Cika Electrónica

Nota de Aplicación: CAN-092
Título: **Cartel matriz de LEDs remoto con HT1632 y XBee Programable**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	18/02/11	

En esta nota de aplicación desarrollaremos un cartel luminoso remoto. El mismo emplea una matriz de LEDs controlada por un HT1632, el cual, a su vez, es controlado por un XBee Programable. Los textos son enviados vía ZigBee a la dirección del cartel, quien los muestra en los LEDs.

El HT1632

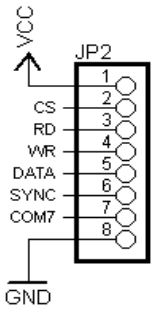
La descripción del HT1632 y un ejemplo de operación pueden obtenerse en el Comentario Técnico CTC-065.

El XBee Programable

La forma de utilización de este módulo se describe en el tutorial CTU-012.

El hardware

Para la conexión con el módulo utilizaremos un conector:



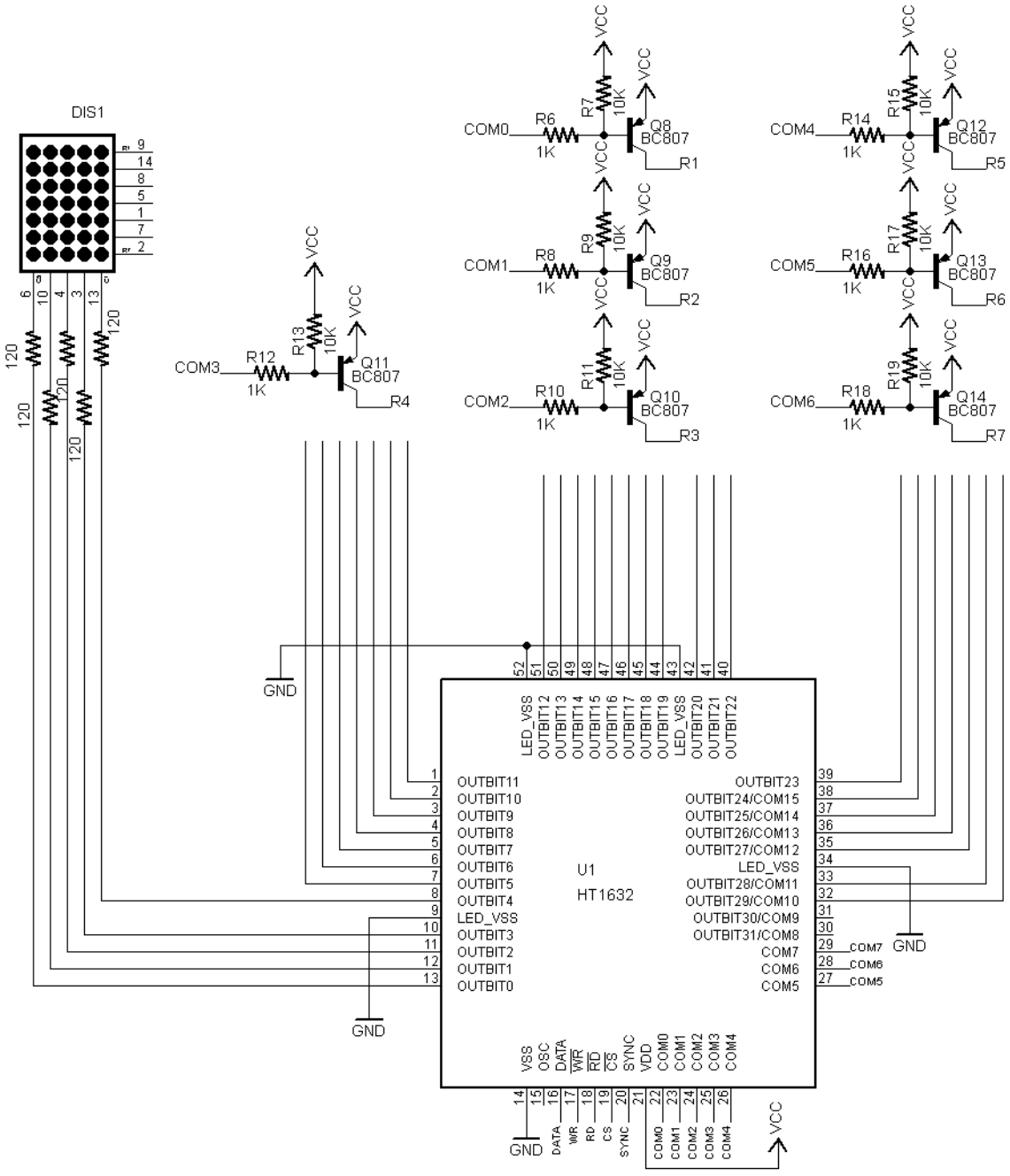
Si bien se trata de una simple aplicación del circuito sugerido en la hoja de datos del chip, requiere su análisis correspondiente. Por ejemplo, para un cartel de 5 caracteres de 7x5 tendremos 7 filas de 5x5=25 LEDs cada una.

La corriente de cada LED mientras está encendido, la controlamos mediante los resistores en serie con los pines OUTBIT. Por ejemplo, con 120Ω tenemos $I_{LED}=\frac{V_{cc}-V_{LED}}{R}=\frac{3\text{V}-1{,}8\,V}{120\,\Omega}=10\,mA$

Dependiendo de la cantidad de LEDs, tendremos la corriente que circulará por los transistores controlados por los COMs. La corriente máxima por fila, para el peor caso de tener todos los LEDs encendidos, es de $I_{PNP}=25\times I_{LED}=250\text{mA}$.

El ciclo de trabajo de cada fila dependerá de la cantidad de COMs configurada en el modo de funcionamiento del chip (8 ó 16). En este ejemplo, el chip funcionará en modo 32x8, por lo que cada fila tiene un ciclo de trabajo de 1/8. la corriente media por fila, para el peor caso de tener todos los LEDs encendidos, es de $\overline{I}=\frac{1}{8}\times I_{PNP}$ y la corriente media por cada LED encendido es de $\overline{I}=\frac{1}{8}\times I_{LED}=1{,}25\,mA$

El siguiente esquemático muestra el hardware utilizado:



Los pines empleados pueden verse en la siguiente tabla, y han sido elegidos de modo de poder conectarse fácilmente al módulo utilizando los pines de la interfaz serie de la placa XBoard[1].

1 La placa XBoard se incluye en los kits de Cika y está disponible por separado.

función	*pin XBee*
$\overline{CS}$	IO_DIO8_DTR
$\overline{RD}$	IO_DIO12_CD
$\overline{WR}$	IO_DIO6_RTS_HOST
DATA	IO_DIO7_CTS_HOST

El software

Conectado el hardware, podemos bajar el código de la aplicación mediante XMODEM seleccionando la opción 'F' en el menú del bootloader. Dicho código corresponde al archivo *Program.abs.bin* del archivo adjunto.

Respecto al Comentario Técnico CTC-065, las modificaciones necesarias son las macros requeridas para nuestro hardware, y la macro para realizar un nibble swap:

```
#define CS_LO IO_DIO8_DTR=0
#define CS_HI  IO_DIO8_DTR=1
#define RD_LO  IO_DIO12_CD=0
#define RD_HI  IO_DIO12_CD=1
#define WR_LO  IO_DIO6_RTS_HOST=0
#define WR_HI  IO_DIO6_RTS_HOST=1
#define DATA   IO_DIO7_CTS_HOST
#define DATA_LO IO_DIO7_CTS_HOST=0
#define DATA_HI IO_DIO7_CTS_HOST=1
#define DATA_ISOUT IO_DIO7_CTS_HOST_D=1
#define DATA_ISIN IO_DIO7_CTS_HOST_D=0
#define CTRLC IO_DIO7_CTS_HOST_PE=1;IO_DIO8_DTR_D=1;IO_DIO12_CD_D=1;IO_DIO6_RTS_HOST_D=1;\
               CS_HI;RD_HI;WR_HI    // control lines high

#define SWAP(a) __asm LDA a ;__asm NSA ;__asm STA a
```

Los detalles de control del procesador e interacción con el bootloader, se explican en el tutorial CTU-012. El listado a continuación hace uso de esos conceptos.

```
#include "pin_mapping.h"      // nombre de los pines del micro mapeados al pinout del XBee
#include "common.h"           // definiciones del bootloader
#include "sharedRAM.h"        // variables compartidas con el bootloader

// nombre de la aplicación
#pragma CONST_SEG APPLICATION_VERSION
static const uint8 version[] = "cartelBee 1.0 alfa 0";

// puntero al nombre de la aplicación en una posición fija y conocida por el bootloader
#pragma CONST_SEG DEFAULT
static const unsigned long pAppVersion @0x0000F1BC = (unsigned long)version;

// diferencias de interpretación de la especificación de C99
typedef unsigned char uint8_t;

// inicialización del micro
#include "MCUinit.h"

// todo lo referido al HT1632
#include "HT1632.h"

// para devolver el control al bootloader
void ResetHardware(APP_RESET_CAUSES Reset_Val){

    DisableInterrupts;
    AppResetCause = Reset_Val;
    if (AppResetCause == APP_CAUSE_REQUEST_SOPT1_NOT_WRITTEN){
      BLResetCause = BL_CAUSE_NOTHING;
    }
    for(;;);           //Wait for WDTimeout
```

```
}
```

El modo API y su software se desarrollan en la Nota de Aplicación CAN-089. El siguiente listado es el código del programa principal, que lee un mensaje y contesta con un prompt. El listado completo se encuentra en el archivo adjunto.

```
#include "apiframe.h"
#include "apicommon.h"
#include "apizb.h"

#define BOOTLOADER_PASSWORD "@#$%~"

void main(void)
{
static APIstruct api,apirx;
static APIdata rxdata;
APIinfo *info;
APIzsend *zs;
APIzrecv *zr;
int ret,tx=0;

        MCU_init();                         // inicialización del micro
        init_HT1632();                      // inicialización del HT1632
        HT1632_cls();                       // borrado de pantalla
        API_initgettingframe(&apirx);       // inicialización de receptor API
        HT1632_string(0,"Hola!");           // envío de mensaje inicial
        IO_DIO4_ADC4_D=0;                   // pin en dip-switch como entrada

        for(;;){
                __RESET_WATCHDOG();         // kick watchdog
                if(!IO_DIO4_ADC4)           // si se selecciona, vuelve al bootloader
                        ResetHardware(APP_CAUSE_BOOTLOADER_MENU);//never returns

                if((ret=API_getframe(&apirx,0))>0){                 // si recibió un mensaje
                        rxdata.len=(unsigned char) ret;             // recuerda longitud
                        memcpy(&rxdata.info,&apirx.frame.info,ret); // copia
                        API_initgettingframe(&apirx);               // ya puede recibir otro
                        info=&rxdata.info;                          // apunta a parte útil
                        switch(info->id){                           // tipo de mensaje
                        case API_APIZRECV:                          // datos
                                zr=(APIzrecv *)&info->data;         // apunta a contenido

/* si recibe un texto en particular, escapa al bootloader (el módulo que llama queda en
comunicación con el bootloader) */
                                if(!memcmp(&zr->data,BOOTLOADER_PASSWORD,5))
                                        ResetHardware(APP_CAUSE_BOOTLOADER_MENU);//never returns

                                (&zr->data)[5]=0;                   // fin de texto
                                HT1632_string(0,&zr->data);         // muestra en display
                                zs=(APIzsend *)&api.frame.info.data; // inicializa respuesta
                                memcpy(zs->addr64,zr->addr64,8);    // copia direcciones
                                zs->addr16[0]=zr->addr16[0];
                                zs->addr16[1]=zr->addr16[1];
                                zs->radius=0;                       // radio
                                zs->options=0;                      // opciones
                                memcpy(&zs->data,"\r\n> ",4);       // prompt
                                api.frame.info.id=API_APIZSEND;     // trama de datos
                                API_buildframe(&api,4+sizeof(APIzsend)); // arma trama
                                API_initsendingframe(&api);         // inicializa envío
                                tx=1;                               // habilita proceso
                                break;
                        default:
                                break;
                        }
                }
                if(tx){                                     // si está el proceso habilitado
                        if(API_sendframe(&api,0)==API_DONE) // llama a la función de envío
                                tx=0;                       // listo
                }
        }
}
```

Misión cumplida