Technical Information Manual

Revision n. 2

3 March 2000

MOD. A303 A

HIGH SPEED CAENET
PC CONTROLLER

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. *Overview*

The Mod. A 303A HIGH SPEED (H.S.) CAENET CONTROLLER is an ISA interface card. It allows the control of an H.S. CAENET network through an IBM PC or compatible.

The card can be mapped either in the I/O or Memory space of the PC (dip switch selectable).

Moreover the Mod. A 303A Controller can operate either in Polling or Interrupt mode (dip switch selectable).

The communication line uses a simple 50 Ω coaxial cable as its physical transmission medium. In order to avoid ground loops line connectors are optocoupled.

A functional block diagram of an H.S. CAENET node is shown in Fig. 1.1

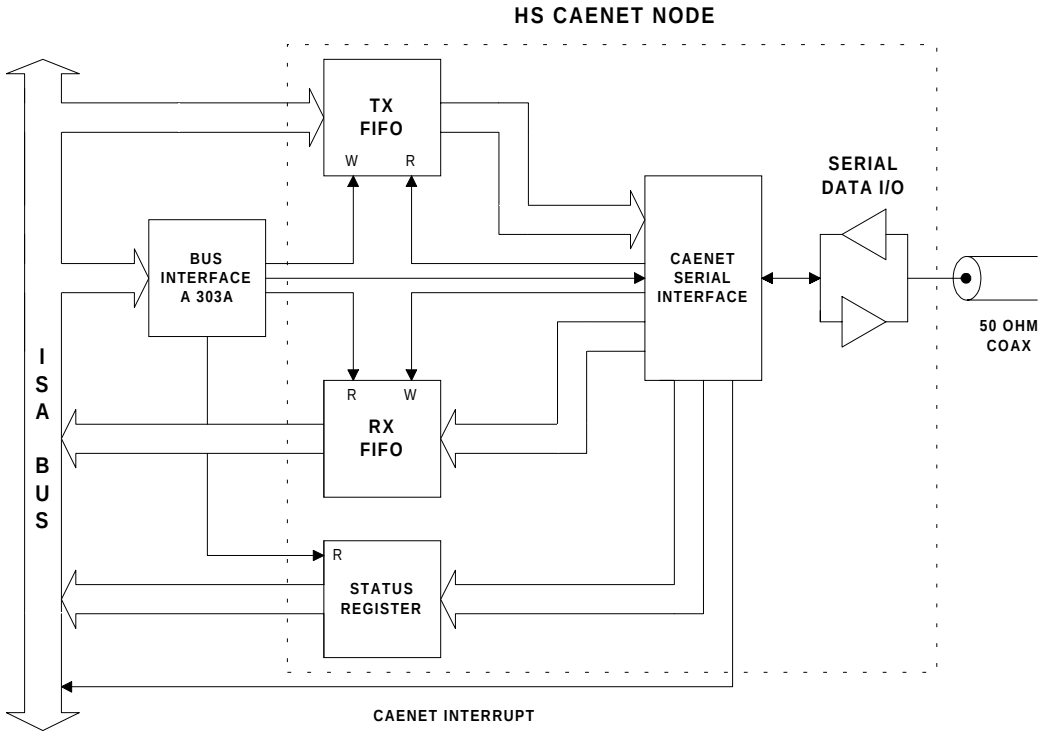


Fig. 1.1: H.S. CAENET node functional block diagram

2. *Configuration and installation*

2.1. Getting started

Since the Mod. A 303A is not a Plug an' Play card, any Operating System is unable to detect its installation or removal from the PC automatically; the card needs to be manually configured by the User with conflict-free PC resources assigned.

The Mod. A 303A can work either in Interrupt or Polling mode. If working in Interrupt mode, the card uses interrupt request line 9 to get service from the CPU. In this case it is necessary to reserve IRQ 9 of PC bus as follows:

- run the PC BIOS Set Up program
- go to the PNP/PCI CONFIGURATION menu
- select IRQ 9 assigned to Legacy ISA

Moreover, if any other Legacy ISA (not Plug an' Play) peripheral is present, the User should check that it is not using IRQ 9.

It is possible to map the Mod. A 303A either in the I/O or Memory space of the PC. The card requires 16 bit of address space and supports 16 bit (4 nibbles) addressing in I/O space and 20 bit (5 nibbles) addressing in Memory space. The base address is set by way of rotary switches SW 1…4 located on the card (see Fig. 2.1)). For example if I/O base address 320h has been selected for the card, the rotary switches must be set as follows:

SW1 Base address [19..16] = 0

SW2 Base address [15..12] = 0

SW3 Base address [11..8] = 3

SW4 Base address [7..4] = 2

2.2. Hardware installation

The card must be plugged into a free ISA slot of the PC motherboard.

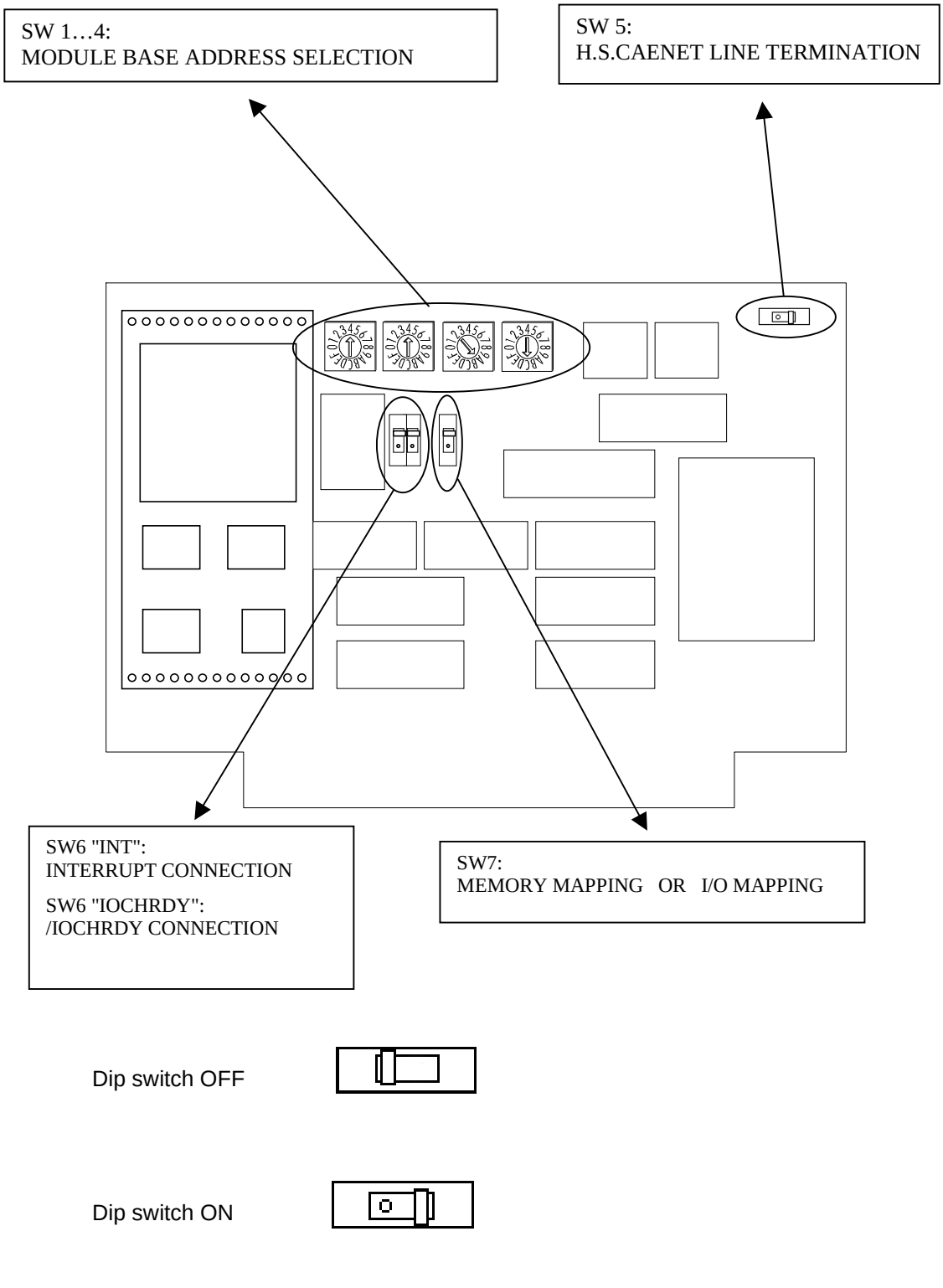


Fig. 2.1: Mod. A 303A components location

2.3. Software installation

The Mod. A 303A drivers together with a C functions library for the device control, demo programs and a diagnostic utility are available from the CAEN FTP support site for both Win32 and Linux Users.

2.3.1. Driver installation

2.3.1.1. Windows 9x Users

No device driver is needed under Windows 9x.

Before running the CAEN software under Windows 9x the Mod. A 303A interrupt generation must be disabled and I/O mapping selected (see § 3.3).

2.3.1.2. Windows NT Users

Before CAEN driver installation under Windows NT the Mod. A 303A interrupt generation must be enabled and I/O mapping selected (see § 3.3).

To install the Windows NT 4.0 device driver proceed as follows:

- Decompress the A303.zip archive which is available at:

  ftp://ftp.caen.it/pub/DocCaen/miscellan/a303a/Win32/

- Log in as Administrator or User with equivalent permissions
- Copy the file /a303drv/a303.sys in the %windir%/system32/drivers directory (%windir% corresponds to C:/WINNT on most systems)
- From the command prompt run the program regini.exe as follows:

  ```
  regini a303.ini
  ```

- Restart the computer

To verify that the driver has been correctly installed, after restarting the computer, select:

Start -> Settings -> Control panel -> Devices

If the A 303A is present in the devices list in "started" status the device driver has been correctly installed.

The driver supports Mod. A 303A as well as Mod. A 303.

2.3.1.3. Linux Users

Before CAEN driver installation under Linux the Mod. A 303A interrupt generation must be enabled and I/O mapping selected (see § 3.3).

To install the Linux (kernel 2.0.32) device driver proceed as follows:

- Decompress the A303.tgz archive which is available at:

  ftp://ftp.caen.it/pub/DocCaen/miscellan/a303a/Linux/

- Log in as root
- Load the driver launching the "A303_load script

To verify that the driver has been correctly installed, after restarting the computer, from the shell prompt type:

```
lsmod
```

If A303Adrv module is listed then the device driver has been correctly installed.

The driver supports Mod. A 303A as well as Mod. A 303

2.4. Diagnostic utility

The diagnostic utility a303diag, which is available for both Win32 and Linux users, verifies that the A 303A board has been correctly installed. To run the diagnostic utility the following command must be typed from the shell prompt:

```
A303diag [A303 I/O base address (hex)]
```

If the following message is displayed installation has been successful:

```
A303 board correctly installed
```

While if the following error message is displayed:

```
A303 reset failed: CAENET error: …
```

the User should check if all operation described in § 2.1 and § 2.3 have been correctly performed.

2.5. Software examples

2.5.1. Demo programs (Win32 Users)

Two demo programs (SY527demo.exe and SY127demo.exe) allowing to control an SY527 or SY127 Universal Multichannel Power Supply System via H.S. CAENET are available in the Bin directory of the A303.zip archive. Two Microsoft Visual C++ 6.0 projects with the demo programs source code and makefile are also available in the SYxxxdemo directory of the A303.zip archive.

The demo programs require the Win32 DLL A303lib.dll installed;

To run the demo program the following command must be typed from the DOS prompt:

```
SYxxxdemo [A303 I/O address (hex)] [crate number (1÷99)]
```

2.5.2. Demo programs (Linux Users)

Two demo programs (SY527demo.exe and SY127demo.exe) allowing to control an SY527 or SY127 Universal Multichannel Power Supply System via H.S. CAENET is available in the Distrib directory of the A303.tgz archive. The demo programs source code and makefile is also available in the Distrib directory of the A303.tgz archive.

To run the demo program the following command must be typed from the shell prompt:

```
SYxxxdemo [A303 I/O address (hex)] [crate number(1÷99)]
```

SYdemo527 and SY127demo use the NCURSES library.

2.6. Demo programs source code

The following source code is fully OS independent.

2.6.1. A303.h

```
/*****************************************************************************/
/*                                                                           */
/*        --- CAEN Engineering Srl - Computing Systems Division ---          */
/*                                                                           */
/*    A303.H                                                                  */
/*                                                                           */
/*    Declarations of routines which permit the control of A303/A module     */
/*    to a generic ANSI C program.                                           */
/*                                                                           */
/*    Created: January 2000                                                  */
/*                                                                           */
/*****************************************************************************/

#define MAX_LENGTH_FIFO 4096
#define TUTTOK             0


/***---------------------------------------------------------------------------

  A303Init
  Must be called before any other call to routines declared here.
  It initializes operating system dependent structures to correctly drive
  the A303/A module mapped at the I/O port specified.
  Arguments:
    Port - the I/O port
  Return value:
    error code, see A303DecodeResp for explaination.
```

```
    ------------------------------------------------------------------------***/
int   A303Init(unsigned long Port);

/***-------------------------------------------------------------------------

  A303SendCommand
  Sends a command to a device in the Caenet chain.
  Arguments:
    code  - the code of the command
    crnum - the crate number of the device in the chain
    buf   - a byte-array containing arguments eventually needed by the
            command specified
    count - length of the previous byte array
  Return value:
    error code, see A303DecodeResp for explaination.

    ------------------------------------------------------------------------***/
int   A303SendCommand(int code, int crnum, void *buf, int count);

/***-------------------------------------------------------------------------

  A303ReadResponse
  Waits for a response from the device.
  Arguments:
    buf   - the response from the device
    count - the length, in bytes, of the response
  Return value:
    error code, see A303DecodeResp for explaination.

    ------------------------------------------------------------------------***/
int   A303ReadResponse(void *buf, int *count);

/***-------------------------------------------------------------------------

  A303Reset
  Resets the A303/A module.
  Arguments:
    none
  Return value:
    error code, see A303DecodeResp for explaination.

    ------------------------------------------------------------------------***/
int   A303Reset(void);

/***-------------------------------------------------------------------------

  A303Timeout
  Sets the time to wait for a response.
  Arguments:
    Timeout - Time to wait in tenth of seconds.
  Return value:
    error code, see A303DecodeResp for explaination.

    ------------------------------------------------------------------------***/
int   A303Timeout(unsigned long Timeout);

/***-------------------------------------------------------------------------

  A303End
  Must be called after any other call to routines declared here.
  It resets operating system dependent structures.
  Arguments:
    none
  Return value:
    error code, see A303DecodeResp for explaination.

    ------------------------------------------------------------------------***/
int   A303End(void);

/***-------------------------------------------------------------------------
```

```
  A303DecodeResp
  Given the value returned from any routine declared here it returns an
  array of char that explains the error. Errors can be of two types: Caenet
  communication errors or operating system dependent.
  Arguments:
    resp - a code returned by any of the routines declared here.
  Return value:
    an explaination string for the error.

    ------------------------------------------------------------------***/
char *A303DecodeResp(int resp);
```

2.6.2. SY 527demo.c

```
/*************************************************************************/
/*                                                                       */
/*        --- CAEN Engineering Srl - Computing Systems Division ---      */
/*                                                                       */
/*    SY527DEMO.C                                                        */
/*                                                                       */
/*    Demonstration about the use of Caenet Routines in communication    */
/*    between A303/A module and SY527 Universal Multichannel Power Supply*/
/*    System                                                             */
/*                                                                       */
/*    Source code written in Ansi C                                      */
/*    Must be linked to console.c and to a303.c                          */
/*                                                                       */
/*    Created: January 2000                                              */
/*                                                                       */
/*************************************************************************/
#include  <stdio.h>
#include  <string.h>
#include  <time.h>
#include  <stdlib.h>
#include  <ctype.h>
#include  <stdarg.h>
#include  "console.h"
#include  "a303.h"

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

#define   MAX_CH_TYPES                            20

#define   IDENT                                    0
#define   READ_STATUS                              1
#define   READ_SETTINGS                            2
#define   READ_BOARD_INFO                          3
#define   READ_CR_CONF                             4
#define   READ_GEN_STATUS                          5
#define   READ_HVMAX                               6

#define   SET_STATUS_ALARM                      0x1a
#define   FORMAT_EEPROM_1                       0x30
#define   FORMAT_EEPROM_2                       0x31
#define   CLEAR_ALARM                           0x32
#define   LOCK_KEYBOARD                         0x33
#define   UNLOCK_KEYBOARD                       0x34
#define   KILL_CHANNELS_1                       0x35
#define   KILL_CHANNELS_2                       0x36
```

```
#define   LIST_CH_GRP                          0x40
#define   MON_CH_GRP                           0x41
#define   ADD_CH_GRP                           0x50
#define   REM_CH_GRP                           0x51

#define   GRP                                     0
#define   NO_GRP                                  1
#define   ADD                                     0
#define   REM                                     1

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

/*
  The following structure contains all the useful information about
  the settings of a channel
*/
struct hvch
{
 char    chname[12];
 long    v0set, v1set;
 ushort  i0set, i1set;
 short   vmax;
 short   rup, rdwn;
 short   trip, dummy;
 ushort  flag;
};

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

/*
  The following structure contains all the useful information about the
  characteristics of every board
*/
struct board
{
 char    name[5];
 char    curr_mis;
 ushort  sernum;
 char    vermaior;
 char    verminor;
 char    reserved[20];
 uchar   numch;
 ulong   omog;
 long    vmax;
 short   imax, rmin, rmax;
 short   resv, resi, decv, deci;
 uchar   dummy1;
};

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

```
struct chtype
{
 char   iumis;
 char   dummy;
 char   reserved[4];
 long   vmax;
 short  imax, rmin, rmax;
 short  resv, resi, decv, deci;
 char   dummy1[4];
};

/*
  The following structure contains all the useful information about
  the alarm status of SY527
*/
struct st_al
{
 unsigned  level   :1;
 unsigned  pulsed  :1;
 unsigned  ovc     :1;
 unsigned  ovv     :1;
 unsigned  unv     :1;
 unsigned  unused  :11;
};

/*
  Globals
*/
static struct board       boards[10];
static int                                code, cratenum;
static float              pow_10[]={ 1.0, 10.0, 100.0, 1000.0};
static struct st_al       status_alarm;
/*
  The following array contains the type of every channel for a given not
  homogeneous board
*/
static char               ch_to_type[32];
static struct chtype      chtypes[MAX_CH_TYPES];
static char               logfile[80];


/***---------------------------------------------------------------------------

  Caenet_comm
  It is a generic wrapper which permits every type of Caenet operation
  with SY527.
  Note that in this example we don't worry about the second parameter of
  A303ReadResponse which indicates the number of bytes read from Caenet

    ---------------------------------------------------------------------***/
static int caenet_comm(void *source_buff, int wr_byte_count, void *dest_buff)
{
 int resp, dummy;

 resp = A303SendCommand( code, cratenum, source_buff, wr_byte_count );
 if( resp != TUTTOK )
    return resp;
 else
   {
    return A303ReadResponse(dest_buff, &dummy);
   }
}

/***---------------------------------------------------------------------------

  Makemenu

    ---------------------------------------------------------------------***/
static int makemenu(void)
{
clrscr();
```

```
highvideo();
con_puts("                    - MAIN MENU -      \n\n\n ");
normvideo();
con_puts(" [A] - Read Module Identifier ");
con_puts(" [B] - Crate Map ");
con_puts(" [C] - Channels Monitor        ");
con_puts(" [D] - Speed test              ");
con_puts(" [E] - Parameter Setting       ");
con_puts(" [F] - Clear Alarms            ");
con_puts(" [G] - Set Alarm Type          ");
con_puts(" [H] - Lock Keyboard           ");
con_puts(" [I] - Unlock Keyboard         ");
con_puts(" [J] - Kill ALL Channels       ");
con_puts(" [K] - Front Panel Status      ");
con_puts(" [L] - Groups Operations       ");
con_puts(" [M] - Format EEPROM           ");
con_puts("\n\n [Q] - Quit ");

return toupper(con_getch());
}

/***-------------------------------------------------------------------------

  Makegrpmenu

    ----------------------------------------------------------------------***/
static int makegrpmenu(void)
{
clrscr();
highvideo();
con_puts("                    - GROUPS MENU -      \n\n\n ");
normvideo();
con_puts(" [A] - Add Channels to a Group ");
con_puts(" [B] - Remove Channels from a Group ");
con_puts(" [C] - Add Channels to all Groups ");
con_puts(" [D] - Remove Channels from all Groups ");
con_puts(" [E] - List Channels of a Group ");
con_puts(" [F] - Parameter Setting       ");
con_puts(" [G] - Monitor Channels of a Group       ");
con_puts("\n\n [Q] - Quit ");

return toupper(con_getch());
}

/***-------------------------------------------------------------------------

  Read_Ident

    ----------------------------------------------------------------------***/
static void read_ident(void)
{
int i,response;
char sy527ident[12];
char tempbuff[22];
code=IDENT;                                  /* To see if sy527 is present */
if((response=caenet_comm(NULL,0,tempbuff)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }
for(i=0;i<11;i++)
    sy527ident[i]=tempbuff[2*i];
sy527ident[i]='\0';
con_printf(" The module has answered : %s\n",sy527ident);
con_puts(" Press any key to continue ");
con_getch();
}

/***-------------------------------------------------------------------------
```

```
  Swap

    ------------------------------------------------------------------------***/
static void swap(char *a, char *b)
{
char temp;

temp = *a;
*a = *b;
*b = temp;
}

/***--------------------------------------------------------------------------

  Swap_Byte

    ------------------------------------------------------------------------***/
static void swap_byte(char *buff,int size)
{
int  i;

for( i=0 ; i<size ; i += 2 )
    swap(buff+i,buff+i+1);
}

/***--------------------------------------------------------------------------

  Swap_Long

    ------------------------------------------------------------------------***/
static void swap_long(void *buff)
{
swap((char *)buff,(char *)buff+3);
swap((char *)buff+1,(char *)buff+2);
}

/***--------------------------------------------------------------------------

  Size_of_Board
  We need the true size in bytes as SY527 sends it via CAENET; it is different
  from sizeof(struct board)

    ------------------------------------------------------------------------***/
static int size_of_board(void)
{
 struct board b;

 return   sizeof(b.curr_mis) + sizeof(b.deci)     + sizeof(b.decv)     +
sizeof(b.imax)
        + sizeof(b.name)     + sizeof(b.numch)    + sizeof(b.omog)     +
sizeof(b.reserved)
        + sizeof(b.resi)     + sizeof(b.resv)     + sizeof(b.rmax)     +
sizeof(b.rmin)
        + sizeof(b.sernum)   + sizeof(b.vermaior) + sizeof(b.verminor) +
sizeof(b.vmax)
        + sizeof(b.dummy1);
}

/***--------------------------------------------------------------------------

  Size_of_Chtype
  We need the true size in bytes as SY527 sends it via CAENET; it is different
  from sizeof(struct chtype)

    ------------------------------------------------------------------------***/
static int size_of_chtype(void)
{
 struct chtype c;

 return   sizeof(c.deci) + sizeof(c.decv)  + sizeof(c.dummy)    + sizeof(c.dummy1)
```

```
        + sizeof(c.imax) + sizeof(c.iumis) + sizeof(c.reserved) + sizeof(c.resi)
        + sizeof(c.resv) + sizeof(c.rmax)  + sizeof(c.rmin)     + sizeof(c.vmax);
}

/***-----------------------------------------------------------------------

  Size_of_Chset
  We need the true size in bytes as SY527 sends it via CAENET; it is different
  from sizeof(struct hvch)

    -----------------------------------------------------------------------***/
static int size_of_chset(void)
{
 struct hvch ch;

 return   sizeof(ch.chname) + sizeof(ch.dummy) + sizeof(ch.flag) + sizeof(ch.i0set)
        + sizeof(ch.i1set)  + sizeof(ch.rdwn)  + sizeof(ch.rup)  + sizeof(ch.trip)
        + sizeof(ch.v0set)  + sizeof(ch.v1set)  + sizeof(ch.vmax);
}

/***-----------------------------------------------------------------------

  Size_of_Chrd
  We need the true size in bytes as SY527 sends it via CAENET; it is different
  from sizeof(struct hvrd)

    -----------------------------------------------------------------------***/
static int size_of_chrd(void)
{
 struct hvrd ch;

 return   sizeof(ch.hvmax) + sizeof(ch.iread) + sizeof(ch.status) + sizeof(ch.vread);
}

/***-----------------------------------------------------------------------

  Build_Bd_Info

    -----------------------------------------------------------------------***/
static int build_bd_info(struct board *bd, char *cnetbuff)
{
int i = sizeof(bd->name);

swap_byte(cnetbuff, size_of_board());

memcpy(bd, cnetbuff, i);

memcpy(&bd->curr_mis, cnetbuff+i, sizeof(bd->curr_mis));
i += sizeof(bd->curr_mis);

memcpy(&bd->sernum, cnetbuff+i, sizeof(bd->sernum));
swap_byte((char *)&bd->sernum,sizeof(bd->sernum));
i += sizeof(bd->sernum);

memcpy(&bd->vermaior, cnetbuff+i, sizeof(bd->vermaior));
i += sizeof(bd->vermaior);

memcpy(&bd->verminor, cnetbuff+i, sizeof(bd->verminor));
i += sizeof(bd->verminor) + sizeof(bd->reserved);

memcpy(&bd->numch, cnetbuff+i, sizeof(bd->numch));
i += sizeof(bd->numch);

memcpy(&bd->omog, cnetbuff+i, sizeof(bd->omog));
swap_long(&bd->omog);
i += sizeof(bd->omog);

memcpy(&bd->vmax, cnetbuff+i, sizeof(bd->vmax));
swap_long(&bd->vmax);
i += sizeof(bd->vmax);
```

```
memcpy(&bd->imax, cnetbuff+i, sizeof(bd->imax));
swap_byte((char *)&bd->imax,sizeof(bd->imax));
i += sizeof(bd->imax);

memcpy(&bd->rmin, cnetbuff+i, sizeof(bd->rmin));
swap_byte((char *)&bd->rmin,sizeof(bd->rmin));
i += sizeof(bd->rmin);

memcpy(&bd->rmax, cnetbuff+i, sizeof(bd->rmax));
swap_byte((char *)&bd->rmax,sizeof(bd->rmax));
i += sizeof(bd->rmax);

memcpy(&bd->resv, cnetbuff+i, sizeof(bd->resv));
swap_byte((char *)&bd->resv,sizeof(bd->resv));
i += sizeof(bd->resv);

memcpy(&bd->resi, cnetbuff+i, sizeof(bd->resi));
swap_byte((char *)&bd->resi,sizeof(bd->resi));
i += sizeof(bd->resi);

memcpy(&bd->decv, cnetbuff+i, sizeof(bd->decv));
swap_byte((char *)&bd->decv,sizeof(bd->decv));
i += sizeof(bd->decv);

memcpy(&bd->deci, cnetbuff+i, sizeof(bd->deci));
swap_byte((char *)&bd->deci,sizeof(bd->deci));
i += sizeof(bd->deci) + sizeof(bd->dummy1);

if( bd->omog & (1L<<17) )      /* The board is not homogeneous */
  {
   int    j, n = bd->numch;
   short  typ;

   memcpy(&typ,cnetbuff+i,sizeof(typ));
   i += sizeof(typ);

   memcpy(ch_to_type, cnetbuff+i, (n&1) ? n+1 : n);
   i += ( (n&1) ? n+1 : n );
   swap_byte(ch_to_type,sizeof(ch_to_type));

   for( j = 0 ; j < typ ; j++ )
      {
       swap_byte(cnetbuff+i, size_of_chtype());

       memcpy(&chtypes[j].iumis, cnetbuff+i, sizeof(chtypes[j].iumis));
       i += sizeof(chtypes[j].iumis) + sizeof(chtypes[j].dummy) +
sizeof(chtypes[j].reserved);

       memcpy(&chtypes[j].vmax, cnetbuff+i, sizeof(chtypes[j].vmax));
       swap_long(&chtypes[j].vmax);
       i += sizeof(chtypes[j].vmax);

       memcpy(&chtypes[j].imax, cnetbuff+i, sizeof(chtypes[j].imax));
       swap_byte((char *)&chtypes[j].imax, sizeof(chtypes[j].imax));
       i += sizeof(chtypes[j].imax);

       memcpy(&chtypes[j].rmin, cnetbuff+i, sizeof(chtypes[j].rmin));
       swap_byte((char *)&chtypes[j].rmin, sizeof(chtypes[j].rmin));
       i += sizeof(chtypes[j].rmin);

       memcpy(&chtypes[j].rmax, cnetbuff+i, sizeof(chtypes[j].rmax));
       swap_byte((char *)&chtypes[j].rmax, sizeof(chtypes[j].rmax));
       i += sizeof(chtypes[j].rmax);

       memcpy(&chtypes[j].resv, cnetbuff+i, sizeof(chtypes[j].resv));
       swap_byte((char *)&chtypes[j].resv, sizeof(chtypes[j].resv));
       i += sizeof(chtypes[j].resv);

       memcpy(&chtypes[j].resi, cnetbuff+i, sizeof(chtypes[j].resi));
       swap_byte((char *)&chtypes[j].resi, sizeof(chtypes[j].resi));
       i += sizeof(chtypes[j].resi);
```

```
        memcpy(&chtypes[j].decv, cnetbuff+i, sizeof(chtypes[j].decv));
        swap_byte((char *)&chtypes[j].decv, sizeof(chtypes[j].decv));
        i += sizeof(chtypes[j].decv);

        memcpy(&chtypes[j].deci, cnetbuff+i, sizeof(chtypes[j].deci));
        swap_byte((char *)&chtypes[j].deci, sizeof(chtypes[j].deci));
        i += sizeof(chtypes[j].deci) + sizeof(chtypes[j].dummy1);
       }
  }

return TUTTOK;
}

/***-----------------------------------------------------------------------

  Build_Chset_Info

    ----------------------------------------------------------------------***/
static void build_chset_info(struct hvch *ch, char *cnetbuff)
{
int i = sizeof(ch->chname);

swap_byte(cnetbuff, size_of_chset());

memcpy(&ch->chname, cnetbuff, i);

memcpy(&ch->v0set,cnetbuff+i, sizeof(ch->v0set));
swap_long(&ch->v0set);
i += sizeof(ch->v0set);

memcpy(&ch->v1set,cnetbuff+i, sizeof(ch->v1set));
swap_long(&ch->v1set);
i += sizeof(ch->v1set);

memcpy(&ch->i0set,cnetbuff+i, sizeof(ch->i0set));
swap_byte((char *)&ch->i0set, sizeof(ch->i0set));
i += sizeof(ch->i0set);

memcpy(&ch->i1set,cnetbuff+i, sizeof(ch->i1set));
swap_byte((char *)&ch->i1set, sizeof(ch->i1set));
i += sizeof(ch->i1set);

memcpy(&ch->vmax,cnetbuff+i, sizeof(ch->vmax));
swap_byte((char *)&ch->vmax, sizeof(ch->vmax));
i += sizeof(ch->vmax);

memcpy(&ch->rup,cnetbuff+i, sizeof(ch->rup));
swap_byte((char *)&ch->rup, sizeof(ch->rup));
i += sizeof(ch->rup);

memcpy(&ch->rdwn,cnetbuff+i, sizeof(ch->rdwn));
swap_byte((char *)&ch->rdwn, sizeof(ch->rdwn));
i += sizeof(ch->rdwn);

memcpy(&ch->trip,cnetbuff+i, sizeof(ch->trip));
swap_byte((char *)&ch->trip, sizeof(ch->trip));
i += sizeof(ch->trip);

memcpy(&ch->dummy,cnetbuff+i, sizeof(ch->dummy));
swap_byte((char *)&ch->dummy, sizeof(ch->dummy));
i += sizeof(ch->dummy);

memcpy(&ch->flag,cnetbuff+i, sizeof(ch->flag));
swap_byte((char *)&ch->flag, sizeof(ch->flag));
i += sizeof(ch->flag);
}

/***-----------------------------------------------------------------------

  Build_Chrd_Info
```

```
    ------------------------------------------------------------------***/
static void build_chrd_info(struct hvrd *ch, char *cnetbuff)
{
int i = sizeof(ch->vread);

swap_byte(cnetbuff, size_of_chrd());

memcpy(&ch->vread, cnetbuff, sizeof(ch->vread));
swap_long(&ch->vread);

memcpy(&ch->hvmax, cnetbuff+i, sizeof(ch->hvmax));
swap_byte((char *)&ch->hvmax, sizeof(ch->hvmax));
i += sizeof(ch->hvmax);

memcpy(&ch->iread, cnetbuff+i, sizeof(ch->iread));
swap_byte((char *)&ch->iread, sizeof(ch->iread));
i += sizeof(ch->iread);

memcpy(&ch->status, cnetbuff+i, sizeof(ch->status));
swap_byte((char *)&ch->status, sizeof(ch->status));
i += sizeof(ch->status);
}

/***-------------------------------------------------------------------------

  Get_Cr_Info

    ------------------------------------------------------------------***/
static int get_cr_info(ushort *cr_cnf)
{
int   i,response;
short bd;
char  cnetbuff[MAX_LENGTH_FIFO];

code=READ_CR_CONF;
if((response=caenet_comm(NULL,0,cr_cnf)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return response;
  }

code=READ_BOARD_INFO;
for( bd=0, i=1 ; bd<10 ; bd++, i = i << 1 )
    if(*cr_cnf & i)
      {
       if((response=caenet_comm(&bd,sizeof(bd),cnetbuff)) != TUTTOK)
         {
          con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
          con_puts(" Press any key to continue ");
          con_getch();
          return response;
         }
       else
          build_bd_info((struct board *)&boards[bd],cnetbuff);
      }

return TUTTOK;
}

/***-------------------------------------------------------------------------

  Crate_Map

    ------------------------------------------------------------------***/
static void crate_map(void)
{
static char *curr_umis[] =
{
```

```
 " A",
 "mA",
 "uA",
 "nA"
};
int         i,bd;
float       im;
ushort      cr_conf;

if(get_cr_info(&cr_conf) != TUTTOK) /* Get information about the Crate Configuration
*/
   return;

clrscr();
con_puts("\n\n                          ---  Crate Map ---       \n\n\n\n\n ");

for( bd=0, i=1 ; bd<10 ; bd++, i = i << 1 )
   {
    con_printf(" Slot %d - ",bd);

    if(cr_conf & i)
      {
       char bdname[6];

       strncpy(bdname,boards[bd].name,5);
       bdname[5] = '\0';
       con_printf(" Mod. %-5s  %3d CH ",bdname,boards[bd].numch);
       con_printf("  %4ldV",boards[bd].vmax);
       im = (float)boards[bd].imax/pow_10[boards[bd].deci];
       con_printf("  %8.2f",im);
       con_printf("%s",curr_umis[boards[bd].curr_mis]);
       con_printf(" --- Ser. %3d, Rel. %d.%02d\n",
               boards[bd].sernum,boards[bd].vermaior,boards[bd].verminor);
      }
    else
       con_printf(" Not Present \n");

   }
con_puts("\n\n\n   Press any key to continue ");
con_getch();
}

/***--------------------------------------------------------------------------

  Ch_monitor

    ------------------------------------------------------------------***/
static void ch_monitor(void)
{
int                     temp,caratt='P',
                        response;
ushort                  bd,ch,ch_addr;
char                    cnetbuff[MAX_LENGTH_FIFO];
static int              page=0;
static struct hvch      ch_set[32];        /* Settings of 32 chs.   */
static struct hvrd      ch_read[32];       /* Status   of 32 chs.   */

do
  {
   clrscr();
   con_printf(" Input Board Number [0 ... 9]: ");
   con_scanf("%d",&temp);
  }
while(temp < 0 || temp > 9);

bd = temp;
code=READ_BOARD_INFO;
if((response=caenet_comm(&bd,sizeof(bd),cnetbuff)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
```

```
   con_getch();
   return;
  }
else
   build_bd_info(&boards[bd], cnetbuff);

clrscr();
con_printf(" Input Board Number [0 ... 9]: %d\n",bd);
highvideo();
if(!page)
   con_puts
("\n Channel     Vmon     Imon    V0set    I0set    V1set    I1set   Flag    Ch# ");
else
   con_puts
("\n Channel     Vmax     Rup    Rdwn     Trip     Status    Ch# ");
normvideo();

gotoxy(1,23);
con_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( ch=0 ; ch < 16 && ch < boards[bd].numch ; ch++ )
         {
          ch_addr = (bd<<8) | ch;

          code = READ_STATUS;
          response = caenet_comm(&ch_addr,sizeof(ch_addr), cnetbuff);
          if( response != TUTTOK )
            {
             con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
             con_puts(" Press any key to continue ");
             con_getch();
             return;
            }
          build_chrd_info(&ch_read[ch], cnetbuff);

          code = READ_SETTINGS;
          response = caenet_comm(&ch_addr,sizeof(ch_addr), cnetbuff);
          if( response != TUTTOK )
            {
             con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
             con_puts(" Press any key to continue ");
             con_getch();
             return;
            }
          build_chset_info(&ch_set[ch], cnetbuff);
         }   /* end for( ch ) */

      if(!page)                                    /* Page 0 of display */
         for( ch=0 ; ch < 16 && ch < boards[bd].numch ; ch++ )
            {
             float scalev, scalei;

             if( boards[bd].omog & (1L<<17) )
               {
                scalev = pow_10[ chtypes[ ch_to_type[ch] ].decv ];
                scalei = pow_10[ chtypes[ ch_to_type[ch] ].deci ];
               }
             else
               {
                scalev = pow_10[ boards[bd].decv ];
                scalei = pow_10[ boards[bd].deci ];
               }

             gotoxy(1,ch+5);
             con_printf(" %9s",ch_set[ch].chname);
             gotoxy(12,ch+5);
             con_printf
```

```
("%07.2f  %07.2f  %07.2f  %07.2f  %07.2f  %07.2f  %4x     %2d ",
ch_read[ch].vread/scalev,ch_read[ch].iread/scalei,ch_set[ch].v0set/scalev,
ch_set[ch].i0set/scalei,ch_set[ch].v1set/scalev,ch_set[ch].i1set/scalei,
ch_set[ch].flag,ch);
            }
      else                                          /* Page 1 of display */
         for( ch=0 ; ch < 16 && ch < boards[bd].numch ; ch++ )
            {
             gotoxy(1,ch+5);
             con_printf(" %9s",ch_set[ch].chname);
             gotoxy(14,ch+5);
             con_printf
("%4d     %3d     %3d    %05.1f       %4x     %2d ",
ch_set[ch].vmax,ch_set[ch].rup,ch_set[ch].rdwn,ch_set[ch].trip/10.0,
ch_read[ch].status,ch);
            }

/* Test the keyboard */
      if(con_kbhit())                              /* A key has been pressed */
         if((caratt=toupper(con_getch())) == 'P') /* They want to change page */
           {
            page = !page;
            clrscr();
            con_printf(" Input Board Number [0 ... 9]: %d\n",bd);
            if(page == 0)
               con_puts
("\n Channel     Vmon    Imon    V0set    I0set    V1set    I1set    Flag    Ch# ");
            else
               con_puts
("\n Channel     Vmax     Rup    Rdwn     Trip     Status    Ch# ");
            gotoxy(1,23);
            con_puts(" Press 'P' to change page, any other key to exit ");
           }

     }  /* End while */
}

/***---------------------------------------------------------------------------

  Par_set

    ---------------------------------------------------------------------***/
static void par_set(int type)
{
float       input_value,
            scale;
ushort      channel, bd, cnet_buff[2];
int         temp,ch,i,g,
            response,
            par=0;
char        choiced_param[10], cnetbuff[MAX_LENGTH_FIFO];
static char *param[] =
{
 "v0set", "v1set", "i0set", "i1set", "vmax", "rup", "rdwn", "trip", NULL
};

clrscr();
if( type == NO_GRP )
  {
   con_printf("\n\n Board: ");                     /* Choice the board   */
   con_scanf("%d",&temp);
   bd = temp;
   code=READ_BOARD_INFO;
   if((response=caenet_comm(&bd,sizeof(bd),cnetbuff)) != TUTTOK)
     {
      con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
      con_puts(" Press any key to continue ");
      con_getch();
      return;
     }
   else
```

```
       build_bd_info((struct board *)&boards[bd],cnetbuff);

   con_printf("\n\n Channel: ");                    /* Choice the channel */
   con_scanf("%d",&ch);
   channel = (bd<<8) | ch;
  }
else
  {
   con_printf("\n\n Group: ");                     /* Choice the group   */
   con_scanf("%d",&g);
  }

con_puts(" Allowed parameters (lowercase only) are:");
for( i = 0 ; param[i] != NULL ; i++ )
    con_puts(param[i]);

while(!par)
     {
      con_printf("\n Parameter to set: ");        /* Choice the parameter */
      con_scanf("%s",choiced_param);
      for( i = 0 ; param[i] != NULL ; i++ )
          if(!strcmp(param[i],choiced_param))
            {
             par=1;
             break;
            }
      if( param[i] == NULL )
         con_puts(" Sorry, this parameter is not allowed");
     }
con_printf(" New value :");                       /* Choice the value         */
con_scanf("%f",&input_value);

if( type == NO_GRP )
   cnet_buff[0] = channel;
else
   cnet_buff[0] = g;
switch(i)                                      /* Decode the par.          */
  {
   case V0SET:
       code = ( ( type == NO_GRP ) ? 0x10 : 0x52 );
       if( type == NO_GRP )
         {
          if( boards[bd].omog & (1L<<17) )
             scale = pow_10[ chtypes[ ch_to_type[ch] ].decv ];
          else
             scale = pow_10[ boards[bd].decv ];
         }
       else
          scale = 10.0;
       input_value *= scale;
       cnet_buff[1]=(ushort)input_value;
       break;
   case V1SET:
       code = ( ( type == NO_GRP ) ? 0x11 : 0x53 );
       if( type == NO_GRP )
         {
          if( boards[bd].omog & (1L<<17) )
             scale = pow_10[ chtypes[ ch_to_type[ch] ].decv ];
          else
             scale = pow_10[ boards[bd].decv ];
         }
       else
          scale = 10.0;
       input_value*=scale;
       cnet_buff[1]=(ushort)input_value;
       break;
   case I0SET:
       code = ( ( type == NO_GRP ) ? 0x12 : 0x54 );
       if( type == NO_GRP )
         {
          if( boards[bd].omog & (1L<<17) )
```

```
                scale = pow_10[ chtypes[ ch_to_type[ch] ].deci ];
              else
                scale = pow_10[ boards[bd].deci ];
             }
           else
              scale = 10.0;                /* Not very correct ... */
           input_value*=scale;
           cnet_buff[1]=(ushort)input_value;
           break;
      case I1SET:
           code = ( ( type == NO_GRP ) ? 0x13 : 0x55 );
           if( type == NO_GRP )
             {
              if( boards[bd].omog & (1L<<17) )
                scale = pow_10[ chtypes[ ch_to_type[ch] ].deci ];
              else
                scale = pow_10[ boards[bd].deci ];
             }
            else
              scale = 10.0;                /* Not very correct ... */
           input_value*=scale;
           cnet_buff[1]=(ushort)input_value;
           break;
      case VMAX:
           code = ( ( type == NO_GRP ) ? 0x14 : 0x56 );
           cnet_buff[1]=(ushort)input_value;
           break;
      case  RUP:
           code = ( ( type == NO_GRP ) ? 0x15 : 0x57 );
           cnet_buff[1]=(ushort)input_value;
           break;
      case RDWN:
           code = ( ( type == NO_GRP ) ? 0x16 : 0x58 );
           cnet_buff[1]=(ushort)input_value;
           break;
      case TRIP:
           code = ( ( type == NO_GRP ) ? 0x17 : 0x59 );
           input_value*=10;                     /* Trip is in 10-th of sec  */
           cnet_buff[1]=(ushort)input_value;
           break;
     }

   if((response=caenet_comm(cnet_buff,sizeof(cnet_buff),NULL)) != TUTTOK)
     {
      con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
      con_puts(" Press any key to continue ");
      con_getch();
     }
   }

   /***-------------------------------------------------------------------------

     Speed_test

       ------------------------------------------------------------------------***/
   static void speed_test(void)
   {
   int i,response;
   char sy527ident[12],loopdata[12];
   char tempbuff[22];
   code=IDENT;                                  /* To see if sy527 is present */
   if((response=caenet_comm(NULL,0,tempbuff)) != TUTTOK)
     {
      con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
      con_puts(" Press any key to continue ");
      con_getch();
      return;
     }
   for(i=0;i<11;i++)
       sy527ident[i]=tempbuff[2*i+1];
   sy527ident[i]='\0';
```

```
con_puts(" Looping, press any key to exit ... ");
/* Loop until one presses a key */
while(!con_kbhit())
     {
      if((response=caenet_comm(NULL,0,tempbuff)) != TUTTOK)
        {
         con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
         con_puts(" Press any key to continue ");
         con_getch();
         return;
        }

      for(i=0;i<11;i++)
          loopdata[i]=tempbuff[2*i+1];
      loopdata[i]='\0';
      if(strcmp(sy527ident,loopdata))    /* Data read in loop are not good */
        {
         con_printf(" Test_loop error: String read = %s\n",loopdata);
         con_puts(" Press any key to continue ");
         con_getch();
         return;
        }
     }  /* end while */
con_getch();
}

/***-------------------------------------------------------------------------

  Clear_Alarm

    -------------------------------------------------------------------------***/
static void clear_alarm(void)
{
int response;

code = CLEAR_ALARM;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }
}

/***-------------------------------------------------------------------------

  Print_Status_Level_Value

    -------------------------------------------------------------------------***/
static void print_status_level_value(void)
{
gotoxy(31,12);
if(status_alarm.level)
   con_printf("High");
else
   con_printf("Low ");
}

/***-------------------------------------------------------------------------

  Print_Status_Pulsed_Value

    -------------------------------------------------------------------------***/
static void print_status_pulsed_value(void)
{
gotoxy(31,13);
if(status_alarm.pulsed)
   con_printf("Pulsed ");
else
   con_printf("Level  ");
```

```
}

/***-------------------------------------------------------------------------

  Print_Status_Ovc_Value

    -------------------------------------------------------------------------***/
static void print_status_ovc_value(void)
{
gotoxy(31,15);
if(status_alarm.ovc)
   con_printf("On  ");
else
   con_printf("Off ");
}

/***-------------------------------------------------------------------------

  Print_Status_Ovv_Value

    -------------------------------------------------------------------------***/
static void print_status_ovv_value(void)
{
gotoxy(31,16);
if(status_alarm.ovv)
   con_printf("On  ");
else
   con_printf("Off ");
}

/***-------------------------------------------------------------------------

  Print_Status_Unv_Value

    -------------------------------------------------------------------------***/
static void print_status_unv_value(void)
{
gotoxy(31,17);
if(status_alarm.unv)
   con_printf("On  ");
else
   con_printf("Off ");
}

/***-------------------------------------------------------------------------

  Status_Menu

    -------------------------------------------------------------------------***/
static int  status_menu(void)
{
int     response;
ushort  value[2];

clrscr();

code = READ_GEN_STATUS;
if((response=caenet_comm(NULL,0,value)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return 0;
  }

memcpy(&status_alarm,value,sizeof(short));

gotoxy(7,9);
highvideo();
con_printf("Select Status Alarm Mode");
normvideo();
```

```
gotoxy(1,12);
con_printf("      A)  Normal Level:");
print_status_level_value();

gotoxy(1,13);
con_printf("      B)  Alarm  Type :");
print_status_pulsed_value();

gotoxy(1,15);
con_printf("      D)  OVC    Alarm:");
print_status_ovc_value();

gotoxy(1,16);
con_printf("      E)  OVV    Alarm:");
print_status_ovv_value();

gotoxy(1,17);
con_printf("      F)  UNV    Alarm:");
print_status_unv_value();

gotoxy(1,20);
con_printf("      Q)  Quit");

gotoxy(7,23);
con_printf("Select item\r\n");

return 1;
}

/***-------------------------------------------------------------------------

  Set_Status_Alarm

    ---------------------------------------------------------------------***/
static void set_status_alarm(void)
{
int c, modified, response;

if(!status_menu())
   return;

while(1)
  {
   modified = 0;

   c = tolower(con_getch());

   switch (c)
     {
      case  'a' : status_alarm.level = !status_alarm.level;
                  print_status_level_value();
                  modified = 1;
                  break;

      case  'b' : status_alarm.pulsed = !status_alarm.pulsed;
                  print_status_pulsed_value();
                  modified = 1;
                  break;

      case  'd' : status_alarm.ovc = !status_alarm.ovc;
                  print_status_ovc_value();
                  modified = 1;
                  break;

      case  'e' : status_alarm.ovv = !status_alarm.ovv;
                  print_status_ovv_value();
                  modified = 1;
                  break;

      case  'f' : status_alarm.unv = !status_alarm.unv;
```

```
                    print_status_unv_value();
                    modified = 1;
                    break;
          }  /* end switch */

         if(modified)
           {
            code = SET_STATUS_ALARM;
            if((response=caenet_comm(&status_alarm,sizeof(short),NULL)) != TUTTOK)
              {
               con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
               con_puts(" Press any key to continue ");
               con_getch();
               return;
              }
            modified = 0;
           }
         if(c == 'q')
            break;

         gotoxy(1,24);
        }     /* end while(1) */
}

/***-------------------------------------------------------------------------

  Lock_Keyboard

    -------------------------------------------------------------------------***/
static void lock_keyboard(void)
{
int response;

code = LOCK_KEYBOARD;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }
}

/***-------------------------------------------------------------------------

  Unlock_Keyboard

    -------------------------------------------------------------------------***/
static void unlock_keyboard(void)
{
int response;

code = UNLOCK_KEYBOARD;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }
}

/***-------------------------------------------------------------------------

  Kill_Channels

    -------------------------------------------------------------------------***/
static void kill_channels(void)
{
int c, response;

clrscr();
gotoxy(2,9);
```

```
con_printf("KILL ALL Channels. Are you sure ? (Y/N) [N]: ");
for(;;)
   {
    c = tolower(con_getch());
    if( c == 'y' || c == 'n' || c == '\r' )
       break;
   }
if( c == 'n' || c == '\r' )
   return;

con_putch('Y');

code = KILL_CHANNELS_1;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }

con_printf("\n\n Executing ... \n");

code = KILL_CHANNELS_2;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }
}

/***-------------------------------------------------------------------------

  Format_EEPROM

    ------------------------------------------------------------------------***/
static void format_eeprom(void)
{
int c, response;

clrscr();
gotoxy(2,9);
con_printf("Format EEPROM. Are you sure ? (Y/N) [N]: ");
for(;;)
   {
    c = tolower(con_getch());
    if( c == 'y' || c == 'n' || c == '\r' )
       break;
   }
if( c == 'n' || c == '\r' )
   return;

con_putch('Y');

code = FORMAT_EEPROM_1;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }

con_printf("\n\n Executing ... \n");

code = FORMAT_EEPROM_2;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }
```

```
}

/***-------------------------------------------------------------------------

  Fpan_Stat

    ------------------------------------------------------------------------***/
static void fpan_stat(void)
{
int     response;
ushort  value[2];

clrscr();

code = READ_GEN_STATUS;
if((response=caenet_comm(NULL,0,value)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }

gotoxy(7,5);
highvideo();
con_printf("SY527 Front Panel Status");
normvideo();

gotoxy(1,10);
con_printf("              V Selected       :");
gotoxy(1,12);
con_printf("              I Selected       :");
gotoxy(1,14);
con_printf("              KILL Status      :");
gotoxy(1,16);
con_printf("              Interlock Status :");
gotoxy(1,18);
con_printf("              HV Enable        :");

gotoxy(7,23);
con_printf("Press any key to exit");

highvideo();

while(!con_kbhit())
  {
   gotoxy(34,10);
   con_printf( (value[1] & (1<<0)) ? "V1SEL" : "V0SEL" );
   gotoxy(34,12);
   con_printf( (value[1] & (1<<1)) ? "I1SEL" : "I0SEL" );
   gotoxy(34,14);
   con_printf( (value[1] & (1<<2)) ? "On " : "Off" );
   gotoxy(34,16);
   con_printf( (value[1] & (1<<3)) ? "On " : "Off" );
   gotoxy(34,18);
   con_printf( (value[1] & (1<<4)) ? "On " : "Off" );

   if((response=caenet_comm(NULL,0,value)) != TUTTOK)
     {
      con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
      con_puts(" Press any key to continue ");
      break;
     }

   delay(500);
  }

con_getch();
normvideo();
}
```

```
/***-------------------------------------------------------------------------

  List_ch_grp

    ---------------------------------------------------------------------***/
static void list_ch_grp(void)
{
int    i, j, response,
       g, endloop = 0;
short  group, d, data[6+320*2];

do
  {
   clrscr();
   gotoxy(1,3);
   con_printf(" Input Group Number [0 .. 15]: ");
   con_scanf("%d",&g);
   if( g >= 0 && g < 16 )
      endloop = 1;
  }
while( !endloop );

code = LIST_CH_GRP;
group = g;

if( ( response = caenet_comm(&group,sizeof(group),data) ) != TUTTOK )
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }

clrscr();
con_printf(" Channels List of Group %02d",g);
for( i = 0, j = 0 ; ( d = data[6+2*i] ) != -1 ;  )
   {
    gotoxy(1+5*j,3+(i%22));
    con_printf("%d.%02d",d>>8,d&0xff);
    i++;
    if( !(i%22) )
       j++;
   }
con_getch();
}

/***-------------------------------------------------------------------------

  Add_rem_ch_grp

    ---------------------------------------------------------------------***/
static void add_rem_ch_grp(int type)
{
int    bd, ch, loop,
       response,
       g, endloop = 0;
short  cbuff[2];
char   *file = ( type == ADD ) ? "addchgrp.txt" : "remchgrp.txt";
FILE   *fp;
time_t t1, t2;

do
  {
   clrscr();
   con_printf("\n\n Input Group Number [1 .. 15]: ");
   con_scanf("%d",&g);
   if( g >= 1 && g < 16 )
      endloop = 1;
  }
while( !endloop );
```

```
if( ( fp = fopen(file,"r") ) == NULL )
  {
   con_printf("\n Add_rem_ch_grp: Problems opening file %s\n",file);
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }

code = ( type == ADD ) ? ADD_CH_GRP : REM_CH_GRP;

/*
  File Format:
  0  10
  2  1
  3  3
  3  4
  3  5
  -1
  Where the first number is the board and the second one is the channel
*/
endloop = loop = 0;
time(&t1);
do
  {
   fscanf(fp,"%d",&bd);
   if( bd == -1 )
      endloop = 1;
   else
     {
      fscanf(fp,"%d",&ch);
      cbuff[0] = g;
      cbuff[1] = (bd<<8) | ch;
      do
        {
         if(    ( response = caenet_comm(cbuff,sizeof(cbuff),NULL ) ) != TUTTOK
             && ( response != -256 ) )
             con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
         if( response == -256 )
            delay(100);
        }
      while( response != TUTTOK );
      loop++;
     }
  }
while( !endloop );
time(&t2);
con_printf("\n %s %d Channels %s Group %d: %ld seconds elapsed\n",
        (type==ADD)?"Added":"Removed",loop,
        (type==ADD)?"to":"from",g,t2-t1);
con_getch();
fclose(fp);
}

/***---------------------------------------------------------------------------

  Add_rem_all_grp

    ---------------------------------------------------------------------***/
static void add_rem_all_grp(int type)
{
int    bd, ch, loop,
       response,
       g, endloop = 0;
short  cbuff[2];
char   *file = ( type == ADD ) ? "addchgrp.txt" : "remchgrp.txt";
FILE   *fp;
time_t t1, t2;

code = ( type == ADD ) ? ADD_CH_GRP : REM_CH_GRP;

if( ( fp = fopen(file,"r") ) == NULL )
```

```
  {
   con_printf("\n Add_rem_ch_grp: Problems opening file %s\n",file);
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }

for( g = 1 ; g < 16 ; g++ )
   {
/*
  File Format:
  0  10
  2  1
  3  3
  3  4
  3  5
  -1
  Where the first number is the board and the second one is the channel
*/
    endloop = loop = 0;
    time(&t1);
    fseek(fp,0L,SEEK_SET);
    do
     {
      fscanf(fp,"%d",&bd);
      if( bd == -1 )
         endloop = 1;
      else
        {
         fscanf(fp,"%d",&ch);
         cbuff[0] = g;
         cbuff[1] = (bd<<8) | ch;
         do
           {
            if(    ( response = caenet_comm(cbuff,sizeof(cbuff),NULL ) ) != TUTTOK
                && ( response != -256 ) )
              {
               con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
               con_puts(" Press any key to continue ");
               con_getch();
               break;
              }
            if( response == -256 )
               delay(100);
           }
         while( response != TUTTOK );
         loop++;
        }
     }
   while( !endloop );
   time(&t2);
   con_printf("\n %s %d Channels %s Group %d: %ld seconds elapsed\n",
           (type==ADD)?"Added":"Removed",loop,
           (type==ADD)?"to":"from",g,t2-t1);

  }

con_getch();
fclose(fp);
}

/***---------------------------------------------------------------------------

  Logga

    ---------------------------------------------------------------------***/
static void logga(char *fmt, ...)
{
va_list  argptr;
time_t   t;
FILE     *fp;
```

```
time(&t);

fp = fopen(logfile,"a+");
fprintf(fp,ctime(&t));
va_start(argptr,fmt);
vfprintf(fp,fmt,argptr);
fclose(fp);
va_end(argptr);
}

/***---------------------------------------------------------------------------

  Mon_ch_grp
  Continously reads group data from crate 1 and 2

    ---------------------------------------------------------------------***/
static void mon_ch_grp(void)
{
FILE                *fp;
unsigned long       i;
short               group;
int                 response,
                    g, endloop = 0;
static struct hvrd  chrd[320];

do
  {
   clrscr();
   con_printf("\n\n Input Group Number [0 .. 15]: ");
   con_scanf("%d",&g);
   if( g >= 0 && g < 16 )
      endloop = 1;
  }
while( !endloop );

con_printf("\n\n Input Log File Name: ");
con_scanf("%s",logfile);

if( ( fp = fopen(logfile,"w") ) == NULL )
  {
   con_printf("\n Problems opening %s\n",logfile);
   con_getch();
   return;
  }
else
   fclose(fp);

code = MON_CH_GRP;
group = g;

con_printf("\n\n Reading loop, press any key to exit ... ");
logga("Start of test\n\n");

for( i = 0 ; !con_kbhit() ; i++ )
   {
    gotoxy(1,12);
    con_printf(" Iteration Nr. %ld",i);
   cratenum = 1;
   if(( response = caenet_comm(&group,sizeof(group),chrd) ) != TUTTOK )
     {
      gotoxy(1,14);
     con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
      logga("Caenet_comm: Crate %d, %s\n\n",cratenum,A303DecodeResp(response));
      delay(500);
     }
    else
     {
      gotoxy(1,14);
      con_printf("                                                                ");
     }
```

```
   cratenum = 2;
   if(( response = caenet_comm(&group,sizeof(group),chrd) ) != TUTTOK )
      {
       gotoxy(1,14);
       con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
        logga("Caenet_comm: Crate %d, %s\n\n",cratenum,A303DecodeResp(response));
        delay(500);
      }
   else
    {
     gotoxy(1,14);
    con_printf("                                                                ");
    }
   } /* end for !con_kbhit */

 logga("End of test\nNumber of Iterations: %ld\n",i);
}

/***--------------------------------------------------------------------------

  Grpmenu

    ----------------------------------------------------------------------***/
static void grpmenu(void)
{
for(;;)
    switch(makegrpmenu())
       {
     case 'A':
        add_rem_ch_grp(ADD);
        break;
     case 'B':
        add_rem_ch_grp(REM);
        break;
     case 'C':
        add_rem_all_grp(ADD);
        break;
     case 'D':
        add_rem_all_grp(REM);
        break;
     case 'E':
        list_ch_grp();
        break;
     case 'F':
        par_set(GRP);
        break;
     case 'G':
        mon_ch_grp();
        break;
     case 'Q':
        return;
     default:
        break;
       }
}

/***--------------------------------------------------------------------------

  End

    ----------------------------------------------------------------------***/
static void end(void)
{
 A303End();
 con_end();
 exit(0);
}

/***--------------------------------------------------------------------------
```

```
  Main Program

    ------------------------------------------------------------------------***/
void main(int argc,char *argv[])
{
 int    response;
 ulong  a303addr;
 ulong  timeout = 100;   // Corresponds to 1 sec

  if(argc != 3)
   {
    puts(" Usage: sy527demo <A303 I/O Base address (in hex)>              ");
    puts("                  <sy527 Caenet number (in hex)>                ");
    exit(0);
   }

 con_init();

 sscanf(argv[1],"%x",&a303addr);
 sscanf(argv[2],"%x",&cratenum);

/*
   This is the first CAENET routine to call !!
*/
if( ( response = A303Init(a303addr) ) != TUTTOK )
   {
    clrscr();
    con_printf(" A303Init failed: %s\n",A303DecodeResp(response));
    con_puts(" Press any key to continue ");
    con_getch();
    end();
   }

if( ( response = A303Reset() ) != TUTTOK )
   {
    clrscr();
    con_printf(" A303Reset failed: %s\n",A303DecodeResp(response));
    con_puts(" Press any key to continue ");
    con_getch();
    end();
   }

if( ( response = A303Timeout(timeout) ) != TUTTOK )
  {
    clrscr();
    con_printf(" A303Timeout failed: %s\n",A303DecodeResp(response));
    con_puts(" Press any key to continue ");
    con_getch();
    end();
  }

/*
  Main Loop
*/
for(;;)
    switch(makemenu())
       {
     case 'A':
       read_ident();
        break;
     case 'B':
        crate_map();
        break;
     case 'C':
        ch_monitor();
        break;
     case 'D':
        speed_test();
        break;
     case 'E':
        par_set(NO_GRP);
```

```
            break;
          case 'F':
             clear_alarm();
             break;
          case 'G':
             set_status_alarm();
             break;
          case 'H':
             lock_keyboard();
             break;
          case 'I':
             unlock_keyboard();
             break;
          case 'J':
             kill_channels();
             break;
          case 'K':
             fpan_stat();
             break;
          case 'L':
             grpmenu();
             break;
         case 'M':
                      format_eeprom();
             break;
         case 'Q':
             end();
             break;
          default:
             break;
            }
    }
```

2.6.3. SY127demo.c

```
/*************************************************************************/
/*                                                                       */
/*        --- CAEN Engineering Srl - Computing Systems Division ---      */
/*                                                                       */
/*    SY127DEMO.C                                                        */
/*                                                                       */
/*    Demonstration about the use of Caenet Routines in communication    */
/*    between A303/A module and SY127 High Voltage System equipped with  */
/*    module A128A High Speed Caenet Communication Controller            */
/*                                                                       */
/*    Source code written in Ansi C                                      */
/*    Must be linked to console.c and to a303.c                          */
/*                                                                       */
/*    Created: January 2000                                              */
/*                                                                       */
/*************************************************************************/
#include  <stdio.h>
#include  <string.h>
#include  <ctype.h>
#include  <stdlib.h>
#include  "console.h"
#include  "a303.h"

#ifndef   uchar
#define   uchar                       unsigned char
#endif
#ifndef   ushort
#define   ushort                     unsigned short
#endif
#ifndef   ulong
#define   ulong                     unsigned long
#endif
```

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

#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   ON_OFF                                   8
#define   CHNAME                                   9

#define   MAKE_CODE(ch,cod)      (((ch)<<8) | (cod))

/*
  Some of the Caenet Codes
*/
#define   IDENT                                 0x0
#define   READ_CH                               0x1
#define   READ_SETTINGS                         0x2
#define   CRATE_MAP                             0x3
#define   READ_PROT                             0x4

#define   FORMAT_EEPROM_1                      0x30
#define   FORMAT_EEPROM_2                      0x31
#define   CLEAR_ALARM                          0x32
#define   SET_PROT                             0x39

/*
  The following structure contains all the useful information about
  the settings and monitorings of a channel
*/
struct hvch
{
 ushort  v0set;
 ushort  v1set;
 ushort  i0set;
 ushort  i1set;
 ushort  rup;
 ushort  rdwn;
 ushort  trip;
 ushort  status;
 ushort  gr_ass;
 ushort  vread;
 ushort  iread;
 ushort  st_phase;
 ushort  st_time;
 ushort  mod_type;
 char    chname[10];
};

/*
  The following structure contains all the useful information about
  the protection word of SY127
*/
struct prot_w
{
 unsigned  pwon    :1;
 unsigned  pswen   :1;
 unsigned  keyben  :1;
 unsigned  alarms  :1;
 unsigned  unused  :12;
};

/*
  Globals
*/
static int            cratenum, code;
```

```
static struct prot_w  protect;

static char *modul_type[] =                     /* Modules types */
{
  " Not Present     ",
  " 2KV    3mA      ",
  " 3KV    3mA      ",
  " 4KV    2mA      ",
  " 8KV    500uA    ",
  " 6KV    1mA      ",
  " 800.0V 500.0uA  ",
  " 8KV    200.0uA  ",
  " 6KV    200.0uA  ",
  " 200.0V 200.0uA  ",
  " 2KV    200.0uA  ",
  " 4KV    200.0uA  ",
  " 6KV    1mA      ",
  " Not Implemented ",
  " 3KV    3mA      ",
  " 4KV    2mA      ",
  " 800.0V 200.0uA  ",
  " 8KV    400.0uA  ",
  " 8KV    200.0uA  ",
  " 10KV   1mA      ",
  " 2KV    6mA      ",
  " 800.0V 400.0uA  ",
  " 10KV   200.0uA  ",
  " 15KV   200.0uA  ",
  " 15KV   1mA      ",
  " 20KV   200.0uA  ",
  " 2.5KV  5mA      ",
  " 1KV    10mA     ",
  " 1KV    200.0uA  ",
  " 20KV   500uA    ",
  " 10KV   2mA      ",
  " I/O Module      ",
  " 200.0V 40uA     ",
  " 800.0V 40uA     ",
  " 2KV    40uA     ",
  " 4KV    40uA     ",
  " 6KV    40uA     ",
  " 8KV    40uA     ",
  " 10KV   40uA     ",
  " 15KV   40uA     ",
  " 20KV   40uA     ",
  " Not Implemented ",
  " Not Implemented ",
  " Not Implemented ",
  " Not Implemented ",
  " Special Module  ",
  " Not Implemented ",
  " Not Implemented ",
};


/***-------------------------------------------------------------------------

  Caenet_comm
  It is a generic wrapper which permits every type of Caenet operation
  with SY127.
  Note that in this example we don't worry about the second parameter of
  A303ReadResponse which indicates the number of bytes read from Caenet

    -------------------------------------------------------------------***/
static int caenet_comm(void *source_buff, int wr_byte_count, void *dest_buff)
{
 int resp, dummy;

 resp = A303SendCommand( code, cratenum, source_buff, wr_byte_count );
 if( resp != TUTTOK )
    return resp;
```

```
 else
   {
    return A303ReadResponse(dest_buff, &dummy);
   }
}

/***-------------------------------------------------------------------------

  Makemenu

    -------------------------------------------------------------------***/
static int makemenu(void)
{


clrscr();
highvideo();
con_puts("                    - MAIN MENU -      \n\n\n ");
normvideo();
con_puts(" [A] - Read Module Identifier ");
con_puts(" [B] - Board  0  Monitor ");
con_puts(" [C] - Board  1  Monitor ");
con_puts(" [D] - Board  2  Monitor ");
con_puts(" [E] - Board  3  Monitor ");
con_puts(" [F] - Board  4  Monitor ");
con_puts(" [G] - Board  5  Monitor ");
con_puts(" [H] - Board  6  Monitor ");
con_puts(" [I] - Board  7  Monitor ");
con_puts(" [J] - Board  8  Monitor ");
con_puts(" [K] - Board  9  Monitor ");
con_puts(" [L] - Speed test              ");
con_puts(" [M] - Parameter Set           ");
con_puts(" [N] - Crate Map         ");
con_puts(" [O] - Format EEPROM           ");
con_puts(" [P] - Clear Alarms            ");
con_puts(" [R] - Set Protections         ");
con_puts("\n\n [Q] - Quit ");

return toupper(con_getch());
}

/***-------------------------------------------------------------------------

  Read_Ident

    -------------------------------------------------------------------***/
static void read_ident(void)
{
int i,response;
char sy127ident[12];
char tempbuff[80];
code=IDENT;                                  /* To see if sy127 is present */
if((response=caenet_comm(NULL,0,tempbuff)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }
for(i=0;i<10;i++)
    sy127ident[i]=tempbuff[2*i];
sy127ident[i]='\0';
con_printf(" The module has answered : %s\n",sy127ident);
con_puts(" Press any key to continue ");
con_getch();
}

/***-------------------------------------------------------------------------

  Crate_Map
```

```
    ------------------------------------------------------------------------***/
static void crate_map(void)
{
int  bd, response;
char cm[10];

code=CRATE_MAP;
if((response=caenet_comm(NULL,0,cm)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }

clrscr();
con_puts("\n\n                          ---  Crate Map ---       \n\n\n\n\n ");

for( bd = 0 ; bd < 10 ; bd++ )
   {
    char *type;

    if( cm[bd] == 0 )
       type = " ";
    else if( cm[bd] > 0 )
       type = " Positive ";
    else
       type = " Negative ";
    con_printf(" Slot %d - ",bd);
    con_printf(" %s %s\n",modul_type[cm[bd] & 0x7f], type);
   }
con_puts("\n\n\n   Press any key to continue ");
con_getch();
}

/***-------------------------------------------------------------------------

  Build_Chrd_Info

    ------------------------------------------------------------------------***/
static void build_chrd_info(struct hvch *ch, char *cnetbuff)
{
int i = sizeof(ch->v0set);

memcpy(&ch->v0set, cnetbuff,   sizeof(ch->v0set));
memcpy(&ch->v1set, cnetbuff+i, sizeof(ch->v1set));
i += sizeof(ch->v1set);
memcpy(&ch->i0set, cnetbuff+i, sizeof(ch->i0set));
i += sizeof(ch->i0set);
memcpy(&ch->i1set, cnetbuff+i, sizeof(ch->i1set));
i += sizeof(ch->i1set);
memcpy(&ch->rup, cnetbuff+i, sizeof(ch->rup));
i += sizeof(ch->rup);
memcpy(&ch->rdwn, cnetbuff+i, sizeof(ch->rdwn));
i += sizeof(ch->rdwn);
memcpy(&ch->trip, cnetbuff+i, sizeof(ch->trip));
i += sizeof(ch->trip);
memcpy(&ch->status, cnetbuff+i, sizeof(ch->status));
i += sizeof(ch->status);
memcpy(&ch->gr_ass, cnetbuff+i, sizeof(ch->gr_ass));
i += sizeof(ch->gr_ass);
memcpy(&ch->vread, cnetbuff+i, sizeof(ch->vread));
i += sizeof(ch->vread);
memcpy(&ch->iread, cnetbuff+i, sizeof(ch->iread));
i += sizeof(ch->iread);
memcpy(&ch->st_phase, cnetbuff+i, sizeof(ch->st_phase));
i += sizeof(ch->st_phase);
memcpy(&ch->st_time, cnetbuff+i, sizeof(ch->st_time));
i += sizeof(ch->st_time);
memcpy(&ch->mod_type, cnetbuff+i, sizeof(ch->mod_type));
i += sizeof(ch->mod_type);
```

```
memcpy(&ch->chname, cnetbuff+i, sizeof(ch->chname));
}

/***--------------------------------------------------------------------------

  Ch_monitor

    --------------------------------------------------------------------***/
static void ch_monitor(int group)
{
int                     i,
                        caratt='P',
                        response,
                        chs=group*4;
char                    cnetbuff[MAX_LENGTH_FIFO];
static float            pow10[]={ 1.0, 10.0, 100.0};
float                   scalei,scalev;
ushort                  channel;
static int              page=0;
static struct hvch      ch_read[4];      /* Four channel each board */

scalev=1.0;
scalei=1.0;

clrscr();
highvideo();
if(!page)
   con_puts
(" Channel     Vmon     Imon    V0set    I0set    V1set    I1set   Groups  Ch#
");
else
   con_puts
(" Channel     Rup    Rdwn     Trip     Status    STPhase STTime ModType Ch# ");
normvideo();

gotoxy(1,23);
con_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<4;i++)
         {
          channel=(uchar)(chs+i);
          code=MAKE_CODE(channel,READ_CH);
          response = caenet_comm(NULL,0,cnetbuff);
          if( response != TUTTOK )
            {
             gotoxy(1,22);
             con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
             con_puts(" Press any key to continue                          ");
             con_getch();
             return;
            }
          build_chrd_info(&ch_read[i], cnetbuff);
         }

/* Display the information */
      if(!page)                                    /* Page 0 of display */
         for(i=0;i<4;i++)
            {
             gotoxy(1,i+5);
             con_printf(" %s",ch_read[i].chname);
             gotoxy(12,i+5);
             con_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_read[i].v0set/scalev,
ch_read[i].i0set/scalei,ch_read[i].v1set/scalev,ch_read[i].i1set/scalei,
ch_read[i].gr_ass,chs+i);
            }
      else                                         /* Page 1 of display */
```

```
         for(i=0;i<4;i++)
            {
             gotoxy(1,i+5);
             con_printf(" %s",ch_read[i].chname);
             gotoxy(14,i+5);
             con_printf
("%3d     %3d    %4d       %4x     %4x    %4x   %4x    %2d\n",
ch_read[i].rup,ch_read[i].rdwn,ch_read[i].trip,
ch_read[i].status,ch_read[i].st_phase,ch_read[i].st_time,ch_read[i].mod_type,chs+
i);
            }

/* Test the keyboard */
      if( con_kbhit() )                        /* A key has been pressed */
         if((caratt=toupper(con_getch())) == 'P') /* They want to change page */
           {
            highvideo();
            page = !page;
            clrscr();
            if(page == 0)
               con_puts
(" Channel     Vmon    Imon    V0set    I0set    V1set    I1set    Groups  Ch#
");
            else
               con_puts
(" Channel     Rup    Rdwn     Trip     Status    STPhase STTime ModType Ch# ");
            normvideo();
            gotoxy(1,23);
            con_puts(" Press 'P' to change page, any other key to exit ");
           }

     }  /* End while */
}

/***---------------------------------------------------------------------------

  Par_set

    ---------------------------------------------------------------------***/
static void par_set(void)
{
float        input_value,
             scale;
static float pow10[] =  { 1.0, 1.0, 1.0};           /* Per ora scale = 1 ... */
ushort       channel,value;
int          i,
             response,
             par=0;
char         choiced_param[10], chname[12];
static char  *param[] =
{
 "v0set", "v1set", "i0set", "i1set", "vmax",
 "rup", "rdwn", "trip", "on/off","name", NULL
};

clrscr();
con_printf("\n\n Channel: ");                    /* Choice the channel */
con_scanf("%d",&i);
channel=(uchar)i;
con_puts(" Allowed parameters (lowercase only) are:");
for( i=0 ; param[i] != NULL ; i++ )
    con_puts(param[i]);
while(!par)
     {
      con_printf("\n Parameter to set: ");        /* Choice the parameter */
      con_scanf("%s",choiced_param);
      for( i=0 ; param[i] != NULL ; i++ )
          if(!strcmp(param[i],choiced_param))
            {
             par=1;
             break;
```

```
            }
        if(param[i] == NULL)
           con_puts(" Sorry, this parameter is not allowed");
       }
con_printf(" New value :");                        /* Choice the value         */
if(i == CHNAME)
  {
   con_puts(" Function not yet implemented ");
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }
else
   con_scanf("%f",&input_value);

switch(i)                                          /* Decode the par.          */
  {
   case V0SET:
       code=MAKE_CODE(channel,16);
       scale=pow10[0];
       input_value*=scale;
       value=(ushort)input_value;
       break;
   case V1SET:
       code=MAKE_CODE(channel,17);
       scale=pow10[0];
       input_value*=scale;
       value=(ushort)input_value;
       break;
   case I0SET:
       code=MAKE_CODE(channel,18);
       scale=pow10[0];
       input_value*=scale;
       value=(ushort)input_value;
       break;
   case I1SET:
       code=MAKE_CODE(channel,19);
       scale=pow10[0];
       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;
   case  ON_OFF:
       code=MAKE_CODE(channel,24);
       value=(ushort)input_value;
       break;
   case CHNAME:
       code=MAKE_CODE(channel,25);
       break;
  }

if(i == CHNAME)
  {
   if((response=caenet_comm(chname,sizeof(chname),NULL)) != TUTTOK)
     {
```

```
      con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
      con_puts(" Press any key to continue ");
      con_getch();
     }
  }
else
  {
   if((response=caenet_comm(&value,sizeof(ushort),NULL)) != TUTTOK)
     {
      con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
      con_puts(" Press any key to continue ");
      con_getch();
     }
  }
}

/***-------------------------------------------------------------------------

  Speed_test

    -------------------------------------------------------------------------***/
static void speed_test(void)
{
int i,response;
char sy127ident[12],loopdata[12];
char tempbuff[80];
code=IDENT;                                    /* To see if sy127 is present */
if((response=caenet_comm(NULL,0,tempbuff)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return;
  }
for(i=0;i<10;i++)
    sy127ident[i]=tempbuff[2*i];
sy127ident[i]='\0';

con_puts(" Looping, press any key to exit ... ");
/* Loop until one presses a key */
while(!con_kbhit())
     {
      if((response=caenet_comm(NULL,0,tempbuff)) != TUTTOK)
        {
         con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
         con_puts(" Press any key to continue ");
         con_getch();
         return;
        }

      for(i=0;i<10;i++)
          loopdata[i]=tempbuff[2*i];
      loopdata[i]='\0';
      if(strcmp(sy127ident,loopdata))    /* Data read in loop are not good */
        {
         con_printf(" Test_loop error: String read = %s\n",loopdata);
         con_puts(" Press any key to continue ");
         con_getch();
         return;
        }
     }  /* end while */
con_getch();
}

/***-------------------------------------------------------------------------

  Format_EEPROM

    -------------------------------------------------------------------------***/
static void format_eeprom(void)
{
```

```
int c, response;

clrscr();
gotoxy(2,9);
con_printf("FORMAT EEPROM. Are you sure ? (Y/N) [N]: ");
for(;;)
   {
    c = tolower(con_getch());
    if( c == 'y' || c == 'n' || c == CR )
       break;
   }
if( c == 'n' || c == CR )
   return;

con_putch('Y');

code = FORMAT_EEPROM_1;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }

con_printf("\n\n Executing ... \n");

code = FORMAT_EEPROM_2;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }
}

/***-------------------------------------------------------------------------

  Clear_Alarm

    -------------------------------------------------------------------------***/
static void clear_alarm(void)
{
int response;

code = CLEAR_ALARM;
if((response=caenet_comm(NULL,0,NULL)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
  }
}

/***-------------------------------------------------------------------------

  Print_Power_On

    -------------------------------------------------------------------------***/
static void print_power_on(void)
{
gotoxy(31,11);
if(protect.pwon)
   con_printf("Enabled ");
else
   con_printf("Disabled");
}

/***-------------------------------------------------------------------------

  Print_Passw
```

```
    ------------------------------------------------------------------------***/
static void print_passw(void)
{
gotoxy(31,12);
if(protect.pswen)
   con_printf("Enabled ");
else
   con_printf("Disabled");
}

/***--------------------------------------------------------------------------

  Print_Keyb

    ------------------------------------------------------------------------***/
static void print_keyb(void)
{
gotoxy(31,13);
if(protect.keyben)
   con_printf("Enabled ");
else
   con_printf("Disabled");
}

/***--------------------------------------------------------------------------

  Print_Alarms

    ------------------------------------------------------------------------***/
static void print_alarms(void)
{
gotoxy(31,15);
if(protect.alarms)
   con_printf("Active   ");
else
   con_printf("No Active");
}

/***--------------------------------------------------------------------------

  Prot_Menu

    ------------------------------------------------------------------------***/
int  prot_menu(void)
{
int      response;
short    pr;

clrscr();

code = READ_PROT;
if((response=caenet_comm(NULL,0,&pr)) != TUTTOK)
  {
   con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
   con_puts(" Press any key to continue ");
   con_getch();
   return 0;
  }

pr &= 7;

memcpy(&protect,&pr,sizeof(short));

gotoxy(7,9);
highvideo();
con_printf("Select Protections");
normvideo();

gotoxy(1,11);
con_printf("      A)  Power ON    :");
print_power_on();
```

```
gotoxy(1,12);
con_printf("      B)  Password    :");
print_passw();

gotoxy(1,13);
con_printf("      C)  Keyboard    :");
print_keyb();

gotoxy(1,15);
con_printf("          Alarms      :");
print_alarms();

gotoxy(1,17);
con_printf("      Q)  Quit");

gotoxy(7,23);
con_printf("Select item\r\n");

return 1;
}

/***-------------------------------------------------------------------------

  Protections

    ------------------------------------------------------------------------***/
void protections(void)
{
int c, modified, response;

if(!prot_menu())
   return;

while(1)
  {
   modified = 0;

   c = tolower(con_getch());

   switch (c)
     {
      case  'a' : protect.pwon = !protect.pwon;
                  print_power_on();
                  modified = 1;
                  break;

      case  'b' : protect.pswen = !protect.pswen;
                  print_passw();
                  modified = 1;
                  break;

      case  'c' : protect.keyben = !protect.keyben;
                  print_keyb();
                  modified = 1;
                  break;
     }  /* end switch */

   if(modified)
     {
      code = SET_PROT;
      if((response=caenet_comm(&protect,sizeof(short),NULL)) != TUTTOK)
        {
         con_printf(" Caenet_comm: %s\n",A303DecodeResp(response));
         con_puts(" Press any key to continue ");
         con_getch();
         return;
        }
      modified = 0;
     }
   if(c == 'q')
```

```
      break;

   gotoxy(1,24);
  }     /* end while(1) */
}

/***--------------------------------------------------------------------------

  Esci

    ----------------------------------------------------------------------***/
void end(void)
{
clrscr();
A303End();
con_end();
exit(0);
}

/***--------------------------------------------------------------------------

  Main Program

    ----------------------------------------------------------------------***/
void main(int argc, char **argv)
{
 int    c, response;
 ulong  a303addr;
 ulong  timeout = 100;   // Corresponds to 1 sec

 if(argc != 3)
   {
    puts(" Usage: sy127demo <A303 I/O Base address (in hex)>              ");
    puts("                  <sy127 Caenet number (in hex)>                ");
    exit(0);
   }

 sscanf(argv[1],"%x",&a303addr);
 sscanf(argv[2],"%x",&cratenum);

 con_init();

/*
   This is the first CAENET routine to call !!
*/
  if( ( response = A303Init(a303addr) ) != TUTTOK )
   {
    con_printf(" A303Init failed: %s\n",A303DecodeResp(response));
    exit(0);
   }

if( ( response = A303Reset() ) != TUTTOK )
   {
    con_printf(" A303Reset failed: %s\n",A303DecodeResp(response));
    end();
   }

if( ( response = A303Timeout(timeout) ) != TUTTOK )
  {
    con_printf(" A303Timeout failed: %s\n",A303DecodeResp(response));
    end();
  }

/*
  Main Loop
*/
for(;;)
    switch(c = makemenu())
       {
     case 'A':
        read_ident();
```

```
            break;
         case 'B':
         case 'C':
         case 'D':
         case 'E':
         case 'F':
         case 'G':
         case 'H':
         case 'I':
         case 'J':
         case 'K':
            ch_monitor(c-'B');
            break;
         case 'L':
            speed_test();
            break;
         case 'M':
            par_set();
            break;
         case 'N':
            crate_map();
            break;
         case 'O':
            format_eeprom();
            break;
         case 'P':
            clear_alarm();
            break;
         case 'R':
            protections();
            break;
         case 'Q':
            end();
            break;
         default:
            break;
          }
}
```

3. *Technical specifications*

3.1. Packaging

Standard PC ISA card

3.2. External Components

CONNECTORS

- 2 LEMO 00 type communication line connectors, 50 Ω impedance.

DISPLAYS

- H.S. CAENET active LED

3.3. Internal Components

(refer to Fig. 2.1)

SWITCHES

- 4 rotary switches SW1..4:

SW 1..4: These are dedicated to the module base address selection:
SW1 : Base address [19..16]
SW2 : Base address [15..12]
SW3 : Base address [11..8]
SW4 : Base address [7..4]

- 4 dip switches SW5..7:

SW 5: For the internal termination of the H.S. CAENET line
ON : internal line termination
OFF : no internal line termination

SW 6 "INT": Enables the operating mode selection:
ON : Interrupt mode
OFF : Polling mode

SW 6 "IOCHRDY": Controls the generation of the signal IOCHRDY on the bus.
ON : IOCHRDY connected
OFF : no IOCHRDY connected

SW 7: Dedicated to the addressing type selection:
ON : Memory space addressing
OFF : I/O space addressing

3.4. Physical Line and Node Capabilities

Output Driver Signal characteristics:	0 to +4 V on 50 Ω impedance
Input Receiver characteristics:	+1.2 V threshold discriminator
RG174 Cable attenuation:	-6.2 dB/100 m @ 1 MHz
H. S. CAENET Node attenuation:	≈ -0.07 dB @ 500 kHz

It is worth noting that the maximum frequency of the H. S. CAENET signal is 500 kHz (a stream of bits toggling between 0 and 1 transmitted at the rate of 1 MBaud).
Through the use of appropriate adapters, one can connect the H. S. CAENET nodes via an RG58 cable, that has better attenuation features (-1.4 dB/100 m @ 1 MHz) at approximately the same cost.

3.5. Power Requirements

+ 5 V 0.65 A

3.6. Default Settings

The card is shipped with the following default settings:

I/O space addressing: ON
I/O address: 380h
Interrupt generation: OFF
IOCHRDY: OFF
LINE TERMINATION: ON

4. *Mod. A 303A hardware description*

4.1. H.S. CAENET Network Operations

H.S. CAENET Network is a send and receive half duplex system; it permits asyncronous serial transmission (1Mbaud rate) of data packet along a simple 50 Ω coaxial cable. Several devices (H.S. CAENET nodes) are able to share the same line to transmit and receive data.
Each node is able to receive the serial data packet and store it automatically in the RX FIFO and transmit the data contained in the TX FIFO. Both FIFOs are 4096 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 4096 bytes.

4.2. H.S. CAENET Node Operation

Basically an H.S. CAENET node can work in 3 distinct modes: Transmit, Receive and Restart.

- in the Transmit mode the node accesses the data stored in the TX FIFO and transmits the data on the cable.
- in the Receive mode the serial packet is stored in the RX FIFO.
- in Restart mode the node does not accept any commands, all the TX and RX buffers are cleared and the interrupt is removed; it remains in this mode until the line is cleared.

The A 303A card directly interfaces the 8 bit PC ISA bus with the two FIFO buffers (TX and RX FIFO see Figure 1.1), and with 6 internal registers which are used for various functions such as FIFOs clearing, starting transmission and reading the node status.
The Host processor can control the node operation in Pollling or Interrupt mode.
Interrupts to the CPU are generated by the A303A:

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

4.3. Registers and Buffers Addressing

The address map of the module A 303A is shown in the following Figure (each location is 8 bit wide)

Register/Buffer	Operation	Address
TX FIFO	WR	BASE ADDRESS + 0
START TX	WR	BASE ADDRESS + 1
LED	WR	BASE ADDRESS + 2
RESET	WR	BASE ADDRESS + 3
RX FIFO	RD	BASE ADDRESS + 0
STATUS REGISTER	RD	BASE ADDRESS + 1
RESET INTERRUPT (rd STATUS)	RD	BASE ADDRESS + 2
CLEAR RX FIFO	RD	BASE ADDRESS + 3

Table 4.1: Address map of Model A303A

4.3.1. TX FIFO

(Base address + 0, write only)

This is the buffer which is loaded with the data to be transmitted; it is arranged in a FIFO logic. By writing at this location the H.S. CAENET active LED lights up.

1.1.1. START TX

(Base address + 1, write only)

By writing at this location the node enters into the transmit mode; and the H.S.CAENET LED turns on.

1.1.2. LED

(Base address +2, write only)

By writing at this location the H.S. CAENET active LED turns on.

1.1.3. RESET

(Base address + 3, write only)

A write access to this location causes the node to enter Restart mode; this causes the following operations:

– the buffers TX FIFO and RX FIFO are cleared

– every interrupt pending is cleared
– every data transfer is aborted
– the H.S.CAENET LED turns off
– the node does not accept any command

It remains in this status until the line is cleared.

1.1.4. RX FIFO

(Base address + 0, read only)

This is the buffer where the node automatically stores the received data; it is arranged in a FIFO logic.

1.1.5. STATUS REGISTER

(Base address + 1, read only)

This register contains the status bits of the H.S. CAENET node; in particular, the cause of the interrupt can be ascertained by reading the STATUS REGISTER (see § 3.6 for the status bits description).

4.3.2. RESET INTERRUPT (rd STATUS)

(Base address + 2, read only)

At this address the status register previously described is available, moreover a read access causes the following operations:

– the H.S. CAENET interrupt is removed (if asserted)
– the three interrupt status bits RFEFF, RXEFF and TXEFF are reset to 1
– the H.S. CAENET LED turns off.

4.3.3. CLEAR RX FIFO

(Base address + 3, read only)

A read access to this address clears the receive buffer RX FIFO.

4.4. H.S.CAENET LED

The H.S. CAENET LED turns on in these cases:

– after a write access to Base address + 2 (LED);
– after a write access to Base address + 0 (TX FIFO); i.e. when data are stored in the TX FIFO;
– after a write access to Base address + 1 (START TX);

It remains on until one of the following operations is performed:

– a write access to Base address + 3 (RESET);
– a read access to Base address + 2 (RESET INTERRUPT and read STATUS);

4.5. Status Register Definition

The following figure shows the status register structure:

7	6	5	4	3	2	1	0
TXACT	RXACT	TXEFF	TFEM	RESTART	RXEFF	RFEFF	RXFEM

Table 4.2: Status register structure

BIT	NAME	DESCRIPTION
0	RXFEM	RX FIFO EMPTY: IF =0 THE RX FIFO IS EMPTY. RXFEM IS SET TO 1 WHEN VALID DATA ARE PRESENT IN THE RX FIFO, AND IS CLEARED AFTER THE RX FIFO HAS BEEN UNLOADED. THIS BIT IS ALSO CLEARED AFTER A RESET OPERATION OR AFTER AN RX FIFO CLEAR.
1	RFEFF	RX FIFO EMPTY: IF =0 THE RX FIFO HAS BEEN COMPLETELY UNLOADED. RFEFF IS SET TO 0 AFTER THE LAST DATA OF THE RECEIVED PACKET HAS BEEN READ FROM THE RX FIFO; IF SW6 "INT" IS ON AN INTERRUPT IS GENERATED. RFEFF IS RESET TO 1 AFTER A RESET OPERATION OR AFTER A RESET INTERRUPT.
2	RXEFF	RX END: IF =0 THE H.S. CAENET NODE HAS RECEIVED A DATA PACKET. RXEFF IS SET TO 0 AT THE END OF THE RECEPTION; IF SW6 "INT" IS ON AN INTERRUPT IS GENERATED. RXEFF IS RESET TO 1 AFTER A RESET OPERATION OR AFTER A RESET INTERRUPT.
3	RESTART	RESTART MODE: IF =0 THE H.S. CAENET NODE IS IN RESTART MODE; RESTART IS SET TO 0 AFTER A RESET OPERATION, IN THIS STATE THE H.S.CAENET NODE DOES NOT ACCEPT ANY COMMAND. RESTART IS RESET TO 1 WHEN THE NODE DETECTS THAT THE LINE IS CLEAR.
4	TFEM	TX FIFO EMPTY: IF =0 THE TX FIFO IS EMPTY. TFEM IS SET TO 1 WHEN DATA ARE STORED IN THE TX FIFO, AND IS CLEARED WHEN THE TX FIFO HAS BEEN UNLOADED BY THE H.S. CAENET NODE OR AFTER A RESET OPERATION.
5	TXEFF	TX END FLIP FLOP: IF =0 THE H.S. CAENET NODE HAS TRANSMITTED A DATA PACKET. TXEFF IS SET TO 0 AT THE END OF THE TRANSMISSION; IF SW6 "INT" IS ON AN INTERRUPT IS GENERATED. TXEFF IS RESET TO 1 AFTER A RESET OPERATION OR AFTER A RESET INTERRUPT.
6	RXACT	RECEIVER ACTIVE: IF =0 THE H.S. CAENET IS IN RECEIVE MODE. RXACT IS SET TO 0 WHEN THE NODE STARTS TO RECEIVE A DATA PACKET. RXACT IS RESET TO 1 AT THE END OF RECEPTION OR AFTER A RESET OPERATION.
7	TXACT	TRANSMITTER ACTIVE: IF =0 THE H.S. CAENET IS IN TRANSMIT MODE. TXACT IS SET TO 0 AFTER A START TX OPERATION. TXACT IS RESET TO 1 AT THE END OF TRANSMISSION OR AFTER A RESET OPERATION.

Table 4.3: Status register description

4.6. Interrupt Generation

Interrupts to the CPU are generated by the A303A:

- after the transmission of a data packet has been completed: TXEFF goes to level = 0
- at the end of reception of a data packet: RXEFF goes to level = 0
- when the RX FIFO has been completely unloaded: RFEFF goes to level = 0

4.7. Interrupt Release

The A303A removes its interrupt and the corresponding interrupt bit (s) are reset to 1 after the following operation:

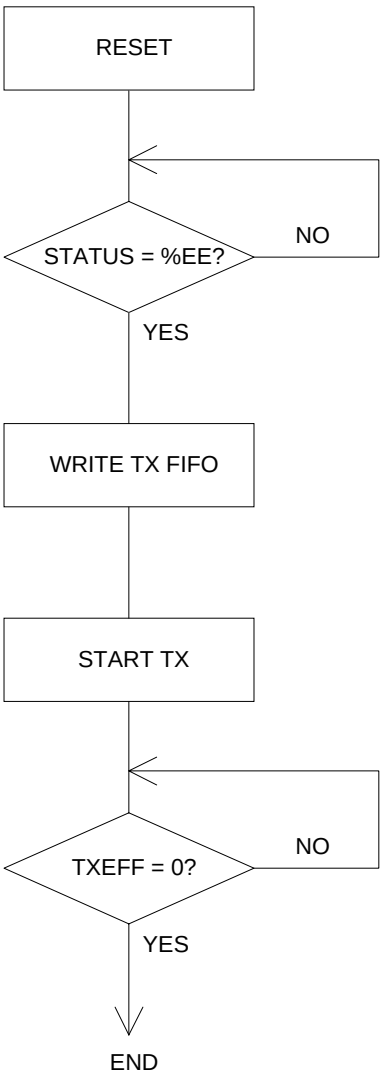
- read access to the RESET INTERRUPT location (Base address + 2).
- write access to the RESET location (Base address + 3).

5. *Developing control software*

This chapter is addressed to the User who wishes to develop his own device driver

5.1. Flow Charts

The following flow charts indicate respectively the Data Transmission routine and the Interrupt Handler of a generic A 303A device driver.



The described routines are shown in a simplified version. There are two distinct cases:

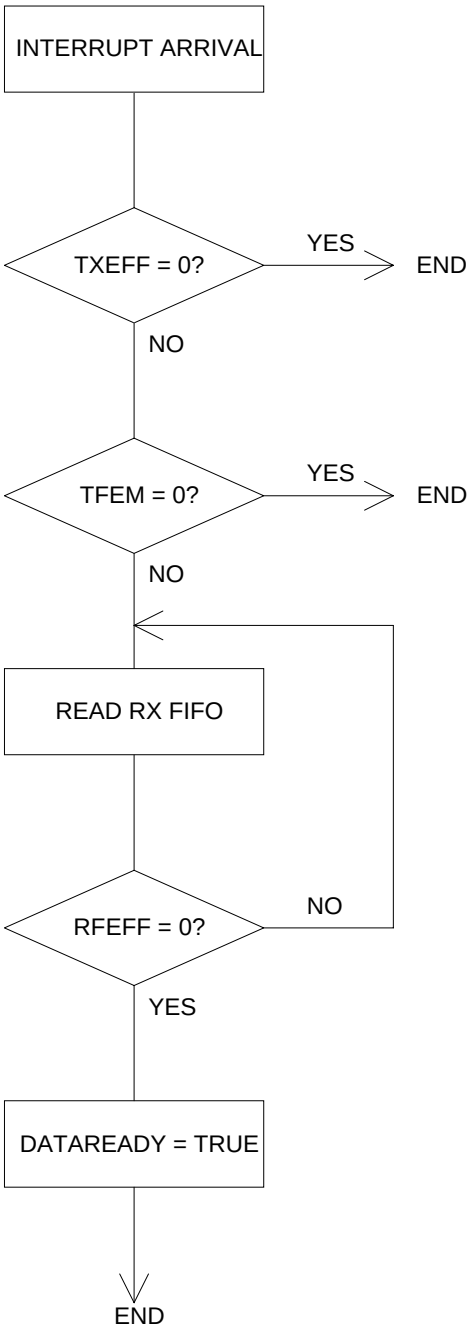
a) The caller is a H.S. CAENET MASTER.

At the exit from this routine, the caller must take care of starting a timer to be sure that the SLAVE answers within a reasonable time (e.g. a few msec). This routine represents a H.S. CAENET command.

b) The caller is an H.S. CAENET SLAVE.

At the exit from this routine, the caller has finished because the routine represents the answer to a previous H.S. CAENET command.

Fig. 5.1: Data Transmission Flow Chart



The driver must also have a Read Data routine (not shown here) that returns to the caller the data arrived via H. S. CAENET provided that the Data Ready flag is TRUE. There are two distinct cases:

a) The caller is an H. S. CAENET MASTER.

The data represent the SLAVE answer to a previous H. S. CAENET command.

b) The caller is an H. S. CAENET SLAVE.

The data represent an H. S. CAENET command.

Fig. 5.2: Interrupt Handler Flow Chart