Západočeská Univerzita v Plzni

Fakulta Elektrotechnická

Katedra Aplikované Elektroniky a Telekomunikací

DOCTORAL THESIS

Antineutrino detector processing system

Candidate: **Ing. Jakub Vlášek**

Supervisor: **Doc. Dr. Ing. Vjačeslav Georgiev**

Pilsen, March 2018

Key words

Inverse beta-decay, DANSS, anti-neutrino, VME, Scintillator, PMT, V1495, S-cube, coincidence

Abstract

This thesis presents a method of data analysis of a multichannel spectroscopic data acquisition system for a large segmented detector of reactor antineutrinos. The method is described, and two implementations are presented. One for PMT based spectrometer for the DANSS detector, built by the Joint Institute for Nuclear Research in Dubna and operating underneath nuclear reactor in the Kalinin Nuclear Power plant and second software based for the S^3 detector, being developed by the Institute of Experimental and Applied Physics of the Czech Technical University in Prague.

Klíčová slova

Inverzní beta rozpad, DANSS, anti-neutrino, VME, Scintillator, PMT, V1495, S-cube, koincidence

Anotace

Tato práce prezentuje metodu zpracování dat multikanálového systému pro získávání dat z segmentovaného detektoru reaktorových antineutrin. Metoda je popsána a její dvě různé implementace. Jedna pro spektrometr založený na fotonásobičích detektoru DANSS, postavený Spojeným ústavem jaderných výzkumů v Dubně a běží v jaderné Kalininské elektrárně, a druhý, softwarově založený pro detektor S^3, který je vyvíjen v Ústavu Technické a Experimentální Fyziky ČVUT v Praze.

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou v rámci doktorského studia na Katedře aplikované elektroniky a telekomunikací Fakulty elektrotechnické Západočeské univerzity v Plzni. Prohlašuji, že jsem tuto práci vytvořil samostatně, s použitím literatury a zdrojů uvedených v seznamu, který je její neoddělitelnou součástí a za pomoci legálních kopií řádně registrovaného, nebo volně šiřitelného softwarového vybavení. V práci nejsou uvedeny žádné citlivé, či utajované skutečnosti podléhající obchodnímu tajemství, nebo vyžadující speciální režim přístupu. Jakékoli využití a uplatnění uvedených postupů a metod je nicméně možné pouze na základě autorské smlouvy a souhlasu Fakulty elektrotechnické Západočeské univerzity v Plzni.

V Plzni dne 31.8.2018

Ing. Jakub Vlášek

…………………....

Acknowledgement

I would like to thank my colleagues who helped during work on my thesis. Especially those from the Dzhelepov Laboratory of Nuclear Problems of the Joint Institute of Nuclear Research in Dubna, Russia, the Institute of Experimental and Applied Physics of the Czech Technical University in Prague and Faculty of Applied Electronics and Telecommunications of the University of West Bohemia. I would like to thank namely following people namely: Zdenek Hons, Vjaceslav Georgievich Egorov, Igor Zhitnikov.

I would also like to thank my thesis supervisor Vjačeslav Georgiev and the supervisor-specialist Ivan Štekl.

This research has been funded by the following grants: Czech Technological Agency TE 01020445, Czech Ministry of Education Youth and Sports INGO II – LG14004, Russian State Atomic Energy Corporation ROSATOM 16.1006 and European Regional Development Fund Project No CZ.02.1.01/0.0/0.0/16_019/0000766.

Contents

List of abbreviations

FPGA	Field Programmable Gate Array
MSPS	Mega Samples Per Second
JINR	Joint Institute of Nuclear Research (Dubna, Russia)
NPP	Nuclear Power Plant
KNPP	Kalinin Nuclear Power Plant
ADC	Analog to Digital Converter
PMT	Photomultiplier Tube
SiMP	Silicon Photomultiplier
AFE	Analog FrontEnd
DAQ	Data Acquisition System
DANSS	Detector of reactor AntiNeutrinos based on Solid Scintillator
VME	Versa Module Europa bus
QDC	charge(Q) to Digital Convertor
IBD	Inverse Beta Decay
WLS	Wavelength shifting
DPP	Digital Pulse Processing

1 Introduction

Neutrinos are a hot topic of contemporary particle physics and their study is a subject of a great interest. Initially, it was assumed that neutrinos are mass-less, however, a discrepancy between the theoretically predicted solar neutrino flux with ground measurements (the "Solar Neutrino Problem" [1]) led to the discovery that neutrinos do have mass and can oscillate between their flavors (electron, tau, muon).

There are three major sources of neutrinos – sun, particle accelerators and nuclear reactors.

The high intensity of neutrino flux generated by nuclear reactors provides a good opportunity to study the elementary behavior of neutrinos. There are experiments, including the RENO (Reactor Experiment for Neutrino Oscillations) [2], which attempt to measure neutrino oscillations with detectors spaced hundreds meters from the reactor. But the amplitude of the neutrino oscillations is a question. Therefore, it is necessary to place the detector as close to the reactor as possible. Unfortunately, existing detectors are mostly based on hydrogen-rich liquids, such as kerosene, which pose a flammable hazard, an unacceptable for a reactor building.

One of the possibilities to avoid the fire hazard and gain access close to the reactor from the power plant operators is to use polystyrene scintillating detectors. This has been used by the DANSS (Detector of reactor AntiNeutrinos based on Solid Scintillator) detector [3], operated by the Joint Institute of Nuclear Research in Dubna and the Institute of Theoretical and Experimental Physics in Moscow. This 1m^3 detector, made from 2500 scintillating polystyrene 1×4×100 cm^3 bars coextruded with Gadolinium layer has been running for two years in a close proximity to the core of the Kalinin NPP. It is mounted in a z-axis moveable platform (8-12m) to explore the short-range neutrino oscillations.

Together with Dzhelepov Laboratory of Nuclear Problems (DLNP) of JINR, the Institute of Experimental and Applied Physics of Czech Technical University Prague are currently constructing a new anti-neutrino detector –S^3 as an evolution of the DANSS design.

S^3 is a smaller detector with a volume of 0.4×0.4×0.4 m^3 consisting of 80 scintillating plates with improved light yield. At present, there are two detectors prototypes under construction. One detector will be placed next to the DANSS experiment in KNPP to provide reference information about the nuclear reactor power level. The second is expected to be placed in the Temelín NPP. Both nuclear power plants are of the same VVER-1000 technical specification.

The DANSS Data acquisition system is based on hardware triggered CAEN charge convertors (QDCs) and is controlled by logic implemented in an Field Programmable Gate Array. The S^3 DAQ is based on self-triggered CAEN multichannel ADC. The data reduction is implemented in software on a PC.

The aim of this thesis is to:

- prepare and design the detection algorithm of the DANSS detector
- design and implement the algorithm as a FPGA firmware for DANSS
- design and implement active cosmic veto FPGA firmware for DANSS
- design and implement the algorithm for the S^3 detector

The method must detect the inverse beta decay process $\tilde{\upsilon} + p \ \rightarrow e^{+} + n$, where antineutrino $\tilde{\upsilon}$ reacts with a proton p and produces positron e^{+} and neutron n.

This thesis is divided into the introduction, eight chapters and the conclusions. The second chapter briefly describes the way neutrinos are detected. Chapters 3 and 4 describe the DANSS and S^3 experiments.

The author's involvement begins with chapter 5, where the DANSS data acquisition system is described in detail. Chapter 6 describes the FPGA firmware newly developed for the DANSS DAQ.

Chapter 7 describes the C++ software implementation of an application capable of online experiment monitoring, visualization, analysis and the inverse beta decays signature detection.

Chapters 8 provides results of technical measurements obtained during the commissioning of the DANSS detector in the KNPP.

2 Detection of neutrinos

Neutrinos are particles which have very small probability of interaction (cross-section) with ordinary matter. Therefore, the detectors have to be either very large (kilometer scale) , as is the case for detection of cosmic neutrinos or very close to a source of neutrinos, e.g. a nuclear reactor. There are two methods of their detection, using Cherenkov radiation and using inverse beta decay detection.

2.1 History

A hypothetical particle, electrically neutral and having mass smaller than that of the proton was first proposed by Wolfgang Pauli in 1930 [4]. In 1933 Enrico Fermi named it a “neutrino” to distinguish it from the newly discovered neutron [5].

At first, neutrinos were considered massless. Their first detection was done by F.Reines and C.Cowan [6] using nuclear reactor as a strong source of antineutrinos using inverse beta decay. They used a liquid scintillator where the electron antineutrino interacts with a proton, emitting a positron and a neutron. The positron annihilates creating two 511 kEv gamma photons and the neutron is captured by a proton creating a deuteron and emitting gamma rays with energy of 2.2 MeV (Figure 2-1).

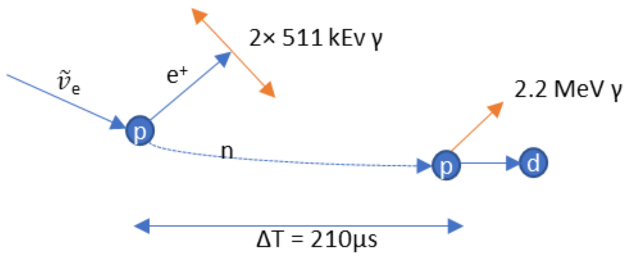


Figure 2-1 Inverse beta decay in a liquid scintillator

Several experiments with solar (R. Davis et al [7]), atmospheric, accelerator and reactor neutrinos have provided the existence of neutrino oscillations driven by non-zero masses and neutrino mixing. Neutrino oscillation is a quantum mechanical phenomenon whereby a neutrino created with a specific lepton flavor (electron, muon, tau) can later be measured to have a different flavor. The probability of measuring a particular flavor for a neutrino varies between 32 known states as it propagates through space. The neutrino oscillations were first predicted by Bruno Pontecorvo in 1957 [8], [9], [10].

Since neutrinos are massless in standard model of electroweak interactions, this verification of neutrino oscillations (and therefore mass) was first strong evidence for physics beyond the standard model. This convincing evidence for neutrino oscillations has been confirmed by other experiments (SNO [11], RENO [2], Daya Bay [12]). Neutrino oscillations experiments are not sensitive to the nature of the neutrino mass and give no information on the absolute neutrino mass scale [8].

Neutrino oscillation is a function of the ratio L/E where L is the distance travelled and E is the neutrino energy. To measure neutrino oscillation for different distances from the source is an important task. Also, several experiments announced the reactor anomaly [13], which can be explained by existence of sterile neutrinos. These questions can be solved by detector of neutrinos located as close as possible to the nuclear reactor [5].

2.2 Cherenkov detection

The interaction of a high energy cosmic neutrino in the detector volume produces electrically charged particle which travels faster than the local speed of light while emitting Cherenkov radiation. The photons are then detected by sensitive light detectors - photomultipliers. The particle track can be then reconstructed from the times of arrival of the Cherenkov photons.

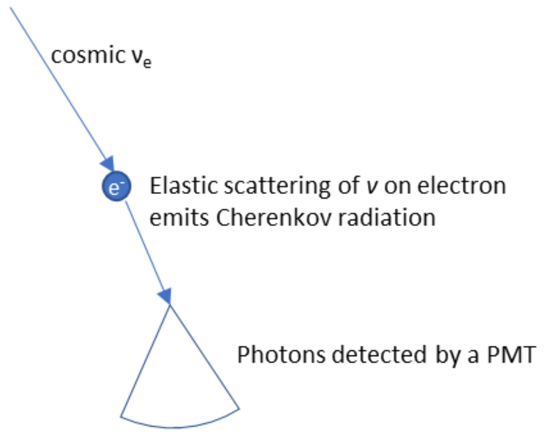

cosmic ν_e
e^-
Elastic scattering of ν on electron emits Cherenkov radiation
Photons detected by a PMT

Figure 2-2 Principle of high energy neutrino detection

Currently, there are several such detectors either operating or being built in the world.

- Super-Kamiokande – operational, 40 m steel tank with ultrapure water [14]
- Ice-Cube – operational, South Pole observatory using about cubic kilometer of Arctic ice as detector volume [15]
- Baikal Gigaton Volume Detector – under construction, 1 km^3 of clean Russian lake Baikal water [16], [17]
- ANTARES/KM3NeT – operational, 1km^3 of Mediterranean sea [18]

2.3 Inverse beta decay detection

The IBD detection method is used mostly in detectors of reactor neutrinos. The antineutrino interacts with a proton producing a positron and a neutron. The positron immediately annihilates with emission of two gamma rays of 511 keV each. The neutron is thermalized by collisions with light nuclei and captured. Neutron capture by Gadolinium emits several gamma rays with the total energy of 8 MeV. The IBD method looks for the characteristic signature of annihilation and following spatially close gamma rays of the neutron capture.

Experiments based on IBD include

- Daya-Bay [12] – Eight 20 ton kerosene based antineutrino detectors in three locations within 2 km of six nuclear reactors in Daya Bay, China
- DANSS [3]– 1 cubic meter polystyrene based detector about 10 m from the nuclear reactor in the Kalinin nuclear power plant, Russia.

3 DANSS Detector

The detector is constructed from 2500 scintillating strips with dimensions of 4×1×100 cm. Each strip is coated by a reflexive layer containing Gadolinium (Figure 3-2) and contains wavelength shifting fibers for light collection.

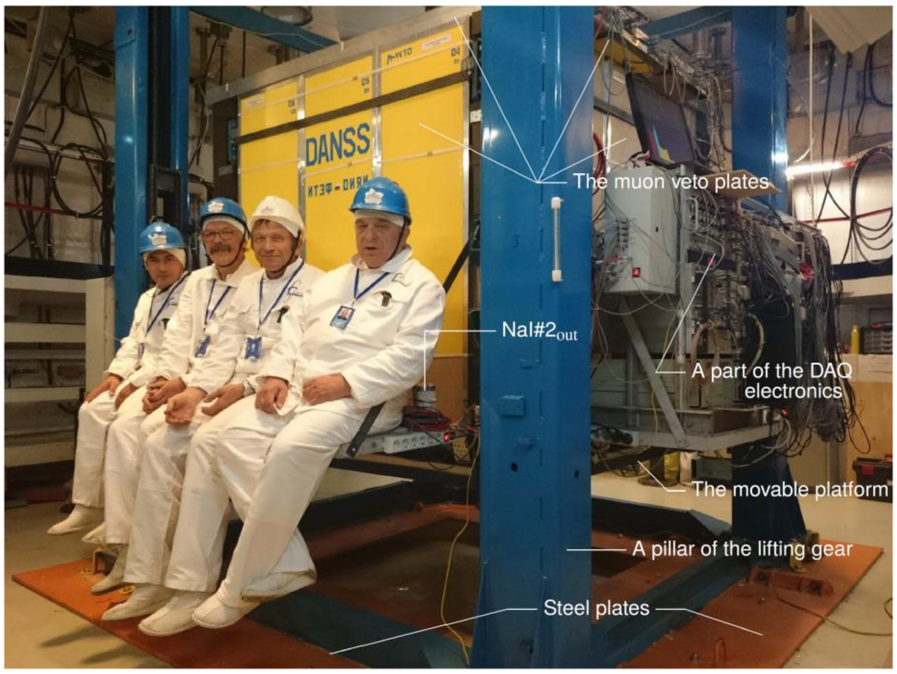
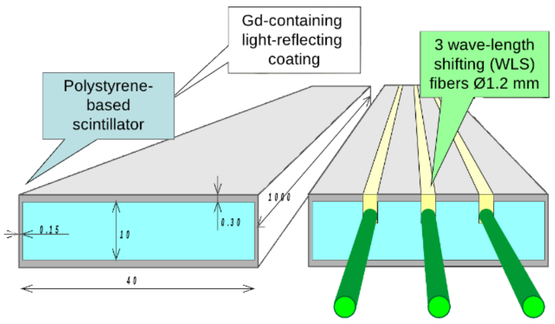


Figure 3-1 DANSS Team members with the detector in the KNPP technical room [3].

Each successive layer of the detector is built perpendicularly to the previous one. Ten parallel layers of 5 neighboring strips form one detector section (module - Figure 3-3). The X and Y sections are intercrossing so that the positional information of the interaction can be extracted. Every detector section is connected to one PMT.

The light is collected through wavelength shifting fibers by PMT and MPPCs. Each bundle of 100 WLS fibers from 50 strips is connected to one PMT. The average yield from single strip of is about 35 photoelectrons per 1 MeV [3].


Gd-containing light-reflecting coating
Polystyrene-based scintillator
3 wave-length shifting (WLS) fibers Ø1.2 mm
1000
0.30
0.15
10
40
Figure 3-2 Scintillating strip

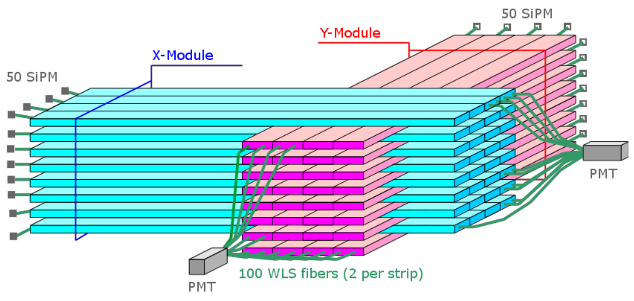


Figure 3-3 Detector internal layout

Every PMT is connected to an analog frontend (AFE) containing the high voltage power supply, pulse shaper, analog comparator and controlling MCU with a DAC. All AFEs are controlled from the acquisition PC via RS-485 bus. The output pulse from the pulse shaper is about 100 ns long and is accompanied with above-set-threshold digital pulse.

The MPPCs are connected to a separate data acquisition system which is not a subject of this thesis.

The DANSS detector uses the inverse beta decay as its detection principle (Figure 3-4). The electron antineutrino coming from the reactor core interacts with a proton inside the scintillator and produces positron and a neutron. The positron annihilates and creates a characteristic pair of 511 keV gamma rays – "*prompt signal*". After 2 to 20 µs the neutron moderates and is captured by Gadolinium. The resulting gamma rays have a total energy of 8 MeV and should be detected within a sphere of about 20 cm from the original neutrino interaction – "*delayed signal*". This method of detection was verified by small scale DANSS detector demonstrator DANSSino [19].

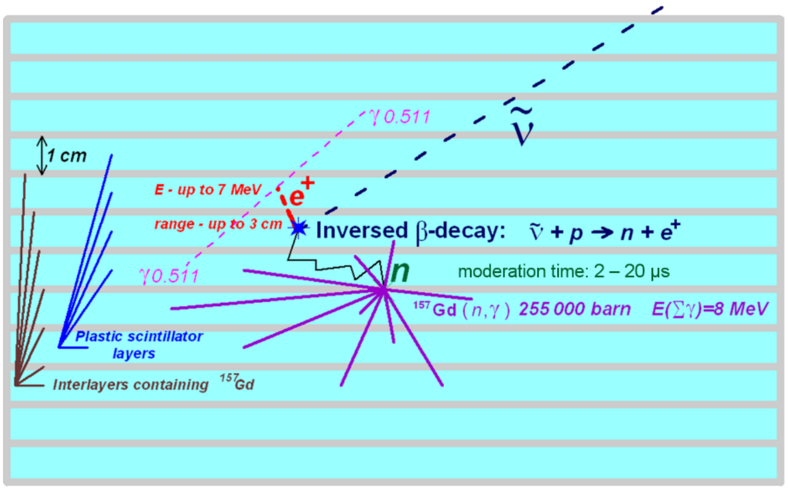

1 cm
E - up to 7 MeV
range - up to 3 cm
e+
γ 0.511
γ 0.511
ν̃
Inversed β-decay: ν̃ + p → n + e+
n
moderation time: 2 – 20 μs
157Gd (n,γ) 255 000 barn E(Σγ)=8 MeV
Plastic scintillator layers
Interlayers containing 157Gd

Figure 3-4 Inverse beta decay detection

Thanks to the segmented nature of the detector and the way the strips are placed, the IBD produces a characteristic signature which can be searched for. Therefore, the data acquisition system must be able to register a coincidence of two events separated by few microseconds. Generally, both prompt and delayed signals can come from the same PMT channel. The DAQ must be sufficiently responsive so that the delayed signal does not fall within the dead time.

Additionally, the DANSS detector is using cosmic muons with vertical tracks for calibration purposes and the DAQ must also record them.

4 S^3 detector

The S^3 detector project is a result of cooperation between DLNP JINR and IEAP Prague. Its main goal is the improvement of the energy resolution by optimizing the scintillating elements. The detector will be placed next to the DANSS detector in the Kalinin NPP and possibly in the Temelin NPP in the Czech Republic. The Czech contribution is being funded by a grant of the Czech Technological Agency TE 01020445 – CK RANUS.

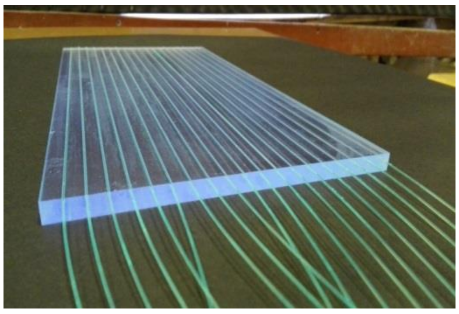


Figure 4-1 S^3 scintillating plate with WLS fibers

The S^3 detector consists of 80 20×40×1 cm plates (Figure 4-1) manufactured by the NUVIA Group placed in a X-Y pattern like the DANSS detector. The plates contain 19 grooves for WLS fibers collecting light. In contrast with the DANSS strips, the plates are not co-extruded with reflective and Gadolinium bearing coating, but manually wrapped in Teflon tape, as the co-extrusion process thermally changes the properties of the scintillator. Low density polyethylene foils doped with $10\%_{wt}$ Gd_2O_3, manufactured by the Institute of Macromolecular Chemistry (IMC) of Czech Academy of Sciences, are placed between the plates. The light from each individual plate is sensed by SiPM and amplified by an analog frontend.

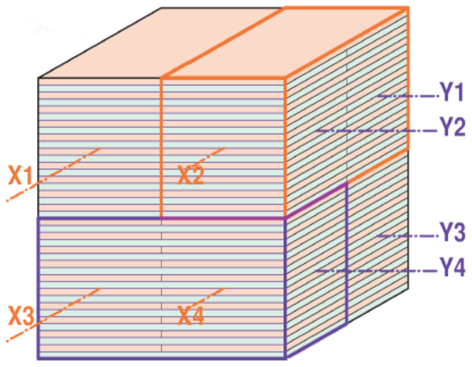


Figure 4-2 S^3 detector schematic

To understand properties of the detector a prototype called S-cubino was constructed (Figure 4-3). It consists of 18 plates divided into two interleaved groups connected to two Hamamatsu R7600U-300 PMTs.

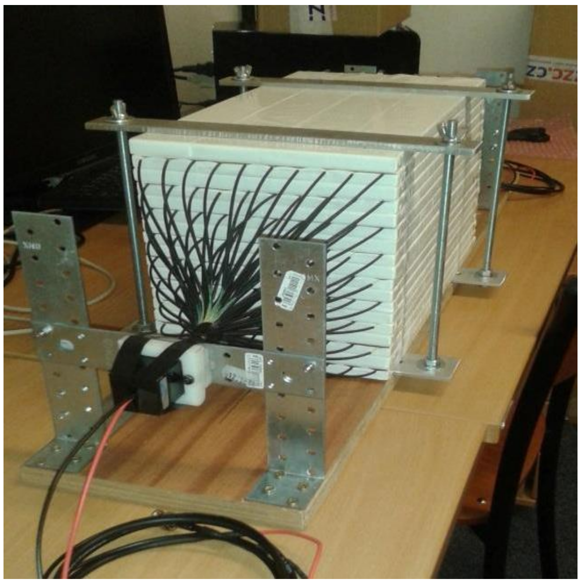


Figure 4-3 S-cubino detector

It is shielded by a 10-cm layer of hundred years old lead (for low internal radioactivity) to shield from gamma radiation, 8 cm of polyethylene for thermal neutron shielding and 8 cm of borated polyethylene for fast neutron shielding.

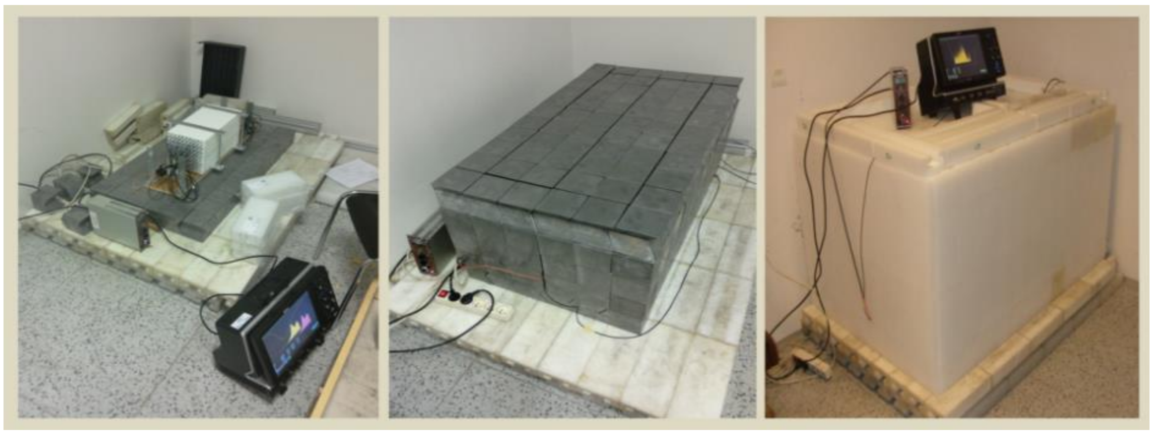


Figure 4-4 Construction of the scubino detector

Initial measurements with the S-cubino detector were made in the building of the Faculty of Mathematics and Physics of the Charles University in Troja in Prague and then the detector was moved into the underground shelter Bezovka in Prague.

The full detector is being constructed as of spring 2018 in the IEAP CTU for testing. All 19 WLS fibers are connected to one SiPM.

In Russia, the second S^3 detector prototype, in parallel development, prototype is being constructed. It will also consist of 80 scintillating plates, but will have use traditional PMTs for light collection. In addition, the detector will have a “gamma catcher” scintillating plates (5x50x100 cm) around the active detector to catch gammas generated inside the detector. As with the DANSS detector there will be active veto system to detect the fast neutron background events mimicking the IBD signature. Together, the detector will have 80 PMT per-plate channels, 16 bigger PMTs connected to the second scintillating side, 8 gamma catcher channels and 16 active veto channels totaling 120 channels to be sampled by the data acquisition system.

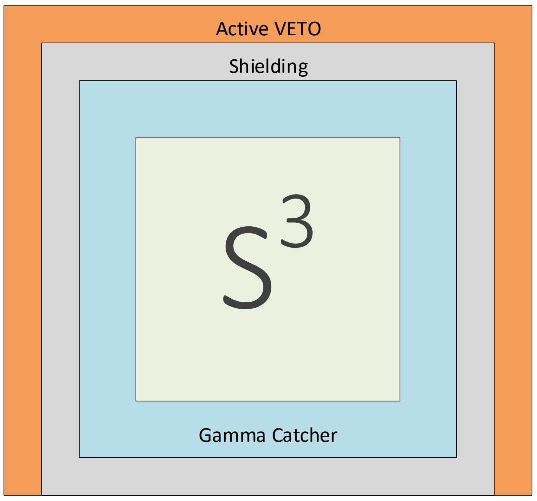


Figure 4-5 S^3 Detector layout, Russian version

The schematic layout of the S^3 detector is shown on Figure 4-5. Figure 4-6 shows the start of the prototype construction with one gamma catcher scintillator placed between lead shielding.

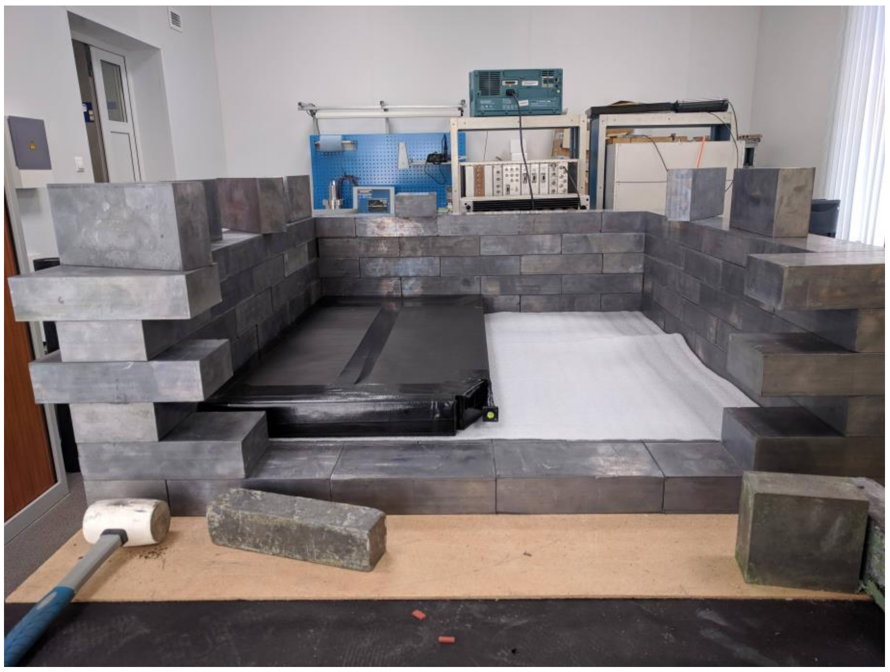


Figure 4-6 Gamma catcher scintillator

5 DANSS spectrometer

The data acquisition system is installed together with the DANSS detector assembly in a technical room underneath the reactor. Since the room is inside controlled-access part of the nuclear powerplant, the DAQ is connected via fiber optic ethernet connection to a control room (Figure 5-1).

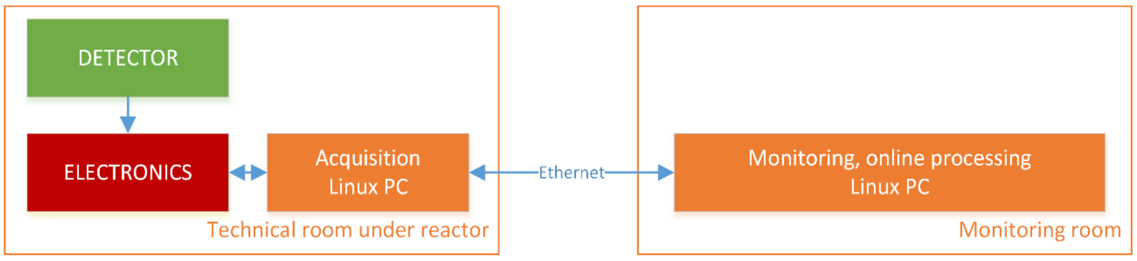

DETECTOR
ELECTRONICS
Acquisition Linux PC
Technical room under reactor
Ethernet
Monitoring, online processing Linux PC
Monitoring room

Figure 5-1 Overall system schematics

The acquisition PC collects data from the measurement electronics and saves them onto a disk. Monitoring PC connects to the DAQ via TCP connection for online monitoring of the detector by on-duty personnel. There is no outside connection out of the monitoring room, therefore acquired data can to be only transferred from the powerplant using an approved (by KNPP) USB Flash drive.

5.1 Signal path

Every PMT channel has its individual analog front-end electronics which is placed inside the detector shielding. A ribbon cable carries configuration RS-485 bus and power for the AFEs. The RS-485 is connected via an interface box to the acquisition PC. The AFE provides a configurable high voltage source for the photomultiplier and control voltage for above-threshold digital trigger comparator. Pulse shaper lengthens and amplifies the pulses so they can be captured by the electronics. Since the trigger logic has a propagation delay, it is necessary to delay the analog signals using a delay line (coax cable with 80 ns t_{pd}).

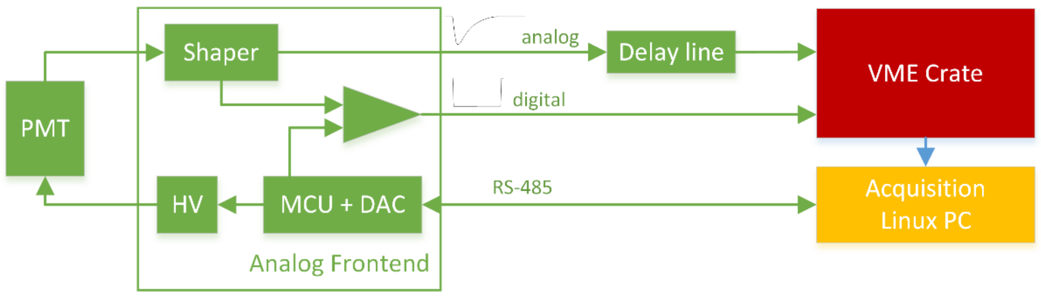


Figure 5-2 PMT Signal Path

5.2 CAEN VME-Based Spectrometer

Overall schematics of the PMT spectrometer are shown in Figure 5-3. The spectrometer is installed in three NIM crates and one VME crate. NIM crate is used for signal fan-out and the VME crate for the spectrometer itself. The spectrometer consists of signal acquisition modules (QDC, ADC, FPGA), control (FPGA) and conversion (FPGA). The author of this thesis has been responsible for the development of the firmware for the FPGA V1495 modules. Hardware of the spectrometer is composed of several modules mentioned below.

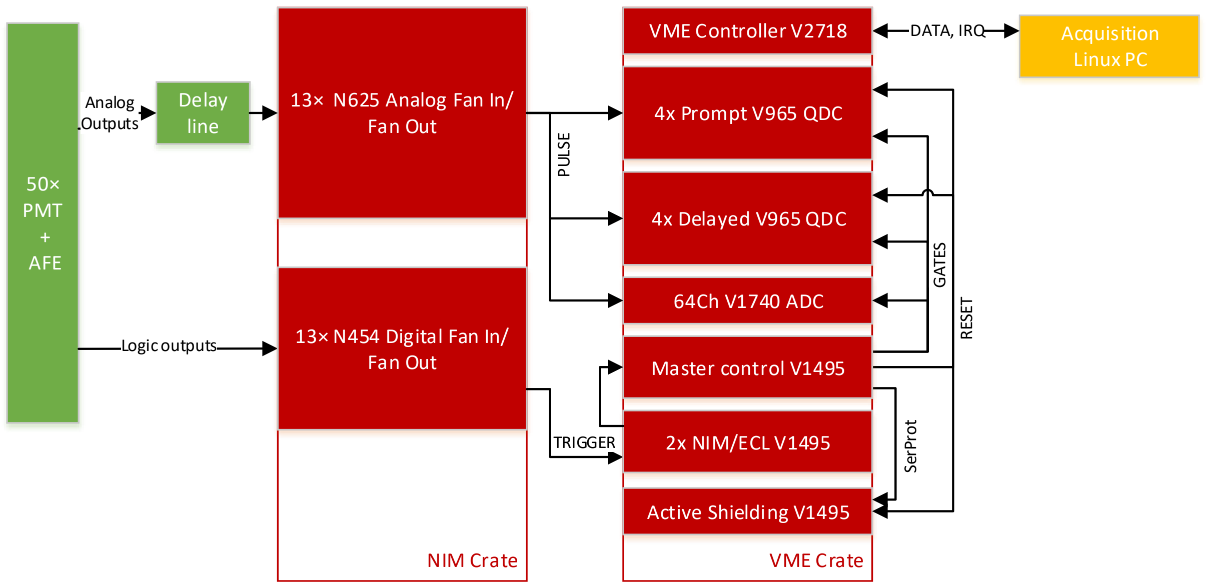


Figure 5-3 Hardware schematics

5.2.1 N625 Quad Linear Analog Fan-In/Fan-Out

The CAEN N625 has 4 groups of 4 analog summed inputs with 4 analog outputs each. This card is used to branch the analog signals from AFEs to Prompt, Delay QDCs and the ADC. For 50 channels, 13 such cards are required.

5.2.2 N454 Logic Fan In/Fan Out

CAEN N454 has 4 groups of NIM logic digital OR-ed inputs with four direct outputs and two negated outputs. It is used for as a signal repeater and a fan-out for debug purposes.

5.2.3 V2718 Controller

The CAEN V2718 VME Controller serves as a bridge between the VME crate bus and a Linux PC with DAQ software. It is connected via a multimode optical fiber link. The VME cards are addressable by 32-bit address, where the first 16 bits are set by either by a rotary switch (VME64 crates) or slot position (VME64X crates). The VME64 bus has bandwidth of

80MB/s. The VME bus supports single read/write, burst 4KB block (BLT) and chained burst (CBLT) accesses modes. The chained block access mode uses a multicast target address which is set for a group of similar adjacent cards. The CBLT is used for reading data from the V965 QDCs. The VME Bus has also 8 IRQ lines, the V2718 bridges them to a PC IRQ.

5.2.4 V965 Charge to Digital Converter (QDC)

The 16-input charge to digital converter integrates the charge on input channels during active level-sensitive GATE signal and converts it using 12 bit ADC and creates an data event accessible via the VME bus. Each channel is converted twice, once with gain of 1× and once with gain of 8×. The GATE signal must precede the analog input by at least 15 ns [20]. The conversion of all channels takes 5.7 µs, with a total dead time of 6.9 µs.

5.2.5 V1740 ADC Digitizer Card

CAEN V1740 is a 64-input 12 bit 62.5 MSPS simultaneous analog to digital converter with a 2 V_{pp} input range. It has a 192k samples per channel memory buffer, which can be divided into up to 1024 separate events (of 192 samples each). The digitizer can be triggered either internally by any channel exceeding a threshold voltage, or externally via an edge-sensitive GATE signal. After a trigger, all enabled channels are sampled and stored to event buffer. Additional triggers during the acquisition window can be either ignored or cause the acquisition to be prolonged (trigger overlapping mode). The digitizer operates without a dead-time. That is, another event can be acquired as soon as previous ends without any delay.

5.2.6 V1495 GPIO FPGA Card

CAEN V1495 is a VME card with an empty user-programmable Altera Cyclone I EP1C20 FPGA with 20K Logical Elements and 36 KiB on-chip ram. The card provides two 32 channel LVDS/ECL/PECL high-density (3M P50E-068-P1-SR1-TG) inputs, one 32 channel LVDS outputs, 2 LEMO NIM/TTL inputs/outputs and one LED. Additionally, three add-on slots are available, which can be filled by 32-channel LVDS/ECL/PECL inputs, 32-channel LVDS output, 8-channel NIM/TTL LEMO input/output daughter cards.

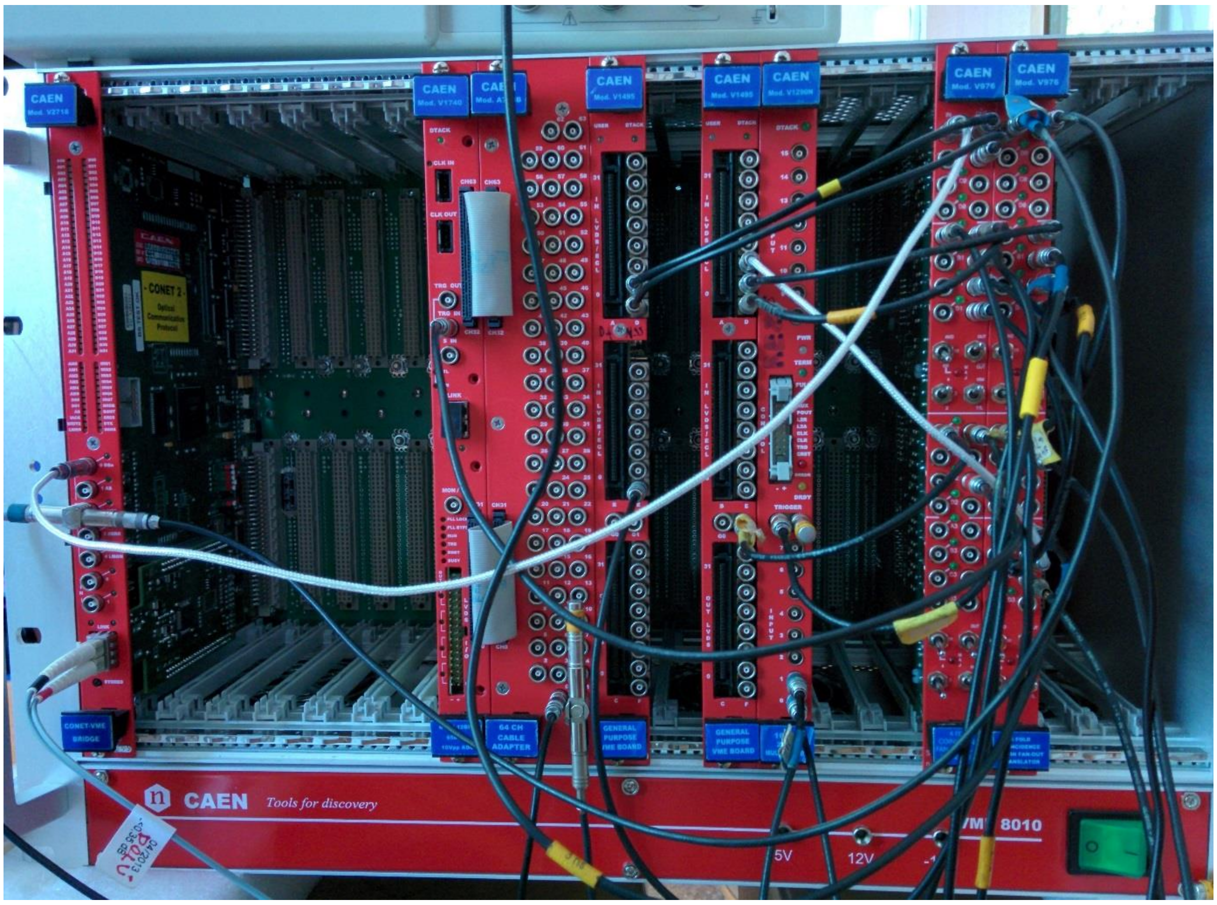


Figure 5-4 Development VME Crate

5.3 Data Acquisition Software

The spectrometer data are collected, monitored and partially on-line processed by a system described in detail in [21]. It is a system running on a Linux PC. An interesting feature of the system is that not only the vendor-specific raw data from the VME modules are written to the disk, but the system also translates them into a unified data(u-data) format formed by a tuple (*channel_id; value*) hiding the complexity and variability of the specific hardware. It also allows on-line calculation of areas (charges) of pulses. That allows creation of data processing programs universally usable for differently configured crates. Both raw and u-data are available for remote access via TCP/IP and so is program control. Figure 5-5 shows the overall DAQ schematics. The measurement-specific acquisition algorithm is described using an XML based language containing sections appropriate for different phases of the measurement (crate initialization, start, IRQ handling operations, end and data storage). The DAQ also contains an Qt/ROOT-based application allowing for on-line status monitoring.

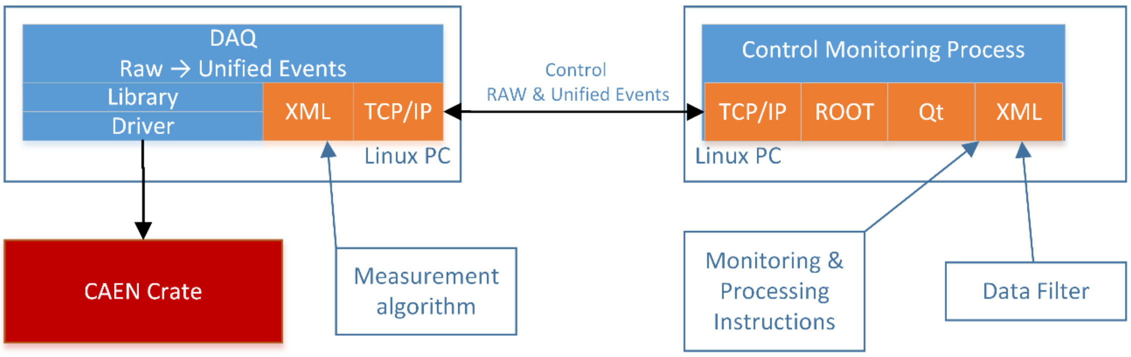


Figure 5-5 DAQ System Architecture

5.4 Data event structure

Figure 5-6 shows simplified example of inputs together with the output data structure of the DAQ event. Each input pulse is identified by its relative timestamp(TS), collected charge from both ADC and QDC and a bit mask of associated active shielding veto events.

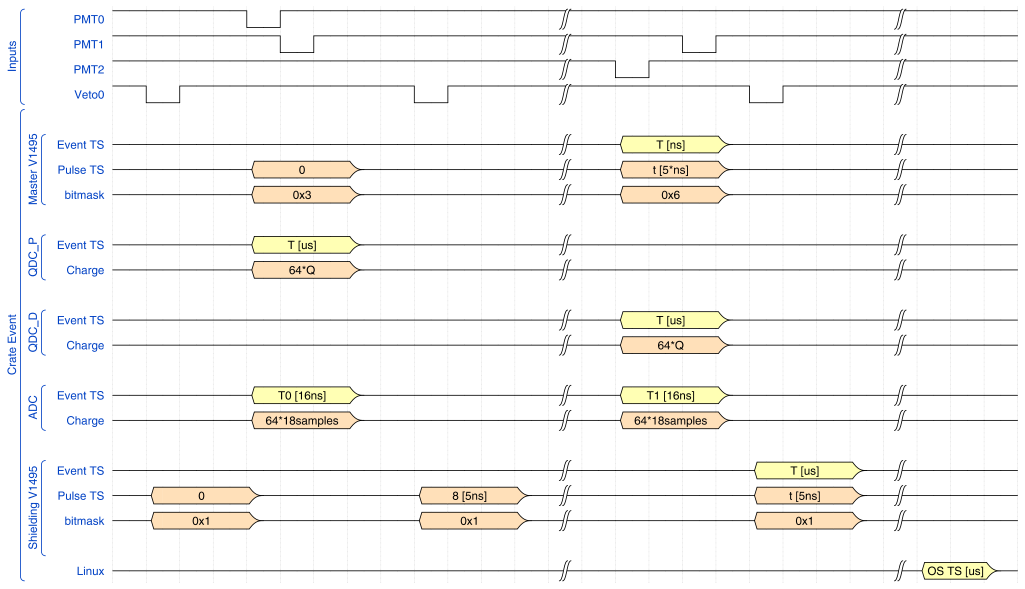


Figure 5-6 DAQ Crate event structure

Figure 5-7 shows the responses of the control FPGA to example trigger inputs. The firmware sends enabling GATE signals to appropriate prompt and delayed QDCs, ADCs and the active shielding FPGA firmware.

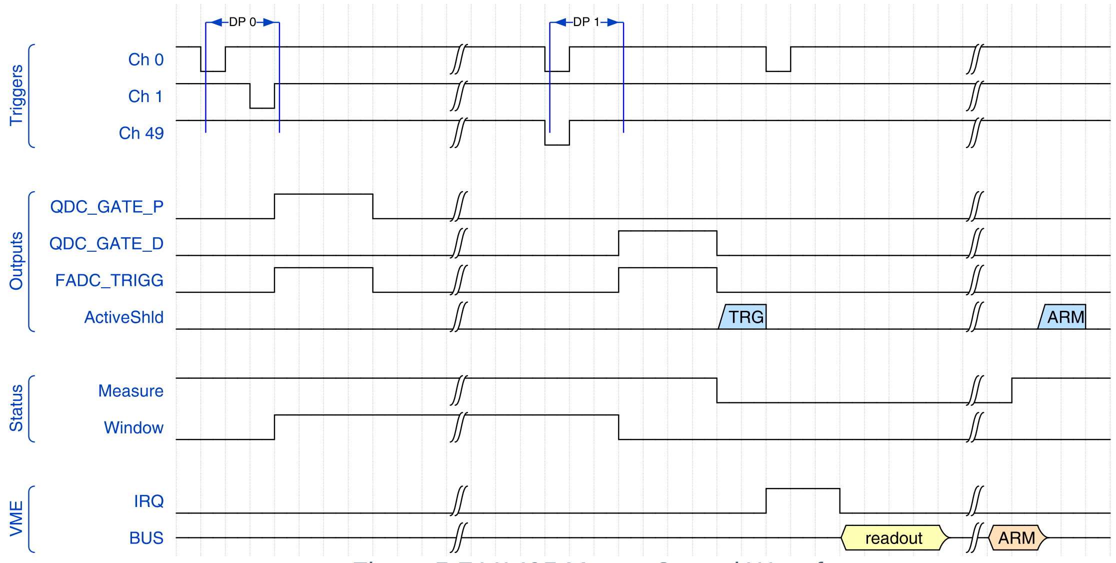


Figure 5-7 V1495 Master Control Waveforms

6 DANSS Firmware Implementation

The development of the FPGA firmware for the V1495 cards of the DANSS PMT spectrometer has been the responsibility of the author of this thesis.

The final configuration of the spectrometer requires the V1495 in 4 roles served by different firmware variations:

- master crate measurement control
- active shielding pattern recorder
- pattern generator.
- NIM/LVDS translator

The master measurement control firmware monitors incoming above-threshold pulses from PMT AFEs and if a configured pattern is observed, sends GATE signals to appropriate QDCs, ADC and the active shielding pattern recorder.

The pattern recorder firmware continuously samples above-threshold pulse outputs of the shielding scintillators and stores events inside a circular buffer with their relative timestamps.

The NIM/LVDS translator is a simple logic level translator.

Pattern generator firmware generates configurable pulses used for testing of the entire data acquisition system.

However, because the functionality can be useful in other experiments, the firmwares have been designed with as much run-time configurability via the VME bus as possible and there is no hardcoded configuration.

6.1 V1495

The firmware is designed for the user FPGA of the CAEN V1495 card used in the spectrometer. V1495 contains a CAEN-programmed system FPGA and an empty 20K LE Cyclone I chip.

The system FPGA interfaces the VME bus, contains 256 32-bit word data readout FIFO and can assert configured VME IRQ line and maps VME read/write accesses to a local bus. User FPGA is connected to 195 user I/O. Schematic diagram is shown in Figure 6-1.

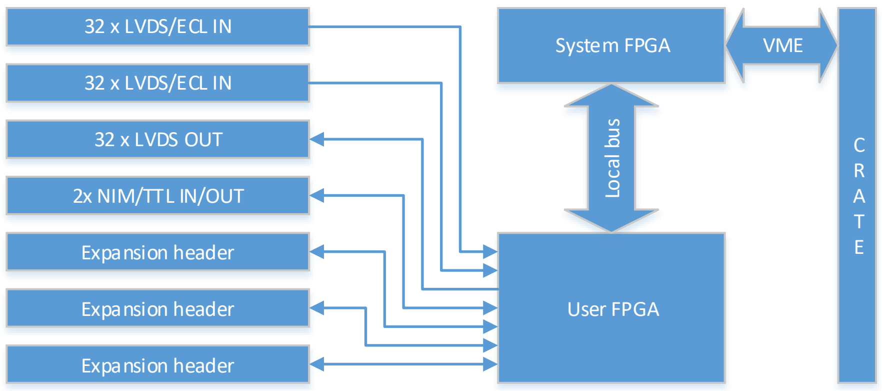


Figure 6-1 V1495 schematic diagram

6.1.1 V1495 local bus interface

The signals used between the system and user FPGAs are shown in Table 6-1. The bus supports single read, single write and burst (BLock Transfer - BLT) read .The local bus supports only 32-bit word VME access mode.

Signal name	Direction	Width [bits]	Description
nRESET	in	1	System reset
CLK	in	1	40 MHz system clock
nADS	in	1	Start of Address Data Cycle
nBLAST	in	1	Terminate transfer
WnR	in	1	Write or Read
nREADY	out	1	Output data ready or BLT request
LAD	inout	16	Bidirectional data/address bus
nIRQ	out	1	Interrupt request

Table 6-1 V1495 local bus signals

Single read and write cycles start with the system FPGA deasserting the nADS signal and placing 16-bit address on the bus. In the single read (Figure 6-2) bus cycle the system FPGA waits until the nREADY signal is deasserted. In the single write (Figure 6-3) cycle data follows the address immediately.

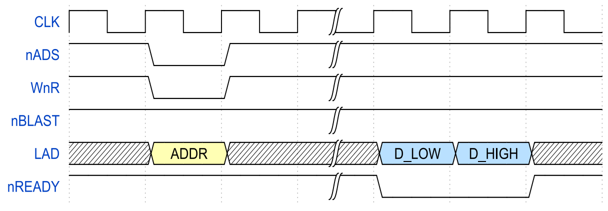


Figure 6-2 Local bus read cycle

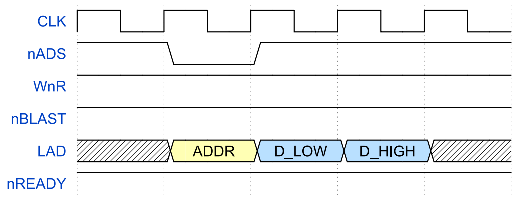


Figure 6-3 Local bus write cycle

The BLT transfer is used to transfer event data from the user FPGA to the system FPGA FIFO for data readout. The cycle begins with the user FPGA deasserting the nREADY signal and the system FPGA performing read access to address 0x0000. If the system FIFO becomes full, nBLAST is deasserted and the transfer is interrupted.

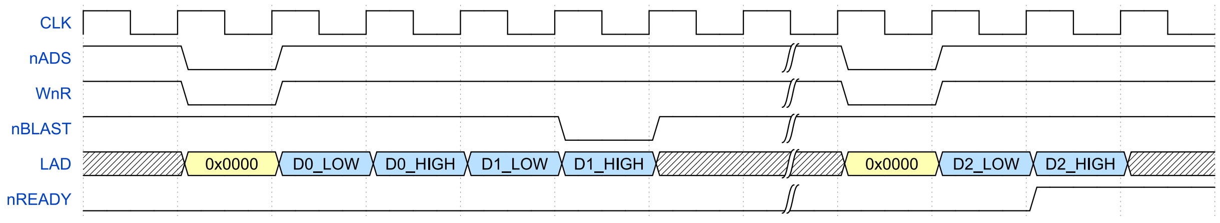


Figure 6-4 Local bus BLT cycle

6.2 Firmware framework

It was impossible to fit all the required roles in one firmware due to the small size and performance of the user FPGA. Therefore, a modular approach to development was used. Each firmware variant is built from a common code base and contains a shared VME interface part, variant-specific part and external signal routing (Figure 6-5).

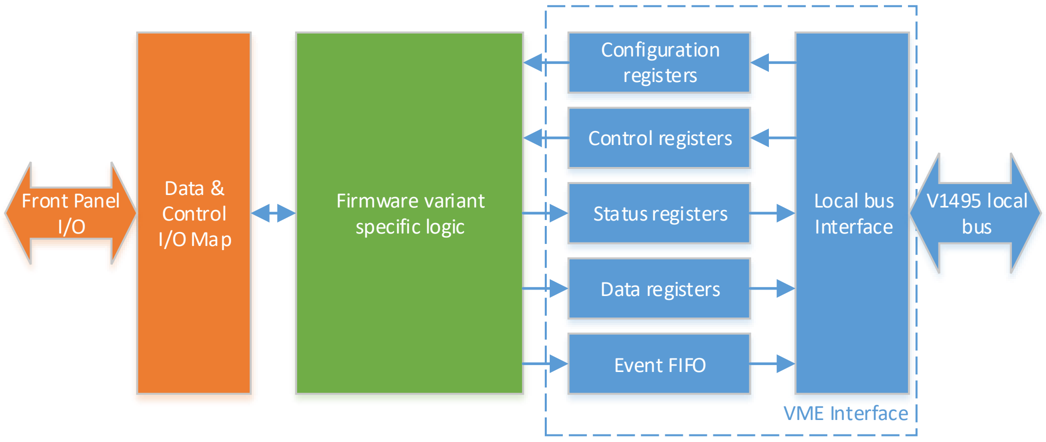


Figure 6-5 Firmware framework structure

6.2.1 Build process

The firmwares are generated from single Quartus project using revisions. Each firmware has its separate revision. Project source code is stored in a git repository. The build process uses external Makefile.

At the beginning of compilation a pre-flow TCL script generates a VHDL file containing the revision name and revision version in the format of YYMMDDcc, where cc is the number of git commits made on the build day. These constants can be read from the firmware via the VME bus to verify used variant and revision of the source code used. A post-flow script converts the firmware bitstream to the required RBF format and renames the output file to reflect the variant name and revision.

After the compilation finishes, make executes a ModelSim testbench which generates a list of all available configuration registers and their default values. The list is then converted to XML format used by the DAQ software and VHDL for testbench creation.

6.3 Firmware building blocks

6.3.1 Configuration and status

The configuration and status block implements VME-accessible 32-bit registers. Each register can be set to either read, write or read/write access. The logic circuit used is shown in Figure 6-6. VHDL implementation of procedure is shown in Listing 6-1 and their example use in Listing 6-2.

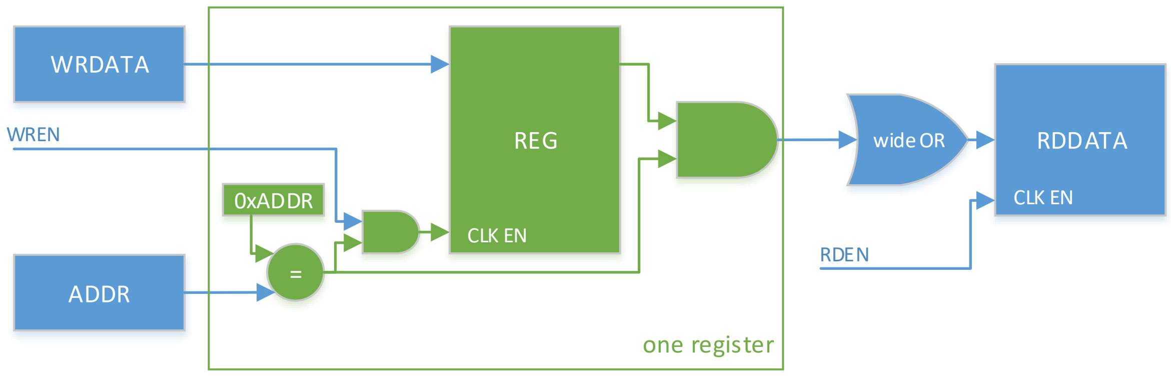


Figure 6-6 Configuration registers schematics

```
procedure reg(raddr : unsigned;                   -- register address
              variable regval : inout unsigned; -- register variable
              racc : reg_t;                     -- register access (r, w, rw)
              purpose : purpose_t;              -- documentation: (cfg / stat)
              pgroup : natural;                 -- documentation: group
              gtype : gatetype_t;               -- documentation: type of gate
              name : string;                    -- documentation: name
              desc : string;                    -- documentation: description
              values : string                   -- documentation: possible values
              ) is
       variable isaddr : std_logic;
       variable tmp    : std_logic_vector(regval'range);
begin
       -- synthesis translate_off
       print_reg(raddr, std_logic_vector(regval), racc, purpose, pgroup,
                 gtype, name, desc, values, "unsigned", regval'length,
                "D32"); -- used by ModelSim in the build flow
       -- synthesis translate_on
       if racc = r or racc = rw then
              isaddr := '0';
              if addr = raddr then
                     isaddr := '1';
              end if;
              tmp := std_logic_vector(regval) and (regval'range => isaddr);
             -- rd is a process-wide variable
              rd(regval'range) := rd(regval'range) or tmp;
       end if;
       if racc = w or racc = rw then
              if addr = raddr and wr_en = '1' then
                     regval := unsigned(wr_data(regval'range));
              end if;
       end if;
end procedure reg;
```

Listing 6-1 VHDL implementation of registers

```
reg(X"100C", c.fw_ver, r, status, "FW_VER", "Firmware version", "YYMMDDrr");
reg(X"101C", c.fw_name, r, status, "FW_NAME", "Firwmware identifier",
      "ASCII encoded");
for i in 0 to 7 loop -- QDC gates
       addr_base := 16#5000# + i * 16#100#;
       sub       := c.gate_conf(i);
       reg(u(addr_base + 16#04#), sub.tmr_wait_gate.duration, w, cfggate, i,
           core_qdc, "GATE_" & to_string(i) & "_PRE_DURATION", "Pre-gate duration",
           "Number of 5 ns clock periods");
```

Listing 6-2 Example use of register

6.3.2 System timestamp

A 32-bit counter running at the core frequency (200 MHz) is used for timestamp of data events. A clock divider (default is 1 μs) and its overflow period can be configured. It is possible to reset the system timestamp either using the serial protocol or a dedicated edge-sensitive time synchronization input.

6.3.3 Signal input

Above-the-threshold signals are synchronized to the 200 MHz core clock domain and configurable signal edge is detected. A de-glitch filter is applied to ignore any additional edges within configured time. Output of this block is an edge signal asserted for one core clock cycle.

6.3.4 Data read-out controller

Data read-out block controls the measurement. After reset, it is initialized to a standby IDLE state. A register write to the ARM register starts the measurement. The read-out controller waits until assertion of EVENT_READY signal from the firmware core. Then it requests a BLT transfer from the system FPGA and after its completion, an IRQ is asserted. It is possible to delay the IRQ request using a settable timer to allow other VME crate modules to finish their processing. The readout controller then either waits for another write to the ARM register or re-arms the system automatically, after a settable timer.

The size of the event transferred to the system FPGA FIFO is tracked and available for reading. If the FIFO is full, automatic rearms are blocked. Reading the FIFO_SIZE register resets it. Thus, the read-out cycle from the point of view of the DAQ is as follows:

1. Write 1 to ARM register

2. Wait for IRQ
3. Read FIFO_SIZE register
4. Block transfer from the V1495 card

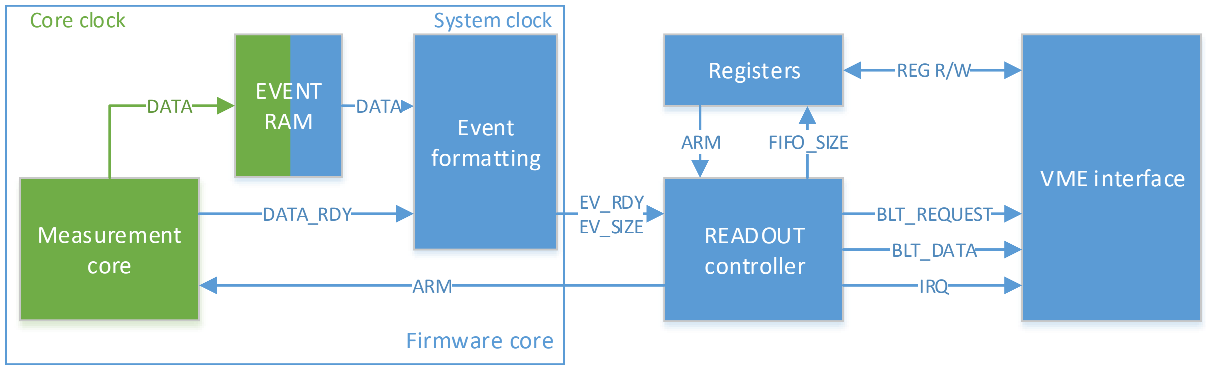


Figure 6-7 Data readout schematics

The relationship of the readout controller to other modules is shown in Figure 6-7.

6.3.5 Serial communication

The V1495 does not allow the user firmware to request a transfer on the VME bus, therefore any communication between two V1495 cards must be performed via the front-panel I/O. A simple pulse width modulation is used. The Table 6-2 lists all currently implemented commands.

Command	Pulse Length [5 ns]	Accepted length [5 ns]	Description
SAMPLE	2	1 – 3	Detector pulse
WIN_FAIL	5	4 – 6	Coincidence window unsuccessful
WIN_OK	8	7 – 9	Coincidence window successful
ARM	11	10 – 12	Re-ARM
TIMESYNC	14	13 – 15	System TS overflow
RESET	23	22 – 24	System reset

Table 6-2 Serial protocol commands

6.3.6 Output GATE generation

6.3.6.1 Single gate

The single output gate generates a level-active signal of configurable polarity which has a programmable pre-gate wait, gate and post-gate duration as shown in Figure 6-8 . The GATE signal is routed to the V1495 front-panel I/O and the BUSY signal can be used internally in the firmware core.

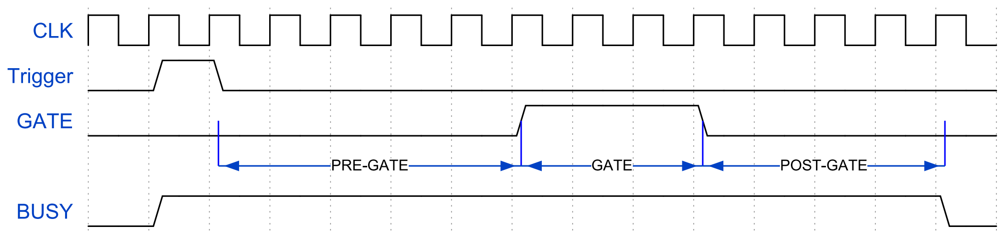


Figure 6-8 Single GATE timing diagram

6.3.6.2 Dual alternating gate

The dual alternating gate has two outputs, GATE1 and GATE2. It can operate in two modes, latency hiding and always alternating. In the latency hiding mode the GATE2 is used for output gates only if the GATE1 output is already busy (Figure 6-9) and not currently generating active gate signal. A configurable timer can lengthen the time the other channel is blocked after the end of active gate signal.

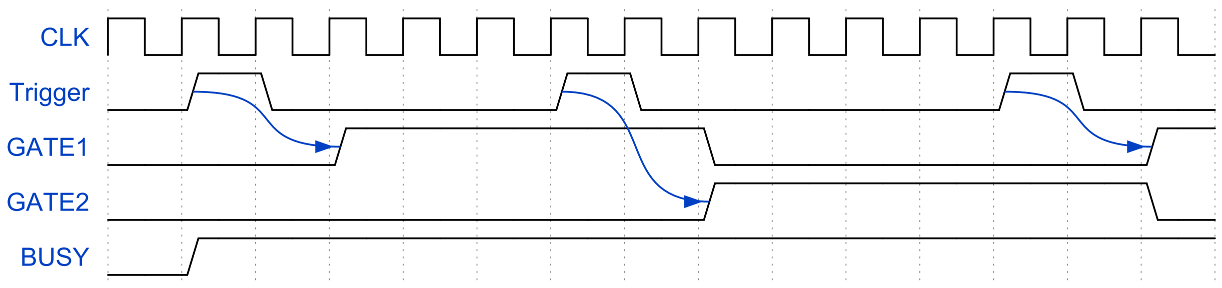


Figure 6-9 Alternating gate - Latency hiding mode

In the always alternating mode (Figure 6-10), additional input DP (Detector Pulse) is used to switch between gate outputs.

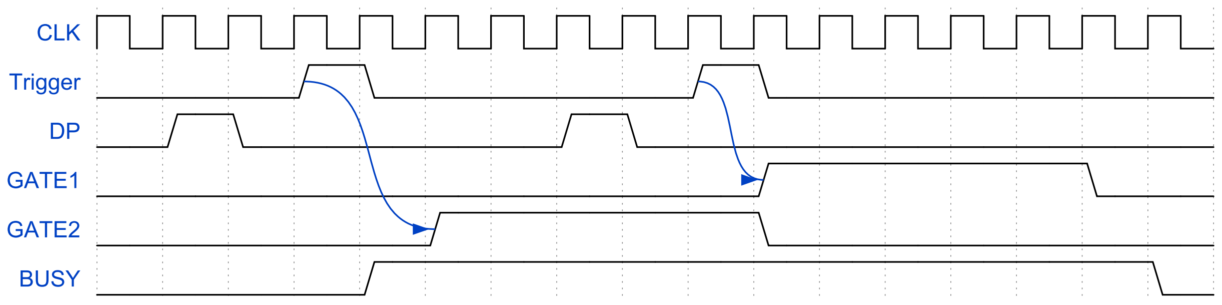


Figure 6-10 Alternating gate - Always alternating

The dual gate can be also configured to swap output channels, generate output in both channels at once and to stop generating pulses after two triggers (until system RE-ARM).

6.3.7 Status gate

Monitoring of internal status of the firmware is possible using the status gate. It is configured using a bitmask register. The output is either selected level-active internal signal or an edge-sensitive signal lengthened using the single output gate. Table 6-3 lists available output signals.

Type	Value	Description
edge	0x1	Any detector pulse
	0x2	Recorded detector pulse
	0x4	Successful coincidence window
	0x8	Unsuccessful coincidence window
	0x10	System RE-ARM
	0x20	External time synchronization
	0x40	System timestamp overflow
level	0x100	System measuring (i.e., not dead-time)
level	0x200	System is in the RUN state
level	0x400	IRQ signal is asserted
level	0x800	Measurement window open
level	0x1000	Waiting for DAQ (active between IRQ assertion and ARM reception)

Table 6-3 Status signals

6.4 Firmware cores

6.4.1 Coincidence core

The coincidence core implements the temporal coincidence of spatial coincidences of 64 input channels operating at 200 MHz (5 ns sampling time). Whenever an edge is sensed at the input, a small (~100 ns) detector pulse coincidence window opens and captures trigger pattern occurring in the detector. Its purpose is to capture a single physical interaction in the detector which was due to the varying propagation delays in the detector and analog electronics chain captured at different clock edges. The DP opens a main coincidence window (~ 100 μs). If the coincidence conditions are met, an event is generated. The event contains a coarse system timestamp and a list of detector pulses with their channel patterns and fine timestamp relative to the main coincidence window start. Schematics of the module are shown in Figure 6-11.

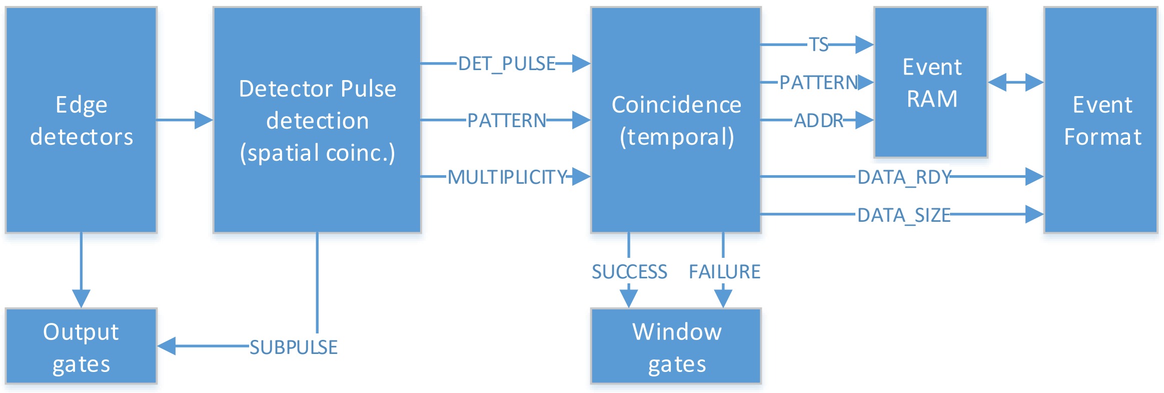


Figure 6-11 Coincidence master schematics

6.4.1.1 Coincidence core modules

The DP spatial coincidence module opens a coincidence window whenever any edge detector asserts a single clock pulse. For the duration of the window, the 64-bit input vector pattern is logically summed. The module also counts the number of ones in the vector. After the window closes, the module asserts the DET_PULSE signal to the main coincidence module together with its 64-bit pattern and multiplicity (i.e. the number of ones in the vector). A dead-time after the window end can be set to disable inputs for configured time.

The main coincidence module operates in two different modes – **fixed and dynamic window**. In fixed window mode, the coincidence window always lasts the configure time and at the end the window is considered as successful if the number of pulses is either within configured range (minimum and maximum) or a multiplicity of any pulse is higher than configured. In the dynamic mode, the window ends as soon as the minimal number of pulses or multiplicity is recorded. Each pulse pattern and its relative timestamp is stored in the Event RAM. A counter keeps track of the total number of detector pulses observed between two successful coincidence windows. This allows the DAQ to calculate the ratio between recorded pulses and total pulses of the detector (i.e. the detection efficiency).

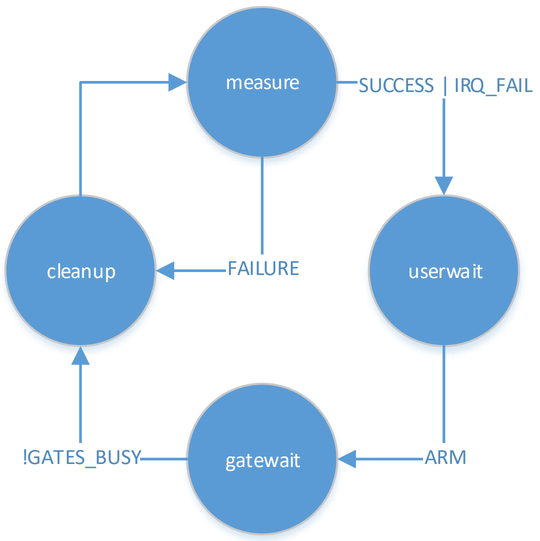


Figure 6-12 Coincidence master measurement state machine

The measurement is controlled by a state machine (Figure 6-12). The *measure* state waits until the main coincidence window finishes. Successful window results in the shared data readout module notification. A configuration option is present to generate IRQ even in case of coincidence window failure (used when there is a need to programmatically clear some other block in the VME crate). In the *userwait* state the machine waits for the DAQ to send the ARM command. The *gatewait* state waits until all output gates are completed. In the *cleanup* state all gates are reset and the state machine transitions back to *measure* state after 2 clock periods.

6.4.1.2 Coincidence core outputs

The coincidence core can be generated with two kinds of output gates – eight 8:1 single gates or four 16:1 dual alternating gates. For the 8:1 single gates, inputs 0 to 7 are mapped to first output channel, 8 to 15 to second and so on. Similarly, channels 0 to 15 are mapped to the first output dual alternating gate. If the always alternating mode is enabled, like in the DANSS spectrometer, the 4 output gates act effectively as one dual output alternating gate.

The core contains one single gate triggered by coincidence window success, one for coincidence window failure and 8 independent configurable status gate outputs.

If enabled, the core is generated with the serial protocol master for control of slave modules (the active shielding pattern recorder).

6.4.1.3 Coincidence core event format

The format of the event (Table 6-4) produced by the core is designed to be similar to other CAEN VME modules to simplify event decoding by the DAQ and users. GEO is an identifier of the card location inside the VME crate and must be configured by the user. CRATE denotes the number of VME crate in the measurement system and is also configured by the user. N_DETECTOR_PULSES is the number of detector pulses detected by the system starting at the end of the previous event. The coarse event timestamp is typically configured to 1 μs period and the HIT_TIME period is always 5 ns, with the first hit having a time of zero.

Header

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	1	0	CRATE[7..0]								DATA_SIZE															

Data size is the length of the event without header and footer.

Coarse Event TS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	0	0	0x02								TS_LOW															
GEO[4..0]					0	0	0	0x03								TS_HIGH															

Detector pulses since last IRQ

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	0	0	0x04								N_DETECTOR_IMPULSES															

Reset after the end of a successful coincidence window.

Event data x N

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	0	0	0x01								HIT_TIME															
PATTERN 31 – 0																															
PATTERN 63 – 32																															

Repeated as many times as required.

Footer

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					1	0	0	EVENT_CNT[23..0]																							

Table 6-4 Coincidence master event bit structure

6.4.2 Pattern recorder core

Pulse pattern recorder core is designed to record events around given point of time, such is the case when registering events from active veto detectors. It samples inputs using 5 ns clock and continuously records Detector Pulses into a circular buffer. Pulses which are over configured numerical limit or are too old are removed. After an external trigger signal is received an event is generated.

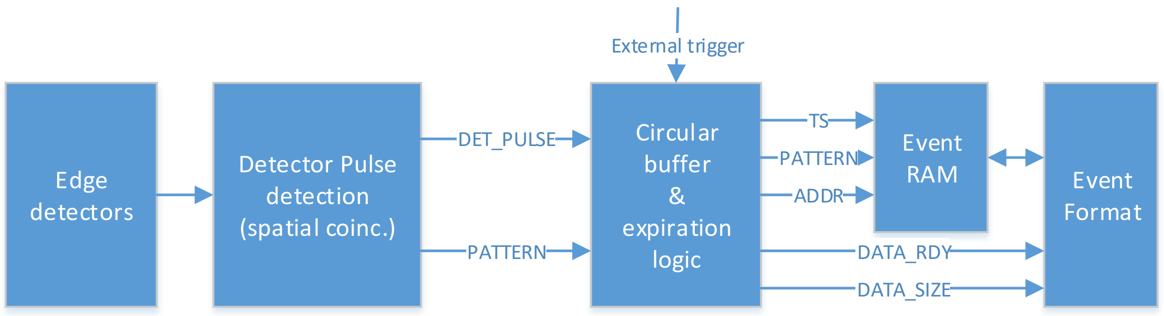


Figure 6-13 Pattern recorder core schematics

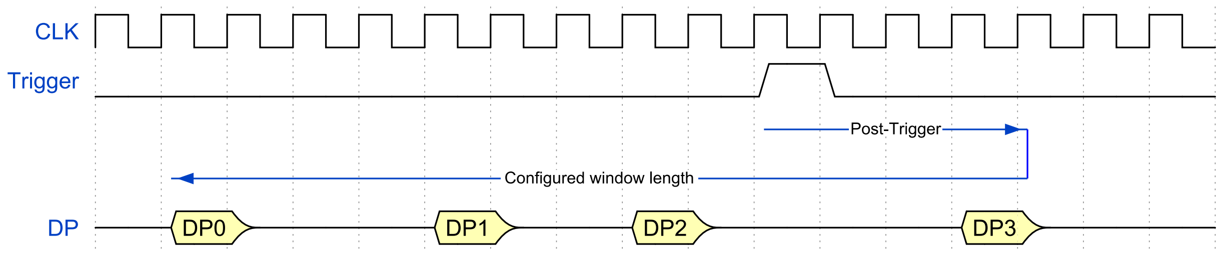


Figure 6-14 Pattern recorder event example

6.4.2.1 Pattern recorder core event format

Header

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	1	0	CRATE[7..0]								DATA_SIZE															

Data size is the length of the event without header and footer.

Coarse Event TS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	0	0	0x02								TS_LOW															
GEO[4..0]					0	0	0	0x03								TS_HIGH															

Detector pulses since last IRQ

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	0	0	0x04								N_DETECTOR_IMPULSES															

Reset after the end of a successful coincidence window.

Event data x N

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					0	0	0	0x01								HIT_TIME															
PATTERN 31 – 0																															
PATTERN 63 – 32																															

Repeated as many times as required.

Footer

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GEO[4..0]					1	0	0	EVENT_CNT[23..0]																							

Table 6-5 Coincidence master event bit structure

6.5 Gate generator core

It is useful to be able to generate artificial pulses of given structure for DAQ testing purposes. The gate generator outputs groups of pulses in a repeated window. After specified number of cycles, an IRQ can be generated. Each channel produces one pulse of specified duration relative to the start of the window in 5 ns steps. After each cycle, this offset can be either incremented or decremented and such adjustment be kept for given number of cycles. The gate generator core is useful for hardware in the loop testing of systems, including testing of various race conditions.

6.6 Firmware resource usage

Table 6-7 shows the Cyclone I FPGA resource usage for different variants of the firmware.

Firmware ID	Description
QDCM	Coincidence firmware
MPPC	Pattern recorder
GGEN	Pattern generator
NECL	NIM->ECL translator
ECLN	ECL->NIM translator

Table 6-6 Firmware ID codes

Firmware ID	Logical element usage	Memory usage [bits]
QDCM	7524	28 672
MPPC	7692	98 304
GGEN	17833	8192
NECL	168	0
ECLN	170	0

Table 6-7 Firmware resource usage

7 S^3 software coincidence analyzer

The spectrometer for the S^3 detector is based on the CAEN 64 channel 12-bit 62.5 MSPS V1740 VME Digitizer (Described in 5.2.5). The V1740 is configured in a self-triggering mode where rising edge on any input triggers simultaneous sampling of all enabled channels. The data is stored in an internal event buffers, IRQ is generated and read out by a PC running the NWVME DAQ software by Zdenek Hons [21]. Captured events are stored on the hard drive together with a system UNIX timestamp.

As part of this thesis, the SwCoinc application for data analysis was created by the author of this thesis. Its design requirements were:

- implement the inverse beta decay search algorithm
- Ability to process offline and online data from the DAQ and oscilloscopes
- online data reduction (coincidence, filtering)
- online data visualization
- ROOT file data output

The software framework has been written in C++ and uses the CERN ROOT framework for data visualization.

The description of the final inverse beta decay search algorithm implementation can be seen on Figure 7-1. The blue rectangles represent processes and green data. At the start of the processing is the raw ADC data file and at the end is a data structure containing coincidences of a group of pulses with likely IBD candidate detector events.

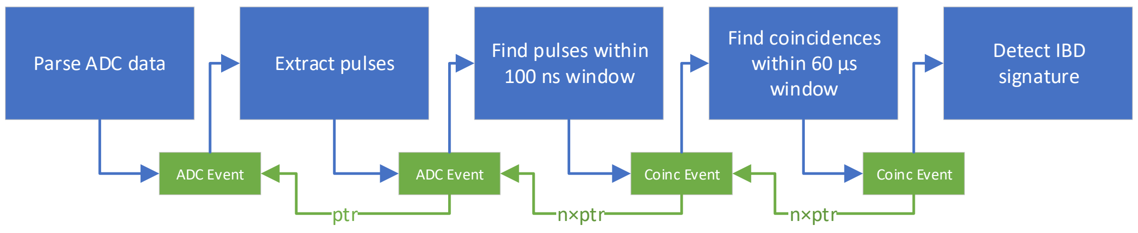


Figure 7-1 SwCoinc data flow

7.1 SwCoinc software description

7.1.1 Data input

The SwCoinc framework reads VME crate events created by the Hons DAQ and LeCroy oscilloscope TRC wave traces and supports offline and online processing.

In the offline mode, a directory or a file is specified as an argument on the command line and the application recursively scans the directories for registered file extensions and ZIP archives containing them. One archive or one directory is defined as a single data acquisition (run). The files are sorted using a regular expression according to their extensions – the VME files have format DDMMYY:HHMMSS-*n*_0.vme_r, where *n* is the file sequence number, and the LeCroy TRC files C*x*_y.TRC, where *x* is the oscilloscope input channel with *y*-th acquisition. Separate sets are created for each input channel.

In the online mode, only VME DAQ is supported and the application takes the path to the VME DAQ binary as an argument and monitors the /proc file system using the Linux inotify facility to detect whether it is running. If found, the /proc/pid/fd directory is scanned for the location of its log file. The VME log file is parsed to detect the state of the DAQ and possible file names of an active run.

The input data files are parsed into the Event class structure. Each *Event* contains aligned ADC samples from all active input channels and the timestamp. When segmented oscilloscope files are read, first *Event* contains first acquisition from all input channel files.

7.1.2 Data analysis

Events are processed through *Analyzer*, a vector of *Filters*. The framework contains two analyzers – ADCPrinter and SCube. ADCPrinter outputs every a trace of ADC *Events* to a separate PNG file. The SCube analyzer implements the spatial and temporal coincidences to detect reactor antineutrinos.

7.1.3 Data visualization

The coincidence software contains graphing classes based on ROOT's TH1 and TGraph. The visualization classes automatically create histograms for individual channels with correct names and axes and graphs of tracked variables depending on time (such as system load, pulses per second). After data processing is done, all graphs and histograms are saved as PNG files to output directory and as well as a ROOT data file.

Online monitoring uses the embedded HTTP ROOT server, which allows online access to open ROOT files via the JSROOT interface (Figure 7-2).

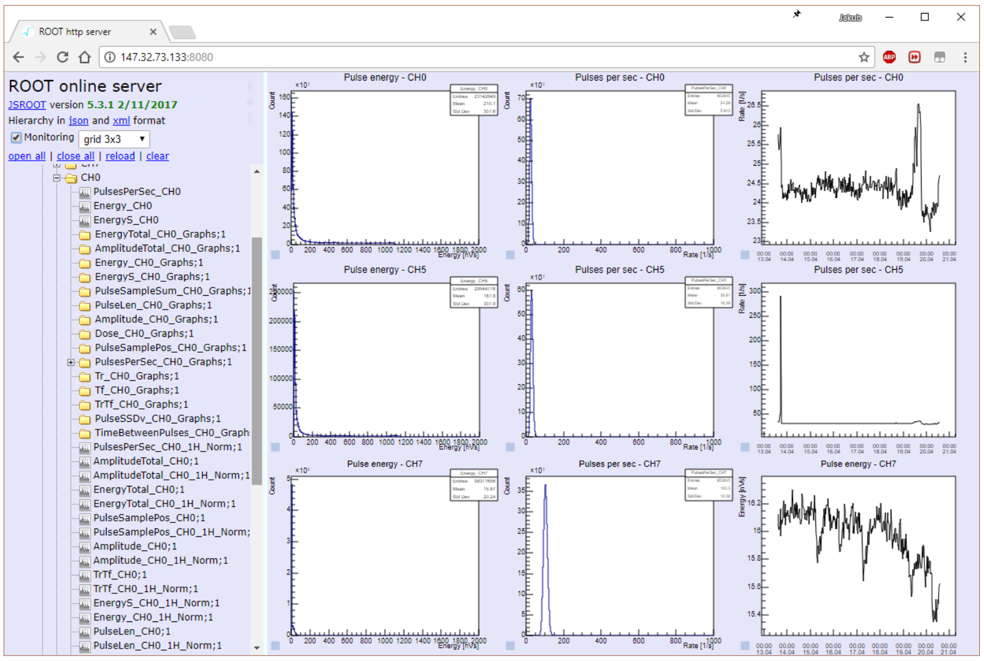


Figure 7-2 JSROOT web interface

7.1.4 Command and control

The JSROOT interface allows starting and stopping of NWVME. The SwCoinc connects to the control interface of the DAQ and issues the start/stop command.

7.2 Analyzer software architecture

SwCoinc program structure is shown on Figure 7-3. Different colors represent separate processing threads which communicate through asynchronous message queues.

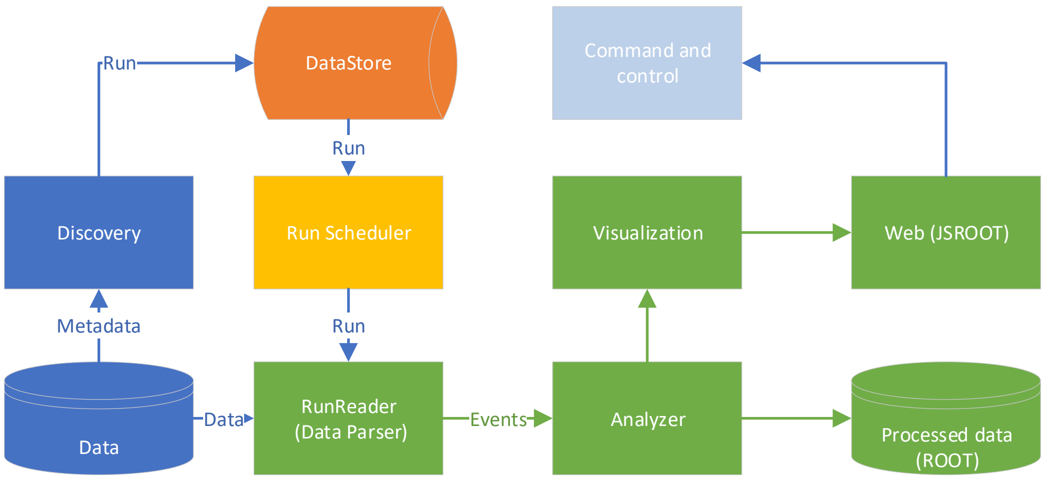


Figure 7-3 SwCoinc architecture

7.3 Analyzer software class overview

This chapter describes the most important C++ classes in the SwCoinc system.

7.3.1 Config

A global singleton class accessible via the GLOBALS macro holding configuration of the SwCoinc system. The configuration can be set via command line arguments and an XML configuration file.

7.3.2 Log

Uses the boost::log classes to allow logging of system events to console, a file or JSROOT console. There are six logging levels supported by – fatal, error, warning, info, debug and trace and each output channel can be set to a different severity level.

7.3.3 RootServer

Starts and stops the ROOT HTTP server. To allow interaction between the JSROOT interface and SwCoinc, it was necessary to use the ROOT's embedded LLVM compiler. A C++ statement with a functional pointer to required SwCoinc function is compiled using the LLVM compiler which can then called by the JSROOT.

```
char buf[256];
sprintf(buf, "typedef void (*fptr)(); fptr setExitFlag = (fptr) %pLL;", &setExitFlag);
gROOT->ProcessLine(buf);

m_serv->RegisterCommand("/ExitRoot", "setExitFlag();", "rootsys/icons/ed_delete.png");
```

Listing 7-1 Connecting ROOT to SwCoinc

7.3.4 IThread

Derivable interface class implementing C++11 std::thread. Each thread executes lambda expressions from a message FIFO in an event loop. The messages are used for inter-thread communication. It also provides the Sync method thread synchronization based on the std::future and std::promise. Listing 7-3 shows part of the implementation detail and Listing 7-3 exemplifies how the message passing is used within the SwCoinc application.

```
typedef std::function<void()> Operation;
class IThread {
       …
       MessageQueue<Operation> m_msgQueue;
       …
}
bool IThread::Sync() {
       std::promise<bool> res;
       auto msg = [&]()
       {
              res.set_value(true);
       };
       m_msgQueue.Write(msg);
```

```
	return res.get_future().get();
}
void IThread::m_Run() {
	m_ThreadStart();
	while (!m_ThreadDone) {
		Operation op;
		if (m_msgQueue.TimeOutRead(op)) {
			op(); // Execute message
		}
	}
}
```

Listing 7-2 Partial IThread implementation

```
class DSScheduler : public IThread {…}
void DSScheduler::ScheduleTask(RunTask task) {
	auto msg = [this, task = std::move(task) ]() mutable
	{
		task.runreader->Execute(task.run, m_DataSink);
	};
	m_msgQueue.Write(std::move(msg));
}
```

Listing 7-3 Asynchronous message passing use example

7.3.5 DataStore

This class keeps track of all data sources and Run instances. It is executed in a separate thread. Spawns the Discoverer and Scheduler threads. Routes messages between the data source discovery thread and scheduling threads.

7.3.6 Scheduler

The scheduler sequentially executes available Runs by creating an instance of a RunReader. For each run a new thread is created. In the future, it should be possible to extend the scheduler to be able to run multiple tasks concurrently. However, at this time ROOT is not completely thread-safe and causes issues whenever one ROOT file is accessed by multiple threads.

7.3.7 Event

The Event class contains measurement data which is processed through the system. The parsers transform raw source data into an instance of this class. Listing 6-1 shows the declaration of this class.

```
class Event
{
public:
	enum class EventTypes
	{
		VME_DATA, TRC_DATA, ADC_DATA, ENERGY_DATA, COINCIDENCE_DATA
	};

	Run *run;
	EventTypes Type;
	Time EventTime = Time::Zero();
	Time LastTime1 = Time::Zero();
	Time LastTime2 = Time::Zero();
	uint64_t EventTimestamp; // TS in nanoseconds
	uint64_t EventN = 0; // Event ID
```

```
        bool hasADC = false;
        bool hasPulse = false;
        bool hasCoincidence = false;
        bool hasVMECrate = false;
        int CoincidenceOrder = 0; // depth of coincidences
        int MergedEvents = 0; // number of overlapped ADC trigger windows merged
        int MergeDelta = 0; // time delta of overlapped triggers
        int CrateEventPos = 0; // position of event inside the crate event
        PulseEventData Pulse; // Energy, from Pulse Extractor
        VMECrateData VME; // TS of start and stop of IRQ handler
        std::map<int, EventChannelData> ChannelData; // ADC data
        std::vector<std::shared_ptr<Event>> CoincidenceData; // events in coinc

        int PeakPulseAmplitude();
        bool hasCoincidenceChannels(const std::vector<int>& channels);
        bool CoincidenceChannelsEqualTo(const std::vector<int>& channels);
        bool hasCoincidenceMap(const std::vector<std::vector<int>>& map);
};
```

Listing 7-4 The Event class

7.3.7.1 VMECrateData

Instance of this class is created for each VME crate event and contains Unix timestamps of the beginning and the end of a single VME IRQ handling.

7.3.7.2 ADCEventData

Contains V1740 event data, both raw ADC samples, calculated sample voltages, trigger threshold, trigger polarity and the sampling period.

7.3.7.3 PulseEventData

The PulseExtractor filter creates a new Event with PulseEventData filled. Such Event keeps pointer to the original ADC Event.

7.3.7.4 CoincidenceData

Event output by the Coincidence filter contains a vector of all Events which are in a defined coincidence window. The CoincidenceOrder member variable holds the level of nesting

7.3.8 DataSource

Interface base class for classes which originate Events. Implements overridable virtual methods listed in Table 6-1. Any class which instantiates a DataSource should call the SetSink() method to set the sink of the source.

Method	Time of call by implementing method
DataStart	At the start of data processing
RunStart	At the start of a single run processing

SendEvent	When an Event to be send is generated
RunEnd	At the end of a single run processing
DataEnd	At the end of data processing

Table 7-1 DataSource interface method

7.3.9 DataSink

Interface base class for classes which process Events. Methods are listed in Table 7-2. The update method is called to re-generate online monitoring graphs every one real time second.

Method	Time of call by a DataSource
DataStart	At the start of data processing
RunStart	At the start of a single run processing
ProcessEvent	When an Event to be send to be processed
RunEnd	At the end of a single run processing
DataEnd	At the end of data processing
Update	Periodically, should run time intensive calculations

Table 7-2 DataSink interface method

7.3.10 Filter and LambdaFilter

The Filter class is a base for classes which act both as a sink and a source. Inheriti from both DataSource and DataSink. The LambdaFilter is a filter which takes a C++ lambda as a parameter to be applied on each Event.

7.3.11 PulseExtractor

Pulse extractor is a Filter which scans the ADC data and searches for spectroscopic pulses, rejects pileups, calculates their energy in nVs and produces an Event for each individual pulse. First, a moving average of the samples is calculated until the delta between the sample and the average is higher than configured threshold. The energy is then calculated as a sum of samples with baseline subtracted. Figure 7-4 shows an example extracted pulse in red as drawn by the PrintADCPulses function. The temporal order of events is preserved by first sorting the events by their timestamps before sending them down the processing chain.

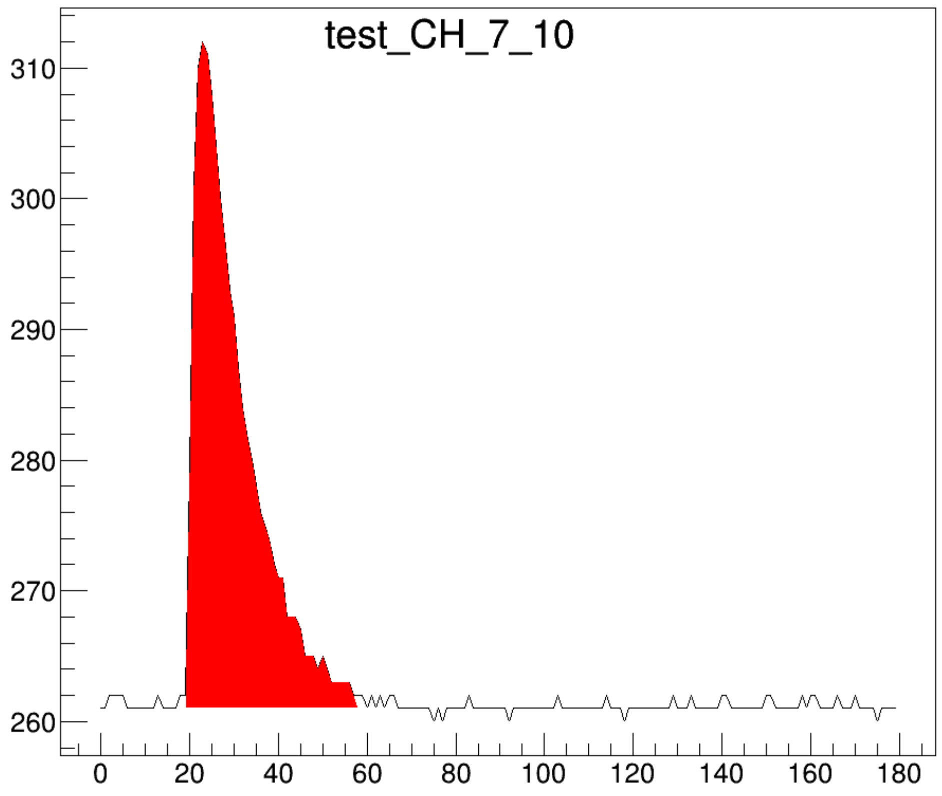


Figure 7-4 Example extracted pulse (ADC value vs sample)

7.3.12 Coincidence

This filter finds events which are in a coincidence. New events are added to a FIFO of configured duration. Before expiring old events from the FIFO, the number of events is checked and if higher than specified threshold is reached a new coincidence Event is generated. Such event contains pointers to the individual events. It is possible to perform coincidence of any type of event, enabling multi-level coincidences as required by the anti-neutrino detector.

7.3.13 DataParser and DataParserFactory

Root class for data parsers. Child classes are instantiated using the ParserFactory interface. Each parser type uses the ParserInfo (Listing 7-5) structure to register (Listing 7-6) with the factory. The Discovery classes then use this data to search for files. Two types of parsers are supported – STREAM_BASED and FILE_BASED, which differ in behavior during data parsing.

```
struct ParserInfo
{
	DataParser::DataParserType Type;	// STREAM or FILE
	std::string ID;				// used for command arguments
	std::string Name;			// Human readable name
	bool supportsZip;			// If .zip files supported
	std::string FileRegex;			// Matches file extension
	std::string ConfigRegex;		// Matches file extension
	std::string ChannelRegex;		// SubMatches channel ID (if any)
```

```
        std::string FileNumberRegex;        // SubMatches file number (if any)
        int AdcMin;
        int AdcMax;
};
```

Listing 7-5 ParserInfo structure

```
// In VMERawDataParser.cpp
struct ParserInfo VMERawDataParser::m_ParserInfo
{
        DataParserType::STREAM_BASED,
                "VME_R",                // ID
                "VME Hons Raw",         // Human Readable name
                false,                  // zip supported
                ".*\\.VME_R",           // Matches file extension
                ".*\\.XML",             // Matches configuration file
                "",                     // SubMatches channel ID
                ".*-(\\d+)_\\d+\\.VME_R", // SubMatches file number,
                0,                      // adcmin
                4096                    // adcmax
};
bool VMERawDataParser::isRegistered =
DataParserFactory::inst().RegisterParserType<VMERawDataParser>(
VMERawDataParser::m_ParserInfo);
```

Listing 7-6 Parser registration

The stream parse function expects to be called each time with a new buffer of data and keeps internal state. File based (TRC) parsers are called once with a buffer containing entire source data. Such is the case for TRC files decompressed from a ZIP file to in-memory buffer.

7.3.13.1 HonsVMEXMLParser

Uses the boost::property_tree classes to read the XML configuration file of NWVME. Looks for the base VME address of the V1740 and its registers settings. Extracts enabled input ADC channels, their offsets, sampling window length and trigger polarity.

7.3.13.2 TrcDataParser and Trc

The Trc class parses data created by LeCroy WaveRunner oscilloscope wave capture and supports both single shot and segmented files. The TrcDataParser encapsulates traces in an Event class. As each TRC file contains samples from one channel a multi channel measurement requires multiple instances of the classes.

7.3.13.3 VMERawDataParser

Parses the stream data generated by the NWVME. The state-machine parser of the data file has following states which transition based on the parsed 32-bit data word.

- IDLE – after program startup
- RUN – RUN_HEADER detected, present at the beginning of the first file
- CRATE – CRATE_HEADER, denotes start of a crate event - generated whenever the VME crate asserts an IRQ

- TIME1 – Unix timestamp of the IRQ wake-up time
- TIME2 – Unix TS after VME crate handling is done
- V1740 – A V1740 event detected in the data

The V1740 supports overlapped trigger mode operation where instead of ignoring triggers occurring during active sampling window it extends the sampling window. It is possible to detect this only by checking the next V1740 event whether it has correct a timestamp occurring at the end of the previous window and may be shorter than the configured window size. Therefore, the VMERawDataParser has to store every event in a queue and send the Event only for analysis only if the following one is unrelated.

The V1740 event parser decodes the bit packed 12-bit ADC data and converts to the Event class and its state-machine is based on the word position of the V1740 event.

7.3.14 Discovery

Root class for source data gathering. Child classes detailed below can be instantiated via the DiscoveryFactory class. One virtual method is exposed - `ScanURI(std::string uri)`, where URI is the target of the data source scanning process.

7.3.14.1 FileDiscovery

Implements the ScanURI method for file system directories or files. Directories are scanned recursively for data and configuration files registered by the data parsers and sorted using boost::regexp. For each data type in a directory or an archive one instance of the Run class is created and added to the DataStore instance.

7.3.14.2 NWVMELogDiscovery

Implements online monitoring of the Hons VME DAQ process log file. When started, with the DAQ copies the configuration file to an output directory, creates a log file. For each Run, a new timestamped data directory is created. The NWVMELogDiscovery class detects the path of the logfile, parses configuration file location and data files.

The class creates a new thread and scans the /proc filesystem every second for a process with matching the executable path specified as SwCoinc command line argument. If found, the processes’ open file descriptor list is scanned for the DAQ log file.

A inotify watch for closing and write access is created for the log file. The file is read in a loop and the state of the DAQ is detected as follows:

- STARTUP – the DAQ is started
- READY - VME crate is configured and the DAQ ready for measurement
- RUNNING – current measurement is running, look for data files
- QUIT – VME DAQ terminated

If the DAQ is in the RUNNING state when the end of the file is reached, a new instance of the Run class is created and added to the DataStore. Any new DAQ data files are then added.

7.3.15 Run class

The Run class encapsulates information of a single measurement. Contains list of data sources and associated configuration. It is able to provide certain data about the measurement, if provided by the data source type, such as length of the measurement, time of first and last events and number of events.

7.3.16 RunReader

A RunReader class instance describes a task. Takes a set of input data, creates instances of parsers, opens the files, reads the data and calls the parsing functions. There is a hierarchy of classes derived from the RunReader, each providing a different way of reading input data.

7.3.16.1 FileStreamRunReader

Simple file based reader for stream based data, such as offline VME, which iterates through the list of all files and reads the data block by block.

7.3.16.2 InotifyFileStreamRunReader

Derived from FileStreamRunReader. Used for online NWVME processing. The DAQ continuously adds new VME events into the output file. Therefore, whenever this RunReader encounters an end of file it waits until a new data is appended, the file is closed by the DAQ, new data file is detected or the acquisition stops. It uses the Linux inotify_watch() to detect file changes.

7.3.16.3 MultiFileRunReader

Used for reading oscilloscope files. Reads contents of entire input file into memory and calls the parser function and does so for all channel files of a single acquisition. From each input channel it reads exactly one event to align the events.

7.3.16.4 ZipMultiFileRunReader

Derived from MultiFileRunReader, but with the ability to decompress files from ZIP archives using the libzip library. Ability to read ZIP files is useful for handling tens thousands of files if a single file per trace mode of the oscilloscope is used.

7.3.16.5 NetStreamRunReader

Reads stream data from a network TCP socket.

7.3.17 VBase

Visualisation base class which holds the TCanvas ROOT object. The Export method causes the canvas to be saved as a PNG file to the output directory. Overloaded virtual getters isPNGLoggable, isPNGLoggable2D, isPNGLoggable2DZ control whether the graphs are additionally saved with (Y, YZ, XYZ) axes set to logarithmic scale. The Update method re-draws the TCanvas if the isDirty member variable is set.

7.3.18 VGraph, VGraphErrors

Holds a TGraph object and the Export function writes the graph to the output ROOT file. VGraphErrors holds a TGraphErrors object for plotting with error bars.

7.3.19 VH1D, VH2D

Its constructor creates the TH1D or TH2D in specified folder of the output ROOT file. Sets histogram title and axes labels. Provide CloneTagScale method for creating a scaled copy of the histogram and is used for generating normalized graphs during image generation at the end of Run processing. The Fill method adds a new value to the histogram.

7.3.20 VTimeGraph

A TGraph encapsulation for plotting of time series data with automatic data reduction. Its constructor has time start, time span, time step arguments and data reduction option.

Possible data reduction and visualization options are specified as a bit mas of the GraphOpts enum class as specified in Table 7-3.

Option	**Description**
None	Plot last value during each time span
SUM	Plot a sum of values during each time span.
AVG	Plot an average of values during each time span.
MAX	Plot the maximum of values during each time span.

MIN	Plot the minimum of values during each time span.
SHOW_MINMAX	Show average and draw minimum and maximum as error bars.
ROLLING	Used for “show last 3 hours” graph. Points older than span are deleted.
RATE	Number of calls to Fill() during each span is added to the graph.

Table 7-3 VTimeGraph options

7.3.21 TimeSeriesHistoGraph

A visualization class which combines the VH1D and VTimeGraph. The Fill member method takes a time point and value arguments. It can automatically generate multiple time-dependent histograms and time series plots. For example, when plotting the count rate of a spectroscopic channel, it can generate a histogram for every hour, day and week in a separate VH1D objects and plot a graph of the rate vs time. Each created object is saved as a separate PNG file during export and as a separate object in the output ROOT file.

7.3.22 MultiVH2D

A template class for creating std::map of VH2D objects. The type of the index is specified as a template parameter. It creates a new VH2D object upon call to Fill() with a new index. The root file path, name of the new instance are formatted using boost::str and boost::format to replace placeholder tokens in path, name and title constructor with an index converted to string.

Listing 6-1 shows an example instantiation of the MultiVH2D object with an int type index. The %1% token in the fields is replaced with the channel number.

```
// in DataSink::RunStart
m_VH2D2.emplace("PulseAmplitudeVsEnergy",
 std::make_unique<MultiVH2D<int>>(run,
  "/Pulse/CH%1%",                          // ROOT file folder path
  "PulseAmplitudeVsEnergy_CH%1%",          // Graph Name
  "Pulse Amplitude vs Energy- CH%1%",      // Graph Title
  "ADC", "Energy [nVs]",                   // Axes labels
  0, 4096, 4096,                           // X axis minimum, maximum, number of bins
  0, EMAX, 100                             // Y axis minimum, maximum, number of bins
));

// in DataSink::ProcessEvent
m_VH2D2["PulseAmplitudeVsEnergy"]->Fill(ev->Pulse.Channel, ev->Pulse.PeakAmplitude,
ev->Pulse.Energy);
```

Listing 7-7 Example of MultiVH2D instantiation

7.3.23 MultiTimeSeriesHistoGraph

A template class for creating std::map of TimeSeriesHistoGraph objects used the same way as MultiVH2D class. Listing 7-8, taken from the scube analyzer, shows an example of an instance of this class used to generate separate spectra for each pulse of each 100 ns spatial coincidence of each 60 us temporal coincidence.

```
// in DataSink::RunStart

m_TSG3.emplace("60us_DANSS_Energy",
 std::make_unique<MultiTimeSeriesHistoGraph<std::tuple<int,int>>>(run,
  "/Neutrino/P%1%/CH%2%",           // ROOT file folder path,
                                    // %1% and %2% are taken from std::tuple values
  "60us_N_Energy_P%1%CH%2%",        // Graph Name
  "60 us coincidence Pulse energy - P%1% CH%2%", // Title: Pulse idx.1 Channel idx.2
  "Energy [nVs]",                   // X axis label
  0, EMAX, 4096,                    // X axis min, max and number of bins
  defTSGopts,                       // Generating options
  defVGopts                         // TimeSeriesGraph options and
));

// in DataSink::ProcessEvent
for (auto const &ced : ev->CoincidenceData)
 for (auto const &e : ced->CoincidenceData)
 {
   m_TSG3["60us_DANSS_Energy"]->Fill(
    std::make_tuple(pulse, e->Pulse.Channel),  // index, pulse is n-th pulse in the
                                               // coincidence window
    ced->EventTime,
    e->Pulse.Energy
   );
   pulse++;
}
```

Listing 7-8 MultiTimeSeriesHistoGraph Example

7.3.24 Analyzer and AnalyzerFactory

The Analyzer base class is a DataSink which is named and registered with the AnalyzerFactory. The user can specify which analyzer is used for data analysis either on command line or in the configuration file. SwCoinc contains two analyzers: adcprint and scube_vme.

7.3.25 ADCPrinter

ADCPrinter class provides a simple analyzer which outputs each ADC Event as a PNG image and Text file output. The code for the analyzer is shown in Listing 7-9 and Listing 7-10.

```
#pragma once
#include <memory>
#include "Event.hpp"
#include "DataSink/DataSink.hpp"

class ADCPrinter : public DataSink
{
public:
       ADCPrinter(std::string name) : DataSink(name) {}
       virtual void ProcessEvent(const std::shared_ptr<Event>& ev);
};
```

Listing 7-9 ADCPrinter.hpp

```
#include "Utility/PrintADCEvent.hpp"
#include "Analyzer2/AnalyzerFactory.hpp"
```

```
#include "ADCPrinter.hpp"
static struct AnalyzerInfo AnalyzerInfo_ADCPrint
{
	"adcprint", // Name used as a command line parameter
	"Exports ADC Events as graphs", // Description
	[]() { return std::make_shared<ADCPrinter>("adcprinter"); } // Factory allocator
};
static bool isRegistered[[gnu::unused]] =
AnalyzerFactory::inst().Register(&AnalyzerInfo_ADCPrint);

void ADCPrinter::ProcessEvent(const std::shared_ptr<Event>& ev)
{
	if (!ev->hasADC)
		return;
	PrintADCEvent("ADC", ev);    // Output Event as PNG
	ExportADCEvent("ADC", ev);   // Output Event as TXT
}
```

Listing 7-10 ADCPrinter.cpp

7.3.26 Scube VME Analyzer (scube_vme)

Implements the Inverse-Beta Decay coincidence search method. Pulses extracted from the raw ADC data are checked for the Detector Pulses using a 100-nanosecond coincidence filter. These coincidence events are run through another 60 microsecond coincidence window. The scube_vme analyzer allocator function is shown in Listing 7-11.

```
static std::shared_ptr<DataSink> scube_vme()
{
  DataSink *v = new DataSinkVector(
  {
    new VMESystemGraphs("VMEGraphs"),
    new ADCEventGraphs("ADCEvents"),
    new PulseExtractor(new DataSinkVector( {
      new PulseGraphs("PulseGraphs"),
      new Coincidence(Duration::OneNanoSecond() * 100, 1, 0, new DataSinkVector({
        new SubCoincidenceGraphs("SubCoincidence"),
        new Coincidence(Duration::OneMicroSecond() * 60, 2, 0, new DataSinkVector({
          new CoincidenceGraphs("Coincidence")
        }))
      }))
    })),
  });
  std::shared_ptr<DataSink> sv(v);
  return sv;
}
```

Listing 7-11 scube_vme allocator function

8 DANSS Firmware Results

The measurement is done in coincidence mode controlled by the V1495. Events from active shielding are collected by second V1495 controlled via serial protocol. Due to parasitic pulses which sometimes occur approximately 500 ns after a pulse, the controlling V1495 is set to ignore any pulse closer than 2 µs. The main coincidence window lasts up to 80 µs. Statistical data from a single run is shown in Table 8-1.

Run length	88 020 s
Block length	60 s
Crate events	18 398 866
Average time between IRQs	4.8 ms
Average IRQ handling duration	112 us
Total handling time	2068 s
Total handling time [%]	2.4%

Table 8-1 DAQ Statistics for one run with QDC only

Figure 8-1 shows the distribution of number of events per second, the discrete nature of the graph is due to the way this statistics is collected by the DAQ system. The number of events per second shows that the peak count rate has been about 550 events per second.

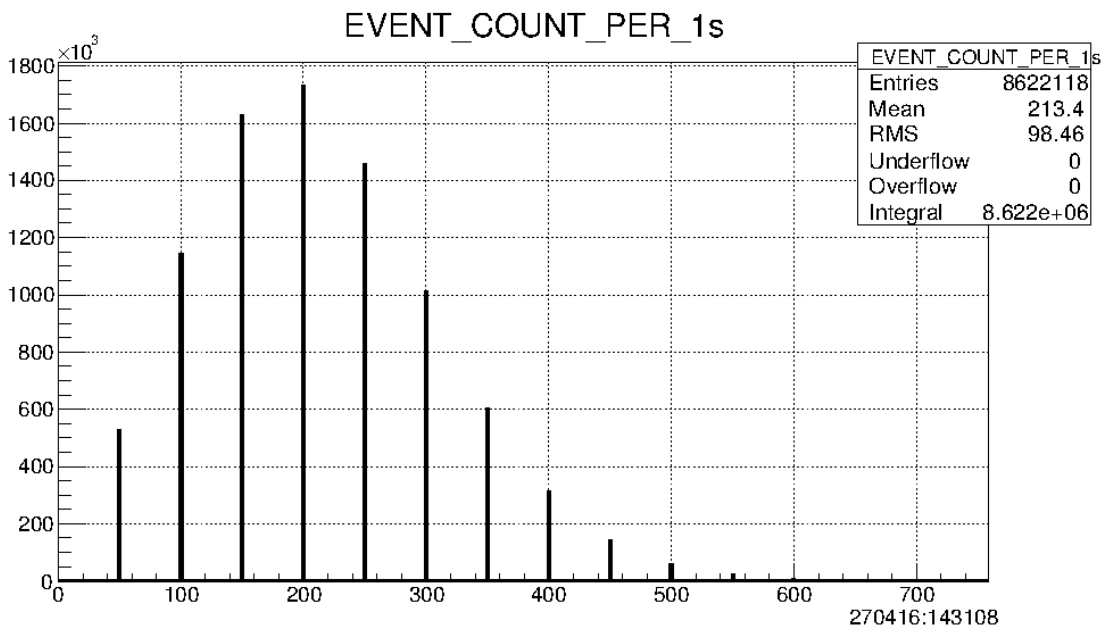


Figure 8-1 Average number of events per second in one run (Counts vs Counts per second)

Time required for crate readout is shown in Figure 8-2. The average time to service an IRQ is 115 µs. First, the source of the interrupt is checked and appropriate VME board status registers are read. Depending on the source, either data are read from QDCs using CBLT and the control V1495, or from active shielding V1495. Because the QDC event does not contain a timestamp, the acquisition process is blocked during the readout process to assure that only related data are put into the crate event. Despite that, it can be seen that the system has been blocked only for 2.4% of the run duration.

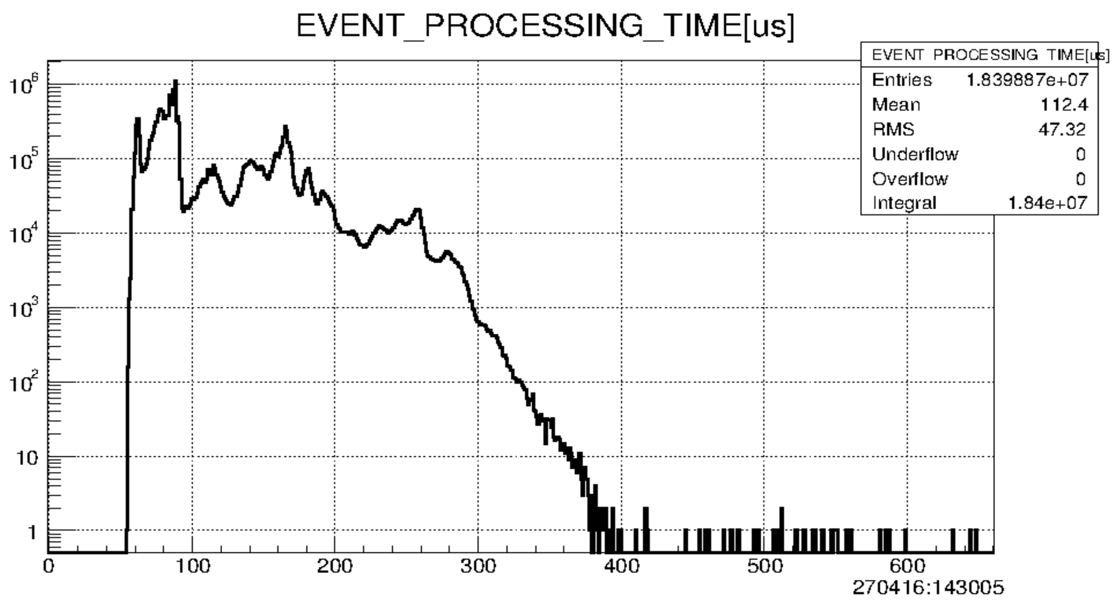


Figure 8-2 IRQ handling time with QDCs only (Counts vs time in us)

Figure 8-3 and Figure 8-4 show typical spectra of randomly selected PMT prompt and delayed channels during the 24-hour run. Both spectra show the cosmic muon peak and have a range of about 56 MeV. The sharp peak at the beginning is the result of all detector channels being converted when any channel triggers the DAQ and is therefore a peak of a zero charge.

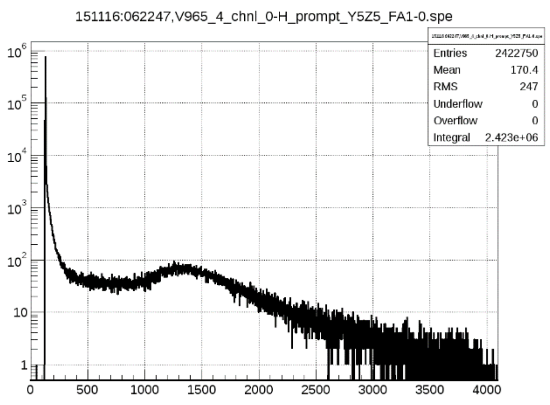


Figure 8-3 Prompt PMT Channel spectrum

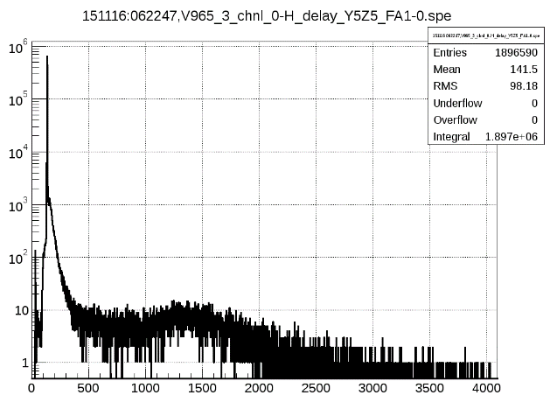


Figure 8-4 Delayed PMT Channel spectrum

Distribution of multiplicities of section pulses in prompt and delayed DANSS pulses and their correlation are shown in Figure 8-5. Both axes contain the number of active detector sections (50 strips connected to 1 PMT) in the DANSS pulses.

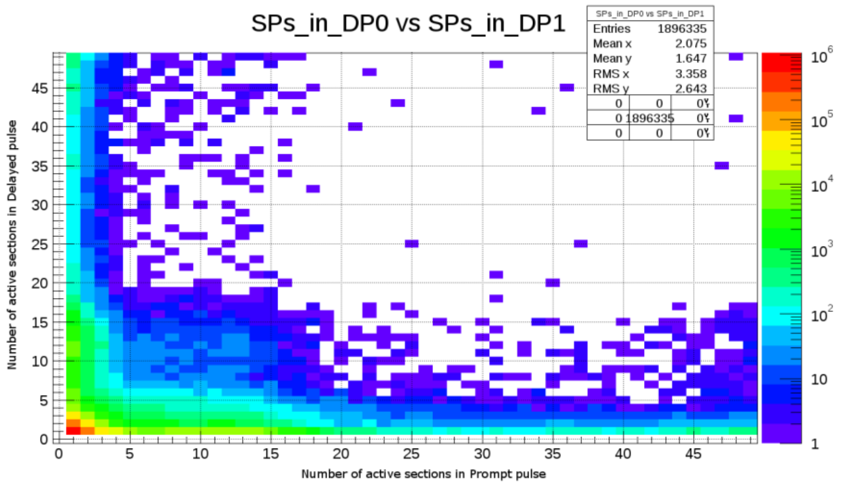


Figure 8-5 Number of active PMT channels in prompt vs delayed DANSS pulses

The spatial correlation of prompt DP and delayed DP are shown in Figure 8-6. X and Y axes contain the PMT channel number. These data are collected from hardware triggers by the V1495 which allows up to 64 logical inputs and only 50 PMTs are connected, explaining the gaps in the data. The figure also shows a hardware problem which was detected in channel 34 analog frontend causing ringing and higher-than-normal coincidence rate. Generally, the dispersion of the parameters is caused by non-uniformity of the shielding (mechanical, position on the lifting mechanism), dispersion of strip parameters, trigger threshold levels and differences in noise levels [3].

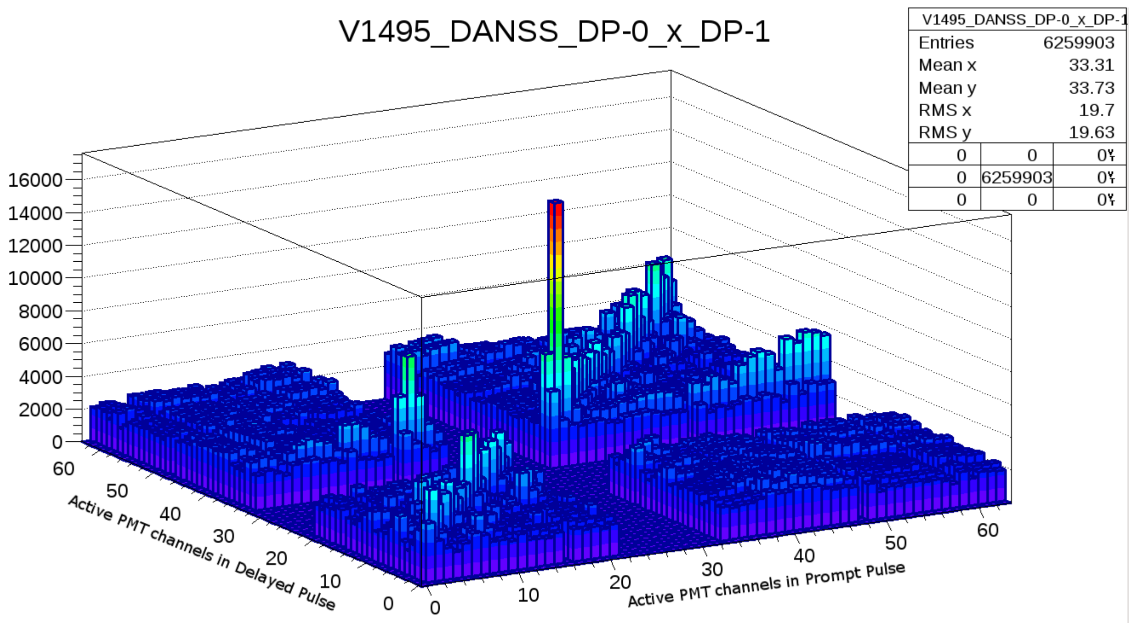


Figure 8-6 Active PMT channels in prompt vs delayed pulses

8.1 Digitizer performance results

Another spectrometer based on the V1740 digitizer is connected to the detector in parallel. Many tests were performed on a small part of the detector in the laboratory. The digitizer capture length is set to 288 ns, or 18 samples. Figure 8-7 and Figure 8-8 show results of pulse digitization capturing interaction of a testing scintillator connected to a DANSS AFE with cosmic muons in laboratory setting.

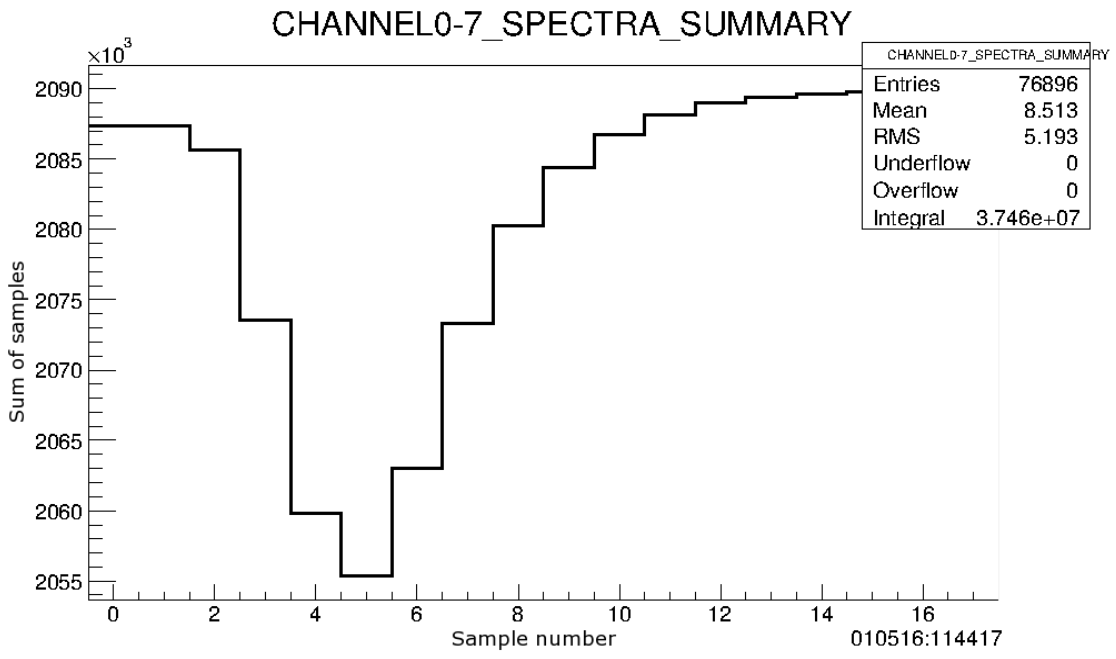


Figure 8-7 Sum of all digitized waveforms

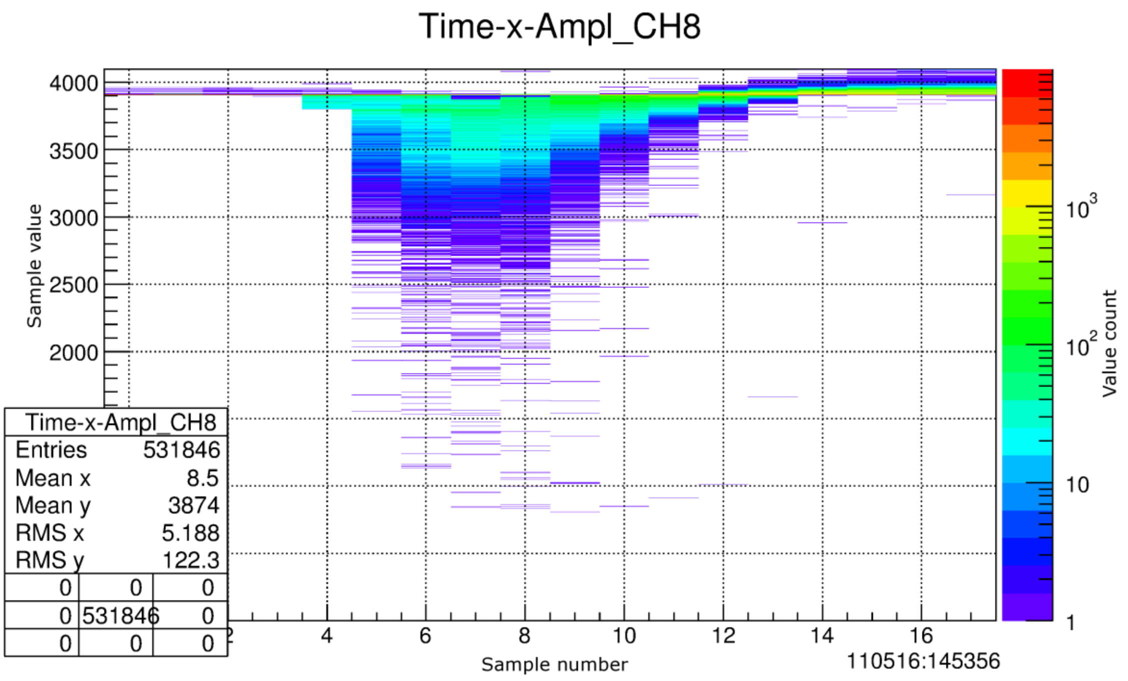


Figure 8-8 Digitizer samples of natural background

Figure 8-9 show spectrum of natural background in the laboratory which was obtained by online charge integration of the digitized window.

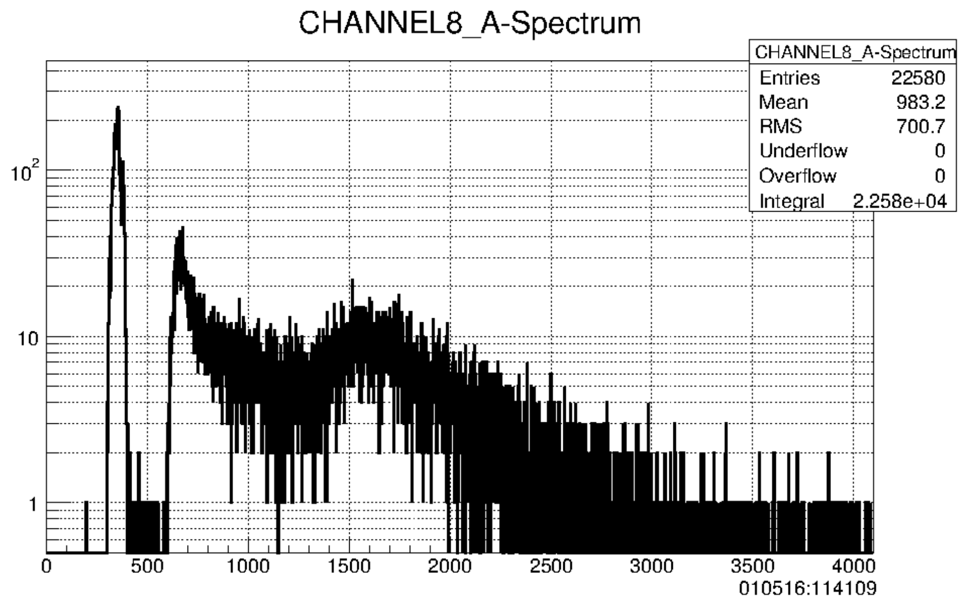


Figure 8-9 Laboratory background spectrum (Counts vs ADC value)

The digitizer is configured with all of its 64 input channels enabled. In this case 64 * 18 samples have to be read during each IRQ. The average readout time is about 120 µs (Figure 8-10), thanks to the digitizer’s compact event structure and support for VME block read (BLT). Like in the QDC spectrometer case it is calculated as a difference between Linux system timestamp just after the acquisition process is woken up by IRQ and a call to wait for further interrupts. The peak around 20 μs is caused when V1740 asserts its interrupt line but the queried status register reports no available event prepared to be read out.

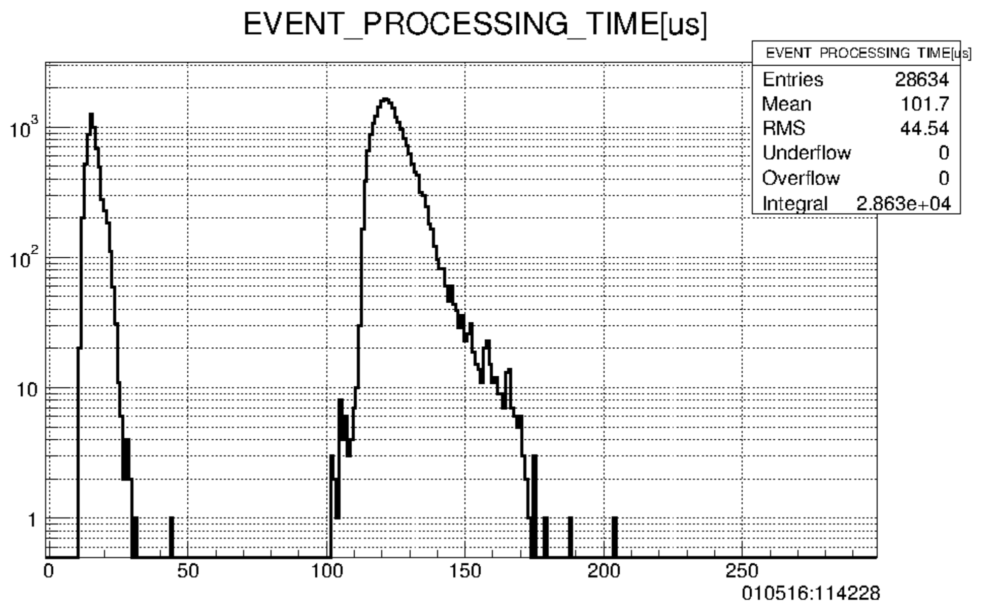


Figure 8-10 Digitizer DAQ readout time (Counts vs time in us)

To ensure practically zero dead time the V1740 is configured to work with 8 FIFO buffers for each channel. When the digitizer internally triggers and captures samples, it generates an interrupt and immediately is ready to capture next event. Given the average count rate of DANSS detector and its active shielding is about 2500 pulses per second and the average time it takes to read one event is about 120 μs, it is clear that it is indeed possible to capture all events in the "event by event" mode.

The V1740 is configured to not allow interleaving triggers. That means, that as soon as any channel triggers internal acquisition, other triggers are ignored and samples from all channels are recorded relative to start of the acquisition window. Pulses which arrive later can be sampled incompletely, however the 288 ns long sampling window and associated 50 ns DANSS window provides for correct sampling of all PMT pulses forming the DANSS Pulse and compatibility of charge integration.

8.2 Combined spectrometer results

The configuration of the DANSS DAQ operating as of fall 2016 contains both the QDC and Digitizer spectrometers operating together in a single crate. Both the QDC and Digitizer are controlled by one V1495

Figure 8-11 and Figure 8-12 show comparisons of a DANSS channel spectra obtained by QDC (black) and the Digitizer (red and green). The FADC energy is calculated as a sum of all samples within the window scaled by a constant.

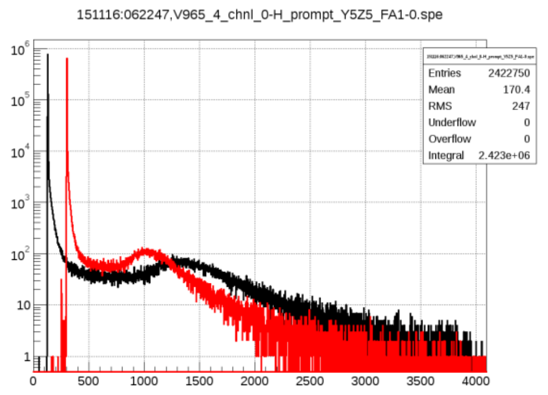


Figure 8-11 Prompt QDC and FADC spectra (Counts vs xDC value)

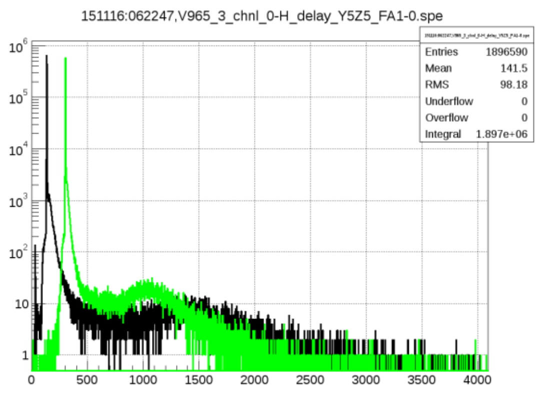


Figure 8-12 Delayed QDC and FADC spectra (Counts vs xDC value)

Figure 8-13 shows the total required interrupt handling time to read data from all 8 QDC, one V1740 digitizer, controlling V1495 and active shielding V1495. The required time varies with event complexity.

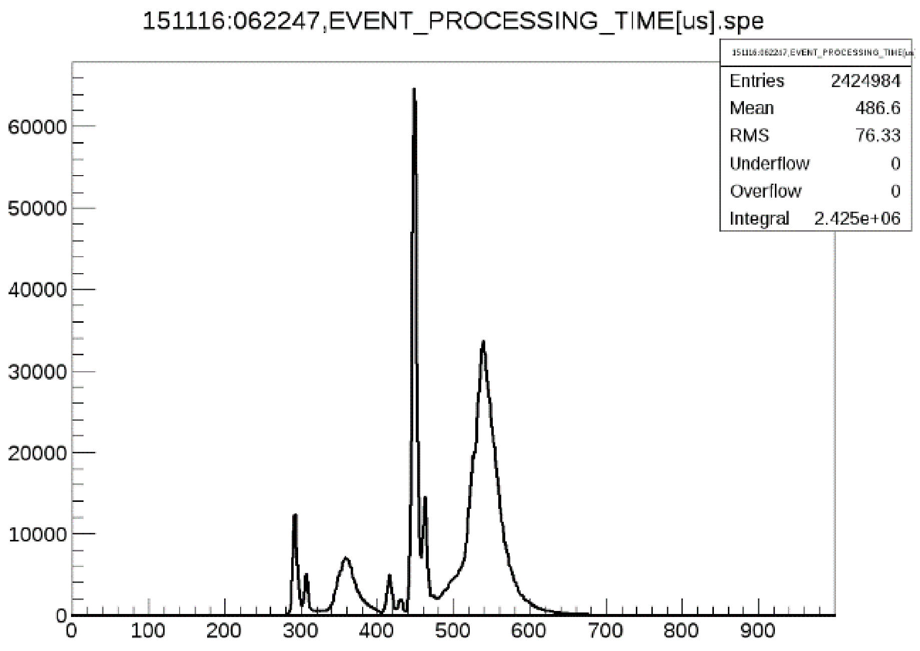


Figure 8-13 IRQ handling time QDC+FADC+Activeshielding (Counts vs time in us)

Table 8-2 shows the statistics of a 28 hour run.

Run time	102 780s
Average IRQ per second	86.70
Average IRQ handling duration	494 us

Total handling time	4398 s
Total handling time [%]	4.28%
Detector DANSS Pulses	101 544 038
Detector DANSS Pulses per second	988
Total active Shielding DANSS Pulses	238 883 876
IRQs	8 911 093
IRQs with one pulse events (≥8 active PMTs)	2 277 876

Table 8-2 DAQ Statistics for one run with QDC, Digitizer and active shielding

9 New data acquisition system proposal – Я^{3}DAQ

During the design and development of the S^3 detector a need for a new DAQ arose. Based on experience with the existing NWVME, the main requirements for the new system are easier configuration, quality of data logging and the capability of on-line reduction of data.

The NWVME XML configuration file is generic and consists of a sequence of raw VME register access commands. This has advantages and disadvantages. The advantage is that it can be used with any card and implement any read-out algorithm without changing the program sources. However, it is rather difficult for ordinary users to change specific configuration detail, such as enabling or disabling specific channel or trigger thresholds. In the DANSS experiment experience, members of the team rotate in the detector control room in the nuclear powerplant. This room has no internet access and therefore configuration cannot be done remotely by an experienced user.

The quality of data logging is important for measuring how well the data acquisition system performs. Information about hardware serial numbers, operating temperature should be logged at the system start and performance metrics, such as number of events per second, should be logged during the measurement. Both histograms and time evolution of the values should be stored. Histograms show the ranges of measured values and time evolution graphs allow the operators to notice abnormal behavior and possibly correlate them with external events (such as hardware failure, time of day, people in the room, etc.). The quality of data logging is also essential for estimation of measurement dead-time.

The third requirement – on-line reduction of data comes from the fact that the DANSS experience has shown the need to collect full spectroscopic spectra from all channels to characterize the detector and hardware triggering (as described previously) removed that possibility. The new S^3 DAQ is based on Flash ADC converters with QDC firmware which allows collection of all pulses without any dead-time. However, to reduce the number of recorded events, on-line processing and filtering is needed. Detector events can be separated into three categories – events which are known to be interesting (such as calibration muons and events with IBD-like signature), events which are known to be not interesting (such as random single events) and can be discarded and events whose category is not known and demand further analysis. For discarded events a summary spectroscopic spectrum is generated.

9.1 Overall architecture

The DAQ should be divided into three conceptual parts – DAQ, the BOX and The View. This reduces the number of possible bugs, similar to the UNIX philosophy – set of small utilities doing one task. The DAQ can be thought as an evolution of the NWVME, the BOX as an evolution of the software processing system described previously in section chapter 7.

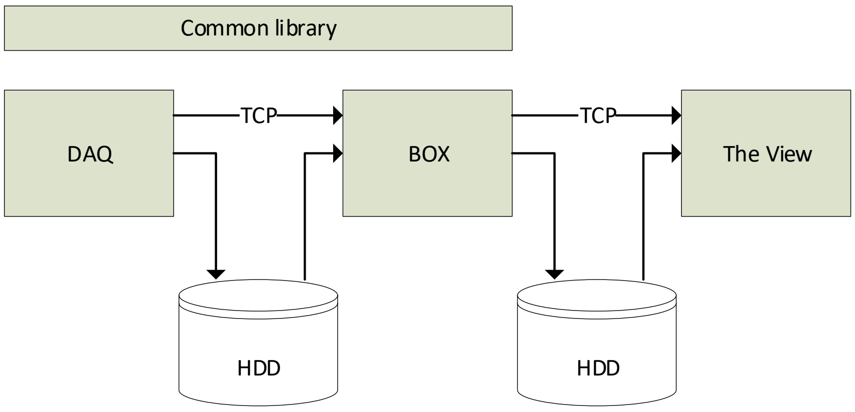


Figure 9-1 DAQ Architecture

The DAQ application is responsible for hardware configuration, operation and data readout. It is instrumented for data quality measurement – such as events per second, buffer occupancy and latency measurement. Data can be stored to disk or made available for network TCP clients. For remote clients a control socket, measurement configuration is also available. During hardware initialization the DAQ needs to test communication with the hardware, verify that the configured card type matches with the actual card and record card information – hardware and firmware revision.

The BOX is responsible for data reduction. Data can come either from a directory structure or network TCP connection. Incoming events from multiple sources (channels, crates, slow-control) is chronologically ordered and passed through user configured set of filters. The filters can be based on temporal or spatial coincidences, energies or channel sources (such as detector, veto, slow control). Processed output is stored in the same format as the source data allowing recursive operation – ie, basic on-line filtering done immediately without writing full measured data to storage and more intensive off-line processing later.

Figure 9-2 shows the block diagram of the proposed BOX – The Soft QDC part is used when reading events generated by the Waveform recording firmware. The input of the Time Ordering Queue is an n-tuple [TimeStamp, Channel, Energy]. Each channel can have a defined class –

in case of S^3 detector it is detector, gamma catcher and the active veto allowing software trigger definition.

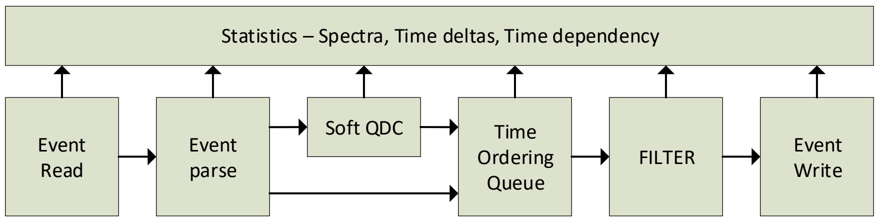


Figure 9-2 The BOX block diagram

Each logical block of the BOX generates its own quality of data statistics and instrumentation. Histograms of time delta between adjacent event are used to spot non-random events and the time dependency graphs are generated in per-hour, per-day and weekly.

The View is designed as a frontend for the user to view the data. Most likely it will be built using Python programming language. Allows the user to view graphs, spectra and do analysis.

Networked operation allows using less powerful (i.e. less generated heat) computer in the airconditioned detector room and have the filtering computer in the control room.

9.2 Types of supported VME Cards

Traditionally, the DLNP JINR has used spectrometers based on CAMAC standard and CAEN QDC converters. It has been decided that new spectrometers, not only for the S^3 detectors will be based on digitizers exclusively as they allow measurement with zero dead time. They are offered with two major kinds of firmware – waveform recording and charge to digital (QDC). It has been measured that a VME Crate with MBLT readout (50MB/s) and one V1725 digitizer (16 channels, 250MSPS, DPP-PSD firmware) allows up to 4 million pulses to be read out every second.

9.2.1 V1495 with custom firmware

The DAQ supports the Gate Generator as described in sectio 6.5 of this thesis.

9.2.2 CAEN Digitizer with ADC waveform recording firmware

These digitizers use analog to digital converters to continuously sample all input channels and store samples in a circular buffer. Whenever a trigger occurs, configured pre and post samples (acquisition window) from all enabled input channels are stored in an event buffer. The internal memory can be divided to store multiple events allowing acquisition without dead-time as long as the memory is not full. Depending on the model, the number of samples stored ranges from 192k to 1.5M samples per channel.

Trigger can be sourced either from external connection (LVDS/LEMO), generated internally by absolute under/above threshold value of any channel or forced by a register write. It is possible to require minimum number of simultaneously triggering channels (coincidence window). Triggers occurring during active acquisition window can be either ignored or cause lengthening of the window, which is then divided into separate events with length up to the configured window length.

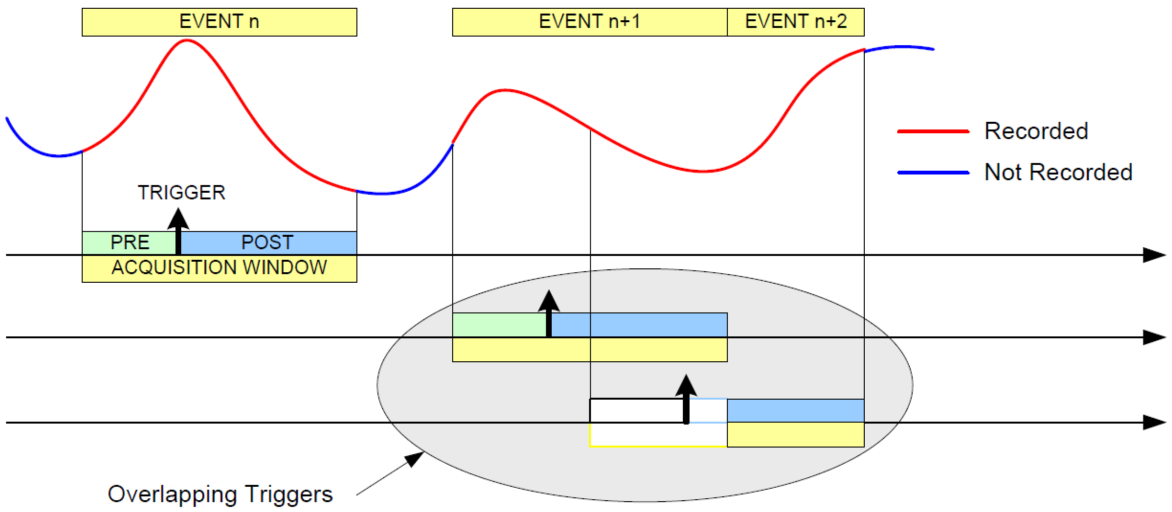


Figure 9-3 ADC Waveform recording firmware modes [22]

9.2.3 CAEN Digitizer with QDC firmware

Digitizers with the Digital Pulse Processing (DPP) firmwares continuously sample each input channel and calculate channel baseline (zero voltage). Whenever a relative threshold is crossed the charge is integrated (sum of relative values). Each channel triggers independently and data from non-triggering channels are not stored. Therefore, this firmware reduces the amount of data recorded and is much better suited for multichannel detectors with relatively sparse events - where only one or few channels out of all have interesting data. The format of the event is configurable and the pulse samples can be also recorded.

The Caen VME V1740D supports the basic DPP-QDC firmware where only one gate is present. Newer Digitizers (V1725, V1730, V1751) support the DPP-PSD firmware where for each pulse two sums are calculated – peak and peak with tail allowing pulse shape discrimination between neutrons and gammas by ratio of the short and long gate charges.

One disadvantage of this firmware is that the resulting VME events (called aggregates by CAEN) are not chronologically ordered and have to be sorted during processing.

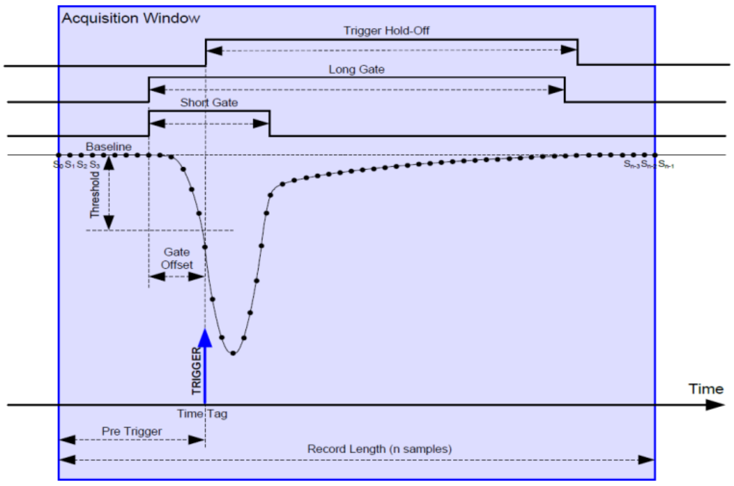


Figure 9-4 DPP-PSD Gates [23]

The DAQ has been tested with following CAEN VME Cards

- V1730 – 16 channel, 500MSPS, 14 bit, Waveform FW
- V1725 – 16 channel, 250MSPS, 14 bit, DPP-PSD FW
- V1740 – 64 channel, 62.5MSPS, 12 bit, Waveform FW
- V1740D – 64 channel, 62.5MSPS, 12 bit, DPP-QDC FW
- V1495 – FPGA card with the author’s Gate Generator FW

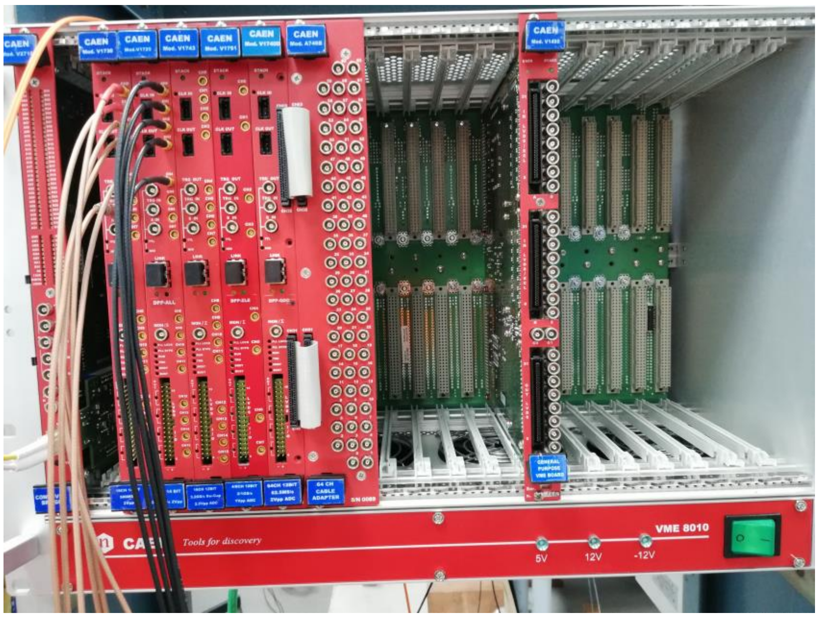


Figure 9-5 Test VME Crate with digitizers

9.3 Configuration

As opposed to the NWVME the new DAQ shall have knowledge about the features of specific hardware and express it in the configuration file in a human readable format. An example of setting channel input ranges and channel thresholds in NWVME is given in Listing 9-1.

```
<!-- Input range -->
<command name="CAENVME_WriteCycle" offset_32_address="0x1028" address_modifier="0x09"
data_width="32" input_value="0x0000" repeat="1"/>
<command name="DELAY" input_value="1000"/>
<command name="CAENVME_WriteCycle" offset_32_address="0x1128" address_modifier="0x09"
data_width="32" input_value="0x0000" repeat="1"/>

<!-- Trigger threshold -->
<command name="CAENVME_WriteCycle" offset_32_address="0x1080" address_modifier="0x09"
data_width="32" input_value="15700" repeat="1"/>
<command name="DELAY" input_value="1000"/>                                  <!-- 1ms v
[us] vydrz -->
<command name="CAENVME_WriteCycle" offset_32_address="0x1180" address_modifier="0x09"
data_width="32" input_value="15700" repeat="1"/>
<command name="DELAY" input_value="1000"/>                                  <!-- 1ms v
[us] vydrz -->
<command name="CAENVME_WriteCycle" offset_32_address="0x1280" address_modifier="0x09"
data_width="32" input_value="15700" repeat="1"/>
```

Listing 9-1 NWVME configuration file example

An example of configuration file of the new DAQ is given in Listing 9-2. The XML configuration file is designed as self-documenting. An example configuration file of all supported hardware is generated by the DAQ command and can be then used as a template by the user. Any user should be able to find and modify important values easily, for example if given an instruction over a phone. The configuration file should also keep the principle of defining any important values once to prevent errors. That is very important for the DPP firmwares, where every channel has many configuration parameters and any change needs to be applied for all channels introducing a space for mistake. The format therefore supports the #CLONE XML parameter which copies all unset options from other configuration node of the same level or below (i.e. channel from channel, channel group from channel group, card from card, VME crate from VME crate)

```
<!-- FrontPanelTrigger: Couple can trigger FP-TRGOUT -->
<!-- FrontPanelTrigger:        Boolean - 0 1 -->
<!-- PulseType: Trigger Pulse Generation -->
<!-- PulseType:         Enumerated: Programmable(0) Threshold(1)  -->
<!-- Trigger: Couple can trigger -->
<!-- Trigger:   Boolean - 0 1 -->
<!-- TriggerLogic: Couple Trigger Logic -->
<!-- TriggerLogic:      Enumerated: AND(0) FIRST(1) OR(3) SECOND(2)  -->
<couple id="0" #CLONE="" FrontPanelTrigger="0" PulseType="Programmable" Trigger="1"
TriggerLogic="OR">
        <!-- Name: Channel name -->
        <!-- Enabled: Channel enabled -->
        <!-- Enabled:   Boolean - 0 1 -->
        <!-- Offset: DAC offset -->
        <!-- Offset:    Unsigned integer range: 0 - 65535 -->
        <!-- PulseWidth: Trigger Coincidence Pulse Width, 16 ns step -->
        <!-- PulseWidth:        Unsigned integer range: 0 - 255 -->
        <!-- DynamicRange: Input dynamic range -->
        <!-- DynamicRange:      Enumerated: 0V5(1) 2V(0)  -->
        <!-- Threshold: Trigger threshold -->
        <!-- Threshold:         Unsigned integer range: 0 - 16383 -->
        <channel id="0" #CLONE="" Name="CH1" Enabled="1" Offset="4096" PulseWidth="2"
DynamicRange="0V5" Threshold="15000"/>
        <channel id="1" #CLONE="0" Name="CH2" Enabled="1"  Threshold="15000"/>
</couple>
<couple id="1" #CLONE="0" Trigger="1" TriggerLogic="OR">
        <channel id="2" Name="VETOUP"  Enabled="1" Threshold="15000"/>
        <channel id="3" Name="VETOBOT" Enabled="0" Threshold="14434"/>
</couple>
```

Listing 9-2 Я^{3}DAQ configuration file example

The DAQ component then translates this hardware specific configuration file into configuration file for the BOX – where only important information for every channel is kept – sampling rate, dynamic ranges and threshold values.

10 Conclusions

This thesis deals with detection of the inverse beta decay in nuclear reactor antineutrino detectors. The author has been involved in two such experiments – DANSS and S^3. The DANSS experiment was developed by the Dzhelepov Laboratory of Nuclear Problems of the Joint Institute of Nuclear Research in Russia. The S^3 experiment is being developed as a collaboration of the Institute of Experimental and Applied Physics of the Czech Technical University in Prague and the DLNP JINR.

The DANSS experiment has been in operational phase since summer 2016 and is still collecting data in the technical room 10 meters underneath the reactor core of Kalinin Nuclear Power Plant in Russia. The S^{3}experiment is, as of August 2018, still being designed and constructed.

The goals of the thesis were to devise and implement the method of detection inverse beta decay and implement them for the two experiments.

For the DANSS experiment a method of a coincidence of a group of pulses (spatial and temporal coincidence) was devised and implemented as an FPGA firmware for the main control card of the data acquisition system. The DANSS spectrometer uses the charge to digital converters for particle energy extraction and an FPGA based control card for trigger control. A variant of the firmware was designed by the author for the active veto system of the detector which records any events occurring around the time of the trigger. The author also participated during the commissioning phase of the DAQ in the Kalinin NPP in technical data analysis.

For the S^3 experiment the same coincidence method was implemented in a C++ application. It can be used for both online and offline processing from trace recording oscilloscopes and CAEN ADC digitizers. The resulting application allows data analysis, visualization, online monitoring and data reduction down to a manageable level. The resulting DAQ has been tested on the S-Cubino prototype.

Further, based on the experience obtained during the work on the thesis a new complete data acquisition system has been proposed and its development has started and will be used for the complete S^3 detector both in Czech republic and Russia.

11 List of relevant publications

11.1 Impacted publications

[IF1] Hons, Z., Vlášek, J. – Data acquisition system for segmented reactor antineutrino detector, Journal of Instrumentation, 2017 JINST 12 P01022, ISSN 1748-0221

[IF2] Alekseev, I et al - DANSS: Detector of the reactor AntiNeutrino based on Solid Scintillator, Journal of Instrumentation, 2016, JINST 11 P11011, ISSN 1748-0221

11.2 Proceedings papers

[PP1] Alekseev, I. et al - Neutrino Physics at Kalinin Nuclear Power Plant: 2002 – 2017, Journal of Physics: Conference Series, Volume 934, Issue 1, 20 December 2017, ISSN 1742-6596

[PP2] Alekseev, I. et al - Detector of the reactor AntiNeutrino based on Solid-state plastic Scintillator (DANSS). Status and first results, Journal of Physics, Conference series, vol 898 ISSN 1742-6588

[PP3] Špavorová, M, et al – Testing of reactor antineutrino detector s-cube, AYSS-2016, p 250-254, ISBN 978-5-9530-0416-9

12 Other results

12.1 Other impacted publications

[OIF1] Mamedov, F, et al – Measurement of radon aktivity in air using electrostatic collection to the Timepix detector, JINST, 2013 JINST C03011, ISSN 1748-0221

12.2 Other conferences

[OC1] Broulím, P, et al - Compact device for detecting single event effects in semiconductor components, TELFOR 2017 Belgrade, Serbia

[OC2] Vlášek J – Use of Altera FPGA for digital signal processing, Elektronika a informatika 2011, Part 2 Elektronika, p 119-120

[OC3] Vlášek J – Effect of sampling frequency on the accuracy of radiation spektrometry, Elektronika a informatika 2012, Part 2, p 143-146

[OC4] Vlášek J – Antineutrino detector, Elektronika a informatika 2013, Part 2, p 93-96

[OC5] Vlášek J, et al, 21st – Software for rail traffic simualator, 21st Telecommunications Forum (TELFOR) Proceedings Papers, p 594-596

[OC6] Křivka J, et al – Hardware for rail traffic simulator, 21st Telecommunications Forum (TELFOR) Proceedings Papers, p 584-586

[OC7] Štětka P, et al – Control and navigation system for mobile platform 21st Telecommunications Forum (TELFOR) Proceedings Papers, p 584-586

13 Prototypes and software

[FZ1] Pavlíček V, Vlášek J - Terminal for control of winding machine tools, 2012

[FZ2] Vlášek J, Georgiev V – USB hub with current measurement, 2012

[FZ3] Vlášek J, Georgiev V – Radiation spectrometer based on time measurement, 2012

[FZ5] Broulím J, Vlášek J, Georgiev V – Programmable power supply, 2013

[FZ5] Vlášek J, et al – Portable USB spectrometer for detection of radon

[SW1] Vlášek J, Georgiev V – Software for rail vehicle simulator, 2013

[SW2] Vlášek J, et al – Software for measurement and data processing for portable USB spectrometer, 2012

14 List of figures

15 List of Listings

16 List of Tables

17 References

1. *Solar Neutrinos.* **Suzuki, Y.** s.l. : World Scientific Publishing Company, 2000, International Journal of Modern Physics A, Vol. 15, pp. 201-228.

2. *Observation of Reactor Electron Antineutrinos Disappearance in the RENO Experiment.* **Collaboration), J. K. Ahn et al. (RENO.** 2012, Phys. Rev. Lett, Vol. 108, p. 191802.

3. *DANSS: Detector of the reactor AntiNeutrino based on Solid Scintillator.* **Egorov, V. et al.** s.l. : arXiv:1606.02896, 2016, JINST, Vol. 11, p. P11011.

4. **Pauli, W.** Letter to L.Meitner and her colleagues. December 4 1930 .

5. **Fukugita, M., Yanagida, T.** *Physics of Neutrinos and applications to astrophysics,.* s.l. : Springer-Vergag Berlin, 2003. ISSN 0172-5998.

6. *Super-Kamiokande Collaboration.* **Fukuda, Y. et al.** 1562, s.l. : Phys. Rev. Lett. , 1998, Vol. 81.

7. *A review of the homestake solar neutrino experiment.* **Davis, R.** s.l. : Elsevier, 1994, Progress in Particle and Nuclear Physics, Vol. 32, pp. 13-32.

8. **Barger V., Marfatia D., Whisnant K. L.** *The Physics of Neutrinos.* s.l. : Princeton University Press, 2012. 0691128537.

9. *Mesonium and antimesonium.* **Pontecorvo, B.** Dubna : Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki, 1957, Vol. 33, p. 549.

10. *Inverse beta processes and nonconservation of lepton charge.* **B, Pontecorvo.** Dubna : Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki, 1957, Vol. 34, p. 247.

11. *The Construction and Anticipated Science of SNOLAB.* **Duncan, F. et al.** 163-180, 2010, Annual Review of Nuclear and Particle Science, Vol. 60, pp. 163-180.

12. *The detector system of the Daya Bay reactor neutrino experiment.* **An, F.P. et al.** s.l. : Elsevier, 2016, NIM A, Vol. 811, pp. 133-161.

13. *Reactor antineutrino anomaly.* **Mention, G. et al.** 073006, s.l. : Phys. Rev. D, 2001, Vol. 83.

14. *The Super-Kamiokande detector.* **Fukuda, S. et al.** 2-3, s.l. : Elsevier, 2003, Vol. 501, pp. 418-462.

15. *IceCube: An instrument for neutrino astronomy.* **Halzen, F., Klein, R.S.** s.l. : AIP, 2010, Review of Scientific Instruments, Vol. 81, p. 081101.

16. *The Baikal underwater neutrino telescope: Design, performance, and first results.* **Belolaptikov, I. A, et al.** 3, s.l. : Elsevier, 1997, Astroparticle Physics, Vol. 7, pp. 263-282.

17. *The BAIKAL neutrino experiment—Physics results and perspectives.* **Belolaptikov, I. A, et al.** 1, s.l. : Elsevier, 2009, NIM A, Vol. 602, pp. 14-20.

18. *ANTARES: The first undersea neutrino telescope.* **Ageron, M. et al.** 1, s.l. : Elsevier, 2011, NIM A, Vol. 656, pp. 11-38.

19. *DANSSino: a pilot version of the DANSS neutrino detector.* **Egorov, V and al, et.** 2013, p. arXiv:1305.3350.

20. **CAEN.** CAEN V965 User Manual. [Online] http://www.caen.it/servlet/checkCaenManualFile?Id=5333.

21. *A versatile DAQ, monitoring and data processing system for nuclear experiments in CAMAC and VME standards.* **Hons, Z.** 2015, p. arXiv:1508.01379.

22. **CAEN S. p. A.** V1740 User Manual. [Online] http://www.caen.it/servlet/checkCaenManualFile?Id=13148.

23. **CAEN S.p.A.** DPP-PSD User Manual. [Online] http://www.caen.it/servlet/checkCaenManualFile?Id=13080.

24. **Egorov, V et al.** DANSSino: a pilot version of the DANSS neutrino detector. [Online] arXiv:1305.3350 [physics.ins-det].

25. **Gilmore, Gordon R.** *Practical Gamma-ray Spectrometry – 2nd Edition.* s.l. : John Wiley & Sons, 2008. 978-0-470-86196-7.

26. **Favi, Claudio and Edoardo, Charbo.** A 17ps Time-to-Digital Converter Implemented in 65nm FPGA Technology. [Online] http://infoscience.epfl.ch/record/139431.

27. **Egorov, V et al.** *DANSSino: a pilot version of the DANSS neutrino detector.* 2013. arXiv:1305.3350 [physics.ins-det].

28. **Gravitational Waves & High Energy Neutrinos.** ***Antares.*** **[Online] [Cited: March 10, 2018.] http://antares.in2p3.fr/users/pradier/gwhen.html.**

29. CAEN S.p.A. DPP-QDC User Manual. [Online] http://www.caen.it/servlet/checkCaenManualFile?Id=12971.