APLICACIONES AVANZADAS CON C18

(I)

COMUNICACIÓN USB HID

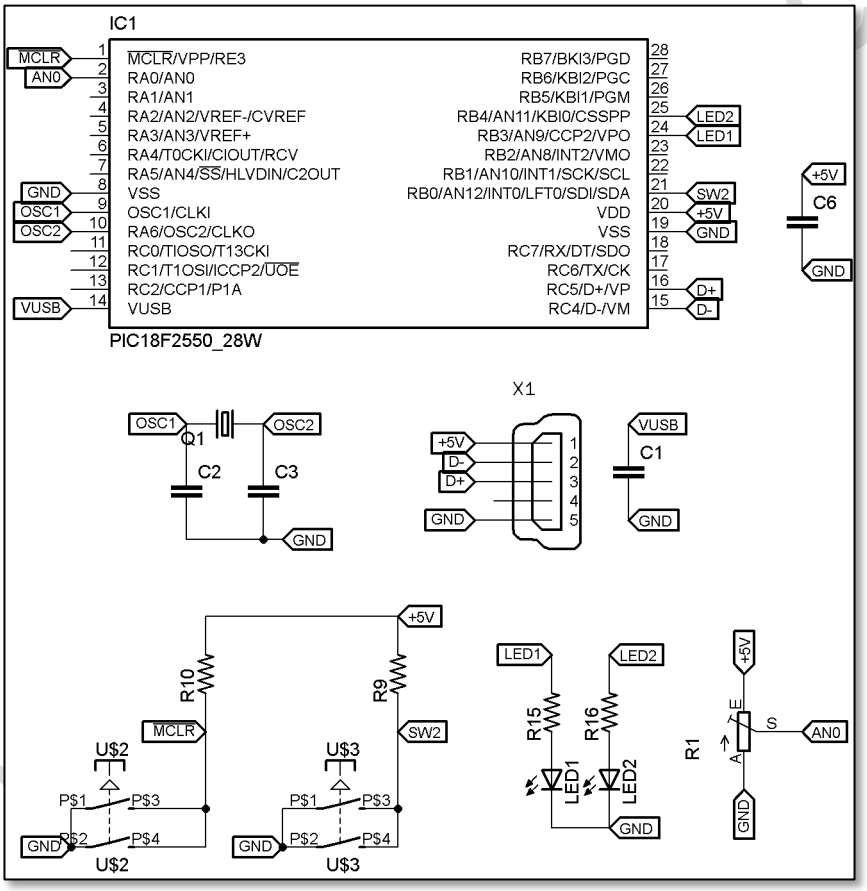
Licencia.

Introducción

En este tutorial voy a tratar de explicar cómo utilizar las librerías de Microchip para otorgar a nuestros proyectos desarrollados en C18 comunicación USB. Para comenzar debemos bajar la aplicación ***Microchip Solutions*** que se nos otorga de forma gratuita, en este caso usaremos la versión *v2010-10-19*:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en547784

También es necesario establecer el hardware sobre el cual vamos a trabajar, para determinar que microcontrolador utilizaremos, valor del cristal externo, en que pines tendremos el/los led/s que nos indicaran el proceso de enumeración del microcontrolador, si utilizaremos un pin del microcontrolador para detectar si se ha conectado a USB y si utilizaremos pin para detectar cuando hay una fuente de alimentación externa (Se puede utilizar cuando la aplicación es alimentada desde USB o desde fuente externa). En nuestro caso el hardware es el siguiente:



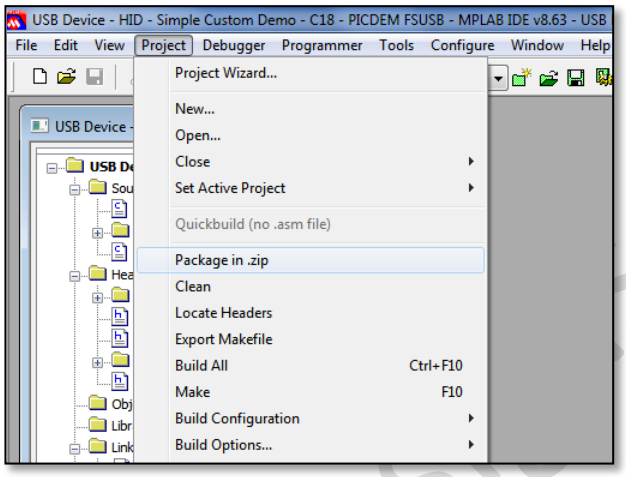
Entonces tenemos:

Microcontrolador	**PIC18F2550**
Cristal	4MHz
Led1	PIN RB3
Led2	PIN RB4
SW2	PIN RB0
POT	AN0, PIN RA0
Sin utilización de SELF_POWER_SENSE y USB_BUS_SENSE.	

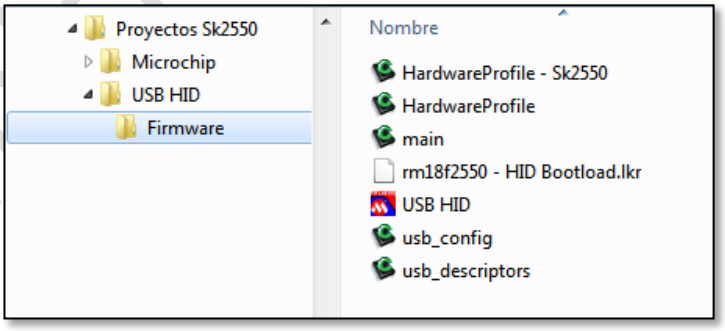
Creación de proyecto.

Dependiendo del microcontrolador que estemos utilizando (PIC18, PIC24 o PIC32) elegimos uno de los ejemplos provistos en *Microchip Solutions*. En nuestro caso vamos a seleccionar comunicación HID, y como utilizamos un **PIC18F2550** como proyecto base se toma *USB Device - HID - Simple Custom Demo - C18 - PICDEM FSUSB* ubicado en *..\Microchip Solutions v2010-10-19\USB Device - HID - Custom Demos\Generic HID – Firmware.*

Abrimos el proyecto y lo empaquetamos en un ZIP:



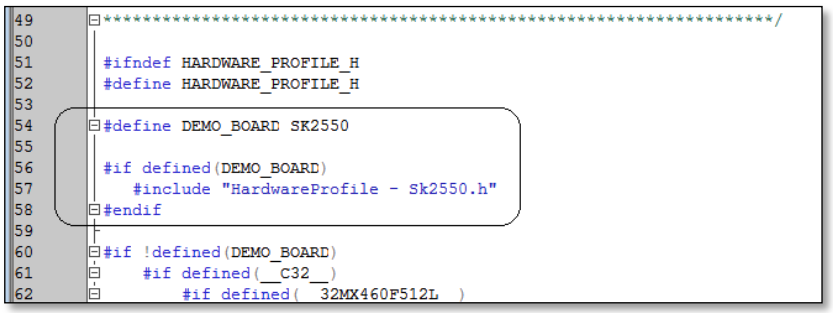
Tomamos el zip generado y lo descomprimimos, las carpetas *Microchip* y *Microchip Solutions v2010-10-19* las colocamos en la carpeta donde trabajamos con nuestros proyectos habitualmente. Ahora podemos cambiarle el nombre a la carpeta y al proyecto a gusto. También veremos que en este caso existe un archivo que define como es el hardware, denominado *HardwareProfile - PICDEM FSUSB*, en mi caso para diferenciarlo lo he cambiado por *HardwareProfile – Sk2550*. También se debe modificar el ***.lkr** del microcontrolador utilizado, ya que por lo general se aprovecha de utilizar alguno de los bootloader USB (En este caso solo basta con cambiar el nombre y dentro de él cambiar el include a *FILES p18f2550.lib*). El resultado sería el siguiente:



Adecuando el proyecto a nuestro hardware.

El USB Framework de microchip está conformado por varios módulos que permiten realizar diversos tipos de proyectos utilizando distintos modos de comunicación USB. Para ello dispone de un par de archivos en los cuales podemos definir como se utilizará en nuestra aplicación y que en modo compilación son accedidos por cada uno de los archivos estableciendo de forma global el modo de funcionamiento. A nivel hardware disponemos de 2 archivos, *HardwareProfile.h* y *HardwareProfile – Sk2550.h*. El primero es el que se

llama de forma global, pero este dependiendo del microcontrolador o la demoboard utilizada selecciona un archivo en especial. Para que en nuestro caso llame al archivo que define nuestro hardware realizamos los siguientes cambios en *HardwareProfile.h*:



```
49  ********************************************************************/
50
51      #ifndef HARDWARE_PROFILE_H
52      #define HARDWARE_PROFILE_H
53
54  #define DEMO_BOARD SK2550
55
56      #if defined(DEMO_BOARD)
57          #include "HardwareProfile - Sk2550.h"
58  #endif
59
60  #if !defined(DEMO_BOARD)
61      #if defined(__C32__)
62          #if defined(  32MX460F512L  )
```

Ahora modificamos *HardwareProfile – Sk2550.h* según nuestra necesidad:

```
40
41  #ifndef HARDWARE_PROFILE_SK2550_H
42  #define HARDWARE_PROFILE_SK2550_H
43
44      //#define USE_SELF_POWER_SENSE_IO
45      #define tris_self_power     TRISAbits.TRISA2    // Input
46      #if defined(USE_SELF_POWER_SENSE_IO)
47      #define self_power          PORTAbits.RA2
48      #else
49      #define self_power          1
50      #endif
51
52      //#define USE_USB_BUS_SENSE_IO
53      #define tris_usb_bus_sense  TRISAbits.TRISA1    // Input
54      #if defined(USE_USB_BUS_SENSE_IO)
55      #define USB_BUS_SENSE       PORTAbits.RA1
56      #else
57      #define USB_BUS_SENSE       1
58      #endif
59
60      #define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
61
62      #define DEMO_BOARD SK2550
63      #define SK2550_USB
64      #define CLOCK_FREQ 48000000
65
66      /** LED ************************************************************/
67      #define mInitAllLEDs()      {LATBbits.LATB3=0;TRISBbits.TRISB3=0;LATBbits.LATB4=0;TRISBbits.TRISB4=0;}
68
69      #define mLED_1              LATBbits.LATB3
70      #define mLED_2              LATBbits.LATB4
71
72
73      #define mGetLED_1()         mLED_1
74      #define mGetLED_2()         mLED_2
75
76
77      #define mLED_1_On()         mLED_1 = 1;
78      #define mLED_2_On()         mLED_2 = 1;
79
80
81      #define mLED_1_Off()        mLED_1 = 0;
82      #define mLED_2_Off()        mLED_2 = 0;
83
84
85      #define mLED_1_Toggle()     mLED_1 = !mLED_1;
86      #define mLED_2_Toggle()     mLED_2 = !mLED_2;
87
88      /** SWITCH *********************************************************/
89      #define mInitAllSwitches()  TRISBbits.TRISB0=1;
90      #define mInitSwitch2()      TRISBbits.TRISB0=1;
91      #define sw2                 PORTBbits.RB0
92
93      /** POT ************************************************************/
94      #define mInitPOT()          {TRISAbits.TRISA0=1;ADCON0=0x01;ADCON2=0x3C;ADCON2bits.ADFM = 1;}
95
96      /** I/O pin definitions ********************************************/
97      #define INPUT_PIN 1
98      #define OUTPUT_PIN 0
99  #endif  //HARDWARE_PROFILE_SK2550_H
```

Como se puede ver indicamos que no utilizaremos *USE_SELF_POWER_SENSE_IO* y *USE_USB_BUS_SENSE_IO*, que se utilizará *Bootloader HID* (Con ello se realizar el mapeo de los vectores de forma adecuada en el main.c), definimos *SK2550_USB* que será utilizado en el main para seleccionar la configuración de fuses correspondiente, y se declaran los pines utilizados para los leds y pulsadores.

Antes de continuar aclaremos que es **USE_SELF_POWER_SENSE_IO** y **USE_USB_BUS_SENSE_IO**:

- *USE_SELF_POWER_SENSE_IO:* Permite utilizar un pin digital del microcontrolador para detectar la presencia de una fuente externa. Con esto determinamos si es un dispositivo auto-alimentado (0) o alimentado de forma externa (1).
- *USE_USB_BUS_SENSE_IO:* Cuando el dispositivo es alimentado con una fuente externa, podemos utilizar un pin digital para determinar cuándo se conecta USB, y de esa manera controlar el estado del módulo. Esto permite encender el modulo solo cuando corresponde ahorrando energía. Si no se utiliza se debe fijar en uno (1) y el módulo USB siempre estará encendido.

Establecida la configuración del hardware pasamos al archivo *main.c* para establecer los fuses del microcontrolador, en nuestro caso queda de la siguiente manera:

```
53  #if defined(SK2550_USB)
54          #pragma config PLLDIV   = 1         // (4 MHz crystal)
55          #pragma config CPUDIV   = OSC1_PLL2
56          #pragma config USBDIV   = 2         // Clock source from 96MHz PLL/2
57          #pragma config FOSC     = XTPLL_XT
58          #pragma config FCMEN    = OFF
59          #pragma config IESO     = OFF
60          #pragma config PWRT     = OFF
61          #pragma config BOR      = ON
62          #pragma config BORV     = 3
63          #pragma config VREGEN   = ON      //USB Voltage Regulator
64          #pragma config WDT      = OFF
65          #pragma config WDTPS    = 32768
66          #pragma config MCLRE    = ON
67          #pragma config LPT1OSC  = OFF
68          #pragma config PBADEN   = OFF
69  //      #pragma config CCP2MX   = ON
70          #pragma config STVREN   = ON
71          #pragma config LVP      = OFF
72          #pragma config XINST    = OFF       // Extended Instruction Set
73          #pragma config CP0      = OFF
74          #pragma config CP1      = OFF
75  //      #pragma config CP2      = OFF
76  //      #pragma config CP3      = OFF
77          #pragma config CPB      = OFF
78  //      #pragma config CPD      = OFF
79          #pragma config WRT0     = OFF
80          #pragma config WRT1     = OFF
81  //      #pragma config WRT2     = OFF
82  //      #pragma config WRT3     = OFF
83          #pragma config WRTB     = OFF       // Boot Block Write Protection
84          #pragma config WRTC     = OFF
85  //      #pragma config WRTD     = OFF
86          #pragma config EBTR0    = OFF
87          #pragma config EBTR1    = OFF
88  //      #pragma config EBTR2    = OFF
89  //      #pragma config EBTR3    = OFF
90          #pragma config EBTRB    = OFF
91  #elif defined(PICDEM_FS_USB)      // Configuration bits for PICDEM FS USB Demo Board (based on PIC18F4550)
```

Al intentar compilar por primera vez nos indica que **sw3** no ha sido definido. Podemos hacer 2 cosas, definirlo como 1 en *HardwareProfile-Sk2550.h* o cambiarlo en *main.c* por **sw2** para de esa manera probar su funcionamiento con alguno de los software's provistos por Microchip tales como *GenericHIDSimpleDemo* o *HID PnP Demo* ubicados en la misma carpeta de donde extrajimos el proyecto base.

Configurando la comunicación USB.

Para configurar las características importantes del USB disponemos del archivo *usb_config.h*, en donde por medio de una lista de directivas #define son establecidas. Las opciones son:

- *EP0_BUFF_SIZE:* Define el tamaño del buffer para el endpoint 0. Puede tener un valor valido de 8, 16, 32 o 64 bytes. Es utilizado durante la compilación del proyecto para reservar el tamaño apropiado del buffer para dicho endpoint. Este es utilizado en el descriptor USB para notificar al USB Host el tamaño del endpoint0.
- *MAX_NUM_INT:* Define el tamaño del array de la configuración activa de cada interface, que puede ser cambiado durante la operación. Los valores válidos son los enteros [0,1,2,…].
- *USB_PING_PONG_MODE:* Define el buffer del modo ping-pong a ser usado durante la ejecución. La función de cada modo se explica en el capítulo USB de la hoja de datos del dispositivo. Las opciones para esta configuración son:
    - USB_PING_PONG__NO_PING_PONG
    - USB_PING_PONG__EP0_OUT_ONLY
    - USB_PING_PONG__FULL_PING_PONG
    - USB_PING_PONG__ALL_BUT_EP0
- *USB_USE_CLASS:* Se utiliza para indicar que clase USB debe ser incluido en el proyecto de código. Las opciones para esta configuración son las clases USB son:
    - UBS_USE_CDC
    - USB_USE_GEN
    - USB_USE_HID
    - USB_USE_MSD

  Estas definiciones son utilizadas en el archivo usb.h para establecer que archivos incluir al proyecto. Por ejemplo al definir USB_USE_HID, los archivos hid.h y hid.c deben ser agregados al proyecto.

- USB_MAX_EP_NUMBER: Indica cuantos endpoints de memoria serán utilizados por el firmware. Esta definición es usada por usbmap.c para reservar el buffer de los registros descriptores de USB.
- USB_USER_DEVICE_DESCRIPTOR: Es la variable en ROM que contiene la información del descriptor del dispositivo.
- USB_USER_CONFIG_DESCRIPTOR: Es la variable en ROM que contiene la información del descriptor de la configuración.
- USB_POLLING: Se utiliza el modo polling para las transacciones USB, en este caso se debe llamar de forma periódica a la función **USBDeviceTasks().**
- USB_INTERRUPT: Se utiliza interrupciones para responder a las transacciones USB.
- USB_SUPPORT_DEVICE: Esta definición habilita el modo dispositivo del stack USB.

- USB_INTERRUPT_LEGACY_CALLBACKS
- USB_ENABLE_ALL_HANDLERS
- USB_ENABLE_SUSPEND_HANDLER
- USB_ENABLE_WAKEUP_FROM_SUSPEND_HANDLER
- USB_ENABLE_SOF_HANDLER
- USB_ENABLE_ERROR_HANDLER
- USB_ENABLE_OTHER_REQUEST_HANDLER
- USB_ENABLE_SET_DESCRIPTOR_HANDLER
- USB_ENABLE_INIT_EP_HANDLER

- USB_ENABLE_EP0_DATA_HANDLER
- USB_ENABLE_TRANSFER_COMPLETE_HANDLER

El Stack USB llama a ciertas funciones (*USBCBxxx*) en respuesta a algunos eventos relacionados con USB. Por ejemplo, si el equipo se está apagando detendrá el envío de tramas, y en respuesta a esto los dispositivos deben disminuir el consumo de VBus a >2.5mA. Entonces en *USBCBSuspend ()* se puede actuar para cumplir tal requerimiento. Con estas definiciones podemos habilitar de forma parcial o todos estos eventos.

En el archivo *USBDescriptors.c* se contiene la información del descriptor USB para el dispositivo. Esta información varía en función de la aplicación. Un descriptor de configuración típica se compone de los siguientes componentes:

- Por lo menos un descriptor de configuración.
- Uno o más descriptores de interfaz.
- Uno o más descriptores de endpoint.

Además, suele haber una cadena de descriptor que proporciona una descripción de texto del dispositivo. Esta estructura provee un mecanismo para que el compilador le informe al sistema operativo el nombre del dispositivo que se ha implementado.

```
//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'M','i','c','r','o','c','h','i','p',' ',
'T','e','c','h','n','o','l','o','g','y',' ','I','n','c','.'
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[22];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'S','i','m','p','l','e',' ','H','I','D',' ',
'D','e','v','i','c','e',' ','D','e','m','o'
}};
```

En este archivo también se define el **Vendor ID** y el **Product ID**. El estándar USB exige que todos los dispositivos, durante la etapa de negociación, se identifiquen con un VID y un PID. Dicho par de valores sirve para conocer el fabricante del dispositivo y el modelo particular del producto que ha sido conectado, por lo tanto, modelos diferentes de un mismo producto generalmente tienen PIDs diferentes. La utilidad principal de estos valores no es solamente la de identificar el dispositivo, sino la de encontrar y cargar los drivers apropiados para el mismo. Por consiguiente, cada driver que se dispone en Windows, viene programado con uno o más PID y VID para los cuales sirve dicho driver. Esta es la forma que tiene Windows para saber si el driver seleccionado es correcto.

Si queremos distribuir nuestro producto de forma comercial, existe una organización llamada *Universal Serial Bus Implementers Forum* que se encarga de proporcionarnos, previo pago correspondiente, un VID valido para que nuestro dispositivo conviva sin problemas con el resto de dispositivos USB del mercado, además tendremos derecho a poner el logo USB en nuestro producto certificando de esta manera que ha pasado los controles pertinentes y cumple con el estándar USB.

Las funciones implementadas.

USBDeviceInit():

Esta función inicializa el dispositivo en el estado por defecto. El módulo USB se restablece por completo incluyendo todas las variables internas, registros, y las banderas de interrupción.

USBDeviceAttach():

Esta función indica al módulo USB que el dispositivo USB ha sido conectado al bus. Esta función debe ser llamada para que el dispositivo comience a ser enumerado en el bus.

USBDeviceTasks():

Mediante esta función se recibe y trasmiten paquetes a través del stack, por lo tanto debe ser llamada periódicamente. Durante el proceso de enumeración debe llamarse con preferencia cada 100us. Hay dos formas de implementarlo y según como se haya definido en usb_config.h. Mediante polling debe ser llamada de forma periódica dentro del bucle principal, en cambio mediante interrupción debe ser llamada por medio un servicio de interrupción de alta prioridad.

USBGetDeviceState():

Lee el estado del dispositivo, las opciones son:

- DETACHED_STATE: El dispositivo no está conectado al bus.
- ATTACHED_STATE: El dispositivo está conectado pero aun no está configurado por el hub/port.
- POWERED_STATE: El dispositivo está conectado y ha sido configurado por el hub/port.
- DEFAULT_STATE: Estado después de recibir un Reset desde Host.
- ADR_PENDING_STATE: Es un indicador interno al recibir desde el host el comando SET_ADDRESS.
- ADDRESS_STATE: Estado en donde el dispositivo tiene una dirección especifica en el bus.
- CONFIGURED_STATE: Dispositivo totalmente enumerado y listo para la tarea específica. También se permite aumentar el consumo de corriente al valor especificado en el descriptor de configuración.

USBHandleBusy(handle):

Función que retorna estado (true o false), indicando si hay datos para procesar enviados desde el host o si han sido enviado los datos anteriormente tratados hacia el host.

HIDTxPacket y HIDRxPacket:

Funciones propias de la clase HID que envían y recibe datos. Se debe indica endpoint, dirección del buffer utilizado para los datos y cantidad de datos.

Variables USB_HANDLE:

USB_HANDLE puede ser usado para leer la longitud de la última transferencia, el estado de la última transferencia, y demás información. Asegurarse de inicializar los objetos USB_HANDLE en NULL (0) para que estén en un estado conocido durante su primer uso.

USBSuspendControl:

es un macro de UCONbits.SUSPND. Bit que setea el modulo USB a reducción de consumo.

Para más información recurrir al Help de stack USB (MCHPFSUSB Library Help) y a la guía de usuario que se provee (Microchip USB Device Firmware Framework User's Guide)

Bueno, como trabajar con todo el código expuesto en el *main.c* de los ejemplos proporcionados por Microchip puede llegar a ser confuso, se entrega el siguiente ejemplo que cumple la misma función, solo que se tiene el código necesario para utilizar solamente el PIC18F2550:

```
#ifndef MAIN_C
#define MAIN_C

/** INCLUDES *******************************************************/
#include "./USB/usb.h"
#include "HardwareProfile.h"
#include "./USB/usb_function_hid.h"

/** CONFIGURATION **************************************************/
        #pragma config PLLDIV   = 1         // (4 MHz crystal )
        #pragma config CPUDIV   = OSC1_PLL2
        #pragma config USBDIV   = 2         // Clock source from 96MHz PLL/2
        #pragma config FOSC     = XTPLL_XT
        #pragma config FCMEN    = OFF
        #pragma config IESO     = OFF
        #pragma config PWRT     = OFF
        #pragma config BOR      = ON
        #pragma config BORV     = 3
        #pragma config VREGEN   = ON      //USB Voltage Regulator
        #pragma config WDT      = OFF
        #pragma config WDTPS    = 32768
        #pragma config MCLRE    = ON
        #pragma config LPT1OSC  = OFF
        #pragma config PBADEN   = OFF
//      #pragma config CCP2MX   = ON
        #pragma config STVREN   = ON
        #pragma config LVP      = OFF
//      #pragma config ICPRT    = OFF       // Dedicated In-Circuit Debug/Programming
        #pragma config XINST    = OFF       // Extended Instruction Set
        #pragma config CP0      = OFF
        #pragma config CP1      = OFF
//      #pragma config CP2      = OFF
//      #pragma config CP3      = OFF
        #pragma config CPB      = OFF
//      #pragma config CPD      = OFF
        #pragma config WRT0     = OFF
        #pragma config WRT1     = OFF
//      #pragma config WRT2     = OFF
//      #pragma config WRT3     = OFF
        #pragma config WRTB     = OFF       // Boot Block Write Protection
        #pragma config WRTC     = OFF
//      #pragma config WRTD     = OFF
        #pragma config EBTR0    = OFF
        #pragma config EBTR1    = OFF
//      #pragma config EBTR2    = OFF
//      #pragma config EBTR3    = OFF
        #pragma config EBTRB    = OFF

/** VARIABLES ******************************************************/
#pragma udata
    #pragma udata USB_VARIABLES=0x500
unsigned char ReceivedDataBuffer[64];
unsigned char ToSendDataBuffer[64];
#pragma udata

USB_HANDLE USBOutHandle = 0;
USB_HANDLE USBInHandle = 0;
BOOL blinkStatusValid = TRUE;

/** PRIVATE PROTOTYPES *********************************************/
void BlinkUSBStatus(void);
void YourHighPriorityISRCode();
void YourLowPriorityISRCode();

#define REMAPPED_RESET_VECTOR_ADDRESS                   0x1000
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS      0x1008
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS       0x1018

extern void _startup (void);        // See c018i.c in your C18 compiler dir
#pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
void _reset (void){
    _asm goto _startup _endasm
}
#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void){
     _asm goto YourHighPriorityISRCode _endasm
}
#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void){
```

```
     _asm goto YourLowPriorityISRCode _endasm
}
#pragma code HIGH_INTERRUPT_VECTOR = 0x08
void High_ISR (void)  {
     _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS _endasm
}
#pragma code LOW_INTERRUPT_VECTOR = 0x18
void Low_ISR (void)   {
     _asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS _endasm
}
#pragma code
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode(){
       #if defined(USB_INTERRUPT)
             USBDeviceTasks();
    #endif
}
#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode(){

}
#pragma code
/******************************************************************/
void main(void){

    // Todos los pines como digitales
    ADCON1 |= 0x0F;
    // Inicializa los pines de los leds (HardwareProfile.h)
    mInitAllLEDs();
    // Inicializa los pines de los pulsadores (HardwareProfile.h)
    mInitAllSwitches();
    // Inicializa Entrada analogica para lectura de potenciometro:
    mInitPOT();
    // Inicializa variables que indican la longitud de la última transferencia, ..
    // ..el estado de la última transferencia, y demás información
    USBOutHandle = 0;
    USBInHandle = 0;
    // Indicamos que queremos leds indicadores titilando
    blinkStatusValid = TRUE;
    // Inicializa el módulo USB.-
    USBDeviceInit();
    // Si usamos interrupciones (usb_config.h) iniciamos enumeracion de dispositivo en bus.
    #if defined(USB_INTERRUPT)
        USBDeviceAttach();
    #endif

    while(1){
        // Funcion que recibe y trasnfiere paquetes USB
        #if defined(USB_POLLING)
        USBDeviceTasks();
        #endif

        // Está habilitada la opcion de titilar leds?
        if(blinkStatusValid){
            BlinkUSBStatus();
        }
        // Estado del dispositivo
        if((USBDeviceState==CONFIGURED_STATE) && (USBSuspendControl==0)){

          if(!HIDRxHandleBusy(USBOutHandle)){             // Hay datos desde el Host?
             switch(ReceivedDataBuffer[0]){               // Que tipo de comando se ha enviado?
                case 0x80:  // Cambiar estado de los leds
                    blinkStatusValid=FALSE; // Desactivamos efecto de titilar leds.-
                    if(mGetLED_1() == mGetLED_2()){
                          mLED_1_Toggle();
                          mLED_2_Toggle();
                    }else{
                          if(mGetLED_1()){
                                mLED_2_On();
                          }else{
                                mLED_2_Off();
                          }
                    }
                break;
                case 0x81:  // Enviar estado de los pulsadores
                    ToSendDataBuffer[0] = 0x81;
                                if(sw2 == 1){
                                        ToSendDataBuffer[1] = 0x01;
```

```
                                }else{
                                                ToSendDataBuffer[1] = 0x00;
                                }
                                if(!HIDTxHandleBusy(USBInHandle)){
                                        USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
                                }
                        break;
                        case 0x37:   //Lectura del potenciomentro
                           // Han sido enviado los datos anteriores?
                           if(!HIDTxHandleBusy(USBInHandle)){
                                  ADCON0bits.GO = 1;              // Comienza conversion
                                  while(ADCON0bits.NOT_DONE);     // Espera

                                  ToSendDataBuffer[0] = 0x37;
                                        ToSendDataBuffer[1] = ADRESL;        //Measured analog voltage LSB
                                  ToSendDataBuffer[2] = ADRESH;        //Measured analog voltage MSB

                                  USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
                           }
                     break;
                    }
                    // Indicamos buffer y cantidad de datos ha recibir para el siguiente paquete.
                    USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
                }
            }
        }
    }


void BlinkUSBStatus(void){
    static WORD led_count=0;

    if(led_count == 0)led_count = 10000U;
    led_count--;

    #define mLED_Both_Off()         {mLED_1_Off();mLED_2_Off();}
    #define mLED_Both_On()          {mLED_1_On();mLED_2_On();}
    #define mLED_Only_1_On()        {mLED_1_On();mLED_2_Off();}
    #define mLED_Only_2_On()        {mLED_1_Off();mLED_2_On();}

    if(USBSuspendControl == 1)
    {
        if(led_count==0)
        {
            mLED_1_Toggle();
            if(mGetLED_1())
            {
                mLED_2_On();
            }
            else
            {
                mLED_2_Off();
            }
        }//end if
    }
    else
    {
        if(USBDeviceState == DETACHED_STATE)
        {
            mLED_Both_Off();
        }
        else if(USBDeviceState == ATTACHED_STATE)
        {
            mLED_Both_On();
        }
        else if(USBDeviceState == POWERED_STATE)
        {
            mLED_Only_1_On();
        }
        else if(USBDeviceState == DEFAULT_STATE)
        {
            mLED_Only_2_On();
        }
        else if(USBDeviceState == ADDRESS_STATE)
        {
            if(led_count == 0)
            {
                mLED_1_Toggle();
                mLED_2_Off();
```

```
            }//end if
        }
        else if(USBDeviceState == CONFIGURED_STATE)
        {
            if(led_count==0)
            {
                mLED_1_Toggle();
                if(mGetLED_1())
                {
                    mLED_2_Off();
                }
                else
                {
                    mLED_2_On();
                }
            }//end if
        }//end if(...)
    }//end if(UCONbits.SUSPND...)

}//end BlinkUSBStatus



// ******************************************
// ************** USB Callback Functions
*****************************************************************
//
*****************************************************************************************************
void USBCBSuspend(void){
    //Example power saving code.  Insert appropriate code here for the desired
    //application behavior.  If the microcontroller will be put to sleep, a
    //process similar to that shown below may be used:

    //ConfigureIOPinsForLowPower();
    //SaveStateOfAllInterruptEnableBits();
    //DisableAllInterruptEnableBits();
    //EnableOnlyTheInterruptsWhichWillBeUsedToWakeTheMicro();   //should enable at least USBActivityIF
as a wake source
    //Sleep();
    //RestoreStateOfAllPreviouslySavedInterruptEnableBits();    //Preferrably, this should be done in
the USBCBWakeFromSuspend() function instead.
    //RestoreIOPinsToNormal();                                  //Preferrably, this should be done in
the USBCBWakeFromSuspend() function instead.

    //IMPORTANT NOTE: Do not clear the USBActivityIF (ACTVIF) bit here.  This bit is
    //cleared inside the usb_device.c file.  Clearing USBActivityIF here will cause
    //things to not work as intended.
}

void USBCBWakeFromSuspend(void){
}

void USBCB_SOF_Handler(void){

}

void USBCBErrorHandler(void){
}

void USBCBCheckOtherReq(void){
    USBCheckHIDRequest();
}

void USBCBStdSetDscHandler(void){
}

void USBCBInitEP(void){
    //enable the HID endpoint
    USBEnableEndpoint(HID_EP,USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP);
    //Re-arm the OUT endpoint for the next packet
    USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
}
void USBCBSendResume(void){
    static WORD delay_count;

    USBResumeControl = 1;                // Start RESUME signaling

    delay_count = 1800U;                // Set RESUME line for 1-13 ms
```

```
    do
    {
        delay_count--;
    }while(delay_count);
    USBResumeControl = 0;
}

BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, WORD size){
    switch(event)
    {
        case EVENT_CONFIGURED:
            USBCBInitEP();
            break;
        case EVENT_SET_DESCRIPTOR:
            USBCBStdSetDscHandler();
            break;
        case EVENT_EP0_REQUEST:
            USBCBCheckOtherReq();
            break;
        case EVENT_SOF:
            USBCB_SOF_Handler();
            break;
        case EVENT_SUSPEND:
            USBCBSuspend();
            break;
        case EVENT_RESUME:
            USBCBWakeFromSuspend();
            break;
        case EVENT_BUS_ERROR:
            USBCBErrorHandler();
            break;
        case EVENT_TRANSFER:
            Nop();
            break;
        default:
            break;
    }
    return TRUE;
}
#endif
```