Microcontroller & MCC
PIC1002 v1.00

A Leading Provider of Smart, Connected and Secure Embedded Control Solutions

CAE Taiwan
Microchip Technology Inc.

July 1, 2022

Major Course (1/2)

- IDE, Compilers, MCC and Development Tools Introduction
- APP1618-SD EVB Introduction
- Microcontroller Introduction
- Program Memory and Data Memory
- Instruction Set Summary
- Program Sections and Linker Classes
- Getting Started with First Project
  - Lab0.0 First Project (ASM)
  - Lab0.1 First Project (C)
  - Lab0 Discuss
- GPIO Architecture
  - Lab1.0 GPIO Output (ASM)
  - Lab2.0 GPIO Input (ASM)

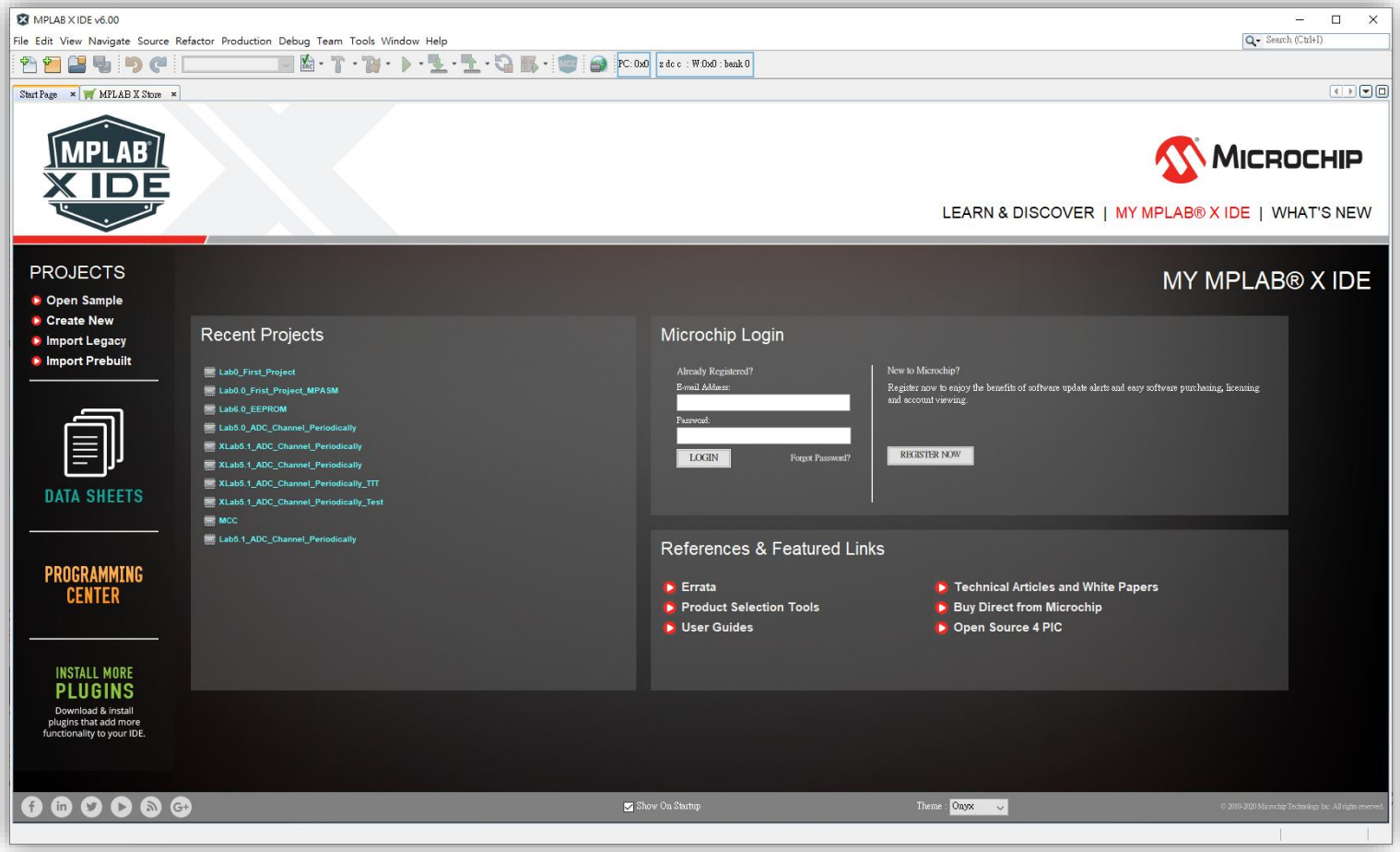
Major Course (2/2)

- Oscillator Architecture
  - 💻Lab3.0 Clock PRIPLL (ASM)
  - 💻Lab3 Discuss
- Interrupt Architecture
- Timer Architecture
  - 💻Lab4.0 Timer2 Interrupt (ASM)
- ADC Architecture
  - 💻Lab5.0 ADC Channel Periodically (ASM)
  - 💻Lab5 Discuss
- EEPROM Architecture
  - 💻Lab6 EEPROM Write and Read (ASM)
  - 💻Lab6 Discuss

IDE, Compiler, MCC & Development Tools Introduction

All in One Development Tools

- **New generation Integrated Development Environment, Support PIC, sic, AVR, CEC and SAM Series MCU/MPU, Provide Plug-in function to extend more advance feature. Java Based, Cross platform, latest version is v6.00.**



C Compiler for 8-bits PIC/dsPIC® MCU

- **XC8 is new generation C compiler, it's supported all 8-bits PIC/AVR MCU (PIC10F/PIC12F /PIC16F /PIC18F, AVR / ATINY).**
- **Base on GNU C, apply GPL License (GNU General Public License).**
- **Provide standard C libraries (printf, strlen, etc..) and Peripheral Libraries (old versions).**
- **All version you can download from Microchip website. It's provided free version.**

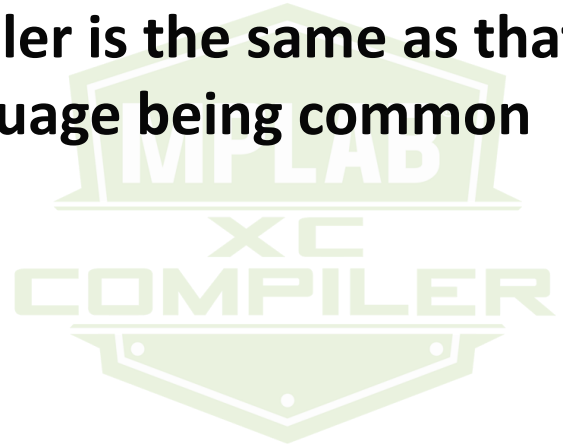
License Type	Installs On	# of Activations	# of Users	Wait Time Between Users	HPA
Workstation License	Workstation	3	1	None	Yes
Subscription License	Workstation	1	1	None	No
Site License	Network	1	Varies by Seat	None	Yes
Network Server License	Network	1	Unlimited	One Hour	Yes
Virtual Machine *	Network	1	N/A	N/A	No
Dongle License	Dongle	N/A	Unlimited	None	No

**This license must be used in addition to a network server or site license to enable the license to work in a virtual machine environment.*

MICROCHIP

MPLAB XC8 PIC Assembler Overview

- The MPASM Assembler is no more built into MPLAB X IDE later than v5.40, It replace to the MPLAB XC8 PIC Assembler, this is bundled with the MPLAB XC8 C compiler.
- The PIC Assembler is a free-standing cross assembler and linker package, supporting all 8-bit PIC® microcontrollers.
- PIC Assembler is not a code compatible replacement for the microchip MPASM, many of the MPASM directives and operators need different coding in the PIC Assembler.
- MPASM can still be used for maintenance of existing projects, but it is no longer being developed and the PIC Assembler is recommended for all new projects.
- The internal assembler application used by the PIC Assembler is the same as that used by the MPLAB XC8 C Compiler tool, with the assembly language being common between both tools.



PIC Assembler Recommended Reading

- **MPLAB® XC8 PIC Assembler User's Guide**
  - This user’s guide describes the use and features of the MPLAB XC8 PIC Assembler.
- **MPLAB® XC8 PIC Assembler Guide For Embedded Engineers**
  - This guide is a getting started guide, describing example projects and commonly used coding sequences used by the MPLAB XC8 PIC assembler. Use this guide if you need to develop new projects using the assembler.
- **MPLAB® XC8 PIC Assembler Migration Guide**
  - This guide is for customers who have MPASM projects and who wish to migrate them to the MPLAB XC8 PIC assembler. It describes the nearest equivalent assembler syntax and directives for MPASM code.

Programmer/Debugger

- **The MPLAB® ICD 4 In-Circuit Debugger/Programmer is Microchip's fastest, cost-effective debugging and programming tool for PIC®, dsPIC®, AVR, SAM and CEC flash microcontrollers.**
- **Features**
  - Full-Speed Real-Time Emulation
  - Target voltage of 1.20V to 5.5V.
  - Can supply up to 1A of power to the target
  - Overvoltage/ Over-current protection
  - Supports multiple breakpoints, stopwatch and source code file debugging
  - Selectable pull-up/pull-down option to the target interface

$328.89

MICROCHIP

Programmer/Debugger

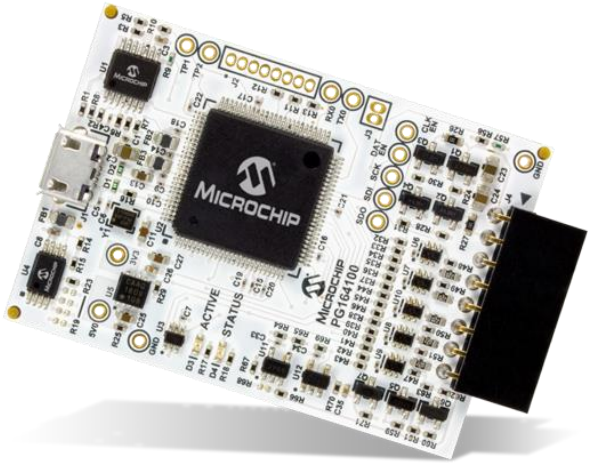
- **The MPLAB® PICkit™ 4 In-Circuit Debugger/Programmer allows fast and easy debugging and programming of all PIC®, dsPIC®, AVR, SAM and CEC flash microcontrollers.**
- **Features**
  - Matches silicon clocking speed.
  - Target voltage of 1.20V to 5.5V.
  - Can supply up to 50mA of power to the target.
  - Minimal current consumption at <100µA from target.
  - Portable USB-powered and RoHS-compliant.
  - 8-pin single in-line header.
  - Backward compatible for demo boards, headers and target systems using 2-wire JTAG and ICSP.
  - Option to be self-powered from the target (2.7V to 5.5V).

$76.99

MICROCHIP

Programmer/Debugger

- **The MPLAB® SNAP In-Circuit Debugger/Programmer allows fast and easy debugging and programming of most PIC®, dsPIC®, AVR, SAM and CEC flash microcontrollers.**
- **Features**
  - Affordable performance
  - Matches silicon clocking speed
  - Target voltage of 1.20V to 5.5V
  - Portable USB-powered
  - CE and RoHS-compliant
  - 8-pin single in-line header
  - Backward compatible for demo boards, headers and target systems using 2-wire JTAG and ICSP



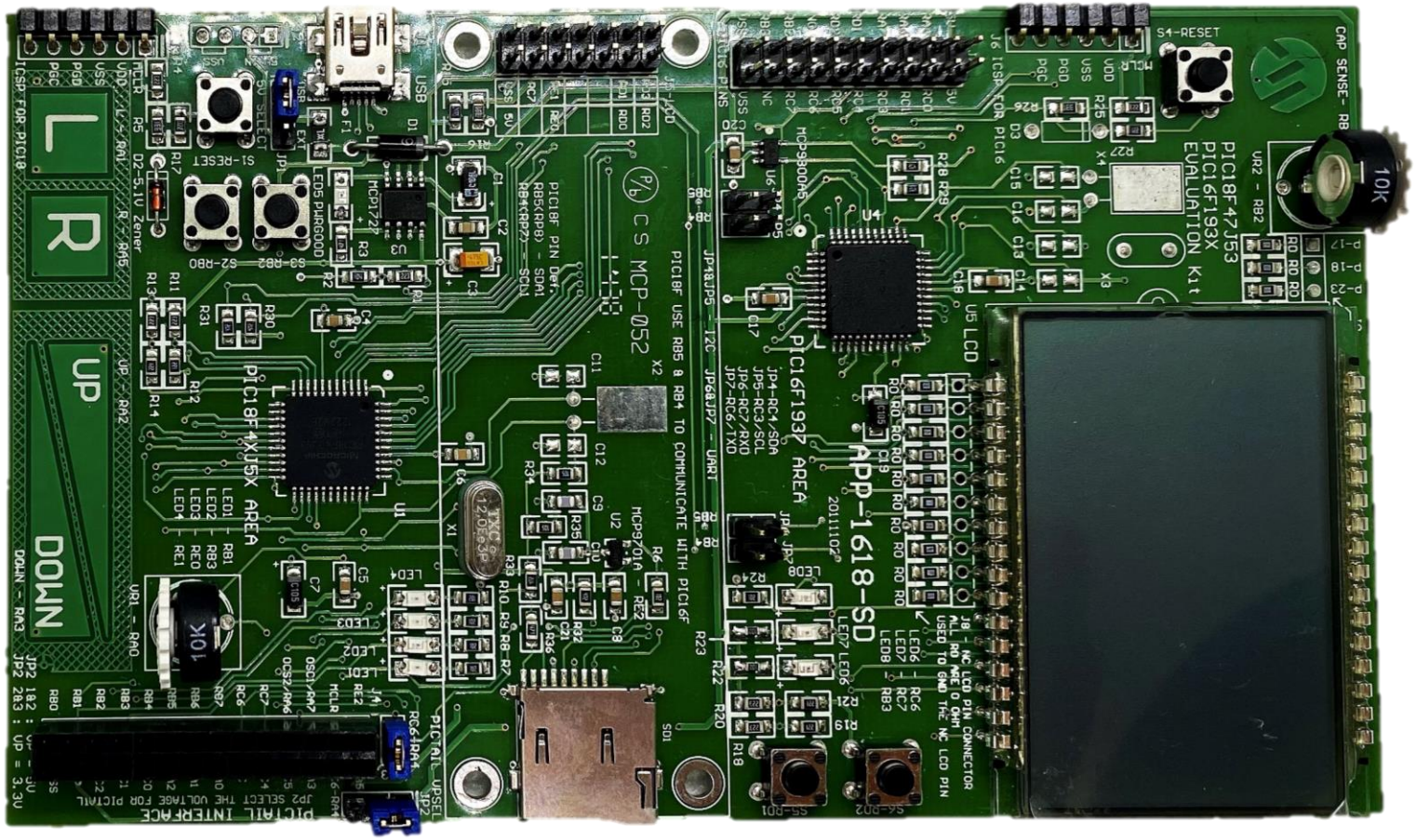
$34.09

MICROCHIP

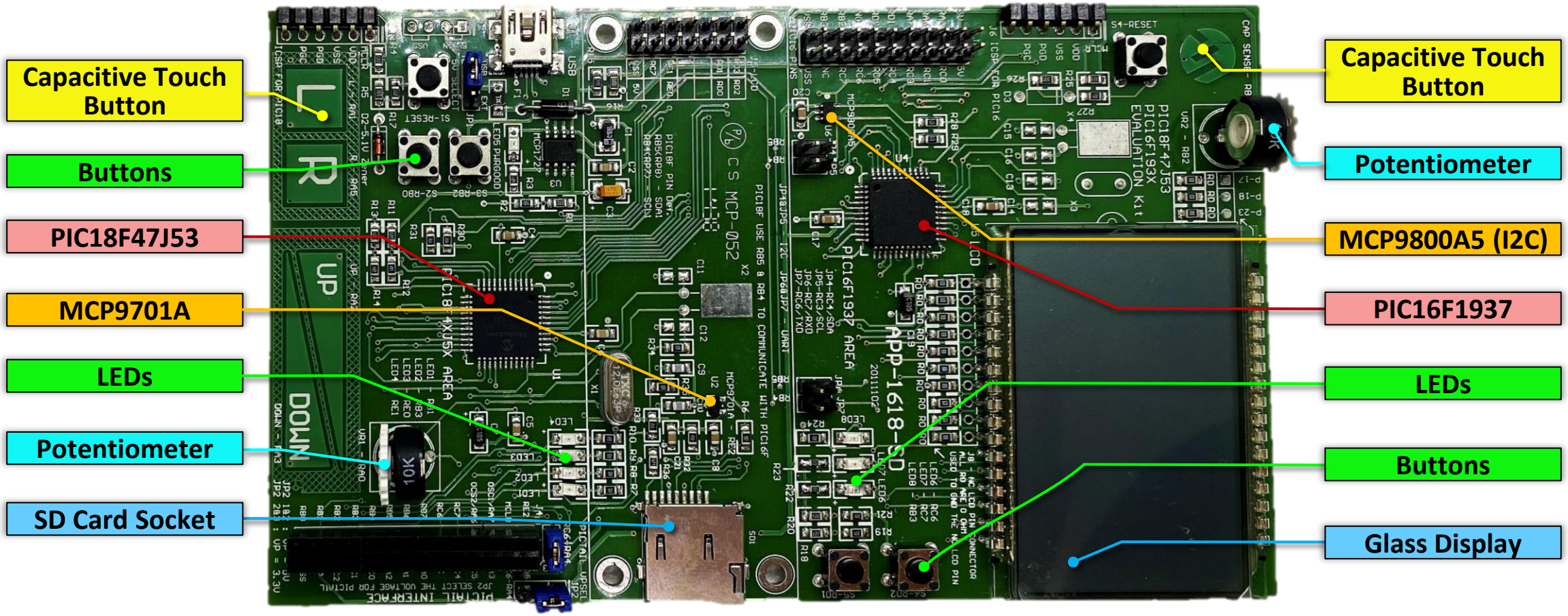
APP1618-SD EVB Introduction

APP1618-SD EVB

- **APP1618-SD EVB combine two Microchip MCUs and Built-on rich components.**
- **PIC16F1937 block**
  - 3 x LEDs, 2 x Buttons.
  - 1 x Potentiometer.
  - 1 x I2C Temperature Sensor.
  - 1 x Capacitive Touch Buttons.
  - 1 x Glass Display.
- **PIC18F47J53 block**
  - 4 x LEDs.
  - 3 x Buttons.
  - 1 x Potentiometer.
  - 1 x Analog Temperature Sensor .
  - 4 x Capacitive Touch Buttons.
  - 1 x MicroSD Socket

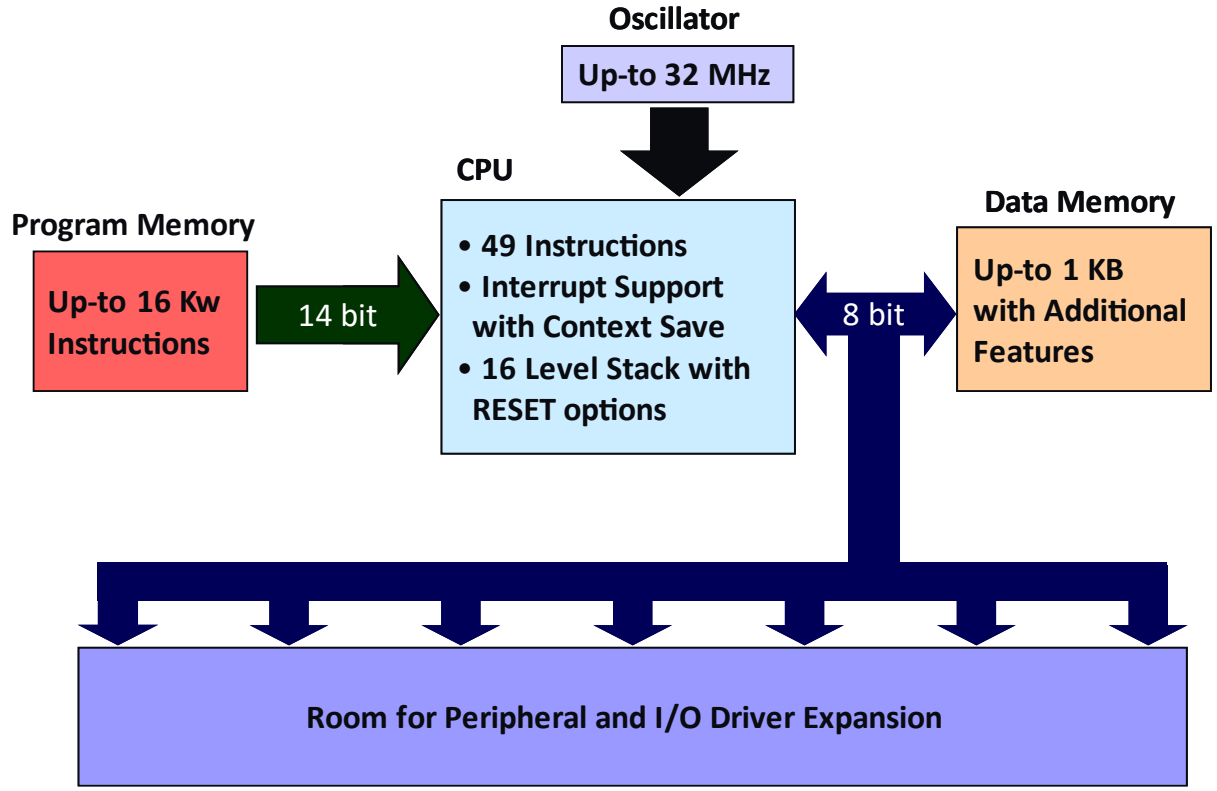


APP1618-SD Overview



PIC16(L)F193X Introduction

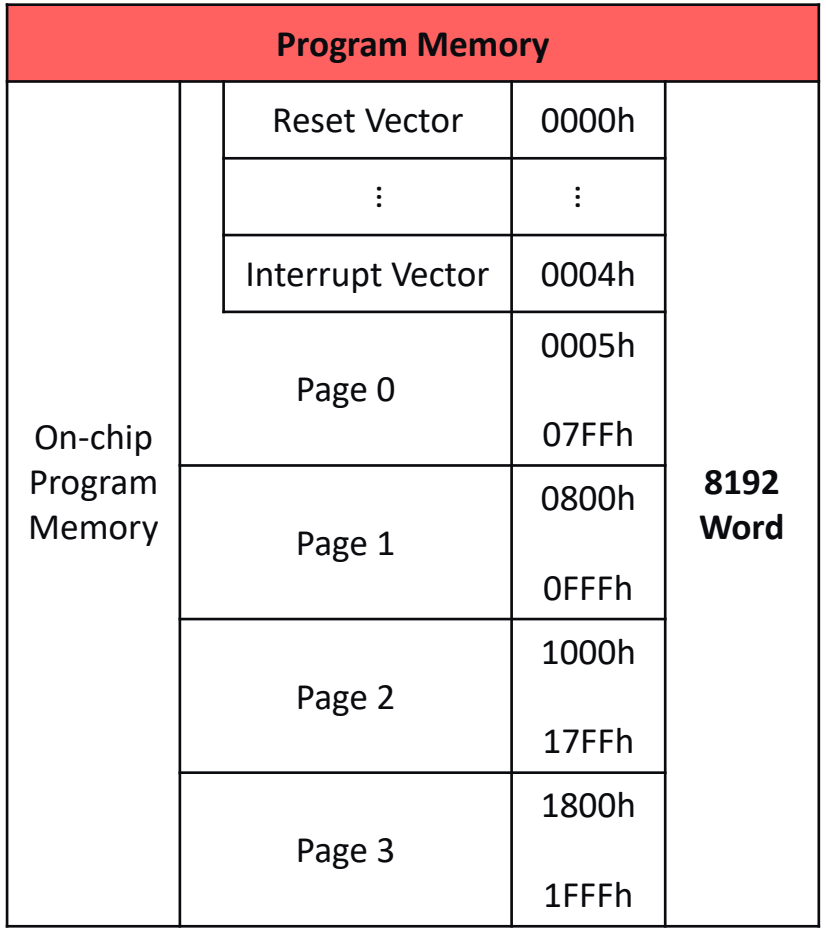
- **This family of devices contain an enhanced mid-range 8-bit CPU core.**
- **High-Performance RISC CPU:**
  - Only 49 Instructions
  - Up to 32 MHz oscillator/clock input
  - Up to 16K x 14 Words of Flash Program Memory
  - Up to 1024 Bytes of Data Memory (RAM)
  - 16-Level Deep Hardware Stack
  - Interrupt Capability with Automatic Context Saving
- **Special Microcontroller Features:**
  - **Precision Internal Oscillator:**
    - Factory calibrated to ±1%, typical
    - Software selectable frequency range from 32 MHz to 31 kHz
    - Power-Saving Sleep mode
  - **High Endurance Flash/EEPROM cell**
    - 100,000 write Flash endurance
    - 1,000,000 write EEPROM endurance
    - Flash/Data EEPROM retention: > 40 years
  - **Wide Operating Voltage Range:**
    - 1.8V-5.5V (PIC16F193X)
    - 1.8V-3.6V (PIC16LF193X)

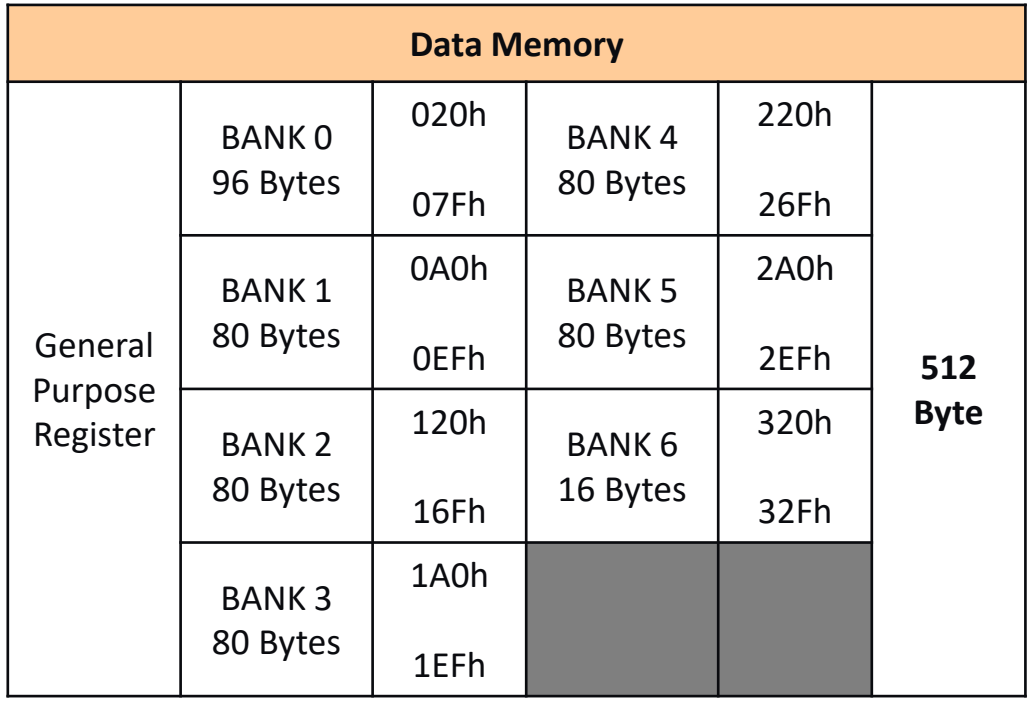


MICROCHIP

PIC16(L)F1937 Program Memory and Data Memory

- 8K x 14 Words of Flash Program Memory
- 512 Bytes of Data Memory (RAM)



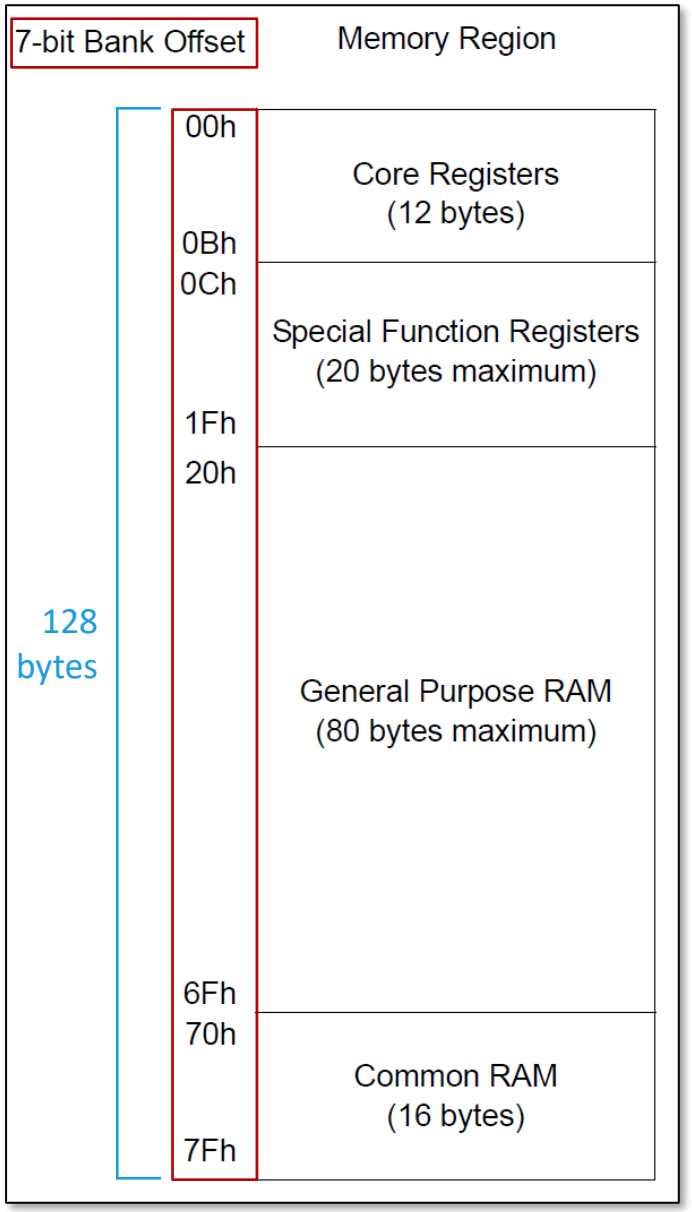


MICROCHIP

PIC16F Series
Program Memory and Data Memory

Data Memory Organization

- **The data memory is partitioned in 32 memory banks with 128 bytes in a bank.**
- **Each bank consists of:**
  - 12 core registers
  - 20 Special Function Registers (SFR)
  - Up to 80 bytes of General Purpose RAM (GPR)
  - 16 bytes of common RAM
- **The active bank is selected by writing the bank number into the Bank Select Register (BSR).**
- **All data memory can be accessed either directly using the file registers.**



MICROCHIP

What's Banks?

- The file register address of the instruction is 7bit.
- Writing the bank number into the Bank Select Register (BSR) to change the actual file register address.
- Change the value of the BSR via **BANKSEL** or **MOVLB**.

Byte-oriented file register operations

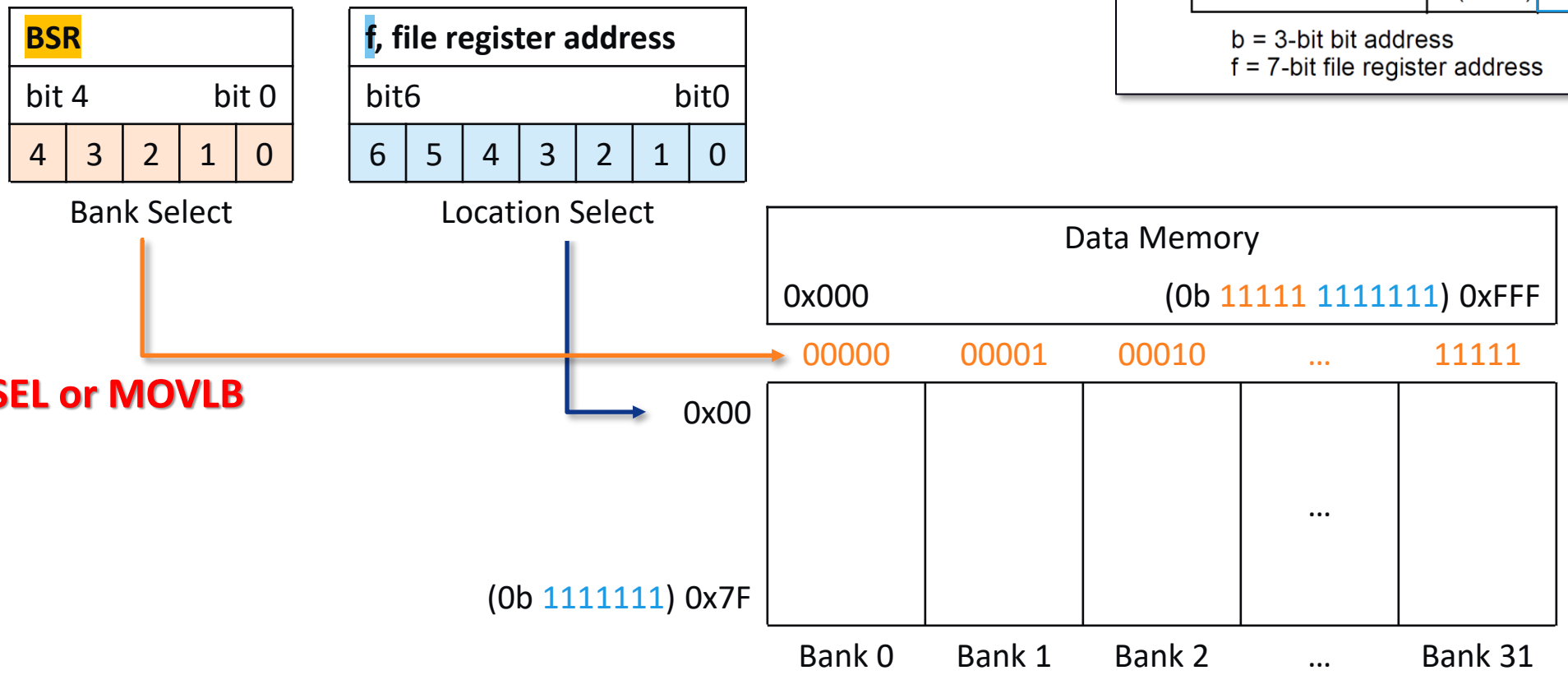
13 … 8	7	6 … 0
OPCODE	d	f (FILE #)

d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Bit-oriented file register operations

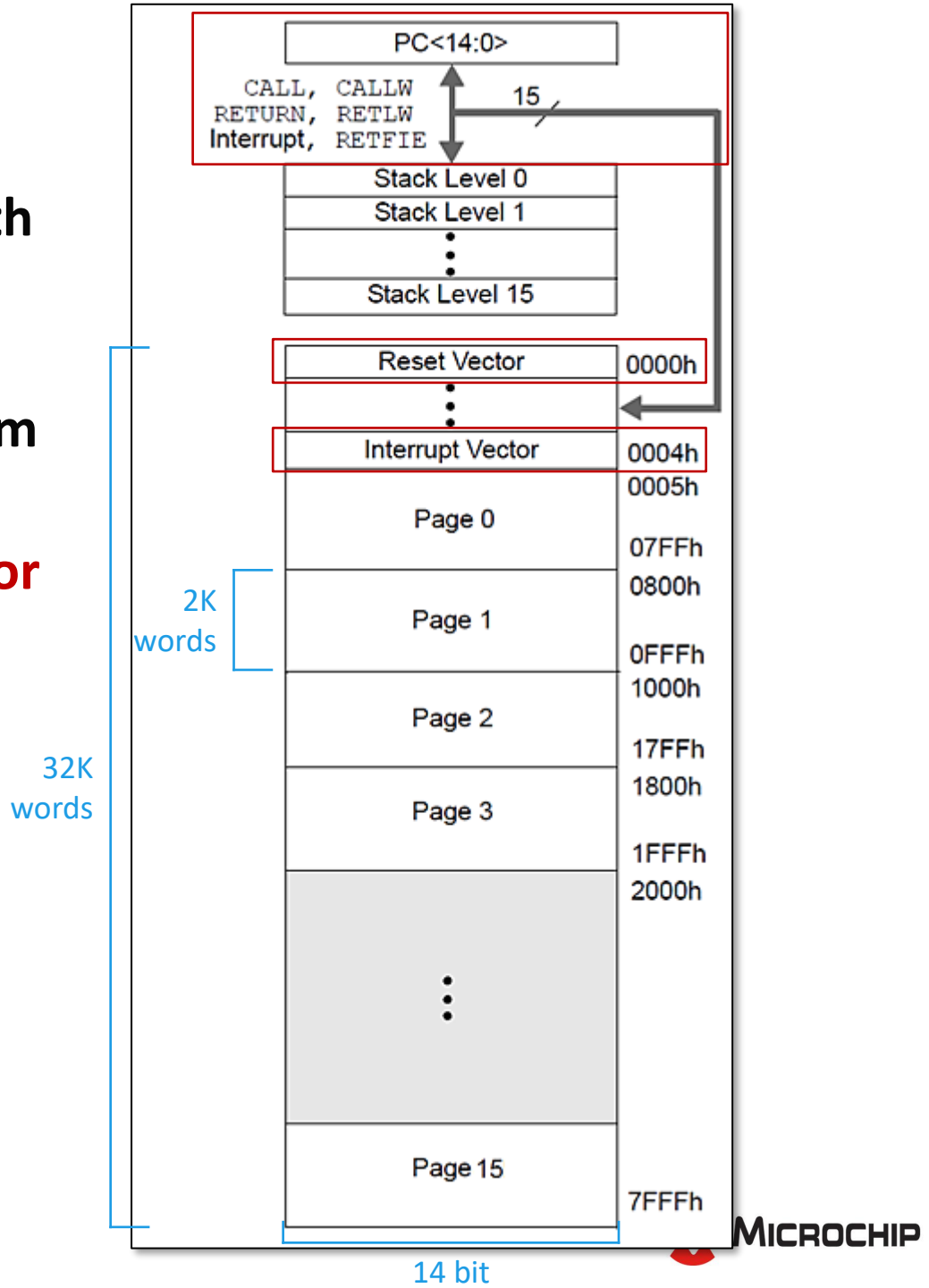
13 … 10	9 … 7	6 … 0
OPCODE	b (BIT #)	f (FILE #)

b = 3-bit bit address
f = 7-bit file register address



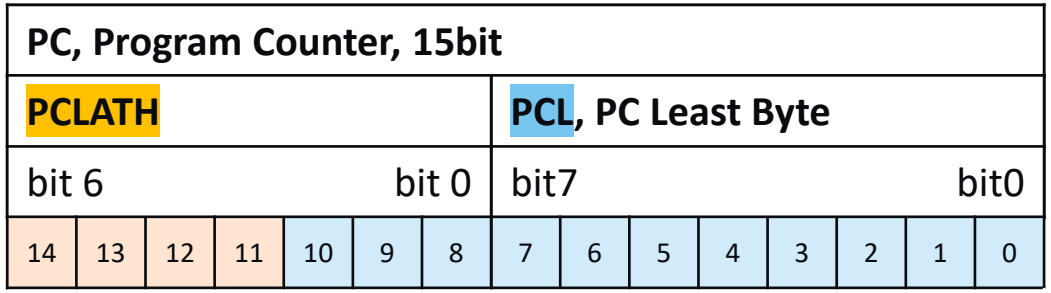
Program Memory Organization

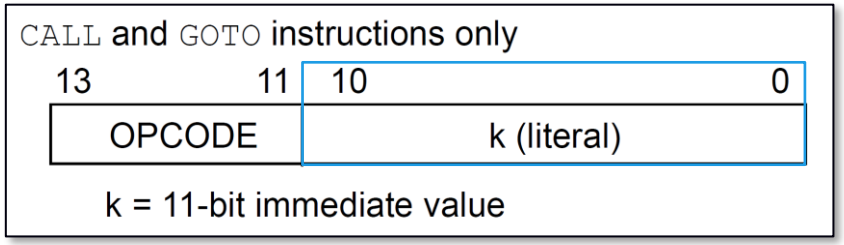
- The Program memory is partitioned in 16 Pages with 2K x 14bits in a Page.
- The enhanced mid-range core has a 15-bit program counter (PC) capable of addressing 32K x 14 program memory space.
- The Reset vector is at 0000h and the Interrupt vector is at 0004h.

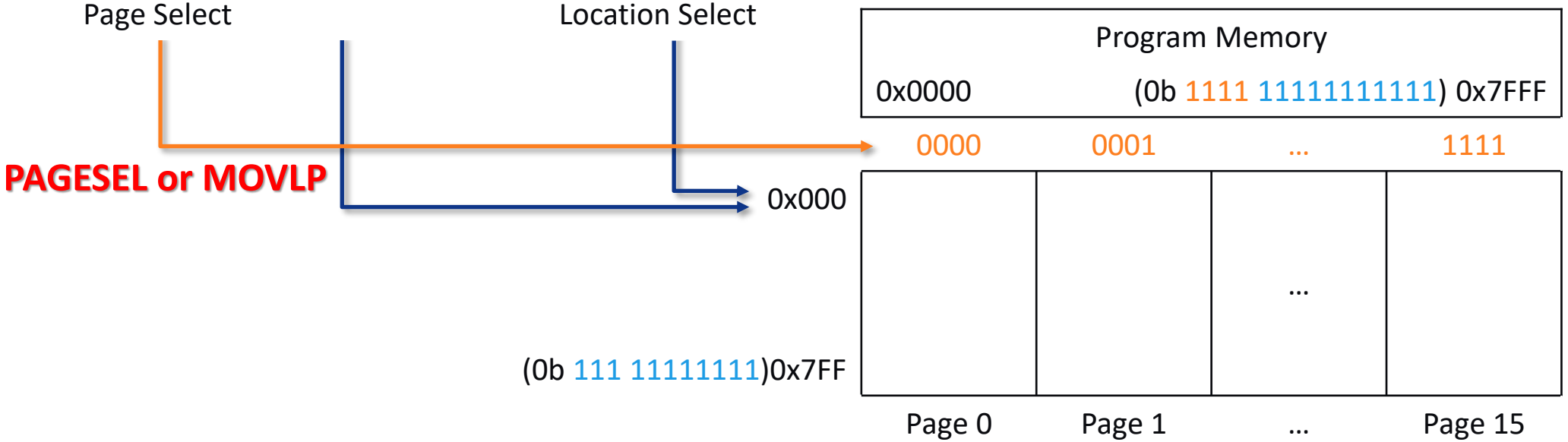


What's Page

- The Program Counter (PC) is 15 bits wide, it can access up to 2^15=32K words (≤0x7FFF).
- The address of the instruction is 11bit.
- The low byte comes from the PCL register, which is a readable and writable register.
- The high byte (PC<14:8>) is not directly readable or writable and comes from PCLATH.
- Change the value of the PCLATH via PAGESEL or MOVLP.







PIC16F Series
Instruction Set Summary

PIC16F Series Instruction Set Summary

- **Each PIC16 instruction is a 14-bit word containing the operation code and all required operands.**
- **The opcodes are broken into three broad categories.**
  - Byte Oriented
  - Bit Oriented
  - Literal and Control
- **All instructions are executed within a single instruction cycle, with the following exceptions, which may take two or three cycles:**
  - Subroutine takes two cycles (CALL, …)
  - Returns from interrupts or subroutines take two cycles (RETURN, …)
  - Program branching takes two cycles (GOTO, …)
  - One additional instruction cycle will be used when any instruction references an indirect file register and the file select register is pointing to program memory.

Opcode Field descriptions and Abbreviation descriptions

- **The following table lists some of the Opcode Field descriptions and Abbreviation descriptions.**

Field	Description	
W	Working register (accumulator)	
f	File register address (0x00 to 0x7F)	
b	Bit address within an 8-bit file register	
k	Literal field, constant data or label	
d	Destination select; store result in W or f	
C	Carry/nBorrow bit	STATUS Register
Z	Zero bit	

Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WREG	8bit Working Register (accumulator)							

Byte-oriented file register operations

13 ... 8	7	6 ... 0
OPCODE	d	f (FILE #)

d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Bit-oriented file register operations

13 ... 10	9 ... 7	6 ... 0
OPCODE	b (BIT #)	f (FILE #)

b = 3-bit bit address
f = 7-bit file register address

Literal and control operations

General

13 ... 8	7 ... 0
OPCODE	k (literal)

k = 8-bit immediate value

Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STATUS	-	-	-	nTO	nPD	Z	DC	C

PIC16F Series Instruction Set Summary

- **The following table lists some of the BYTE-Oriented File Register and LITERAL Operations.**

Mnemonic, Operands		Description	Cycles	Status Affected
BYTE-Oriented File Register and LITERAL Operations				
MOVF	f, d	Move f	1	Z
MOVLW	k	Move literal to W	1	
MOVWF	f	Move W to f	1	
CLRF	f	Clear f	1	Z
INCF	f, d	Increment f	1	Z
DECF	f, d	Decrement f	1	Z
ADDWF	f, d	Add W and f	1	C, Z
SUBWF	f, d	Subtract W from f	1	C, Z
MOVLB	k	Move literal to BSR	1	
MOVLP	k	Move literal to PCLATH	1	

Field	Description
W	Working register (accumulator)
f	Register file address (0x00 to 0x7F)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
d	Destination select; store result in W or f
C	Carry/nBorrow bit (STATUS Register)
Z	Zero bit (STATUS Register)

PIC16F Series Instruction Set Summary

- **Example of the BYTE-Oriented File Register and LITERAL Operations.**

Field	Description
W	Working register (accumulator)
f	Register file address (0x00 to 0x7F)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
d	Destination select; store result in W or f
C	Carry/nBorrow bit (STATUS Register)
Z	Zero bit (STATUS Register)

			Example:	Result:
MOVF	f, d	Z	movf X, w	WREG = X, (if WREG = 0, Z = 1)
			movf X, f	X = X, (if X = 0, Z = 1)
MOVLW	k		movlw 10	WREG = 10
MOVWF	f		movwf X	X = WREG
CLRF	f	Z	clrf X	X = 0, (if X = 0, Z = 1)
INCF	f, d	Z	incf X, w	WREG = X + 1, (if WREG = 0, Z = 1)
			incf X, f	X = X + 1, (if X = 0, Z = 1)
DECF	f, d	Z	decf X, w	WREG = X - 1, (if WREG = 0, Z = 1)
			decf X, f	X = X - 1, (if X = 0, Z = 1)
ADDWF	f, d	C, Z	addwf X, w	WREG = X + WREG, (if WREG = 0, Z = 1 ; if Carry, C = 1)
			addwf X, f	X = X + WREG, (if X = 0, Z = 1 ; if Carry, C = 1)
SUBWF	f, d	C, Z	subwf X, w	WREG = X - WREG, (if WREG = 0, Z = 1 ; if Borrow, C = 0)
			subwf X, f	X = X - WREG, (if X = 0, Z = 1 ; if Borrow, C = 0)
MOVLB	k		movlb 3	BSR(Bank) = 3
MOVLP	k		movlp 3	PCLATH(Page) = 3

PIC16F Series Instruction Set Summary

- **The following table lists some of the BIT-Oriented File Register Operations.**

Mnemonic, Operands		Description	Cycles	Status Affected
BIT-Oriented File Register Operations				
BCF	f, b	Bit Clear f	1	
BSF	f, b	Bit Set f	1	
BIT-Oriented Skip Operations				
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	

Field	Description
W	Working register (accumulator)
f	Register file address (0x00 to 0x7F)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
d	Destination select; store result in W or f
C	Carry/nBorrow bit (STATUS Register)
Z	Zero bit (STATUS Register)

PIC16F Series Instruction Set Summary

Field	Description
W	Working register (accumulator)
f	Register file address (0x00 to 0x7F)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
d	Destination select; store result in W or f
C	Carry/nBorrow bit (STATUS Register)
Z	Zero bit (STATUS Register)

- **Example of the BIT-Oriented File Register Operations.**

		Example:	Result:
BCF	f, b	**bcf X, 3**	Set X bit 3 = 0
BSF	f, b	**bsf X, 3**	Set X bit 3 = 1
BTFSC	f, b	**btfsc X, 5**	if X bit 5 = 0, skip 1 command ; else, run next command
BTFSS	f, b	**btfss X, 5**	if X bit 5 = 1, skip 1 command ; else, run next command

PIC16F Series Instruction Set Summary

- **The following table lists some of the Control Operations.**

Mnemonic, Operands		Description	Cycles	Status Affected
Control Operations				
GOTO	k	Go to address	2	
CALL	k	Call Subroutine	2	
RETURN		Return from Subroutine	2	
RETFIE		Return from interrupt	2	

Field	Description
W	Working register (accumulator)
f	Register file address (0x00 to 0x7F)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
d	Destination select; store result in W or f
C	Carry/nBorrow bit (STATUS Register)
Z	Zero bit (STATUS Register)

PIC16F Series Instruction Set Summary

Field	Description
W	Working register (accumulator)
f	Register file address (0x00 to 0x7F)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
d	Destination select; store result in W or f
C	Carry/nBorrow bit (STATUS Register)
Z	Zero bit (STATUS Register)

- **Example of the Control Operations.**

		Example:	Result:
GOTO	k	**goto Label1**	Goto the **Label1:** address
		goto $+2	Goto the (current address + 2)
CALL	k	**call Function1**	Goto the **Function1:** , until the **RETURN** instruction is executed
RETURN		**return**	Return to the next address before executing **CALL**
RETFIE		**retfie**	Return to the next address before executing **Interrupt**

MICROCHIP

Preprocessor Directives

- **MPLAB XC8 accepts several specialized preprocessor directives, in addition to the standard directives.**
- **The following table lists some of the Preprocessor Directives.**

Directive	Purpose
Preprocessor Directives	
#include	Include text file into source.
#define	Define preprocessor macro.

Assembler Directives

- **Assembler directives, or pseudo-ops, are used in a similar way to instruction mnemonics, with the exception of PAGESEL and BANKSEL, these directives do not generate instructions.**
- **The following table lists some of the Assembler Directives.**

Directive	Purpose
Assembler Directives	
PSECT	Declares or resumes program section.
ORG	Sets location counter within current psect.
CONFIG	Specifies configuration bits.
END	Ends assembly
GLOBAL	Makes symbols accessible to other modules.
DS	Reserves storage.
BANKSEL	Generates code to select bank of operand.
PAGESEL	Generates set/clear instruction to set PCLATH bits for this page.

PIC16F Series
Program Sections and Linker Classes

Linker Classes

- **The linker uses classes to represent memory ranges in which psects can be linked.**
- **The assembler driver passes a default set of such options to the linker, based on the selected target device.**
- **Psects are typically allocated free memory from the class they are associated with.**

- **The following linker classes are defined once you include <xc.inc>.**

Program Memory Classes	
CODE	Consists of ranges that map to the program memory pages on the target device and are used for psects containing executable code.
STRCODE	Defines a single memory range that covers the entire program memory.

Data Memory Classes	
RAM	Consists of ranges that cover all the general purpose RAM memory of the target device.
BANK*n*	(where *n* is a bank number) — each consist of a single range that covers the general purpose RAM in that bank.
COMMON	Consists of a single memory range that covers the common RAM.

Program Sections

- Program sections, or psects, are simply a section of code or data.
- They are a way of grouping together parts of a program (via the psect's name) even though the source code cannot be physically adjacent in the source file, or even where spread over several modules.
- The PSECT assembler directive is used to define a psect, it identified by a name and has several attributes.
- If you create your own psects, try to associate them with an existing linker class.
- As you have no control over where this psect is linked, it recommended that a PSECT directive always be placed before the code and objects.
- When writing assembly code, you can use the psects provided once <xc.inc> has been included. These have a similar name and function to the sections used by the 8-bit MPASM assembler.

Psect name	Linker class	Purpose
code	CODE	To hold executable code.
data	STRCODE	To hold data in program memory.
udata	RAM	To hold objects allocatable anywhere in GPR.
udata_bank*n*	BANK*n*	To hold object allocatable in a particular data memory bank.
udata_shr	COMMON	To hold objects allocatable in common memory.

A Basic Example For Mid-range Devices

Psect name	Linker class
code	CODE
data	STRCODE
udata	RAM
udata_bank*n*	BANK*n*
udata_shr	COMMON

```
#include <xc.inc>

PSECT udata_bank0
uData1:
   DS 1
uData2:
   DS 1

PSECT resetVec,class=CODE,delta=2,abs
   ORG 0x00
resetVec:
   PAGESEL main
   goto main

PSECT code
main:

loop:

   goto loop

   END
```

Link Address by Assembler

Data Memory

General Purpose Register	BANK 0 96 Bytes	020h 07Fh	BANK 4 80 Bytes	220h 26Fh	512 Byte
	BANK 1 80 Bytes	0A0h 0EFh	BANK 5 80 Bytes	2A0h 2EFh	
	BANK 2 80 Bytes	120h 16Fh	BANK 6 16 Bytes	320h 32Fh	
	BANK 3 80 Bytes	1A0h 1EFh			

Program Memory

On-chip Program Memory	Reset Vector	0000h	8192 Word
	⋮	⋮	
	Interrupt Vector	0004h	
	Page 0	0005h 07FFh	
	Page 1	0800h 0FFFh	
	Page 2	1000h 17FFh	
	Page 3	1800h 1FFFh	

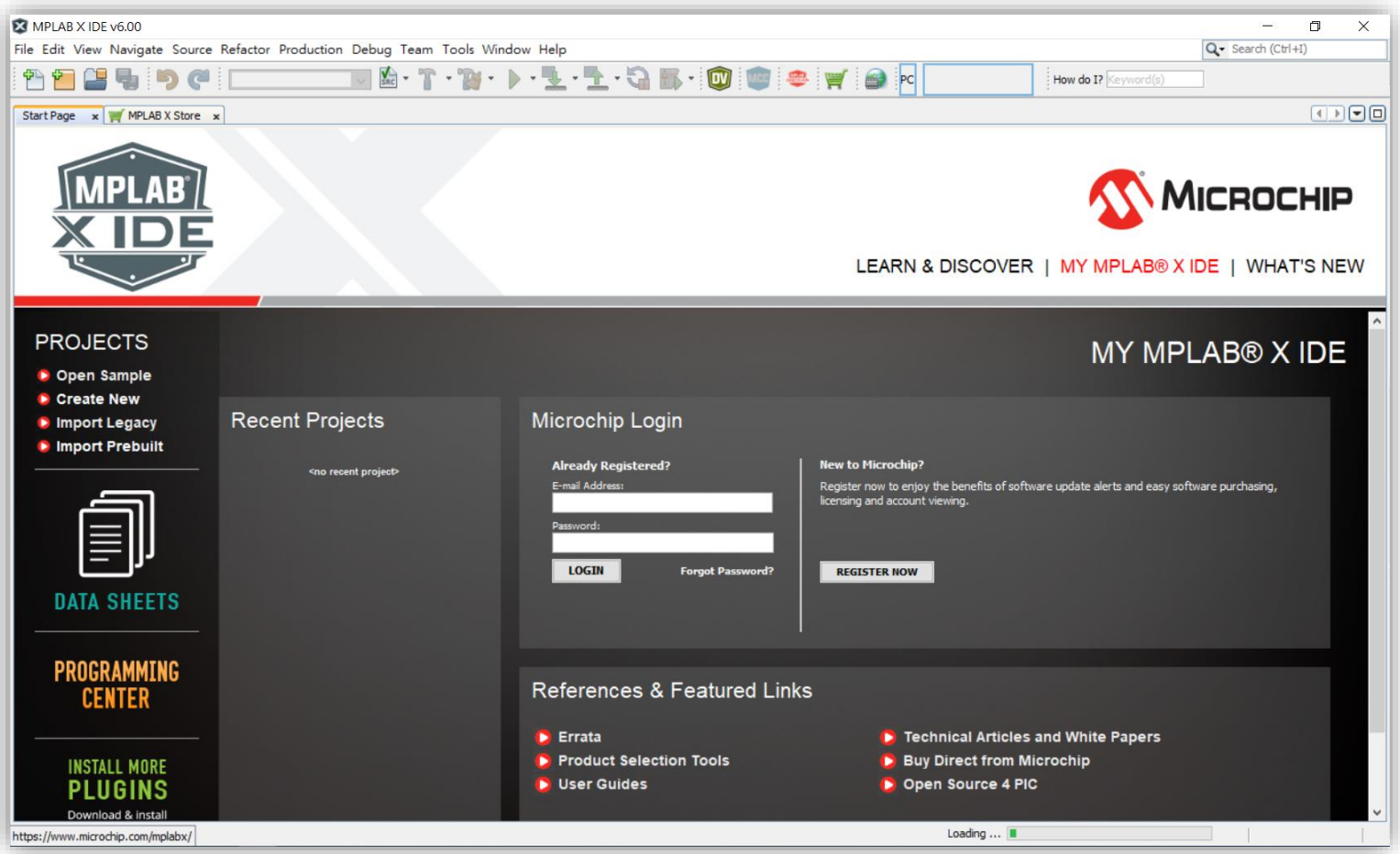
MICROCHIP

First Project

Lab Preparation

Step 1

- Open **MPLAB X IDE** firstly.

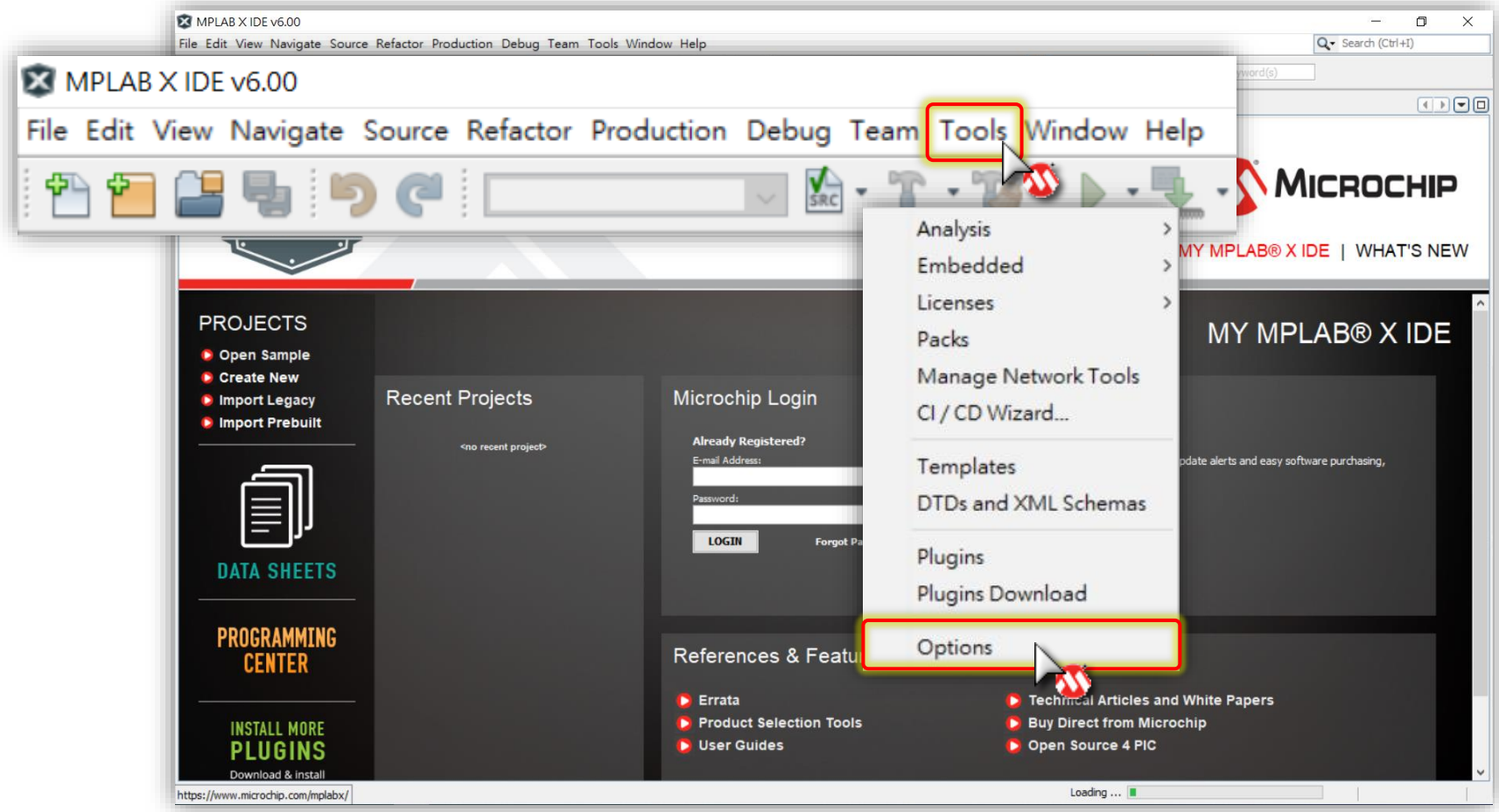


MICROCHIP

Lab Preparation

Step 2

- **Select : Tools ▸ Options**

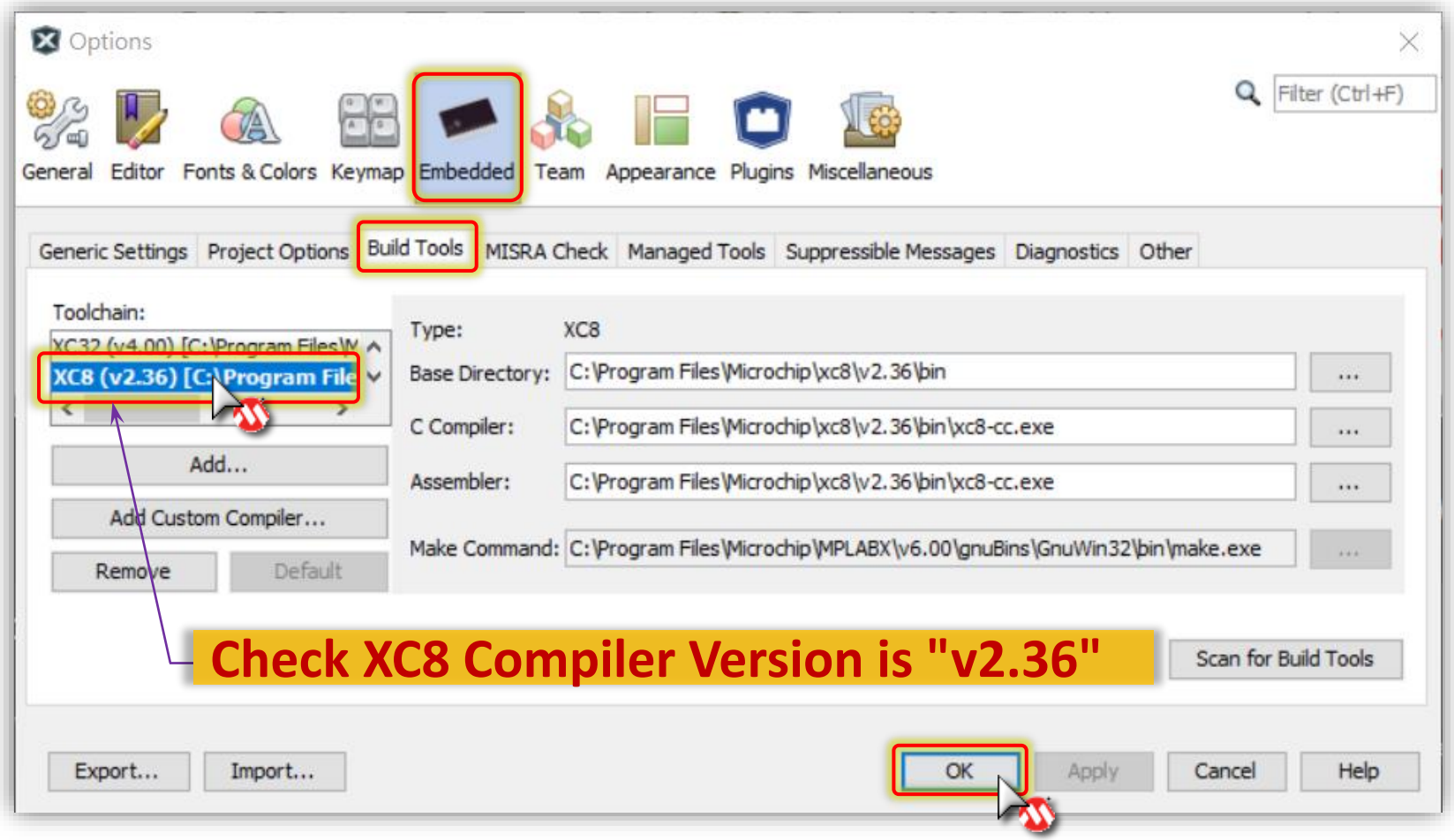


MICROCHIP

Lab Preparation

Step 3

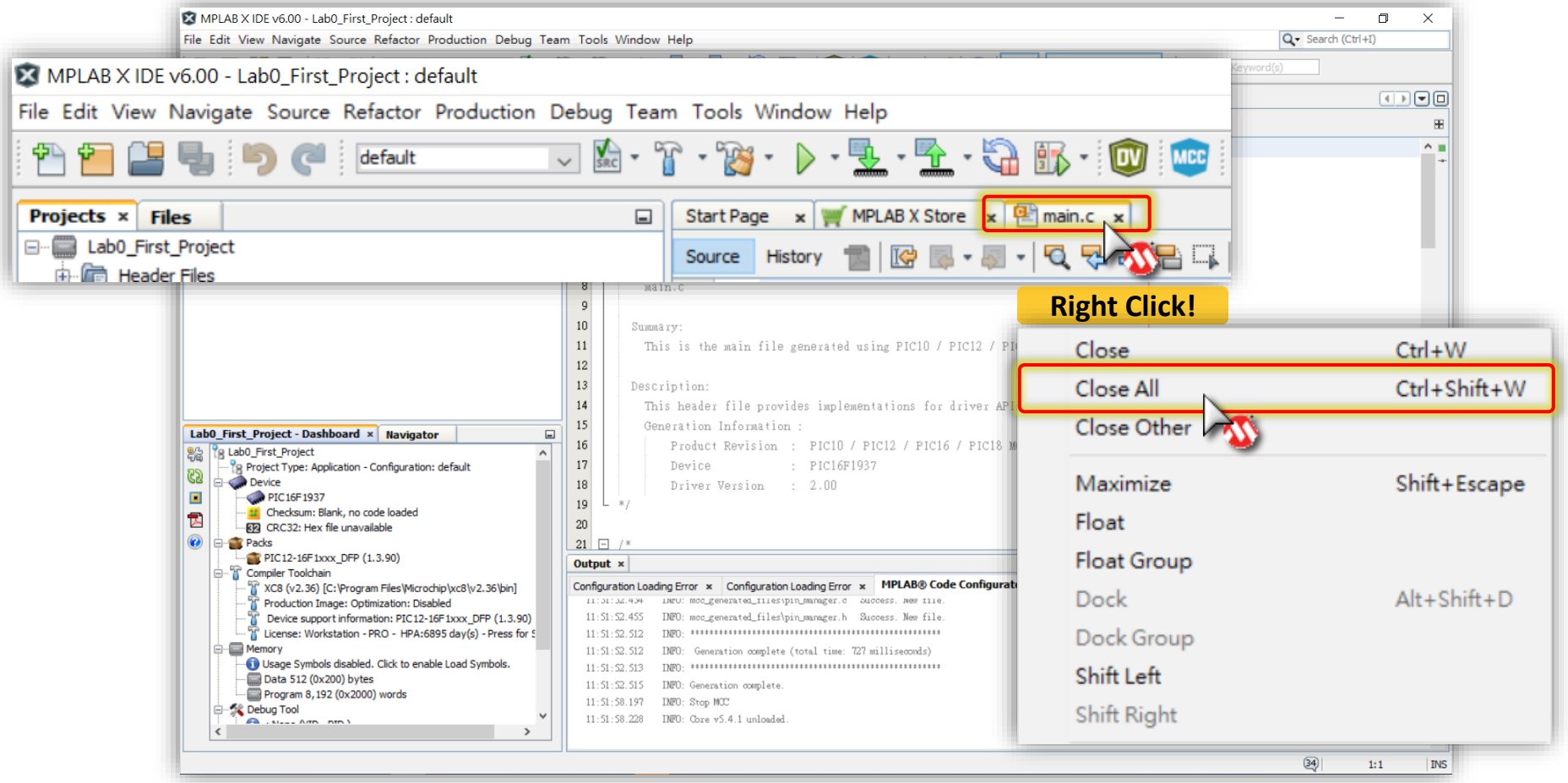
- **Select : Embedded ▸ Build Tools**



Lab Preparation

Step 4

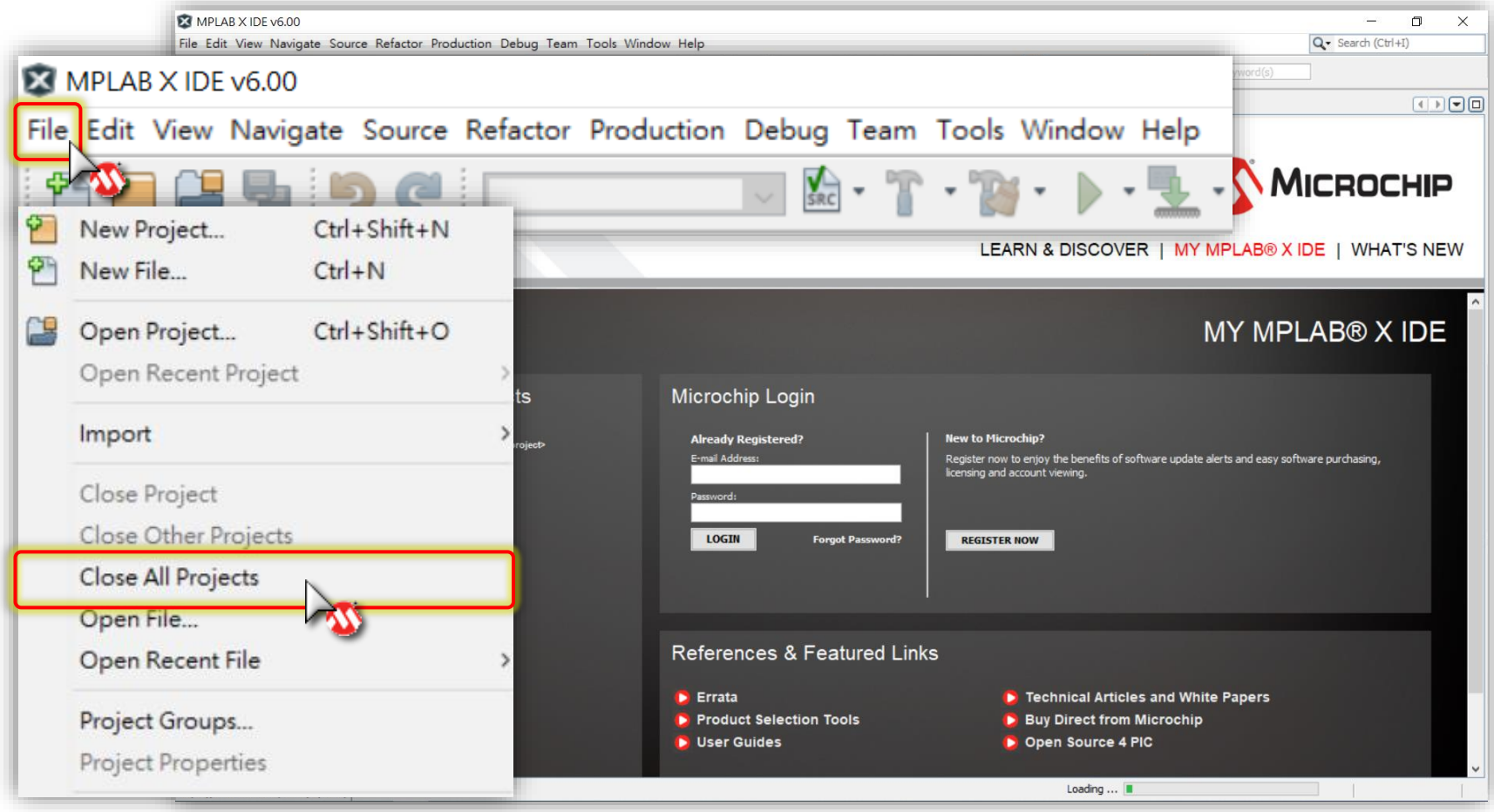
- To prevent opened files to confuse our implementation.
- Please [Right Click] on opened file and choose Close All.



Lab Preparation

Step 5

- To prevent opened projects to confuse our main project.
- Please select File ▸ Close All Projects



MICROCHIP

Lab0 First Project

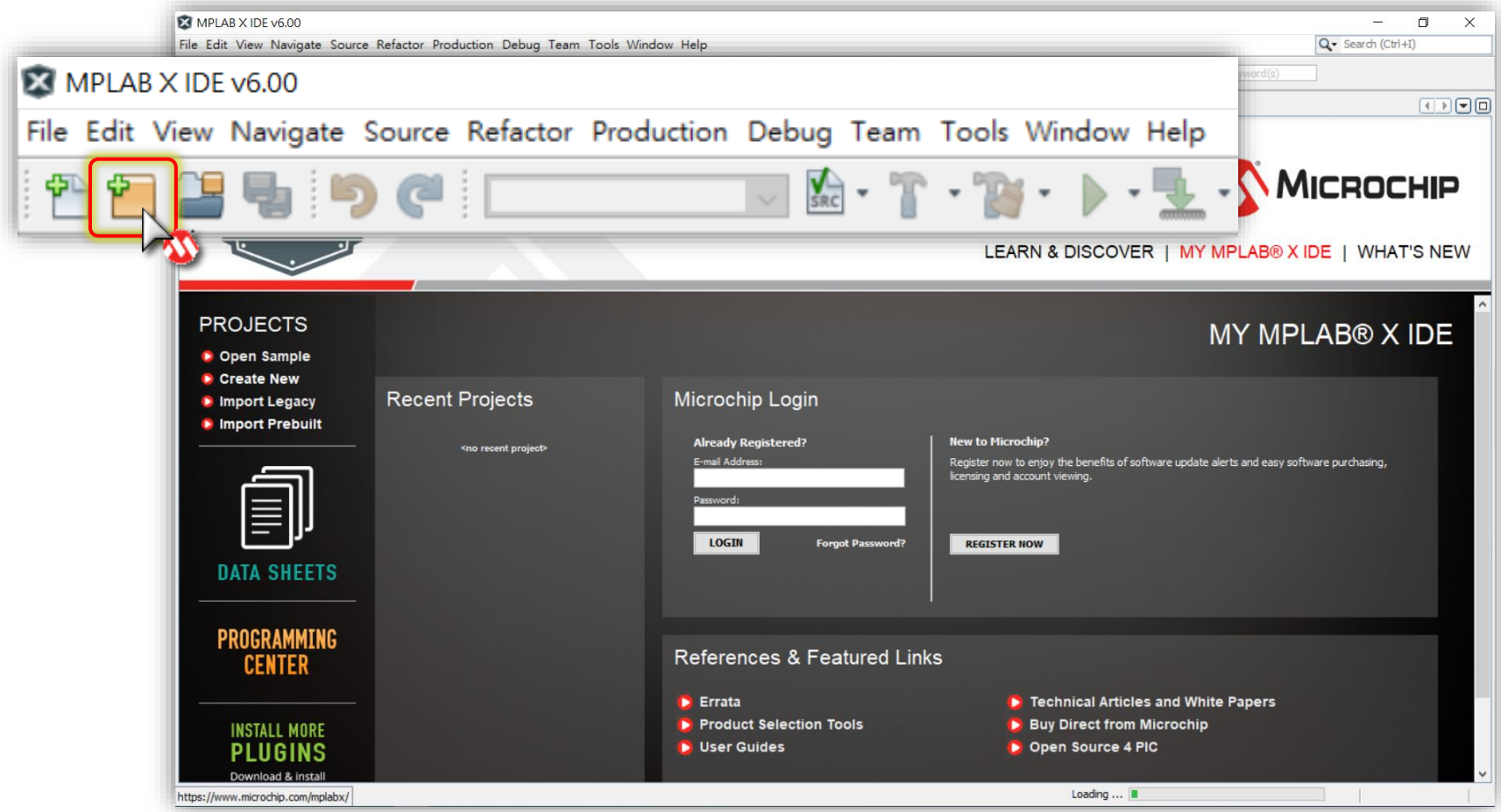
- Base on current project or Open .\Lab0.0_xxx.X to do exercises
- Try to use XC8 PIC Assembler to create your First Project.
- To understanding MPLAB X IDE basic operation.
- Learn how to use edit, build, project manager and program functions.

How to start ?

Lab0 First Project

Step 1

- **Execute Project Wizard Select : File ▸ New Project or Click icon** 

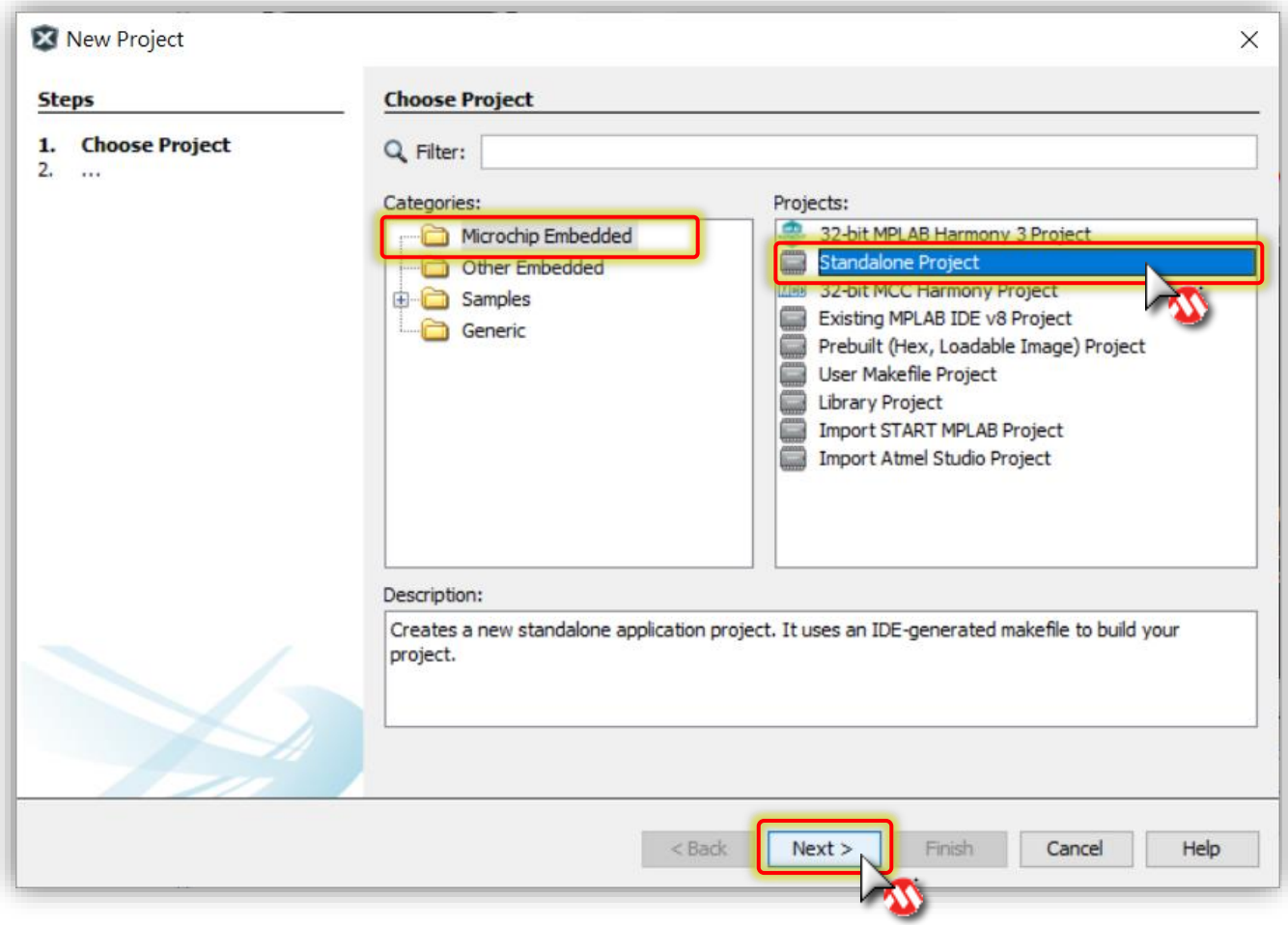


MICROCHIP

Lab0 First Project

Step 2

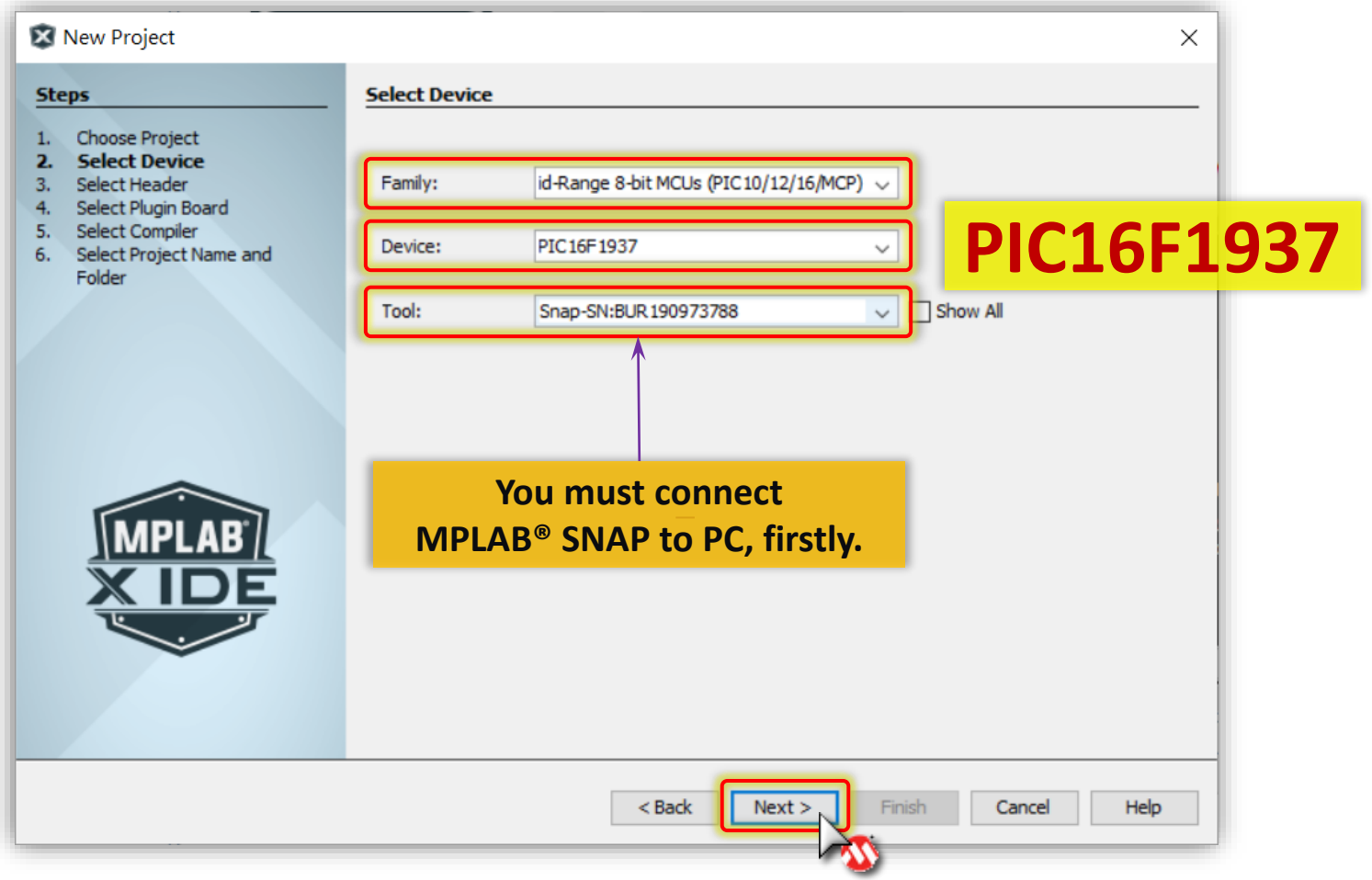
- Categories : Microchip Embedded
- Projects : Standalone Project



Lab0 First Project

Step 3

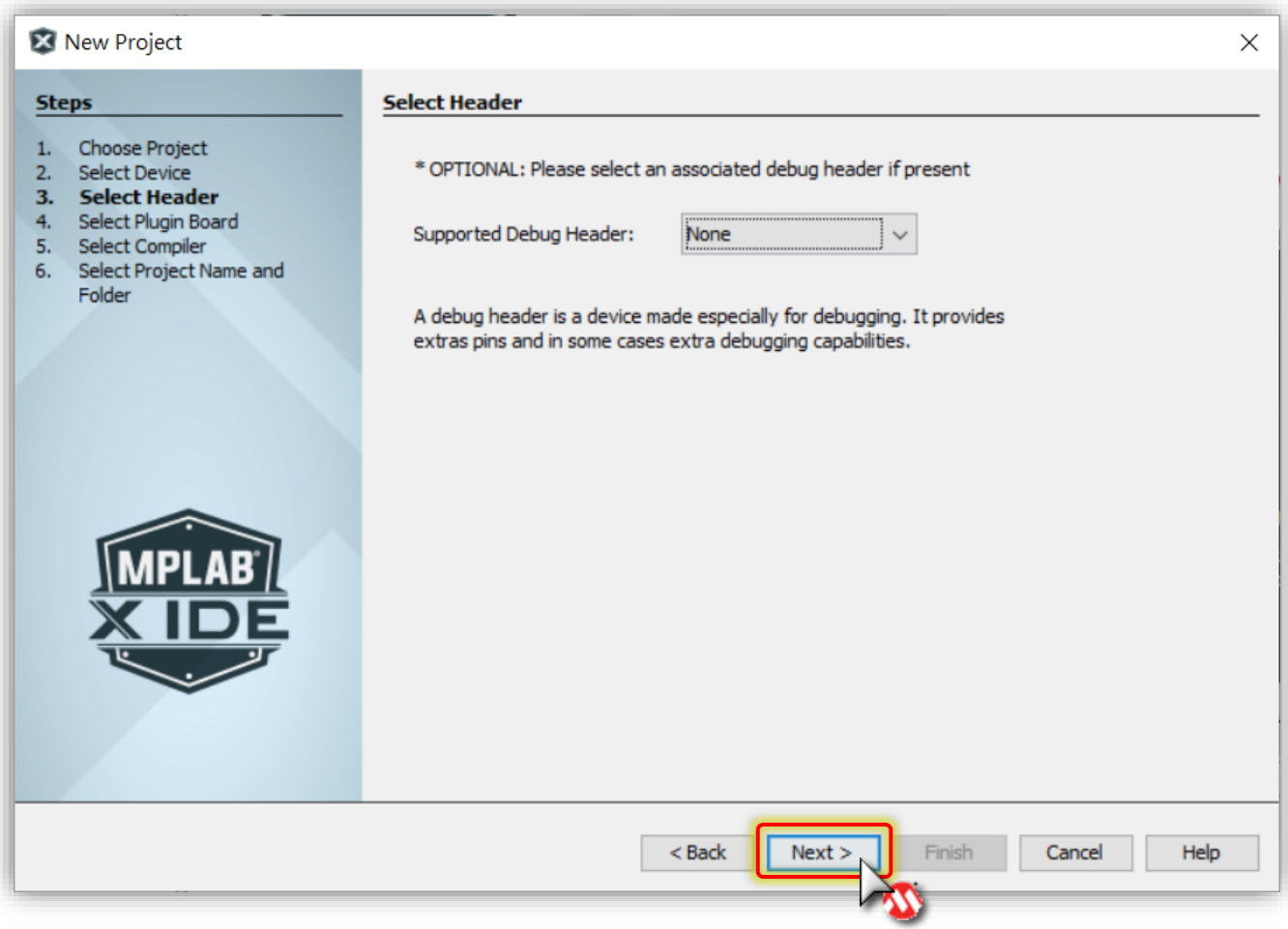
- Family : Mid-Range 8-bit MCUs (PIC10/12/16/MCP) , Device : PIC16F1937
- Tool : Snap-SN : BURXXXXXXXXX



Lab0 First Project

Step 4

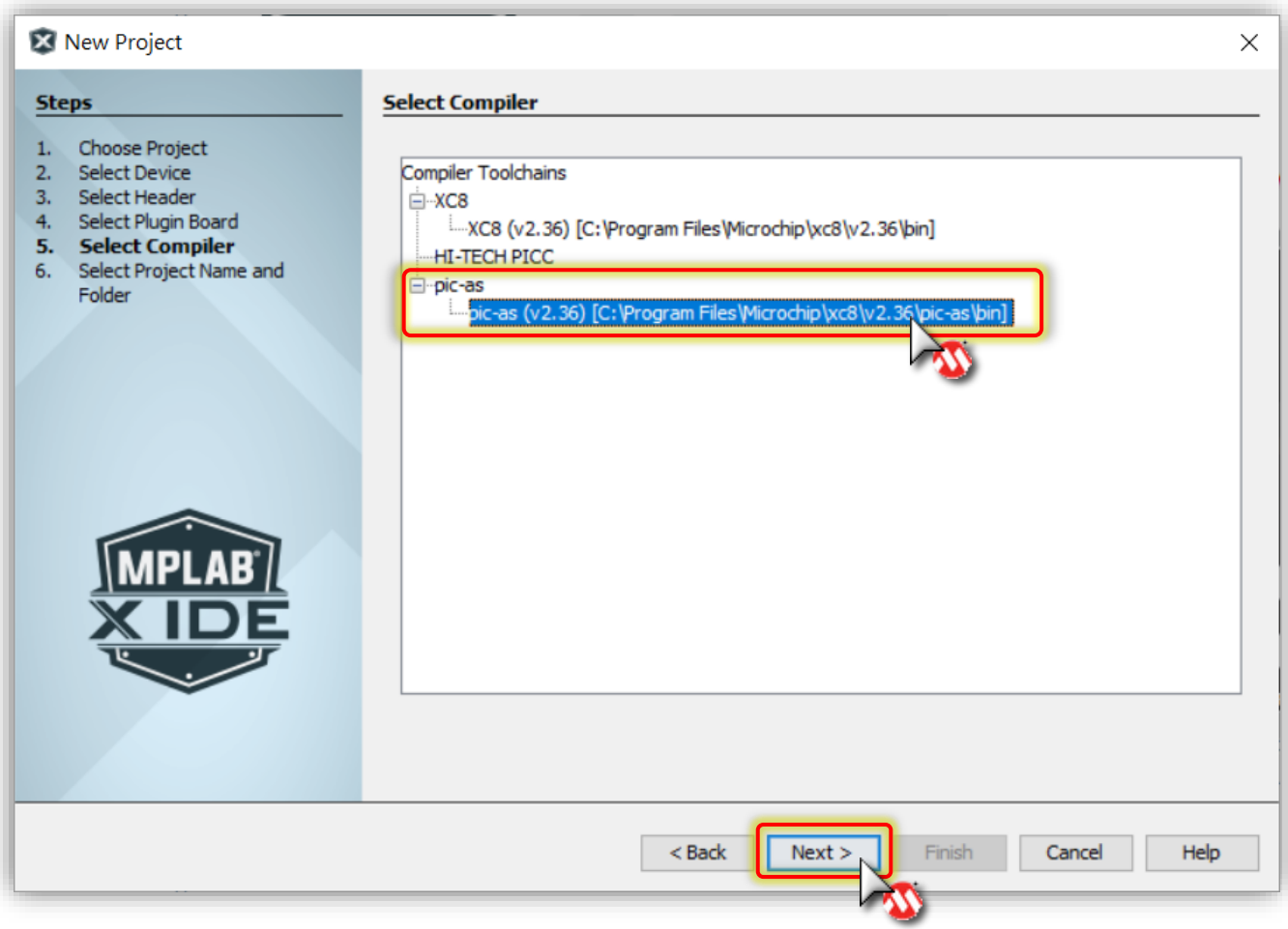
- **Supported Debug Header : None**



Lab0 First Project

Step 5 (Use pic-as)

- **Compiler Toolchains : pic-as (v2.36) [C:\Program Files\Microchip\xc8\v2.36\pic-as\bin]**

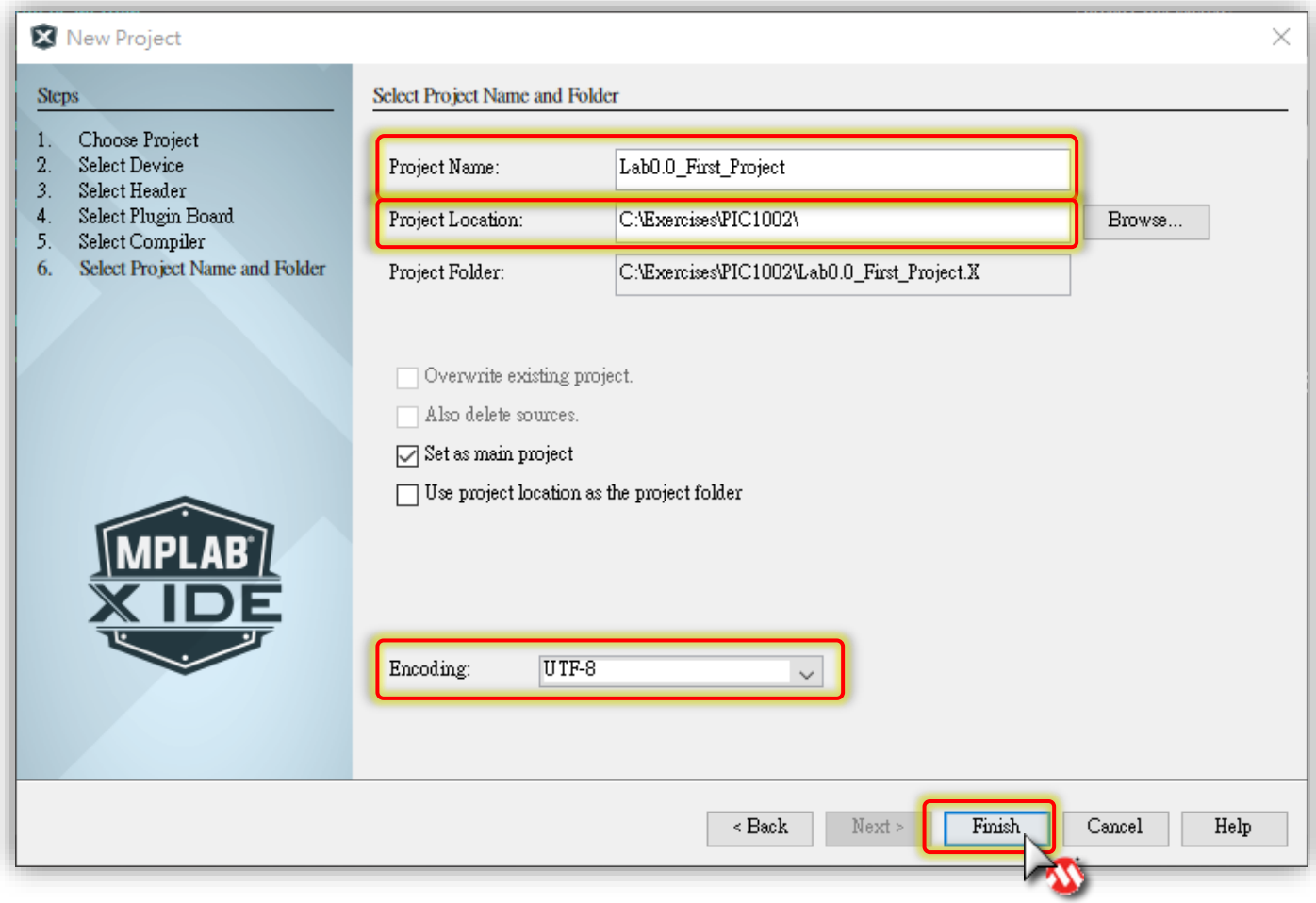


MICROCHIP

Lab0 First Project

Step 6

- Project Name : Lab0.0_First_Project , Project Location : C:\Exercises_PIC1002
- Encoding : UTF-8

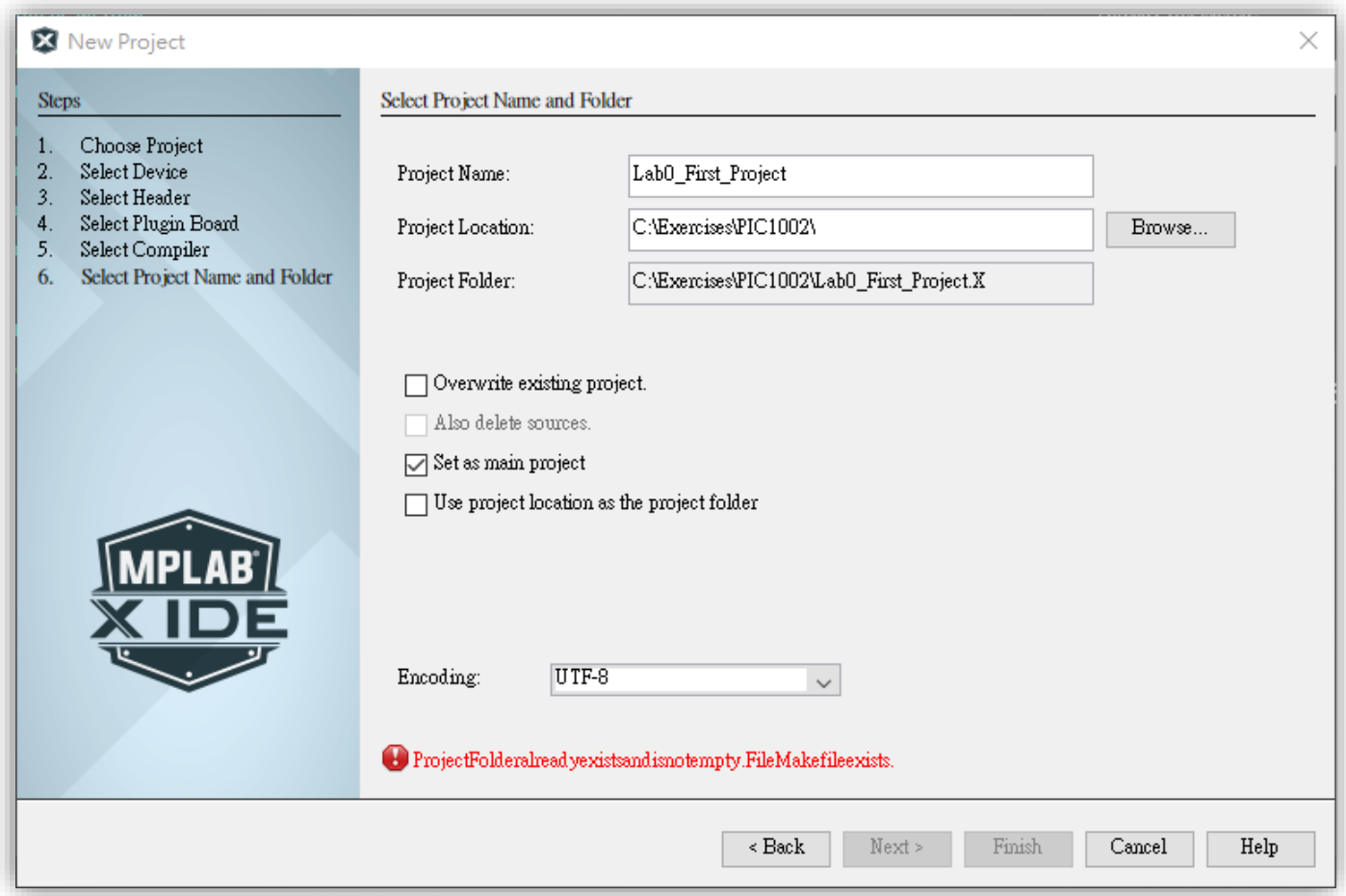


MICROCHIP

Lab0 First Project

Notice

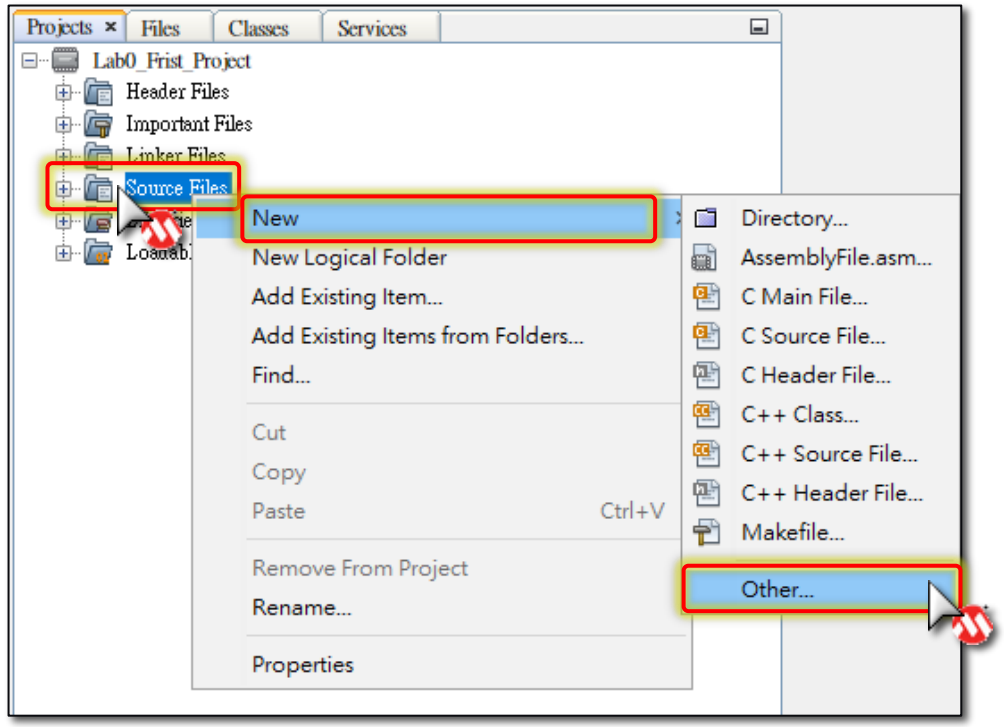
- **If an old project already exist at the same location, you must delete it firstly or change to new project folder.**

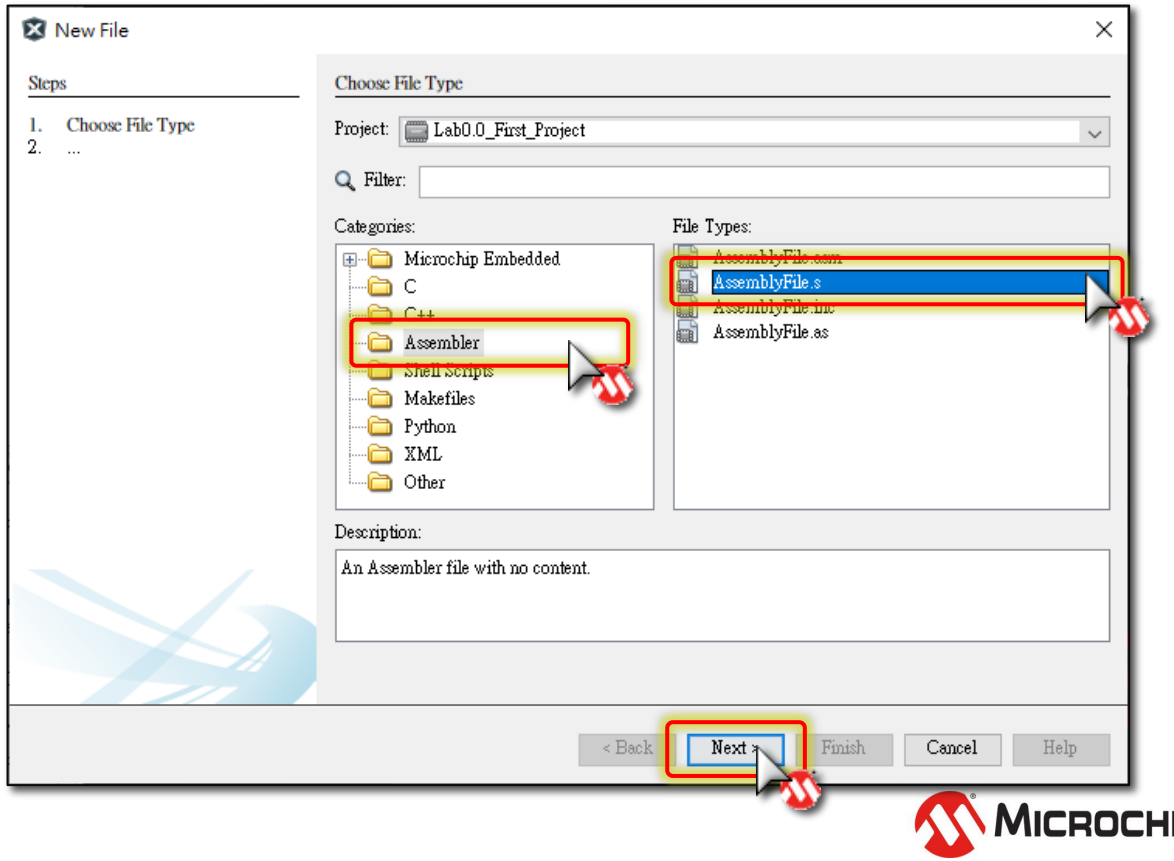


Lab0 First Project

Step 7

- **Create a main.c file in the project.**
  Projects ▸ Source Files ▸ New ▸ Other… ▸ New File ▸ Assembler ▸ AssemblyFile.s

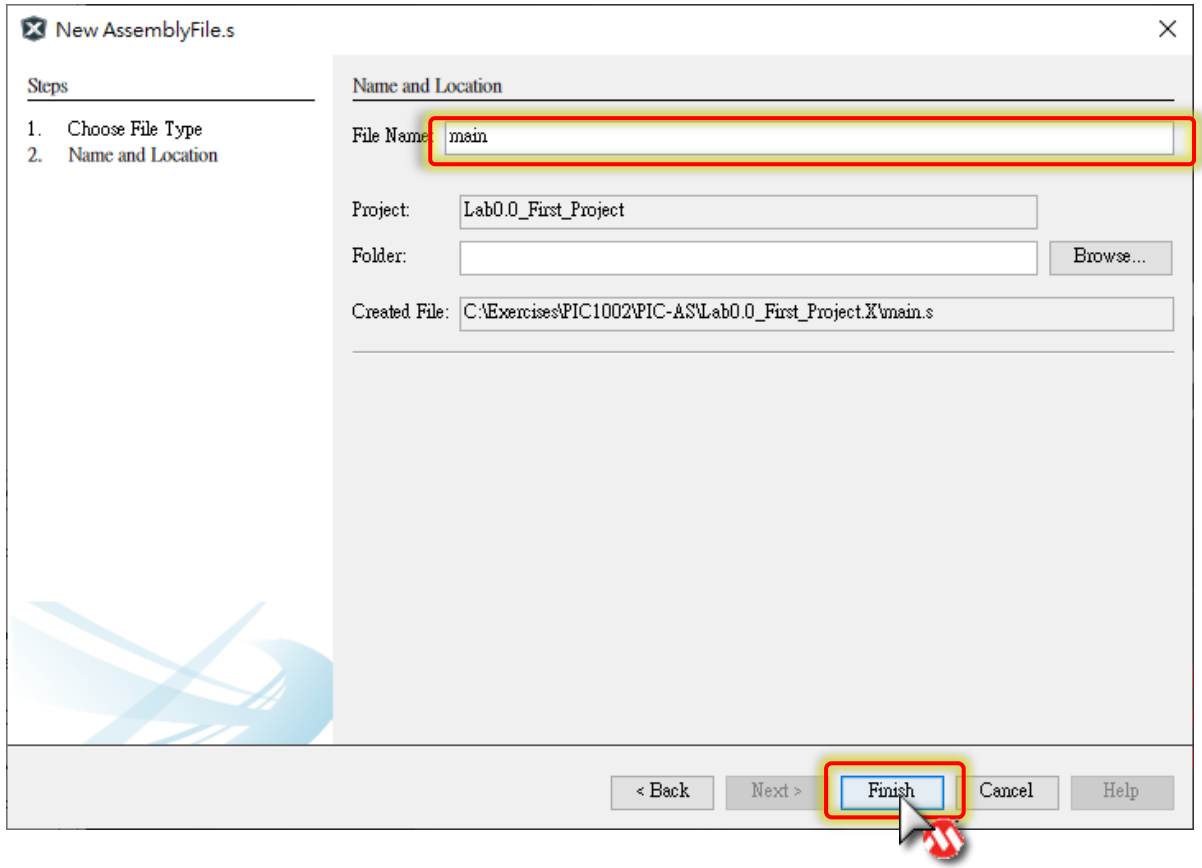




Lab0 First Project

Step 8

- **File Name : main**



Lab0 First Project

Step 9

a. Add code segment to your main.s

```
// TODO 0.01
#include <xc.inc>

// TODO 0.02
PSECT resetVec,class=CODE,delta=2,abs
   ORG 0x00
resetVec:
   PAGESEL main ;jump to the main routine
   goto main

// TODO 0.03
PSECT code
main:

loop:

   goto loop ;main loop

   END
```

Lab0 First Project

Step 10

Add code segment to your main.s

```
// TODO 0.01
#include <xc.inc>
…

// TODO 0.04
PSECT udata_bank0
delay_ms_count:
   DS 1
delay_ms_count_1:
   DS 1
;-----------------------------------------------------------------------
PSECT code
delay_ms:
…
return
;-----------------------------------------------------------------------
```

Lab0 First Project

Step 11

a **Add code segment to your main.s**

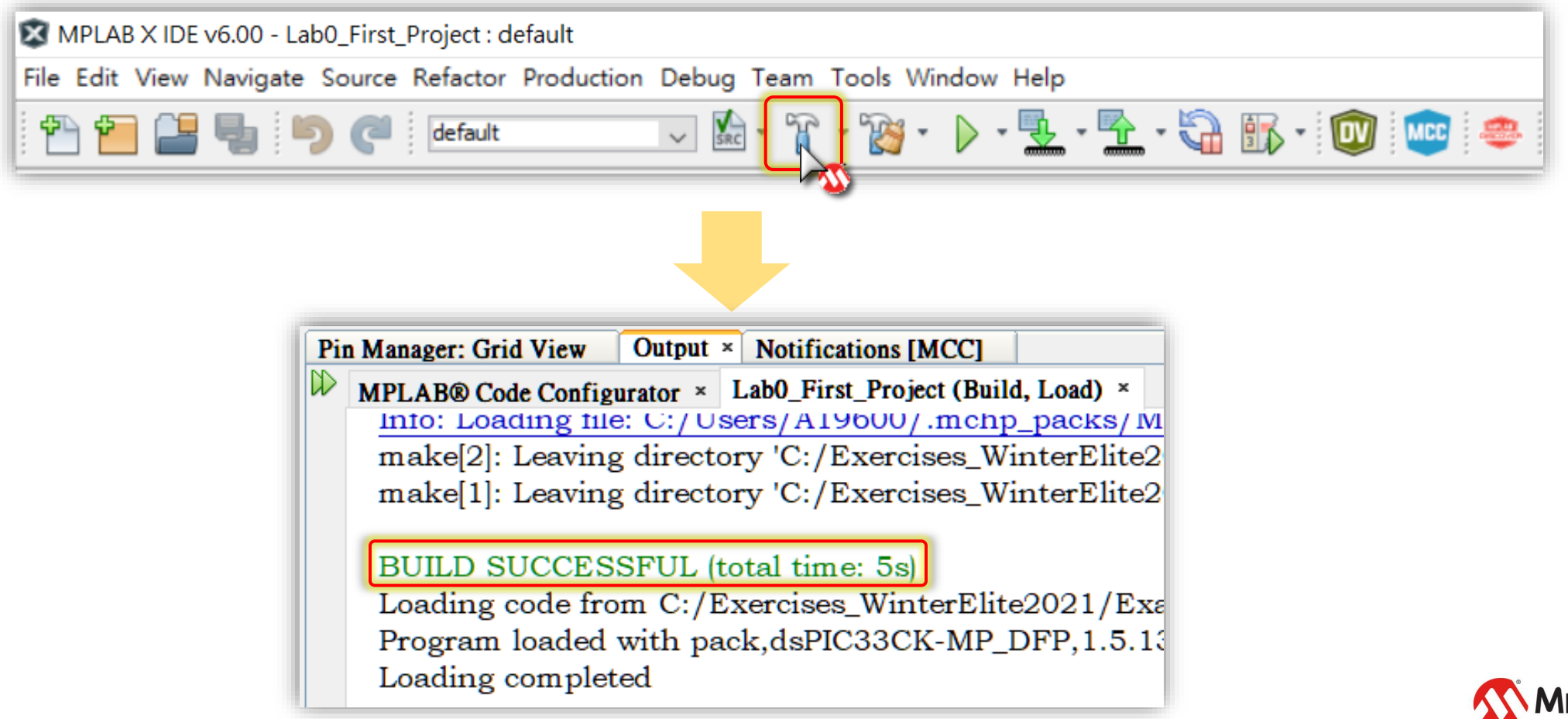
```
// TODO 0.05
; CONFIG1
  CONFIG  FOSC = INTOSC      ; INTOSC oscillator
  CONFIG  WDTE = OFF         ; WDT disabled

; CONFIG2
  CONFIG  PLLEN = OFF        ; 4x PLL disabled

// TODO 0.01
#include <xc.inc>
…
```

b **Program firmware to target board then observe result.**

Lab0 First Project

Step 12

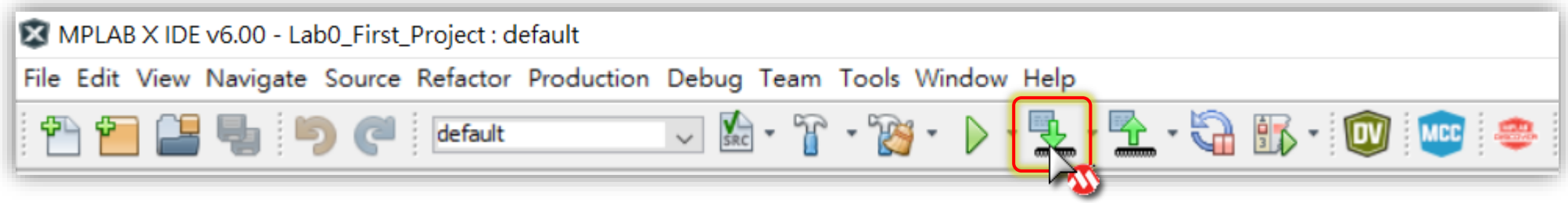
- Select **Build Main Project** icon
- Make sure compiled result is **BUILD SUCCESSFUL**.

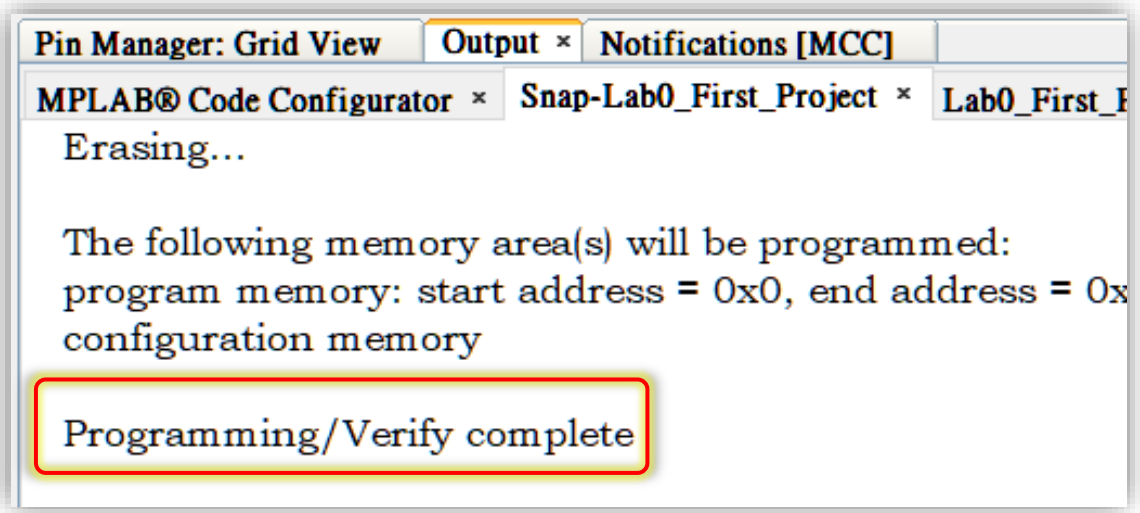


Lab0 First Project

Step 13

- Select **Make & Program Device Main Project** icon 
- Make sure the result is **Programming/Verify complete**.

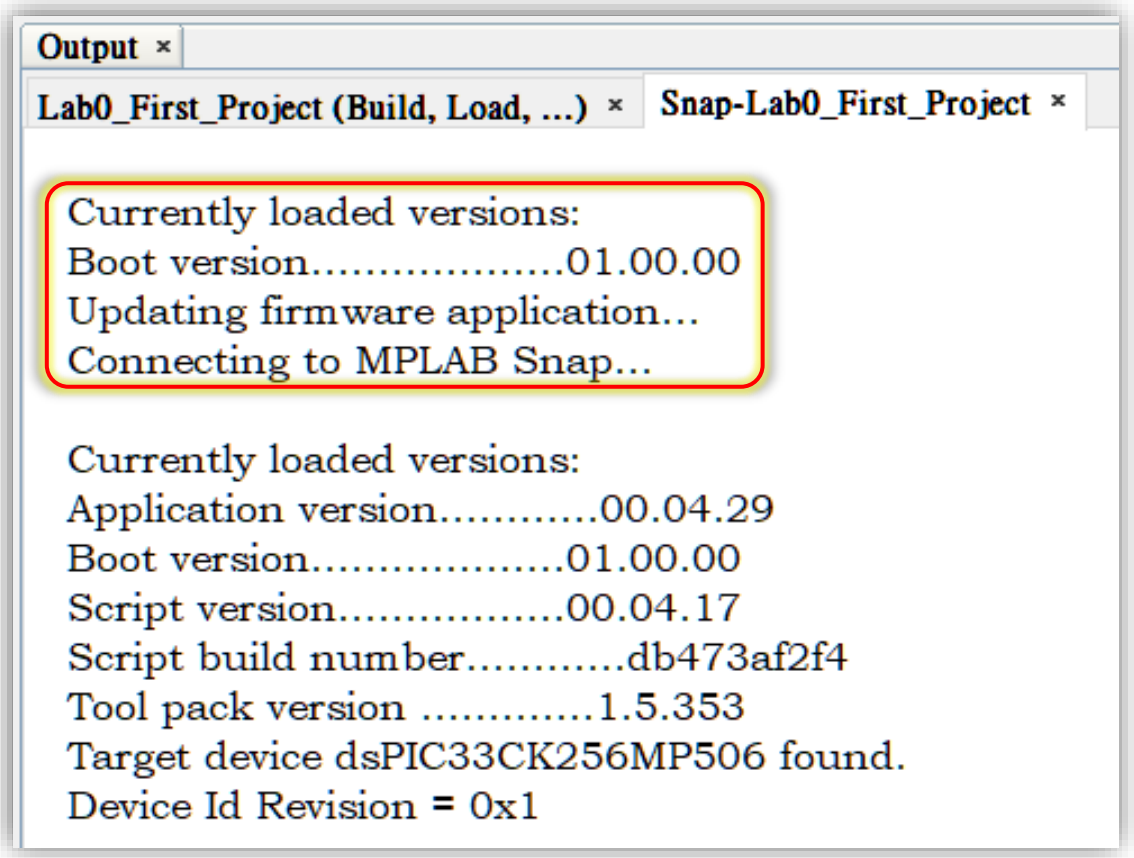




Lab0 First Project

Notice

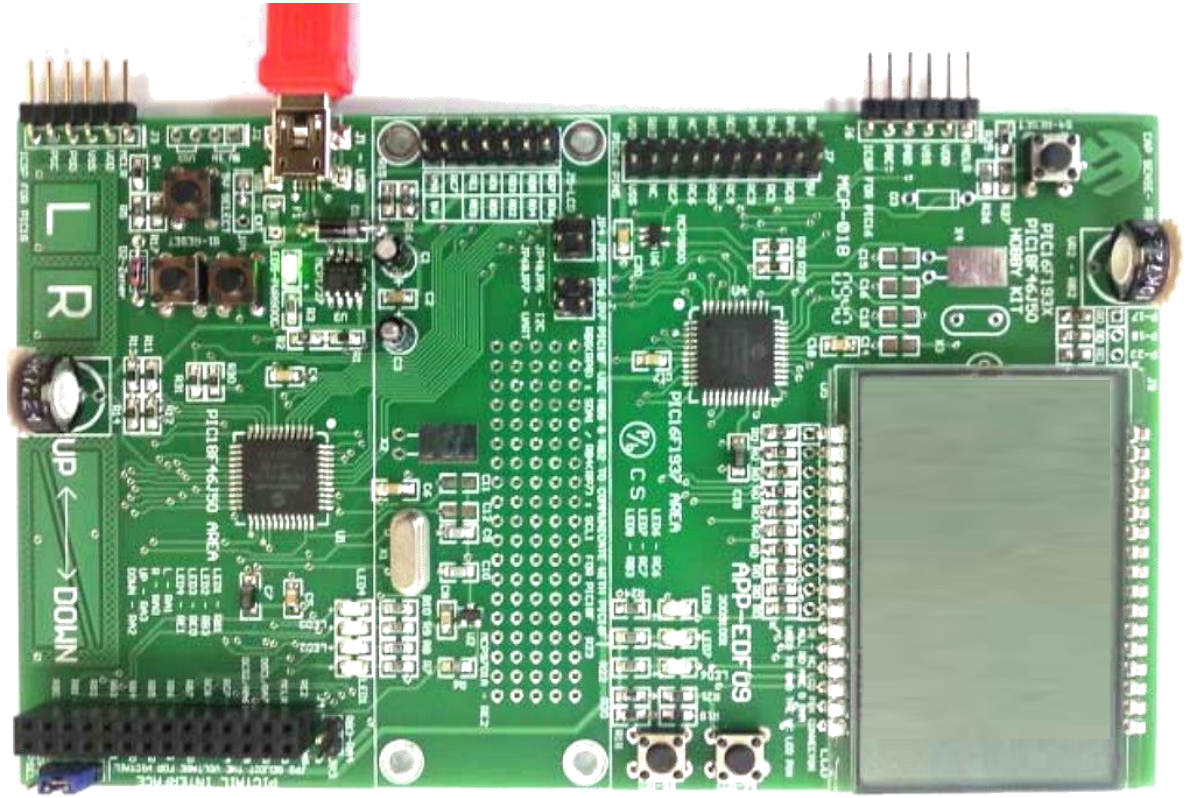
- **MPLAB X IDE will auto upgrade the firmware of SNAP to keep it up to date, it might take couple minutes to wait for finish. Your project will start programming after this.**



Lab0 First Project

Result

Nothing happen ????



Lab0 First Project

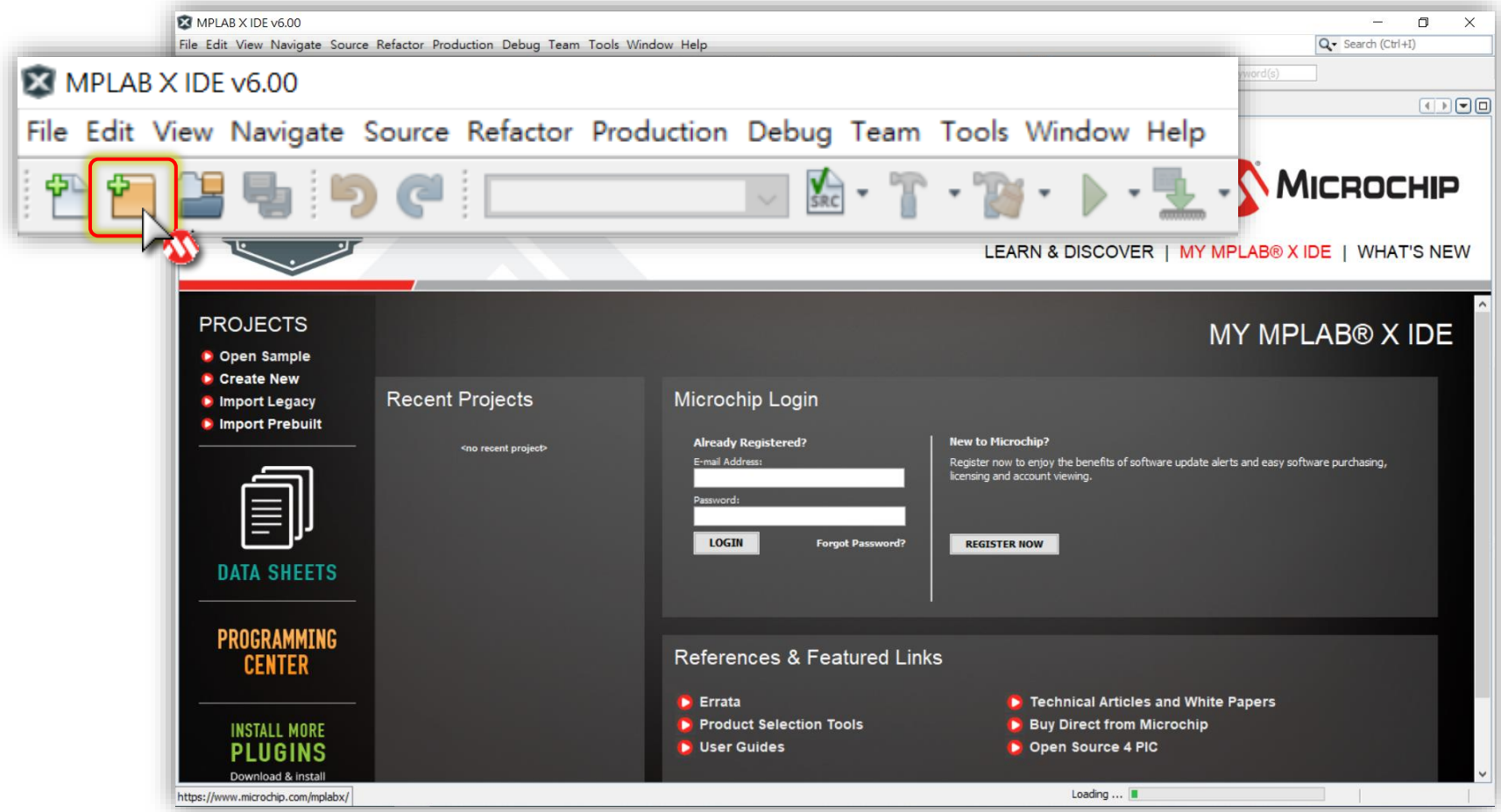
- Base on current project or Open .\Lab0.1_xxx.X to do exercises
- Try to use Bare-Metal C to create your First Project.
- To understanding MPLAB X IDE basic operation.
- Learn how to use edit, build, project manager and program functions.

How to start ?

Lab0 First Project

Step 1

- **Execute Project Wizard Select : File ▸ New Project or Click icon** 

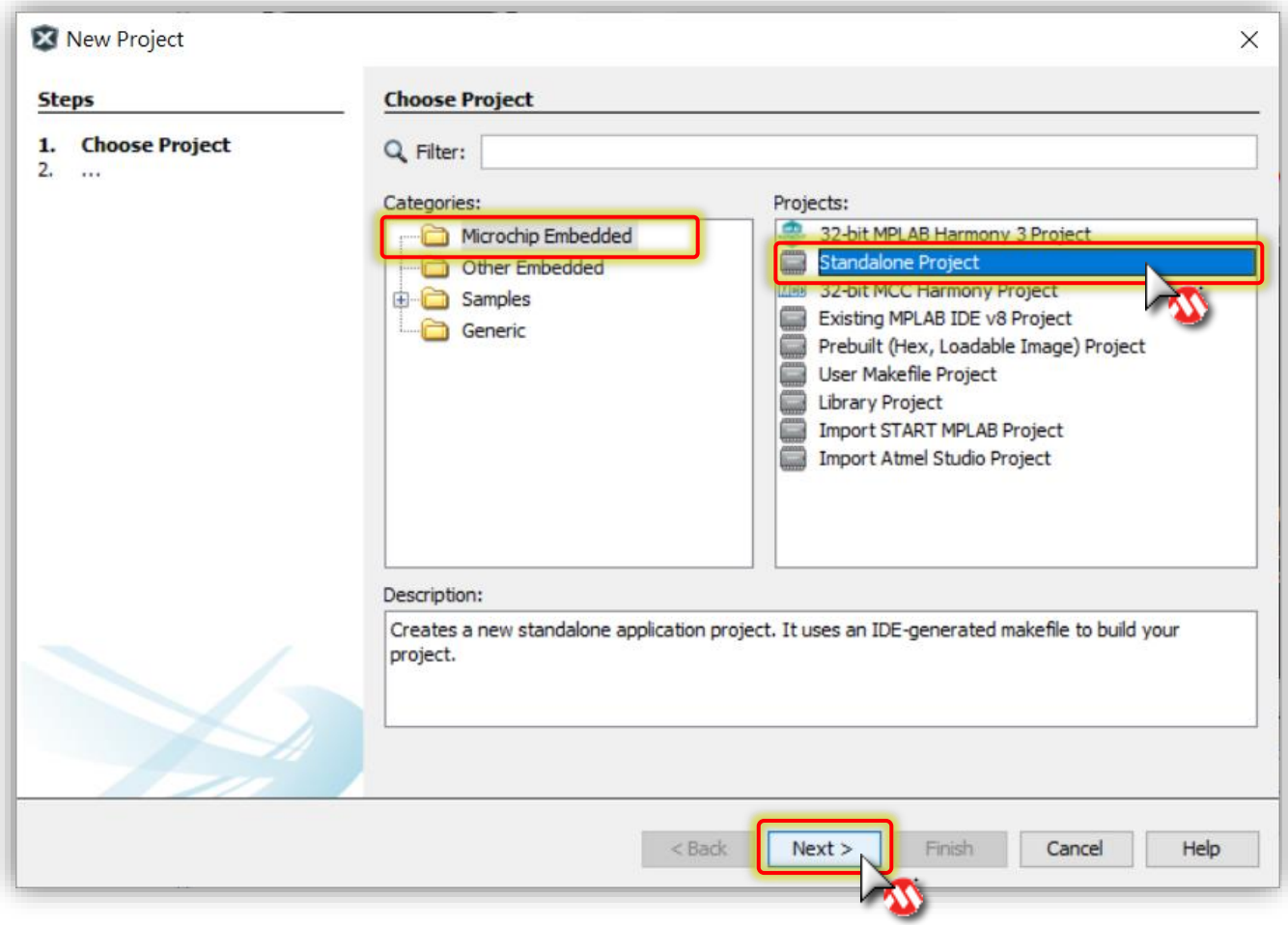


MICROCHIP

Lab0 First Project

Step 2

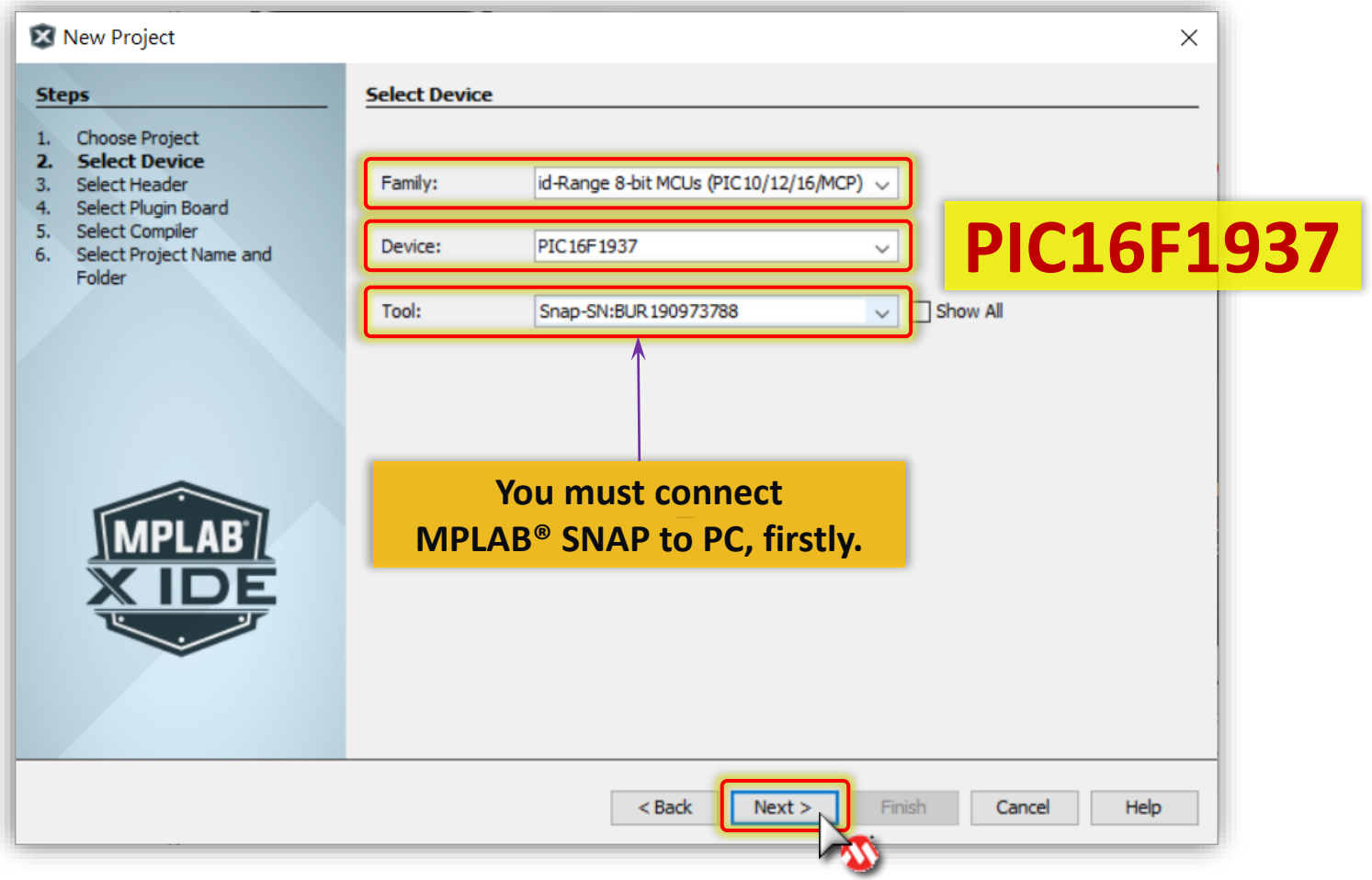
- Categories : Microchip Embedded
- Projects : Standalone Project



Lab0 First Project

Step 3

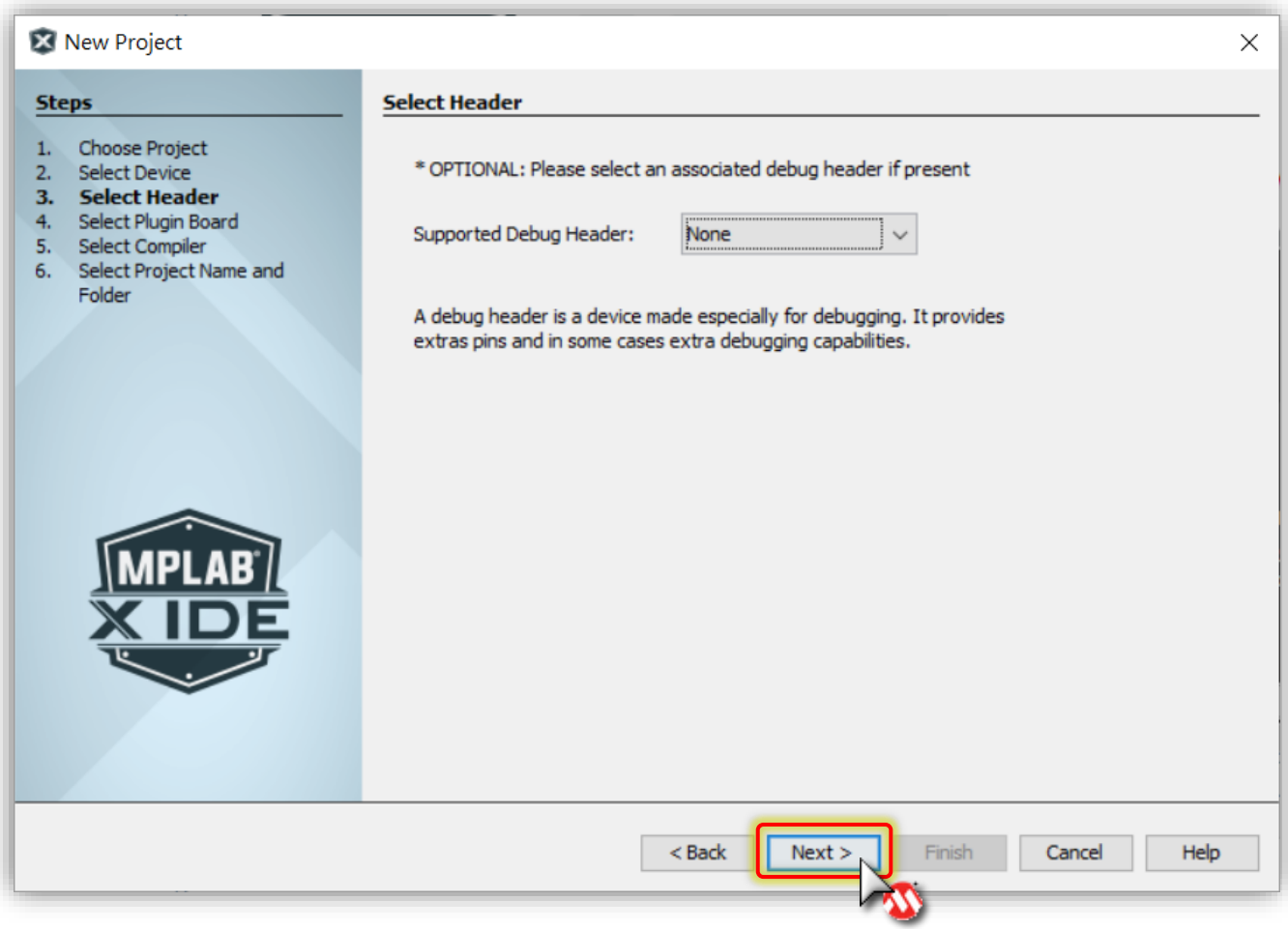
- Family : Mid-Range 8-bit MCUs (PIC10/12/16/MCP) , Device : PIC16F1937
- Tool : Snap-SN : BURXXXXXXXXX



Lab0 First Project

Step 4

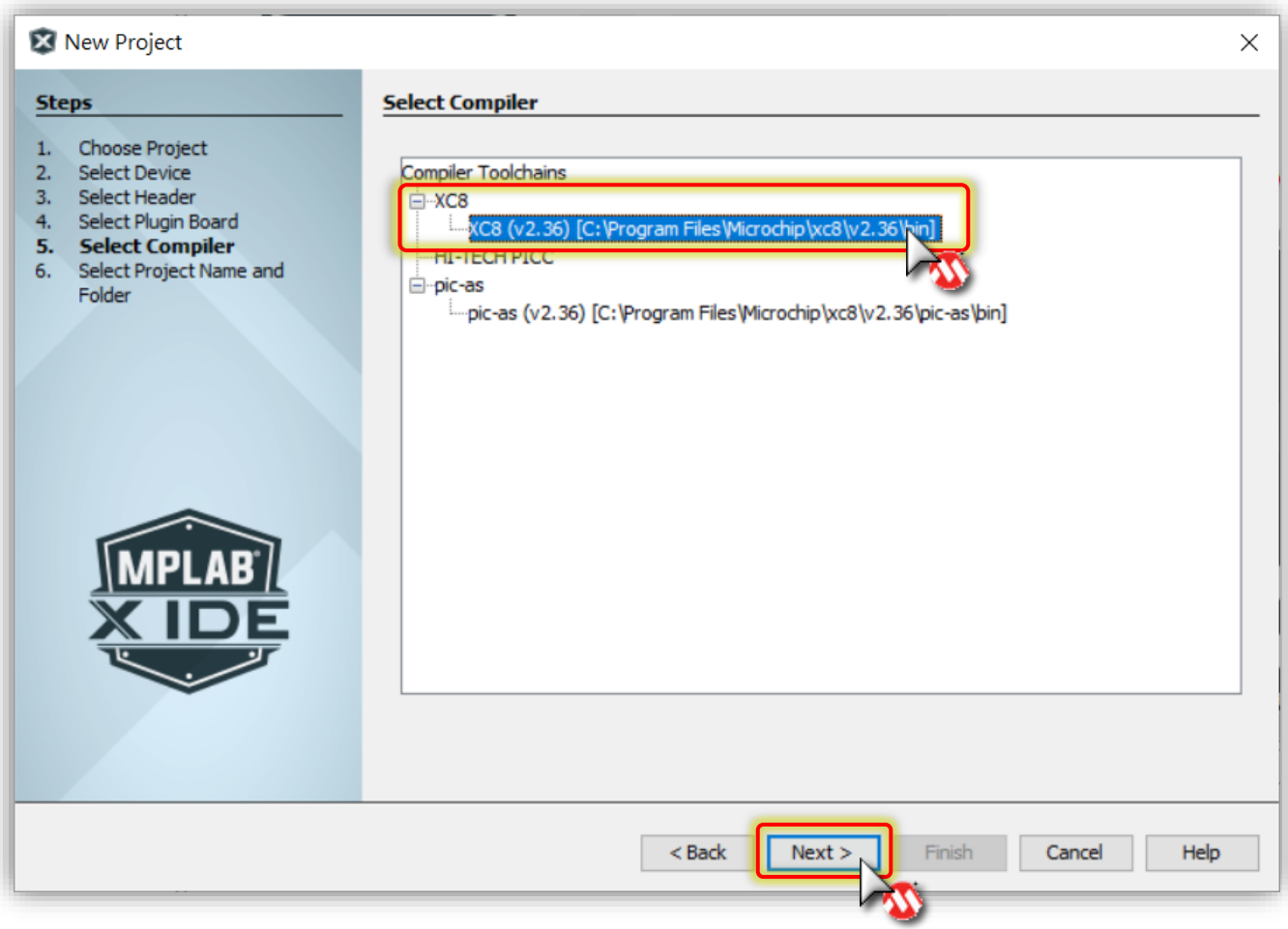
- **Supported Debug Header : None**



Lab0 First Project

Step 5 (Use Bare metal C and MCC)

- **Compiler Toolchains : XC8 (v2.36) [C:\Program Files\Microchip\xc8\v2.36\bin]**

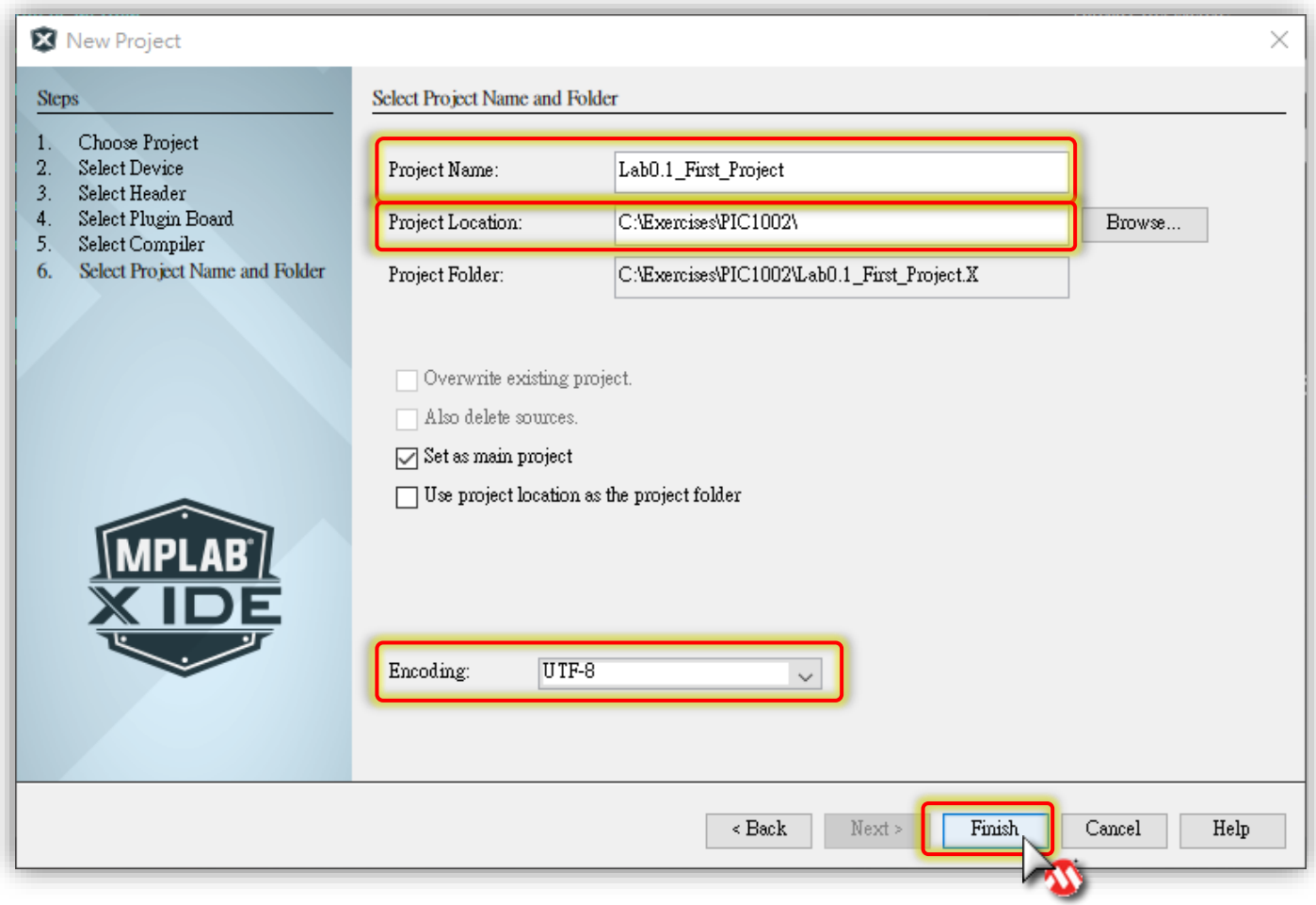


MICROCHIP

Lab0 First Project

Step 6

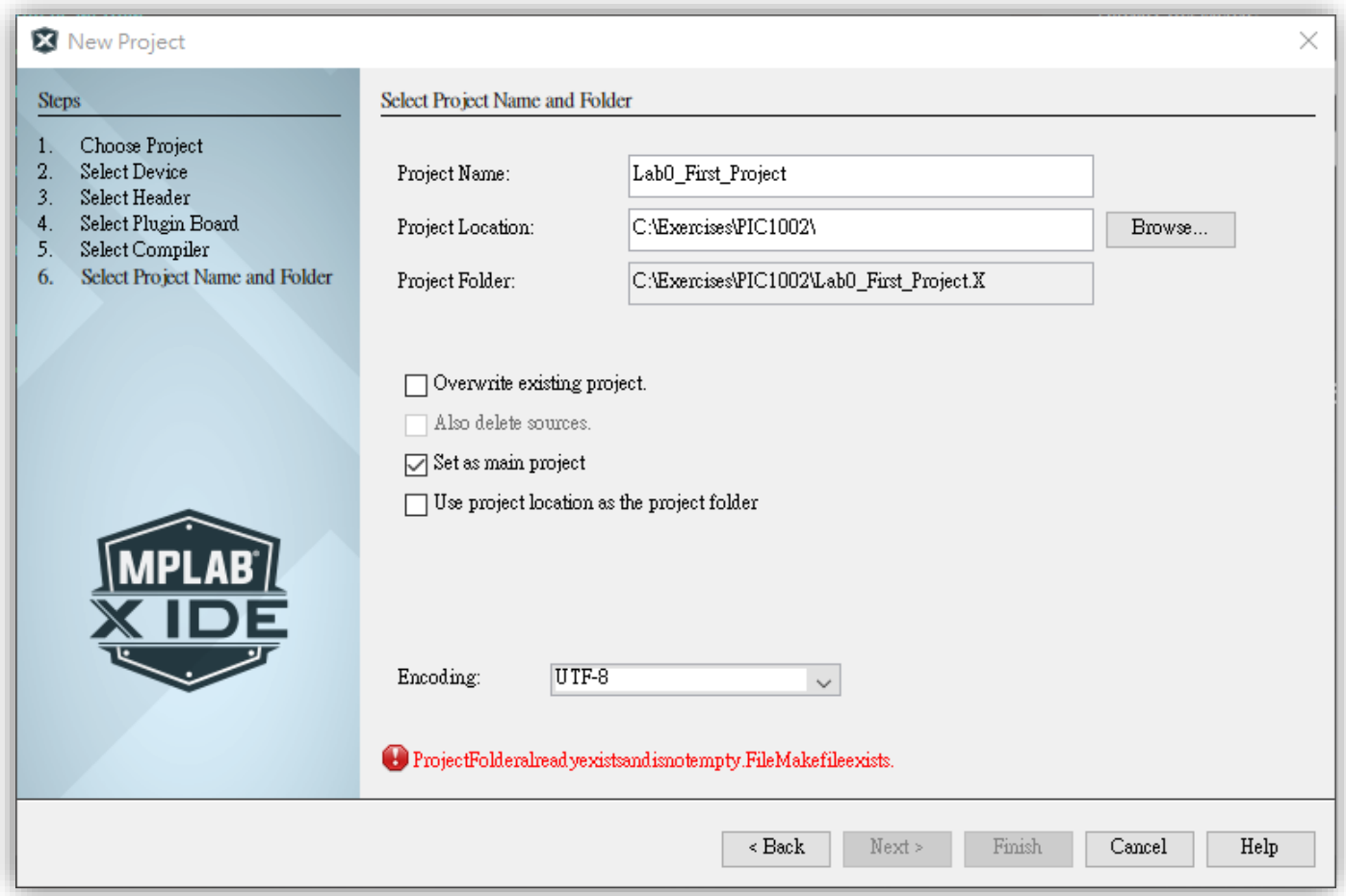
- Project Name : Lab0.1_First_Project , Project Location : C:\Exercises_PIC1002
- Encoding : UTF-8



Lab0 First Project

Notice

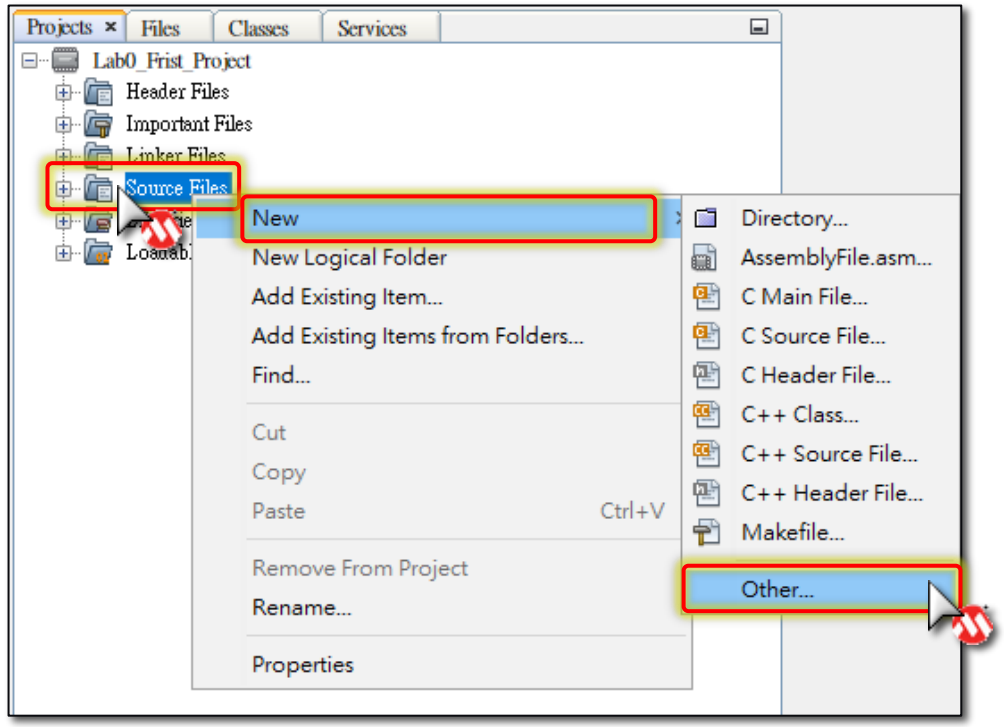
- **If an old project already exist at the same location, you must delete it firstly or change to new project folder.**

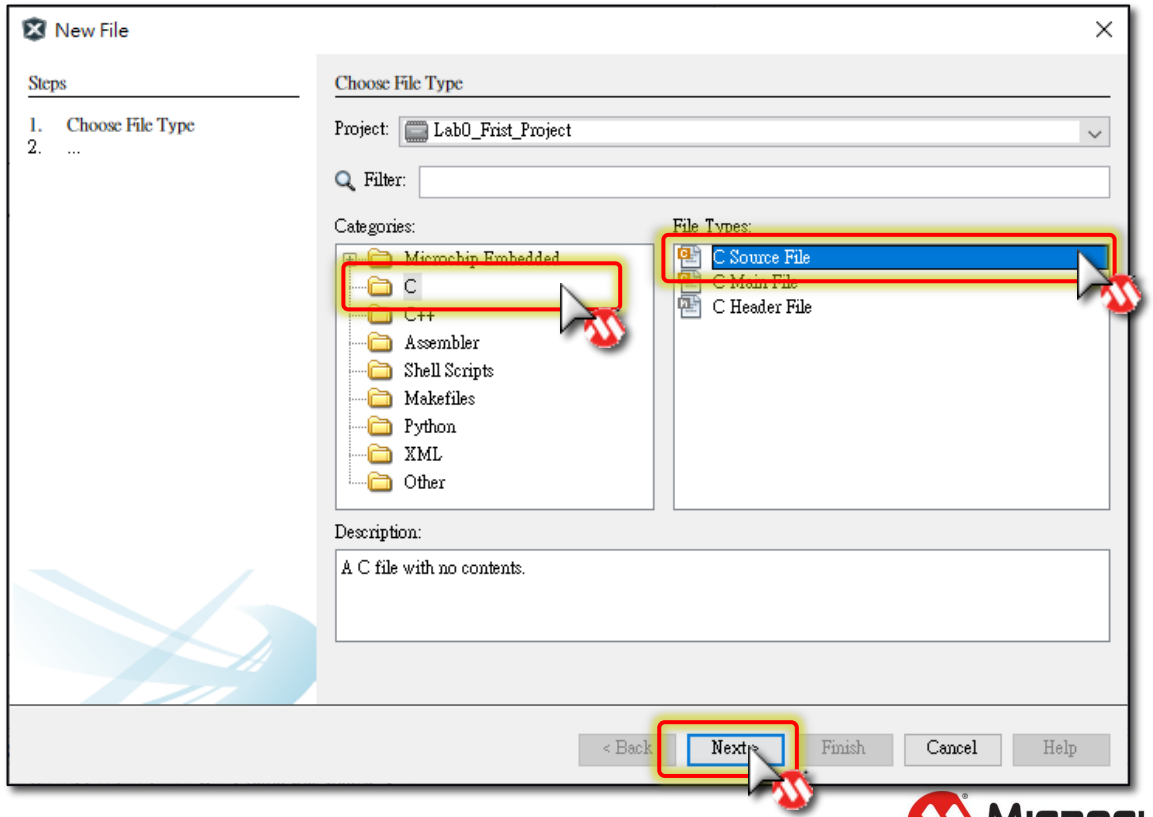


Lab0 First Project

Step 7

- **Create a main.c file in the project.**
  Projects ▸ Source Files ▸ New ▸ Other… ▸ New File ▸ C ▸ C Source File

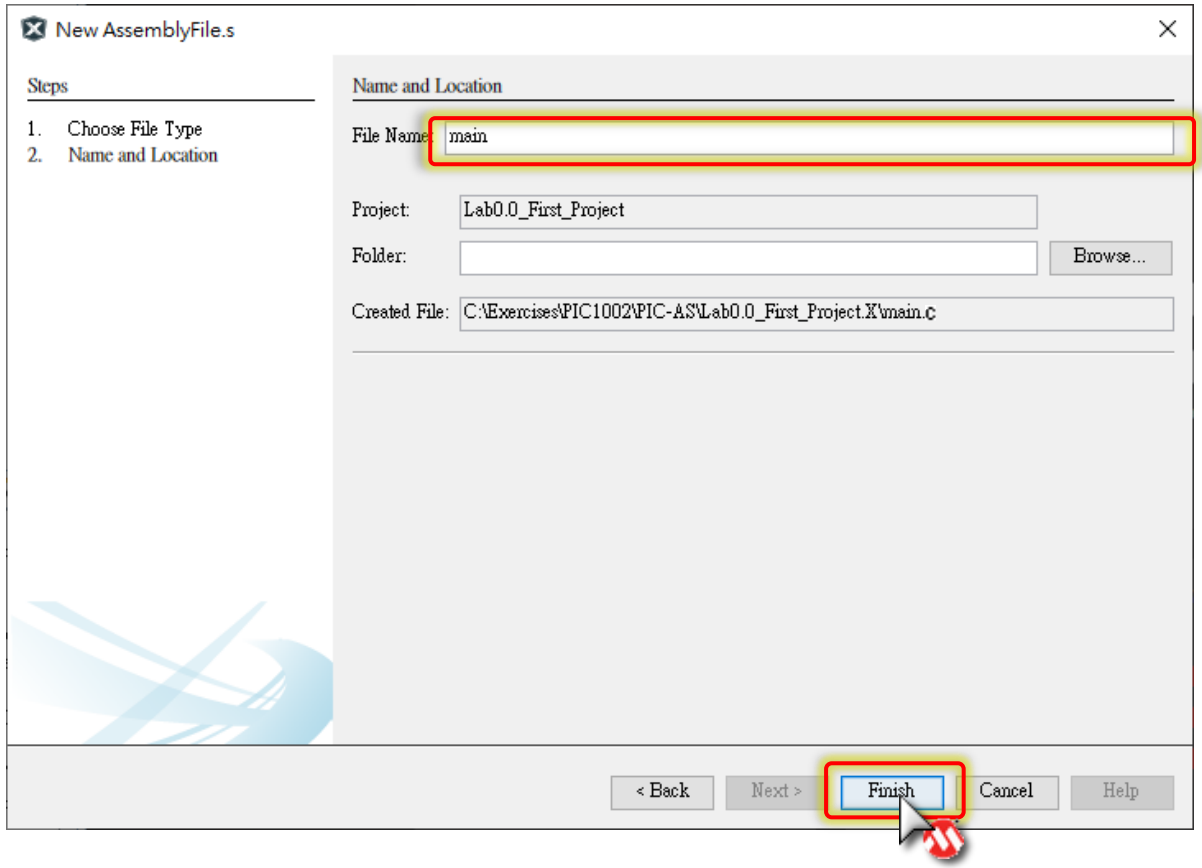




Lab0 First Project

Step 8

- File Name : main



Lab0 First Project

Step 9

a. Add code segment to your main.c

```
// TODO 0.01
#include <xc.h>

// TODO 0.02
void main(void)
{
   // TODO 0.03
   while (1)    //main loop
   {

   }

   return;
}
```

Lab0 First Project

Step 10

a **Add code segment to your main.c**

```
// TODO 0.01
#include <xc.h>
…

// TODO 0.04
unsigned char delay_ms_count;
unsigned char delay_ms_count_1;
;-------------------------------------------------------------------------
void delay_ms(unsigned char count)
{
…
}
;-------------------------------------------------------------------------
```

Lab0 First Project

Step 11

a. **Add code segment to your main.c**

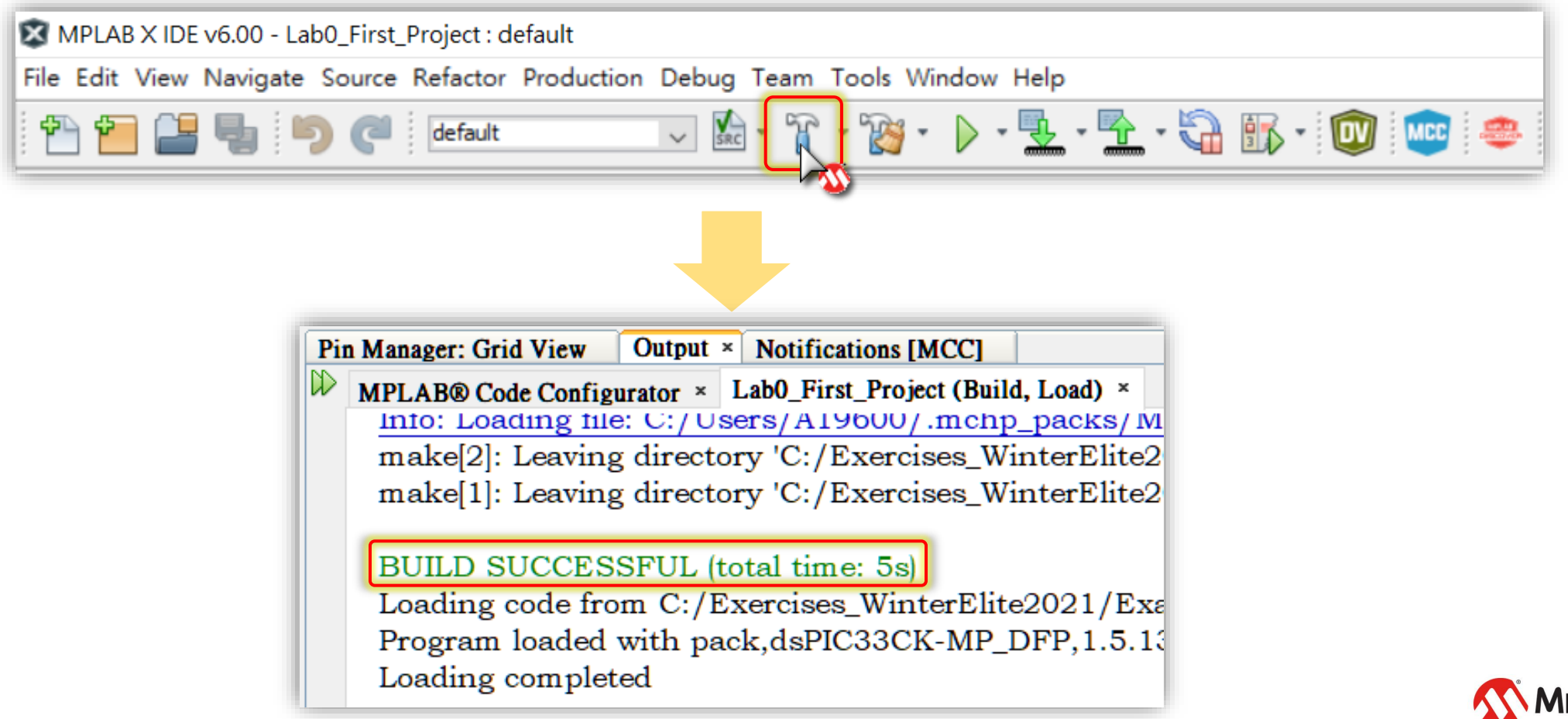
```
// TODO 0.05
// CONFIG1
#pragma config FOSC = INTOSC        // INTOSC oscillator
#pragma config WDTE = OFF           // WDT disabled

// CONFIG2
#pragma config PLLEN = OFF          // 4x PLL disabled

// TODO 0.01
#include <xc.h>
...
```

b. **Program firmware to target board then observe result.**

Lab0 First Project

Step 12

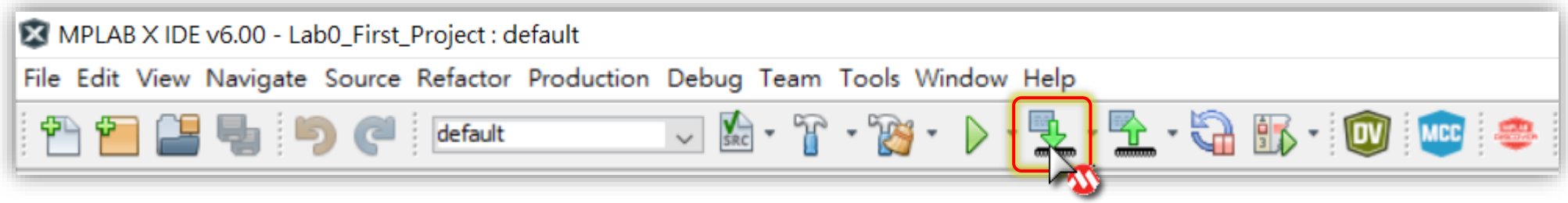
- Select **Build Main Project** icon
- Make sure compiled result is **BUILD SUCCESSFUL**.

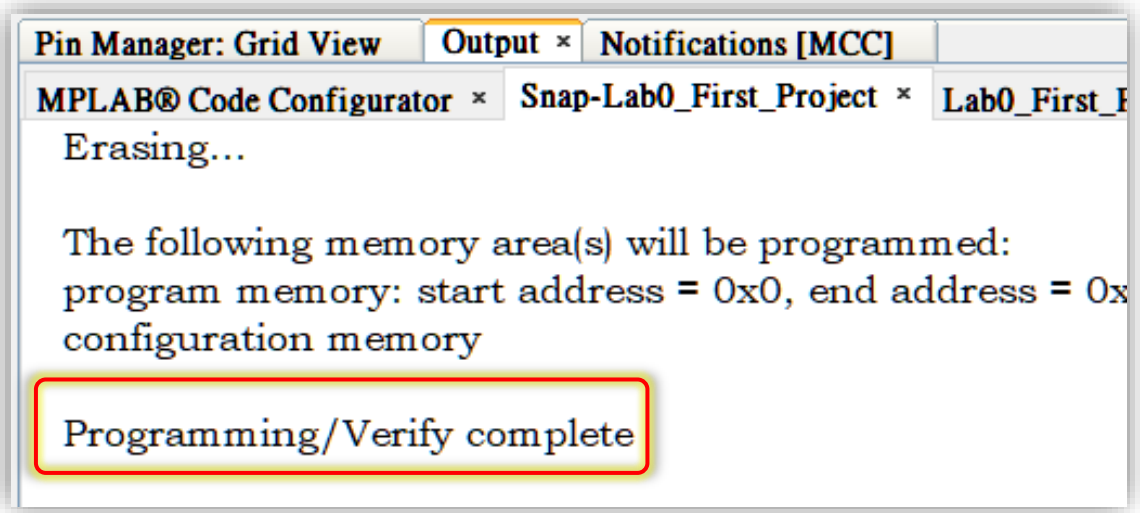


Lab0 First Project

Step 13

- Select **Make & Program Device Main Project** icon 
- Make sure the result is **Programming/Verify complete**.

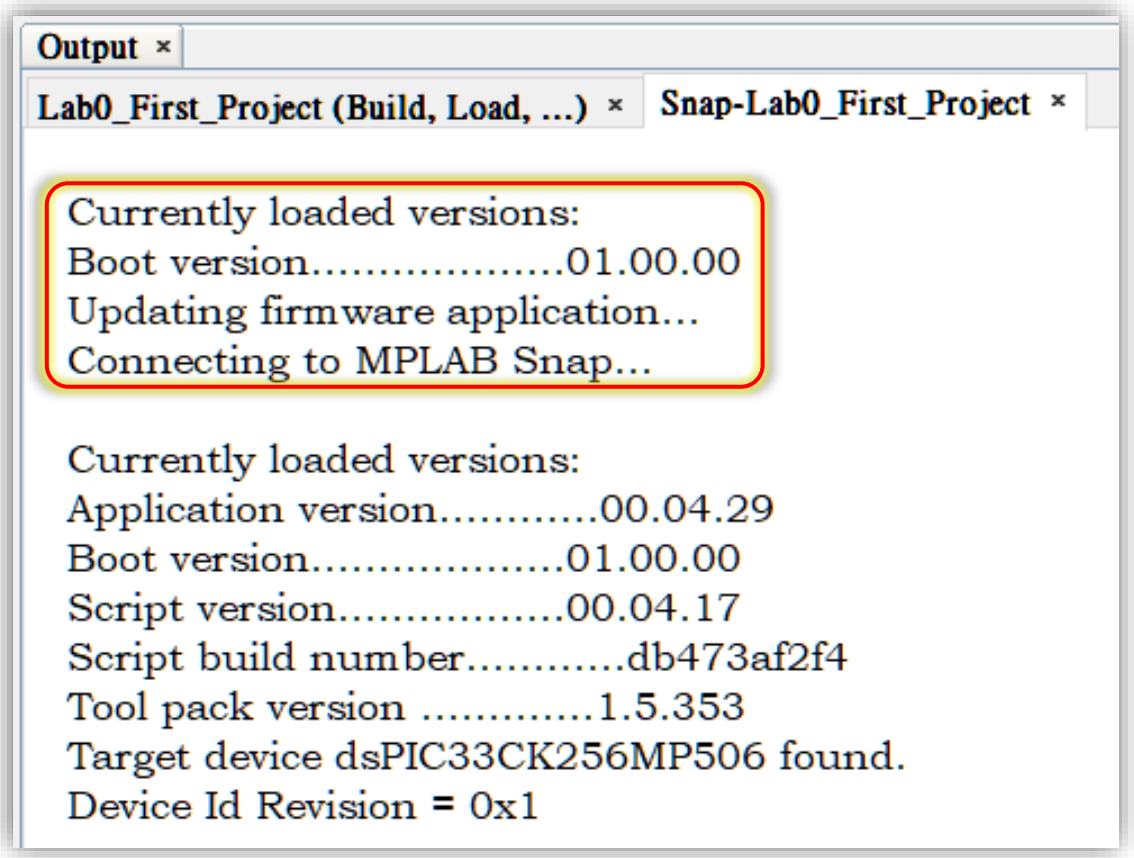




Lab0 First Project

Notice

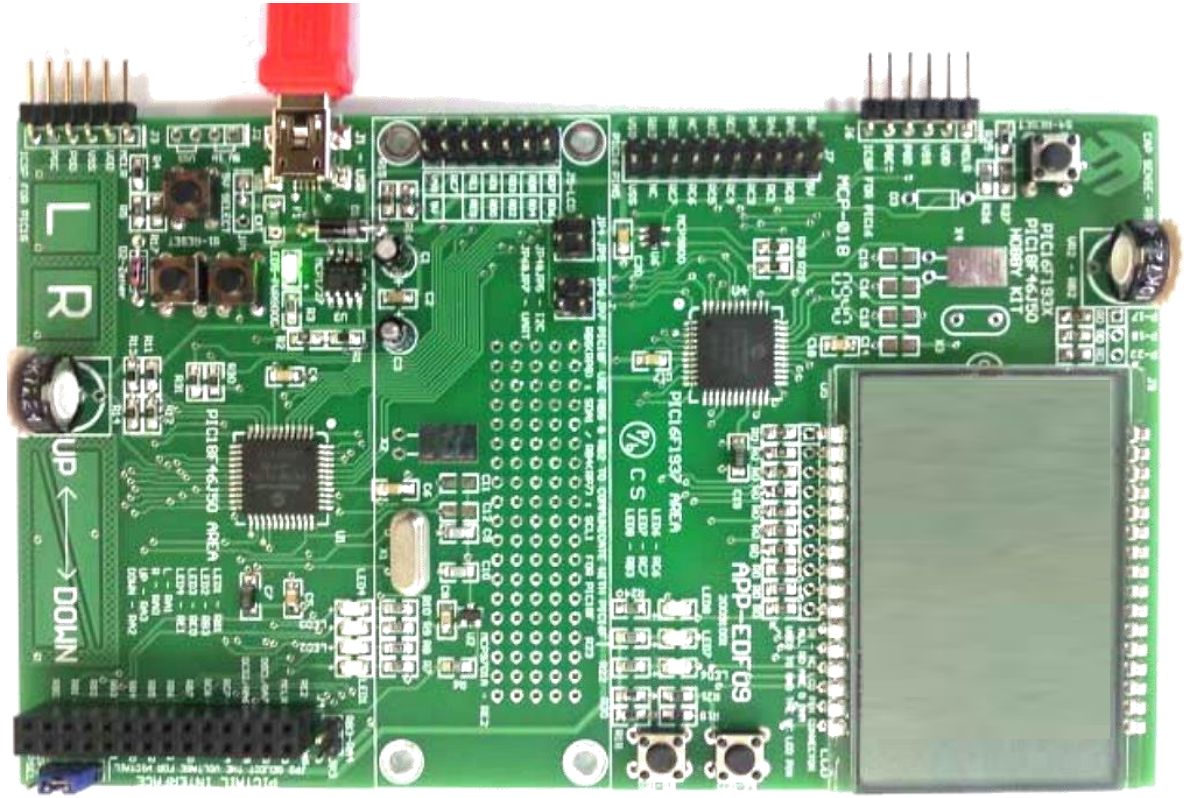
- **MPLAB X IDE will auto upgrade the firmware of SNAP to keep it up to date, it might take couple minutes to wait for finish. Your project will start programming after this.**



Lab0 First Project

Result

Nothing happen ????



Discuss

PIC Assembler vs MPASM Assembler

- **Discuss**
  - What is the difference between the First Project of XC8 PIC Assembler and MPASM Assembler ?

XC8 PIC Assembler

```
// TODO 0.05
; CONFIG1
  CONFIG  FOSC = INTOSC                 ; INTOSC oscillator
  CONFIG  WDTE = OFF                    ; WDT disabled

; CONFIG2
  CONFIG  PLLEN = OFF                   ; 4x PLL disabled

// TODO 0.01
#include <xc.inc>

// TODO 0.04
PSECT udata_bank0
delay_ms_count:
   DS 1
delay_ms_count_1:
   DS 1
;-------------------------------------------------------------------------------
PSECT code
delay_ms:
   BANKSEL delay_ms_count
   movwf BANKMASK(delay_ms_count)
   ...
;-------------------------------------------------------------------------------

// TODO 0.02
PSECT resetVec,class=CODE,delta=2,abs
   ORG 0x00
resetVec:
   PAGESEL main ;jump to the main routine
   goto main

// TODO 0.03
PSECT code
main:

loop:

   goto loop ;main loop

   END
```

MPASM Assembler

```
; TODO 0.05
; CONFIG1
 __CONFIG _CONFIG1, _FOSC_INTOSC & _WDTE_OFF       ; INTOSC oscillator, WDT disabled

; CONFIG2
 __CONFIG _CONFIG2, _PLLEN_OFF                     ; 4x PLL disabled


; TODO 0.01
#include "p16f1937.inc"

; TODO 0.04
udata 0x20
delay_ms_count RES 1

delay_ms_count_1 RES 1

;-------------------------------------------------------------------------------
ORG 0x10
delay_ms:
   BANKSEL delay_ms_count
   movwf delay_ms_count
   ...
;-------------------------------------------------------------------------------

; TODO 0.02
ORG 0x00

resetVec:
   PAGESEL main ;jump to the main routine
   goto main

; TODO 0.03
ORG 0x700
main:

loop:

   goto loop ;main loop

   END
```

MICROCHIP

Assembly vs C

- **Discuss**
  - What is the difference between the First Project of Assembly and Bare Metal C ?

XC8 PIC Assembler

```
// TODO 0.05
; CONFIG1
  CONFIG  FOSC = INTOSC                 ; INTOSC oscillator
  CONFIG  WDTE = OFF                    ; WDT disabled

; CONFIG2
  CONFIG  PLLEN = OFF                   ; 4x PLL disabled

// TODO 0.01
#include <xc.inc>

// TODO 0.04
PSECT udata_bank0
delay_ms_count:
   DS 1
delay_ms_count_1:
   DS 1
;-----------------------------------------------------------------------
PSECT code
delay_ms:
   BANKSEL delay_ms_count
   movwf BANKMASK(delay_ms_count)
   ...
;-----------------------------------------------------------------------

// TODO 0.02
PSECT resetVec,class=CODE,delta=2,abs
   ORG 0x00
resetVec:
   PAGESEL main ;jump to the main routine
   goto main

// TODO 0.03
PSECT code
main:

loop:

   goto loop ;main loop

   END
```

Bare Metal C

```
// TODO 0.05
// CONFIG1
#pragma config FOSC = INTOSC       // INTOSC oscillator
#pragma config WDTE = OFF          // WDT disabled

// CONFIG2
#pragma config PLLEN = OFF         // 4x PLL disabled

// TODO 0.01
#include <xc.h>

// TODO 0.04

unsigned char delay_ms_count;

unsigned char delay_ms_count_1;


//-----------------------------------------------------------------------
void delay_ms(unsigned char count)
{
...
}
//-----------------------------------------------------------------------

// TODO 0.02
void main(void)
{



   // TODO 0.03

   while (1)   //main loop
   {

   }

   return;
}
```

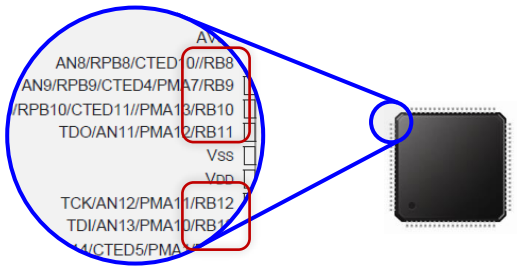
MICROCHIP

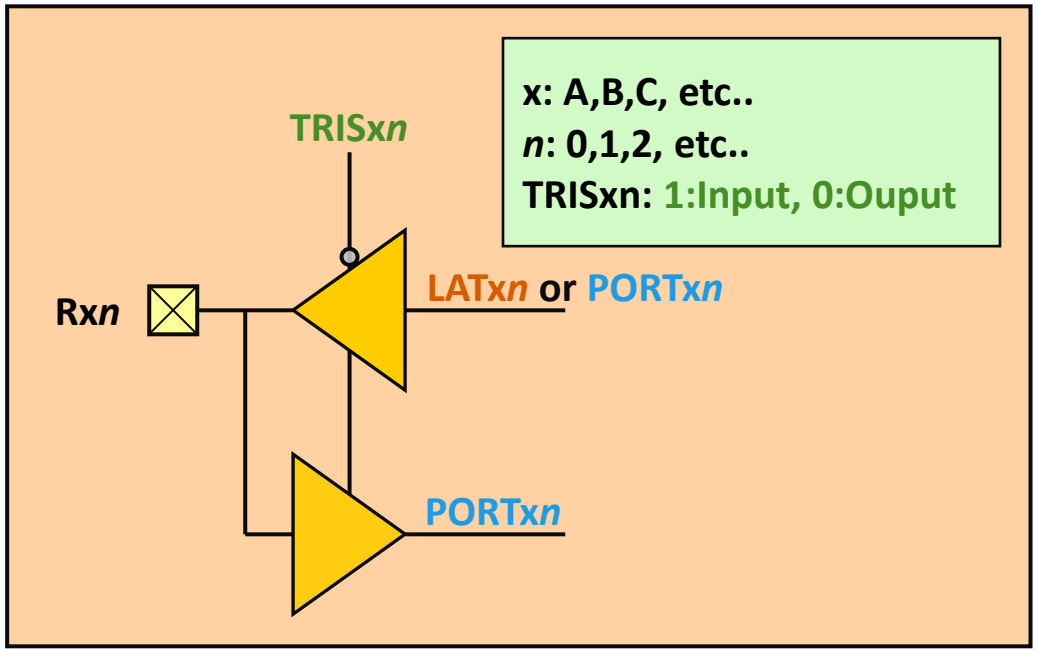
GPIO Architecture

GPIO Output

GPIO Block Diagram

- **Please refer to the block diagram, this is the concept for programming model. The real and detail block diagram please refer to the following slide.**
- **TRIS: To determine Input or Output.**
  1 for Input, 0 for Output.
- **LAT: set output drive value for Output pins.**
- **PORT: reaction logical level from Input pins.**





GPIO Input Peripheral Register

TRISx: PORTx Tri-state Register → 1:Input or 0:Output

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
TRISx7	TRISx6	TRISx5	TRISx4	TRISx3	TRISx2	TRISx1	TRISx0
bit 7							bit 0

LATx: PORTx Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATx7	LATx6	LATx5	LATx4	LATx3	LATx2	LATx1	LATx0
bit 7							bit 0

PORTx: PORTx Register → 1:High or 0:Low

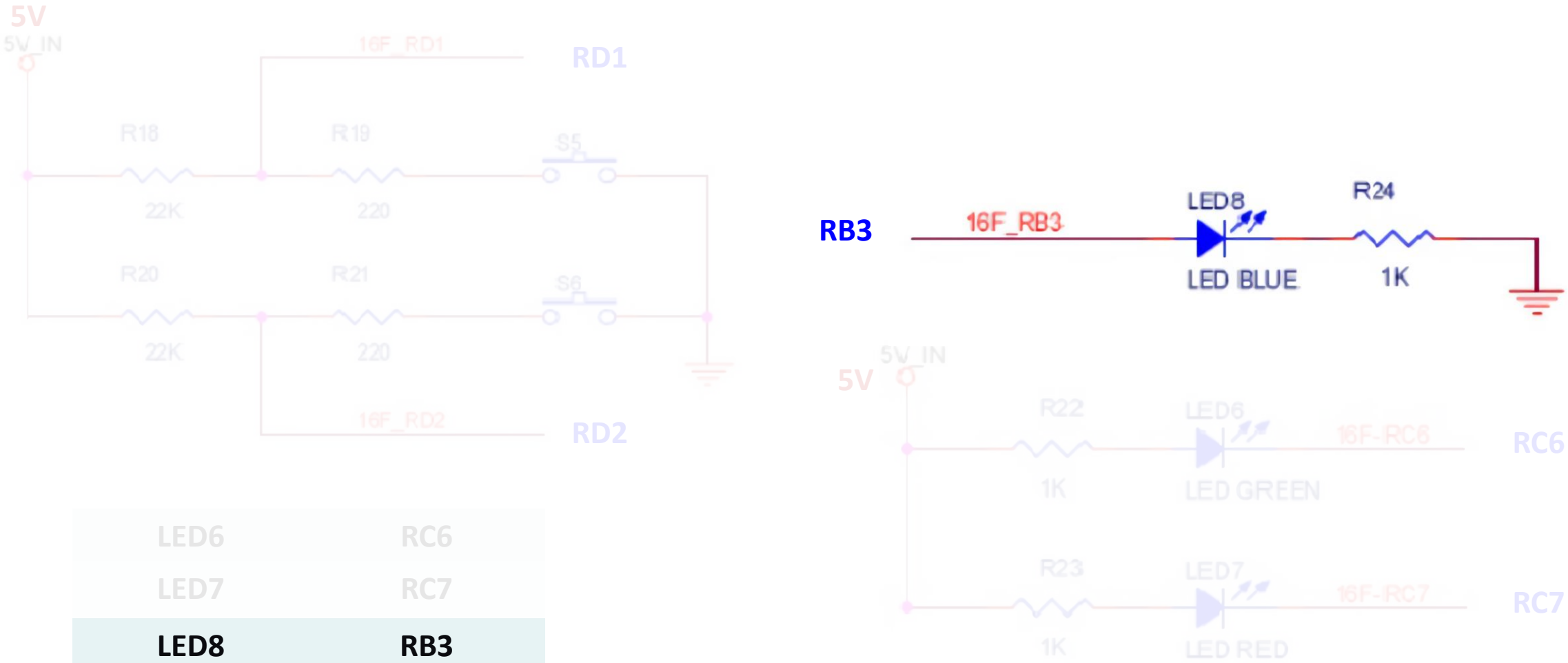
R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0
bit 7							bit 0

Lab1 GPIO Output

- Base on current project or Open .\Lab1.0_xxx.X to do exercises
- Try to control LED continues toggle (ON or OFF period ≈ 200mS).
- RB3 ▸ LED (LED8).

How to start ?

Check Schematic



LED6	RC6
LED7	RC7
LED8	**RB3**
S5	RD1
S6	RD2

→ 1:LED Light or 0:LED Dark

Lab1 GPIO Output

Step 1

a **Add code segment to your main**

```
main:
   // TODO 1.01
   BANKSEL TRISB
   bcf BANKMASK(TRISB),3

loop:
   // TODO 1.02
   ;Enable LED8
   BANKSEL LATB
   bsf BANKMASK(LATB),3
   movlw 25 ;25*8ms=200ms
   call delay_ms
   ;Disable LED8
   BANKSEL LATB
   bcf BANKMASK(LATB),3
   movlw 25 ;25*8ms=200ms
   call delay_ms

goto loop
```

Period Control

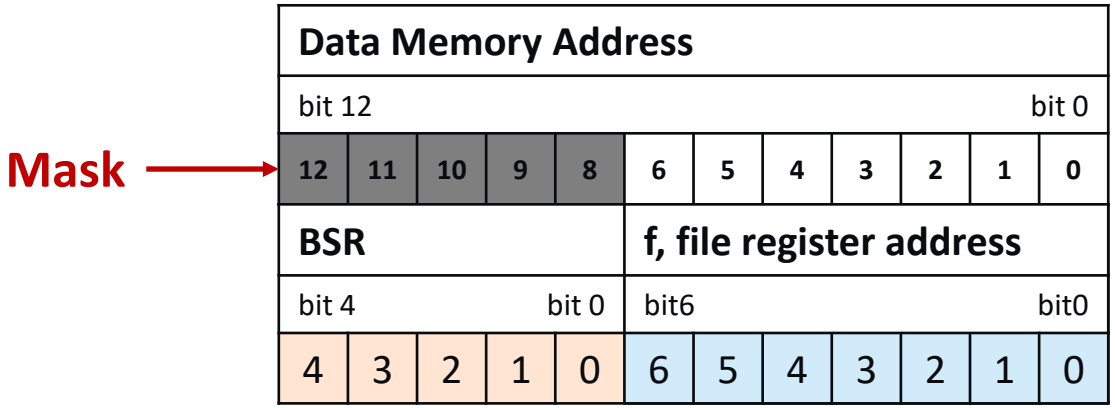
TRISB: PORTB Tri-state Register → 1:Input or 0:Ouput

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
bit 7							bit 0

0 → Output

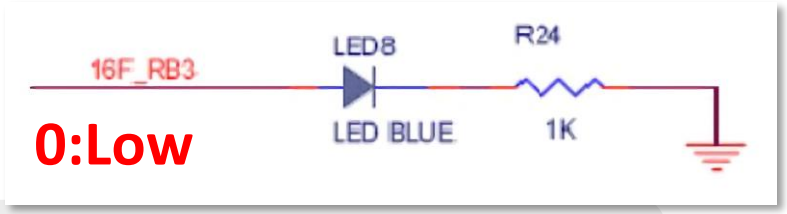
File Register Address Masking

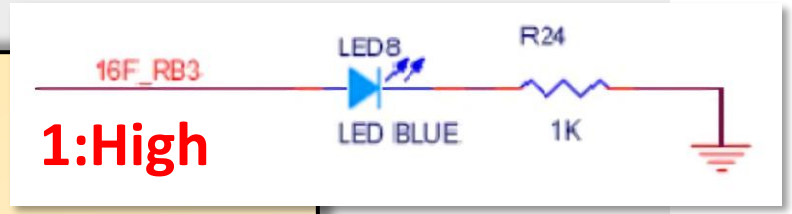
- **It is recommended that you mask all addresses used by PIC file register instructions whose operand represents a bank offset.**
- **The PIC Assembler will issue a fixup overflow error (for symbolic operands) or warning (for absolute operands) should it detect that there is superfluous bank information in a file register address operand.**
- **There are two common ways that you can mask the address operand.**
  - Mask out the bank information from the address using a bitwise AND (& operator), typically using the BANKMASK() macro.
  - Mask out the bank information from the address using a bitwise XOR (^ operator).



Lab1 GPIO Output

Step 2





a **Add code segment to your loop**

```
main:
   // TODO 1.01
   BANKSEL TRISB
   bcf BANKMASK(TRISB),3

loop:
   // TODO 1.02
   ;Enable LED8
   BANKSEL LATB
   bsf BANKMASK(LATB),3
   movlw 25 ;25*8ms=200ms
   call delay_ms
   ;Disable LED8
   BANKSEL LATB
   bcf BANKMASK(LATB),3
   movlw 25 ;25*8ms=200ms
   call delay_ms

goto loop
```

Period Control

LATB: PORTB Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0
bit 7				0 or 1			bit 0

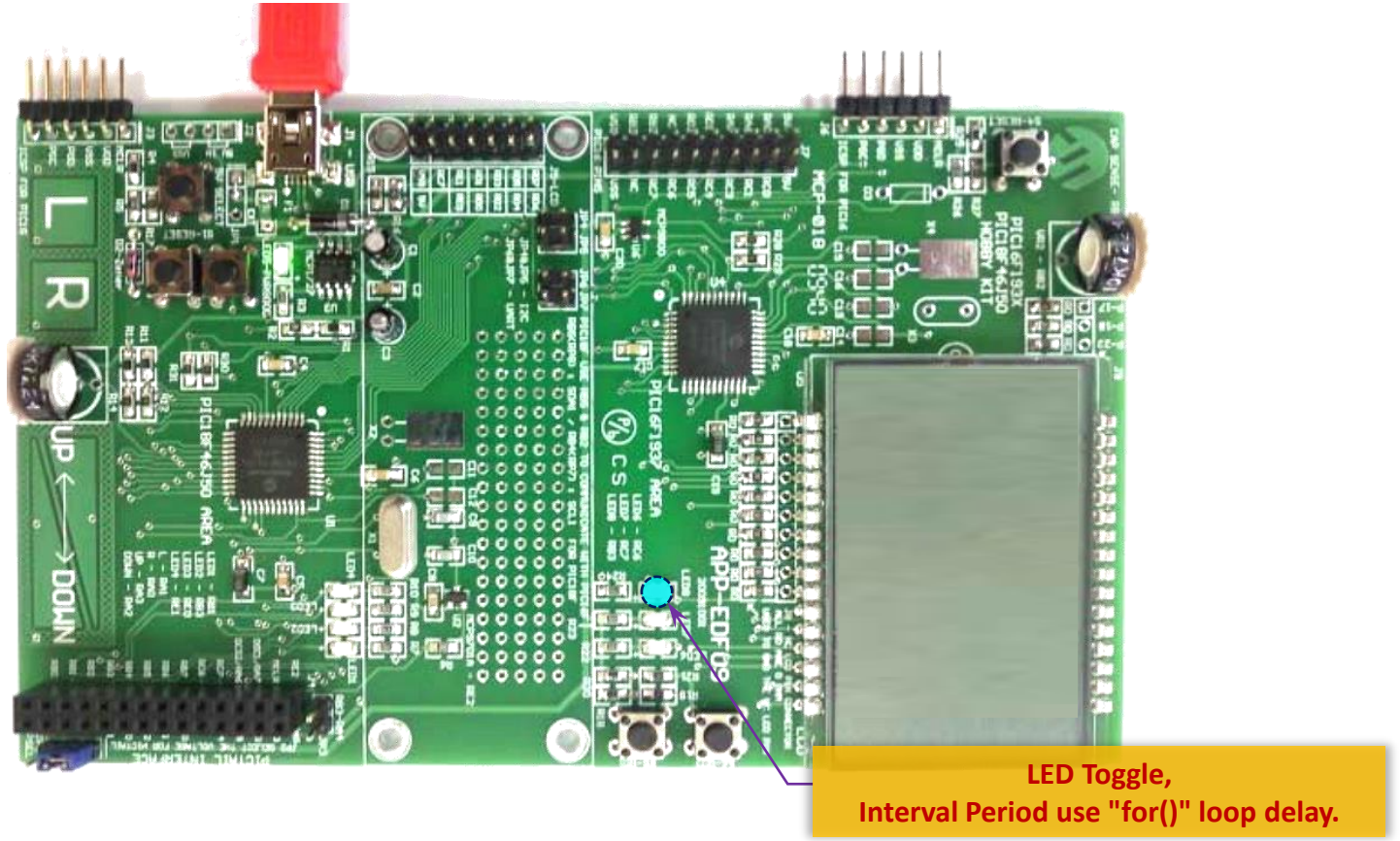
b **Program firmware to target board then observe result.**

MICROCHIP

Lab1 GPIO Output

Step 3 & Result

- Try program compiled firmware to your target board.
  - Select **Make and Program Device Main Project** icon 
  - Make sure program result ▸ **Programming/Verify complete**

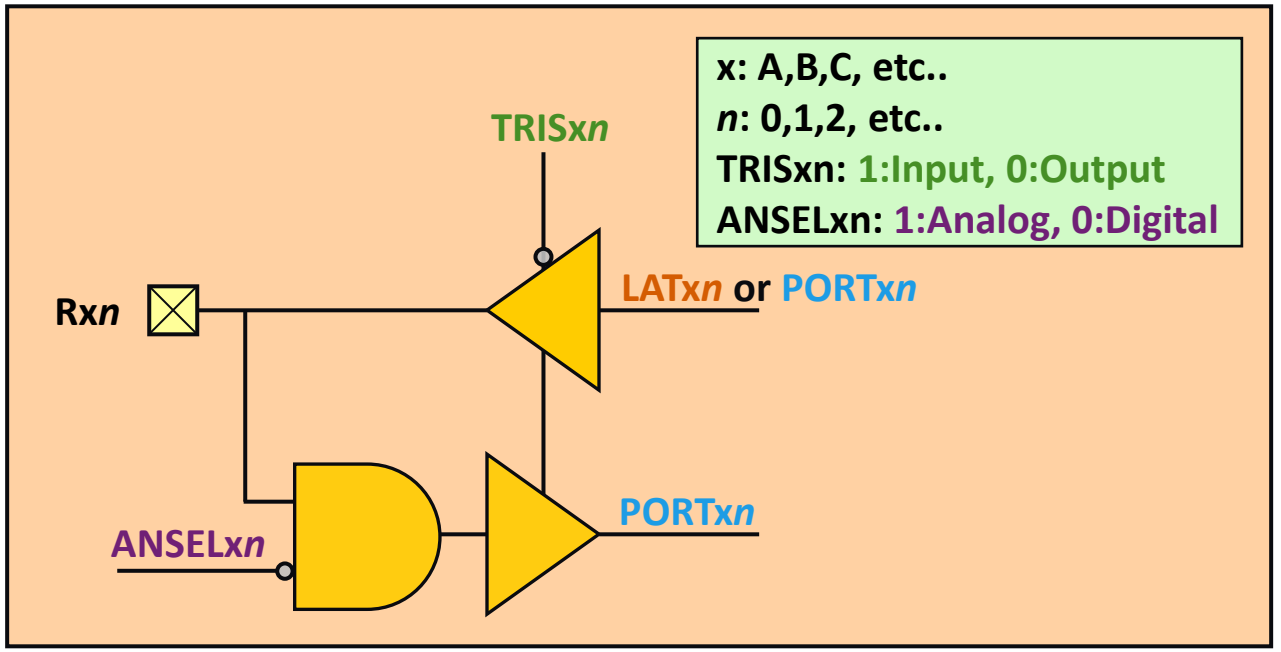


MICROCHIP

GPIO Input

GPIO Block Diagram

- Please refer to the block diagram, this is the concept for programming model. The real and detail block diagram please refer to the following slide.
- TRIS: To determine Input or Output.
  1 for Input, 0 for Output.
- LAT: set output drive
  value for Output pins.
- PORT: reaction logical
  level from Input pins.
- ANSEL: To determine
  Analog input or Digital I/O.
  1 for Analog, 0 for Digital.



GPIO Input Peripheral Register

TRISx: PORTx Tri-state Register → 1:Input or 0:Output

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
TRISx7	TRISx6	TRISx5	TRISx4	TRISx3	TRISx2	TRISx1	TRISx0
bit 7							bit 0

PORTx: PORTx Register → 1:Pin is > V_{IH} or 0:Pin is < V_{IL}

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0
bit 7							bit 0

ANSELx: PORTx Analog Select Register → 1:Analog or 0:Digital

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
ANSx7	ANSx6	ANSx5	ANSx4	ANSx3	ANSx2	ANSx1	ANSx0
bit 7							bit 0

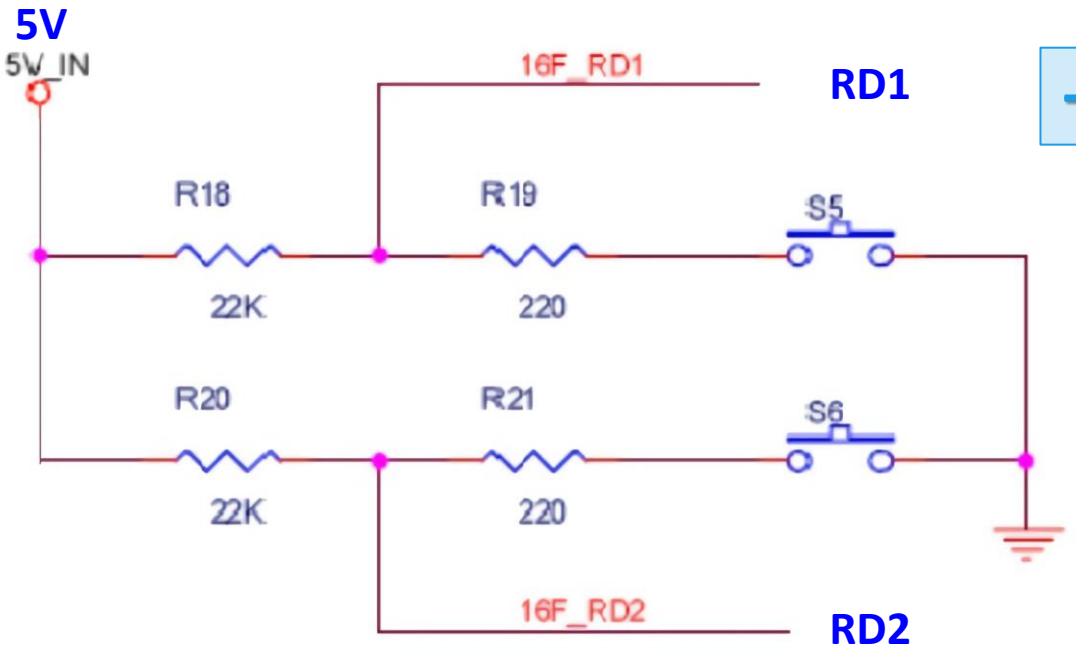
Lab2 GPIO Input

- Base on current project or Open .\Lab2.0_xxx.X to do exercises
- Try to set RD1, RD2 to digital input as get button status.
- Try use button to control LED light or dark.
  - S5 Pressed -> LED6 Light, S5 Released -> LED6 Dark
  - S6 Pressed -> LED7 Light, S6 Released -> LED7 Dark
- RB03 ▸ LED(LED1).
- RD1 ▸ Button(S5).
- RD2 ▸ Button(S6).
- RC6 ▸ LED(LED6).
- RC7 ▸ LED(LED7).

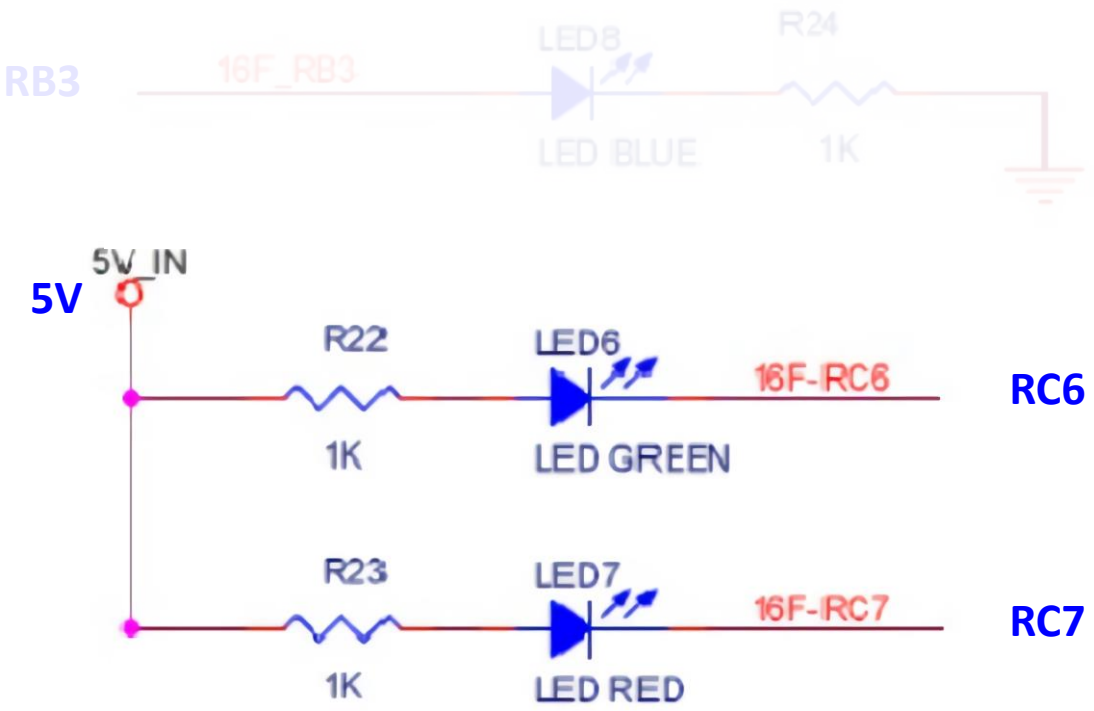
Let’s go!

MICROCHIP

Check Schematic



→ 0:Button Press or 1:Button Release



LED6	RC6
LED7	RC7
LED8	RB3
S5	RD1
S6	RD2

→ 0:LED Light or 1:LED Dark

MICROCHIP

Lab2 GPIO Input

Step 1

a **Add code segment below #include**

```
#include <xc.inc>

// TODO 2.01
#define LED6 BANKMASK(LATC),6
#define LED7 BANKMASK(LATC),7
#define S5 BANKMASK(PORTD),1
#define S6 BANKMASK(PORTD),2
...
```

LATC: PORTC Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7							bit 0

LED7 LED6

PORTD: PORTD Register → 1:Pin is > V_{IH} or 0:Pin is < V_{IL}

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
bit 7							bit 0

S6 S5

MICROCHIP

Lab2 GPIO Input

Step 2

a **Add code segment to your main**

```
main:
    ...
    // TODO 2.02
    BANKSEL TRISC
    bcf BANKMASK(TRISC),6
    bcf BANKMASK(TRISC),7
    BANKSEL LATC
    bsf LED6
    bsf LED7

    // TODO 2.03
    BANKSEL TRISD
    bsf BANKMASK(TRISD),1
    bsf BANKMASK(TRISD),2
    BANKSEL ANSELD
    bcf BANKMASK(ANSELD),1
    bcf BANKMASK(ANSELD),2
```

TRISC: PORTC Tri-state Register → 1:Input or 0:Output

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
bit 7							bit 0

0 0 → Output

LATC: PORTC Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7							bit 0

1 1 → LED Dark

TRISD: PORTD Tri-state Register → 1:Input or 0:Output

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0
bit 7							bit 0

1 1 → Input

ANSELD: PORTD Analog Select Register → 1:Analog or 0:Digital

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
ANSD7	ANSD6	ANSD5	ANSD4	ANSD3	ANSD2	ANSD1	ANSD0
bit 7							bit 0

0 0 → Digital

MICROCHIP

Lab2 GPIO Input

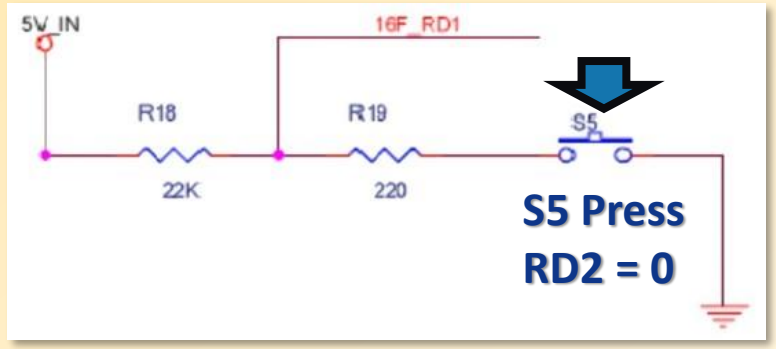
Step 3

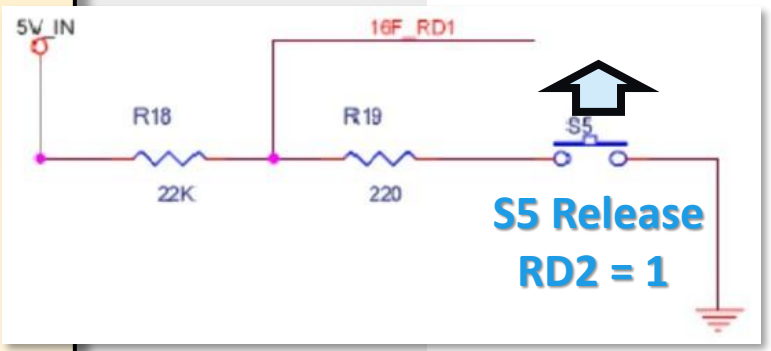
a Add code segment to your loop

```
loop:
   ...
   // TODO 2.04
   BANKSEL PORTD
   btfss S5
   goto LED6_Light
   goto LED6_Dark
LED6_Light:
   BANKSEL LATC
   bcf LED6
   goto LED6_END
LED6_Dark:
   BANKSEL LATC
   bsf LED6
   goto LED6_END
LED6_END:

goto loop
```

PORTD: PORTD Register → 1:Pin is > V_{IH} or 0:Pin is < V_{IL}

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
bit 7						S5	bit 0





MICROCHIP

Lab2 GPIO Input

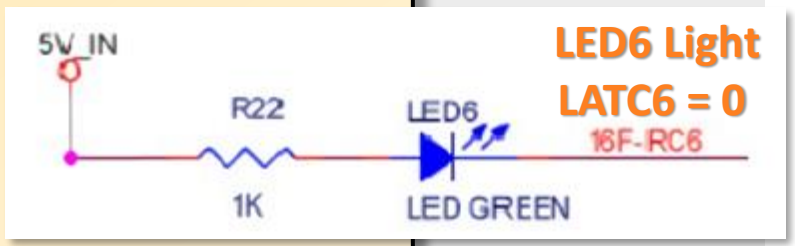
Step 3

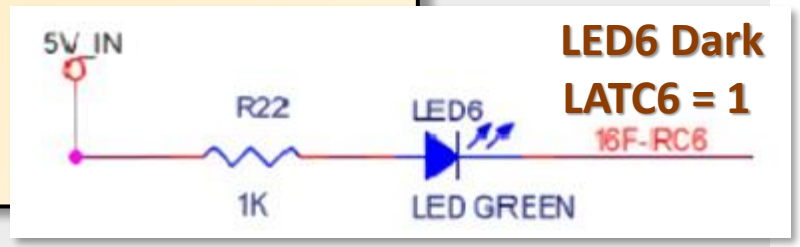
a **Add code segment to your loop**

```
loop:
    ...
    // TODO 2.04
    BANKSEL PORTD
    btfss S5
    goto LED6_Light
    goto LED6_Dark
LED6_Light:
    BANKSEL LATC
    bcf LED6
    goto LED6_END
LED6_Dark:
    BANKSEL LATC
    bsf LED6
    goto LED6_END
LED6_END:

goto loop
```

LATC: PORTC Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7	LED6						bit 0





MICROCHIP

Lab2 GPIO Input

Step 3

a **Add code segment to your loop**

```
loop:
    ...
    // TODO 2.04
    BANKSEL PORTD
    btfss S5
    goto LED6_Light
    goto LED6_Dark
LED6_Light:
    BANKSEL LATC
    bcf LED6
    goto LED6_END
LED6_Dark:
    BANKSEL LATC
    bsf LED6
    goto LED6_END
LED6_END:

goto loop
```

PORTD: PORTD Register → 1:Pin is > V_{IH} or 0:Pin is < V_{IL}

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
bit 7						S5	bit 0

LATC: PORTC Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7	LED6						bit 0

MICROCHIP

Lab2 GPIO Input

Step 4

a **Add code segment to your loop**

```
loop:
    ...
    // TODO 2.05
    BANKSEL PORTD
    btfss S6
    goto LED7_Light
    goto LED7_Dark
LED7_Light:
    BANKSEL LATC
    bcf LED7
    goto LED7_END
LED7_Dark:
    BANKSEL LATC
    bsf LED7
    goto LED7_END
LED7_END:

goto loop
```

PORTD: PORTD Register → 1:Pin is > V_{IH} or 0:Pin is < V_{IL}

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
bit 7					S6		bit 0

LATC: PORTC Data Latch Register → 1:High or 0:Low

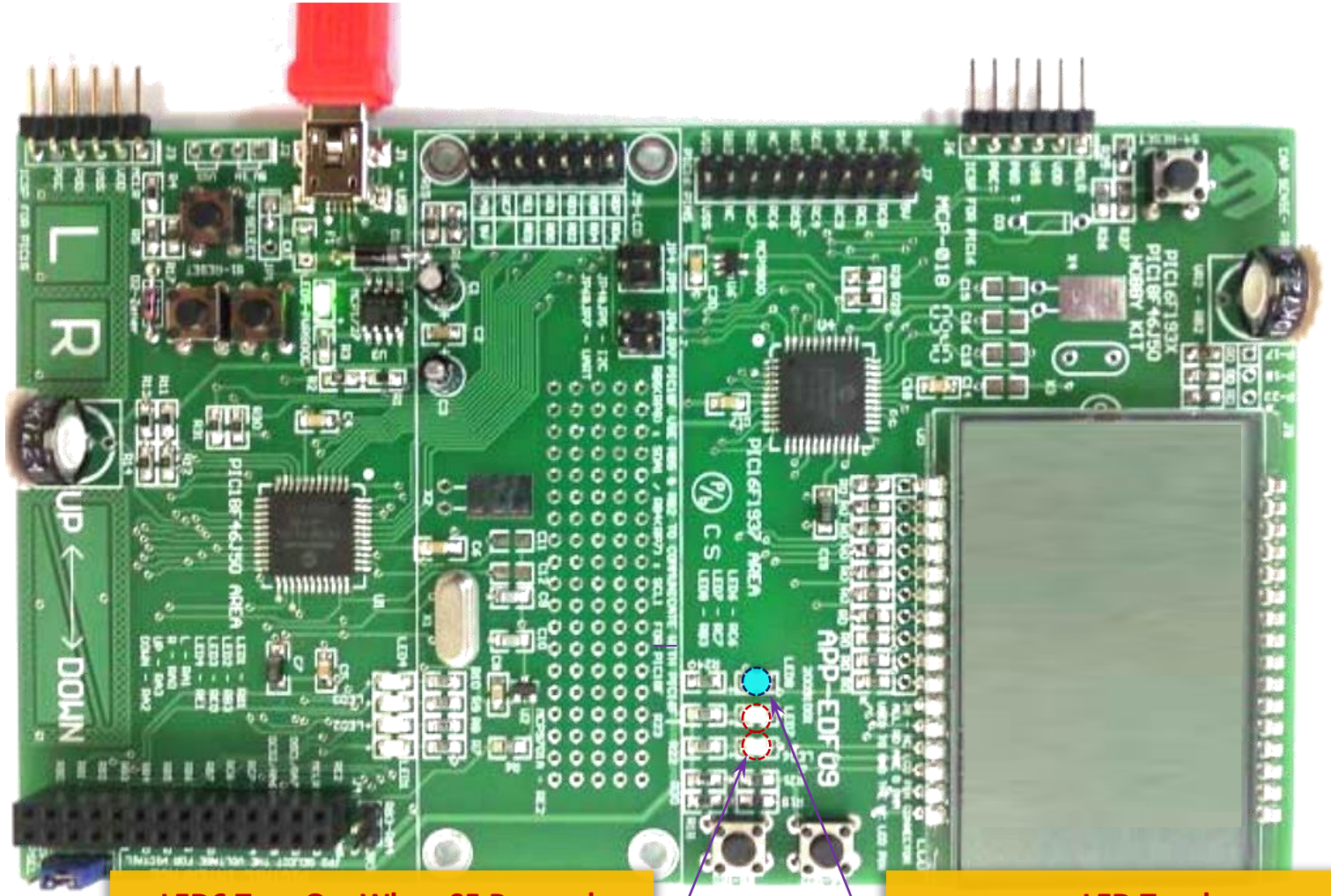
R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7 LED7							bit 0

b **Program firmware to target board then observe result.**

MICROCHIP

Lab2 GPIO Input

Result



LED6 Turn On, When S5 Pressed.
LED7 Turn On, When S6 Pressed.

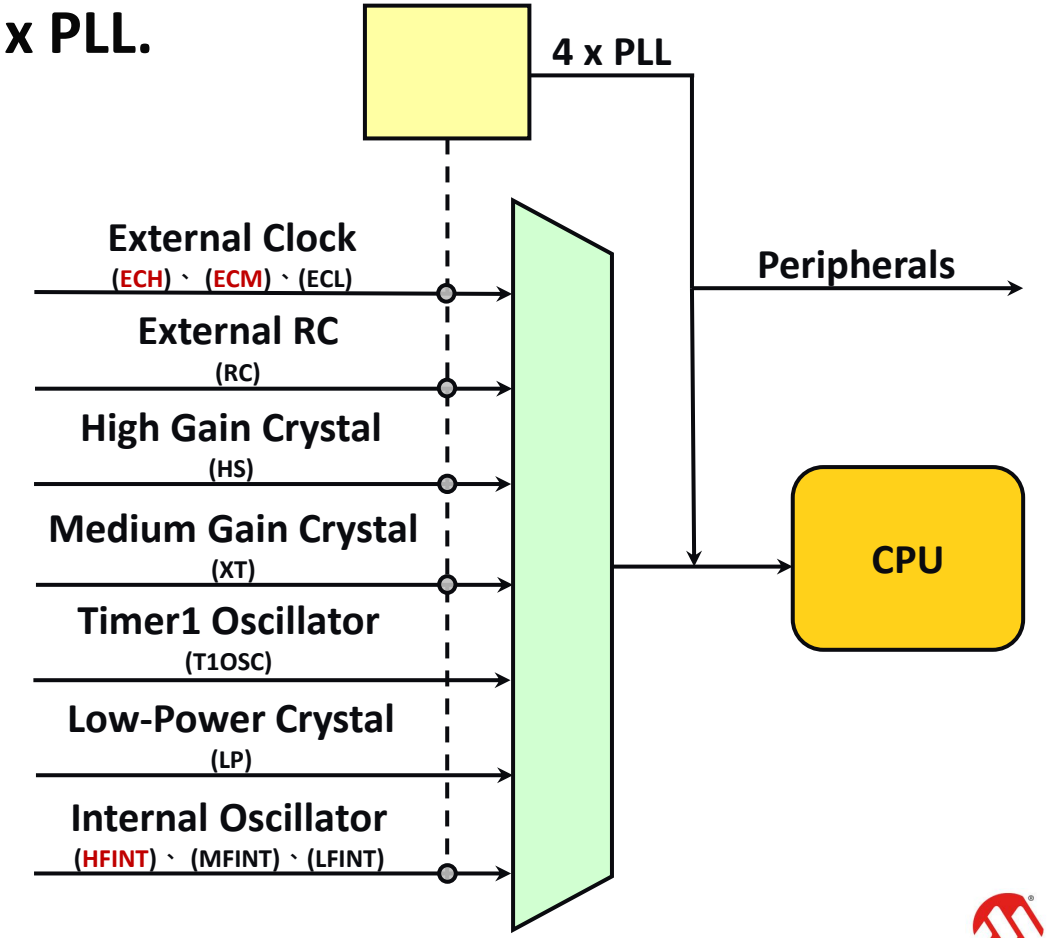
LED Toggle,
Interval Period use "for()" loop delay.

MICROCHIP

Oscillator Architecture

PIC16F Series Clocks

- **Microchip PIC16F1937 MCU available clock sources are Internal Oscillator, Low Power Crystal and External Clock and RC or etc..**
- **Software-controllable switching between various clock sources.**
- **The clocks can be divided or multiply by 4 x PLL.**
  - ECL：0MHz to 0.5MHz
  - ECM：0.5MHz to 4MHz
  - ECH：4MHz to 8MHz
  - LP : 32kHz
  - XT：up to 4MHz
  - HS： 4MHz to 20MHz
  - INTOSC：31kHz to 32MHz

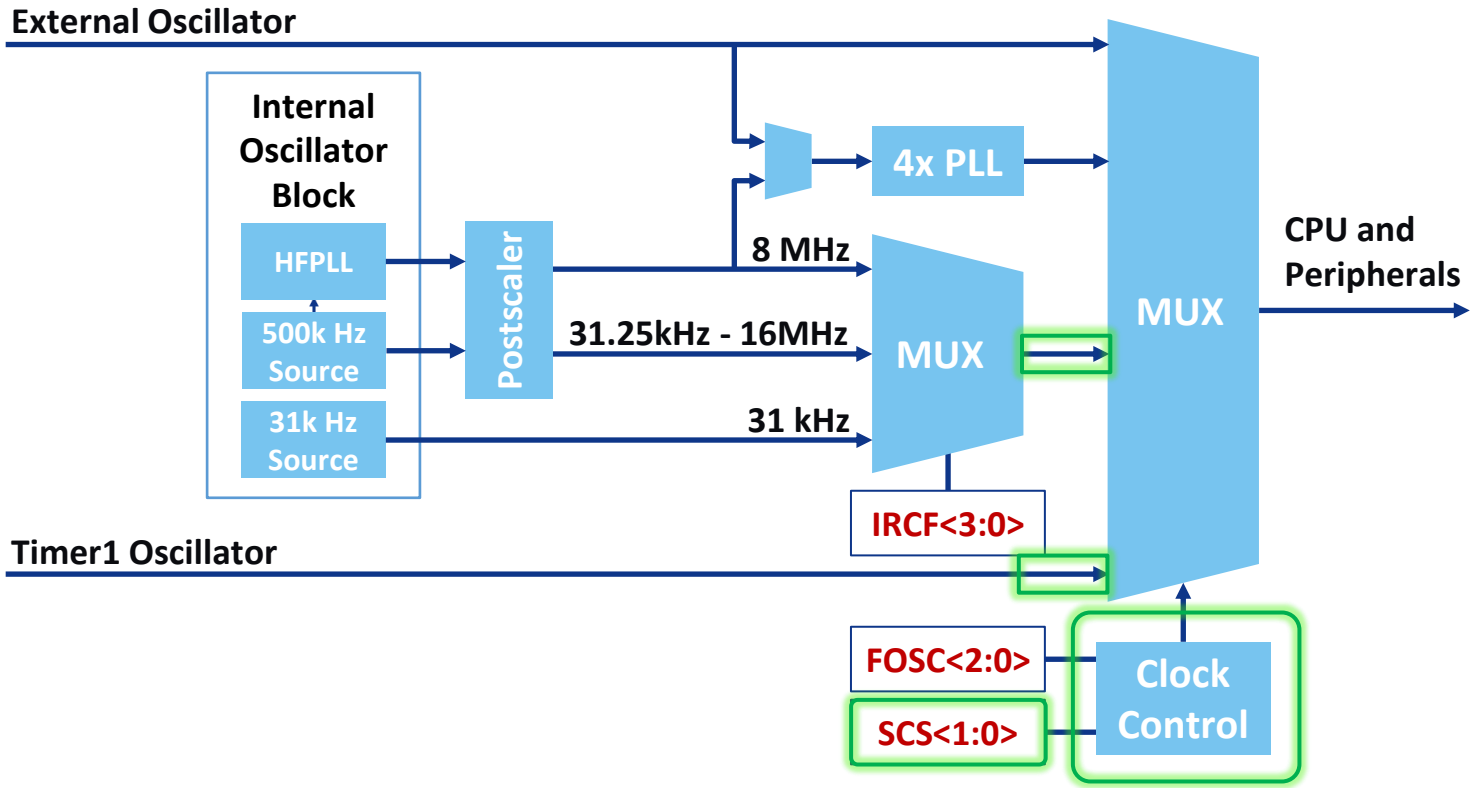


Oscillator Setting

- **System Clock Select by SCS<1:0> in OSCCON.**

OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0/0	R/W-0/0	R/W-1/1	R/W-1/1	R/W-1/1	U-0	R/W-0/0	R/W-0/0
SPLLEN	IRCF<3:0>				-	SCS<1:0>	
bit 7							bit 0



SCS<1:0>: System Clock Select bits

1x = Internal oscillator block

01 = Timer1 oscillator

00 = Clock determined by FOSC<2:0>

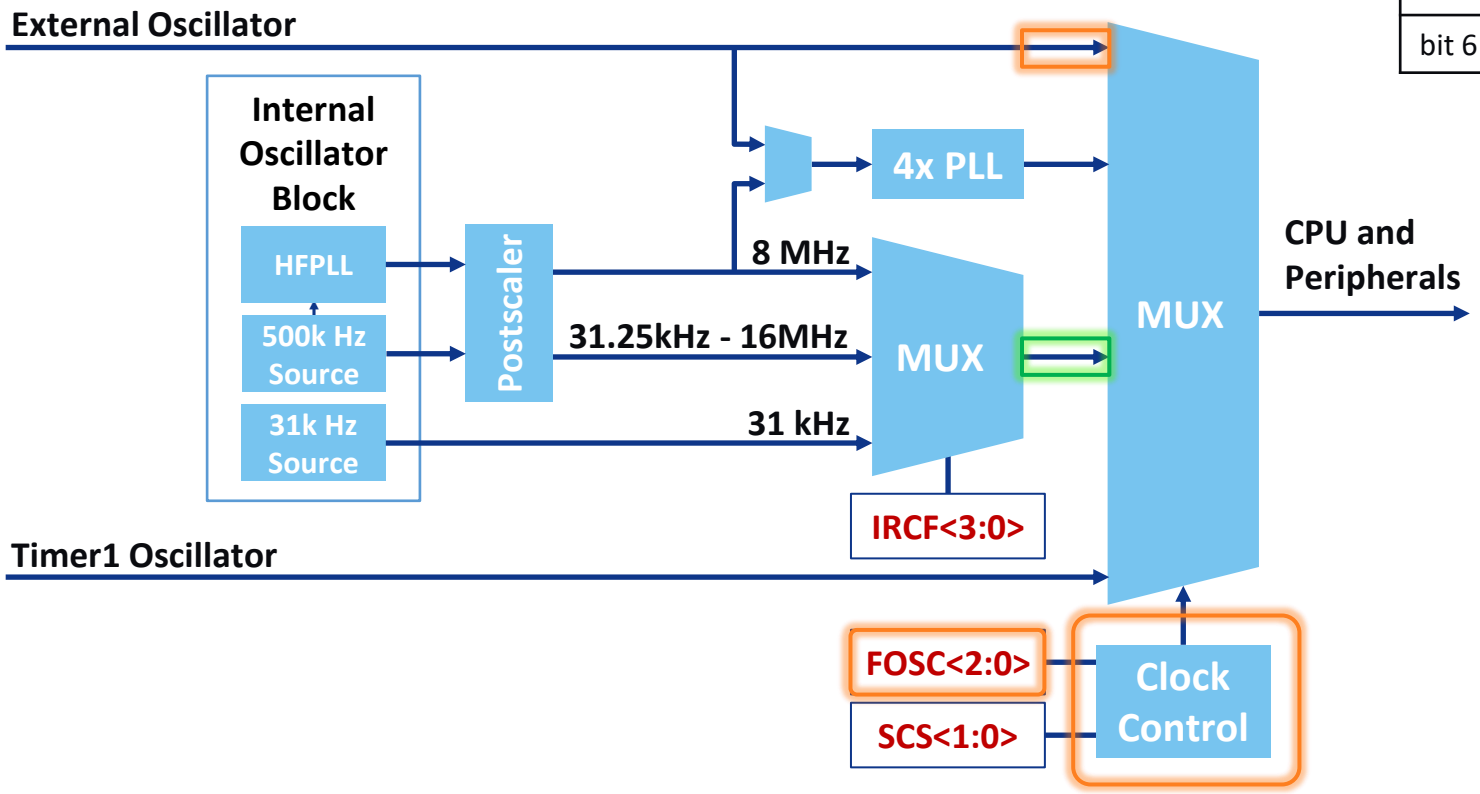
MICROCHIP

Oscillator Setting

- **Oscillator Select by FOSC<2:0> in Configuration Word 1.**

CONFIGURATION WORD 1

R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1
FCMEN	IESO	nCLKOUTEN	BOREN1	BOREN0	nCPD	nCP
bit 13						
R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1
MCLRE	nPWRTE	WDTE1	WDTE0	FOSC2	FOSC1	FOSC0
bit 6						bit 0



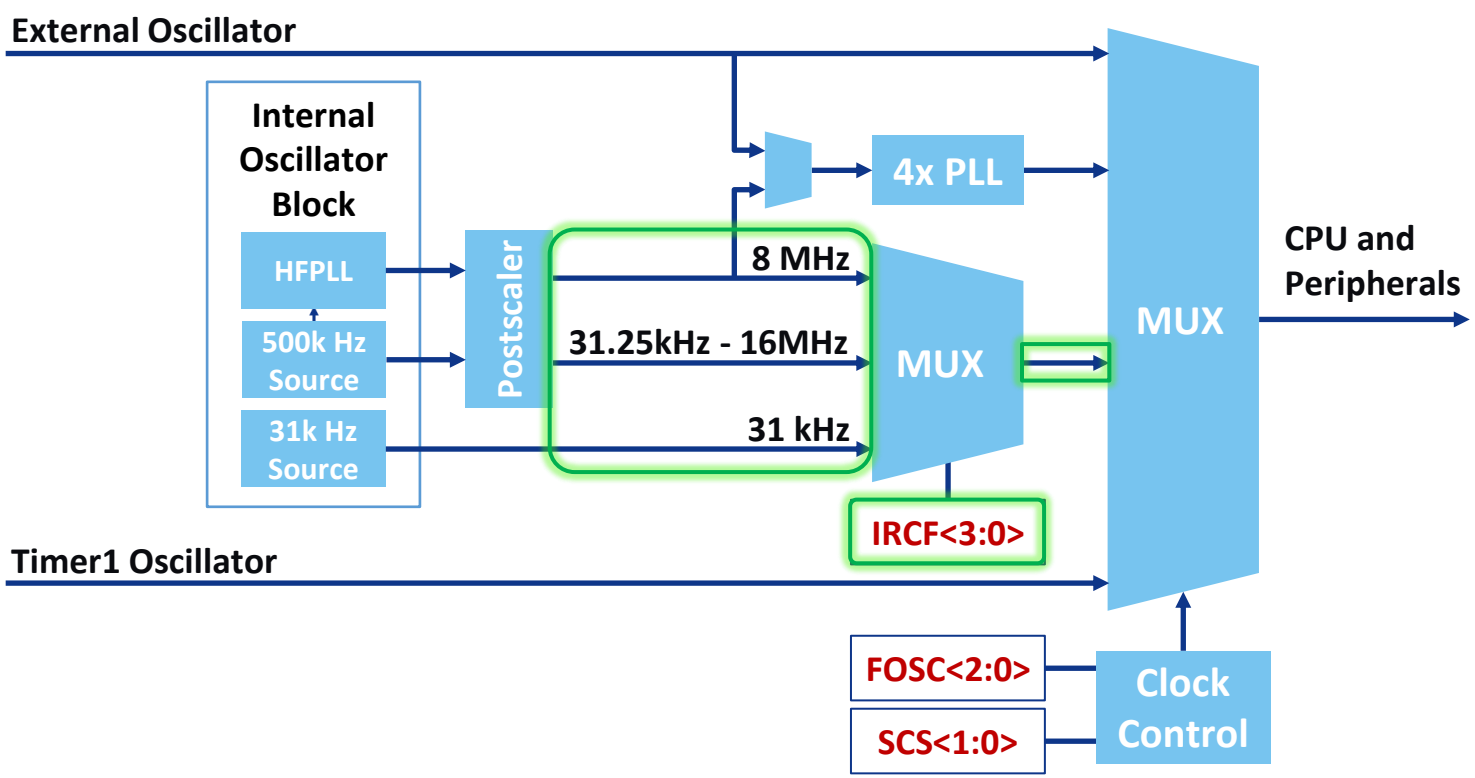
FOSC<2:0>: Oscillator Selection bits

111 = ECH: External Clock, High-Power mode
110 = ECM: External Clock, Medium-Power mode
101 = ECL: External Clock, Low-Power mode
100 = INTOSC oscillator
011 = EXTRC oscillator
010 = HS oscillator: High-speed crystal/resonator
001 = XT oscillator: Crystal/resonator
000 = LP oscillator: Low-power crystal

MICROCHIP

Oscillator Setting

- **Internal Oscillator Frequency Select by IRCF<3:0> in OSCCON.**



OSCCON: OSCILLATOR CONTROL REGISTER

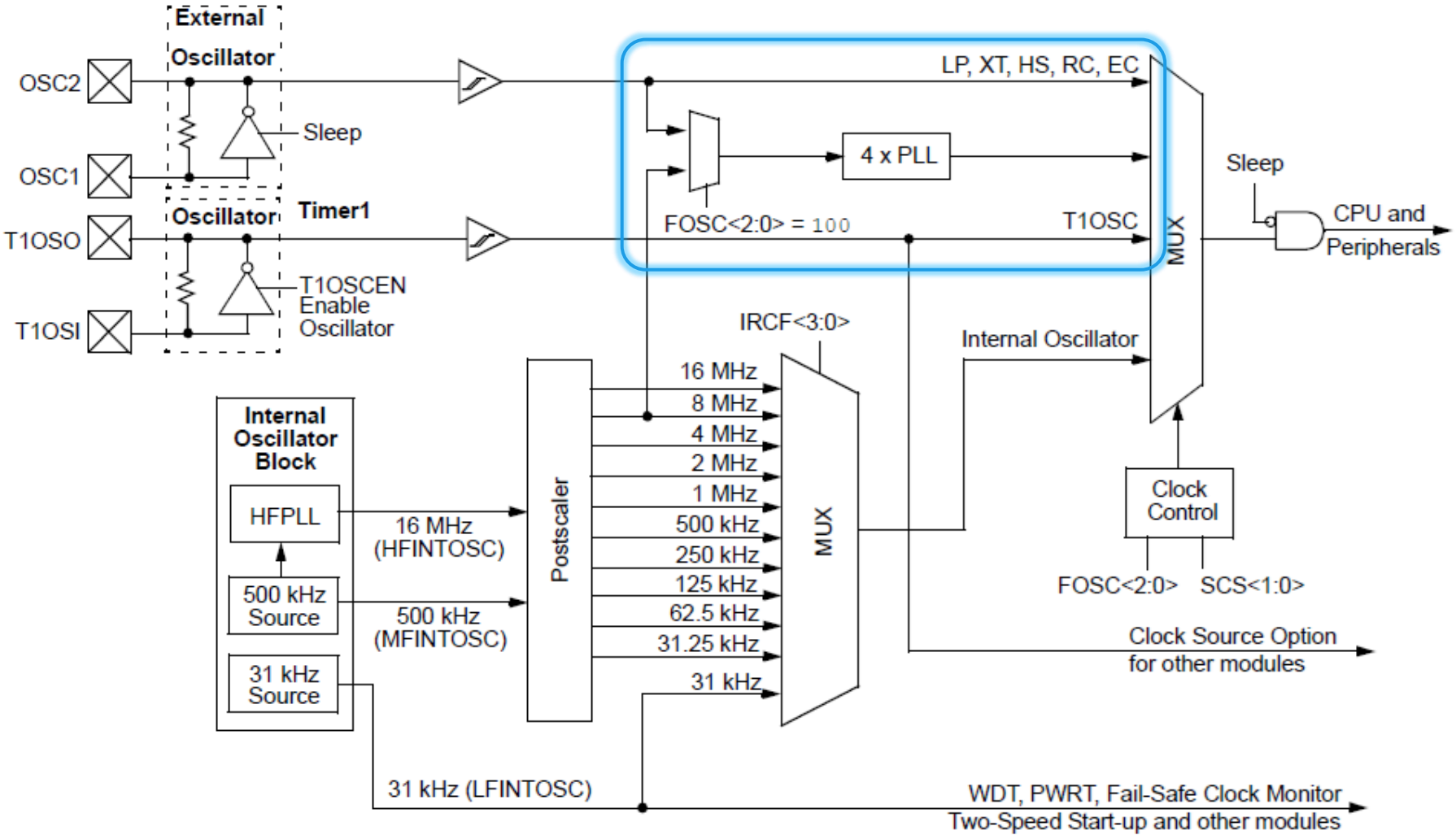
R/W-0/0	R/W-0/0	R/W-1/1	R/W-1/1	R/W-1/1	U-0	R/W-0/0	R/W-0/0
SPLLEN	IRCF<3:0>				-	SCS<1:0>	
bit 7							bit 0

IRCF<3:0>: Internal Oscillator Frequency Select bits

000x = 31 kHz LF
0010 = 31.25 kHz MF
0011 = 31.25 kHz HF
0100 = 62.5 kHz MF
0101 = 125 kHz MF
0110 = 250 kHz MF
0111 = 500 kHz MF (default upon Reset)
1000 = 125 kHz HF
1001 = 250 kHz HF
1010 = 500 kHz HF
1011 = 1 MHz HF
1100 = 2 MHz HF
1101 = 4 MHz HF
1110 = 8 MHz or 32 MHz HF
1111 = 16 MHz HF

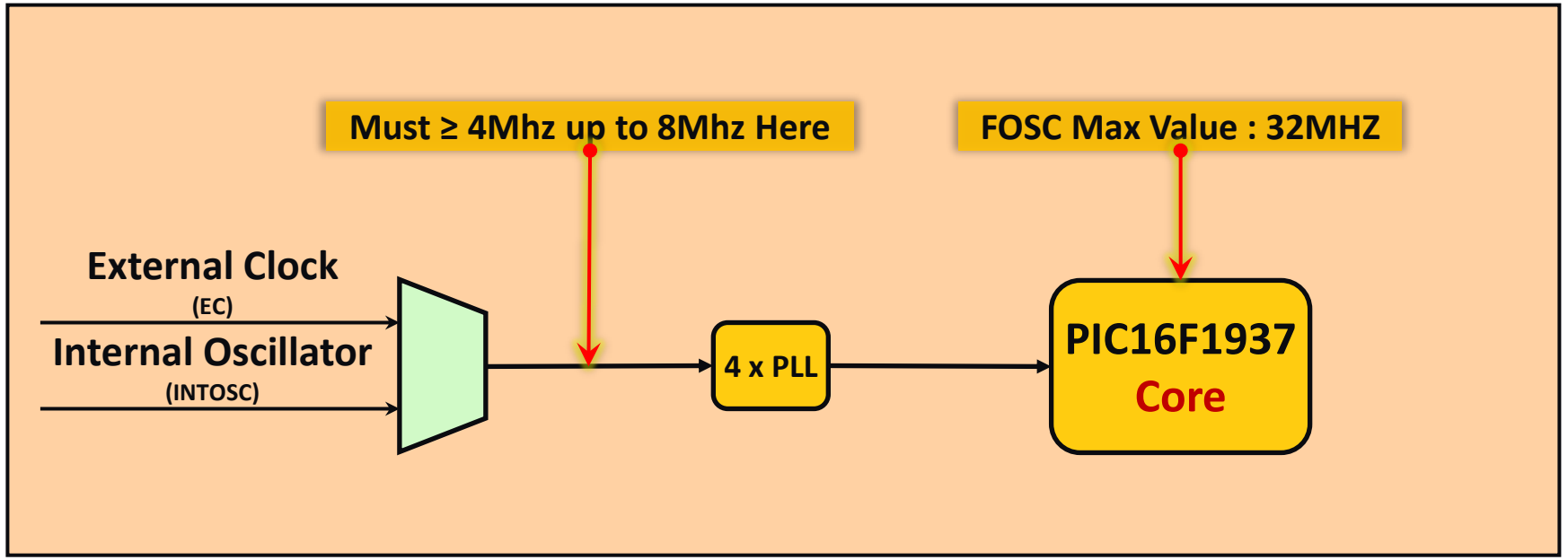
Phase Locked Loop

- **A block diagram of the PLL module.**



Phase Locked Loop

- Clock input must 4MHz to 8MHz after PLLPRE, which can use External Clock (Crystal or External Clock) or Internal Fast RC as clock source.



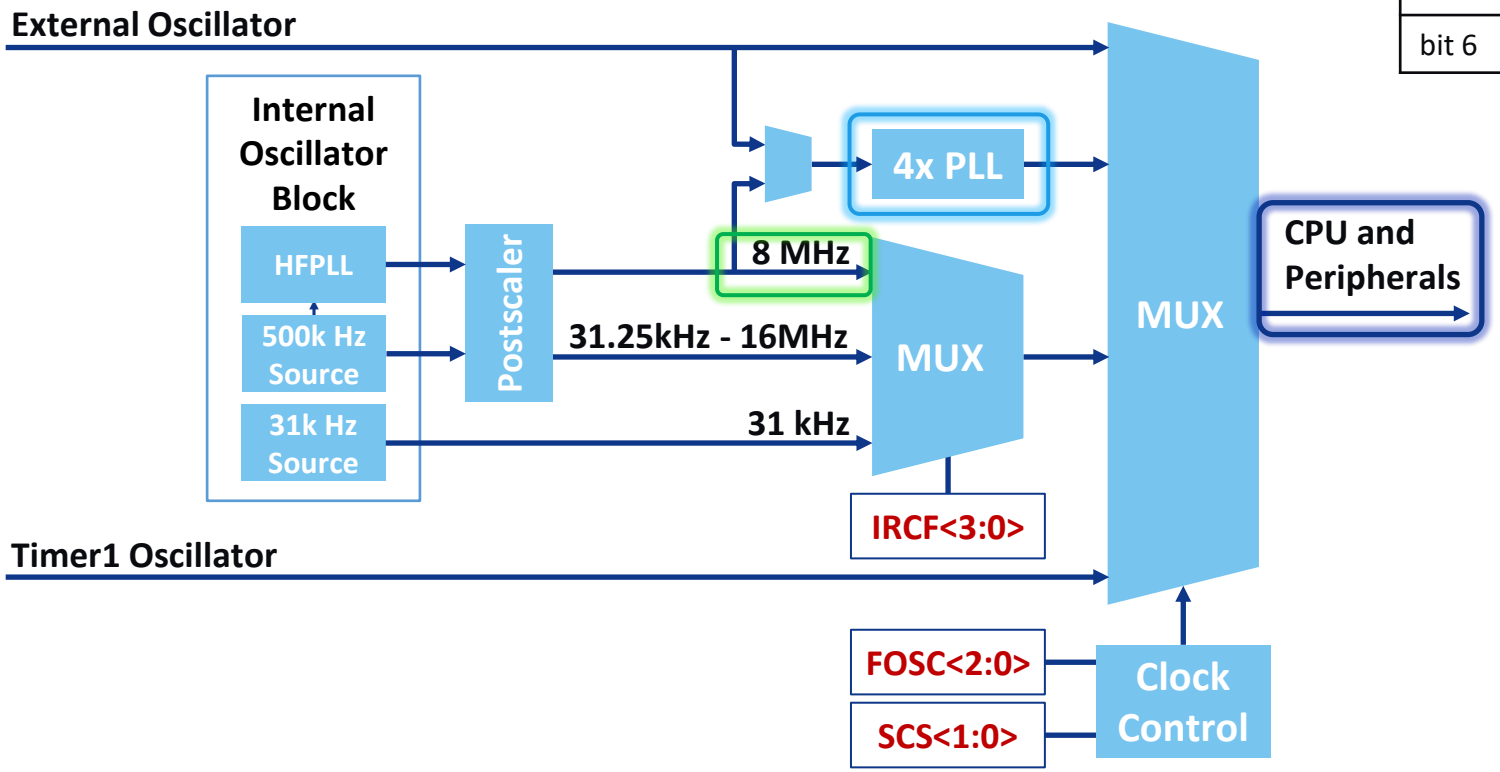
Oscillator PLL Setting

- **PLL Enable by PLLEN bit in Configuration Word 2.**

CONFIGURATION WORD 2

R/P-1/1	R/P-1/1	U-1	R/P-1/1	R/P-1/1	R/P-1/1	U-1
LVP	nDEBUG	-	BORV	TVREN	PLLEN	-
bit 13						

U-1	R/P-1/1	R/P-1/1	U-1	U-1	R/P-1/1	R/P-1/1
-	VCAPEN<1:0>		-	-	WRT1	WRT0
bit 6						bit 0



System Clock
= 8MHz × (4× PLL) = 32MHz

When use Internal Oscillator and 4× PLL, must be set to the 8MHz.

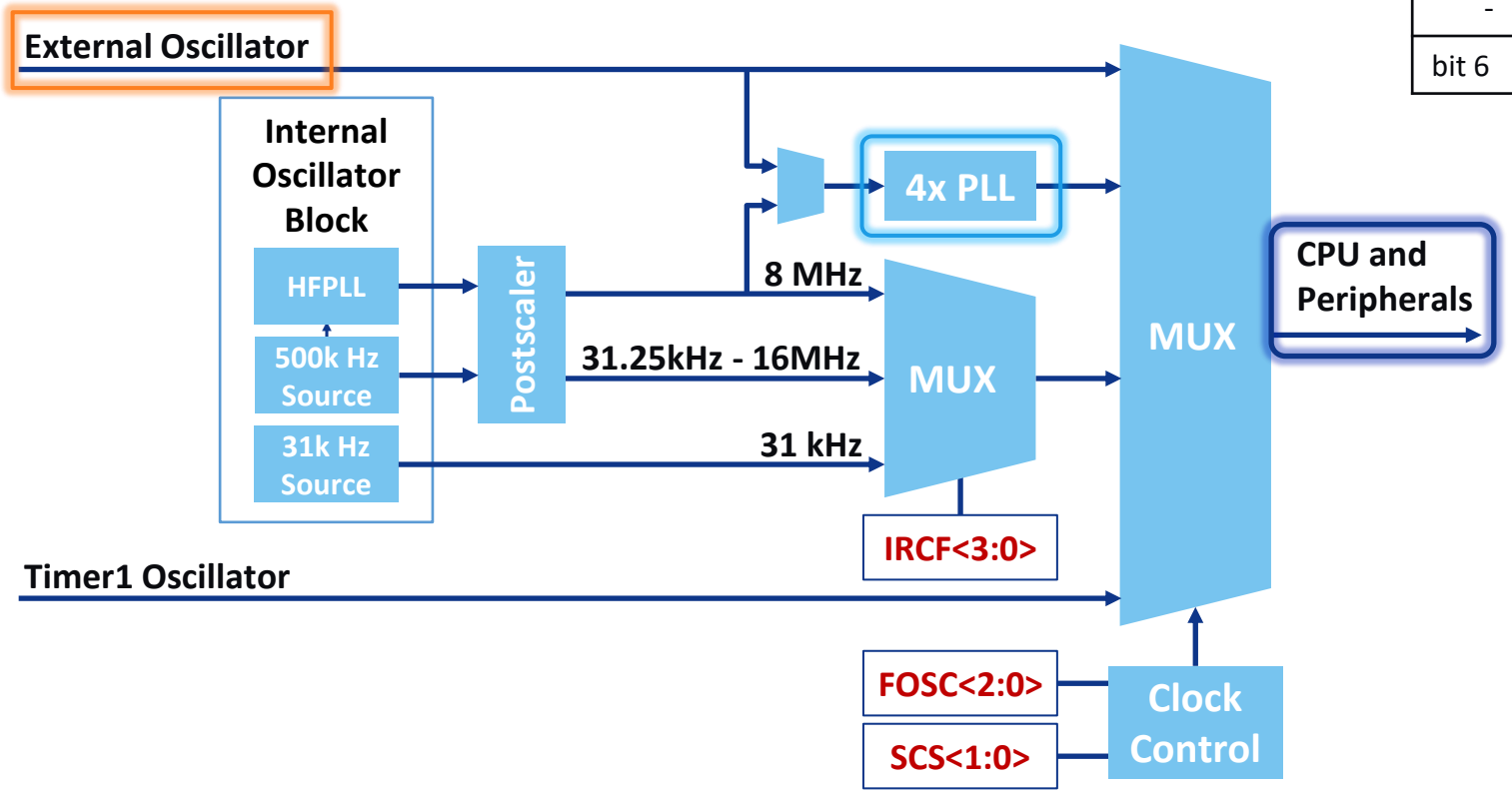
Oscillator PLL Setting

- **PLL Enable by PLLEN bit in Configuration Word 2.**

CONFIGURATION WORD 2

R/P-1/1	R/P-1/1	U-1	R/P-1/1	R/P-1/1	R/P-1/1	U-1
LVP	nDEBUG	-	BORV	TVREN	PLLEN	-
bit 13						

U-1	R/P-1/1	R/P-1/1	U-1	U-1	R/P-1/1	R/P-1/1
-	VCAPEN<1:0>		-	-	WRT1	WRT0
bit 6						bit 0



System Clock(External)
= 4MHz to 8MHz × (4× PLL)

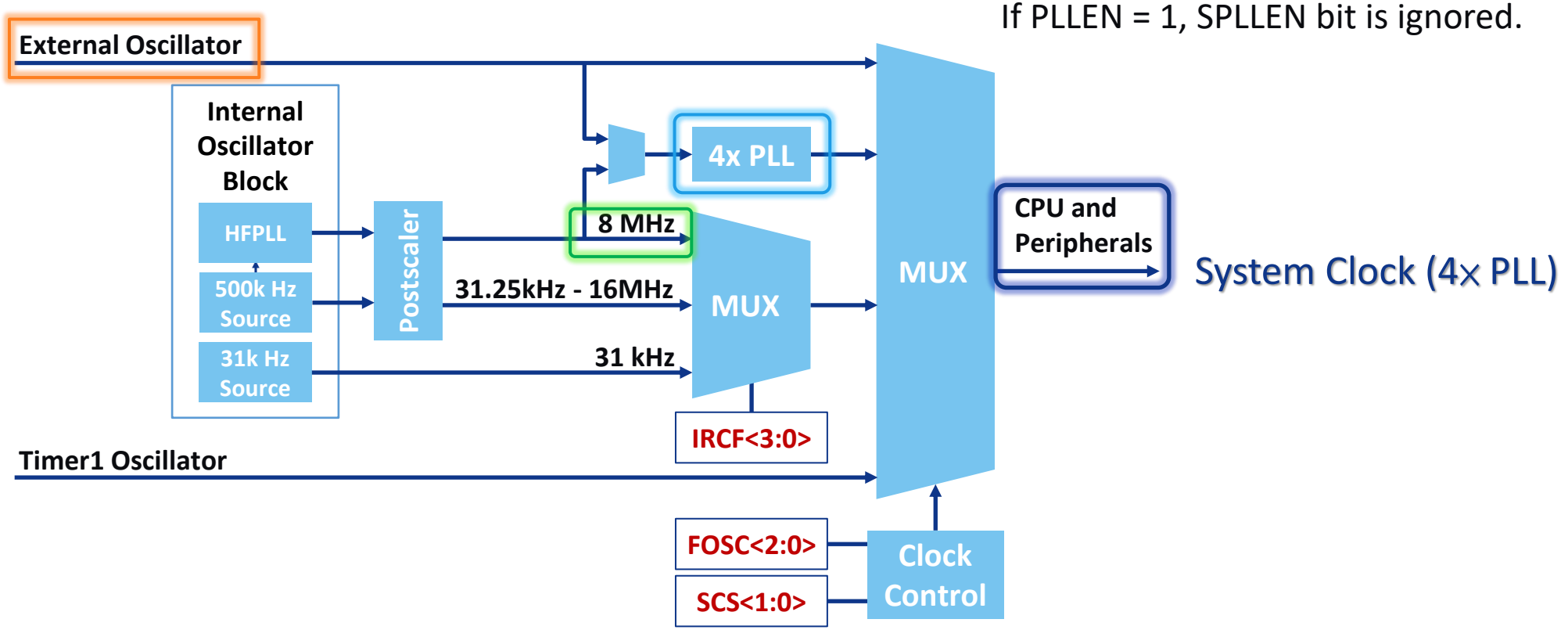
Oscillator PLL Setting

- **PLL Enable by SPLLEN bit in OSCCON when PLLEN bit in Configuration Word 2 is set 0.**

OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0/0	R/W-0/0	R/W-1/1	R/W-1/1	R/W-1/1	U-0	R/W-0/0	R/W-0/0
SPLLEN	IRCF<3:0>				-	SCS<1:0>	
bit 7							bit 0

If PLLEN = 1, SPLLEN bit is ignored.



Lab3 Clock Setting

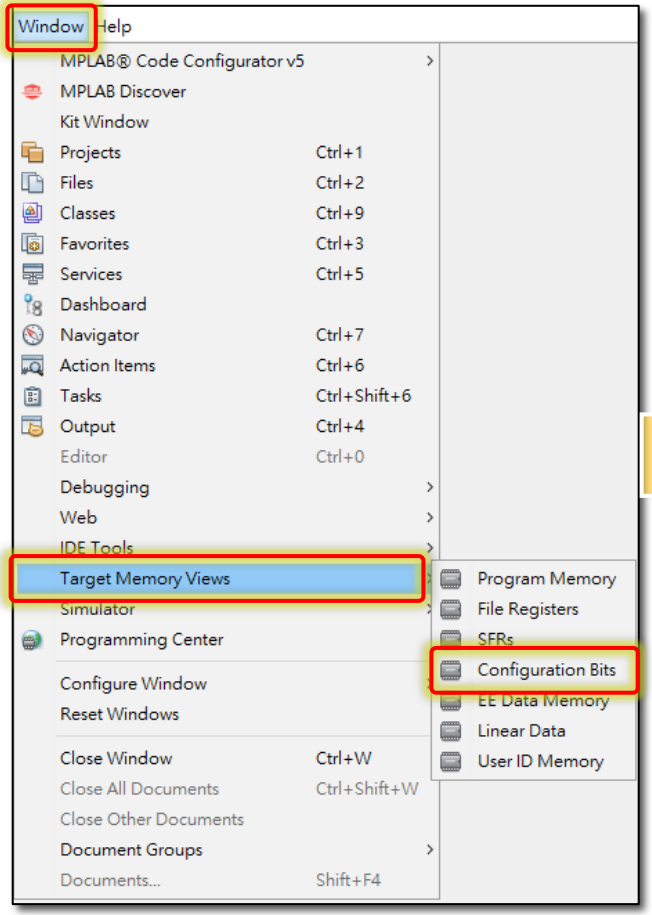
- Base on current project or Open . \Lab3.0_xxx.X to do exercises
- Try provide maximum frequency (4MHz) to MCU.

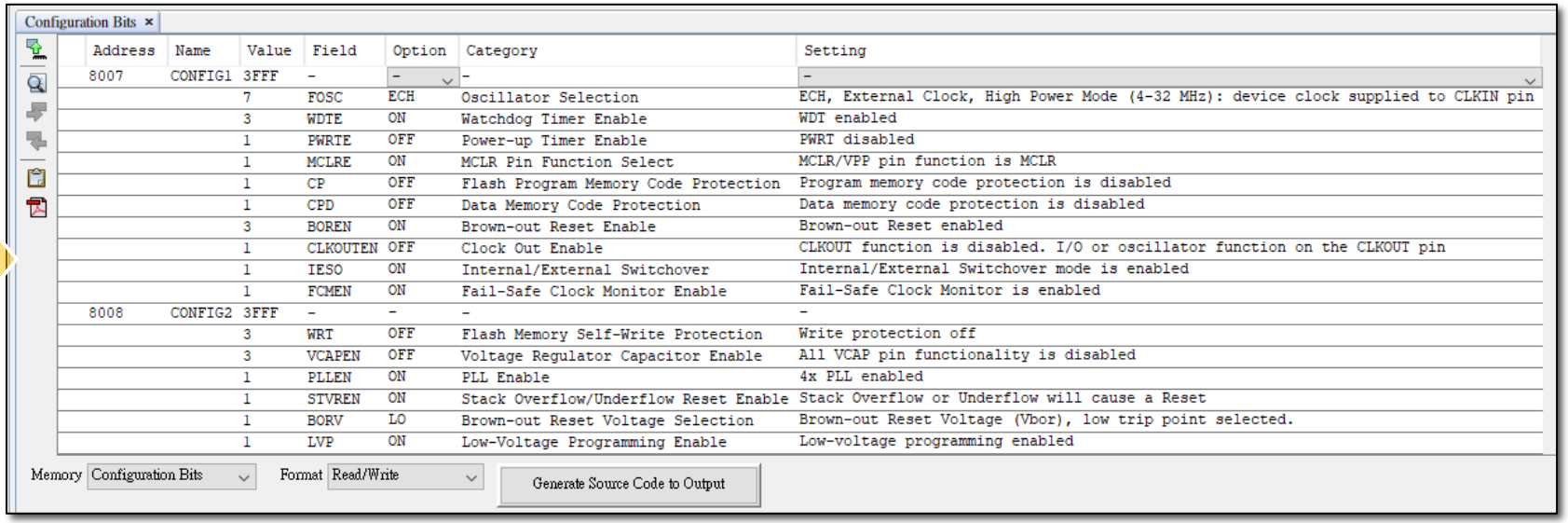
Let’s go!

Configuration Bits Window

Step 1 (Optional)

- **Create a main.c file in the project.**
  Menu Window ▸ Target Memory Views ▸ Configuration Bits

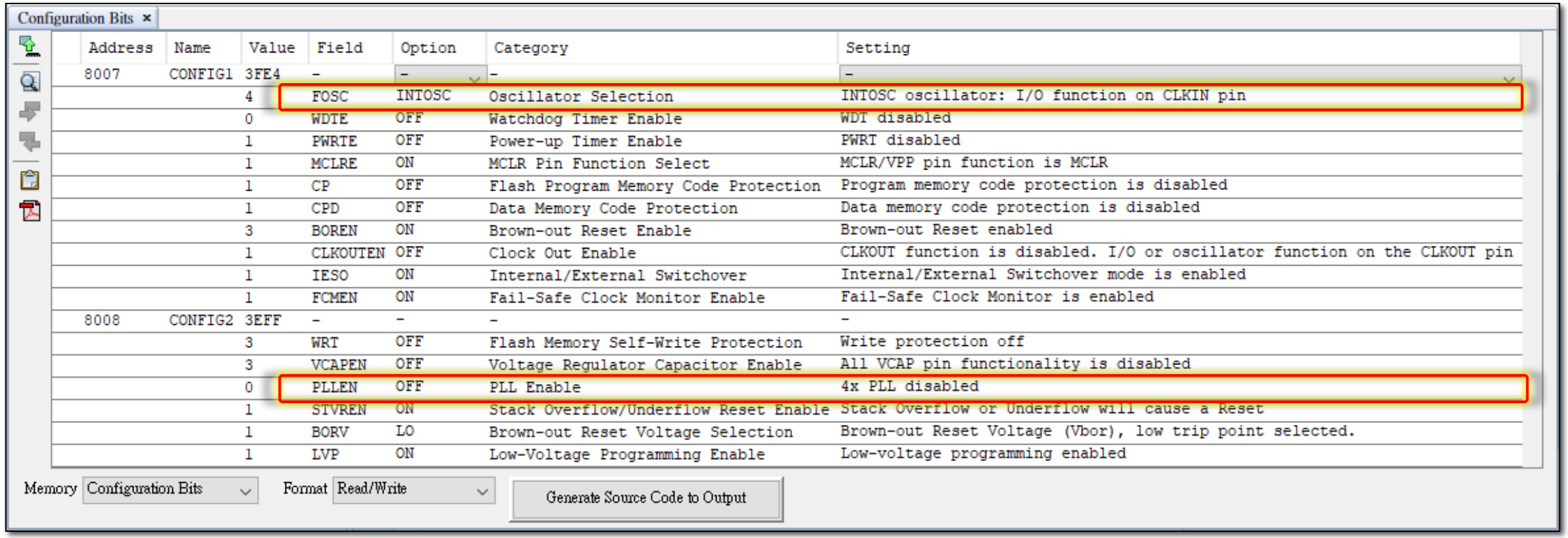




Configuration Bits Window

Step 2 (Optional)

- **Create a main.c file in the project.**
  Menu Window ▸ Target Memory Views ▸ Configuration



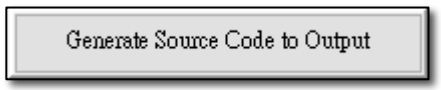
Set the others bit same the sample picture setting.

MICROCHIP

Configuration Bits Window

Step 3 (Optional)

- **Click [Generate Source Code to Output] Button to get your Config Code**



Configuration Bits | Output - Config Bits Source

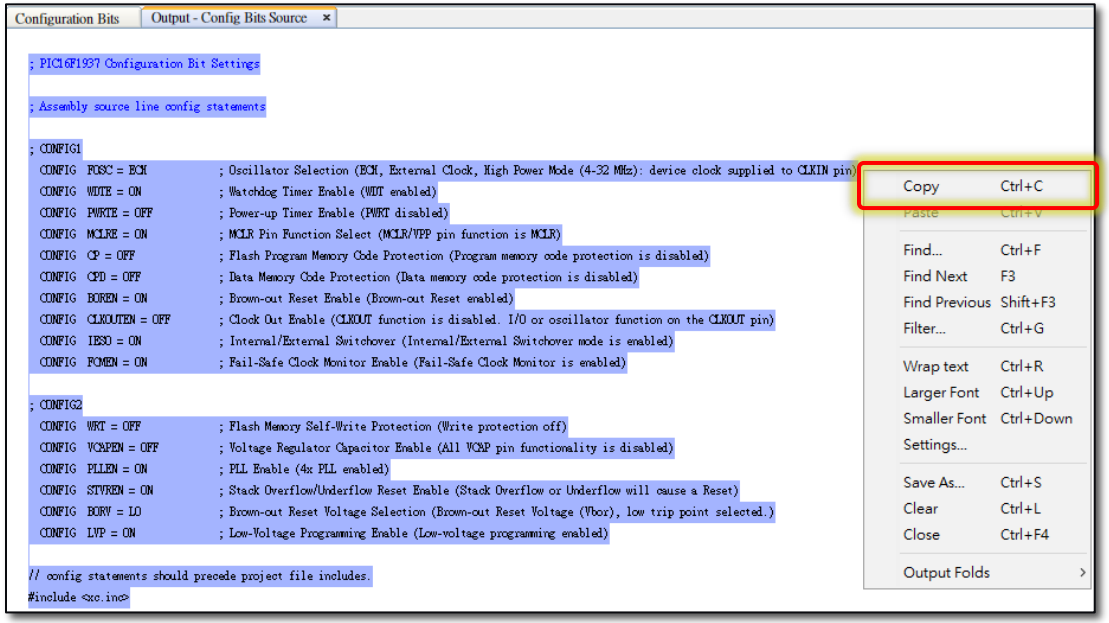
```
; PIC16F1937 Configuration Bit Settings

; Assembly source line config statements

; CONFIG1
  CONFIG  FOSC = ECH            ; Oscillator Selection (ECH, External Clock, High Power Mode (4-32 MHz): device clock supplied to CLKIN pin)
  CONFIG  WDTE = ON             ; Watchdog Timer Enable (WDT enabled)
  CONFIG  PWRTE = OFF           ; Power-up Timer Enable (PWRT disabled)
  CONFIG  MCLRE = ON            ; MCLR Pin Function Select (MCLR/VPP pin function is MCLR)
  CONFIG  CP = OFF              ; Flash Program Memory Code Protection (Program memory code protection is disabled)
  CONFIG  CPD = OFF             ; Data Memory Code Protection (Data memory code protection is disabled)
  CONFIG  BOREN = ON            ; Brown-out Reset Enable (Brown-out Reset enabled)
  CONFIG  CLKOUTEN = OFF        ; Clock Out Enable (CLKOUT function is disabled. I/O or oscillator function on the CLKOUT pin)
  CONFIG  IESO = ON             ; Internal/External Switchover (Internal/External Switchover mode is enabled)
  CONFIG  FCMEN = ON            ; Fail-Safe Clock Monitor Enable (Fail-Safe Clock Monitor is enabled)

; CONFIG2
  CONFIG  WRT = OFF             ; Flash Memory Self-Write Protection (Write protection off)
  CONFIG  VCAPEN = OFF          ; Voltage Regulator Capacitor Enable (All VCAP pin functionality is disabled)
  CONFIG  PLLEN = ON            ; PLL Enable (4x PLL enabled)
  CONFIG  STVREN = ON           ; Stack Overflow/Underflow Reset Enable (Stack Overflow or Underflow will cause a Reset)
  CONFIG  BORV = LO             ; Brown-out Reset Voltage Selection (Brown-out Reset Voltage (Vbor), low trip point selected.)
  CONFIG  LVP = ON              ; Low-Voltage Programming Enable (Low-voltage programming enabled)

// config statements should precede project file includes.
#include <xc.inc>
```

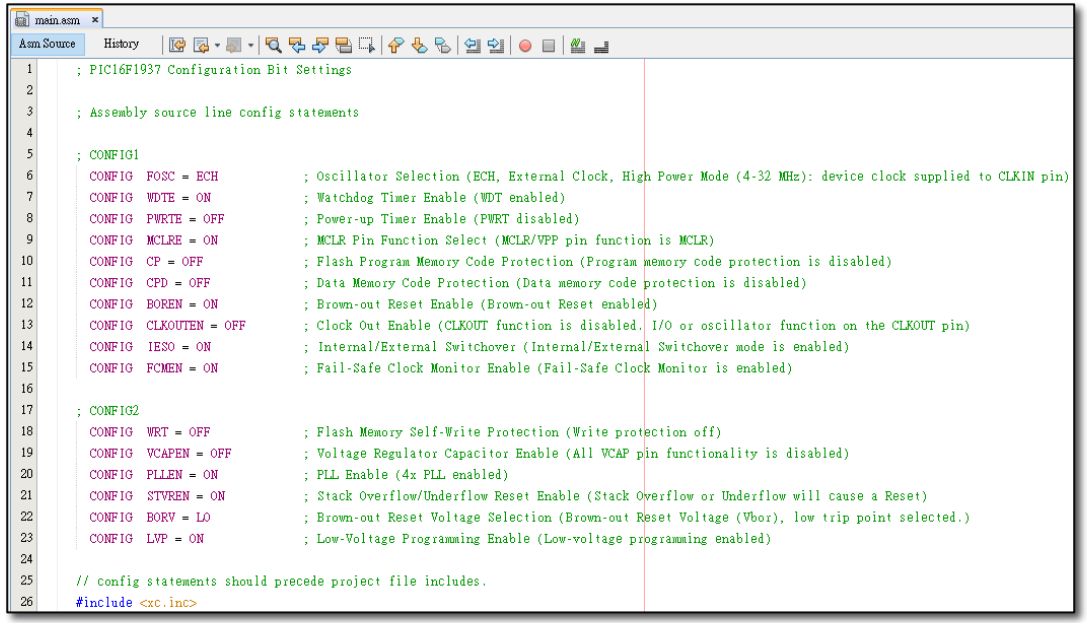
MICROCHIP

Configuration Bits Window

Step 4 (Optional)

- **Copy the contents of the Output window to the main.s**





Lab3 Clock Setting

Step 5

a Add code segment to your main.s

```
; CONFIG1
  CONFIG  FOSC = INTOSC      ; INTOSC oscillator
  CONFIG  WDTE = OFF
  CONFIG  PWRTE = OFF
  CONFIG  MCLRE = ON
  CONFIG  CP = OFF
  CONFIG  CPD = OFF
  CONFIG  BOREN = ON
  CONFIG  CLKOUTEN = OFF
  CONFIG  IESO = ON
  CONFIG  FCMEN = ON
```

CONFIGURATION WORD 1

R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1
FCMEN	IESO	nCLKOUTEN	BOREN1	BOREN0	nCPD	nCP
bit 13						bit 7

R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1	R/P-1/1
MCLRE	nPWRTE	WDTE1	WDTE0	FOSC2	FOSC1	FOSC0
bit 6						bit 0

INTOSC oscillator

Set the others bit same the sample code setting.

MICROCHIP

Lab3 Clock Setting

Step 6

a **Add code segment to your main.s**

```
; CONFIG1
…

; CONFIG2
  CONFIG  WRT = OFF
  CONFIG  VCAPEN = OFF
  CONFIG  PLLEN = OFF          ; PLL Disable
  CONFIG  STVREN = ON
  CONFIG  BORV = LO
  CONFIG  LVP = ON
```

CONFIGURATION WORD 2

R/P-1/1	R/P-1/1	U-1	R/P-1/1	R/P-1/1	R/P-1/1	U-1
LVP	nDEBUG	-	BORV	TVREN	PLLEN	-
bit 13					OFF	bit 7

U-1	R/P-1/1	R/P-1/1	U-1	U-1	R/P-1/1	R/P-1/1
-	VCAPEN<1:0>		-	-	WRT1	WRT0
bit 6						bit 0

Set the others bit same the sample code setting.

MICROCHIP

Lab3 Clock Setting

Step 7

a **Add code segment to your main**

```
main:
   // TODO 3.03
   BANKSEL OSCCON
   movlw 0b01101000 ; IRCF 4MHz
   movwf BANKMASK(OSCCON)
   ...
```

OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0/0	R/W-0/0	R/W-1/1	R/W-1/1	R/W-1/1	U-0	R/W-0/0	R/W-0/0
SPLLEN	IRCF<3:0>				-	SCS<1:0>	
bit 7							bit 0

1101 = 4MHz

Keep the others bit default value.

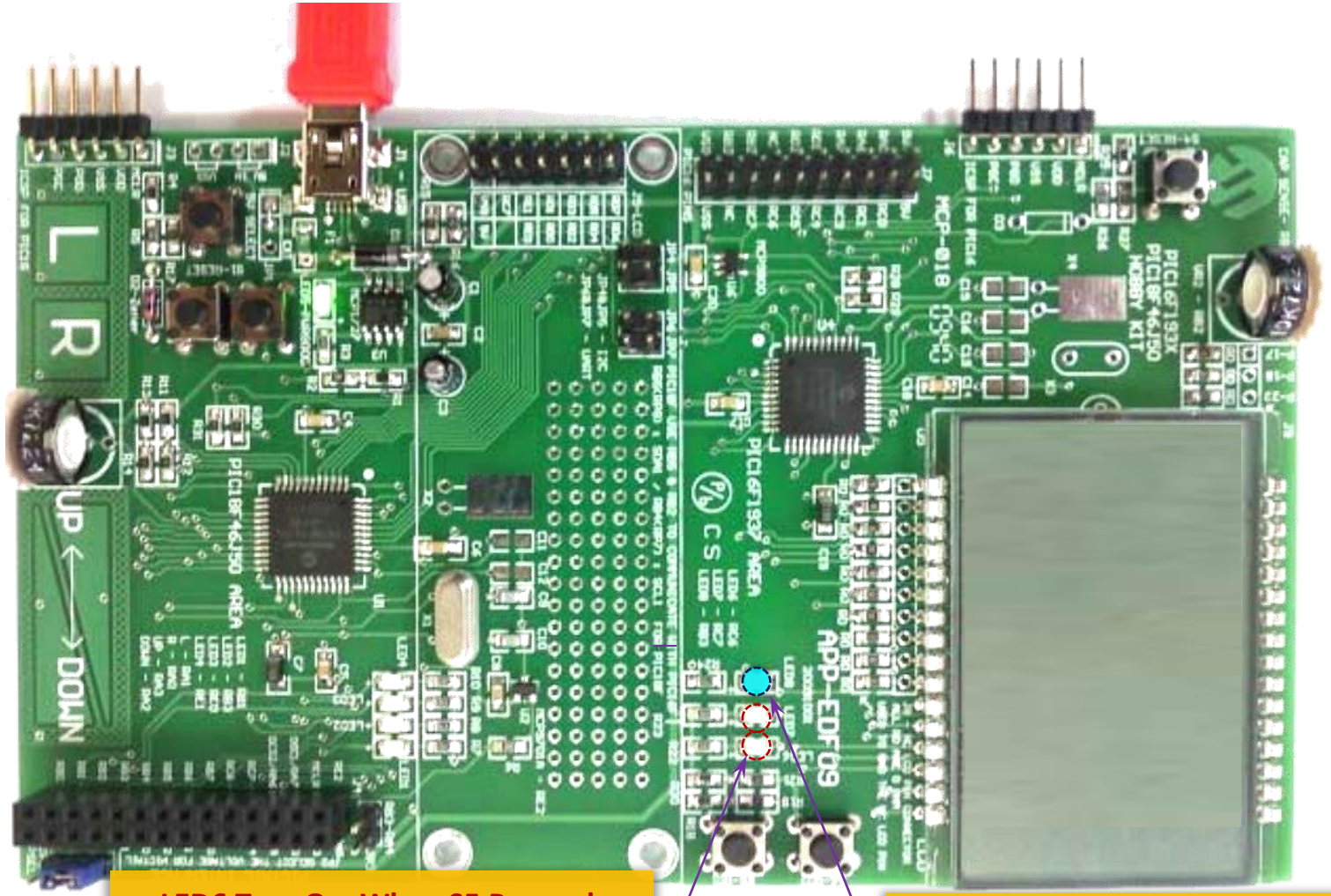
IRCF<3:0>: Internal Oscillator Frequency Select bits
000x = 31 kHz LF
0010 = 31.25 kHz MF
...
0111 = 500 kHz MF (default upon Reset)
1000 = 125 kHz HF
1001 = 250 kHz HF
1010 = 500 kHz HF
1011 = 1 MHz HF
1100 = 2 MHz HF
1101 = 4 MHz HF
1110 = 8 MHz or 32 MHz HF
1111 = 16 MHz HF

b **Program firmware to target board then observe result.**

MICROCHIP

Lab3 Clock Setting

Result



LED6 Turn On, When S5 Pressed.
LED7 Turn On, When S6 Pressed.

LED Blink Speed More Quickly.

Delay Function

- **Discuss**
  - How to produce a Delay function in ms?
  - **FOSC = 4Mhz**

```
PSECT code

delay_ms:
    BANKSEL delay_ms_count
    movwf BANKMASK(delay_ms_count)

delay_ms_0:
    BANKSEL delay_ms_count_1
    movlw 6+2 ;256-x
    movwf BANKMASK(delay_ms_count_1)
    nop

delay_ms_1:
    incf BANKMASK(delay_ms_count_1),f
    btfss ZERO
    goto delay_ms_1


    BANKSEL delay_ms_count
    decfsz BANKMASK(delay_ms_count),f
    goto delay_ms_0

    return
```

```
//2us
; 1
; 1

//3us
; 1
; 1
; 1
; 1

//normal: 1+1+2 us ; last: 1+2+0 us
; 1
; false:1, true:2
; 2

//normal: 1+1+2 us ; last: 1+2+0 us
; 1
; false:1, true:2
; 2

; 2
```

delay_ms_0 = 993us + 3us + 4us = 1000us

= 2us + (delay_ms_0) * ms_count - 1 us + 2us

= (ms_count * 1ms) + 3us

= delay_ms_1 + 3us + 4us + 1us

= (1+1+2)us*(256 - ms_count_1) - 1 us = 999us (ms_count_1 = 6)

→ (1+1+2)us*(256 - ms_count_1) - 1 us = 992us (ms_count_1 = 8)

MICROCHIP

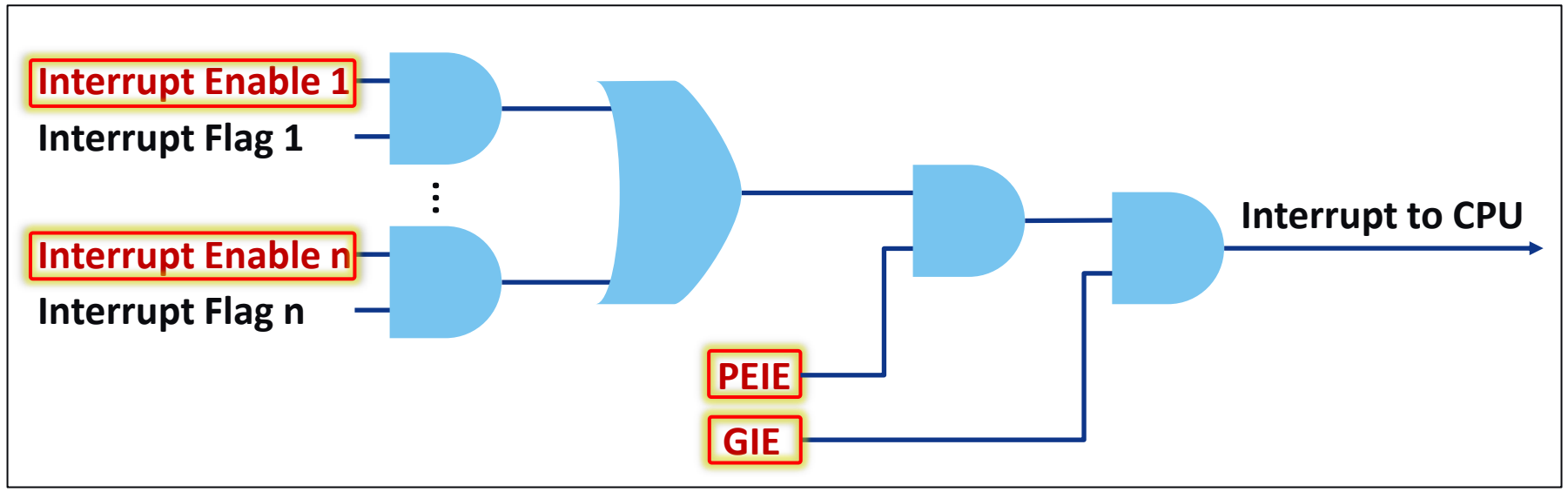
Interrupt Architecture

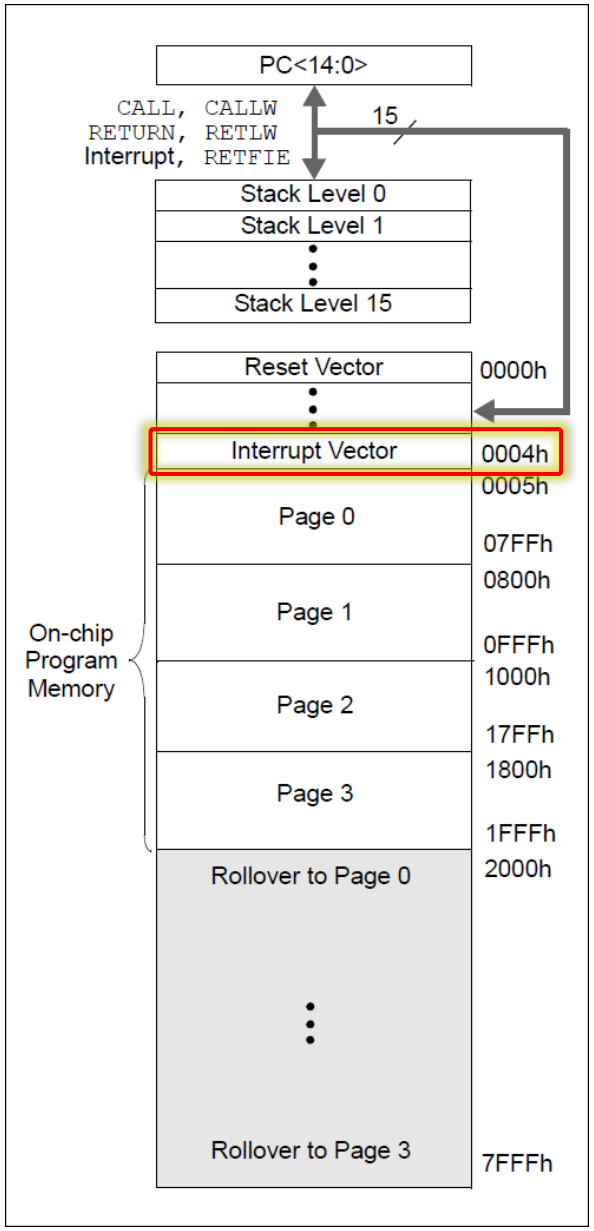
What’s Interrupt ?

- In general, processor is executed in sequence according to the program's flow.
- However, if some peripherals or event needs immediate attention then an interrupt flag been set as high-priority and requiring interrupting current process to serve it.
- Processor will jump to execute the interrupt service routine (ISR, Interrupt Service Routine) to service peripherals or events.

PIC16F Series Interrupt Architecture

- The interrupt enabled by setting the GIE, PEIE, and Interrupt Enable bit(s).
- The following events happen when an interrupt event occurs while the GIE bit is set:
  - GIE bit is cleared, then reset after return from Interrupt.
  - Current Program Counter is pushed onto the stack.
  - Critical registers are automatically saved to the shadow registers.
  - Current Program Counter is loaded with the interrupt vector 0004h.





MICROCHIP

Interrupt Function

- The firmware within the Interrupt Service Routine (ISR) should determine the source of the interrupt by polling the interrupt flag bits.
- The interrupt flag bits must be cleared before exiting the ISR to avoid repeated interrupts.
- The RETFIE(Return from Interrupt) instruction exits the ISR by popping the previous address from the stack, restoring the saved context from the shadow registers and setting the GIE bit.

```
PSECT isrVec, class=CODE, delta=2, abs
   ORG 4H
isr:
   PAGESEL $
   BANKSEL PIRn
   btfss xxxIF
   goto xxx _ISR_END
   bcf xxxIF
   ; Add your Interrupt application code
   …
   …
   …
xxx _ISR_END:
   …
exitISR:
   retfie ;End the Interrupt Function
```

Interrupt Control Register

INTCON: Interrupt Control Register

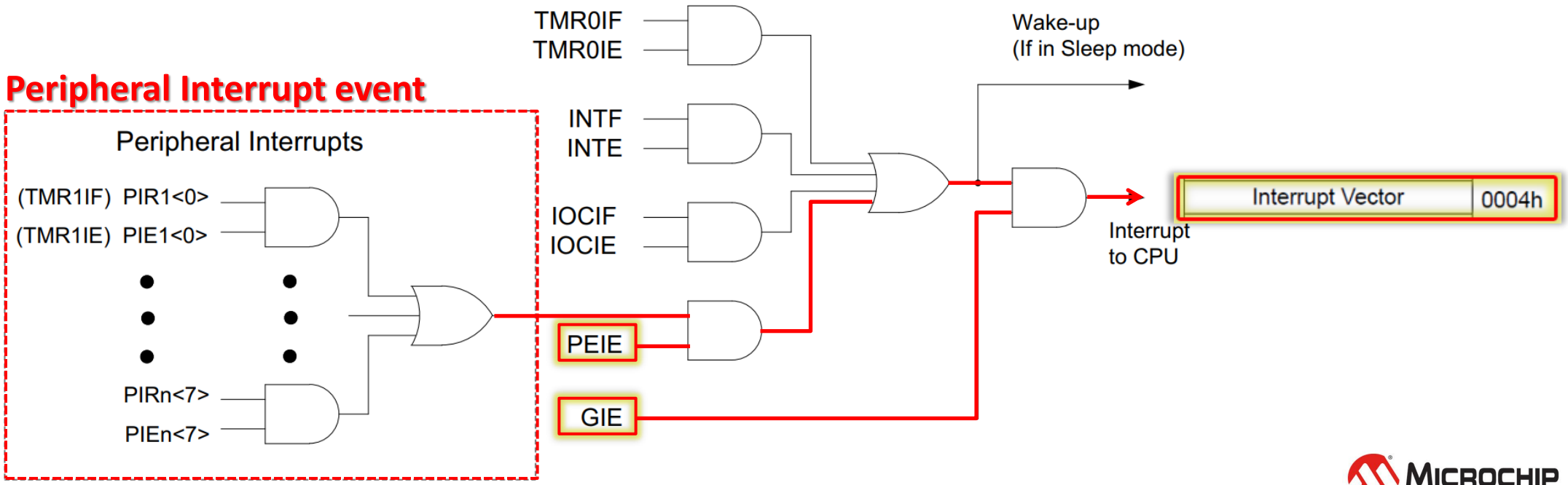
R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF
bit 7							bit 0

Interrupt Control Register

- **Interrupts are disabled upon any device Reset, they are enabled by setting the bits.**

INTCON: Interrupt Control Register

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF
bit 7							bit 0



Timer Architecture

Timer or Counter ?

- **Counter or Timer is a clock counter, use to count how many clocks into module after start.**
- **There are two kinds of generally called Timer or Counter. In fact, it's same.**
  - The Input Clock unknown : Just know how many counts, called Counter.
  - The Input Clock known : Can use count and clock period to calculate time, called Timer.
- **Timer can set a notify condition, like overflow, equal some value etc.**
  - If interrupt disable user must check flag, as soon as possible.
  - If interrupt enable : Timer will notify CPU, if condition is true.
- **PIC16F1937 provide one 16-bit timer (TMR1) and four 8-bits timers.**

Timer2/4/6 Control Register

TxCON: TIMERx Control Register

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
-	TxOUTPS<3:0>				TMRxON	TxCKPS<1:0>	
bit 7							bit 0

PRx: TIMERx Period Register

R/W
PRx<7:0>
bit 7 bit 0

TMRx: TIMERx Module Register

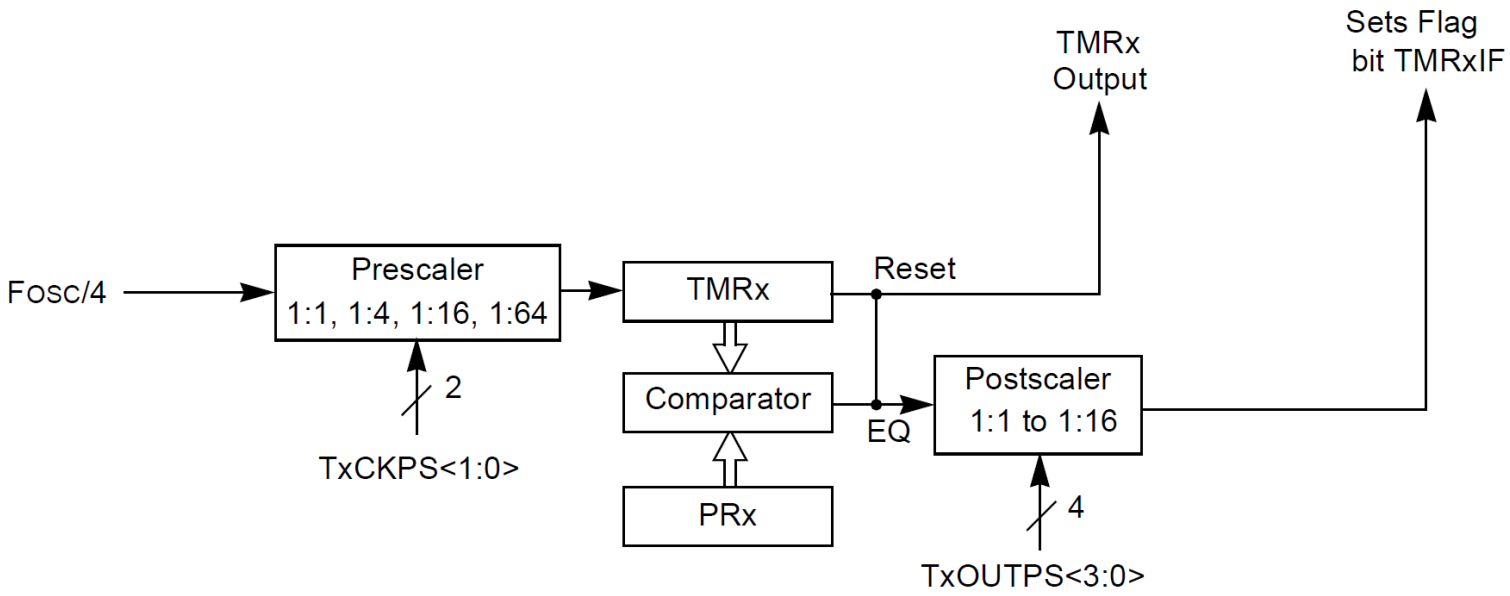
R/W
TMRx<7:0>
bit 7 bit 0

TIMERx Control Register

- **Timer2/4/6 is enabled by configuring the TMRxON.**

TxCON: TIMERx Control Register

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
-	TxOUTPS<3:0>				TMRxON	TxCKPS<1:0>	
bit 7							bit 0

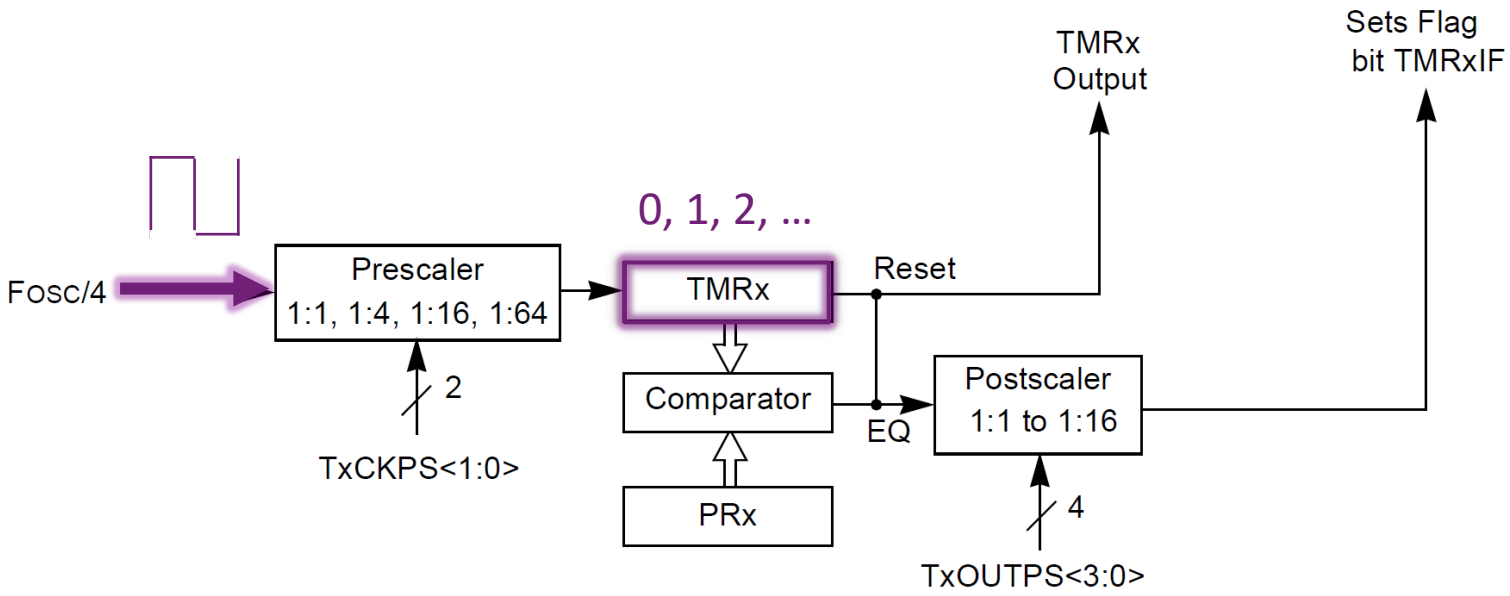


TIMERx Module Register

- **TMRx increments from 00h on each clock edge.**

TMRx: TIMERx Module Register

R/W
PRx<7:0>
bit 7 bit 0

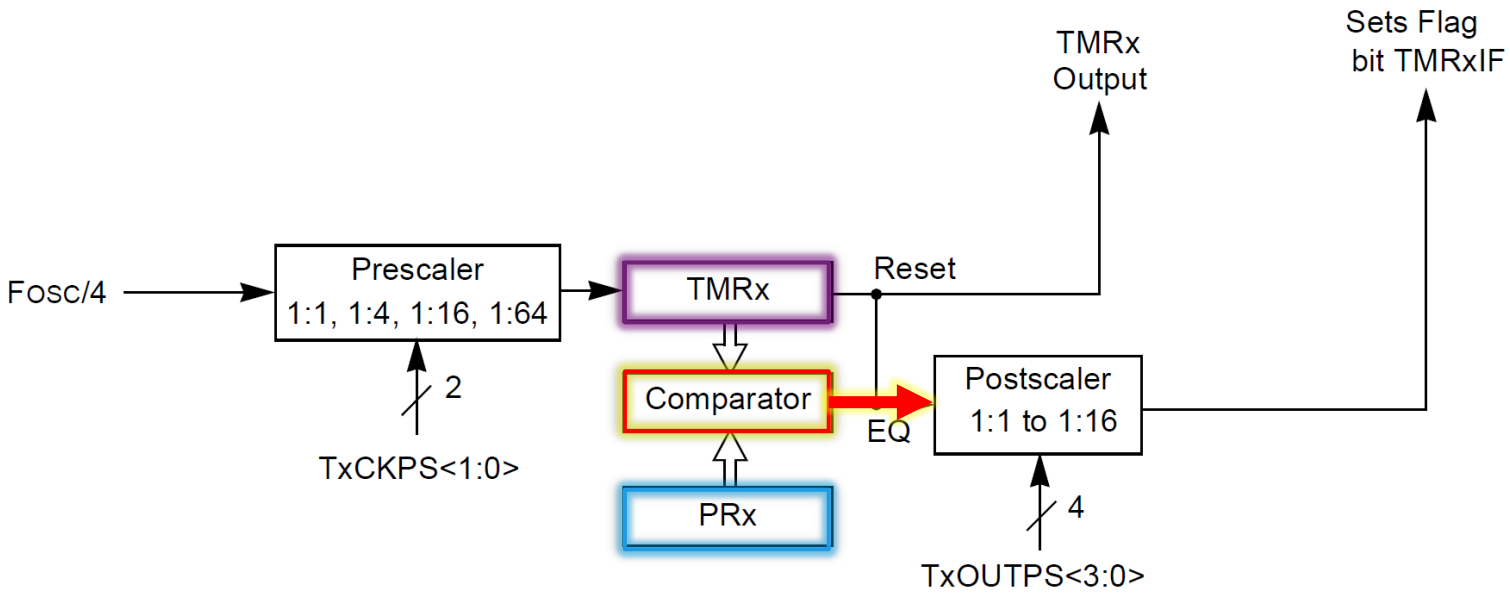


TIMERx Period Register

- **TMRx is compared to that of the Period register, PRx, on each clock cycle.**

PRx: TIMERx Period Register

R/W	
PRx<7:0>	
bit 7	bit 0

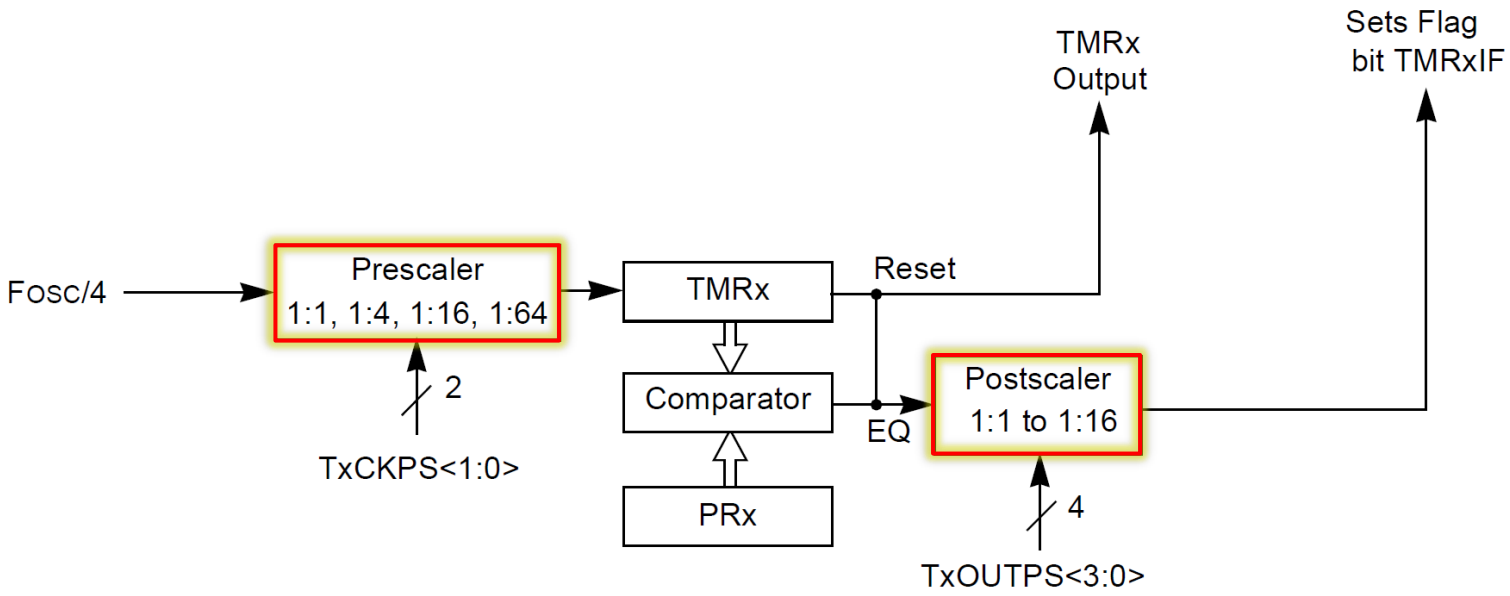


Software programmable Prescaler and Postscaler

- **These options are selected by the Prescaler or Postscaler control bits.**

TxCON: TIMERx Control Register

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
-	TxOUTPS<3:0>				TMRxON	TxCKPS<1:0>	
bit 7							bit 0



Timer0/1/2 Interrupt Control Registers

INTCON: Interrupt Control Register

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF
bit 7							bit 0

PIE1: Peripheral Interrupt Enable Register 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

PIR1: Peripheral Interrupt Request Register 1

R/W-0/0	R/W-0/0	R-0/0	R-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Timer4/6 Interrupt Control Registers

PIE3: Peripheral Interrupt Enable Register 3

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	U-0	R/W-0/0	U-0
-	CCP5IE	CCP4IE	CCP3IE	TMR6IE	-	TMR4IE	-
bit 7							bit 0

PIR3: PERIPHERAL INTERRUPT REQUEST REGISTER 3

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
-	CCP5IF	CCP4IF	CCP3IF	TMR6IF	-	TMR4IF	-
bit 7							bit 0

Timer2/4/6 Interrupt

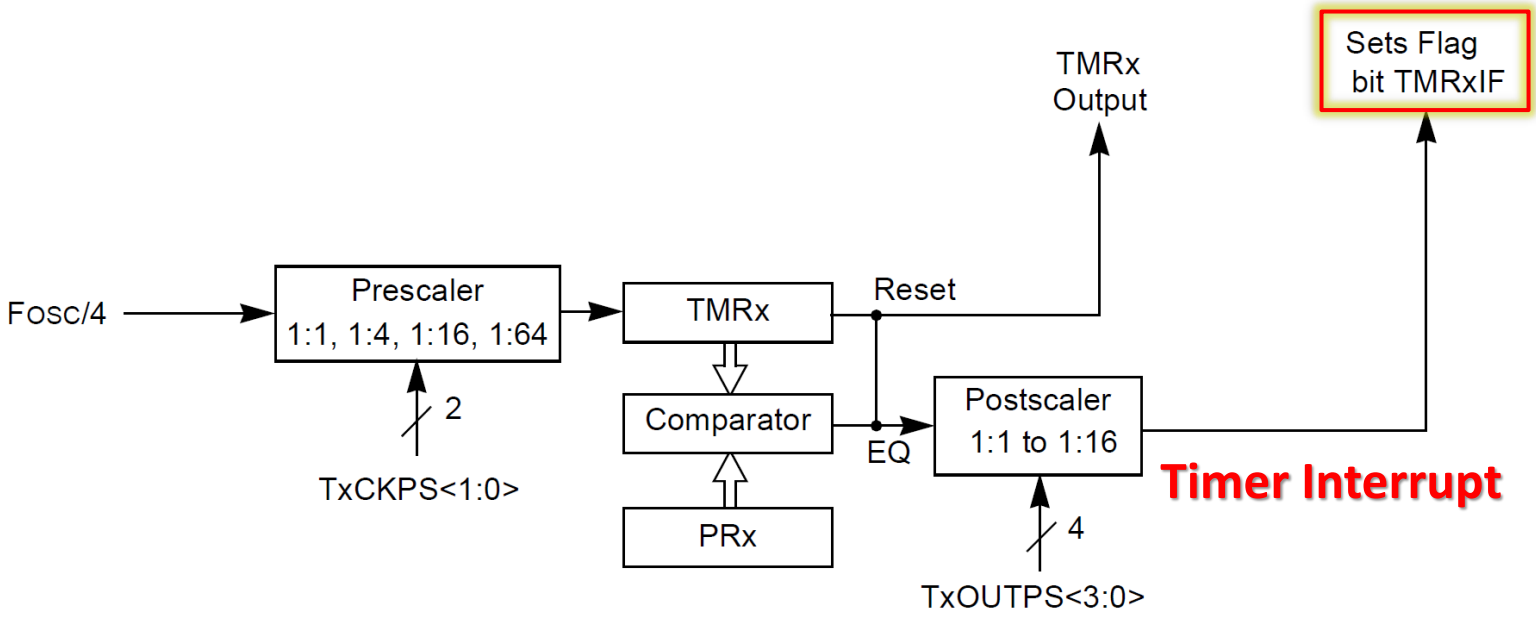
- **Timer2/4/6 can also generate an optional device interrupt.**

PIR1: Peripheral Interrupt Request Register 1

R/W-0/0	R/W-0/0	R-0/0	R-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Timer 2 Interrupt

R/W-0/0	
TMR4IF	Timer 4 Interrupt
R/W-0/0	
TMR6IF	Timer 6 Interrupt



MICROCHIP

Lab4 Timer2 Interrupt

- Base on current project or Open .\Lab4.0_xxx.X to do exercises
- Try to add TMR2 to replace software delay to control LEDs toggle period.
- The Timer2 setup to timer mode, timer ON or OFF period is 200ms.
- RB3 ▸ LED (LED8).

Let’s go!

Calculate the Timer Period Setting

- **Target's timer period is 200ms.**

Step 1. Confirm the current setting

- Check current system instruction clock (FOSC/4) = 4MHz/4 = 1Mhz. (Lab3)
- Timer Holding Register is 8-bit. (range is 256, 1/1Mhz*256 = 256us, 256us < 200ms)

Step 2. Adjust Prescaler & Postscaler

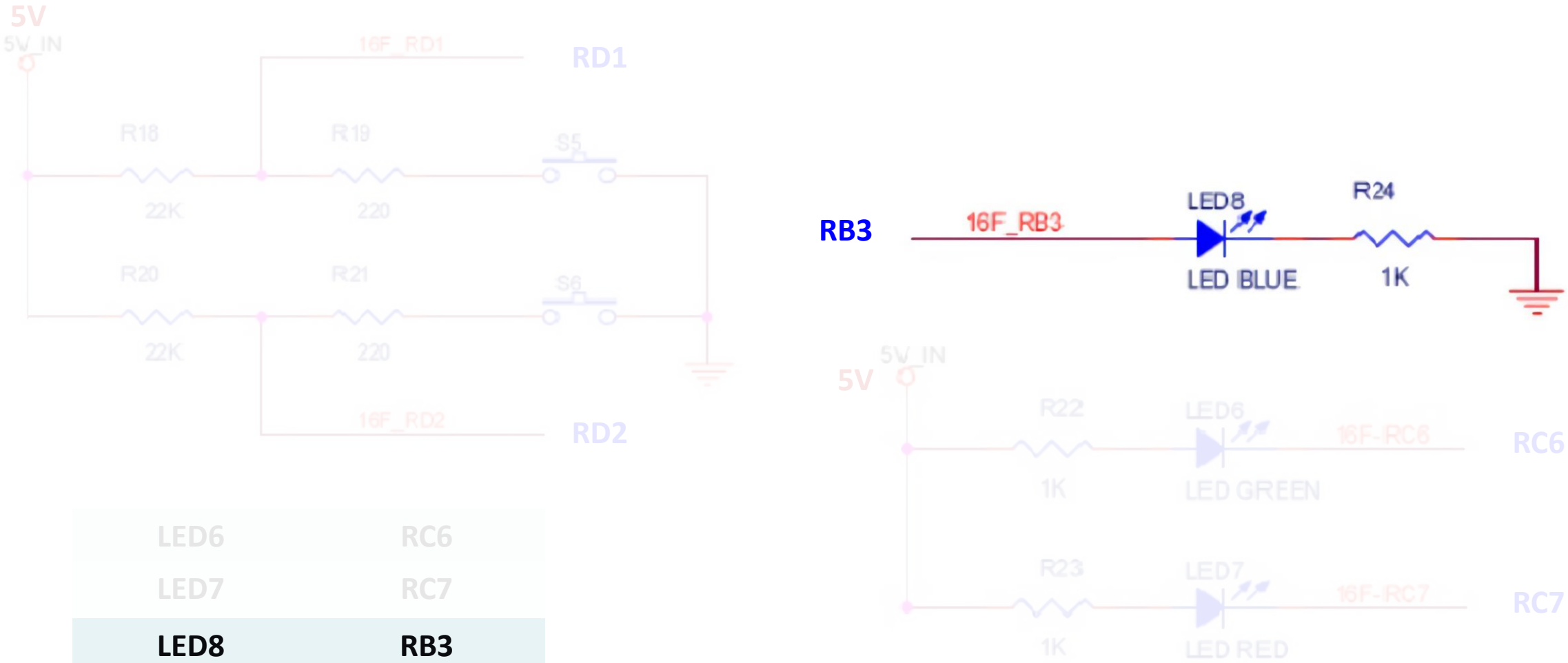
- Select **Prescaler is 64**, 1/(1Mhz/64)*256 = 16.384ms, 16.384ms < 200ms.
- Select **Postscaler is 1:16**, 1/(1Mhz/64/16)*256 = 262.144ms, 262.144ms > 200ms.

Step 3. Set Timer Period to 200ms

- 1/(1Mhz/64/16) = 1.024ms
- 200ms / 1.024ms = 195.3125, The value must be an integer.
- 1.024ms*195 = 199.68ms
- 1.024ms*196 = 200.704ms

→ **Timer Period Setting is (195-1) or (196-1).**
(0~194) or (0~195)

Check Schematic



LED6	RC6
LED7	RC7
LED8	**RB3**
S5	RD1
S6	RD2

→ 1:LED Light or 0:LED Dark

Lab4 Timer2 Interrupt

Step 1

a **Remove "TODO 1.02" and Add code segment below #include**

```
#include <xc.inc>
...
// TODO 4.01
#define LED8 BANKMASK(LATB),3
;Fosc=4Mhz, Fosc/4=1MHz
;1us*16*(195)*64 = 199.68ms
#define TMR2_Period 194 ;(195-1)
...
```

PR2: TIMER2 PERIOD REGISTER

R/W
PR2<7:0>
bit 7 — bit 0

Period = 194 ≈ 200ms

LATB: PORTB Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0
bit 7							bit 0

Lab4 Timer2 Interrupt

Step 2

a **Add code segment to your main**

```
main:
    ...
    // TODO 4.02
    BANKSEL PR2
    movlw TMR2_Period ; 200ms
    movwf BANKMASK(PR2)
    BANKSEL TMR2
    clrf BANKMASK(TMR2)
    BANKSEL PIR1
    bcf TMR2IF
    BANKSEL PIE1
    bsf TMR2IE
    BANKSEL T2CON
    ;T2CKPS 1:64; T2OUTPS 1:16; TMR2ON on
    movlw 0b01111111
    movwf BANKMASK(T2CON)
```

PR2: TIMER2 PERIOD REGISTER

R/W
PR2<7:0>
bit 7 — bit 0

Period ≈ 200ms

TMRx: TIMERx MODULE REGISTER

R/W
TMRx<7:0>
bit 7 — bit 0

0 = Clear Timer Conter

T2CON: TIMER2 CONTROL REGISTER

1 = ON

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
-	T2OUTPS<3:0>				TMR2ON	T2CKPS<1:0>	
bit 7							bit 0

1111 = 1:16 Postscaler

11 = Prescaler is 64

MICROCHIP

Lab4 Timer2 Interrupt

Step 2

a **Add code segment to your main**

```
main:
    ...
    // TODO 4.02
    BANKSEL PR2
    movlw TMR2_Period ; 200ms
    movwf BANKMASK(PR2)
    BANKSEL TMR2
    clrf BANKMASK(TMR2)
    BANKSEL PIR1
    bcf TMR2IF
    BANKSEL PIE1
    bsf TMR2IE
    BANKSEL T2CON
    ;T2CKPS 1:64; T2OUTPS 1:16; TMR2ON on
    movlw 0b01111111
    movwf BANKMASK(T2CON)
```

PIR1: Peripheral Interrupt Request Register 1

R/W-0/0	R/W-0/0	R-0/0	R-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

0 = reset

PIE1: Peripheral Interrupt Enable Register 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

1 = enable

MICROCHIP

Lab4 Timer2 Interrupt

Step 3

a **Add code segment to your main**

```
main:
    ...
    // TODO 4.02
    ...
    // TODO 4.03
    BANKSEL INTCON
    bsf PEIE
    bsf GIE
```

INTCON: Interrupt Control Register

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF
bit 7	1 = Enable						bit 0

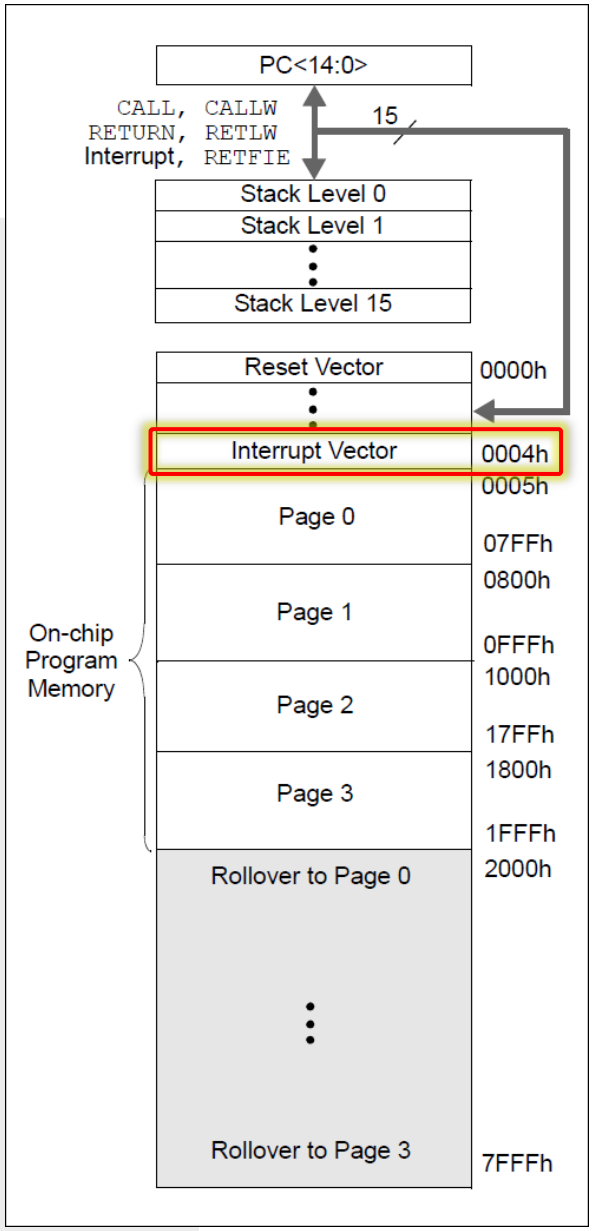
MICROCHIP

Lab4 Timer2 Interrupt

Step 4

a **Add code segment below #include**

```
// TODO 4.04
PSECT isrVec,class=CODE,delta=2,abs
   ORG 4H
isr:
   PAGESEL $
   …

exitISR:
   retfie
```



MICROCHIP

Lab4 Timer2 Interrupt

Step 5

a Add code segment to your isr

```
isr:
   ...

   // TODO 4.05
   BANKSEL PIR1
   btfss TMR2IF
   goto TMR2IF_END
   bcf TMR2IF                ; clear the interrupt flag
   BANKSEL TMR2
   clrf BANKMASK(TMR2)  ; reload the timer
   ...

TMR2IF_END:
   ...

exitISR:
   retfie
```

PIR1: Peripheral Interrupt Request Register 1

R/W-0/0	R/W-0/0	R-0/0	R-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7						0 = reset	bit 0

TMRx: TIMERx MODULE REGISTER

R/W
TMRx<7:0>
bit 7 — 0 = Clear Timer Conter — bit 0

Lab4 Timer2 Interrupt

Step 6

a **Add code segment to your isr**

```
isr:
   // TODO 4.05
   ...
   // TODO 4.06
   BANKSEL LATB
   btfss LED8
   goto LED8_Light
   goto LED8_Dark
LED8_Light:
   bsf LED8
   goto LED8_END
LED8_Dark:
   bcf LED8
   goto LED8_END
LED8_END:

TMR2IF_END:

exitISR:
   retfie
```

LATB: PORTB Data Latch Register **LED8** → 1:High or 0:Low

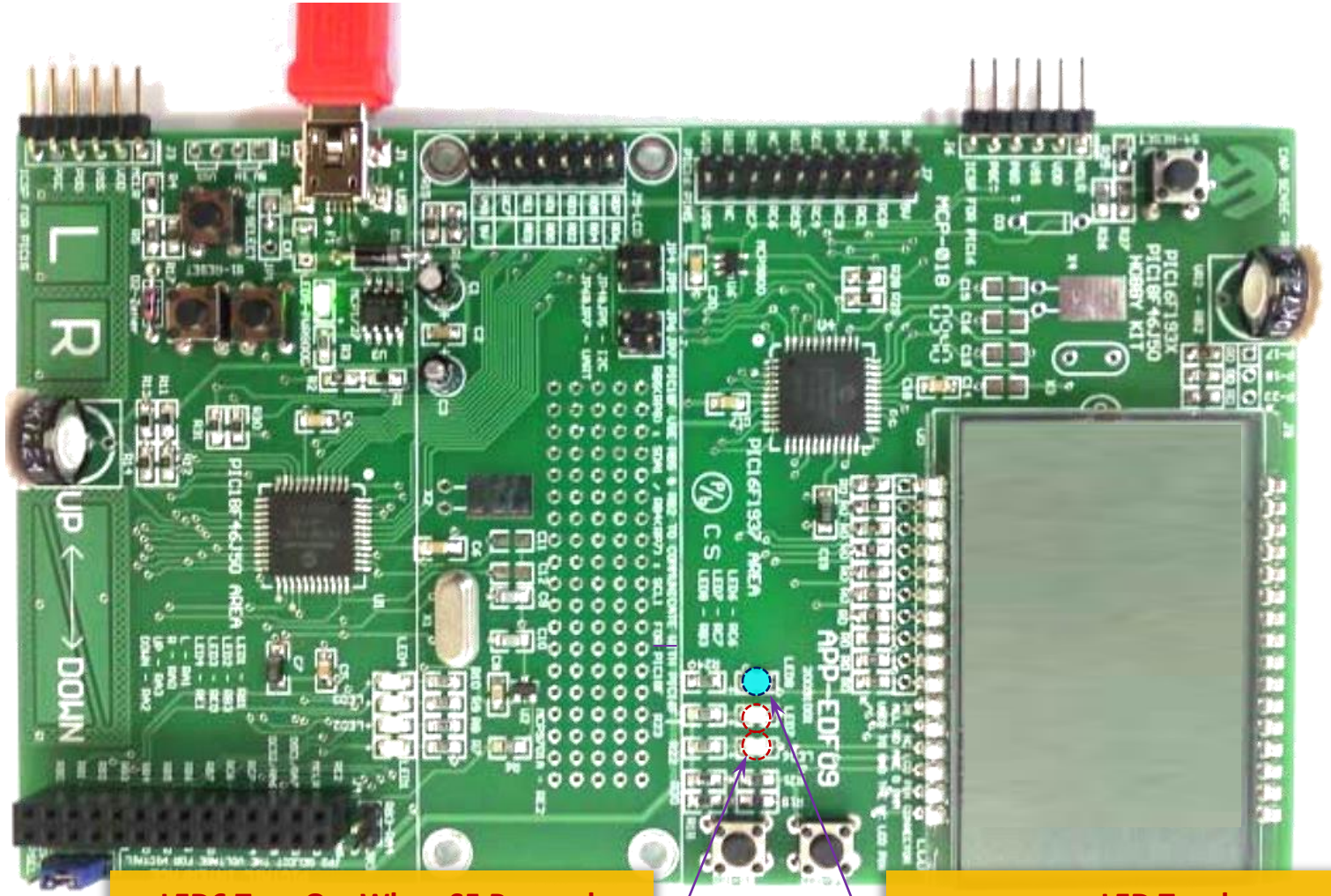
R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0
bit 7				**0 or 1**			bit 0

b **Program firmware to target board then observe result.**

MICROCHIP

Lab4 Timer2 Interrupt

Result



LED6 Turn On, When S5 Pressed.
LED7 Turn On, When S6 Pressed.

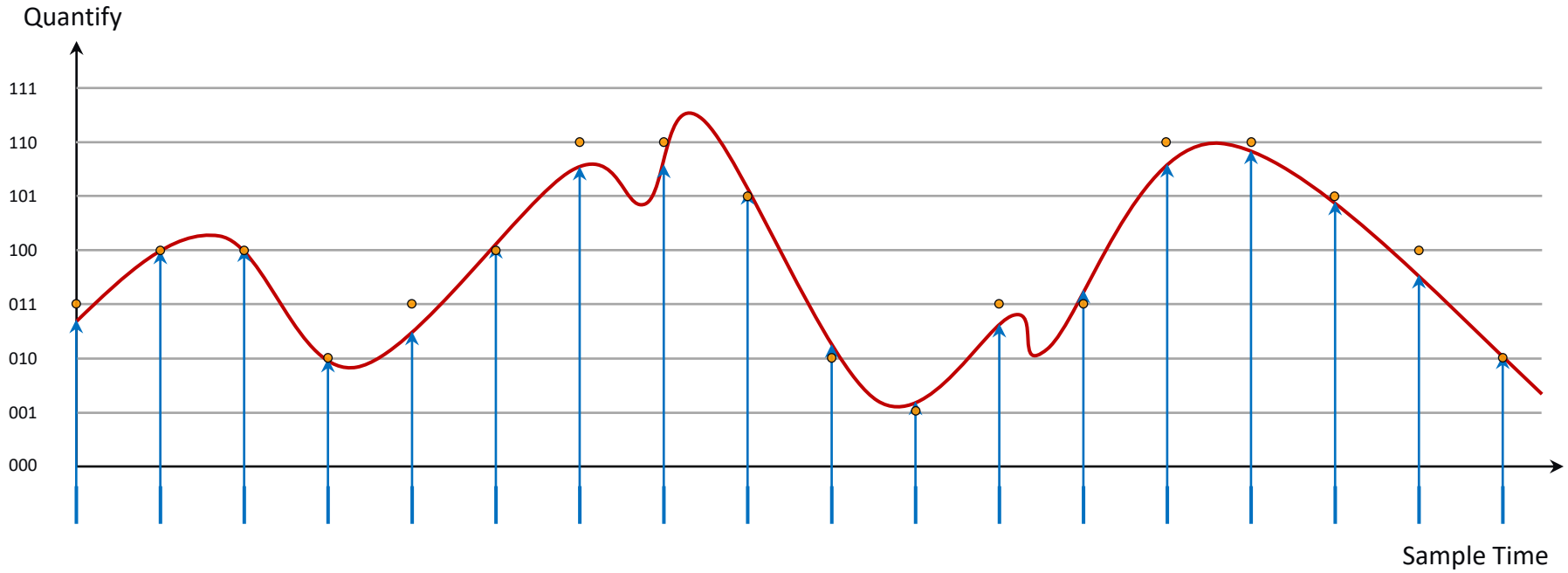
LED Toggle,
Interval Period use "Timer2" callback.

MICROCHIP

ADC Architecture

What’s ADC ?

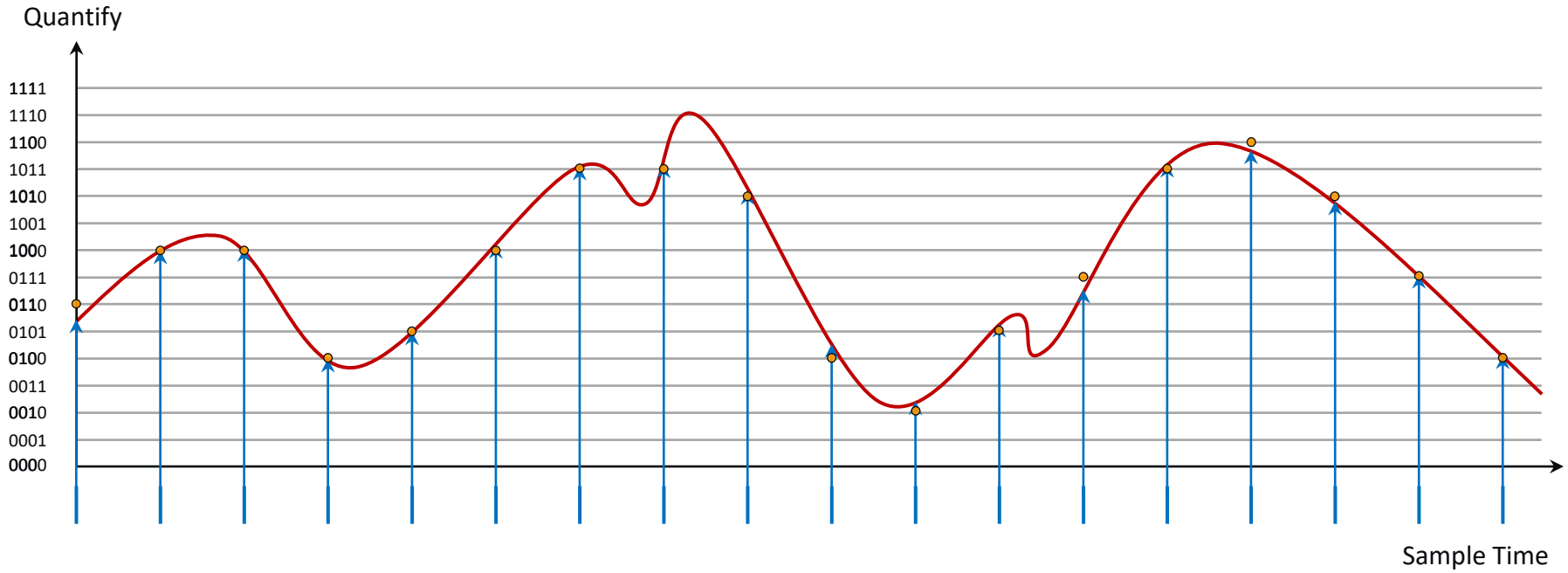
- **The Analog-to-Digital Converter(ADC) converts a signal from the time/amplitude continuous domain into a time/amplitude discrete domain.**



MICROCHIP

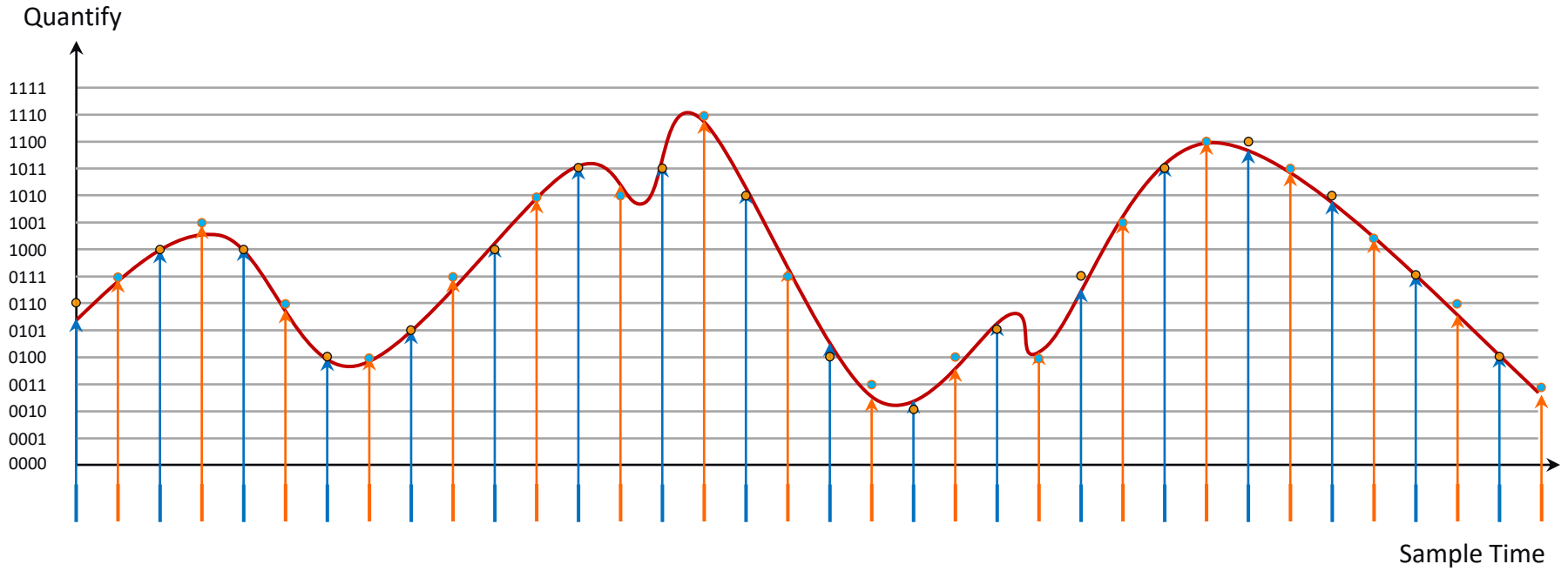
Resolution

- **The resolution of the converter indicates the number of discrete values. The resolution determines the magnitude of the quantization error. (wiki)**



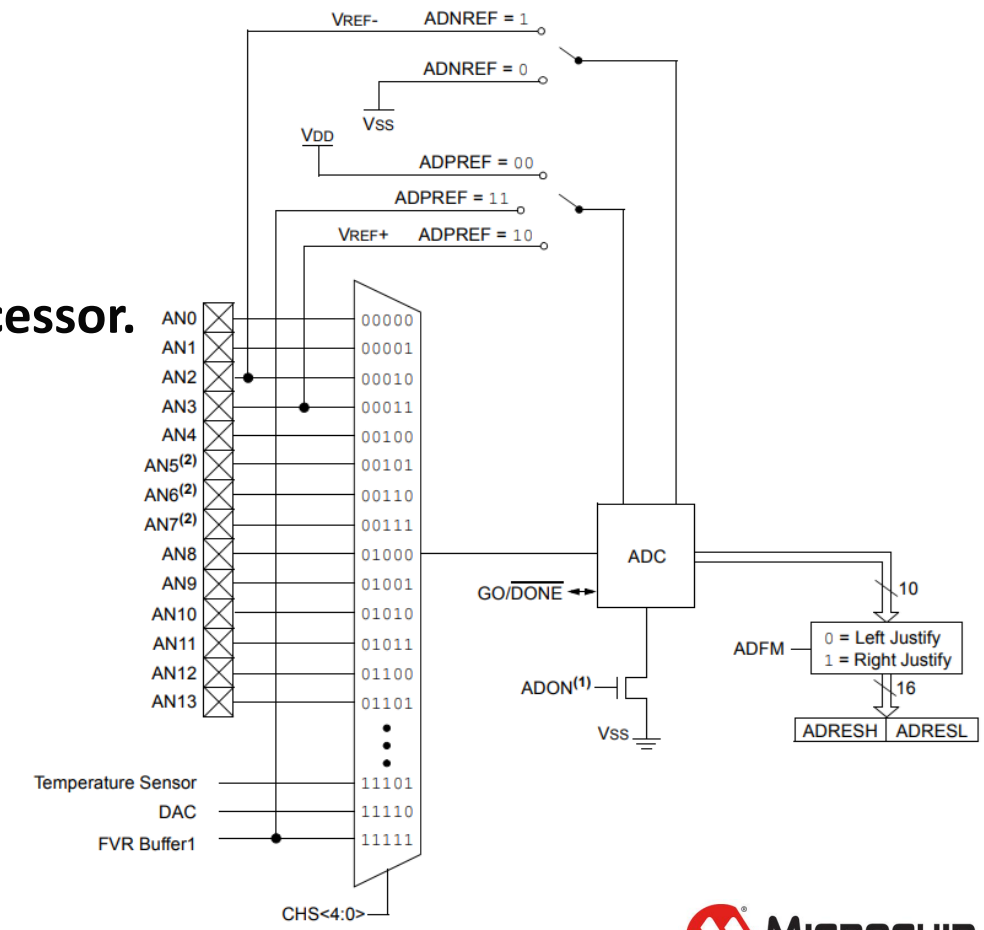
Sample Rate

- The analog signal is required to define the rate at which new digital values are sampled from the analog signal.
- This faithful reproduction is only possible if the sampling rate is higher than twice the highest frequency of the signal. (wiki)



PIC16F Series ADC Features

- **Analog-to-Digital Converter (ADC) includes the following features:**
  - **10-bit resolution and up to 14 channels analog inputs multiplexed into one A/D converter.**
  - **5 µs typical sampling time, as fast as 11 µs conversion time.**
  - **A/D sampling frequency up to 62.5 kHz.**
  - **External reference inputs, VREF+ & VREF-.**
  - **A/D Result can be left or right justified.**
  - **10-bit resolution with ±1 LSb accuracy.**
  - **A/D conversion during SLEEP. A/D complete can wake processor.**



ADC Control Registers

ADCON0: A/D Control Register 0

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
-	CHS<4:0>					GO/nDONE	ADON
bit 7							bit 0

ADCON1: A/D Control Register 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
ADFM	ADCS<2:0>			-	ADNREF	ADPREF<1:0>	
bit 7							bit 0

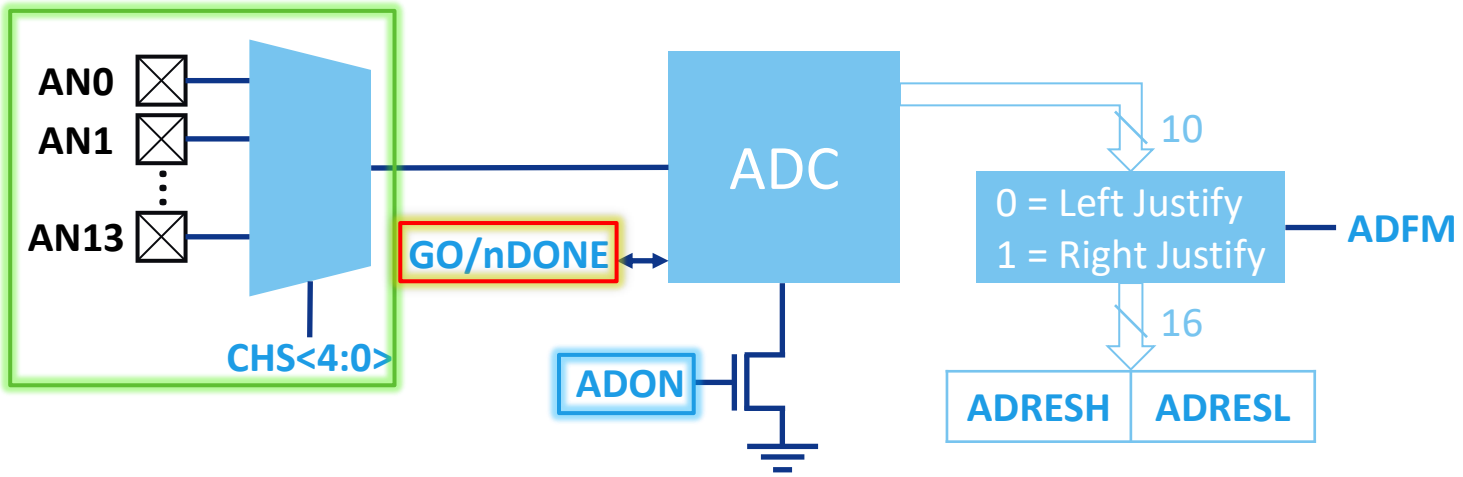
ADC Setting

- **Select Analog Channel and then Enable the ADC Module.**

ADCON0: A/D Control Register 0

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
-	CHS<4:0>					GO/nDONE	ADON
bit 7							bit 0

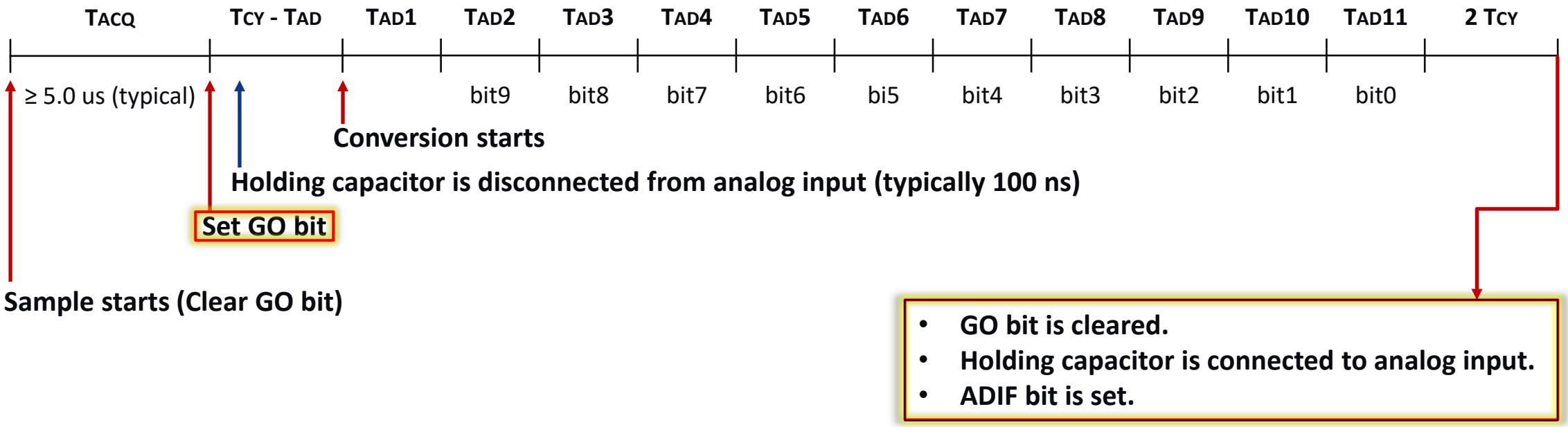
Configure port and Select ADC input channel.



Sample and Conversion Clock

- **The time range of sample (T_{ACQ}) ≥ 5.0us** (typical).
- **The time to complete one bit conversion is defined as T_{AD}.**
- **One full 10-bit conversion requires 11 T_{AD} periods.**
- **The time range of T_{AD} ≥ 1.0us and ≤ 6.0us.**

ADC Clock Period (TAD)		Device Frequency (FOSC)					
ADC Clock Source	ADCS<2:0>	32 MHz	20 MHz	16 MHz	8 MHz	4 MHz	1 MHz
Fosc/2	000	~~62.5 ns~~	~~100 ns~~	~~125 ns~~	~~250 ns~~	~~500 ns~~	2.0 us
Fosc/4	100	~~125 ns~~	~~200 ns~~	~~250ns~~	~~500 ns~~	1.0 us	4.0 us
Fosc/8	001	~~0.5 us~~	~~400 ns~~	~~0.5 us~~	1.0 us	2.0 us	8.0 us
Fosc/16	101	~~800 ns~~	~~800 ns~~	1.0 us	2.0 us	4.0 us	16.0 us
Fosc/32	010	1.0 us	1.6 us	2.0 us	4.0 us	8.0 us	32.0 us
Fosc/64	110	2.0 us	3.2 us	4.0 us	8.0 us	16.0 us	64.0 us
FRC	X11	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us



- **GO bit is cleared.**
- **Holding capacitor is connected to analog input.**
- **ADIF bit is set.**

MICROCHIP

ADC Setting

- **Select Conversion ADC Clock Period (TAD) to inside of range.**

ADCON1: A/D Control Register 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
ADFM	ADCS<2:0>			-	ADNREF	ADPREF<1:0>	
bit 7							bit 0

ADC Clock Period (TAD)		Device Frequency (FOSC)					
ADC Clock Source	**ADCS<2:0>**	**32 MHz**	**20 MHz**	**16 MHz**	**8 MHz**	**4 MHz**	**1 MHz**
Fosc/2	000	~~62.5 ns~~	~~100 ns~~	~~125 ns~~	~~250 ns~~	~~500 ns~~	2.0 us
Fosc/4	100	~~125 ns~~	~~200 ns~~	~~250ns~~	~~500 ns~~	1.0 us	4.0 us
Fosc/8	001	~~0.5 us~~	~~400 ns~~	~~0.5 us~~	1.0 us	2.0 us	8.0 us
Fosc/16	101	~~800 ns~~	~~800 ns~~	1.0 us	2.0 us	4.0 us	16.0 us
Fosc/32	010	1.0 us	1.6 us	2.0 us	4.0 us	8.0 us	32.0 us
Fosc/64	110	2.0 us	3.2 us	4.0 us	8.0 us	16.0 us	64.0 us
FRC	X11	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us	1.0-6.0 us

TAD ≥ 1.0us and ≤ 6.0us

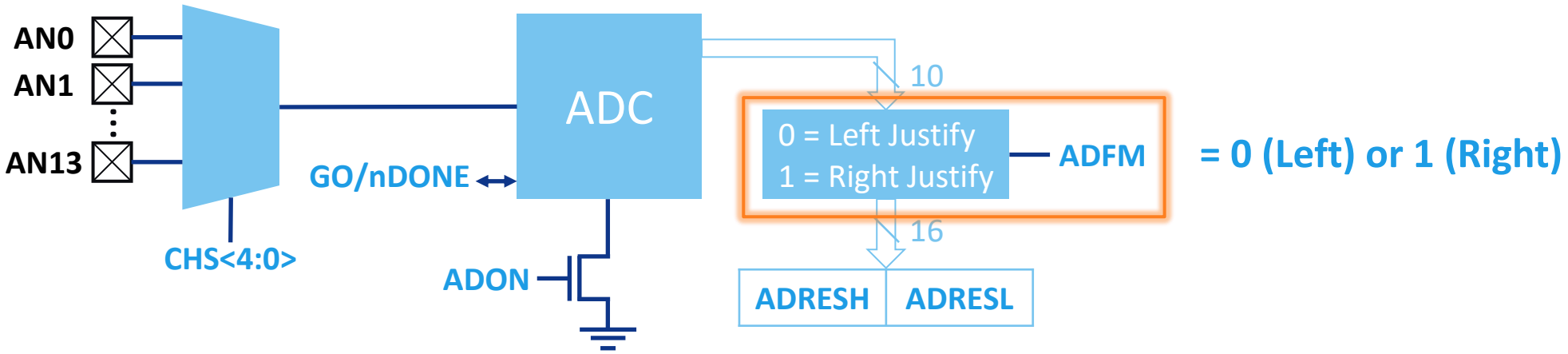
MICROCHIP

ADC Setting

- **Select Conversion Clock and Output Format.**

ADCON1: A/D Control Register 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
ADFM	ADCS<2:0>			-	ADNREF	ADPREF<1:0>	
bit 7							bit 0



ADC Registers

ADRESH: ADC Result Register High (ADRESH), ADFM = 0

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
ADRES<9:2>							
bit 7							bit 0

ADRESL: ADC Result Register Low (ADRESL), ADFM = 0

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
ADRES<1:0>		-	-	-	-	-	-
bit 7							bit 0

ADFM = 0 (Left)

ADRESH: ADC Result Register High (ADRESH), ADFM = 1

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
-	-	-	-	-	-	ADRES<9:8>	
bit 7							bit 0

ADRESL: ADC Result Register Low (ADRESL), ADFM = 1

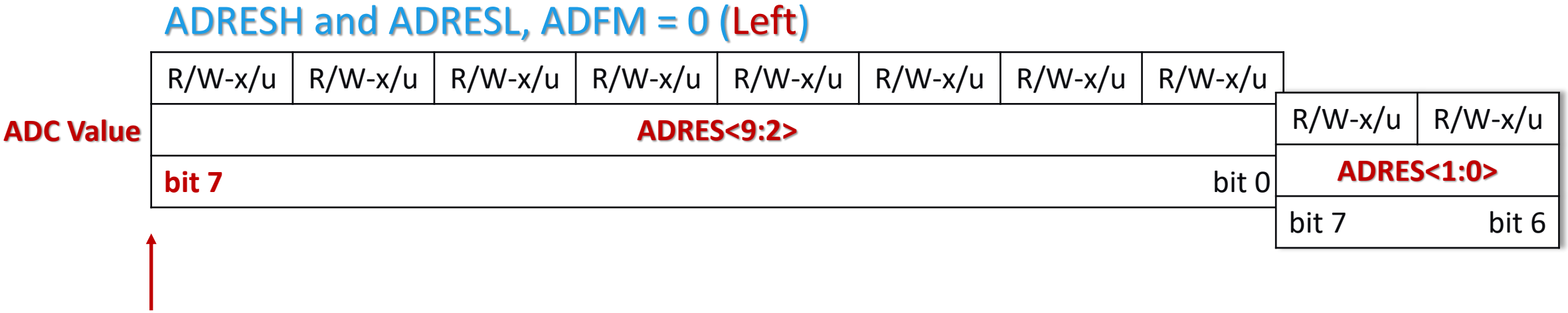
R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
ADRES<7:0>							
bit 7							bit 0

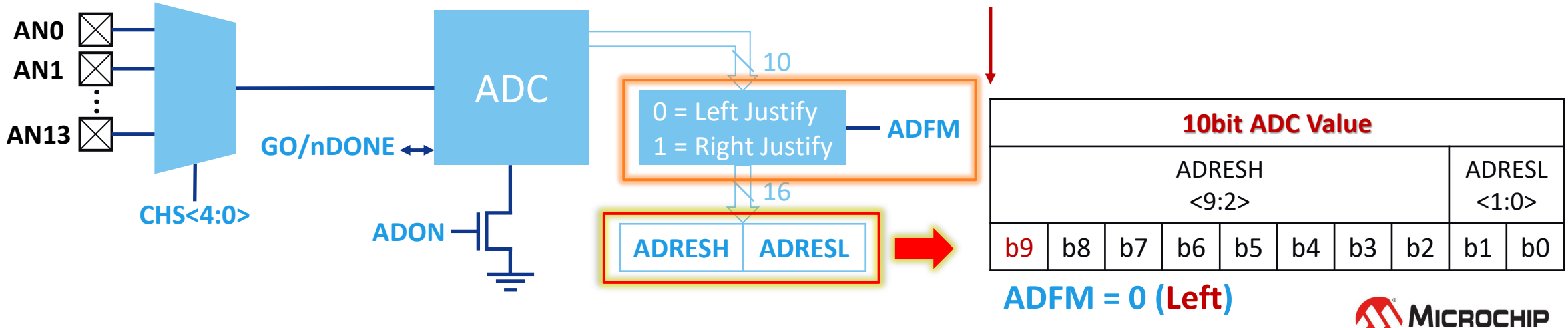
ADFM = 1 (Right)

MICROCHIP

ADC Setting

- **ADC result register format setting.**





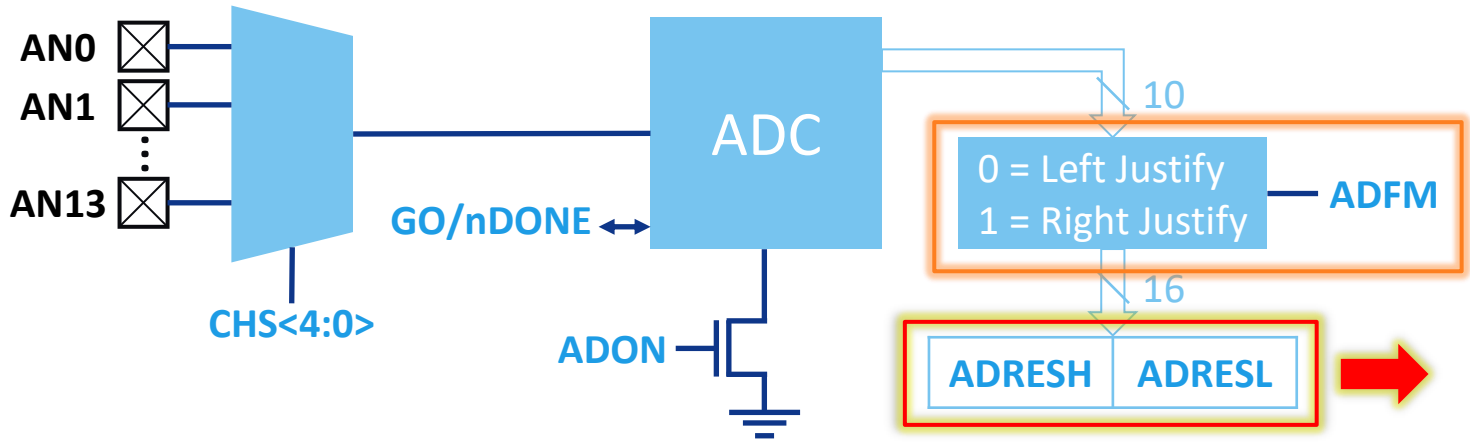
ADC Setting

- **ADC result register format setting.**

ADRESH and ADRESL, ADFM = 1 (Right)

ADC Value

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
ADRES<9:8>		ADRES<7:0>							
bit 1	bit 0	bit 7							bit 0



10bit ADC Value									
ADRESH <1:0>		ADRESL <7:0>							
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

ADFM = 1 (Right)

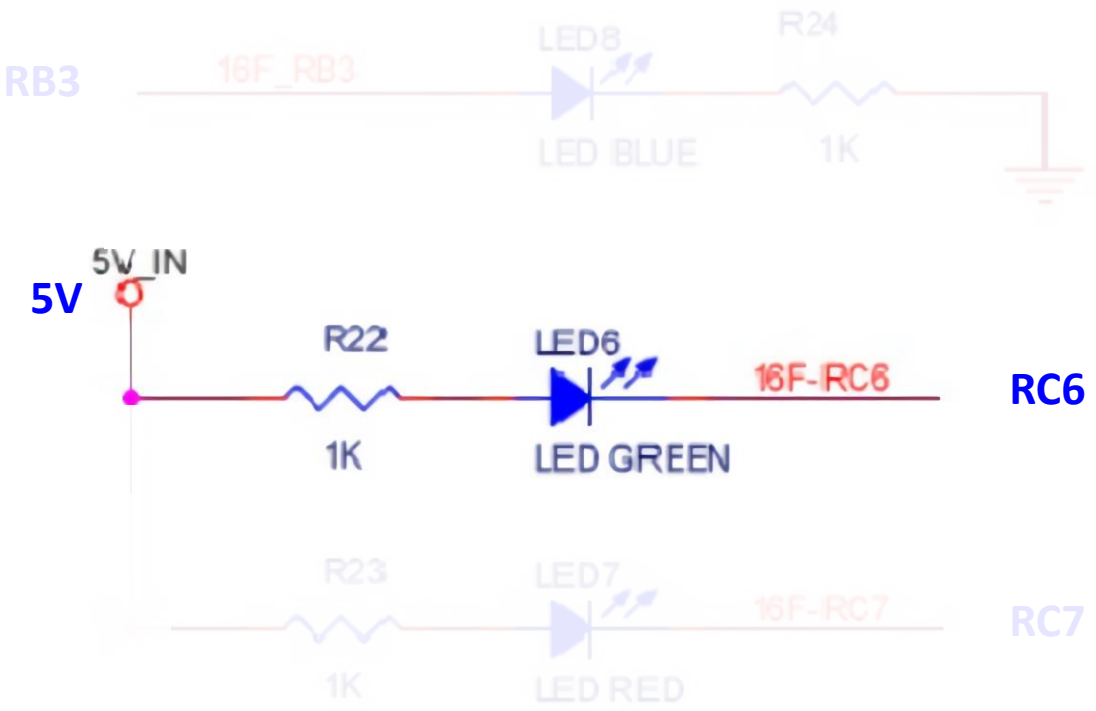
MICROCHIP

Lab5 ADC Channel Periodically

- Base on current project or Open .\Lab5.0_xxx.X to do exercises
- Try to add ADC module and conversion Potentiometer (VR2) voltage.
  - ADC Result ≥ 512, turn on LED6.
  - ADC Result < 512, turn off LED6.
- " Clock Source " ▸ " Fosc/4 "
- " Result Alignment " ▸ " Right "
- " Channel Select " ▸ " AN8(RB2) "
- " Conversion Time" ▸ " 200ms"

Let’s go!

Check Schematic



LED6	RC6
LED7	RC7
LED8	RB3
S5	RD1
S6	RD2

→ **0**:LED Light or **1**:LED Dark

MICROCHIP

Lab5 ADC Channel Periodically

Step 1

a **Remove "TODO 2.04" and Add code segment to your main**

```
main:
    ...
    // TODO 5.01
    BANKSEL TRISB
    bsf BANKMASK(TRISB),2
    BANKSEL ANSELB
    bsf BANKMASK(ANSELB),2
    ...
```

ANSELB: PORTB Analog Select Register → 1:Analog or 0:Digital

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
ANSB7	ANSB6	ANSB5	ANSB4	ANSB3	ANSB2	ANSB1	ANSB0
bit 7							bit 0

1 = Analog

TRISB: PORTB Tri-state Register → 1:Input or 0:Output

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
bit 7							bit 0

1 = Input

MICROCHIP

Lab5 ADC Channel Periodically

Step 2

a **Add code segment to your main**

```
main:
    ...
    // TODO 5.01
    ...
    // TODO 5.02
    BANKSEL ADCON1
    movlw 0b11000000 ;ADFM right; ADCS FOSC/4;
    movwf BANKMASK(ADCON1)
    BANKSEL ADCON0
    movlw 0b00100001 ;CHS AN8; ADON enable;
    movwf BANKMASK(ADCON0)
    ...
```

ADCON1: A/D Control Register 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
ADFM	ADCS<2:0>			-	ADNREF	ADPREF<1:0>	
bit 7							bit 0

1 = Right 100 = FOSC/4 → T_{AD} = 1us

ADCON0: A/D Control Register 0

1 = Enable ADC

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
-	CHS<4:0>					GO/nDONE	ADON
bit 7							bit 0

00100 = AN8

MICROCHIP

Lab5 ADC Channel Periodically

Step 3

a **Add code segment to your isr**

```
isr:
   ...
   // TODO 5.03
   BANKSEL ADCON0
   bsf GO_nDONE
   ...
exitISR:
```

ADCON0: A/D Control Register 0

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
-	CHS<4:0>					GO/nDONE	ADON
bit 7							bit 0

1 = Starts an A/D conversion cycle.

MICROCHIP

Lab5 ADC Channel Periodically

Step 4

a **Add code segment to your loop**

```
loop:
    ...
    // TODO 5.04
    BANKSEL ADCON0
    btfsc GO_nDONE
    goto ADC_END
    ...

ADC_END:
```

ADCON0: A/D Control Register 0

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R-0/0
-	CHS<4:0>					GO/nDONE	ADON
bit 7							bit 0

0 = A/D conversion completed.

Lab5 ADC Channel Periodically

Step 5.1

10 xxxxxxxx ≥ 512

0x xxxxxxxx < 512

a **Add code segment to your loop**

```
loop:
   // TODO 5.04
   ...
   // TODO 5.05.1
   BANKSEL ADRESH
   ;10bit ADC, ADFM right
   movlw 0b10 ; 512 = 0b 10 00000000
   subwf BANKMASK(ADRESH),w ; ADRESH – 512
   btfss CARRY ; nBorrow bit (0 → ADRESH < 512)
   goto LED6_Light   ; 1: ADRESH ≥ 512
   goto LED6_Dark   ; 0: ADRESH < 512
LED6_Light:
   BANKSEL LATC
   bcf LED6
   goto LED6_END
LED6_Dark:
   BANKSEL LATC
   bsf LED6
   goto LED6_END
LED6_END:

ADC_END:
```

ADRESH: ADC Result Register High (ADRESH), ADFM = 1

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
-	-	-	-	-	-	ADRES<9:8>	
bit 7							bit 0

ADRESL: ADC Result Register Low (ADRESL), ADFM = 1

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
ADRES<7:0>							
bit 7							bit 0

b **Program firmware to target board then observe result.**

MICROCHIP

Lab5 ADC Channel Periodically

Step 5.2 (additional)

1x xxxxxxxx ≥ 512

0x xxxxxxxx < 512

a **Add code segment to your loop**

```
loop:
   // TODO 5.04
   ...
   // TODO 5.05.2
   BANKSEL ADRESH
   ;10bit ADC, ADFM right
   btfsc BANKMASK(ADRESH),1 ; ≥512
   goto LED6_Light
   goto LED6_Dark


LED6_Light:
   BANKSEL LATC
   bcf LED6
   goto LED6_END
LED6_Dark:
   BANKSEL LATC
   bsf LED6
   goto LED6_END
LED6_END:

ADC_END:
```

ADRESH: ADC Result Register High (ADRESH), ADFM = 1

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
-	-	-	-	-	-	ADRES<9:8>	
bit 7							bit 0

ADRESL: ADC Result Register Low (ADRESL), ADFM = 1

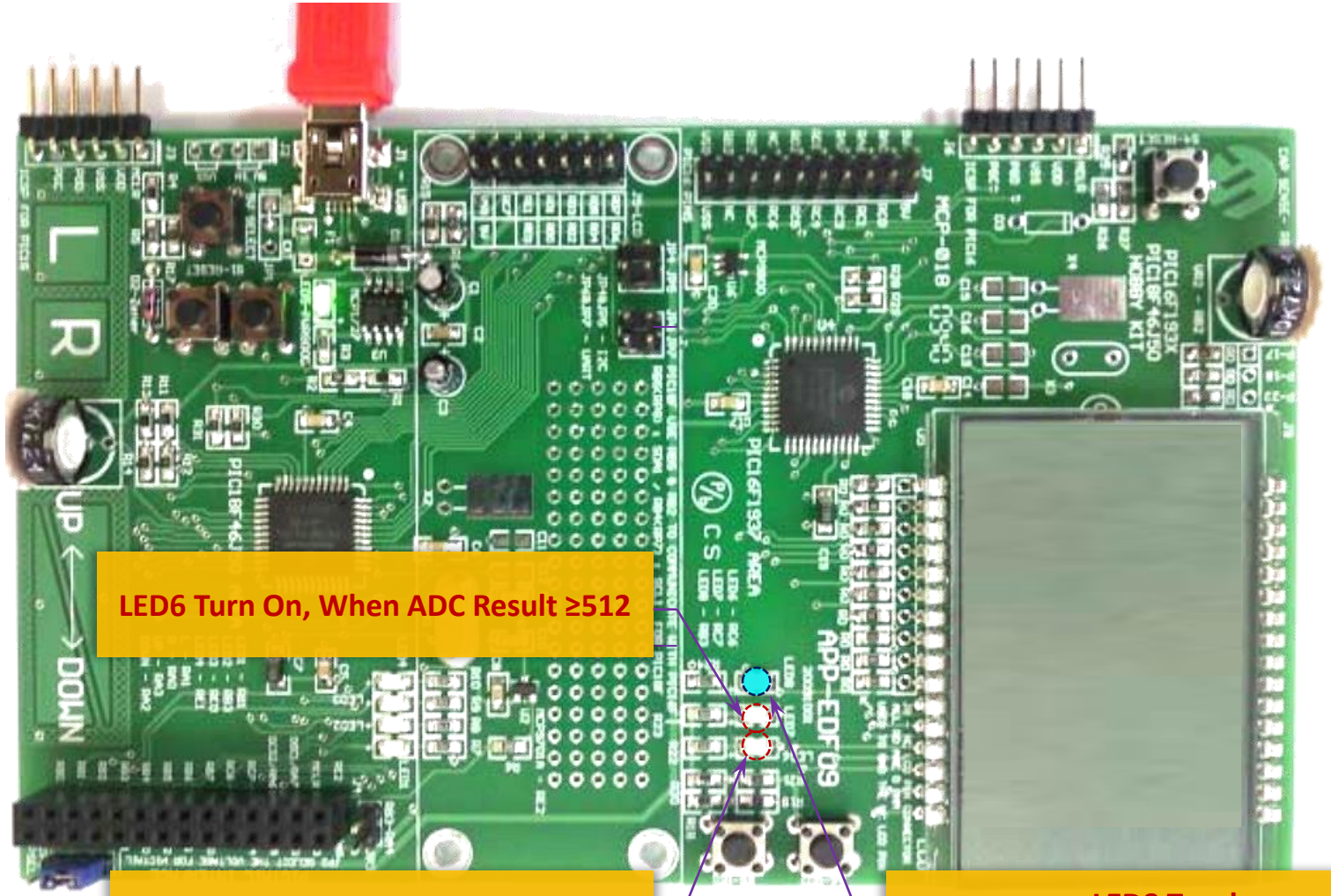
R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
ADRES<7:0>							
bit 7							bit 0

b **Program firmware to target board then observe result.**

MICROCHIP

Lab5 ADC Channel Periodically

Result



Discuss

ADC Result Register Format

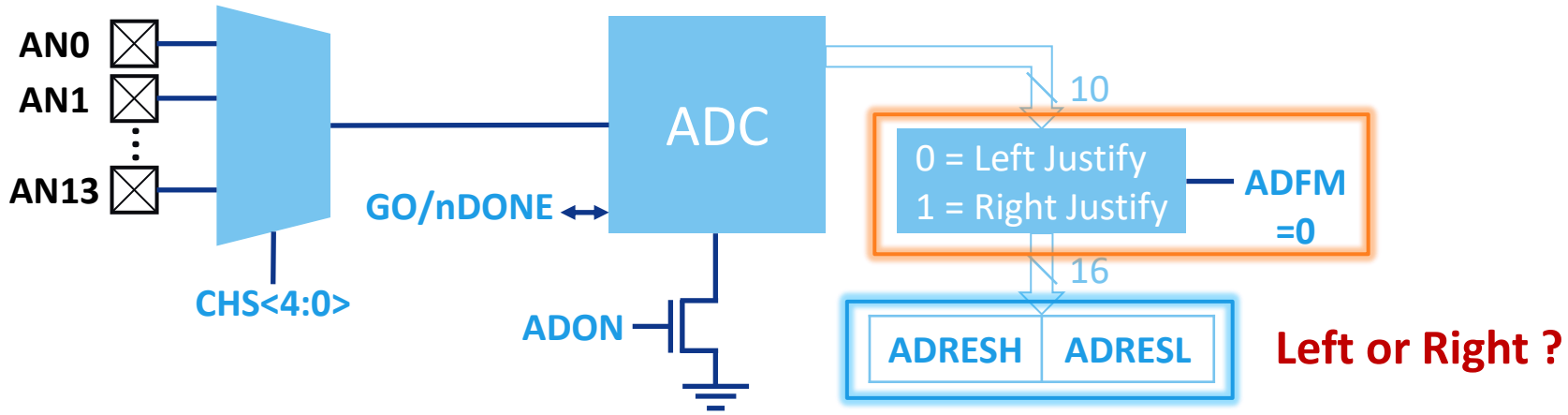
- **Discuss**
  - Select Format is **Left or Right?** Which format is used in which situation?

ADRESH and ADRESL, ADFM = 0 (Left)

10bit ADC Value									
ADRESH <9:2>								ADRESL <1:0>	
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

ADRESH and ADRESL, ADFM = 1 (Right)

10bit ADC Value									
ADRESH <1:0>		ADRESL <7:0>							
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0



MICROCHIP

ADC Result Register Format

- **Discuss**
  - Select Format is **Left or Right?** Which format is used in which situation?

ADRESH and ADRESL, ADFM = 0 (Left)

10bit ADC Value									
ADRESH <9:2>								ADRESL <1:0>	
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

ADRESH and ADRESL, ADFM = 1 (Right)

10bit ADC Value									
ADRESH <1:0>		ADRESL <7:0>							
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

The first value is taken.

Working Register							
WREG<7:0>							
1 byte							
b7	b6	b5	b4	b3	b2	b1	b0

The second value is taken.

Working Register							
WREG<7:0>							
1 byte							
b7	b6	b5	b4	b3	b2	b1	b0

Reducing the amount of instruction can increase efficiency.

Only 1 byte can handle at the same time.

MICROCHIP

ADC Result Register Format

- **Discuss**
  - Get the Rough of the ADC value to you need.

ADRESH and ADRESL, ADFM = 0

10bit ADC Value									
ADRESH <9:2>								ADRESL <1:0>	
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Use 1 byte to obtain the rough value of 0 to 1020 (0,4,8,…,1020).

Check 1		Value	Check 2		Check 3		Base	Real Value	
ADRESH (WREG Byte)	0~255	(0~255) x 4 = **0~1020** (0, 4, 8, … , 1020)	ADRESL Bit 1 (WREG Bit 1)	1	ADRESL Bit 0 (WREG Bit 0)	1	+3	(0~1020) + 3	= 0~1023
			ADRESL Bit 1 (WREG Bit 1)	1	ADRESL Bit 0 (WREG Bit 0)	0	+2	(0~1020) + 2	
			ADRESL Bit 1 (WREG Bit 1)	0	ADRESL Bit 0 (WREG Bit 0)	1	+1	(0~1020) + 1	
			ADRESL Bit 1 (WREG Bit 1)	0	ADRESL Bit 0 (WREG Bit 0)	0	+0	(0~1020) + 0	

ADRESH and ADRESL, ADFM = 1

		10bit ADC Value							
ADRESH <1:0>		ADRESL <7:0>							
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Check 1		Check 2		Base	Check 3		Real Value	
ADRESH Bit 1 (WREG Bit 1)	1	ADRESH Bit 0 (WREG Bit 0)	1	+768	ADRESL (WREG Byte)	0~255	768 + (0~255)	= 0~1023
ADRESH Bit 1 (WREG Bit 1)	1	ADRESH Bit 0 (WREG Bit 0)	0	+512			512 + (0~255)	
ADRESH Bit 1 (WREG Bit 1)	0	ADRESH Bit 0 (WREG Bit 0)	1	+256			256 + (0~255)	
ADRESH Bit 1 (WREG Bit 1)	0	ADRESH Bit 0 (WREG Bit 0)	0	+0			0 + (0~255)	

Use 1 byte to obtain the value of 0 to 255 (0,1,2,…,255).

MICROCHIP

ADC Result Register Format

11 xxxxxxxx ≥ 768
10 xxxxxxxx ≥ 512
01 xxxxxxxx ≥ 256
00 xxxxxxxx < 256

- **Discuss**
  - Get the value of Range to you need, divide the ADC values into four intervals.

ADRESH and ADRESL, ADFM = 0

		10bit ADC Value							
		ADRESH <9:2>						ADRESL <1:0>	
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Check 1		Check 2		Actual	Range	
ADRESH Bit 1 (WREG Bit1)	1	ADRESH Bit 0 (WREG Bit 0)	1	≥768	768~1023 (256)	Function 1
ADRESH Bit 1 (WREG Bit1)	1	ADRESH Bit 0 (WREG Bit 0)	0	≥512	512~767 (256)	Function 2
ADRESH Bit 1 (WREG Bit1)	0	ADRESH Bit 0 (WREG Bit 0)	1	≥256	256~511 (256)	Function 3
ADRESH Bit 1 (WREG Bit1)	0	ADRESH Bit 0 (WREG Bit 0)	0	<256	0~255 (256)	Function 4

ADRESH and ADRESL, ADFM = 1

		10bit ADC Value							
ADRESH <1:0>		ADRESL <7:0>							
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Check 1		Check 2		Actual	Range	
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	1	≥768	768~1023 (256)	Function 1
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	0	≥512	512~767 (256)	Function 2
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	1	≥256	256~511 (256)	Function 3
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	0	<256	0~255 (256)	Function 4

MICROCHIP

ADC Result Register Format

- **Discuss**
  - Get the value of Range to you need, divide the ADC values into eight intervals.

11 1xxxxxxx ≥ 896
11 0xxxxxxx ≥ 768
10 1xxxxxxx ≥ 640
10 0xxxxxxx ≥ 512
01 1xxxxxxx ≥ 384
01 0xxxxxxx ≥ 256
00 1xxxxxxx ≥ 128
00 0xxxxxxx < 128

ADRESH and ADRESL, ADFM = 0

	10bit ADC Value								
	ADRESH <9:2>							ADRESL <1:0>	
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Check 1		Check 2		Check 3		Actual	Range	
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	1	ADRESH Bit 7 (WREG Bit 5)	1	≥896	896~1023 (128)	Function 1
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	1	ADRESH Bit 7 (WREG Bit 5)	0	≥768	768~895 (128)	Function 2
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	0	ADRESH Bit 7 (WREG Bit 5)	1	≥640	640~767 (128)	Function 3
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	0	ADRESH Bit 7 (WREG Bit 5)	0	≥512	512~639 (128)	Function 4
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	1	ADRESH Bit 7 (WREG Bit 5)	1	≥384	384~511 (128)	Function 5
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	1	ADRESH Bit 7 (WREG Bit 5)	0	≥256	256~383 (128)	Function 6
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	0	ADRESH Bit 7 (WREG Bit 5)	1	≥128	128~255 (128)	Function 7
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	0	ADRESH Bit 7 (WREG Bit 5)	0	<128	0~127 (128)	Function 8

ADRESH and ADRESL, ADFM = 1

	10bit ADC Value								
ADRESH <1:0>			ADRESL <7:0>						
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Check 1		Check 2		Check 3		Actual	Range	
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	1	ADRESL Bit 7 (WREG Bit 7)	1	≥896	896~1023 (128)	Function 1
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	1	ADRESL Bit 7 (WREG Bit 7)	0	≥768	768~895 (128)	Function 2
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	0	ADRESL Bit 7 (WREG Bit 7)	1	≥640	640~767 (128)	Function 3
ADRESH Bit 9 (WREG Bit 7)	1	ADRESH Bit 8 (WREG Bit 6)	0	ADRESL Bit 7 (WREG Bit 7)	0	≥512	512~639 (128)	Function 4
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	1	ADRESL Bit 7 (WREG Bit 7)	1	≥384	384~511 (128)	Function 5
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	1	ADRESL Bit 7 (WREG Bit 7)	0	≥256	256~383 (128)	Function 6
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	0	ADRESL Bit 7 (WREG Bit 7)	1	≥128	128~255 (128)	Function 7
ADRESH Bit 9 (WREG Bit 7)	0	ADRESH Bit 8 (WREG Bit 6)	0	ADRESL Bit 7 (WREG Bit 7)	0	<128	0~127 (128)	Function 8

Use 1 byte to obtain the 2 ~ 256 (2^N) interval.

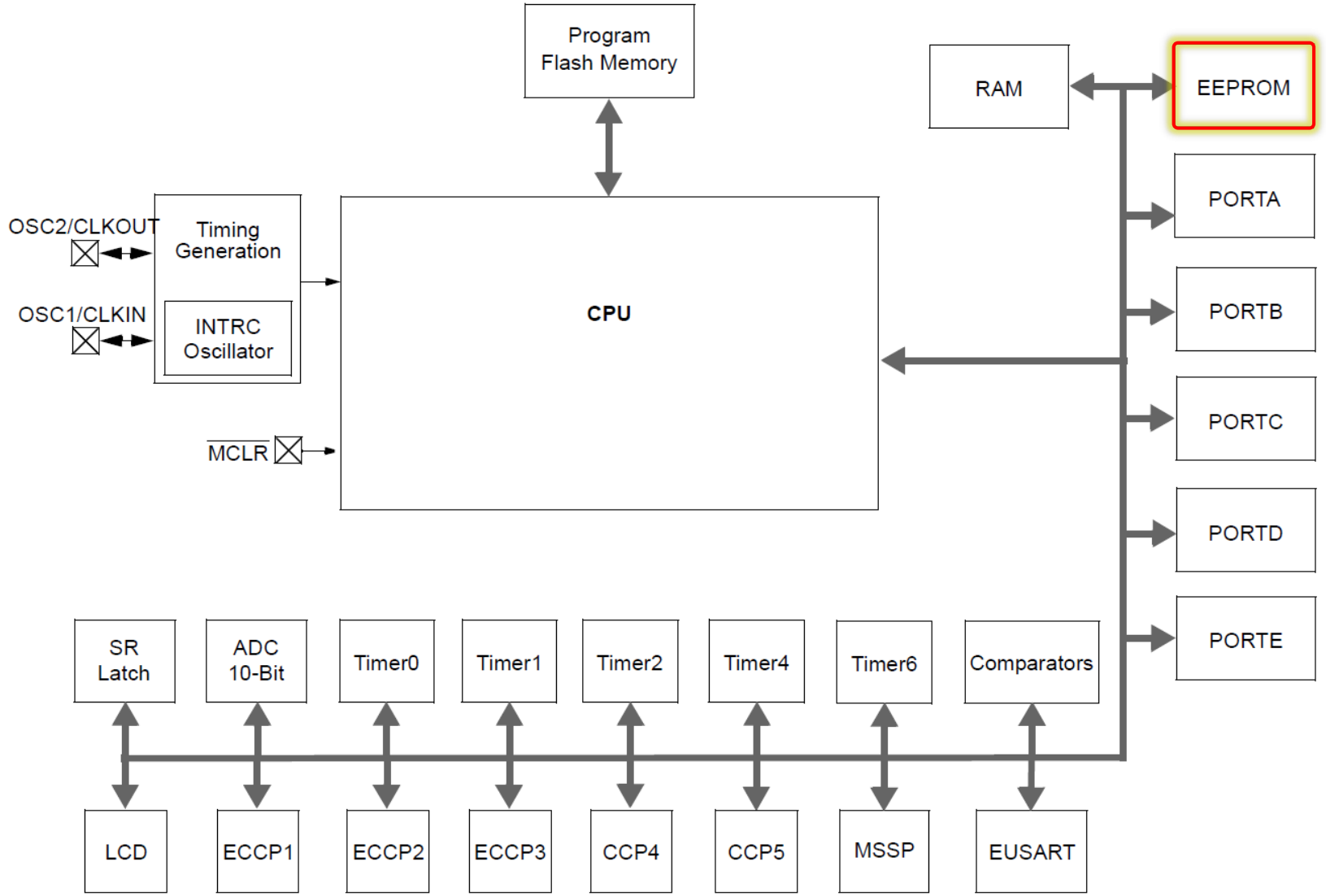
MICROCHIP

EEPROM Architecture

What’s Data EEPROM ?

- **The data EEPROM is a high-endurance, byte addressable array that has been optimized for the storage of frequently changing information (1M write EEPROM endurance).**
- **EEPROM can be written to byte-by-byte.**

PIC16(L)F1934/6/7 BLOCK DIAGRAM



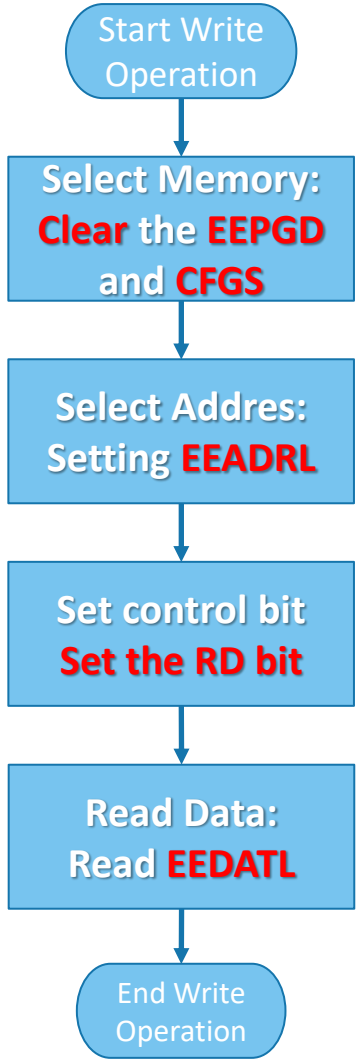
MICROCHIP

Data EEPROM Control

- The data EEPROM are indirectly addressed through the Special Function Registers (SFRs), there are four SFRs used to access these data EEPROM memories:
  - EECON1 (EEPROM Control 1 Register)
  - EECON2 (EEPROM Control 2 Register)
  - EEADRL (EEPROM Address Low Byte Register)
  - EEDATL (EEPROM Data Low Byte Register)
- When selecting a EEPROM address, only the LSB(EEADRL and EEDATL) available, this can address up to a maximum of 256 bytes of data EEPROM.
- Control bits RD and WR initiate read and write, respectively. These bits cannot be cleared, only set, in software.
- To Read or Write an EEPROM data location, the user must clear the EEPGD(Flash Program/Data EEPROM Memory Select bit) and CFGS(Flash Program/Data EEPROM or Configuration Select bit) control bits of the EECON1 register.
- The WREN(Program/Erase Enable bit) bit, when set, will allow a write operation to occur.

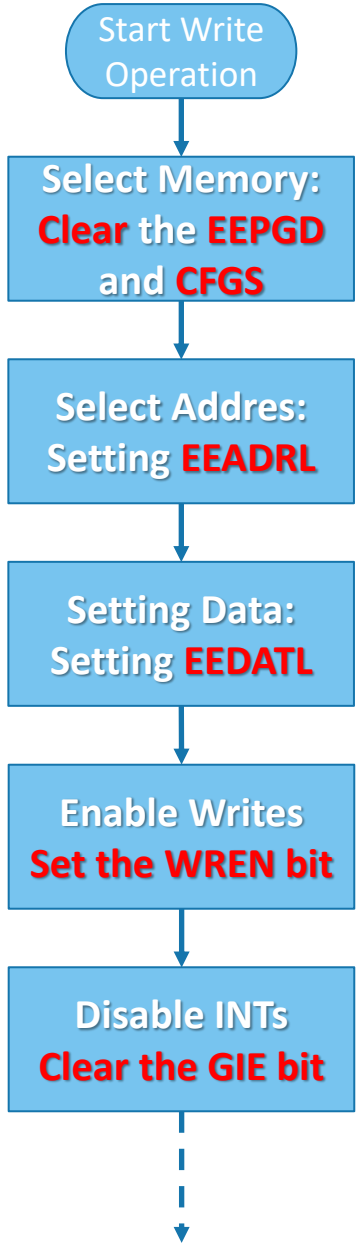
Reading the Data EEPROM Memory

- To read an EEPROM data location, the user must first write the address to the EEADRL register, and then set control bit RD.
- The data is available at the very next cycle, in the EEDATL register; therefore, it can be read in the next instruction.
- EEDATL will hold this value until another read or until it is written to by the user (during a write operation).



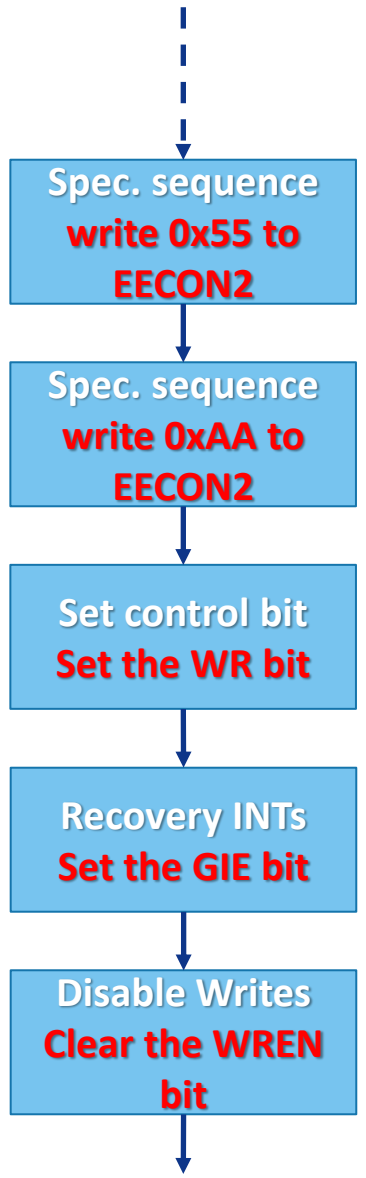
Writing to the Data EEPROM Memory

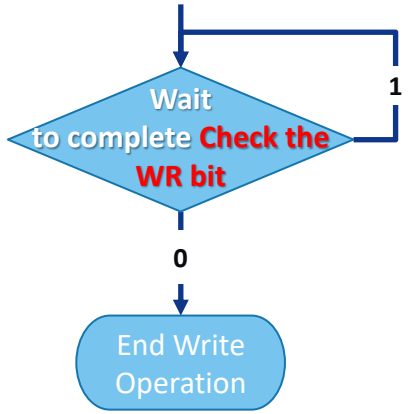
- To write an EEPROM data location, the user must first write the address to the EEADRL register and the data to the EEDATL register.
- WREN bit in EECON1 must be set to enable write. The user should keep the WREN bit clear at all times, except when updating EEPROM.
- Interrupts should be disabled during this code segment.



Writing to the Data EEPROM Memory

- Then the user must follow a specific sequence to initiate the write for each byte (write 0x55 to EECON2, write 0xAA to EECON2, then set the WR bit).
- An EEPROM byte write automatically erases the location and writes the new data (erase before write).
- At the completion of the write cycle, the WR bit is cleared in hardware.





MICROCHIP

EEPROM Control Register

EECON1: EEPROM Control 1 Register

R/W-0/0	R/W-0/0	R/W-0/0	R/S/HC-0/0	R/W-x/q	R/W-0/0	R/S/HC-0/0	R/S/HC-0/0
EEPGD	CFGS	LWLO	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

EECON2: EEPROM Control 2 Register

W-0/0	W-0/0	W-0/0	W-0/0	W-0/0	W-0/0	W-0/0	W-0/0
EEPROM Control Register 2							
bit 7							bit 0

EEPROM Address Byte Register

EEDATH: EEPROM Address High Byte Register

U-1	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
-	EEADR<14:8>						
bit 7							bit 0

EEADRL: EEPROM Address Low Byte Register

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
EEADR<7:0>							
bit 7							bit 0

EEPROM Data Byte Register

EEDATH: EEPROM Data High Byte Register

U-0	U-0	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
-	-	EEDAT<13:8>					
bit 7							bit 0

EEDATL: EEPROM Data Low Byte Register

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
EEDAT<7:0>							
bit 7							bit 0

Lab6 EEPROM Read and Write

- Base on current project or Open .\Lab6.0_xxx.X to do exercises
- Try to set RD2 to digital input as get button status.
- Try use button to Increase the value of the address 0x00.
  - S6 Presse then Release -> Increase the value of the EEPROM with in 0~3.
  - Show the binary value of the EEPROM with LED6 and LED7.
- RB3 ▸ LED(LED1).
- RD2 ▸ Button(S6).
- RC6 ▸ LED(LED6).
- RC7 ▸ LED(LED7).

Let’s go!

MICROCHIP

Lab6 EEPROM Read and Write

- **Create variables and subroutines to read and write EEPROM Memory.**

Variables:

```
PSECT udata_bankx
EE_ADDR :
   DS 1
EE_DATA :
   DS 1
```

EE_ADDR: 1 Byte Customize Variable

EE_ADDR<7:0>
bit 7 ... bit 0

EE_DATA: 1 Byte Customize Variable

EE_DATA<7:0>
bit 7 ... bit 0

Subroutines:

EE_Read: Read a byte from address of the EEPROM.

EE_Write: Write a byte to the address of the EEPROM.

PSECT code
EE_Read:

```
...
; EE_ADDR → EEADRL
...
...
; EEDATL → EE_DATA
return
```

PSECT code
EE_Write:

```
...
; EE_ADDR → EEADRL
; EE_DATA → EEDATL
...
...
return
```

EEADRL: EEPROM Address Low Byte Register

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
EEADR<7:0>							
bit 7							bit 0

EEDATL: EEPROM Data Low Byte Register

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
EEDAT<7:0>							
bit 7							bit 0

MICROCHIP

Lab6 EEPROM Read and Write

Step 1

	BANK 3
180h	INDF0
181h	INDF1
182h	PCL
183h	STATUS
184h	FSR0L
185h	FSR0H
186h	FSR1L
187h	FSR1H
188h	BSR
189h	WREG
18Ah	PCLATH
18Bh	INTCON
18Ch	ANSELA
18Dh	ANSELB
18Eh	—
18Fh	ANSELD(1)
190h	ANSELE(1)
191h	EEADRL
192h	EEADRH
193h	EEDATL
194h	EEDATH
195h	EECON1
196h	EECON2
197h	—
198h	—
199h	RCREG
19Ah	TXREG
19Bh	SPBRGL
19Ch	SPBRGH
19Dh	RCSTA
19Eh	TXSTA
19Fh	BAUDCON
1A0h	General Purpose Register 80 Bytes
1EFh	
1F0h	Accesses 70h – 7Fh
1FFh	

a **Remove “TODO 2.04, 5.04, 5.05”, and Add code segment to your main.s**

```
#include <xc.inc>
…

// TODO 6.01
PSECT udata_bank3
EE_ADDR:
   DS 1
EE_DATA:
   DS 1
…
```

EE_ADDR: 1 Byte Customize Variable

EE_ADDR<7:0>	
bit 7	bit 0

EE_DATA: 1 Byte Customize Variable

EE_DATA<7:0>	
bit 7	bit 0

MICROCHIP

Lab6 EEPROM Read and Write

Step 2

	BANK 3
180h	INDF0
181h	INDF1
182h	PCL
183h	STATUS
184h	FSR0L
185h	FSR0H
186h	FSR1L
187h	FSR1H
188h	BSR
189h	WREG
18Ah	PCLATH
18Bh	INTCON
18Ch	ANSELA
18Dh	ANSELB
18Eh	—
18Fh	ANSELD(1)
190h	ANSELE(1)
191h	EEADRL
192h	EEADRH
193h	EEDATL
194h	EEDATH
195h	EECON1
196h	EECON2
197h	—
198h	—
199h	RCREG
19Ah	TXREG
19Bh	SPBRGL
19Ch	SPBRGH
19Dh	RCSTA
19Eh	TXSTA
19Fh	BAUDCON
1A0h – 1EFh	General Purpose Register 80 Bytes
1F0h – 1FFh	Accesses 70h – 7Fh

a **Add code segment to your main.s**

```
// TODO 6.01
...
// TODO 6.02
PSECT code
EE_Read:
   movlb 3    ;Set the BANK to 3
   bcf EEPGD
   bcf CFGS
   movf BANKMASK(EE_ADDR),w
   movwf BANKMASK(EEADRL)
   bsf RD      ;EE Read
   movf BANKMASK(EEDATL),w
   movwf BANKMASK(EE_DATA)
   return
```

EECON1: EEPROM Control 1 Register

R/W-0/0	R/W-0/0	R/W-0/0	R/S/HC-0/0	R/W-x/q	R/W-0/0	R/S/HC-0/0	R/S/HC-0/0
EEPGD	CFGS	LWLO	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

= 0 = 0 = 1

EEADRL: EEPROM Address Low Byte Register

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
EEADR<7:0>							
bit 7							bit 0

= EE_ADDR

EEDATL: EEPROM Data Low Byte Register

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
EEDAT<7:0>							
bit 7							bit 0

= EE_DATA

MICROCHIP

Lab6 EEPROM Read and Write

Step 3

	BANK 3
180h	INDF0
181h	INDF1
182h	PCL
183h	STATUS
184h	FSR0L
185h	FSR0H
186h	FSR1L
187h	FSR1H
188h	BSR
189h	WREG
18Ah	PCLATH
18Bh	INTCON
18Ch	ANSELA
18Dh	ANSELB
18Eh	—
18Fh	ANSELD(1)
190h	ANSELE(1)
191h	EEADRL
192h	EEADRH
193h	EEDATL
194h	EEDATH
195h	EECON1
196h	EECON2
197h	—
198h	—
199h	RCREG
19Ah	TXREG
19Bh	SPBRGL
19Ch	SPBRGH
19Dh	RCSTA
19Eh	TXSTA
19Fh	BAUDCON
1A0h	General Purpose Register 80 Bytes
1EFh	
1F0h	Accesses 70h – 7Fh
1FFh	

a **Add code segment to your main.s**

```
// TODO 6.02
...
// TODO 6.03
PSECT code
EE_Write:
   movlb 3    ;Set the BANK to 3
   bcf EEPGD
   bcf CFGS
   movf BANKMASK(EE_ADDR),w
   movwf BANKMASK(EEADRL)
   movf BANKMASK(EE_DATA),w
   movwf BANKMASK(EEDATL)
   bsf WREN ;Enable writes
   bcf GIE     ;Disable INTs.
   ...
```

EECON1: EEPROM Control 1 Register

R/W-0/0	R/W-0/0	R/W-0/0	R/S/HC-0/0	R/W-x/q	R/W-0/0	R/S/HC-0/0	R/S/HC-0/0
EEPGD	CFGS	LWLO	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

= 0 = 0 = 1

EEADRL: EEPROM Address Low Byte Register

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
EEADR<7:0>							
bit 7							bit 0

= EE_ADDR

EEDATL: EEPROM Data Low Byte Register

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
EEDAT<7:0>							
bit 7							bit 0

= EE_DATA

MICROCHIP

Lab6 EEPROM Read and Write

Step 3

a **Add code segment to your main.s**

```
// TODO 6.02
...
// TODO 6.03
PSECT code
EE_Write:
   ...
   movlw 0x55
   movwf BANKMASK(EECON2)
   movlw 0xAA
   movwf BANKMASK(EECON2)
   bsf WR      ;Set WR bit to begin write
   bsf GIE     ;Reset Interrupts
   bcf WREN    ;Disable writes
   btfsc WR    ;Wait for write to complete
   goto $-1
   return      ;complete
```

EECON1: EEPROM Control 1 Register

R/W-0/0	R/W-0/0	R/W-0/0	R/S/HC-0/0	R/W-x/q	R/W-0/0	R/S/HC-0/0	R/S/HC-0/0
EEPGD	CFGS	LWLO	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

= 1 (cleared at the complete)

= 0 (disable after write)

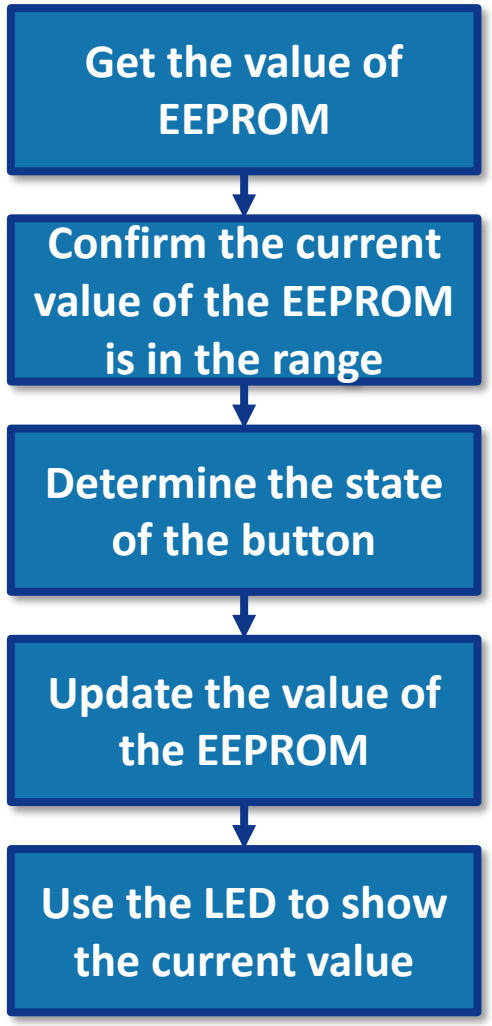
EECON2: EEPROM Control 2 Register

W-0/0	W-0/0	W-0/0	W-0/0	W-0/0	W-0/0	W-0/0	W-0/0
EEPROM Control Register 2							
bit 7							bit 0

→ 0x55 → 0xAA

Lab6 EEPROM Read and Write

- **Press the button to modify the value of the EEPROM Memory, and then show the result by the LED.**

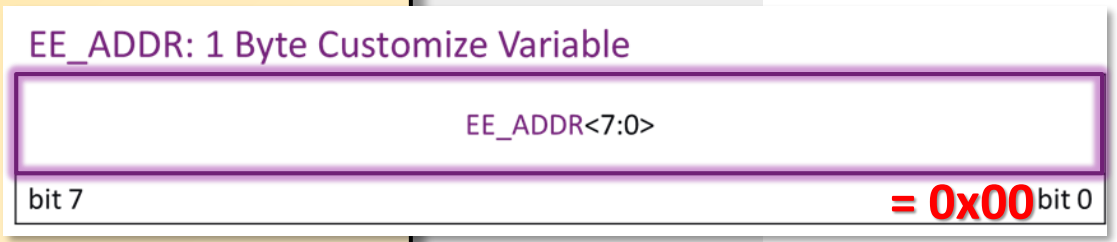


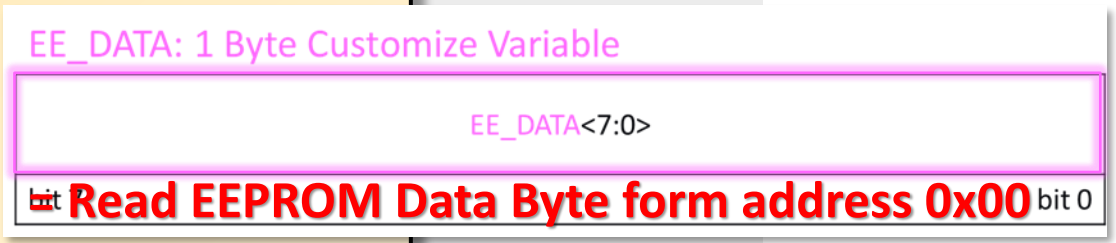
Lab6 EEPROM Read and Write

Step 4

a **Add code segment to your main**

```
// TODO 6.04
BANKSEL EE_ADDR
movlw 0x00
movwf BANKMASK(EE_ADDR)
call EE_Read

// TODO 6.05
;Check if value is out of range
BANKSEL EE_DATA
movlw 3+1
subwf BANKMASK(EE_DATA),w ; EE_DATA - (3+1)
btfss CARRY   ; nBorrow bit (0 → EE_DATA ≤ 3)
goto loop                 ; 0: EE_DATA ≤ 3
clrf BANKMASK(EE_DATA)  ; 1: EE_DATA > 3
call EE_Write

loop:
...
```





Lab6 EEPROM Read and Write

Step 5

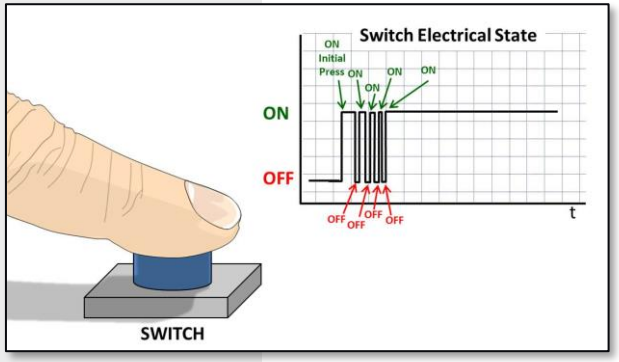
Determine the state of the button

a Add code segment to your loop

```
loop:
   ...
// TODO 6.06
   BANKSEL PORTD
   btfsc S6
   goto BTN_S6_Release
   ;Debunce
   movlw 10 ;10*(1ms)=10ms
   call delay_ms
   BANKSEL PORTD
   btfsc S6
   goto BTN_S6_Release
   ; Add S6 Press application code here.
   ...

BTN_S6_Release:
```

PORTD: PORTD Register → 1:Pin is > V_{IH} or 0:Pin is < V_{IL}

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
bit 7						S6	bit 0



Lab6 EEPROM Read and Write

Update the value of the EEPROM

Step 6

a **Add code segment to your loop**

```
// TODO 6.06
...
 ; Add S6 Press application code here.
// TODO 6.07
BANKSEL EE_DATA
incf BANKMASK(EE_DATA), f
;Check if value is out of range.
movlw 3+1 ;0:00,1:01,2:10,3:11
subwf BANKMASK(EE_DATA),w ; EE_DATA - (3+1)
btfsc CARRY                ; nBorrow bit (0 → EE_DATA ≤ 3)
clrf BANKMASK(EE_DATA)     ; 1: EE_DATA > 3
call EE_Write              ; 0: EE_DATA ≤ 3
;Wait the S6 Release
BANKSEL PORTD
btfss S6
goto $-1

BTN_S6_Release:
```

EE_DATA: 1 Byte Customize Variable

EE_DATA<7:0>	
bit 7	bit 0

increase each time a key is pressed.

Lab6 EEPROM Read and Write

Use the LED to show the current value

- **Use binary to represent the result.**

EE_DATA: 1 Byte Customize Variable

EE_DATA<7:0>	bit 1	bit 0
bit 7		bit 0

EE_DATA			LED7	LED6
Byte	bit1	bit0		
3	1	1	Light	Light
2	1	0	Light	Dark
1	0	1	Dark	Light
0	0	0	Dark	Dark

LATC: PORTC Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7							bit 0

LED7 LED6

Lab6 EEPROM Read and Write

Use the LED to show the current value

Step 7

a Add code segment to your loop

```
    // TODO 6.07
    ...

    // TODO 6.08
    BANKSEL EE_DATA
    btfsc BANKMASK(EE_DATA), 0
    goto LED6_Light
    goto LED6_Dark
LED6_Light:
    BANKSEL LATC
    bcf LED6
    goto LED6_END
LED6_Dark:
    BANKSEL LATC
    bsf LED6
    goto LED6_END
LED6_END:
```

EE_DATA: 1 Byte Customize Variable

EE_DATA<7:0>	bit 0
bit 7	bit 0

Check the Bit

LATC: PORTC Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7							bit 0

LED6

EE_DATA			LED7	LED6
Byte	bit1	bit0		
3	1	1	Light	Light
2	1	0	Light	Dark
1	0	1	Dark	Light
0	0	0	Dark	Dark

MICROCHIP

Lab6 EEPROM Read and Write

Use the LED to show the current value

Step 8

a Add code segment to your loop

```
    // TODO 6.08
    ...

    // TODO 6.09
    BANKSEL EE_DATA
    btfsc BANKMASK(EE_DATA), 1
    goto LED7_Light
    goto LED7_Dark
LED7_Light:
    BANKSEL LATC
    bcf LED7
    goto LED7_END
LED7_Dark:
    BANKSEL LATC
    bsf LED7
    goto LED7_END
LED7_END:
```

EE_DATA: 1 Byte Customize Variable

EE_DATA<7:0>							bit 1
bit 7							bit 0

Check the Bit

LATC: PORTC Data Latch Register → 1:High or 0:Low

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7							bit 0

LED7

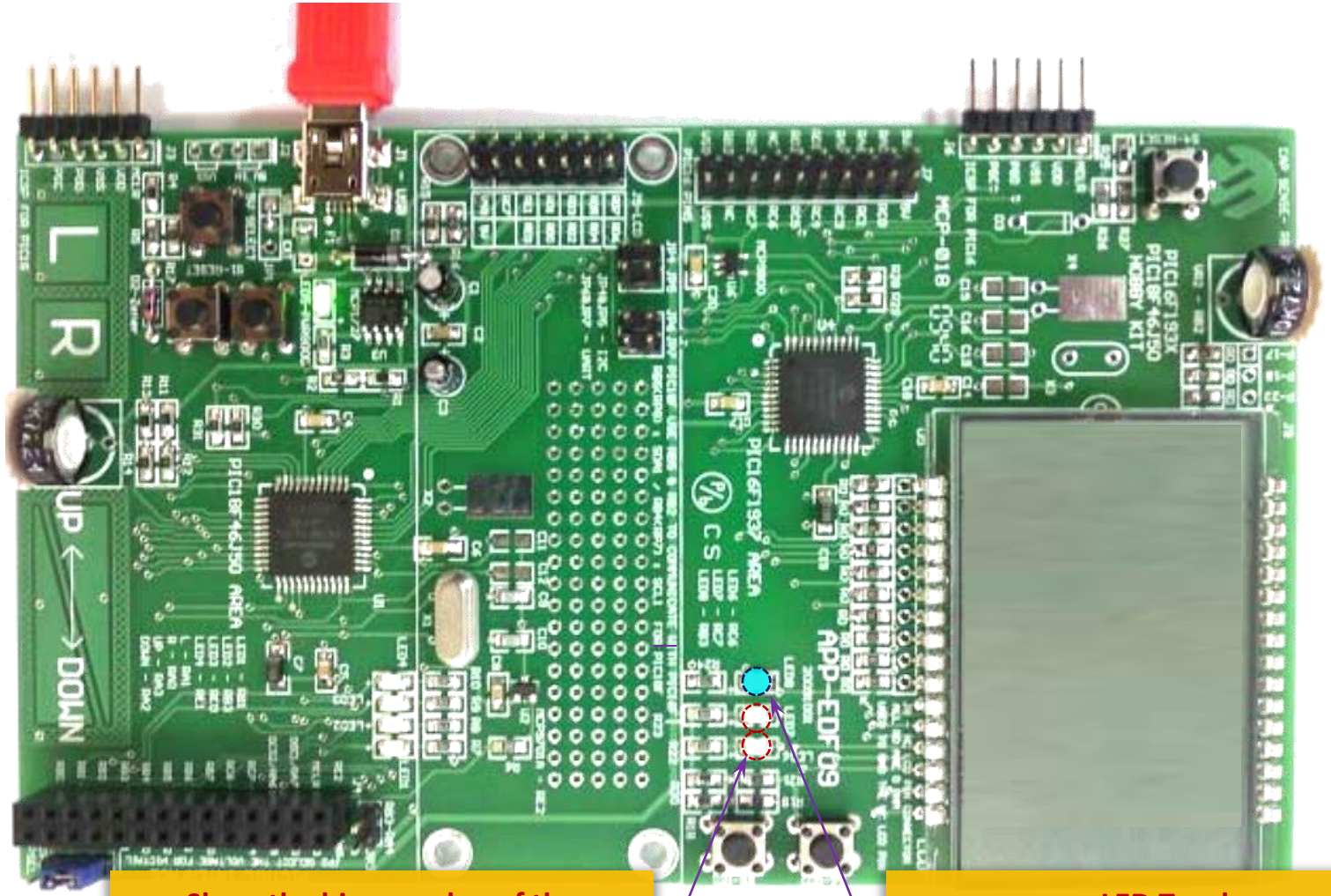
EE_DATA			LED7	LED6
Byte	bit1	bit0		
3	1	1	Light	Light
2	1	0	Light	Dark
1	0	1	Dark	Light
0	0	0	Dark	Dark

b Program firmware to target board then observe result.

MICROCHIP

Lab6 EEPROM Read and Write

Result



Show the binary value of the EEPROM with LED6 and LED7.

LED Toggle,
Interval Period use "Timer2" callback.

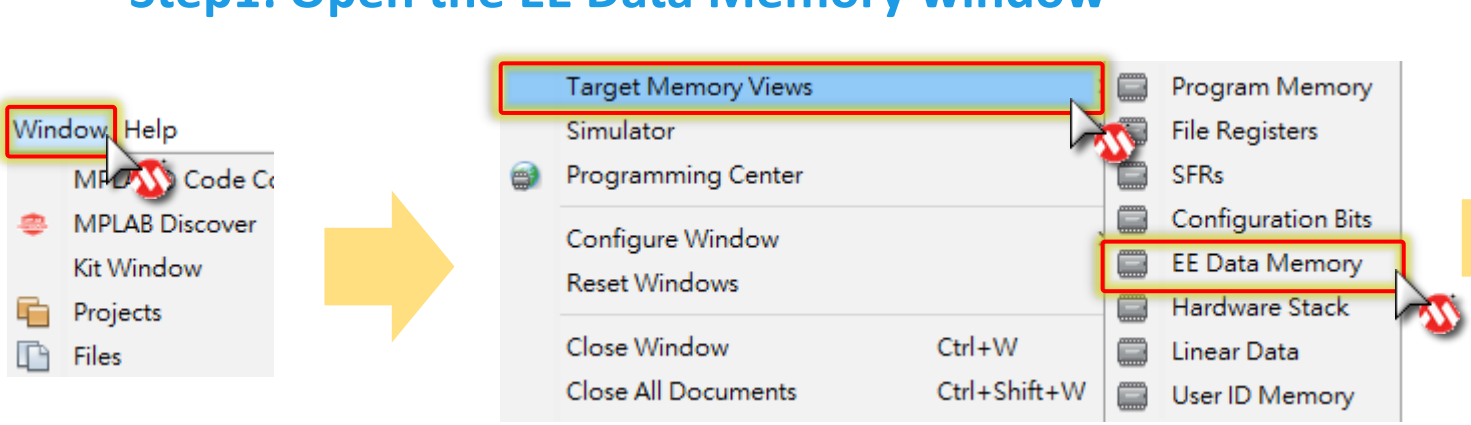
MICROCHIP

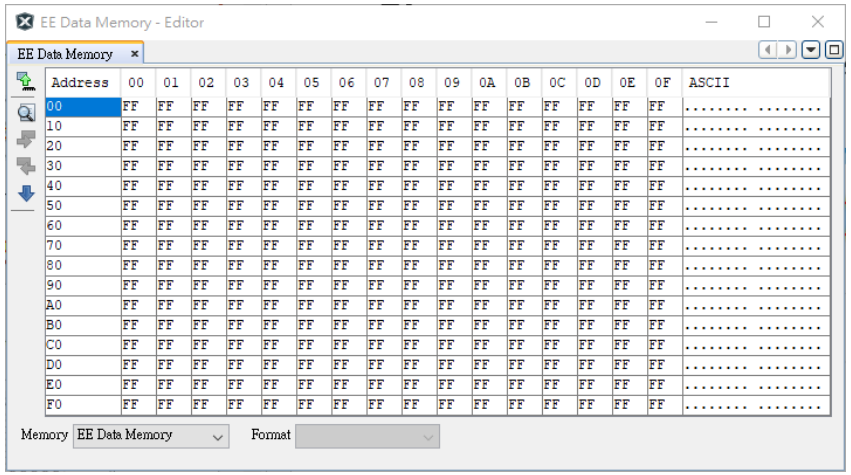
Discuss

EE Data Memory

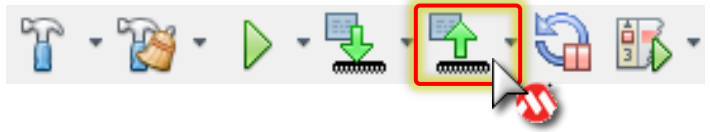
- **Discuss**
  - How to Read the current value of EEPROM Data Memory?

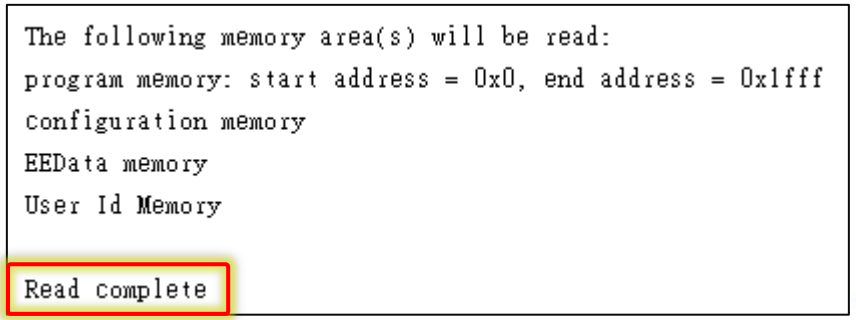
  - **Step1: Open the EE Data Memory window**





  - **Step2: Click the Read Device Memory button**

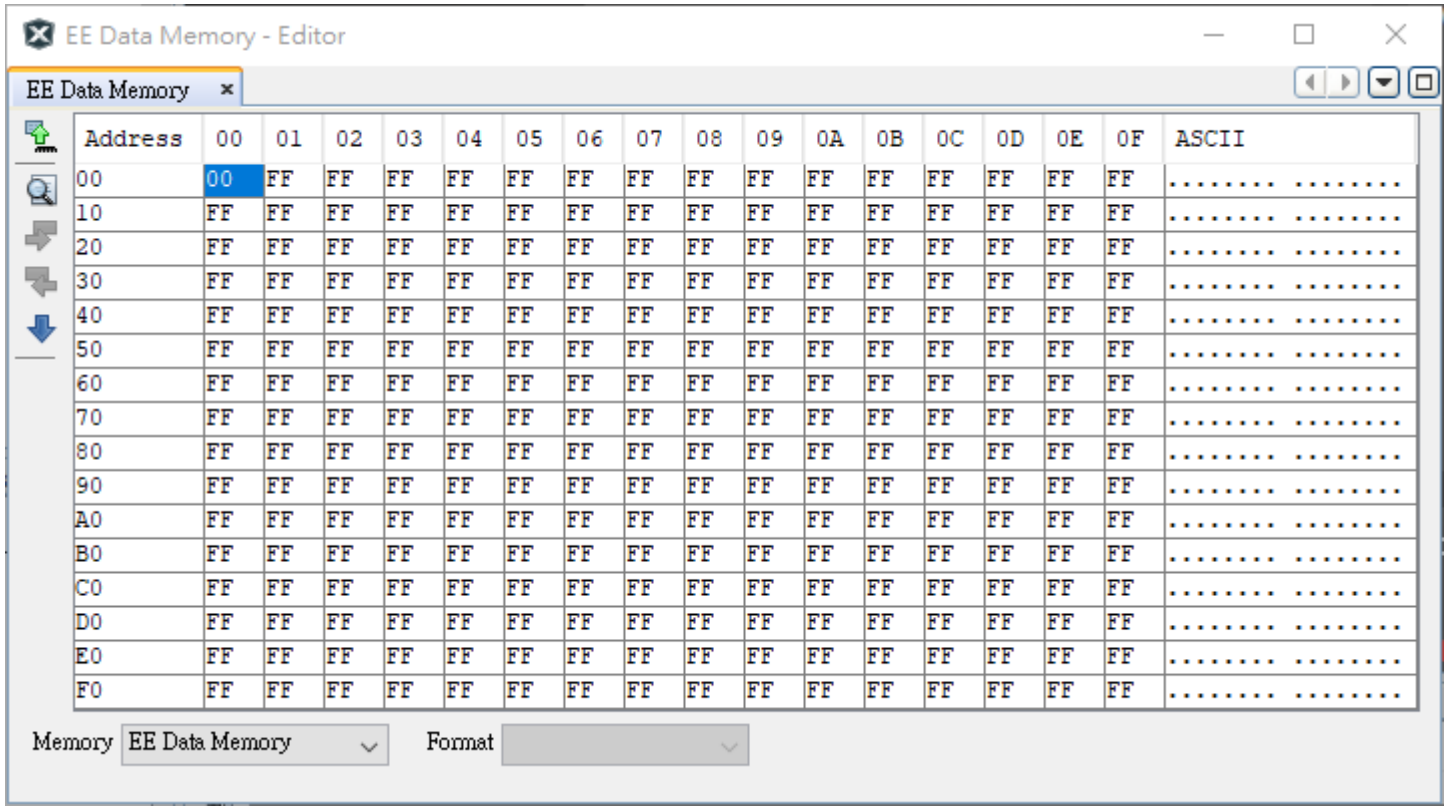




EE Data Memory

- **Discuss**
  - How to Read the current value of EEPROM Data Memory?

  - **Step3: Read the current EEPROM Data Memory value of the address**



Microchip Trademarks

Overview

Microchip Technology Incorporated's products are marketed and sold under one or more of the following trademarks belonging to Microchip or one of Microchip's subsidiaries. Questions regarding use of these trademarks should be addressed to the Microchip's Legal Department via email: legal.department@microchip.com.

The following are registered trademarks of Microchip in the U.S.A. and other countries:

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA

The following are registered trademarks of Microchip in the U.S.A:

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL

The following are registered trademarks of Microchip in other countries: BeaconThings, GestIC, Silicon Storage Technology

The following are trademarks of Microchip in the U.S.A. and other countries:

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA

The following is a service mark of Microchip in the U.S.A.: SQTP

The following are logos of Microchip in the U.S.A. and other countries: The Adaptec logo, Frequency on Demand, Silicon Storage Technology, Symmcom, and Trusted Time

The following is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries: GestIC

MICROCHIP