difficult. This based on making graphical objects and trying to make coordinates where data was drawn.



```
Time stamp 29.11.2009 3:07:52
-------------------------------------------------------------
Reading file
Data reading was successful. 1000 events read
Total sigma in all channels 0,63
his settings current pedestal is 3,13

These are coordinates where hit happened.

Event    Channel    Primary Hit ADC    OverAll ADC
0         13        8,136              8,136
5         6         8,11               8,11
9         16        28,045             302,191

Events 1000
Hits X times sigma 3
Cluster A hits 0
Cluster B hits 140
No hits on events 997

Values

Average Channel value: 31,93
```

Select Histogram: Noise histogram

Raw channel histogram: Channel 102

Common mode histogram: Channel 14

Noise Histogram: Channel 5

Figure 16. The Ascii data analyser. This is a picture of the analyser program that was made to analyse the ascii files produced in the ARCS. Open *ascii file* button opens file and at the same time puts the data in the memory. *Save/open* results is the log text screen. Values and setting opens windows where the user can change few parameters in program. The noise test makes calculations which are shown then when selected in Histogram window. Under *Select histogram* the user can choose what channel is shown in right side of the screen.

Math calculations

The program calculates the noise of every channel which is standard deviation from the mean of the channel. Common mode subtracted noise is done by calculating every channel mean and then getting the standard deviation of the channel averages. The CM noise is calculated by subtracting CM value from the channel's noise value. CM is the standard error in the channel and this can be taken out from the noise to see how much the external noise is. Appendix 2: Looping shows how the calculations were made.

The noise distribution is the histogram of the noise. Normally this graph is centred on zero, with most occurrence, and if the detector is good its occurrence drops fast when going off from the zero. Every channel calculated noise is sorted in 0.5 slots, going out from zero in both directions. The loop counts the noise value occurrence to the array. Occurrence of the certain value is put into histogram by multiplying the counts by factor *x* and drawing this presentation by using the graphic pens to the general window. At the same time a gauss fit is plotted to this graph. This part of the code was not finished at the end. Some of the fit made were possible but mainly the fit results were really absurd.

The next step was to try making the pulse height histogram. This was done by calculating the sum of charge that is over threshold in all channels in one event and filling histogram with this sum. This was done to all events. The graph shows what kind of signal we can except from the detector. Drawing this is done by similar way as making the noise distribution. Trying to do this and making fit to this proved to be hard task or even impossible in C#. Before solving this problem a decision was made to cancel this C# code project.

Results in C# analysing code

In the start of making this C# code it seemed like hard work. In the middle of it changed to be easy and in the end it was useless. If I had more information about ways to analysing data, this could have been really different from the beginning. It was good to learn C# again, but it was the wrong time and place. The total time that was spent on this code was nearly 3 – 4 weeks. The code made during this time was useless. The code and what we could do with it was presented to Leszek Ropelewski and it was decided to try another approach.

By using the program, analysing the data took too much time. Drawing the graphs was not made well and the fits were nowhere near they should be. The calculations were too time-consuming to be changed. Putting the code together was bad, so it was decided to switch to the ROOT. Trying to make analysing program in C#, when something like ROOT exist, was a false step.

5.2.2 ROOT

The ROOT is object oriented framework, which provides the tools of analysing large amounts of data in very efficient way [16]. ROOT is used widely in field of physics because it has many tools of analysing the data. The ROOT is a good platform to build data analysis and simulation systems. The ROOT can use the external libraries during its analysis, making it easy to make changes to its calculation routines. Implementing macros in C++ code the user can easily make modification to calculations and use ROOT more efficiently in the analysis. Using ROOT does not require so much coding skill, because the user has to know only the commands, how to ask the ROOT to analyse the data. The ROOT works in Linux environment. To use this program the user needs to have Linux environment in his computer to be able to use the ROOT.

The ROOT is an open source program working under LGPL licence [17] which guarantees the use of this program freely. In making the ROOT the users have been able to tell how the ROOT could better serve its users. This has made the program better, because the users are constantly saying how to make ROOT better, because the users knows exactly what they need from ROOT.

Starting with Root

The final objective was to make code in ROOT that reads the data from the ascii file and makes histograms from the data. The histogram was fitted with Landau fit and the user can also do some other fits. The histogram gives good picture what kind of signals were gotten from detector and what is the noise level. In ROOT it is possible to do some other calculations, too.

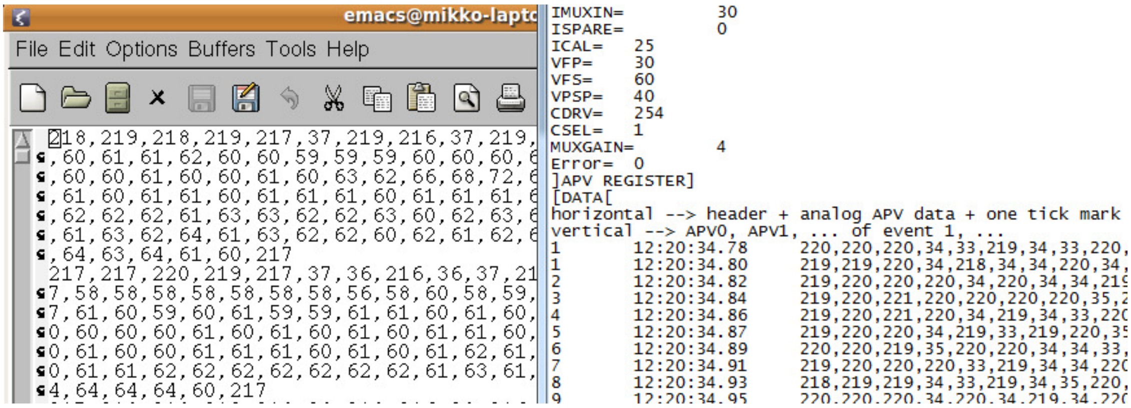
The first thing in starting to learn the ROOT was to open extensive manuals and read how to use the ROOT. Matteo Alfonsi (PH/DT) and Gabriele Croci (PH/DT) gave much help in the start of this analysis code. They were my co workers during the thesis. The basic structure of the code was made in two or three days. ROOT has extensive libraries of routines and functions and all the code was not needed to be made again. ROOT has all basic tools in analysing data in large amounts. This has been made directly into the program. With ROOT in background and having few macros made in C++ the raw ascii files were analysed.

Filtered raw data file

The ARCS software raw data function saves data into the ascii file. From this file the data was extracted. The data was filtered in Linux environment by taking the ascii raw data file in file window and making another file using following command line.

```
cat Raw_Data_1402x.dat | grep ':' | grep -v 'Date' | awk '{print $3}' > run1402x.dat
```

This line takes the source file “Raw_Data_1402x.dat” and makes modifications to the text inside and saves this data to another file “run1402x.dat”. Step 1: *grep ‘:’* command takes all the lines that has mark “:“semicolon in the line. This command filters off all the APV register value and text lines at the beginning of the file. Step 2: *grep –v ‘Date’* command does not take the lines that has the word “Data”. Now the lines that have measurement data on it are written to the new file, but in the line the are also the *event number* and the *time* values in these lines. The measurement information is the third string in the line, so we want only select this string to be written to the new file. The string is selected with *awk ‘{print $3}’*. This code transforms the output by printing only the third string in the line to the new file. [18]



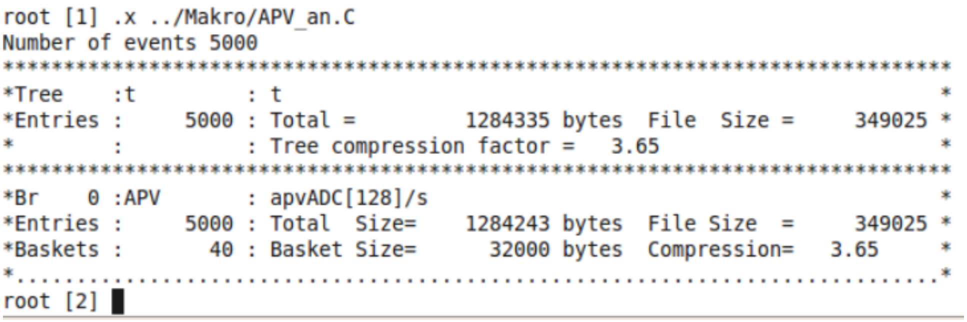
```
218,219,218,219,217,37,219,216,37,219,
,60,61,61,62,60,60,59,59,59,60,60,60,6
,60,60,61,60,60,61,60,63,62,66,68,72,6
,61,60,61,61,60,61,61,61,60,61,61,61,6
,62,62,62,61,63,63,62,62,63,60,62,63,6
,61,63,62,64,61,63,62,62,60,62,61,62,6
,64,63,64,61,60,217
217,217,220,219,217,37,36,216,36,37,21
7,58,58,58,58,58,58,58,56,58,60,58,59,
7,61,60,59,60,61,59,59,61,61,60,61,60,
0,60,60,60,61,60,61,60,61,60,61,61,60,
0,61,60,60,61,61,61,60,61,60,61,62,61,
0,61,61,62,62,62,62,62,62,62,61,63,61,
4,64,64,64,60,217
```

```
IMUXIN=         30
ISPARE=         0
ICAL=   25
VFP=    30
VFS=    60
VPSP=   40
CDRV=   254
CSEL=   1
MUXGAIN=        4
Error=  0
]APV REGISTER]
[DATA[
horizontal --> header + analog APV data + one tick mark
vertical --> APV0, APV1, ... of event 1, ...
1       12:20:34.78     220,220,220,34,33,219,34,33,220,
1       12:20:34.80     219,219,220,34,218,34,34,220,34,
2       12:20:34.82     219,220,220,220,34,220,34,34,219
3       12:20:34.84     219,220,221,220,220,220,220,35,2
4       12:20:34.86     219,220,221,220,34,219,34,33,220
5       12:20:34.87     219,220,220,34,219,33,219,220,35
6       12:20:34.89     220,220,219,35,220,220,34,34,33,
7       12:20:34.91     219,220,220,220,33,219,34,34,220
8       12:20:34.93     218,219,219,34,33,219,34,35,220,
9       12:20:34.95     220,220,220,34,220,34,219,34,220
```

Figure 17. Ascii files.On the right there is the original file and on the left there is the file which have been filtered. You can see that the data part has been transferred to the new file.

Transferring the data to the tree

The next step was to make this data more usable and in smaller file size. The data was saved in a tree [19].Tree is a way to storage data. This makes the data more compact and makes the rest of the steps of processing data easier. The file size drops considerably in this process. Running *APV_an.C* macro in console makes this process. The data file name was changed in the code every time when analysing different data file. The data file needed to be in the same folder where the ROOT was launched at. The code of the macro is in *Appendix 3: Saving data to Tree* and it is explained there more detailed.

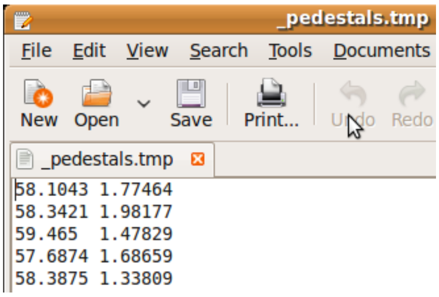


```
root [1] .x ../Makro/APV_an.C
Number of events 5000
******************************************************************************
*Tree    :t         : t                                                      *
*Entries :     5000 : Total =         1284335 bytes  File  Size =     349025 *
*        :          : Tree compression factor =   3.65                       *
******************************************************************************
*Br    0 :APV       : apvADC[128]/s                                          *
*Entries :     5000 : Total  Size=    1284243 bytes  File Size  =     349025 *
*Baskets :       40 : Basket Size=      32000 bytes  Compression=   3.65     *
*............................................................................*
root [2] █
```

Figure 18. Using the APV_an.C macro in the console. Running this macro in the console gives data of tree to the screen.

Acquiring the pedestal and the noise data

The data will be analysed with the *ARCanalysis.C* macro. This macro gets the mean and the sigma of mean values from pedestal run file. The pedestal and the sigma data are written in the _pedestals.tmp file after they are calculated in the macro. This file will be used when the data run is analysed. At the same time this macro puts all the data in the PC memory. This is not done in the next step, but this will define what files are used in the next macro *totalcharge.C*. The ARCanalysis.C macro is in *Appendix 4: Mean and Sigma.*



```
58.1043 1.77464
58.3421 1.98177
59.465  1.47829
57.6874 1.68659
58.3875 1.33809
```

Figure19. Mean and sigma values in _pedestal.tmp file.

Making the Pulse Height Spectrum

What is Pulse Height Spectrum?

It is a histogram which shows how often certain value is occurring in the evaluated data. In this case the program takes data and search channel by channel and determinates if there is signal at that point of time. This is done by measuring channel's pedestal values during series where there is no sure signal and the same time it calculates the sigma of the pedestal which is the noise. This is calculated by using standard deviation [20]. This kind of histogram can be seen in Figure 20 on page 36.

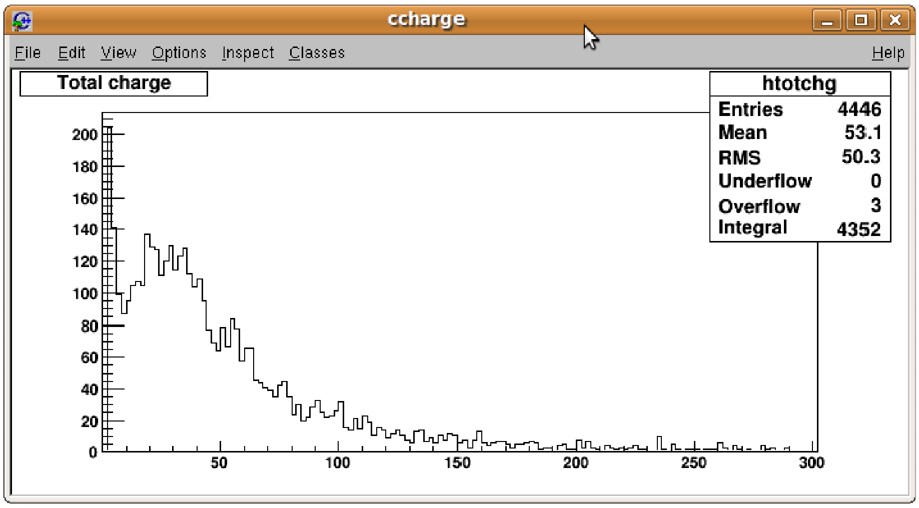
The totalcharge.C macro

The Totalcharge.C has three steps: start, run and finish. First the preparation for event, second the work itself and the lastly displaying the results in the histogram in screen. The totalcharge.C macro is in *Appendix 5: Totalcharge.*

When the macro ARCanalysis.C was ran all the experimental data was put in the PC memory. The totalcharge.C macro refers to that memory space and uses that data from the memory. First, the macro totalcharge.C does preparations like setting what libraries are to be used in the code, deleting/ making histograms, gets the pedestal and pedestal sigma values from file *_pedestals.tmp* and asks parameter that affects the outputted PHS histogram. Inside the `void totalcharge::Begin(TTree * /*tree*/)` paranthesis these preparations are done.

The second stage in this code is `Bool_t totalcharge::Process(Long64_t entry)`. Inside this paranthesis all the work on the data is done. Because the data is saved in the tree, the code fetches the on entry (event) a time and analyses every channel`s values at time. Every measurements are evaluated channel by channel to see if there is something that is interesting in the channel. In short if the value, in this point of time is higher than this channels pedestal value, plus sigma (noise) times factor *x* (default 3), the code decides that there was a signal at that point of time. If this happens, the code sums this value over pedestal in this event. When all channels in the current event are checked, this sum of signals of the 128 channels is filled in the histogram. At the same time the macro keeps a track, how many times in one event channels had signal in it. This means every event will have the "cluster size" 0 -128 which is filled in the histogram. The last thing in the code is to reset the `sum_allchannels=0;` variable for the next event. The code keeps analysing the tree until the last entry and after that the code enters the third part of the code.

The last part of code is `void totalcharge::Terminate()` that ends this macro. This third part of the code prints the results in two histogram: the *total charge* and the *treshold*. The total charge is the PHS histogram and the treshold is kind of cluster size histogram. The treshold will show how many times certain number of strips had signal at same point of time.



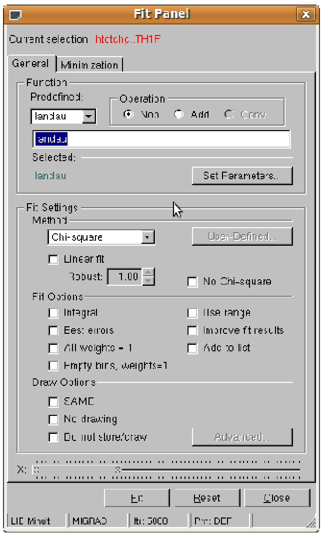
htotchg

Entries	4446
Mean	53.1
RMS	50.3
Underflow	0
Overflow	3
Integral	4352

Figure 20. PHS gives the distribution of the signals. From this picture we can say that there is no clear separation of the pedestal and the signal. A possible reason for this is too much noise in the signal. Separation means that there is a smaller peak before the real signal and in this figure the signal peak should be more right. The Most Probably Value (MPV) tell more about this histogram, more about this in **Fits.**

Fits

After obtaining the *Figure 20* on the page 36 histogram, we need to analyse it. This was one of the main points when starting this project. The ROOT gives good tools for the analysis. With a right-click in histogram can be selected Figure 21 on page 36 Fit Panel. In the fit panel the range of fits and what fits are used for be can changed. Depending on the settings in the histogram, this fits automatically prints many values to the histogram.



Current selection htotchg::TH1F

Predefined: landau

● Nop ○ Add ○ Conv

landau

Selected: landau

Method: Chi-square

□ Linear fit

Robust: 1.00

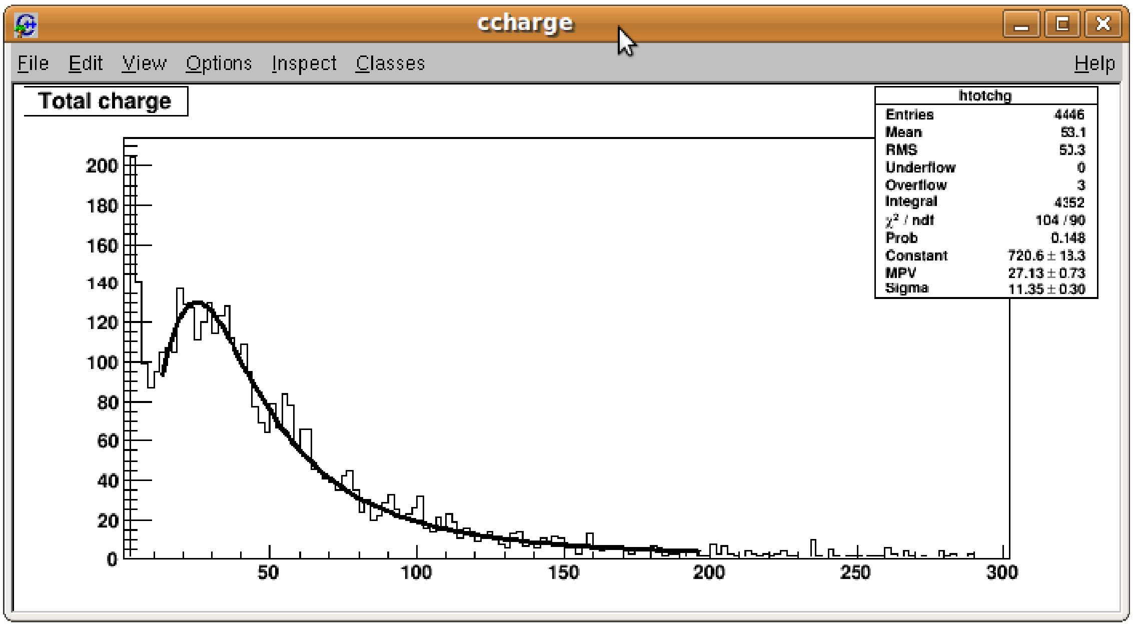
□ No Chi-square

□ Integral
□ Use range
□ Best errors
□ Improve fit results
□ All weights = 1
□ Add to list
□ Empty bins, weights=1

□ SAME
□ No drawing
□ Do not store/draw

LIB Minuit | MIGRAD | Itr: 5000 | Prn: DEF

Figure 21. Fit Panel.



htotchg	
Entries	4446
Mean	53.1
RMS	53.3
Underflow	0
Overflow	3
Integral	4352
χ^2 / ndf	104 / 90
Prob	0.148
Constant	720.6 ± 13.3
MPV	27.13 ± 0.73
Sigma	11.35 ± 0.30

Figure 22. The histogram after fitted with the landau. Notice the change in the right upper corner. Using fits needs some time as well understanding what these fits are measuring. The most important value in this exam was the MPV or the most probably value. This tells right away how big signal we can except from a detector if there are a hit or hits. Altering settings or setup the user can see the changes in read signal in this figure.

5.3 Tests

The tests took more than 50 % of the total time of the work. These tests were made to gain understanding, how the user can change the settings and use the ARC setup to its fullest. Later on testing it was targeted on making the latency scans and PHS from the chamber. These tests are explained in four category but these were only the main tests. Some smaller tests were made along the way.

5.3.1 Testing the electronics

These group tests were more like practising with the electronics. The main goal in these tests was to get an idea how to take data. This data was used in making the C# analysing code. In these tests the raw data was collected and used in C# analyser. The first attempts in DAQ were meant to calculate noise in the APV. In the data run there were taken normally 10 000 events. These test had not any clear purpose, only random DAQ. The data was taken from bare APV that was not connected to any device. The data was collected with random software trigger.

5.3.2 Latency test with the dual timer

The second set of test was made more professionally, with a clear objective. The purpose of the test was to pin-point the best latency value in the ARC setup that was connected to the triple-GEM detector. These tests were made professionally; it had a clear objective and the environmental data affecting to the test were recorded to the logbook.

The tests started by making a pulse signal with the dual timer #1. This was set to give pulses in one time interval. The pulse was given in three different devices. The first pulse was directed to the “antenna” which was placed over triple-GEM reading pads. When this NIM signal circulated over the “antenna” it made a signal to the reading pads under it. The antenna is Lim cable with a loop at the end. The second device where the dual timer#1 pulse was sent was the dual timer#2. When the pulse came to the dual timer#2 it delayed the pulse. After the delay time was over the dual timer#2 sent a new pulse which was the trigger to DAQ on the ARC board trigger port. The third device was the oscilloscope. The delayed trigger signal from dual timer#2 was also sent to the oscilloscope.

The test was made by setting the trigger (timer#2) and signal (timer#1) in the oscilloscope. The delay values were changed and changes in the ARCS program’s *Monitoring* screen were observed by sight. This was done by changing delay in dual timer#2. By altering the delay in the dual timer#2 the trigger pulse was moved in the oscilloscope. When the best delay was set the events had more pulses in the ARCS *Monitoring*. At first, the delay was set to zero and then going fast it was delayed by 4 µs, to seen the overall picture. Then with steps of 0.5 µs the trigger was delayed up to 4 µs. In every step the delay was written from oscilloscopes screen and the strength of the captured signal was graded by sight in the ARCS program *Monitoring*. The best delay was decided based on the evaluated data.

The best delay using this method was estimated to be around 2-3 µs. But this test heavily relied upon the operators subjective thoughts. So this test was not so accurate and needed to be made in a different way. Also the way of setting the delays was not so good, because the manually altered delay in the dual timers is hard and you need to be able to access the setup all the time.

5.3.4 Latency tests with software settings

The first two groups of tests were made more or less based on the visual observation, but in the third test group the electronics and the analysis code took part. This test purpose was to alter the trigger signal latency using the ARC system internal setting; or to be more precise, to change the APV25-S1 chip register values. The variable that was changed during test was the latency. One step change in the latency register means 25 ns delay in DAQ.

In this test one APV25-S1 chip was connected to the triple-GEM chamber. Over the triple-GEM a strontium source was placed which shoots particles as it went halves. These particles go through the triple-GEM and the chamber's behaviour is observed as voltage in the strips. After this those particles arrive to the scintilator under the triple-GEM. The scintilator (photon multiplier) created the trigger signal when an electron went through. This signal was sent to the preamplifier. From the preamplifier the trigger was send to discriminator. Discriminator made a clear signal from e.g. sine wave as the voltage in the input was rising - or falling edge. The trigger signal (square wave) made in discriminator was lead to the ARC board trigger port. The DAQ read the strips in the chamber and collected the signal as the particles had gone through. This data was then written to the ascii file.

Before making the tests the settings were determined. This was done by setting a good APV registers values and mode to the APV. The mode in this test was important. Depending on the mode the latency which was needed to collect the data from APV was different. The test was done by collecting the raw data with different latencies. The collected data was analysed with the ROOT code. The MPV value, from Landau fit with different latencies was written down and put into an Excel chart. The results were plotted with the corresponding latency value. This chart is on page 40.

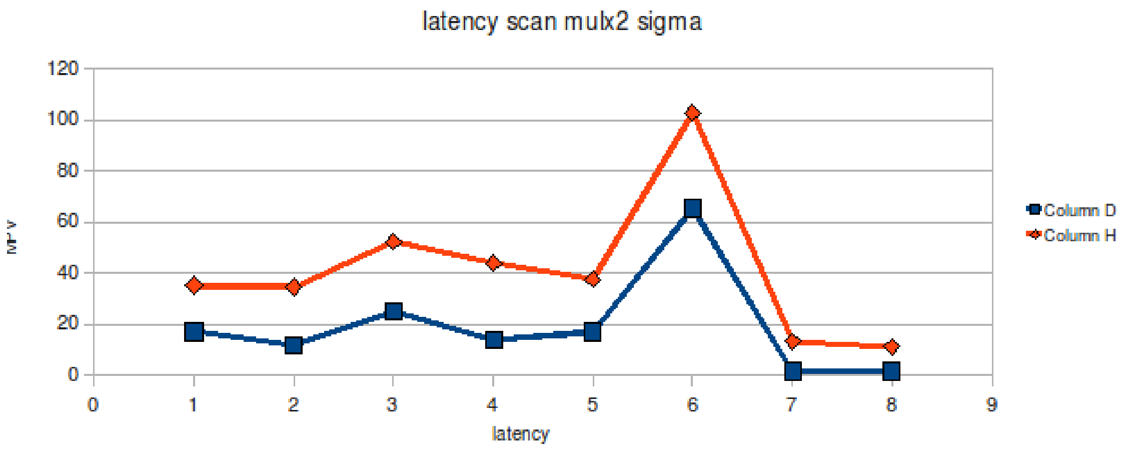


Figure 23. The graph shows how the amount of collected signal changes with different latencies. The best latency is where the MPV is the highest and in this case it was when latency was 6 in register. Converted from the clock pulses the best latency was 150 ns.

The test starts by making a logbook which holds the basic information about current test like environmental data, run names, run type or what were the used settings. The test run started by giving a run number or name for the hybrid before data run. The run name had to be different every time, because otherwise the new data would be written after the last one. This forced the user to stop the program after data run and make the same preparations before starting taking new data. This was done because the analysis program could only process file with one data run. The first run in the tests was the pedestal run. From the pedestal run the noise and pedestal levels in each channel were determined. After the pedestal run there was the latency scan. The latency scan started at value 1 and was increased until the signal was not visible in the display.

After acquiring the data with selected latencies the data were analysed with ROOT code. In the ROOT the data was filtered and put into a tree and after that it was analysed. The Analysis was done by making Landau distribution fit to the PHS histogram. From this fit the user took the MPV. The idea of this test was to find out the best latency by finding the highest MPV with some latency value. This was achieved and the best latency in this setting was determined to be 150 ns. This latency test was a big improvement when compared to the earlier latency test. In this test the data was analysed with ROOT program, so the test did not relied on the user judgement.

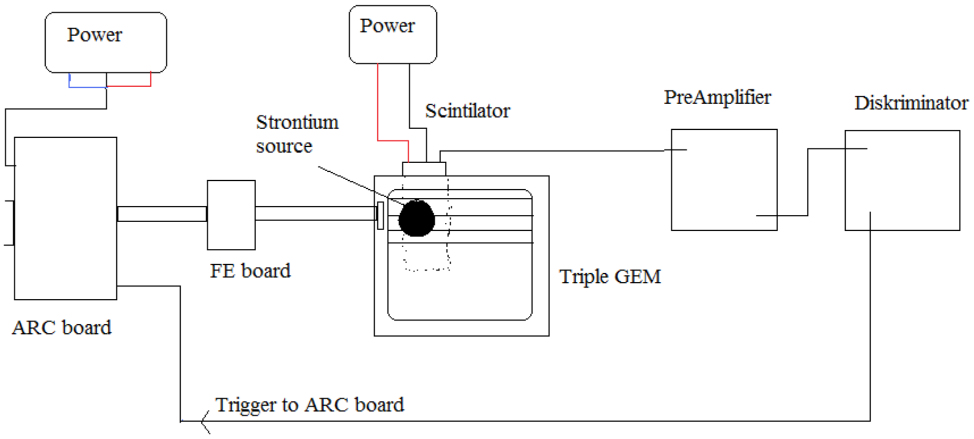


Figure 24. This is the drawing of the setup. This describes the trigger signal route. The strontium source scatters and particles go through the detector, where energy is given to the strips, and then those particles reach to the scintilator. The scintilator makes pulse when an electron goes through and this is lead to PreAmp. After this it goes through discriminator and then as a trigger signal to ARC board trigger board.

5.3.5 Taking data for the Pulse Height Spectrum

The previous tests before this were only practise for the PHS test. The first test was getting to know about the electronics, the second test used hardware to alter the latency and the third test concentrated on finding the good latency value using software settings and analysed the data with ROOT macros. The fourth test uses all the knowledge and methods used in the previous tests. This test purpose was to use the best measured latency to draw PHS histogram. The setup was the same as in the third latency test.

The PHS test was a similar to the latency test, but in this test the fixed latency 6 was used. The analysis was done at the same time as taking the data. In the analysis was the code made in ROOT that could make a spectrum of signals that were over the threshold. Modifications to the earlier APV register settings were done and this lead to changes in the macro code. During the test different modes in APV were used and this lead to changes also in the analysis.

The data can be read in three different modes from APVs: 1 sample, 3 sample or deconvolution. All the earlier tests were done using 1 sample mode, but after reading APV manual [13] it was decided to try also other reading modes. This experiment lead using 3 sample mode because it is more suited for looking the good PHS histogram. The 3 sample modes take three consecutive frames of data per one trigger. This makes it possible to get more accurate picture of the signal in the strip. This led to lowering the latency value to 4 because now the “best 6 latency” was in the middle of the three consecutive frames. The Deconvolution mode was for the high rate of DAQ and it needed much larger latency value and because during the test no signal was found with this configuration.

Although a better mode to get good data was found, the signal was not separated from the noise in the PHS. This was thought to be because of no good shielding or grounding, but after making groundings the PHS was not getting any better. The other solution was to increase the gain of the triple GEM by increasing the voltage in the chamber. The HV was put from 3.8 kV to 4.2 kV which was the safety limit. Even after these measures the signal was not separated from the noise. This was seen in the PHS histogram because there was not a smaller value peak before the main peak. The collected data was then analysed and landau fit was done to the histogram. The Landau fit must be done well or otherwise the results are not good.

5.3.6 Future tests

The important test in future is to read two APVs at the same time if it is possible with this ARC system. These two APV then makes x/y coordinates and these are then read to get the full charge collected from triple GEM detector. This is necessary because when a charged particle goes through triple GEM the charge is divided for these two sets of strips. To get data from the detector it’s necessary to get data from the both coordinates. To do this the second ARC#2 systems was built up, but some problems with this setup meant that this was not possible.

6 Summary and conclusions

With the ARC setup the user can collect data from the detectors. After acquiring data the user can use the macros in ROOT to determinate what kind of signals we are reading from readout. This is presented in PHS histogram in ROOT. The histogram can be fitted by using *Fit Panel.*

Making the setup went well but there were some problems with the analysing code. Wasted time on trying to make analysis in C# was a bad mistake and it took time from other things. With the ROOT it was easy to make the analysing code. With the help of ROOT I managed to make the code that analysed the acquired data and after that it was only about making the adjustments in the code. Making the code, using root and analysis went well in ROOT.

The tests evolved and in every setup something was changed and it kept testing fresh. Testing was a new thing and I learned few things. Theory was clear but I did not always understand what I was looking for in the test. I did not see the big picture. The tests were necessary part in the work to get the necessary data for the analysis. Other point in the testing was that it showed that the electronics and software were really working.

After the ARC setup was ready two manuals *ARC User Guide and HowTakeRawDataWithARCS* were made. *ARC User Guide* goes through mainly ARCS program and its possibilities. This was 31 pages long, but it does not give user so much useful information about the use of the ARC setup. *Appendix 6: HowTakeRawDataWithARCS* gives the user instructions how to collect data with this setup. It is only four pages long, but is more useful to the user.

From the start one minor thing was missing during this work: planning. We did not plan this project well. I went to Cern and got instructions to make a working ARC system. I did not really understand what the Leszek wanted me to do with this ARC system. If I had known earlier the whole idea of having the ARC in lab I could have done better. I did what I went to do and that was building and studying the ARC electronics and ARCS program. Basically ARCS was not made for to be as DAQ only. The electronics can collect data fast, but the software was made for testing. I tried to find information how to read two hybrids at a time, without any results. When I got the electronics working I tried immediately to analyse data. At this point I should have found out what we could do with this system. This way we might have decided trying another approach.

In the future the ideal solution in this project would be a new LabView program. The program could be made from the ARCS by finding the parts that are communicating with the electronics. With this I mean making only a program that collects data from electronics and saves this data directly into file. Because if we have data in files its easy to use ROOT to analyse data. With this we could perhaps solve the two axes reading. Making two systems with a same trigger could be the solution as well and trying to get same time stamp to identify the event. The other things to solve are more permanent setup for electronics and finding out how to reduce noise.

Sources

[1] Cern, http://public.web.cern.ch/public/en/About/About-en.html, 2.12.2009

[2] Large Hadron Collider, http://fi.wikipedia.org/wiki/Large_Hadron_Collider, 2.12.2009

[3] Picture of the LHC from air, http://blogs.discovermagazine.com/cosmicvariance/files/uploads/LHC_arial.JPG, 2.12.2009

[4] RWTH AACHEN University, http://www.rwth-aachen.de/go/id/bdz/, 2.12.2009

[5] ARCS software, http://www.physik.rwth-aachen.de/institute/institut-iiib/forschung/cms/detektorentwicklung/arc/arcs/, 2.12.2009

[6] ARC board and FE adapter, http://www.physik.rwth-aachen.de/institute/institut-iiib/forschung/cms/detektorentwicklung/arc/arc-board/, 2.12.2009

[7] PCMIO manual, http://www.physik.rwth-aachen.de/fileadmin/user_upload/www_physik/Institute/Inst_3B/Forschung/CMS/Detektorentwicklung/ARC/PCMIO.pdf, 2.12.2009

[8] NIM definition, http://www.physics.mcgill.ca/~francois/projects/spark/nim2.html, 3.12.2009

[9] Application-specific integrated circuit, http://en.wikipedia.org/wiki/Application-specific_integrated_circuit, 3.12.2009

[10] APV25-S1 User Guide 2.2, http://doc.cern.ch//archive/electronic/other/generic/public/cer-002725643.pdf, 3.12.2009

[11] ARCS manual, http://www.physik.rwth-aachen.de/fileadmin/user_upload/www_physik/Institute/Inst_3B/Forschung/CMS/Detektorentwicklung/ARC/arcs.pdf, 3.12.2009

[12] ARC board manual, http://www.physik.rwth-aachen.de/fileadmin/user_upload/www_physik/Institute/Inst_3B/Forschung/CMS/Detektorentwicklung/ARC/arc.pdf, 3.12.2009

[13] Ultimate Boot CD, http://www.ubcd4win.com/, 2.12.2009

[14] Postscript Viewer, http://downloads.zdnet.com/abstract.aspx?docid=834275, 2.12.2009

[15] Jones, B. L. 2001. Sams Teach Yourself C# in 21 Days, USA: Sams publishing

[16] ROOT, http://root.cern.ch/drupal/content/about, 3.12.2009

[17] LGPL open source license, http://root.cern.ch/drupal/content/license, 3.12.2009

[18] Linux instructions, http://lowfatlinux.com/, 4.12.2009

[19] Manual of Trees in programming, ftp://root.cern.ch/root/doc/12Trees.pdf, 7.12.2009

[20] Standard deviation, http://en.wikipedia.org/wiki/Standard_deviation, 9.12.2009

Appendices

1. Linereader, code examples
2. Looping, code examples
3. Saving data to tree, full macro
4. Mean and sigma, full macro
5. Totalcharge, full macro
6. HowTakeRawDataWithARCS, instructions

```
public void Read() //This start reading process
        {
            Hit_60p =0;  // this is for hit sections values to normal
            Hit_80p = 0;
            readed_lines = 0; /// put to readed lines values to zero
            string sourceFile = null;
            OpenFileDialog dlg = new OpenFileDialog();
/// this start openfile dialog and we
/// can select file we want ot analyse
            dlg.Filter = "DAT Files (*.dat)|*.DAT";
            // this is file type filter and you cant
                                //select other types of file than in the filter
            if (dlg.ShowDialog() == DialogResult.OK)
           /// if system gives positive result
                        /// file exits we do the things inside the clause
            {
                sourceFile = dlg.FileName;//Create new fileinfo to open up a
file
                FileInfo theSourceFile = new FileInfo(sourceFile);
     StreamReader reader = theSourceFile.OpenText();// create a text    reader
for that file
                Results_Write("Reading file");
                lineParser(reader); /// lineParser reads and separates data
form texts and then readed line is put into array
            }
            else
            {
                Console.WriteLine("File not found: " + sourceFile);
                Results_Write("File not found: " + sourceFile);
            }
        }


public void lineParser(StreamReader reader) // method to do
        {
            string line;
            do
            {
line = reader.ReadLine(); //we specify that lines values is readed
           if (line == null) /// fail safe that there wont be errors in
nullreference if we try to refer value that dont exist
                {
                    break; /// if the reader read empty line the processs is
finished and all data from file is
                    /// rerad. So we jump out the braked at break command
                }
                if (line.Length > 150) /// if the lines lenght is bigger than
150 it means line is data line
                {  // so we use this branch of the code to do data handling
                    cleanLine_Data(line);
                }
                else /// all rest lines are text and some parts of the text we
collect to memory
                {
                    cleanLine(line);// here we take other data in array
                }
                readed_lines++;//how many line of text there were readed in
file
            } while (line != null);/until reader reads empty line this
continues
```

```
            k_event = 0;    // we null the data storing values that next time
the program is working
            j_channel = 0;            //  like usuall
           MessageBox.Show("Data reading was successful. "  +(readed_lines-32)
+ " events read");
            Results_Write("Data reading was successful. " + (readed_lines - 32)
+ " events read");
        }

public string cleanLine(string line)// this line if there is no data on line or
if the line is under 150 lenght
        {  //// if line has following word we take it in APV_settings array

if((line.Contains("File"))||(line.Contains("Date"))||(line.Contains("I2C.Addres
s"))
||(line.Contains("Mode"))||(line.Contains("Latency")||(line.Contains("IPRE"))||
(line.Contains("IPCASC"))||(line.Contains("IPSF"))||(line.Contains("ISHA"))||
(line.Contains("ISSF"))||(line.Contains("IPSP"))||(line.Contains("IMUXIN"))||
(line.Contains("ISPARE"))||(line.Contains("ICAL"))||(line.Contains("VFP"))||
(line.Contains("VFS"))||(line.Contains("VPSP"))||(line.Contains("CDRV"))||
(line.Contains("CSEL"))||(line.Contains("MUXGAIN"))||(line.Contains("Error"))))
                 {
                        APV_setting.Add(line);
                 }
            return line;
        }
//this is the where next phase data is put, taken from start of main program
private string[,] channel_data = new string[120000, 128];

public string cleanLine_Data(string line)//this put data in table
        {
          string[] modified = System.Text.RegularExpressions.Regex.Split(line,
"\\t+", RegexOptions.None); //first delimiter is tabulator so we separate
event, time and data
           time.Add(modified[1]); //put current time in array
           string data = modified[2]; ///data string to further examination
           char[] delim = { ',', ' ' };  /// makes new deliminators
           string [] strArr = data.Split(delim); // separate data string to
array

/// 12 place in string table strArr is header data we dont take this in this
table here we put first time, in table place 0,0 , event 1 / channel 1
valuebecause one is event consits 128 channel values. so we have to write all
channels values to this row and when we have done this we change the row for
the next part is hit calculation and is all data from test. horisontal plane is
channel and vertical is events

// every time program writes every channels data to table
                for( j_channel = 0; j_channel< 128; j_channel++)
                {
// this +12 is shift, which purpose is to skip header data and we begin wrote
daata at table position 13,number is 12
                    channel_data[k_event, j_channel] = strArr[j_channel + 12];
                }
// forthe next time we tell the program to change row where the data will be
put
                k_event++;
                return line;

}
```

```
public void Noise()
        {
       //    Noise part we take every event data and calculate noise in the
channel. Two ways first calculate channels mean. Then we calculate difference
value with every channel --> Noise. SEcond way is to calculate every channels
mean and subract this valu in every channel.

       double [ ] avg_noise_in_channel = new double [128];
       double[,] channel_SD_noise = new double[readed_lines - 32,128];
       double[,] channel_CM_subracted_noise = new double[readed_lines -
32,128];


            double common_mode_sum = 0;
            for (k_event = 0; k_event < readed_lines - 32; k_event++)
            {
                for (j_channel = 0; j_channel < 128; j_channel++)
                {
           common_mode_sum = common_mode_sum +
           double.Parse(channel_data[k_event, j_channel].ToString());
                }
                common_mode_in_event_n[k_event] =  (common_mode_sum / 128);
                common_mode_sum = 0;
            }

//calculate common mode average

            common_mode_avg = 0;
            common_mode_sum =0;
            for(k_event =0; k_event < readed_lines -32; k_event++)
            {
                      common_mode_sum = common_mode_sum +
                      double.Parse(common_mode_in_event_n[k_event].ToString());
            }
            common_mode_avg = (common_mode_sum / (readed_lines -32));

// calculate common mode sigma

            double common_mode_divided =0;
            common_mode_sigma = 0;
            common_mode_sum  = 0;
            for (k_event = 0; k_event < readed_lines - 32; k_event++)
            {
                common_mode_sum  = common_mode_sum  +
Math.Pow((double.Parse(common_mode_in_event_n[k_event].ToString()) -
common_mode_avg), 2);
            }
            common_mode_divided = common_mode_sum / (readed_lines - 32);
            common_mode_sigma = Math.Sqrt(common_mode_divided);

///These are examples how looping we can do big calculations even when there is
100000 numerical values. This also shows how the calculations where made is
this C# code.
```

Appendix 3: Saving data to tree

```
{
  gROOT->Reset();                // reset values
  gROOT->SetStyle("Plain");      // set values
  ifstream infile;               // makes a reader that reads lines in file
  TFile rootfile("run1402x.root","recreate"); /// opens/makes a root file
  infile.open("run1402x.dat"); //// opens the file which is read

  short apvADC[128];             /// array where all events data is saved
  char comma;                    //separator
  short dummy;                   // “trash bin”

  TTree t("t","t"); // makes a tree where data is saved
  t.Branch("APV",apvADC, "apvADC[128]/s"); // each event is a brach in a tree
  int j=0;            // event counter
                                 // one branch has 128 channels value in event

  while(infile.good() )     //// loop continues until the file is ok to read
    {
      for(int i=0; i<12;i++)   //// goes first 12 data number in line
           {
           if(!infile.good()) break; //keep sure that readed file isnt
corrupted
           infile >> dummy >> comma; //puts read value to dummy.
           }
      for(int i=0; i<128; i++)   // after we are skip 12 lines, starts the code
           {
           // write data to file
           if(!infile.good()) break;
           infile >> apvADC[i] >> comma; // readed value is put in
           }
      infile >> dummy; // last end mark to dummy
      if (!infile.good()) break;
      t.Fill(); /// fill the tree with event data
      j++; // number of events
    }
  cout << "Number of events " << j << endl;/// this prints number of events
  t.Print();//// we print the values of the tree in screen
  t.Write();  //// we write the tree into root file
  rootfile.Close();/// after writing the data we close the root file
}
```

```
{
  gROOT.Reset();
  gStyle->SetOptFit(1111); // fit options
  gStyle->SetOptStat(1111111);// values in plots right up corner

  TFile file("run1401x.root");/// data file what is processing/needs to change
for every file
  TTree *tdata = (TTree *)file.Get("t;1"); // we made a tree and put the in
  //APV_an.C made trees to *tdata three. This is pointer three that makes it
pointing to the this three file
  TFile pedfile("run1400xped.root"); /// pedestal data file and put it into the
*tped three.
  TTree *tped = (TTree *)pedfile.Get("t;1"); // points data in file

  ofstream pedestals; /// writer that write something to file
  pedestals.open("_pedestals.tmp"); //makes/overwrites pedestal file
  if(!pedestals.is_open()) // something to say if file doesnt open
    cout << "\n\n  ERROR: cannot open the file _pedestals.tmp"
         << " (check if you have writing rights!!!).\n"
         << endl;

  double mean_all[128]; // array for means
  double sigma_all[128];// array for sigma values of noise
  double mean_sum=0; // needed for calculation

  TF1 gaus("gaus","gaus"); // gauss
  gaus.SetRange(0,250);                    // where value is
  gaus.SetLineColor(2);                    // colour of the fit

  TGraph all_mean;                         // a graphical object
  TGraph all_sigma;                        // same
  char fileaction[80];
// histogram for purpose to get mean and sigma values in channels
  TH1F hped("hped", "Pedestal distribution", 250, 0, 250);
  for (int i=0; i<128; i++)
    {
      sprintf(fileaction,"apvADC[%d]>>hped",i); //gets data from pedestal run
file
      tped.Draw(fileaction);
      hped.Fit("gaus","R");// gauss fit in channels data
      hped.Draw();    // draw this fit in all shows last channel data
      double mean = gaus.GetParameter(1); // gets channel 1,2,3,... mean
      double sigma = gaus.GetParameter(2);// 2 means sigma value in channel
      cout << "Mean" << "\t"<< i << "\t"<< mean << endl; //prints value
      cout << "Sigma" << "\t"<< i << "\t"<< sigma << endl;
      mean_all[i]=mean; // put value in array
      sigma_all[i]=sigma;
      mean_sum = mean_sum + mean; // some check
      all_mean.SetPoint(i,i,mean); // makes points for plot
      all_sigma.SetPoint(i,i,sigma);
      if (pedestals.good())
                      pedestals << mean << "\t" << sigma << "\n"; // write
values to file if file is good
      sigma = 0;// reset value
      mean  = 0;
  }
   cout << "Mean sum" << "\t"<< i << "\t"<< mean_sum << endl; /// print in
konsole
   if (!pedestals.good()) // if file isnt good this is printed
     cout << "\n\n  ERROR: Problems when writing the file _pedestals_.tmp\n"
          << endl;
   pedestals.close(); // closes file after the data is writen to it

  TCanvas mean_sigma; // screen where the plots can be draw
  mean_sigma->Divide(2,1); // canvas is cut in two section, two in one line
  mean_sigma->cd(1);  // take section 1 in use
  all_mean.Draw("apl");          // and draw mean values to it
  mean_sigma->cd(2); // section 2 in use
  all_sigma.Draw("apl");// draw sigma values in channel

  // put data to memory ready for next step in analysis
  char drawingstr[80];
  char apvstr[10];
  for(int i=0; i<128;i++)
    {
      sprintf(drawingstr,"- apvADC[%d] + %f",i,mean_all[i]);
      sprintf(apvstr,"apv%d",i);
      tdata->SetAlias(apvstr,drawingstr);
    }
}
```

```
//tdata->Process("../Makro/totalcharge.C+","5") used in ROOT to call this
you have write folder where this macro when using this
this is the using libraries or functions. This defines in program what
we are using in this code

//// SOME part of the code is formatted but key parts are same
////This is a one structure in root. When using this structure first
//// you have begin clause that do start things like making array or define
some values,then you have process clause in this clause when analysing one
entry in tree this  is used and all the things inside this parantesis is done.
Third is terminate parantesis this is done when all entries in tree are went
through and this is like presenting the results so when all data is analysed
this clause will make the histograms  and other plots. This code basically
reads one entry starting from event 1 to +10000 or more and makes the
calculation written in process clause.

#define totalcharge_cxx
#include "totalcharge.h"
#include <iostream> /// line readers
#include <TH2.h> /// histograms
#include <TStyle.h>
#include <fstream> ///
#include <TCanvas.h>  /// canvas where plots are drawn

TH1F* htotchg; /// make a histogram for total charge
TH1F* treshold;//// treshold or cluster size
TCanvas* t2;   /// canvas for treshold
TCanvas* ccharge; /// canvas for totalcharge

void totalcharge::Begin(TTree * /*tree*/)  /// this is done in begining, only
once
{
  TString option = GetOption(); // getoption number of sigmas of noise default
3, number x in process line
  sigma_mul=option.Atof();// gives inputted value to sigma multiplier

  std::cout << "I begin to analyse events with options: " // print this and ask
the sigmamultiplier
                << sigma_mul
                << std::endl;

  gDirectory -> Delete("treshold"); // delete existing histogram from folder
  gDirectory -> Delete("htotchg");//same
  htotchg = new TH1F("htotchg","Total charge",500,0.01, 1000.01);//makes a new
histogram
  treshold = new TH1F("treshold", "Treshold", 130,0, 130);//same

  t2 = new TCanvas("t2","t2");//makes a canvas for the histograms
  ccharge = new TCanvas("ccharge","ccharge");//same

   ifstream pedfile; // makes file reader
   pedfile.open("_pedestals.tmp"); //open pedestal file what was done in
earlier ARCanalysis.C macro
   int i;
   for(i=0; i<128; i++){    // read 128 lines first value is ch_ped next
sigma_ped in line
       pedfile >> ch_ped[i]; // first read value in line is channels
pedestal/mean value
       pedfile >> sigma_ped[i]; // second in line is the standard deviation of
the pedestal data
       if(!pedfile.good() ) break; /// if some errors we go out the loop
```

```
     }
   if(i != 128) std::cout << "ERROR while reading pedestals" << std::endl;
   // if the selected _pedestals.tmp is wrong or error occurs this message is
displayed in konsole
}
void totalcharge::SlaveBegin(TTree * /*tree*/) // get the option
{
   TString option = GetOption();
}
Bool_t totalcharge::Process(Long64_t entry) // this is done in every entry of
tree
{
  double sum_allchannels = 0;/// reset the total sum of channel from previous
event
  GetEntry(entry); //this gets one entry from tree, one event data is get from
all channels
  int k=0;
  for(int i=0; i<128 ; i++) /// in one entry has 128 values
    {
      if(APV[i] > (ch_ped[i] + sigma_mul*sigma_ped[i
//data whivh is analysed inst inverted we need to use normal way to analyse
data,
//but if we are using different modes during data this need to changed
           {
             sum_allchannels +=   APV[i] - ch_ped[i];
             /// "a += b" means "a = a + b" also we sum the charge
             //over the threshold to totalcharge in this event
             k=k+1; // counter that counts how many times the
             //treshold is go over basically primitive clustering algorithm
           }
    else// if the value is lower or egual we increase the totalcharge by zero
           {sum_allchannels += 0;} /if there is nothing to sum we sum the sum
by 0
 }
  treshold->Fill(k); // this events clustersize is put into the histogram
  if(sum_allchannels > 0.1) htotchg->Fill(sum_allchannels);
  /// if the sum is over 0.1 we fill the totalcharge histogram
  sum_allchannels=0; /// we reset the totalcharge after calculating it in event
   return kTRUE;
}
void totalcharge::Terminate() /// this block is prosessed after the "process"
pragets is run.
//After all events are plotted to the histogram we run this part of the code.
{
  ccharge->cd(); //open the made canvas to use
  htotchg->Draw();// draw the totalcharge histogram to the opened canvas
  TCanvas t2;
  t2->cd();//treshold or cluster histogram is drawn to this canvas
  treshold->Draw();
  std::cout << "Analysis done" << std::endl;//std is needed to able to print to
konsole
}
```

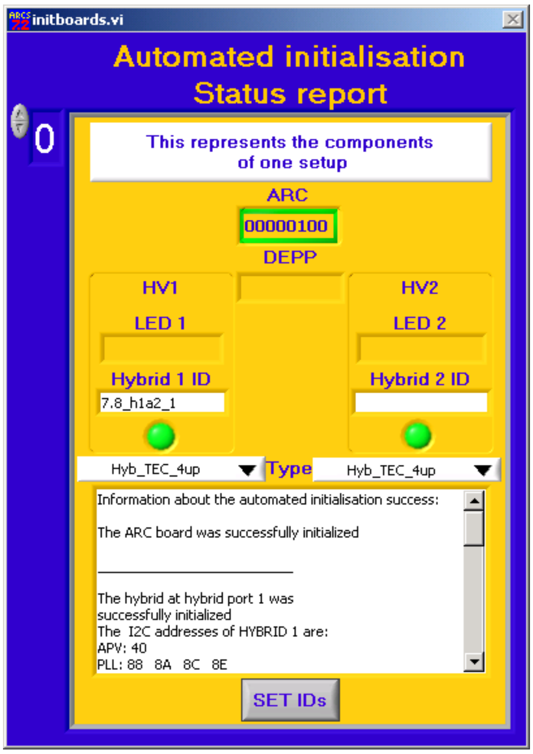
How to take raw data with ARCS

Next is instruction how to take raw data with ARCS program.

1. First you should check the connections and ARC board power. Bad connection or power failure prevent program working.

2. Next if the hardware is all right you can start the program. If you are given the initboards VI (virtual instrument) there were no problems during initialization. If not possible reasons for errors are that you had not installed GiveIO, ARC board power is not ON or wrong connection in hardware. Wrong connection can happen easily in totem_apv_adapter_test_card and hybrid connection. Now the test card end is marked with black spot to indicate the correct connection. If you hadn't installed or make the GiveIO service start up program gives you message that program cant make connection to GiveIO service. This can be solved by going manufactures pages [1] and installing the GiveIO or making it START UP behaviour automatic.

3. In initboards VI you give two hybrids, which have one APV connected, a name. This name is this session data storage. In programs folder there is a config file that defines where the data is put. In this case it is at the moment *e:/arcsdata.* After taking data you can find the data of this session in this folder with name "Raw_Data_HybridID". So user gives IDs to hybrids that are connected to ARC board and you should select correct TYPE in this VI but should not bother there isn't right alternative in here so leave the TYPE that is default. This TYPE is description of the setting. So how many APVs or do we have a detector connected to system. But every option has four or six APVs so no bother. This has I think only effect on *FAST TEST* in program. *SET IDs* will close this VI and start the ARCS.



ARC: 00000100

Hybrid 1 ID: 7.8_h1a2_1

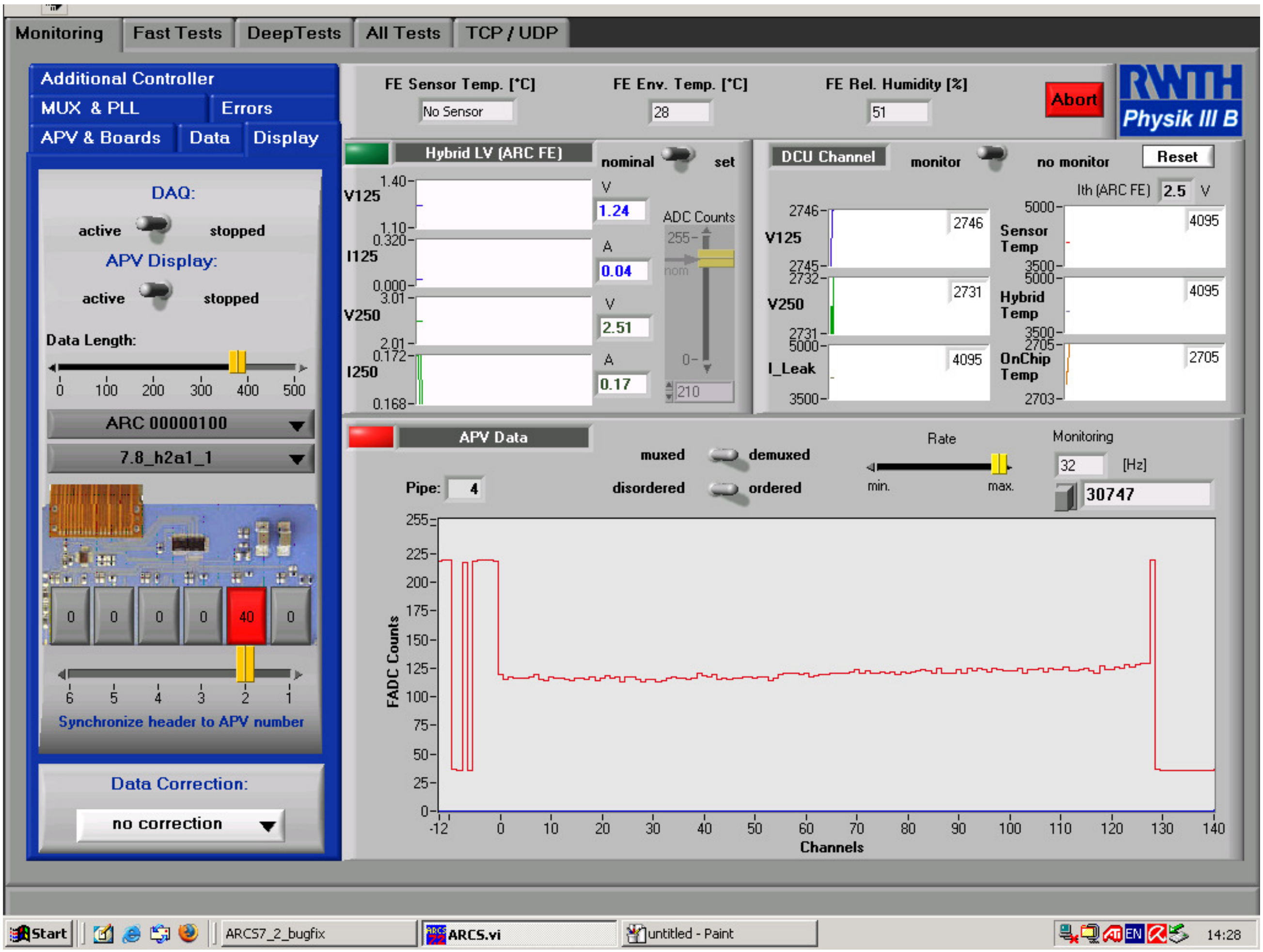
Hybrid 1 type: Hyb_TEC_4up

Hybrid 2 type: Hyb_TEC_4up

```
Information about the automated initialisation success:

The ARC board was successfully initialized

______________________________

The hybrid at hybrid port 1 was
successfully initialized
The I2C addresses of HYBRID 1 are:
APV: 40
PLL: 88  8A  8C  8E
```

Figure1. initboards virtual instrument

4. In ARCS user goes to *Monitoring* and *Display – tap.* This tap shows all hybrids connected to ARC board. Here user selects the board where to take data from. In figure2 the hybrids name is “7.8_h2a1_1” and clicking the arrow user can alter the read hybrid to another.



FE Sensor Temp. [°C]: No Sensor

FE Env. Temp. [°C]: 28

FE Rel. Humidity [%]: 51

DAQ: active / stopped

APV Display: active / stopped

Data Length: 0 100 200 300 400 500

ARC 00000100

7.8_h2a1_1

0 0 0 0 40 0

6 5 4 3 2 1

Data Correction: no correction

Hybrid LV (ARC FE): nominal / set

V125: V 1.24

I125: A 0.04

V250: V 2.51

I250: A 0.17

ADC Counts 255 nom 0 210

DCU Channel: monitor / no monitor

Ith (ARC FE) 2.5 V

V125: 2746

V250: 2731

I_Leak: 4095

Sensor Temp: 4095

Hybrid Temp: 4095

OnChip Temp: 2705

muxed / demuxed

disordered / ordered

Pipe: 4

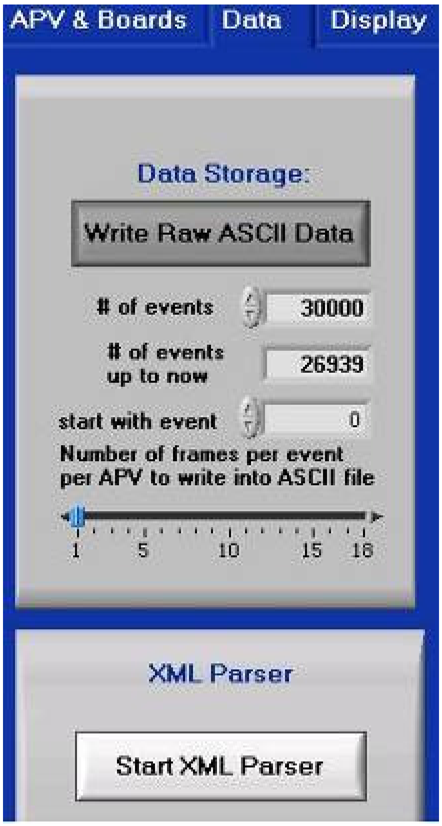
Rate: min. max.

Monitoring: 32 [Hz]

30747

Figure2. Monitoring and display – tap

5. After selecting the hybrid user goes to *Monitoring* and *Data – tap*. In this tap user can select the number of taken events, event where the DAQ starts and how many frames are taken per event. This number of frames should be used as one until user understands how this affects the DAQ. *# of events up to now* is counter that so how many events we have now recorded. DAQ starts when *Write Raw ASCII Data* button is pressed and pressing second time stops the DAQ even if the all events aren't required. DAQ stops when all events is recorded.



# of events 30000

# of events up to now 26939

start with event 0

Number of frames per event per APV to write into ASCII file

Figure3. Data – tap

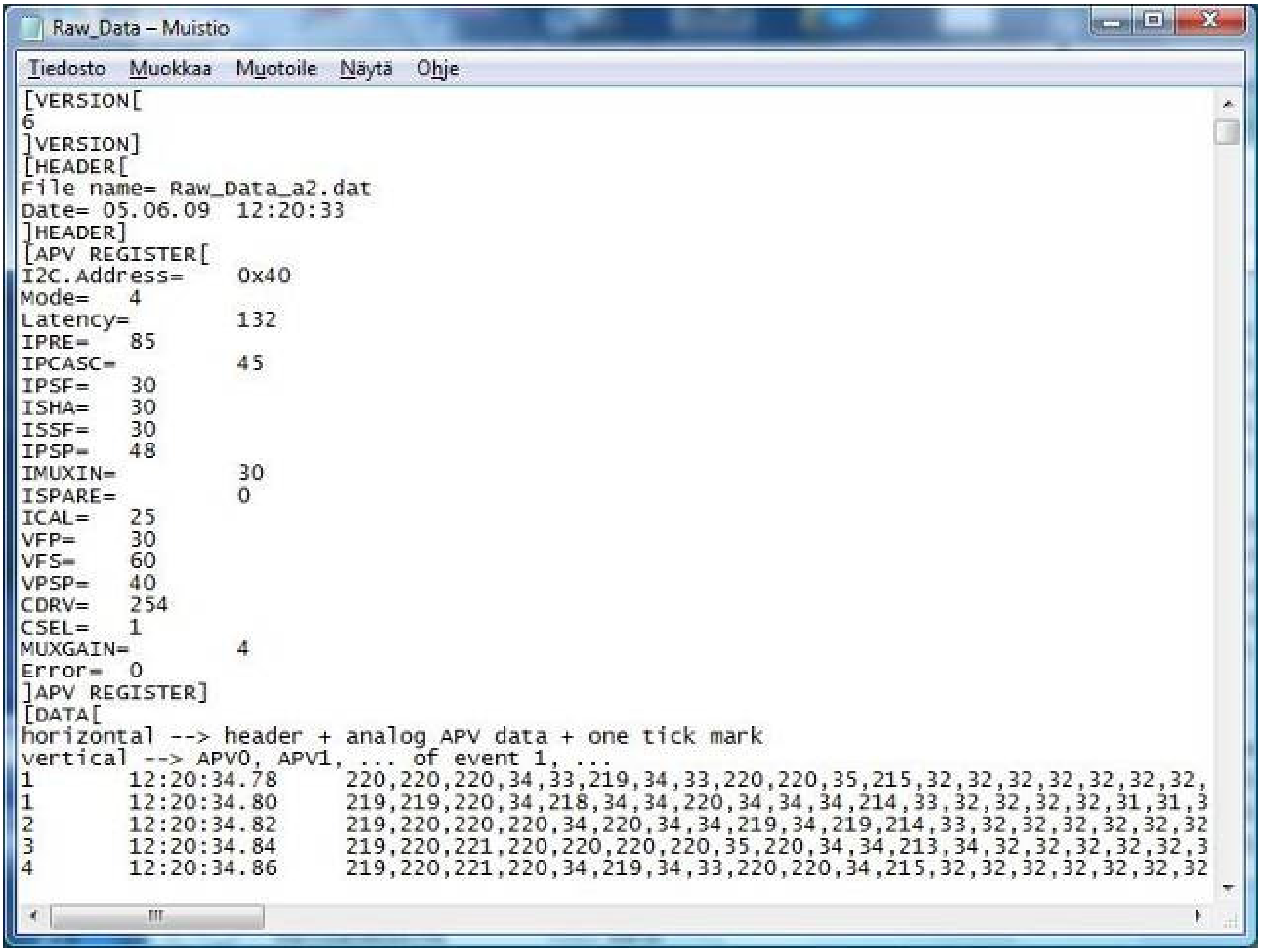
6. After data is taken from hybrid user closes the program and opens the folder where the data was saved. This case in folder *e:/arcsdata.* This session data is here with name *Raw_Data_HybridID.* This data is now in ascii – file. To use this data this file has to process properly. Data is saved in ascii – file. At beginning this file is saved the current settings of the APV. Data read from strips is non-consecutive because of the multiplexer. Next paragraph is taken from APV25-S1 User Guide 2.2 [2].

   Channel Order
   Due to the tree structure of the analogue multiplexer, the order that channels are read out through the analogue output is non-consecutive. The multiplexer is constructed in three stages, if 'n' is the order in which the channels appear (starting at 0,1,2,3,4 etc), then the physical channel number is defined by: [2]

   Channel No. = 32 * (n MOD 4) + 8 * INT(n / 4) - 31 * INT(n / 16)

   This leads to think that current program might not be correct but about this I am not sure. This is because you can order the data in settings as ordered or disordered. And default TYPE setting *Hyb_TEC_4up* has ordered as selected so I am not sure if this effect also data that is saved on ascii – file. But its true there where strange behaviour in analysed data that has three peaks in channel data. This is small fine tuning that didn't affect in trying to make pulse height spectrum.

   More about this ascii – file. After APV settings data is saved APV by APV. This means every row is APV and its data. Because in setting have only one APV that we can read at time this file has only one APVs data. Data row has first the event number, time and then data from APV. First three numbers are logical ones, eight next numbers are address, and following 128 are channel data ordered/disordered and at the end of this is tick mark which tells that this APVs data has ended.



```
[VERSION[
6
]VERSION]
[HEADER[
File name= Raw_Data_a2.dat
Date= 05.06.09  12:20:33
]HEADER]
[APV REGISTER[
I2C.Address=    0x40
Mode=   4
Latency=        132
IPRE=   85
IPCASC=         45
IPSF=   30
ISHA=   30
ISSF=   30
IPSP=   48
IMUXIN=         30
ISPARE=         0
ICAL=   25
VFP=    30
VFS=    60
VPSP=   40
CDRV=   254
CSEL=   1
MUXGAIN=        4
Error=  0
]APV REGISTER]
[DATA[
horizontal --> header + analog APV data + one tick mark
vertical --> APV0, APV1, ... of event 1, ...
1       12:20:34.78     220,220,220,34,33,219,34,33,220,220,35,215,32,32,32,32,32,32,32,
1       12:20:34.80     219,219,220,34,218,34,34,220,34,34,34,214,33,32,32,32,32,31,31,3
2       12:20:34.82     219,220,220,220,34,220,34,34,219,34,219,214,33,32,32,32,32,32,32
3       12:20:34.84     219,220,221,220,220,220,220,35,220,34,34,213,34,32,32,32,32,32,3
4       12:20:34.86     219,220,221,220,34,219,34,33,220,220,34,215,32,32,32,32,32,32,32
```

Figure4. Ascii – file

Links

1. ARC system (14.10.09), http://www.physik.rwth-aachen.de/institute/institut-iiib/forschung/cms/detektorentwicklung/arc/
2. APV25-S1 User Guide 2.2 (14.10.09), http://doc.cern.ch//archive/electronic/other/generic/public/cer-002725643.pdf