An introduction to

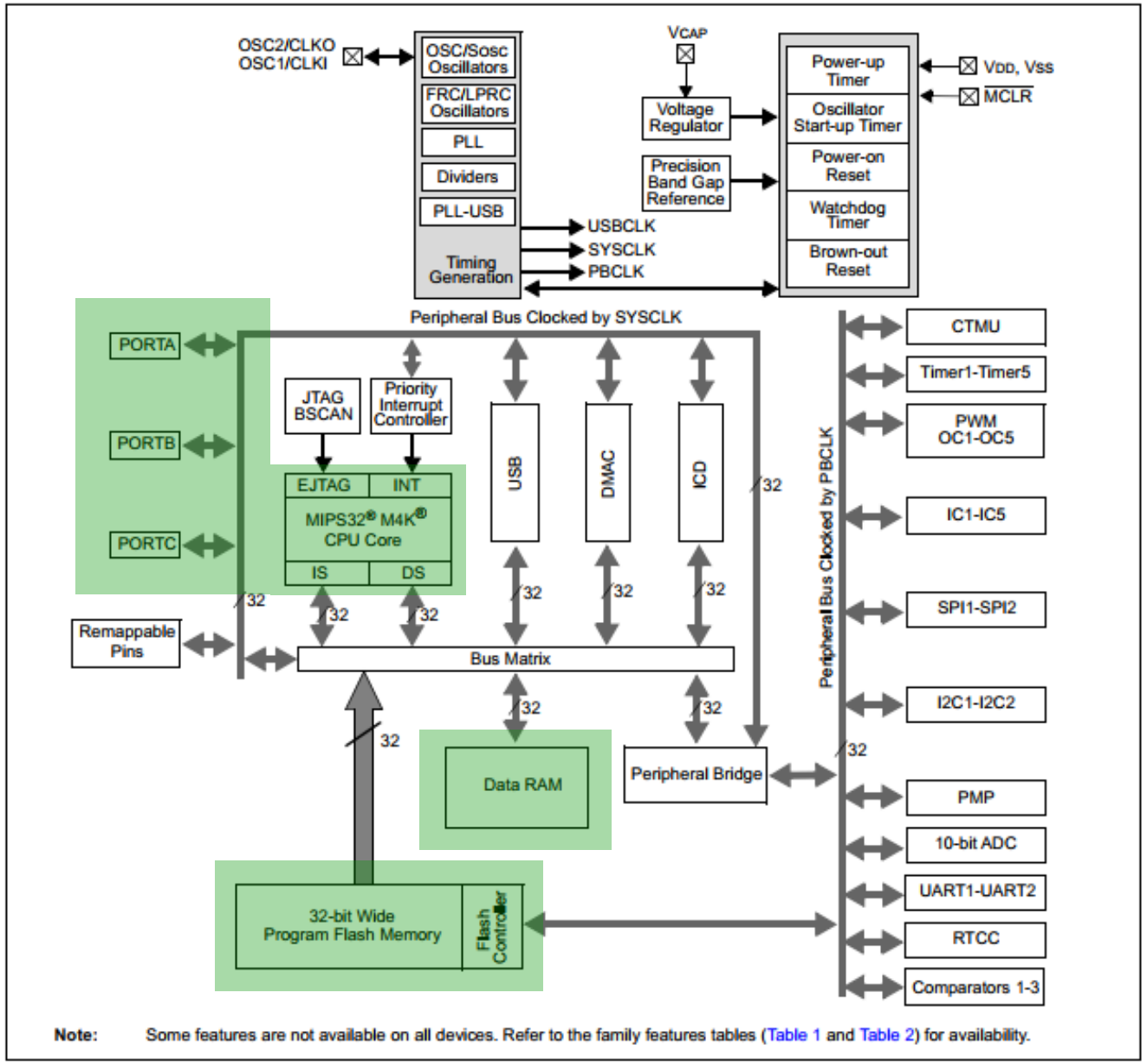
PIC32 MX110F016B Microcontroller

Created by Mojtaba Pour Ali Mohammadi and Sina Akbari

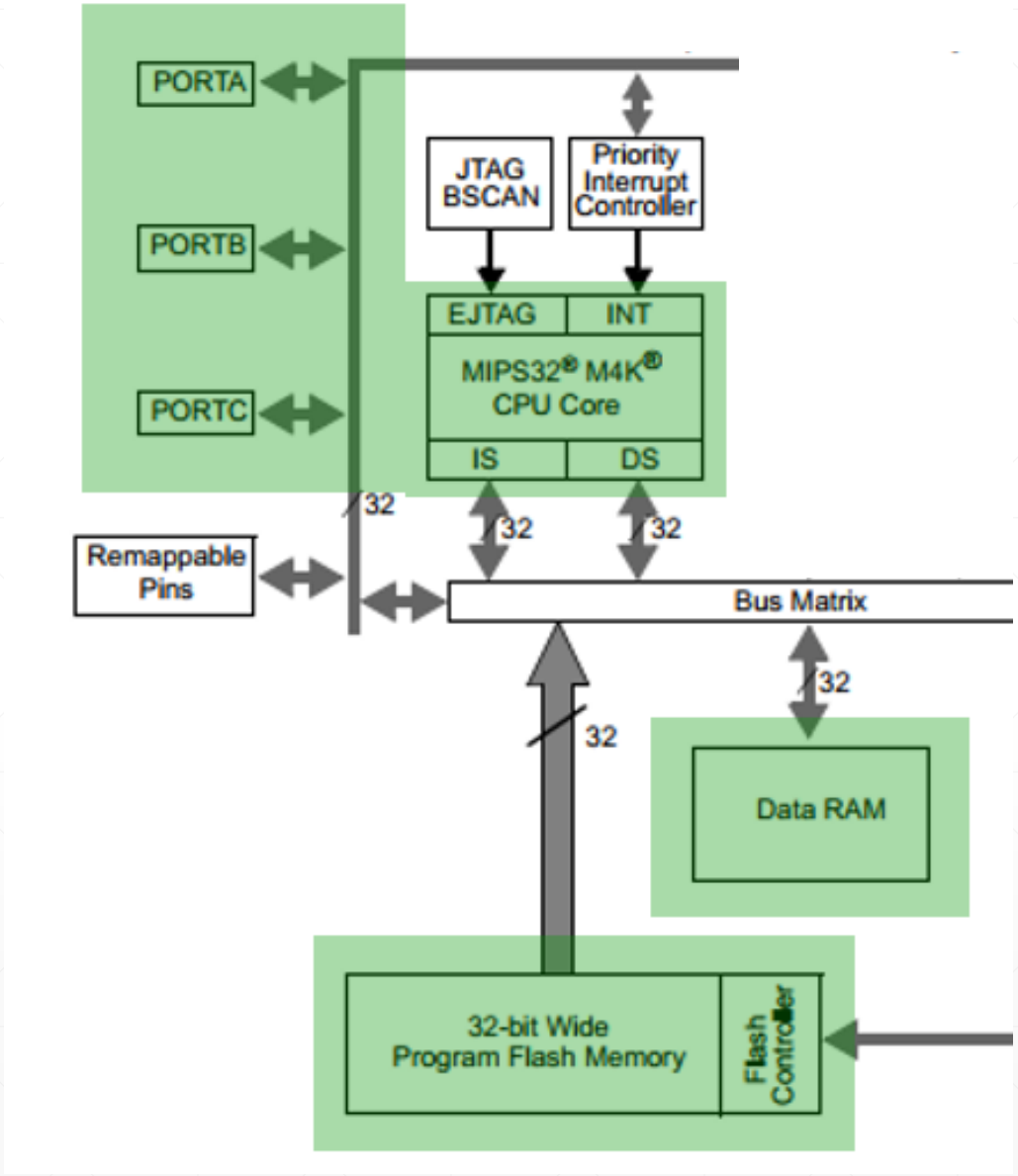
Device Overview

- 28-PIN SPDIP
- 4GB Flexible Virtual Memory Space
- MIPS32® M4K® Processor Core
- Complies with MIPS32® Instruction Set

FIGURE 1-1: BLOCK DIAGRAM



Note: Some features are not available on all devices. Refer to the family features tables (Table 1 and Table 2) for availability.


PORTA
PORTB
PORTC
JTAG BSCAN
Priority Interrupt Controller
EJTAG
INT
MIPS32® M4K®
CPU Core
IS
DS
32
32
32
Remappable Pins
Bus Matrix
32
32
Data RAM
32-bit Wide
Program Flash Memory
Flash Controller

Device Pinout

Pin #	Full Pin Name
1	$\overline{\text{MCLR}}$
2	VREF+/CVREF+/AN0/C3INC/RPA0/CTED1/RA0
3	VREF-/CVREF-/AN1/RPA1/CTED2/RA1
4	PGED1/AN2/C1IND/C2INB/C3IND/RPB0/RB0
5	PGEC1/AN3/C1INC/C2INA/RPB1/CTED12/RB1
6	AN4/C1INB/C2IND/RPB2/SDA2/CTED13/RB2
7	AN5/C1INA/C2INC/RTCC/RPB3/SCL2/RB3
8	VSS
9	OSC1/CLKI/RPA2/RA2
10	OSC2/CLKO/RPA3/PMA0/RA3
11	SOSCI/RPB4/RB4
12	SOSCO/RPA4/T1CK/CTED9/PMA1/RA4
13	VDD
14	PGED3/RPB5/PMD7/RB5

Pin #	Full Pin Name
15	PGEC3/RPB6/PMD6/RB6
16	TDI/RPB7/CTED3/PMD5/INT0/RB7
17	TCK/RPB8/SCL1/CTED10/PMD4/RB8
18	TDO/RPB9/SDA1/CTED4/PMD3/RB9
19	VSS
20	VCAP
21	PGED2/RPB10/CTED11/PMD2/RB10
22	PGEC2/TMS/RPB11/PMD1/RB11
23	AN12/PMD0/RB12
24	AN11/RPB13/CTPLS/PMRD/RB13
25	CVREFOUT/AN10/C3INB/RPB14/SCK1/CTED5/PMWR/RB14
26	AN9/C3INA/RPB15/SCK2/CTED6/PMCS1/RB15
27	AVSS
28	AVDD

Referred to the datasheet

An introduction to the boards

Microcontroller board

This PCB board is created for easy use of the user:

1. The microcontroller itself which is utilized on a board for easier use.

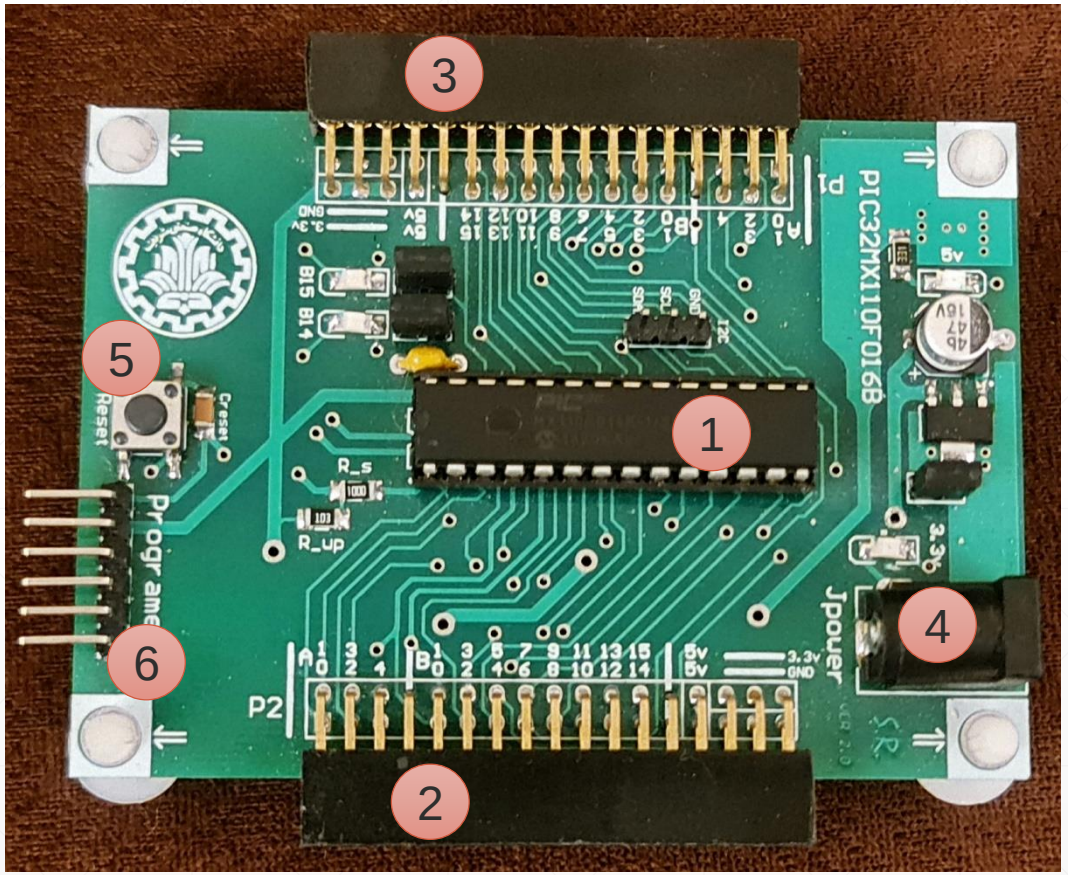
2. A and B pins and VCC and the ground are wired from the microcontroller and the power supply to this part to be connected to an led or a button.

3. Exactly the same as 2.

4. Should be plugged into the power outlet via the provided adapter.

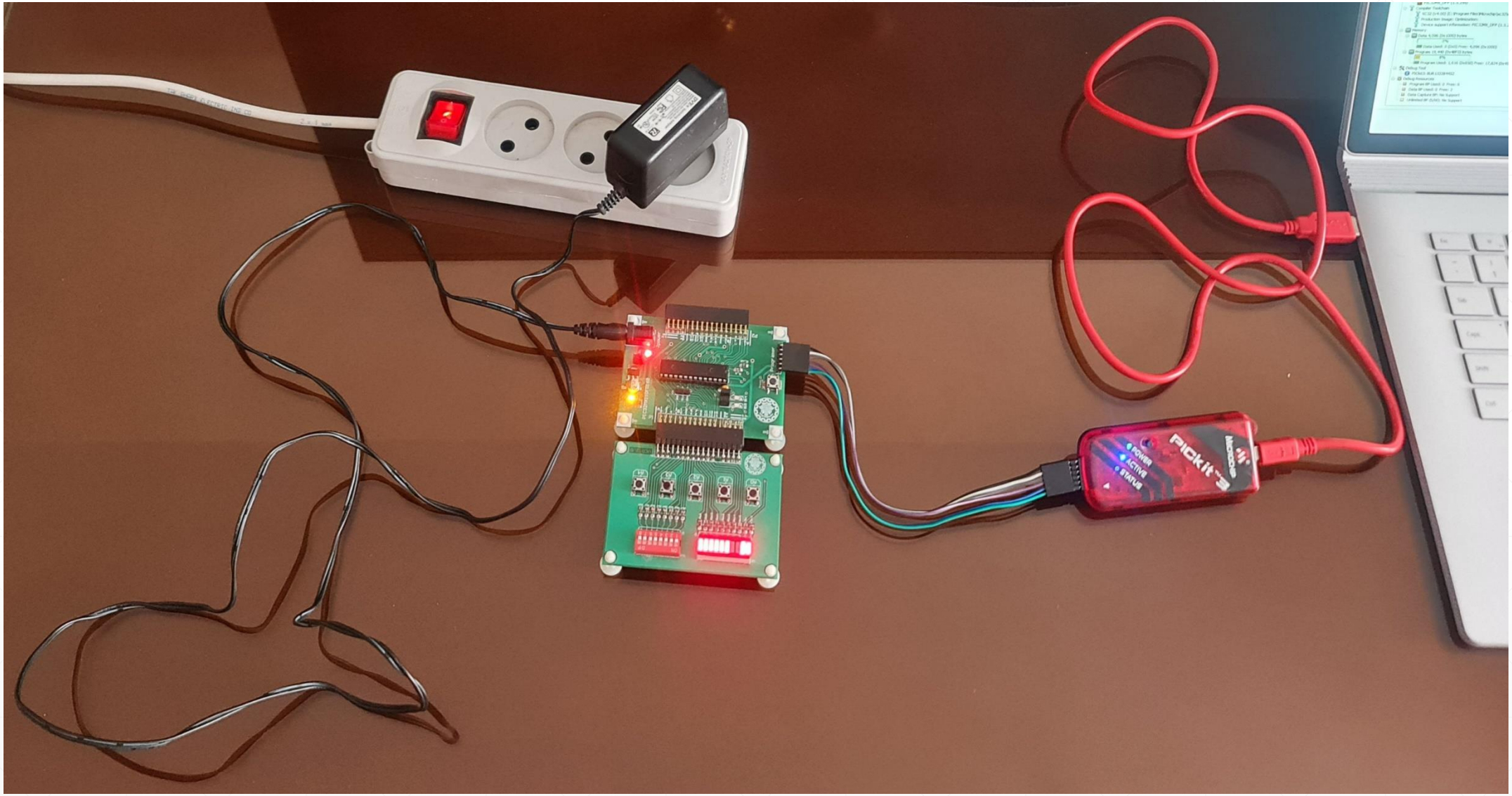
5. The reset button resets the microcontroller.

6. As it is necessary to program the microcontroller (that is saving the favorable program in the program memory), a laptop should be attached to it by connecting the programmer to this section.



Programmer

- An assembly or C code is created on a laptop, while the code must be in the program memory of the microcontroller. To do so a tool named PICkit3 is used, which is a programmer that connects the laptop to the microcontroller board via the cables provided and programs the microcontroller.


POWER
ACTIVE
STATUS
Pickit 3

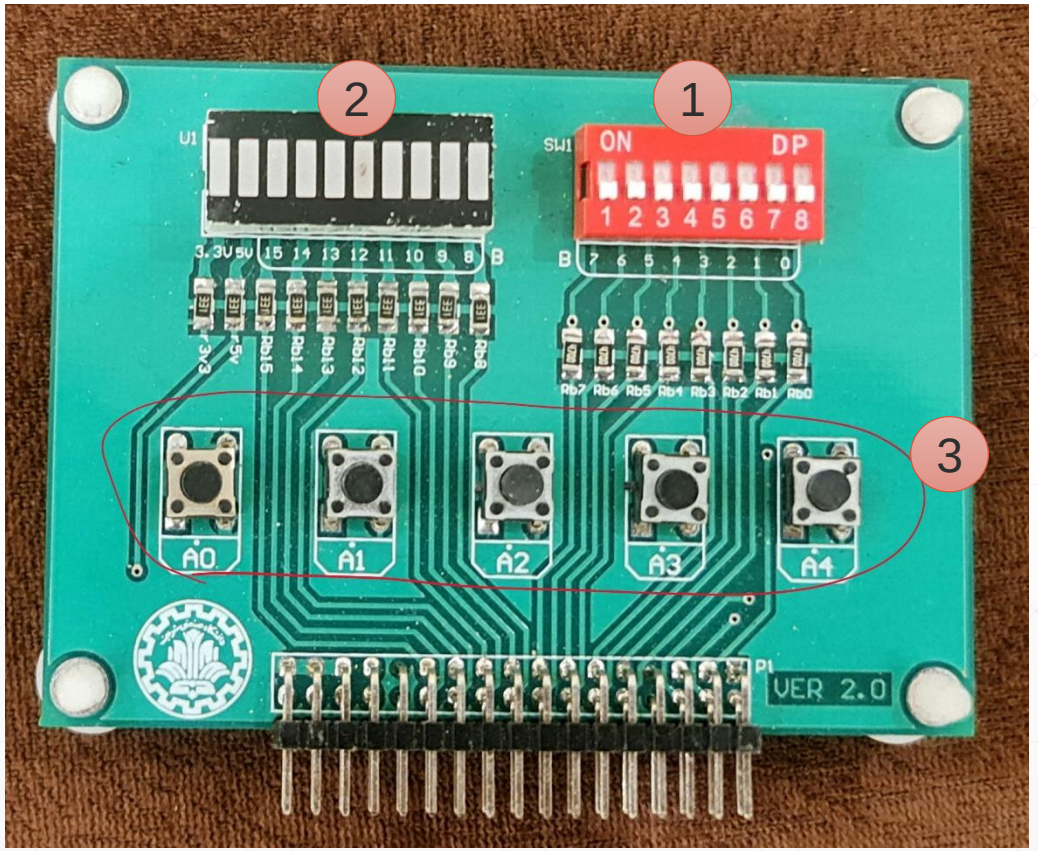
Button-LED board

This board has some fixed buttons and LEDs. You simply need to attach it to the bottom or top of the microcontroller board for the buttons and the LEDs to be connected to the microcontroller pins, A and B.

1. These switches are B0 to B7 pins if connected to the microcontroller board (inputs)

2. These LEDs are B8 to B15 pins and 5V and 3.3V power check if connected to the microcontroller board (outputs)

3. These buttons are A0 to A4 pins if connected to the microcontroller board (inputs). These buttons are pull-down buttons (what does it mean?).

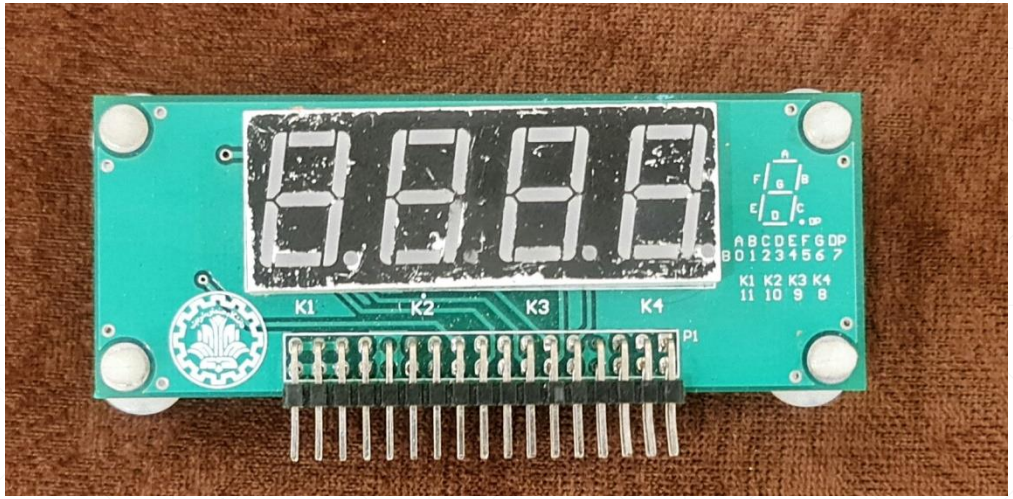


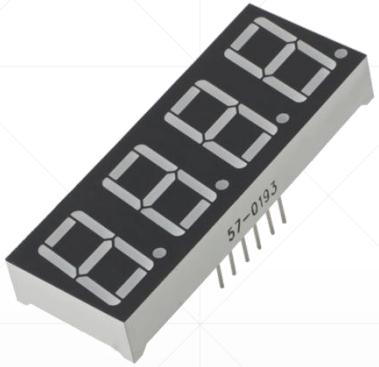
7-Seg board

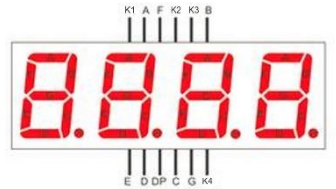
The 4-digit seven-segment display has 12 pins that are connected to B0-B11 if you simply attach it to the bottom or the top of the microcontroller board.

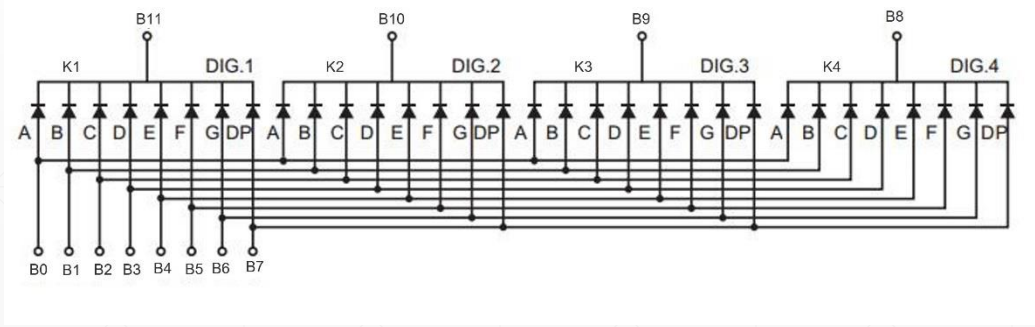
- You will learn to use it in future labs.

This board can NOT be used with “Button-LED board” at the same time





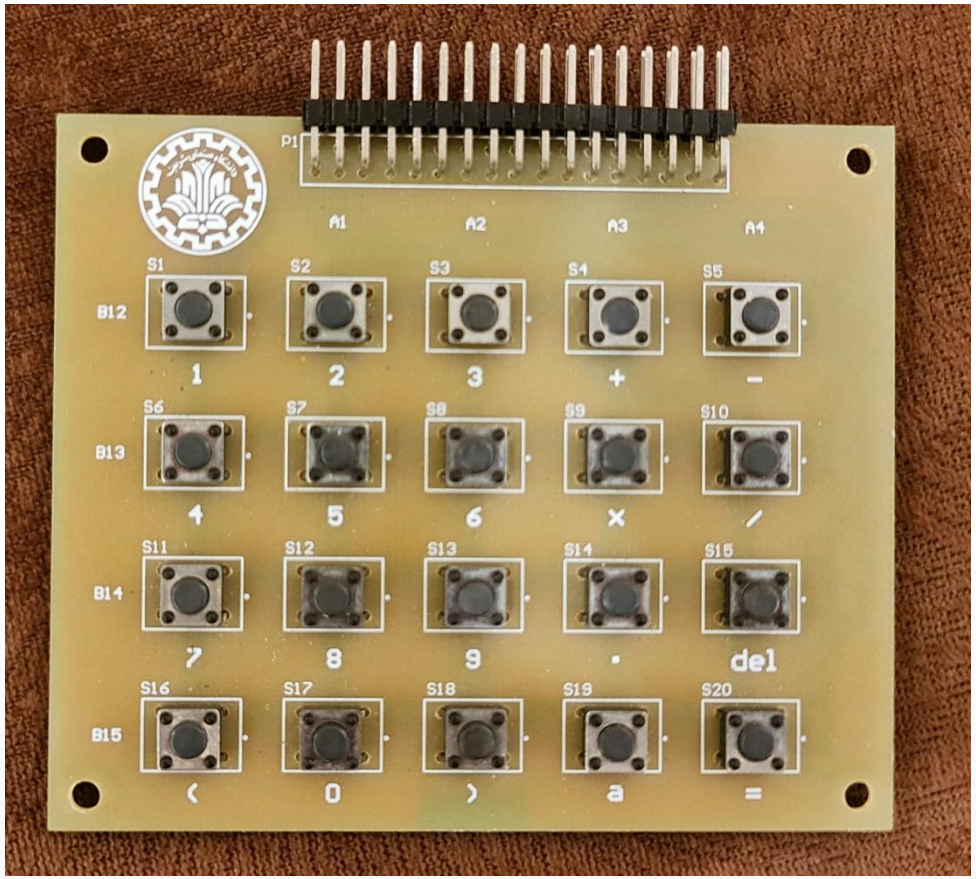




Keypad board

The keypad has 9 pins which are connected to B12-B15 as rows and A0-A4 as columns if you simply attach it to the bottom or the top of the microcontroller board.

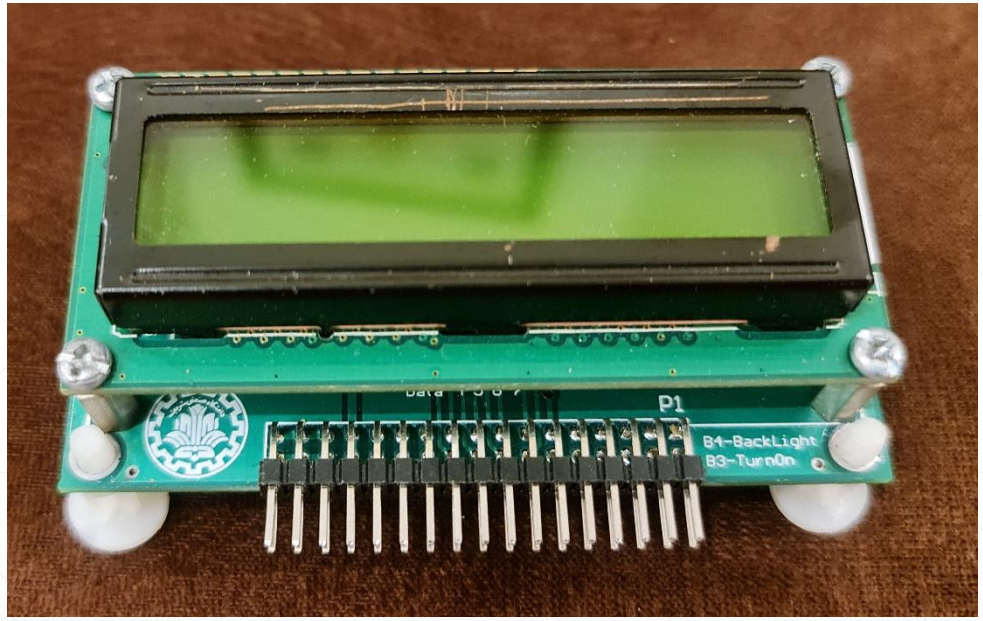
- You will learn to use it in future labs.

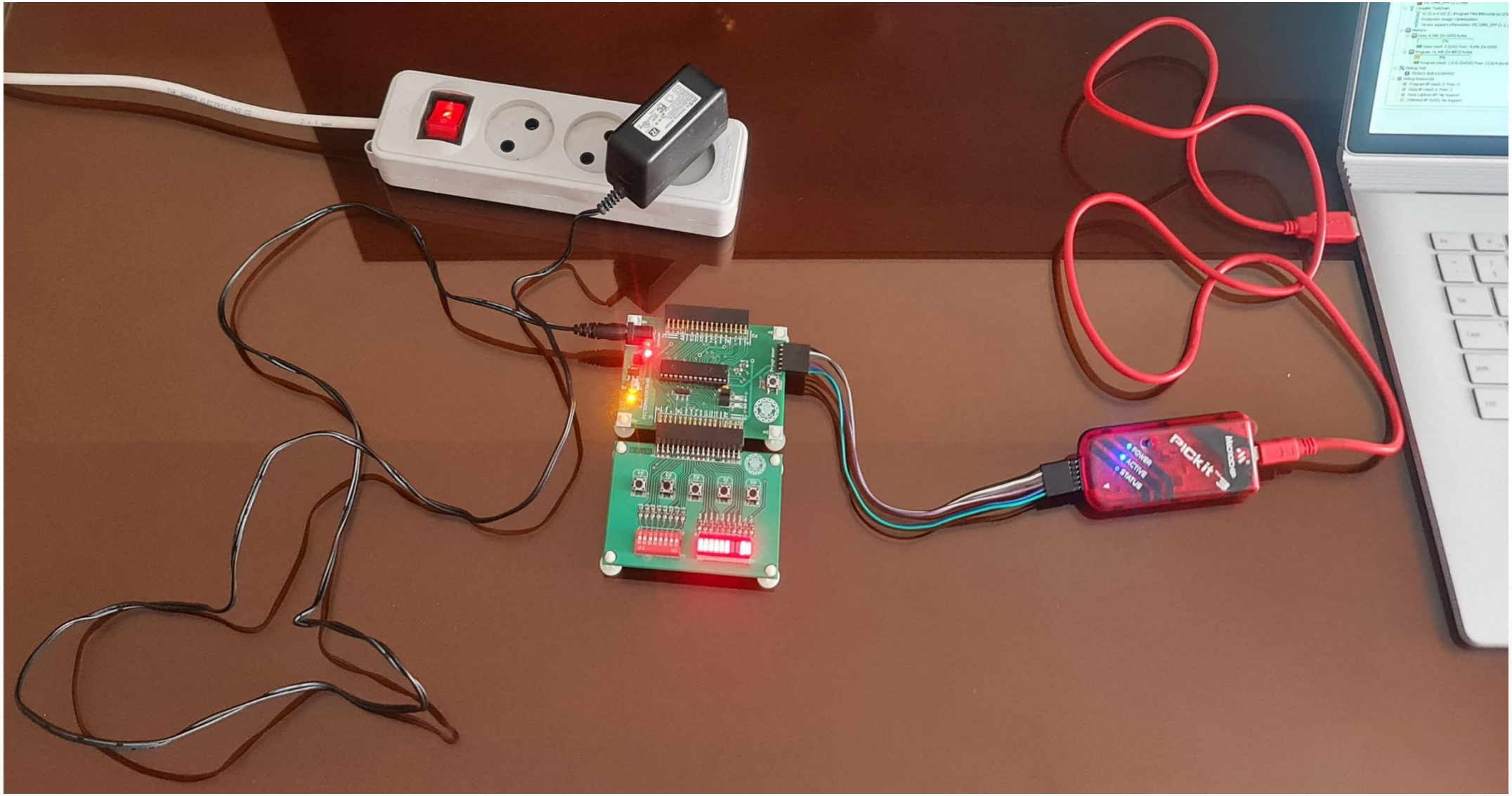


LCD board

The keypad has 8 pins that are connected to B4-B11 if you simply attach it to the bottom or the top of the microcontroller board.

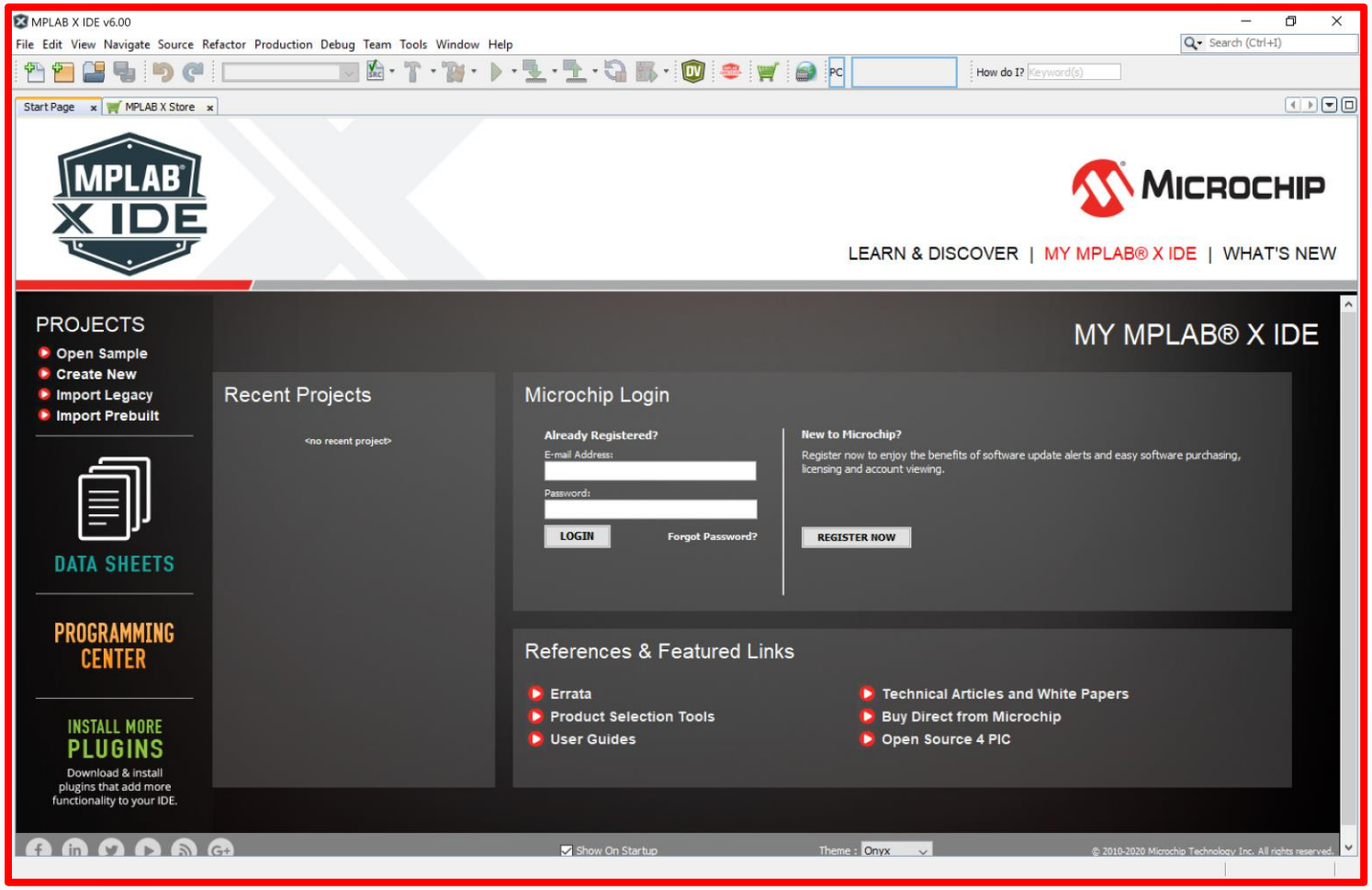
- You will learn to use it in future labs.




POWER
ACTIVE
STATUS
Pickit 3

Getting Started with

MPLAB X IDE



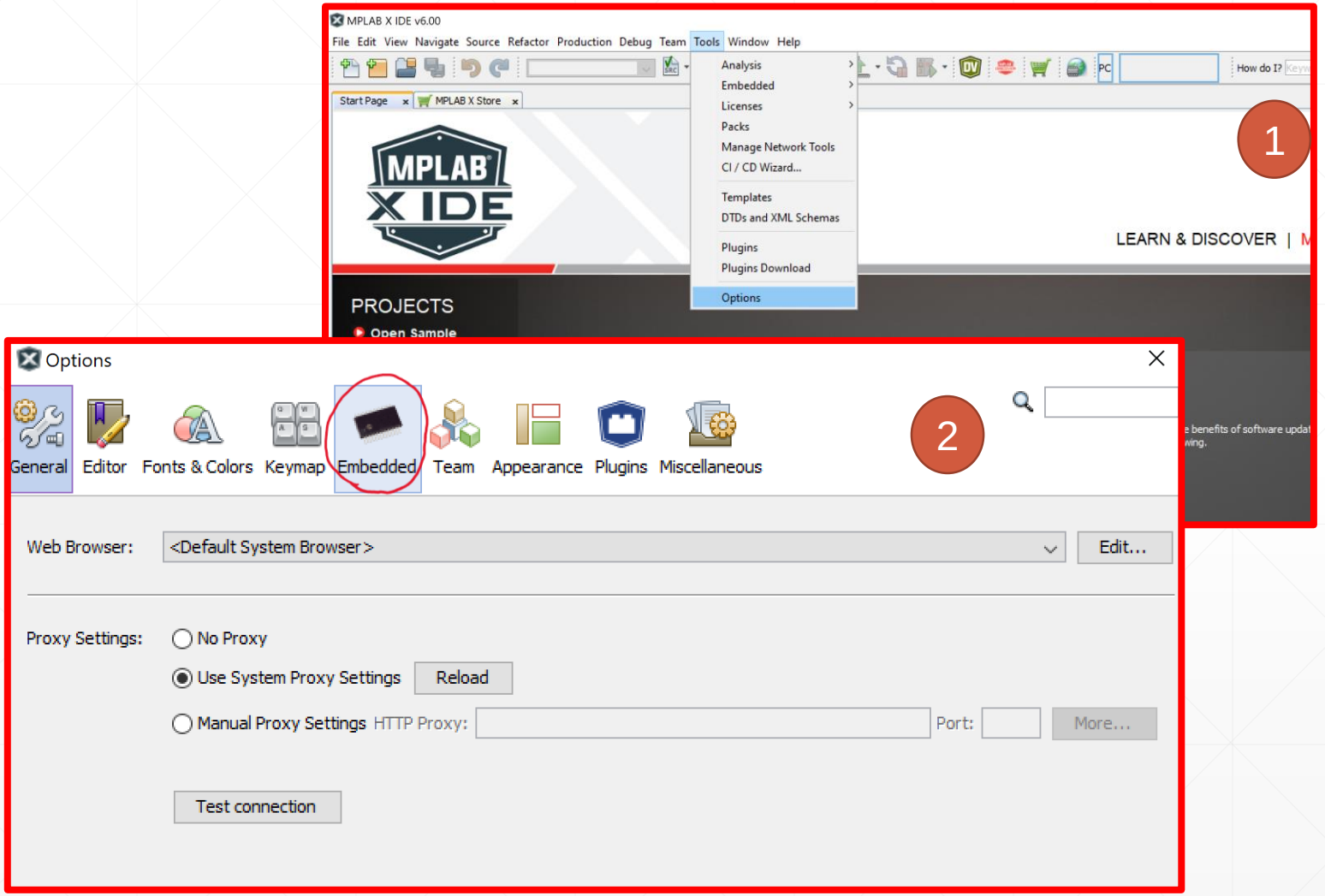
MPLAB X IDE

Microchip's IDE for PIC microcontrollers family.

*NOTE: Make sure you've installed MPLAB X IDE and XC32 compiler before the next lab session.

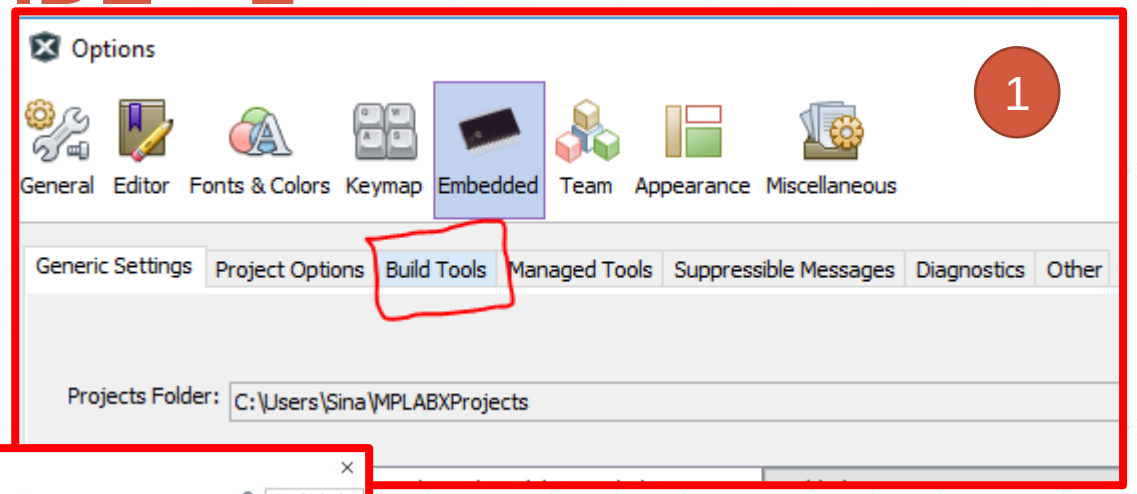
Getting Started with MPLAB X IDE – 1

- Checking whether your XC32 compiler is installed correctly
- Tools > Options >Embedded tab

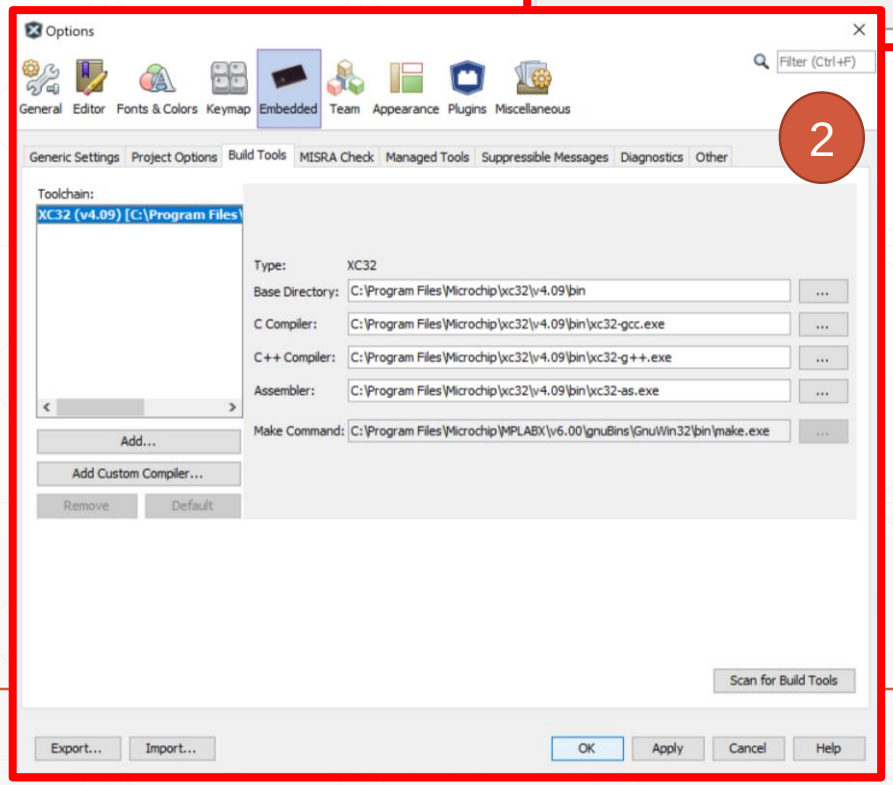


Getting Started with MPLAB X IDE – 2

- Checking whether your XC32 compiler is installed correctly

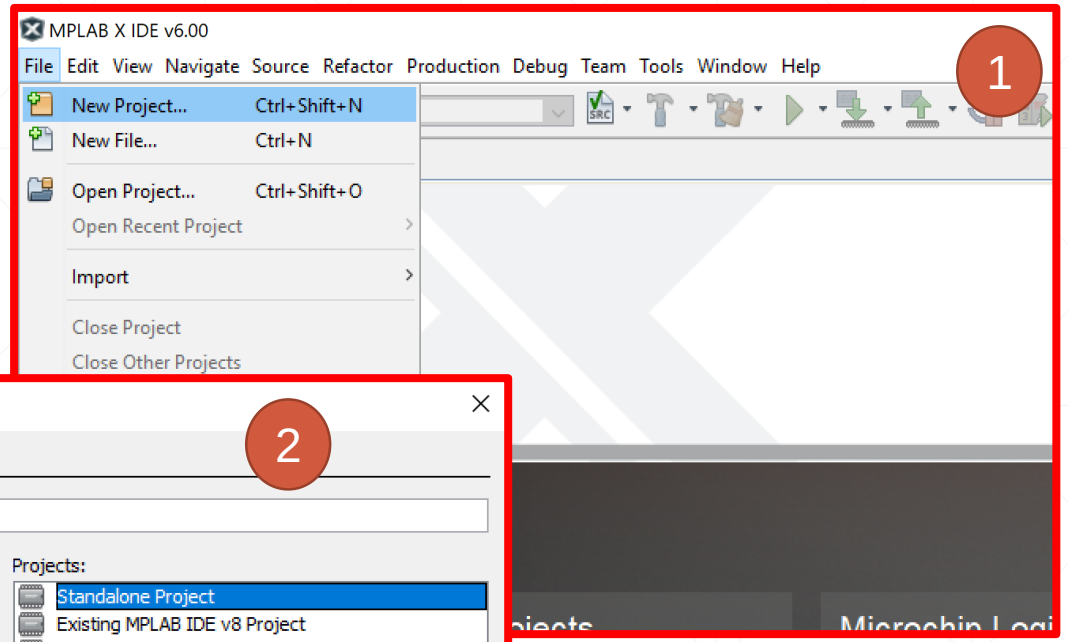


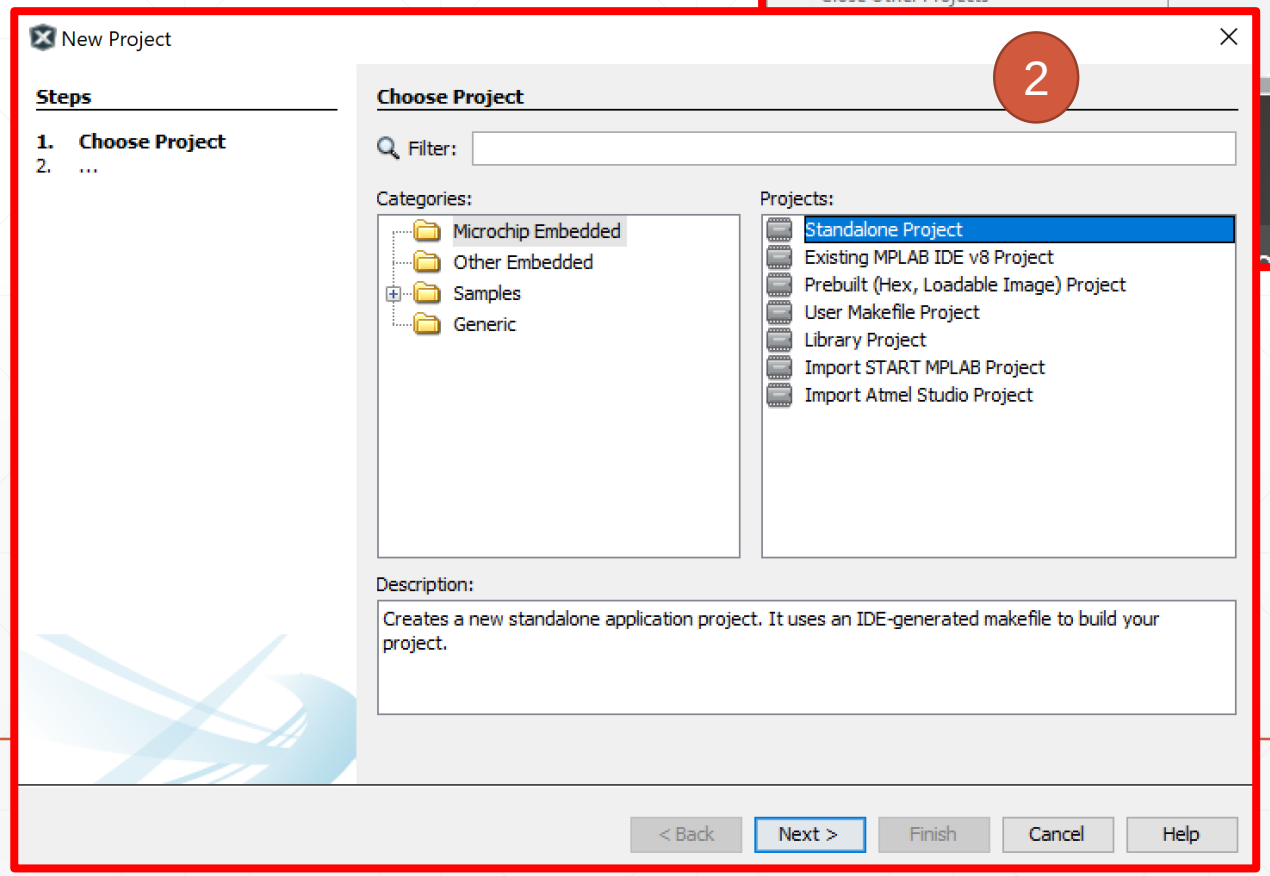
- Build Tools tab > select the base directory where your compiler has been installed



Getting Started with MPLAB X IDE – 3

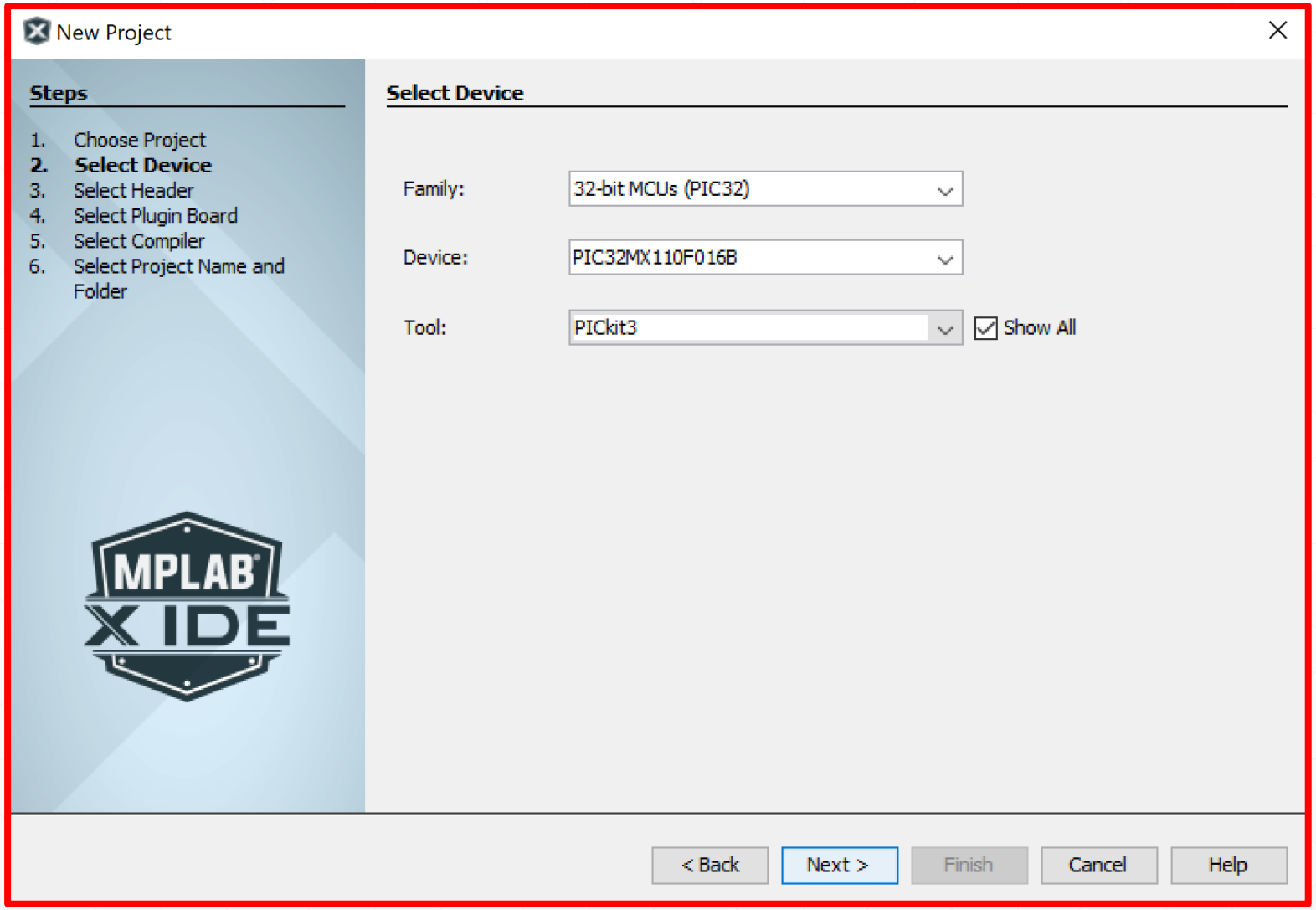
- It’s time to open a new project!
- Select Standalone Project and click on Next>





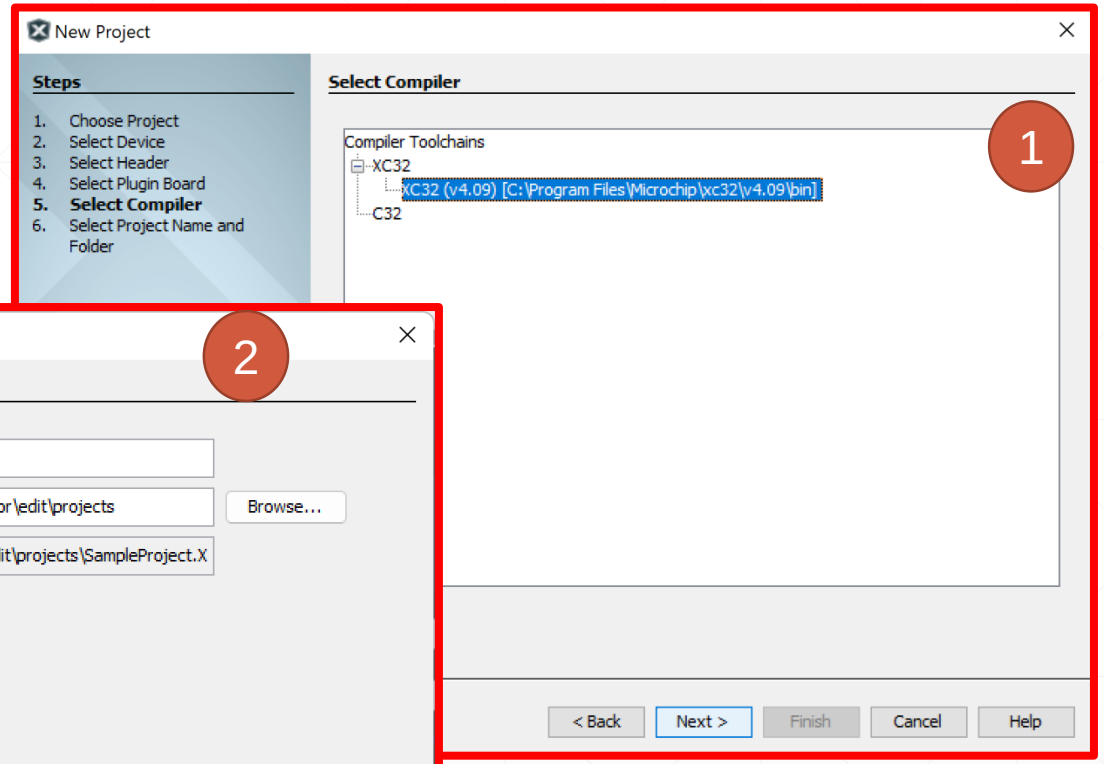
Getting Started with MPLAB X IDE – 4

- Under 32-bit MCUs Family, select PIC32MX110F016B
- Then choose PICkit3 after checkmarking the Show all tick

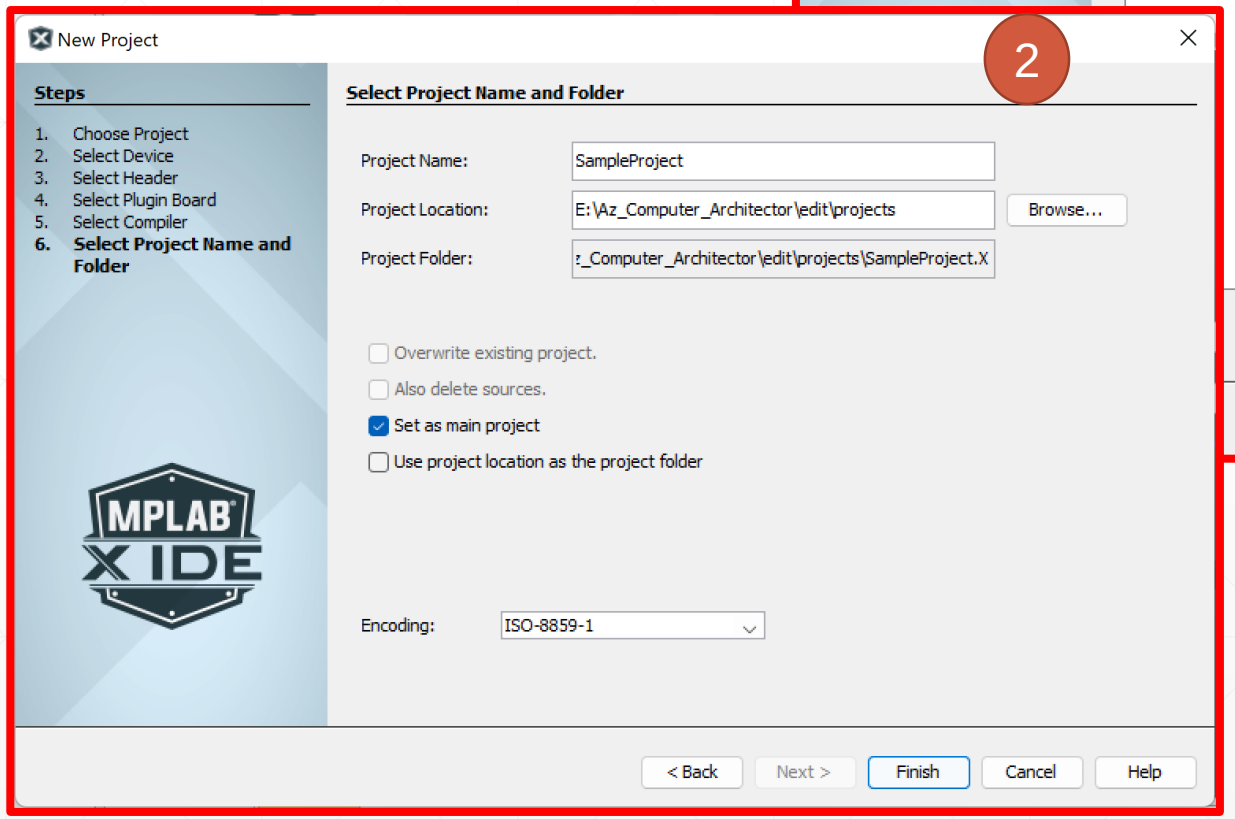


Getting Started with MPLAB X IDE – 5

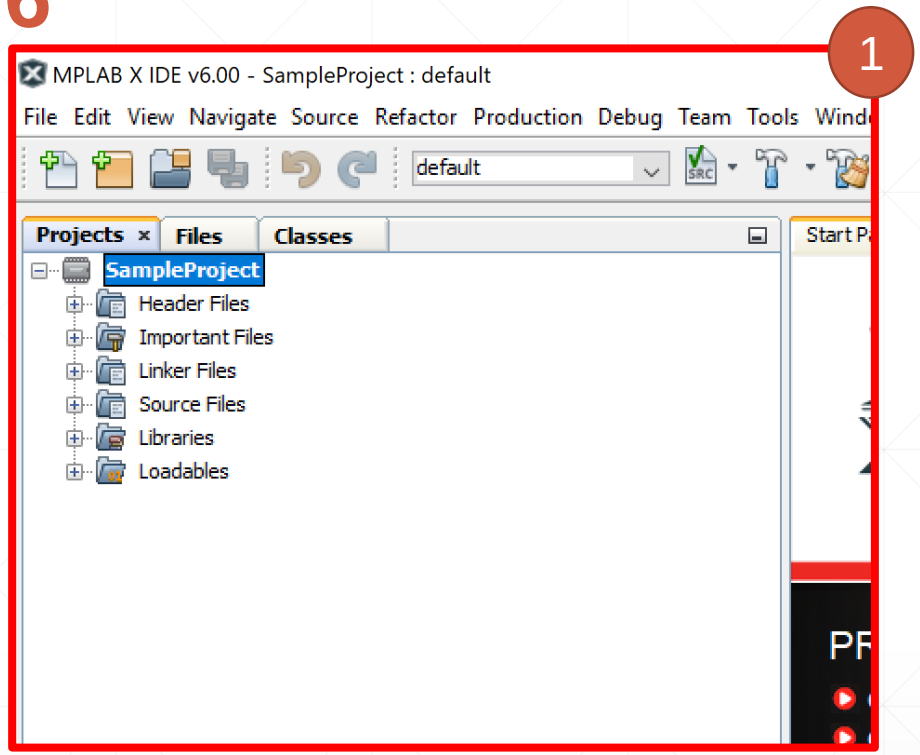
- Select XC32 as your compiler toolchain.



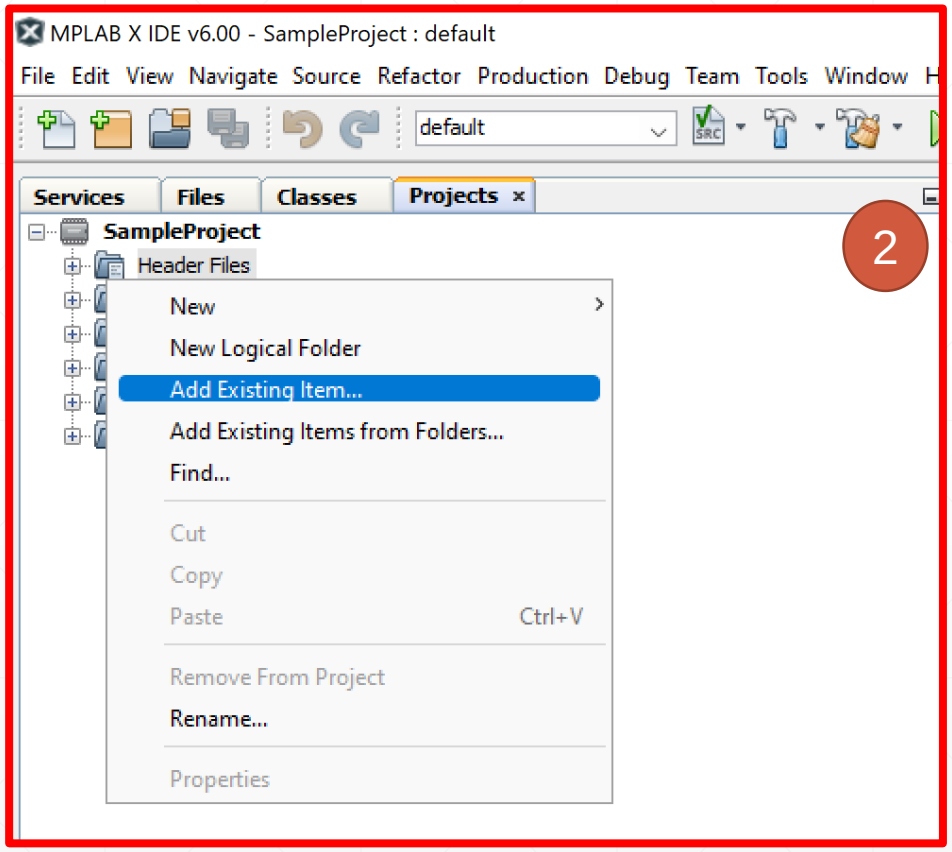
- Choose a name for your project and a location to save it.
- Click Finish!



Getting Started with MPLAB X IDE – 6



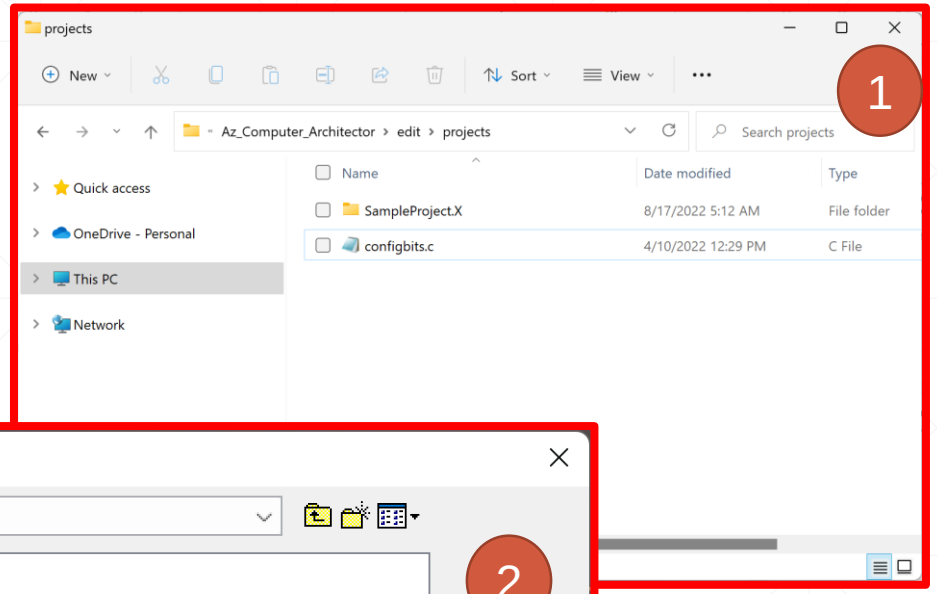
- You can see your project here.

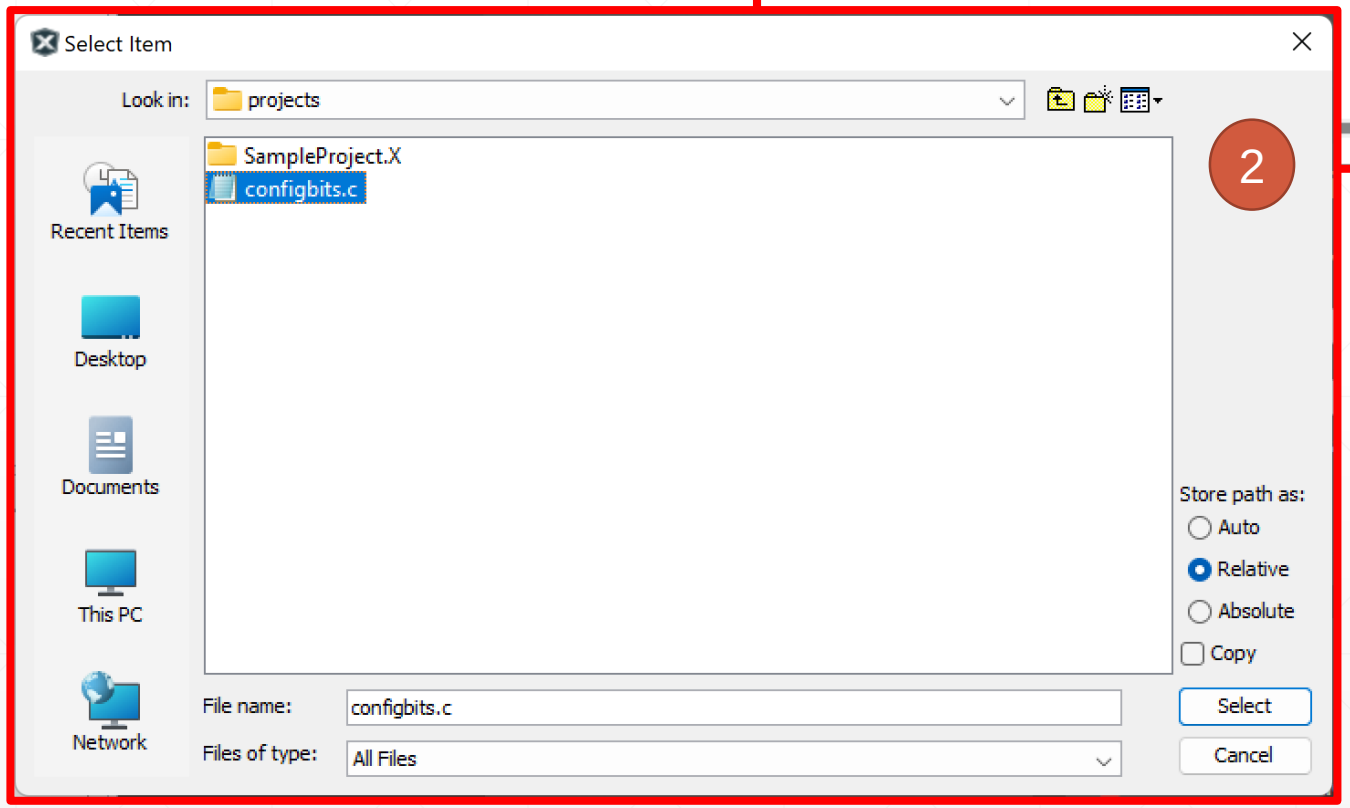


- Now you're going to add some files to your project.
- Right-click on Header Files > Add Existing Item…

Getting Started with MPLAB X IDE – 7

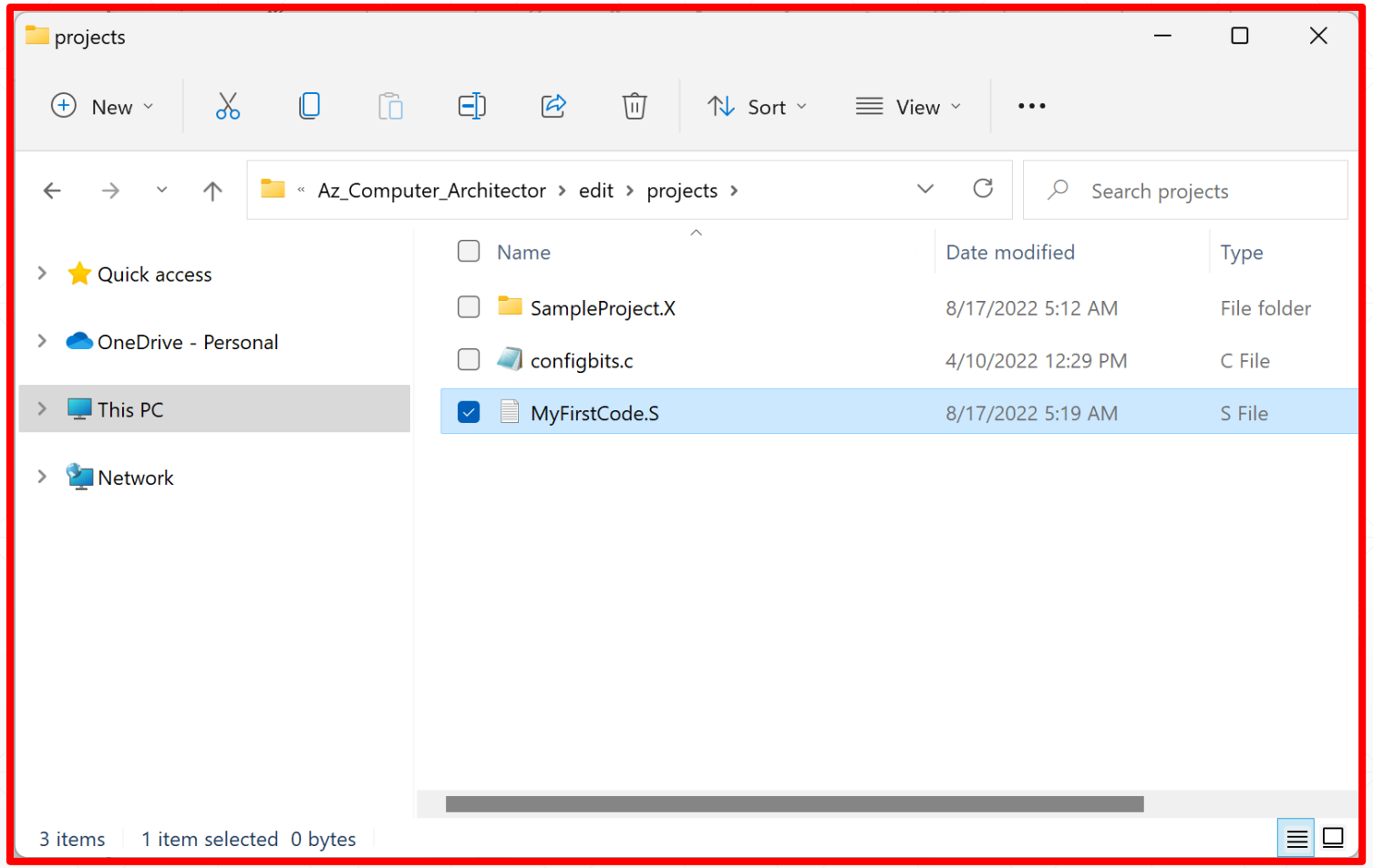
- You have been provided a file named “configbits.c” for the first lab session. Copy it to the directory where you have saved your project and then add it here as a header file. Extra information on this file will be provided later.





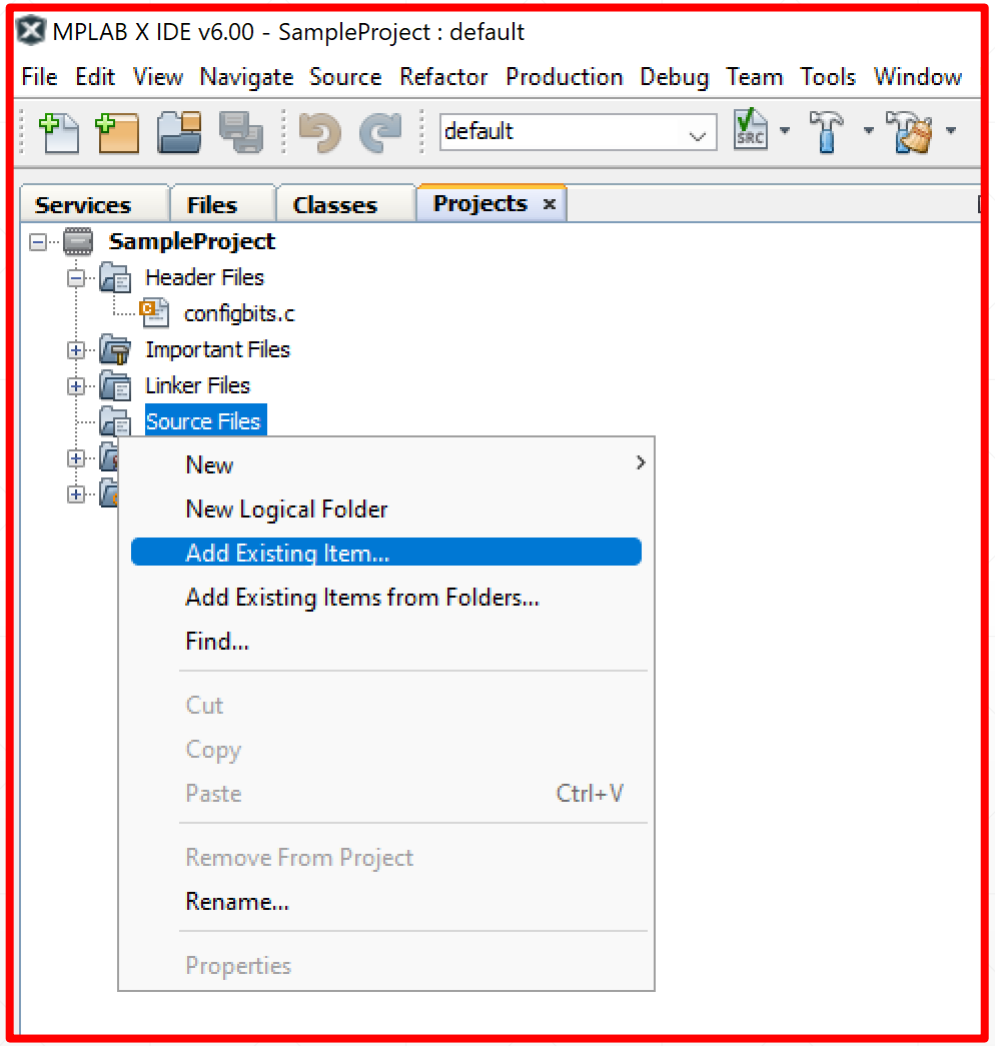
Creating your assembly file

- Create a text file in the directory of your project, and then save it with a ".S" extension.
- Note the capital S. (not .s)
- This will be the main file in which you will write your assembly code.



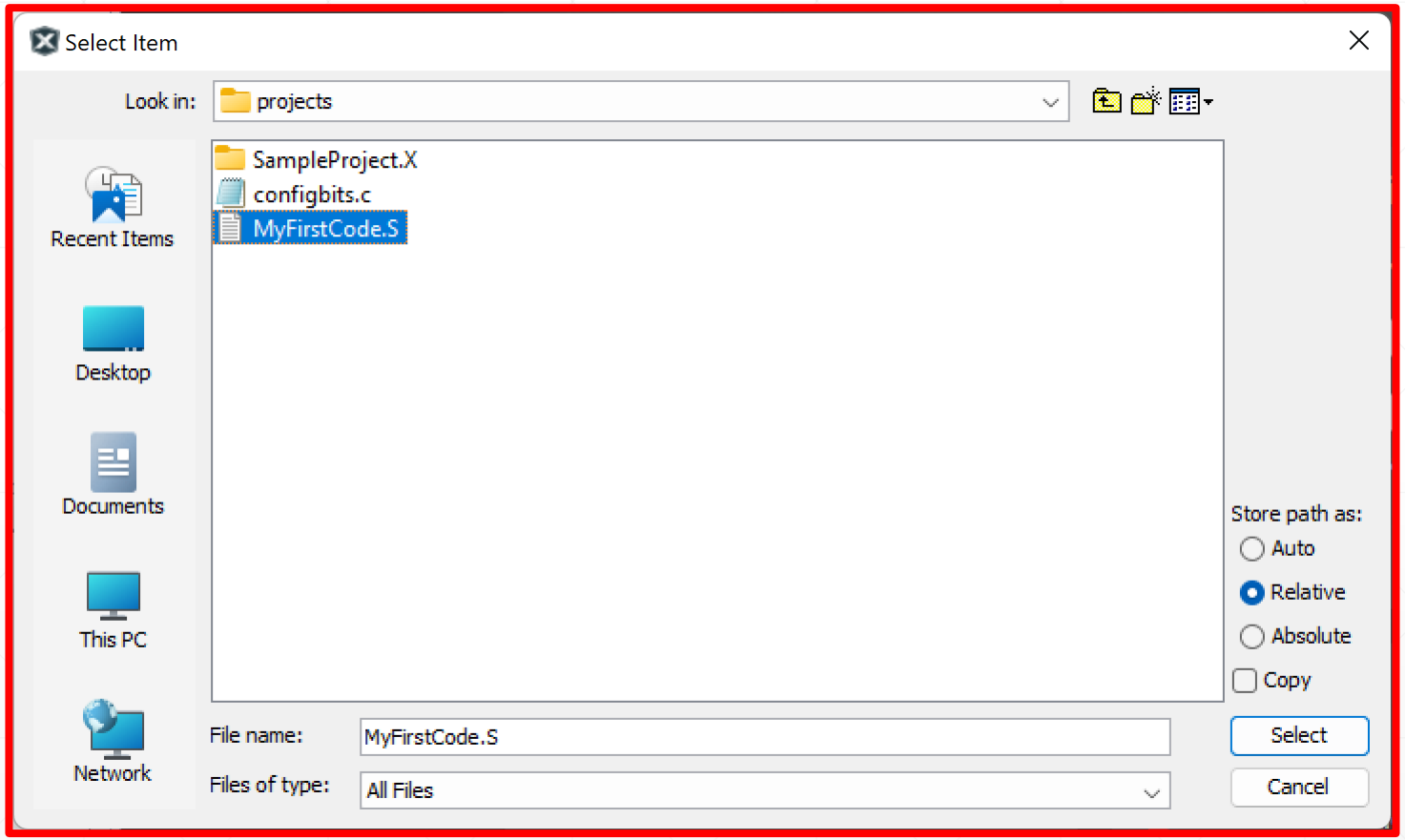
Getting Started with MPLAB X IDE – 8

- You're going to add the file you just created, to your project.
- Right-click on Source Files > Add Existing Item…



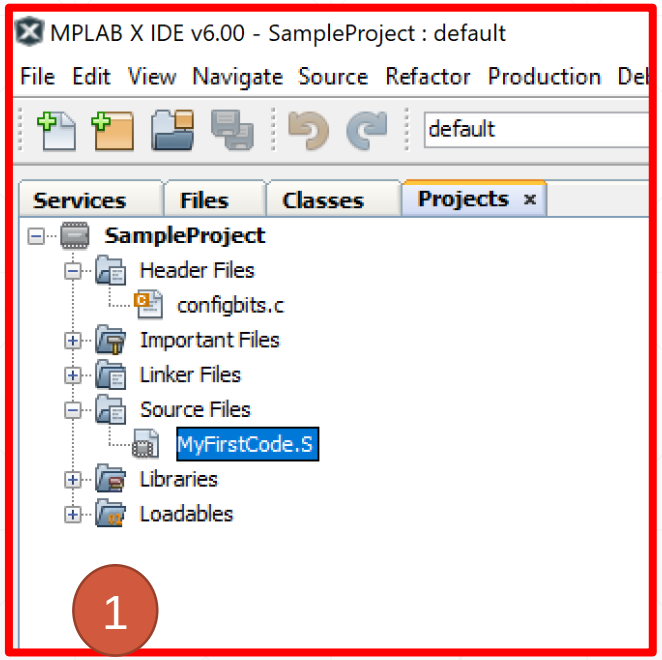
Getting Started with MPLAB X IDE – 9

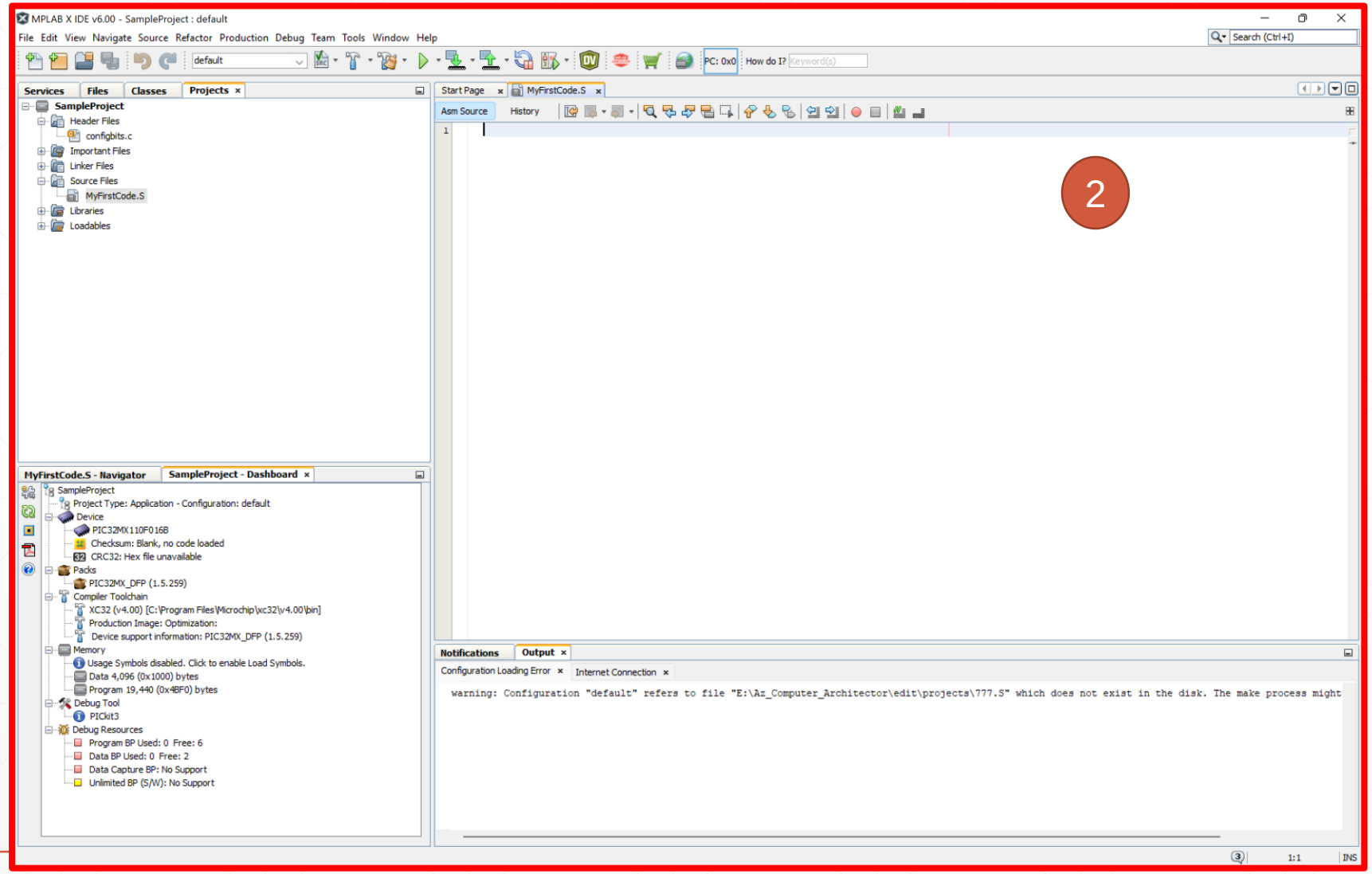
- Now add the .S file you just created.



Getting Started with MPLAB X IDE – 10

- Now you can double-click on the assembly file to open it and start coding!

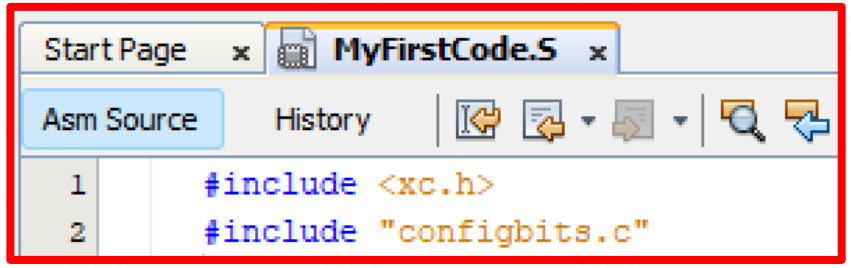




Write Your First Program!

First Things First!

- Include the compiler library.
- Include the header file you've added.

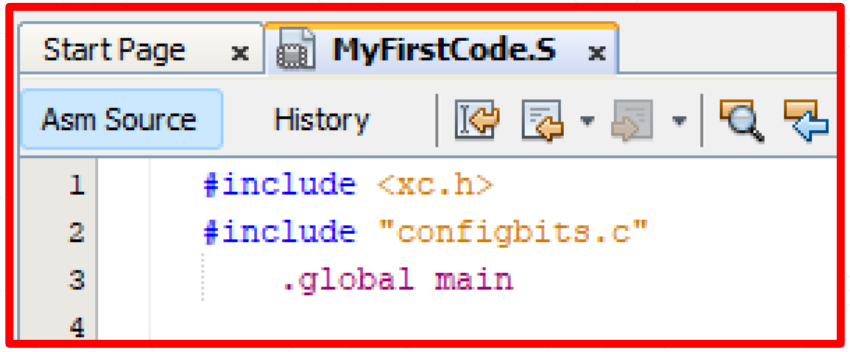



```
#include <xc.h>
#include "configbits.c"
```

.global

- .global :

Define all global symbols here, i.e. main in this program




```
#include <xc.h>
#include "configbits.c"
    .global main
```

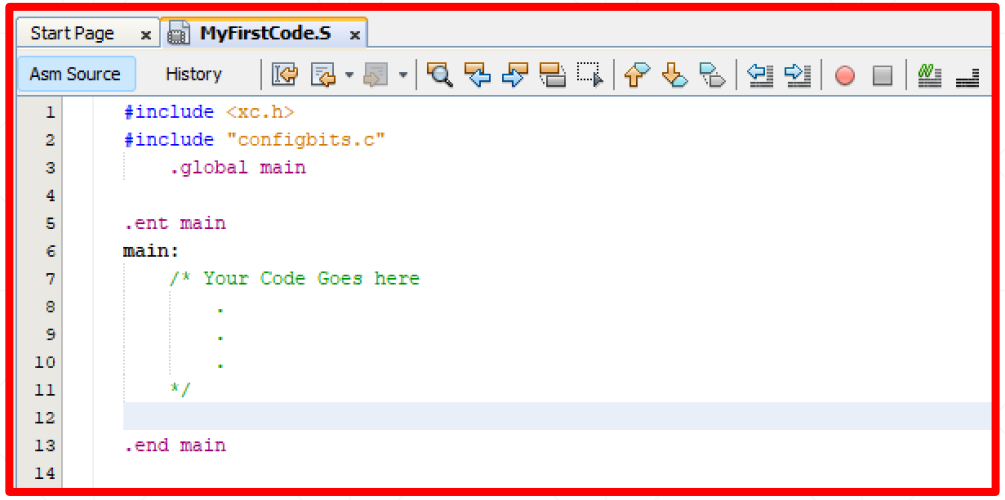
main

- main

This is where the PIC32 start-up code will jump to after initial set-up.

It's just as in C code where we had main() function.

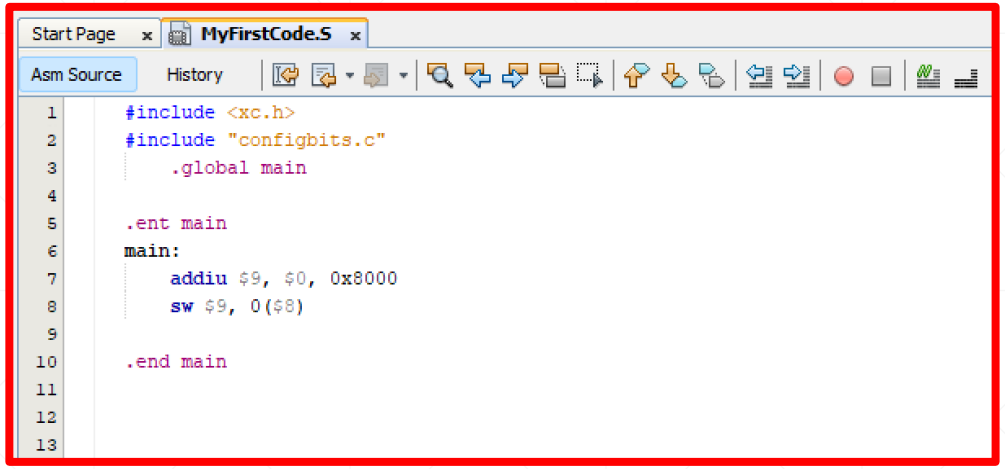
So your code will always begin with .ent main and end with .end main.




```
#include <xc.h>
#include "configbits.c"
    .global main

.ent main
main:
    /* Your Code Goes here
                    .
                    .
                    .
    */

.end main
```

How to code!

- The instruction set for this IC is exactly the same as that of MIPS32 instruction set you've learned.




```
#include <xc.h>
#include "configbits.c"
    .global main

.ent main
main:
    addiu $9, $0, 0x8000
    sw $9, 0($8)

.end main
```

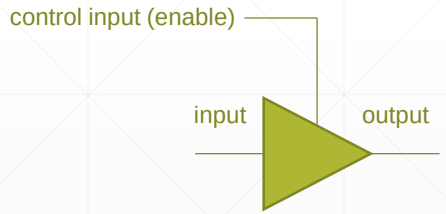
I/O ports

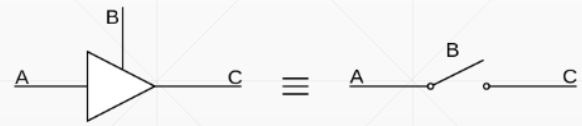
Tri-state buffer

Tri-state buffer: It serves as a wire if the control input is 1 (meaning the buffer is active, transmitting data from its input to its output). But the output is high impedance if the control input is 0, and both circuits connected to the buffer's input and output are completely isolated from each other. (The buffer serves as an open circuit if control input = 0.)

To have a pin with the capability of being both input and output, a tri-state buffer should be used. In this case, the control input is set if the pin is supposed to serve as an output, transmitting data from the circuit at the input of the buffer to the outer pin, and the control input is cleared if the pin is supposed to serve as an input. In the latter one, if we do not use a buffer and use a wire instead we will see X as the value of the wire while two values are driven to it.

input	Control input	output
0	0	Hi-Z
1	0	Hi-Z
0	1	0
1	1	1



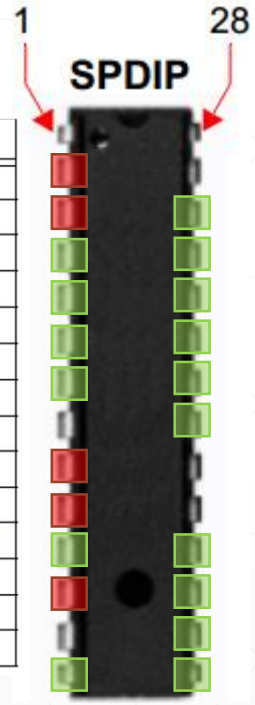


I/O Ports

- PIC32MX110F016B has 2 I/O ports: PORTA & PORTB
- PORTA consists of 5 pins named RA0 through RA4
- PORTB consists of 16 pins named RB0 through RB15
- Each I/O pin may have other applications. For instance, RA2 may be used as OSC1 (an outer oscillator for the CPU clock) while programming the microcontroller. In this course they are only used as I/O ports.

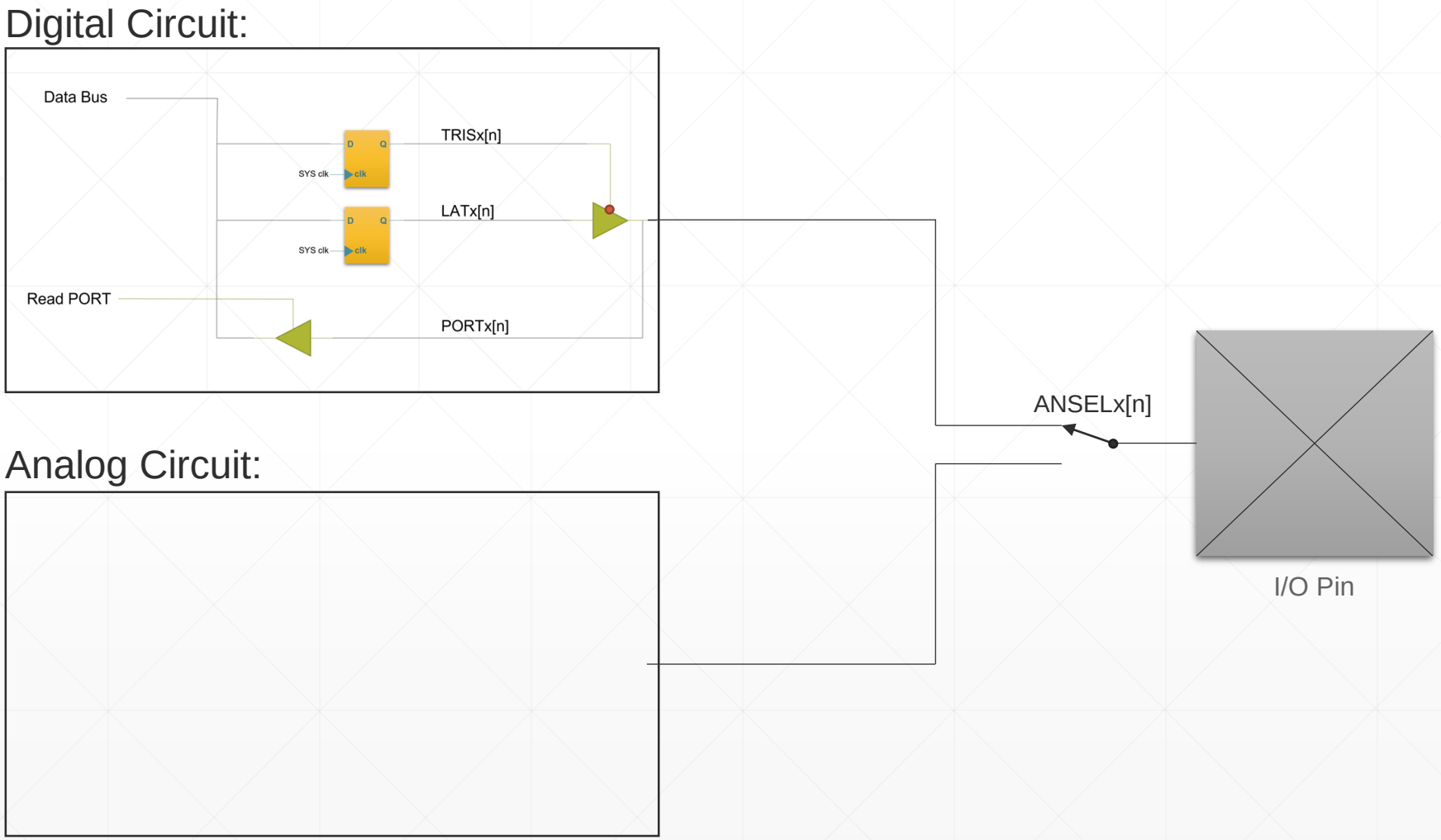
Pin #	Full Pin Name
1	MCLR
2	VREF+/CVREF+/AN0/C3INC/RPA0/CTED1/RA0
3	VREF-/CVREF-/AN1/RPA1/CTED2/RA1
4	PGED1/AN2/C1IND/C2INB/C3IND/RPB0/RB0
5	PGEC1/AN3/C1INC/C2INA/RPB1/CTED12/RB1
6	AN4/C1INB/C2IND/RPB2/SDA2/CTED13/RB2
7	AN5/C1INA/C2INC/RTCC/RPB3/SCL2/RB3
8	VSS
9	OSC1/CLKI/RPA2/RA2
10	OSC2/CLKO/RPA3/PMA0/RA3
11	SOSCI/RPB4/RB4
12	SOSCO/RPA4/T1CK/CTED9/PMA1/RA4
13	VDD
14	PGED3/RPB5/PMD7/RB5

Pin #	Full Pin Name
15	PGEC3/RPB6/PMD6/RB6
16	TDI/RPB7/CTED3/PMD5/INT0/RB7
17	TCK/RPB8/SCL1/CTED10/PMD4/RB8
18	TDO/RPB9/SDA1/CTED4/PMD3/RB9
19	VSS
20	VCAP
21	PGED2/RPB10/CTED11/PMD2/RB10
22	PGEC2/TMS/RPB11/PMD1/RB11
23	AN12/PMD0/RB12
24	AN11/RPB13/CTPLS/PMRD/RB13
25	CVREFOUT/AN10/C3INB/RPB14/SCK1/CTED5/PMWR/RB14
26	AN9/C3INA/RPB15/SCK2/CTED6/PMCS1/RB15
27	AVSS
28	AVDD



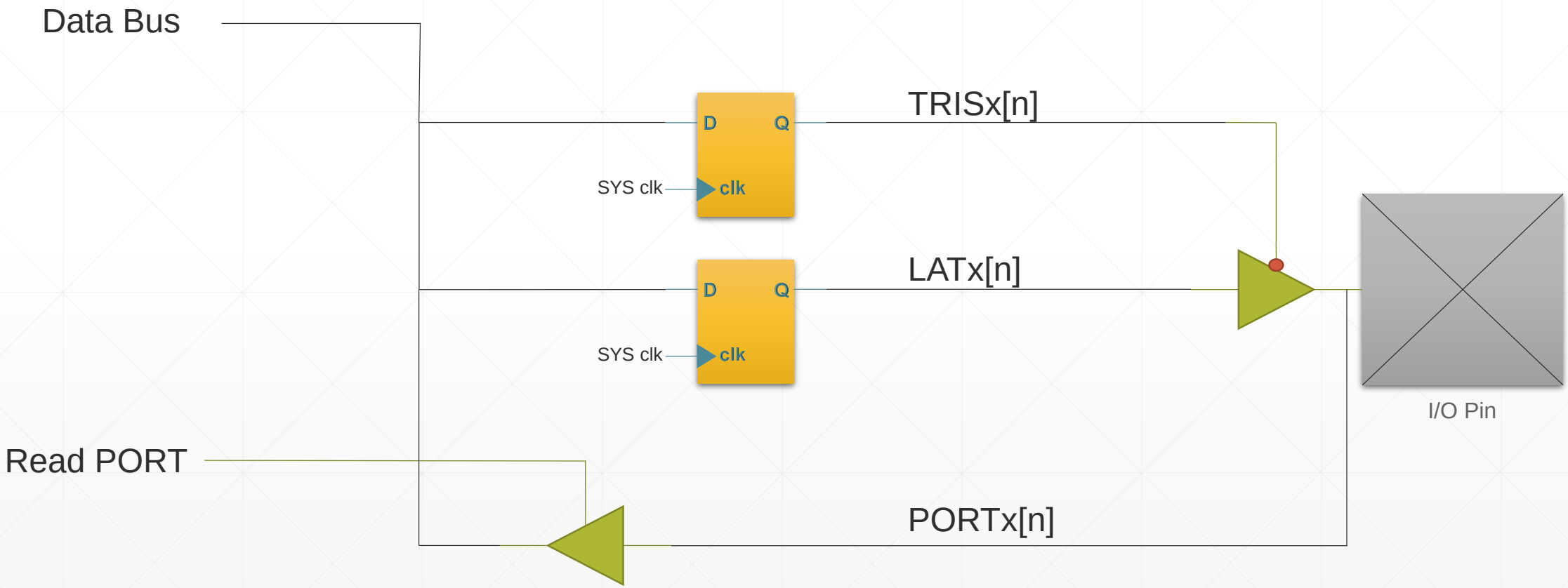
Block diagram I/O Pins

- The ANSELx[n] serves as a switch, connecting the pin to the digital circuit if cleared, while connecting it to the analog circuit if set. You can change the value of the ANSELx[n] in your code.
- In this course we merely use the digital circuit.



Block diagram I/O Pins

The data bus is connected to the memory



ANSELx, TRISx, LATx and PORTx Registers

- Each pin can be used as analog or digital, so you need to control it in your code by adjusting ANSELx bits. The pin is analog if the corresponding ANSELx bit is set and is digital if it is cleared. (Its default value is 1 for every bits of ANSELx.)
- Each pin can also operate as input or output, so you need to control it in your code as well by adjusting TRISx bits. The pin is input if the corresponding TRISx bit is set and is output if it is cleared.
- For instance the RB1 pin is digital and input if

ANSELB = xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xx0x

TRISB = xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xx1x

ANSELx, TRISx, LATx and PORTx Registers

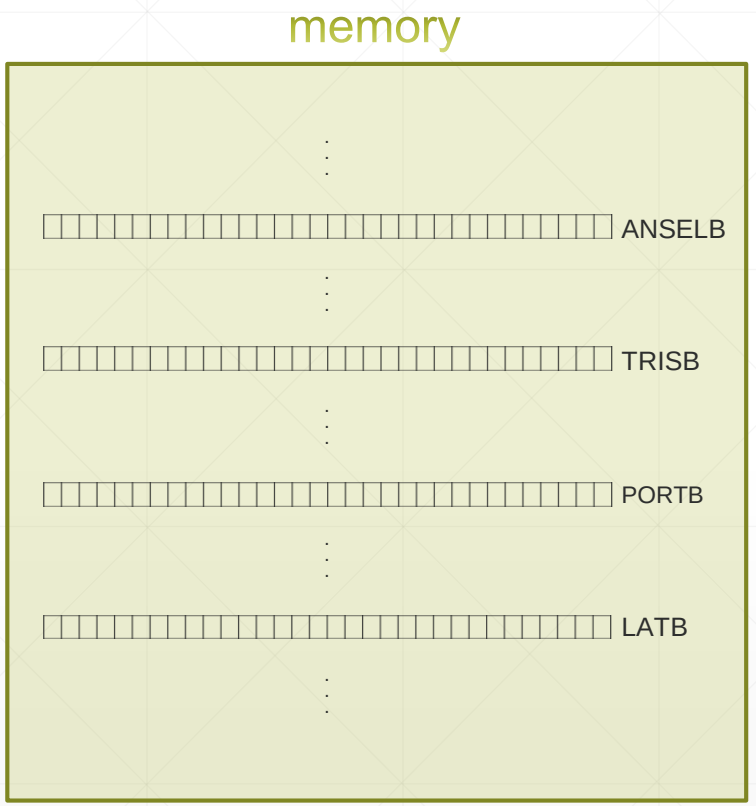
- To have the value of a pin which is input, the value of the corresponding PORTx bit should be read. For example, if we want to read the value of RB1 (while it is set as a digital input), we should read the second LSB bit of PORTB:

PORTB = ????_????_????_????_????_????_????_??**?**?

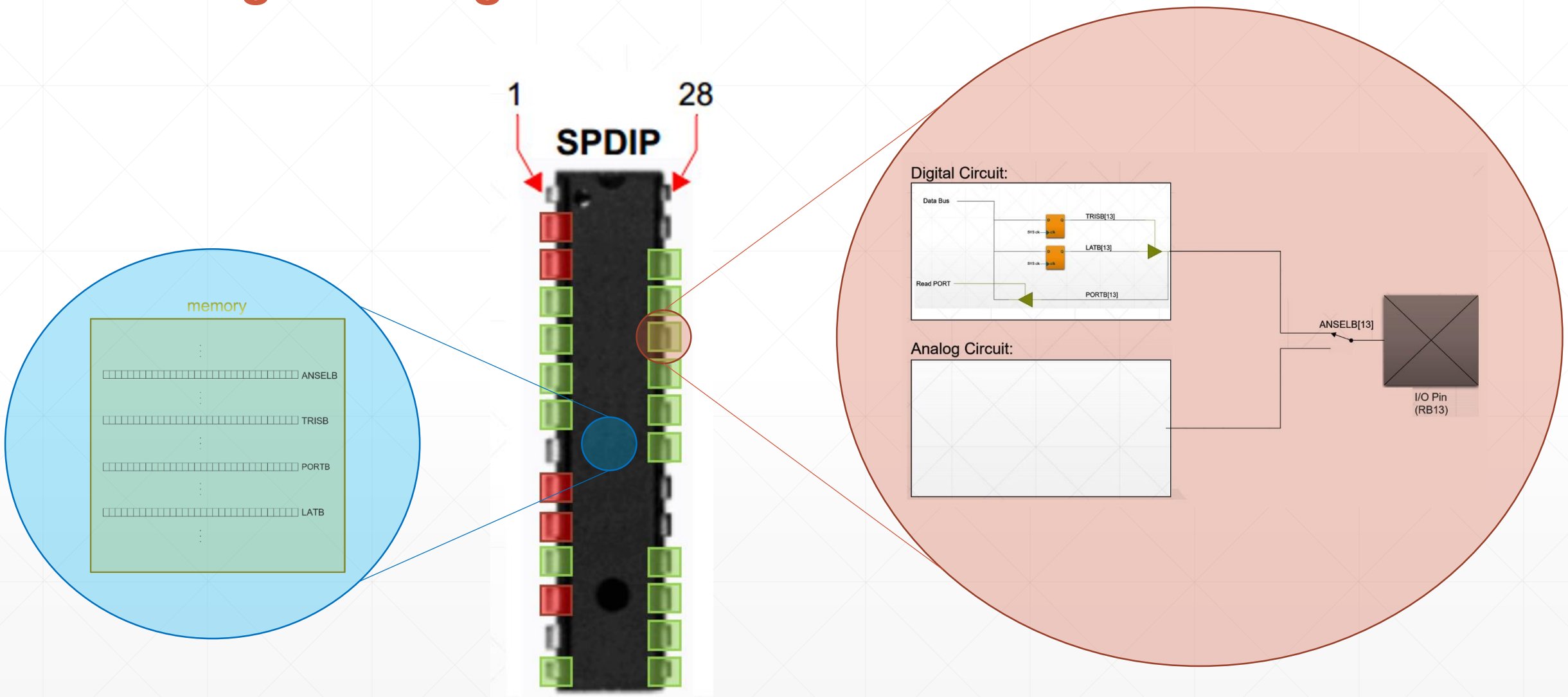
- To give value to a pin which is output, we should give value to the corresponding LATx bit. For example, if we want RA0 to RA3 pins to have the value of 1 (while they are set as digital outputs):

LATA = xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_1111

- 4 words in the memory of the microcontroller are allocated to port B and 4 to port A. You can get access to the words by having simple assembly instructions programmed into the microcontroller if you know the address of the words, like “sw” and “lw”. These words are named ANSELB, TRISB, LATB, PORTB and ANSELA, TRISA, LATA, PORTA. (A simple model of it is shown for port B in the following picture.)



Programming model of an I/O Port



One last point

11.1.3 I/O PORT WRITE/READ TIMING

One instruction cycle is required between a port direction change or port write operation and a read operation of the same port. Typically this instruction would be an `NOP`.

la Pseudo Instruction

- This is the only pseudo instruction you are allowed to use during this course!
- la (load address) is a pseudocode that returns the address of a register on the memory space.

  For instance, la $8, ANSELB will load the address of ANSELB into $8 register.

Literal Value Operands in Assembly

3.3.5.2 LITERAL-VALUE OPERANDS

Literal values can be hexadecimal, octal, binary, or decimal format. Hexadecimal numbers are distinguished by a leading 0x. Octal numbers are distinguished by a leading 0. Binary numbers are distinguished by a leading 0B or 0b. Decimal numbers require no special leading or trailing character.

Example:

`0xe`, `016`, `0b1110` and `14` all represent the literal value 14.

`-5` represents the literal value -5.

`symbol` represents the value of `symbol`.

NOTE: Multiply/Divide instructions in XC32

MIPS mult/multu/div/divu instructions are overloaded in XC32 assembler used in MPLAB.

Let's use **`muldiv`** in place of above four instructions. Then:

1- **`muldiv $i,$j,$k`** is translated to two following instructions and means **`$i = $j */ $k`**

```
muldiv $j, $k
mflo $i
```

2- **`muldiv $j,$k`** is translated to two following instructions and means **`$j = $j */ $k`**

```
muldiv $j, $k
mflo $j
```

Hint: If you need exact operation of MIPS mult/div instructions then use: **`muldiv $0,$j,$k`**

Sample Program

- Here is a sample code which is meant to read data from RB0 and show it on RB8 continuously.
- This code is sent to the microcontroller and saved in its program memory to run from the beginning to the end.

Start Page | MyFirstCode.S

```
/*****************************************************************************
 * Description:
 * This project demonstrates how to program the PIC32 microcontroller so as to show the state of a switch (attached to the RB0 pin) on an LED (attached to the RB8 pin), in real time.
 *****************************************************************************/
#include <xc.h>
#include "configbits.c"
    .global main

.ent main
main:
    // As we want the B port (meaning pins B0, B1, ..., B15) to be digital I/O, ANSELB = 0x0000:
    la $8, ANSELB// Now $8 contains the memory address of ANSELB (which is 0xbf88_6100 as the datasheet asserts), so now we can edit the ANSELB.
    sw $0, 0($8)// ANSELB = 0x0000
    /* The equivalent instruction for the instructions above is:
     * lui $8, 0xbf88
     * sw $0, 0x6100($8)
     */


    /* Now we are going to assign B0, B1, ..., B7 as inputs to read from these pins and B8, B9, ..., B15 as outputs. (Note that in this particular project we do not need RA pins)
     * So TRISB[0] = TRISB[1] = ... = TRISB[7] = 1 and TRISB[8] = TRISB[9] = ... = TRISB[15] = 0.
     * Therefore TRISB = 0x00ff
     */
    la $8, TRISB// Now $8 contains the memory address of TRISB (which is 0xbf88_6110 as the datasheet asserts), so now we can edit the TRISB.
    ori $9, $0, 0x00ff
    sw $9, 0($8)// TRISB = 0x00ff
    /* The equivalent instruction for the instructions above is:
     * lui $8, 0xbf88
     * ori $9, $0, 0x00ff
     * sw $9, 0x6110($8)
     */
    // Now RB0 to RB7 are set as inputs and RB8 to RB15 are set as outputs.

    // We should have the address of PORTB to read from it and to detect any input; while we also need to have the address of LATB to write in it and show outputs:
    la $10, PORTB// Now $10 contains the memory address of PORTB (wich is 0xbf88_6120 as the datasheet asserts), so now we can read PORTB.
    la $11, LATB// Now $11 contains the memory address of LATB (which is 0xbf88_6130 as the datasheet asserts), so now we can edit the LATB.


    loop:
        lw $8, 0($10)// The input to our pins from an outer element (the swithces) is live detected and saved in the $8 register from PORTB.
        andi $8, $8, 0x0001// The $8 is masked somehow we only detect RB0. Now the LSB of $8 shows whether the RB0 switch has been set.
        sll $8, $8, 8
        sw $8, 0($11)// The RB0 input is copied to the RB8 output, live.
    j loop


.end main
```

Sample Program

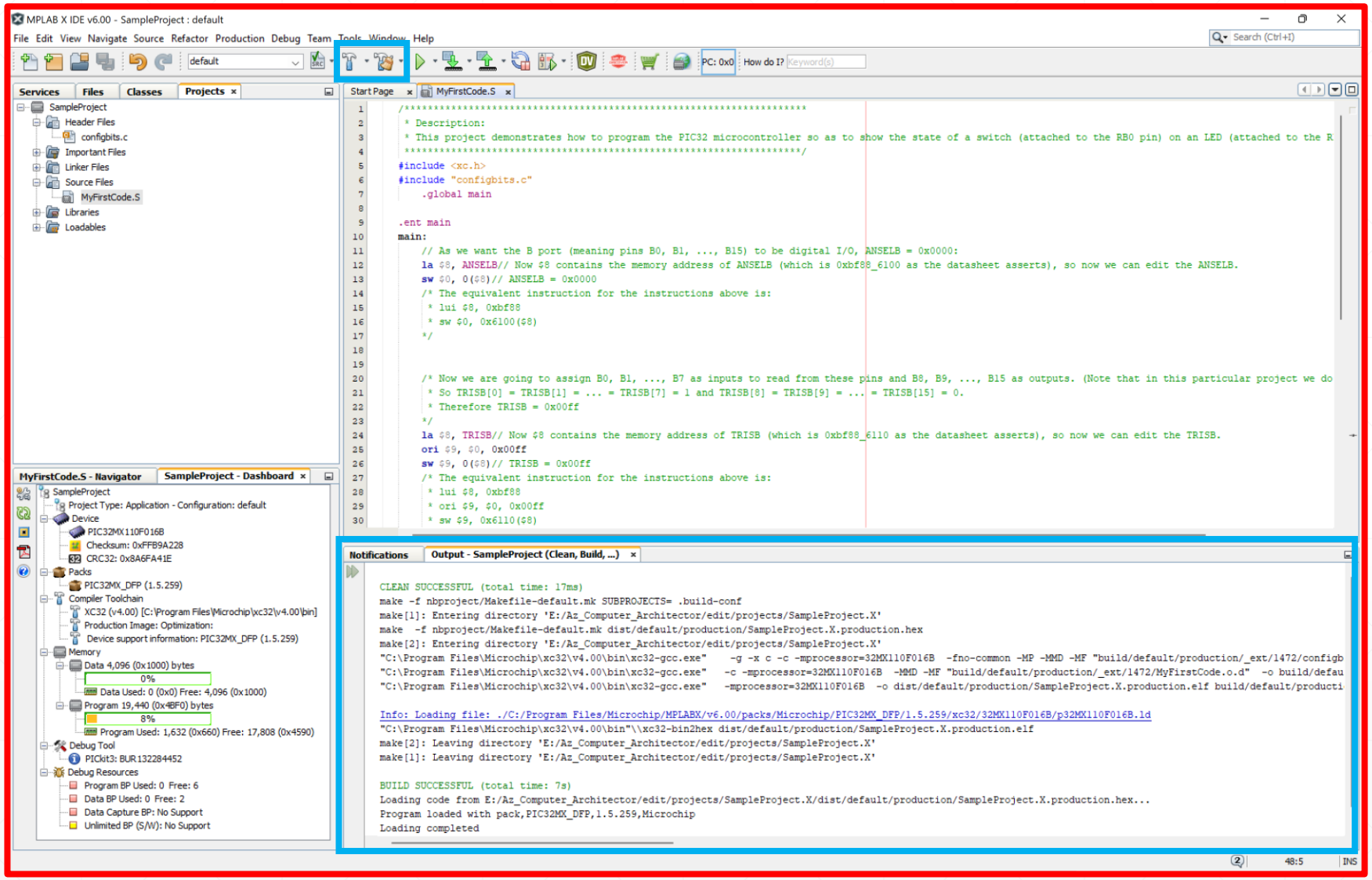
```
/*****************************************************************************
 * Description:
 * This project demonstrates how to program the PIC32 microcontroller so as to show the state of a switch (attached to the RB0 pin) on an LED (attached to the RB8 pin), in real time.
 *****************************************************************************/
#include <xc.h>
#include "configbits.c"
    .global main

.ent main
main:
    // As we want the B port (meaning pins B0, B1, ..., B15) to be digital I/O, ANSELB = 0x0000:
    la $8, ANSELB// Now $8 contains the memory address of ANSELB (which is 0xbf88_6100 as the datasheet asserts), so now we can edit the ANSELB.
    sw $0, 0($8)// ANSELB = 0x0000
    /* The equivalent instruction for the instructions above is:
     * lui $8, 0xbf88
     * sw $0, 0x6100($8)
    */


    /* Now we are going to assign B0, B1, ..., B7 as inputs to read from these pins and B8, B9, ..., B15 as outputs. (Note that in this particular project we do not need RA pins)
     * So TRISB[0] = TRISB[1] = ... = TRISB[7] = 1 and TRISB[8] = TRISB[9] = ... = TRISB[15] = 0.
     * Therefore TRISB = 0x00ff
    */
    la $8, TRISB// Now $8 contains the memory address of TRISB (which is 0xbf88_6110 as the datasheet asserts), so now we can edit the TRISB.
    ori $9, $0, 0x00ff
    sw $9, 0($8)// TRISB = 0x00ff
    /* The equivalent instruction for the instructions above is:
     * lui $8, 0xbf88
     * ori $9, $0, 0x00ff
     * sw $9, 0x6110($8)
    */
    // Now RB0 to RB7 are set as inputs and RB8 to RB15 are set as outputs.

    // We should have the address of PORTB to read from it and to detect any input; while we also need to have the address of LATB to write in it and show outputs:
    la $10, PORTB// Now $10 contains the memory address of PORTB (which is 0xbf88_6120 as the datasheet asserts), so now we can read PORTB.
    la $11, LATB// Now $11 contains the memory address of LATB (which is 0xbf88_6130 as the datasheet asserts), so now we can edit the LATB.


    loop:
                    lw $8, 0($10)// The input to our pins from an outer element (the swithces) is live detected and saved in the $8 register from PORTB.
                    andi $8, $8, 0x0001// The $8 is masked somehow we only detect RB0. Now the LSB of $8 shows whether the RB0 switch has been set.
                    sll $8, $8, 8
                    sw $8, 0($11)// The RB0 input is copied to the RB8 output, live.
    j loop


.end main
```

Programming Your Code into the Microcontroller

Getting Started with MPLAB X IDE – 11

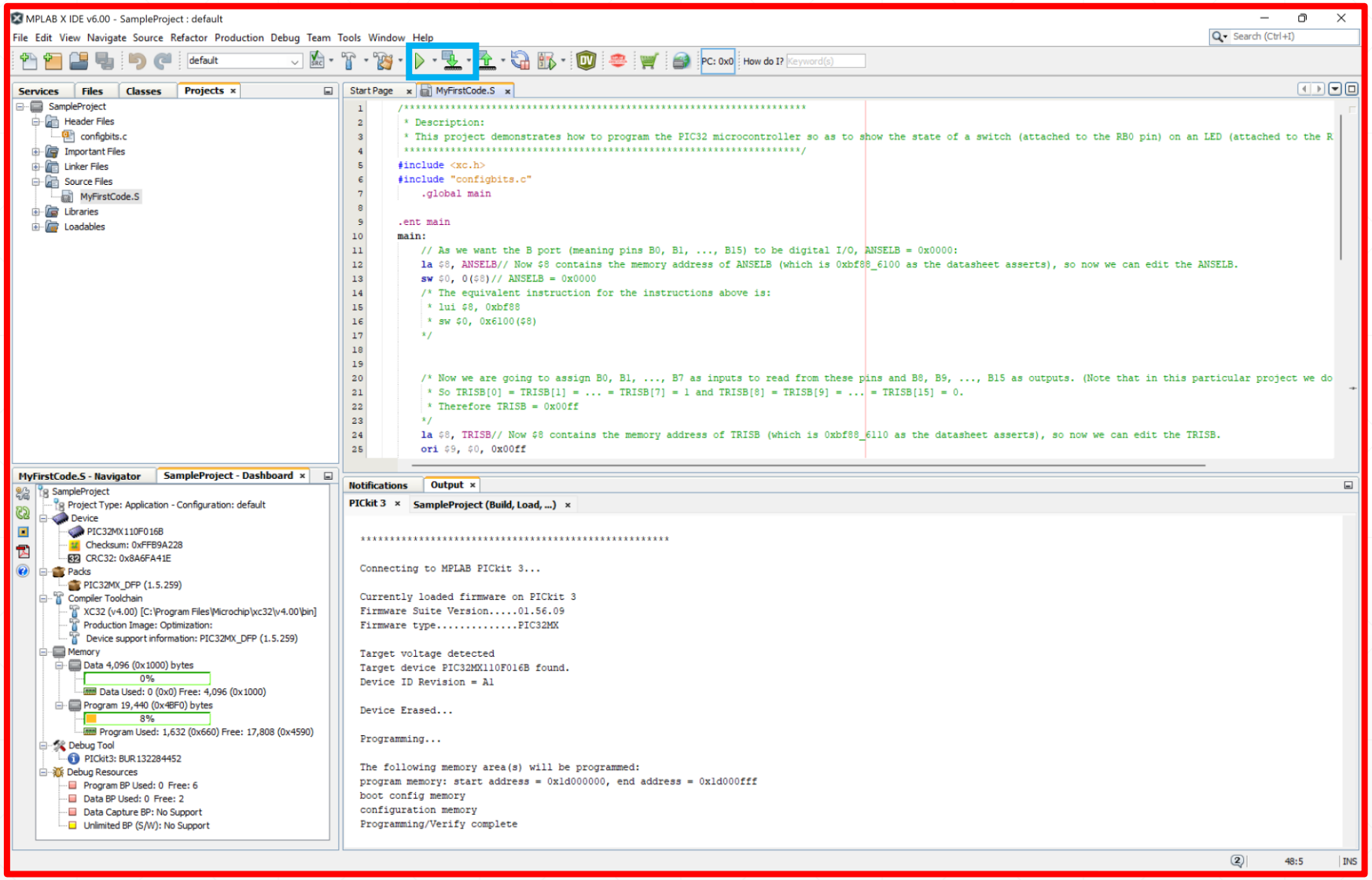
- Now that you have written your first piece of code, you can test it. Click on the “Build” button. (Or clean and build for the next times)

- If your code has no syntax errors, you are going to see something like this in the output window below the MPLAB X IDE window:



Getting Started with MPLAB X IDE – 12 : Final Move!

- After making sure that everything is ok, you're going to program your code into PIC32 micro.
- Plug in the PICkit3 Programmer kit into your computer.
- Click on either "Run Main Project" or "Make and Program Device" button in order to program the device.
- Select the right Programmer Device "PICkit3" (if asked for) and start programming.



What is the clock frequency?

- PIC32 MX110F016B provides different options for the user to have various clock frequencies.
- The microcontroller has two inner oscillators and is also able to gain its clock by an outer one.
- In the configbits.c file, the clock is set from one of the inner oscillators at 10 MHz.

CPU clock = 10 MHz

That’s it! Enjoy.