Sistemas Electrónicos Para Automatización

PRÁCTICA 5: *Uso del GUI Composer.*

17/11/2021

Francisco Javier Román Cortés 4º GIERM

Introducción

Llegamos a la última práctica de la parte de microcontroladores, luego lo que se busca es lograr varias cosas, como son el presentar y trabajar con el GUI Composer de *Texas Instruments* como otra posible alternativa para la creación de una interfaz para el microcontrolador con el que trabajamos.

También buscamos dejar cerrado con esta práctica el diseño de un SEPA, tras haber conocido bastantes elementos funcionales de diseño que permiten multitud de aplicaciones, como hemos visto durante el desarrollo de estas prácticas.

En este caso, para esta práctica, trabajaremos con los siguientes elementos a nuestra disposición:

- Connected Launchpad con micro TIVA TM4C1294NCPDT.
- *Sensors Boosterpack*.
- Placa con driver L293D y servomotor.
- Ejemplo 8 (Inicialización y lectura de sensores del *Sensors Boosterpack*) y librería sensorlib2.
- Archivos de la práctica 3 y 4, como referencia para recordar el manejo del servomotor y la lectura y representación de magnitudes de los diferentes sensores del *Sensors Boosterpack.*

Fundamento teórico

Es necesario conocer el funcionamiento de los diferentes sensores del *Sensors Boosterpack*, destacando que de la práctica 4 ya conocemos bastante bien los sensores ambientales. Sin embargo, hay que profundizar más en el resto de los sensores (acelerómetro, magnetómetro y giróscopo), para poder comprender y realizar ciertos apartados del último ejercicio de esta práctica.

Además, hay que indagar en las posibilidades que ofrece la herramienta de programación de interfaces en la nube "*GUI Composer*", propiedad de *Texas Instruments.* Para ello, se ha dedicado cierta parte de la sesión presencial a explicar y entender el funcionamiento de esta herramienta, para notar que puede ser muy útil para crear una interfaz (HMI) atractiva y funcionalmente más útil que la pantalla VM800.

Realización de la práctica

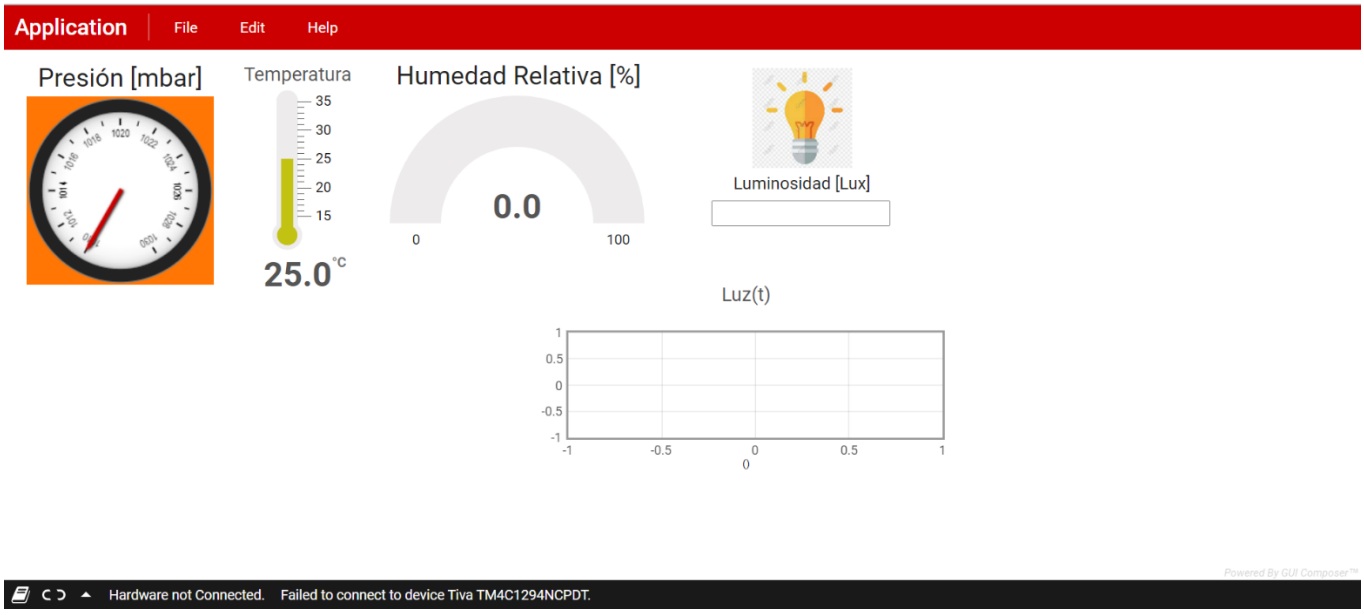
1. Estación *remota* de medida de magnitudes ambientales

Para el primer ejercicio, se pide diseñar una interfaz que muestre las medidas ***ambientales*** presentes en el *Sensors Boosterpack*. Si concretamos en que *widgets* queremos tener:

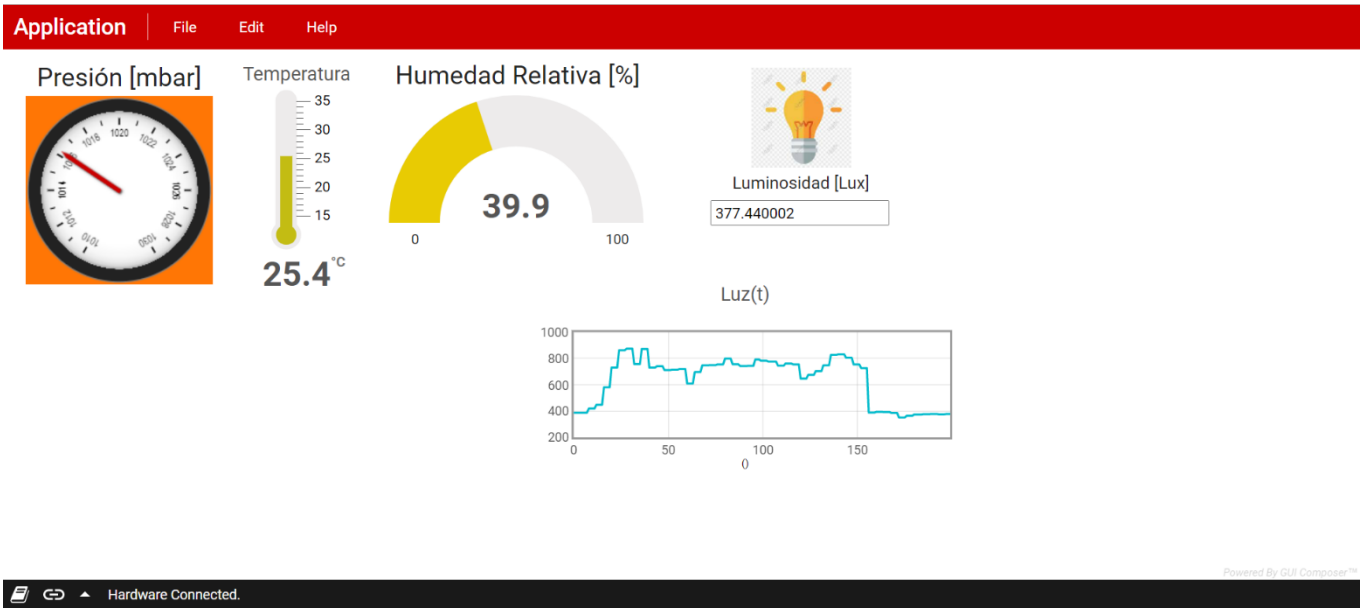
- Termómetro (del BME280).
- Barómetro, indicando presión atmosférica (también del BME 280).
- Medidor de humedad relativa (también del BME 280).
- Medidor de Luminosidad (del OPT3001).
- Gráfica de la evolución temporal de la luminosidad (OPT3001).

Para realizar esta interfaz, por tanto, tras crear un nuevo proyecto en la herramienta online *GUI Composer*, configurar el tipo de transporte (XDS), el microcontrolador concreto que usamos, la conexión Stelaris, etc. Se tiene el espacio de trabajo preparado para desarrollar la interfaz pedida. Lo único que tendremos que ir modificando será el ".out" resultado de las diferentes compilaciones de nuestros programas desarrollados en CCS, para que la interfaz online ejecute la última versión compilada de nuestro programa. (OJO, hay que cambiar el nombre a los ".out" que vayamos cargando para que la herramienta lo reconozca bien, se trata de una especie de *bug* de la herramienta). Teniendo esto en cuenta, se ha desarrollado para este primer ejercicio la siguiente interfaz:

-Interfaz sin haber conectado la placa:



Tras haber mostrado la apariencia de la interfaz, podemos mostrar también dicha interfaz realizando mediciones al conectar la placa y habiéndole incluido el ".out" con el código necesario para comunicar las variables globales (*GUI Composer* necesita recibir la información para los widgets mediante variables globales del programa de CCS que se le cargue) de las diferentes mediciones ambientales.



En esta última captura, hemos visto como en la gráfica de evolución de la luminosidad es notorio el haber aplicado luz con el flash del móvil durante cierto tiempo, para luego retirarlo y ver como la luminosidad vuelve a estabilizarse en unos 380 lux. Para el resto de las medidas ambientales son las presentes en el entorno, es decir, no se ha hecho nada para alterarlas.

Para este ejercicio, el programa que se ha utilizado es el ejemplo 8 proporcionado en Enseñanza Virtual, aunque lo adjuntaré en la memoria ya que lo he comentado brevemente.

Es el ".out" resultado de compilar este programa el que se carga en el proyecto de *GUI Composer* para este ejercicio.

```
//Programa para mostrar las medidas ambientales por la UART, y en este caso,
también por el GUI Composer//
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <math.h>               //Librerías necesarias

#include "driverlib2.h"
#include "utils/uartstdio.h"

#include "HAL_I2C.h"
#include "sensorlib2.h"



// ===========================================================================
// Function Declarations
// ===========================================================================


//Variable para el reloj de 120MHz
int RELOJ;

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

//Variable de control para indicar cuando "refrescar" las medidas
char Cambia=0;

/************* Variables para los sensores *****************/
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;

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

//Función para contar el tiempo elapsed entre varios instantes
void IntTick(void){
    ticks++;
}

int main(void) {


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


    //Configuración del TIMER a 500ms periódico
    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);

    //Configuración de la 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);
```

```
    //Si detecta el Sensors Boosterpack en la posición 1, configura los pines
y demás para esa posición
    if(Detecta_BP(1))
            {
            UARTprintf("\n--------------------------------------");
            UARTprintf("\n  BOOSTERPACK detectado en posicion 1");
            UARTprintf("\n   Configurando puerto I2C0");
            UARTprintf("\n--------------------------------------");
            Conf_Boosterpack(1, RELOJ);
            }
    //Si detecta el Sensors Boosterpack en la posición 2, configura los pines
y demás para esa posición
            else if(Detecta_BP(2))
                {
                UARTprintf("\n--------------------------------------");
                UARTprintf("\n  BOOSTERPACK detectado en posicion 2");
                UARTprintf("\n   Configurando puerto I2C2");
                UARTprintf("\n--------------------------------------");
                Conf_Boosterpack(2, RELOJ);
                }
            else
                {
                UARTprintf("\n--------------------------------------");
                UARTprintf("\n  Ningun BOOSTERPACK detectado   :-/  ");
                UARTprintf("\n              Saliendo");
                UARTprintf("\n--------------------------------------");
                return 0;
                }

    //A partir de aquí empieza a testear los diferentes sensores y se van
inicializando y comprobando si funcionan correctamente o no
     UARTprintf("\033[2J \033[1;1H Inicializando OPT3001... ");
     Sensor_OK=Test_I2C_Dir(OPT3001_SLAVE_ADDRESS);
     if(!Sensor_OK)
     {
         UARTprintf("Error en OPT3001\n");
         Opt_OK=0;

     }
     else
     {
         OPT3001_init();
         UARTprintf("Hecho!\n");
         UARTprintf("Leyendo DevID... ");
         DevID=OPT3001_readDeviceId();
         UARTprintf("DevID= 0X%x \n", DevID);
         Opt_OK=1;
     }
     UARTprintf("Inicializando ahora el TMP007...");
     Sensor_OK=Test_I2C_Dir(TMP007_I2C_ADDRESS);
     if(!Sensor_OK)
     {
         UARTprintf("Error  en TMP007\n");
         Tmp_OK=0;
     }
     else
     {
         sensorTmp007Init();
```

```
            UARTprintf("Hecho! \nLeyendo DevID... ");
            DevID=sensorTmp007DevID();
            UARTprintf("DevID= 0X%x \n", DevID);
            sensorTmp007Enable(true);
            Tmp_OK=1;
        }
        UARTprintf("Inicializando BME280... ");
        Sensor_OK=Test_I2C_Dir(BME280_I2C_ADDRESS2);
        if(!Sensor_OK)
        {
            UARTprintf("Error en BME280\n");
            Bme_OK=0;
        }
        else
        {
            bme280_data_readout_template();
            bme280_set_power_mode(BME280_NORMAL_MODE);
            UARTprintf("Hecho! \nLeyendo DevID... ");
            readI2C(BME280_I2C_ADDRESS2,BME280_CHIP_ID_REG, &DevID, 1);
            UARTprintf("DevID= 0X%x \n", DevID);
            Bme_OK=1;
        }
        Sensor_OK=Test_I2C_Dir(BMI160_I2C_ADDR2);
        if(!Sensor_OK)
        {
            UARTprintf("Error en BMI160\n");
            Bmi_OK=0;
        }
        else
        {
            UARTprintf("Inicializando BMI160, modo NAVIGATION... ");
            bmi160_initialize_sensor();
            bmi160_config_running_mode(APPLICATION_NAVIGATION);
            UARTprintf("Hecho! \nLeyendo DevID... ");
            readI2C(BMI160_I2C_ADDR2,BMI160_USER_CHIP_ID_ADDR, &DevID, 1);
            UARTprintf("DevID= 0X%x \n", DevID);
            Bmi_OK=1;
        }


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

    //Tras haber inicializado los sensores y comprobado cual funciona y cual
no, entramos en el "loop" donde iremos actualizando las medidas

    while(1)
    {
        if(Cambia==1){  //Cuando se active la variable cambia entra a
actualizar medidas (el timer la activa cada 500ms y al entrar se desactiva,
para que luego se quede de nuevo esperando al timer)
            Cambia=0;
            ticks=0;
            if(Opt_OK)              //Si el OPT3001 funciona correctamente
            {
                lux=OPT3001_getLux();   //Se obtiene la luminosidad con toda
su precisión
```

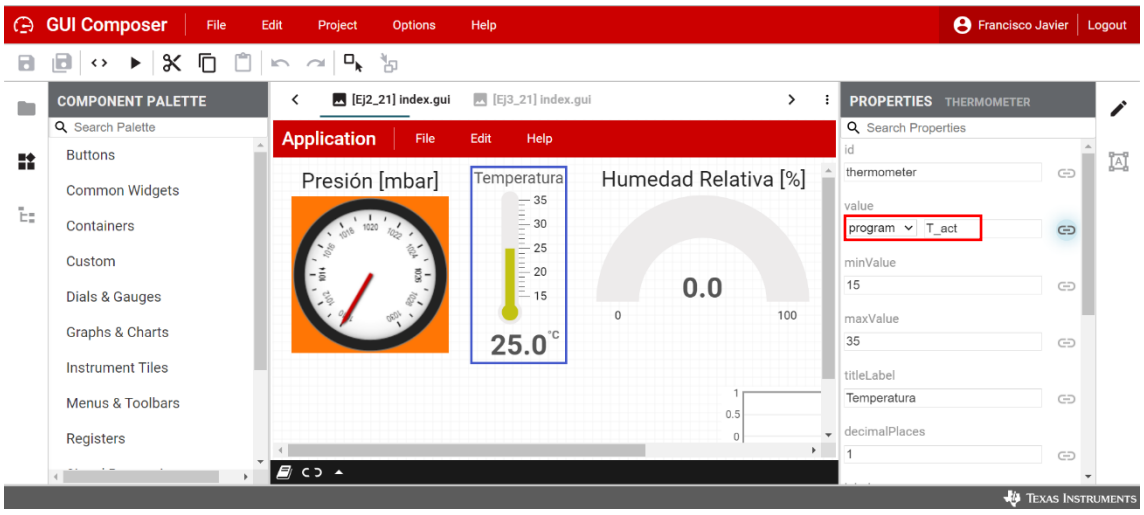
```
                lux_i=(int)round(lux);  //Se transforma la luminosidad a
entero redondeado
            }
            if(Tmp_OK)              //En mi caso TMP007 daba error al no
tenerlo, así que esta parte se la saltaría
            {
                sensorTmp007Read(&T_amb, &T_obj);
                sensorTmp007Convert(T_amb, T_obj, &Tf_obj, &Tf_amb);
                T_amb_i=(short)round(Tf_amb);
                T_obj_i=(short)round(Tf_obj);
            }
            if(Bme_OK)              //Si el BME280 funciona correctamente
            {
                returnRslt = bme280_read_pressure_temperature_humidity(
                        &g_u32ActualPress, &g_s32ActualTemp,
&g_u32ActualHumity);   //Guarda en sus variables correspondientes las 3
medidas (Temperatura, Presión y Humedad)
                T_act=(float)g_s32ActualTemp/100.0;     //Recogemos en
nuestras variables globales, formateadas como queremos, las 3 medidas del
BME280
                P_act=(float)g_u32ActualPress/100.0;
                H_act=(float)g_u32ActualHumity/1000.0;
            }
            if(Bmi_OK)              //Si el BMI160 funciona correctamente
            {
                bmi160_bmm150_mag_compensate_xyz(&s_magcompXYZ);    //Recoge
la información de las 3 componentes del magnetómetro en una struct
                bmi160_read_accel_xyz(&s_accelXYZ);                 //Recoge
la información de las 3 componentes del acelerómetro en una struct
                bmi160_read_gyro_xyz(&s_gyroXYZ);                   //Recoge
la información de las 3 componentes del giróscopo en una struct
            }
            tiempo=ticks;
            if(Opt_OK)              //Si el OPT3001 funciona correctamente
            {
                UARTprintf("\033[10;1H----------------------------\n");
//Muestra la medida de luminosidad por la UART
                sprintf(string,"  OPT3001: %5.3f Lux   \n",lux);
                UARTprintf(string);
            }
            if(Tmp_OK)              //En mi caso TMP007 daba error al no
tenerlo, así que esta parte se la saltaría
            {
                UARTprintf("-----------------------------------------\n");
                sprintf(string,"  TMP007:  T_a:%.3f, T_o:%.3f \n", Tf_amb,
Tf_obj);
                UARTprintf(string);
            }
            if(Bme_OK)              //Si el BME280 funciona correctamente
            {
                UARTprintf("-----------------------------------------\n");
//Muestra la medida de presión, humedad y temperatura por la UART
                sprintf(string, "  BME: T:%.2f C  P:%.2fmbar  H:%.3f
\n",T_act,P_act,H_act);
                UARTprintf(string);
            }
            if(Bmi_OK)          //Si el BMI160 funciona correctamente
            {
```

```
                //Muestra las 3 componentes (XYZ) del magnetómetro:
                UARTprintf("-----------------------------------------------
\n");
//                sprintf(string, "  BMM:
X:%6d\033[17;22HY:%6d\033[17;35HZ:%6d
\n",s_magcompXYZ.x,s_magcompXYZ.y,s_magcompXYZ.z);
                sprintf(string, "  BMM:  X:%6d\tY:%6d\tZ:%6d
\n",s_magcompXYZ.x,s_magcompXYZ.y,s_magcompXYZ.z);
                UARTprintf(string);
                //Muestra las 3 componentes (XYZ) del acelerómetro:
                UARTprintf("-----------------------------------------------
\n");
                sprintf(string, "  ACCL: X:%6d\tY:%6d\tZ:%6d
\n",s_accelXYZ.x,s_accelXYZ.y,s_accelXYZ.z);
                UARTprintf(string);
                //Muestra las 3 componentes (XYZ) del giróscopo:
                UARTprintf("-----------------------------------------------
\n");
                sprintf(string, "  GYRO: X:%6d\tY:%6d\tZ:%6d
\n",s_gyroXYZ.x,s_gyroXYZ.y,s_gyroXYZ.z);
                UARTprintf(string);
            }
            UARTprintf("-----------------------------------------------\n");
            sprintf(string, "TLect: %d ticks (x0.1ms)",tiempo);
            UARTprintf(string);
            sprintf(string, "\nTRep: %d ticks (x0.1ms)",ticks-tiempo);
            UARTprintf(string);
            //Muestra el elapsed time entre la lectura de los sensores y el
elapsed time durante la representación de los datos por la UART
        }
    }



    return 0;
}


//Rutina de interrupción del timer (cada 500ms)
void Timer0IntHandler(void)
{
      TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag
      Cambia=1;                          //Como ya hemos dicho, activa la
variable "Cambia", para que el loop actúe y se refresquen las mediciones
      SysCtlDelay(100);
}
```

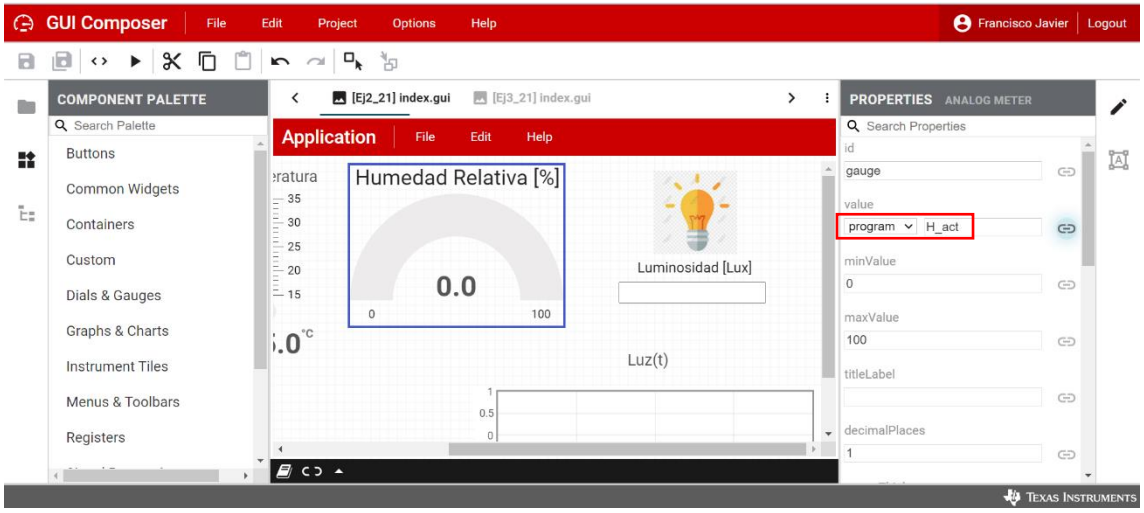
De esta manera, como vemos en este código, estamos básicamente actualizando las diferentes medidas de los sensores cada 500ms. Como para este primer ejercicio nos interesa que la interfaz del *GUI Composer* reciba la información de los sensores ambientales en cada widget correspondiente, tras haber configurado adecuadamente los protocolos de transporte entre *GUI Composer* y la *Connected Launchpad*, basta con indicarle a cada widget que variable global (que guarda la medida de cierto sensor) es la que tiene representar. Podemos ver cómo esto se ha indicado a la hora de modificar las *properties* de cada widget en algunas capturas que se adjuntan:

- Para el Gauge de presión:

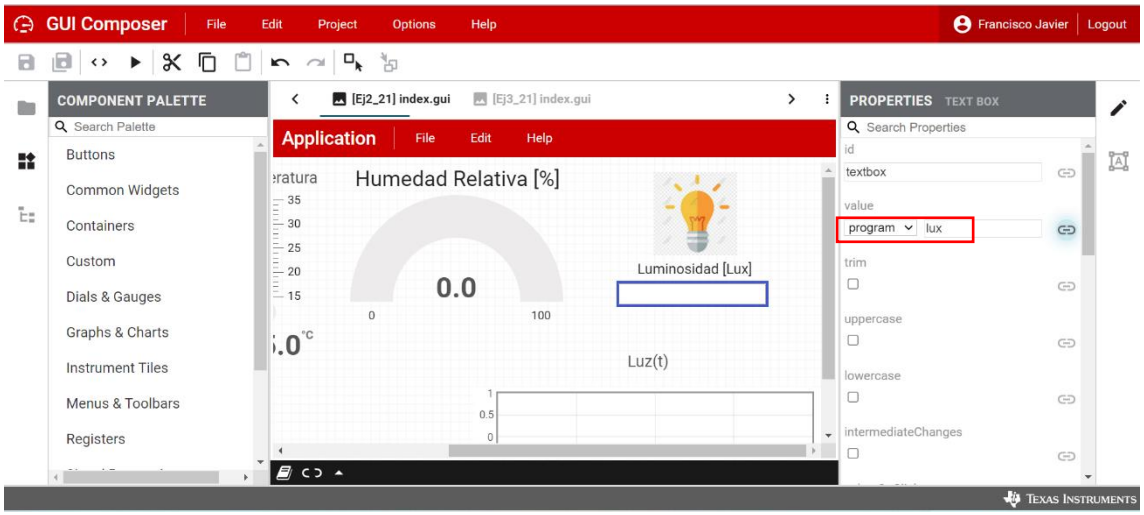
- Para el termómetro:



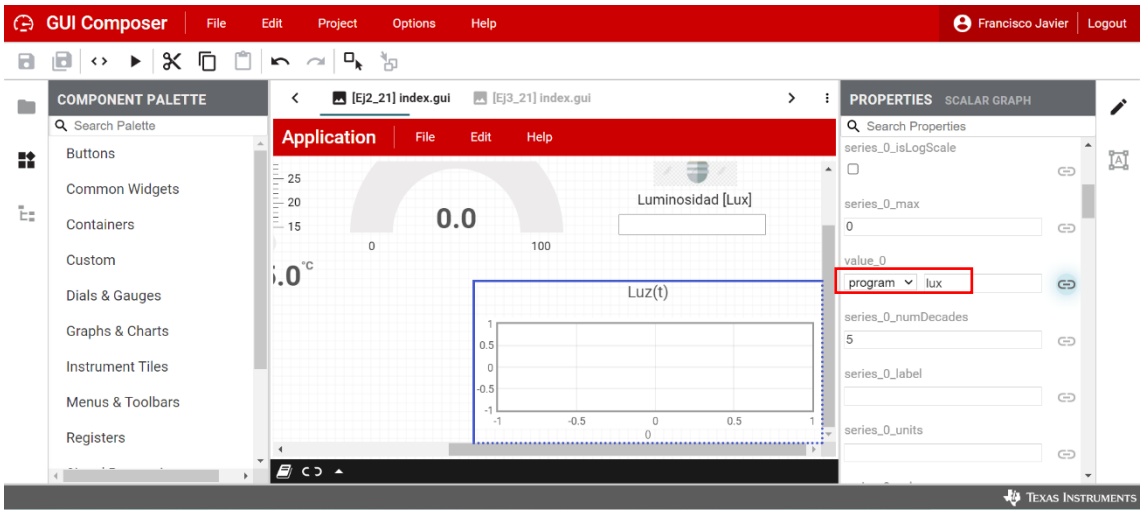
- Para el medidor de humedad relativa:



- Para el medidor de luminosidad:



- Para la gráfica que muestra la evolución de luminosidad:



Mostrando cada una de las configuraciones de cada widget (como se puede apreciar es un poco repetitivo), vemos que es bastante fácil crear y diseñar una interfaz que interactúe con los sensores que tenemos disponibles.

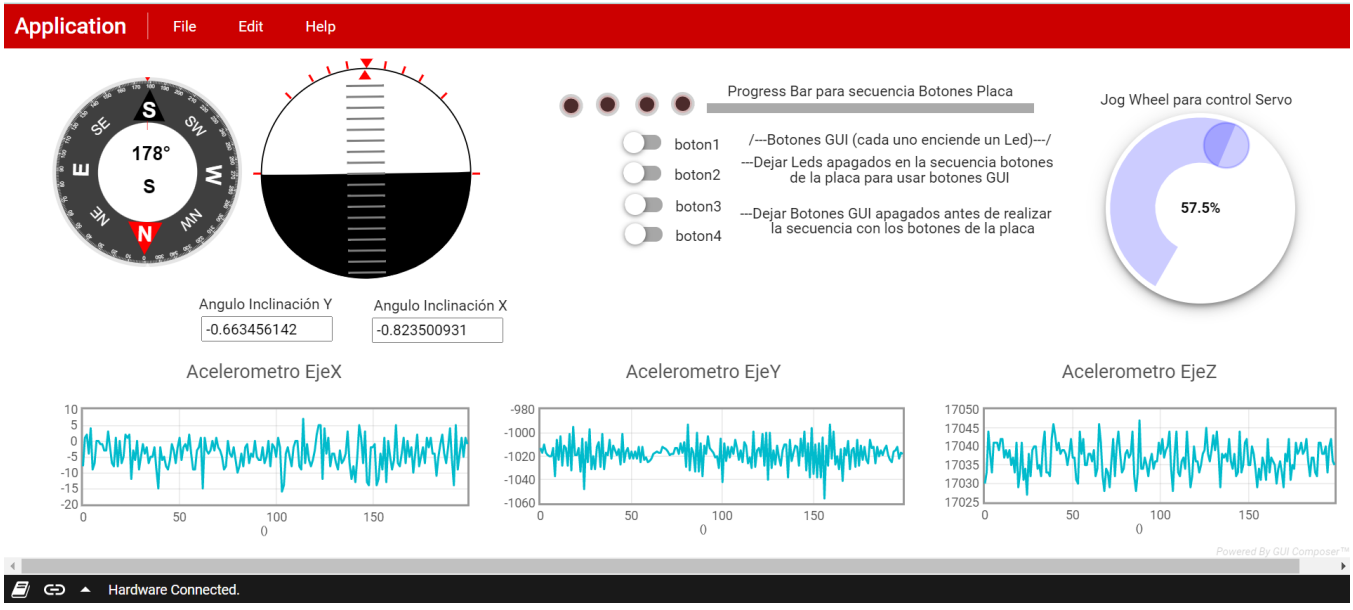
Por último, antes de pasar al segundo ejercicio propuesto, como de costumbre, recomiendo acudir al vídeo adjunto de demostración para ver el funcionamiento de manera más dinámica que con simples imágenes.

2. Gestión *remota* de un sistema móvil 3D

Para el segundo apartado de esta práctica, se busca realizar otro interfaz mediante *GUI Composer*. De esta manera, se pide ahora trabajar con los sensores que hasta ahora no se han usado, como son magnetómetro, acelerómetros y giróscopos. Para usar estos sensores e identificarlos en funcionamiento, se propone una especie de sistema que muestre la posición relativa de la placa, para ello, se busca indicar la orientación mediante una brújula. Por otra parte, se busca representar las medidas de los 3 acelerómetros, para ello, se pueden representar cada una de los componentes (XYZ) en una *Scalar Graph*, aunque es más interesante por ejemplo usar un *Attitude display*, que muestra la inclinación de la placa básicamente.

Además, para salir un poco de los sensores, se propone también poder controlar los leds y el servomotor a través de la propia interfaz del *GUI Composer*.

De esta manera, antes de explicar el funcionamiento, se muestra la interfaz realizada para este ejercicio, que resuelve los problemas planteados:



Vemos aquí como se ha planteado el sistema de localización de la placa en la parte de la izquierda e inferior de la interfaz, indicando las medidas de los acelerómetros, así como los ángulos de inclinación respecto al eje X y al eje Y, calculados mediante las medidas de dichos acelerómetros, siendo estos ángulos de inclinación lo que se representa en el *Attitude display*.

Por su parte para la brújula, se utilizan las medidas del magnetómetro para representar el ángulo que forma la orientación de la placa con respecto del Norte.

Finalmente, para la parte superior derecha de la interfaz, tenemos presente los elementos que permiten controlar tanto los leds de la placa y su estado, como la posición del servomotor.

Destacar el hecho de que la parte de los leds funciona de la siguiente manera:

- Si se pulsa el B1 de la placa (situado al borde de la placa), se van encendiendo los leds de uno en uno hasta saturar en todos encendidos, mientras que, si se pulsa B2, se van apagando de uno en uno hasta saturar en todos apagados. De ahí el sentido de la progress bar de la interfaz, que indica el progreso hasta tener todos encendidos.
- Por su parte tenemos 4 indicadores correspondiente al estado de los leds de la placa, que se encenderán de igual forma que los propios leds de la placa
- Finalmente, tenemos 4 toggle buttons en la interfaz, de manera que cada uno de ellos enciende y apaga un led asignado a dicho botón. Se ha implementado la capacidad de encender varios de ellos a la vez en cualquiera de las combinaciones.

Para no repetir tantas capturas mostrando qué variables recibe cada uno de los widgets, y a sabiendas de que el código se va a mostrar después de esto en la memoria, voy a adjuntar una lista indicando que variable mide cada widget, a la que se podrá acudir tras conocer qué hace cada variable en el código posterior.

En todos los widgets las variables se configuran con "*program":*

- **Brújula:** "value" --> angulo
- **Attitude display:** "rotation" --> angulo_accelY ///// "tilt" --> angulo_accelX
- **Text box (Ángulo Inclinación Y):** "value" --> angulo_accelY
- **Text box (Ángulo Inclinación X):** "value" --> angulo_accelX
- **Scalar Graph acelerómetro (componente X):** "value_0" --> s_accelXYZ.x
- **Scalar Graph acelerómetro (componente Y):** "value_0" --> s_accelXYZ.y
- **Scalar Graph acelerómetro (componente Z):** "value_0" --> s_accelXYZ.z
- **Led1,2,3,4:** "on" --> led1, led2, led3, led4
- **Boton1,2,3,4:** "checked" --> boton1, boton2, boton3, boton4
- **Progress Bar:** "value" --> progreso
- **Jog Wheel (Servo):** "value" --> posicion

Tras ver aquí con qué variables se han configurado los diferentes widgets, podemos analizar cómo se han desarrollado ciertos aspectos del código para que realice la funcionalidad pedida.

```
uint32_t reloj=0;
int estado;
int progreso; //Progreso de la progress bar
int boton1; //Boton 1 de la GUI
int boton2; //Boton 2 de la GUI
int boton3; //Boton 3 de la GUI
int boton4; //Boton 4 de la GUI
int led1,led2,led3,led4;
int PeriodoPWM;
volatile int posicion = media;
char string[80];
float angulo;
float declinacion_magnetica = 0.6167; //decl_mag en Sevilla de -0º 37' https://www.magnetic-declination.com/
float angulo_accelY;
float angulo_accelX;

// 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;
```

Aquí podemos ver las principales variables globales que hemos listado antes y que comunican la información de interés a la interfaz del *GUI Composer*.

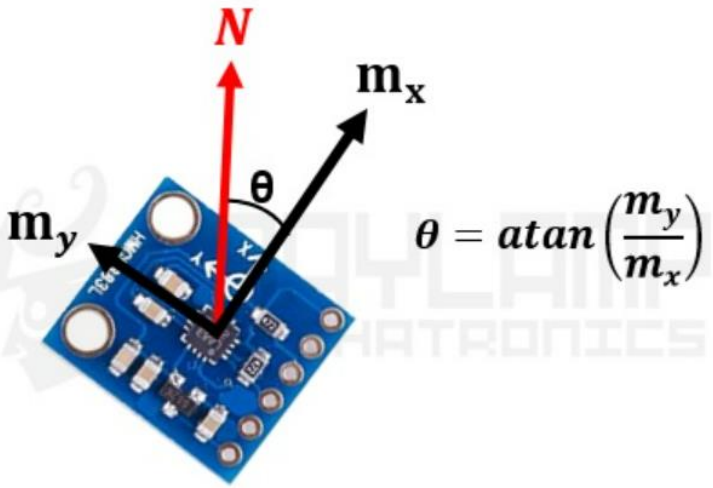
```
while(1){

    if(Bmi_OK)
    {
        bmi160_bmm150_mag_compensate_xyz(&s_magcompXYZ);
        bmi160_read_accel_xyz(&s_accelXYZ);
        bmi160_read_gyro_xyz(&s_gyroXYZ);
    }

    //Control Servo:
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, (int)posicion);

    enciende_leds(estado);      //cambia los led.
    //if(!boton1 && !boton2 && !boton3 && !boton4) enciende_leds(0);
    //Para encender un led con cada uno:
    if(boton1) enciende_leds(5);
    if(boton2) enciende_leds(6);
    if(boton3) enciende_leds(7);
    if(boton4) enciende_leds(8);
    //Para que si hay varios botones activos en la GUI se enciendan varios leds a la vez
    if(boton1 && boton2) enciende_leds(2);
    if(boton1 && boton3) enciende_leds(9);
    if(boton1 && boton4) enciende_leds(10);
    if(boton2 && boton3) enciende_leds(11);
    if(boton2 && boton4) enciende_leds(12);
    if(boton3 && boton4) enciende_leds(13);
    if(boton2 && boton3 && boton4) enciende_leds(14);
    if(boton1 && boton3 && boton4) enciende_leds(15);
    if(boton1 && boton2 && boton4) enciende_leds(16);
    if(boton1 && boton2 && boton3) enciende_leds(3);
    if(boton1 && boton2 && boton3 && boton4) enciende_leds(4);
```

Tras obviar todas las configuraciones y comprobación de sensores, vemos en este fragmento como con una simple línea controlamos el servomotor, modificando la variable “posicion” desde la propia interfaz. En cuanto a la línea ***enciende_leds(estado),*** su función es encender los leds en el caso de que se estén pulsando B1 o B2 en la placa. Respecto a las siguientes líneas, tenemos la configuración de los botones de la interfaz para encender los leds con cualquier combinación de botones ON/OFF posible, queda un poco “barroco”, pero añade cierto plus a la simple posibilidad de encender y apagar un led con cada botón, sin poder tener varios a la vez.

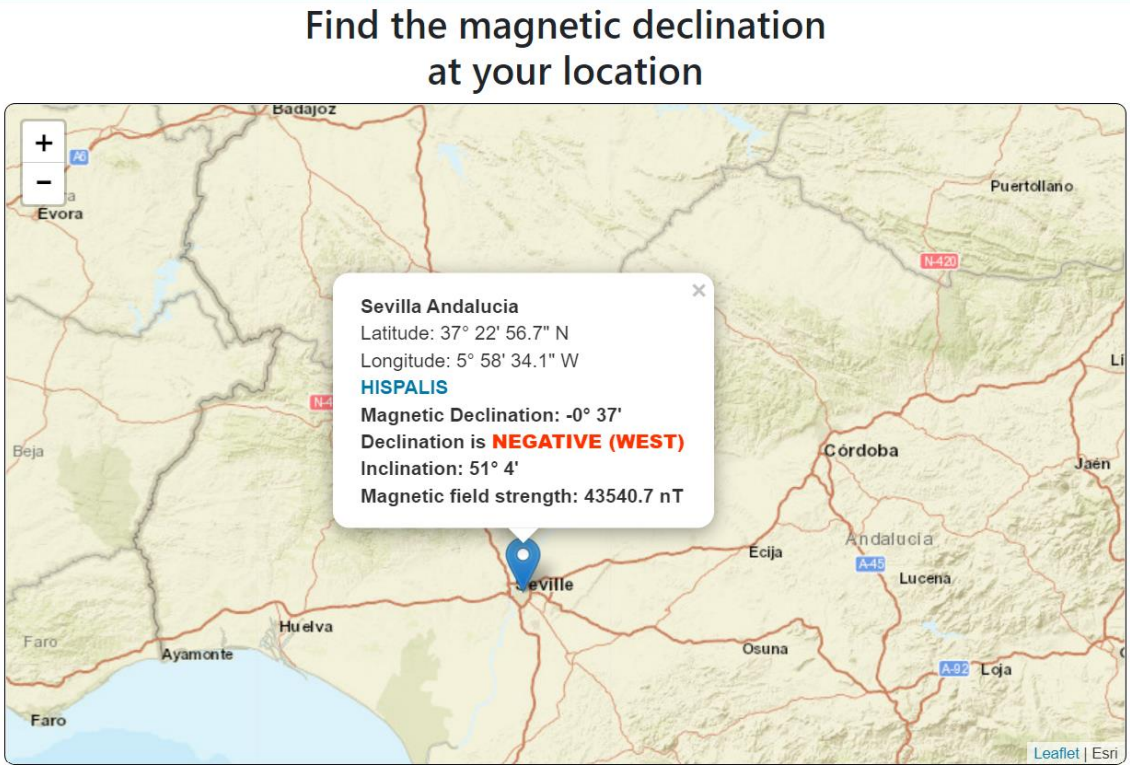
```
if(Bmi_OK)
            {
                UARTprintf("\033[10;1H----------------------------\n");
                UARTprintf("-------------------------------------------------\n");
//                sprintf(string, "  BMM:  X:%6d\033[17;22HY:%6d\033[17;35HZ:%6d  \n",s_magcompXYZ.x,s_magcompXYZ.y,s_magcompXYZ.z);
                sprintf(string, "  BMM:  X:%6d\tY:%6d\tZ:%6d  \n",s_magcompXYZ.x,s_magcompXYZ.y,s_magcompXYZ.z);

                UARTprintf(string);
                UARTprintf("-------------------------------------------------\n");
                sprintf(string, "  ACCL: X:%6d\tY:%6d\tZ:%6d  \n",s_accelXYZ.x,s_accelXYZ.y,s_accelXYZ.z);
                UARTprintf(string);
                UARTprintf("-------------------------------------------------\n");
                sprintf(string, "  GYRO: X:%6d\tY:%6d\tZ:%6d  \n",s_gyroXYZ.x,s_gyroXYZ.y,s_gyroXYZ.z);
                UARTprintf(string);

                //Cálculo del ángulo que forma la orientación de la placa respecto del Norte:(BRUJULA)
                angulo = atan2(s_magcompXYZ.y,s_magcompXYZ.x);  //Obtener el ángulo del eje X respecto al Norte
                angulo = angulo*(180/M_PI); //Convertir de rads a grados sexagesimales
                angulo = angulo-declinacion_magnetica; //Corrección de la declinación magnética
                //calculamos el ángulo equivalente de [-180 180] a [0 360]
                if(angulo<0) angulo=angulo+360;
                angulo = (int)angulo;

                //Cálculo ángulo de inclinación en el Eje Y del sensors
                angulo_accelY = atan(s_accelXYZ.y / sqrt(pow(s_accelXYZ.x, 2) + pow(s_accelXYZ.z, 2)))*(180.0 / M_PI);
                angulo_accelY = angulo_accelY + 2.80; //Compensación desfase al estar horizontal¿?

                //Cálculo ángulo de inclinación en el Eje X del sensors
                angulo_accelX = atan(s_accelXYZ.x / sqrt(pow(s_accelXYZ.y, 2) + pow(s_accelXYZ.z, 2)))*(180.0 / M_PI);
                angulo_accelX = angulo_accelX - 0.80; //Compensación desfase al estar horizontal¿?

            }
```

Aquí terminamos de concretar los cálculos necesarios para el sistema de orientación y su representación.

Para el cálculo de orientación de la placa respecto al norte, se basa en lo siguiente:



Tras realizar los cálculos necesarios y transformar a grados sexagesimales, se corrige el fenómeno de la declinación magnética (https://www.magnetic-declination.com/), que depende de la zona geográfica desde donde se esté midiendo:



Luego de aquí parte esa corrección que se aprecia en el código. Finalmente, el último paso es transformar el resultado del ángulo que estará en el rango [-180, 180] al rango [0, 360], que es el necesario para el widget de la brújula.

En cuanto a los acelerómetros, para los *Scalar Graphs*, basta con mandarle las componentes del struct s_accelXYZ.x/y/z según queramos representar una componente u otra.

Sin embargo, para los ángulos de inclinación respecto de X y de Y, que es lo que recibirá el *Attitude display,* es necesario realizar ciertos cálculos.

Podemos apreciar en la expresión usada para ambos ángulos de inclinación del código, que se trata de una composición de las 3 componentes XYZ de los acelerómetros para obtener el ángulo de inclinación correspondiente, con un paso además de conversión del ángulo de radianes a grados sexagesimales.

Se ha añadido además una especie de corrección, ya que, al colocar la placa sobre una superficie horizontal, indicaba cierta inclinación que puede corregirse con esa compensación posterior.

Con esto, ya tenemos un sistema para representar la localización/orientación de la placa, así como un control remoto de los leds y del servomotor.

Aunque en estas páginas anteriores he comentado lo fundamental de este segundo ejercicio, adjuntaré el código completo a continuación:

```
//Programa para mostrar ciertas medidas por la UART y, principalmente,
calcular ciertos parámetros para representar posición/localización de la
//placa mediante GUI Composer, así como una especie de control remoto para
los leds y el servomotor conectado a la placa//
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "driverlib2.h"             //Librerías necesarias
#include <math.h>
#include "utils/uartstdio.h"

#include "HAL_I2C.h"
#include "sensorlib2.h"

#define MSEC 40000 //Valor para 1ms con SysCtlDelay()
#define MaxEst 17

//Macros para el estado de los botones B1 y B2 de la placa
#define B1_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)
#define B1_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0))
#define B2_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1)
#define B2_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1))

//Macros para el estado de los 4 leds de la placa
#define LED1 GPIOPinRead(GPIO_PORTN_BASE, GPIO_PIN_1)
#define LED2 GPIOPinRead(GPIO_PORTN_BASE, GPIO_PIN_0)
#define LED3 GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)
#define LED4 GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0)

#define media 2700;      // Posición central del servo

int LED[MaxEst][4]={0,0,0,0,    //Matriz de estados de los leds para
encenderlos más cómodamente, a través de la función enciende_leds
             1,0,0,0,
             1,1,0,0,
             1,1,1,0,
             1,1,1,1,
             1,0,0,0,
             0,1,0,0,
             0,0,1,0,
             0,0,0,1,
             1,0,1,0,
             1,0,0,1,
             0,1,1,0,
             0,1,0,1,
             0,0,1,1,
             0,1,1,1,
             1,0,1,1,
             1,1,0,1,
             1,1,1,0

};


//Declaración de variables necesarias
uint32_t reloj=0;
int estado;     //variable estado para encender los leds según las
pulsaciones de B1 y B2 de la placa
int progreso; //Progreso de la progress bar
```

```
int boton1; //Boton 1 de la GUI
int boton2; //Boton 2 de la GUI
int boton3; //Boton 3 de la GUI
int boton4; //Boton 4 de la GUI
int led1,led2,led3,led4;    //Variables para encender o apagar los leds en la
GUI
int PeriodoPWM;             //Periodo del PWM
volatile int posicion = media;      //Se inicializa la posicion del servo a
la posición media
char string[80];                //String para mensajes UART
float angulo;                           //Variable para guardar el ángulo del
Eje X respecto del Norte (para la brújula)
float declinacion_magnetica = 0.6167; //decl_mag en Sevilla de -0º 37'
https://www.magnetic-declination.com/
float angulo_accelY;                    //Variable para guardar el ángulo de
inclinación del Eje Y (calculado mediante composición de XYZ de
acelerómetros)
float angulo_accelX;                    //Variable para guardar el ángulo de
inclinación del Eje X (calculado mediante composición de XYZ de
acelerómetros)

// 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;

 uint8_t Sensor_OK=0;
 #define BP 2
 uint8_t Opt_OK, Tmp_OK, Bme_OK, Bmi_OK;
 int DevID=0;

/*******************************************************
 * Rutina de interrupción del puerto J.
 * Leo los pines, y en cada caso hago la función necesaria,
 * (incrementar o decrementar el estado)
 * Establezco 20ms de debouncing en cada caso
 * Y debo borrar el flag de interrupción al final.
 *******************************************************/
void rutina_interrupcion(void)
{
    if(B1_ON)
    {
        while(B1_ON);
        SysCtlDelay(20*MSEC);
        estado++;
        if(estado==5) estado=5-1;               //Incrementa el estado. Si
mayor que 5, satura.
        progreso = estado;
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_0);
    }
```

```
    if(B2_ON)
      {
          while(B2_ON);
          SysCtlDelay(20*MSEC);
          estado--; if(estado==-1) estado=0;      //Decrementa el estado. Si
menor que cero, satura.
          progreso = estado;
          GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_1);
      }

}


/*****
 * Función para encender los leds usando la codificación que aparece en la
matriz LED
 * La matriz LED hace de "circuito combinacional de salida", calculando el
valor
 * de las salidas en cada estado
 */

void enciende_leds(uint8_t Est)
{
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1, GPIO_PIN_1*LED[Est][0]);
    GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, GPIO_PIN_0*LED[Est][1]);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_PIN_4*LED[Est][2]);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_PIN_0*LED[Est][3]);
}

int main(void)
{
    int estado_ant;
    //Fijar velocidad a 120MHz
    reloj=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

    //Habilitar los periféricos implicados: GPIOF, J, N
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);    //Se habilita puerto G
(PIN PWM)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);     //Se habilita generador
PWM 0

    /*******************************   CONFIGURACION PWM
*******************************/
        PWMClockSet(PWM0_BASE,PWM_SYSCLK_DIV_64);       // al PWM le llega un
reloj de 1.875MHz
        GPIOPinConfigure(GPIO_PG0_M0PWM4);              //Configurar el pin0
del puerto G (PG0) a PWM (tiene 2 pasos esta config)
        GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_0);
        PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);    //Configurar el pwm0, contador descendente y sin
sincronización (actualización automática)
        PeriodoPWM=37499;                               // 50Hz  a 1.875MHz;
50 Hz es lo que usan de pwm los servos "baratillos";; Se obtiene como:
(1875000/50)-1 = 37500-1 = 37499
        PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, PeriodoPWM);      //frec:50Hz
```

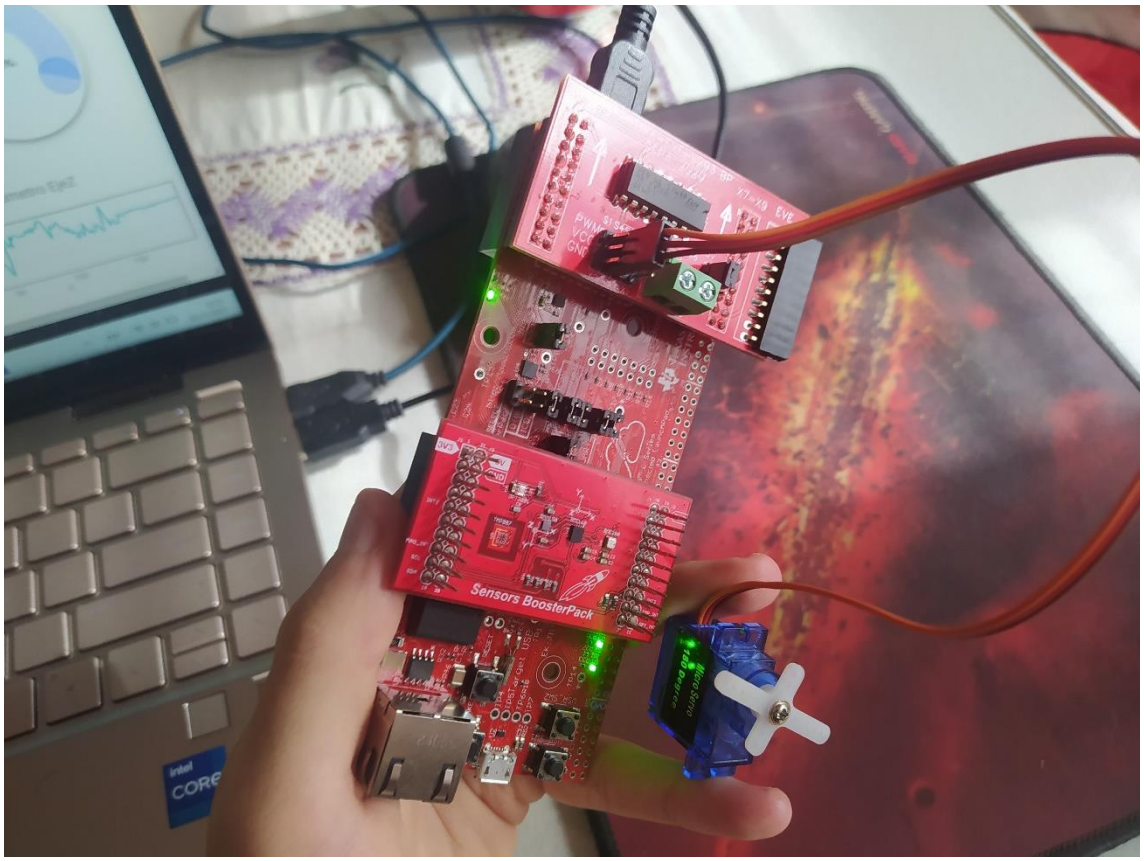
```
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, posicion);
//Inicialmente, 1ms // Para que el servo se inicialice en la posición central
        PWMGenEnable(PWM0_BASE, PWM_GEN_2);                    //Habilita el
generador 2 (donde está PWM4)
        PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT , true);       //Habilita la
salida 4

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

    //Definir tipo de pines
    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
    GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);      //J0 y
J1: entradas

GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO
_PIN_TYPE_STD_WPU); //Pullup en J0 y J1

    //Configuración de la 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);

    estado=0;       //se inicializa el estado a 0
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);       //Habilitar el
Pto J durante el Sleep

    SysCtlPeripheralClockGating(true);                      //Habilitar el
apagado selectivo de periféricos
    GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);  //Habilitar pines
de interrupción J0, J1
    GPIOIntRegister(GPIO_PORTJ_BASE, rutina_interrupcion);  //Registrar
(definir) la rutina de interrupción
    IntEnable(INT_GPIOJ);                                   //Habilitar
interrupción del pto J
    IntMasterEnable();                                      //Habilitar
globalmente las ints

    if(Detecta_BP(1)) Conf_Boosterpack(1, reloj);       //Configurar el
sensors boosterpack según la posición en que esté conectado
    else if(Detecta_BP(2)) Conf_Boosterpack(2, reloj);
    else return 0;                                      //Si no se detecta se
acaba el programa

    //Para este ejercicio basta con inicializar y comprobar el funcionamiento
del sensor BMI160
    Sensor_OK=Test_I2C_Dir(BMI160_I2C_ADDR2);
    if(!Sensor_OK)
        {
            UARTprintf("Error en BMI160\n");
            Bmi_OK=0;
        }
        else
```

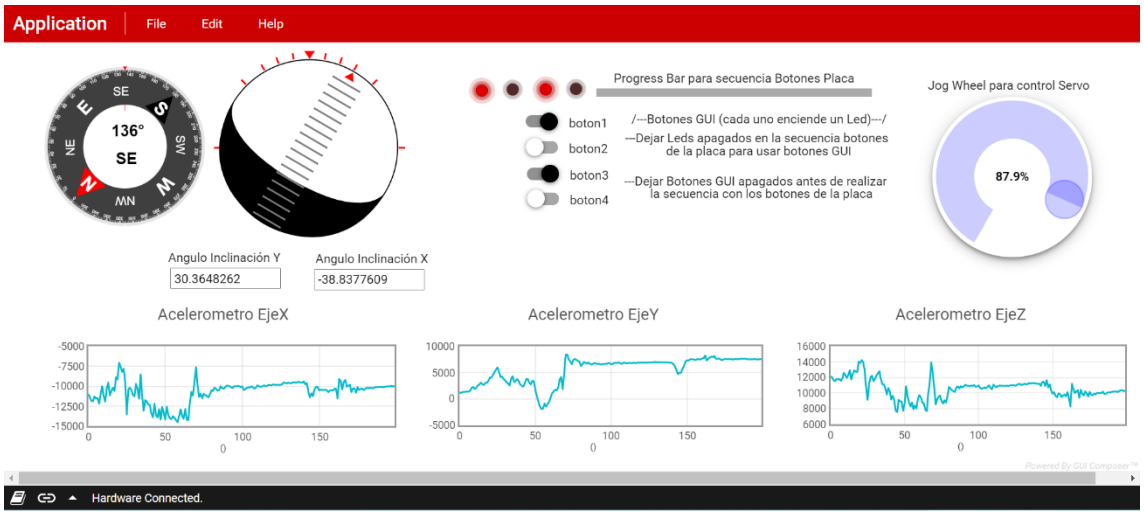
```
         {
             UARTprintf("Inicializando BMI160, modo NAVIGATION... ");
             bmi160_initialize_sensor();
             bmi160_config_running_mode(APPLICATION_NAVIGATION);
             UARTprintf("Hecho! \nLeyendo DevID... ");
             readI2C(BMI160_I2C_ADDR2,BMI160_USER_CHIP_ID_ADDR, &DevID, 1);
             UARTprintf("DevID= 0X%x \n", DevID);
             Bmi_OK=1;
         }

    while(1){       //"Loop"

        if(Bmi_OK)                      //Si el BMI160 funciona
correctamente:
        {
            bmi160_bmm150_mag_compensate_xyz(&s_magcompXYZ);        //Se
recogen las diferentes medidas en las structs correspondientes
            bmi160_read_accel_xyz(&s_accelXYZ);
            bmi160_read_gyro_xyz(&s_gyroXYZ);
        }

        //Control Servo:
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, (int)posicion);      //Se
recibe la posicion desde la GUI

        enciende_leds(estado);      //cambia los led al pulsar B1 o B2 de la
placa
        //Para encender los leds mediante los botones de la GUI:
        //Para encender un led con cada uno:
        if(boton1) enciende_leds(5);
        if(boton2) enciende_leds(6);
        if(boton3) enciende_leds(7);
        if(boton4) enciende_leds(8);
        //Para que si hay varios botones activos en la GUI se enciendan
varios leds a la vez
        if(boton1 && boton2) enciende_leds(2);
        if(boton1 && boton3) enciende_leds(9);
        if(boton1 && boton4) enciende_leds(10);
        if(boton2 && boton3) enciende_leds(11);
        if(boton2 && boton4) enciende_leds(12);
        if(boton3 && boton4) enciende_leds(13);
        if(boton2 && boton3 && boton4) enciende_leds(14);
        if(boton1 && boton3 && boton4) enciende_leds(15);
        if(boton1 && boton2 && boton4) enciende_leds(16);
        if(boton1 && boton2 && boton3) enciende_leds(3);
        if(boton1 && boton2 && boton3 && boton4) enciende_leds(4);

        // Para indicar a los 4 leds de la GUI cuando deben estar encendidos
o apagados, básicamente se encienden o apagan al reconocer si su led
        //asociado está encendido o apagado.
        if(LED1) led1 = 1;
        else led1 = 0;

        if(LED2) led2 = 1;
        else led2 = 0;

        if(LED3) led3 = 1;
        else led3 = 0;
```

```
        if(LED4) led4 = 1;
        else led4 = 0;

        if(Bmi_OK)              //Si el BMI160 funciona correctamente:
                    {
                        //Se muestran las medidas por la UART:
                        UARTprintf("\033[10;1H----------------------------
\n");
                        UARTprintf("----------------------------------------
-------\n");
        //                sprintf(string, "  BMM:
X:%6d\033[17;22HY:%6d\033[17;35HZ:%6d
\n",s_magcompXYZ.x,s_magcompXYZ.y,s_magcompXYZ.z);
                        sprintf(string, "  BMM:  X:%6d\tY:%6d\tZ:%6d
\n",s_magcompXYZ.x,s_magcompXYZ.y,s_magcompXYZ.z);

                        UARTprintf(string);
                        UARTprintf("----------------------------------------
-------\n");
                        sprintf(string, "  ACCL: X:%6d\tY:%6d\tZ:%6d
\n",s_accelXYZ.x,s_accelXYZ.y,s_accelXYZ.z);
                        UARTprintf(string);
                        UARTprintf("----------------------------------------
-------\n");
                        sprintf(string, "  GYRO: X:%6d\tY:%6d\tZ:%6d
\n",s_gyroXYZ.x,s_gyroXYZ.y,s_gyroXYZ.z);
                        UARTprintf(string);

                        //Cálculo del ángulo que forma la orientación de la
placa respecto del Norte:(BRUJULA)
                        angulo = atan2(s_magcompXYZ.y,s_magcompXYZ.x);
//Obtener el ángulo del eje X respecto al Norte
                        angulo = angulo*(180/M_PI); //Convertir de rads a
grados sexagesimales
                        angulo = angulo-declinacion_magnetica; //Corrección
de la declinación magnética
                        //calculamos el ángulo equivalente de [-180 180] a [0
360]
                        if(angulo<0) angulo=angulo+360;
                        angulo = (int)angulo;

                        //Cálculo ángulo de inclinación en el Eje Y del
sensors
                        angulo_accelY = atan(s_accelXYZ.y /
sqrt(pow(s_accelXYZ.x, 2) + pow(s_accelXYZ.z, 2)))*(180.0 / M_PI);
                        angulo_accelY = angulo_accelY + 2.80; //Compensación
desfase al estar horizontal¿?

                        //Cálculo ángulo de inclinación en el Eje X del
sensors
                        angulo_accelX = atan(s_accelXYZ.x /
sqrt(pow(s_accelXYZ.y, 2) + pow(s_accelXYZ.z, 2)))*(180.0 / M_PI);
                        angulo_accelX = angulo_accelX - 0.80; //Compensación
desfase al estar horizontal¿?

                    }
        //SysCtlSleep();              //Dormir...
    }
}
```

Finalmente, para terminar la memoria, aunque recomiendo ver el vídeo adjunto, voy a incluir unas imágenes para mostrar el funcionamiento de esta interfaz:

- Primer test sobre sistema:





- Segundo test sobre sistema: