Cika Electrónica

Nota de Aplicación: CAN-016
Título: **Display PG320240FRST (Touch Screen) con Dynamic C 8**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	27/10/03	

Portamos las bibliotecas de funciones con soporte para displays gráficos y pantallas sensibles al tacto o touch screens, que incluye Dynamic C en su versión 8, para su utilización con el display PG320240FRST de Powertip, de modo de aprovechar la mayor cantidad posible de este código para acortar nuestros tiempos de desarrollo. Aprovechamos el análisis de estas bibliotecas de funciones desarrollado en las notas de aplicación CAN-013 y CAN-014, el hardware para conexión del display en la CAN-012, y el hardware de lectura de touch screen desarrollado en la CAN-015; desarrollando nuestra propia biblioteca de funciones: *Cika320240FRST.lib*, donde alojaremos las funciones que portamos.

Display

SED1335F.LIB

Vimos en la CAN-013 que esta biblioteca contiene las funciones de bajo nivel para el controlador de displays LCD SED1335. Esta biblioteca fue escrita para un producto de Z-World, el OP7200, cuyo circuito eléctrico se provee con la documentación de Dynamic C versión 8 y versiones Premier anteriores. Analizando su funcionamiento, vemos que utiliza un display no-inteligente, con un controlador externo, pero el esquema es básicamente igual al que desarrolláramos en la CAN-012, por lo que podemos aprovechar ese desarrollo como hardware de base. Las diferencias fundamentales son:

- ✔ No controlaremos la operación del backlight, queda a gusto del usuario este desarrollo
- ✔ El control del contraste de nuestro display se realiza mediante un preset en su parte posterior

Con estas salvedades, podemos perfectamente utilizar esta biblioteca de funciones sin modificaciones. No obstante, deberá tenerse en cuenta que cualquier llamada a las funciones *glSetContrast()* y *glBackLight()* operará sobre un hardware que no incluímos, y no debería superponerse a otra parte de nuestro desarrollo final. Como ésto opera sobre espacio de I/O externo, no deberemos utilizar esos espacios. Consulte el fuente de *SED1335F.LIB* para mayor información.

Touch Screen

TS_R4096.LIB

Vimos en la CAN-014 que esta biblioteca contiene las funciones de bajo nivel para el controlador de touch screens ADS7843. En la CAN-015 desarrollamos un hardware para leer la touch screen de este display. Esta biblioteca también fue desarrollada para el OP7200, y si bien el hardware es diferente, la usaremos como esqueleto para desarrollar el soporte para nuestro hardware.

Como vimos en la CAN-014, debemos portar las funciones de más bajo nivel:

- ➔ *_adcTouchScreen()*: es la que realiza la lectura del chip controlador de la touchscreen
- ➔ *TsActive()*: es la que monitorea la presión sobre la touchscreen.

Las demás funciones podemos utilizarlas tal cual son, o por supuesto adecuarlas a nuestro criterio, como más nos agrade.

Desarrollo de drivers

_adcTouchScreen()

Utilizaremos el conocimiento adquirido en la CAN-010 al desarrollar la conexión del MCP3204 mediante una interfaz SPI. Si bien no corresponde a esta función, deberemos tener en cuenta a la hora de desarrollar la rutina de inicialización, que estamos utilizando la interfaz SPI e inicializar el hardware.

Esta función es la encargada de leer el chip controlador, devolviendo un valor entre 0 y 4096, dado que el ADS7843 es un conversor de 12-bits con un multiplexor interno para conmutar la polarización y medición de las membranas. Por lo tanto, para evitar modificar las demás rutinas, "interceptaremos" los comandos del ADS7843 y obraremos sobre nuestro hardware, retornando los datos en idéntico formato. En vez de armar por software el comando a enviar al MCP3204, utilizaremos comandos pre-definidos.

```
int _adcTouchScreen( int cmd )
{
/*
       START | SINGLE | ((Channel/4)<<5) | ((Channel/2)<<4) | ((Channel&1)<<3);
       Command <<= 3;                        // posición para obtener LSB justif a la derecha
       Command = SwapBytes ( Command );      // pone el MSB primero (Z80 es LSB primero)
   X: CH0: C0<<3 -> 0600 -> 0006 (GETX)
   Y: CH1: C8<<3 -> 0640 -> 4006 (GETY)
*/

int Command, j;
struct                          // 24 bits
{      char    b;
       int     i;
} Data;

       switch(cmd){
          case 0xD0:
          default:
               BitWrPortI ( PEDR, &PEDRShadow, 1, 1 );      // B=1, lado izquierdo a masa
               BitWrPortI ( PEDR, &PEDRShadow, 0, 0 );      // A=0, lado derecho a positivo
               Command = GETX;
               break;
          case 0x90:
               BitWrPortI ( PEDR, &PEDRShadow, 1, 0 );      // A=1, arriba a masa
               BitWrPortI ( PEDR, &PEDRShadow, 0, 1 );      // B=0, abajo a positivo
               Command = GETY;
               break;
       }
#asm
       WAIT_50usX(40);                                      // Estabiliza conmutación
#endasm
       BitWrPortI ( PDDR, &PDDRShadow, 0, 3 );              // habilita /CS
       SPIWrRd ( &Command, &Data, 3 );                      // el 3er byte no interesa
       BitWrPortI ( PDDR, &PDDRShadow, 1, 3 );              // remueve /CS (1=off)
       j = Data.i;
       j = SwapBytes ( j ) & 0x0FFF;                        // LSB primero, 12 bits

       BitWrPortI ( PEDR, &PEDRShadow, 1, 0 );              // remueve polarización
       BitWrPortI ( PEDR, &PEDRShadow, 1, 1 );
       return(j);
}

// invierte (swap) los bytes de un entero
// parámetro y resultado en HL
#asm
SwapBytes::
       ld a, L        ; salva LSB
       ld L, H        ; MSB -> LSB
       ld H, A        ; recupera LSB en MSB
       ret
#endasm
```

La macro *WAIT_50usX()* que utilizamos pertenece al cuerpo de la biblioteca de funciones, dado que la usaremos más adelante en otra función.En nuestro desarrollo utilizamos un loop, el usuario puede utilizar lo que más le convenga.

TsActive()

Esta función es la encargada de detectar la presencia de presión sobre la pantalla. Con un chip controlador como el ADS7843, esto es una tarea fácil, dado que dispone de un pin destinado específicamente a esta tarea. Con nuestro hardware, haremos una lectura de una de las coordenadas y compararemos el valor devuelto con los valores que observamos de antemano como normales. Estos valores son siempre menores que los que se obtienen al presionar sobre el extremo izquierdo superior de la pantalla, pero su valor absoluto dependerá de cada hardware, por lo que el usuario deberá trabajar sobre este aspecto para su aplicación en particular.

```
int TsActive( void )
{
int Command,j;
struct                          // 24 bits
{       char    b;
        int     i;
} Data;

                BitWrPortI ( PEDR, &PEDRShadow, 1, 1 );      // B=1, lado izquierdo a masa
                BitWrPortI ( PEDR, &PEDRShadow, 0, 0 );      // A=0, lado derecho a positivo
        #asm
        WAIT_50usX(40);                                      // Estabiliza conmutación
        #endasm
        Command = GETX;
        BitWrPortI ( PDDR, &PDDRShadow, 0, 3 );              // habilita /CS
        SPIWrRd ( &Command, &Data, 3 );                      // el 3er byte no interesa
        BitWrPortI ( PDDR, &PDDRShadow, 1, 3 );              // remueve /CS (1=off)
        j = Data.i;
        j = SwapBytes ( j ) & 0x0FFF;                        // LSB primero, 12 bits

        BitWrPortI ( PEDR, &PEDRShadow, 1, 0 );              // remueve polarización
        BitWrPortI ( PEDR, &PEDRShadow, 1, 1 );
        return(j>180);                                       // si hay presión devuelve 1
}
```

Como tenemos pensado hacer crecer esta biblioteca en el futuro, agregando diversas opciones de hardware de lectura de touch screen, y preservar el código actual en caso de incorporar el ADS7843 a nuestro portfolio de productos, utilizaremos ensamblado condicional en varias partes del código. No lo incluímos en el ejemplo por claridad.

Inicialización

Debemos escribir una función que inicialice los diversos componentes. Tomamos como ejemplo la inicialización del OP7200 en OP7200.LIB y la modificamos para servir nuestros propósitos.

```
void brdInit (void)
{
        auto int cmd;

        #GLOBAL_INIT {__brdInitFlag   = FALSE;}
        #GLOBAL_INIT
        {
//              __keyInitFlag   = FALSE;      // para inicialización de teclado
                loopsPer_50us = (int)(19200L*32*freq_divider/1000000L)+1;   // para demoras
        }

        // PD3= salida, activo (sin open drain), fuerza a 1
        WrPortI(PDDDR, &PDDDRShadow,(PDDDRShadow | 0x8));
        WrPortI(PDDCR, &PDDCRShadow,(PDDCRShadow & ~0x8));
```

```
        WrPortI(PDDR, &PDDRShadow, (PDDRShadow | 0x8));

        // Inicializa PE0 y PE1 como salidas comunes
        WrPortI(PEDR, &PEDRShadow, (PEDRShadow  |  0x03));
        WrPortI(PEFR, &PEFRShadow, (PEFRShadow  & ~0x03));

        WrPortI(PEDDR, &PEDDRShadow, (PEDDRShadow | 0x03));
        WrPortI(PEDR, &PEDRShadow,   (PEDRShadow  | 0x03));

#if TSCONTROLLER == 1
        SPIinit();            // inicializa interfaz SPI
#endif

        // Inicializa PE7 como chip select
        WrPortI(PEFR, &PEFRShadow,   (PEFRShadow  | 0x80));
        // salida
        WrPortI(PEDDR, &PEDDRShadow, (PEDDRShadow | 0x80));

        // chip select con 3 wait-states, writes permitidos
        WrPortI(IB7CR, &IB7CRShadow,  0x88);

        // Ports D y E clockeados por PCLK/2
        WrPortI(PDCR, &PDCRShadow, 0x00);
        WrPortI(PECR, &PECRShadow, 0x00);

        __brdInitFlag = TRUE;
        __InitTSpower  = 0;
                // Inicializa la lectura de TouchScreen
                TsCalibEERd();
                _adcTouchScreen(0xD0);
                _adcTouchScreen(0x90);
        __anaPWRDelay = MS_TIMER + 250;
        __InitTSpower  = 1;
}
```

Muchas de las variables son globales, y se definen en el cuerpo de la biblioteca de funciones. Las incluímos para modificar lo menos posible las funciones existentes.

Calibración

A fin de poder seguir re-utilizando la mayor cantidad posible de código y ejemplos, conservamos el esquema de *TS_R4096.LIB* para guardar y recuperar las constantes de calibración en el user block, descripto en la CAN-014.

El esquema de calibración utiliza la teoría desarrollada en la CAN-015, con las constantes en una estructura:

```
typedef struct
{
        int x_offset;
        int y_offset;
        float x_gainfactor;
        float y_gainfactor;
} tc_cal;
```

1- Obtención de las constantes de calibración asociando dos puntos en pantalla del display $(X_1;Y_1)$ y $(X_2;Y_2)$, con las coordenadas devueltas por el sistema de touch screen $(x_1;y_1)$ y $(x_2;y_2)$, como se observa en *TsCalib()*:

```
        _adcCalibTS.x_offset = x1;
        _adcCalibTS.y_offset = y1;

        _adcCalibTS.x_gainfactor = (float)LCD_XS/(float)(x2-x1);
        _adcCalibTS.y_gainfactor = (float)LCD_YS/(float)(y2-y1);
```

2- Corrección de los datos al momento de usarlos, como se observa en TsXYvector():

```
*xkey -= _adcCalibTS.x_offset;
*ykey -= _adcCalibTS.y_offset;

*xkey =  (int) (*xkey*_adcCalibTS.x_gainfactor);
*ykey =  (int) (*ykey*_adcCalibTS.y_gainfactor);
```

La calibración de la touch screen se puede realizar con la utilidad *CAL_TOUCHSCREEN.C*, que viene incluída en la distribución de Dynamic C versión 8, en el directorio:
<RABBIT_PATH>\Samples\OP7200\LCD_TouchScreen.

Una vez calibrada, cada vez que carguemos un nuevo ejemplo, la rutina de inicialización recuperará las constantes del user block, por lo que la pantalla seguirá funcionando. Si bien la calibración depende de las condiciones de temperatura, es lo suficientemente estable para aplicaciones de selección y para los alcances de esta nota de aplicación.

Si recibimos algún error respecto a la existencia o estructura del user block, podemos generarlo mediante la utilidad *CHANGE_USERBLOCK.C*, que se adjunta.

Demás funciones

Las demás funciones las tomamos, practicamente sin modificaciones de *TS_R4096.LIB*. Puede consultar el fuente para más detalles.

Biblioteca de funciones

El armado de una biblioteca de funciones responde a los lineamientos de Dynamic C, puede consultar, si así lo desea, un tutorial sobre Dynamic C (CTU-001) que explica ésto con un poco más de detalle. Sintetizando, se trata de generar encabezados (headers) que son detectados por Dynamic C y sirven para reconocer las funciones y su formato, generar el sistema de ayuda (help), e incluír apropiadamente las variables y definiciones necesarias. Agregamos además la inclusión de todas las demás bibliotecas de funciones y soporte de tipografías conocidas, de modo que el usuario solamente deba incluír esta biblioteca de funciones:

```
/* START LIBRARY DESCRIPTION *********************************************
Cika320240FRST.LIB 1.0

Developed by Cika Electrónica S.R.L, portions of the code are
based on samples included in Dynamic C distribution, and so are
Copyright (c) 2002, Z-World

DESCRIPTION:   Support for Powertip PG320240FRS with touch screen

SUPPORT LIBS:

END DESCRIPTION **********************************************************/


/*** BeginHeader */
#ifndef __Cika320240FRSTLIB
       #define __Cika320240FRSTLIB
       #use "sed1335F.lib"
       #use "graphic.lib"
       #use "6x8l.lib"
       #use "8x10l.lib"
       #use "10x16l.lib"
       #use "12x16l.lib"
       #use "17x35l.lib"
       #use "TERMINAL9.LIB"
       #use "TERMINAL12.LIB"
       #use "TERMINAL14.LIB"
       #use "glmenu.lib"
       #use "glTouchScreen.lib"
```

```
	#if TSCONTROLLER == 1
		#define SPI_SER_B
		#define SPI_CLK_DIVISOR 5
		#use SPI.LIB
		#define GETX 0x0006
		#define GETY 0x4006
	#endif
/*** EndHeader */

< cuerpo de la biblioteca de funciones >

/*** BeginHeader */
#endif
/*** EndHeader */
```

Para incluír esta biblioteca de funciones, debemos poner en nuestro programa:

```
#define TSCONTROLLER  1
#use "Cika320240FRST.lib"
```

La primera linea indica que estamos utilizando la implementación actual (esto es por compatibilidad futura); la segunda incluye la biblioteca, la cual, a su vez, incluye todas las otras soportadas.

La totalidad del código de la biblioteca puede observarse en archivo adjunto.

Para que Dynamic C reconozca esta biblioteca de funciones, deberemos agregarla en *LIB.DIR* (sito en el directorio de instalación de Dynamic C), o bien proveer nuestro propio archivo y configurarlo en las opciones de proyecto (Options -> Project Opctions -> Compiler -> Advanced).

Ejemplos

Si bien la mayoría de los ejemplos que vienen para el OP7200 funcionan con menores modificaciones, hemos incluído uno mostrando gran parte de las posibilidades gráficas. Para que este y otros ejemplos se ejecuten correctamente, deberemos primero calibrar la touchscreen, y guardar las constantes en el user block.

```
// Usamos el hardware de lectura de la touch screen basado en MCP3204
#define TSCONTROLLER  1
/* Esto incluye la biblioteca de funciones de Cika que soporta el display de 320x240 de
Powertip */
#use "Cika320240FRST.lib"

#memmap xmem

// íconos
#use "ledon_bmp.lib"
#use "ledoff_bmp.lib"
#use "button_bmp.lib"

int led1,led2;
fontInfo fi8x10,fi10x12,fi6x8;

void led1show(int state)
{
	if (state)
		glXPutBitmap(8, 8, 90,90, ledon_bmp);
	else
		glXPutBitmap(8, 8, 90,90, ledoff_bmp);
}

void led2show(int state)
{
	if (state)
		glXPutBitmap(208, 8, 90,90, ledon_bmp);
	else
		glXPutBitmap(208, 8, 90,90, ledoff_bmp);
}
```

```
unsigned  long userX;         // Área de botones

void main()
{
int btn;

        brdInit();            // Inicializa hardware
        glInit();             // Inicializa biblioteca de funciones gráficas

        glXFontInit ( &fi6x8,6,8,0x20,0x7E,Font6x8 );       // Inicializa tipografía

        glBlankScreen();                                    // borra pantalla
        userX = btnInit( 3 );
        btnCreateBitmap(userX,1,0,110,0,1,button_bmp,90,90);        // define botones
        btnCreateBitmap(userX,2,200,110,0,1,button_bmp,90,90);
        btnAttributes(userX,1,0,0,0,0);
        btnAttributes(userX,2,0,0,0,0);

        led1=1;
        led2=0;

        while (!btnDisplayLevel(userX,1)); // Muestra todos los botones de nivel 1
        led1show(led1==1);
        led2show(led2==1);
        glPrintf(4,230,&fi6x8,"Demo Cika320240FRST.lib, Cika Electronica S.R.L.");

           while (1) {

                costate {                     // waitfor debe ir dentro de un costate
                       waitfor ( ( btn = btnGet(userX) ) >= 0 );
                       switch (btn){
                              case 1:
                                 led1^=1;
                                 led1show(led1);
                                      break;
                              case 2:
                                 led2^=1;
                                 led2show(led2);
                                      break;
                       }
                }
           }
}
```

Las bibliotecas *led_on.lib*, *led_off.lib* y *button.lib* fueron generadas con la utilidad fbmcnvtr, provista con Dynamic C. Convierte fonts y bitmaps a formato library. Lo mismo que con *Cika320240FRST.lib*, deben estar listadas en el archivo índice de biblioteas de funciones (*LIB.DIR* o equivalente)

Para que algunos ejemplos compilen correctamente, deberemos proveer una función buzzer(), aunque no es necesario que realice alguna función en particular. Esto se debe a que las funciones de la biblioteca GLTOUCHSCREEN.LIB incluyen soporte para “keyclick”.

```
int buzzer()
{
}
```