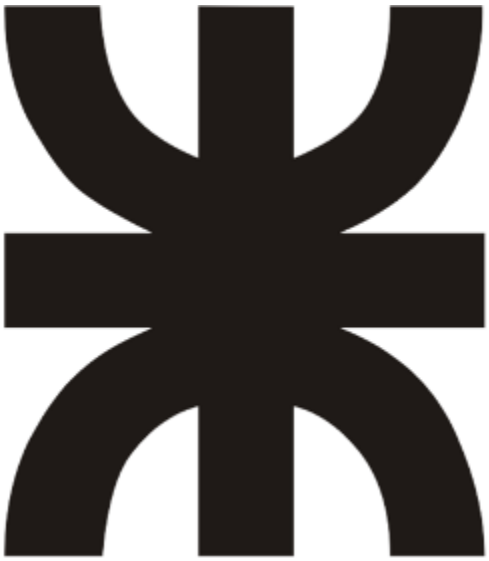
Universidad Tecnológica Nacional

Facultad Regional Tucumán



Técnicas Digitales II 2022

Mostrar información en pantalla TFT usando el protocolo SPI en PSoC6

Implementación de un cronómetro simple en kit de desarrollo PSoC6

Integrantes:

- Arroyo, Micaela
- Ramos, Adrián David

Fecha: 22 de febrero de 2023

Universidad Tecnológica Nacional
Facultad Regional de Tucumán

Introducción

Este informe presenta una guía sobre cómo utilizar la interfaz SPI para mostrar información en una pantalla TFT utilizando la placa de desarrollo PSoC6 de Infineon.
El objetivo de este informe es demostrar cómo utilizar esta interfaz y la pantalla TFT para la presentación de información.

En este informe se presentará un proyecto práctico para la construcción de un pequeño cronómetro. Aunque el cronómetro es un proyecto sencillo, el objetivo principal es demostrar cómo se pueden utilizar los módulos de la placa PSoC6, tales como el módulo SPI en conjunto con un display TFT, el módulo PWM para activar una bocina y el módulo de interrupciones para el control del cronómetro.

La interfaz Serial Peripheral Interface (SPI) es un protocolo de comunicación síncrono utilizado para la transferencia de datos entre dispositivos electrónicos. La pantalla TFT (Thin Film Transistor) es una tecnología de pantalla de alta calidad que se utiliza en dispositivos electrónicos modernos, como teléfonos inteligentes, tabletas y dispositivos de navegación GPS.

Objetivo

Al final de este informe, los lectores tendrán un conocimiento completo sobre cómo utilizar la interfaz SPI y la pantalla TFT para la presentación de información en la placa PSoC6 de Infineon. Los lectores también estarán en capacidad de aplicar estos conocimientos para desarrollar sus propios proyectos y soluciones personalizadas.

Conocimiento previo requerido

- Creación de proyectos en el entorno ModusToolbox
- Utilización de la librería HAL para inicializar puertos GPIO
- Manejo básico del módulo PWM
- Manejo de interrupciones mediante el registro de un evento y una función callback

En caso de tener alguna duda respecto a los puntos anteriormente mencionados lo invitamos a repasar los primeros dos niveles del manual de entrenamiento para el uso de ModusToolbox proporcionado por Infineon en su repositorio de github.
Enlace: https://github.com/Infineon/training-modustoolbox

Componentes utilizados

- Kit de desarrollo de prototipos PSoC™ 6 Wi-Fi BT (CY8CPROTO-062-4343W)
- Display TFT 0.96” 80X160 RGB IPS LCD
    - Driver: St7735
    - Protocolo de comunicación: SPI
- Bocina tipo Buzzer/Beeper
- Protoboard y cables para la conexión

Software de desarrollo

- Entorno de Desarrollo ModusToolbox de Infineon
- SEGGER emWin Middleware Library

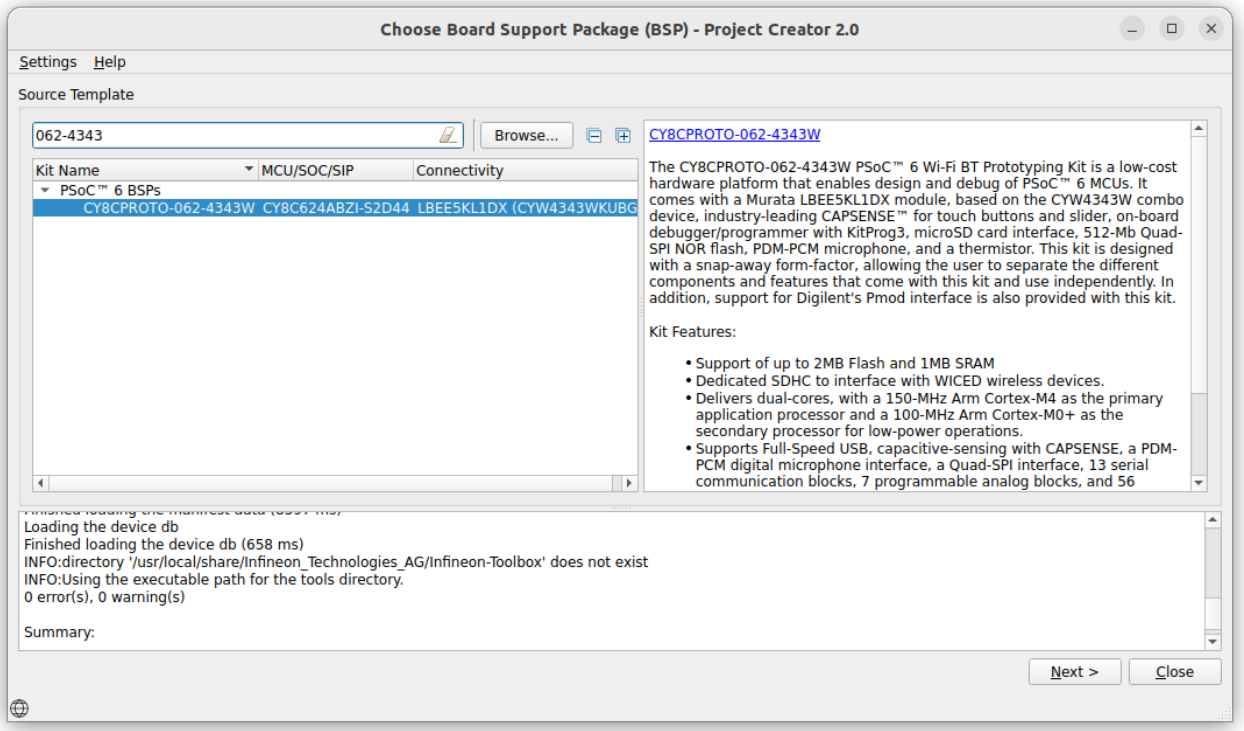
Conexión de la pantalla TFT

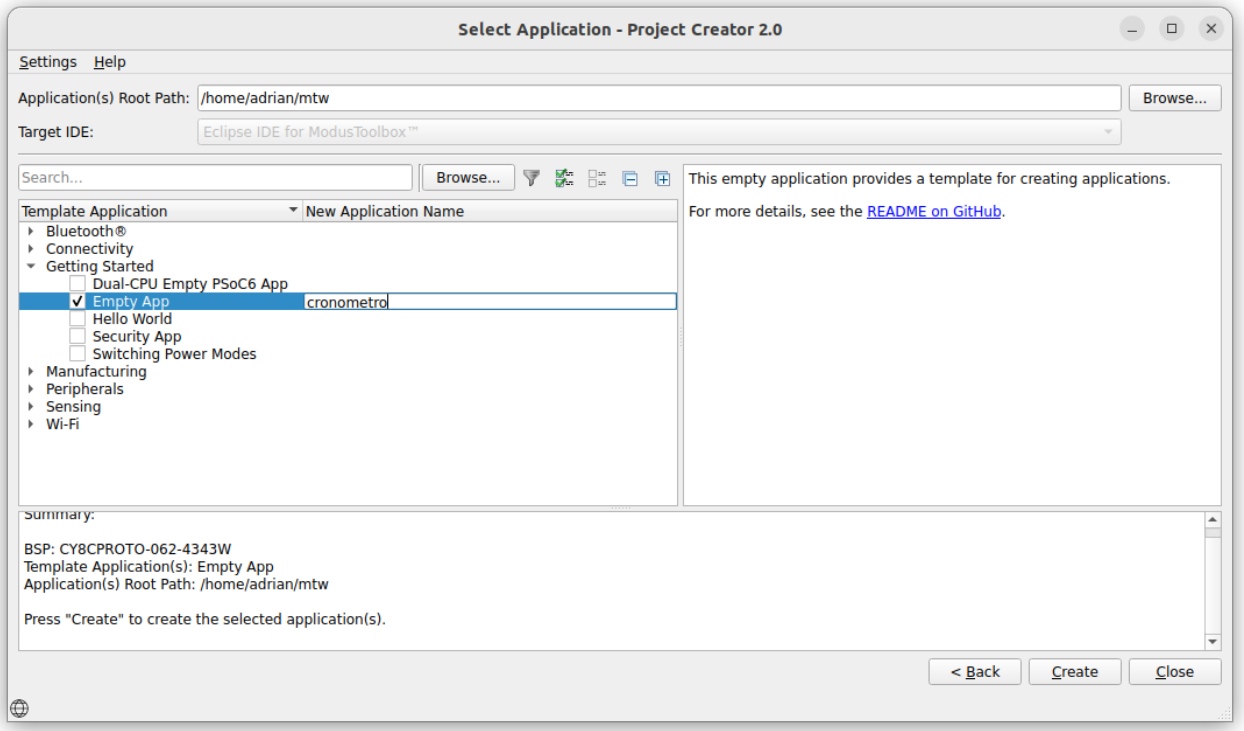
Como se mencionó anteriormente la pantalla a utilizar trabaja con el driver St7735 y el protocolo de comunicación SPI.

Los pines disponibles en este componente son:

- **GND**: Conexión a tierra
- **VCC**: Conexión a tensión positiva (3.3V a 5V)
- **SCL**: Entrada de reloj SPI
- **SDA**: Entrada de datos (MOSI)
- **RES**: Reset
- **DC**: Entrada Data/Command, para indicar si estamos ingresando un dato o un comando
- **BLK**: Backlight, controla el brillo de la pantalla

Para realizar el conexión primero creamos un nuevo proyecto en ModusToolbox, seleccionando el Paquete de Soporte adecuado para nuestra placa (CY8CPROTO-062-4343W) y una plantilla vacía.



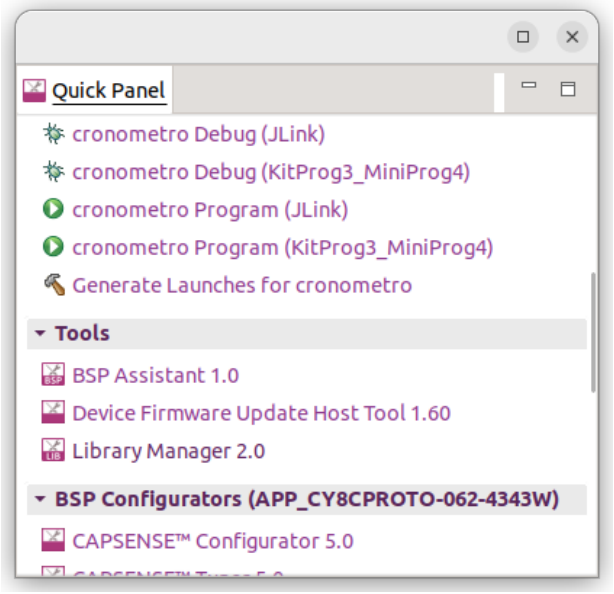


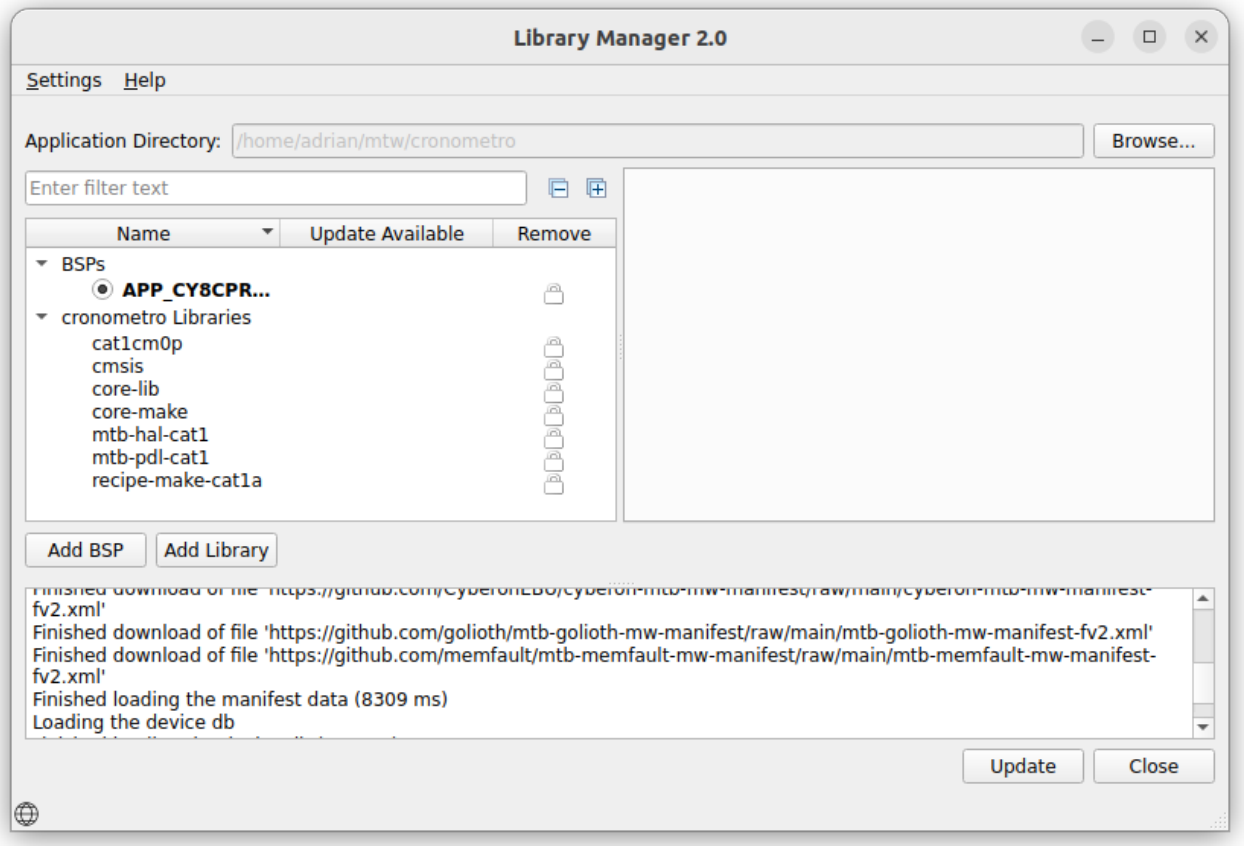
Importar la librería emWin

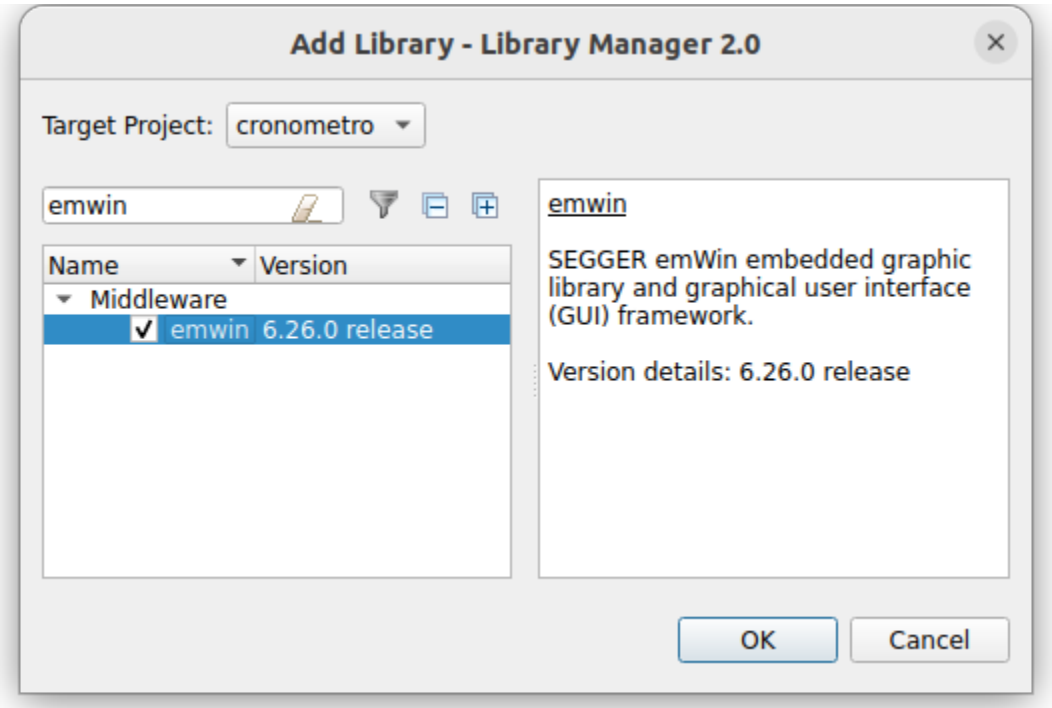
Para controlar el dispositivo se utiliza la librería emWin de SEGGER, que es un framework que permite el desarrollo de interfaces gráficas de usuario (GUI) eficientes y adaptables para cualquier aplicación que utilice una pantalla gráfica, independientemente del procesador y el controlador de pantalla utilizados. El fabricante de microcontroladores SEGGER Microcontroller desarrolló emWin y Cypress ha obtenido una licencia para utilizar la biblioteca emWin de SEGGER y ofrecer la misma de forma gratuita a sus clientes.

En este tutorial se detallan los pasos a seguir para la conexión del dispositivo TFT presentado anteriormente, para la comunicación con otros controladores u otra interfaz de comunicación se debe recurrir a la guía de inicio rápido proporcionada por Infineon.
https://infineon.github.io/emwin/emwin_overview/html/index.html

Utilizando el panel rápido accedemos al administrador de librerías. Para importar la libreria emWin al proyecto.



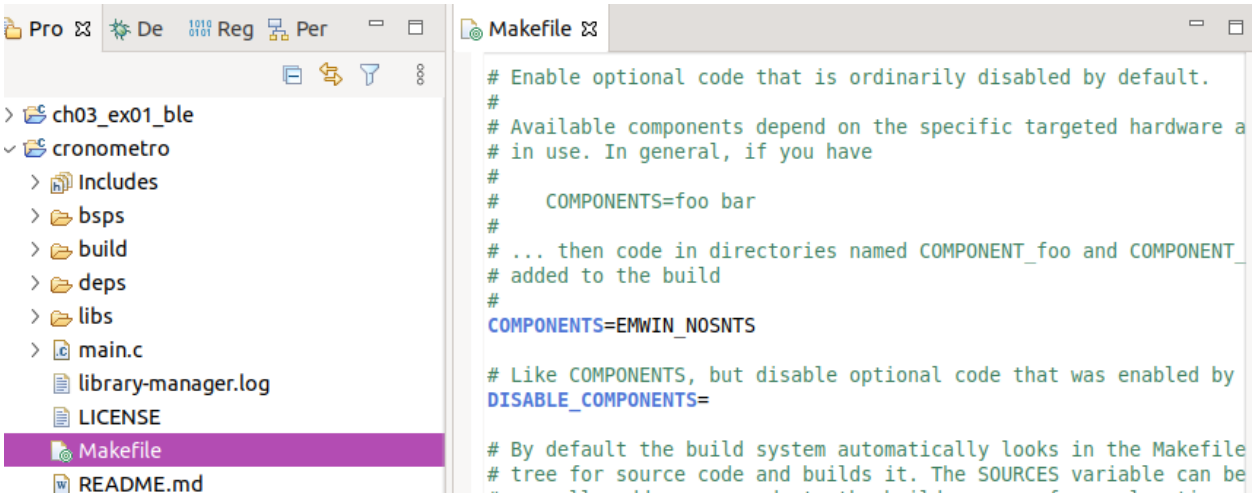

Library Manager 2.0
Settings
Help
Application Directory:
/home/adrian/mtw/cronometro
Browse...
Enter filter text
Name
Update Available
Remove
BSPs
APP_CY8CPR...
cronometro Libraries
cat1cm0p
cmsis
core-lib
core-make
mtb-hal-cat1
mtb-pdl-cat1
recipe-make-cat1a
Add BSP
Add Library
fv2.xml'
Finished download of file 'https://github.com/golioth/mtb-golioth-mw-manifest/raw/main/mtb-golioth-mw-manifest-fv2.xml'
Finished download of file 'https://github.com/memfault/mtb-memfault-mw-manifest/raw/main/mtb-memfault-mw-manifest-fv2.xml'
Finished loading the manifest data (8309 ms)
Loading the device db
Update
Close


Add Library - Library Manager 2.0
Target Project:
cronometro
emwin
Name
Version
Middleware
emwin 6.26.0 release
emwin
SEGGER emWin embedded graphic library and graphical user interface (GUI) framework.
Version details: 6.26.0 release
OK
Cancel

Configuración del Makefile

Es necesario indicar al compilador que no se hará uso del sistema de multitareas o el soporte táctil en este proyecto. Para esto se edita el archivo Makefile y en la lista de componentes se añade.

COMPONENTS=EMWIN_NOSNTS



Selección del driver

En la guía de emWin proporcionada por Infineon se muestra una tabla con los controladores de display soportados

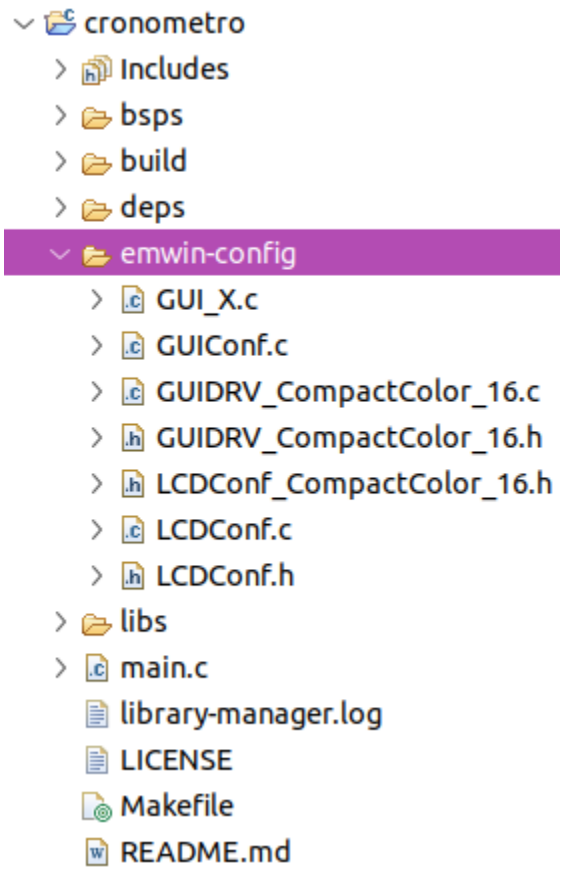
Driver	Supported display controller / Purpose of driver	Supported bits/pixel
GUIDRV_BitPlains	This driver can be used for solutions without a display controller. It manages separate bit-plains for each color bit. This driver does not have any display-controller-specific code and can be used for solutions that require color bits in separate plains.	1 - 8
GUIDRV_CompactColor_16	**Ampire:** FSA506 **Epson:** S1D13742, S1D13743, S1D19122 **FocalTech:** FT1509 **Himax:** HX8301, HX8312A, HX8325A, HX8340, HX8347, HX8352, HX8352B, HX8353 **Hitachi:** HD66766, HD66772, HD66789 **Ilitek:** ILI9161, ILI9220, ILI9221, ILI9320, ILI9325, ILI9326, ILI9328, ILI9342, ILI9481 **LG Electronics:** LGDP4531, LGDP4551 **MagnaChip:** D54E4PA7551 **Novatek:** NT39122, NT7573 **OriseTech:** SPFD5408, SPFD54124C, SPFD5414D, SPFD5420A **Renesas:** R61505, R61509, R61516, R61526, R61580, R63401 **Samsung:** S6D0110A, S6D0117, S6D0128, S6D0129, S6D04H0 **Sharp:** LCY-A06003, LR38825 **Sitronix:** ST7628, ST7637, ST7687, ST7712, ST7715, ST7735, ST7787, ST7789 **Solomon:** SSD1284, SSD1289, SSD1298, SSD1355, SSD1961, SSD1963, SSD2119 **Toshiba:** JBT6K71	16

Se observa que los dispositivos con el controlador de display ST7735 de Sitronix son soportados por el driver **GUIDRV_CompactColor_16**.

Una vez que sabemos que driver se utilizará se copian los archivos de configuración de **CompactColor_16** al proyecto. Para esto se crea una carpeta llamada “emwin-config” donde se pegarán los archivos de configuración del driver. A continuación se detalla la ruta de origen de cada uno de los archivos.

- mtb_shared/emwin/Config/CompactColor_16/GUI_X.c
- mtb_shared/emwin/Config/CompactColor_16/GUIConf.c
- mtb_shared/emwin/Config/CompactColor_16/GUIDRV_CompactColor_16.c
- mtb_shared/emwin/Config/CompactColor_16/GUIDRV_CompactColor_16.h
- mtb_shared/emwin/Config/CompactColor_16/LCDConf_CompactColor_16.h
- mtb_shared/emwin/Config/CompactColor_16/LCDConf.c
- mtb_shared/emwin/Config/CompactColor_16/LCDConf.h

El directorio del proyecto debe verse así:



Adaptar el driver

Los archivos de configuración copiados junto a la librería emWin contienen todas las funciones necesarias para trabajar con un display. Sin embargo requieren una ligera adaptación. Es necesario indicarle a la librería, el controlador de la pantalla, el tamaño de nuestra pantalla, los pines de comunicación con la placa y la implementación de las funciones para enviar comandos a través del protocolo SPI.
A continuación se detalla las modificaciones que requiere cada fichero.

GUIDRV_CompactColor_16.c

Este fichero no requiere modificación. En el comentario del inicio del fichero existe una lista con los controladores de display soportados y un número de identificación. Localiza el controlador de tu pantalla y anota el número de identificación. Para este caso la identificación de la configuración del controlador de display ST7735 corresponde a **66709**.

```
Configuration of LCD_CONTROLLER:

- 66700: Sharp LR38825
- 66701: Renesas R63401, R61509
         OriseTech SPFD5420A
         Ilitek ILI9326
- 66702: Solomon SSD1284, SSD1289, SSD1298
- 66703: Toshiba JBT6K71
- 66704: Sharp LCY-A06003
- 66705: Samsung S6D0129
- 66706: MagnaChip D54E4PA7551
- 66707: Himax HX8312A
- 66708: FocalTech FT1509
         Ilitek ILI9320, ILI9325, ILI9328
         LG Electronics LGDP4531, LGDP4551
         OriseTech SPFD5408
         Renesas R61505, R61580
- 66709: Epson S1D19122
         Himax HX8353, HX8325A
         Ilitek ILI9342
         Novatek NT39122
         Orisetech SPFD54124C, SPFD5414D
         Renesas R61516, R61526
         Sitronix ST7628, ST7637, ST7687, ST7735, ST7715
         Solomon SSD1355, SSD1961, SSD1963
- 66710: Novatek NT7573
- 66711: Epson S1D13742, S1D13743
```

LCDConf_CompactColor_16.h

En este archivo de configuración se colocará el número de identificación de configuración extraído en el paso anterior (línea 60).
Además nos permite definir la orientación de la pantalla (línea 70) para colocarla en modo vertical o modo horizontal.

```
54 /*************************************
55 *
56 *       General configuration of LCD
57 *
58 *************************************
59 */
60 #define LCD_CONTROLLER          66709
61
62 #define LCD_BITSPERPIXEL        16
63
64 //
65 // Data bus width selection
66 //
67 #define LCD_USE_PARALLEL_16     0
68
69 //
70 // LCD orientation settings
71 //
72 #define LCD_MIRROR_X            0
73 #define LCD_MIRROR_Y            0
74 #define LCD_SWAP_XY             1
75
```

Aquí además definiremos dos funciones: **write_data** y **read_data**. Que se implementarán en LCDConf.h . Estas funciones permitirán enviar y recibir información mediante SPI.

```
76 /*************************************
77 *
78 *       Simple bus configuration
79 *
80 *************************************
81 */
82
83 void write_data(U8 data);
84 U8 read_data(void);
85
86 void Display_WriteM8_A1(U8 *wrData, int
87 void Display_WriteM8_A0(U8 *wrData, int
88 void Display_ReadM8_A1(U8 *rdData, int
```

LCDConf.h

En este fichero es necesario definir una estructura de datos para determinar los puertos GPIO que se utilizarán en la comunicación SPI

```
60 typedef struct {
61     cyhal_gpio_t MOSI;
62     cyhal_gpio_t MISO;
63     cyhal_gpio_t SCK;
64     cyhal_gpio_t SS;
65     cyhal_gpio_t dc;
66     cyhal_gpio_t rst;
67 } SPI_pin;
```

LCDConf.c

En este archivo se configura el tamaño del display

```
69 // Physical display size
70 //   The display size should be adapted in order to match the size of
71 //   the target display.
72 //
73 #define XSIZE_PHYS 105
74 #define YSIZE_PHYS 160
```

En la sección “Display Access functions” se definen un objeto SPI y un objeto de la estructura SPI_pin definida en el paso anterior.

```
108 /*********************************************
109 *
110 *       Display access functions
111 *
112 *********************************************
113 */
114
115 cyhal_spi_t mSPI;
116
117 static const SPI_pin *pin;
118
```

Además se implementan las funciones para escribir y leer datos a través de SPI

```
119 void write_data(uint8_t data)
120 {
121     cyhal_spi_send(&SPI,data);
122 }
123
124 uint8_t read_data(void)
125 {
126     uint32_t receive_data = 0u;
127     cyhal_spi_recv(&SPI, &receive_data);
128     return receive_data;
129 }
```

```
149 void Display_WriteM8_A1(U8 *wrData, int numbytes)
150 {
151     int i;
152     cyhal_gpio_write(pin->dc, 1u);
153     for (i = 0; i < numbytes; i++)
154     {
155         write_data(wrData[i]);
156     }
157 }
158
159 void Display_WriteM8_A0(U8 *wrData, int numbytes)
160 {
161     int i;
162     cyhal_gpio_write(pin->dc, 0u);
163     for (i = 0; i < numbytes; i++)
164     {
165         write_data(wrData[i]);
166     }
167 }
168
169 void Display_ReadM8_A1(U8 *rdData, int numbytes)
170 {
171   GUI_USE_PARA(rdData);
172   GUI_USE_PARA(numbytes);
173 }
174
175 void Display_Write8_A0(U8 byte)
176 {
177     cyhal_gpio_write(pin->dc, 0u);
178     write_data(byte);
179 }
180
181 void Display_Write8_A1(U8 byte)
182 {
183     cyhal_gpio_write(pin->dc, 1u);
184     write_data(byte);
185 }
```

También se implementa una función para inicializar el protocolo SPI y una función para activar el pin de reinicio de la pantalla.

```
131 void SPI_st7735_write_reset_pin(bool value)
132 {
133     cyhal_gpio_write(pin->rst, value);
134 }
135
136 cy_rslt_t SPI_init8(const SPI_pins *data)
137 {
138     pins = data;
139     cy_rslt_t rslt = cyhal_spi_init(&mSPI, pin->MOSI, pin->MISO, pin->SCK, pin->SS, NULL, 8, CYHAL_SPI_MODE_00_MSB, false);
140     if (CY_RSLT_SUCCESS == rslt)
141     rslt = cyhal_spi_set_frequency(&mSPI, 8000000);
142     if (CY_RSLT_SUCCESS == rslt)
143     rslt = cyhal_gpio_init(pin->dc, CYHAL_GPIO_DIR_OUTPUT, CYHAL_GPIO_DRIVE_STRONG, 0u);
144     if (CY_RSLT_SUCCESS == rslt)
145     rslt = cyhal_gpio_init(pin->rst, CYHAL_GPIO_DIR_OUTPUT, CYHAL_GPIO_DRIVE_STRONG, 1u);
146     return rslt;
147 }
```

Seguidamente se asignan constantes a los comandos del driver ST7735, los mismos pueden encontrarse en el manual de uso del driver.

```
187 /*************************************************************************
188 *
189 *       ST7735 Commands
190 *
191 *************************************************************************
192 */
193 #define ST7735_NOP     0x00
194 #define ST7735_SWRESET 0x01
195 #define ST7735_RDDID   0x04
196 #define ST7735_RDDST   0x09
197
198 #define ST7735_SLPIN   0x10
199 #define ST7735_SLPOUT  0x11
200 #define ST7735_PTLON   0x12
201 #define ST7735_NORON   0x13
202
203 #define ST7735_INVOFF  0x20
204 #define ST7735_INVON   0x21
205 #define ST7735_DISPOFF 0x28
206 #define ST7735_DISPON  0x29
207 #define ST7735_CASET   0x2A
208 #define ST7735_RASET   0x2B
209 #define ST7735_RAMWR   0x2C
210 #define ST7735_RAMRD   0x2E
211
212 #define ST7735_PTLAR   0x30
213 #define ST7735_COLMOD  0x3A
214 #define ST7735_MADCTL  0x36
215
216 #define ST7735_FRMCTR1 0xB1
217 #define ST7735_FRMCTR2 0xB2
218 #define ST7735_FRMCTR3 0xB3
219 #define ST7735_INVCTR  0xB4
220 #define ST7735_DISSET5 0xB6
221
222 #define ST7735_PWCTR1  0xC0
223 #define ST7735_PWCTR2  0xC1
224 #define ST7735_PWCTR3  0xC2
225 #define ST7735_PWCTR4  0xC3
226 #define ST7735_PWCTR5  0xC4
227 #define ST7735_VMCTR1  0xC5
228
229 #define ST7735_RDID1   0xDA
230 #define ST7735_RDID2   0xDB
231 #define ST7735_RDID3   0xDC
232 #define ST7735_RDID4   0xDD
233
234 #define ST7735_PWCTR6  0xFC
235
236 #define ST7735_GMCTRP1 0xE0
237 #define ST7735_GMCTRN1 0xE1
```

Finalmente se implementa la función **_InitController()** que contiene la secuencia de comandos necesarios para inicializar el controlador del display.

```
246 static void _InitController(void)
247 {
248   /* Set up the display controller and put it into operation. If the
249    *  display controller is not initialized by any external routine
250    *  this needs to be adapted by the customer.
251    */
252     /* Reset the display controller */
253             SPI_st7735_write_reset_pin(0u);
254             GUI_Delay(100);
255             SPI_st7735_write_reset_pin(1u);
256             GUI_Delay(50);
257             Display_Write8_A0(ST7735_SWRESET);
258             GUI_Delay(200);
259             Display_Write8_A0(ST7735_SLPOUT);
260             GUI_Delay(255);
261             Display_Write8_A0(ST7735_FRMCTR1);
262             Display_Write8_A1(0x01);
263             Display_Write8_A1(0x2C);
264             Display_Write8_A1(0x2D);
265             Display_Write8_A0(ST7735_FRMCTR2);
266             Display_Write8_A1(0x01);
267             Display_Write8_A1(0x2C);
268             Display_Write8_A1(0x2D);
269             Display_Write8_A0(ST7735_FRMCTR3);
270             Display_Write8_A1(0x01);
271             Display_Write8_A1(0x2C);
272             Display_Write8_A1(0x2D);
273             Display_Write8_A1(0x01);
274             Display_Write8_A1(0x2C);
275             Display_Write8_A1(0x2D);
276             Display_Write8_A0(ST7735_INVCTR);
277             Display_Write8_A1(0x07);
278             Display_Write8_A0(ST7735_PWCTR1);
279             Display_Write8_A1(0xA2);
280             Display_Write8_A1(0x02);
281             Display_Write8_A1(0x84);
282             Display_Write8_A0(ST7735_PWCTR2);
283             Display_Write8_A1(0xC5);
284             Display_Write8_A0(ST7735_PWCTR3);
285             Display_Write8_A1(0x0A);
286             Display_Write8_A1(0x00);
287             Display_Write8_A0(ST7735_PWCTR4);
288             Display_Write8_A1(0x8A);
289             Display_Write8_A1(0x2A);
290             Display_Write8_A0(ST7735_PWCTR5);
291             Display_Write8_A1(0x8A);
292             Display_Write8_A1(0xEE);
293             Display_Write8_A0(ST7735_VMCTR1);
```

Lógica principal de la aplicación

En el fichero main.c se incluye las librerías de interfaz gráfica GUI.h, nuestra configuración adaptada de emWin LCDConf.h y la librería stdio.h que se utiliza para dar formato al texto mostrado en pantalla.

```
43 /************************
44  * Header Files
45  ************************/
46 #include "cyhal.h"
47 #include "cybsp.h"
48 #include "cy_pdl.h"
49 #include "GUI.h"
50 #include "LCDConf.h"
51 #include <stdio.h>
52
```

Seguidamente se define una constante que determina la prioridad de la interrupción que será asignada al botón de usuario de la placa.

```
/************************************
 * Macros
 ************************************/
#define GPIO_INTERRUPT_PRIORITY (7u)
```

Se define el prototipo de la función de callback que será llamada al presionar el botón. Esta función pone en pausa o reanuda el conteo según el caso.

```
59 /*****************************************************************************
60  * Function Prototypes
61  *****************************************************************************/
62
63 static void button_isr(void *handler_arg, cyhal_gpio_event_t event);
64
```

En las variables globales se define:

La estructura SPI_pin donde se asigna un puerto de la placa a cada pin SPI, nótese que el pin MISO no se utiliza para este proyecto pero es necesario definirlo de todas formas.

La configuración de la función callback para la interrupción.

Las variables segundos, minutos y horas para llevar la cuenta del tiempo transcurrido. Además de una variable booleana para indicar si el cronómetro se encuentra o no en pausa.

Un objeto PWM para controlar el tono de la bocina.

```
65
66 /*******************************************
67 * Global Variables
68 *******************************************
69
70 const SPI_pins pins =
71 {
72     .MOSI = P5_0,  //SDA
73     .MISO = P5_1,  //MISO
74     .SCK = P5_2,   //SCL
75     .SS = P5_3,    //CS
76     .dc = P9_7,    //RS
77     .rst= P9_4,    //RST
78 };
79
80 /* GPIO callback initialization structure */
81 cyhal_gpio_callback_data_t cb_data =
82 {
83 .callback = button_isr,
84 .callback_arg = NULL
85 };
86
87 int segundos, minutos, horas;
88
89 bool pause = false;
90
91 cyhal_pwm_t pwm_obj;
```

La función de callback simplemente permuta la variable “pause” y emite una alerta sonora

```
92 /*******************************************************************************
93 * Function Definitions
94 *******************************************************************************/
95
96 /* Interrupt callback function */
97 static void button_isr(void *handler_arg, cyhal_gpio_event_t event)
98 {
99     pause = !pause;
100
101     cyhal_pwm_set_duty_cycle(&pwm_obj, 20, 700);
102     cyhal_pwm_start(&pwm_obj);
103     CyDelay(100);
104     cyhal_pwm_stop(&pwm_obj);
105     CyDelay(25);
106     cyhal_pwm_start(&pwm_obj);
107     CyDelay(100);
108     cyhal_pwm_stop(&pwm_obj);
109
110     cyhal_pwm_set_duty_cycle(&pwm_obj, 20, 1000);
111 }
112
```

Para inicializar la pantalla, en la función main se llama a GUI_INIT, se establece el color de fondo, el color de la fuente y se coloca el texto “Cronómetro” en la posición (5, 30). Además se selecciona una fuente diferente para dibujar el conteo del cronómetro.

```
GUI_Init();
GUI_SetBkColor(GUI_WHITE);
GUI_Clear();
GUI_SetColor(GUI_BLACK);
GUI_DispStringAt("Cronometro", 5, 30);
GUI_SetFont(GUI_FONT_10_ASCII);
```

Luego se inicializa el LED de usuario
Y se inicializa el PWM en el pin P8_0, además se emite una alerta sonora para indicar que el dispositivo se encendió y la cuenta está por comenzar.

```
cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
        CYHAL_GPIO_DRIVE_STRONG, 0);

/* Initialize PWM on the supplied pin and assign a new clock */
cyhal_pwm_init(&pwm_obj, P8_0, NULL);
cyhal_pwm_set_duty_cycle(&pwm_obj, 20, 500);

cyhal_pwm_start(&pwm_obj);
CyDelay(200);
cyhal_pwm_stop(&pwm_obj);
CyDelay(100);
result = cyhal_pwm_start(&pwm_obj);
CyDelay(200);
cyhal_pwm_stop(&pwm_obj);

result = cyhal_pwm_set_duty_cycle(&pwm_obj, 20, 1000);
```

También es necesario inicializar el botón de usuario e indicar que ante una interrupción se llamará a la función de callback.

```
/* Initialize the button and setup the interrupt */
 cyhal_gpio_init(CYBSP_USER_BTN, CYHAL_GPIO_DIR_INPUT,
CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);

 cyhal_gpio_register_callback(CYBSP_USER_BTN, &cb_data);
 cyhal_gpio_enable_event(CYBSP_USER_BTN, CYHAL_GPIO_IRQ_FALL,
GPIO_INTERRUPT_PRIORITY, true);
```

Se establecen los valores iniciales del conteo en cero. Y durante el loop infinito se realizan las pausas e incrementos necesarios mientras se actualiza el valor en pantalla mediante la función **GUI_DispStringAt()**

```
segundos = 0;
minutos = 0;
horas = 0;

while(1)
{
      if (pause) {
          cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_OFF);
          cyhal_pwm_stop(&pwm_obj);
          continue;
      }
      if (segundos == 60){
         minutos++;
         segundos = 0;
      }

      if (minutos == 60){
         horas++;
         minutos = 0;
      }

      char buffer[16];
      sprintf(buffer, "%02d:%02d:%02d", horas, minutos, segundos);
      GUI_DispStringAt(buffer, 5, 53);

      cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_ON);
      cyhal_pwm_stop(&pwm_obj);

      CyDelay(500);

      if (!pause) {
          segundos++;
          cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_OFF);
          CyDelay(500);
          cyhal_pwm_start(&pwm_obj);
      }

}
```