Technical Information Manual

Revision n. 10
01 July 2014

MOD. V2718 • VX2718

VME – PCI
OPTICAL LINK BRIDGE
MANUAL REV. 10

NPO:
00106/03:V2718.MUTx/10

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

LIST OF FIGURES

LIST OF TABLES

1. General description

1.1. Overview

The Mod. V2718 is a 1-unit wide 6U VME master module, which can be interfaced to the CONET (Chainable Optical NETwork) and controlled by a standard PC equipped with the PCI card CAEN Mod. A2818 or the PCIe card A3818. The A2818 is a 32-bit 33 MHz PCI card supporting both the CONET1 old version of CAEN optical link communication portocol and the latest CONET2 one (for details, please refer to the Application Note "AN2472 - CONET1 to CONET2 migration" freely downloadable on CAEN website at *Home / Document Library*). The A3818 is a PCI Express (v1.1 or higher) card that can plug into both x8 and x16 PCI Express slot and allows the control of up to 4 CONET2 independent networks. A3818 doesn't support CONET1 protocol.
The communication path uses optical fiber cables as physical transmission line. Up to 8 V2718 VME masters can be controlled by one A2818 PCI controller, while up to 32 ones can be controlled by one A3818 PCIe controller.

The module is capable of performing all the cycles foreseen by the VME64X specifications[1].

Important note: the Mod. VX2718 is the VME64X mechanics version of the module and requires a VME64X type crate; the Mod. VN2738 is the VNX9 (9 Unit) mechanics version of the module and requires a VNX9 type crate. In the present manual the "generic" term "V2718" refers to all versions, except as otherwise specified.

The module can work in a "multimaster" system with the possibility of operating as a system controller, in this case (which is the default option as the board is inserted in the slot 1), it works as Bus Arbiter, Sysclock Driver, IACK Daiy Chain Driver, etc.

The module features a LED display which allows to monitor the VME bus activity in detail. The front panel features 5 TTL/NIM programmable outputs on LEMO 00 connectors (default assignment is: DS, AS, DTACK, BERR signals and the output of a programmable Location Monitor) and two programmable TTL/NIM inputs (on LEMO 00 connectors).[2]

Operation as a Slave module is available for reading the Dataway display and the Internal Test RAM.
The sustained data transfer rate is up to 70 MByte/s by using CONET1 and up to 80 MByte/s by using CONET2. Thanks to the 128KB memory buffer, the activity on the VME bus is not slowed down by the transfer rate on the CONET when several V2718's share the same network. The firmware updates, documentation and useful software supporting the use with the most common PC platforms (Windows XP/VISTA/7, Linux), are available on CAEN website at:
Home / Products / Modular Pulse Processing Electronics / VME / Controller (VME) / V2718.

[1] 2eVME cycles and 3U boards cycles are not implemented yet.

[2] LED display and TTL/NIM I/Os are not available on Mod. V/VX2718LC versions

Table 1.1: Available items

Code	Description	LED display	TTL/NIM I/Os	Form factor
WK2718LCXAAA	V2718KITLC - VME-PCI Bridge (V2718) + PCI Optical Link (A2818) + Optical Fibre 5m duplex (AY2705)	no	no	VME6U
WK2718XAAAAA	V2718KIT - VME-PCI Bridge (V2718) + PCI Optical Link (A2818) + Optical Fibre 5m duplex (AY2705)	yes	yes	VME6U
WKX2718LCXAA	VX2718KITLC - VME-PCI Bridge (VX2718) + PCI Optical Link (A2818) + Optical Fibre 5m duplex (AY2705)	no	no	VME64X
WKX2718XAAAA	VX2718KIT - VME-PCI Bridge (VX2718) + PCI Optical Link (A2818) + Optical Fibre 5m duplex (AY2705)	yes	yes	VME64X
WK2718XBAAAA	V2718KITB - VME-PCI Bridge (V2718) + PCIe Optical Link (A3818A) + Optical Fibre 5m duplex (AY2705)	yes	yes	VME6U
WKX2718XBAAA	VX2718KITB - VME-PCI Bridge (VX2718) + PCIe Optical Link (A3818A) + Optical Fibre 5m duplex (AY2705)	yes	yes	VME64X
WV2718LCXAAA	V2718LC - VME-PCI Bridge	no	no	VME6U
WV2718XAAAAA	V2718 - VME-PCI Bridge	yes	yes	VME6U
WVX2718LCXAA	VX2718LC - VME-PCI Bridge	no	no	VME64X
WVX2718XAAAA	VX2718 - VME-PCI Bridge	yes	yes	VME64X
WKN2738XAAAA	VN2738KIT - VN2738 + A2818 Kit + Optical Fibre 20m	yes	yes	VME9U
WVN2738XAAAA	VN2738 - 9U VME-PCI Bridge	yes	yes	VME9U
WA2818XAAAAA	A2818 - PCI Optical Link			PCI
WA3818AXAAAA	A3818A - PCIe 1 Optical Link			PCIe
WA3818BXAAAA	A3818B - PCIe 2 Optical Link			PCIe
WA3818CXAAAA	A3818C - PCIe 4 Optical Link			PCIe
WAI2703XAAAA	AI2703 - Optical Fibre 30cm. simplex			
WAI2705XAAAA	AI2705 - Optical Fibre 5 m. simplex			
WAI2720XAAAA	AI2720 - Optical Fibre 20 m. simplex			
WAI2730XAAAA	AI2730 - Optical Fibre 30 m. simplex			
WAI2740XAAAA	AI2740 - Optical Fibre 40 m. simplex			
WAY2705XAAAA	AY2705 - Optical Fibre 5 m. duplex (Rohs compliant)			
WAY2720XAAAA	AY2720 - Optical Fibre 20 m. duplex			
WAY2730XAAAA	AY2730 - Optical Fibre 30 m. duplex			

1.2. Block diagram

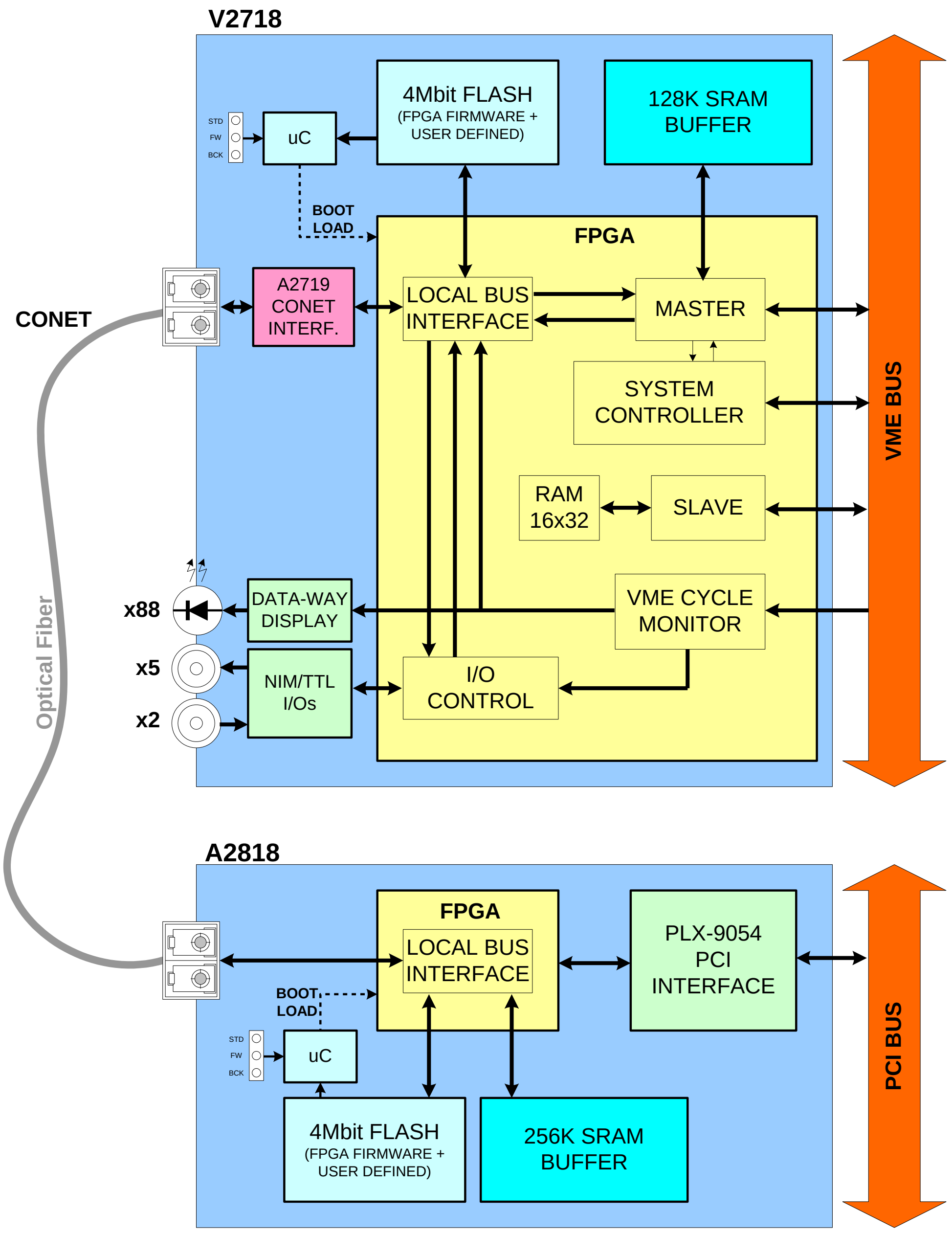


Fig. 1.1: Mod. V2718 block diagram

The FPGA (Field Programmable Gate Array) is the module's core; it implements the CONET communication protocol, the LED display and I/O connectors management on the front side and the VME Master on the backside.
A 128 kbyte buffer allows to provide a temporary data storage during VME cycles: the VME data rate is thus decoupled from the PCI rate and may take place at full speed.

1.3. CONET Layout

Thanks to Daisy chain capability, a single A2818 CONET controller can control up to 8 V2718 VME masters, while a single A3818 CONET2 controller (e.g. the 4-link model) an control upt to 32 V2718 VME masters. For this purpose, various types of cables are available:

Table 1.2: CONET cables specifications

Cable:	Length:	Connector:
X-30	30 m	1 LC Duplex + 2 LC Simplex
X-20	20 m	1 LC Duplex + 2 LC Simplex
X-5	5 m	1 LC Duplex + 2 LC Simplex
I-40	40 m	2 LC Simplex
I-30	30 m	2 LC Simplex
I-20	20 m	2 LC Simplex
I-5	5 m	2 LC Simplex
I-3	30 cm	2 LC Simplex

If the network is composed by one A2818, or A3818, and only one V2718, then it is suggested to use X-type cables: such cables have a duplex connector on the A2818 or A3818 side and two simplex connectors on the crate side; the simplex connector with the black wrap is for the RX line and the one with the red wrap is for the TX. If more than one V2718 is present, the best solution is to use the X-type cable for connecting the A2818, or the A3818, with the first and the last module and the I-type for connecting intermediate modules. An example using the A2818 is given in **Fig. 1.2**.

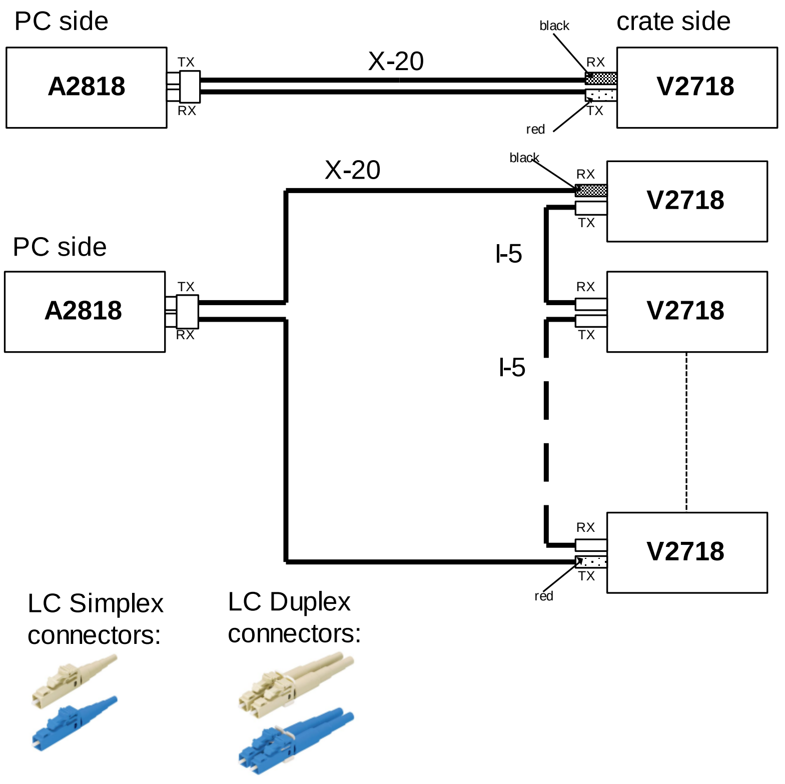


Fig. 1.2: CONET cables layout

2. VME Interface

The V2718 provides all of the addressing and data transfer modes documented in the VME64 specification (except A64 and those intended to improve 3U applications, i.e. A40 and MD32). The V2718 is also compatible with all VME bus modules compliant to pre-VME64 specifications. As VME bus master, the V2718 supports Read-Modify-Write (RMW), and Address-Only-with-Handshake (ADOH) but does not accept RETRY* as a termination from the VME bus slave. The ADOH cycle is used to implement the VME bus Lock command allowing the PC Host to lock VME bus resources.

2.1. VME bus Requester

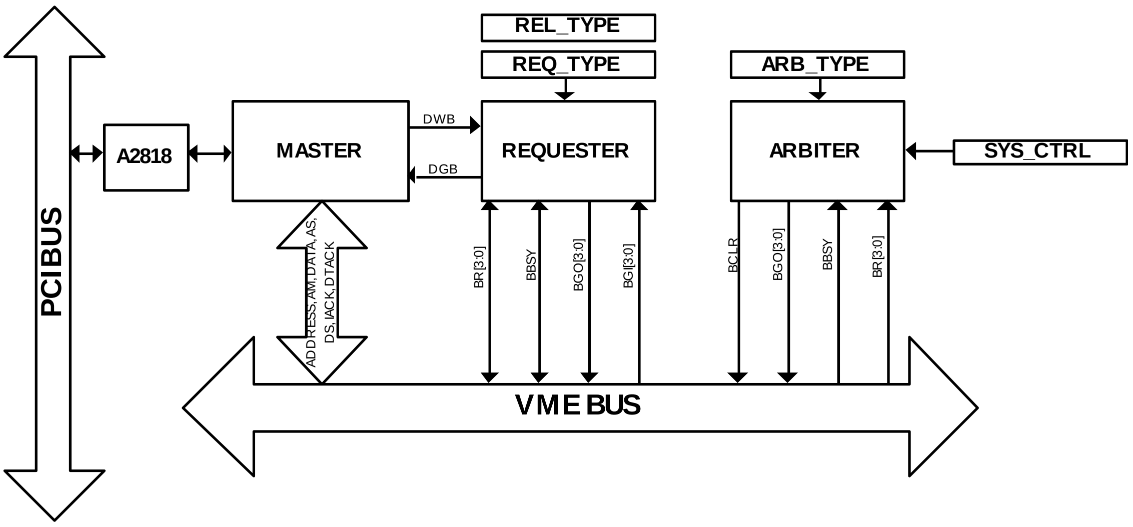


Fig. 2.1: Internal Arbitration for VME bus Requests

When the V2718 operates as *VME bus Requester*, the functional sequence is the following:

- The PCI bus sends a VME bus access request
- The Master asserts DWB (Device Want Bus), and waits for DGB (Device Grant Bus)
- The Requester requests the bus to the Arbiter, via VME (whether the Arbiter is the V2718 itself or not); when the Arbiter has granted the bus, the Requester asserts DGB and BBSY (on the bus)
- The Master performs the the VME cycle, then releases DWB
- If REL_TYPE is RWD (Release When Done), then the Requester releases BBSY

2.1.1. Fair and Demand Request modes

The V2718 produces requests on all VME bus request levels: BR3*, BR2*, BR1*, and BR0*. The default setting is for level 3 VME bus request. The request level is a global programming option set through the Bus Request field in the Control register (see § **2.13.2**).
The programmed request level is used by the VME bus Master Interface regardless of the channel currently accessing the VME bus Master Interface.
The Requester may be programmed for either Fair or Demand mode. The request mode is a global programming option set through the Requester Type bit in the Control register.
In *Fair mode*, the V2718 does not request the VME bus until there are no other VME bus requests pending at its programmed level. This mode ensures that every requester on an equal level has access to the bus.
In *Demand mode*, the requester asserts its bus request regardless of the state of the BRn* line. By requesting the bus frequently, requesters far down the daisy chain may be prevented from ever obtaining bus ownership. This is referred to as “starving” those requesters. Note that in order to achieve fairness, all bus requesters in a VME bus system must be set to Fair mode.

2.1.2. VME bus Release

The Requester can be configured as either RWD (release when done) or ROR (release on request) using the Release Type bit in the Control register. The default setting is for RWD: the bus is released as soon as the VME access is terminated; in case of BLT/MBLT cycles, the access is terminated either when the N required bytes are transferred (although the cycle is divided into several blocks according to the VME boundaries) or when BERR* is asserted. ROR means the master releases BBSY* only if a bus request is pending from another VMEbus master and once the channel that is the current owner of the VME bus Master Interface is done. Ownership of the bus may be assumed by another channel without re-arbitration on the bus if there are no pending requests on any level on the VME bus.

2.2. Addressing capabilities

V2718 generates A16, A24, A32, CR/CSR and LCK address phases on the VME bus. Address Modifiers of any kind (supervisor/non-privileged and program/data) are also programmed through the PCI bus: the V2718 does not handle the AM: the PC Host passes them via PCI as VME cycle parameters. The AM broadcasting depends on the PC drivers.
The master generates ADdress-Only-with-Handshake (ADOH) cycles in support of lock commands for A16, A24, and A32 spaces.

Supported addressing:

A16, A24, A32, CR/CSR	for R/W, RMW, ADO and ADOH
A16, A24, A32	for BLT
A16, A24, A32	for MBLT
ADO	Address Only
ADOH	Address Only with Handshake

2.3. Data transfer capabilities

The V2718 supports the following cycles:

Cycle Type	
R/W	Single Read/Write
RMW	Read Modify Write
BLT	Block Transfer
MBLT	Multiplexed Block Transfer

Data sizing	
D08(EO), D16, D32	for R/W, RMW, BLT[3]
D64	for MBLT

- BLT/MBLT cycles may be performed with either address increment or with fixed address (FIFO mode)
- BLT/MBLT cycles are split at hardware level when the boundary (BLT = Nx256 bytes; MBLT = Nx2 Kbytes) is met: AS is released and then re-asserted, the VME bus is not re-arbitered. The boundaries are neglected in FIFO operating mode.
- Non aligned accesses are not supported.

It is then possible to perform data cycles (single and BLT) with hardware byte swapping. The “Swapped” cycles are called: D16_swapped, D32_swapped and D64_swapped. Such cycles will return “swapped” data, in the following way:

D16_swapped: Byte0 ↔ Byte1, Byte1 ↔ Byte0
D32_swapped: Byte0 ↔ Byte3, Byte1 ↔ Byte2, Byte2 ↔ Byte1, Byte3 ↔ Byte0
D32_swapped: Byte0 ↔ Byte7, Byte1 ↔ Byte6, Byte2 ↔ Byte5, Byte3 ↔ Byte4, Byte4 ↔ Byte3, Byte5 ↔ Byte2, Byte6 ↔ Byte1, Byte7 ↔ Byte0

2.4. Interrupt capabilities

The VME Bus interrupts are transferred to the PCI BUS through the CONET. The interrupt latency (i.e. the interval between the interrupt appearance on the VME bus and the time the interrupt is activated on the PCI bus) is always shorter than 5 µs.
The V2718 supports the following IACK cycles:

IACK:	D08, D16, D32

VME Bus Interrupts can be individually masked for each V2718 in the chain.
In order to enable the generation of PCI bus interrupts following VME bus interrupts, the IRQEnable function (see § **4.3.48**) must be used; then it is necessary to call IRQWait (see § **4.3.50**) in order to wait for the interrupt. When the IRQWait function returns, the VME bus interrupts are disabled, so an IACK can be performed in order to obtain the vector and, for RORA interrupts, the access to the interrupter must be performed in order to stop the interrupt generation. If it is necessary to receive other VME bus interrupts, the IRQEnable must be called again.

[3] BLT08 not implemented

2.5. Cycle terminations

The V2718 accepts BERR* or DTACK* as cycle terminations. BERR* is handled as cycle termination whether it is produced by the V2718 itself or by another board. The Status word broadcasted as the cycle is acknowledged, informs the PC HOST about the cycle termination type (BERR* or DTACK*).

2.6. Slave

When the V2718 operates as slave, it responds to VME cycles (which must be initiated by another master, i.e. a V2718 cannot *address itself* as a slave) for accessing the Dataway Display internal registers and a Test RAM (32 x 16). The V2718 is accessed both with A32 and A24 base address (see § **3.5.1**); the module is provided with only two rotary switches for board addressing, so the addressing mode is selected via the dip switch 3 (A24⟶ PROG_3 = OFF; A32⟶ PROG_3 = ON), see § **3.5.1**.

The Address map for V2718 is listed in Table 2.1. All register addresses are referred to the Base Address of the board, i.e. the addresses reported in the Tables are the offsets to be added to the board Base Address.

Table 2.1: Address Map for the Model V2718

ADDRESS	REGISTER/CONTENT	ADDR_MODE	DATA_MODE	R/W
Base + %0000÷%00FC	Test RAM	A24/A32	D32, BLT32, MBLT	Read/Write
Base + %1000	Display Address	A24/A32	D32	Read only
Base + %1004	Display Data	A24/A32	D32	Read only
Base + %1008	Display Control	A24/A32	D32	Read only

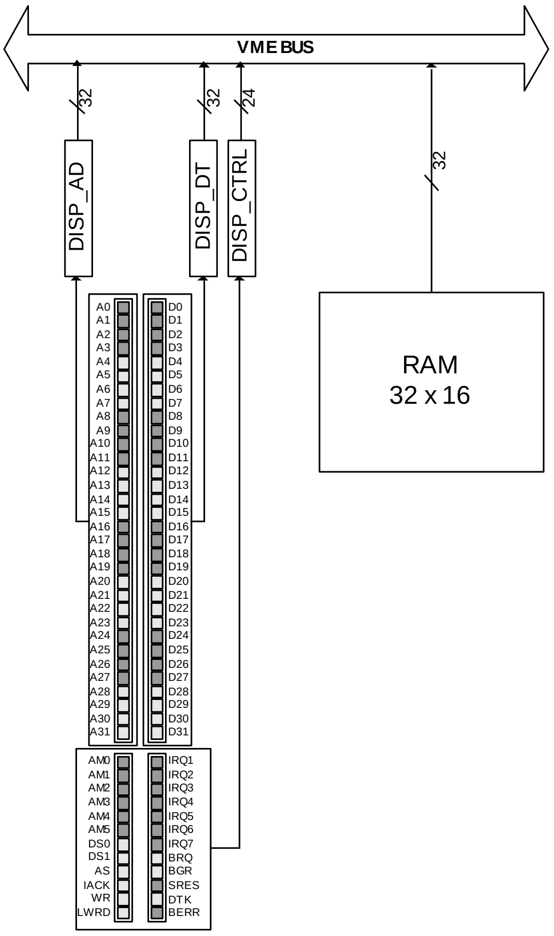

VME BUS
32
32
24
32
DISP_AD
DISP_DT
DISP_CTRL
RAM 32 x 16
A0 … A31
D0 … D31
AM0 AM1 AM2 AM3 AM4 AM5 DS0 DS1 AS IACK WR LWRD
IRQ1 IRQ2 IRQ3 IRQ4 IRQ5 IRQ6 IRQ7 BRQ BGR SRES DTK BERR

Fig. 2.2: V2718 Slave operation

2.7. Location Monitor

The V2718 monitors the cycles on the bus, whether they are held by itself or by other masters, and produces a Trigger Out LMON signal as soon as a particular cycle is performed (see **Fig. 2.3**). The LMON out is available by default as front panel signal.

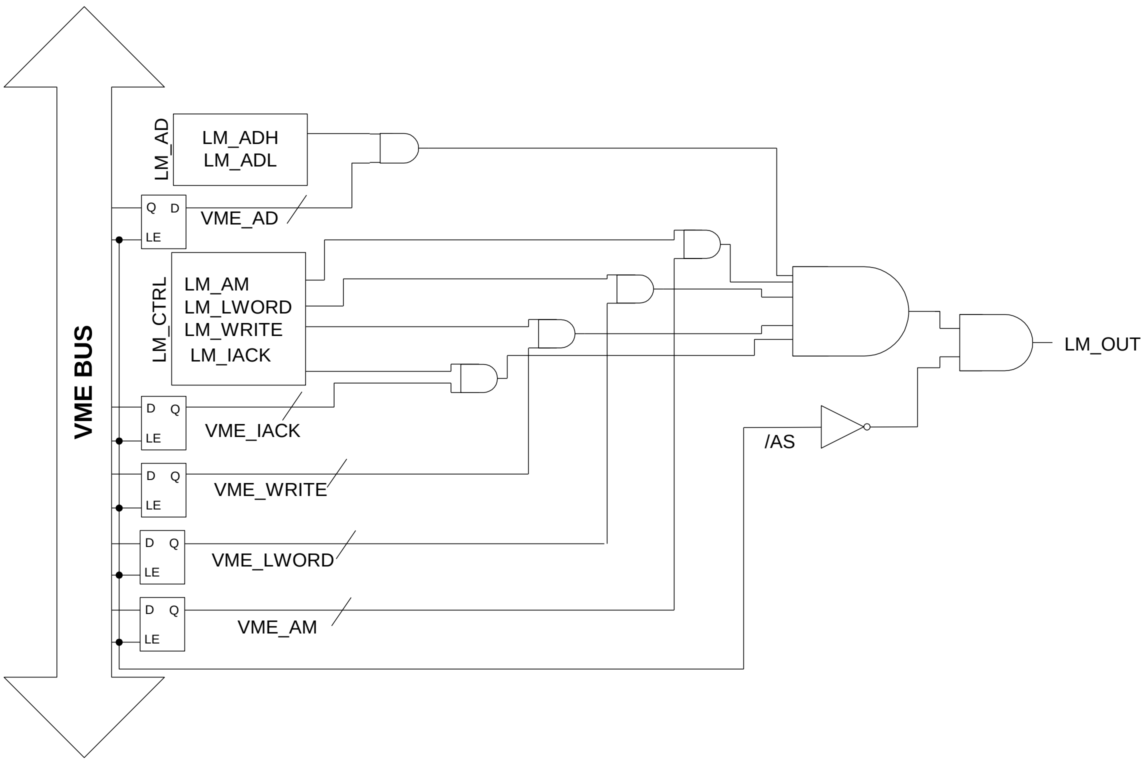


Fig. 2.3: The Location Monitor

2.8. VME bus First Slot Detector

The First Slot Detector module samples BG3IN* immediately after reset to determine whether the V2718 resides in slot 1. The VME bus specification requires that BG[3:0]* lines be driven high during reset. This means that if a board is preceded by another board in the VME bus system, it will always sample BG3IN* high after reset. BG3IN* can only be sampled low after reset by the first board in the crate (there is no preceding board to drive BG3IN* high). If BG3IN* is sampled at logic low immediately after reset (due to the master internal pull-down), then the V2718 is in slot 1 and becomes SYSTEM CONTROLLER: otherwise, the SYSTEM CONTROLLER module is disabled. This mechanism may be overridden via dip switch setting: the SYSTEM CONTROLLER bit is “forced” to one by setting to ON PROG_0, and is “forced” to zero by setting to ON PROG_1; note that such switches must always be in “opposite” positions (see § **3.5.1**).

2.9. System Controller Functions

When located in Slot 1 of the VME crate, the V2718 assumes the role of SYSTEM CONTROLLER and sets the SYSTEM CONTROLLER status bit in the STATUS register. In accordance with the VME64 specification, as SYSTEM CONTROLLER the V2718 provides:

– a system clock driver,
– an arbitration module,
– an IACK Daisy Chain Driver (DCD)
– a bus timer.

2.9.1. System Clock Driver

The V2718 provides a 16.66 MHz SYSCLK signal when configured as System Controller.

2.9.2. Arbitration Module

When the V2718 is SYSTEM CONTROLLER, the Arbitration Module is enabled. The Arbitration
Module supports the following arbitration modes:

– Fixed Priority Arbitration Mode (PRI),
– Round Robin Arbitration Mode (RRS) (default setting).

These are set with the Arbiter bit in the STATUS register

2.9.2.1. Fixed Priority Arbitration Mode (PRI)

In this mode, the order of priority is BR[3], BR[2], BR[1], and BR[0] as
defined by the VME64 specification. The Arbitration Module issues a Bus Grant (BGO [3:0]) to the highest requesting level.
If a Bus Request of higher priority than the current bus owner becomes asserted, the Arbitration Module asserts BCLR until the owner releases the bus (BBSY is negated).

2.9.2.2. Round Robin Arbitration Mode (RRS)

This mode arbitrates all levels in a round robin mode, repeatedly scanning from levels 3 to 0.
Only one grant is issued per level and one owner is never forced from the bus in favor of another requester (BCLR is never asserted).
Since only one grant is issued per level on each round robin cycle, several scans will be required to service a queue of requests at one level.

2.10. Bus Timer

A programmable bus timer allows users to select a VMEbus time-out period. The time-out period is programmed through the Bus Timeout bit in the Control register (= 0 → timeout = 50 µs; = 1 → timeout = 400µs). The VMEbus Timer module asserts BERR if a VMEbus transaction times out (indicated by one of the VMEbus data strobes remaining asserted beyond the time-out period).

2.11. IACK Daisy Chain Driver

The V2718 can operate as IACK Daisy Chain Driver: it drives low the IACKOUT line of the first slot, thus starting the chain propagation, as soon as it detects an Interrupt Acknowledge cycle by an Interrupt Handler, that could be the V2718 itself.

2.12. VME64X Cycles not yet implemented

Presently the module does not implement the following functions, foreseen by the VME64X:

Unaligned Transfer (UAT)

MD32 cycles

2eVME cycles

BLT08 cycles

A64 addresing

Cycles terminated with RETRY

2.13. Internal registers

Table 2.2: Registers map

NAME	ADDRESS	Type	Nbit	Function
STATUS	00	read	16	Status register
VME_CTRL	01	read/write	16	VME control register
FW_REV	02	read only	16	Firmware revision
FW_DWNLD	03	read/write	8	Firmware download
FL_ENA	04	read/write	1	Flash enable
IRQ_STAT	05	read only	7	IRQ status
IRQ_MASK	06	read/write	7	IRQ mask
IN_REG	08	read/write	7	Front panel input register
OUT_REG_S	0A	read/write	11	Front panel output register set
IN_MUX_S	0B	read/write	12	Input multiplexer set
OUT_MUX_S	0C	read/write	15	Output multiplexer set
LED_POL_S	0D	read/write	7	LED polarity set
OUT_REG_C	10	write only	11	Front panel output register clear
IN_MUX_C	11	write only	12	Input multiplexer clear
OUT_MUX_C	12	write only	15	Output multiplexer clear
LED_POL_C	13	write only	7	LED polarity clear
PULSEA_0	16	read/write	16	Period and width of pulser A
PULSEA_1	17	read/write	10	# pulses and range of pulser A
PULSEB_0	19	read/write	16	Period and width of pulser B
PULSEB_1	1A	read/write	10	# pulses and range of pulser B
SCALER0	1C	read/write	11	End Count Limit and Autores of scaler
SCALER1	1D	read only	10	Counter value of scaler
DISP_ADL	20	read only	16	Display AD [15:0]
DISP_ADH	21	read only	16	Display AD [31:16]
DISP_DTL	22	read only	16	Display DT [15:0]
DISP_DTH	23	read only	16	Display DT [31:16]
DISP_PC1	24	read only	12	Display control left bar
DISP_PC2	25	read only	12	Display control right bar
LM_ADL	28	read/write	16	Local monitor AD [15:0]
LM_ADH	29	read/write	16	Local monitor AD [31:16]
LM_C	2C	read/write	9	Local monitor controls

2.13.1. Status register

(Base Address + 0x00, D16, read/write)

This register contains information on the status of the module.

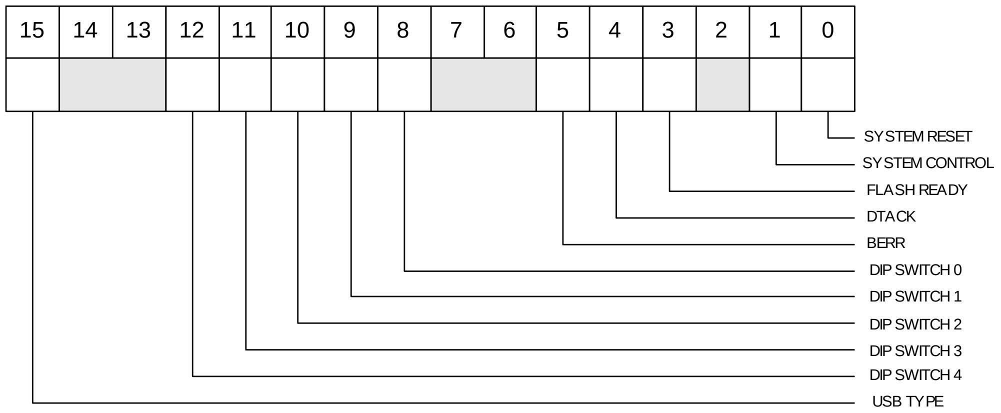


Fig. 2.4: Status Register

SYSTEM RESET:	0 = Inactive 1 = Active
SYSTEM CONTROL:	0 = Disabled 1 = Enabled
DTACK:	1 = Last cycle terminated with DTACK 0 = Any other case
BERR:	1 = Last cycle terminated with BERR 0 = Any other case
DIP SWITCH [4:0]:	0 = Switch set to OFF 1 = Switch set to ON

2.13.2. Control register

(Base Address + 0x01, D16, read/write)

This register allows performing some general settings of the module.

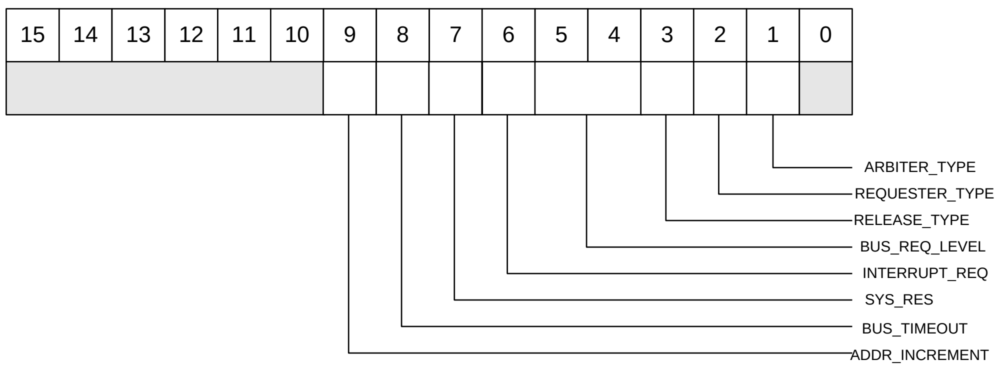


Fig. 2.5: Control Register

Arbiter Type:	0 = Fixed Priority 1 = Round Robin
Requester Type:	0 = Fair 1 = Demand
Release Type:	0 = Release when done 1 = Release on request
Bus Timeout:	0 = 50 µs 1 = 1400 µs
Address Increment:	0 = Enabled 1 = Disabled (FIFO mode)

2.13.3. Firmware Revision register

(Base Address + 0x02, D16, read only)

This register contains the firmware revision number coded on 16 bit. For example the REV. X.Y would feature:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X								Y							

Fig. 2.6: Firmware Revision Register

2.13.4. Firmware Download register

(Base Address + 0x03, D16, read/write)

This register is reserved for internal use only.

2.13.5. Flash Enable register

(Base Address + 0x04, D16, read/write)

This register is reserved for internal use only.

2.13.6. IRQ Status register

(Base Address + 0x05, D16, read only)

This register allows to monitor the IRQ lines status (1 = Active, 0 = Inactive).

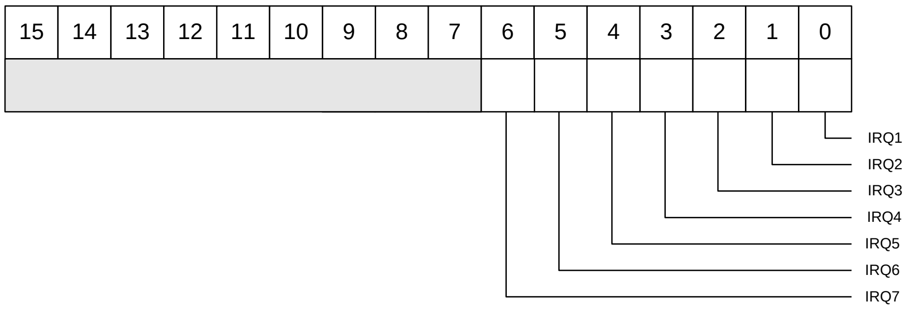


Fig. 2.7: IRQ Status register

2.13.7. IRQ Mask register

(Base Address + 0x06, D16, read/write)

This register allows to mask the IRQ lines (1 = Masked, 0 = Unmasked). If one line is masked, the interrupt on the VME bus is not transmitted to the PCI bus.

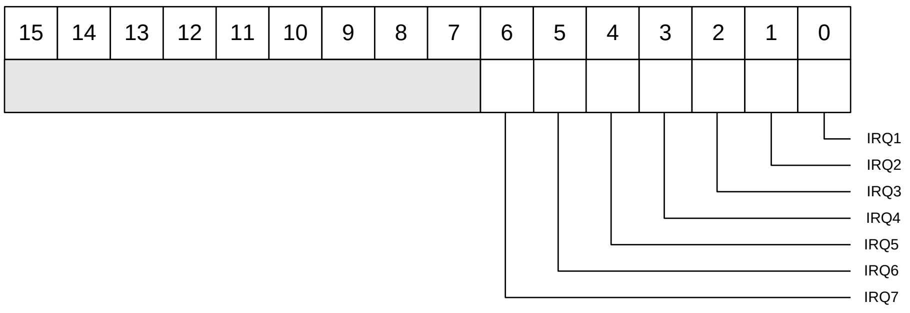


Fig. 2.8: IRQ Mask register

2.13.8. Input register

(Base Address + 0x08, D16, read/write)

This register carries the input register pattern.

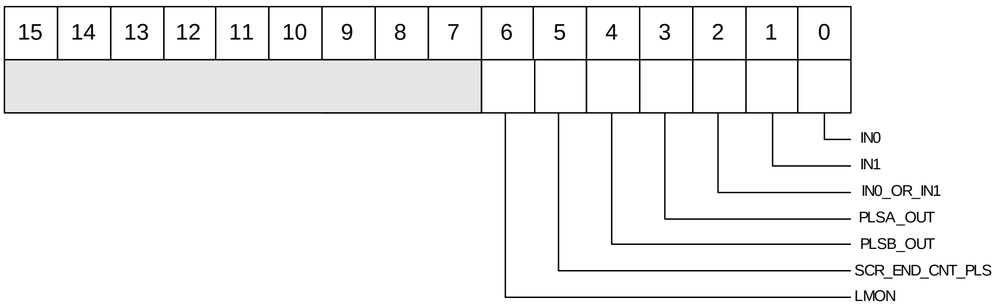


Fig. 2.9: Input register

2.13.9. Output set register

(Base Address + 0x0A, D16, read/write)

This register allows to set the output register pattern: 1 = set; 0 = leave previous setting

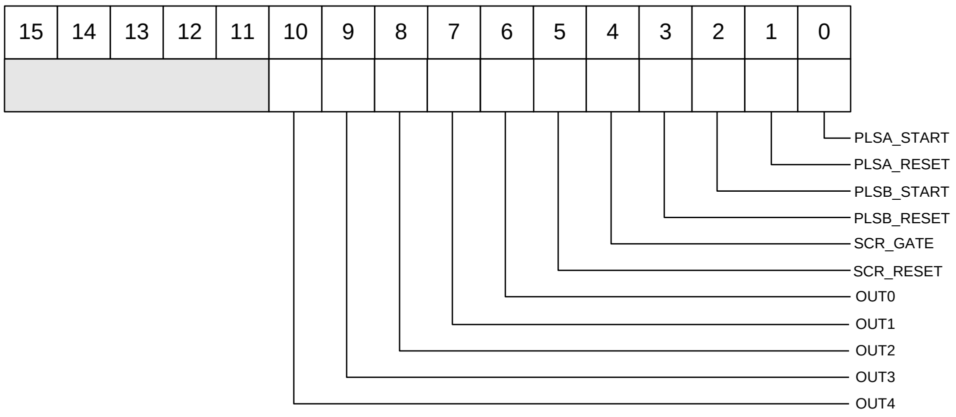


Fig. 2.10: Output set register

2.13.10. Output clear register

(Base Address + 0x10, D16, write only)

This register allows to clear the output register pattern (1 = Clear, 0 = leave previous setting).

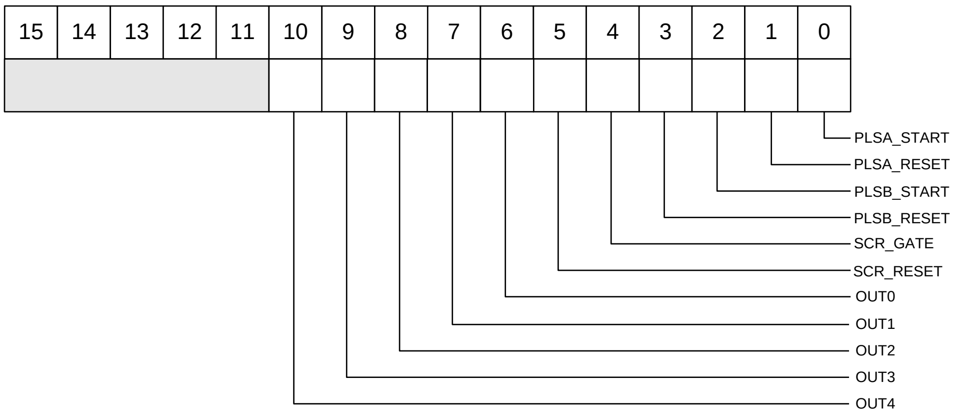


Fig. 2.11: Output set register

2.13.11. Input Multiplexer Set register

(Base Address + 0x0B, D16, read/write)

This register allows to set the IN_0 and IN_1 polarity as well as the source of Pulsers/Scaler Signals: 1 = set; 0 = leave previous setting

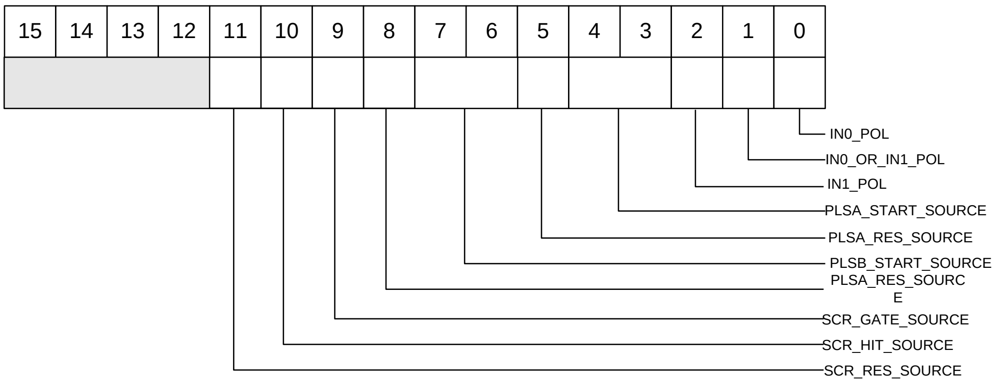


Fig. 2.12: Input Multiplexer register

INPUT POLARITY:	0 = Direct 1 = Inverted
PULSER START SOURCE:	00 = SYSRES Button (short pressure) or Software 01 = IN_0 10 = IN_1 11 = IN_0 OR IN_1
PULSER A RESET SOURCE:	0 = Output register 1 = Input 0

PULSER B RESET SOURCE:	0 = Output register 1 = Input 1
SCALER GATE SOURCE:	0 = Output register 1 = Input 1
SCALER HIT SOURCE:	0 = Output register 1 = Input 0
SCALER RESET SOURCE:	0 = Output register 1 = Input 1

2.13.12. Input Multiplexer Clear register

(Base Address + 0x11, D16, write only)

This register allows to clear the Input Multiplexer settings (1 = Clear, 0 = leave previous setting).

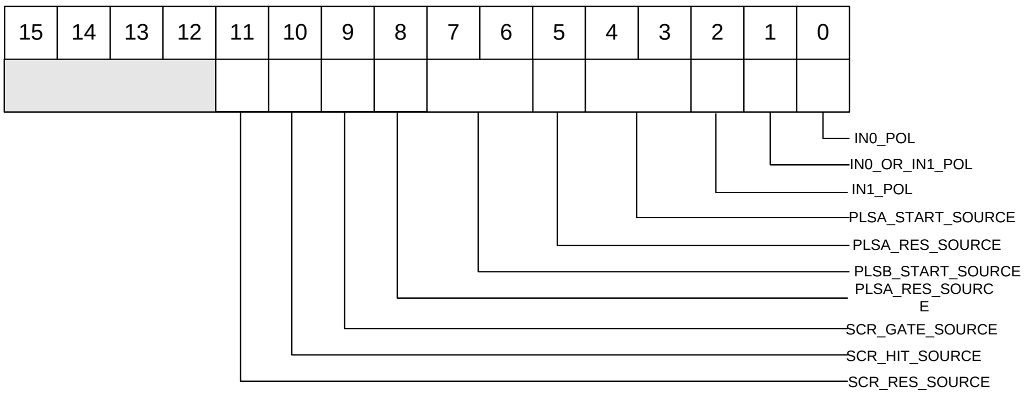


Fig. 2.13: Input Multiplexer register

2.13.13. Output Multiplexer Set register

(Base Address + 0x0C, D16, read/write)

This register allows to set the OUT[4..0] polarity as well as the source of such signals: 1 = set; 0 = leave previous setting.

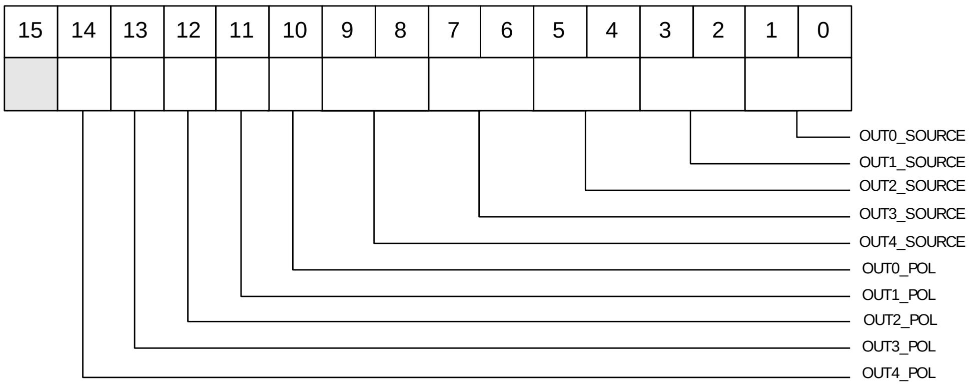


Fig. 2.14: Output Multiplexer Set register

OUTPUT_0 SOURCE:	00 = Data Strobe 01 = Input 0 AND Input 1 10 = Pulser A Output 11 = Output Register
OUTPUT_1 SOURCE:	00 = Address Strobe 01 = Input 0 AND Input 1 10 = Pulser A Output 11 = Output Register
OUTPUT_2 SOURCE:	00 = Data Acknowledge 01 = Input 0 AND Input 1 10 = Pulser B Output 11 = Output Register
OUTPUT_3 SOURCE:	00 = Bus Error 01 = Input 0 AND Input 1 10 = Pulser B Output 11 = Output Register
OUTPUT_4 SOURCE:	00 = Location Monitor 01 = Input 0 AND Input 1 10 = Scaler End Count 11 = Output Register
OUTPUT POLARITY:	0 = Direct 1 = Inverted

2.13.14. Output Multiplexer Clear register

(Base Address + 0x12, D16, write only)

This register allows to clear the Output Multiplexer settings (1 = Clear, 0 = leave previous setting)

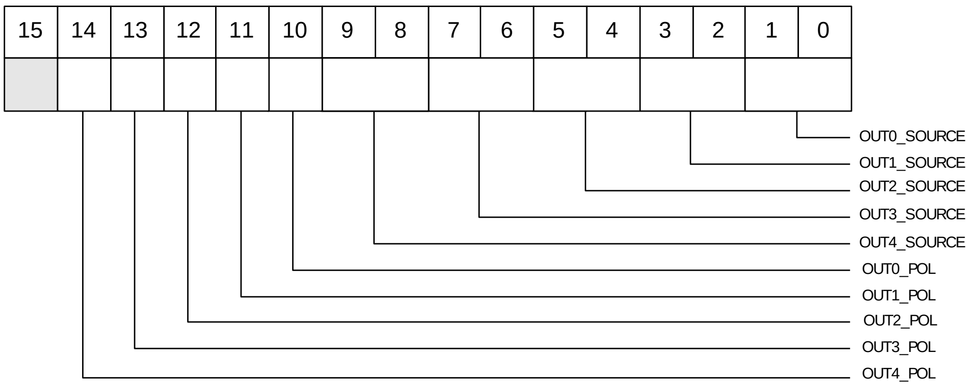


Fig. 2.15: Output Multiplexer Set register

2.13.15. LED Polarity set register

(Base Address + 0x0D, D16, read/write)

This register allows to set the LED polarity status (1 = set; 0 = leave previous setting).

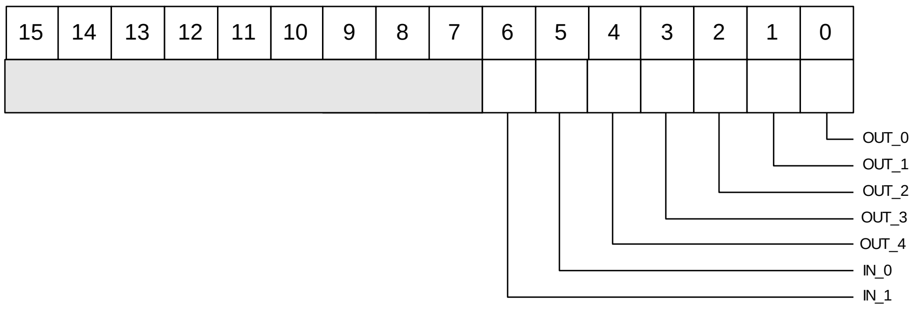


Fig. 2.16: LED Polarity set register

2.13.16. LED polarity clear register

(Base Address + 0x13, D16, write only)

This register allows to clear the LED polarity set via the LED Polarity set register (1 = Clear, 0 = leave previous setting).

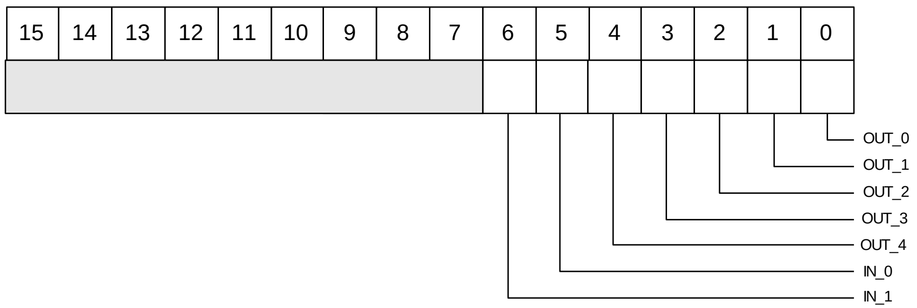


Fig. 2.17: LED polarity clear register

2.13.17. Pulser A 0 register

(Base Address + 0x16, D16, read/write)

This register allows to set the period and width of the relevant Pulser, measured in range steps (see § **2.13.18**).

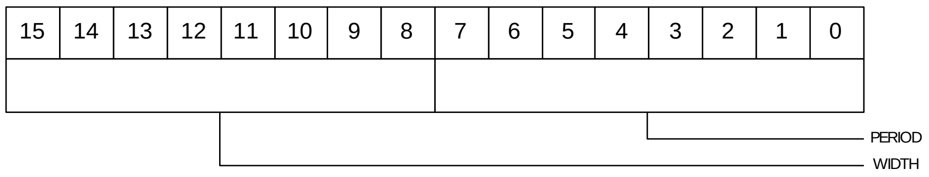


Fig. 2.18: Pulser A 0 register

2.13.18. Pulser A 1 register

(Base Address + 0x17, D17, read/write)

This register allows to set the number of pulses and the range of the relevant Pulser.

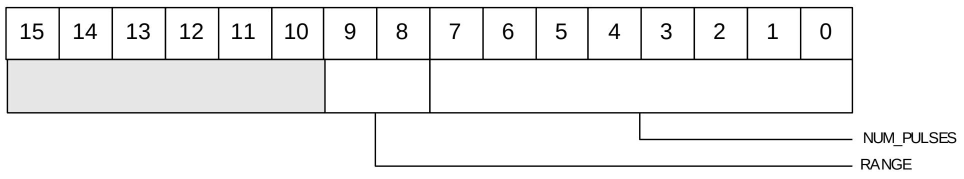


Fig. 2.19: Pulser A 1 register

RANGE:	00 → 25 ns
	01 → 1.6 µs
	10 → 400 µs
	11 → 104 ms

2.13.19. Pulser B 0 register

(Base Address + 0x19, D16, read/write)

This register allows to set the period and width of the relevant Pulser, measured in range steps (see § **2.13.20**).

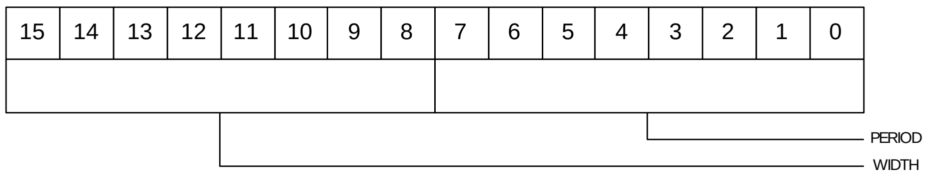


Fig. 2.20: Pulser B 0 register

2.13.20. Pulser B 1 register

(Base Address + 0x1A, D16, read/write)

This register allows to set the number of pulses and the range of the relevant Pulser.

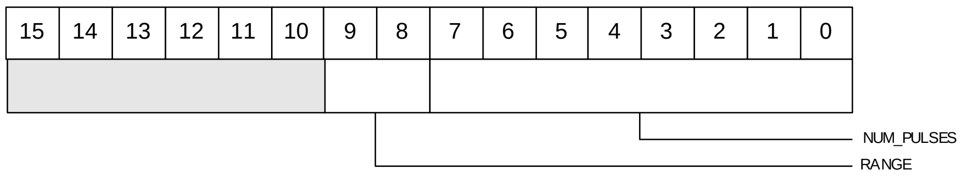


Fig. 2.21: Pulser B 1 register

RANGE:
00 → 25 ns
01 → 1.6 µs
10 → 400 µs
11 → 104 ms

2.13.21. Scaler 0 register

(Base Address + 0x1C, D16, read/write)

This register allows to set the Scaler END_COUNT_LIMIT and to enable the AUTO_RESET option (1 = enabled).

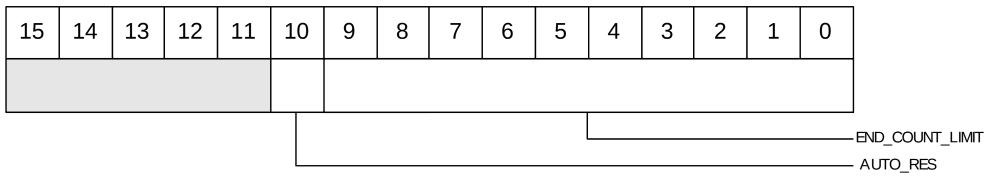


Fig. 2.22: Scaler 0 register

2.13.22. Scaler 1 register

(Base Address + 0x1D, D16, read only)

This register allows to monitor the hits accumulated by the Scaler.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
▨						HITS COUNT									

Fig. 2.23: Scaler 1 register

2.13.23. Display Address Low register

(Base Address + 0x20, D16, read only)

This register allows to monitor the LED Display Address bits[15..0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISP_AD[15:0]															

Fig. 2.24: Display Address Low register

2.13.24. Display Address High register

(Base Address + 0x21, D16, read only)

This register allows to monitor the LED Display Address bits[31..16].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISP_AD[31:16]															

Fig. 2.25: Display Address High register

2.13.25. Display Data Low register

(Base Address + 0x22, D16, read only)

This register allows to monitor the LED Display Data bits[15..0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISP_DATA[15:0]															

Fig. 2.26: Display Address Low register

2.13.26. Display Data High register

(Base Address + 0x23, D16, read only)

This register allows to monitor the LED Display Data bits[31..16].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISP_DATA[31:16]															

Fig. 2.27: Display Data High register

2.13.27. Display Control Left register

(Base Address + 0x24, D16, read only)

This register allows to monitor the LED Display Control Left bar.

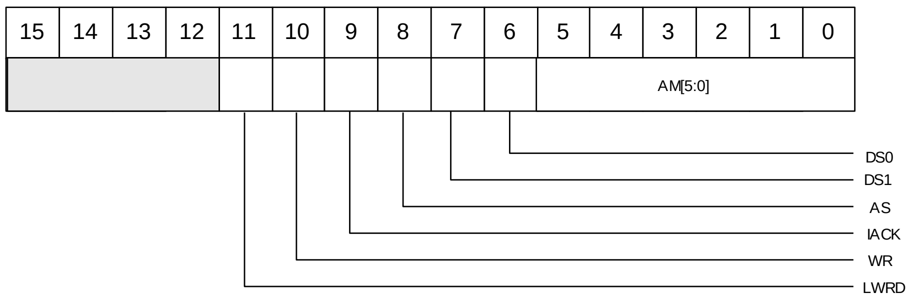


Fig. 2.28: Display Control Left register

2.13.28. Display Control Right register

(Base Address + 0x25, D16, read only)

This register allows to monitor the LED Display Control Left bar.

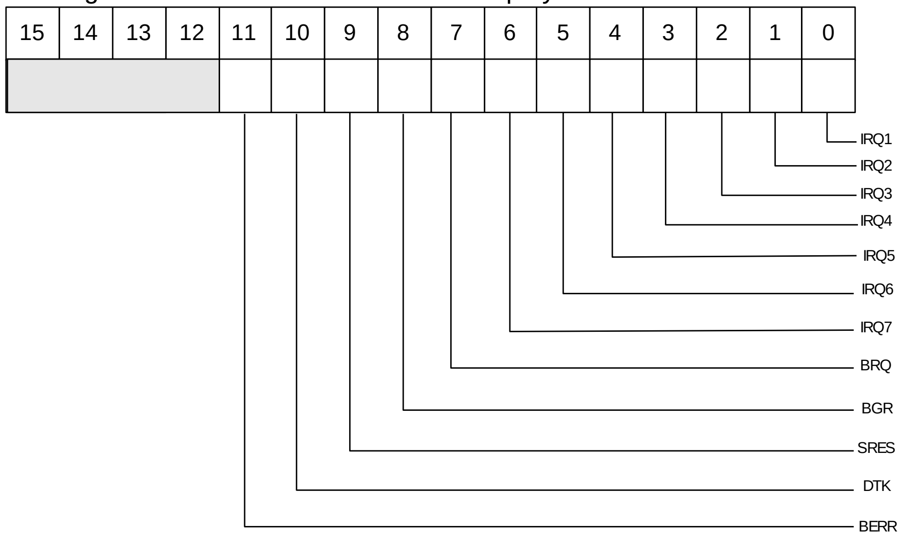


Fig. 2.29: Display Control Left register

2.13.29. Location Monitor Address Low register

(Base Address + 0x28, D16, read/write)

This register allows to set/monitor the Location monitor Address bits[15..0]; see § **2.7**.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LMON_AD [15:0]															

Fig. 2.30: Location Monitor Address Low register

2.13.30. Location Monitor Address High register

(Base Address + 0x29, D16, read/write)

This register allows to set/monitor the Location monitor Address bits[31..16]; § **2.7**.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LMON_AD [31:16]															

Fig. 2.31: Location Monitor Address Low register

2.13.31. Location Monitor Control register

(Base Address + 0x2C, D16, read/write)

This register allows to set/monitor the Location monitor control parameters; see § **2.7**

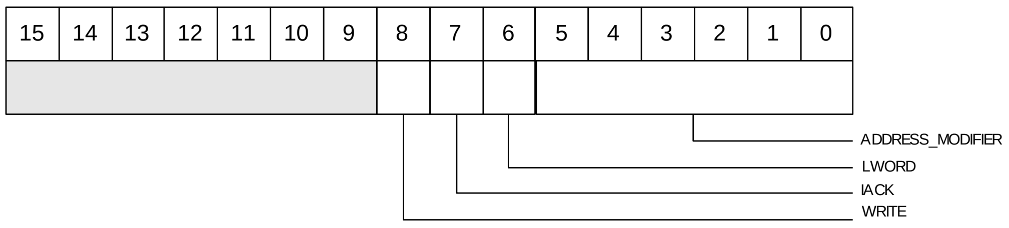


Fig. 2.32: Location Monitor control register

3. Technical specifications

3.1. Packaging

The Model V2718 is a 1-unit wide 6U high VME module.
The Mod. A2818 is a 32-bit 33 MHz PCI Bus card
The Mod. A3818 is a PCI Express (v1.1 or higher) Bus card compatible with x8 and x16 PCI Express slot.

3.2. Power requirements

Board	**V2718**		**A2818**	**A3818**
Power supplies	+5 V	1 A (running)	+3.3V/+5V (jumper selectable[4])	+12 V / +3.3 V
		0.8 A (idle)		
	-12V	150 mA (all NIM outputs active)		
		40 mA (TTL outputs or outputs not active)		
	+12V	0 A (connected but not used)		

[4] The A2818 power supply must be selected according to the used PCI bus DC power supply (+3.3V/+5V); see § **3.5.2**.

3.3. Front Panel

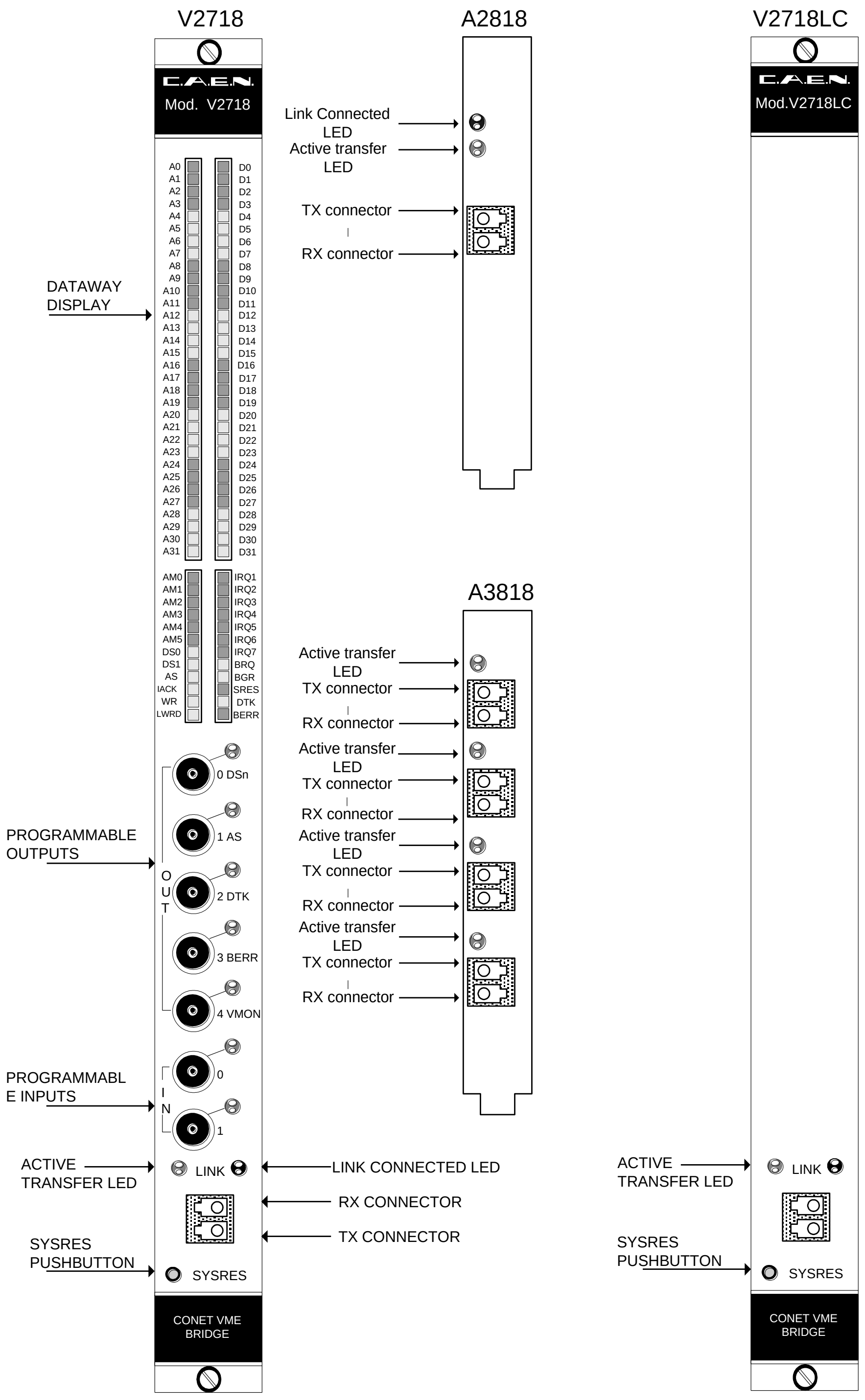


Fig. 3.1: Mod. V2718 and A2818/A3818 front panels

3.4. V2718 and A2818/A3818 External components

3.4.1. V2718 connectors

The location of the connectors is shown in Fig. 3.1. Their electromechanical specifications are listed here below.

TX/RX[5]:	Mechanical specifications: LC type connector; to be used with Multimode 62.5/125µm cable with LC connectors on both sides
PROGRAMMABLE In/Out:	Mechanical specifications: LEMO 00 connectors Electrical specifications: standard NIM/TTL signals (dip switch selectable), 50 Ω impedance

3.4.2. V2718 buttons

SYSRES pushbutton: *Long touch* (>2 s) for SYSRES generation
Short touch for Manual START of Pulsers (see § **3.6.1**)

3.4.3. A2818 connectors

The location of the connectors is shown in **Fig. 3.1**. Their electromechanical specifications are listed here below.

TX/RX:	Mechanical specifications: LC type connector; to be used with Multimode 62.5/125µm cable with LC connectors on both sides.

3.4.4. A3818 connectors

The location of the connectors is shown in **Fig. 3.1**. Their electromechanical specifications are listed here below.

TX/RX:	Mechanical specifications: LC type connector; to be used with Multimode 62.5/125µm cable with LC connectors on both sides.

[5] Two Leds indicate the Link activity: green = connection active, yellow = data transfer

3.5. V2718 Internal hardware components

In the following some hardware setting components, located on the boards, are listed. See **Fig. 3.5** for their exact location on the PCB and their settings.

3.5.1. Switches

ROTARY SWITCHES:	*Type:* 2 rotary switches. *Function:* they allow to select the VME base address of the module, when it operates in slave mode. See Fig. 2.2 for their location.
PROG_0[6]:	*Type:* DIP switch. *Function:* Forces the System Controller to be enabled, regardless the 1st Slot detection ***ON*** : SYSTEM CONTROLLER enabled ***OFF***: don't care
PROG_1:	*Type:* DIP switch. *Function:* Forces the System Controller to be disabled, regardless the 1st Slot detection ***ON***: SYSTEM CONTROLLER disabled ***OFF***: don't care
PROG_2:	*Type:* DIP switch. *Function:* When this switch is ON, the master initiates the VME cycles without waiting the Bus Grant from the arbiter; this setting must be used only for test purposes, since conflicts may occur when more VME masters are present. ***ON***: Requester bypassed ***OFF***: don't care
PROG_3:	*Type:* DIP switch. *Function:* Selects between A24 and A32 mode for the SLAVE addressing (see Fig. 2.2) ***ON***: The board responds only to A32 cycles (bits [31..24] b.a., bits [23..16] don't care) ***OFF***: The board responds only to A24 cycles (bits [31..24] b.a., bits [23..16] don't care)

[6] If PROG_0 is set to ON, then PROG_1 must be set to OFF and vice versa.

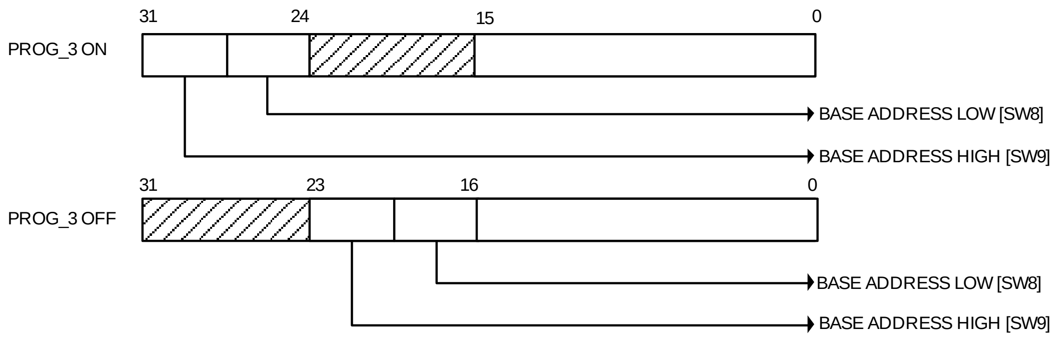


Fig. 3.2: PROG_3 Switch setting

PROG_4:	*Type:* DIP switch. *Function:* not used
I/O:	*Type:* DIP switch. *Function:* it allows the selection between NIM and TTL I/O signals ***RIGHT***: TTL ***LEFT***: NIM

3.5.2. Internal jumpers

Three jumpers (one on the V2718, one on the A2719 piggy back board and one on the A2818 PCI board) allow to select whether the “Standard” or the “Back up” firmware must be loaded at power on; jumpers’ position is shown in **Fig. 3.3**. The A2818 is supplied by the PCI bus; one jumper on the A2818 allows to select the power supply (+3.3V or +5V); if you are using a +5V PCI bus, then jumper position must be 1-2, if you are using a +3.3V PCI bus, then jumper position must be 2-3; please refer to the used PCI bus specification in order to select the proper power supply.

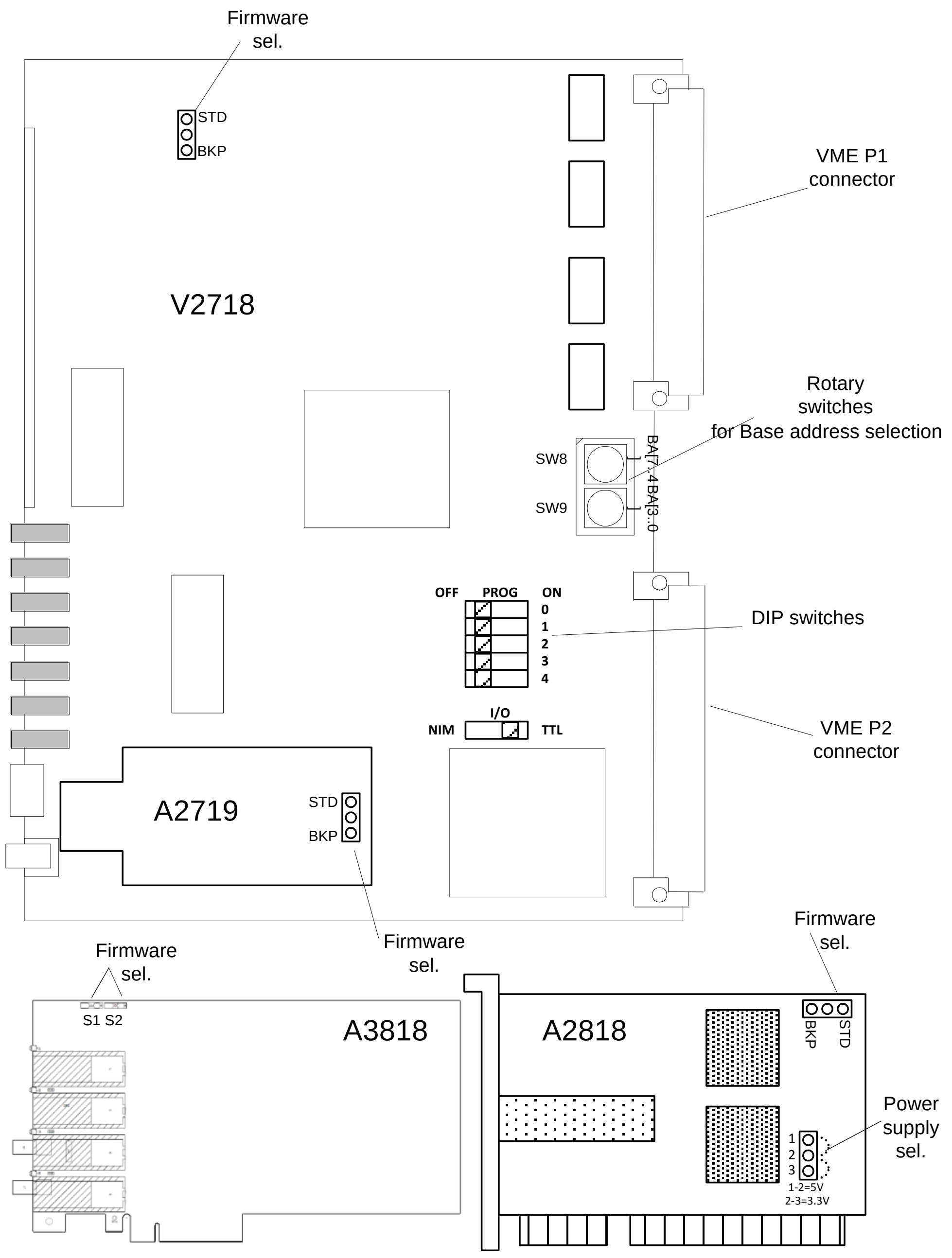


Fig. 3.3: Component Location

In case of A3818, an on-board FLASH stores 4 firmware images in 4 different pages. By operating the two switches S1 and S2 on the board (see **Fig. 3.3**), the user can manually select the copy of the firmware to be loaded in the FPGA at power-on:

S2 = A, S1 = A -> firmware on page 0
S2 = A, S1 = B -> firmware on page 1
S2 = B, S1 = A -> firmware on page 2
S2 = B, S1 = B -> firmware on page 3

3.6. Programmable Input/Output

The V2718 front panel houses 7 LEMO 00 type connectors, 5 outputs and 2 inputs; signals can be either NIM or TTL (dip-switch selectable). Seven green LEDs (one per connector) light up as the relevant signal is active. All the signals can perform several functions, default setting of the output signals is:
DS (either DS0 or DS1)
AS
DTACK
BERR
LMON (output of Location Monitor)

All the siganls, whose detailed description is reported in § **2**, may be connected to other logic functions; the available functions are listed in the following table:

Table 3.1: FPGA available functions

	Availability	*Input*	*Output*	*Register*
Timer & Pulse Generator	2	2	1	3
Scaler	1	3	1	2
Coincidence	1	2	1	0
Input Register	1	2	-	1
Output Register	1	-	5	1
Location Monitor	1	VME bus	1	-

3.6.1. Timer & Pulse Generator

It is an unit which produces a burst of N pulses (N can be infinite, i.e. the pulses are countinuously generated), whose period T and duration W are programmable (see § **2.13.17**, § **2.13.18**, § **2.13.19** and § **2.13.20**). The burst START can be sent either as input signal (on one LEMO input connector) or as manual/software command. A RESET can interrupt the sequence and set to zero the outputs. These modules can be used, for example, as:

– Clock Generator
– Burst Generator
– Monostable
– Gate and Delay Generator
– Set-Reset Flip-Flop

3.6.2. Scaler

It is a counter with the GATE input for enabling the counter and the counter RESET input. The counter has the programmable END_COUNT_LIMIT parameter; LIMIT can be set in the 0 ÷ 1023 range; if LIMIT = 0, the scaler counts countinuously and produces an END_CNT_PULSE every 1024 hits (each time ZERO is met); the scaler can be halt via the RESET input. If END_COUNT_LIMIT = N (N ≠ 0), the scaler counts up to N hits, then produces END_CNT_PULSE; if AUTORES is enabled, the scaler, after N hits, returns to zero and can accept new hits to count, otherwise it halts.

3.6.3. Coincidence

It is a two input OR port. Since each input and output can be negated, it can operate also as AND. The Coincidence output can be connected either to other units input or to an output connector.

3.6.4. Input/Output Register

The output signals can be programmed via an Output Register, while the input signals can be monitored via an Input Register.

3.7. I/O internal connections

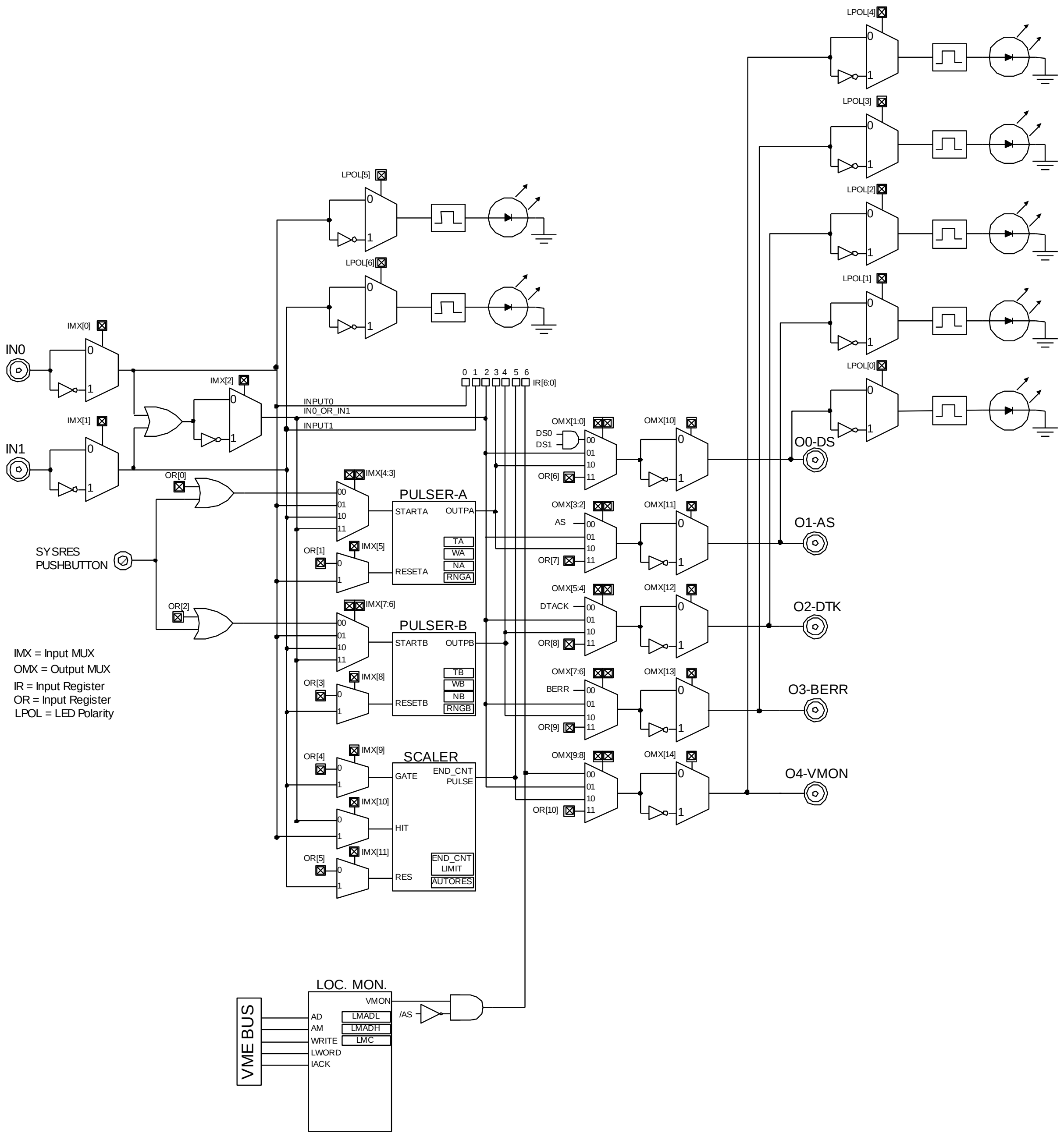


Fig. 3.4: Input/Output connections scheme

3.8. VME Dataway Display

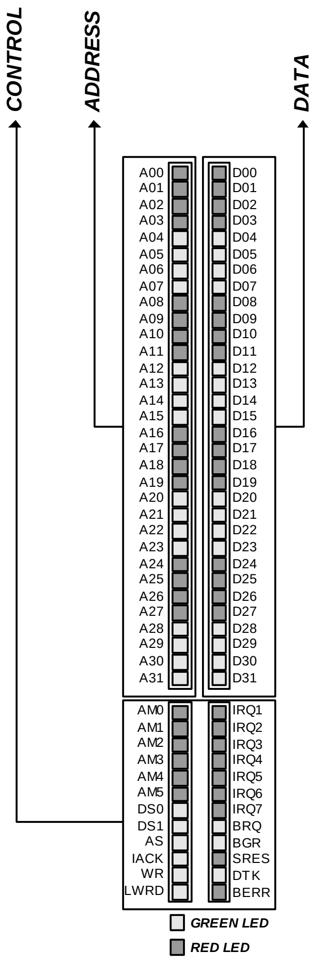


Fig. 3.5: Dataway Display layout

The V2718 is provided with a 88 LED Dataway Display; such LEDs report the VME Bus status (address, data and control lines) related to the latest cycle.

ADDR[31:0], **AM[5:0]**, **IACK**, **WRITE** and **LWORD**: These LEDs are frozen on the AS leading edge and remain stable until the next cycle.

DATA[31:0]: These LEDs are frozen either on the DS leading edge during the write cycles, or on the DTACK (or BERR) leading edge during the read cycles. The datum remains stable until the next cycle. In case of BLT cycles, the last read datum remains visible.

DS0 and **DS1**: These LEDs turn on as the signal is active during the cycle just executed; they remain stable until the next cycle.

AS: This LED flashes on the AS leading edge; it is used for signalling a cycle execution.

BGR: This LED flashes as any Bus Grant line (BG[3:0]) is active.

BRQ: This LED flashes as any Bus Request line (BR[3:0]) is active.

SRES: This LED flashes as the SYSRES is active.

DTK: This LED turns on if the cycle just executed was terminated with a DTACK asserted by a slave; it remains on until the next cycle.

BERR: This LED turns on if the cycle just executed was terminated with a BERR; it remains on until the next cycle.

The LEDs status can be monitored also via the relevant registers (0x20 through 0x25), when the module operates as slave; in this case the VME cycle executed for the LED display readout does not cause the display update: the display shows the status related to the previous cycle.

3.9. Firmware upgrade

The V2718, its A2719 mezzanine board and the A2818 PCI board can store two firmware versions each, called STD and BKP respectively; at Power On, a microcontroller reads the Flash Memory and programs the modules with the firmware version selected via the relevant jumper (see **Fig. 3.3**), which can be placed either on the STD position, or in the BKP position.
A3818 controller, as introduced in § **3.5.2**, can store 4 firmware versions manually selectable through the two on-board dedicated switches.

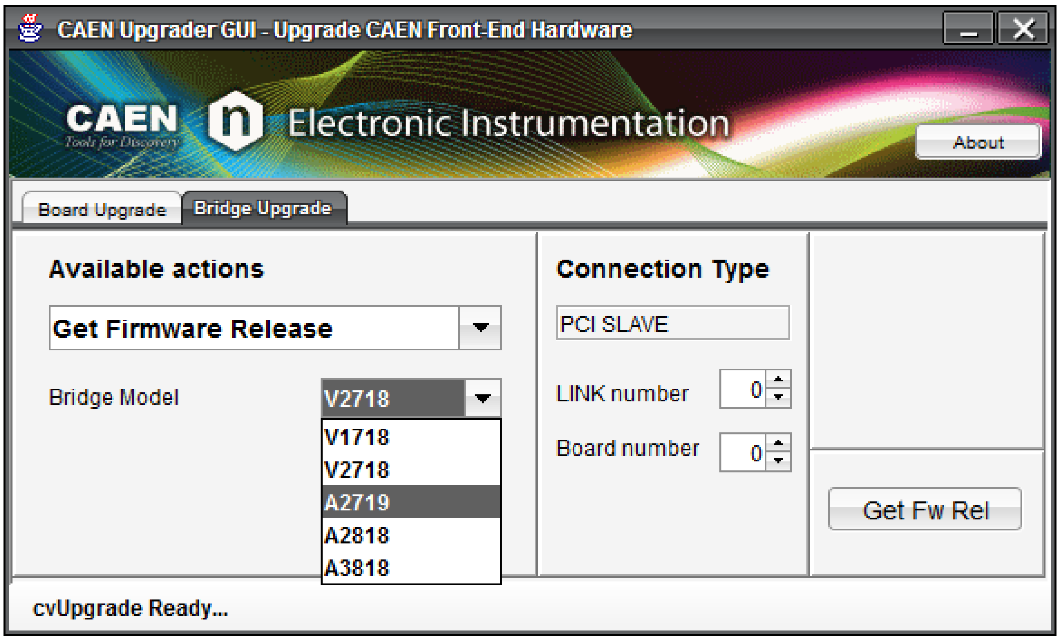
Firmware upgrade files for V2718 are available for free download from CAEN website at:
Home / Products / Modular Pulse Processing Electronics / VME / Controller (VME) / V2718
The zipped package includes the new firmware release files:

V1718VUB_RevXY.rbf	for the V2718
A2719CI_RevXY.rbf	for the A2719

Firmware upgrade files for A2818 and A3818 controllers are available for free download from CAEN website at:
Home / Products / Modular Pulse Processing Electronics / PCI/PCIe / Optical Controllers / <CONTROLLER>

CAEN provides the CAENupgrader software tool which makes possible to read the release of the currently running copy of the firmware and upgrade the board firmware via optical link, by writing the Flash. The software installation package, compliant to Windows and Linux platforms, and the relevant documentation can be free downloaded from CAEN website at:
Home / Products / Firmware/Software / Digitizer Software / Configuration Tools / CAENUpgrader

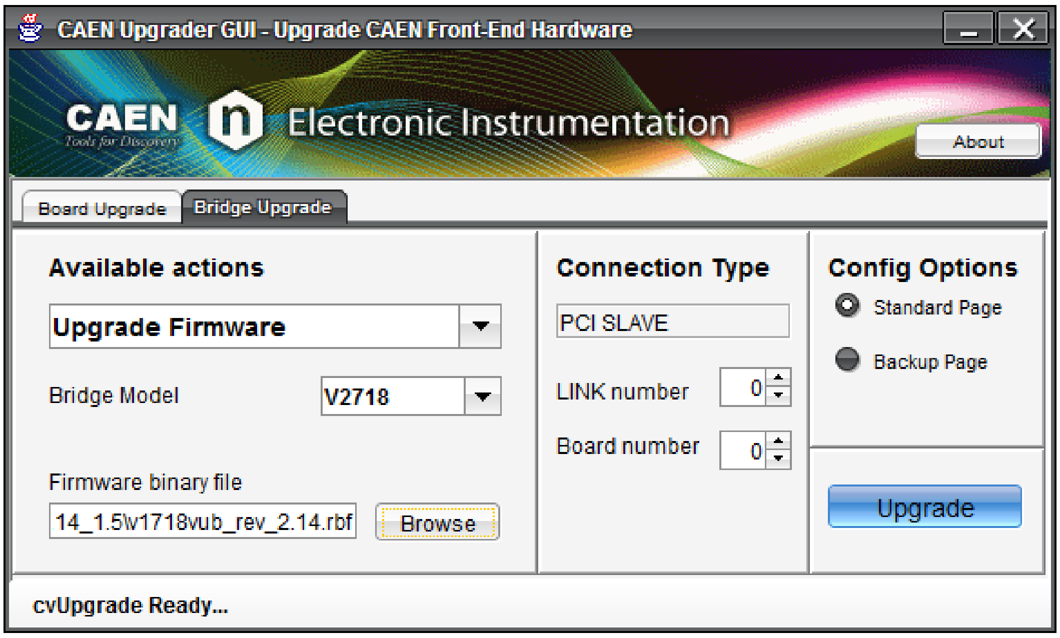
In order to read the firmware of the boards, the user must select in the "Bridge Upgrade" tab the "Get FW Release" option, set the "Bridge Model" and configure the connection parameters.


CAEN Upgrader GUI - Upgrade CAEN Front-End Hardware
CAEN Tools for Discovery Electronic Instrumentation
About
Board Upgrade
Bridge Upgrade
Get Fw Rel

Available actions: Get Firmware Release
Bridge Model: V2718 (V1718, V2718, A2719, A2818, A3818)
Connection Type: PCI SLAVE
LINK number: 0
Board number: 0
cvUpgrade Ready...

Fig. 3.6: CAENUpgrader's "Get FW Release" view

In order to upgrade the firmware of the boards, the user must select in the “Bridge Upgrade” tab the “Upgrade Firmware” option, set the “Bridge Model”, configure the connection parameters and point to the firmware file to be uploaded. The tab allows also to select which FLASH page to be written.


CAEN Upgrader GUI - Upgrade CAEN Front-End Hardware

Board Upgrade | Bridge Upgrade

Available actions: Upgrade Firmware
Bridge Model: V2718
Firmware binary file: 14_1.5\v1718vub_rev_2.14.rbf

Connection Type: PCI SLAVE
LINK number: 0
Board number: 0

Config Options: [✓ Standard Page] Backup Page

cvUpgrade Ready...

Fig. 3.7: CAENUpgrader’s “Upgrade Firmware” view

The VME “Board number” parameter refers to the position of the V2718 in the CONET network and ranges from 0 to 7, as shown the following picture:

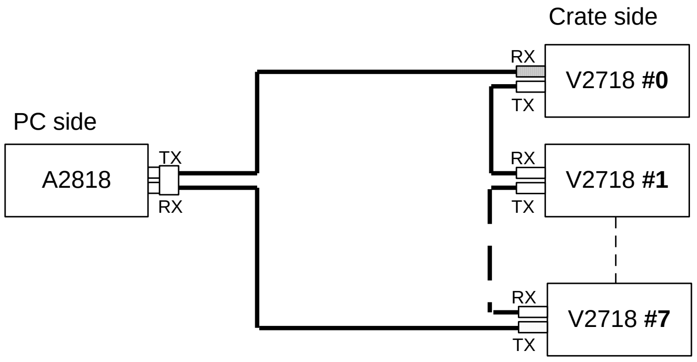


Fig. 3.8: VME Index in the CONET network

the PCI/PCIe “Link number” parameters indicates which link of A2818 or A3818 is used; the index starts from 0 (1st Optical link port in the 1st slot used). It is not known a priori which is the first slot used (it depends on the motherboard of the PC used.).
IMPORTANT NOTE: if A2818 and A3818 are installed together, the A2818 have lower index assigned.

It is suggested to upgrade the three devices in the following order: A2818 (or A3818), A2719, V2718.

THE CONET2 PROTOCOL AND THE PREVIOUS CONET1 PROTOCOL ARE NOT COMPATIBLE. PLEASE, REFER THE APPLICATION NOTE "AN2472 CONET1 TO CONET2 MIGRATION" (CAEN WEBSITE AT: HOME / DOCUMENT LIBRARY) FOR INSTRUCTIONS ON HOW TO UPGRADE YOUR SYSTEM FROM CONET1 TO CONET2.

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 AGAIN AND THE BOARD MUST BE SENT TO CAEN IN REPAIR!

In case of failures while programming the STD page of the FLASH on the A2719 mezzanine, compromising the communication with the V2718, the user can perform the following recovering procedure as first attempt:

- Power off the crate and extract the V2718

- Set the dedicated jumper on the A2719 to the BKP position

- Insert the V2718 in the crate and power on

- Use CAENUpgrader to read the A2719 firmware revision (in this case the one of the BKP copy). If this succeeds, it is possible to communicate again with the board

- Use CAENUpgarder to upload a compliant firmware on the STD page of the A2719's FLASH

- Power off the crate and extract the V2718

- Set the dedicated jumper on the A2719 back to the STD position

- Insert the V2718 in the crate and power on

- Use CAENUpgrader to read the firmware revision (in this case the one of the STD copy) so that the board is operative again

In case the procedure above also fails, the board needs to be sent back to CAEN in repair (see § **5** for contacts).

At Power On (or after pushing the SYSRES button for 2 s at least), the selected firmware revision is also shown by the A00..A15 front panel leds.

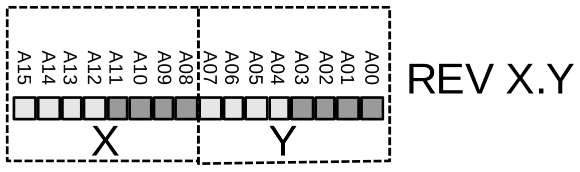


Fig. 3.9: Firmware revision on the Dataway Display

3.10. V2718 technical specifications table

Table 3.2: Mod. V2718 technical specifications

Packaging	1-unit wide and 6U high VME module
PC Interface	Optical Link from PCI / PCIe Bus
Transfer rate[7]	~ 70 MByte/s using the CONET1 old optical link protocol ~ 80 MByte/s using the CONET2 new optical link protocol
Addressing	A16, A24, A32, CR/CSR, LCK; ADO, ADOH cycles

Data cycles	D08, D16, D32 for R/W and RMW;
	D16, D32 for BLT;
	D64 for MBLT

Interrupt cycles	D08, D16, D32, IACK cycles (IRQ[7:1] D08, D16, D32, IACK cycles (IRQ[7:1] passed from VME to the PCI/PCIe BUS through optical link)
LED display	Data bus, address bus, address modifier, interrupt request, control signals
Panel outputs	5 NIM/TTL programmable (default: DSn, AS, DTACK, BERR, LMON)
Panel inputs	2 NIM/TTL programmable

[7] Transfer rate supported in MBLT read cycles (block size = 32 kByte), using a PC host with Windows XP or Linux.

4. Software overview

4.1. Software User Interface

An user friendly interface has been developed for the module's control, the following sub sections will show the features of the software, which is, anyway, mostly self explanatory.

4.1.1. Software User Interface: Installation

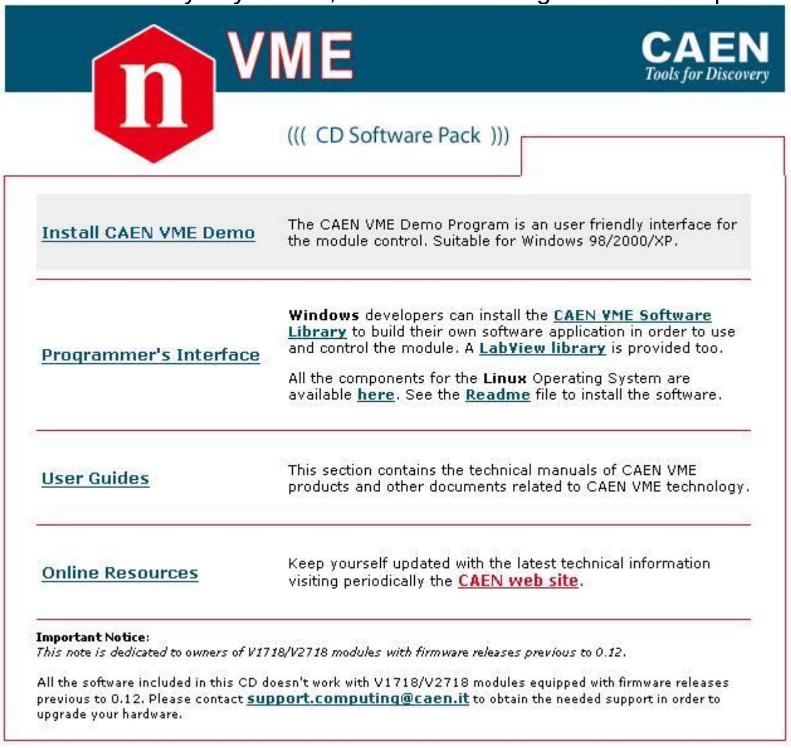
The following instructions will help through the module installation; the package includes:

– V2718 VME Board
– A2818 PCI Board
– Software & Documentation Pack CD
– User Manual

Before you begin, be sure that:

– the V2718 is not connected to your computer;
– the V2718 supports your operating system.

Place the CD in the CD tray in your PC, then the following window will open:



Install CAEN VME Demo — The CAEN VME Demo Program is an user friendly interface for the module control. Suitable for Windows 98/2000/XP.

Programmer's Interface — **Windows** developers can install the CAEN VME Software Library to build their own software application in order to use and control the module. A LabView library is provided too.

All the components for the **Linux** Operating System are available here. See the Readme file to install the software.

User Guides — This section contains the technical manuals of CAEN VME products and other documents related to CAEN VME technology.

Online Resources — Keep yourself updated with the latest technical information visiting periodically the CAEN web site.

Important Notice:
This note is dedicated to owners of V1718/V2718 modules with firmware releases previous to 0.12.

All the software included in this CD doesn't work with V1718/V2718 modules equipped with firmware releases previous to 0.12. Please contact support.computing@caen.it to obtain the needed support in order to upgrade your hardware.

Fig. 4.1: The Software & Documentation Pack CD introduction

– Click on "Install CAEN VME Demo" in order to install the provided user friendly interface which allows an easy and immediate control of the module (see § **4.1.3**)

– Click on “Programmer’s Interface” in order to install the provided Software Library which allows experienced developers to build their own applications for the module control (see § **4.2**); a C example program file is installed too.

4.1.2. Hardware Installation

1. The A2818 is a plug-and-play PCI card and must be plugged into one PCI slot (either 5 V or 3.3 V supplied, see § **3.5.2**) of the PC motherboard.
2. Connect the TX connector of the A2818 to the RX connector of the first V2718 of the CONET network, via the optical fiber cable.
3. If you are using I-type cables (see § **1.3**), then connect the TX connector of the first V2718 of the CONET network to the RX connector of the second V2718 (if existing) and so on, until the last module in the chain, whose TX connector must be connected to the A2818 RX connector ; if only one V2718 is present, then its TX connector must be connected to the RX connector of the A2818.
   If you have only one V2718 in your network and you are using X-type cables (see § **1.3**), then simply plug it into the TX/RX connectors of the A2818 and V2718.
4. Now the network is ready for operation.

4.1.3. CAENVME Demo: The Main Menu[8]

The Main Menu allows to perform and monitor the supported Data and IRQ cycles.

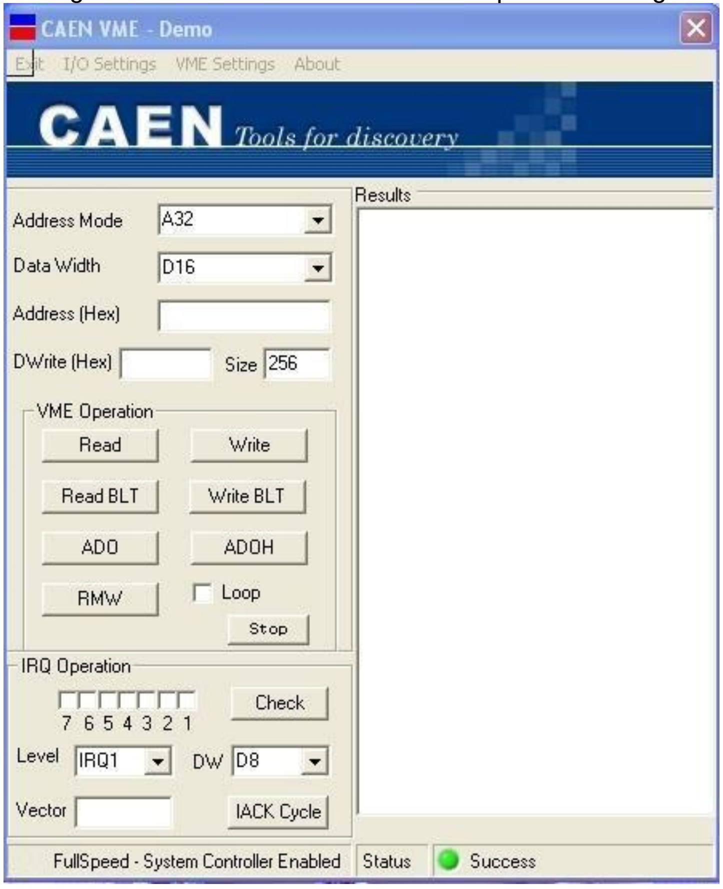
Data cycles:

Once the address mode and the data width are selected, the User has to write the address where the cycle must be performed and the eventual datum to be written; then the VME Operation buttons allows to select the desired cycle. The operation results are shown in the relevant field.

The status bar at the window's bottom allows to detect eventual errors on the bus.

IRQ cycles:

Seven boxes allow to detect an input request on the bus, by clicking on the "Check" button; the remaining fields allow to broadcast an interrupt acknowledge CYCLE.



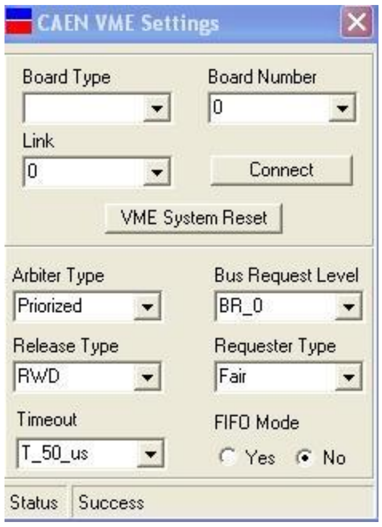
Address Mode: A32
Data Width: D16
Address (Hex):
DWrite (Hex): Size: 256
Level: IRQ1 DW: D8
Vector:
FullSpeed - System Controller Enabled | Status | Success

Fig. 4.2: The Main Menu

[8] Only for Windows 98/2000/XP

4.1.4. Software User Interface: I/O Setting Menu – VME Settings

The VME Settings Menu allows to perform the VME general settings of the V2718; the VME Settings are explained in detail in § **2**. *Board type* must be set to V2718, *Link* is the PCI slot of the used A2818 (0 to 4) and *Board number* is the V2718 position in the daisy chain.

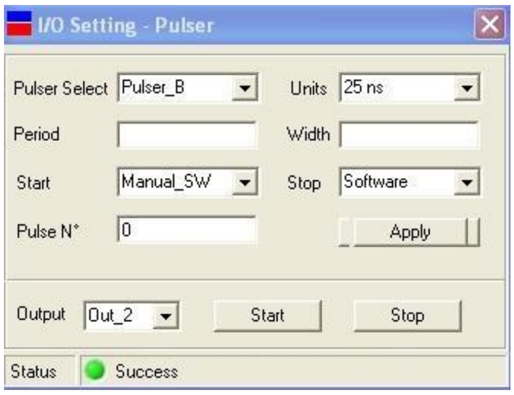

CAEN VME Settings

Board Type:
Board Number: 0
Link: 0
Arbiter Type: Priorized
Bus Request Level: BR_0
Release Type: RWD
Requester Type: Fair
Timeout: T_50_us
FIFO Mode: Yes [✓ No]
Status: Success

Fig. 4.3: The I/O Setting Menu – VME Settings

4.1.5. Software User Interface: I/O Setting Menu – Pulser

The Pulser Setting Menu allows to perform the settings of the V2718 built in pulsers (see § **3.7**). The V2718 features two internal pulsers (Pulser A and Pulser B); the output pulses are provided in the following way: Out_0 or Out_1 for Pulser A, Out_2 or Out_3 for Pulser B. The programmable parameters are the step units, the period, width and number of produced pulses. Start can be sent via software, via the SYSRES button (short pressure) or via the Input_0/Input_1 signals. Stop can be sent either via software or via the Input_0 (Pulser A) and Input_1 (Pulser B). The pulsers can be reset via the front panel SYSRES button (long pressure). Refer also to § **2.13.11**.

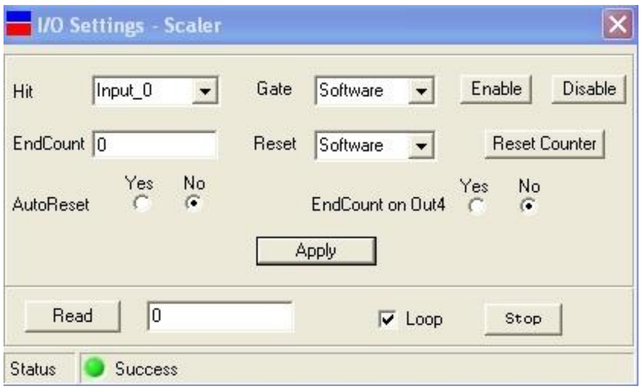

I/O Setting - Pulser

Pulser Select: Pulser_B
Units: 25 ns
Period:
Width:
Start: Manual_SW
Stop: Software
Pulse N°: 0
Output: Out_2
Status: ● Success

Fig. 4.4: The I/O Setting Menu – Pulser

4.1.6. Software User Interface: I/O Setting Menu – Scaler

The Scaler Setting Menu allows to perform the settings of the V2718 built in scaler (see § **3.7**). The V2718 features an internal scaler, which counts hits arriving on the enabled front panel input (Input_0 or Input_1). Gate and Reset signals can be sent either on the unused input connector or software generated; an End_Count_Pulse is eventually available on Out_4. The End_Count field allows to set the number of hits to be stored (End_Count_Limit); Auto Reset and Loop options can be either enabled or disabled independently. The lowest field allows to read the stored hits. Refer also to § **2.13.21**.

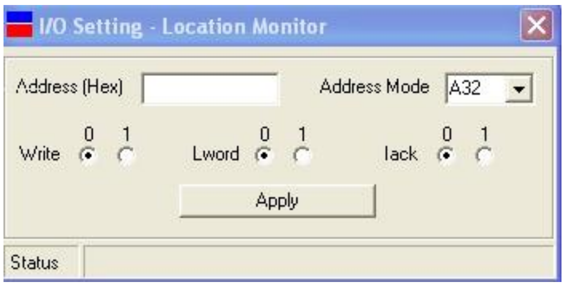

I/O Settings - Scaler

Hit: Input_0
Gate: Software
EndCount: 0
Reset: Software
AutoReset: Yes [✓ No]
EndCount on Out4: Yes [✓ No]
Read: 0
[✓ Loop]
Status: Success

Fig. 4.5: The I/O Setting Menu – Scaler

4.1.7. Software User Interface: I/O Setting Menu – Location Monitor

The Location Monitor Setting Menu allows to produce an output signal when a particular VME cycle, at a particular base address, is detected; see § **2.7** for details.

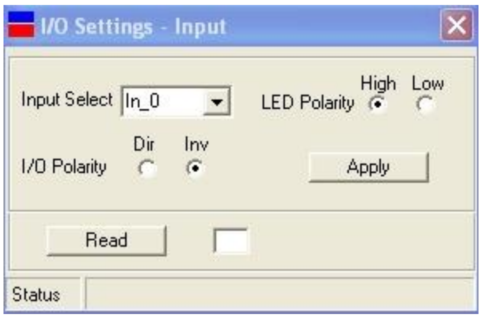

I/O Setting - Location Monitor

Address (Hex):
Address Mode: A32
Write: [✓ 0] 1
Lword: [✓ 0] 1
Iack: [✓ 0] 1
Status:

Fig. 4.6: The I/O Setting Menu – Location Monitor

4.1.8. Software User Interface: I/O Setting Menu – Input

The Input Setting Menu allows to set the polarity of Input_0, Input_1 and of the relevant LEDs see also § **2.13.11** and § **2.13.15**.


I/O Settings - Input

Input Select: In_0

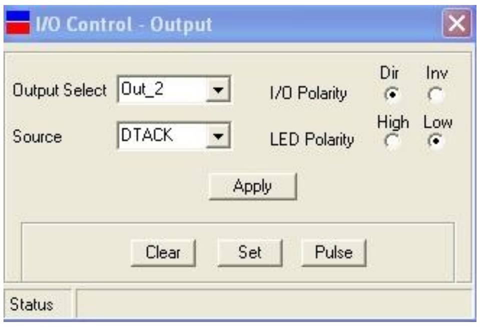
LED Polarity: [✓ High] Low

I/O Polarity: Dir [✓ Inv]

Fig. 4.7: The I/O Setting Menu – Input

4.1.9. Software User Interface: I/O Setting Menu – Output

The Output Setting Menu allows to set the polarity of Output [0;4] and of the relevant LEDs, as well as to select the output source and to produce an output pulse at will, see also § **2.13.13**.


I/O Control - Output

Output Select: Out_2

Source: DTACK

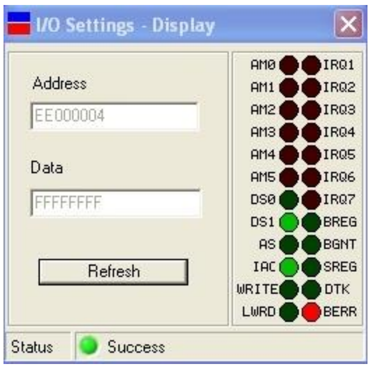
I/O Polarity: [✓ Dir] Inv

LED Polarity: High [✓ Low]

Fig. 4.8: The I/O Setting Menu – Input

4.1.10. Software User Interface: I/O Setting Menu – Display

The Display Setting Menu allows actually to monitor the status of the Display corresponding to a serviced cycle, see also § **2.13.23** through § **2.13.28**.


I/O Settings - Display

Address: EE000004

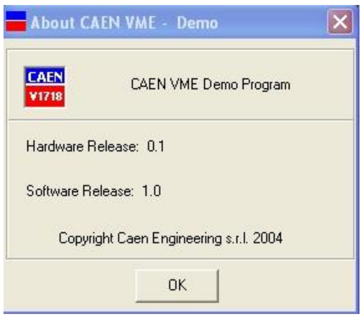
Data: FFFFFFFF

Status: Success

Fig. 4.9: The I/O Setting Menu – Display

4.1.11. Software User Interface: I/O Setting Menu – About

The About Setting Menu allows to detect the revision number of the running software and firmware.


About CAEN VME - Demo

CAEN VME Demo Program

Hardware Release: 0.1

Software Release: 1.0

Copyright Caen Engineering s.r.l. 2004

OK

Fig. 4.10: The I/O Setting Menu – Display

4.2. CAENVMELib introduction

This section describes the CAENVMELib library and its implemented functions. CAENVMELib is a set of ANSI C functions which permits an user program the use and the configuration of the V2718.
The present description refers to CAENVMELib Rel. 1.x, 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.

4.3. CAENVMELib 1.x description

4.3.1. CAENVME_SWRelease

Parameters:
[out] SwRel: Returns the software release of the library.

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

Description:
Permits to read the software release of the library.

CAENVME_API
CAENVME_SWRelease(char *SwRel);

4.3.2. CAENVME_Init

Parameters:

[in] BdType	: The model of the bridge (V2718).
[in] Link	: The index of the A2818.
[in] BdNum	: The board number in the link.
[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 a 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);
```

4.3.3. CAENVME_BoardFWRelease

Parameters:

[in] Handle	: The handle that identifies the device.
[out] FWRel	: Returns the firmware release of the device.

Returns:

An error code about the execution of the function.

Description:

Permits to read the firmware release loaded into the device.

```
CAENVME_API
CAENVME_BoardFWRelease(long Handle, char *FWRel);
```

4.3.4. CAENVME_End

Parameters:

[in] Handle: The handle that identifies the device.

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);
```

4.3.5. CAENVME_ReadCycle

Parameters:

[in] Handle	: The handle that identifies the device.
[in] Address	: The VME bus address.
[out] Data	: The data read from the VME bus.
[in] AM	: The address modifier (see CVAddressModifier enum).
[in] DW	: The data width.(see CVDataWidth enum).

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, CVAddressModifier AM, CVDataWidth DW);

4.3.6. 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 (see CVAddressModifier enum).
[in] DW	: An array of data widths.(see CVDataWidth enum).

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);

4.3.7. CAENVME_RMWCycle

Parameters:

[in]	Handle: The handle that identifies the device.
[in]	Address: The VME bus address.
[in/out]	Data: The data read and then written to the VME bus.
[in]	AM: The address modifier (see CVAddressModifier enum).
[in]	DW: The data width.(see CVDataWidth enum).

Returns:

An error code about the execution of the function.

Description:

The function performs a Read-Modify-Write cycle. The Data parameter is bidirectional: it is used to write the value to the VME bus and to return the value read.

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

4.3.8. 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 (see CVAddressModifier enum).
[in]	DW	: The data width.(see CVDataWidth enum).

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).

4.3.9. 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 (see CVAddressModifier enum).
[in] DW	: An array of data widths.(see CVDataWidth enum).

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);

4.3.10. 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 (see CVAddressModifier enum).
[in] DW	: The data width.(see CVDataWidth enum).
[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);

4.3.11. 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 (see CVAddressModifier enum).
[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);

4.3.12. CAENVME_BLTWriteCycle

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Address	: The VME bus address.
[in]	Buffer	: The data to be written to the VME bus.
[in]	Size	: The size of the transfer in bytes.
[in]	AM	: The address modifier (see CVAddressModifier enum).
[in]	DW	: The data width (see CVDataWidth enum).
[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 write cycle.

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

4.3.13. CAENVME_MBLTWriteCycle

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Address	: The VME bus address.
[in]	Buffer	: The data to be written to the VME bus.
[in]	Size	: The size of the transfer in bytes.
[in]	AM	: The address modifier (see CVAddressModifier enum).
[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 write cycle.

```
CAENVME_API
CAENVME_MBLTWriteCycle(long Handle, unsigned long Address, unsigned char
*Buffer, int size, CVAddressModifier AM, int *count);
```

4.3.14. CAENVME_ADOCycle

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Address	: The VME bus address.
[in]	AM	: The address modifier (see CVAddressModifier enum).

Returns:

An error code about the execution of the function.

Description:

The function performs a VME address only.

```
CAENVME_API
CAENVME_ADOCycle(long Handle, unsigned long Address, CVAddressModifier AM);
```

4.3.15. CAENVME_ADOHCycle

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Address	: The VME bus address.
[in]	AM	: The address modifier (see CVAddressModifier enum).

Returns:

An error code about the execution of the function.

Description:

The function performs a VME address only with handshake cycle.

```
CAENVME_API
CAENVME_ADOHCycle(long Handle, unsigned long Address, CVAddressModifier AM);
```

4.3.16. CAENVME_SetPulserConf

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	PulSel	: The pulser to configure (see CVPulserSelect enum).
[in]	Period	: The period of the pulse in time units.
[in]	Width	: The width of the pulse in time units.
[in]	Unit	: The time unit for the pulser configuration (see CVTimeUnits enum).
[in]	PulseNo	: The number of pulses to generate (0 = infinite).
[in]	Start	: The source signal to start the pulse burst (see CVIOSources enum).
[in]	Reset	: The source signal to stop the pulse burst (see CVIOSources enum).

Returns:

An error code about the execution of the function.

Description:

The function permits to configure the pulsers. All the timing parameters are expressed in the time units specified. The start signal source can be one of: front panel button or software (cvManualSW), input signal 0 (cvInputSrc0),input signal 1 (cvInputSrc1) or input coincidence (cvCoincidence). The reset signal source can be: front panel button or software (cvManualSW) or, for pulser A the input signal 0 (cvInputSrc0), for pulser B the input signal 1 (cvInputSrc1).

```
CAENVME_API
CAENVME_SetPulserConf(long Handle, CVPulserSelect PulSel, unsigned char Period,
unsigned char Width, CVTimeUnits Unit, unsigned char PulseNo, CVIOSources Start,
CVIOSources Reset);
```

4.3.17. CAENVME_SetScalerConf

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Limit	: The counter limit for the scaler.
[in]	AutoReset	: Enable/disable the counter auto reset.
[in]	Hit	: The source signal for the signal to count (see CVIOSources enum).
[in]	Gate	: The source signal for the gate (see CVIOSources enum).
[in]	Reset	: The source signal to stop the counter (see CVIOSources enum).

Returns:

An error code about the execution of the function.

Description:

The function permits to configure the scaler. Limit range is 0 - 1024 (10 bit). The hit signal source can be: input signal 0 (cvInputSrc0) or input coincidence (cvCoincidence). The gate signal source can be: front panel button or software (cvManualSW) or input signal 1 (cvInputSrc1). The reset signal source can be: front panel button or software (cvManualSW) or input signal 1 (cvInputSrc1).

CAENVME_API
CAENVME_SetScalerConf(long Handle, short Limit, short AutoReset, CVIOSources Hit, CVIOSources Gate, CVIOSources Reset).

4.3.18. CAENVME_SetOutputConf

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	OutSel	: The ouput line to configure (see CVOutputSelect enum).
[in]	OutPol	: The output line polarity (see CVIOPolarity enum).
[in]	LEDPol	: The output LED polarity (see CVLEDPolarity enum).
[in]	Source	: The source signal to propagate to the output line (see CVIOSources enum).

Returns:

An error code about the execution of the function.

Description:

The function permits to configure the output lines of the module. It can be specified the polarity for the line and for the LED. The output line source depends on the line as figured out by the following table:

Table 4.1: Source selection

SOURCE SELECTION					
		cvVMESignals	cvCoincidence	cvMiscSignals	cvManualSW
OUTPUT	0	DS	Input Coinc.	Pulser A	Manual/SW
	1	AS	Input Coinc.	Pulser A	Manual/SW
	2	DTACK	Input Coinc.	Pulser B	Manual/SW
	3	BERR	Input Coinc.	Pulser B	Manual/SW
	4	LMON	Input Coinc.	Scaler end	Manual/SW

CAENVME_API
CAENVME_SetOutputConf(long Handle, CVOutputSelect OutSel, CVIOPolarity OutPol, CVLEDPolarity LEDPol, CVIOSources Source).l

4.3.19. CAENVME_SetInputConf

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	InSel	: The input line to configure (see CVInputSelect enum).
[in]	InPol	: The input line polarity (see CVIOPolarity enum).
[in]	LEDPol	: The output LED polarity (see CVLEDPolarity enum).

Returns:

An error code about the execution of the function.

Description:

The function permits to configure the input lines of the module. It can be specified the polarity for the line and for the LED.

CAENVME_API
CAENVME_SetInputConf(long Handle, CVInputSelect InSel, CVIOPolarity InPol, CVLEDPolarity LEDPol).

4.3.20. CAENVME_GetPulserConf

Parameters:

[in] Handle	: The handle that identifies the device.
[in] PulSel	: The pulser to configure (see CVPulserSelect enum).
[out] Period	: The period of the pulse in time units.
[out] Width	: The width of the pulse in time units.
[out] Unit	: The time unit for the pulser configuration (see CVTimeUnits enum).
[out] PulseNo	: The number of pulses to generate (0 = infite).
[out] Start	: The source signal to start the pulse burst (see CVIOSources enum).
[out] Reset	: The source signal to stop the pulse burst (see CVIOSources enum).

Returns:

An error code about the execution of the function.

Description:

The function permits to read the configuration of the pulsers.

CAENVME_API
CAENVME_GetPulserConf(long Handle, CVPulserSelect PulSel, unsigned char *Period, unsigned char *Width, CVTimeUnits *Unit, unsigned char *PulseNo, CVIOSources *Start, CVIOSources *Reset).

4.3.21. CAENVME_GetScalerConf

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Limit	: The counter limit for the scaler.
[out] AutoReset	: The auto reset configuration.
[out] Hit	: The source signal for the signal to count (see CVIOSources enum).
[out] Gate	: The source signal for the gate (see CVIOSources enum).
[out] Reset	: The source signal to stop the counter (see CVIOSources enum).

Returns:

An error code about the execution of the function.

Description:

The function permits to read the configuration of the scaler.

CAENVME_API
CAENVME_GetScalerConf(long Handle, short *Limit, short *AutoReset, CVIOSources *Hit, CVIOSources *Gate, CVIOSources *Reset).

4.3.22. CAENVME_SetOutputConf

Parameters:

[in] Handle	: The handle that identifies the device.
[in] OutSel	: The ouput line to configure (see CVOutputSelect enum).
[out] OutPol	: The output line polarity (see CVIOPolarity enum).
[out] LEDPol	: The output LED polarity (see CVLEDPolarity enum).
[out] Source	: The source signal to propagate to the output line (see CVIOSources enum).

Returns:

An error code about the execution of the function.

Description:

The function permits to read the configuration of the output lines.

CAENVME_API
CAENVME_GetOutputConf(long Handle, CVOutputSelect OutSel, CVIOPolarity *OutPol, CVLEDPolarity *LEDPol, CVIOSources *Source).

4.3.23. CAENVME_ReadRegister

Parameters:

[in] Handle: The handle that identifies the device.
[in] Reg: The internal register to read (see CVRegisters enum).
[out] Data: The data read from the module.

Returns:

An error code about the execution of the function.

Description:

The function permits to read some internal registers: the input register, the output register and the status register. For the meaning and decoding of register bits see CVStatusRegisterBits, CVInputRegisterBits and CVOutputRegisterBits definitions and comments.

CAENVME_API
CAENVME_ReadRegister(long Handle, CVRegisters Reg, unsigned short *Data).

4.3.24. CAENVME_SetOutputRegister

Parameters:

[in] Handle	: The handle that identifies the device.
[in] Mask	: The lines to be set.

Returns:

An error code about the execution of the function.

Description:

The function sets the specified lines. Refer the CVOutputRegisterBits enum to compose and decode the bit mask.

CAENVME_API
CAENVME_SetOutputRegister(long Handle, unsigned short Mask).

4.3.25. CAENVME_ClearOutputRegister

Parameters:

[in] Handle	: The handle that identifies the device.
[in] Mask	: The lines to be cleared.

Returns:

An error code about the execution of the function.

Description:

The function clears the specified lines. Refer the CVOutputRegisterBits enum to compose and decoding the bit mask.

CAENVME_API
CAENVME_ClearOutputRegister(long Handle, unsigned short Mask).

4.3.26. CAENVME_PulseOutputRegister

Parameters:

[in] Handle	: The handle that identifies the device.
[in] Mask	: The lines to be pulsed.

Returns:

An error code about the execution of the function.

Description:

The function produces a pulse on the specified lines by setting and then clearing them. Refer the CVOutputRegisterBits enum to compose and decode the bit mask.

CAENVME_API
CAENVME_PulseOutputRegister(long Handle, unsigned short Mask).

4.3.27. CAENVME_ReadDisplay

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Value	: The values read from the module (see CVDisplay enum).

Returns:

An error code about the execution of the function.

Description:

The function reads the VME data display on the front panel of the module. Refer to the CVDisplay data type definition and comments to decode the value returned.

CAENVME_API
CAENVME_ReadDisplay(long Handle, CVDisplay *Value).

4.3.28. CAENVME_SetArbiterType

Parameters:

[in]	Handle:	The handle that identifies the device.
[in]	Value:	The type of VME bus arbitration to implement (see CVArbiterTypes enum).

Returns:

An error code about the execution of the function.

Description:

The function sets the behaviour of the VME bus arbiter on the module.

CAENVME_API
CAENVME_SetArbiterType(long Handle, CVArbiterTypes Value).

4.3.29. CAENVME_SetRequesterType

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Value	: The type of VME bus requester to implement (see CVRequesterTypes enum).

Returns:

An error code about the execution of the function.

Description:

The function sets the behaviour of the VME bus requester on the module.

CAENVME_API
CAENVME_SetRequesterType(long Handle, CVRequesterTypes Value).

4.3.30. CAENVME_SetReleaseType

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Value	: The type of VME bus release policy to implement (see CVReleaseTypes enum).

Returns:

An error code about the execution of the function.

Description:

The function sets the release policy of the VME bus on the module.

CAENVME_API
CAENVME_SetReleaseType(long Handle, CVReleaseTypes Value).

4.3.31. CAENVME_SetBusReqLevel

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Value	: The type of VME bus requester priority level to set (see CVBusReqLevels enum).

Returns:

An error code about the execution of the function.

Description:

The function sets the specified VME bus requester priority level on the module.

CAENVME_API
CAENVME_SetBusReqLevel(long Handle, CVBusReqLevels Value).

4.3.32. CAENVME_SetTimeout

Parameters:

[in] Handle: The handle that identifies the device.
[in] Value: Value of VME bus timeout to set (see CVVMETimeouts enum).

Returns:

An error code about the execution of the function.

Description:

The function sets the specified VME bus timeout on the module.

CAENVME_API
CAENVME_SetTimeout(long Handle, CVVMETimeouts Value).

4.3.33. CAENVME_SetFIFOMode

Parameters:

[in]	Handle	: The handle that identifies the device.
[in]	Value	: Enable/disable the FIFO mode.

Returns:

An error code about the execution of the function.

Description:

The function enables/disables the auto increment of the VME addresses during the block transfer cycles. With the FIFO mode enabled the addresses are not incremented.

CAENVME_API
CAENVME_SetFIFOMode(long Handle, short Value).

4.3.34. CAENVME_GetArbiterType

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Value	: The type of VME bus arbitration implemented (see CVArbiterTypes enum).

Returns:

An error code about the execution of the function.

Description:

The function get the type of VME bus arbiter implemented on the module.

CAENVME_API
CAENVME_GetArbiterType(long Handle, CVArbiterTypes *Value).

4.3.35. CAENVME_GetRequesterType

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Value	: The type of VME bus requester implemented (see CVRequesterTypes enum).

Returns:

An error code about the execution of the function.

Description:

The function get the type of VME bus requester implemented on the module.

CAENVME_API
CAENVME_GetRequesterType(long Handle, CVRequesterTypes *Value).

4.3.36. CAENVME_GetReleaseType

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Value	: The type of VME bus release policy implemented (see CVReleaseTypes enum).

Returns:

An error code about the execution of the function.

Description:

The function get the type of VME bus release implemented on the module.

CAENVME_API
CAENVME_GetReleaseType(long Handle, CVReleaseTypes *Value).

4.3.37. CAENVME_GetBusReqLevel

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Value	: The type of VME bus requester priority level (see CVBusReqLevels enum).

Returns:

An error code about the execution of the function.

Description:

The function reads the VME bus requester priority level implemented on the module.

CAENVME_API
CAENVME_GetBusReqLevel(long Handle, CVBusReqLevels *Value).

4.3.38. CAENVME_GetTimeout

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Value	: The value of VME bus timeout (see CVVMETimeouts enum).

Returns:

An error code about the execution of the function.

Description:

The function reads the specified VME bus timeout setting of the module.

CAENVME_API
CAENVME_GetTimeout(long Handle, CVVMETimeouts *Value).

4.3.39. CAENVME_GetFIFOMode

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Value	: The FIFO mode read setting.

Returns:

An error code about the execution of the function.

Description:

The function reads whether the auto increment of the VME addresses during the block transfer cycles is enabled (0) or disabled (!=0).

CAENVME_API
CAENVME_GetFIFOMode(long Handle, short *Value).

4.3.40. CAENVME_SystemReset

Parameters:

[in] Handle : The handle that identifies the device.

Returns:

An error code about the execution of the function.

Description:

The function performs a system reset on the module.

CAENVME_API
CAENVME_SystemReset(long Handle).

4.3.41. CAENVME_ResetScalerCount

Parameters:

[in] Handle : The handle that identifies the device.

Returns:

An error code about the execution of the function.

Description:

The function resets the counter of the scaler.

CAENVME_API
CAENVME_ResetScalerCount(long Handle).

4.3.42. CAENVME_EnableScalerGate

Parameters:

[in] Handle : The handle that identifies the device.

Returns:

An error code about the execution of the function.

Description:

The function enables the gate of the scaler.

CAENVME_API
CAENVME_EnableScalerGate(long Handle).

4.3.43. CAENVME_DisableScalerGate

Parameters:

[in] Handle : The handle that identifies the device.

Returns:

An error code about the execution of the function.

Description:

The function disables the gate of the scaler.

```
CAENVME_API
CAENVME_DisableScalerGate(long Handle).
```

4.3.44. CAENVME_StartPulser

Parameters:

[in] Handle : The handle that identifies the device.
[in] PulSel : The pulser to configure (see CVPulserSelect enum).

Returns:

An error code about the execution of the function.

Description:

The function starts the generation of the pulse burst if the specified pulser is configured for manual/software operation.

```
CAENVME_API
CAENVME_StartPulser(long Handle, CVPulserSelect PulSel).
```

4.3.45. CAENVME_StopPulser

Parameters:

[in] Handle : The handle that identifies the device.
[in] PulSel : The pulser to configure (see CVPulserSelect enum).

Returns:

An error code about the execution of the function.

Description:

The function stops the generation of the pulse burst if the specified pulser is configured for manual/software operation.

```
CAENVME_API
CAENVME_StopPulser(long Handle, CVPulserSelect PulSel).
```

4.3.46. CAENVME_IACKCycle

Parameters:

[in] Handle	: The handle that identifies the device.
[in] Level	: The IRQ level to aknowledge (see CVIRQLevels enum).
[in] DW	: The data width.(see CVDataWidth enum).

Returns:

An error code about the execution of the function.

Description:

The function performs a VME interrupt acknowledge cycle.

CAENVME_API
CAENVME_IACKCycle(long Handle, CVIRQLevels Level, void *Vector, CVDataWidth DW).

4.3.47. CAENVME_IRQCheck

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Mask	: A bit-mask 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).

4.3.48. 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).

4.3.49. 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).

4.3.50. 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).

4.3.51. CAENVME_ReadFlashPage

Parameters:

[in] Handle	: The handle that identifies the device.
[out] Data	: The data to write.
[in] PageNum	: The flash page number to write.

Returns:

An error code about the execution of the function.

Description:

The function reads the data from the specified flash page.

CAENVME_API
CAENVME_ReadFlashPage(long Handle, unsigned char *Data, int PageNum).

4.3.52. CAENVME_WriteFlashPage

Parameters:

[in] Handle : The handle that identifies the device.
[in] Data : The data to write.
[in] PageNum : The flash page number to write.

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

Description:
The function writes the data into the specified flash page.

CAENVME_API
CAENVME_WriteFlashPage(long Handle, unsigned char *Data, int PageNum).

4.3.53. CAENVME_SetInputConf

Parameters:
[in] Handle : The handle that identifies the device.
[in] InSel : The input line to configure (see CVInputSelect enum).
[in] InPol : The input line polarity (see CVIOPolarity enum).
[in] LEDPol : The output LED polarity (see CVLEDPolarity enum).

Returns:
An error code about the execution of the function.
Description:
The function permits to configure the input lines of the module. It can be specified the polarity for the line and for the LED.

CAENVME_API
CAENVME_SetInputConf(long Handle, CVInputSelect InSel, CVIOPolarity InPol, CVLEDPolarity LEDPol).

4.3.54. CAENVME_SetLocationMonitor

Parameters:
[in] Handle : The handle that identifies the device.
[in] Address :
[in] Write :
[in] Lword :
[in] Icak :

Returns:
An error code about the execution of the function.
Description:
The function sets the Location Monitor.

CAENVME_API
CAENVME_SetLocationMonitor(long Handle, unsigned long Address, CVAddressModifier Am, short Write,short Lword, short Iack).

4.3.55. CAENVME_WriteRegister

Parameters:

[in] Handle : The handle that identifies the device.
[in] Reg : The internal register to read (see CVRegisters enum).
[in] Data: The data to be written to the module.

Returns:

An error code about the execution of the function.

Description:

The function permits to write to all internal registers.

CAENVME_API
CAENVME_WriteRegister(long Handle, CVRegisters Reg, unsigned short Data).

5. Technical Support

CAEN makes available the technical support of its specialists at the e-mail addresses below:

support.nuclear@caen.it
(for questions about the hardware)

support.computing@caen.it
(for questions about software and libraries)