Phần 2:

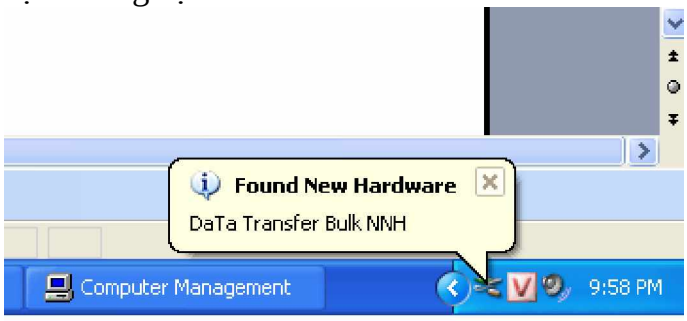
Viết Fimware cho Pic18f4550 (hoặc Pic18f2550)

1. Trình biên dịch:

Có thể viết Fimware bằng tất cả các trình biên dịch thông dụng, song theo ý kiến của tôi CCS là trình biên dịch hỗ trợ viết Fimware cho chíp USB khá tốt. Trong ví dụ của CCS có các ví dụ cho cả HID, Costume Driver và CDC. Đồng thời các thư viện hàm cho USB được xây dựng tương đối thuận lợi khi sử dụng vì vậy rất thuận lợi để các bạn có thể nhanh chóng thực hiện USB.

2. Các thức thử nghiệm:

Sau khi phần cứng đã được đấu nối đầy đủ bạn có thể nạp thử
các các fimware có sẵn tương thích với phần cứng và cắm thiết bị vào máy tính. Nếu phần cứng đúng và fimware tương thích máy tính sẽ thông báo "*Found New Hardware*" và đòi hỏi cài đặt Driver cho thiết bị. Cần lưu ý nếu phần mền trong chíp cài đặt tần số thạch anh không khớp với thạch anh phần cứng sẽ gây ra lỗi và máy tính không thể nhận diện thiết bị. Nếu khi đã tương thích về thạch anh mà máy tính vẫn không detect thiết bị thì có thể phần cứng của bạn vẫn còn vấn đề hoặc cổng USB có vấn đề về tiếp xúc hãy kiểm tra kĩ.
Nếu thiết bị của bạn không bị lỗi và fimware chính xác:



Khi nó đòi Driver hãy tạm thời bỏ qua và thực hiện tạo Driver mới cho nó bằng phần mềm WinDriver, khi tạo xong Driver thì thiết bị sẽ được tự động update driver, chi tiết việc tạo driver được hướng dẫn ở phần 3.

3. Các thư viện và hàm sử dụng chính để viết USB:

Có 3 file bạn cần include vào project CCS của bạn là:

- *#include <pic18_usb.h>*
- *#include <usb.c>*
- *#include <usb_desc_scope1.h>*

Hai file trên có sẵn trong thư viện của CCS chứa các định nghĩa và các hàm phục vụ cho giao tiếp USB, file thứ 3 là file mô tả thiết bị được chỉnh sửa từ file *usb_desc_scope.h* cũng có sẵn trong thư viện của CCS để phù hợp với yêu cầu của bạn. Ngoài ra còn một file nữa là:

- *#include .đường dẫn/ usb_demo_bulk.h>*

File này không có sẵn trong CCS như các file trên. Nó được tạo ra khi bạn lập một Project trên CCS qua PIC Wizard, tên file do bạn đặt.

4. Các hàm điều khiển và giao tiếp USB:

Khi xem các mã nguồn của các file trên trong CCS, bạn sẽ thấy rất nhiều hàm và định nghĩa khó hiểu. Nhưng phần lớn bạn sẽ không cần quan tâm tới các hàm đó vì chúng được xây dựng để trình biên dịch sử dụng. Cái chúng ta quan tâm chỉ là tập hàm "User Functions" mà CCS đã xây dựng sẵn:

```
////   **********   USER FUNCTIONS   **********   ////
////                                              ////
//// usb_init() - Initializes the USB stack, the USB peripheral and  ////
////           attaches the unit to the usb bus.  Enables        ////
////           interrupts.                            ////
////                                              ////
//// usb_init_cs() - A smaller usb_init(), does not attach unit      ////
////           to usb bus or enable interrupts.                ////
////                                              ////
//// usb_put_packet() - Sends one packet to the host.              ////
////                If you need to send a message that spans     ////
////                more than one packet then see usb_puts() in  ////
////                usb.c                                 ////
////                                              ////
//// usb_kbhit() - Returns true if OUT endpoint contains data from   ////
////           host.                                  ////
////                                              ////
//// usb_rx_packet_size() - Returns the size of packet that was      ////
////           received.  usb_kbhit() must return TRUE else      ////
////           this is not valid.  Don't forget in USB there     ////
////           are 0 len packets!                          ////
////                                              ////
//// usb_get_packet() - Gets one packet that from the host.         ////
////                usb_kbhit() must return true before you call ////
////                this routine or your data may not be valid.  ////
////                Once usb_kbhit() returns true you want to    ////
////                call this as soon as possible to get data    ////
////                out of the endpoint buffer so the PC can     ////
////                start sending more data, if needed.         ////
////                This only receives one packet, if you are    ////
////                trying to receive a multi-packet message     ////
////                see usb_gets() in usb.c.                ////
////                                              ////
//// usb_detach() - De-attach USB from the system.                ////
////                                              ////
//// usb_attach() - Attach USB to the system.                     ////
////                                              ////
//// usb_attached() - Returns TRUE if the device is attached to a    ////
////              USB cable.  A macro that looks at the defined  ////
////              connection sense pin.                     ////
////                                              ////
//// usb_task() - Keeps track of connection sense, calling          ////
////           usb_detach() and usb_attach() when needed.        ////
////                                              ////
//// For more documentation on these functions read the comments at  ////
```

```
//// each function.                                    ////
////                                                  ////
//// The other functions defined in this file are for use by the    ////
//// USB code, and is not meant to be used by the user.             ////
```

Các bạn có thể dễ dàng tìm hiểu thêm cách thức sử dụng các hàm này qua các Example và các Comment của CCS. Với các hàm này bạn đã có thể điều khiển modul USB của pic18 khá linh hoạt và có thể mở rộng chúng để phù hợp với mục đích của bạn.

5. Tạo lại file mô tả thiết bị *usb_desc_scope1.h* được thực hiện như sau:

```
#DEFINE USB_TOTAL_CONFIG_LEN    32 //config+interface+class+endpoint

  //configuration descriptor
  char const USB_CONFIG_DESC[] = {
  //config_descriptor for config index 1
       USB_DESC_CONFIG_LEN,     //length of descriptor size
       USB_DESC_CONFIG_TYPE,        //constant CONFIGURATION (0x02)
       USB_TOTAL_CONFIG_LEN,0,  //size of all data returned for this config
       1,      //number of interfaces this device supports
       0x01,                    //identifier for this configuration.  (IF we had more than one
configurations)
       0x00,             //index of string descriptor for this configuration
       0xC0,               //bit 6=1 if self powered, bit 5=1 if supports remote wakeup (we don't),
bits 0-4 reserved and bit7=1
       0x32,                //maximum bus power required (maximum milliamperes/2)  (0x32 =
100mA)

  //interface descriptor 0 alt 0
       USB_DESC_INTERFACE_LEN,  //length of descriptor
       USB_DESC_INTERFACE_TYPE,      //constant INTERFACE (0x04)
       0x00,             //number defining this interface (IF we had more than one interface)
       0x00,             //alternate setting
       2,      //number of endpoints, not counting endpoint 0.
       0xFF,             //class code, FF = vendor defined
       0xFF,             //subclass code, FF = vendor
       0xFF,             //protocol code, FF = vendor
       0x00,             //index of string descriptor for interface

  //endpoint descriptor
       USB_DESC_ENDPOINT_LEN, //length of descriptor
       USB_DESC_ENDPOINT_TYPE,     //constant ENDPOINT (0x05)
       0x81,             //endpoint number and direction (0x81 = EP1 IN)
       0x02,             //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is interrupt)
       USB_EP1_TX_SIZE & 0xFF,USB_EP1_TX_SIZE >> 8,          //maximum packet size
supported
       0x01,             //polling interval in ms. (for interrupt transfers ONLY)

  //endpoint descriptor
```

```
        USB_DESC_ENDPOINT_LEN, //length of descriptor
        USB_DESC_ENDPOINT_TYPE,     //constant ENDPOINT (0x05)
        0x01,           //endpoint number and direction (0x01 = EP1 OUT)
        0x02,           //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is interrupt)
        USB_EP1_RX_SIZE & 0xFF,USB_EP1_RX_SIZE >> 8,          //maximum packet size supported
        0x01,           //polling interval in ms. (for interrupt transfers ONLY)

    };

    //****** BEGIN CONFIG DESCRIPTOR LOOKUP TABLES ********
    //since we can't make pointers to constants in certain pic16s, this is an offset table to find
    //  a specific descriptor in the above table.

    //NOTE: DO TO A LIMITATION OF THE CCS CODE, ALL HID INTERFACES MUST START AT 0 AND BE SEQUENTIAL
    //          FOR EXAMPLE, IF YOU HAVE 2 HID INTERFACES THEY MUST BE INTERFACE 0 AND INTERFACE 1
    #define USB_NUM_HID_INTERFACES   0

    //the maximum number of interfaces seen on any config
    //for example, if config 1 has 1 interface and config 2 has 2 interfaces you must define this as 2
    #define USB_MAX_NUM_INTERFACES   1

    //define how many interfaces there are per config.  [0] is the first config, etc.
    const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};

    #if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
      #error USB_TOTAL_CONFIG_LEN not defined correctly
    #endif


////////////////////////////////////////////////////////////////
///
///   start device descriptors
///
////////////////////////////////////////////////////////////////

    //device descriptor
    char const USB_DEVICE_DESC[] ={
        USB_DESC_DEVICE_LEN,        //the length of this report
        0x01,           //constant DEVICE (0x01)
        0x10,0x01,          //usb version in bcd
        0x00,           //class code (if 0, interface defines class.  FF is vendor defined)
        0x00,           //subclass code
        0x00,           //protocol code
        USB_MAX_EP0_PACKET_LENGTH,     //max packet size for endpoint 0. (SLOW SPEED SPECIFIES 8)
        0xd8,0x04,          //vendor id (0x04D8 is Microchip)
        0x01,0x01,          //product id
        0x00,0x01,          //device release number
```

```
        0x01,                   //index of string description of manufacturer. therefore we point to
string_1 array (see below)
        0x02,               //index of string descriptor of the product
        0x00,               //index of string descriptor of serial number
        USB_NUM_CONFIGURATIONS   //number of possible configurations
    };


////////////////////////////////////////////////////////////////
///
///   start string descriptors
///   String 0 is a special language string, and must be defined.  People in U.S.A. can leave this
alone.
///
///   You must define the length else get_next_string_character() will not see the string
///   Current code only supports 10 strings (0 thru 9)
///
////////////////////////////////////////////////////////////////

//the offset of the starting location of each string.
//offset[0] is the start of string 0, offset[1] is the start of string 1, etc.
const char USB_STRING_DESC_OFFSET[]={0,4,12};

#define USB_STRING_DESC_COUNT sizeof(USB_STRING_DESC_OFFSET)

char const USB_STRING_DESC[]={
    //string 0
        4, //length of string index
        USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
        0x09,0x04,   //Microsoft Defined for US-English
    //string 1
        8, //length of string index
        USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
        'B',0,
        'M',0,
        'E',0,
    //string 2
        46, //length of string index
        USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
        'D',0,
        'a',0,
        'T',0,
        'a',0,
        ' ',0,
        'T',0,
        'r',0,
        'a',0,
        'n',0,
        's',0,
        'f',0,
        'e',0,
        'r',0,
```

```
        ' ',0,
        'B',0,
        'u',0,
        'l',0,
        'k',0,
        ' ',0,
        'N',0,
        'N',0,
        'H',0,
};
#ENDIF
```

Bạn không cần phải chỉnh sửa gì nhiều trong file này, chỉ cần lưu ý đến một số điểm mà tôi đã đánh dấu bằng màu đỏ, tại đó đã có các chú thích bằng tiếng anh rất rõ về ý nghĩa của chúng. Đó là số thiết bị được hỗ trợ giao tiếp, số điểm cuối, việc khởi tạo các đường ống truyền và nhận, cỡ của gói truyền và phương thức truyền. Ở đây tôi truyền theo loại BULK. Đó là những thông số bạn cần quan tâm nhưng không cần sửa.
Các thông số sau là vendor id & product id bạn có thể sửa tùy ý miễn là không trùng với thiết bị đã có trong PC của bạn. Cuối cùng là string index bạn có sửa lại theo tên mà bạn mong muốn, chú ý rằng chiều dài của chuỗi ký tự phải phù hợp với khai báo.
Còn một số khai báo nữa nhưng tôi để vào trong file khác để tiện việc sửa đổi, cụ thể được để trong file *usb_demo_bulk.h*

6. Quản lý file *usb_demo_bulk.h:*

Như đã nói ở trên file này được tạo ra khi ta lập Project trong CCS, bây giờ ta thêm vào trong đó một số khai báo:

```
#include <18F4550.h>
#device adc=8

#FUSES NOWDT                    //No Watch Dog Timer
#FUSES WDT128                   //Watch Dog Timer uses 1:128 Postscale
#FUSES EC_IO                    //External clock
#FUSES NOPROTECT                //Code not protected from reading
#FUSES BROWNOUT                 //Reset when brownout detected
#FUSES BORV20                   //Brownout reset at 2.0V
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOCPD                    //No EE protection
#FUSES STVREN                   //Stack full/underflow will cause reset
#FUSES NODEBUG                  //No Debug mode for ICD
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOWRT                    //Program memory not write protected
#FUSES NOWRTD                   //Data EEPROM not write protected
#FUSES IESO                     //Internal External Switch Over mode enabled
#FUSES FCMEN                    //Fail-safe clock monitor enabled
#FUSES PBADEN                   //PORTB pins are configured as analog input channels on RESET
#FUSES NOWRTC                   //configuration not registers write protected
#FUSES NOWRTB                   //Boot block not write protected
#FUSES NOEBTR                   //Memory not protected from table reads
```

```
#FUSES NOEBTRB              //Boot block not protected from table reads
#FUSES NOCPB               //No Boot Block code protection
#FUSES MCLR                //Master Clear pin enabled
#FUSES LPT1OSC             //Timer1 configured for low-power operation
#FUSES NOXINST       //Extended set extension and Indexed Addressing mode disabled
#FUSES PLL3                // PLL PreScaler 3
#FUSES USBDIV
#FUSES VREGEN
#FUSES CPUDIV1
#FUSES HSPLL

#use delay(clock=12000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)

#DEFINE USB_HID_DEVICE     FALSE
#define USB_EP1_TX_ENABLE     USB_ENABLE_BULK     //turn on EP1 for IN bulk/interrupt transfers
#define USB_EP1_RX_ENABLE     USB_ENABLE_BULK     //turn on EP1 for OUT bulk/interrupt transfers
#define USB_EP1_TX_SIZE    64  //size to allocate for the tx endpoint 1 buffer
#define USB_EP1_RX_SIZE    8   //size to allocate for the rx endpoint 1 buffer

void setup()
{
  setup_adc_ports(AN0|VSS_VDD);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(FALSE);
  setup_wdt(WDT_OFF);
  setup_timer_0(RTCC_INTERNAL);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  setup_low_volt_detect(FALSE);
  setup_oscillator(False);
}
```

Những điểm chú ý tôi đã đánh dấu ở trên, bạn cần lưu ý rằng tần số thạnh anh ngoài sẽ được chia để được dao động 4M đây là yêu cầu bắt buộc khi sử dụng PLL. Ở đây tôi dùng thạch anh 12M nên PLL=3. Một số tham số khác bạn có thể xem chi tiết trong datasheet.

Như vậy đến đây ta chỉ còn một công việc nữa là viết hàm main.

7. chương trình chính:

```
#include "E:\MICROCONTROL\PIC\USB\usb_demo_bulk.h"
#include <pic18_usb.h>
#include <usb_desc_scope1.h>
#include <usb.c>

void usb_debug_task(void)
{
```

```
   static int8 last_connected;
   static int8 last_enumerated;
   int8 new_connected;
   int8 new_enumerated;

   new_connected=usb_attached();
   new_enumerated=usb_enumerated();

   if (new_connected && !last_connected)
     printf("\r\n\nUSB connected, waiting for enumaration...");
   if (!new_connected && last_connected)
     printf("\r\n\nUSB disconnected, waiting for connection...");
   if (new_enumerated && !last_enumerated)
     printf("\r\n\nUSB enumerated by PC/HOST");
   if (!new_enumerated && last_enumerated)
     printf("\r\n\nUSB unenumerated by PC/HOST, waiting for enumeration...");

   last_connected=new_connected;
   last_enumerated=new_enumerated;
}

void main()
{
  int8 out_data[2];
  int8 in_data[2];
  int8 send_timer=0;
  int8 count=0;
  int16 i;

   setup();

   // TODO: USER CODE!!
   printf("\r\n Transfer BULK Example");
   usb_init_cs();

   while (TRUE)
     {
       usb_task();
       usb_debug_task();
       if(usb_enumerated())
         {
           if (!send_timer)
             {
               count++;
               send_timer=250;
               out_data[0]=count;
               if (usb_put_packet(1, out_data, 1, USB_DTS_TOGGLE))
               printf("\r\n<-- Sending 2 bytes: 0x%X", out_data[0]);
             }
           if (usb_kbhit(1))
             {
                usb_get_packet(1, in_data, 1);
```

```
                printf("\r\n--> Received data: 0x%X",in_data[0]);;
            }
        send_timer--;
        delay_ms(1);
        }
    }
```

Cư bản chương trình trên giống với ví dụ của CCS. Trong chương trình sử dụng hàm usb_debug_task() dùng để gỡ rối bằng giao tiếp UART, nếu bạn sử dụng laptop không có cổng COM thì có thể thay chúng bằng việc hiển thị ra LED.

Trong chương trình chính thực hiện cứ 250ms thì truyền qua bus USB lên PC giá trị *count*, giá trị này sau mỗi lần truyền được tăng lên 1, khi đến giá trị 0xFF thì tự động trả về 0. Trong chương trình còn liên tục kiểm tra xem điểm cuối có nhận được dữ liệu từ PC không, nếu có thì lấy dữ liệu trong bộ đệm về biến in_data. Cả truyền và nhận đều được kiểm tra bằng việc hiển thị qua UART. Đến đây ta đã hoàn tất việc viết fimware cho VĐK, ở đây tôi dùng PIC18F4550 việc viết chương trình cho PIC18f2550 không có gì khác.

Chúng ta bắt tay vào việc tạo driver cho thiết bị và viết một chương trình giao diện đơn giản bằng C#.