

Sistemas Electrónicos Para Automatización

PRÁCTICA 4: *Monitorización y Control de Variables Ambientales.*

10/11/2021

Francisco Javier Román Cortés 4º GIERM

Introducción

En esta cuarta práctica de la parte de microcontroladores, vamos a seguir profundizando sobre las posibilidades del sistema VM800 como interfaz, donde además vamos a entrar a trabajar con el Sensors Boosterpack, que presenta una serie de sensores a partir de los cuales mostraremos magnitudes ambientales en la pantalla del sistema VM800, combinando ambos elementos para conseguir cierta funcionalidad concreta.

Fundamento teórico

Es necesario conocer el manejo y programación de la pantalla VM800, con la cual se trabajó intensivamente en la práctica 3, aunque en esta apareció una novedad, el diseño de funciones personalizadas para representar widgets diseñados por nosotros mismos en la pantalla, estas funciones podrían ser incluidas en la librería FT800_TIVA.h similares a otras muchas que esta librería incluye o bien en el programa principal (.c), esto ya a preferencia del alumno.

Además de gestionar relativamente bien la pantalla, deberemos entender como trabajan los sensores y qué mide cada uno antes de tratar de trabajar con sus mediciones. Esto se vio en los primeros temas de teoría de manera que, a falta de refrescar la información, se conocen bien los sensores de este Sensors Boosterpack.

Realización de la práctica

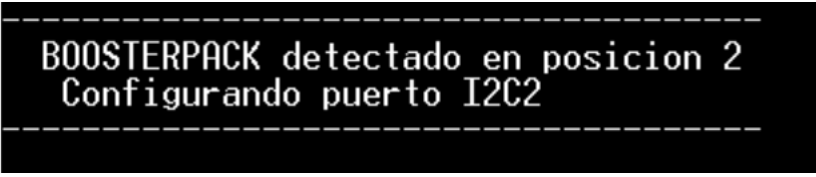
1. Ejecución del ejemplo 8

En primer lugar, antes de proceder a enfrentarse a los ejercicios propuestos para esta práctica, se propone como en prácticas anteriores, acudir a ejemplos planteados y tratar de comprenderlos para aclarar una serie de conceptos sobre la programación y facilitar el planteamiento de los siguientes ejercicios.

En este ejemplo vemos como utilizando la UART, mostramos una serie de mensajes para comprobar el correcto funcionamiento de los sensores. De esta manera, tenemos tanto una serie de mensajes para saber en qué posición hemos conectado el Sensors Boosterpack, así como mensajes de inicialización de los diferentes sensores para comprobar que funcionan correctamente. Normalmente, ninguno dará error, salvo en el caso especial del TMP007, que la mayoría de boosterpacks carecen de dicho sensor y, por tanto, nos mostrará un error en la inicialización de este sensor. Finalmente, se muestran por la UART las diferentes medidas de todos los sensores funcionales refrescándose cada cierto tiempo.

De esta manera, partiendo de este código, se puede plantear como obtener una representación de magnitudes ambientales usando la pantalla VM800 en lugar de la UART, teniendo así una interfaz más funcional y atractiva que simplemente texto sobre fondo negro, que es lo que ofrece la UART.

Aunque no sea necesario, adjunto una captura de las medidas sobre la UART para comprobar la correcta ejecución de este ejemplo inicial.



```
-----  
BOOSTERPACK detectado en posicion 2  
Configurando puerto I2C2  
-----
```

```

Iniciando OPT3001... Hecho!
Leyendo DevID... DevID= 0X3001
Iniciando ahora el TMP007...Error en TMP007
Iniciando BME280... Hecho!
Leyendo DevID... DevID= 0X3060
Iniciando BMI160, modo NAVIGATION... Hecho!
Leyendo DevID... DevID= 0X30d1

-----
OPT3001: 84.320 Lux
-----
BME: T:21.95 C P:998.99mbar H:42.817
-----
BMM: X: -631 Y: 259 Z: 288
-----
ACCL: X: -143 Y: -984 Z: 16995
-----
GYRO: X: 21 Y: 7 Z: -14
-----
TLect: 12 ticks (x0.1ms)
TRep: 09 tick (x0.1m)

```

Comprobamos en este caso, como aparece el error del sensor TMP007, que como hemos comentado, implica que el Sensors Boosterpack carece de dicho sensor.

2. Sensor de magnitudes ambientales con interfaz digital

Pasamos ahora, al primer ejercicio propiamente dicho a realizar en la práctica. En este ejercicio básicamente haremos algo similar al ejemplo 8 anteriormente descrito, de manera que mostraremos por la pantalla VM800 la información de algunos de los sensores, concretamente las del BME280 (Temperatura, Presión y Humedad) y la luminosidad del OPT3001. Esta información será actualizada 2 veces por segundo, redibujando la pantalla.

Este primer ejercicio es bastante simple y, por ello, no requiere de máquinas de estado ni estructuras de programación complejas, que requieran de planteamiento a mano y un cierto desarrollo previo antes de la programación, como alguna de las prácticas anteriores.

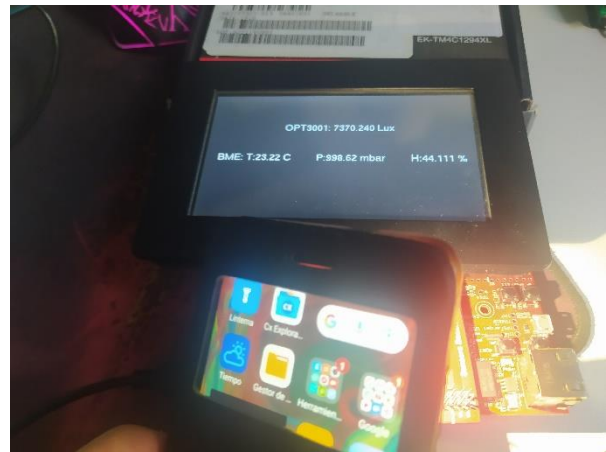
De esta manera, como resumen simple de cómo se resuelve este primer ejercicio, tenemos que debido a la similitud entre funcionalidad del ejemplo 8 y lo que requiere este ejercicio, podemos “reacondicionar” el programa del ejemplo 8 para este ejercicio. Por tanto, combinándolo con los elementos necesarios para manejar la pantalla, logramos mostrar estas magnitudes ambientales por la pantalla en lugar de por la UART.

Antes de comentar y explicar el código, podemos mostrar los resultados del programa en imágenes para ver como aparecen las magnitudes ambientales en la pantalla de los sensores especificados, en texto blanco sobre fondo negro. Adjuntaré varias fotos, explicando el por qué de ciertas medidas obtenidas.

-Medida “normal” sin alterar el entorno



-Medida de luminosidad elevada (linterna)



-Medida alterando la temperatura (se ha usado un secador a una distancia prudencial)



Por tanto, como ya se ha dicho, no hay mucho análisis previo para este ejercicio, pudiendo pasarse a adjuntar el código íntegro desarrollado para este ejercicio (bastante comentado), para su posterior explicación, como de costumbre.

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>           //Declaración de librerías necesarias
#include <stdbool.h>
#include "driverlib2.h"
#include <stdlib.h>
#include <math.h>
#include "FT800_TIVA.h"
#include "HAL_I2C.h"
#include "sensorlib2.h"

// =====
//  VARIABLES PARA CONFIGURACIÓN Y TRABAJO DE LA PANTALLA
// =====
#define dword long
#define byte char

#define MSEC 40000 //Valor para 1ms con SysCtlDelay()

char chipid = 0;           // Holds value of Chip ID read from
the FT800

unsigned long cmdBufferRd = 0x00000000;      // Store the value read from
the REG_CMD_READ register
unsigned long cmdBufferWr = 0x00000000;      // Store the value read from
the REG_CMD_WRITE register
unsigned int t=0;
//
#####
#####
// User Application - Initialization of MCU / FT800 / Display
//
#####
#####
unsigned long POSX, POSY, BufferXY;
unsigned long POSYANT=0;                      //Variables y constantes
relacionadas con la configuración de la pantalla
unsigned int CMD_Offset = 0;
unsigned long REG_TT[6];
const int32_t REG_CAL[6]={21696, -78, -614558, 498, -17021, 15755638};
const int32_t REG_CAL5[6]={32146, -1428, -331110, -40, -18930, 18321010};

#define NUM_SSI_DATA          3

// =====
//  VARIABLES PARA CONFIGURACIÓN Y TRABAJO DE LOS SENSORES
// =====
char Cambia=0;           //Variable para "refrescar" el sistema cada 500ms, no se
hace con Sistema Bajo Consumo esta vez
char string[80];         //Array para usar sprintf y sacar diferentes mensajes por
pantalla
int DevID=0;             //Para mostrar la DevID, se usaba con la UART, ahora ya
no
int RELOJ;               //Variable para guardar el reloj de 120MHz
```

```

// OPT3001
int lux_i;           //Variables para la luz (OPT3001)
float lux;           //Variable para recoger la medida de la luz del sensor
OPT3001

// BME280 (Mide Temperatura, Presión y Humedad)
int returnRslt;      //se usa para ver si la función que
obtiene las medidas del BME280 devuelve éxito(0) o error(-1)
int g_s32ActualTemp = 0;
unsigned int g_u32ActualPress = 0; //Recogeremos aquí las medidas que
nos da la función bme280_read_pressure_temperature_humidity()
unsigned int g_u32ActualHumidity = 0;
float T_act,P_act,H_act; //Tras calcular y formatear las
medidas, guardaremos aquí las medidas con el formato deseado
bool BME_on = true; //Entiendo que este bool sería para
hacer lo que se hace con Bme_OK, pero no se usa.

uint8_t Sensor_OK=0; //Para comprobar si el acceso a I2C
correspondiente es exitoso o erróneo
#define BP 2 //Define por si se conecta en la posición
2 del boosterpack
uint8_t Opt_OK, Bme_OK; //Para registrar en variables si el
acceso a los sensores fue exitoso o no

void Timer0IntHandler(void); //Prototipo para la rutina de interrupción
del Timer

int main(void)
{
    int i;
    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 12000000);
    /***** CONF PANTALLA *****/
    HAL_Init_SPI(1, RELOJ); //Boosterpack a usar,
    Velocidad del MC
    Inicia_pantalla(); //Arranque de la pantalla
    //
    =====
    // Delay before we begin to display anything
    //
    =====
    SysCtlDelay(RELOJ/3);

    /***** TIMER0 *****/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); //Habilita T1
    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM); //T1 a 120MHz
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); //T1 periódico y
conjunto (32b)
    TimerLoadSet(TIMER0_BASE, TIMER_A,(RELOJ/2)-1); //Se carga el
timer1 con 500ms
    TimerIntRegister(TIMER0_BASE,TIMER_A,Timer0IntHandler); //Cada vez
que cuente 500ms que entre en la interrupción y refresque el sistema
    IntEnable(INT_TIMER0A); //Habilitar las
interrupciones globales de los timers
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Habilitar las
interrupciones de timeout
    IntMasterEnable();

```

```

    TimerEnable(TIMER0_BASE, TIMER_A); //Habilitar
Timer0 (no hace falta recargarlo, se activa al principio y que cuente
"infinito")

/*****
*****/

/***** UART
*****/
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
UARTStdioConfig(0, 115200, RELOJ);

/*****
*****/

#ifdef VM800B35
    for(i=0;i<6;i++) Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL[i]);
#endif
#ifdef VM800B50
    for(i=0;i<6;i++) Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL5[i]);
#endif

/***** INICIALIZACIÓN DE SENSORES
*****/

//Configuración Boosterpack según su conexión en la placa
if(Detecta_BP(1)) Conf_Boosterpack(1, RELOJ);
else if(Detecta_BP(2)) Conf_Boosterpack(2, RELOJ);
else return 0; //Si no detecta que se ha conectado en ninguna de
las posiciones se acaba el programa

//Inicializar OPT3001
Sensor_OK=Test_I2C_Dir(OPT3001_SLAVE_ADDRESS);
if(!Sensor_OK) Opt_OK=0;
else{
    OPT3001_init();
    DevID=OPT3001_readDeviceId();
    Opt_OK=1;
}

//Inicializar BME280
Sensor_OK=Test_I2C_Dir(BME280_I2C_ADDRESS2);
if(!Sensor_OK) Bme_OK=0;
else{
    bme280_data_readout_template();
    bme280_set_power_mode(BME280_NORMAL_MODE);
    readI2C(BME280_I2C_ADDRESS2,BME280_CHIP_ID_REG, &DevID, 1);
    Bme_OK=1;
}

while(1){ //Bucle infinito repitiéndose

    if(Cambia==1){ //Cada 500ms, el timer pone a 1 "Cambia" y ejecutamos
este código

```

```

        Cambia=0; //Se resetea "Cambia" para que
        vuelva a esperar arriba a que el timer la active
        Nueva_pantalla(16,16,16); //Se prepara una nueva pantalla
        if(Opt_OK){ //Si el sensor OPT3001
funciona correctamente:
            lux=OPT3001_getLux(); //Obtenemos la medida de la
            luz en lux (aquí la guarda en float)
            lux_i=(int)round(lux); //Guarda en otra variable la
            medida de lux convertida a entero por si interesa
        }
        if(Bme_OK){ //Si el sensor BME280 funciona
correctamente:
            returnRslt =
bme280_read_pressure_temperature_humidity(&g_u32ActualPress,
&g_s32ActualTemp, &g_u32ActualHumity); //Recoge las medidas del BME280
            T_act=(float)g_s32ActualTemp/100.0; //Recogemos en
formato float la temperatura medida
            P_act=(float)g_u32ActualPress/100.0; //Recogemos en
formato float la presión medida
            H_act=(float)g_u32ActualHumity/1000.0; //Recogemos en
formato float la humedad medida
        }
        if(Opt_OK){ //Si el sensor OPT3001 funciona
correctamente:
            sprintf(string," OPT3001: %5.3f Lux ",lux);
//Guardamos una string con la medida de la luz
            ComColor(255,255,255); //Elige
color blanco
            ComTXT(HSIZE/2,VSIZE/4, 23, OPT_CENTERX,string);
//Mostramos por pantalla la medida de la luz
        }
        if(Bme_OK){ //Si el sensor BME280 funciona
correctamente:
            sprintf(string, " BME: T:%.2f C          P:%.2f mbar
H:%.3f %",T_act,P_act,H_act); //Guardamos una string con la medida de la
            temperatura, presión y humedad
            ComColor(255,255,255);
//Elige color blanco
            ComTXT(HSIZE/2,VSIZE/2, 23, OPT_CENTERX,string);
//Mostramos por pantalla las 3 medidas
        }
        Dibuja(); //Pinta en la pantalla los elementos anteriores
    }
}
return 0;
}

void Timer0IntHandler(void) //Rutina de interrupción del Timer0
{
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag de la
interrupción
    Cambia=1; //Vuelve a activar la
variable "Cambia" para que se actualice la pantalla y las medidas 2 veces por
segundo
    SysCtlDelay(100); //Pequeño delay por
"bug" conocido e indicado por el manual
}

```


Con esto, podemos proceder a explicar el código. Como de costumbre y al igual que en el resto de las prácticas, voy a ir explicando desde el principio hasta el final, aunque he notado que en ciertos casos cuando se crean funciones y luego se llaman en el main puede ser algo confuso este orden “cronológico”, pero creo que explicándolo correctamente se puede seguir bien el razonamiento realizado.

Primeramente, declaramos las librerías necesarias para nuestro programa. Tras ello, tenemos los defines y declaración de variables agrupados que corresponden a la pantalla VM800, que prácticamente se configuran de la misma forma que en la práctica 3, por lo que tampoco tiene mucho interés detenerse en ellas.

Ahora se llega a las variables que se usarán con los sensores, vemos que la variable cambia empieza a 0, y como indica su comentario, será la que permita el refresco del sistema cada 500ms mediante su valor 0 ó 1. Declaramos un array llamado string, con tamaño de sobra, para usar sprintf e imprimir texto mediante esta string. DevID, es una especie de identificador que se ha dejado, pero ya no se usa en este ejercicio. RELOJ es la conocida variable para guardar la configuración del reloj de 120MHz en este caso.

Con esto, llegamos a las variables del OPT3001, donde tendremos una tipo float para guardar la medición de luminosidad con decimales, y otra para convertir esa medición a entero.

Ahora, respecto al BME280, configuramos una serie de variables con el formato necesario del fabricante, donde recogeremos los valores medidos mediante una función propia del fabricante y luego mediante ciertos cálculos, las guardaremos como nos interesan en las variables tipo float: “T_act, P_act y H_act”.

Finalmente tenemos una serie de variables de configuración fundamentalmente para comprobar si los sensores están funcionales y el define BP 2 para comprobar en que zona del boosterpack se ha conectado el Sensors Boosterpack.

Definimos el prototipo de la rutina de interrupción del timer y entramos en el main. Configuramos aquí el reloj de 120MHz y a continuación se configura la pantalla, con la función HAL_Init_SPI(1,RELOJ), donde le indicamos que se ha conectado en el boosterpack 1 y se le pasa también el reloj de 120 MHz. Tras ello, se arranca la pantalla con otra función y se introduce un cierto delay o espera, especificado por el fabricante.

Se configura también el Timer0 como periódico y con una carga de 500ms, para actualizar la pantalla con las nuevas mediciones cada 500ms. Se asocia la rutina de interrupción del timer y se activa dicho timer.

Ahora, inicializamos los sensores, primero, configurando el boosterpack según en que posición se ha conectado el Sensors Boosterpack.

A continuación, se inicializan el OPT3001 y el BME280. Entramos finalmente en el while(1), donde, mediante el uso de la variable “Cambia” y su interacción con el timer, sólo trabajará (no entender mal, no está en modo de bajo reposo, simplemente no hace nada mientras) cada 500ms. Una vez el timer reactiva la variable cambia, se cumple la condición y, antes que nada, reseteamos “cambia”, y preparamos una nueva pantalla. A continuación, si ambos sensores son correctos, guardamos en las variables correspondientes cada una de las medidas. Finalmente, mostramos por pantalla tanto la medida de luminosidad del OPT3001 en [lux] en la parte superior de la pantalla como las 3 medidas del BME280 por debajo de la medida anterior, ambas centradas en la pantalla, con texto en color blanco sobre fondo negro.

Tras esto, tenemos el Dibuja(), para mostrar toda lo que se ha comentado por la pantalla.

Por último, tenemos la rutina de interrupción del timer, que se activa cada 500ms y como ya hemos ido comentando, principalmente activa de nuevo la variable “cambia”, para reejecutar el código anterior dentro del while(1), aparte de borrarse el flag del timer.

3. Sensor de magnitudes ambientales con interfaz analógico y control fuzzy de luz

Tras haber logrado realizar el ejercicio anterior, se plantea profundizar más y realizar una interfaz más interesante en la que predomine un estilo analógico, para ello, se deben crear una serie de *widgets* mediante los cuales representaremos la información de las diferentes magnitudes ambientales.

En este caso, en lugar de plantar el código y analizarlo en un texto extenso, creo que es más explicativo ir widget por widget explicando cómo funciona y cómo se ha diseñado para no perderse luego buscando a lo largo del código cual era cada uno. Destacar que las funciones se han incluido dentro del .c, es decir no se ha modificado la librería FT800_TIVA.

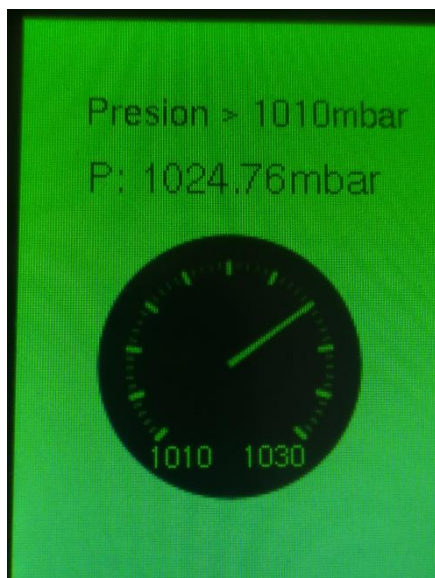
Por tanto, realizaré la explicación de esta manera, siguiendo el orden de widgets propuesto en el enunciado:

- **Medidor tipo Gauge para presión**

Se trata de un medidor analógico tipo “velocímetro”, cuya aguja va a indicar la presión atmosférica en un rango entre 1010 y 1030 mbar. Como veremos, por diferencia entre rangos de presión según donde se mida, he hecho un reescalado del Gauge al rango entre 990 y 1010 mbar, ya que en mi casa difícilmente pasa de los 1000 mbar, lo cual no es funcional en el rango de presiones propuesto. Sin embargo, mientras lo hacía en Sevilla, en la biblioteca, si que marcaba unos 1014 mbar, lo cual sí encaja en el rango propuesto.

Adjunto fotos de las dos posibilidades del Gauge según la presión esté por debajo o por encima de 1010 mbar, así como para mostrar su diseño.

-Medida a más de 1010 mbar



-Medida a menos de 1010 mbar



Hay además una función usada para todos los widgets, que adapta las medidas tomadas a cierto rango de funcionamiento del widget en cuestión y, por tanto, conviene comentarla, ya que va a aparecer con frecuencia:

```
166//Función para mapear el rango de las variables ambientales para que se adapten correctamente al rango
167double map(float valor, float entradaMin, float entradaMax, float salidaMin, float salidaMax) //Dest:
168{
169    return (((valor-entradaMin)*(salidaMax-salidaMin))/(entradaMax-entradaMin))+salidaMin);
170}
171
```

Ahora mostraré tanto la función que genera el Gauge, como la parte del código donde se configura y se llama a dicha función:

```
83// Función para pintar un Gauge(Indicador de aguja tipo velocímetro), basada en el prototipo de la FTDI Programmer Guide
84void ComGauge( int16_t x,int16_t y,int16_t r,uint16_t options,uint16_t major,uint16_t minor,uint16_t val,uint16_t range ){
85    EscribeRam32(CMD_GAUGE);
86    EscribeRam16(x);
87    EscribeRam16(y); //8 parametros de 16bits, 16 bytes: BIEN
88    EscribeRam16(r); //si fuesen 7 parametros de 16 bits, hay que mandar un ultimo EscribeRam16 con ceros.
89    EscribeRam16(options);
90    EscribeRam16(major);
91    EscribeRam16(minor);
92    EscribeRam16(val);
93    EscribeRam16(range);
94}
as
```

Esta parte siguiente va dentro del while(1).

```
if(Bme_OK) //Si el sensor BME280 funciona correctamente:
{
    //Se ha notado que en Olivares (mi pueblo, la presión oscila entre 990 y 1010mbar)(a veces lo supera)
    //Luego según si está probándose en Sevilla (que tiene más presión) o Olivares,
    //He decidido reescalar el gauge en el caso de que la presión suba o baje por encima de 1010
    //Ya que si no el gauge al salirse de los límites hace "cosas raras".
    //El Gauge se reescala entre 990 y 1010 si la presión baja de 1010 y 1030 si sube de 1010mbar
    // P_act = 1024.76; //probar si reescala el Gauge entre "Presion>1010mbar" y "Presion<1010mbar" //comentar o descomentar

    if(P_act >= 990 && P_act <= 1010){ //Reescalado del Gauge para Rango entre 990 y 1010 si Presión medida es inferior a 1010 mbar
        ComColor(65,202,42); //Color verde "fondo" para aguja y marcadores del gauge
        ComBgcolor(0,0,0); //Color negro para el fondo del gauge
        ComGauge(HSIZE/6,VSIZ/2,50,OPT_FLAT,11,5,map(P_act,990,1010,0,20),20); //Se crea el gauge con la función anterior, mapeando los
        sprintf(string, "P: %.2f mbar", P_act); //Guardamos string con la presión medida
        ComColor(65,202,42); //Color verde "fondo" para números 990 y 1010, indicadores mínimo y máximo del rango del gauge e
        //ComBgcolor(0,0,0);
        ComTXT(50,VSIZ/5+110, 20, 0,"990"); //Pintamos los indicadores mínimo y máximo del rango del gauge en este caso
        ComTXT(25+60,VSIZ/5+110, 20, 0,"1010");
        ComColor(0,0,0); //Elige color negro
        ComTXT(25,-25+VSIZ/5, 21, 0,"Presion < 1010mbar"); //Indicamos por encima de la medida de la presión que estamos en el caso esc.
        ComTXT(25,VSIZ/5, 22, 0,string); //Pintamos el string con la presión medida
    }
    if(P_act >= 1010 && P_act <= 1030){ //Reescalado del Gauge para Rango entre 1010 y 1030 si Presión medida es superior a 1010 mbar
        ComColor(65,202,42); //Color verde "fondo" para aguja y marcadores del gauge
        ComBgcolor(0,0,0); //Color negro para el fondo del gauge
        ComGauge(HSIZE/6,VSIZ/2,50,OPT_FLAT,11,5,map(P_act,1010,1030,0,20),20); //Se crea el gauge con la función anterior, mapeando los
        sprintf(string, "P: %.2fmbar", P_act); //Guardamos string con la presión medida
        ComColor(65,202,42); //Color verde "fondo" para números 1010 y 1030, indicadores mínimo y máximo del rango de
        //ComBgcolor(0,0,0);
        ComTXT(50,VSIZ/5+110, 20, 0,"1010"); //Pintamos los indicadores mínimo y máximo del rango del gauge en este caso
        ComTXT(25+60,VSIZ/5+110, 20, 0,"1030");
        ComColor(0,0,0); //Elige color negro
        ComTXT(25,-25+VSIZ/5, 21, 0,"Presion > 1010mbar"); //Indicamos por encima de la medida de la presión que estamos en el caso esc.
        ComTXT(25,VSIZ/5, 22, 0,string); //Pintamos el string con la presión medida
    }
}
```

Donde como vemos en función del valor de P_act, básicamente si está por encima o por debajo de 1010mbar, se reescala el gauge entre dos rangos de presiones ya comentados. La mayoría de órdenes son para mostrar texto e información lo cual no tiene mucho interés comentarlo, pero interesa ver como se ha usado la función map(), comentada anteriormente para que la presión medida se ajuste al rango completo del gauge. Además, destacar también

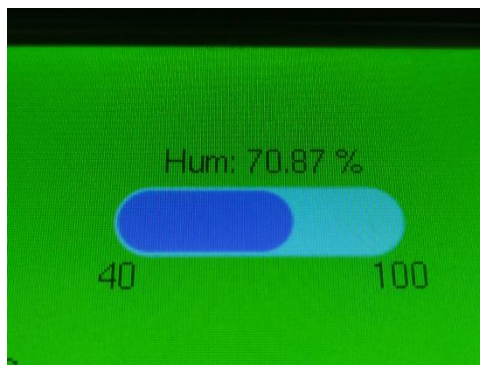
que se puede configurar tanto el color del fondo del gauge (en este caso negro), como el color de las marcas y de la aguja (en este caso verdes). Vemos que simplemente haciendo uso de la pequeña función creada ComGauge(), es fácil pintar un medidor tipo Gauge, siendo además fácil también configurar sus rangos de manera cómoda mediante la función map().

Para explicar al menos, alguna vez, el funcionamiento de map(), si nos ponemos en el primer caso: La presión medida está entre 990 y 1010 mbar, entonces básicamente como el gauge va a tener de rango de 0 a 20, mapeamos el valor de P_act, de manera que el rango 990 a 1010 se adapte al rango 0 a 20. Esto será bastante análogo para el resto de widgets, así que es probable que lo dé por explicado para no repetirlo mucho.

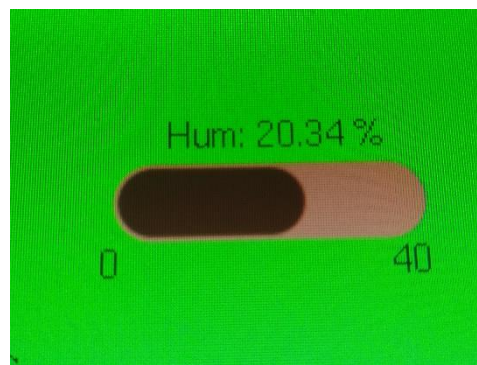
- **Medidor tipo Progress Bar para humedad**

Básicamente se pide plantear un medidor tipo barra de progreso para la medida de humedad, la cual se reescale entre 40 y 100 % si está por encima de 40% y entre 0 y 40% si está por debajo de dicho valor. Se ha añadido además un cierto cambio en el diseño que visualmente indica que la humedad es baja.

-Humedad > 40%



-Humedad < 40%



En este caso también se ha creado una función ComProgress(), basándose en la guía que queda así:

```
112// Función para pintar un Progress Bar(Barra de progreso), basada en el prototipo de la FTDI Programmer Guide
113void ComProgress( int16_t x,int16_t y,int16_t w,int16_t h,uint16_t options,uint16_t val,uint16_t range){
114    Escribiram32(CMD_PROGRESS);
115    Escribiram16(x);
116    Escribiram16(y);
117    Escribiram16(w);           //8 parametros de 16bits, 16 bytes: BIEN
118    Escribiram16(h);           //si fuesen 7 parametros de 16 bits, hay que mandar un ultimo Escribiram16 con ceros.
119    Escribiram16(options);      //En este caso sí hay que mandarle los ceros al final
120    Escribiram16(val);
121    Escribiram16(range);
122    Escribiram16(0);
123}
```

Vemos que es bastante similar a la función que pintaba el Gauge, salvo que ahora necesitamos rellenar con ceros al final, como se indica, el resto cambia poco.

En cuanto a su llamada, reescalado, y rediseño podemos verlo en la forma en que se llama a esta función dentro del while (1):

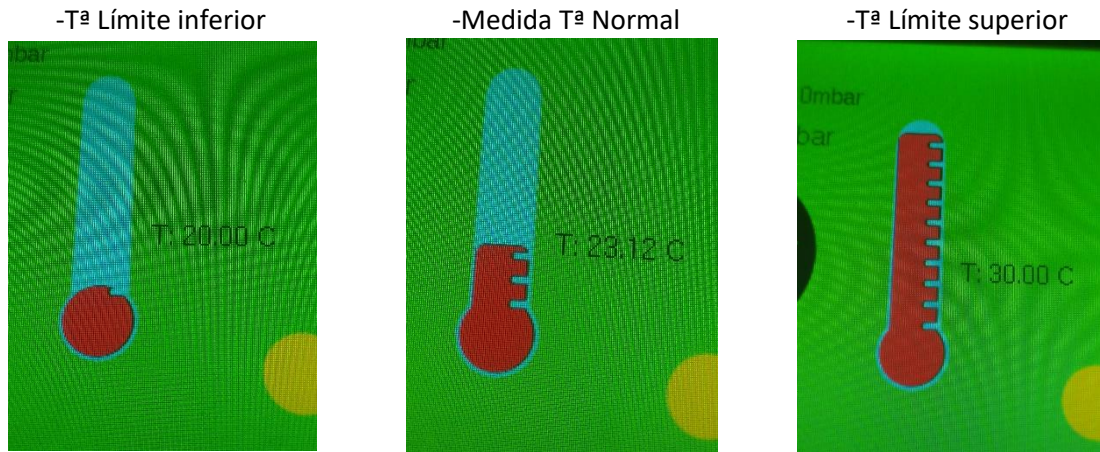
```
//Tras el Termómetro, vamos con la humedad (progress bar):
ComColor(0,0,0); //Elige color negro
//H_act = 20.34; //Para probar el cambio de color en el progressbar en el rango 0-40;
sprintf(string, "Hum: %.2f %%", H_act); //Guardamos string con la humedad medida
ComTXT(-30+3*HSIZE/4, VSIZE/4-20, 21, 0, string); //Pintamos la string con la humedad medida por encima de la progress bar
//Hacemos el reescalado según si humedad está en [0,40] o [40,100], además cambia el estilo de la progress bar
//Si humedad está en [40,100]:
if(H_act < 100 && H_act > 40){
    ComColor(148,84,246); //Elige color azul oscuro
    ComBgcolor(161,255,255); //Elige azul celeste
    ComProgress(-35+3*HSIZE/4, VSIZE/4, 90, 30, OPT_FLAT, map(H_act, 40, 100, 0, 100), 100); //Se pinta una progress bar con la hum
    ComColor(0,0,0); //Se elige color negro
    ComTXT(-55+3*HSIZE/4, VSIZE/4+30, 21, 0, "40"); //Se pinta valor mínimo del progress bar
    ComTXT(55+3*HSIZE/4, VSIZE/4+30, 21, 0, "100"); //Se pinta valor máximo del progress bar
}
//Si humedad está en [0,40]:
if(H_act < 40 && H_act > 0){
    ComColor(122,50,0); //Elige color naranja oscuro
    ComBgcolor(229,154,102); //Elige color naranja claro
    ComProgress(-35+3*HSIZE/4, VSIZE/4, 90, 30, OPT_FLAT, map(H_act, 0, 40, 0, 100), 100); //Se pinta una progress bar con la hum
    ComColor(0,0,0); //Se elige color negro
    ComTXT(-55+3*HSIZE/4, VSIZE/4+30, 21, 0, "0"); //Se pinta valor mínimo del progress bar
    ComTXT(55+3*HSIZE/4, VSIZE/4+30, 21, 0, "40"); //Se pinta valor máximo del progress bar
}
```

Vemos como el trabajo de crear el widget es bastante similar a lo realizado para el Gauge, de manera que se configuran el color de textos y demás y se muestra cierta información de interés e incluso los límites de la Progress Bar. En cuanto a la Progress Bar, vemos como, según el rango en el que se encuentre la humedad medida, la progress bar se pinta con colores diferentes como vimos en las fotos anteriores, así como se mapea también en los rangos correspondientes a cada caso, recorriendo en ambos el rango completo de la progress bar, en función del valor de H_act.

- Medidor tipo Termómetro para la temperatura

En este widget en concreto, el diseño es más artesanal, ya que no podemos basarnos en la guía que nos proporcione un CMD_ETC que pinte el widget fácilmente. Por tanto, hay que diseñar el termómetro manualmente, y así se ha hecho.

Muestro primero fotografías del resultado:



Vemos como incluso realizando el diseño a mano, queda un resultado medianamente decente para lo que sería la representación de un termómetro analógico.

```
96 // Función para pintar un Termómetro tipo analógico, se hace mediante círculos y rectángulos de colores dispuestos convenientemente
97 void ComTemp(float altura){ //Se le pasa como parámetro la altura del "líquido rojo" del termómetro que sube o baja según la temperatura
98   ComColor(0,255,224);
99   ComCirculo(HSIZE/2-HSIZE/9,VSIZ/2+VSIZ/5,20);
100   ComRect(HSIZE/2-HSIZE/7+5, VSIZ/4, HSIZE/2-HSIZE/9+10, VSIZ/2+VSIZ/5, true);
101   ComCirculo(HSIZE/2-HSIZE/7+15,VSIZ/4-3,14);
102   ComColor(255,0,0);
103   ComCirculo(HSIZE/2-HSIZE/9,VSIZ/2+VSIZ/5,18);
104   ComRect(HSIZE/2-HSIZE/7+7, altura, HSIZE/2-HSIZE/9+8, VSIZ/2+VSIZ/5, true);
105   for(j=0;j<=9;j++){ //Bucle para pintar las rayas de decoración del termómetro para que sea más estético
106     ComColor(0,255,224);
107     ComLine(HSIZE/2-HSIZE/7+22, (VSIZ/2+VSIZ/5-15)-j*((-(VSIZ/4-3)+(VSIZ/2+VSIZ/5))/10), HSIZE/2-HSIZE/9+12, (VSIZ/2+VSIZ/5-15)-j*((-(VSIZ/4-3)+(VSIZ/2+VSIZ/5))/10), 2);
108   }
109 }
110 }
```

Vemos como la creación del termómetro se basa en una serie de círculos y rectángulos primero de color celeste para obtener el recipiente en sí, y posteriormente un círculo rojo de líquido + barra vertical roja variable en función de la temperatura (vemos como se le pasa altura a la función como parámetro que afecta a la altitud del rectángulo rojo vertical). Destacar además el bucle for + ComLine, el cual origina esas “marcas” a la derecha del termómetro que son un elemento decorativo más que otra cosa.

```
//Tras el Gauge, vamos con el termómetro:
//T_act = 20.00;
//T_act = 30.00; //Para enseñar/comprobar los valores extremos
sprintf(string, "T: %.2f C", T_act); //Guardamos string cc
ComTXT(HSIZE/2+5,VSIZ/2, 21, OPT_CENTERX,string); //Pintamos
//Saturaciones; No afectan a la medida mostrada ya que antes, a
if(T_act<20) T_act = 20; //Saturamos el el valor i
if(T_act>30) T_act = 30; //Saturamos el el valor s
ComTemp(map(T_act,20,30,VSIZ/2+VSIZ/5,VSIZ/4-3)); //Cre
```

Vemos como simplemente se muestra la información de la medición por pantalla, tras mostrarla, se saturan los extremos de T_act, para que la barra roja vertical no “haga cosas raras”, como sobresalir por debajo o por encima.

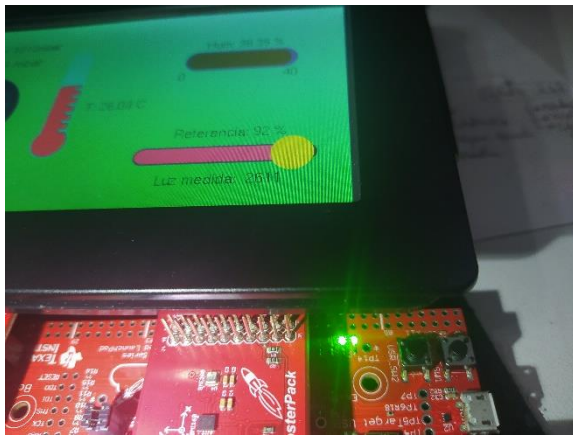
Finalmente, llamamos a ComTemp(), pasándole como información de “altura”, el mapeo de T_act, adaptando el rango 20-30 a las posiciones verticales mínima y máxima de la barra vertical, para que el termómetro pintado funcione como debe.

- Control deslizante para la luminosidad

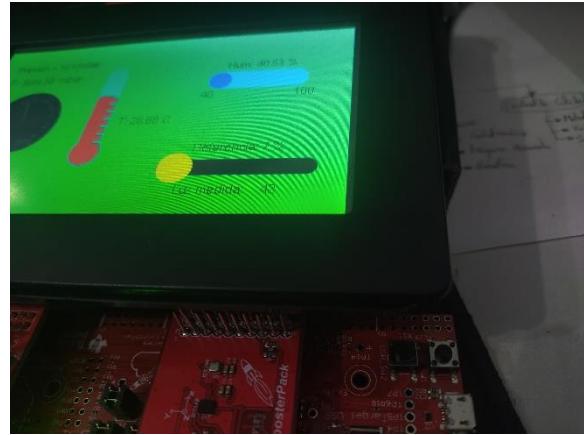
Se propone diseñar un control deslizante mediante el cual fijamos una referencia luminosa deseada, en % [0%,100%], de manera que, según la referencia luminosa requerida y la medición de luz ambiental, haremos un control sobre los leds de la placa, simulando lo que sería un sistema de iluminación que por si sólo pudiese adaptar la luminosidad medida a la referencia requerida. Obviamente esto no es posible con los leds de la placa, ya que no tienen potencia para modificar la luminosidad, pero como se ha dicho, podría ser útil si se cuenta con Leds potentes u otras fuentes de iluminación.

Dicho esto, se puede adjuntar algunas fotos de ciertos casos concretos para comprobar su funcionamiento (Hay muchas posibilidades), para posteriormente analizar cómo se ha desarrollado.

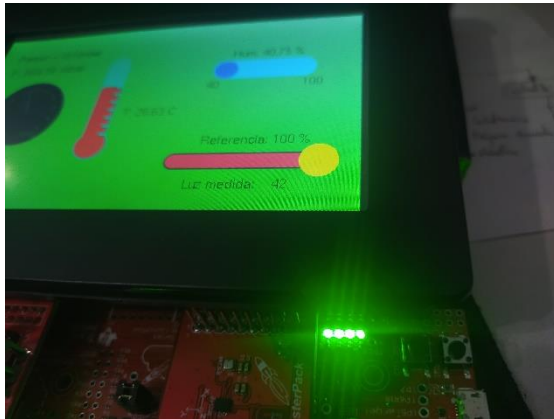
-Caso 1



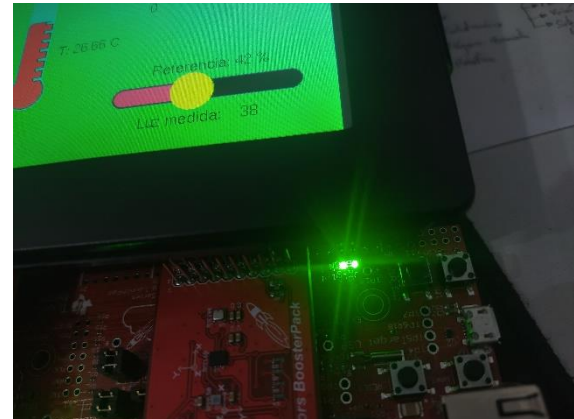
-Caso 2



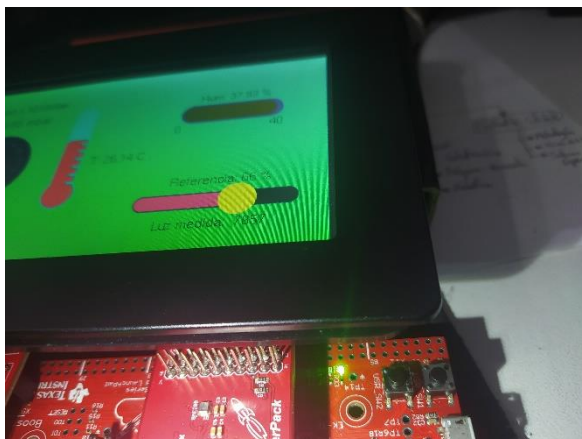
-Caso 3



-Caso 4



-Caso 5



Para comprender que caso es cada uno, vamos a acudir a la siguiente representación de la tabla de casos, con ciertos colores identificativos:

| | Referencia de luz | | | | |
|-------------|-------------------|---------------|---------------|---------------|----------------|
| Luz medida | Entre 0 y 19 | Entre 20 y 39 | Entre 40 y 59 | Entre 60 y 79 | Más de 80 |
| Lux<100 | Todo OFF | L1 | L1, L2 | L1, L2, L3 | L1, L2, L3, L4 |
| Lux<1000 | Todo OFF | Todo OFF | L1 | L1, L2 | L1, L2, L3 |
| Lux<10000 | Todo OFF | Todo OFF | Todo OFF | L1 | L1, L2 |
| Lux <40.000 | Todo OFF | Todo OFF | Todo OFF | Todo OFF | L1 |
| Lux >40.000 | Todo OFF | Todo OFF | Todo OFF | Todo OFF | Todo OFF |

Con esto vemos como el control funciona según la referencia dada con el Slider y la luz medida por el sensor.

Procedemos a la parte de código:

```

125// Función para pintar un Slider(Control deslizante), basada en el prototipo de la FTDI Programmer Guide
126void ComSlider( int16_t x,int16_t y,int16_t w,int16_t h,uint16_t options,uint16_t val,uint16_t range){
127    EscribeRam32(CMD_SLIDER);
128    EscribeRam16(x);
129    EscribeRam16(y);
130    EscribeRam16(w);           //8 parametros de 16bits, 16 bytes: BIEN
131    EscribeRam16(h);           //si fuesen 7 parametros de 16 bits, hay que mandar un ultimo EscribeRam16 con ceros.
132    EscribeRam16(options);      //En este caso si hay que mandarle los ceros al final
133    EscribeRam16(val);
134    EscribeRam16(range);
135    EscribeRam16(0);
136}
---
```

Poco que comentar de esta función, muy similar a las anteriores en las que ya disponíamos de un prototipo en la guía de FTDI.

```

138//Función para controlar los leds en función de la luz dada como referencia y de la luz medida mediante el sensor
139void Controla_Leds(int RefLux,int MedLux){
140    if(RefLux >= 0 && RefLux <= 19) enciende_leds(0); //Para esta ref da igual el valor de Lux medido, todos los leds apagados.
141
142    if(RefLux >= 20 && RefLux <= 39){ //Si la referencia está entre 20% y 39%:
143        if(MedLux < 100) enciende_leds(1); //Si la luz medida es menor a 100lux, enciende L1
144        else enciende_leds(0); //Si no apaga todos los leds
145
146    if(RefLux >= 40 && RefLux <= 59){ //Si la referencia está entre 40% y 59%:
147        if(MedLux < 100) enciende_leds(2); //Si la luz medida es menor a 100lux, enciende L1 y L2
148        else if(MedLux >= 100 && MedLux < 1000) enciende_leds(1); //Si la luz medida es mayor a 100lux y menor a 1000lux, enciende L1
149        else enciende_leds(0); //Si no apaga todos los leds
150
151    if(RefLux >= 60 && RefLux <= 79){ //Si la referencia está entre 60% y 79%:
152        if(MedLux < 100) enciende_leds(3); //Si la luz medida es menor a 100lux, enciende L1, L2 y L3
153        else if(MedLux >= 100 && MedLux < 1000) enciende_leds(2); //Si la luz medida es mayor a 100lux y menor a 1000lux, enciende L1 y L2
154        else if(MedLux >= 1000 && MedLux < 10000) enciende_leds(1); //Si la luz medida es mayor a 1000lux y menor a 10000lux, enciende L1
155        else enciende_leds(0); //Si no apaga todos los leds
156
157    if(RefLux >= 80){ //Si la referencia está por encima del 80%:
158        if(MedLux < 100) enciende_leds(4); //Si la luz medida es menor a 100lux, enciende L1, L2, L3 y L4
159        else if(MedLux >= 100 && MedLux < 1000) enciende_leds(3); //Si la luz medida es mayor a 100lux y menor a 1000lux, enciende L1, L2 y L3
160        else if(MedLux >= 1000 && MedLux < 10000) enciende_leds(2); //Si la luz medida es mayor a 1000lux y menor a 10000lux, enciende L1 y L2
161        else if(MedLux >= 10000 && MedLux < 40000) enciende_leds(1); //Si la luz medida es mayor a 10000lux y menor a 40000lux, enciende L1
162        else enciende_leds(0); //Si no apaga todos los leds
163    }
164}
```

En esta función por su parte, la llamaremos desde el while(1), pasándole como hemos dicho en la explicación anterior tanto la referencia dada mediante el Slider como la luminosidad medida por el OPT3001. Con esta información esta función encenderá los leds de la forma que convenga según las especificaciones de la tabla anterior.


```

if(Opt_OK)           //Si el sensor OPT3001 funciona correctamente:
{
    //Si se pulsa en la región del slider:
    if(POSX>(HSIZE-HSIZE/2) && POSX<(250+HSIZE-HSIZE/2) && POSY>(-50+VSIZE-VSIZE/4) && POSY<(70+VSIZE-VSIZE/4)){ //Se ha pulsado en la zona del slider
        referencialux = map(POSX,50+HSIZE-HSIZE/2,200+HSIZE-HSIZE/2,0,100); //El valor de referencia de la luz en % se mapea al rango 0 a 100
    }
    //Para captar mejor los extremos del Slider:
    if(POSX<50+HSIZE-HSIZE/2) referencialux = 0; //Si se pulsa un poco a la izquierda del extremo inicial del slider
    if(POSX>200+HSIZE-HSIZE/2) referencialux = 100; //Si se pulsa un poco a la derecha del extremo final del slider
}
ComColor(255,140,49); //Color del slider (Naranja claro) a la Izquierda de la bola
ComBgcolor(0,0,0); //Color del slider (Negro) a la Derecha de la bola
ComFgcolor(255, 255, 0); //Bola amarilla (Knob)
ComSlider( 50+HSIZE-HSIZE/2,VSIZE-VSIZE/4,150,20,OPT_FLAT,referencialux,100); //Se dibuja un slider en la parte derecha de la pantalla
ComColor(0,0,0); //Se elige color negro
sprintf(string, "Referencia: %d %%", referencialux); //Se pinta encima del slider el porcentaje de referencia
ComTXT(75+HSIZE-HSIZE/2,-25+VSIZE-VSIZE/4, 21, 0,string); //Se pinta encima del slider el porcentaje de referencia
ComColor(0,0,0);
sprintf(string, "Luz medida: %5.0f ", lux); //Se pinta debajo del slider la luz medida
ComTXT(65+HSIZE-HSIZE/2,30+VSIZE-VSIZE/4, 21, 0,string); //Se pinta debajo del slider la luz medida
Controla_Leds(referencialux,(int)lux); //Se llama a la función Controla_Leds()
}

```

Aquí vemos que, si el sensor OPT3001 funciona correctamente, tenemos una condición para en caso de que se pulse en la zona de la pantalla donde está pintado el Slider, se actualice el valor de referencialux, mapeando el valor de POSX (coordenada horizontal de la pulsación), adaptando el rango desde donde empieza el Slider a donde este termina, al rango 0 a 100 (ya que la referencia se expresa en %).

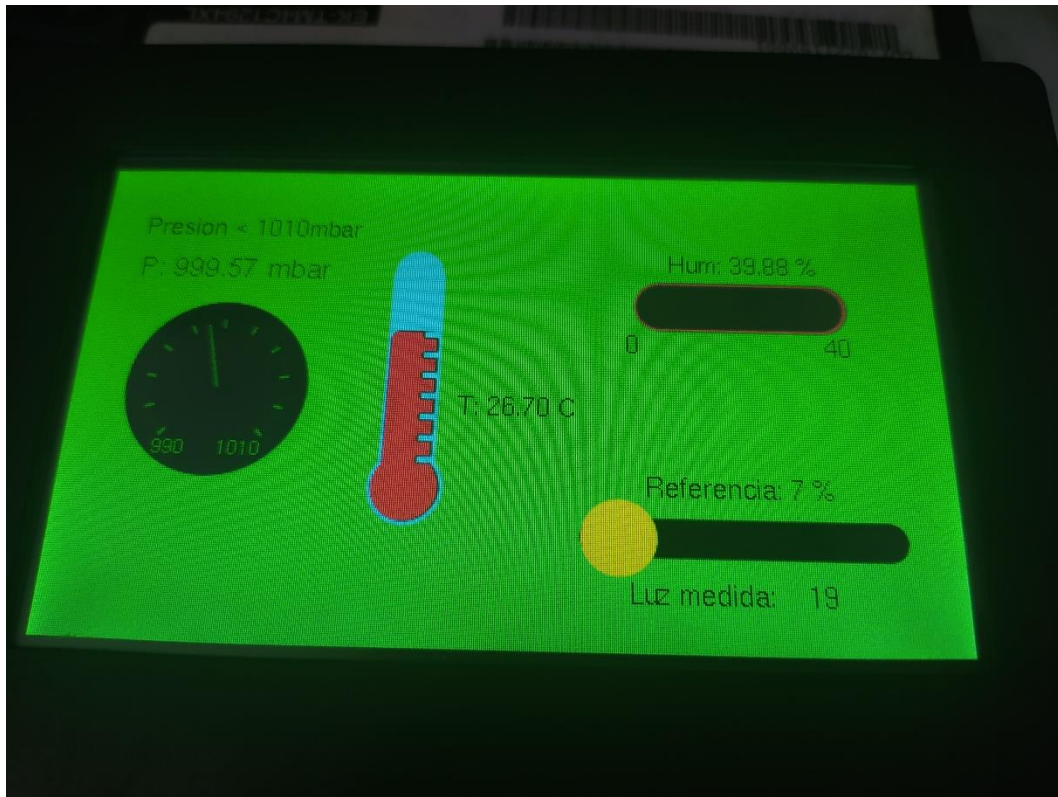
Se ha hecho un pequeño truco, ya que era problemático la detección de referencias extremas (<5% y >95%), de manera que si se pulsa algo más a la izquierda del slider se pone referencialux a 0 y lo mismo si se pulsa a la derecha se pone a 100, de esta manera logramos una sensación menos errática, sin estropear el rango del slider.

Destacar además que el refresco de la pantalla se realiza cada 250ms en lugar de cada 500, para mejorar la fluidez del movimiento del Slider, ya que se trababa bastante. Explicado esto, comentar que se vuelve a pintar cierto texto informativo mostrando tanto la referencia aportada con el slider como la luminosidad medida. Respecto a la llamada a ComSlider, es lo típico hasta ahora, los parámetros para posicionar el Slider, así como el valor mapeado de la referencia, guardado en referencialux, con rango 100 en el último parámetro para que vaya de 0 a 100 el rango del Slider.

Al final del todo tenemos la llamada a Controla_Leds(), donde le pasamos la referencialux y la luminosidad medida con el sensor, en este caso transformándola a entero por ser esto conveniente para la función de control.

- Interfaz Completa

Con esto quedan explicados los 4 widgets que aparecen en la interfaz, así que veo conveniente poner una imagen de la interfaz conjunta con un funcionamiento normal:



Tras haber comentado con cierto detenimiento cada uno de los Widgets, procedo a adjuntar el código íntegro en la memoria:

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>           //Declaración de librerías necesarias
#include <stdbool.h>
#include "driverlib2.h"
#include <stdlib.h>
#include <math.h>
#include "FT800_TIVA.h"
#include "HAL_I2C.h"
#include "sensorlib2.h"

// =====
//  VARIABLES PARA CONFIGURACIÓN Y TRABAJO DE LA PANTALLA
// =====
#define dword long
#define byte char

#define MSEC 40000 //Valor para 1ms con SysCtlDelay()

char chipid = 0;           // Holds value of Chip ID read from
the FT800
```

```

unsigned long cmdBufferRd = 0x00000000;           // Store the value read from
the REG_CMD_READ register
unsigned long cmdBufferWr = 0x00000000;           // Store the value read from
the REG_CMD_WRITE register
unsigned int t=0;
//
#####
#####
// User Application - Initialization of MCU / FT800 / Display
//
#####
#####
unsigned long POSX, POSY, BufferXY;
unsigned long POSYANT=0;                          //Variables y constantes
relacionadas con la configuración de la pantalla
unsigned int CMD_Offset = 0;
unsigned long REG_TT[6];
const int32_t REG_CAL[6]={21696, -78, -614558, 498, -17021, 15755638};
const int32_t REG_CAL5[6]={32146, -1428, -331110, -40, -18930, 18321010};

#define NUM_SSI_DATA          3
#define MaxEst 5

// =====
//  VARIABLES PARA CONFIGURACIÓN Y TRABAJO DE LOS SENSORES
// =====
char Cambia=0;
char string[80];
int DevID=0;
int RELOJ;

int LED[MaxEst][4]={                                //Matriz de estados
de los LEDS para actuar sobre ellos
    0,0,0,0,
    1,0,0,0,
    1,1,0,0,
    1,1,1,0,
    1,1,1,1,
};

// OPT3001
int lux_i;
float lux;
int referencialux;

// BME280 (Mide Temperatura, Presión y Humedad)
int returnRslt;                                     //se usa para ver si la función que
obtiene las medidas del BME280 devuelve éxito(0) o error(-1)
int g_s32ActualTemp = 0;
unsigned int g_u32ActualPress = 0;                  //Recogeremos aquí las medidas que
nos da la función bme280_read_pressure_temperature_humidity()
unsigned int g_u32ActualHumity = 0;
float T_act,P_act,H_act;                            //Tras calcular y formatear las
medidas, guardaremos aquí las medidas con el formato deseado
bool BME_on = true;                                  //Entiendo que este bool sería para
hacer lo que se hace con Bme_OK, pero no se usa.

uint8_t Sensor_OK=0;                                //Para comprobar si el acceso a I2C
correspondiente es exitoso o erróneo

```

```

#define BP 2 //Define por si se conecta en la posición
2 del boosterpack
uint8_t Opt_OK, Bme_OK; //Para registrar en variables si el
acceso a los sensores fue exitoso o no

int j;

void Timer0IntHandler(void); //Prototipos para las rutinas de
interrupción de los Timers

void enciende_leds(uint8_t Est) //Función para encender leds más
cómodamente
{
    GPIOWrite(GPIO_PORTN_BASE, GPIO_PIN_1, GPIO_PIN_1*LED[Est][0]);
    GPIOWrite(GPIO_PORTN_BASE, GPIO_PIN_0, GPIO_PIN_0*LED[Est][1]);
    GPIOWrite(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_PIN_4*LED[Est][2]);
    GPIOWrite(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_PIN_0*LED[Est][3]);
}

// Función para pintar un Gauge(Indicador de aguja tipo velocímetro), basada
en el prototipo de la FTDI Programmer Guide
void ComGauge( int16_t x,int16_t y,int16_t r,uint16_t options,uint16_t
major,uint16_t minor,uint16_t val,uint16_t range ){
    EscribirRam32(CMD_GAUGE);
    EscribirRam16(x);
    EscribirRam16(y); //8 parametros de 16bits, 16 bytes: BIEN
    EscribirRam16(r); //si fuesen 7 parametros de 16 bits, hay que
mandar un ultimo EscribirRam16 con ceros.
    EscribirRam16(options);
    EscribirRam16(major);
    EscribirRam16(minor);
    EscribirRam16(val);
    EscribirRam16(range);
}

// Función para pintar un Termómetro tipo analógico, se hace mediante
círculos y rectángulos de colores dispuestos convenientemente
void ComTemp(float altura){ //Se le pasa como parámetro la altura del
"líquido rojo" del termómetro que sube o baja según la temperatura
    ComColor(0,255,224);
    ComCirculo(HSIZE/2-HSIZE/9,VSIZ/2+VSIZ/5,20);
    ComRect(HSIZE/2-HSIZE/7+5, VSIZ/4, HSIZE/2-HSIZE/9+10, VSIZ/2+VSIZ/5,
true);
    ComCirculo(HSIZE/2-HSIZE/7+15,VSIZ/4-3,14);
    ComColor(255,0,0);
    ComCirculo(HSIZE/2-HSIZE/9,VSIZ/2+VSIZ/5,18);
    ComRect(HSIZE/2-HSIZE/7+7, altura, HSIZE/2-HSIZE/9+8, VSIZ/2+VSIZ/5,
true);
    for(j=0;j<=9;j++){ //Bucle para pintar las rayas de
decoración del termómetro para que sea más estético
        ComColor(0,255,224);
        ComLine(HSIZE/2-HSIZE/7+22, (VSIZ/2+VSIZ/5-15)-j*((-(VSIZ/4-
3)+(VSIZ/2+VSIZ/5))/10), HSIZE/2-HSIZE/9+12, (VSIZ/2+VSIZ/5-15)-j*((-
(VSIZ/4-3)+(VSIZ/2+VSIZ/5))/10), 2);
    }
}

```

```

// Función para pintar un Progress Bar(Barra de progreso), basada en el
prototipo de la FTDI Programmer Guide
void ComProgress( int16_t x,int16_t y,int16_t w,int16_t h,uint16_t
options,uint16_t val,uint16_t range){
    EscribeRam32(CMD_PROGRESS);
    EscribeRam16(x);
    EscribeRam16(y);
    EscribeRam16(w);           //8 parametros de 16bits, 16 bytes: BIEN
    EscribeRam16(h);           //si fuesen 7 parametros de 16 bits, hay que
mandar un ultimo EscribeRam16 con ceros.
    EscribeRam16(options);     //En este caso sí hay que mandarle los ceros
al final
    EscribeRam16(val);
    EscribeRam16(range);
    EscribeRam16(0);
}

// Función para pintar un Slider(Control deslizante), basada en el prototipo
de la FTDI Programmer Guide
void ComSlider( int16_t x,int16_t y,int16_t w,int16_t h,uint16_t
options,uint16_t val,uint16_t range){
    EscribeRam32(CMD_SLIDER);
    EscribeRam16(x);
    EscribeRam16(y);
    EscribeRam16(w);           //8 parametros de 16bits, 16 bytes: BIEN
    EscribeRam16(h);           //si fuesen 7 parametros de 16 bits, hay que
mandar un ultimo EscribeRam16 con ceros.
    EscribeRam16(options);     //En este caso sí hay que mandarle los ceros
al final
    EscribeRam16(val);
    EscribeRam16(range);
    EscribeRam16(0);
}

//Función para controlar los Leds en función de la luz dada como referencia y
de la luz medida mediante el sensor
void Controla_Leds(int RefLux,int MedLux){
    if(RefLux >= 0 && RefLux <= 19) enciende_leds(0); //Para esta ref da
igual el valor de Lux medido, todos los leds apagados.

    if(RefLux >= 20 && RefLux <= 39){ //Si la referencia está entre 20% y
39%:
        if(MedLux < 100) enciende_leds(1); //Si la luz medida es menor a
100lux, enciende L1
        else enciende_leds(0);} //Si no apaga todos los leds

    if(RefLux >= 40 && RefLux <= 59){ //Si la referencia está entre 40% y
59%:
        if(MedLux < 100) enciende_leds(2); //Si la luz medida es menor a
100lux, enciende L1 y L2
        else if(MedLux >= 100 && MedLux < 1000) enciende_leds(1); //Si la
luz medida es mayor a 100lux y menor a 1000lux, enciende L1
        else enciende_leds(0);} //Si no apaga todos los leds

    if(RefLux >= 60 && RefLux <= 79){ //Si la referencia está entre 60% y
79%:
        if(MedLux < 100) enciende_leds(3); //Si la luz medida es menor a
100lux, enciende L1, L2 y L3

```

```

        else if(MedLux >= 100 && MedLux < 1000) enciende_leds(2); //Si la
        luz medida es mayor a 100lux y menor a 1000lux, enciende L1 y L2
        else if(MedLux >= 1000 && MedLux < 10000) enciende_leds(1); //Si la
        luz medida es mayor a 1000lux y menor a 10000lux, enciende L1
        else enciende_leds(0);} //Si no apaga todos los leds

    if(RefLux >= 80){ //Si la referencia está por encima
del 80%:
        if(MedLux < 100) enciende_leds(4); //Si la luz medida es menor a
100lux, enciende L1, L2, L3 y L4
        else if(MedLux >= 100 && MedLux < 1000) enciende_leds(3); //Si
la luz medida es mayor a 100lux y menor a 1000lux, enciende L1, L2 y L3
        else if(MedLux >= 1000 && MedLux < 10000) enciende_leds(2); //Si
la luz medida es mayor a 1000lux y menor a 10000lux, enciende L1 y L2
        else if(MedLux >= 10000 && MedLux < 40000) enciende_leds(1); //Si
la luz medida es mayor a 10000lux y menor a 40000lux, enciende L1
        else enciende_leds(0);} //Si
no apaga todos los leds
    }

```

//Función para mapear el rango de las variables ambientales para que se adapten correctamente al rango funcional de los widgets desarrollados

```

double map(float valor, float entradaMin, float entradaMax, float salidaMin,
float salidaMax) //Destacar que mapeamos con valores tipo float para no
perder precisión

```

```

{
    return (((valor-entradaMin)*(salidaMax-salidaMin))/(entradaMax-
entradaMin))+salidaMin);
}

```

```

int main(void)

```

```

{
    int i;

    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);
    /***** CONF PANTALLA *****/
    HAL_Init_SPI(1, RELOJ); //Boosterpack a usar,
    Velocidad del MC
    Inicia_pantalla(); //Arranque de la pantalla
    //
    =====
    // Delay before we begin to display anything
    //
    =====
    SysCtlDelay(RELOJ/3);

    /***** TIMER0 *****/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); //Habilita T1
    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM); //T1 a 120MHz
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); //T1 periódico y
conjunto (32b)
    TimerLoadSet(TIMER0_BASE, TIMER_A, (RELOJ/4)-1); //Se carga el
timer1 con 250ms

```

```

    TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0IntHandler);           //Cada vez
    que cuente 250ms que entre en la interrupción y refresque el sistema
    IntEnable(INT_TIMER0A);                                           //Habilitar las
    interrupciones globales de los timers
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);                 //Habilitar las
    interrupciones de timeout
    IntMasterEnable();
    TimerEnable(TIMER0_BASE, TIMER_A);                               //Habilitar
    Timer0 (no hace falta recargarlo, se activa al principio y que cuente
    "infinito")

/*****
*****/

    /*****/
    Leds
    *****/

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);                    //Se habilita puerto F
    (LEDS)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);                    //Se habilita puerto N
    (LEDS)
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_4); //F0 y
    F4: salidas
    GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1); //N0 y
    N1: salidas

/*****
*****/

#ifdef VM800B35
    for(i=0;i<6;i++)      Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL[i]);
#endif
#ifdef VM800B50
    for(i=0;i<6;i++)      Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL5[i]);
#endif

/*****
*****/

    //Configuración Boosterpack según su conexión en la placa
    if(Detecta_BP(1)) Conf_Boosterpack(1, RELOJ);
    else if(Detecta_BP(2)) Conf_Boosterpack(2, RELOJ);
    else return 0;           //Si no detecta que se ha conectado en ninguna de
    las posiciones se acaba el programa

    //Inicializar OPT3001
    Sensor_OK=Test_I2C_Dir(OPT3001_SLAVE_ADDRESS);
    if(!Sensor_OK) Opt_OK=0;
    else{
        OPT3001_init();
        DevID=OPT3001_readDeviceId();
        Opt_OK=1;
    }

    //Inicializar BME280
    Sensor_OK=Test_I2C_Dir(BME280_I2C_ADDRESS2);
    if(!Sensor_OK) Bme_OK=0;
    else{
        bme280_data_readout_template();
        bme280_set_power_mode(BME280_NORMAL_MODE);
    }

```



```

    readI2C(BME280_I2C_ADDRESS2,BME280_CHIP_ID_REG, &DevID, 1);
    Bme_OK=1;
}

while(1){                                     //Bucle infinito repitiéndose

    if(Cambia==1){                             //Cada 250ms, el timer pone a 1 "Cambia" y
    ejecutamos este código
        Cambia=0;                             //Se resetea "Cambia" para que vuelva a
        esperar arriba a que el timer la active
        Nueva_pantalla(16,16,16);             //Se prepara una nueva
        pantalla
        Lee_pantalla();                       //Se lee la pantalla, en
        este caso para poder saber si estamos intentando actuar sobre el slider.
        ComColor(81,255,94);                 //Elige el color para el
        siguiente elemento, VERDE CLARO en este caso
        ComRect(0, 0, HSIZE, VSIZE, true);    //Se pinta todo
        el fondo de la pantalla de verde

        if(Opt_OK)                            //Si el sensor OPT3001 funciona
        correctamente:
        {
            lux=OPT3001_getLux();              //Obtenemos la medida de
            la luz en lux (aquí la guarda en float)
            lux_i=(int)round(lux);             //Guarda en otra variable
            la medida de lux convertida a entero por si interesa
        }
        if(Bme_OK)                            //Si el sensor BME280 funciona
        correctamente:
        {
            returnRslt =
            bme280_read_pressure_temperature_humidity(
            &g_u32ActualPress, &g_s32ActualTemp,
            &g_u32ActualHumity); //Recoge las medidas del BME280
            T_act=(float)g_s32ActualTemp/100.0;
            //Recogemos en formato float la temperatura medida
            P_act=(float)g_u32ActualPress/100.0;
            //Recogemos en formato float la presión medida
            H_act=(float)g_u32ActualHumity/1000.0;
            //Recogemos en formato float la humedad medida
        }
        if(Opt_OK)                            //Si el sensor OPT3001 funciona
        correctamente:
        {
            //Si se pulsa en la región del slider:
            if(POSX>(HSIZE-HSIZE/2) && POSX<(250+HSIZE-HSIZE/2)
            && POSY>(-50+VSIZE-VSIZE/4) && POSY<(70+VSIZE-VSIZE/4)){ //Se han dejado
            "márgenes" para pulsar alrededor de la slider para un funcionamiento más
            "fluido"

            referencialux = map(POSX,50+HSIZE-
            HSIZE/2,200+HSIZE-HSIZE/2,0,100); //El valor de referencia de la luz en %
            se mapea con la longitud desde donde empieza el slider a donde acaba a
            valores entre 0 y 100

            //Para captar mejor los extremos del Slider:
            if(POSX<50+HSIZE-HSIZE/2) referencialux = 0;
            //Si se pulsa un poco a la izquierda del extremo inicial del slider, la
            referencia será 0%

```



```

        if(POSX>200+HSIZE-HSIZE/2) referencialux = 100;
//Si se pulsa un poco a la derecha del extremo final del slider, la
referencia será 100%
    }
    ComColor(255,140,49);           //Color del slider
(Naranja claro) a la Izquierda de la bola
    ComBgcolor(0,0,0);             //Color del slider
(Negro) a la Derecha de la bola
    ComFgcolor(255, 255, 0);       //Bola amarilla (Knob)
    ComSlider( 50+HSIZE-HSIZE/2,VSIZE-
VSIZE/4,150,20,OPT_FLAT,referencialux,100); //Se dibuja un slider en la
parte derecha y abajo de la pantalla, el cual se moverá al coger los valores
de referencialux entre 0 y 100 según se haya pulsado el "Knob"
    ComColor(0,0,0);
//Se elige color negro
    sprintf(string, "Referencia: %d %%", referencialux);
    ComTXT(75+HSIZE-HSIZE/2,-25+VSIZE-VSIZE/4, 21,
0,string); //Se pinta encima del slider el porcentaje
de referencia que corresponde al desplazamiento del knob realizado
    ComColor(0,0,0);
    sprintf(string, "Luz medida: %5.0f ", lux);
    ComTXT(65+HSIZE-HSIZE/2,30+VSIZE-VSIZE/4, 21,
0,string); //Se pinta debajo del slider la luz medida
por el sensor en [lux]
    Controla_Leds(referencialux,(int)lux);
//Se llama a la función Controla_Leds(), pasándole tanto la referencia
establecida con el slider en [%] como la medida de luz en [lux] en formato
entero, de manera que encenderá los leds como convenga
    }
    if(Bme_OK) //Si el sensor BME280 funciona
correctamente:
    {
        //Se ha notado que en Olivares (mi pueblo, la
presión oscila entre 990 y 1010mbar)(a veces lo supera)
        //Luego según si está probándose en Sevilla (que
tiene más presión) o Olivares,
        //He decidido reescalar el gauge en el caso de
que la presión suba o baje por encima de 1010
        //Ya que si no el gauge al salirse de los límites
hace "cosas raras".
        //El Gauge se reescala entre 990 y 1010 si la
presión baja de 1010 y 1010 y 1030 si sube de 1010mbar
        // P_act = 1024.76; //probar si reescala el Gauge
entre "Presion>1010mbar" y "Presion<1010mbar" //comentar o descomentar

        if(P_act >= 990 && P_act <= 1010){ //Reescalado
del Gauge para Rango entre 990 y 1010 si Presión medida es inferior a 1010
mbar
            ComColor(65,202,42);           //Color verde
"fondo" para aguja y marcadores del gauge
            ComBgcolor(0,0,0);             //Color negro
para el fondo del gauge

    ComGauge(HSIZE/6,VSIZE/2,50,OPT_FLAT,11,5,map(P_act,990,1010,0,20),20);
//Se crea el gauge con la función anterior, mapeando los valores de presión
entre 990 y 1010mbar al rango 0-20, para que se recorra bien el gauge según
presiones medidas entre 990 y 1010

```

```

        sprintf(string, "P: %.2f mbar", P_act);
//Guardamos string con la presión medida
        ComColor(65,202,42); //Color verde
        "fondo" para números 990 y 1010, indicadores mínimo y máximo del rango del
        gauge en este caso
        //ComBgcolor(0,0,0);
        ComTXT(50,VSIZ/5+110, 20, 0,"990");
//Pintamos los indicadores mínimo y máximo del rango del gauge en este caso
        ComTXT(25+60,VSIZ/5+110, 20, 0,"1010");
        ComColor(0,0,0);

//Elige color negro
        ComTXT(25,-25+VSIZ/5, 21, 0,"Presion <
1010mbar"); //Indicamos por encima de la medida de la presión que estamos
en el caso escalado a presión_medida < 1010 mbar
        ComTXT(25,VSIZ/5, 22, 0,string);
//Pintamos el string con la presión medida
    }
    if(P_act >= 1010 && P_act <= 1030){ //Reescalado
del Gauge para Rango entre 1010 y 1030 si Presión medida es superior a 1010
mbar
        ComColor(65,202,42); //Color verde
        "fondo" para aguja y marcadores del gauge
        ComBgcolor(0,0,0); //Color negro
para el fondo del gauge

        ComGauge(HSIZ/6,VSIZ/2,50,OPT_FLAT,11,5,map(P_act,1010,1030,0,20),20);
//Se crea el gauge con la función anterior, mapeando los valores de presión
entre 1010 y 1030mbar al rango 0-20, para que se recorra bien el gauge según
presiones medidas entre 1010 y 1030
        sprintf(string, "P: %.2fmbar", P_act);
//Guardamos string con la presión medida
        ComColor(65,202,42);
//Color verde "fondo" para números 1010 y 1030, indicadores mínimo y máximo
del rango del gauge en este caso
        //ComBgcolor(0,0,0);
        ComTXT(50,VSIZ/5+110, 20, 0,"1010");
//Pintamos los indicadores mínimo y máximo del rango del gauge en este caso
        ComTXT(25+60,VSIZ/5+110, 20, 0,"1030");
        ComColor(0,0,0);

//Elige color negro
        ComTXT(25,-25+VSIZ/5, 21, 0,"Presion >
1010mbar"); //Indicamos por encima de la medida de la presión que estamos
en el caso escalado a presión_medida > 1010 mbar
        ComTXT(25,VSIZ/5, 22, 0,string);
//Pintamos el string con la presión medida
    }

    //Tras el Gauge, vamos con el termómetro:
    //T_act = 20.00;
    //T_act = 30.00; //Para enseñar/comprobar los
valores extremos del termómetro
        sprintf(string, "T: %.2f C", T_act);
//Guardamos string con la presión medida
        ComTXT(HSIZ/2+5,VSIZ/2, 21, OPT_CENTERX,string);
//Pintamos la string con la medida de la temperatura a la derecha del
termómetro

        //Saturaciones; No afectan a la medida mostrada ya
que antes, al mostrarla, no está saturada

```

```

        if(T_act<20) T_act = 20; //Saturamos
el el valor inferior de la temperatura a 20 tras mostrarla para que la "barra
roja" no baje hacia abajo del termómetro
        if(T_act>30) T_act = 30; //Saturamos
el el valor superior de la temperatura a 30 tras mostrarla para que la "barra
roja" no sobresalga por encima del termómetro
        ComTemp(map(T_act,20,30,VSIZ/2+VSIZ/5,VSIZ/4-
3)); //Creamos el termómetro, pasándole la altura, que corresponde al
valor de la Temperatura medida, mapeando el rango entre 20 y 30, a las
alturas mínimas y máximas dentro del termómetro

        //Tras el Termómetro, vamos con la humedad
(progress bar):
        ComColor(0,0,0);
//Elige color negro
        //H_act = 20.34; //Para probar el cambio de
color en el progressbar en el rango 0-40;
        sprintf(string, "Hum: %.2f %%", H_act);
//Guardamos string con la humedad medida
        ComTXT(-30+3*HSIZE/4,VSIZ/4-20, 21, 0,string);
//Pintamos la string con la humedad medida por encima de la progress bar
        //Hacemos el reescalado según si humedad está en
[0,40] o [40,100], además cambia el estilo de la progress bar
        //Si humedad está en [40,100]:
        if(H_act < 100 && H_act > 40){
color azul oscuro
        ComColor(148,84,246); //Elige
azul celeste
        ComBgcolor(161,255,255); //Elige
        ComProgress(-
35+3*HSIZE/4,VSIZ/4,90,30,OPT_FLAT,map(H_act,40,100,0,100),100); //Se
pinta una progress bar con la humedad mapeada desde el rango [40,100] al
rango [0,100], para que se recorra todo el progress bar (de 0 a 100)
        ComColor(0,0,0);
//Se elige color negro
        ComTXT(-55+3*HSIZE/4,VSIZ/4+30, 21, 0,"40");
//Se pinta valor mínimo del progress bar
        ComTXT(55+3*HSIZE/4,VSIZ/4+30, 21, 0,"100");
//Se pinta valor máximo del progress bar
        }
        //Si humedad está en [0,40]:
        if(H_act < 40 && H_act > 0){
        ComColor(122,50,0);
//Elige color naranja oscuro
        ComBgcolor(229,154,102);
//Elige color naranja claro
        ComProgress(-
35+3*HSIZE/4,VSIZ/4,90,30,OPT_FLAT,map(H_act,0,40,0,100),100); //Se
pinta una progress bar con la humedad mapeada desde el rango [0,40] al rango
[0,100], para que se recorra todo el progress bar (de 0 a 100)
        ComColor(0,0,0);
//Se elige color negro
        ComTXT(-55+3*HSIZE/4,VSIZ/4+30, 21, 0,"0");
//Se pinta valor mínimo del progress bar
        ComTXT(55+3*HSIZE/4,VSIZ/4+30, 21, 0,"40");
//Se pinta valor máximo del progress bar
        }
    }
}

```

```

        Dibuja();                                //Se pintan en la pantalla todos los
elementos anteriores                            }
    }

    return 0;

}

void Timer0IntHandler(void)                      //Rutina de interrupción del Timer0
{
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag de la
interrupción
    Cambia=1;                                    //Vuelve a activar la
variable "Cambia" para que se actualice la pantalla y las medidas 4 veces por
segundo
    SysCtlDelay(100);                            //Pequeño delay por
"bug" conocido e indicado por el manual
}

```

En este caso, habiendo comentado los widgets aisladamente y con el necesario detenimiento, y dado la similitud de este programa con el anterior respecto a la impresión de elementos por pantalla considero que es repetitivo analizar de nuevo todo el código como suelo hacer.

Simplemente recalcar que las funciones diseñadas para cada widget están ubicadas antes del main() y posteriormente son llamadas dentro del while(1), y siendo configuradas previamente a su llamada para que se pinten los widgets de la manera deseada.

Con esto termina esta memoria y recomiendo también como de costumbre, acudir a los vídeos explicativos, aunque en este caso sí que considero que las imágenes son bastante explicativas por sí solas.