Technical Information Manual

Revision n. 10

16 March 2009

MOD. V1720

8 CHANNEL 12 BIT

250 MS/S DIGITIZER

MANUAL REV.10

NPO:

00103/05:V1720x.MUTx/10

CAEN will repair or replace any product within the guarantee period if the Guarantor declares that the product is defective due to workmanship or materials and has not been caused by mishandling, negligence on behalf of the User, accident or any abnormal conditions or operations.

CAEN declines all responsibility for damages or injuries caused by an improper use of the Modules due to negligence on behalf of the User. It is strongly recommended to read thoroughly the CAEN User's Manual before any kind of operation.

CAEN reserves the right to change partially or entirely the contents of this Manual at any time and without giving any notice.

Disposal of the Product

The product must never be dumped in the Municipal Waste. Please check your local regulations for disposal of electronics products.

TABLE OF CONTENTS

1. GENERAL DESCRIPTION 8

1.1. OVERVIEW 8

1.2. BLOCK DIAGRAM 9

2. TECHNICAL SPECIFICATIONS 10

2.1. PACKAGING AND COMPLIANCY 10

2.1.1. Supported VME Crates 10

2.1.2. Stand Alone operation 10

2.2. POWER REQUIREMENTS 10

2.3. FRONT PANEL 11

2.4. EXTERNAL CONNECTORS 12

2.4.1. ANALOG INPUT connectors 12

2.4.2. CONTROL connectors 12

2.4.3. ADC REFERENCE CLOCK connectors 13

2.4.4. Digital I/O connectors 13

2.4.5. Optical LINK connector 14

2.5. OTHER FRONT PANEL COMPONENTS 14

2.5.1. Displays 14

2.6. INTERNAL COMPONENTS 14

2.7. TECHNICAL SPECIFICATIONS TABLE 16

3. FUNCTIONAL DESCRIPTION 17

3.1. ANALOG INPUT 17

3.1.1. Single ended input 17

3.1.2. Differential input 17

3.2. CLOCK DISTRIBUTION 18

3.2.1. Direct Drive Mode 18

3.2.2. PLL Mode 19

3.2.3. Trigger Clock 19

3.2.4. Output Clock 19

3.2.5. AD9510 programming 19

3.2.6. PLL *programming 20*

3.2.7. Direct Drive *programming 20*

3.2.8. Configuration file 20

3.2.9. Multiboard synchronisation 21

3.3. ACQUISITION MODES 22

3.3.1. Acquisition run/stop 22

3.3.2. Data acquisition 22

3.3.3. Acquisition Triggering: Samples and Events 23

3.3.3.1. Custom size events 24

3.3.4. Event structure 24

3.3.4.1. Header 25

3.3.4.2. Samples 26
3.3.4.3. Event format examples 26
3.4. ZERO SUPPRESSION 29
3.4.1. Zero Suppression Algorithm 29
3.4.1.1. Full Suppression based on the amplitude of the signal 29
3.4.1.2. Zero Length Encoding ZLE 29
3.4.2. Zero Suppression Examples 30
3.5. TRIGGER MANAGEMENT 33
3.5.1. External trigger 34
3.5.2. Software trigger 34
3.5.3. Local channel auto-trigger 34
3.5.3.1. Trigger coincidence level 35
3.5.4. Trigger distribution 36
3.6. FRONT PANEL I/OS 36
3.7. ANALOG MONITOR 37
3.7.1. Trigger Majority Mode (Monitor Mode = 0) 37
3.7.2. Test Mode (Monitor Mode = 1) 38
3.7.3. Buffer Occupancy Mode (Monitor Mode = 3) 38
3.7.4. Voltage Level Mode (Monitor Mode = 4) 38
3.8. TEST PATTERN GENERATOR 39
3.9. RESET, CLEAR AND DEFAULT CONFIGURATION 39
3.9.1. Global Reset 39
3.9.2. Memory Reset 39
3.9.3. Timer Reset 39
3.10. VMEBUS INTERFACE 39
3.10.1. Addressing capabilities 39
3.10.1.1. Base address 39
3.10.1.2. CR/CSR address 40
3.10.1.3. Address relocation 40
3.11. DATA TRANSFER CAPABILITIES 41
3.12. EVENTS READOUT 41
3.12.1. Sequential readout 41
3.12.1.1. SINGLE D32 42
3.12.1.2. BLOCK TRANSFER D32/D64, 2eVME 42
3.12.1.3. CHAINED BLOCK TRANSFER D32/D64 42
3.12.2. Random readout (to be implemented) 43
3.13. OPTICAL LINK 43
3.14. CAENVMELIB LIBRARY 44
3.14.1. CAENVME_Init 45
3.14.2. CAENVME_End 45
3.14.3. CAENVME_ReadCycle 45
3.14.4. CAENVME_WriteCycle 46
3.14.5. CAENVME_MultiRead 46
3.14.6. CAENVME_MultiWrite 46

3.14.7. CAENVME_BLTReadCycle 47
3.14.8. CAENVME_FIFOBLTReadCycle 47
3.14.9. CAENVME_MBLTReadCycle 47
3.14.10. CAENVME_FIFOMBLTReadCycle 48
3.14.11. CAENVME_IRQCheck 48
3.14.12. CAENVME_IRQEnable 49
3.14.13. CAENVME_IRQDisable 49
3.14.14. CAENVME_IRQWait 49

4. VME INTERFACE 50

4.1. REGISTERS ADDRESS MAP 50
4.2. CONFIGURATION ROM (0XF000-0XF084; R) 51
4.3. CHANNEL N ZS_THRES (0X1N24; R/W) 52
4.4. CHANNEL N ZS_NSAMP (0X1N28; R/W) 52
4.5. CHANNEL N THRESHOLD (0X1N80; R/W) 52
4.6. CHANNEL N OVER/UNDER THRESHOLD (0X1N84; R/W) 53
4.7. CHANNEL N STATUS (0X1N88; R) 53
4.8. CHANNEL N AMC FPGA FIRMWARE (0X1N8C; R) 53
4.9. CHANNEL N BUFFER OCCUPANCY (0X1N94; R) 53
4.10. CHANNEL N DAC (0X1N98; R/W) 53
4.11. CHANNEL N ADC CONFIGURATION (0X1N9C; R/W) 53
4.12. CHANNEL CONFIGURATION (0X8000; R/W) 54
4.13. CHANNEL CONFIGURATION BIT SET (0X8004; W) 54
4.14. CHANNEL CONFIGURATION BIT CLEAR (0X8008; W) 54
4.15. BUFFER ORGANIZATION (0X800C; R/W) 54
4.16. BUFFER FREE (0X8010; R/W) 55
4.17. CUSTOM SIZE (0X8020; R/W) 55
4.18. ACQUISITION CONTROL (0X8100; R/W) 55
4.19. ACQUISITION STATUS (0X8104; R) 56
4.20. SOFTWARE TRIGGER (0X8108; W) 56
4.21. TRIGGER SOURCE ENABLE MASK (0X810C; R/W) 56
4.22. FRONT PANEL TRIGGER OUT ENABLE MASK (0X8110; R/W) 57
4.23. POST TRIGGER SETTING (0X8114; R/W) 58
4.24. FRONT PANEL I/O DATA (0X8118; R/W) 58
4.25. FRONT PANEL I/O CONTROL (0X811C; R/W) 58
4.26. CHANNEL ENABLE MASK (0X8120; R/W) 58
4.27. ROC FPGA FIRMWARE REVISION (0X8124; R) 59
4.28. EVENT STORED (0X812C; R) 59
4.29. SET MONITOR DAC (0X8138; R/W) 60

4.30. BOARD INFO (0X8140; R) 60
4.31. MONITOR MODE (0X8144; R/W) 60
4.32. EVENT SIZE (0X814C; R) 60
4.33. VME CONTROL (0XEF00; R/W) 60
4.34. VME STATUS (0XEF04; R) 61
4.35. BOARD ID (0XEF08; R/W) 61
4.36. MCST BASE ADDRESS AND CONTROL (0XEF0C; R/W) 61
4.37. RELOCATION ADDRESS (0XEF10; R/W) 61
4.38. INTERRUPT STATUS ID (0XEF14; R/W) 61
4.39. INTERRUPT EVENT NUMBER (0XEF18; R/W) 62
4.40. BLT EVENT NUMBER (0XEF1C; R/W) 62
4.41. SCRATCH (0XEF20; R/W) 62
4.42. SOFTWARE RESET (0XEF24; W) 62
4.43. SOFTWARE CLEAR (0XEF28; W) 62
4.44. FLASH ENABLE (0XEF2C; R/W) 62
4.45. FLASH DATA (0XEF30; R/W) 62
4.46. CONFIGURATION RELOAD (0XEF34; W) 62

5. INSTALLATION 63

5.1. POWER ON SEQUENCE 63
5.2. POWER ON STATUS 63
5.3. FIRMWARE UPGRADE 63
5.3.1. V1720 Upgrade files description 64

LIST OF FIGURES

FIG. 1.1: MOD. V1720 BLOCK DIAGRAM 9
FIG. 2.1: MOD. V1720 FRONT PANEL 11
FIG. 2.2: MCX CONNECTOR 12
FIG. 2.3: AMP DIFFERENTIAL CONNECTOR 12
FIG. 2.4: AMP CLK IN/OUT CONNECTOR 13
FIG. 2.5: PROGRAMMABLE IN/OUT CONNECTOR 13
FIG. 2.6: LC OPTICAL CONNECTOR 14
FIG. 2.7: ROTARY AND DIP SWITCHES LOCATION 15
FIG. 3.1: SINGLE ENDED INPUT DIAGRAM 17
FIG. 3.2: DIFFERENTIAL INPUT DIAGRAM 17
FIG. 3.3: CLOCK DISTRIBUTION DIAGRAM 18

FIG. 3.4: CAENPLLCONFIG MAIN MENU 20
FIG. 3.5: SAMPLING CLOCK PHASE SHIFT 21
FIG. 3.6: DATA STORAGE 23
FIG. 3.7: TRIGGER OVERLAP 24
FIG. 3.8: EVENT ORGANIZATION (STANDARD MODE), NORMAL FORMAT 26
FIG. 3.9:EVENT ORGANIZATION (PACK2.5 MODE), NORMAL FORMAT 27
FIG. 3.10: EVENT ORGANIZATION (STANDARD MODE), ZERO LENGTH ENCODING 27
FIG. 3.11: EVENT ORGANIZATION (PACK2.5 MODE), ZERO LENGTH ENCODING 28
FIG. 3.12: ZERO SUPPRESSION EXAMPLE 30
FIG. 3.13: EXAMPLE WITH POSITIVE LOGIC AND NON-OVERLAPPING N_{LBK} / N_{LFWD} 30
FIG. 3.14: EXAMPLE WITH NEGATIVE LOGIC AND NON-OVERLAPPING N_{LBK} / N_{LFWD} 31
FIG. 3.15: EXAMPLE WITH POSITIVE LOGIC AND NON OVERLAPPING N_{LBK} 32
FIG. 3.16: EXAMPLE WITH POSITIVE LOGIC AND OVERLAPPING N_{LBK} 33
FIG. 3.17: BLOCK DIAGRAM OF TRIGGER MANAGEMENT 34
FIG. 3.18: LOCAL TRIGGER GENERATION 35
FIG. 3.19: LOCAL TRIGGER RELATIONSHIP WITH COINCIDENCE LEVEL 36
FIG. 3.20: MAJORITY LOGIC (2 CHANNELS OVER THRESHOLD; BIT[6] OF CH. CONFIG. REGISTER =0) 38
FIG. 3.21: A24 ADDRESSING 40
FIG. 3.22: A32 ADDRESSING 40
FIG. 3.23: CR/CSR ADDRESSING 40
FIG. 3.24: SOFTWARE RELOCATION OF BASE ADDRESS 40
FIG. 3.25: EXAMPLE OF BLT READOUT 42
FIG. 3.26: EXAMPLE OF RANDOM READOUT 43
FIG. 3.27: OPTICAL LINK DAISY CHAIN 44

LIST OF TABLES

TABLE 1.1: MOD. V1720 VERSIONS 8
TABLE 2.1: MOD. V1720 POWER REQUIREMENTS 10
TABLE 2.2: FRONT PANEL LEDS 14
TABLE 2.3: MOD. V1720 TECHNICAL SPECIFICATIONS 16
TABLE 3.1: BUFFER ORGANIZATION 23
TABLE 3.2: FRONT PANEL I/OS DEFAULT SETTING 36
TABLE 4.1: ADDRESS MAP FOR THE MODEL V1720 50
TABLE 4.2: ROM ADDRESS MAP FOR THE MODEL V1720 51
TABLE 4.3: OUTPUT BUFFER MEMORY BLOCK DIVISION 54

1. General description

1.1. Overview

The Mod. V1720 is a 1-unit wide VME 6U module housing a 8 Channel 12 bit 250MS/s Flash ADC Waveform Digitizer with threshold Auto-Trigger capabilities.
The boards are available with different input range, memory and connector configuration, as summarised by the following table:

Table 1.1: Mod. V1720 versions

Model	Input type	Sampling frequency	SRAM Memory	Form factor
V1720	Single ended	250MS/s	1.25 Msamples / ch	6U-VME64
V1720B	Single ended	250MS/s	10 Msamples / ch	6U-VME64
V1720C	Differential	250MS/s	1.25 Msamples / ch	6U-VME64
V1720D	Differential	250MS/s	10 Msamples / ch	6U-VME64
VX1720	Single ended	250MS/s	1.25 Msamples / ch	6U-VME64X
VX1720B	Single ended	250MS/s	10 Msamples / ch	6U-VME64X
VX1720C	Differential	250MS/s	1.25 Msamples / ch	6U-VME64X
VX1720D	Differential	250MS/s	10 Msamples / ch	6U-VME64X

Reported Memory values can be achieved by enabling the Pack2.5 option, which allows to write “two and a half” samples in a 32 bit longword (see § 3.3.4).

The DC offset of the signal can be adjusted channel per channel by means of a programmable 16bit DAC.

The board features a front panel clock/reference In/Out and a PLL for clock synthesis from internal/external references. This allows multi board phase synchronizations to an external clock source or to a V1720 clock master board.

The data stream is continuously written in a circular memory buffer; when the trigger occurs the FPGA writes further N samples for the post trigger and freezes the buffer that then can be read either via VME or via Optical Link; the acquisition can continue without dead-time in a new buffer. Each channel has a SRAM memory, divided in buffers of programmable size.

VME and Optical Link accesses take place on independent paths and are handled by the on-board controller, therefore when accessed through Optical Link the board can be operated outside the VME Crate (see § 2.1).

The trigger signal can be provided via the front panel input as well as via the VMEbus, but it can also be generated internally, as soon as a programmable voltage threshold is reached. The individual Auto-Trigger of one channel can be propagated to the other channels and onto the front panel Trigger Output.

The VME interface is VME64X compliant and the data readout can be performed in Single Data Transfer (D32), 32/64 bit Block Transfer (BLT/MBLT), 2eVME, 2eSST and 32/64 bit Chained Block Transfer (CBLT).

The board houses a daisy chainable Optical Link able to transfer data at 80 MB/s, thus it is possible to connect up to eight V1720 (64 ADC channels) to a single Optical Link Controller (Mod. A2818, see Accessories/Controller).
The V1720 can be controlled and readout through the Optical Link in parallel to the VME interface.

1.2. Block Diagram

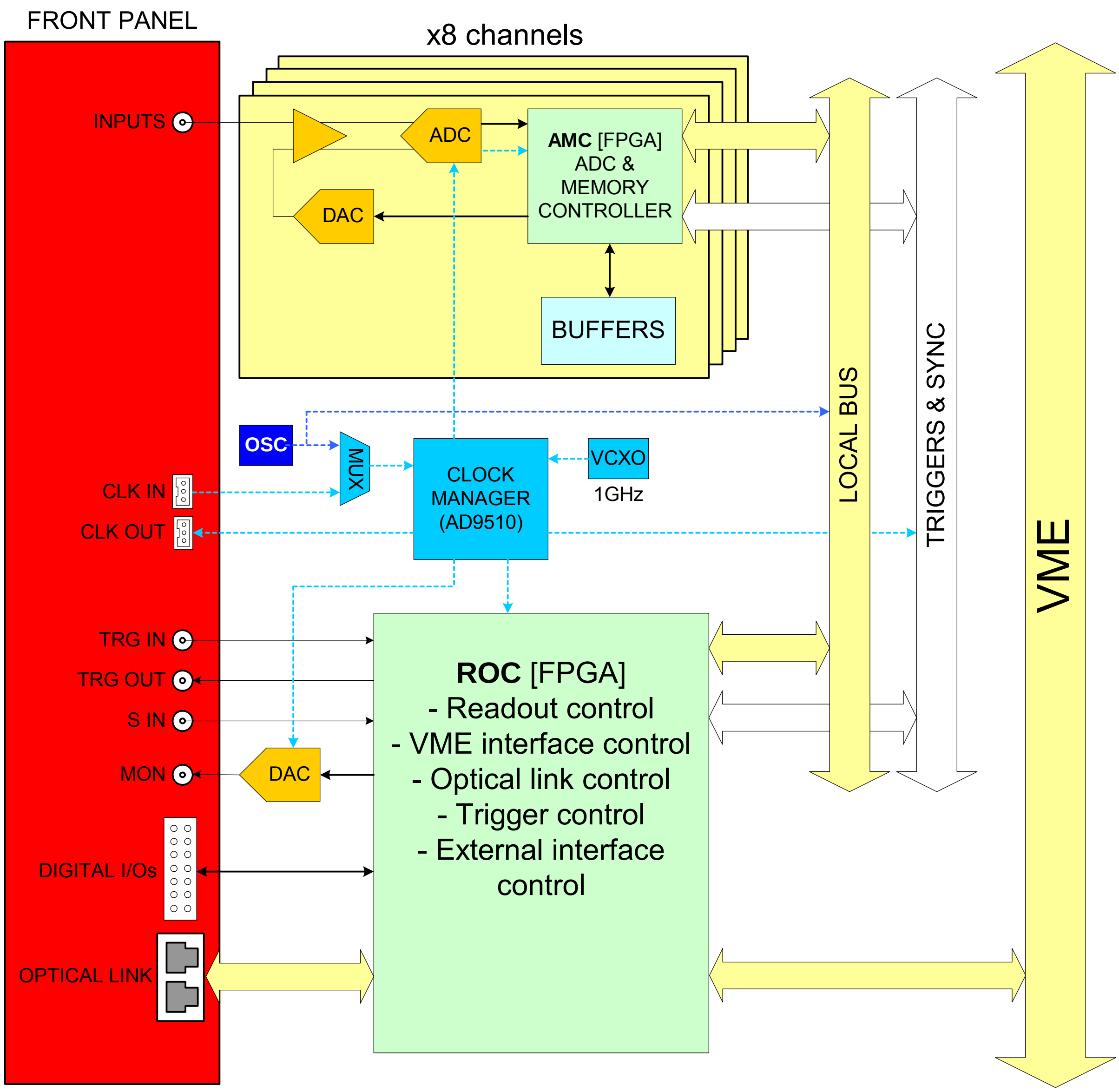

Fig. 1.1: Mod. V1720 Block Diagram

The function of each block will be explained in detail in the subsequent sections.

2. Technical specifications

2.1. Packaging and Compliancy

2.1.1. *Supported VME Crates*

The module is housed in a 6U-high, 1U-wide VME unit. The board hosts the VME P1, and P2 connectors and fits into both VME/VME64 standard and V430 backplanes.
VX1720 versions fit VME64X compliant crates.

2.1.2. *Stand Alone operation*

When accessed through Optical Link (see § 3.13) the board can be operated outside the VME Crate. It is up to the User to provide the required power supplies (see § 2.2) and adequate cooling ventilation.

2.2. Power requirements

The power requirements of the module are as follows:

Table 2.1: Mod. V1720 power requirements

+5 V	**4.0 A**
+12 V	**0.2 A**
-12 V	**0.2 A**

2.3. Front Panel

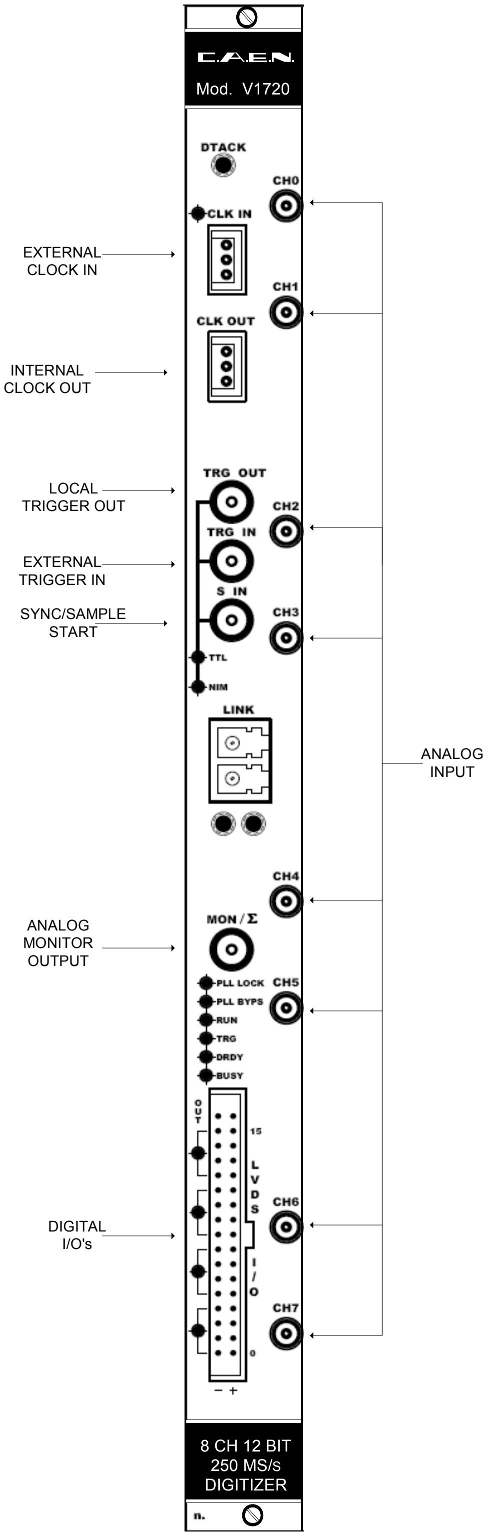

Fig. 2.1: Mod. V1720 front panel

2.4. External connectors

2.4.1. *ANALOG INPUT connectors*

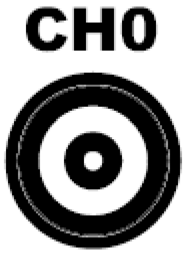

Fig. 2.2: MCX connector

Single ended version (see options in § 1.1):
Function:
Analog input, single ended, input dynamics: 2Vpp Zin=50Ω
Mechanical specifications:
MCX connector (CS 85MCX-50-0-16 SUHNER)

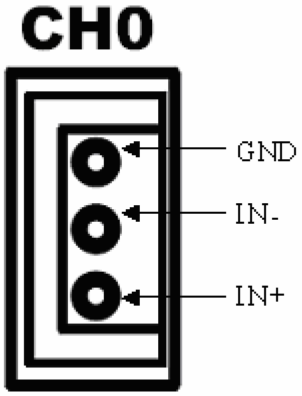

Fig. 2.3: AMP Differential connector

Differential version (see options in § 1.1):
Function:
Analog input, differential, input dynamics: 2.25Vpp Zin=100Ω or 10Vpp Zin=1KΩ
Mechanical specifications:
AMP 3-102203-4 AMP MODUII

2.4.2. *CONTROL connectors*

Function:
- TRG OUT: Local trigger output (NIM/TTL, on Rt = 50Ω)
- TRG IN: External trigger input (NIM/TTL, Zin= 50Ω)
- SYNC/SAMPLE/START: Sample front panel input (NIM/TTL, Zin=50Ω)
- MON/Σ: DAC output 1Vpp on Rt=50Ω

Mechanical specifications:
00-type LEMO connectors

2.4.3. ADC REFERENCE CLOCK connectors

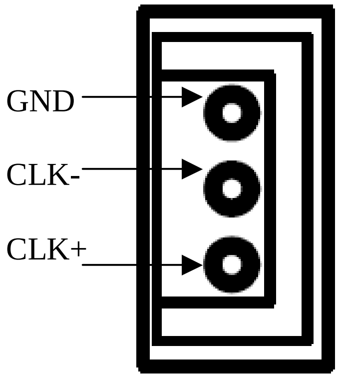

Fig. 2.4: AMP CLK IN/OUT Connector

Function:
CLK IN: External clock/Reference input, AC coupled (diff. LVDS, ECL, PECL, LVPECL, CML), Zdiff= 110Ω.
Mechanical specifications:
AMP 3-102203-4 connector
Function:
CLOCK OUT: Clock output, DC coupled (diff. LVDS), Zdiff= 110Ω.
Mechanical specifications:
AMP 3-102203-4 AMP MODUII

2.4.4. Digital I/O connectors

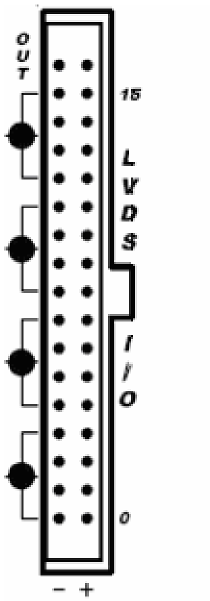

Fig. 2.5: Programmable IN/OUT Connector

Function: N.16 programmable differential LVDS I/O signals, Zdiff_in= 110 Ohm. Four Indipendent signal group 0÷3, 4÷7, 8÷11, 12÷15, In / Out direction control; see also § 3.6.
Mechanical specifications: 3M-7634-5002- 34 pin Header Connector

2.4.5. Optical LINK connector

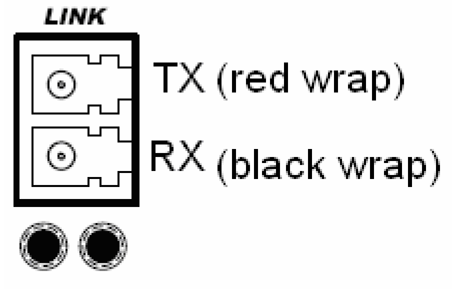

Fig. 2.6: LC Optical Connector

Mechanical specifications:
LC type connector; to be used with Multimode 62.5/125μm cable with LC connectors on both sides
Electrical specifications:
Optical link for data readout and slow control with transfer rate up to 80MB/s; daisy chainable.

2.5. Other front panel components

2.5.1. Displays

The front panel hosts the following LEDs:

Table 2.2: Front panel LEDs

Name:	Colour:	Function:
DTACK	green	VME read/write access to the board
CLK_IN	green	External clock enabled.
NIM	green	Standard selection for CLK I/O, TRG OUT, TRG IN, S IN.
TTL	green	Standard selection for CLK I/O, TRG OUT, TRG IN, S IN.
LINK	green/yellow	Network present; Data transfer activity
PLL _LOCK	green	The PLL is locked to the reference clock
PLL _BYPS	green	The reference clock drives directly ADC clocks; the PLL circuit is switched off and the PLL_LOCK LED is turned off.
RUN	green	RUN bit set (see § 4.19)
TRG	green	Triggers are accepted
DRDY	green	Event/data (depending on acquisition mode) are present in the Output Buffer
BUSY	red	All the buffers are full
OUT_LVDS	green	Signal group OUT direction enabled.

2.6. Internal components

SW2,4,5,6 "Base Addr. [31:16]": *Type:* 4 rotary switches
Function: Set the VME base address of the module.

SW3 "CLOCK SOURCE" Type Dip Switch
Function: Select clock source (External or Internal)

SW1 "FW" Type: Dip Switch.

Function: it allows to select whether the "Standard" (STD) or the "Back up" (BKP) firmware must be loaded at power on; (default position: STD).

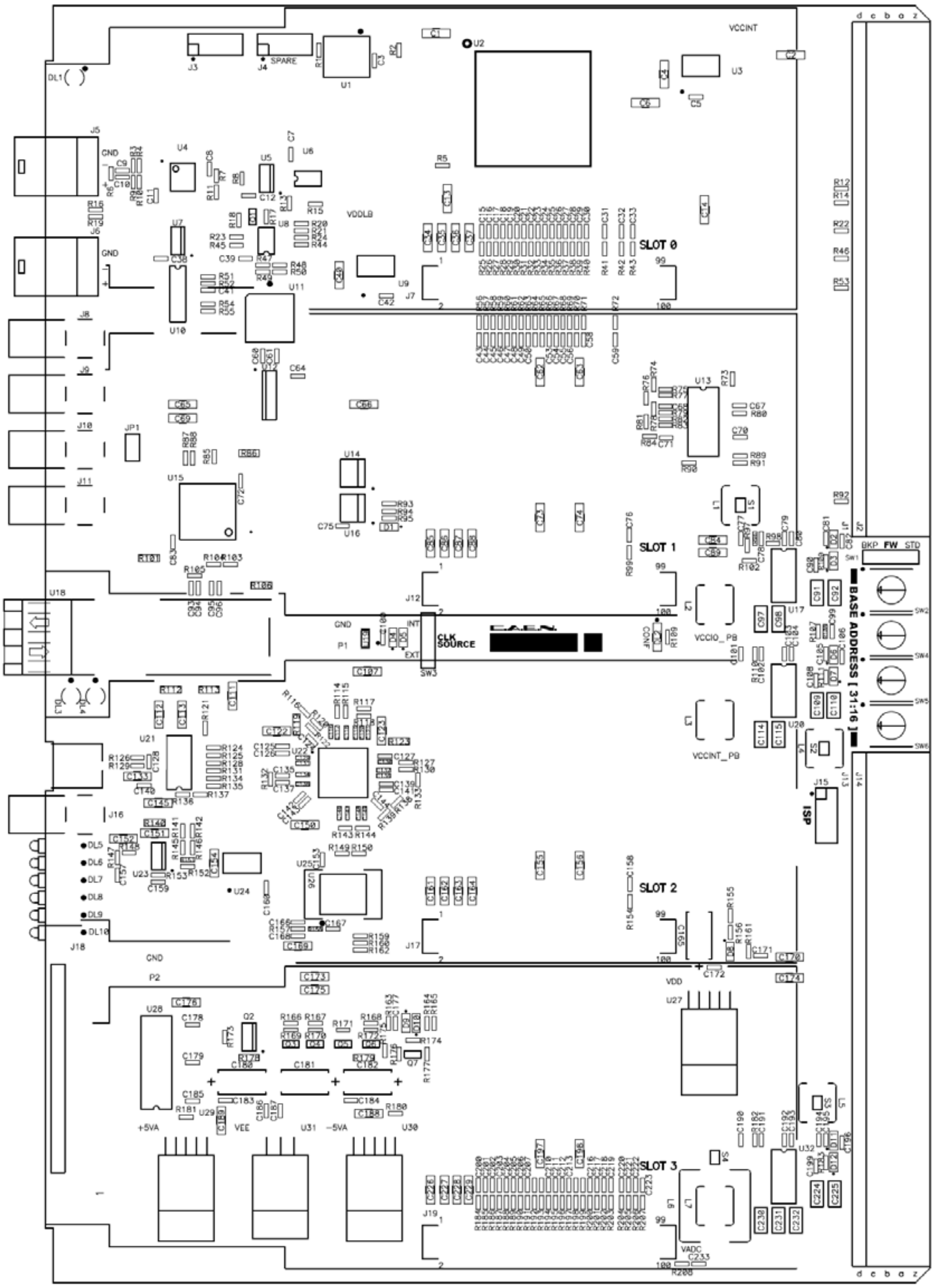

Fig. 2.7: Rotary and dip switches location

2.7. Technical specifications table

Table 2.3: Mod. V1720 technical specifications

Package	1-unit wide VME 6U module
Analog Input	8 channels, single-ended (SE) or differential. Input range: 2 Vpp; Bandwidth: 125 MHz. Programmable DAC for Offset Adjust x ch. (SE only).
Digital Conversion	Resolution: 12 bit; Sampling rate: 10 to 250 MS/s simultaneously on each channel; multi board synchronization (one board can act as clock master). External Gate Clock capability (NIM/TTL) for burst or single sampling mode.
ADC Sampling Clock generation	Three operating modes: - PLL mode - internal reference (50 MHz loc. oscillator). - PLL mode - external reference on CLK_IN (Jitter<100ppm). - PLL Bypass mode: Ext. clock on CLK_IN drives directly ADC clocks (Freq.: 10 ÷ 250 MHz).
CLK_IN	AC coupled differential input clock LVDS, ECL, PECL, LVPECL, CML
CLK_OUT	DC coupled differential LVDS output clock, locked to ADC sampling clock. Freq.: 10 ÷ 250MHz.
Memory Buffer	1.25 M sample/ch or 10 M sample/ch; Multi Event Buffer with independent read and write access. Programmable event size and pre-post trigger. Divisible into 1 ÷ 1024 buffers.
Trigger	Common External TRGIN (NIM or TTL) and VME CommandIndividual channel autotrigger (time over/under threshold)TRGOUT (NIM or TTL) for the trigger propagation to other V1720 boards.
Trigger Time Stamp	32bit – 8ns (34s range). Sync input for Time Stamp alignment
AMC FPGA	One Altera Cyclone EP1C4 per channel
Optical Link	Data readout and slow control with transfer rate up to 80 MB/s, to be used instead of VME bus. Daisy chainable: one A2818 PCI card can control and read eight V1720 boards in a chain.
VME interface	VME64X compliant D32, BLT32, MBLT64, CBLT32/64, 2eVME, 2eSST, Multi Cast CyclesTransfer rate: 60MB/s (MBLT64), 100MB/s (2eVME), 160MB/s (2eSST). Sequential and random access to the data of the Multi Event Buffer. The Chained readout allows to read one event from all the boards in a VME crate with a BLT access.
Upgrade	V1720 firmware can be upgraded via VME
Software	General purpose C Libraries and Demo Programs (CAENScope).
Zero Suppression[1]	- Full Suppression based on the signal amplitude (ZS_AMP) - Zero Length Encoding (ZLE); see § 3.4
Analog Monitor *(to be implemented)*	12bit / 125MHz DAC FPGA controlled, five operating modes: - Waveform Generator: 1 Vpp ramp generator. - Majority: output signal is proportional to the number of ch. under/over threshold (1 step = 1.25mV). - Analog Inspection: data stream from one channel ADC drives directly the DAC input producing the channel input signal (1 Vpp). - Buffer Occupancy: output signal is proportional to the Multi Event Buffer Occupancy: 1 buffer ~ 1mV. - Voltage level: output signal is a programmable voltage level.
LVDS I/O	16 general purpose LVDS I/O controlled by the FPGA Busy, Data Ready, Memory full, Individual Trig-Out and other function can be programmed An Input Pattern from the LVDS I/O can be associated to each trigger as an event marker

[1] Available with Piggy Back Rev. 0.5 and Firmware Rev. 0.5

3. Functional description

3.1. Analog Input

The module is available either with single ended (on MCX connector) or, on request, differential (on Tyco MODU II 3-pin connector) input channels.

3.1.1. Single ended input

Input dynamics is 2V (Zin= 50 Ω). A 16bit DAC allows to add a DC offset to the signal in the ±1 V range.
The input bandwidth ranges from DC to 250 MHz (with 1st order anti-aliasing low pass filter).

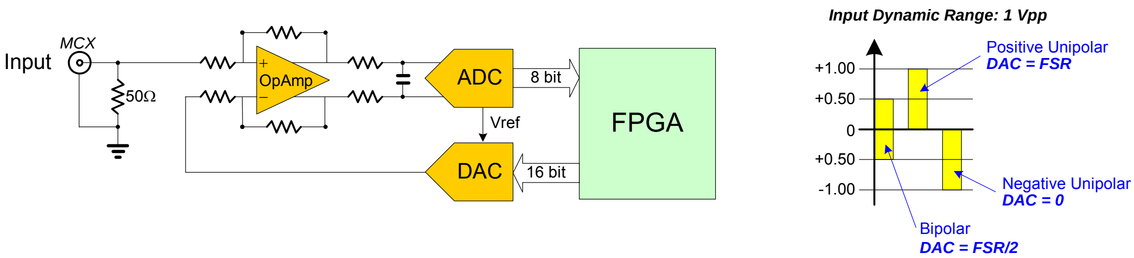

Fig. 3.1: Single ended input diagram

3.1.2. Differential input

Input dynamics is 2Vpp (Zin= 50 Ω).
The input bandwidth ranges from DC to 250 MHz (with 1st order anti-aliasing low pass filter).

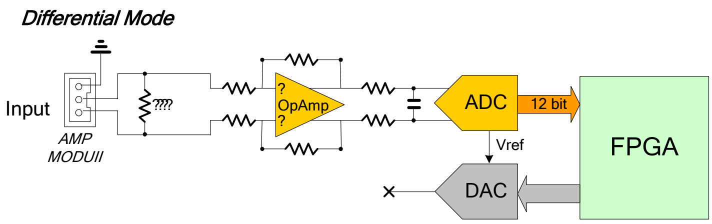

Fig. 3.2: Differential input diagram

3.2. Clock Distribution

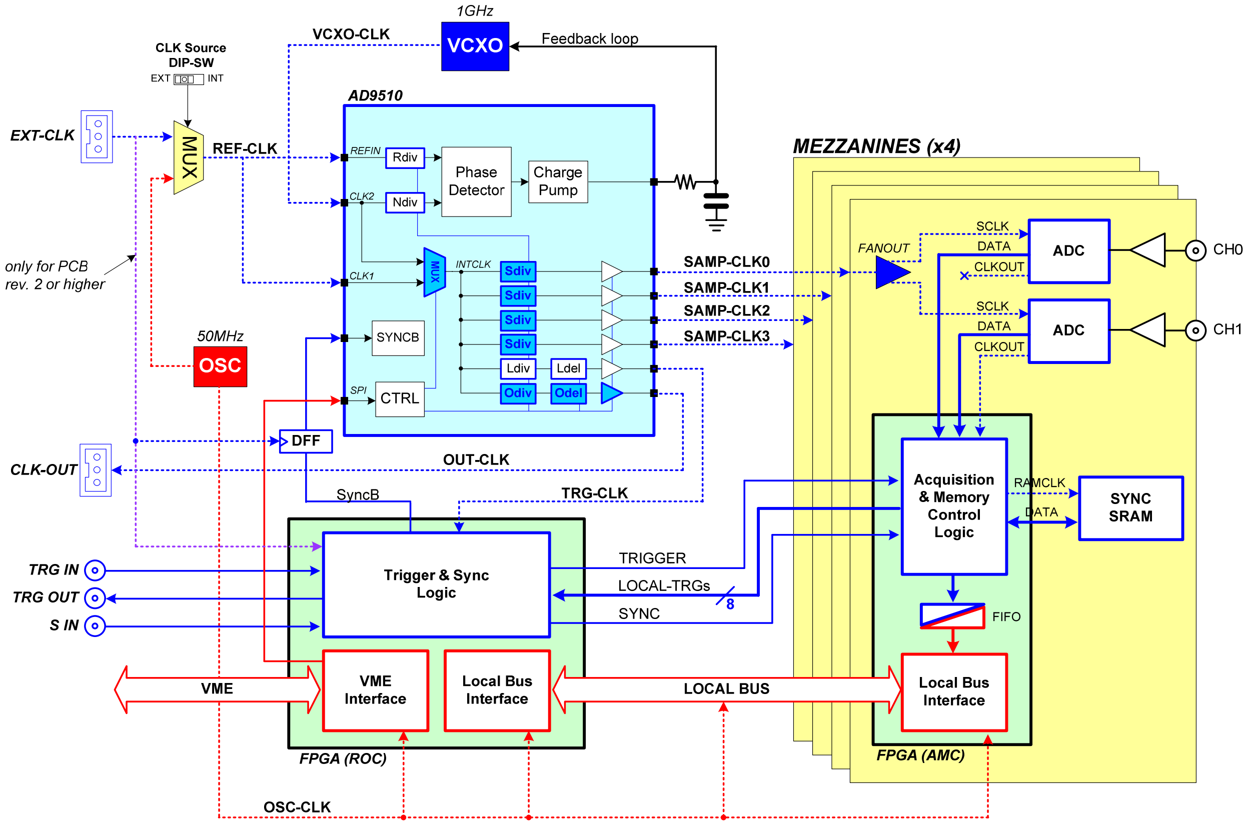

Fig. 3.3: Clock distribution diagram

The module clock distribution takes place on two domains: OSC-CLK and REF-CLK; the former is a fixed 50MHz clock provided by an on board oscillator, the latter provides the ADC sampling clock.
OSC-CLK handles both VME and Local Bus (communication between motherboard and mezzanine boards; see red traces in the figure above).
REF-CLK handles ADC sampling, trigger logic, acquisition logic (samples storage into RAM, buffer freezing on trigger) through a clock chain. Such domain can use either an external (via front panel signal) or an internal (via local oscillator) source (selection is performed via dip switch SW1, see § 2.6); in the latter case OSC-CLK and REF-CLK will be synchronous (the operation mode remains the same anyway).
REF-CLK is processed by AD9510 device, which delivers 6 clock out signals; 4 signals are sent to ADCs, one to the trigger logic and one to drive CLK-OUT output (refer to AD9510 data sheet for more details:
http://www.analog.com/UploadedFiles/Data_Sheets/AD9510.pdf); two operating modes are foreseen: ***Direct Drive Mode*** and ***PLL Mode***

3.2.1. Direct Drive Mode

The aim of this mode is to drive externally the ADCs' Sampling Clock; generally this is necessary when the required sampling frequency is not a VCXO frequency submultiple. The only requirement over the SAMP-CLK is to remain within the ADCs' range.

3.2.2. PLL Mode

The AD9510 features an internal Phase Detector which allows to couple REF-CLK with VCXO (1 GHz frequency); for this purpose it is necessary that REF-CLK is a submultiple of 1 GHz.
AD9510 default setting foresees the board internal clock (50MHz) as clock source of REF-CLK.
This configuration leads to *Ndiv* = 100, *Rdiv* = 5, thus obtaining 10MHz at the Phase Detector input and CLK-INT = 1GHz.
The required 250 MHz Sampling Clock is obtained by processing CLK-INT through Sdiv dividers.
When an external clock source is used, if it has 50MHz frequency, then AD9510 programming is not necessary, otherwise Ndiv and Rdiv have to be modified in order to achieve PLL lock.
A REF-CLK frequency stability better than 100ppm is mandatory.

3.2.3. Trigger Clock

TRG-CLK signal has a frequency equal to ½ of SAMP-CLK; therefore a 2 samples "uncertainty" occurs over the acquisition window.

3.2.4. Output Clock

Front panel Clock Output is User programmable. Odiv and Odel parameters allows to obtain a signal with the desired frequency and phase shift (in order to recover cable line delay) and therefore to synchronise daisy chained boards. CLK-OUT default setting is OFF, it is necessary to enable the AD9510 output buffer to enable it.

3.2.5. AD9510 programming

CAEN has developed a software tool which allows to handle easily the clock parameters:
CAENPLLConfig is a software tool which allows the PLL management, whenever the module is controlled through a CAEN VME Controller
(see http://www.caen.it/nuclear/function1.php?fun=vmecnt).
The tool is developed through open source classes wxWidgets v.2.6.3
(see http://www.wxwidgets.org/)
and requires the CAENVMETool API's to be installed
(they can be downloaded at http://www.caen.it/nuclear/lista-sw.php?mod=V1718 with the *SW package for CAEN VME Bridges & Slave Boards*).
CAENPLLConfig is available at: http://www.caen.it/nuclear/lista-sw.php?mod=V1720
And must be simply run on the PC connected to the used CAEN VME Controller
The User has to select the **board type** and **base address** (in the ADC BOARD field), then the used mode (**PLL** or **Direct Feed/BYPASS** in the INPUT field); see figure below:

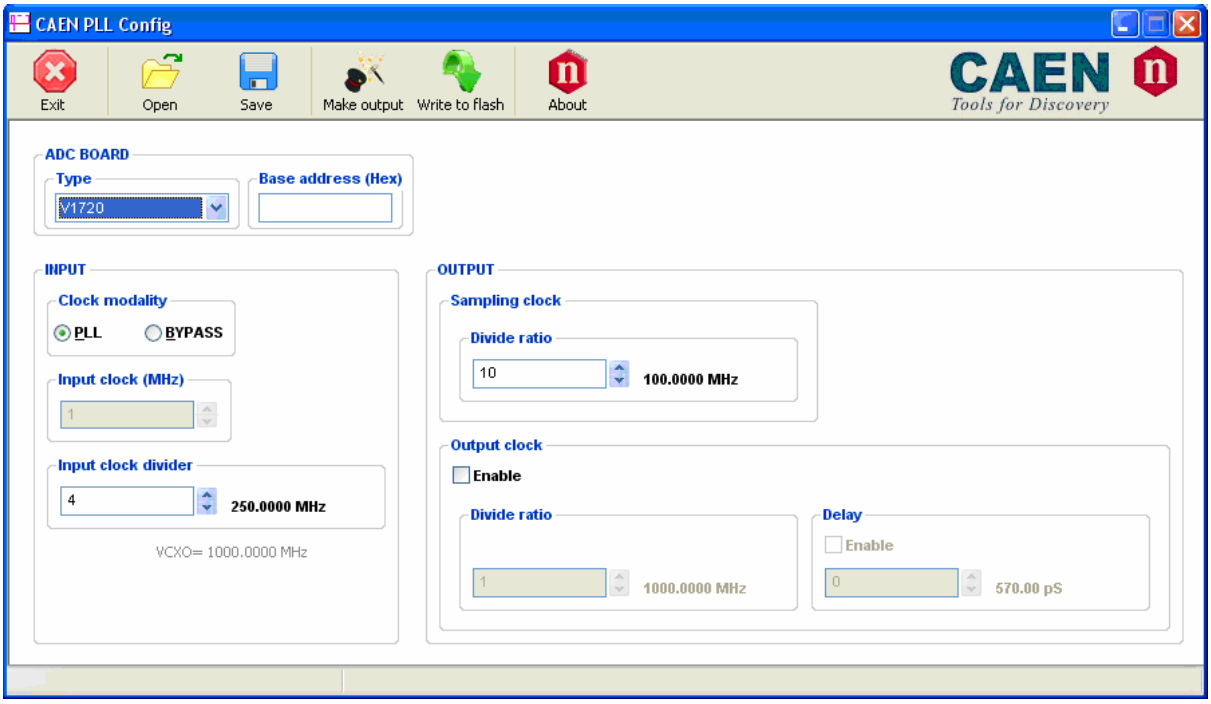

Fig. 3.4: CAENPLLConfig Main menu

3.2.6. PLL *programming*

In PLL mode the User has to enter the divider for input clock frequency (**input clock divider** field in CAENPLLConfig Main menu); since the VCXO frequency is 1GHz, in order to use, for example, a 50MHz ExtClk, the divider to be entered is 20.
Then it is necessary to set the parameters for sampling clock and CLK_OUT (**enable**, **divide ratio** and **phase shift/delay** in **Output Clock** field of CAENPLLConfig Main menu); the tool refuses wrong settings for such parameters.

3.2.7. Direct Drive *programming*

In Direct Drive/BYPASS mode, the User can directly set the input frequency (**Input Clock** field, real values are allowed). Given an input frequency, it is possible to set the parameters in order to provide the required signals.

3.2.8. *Configuration file*

Once all parameters are set, the tool allows to save the configuration file which includes all the AD9510 device settings (**SAVE** button in the upper toolbar of CAENPLLConfig Main menu). It is also possible to browse and load into the AD9510 device a pre existing configuration file (**OPEN** button in the upper toolbar of CAENPLLConfig Main menu). For this purpose it is not necessary the board power cycle.

3.2.9. *Multiboard synchronisation*

In order to allow several V1720s to work synchronously (same sampling clock for all channels) it is necessary to use the external clock. For such purpose, two solutions are possible

- a daisy chain where the clock is propagated from one board to another, with the first board used as a “clock master” (whose source could be either the internal clock or an external reference managed by the User),
- a tree structure, with an equalized clock distributor (fan-out unit with “low skew” outputs and constant cables length).

In both cases, the goal is to have all REF-CLK signals with the same phase. Since the PLL aligns the phase of VCXO output signal to REF-CLK, the result of synchronization is that all V1720s have the 1GHz VCXO output signals perfectly aligned in phase.
However, despite the V1720s having all the same 1GHz reference, it is not guaranteed that the sampling clock is in its turn aligned. In fact the use of clock dividers to produce the sampling clock, may lead such signals to have different phases, as shown in the following picture, where two 250MHz (divider = 4, see § 3.2.5) are obtained from a 1GHz VCXO output.

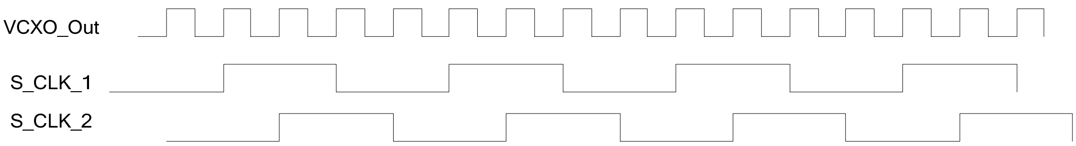

Fig. 3.5: Sampling clock phase shift

In order to keep all dividers outputs aligned, the AD9510 is provided with a SYNCB input (see § 3.2); all dividers are put in phase on a SYNCB edge. This is done automatically within a board at any board reset, therefore it is guaranteed that one board has the same sampling clock for all channels. However if it is necessary to synchronize sampling clock on more V1720s, then SYNCB signals have to be synchronized in their turn as well.
On modules with printed board Rev.2 (or greater), synchronization is achieved by piloting SYNCB through a D-Edge Triggered Flip Flop receiving EXT_CLK as clock input.
In this way it is ensured that the SYNCBs of all modules have the same phase. On modules with printed board Rev.1, however, the synchronization SYNCB can be obtained through the S-IN signal. In fact on S-IN leading edge, when the board is properly programmed (see § 3.3.1), the ROC FPGA sends a pulse on SYNCB. In order to avoid “uncertainty”, it is necessary that S-IN is sent to all the modules in phase with EXT-CLK: this will allow all V1720s to receive it with the same clock period.
After the synchronization of sampling clock signals, the modules will be also in phase with each other and all samples will be written into memory all at the same time.
However, in order to ensure that the windows of acquisition related to the external trigger signal are also perfectly aligned, it is also necessary that the TRG-IN signal is sent to all modules synchronously with EXT-CLK and in accordance with the setup time related to its leading edge.
In fact, if EXT-TRG is not correlated with EXT-CLK, a board might sense the trigger in a certain period of the clock while another might sense it in the subsequent. Therefore an uncertainty of 1 EXT-CLK period would occur (and then 1 SAMP-CLK hit) on the position of the acquired stored buffer with respect to the trigger arrival time.
The distribution of trigger can be simplified through the use of a daisy-chain: the external trigger signal is sent to the first board in the chain, and this, in coincidence with the TRG-IN received, gets triggered and generates a TRG-OUT which is in turn fed to the

adjacent board TRG-IN and so on. There is a fixed latency of few clock hits between TRG-IN and TRG-OUT, the value of this latency depends on the loaded firmware version; this latency, which spreads from board to board, can be easily rejected by acting on the value of the Post Trigger (see § 3.3) in order to have acquisition windows of all modules perfectly aligned.
If the external trigger entering the first board is asynchronous, then a one sample uncertainty occurs, as described above; when this uncertainty is resolved on the first board, all the other ones will be aligned to it. If a precise temporal relationship between trigger and samples is required (such as repeated acquisitions where a jitter on the position of the signal in the acquisition window is a major issue), it is suggested to use one input channel (among all the V1720s in the chain) to sample the trigger signal itself: this will allow to reconstruct off-line the trigger edge position in the acquisition window, with a resolution smaller than the sampling period (through interpolation).

3.3. Acquisition Modes

3.3.1. Acquisition run/stop

The acquisition can be started in two ways, according to Acquisition Control register Bits [1:0] setting (see § 4.18):

- setting the RUN/STOP bit (bit[2]) in the Acquisition Control register (bits [1:0] of Acquisition Control must be set to REGISTER-CONTROLLED RUN MODE or S-IN GATE MODE)
- driving S_IN signal high (bits [1:0] of Acquisition Control must be set to 01, S-IN CONTROLLED RUN MODE)

Subsequently acquisition is stopped either:

- resetting the RUN/STOP bit (bit[2]) in the Acquisition Control register (bits [1:0] of Acquisition Control must be set to REGISTER-CONTROLLED RUN MODE or S-IN GATE MODE)
- driving S_IN signal low (bits [1:0] of Acquisition Control set to 01, S-IN CONTROLLED RUN MODE)

3.3.2. Data acquisition

It is possible to use the S_IN signal (see § 2.4.2) as “gate” to enable samples storage. The samples produced by the 250 MHz ADC are stored in memory only if they are validated by the S_IN signal, otherwise they are rejected; data storage takes place by groups of 4 samples (two 32 bit long words) per time in normal operation, 5 samples per time by using Pack2.5 mode (see § 3.3.4). All the values sampled as the S-IN signal is active (high) are stored; for this purpose it is necessary to:
Set bits [1:0] of Acquisition Control register to S-IN GATE MODE
All the values sampled as the S-IN signal is active (high) are stored.

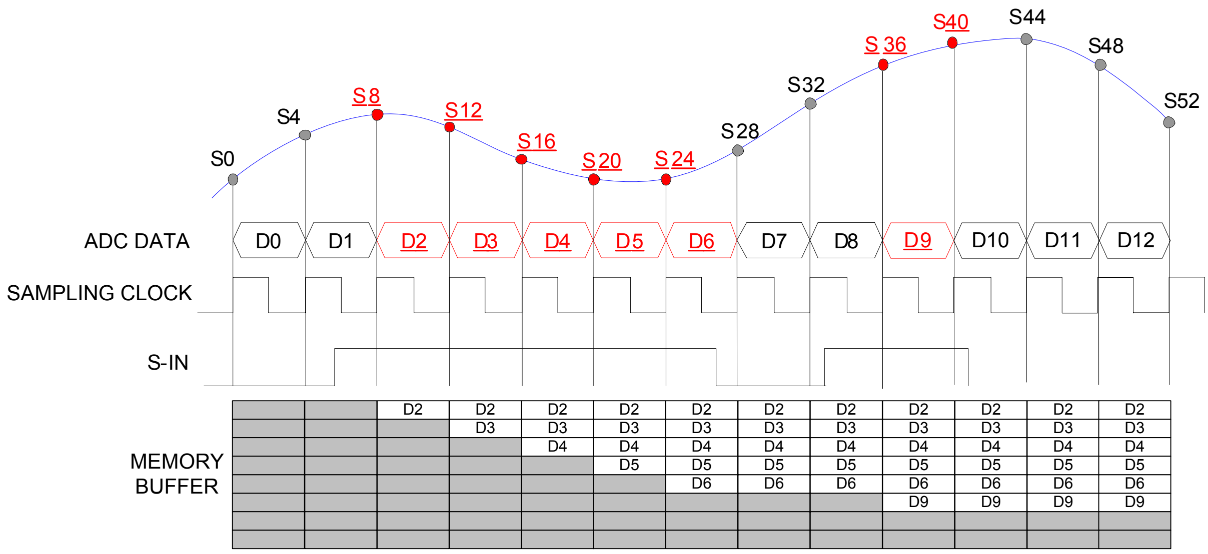

Fig. 3.6: Data storage

3.3.3. Acquisition Triggering: Samples and Events

When the acquisition is running, a trigger signal allows to:

- store a Trigger Time Tag (TTT): the value of a 32 bit counter which steps on with 125MHz frequency and represents a time reference
- increment the EVENT COUNTER (see § 4.28)
- fill the active buffer with the pre/post-trigger samples, whose number is programmable (Acquisition window width, § 4.23), freezing then the buffer for readout purposes, while acquisition continues on another buffer

Table 3.1: Buffer Organization

REGISTER (see § 4.15)	BUFFER NUMBER	SIZE of one BUFFER (samples)			
		SRAM 1.25 MB/ch		SRAM 10 MB/ch	
		Std.	Pack2.5	Std.	Pack2.5
0x00	1	1M	1.25M	8M	10M
0x01	2	512K	640K	4M	5M
0x02	4	256K	320K	2M	2.5M
0x03	8	128K	160K	1M	1.25M
0x04	16	64K	80K	512K	640K
0x05	32	32K	40K	256K	320K
0x06	64	16K	20K	128K	160K
0x07	128	8K	10K	64K	80K
0x08	256	4K	5K	32K	40K
0x09	512	2K	2.5K	16K	20K
0x0A	1024	1K	1.25K	8K	10K

An event is therefore composed by the trigger time tag, pre- and post-trigger samples and the event counter.
Overlap between "acquisition windows" may occur (a new trigger occurs while the board is still storing the samples related to the previous trigger); this overlap can be either rejected or accepted (programmable via VME).
If the board is programmed to accept the overlapped triggers, as the "overlapping" trigger arrives, the current active buffer is filled up, then the samples storage continues on the subsequent one.

In this case events will not have all the same size (see figure below).

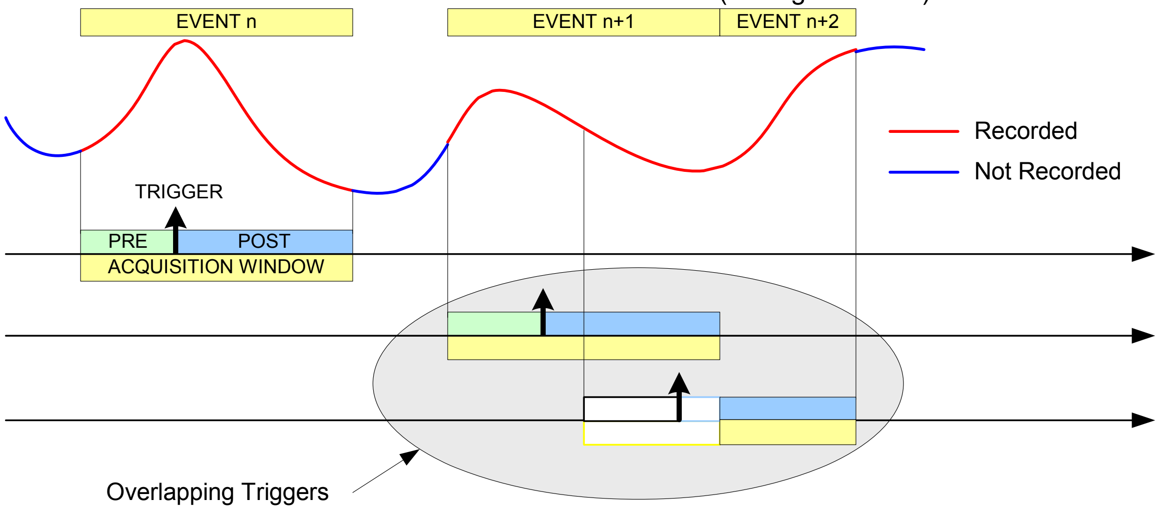

Fig. 3.7: Trigger Overlap

A trigger can be refused for the following causes:

- acquisition is not active
- memory is FULL and therefore there are no available buffers
- the required number of samples for building the pre-trigger of the event is not reached yet; this happens typically as the trigger occurs too early either with respect to the RUN_ACQUISITION command (see § 3.3.1) or with respect to a buffer emptying after a MEMORY_FULL status
- the trigger overlaps the previous one and the board is not enabled for accepting overlapped triggers

As a trigger is refused, the current buffer is not frozen and the acquisition continues writing on it. The Event Counter can be programmed in order to be either incremented or not. If this function is enabled, the Event Counter value identifies the number of the triggers sent (but the event number sequence is lost); if the function is not enabled, the Event Counter value coincides with the sequence of buffers saved and readout.

3.3.3.1. Custom size events

It is possible to make events with a number of Memory locations, which depends on Buffer Organization register setting (see § 4.15) smaller than the default value. One memory location contains two ADC samples and the maximum number of memory locations N_{LOC} is therefore half the maximum number of samples per block NS = 512K/Nblocks (640K/Nblocks when Pack2.5 mode is used).

Smaller N_{LOC} values can be achieved by writing the number of locations N_{LOC} into the Custom Size register (see § 4.17).

N_{LOC} = 0 means “default size events”, i.e. the number of memory locations is the maximum allowed.

N_{LOC} = N1, with the constraint 0<N1<½NS (0<N1<2/5NS with Pack2.5), means that one event will be made of 2·N1 samples (2.5·N1 samples with Pack2.5).

3.3.4. Event structure

An event is structured as follows:

- Header (4 32-bit words)

– Data (variable size and format)

The event can be readout either via VME or Optical Link; data format is 32 bit long word, therefore each long_word contains 4 samples.

3.3.4.1. Header

It is composed by four words, namely:

– Size of the event (number of 32 bit long words)
– Board ID (GEO); Bit24; data format: 0= normal format; 1= *Zero Length Encoding* data compression method enabled (*To be implemented*); 16 bit pattern, latched on the LVDS I/O as one trigger arrives (see § 4.25); Channel Mask (=1: channels participating to event; ex CH5 and CH7 participating→Ch Mask: 0xA0, this information must be used by the software to acknowledge which channel the samples are coming from)
– Event Counter: It is the trigger counter; it can count either accepted triggers only, or all triggers (see § 4.17).
– Trigger Time Tag: It is a 32 bit counter (31 bit count + 1 overflow bit), which is reset either as acquisition starts or via front panel Reset signal (see § 3.8), and is incremented at each sampling clock hit. It is the trigger time reference.Event structure

3.3.4.2. Samples

Stored samples; data from masked channels are not read.

3.3.4.3. Event format examples

The event format is shown in the following figure (case of 8 channels enabled, with *Zero Length Encoding* disabled and enabled respectively; see § 3.3.4.1 and § 3.4.1.2):
An event is structured as follows:

– identifier (Trigger Time Tag, Event Counter)
– samples caught in the acquisition windows

The event can be stored in the board memories (and can be readout via VME) in two ways: data format is 32 bit long word, and each long_word may contain 2 samples (Standard mode) or "two and a half" (Pack2.5 mode), depending on Channel Configuration register setting (see § 4.12).
The event format is therefore one of the following:

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	
1 0 1 0 EVENT SIZE	EVENT SIZE	HEADER
BOARD-ID RES 0 PATTERN	PATTERN CHANNEL MASK	HEADER
reserved EVENT COUNTER	EVENT COUNTER	HEADER
TRIGGER TIME TAG	TRIGGER TIME TAG	HEADER
0 0 0 0 SAMPLE [1] - CH[0]	0 0 0 0 SAMPLE [0] - CH[0]	DATA CH0
0 0 0 0 SAMPLE [3] - CH[0]	0 0 0 0 SAMPLE [2] - CH[0]	DATA CH0
• • •	• • •	DATA CH0
0 0 0 0 SAMPLE [N-1] - CH[0]	0 0 0 0 SAMPLE [N-2] - CH[0]	DATA CH0
0 0 0 0 SAMPLE [1] - CH[1]	0 0 0 0 SAMPLE [0] - CH[1]	DATA CH1
0 0 0 0 SAMPLE [3] - CH[1]	0 0 0 0 SAMPLE [2] - CH[1]	DATA CH1
• • •	• • •	DATA CH1
0 0 0 0 SAMPLE [N-1] - CH[1]	0 0 0 0 SAMPLE [N-2] - CH[1]	DATA CH1
• • •	• • •	• • •
0 0 0 0 SAMPLE [1] - CH[7]	0 0 0 0 SAMPLE [0] - CH[7]	DATA CH7
0 0 0 0 SAMPLE [3] - CH[7]	0 0 0 0 SAMPLE [2] - CH[7]	DATA CH7
• • •	• • •	DATA CH7
0 0 0 0 SAMPLE [N-1] - CH[7]	0 0 0 0 SAMPLE [N-2] - CH[7]	DATA CH7

Fig. 3.8: Event Organization (standard mode), normal format

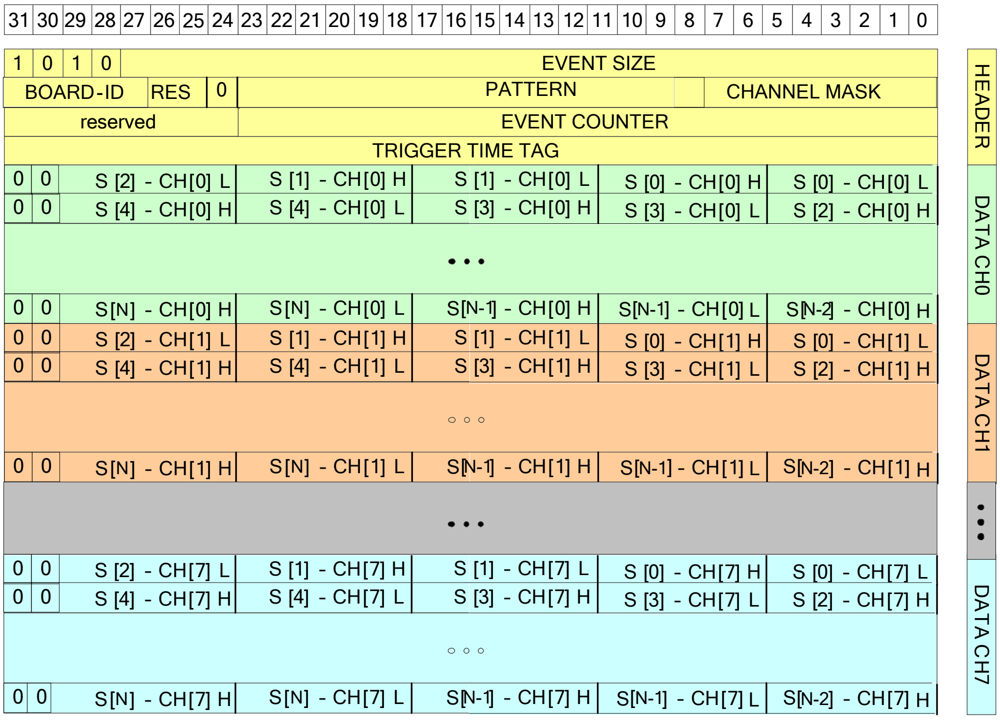

Fig. 3.9:Event Organization (Pack2.5 mode), normal format

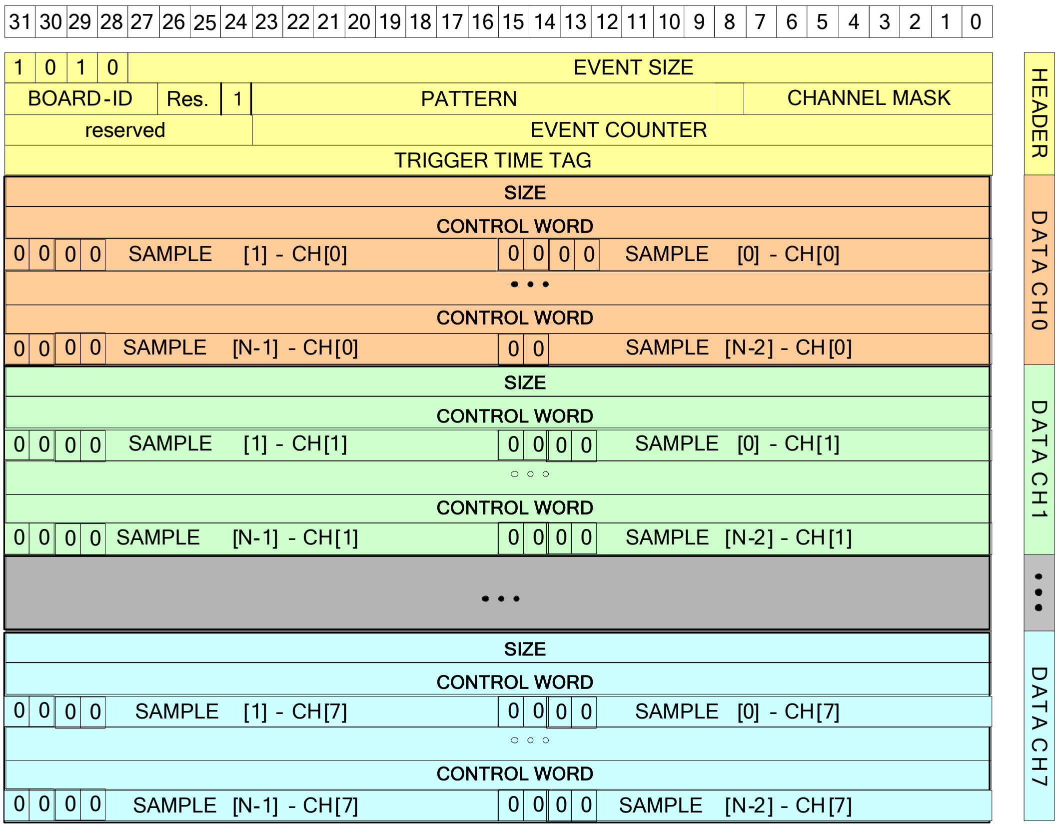

Fig. 3.10: Event Organization (standard mode), Zero Length Encoding

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

HEADER	1	0	1	0	EVENT SIZE		
	BOARD ID	RES.	1	PATTERN	CHANNEL MASK		
	RESERVED	EVENT COUNTER					
	TRIGGER TIME TAG						
DATA CHANNEL 0	SIZE						
	CONTROL WORD						
	0	0	S(2) - CH(0) L	S(1) - CH(0) H	S(1) - CH(0) L	S(0) - CH(0) H	S(0) - CH(0) L
	0	0	S(4) - CH(0) H	S(4) - CH(0) L	S(3) - CH(0) H	S(3) - CH(0) L	S(2) - CH(0) H
	.....						
	CONTROL WORD						
	0	0	S(N-2) - CH(0) L	S(N-3) - CH(0) H	S(N-3) - CH(0) L	S(N-4) - CH(0) H	S(N-4) - CH(0) L
	0	0	S(N) - CH(0) H	S(N) - CH(0) L	S(N-1) - CH(0) H	S(N-1) - CH(0) L	S(N-2) - CH(0) H
DATA CHANNEL 1	SIZE						
	CONTROL WORD						
	0	0	S(2) - CH(1) L	S(1) - CH(1) H	S(1) - CH(1) L	S(0) - CH(1) H	S(0) - CH(1) L
	0	0	S(4) - CH(1) H	S(4) - CH(1) L	S(3) - CH(1) H	S(3) - CH(1) L	S(2) - CH(1) H
	.....						
	CONTROL WORD						
	0	0	S(N-2) - CH(1) L	S(N-3) - CH(1) H	S(N-3) - CH(1) L	S(N-4) - CH(1) H	S(N-4) - CH(1) L
	0	0	S(N) - CH(1) H	S(N) - CH(1) L	S(N-1) - CH(1) H	S(N-1) - CH(1) L	S(N-2) - CH(1) H
.....	.....						
DATA CHANNEL 7	SIZE						
	CONTROL WORD						
	0	0	S(2) - CH(7) L	S(1) - CH(7) H	S(1) - CH(7) L	S(0) - CH(7) H	S(0) - CH(7) L
	0	0	S(4) - CH(7) H	S(4) - CH(7) L	S(3) - CH(7) H	S(3) - CH(7) L	S(2) - CH(7) H
	.....						
	CONTROL WORD						
	0	0	S(N-2) - CH(7) L	S(N-3) - CH(7) H	S(N-3) - CH(7) L	S(N-4) - CH(7) H	S(N-4) - CH(7) L
	0	0	S(N) - CH(7) H	S(N) - CH(7) L	S(N-1) - CH(7) H	S(N-1) - CH(7) L	S(N-2) - CH(7) H

Fig. 3.11: Event Organization (Pack2.5 mode), Zero Length Encoding

3.4. Zero suppression

The board implements two algorithms of "Zero Suppression" and "Data Reduction"[2]
- Full Suppression based on the signal amplitude (ZS_AMP),
- Zero Length Encoding (ZLE),

The algorithm to be used is selected via Control register, and its configuration takes place via two more registers (CHANNEL n ZS_THRES and CHANNEL n ZS_NSAMP).
It must be noticed that one datum (64 bit long word) contains 4 samples (5 samples with Pack2.5 mode): therefore one datum is considered over threshold as at least one sample reaches (or exceeds) the threshold.
As a consequence, one datum is considered under threshold as all 4 samples (5 samples with Pack2.5 mode) remain smaller than the threshold.

3.4.1. Zero Suppression Algorithm

3.4.1.1. Full Suppression based on the amplitude of the signal

Full Suppression based on the signal amplitude allows to discard a full event if the signal does not exceed the programmed threshold for Ns subsequent data at least (Ns is programmable, see § 4.4).
It is also possible to configure the algorithm with "negative" logic: in this case the event is discarded if the signal does not remain under the programmed threshold for Ns subsequent data at least.

3.4.1.2. Zero Length Encoding ZLE

Zero Length Encoding allows to transfer the event in compressed mode, discarding either the data under the threshold set by the User (positive logic) or the data over the threshold set by the User (negative logic).

With Zero Length Encoding it is also possible to set N_{LBK} (LOOK BACK), the number of data to be stored before the signal crosses the threshold and/or, N_{LFWD} (LOOK FORWARD), the number of data to be stored after the signal crosses the threshold (see § 4.3).
In this case the event of each channel has a particular format which allows the construction of the acquired time interval:

- **Total size of the event (total number of transferred 32bit data words)**
- **Control word**
- [stored valid data, if control word is "good"]
- **Control word**
- [stored valid data, if control word is "good"]
- ...

The total size is the number of 32 bit data that compose the event (including the size itself).
The control word has the following format:

Bit	Function
[31]	0: skip 1: good
[20:0]	stored/skipped words

[2] Available with Piggy Back Rev. 0.5 and Firmware Rev. 0.5

If the control word type is "good", then it will be followed by as many 32bit data words as those indicated in the "stored/skipped words" field; if the control word type is "skip" then it will be followed by a "good" control world, unless the end of event is reached.
IMPORTANT NOTE: the maximum allowed number of control words is 14; therefore the ZLE is active within the event until the 14th transition between a "good" and a "skip" zone (or between a "skip" and a "good" zone). All the subsequent samples are considered "good" and stored.

3.4.2. *Zero Suppression Examples*

If the input signal is the following (N_1 , N_2 .. N_n, N_{LBK} , N_{LFWD} are 64bit longwords = 4 samples each[3]):

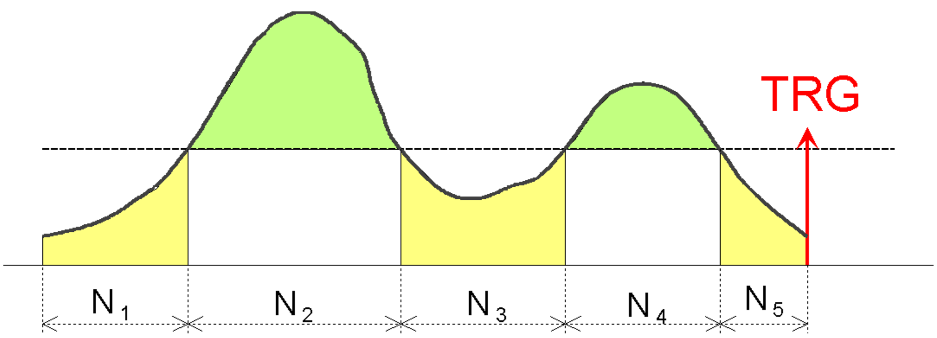

Fig. 3.12: Zero Suppression example

If the algorithm works in positive logic, and
$\mathbf{N_{LBK} < N_1}$;
$\mathbf{N_{LFWD} < N_5}$;
$\mathbf{N_{LBK} + N_{LFWD} < N_3}$;

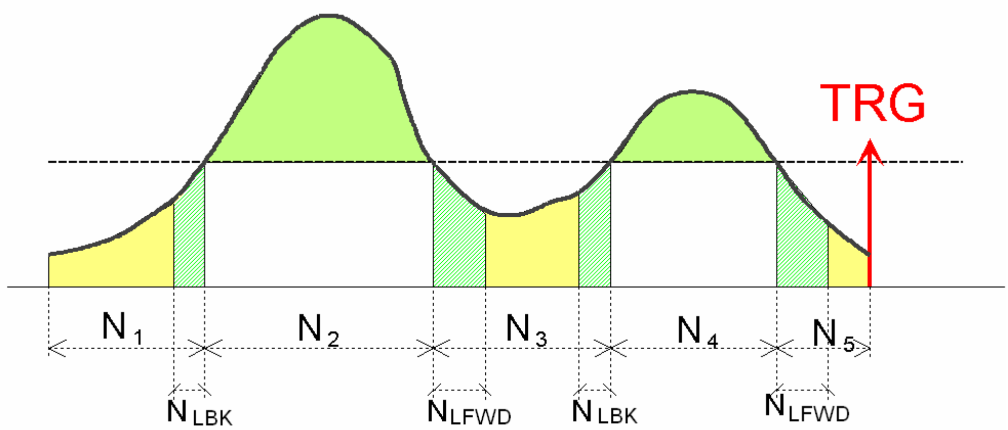

Fig. 3.13: Example with positive logic and non-overlapping N_{LBK} / N_{LFWD}

then the readout event is:

[3] 5 samples each with Pack2.5 mode

$N'_2 + N'_4$ + 5 (control words) + 1 (size)
Skip $N'_1 = 2(N_1 - N_{LBK})$
Good $N'_2 = 2(N_{LBK} + N_2 + N_{LFWD})$
... N'_2 words with samples over threshold
Skip $N'_3 = 2(N_3 - N_{LFWD} - N_{LBK})$
Good $N'_4 = 2(N_{LBK} + N_4 + N_{LFWD})$
... N'_4 words with samples over threshold
Skip $N'_5 = 2(N_5 - N_{LFWD})$

If the algorithm works in negative logic, and
$\mathbf{N_{LBK} + N_{LFWD} < N_2}$;
$\mathbf{N_{LBK} + N_{LFWD} < N_4}$;

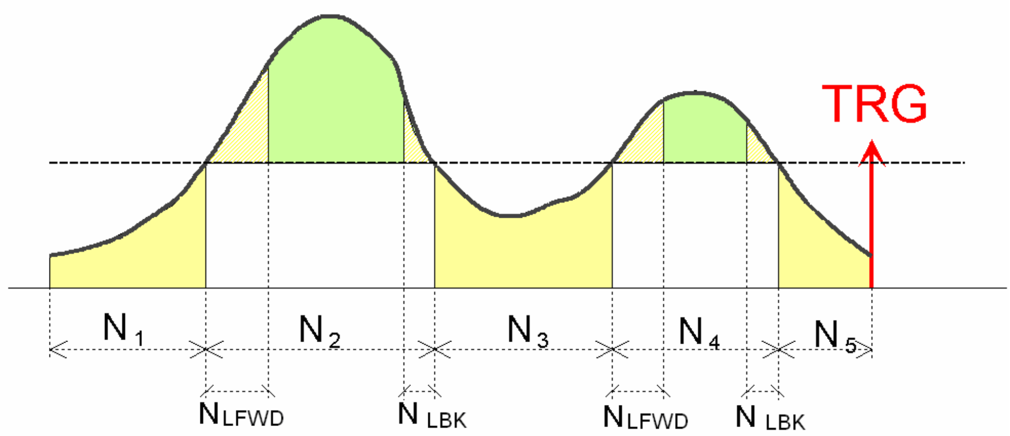

Fig. 3.14: Example with negative logic and non-overlapping N_{LBK} / N_{LFWD}

then the readout event is:
$N'_1 + N'_3 + N'_5$ + 5 (control words) + 1 (size)
Good $N'_1 = 2(N_1 + N_{LFWD})$
... N'_1 words with samples under threshold
Skip $N'_2=2(N_2 - N_{LFWD} - N_{LBK})$
Good $N'_3 = 2(N_{LBK} + N_3 + N_{LFWD})$
... N'_3 words with samples under threshold
Skip $N'_4=2(N_4 - N_{LFWD} - N_{LBK})$
Good $N'_5 = 2(N_{LBK} + N_5)$
... N'_5 words with samples under threshold

In some cases the number of data to be discarded can be smaller than $\mathbf{N_{LBK}}$ and $\mathbf{N_{LFWD}}$:

1) If the algorithm works in positive logic, and
$\mathbf{N_1 \leq N_{LBK} < N_3}$;
$\mathbf{N_{LFWD} = 0}$;

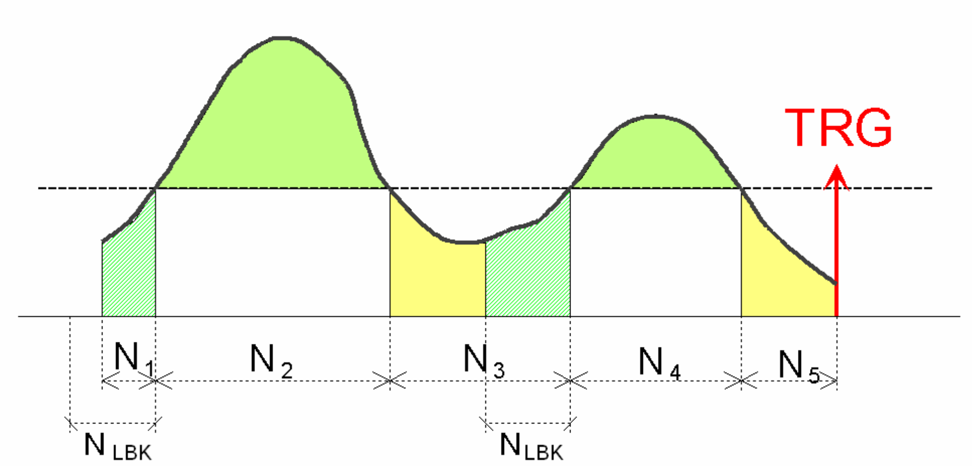

Fig. 3.15: Example with positive logic and non overlapping N_{LBK}

then the readout event is:
$N'_1 + N'_2 + N'_4$ + 5 (control words) + 1 (size)
Good $N'_1 + N'_2 = 2(N_1 + N_2)$
... $N'_1 + N'_2$ words with samples over threshold
Skip $N'_3 = 2(N_3 - N_{LBK})$
Good $N'_4 = 2(N_{LBK} + N_4)$
... N'_4 words with samples over threshold
Skip $N'_5 = 2N_5$

2) If the algorithm works in positive logic, and
$\mathbf{N_{LBK} = 0}$;
$\mathbf{N_5 \leq N_{LFWD} < N_3}$;

then the readout event is:
$N'_2 + N'_4 + N'_5$ + 5 (control words) + 1 (size)
Skip $N'_1 = 2N_1$
Good $N'_2 = 2(N_2 + N_{LFWD})$
... N'_2 words with samples over threshold
Skip $N'_3 = 2(N_3 - N_{LFWD})$
Good $N'_4 + N'_5$ ($N'_5 = 2N_5$...)
... $N'_4 + N'_5$ words with samples over threshold

3) If the algorithm works in positive logic, and
$\mathbf{N_{LBK} = 0}$;
$\mathbf{N_3 \leq N_{LFWD} < N_5}$;

then the readout event is:
N'_2 + 3 (control words) + 1 (size)
Skip $N'_1 = 2N_1$
Good $N'_2 = 2(N_2 + N_3 + N_4 + N_{LFWD})$
... N'_2 words with samples over threshold
Skip $N'_5 = 2(N_5 - N_{LFWD})$

4) If the algorithm works in positive logic, and
$\mathbf{N_3 \leq N_{LBK} < N_1}$;
$\mathbf{N_{LFWD} = 0}$;

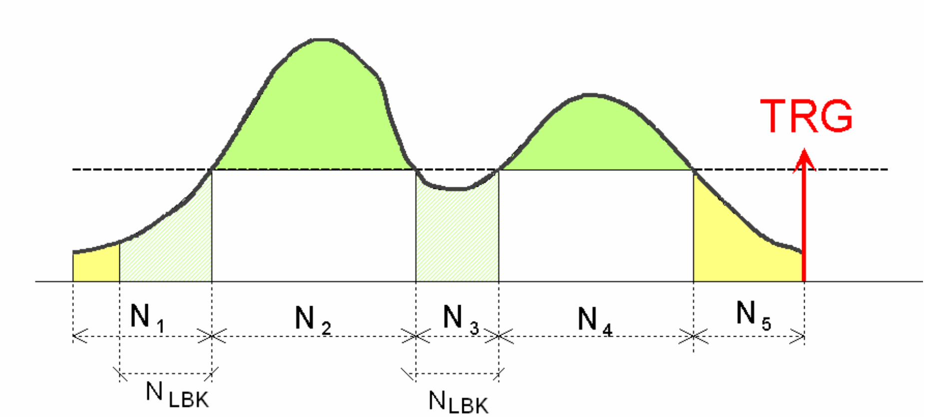

Fig. 3.16: Example with positive logic and overlapping N_{LBK}

then the readout event is:
$N'_2 + N'_4$ + 4 (control words) + 1 (size)
Skip $N'_1 = 2(N_1 - N_{LBK})$
Good $N'_2 = 2(N_{LBK} + N_2)$
... N'_2 words with samples over threshold
Good $N'_4 = 2(N_3 + N_4)$
... N'_4 words with samples over threshold
Skip $N'_5 = 2N_5$
N.B: In this case there are two subsequent “GOOD” intervals.

5) If the algorithm works in positive logic, and
$\mathbf{0 < N_{LBK} < N_1}$;
$\mathbf{N_{LFWD} < N_5}$;
$\mathbf{N_{LBK} + N_{LFWD} \geq N_3}$.
then the readout event is:
$N'_2 + N'_4$ + 4 (control words) + 1 (size)
Skip $N'_1 = 2(N_1 - N_{LBK})$
Good $N'_2 = 2(N_{LBK} + N_2 + N_{LFWD})$
... N'_2 words with samples over threshold
Good $N'_4 = 2(N_3 - N_{LFWD}) + 2N_4 + 2N_{LFWD}$
... N'_4 words with samples over threshold
Skip $N'_5 = 2(N_5 - N_{LFWD})$
N.B: In this case there are two subsequent “GOOD” intervals.

These examples are reported with positive logic; the compression algorithm is the same also working in negative logic.

3.5. Trigger management

All the channels in a board share the same trigger: this means that all the channels store an event at the same time and in the same way (same number of samples and same position with respect to the trigger); several trigger sources are available.

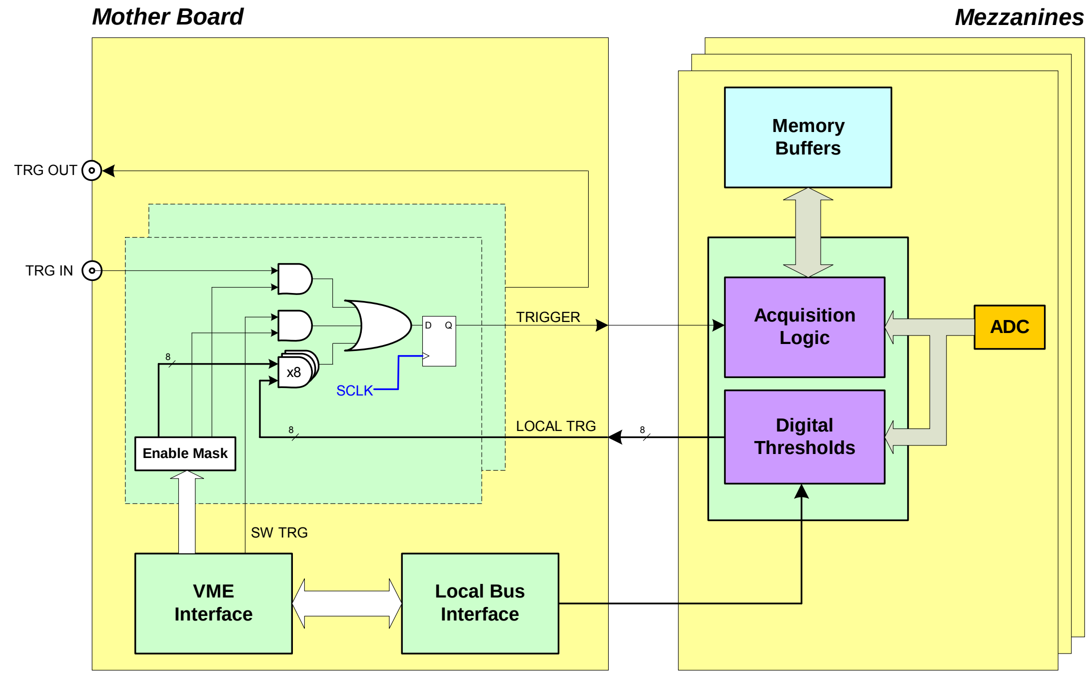

Fig. 3.17: Block diagram of Trigger management

3.5.1. External trigger

External trigger can be NIM/TTL signal on LEMO front panel connector, 50 Ohm impedance. The external trigger is synchronised with the internal clock (see § 3.2.3); if External trigger is not synchronised with the internal clock, a one clock period jitter occurs.

3.5.2. Software trigger

Software trigger are generated via VME bus (write access in the relevant register, see § 4.20).

3.5.3. Local channel auto-trigger

Each channel can generate a local trigger as the digitised signal exceeds the Vth threshold (ramping up or down, depending on VME settings), and remains under or over threshold for Nth “4/5 samples groups” (depending on selected storage mode, see § 3.3.4) at least (Nth is programmable via VME). The Vth digital threshold, the edge type, and the minimum number Nth of [4/5 samples] are programmable via VME register accesses, see § 4.3 and § 4.6; actually local trigger is delayed of Nth [4/5 samples] with respect to the input signal.

N.B.: the local trigger signal does not start directly the event acquisition on the relevant channel; such signal is propagated to the central logic which produces the global trigger, which is distributed to all channels (see § 3.5.4).

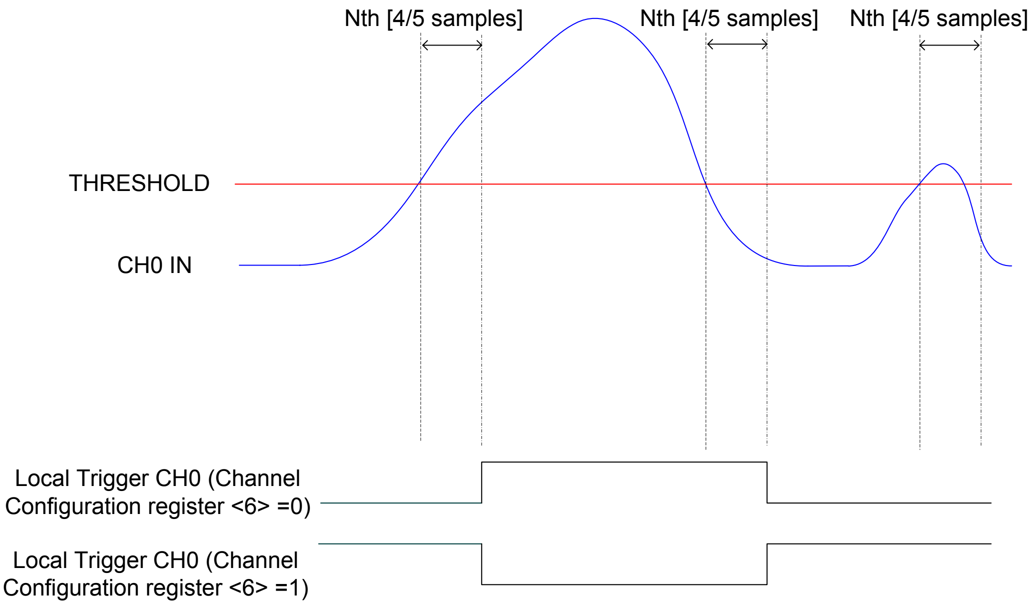

Fig. 3.18: Local trigger generation

3.5.3.1. Trigger coincidence level

It is possible to set the minimum number of channels that must be over threshold, beyond the triggering channel, in order to actually generate the local trigger signal. If, for example, Trigger Source Enable Mask (see § 4.21) bits[7:0]=FF (all channels enabled) and Local trigger coincidence level = 1 (bits [26:24]), whenever an enabled channel exceeds the threshold, the trigger will be generated only if at least another channel is over threshold at that moment. Local trigger coincidence level must be smaller than the number of channels enabled via bit[7:0] mask. The following figure shows examples with Local trigger coincidence level = 1 and = 0.

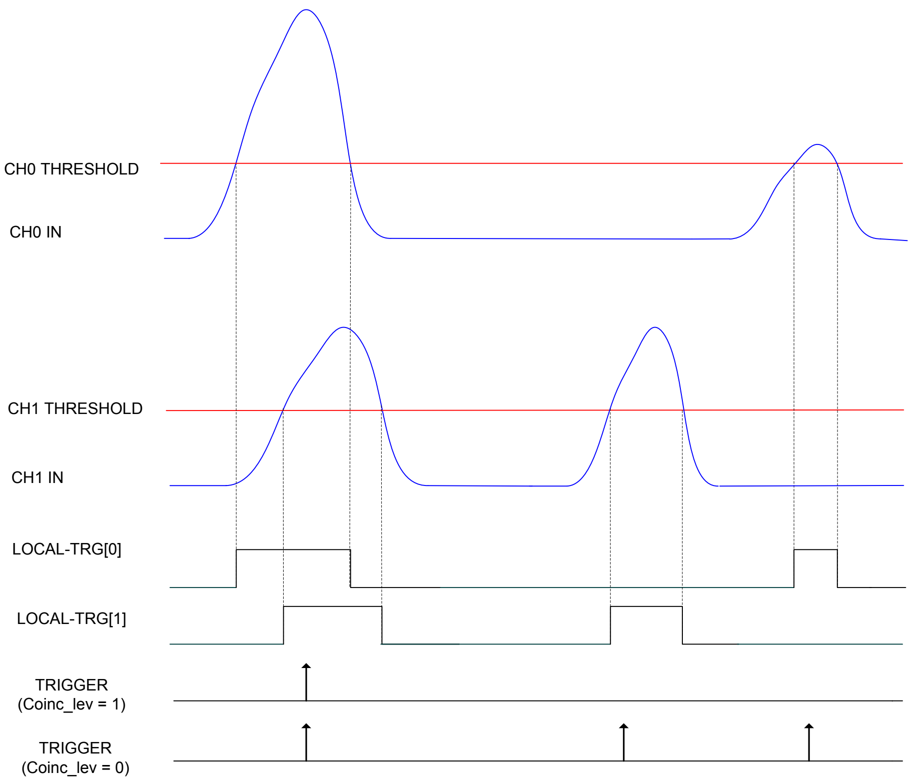

Fig. 3.19: Local trigger relationship with Coincidence level

3.5.4. Trigger distribution

The OR of all the enabled trigger sources, after being synchronised with the internal clock, becomes the global trigger of the board and is fed in parallel to all the channels, which store an event.

A Trigger Out is also generated on the relevant front panel TRG_OUT connector (NIM or TTL), and allows to extend the trigger signal to other boards.

For example, in order to start the acquisition on all the channels in the crate, as one of the channels ramps over threshold, the Local Trigger must be enabled as Trigger Out, the Trigger Out must then be fed to a Fan Out unit; the obtained signal has to be fed to the External Trigger Input of all the boards in the crate (including the board which generated the Trigger Out signal).

3.6. Front Panel I/Os

The V1720 is provided with 16 programmable general purpose LVDS I/O signals. Signals can be programmed via VME (see § 4.24 and § 4.25).

Default configuration is:

Table 3.2: Front Panel I/Os default setting

Nr.	Direction	Description
0	out	Ch 0 Trigger Request
1	out	Ch 1 Trigger Request
2	out	Ch 2 Trigger Request

Nr.	Direction	Description
3	out	Ch 3 Trigger Request
4	out	Ch 4 Trigger Request
5	out	Ch 5 Trigger Request
6	out	Ch 6 Trigger Request
7	out	Ch 7 Trigger Request
8	out	Memory Full
9	out	Event Data Ready
10	out	Channels Trigger
11	out	RUN Status
12	in	Trigger Time Tag Reset (active low)
13	in	Memory Clear (active low)
14	-	RESERVED
15	-	RESERVED

3.7. Analog Monitor

The board houses a 12bit (100MHz) DAC with 0÷1 V dynamics on a 50 Ohm load (see Fig. 1.1), whose input is controlled by the ROC FPGA and the signal output (driving 50 Ohm) is available on the MON/Σ output connector. MON output of more boards can be summed by an external Linear Fan In.
This output is delivered by a 12 bit DAC.

The DAC control logic implements four operating modes:
- Trigger Majority Mode (Monitor Mode = 0)
- Test Mode (Monitor Mode = 1)
- Buffer Occupancy Mode (Monitor Mode = 3)
- Voltage Level Mode (Monitor Mode = 4)

Operating mode is selected via Monitor Mode register (see § 4.31); Monitor Mode = 2 is reserved for future implementation.

3.7.1. Trigger Majority Mode (Monitor Mode = 0)

It is possible to generate a Majority signal with the DAC: a voltage signal whose amplitude is proportional to the number of channels under/over (see § 4.12) threshold (1 step = 125mV); this allows, via an external discriminator, to produce a global trigger signal, as the number of triggering channels has exceeded a particular threshold.

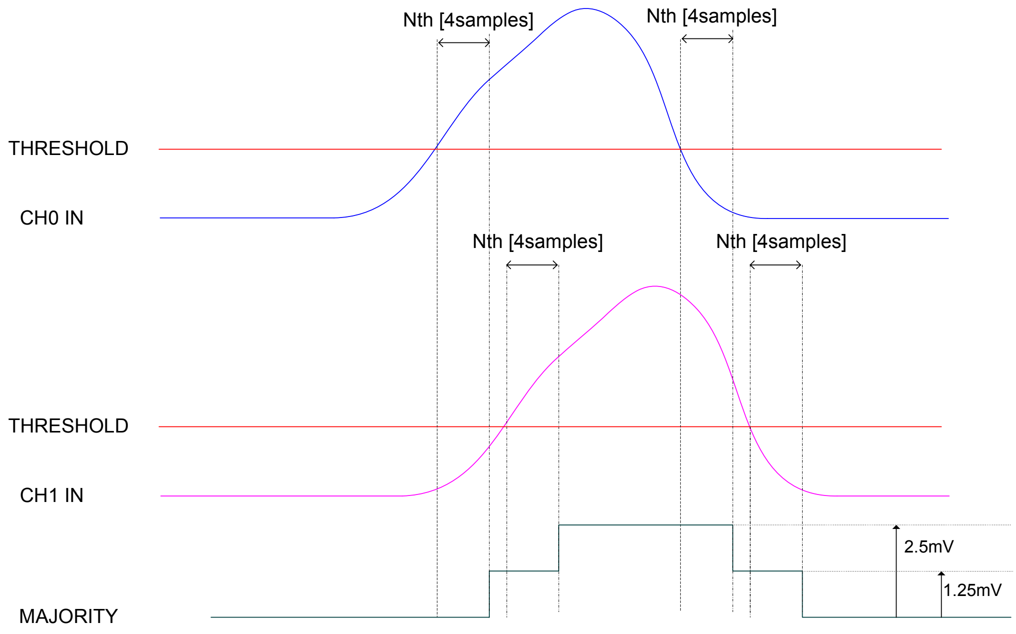

Fig. 3.20: Majority logic (2 channels over threshold; bit[6] of Ch. Config. Register =0)

In this mode the MON output provides a signal whose amplitude is proportional to the number of channels over the trigger threshold. The amplitude step (= +1 channel over threshold) is 125mV.

3.7.2. Test Mode (Monitor Mode = 1)

In this mode the MON output provides a sawtooth signal with 1 V amplitude and 30.518 Hz frequency.

3.7.3. Buffer Occupancy Mode (Monitor Mode = 3)

In this mode, MON out provides a voltage value proportional to the number of buffers filled with events; step: 1 buffer = 0.976 mV. .
This mode allows to test the readout efficiency: in fact if the average event readout throughput is as fast as trigger rate, then MON out value remains constant; otherwise if MON out value grows in time, this means that readout rate is slower than trigger rate.

3.7.4. Voltage Level Mode (Monitor Mode = 4)

In this mode, MON out provides a voltage value programmable via the 'N' parameter written in the SET MONITOR DAC register, with: Vmon = 1/4096*N (Volt).

3.8. Test pattern generator

The FPGA AMC can emulate the ADC and write into memory a ramp (0, 1, 2, 3,…FF, FF, FE.., 0) for test purposes. It can be enabled via Channel Configuration register, see § 4.12.

3.9. Reset, Clear and Default Configuration

3.9.1. Global Reset

Global Reset is performed at Power ON of the module or via a VME RESET (SYS_RES), see § 4.42. It allows to clear the data off the Output Buffer, the event counter and performs a FPGAs global reset, which restores the FPGAs to the default configuration. It initialises all counters to their initial state and clears all detected error conditions.

3.9.2. Memory Reset

The Memory Reset clears the data off the Output Buffer.
The Memory Reset can be forwarded via either a write access to Software Clear Register (see § 4.43) or with a pulse sent to the front panel Memory Clear input (see § 3.6).

3.9.3. Timer Reset

The Timer Reset allows to initialize the timer which allows to tag an event. The Timer Reset can be forwarded with a pulse sent to Trigger Time Tag Reset input (see § 3.6).

3.10. VMEBus interface

The module is provided with a fully compliant VME64/VME64X interface (see § 1.1), whose main features are:
- EUROCARD 9U Format
- J1/P1 and J2/P2 with either 160 pins (5 rows) or 96 (3 rows) connectors
- A24, A32 and CR-CSR address modes
- D32, BLT/MBLT, 2eVME, 2eSST data modes
- MCST write capability
- CBLT data transfers
- RORA interrupter
- Configuration ROM

3.10.1. Addressing capabilities

3.10.1.1. Base address

The module works in A24/A32 mode. The Base Address of the module can be fixed through four rotary switches (see § 2.6) and is written into a word of 24 or 32 bit.
The Base Address can be selected in the range:

0x000000 ←→ 0xFF0000 A24 mode

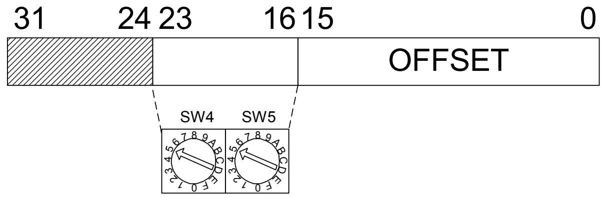

Fig. 3.21: A24 addressing

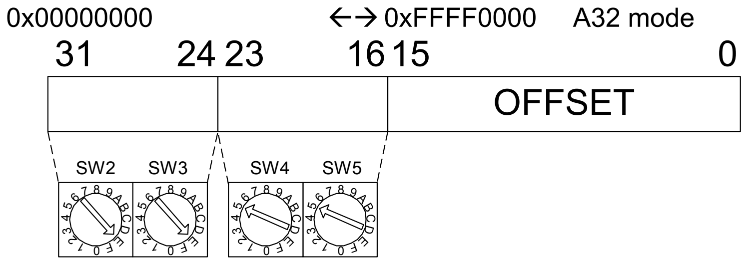

Fig. 3.22: A32 addressing

The Base Address of the module is selected through four rotary switches (see § 2.6), then it is validated only with either a Power ON cycle or a System Reset (see § 3.8).

3.10.1.2. CR/CSR address

GEO address is picked up from relevant backplane lines and written onto bit 23..19 of CR/CSR space, indicating the slot number in the crate; the recognised Address Modifier for this cycle is 2F. *This feature is implemented only on versions with 160pin connectors*.

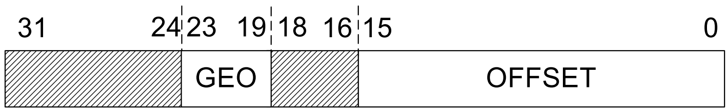

Fig. 3.23: CR/CSR addressing

3.10.1.3. Address relocation

Relocation Address register (see § 4.37) allows to set via software the board Base Address (valid values ≠ 0). Such register allows to overwrite the rotary switches settings; its setting is enabled via VME Control Register (see § 4.29). The used addresses are:

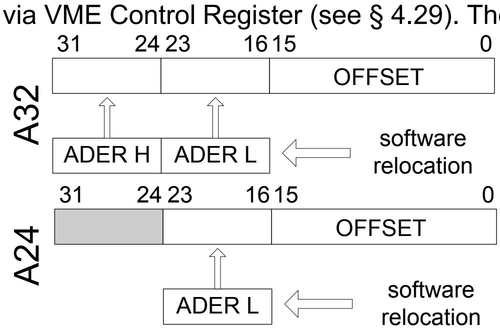

Fig. 3.24: Software relocation of base address

3.11. Data transfer capabilities

The board supports D32 single data readout, Block Transfer BLT32 and MBLT64, 2eVME and 2eSST cycles. Sustained readout rate is up to 60 MB/s with MBLT64, up to 100 MB/s with 2eVME and up to 160 MB/s with 2eSST.

3.12. Events readout

3.12.1. Sequential readout

The events, once written in the SRAMs (Memory Event Buffers), become available for readout via VME. During the memory readout, the board can continue to store more events (independently from the readout) on the free buffers. The acquisition process is therefore “deadtimeless”, until the memory becomes full.

Although the memories are SRAMs, VMEBus does not handle directly the addresses, but takes them from a FIFO. Therefore, data are read from the memories sequentially, according to the selected Readout Logic, from a memory space mapped on 4Kbytes (0x0000÷0x0FFC).

The events are readout sequentially and completely, starting from the Header of the first available event, followed by the Trigger Time Tag, the Event Counter and all the samples of the channels (from 0 to 7). Once an event is completed, the relevant memory buffer becomes free and ready to be written again (old data are lost). After the last word in an event, the first word (Header) of the subsequent event is readout. It is not possible to readout an event partially (see also § 3.3.4).

3.12.1.1. SINGLE D32

This mode allows to readout a word per time, from the header (actually 4 words) of the first available event, followed by all the words until the end of the event, then the second event is transferred. The exact sequence of the transferred words is shown in § 3.3.4.
We suggest, after the 1st word is transferred, to check the Event Size information and then do as many D32 cycles as necessary (actually Event Size -1) in order to read completely the event.

3.12.1.2. BLOCK TRANSFER D32/D64, 2eVME

BLT32 allows, via a single channel access, to read N events in sequence, N is set via the BLT Event Number register (see § 4.40).
The event size depends on the Buffer Size Register setting (§ 4.15); namely:

```
[Event Size] = [8*(Block Size)] + [16 bytes]
```

Then it is necessary to perform as many cycles as required in order to readout the programmed number of events.
We suggest to enable BERR signal during BLT32 cycles, in order to end the cycle avoiding filler readout. The last BLT32 cycle will not be completed, it will be ended by BERR after the #N event in memory is transferred (see example in the figure below).

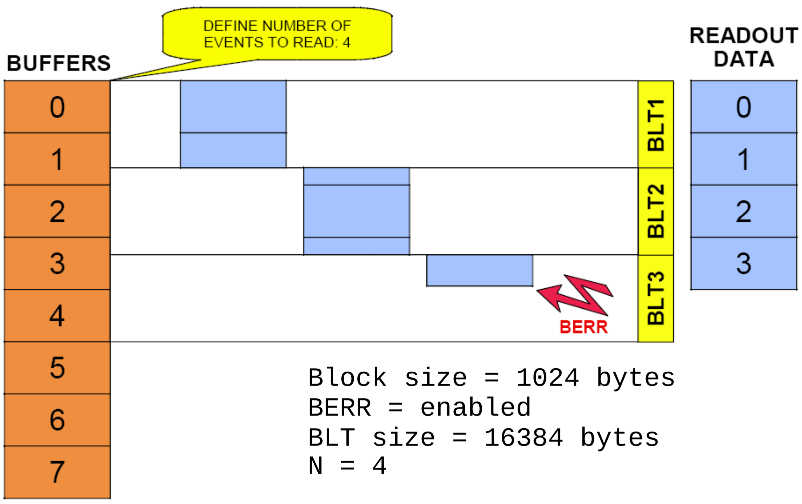

Fig. 3.25: Example of BLT readout

Since some 64 bit CPU's cut off the last 32 bit word of a transferred block, if the number of words composing such block is odd, it is necessary to add a dummy word (which has then to be removed via software) in order to avoid data loss. This can be achieved by setting the ALIGN64 bit in the VME Control register (see § 4.29).

MBLT64 cycle is similar to the BLT32 cycle, except that the address and data lines are multiplexed to form 64 bit address and data buses.

The 2eVME allows to achieve higher transfer rates thanks to the requirement of only two edges of the two control signals (DS and DTACK) to complete a data cycle.

3.12.1.3. CHAINED BLOCK TRANSFER D32/D64

The V1720 allows to readout events from more daisy chained boards (Chained Block Transfer mode).

The technique which handles the CBLT is based on the passing of a token between the boards; it is necessary toverify that the used VME crate supports such cycles.
Several contiguous boards, in order to be daisy chained, must be configured as “first”, “intermediate” or “last” via MCST Base Address and Control Register (see § 4.36). A common Base Address is then defined via the same register; when a BLT cycle is executed at the address CBLT_Base + 0x0000 ÷ 0x0FFC, the “first” board starts to transfer its data, driving DTACK properly; once the transfer is completed, the token is passed to the second board via the IACKIN-IACKOUT lines of the crate, and so on until the “last” board, which completes the data transfer and asserts BERR (which has to be enabled): the Master then ends the cycle and the slave boards are rearmed for a new acquisition.
If the size of the BLT cycle is smaller than the events size, the board which has the token waits for another BLT cycle to begin (from the point where the previous cycle has ended).

3.12.2. Random readout (to be implemented)

Events can be readout partially (not necessarily starting from the first available) and are not erased from the memories, unless a command is performed. In order to perform the random readout it is necessary to execute an ***Event Block Request*** via VME.
Indicating the event to be read (page number = 12 bit datum), the offset of the first word to be read inside the event (12 bit datum) and the number of words to be read (size = 10 bit datum). At this point the data space can be read, starting from the header (which reports the required size, not the actual one, of the event), the Trigger Time Tag, the Event Counter and the part of the event required on the channel addressed in the Event Block Request.
After data readout, in order to perform a new random readout, it is necessary a new Event Block Request, otherwise Bus Error is signalled. In order to empty the buffers, it is necessary a write access to the Buffer Free register (see § 0): the datum written is the number of buffers in sequence to be emptied.

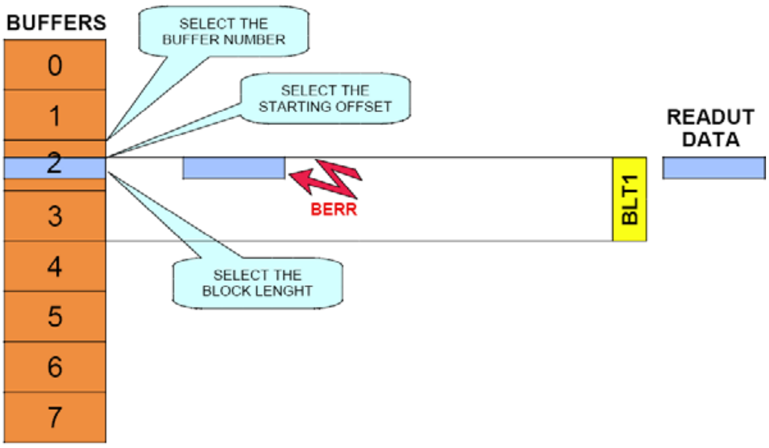

Fig. 3.26: Example of random readout

3.13. Optical Link

The board houses a daisy chainable Optical Link able to transfer data at 80 MB/s, therefore it is possible to connect up to eight V1720 to a single Optical Link Controller: a standard PC equipped with the PCI card CAEN Mod. A2818.

The A2818 is a 32-bit 33 MHz PCI card; the communication path uses optical fiber cables as physical transmission line (Mod. AY2705, AY2720, AI2705, AI2720).
AY2705 and AY2720 have a duplex connector on the A2818 side and two simplex connectors on the board side; the simplex connector with the black wrap is for the RX line (lower) and the one with the red wrap is for the TX (higher); see also § 2.4.5).
The Optical Link allows to perform VME read (Single data transfer and Block transfers) and write (Single data transfer) operations.
See also the web page: http://www.caen.it/nuclear/product.php?mod=A2818
The parameters for read/write accesses via optical link are the same used by VME cycles (Address Modifier, Base Address, data Width, etc); wrong parameter settings cause Bus Error.
VME Control Register bit 3 (see § 4.33) allows to enable the module to broadcast an interrupt request on the Optical Link; an 8 bit mask (see § 3.14.12 and § 3.14.13) allows to enable the corresponding A2818's to propagate the interrupt on the PCI bus as a request from the Optical Link is sensed.
VME and Optical Link accesses take place on independent paths and are handled by board internal controller, with VME having higher priority; anyway it is better to avoid accessing the board via VME and Optical Link simultaneously.

The following diagram shows how to connect V1720 modules to the Optical Link:

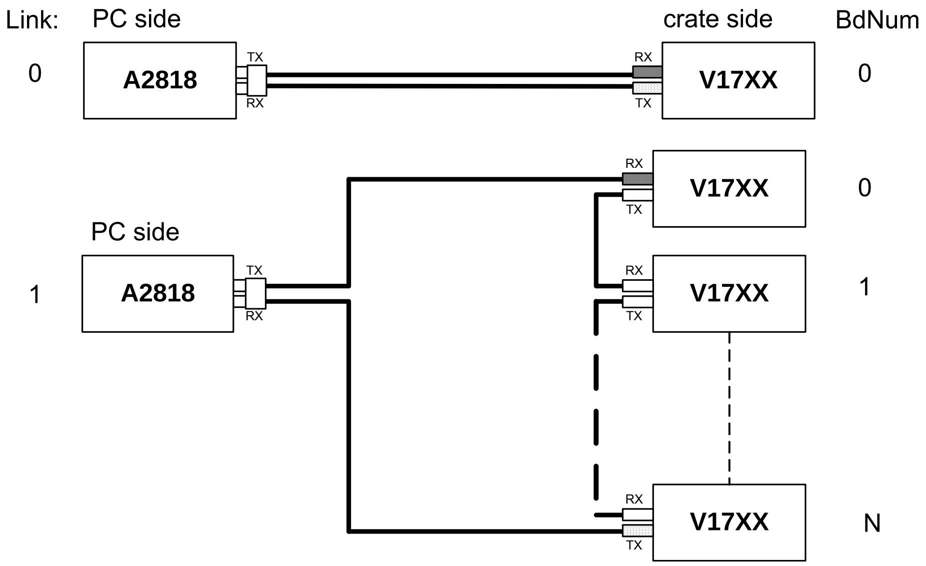

Fig. 3.27: Optical Link daisy chain

3.14. CAENVMELib library

The Optical Link can be operated through the CAENVMELib library: a set of ANSI C functions which permits an user program the use and the configuration of the modules.
The present description refers to CAENVMELib, available in the following formats:

– Win32 DLL (CAEN provides the CAENVMELib.lib stub for Microsoft Visual C++ 6.0)

– Linux dynamic library

CAENVMELib is logically located between an application like the samples provided and the device driver.

3.14.1. CAENVME_Init

Parameters:
[in] BdType : The model of the board (V2718).
[in] Link : The index of the A2818 (see figure above).
[in] BdNum : The board number in the link (see figure above).
[out] Handle : The handle that identifies the device.

Returns:
An error code about the execution of the function.

Description:
The function generates an opaque handle to identify the module attached to the PC. It must be specified only the module index (BdNum) because the link is PCI.

CAENVME_API
CAENVME_Init(CVBoardTypes BdType, short Link, short BdNum, long *Handle);

3.14.2. CAENVME_End

Parameters:
[in] Handle: The handle that identifies the module.

Returns:
An error code about the execution of the function.

Description:
Notifies the library about the end of work and free the allocated resources.

CAENVME_API
CAENVME_End(long Handle);

3.14.3. CAENVME_ReadCycle

Parameters:
[in] Handle : The handle that identifies the device.
[in] Address : The VME bus address[4].
[out] Data : The data read from the VME bus.
[in] AM : The address modifier .
[in] DW : The data width.

Returns:
An error code about the execution of the function.
Description:
The function performs a single VME read cycle.

CAENVME_API
CAENVME_ReadCycle(long Handle, unsigned long Address, void *Data,

[4] The Board base address set via rotary switches (see § 2.6)

CVAddressModifier AM, CVDataWidth DW);

3.14.4. CAENVME_WriteCycle

Parameters:

[in] Handle : The handle that identifies the device.
[in] Address : The VME bus address.
[in] Data : The data written to the VME bus.
[in] AM : The address modifier.
[in] DW : The data width.

Returns:

An error code about the execution of the function.

Description:

The function performs a single VME write cycle.

CAENVME_API
CAENVME_WriteCycle(long Handle, unsigned long Address, void *Data,
CVAddressModifier AM, CVDataWidth DW);

3.14.5. CAENVME_MultiRead

Parameters:

[in] Handle : The handle that identifies the device.
[in] Address : An array of VME bus addresses.
[out] Data : An array of data read from the VME bus.
[in] AM : An array of address modifiers.
[in] DW : An array of data widths.

Returns:

An array of error codes about the execution of the function.

Description:

The function performs a sequence of VME read cycles.

CAENVME_API
CAENVME_MultiRead(long Handle, unsigned long Address, void *Data,
CVAddressModifier AM, CVDataWidth DW);

3.14.6. CAENVME_MultiWrite

Parameters:

[in] Handle : The handle that identifies the device.
[in] Address : An array of VME bus addresses.
[in] Data : An array of data written to the VME bus.
[in] AM : An array of address modifiers.
[in] DW : An array of data widths.

Returns:

An array of error codes about the execution of the function.

Description:

The function performs a sequence of VME write cycles.

CAENVME_API
CAENVME_ReadCycle(long Handle, unsigned long Address, void *Data, CVAddressModifier AM, CVDataWidth DW);

3.14.7. CAENVME_BLTReadCycle

Parameters:
[in] Handle : The handle that identifies the device.
[in] Address : The VME bus address.
[out] Buffer : The data read from the VME bus.
[in] Size : The size of the transfer in bytes.
[in] AM : The address modifier.
[in] DW : The data width.
[out] count : The number of bytes transferred.

Returns:
An error code about the execution of the function.

Description:
The function performs a VME block transfer read cycle. It can be used to perform MBLT transfers using 64 bit data width.

CAENVME_API
CAENVME_BLTReadCycle(long Handle, unsigned long Address, unsigned char *Buffer, int Size, CVAddressModifier AM, CVDataWidth DW, int *count);

3.14.8. CAENVME_FIFOBLTReadCycle

Parameters:
[in] Handle : The handle that identifies the device.
[in] Address : The VME bus address.
[out] Buffer : The data read from the VME bus.
[in] Size : The size of the transfer in bytes.
[in] AM : The address modifier
[in] DW : The data width.
[out] count : The number of bytes transferred.

Returns:
An error code about the execution of the function.
Description:
The function performs a VME block transfer read cycle. It can be used to perform MBLT transfers using 64 bit data width. The Address is not incremented on the VMEBus during the cycle.

CAENVME_API
CAENVME_FIFOBLTReadCycle(int32_t Handle, uint32_t Address, void *Buffer, int Size, CVAddressModifier AM, CVDataWidth DW, int *count);

3.14.9. CAENVME_MBLTReadCycle

Parameters:
[in] Handle : The handle that identifies the device.
[in] Address : The VME bus address.

[out] Buffer : The data read from the VME bus.
[in] Size : The size of the transfer in bytes.
[in] AM : The address modifier.
[out] count : The number of bytes transferred.

Returns:
An error code about the execution of the function.

Description:
The function performs a VME multiplexed block transfer read cycle.

CAENVME_API
CAENVME_MBLTReadCycle(long Handle, unsigned long Address, unsigned char *Buffer, int Size, CVAddressModifier AM, int *count);

3.14.10. CAENVME_FIFOMBLTReadCycle

Parameters:
[in] Handle : The handle that identifies the device.
[in] Address : The VME bus address.
[out] Buffer : The data read from the VME bus.
[in] Size : The size of the transfer in bytes.
[in] AM : The address modifier.
[out] count : The number of bytes transferred.

Returns:
An error code about the execution of the function.

Description:
The function performs a VME multiplexed block transfer read cycle.
The Address is not incremented on the VMEBus during the cycle.

CAENVME_API
CAENVME_FIFOMBLTReadCycle(int32_t Handle, uint32_t Address, void *Buffer,
int Size, CVAddressModifier AM, int *count);

3.14.11. CAENVME_IRQCheck

Parameters:
[in] Handle : The handle that identifies the device.
[out] Mask : A bit-mask[5] indicating the active IRQ lines

Returns:
An error code about the execution of the function.

Description:
The function returns a bit mask indicating the active IRQ lines.

CAENVME_API
CAENVME_IRQCheck(long Handle, byte *Mask);

---

[5] Actually only bit 0 in the mask is meaningful

3.14.12. CAENVME_IRQEnable

Parameters:
[in] Handle : The handle that identifies the device.
[in] Mask : A bit-mask indicating the IRQ lines.

Returns:
An error code about the execution of the function.

Description:
The function enables the IRQ lines specified by Mask.

```
CAENVME_API
CAENVME_IRQEnable(long dev, unsigned long Mask);
```

3.14.13. CAENVME_IRQDisable

Parameters:
[in] Handle : The handle that identifies the device.
[in] Mask : A bit-mask indicating the IRQ lines.

Returns:
An error code about the execution of the function.

Description:
The function disables the IRQ lines specified by Mask.

```
CAENVME_API
CAENVME_IRQDisable(long dev, unsigned long Mask);
```

3.14.14. CAENVME_IRQWait

Parameters:
[in] Handle : The handle that identifies the device.
[in] Mask : A bit-mask indicating the IRQ lines.
[in] Timeout : Timeout in milliseconds.

Returns:
An error code about the execution of the function.

Description:
The function waits the IRQ lines specified by Mask until one of them raise or timeout expires.

```
CAENVME_API
CAENVME_IRQWait(long dev, unsigned long Mask, unsigned long Timeout)
```

4. VME Interface

The following sections will describe in detail the VME-accessible registers content.

4.1. Registers address map

Table 4.1: Address Map for the Model V1720

REGISTER NAME	ADDRESS	ASIZE	DSIZE	MODE	H_RES	S_RES	CLR
EVENT READOUT BUFFER	0x0000-0x0FFC	A24/A32/A64	D32	R	X	X	X
Channel n ZS_THRES	0x1n24	A24/A32	D32	R/W	X	X	
Channel n ZS_NSAMP	0x1n28	A24/A32	D32	R/W	X	X	
Channel n THRESHOLD	0x1n80	A24/A32	D32	R/W	X	X	
Channel n TIME OVER/UNDER THRESHOLD	0x1n84	A24/A32	D32	R/W	X	X	
Channel n STATUS	0x1n88	A24/A32	D32	R	X	X	
Channel n AMC FPGA FIRMWARE REVISION	0x1n8C	A24/A32	D32	R			
Channel n BUFFER OCCUPANCY	0x1n94	A24/A32	D32	R	X	X	X
Channel n DAC	0x1n98	A24/A32	D32	R/W	X	X	
Channel n ADC CONFIGURATION	0x1n9C	A24/A32	D32	R/W	X	X	
CHANNEL CONFIGURATION	0x8000	A24/A32	D32	R/W	X	X	
CHANNEL CONFIGURATION BIT SET	0x8004	A24/A32	D32	W	X	X	
CHANNEL CONFIGURATION BIT CLEAR	0x8008	A24/A32	D32	W	X	X	
BUFFER ORGANIZATION	0x800C	A24/A32	D32	R/W	X	X	
BUFFER FREE	0x8010	A24/A32	D32	R/W			
CUSTOM SIZE	0x8020	A24/A32	D32	R/W	X	X	
ACQUISITION CONTROL	0x8100	A24/A32	D32	R/W	X	X	
ACQUISITION STATUS	0x8104	A24/A32	D32	R			
SW TRIGGER	0x8108	A24/A32	D32	W			
TRIGGER SOURCE ENABLE MASK	0x810C	A24/A32	D32	R/W	X	X	
FRONT PANEL TRIGGER OUT ENABLE MASK	0x8110	A24/A32	D32	R/W	X	X	
POST TRIGGER SETTING	0x8114	A24/A32	D32	R/W	X	X	
FRONT PANEL I/O DATA	0x8118	A24/A32	D32	R/W	X	X	
FRONT PANEL I/O CONTROL	0x811C	A24/A32	D32	R/W	X	X	
CHANNEL ENABLE MASK	0x8120	A24/A32	D32	R/W	X	X	
ROC FPGA FIRMWARE REVISION	0x8124	A24/A32	D32	R			

REGISTER NAME	ADDRESS	ASIZE	DSIZE	MODE	H_RES	S_RES	CLR
EVENT STORED	0x812C	A24/A32	D32	R	X	X	X
SET MONITOR DAC	0x8138	A24/A32	D32	R/W	X	X	
BOARD INFO	0x8140	A24/A32	D32	R			
MONITOR MODE	0x8144	A24/A32	D32	R/W	X	X	
EVENT SIZE	0x814C	A24/A32	D32	R	X	X	X
VME CONTROL	0xEF00	A24/A32	D32	R/W	X		
VME STATUS	0xEF04	A24/A32	D32	R			
BOARD ID	0xEF08	A24/A32	D32	R/W	X	X	
MULTICAST BASE ADDRESS & CONTROL	0xEF0C	A24/A32	D32	R/W	X		
RELOCATION ADDRESS	0xEF10	A24/A32	D32	R/W	X		
INTERRUPT STATUS ID	0xEF14	A24/A32	D32	R/W	X		
INTERRUPT EVENT NUMBER	0xEF18	A24/A32	D32	R/W	X	X	
BLT EVENT NUMBER	0xEF1C	A24/A32	D32	R/W	X	X	
SCRATCH	0xEF20	A24/A32	D32	R/W	X	X	
SW RESET	0xEF24	A24/A32	D32	W			
SW CLEAR	0xEF28	A24/A32	D32	W			
FLASH ENABLE	0xEF2C	A24/A32	D32	R/W	X		
FLASH DATA	0xEF30	A24/A32	D32	R/W	X		
CONFIGURATION RELOAD	0xEF34	A24/A32	D32	W			
CONFIGURATION ROM	0xF000-0xF3FC	A24/A32	D32	R			

4.2. Configuration ROM (0xF000-0xF084; r)

The following registers contain some module's information, they are D32 accessible (read only):

- **OUI**: manufacturer identifier (IEEE OUI)
- **Version**: purchased version
- **Board ID**: Board identifier
- **Revision**: hardware revision identifier
- **Serial MSB**: serial number (MSB)
- **Serial LSB**: serial number (LSB)

Table 4.2: ROM Address Map for the Model V1720

Description	Address	Content
checksum	0xF000	*0xA4*
checksum_length2	0xF004	0x00

Description	Address	Content
checksum_length1	0xF008	0x00
checksum_length0	0xF00C	0x20
constant2	0xF010	0x83
constant1	0xF014	0x84
constant0	0xF018	0x01
c_code	0xF01C	0x43
r_code	0xF020	0x52
oui2	0xF024	0x00
oui1	0xF028	0x40
oui0	0xF02C	0xE6
vers	0xF030	V1720, VX1720: 0x11 V1720B, VX1720B: 0x40 V1720C, VX1720C: 0x12 V1720D, VX1720D: 0x41
board2	0xF034	V1720: 0x00 VX1720: 0x01
board1	0xF038	0x06
board0	0xF03C	0xB8
revis3	0xF040	*0x00*
revis2	0xF044	*0x00*
revis1	0xF048	*0x00*
revis0	0xF04C	*0x01*
sernum1	0xF080	*0x00*
sernum0	0xF084	*0x16*

These data are written into one Flash page; at Power ON the Flash content is loaded into the Configuration RAM, where it is available for readout.

4.3. Channel n ZS_THRES (0x1n24; r/w)

Bit	Function
[31]	0 = Positive Logic 1 = Negative Logic
[30:11]	*reserved*
[11:0]	The 12 LSB represent the value to be compared with each sample of the event, and see if it is "good" or "skip" type. (see § 3.4 and § 4.12)

4.4. Channel n ZS_NSAMP (0x1n28; r/w)

Bit	Function
[31:0]	With "Full Suppression based on the amplitude" (ZS AMP), bits [20:0] allow to set the number Ns of subsequent data which must be found over/under threshold (depending on the used logic) necessary to validate the event; if this field is set to 0, it is considered "1". With "Zero length encoding" (ZLE) bit [31:16] allows to set/read N_{LBK}: the number of data to be stored before the signal crosses the threshold. bit [15:0] allows to set/read N_{LFWD}: the number of data to be stored after the signal crosses the threshold (see § 3.4 and § 4.12)

4.5. Channel n Threshold (0x1n80; r/w)

Bit	Function
[11:0]	Threshold Value for Trigger Generation

Each channel can generate a local trigger as the digitised signal exceeds the Vth threshold, and remains under or over threshold for Nth [4 samples; 5 samples in Pack2.5

mode] at least; local trigger is delayed of Nth [4/5 samples] with respect to input signal. This register allows to set Vth (LSB=input range/12bit); see also § 3.5.3.

4.6. Channel n Over/Under Threshold (0x1n84; r/w)

Bit	Function
[11:0]	Number of Data under/over Threshold

Each channel can generate a local trigger as the digitised signal exceeds the Vth threshold, and remains under or over threshold for Nth [4/5 samples] at least; local trigger is delayed of Nth [4 samples; 5 samples in Pack2.5 mode] with respect to input signal. This register allows to set Nth; see also § 3.5.3.

4.7. Channel n Status (0x1n88; r)

Bit	Function
[5]	Buffer free error: 1 = trying to free a number of buffers too large
[4:3]	*reserved*
[2]	Channel n DAC (see § 4.10) Busy 1 = Busy 0 = DC offset updated
[1]	Memory empty
[0]	Memory full

4.8. Channel n AMC FPGA Firmware (0x1n8C; r)

Bit	Function
[31:16]	Revision date in Y/M/DD format
[15:8]	Firmware Revision (X)
[7:0]	Firmware Revision (Y)

Bits [31:16] contain the Revision date in Y/M/DD format.
Bits [15:0] contain the firmware revision number coded on 16 bit (X.Y format).
Example: revision 1.3 of 12th June 2007 is: 0x760C0103

4.9. Channel n Buffer Occupancy (0x1n94; r)

Bit	Function
[10:0]	Occupied buffers (0..1024)

4.10. Channel n DAC (0x1n98; r/w)

Bit	Function
[15:0]	DAC Data

Bits [15:0] allow to define a DC offset to be added the input signal in the ±1V range, see also § 3.1.1. When Channel n Status bit 2 is set to 0, DC offset is updated (see § 4.7).

4.11. Channel n ADC Configuration (0x1n9C; r/w)

Bit	Function
[15:0]	T.B.D.

This register allows to pilot the relevant ADC signals. See the LTC2242-12 - 12-Bit, 250Msps ADC data sheet for details.

4.12. Channel Configuration (0x8000; r/w)

Bit	Function
[19:16]	Allows to select Zero Suppression algorithm: 0000 = no zero suppression (default); 0010 = zero length encoding (ZLE); 0011 = full suppression based on the amplitude (ZS AMP)
[18:12]	*reserved*
[11]	0 = Pack2.5 disabled 1 = Pack2.5 enabled
[10:7]	*reserved*
[6]	0 = Trigger Output on Input Over Threshold 1 = Trigger Output on Input Under Threshold allows to generate local trigger either on channel over or under threshold (see § 4.3 and § 4.6)
[4]	0 = Memory Random Access 1 = Memory Sequential Access
[3]	0 = Test Pattern Generation Disabled 1 = Test Pattern Generation Enabled
[1]	0 = Trigger Overlapping Not Enabled 1 = Trigger Overlapping Enabled Allows to handle trigger overlap (see § 3.3.3)
[0]	*reserved*

This register allows to perform settings which apply to all channels.
It is possible to perform selective set/clear of the Channel Configuration register bits writing to 1 the corresponding set and clear bit at address 0x8004 (set) or 0x8008 (clear) see the following § 4.13 and 4.14. Default value is 0x10.

4.13. Channel Configuration Bit Set (0x8004; w)

Bit	Function
[7:0]	Bits set to 1 means that the corresponding bits in the Channel Configuration register are set to 1.

4.14. Channel Configuration Bit Clear (0x8008; w)

Bit	Function
[7:0]	Bits set to 1 means that the corresponding bits in the Channel Configuration register are set to 0.

4.15. Buffer Organization (0x800C; r/w)

Bit	Function
[3:0]	BUFFER CODE

The BUFFER CODE allows to divide the available Output Buffer Memory into a certain number of blocks, according to the following table:

Table 4.3: Output Buffer Memory block division

REGISTER (see § 4.15)	BUFFER NUMBER	SIZE of one BUFFER (samples)	
		SRAM 1.25MS/ch (V1720A/C)	SRAM 10MS/ch (V1720B/D)

		Std.	Pack2.5	Std.	Pack2.5
0x00	1	1M	1.25M	8M	10M
0x01	2	512K	640K	4M	5M
0x02	4	256K	320K	2M	2.5M
0x03	8	128K	160K	1M	1.25M
0x04	16	64K	80K	512K	640K
0x05	32	32K	40K	256K	320K
0x06	64	16K	20K	128K	160K
0x07	128	8K	10K	64K	80K
0x08	256	4K	5K	32K	40K
0x09	512	2K	2.5K	16K	20K
0x0A	1024	1K	1.25K	8K	10K

A write access to this register causes a Software Clear, see § 3.9. This register must not be written while acquisition is running.

4.16. Buffer Free (0x8010; r/w)

Bit	Function
[11:0]	N = Frees the first N Output Buffer Memory Blocks, see § 4.15

4.17. Custom Size (0x8020; r/w)

Bit	Function
[31:0]	0= Custom Size disabled N_{LOC} ($\neq$0) = Number of memory locations per event (1 location = 2 samples or 2 locations = 5 samples when Pack2.5 mode is used see § 3.3.4)

This register must not be written while acquisition is running.

4.18. Acquisition Control (0x8100; r/w)

Bit	Function
[3]	0 = COUNT ACCEPTED TRIGGERS 1 = COUNT ALL TRIGGERS allows to reject overlapping triggers (see § 3.3.3)
[2]	0 = Acquisition STOP 1 = Acquisition RUN allows to RUN/STOP Acquisition
[1:0]	00 = REGISTER-CONTROLLED RUN MODE 01 = S-IN CONTROLLED RUN MODE 10 = S-IN GATE MODE 11 = MULTI-BOARD SYNC MODE

Bit [2] allows to Run and Stop data acquisition; when such bit is set to 1 the board enters Run mode and a Memory Reset (see § 3.9.2) is automatically performed. When bit [2] is reset to 0 the stored data are kept available for readout. In Stop Mode all triggers are neglected.

Bits [1:0] descritpion:

00 = REGISTER-CONTROLLED RUN MODE: multiboard synchronisation via S_IN front panel signal

– RUN control: start/stop via set/clear of bit[2]
– GATE always active (Continuous Gate Mode)

01 = S-IN CONTROLLED RUN MODE: Multiboard synchronisation via S-IN front panel signal

– S-IN works both as SYNC and RUN_START command

– GATE always active (Continuous Gate Mode)
10 = S-IN GATE MODE
– Multiboard synchronisation is disabled
– S-IN works as Gate signal set/clear of RUN/STOP bit
11 = MULTI-BOARD SYNC MODE
– Used only for Multiboard synchronisation

4.19. Acquisition Status (0x8104; r)

Bit	Function
[8]	Board ready for acquisition (PLL and ADCs are synchronised correctly) 0 = not ready 1 = ready This bit should be checked after software reset to ensure that the board will enter immediatly run mode after RUN mode setting; otherwise a latency between RUN mode setting and Acquisition start might occur.
[7]	PLL Status Flag (see § 2.5.1): 0 = PLL loss of lock 1 = no PLL loss of lock NOTE: flag can be restored to 1 via read access to Status Register (see § 4.34)
[6]	PLL Bypass mode (see § 2.5.1): 0 = No bypass mode 1 = Bypass mode
[5]	Clock source (see § 2.6): 0 = Internal 1 = External
[4]	EVENT FULL: it is set to 1 as the maximum nr. of events to be read is reached
[3]	EVENT READY: it is set to 1 as at least one event is available to readout
[2]	0 = RUN off 1 = RUN on
[1:0]	*reserved*

4.20. Software Trigger (0x8108; w)

Bit	Function
[31:0]	A write access to this location generates a trigger via software

4.21. Trigger Source Enable Mask (0x810C; r/w)

Bit	Function
[31]	0 = Software Trigger Disabled 1 = Software Trigger Enabled
[30]	0 = External Trigger Disabled 1 = External Trigger Enabled
[29:27]	*reserved*
[26:24]	Local trigger coincidence level (default = 0)
[23:8]	*reserved*
[7]	0 = Channel 7 trigger disabled 1 = Channel 7 trigger enabled
[6]	0 = Channel 6 trigger disabled 1 = Channel 6 trigger enabled

[5]	0 = Channel 5 trigger disabled 1 = Channel 5 trigger enabled
[4]	0 = Channel 4 trigger disabled 1 = Channel 4 trigger enabled
[3]	0 = Channel 3 trigger disabled 1 = Channel 3 trigger enabled
[2]	0 = Channel 2 trigger disabled 1 = Channel 2 trigger enabled
[1]	0 = Channel 1 trigger disabled 1 = Channel 1 trigger enabled
[0]	0 = Channel 0 trigger disabled 1 = Channel 0 trigger enabled

This register bits[0,7] enable the channels to generate a local trigger as the digitised signal exceeds the Vth threshold (see § 3.5.3). Bit0 enables Ch0 to generate the trigger, bit1 enables Ch1 to generate the trigger and so on.
Bits [26:24] allows to set minimum number of channels that must be over threshold, beyond the triggering channel, in order to actually generate the local trigger signal; for example if bit[7:0]=FF (all channels enabled) and Local trigger coincidence level = 1, whenever one channel exceeds the threshold, the trigger will be generated only if at least another channel is over threshold at that moment. Local trigger coincidence level must be smaller than the number of channels enabled via bit[7:0] mask.
EXTERNAL TRIGGER ENABLE (bit30) enables the board to sense TRG-IN signals
SW TRIGGER ENABLE (bit 31) enables the board to sense software trigger (see § 4.20).

4.22. Front Panel Trigger Out Enable Mask (0x8110; r/w)

Bit	Function
[31]	0 = Software Trigger Disabled 1 = Software Trigger Enabled
[30]	0 = External Trigger Disabled 1 = External Trigger Enabled
[29:8]	*reserved*
[7]	0 = Channel 7 trigger disabled 1 = Channel 7 trigger enabled
[6]	0 = Channel 6 trigger disabled 1 = Channel 6 trigger enabled
[5]	0 = Channel 5 trigger disabled 1 = Channel 5 trigger enabled
[4]	0 = Channel 4 trigger disabled 1 = Channel 4 trigger enabled
[3]	0 = Channel 3 trigger disabled 1 = Channel 3 trigger enabled
[2]	0 = Channel 2 trigger disabled 1 = Channel 2 trigger enabled
[1]	0 = Channel 1 trigger disabled 1 = Channel 1 trigger enabled
[0]	0 = Channel 0 trigger disabled 1 = Channel 0 trigger enabled

This register bits[0,7] enable the channels to generate a TRG_OUT front panel signal as the digitised signal exceeds the Vth threshold (see § 3.5.3).
Bit0 enables Ch0 to generate the TRG_OUT, bit1 enables Ch1 to generate the TRG_OUT and so on.
EXTERNAL TRIGGER ENABLE (bit30) enables the board to generate the TRG_OUT
SW TRIGGER ENABLE (bit 31) enables the board to generate TRG_OUT (see § 4.20).

4.23. Post Trigger Setting (0x8114; r/w)

Bit	Function
[31:0]	Post trigger value

The register value sets the number of post trigger samples. The number of post trigger samples is :
Npost = PostTriggerValue*4 + ConstantLatency; where:
Npost = number of post trigger samples.
PostTriggerValue = Content of this register.
ConstantLatency = constant number of samples added due to the latency associated to the trigger processing logic in the ROC FPGA; this value is constant, but the exact value may change between different firmware revisions.

4.24. Front Panel I/O Data (0x8118; r/w)

Bit	Function
[15:0]	Front Panel I/O Data

Allows to Readout the logic level of LVDS I/Os and set the logic level of LVDS Outputs.

4.25. Front Panel I/O Control (0x811C; r/w)

Bit	Function
[15]	0 = I/O Normal operations: TRG-OUT signals outside trigger presence (trigger are generated according to Front Panel Trigger Out Enable Mask setting, see § 4.22) 1= I/O Test Mode: TRG-OUT is a logic level set via bit 14
[14]	1= TRG-OUT Test Mode set to 1 0 = TRG-OUT Test Mode set to 0
[13:8]	*reserved*
[7:6]	00 = General Purpose I/O 01 = Programmed I/O 10 = Pattern mode: LVDS signals are input and their value is written into header PATTERN field (see § 3.3.4)
[5]	0 = LVDS I/O 15..12 are inputs 1 = LVDS I/O 15..12 are outputs
[4]	0 = LVDS I/O 11..8 are inputs 1 = LVDS I/O 11..8 are outputs
[3]	0 = LVDS I/O 7..4 are inputs 1 = LVDS I/O 7..4 are outputs
[2]	0 = LVDS I/O 3..0 are inputs 1 = LVDS I/O 3..0 are outputs
[1]	0= panel output signals (TRG-OUT/CLKOUT) enabled 1= panel output signals (TRG-OUT/CLKOUT) enabled in high impedance
[0]	0 = TRG/CLK are NIM I/O Levels 1 = TRG/CLK are TTL I/O Levels

Bits [5:2] are meaningful for General Purpose I/O use only

4.26. Channel Enable Mask (0x8120; r/w)

Bit	Function
[7]	0 = Channel 7 disabled 1 = Channel 7 enabled
[6]	0 = Channel 6 disabled 1 = Channel 6 enabled

[5]	0 = Channel 5 disabled 1 = Channel 5 enabled
[4]	0 = Channel 4 disabled 1 = Channel 4 enabled
[3]	0 = Channel 3 disabled 1 = Channel 3 enabled
[2]	0 = Channel 2 disabled 1 = Channel 2 enabled
[1]	0 = Channel 1 disabled 1 = Channel 1 enabled
[0]	0 = Channel 0 disabled 1 = Channel 0 enabled

Enabled channels provide the samples which are stored into the events (and not erased). The mask cannot be changed while acquisition is running.

4.27. ROC FPGA Firmware Revision (0x8124; r)

Bit	Function
[31:16]	Revision date in Y/M/DD format
[15:8]	Firmware Revision (X)
[7:0]	Firmware Revision (Y)

Bits [31:16] contain the Revision date in Y/M/DD format.
Bits [15:0] contain the firmware revision number coded on 16 bit (X.Y format).

4.28. Event Stored (0x812C; r)

Bit	Function
[31:0]	This register contains the number of events currently stored in the Output Buffer

This register value cannot exceed the maximum number of available buffers according to setting of buffer size register.

4.29. Set Monitor DAC (0x8138; r/w)

Bit	Function
[11:0]	This register allows to set the DAC value (12bit)

This register allows to set the DAC value in Voltage level mode (see § 2.7).
LSB = 0.244 mV, terminated on 50 Ohm.

4.30. Board Info (0x8140; r)

Bit	Function
[15:8]	Memory size (Mbyte/channel)
[7:0]	Board Type: 1 = V1720

4.31. Monitor Mode (0x8144; r/w)

Bit	Function
[2:0]	This register allows to encode the Analog Monitor (see § 3.7) operation: 000 = majority 001 = waveform generator (saw tooth ramp) 010 = *reserved* 011 = buffer occupancy 100 = voltage level

4.32. Event Size (0x814C; r)

Bit	Function
[31:0]	Nr. of 32 bit words in the next event

4.33. VME Control (0xEF00; r/w)

Bit	Function
[7]	0 = Release On Register Access (RORA) Interrupt mode (default) 1 = Release On AcKnowledge (ROAK) Interrupt mode
[6]	0 = RELOC Disabled (BA is selected via Rotary Switch; see § 2.6) 1 = RELOC Enabled (BA is selected via RELOC register; see § 4.37)
[5]	0 = ALIGN64 Disabled 1 = ALIGN64 Enabled (see § 3.12.1.2)
[4]	0 = BERR Not Enabled; the module sends a DTACK signal until the CPU inquires the module 1 = BERR Enabled; the module is enabled either to generate a Bus error to finish a block transfer or during the empty buffer read out in D32
[3]	0 = Optical Link interrupt disabled 1 = Optical Link interrupt enabled
[2 :0]	Interrupt level (0= interrupt disabled)

Bit [7]: this setting is valid only for interrupts broadcasted on VMEbus; interrupts broadcasted on optical link feature RORA mode only.

- In RORA mode, interrupt status can be removed by accessing VME Control register (see § 4.33) and disabling the active interrupt level.
- In ROAK mode, interrupt status is automatically removed via an interrupt acknowledge cycle.

Interrupt generation is restored by setting an Interrupt level > 0 via VME Control register.

4.34. VME Status (0xEF04; r)

Bit	Function
[2]	0 = BERR FLAG: no Bus Error has occurred 1 = BERR FLAG: a Bus Error has occurred (this bit is re-set after a status register read out)
[1]	0 = The Output Buffer is not FULL; 1 = The Output Buffer is FULL.
[0]	0 = No Data Ready; 1 = Event Ready

4.35. Board ID (0xEF08; r/w)

Bit	Function
[4 :0]	GEO

- VME64X versions: this register can be accessed in read mode only and contains the GEO address of the module picked from the backplane connectors; when CBLT is performed, the GEO address will be contained in the EVENT HEADER Board ID field (see § 3.3.4).
- Other versions: this register can be accessed both in read and write mode; it allows to write the correct GEO address (default setting = 0) of the module before CBLT operation. GEO address will be contained in the EVENT HEADER Board ID field)

4.36. MCST Base Address and Control (0xEF0C; r/w)

Bit	Function
[9:8]	Allows to set up the board for daisy chaining: 00 = disabled board 01 = last board 10 = first board 11 = intermediate
[7:0]	These bits contain the most significant bits of the MCST/CBLT address of the module set via VME, i.e. the address used in MCST/CBLT operations.

4.37. Relocation Address (0xEF10; r/w)

Bit	Function
[15..0]	These bits contains the A31...A16 bits of the address of the module: it can be set via VME for a relocation of the Base Address of the module.

4.38. Interrupt Status ID (0xEF14; r/w)

Bit	Function
[31..0]	This register contains the STATUS/ID that the module places on the VME data bus during the Interrupt Acknowledge cycle

4.39. Interrupt Event Number (0xEF18; r/w)

Bit	Function
[9:0]	INTERRUPT EVENT NUMBER

If interrupts are enabled, the module generates a request whenever it has stored in memory a Number of events > INTERRUPT EVENT NUMBER

4.40. BLT Event Number (0xEF1C; r/w)

Bit	Function
[7:0]	This register contains the number of complete events which has to be transferred via BLT/CBLT (see § 3.12.1.2).

4.41. Scratch (0xEF20; r/w)

Bit	Function
[31:0]	Scratch (*to be used to write/read words for VME test purposes*)

4.42. Software Reset (0xEF24; w)

Bit	Function
[31:0]	A write access to this location allows to perform a software reset

4.43. Software Clear (0xEF28; w)

Bit	Function
[31:0]	A write access to this location clears all the memories

4.44. Flash Enable (0xEF2C; r/w)

Bit	Function
[0]	0 = Flash write ENABLED 1 = Flash write DISABLED

This register is handled by the Firmware upgrade tool.

4.45. Flash Data (0xEF30; r/w)

Bit	Function
[7:0]	Data to be serialized towards the SPI On board Flash

This register is handled by the Firmware upgrade tool.

4.46. Configuration Reload (0xEF34; w)

Bit	Function
[31:0]	A write access to this register causes a software reset (see § 3.8), a reload of Configuration ROM parameters and a PLL reconfiguration.

5. Installation

– The Mod. V1720 fits into all 6U VME crates.
– VX1720 versions require VME64X compliant crates
– Turn the crate OFF before board insertion/removal
– Remove all cables connected to the front panel before board insertion/removal

CAUTION

ALL CABLES MUST BE REMOVED FROM THE FRONT PANEL BEFORE EXTRACTING THE BOARD FROM THE CRATE!

5.1. Power ON sequence

To power ON the board follow this procedure:

1. insert the V1720 board into the crate
2. power up the crate

5.2. Power ON status

At power ON the module is in the following status:

- the Output Buffer is cleared;
- registers are set to their default configuration (see § 4)

5.3. Firmware upgrade

The board can store two firmware versions, called STD and BKP respectively; at Power On, a microcontroller reads the Flash Memory and programs the module with the firmware version selected via the JP2 jumper (see § 2.6), which can be placed either on the STD position (left), or in the BKP position (right). It is possible to upgrade the board firmware via VME, by writing the Flash; for this purpose, download the software package available at:
http://www.caen.it/nuclear/product.php?mod=V1720

The package includes the new firmware release file:

- v1720_revX.Y_W.Z..rbf

and the V1720 firmware upgrade tool:

- CVUpgrade.exe (windows executable)
- CVUpgrade tool (source code and VC++ project)

For upgrading the firmware, utilizing CVUpgrade.exe, open a DOS shell, then launch

```
CVUpgrade FileName BaseAdd [image] [/fast] [/nover]
```

where:

- **FileName** is the RBF file
- **BaseAdd** is the Base Address (Hex 32 bit) of the V1720
- **image** is '/standard' (default) or '/backup'
- **'/fast'** enables fast programming (MultiRead/Write with CAEN Bridge)'
- **'/nover'** disables programming check

N.B.: it is strongly suggested to upgrade ONLY one of the stored firmware revisions (generally the STD one): if both revision are simultaneously updated, and a failure occurs, it will not be possible to upload the firmware via VME again!

5.3.1. V1720 Upgrade files description

The board hosts one FPGA on the mainboard and one FPGA for each of the eight channels. The channel FPGAs firmware is identical. A unique file is provided that will updated all the FPGA at the same time.

ROC FPGA MAINBOARD FPGA (Readout Controller + VME interface)

There is one FPGA Altera Cyclone EP1C20.

AMC FPGA CHANNEL FPGA (ADC readout/Memory Controller):

There is one FPGA Altera Cyclone EP1C4

All FPGAs can be upgraded via VMEBUS;

CVUpgrade utility program must be used for this purpose.

The programming file has the extension RBF and its name follows this general scheme:

v1720_revX.Y_W.Z.RBF

where:

- X.Y is the major/minor revision number of the mainboard FPGA
- W.Z is the major/minor revision number of the channel FPGA

WARNING: you can restore the previous FW revision in case there is a failure when you run the upgrading program. There is a jumper on the mainboard that allows to select the "backup" copy of the firmware. You must upgrade all the FPGAs and keep the revisions aligned; it is not guaranteed that the latest revision of one FPGA is compatible with an older revision.

Upgrade examples:

1) Upgrade to Rev 1.2(main FPGA)/Rev 0.2 (channel FPGA) of the standard page of the V1720:

CVUpgrade v1720_rev1.2_0.2.rbf 32100000 /standard

2) Upgrade to Rev 1.2(main FPGA)/Rev 0.2 (channel FPGA) of the backup page of the V1720:

CVUpgrade v1720_rev1.2_0.2.rbf 32100000 /backup

3) Upgrade to Rev 1.2(main FPGA)/Rev 1.1 (channel FPGA) of the standard page of the V1720:

CVUpgrade v1720_rev1.2_1.1.rbf 32100000 /standard

The board can store two firmware versions, called STD and BKP respectively; at Power On, a microcontroller reads the Flash Memory and programs the module with the firmware version selected via the JP2 jumper (see § 2.6), which can be placed either on the STD position (left), or in the BKP position (right). It is possible to upgrade the board firmware via VME, by writing the Flash; for this purpose, download the software package available at:
http://www.caen.it/nuclear/product.php?mod=V1720

The package includes the new firmware release file:

- V1720_rN_revX.Y_W.Z..rbf

and the V1720 firmware upgrade tool:

- CVUpgrade.exe (windows executable)
- CVUpgrade tool (source code and VC++ project)

For upgrading the firmware, utilizing CVUpgrade.exe, open a DOS shell, then launch

```
CVUpgrade FileName BaseAdd [image] [/fast] [/nover]
```

where:

- **FileName** is the RBF file
- **BaseAdd** is the Base Address (Hex 32 bit) of the V1720
- **image** is '/standard' (default) or '/backup'
- **'/fast'** enables fast programming (MultiRead/Write with CAEN Bridge)'
- **'/nover'** disables programming check

N.B.: it is strongly suggested to upgrade ONLY one of the stored firmware revisions (generally the STD one): if both revision are simultaneously updated, and a failure occurs, it will not be possible to upload the firmware via VME again!