LIBRERIA CTS
CONTROL Y PROGRAMACION DE LA TARJETA CT6811 DESDE EL PC

Juan González Gómez
Microbotica, S.L

Febrero, 2000

PREFACIO

Este documento describe el interfaz de la libreria CTS, para la programación en ***LINUX*** de la tarjeta CT6811 de Microbotica. Si el lector quiere programar aplicaciones autónomas, entonces este documente no es para él, aunque le puede resolver la vida para la adquisición de datos.

La tarjeta CT6811 fue desarrollada inicialmente para aplicaciones autónomas, esto es, sistemas que no dependen del PC para su funcionamiento. El ejemplo más claro son los microbots. Una vez que se ha programado un microbot con un comportamiento determinado, se graba el programa en la memoria EEPROM y el robot es totalmente autónomo.

Sin embargo, esta tarjeta permite comunicar el PC con el mundo exterior, controlando circuitos externos o monitorizando su estado. ¿Quien no ha soñado nunca con encender y apagar las luces de su casa desde el PC? ¿Y controlar la maqueta del tren? ¿Y controlar un brazo robot desde el PC?. El problema siempre es el mismo: ¿Como “saco” la información desde el PC y la llevo a mis circuitos? ¿Como puedo por ejemplo activar o desactivar un relé?.

En este documento se presenta una arquitectura que permite realizar este tipo de proyectos. Además se facilitan las herramientas para poder realizarlos de una manera rápida y sencilla.

Este documento se ha desarrollado para versión 1.4 de la libreria CTS, que fue diseñada para trabajar con versiones 2.2.10 o superiores del Kernel de Linux.

LICENCIA

www.microbotica.es

Índice general

Índice de figuras

Capítulo 1

LA LIBRERIA CTS

1.1. ARQUITECTURA PARA EL CONTROL DESDE EL PC

En esta sección se presenta la arquitectura empleada por los ingenieros de Microbótica para controlar circuitos desde el PC. El diagrama de bloques se muestra en la figura 1.1. Nuestro programa de aplicación tiene acceso a todos los recursos del 68HC11 por medio de la libreria CTS y del servidor CTSERVER. Acceso a todos los recursos quiere decir que desde nuestro programa en C tendremos acceso a cualquiera de los puertos de E/S del 6811, memoria, SPI, conversores A/D etc... ¡¡Es como si el PC tuviese todos esos recursos dentro!!.

Esto se vera más claro con un ejemplo. En la figura 1.2 se muestra un fragmento de codigo en C que hace parpadear el led de la CT6811. Este led se encuentra conectado en el bit 6 del puerto A. En este programa se presenta una función que cambia de estado el led. Primero se lee el valor del puerto A con la instrucción LOAD, se cambia de estado el bit 6 y se envía al puerto A con la instruccion STORE. Fácil, ¿No?.

Sin embargo esta arquitectura tiene un inconveniente. Puesto que la velocidad de transmisión por el puerto serie está limitada, la velocidad con la que podemos actuar sobre los recursos del 6811 no es tan elevada como lo sería si se ejecutase el código directamente en el propio 6811. Esto tiene mucho sentido

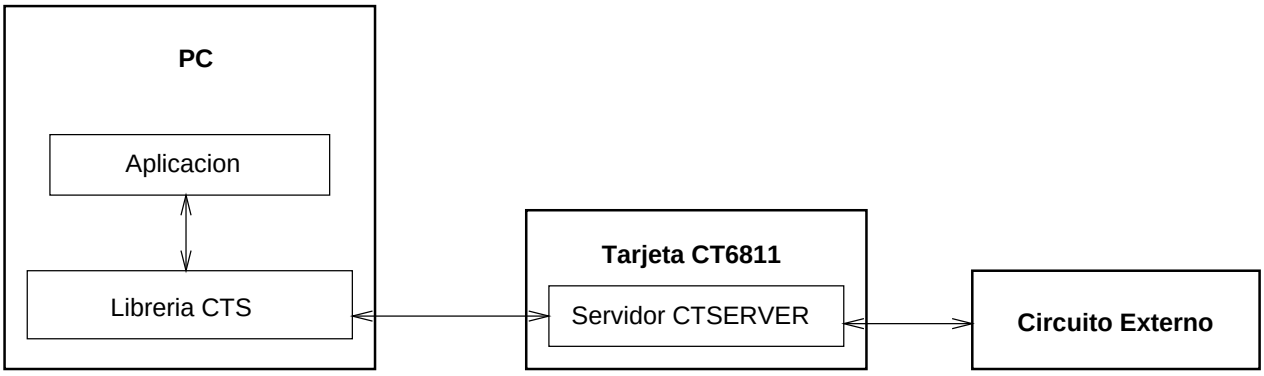

Figura 1.1: Arquitectura para el control desde el PC

```
cambiar_estado_led()
{
  byte n;

  load(&n,PORTA); /* Leer estado del puerto A */
  n^=0x40;        /* Cambiar de estado bit 6 */
  store(n,PORTA); /* Escribir en el puerto A */
}
```

Figura 1.2: Fragmento de codigo para hacer parpadear el led de la CT6811

puesto que cada vez que se ejecuta un instrucción STORE para almacenar un valor en una posición de memoria del 6811, la libreria CTS está enviando una serie de bytes al CTSERVER a través del puerto serie. El CTSERVER recibe estos bytes y realiza la operación de almacenamiento.

1.2. Un primer ejemplo

Antes de entrar en los detalles de la libreria CTS, se presenta un ejemplo de control de la CT6811 desde el PC. Se trata del típico ejemplo que hace parpadear el led de la CT6811. Más adelante se explican todas las funciones en profundidad. Aunque el lector no comprenda todos los detalles, con este ejemplo se muestra lo fácil que resulta acceder a la CT6811 desde el PC y le sirve para ir centrando las ideas.

```
/*************************************************************************/
/* LEDPARPA.C c) MICROBOTICA, S.L. ENERO 2000                            */
/*************************************************************************/
/*                                                                       */
/* Ejemplo de prueba de la libreria CTS. Se hace parpadear el LED de la  */
/* Tarjeta CT6811.                                                       */
/*************************************************************************/

#include "r6811pc.h"
#include "serie.h"
#include "bootstrp.h"
#include "stdio.h"

int i=0;
int n=0;

void accion_load()
/*****************************************/
/* Accion al realiar al cargar el servidor  */
/*****************************************/
{
  if (n==16 || i==255) {
    n=0;
    printf ("*");
```

```
    fflush(stdout);
  }
  i++;
  n++;
}

int cargar_ctserver()
/*****************************************************************/
/* Cargar el CTSERVER. El CTSERVER que se carga esta configurado  */
/* para la velocidad de 7680 baudios.                             */
/*****************************************************************/
{

byte cts9600[256]={
0x20,0x05,0x00,0x00,0x00,0x00,0x00,0xCE,0x10,0x00,0x1F,0x2E,0x40,0xFC,0x86,
0x30,0xA7,0x2B,0xCE,0x10,0x00,0x8D,0x66,0x97,0x06,0x81,0x44,0x27,0x13,0x81,
0x43,0x27,0x54,0x81,0x41,0x27,0x17,0x81,0x42,0x27,0x0E,0x81,0x45,0x27,0x6A,
0x7E,0x00,0x12,0x86,0x4A,0x8D,0x50,0x7E,0x00,0x12,0x7C,0x00,0x06,0x20,0x03,
0x7F,0x00,0x06,0x8D,0x4A,0x18,0xDF,0x02,0x8D,0x45,0x18,0xDF,0x04,0x18,0x8C,
0x00,0x00,0x27,0x23,0x18,0xDE,0x02,0x96,0x06,0x4D,0x26,0x07,0x18,0xA6,0x00,
0x8D,0x28,0x20,0x05,0x8D,0x1D,0x18,0xA7,0x00,0x18,0x08,0x18,0xDF,0x02,0x18,
0xDE,0x04,0x18,0x09,0x18,0xDF,0x04,0x20,0xD7,0x7E,0x00,0x12,0x8D,0x14,0x18,
0xDF,0x02,0x18,0x6E,0x00,0x1F,0x2E,0x20,0xFC,0xA6,0x2F,0x39,0x1F,0x2E,0x80,
0xFC,0xA7,0x2F,0x39,0x36,0x37,0x8D,0xEE,0x16,0x8D,0xEB,0x18,0x8F,0x33,0x32,
0x39,0x8D,0xF2,0x18,0xDF,0x02,0x8D,0xED,0x18,0xDF,0x04,0x18,0x8C,0x00,0x00,
0x27,0x34,0x18,0xDE,0x02,0xC6,0x16,0xF7,0x10,0x3B,0x18,0xE7,0x00,0xC6,0x17,
0xF7,0x10,0x3B,0x8D,0x28,0xC6,0x02,0xF7,0x10,0x3B,0x8D,0xBD,0x18,0xA7,0x00,
0xC6,0x03,0xF7,0x10,0x3B,0x8D,0x17,0x8D,0xB8,0x18,0x08,0x18,0xDF,0x02,0x18,
0xDE,0x04,0x18,0x09,0x18,0xDF,0x04,0x20,0xC6,0x7F,0x10,0x3B,0x7E,0x00,0x12,
0x18,0x3C,0x18,0xCE,0x0D,0x10,0x18,0x09,0x18,0x8C,0x00,0x00,0x26,0xF8,0x18,
0x38,0x39,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,};

  printf ("\nCargando CTSERVER:\n\n");
  set_break_timeout(500000);
  printf("0%%     50%%   100%%\n");
  printf("................\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");
  fflush(stdout);

  if (cargar_ramint(cts9600,accion_load)==0) {
    printf ("\nError:%s\n",getloaderror());
    return 0;
  };
  printf ("\nOK!\n");
  return 1;
}

main()
{
  int n;
```

```
  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
    exit(1);
  }
  if (cargar_ctserver()==0) {
    cerrar_puerto_serie();
    exit(1);
  }
  baudios(9600);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_puerto_serie();
    exit(1);
  }

  for (i=0; i<5; i++) {
    store(0x40,PORTA);    /* Encender led */
    usleep(200000);
    store(0x00,PORTA);    /* Apagar led   */
    usleep(200000);
  }
  cerrar_puerto_serie();
  printf ("\n\n");
}
```

El programa a simple vista puede resultar grande y complicado. Nada en absoluto. Fijémonos en el programa principal. Lo primero que se hace es llamar a la función *abrir_puerto_serie*, para inicializar las comunicaciones serie. El puerto seleccionado donde se encuentra la tarjeta CT6811 es el COM2 (/dev/ttyS1). Si existe algún error al acceder al puerto serie se muesta.

A continuación se llama a la función *cargar_ctserver()*, que está implementada en el propio programa. Esta función simplemente carga en la RAM interna del 6811 el CTSERVER para que nuestro programa de aplicación se pueda comunicar con él. Esta función se tendrá que utilizar siempre, salvo que el servidor se encuentre grabado en la memoria EEPROM, lo que suele ser normal cuando el prototipo que se está diseñando se termina.

El CTSERVER que se ha cargado en la RAM es una versión que funciona a 9600 baudios, y por ello se configura el PC para trabajar a esta velocidad con la función *baudios*. Para cargar el ctserver en la RAM del 6811 el PC estaba configurado para funcionar a una velocidad de 7680 baudios (Velocidad del puerto serie del 6811 al arrancar en modo bootstrap).

Una vez cargado el ctserver y configurada la velocidad, hay que comprobar que efectivamente hay comunicación con el servidor. Esto se consigue con la función *check_conexion()*. Al ejecutarse esta función se envía un mensaje al ctserver y se espera su respuesta. Si la respuesta no llega en un plazo de tiempo será debido a que el ctserver no se está ejecutando en la CT6811 y la comunicación es imposible. En este caso se devuelve un mensaje de error.

Finalmente llega el verdadero programa, el que hace parpadear el led. Hasta ahora todo ha sido configuración del sistema. Con la función store() se envía el valor 0x40 al puerto A para encender el

led, se espera 200 milisegundos, se envía el valor 0 para apagar el led, se espera otros 200 milisegundos y se repite el bucle 5 veces.

Cuando el bucle finaliza se llama a la función *cerrar_puerto_serie* para dejar de trabajar con el puerto serie y el programa termina.

1.3. LOS MODULOS DE LA LIBRERIA CTS

La principal misión de la libreria CTS es la de comunicarse con el programa CTSERVER y proporcionar las funciones load y store. Sin embargo dispone de muchas otras funciones que son muy útiles para desarrollar programas para la CT6811. Los programas mcboot, downmcu, ctdialog, ct294,... y en general todas las cttools han sido programadas utilizando llamadas a la librería CTS.

La libreria CTS se divide en cuatro módulos:

1. **Módulo serie**. Agrupa todas las funciones necesarias para la comunicación con cualquier dispositivo conectado al puerto serie.
2. **Módulo S19**. Funciones para el manejo de ficheros .S19
3. **Módulo Bootstrap**. Gestión del modo bootstrap de la CT6811 (Reset, salta a la EEPROM, carga de programas en RAM...)
4. **Módulo Ctserver**. Funciones de alto nivel para acceder a los recursos de la CT6811

1.4. LAS FUNCIONES DE INTERFAZ

A continuacion se listan todas las funciones que pueden ser utilizadas por las aplicaciones basadas en la libreria CTS.

1.4.1. Modulo serie

- *int abrir_puerto_serie(int puerto);* Comenzar a trabajar con el puerto serie
- *void cerrar_puerto_serie()*; Dejar de trabajar con el puerto serie
- *int baudios(int vel);* Establecer la velocidad de comunicaciones
- *int vaciar_buffer_tx();* Vaciar el buffer de transmisión
- *int vaciar_buffer_rx()*; Vaciar el buffer de recepción
- *int enviar_bloque(char *cad,int tam);* Enviar un bloque de datos
- *int enviar_car(char car);* Enviar un caracter
- *int enviar_break();* Enviar señal de Break

- *int car_waiting()*; Devolver el numero de caracteres en el buffer de recepción, pendientes de ser leidos.
- *int wait_break(int plazo);* Esperar a que se reciba una señal de BREAK en un plazo de tiempo establecido.
- *int bufftx_waiting();* Devolver el numero de caracteres en el buffer de transmisión, pendientes de ser enviados.
- *char leer_car();* Leer un caracter del buffer de recepción
- *char leer_car_plazo(int plazo, int *timeout);* Leer un caracter del buffer de recepción con plazo de tiempo.
- i*nt dtr_on();* Poner la señal del DTR a on
- *int dtr_off();* Poner la señal del DTR a off
- *int getserial_fd();* Obtener el descriptor del puerto serie, para acceder directamente.
- *char *getserial_error();* Devolver la cadena asociada al ultimo error producido

1.4.2. Modulo S19

- *int abrir_s19(char *fich,S19 *ss19,int modo);* Abrir un archivo S19 para trabajar con el
- *void cerrar_s19(S19 ss19);* Cerrar archivo S19
- *int leerdir_s19(S19 ss19,int nreg, unsigned int *dir);* Leer el campo direccion del registro especificado
- *int leercod_s19(S19 ss19,int nreg, byte *cod, byte *tam, byte *crc);* Leer campo código del registro indicado
- *byte calcular_crc(byte *bloque, int tam);* Calcular CRC de un bloque de datos
- *unsigned int getnbytes19(S19 ss19);* Devolver el númeo de bytes del fichero S19
- *unsigned int getnregs19(S19 ss19);* Devolver el número de registros
- *char *geterrors19();* Obtener la cadena de error producida en la última apertura del fichero
- *void s19toramint(S19 f,byte *ramint, byte *ramintoc);* Obtener la matriz de código comprendida entre las direcciones 0-255. El resto se desprecia.
- *void s19toeeprom(S19 f,byte *eeprom, byte *eepromoc);* Obtener la matriz de código comprendida entre las direcciones B600-B7FF

- *int situacion_progs19(S19 fs19, int *ov);* Comprobar la situación de un programa: RAM interna, EEPROM o RAM externa
- *void raminttoc(byte *ramint, char *cadc);* Generar una matriz en C con el codigo de un programa para la RAM interna.

1.4.3. Modulo bootstrp

- *void set_break_timeout(unsigned long timeout);* Cambiar el valor del timeout para recibir la señal de break
- *void set_eco_checking(byte eco);* Cambiar el estado de la comprobación del eco
- *void resetct6811();* Reset de la CT6811
- i*nt okreset();* Realizar un reset y comprobar que se recibe la señal de BREAK
- *int jump_eeprom();* Realizar un reset y saltar a la EEPROM
- *int jump_ram();* Realizar un reset y saltar a la RAM
- *int cargar_ramint(byte *ramint, void (*car)());* Cargar una matriz con codigo en la RAM interna
- *int cargars19_ramint(S19 fs19, void (*car)());* Cargar fichero S19 en la RAM interna
- *char *getloaderror();* Devolver el error producido en la última carga de programas

1.4.4. Modulo ctserver

- int load(byte *valor, uint dir); Leer un dato de una posición de memoria
- int load_block(byte *buff, uint tam, uint dir, void (*reccar)()); Leer un bloque de datos a partir de una dirección de memoria
- void store(byte valor, uint dir); Almacenar un dato en una posición de memoria
- void store_block(byte *buff, uint tam, uint dir, void (*sendcar)()); Almacenar un bloque de datos
- int store_block_eeprom(byte *buff, uint tam, uint dir, void (*eeprcar)()); Grabar un bloque de datos en la memoria eeprom
- int check_conexion(); Comprobar si hay conexión con el servidor
- void execute(uint dir); Ejecutar el código que se encuentra a partir de la dirección indicada
- int hay_conexion(); Devolver el estado de la última conexión

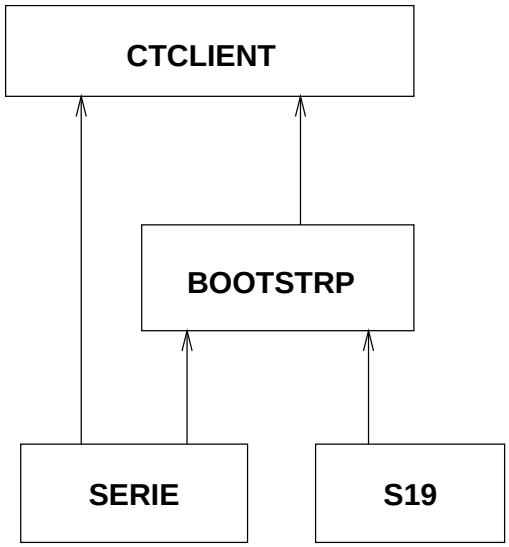

Figura 1.3: Dependencia entre los módulos de la libreria CTS

1.5. DEPENDENCIAS ENTRE LOS DIFERENTES MODULOS

En la figura 1.3 se muestra la dependencia entre los 4 módulos que componen la librería CTS. El módulo Bootstrap utiliza funciones de S19 y SERIE. CTCLIENT llama a funciones de SERIE y BOOTRAP. Un programa de aplicación típico utilizará CTCLIENT y BOOTSTRAP.

Capítulo 2

MODULO SERIE

2.1. INTRODUCCION

El módulo serie consta de una serie de funciones que permiten manejar un puerto serie. Las funciones de interfaz son iguales que las empleadas en la versión de MS-DOS, con el objetivo de buscar portabilidad pero limitando la funcionalidad que se puede conseguir con Linux.

Este módulo es útil para controlar cualquier periférico conectado al puerto serie, como un módem o la tarjeta CT6811, haciendo que el programador se olvide de los detalles concretos de manejo los puertos serie.

En este capítulo se explica cómo manejar el módulo serie, pero no se explica cómo está implementando, dejándose para el lector el estudio de las fuentes.

2.2. UTILIZACION

2.2.1. Apertura y cierre del puerto serie

Antes de trabajar con cualquiera de las funciones del módulo serie es necesario llamar a la función de apertura del puerto serie. Hay que tener en cuenta que también hay que llamar a esta función cuando se emplea un módulo que realiza llamadas al módulo serie (Ver apartado 1.5).

Existen dos constantes definidas en serie.h: COM1 y COM2 que nos indican qué puerto serie es con el que se quiere trabajar. En LINUX estos puertos se corresponden con /dev/ttyS0 y /dev/ttyS1 respectivamente.

De igual manera que se ha abierto el puerto serie, al terminar nuestra aplicación lo debemos cerrar. En la figura 2.1 se muestra el esquema típico de utilización de las funciones para abrir y cerrar el puerto. La función *getserial_error()* devuelve la cadena de error en caso de que éste se produzca.

Al abrir el puerto serie se activa la señal DTR que hace que la CT6811 se quede en estado de RESET. Hasta que no se desactive esta señal la CT6811 no funcionará.

```
if (abrir_puerto_serie(COM2)==0) {
  printf (“Error al abrir puerto serie:%s\n”,getserial_error());
}

/* Aqui vendría el código de la aplicación */

cerrar_puerto_serie();
```

Figura 2.1: Apertura y cierre del puerto serie

```
...
baudios(7680);
dtr_off();
..
..
dtr_on();
baudios(1200);
..
```

Figura 2.2: Cambio de velocidad y modificación del DTR

2.2.2. Configuración de la velocidad y de la señal DTR

Una vez abierto el puerto serie, lo segundo será establecer la velocidad con la que se quiere trabajar. Esto se consigue con la función baudios(). Esta función sólo permite establecer tres velocidades: 1200, 7680 y 9600 Baudios. Sólo se han implementando estas velocidades porque son las tres únicas a las que se puede comunicar la CT6811 con el PC.

Para manejar la señal DTR existen las funciones dtr_on() y dtr_off() que activan y desactivan el DTR respectivamente. La señal DTR se utiliza para realizar un reset de la CT6811.

En la figura 2.2 se muestra un ejemplo de utilización de estas funciones.

2.2.3. Envío de datos

Para enviar datos existen dos funciones: *enviar_car()* para enviar un carácter y *enviar_bloque ()* para enviar un bloque de datos. Todos los datos pendientes de ser enviados se almacenan en un buffer del Kernel, y se van enviando por el puerto serie. La función *bufftx_waiting()* permite conocer el número de datos almacenados en este buffer. Estos son datos que están pendientes de ser enviados pero que todavía no lo han hecho. Es posible vaciar este buffer llamando a la función *vaciar_buffer_tx()*.

En la figura 2.3 se muestra un ejemplo en el que se envía la cadena ATZ a un modem.

La función enviar_break() permite enviar una señal de BREAK. Esta función no se emplea con la CT6811, pero se ha incluido para otro tipo de periféricos.

```
char cad[]={“ATZ”};

enviar_bloque(cad,strlen(cad));
enviar_car(’\r’);
```

Figura 2.3: Ejemplo de envío de datos

2.2.4. Recepción de datos

Los datos que llegan por el puerto serie son recibidos por el kernel y situados en un buffer de entrada donde se quedan almacenados hasta que la aplicación los pueda leer. La aplicación tiene 3 opciones a la hora de leer los datos:

1. Quedarse esperando a que llegue algún carácter. Durante esta espera la aplicación está bloqueada y no puede realizar ninguna otra operación.

2. Quedarse esperando a que llegue algún carácter durante un tiempo determinado. Si trancurrida la espera no se ha recibido ningún carácter la aplicación retoma el control.

3. Realizar otra operación mientras se está esperando a recibir datos. Cada cierto tiempo la aplicación comprueba si ha llegado algún caracter al buffer de recepción. Si no es así continúa haciendo otras operaciones.

La primera forma de leer datos se consigue con la función *leer_car()*. Esta función devuelve el siguiente carácter del buffer de recepción, y si no hay ninguno se queda esperando a que llegue. No es muy habitual llamar directamente a esta función porque la mayoría de las veces no interesa que nuestra aplicación se quede bloqueada.

Para la segunda forma de leer datos se llama a la función *leer_car_plazo()*. Si algún carácter en el buffer de recepción se devuelve. Si no lo hay, se queda esperando un cierto tiempo, definidio en microsegundos, hasta que llegue el carácter. Si transcurre el plazo y no se recibe nada se informa a la aplicación de que se ha producido un TIMEOUT. Esta es la función que normalmente se utiliza, pues permite detectar que se ha perdido la conexión con el periférico serie, bien porque el cable de conexión se ha soltado, bien porque el periférico no está alimentado.

La tercera forma, que suele ser bastante común, se emplea llamando a la función *car_waiting()*, que devuelve el número de caracteres en el buffer de recepción. Si esta función devuelve el valor 0, quiere decir que no hay caracteres esperando a ser leidos, por lo que la aplicación puede realizar otra operación.

En la figura 2.4 se presenta un ejemplo en el que se imprime todo lo que se recibe por el puerto serie, hasta recibir el valor 27, correspondiente a ESC.

Existen dos funciones más relacionadas con la recepción. Vaciar_buffer_rx() elimina todos los caracteres que se encuentran en el buffer de recepción. La función wait_break() espera a recibir una señal de BREAK en el plazo de tiempo especificado.

```
do {
  if (car_waiting()) {
    c=leer_car();
    printf (“%c”,c);
  }
} while (c!=27);
```

Figura 2.4: Ejemplo de recepción de caracteres

2.2.5. Otras operaciones con el puerto serie

La función *getserial_fd()* devuelve el descriptor del fichero asociado al puerto serie. Este descriptor es necesario para realizar operaciones directamente sobre el puerto serie, como por ejemplo cambiar la velocidad a un valor diferente de los soportados por la función *baudios()*, o actuar sobre otras señales diferentes al DTR. También se puede utilizar para llamar directamente a la función *read()* y leer datos sin utilizar *leer_car()*.

2.3. EJEMPLOS

A continuación se muestra un ejemplo de un terminal de comunicaciones que envía por el puerto serie todo lo que se escribe y presenta en pantalla todo lo que se recibe. El terminal permite cambiar la velocidad de transmisión entre los valores 1200, 7680 y 9600, así como actuar sobre el DTR.

```
/*****************************************************************/
/* TERMINAL.C (c) Microbotica, S.L. Enero 2000 */
/*---------------------------------------------------------------*/
/* Ejemplo del modulo serie.c. */
/* Pequeño terminal de comunicaciones. */
/*****************************************************************/

#include "serie.h"

#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <sys/ioctl.h>

#define ESC 27

main()
{
 int n;
 char cad[50];
```

```
  char c,d;
  int quit=0;

  if (abrir_puerto_serie(COM2)==0) {
  printf ("Error al abrir puerto serie:%s\n",getserial_error());
  exit(1);
  }
  dtr_off();

  printf ("\nTerminal de comunicaciones. Pulse ESC para terminar\n");
  printf ("Teclas: 1.- DTR ON\n");
  printf (" 2.- DTR OFF\n");
  printf (" 3.- Enviar una cadena\n");
  printf (" 4.- Velocidad 1200\n");
  printf (" 5.- Velocidad 7680\n");
  printf (" 6.- Velocidad 9600\n");
  printf ("\n\n");

  abrir_consola();

  do {
  if (kbhit()) {
  c=getch();
  switch(c) {
  case '1' : dtr_on();
  printf ("DTR ON\n");
  break;
  case '2' : dtr_off();
  printf ("DTR OFF\n");
 break;
  case '3' : strcpy(cad,"Hola..");
  enviar_bloque(cad,strlen(cad));
  break;
  case '4' : baudios(1200); break;
  case '5' : baudios(7680); break;
  case '6' : baudios(9600); break;
  case '\n': c='\r';
  default : enviar_car(c);
  }
  }
  if (car_waiting()) {
  d=leer_car();
```

```
 if (d!=0) printcar(d);
 else printcar('*');
 }
 } while (c!=ESC);

 printf ("\n\n");
 cerrar_consola();
 cerrar_puerto_serie();
}
```

2.4. INTERFAZ

Salvo que se especifique lo contrario, las funciones devuelven el valor 1 si no se ha producido ningún error. Devuelve 0 en caso de error. En ese caso se puede obtener la cadena de error llamando a la función **getserial_error().*

- ***int abrir_puerto_serie(int puerto);*** Comenzar a trabajar con el puerto serie. El parámetro puerto indica el puerto serie a utilizar: 0 para COM1 (/dev/ttyS0) y 1 para COM2 (/dev/ttyS1).
- ***void cerrar_puerto_serie();*** Dejar de trabajar con el puerto serie
- ***int baudios(int vel);*** Establecer la velocidad de comunicaciones. Las velocidades permitidas son 1200, 7680 y 9600, que son los únicos valores que se permiten en el parámetro vel. Si se especifica un valor de velocidad erróneo se toma por defecto 9600.
- ***int vaciar_buffer_tx();*** Vaciar el buffer de transmisión
- ***int vaciar_buffer_rx()***; Vaciar el buffer de recepción
- ***int enviar_bloque(char *cad,int tam);*** Enviar un bloque de datos por el puerto serie. El parámetro *tam* indica el número de bytes que componen el bloque a enviar.
- ***int enviar_car(char car);*** Enviar un carácter por el puerto serie.
- ***int enviar_break();*** Enviar señal de Break
- ***int car_waiting()***; Devolver el número de caracteres en el buffer de recepción, pendientes de ser leidos.
- ***int wait_break(int plazo);*** Esperar a que se reciba una señal de BREAK en un plazo de tiempo establecido. El plazo de tiempo está especificado en microsegundos.
- ***int bufftx_waiting();*** Devolver el número de caracteres en el buffer de transmisión, pendientes de ser enviados.

- ***char leer_car();*** Leer un carácter del buffer de recepción
- ***char leer_car_plazo(int plazo, int *timeout);*** Leer un carácter del buffer de recepción con plazo de tiempo. El parámetro plazo especifica el tiempo a esperar en microsegundos. La función devuelve un 1 en timeout si se ha sobrepasado el plazo de tiempo sin recibir nada.
- **i*nt dtr_on();*** Poner la señal del DTR a on
- ***int dtr_off();*** Poner la señal del DTR a off
- ***int getserial_fd();*** Obtener el descriptor del puerto serie, para acceder directamente.
- ***char *getserial_error();*** Devolver la cadena asociada al ultimo error producido

2.5. LIMITACIONES

Esta librería está limitada a la utilización de un único puerto serie, no pudiendose gestionar más de un puerto serie en cada programa. La filosofía de programación es del tipo MS-DOS y no orientada a procesos como cabría esperar de un sistema UNIX.

Capítulo 3

MODULO S19

3.1. INTRODUCCION

El módulo S19 permite trabajar con ficheros con el formato .S19 de Motorola. Este formato codifica los programas ejecutables en líneas de caracteres ascii (Registros), de tal forma que los ficheros .S19 son archivos ASCII que se pueden editar. Cada línea (Registro) comienza siempre por el carácter S y está formado por caracteres ASCII pertenecientes al conjunto {'0'-'9','A'-'F'}. Estos registros contienen información sobre el código máquina de nuestro programa así como de las direcciones en las que se tiene que cargar dicho código. No es el objetivo de este documento explicar en detalle el formato .S19. Si se quieren más detalles se recomienda leer el documento 'Control de la CT6811 desde el PC', que trata sobre la librería CTSERVER para MS-DOS. Este documento se puede encontrar en el web de Microbótica, S.L.

Con este módulo seremos capaces de leer la información que está contenida en los archivos .S19 sin necesidad de compreder el formato.

3.2. UTILIZACION

3.2.1. Apertura y cierre de ficheros .S19

Los ficheros S19 se tratan en este módulo como objetos del tipo S19, sobre los que podemos realizar operaciones. Antes de trabajar con un archivo S19 hay que crear una variable de este tipo e inicializarla con el fichero a trabajar. Esto es lo que realiza la función *abrir_s19()*. Se extrae del archivo toda la información necesaria y se guarda en la variable del tipo s19. Después de trabajar con este fichero habrá que llamar a la función *cerrar_s19()* para liberar toda la memoria tomada.

En la figura 3.1 se muestra un ejemplo de apertura y cierre del fichero "ledp.s19". Si ha habido algún error al leer el archivo .S19, como por ejemplo que NO sea un fichero en este formato o que sea un fichero .S19 corrupto, la función devuelve el valor 0. En este caso la función ***geterrors19()*** devuelve la cadena de error asociada.

La función abrir_s19() tiene tres parámetros. El primero especifica el nombre del fichero .S19. El segundo la variable sobre la que guardar la información y el tercero el modo de apertura. Un 1 abre

```
S19 mi_s19;

if (abrir_s19("ledp.s19",&mi_s19,1)==0) {
  printf ("Error:%s\n",geterrors19());
  return 0;
}

 /* Realizar operaciones sobre el fichero */

cerrar_s19(mi_s19);
```

Figura 3.1: Ejemplo de apertura y cierre de ficheros .S19

```
printf ("Bytes:%u\n",getnbytes19(mi_s19));
printf ("Registros:%u\n",getnregs19(mi_s19));
```

Figura 3.2: Lectura del número de bytes de y registros de un fichero .S19

el fichero y comprueba el CRC, por lo que se pueden detectar ficheros .S19 corruptos. Un 0 abre sin comprobar el CRC. Normalmente siempre se abrirá con un 1 para mayor seguridad.

3.2.2. Numero de registros y numero de bytes de un fichero .S19

Una vez abierto un fichero .S19, con la función getnbytes() podemos conocer el número de bytes que va a ocupar el programa en la memoria del micro, información muy útil para calcular desbordamientos. (¡¡Un programa mayor de 256 bytes no cabe en la memoria interna de lo micros A1 y E2!!).

La función getnregs() devuelve el número de registros. Esta información no es muy útil de cara al usuario pero sí lo es para el programador. El registro es una unidad que contiene código, una dirección de memoria en donde colocarlo y el tamaño del código. Todo el código de un registro se encuentra en posiciones de memoria contiguas. Diferentes registros pueden tener códigos en diferentes partes de memoria. Moviéndonos por todos los registros podremos conocer el mapa de memoria utilizado por el fichero .S19. En la figura 3.2 se presenta un fragmento de código que imprime el número de bytes y de registros de un fichero S19.

3.2.3. Lectura de registros

Para acceder a la información contenida en los registros se utilizan las funciones *leerdir_s19()* y *leercod_s19()*. La primera devuelve la dirección del registro especificado. La segunda el código, el tamaño y el byte de CRC. Ambas funciones devuelve 1 si el registro solicitado existe, 0 en caso contrario. En la figura 3.3 se muestra un fragmento de código que accede a todos los registros y obtiene la información. Este código podría formar parte de un programa cargador para la RAM externa, como el programa ctload.

```
int reg;
int total_regs;
int unsigned int dir;
byte codigo[40];
byte crc;
byte tam;

total_regs=getnregs19(mi_s19);   /* Obtener numero de registros */
for (reg=1;reg<=total_regs; reg++) {
  leerdir_s19(mi_s19,reg,&dir);            /* Leer direccion */
  leercod_s19(mi_s19,reg,codigo,&tam,&crc);/* Leer codigo    */

  /* Realizar las operaciones necesarias*/
}
```

Figura 3.3: Lectura de registros

3.2.4. Otras operaciones con ficheros S19

Con las funciones anteriores es posible implementar cualquier tipo de programa que maneje ficheros .S19. No obstante, en el módulo s19 se presentan algunas funciones que apoyándose en las anteriores ofrecen un servicio de más alto nivel, como por ejemplo saber si un fichero S19 se ha creado para la RAM interna, la EEPROM o la RAM externa.

La función *situacion_progs19()* comprueba si el fichero S19 se ha creado para la RAMinterna, EEPROM o RAM externa. Los valores que devuelve son:

- 1 si es un programa para la ram interna, es decir, que todas sus instrucciones se encuentran dentro de las posiciones de memoria 0-255. Se devuelve ov=1 si es para la ram interna pero la desborda.
- 2 si es un programa para la eeprom interna. Se comprueba que todas las posiciones de memoria se encuentren entre las direccion 0xB600-0xB7FF. ov=1 indica que se ha desbordado la memoria eeprom
- 3 Indica que es un programa para la RAM externa.

En la figura 3.4 se muestra un fragmento de programa que utiliza la función situacion_progs19() para imprimir en pantalla información sobre un fichero .S19

La función s19toramint() introduce el código de un fichero S19 en una matriz de tamaño 256 bytes. Sólo se introducen los 256 bytes de la ram interna. La función también genera una matriz de ocupación para indicar qué posiciones dentro de los 256 bytes de la ram interna tienen código y cuáles están vacías. Esta matriz de ocupación como elementos 0’s y 1’s. Un 0 indica que la posición está libre y un 1 que está ocupada. Esto nos permite por ejemplo presentar en la pantalla un “mapa” de la memoria interna que utiliza un programa. En la figura 3.5 se muestra un ejemplo que imprime en pantalla un ’*’ si la posicion no está ocupada o el valor del byte si está ocupada.

La función s19toeeprom() funciona de la misma manera pero con la memoria eeprom. Las matrices de datos y ocupación son de 512 bytes.

```
S19 mi_s19;
int ov;

 /* Abrir fichero .S19 y otras operaciones */

leerdir_s19(mi_s19,1,&dir);   /* Leer direccion del primer registro */

printf ("Situacion programa   : ");

switch(situacion_progs19(mi_s19,&ov)) {
   case 1 : printf ("RAM INTERNA");
            if (ov) printf (" (Desbordamiento)");
            break;
   case 2 : printf ("EEPROM INTERNA");
            if (ov) printf ("(Desbordamiento)");
            break;
    case 3 : printf ("RAM EXTERNA");
            break;
}
```

Figura 3.4: Ejemplo de la función situacion_progs19.

int reg;

```
byte ramint[256];
byte ramintoc[256];
S19 mi_s19;
int i;

 /* Apertura de S19 y otras operaciones */

s19toramint(mi_s19,ramint,ramintoc);
for (i=0; i<256; i++) {
  if (ramintoc[i]) printf (“%X “,ramint[i]);
  else printf (“*”);
}
```

Figura 3.5: Ejemplo de utilizacion de la funcion s19toramint

```
S19 mi_s19;
char ramint[256];
char ramintoc[256];
char cadc[1350];

/* Abrir fichero S19 y realizar otras operaciones */

s19toramint(mi_s19,ramint,ramintoc);
raminttoc(ramint,cadc);
printf ("%s\n",cadc);
```

Figura 3.6: Ejemplo de utilización de la función raminttoc.

Muchas veces es necesario disponer de una matriz para la ram interna en código ascii, de forma que se puedan tener programas s19 en ascii para incluir en los programas fuentes en C. Esto se consigue con la función raminttoc(). En la figura 3.6 se muestra un fragmento de un ejemplo de utilización.

3.3. EJEMPLOS

3.3.1. Programa leers19

Este programa de ejemplo lee un fichero S19 e imprime información sobre el fichero.

```
/*
 ***************************************************************************
 * LEERS19     (c) Microbotica,S.L. Enero 1998.                            *
 ***************************************************************************
 *                                                                         *
 * Ejemplo de utilizacion de la libreria S19.C.                            *
 *                                                                         *
 *   Este programa lee un archivo en formato .S19 y devuelve el numero de  *
 * bytes que ocupa el codigo, el numero de registros del tipo 1 y la direc- *
 * cion de comienzo del codigo/datos.                                      *
 *                                                                         *
 ***************************************************************************
*/
#include "s19.h"

main()
{
  S19 mi_s19;               /* Fichero S19        */
  char cad[80];             /* Nombre del fichero */
  char *caderror;           /* Cadena de error    */
  unsigned int dir;         /* Direccion comienzo */
  int ov;


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

```
  /* ---- ABRIR EL FICHERO S19 ----- */
  /* -------------------------------*/
  printf ("\nNombre fichero: ");
  gets(cad);
  if (abrir_s19(cad,&mi_s19,1)==0) {   /* Si se ha producido un error */
    caderror=(char *)geterrors19();    /* Leer la cadena de error     */
    printf ("Error:%s\n",caderror);   /* Imprimir mensaje de error   */
    return 0;
  }
  printf ("\n");

  /* ----------------------------------------- */
  /* ---- MOSTRAR LA INFORMACION DEL FICHERO ---- */
  /* ----------------------------------------- */
  printf ("Numero de registros  :%u\n",getnregs19(mi_s19));
  printf ("Tamano en Bytes      :%u\n",getnbytes19(mi_s19));

  leerdir_s19(mi_s19,1,&dir);   /* Leer direccion del primer registro */
  printf ("Direccion de comienzo:%X\n",dir);

  printf ("Situacion programa   : ");

  switch(situacion_progs19(mi_s19,&ov)) {
    case 1 : printf ("RAM INTERNA");
             if (ov) printf (" (Desbordamiento)");
             break;
    case 2 : printf ("EEPROM INTERNA");
             if (ov) printf ("(Desbordamiento)");
             break;
    case 3 : printf ("RAM EXTERNA");
             break;
  }

  printf ("\n\n");

  /* ------------------------------ */
  /* ---- CERRAR EL FICHERO S19 ---- */
  /* ------------------------------ */
  cerrar_s19(mi_s19);
  return 0;
}
```

3.3.2. Programa dumps19

Este programa lee un archivo S19 e imprime el valor de todos los campos de los registros leídos.

```
/*
 ***************************************************************************
 * DUMPS19     (c) Microbotica, S.L. Febrero 1998.                         *
 ***************************************************************************
 *                                                                         *
```

```
 * Ejemplo de utilizacion de la libreria S19.C.                              *
 *                                                                           *
 *   Este programa lee un archivo en formato .S19 y vuelca los campos de     *
 * codigo/datos y de direccion.                                              *
 *                                                                           *
 *****************************************************************************
*/

#include "stdio.h"

/* -- Librerias del Grupo J&J -- */
#include "ascbin.h"
#include "s19.h"

main()
{
  char *caderror;
  char cad2[5];
  char cad[80];
  byte codigo[37];
  byte crc;
  byte tam;
  int i,n;
  unsigned int dir;
  unsigned int nreg;
  S19 mis19;      /* Fichero S19 */

  /* ------------------------------- */
  /* ---- ABRIR EL FICHERO S19 ----- */
  /* -------------------------------*/
  printf ("\nNombre fichero: ");
  gets(cad);
  if (abrir_s19(cad,&mis19,1)==0) {
    caderror=(char *)geterrors19();
    printf ("Error:%s\n",caderror);
    return 0;
  }

  /* --------------------------- */
  /* ---- VOLCAR EL FICHERO  ---- */
  /* --------------------------- */
  nreg=getnregs19(mis19);
  printf ("\n");
  printf ("Reg. Dir. tam.                 Codigo/datos\n");

  for (i=1; i<=nreg; i++) {
    leerdir_s19(mis19,i,&dir);
    leercod_s19(mis19,i,codigo,&tam,&crc);
    printf ("%3u%4X %4u ",i,dir,tam);
    for (n=0; n<tam; n++) {
      bytetochar2(codigo[n],cad2);
```

```
      printf ("%s",cad2);
    }
    printf ("\n");

  }

  /* ------------------------------ */
  /* ---- CERRAR EL FICHERO S19 ---- */
  /* ------------------------------ */

  cerrar_s19(mis19);
  return 0;
}
```

3.3.3. Programa views19c

Programa para imprimir una matriz en C, que contiene los datos de un fichero .S19.

```
/*
 ***************************************************************************
 * VIEWS19C.C  (c) Microbotica, S.L Febrero 1998.                          *
 ***************************************************************************
 *                                                                         *
 * Programa para convertir archivos .S19 para la RAM INTERNA en su         *
 * representacion como una matriz de 256 bytes en C.                       *
 *                                                                         *
 ***************************************************************************
*/

#include "stdio.h"

/* --- Librerias del Grupo J&J ---- */

#include "s19.h"

main()
{
  char cad[80];
  char *caderror;
  char cadc[1350];
  byte ramint[256];
  byte ramintoc[256];
  S19 mi_s19;

  printf ("\nNombre fichero: ");
  gets(cad);
  if (abrir_s19(cad,&mi_s19,1)==0) {   /* Si se ha producido un error */
    caderror=(char *)geterrors19();    /* Leer la cadena de error     */
    printf ("Error:%s\n",caderror);   /* Imprimir mensaje de error   */
    return 0;
  }
```

```
    printf ("\n");

    s19toramint(mi_s19,ramint,ramintoc);
    raminttoc(ramint,cadc);
    printf ("%s\n",cadc);

    cerrar_s19(mi_s19);
    return 0;
}
```

3.4. INTERFAZ

- *int abrir_s19(char *fich,S19 *ss19,int modo);* Abrir un archivo S19 para trabajar con el
- *void cerrar_s19(S19 ss19);* Cerrar archivo S19
- *int leerdir_s19(S19 ss19,int nreg, unsigned int *dir);* Leer el campo direccion del registro especificado
- *int leercod_s19(S19 ss19,int nreg, byte *cod, byte *tam, byte *crc);* Leer campo código del registro indicado
- *byte calcular_crc(byte *bloque, int tam);* Calcular CRC de un bloque de datos
- *unsigned int getnbytes19(S19 ss19);* Devolver el númeo de bytes del fichero S19
- *unsigned int getnregs19(S19 ss19);* Devolver el número de registros
- *char *geterrors19();* Obtener la cadena de error producida en la última apertura del fichero
- *void s19toramint(S19 f,byte *ramint, byte *ramintoc);* Obtener la matriz de código comprendida entre las direcciones 0-255. El resto se desprecia.
- *void s19toeeprom(S19 f,byte *eeprom, byte *eepromoc);* Obtener la matriz de código comprendida entre las direcciones B600-B7FF
- *int situacion_progs19(S19 fs19, int *ov);* Comprobar la situación de un programa: RAM interna, EEPROM o RAM externa
- *void raminttoc(byte *ramint, char *cadc);* Generar una matriz en C con el codigo de un programa para la RAM interna.

3.5. LIMITACIONES

Aunque este módulo es genérico para trabajar con ficheros .S19, las funciones s19toeeprom, situacion_progs19 y raminttoc suponen que se está trabajando con microcontroladores 68hc11 con la ram interna de 256 bytes y la eeprom de 512bytes en las posiciones 0xB600-0xB7FF. Para otros modelos de micro estas funciones no son válidas y hay que programárselas “a pelo”.

Capítulo 4

MODULO BOOTSTRAP

4.1. INTRODUCCION

El módulo Bootsrap permite controlar el 68hc11 cuando éste se encuentra en modo bootstrap, permitiéndonos cargar programas en la ram interna o saltar a eeprom. Este módulo presupone que se está utilizando la tarjeta CT6811 o compatibles y que por ello se puede realizar un reset 'software' utilizando la señal DTR del puerto serie. Con este módulo se pueden realizar programas como el downmcu, mcboot o ctreset.

4.2. UTILIZACION

4.2.1. Funciones para hacer reset de la ct6811

Las funciones resetct6811() y okreset() permiten hacer un reset software de la CT6811. La primera función realiza sólo un reset y es válida para cualquier modo de funcionamiento de la CT6811. Para que sea efectivo tiene que estar el jumper JP7 de la CT6811 en la posición ON.

La función okreset() hace un reset y espera a que se reciba la señal de BREAK proveniente del modo bootstrap del 68hc11. Si no se recibe el break puede ser debido a que el 68hc11 no está en modo bootstrap o que existe un fallo en las comunicaciones. En la sección 4.3.1 se puede ver un ejemplo en el que se utilizan estas dos funciones.

4.2.2. Funciones para saltar a la ram y la eeprom

Las funciones *jump_eeprom()* y *jump_ram()* son similares, salvo que una salta a la eeprom y la otra a la ram. Ambas funciones realizan un reset de la ct6811 y esperan a recibir la señal de Break que indica que la CT6811 se encuentra en el modo bootstrap. Las funciones devuelve un 1 si se ha podido realizar la operación o un 0 si no se ha recibido la señal de Break.

4.2.3. Funciones para cargar programas en la ram interna

Con cargar_ramint() se carga una matriz de 256 datos en la ram interna del 68hc11. Cada vez que se recibe el eco de dato enviado, se llama a la función pasada como parametro, que normalmente actualizará una barra de estado indicando el porcentaje de carga. Esta función devuelve 1 si el programa se ha cargado con éxito ó 0 si se ha producido un error. En ese caso, llamando a la función *getloaderror* se obtiene la cadena de error. En la sección 4.3.2 se muestra un ejemplo que carga una matriz de datos en la ram interna del 6811.

La función cargars19_ramint() carga un fichero .S19 en la ram interna. Este fichero previamente ha tenido que ser abierto utilizando la función abrir_s19() del módulo bootstrap. En la sección 4.3.3 se muestra un programa que carga el fichero ledp.s19 en la ram interna. La función devuelve 0 si se ha producido algún error y en ese caso llamando a *getloaderror* se obtiene la cadena de error.

4.2.4. Otras funciones

El plazo de tiempo para esperar que se reciba una señal de break se puede cambiar con la función *set_break_timeout()*. Hay que especificar el tiempo en microsegundos (1 seg = 1000000microsegundos). Esta función es útil si se quiere dar la posibilidad de realizar un reset manual, asignando por ejemplo un valor de 5 segundos para el plazo de tiempo. Cada vez que se invoque a alguna función que tenga que esperar la recepción de un break, como por ejemplo okreset(), se esperará ese tiempo. Si transcurrido el plazo no llega el break se determinará error de timeout.

Cuando se envía un programa a la ram interna, el 68hc11 hace un eco de todo lo que recibe. Ese eco recibido se comprueba con el dato enviado para ver si se ha recibido lo mismo que se ha enviado o no. Esta opción de comprobación se puede desconectar llamando a la función set_eco_checking(). Los valores que se le pasan a esta función son ON para establecer comprobación del eco y OFF para no hacer caso.

Cuando se ha producido un error al llamar a alguna de las funciones del módulo bootstrap se puede llamar a la función *getloaderror* que devuelve la cadena asociada al último error producido.

4.3. EJEMPLOS

4.3.1. Programa ctd

Este ejemplo es el precursos del programa ctdetect.

```
/*********************************************************************/
/* CTD.C  (c) Microbotica, S.L Enero 1998.                           */
/* ----------------------------------------------------------------  */
/* Programa ejemplo de prueba de la libreria Bootstrp.c              */
/* Se trata del clasico programa para autodetectar la CT6811.        */
/*********************************************************************/

#include "serie.h"
```

```
main()
{
  int puerto=1;
  int ok=0,nok=0;
  int intento;
  char cad[80];

  set_break_timeout(100000);  /* Poner el Timeout a 100mseg */

  printf ("Autodeteccion de la tarjeta CT6811\n");
  printf ("Se busca en COM2\n\n");

  printf ("Explorando puerto COM%u\n",puerto+1);
  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
    exit(1);
  }
  baudios(7680);

  for (intento=1; intento<=3; intento++) {
    printf (cad,"Intento%u.....",intento);
    resetct6811();
    if (okreset()==0) {
      printf ("TIMEOUT\n");
      nok++;
    }
    else {
      printf ("OK\n");
      ok++;
    }
  }
  printf ("\n");
  if (ok==3) printf ("Tarjeta CT6811 detectada en COM%u\n\n",puerto+1);
  cerrar_puerto_serie();
}
```

4.3.2. Programa ledp

Ejemplo que carga en la ram interna un programa que hace parpadear el led de la CT6811.

```
/*
 ***************************************************************************
 * LEDP.C      (c) Microbotica, S.L. Febrero 1998.                         *
 ***************************************************************************
 *                                                                         *
 *                                                                         *
 *   Se carga en la CT6811 un programa que hace parpadear el LED. Este     *
 * programa no se toma de un archivo .S19 sino que se encuentra dentro del *
 * propio codigo.                                                          *
 *                                                                         *
```

```
 *********************************************************************************
*/

#include "stdio.h"
#include "serie.h"
#include "bootstrp.h"

int i;

void prueba()
{
  static int n=0;
  static int p=0;

  print("*");

  if (n==24) {
    n=0;
    p+=10;
  }
  n++;
  i++;
}

main()
{
  /* Matriz que contiene el programa a enviar a la CT6811 */

  byte programint[256]={
0xB6,0x10,0x00,0x88,0x40,0xB7,0x10,0x00,0x18,0xCE,0x5F,0xFF,0x18,0x09,0x18,
0x8C,0x00,0x00,0x26,0xF8,0x20,0xEA,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,};

  abrir_puerto_serie(COM2);  /* Abrir sesion con puerto serie COM2 */
  print ("\nCargando programa: ");
```

```
    set_break_timeout(500000);

    if (cargar_ramint(programint,prueba)==0) {
      printf ("\nError:%s",getloaderror());
      cerrar_puerto_serie();
      return 0;
    };

    printf ("\nOK!\n");

    cerrar_puerto_serie();
    return 0;

  }
```

4.3.3. Programa ramint

Programa para cargar el fichero ledp.s19 en la ram interna de la CT6811.

```
/*
*************************************************************************
* RAMINT.C (c) Microbotica, S.L Febrero 1998. *
*************************************************************************
* Ejemplo de prueba de la libreria BOOTSTRP.C Se carga en la ram *
* interna de la CT6811 el fichero LEDP.S19. *
*************************************************************************
*/

#include <stdio.h>

#include "serie.h"
#include "s19.h"
#include "bootstrp.h"

typedef unsigned int uint;

int i=0;
int n=0;

void carga()
{
 if (n==16 || i==255) {
 n=0;
 print ("*");
 }
 i++;
```

```
 n++;
}

main()
{
 char cad[80];
 char *caderror;
 S19 fs19;

 if (abrir_s19("ledp.s19",&fs19,1)==0) {
 caderror=(char *)geterrors19();
 printf ("Error:%s\n",caderror);
 return 0;
 }
 printf ("\n");

 if (abrir_puerto_serie(COM2)==0) {
 printf ("Error al abrir puerto serie:%s\n",getserial_error());
 exit(1);
 }

 set_break_timeout(500000);

 printf("0%% 50%% 100%%\n");
 print ("...............\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");

 if (cargars19_ramint(fs19,carga)==0) {
 printf ("\nError:%s\n",getloaderror());
 }
 else {
printf (" OK!!\n\n");
 }

 cerrar_s19(fs19);
 cerrar_puerto_serie();

 return 0;
}
```

4.4. INTERFAZ

- *void set_break_timeout(unsigned long timeout);* Establecer el plazo de tiempo en microsegundos para determinar que una señal de break no ha llegado.
- *void set_eco_checking(byte eco);* Establecer si la comprobación del eco está activa o no al cargar un programa en la ram interna.
- *void resetct6811()*; Realizar un reset software de la ct6811
- *int okreset()*; Realizar un reset software de la ct6811 y esperar a recibir la señal de BREAK. Si el reset se ha realizado correctamente se devuelve un 1 y 0 en caso de error.
- i*nt jump_eeprom()*; Realizar un reset y saltar a la eeprom interna. Se devuelve 0 si no se ha realizado el reset correctamente.
- *int jump_ram();* Realizar un reset y saltar a la ram interna. Se devuelve 0 si no se ha realizado el reset correctamente
- *int cargar_ramint(byte *ramint, void (*car)());* Cargar un programa en la ram interna. Cada vez que se recibe el eco de un dato enviado se llama a la función pasada como parámetro.
- *int cargars19_ramint(S19 fs19, void (*car)())*; Cargar un fichero .S19 en la ram interna. Cada vez que se recibe el eco de un dato enviado se llama a la función pasada como parámetro.
- *char *getloaderror();* Devolver la cadena asociada al último error producido.

4.5. LIMITACIONES

Este módulo sólo funciona con los micros de la familia 68hc11 que tienen 256 bytes de memoria ram, como son los modelos A0, A1 y E2. Con el modelo E1 y E9 no es posible utilizar las funciones de carga de programas en la ram interna.

Capítulo 5

MODULO CTCLIENT

5.1. INTRODUCCION

El módulo ctclient es realmente el módulo que nos va a permitir controlar las aplicaciones desde el PC. Contiene todas las funciones necesarias para acceder a la memoria del 68hc11, tanto en lectura como en escritura. Para poder trabajar con este módulo es necesario que el servidor ctserver se encuentra ya cargado en el 68hc11, bien en la ram o bien en la eeprom. En todos los ejemplos presentados el ctserver se carga en la ram interna, aunque en las aplicaciones reales se grabe en la memoria eeprom. Para cargar el ctserver en la ram interna se utilizan funciones de los módulos anteriormente descritos.

5.2. UTILIZACION

5.2.1. Funciones STORE para almacenar datos en el 6811

Existen tres funciones para almacenar datos en la memoria del 68HC11. La función store() almacena un byte, store_block() almacena un bloque de bytes y store_block_eeprom() almacena un bloque en la memoria EEPROM. En la figura 5.1 se muestra un ejemplo de utilización. El primer parámetro indica el valor a almacenar y el segundo la dirección del mapa de memoria del 6811 donde almacenarlo. En el fichero r6811pc.h se han definido todas las constantes necesarias para acceder a los recursos del 68HC11. Así en vez de utilizar la dirección 0x1000 para referirse al puerto A se puede utilizar la etiqueta PORTA.

La función store_block() permite almacenar un bloque de datos. Además, se llama a una función que se le pasa como parámetro cada vez que se almacena un byte. Normalmente se le pasa una función nula. En la figura 5.2 se muestra un ejemplo de utilización que guarda la cadena “hola” a partir de la dirección

```
store(0x40, 0x1000); /* Encender el led de la CT6811 */

store(0x40,PORTA); /* Hace lo mismo pero utilizando la constante PORTA, definida en e
```

Figura 5.1: Ejemplo de utilización de store()

```
void nada()
 /* Funcion que no hace nada */
{
}

main()
{
   byte block[]={'H','O','L','A'};

  /* ..... */

  store_block(block,4,0x8000,nada);
}
```

Figura 5.2: Ejemplo de utilización de store_block()

0x8000 del 68hc11 (Se supone que se tiene memoria externa).

La función store_block_eeprom() funciona de la misma manera que store_block() pero grabando el dato en la memoria EEPROM. Se puede ver un ejemplo de utilización en el programa de ejemplo de la sección 5.3.4.

Hay que tener cuidado con las funciones store para no "machacar" el propio programa ctserver. Por ejemplo si se hace un store en la direccion 0x10, se modifica el programa ctserver por lo que casi seguro se perderá la conexión o el servidor funcionará incorrectamente (Los efectos pueden ser imprevisibles). Por ello la función store se emplea para acceder a los registros internos del 68hc11 o a la memoria externa, pero no conviene utilizarlo con la ram interna (Salvo que el servidor se encuentre en RAM externa).

5.2.2. Funciones LOAD para leer datos del 6811

Estas funciones permiten leer datos de la memoria del 68hc11. Existen dos funciones: *load()* para leer un byte y *load_block()* para leer un bloque de datos. En la figura 5.3 se muestra un ejemplo de utilización.

5.2.3. Función de ejecución

Mediante la función *execute()* se puede ejecutar un trozo de código situado en una dirección de memoria del 68hc11. Después de llamar a esta función la conexión con el ctserver se pierde porque se deja de ejecutar. Esta función es muy util para cargar programas en la ram externa y ejecutarlos.

5.2.4. Funciones de control de la conexión

Para saber si el ctserver está vivo, se emplea la función *check_conexion()*, la cual envía un valor al ctserver y éste responde. Si desde el PC se recibe el valor correcto, check_conexion() devuelve un 1 indicando que la conexión está activa. Si se pierde devuelve un 0. En todos los programas que utilicen el

```
void nada()
 /* Funcion que no hace nada */
{
}

main()
{
   byte dato;
   byte datos[10];

  /* ..... */

  load(&dato,PORTA); /* Leer puerto A */
  load_block(datos,10,0x8000,nada); /* Leer un bloque de 10 bytes */
}
```

Figura 5.3: Ejemplo de utilización de las funciones load.

ctserver es necesario llamar de cuando en cuando a esta función para detectar por ejemplo si el usuario a apretado el botón de reset y no hay conexión con la ct6811.

La función *hay_conexion()* guarda el estado de la última conexión con el ctserver. Cada vez que se llama a check_conexion() o a alguna instrucción de load() el estado de la conexión con el ctserver se detecta. La principal diferencia entre hay_conexion() y check_conexion() es que la primera no envía nada a la ct6811, mientras que check_conexion() envía un valor y espera recibir respuesta.

5.3. EJEMPLOS

En todos los ejemplos lo primero que se hace es cargar el ctserver en la ram interna. Esto se realiza con la función cargar_ctserver() que contiene el programa ctserver.s19 en una matriz. Esto se puede realizar con el programa cts19toc que convierte un programa .s19 en una matriz de este tipo.

Puesto que la función cargar_ctserver() es la misma para todos los ejemplos, sólo se presenta en el primero de ellos, eliminándose en el resto.

5.3.1. Programa kledp: parpadeo del led de la CT6811

```
/*************************************************************************/
/* KLEDP.C     (c) Microbotica, S.L. Marzo 2000                          */
/*************************************************************************/
/*                                                                       */
/* Ejemplo de prueba de la libreria CTCLIENT.C                           */
/* Se trata del tipico programa del ledp parpadeante.                    */
/*   Primero se carga el CTSERVER y luego se establece la conexion. El   */
/* CTSERVER se carga a partir de una matriz interna                      */
/*                                                                       */
/*************************************************************************/
```

```
#include "r6811pc.h"
#include "serie.h"
#include "bootstrp.h"

int i=0;
int n=0;

void accion_load()
/*****************************************/
/* Accion al realiar al cargar el servidor  */
/*****************************************/
{
  if (n==16 || i==255) {
    n=0;
    print ("*");
  }
  i++;
  n++;
}

int cargar_ctserver()
/*************************************************************/
/* Cargar el CTSERVER. El CTSERVER que se carga esta configurado  */
/* para la velocidad de 7680 baudios.                          */
/*************************************************************/
{

byte cts9600[256]={
0x20,0x05,0x00,0x00,0x00,0x00,0x00,0xCE,0x10,0x00,0x1F,0x2E,0x40,0xFC,0x86,
0x30,0xA7,0x2B,0xCE,0x10,0x00,0x8D,0x66,0x97,0x06,0x81,0x44,0x27,0x13,0x81,
0x43,0x27,0x54,0x81,0x41,0x27,0x17,0x81,0x42,0x27,0x0E,0x81,0x45,0x27,0x6A,
0x7E,0x00,0x12,0x86,0x4A,0x8D,0x50,0x7E,0x00,0x12,0x7C,0x00,0x06,0x20,0x03,
0x7F,0x00,0x06,0x8D,0x4A,0x18,0xDF,0x02,0x8D,0x45,0x18,0xDF,0x04,0x18,0x8C,
0x00,0x00,0x27,0x23,0x18,0xDE,0x02,0x96,0x06,0x4D,0x26,0x07,0x18,0xA6,0x00,
0x8D,0x28,0x20,0x05,0x8D,0x1D,0x18,0xA7,0x00,0x18,0x08,0x18,0xDF,0x02,0x18,
0xDE,0x04,0x18,0x09,0x18,0xDF,0x04,0x20,0xD7,0x7E,0x00,0x12,0x8D,0x14,0x18,
0xDF,0x02,0x18,0x6E,0x00,0x1F,0x2E,0x20,0xFC,0xA6,0x2F,0x39,0x1F,0x2E,0x80,
0xFC,0xA7,0x2F,0x39,0x36,0x37,0x8D,0xEE,0x16,0x8D,0xEB,0x18,0x8F,0x33,0x32,
0x39,0x8D,0xF2,0x18,0xDF,0x02,0x8D,0xED,0x18,0xDF,0x04,0x18,0x8C,0x00,0x00,
0x27,0x34,0x18,0xDE,0x02,0xC6,0x16,0xF7,0x10,0x3B,0x18,0xE7,0x00,0xC6,0x17,
0xF7,0x10,0x3B,0x8D,0x28,0xC6,0x02,0xF7,0x10,0x3B,0x8D,0xBD,0x18,0xA7,0x00,
0xC6,0x03,0xF7,0x10,0x3B,0x8D,0x17,0x8D,0xB8,0x18,0x08,0x18,0xDF,0x02,0x18,
0xDE,0x04,0x18,0x09,0x18,0xDF,0x04,0x20,0xC6,0x7F,0x10,0x3B,0x7E,0x00,0x12,
0x18,0x3C,0x18,0xCE,0x0D,0x10,0x18,0x09,0x18,0x8C,0x00,0x00,0x26,0xF8,0x18,
0x38,0x39,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,};


byte cts7680[256]={
0x20,0x05,0x00,0x00,0x00,0x00,0x00,0xCE,0x10,0x00,0x1F,0x2E,0x40,0xFC,0xCE,
```

```
0x10,0x00,0x8D,0x66,0x97,0x06,0x81,0x44,0x27,0x13,0x81,0x43,0x27,0x54,0x81,
0x41,0x27,0x17,0x81,0x42,0x27,0x0E,0x81,0x45,0x27,0x6A,0x7E,0x00,0x0E,0x86,
0x4A,0x8D,0x50,0x7E,0x00,0x0E,0x7C,0x00,0x06,0x20,0x03,0x7F,0x00,0x06,0x8D,
0x4A,0x18,0xDF,0x02,0x8D,0x45,0x18,0xDF,0x04,0x18,0x8C,0x00,0x00,0x27,0x23,
0x18,0xDE,0x02,0x96,0x06,0x4D,0x26,0x07,0x18,0xA6,0x00,0x8D,0x28,0x20,0x05,
0x8D,0x1D,0x18,0xA7,0x00,0x18,0x08,0x18,0xDF,0x02,0x18,0xDE,0x04,0x18,0x09,
0x18,0xDF,0x04,0x20,0xD7,0x7E,0x00,0x0E,0x8D,0x14,0x18,0xDF,0x02,0x18,0x6E,
0x00,0x1F,0x2E,0x20,0xFC,0xA6,0x2F,0x39,0x1F,0x2E,0x80,0xFC,0xA7,0x2F,0x39,
0x36,0x37,0x8D,0xEE,0x16,0x8D,0xEB,0x18,0x8F,0x33,0x32,0x39,0x8D,0xF2,0x18,
0xDF,0x02,0x8D,0xED,0x18,0xDF,0x04,0x18,0x8C,0x00,0x00,0x27,0x34,0x18,0xDE,
0x02,0xC6,0x16,0xF7,0x10,0x3B,0x18,0xE7,0x00,0xC6,0x17,0xF7,0x10,0x3B,0x8D,
0x28,0xC6,0x02,0xF7,0x10,0x3B,0x8D,0xBD,0x18,0xA7,0x00,0xC6,0x03,0xF7,0x10,
0x3B,0x8D,0x17,0x8D,0xB8,0x18,0x08,0x18,0xDF,0x02,0x18,0xDE,0x04,0x18,0x09,
0x18,0xDF,0x04,0x20,0xC6,0x7F,0x10,0x3B,0x7E,0x00,0x0E,0x18,0x3C,0x18,0xCE,
0x0D,0x10,0x18,0x09,0x18,0x8C,0x00,0x00,0x26,0xF8,0x18,0x38,0x39,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,};


  print ("\nCargando CTSERVER:\n\n");
  set_break_timeout(500000);
  printf("0%%     50%%   100%%\n");
  print ("...............\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");

  if (cargar_ramint(cts9600,accion_load)==0) {
    printf ("\nError:%s\n",getloaderror());
    return 0;
  };
  printf ("\nOK!\n");
  return 1;
}

void mover_palito()
{
  static char fase[5]={'-','\\','|','/'};
  static int pf=0;

  print ("\b\b\b\b\b\b\b\b\b\bEstablecida");
  printcar(' ');
  printcar(fase[pf]);
  print("\b\b");;  /* Dibujar el palito que gira */
  pf=(pf+1)% 4;
}

main()
{
  byte valor;
  char c;

  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
```

```
    exit(1);
  }
  abrir_consola();
  baudios(7680);
  if (cargar_ctserver()==0) {
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  baudios(9600);
  usleep(200000);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  valor=0xFF;
  /* ---- Bucle principal --- */
  do {
    check_conexion();         /* Comprobar conexion */
    if (hay_conexion()) {
      store(valor,PORTA);     /* Mandar valor al puerto A */
      valor^=0x40;
      mover_palito();
    }
    else {
      print ("\b\b\b\b\b\b\b\b\b\b\bPerdida....  \b\b");
    }
  } while (!kbhit());
  if (!hay_conexion()) print ("\n\nConexion perdida..\n");
  getch();
  cerrar_puerto_serie();
  cerrar_consola();
  printf ("\n\n");
}
```

5.3.2. Programa kad: Lectura de los conversores A/D

```
/*
 ***************************************************************************
 * Kad.C     (c) Microbótica, S.L. Marzo 2000                              *
 ***************************************************************************
 * Ejemplo de prueba de la libreria CTCLIENT.C                             *
 * Lectura de 4 canales A/D                                                *
 ***************************************************************************
*/
```

```
#include "r6811pc.h"
#include "serie.h"
#include "bootstrp.h"


void nada()
{
}

void pad()
{
  byte ad;
  byte muestra[4];
  byte x;
  int i;
  char cad[4][4];
  byte hola[50];

  store(0x80,OPTION);  /* Configurar conversor */
  store(0x20,ADCTL);
  print("  ");
  do {
    load(&ad,ADCTL);  /* Comprobar si conexion realizada */
    if (hay_conexion()) {
      if ((ad & 0x80)==0x80) {
        load_block(muestra,4,ADR1,nada);
        sprintf(hola,"%2X%2X%2X%2X",muestra[0],muestra[1],muestra[2],muestra[3]);
        for (i=0; i<4; i++) {
          muestra[i]=muestra[i]&0xFF;
          bytetochar2(muestra[i],cad[i]);
          print(cad[i]); print(" ");
        }
        print ("\b\b\b\b\b\b\b\b\b\b\b\b");
      }
    }
  } while (hay_conexion() && !kbhit());
}

main()
{
  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
    exit(1);
  }
  abrir_consola();
  baudios(7680);
  if (cargar_ctserver()==0) {
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
```

```
  usleep(200000);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }

  pad();
  if (!hay_conexion()) print ("\n\nConexion perdida..\n");
  getch();

  cerrar_puerto_serie();
  cerrar_consola();
  printf ("\n\n");
}
```

5.3.3. Programa ctdumpee: Volcado de la memoria eeprom

```
/*
 ****************************************************************************
 * CTDUMPEE.C  (c) Microbotica, S.L.Marzo 2000                              *
 ****************************************************************************
 *                                                                          *
 * Programa ejemplo de las llamadas a los servicios del CTSERVER.           *
 *                                                                          *
 * El programa vuelca el contenido de la EEPROM en ASCII                    *
 *                                                                          *
 ****************************************************************************
*/

#define EEPROM 0xB600

#include "stdio.h"

#include "r6811pc.h"
#include "serie.h"
#include "bootstrp.h"


void car_rec()
{
  static nbyte=0;

  nbyte++;
  if (nbyte==8) {
    print ("*");
    nbyte=0;
  }
```

```
}

main()
{
  byte eeprom[512];
  byte ok;
  int x,y;
  int i;

  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
    exit(1);
  }
  abrir_consola();
  baudios(7680);
  if (cargar_ctserver()==0) {
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  usleep(200000);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  printf ("\nPrograma para volcar la EEPROM en ASCII\n\n");
  print ("Leyendo:");
  print ("..............................................................");
  for(i=0; i<64; i++) {
    print ("\b");
  }
  /* ---- Leer memoria eeprom y meterla en el buffer ----- */
  ok=load_block(eeprom,512,EEPROM,car_rec);
  if (ok) {            /* Si no ha habido errores .. Imprimir */
    printf ("\n\n");
    x=0;
    for (i=0; i<512; i++) {
      if (eeprom[i]>31 && eeprom[i]<127) printf ("%c",eeprom[i]);
      else printf (".");
      x++;
      if (x==32) {
        x=0;
        printf ("\n");
      }
    }
    printf ("\nVolcado completado\n\n");
  }
```

```
  else printf ("\nConexion PERDIDA\n\n");
  cerrar_puerto_serie();
  cerrar_consola();
  printf ("\n\n");
}
```

5.3.4. Programa ctsaveee: Grabación de datos en la memoria eeprom

```
/*
 ************************************************************************
 * CTSAVEEE.C     (c) Microbótica, S.L. Marzo 2000                      *
 ************************************************************************
 *                                                                      *
 * Ejemplo de prueba de la libreria CTCLIENT.C                          *
 *                                                                      *
 * Grabar en la memoria EEPROM una cadena                               *
 *                                                                      *
 ************************************************************************
*/


#include "r6811pc.h"
#include "serie.h"
#include "bootstrp.h"

void accion_grabar()
{
  print ("*");
}

void grabar_eeprom()
{
  int ok;
  byte cad[255];

  printf ("Introduzca cadena a grabar en la EEPROM:\n");
  cerrar_consola();
  gets(cad);
  abrir_consola();
  ok=store_block_eeprom(cad,strlen(cad),0xB600,accion_grabar);
  if (ok==1) printf ("OK\n");
  else printf(" Error!!\n");
  printf ("\n");
}

main()
{
  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
    exit(1);
  }
```

```
  abrir_consola();
  baudios(7680);
  if (cargar_ctserver()==0) {
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  usleep(200000);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }

  grabar_eeprom();

  cerrar_puerto_serie();
  cerrar_consola();
  printf ("\n\n");
}
```

5.3.5. Programa spi1: Acceso al puerto F a través del spi

En muchas aplicaciones es necesario disponer de más puerto para la CT6811, sobre todo si se ha ampliado con RAM externa porque se pierden los puertos B y C. A través del SPI es posible tener un puerto de entrada y otro de salida. Estos puertos se denominan puertos F. El puerto F1 es de salida y el puerto F2 es de entrada. Este ejemplo supone que se tienen conectados 8 leds en el puerto F1.

```
/*
 ***************************************************************************
 * Spi1.C     (c) Microbótica, S.L  Marzo 2000                             *
 ***************************************************************************
 * Ejemplo de prueba de la libreria CTCLIENT.C                              *
 *                                                                          *
 *  Se mueven los leds imitando al coche fantastico...                      *
 ***************************************************************************
*/

#include "r6811pc.h"
#include "serie.h"

#include "bootstrp.h"


void puertof1(byte d)
{
  byte temp;
```

```
  store(d,SPDR);
  load(&temp,SPSR);
  load(&temp,SPDR);
  store(0xFF,PORTD);
  store(0x00,PORTD);
}

void spi1()
{
  int i;

  /* Configuracion SPI */
  store(0x38,DDRD);
  store(0x5C,SPCR);
  store(0xFF,PORTD);
  do {
    for (i=1; i<=0x80; i=i<<1) {
      puertof1(i);
    }
    for (i=0x80; i>0; i=i>>1) {
      puertof1(i);
    }
  } while (!kbhit() && hay_conexion());
}

main()
{
  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
    exit(1);
  }
  abrir_consola();
  baudios(7680);
  if (cargar_ctserver()==0) {
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  usleep(200000);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  spi1();
  if (!hay_conexion()) print ("\n\nConexion perdida..\n");
  getch();
```

```
  cerrar_puerto_serie();
  cerrar_consola();
  printf ("\n\n");
}
```

5.3.6. Programa spi2: Acceso al puerto F a través del spi

```
/*
 ******************************************************************************
 * SPI2.C     (c) Microbótica, S.L. Marzo 2000                                *
 ******************************************************************************
 * Ejemplo de prueba de la libreria CTCLIENT.C                                *
 * Se muestra el estado de los switches en los leds y en pantalla.            *
 ******************************************************************************
*/

#include "r6811pc.h"
#include "serie.h"
#include "bootstrp.h"


void puertof1(byte d)
{
  byte temp;

  store(d,SPDR);
  load(&temp,SPSR);
  load(&temp,SPDR);
  store(0xFF,PORTD);
  store(0x00,PORTD);
}

void puertof2(byte *d)
{
  byte temp;

  store(0x00,PORTD);
  store(0xFF,PORTD);
  store(0,SPDR);
  load(&temp,SPSR);
  load(d,SPDR);
}

void spi2()
{
  byte i;
  byte d;
  char cad[3];

  store(0x38,DDRD);
```

```
  store(0x5C,SPCR);
  store(0xFF,PORTD);
  print ("Valor leido: ");
  i=0;
  do {
    if (hay_conexion()) {
      puertof1(i);
      bytetochar2(i,cad);
      print (cad); print("\b\b");
    }
    puertof2(&i);
    i=i & 0xFF;
  } while (!kbhit() && hay_conexion());
}

main()
{
  if (abrir_puerto_serie(COM2)==0) {
    printf ("Error al abrir puerto serie:%s\n",getserial_error());
    exit(1);
  }
  abrir_consola();
  baudios(7680);
  if (cargar_ctserver()==0) {
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  usleep(200000);

  check_conexion();        /* Comprobar conexion */
  if (!hay_conexion()) {
    printf("No hay conexion");
    cerrar_consola();
    cerrar_puerto_serie();
    exit(1);
  }
  spi2();
  if (!hay_conexion()) print ("\n\nConexion perdida..\n");
  getch();
  cerrar_puerto_serie();
  cerrar_consola();
  printf ("\n\n");
}
```

5.4. INTERFAZ

- *int load(byte *valor, uint dir)* Leer una posicón de la memoria del 6811 y devolver el valor
- *int load_block(byte *buff, uint tam, uint dir, void (*reccar)());* Leer un bloque de datos de la me-

moria del 6811

- *void store(byte valor, uint dir);* Almacenar un valor en la posición de memoria del 6811 indicada.
- *void store_block(byte *buff, uint tam, uint dir, void (*sendcar)());* Almacenar un bloque de datos en la memoria del 6811
- *int store_block_eeprom(byte *buff, uint tam, uint dir, void (*eeprcar)());* Grabar un bloque de datos en la memoria EEPROM del 6811.
- *int check_conexion();* Comprobar si el servidor ctserver vive y por tanto si la conexión se mantiene.
- *void execute(uint dir);* Empezar a ejecutar a partir de la dirección indicada
- *int hay_conexion();* Comprobar el estado de la última conexión.