KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

CAMPUS GROUP T LEUVEN

Microcontrollers LAB 1

Basic I/O & Data filter

Coordinator: Prof. L. Geurts

Cooperator: T. Stas

.

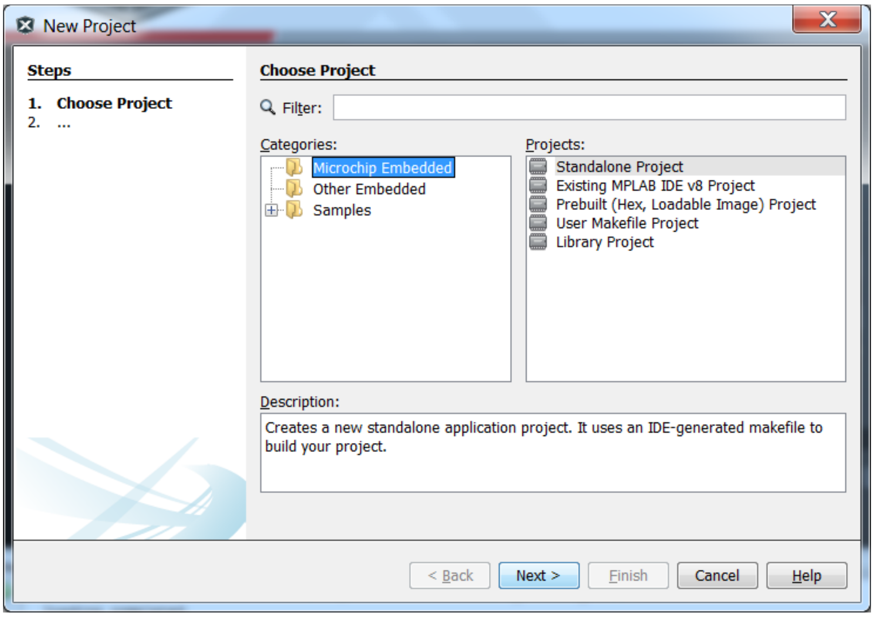
Academic Year 2019-2020

Software environment

The software MPlab X IDE is available on the Microchip website for free. Please install this software on your computer before the start of this lab.

Create a new project

Once you have the development environment opened, start a new project by going to File>> New Project.



New Project

Steps

1. Choose Project
2. ...

Choose Project

Filter:

Categories:
- Microchip Embedded
- Other Embedded
- Samples

Projects:
- Standalone Project
- Existing MPLAB IDE v8 Project
- Prebuilt (Hex, Loadable Image) Project
- User Makefile Project
- Library Project

Description:

Creates a new standalone application project. It uses an IDE-generated makefile to build your project.

Figure 1: Create new project in MPlab

If you create a new project, follow these steps:

1. Choose for a “Microchip Embedded – Standalone Project”, click on next.
2. Configure your project to work with the correct device: PIC18F25k50.
3. Set your hardware device to “Simulator”.
4. Select Compiler Toolchain. Set your active tool suite to ‘XC8’. This allows us to work with a C compiler. The compiler translates your C program into a format that your microcontroller can understand.
   Check that the Toolsuite Contents are presented into green dot.
5. Save your project in your personal folder. Check box to “Set as main project”. Click on Finish.

Select the PIC18F25k50 as your chip.

Programming in C

On Toledo you can find a template C file to start from in this first lab session. This template contains some variable and function declarations as well as initializations. Add the template file to your project:

1. Download the “main.c” file from Toledo.
2. Rightclick on the “Source file”-directory in your MPlab project.
3. Select “Add existing item” and browse for the “main.c” file. Check the box to COPY the file! The file will be copied to your project directory.

Look through the template, read the comments and try to understand its general layout. For this lab, you should concern yourself with editing the content of functions initChip() and main(). The example code shows a simple 8 bit up/down counter controlled by RA0.

RA0 = 1 : UP counter
RA0 = 0 : DOWN counter

Syntax to assign values to register

You can easily assign values to registers in the microcontroller using hexadecimal or binary formats.

```
TRISA = 0xFF;     //Give all bits on register TRISA the value of ‘1’
                  //(hex representation)
```

Value ‘1’ : this port bit acts as input.
Value ‘0’ : this port bit acts as output.

```
LATC = 0b11000000;    // Give pins 7 and 6 on Port C the value of ‘1’
                      //(binary representation)
```

You can also work with one particular pin on a register by adding ‘bits.’ after the register name followed by the name of the pin.

```
LATCbits.LC1  = 1; //Assign a logical ‘1’ to pin 1 on port C
LATCbits.LC2  = PORTAbits.RA0 //Assign the value of Port A, pin 0 to Lat C, pin 2.
```

You should consult the data sheet to find out the pin and register names.

Data types

Type	Size (bits)	Value range
(signed) char	8	-128 to 127
unsigned char	8	0 to 255
int	16	-2^{15} to 2^{15}-1
unsigned int	16	0 to 2^{16}

Building a program

Once you are satisfied with your program, build it by going to Run>>Build main project or use the button 🔨. If your program contains errors, you will receive an error message in the bottom window.

If your program builds successfully, and there are two steps can be done next. You can simulate your program and you can download your program onto your microcontroller.

Simulating a program

Once you have a finished and compiled project in MPLABX, you can start to debug the program by clicking the button . Then this debugging tool bar will be showed on the window.

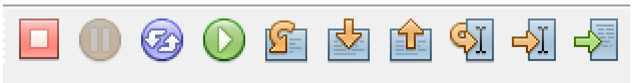


Figure 2: Debugging toolbar

There are three ways to run the program. Each can be selected under the Debugger pull-down menu or by selecting the buttons on the toolbar. These are:

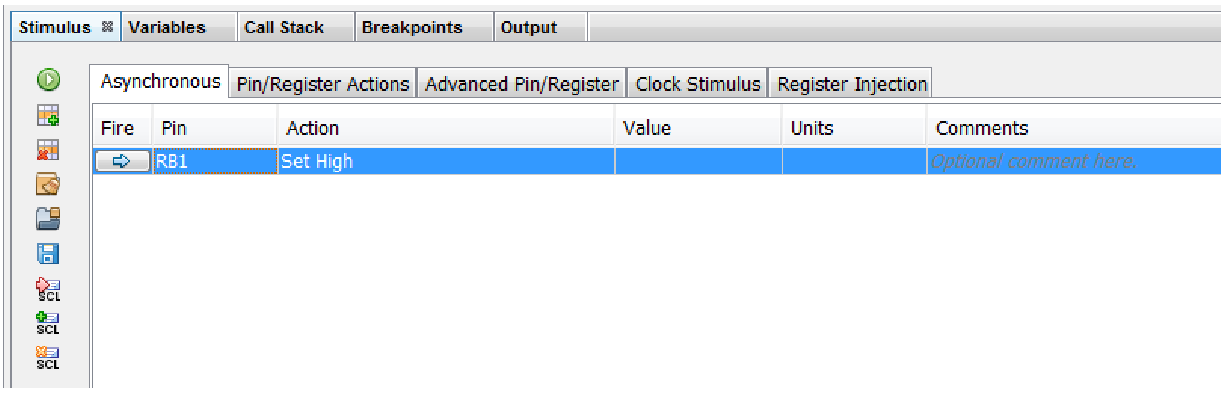
- Single Step. This allows you to step through the program one instruction at a time. MPLABX uses the terminology 'Step Into' for this mode.
- Reset. Reset the whole program and start over again.
- Run. This runs the program, but does not update on-screen windows as it runs. It does, however, accept stimulus input.
- Break points. You can also insert breakpoints by double clicking on the grey area next to a line in the program. Do not insert break point on non-executable lines (ie. Comment lines, parentheses, blanks).

It is also possible to Step Over a subroutine or Step Out of one. Each of these also has a button on the toolbar. These are especially useful for delay routines, which on a simulator may take an unacceptably long time to simulate.

Generating port inputs

To complete simulate the project, it is necessary to have virtual input ports and output ports. Select Window → Simulator → Stimulus.

Explore the dialogue box that appears at the bottom—it allows you to set up different types of inputs at the Port pins, which are initiated by pressing the "Fire" button ⇨ at the appropriate moment.



Stimulus | Variables | Call Stack | Breakpoints | Output

Asynchronous | Pin/Register Actions | Advanced Pin/Register | Clock Stimulus | Register Injection

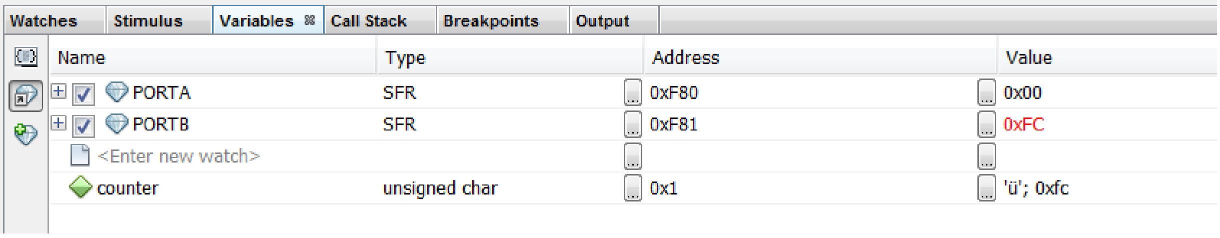
Fire	Pin	Action	Value	Units	Comments
⇨	RB1	Set High			Optional comment here.

Figure 3: Stimulus window

Each pin has five operations: Set High, Set Low, Toggle, Pulse low and Pulse High. Normally, we only use first three operations.

View microcontroller registers and variables

You can observe a number of microcontroller features during simulation, including program memory, SFRs, data memory and so on. A window can be opened for each of these by selecting Window → Debugging → Variables.



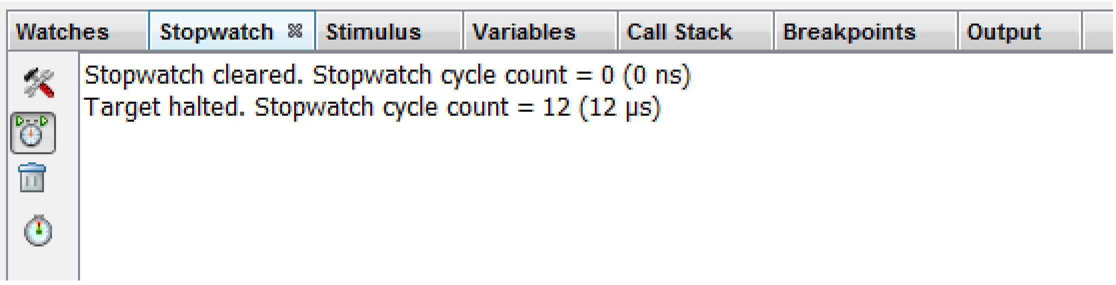
Name	Type	Address	Value
☑ PORTA	SFR	0xF80	0x00
☑ PORTB	SFR	0xF81	0xFC
<Enter new watch>			
counter	unsigned char	0x1	'ü'; 0xfc

Figure 4: Variables window

You can add a new "watch" to the registers you would like to have a look at by selecting , and search in the SFR's. A variable or register that receives a new value receives a red colour for its value. The value representation can be chosen decimal, hexadecimal, binary by right clicking → "Display value column as".

Measure time

It is possible to check your timings already at simulation level. But first our simulator needs to know at which clock speed our microcontroller is working. Therefore in the Project Properties you can find the options for the Simulator. **Change the Instruction Frequency (Fcyc) to 48 MHz**. Now it is possible to use the Stopwatch. Go to Window → Debugging → Stopwatch.



```
Stopwatch cleared. Stopwatch cycle count = 0 (0 ns)
Target halted. Stopwatch cycle count = 12 (12 µs)
```

Figure 5: Stopwatch window

Reset the stopwatch to start the time at 0 seconds. Make sure you have a breakpoint at the end of the piece of code from which you would like to measure the timing. When you now RUN your program, the stopwatch will indicate time between your start line, and the breakpoint. You can check the timing of the whole main loop, but also partial routines.

Assignment:

Try to simulate the template program. Can you see the local variable "COUNTER" changing? Do you also see the PORTB register changing? Can you switch between an UP or a DOWN counter by stimulating pin RA0?

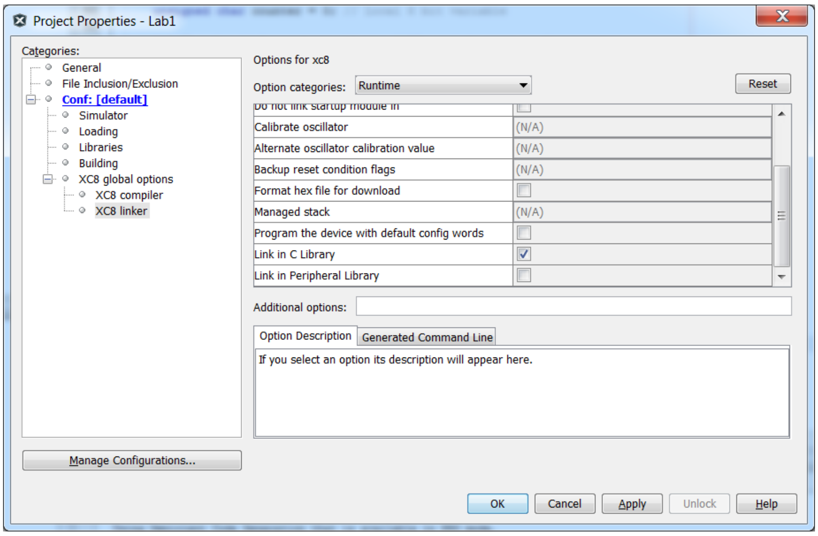
Downloading a program

Configuration for the bootloader

Don't forget to make these property changes every time you start a new project!

The PIC's memory contains a bootloader as well as the program to be executed. The bootloader resides at address 0x000 – 0xFFF. The executable program begins at address 0x1000 (reset vector) while high and low priority interrupt vectors are situated at addresses 0x1008 and 0x1018 respectively. When the program starts, the program counter points to the reset vector and executes the program line-by-line. When an interrupt occurs, the counter jumps to an interrupt vector address, which then points it to the interrupt routine. To be able to program the device with well programmed bootloader, the following configuration should be done for XC8 v2.00 compiler:

1. Right click on the project and go to the project properties.
   XC8 linker: Option categories =>Runtime
   Uncheck "Program the device with default config words"



Options for xc8

Option categories: Runtime

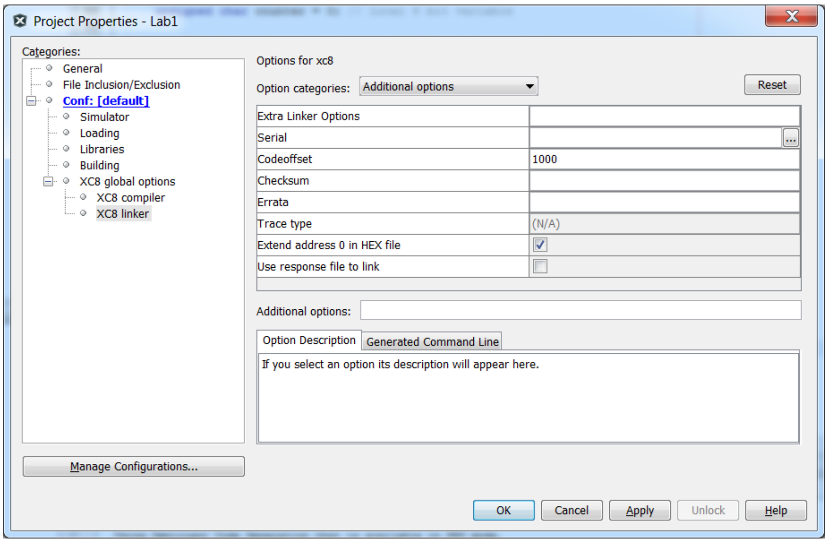
Do not link startup module in	□
Calibrate oscillator	(N/A)
Alternate oscillator calibration value	(N/A)
Backup reset condition flags	(N/A)
Format hex file for download	□
Managed stack	(N/A)
Program the device with default config words	□
Link in C Library	☑
Link in Peripheral Library	□

Additional options:

Option Description | Generated Command Line

If you select an option its description will appear here.

2. XC8 linker: Option categories=> Additional options
   Codeoffset": 1000
   Uncheck "Extend address 0 in HEX file" (check for older versions)



Options for xc8

Option categories: Additional options

Extra Linker Options	
Serial	
Codeoffset	1000
Checksum	
Errata	
Trace type	(N/A)
Extend address 0 in HEX file	☑
Use response file to link	□

Additional options:

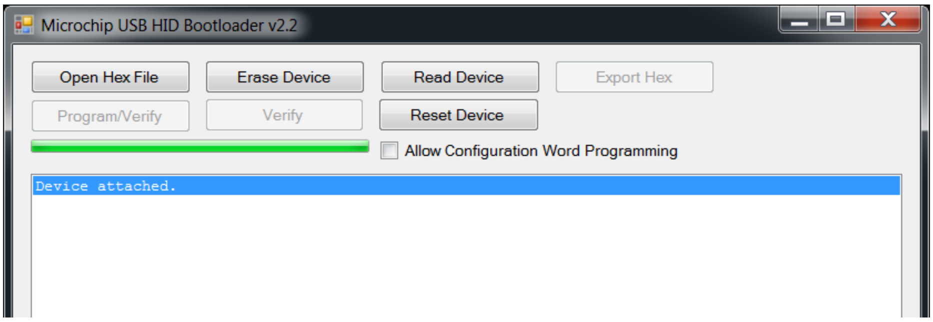
Option Description | Generated Command Line

If you select an option its description will appear here.

3. Click "apply" to save the settings

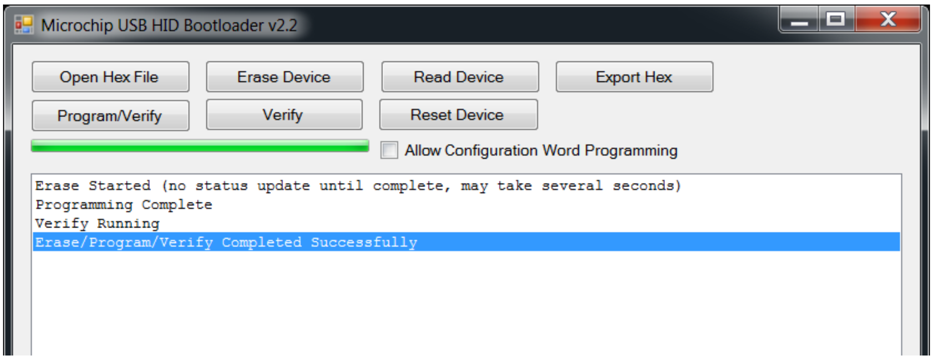
Download

4. Open HIDBootLoader.exe.
5. Plug the board into the computer using a USB cable. Put the PIC in bootloader mode by pushing both S1 and S2 buttons, then releasing the S1 button while the S2 button is still pressed. Your computer should recognize a new device, and LED 2 should start blinking.
6. If the microcontroller is in bootloader mode, Microchip USB HID Bootloader will give notice “Device attached”.



```
Device attached.
```

1. Select “Open Hex File” and browse for the .hex file the compiler has generated. This will be found in your project folder\dist\default\production.
2. Now press the Program/Verify button. Once your microcontroller has been programmed, you should see following message:



```
Erase Started (no status update until complete, may take several seconds)
Programming Complete
Verify Running
Erase/Program/Verify Completed Successfully
```

3. Then, push the S1 button to see your program in action in real life.

Assignment:

Try to download the template program. Connect 8 LEDs to PORTB and use RA0 as input control. You will notice the program is being executed at the processor speed, therefore the counter is counting faster than the human eye can see. Try to slow down the counter by using a DELAY function.

Assignment “Data filter on PIC”:

Notice the array declaration at the start of the template code.

This array of data has 512 samples (unsigned 8 bit values). This array represents a triangle wave starting at 0, increasing to 255 and then decreasing back to 0. However, quite some noise is included on the input data.

The target of the lab session is to filter the input data. The filter to be used is the moving average filter, length 8 values. This means that the filter will take a new value and the previous 7 values and will calculate the average. Each new input sample will result in a new filtered output sample.

At start-up the previous values are all zero. Think about efficiency when you implement the filter!

Thermometer readout

The so-called thermometer readout is a "linear representation" of values, as opposed to the binary representation. An 8-bit value can be represented by 8 output LEDs : LED0,LED1 ....LED7 as follow :

Input < 16 : all LED's are off

Input >= 16 : LED0 is on , all others are off

Input >= 48 (16+32) : LED0, LED1 is on, all others are off

Input >= 80 (16+64) : LED0, LED1, LED2 are on, all others are off

. . . : . . .

Input >= 240(16+224) : all LEDs are on

Write the C program to implement the moving averaging filter. PortB should be used to connect the 8 LED's. A reset input switch S1 is required. Also another switch S2 is required to swap the LED display: either the raw (=unfiltered) data or the filtered data should be visible.

A software delay should be implemented to allow for a readout of the LED's at "human speed".

The LED display will suffer from jitter when displaying the raw data. The filtered data should show a smooth display.

References

MPLAB XC8 C Comiler User’s Guide for PIC MCU
(http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_XC8_C_Compiler_User_Guide_for_PIC.pdf)