

Technical Information Manual

Revision n. 0

22 December 1992

MOD. V288
H.S. CAENET
VME CONTROLLER

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

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



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

TABLE OF CONTENTS

TABLE OF CONTENTS	i
1. DESCRIPTION.....	1
1.1.FUNCTIONAL DESCRIPTION.....	1
2. SPECIFICATION.....	3
2.1. PACKAGING	3
2.2. EXTERNAL COMPONENTS	3
2.3. INTERNAL COMPONENTS.....	3
2.4.POWER REQUIREMENTS	3
3. OPERATING MODES.....	5
3.1. H.S. CAENET NETWORK OPERATION	5
3.2. H.S. CAENET NODE OPERATION.....	6
3.3. THE MOD. V288 H.S. CAENET VME CONTROLLER.....	6
3.4. MOD V288 OPERATING MODES.....	7
3.4.1. RESET.....	7
3.4.2. STATUS REGISTER.....	7
3.4.3. DATA PACKET STORAGE.....	7
3.4.4. START TRANSMISSION.....	8
3.4.5. WAITING FOR THE SLAVE RESPONSE	8
3.4.6. READING THE RESPONSE	9
3.5. V288 - SLAVE COMMUNICATION SEQUENCE	10
4. VME INTERFACE.....	11
4.1. V288 ADDRESSING CAPABILITY	11
4.2. V288 DATA TRANSFER CAPABILITY	11
4.3. TRANSMIT DATA BUFFER.....	12
4.4. RECEIVE DATA BUFFER.....	12
4.5. STATUS REGISTER.....	12
4.6. TRANSMISSION REGISTER.....	13
4.7. RESET REGISTER.....	13
4.8. INTERRUPT VECTOR REGISTER.....	13
4.9. V288 INTERRUPTER CAPABILITY	13
4.10. V288 INTERRUPT LEVEL.....	13
5. C.A.E.N. PROTOCOL	15
5.1. H.S. CAENET NETWORK FOR REMOTING CONTROL.....	15
5.2. H.S. CAENET NETWORK IMPLEMENTATION: CAEN APPROACH.....	15
5.3. MASTER TO SLAVE DATA COMPOSITION	17
5.4. SLAVE TO MASTER DATA COMPOSITION	17
5.5. ERROR CODES	18
6. REFERENCES.....	19
APPENDIX A SOFTWARE BUGS AS OF SEPTEMBER 1992	A.1
A.1. H.S. CAENET OPERATION BUGS.....	A.1
APPENDIX B SOFTWARE EXAMPLES.....	B.1



1. DESCRIPTION

1.1.FUNCTIONAL DESCRIPTION

The Model V288 HIGH SPEED (H.S.)CAENET VME CONTROLLER has been designed to control an H.S. CAENET network through the VME bus. It houses an H.S. CAENET Node and a Control Logic (microprocessor based) which integrates the functions of Node controller and Network error handler.

Standard VME cycles allow the user to easily control the serial communication on the H.S. CAENET network according to the typical MASTER/SLAVE communication protocol, where the VME controller assumes the MASTER function.

It is composed of a collection of registers, for the operation control, and two memory buffers for the data packet transmitted and received, arranged in a FIFO logic 16 bit wide 256 word deep.

In the memory buffer for the received data are also stored error messages generated by the on-board Control logic when the H.S. CAENET operation has failed

The Module is an A24 D16 VME Slave; Its Base Address is programmable through dip switches located on the Board.

The module operations can be software controlled in polling mode or can be handled via interrupt facility. It houses a VME ROAK INTERRUPTER [1] that generates a VME interrupt (if enabled) as soon as the data packet (or the error message) is stored in the receive buffer.

The communication line uses a simple 50 ohm coaxial cable as physical medium.

The data transfer rate is 1 MBaud.

(A functional block diagram is shown in Fig. 1.1.)

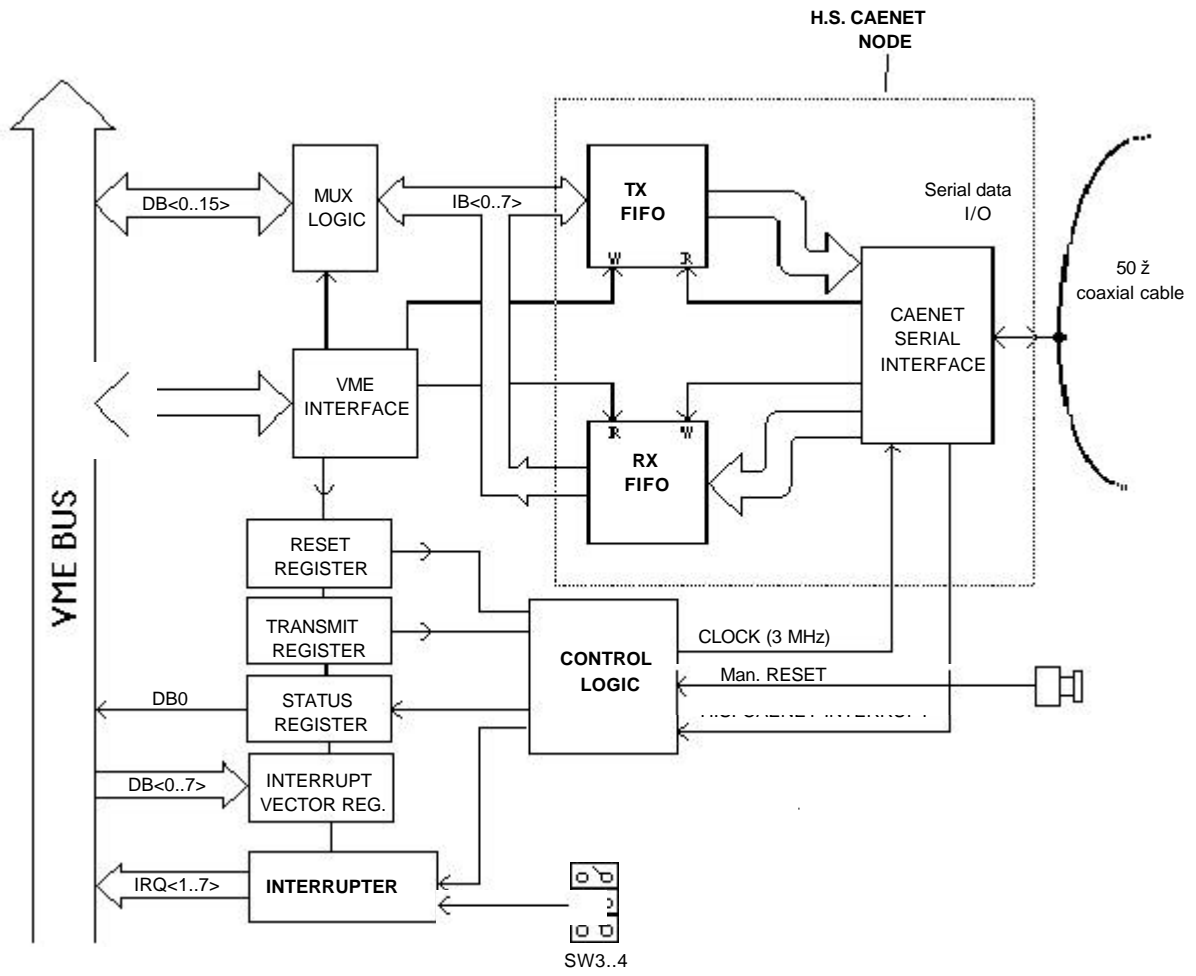


Fig 1.1 Mod. V288 block diagram

2. SPECIFICATION

2.1. PACKAGING

1-unit wide VME module.

2.2. EXTERNAL COMPONENTS

(refer to Fig. 2.1).

CONNECTORS

- No. 1 "SERIAL LINE" LEMO 00 type, 50 Ohm connector;
connector for the H.S. CAENET communication line.

LEDs

- No.1 "DATA",red LED;
is On when the H.S. CAENET Node is active.

SWITCHES

- No.1 "RESET"push button;
by pushing this button the V288 enters in restart mode; this causes the following operations:
 - the buffers are cleared;
 - every VME interrupt pending is cleared;
 - every data transfer is aborted;
 - the V288 does not accept any command.

It remains in this status for about 3 msec.

2.3. INTERNAL COMPONENTS

SWITCHES

- No.3 dip-switches SW2,SW5,SW6;
These dip-switches allow the selection of the VME Base address.
- No.2 dip-switches SW3,SW4;
These dip-switches allow the selection of the VME interrupt level.

2.4.POWER REQUIREMENTS

+ 5V 2 A

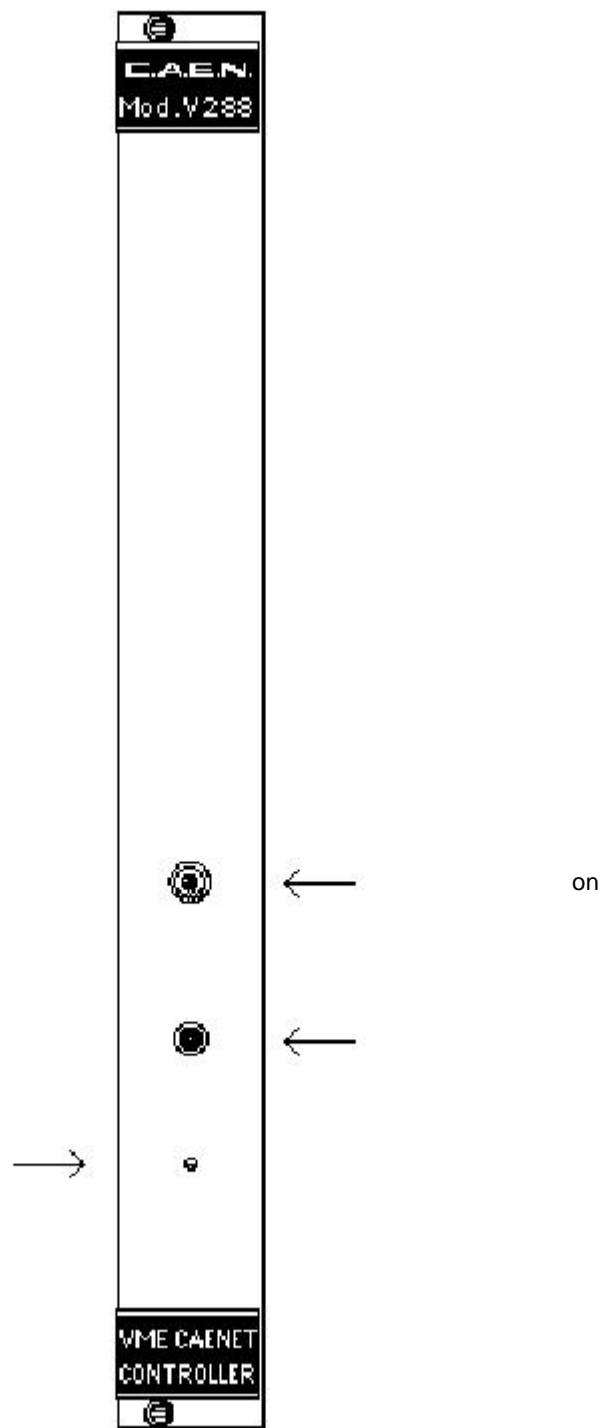


Fig 2.1 Mod. V288 Front Panel

3. OPERATING MODES

3.1. H.S. CAENET NETWORK OPERATION

H.S. CAENET Network is a send and receive half duplex system; It permits asynchronous serial transmission of data packet along simple 50 ohm coaxial cable. Several devices (H.S. CAENET Nodes) are able to share the same media to transmit and receive data.

Each Node is able to receive the serial data packet and store it automatically in the RX FIFO and to transmit the data contained in the TX FIFO. Both FIFOs are 512 byte deep.

The H.S. CAENET Node listen for clear coax before transmitting but it is not able to detect collisions on the cable; for this reason it is important to avoid line contention i.e. the Nodes should not attempt to transmit at the same time.

Usually transfers between H.S. CAENET Nodes take place according to the typical MASTER/SLAVES communication: there is a single H.S. CAENET MASTER that initiates the transmission, all the SLAVES receive the data, and only the SLAVE addressed then accesses the serial line to transmit the data requested by the MASTER.

The maximum data packet length is 512 bytes.

An external clock source provides the basic time reference in the Node: the clock frequency is three times the data transfer rate. Actually the clock frequency used in the H.S. CAENET network is 3 MHz (1 MBaud transfer rate);

The Node is seen as an 8 bit peripheral device composed of a collection of registers and two memory buffers arranged in FIFO logic. It generates an interrupt (H.S. CAENET interrupt) upon:

- the completion of a transmission of a data packet
- the reception of a data packet
- when the RX FIFO has been completely unloaded.

3.2. H.S. CAENET NODE OPERATION

The basic operation of the H.S. CAENET Node consists in 3 distinct modes:

Transmit, Receive and Restart mode.

- In the Transmit mode the Node accesses the data stored in the TX FIFO and transmits them on the cable.
- In the Receive mode the serial packet is stored in the RX FIFO.
- In Restart mode the Node does not execute any commands, all the TX and RX buffers are cleared and the H.S. CAENET interrupt is removed; it remains in this mode until it detects that the line is clear.

3.3. THE MOD. V288 H.S. CAENET VME CONTROLLER

The Model V288 H.S. CAENET VME CONTROLLER has been designed to easily control an H.S. CAENET network through the VME bus. It houses an H.S. CAENET Node and a Control logic (microprocessor based with dedicated firmware) which integrates the functions of Node controller with Network error reporting and H.S. CAENET interrupt handler.

The Control logic directly controls the on-board H.S. CAENET Node and its multiplexing logic interfaces the 16 bit VME data bus with the two 8 bit FIFOs so as they are seen by VME as 16 bit buffers. It receives commands from a VME Master in a way that few standard VME cycles allow the user to easily control the serial communication on the H.S. CAENET network; the communication is performed according to the typical MASTER/SLAVE communication protocol, where the VME Master assumes the H.S. CAENET MASTER function.

The Module is an A24 D16 VME Slave, it is composed of a collection of registers, for the operation control, and two memory buffer for the data packets transmitted and received, arranged in a FIFO logic 16 bit wide 256 word deep (see Fig.1.1)

The Status Register content indicates whether the previous H.S. CAENET Operation has been successfully performed; in particular shows when the valid data are read from the Receive Data Buffer.

In the memory buffer for the received data are also stored error messages generated by the Control logic when the H.S. CAENET operation has failed.

The module operations can be software controlled in polling mode or can be handled via interrupt facility. It houses a VME ROAK INTERRUPTER [1] that generates a VME interrupt (if enabled) as soon as the data packet (or the error message) is stored in the receive buffer.

The interrupt vector is software programmable while the interrupt level is selectable via dip switches;

The Mod. V288 registers are described in the Table 4.1.

3.4. MOD V288 OPERATING MODES

The following paragraphs describe the various operations that can be performed via VME and the use of the V288 registers to accomplish an H.S. CAENET communication.

3.4.1. RESET

It is possible to reset the Mod V288 in these ways :

- by pushing the Front Panel push-button "RESET";
- by writing via VME at the address: Base + 6 (Reset Register).

After one of these operation the V288 enters in Restart Mode; this causes the following operation:

- the buffers are cleared;
- every VME interrupt pending is cleared;
- every data transfer is aborted;
- the V288 does not accept any command.

It remains in this status for about 3 msec.

3.4.2. STATUS REGISTER

The LSB of the V288 Status Register indicates if the previous H.S. CAENET Operation handled via VME is valid or not. The RESET operation (see above) is not considered as a H. S. CAENET operation, thus the content of the Status Register is not valid after a RESET.

Status Register bit 0= 1 "No valid operation";

Status Register bit 0= 0 "Valid operation".

The Status Register is available at the VME address Base + 2.

3.4.3. DATA PACKET STORAGE

The data to be transmitted are stored in the TX FIFO by performing subsequent VME write accesses to the address Base + 0 (Transmit Data Buffer); The buffer is arranged in FIFO logic 16 bit wide.

After any writing operation the user is recommended to read the content of the Status Register. A "No valid operation" means that the Transmit Data Buffer is not available for data storage. This may occur in these cases:

- if the H.S. CAENET Node is active (it is transmitting a previous data packet or it is receiving the SLAVE response data packet);
- if the Transmit Data Buffer is full (the maximum number of 16 bit data stored is 256).

3.4.4. START TRANSMISSION

An access in writing to the VME address Base + 4 (Transmission Register) enables the Control logic to transmit on the cable the data packet stored in the Transmit Data Buffer.

The logic first check if the Buffer is empty,

- if not, the logic sets the H.S.CAENET Node in the Transmit mode and the data packet is transmitted on the cable.
- If the Buffer is empty the control logic does not activate the transmission and write an error code in the Receive data buffer (error %FFFD see Tab. 5.3).

The content of the Status register indicates if the Start Transmission command has been recognized by the Control logic; a "No valid operation" means that the H.S CAENET Node is not able to transmit data. This may happen if the H.S. CAENET Node is active (it is transmitting a previous data packet or it is receiving the SLAVE response).

If a "valid operation" is contained in the Status register the user will receive the SLAVE Response or a Control logic message within a maximum period of 500 msec (SLAVE Response Time-out).

The Response is available at the address Base + 0 (Receive Data Buffer).

3.4.5. WAITING FOR THE SLAVE RESPONSE

The Control logic waits for the SLAVE Response for about 500 msec; if no data packet is received within this period the Control logic stores an error code in the Receive Data Buffer (error %FFFF see table 5.3).

If a data packet is received from the cable within the SLAVE Response Time-out, the Control logic checks if it has the correct header: if not, clears the Receive Data Buffer and stores in it an error code (error % FFFE see table 5.3).

After these operation, the Receive Data Buffer contains valid data (an error code or the SLAVE response) for the VME Master that has initiated the H.S. CAENET communication.

At this point the Control logic enables the VME reading, and if the interrupt level selected on the dip switches is different from 0, a VME Interrupt on the corresponding IRQ line [1] is generated.

3.4.6. READING THE RESPONSE

The User after the transmission of the data packet expects a response in the Receive Data Buffer; the presence of valid data can be recognized in two different ways: In polling mode or by the use of the VME interrupt.

Polling mode:

- After the Start Transmission operation, the user reads the content of the Receive Data Buffer (address Base + 0) and successively the Status Register (address Base + 2), this two read operations must be repeated until a "Valid Operation" is contained in the Status Register; this means that the data read is the first data of the Response.

VME interrupt

- The generation of the VME interrupt means that valid data are present in the Receive Data Buffer

Then the reading of the Receive Data buffer and the Status Register must be repeated until a "No valid operation" is obtained.

3.5. V288 - SLAVE COMMUNICATION SEQUENCE

The operations previously described are summarized in the following report:

- **write the data packet in the Transmit Data Buffer**; in the packet is contained the H.S. CAENET address of the SLAVE (see Tab 5.1 for the data structure).

for each data:

- write the data in the Transmit Data Buffer
- read the Status Register :
- if Status Register ="Valid operation"
 - {
 - the data is stored in the buffer.
 - }
- else
 - {
 - error.
 - }

- **Transmit the data packet:**

- Access in write the Transmission Register
- read the Status Register
- if Status Register = "Valid Operation"
 - {
 - the V288 H.S. CAENET Node enters in the transmit mode and the data packet stored is transmitted on the cable.
 - }
- else
 - {
 - error.
 - }

- **Wait for the SLAVE response**

- if the Interrupt is enabled
 - {
 - wait for V288 interrupt
 - }
- else
 - {
 - read the Receive Data Buffer
 - read the Status Register
 - if Status Register = "No Valid Operation" discard the data and repeat the two read operation.
 - if Status Register = "Valid Operation" accept the data read: it may be the first data of the SLAVE response data packet or a Control Logic error message; go to the Read Response section
 - }

- **Read SLAVE response**

- read the Receive data buffer
- read the Status Register :
- if Status Register = "Valid Operation" accept the data read: and repeat the two read operations.
- if Status Register = "No Valid Operation" discard the data read and exit: the Receive Data Buffer is empty.

4. VME INTERFACE

4.1. V288 ADDRESSING CAPABILITY

The module works in A24 mode; this means that the module address must be specified in a field of 24 bits.

The Address Modifiers recognized by the module are

- AM = %39 : Standard user data access
- AM = %3A : Standard user program access
- AM = %3D : Standard supervisor data access
- AM = %3E : Standard supervisor program access

The module's Base address is fixed by dip switches located on the board (see Fig 4.1)
The Base address can be selected in the range:

%00 0000 <-> %FF FFF0

4.2. V288 DATA TRANSFER CAPABILITY

The registers and the buffers are accessible in D16 mode.

Table 4.1: Mod V288 Registers

NAME	TYPE	ADDRESS	FUNCTION
Transmit Data Buffer	Write only	Base Address + 00	Transmit data storage
Receive Data Buffer	Read only	Base Address + 00	Receive data storage
Status Register	Read only	Base Address + 02	After an H.S.CAENET operation has been performed, the Status Register bit 0 indicates whether the operation is valid or not 0 = Valid Operation 1 = No Valid Operation
Transmission Register.	Write only	Base Address + 04	By writing into this register the Transmit Data Buffer content is transmitted on the cable
Reset Register	Write only	Base Address + 06	Module's Reset
Interrupt Vector Register	Write only	Base Address + 08	Interrupt vector programming register

4.3. TRANSMIT DATA BUFFER

(Base Address + 0 write access)

This is the buffer which is loaded with the data packet to transmit, it is arranged in a FIFO logic 16 bit wide; (the data packet transmitted is composed of 16 bit words as shown in Tab 5.1).

4.4. RECEIVE DATA BUFFER

(Base Address + 0 read access)

This is the buffer where the H.S. CAENET Node automatically stores the data packet received from the SLAVE or, if the H.S. CAENET operation has failed, the Control Logic stores an error code. It is arranged in a FIFO logic 16 bit wide; (the data packet received is composed of 16 bit words as shown in Tab 5.2).

4.5. STATUS REGISTER

(Base Address + 2 read only)

The LSB bit of the V288 Status Register indicates if the previous H.S. CAENET Operation handled via VME is valid or not

Status Register bit 0= 1	No valid operation;
Status Register bit 0= 0	Valid operation.

The bits from 1 to 15 are unused and are read as "one".

After one of the following operations the user is recommended to read the Status Register:

- **write data in the Transmit Data buffer:** it indicates if the datum written has been stored or not in the Transmit Data Buffer; a "No valid operation" means that the Transmit Data Buffer is not available for data storage. This may happen in these cases:
 - if the H.S. CAENET Node is active (it is transmitting a previous data packet or it is receiving the SLAVE response data packet);
 - if the Transmit Data Buffer is full (the maximum number of data stored is 256).
- **write in the Transmission Register** (Start data packet transmission): it indicates if the Start Transmission command has been recognized by the Mod. V288; a "No valid operation" means that the H.S CAENET Node is not able to transmit data. This may happen if the H.S. CAENET Node is active (it is transmitting a previous data packet or it is receiving the SLAVE response);
- **read data from the Receive Data Buffer:** it indicates if the data read is valid or not.

4.6. TRANSMISSION REGISTER

(Base Address + 4 write only)

An access in writing at this location enables the V288 Control Logic to transmit on the cable the data stored in the Transmit Data Buffer. If this operation is performed with the Transmit Data Buffer empty, The Control logic stores an error message in the Receive Data Buffer (error %FFFD see Table 5.3).

4.7. RESET REGISTER

(Base address + 6, write only)

An access in writing to this location causes the V288 to enter in restart mode; this causes the following operations:

- the buffers are cleared;
- every interrupt pending is cleared;
- every data transfer is aborted;
- the V288 does not accept any command.

It remains in this status for about 3 msec.

4.8. INTERRUPT VECTOR REGISTER

(Base address + 8, write only)

The value written in this 8 bit register is the STATUS/ID that the V288 INTERRUPTER places on the VME data bus during the Interrupt Acknowledge Cycle.

4.9. V288 INTERRUPTER CAPABILITY

The V288 module houses a VME ROAK INTERRUPTER D08(o) type[1].This means that:

- it responds to 8 bit, 16 bit and 32 bit interrupt acknowledge cycles providing an 8-bit STATUS/ID on the VME data lines D00..D07.
- it removes its interrupt request when the VME Master reads the V288 STATUS/ID during the Interrupt Acknowledge Cycle (ROAK: Release On Acknowledge).

4.10. V288 INTERRUPT LEVEL

The interrupt level corresponds to the value set on the two dip-switches SW4,SW3 as described in Fig. 4.1.

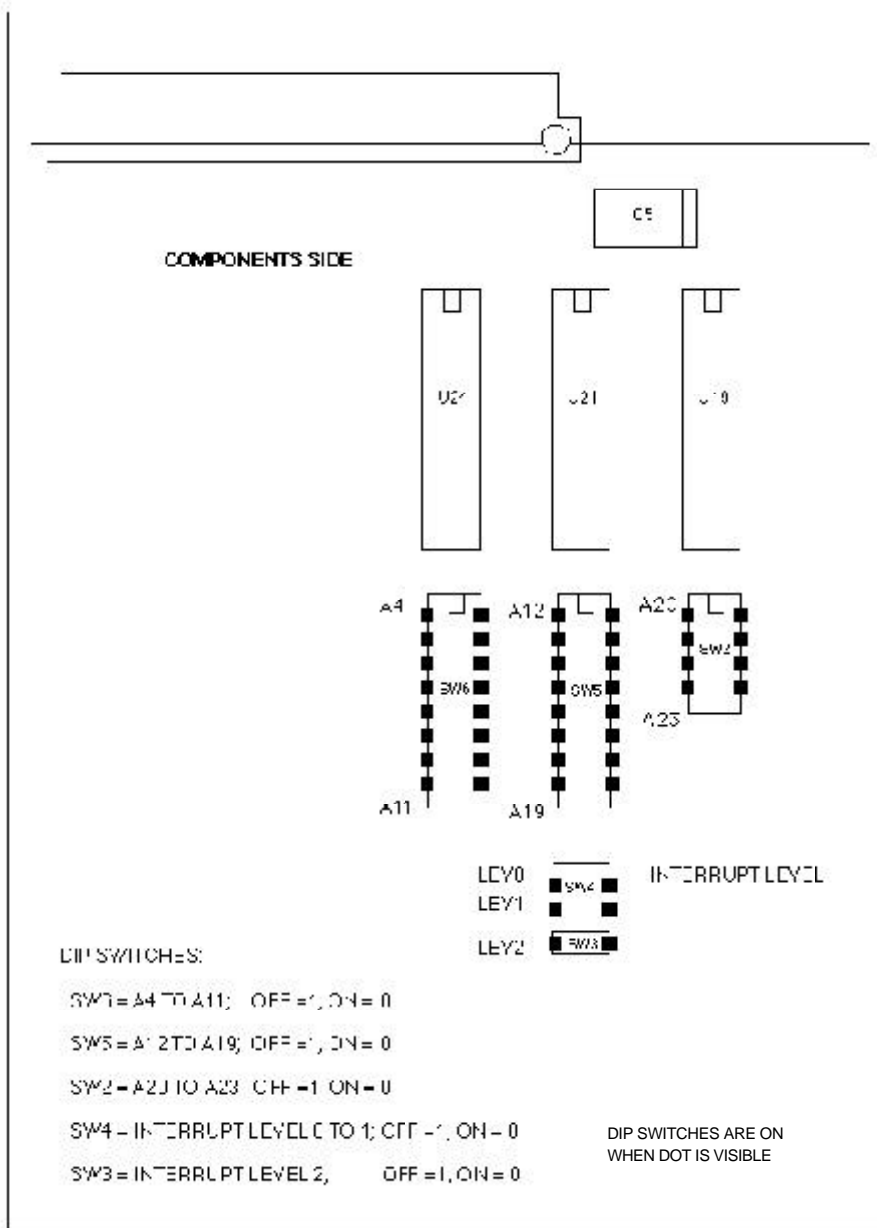


Fig 4.1 Mod. V288 dip switches setting.

5. C.A.E.N. PROTOCOL

5.1. H.S. CAENET NETWORK FOR REMOTING CONTROL.

H.S. CAENET provides a unique way of remotely controlling "passive" electronic modules; modules which have not been designed for specific data acquisition purposes, and therefore have no bus interfaces like CAMAC or VME.

NIM modules (Delay Units, Attenuators, Amplifiers, I/O registers) as well as H.V. supplies have several parameters which need to be adjusted under computer control; many CAEN units houses an H.S. CAENET Node inside the module that allows the possibility of linking devices of different types and functionality with a central controller.

5.2. H.S. CAENET NETWORK IMPLEMENTATION: CAEN APPROACH.

CAEN has developed a transmission protocol via H.S. CAENET line that permits the monitoring and control of many CAEN units from a single controller. This network has the following structure and protocol:

The transfers between H.S. CAENET Nodes take place according to the typical MASTER/SLAVES communication:

- There is a single MASTER : H.S. CAENET Controller;
- The SLAVES are daisy-chained on the network, and are identified by an address code (from 0 to 99); the address is usually selectable via thumb-wheel switch located on the front panel of the module;
- the H.S. CAENET MASTER initiates the transmission, all the SLAVES receive the data, and only the SLAVE addressed then accesses the serial line to transmit the data requested by the MASTER.

In this way is possible from a single point to control up to 100 SLAVES.

5.3. MASTER TO SLAVE DATA COMPOSITION

The MASTER to SLAVE data have to be written in the Transmit Data Buffer, by performing subsequent write accesses to the address Base +0, and should have the following structure.

Table 5.1: MASTER to SLAVE data composition

Order	Operation	Address	Datum (HEX)	Meaning
1	Write	Base Ad. + 0	%0001	H.S. CAENET Controller identified code
2	Write	Base Ad.+ 0	%00XX	SLAVE address code (0..99)
3	Write	Base Ad.+ 0	Code	Code of the operation to be performed
4 to 256	Write	Base Ad.+ 0	Set	Eventual set values

5.4. SLAVE TO MASTER DATA COMPOSITION

The answer data coming from the SLAVE or a Control Logic error message is stored into the V288 Receive Data buffer. The VME Master can get it by performing subsequent read accesses to the address Base +0. The following Table shows the structure of the SLAVE data packet:

Table 5.2: SLAVE to MASTER data composition

Order	Operation	Address	Datum	Meaning
1	Read	Base Ad + 0	Error Code	Error code
2 to 255(*)	Read	Base Ad + 0	value	Eventual Parameter value

(*) The first data of the packet is read and checked by the Control Logic as shown in § 3.4.5.

The Error codes are described in Tab. 5.3.

5.5. ERROR CODES

The Error codes are described in the following Table:

Table 5.3: Error Codes

Datum (Hex)	Meaning
%0	Successful operation.
%FFFD	No data to be transmitted; it has tried to start a transmission with the Transmit data Buffer empty. (Control logic error message).
%FFFE	The H.S. CAENET Controller identifier is incorrect. (Control Logic error message)
%FFFF	The addressed SLAVE does not exist. This message are generated after a period of 500 msec. (Control Logic error message).

All the other possible error codes from the SLAVE module have the format "%FFnn" where "nn" can be any number.

6. REFERENCES

[1] VMEbus specification Manual Revision C.1 October 1985

APPENDIX A SOFTWARE BUGS AS OF SEPTEMBER 1992

This appendix contains the software bugs recognized as of September 1992

A.1. H.S. CAENET OPERATION BUGS

If one of the SLAVE in the H.S. CAENET Network has the address code = 0 the Network communications do not work.

Do not use SLAVE address code = 0.

APPENDIX B SOFTWARE EXAMPLES

The detail of using the Mod.V288 to communicate with an H.S. CAENET SLAVE are explained by means of complete examples:

- VMECAENET.H: Declaration for communication via VME with the Mod. V288

- VMCAENET.C : Caenet Package for V288 Module

These two listings describes the function and general design of a driver for the Mod V288; all the possible errors are handled, included the VME Buserror.

VMESY403.C : Demonstration on the use of Caenet Routines in communication between V288 module and the SY403 HighVoltage System

This example is to be used as guides in creating a communication software between the V288 and an H.S.CAENET SLAVE module

```

/*****
/*
/*          -----          C . A . E . N .          SpA          -----
/*
/*          VMCAENET.H - Declarations for communication with V288 Module
/*
/*          *****/

#ifndef uchar
#define uchar          unsigned char
#endif
#ifndef ushort
#define ushort          unsigned short
#endif

/* Constants for vme_cycles routines */
#define BYTE          1
#define WORD          2
#define LWORD          4

/*
Errors returned by caenet_read and caenet_write; the positive ones
are depending from V288 Module and not from CAENET network
*/
#define TUTTOK          0
#define E_NO_Q_IDENT          1
#define E_NO_Q_CRATE          2
#define E_NO_Q_CODE          3
#define E_NO_Q_DATA          4
#define E_NO_Q_TX          5
#define E_NO_Q_RX          6
#define E_LESSDATA          7
#define E_BUSERR          8

/* Number of iterations before deciding that V288 does not answer */
#define TIMEOUT          -1
#define Q          (ushort)0xfffe
#define V288          1

/* Registers of V288 Module */
#define STATUS          (v288addr+0x02)
#define TXMIT          (v288addr+0x04)

#define LOBYTE(x)          (uchar)((x)&0xff)
#define HIBYTE(x)          (uchar)(((x)&0xff00) >> 8)

/*Interface between the user program and V288; these functions are defined
in file Vmcaenet.c */
int caenet_read();
int caenet_write();

/* Declarations of Global Variables defined in the user program */
extern unsigned          v288addr,craten;
extern ushort          code;

/*****
/*
/*          -----          C . A . E . N .          SpA          -----
/*
/*          VMCAENET.C - Caenet Package for V288 Module
/*
/*          *****/

```



```

#include "vmcaenet.h"

/****-----
Read_data
-----****/
int read_data(datovme)
ushort *datovme;
{
ushort q=0;
vme_read(v288addr, datovme, WORD);
vme_read(STATUS, &q, WORD);
return((q == Q) ? TUTTOK : TIMEOUT);
}

/****-----
Wait_resp
-----****/
int wait_resp(datovme)
ushort *datovme;
{
int i=0;
ushort q=0;
while(i!=TIMEOUT && q!=Q)
{
vme_read(v288addr, datovme, WORD);
vme_read(STATUS, &q, WORD);
i++;
}
return((i == TIMEOUT) ? TIMEOUT : TUTTOK);
}

/****-----
Send_comm
-----****/
int send_comm(vmeaddress, datovme)
unsigned int vmeaddress;
ushort datovme;
{
int i=0;
ushort q=0;
while(i!=TIMEOUT && q!=Q)
{
if(!vme_write(vmeaddress, &datovme, WORD))
return E_BUSERR;
vme_read(STATUS, &q, WORD);
i++;
}
return((i == TIMEOUT) ? TIMEOUT : TUTTOK);
}

/****-----

```

Caenet_read: Called by user programs to load "byte_count" bytes from CAENET into the buffer pointed by "*dest_buff".

The VME address of V288, the CAENET crate number and the

CAENET code are found in global variables.

Caenet_read returns TUTTOK = 0 if everything has worked;
It returns one from seven different errors (defined as positive constants in Vmcaenet.h) if it has received one error which strictly depends from V288 Module;
It returns a negative error (depending from the CAENET slave module) if the CAENET communication has not worked.

Remember: Module V288 can return three "general" negative errors related to the CAENET network that this routine does not handle separately from the "slave specific" ones.

```

-----***/
int caenet_read(dest_buff,byte_count)
uchar *dest_buff;
int byte_count;
{
int i,esito;
ushort mstident=V288,datatemp;
short dato;

if((esito=send_comm(v288addr,mstident)) == TIMEOUT)
return E_NO_Q_IDENT;
else if(esito == E_BUSERR)
return esito;

/* Transmit Crate Number */
if((esito=send_comm(v288addr,(ushort)craten)) == TIMEOUT)
return E_NO_Q_CRATE;
else if(esito == E_BUSERR)
return esito;

/* Transmit Code */
if((esito=send_comm(v288addr,(ushort)code)) == TIMEOUT)
return E_NO_Q_CODE;
else if(esito == E_BUSERR)
return esito;

/* Start Transmission */
if((esito=send_comm(TXMIT,mstident)) == TIMEOUT)
return E_NO_Q_TX;
else if(esito == E_BUSERR)
return esito;

if(wait_resp(&dato) == TIMEOUT)
return E_NO_Q_RX;

if(dato == TUTTOK) /* Test on the operation */
for(i=0;i<byte_count;i+=2)
{
if(read_data(&datatemp) == TIMEOUT && i<byte_count-1)
return E_LESSDATA;
dest_buff[i] = HIBYTE(datatemp);
dest_buff[i+1] = LOBYTE(datatemp);
}
return dato;
}

/****-----

```

Caenet_write: Called by user programs to transfer "byte_count" bytes to CAENET from the buffer pointed by "*source_buff".

The VME address of V288, the CAENET crate number and the CAENET code are found in global variables.

Caenet_write returns TUTTOK = 0 if everything has worked; It returns one from seven different errors (defined as positive constants in Vmcaenet.h) if it has received one error which strictly depends from V288 Module; It returns a negative error (depending from the CAENET slave module) if the CAENET communication has not worked.

Remember: Module V288 can return three "general" negative errors related to the CAENET network that this routine does not handle separately from the "slave specific" ones.

```

-----***/
int caenet_write(source_buff,byte_count)
uchar *source_buff;
int byte_count;
{
int i,esito;
ushort mstident=V288,datatemp;
short dato;

if((esito=send_comm(v288addr,mstident)) == TIMEOUT)
return E_NO_Q_IDENT;
else if(esito == E_BUSERR)
return esito;

/* Transmit Crate Number */
if((esito=send_comm(v288addr,(ushort)craten)) == TIMEOUT)
return E_NO_Q_CRATE;
else if(esito == E_BUSERR)
return esito;

/* Transmit Code */
if((esito=send_comm(v288addr,(ushort)code)) == TIMEOUT)
return E_NO_Q_CODE;
else if(esito == E_BUSERR)
return esito;

/* Transmit data */
for(i=0;i<byte_count;i+=2)
{
datatemp=(ushort)source_buff[i]<<8 | source_buff[i+1];
if((esito=send_comm(v288addr,datatemp)) == TIMEOUT)
return E_NO_Q_DATA;
else if(esito == E_BUSERR)
return esito;
}

/* Start transmission */
if((esito=send_comm(TXMIT,mstident)) == TIMEOUT)
return E_NO_Q_TX;
else if(esito == E_BUSERR)
return esito;

if(wait_resp(&dato) == TIMEOUT)
return E_NO_Q_RX;

return dato;
}

```



```

/*****
/*
/*          -----      C . A . E . N .      SpA          -----
/*
/*    VMESY403.C - Demonstration on the use of Caenet Routines in
/*                  communication between V288 module and SY403 High
/*                  Voltage System Version 1.06
/*
/*                  06/05/91 - Created
/*                  10/14/91 - Updated      System Software Version 1.08
/*                  11/15/91 - Updated      System Software Version 1.27
/*                  04/01/92 - Updated      System Software Version 1.40
/*
*****/

```

```

#include <stdio.h>
#include <strings.h>
#include "vmcaenet.h"

```

```

#ifndef uchar
#define uchar          unsigned char
#endif
#ifndef ushort
#define ushort        unsigned short
#endif

```

```

#define ESC            0x1b
#define CR             0x0d
#define BLANK         0x20

```

```

#define EUROCOM       0xff000000

```

```

#define IDENT         0
#define READ_STATUS   1
#define READ_SETTINGS 2
#define READ_LIMITS   3

```

```

#define V0SET         0
#define V1SET         1
#define I0SET         2
#define I1SET         3
#define VMAX          4
#define RUP            5
#define RDWN          6
#define TRIP          7

```

```

#define ISPRESENT(x)  (ch_read[(x)].status&(1<<2))
#define MAKE_CODE(ch,cod)  (((ch)<<8) | (cod))

```

```

/*
   The following macro transforms the V288 input address in a "good"
   VME address for Standard Accesses by Eltec CPU board
*/

```

```

#define UPDATE(addr)  ((unsigned int)EUROCOM + addr)

```

```

/*
   The following structure contains all the useful information about
   the settings of a channel
*/

```

```

struct hvch
{
  char  chname[12];

```

```

long   v0set;
long   v1set;
short  i0set;
short  ilset;
short  vmax;
short  rup;
short  rdwn;
short  trip;
ushort flag;
};

/*
   The following structure contains all the useful information about
   the status of a channel
*/
struct hvrdr
{
  long   vread;
  short  iread;
  ushort status;
};

/*
   The following structure contains all the useful information about the
   voltage and current limits of every board
*/
struct vi_max
{
  short  vmax[4];
  short  imax[4];
  short  resv[4];
  short  resi[4];
  short  decv[4];
  short  deci[4];
};

/*
   Globals
*/
int          y; /* File conio.c needs it */
unsigned     v288addr, craten;
ushort       code; /* Caenet code */
struct vi_max max_vi;

/****-----

Makemenu

-----****/
int makemenu()
{
  int c;
  clrscr();
  highvideo();
  puts("          - MAIN MENU -          \n\n\n ");
  normvideo();
  puts(" [0] - Read Module Identifier ");
  puts(" [1] - Channels 0..15 Monitor ");
  puts(" [2] - Channels 16..31 Monitor ");
  puts(" [3] - Channels 32..47 Monitor ");
  puts(" [4] - Channels 48..63 Monitor ");
  puts(" [5] - Parameter Setting      ");
  puts(" [6] - Speed test             ");
}

```

```

puts("\n\n [7] - Quit ");
while((c=getch()-'0') < 0 && c > 7);
return c;
}

/****-----
Read_Ident
-----****/
void read_ident()
{
int i,response;
char sy403ident[12];
char tempbuff[22];
code=IDENT; /* To see if sy403 is present */
if((response=caenet_read(tempbuff,22)) != TUTTOK && response != E_LESSDATA)
{
printf(" Caenet_read: Error number %d received\n",response);
puts(" Press any key to continue ");
getch();
return;
}
for(i=0;i<11;i++)
sy403ident[i]=tempbuff[2*i+1];
sy403ident[i]='\0';
printf(" The module has answered : %s\n",sy403ident);
puts(" Press any key to continue ");
getch();
}

/****-----
Get_limits
-----****/
int get_limits()
{
int response;

code=READ_LIMITS;
if((response=caenet_read(&max_vi,sizeof(struct vi_max))) != TUTTOK)
{
printf(" Caenet_read: Error number %d received\n",response);
puts(" Press any key to continue ");
getch();
}
return response;
}

/****-----
Ch_monitor
-----****/
void ch_monitor(group)
int group;
{
int i,
caratt='P',
response,
chs=(group-1)*16;
static float pow10[]={ 1.0, 10.0, 100.0};

```

```

float          scalei,scalev;
ushort        channel;
static int    page=0;
static struct hvch  ch_set[16];          /* Settings of 16 chs. */
static struct hvrđ  ch_read[16];       /* Status   of 16 chs. */

scalev=powl0[max_vi.decv[group-1]];
scalei=powl0[max_vi.deci[group-1]];
clrscr();
highvideo();
if(!page)
    puts
(" Channel      Vmon      Imon      V0set      I0set      V1set      I1set      Flag      Ch#
");
else
    puts
(" Channel      Vmax      Rup      Rdnw      Trip      Status      Ch# ");
normvideo();

gotoxy(1,23);
puts(" Press 'P' to change page, any other key to exit ");

while(caratt == 'P') /* Loops until someone presses a key different from P */
{

/* First update from Caenet the information about the channels */
    for(i=0;i<16;i++)
    {
        channel=(uchar)(chs+i);
        code=MAKE_CODE(channel,READ_STATUS);
        if((response=caenet_read((char *)&ch_read[i],sizeof(struct hvrđ)) != TUTTOK)
        {
            gotoxy(1,22);
            printf(" Caenet_read: Error number %d received\n",response);
            puts(" Press any key to continue                ");
            getch();
            return;
        }
        code=MAKE_CODE(channel,READ_SETTINGS);
        if((response=caenet_read((char *)&ch_set[i],sizeof(struct hvch)) != TUTTOK)
        {
            gotoxy(1,22);
            printf(" Caenet_read: Error number %d received\n",response);
            puts(" Press any key to continue                ");
            getch();
            return;
        }
    }

/* Then test if this group is present in the sistem */
    if(!ISPPRESENT(0))
    {
        gotoxy(1,22);
        puts(" Sorry, this group is not present ");
        puts(" Press any key to continue                ");
        getch();
        return;
    }

/* If the group is present, display the information */
    if(!page)
        /* Page 0 of display */
        for(i=0;i<16;i++)
        {

```



```

        gotoxy(1,i+5);
        printf(" %9s",ch_set[i].chname);
        gotoxy(12,i+5);
        printf
("%07.2f %07.2f %07.2f %07.2f %07.2f %07.2f %4x %2d \n",
ch_read[i].vread/scalev,ch_read[i].iread/scalei,ch_set[i].v0set/scalev,
ch_set[i].i0set/scalei,ch_set[i].vlset/scalev,ch_set[i].ilset/scalei,
ch_set[i].flag,chs+i);
    }
    else /* Page 1 of display */
        for(i=0;i<16;i++)
        {
            gotoxy(1,i+5);
            printf(" %9s",ch_set[i].chname);
            gotoxy(14,i+5);
            printf
("%4d %3d %3d %05.1f %4x %2d \n",
ch_set[i].vmax,ch_set[i].rup,ch_set[i].rdwn,ch_set[i].trip/10.0,
ch_read[i].status,chs+i);
        }

/* Test the keyboard */
    if(_gs_rdy(0) != -1) /* A key has been pressed */
        if((caratt=toupper(getch())) == 'P') /* They want to change page */
        {
            highvideo();
            page = !page;
            clrscr();
            if(page == 0)
                puts
(" Channel      Vmon      Imon      V0set      I0set      Vlset      Ilset      Flag      Ch#
");
            else
                puts
(" Channel      Vmax      Rup      Rdwn      Trip      Status      Ch# ");
            normvideo();
            gotoxy(1,23);
            puts(" Press 'P' to change page, any other key to exit ");
        }
    } /* End while */
}

/****-----
Par_set
-----****/
void par_set()
{
float      input_value,
           scale;
static float pow10[] = { 1.0, 10.0, 100.0};
ushort     channel,value;
int        i,
           response,
           par=0;
char       choiced_param[10];
static char *param[] =
{
    "v0set", "vlset", "i0set", "ilset", "vmax", "rup", "rdwn", "trip"
};

```

```

clrscr();
printf("\n\n Channel: ");
scanf("%d",&i);
channel=(uchar)i;
puts(" Allowed parameters (lowercase only) are:");
for(i=0;i<8;i++)
    puts(param[i]);
while(!par)
    {
    printf("\n Parameter to set: ");
    scanf("%s",choiced_param);
    for(i=0;i<8;i++)
        if(!strcmp(param[i],choiced_param))
            {
            par=1;
            break;
            }
    if(i==8)
        puts(" Sorry, this parameter is not allowed");
    }
printf(" New value :");
scanf("%f",&input_value);

switch(i)
    {
    case V0SET:
        code=MAKE_CODE(channel,16);
        scale=pow10[max_vi.decv[channel/16]];
        input_value*=scale;
        value=(ushort)input_value;
        break;
    case V1SET:
        code=MAKE_CODE(channel,17);
        scale=pow10[max_vi.decv[channel/16]];
        input_value*=scale;
        value=(ushort)input_value;
        break;
    case I0SET:
        code=MAKE_CODE(channel,18);
        scale=pow10[max_vi.deci[channel/16]];
        input_value*=scale;
        value=(ushort)input_value;
        break;
    case I1SET:
        code=MAKE_CODE(channel,19);
        scale=pow10[max_vi.deci[channel/16]];
        input_value*=scale;
        value=(ushort)input_value;
        break;
    case VMAX:
        code=MAKE_CODE(channel,20);
        value=(ushort)input_value;
        break;
    case RUP:
        code=MAKE_CODE(channel,21);
        value=(ushort)input_value;
        break;
    case RDWN:
        code=MAKE_CODE(channel,22);
        value=(ushort)input_value;
        break;
    case TRIP:
        code=MAKE_CODE(channel,23);

```

```

        input_value*=10;                                /* Trip is in 10-th of sec */
        value=(ushort)input_value;
        break;
    }

if((response=caenet_write(&value,sizeof(ushort))) != TUTTOK)
{
    printf(" Caenet_write: Error number %d received\n",response);
    puts(" Press any key to continue ");
    getch();
}
}

/****-----

Speed_test

-----****/
void speed_test()
{
    int i,response;
    ushort channel;
    char sy403ident[12],loopdata[12];
    char tempbuff[22];
    code=IDENT;                                        /* To see if sy403 is present */
    if((response=caenet_read(tempbuff,22)) != TUTTOK && response != E_LESSDATA)
    {
        printf(" Caenet_read: Error number %d received\n",response);
        puts(" Press any key to continue ");
        getch();
        return;
    }
    for(i=0;i<11;i++)
        sy403ident[i]=tempbuff[2*i+1];
    sy403ident[i]='\0';

    puts(" Looping, press any key to exit ... ");
    /* Loop until one presses a key */
    while(_gs_rdy(0) == -1)
    {
        if((response=caenet_read(tempbuff,22)) != TUTTOK && response != E_LESSDATA)
        {
            printf(" Caenet_read: Error number %d received\n",response);
            puts(" Press any key to continue ");
            getch();
            return;
        }
        for(i=0;i<11;i++)
            loopdata[i]=tempbuff[2*i+1];
        loopdata[i]='\0';
        if(strcmp(sy403ident,loopdata)) /* Data read in loop are not good */
        {
            printf(" Test_loop error: String read = %s\n",loopdata);
            puts(" Press any key to continue ");
            getch();
            return;
        }
    } /* end while */
    getch();
}

/****-----

```

```

Esci

-----***/
void esci()
{
clrscr();
deinit_buserr();          /* "Kill" my Bus Error Handler      */
exit(0);
}

/****-----

Main Program

-----***/
void main(argc,argv)
int argc;
char **argv;
{
if(argc != 3)
{
puts(" Usage: vmesy403 <v288 vme address (in hex)> <sy403 Caenet number (in
hex)>");
exit(0);
}
sscanf(++argv,"%x",&v288addr);
sscanf(++argv,"%x",&craten);

v288addr=UPDATE(v288addr);          /* For Eltec E-6 VME board */
init_buserr();                     /* To handle Bus Error      */

if(get_limits() != TUTTOK)         /* Get information about the boards */
esci();

/*
Main Loop
*/
for(;;)
switch(makemenu())
{
case 0:
read_ident();
break;
case 1:
ch_monitor(1);
break;
case 2:
ch_monitor(2);
break;
case 3:
ch_monitor(3);
break;
case 4:
ch_monitor(4);
break;
case 5:
par_set();
break;
case 6:
speed_test();
break;
case 7:

```

```
        esci();  
        break;  
    default:  
        break;  
    }  
}
```