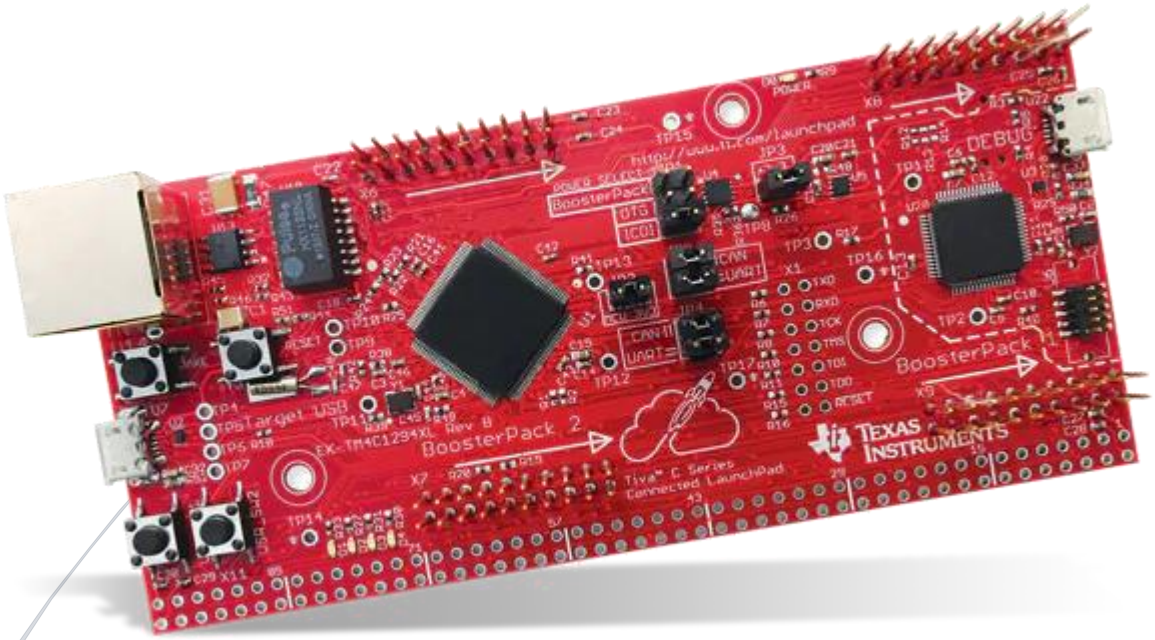
4ºGIERM

Práctica 4 SEPA:

Monitorización y control de variables ambientales



Daniel Peinado Ramírez, Francisco Javier Román Escorza
SISTEMAS ELECTRÓNICOS PARA LA AUTOMATIZACIÓN

1. Primer Ejercicio: Sensor de magnitudes ambientales con interfaz digital.

Partiendo del ejemplo suministrado (Ejemplo 8), realizar un programa que muestree la temperatura, presión y humedad relativa del BME280 así como la luminosidad del OPT3001. Dichas variables se actualizarán 2 veces por segundo, redibujando la pantalla. Para la primera versión, en la pantalla VM800 se mostrarán únicamente mensajes de texto con el valor actual, cambiando el color del texto (y del fondo del mismo, si se considera conveniente) en función de que esté dentro o fuera de unos márgenes:

Magnitud/Color	Azul	Blanco	Rojo
Temperatura	T<20	20<T<25	T>25
Presión	P<1000	1000<P<1015	P>1015
Humedad relativa	H<30%	30%<H<60%	H>60%
Luz	Lux<100	100<Lux<1000	Lux>1000

En el primer ejercicio procedemos a realizar un medidor de temperatura, presión, humedad y luz digital gracias al uso del *"Sensors Booster Pack"* en la ranura 2 de la placa. Para realizar las mediciones de cada una de las variables, hemos utilizado como base el código del ejemplo 8. En nuestro caso, no utilizaremos los sensores T_amb y T_obj, debido a que nuestra placa no los dispone. Tampoco haremos uso del acelerómetro; además desechamos todo aquello relacionado con la configuración de la UART y la representación de caracteres por puerto serie. En vez de mostrar los datos por puerto serie como en el ejemplo, utilizaremos la pantalla FT800 que ya hemos utilizado varias veces.

El texto para indicar cada una de las mediciones aparece centrado en la pantalla y en filas equiespaciadas entre sí. Se ha dividido la longitud vertical entre 9 y colocando cada una de las mediciones separadas 2/9 de la longitud de la pantalla, comenzando en 1,5 y acabando en 7,5.

Para determinar el color del texto por pantalla, establecemos la condición de que si la variable en cuestión supera el umbral máximo que aparece en la tabla, se escribirá en color rojo; si es menor que el umbral mínimo, pasará a ser azul y que en cualquier otro caso (es decir, que se encuentre entre el mínimo y el máximo) será blanco.

- ***Código:***

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <math.h>
#include "driverlib2.h"
#include "utils/uartstdio.h"
#include "HAL_I2C.h"
#include "sensorlib2.h"
#include "FT800_TIVA.h"

//Definimos cada uno de los colores en que vamos a utilizar, en formato
RGB
#define gris 60,60,60
#define rojo 255,0,0
```

```
#define azul 0,0,255
#define blanco 255,255,255

//Divisiones del ancho y largo de la pantalla
#define divHor HSIZE/2
#define divVer VSIZE/9
//Definición de cadenas de texto

//
================================================================================
// Function Declarations
//
================================================================================
#define dword long
#define byte char

#define PosMin 750
#define PosMax 1000

#define XpMax 286
#define XpMin 224
#define YpMax 186
#define YpMin 54

unsigned int Yp=120, Xp=245;
//
================================================================================
// Variable Declarations
//
================================================================================

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;
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

int RELOJ, Flag_ints;
```

```
//Función del SLEEP fake, utilizar en caso de que dé problemas al usar
el debugger
//#define SLEEP SysCtlSleep()
#define SLEEP SysCtlSleepFake()

void SysCtlSleepFake(void)
{
 while(!Flag_ints);
 Flag_ints=0;
}
//Creamos cadenas para cada una de las variables que vamos a medir
void Timer0IntHandler(void);

char temp[]= "Temperatura = 000.00ºC";
char pres[]= "Presion = 000000 mbar";
char hum[]= "Humedad = 00.00 0/0";
char luz[]= "Luz = 0000 lux";

char Cambia=0;

float lux;
char string[80];
int DevID=0;

int16_t T_amb, T_obj;

 float Tf_obj, Tf_amb;
 int lux_i, T_amb_i, T_obj_i;

 // BME280
 int returnRslt;
 int g_s32ActualTemp   = 0;
 unsigned int g_u32ActualPress  = 0;
 unsigned int g_u32ActualHumity = 0;
// struct bme280_t bme280;

 // BMI160/BMM150
 int8_t returnValue;
 struct bmi160_gyro_t        s_gyroXYZ;
 struct bmi160_accel_t       s_accelXYZ;
 struct bmi160_mag_xyz_s32_t s_magcompXYZ;


 //Calibration off-sets
 int8_t accel_off_x;
 int8_t accel_off_y;
 int8_t accel_off_z;
 int16_t gyro_off_x;
 int16_t gyro_off_y;
 int16_t gyro_off_z;
 float T_act,P_act,H_act;
 bool BME_on = true;

 int T_uncomp,T_comp;
char mode;
long int inicio, tiempo;

volatile long int ticks=0;
uint8_t Sensor_OK=0;
```

```
#define BP 2
uint8_t Opt_OK, Tmp_OK, Bme_OK, Bmi_OK;

void IntTick(void){
    ticks++;
}
int main(void) {

    //Configuración del Timer0
    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ/2 -1);
    TimerIntRegister(TIMER0_BASE, TIMER_A,Timer0IntHandler);
    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable();
    TimerEnable(TIMER0_BASE, TIMER_A);

    HAL_Init_SPI(1, RELOJ);  //Boosterpack a usar para la pantalla (1),
Velocidad del MC
    Inicia_pantalla();       //Arranque de la pantalla

    SysCtlDelay(RELOJ/3);

    if(Detecta_BP(1))     Conf_Boosterpack(1, RELOJ);
    else if(Detecta_BP(2))   Conf_Boosterpack(2, RELOJ);
    else     return 0;

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

     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;
     }
    SysTickIntRegister(IntTick);
    SysTickPeriodSet(12000);
    SysTickIntEnable();
    SysTickEnable();

    while(1)
    {
        SLEEP;

            if(Opt_OK)
```

```
            {//Luz
                lux=OPT3001_getLux();
                lux_i=(int)round(lux);
            }

            if(Bme_OK)
            {//Cálculo de la temperatura, presión y humedad
                returnRslt =
bme280_read_pressure_temperature_humidity(&g_u32ActualPress,
&g_s32ActualTemp, &g_u32ActualHumity);
                T_act=(float)g_s32ActualTemp/100.0;
                P_act=(float)g_u32ActualPress/100.0;
                H_act=(float)g_u32ActualHumity/1000.0;
            }

        sprintf(temp,"Temperatura = %.2f C",T_act);//Reescribimos la
temperatura, con 2 decimales según la  medida cada ciclo de reloj
        sprintf(pres, "Presion = %.1f mbar",P_act);//Reescribimos la
presión, con 1 decimal según la medida
        sprintf(hum,"Humedad =  %.1f %c",H_act,37);//Reescribimos la
humedad, con 1 decimal según la  medida
        sprintf(luz, "Luz = %d lux",lux_i); //Reescribimos la intensidad
luminosa, sin decimales según la medida
        Lee_pantalla();
        Nueva_pantalla(gris);

        //Decidir el color de T
        if (T_act<20)        ComColor(azul); //Si la temperatura es
menor a 20ºC: azul
        else if (T_act>25)   ComColor(rojo); //Si la temperatura es
mayor a 30ºC: rojo
        else                 ComColor(blanco); //Entre ambas
temperaturas: blanco
        ComTXT(divHor,1.5*divVer, 29, OPT_CENTER, temp); //Texto
centrado, 1º fila

        //Decidir el color de P
        if (P_act<1000)        ComColor(azul); //Si la presión es menor
a 1000 mbar: azul
        else if (P_act>1015)   ComColor(rojo); //Si la presión es menor
a 1015 mbar: rojo
        else                 ComColor(blanco); //Entre ambas presiones:
blanco
        ComTXT(divHor,3.5*divVer, 29, OPT_CENTER, pres); //Texto
centrado, 2º fila

        //Decidir el color de H
        if (H_act<30)        ComColor(azul); //Si la humedad es menor al
30%: azul
        else if (H_act>60)   ComColor(rojo); //Si la humedad es mayor al
60%: azul
        else                 ComColor(blanco); //Entre ambas, blanco
        ComTXT(divHor,5.5*divVer, 29, OPT_CENTER, hum); //Texto
centrado, 3º fila

        //Decidir el color de L
        if (lux_i<100)        ComColor(azul); //Si la luz es menor a 100
lux: azul
```

```
        else if (lux_i>1000)   ComColor(rojo); //Si la luz es mayor a
1000 lux: azul
        else                   ComColor(blanco); //Entre ambas, blanco
        ComTXT(divHor,7.5*divVer, 29, OPT_CENTER, luz); //Texto
centrado, 4º fila

        Dibuja();//Representar en pantalla lo anterior
 }

}
//Función de interrupción del Timer
void Timer0IntHandler(void)
{
   TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag
    Flag_ints=1;
   Cambia=1;
}
```

-

2. Segundo Ejercicio: Sensor de magnitudes ambientales con interfaz analógico y control fuzzy de luz.

Una vez funcionando este caso, se desea realizar un interfaz con estilo analógico, de manera que se tenga:

- ***un medidor tipo Gauge que señale la presión atmosférica, teniendo como rango entre 1000 y 1030 mbar.***
- ***Una barra tipo Progress Bar que visualice la humedad relativa (en %, con 2 decimales, y ajustando la escala entre 40% y 100%. Si la humedad estuviese por debajo del 40%, reajustar la barra con la escala de 0 a 40%.***
- ***La temperatura se mostrará como una barra vertical, llenada de manera proporcional y mostrando al lado del nivel el valor de la temperatura. La barra tendrá como valores extremos 20º y 30º. Se puede "decorar" para hacerla parecer un termómetro.***
- ***Un control deslizante que servirá para fijar el valor de luminosidad requerido, entre 0 y 100%. Este valor deseado se comparará con la luz medida con el medidor de luz y se encenderán una serie de leds en la placa, dependiendo del caso, según la tabla siguiente:***

	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

Para el segundo ejercicio, realmente procedemos de la misma forma para realizar las mediciones de cada una de las variables, lo que realmente va a cambiar es la forma en la que representamos dichos valores por pantalla, así como la activación de los leds según corresponda.

Con una serie de funciones, y pasándole los parámetros necesarios, podemos representar en la pantalla gráficamente el termómetro, el sensor de humedad, el barómetro y el sensor de luz de forma analógica.

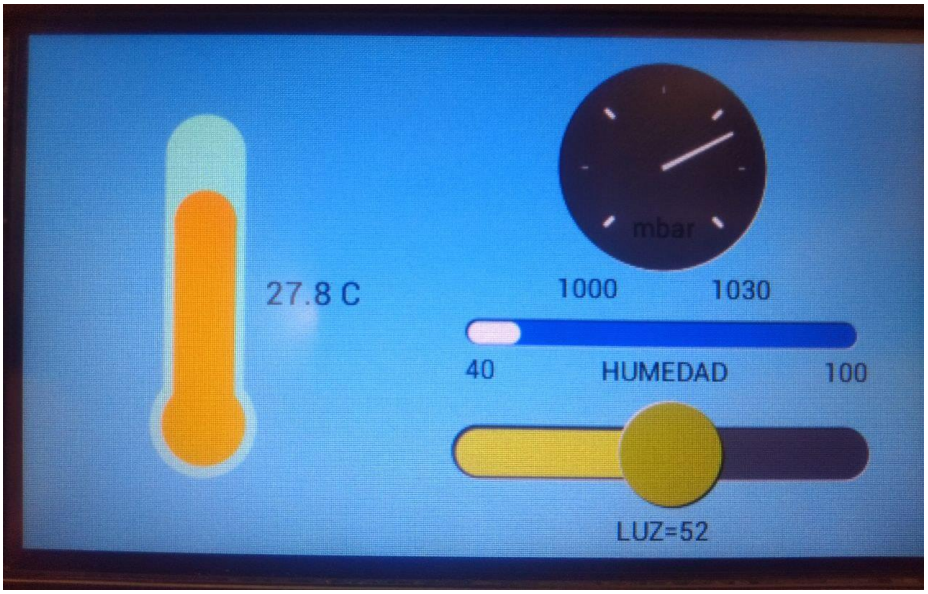


Ilustración 1: Representación en la pantalla

Principalmente utilizaremos 4 funciones para representar cada una de las variables.

- Gauge: Lo utilizamos para representar la presión. El medidor oscila entre 1000 y 1030 mbar. En su función, especificamos la posición x, y, el radio de la circunferencia del medidor, el efecto 3D (en caso de que se utilice, entre otras opciones gráficas), el número de marcas gruesas y finas del medidor y los valores mínimo y máximo.
- Slider: En función de la posición de la bola y la luz que recibe el sensor, enciende o apaga los 4 leds según las especificaciones de la tabla del enunciado. Le pasamos las posiciones, x, y, ancho de la bola, de la barra, opciones de diseño como con el gauge, el valor de la luz y el rango (desde 0 hasta el valor máximo).
- Termómetro: Representa la temperatura. A partir de las posiciones x, y, la altura a la que llegará el mercurio de la temperatura, el radio de la parte circular el valor de la temperatura medido, la temperatura máxima y la mínima, la función calcula la temperatura y la representa en función de los datos proporcionados. Los valores mínimo y máximo son 20 y 30 respectivamente. El color del mercurio pasará a rojo si superamos la temperatura máxima, a azul si pasamos por debajo de ella, o a naranja si estamos comprendidos en el rango.
- Progress bar: Representa la humedad. Está comprendida entre los valores 40 y 100. La función recibe prácticamente los mismos parámetros que el slider de la luz, pero en este caso el valor es el de la humedad. En el caso de que sea menor al 40%, la barra cambiará a otra que representa el porcentaje de humedad entre 0 y 40.
- Para encender los leds hemos definido matrices de cada una de las posibilidades según el estado del slider, y en función de la luz recibida, se encenderán unos leds u otros.

- ***Código:***

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <math.h>
#include "driverlib2.h"
#include "utils/uartstdio.h"
#include "HAL_I2C.h"
#include "sensorlib2.h"
#include "FT800_TIVA.h"

//Definición de colores
#define blanco 240,240,240
#define gris 60,60,60
#define rojo 255,0,0
#define azul 31,75,171
#define fondo 111, 186, 209
#define negro 0,0,0
#define amarillo 199,204,51
#define amarillo_oscuro 144,148,13

//Definición de los valores máximos y mínimos para medidas de sensores
#define Gauge_max 1030
#define Gauge_min 1000
#define Temp_max 30
#define Temp_min 20
#define Hum_min 0
#define Hum_med 40
#define Hum_max 100
#define Luz_min 0
#define Luz_max 100

// Definiciones de la pantalla
#define dword long
#define byte char

#define PosMin 750
#define PosMax 1000

#define XpMax 286
#define XpMin 224
#define YpMax 186
#define YpMin 54

unsigned int Yp=120, Xp=245;

// Variables usadas en la pantalla
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;
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};

//#####################################################################
//Definición de las funciones de los WIDGETS
//#####################################################################

//GAUGE
void ComGauge( int16_t x, int16_t y, int16_t r, uint16_t options,
uint16_t major, uint16_t minor,  uint16_t val, int16_t min, uint16_t
max)
{
    ComColor(blanco);                    //Color aguja y major y minor
    ComBgcolor(gris);                   //Color de la esfera del reloj
    //Sacar el gauge predeterminado
    EscribeRam32(CMD_GAUGE);
    EscribeRam16(x);
    EscribeRam16(y);
    EscribeRam16(r);
    EscribeRam16(options);
    EscribeRam16(major);
    EscribeRam16(minor);
    EscribeRam16(val-min);
    EscribeRam16(max-min);                  //corresponde a range del
CMD_GAUGE

    //Texto
    ComColor(negro);
    char c[5];
    sprintf(c, "%d", max);
    ComTXT(x+r*3/4, y+r*6/5, 26, OPT_CENTER, c);
    sprintf(c, "%d", min);
    ComTXT(x-r*3/4, y+r*6/5, 26, OPT_CENTER, c);
    sprintf(c, "mbar");
    ComTXT(x, y+r*3/5, 26, OPT_CENTER, c);
}

//PROGRESS BAR
//referencia->  pag 191 "FT800 Series ProgrammerGuide"

void ComProgress(int16_t x, int16_t y, int16_t w, int16_t h, int16_t
options, int16_t val, int16_t min, int16_t mitad, int16_t max)
{
    char c[8];
    ComColor(negro);
    sprintf(c,"HUMEDAD");
    ComTXT(x+w/2, y+2*h, 26, OPT_CENTER, c);
    int range, value;
    //División del rango total en 2
```

```
        if (val>=mitad) {
            //Reasignación del rango y el valor dentro del subrango
            range=max-mitad;
            value=val-mitad;
            //Impresión de valor mínimo y maximo del subrango
            sprintf(c,"%d", mitad);
            ComTXT(x, y+2*h, 26, OPT_CENTER, c);
            sprintf(c,"%d", max);
            ComTXT(x+w, y+2*h, 26, OPT_CENTER, c);
            }
        else {
            //Reasignación del rango y el valor dentro del subrango
            range=mitad-min;
            value=val;
            //Impresión de valor mínimo y maximo del subrango
            sprintf(c,"%d", min);
            ComTXT(x, y+2*h, 26, OPT_CENTER, c);
            sprintf(c,"%d", mitad);
            ComTXT(x+w, y+2*h, 26, OPT_CENTER, c);
            }
        ComBgcolor(azul);
        ComColor(blanco);
        EscribeRam32(CMD_PROGRESS);
        EscribeRam16(x);
        EscribeRam16(y);
        EscribeRam16(w);
        EscribeRam16(h);
        EscribeRam16(options);
        EscribeRam16(value);
        EscribeRam16(range);
        EscribeRam16(0);


    }

    //TERMÓMETRO
    void ComTermometro(int x, int y, int H, int R,float val, int min, int
    max)
    {
        int r = 0.6*R;                  //Radio menor
        int r_int=0.4*R;                //Tamaño principal interior
        int hval=H*(val-min)/(max-min); //altura de la temperatura

        //Fondo
        ComColor(149,254,232);          //Azulito para el fondo
        ComLineWidth(0);
        ComCirculo(x,y,R);
        ComRect(x-r, y-H, x+r, y, 1);
        ComCirculo(x, y-H-80/16, r+80/16);

        //Saturación de temperatura y selección de color
        if (hval<0) {
            hval=0;
            ComColor(0,0,255);          //Color azul para temperatura por
    debajo de la minima
            }
        else if (hval>H){
            hval=H;
```

```
            ComColor(255,0,0);          //Color rojo para temperatura por
encima de la máxima
            }
        else ComColor(255,165,0);       //Color naranja por defecto

        //"Mercurio"
        ComCirculo(x,y,R*0.8);
        ComRect(x-r_int, y-hval, x+r_int, y, 1);
        ComCirculo(x, y-hval, r_int+80/16);

        //Texto
        char c[8];
        sprintf(c, "%.1f C", val);
        ComColor(negro);
        ComTXT(x+2*R, y-H/2, 27, OPT_CENTER,c);
}

//SLIDER
//Solo lo dibuja
void ComSlider( int16_t x, int16_t y, int16_t w, uint16_t h, uint16_t
options, uint16_t val , uint16_t range)
{

            ComBgcolor(gris);       //Color fondo
            ComFgcolor(amarillo_oscuro);       //Color bola
            ComColor(amarillo);       //Color barra
            EscribeRam32(CMD_SLIDER);
            EscribeRam16(x);
            EscribeRam16(y);
            EscribeRam16(w);
            EscribeRam16(h);
            EscribeRam16(options);
            EscribeRam16(val);
            EscribeRam16(range);
            EscribeRam16(0);
            ComColor(negro);
            char c[10];
            sprintf(c, "LUZ=%d", val);
            ComTXT(x+w/2, y+2*h, 26, OPT_CENTER, c);
}
//Lee y dibuja el Slider
//Función iterativa que va actualizando el valor al pulsar el slider, y
si no lo mantiene
int slider(int16_t x, int16_t y, int16_t w, uint16_t h, uint16_t
options, int16_t value, uint16_t min, uint16_t max)
{
    Lee_pantalla();
    if (POSX>x && POSX<(x+w) && POSY>y && POSY<(y+h))       //Detectar
si alguna parte del slider está siendo pulsada
        value=(POSX-x)*(max-min)/w;                         //Calcular
el valor de la posición pulsada
    ComSlider(x,y,w,h,options, value-min, max-min);         //Dibujar
Slider
    return value;                                           //Devolver
el valor válido
}


#define NUM_SSI_DATA        3
```

```
int RELOJ, Flag_ints;


//Función del SLEEP fake, utilizar en caso de que dé problemas al usar
el debugger
//#define SLEEP SysCtlSleep()
#define SLEEP SysCtlSleepFake()

void SysCtlSleepFake(void)
{
 while(!Flag_ints);
 Flag_ints=0;
}

void Timer0IntHandler(void);

//Matrices utilizadas para enceder los led en funcion de [luz
medida][luz deseada]
bool l1[5][5]={{0,1,1,1,1},
               {0,0,1,1,1},
               {0,0,0,1,1},
               {0,0,0,0,1},
               {0,0,0,0,0}};
bool l2[5][5]={{0,0,1,1,1},
               {0,0,0,1,1},
               {0,0,0,0,1},
               {0,0,0,0,0},
               {0,0,0,0,0}};
bool l3[5][5]={{0,0,0,1,1},
               {0,0,0,0,1},
               {0,0,0,0,0},
               {0,0,0,0,0},
               {0,0,0,0,0}};
bool l4[5][5]={{0,0,0,0,1},
               {0,0,0,0,0},
               {0,0,0,0,0},
               {0,0,0,0,0},
               {0,0,0,0,0}};

//Variables para comprobar cuanta luz hay y cuanta queremos
int estadoLuz=4, estadoSlider=4;


char Cambia=0;
float lux;
char string[80];
int DevID=0;

int16_t T_amb, T_obj;

 float Tf_obj, Tf_amb;
 int lux_i, T_amb_i, T_obj_i;
 int luzDes=Luz_min;                        //Luz deseada, medida a
traves del Slider. Empieza como Luz_min


 // BME280
 int returnRslt;
```

```
 int g_s32ActualTemp   = 0;
 unsigned int g_u32ActualPress  = 0;
 unsigned int g_u32ActualHumity = 0;

 unsigned int actValores=0;

 float T_act,P_act,H_act;
 bool BME_on = true;

 int T_uncomp,T_comp;
 char mode;
 long int inicio, tiempo;

volatile long int ticks=0;
uint8_t Sensor_OK=0;
#define BP 2
uint8_t Opt_OK, Tmp_OK, Bme_OK, Bmi_OK;

void IntTick(void){
    ticks++;
}
int main(void) {

    //Habilitar los leds,
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);

    //Habilitar leds en el Sleep
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPION);

    //Definir los pines de los leds como salidas
    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

    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);


    //Configuración del Timer 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ/20 -1);
    TimerIntRegister(TIMER0_BASE, TIMER_A,Timer0IntHandler);
    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable();
    TimerEnable(TIMER0_BASE, TIMER_A);

    HAL_Init_SPI(1, RELOJ);  //Boosterpack a usar, Velocidad del MC
    Inicia_pantalla();       //Arranque de la pantalla

    if(Detecta_BP(1))       Conf_Boosterpack(1, RELOJ);
    else if(Detecta_BP(2))  Conf_Boosterpack(2, RELOJ);
    else return 0;
```

```
        SysCtlDelay(RELOJ/3);

     //Calibración predeterminada de la pantalla
            int i;
         #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


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

         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)
        {
            SLEEP;
            if(ticks>=10){
                ticks=0;
                if(Opt_OK)
                {
                    lux=OPT3001_getLux();
                    lux_i=(int)round(lux);
                }

                if(Bme_OK)
                {
                    returnRslt = bme280_read_pressure_temperature_humidity(
                            &g_u32ActualPress, &g_s32ActualTemp,
&g_u32ActualHumity);
                    T_act=(float)g_s32ActualTemp/100.0;
                    P_act=(float)g_u32ActualPress/100.0;
                    H_act=(float)g_u32ActualHumity/1000.0;
                }}

            //Comprobación cantidad de luz/estado luz
            if (lux<100)        estadoLuz=0;
            else if(lux<1000)   estadoLuz=1;
            else if(lux<10000)  estadoLuz=2;
            else if (lux<40000) estadoLuz=3;
            else                estadoLuz=4;
```

```
        //Comprobación luz deseada/estado Slider
        if (luzDes<20)      estadoSlider=0;
        else if (luzDes<40) estadoSlider=1;
        else if (luzDes<60) estadoSlider=2;
        else if (luzDes<80) estadoSlider=3;
        else                estadoSlider=4;

        //Escritura de los leds mediante las matrices l1,l2,l3,l4 y el
estado de la luz y del Slider

GPIOPinWrite(GPIO_PORTN_BASE,GPIO_PIN_1,GPIO_PIN_1*l1[estadoLuz][estadoS
lider]);

GPIOPinWrite(GPIO_PORTN_BASE,GPIO_PIN_0,GPIO_PIN_0*l2[estadoLuz][estadoS
lider]);

GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,GPIO_PIN_4*l3[estadoLuz][estadoS
lider]);

GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_PIN_0*l4[estadoLuz][estadoS
lider]);

        //Dibujo de la pantalla
        Lee_pantalla();
        Nueva_pantalla(fondo);
        //Dibujo de los widgets
        ComTermometro(HSIZE*2/10, VSIZE*0.75, VSIZE*0.5,
HSIZE*0.06,T_act,Temp_min,Temp_max);
        ComGauge(HSIZE*7/10, VSIZE/4, VSIZE*0.8/4, 0, (Gauge_max-
Gauge_min)/10, 2,  P_act, Gauge_min, Gauge_max);
        luzDes = slider(HSIZE/2,VSIZE*0.75, HSIZE*4/10, VSIZE*0.1, 0,
luzDes, Luz_min, Luz_max);
        ComProgress(HSIZE/2, VSIZE*0.55, HSIZE*4/10, VSIZE*0.05, 0,
H_act, Hum_min, Hum_med, Hum_max);

        Dibuja();
 }

}
void Timer0IntHandler(void)
{
   TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag
    Flag_ints=1;
   ticks++;
   //SysCtlDelay(100);
}
```