

Tutorial 1: Implementation of a USB based PIC-to-PC communication

I. INTRODUCTION

The objective of this tutorial is to introduce some basic free software tools used for intermediate PIC programming. Supplementary information is given on programming PIC's which have USB hardware support. The tutorial focuses on the installation, integration and settings of these tools and at the end of the document, an example application with 18F4553 is provided to demonstrate the PIC-to-PC communication via USB.

Prerequisites:

Familiarity with C programming is required. Object oriented programming background might be helpful in understanding the examples.

II. Firmware Tools

The following device programming software are sufficient and effective tools for PIC firmware implementation. These will be briefly explained in the subsections.

A. MPLAB IDE

MPLAB[®] IDE is a powerful firmware development tool which supports many compiling options and debugging interfaces hence provides a flexible and structural development environment for the programmer. It can run on Windows[®] platform as a 32-bit application. The environment is simply a notepad application which is assisted by supplementary features. The tool is freely provided by Microchip[®] and its final release can be downloaded from: www.microchip.com (can be found at the design section of the page). Previous versions can be found at the archive page:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023073

Here, the installation of the version 8.43 of MPLAB will be explained. The installation procedure can differ according to the final release of the tool. Therefore, consult the relevant documents before installing the final versions.

1) Download the MPLAB 8.43 from the webpage¹:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

2) Extract the .zip package to a folder

3) If there is a previous installation of the software (older than 8.43), remove it prior to installation.

¹ If the pages are updated, you can find the final versions (as well as the previous releases) of the software tools presented in this document from the relevant sections of the microchip webpage or archives.



- 4) In the extracted folder, run setup.exe with an administrative profile and follow the recommended remarks, accept the license terms, select complete installation etc.
- 5) At this stage, MPLAB will validate your installation which will take about 10-30 minutes depending on how you follow the instructions. If you forgot to remove the previous installations, the program will first try to uninstall those configurations.
- 6) After a while, the installation program will ask you to install the HI-TECH C compiler which is the default compiler for PIC10/12/16 microcontrollers to be used within MPLAB. Select Yes to install the final version of the compiler (9.70 is the final release of this compiler for now).



Figure 1: Installation of HI-TECH C Compiler

- 7) Click Next, accept the license agreement, note on the installation directory, check the "Add the environment path" option.
- 8) Restart your computer

After the installation of MPLAB IDE is complete, you can see the documentation related to various tools and patches used in MPLAB.

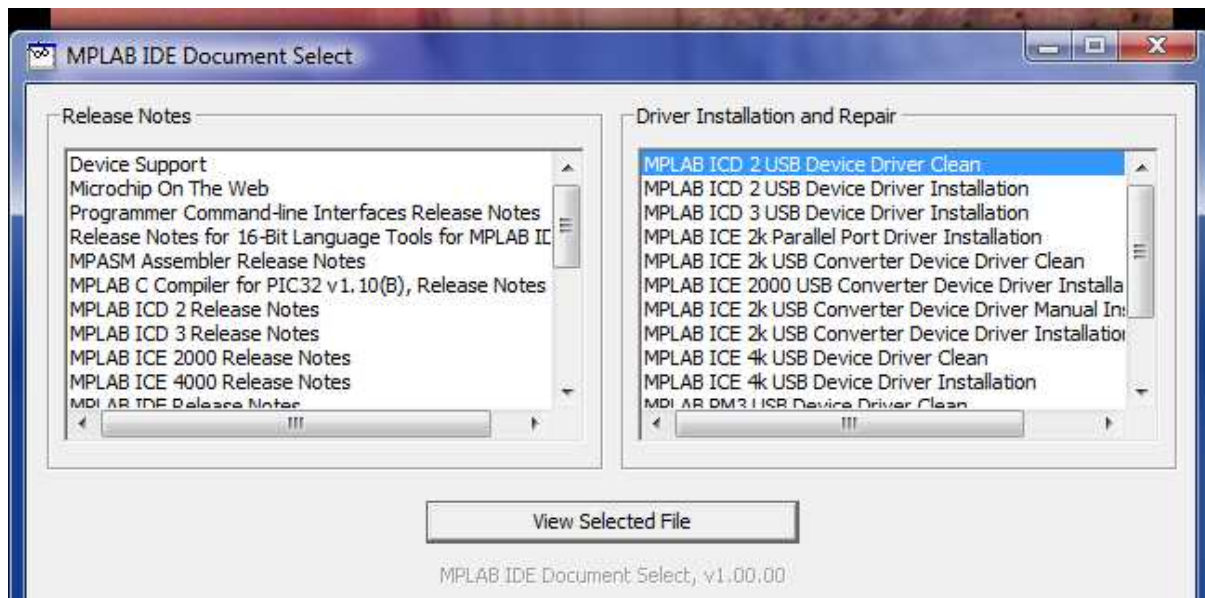


Figure 2: Release Notes and Installed Tools for MPLAB

If you have any problems with the installation steps, refer to the documentations of MPLAB from its webpage. From the webpage, you can also find useful links on writing codes in C with example applications documented by MPLAB communities and forums, describing the application procedures (hardware configurations, compiling options) as well (http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1408). In this document, only the procedure regarding the implementation of a simple PIC-to-PC communication via USB is presented. You are encouraged to refer to the documentations, examples and forums of the software/hardware publishers in the implementation steps of task-specific applications (A to D conversion, PWM, Timers etc).

B. MPLAB C18 Compiler

MPLAB[®] C18 compiler is one of the compiling tools for programming PIC18 MCUs. It is supported by MPLAB IDE as a compiler; hence very little effort is needed by the programmer to compile the source code. There are also other C compiler tools of MPLAB for programming 16 and 32 bit PIC microcontrollers (PIC24 and PIC32). However, only the C18 compiler is considered in this tutorial as it is the primary tool for programming the most commonly used PIC18 devices in the market (PIC18F4550, PIC18F2550, PIC18F4450 etc.).

The tool was known as MPLAB C18 Compiler Student Edition and was able to support code optimizations for 60 days. Lately, it was upgraded to separate final (free) releases in which the code optimizations are limited for academic use. Free versions of the C18 compiler with code optimizations for 60 days are available on the webpage (Download the standard evaluation version):

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en010014.

One can find the installation, upgrades and the **SUPPORTED DEVICES**² in the release notes of the C18 compiler: <http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB-C18-README.html>. However, in order to perform accurate installation, some of the key points during this procedure are worth to be underlined here.

- 1) Download the standard 60 day evaluation version (v3.34) from the link above, run the program with an administrative profile.
- 2) Specify the installation directory. It is highly recommended to define the directory as C:\MCC18
- 3) Select all the options as in Figure 3:

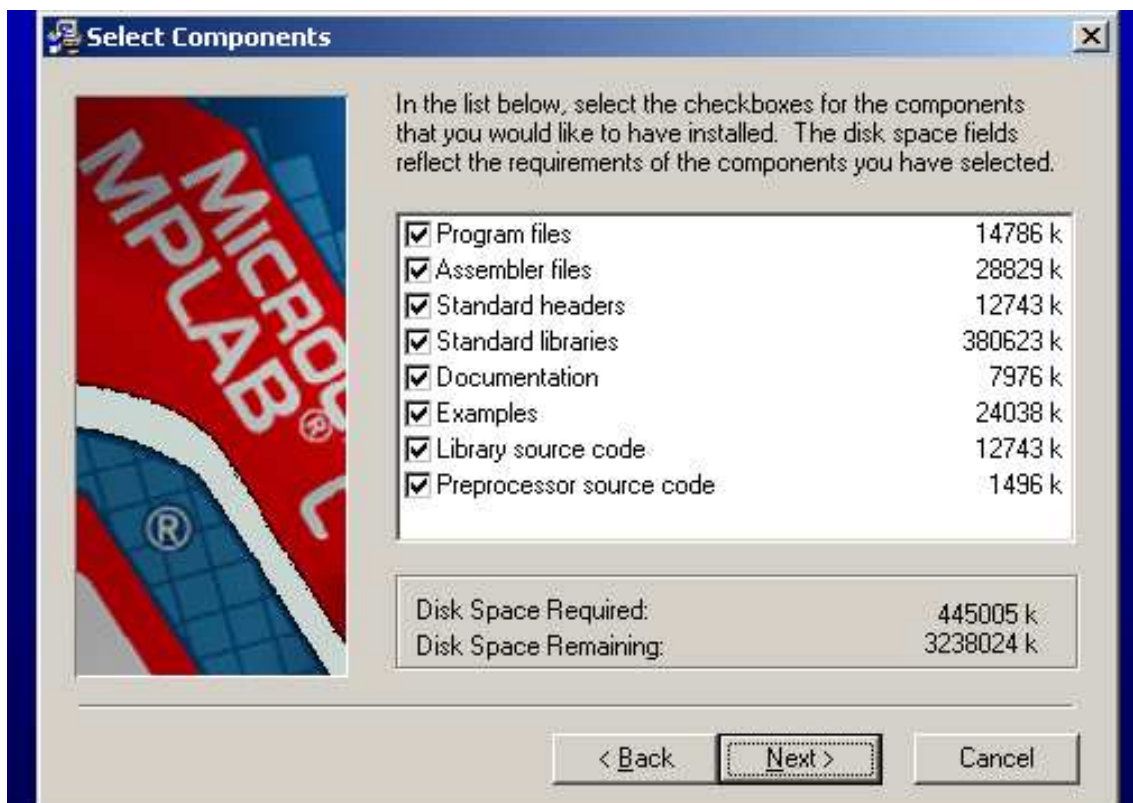


Figure 3: Installation of C18 Compiler, assigning the components to be installed

- 4) Select all the inclusion options for the environment variables and path as in Figure 4.

² The supported devices are important in selecting the compiler (or vice versa). C18 Compiler does not support the commonly used PIC16F877 or PIC16F84 devices. However there are other compilers like CCS C compiler or (HITECH C as a free one) which can be used to program these. These compilers can also be integrated to MPLAB environment, the integration procedure of which may be the topic of other tutorials for this course.

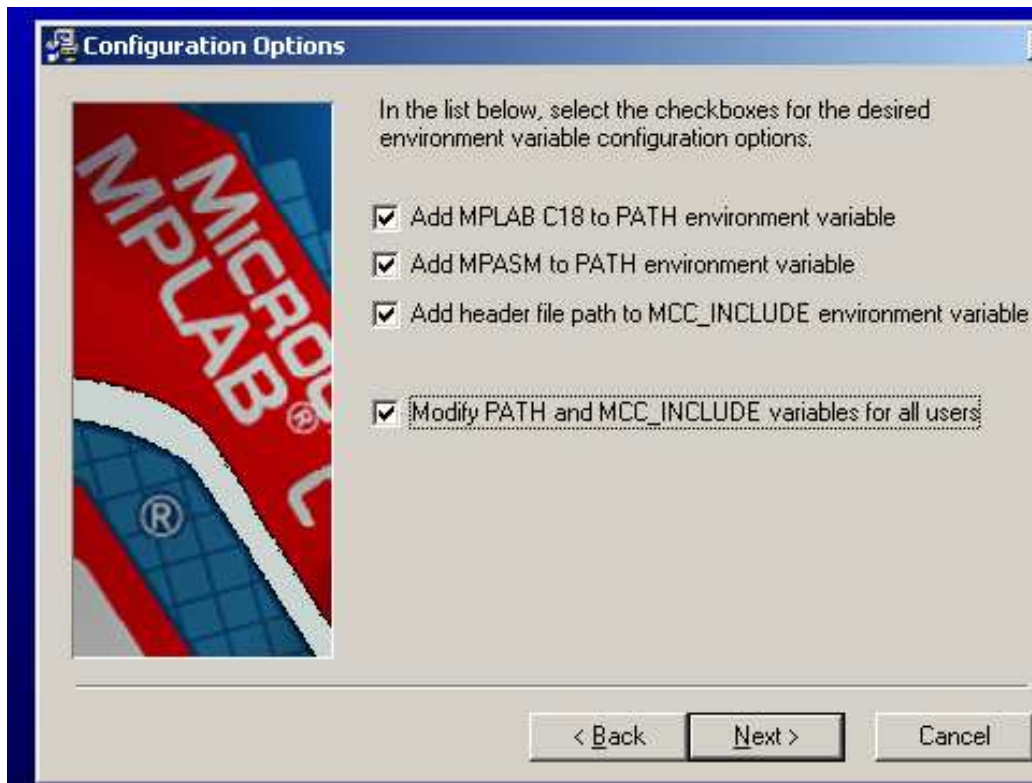


Figure 4: Installation of C18 Compiler, settings for path and env. variables

- 5) Select all the options to configure MPLAB to fully manage the compiler (Figure 5).

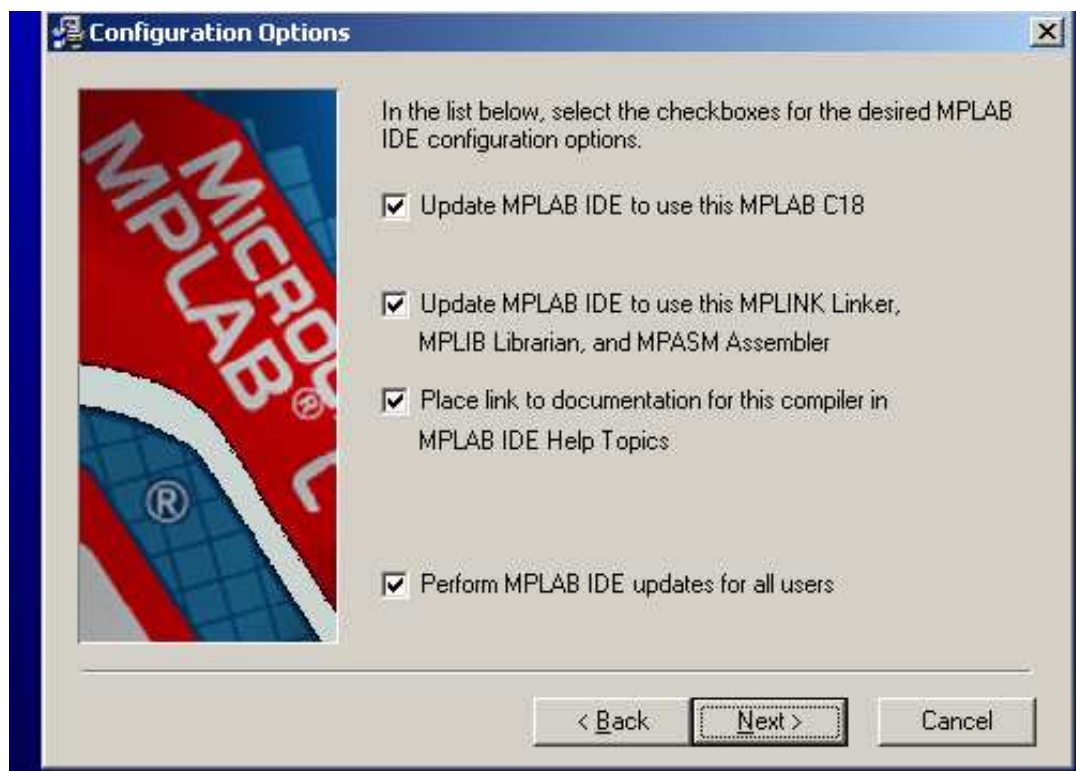


Figure 5: Installation of C18 Compiler, settings for MPLAB

Once the installation is complete, you might have to restart your system. The documentation related to the libraries of the C18 compiler can be found at: http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_C18_Libraries_51297f.pdf. Previous releases can be found from the archives of the site.

C. Microchip FS-USB Framework

With the growing technology, nearly all personal computers (especially the portable ones) prefer using USB communication with the peripheral devices. The reason is that USB has some advantages as compared to the other communication protocols (RS232, RS485, parallel communication) such as high bandwidth in transfer, serial communication etc. Therefore, with the increase in the need for USB communication, devices (MCUs or MPUs) that have built-in USB hardware (device or host) became available in market.

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. The reader can find detailed information on USB2.0 at <http://www.usb.org/developers/docs/>

It is worth mentioning here that the Microchip USB framework to be described supports Full Speed data transfer. In other words, the maximum communication speed that can be achieved with these devices and framework is 12Mbps/s. Furthermore, the framework is supports PIC24F and PIC32 as well as PIC18 devices. Therefore, one should consider the bandwidth and device requirements of a specific application before choosing the framework as a software tool.

The Microchip FSUSB framework v2.6 (final release at the moment) is available at:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537044. (You can also find the previous releases at the archive page of microchip)

You can download it for free and easily install it in your system (it is not an installation actually; only extract the package in a specific directory). You can find a detailed explanation of the framework (for each example application it contains) in the documentation.

III. PC-side Application - Software Tools

As the main focus of this tutorial 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 as well as the firmware. For this purpose, you can choose any program that can support the control of the data transfer. Furthermore, you can develop the source code by yourself using the functionalities of the development environment. Some examples of these environments are Matlab, Visual Studio, Visual Studio.NET and Borland.

The programming environment chosen to implement the USB application example in this tutorial is Microsoft Visual Studio.NET 2005 Professional Edition, the installation package of which can be found at:

ftp://ftp.cc.metu.edu.tr/PackagePrograms/Microsoft/Visual_Studio_Pro_2005_English/.

It is provided by our university to METU students to develop .NET based applications for free. But as mentioned before, one can implement nearly the same applications using other programming environments.

IV. Application Example: Communication Device Class

As a simple application, the implementation of a communication device class is given as an example in this tutorial. The application that will be discussed here is based on the Microchip USB framework, the examples of which are located in your installation directory. Assuming that the Microchip framework is extracted to C:\Microchip Solutions, the example we will be going over here is the USB Device – CDC – Basic Demo which basically is an application emulating the COM port over USB.

Before the start, it will be beneficial for you to understand the basic idea of bootloading. For this reason, please refer to the tutorial related to "In Circuit Serial Programming and Bootloaders" which is available in the course webpage.

For the moment, it is assumed the device PIC18F4553 is available for the project. This device is supported by all the firmware tools described above. However, the examples in the Microchip Solutions directory are provided for the device 18F4550 (and some other PIC devices) which is a more commonly used PIC USB device (and more available one in the market). Both devices belong to the same family with slight differences in their hardware. For example PIC18F4553 supports 12bit A/D conversion whereas PIC18F4550 has 10bit A/D converter etc. (See their documentations or datasheets).

To begin with, there are 3 folders in the Microchip Solutions/USB Device – CDC – Basic Demo directory, namely the firmware, inf (the driver file) and PC Software Example. First



enter the firmware directory. You will see many MPLAB projects prepared for different PIC devices with several programming hardware options. For our example, you will select the one named USB Device-CDC-Basic Demo-C18-PICDEM FSUSB.mcp (Be careful, you will select the file with .mcp extension). Open the file, you will see the project on the MPLAB environment (Figure 6).

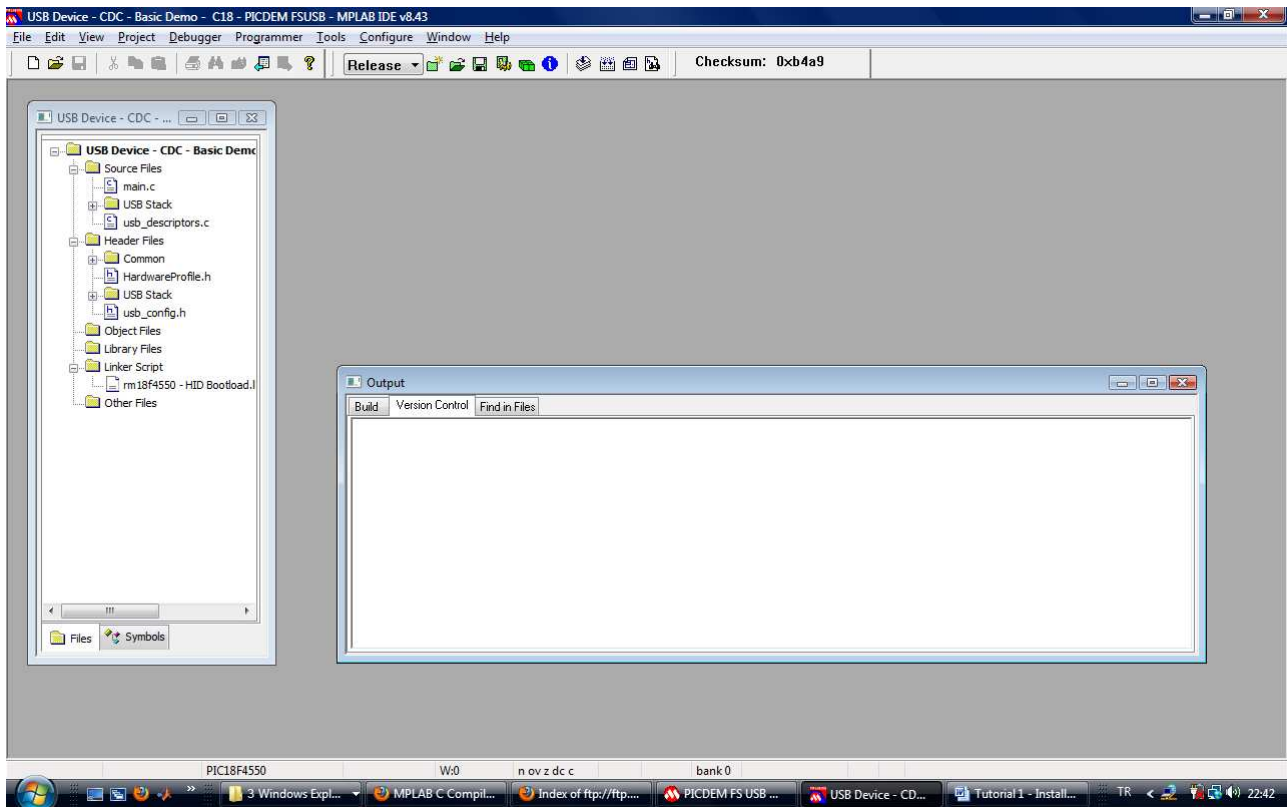


Figure 6: User Interface of MPLAB IDE

Settings for MCHP USB Bootloader and Device Selection

You can see the project files and their organization in the described firmware framework on the Project window (the one on the left of the screen in Figure 6. If it is not seen, select View -> Project). Usually, it is better to work in an ordered workspace. Therefore, you can make the windows dockable by clicking on the icons on top-left corner (Figure 7).

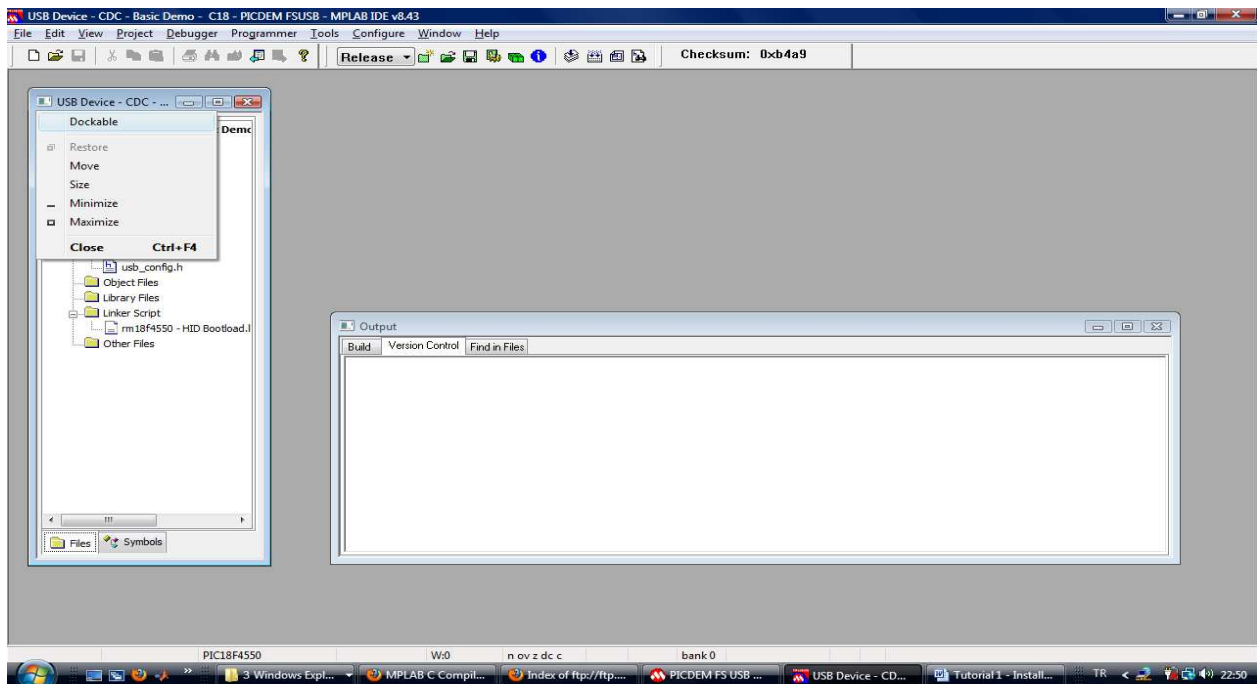


Figure 7: CDC Example, configuring your workspace

First of all, this firmware is prepared to run with the HID bootloader (see the tutorial about bootloaders). To change the project to work with the Microchip (MCHPUSB vendor) bootloader, you should change the linker script (rm 18f4550 – HID Bootload.lkr). Right click the linker script folder and select add files.

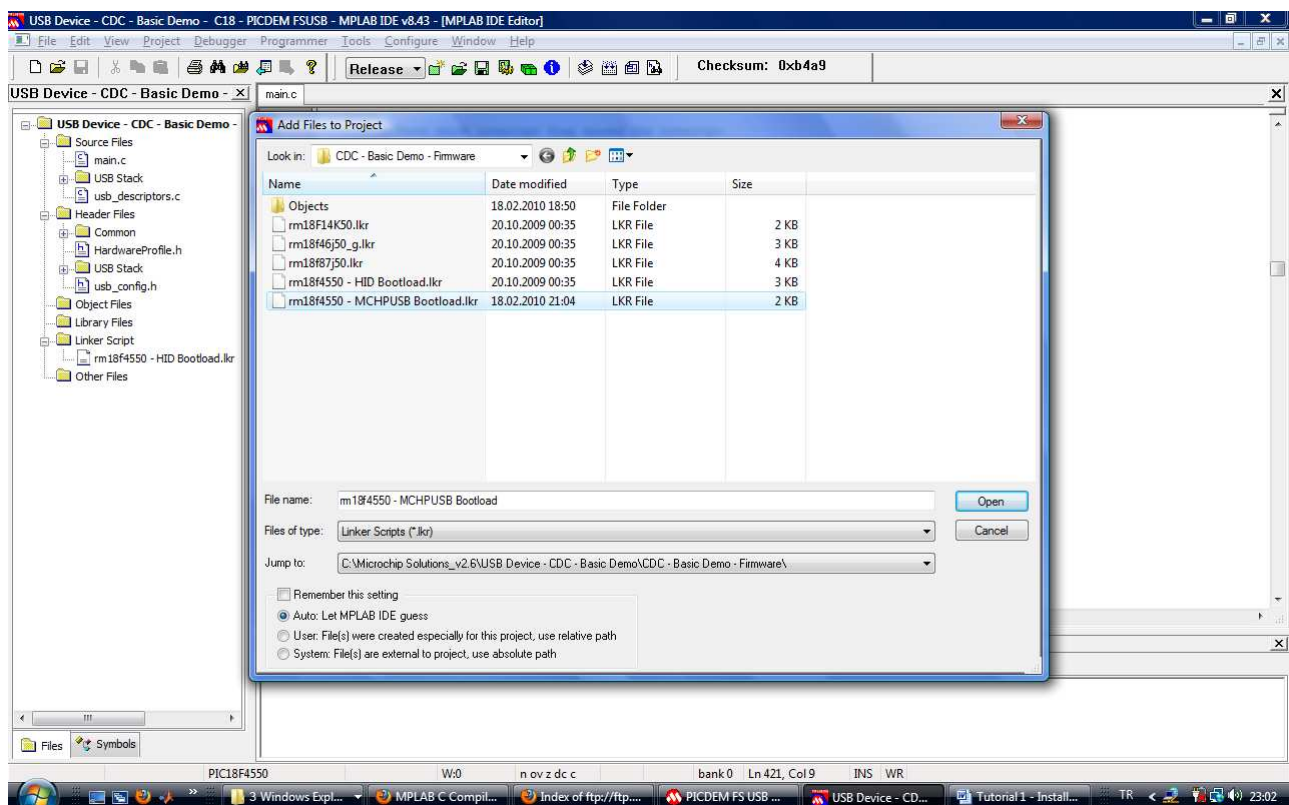


Figure 8: CDC Example, bootloader linker file settings

Select rm18f4550 – MCHPUSB Bootload.lkr -> Open. Now you have two linker scripts. One of them should be removed. Right click on HID Bootload linker script -> Remove.

Now, as we are working with 18F4553 (not 18F4550) we should do some modifications on the MCHPUSB linker script. Open the rm18f4550 – MCHPUSB Bootload.lkr file by double clicking on it. Change the line containing p18f4550.lib as p18f4553.lib, save the file by ctrl+S. Right click on the workspace containing the codes for the linker file (Figure 9)

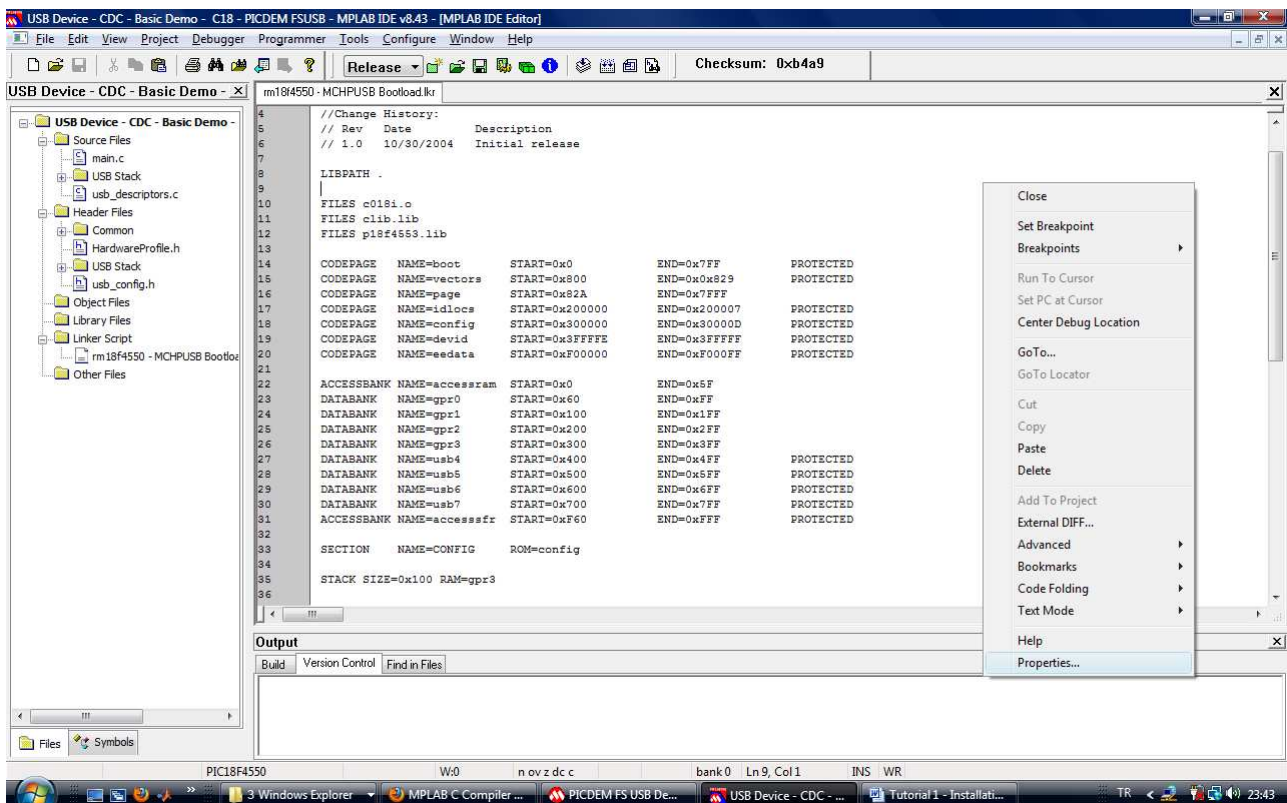


Figure 9: CDC Example, linker script modification

Click Properties. Select Use tabbed window to see all open documents in one page.

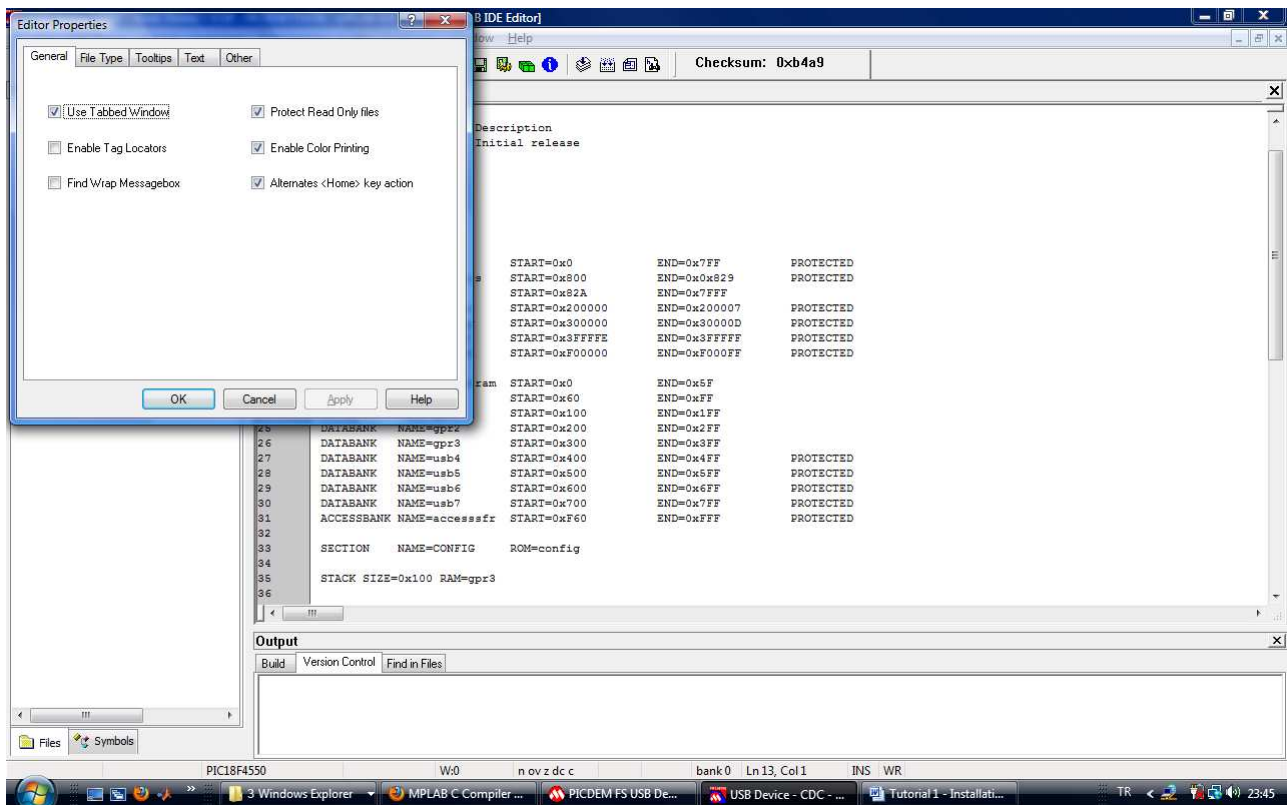


Figure 10: CDC Example, workspace configuration

Select Ok. As you can see on the linker script, it contains some information on the memory of the device you are programming. There are inaccessible regions for protecting the internal variables used in the USB framework. Other unprotected regions are spared to the programmer for the static variables and application code memory. It is highly recommended not to change the region declarations in this file.

Click on Project on the Menu bar and choose "Select Language Toolsuite". Make sure that the active toolsuite is Microchip C18 and check the assembler, linker, compiler and librarian directories. Click Ok.

Open HardwareProfile.h from the project window by double clicking on the file. Find the line containing #include "HardwareProfile - PICDEM FSUSB.h".

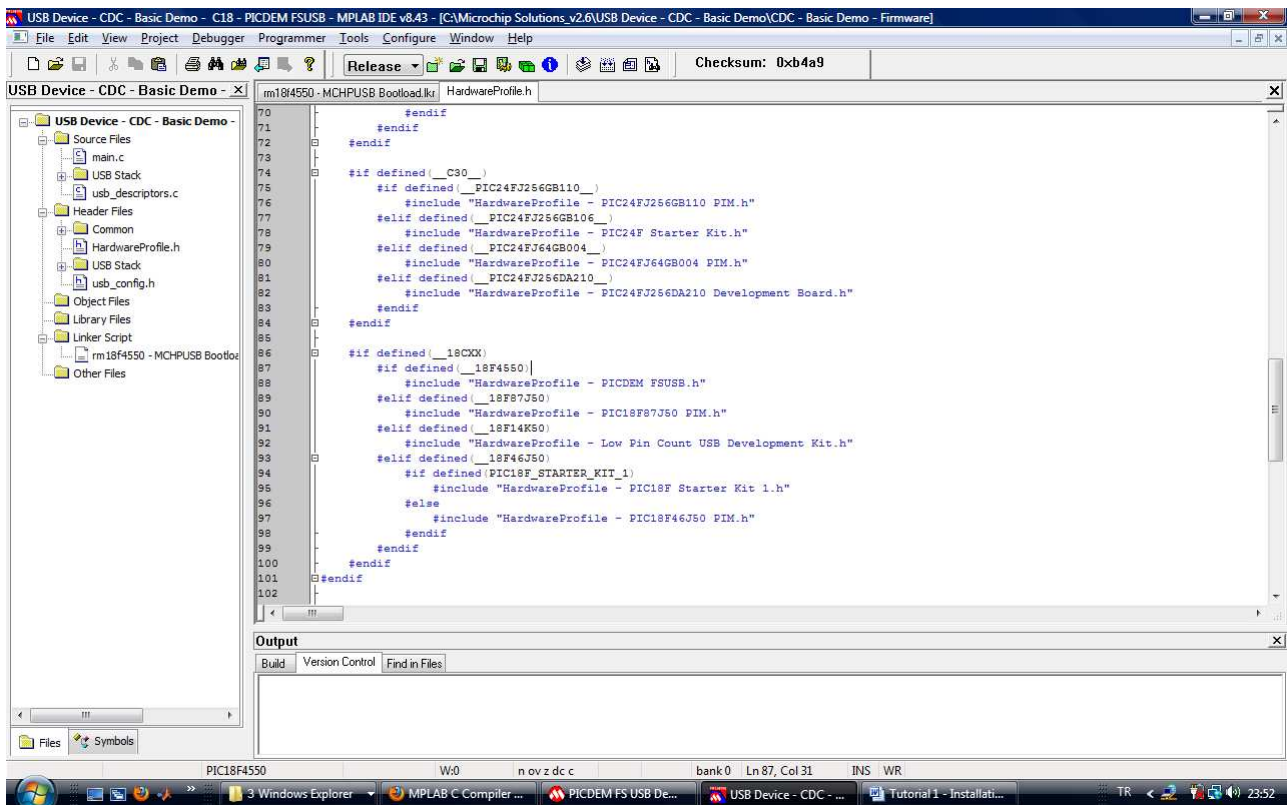


Figure 11: CDC Example, modification on "HardwareProfile.h"

Change the above line to `#if (defined(__18F4550)|| defined(__18F4553))` to include the 18f4553 device into the framework. Right click on the **line** containing "HardwareProfile - PICDEM FSUSB.h" (Figure 12)

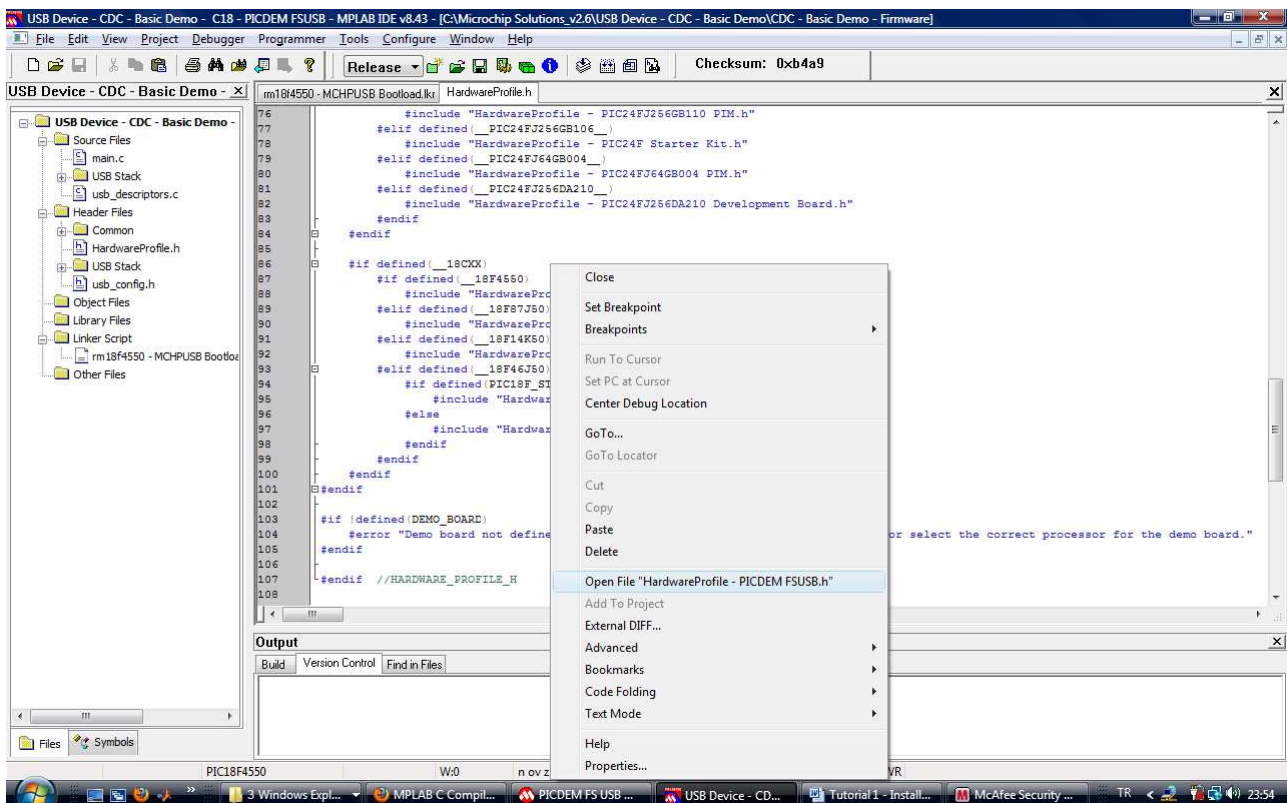


Figure 12: CDC Example, accessing "HardwareProfile – PICDEM FSUSB.h"



Select Open File "HardwareProfile – PICDEM FSUSB.h". The reason to do is that we should change the programming hardware profile to work with the MCHPUSB bootloader. You might have noticed that this file is not included in the project view. So there is no other way to access this document. However, you can include it to your project and access that file more easily. In that file, find the line containing #define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER. Comment that line and uncomment the line #define PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER which is on top of that.

Now, almost everything is ok and one final setting is left for the firmware. Click on Configure on the Menu Bar and choose "Select Device".

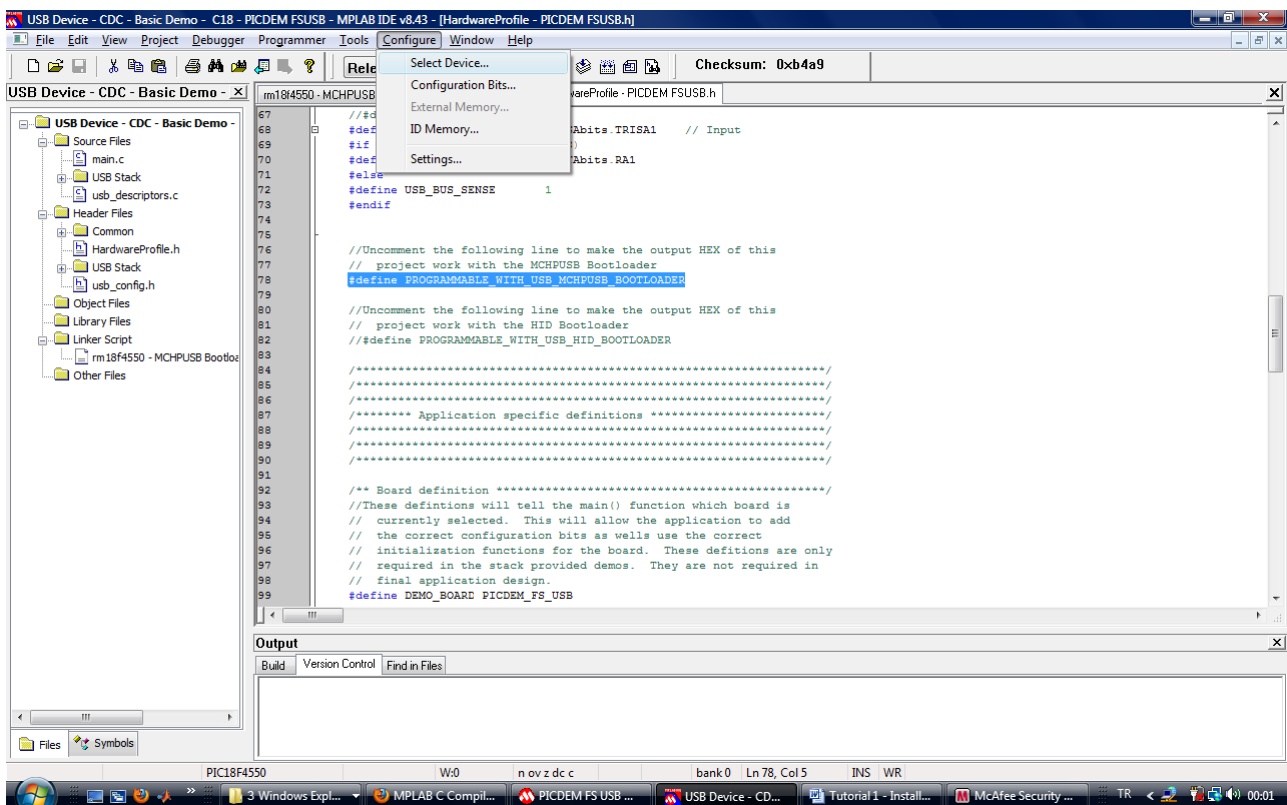


Figure 13: CDC Example, device selection

Change your device to PIC18F4553. Select Ok.

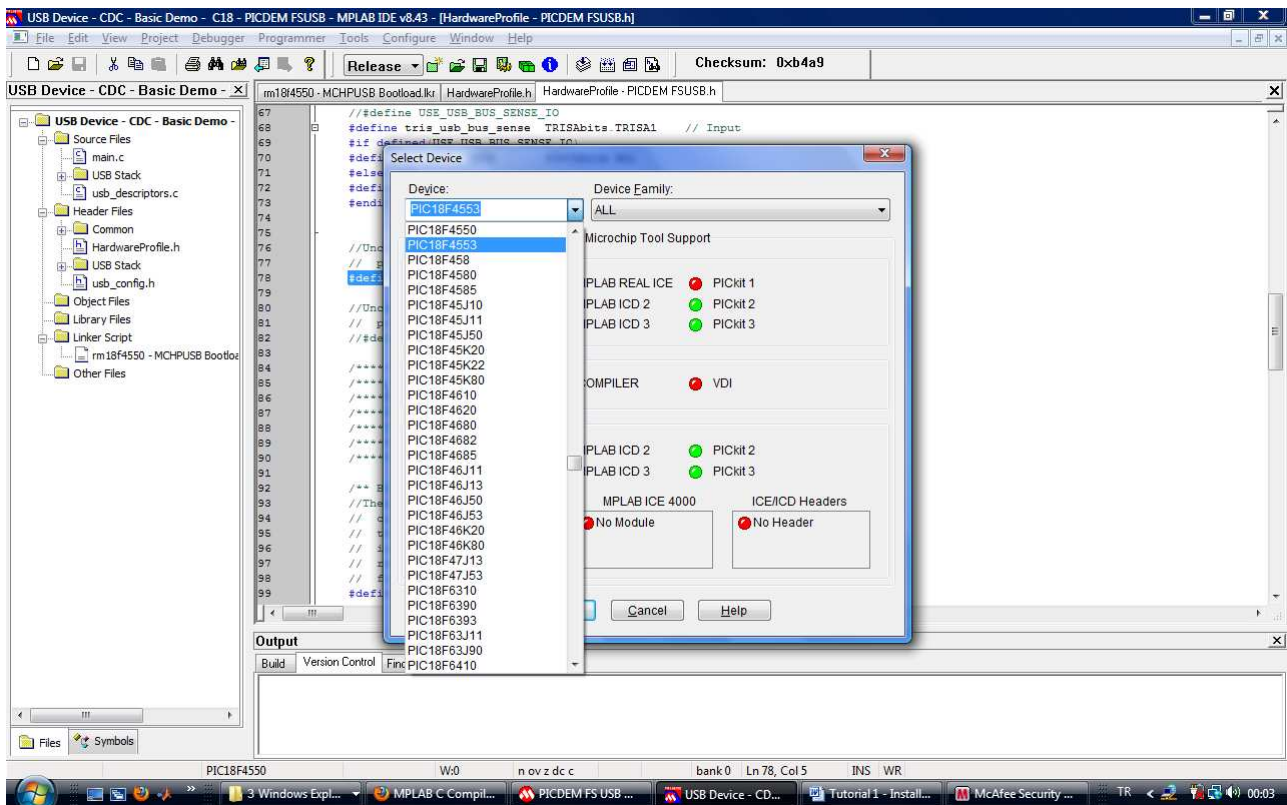


Figure 14: CDC Example, Changing the device

As mentioned before, this default project example has been prepared for the device 18F4550. If you prefer 18F4550 in your application, you don't need to do such device selections in the default application. Furthermore, if you prefer HID bootloader, you don't need to do the changes in the linker script and hardware profile.h files.

Now, click on Build All to build the project . If you had done everything right, you will see "BUILD SUCCEEDED" on the output window which means that everything goes well 😊.

Brief explanation of the firmware example

To summarize what we have done up to this point, we changed the framework example to work with the PIC18F4553 device and modified the codes to make our device be programmable with the MCHPUSB bootloader. Now, let's dig into the example a bit and trace some codes to understand the operation of this example and organization of the firmware framework. Open the file main.c. You will first see some codes related to the default configuration of the supported devices and their programming environment in this file. Some of these are code protection, watchdog timer, PLL division etc. Just pass them and stop when you see the included header files, global variable and function prototype declarations.

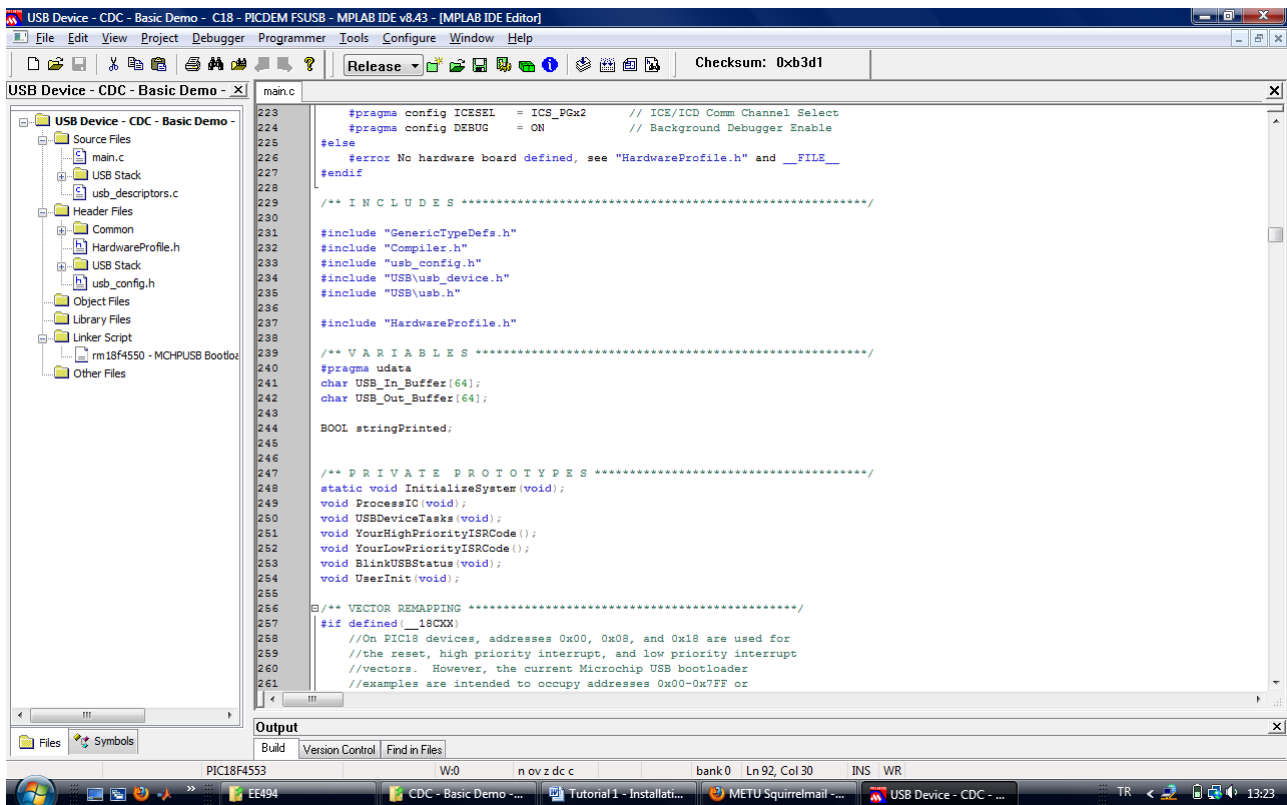


Figure 15: CDC Example, understanding the firmware

In the variables section, you will see two char arrays namely the USB_In_Buffer and USB_Out_Buffer, both having length 64. Here, USB_In_Buffer is the array used to send the data to PC and USB_Out_Buffer is the one used to receive the packages from PC (Don't get confused, the names In and Out refers to the PC side in this framework). There is also a variable named stringPrinted of type BOOL (later, you will see what it does). Now, go below and find the main function.

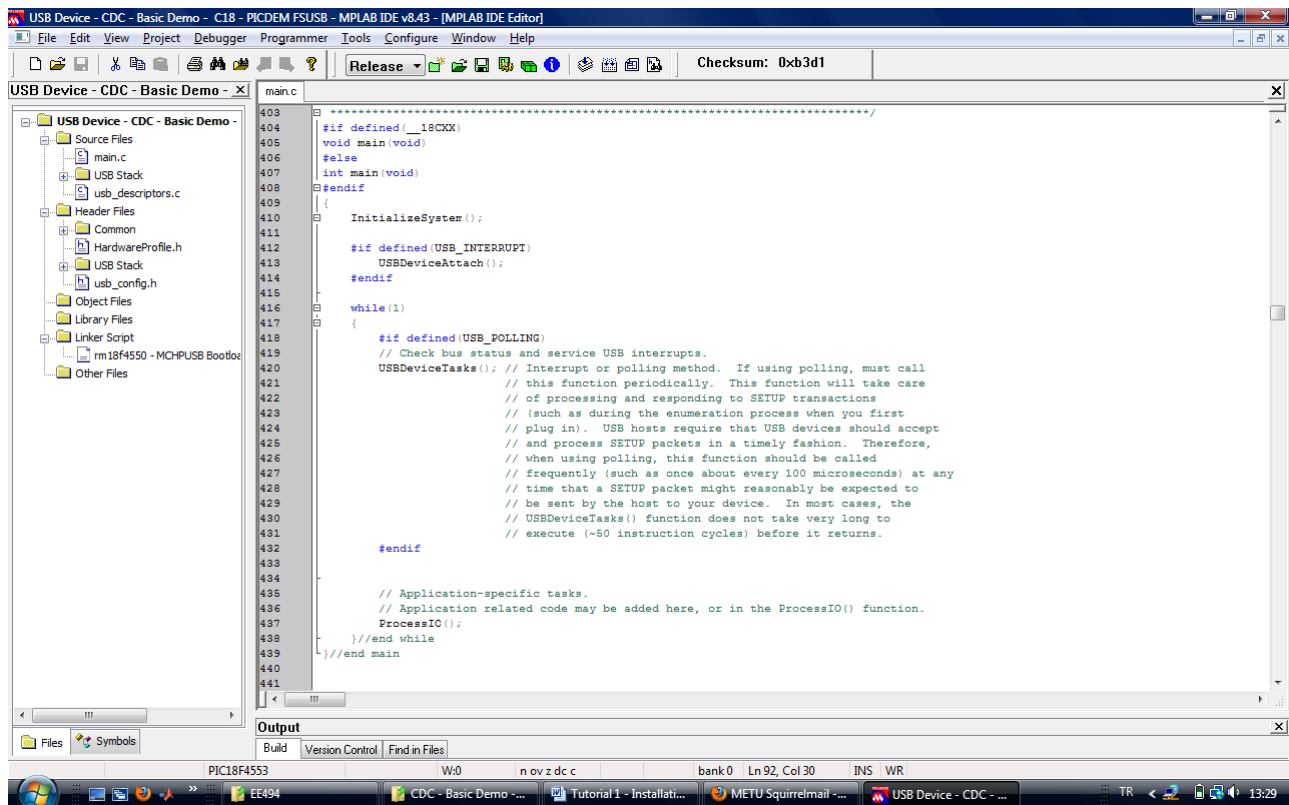


Figure 16: CDC Example, main function in the firmware

Here you can see that the main function calls the function `ProcessIO()` in an infinite loop (`while(1)`) and just before that, it performs some USB tasks (You can have a look at those functions by searching their declarations with `ctrl+shift+F`). Now, find the declaration of the function `ProcessIO()` in which the USB transfer operations are performed.

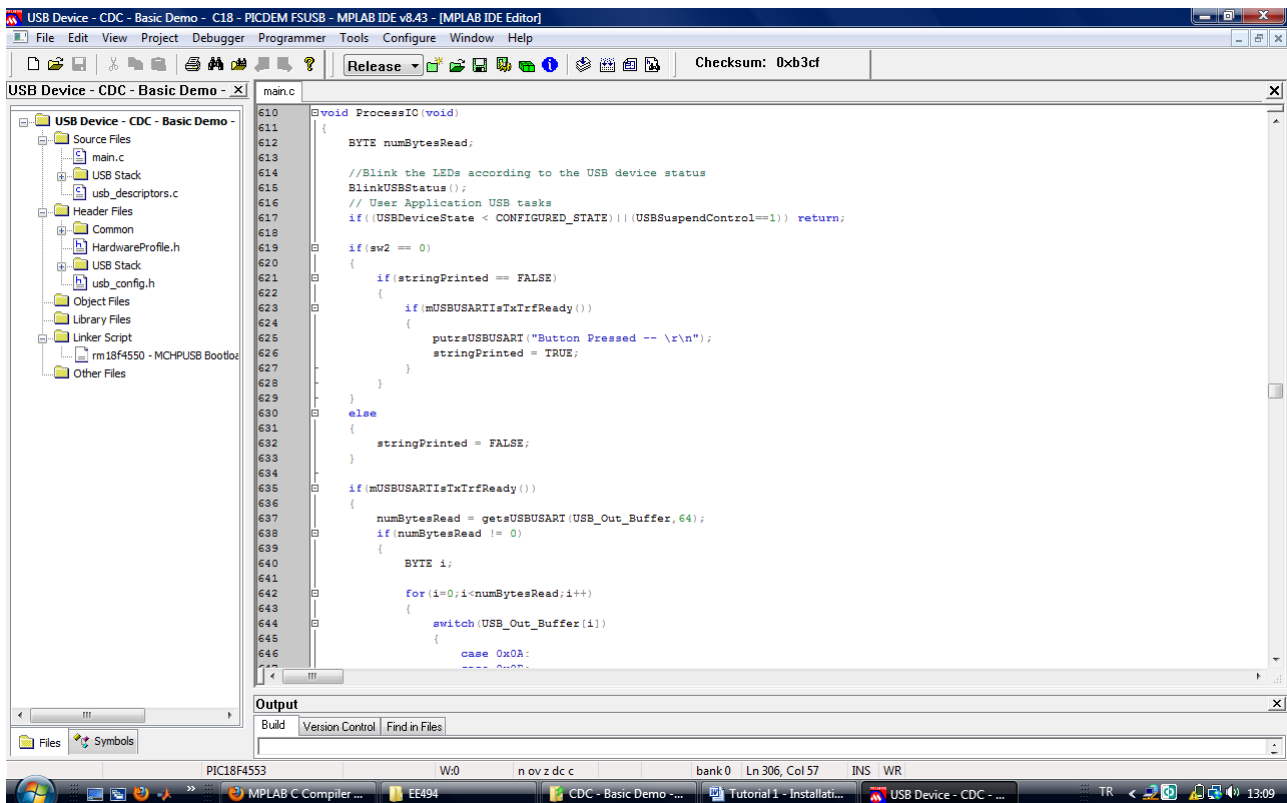


Figure 17: CDC Example, input and output functions in the framework

Here, you will see that, the function `BlinkUSBStatus` is called first in which the two LEDs connected to pins D0 and D1 are blinked for demonstrating the USB status (see the definitions of the USB states used in this framework in the function definition). After that an if statement is given for the switch 2 (you can find the definition of `sw2` in "HardwareProfile - PICDEM FSUSB.h"). You can easily figure out that when this switch is pressed a "Button Pressed" string is sent to the PC. The global variable `stringPrinted` is used to disable this event being fired more than once when pressed.

After that, the main if statement controlling the I/O operations comes. When the USB is ready for the transfer (`if(mUSBUSARTIsTxTrfReady())`), the device prompts (`getsUSBUSART(USB_Out_Buffer,64)`) whether there is any data sent by the PC. When the number of received bytes is larger than 0, the received data is incremented by 1 (except 0x0D case) and sent back to PC (`putUSBUSART(USB_In_Buffer,numBytesRead)`).

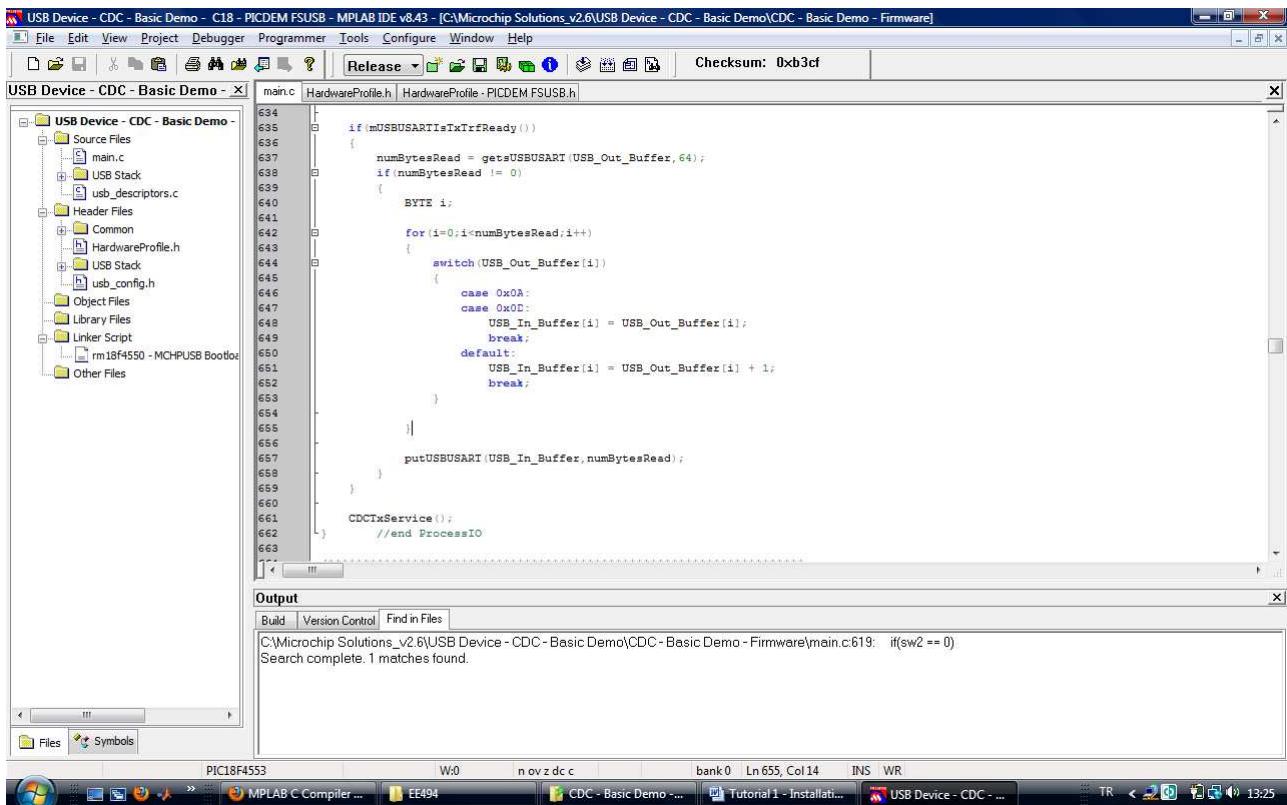


Figure 18: CDC Example, understanding the firmware operation

Setting Up the Hardware

For this example, you should construct the following circuit which consists of minimal discrete components and your microcontroller. Again here, it is assumed that the bootloader firmware was previously programmed to the device (see the tutorials on bootloaders).

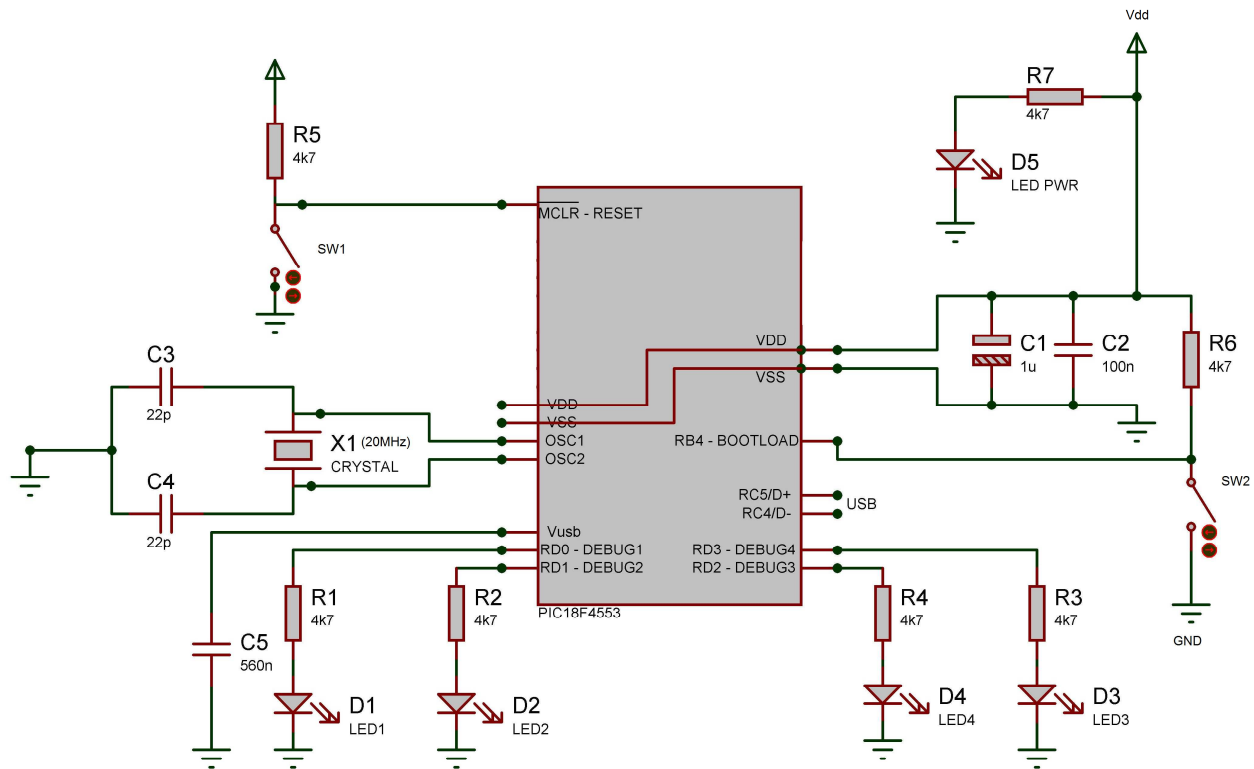


Figure 19: The primitive hardware used in the framework

The crystal used in this circuit is switching at 20MHz. However, as you dig into this firmware, you will see that the speed required for USB transfer is 48MHz. Therefore, the speed is increased by internal PLL generators (see the configuration codes in main.c) to run the device at 48MHz. Hence, you can use any crystal working at different speeds provided that you change the PLL settings accordingly (This PLL property is only valid for USB PIC devices; you may not do it for example with PIC16F877).

The most important issue you should pay attention to is that you are going to supply your circuit from your PC via USB port. Therefore, **be careful when you prepare the wiring to connect your device to the port. You may cause a permanent damage to your computer. You should connect the Vdd and GND of your circuit in Figure 19 to appropriate pins of the USB connector.** By the way, do not confuse the Vusb pin of the PIC18 device with the supply pin of the USB bus; they are not the same node. You only place a capacitor (560nF) to that pin of your USB PIC device.

In Figure 20, you can see the picture and the pin diagrams of the horizontal-A type male USB connector (for the device side) and its socket (for the PC side).

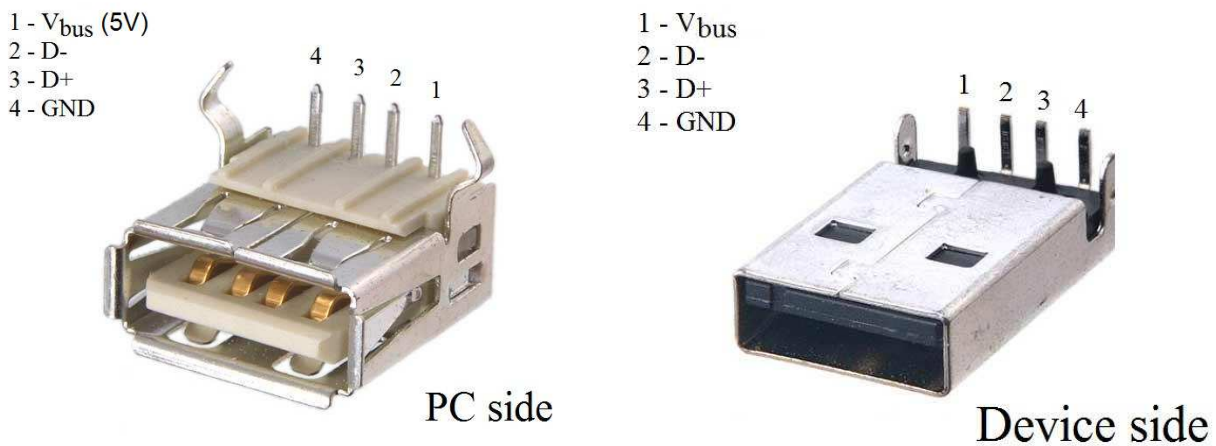


Figure 20: USB connectors and their pin configurations.

When you connect the pins in the wrong order (V_{bus} and GND) **you can burn your device or PC.**

IMPORTANT NOTE: There are some power limitations of USB hosts described by the usb 2.0 protocol. It supplies 5V (5.1V max) and can provide a maximum current of 500mA according to those specifications. Therefore, think twice if you want, for example, to drive a motor from USB which requires a large amount of current. You can cause a permanent damage to your computer if you exceed those specifications.

When constructing your circuit on the protoboard, be as neat as possible; use short cables, avoid jumps over components, use similar cables for each line (power, gnd) etc. The following is an example of a "neat" looking circuit which reduces the undesired problems caused by messy construction of a circuit.

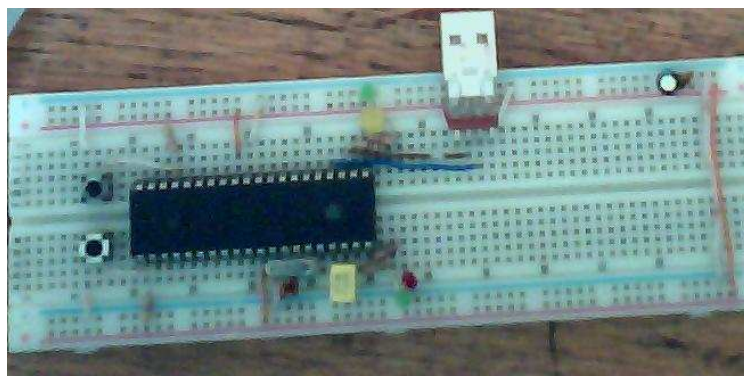


Figure 21: An example of a "neat" looking circuit

Programming the firmware via Bootloader

Although this section has been demonstrated in the tutorials on bootloaders, for the sake of completeness in the example, it is also provided here.

After you make sure that you have accurately implemented the circuit on board, plug it into one of the USB ports. Run the bootloader application "PDFUSB.exe" located on "C:\Microchip Solutions\USB Tools\Pdfsusb".

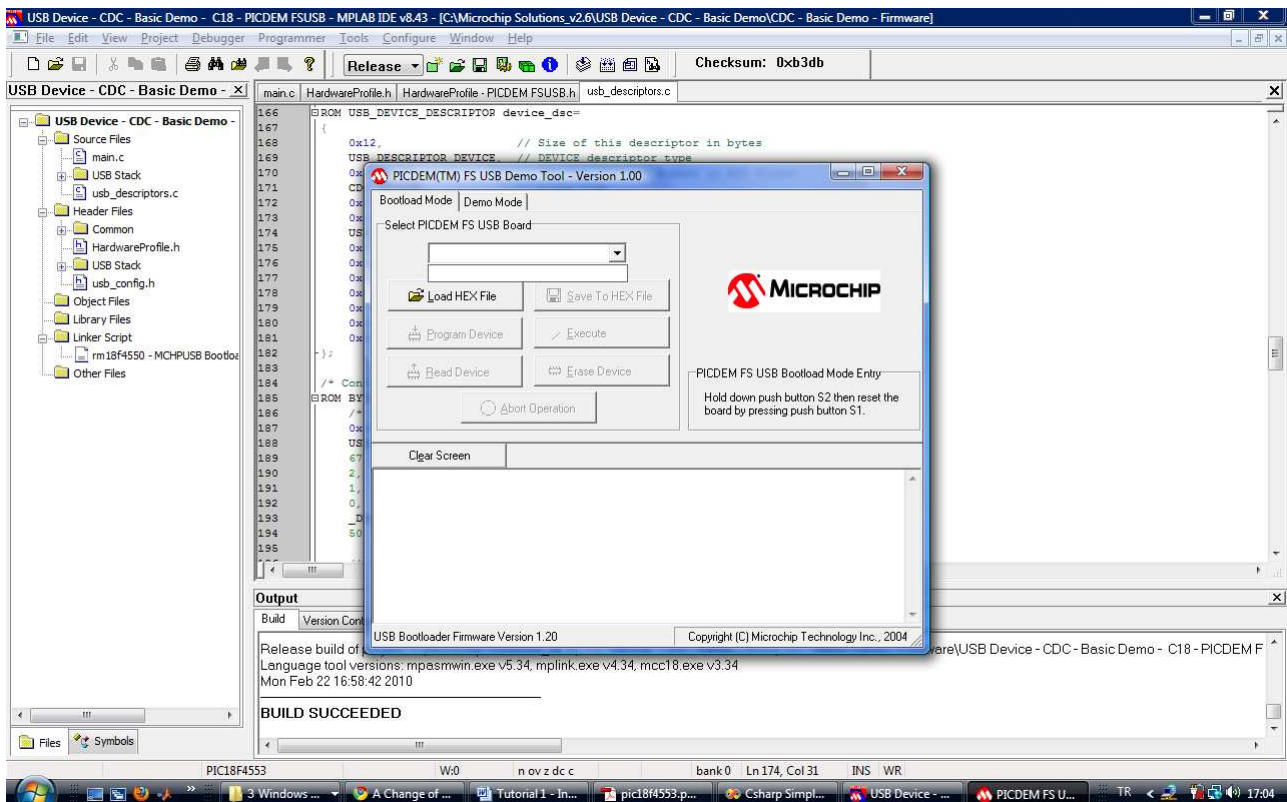


Figure 22: Programming the firmware via bootloader

When you click on the device list you will not see any device connected to your PC as a bootloading device. Now, press and hold the bootloading button (RB4) and while holding that press reset button. Release the reset button and after that release the bootloading button; you will see the bootloader firmware start to run on PIC device (you will notice the blinking on Led1 and Led2) and will be recognized by your PC. If the leds do not blink alternatively then there is a problem (probably you connected D+ and D- pins wrong).

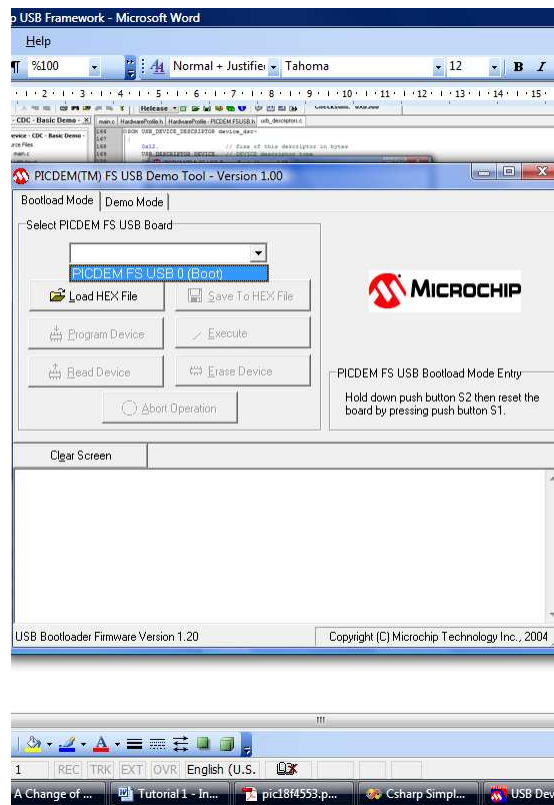


Figure 23: Running the bootloader

Select the boot device from the list and after that load the HEX file that you compiled (CDC Basic Demo).

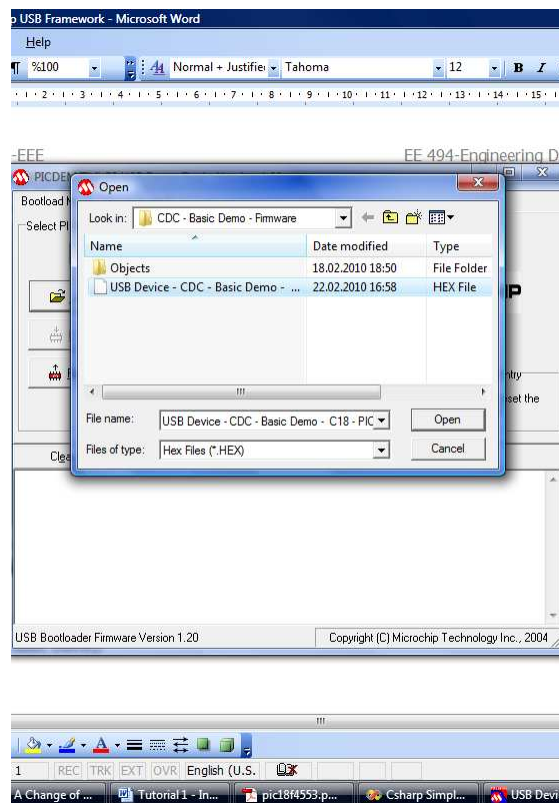


Figure 24: Loading the HEX file to be programmed

Select "No" to use the board's default configuration settings in the upcoming message. Press "Program Device" and wait until programming ends. If you see no error messages then you can press "Execute" (or the reset button on the device) to run the CDC firmware.

Introducing the Device to PC – Driver file installation

As your USB device is an abstract device for your computer you should define it to your PC. To do this you should use the .inf files supplied by the firmware. After you start your CDC firmware, your computer will ask the location of the driver file prompting "a new hardware found" message.

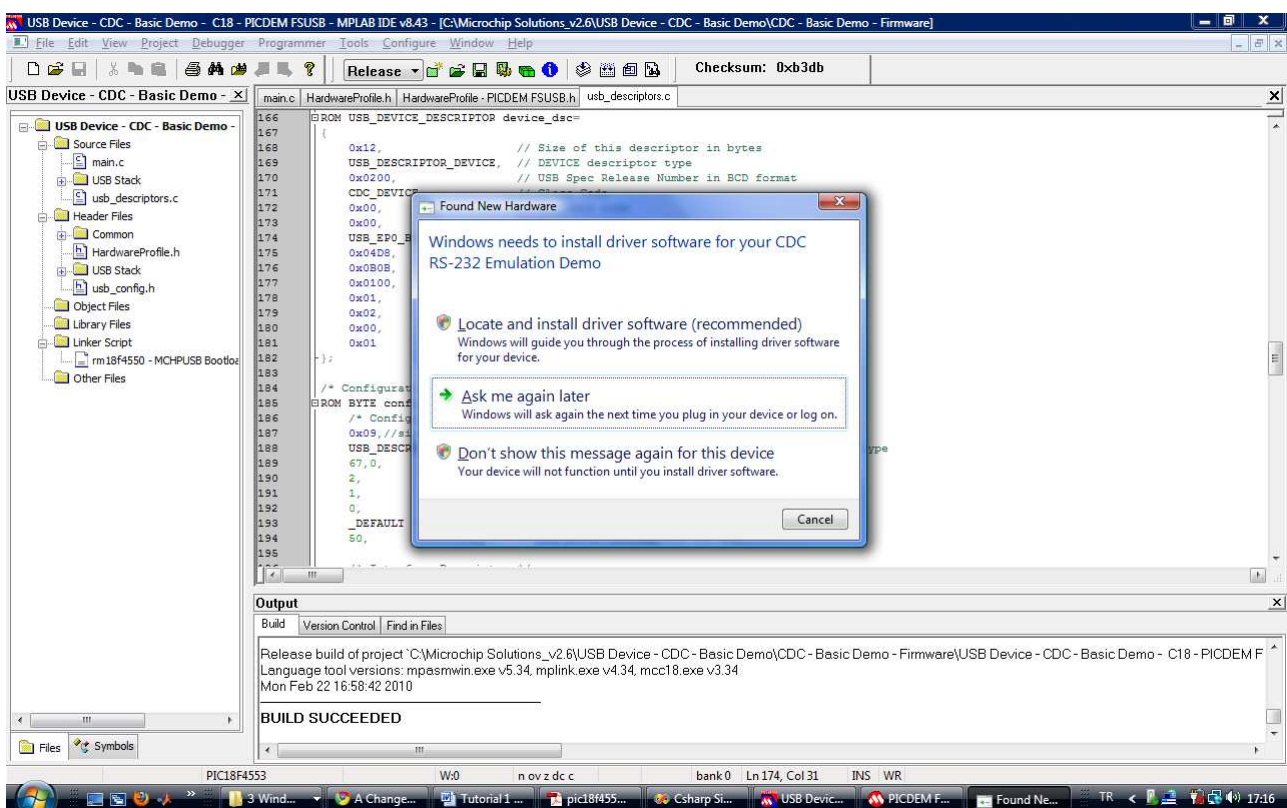


Figure 25: Driver file installation for CDC device

Select "Locate and install driver software (recommended)". First, your system will search for the driver file but eventually it can not find it. Then, you should locate the file by yourself.

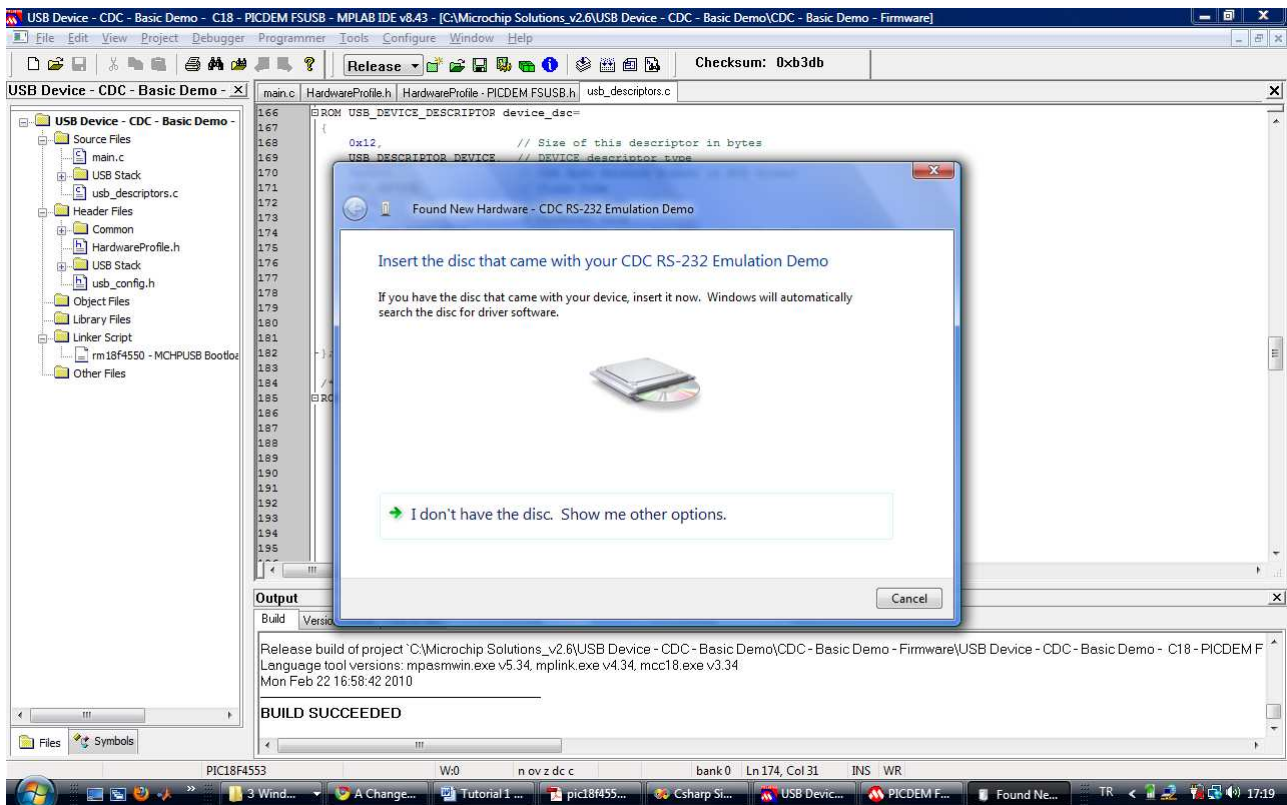


Figure 26: Installing the CDC driver

Select "I don't have the disc. Show me other options". Browse for file and select the folder inf in CDC Basic Demo directory.

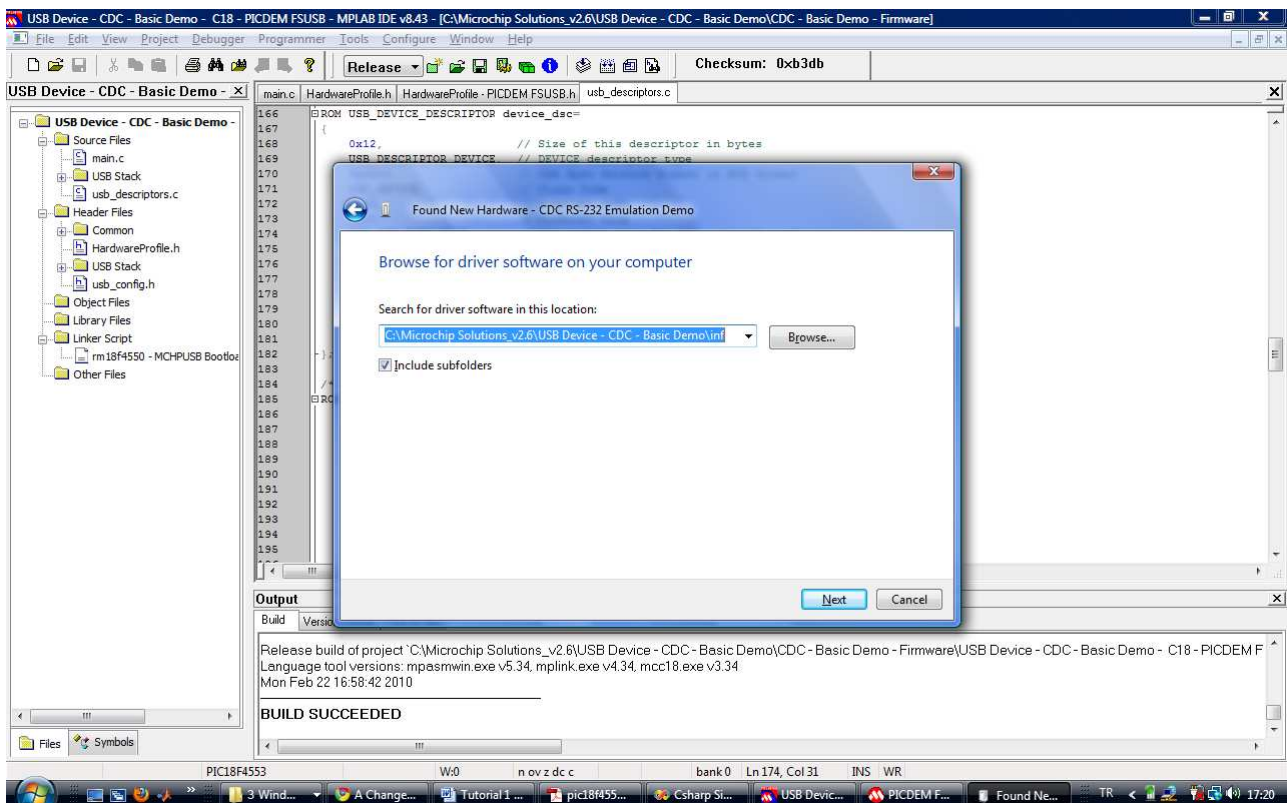


Figure 27: Settings for the driver installation



As you are the publisher of the device, it will not be verified by your system. Select "Install this driver software anyway".

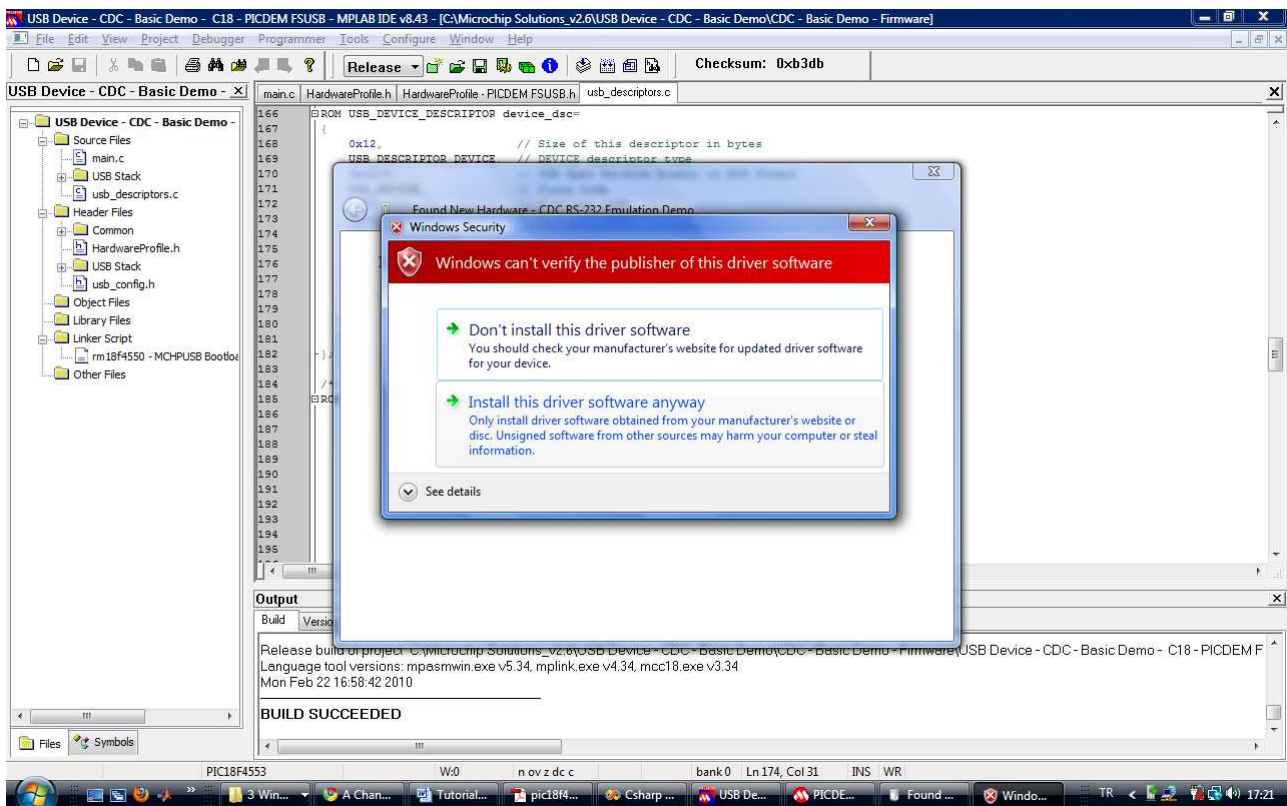


Figure 28: Final step in the CDC driver installation

After that, if you see an alternating blink on leds 1 and 2, then that means everything is done properly.

Explanation of the PC side application

As stated before, the Visual Studio .NET 2005 Professional Edition is chosen to implement the software running on the PC side. The programming language is preferred as C# which is very similar to Java in terms of syntax. You can easily adapt your skills to use this language if you have background in programming C++. In C#, you do not have to deal with pointers, freeing memory etc. which are some of the advantages of this language in programming.

Now, open the directory C:\Microchip Solutions\USB Device - CDC - Basic Demo\PC Software Example\VCsharp 2005\Basic Communication. Run the "Csharp Simple CDC Demo" solution file.

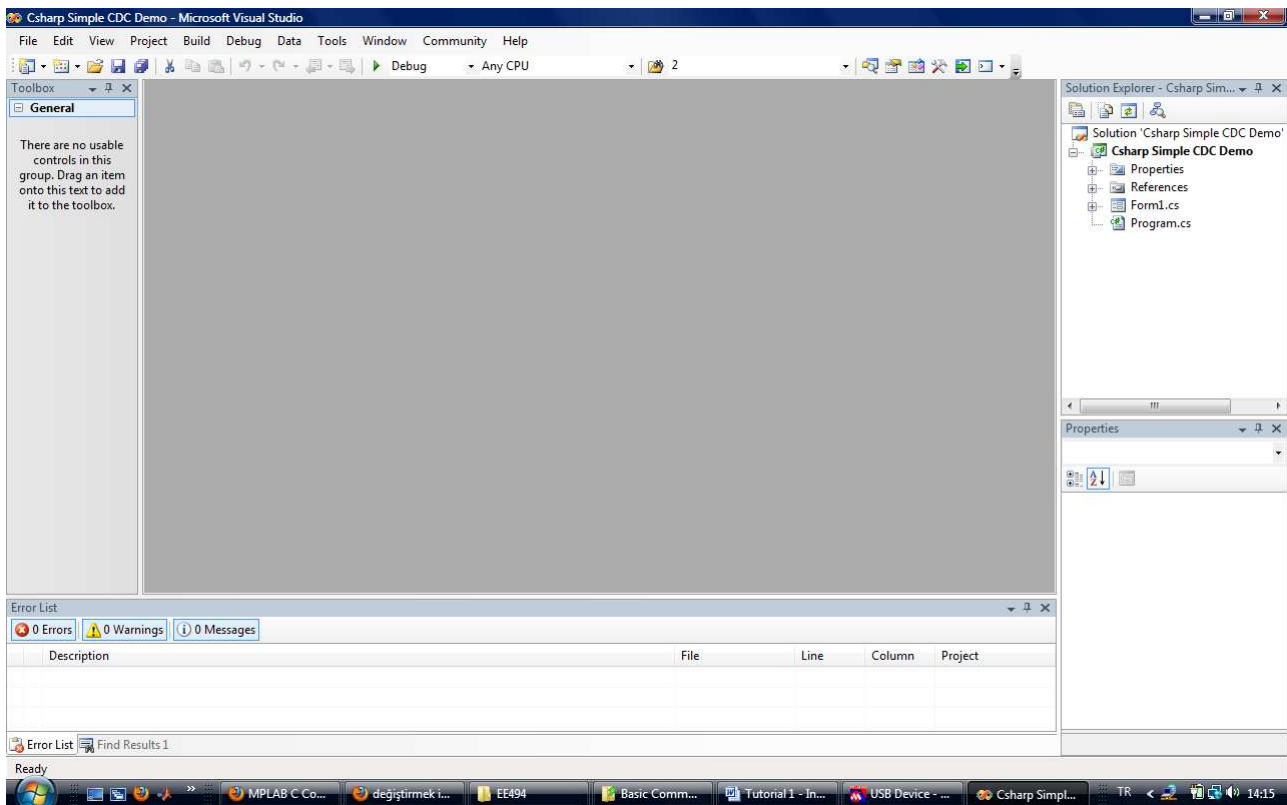


Figure 29: PC side application, VS .NET 2005 - C# user interface

Here you will see the programming environment of Visual Studio .NET 2005. On the right panel, you will see the Solution Explorer. Double click on Form1.cs to see the windows form created for this example.

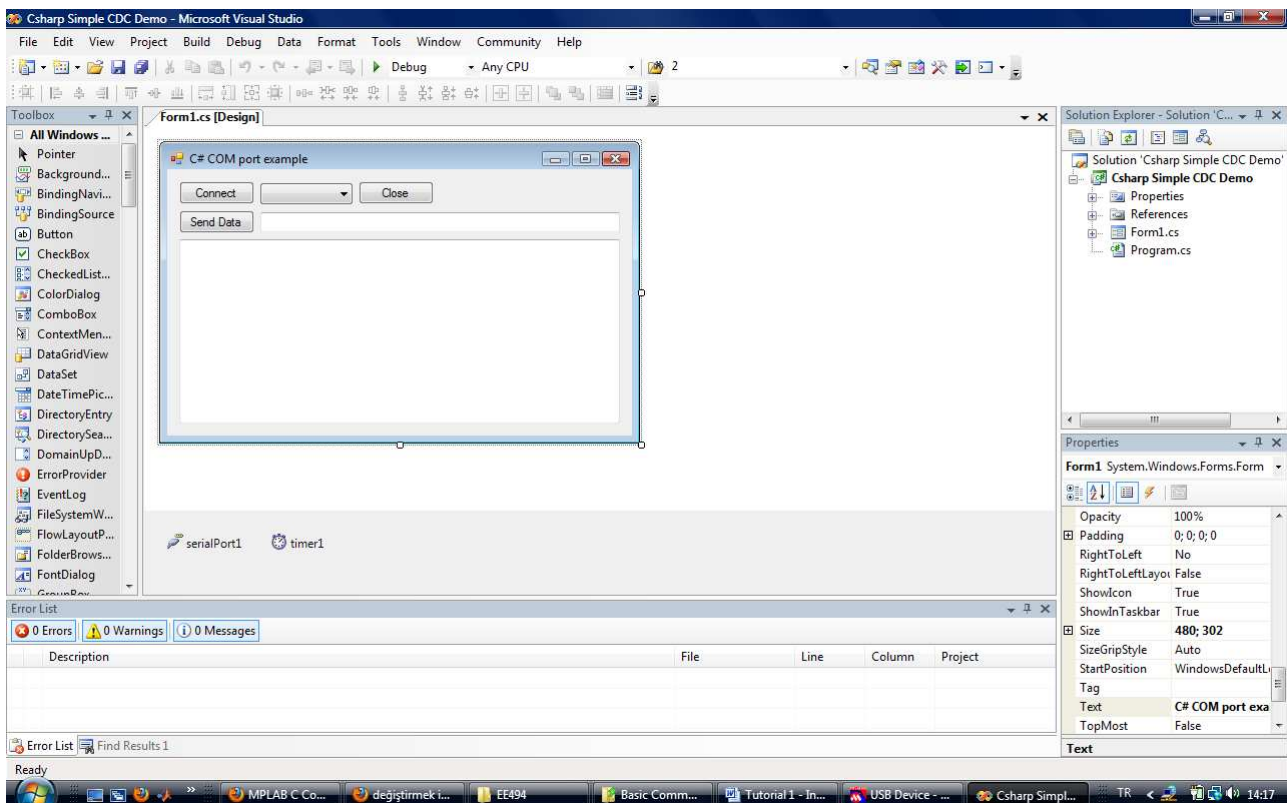


Figure 30: PC side application, Windows Forms application - designer view

You can see that there are some buttons and textboxes on the form which are used for connection to the PIC, sending and receiving the data. There is the serialPort object which interfaces the Com port utilities. Before you analyze the code, first press the "Debug" button to see the operation of the program (Actually you already know the operation as we traced the firmware code before).

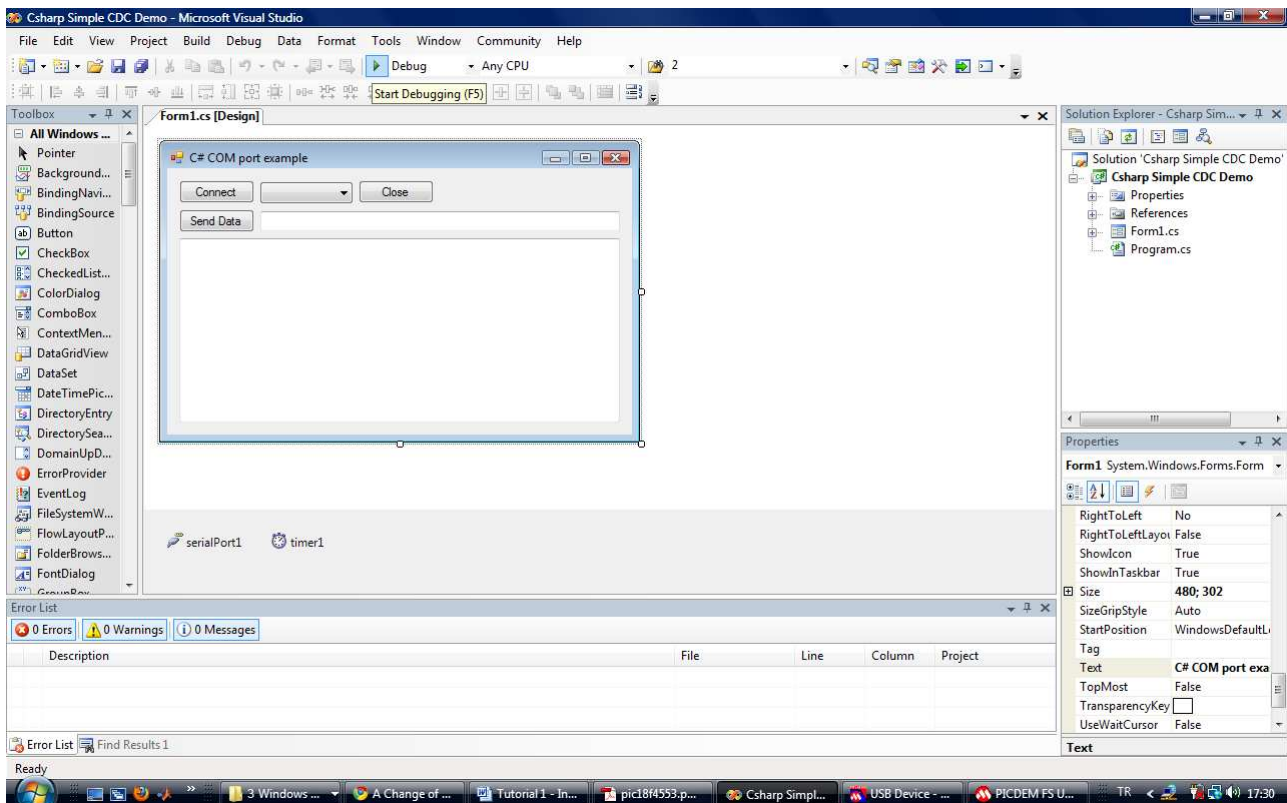


Figure 31: PC side application, running the application

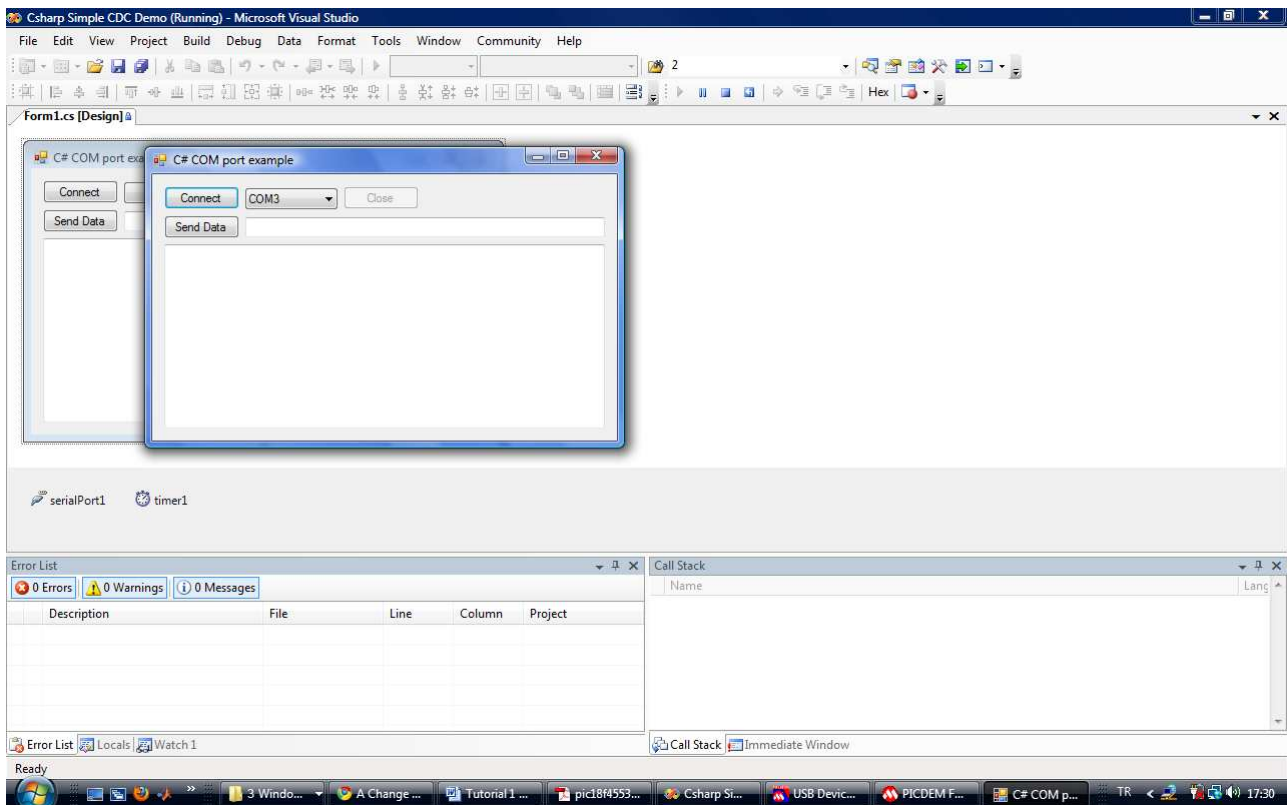


Figure 32: PC side application, selecting the COM port

You will see that the application runs and waits for the user to connect one of the COM ports. You will select the one created as a virtual COM port when you plug the device (the one at the end of the list).

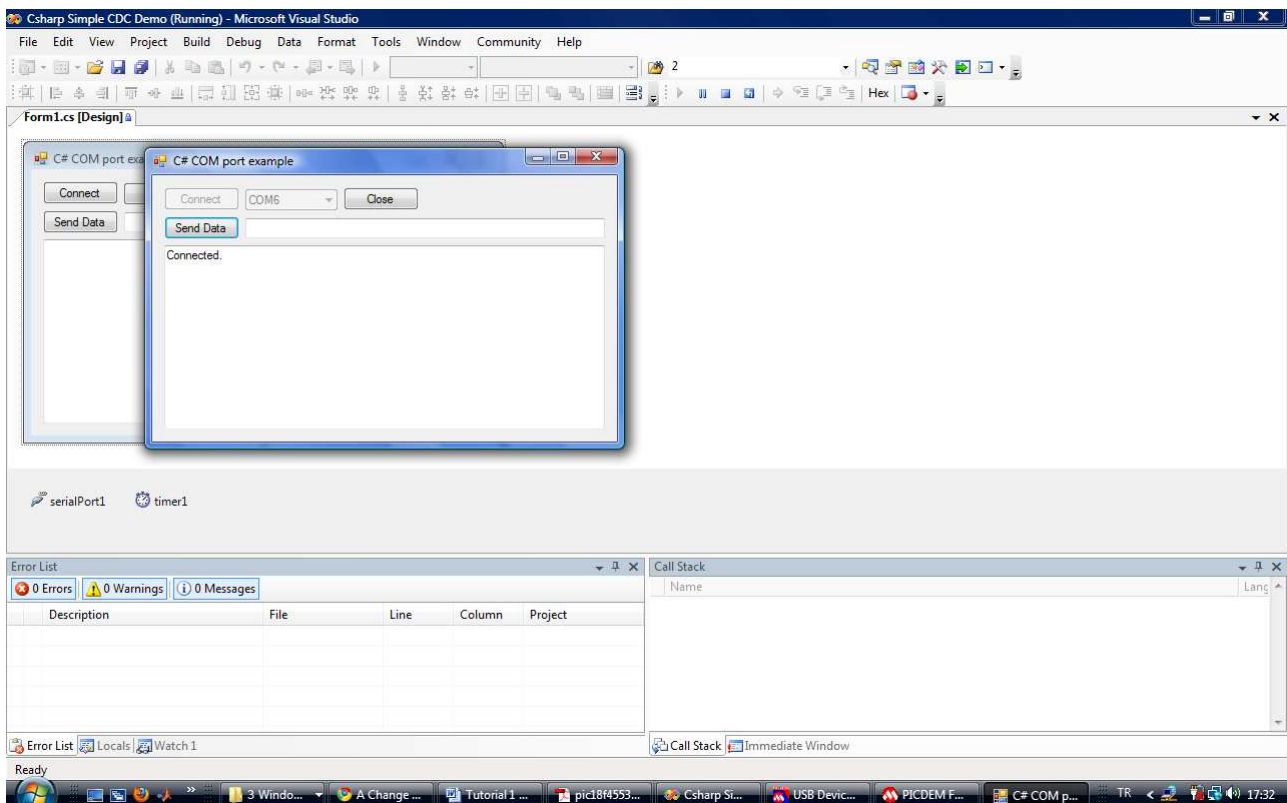


Figure 33: PC side application, connecting to the CDC device

Type 5 to the textbox next to the button "Send Data" and push "Send Data". You will see that the data is (char by char) incremented by 1 and sent back to PC

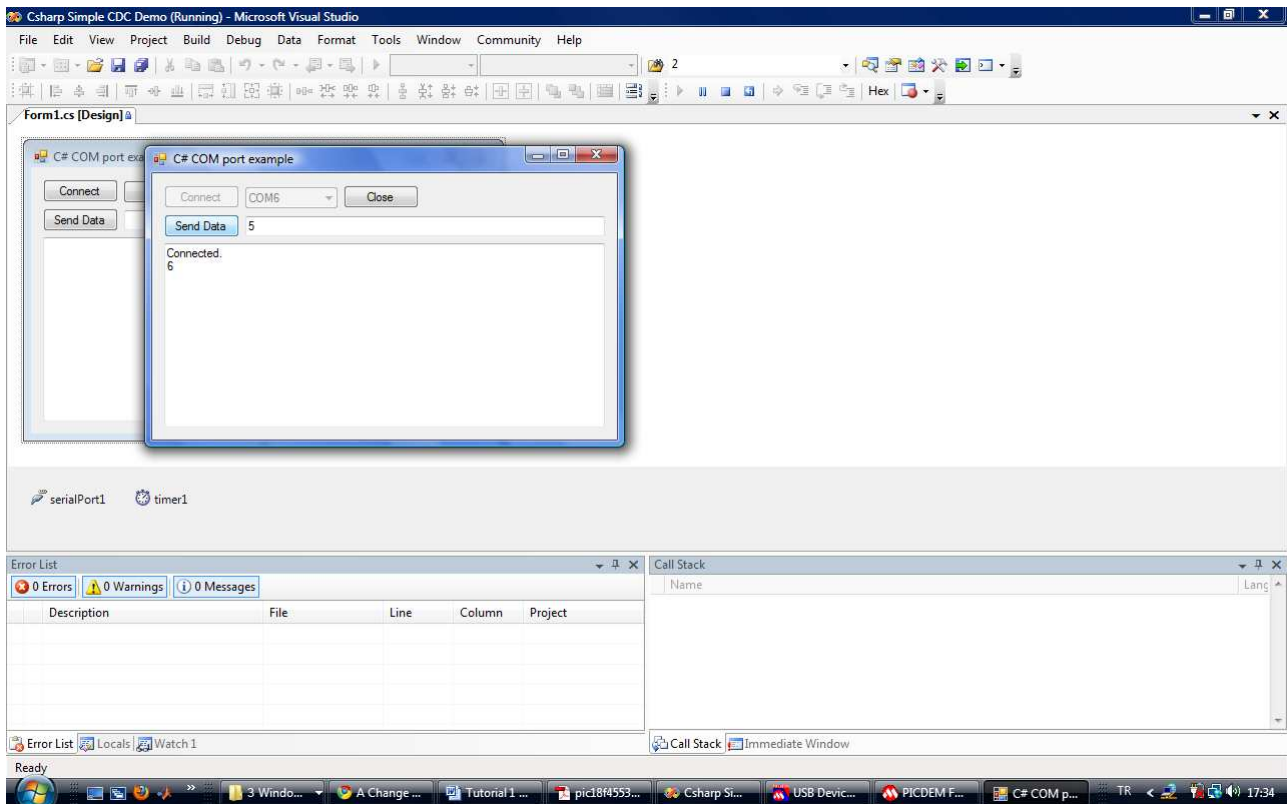


Figure 34: PC side application, sending and receiving data

Press the button RB4 and observe the screen.

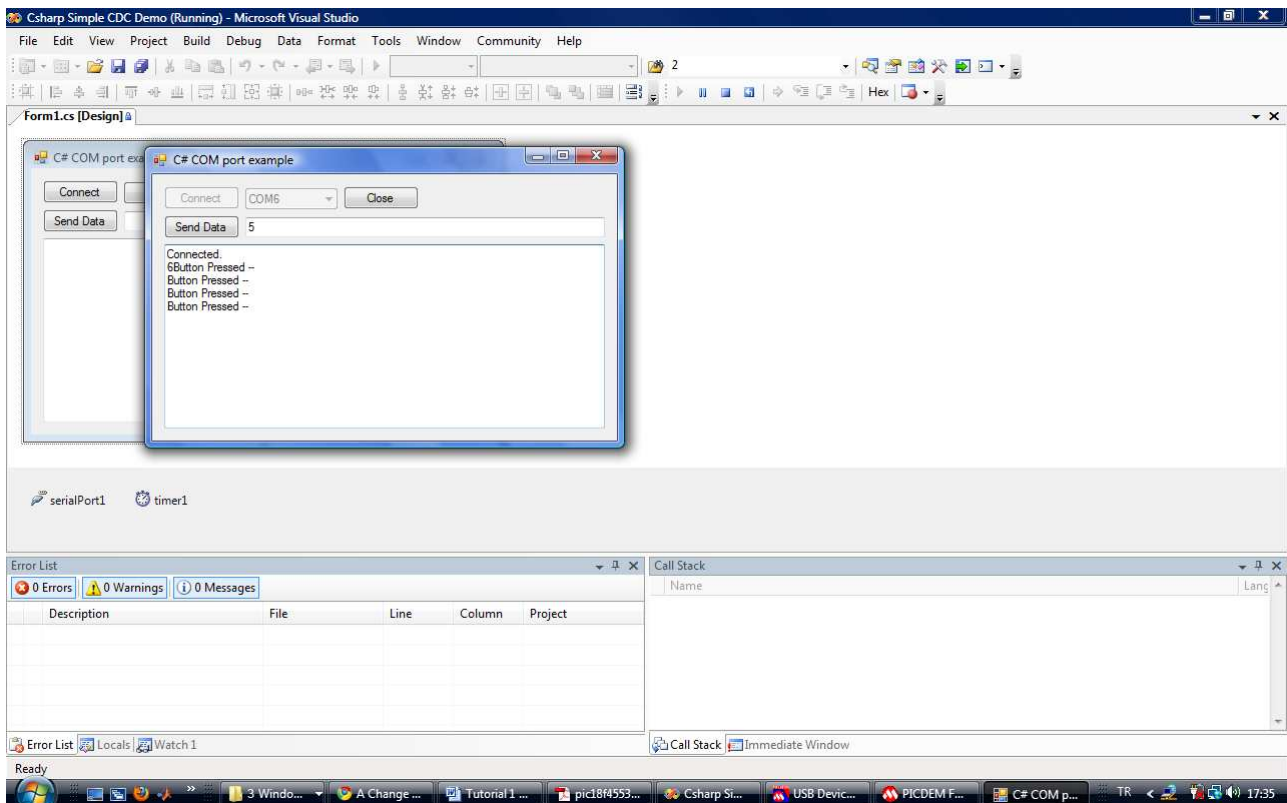


Figure 35: PC side application, pressing the bootloader button during runtime

You can easily understand the operation of the code by looking at the events and function declarations. It is a well commented code allowing the programmer to understand the basic operations used to send and receive functions used in serial USB communication.

You can also have a look at other examples in the framework (HID, Generic etc). All are prepared in the same manner with slight differences in send & receive functions. You can refer to the documentations of the framework for detailed information and explanation.

For any questions or problems related to the MCHP framework, you can easily find your answers just by googling through the internet and refer to forums, datasheets, documentations etc.

Some Useful Links

- <http://msdn.microsoft.com/en-us/vcsharp/default.aspx>
- <http://www.functionx.com/vcsharp/index.htm>
- MCHP Framework: <http://ww1.microchip.com/downloads/en/DeviceDoc/51679b.pdf>
- Forums on Microchip: <http://www.microchip.com/forums/tt.aspx?forumid=-1>
- ...

prepared by Balkar Erdoğan

