Draft 1.0: Manual for AMANDA II Transient Waveform Recorder DAQ - 2004

Karl Heinz Becker, Timo Messarius, Wolfgang Wagner

Jan 26th 2004

Contents

1 A short description of the new AMANDA DAQ **2**
1.1 Transient Waveform Recorder - The heart of the system 2
1.2 The GPS-Latch GPS2VME 3
1.3 The Digital Signal Processor DSP 4
1.3.1 The Pulse Extraction Algorithm 5
1.4 The eventbuilder process and dataflow 5
1.5 The trigger system 8

2 Running the system **9**
2.1 Automatic mode 10
2.2 Starting the system manually 11
2.3 The Initialisation process 11
2.4 Stopping the system and shutting down 12

3 Description of the config file TWR.cnf **13**

4 Filename convention **15**

5 Layout of the binary file format **16**

6 Errors and Exitcodes **18**

1 A short description of the new AMANDA DAQ

This document describes the current version of the Transient Waveform Recorder Data Acquisition system (TWRDaq). In contrast to the current AMANDA DAQ system which is referred as MuonDaq throughout this document, the TWRDaq is sampling the complete pulseform (waveform) of the PMT pulses.
The current TWRDaq system, which is displayed in figure 1 is a modified version of the DAQ built during the Poleseason 2002/2003. The goal was to make the system more reliable and to increase the trigger rate up to about 200 Hz. The TWR system is still triggered by the DMADD trigger system of the MuonDaq.
The heart of the TWRDaq are Flash ADCs called Transient Waveform Recorder (TWR), which are VME modules located in the crates 0 to 5. Since the maximum power consumption of the VME crates is about 100A at 5V the number of TWRs is limited to 14 modules per crate. These crates are controlled by a SIS3100 module (slot1) acting as bus master. The SIS3100 contains a programmable DSP that is used for data collection and compression.
At the same time the SIS3100 serves as a VME to VME bridge. The SIS3100 is connected via two optical fibre to a similar module in the master crate, which receives the data and stores it in a FIFO on request.
One of the VME crates holds an additional GPS-latch (GPS2VME) latching the actual time for every incoming trigger. This module is currently sitting in crate 3. The necessary trigger logic and some level converters are located in a NIM crate below the master crate.
The system is started and controlled by the DAQ software running on a Linux PC (event-builder on twr-daq), which is connected to the master crate by an additional bridge consisting of a SIS3100 module in the master crate and a SIS1100 (PCI card) in the PC, linked by an optical fibre.
The various parts and the readout process will be described on the following pages.

1.1 Transient Waveform Recorder - The heart of the system

The Transient Waveform Recorder TWR build by SIS [1)] is the heart of the recording system. It is a 12 bit Flash ADC sampling continuously the incoming signal at a rate of about 100 MHz in a ring buffer with a length of 1024 samples covering a total time window of 10.24 μsec. After every external trigger pulse (event-trigger), which serves as a COMMON STOP TRIGGER the TWR switches to the next ring buffer and starts sampling for the next event. Every TWR has 8 channels and and a total voltage range of 5V. In order to sample the PMT pulses at the right level, the range of the TWR is adjusted to 1V to -4V. For the optical channels of AMANDA the fast output of the ORBs and the ORMs are chosen as input signal, for the electrical channels the fast output of the SWAMPS is used. Every TWR has two memorybanks containing each 128 events. The TWRs are running in *Autobankswitchmode*, which reduces the deadtime of the system by separating the readout and the data-taking process. After starting the sampling, the first memory bank is filled. While reading out the first memory bank, the second memorybank is filled. As long as the readout time for one memory bank is not longer than the time for 128 triggers, the

[1)] Struck Innovative Systems - Hamburg

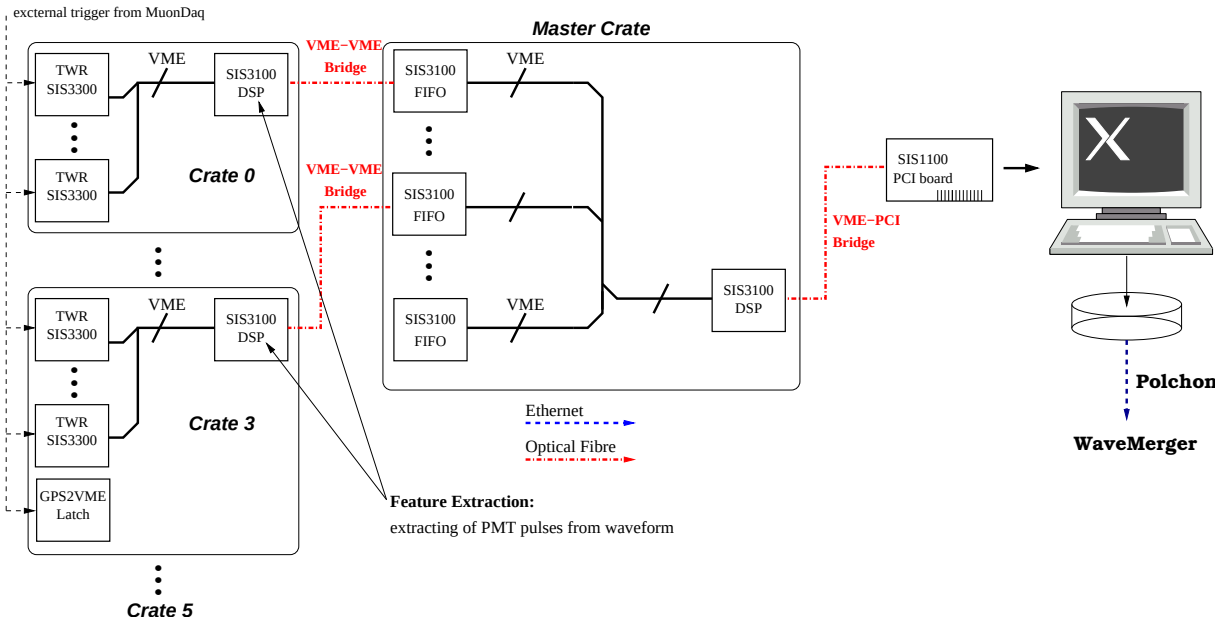


FIGURE 1: *The complete TWR DAQ system.*

data-taking and the readout process are independent.
The TWR is equipped with a timestamp counter running with clock speed or with a downscaled clock speed, which enables it to measure the time between events. With downscaling the timestamp clock speed, the sample clock speed is not affected. This feature of the TWR enables the TWR daq to find any synchronisation errors between the TWRs by comparing the time differences between same events in different TWRs.
Since the number of waveforms with a PMT pulse in it is rather small, it is essential to detect these valid waveforms and store the information in a dedicated register. During the readout process only valid waveforms are read out. Every channel has an independent programmable threshold, which is set during initialisation. A waveform is valid, if at least one of the 1024 values is below the threshold. Thus the thresholds can be adjusted to the individual noise behaviour of each OM.
After 128 triggers one memory bank is filled and it is read out by the Digital Signal Processor (DSP), which is located on the SIS3100 module 1.3.

1.2 The GPS-Latch GPS2VME

The GPS2VME latch is basically a clock which is synchronised externally by a Global Positioning System (GPS) clock. It is triggered externally and latches for every trigger the actual time information consisting of 4 longwords. For more information see GPS2VME
The external trigger signal is delivered simultanuously to all TWRs and to the GPS-latch, which is synchronized by a connected GPS-clock and samples a timestamp for every event.

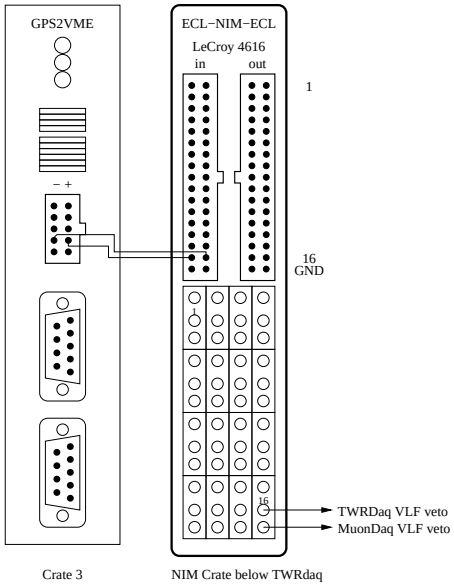


FIGURE 2: *The cabling for the VLF trigger veto.*

The GPS-Latch holds a FIFO containing the timestamps each consisting of 4 longwords and is read out together with the TWRs every 128 events.
In addition the GPS latch produces the VLF trigger veto for the MuonDaq and the TWRDaq. Never switch off crate 3 holding the GPS latch during VLF times.
The cabling for the VLF trigger is shown in figure 2

1.3 The Digital Signal Processor DSP

On the VME bridge module (SIS3100) in the slave crates, which acts as a VME master in these crates, a Digital Signal Processor (DSP) is located. The DSP is a processor optimized for large data amounts and small algorithms. The programs are written in assembler. Thus, it is very fast. The DSP reads out all the TWRs and the GPS Latch and compresses the waveform by extracting the pulse from the complete waveform. Since the PMT pulses of an electrical channel have a width of about 300 nsec and the pulses of an optical channel occupy only 30 nsec, this is an effective way to reduce the data amount at an early stage. The data is stored in a memory located in the same module.
The program for the DSP is loaded and started during initialisation by the DAQ software on the Linux PC (eventbuilder running on twr-daq.spol.gov). For communication between the eventbuilder and the DSP several registers located in the SDRAM of the SIS3100 are used.
As soon as the DSP has finished extracting the pulses and sorting the data of one memory bank, it sets a command to the DPM_PARAMRETER_CMD register and waits for the signal in the DPM_PARAMETER_ACK register from the eventbuilder, confirming the

successfull data taking of the previous bank.

1.3.1 The Pulse Extraction Algorithm

The Pulse Extraction Algorithm or Feature Extraction FE as it is also called, is described in figure 4. The DSP reads from the TWR an array of 1024 values and extracts from it the PMT pulse and a programmable number of preceeding (N_PREC) and following (N_FOLL) values. The extracted pulses are stored in a dataformat described in figure 9. At first the current baseline is determined. Since the baseline can vary with time, an estimate for the baseline is helpful for the analysis. The procedure estimating the baseline is described in figure 3. At first, the DSP starts a loop over the first 32 values of the waveform. If no value below the threshold is detected, the average value of these 32 values is considered as an estimate for the baseline. If at least one value below the threshold is detected, the baseline is determined by the last 32 values of the waveform. It is assumed that the probability for a pulse at the very beginning of the waveform and at the same time at the very end is pretty low.
The Pulse Extraction Algorithm searches for a pulse in the waveform using the same threshold as the TWR. As soon as a value below the threshold is detected, the loop counter is set back by N_PREC and a fragment is opened. After the preceeding values are added to the fragment the pulse itself is added using a lower threshold for the pulse end than for detecting it. This smaller threshold is calculated as baseline - 4. Since the slope of the leading edge of the PMT pulse is much smaller than that of the trailing edge, this lower threshold for the end of the pulse proved to be a better solution.
However memory space for a program in the DSP is limited and the algorithm has to be kept very simple. The current algorithm as displayed in figure 4 does not recognize under certain circumstances, if two fragments are overlapping. In order to reduce the amount of data and to make the further analysis easier, these overlapping fragments are merged to one compact fragment by a simple routine in the eventbuilder.

1.4 The eventbuilder process and dataflow

The eventbuilder process running on the Linux-computer twr-daq.spole.gov, which is equipped with a SIS1100 PCI card reads the data from the master VME crate and sorts the data according to events. In the rawdata coming from the TWRs the data is sorted by crates.
A noise trigger or a lost trigger in one of the modules may affect the events strongly because hits in the TWRs would be associated with a wrong GPs timestamp. In order to detect inconsistencies in time, the time difference between event 127 and event 128 is calculated using the TWR timestampcounter and the GPS time for every block of 128 events. By comparison of these values every time inconsistency can be detected. In such a case the trigger is blocked and the system is restarted.
Errors which are found during the readout like for synchronisation between the TWR timestamps and the GPS times can be found in an errorfile which is opened for every run (errorlog_run_XXXX.msg).
The output of the eventbuilderprogram is a binaryfile, which format can be seen in fig-

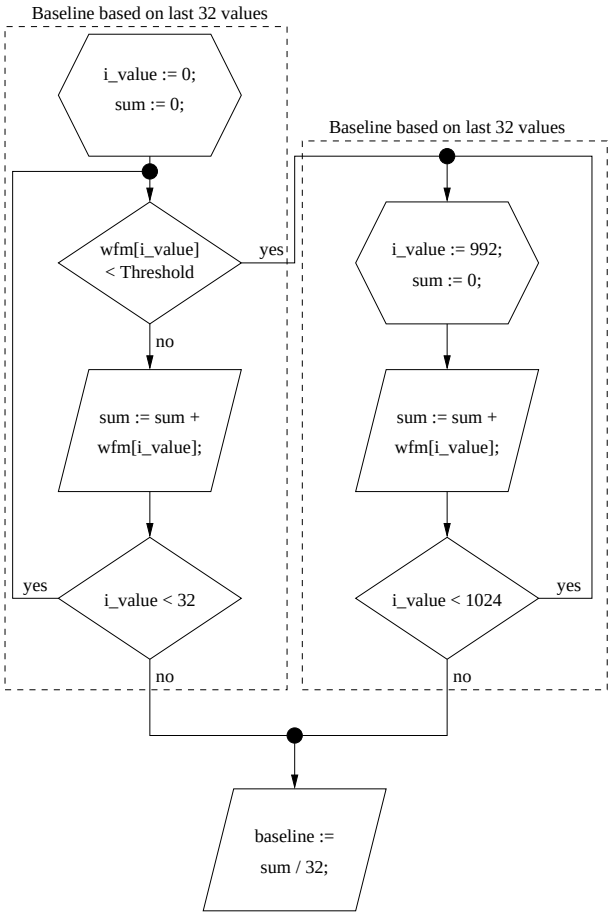


FIGURE 3: *The procedure for estimating the baseline.*

ure 8. The header of this file contains the whole structure of the system and all important numbers, like the threshold applied for the feature extraction. The structure of the header can be seen in figure 7. The eventbuilder writes all the data in files to the local disk (**/export/seal1/datadir**). The filename convention is described in chapter 4. The data is transported to other disks on the bos machines. The further dataflow is well described in the manual pages of the datahandler. The raw TWR data is afterwards written to tape and send to other related processes.
In order to verify the quality and to get as much as possible information from both MuonDaq and TWRDaq, the TWR-data has to be merged with the muon-data. This means to identify events which belong to the same physical event and is performed by the wavemerger program by Jens Ahrens.
The complete dataflow is shown in figure 5. The data is collected in the directory /export/seal1/datadir on the local disk on twr-daq in the MAPO. POLECHOMPER transports this data from the AMANDA site to the computers in the Back of Science (BOS). All the rawdata is stored on SDLT tapes.
In order to transmit the most important (high energy) data at once to the Northern hemi-

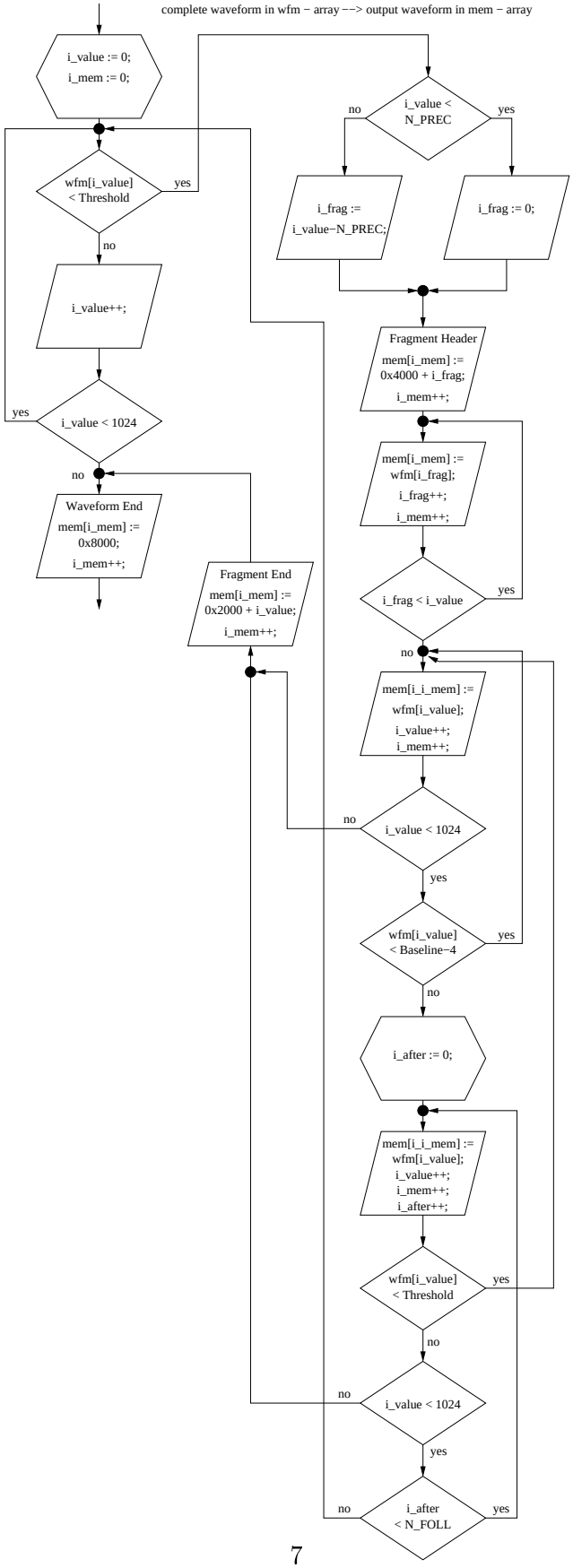


FIGURE 4: *The Feature Extraction Algorithm.*

sphere a filter process is applied to the data. The reader written by Jens Ahrens from Mainz reads the raw data and applies a cut on the number of waveforms per event, which is currently set to 120. Events pssing this cut are written into a root format and send to the merging process, which searches the data from the MuonDaq for events matching the TWR events and writes them into the F2k format. In addition to these high energy events a downscaled stream of normal events are send also to merging process. Currently every 25th event is selected, which helps to understand these low energy events in the Northern hemisphere. However a lot of these low energy events are not written to F2k, because there are no matching events from the MuonDaq. The data produced by the merging process is send to the satellite link and transmitted to the Northern hemisphere.
For monitoring the system, the reader produces histograms which are written into a root format and later merged with other root monitoring files from the MuonDaq to a run based monitoring file and transmitted to the Northern hemisphere by satellite.

1.5 The trigger system

Since the TWR DAQ in the current version has no internal trigger system, it has to use a trigger distributed by the current muon DAQ. While in 2003 the normal tMuonDaq trigger (majority = 24) with a frequency of about 90 Hz was used, it is intended to use in 2004 the majority = 18 trigger which is still delivered by the trigger system of the MuonDaq. The original trigger signal from the DMADD goes through a veto-logic to stabilise the system. An artifical deadtime of about 20 μsec is used since a trigger incoming within the actual time window of a previous event would led to a half filled waveform window and thus result in corrupted data. The whole trigger system can be seen in figure 6.

The external trigger pulses from the DMADD (event-trigger) is used as input for a level three coincidence. Only, if the TRIGGER ENABLE from the TWR daq and the and the VETO DISABLE show also a logic 1, the trigger is passing and starts the TWR. The veto is done by a gate generator which is connected to the output of the coincidence module with a delay of about 30ns. Since the inverse output of the gate generator is connected to the logic coincidence, it will disable the output of the coincidence module for the chosen length of the gate signal.
The passing trigger is distributed by a logic fan out to the different crates holding the TWRs and GPS Latches. In order to reduce the cabling in front of the crates and to make the system more reliable, the trigger is distributed via the P2 bus on the VME backplane. The trigger is received by a converter board in the crates changing the signal level to ECL and transmitting it to the backplane. This leads to few cabling and a more reliable trigger distribution.
The incoming trigger signal is delayed by the TWR for a programmable number of clock cycles (internal clock of the TWR 100MHz) and serves as a COMMON STOP TRIGGER to the TWR, which switches to the next ring buffer and starts sampling for the next event. This programmable delay is useful for levelling of the different leading edge times of OMs in different depths. The first daq in 2002 used a fixed delay of about 7μsec, which leads to a time window of about 3μsec before and 7μsec after the trigger arrival (in order to get also a large amount of afterpulses).
This external trigger signal is delivered simultanuously to all TWRs and to the GPS-latch,

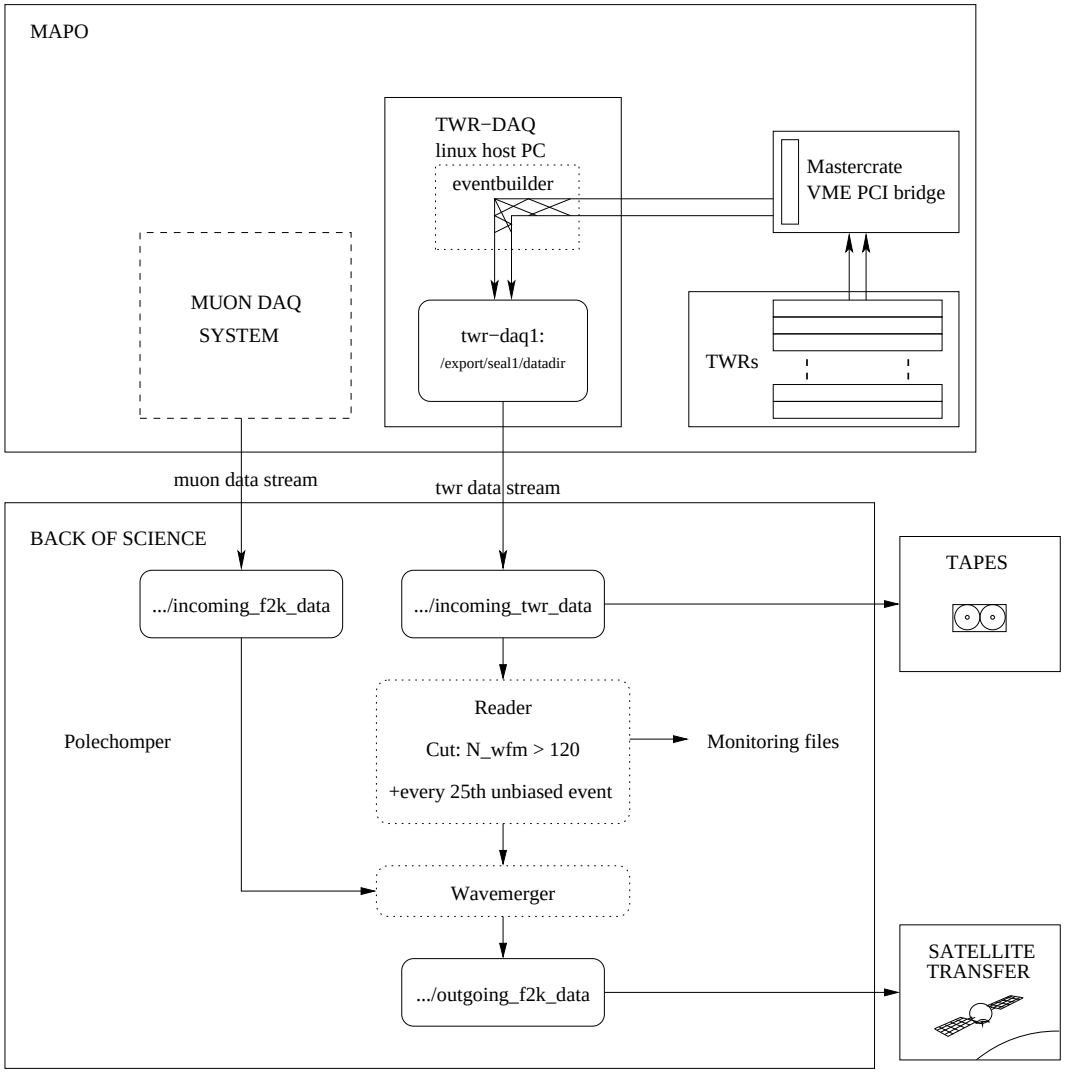


FIGURE 5: *Dataflow overview*

which is synchronised by a connected GPS-clock and samples a timestamp for every event. The GPS-Latch holds a FIFO containing the timestamps.
To reduce the amount of data all empty waveforms, that means all waveforms with signals below a certain threshold are not read out by the DSP.
A description of the trigger system including all important timedelays can be seen in figure 6.

2 Running the system

There are two possibilities to run the TWR-daq. Normally the TWR-daq is started together with the muon-daq by a PERL script in *Automatic Mode*. For testing purposes this

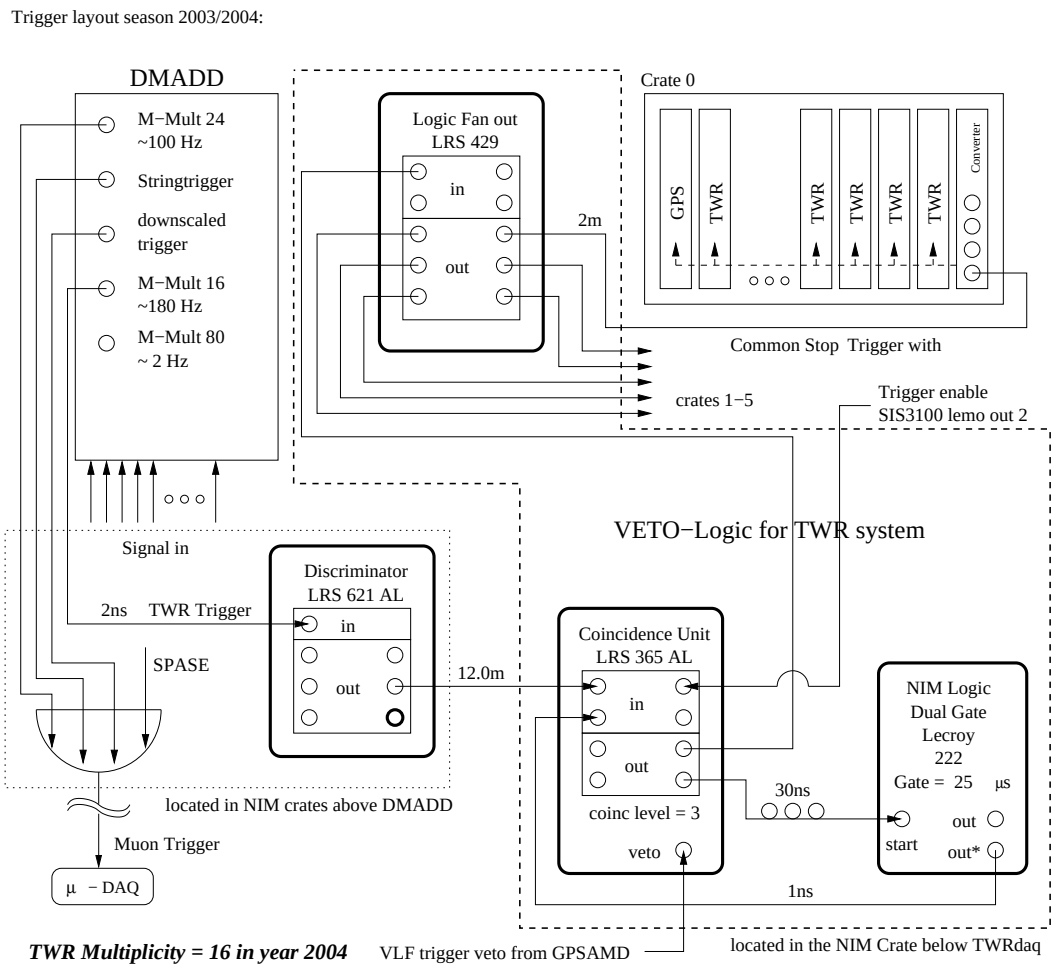


FIGURE 6: *Trigger setup overview*

document describes also running the TWR-DAQ step by step manually.

2.1 Automatic mode

The TWR is started by the PERL script remote_start_twr_daq.pl on amanda-daq, whenever the muon-daq is started. It is located in the directory daqops/current_daq. The corresponding stop-script, remote_stop_twr_daq.pl is located at the same position.
The start-script starts another script, start_twr_daq.pl on twr-daq1, which is located in the directory /home/daq/current_daq. start_twr_daq.pl starts all the necessary processes on twr-daq1 and rio2. Stopping the TWR-daq is done by the script remote_stop_twr_daq.pl on amanda-daq and stop_twr_daq.pl. Both start_twr_daq.pl and stop_twr_daq.pl can be started independent from the muon-daq on twr-daq1. In case of problems the output of the eventbuilder can be checked by the log-file log.eventbuilder, which can be found in

/home/daq/current_daq. In addition there is a restart-script, which stops the system and starts it again. These scripts use the same commands as described in chapter 2.2.

2.2 Starting the system manually

First the Linuxserver twr-daq.spole.gov has to be started. Login as user daq and change to the directory /export/seal1/daq/current_daq. In this directory you will find the event-builder program.
When the power on the VME-crate is turned on, the system is ready to run

./eventbuilder <run no.> <file no.> <run comment>

where file no. is the number of the first file, which is incremented for every following file of this run. The run command is only an option. It is a string which is written to the errorlog file at begin. The last used runnumber and filenumber are stored in last_run.txt. If a new run is started with the same runnumber as before and a filenumber smaller than the last filenumber, the filenumber is incremented automatically.
When starting the eventbuilder writes a lockfile eventbuilder.lock. If the eventbuilder is already started or aborted abnormally, this lockfile prevents a restart.
If a fatal error is detected during the start procedure the eventbuilder exits with an exit code. The cmplete descrition of exit codes can be found in chapter 6.

2.3 The Initialisation process

When starting the system the configuration file TWR.cnf containing all important information is loaded. This file is described in chapter 3 and locates on twr-daq1.spole.gov in the directory /export/seal1/daq/current_daq.
The initialisation includes a small routine determining the baseline of each channel. The TWRs are started and will sample 128 waveforms and will determine the baseline for every channel by integrating over all waveforms and calculating the mean value. Since most of the waveforms are empty this is a good approximation for the baseline.
These values are integrated to the data stream and will appear in the header of every raw data file. An analysis of baseline shifts is possible in this way and the daq is independent from long term baseline shifts during the winter.
After this, the initialisation of the TWR will start. The threshold for detecting valid waveforms is calculated by
(threshold for TWR) = (baseline) - (threshold from config file)
The GPS-Latch is started and the TWRs are armed for sampling. Then the DSP program is loaded and started.
In order to start the system synchronously, the trigger is blocked by the TWRDaq during initialisation and will be enabled at the beginning of the run. This is explained in detail in the description of the trigger system 1.5 Then the acqusition is started by arming the TWR and the GPSAMD for sampling.

2.4 Stopping the system and shutting down

The eventbuilder stops working when receiving a SIGINT signal. While stopping the last file is closed and the trigger is disabled.
If the eventbuilder was started manually, it can be stopped in the shell where it was started with
CTRL-C or by
kill -s INT <pid of eventbuilder>

3 Description of the config file TWR.cnf

The TWR.cnf, which contains important information on the system like the number of the connected OMs, the TWRs and the applied threshold, is located on: twr-daq in /export/seal1/daq/current_daq
All hardware changes have to be mirrored in this file. The clock_predivider sets the speed of the internal timestamp-counter relative to the TWR frequency of 100 MHz. A value of 100 means that the timestamp-counter is running with a frequency of 100 MHz / 100 = 1 MHz and has an accuracy of 1μsec.

A typical TWR.cnf can be seen here:

GENERAL

/* PCI driver name */
PCI_DRIVER_NAME /usr/local/sis1100/nod/sis1100
`Directory in which the SIS1100 driver is located.`

/* Data output directory */
DATA_OUT_DIRECTORY /export/seal1/datadir/
`Output directory for the data.`

/* Number of 128-event blocks per file */
EVENTBLOCKS_PER_FILE 40
`This number multiplied by 128 gives the number of events per rawdata file.`
/* VLF trigger stuff */
VLF_TRIGGER_VETO 0
DURATION_OF_VETO 60
OVERLAP 3
TIME_0 0
TIME_1 15
TIME_2 30
TIME_3 45
`The GPS latch will produce 4 trigger vetos with a length of DURATION seconds beginning at TIME_0, TIME_1, TIME_2, TIME_3 minutes of each hour. This veto is extended in both directions by OVERLAP seconds.`

/* DSP parameters for Feature Extraction */
PRECEEDING_VALUES_OPTICAL 0x3
FOLLOWING_VALUES_OPTICAL 0x4
PRECEEDING_VALUES_ELECTRICAL 0x10
FOLLOWING_VALUES_ELECTRICAL 0x12
`These parameters define how the Pulse Extraction in the DSP works. See 1.3`

/* clock predivider for internal TWR clock */

CLOCK_PREDIVIDER 1

The clock_predivider sets the speed of the internal timestamp counter relative to the TWR frequency of 100 MHz. A value of 100 means that the timestamp counter is running with a frequency of 100 MHz / 100 = 1 MHz and has an accuracy of 1μsec.

STATUS_REG 0x102

ACQ_CONTROL_REG 0x2b4

EVT_CONFIG_REG 0x2c

TFCC 1022

These values define various parameters for the TWR, for example the waveform window length

N_CRATES 6

The number of crates holding TWRs.

CRATE_0

/* local VME base of bridge module in master crate */

BASE_BRIDGE 0x04000000

The base address of the local SIS3100 bridge module in the master crate connecting to this remote crate

/* Sharc program */

/* has to be located in direcory current_daq */

DSP_PROGRAM s3300v61.ldr

The DSP file for the remote SIS3100 bridge module in this crate

BASE_GPS 0x10000000

The remote VME base address of the GPS2VME in this remote crate. If it is set to 0xffffffff there is no GPS2VME in this crate

N_TWR 0x10

The number of TWRs in this crate

BASE_TWR 0x00000000

The VME base address of the first TWR

EXTERN_STRT_DEL 0

The start delay for this TWR. It has no meaning for the current DAQ.

EXTERN_STOP_DEL 700

The stop delay for this TWR. This number multiplied by 10nsec defines the delay time for an incoming trigger. A value of 700 means, that the waveform covers about 3000nsec before the trigger and 7000nsec after the trigger.

TWR_OM 640 641 643 644 645 646 647 648

The number of the conected OM beginning with 1

OPTICAL 1 1 0 0 0 0 0 0

If the OM is read out by an optical fibre, the value is set to 1 otherwise 0

TWR_BASELINE 4000 4000 4000 4000 4000 4000 4000 4000

```
The baseline for the OM. Currently this value has no meaning, since the
baseline is determined automatically at the begin of each run.
```
TWR_THRESHOLD 20 20 20 20 20 20 20 20
```
The threshold for the channel is determined by:
<determined baseline> - TWR_THRESHOLD and is programmed into the TWR.
The thresholds are manually set and optimized for every single OM.
```
...

BASE_TWR	0xf0000000							
EXTERN_STRT_DEL	0x0							
EXTERN_STOP_DEL	700							
TWR_OM	123	124	125	126	127	128	129	130
OPTICAL	1	1	0	0	0	0	0	0
TWR_BASELINE	4000	4000	4000	4000	4000	4000	4000	4000
TWR_THRESHOLD	20	20	20	20	20	20	20	20

CRATE_2	
BASE_100MHz	0x0
BASE_GPS	0x10000000
N_TWR	0x10

...

All changes in the setup like changing OMs connected to the TWRs have to be documented in the TWR.cnf !

4 Filename convention

The convention for the binary filenames is

twr_yyyy_ddd_rrrr_ffff_bbbbb_eeeee.dat

with

yyyy	year
dddd	day in year
rrrr	run number as received from muon daq
ffff	file number incremented with every new file of actual run
bbbbb	time of first event in file in seconds of actual day
eeeee	time of last event in file in seconds of actual day

5 Layout of the binary file format

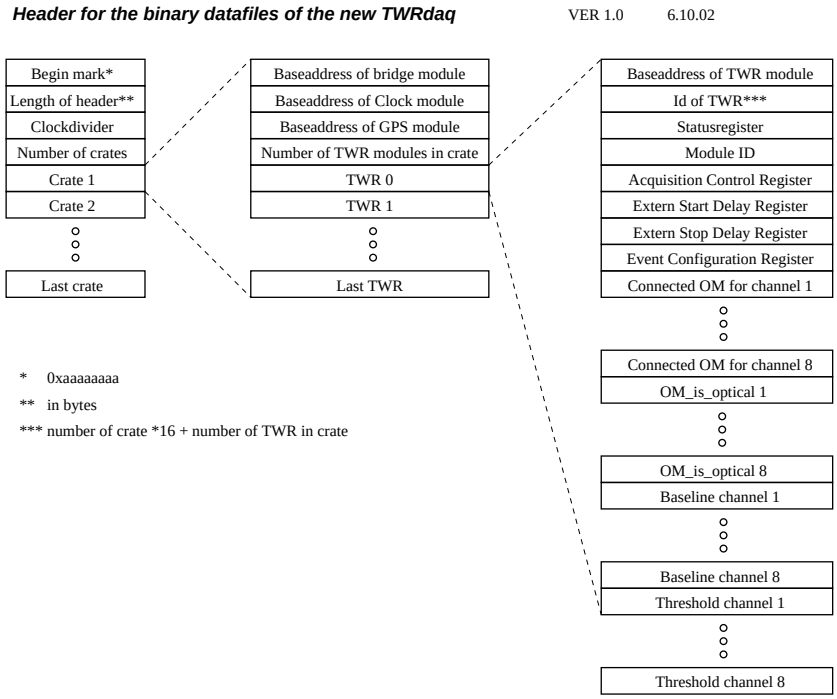


FIGURE 7: *Header of binaryfile*

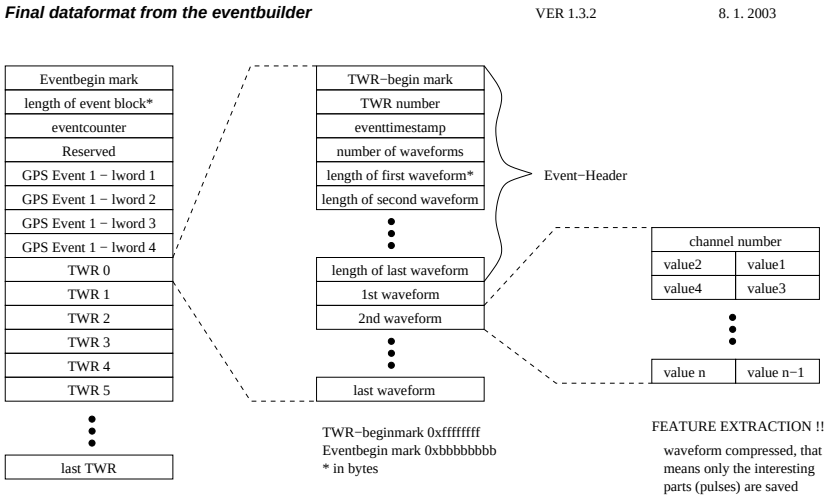
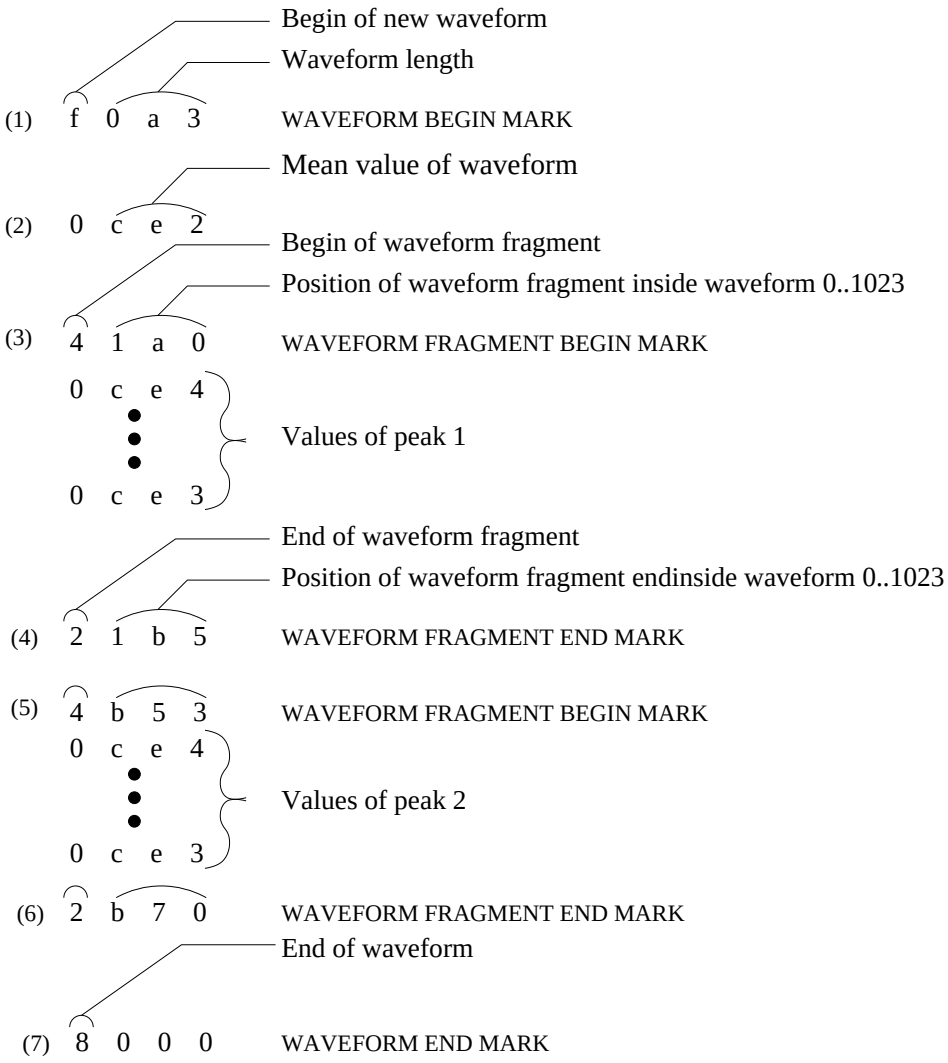


FIGURE 8: *Dataformat of binaryfile*



Each waveform begins with WAVEFORM BEGIN MARK accompanyed by the length of the whole waveform in bytes followed by the mean value of the waveform (2). Two waveform fragments are stored in this example beginning with WAVEFORM FRAGMENT BEGIN MARK (3) and (5) accompanyed by the position of the fragment inside the waveform and terminated with a WAVEFORM FRAGMENT END MARK (4) and (6). The waveform is closed by a WAVEFORM END MARK (7). To align the waveform to full 32 bit words, a filling word 0xaaaa is inserted if ne− cessary.

FIGURE 9: *Dataformat of compressed waveform*

6 Errors and Exitcodes

List of possible errorcodes obtained as exitcodes from the eventbuilder or/and written in the errorlog_run_xxxx.msg file. The return values give oly a rough hint for the possible error. For more information see the errorlog_run_xxxx.msg file.
If the eventbuilder aborted abnormally it can occur that a blocktransfer was not finished. In this case sometimes there is only the way to power cycle the crates.

General errors leading to an exit of the program		
Error	Code	Possible reasons
ERROR_NO_ARGUMENTS	1	Missing argument when starting the eventbuilder. Use: eventbuilder <run_no> <file_no>
ERROR_SIGNAL_HANDLER	2	Could not initialise signal handler.
ERROR_ALREADY_RUNNING	3	eventbuilder.lock exists already in this directory. The eventbuilder is already running or aborted abnormally without deleting the lock file.
ERROR_ALLOCATING_MEMORY	4	No free memory available. Cannot allocate memory.
ERROR_ARGS_NOT_NUMERICAL	5	The arguments are not numerical. Probably an error in PERL skript.
ERROR_CONFIRM_RUNNUMBER	10	last_run.txt with last used run_no and file_no does not exist or cannot be opened.
ERROR_CONFIG_FILE	11	Error occured while reading the config file. See message in errorlog_run_xxx.msg and list of possible Errors.
ERROR_OPEN_FIRST_FILE ERROR_OPEN_NEXT_FILE	12 13	Cannot open first/next file of run. Wrong datadirectory in TWR.cnf set ?
ERROR_WRITE_TO_FILE	14	Cannot write to file.

Errors during VME access leading to exit of the program		
Error	Code	Possible reasons
ERROR_OPEN_PCI	20	Connection to SIS1100 PCI-VME bridge cannot be opened. Driver not installed or VME crate switched off.
ERROR_INIT_GPS	21	GPS latch cannot be initialised. The GPS latch is wrongly mapped in the TWR.cnf, the VME base address is wrong or a problem with the local bridge occured.
ERROR_ARMING_TWRS ERROR_STARTING_TWRS	22 23	TWRs cannot be armed/started. Error in TWR.cnf,problem with local bridge module or wrong base address of TWR. See errorlog_run_xxxx.msg for more information.
ERROR_ENABLING_TRIGGER ERROR_GET_BASELINE ERROR_READING_GPS ERROR_DISABLING_TRIGGER ERROR_STOPPING_TWRS ERROR_INIT_TWR ERROR_CLEARING_TWRS ERROR_WRITING_HEADER ERROR_RESET_SHARCS ERROR_LOADING_SHARCS ERROR_INIT_SHARCS ERROR_STOPPING_SHARCS	24 25 26 27 28 29 30 31 32 33 34 35	These errors can occur only, if the a problem with the communication between the eventbuilder and the bridges exists, for example if a crate is not switched on or a previous run aborted abnormally leaving a blocktransfer via the bridges unfinished. In this case only power cycling the crate helps.
ERROR_SDRAM_READ	50	Cannot read from SDRAM - A restart of the system will occur.

Error during data readout		
Error	Code	Possible reasons
ERROR_TIMEOUT	51	The eventbuilder waited for more than 1.5 min for new data from the DSP. A restart of the system will occur.
ERROR_DATA_LENGTH_ZERO	52	Read datalength = 0 - A restart of the system will occur.
ERROR_DATA_LENGTH_TOO_LONG	53	Read datalength, which is larger than the maximum buffer size. - A restart of the system will occur.
ERROR_SDRAM_BLOCK_READ	54	Error during block transfer - A restart of the system will occur.
Errors related to waveform		
ERROR_IN_WFM ERROR_BAD_ALIGNMENT ERROR_WRONG_TWR_NO ERROR_NO_WFM_BEGIN ERROR_NO_FRAGMENT_BEGIN ERROR_NO_FRAGMENT_END ERROR_NO_WFM_END	60 61 62 70 71 72 73	Error in waveform format. Probably caused by an error during readout or in the DSP. A restart of the system will occur.

VME errors (right 3 digits)		
Error	Code	Explanation
RE_NRDY	0x202	remote station not ready
RE_PROT	0x206	protocol error (illegal request)
RE_TO	0x207	protocol timeout
RE_BERR	0x211	VME Bus Error
RE_RETRY	0x212	VME retry (not used)
RE_ARB	0x214	Arbitration timeout, could not get mastership