SISTEMAS ELECTRÓNICOS PARA LA AUTOMATIZACIÓN

Proyecto de microcontroladores

Grado en Ingeniería Electrónica, Mecatrónica y Robótica

Índice

Autores: Haes-Ellis, Richard Mark
Montes Grova, Marco Antonio

1. Introducción al proyecto

En este proyecto se desarrollara un HID(*Human Interface Device*) para el control del puntero de un host, en este caso un ordenador, haciendo uso del sensorpack *BOOSTXL-SENSORS* y el microcontrolador *Tiva TM4C1294*, ambos del fabricante *Texas Instruments*.
En primer lugar, se realizara una introduccion al hardware empleado en el mismo para, posteriormente, abordar el software. Tras ello, en los siguientes puntos del proyecto, se trataran aspectos como el funcionamiento a alto nivel del proyecto como los codigos implementados en propio microcontrolador.
En un ultimo apartado se trataran futuras o posibles mejoras y sus posibles aplicaciones.

1.1. Descripcion del hardware empleado

Se ha empleado como placa de desarrollo del proyecto el modelo TM9C1294 de la serie Tiva C de Texas Instrument. Este microntrolador es el que se muestra a continuación:

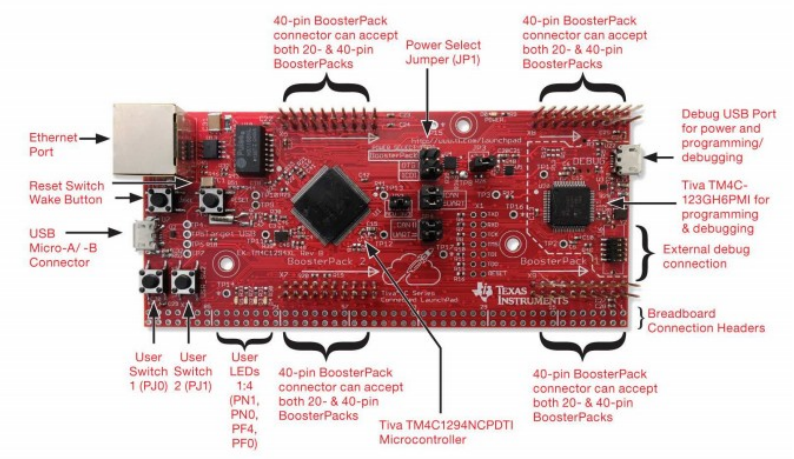

Figura 1: Microcontrolador empleado en el proyecto

Como se puede observar, la placa incorpora una conexion Ethernet, un puerto USB y una serie de leds. Además tiene un microcontrolador integrado, el Tiva TM123GH, conectado a un puerto USB cuya funcionalidad sera programar el microcontrolador principal.
Tambien se dispone de dos pares de ristras de pines destinadas al conexionado de dos Boosterpacks distintos, de tal modo que se puedan ampliar las funcionalidades como pueden ser una pantalla o el boosterpack empleado en este proyecto, BOOSTXL-SENSORS.

Para conocer informacion adiccional sobre el microcontrolador implementado en la placa puede la guia propocionada por el fabricante: http://www.ti.com/lit/ug/spmu365c/spmu365c.pdf.

Se ha optado por el uso de este microcontrolador para este proyecto por su capacidad de comunicar con un host por el puerto USB teniendo control total del periferico, de esta forma podemos emular dispostivos cotidianos como un teclado o un ratón. Además dispone de varios perifericos de comunicacion serie, los cuales nos sirven para acoplar dispostivos de radiofrecuencia y de esta forma eliminar el cableado.

En cuanto al Boosterpack empleado, BOOSTXL-SENSORS, es el mostrado a continuacion:

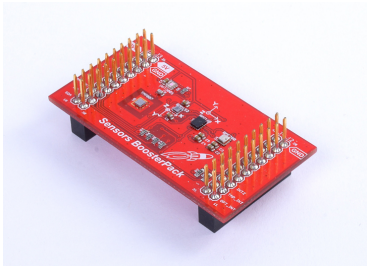

Figura 2: Boosterpack empleado en el proyecto

Contiene una gran cantidad de sensores como son una IMU, un magnetometro, sensores de temperatura y humedad del ambiente y de luminosidad. En lo que a este proyecto respecta, el principal sensor que se empleara sera la IMU integrada, la *BMI160* de 6 ejes, es decir, se encuentra formada por un acelerometro de 3 ejes y un giroscopio de 3 ejes.

La medida del acelerometro sera dada en g, la cual puede ser estimulada modificando la orientacion respecto a la gravedad de la tierra, o cambiando la velocidad a lo largo de un eje. En cuanto a la medida del giroscopio sera dada en grados por segundo, la cual se estimulara girando la placa respecto sus ejes absolutos.
En este proyecto, se emplearan las medidas asociadas a las velocidades angulares en torno a los ejes X y Z, ya que son las unicas empleadas para posicionarse en el plano 2D que forma la pantalla del computador. Se mostrara a continuacion una imagen en la cual se mostrara el movimento en torno al eje X que generara una variacion de la velocidad angular medida por el giroscpio:

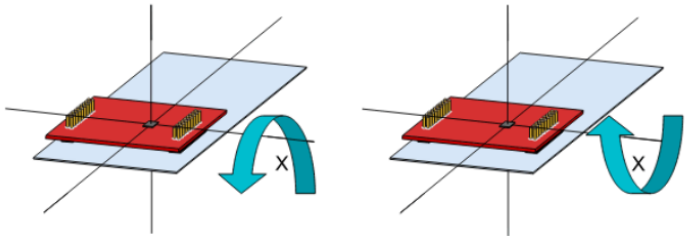

Figura 3: Movimento en torno al eje X que genera una variacion de velocidad angular

La variación del angulo en torno al eje Z se medira del mismo modo, como se muestra a continuación:

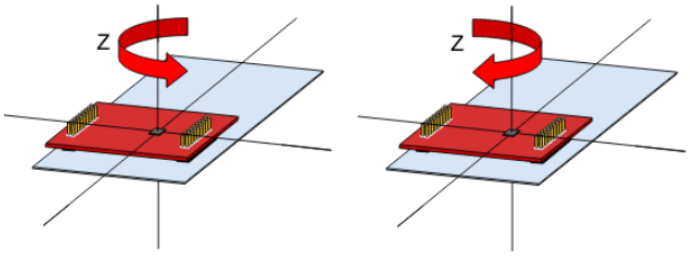

Figura 4: Movimento en torno al eje Z que genera una variacion de velocidad angular

La comunicación del sensor con otros sensores del boosterpack y con la placa sera por medio de I^2C, el cual es el principal bus serie de datos, empleado para la comunicación entre elementos de un circuito.

Cabe destacar que los datos adquiridos del sensor viene con un ruido que afecta el funcionameitno del dispositovo, por lo que se ha empleado un filtro paso bajo, que no viene ser mas que haciendo la media de las 3 muestras anteriores. Los resutados de dicho filtro se encuentra en la siguente figura;

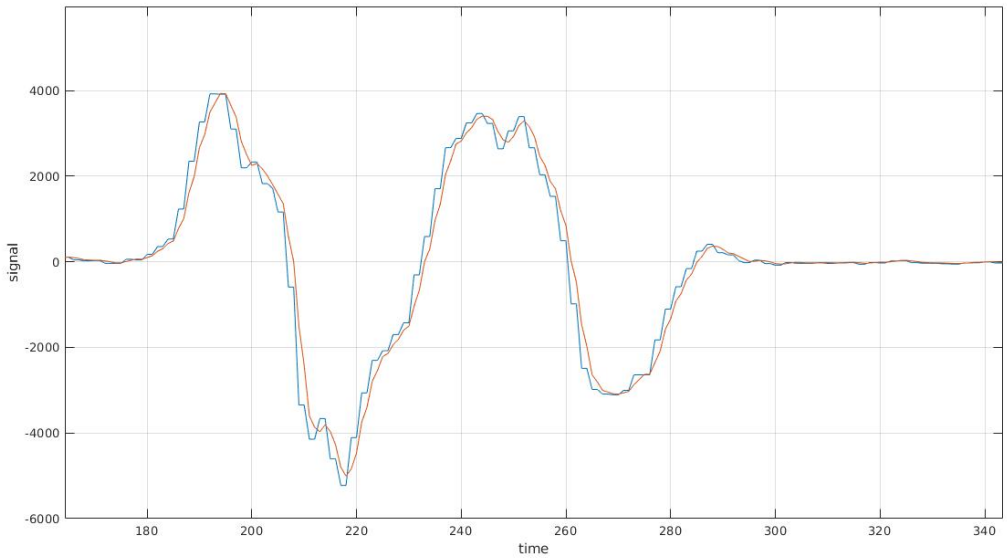

Figura 5: Grafica de filtrado de datos del acelerometro

Al igual que antes, se puede consultar datos concretos de cada sensor en la guia proporcionada por el fabricante: http://www.ti.com/lit/ug/slau666b/slau666b.pdf

1.2. Descripcion del software empleado

En cuanto al software empleado se basara en la API proporcionada por el fabricante, *TivaWare* para la familia de microcontoladores TIVA. Esta API contiene los elementos que se muestran en la siguiente imagen:

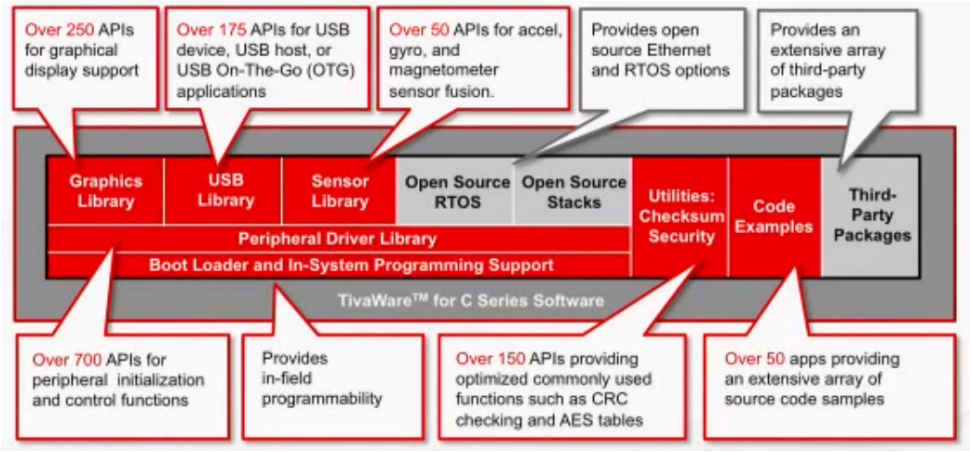

Figura 6: Estructura de la API Tivaware

Principalmente, se emplearan las liberias asociadas al manejo de los perifericos, *Peripheral Driver Library*, al manejo del puerto USB, *USB Library* y al manejo de sensores, *Sensor Library*.
Ademas de la gran ayuda brinda la API, se proporcionan una serie de ejemplos de uso que ayudaran al desarrollo del software propio.

Ademas de ello, ha sido necesario el uso de librerias para el manejo de los GPio de la placa y la comunicacion por el puerto serie de la UART.
Ademas de ello, se han empleado una serie de declaraciones proporciondadas por el profesor para solventar el desajuste de la actualizacion de las librerías, *driverlib2.h* y *sensorlib2.h*.

2. Funcionamiento del proyecto

En este apartado, se desarrollarán las principales funcionalidades implementadas en el proyecto. Las principales funciones son:

- Desarrollo de un HID con el microcontrolador.
- Empleo del Boosterpack para tomar la definir la posicion del puntero en la pantalla.
- Filtrado de las medidas tomadas a nivel de software.

Un HID(*Human Interface Device*) es una arquitectura de comunicacion empleada para comunicar los perifericos de interaccion humana como pueden ser ratones o teclados.
La comunicación entre el dispositivo HID y el host se realiza a través de un conjunto de estructuras de ïnformes"definidas por el dispositivo que el host puede consultar. Los informes se definen tanto para la comunicación de la entrada del dispositivo con el host y para la selección de salidas y funciones del host.

Además de la flexibilidad que ofrece la arquitectura básica, los dispositivos HID también se benefician de una gran universalidad entre sistemas operativos, lo que significa que no es necesario desarrollar un driver, sobre todo en el caso de dispositivos estándar como teclados y joysticks.
A pesar de estas ventajas, el uso de HID tiene un inconveniente. La tasa de datos que pueden transferirse esta limitada a un máximo de 64 KB/s.

Se empleara comunicacion mediante USB. El puerto USB de la familia Tiva TM4C de microcontroladores soporta 3 modos de funcionamiento:

- **Host mode**: Permite conectar un teclado o un raton al microcontrolador.
- **Device mode**:Establece una comunicacion con el PC a traves del USB.
- **On-The-Go mode**: Permite multiplexar el USB entre hosts y dispositivos.

En este proyecto se empleara el USB en modo *Device* o dispositivo, ya que se busca controlar el puntero del ordenador con el microcontrolador.
Durante el desarrollo del codigo de programacion se detallara el modo en el que se establece esta comunicacion entre el ordenador y el microcontrolador por medio del USB para crear el HID buscado.

Por otro lado, para controlar la posicion del puntero en la pantalla, se hara uso del giroscopio que, empleando la velocidad angular obtenida por el mismo, se podra estimar la posicion del puntero en el plano de la pantalla a partir de la definicion de un sistema inicial.
Como se mostro cuando se explico el boosterpack empleado que contiene el sensor, se tomaran las medidas de la velocidad angular en torno a los ejes X y Z.
También mencionar la pulsación de los botones del mouse, que se asociarán a los botones 1 y 2 de la placa.

3. Código de programación desarrollado

A continuación se mostrará un breve resumen de los diferentes apartados del código ya que en el mismo se encuentra detalladamente comentado cada funcionalidad.

- Instanciación de librerías
- Declaración de variables
- Funciones
- Codigo principal

3.1. Instanciacion de librerias

En el codigo principal, es decir, en *usb_ dev_ mouse.c*, se hara la declaracion de librerias. Se emplearán las librerías genericas proporcionadas por el fabricante de manejo del hardware integrado en el microcontrolador. Además, se incluirán las librerías que emplearán de la *Driverlib* y de la *Usblib*, sobre todo las dedicadas al manejo del USB HID mouse, además de los drivers de los botones y los pines del microcontrolador. También se instanciará el fichero que contiene el struct necesario que contiene informacion del fabricante entre otros. De esta forma podemso emular un HID con el microcontrolador.

```
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>

// Librerias genericas
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"

// Librerias del dirverlib
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/uart.h"

// Libreria del puerto USB y dispositivos HID
#include "usblib/usblib.h"
#include "usblib/usbhid.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdhid.h"
#include "usblib/device/usbdhidkeyb.h"
#include <usblib/device/usbdhidmouse.h>

// Libreria para los GPIO
#include "drivers/buttons.h"
#include "drivers/pinout.h"

// Informacion relativa a nuestro raton
#include "usb_mouse_structs.h"

// Libreria del puerto serie
#include "utils/uartstdio.h"

// Librerias para el sensor gyroscopo
#include "HAL_I2C.h"
#include "sensorlib2.h"
#include "driverlib2.h"
```

Listing 1: Instanciacion de librerías

3.2. Declaración de variables

Tras ello, se pasará a los *#defines* y las variables del código. Algunos de estos defines, son susceptibles a cambios cómo puede ser en el caso del boosterpack de la placa dónde se pinchen los sensores o el número de muestras que se toman para el filtro. Sin embargo, los tres primeros no lo serán, pues son importantes para la ejecución del programa.

```
#define SYSTICKS_PER_SECOND 100 // Ticks por segundo para contar cada 1 ms
#define MAX_SEND_DELAY 80 // Tiempo supuesto que tarda en mandar un reporte
#define MOUSE_REPORT_BUTTON_RELEASE 0x00 // Macro para definir boton sin pulsar
#define N 3 // Numero de muestras a filtrar
#define BP 2 // Posicion del booosterpack
```

Listing 2: Defines del código

En lo que a las variables empleadas en el código concierne, se definirán una serie de variables encargadas de mantener el estado del HID implementado. Tras ello, se definirán una serie de variables necesarias y empleadas en para tomar los datos de la IMU y filtrarla para poder emplear estos datos para el desplazamiento del puntero.
Por último, se definirá un `volatile enum` para conocer el estado del ratón, el cuál servirá para la logica de programa que gobierna el HID.

```
volatile bool g_bConnected = false; // Varibale que indica si esta conectado a PC
volatile bool g_bSuspended = false; // Variable que indica si se ha desconectado
    ↪ del bus USB
volatile uint32_t g_ui32SysTickCount;         // Contador del sistema (RELOJ)
uint32_t g_ui32PrevSysTickCount = 0; // Almacena valor del contador para contar
    ↪ tiempo

// Buffer para la UART
char string[50];

// ID del sensor BMI160
int  DevID=0;

// Almacena variables del gyroscopo
struct bmi160_gyro_t s_gyroXYZ;

// DATOS DE CALIBRACION
int16_t gyro_off_x = 6;                 // Offset del eje x
int16_t gyro_off_y = -19;         // Offset del eje y
int16_t gyro_off_z = -19;         // Offset del eje z

// Variables de sensibilidad
int32_t scaling = 23;          // Rango [1,Inf] A mas valor menos sensible
int32_t thresh  = 2;          // Rangp [1,Inf] A mas valor menos responde a
    ↪ movimientos

// Buffers para almacenar muestras para el filtrado
int32_t xfilterBuff[N]; // Eje x
int32_t yfilterBuff[N]; // Eje y

// Codigo de error
```

```
29 uint8_t cod_err=0;
30 uint8_t Bme_OK = 0, Bmi_OK;
31
32 // Datos filtrados
33 int32_t xdata = 0;                  // Eje x
34 int32_t ydata = 0;                  // Eje y
35
36 // Datos procesados (Escalado y Umbral)
37 int8_t xDistance = 0;          // Eje x
38 int8_t yDistance = 0;          // Eje y
39
40 // Variables de estado
41 uint8_t movChange = 0;         // Indica cambio de movimiento
42 uint8_t butChange = 0;         // Indica cambio en el estado de los botones
43
44 // Varibles de estado del raton
45 volatile enum{
46         STATE_UNCONFIGURED,         // Raton sin configurar
47         STATE_IDLE,                         // Nada que mandar y a la espera de
    ↪  datos
48         STATE_SUSPEND,                // Estado de suspenso
49         STATE_SENDING                // Esperando a los datos para enviar (No lo
    ↪  usamos)
50 }
51
52 // Inicialmente marcamos el dispositivo como no configurado
53 g_iMouseState = STATE_UNCONFIGURED;
```

3.3. Funciones empleadas

Una vez analizadas las variables y librerias empleadas en el código principal se pasará al análisis de las funciones. La funciones empleadas en éste código serán:

- **SysTickIntHandler**:
  Esta función será la encargada de ir incrementando una cuenta en base a un reloj configurable, en nuestro caso lo hemos configurado de tal forma que cuente 100 ticks por segundo.

```
1 // Interrupcion de cuenta de reloj del sistema
2 void SysTickIntHandler(void){
3         g_ui32SysTickCount++;
4 }
```

Listing 3: Defines del código

- **WaitForSendIdle**:
  Esta funcion sera la encargada de realizar una espera para el envio de señales por el USB o, funcionará a modo de timeout del sistema.

```
bool WaitForSendIdle(uint32_t ui32TimeoutTicks){
uint32_t ui32Start, ui32Now, ui32Elapsed;
ui32Start = g_ui32SysTickCount; // Medimos el tiempo actual
ui32Elapsed = 0;

// Mientras no haya timeout
while(ui32Elapsed < ui32TimeoutTicks)
{
        // Si esta el raton en estado de espera o no confugurado retornamos inmediatamete .
        if((g_iMouseState == STATE_IDLE) || (g_iMouseState == STATE_UNCONFIGURED))
        {
                return(true);
        }
        // Determinamos cuanto tiempo ha trascurrido desde que hemos esperado
        // deberia funcionar para una  vuelta  entera  de  g_ui32SysTickCount.
        ui32Now = g_ui32SysTickCount; // Medimos el tiempo actual

        // En el caso de que haya buffer overflow y de la vuelta (FF -> 00)
        // medimos la diferencia correspondiente
        ui32Elapsed = (ui32Start < ui32Now) ? (ui32Now - ui32Start) :
        (((uint32_t)0xFFFFFFFF - ui32Start) + ui32Now + 1);
}
// Si hemos llegado aqui esque ha pasado una vuelta entera, es decir 2^32 ticks
// de g_ui32SysTickCount, que se traduce a (0.001ms/tick)*(2^32tick) = 49.71 dias..osea...
return(false);
}
```

Listing 4: Defines del código

- **HIDMouseHandler**:
  La función que se muestra a continuación será la encargada de manejar el estado del raton que gobierna el funcionamiento del HID implementado en funcion del `volatile enum` definido anteriormente. Esta enumentacion contendrá los posibles estados en los que se puede encontrar el HID, es decir, conectado, desconectado, transmisión completada, suspendido o cuando se recupera tras la desconexion.

```
uint32_t HIDMouseHandler(void *pvCBData, uint32_t ui32Event,uint32_t ui32MsgData, void *pvMsgData){
switch (ui32Event) {
        // Si se conecta al bus, ui32Event se pondra a USB_EVENT_CONNECTED
        case USB_EVENT_CONNECTED:
        {
                g_iMouseState = STATE_IDLE; // Estado de espera
                g_bConnected = true; // Inidcamos conexion
                g_bSuspended = false;         // Y no suspenso
                break;
        }
        // Si se desconecta al bus ui32Event se pondra a USB_EVENT_DISCONNECTED.
        case USB_EVENT_DISCONNECTED:
        {
                g_iMouseState = STATE_UNCONFIGURED; // Estado desconfigurado
                g_bConnected = false; // Indicamos desconexion
                break;
        }
        // Nos vamos al estado de espera despues de haber enviado informacion
        case USB_EVENT_TX_COMPLETE:
        {
                g_iMouseState = STATE_IDLE; // Estado de espera
                break;
        }
        // Si se ha suspendido el bus USB ui32Event saltara al estado USB_EVENT_SUSPEND
        case USB_EVENT_SUSPEND:
        {
                g_iMouseState = STATE_SUSPEND; // Estado de suspension
                g_bSuspended = true; // Indicamos suspension
                break;
        }
        // Si el bus se recupera volvemos al estado de IDLE
        case USB_EVENT_RESUME:
        {
                g_iMouseState = STATE_IDLE; // Estado de espera
                g_bSuspended = false; // Indicamos no suspension
                break;
        }

        // Cualquier otro evento la ignoramos
        default:{ break;}
}
        return (0);
}
```

Listing 5: Defines del código

- **Filter**:
  Por último, se mostrará el filtro diseñado para tomar las medidas de la IMU del sensorpack, el argumento de entrada será el dato dado por la estructura `struct bmi160_gyro_t s_gyroXYZ` menos el offset definido para el calibrado y el buffer diseñado para almacenar los datos tomados.

```
int32_t filter(int32_t sensVal, int32_t values[N])
{
        int8_t i = 0;
        int8_t j = 0;
        int32_t avg;

        // Desplazamos todos los elementos a la izquierda
        for (i = 0; i < N - 1; i++)
        {
                values[i] = values[i + 1];
        }

        // Introducimos la medida al ultimo elemento del vector
        values[N - 1] = sensVal;

        // Calculamos el valor medio de todos los elementos del vector
        avg = 0;
        for (i = 0; i < N; i++)
        {
                avg = avg + values[i];
        }
        return avg / N; // Dividimos para la media
}
```

Listing 6: Defines del código

3.4. Programa principal

Para no sobrecargar la lectura, se ha optado por incluir el codigo principal del proyecto en un anexo y se comentarán a continuación los aspectos destacables.
En primer lugar se definirán las variables locales y se inicializará los perifericos y el reloj del microntrolador. Ademas de ello, se configurará el HID pasandole la estructura creada en *usb_mouse_struct.c*. Esta inicializacion y configuracion del sistema ocurre entre las lineas
Tras ello, se verificará el correcto funcionamiento de la IMU y comenzará el bucle infinito.
En él se esperará que se conecte el microcontrolador para iniciar el modo host. Una vez hecho esto, en funcion de los eventos que ocurran, es decir, conexion o desconexion del micro, se leeran datos de la IMU. Estos datos seran filtrados y escalados para posteriormente ser enviados al host.
En cuanto a la emulación de los botones del mouse, se emplearán los botones 1 y 2 de la placa para ello. Se leerán las pulsaciones empleando flancos de subida o bajada.
Por último, se enviarán los datos al host y se volverá al inicio del bucle.

4. Posibles mejoras del proyecto

En cuando a la principal mejora del proyecto, se basara en la implementación de una comunicación inalambrica en el mismo. Durante el desarrollo del proyecto se plantearon diversas vias posibles de implementacion:

- Implementación de una comunicación basada en radio-frecuencia. Esta via se planteo empleando el boosterpack del fabricante *Texas Instrument*, CC110L, el cual emplea el protocolo de comunicacion SimpliciTI. Sin embargo, este modulo de radio frecuencia, ha sido disenado para su utilizacion con el microcontrolador MSP430, y es poco compatible con otros microcontroladores, aunque sea del mismo fabricante.

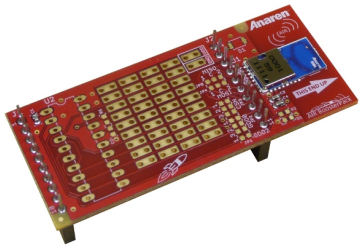

Figura 7: 430BOOST-CC110L Boosterpack

- Implementacion de una comunicacion Wi-Fi. Para la implementacion de este modo de comunicacion, se haria uso del modulo ESP01, el cual es un modulo Wi-Fi de bajo coste, que puede ser configurado como punto de acceso o como cliente y enviar mensajes TCP entre varios para comunicarse entre ellos.

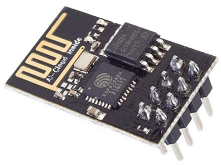

Figura 8: Modulo WiFi ESP01

Esta ultima via de comunicacion, es la que se ha considerado mas factible ya que los modulos ESP01 emplean comandos AT, es decir, el conjunto de comandos Hayes, los cuales son un conjunto de comandos empleados para configurar y parametrizar los modems.
La comunicacion del modulo ESP01 es mediante UART(*Universal Asynchronous Receiver-Transmitter*), y debido a que el microntrolador posee 8 puertos UART, es una aplicación bastante factible. Empleando las funciones para enviar datos por la UART en función de la dirección base de la misma, las cuales se encuentran ya diseñadas en el directorio raiz de este proyecto, seria posible establecer una comunicacion serial con el modulo Wi-Fi. Sin embargo, una vez establecida dicha comunicación serial, la cual conlleva consigo una sincronizacion de relojes entre las UART y la comunicación USB para implementar el dispositivo.
Una vez establecida esta comunicación, es necesario trazar un entramado de conexiones de red para crear un servidor TCP en el ESP01 a modo de punto de acceso al cual se pueda conectar el otro, el cual se encuentra en el otro microcontrolador, de tal modo que le envíe los datos por tramas TCP asociados a la IMU y la pulsación de los botones.

No se ha optado por implementarlo en este proyecto, debido a la necesidad del uso de los objetos inherentes al lenguaje de programacion C++ y sus clases para establecer un buen entramado de conexiones de red. Además de ello presentó una notable carga de tiempo de trabajo la sincronización de los relojes.

5. Anexos

5.1. Código del programa principal

```
int main(void)
{
bool bLastSuspend;
uint32_t ui32SysClock;
uint32_t ui32PLLRate;

// Run from the PLL at 120 MHz.
ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
    ↪  SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480),120000000);

// Configuramos el boosterpack
Conf_Boosterpack(BP, ui32SysClock);

// Configuramos los pines de la uart (ETHERNET|UART)
PinoutSet(false, true);

// Inicializamos los botones de la placa
ButtonsInit();

// Habilitamos el periferico UART0
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

// Inicializamos la UART para la consola .
UARTStdioConfig(0, 115200, ui32SysClock);

// Inicialmente el raton estara desconfigurado
g_bConnected = false;
g_bSuspended = false;
bLastSuspend = false;

// Inicializamos el stack del USB para el modo dispositivo
USBStackModeSet(0, eUSBModeDevice,      0);

// Le decimos a la libreria USB el clock de la CPU y la frecuncia de la PLL
// Es requerido para las placas TM4C129.
SysCtlVCOGet(SYSCTL_XTAL_25MHZ,            &ui32PLLRate);
USBDCDFeatureSet(0, USBLIB_FEATURE_CPUCLK, &ui32SysClock);
USBDCDFeatureSet(0, USBLIB_FEATURE_USBPLL, &ui32PLLRate);

// Pasamos informacion de nuestro dispositivo al dirver USB HID
// Inicializamos el controlador USB y conectamos el dispositvo al bus
USBDHIDMouseInit(0, (tUSBDHIDMouseDevice *)&g_sMouseDevice);

// Configuramos el reloj del sistema para contar 100 veces por segudo
ROM_SysTickPeriodSet(ui32SysClock / SYSTICKS_PER_SECOND);
ROM_SysTickIntEnable();
ROM_SysTickEnable();

// Mensaje Inicial
UARTprintf("\033[2J\033[H\n");
```

```
50  UARTprintf("*******************************\n");
51  UARTprintf("*         usb-mouse                  *\n");
52  UARTprintf("*******************************\n");
53
54  // Comprobamos el funcionamiento del sensor
55  UARTprintf("\033[2J \033[1;1H Inicializando BMI160... ");
56  cod_err = Test_I2C_dir(2, BMI160_I2C_ADDR2);
57  if (cod_err)
58  {
59          // Fallo del sensor
60          UARTprintf("Error 0X%x en BMI160\n", cod_err);
61          Bmi_OK = 0;
62  }
63  else
64  {
65          // Exito
66          UARTprintf("Inicializando BMI160, modo NAVIGATION... ");
67          bmi160_initialize_sensor();
68          bmi160_config_running_mode(APPLICATION_NAVIGATION);
69          UARTprintf("Hecho! \nLeyendo DevID... ");
70          readI2C(BMI160_I2C_ADDR2, BMI160_USER_CHIP_ID_ADDR, &DevID, 1);
71          UARTprintf("DevID= 0X%x \n", DevID);
72          Bmi_OK = 1;
73  }
74
75  /* BUCLE PRINCIPAL ******************************************************/
76  /* Empezamos esperando a que se conecte el micro a algun host , luego */
77  /* Luego entramos en el manejador de botones y movimiento del sensor  */
78  /* Si por lo que sea nos desconectamos del host volvemos a la espera  */
79  /************************************************************************/
80  while (1)
81  {
82  uint8_t ui8Buttons;
83  uint8_t ui8ButtonsChanged;
84
85  UARTprintf("\nEsperamos al host...\n");
86
87  // Indica que el micro aun no esta listo para usar
88  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0);
89
90  // Nos quedamos esperado si no esta conectado al host (PC)
91  while (!g_bConnected){}
92
93  // Indica que el micro esta listo para usar
94  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 1);
95
96  // Una vez connectada informamos por UART
97  UARTprintf("\nHost conectado...\n");
98
99  // Marcamos el estado de espera
100 g_iMouseState = STATE_IDLE;
101
102 // Declaramos variable de botones
103 uint8_t currB1State,prevB1State = 0;
```

```
104 uint8_t currB2State,prevB2State = 0;
105 uint8_t butReport = 0;
106
107 // En principio marcamos como bus no suspenso (Ya que nos acabamos de conectar)
108 bLastSuspend = false;
109
110 // Continuamos con nuestra logica de programa (Funcionnalidad del raton)
111 // mientras estamos conectados. Esta variable es manejada por el MouseHandler
112 // en funcion de los eventos que ocurran (Conexion/Desconecion/Suspension..)
113 while (g_bConnected)
114 {
115         // Comprobamos si el estado de suspenso ha cambiado
116         if (bLastSuspend != g_bSuspended)
117         {
118                 // En caso de que si informamos por UART
119                 bLastSuspend = g_bSuspended;
120                 if (bLastSuspend)
121                 {
122                         UARTprintf("\nBus Suspended ... \n");
123                 }
124                 else
125                 {
126                         UARTprintf("\nHost Connected ... \n");
127                 }
128         }
129         // Si estamos en el estado de espera podemos realizar las funcionalidades
    ↪ normales
130         if (g_iMouseState == STATE_IDLE)
131         {
132                 // Si ha pasado mas de 10 ms actualizamos
133                 if (g_ui32SysTickCount - g_ui32PrevSysTickCount > 1)
134                 {
135                         // Reseteamos el cotador
136                         g_ui32PrevSysTickCount = g_ui32SysTickCount;
137                         // Si el sensor esta bien
138                         if (Bmi_OK)
139                         {
140                                 // Leemos los datos por I2C
141                                 bmi160_read_gyro_xyz(&s_gyroXYZ);
142
143                                 // Filtramos los datos
144                                 xdata =
    ↪ filter(s_gyroXYZ.x-gyro_off_x,xfilterBuff);
145                                 ydata =
    ↪ filter(s_gyroXYZ.z-gyro_off_z,yfilterBuff);
146
147                                 // QUE RANGO TOMA s_gyroXYX ? ----> 16 bits!!!
148                                 // Se DEBE escalar desde -32768 a 32767. Lo
    ↪ hacemos por casting
149                                 if((xdata/scaling) < thresh && (xdata/scaling) >
    ↪ -thresh)
150                                 {       // Si esta dentro del umbral
    ↪ [-thresh,thres] rechazamos
151                                         yDistance = 0;
```

```
152                                 }
153                                 else
154                                 {        // En cualquier otro caso lo aceptamosy
    ↪  escalamos el dato
155                                         yDistance = -(int8_t)(xdata/scaling);
156                                 }
157
158                                 // Lo mismo para el eje x
159                                 if((ydata/scaling) < thresh && (ydata/scaling) >
    ↪  -thresh)
160                                 {
161                                         xDistance = 0;
162                                 }
163                                 else
164                                 {
165                                         xDistance = -(int8_t)(ydata/scaling);
166                                 }
167
168                                 // Indicamos entonces que el raton se ha movido
169                                 movChange = 1;
170                         }
171                 }
172
173                 // Comprobamos si los botones han sido pulsados
174                 ButtonsPoll(&ui8ButtonsChanged, &ui8Buttons);
175
176                 // Actualizamos las variables de estado de los botones
177                 currB1State = (ui8Buttons & LEFT_BUTTON);
178                 currB2State = (ui8Buttons & RIGHT_BUTTON);
179
180                 butChange = 0;
181
182                 // Detectamos flancos de subida o bajada
183                 if (currB1State && !prevB1State)         // SUBIDA (0->1)
184                 {
185                         prevB1State = 1; // Actualizamos el valor posterior
186                         butChange   = 1; // Indicamos cambio de estado
187                         butReport = MOUSE_REPORT_BUTTON_2;
188                 }
189                 else if (!currB1State && prevB1State) // BAJADA (1->0)
190                 {
191                         prevB1State = 0; // Actualizamos el valor posterior
192                         butChange   = 1; // Indicamos cambio de estado
193                         butReport = MOUSE_REPORT_BUTTON_RELEASE;
194                 }
195
196                 // Detectamos flancos de subida o bajada
197                 if (currB2State && !prevB2State)         // SUBIDA (0->1)
198                 {
199                         prevB2State = 1; // Actualizamos el valor posterior
200                         butChange   = 1; // Indicamos cambio de estado
201                         butReport = MOUSE_REPORT_BUTTON_1;
202                 }
203                 else if (!currB2State && prevB2State) // BAJADA (1->0)
```

```
204                 {
205                         prevB2State = 0; // Actualizamos el valor posterior
206                         butChange   = 1; // Indicamos cambio de estado
207                         butReport = MOUSE_REPORT_BUTTON_RELEASE;
208                 }
209
210                 // Solo mandamos reportes al host si ha habido cambios
211                 if(butChange || movChange)
212                 {
213                         // Indicamos estado de envio
214                         g_iMouseState = STATE_SENDING;
215                         uint32_t ui32Retcode = 0;
216                         uint8_t  bSuccess = 0;
217                         uint32_t numAtemp = 0;
218                                 // Mandamos el reportaje continuamente si falla
219                         while(!bSuccess && numAtemp < 60000)
220                         {
221                                 numAtemp++; // Numero de intentos
222                                 // Mandamos el reporte
223                                 ui32Retcode = USBDHIDMouseStateChange((void *)
    ↪ &g_sMouseDevice,
224                                 xDistance,        // Desplazamiento de pixeles en
    ↪ el eje x
225                                 yDistance,        // Desplazamiento de pixeles en
    ↪ el eje y
226                                 butReport); // Estado de los botones
227
228                                 // Si ha habido exito enviando el reporte
229                                 if (ui32Retcode == MOUSE_SUCCESS)
230                                 {
231                                         // Esperamos a que el host reciba el
    ↪ reportaje si ha ido bien
232                                         bSuccess =
    ↪ WaitForSendIdle(MAX_SEND_DELAY);
233
234                                         // Se ha acabado el tiempo y no se ha
    ↪ puesto en IDLE?
235                                         if (!bSuccess)
236                                         {
237                                                 // Asumimos que el host se ha
    ↪ desconectado
238                                                 g_bConnected = false;
239                                         }
240                                 }
241                                 else
242                                 {
243                                         // Error al mandar reporte ignoramos
    ↪ petcion e informamos
244                                         // UARTprintf("No ha sido posible enviar
    ↪ reporte.\n");
245                                         bSuccess = false;
246                                 }
247                         }
248                         // Reseteamos las variables de cambios de estado
```

```
249                             butChange = 0;
250                             movChange = 0;
251                     }
252             }
253     }
254     }
255     }
```