KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

CAMPUS GROUP T LEUVEN

Microcontrollers LAB 5

USB on the PIC

Coordinator: L. Geurts

Cooperator: T. Stas

.

Academic Year 2019-2020

Target of this lab session

In this lab session we will focus on the settings of the bootloader and the working principle of the USB of the microcontroller as a communication device. On Toledo you can find all files:

- Zipfile with template code for bootloader
- Zipfile with template code for USB
- Zipfile with free-usb-analyze.exe for analyze USB ports or go to https://freeusbanalyzer.com/
- C18 compiler to compile the code of the bootloader.

This lab session is made of several lab exercises named as lab1, lab2,… you need to fill in the correct answer of these assignments on a separate paper and hand over a written example to your coach at the end of the lab.

Bootloader settings

The Lab coach now has burned a not working bootloader on your microcontroller. Now it is up to you to configure it again with the proper settings. On Toledo you can find the bootloader files. Open it in MPLAB.

Lab1

In the main.c you see all configuration bits needed to let the USB work. Change the first five configuration bits (USB at full speed) corresponding to the CONFIG1L and CONFIG1H register of the microcontroller in order to have a 6 MHz (In case you want to use the USB at low speed) USB clock derived from a 48 MHz CPU clock, if you know that later on, you will set the internal oscillator of the PIC18F25K50 to 16 MHz. Use the information of the following table:

Write down your answers on the assignment paper.

PLLSEL	
PLLSEL = PLL4X	4x xlock multiplier
PLLSEL = PLL3X	3x clock multiplier
CFGPLLEN	
CFGPLLEN = OFF	PLL Disabled
CFGPLLEN = ON	PLL Enabled
CPUDIV	
CPUDIV = NOCLKDIV	CPU uses system clock (no division)
CPUDIV = CLKDIV2	CPU uses system clock divided by 2
CPUDIV = CLKDIV3	CPU uses system clock divided by 3
CPUDIV = CLKDIV6	CPU uses system clock divided by 6
LS48MHZ	
LS48MHZ = SYS24x4	System clock at 24MHz, USB clock divider is 4
LS48MHZ = SYS48x8	System clock at 48MHz, USB clock divider is 8
FOSC	
FOSC = LP	LP oscillator
FOSC = XT	XT oscillator
FOSC = HSH	HSH oscillator high power 16MHz to 25MHz
FOSC = HSM	HSM oscillator, medium power 4MHz to 16MHz
FOSC = ECHCLKO	EC oscillator, high power 16MHz to 48MHz, clock output on OSC2
FOSC = ECHIO	EC oscillator, 16MHz, to 48MHz
FOSC = RCCLKO	External RC oscillator, output on OSC2
FOSC = RCIO	External RC oscillator
FOSC = INTOSCIO	Internal oscillator
FOSC = INTOSCCLKO	Internal , output on OSC2
FOSC = ECMIO	EC oscillator, medium power 4MHz to 16MHz, output on OSC2
FOSC = ECLCLKO	EC oscillator, low power <4MHz, output on OSC2
FOSC = ECLIO	EC oscillator, low power <4MHz

Lab2

Go to usbdrv.h. You see some bit settings necessary for USB operations (Line 47). These bits correspond to the UCFG register of the microcontroller. Now go to usbcfg.h and set the first five bits of this register using an OR operation (line 45). To make the USB work at full speed you have to enable the internal pull-up resistors, make the on-chip transceiver active and disable the ping-pong buffers with MODE_PP (line 44).

Write down your answer on the assignment paper

Burn the bootloader on the microcontroller using a PICKIT.

Oscillator settings

Now that the bootloader is configured and burned onto the microcontroller you can open the usb_template.zip from Toledo and open it with MPLAB.

Lab3

Configure the oscillator frequency in the UserInit() function on line 38.
Write down your answer on the assignment paper.

USB Framework

Here, the USB2.0 framework provided by Microchip will be explained briefly. The framework consists of several packages and examples that are used to implement USB device or USB host applications. The classes supported by this framework are Communication Device Class (CDC), Human Interface Device(HID), Mass Storage Device (MSD) and Generic class devices. Furthermore, various data transfer types are possible with this framework such as bulk, interrupt and isochronous modes.

As the main focus of this lab is to implement a USB based communication system between a microcontroller and a PC, there is also a need to use some software in the PC side to handle the sending and receiving of the data. For this purpose, you can choose any program that can support the control of the data transfer. In our lab you can download the tool from the Toledo.

Let's briefly introduce you how the firmware works.

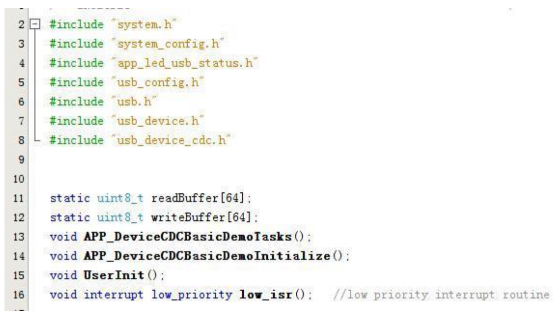
What is it?

The usb_template.zip on Toledo contains microchip USB Framework. Here you can see that in the main on line 25 the function USBDeviceInit() is called. Calling this function will execute the macro SetConfigurationOption. This macro will set the necessary registers for the USB.

Lab4
What are these registers called and what are the settings?
Write down your answer on the assignment paper.

What’s inside?

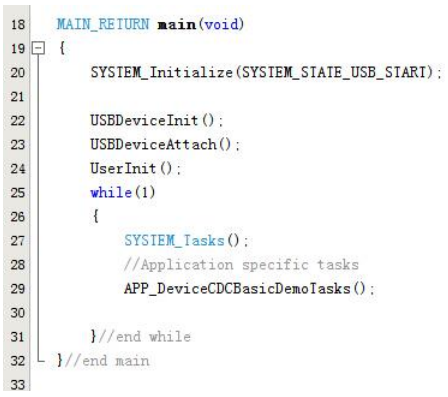
The framework(written in C) has main program which is calling quite some lower level functions. The users should not interfere with these functions. All the user interaction is at the level of main program. First please open the “main.c” file. In the variables section(Fig1,Line 11), you will see two char arrays namely the readBuffer and writeBuffer, both having length 64 bytes. Here, writeBuffer is the array used to send the data to PC and readBuffer is the one used to receive the packages from PC.



```
 2  #include "system.h"
 3  #include "system_config.h"
 4  #include "app_led_usb_status.h"
 5  #include "usb_config.h"
 6  #include "usb.h"
 7  #include "usb_device.h"
 8  #include "usb_device_cdc.h"
 9
10
11  static uint8_t readBuffer[64];
12  static uint8_t writeBuffer[64];
13  void APP_DeviceCDCBasicDemoTasks();
14  void APP_DeviceCDCBasicDemoInitialize();
15  void UserInit();
16  void interrupt low_priority low_isr();   //low priority interrupt routine
```

Figure 1: USB buffers

In figure 2 you can see that the main function (Line 18) first calls the Initialize functions (SYSTEM_Initialize, USBDeviceInit, USBDeviceAttach,UserInit) and calls afterwards the function APP_DeviceCDCBasicDemoTasks() in an infinite loop (while(1)) and just before that, it performs some USB tasks.

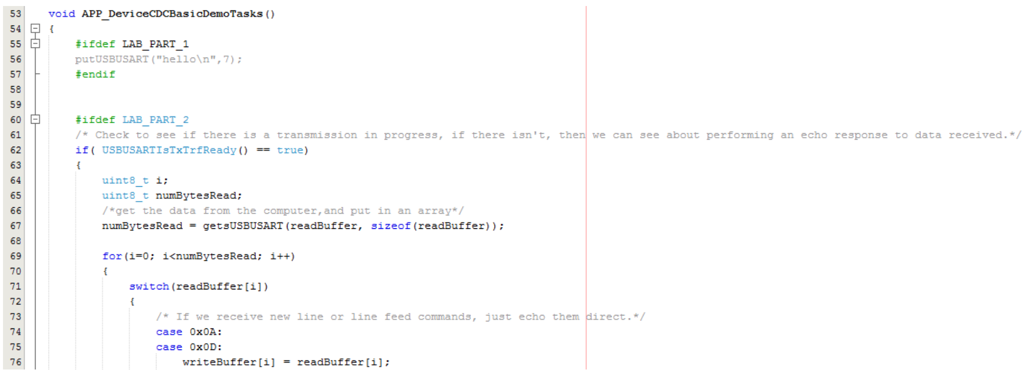


```
18  MAIN_RETURN main(void)
19  {
20      SYSTEM_Initialize(SYSTEM_STATE_USB_START);
21
22      USBDeviceInit();
23      USBDeviceAttach();
24      UserInit();
25      while(1)
26      {
27          SYSTEM_Tasks();
28          //Application specific tasks
29          APP_DeviceCDCBasicDemoTasks();
30
31      }//end while
32  }//end main
33
```

Figure 2: Main function

PC sending Data to the PIC

First check if the USB is ready to start the transfer (if(USBUSARTIsTxTrfReady() == true)). Then the function(getsUSBUSART(readBuffer, sizeof(readBuffer))) returns the number of received bytes. And Data is available in the readBuffer.



```
void APP_DeviceCDCBasicDemoTasks()
{
    #ifdef LAB_PART_1
    putUSBUSART("hello\n",7);
    #endif


    #ifdef LAB_PART_2
    /* Check to see if there is a transmission in progress, if there isn't, then we can see about performing an echo response to data received.*/
    if( USBUSARTIsTxTrfReady() == true)
    {
        uint8_t i;
        uint8_t numBytesRead;
        /*get the data from the computer,and put in an array*/
        numBytesRead = getsUSBUSART(readBuffer, sizeof(readBuffer));

        for(i=0; i<numBytesRead; i++)
        {
            switch(readBuffer[i])
            {
                /* If we receive new line or line feed commands, just echo them direct.*/
                case 0x0A:
                case 0x0D:
                    writeBuffer[i] = readBuffer[i];
```

Figure 3: Example of USB transfer

PIC sending Data to PC

Data should be stored in the write Buffer.
Use function (putUSBUSART(writeBuffer, numBytesRead)) where “numBytesWrite” is the number of bytes to transmit to send to PC.

IMPORTANT NOTICE

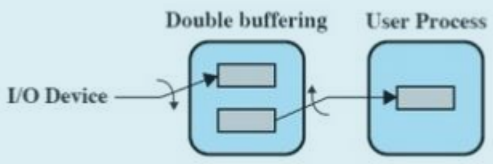
1. The RB0 and RB1 pins are reserved for the system to indicate the status of the usb device. Please make sure in your program, don’t touch those two bits.

2. The USB program controls the flow of the “main” program. That “main” program is in an endless loop, so that all USB activities can be executed repeatedly. The user function is called by the “main”. That user function should NOT contain an endless loop, else the USB functionality will be suspended.

3. For additional code like AD conversions and timers, use the low priority interrupt. The high priority is used by the USB.

Side information: Ping-Pong Buffer

A Ping Pong Buffer is a kind of a circular buffer. More specific it is a double buffer used to enhance the speed of a device that overlaps the I/O tasks with the data processing tasks. In general, in a ping pong communication, the ping is the transmission of the packet to the opposite host and the pong is the response from the other host. One buffer is used to store a block of data in a way that a reader will see a complete (old) version of the data, while in the other buffer a writer is creating a new data set. When the new block of data is complete, the reader and writer will switch their pointers to the two buffers. As a result, the usage of ping pong buffer increases the overall throughput of a device.



Lab5

In the code there are two defined compilations at line 10 and 11. In PART1, the PIC18 will receive an positive number from the PC, increments it by one and sends it back to the PC. Use the ‘Free Device Monitoring Studio’ to answer the following questions.

- How many packets are sent during one transaction between PC and microcontroller?
- What type of ‘usb transfer’ has occurred?
- How many byte is your TransferBuffer?
- Is the TransferBuffer for sending and receiving the same?
- What is the address of your pipehandler?
- Is the pipeHandler for sending and receiving the same?
- How is your payload sent over USB?
- What is your vendorID and procesID if you now that both ID’s have a size of 2 bytes.

Lab6

If you disable part1 and enable part2 in the code then you see that the microcontroller is continuously sending data to the PC. Answer the following questions:

- What is the speed (How many packets/second are sent)?
- How many bytes/packet are sent?
- Is it good practice to send floating points? Explain your answer.
- Measure the frequency on the datalines of the USB. What do you measure? Explain.