HSM 8170

Triple Port High Speed Memory

User's Manual, version 2.1
Designation: DOC 8170/UM
PN: 085.239

Version 2.1 - May 1992

CREATIVE ELECTRONIC SYSTEMS S.A.

Warranty Information

The information in this document has been checked carefully and is thought to be entirely reliable. However, no responsibility is assumed in case of inaccuracies. Furthermore, CES reserves the right to change any of the products described herein to improve reliability, function or design. CES neither assumes any liability arising out of the application or use of any product or circuit described herein nor conveys any licence under its patent rights or the rights of others.

Creative Electronic Systems,
70, Route du Pont-Butin
P.O. Box 107
CH-1213 PETIT-LANCY 1
SWITZERLAND

Contents

1. General Description

1.1 Module Description

The High-Speed Memory HSM 8170 is a one-slot VME module designed to provide a fast data acquisition unit. The HSM is built around a triple port static memory, allowing acquisition of fast ECL 16- or 32-bit data and read-out through the VME or VSB busses. The acquisition port meets the requirements of the FERA (Fast Encoding and Read-out ADC) protocol from LeCroy. The HSM 8170 can be equipped with either 512 Kbytes or 1 Mbyte of acquisition memory, nevertheless, several modules can be put in cascade or in flip-flop in order to increase the size of this acquisition memory.

1.2 Features

- 16- or 32-bit fast differential ECL acquisition port
- Acquisition logic compliant with the FERA protocol
- 10 Mcycles/s acquisition speed
- 64 x 32-bit words FIFO for data acquisition continuity
- 512 Kbytes or 1 Mbyte of fast acquisition memory (100 ns per word)
- Possibility to put modules in cascade for a larger acquisition memory area
- Possibility to configure two modules in flip-flop for a larger acquisition memory area
- VSB and VME read-out ports supporting block ascending mode (256 bytes maximum)
- 8-level VME Interrupter
- VME slave base address selectable with 5 jumpers
- VSB slave base address determined by the VSB geographical addressing
- Programmable boundary address logic for end of acquisition recognition

1.3 Performance

The acquisition port logic (write only) allows to reach up to 10 Mcycles/s with handshake. The transfer speed in the memory depends on the response time of the associated system.

VME and VSB Read / Write access times corresponding to a connection between an HSM 8170 and a FIC 823x are given in the following table

VME port, write access in word mode	430 ns per 32-bit word
VME port, read access in word mode	370 ns per 32-bit word
VME port, write access in block mode	340 ns per 32-bit word
VME port, read access in block mode	380 ns per 32-bit word

VSB port, write access in word mode	440 ns per 32-bit word
VSB port, read access in word mode	440 ns per 32-bit word
VSB port, write access in block mode	290 ns per 32-bit word
VSB port, read access in block mode	350 ns per 32-bit word

1.4 Block diagram description

Block Diagram

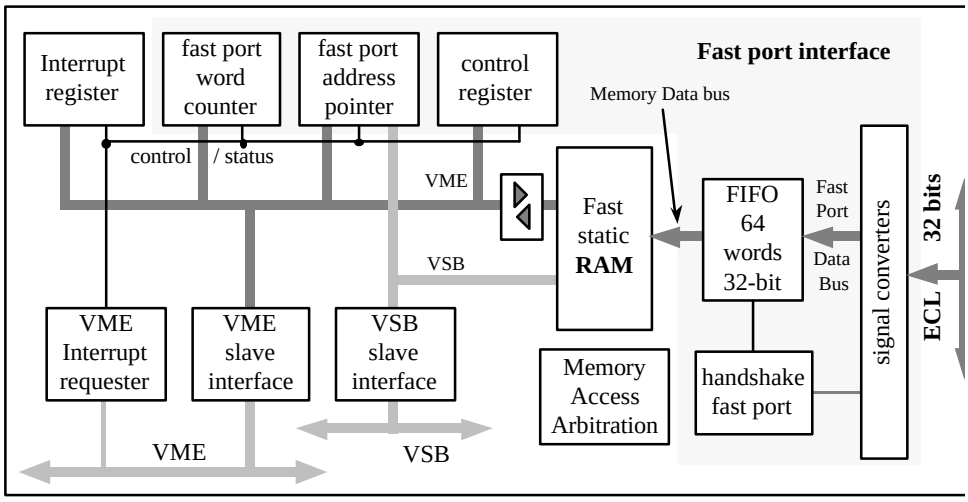


Fig 1.1

Note For a question of legibility, the block diagram presented above is simplified. A complete description of the different busses and blocks can be found hereafter.

32-bit VME data bus

The 32-bit internal bus is active when a VME cycle is recognized by the decoding logic of the VME slave port. The cycle may involve either the acquisition memory, or the different control registers of the unit. In the case of the cycle addressing the memory, an access request to the arbitration logic is done. This request will be granted only when pending transactions on the two other ports are completed.

32-bit VSB data bus

The 32-bit internal bus is active when a VSB cycle is recognized by the decoding logic of the VSB slave port. This request is granted only when the pending transactions on the acquisition port are completed.

19-bit VME Register data bus

The 19-bit internal bus, derived from the VME 32-bit data bus, is active when a VME cycle concerning the registers is recognized by the decoding logic of the VME slave port. This bus is independent of the acquisition memory's bus in order not to penalize access to the VSB and FAST PORT during access to the control and interrupt registers or when an interrupt acknowledge cycle is received.

32-bit Memory data bus

The 32-bit internal bus can be accessed by the 3 ports which are by order of decreasing priority: FAST PORT, VSB and VME. The access request to this bus is granted by the arbitration logic.

In order to achieve a short selection time, the memory is permanently active on the 32-bit Memory data bus (FAST PORT).

FIFO

A 64 x 32-bit words FIFO is inserted between the ECL / TTL converters and the acquisition memory in order to handle data during VME or VSB accesses on the memory.

32-bit Fast port data bus

The 32-bit internal acquisition bus is unidirectional and transmits data coming from the two acquisition input connectors into the FIFO. The data has been previously converted into TTL by differential converters.

Memory

The memory is mounted onto a special 68-pin socket. Depending on the option chosen, the socket may include one or two CES static memory modules. The global memory has a capacity of 512 Kbytes or 1 Mbyte (SRAM with an access time of 100 ns).

VME Slave Interface

This part concerns the drivers and receivers for the VMEbus, the decoding logic of the two address areas recognized by the slave port, the handshake logic between the VMEbus and the internal resources of the module as well as the address pointer for VME block-mode access.

The VME port works according to the extended addressing mode A32. It supports 16- or 32-bit data transfer either in single or in block-mode (256 bytes maximum).

In summary, the characteristics of the VME slave port are: D16 / D32 A32 BLT.

VME Interrupt Requester

This logic monitors the generation of the four interrupt sources of the module on the seven levels supported by the VMEbus. The STATUS / ID transferred is 8 bits wide. It contains information about the interrupt source, the interrupt vector's value on interrupt and the geographical address on VSB. The clearing of the most priority interrupt source is automatically done at the end of the interrupt acknowledge cycle.

In summary, the characteristics of the interrupt requester are: D08 (O) I (1-2-3-4-5-6-7) ROAK.

VSB Slave Interface

This part contains drivers and receivers of the VSBbus, the decoding logic of the address field recognized by the slave port, the handshake logic between the VSBbus and the memory as well as the address pointer for VSB block-mode access.

The VSB port can work according to the addressing mode A32 I/O address space + system address space. It supports data transfers in 32-bit in block-mode (256 bytes maximum), and 16 or 32-bit transfers in single transfer mode.

In summary, the characteristics of the VSB slave port are: D16 / D32 A32, D32 A32 BLT.

Fast port Interface

This part is delimited on the block diagram by a gray area and includes the following sections:

? The signal converters which allow the conversion of the ECL data connected to the input connectors into TTL logic. The conversion only consists on a translation of the level of the input signal.

? The FIFO have a depth of 64 x 32-bit words. It can receive data when the acquisition is enabled with a speed of 10 Mcycles/s. This FIFO allows the read-out of the data memory from the VSB or VME ports during acquisition.

? The handshake logic, which controls that each word sent has been correctly memorized in the FIFO, then in the memory.

p The acquisition control which can accept the transfer requests, if the internal registers have been correctly initialized and if the status present on the different inputs conditioning the acquisition are correct.

Œ The address pointer which determines at which address the data coming from the FAST PORT are transferred in the memory.

œ The word counter which defines the size of the memory reserved for acquisition. This counter has a logic for testing the boundaries of the memory space reserved for acquisition. This logic can stop acquisition at the end of transfer of a block of data.

Arbitration FAST PORT - VSB - VME

This logic monitors the access request to the memory by the three ports. The priorities to these ports are given in decreasing order as follows:

FAST PORT (only if acquisition enabled)
VSB
VME

When the memory access request is granted, the port concerned holds the access until the end of the cycle (single or block-mode). For the FAST PORT, the access to the memory is maintained until the FIFO is empty. When the acquisition is enabled and when there is no request on the VSB or VME ports, the arbitration logic selected by default is the FAST PORT. This is done in order to handle the transfer request coming from this port with priority. In this case, the memory is permanently selected by the address pointer.

1.5 Ordering information and Reference options

HSM 8170	High speed memory with 3 ports (FAST, VME, VSB). VME and VSB supporting DMA block mode and requiring one of the two options listed below.
OPT 8170/5	512 Kbytes Static RAM triple port memory.
OPT 8170/6	1 Mbyte Static RAM triple port memory.

1.6 Power supply requirements

The HSM 8170 equipped with 1 Mbyte of SRAM requires:

+ 5 V 5.8 A
- 12 V 1 A

Note The internal voltages - 5 V and - 2 V used by the ECL logic are produced from the - 12 V voltage and are dependent on two regulators.

1.7 Reference manuals

The following publications may provide helpful informations. They are not shipped with this manual.

Document	**Manufacturer**
VMEbus specifications rev. C	VITA / IEC
VSBbus specifications rev. C	VITA / IEC
System 4300. Fast Encoding and Readout ADC (FERA)	LeCroy
FASTBUS Segment Manager / Interface opt ECL 1821	LeCroy

CAMAC 8 input ADC 4418 (User's Guide)	SILENA

2. VME Slave Interface

2.1 General Description

The VME slave interface allows to access all resources of the module. The access space is divided in two areas 1 Mbyte wide each. The lowest one concerns accesses to the acquisition memory, the second one, accesses to the control and status registers.

The VME slave interface can only accept aligned data transfers (no byte transfer).

- The memory area can be accessed in two data transfer modes (single or block-mode) D16 or D32.
- The address space corresponding to the register can only be accessed in single transfer mode D16 or D32.

VME Slave Resources Mapping

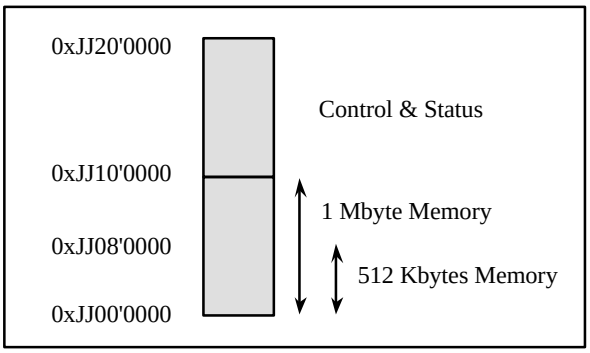


Fig 2.1

The VME slave base address can be configured with five jumpers located on the module (J09...J05), which determine the address bits <A28...A24>. By convention in this manual, the VME slave base address is given as 0xJJ00'0000, "J" depending on the position of the five jumpers.

2.2 AM decoding

The HSM 8170 decodes the following Address Modifiers codes:

? Memory D16 or D32 selection

extended mode A32

AM = 0x09 Non-privileged Data Access
AM = 0x0B Non-privileged Block Transfer
AM = 0x0D Supervisory Data Access
AM = 0x0F Supervisory Block Transfer

? Registers D16 or D32 selection

extended mode A32

AM = 0x09 Non-privileged Data Access
AM = 0x0D Supervisory Data Access

2.3 Extended access A32 - D16 / D32

The extended access mode allows the addressing of the module in any one of the 32 windows (defined by jumpers J05 to J09) in the 4 Gbytes corresponding to the VME A32 address field. The three most significant address bits <A31..A29>, as well as the 3 bits <A23..A21> are set to "0".

Acquisition memory field

A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
0	0	0	J09	J08	J07	J06	J05	0	0	0	⓪	SRAM Addressing			

A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
SRAM Addressing															0

Registers field

A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
0	0	0	J09	J08	J07	J06	J05	0	0	0	①	Registers			

A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
Registers															0

2.4 Block mode access

The VME port supports the block ascending mode (BLT) in D16 and D32 mode. The address counter associated with this mode can receive a block composed of 256 bytes. This mode is characterized by the AM codes 0x0B and 0x0F. It allows a maximum acquisition speed on the VME port for the transfer cycles to be continued with the maximum speed of the memory. In this mode, the phase of arbitration and loading of the address is made only at the beginning of the block-transfer cycles. The following memory accesses follow at a speed which only depends on the access time of the memory and the controller.

If this type of access is done during acquisition, it is advised to limit the number of words for each block transfer in order to avoid the overflow of the FIFO capacity. This FIFO allows the FAST PORT to work at the maximum speed even during the memory access from the VME port. An overflow of the FIFO capacity can only be triggered by a block-mode transfer, which lasts more than 6 µs when the FAST PORT works at the maximum speed. The handshake allows to limit the speed of the data acquisition and insures that all transfers are carried out.

Warning In the case that the FAST PORT works at the maximum speed of 100 ns per word without handshake, it is imperative that the module is not accessed neither from VME nor from VSB. No polling mode is allowed and the only communication mode between the HSM and its associated CPU should be the use of the interrupts.

For a READ operation between the HSM and its CPU through VME, the speed limit of the module (after the arbitration phase and address loading) is 125 ns per 32-bit word (true speed with the FIC 823x, 300 ns per 32-bit word).

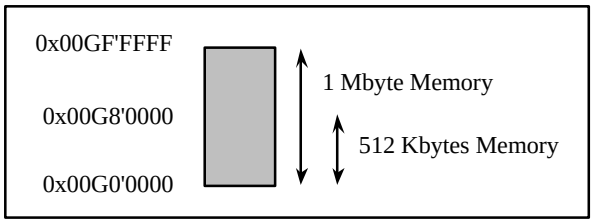
3. VSB Slave Interface

3.1 General Description

The VSB slave port only allows to access the acquisition memory only. This access can be made in 16-bit or 32-bit, depending on the single-transfer mode. Accessing the memory in 32-bit is also foreseen in the block-transfer mode, with a maximum block length of 256 bytes.

VSB Slave Resources Mapping

Fig 3.1



The VSB slave base address depends on the physical position of the module in the VSB crate (geographical addressing). Refer to VSB specifications for more information. The geographical addressing modifies the three bits <A22...A20> of the VSB slave address. By convention in this manual, the VSB slave base address is given as 0x00G0'0000, "G" depending on the location of the HSM 8170 in the VSB crate.

The VSB bus can support a maximum of 6 slots. Considering the slot occupied by the VSB master module, this bus can only receive a maximum of five HSM 8170 memories.

3.2 VSB Address Decoding

Twenty bits of the address field are used for the addressing of the static memory. The bits <A22..A20> are decoded by a PAL, which selects the base address accepted by the module. The decoding of these 3 bits depends on the three lines (GA0, GA1, GA2) of the VSB Geographical Addressing.

Memory addressing on VSB

A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
0	0	0	0	0	0	0	0	0	GA2	GA1	GA0	SRAM Addressing			

A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
SRAM Addressing															0

3.3 Geographical Decoding

The three geographical addressing bits GA2...GA0 (<A22...A20>) allow the selection of the VSB slave base address for each HSM 8170 memory present on the VSB bus, according to its position in the crate.

Those three bits are transmitted to a VME unit through the STATUS / ID byte during an interrupt acknowledge cycle.

When no VSB backplane is present in the crate where the HSM 8170 is plugged, the three physical lines GA2...GA0 are pulled-up and the bits <A22...A20> are set to "1".

Considering the slot occupied by the VSB master module, the VSB bus can receive a maximum of five HSM 8170 memories.

The correspondence between the slot number and the base address is defined below:

Slot	**1**	**2**	**3**	**4**	**5**	**6**
GA	000	001	010	011	100	101
Addr. windows VSB (Hexa)	----------	001xxxxx	002xxxxx	003xxxxx	004xxxxx	005xxxxx
STATUS/ID byte VME (binary)	----------	xxx110xx	xxx101xx	xxx100xx	xxx011xx	xxx010xx

3.4 Block Mode Access

The VSB slave port supports block transfers for memory access in D32 mode. It can accept up to 256 bytes.

The VSB port supports block ascending mode (BLT) in D32 mode only. The address counter associated to this mode allows us to receive a block composed of 64 words of 32-bit. This mode is characterized on the VSBbus by maintaining the PAS line until the end of the block. This transfer mode assures a maximum acquisition speed on the VSB port, for the data transfer cycles are continued to the maximum speed of the memory. In this mode, the phase of arbitration and loading of the address is made only at the beginning of the block-transfer cycles. The following memory access follows at a speed which depends only on the access time of the memory and the controller.

If this type of access is done during acquisition, it is advised to limit the number of words for each block transfer in order to avoid the overflow of the FIFO capacity. This FIFO allows the FAST PORT to work at the maximum speed even during the memory access from the VSB port. An overflow of the FIFO capacity can only be triggered by a block-mode transfer, which lasts more than 6 µs when the FAST PORT works at the maximum speed. This overflow provokes a loss in the acquisition speed. The handshake permits in this case to limit the speed of the data acquisition and ensures that all transfers are carried out. In the case of the FAST PORT working at the maximum speed of 100 ns per word without handshake, it is imperative to limit the number of words transferred per block, in order that the total time for each block transfer does not exceed 6 µs. An overflow of this limit would provoke data loss from the FAST PORT.

It must be noted that when used with the FIC 8230, the number of words transferred in this mode is limited to 4 and consequently avoid the loss of data even if the FAST PORT is used without handshake.

For a READ operation between an HSM 8170 and its associated CPU through the VSB bus, the speed limit of the module, after the phase of arbitration and address loading, is 100 ns per 32-bit word (true speed with the FIC 8230, 250 ns per 32-bit word).

4. VME and VSB Slave Resources

4.1 Data transfer capabilities

The different registers as well as the memory are accessible in 16-bit and 32-bit. The 8-bit mode is not used and gives a BERR response except for the Interrupt Request Register for which access in 8-bit is allowed.

Warning The Control and Status registers are accessible from VME only. The VSB can only access the memory.

4.2 VME Interrupt Requester

The VME interrupt requester can send an interrupt on one of the seven levels supported by VME bus. These interrupts depend on the status of the Interrupt and Control Register. The module includes four interrupt sources which are differentiated by the two least significant bits of the interrupt vector. When the module sends an interrupt on one of the seven levels, it is waiting for an interrupt acknowledge corresponding to its level and immediately transmits its STATUS / ID vector. At the end of the interrupt cycle, the interrupt source having sent this vector is then cleared. The interrupt vector is 8 bits wide.

In summary, the characteristics of the interrupt requester are D08 (O) I (1-2-3-4-5-6-7) ROAK.

This register is used for the following:

- Programming the level which will be used when the module will send an interrupt.
- Clearing the pending interrupt source, if it has not been acknowledged by the VME interrupt handler.
- Reading the STATUS / ID vector which will be transmitted during an interrupt acknowledge cycle sent by the interrupt handler which supports the level defined by this same register.

Interrupt Register (R/W)

Address VME: 0xJJ10'0000 - 0xJJ10'0003
AM 0x09 / 0x0D
Extended Addressing Mode

D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
reserved (always set to "1")															

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
R3	R2	R1	R0	CI	IR2	IR1	IR0	SW3	SW2	SW1	GA2	GA1	GA0	S1	S0

bits <D31...D24>	Reserved	(set to "1")
bits <D23...D16>	Reserved	(set to "1")
bits <D15...D12>	R3...R0	Reserved. Do not use. Should not be taken into consideration

4.2.1 Interrupt Clearing

bit <D11>	W	CI	Clear Pending Interrupt. The bit <D11> can clear a pending interrupt request. This possibility is useful when, for example, the module sends two interrupts simultaneously. In this case, the highest priority source will be cleared by the interrupt acknowledge cycle. The resident program in the interrupt handler can control if another source is present. If so, this source can be directly handled and cleared by this bit. To execute this operation on the pending interrupt source with the highest priority, we have to successively set and clear this bit. To clear a pending interrupt, this interrupt must be enabled into the Control Register (§ 4.3)

4.2.2 Interrupt Levels

bits <D10...D08>	R/W	IR2...IR0	IRQ levels. The selection of the level corresponding to the interrupt request depends on the three bits <D10...D08>. The value is binary coded. Level 7 has the highest priority, and level 1 the lowest priority. If level 0 is loaded, the interrupts will not be transmitted on the VMEbus. If the interrupts are to be handled, it is necessary to select in this register the level which corresponds to the possibilities of the interrupt handler. In fact, some of these handle only interrupts sent on level IRQ7.

4.2.3 STATUS / ID Vector

The bits <D07...D00> constitute the STATUS / ID Vector.

bit <D07>	R	SW3	Interrupt Vector's value on interrupt. Reflects the status of the jumper J12
bit <D06>	R	SW2	Interrupt Vector's value on interrupt. Reflects the status of the jumper J11
bit <D05>	R	SW1	Interrupt Vector's value on interrupt. Reflects the status of the jumper J10
bits <D04...D02>	R	GA2...GA0	Geographical addressing on VSB. If no VSB backplane is present in the crate, those bits are read as "1"
bits <D01...D00>	R	S1...S0	Source Interrupt. They depend on the four interrupt sources. Only the interrupt source with the highest priority gives its address in the STATUS / ID byte.

The two least significant bits <D01...D00> encode the Source Interrupt as follows (by order of decreasing priority):

S1	S0		
1	1	FIFO overflow	Interrupt indicating that the FAST PORT has received a data which was not memorized in the FIFO. This can occur when the FAST PORT is used without handshake at the maximum speed and when the data processed is handled simultaneously on one of the VME or VSB ports in block-mode. This interrupt case indicates a loss of data and provokes the end of the acquisition on the FAST PORT.
1	0	Memory full	Interrupt indicating that the word counter has reached "0", which means that the memory limits have been reached. This interruption also provokes the end of the acquisition on the FAST PORT.
0	1	Memory overflow	Interrupt indicating that the memory boundaries have been reached and that the FAST PORT interface is completing acquisition when the BUSY input becomes cleared.
0	0		End of acquisition on FAST PORT or External Interrupt released from the front panel.

At power on or during a SYSRESET, the bits Read / Write of the interrupt register are cleared.

4.3 Control Register

Control Register (R/W)

Address VME: 0xJJ10'0004 - 0xJJ10'0007
AM 0x09 / 0x0D
Extended Addressing Mode

D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
reserved (always set to "1")															

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
L2	L1	L0	EDA	ED3	ED2	ED1	ED0	FB	ST2	ST1	ST0	IS3	IS2	IS1	IS0

bits <D31...D24> Reserved Always set to "1"

bits <D23...D16> Reserved Always set to "1"

4.3.1 Memory Overflow

bits <D15...D13> L2...L0 Overflow limits. The 3 bits <D15..D13> indicate when the acquisition on the input port is stopped. They allow the user to select the limits of the number of acquisitions in order they can be transferred into the memory. They have two functions for the module:

? When the contents of the word counter is equal to the value selected by this register, the memory overflow interrupt source is triggered. This interrupt source appears only during the flow of the word counter value N to N+1. (N = value defined by the register).

? When the contents of the word counter is equal or inferior to the value selected by the register, the acquisition on the FAST PORT will be suspended when the BUSY input is brought to zero.

The three bits encode the number of acquisition available before a memory overflow interrupt in the following manner:

L2	L1	L0	
0	0	0	Memory Overflow disabled
0	0	1	512 words
0	1	0	1024 words
0	1	1	1536 words
1	0	0	2048 words
1	0	1	2560 words
1	1	0	3072 words
1	1	1	3582 words

4.3.2 Enable / Disable Acquisition

bit <D12> EDA Enable / Disable Acquisition. The bit <D12> allows the user to control the acquisition from the VME. It is indispensable that this bit is set, so that the acquisition can be carried out on the FAST PORT. The inputs located on the front panel will be active only when this bit is set to 1. When the acquisition is stopped by means of this bit, the end of acquisition interrupt source is not transmitted.

4.3.3 Enable / Disable Interrupts

The 4 bits <D11..D08> allow the user to select which interrupt source will be authorized to send an interrupt request on the VME. A "1" authorizes the transmission of the interrupt if the level at which it should be sent is selected beforehand.

bit <D11> ED3 When set to "1", enables Interrupt on End of Acquisition.
When set to "0", disables Interrupt on End of Acquisition.

bit <D10> ED2 When set to "1", enables Interrupt on Memory Full.
When set to "0", disables Interrupt on Memory Full.

bit <D09> ED1 When set to "1", enables Interrupt on Memory Overflow.
When set to "0", disables Interrupt on Memory Overflow.

bit <D08> ED0 When set to "1", enables Interrupt on FIFO Overflow.
When set to "0", disables Interrupt on FIFO Overflow.

4.3.4 Fast Port Status

The status of the bits <D07..D04> (Read only) allows the user to control the input configuration and the functioning of the FAST PORT.

bit <D07>	FB	FERA Busy. This bit indicates in real-time the status of the BUSY line of the FERA bus. When read as "0", a transfer is actually active. If read as "1", no transfer is active.
bit <D06>	ST2	FIFO not empty. Read as "1" indicates that the FIFO has some data to transmit into the memory.
bit <D05>	ST1	Acquisition on 16- or 32-bit. This bit reflects the status of the jumper J04. Read as "1", this bit indicates that the acquisition is on 16-bit. The acquisition on the 32-bit memory is carried out in two transfers. The first transfer is carried out from DATA00 to DATA15 fields, while the second transfer is made from DATA16 to DATA31. In this mode, the word counter is decremented and the address pointer is incremented at each transfer. When this bit is read as "0", the FAST PORT is configured as a 32-bit input port. The acquisition in the 32-bit memory is made in one transfer in the DATA00 to DATA31 fields. In this mode, the word counter is decremented and the address pointer is incremented twice at each transfer. Figures 4.1 and 4.2 (§ 4.4) show how the incoming data are handled in the memory according to the 16- or 32-bit acquisition mode selected.
bit <D04>	ST0	Acquisition ON. Read as "1" indicates that the acquisition is ON. This bit reflects the status of the flip-flop which controls the acquisition on the FAST PORT, as well as the CARRY input.

4.3.5 Source Register Interrupts

The four bits <D03..D00> allow the user to know which interrupt source is present:

bit <D03>	IS3	End of Acquisition. Read as "1", indicates that an End of Acquisition Interrupt has occurred. This interrupt source is triggered by the changing of the flip-flop <ACQFF> status from 1 to 0 (this flip-flop controls acquisition on the fast port). The conditions causing the interrupt are the following: • Pulse on STOP input • FIFO OVERFLOW interrupt • MEM FULL interrupt • MEM OVERFLOW if input BUSY = 0

bit <D02>	IS2	Memory Full. Read as "1" indicates that a Memory Full Interrupt has occurred. This interrupt source is triggered when the transfer request on the FAST PORT has been accepted and the contents of the word counter is equal to 0. This interrupt signals that there is no space available for the next transfer requests that may follow. In the case of the FAST PORT being used without handshake, the following data are lost. If the handshake is used, the transfers are suspended until there is enough free space in the memory and the new conditions of acquisition are defined. It must be noted that if the memory overflow conditions have been correctly chosen and that the BUSY input is used, this interrupt must never occur.
bit <D01>	IS1	Memory Overflow. Read as "1" indicates that a Memory Overflow Interrupt has occurred. This interrupt source is triggered when the transfer request on the FAST PORT has been accepted and the following conditions are fulfilled: ? Word counter contents is equal or smaller than the limits fixed in the memory overflow register. ? BUSY input = 0 This interrupt signals that an end of transfer in a whole number of events has been detected and that there is not enough space in the memory to allow data transfer of the following acquisition.
bit <D00>	IS0	FIFO Overflow. Read as "1" indicates that a FIFO Overflow Interrupt has occurred. This interrupt source is triggered when the transfer request on the FAST PORT has not been accepted, because the FIFO was full. This interrupt can occur only in the case of the acquisition being used without handshake and the incoming data arriving faster than the FIFO can access memory. This interrupt indicates loss of data, but will only occur in the case that there are memory accesses in block mode of duration greater than 6 µs, concurrent with the data acquisition.

The conditions causing the clearing of those interruptions are the following:

- POWER ON
- SYSRESET
- IACK VME cycle if this interrupt source is authorized and is of the highest priority.
- SET / CLEAR sequence on bit <D11> of the interrupt register if this source is of the highest priority.

Those sources can only send a VME interrupt request if the two following conditions are fulfilled:

? The bit corresponding to the source in the enable interrupt register must be set to 1.

? The contents of the IRQ level register must be unequal to 0.

4.4 Address Pointer

Address Pointer (R/W)

Address VME: 0xJJ10'0008 - 0xJJ10'000B
AM 0x16 / 0x1D
Extended Addressing Mode

Only D16 or D32 accesses are authorized in this register.

D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
not used (always set to "1")													P18	P17	P16

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
P15	P14	P13	P12	P11	P10	P09	P08	P07	P06	P05	P04	P03	P02	P01	P00

bits <D31..D19> Reserved Always set to "1"

bits <D18..D16> P18...P16 MSBs of the memory address where the next data transfer from the FAST PORT will be stored

bits <D15..D00> P15...P00 LSBs of the memory address where the next data transfer from the FAST PORT will be stored

This 19-bit counting register allows the user to select the address where the next data transfer from the FAST PORT must be stored. At the end of each transfer into the memory, this register is incremented. The conditions of loading and increment of this register depend on the acquisition mode selected on the FAST PORT.

16-bit mode

When the acquisition is made in this mode, this pointer selects a 16-bit word. As the memory is 32-bit wide, this pointer will alternatively select the address corresponding to <D15...D00>, and then that corresponding to <D31...D16>. The contents of the pointer is incremented by one at the end of each transfer into the memory. Any address value can be loaded in this register.

32-bit mode

When the acquisition is made in this mode, this pointer selects a 32-bit word. As the memory is 32-bit wide, this pointer will select the address corresponding to <D31...D00>. The contents of the pointer is incremented by one at the end of each transfer into the memory. Only even addresses can be loaded in this register.

Warning This register can only be loaded when the acquisition on the FAST PORT is disabled (bits D12, D06 and D04 = 0 in the control register). If this register is loaded during a FIFO transfer into the memory, this will cause addressing errors.

16-bit Acquisition Mode

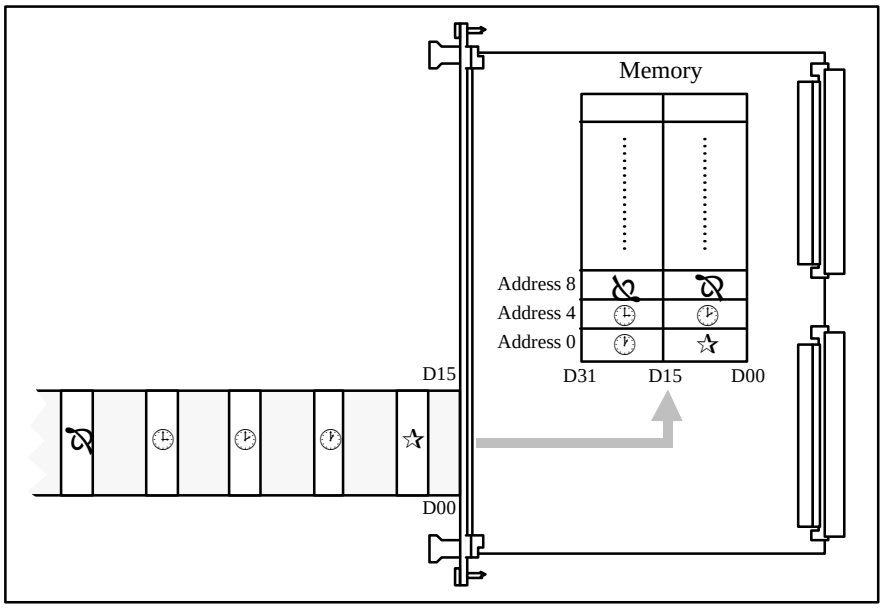


Fig 4.1

32-bit Acquisition Mode

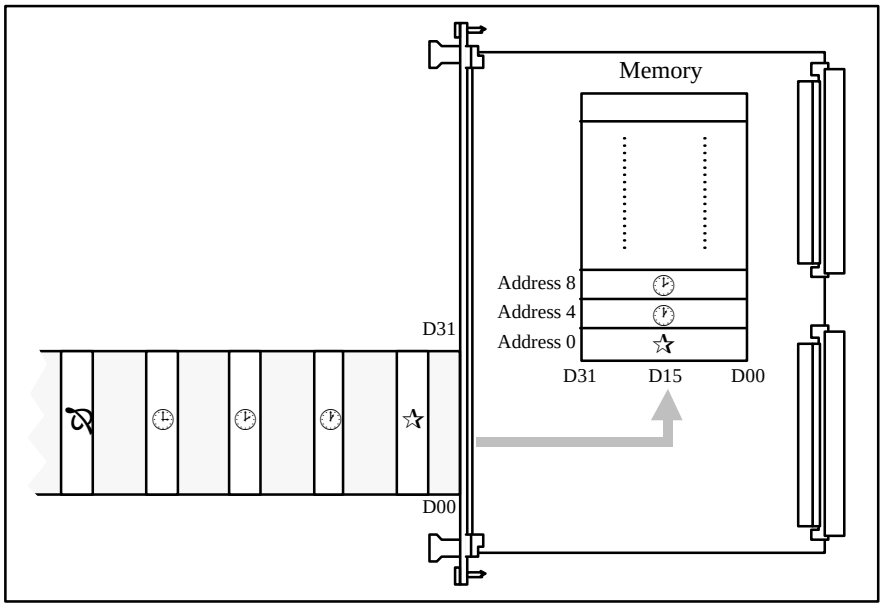


Fig 4.2

4.5 Word Counter

Acquisition Word Counter (R/W)

Address VME: 0xJJ10'000C - 0xJJ10'000F
AM 0x16 / 0x1D
Extended Addressing Mode

Only D16 or D32 accesses are authorized in this register.

D31	D30	D29	D28	D27	D26	D25	D24	D23	D22	D21	D20	D19	D18	D17	D16
not used (always set to "1")												C19	C18	C17	C16

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
C15	C14	C13	C12	C11	C10	C09	C08	C07	C06	C05	C04	C03	C02	C01	C00

bits <D31..D20> Reserved Always set to "1"

bits <D19..D16> C19...C16 MSBs of the setting of the size of the memory where data coming from the FAST PORT is to be transferred

bits <D15..D00> C15...C00 LSBs of the setting of the size of the memory where data coming from the FAST PORT is to be transferred

This 19-bit down counter allows the user to select the size of the memory where data coming from the FAST PORT is to be transferred. This register is decremented after each transfer from the FAST PORT into the FIFO.

The word counter is able to handle more than the maximum number of memory positions in order to take into account the free spaces when the handling of data already transferred is simultaneously executed with the acquisition.

The number contained in this register corresponds to 16-bit words if the FAST PORT is configured in 16-bit mode, or 32-bit words in 32-bit mode.

The counting register determines the interrupt conditions of MEMORY OVERFLOW and MEMORY FULL.

Warning This register can only be loaded when the acquisition on the FAST PORT is disabled (bits D12, D06 and D04 set to "0" in the Control Register). If this register is loaded during a FAST PORT transfer into the FIFO, this will cause an error in the accepted words number.

4.6 Memory Access

Data High-Speed Memory (R/W)

Address VME or VSB: Base address + 0x000x'xxxx
AM 0x09 / 0x0B / 0x0D / 0x0F
Extended Addressing Mode

This 32-bit static memory can have a size of 512 Kbytes or 1 Mbyte. It can be accessed from three ports in write (FAST PORT, VSB, VME), and from two ports (VSB, VME) in read/write.

When the acquisition port is disabled, this memory can be used as a fast VME / VSB memory. In this case, a VSB access has priority over a VME access.

When the acquisition is enabled, in absence of requests on the VSB / VME ports, the FAST PORT selects this memory permanently in order to maintain a minimum delay for a write request from this port. For that reason, the VME / VSB access time can be slowed down.

4.7 SYSRESET Command

This VME control line allows to initialize the HSM 8170 module. It operates on the FIFOs, the internal logic of the module as well as the control and interrupt registers. After this command, the FIFOs are empty and the read/write bits of the control and interrupt registers are cleared.

5. Fast Port

5.1 General Description

This port is unidirectional and allows to receive 16- or 32-bit data from the FERA bus and to write them into the memory at a speed of 10 Mcycles/s. The data coming from the FERA bus are driven to the memory through a 64-word depth FIFO, which guarantees the continuity of acquisition. Thanks to the FIFO, the acquisition will be assured even if the memory is accessed from the VME or VSB bus.

The functioning of this port depends on the initialization of the internal registers on the one hand, and also on the status of the NIM and ECL inputs located on the front panel of the module on the other hand. When the acquisition is authorized by these different conditions, the FIFO associated to this port has the highest priority on the arbitration logic of the memory access request. Once this arbitration logic grants the memory access to the FIFO, they maintain arbitration until it is empty. The memory access request, coming from the two other ports, can only be granted when the FIFO is empty.

FIFO Implementation

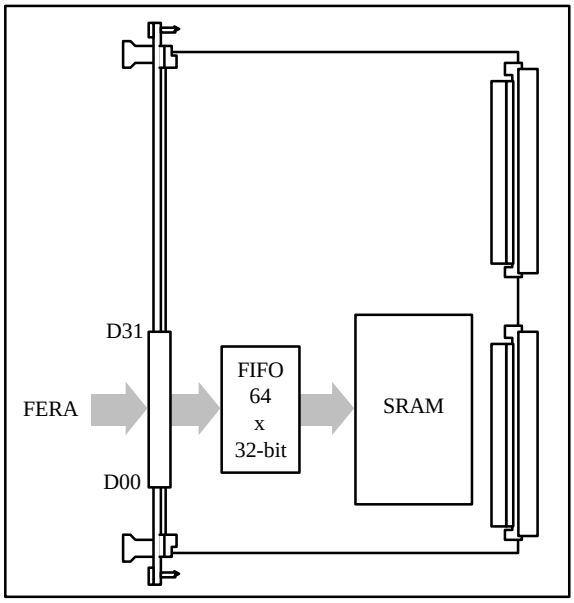


Fig 5.1

In order to assure the compatibility between the different systems, the external control lines of this port permit the reception and transmission of NIM and ECL differential levels. The data input bus is provided with fast differential receivers which accept differential ECL input signals.

5.2 ECL Control Inputs / Outputs

ECL Differential Control Inputs

Those inputs are terminated on a 100 ? impedance.

VETO This input allows to inhibit the WSI input when the module is daisy-chained with other HSM 8170 memories. In this application, this input is connected to the FULL* output of the module at the top of the daisy-chain constituted by connections FULL*-VETO (see § 5.5.3 Memories in cascade).

WSI* Write Strobe Input. This input indicates that the data are present on the 16- or 32-bit data bus. Minimum pulse width: 30 ns.

BUSY When active, this input indicates that a transfer is currently active. When this signal becomes inactive, the acquisition is completed.

ECL Differential Control Outputs

FULL* This output becomes active when the word counter reaches the value "0".

ACK Acknowledge. This output becomes active when the module has transferred the data presented on the input bus in the FIFO. This signal is an echo of the WSI input signal (if accepted by the acquisition logic). This answer will be inhibited on the following conditions:

- the BUSY line is at "0"
- the acquisition is disabled
- the word counter is at "0"
- the input VETO is at "1"
- the input CARRY is at "1"

OVF Overflow. This output is active once the contents of the word counter is equal or smaller than the value specified by the overflow register. It will stay active until the re-initialization of the word counter. This signal will be used to inhibit all new acquisition.

5.3 NIM Control Inputs / Outputs

NIM Control Inputs

Those inputs are terminated on a 50 ? impedance.

START This input allows to start or restart an acquisition, if the acquisition is authorized by the VME control register. Minimum pulse width: 20 ns.

STOP This input allows to stop an acquisition initiated by a START pulse. Minimum pulse width: 20 ns.

CARRY This input allows to inhibit the action of the WSI input.

WSI Write Strobe Input (see WSI* ECL description in § 5.2). The 50 ? input terminator can be by passed with the help of jumper J03, in order to distribute this signal onto other boards.

NIM Control Outputs

ACQ ON Acquisition ON. This output is at "1" when an acquisition is ON. This output must be terminated on a 50 ? terminator (jumper J03). In case of many HSM connected to WSI, only the last one must be terminated.

IRO Inhibit read-out. This output is used to inhibit the read-out process until the memory can accept new data. The status of this output is conditioned by the acquisition flip-flop as well as by the CARRY input. This signal can be "ORed" with the other memories, when many memories are used in cascade on the same acquisition bus.

WAO Write Acknowledge Output. This signal can be "ORed" with the other memories in the case of many memories being used in cascade on the same acquisition bus (see ACK ECL description in § 5.2).

OVF Overflow. This output is active once the contents of the word counter is equal or smaller than the value specified by the overflow register. It will become active until the re-initialization of the word counter. This signal will be used to prohibit all new acquisition.

5.4 ECL Data Inputs

If the acquisition is authorized, the two 34-pin connectors allow to receive a 32-bit word in the memory through the FIFO, each time a WSI signal appears. The reception speed of these words can reach 10 Mcycles/s. These inputs are differential. The input terminators are placed on some sockets in order to eliminate or modify them. The module is factory configured to work with ECL levels terminated on 100 ? .

In case of more than one HSM connected to the data bus, only the last one must be terminated. All the other resistor networks must be removed.

5.5 Interfacing the HSM 8170

5.5.1 Acquisition Mode with one HSM and Handshake

The mode used can vary slightly depending on the option chosen. The following diagram shows the transactions between the peripherals and the memory of the acquisition port. This mode is valid when receiving data from a FERA system.

Modules Connection with Handshake Mechanism

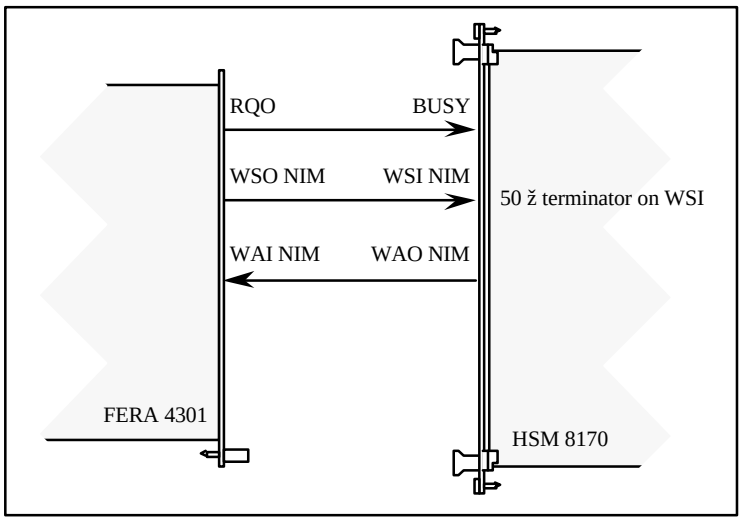


Fig 5.2

Fast port timing with handshake

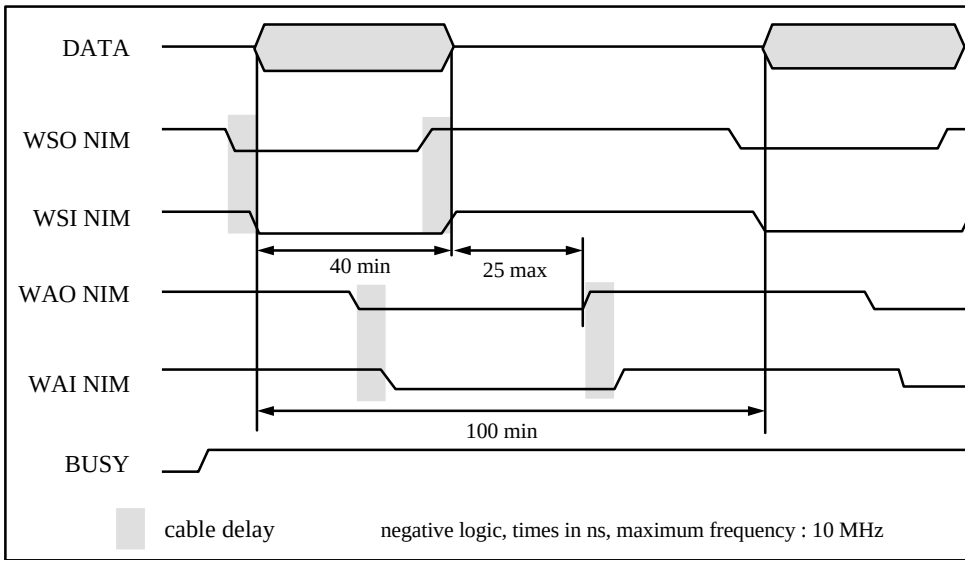


Fig 5.3

We can see in the diagram above that the acquisition speed depends partly on the WSI-WAO delay due to the memory response time in case that it is accessed through the VME or VSB ports.

5.5.2 Acquisition Mode with one HSM and no Handshake

In the case of a slow-down related to a too great cable propagation delay, the module can be used without handshake. This mode requires certain precautions because it can lead, in certain cases, to the loss of data. Two cases causing the loss of data can occur:

? In the first case, it is signaled by a FIFO OVERFLOW interrupt linked to the transfers executed on the VME or VSB ports in block-mode, and over a too long length of time.

? In the second case, data loss will occur when the acquisition is disabled either by an external command STOP or by an overflow of the memory capacity (MEM FULL).

This mode must not be used when associating the boards in cascade.

Modules Connection without Handshake Mechanism

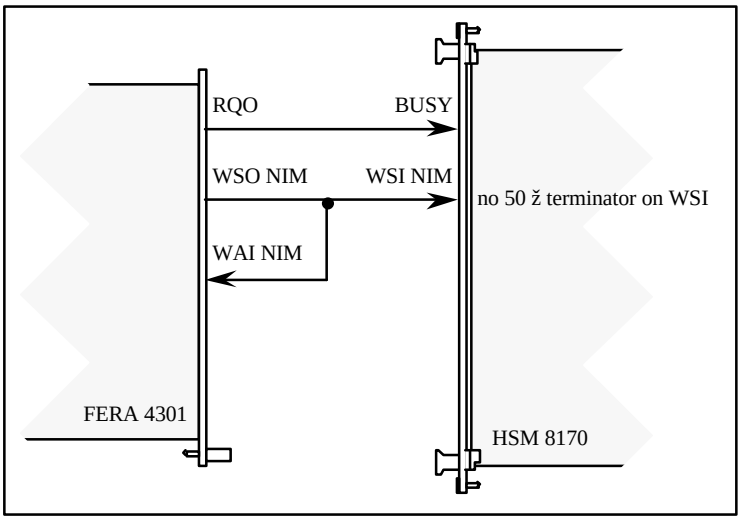


Fig 5.4

To use the module in this mode, we have to directly close the WSO output on the WAI input of the system sending data and connecting it to the WSI input of the HSM 8170 memory (the output WAO of the memory will not be used).

If the cable length is shorter than 1.5 meter, no 50 ? terminator is required on the WSI line.

If the cable length is longer than 1.5 meter, a 50 ? terminator is required on the WSI line. In this case, we must make sure that the NIM WSO output of the peripheral equipment has a fan-out of 2 (consumption of 36 mA).

Fast port timing without handshake

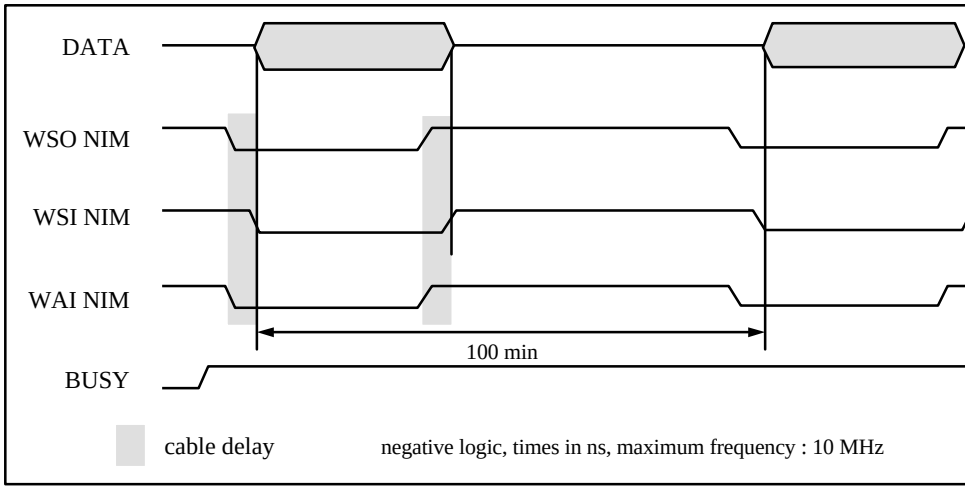


Fig 5.5

5.5.3 Acquisition Mode with more than one HSM in Cascade

In some cases, the memory size of one HSM can be too small to record an entire event. For that reason, the size of the memory can be extended by the use of several HSM configured in cascade. Up to 8 (read-out from VSB) or 20 (read-out from VME) memories can be cascaded, providing a total memory capacity of 8 or 20 Mbytes maximum.

When many HSMs are put in cascade, the windows of the memories are placed into a contiguous address space in VSB as long as the HSM are placed in a contiguous order in the crate. This is not the case for the VME addressing space. Refer to Memory addressing from VME or VSB for more information.

Note This mode is simply a memory size extension.

Take the case of HSM A and HSM B connected in "Cascaded Mode" according to figure 5.6. The HSM A must be first initialized and started. At that time, the acquisition can begin. The HSM B is then immediately initialized and started but remains in "Acq Request" state until the HSM A is full. As soon as the HSM A is full, the HSM B continues the acquisition. When the HSM B is full, the acquisition is stopped. This principle can be easily extended to the use of several HSMs connected in Cascade.

Cable Connection for HSMs in Cascade

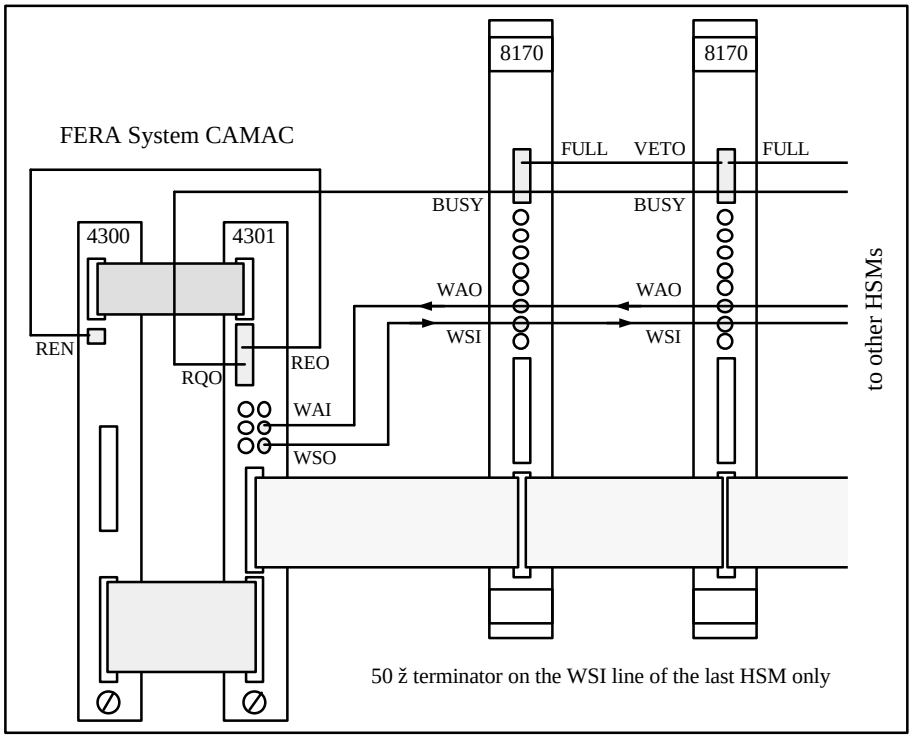


Fig 5.6

5.5.4 Acquisition Mode with two HSMs in Flip-Flop

As seen previously, the HSM allows us to perform readout either from VME or VSB during an acquisition on the FERA bus.

This readout possibility is interesting but has the disadvantage of considerably slowing down the acquisition speed because the memory access must be shared between the FERA and the VME / VSB busses.

A solution avoiding loss of speed is to use two HSMs connected to the FERA bus in Flip-Flop mode. The acquisition is performed on one HSM, while the readout is done on the other one via VME or VSB.

Take the case of HSM A and HSM B connected in Flip-Flop mode according to figure 5.7. The HSM A is initialized and started. At that time, the acquisition can begin. The HSM B is then immediately initialized and started but remains in "Acq Request" state until HSM A is full. As soon as the acquisition is running, the ACQ ON output signal of the HSM A is asserted. This signal is connected to the CARRY input of the HSM B. When the HSM A is full, its ACQ ON output becomes de-asserted, which allows the HSM B to start the acquisition. The HSM B then asserts its ACQ ON output, which disables acquisition for HSM A.

From the FERA point of view, nothing has changed and the transfer continues normally. While the HSM B is running, the HSM A's memory can be emptied through VME or VSB without loss of speed in the acquisition. When the HSM A is emptied, it is re-initialized and re-started, but remains in "Acq Request" state until HSM B is full. When the HSM B is full, the HSM A continues the acquisition, without loss of data.

Cable Connection for HSMs in Flip-Flop

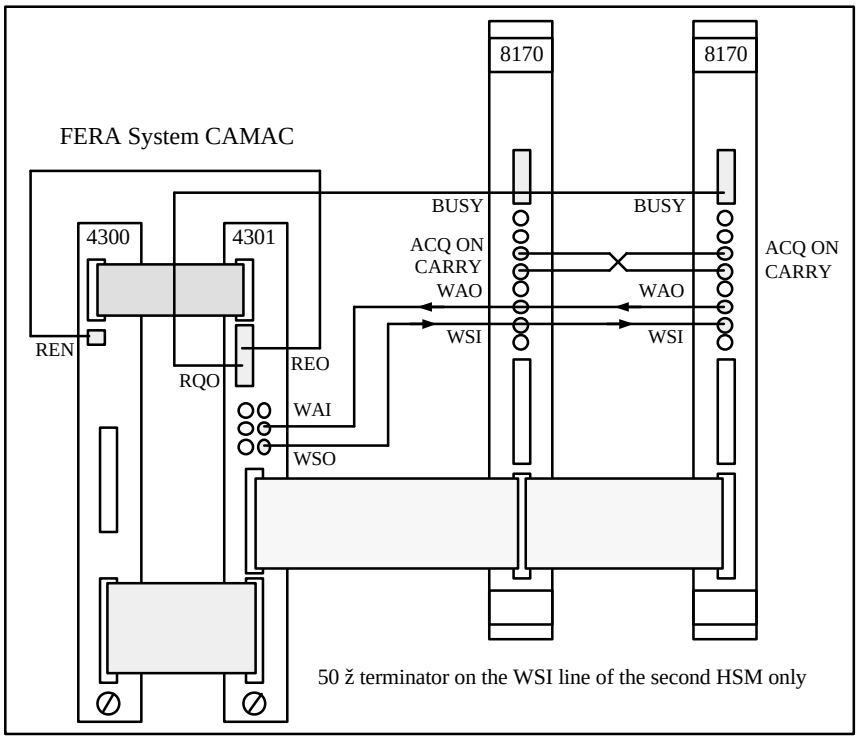


Fig 5.7

6. Installation Notes

Your HSM 8170 is provided with the following jumpers' factory setting:

Function	Jumper N°	Factory setting	Meaning
Terminator on WSI	J03	installed	WSI line terminated
Acquisition mode	J04	32-bit	
VME address offset	J05	installed	<A24> = "0"
	J06	installed	<A25> = "0"
	J07	installed	<A26> = "0"
	J08	installed	<A27> = "0"
	J09	installed	<A28> = "0"
Status / ID byte	J10	installed	<D05> Status / ID = "0"
	J11	installed	<D06> Status / ID = "0"
	J12	installed	<D07> Status / ID = "0"

6.1. Installation in the VME / VSB system

6.1.1 Setting the VME slave port

The HSM 8170 occupies a 2 Mbytes window in the VME extended addressing space (A32). The VME slave port base address is set with jumpers J05 to J09.

Jumpers	J05	J06	J07	J08	J09
Address bit	<A24>	<A25>	<A26>	<A27>	<A28>

It is important to note that a jumper "ON" (installed) sets the corresponding address bit in logical level "0".

Address bit <A20> is used for selecting between memory space and register space.

<A20> = 0 -> memory addressing (1 Mbyte)
<A20> = 1 -> register addressing (4 x 32-bit register)

Example

J05	:	on	->	<A24> = 0	0xF40xxxxx: address for memory access
J06	:	on	->	<A25> = 0	0xF410000x: address for register access
J07	:	off	->	<A26> = 1	
J08	:	on	->	<A27> = 0	
J09	:	off	->	<A28> = 1	

6.1.2 Setting the VSB slave port

The HSM 8170 occupies a 1 Mbyte window on the VSB (only the memory is mapped). The VSB slave port base address is defined by the geographical position of the HSM 8170 on the VSBbus.

Position on VSBbus	VSB address memory
1	VSB master
2	0x001xxxxx
3	0x002xxxxx
4	0x003xxxxx
5	0x004xxxxx
6	0x005xxxxx

6.2 Single Memory

This application allows us to have an acquisition system able to stock up to 256 Kwords of 32-bit. With such a system, it is possible to simultaneously execute data processing by the way of the VME or VSBbus during acquisition. The acquisition speed is not slowed down during VME and VSB access because requests coming from the acquisition port are handled with the highest priority level. This makes the access time on the VME and VSB ports vary according to the rate of data going into the acquisition port. The access time to the memory will be a minimum between acquisition bursts and will increase in relation to the transfer rate on the acquisition port.

6.2.1 Installation

After having configured the jumpers and placed the module in the crate, the HSM 8170 is to be interconnected to the associated equipment as explained in § 6.4, taking care to limit the length of the connections.

Warning The length of the cable bringing the data on the input port must be equal to the cable transmitting the WSI signal.

6.2.2 Initialization

In order to start an acquisition, commands have to be given in the following order:

Step 1	*SysReset*	This command must only be given once at the initialization of the VME system. After this command, all functions are disabled.
Step 2	*Load Word Counter*	Selection of the memory size reserved for acquisition. The value given in this counter corresponds to the maximum number of words to be accepted. This number corresponds to a configuration of the FAST PORT to either 16 or 32-bit words.
Step 3	*Load Address Pointer*	Selection of the address where the first word coming from the fast port is written. This pointer is incremented once or twice after each transfer in the memory. Depending on the configuration on the FAST PORT, this address corresponds to a 16 or 32-bit word.
Step 4	*Load Control Register*	Selection of the Memory Overflow limits, authorizes interrupt sources as well as acquisition on the FAST PORT.

After this last command, the module is able to accept transfer requests made by way of the WSI input. The acquisition will end either when the MEM OVERFLOW or MEM FULL internal conditions appear or when the STOP ACQ external conditions are received. At the end of the acquisition, after the reading of data by one of the VME or VSB ports, it will be necessary to re-initialize the WORD COUNTER then the ADDRESS POINTER. The acquisition will then be able to continue either by re-initializing the CONTROL Register, or by applying a pulse on the START ACQ input.

6.3 Checking the HSM 8170

Once the HSM 8170's base address has been set, the register and memory space can be accessed in A32 D32 or A32 D16 mode through VME. You now should be able to read or write in the internal memory.

The four internal registers are the following:

Register base address + 0x0	->	Interrupt & Status register	(STR)
Register base address + 0x4	->	Control register	(CTR)
Register base address + 0x8	->	Address pointer register	(APR)
Register base address + 0xC	->	Word counter register	(WCR)

Although some bits in STR and CTR are Write only, these registers can be overwritten. Loading them with data = 0, you should then read (in 32-bit mode):

STR	->	0xFFFF'00F4
CTR	->	0xFFFF'0000
APR	->	0xFFF8'0000
WCR	->	0xFFF0'0000

By writing 0xFFFF in the CTR register, the following LEDs should be on:

- ECL PORT
- OVERFLOW
- MEM FULL
- ENBL ACQ

and you should then read:

CTR -> 0xFFFF'FF16

By writing again 0x0 in the CTR, all the LEDs should go off.

6.4 An Example of Minimal Configuration

Hardware Connections

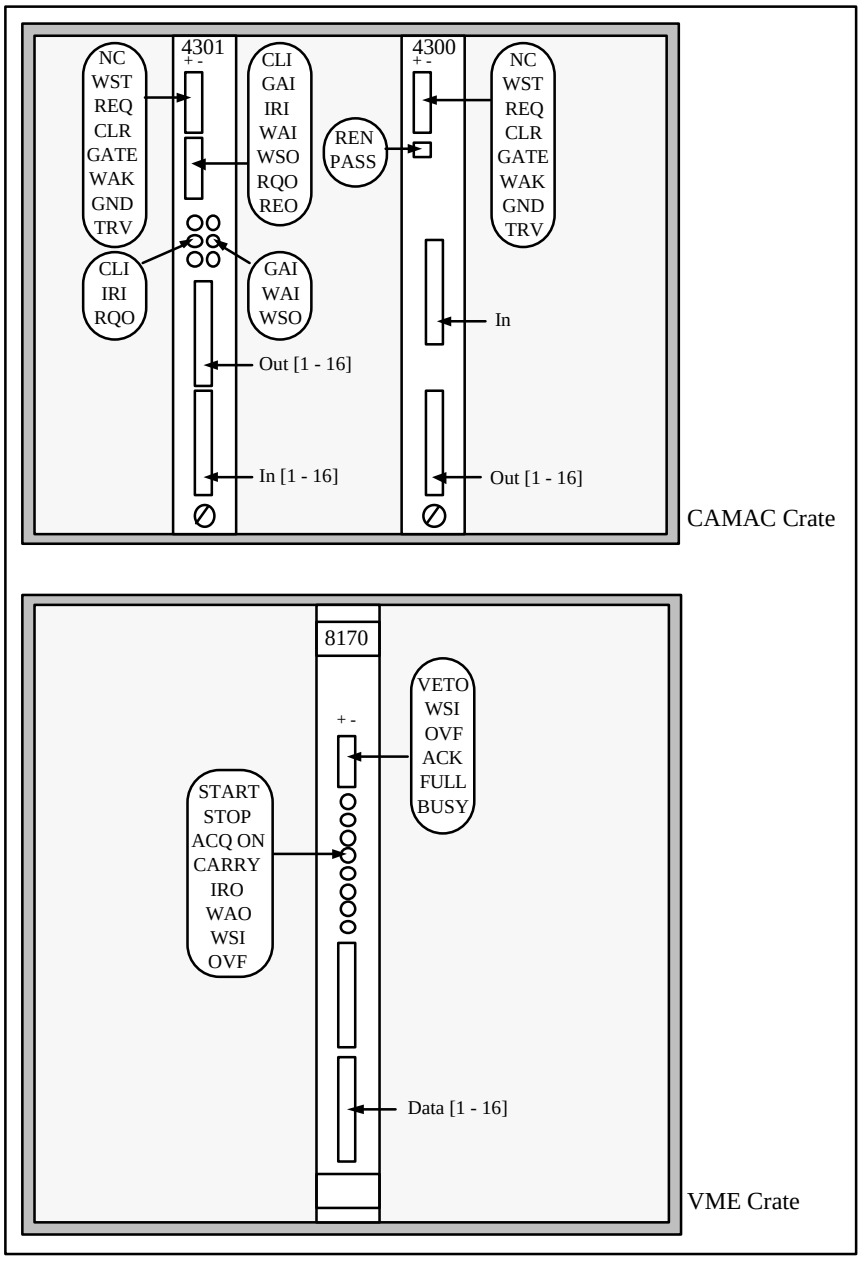


Fig 6.1

•	Connect	[4301]	NC	to	[4300]	NC
			WST			WST
			REQ			REQ
			CLR			CLR
			GATE			GATE
			WAK			WAK
			GND			GND
			TRV			TRV
•	Connect	[4301]	In [1 - 16]	to	[4300]	Out [1 - 16]
•	Connect	[4301]	REO	to	[4300]	REN
•	Connect	[4301]	RQO	to	[8170]	BUSY
•	Connect	[4301]	WAI	to	[8170]	WAO
•	Connect	[4301]	WSO	to	[8170]	WSI
•	Connect	[4301]	Out [1 - 16]	to	[8170]	Data [1 - 16]

Software Initialization	
4301 clearing	NAF = N(slot 4301) A(0) F(9)
4300 Initialization	NAF = N(slot 4300) A(0) F(16) Value = 4400 NAF = N(slot 4300) A(0) F(0) NAF = N(slot 4300) A(0) F(9)
HSM Initialization	HSM addr = 0x0 HSM Word Count = 0x10
Start of the acquisition	HSM Control |= 0x1000 (syntax C)
Read-Out CAMAC	NAF = N(slot 4300) A(0) F(9)
Test acquisition generation	NAF = N(slot 4300) A(0) F(25) Value = 0x0
Waiting for the end of the Read-out CAMAC	Polling on the LAM of the 4300
Stop of the acquisition	HSM Control & = ~0x1000 (syntax C)

6.5 Front panel

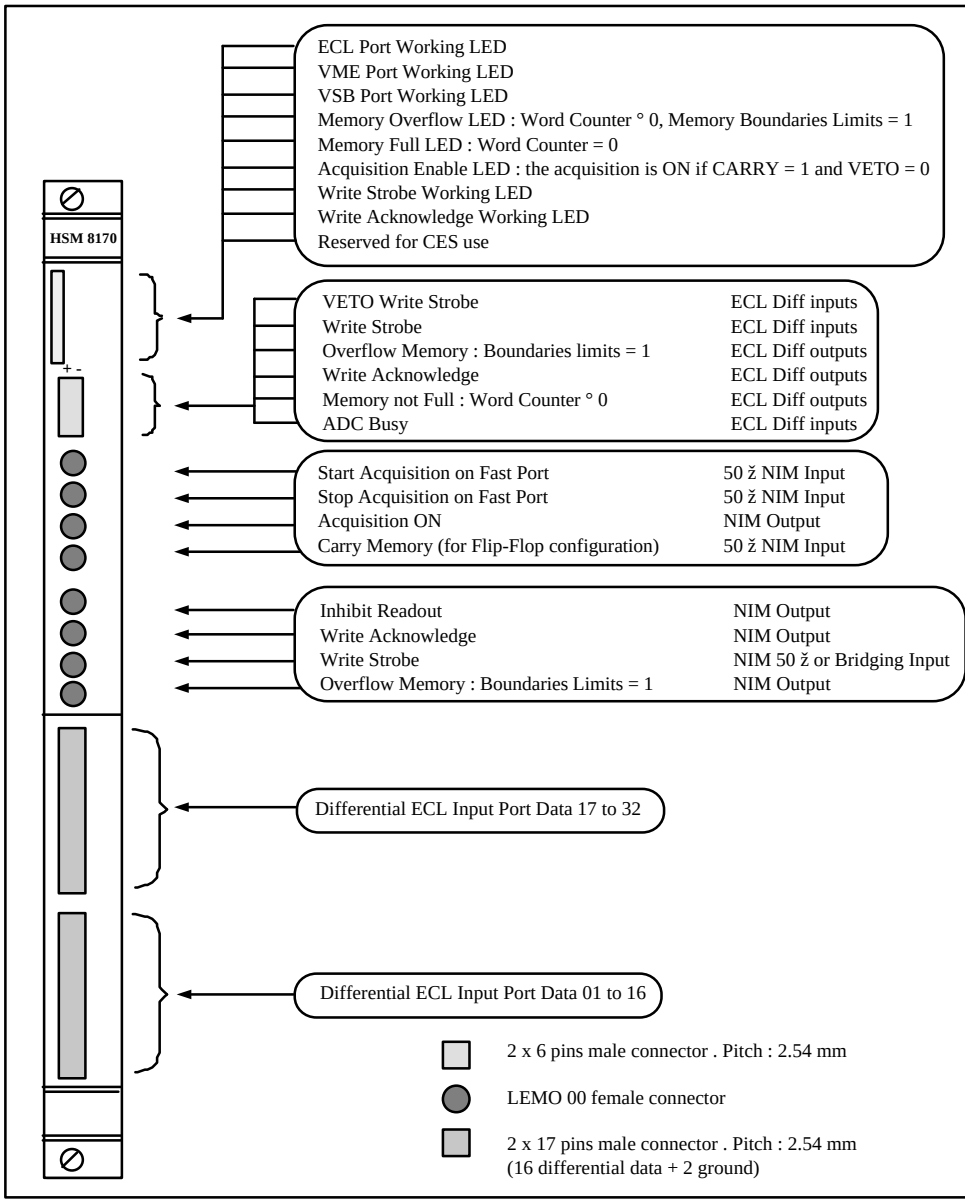


Fig 6.2

LED Display

ECL PORT When lit, this LED indicates that the arbitration logic authorizes the access of the FAST PORT to the memory for write operations. This port is permanently selected when the acquisition is enabled and when there are no transfer requests on the two other ports.

VME PORT When lit, this LED indicates that the arbitration logic authorizes the access of the VME PORT to the memory for read / write operations.

VSB PORT When lit, this LED indicates that the arbitration logic authorizes the access of the VSB PORT to the memory for read / write operations.

OVERFLOW When lit, this LED indicates that the border limits of the memory fixed in the control register have been reached and consequently the acquisition will end when the BUSY input returns to 0.

MEM FULL When lit, this LED indicates that the memory capacity which is reserved for acquisition is full and that the acquisition is either disabled or will be disabled at the next transfer.

ENBL ACQ When lit, this LED indicates that the acquisition is enabled on the FAST PORT. The status of this LED reflects that of the flip-flop which controls acquisition. Its status depends on the different variables internally or externally.

WSI Write Strobe Input. When lit, this LED indicates that the transfer requests are carried out from the acquisition port. Usually, the WSI pulses are so short that it is not possible to see the LED flashing. If the LED is ON, it means that the WSI is stuck, waiting for the corresponding WAO (abnormal situation, see diagrams below).

WAO Write Acknowledge Output. When lit, this LED indicates that the transfer requests are accepted from the acquisition port. Usually, the WAO pulses are so short that it is not possible to see the LED flashing. If the LED is ON, it means that the WAO is stuck, waiting for the corresponding WSI (abnormal situation, see diagrams below).

TERMIN This LED is reserved for CES use (always lit).

WSI and WAO timings

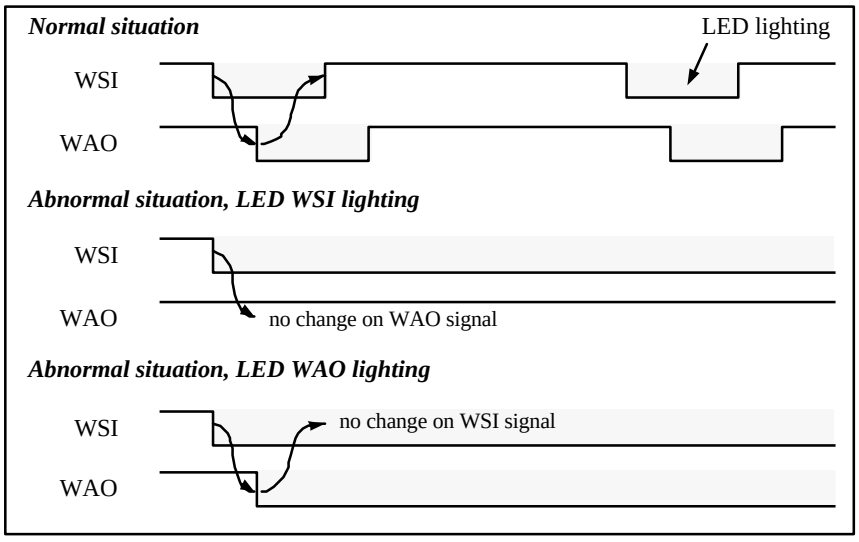


Fig 6.3

6.6 Jumpers Location

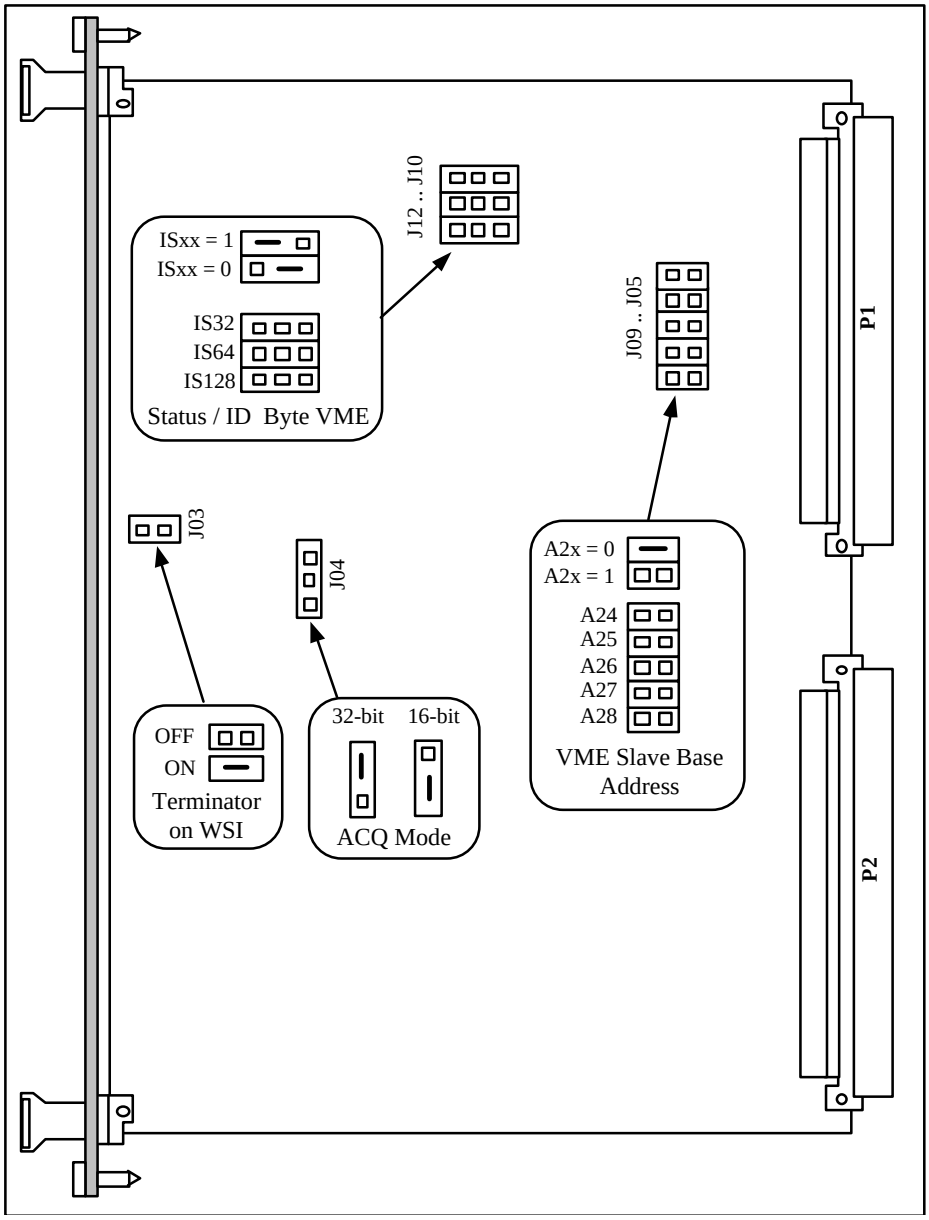
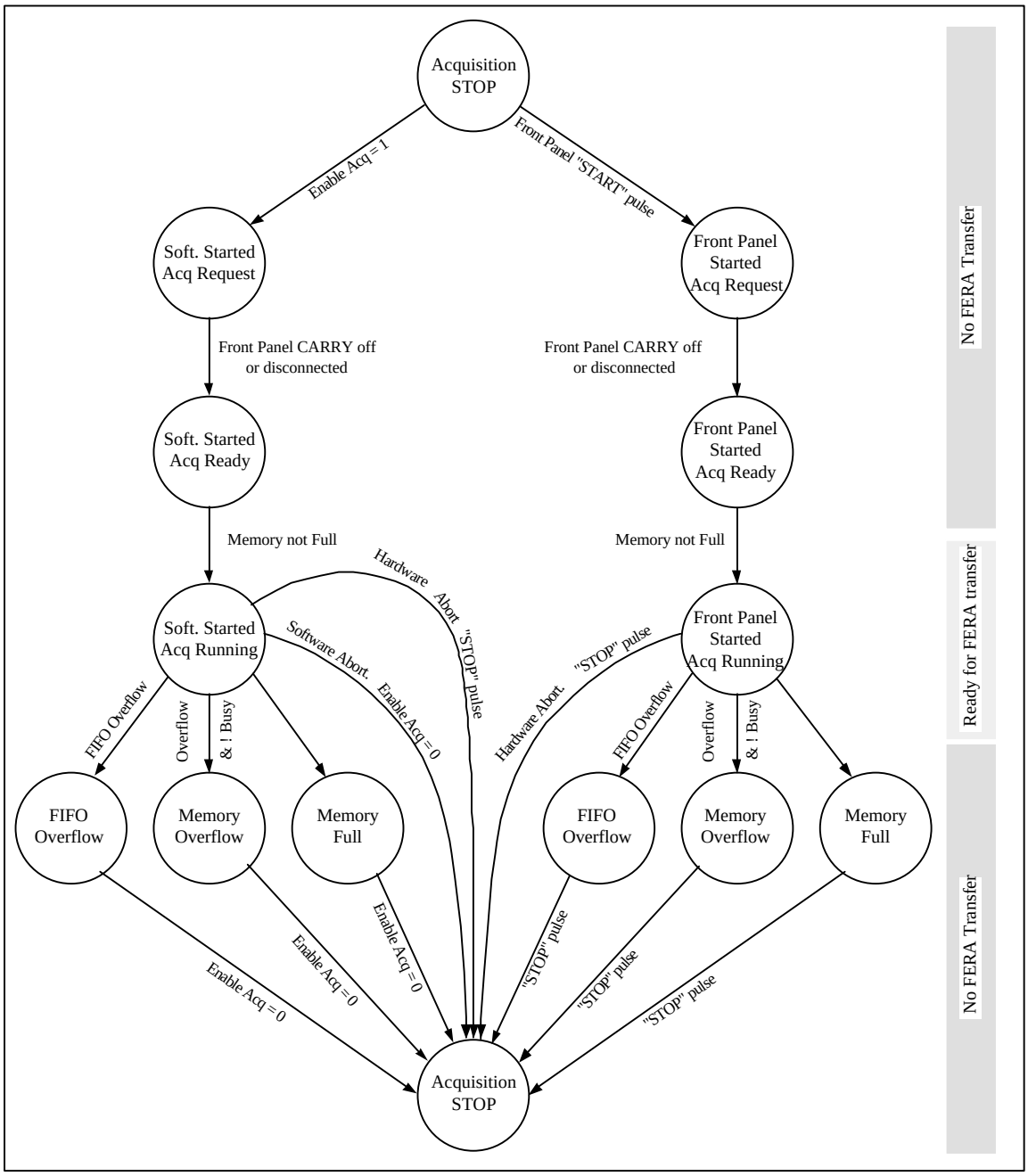


Fig 6.4

6.7 HSM Acquisition State Diagram

Fig 6.5



6.8 Differences between the old and the new HSMs

CES has made some modifications on the HSM 8170. The new version of the HSM 8170 is routed on the Printed Circuit Board referenced 239.3 or has been implemented in case of an up-grade on the old modules, referenced 239.2.

6.8.1 Hardware Differences

FERA Busy

On the old version of the HSM, the data transfer on the FERA bus was possible, even if the FERA Busy line was not connected. On the new version, the FERA Busy must imperatively be connected.

WSI ECL Line Polarity

On the old version, the polarity of the WSI ECL Line was incorrect. The user had to connect WSO+ of the FERA equipment to WSI- of the HSM and WSO- (FERA) to WSI+ (HSM) for a normal operation. On the new version, the polarity is correct.

Front Panel Data Bus

On the old version, the FERA data bus could accept TTL or ECL data. On the new version, only ECL data are allowed on the front panel data bus. The jumpers (J01 and J02) which allowed the selection of the TTL or ECL data have been removed.

VSB Geographical Addressing

On the old version, three jumpers (J13 to J15) allowed the modification of the VSB geographical addressing. On the new version, the VSB geographical addressing can no more be modified and is only determined by the position of the module in the VSB backplane. The jumpers have been removed.

6.8.2 Functionality Differences

Overflow

On the new version of the HSM, the overflow condition does not stop the acquisition as long as the FERA Busy line is active. The HSM accepts the data of the block. At the end of the transfer, as soon as the FERA Busy line becomes inactive, the overflow condition inhibits any new transfer. The HSM must then be re-initialized before initiating a new transfer. The interrupt, associated to an overflow condition, is generated as soon as the overflow condition is present and the FERA Busy line is deactivated.

Acquisition Control

On the new HSM, the control of the acquisition differs a little from that of the old version. The Start and Stop conditions are not exactly identical. Please refer to the Fast Port chapter for more information.

7. Software Libraries

7.1 hsmlib.c module

```
/*********************************************************/
/*                                                       */
/*               LIBRARY FOR THE HSM8170                 */
/*                                                       */
/*********************************************************/
/*                                                       */
/*               MODULE: hsmlib.c                        */
/*               VERSION:1.00                            */
/*               AUTHOR: J-F GILOT                       */
/*               DATE: 06/01/89                          */
/*                                                       */
/*    Copyright 1989, CES Creative Electronic System SA  */
/*    70 route du Pont-Butin, CH-1213 Petit-Lancy, Geneva*/
/*               All Rights Reserved                     */
/*                                                       */
/*********************************************************/
/*                                                       */
/*       This library contains a set of 'C ' functions   */
/*    declared in the include file "hsmlib.h". The       */
/*    structures used are declared in the include file   */
/*    "hsmreg.h". These functions allows the user to     */
/*    initialize and manage the triple ported High       */
/*    Speed Memory (HSM8170) from C.E.S.                 */
/*                                                       */
/*********************************************************/

#include "\usr\include\hsm\hsmreg.h"
#include "\usr\include\hsm\hsmlib.h"

/*********************************************************/
/*       HsmVmeMap(p_addr):                              */
/*       ------------------                              */
/*            This function maps the HSM8170 VME         */
/*       physical address to the processor virtual       */
/*       address space. It must be called at least       */
/*       ones before any other function call. This       */
/*       function is written for the FIC8230 and         */
/*       must be adapted to the processor used.          */
/*                                                       */
/*            input : p_addr =  board physical address   */
/*                              on VMEbus.               */
/*            output: hsm = virtual address used by the  */
/*                          processor to access the board*/
/*********************************************************/

long *HsmVmeMap(p_addr)

long p_addr;

{

      hsm = (struct hsm *)(0x80000000 + p_addr);
      return(hsm);
}
```

```
/*********************************************************/
/*                                                       */
/*       HsmRegMap():                                    */
/*       ------------                                    */
/*            This function maps the HSM8170 registers   */
/*       somewhere in the processor memory. It must be   */
/*       called at least ones before any manipulation    */
/*       of the declared bit field structures. At the    */
/*       end of every manipulation don't forget to copy  */
/*       the result in the physical register.            */
/*                                                       */
/*            input : none                               */
/*            output: 0                                  */
/*                                                       */
/*********************************************************/

long HsmRegMap()

{

      hsm_intr = (struct hsm_intr *)((long)(&hsm_reg[0));
      hsm_ctrl = (struct hsm_ctrl *)((long)(&hsm_reg[1));

      return(0);

}

/*********************************************************/
/*                                                       */
/*       HsmRegCpy(fct);                                 */
/*       ---------------                                 */
/*            Copy HSM8170 registers from (write) or     */
/*    to (read) the processor local memory (reserved     */
/*    with a call to 'HsmRegMap()') were they are        */
/*    manipulated.                                       */
/*                                                       */
/*            input: fct = 0 -> read registers           */
/*                         1 -> write registers          */
/*            output:  0 -> success                      */
/*                    -1 -> input code not defined       */
/*                                                       */
/*********************************************************/

long HsmRegCpy(fct)

long fct;

{

   switch(fct)
   {
      case 0:
         *hsm_intr = hsm->intr;       /* read interrupt register */
         *hsm_ctrl = hsm->ctrl;       /* read control register */
         break;
      case 1:
         hsm->intr = *hsm_intr;       /* write interrupt register */
         hsm->ctrl = *hsm_ctrl;       /* write control register */
         break;
      default:
         return(-1);                  /* bad function code */
   }
   return(0);
}
```

```
/*********************************************************/
/*                                                       */
/*       HsmReset():                                     */
/*       ----------                                      */
/*            This function resets the HSM8170 in a      */
/*    known state. Memory overflow boundary, memory      */
/*    pointer, memory counter and interrupt level are    */
/*    set to 0. Any interrupt pending is cleared and     */
/*    interrupt sources are disabled. Data Acquisition   */
/*    on fast port is also disabled. All locations in    */
/*    memory are cleared.                                */
/*                                                       */
/*            input : none                               */
/*            output: 0                                  */
/*                                                       */
/*********************************************************/

long HsmReset()

{

      long dat, len;

              /* Reset Registers */
      hsm->mem_ptr = 0x0;    /* Write 0 in Memory Pointer Register */
      hsm->wrd_cnt = 0x0;    /* Write 0 in Word Counter Register */
      hsm->ctrl = 0x0;       /* Clear Control Register */
      hsm->intr = 0x0;       /* Clear Interrupt Register */

              /* Reset any pending Interrupt Sources */
      hsm->ctrl = 0xf00;     /* Enable the four interrupt sources */
      for(dat=4;dat;dat--)
      {
        HsmIntClr();         /* Clear Interrupt Source */
      }
      hsm->ctrl = 0x0;       /* Disable interrupt sources */

              /* Clear Memory */
      dat = 0; len = MEM_LEN;
      HsmMemFil(dat, &hsm->mem[0, len);

      return(0);

}
```

```
/*********************************************************/
/*                                                       */
/*       HsmInit(iscr, ilev, ovf, ptr, cnt, usf):        */
/*       ----------------------------------------        */
/*            This function first resets the HSM8170.    */
/*    It then sets the registers according to the input  */
/*    parameters.                                        */
/*                                                       */
/*            input : isrc = interrupt sources to enable */
/*                    ilev = interrupt request level     */
/*                    ovf = memory overflow boundary     */
/*                    ptr = memory pointer               */
/*                    cnt = word counter                 */
/*                    uflg = user flag                   */
/*            output: 0                                  */
/*                                                       */
/*********************************************************/

long HsmInit(isrc, ilev, ovf, ptr, cnt, uflg)

long isrc, ilev, ovf, ptr, cnt, uflg;

{

      long fct=1;                /* Set function code = Over-Write */

      HsmReset();                /* Reset HSM8170 */

      *hsm_intr = 0x0;           /* Clear storage for interrupt register */
      *hsm_ctrl = 0x0;           /* Clear storage for control register */

      hsm->wrd_cnt = cnt;        /* Over-Write Word Counter */
      hsm->mem_ptr = ptr;        /* Over-Write Memory pointer */
      hsm_ctrl->isc = isrc;      /* Enable Interrupt Sources */
      hsm_intr->lev = ilev;      /* Set Interrupt Level */
      hsm_intr->usf = uflg;      /* Set User Flags */
      hsm_ctrl->ovf = ovf >> 9;  /* Set Memory Overflow Boundary */

      HsmRegCpy(fct);            /* Over-Write HSM8170 registers */

      return(0);

}

/*********************************************************/
/*                                                       */
/*       HsmAcqSts():                                    */
/*       ------------                                    */
/*            This function read the three status bit    */
/*    of the acquisition port.                           */
/*                                                       */
/*            input : none                               */
/*            output: fps = fast port status             */
/*                                                       */
/*********************************************************/

long HsmAcqSts()

{

      *hsm_ctrl = hsm->ctrl;     /* Read Control Register */

      return(hsm_ctrl->fps);     /* Return Fast Port Status */

}
```

```
/*********************************************************/
/*                                                       */
/*       HsmAcqEna(fct):                                 */
/*       ---------------                                 */
/*            This function enable or disable data       */
/*    acquisition on the fast port.                      */
/*                                                       */
/*            input : fct =-1 -> disable acquisition port  */
/*                          1 -> enable acquisition  port  */
/*                          0 -> read acquisition bit    */
/*            output: acq = acquisition bit value        */
/*                                                       */
/*********************************************************/

long HsmAcqEna(fct)

long fct;

{

      *hsm_ctrl = hsm->ctrl;           /* Read Control Register */

      switch(fct)
      {
         case 0 : break;                          /* Read Acquisition Bit */
         case 1 : hsm_ctrl->acq = 1; break;       /* Set Acquisition Bit */
         case -1: hsm_ctrl->acq = 0; break;       /* Clear Acquisition Bit */
         default: return(-1);                     /* Bad Function Code */
      }

      hsm->ctrl = *hsm_ctrl;           /* Write Control Register */
      *hsm_ctrl = hsm->ctrl;           /* Read Back Control Register */

      return(hsm_ctrl->acq);           /* Return acquisition bit value */

}

/*********************************************************/
/*                                                       */
/*       HsmIntSrc():                                    */
/*       ------------                                    */
/*            This function returns the interrupt        */
/*    source status.                                     */
/*                                                       */
/*            input : none                               */
/*            output: interrupt source status            */
/*                                                       */
/*********************************************************/

long HsmIntSrc()

{

      *hsm_ctrl = hsm->ctrl;      /* Read Control Register */

      return(hsm_ctrl->src);      /* Return Interrupt Source Status */

}
```

```
/*********************************************************/
/*                                                       */
/*       HsmIntVec():                                    */
/*       ------------                                    */
/*            This function returns the interrupt        */
/*    vector of the last interrupting source.            */
/*                                                       */
/*            input : none                               */
/*            output: interrupt vector                   */
/*                                                       */
/*********************************************************/

long HsmIntVec()

{

      *hsm_ctrl = hsm->ctrl;      /* Read Control Register */

      return(hsm_ctrl->vec);      /* Return Interrupt Source Status */

}

/*********************************************************/
/*                                                       */
/*       HsmIntLev(fct, ilev):                           */
/*       ---------------------                           */
/*            This function over-write and return the    */
/*    VME interrupt request level.                       */
/*                                                       */
/*            input : fct = 0 -> Read Level              */
/*                          1 -> Over-Write Level        */
/*                    ilev = Interrupt Level             */
/*            output: actual interrupt request level     */
/*                                                       */
/*                                                       */
/*                                                       */
/*********************************************************/

long HsmIntLev(fct, ilev)

long fct, ilev;

{
      switch(fct)
      {
         case 0 : break;                          /* Read Level */
         case 1 : hsm_intr->lev = ilev & 0x7;     /* Over-Write Level */
                  break;
         default: return(-1);                     /* Bad Function Code */
      }
      return(hsm_intr->lev);                      /* Return Level */
}
```

```
/*********************************************************/
/*                                                       */
/*       HsmIntEna(fct,isc):                             */
/*       -------------------                             */
/*            This function enable or disable interrupt  */
/*    sources by setting bits in source control word.    */
/*                  input : fct = 0 -> read source control*/
/*                                1 -> Over-Write isc    */
/*                                2 -> Or-Write isc      */
/*                                3 -> And-Write isc     */
/*                          isc = source control bits    */
/*                  output: isc = new source control bits*/
/*                                                       */
/*********************************************************/

long HsmIntEna(fct,isc)

long fct,isc;

{
      *hsm_ctrl = hsm->ctrl;           /* Read Control Register */

      switch(fct)
      {
         case 0 : return(hsm_ctrl->isc);
                  break;                          /* Return Source Control */
         case 1 : hsm_ctrl->isc = isc;
                  break;                          /* Over-Write isc */
         case 2 : hsm_ctrl->isc = hsm_ctrl->isc | isc;
                  break;                          /* Or-Write isc */
         case 3 : hsm_ctrl->isc = hsm_ctrl->isc & isc;
                  break;                          /* And-Write isc */
         default: return(-1);                     /* Bad Function Code */
      }

      hsm->ctrl = *hsm_ctrl;           /* Write Control Register */
      *hsm_ctrl = hsm->ctrl;           /* Read Back Control Register */

      return(hsm_ctrl->isc);           /* Return Interrupt Source Control */

}

/*********************************************************/
/*                                                       */
/*       HsmIntClr():                                    */
/*       ------------                                    */
/*            This function resets the highest priority  */
/*    pending interrupt.                                 */
/*                                                       */
/*            input : none                               */
/*            output: 0                                  */
/*                                                       */
/*********************************************************/

long HsmIntClr()

{
      hsm->intr = hsm->intr | 0x800;   /* Set Interrupt pending bit */
      hsm->intr = hsm->intr & 0xf7ff;  /* Clear Interrupt Pending bit */
      return(0);
}
```

```
/*********************************************************/
/*                                                       */
/*       HsmMemOvf(fct, ovf, cpy):                       */
/*       -------------------------                       */
/*            This function sets or read-back the        */
/*    memory overflow boundary. If the update flag       */
/*    is set, register value is copied in local          */
/*    memory, then changed and written back.             */
/*                                                       */
/*            input : fct = 0 -> read overflow boundary  */
/*                          1 -> set overflow boundary   */
/*                    ovf = overflow boundary (in words) */
/*                          values: 0x200, 0x400,.. 0xe00*/
/*                    cpy = 0 -> no update of register   */
/*                          1 -> update register         */
/*            output: overlow boundary value             */
/*                                                       */
/*********************************************************/

long HsmMemOvf(fct, ovf, cpy)

long fct, ovf, cpy;

{

                /* if update flag -> read control register */
      if(cpy == 1) *hsm_ctrl = hsm->ctrl;

                /* if read function -> return boundary value */
      if(fct == 0) return(hsm_ctrl->ovf << 9);

                /* test if boundary value is allowed */
      if((ovf & 0xfffff1ff) != 0) return(-1);

                /* if write function -> set boundary value */
      switch(fct)
      {
         case 1 : hsm_ctrl->ovf = ovf >> 9;      /* Over-Write Boundary */
                  break;
         default: return(-1);                    /* Bad Function Code */
      }

                /* if update flag -> write control register */
      if(cpy == 1) hsm->ctrl = *hsm_ctrl;

                /* return boundary value */
      return(hsm_ctrl->ovf << 9);

}
```

```
/*********************************************************/
/*                                                       */
/*       HsmMemPtr(fct,ptr):                             */
/*       -------------------                             */
/*            This function reads and if needed over-    */
/*    writes the memory pointer.                         */
/*                                                       */
/*            input : fct = 0 -> read memory pointer     */
/*                          1 -> over-write memory pointer */
/*            output: actual memory pointer              */
/*                                                       */
/*********************************************************/

long HsmMemPtr(fct, ptr)

long fct, ptr;

{
      switch(fct)
      {
         case 0 : break;                          /* Read Pointer */
         case 1 : hsm->mem_ptr = ptr & 0x7ffff;  /* Over-Write Pointer */
                  break;
         default: return(-1);                     /* Bad Function Code */
      }
      return(hsm->mem_ptr & 0x7ffff);             /* Return Memory Pointer */
}

/*********************************************************/
/*                                                       */
/*       HsmMemCnt():                                    */
/*       ------------                                    */
/*            This function reads and if needed over-    */
/*    writes the word counter.                           */
/*                                                       */
/*            input : fct = 0 -> read word counter       */
/*                          1 -> over-write word counter */
/*            output: actual word counter                */
/*                                                       */
/*********************************************************/

long HsmMemCnt(fct, cnt)

long fct, cnt;

{
      switch(fct)
      {
         case 0 : break;                          /* Read Counter */
         case 1 : hsm->wrd_cnt = cnt & 0xfffff;  /* Over-Write Counter */
                  break;
         default: return(-1);                     /* Bad Function Code */
      }
      return(hsm->wrd_cnt & 0xfffff);             /* Return Word Counter */
}
```

```
/*********************************************************/
/*                                                       */
/*       HsmMemMov(src, des, len):                       */
/*       -------------------------                       */
/*            This function copies 'len' long word from  */
/*    'src' to 'dest'. After every transfer the source   */
/*    and destination pointer are incremented            */
/*            input : src = pointer to the source start  */
/*                          address                      */
/*                    des = pointer to the destination   */
/*                          start address                */
/*                    len = number of long word to be    */
/*                          transferred                  */
/*            output: 0                                  */
/*                                                       */
/*********************************************************/

long HsmMemMov(src, des, len)

register long *src, *des, len;

{
      while(len--) *des++ = *src++;
      return(0);
}

/*********************************************************/
/*                                                       */
/*       HsmMemFil(dat, dest, len):                      */
/*       --------------------------                      */
/*            This function copies 'len' data 'dat' in   */
/*    'des'. After every transfer the destination pointer*/
/*    is incremented.                                    */
/*            input : dat = data to be transfered        */
/*                    des = pointer to the destination   */
/*                          start address                */
/*                    len = number of data to be         */
/*                          transferred                  */
/*            output: 0                                  */
/*                                                       */
/*********************************************************/

long HsmMemFil(dat, dest, len)

register long dat, *dest, len;

{
      while(len--) *dest++ = dat;
      return(0);
}
```

7.2 hsmlib.h module

```
/*********************************************************/
/*                                                       */
/*          HSM8170 LIBRARY FUNCTION  DECLARATION        */
/*                                                       */
/*********************************************************/
/*                                                       */
/*               MODULE: hsmlib.h                        */
/*               VERSION:1.00                            */
/*               AUTHOR: J-F GILOT                       */
/*               DATE: 06/05/89                          */
/*                                                       */
/*    Copyright 1989, CES Creative Electronic System SA  */
/*    70 route du Pont-Butin, CH-1213 Petit-Lancy, Geneva*/
/*               All Rights Reserved                     */
/*                                                       */
/*********************************************************/
/*                                                       */
/*               MODIFICATIONS:                          */
/*                                                       */
/*********************************************************/
/*                                                       */
/*            This file contains  the declaration of     */
/*    the functions defined in the library hsmlib.c      */
/*                                                       */
/*********************************************************/

long *HsmVmeMap();      /* Map the HSM8170 base address */
long HsmRegMap();       /* Map a location in local memory to copy registers*/
long HsmRegCpy();       /* Copy registers from or to local memory */
long HsmReset();        /* Reset HSM8170 registers in a known state */
long HsmInit();         /* Initialize HSM8170 */
long HsmAcqSts();       /* Read fast port status /
long HsmAcqEna();       /* Enable-Disable acquistion on fast port */
long HsmIntSrc();       /* Read interrupt sources state */
long HsmIntVec();       /* Read interrupt vector of last interrupt request */
long HsmIntLev();       /* Read-Write Interrupt Request Level on VME */
long HsmIntLev();       /* Enable-Disable Interrupt Requests */
long HsmIntClr();       /* Clears Interrupt Pending */
long HsmMemOvf();       /* Read-Write Memory Overflow Boundaries */
long HsmMemPtr();       /* Read-Write Memory Pointer */
long HsmMemCnt();       /* Read-Write Memory Counter */
long HsmMemMov();       /* Copy 'len' words from source to destination */
long HsmMemFil();       /* Fill 'len' words in destination with 'dat' */
```

7.3 hsmreg.h

```
/*********************************************************/
/*                                                       */
/*          HSM8170 REGISTERS AND MEMORY                 */
/*                                                       */
/*********************************************************/
/*               MODULE: hsmreg.h                        */
/*               VERSION:1.00                            */
/*               AUTHOR: J-F GILOT                       */
/*               DATE: 06/01/89                          */
/*                                                       */
/*    Copyright 1989, CES Creative Electronic System SA  */
/*    70 route du Pont-Butin, CH-1213 Petit-Lancy, Geneva*/
/*               All Rights Reserved                     */
/*********************************************************/
/*                                                       */
/*               MODIFICATIONS:                          */
/*                                                       */
/*********************************************************/
/*                                                       */
/*            This file contains  the declaration of     */
/*    a structure representing the HSM8170 internal      */
/*    configuration. In the initialization routine the   */
/*    pointer '*hsm' must be assigned to the VME base    */
/*    address chosen (by jumpers) for the module.        */
/*                                                       */
/*********************************************************/

#define MEM_LEN    0x40000   /* Memory size (in long words) */

struct hsm
{
      long mem[0x40000];     /* Memory (1Mbyte Window in VME space) */
      long intr;             /* Interrupt Status Register */
      long ctrl;             /* Control Register */
      long mem_ptr;          /* Address Pointer Register */
      long wrd_cnt;          /* Word Counter Register */
} *hsm;

struct hsm_intr              /* interrupt status register */
{
  unsigned vec : 8;                   /* Interrupt Vector */
  unsigned lev : 3;                   /* Interrupt Request Level */
  unsigned clr : 1;                   /* Clear Interrupt Flag */
  unsigned usf : 4;                   /* User Flags */
  unsigned rsv :16;                   /* Not Used */
} *hsm_intr;

struct hsm_ctrl              /* control register */
{
  unsigned src : 4;                   /* Sources of interrupt */
  unsigned fps : 3;                   /* Fast Port status */
  unsigned one : 1;                   /* Always set to 1 */
  unsigned isc : 4;                   /* Enable-Disable interrupt sources */
  unsigned acq : 1;                   /* Enable-Disable Acquisition */
  unsigned ovf : 3;                   /* Memory Overflow Boundaries */
  unsigned rsv :16;                   /* Not Used */
} *hsm_ctrl;

long hsm_reg[2];         /* two long words to manipulate the registers */
```