Cika Electrónica

Nota de Aplicación: CAN-073
Título: **Control de Personal con Ramtron VRS51L3074**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	02/03/07	idea original en CAN-024, port con modificaciones mayores

Nos encontramos esta vez para desarrollar una herramienta de control de personal, destinada a registrar el horario de ingreso y egreso mediante la identificación por elementos RFID, es decir, las conocidas tarjetas de proximidad y los modernos llaveros. El empleado recibe confirmación visual y auditiva, oyendo una chicharra (o un mensaje de voz mediante un chip de Aplus) y observando en un display gráfico su nombre y la hora y fecha en la cual se registra su tarjeta. El interesado en controlar al empleado, puede observar los horarios en orden decreciente en el mismo display, u obtener un listado con los campos separados por TABs, el cual puede procesar y filtrar automáticamente con cualquier herramienta orientada a procesar registros (*awk*, por ejemplo en ambiente Unix), o incluso manualmente con cualquier planilla de cálculo.

Aprovechamos el desarrollo de las notas de aplicación CAN-059 y CAN-060 para conectar nuestro display, CAN-068 para el manejo de la memoria no-volátil FRAM, CAN-069 para la implementación de software timers, CAN-062 para el Processor Companion que usamos como RTC, y CAN-070 para el manejo de interfaz serie; como lector utilizaremos el GP8F-R2. Para un control de acceso de muy bajo costo basado en este mismo lector RFID, el lector interesado puede consultar la CAN-072.

Operación remota y configuración

El equipo se controla por el puerto serie, en el cual recibe comandos terminados por un retorno de carro, entregando o no respuesta según corresponda. La sintaxis es rígida, dado que se orienta a control automático, pero es simple como para realizarlo manualmente. Sin embargo, no soporta edición, debe tipearse el comando sin errores, aunque existe un mínimo de chequeo para sobrevivir a errores simples.

Los comandos (y sus formatos) soportados son:

`T<CR>` devuelve un string con fecha y hora actual, por ejemplo: `02 MAR 07 10:20:05<CR><LF>`

`ST dd mm aa hh mm<CR>` ajusta fecha y hora y luego la muestra en el mismo formato que el comando 'T', por ejemplo:

`ST 02 03 07 10 30<CR>` setea la fecha y hora a 2 de marzo del 2007, 10:30:00

`R<CR>` lista los registros de acceso, por ejemplo:

```
02 MAR 07 06:00:21      Hendrix,Jimi<CR><LF>
02 MAR 07 09:00:12      Mas,Alguien<CR><LF>
```

`I<CR>` lista los IDs de la base, de 1 a N, por ejemplo:

```
01    Hendrix,Jimi<CR><LF>
02    Mas,Alguien<CR><LF>
```

`D x<CR>` borra un ID de la base. El parámetro 'x' es un número entero positivo decimal, de 1 a N. Ejemplo:

```
D 2<CR>
```

`PR<CR>` purga los registros (borra todos los datos) de acceso.

`PI<CR>` purga los IDs (borra todos los datos).

`L nombre<CR>` coloca al equipo en modo 'aprender', el primer RFID ingresado que no figure anteriormente en la base será aprendido con ese nombre e ingresado en la base. El parámetro 'nombre' es un string de hasta 16 caracteres, desde el tercero hasta el <CR> de la línea de comando. Ejemplo:

```
L Paz,Haya<CR>
```

Descripción del GP8F-R2

El módulo GP8F-R2 lee tarjetas o tags RFID read-only de 64-bits, código Manchester a 125KHz. Posee una salida para conectar un LED, que permanece encendido y aumenta su intensidad al aproximar un RFID. El ID correspondiente se entrega por un terminal en formato 8 bits serie asincrónico, a 9600 bps, sin paridad, a nivel lógico TTL, lo cual lo hace excelente para ser leído desde un micro. El formato lógico responde al siguiente esquema, en ASCII:

```
<STX> <DATA (10 bytes)> <CR> <LF> <ETX>
```

STX: ASCII Start of Text (0x02)
ETX: ASCII End of Text (0x03)
CR: ASCII Carriage Return (0x0D)
LF: ASCII Line Feed (0x0A)

El campo de DATA es una representación en ASCII del ID del RFID, representando 5 bytes binarios (40 bits) de la siguiente forma:

byte ASCII
C1 43 31

Por ejemplo, el ID `60 22 57 C0 31,` se transmite de la siguiente forma:

```
02 36 30 32 32 35 37 43 30 33 31 0D 0A 03
```

Recordemos que este tipo de RFID tags de 64 bits 125 Khz utiliza 9 bits para header, 40 para datos, 14 bits de paridad y uno de stop.

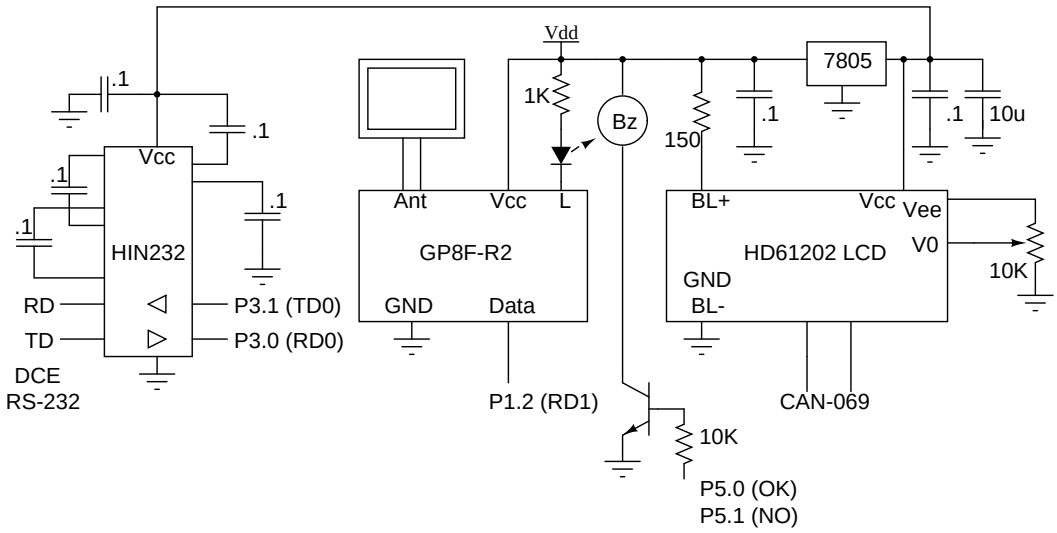
Hardware

Conectamos un display LCD gráfico como indica la CAN-069. Utilizamos el port serie 1 para recibir la información del GP8F-R2, y el port serie 0 para la comunicación con un dispositivo maestro que permita configurar y controlar el equipo. En este caso utilizamos una PC o equivalente por puerto serie, pero bien puede utilizarse un puerto USB con un FT232BM (ver CAN-025) o cualquier otra alternativa.

El Processor Companion se conecta al bus I^2C como indica la CAN-062 .En nuestro caso, utilizamos el kit de desarrollo del VRS51L3074 que lo incluye, junto con alimentación y debugging.

En el esquemático que figura a continuación se muestra la parte del hardware agregada al kit de desarrollo, y no cubierta en otras notas de aplicación. El mismo muestra un buzzer pero en realidad disponemos de dos pines, uno de ellos (*OK*) se activa cuando el RFID es reconocido, el otro (*NO*), en el caso contrario. De este modo, es posible controlar un chip de reproducción de voz de Aplus y emitir un mensaje hablado informando si se permite o no el acceso. De un modo similar, es igual de simple controlar un relé que active el solenoide o traba de una puerta, agregando una salida o tomando la salida *OK*.

El contenido de la información se preserva ante un corte de alimentación mediante la utilización de la FRAM que incluye el VRS51L3074. La tensión de alimentación, *Vdd*, es de 12V.



Software

La primera vez que arranca el sistema, no tendrá datos válidos en las variables que se almacenan en FRAM. Para corregir este problema sin complicar el código, el administrador dispone de la opción de purgar la base de IDs y la de registros, lo cual coloca ambos contadores en cero. Para los alcances de una nota de aplicación, esto no es demasiado problema, si el lector planea hacer un producto comercial, deberá planear algún sistema de detección de inconsistencias o "factory defaults" y obrar en consecuencia.

Cada empleado tendrá un dispositivo RFID que lo identifica (llavero, tarjeta, etc). Para memorizar dichos IDs, el administrador ingresa al sistema y solicita el aprendizaje de un ID, ingresando el nombre del empleado. En este momento, el equipo indica en el display que acerque al lector el RFID, el cual será memorizado en una tabla.

A fin de minimizar los requerimientos de memoria y poder almacenar mayor cantidad de registros, cuando se produce un evento (lectura de un ID válido), se almacena un registro que contiene la información de fecha y el número de orden en la tabla de IDs del empleado cuyo ID se acaba de reconocer. Esto requiere que cada vez que se solicita un listado, el mismo deba ser generado tomando la información de los registros y buscando el nombre del empleado en la tabla de IDs, pero minimiza la cantidad de memoria requerida sin complicar demasiado el proyecto. La información de fecha y hora se guarda en formato BCD, tal cual se recibe del RTC.

Todas las funciones relacionadas con el manejo del display LCD, puede obtenerlas de CAN-069 y CAN-070, las necesitará, junto a las otras mencionadas, para compilar el código.

Primero incluimos algunas definiciones:

```
#define HOLD_TIME      5
#define BEEP_TIME      10

#define NUM_ENTRIES    60
#define NUM_RECORDS    256

#define NAME_LEN       16
```

Las mismas corresponden al tiempo de persistencia de la información en el display en segundos, duración del beep en decenas de milisegundos, cantidad máxima de IDs de empleados, cantidad máxima de registros en memoria, y longitud del nombre. Los tamaños de buffer para la interfaz serie fueron definidos en los header files correspondientes, según CAN-070.

Definimos a continuación las estructuras para guardar los IDs y los registros, más algunas variables globales que utilizaremos más adelante. Hemos incorporado un mecanismo de seguridad en puntero y cuenta de la base de registros, de modo que si por alguna eventualidad llega a ocurrir un reset en medio de una modificación de estas variables, es posible recuperar el valor original.

```
typedef struct {
```

```
   char name[NAME_LEN+1];
   unsigned char rfid[6];
} rfid_record;

typedef struct {
   unsigned int id;
   struct RTCtm time;
} time_record;

struct ptrstruct {
       unsigned int index;
       unsigned int count;
       unsigned int bkindex;
       unsigned int bkcount;
       unsigned char tag;
};

__xdata at 0x8000 rfid_record rfid_base[NUM_ENTRIES];
__xdata at 0x8000+sizeof(rfid_record)*NUM_ENTRIES time_record time_base[NUM_RECORDS];
__xdata at 0x9F00 struct ptrstruct bufpos;
__xdata at 0x9F00+sizeof( struct ptrstruct) struct ptrstruct bufpos2;
__xdata char linebuf[21],serbuf[21+16+5],learnbuf[21];

unsigned int entries,rcrdidx,rcrdcnt;

/* Program states */
#define ESCRACHATE    1
#define LEARN                 2

static int sstate;
```

Escribimos ahora una simple función que nos permita presentar comodamente la información de fecha y hora, la cual se obtiene en BCD del RTC o de un registro, y se transforma a ASCII en un buffer, el cual utilizaremos tanto para la presentación en el display como para emitir los registros por el puerto serie.

```
void printdatetime(__xdata struct RTCtm *rtc)
{
__code const static char *  __code months[]={
"---","ENE","FEB","MAR","ABR","MAY","JUN","JUL","AGO","SEP","OCT","NOV","DIC"
};
__xdata char *ptr;
__code char *cptr;
register unsigned char aux;

       ptr=linebuf;                                  // apunta a destino
       cptr=months[bcd2bin(rtc->tm_mon)];            // apunta a nombre del mes en flash
       aux=rtc->tm_mday;                             // apunta a día en BCD
       *(ptr++)=(aux>>4)+0x30;                       // convierte a ASCII y copia
       *(ptr++)=(aux&0x0F)+0x30;
       *(ptr++)=' ';                                 // separador
       *(ptr++)=*(cptr++);                           // copia nombre del mes
       *(ptr++)=*(cptr++);
       *(ptr++)=*(cptr++);
       *(ptr++)=' ';                                 // separador
       aux=rtc->tm_year;                             // año
       *(ptr++)=(aux>>4)+0x30;
       *(ptr++)=(aux&0x0F)+0x30;
       *(ptr++)=' ';
       aux=rtc->tm_hour;
       *(ptr++)=(aux>>4)+0x30;
       *(ptr++)=(aux&0x0F)+0x30;
       *(ptr++)=':';
       aux=rtc->tm_min;
       *(ptr++)=(aux>>4)+0x30;
       *(ptr++)=(aux&0x0F)+0x30;
       *(ptr++)=':';
       aux=rtc->tm_sec;
       *(ptr++)=(aux>>4)+0x30;
       *(ptr++)=(aux&0x0F)+0x30;
       *(ptr++)=0;                                   // fin de string
}
```

La función siguiente se encarga de aprender un ID. Recibe el nombre del usuario junto con el comando y setea al programa principal en el estado de aprendizaje, de modo que la primera RFID que sea leída será ingresada en la base. Todo lo que hacemos es obtener el nombre e informarle al usuario que debe acercarse al lector. No tomamos más recaudos que ignorar el ingreso si la base está llena

```
void learn()
{
unsigned char len;

       if(entries<NUM_ENTRIES){                                    // hay espacio ?
         len=(ser0rxbytes<=NAME_LEN+2)?ser0rxbytes-2:NAME_LEN;     // largo del nombre
         ser0rxbuf[len+1]=0;                                       // fin de string
         strcpy(learnbuf,&ser0rxbuf[2]);                           // copia a buffer
         sstate=LEARN;                                             // cambia estado
         redraw=1;                                                 // actualizar display
       } else {
         // FULL
       }
}
```

La siguiente función es llamada al pedirse el listado de IDs. El loop principal irá llamando a esta función hasta que retornemos con el valor -1, lo cual haremos cuando hayamos terminado de imprimir todas las entradas. Evitamos el uso de funciones como *sprintf()* debido a la excesiva ocupación de RAM y flash:

```
int list(void)
{
static unsigned int printline=0;
unsigned char len;

       if(ser0txbytes)                                             // espera buffer libre
             return(printline);
       if(printline < entries){                                    // para cada ID
         len=bin2bcd(printline+1);                                 // imprime #ID+1
         serbuf[0]=(len>>4)+0x30;
         serbuf[1]=(len&0x0F)+0x30;
         serbuf[2]='\t';                                           // separador
         strcpy(&serbuf[3],rfid_base[printline].name);             // nombre
         len=strlen(serbuf);
         serbuf[len++]='\r';                                       // fin de línea
         serbuf[len++]='\n';
         ser0send(serbuf,len);                                     // envía
         return(++printline);                                      // siguiente
       }
        else {
         printline=0;                                              // terminó, resetea cuenta
          return(-1);                                              // indica finalización
       }
}
```

De igual modo, la siguiente función muestra todos los registros en memoria, basándose en el mismo principio

```
int show(void)
{
static int printline=0;
static unsigned char len;
register int j;
__xdata char *name;

       if(ser0txbytes)
             return(printline);
       if(printline < rcrdcnt){
         j=rcrdidx-rcrdcnt+printline;              // toma primer registro (más antiguo)
         if(j<0)                                   // en un buffer circular
           j+=NUM_RECORDS;
         else if(j>=NUM_RECORDS)
           j-=NUM_RECORDS;
         printdatetime(&time_base[j].time);        // traduce fecha y hora a ASCII en buffer
         strcpy(serbuf,linebuf);                   // copia a buffer de transmisión
         strcat(serbuf,"\t");                      // separador
         name=rfid_base[time_base[j].id].name;     // copia nombre
         strcat(serbuf,name);
         len=strlen(serbuf);
```

```
          serbuf[len++]='\r';                          // fin de línea
          serbuf[len++]='\n';
          ser0send(serbuf,len);
          return(++printline);
        }
        else {
          printline=0;
          return(-1);
        }
}
```

Como nadie es perfecto, agregamos una función que nos permita borrar un ID, compactando la base de IDs moviendo los restantes de forma de ocupar el espacio vacío. Recibimos el número de ID a borrar, el cual el usuario ingresó al observar el listado. Como bien dice el comentario, un reset en medio de esta operación destruiría la base de datos, por lo que si no es posible evitarlo, el interesado deberá re-escribirlo de un modo más seguro. Debido a que el borrar un ID es un procedimiento que se realiza contadas veces, y el desarrollo del esquema de sguridad supera los alcances de esta nota de aplicación, decidimos evitarlo.

```
void delete()
{
unsigned int id,i;

       id=atoi(&ser0rxbuf[2]);                              // obtiene ID (1~entries)
       if((id<=entries)&&entries&&id){
         id--;
         for(i=id+1;i<entries;i++){                         // mueve ocupando el lugar
           strcpy(rfid_base[i-1].name,rfid_base[i].name);
           memcpy(rfid_base[i-1].rfid,rfid_base[i].rfid,5);
         }
         entries--;                                         // uno menos
         updateFRAM2();                                     //actualiza FRAM
       }
//  WARNING NOT POWEROFF-FRIENDLY !!!
}
```

Para inicializar correctamente el sistema, proveemos la opción de borrar la base de IDs y los registros.

```
void purgeid()
{
       entries=0;
       updateFRAM2();
}

void purgedb(void)
{
       rcrdidx=0;
       rcrdcnt=0;
       updateFRAM();
       redraw=1;                    // elimina registros de pantalla
}
```

La siguiente función setea fecha y hora, los datos fueron recibidos junto on el comando

```
void setdatetime()
{
unsigned int month, day, year, hour, minute;


    day = atoi(&ser0rxbuf[2]);                          // extrae de línea de comando
    if (day > 0 && day < 32) {
      month = atoi(&ser0rxbuf[5]);
      if (month > 0 && month < 32) {
        year = atoi(&ser0rxbuf[8]);
        if (year < 100) {
          hour = atoi(&ser0rxbuf[11]);
          if (hour < 24) {
            minute = atoi(&ser0rxbuf[14]);
            if (minute < 60) {
              rtcval.tm_sec  = 0;                       // 0-59
              rtcval.tm_min  = bin2bcd(minute);         // 0-59
              rtcval.tm_hour = bin2bcd(hour);           // 0-23
              rtcval.tm_mday = bin2bcd(day);            // 1-31
```

```
            rtcval.tm_mon  = bin2bcd(month);       // 1-12
            rtcval.tm_year = bin2bcd(year);        // 0-99
            RTCSet(0xD0, &rtcval);                 // actualiza RTC si es coherente
          }
        }
      }
    }
  }
}
```

La siguiente función muestra fecha y hora en la interfaz serie, usando el mismo principio que las rutinas de listados

```
int datetime(void)
{
  unsigned char len;

      if(ser0txbytes)                              // espera buffer libre
            return(0);
      printdatetime(&rtcval);                      // traduce a ASCII en buffer
      strcpy(serbuf,linebuf);                      // copia a buffer de transmisión
      len=strlen(serbuf);
      serbuf[len++]='\r';                          // fin de línea
      serbuf[len++]='\n';
      ser0send(serbuf,len);                        // envía
      return(-1);
}
```

Ahora sí, esta función obtiene el RFID en binario extrayéndolo del paquete ASCII que recibimos del GP8F-R2. Comentada está la versión original, más didáctica, la que se ejecuta es un poco más eficiente sin perder demasiado la claridad:

```
void decode(__xdata unsigned char *rfid)
{
/*int i;
  for(i=0;i<5;i++){
    rfid[2*i+1]-=0x30;
    if(rfid[2*i+1]>=10)
      rfid[2*i+1]-=7;
    rfid[2*i+2]-=0x30;
    if(rfid[2*i+2]>=10)
      rfid[2*i+2]-=7;
    rfid[i]=(rfid[2*i+1]<<4) | rfid[2*i+2];
  }
*/
__xdata unsigned char *ptr;
unsigned char i;

  ptr=rfid+1;
  i=5;
  while(i--){
    *ptr-=0x30;
    if(*ptr>=10)
      *ptr-=7;
    ptr++;
    *ptr-=0x30;
    if(*ptr>=10)
      *ptr-=7;
    *(rfid++)=(*(ptr-1)<<4) | *ptr;
    ptr++;
  }
}
```

La siguiente función busca un ID en la base de IDs, para ver si corresponde a uno conocido. Devuelve el número de ID ó -1 si no lo encuentra

```
int lookup(unsigned char *id)
{
int i;
       for(i=0;i<entries;i++){
       if(!memcmp(id,rfid_base[i].rfid,5))
       return(i);
```

```
   }
   return(-1);
}
```

Para generar el listado de registros en el display utilizamos la siguiente rutina, la cual observa una variable global para decidir si mostrar los 5 últimos registros (más recientes) de forma reducida (hh:mm|nomb), o los últimos 2 en forma ampliada (todos los datos). La selección se efectúa presionando un switch, en nuestro caso el que incluye el kit de desarrollo en el pin P3.2. La lectura del switch la realiza el loop principal.

```
void showonscreen(void)
{
register int record;
register int j;


       record=0;
       if(compact){                                          // modo compacto
         while((record<5)&&(rcrdcnt>record)){                // 5 registros
               j=rcrdidx-record-1;                           // más recientes
               if(j<0)                                       // en un buffer circular
                 j+=NUM_RECORDS;
               printdatetime(&time_base[j].time);            // traduce fechay hora a ASCII
               memcpy(linebuf,linebuf+10,5);                 // extrae hh:mm
               linebuf[5]='|';                               // separador
               memcpy(&linebuf[6],rfid_base[time_base[j].id].name,14); // nombre (14 chars)
               linebuf[20]=0;                                // copia nombre reducido
               LCD_print6at(3+record++,0,linebuf);           // imprime registro, siguiente
         }
       }
        else {                                               // modo expandido
         while((record<2)&&(rcrdcnt>record)){                // 2 registros
               j=rcrdidx-record-1;                           // más recientes
               if(j<0)                                       // en un buffer circular
                 j+=NUM_RECORDS;
               LCD_print8at(3+(record<<1),0,0,rfid_base[time_base[j].id].name); // nombre
               printdatetime(&time_base[j].time);            // fecha y hora
               LCD_print6at(4+(record<<1),1,linebuf);
               record++;                                     // siguiente
         }
       }
}
```

Llegamos finalmente al cuerpo del programa principal. Luego de la inicialización, procesaremos cada una de las diversas tareas dentro de un loop.

```
main()
{
int i;
__xdata unsigned char RFID[15];
__code const static char wrong[]="El RFID leido";
__bit  listing=0,showing=0,datetiming=0,rxing0=0,rxing1=0;

// Software timers, nombres más simples de identificar
#define timer SEC_TIMERS[0]
#define r0timer SEC_TIMERS[1]
#define btimer CS_TIMERS[0]
#define r1timer CS_TIMERS[1]
#define dbnctmr CS_TIMERS[2]

// lectura del switch de modo compacto o expandido
#define SWITCH0 (!P3_2)


  DEVMEMCFG  |= 0xC0;                        // Activa FRAM
  init_softtimers();
  sstate=ESCRACHATE;
  //    bufpos.index=0;                     // para la primera vez, desarrollando
  //    bufpos.count=0;
  //     bufpos.tag=0;
  //     bufpos2.count=0;
  //     bufpos2.tag=0;
  if(bufpos.tag){                            // crash ?
```

```
  rcrdidx=bufpos.bkindex;                  // Recupera de copia de backup en FRAM
  rcrdcnt=bufpos.bkcount;
}
else {
  rcrdidx=bufpos.index;                    // Recupera de copia principal en FRAM
  rcrdcnt=bufpos.count;
}
if(bufpos2.tag){                           // ídem para entries
  entries=bufpos2.bkcount;
}
else {
  entries=bufpos2.count;
}

redraw=1;
compact=1;
LCD_init();
LCD_clear();
LCD_print8at(1,1,'B',"Time logger R2");    // presentación
LCD_print8at(3,3,'B',"v1.0 READY");
LCD_rectangle(4,4,123,44);
LCD_print6at(7,1,"* Cika Electronica");
Init_I2C();
Init_RTC();                                //inicializa RTC
// Demo sin battery/capacitor backup: (ponemos OSCEN=0 en CAL/Control)
//RTCWrite(0xD0,0x01,0x00);
// Ídem, ponemos el RTC en cero (lo que haya en rtcstart)
//RTCSet(0xD0, rtcstart);
ser0init(40000000/9600);                           // serial 0 a 9600
P5PINCFG&=~0x1F;                                   // P5.0,1,2,3,4 output; buzzer/voice
P5=0;
P4PINCFG&=~(1<<4);                                 // P4.4 output (RTS/CTS, no lo usamos)
P3PINCFG|=((1<<3)|(1<<2));                         // P3.2,3 input
ser1init(40000000/9600);                           // serial 1 a 9600 (GP8)
GENINTEN = 0x03;                                   //Enable global interrupts
timer=HOLD_TIME;
while(1){

      if(!btimer)
        P5&=~0x03;                                 // Buzzers/voice off

      if(listing)
        if(list()==-1)                             // llama a list hasta que termine
          listing=0;
      if(showing)
        if(show()==-1)
          showing=0;
      if(datetiming)
        if(datetime()==-1)
          datetiming=0;

      RTCReadAll(0xD0,(unsigned char *)&rtcval);           // Lee RTC
      if(!timer){                                         //  no actualiza si está en HOLD
        if(redraw){                                       // no borra si no hay cambios
              LCD_clear();
              LCD_print8at(0,0,0,"Cika Time Logger");
              LCD_line(0,17,127,17);
              if(sstate==LEARN)                           // mensaje en modo LEARN
                LCD_print8at(7,0,0,"*Acerque el RFID");
              else
                showonscreen();                           // registros en modo ESCRACHATE
              redraw=0;
        }
        printdatetime(&rtcval);                           // fecha y hora en pantalla
        LCD_print6at(1,1,linebuf);
      }
      else redraw=1;                       // actualiza al terminar el período de HOLD

      if(ser0rxbytes&&!rxing0){         // hay algo en el buffer ?
        r0timer=30;                     // inicia timeout
        rxing0=1;
      }
      else if(rxing0&&!r0timer){        // expira timer: paquete incompleto
        ser0flush();                    // descarta
```

```
  rxing0=0;
}
if((ser0peek()==0x0D) && !listing && !showing && !datetiming){ // fin de mensaje
  switch(toupper(ser0rxbuf[0])){            // procesa comandos
  case 'L':
    learn();
    break;
  case 'I':
    listing=1;
    break;
  case 'R':
    showing=1;
    break;
  case 'P':
    switch(toupper(ser0rxbuf[1])){
    case 'R':
      purgedb();
      break;
    case 'I':
      purgeid();
      break;
    }
    break;
  case 'T':
    datetiming=1;
    break;
  case 'D':
    delete();
    break;
  case 'S':
    if(toupper(ser0rxbuf[1])=='T'){
      setdatetime();
      datetiming=1;
    }
    break;
  }
  ser0flush();                              // libera buffer
  rxing0=0;                                 // permite detección de timeout
}

if(SWITCH0){                                // switch presionado ?
  if(!dbnctmr){                             // por primera vez ?
    compact^=1;                             // cambia de modo
    redraw=1;                               // actualiza display
  }
  dbnctmr=6;                                // resetea timer anti-rebote
}

if(ser1rxbytes&&!rxing1){                    // procesa timeout de GP8
  r1timer=30;
  rxing1=1;
}
else if(rxing1&&!r1timer){
  ser1flush();
  rxing1=0;
}
if(ser1rxbytes==14){                        // paquete completo ?
        memcpy(RFID,ser1rxbuf,14);          // salva
       ser1flush();                         // libera buffer
       rxing1=0;                            // permite timeout
       if((RFID[0]==2)&&(RFID[13]==3)){     // comprueba STX y ETX
         decode(RFID);                      // traduce RFID a binario
         LCD_clear();                       // borra LCD
         switch (sstate){
         case ESCRACHATE:                   // modo ESCRACHATE
           if((i=lookup(RFID))!= -1){       // el ID está en la base
             printdatetime(&rtcval);        // rtcval sólo cambia al leer el RTC
             LCD_print6at(2,1,linebuf);     // muestra fecha y hora
             LCD_print8at(4,0,0,rfid_base[i].name); // muestra nombre
             time_base[rcrdidx].id=i;       // guarda fechay hora en registro
             memcpy((void *) &time_base[rcrdidx].time,(void *) &rtcval,
               sizeof(struct RTCtm));
             rcrdcnt++;                     // incrementa la cuenta
             if(++rcrdidx>=NUM_RECORDS)     // en un buffer circular
```

```
                    rcrdidx=0;
                  if(rcrdcnt >= NUM_RECORDS)
                    rcrdcnt=NUM_RECORDS;
                                        // actualiza punteros en FRAM
                  updateFRAM();
                  P5_0=1;               // buzzer/voice
                }
                else {                  // el ID NO está en la base
                  LCD_print6at(2,3,wrong);
                  LCD_print6at(4,0,"no figura en la base");
                  P5_1=1;               // beep/voice
                }
                btimer=BEEP_TIME;       // timer para beep/voice
                break;
              case LEARN:               // modo LEARN
                if((i=lookup(RFID))!= -1){  // el ID está en la base
                  LCD_print6at(2,3,wrong);
                  LCD_print6at(4,0,"ya figura en la base");
                }
                else {                  // el ID es nuevo
                  LCD_print6at(0,0,"Ingresado como:");
                  strcpy(rfid_base[entries].name,learnbuf);
                  memcpy(rfid_base[entries].rfid,RFID,5);       // muestra nombre
                  LCD_print8at(2,0,0,rfid_base[entries++].name);
                                        // actualiza IDs en FRAM
                  updateFRAM2();
                }
                sstate=ESCRACHATE;      // sale de modo LEARN
                break;
              default:
                break;
              }
              timer=HOLD_TIME;          // persistencia del display
            }
        }
  }
}
```

Para el manejo de las interfaces serie hemos implementado un pequeño esquema de seguridad. Para la lectura del GP8 esperamos a recibir un paquete completo, lo cual sucederá cuando reciba 14 caracteres, que almacenará en RFID; o hayan transcurrido 300ms sin que reciba ningún caracter luego de empezada la recepción de un paquete. Para la recepción de un comando, permitimos una demora de hasta 30 segundos desde el primer caracter en el buffer hasta que recibimos el retorno de carro que indica la finalización del comando.

El esquema de seguridad para sobrevivir a resets en medio de la actualización de variables críticas corresponde a mantener una copia de seguridad de las variables y detectar mediante un tag si la copia principal es válida. Es decir, actualizamos la copia de seguridad, invalidamos la copia principal, actualizamos al copia principal, y finalmente la validamos. En el caso del índice de la base y la cuenta de registros es obvio, porque ambas variables están fuertemente relacionadas y deben guardar consistencia; en el caso de la cuenta de IDs, al ser un entero, es una variable multi-byte, y el core MCS-51 no dispone de instrucciones para manejo atómico de estas variables, por lo que debemos protegernos también.

Compilación

El proyecto se basa en las notas de aplicación anteriormente mencionadas, por lo que el interesado en compilarlo deberá obtener dichas notas (disponibles de forma gratuita). Como se podrá observar en el listado completo del programa en el archivo adjunto, los *#includes* hacen referencia a directorios dentro de un filesystem Linux, en el cual cada nota de aplicación tiene su directorio. Las modificaciones para compilarlo en otro entorno son realmente simples.