Appendice A

Programmi di gestione dell'apparato

A.1 Programma di comunicazione su porta seriale

```
/* INCLUDE */
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <string.h>

/* DEFINE */
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define OK 137
#define TIMEOUT 147
#define ERROR 157
```

```
/* ROUTINE */
int scrittura_seriale (int fd, char buf[255], int nbyte);
int lettura_seriale (int fd, char *dato, int *nbyte);

int main()
{
/* le impostazioni di BAUDRATE sono definite in <termios.h>*/
#define BAUDRATE B4800
/* cambiare questa variabile in funzione della porta seriale usata */
#define MODEMDEVICE "/dev/ttyS2" /* COM3 */

int fd, nbyte, ndato, verifica;
int i, j;
char dato[511];
volatile int STOP=FALSE;
/* contengono le impostazioni della porta prima e durante l'utilizzo */
struct termios oldtio,newtio;

/* attiva la porta */
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
if (fd <0) { perror(MODEMDEVICE); exit(-1); }

tcgetattr(fd,&oldtio); /* salva le impostazioni correnti della porta */

bzero(&newtio, sizeof(newtio));
/* setta le nuove impostazioni di porta
 * BAUDRATE imposta bps rate
 * CRTSCTS controllo di flusso hardware (usare solo se il cavo seriale ha
 *  tutte le linee necessarie)
 * CS8 8n1 (8 bit, no parita, 1 bit di stop)
 * CLOCAL connessione locale
 * CREAD abilita la ricezione dei caratteri */
newtio.c_cflag = BAUDRATE | CSTOPB | CS8 | CLOCAL | CREAD; /* CONTROL */
```

```
/* IGNPAR ignora i byte con errori di parita
 * ICRNL associa CR con NL (altrimenti l'invio di CR non terminerebbe
 *  l'input) */
newtio.c_iflag = IGNPAR; /* INPUT */
newtio.c_oflag &= ~OPOST; /* OUTPUT */
/* senza ICANON disabilita l'input canonico e disabilita l'eco */
newtio.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); /* RAW DATA */
/* inizializza i caratteri di controllo */
newtio.c_cc[VMIN]=0; /* CONTR-CHAR */
newtio.c_cc[VTIME]=20; /* timeout 2 secondi */
/* abilita le impostazioni per la porta */
tcflush(fd,TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);
/* rende asincrono il descrittore di file
 * (the manual page says only O_APPEND and O_NONBLOCK,
 * will work with F_SETFL...) */
fcntl(fd, F_SETFL, FASYNC);

verifica = FALSE;
verifica = scrittura_seriale (fd, "a", 1);
if (verifica==OK)
printf("inviato correttamente il carattere\n");

ndato = 0; // inizializzo il
dato[ndato] = 0; // buffer di lettura
STOP=FALSE;

while(STOP==FALSE) {
verifica = FALSE;
verifica = lettura_seriale (fd, &dato[ndato], &nbyte);
if (verifica==TIMEOUT) {
printf("timeout\n");
STOP=TRUE;
```

```
}
else if (verifica==ERROR) {
printf("errore\n");
STOP=TRUE;
}
else {
printf("\ndat= %c", dato[ndato]);
ndato = ndato + nbyte;

if (dato[ndato-1]=='\n') // la stringa deve essere "#\r\n"
if (dato[ndato-2]=='\r')
if (dato[ndato-3]=='#') {
printf("lunghezza dato= %d\n",ndato);
STOP=TRUE; // controlla la fine della stringa
}
}
}

/* ripristina le vecchie impostazioni di porta */
tcsetattr(fd,TCSANOW,&oldtio);

return OK;
} /* FINE DEL MAIN */

/*************************************************************************
 * scrittura seriale invia sulla porta seriale il comando opportuno *
 *************************************************************************/
int scrittura_seriale (int fd, char buf[255], int nbyte)
{
write(fd, buf, nbyte);
printf("%s", buf);
return OK;
}
```

```
/*************************************************************************
 * lettura seriale aspetta il numero fissato di caratteri *
 *************************************************************************/
int lettura_seriale (int fd, char *dato, int *nbyte)
{
char buf[255];
int res, i;
int indice=0;

res = read (fd,buf,255);

if (res==0) {
return TIMEOUT;
}

else if (res==-1) {
return ERROR;
}

else {
buf[res]=0;

for (i=0; i<=res; i++) {
*(dato+indice) = buf[i];
indice++;
}

*nbyte=res;
printf("\nBuf= %s", buf);

return OK;
}
}
```

A.2 Programma di lettura dell'ADC

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <libdevmem.h>

#define MAP_SIZE 4096UL

#define TRUE 1

static short int read_ADC(unsigned int channel, unsigned short *valore);

int main()
{
unsigned short input;
unsigned int canale;
int ritorno;
int i;

for (i=0; i<=15; i++) { // scansione di tutti i canali
ritorno=read_ADC(i, &input);
printf( "canale %d = %d", i, input);
}
return 0;
}
```

```
static short int read_ADC(unsigned int channel, unsigned short *valore)
{
/* Lettura dei valori digitali convertiti dall'ADC dei 16 fotodiodi
*
 * base=0x30000180 e' l'indirizzo mappato in memoria della scheda
*
 * val imposta il canale che deve essere letto nel seguente modo:
*
 * ch 0 1 2 3 4 5 6 7 *
 * byte 08 80 09 90 0A A0 0B B0 *
 * ch 8 9 10 11 12 13 14 15 *
 * byte 0C C0 0D D0 0E E0 0F F0 */

#define OK 0 /* Codice di ritorno  quando non ci sono errori */
#define Time_Out 1 /* Codice di ritorno quando c'e' time-out */
#define No_Channel 2 /* Codice di ritorno quando non esiste il canale */
#define TIMEOUT 1000 /* Valore del time-out in decine di us */
#define base 0x30000180 /* indirizzo ADC */

unsigned int Canale[16];
unsigned char byl,byh;
unsigned int TO=0;

/* codici dei canali */
Canale[0]=0x08;
Canale[1]=0x80;
Canale[2]=0x09;
Canale[3]=0x90;
Canale[4]=0x0a;
Canale[5]=0xa0;
Canale[6]=0x0b;
Canale[7]=0xb0;
Canale[8]=0x0c;
```

```
Canale[9]=0xc0;
Canale[10]=0x0d;
Canale[11]=0xd0;
Canale[12]=0x0e;
Canale[13]=0xe0;
Canale[14]=0x0f;
Canale[15]=0xf0;

if (channel>15) return No_Channel;

off_t target;
libdevmem_handle h;
target = base;
h = libdevmem_open(target, MAP_SIZE);

libdevmem_write_uint8(h, 1, Canale[channel]); /* selezione del canale */
while(!(libdevmem_read_uint8(h,1)&0x01)) /* aspetto la conversione */
{
usleep(10);
if(TO++>TIMEOUT) return Time_Out;
}

byl = libdevmem_read_uint8(h, 2); /* 8 bit meno significativi */
byh = libdevmem_read_uint8(h, 3); /* 4 bit piu' significativi */
byh = byh & 0x0f;
*valore = (0x100*byh + byl);

libdevmem_close(h);

return OK;
}
```

A.3 Programma di gestione delle linee digitali di output

```
/* Con l'ausilio di tre linee di output della scheda viper si vuole pilotare
 * il generatore di corrente collegato ai due led. La sequenza dei valori da
 * associare alle linee è la seguente:
 * 1. 0-0-0 generatore spento, entrambi i led spenti
 * 2. 1-0-1 generatore acceso, corrente 20mA, led alto acceso
 * 3. 1-1-1 generatore acceso, corrente 40mA, led alto acceso
 * 4. 0-0-0 generatore spento, entrambi i led spenti
 * 5. 1-0-0 generatore acceso, corrente 20mA, led basso acceso
 * 6. 1-1-0 generatore acceso, corrente 40mA, led basso acceso
 * 7. 0-0-0 generatore spento, entrambi i led spenti
 * All'accensione della scheda viper le linee di output 0,1,2 sono settate
 * su 0, le linee 3,4,5,6,7 sono settate su 1. Utilizzo le linee 0,1,3. */

#include <stdio.h>
#include <stdlib.h>
#include <libdevmem.h>

#define MAP_SIZE 4096UL
#define acceso 0x00100000 /* bit 0 */
#define corrente 0x00200000 /* bit 1 */
#define altobasso 0x00800000 /* bit 3 */
#define set0 0x40E00024 /* set low */
#define set1 0x40E00018 /* set high */

int main() {
unsigned long writeval;
off_t target;
libdevmem_handle h;
h = libdevmem_open(target, MAP_SIZE);
```

```
/* 1. LED SPENTI 0-0-0 */
target = set0;
writeval = acceso | corrente | altobasso;
libdevmem_write_uint32(h, 0, writeval);
printf( "LED SPENTI 0-0-0\n" );

/* 2. LED ALTO ACCESO 20 mA 1-0-1 */
target = set1;
writeval = acceso | altobasso;
libdevmem_write_uint32(h, 0, writeval);
printf( "LED ALTO ACCESO 20 mA 1-0-1\n" );

/* 3. LED ALTO ACCESO 40 mA 1-1-1 */
target = set1;
writeval = acceso | corrente | altobasso;
libdevmem_write_uint32(h, 0, writeval);
printf( "LED ALTO ACCESO 40 mA 1-1-1\n" );

/* 4. LED SPENTI 0-0-0 */
target = set0;
writeval = acceso | corrente | altobasso;
libdevmem_write_uint32(h, 0, writeval);
printf( "LED SPENTI 0-0-0\n" );

/* 5. LED BASSO ACCESO 20 mA 1-0-0 */
target = set1;
writeval = acceso;
libdevmem_write_uint32(h, 0, writeval);
printf( "LED BASSO ACCESO 20 mA 1-0-0\n" );

/* 6. LED BASSO ACCESO 40 mA 1-1-0 */
target = set1;
writeval = acceso | corrente;
```

```
libdevmem_write_uint32(h, 0, writeval);
printf( "LED BASSO ACCESO 40 mA 1-1-0\n" );

/* 7. LED SPENTI 0-0-0 */
target = set0;
writeval = acceso | corrente | altobasso;
libdevmem_write_uint32(h, 0, writeval);
printf( "LED SPENTI 0-0-0\n" );

libdevmem_close(h);
return 0;
}
```