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 (1/3)

- IDE, Compilers, MCC and Development Tools Introduction
- APP1618-SD EVB Introduction
- Getting Started with First Project
  - Lab0 First Project (MCC)
- GPIO Architecture
  - Lab1.2 GPIO Output (MCC)
  - Lab2.2 GPIO Input (MCC)
- Oscillator Architecture
  - Lab3.2 Clock PRIPLL (MCC)
- Interrupt Architecture
- Timer Architecture
  - Lab4.2 Timer2 Interrupt (MCC)

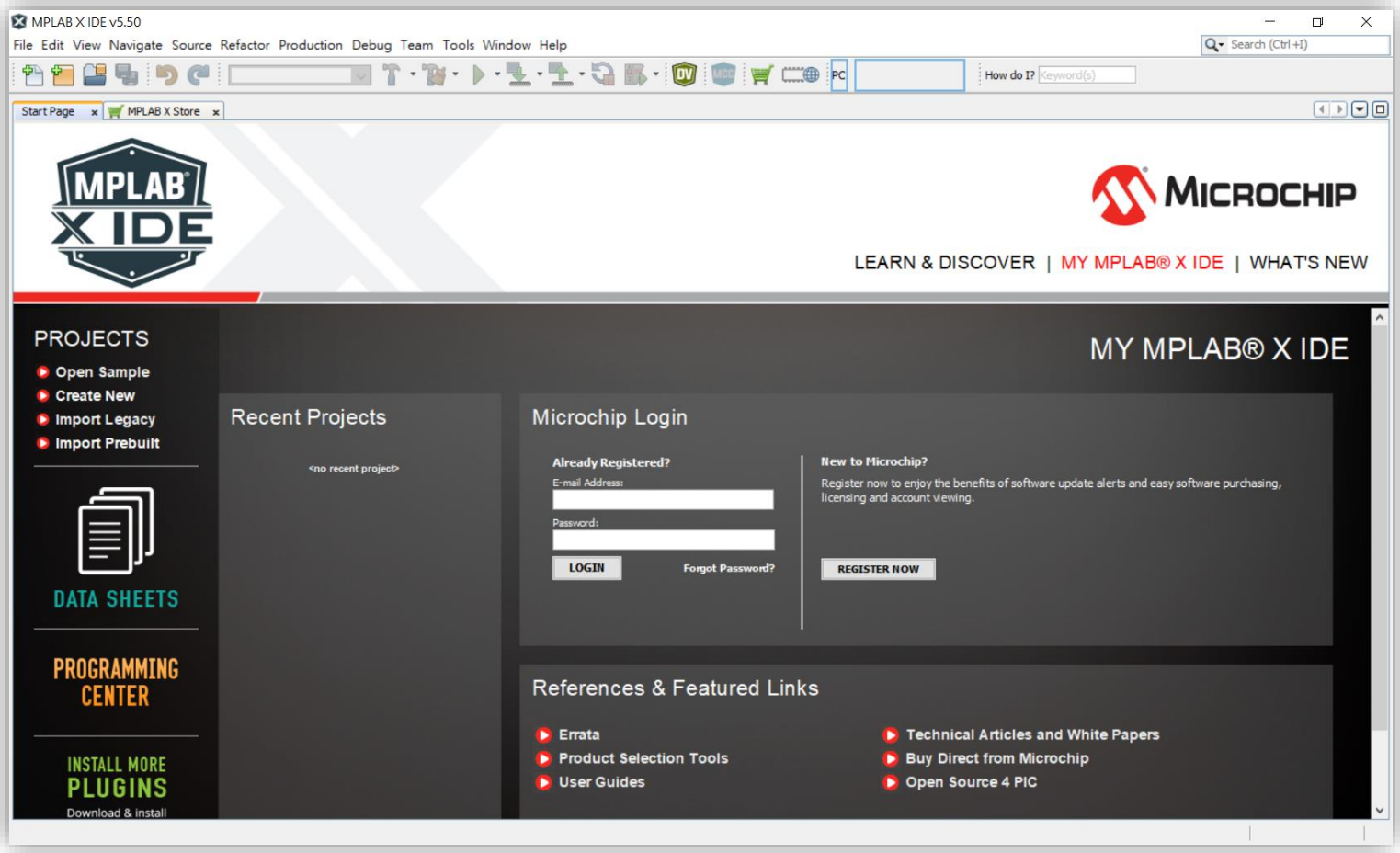
Major Course (2/3)

- ADC Architecture
  - 💻Lab5.2 ADC Channel Periodically (MCC)
- LCD Architecture
  - 💻Lab7 LCD Display (MCC)
- UART Architecture
  - 💻Lab8 UART Printf (MCC)
  - 💻Lab9 UART Communication (MCC)
- I2C Architecture
  - 💻Lab10 Temperature Sensor (MCP9800) Operation (MCC)

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.**



MICROCHIP

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.**

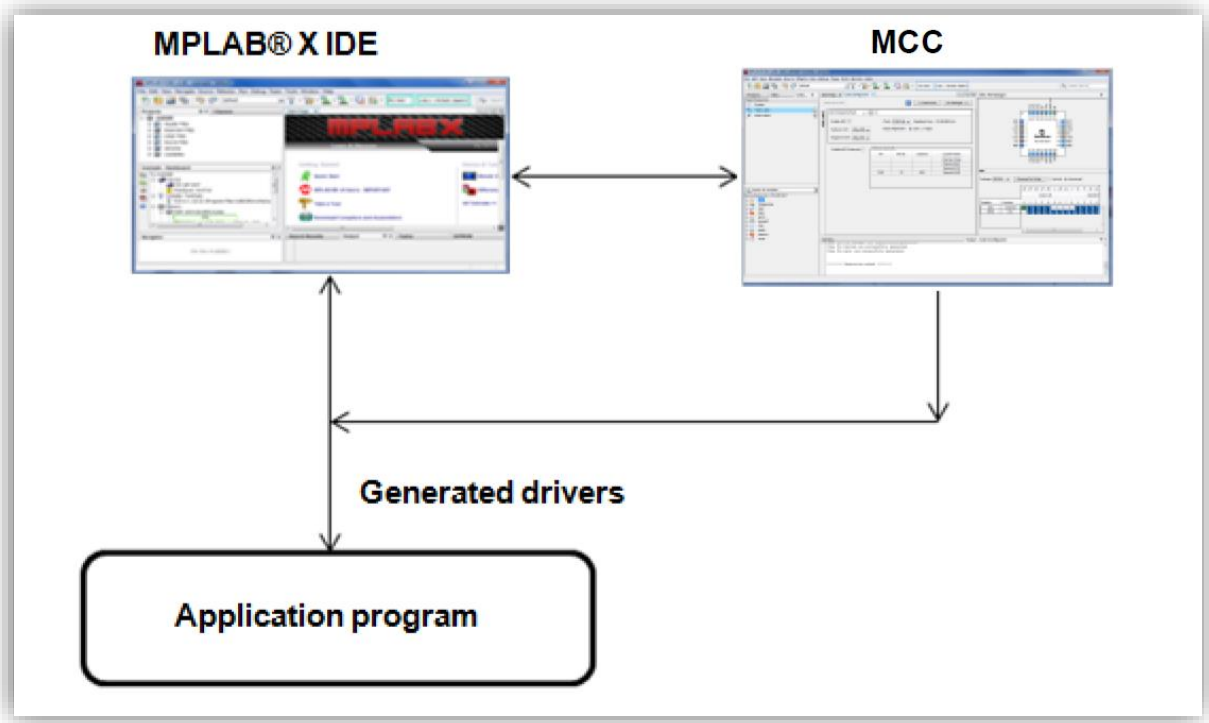
License Type	Installs On	# of Activations	# of Users	Wait Time Between Users	HPA
Workstation License	Workstation	3	1	None	Yes
Subscription License	Workstation	1	1	None	No
Site License	Network	1	Varies by Seat	None	Yes
Network Server License	Network	1	Unlimited	One Hour	Yes
Virtual Machine *	Network	1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

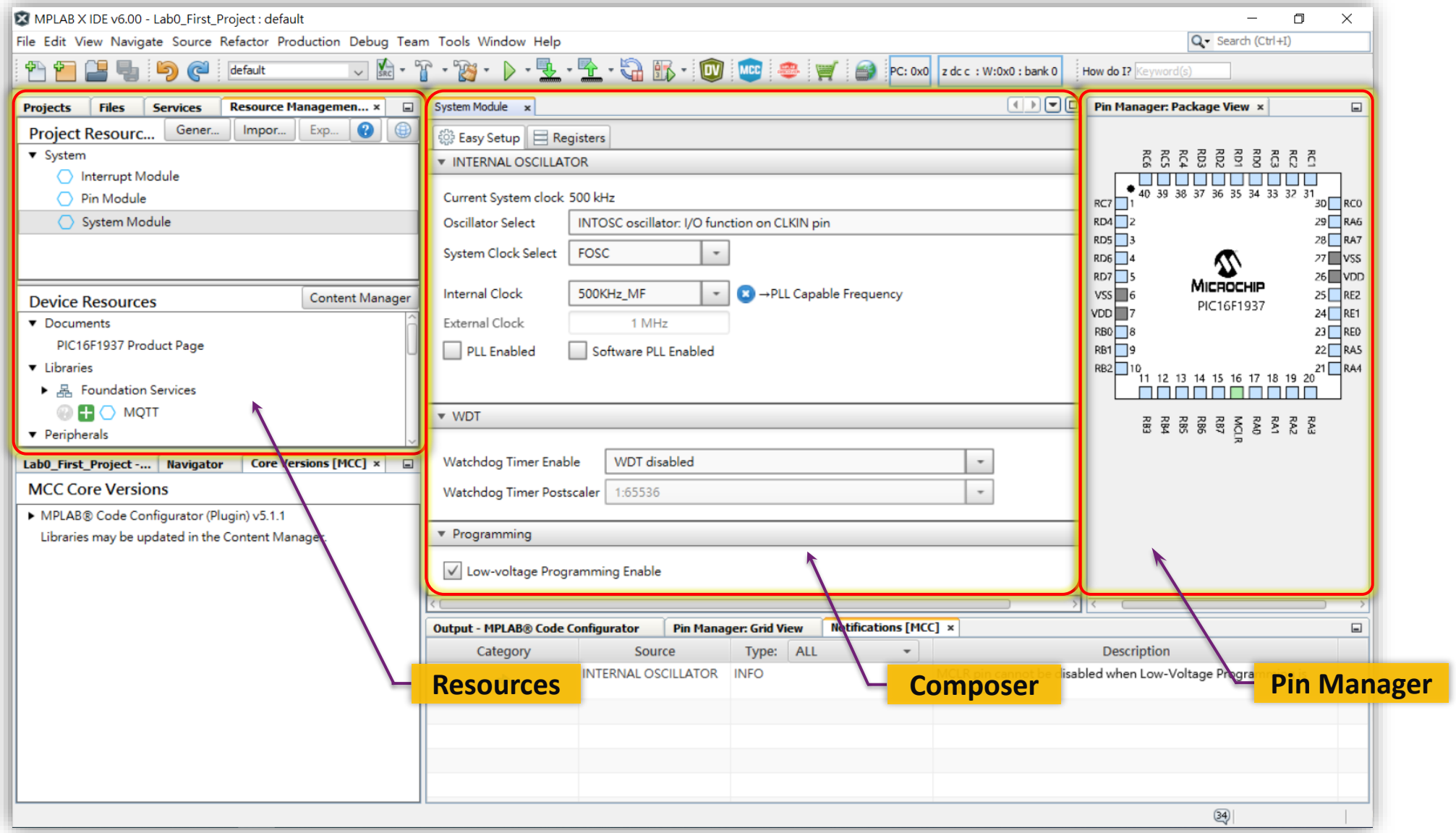
- **MPLAB® Code Configurator - MCC is a free, graphical programming environment that generates seamless, easy-to-understand C code to be inserted into your project.**
- **Supports 8-bits, 16-bits and limited 32-bits PIC® microcontrollers.**



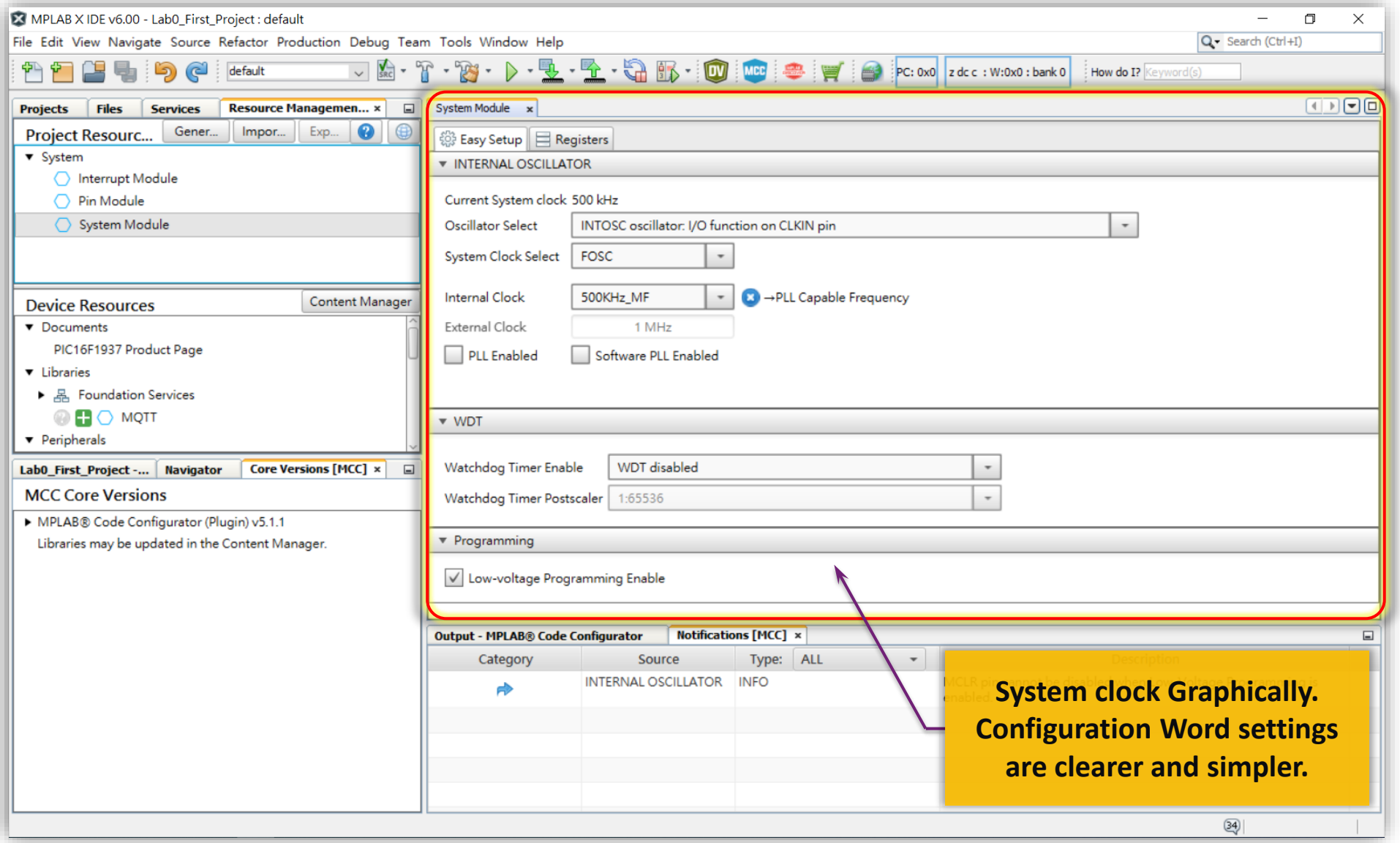
MPLAB® CODE CONFIGURATOR

MICROCHIP

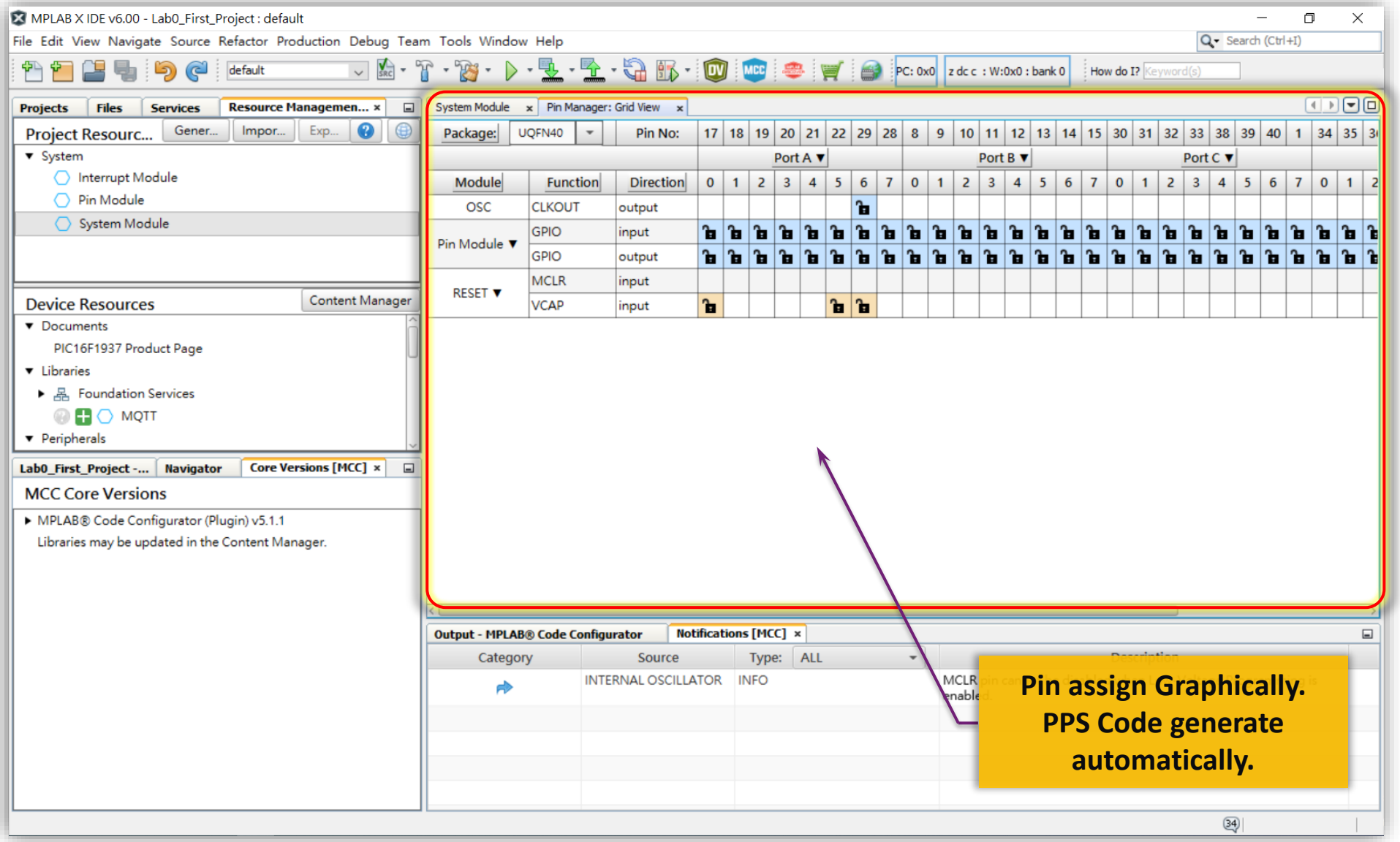
MCC's GUI Quick View



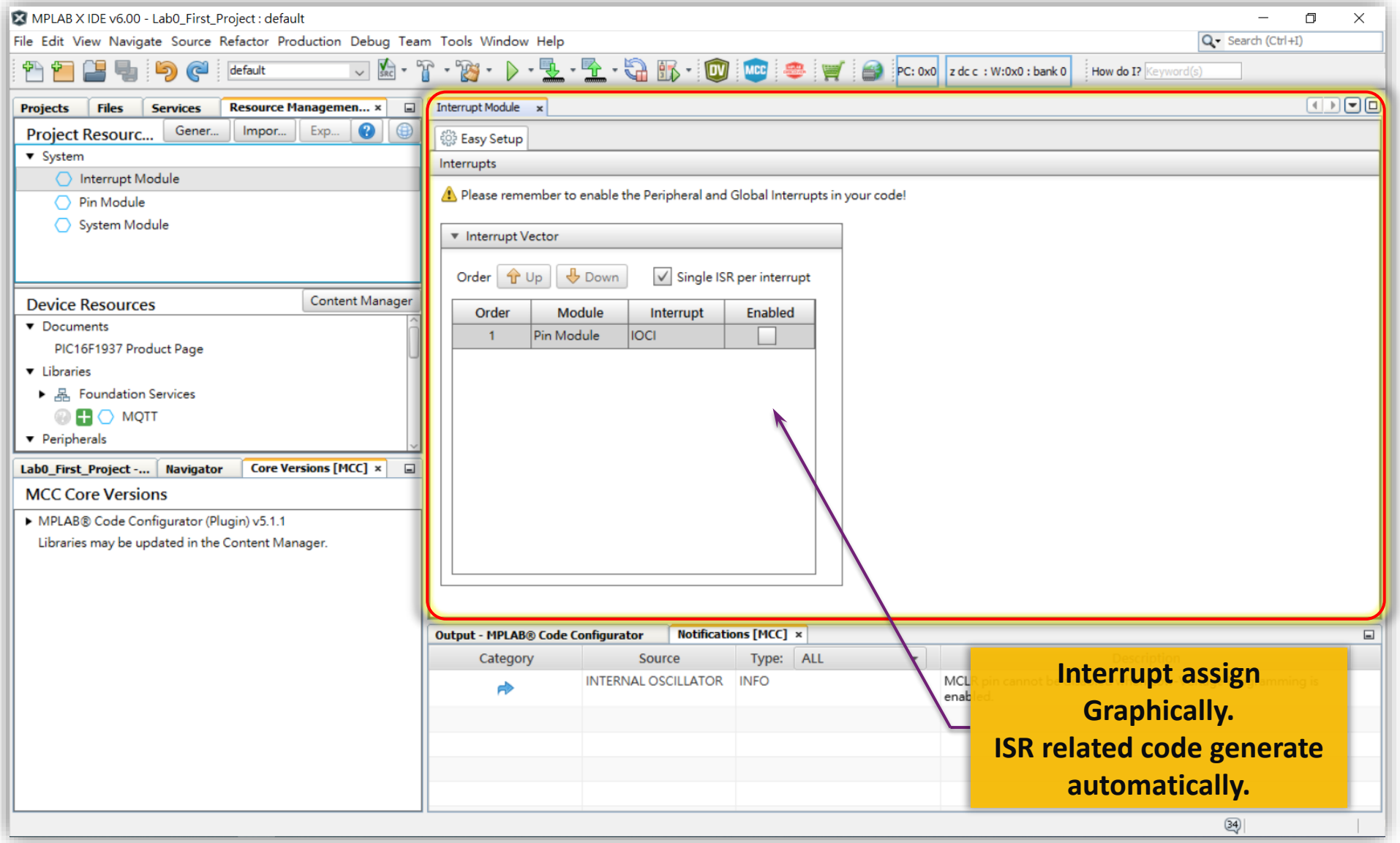
MCC's GUI Quick View



MCC's GUI Quick View



MCC’s GUI Quick View



MCC’s GUI Quick View

More ?

www.microchip.com/mcc

Programmer/Debugger

ICD4 (Part No. DV164045)

- **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

PICKit4 (Part No. PG164140)

- **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.**

$76.99

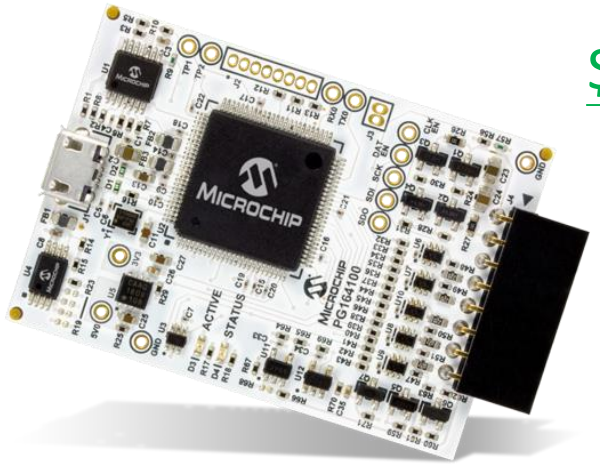
- **Features**
  - Matches silicon clocking speed.
  - Target voltage of 1.20V to 5.5V.
  - Can supply up to 50mA of power to the target.
  - Minimal current consumption at <100µA from target.
  - Portable USB-powered and RoHS-compliant.
  - 8-pin single in-line header.
  - Backward compatible for demo boards, headers and target systems using 2-wire JTAG and ICSP.
  - Option to be self-powered from the target (2.7V to 5.5V).

MICROCHIP

Programmer/Debugger

SNAP (Part No. PG164100)

- **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



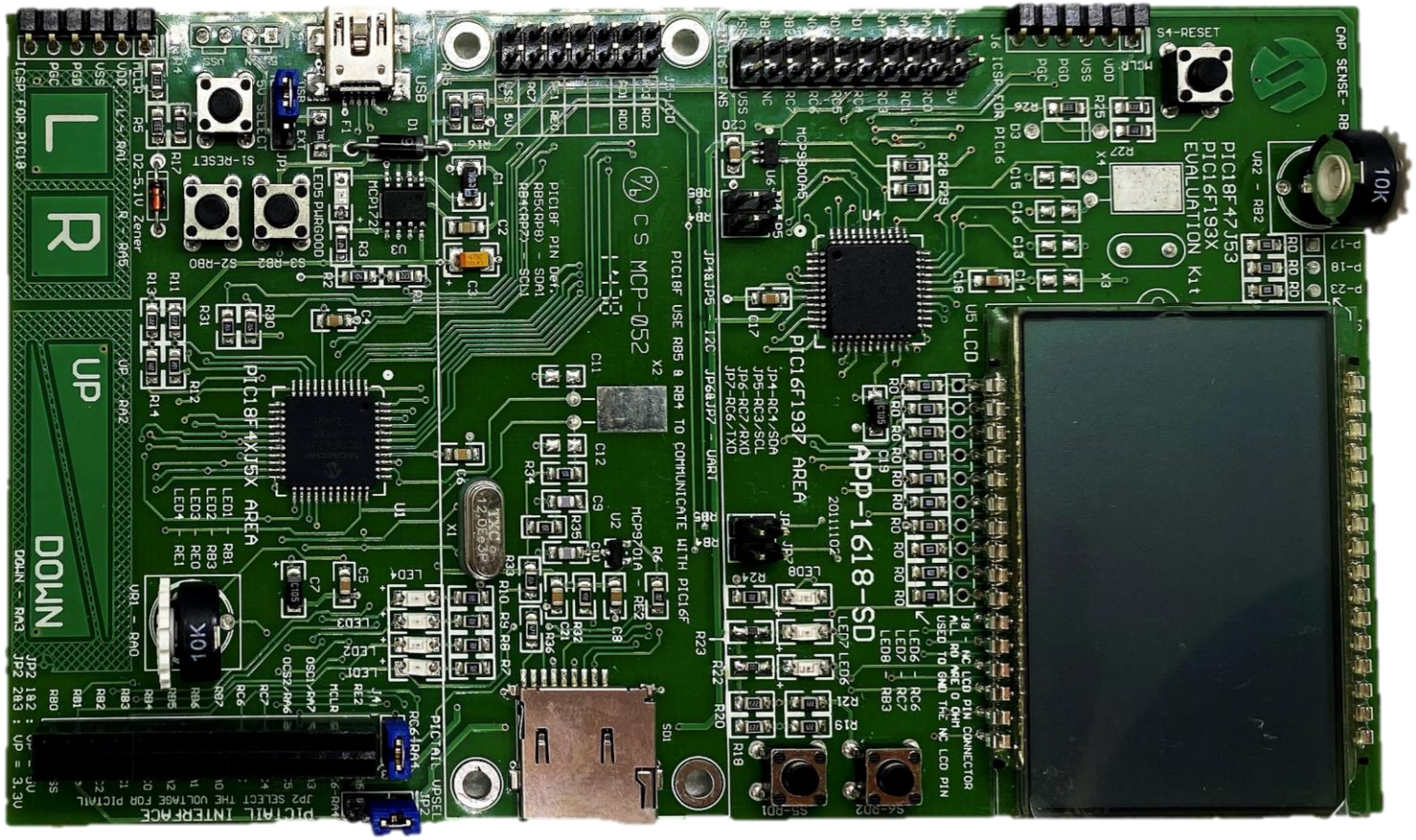
$34.09

MICROCHIP

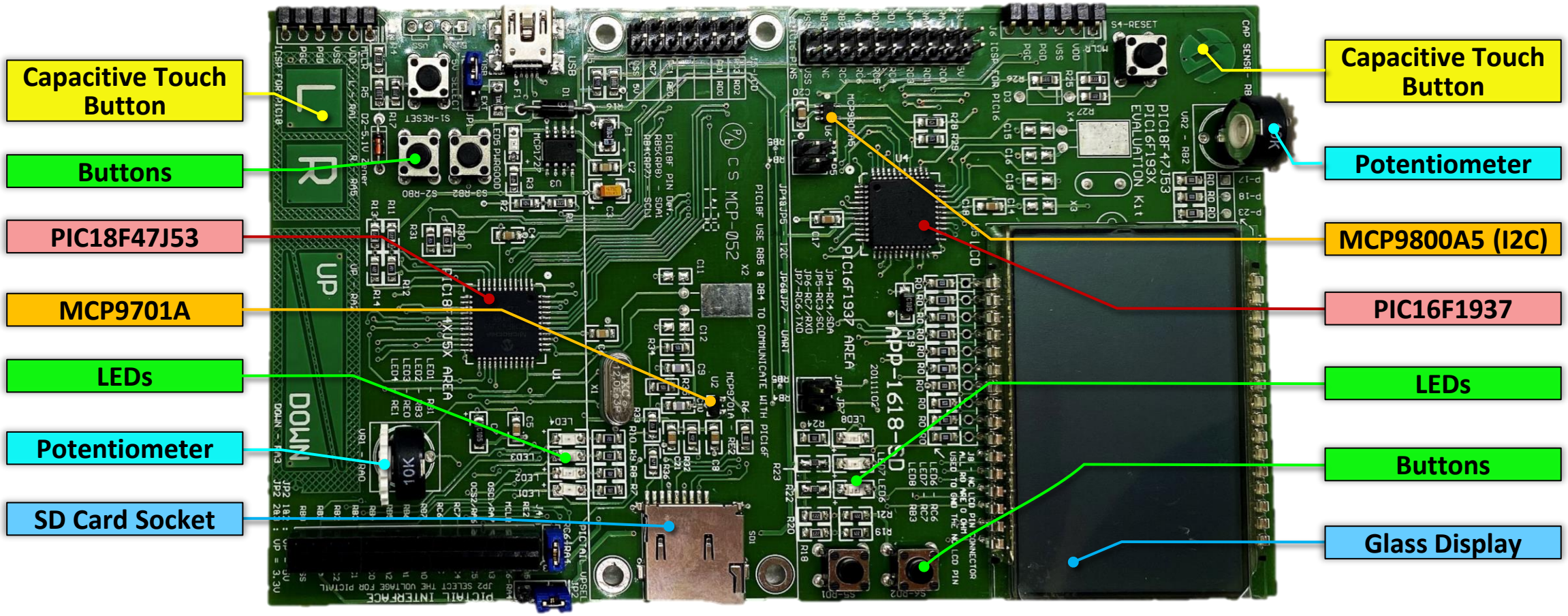
APP1618-SD EVB Introduction

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

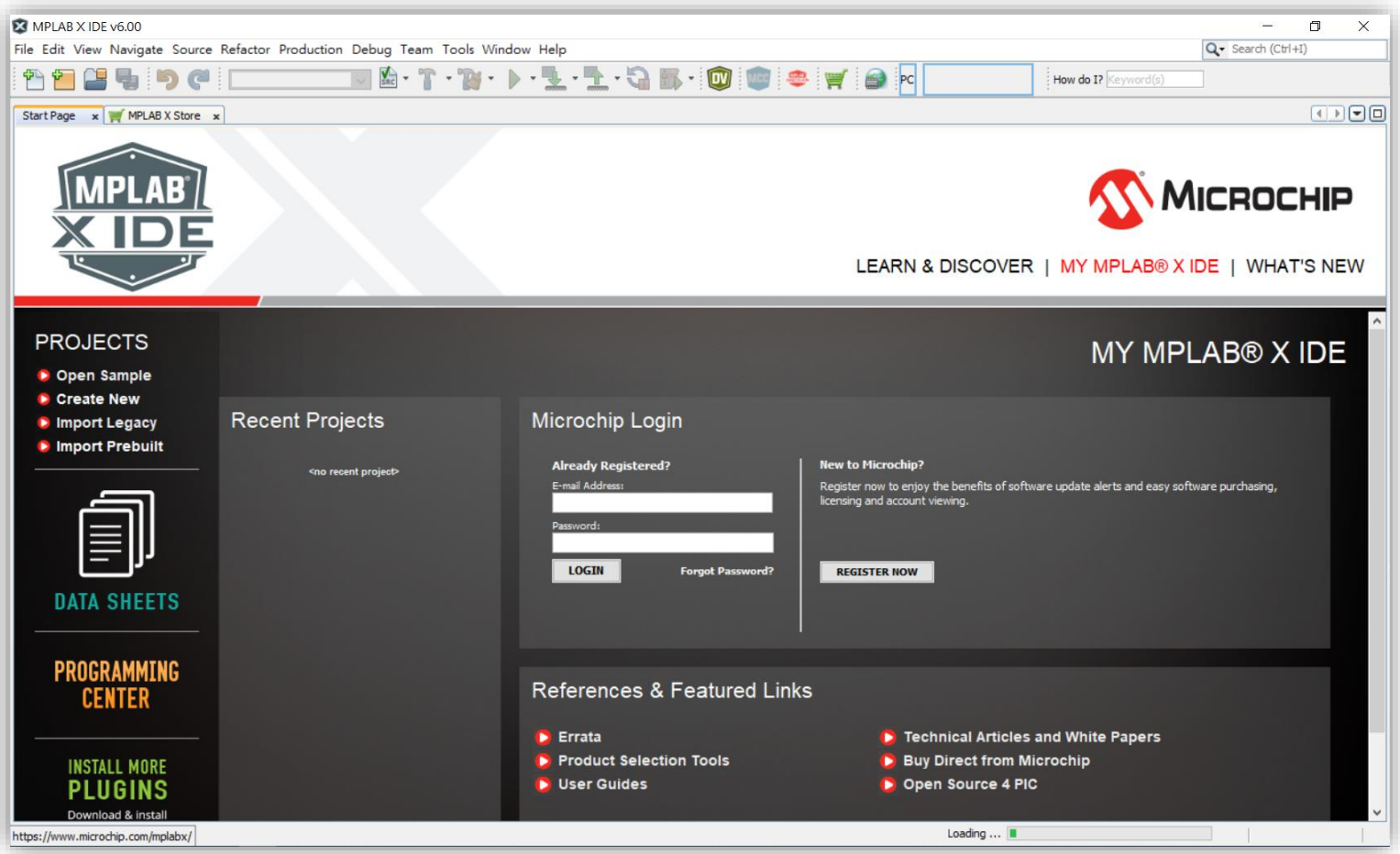


First MCC Project

Lab Preparation

Step 1

- Open **MPLAB X IDE** firstly.

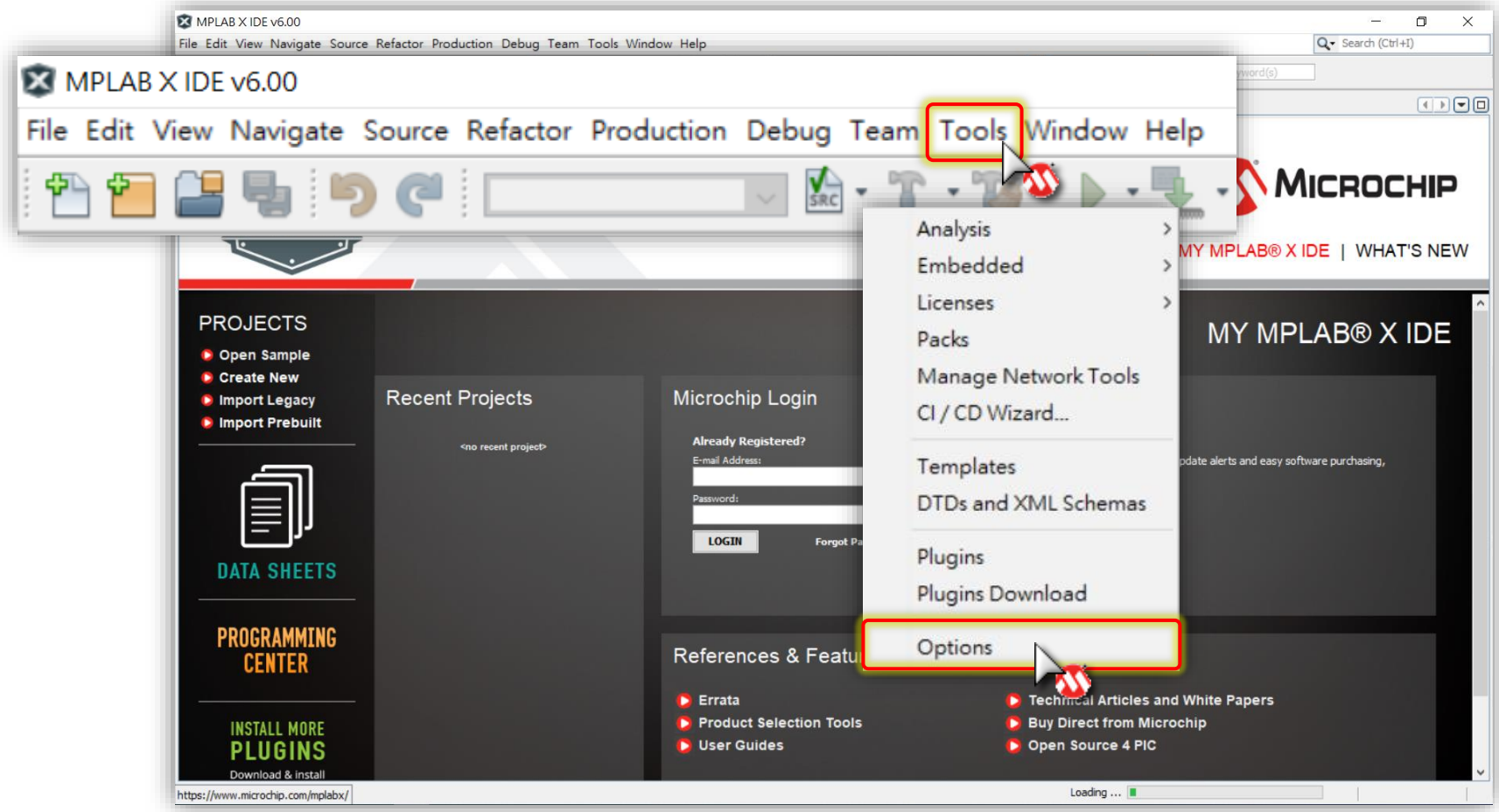


MICROCHIP

Lab Preparation

Step 2

- **Select : Tools ▸ Options**

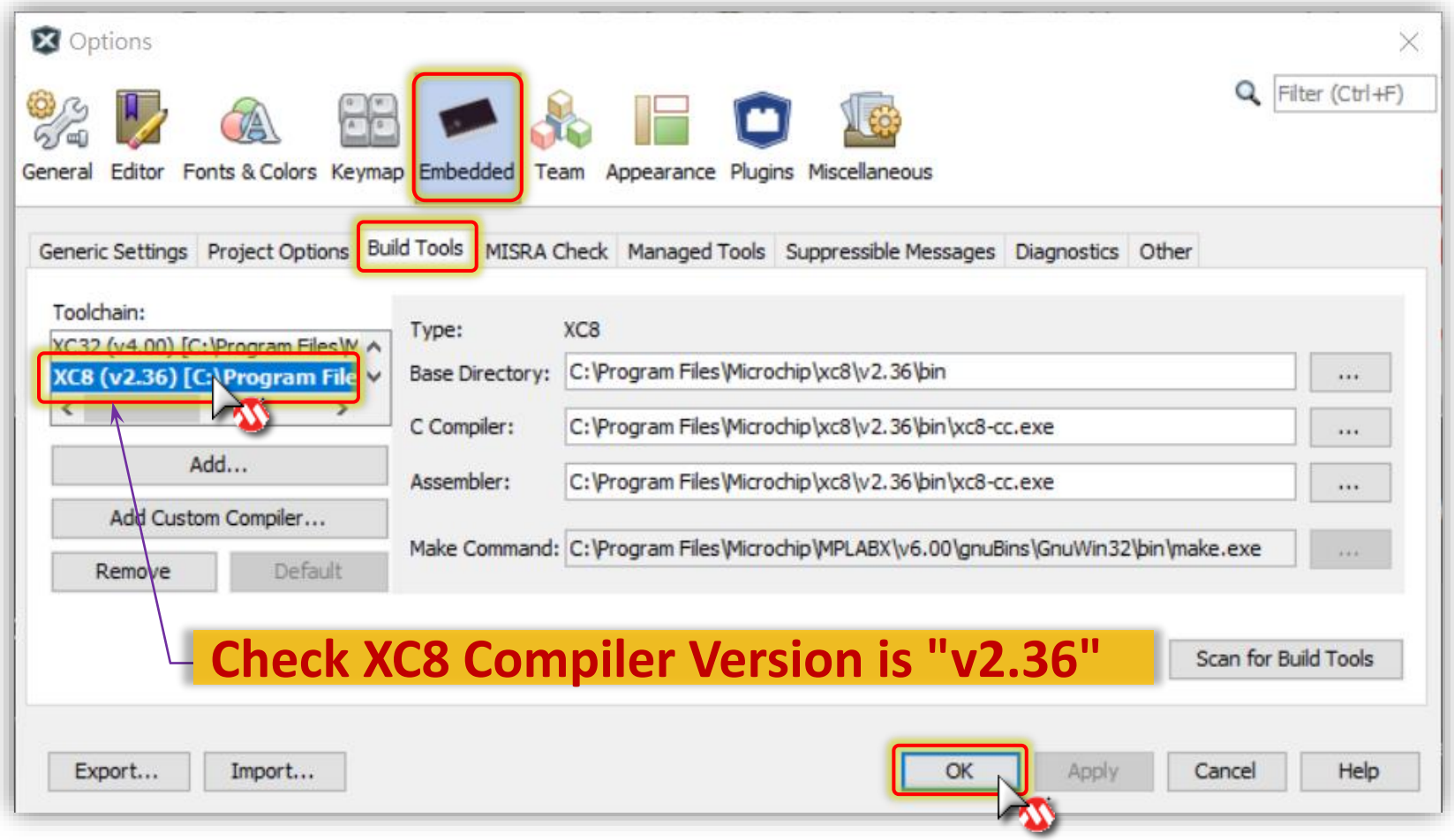


MICROCHIP

Lab Preparation

Step 3

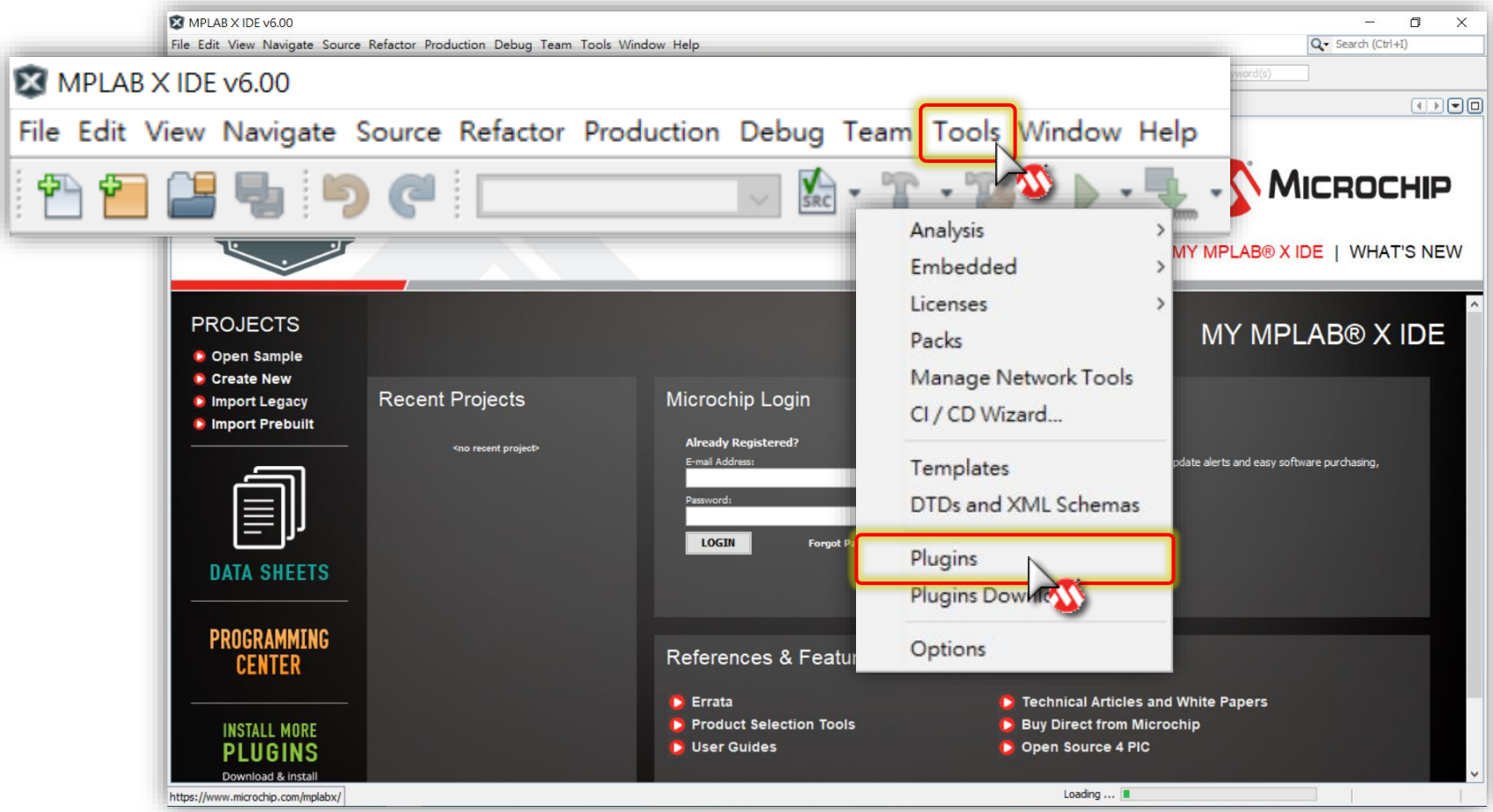
- **Select : Embedded ▸ Build Tools**



Lab Preparation

Step 4

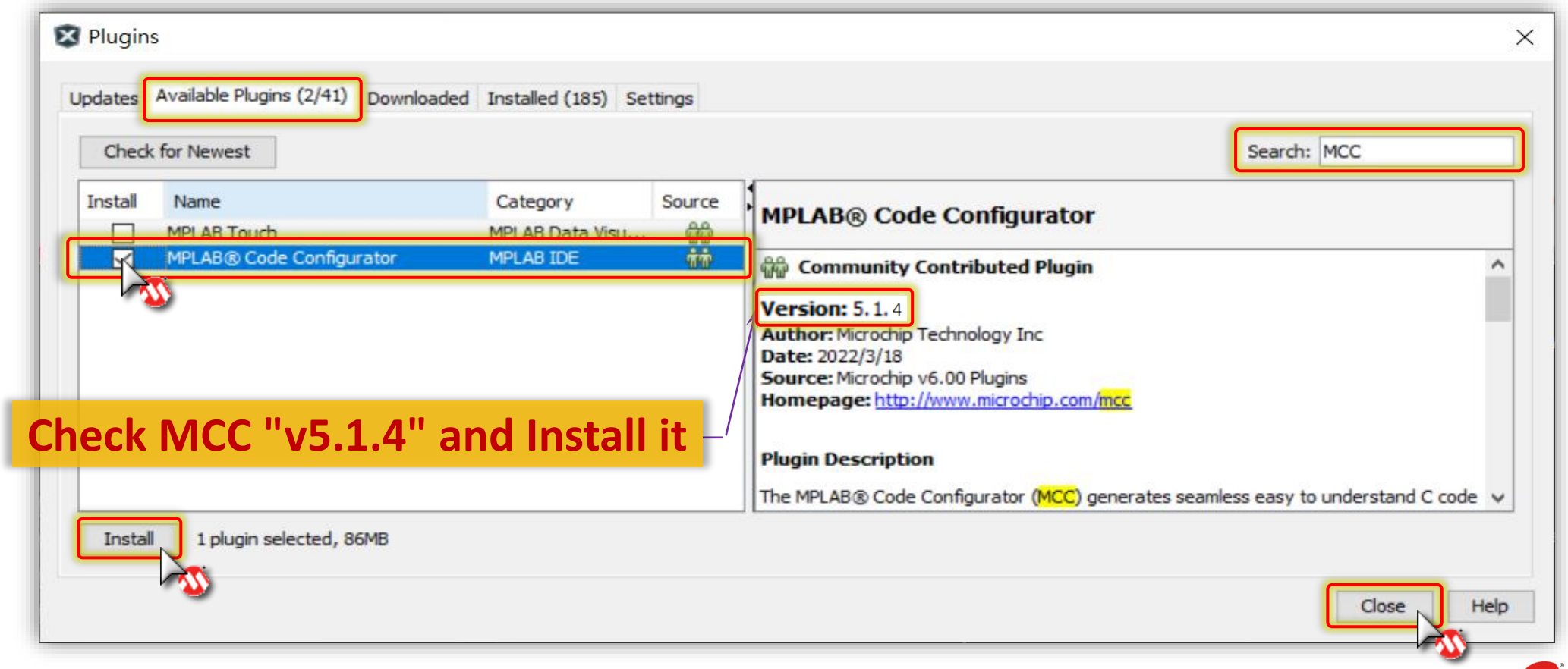
- **Select : Tools ▸ Plugins**



Lab Preparation

Step 5 (MCC not installed yet)

- Select : **Available Plugins**
- Search : **MCC**

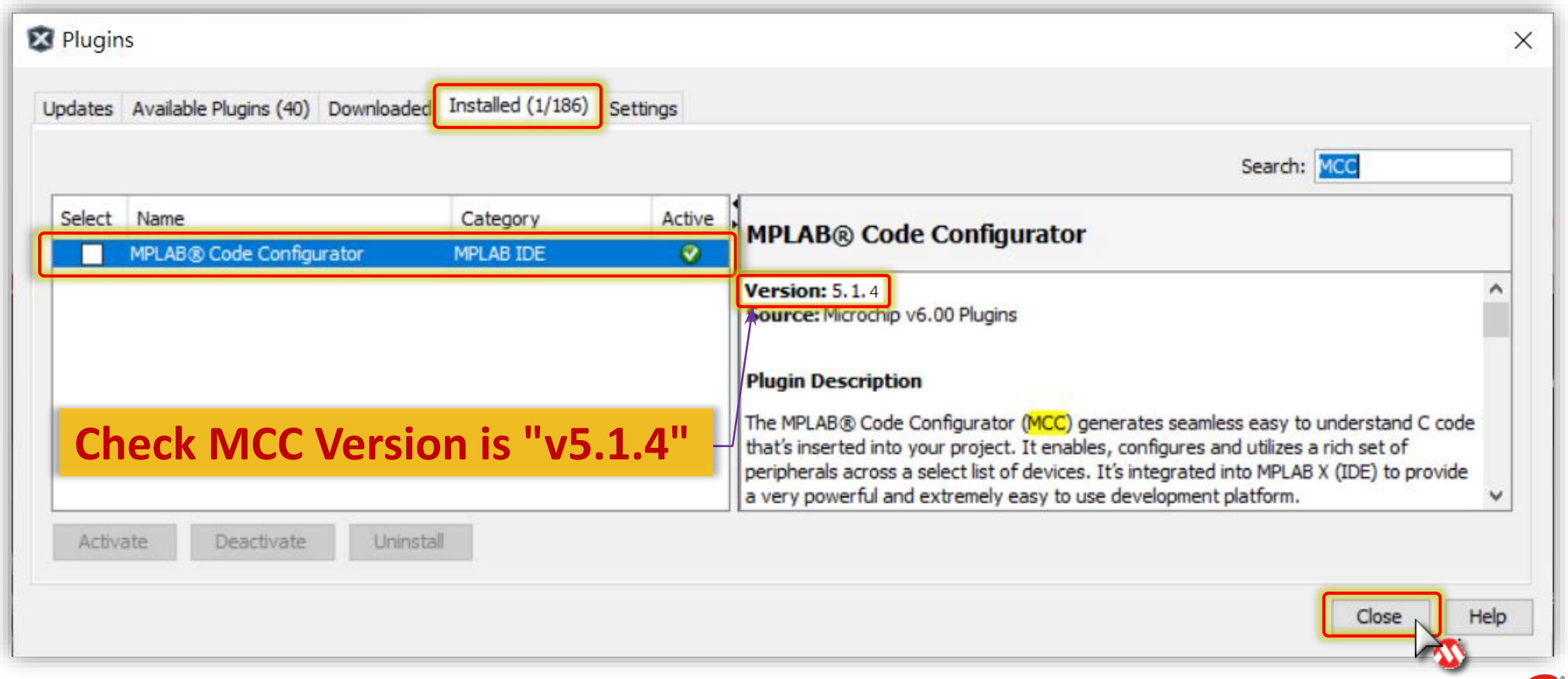


MICROCHIP

Lab Preparation

Step 5 (MCC has installed)

- Select : Installed
- Search : MCC

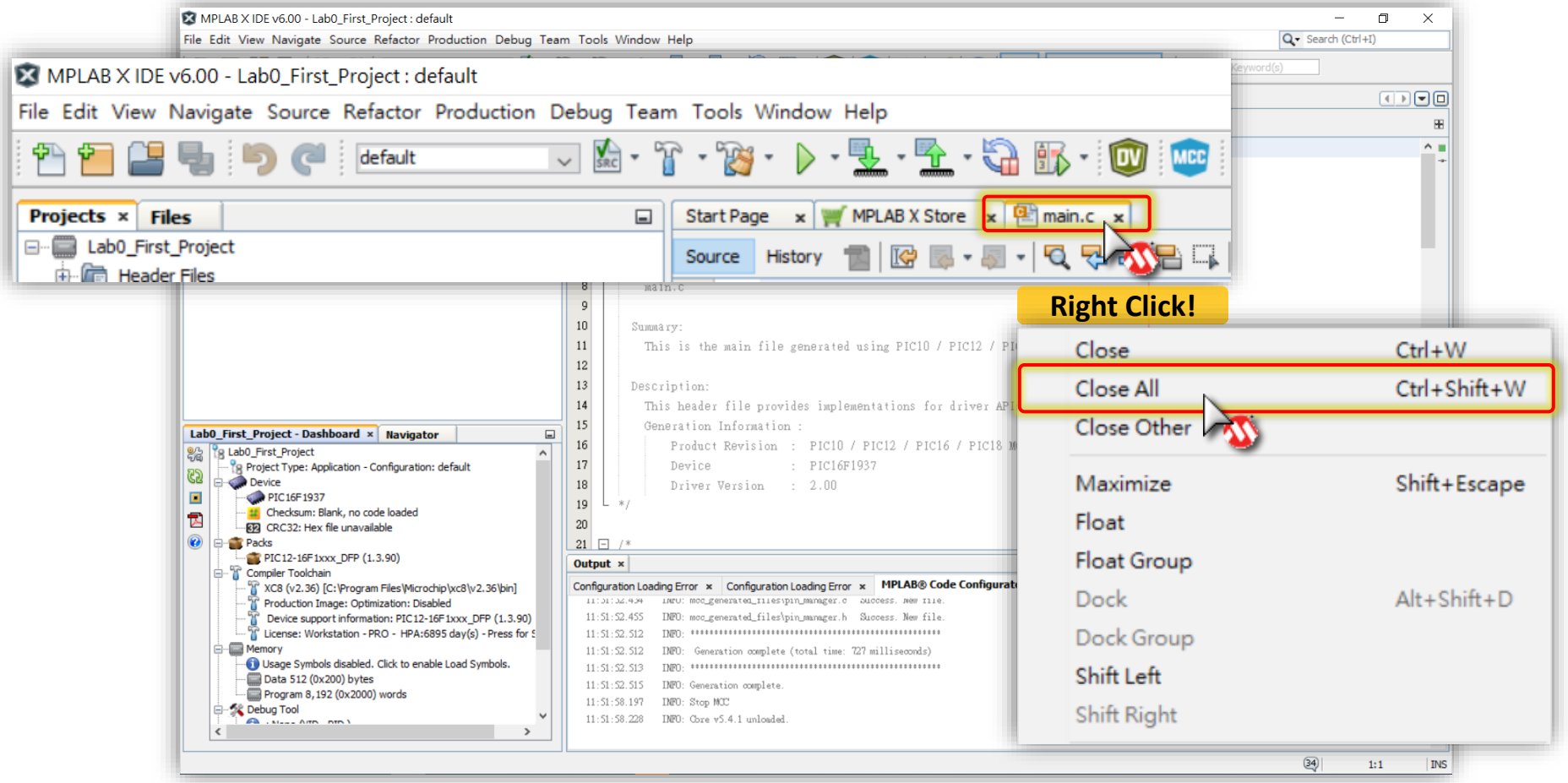


MICROCHIP

Lab Preparation

Step 6

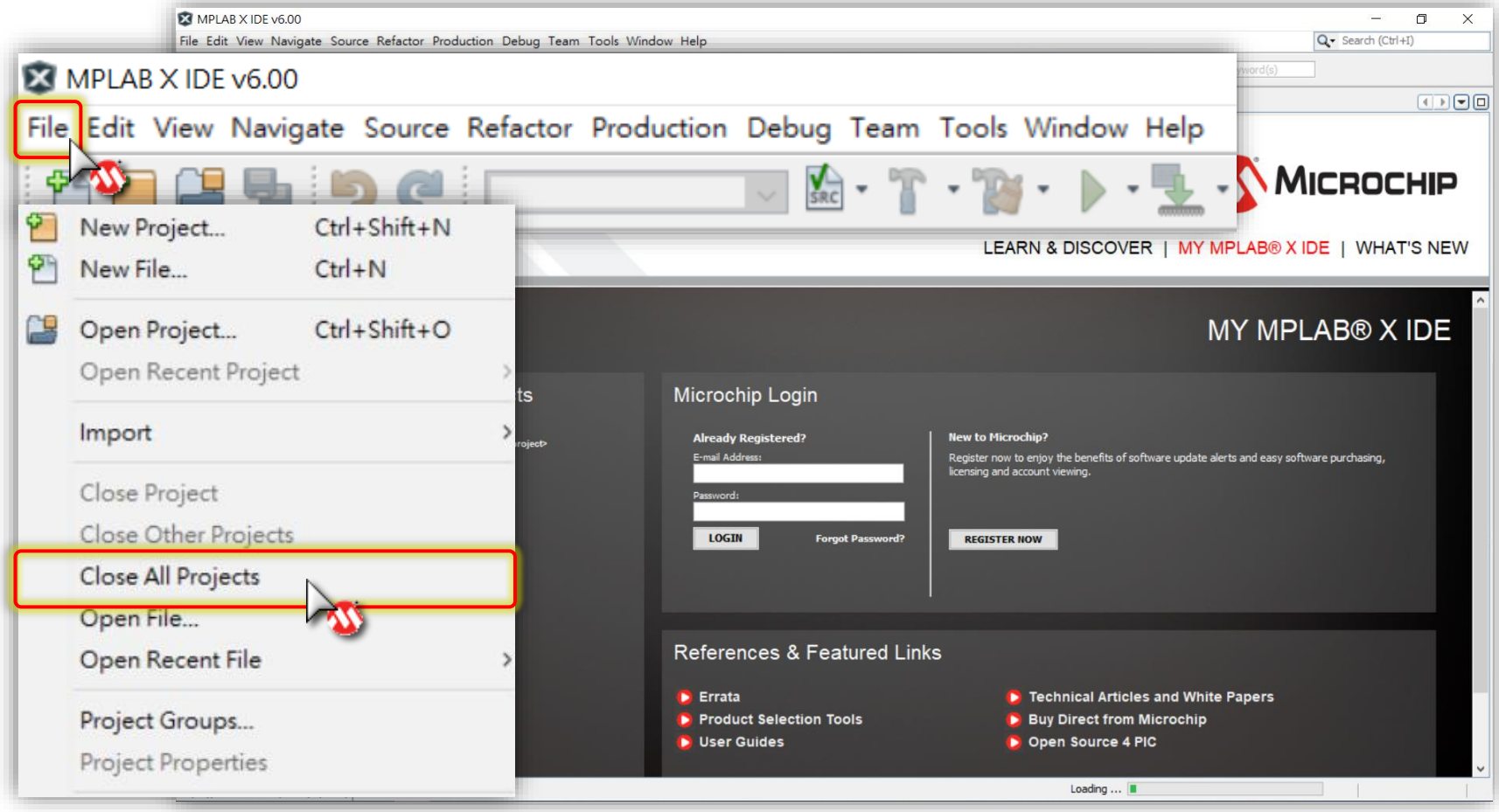
- To prevent opened files to confuse our implementation.
- Please [Right Click] on opened file and choose Close All.



Lab Preparation

Step 7

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



MICROCHIP

Lab0 First Project

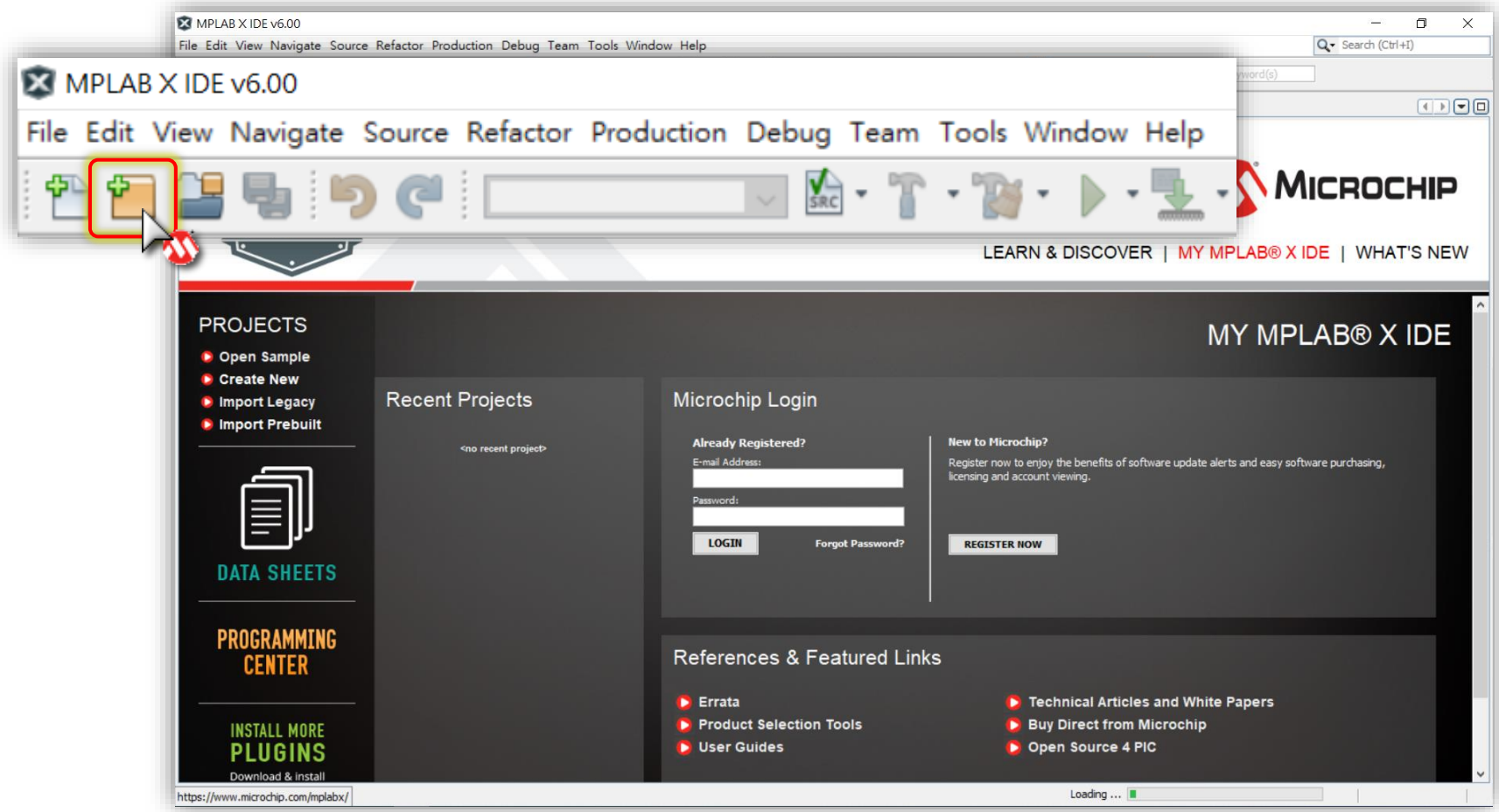
- Try to create your first MPLAB X IDE Project.
- Try to create your first MCC style Project.
- To understanding MPLAB X IDE basic operation.
- Learn how to use edit, build, project manager and program functions.

How to start ?

Lab0 First Project

Step 1

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

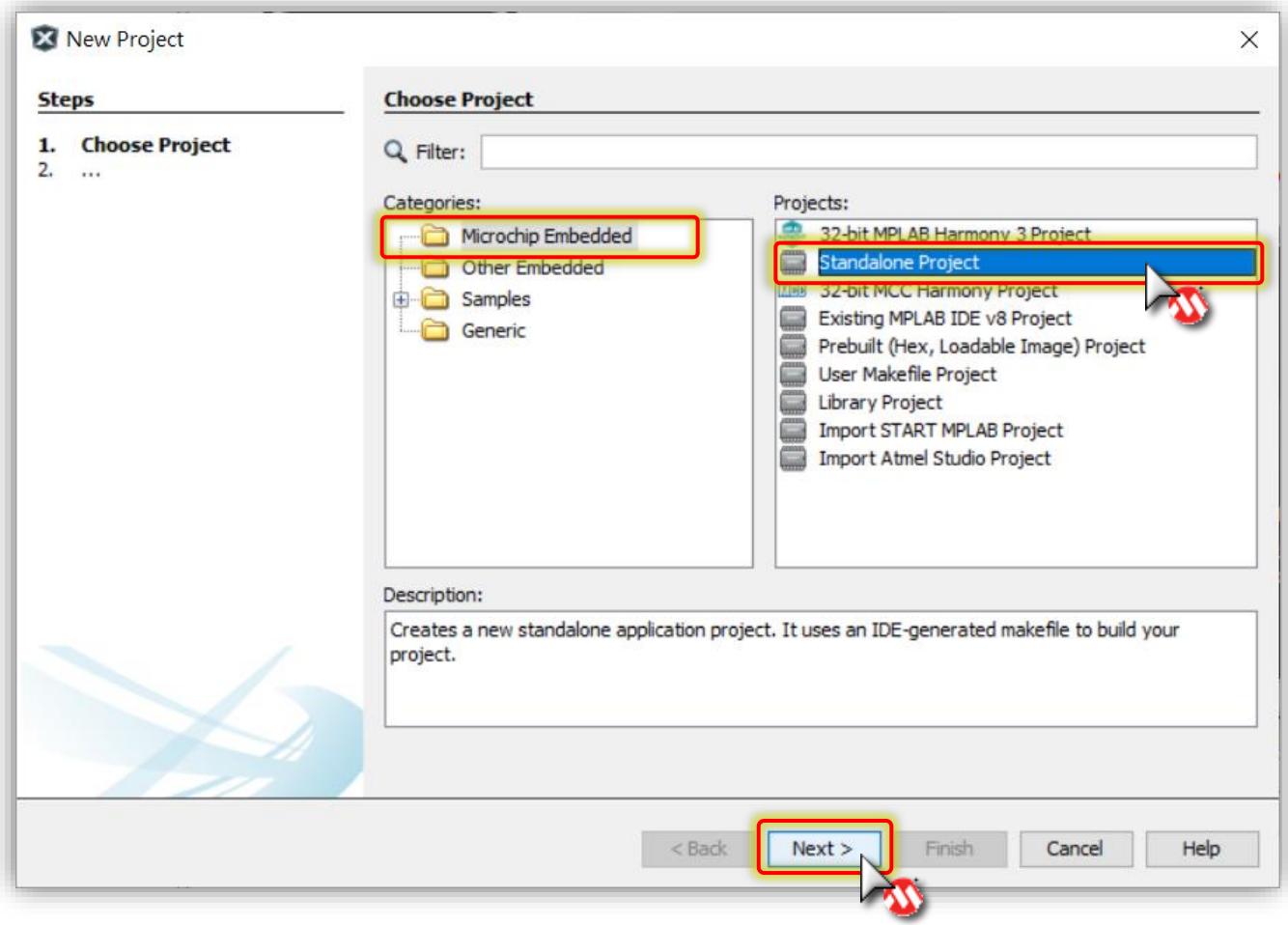


MICROCHIP

Lab0 First Project

Step 2

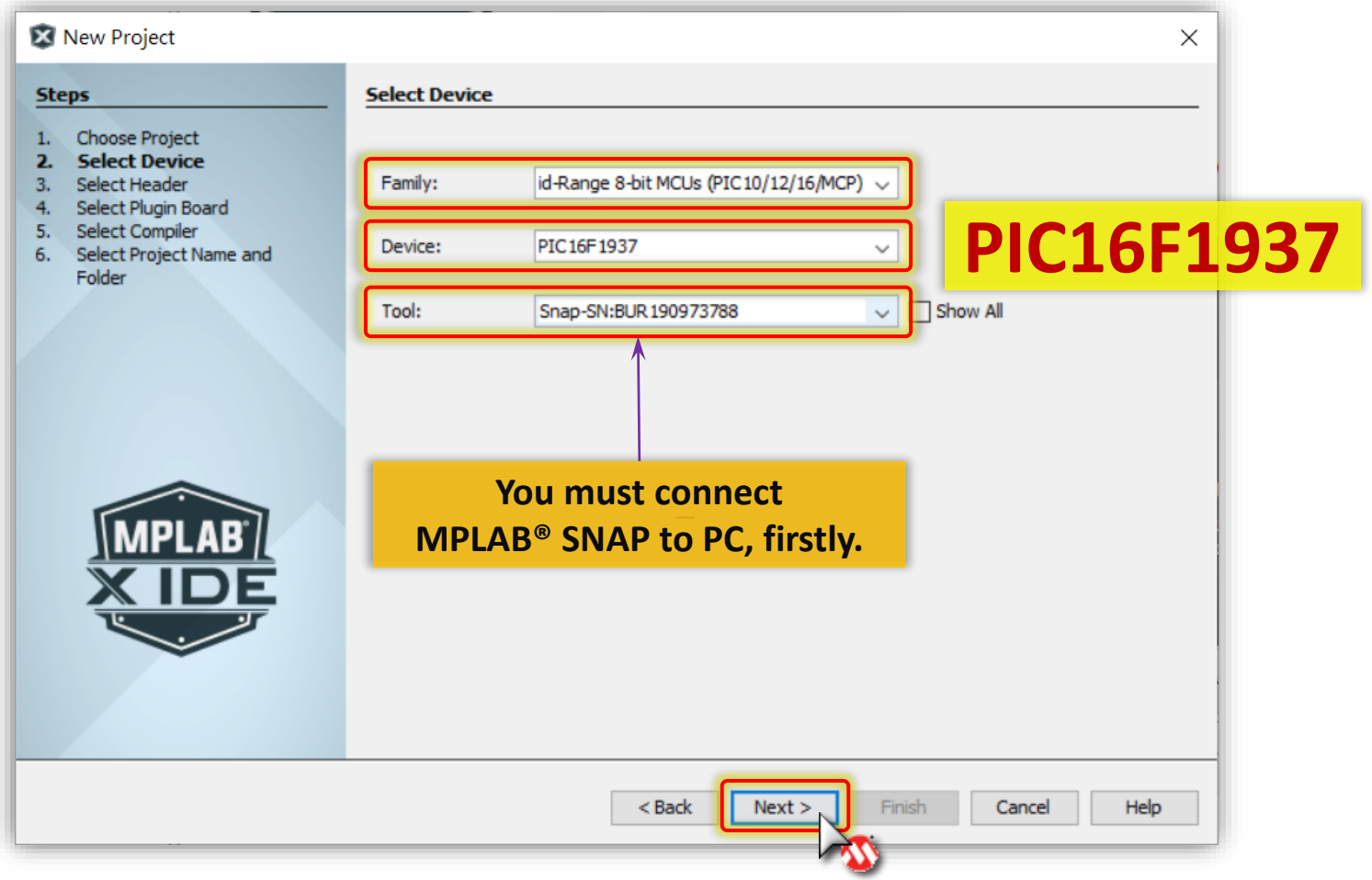
- Categories : Microchip Embedded
- Projects : Standalone Project



Lab0 First Project

Step 3

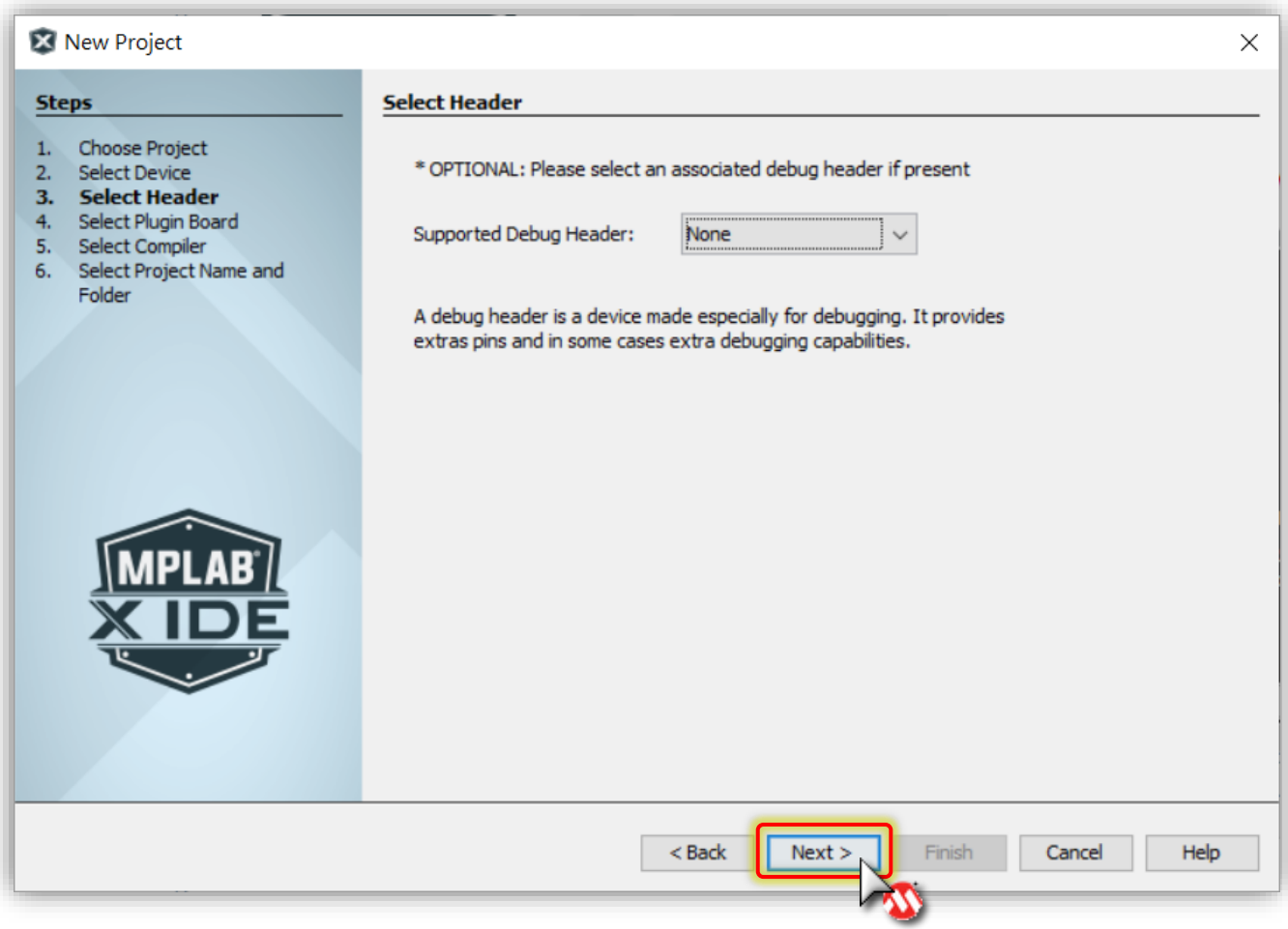
- Family : Mid-Range 8-bit MCUs (PIC10/12/16/MCP) , Device : PIC16F1937
- Tool : Snap-SN : BURXXXXXXXXX



Lab0 First Project

Step 4

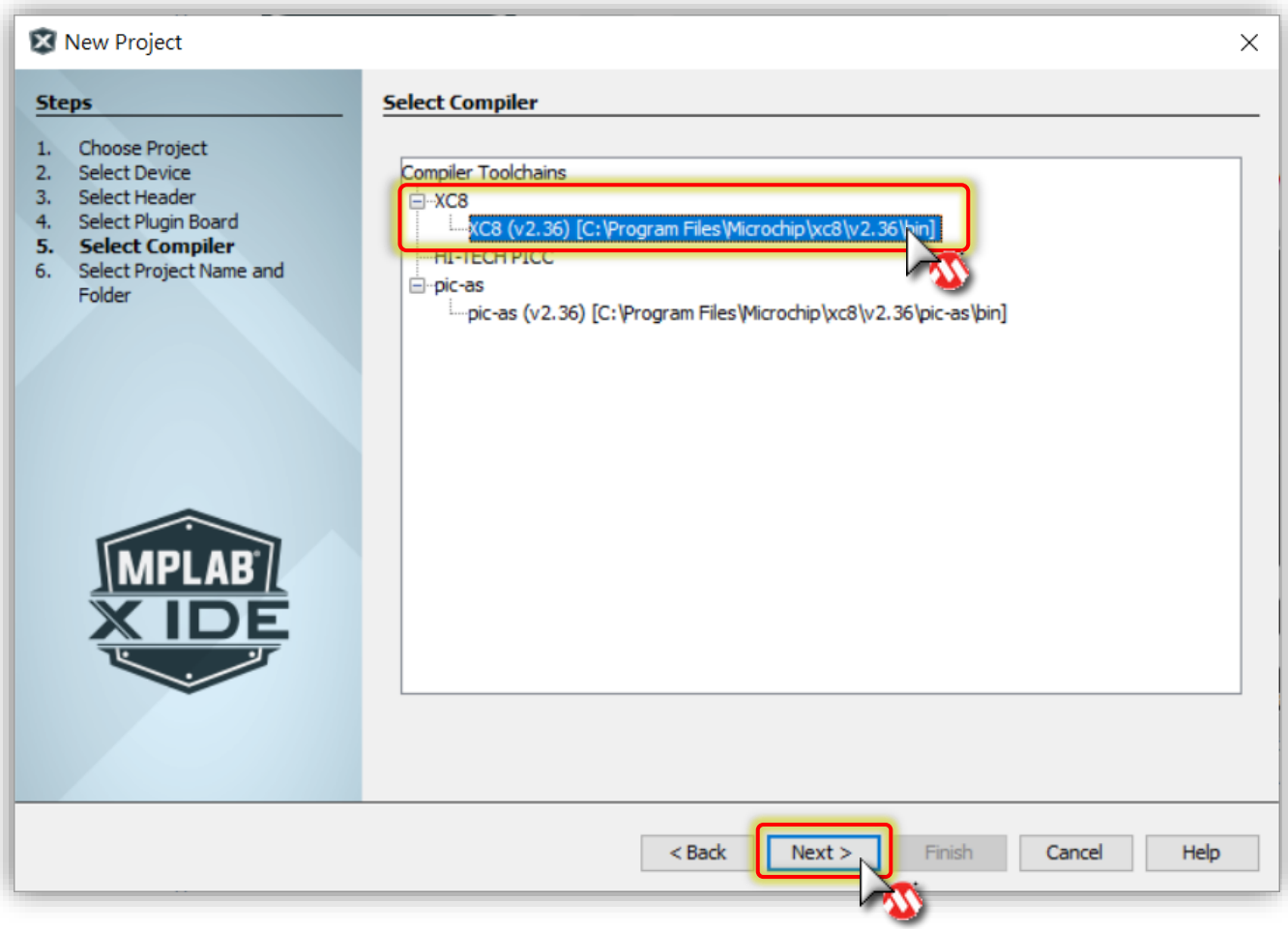
- **Supported Debug Header : None**



Lab0 First Project

Step 5 (Use Bare metal C and MCC)

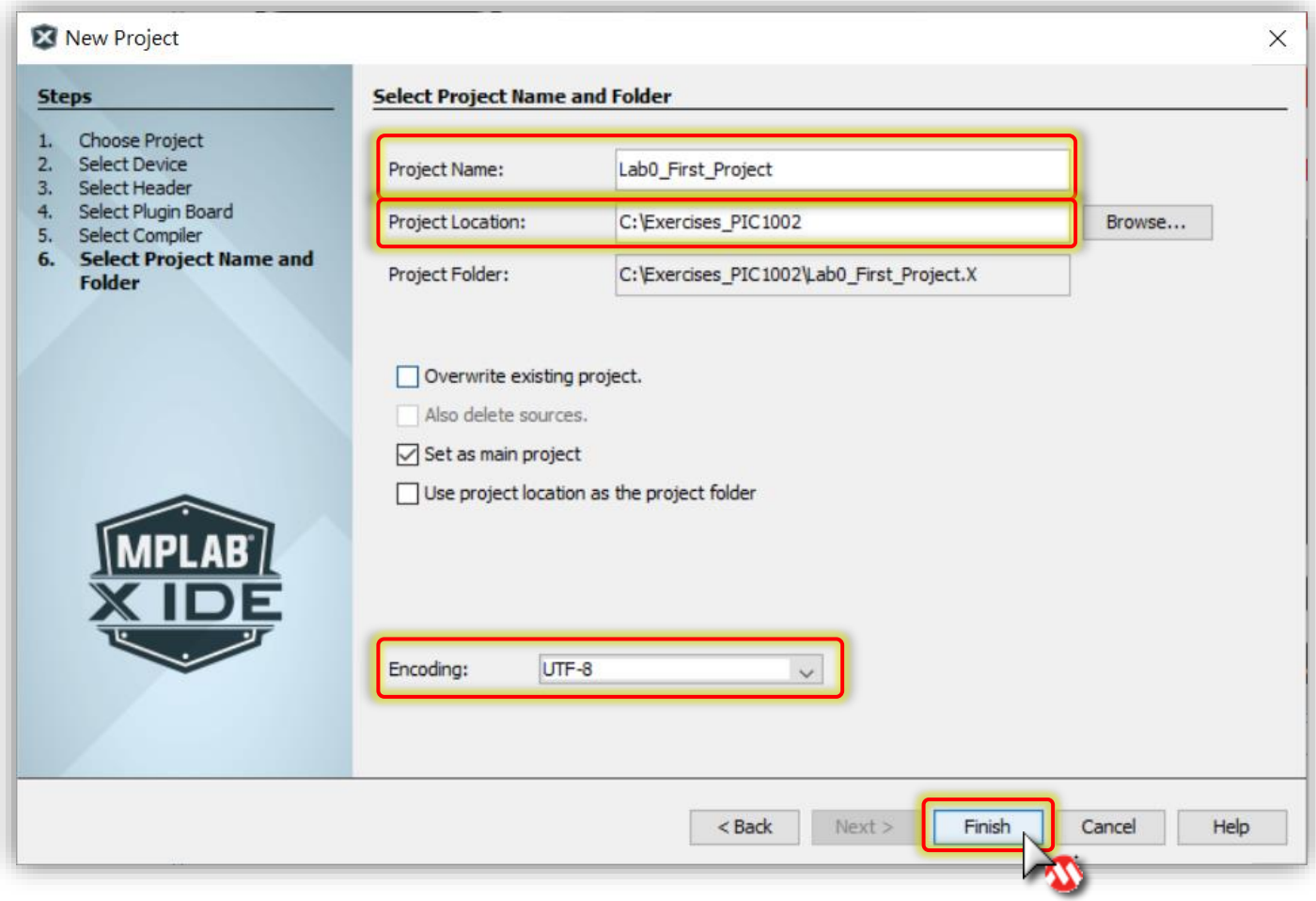
- **Compiler Toolchains : XC8 (v2.36) [C:\Program Files\Microchip\xc8\v2.36\bin]**



Lab0 First Project

Step 6

- Project Name : Lab0_First_Project , Project Location : C:\Exercises_PIC1002
- Encoding : UTF-8

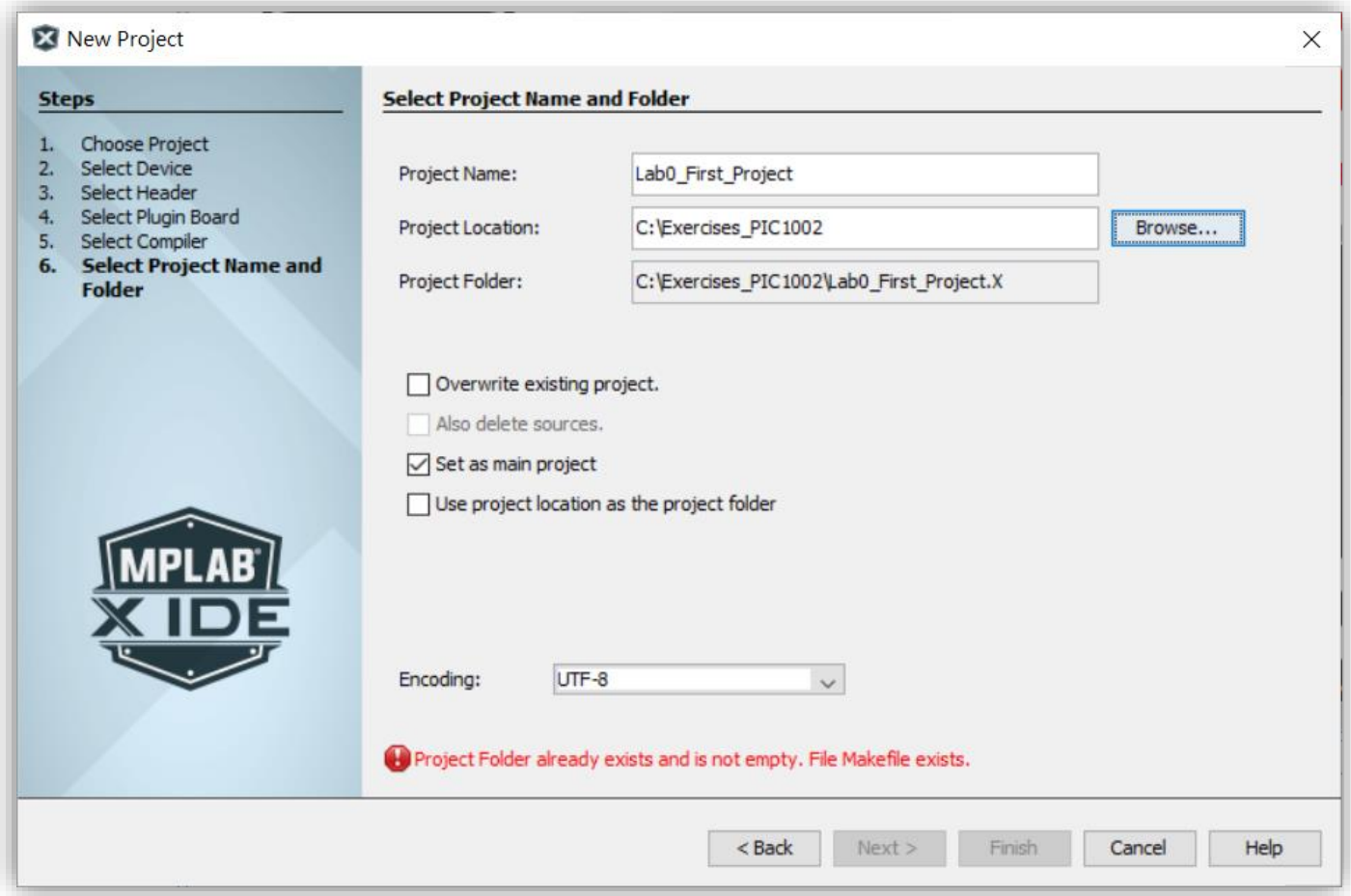


MICROCHIP

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.**

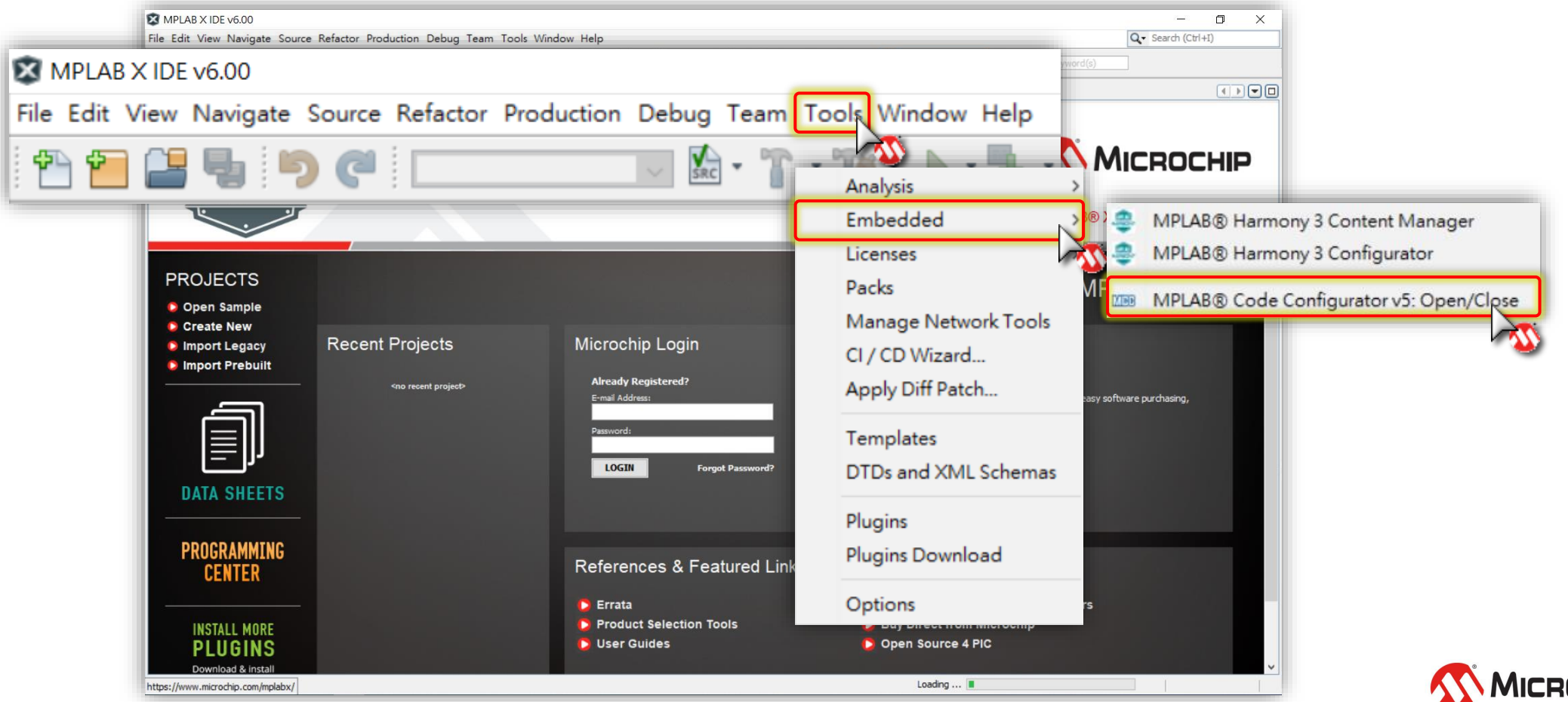


MICROCHIP

Lab0 First Project

Step 7

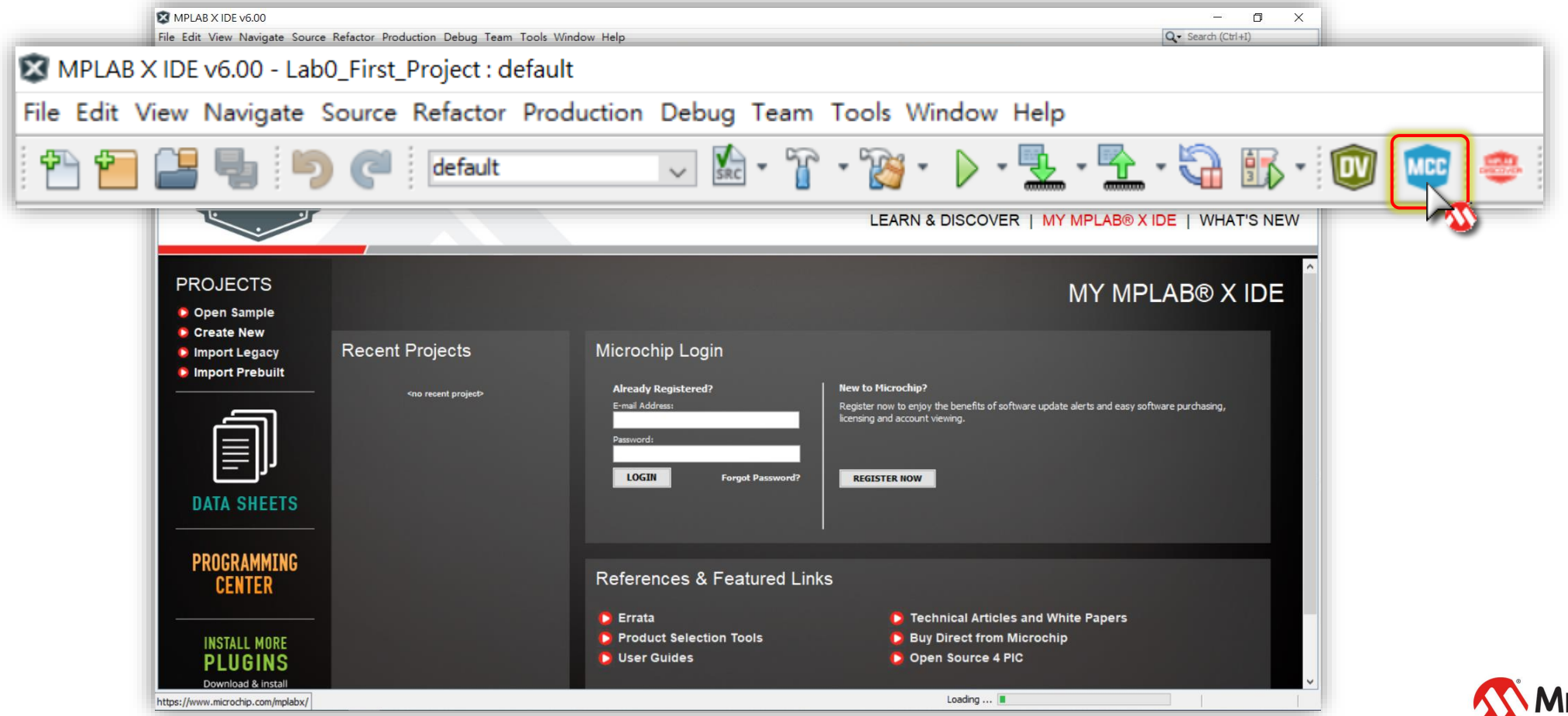
- Menu Tools ▸ Embedded ▸ MPLAB Code Configurator v5: Open/Close or Click



Lab0 First Project

Step 7

- Menu Tools ▸ Embedded ▸ MPLAB Code Configurator v5: Open/Close or Click 

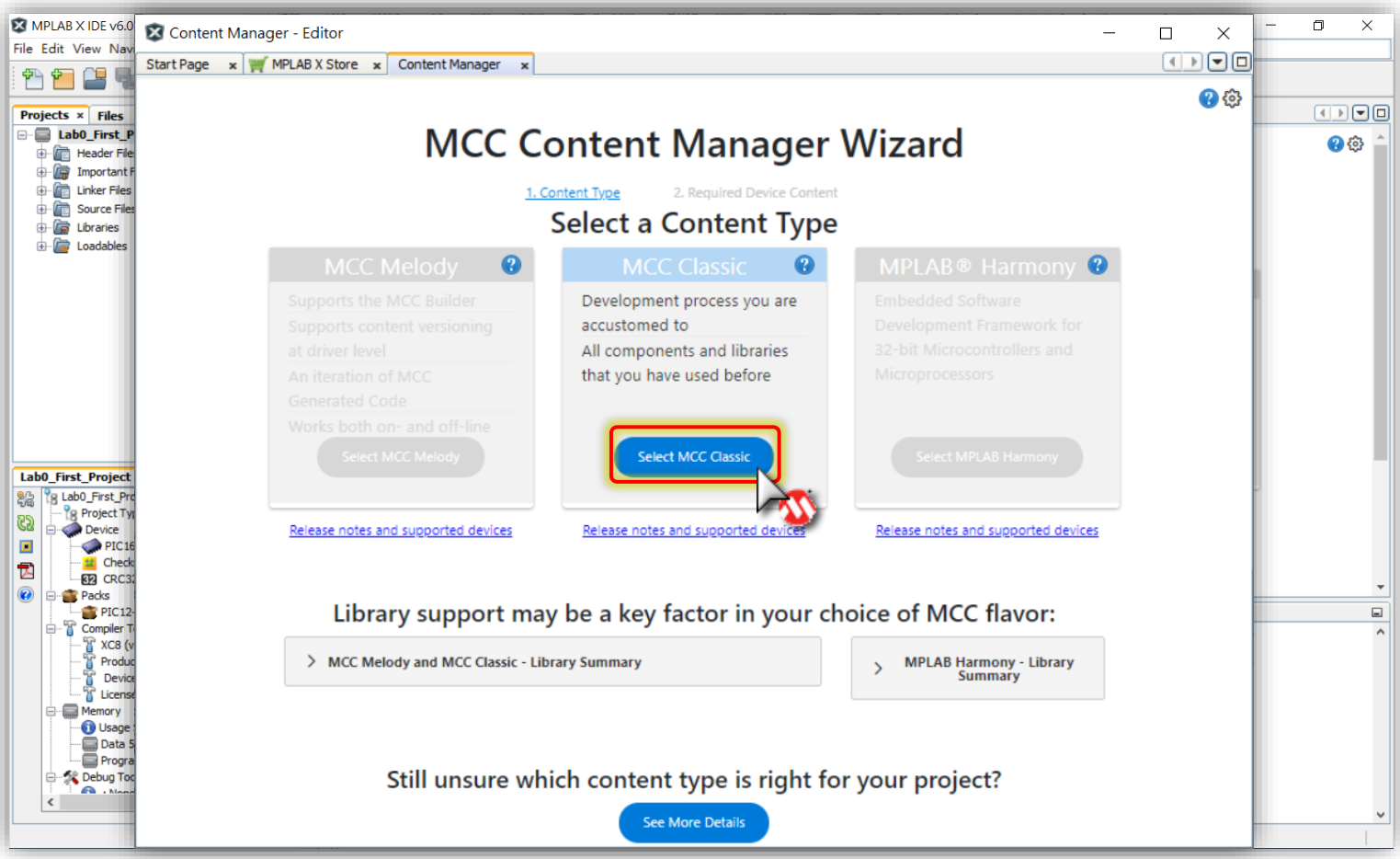


MICROCHIP

Lab0 First Project

Step 8

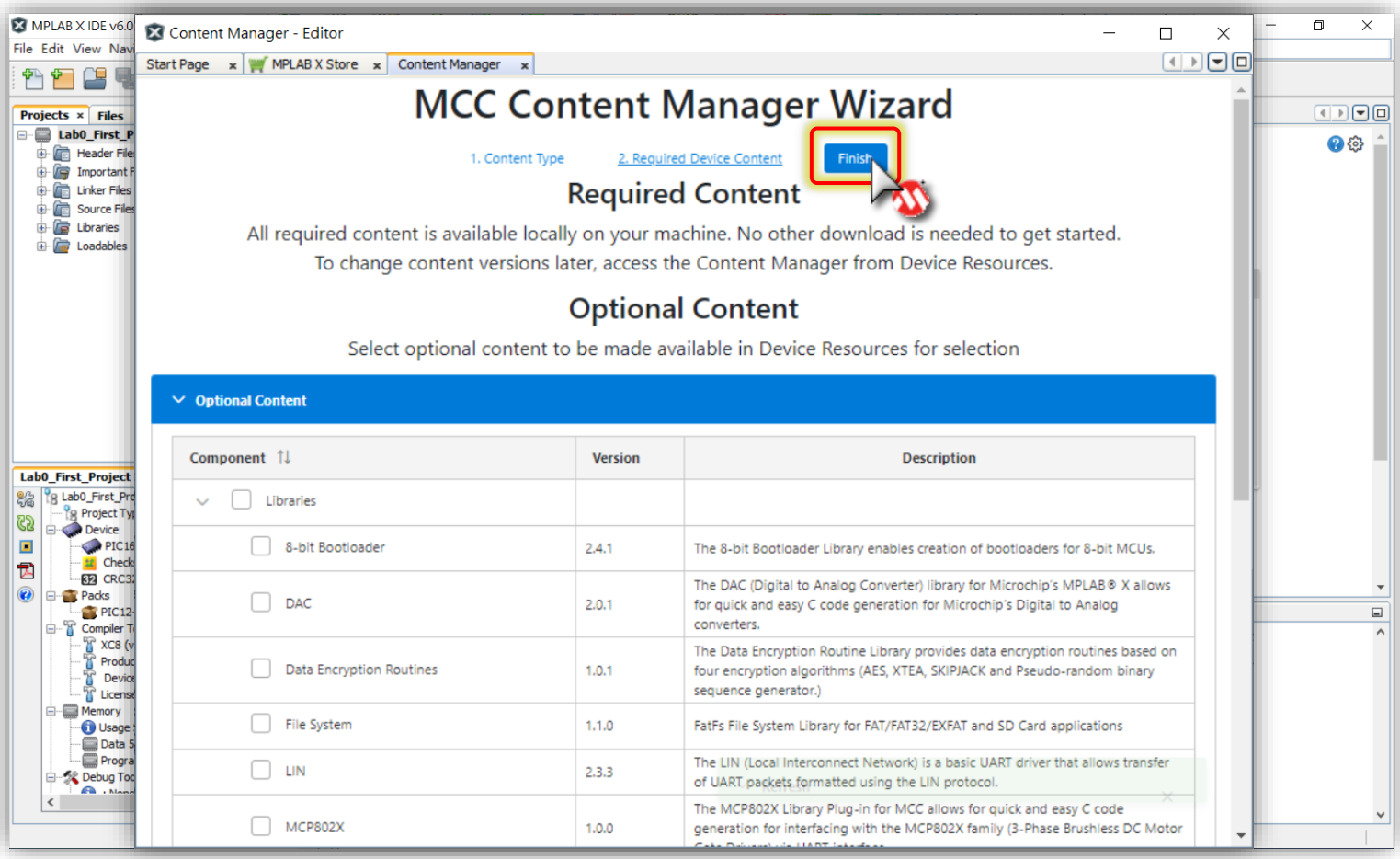
- **Select a Content Type : Select MCC Classic.**



Lab0 First Project

Step 9

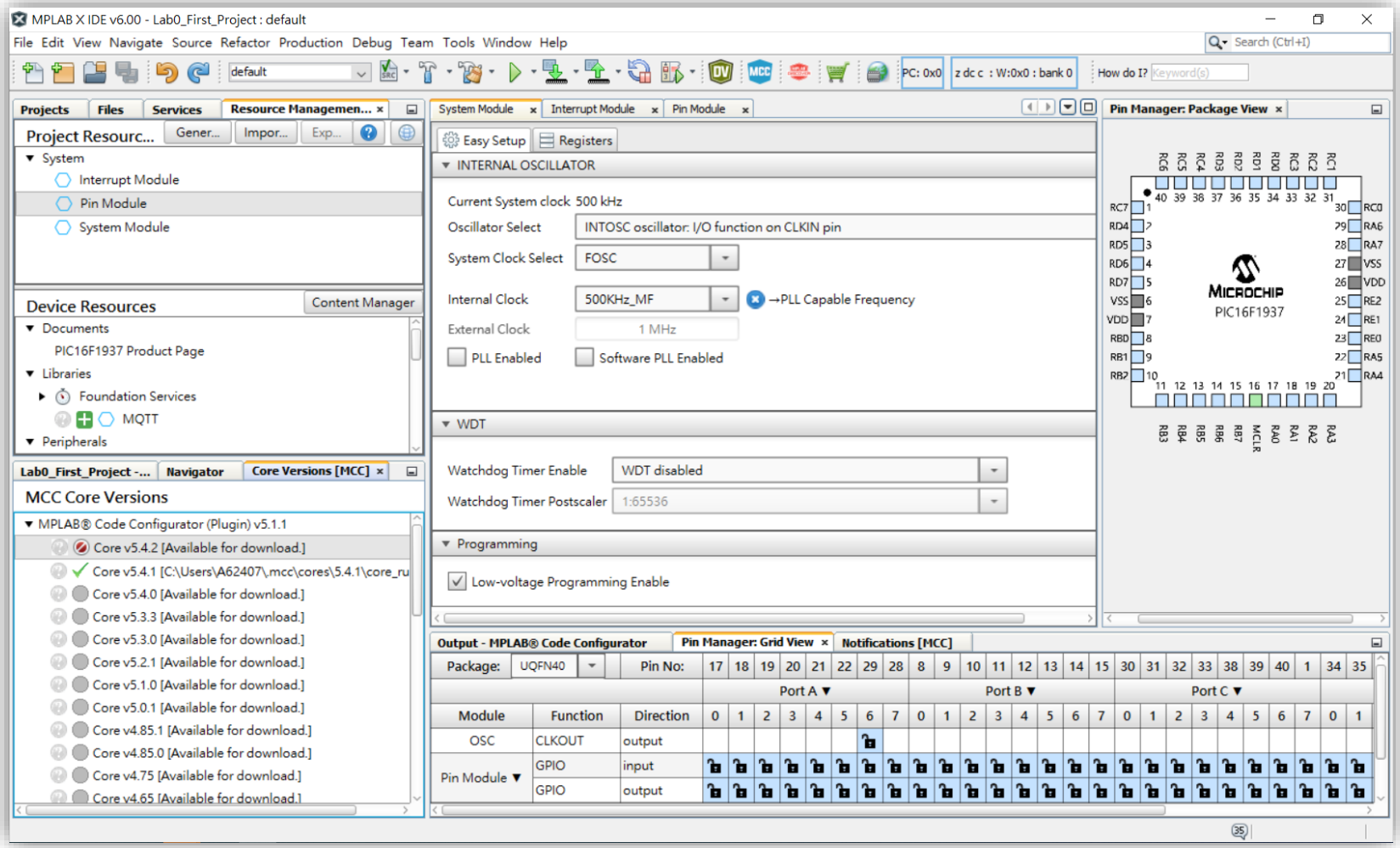
- **You can add the available Libraries provided in the Device resource to the project or click Finish.**



Lab0 First Project

Step 10

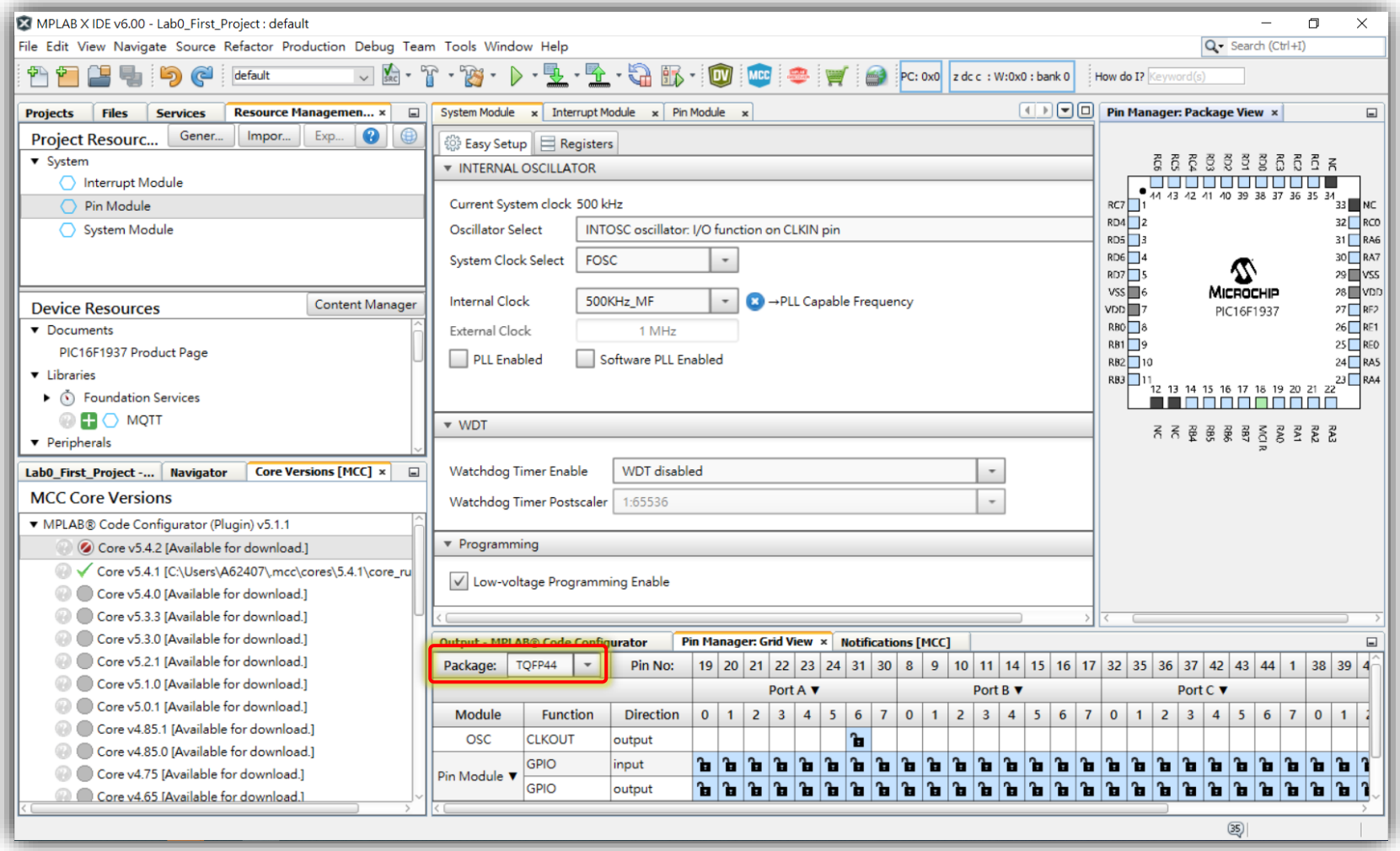
- **MCC Configurator interface show up.**



Lab0 First Project

Step 11

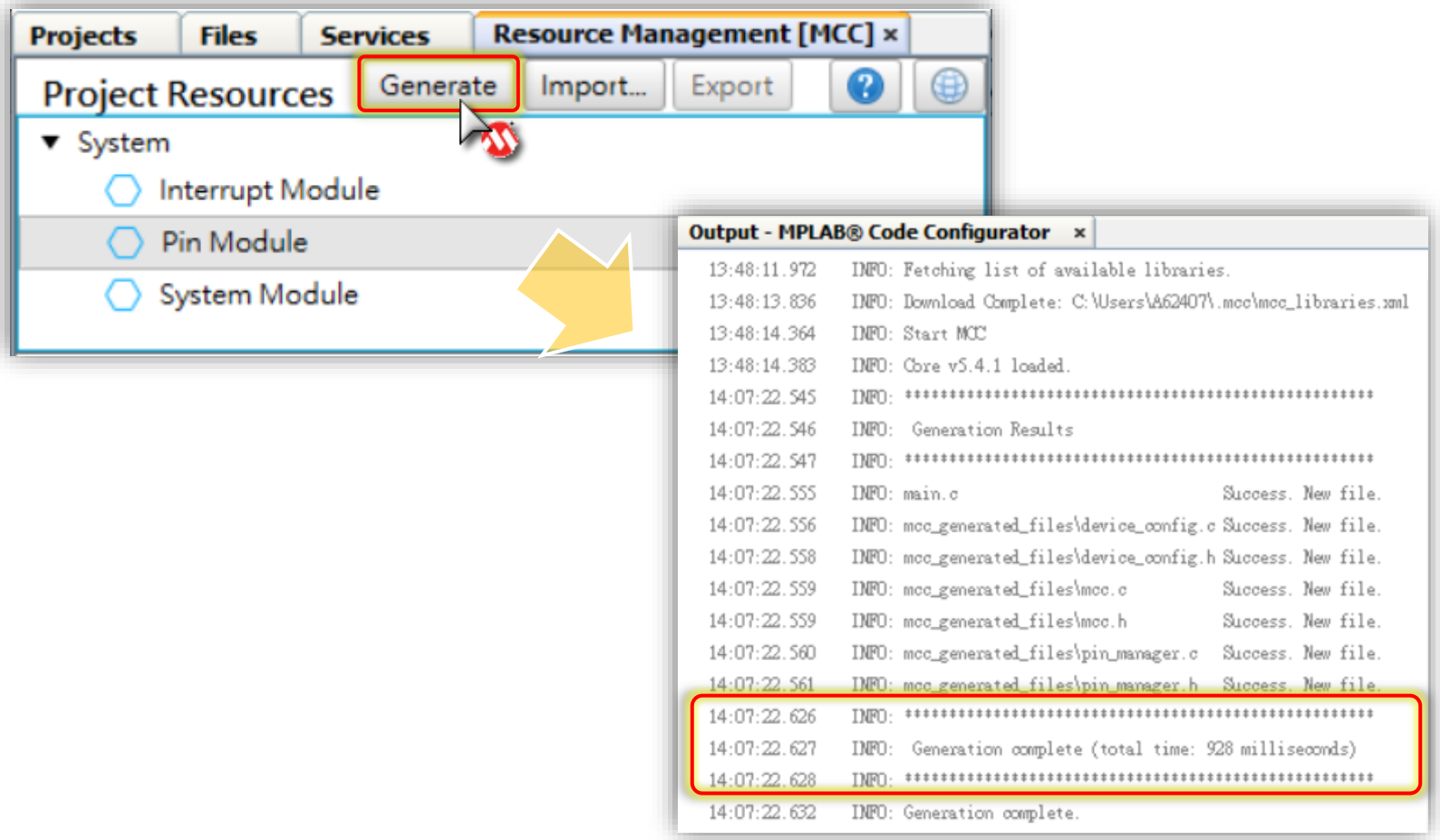
- **Change package : UQFN40 ▸ TQFP44**

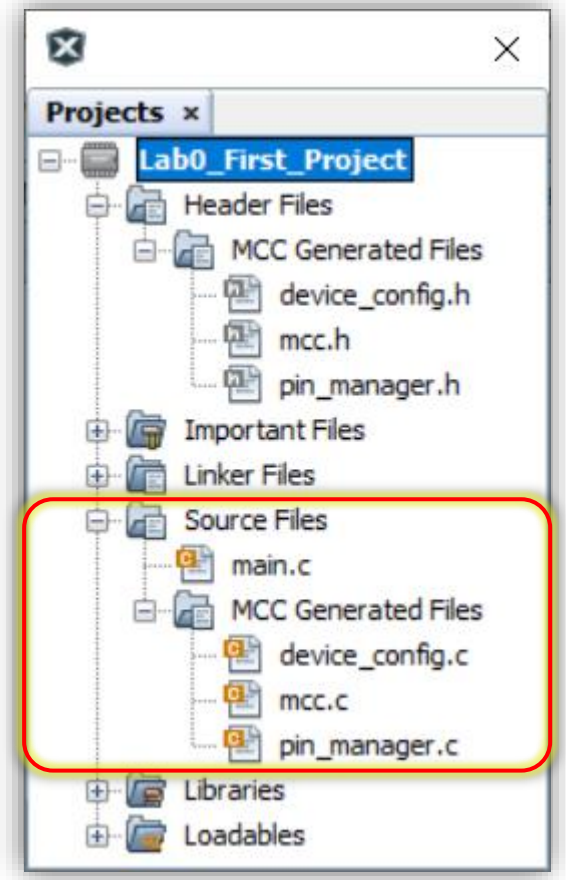


Lab0 First Project

Step 12

- Click [Generate] Button to get your first MCC Project.
- MCC to generate files include main.c automatically.

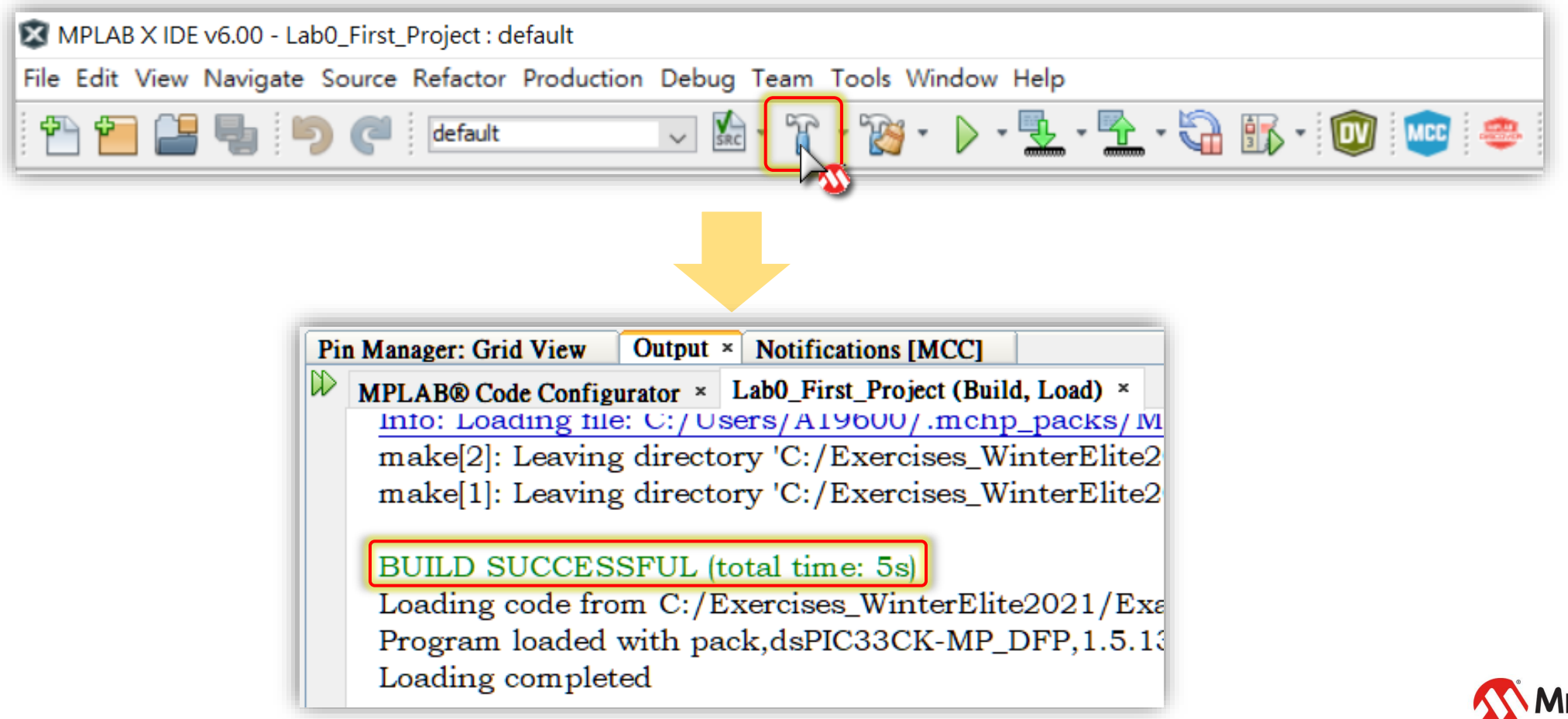




Lab0 First Project

Step 13

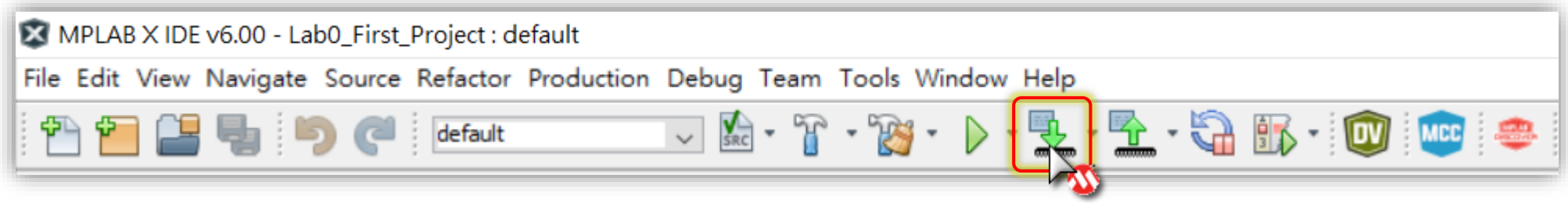
- Select **Build Main Project** icon
- Make sure compiled result is **BUILD SUCCESSFUL.**

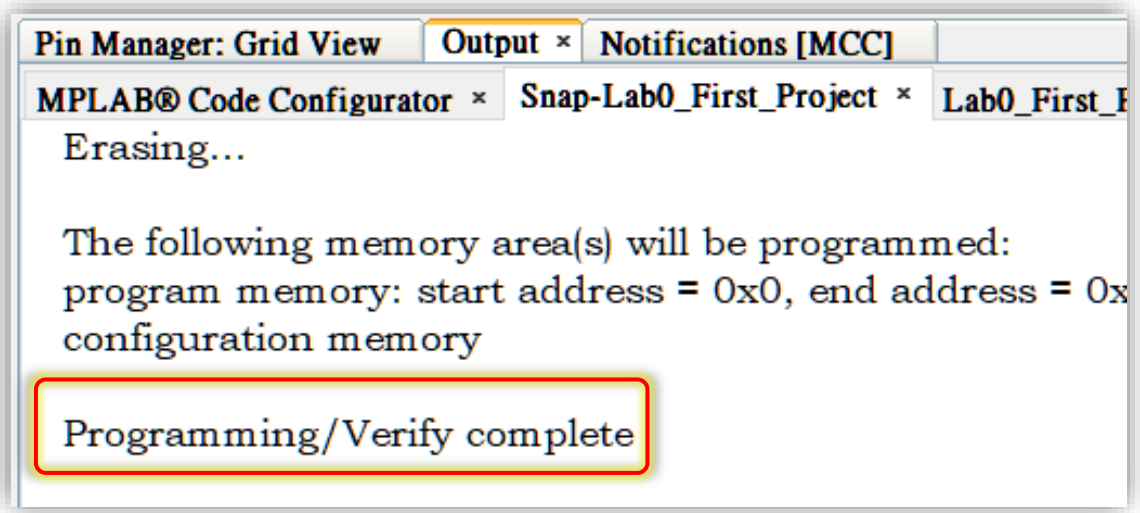


Lab0 First Project

Step 14

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



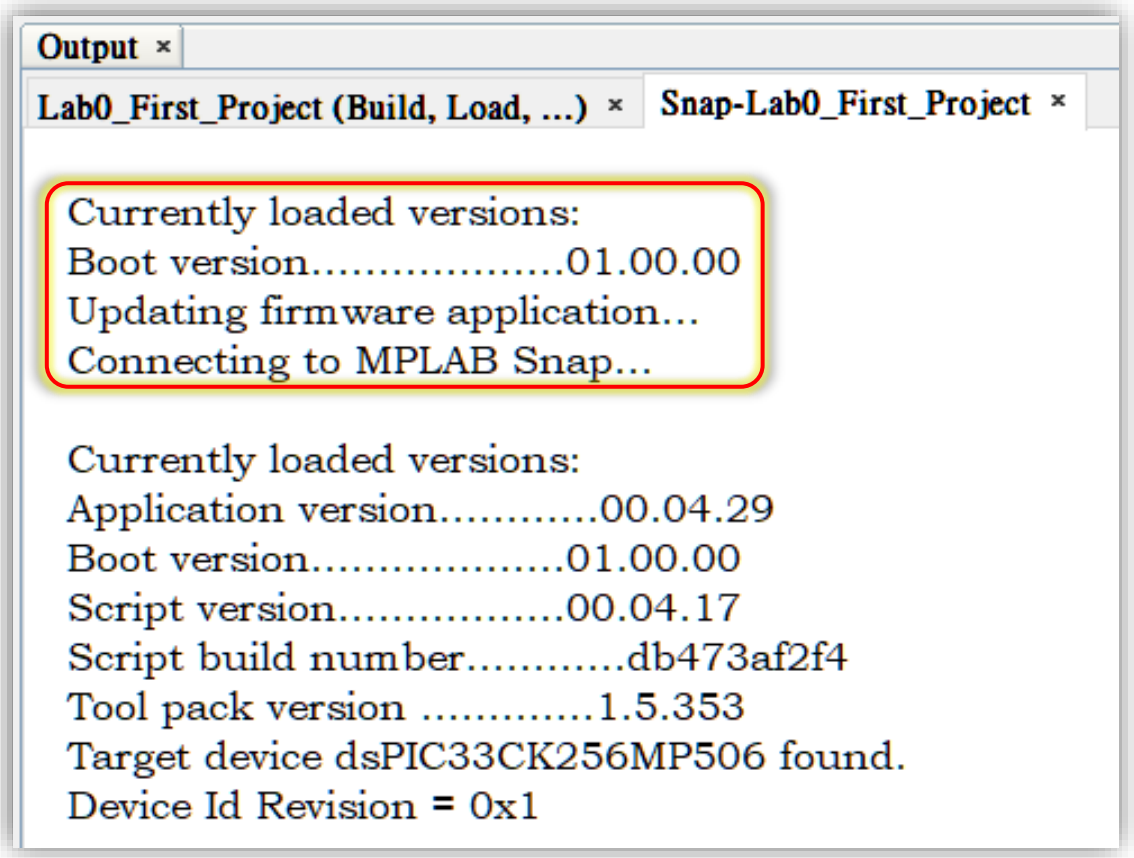


MICROCHIP

Lab0 First Project

Notice

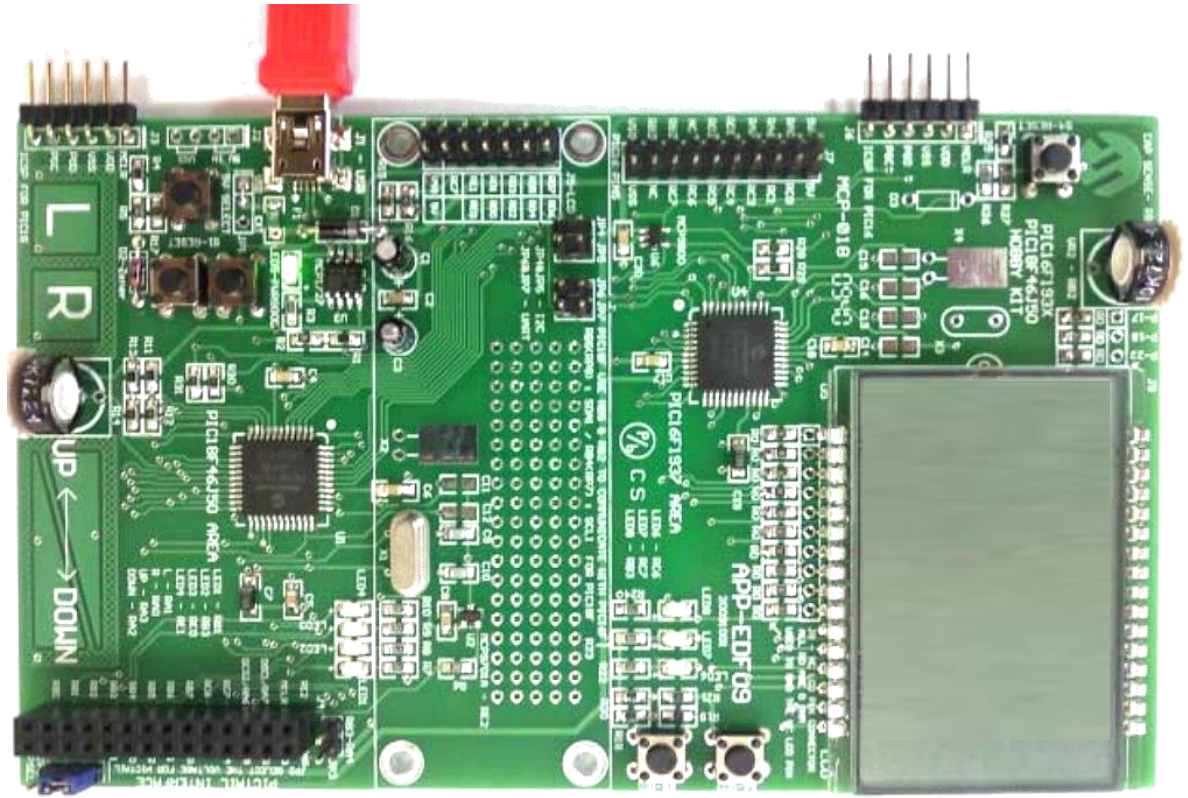
- **MPLAB X IDE will auto upgrade the firmware of SNAP to keep it up to date, it might take couple minutes to wait for finish. Your project will start programming after this.**



Lab0 First Project

Result

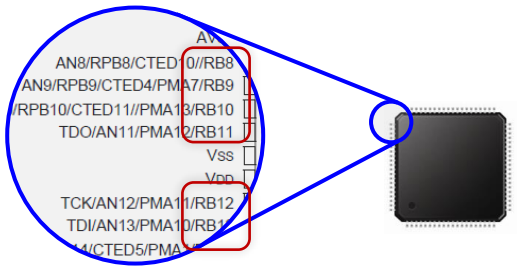
Nothing happen ????

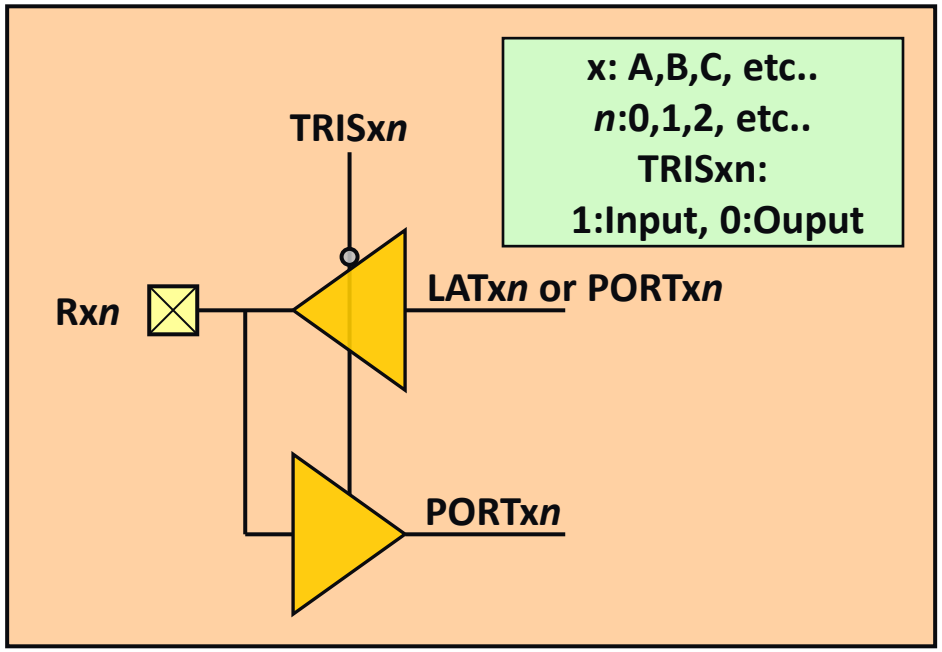


GPIO Architecture

GPIO Block Diagram

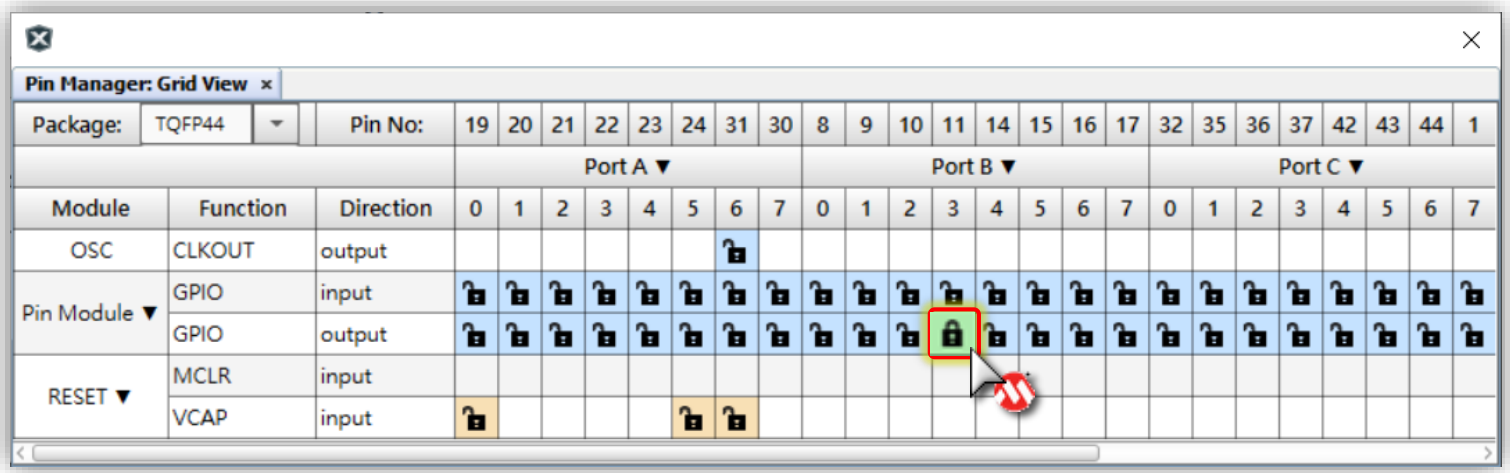
- Please refer to the block diagram, this is the concept for programming model. The real and detail block diagram please refer to the following slide.
- TRIS: To determine Input or Output.
  1 for Input, 0 for Output.
- LAT: set output drive
  value for Output pins.
- PORT: reaction logical
  level from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

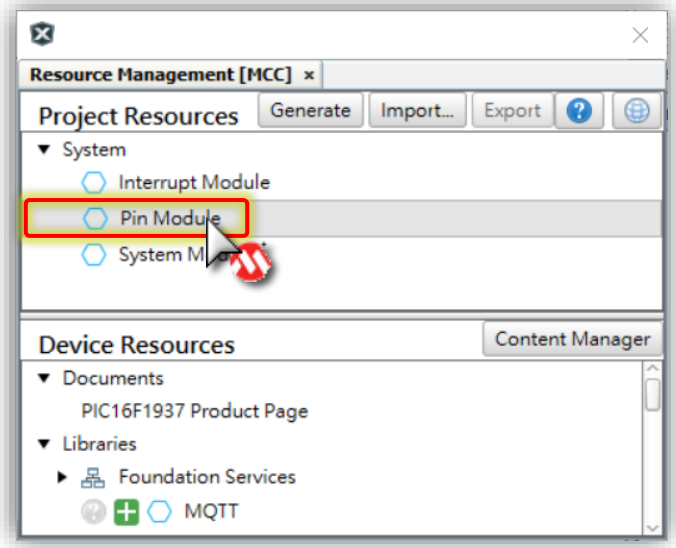


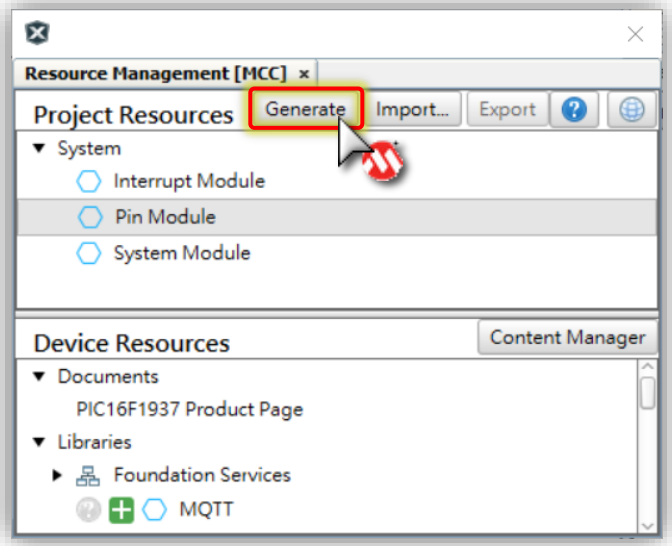


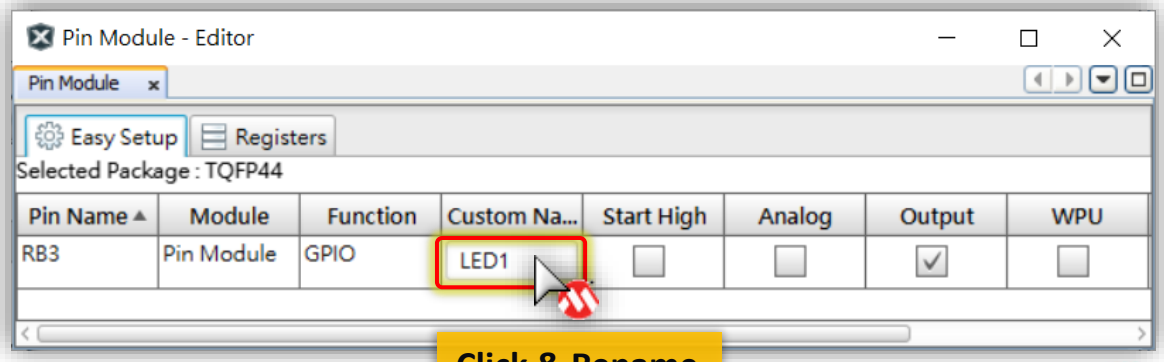
GPIO Setting of MCC

- **Just Click & Select. Code Generate Automatically.**









MICROCHIP

GPIO Functions of MCC

- **MCC Provide below functions for GPIO:**

  Prototype definition : "pin_manager.h"

  - **OUT operation**

    *CustomName*_SetHigh();
    *CustomName*_SetLow();
    *CustomName*_Toggle();
    *CustomName*_SetDigitalOutput();

  - **IN operation**

    *CustomName*_GetValue();
    *CustomName*_SetDigitalInput();

Lab1 GPIO Output

Lab1 GPIO Output

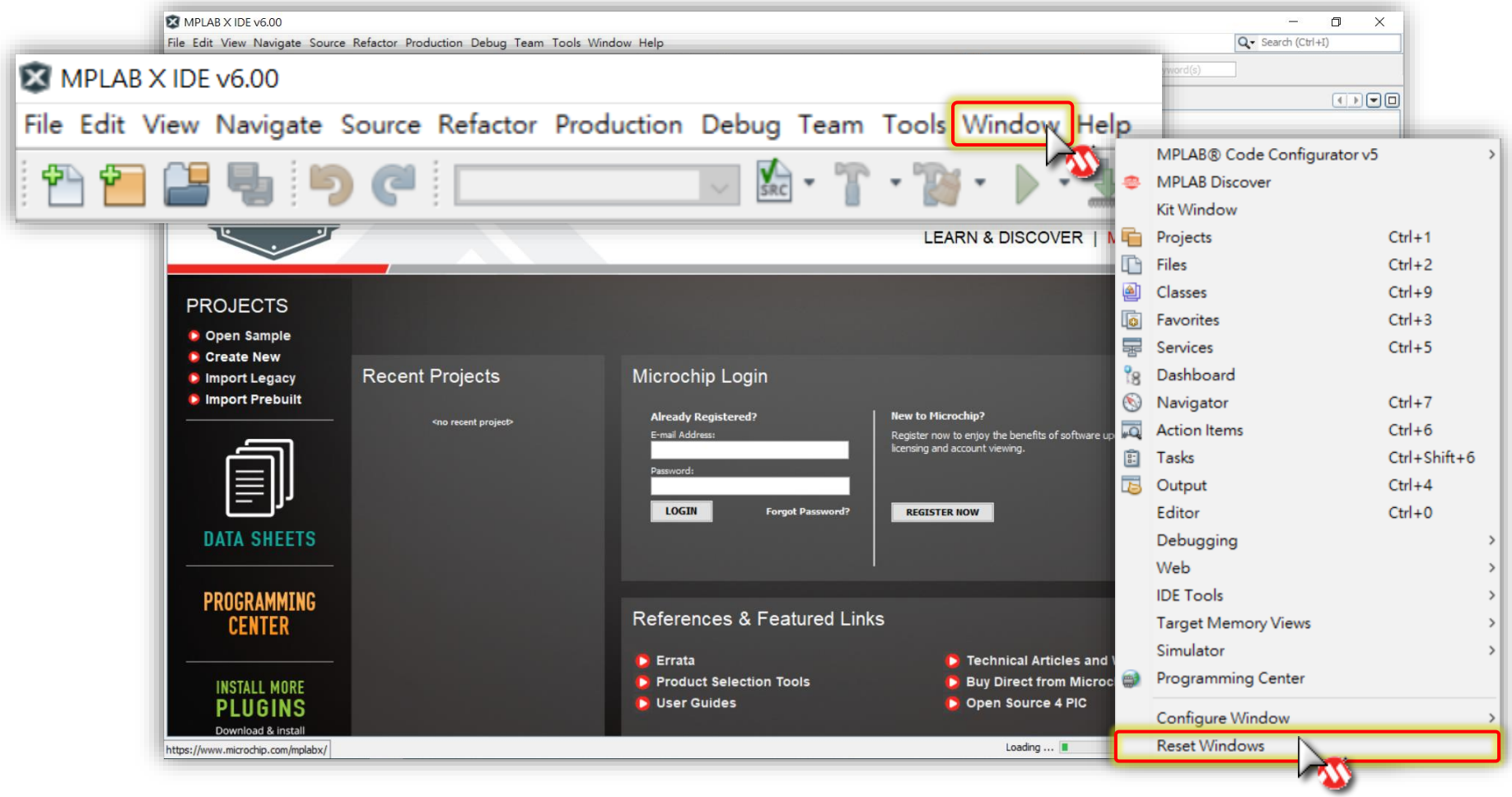
- Base on current project or Open .\Lab1.2_xxx.X to do exercises
- Try to use MCC to generate your first MCC style code and control GPIO base on MCC style.
- Try to set the four GPIO pins to digital output mode and toggle levels (period around 500ms ~ 1S), individually.
- RB03 ▸ LED(LED1).

How to start ?

MPLAB X IDE

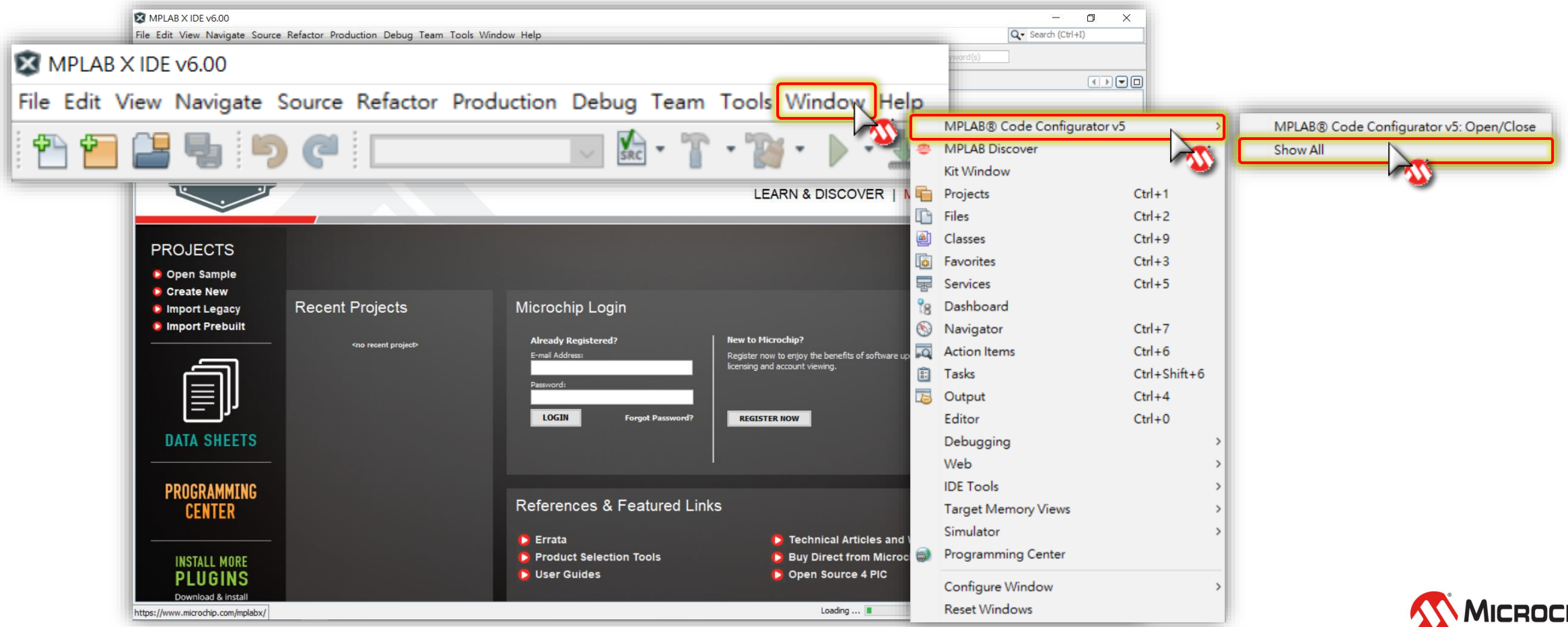
Tips

- **Too many windows, You may can't find the window you want. MPLAB X IDE provide named Reset Windows to go back to default layout.**



MCC Hints

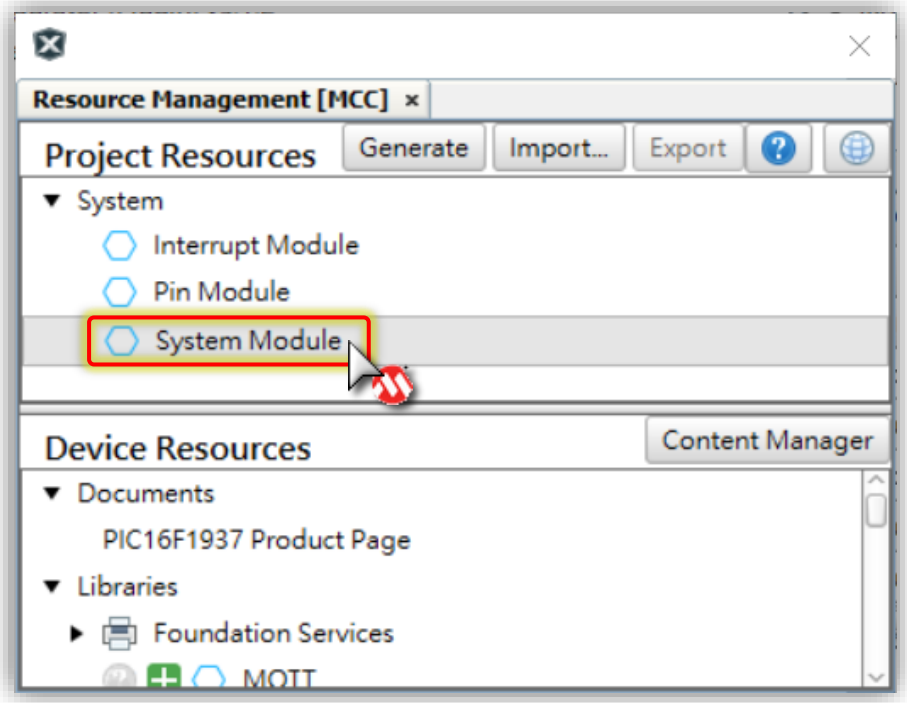
- **You may can’t find the MCC window you want also. MCC provide named Show All to show all MCC windows.**

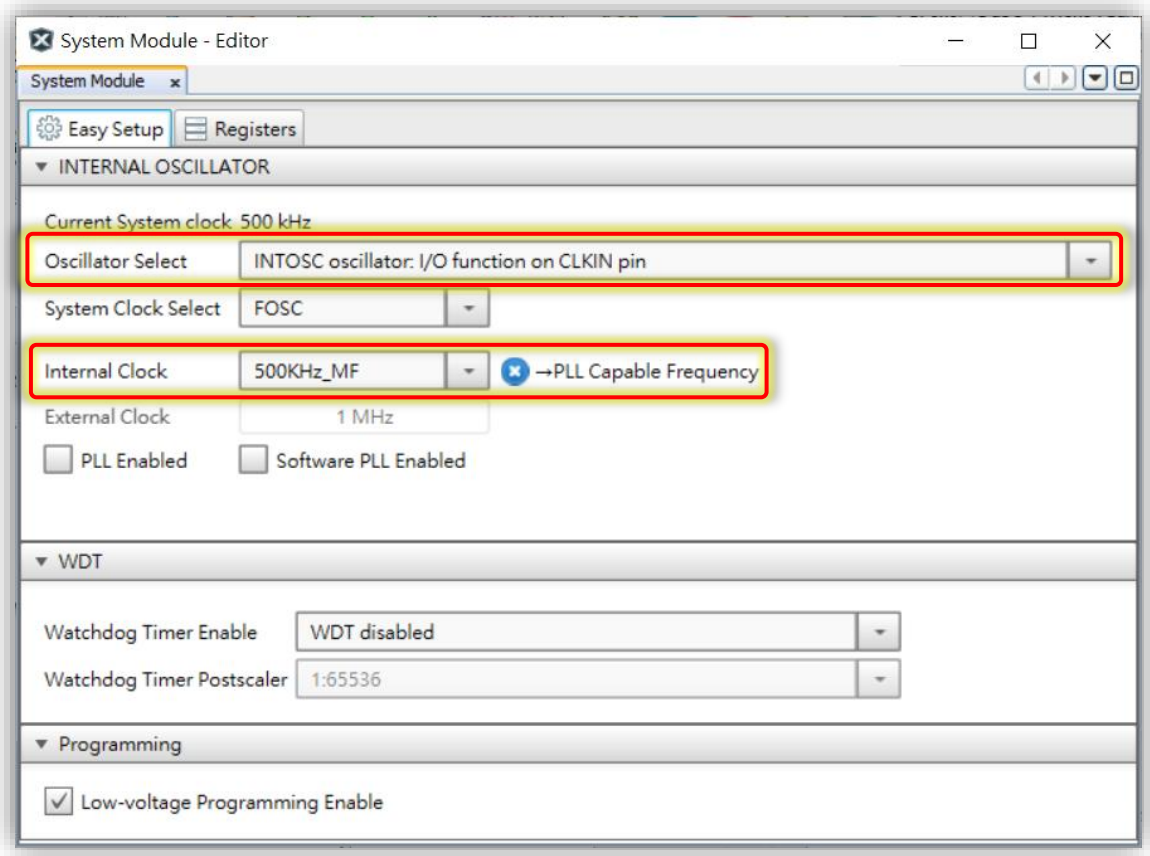


Lab1 GPIO Output

Step 1

- **Set System Clock.**
  - **Oscillator Select : INTOSC oscillator. I/O function on CLKIN pin**
  - **Internal Clock : 500KHz_MF**

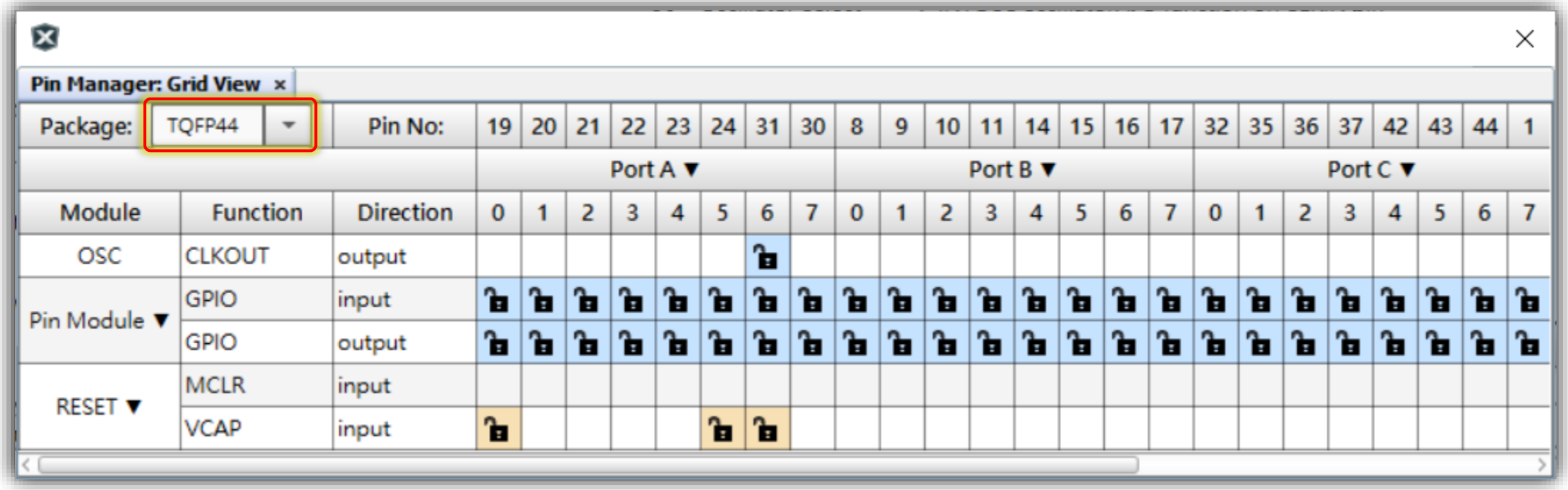




Lab1 GPIO Output

Step 2

- **Change package**
  - **UQFN40 ▸ TQFP44**

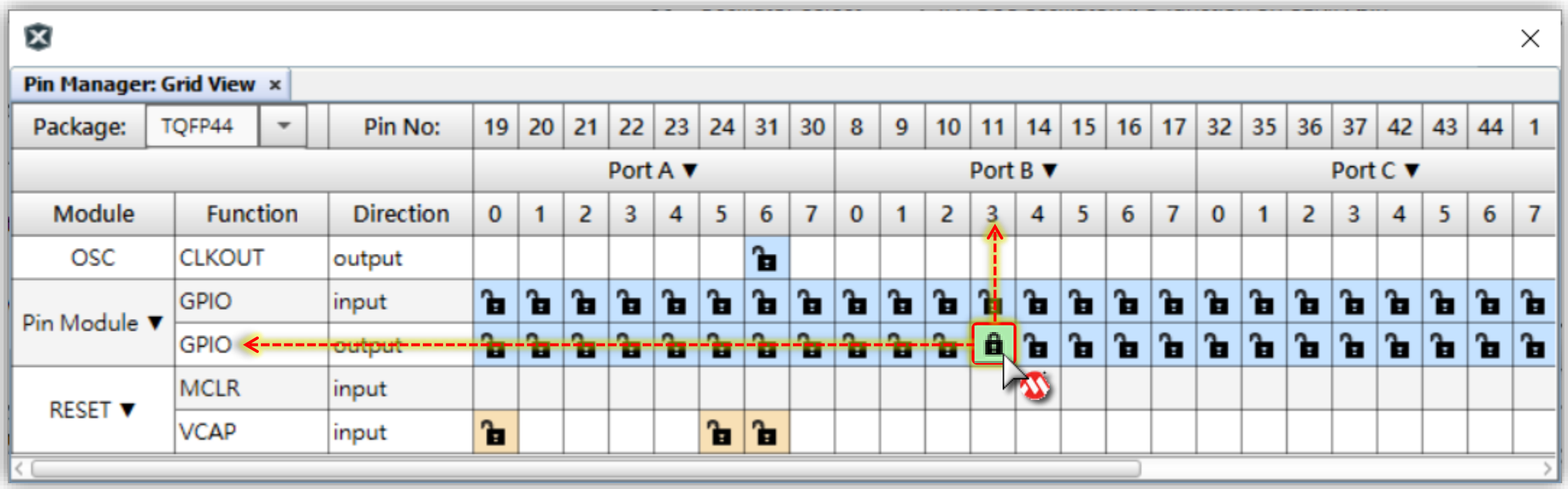


MICROCHIP

Lab1 GPIO Output

Step 3

- Set PORTB, Pin3 (RB3) to digital output mode.
- Pin Manager : Gird View : Click RB3 lock to GPIO output.

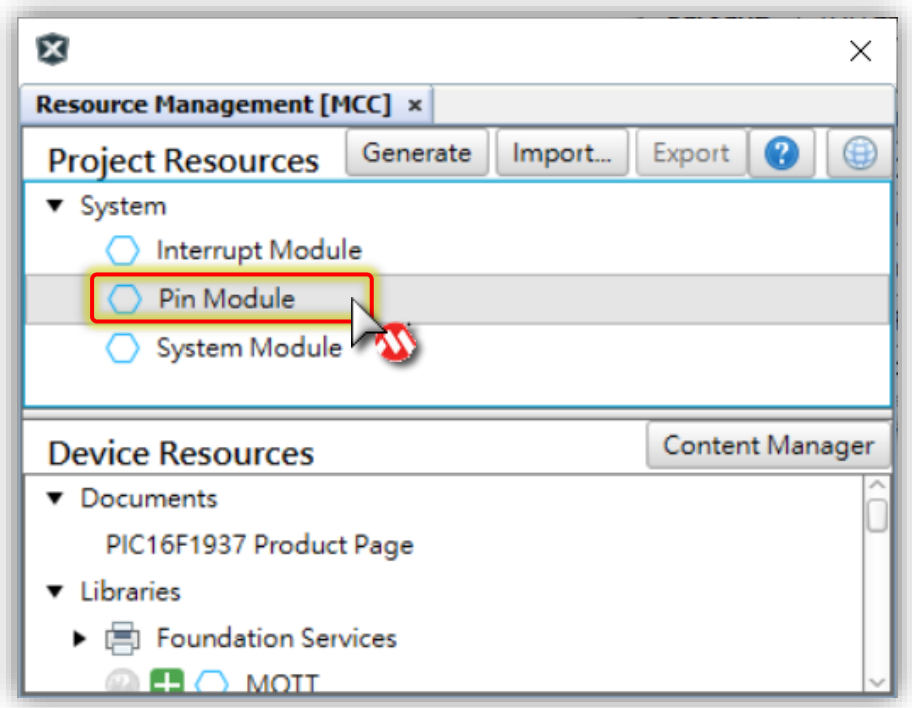


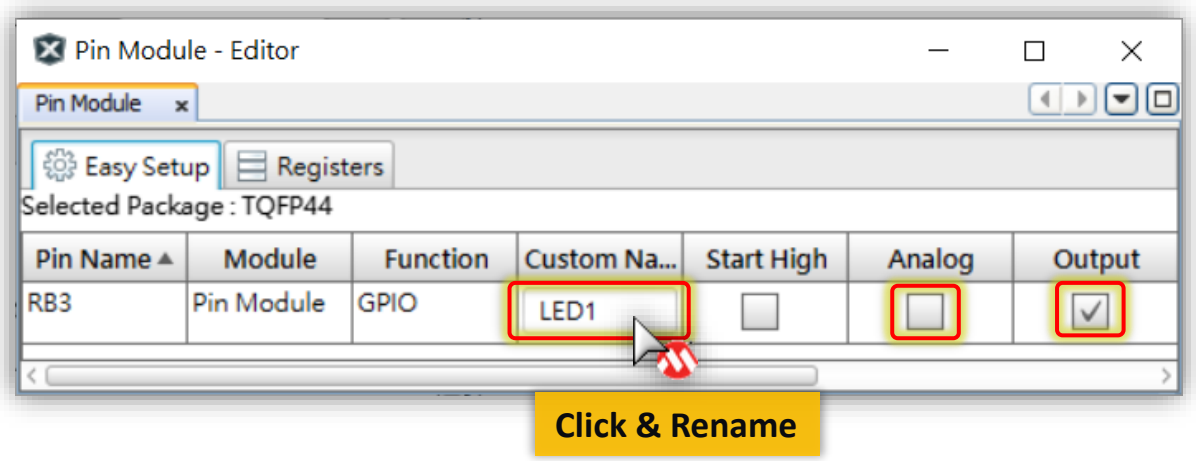
MICROCHIP

Lab1 GPIO Output

Step 4

- **Set Alias for RB3.**
  - **Pin Module : uncheck Analog, Check Output, Custom Name ▸ LED1.**

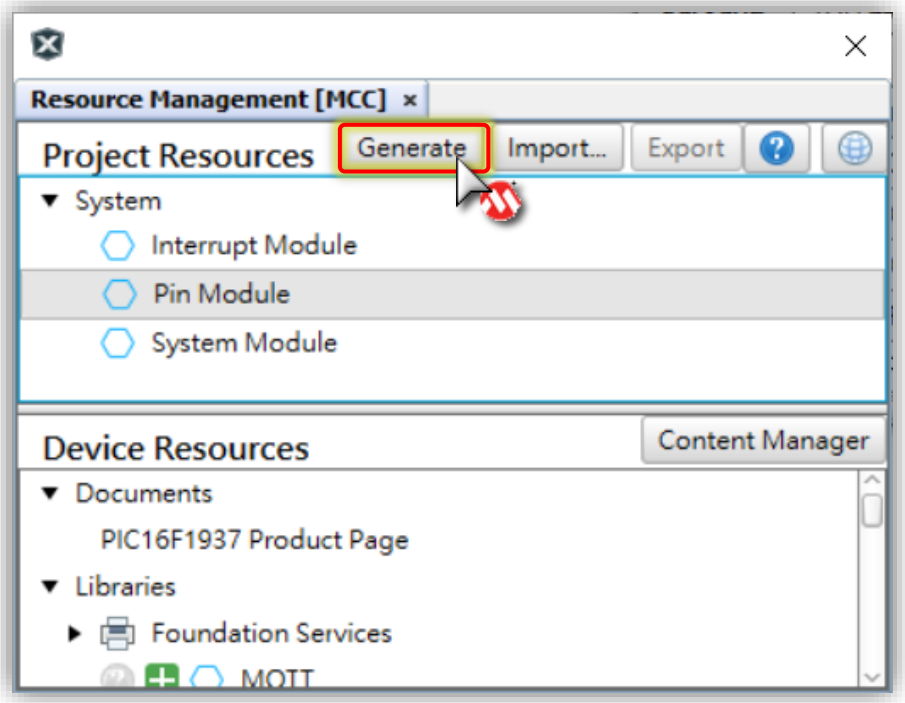


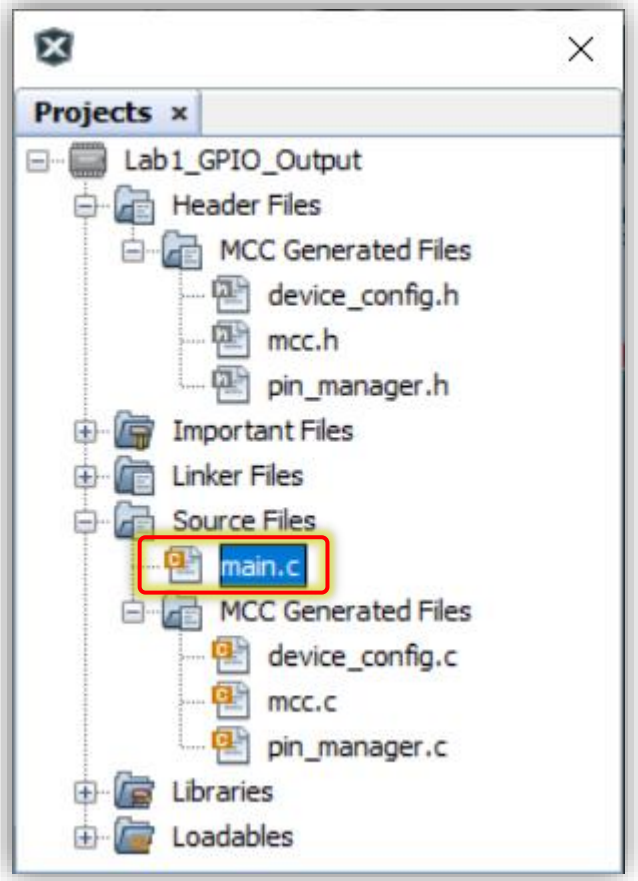


Lab1 GPIO Output

Step 5

- Click Generate Button to generate your code.
- All files generate by MCC include main.c automatically.

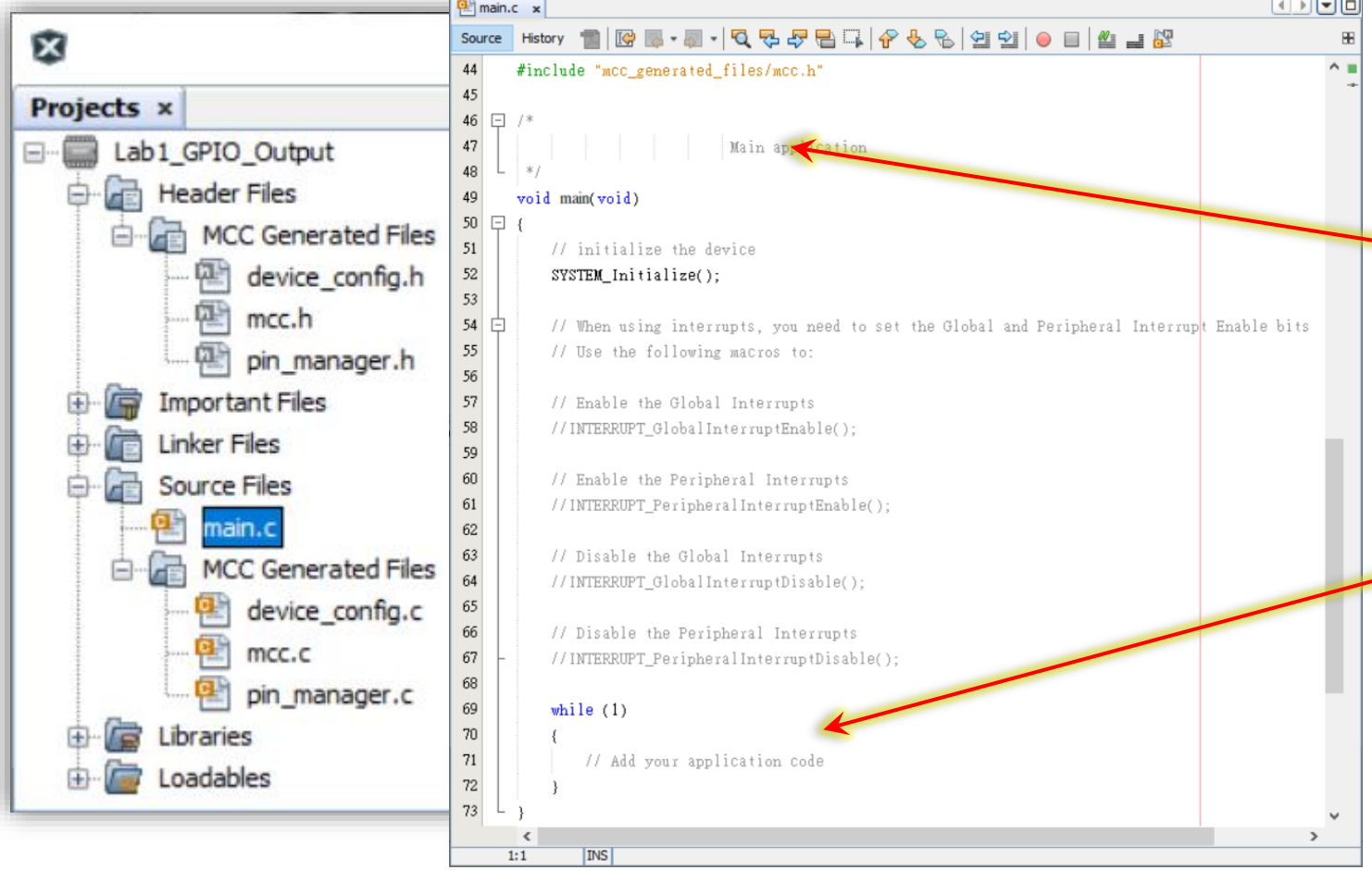




Lab1 GPIO Output

Step 6

- **Try to add code to main function Double Click main.c to view & and add below code segment in it.**

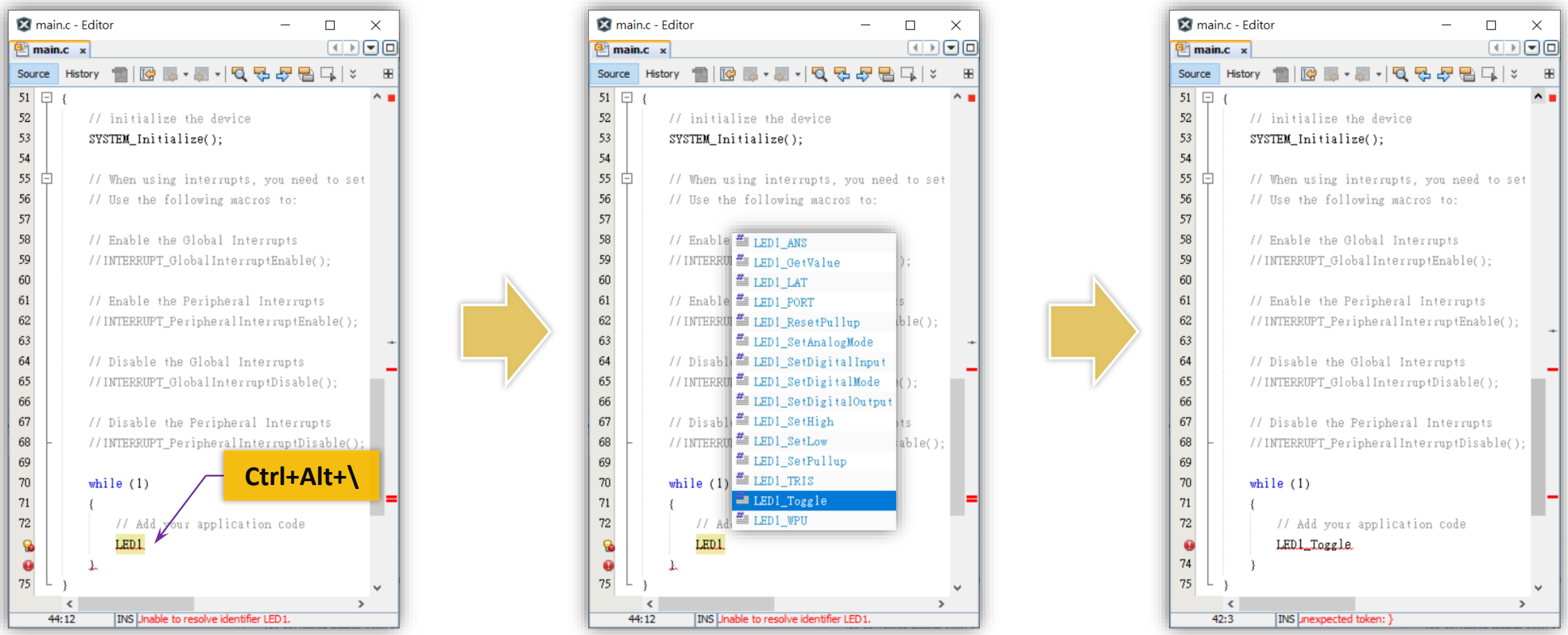


Add below code to main()

```
#include "mcc_generated_files/mcc.h"

uint32_t  i = 0;

int main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    while (1)
    {
        // Add your application code
        LED1_Toggle();
        for (i = 0; i < 2800; i++);
    }
}
```

MPLAB X IDE Hints

- **Hot key "Ctrl + Alt + \"**
  - **Type LED1 first, then use "Ctrl + Alt + \" to find you want.**

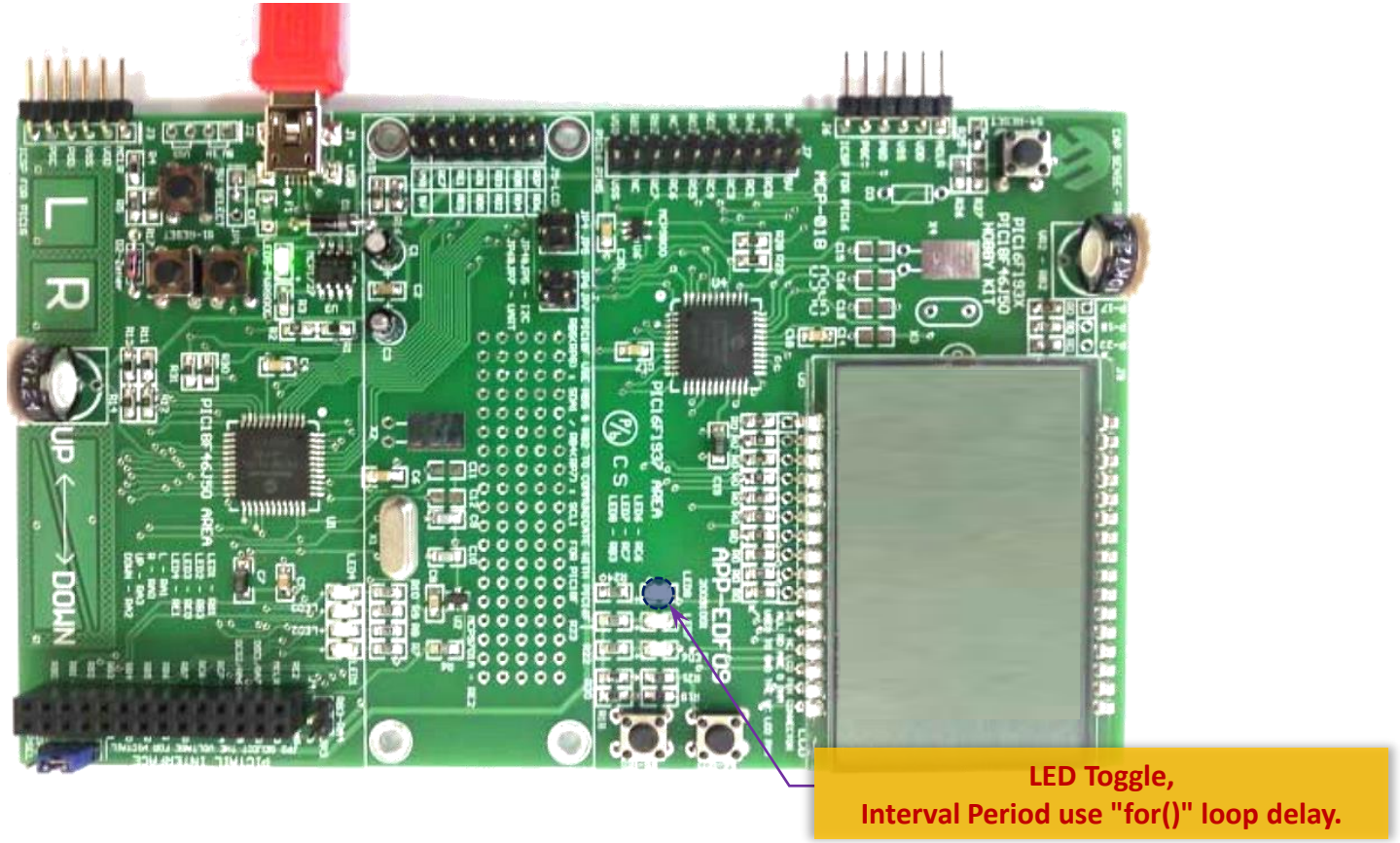


MICROCHIP

Lab1 GPIO Output

Step 7 & 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

Lab1 GPIO Output

Code Example

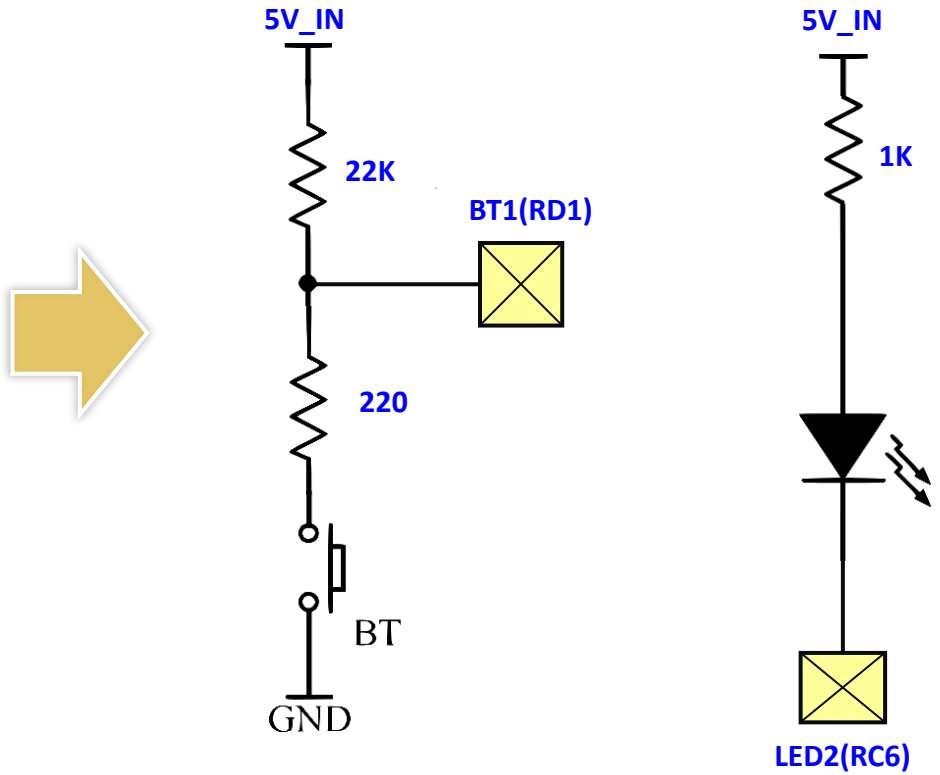
```
#include "mcc_generated_files/mcc.h"

uint32_t  i = 0;

int main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    // When using interrupts, you need to set the Global and Peripheral Interrupt Enable bits
    // Use the following macros to:

    // Enable the Global Interrupts
    //INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptEnable();

    // Disable the Global Interrupts
    //INTERRUPT_GlobalInterruptDisable();

    // Disable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptDisable();

    while (1)
    {
        // Add your application code
        LED1_Toggle();
        for (i = 0; i < 2800; i++);
    }
}
```

Lab2 GPIO Input

Lab2 GPIO Input

- *CustomName*_GetValue(); function is used for get digital input level from outside.
- The simple example code shows below:

```
if(BT1_GetValue())
{
   LED2_SetHigh();
}
else
{
   LED2_SetLow();
}
```



Lab2 GPIO Input

- Base on current project or Open .\Lab2.2_xxx.X to do exercises
- Try to set RD01, RD02 to digital input as get button status.
- Try use button to control LED light or dark.
  - BT1 Pressed -> LED2 Light, BT1 Released -> LED2 Dark
  - BT2 Pressed -> LED3 Light, BT2 Released -> LED3 Dark
- RB03 ▸ LED(LED1).
- RD01 ▸ Button(BT1).
- RD02 ▸ Button(BT2).
- RC06 ▸ LED(LED2).
- RC07 ▸ LED(LED3).

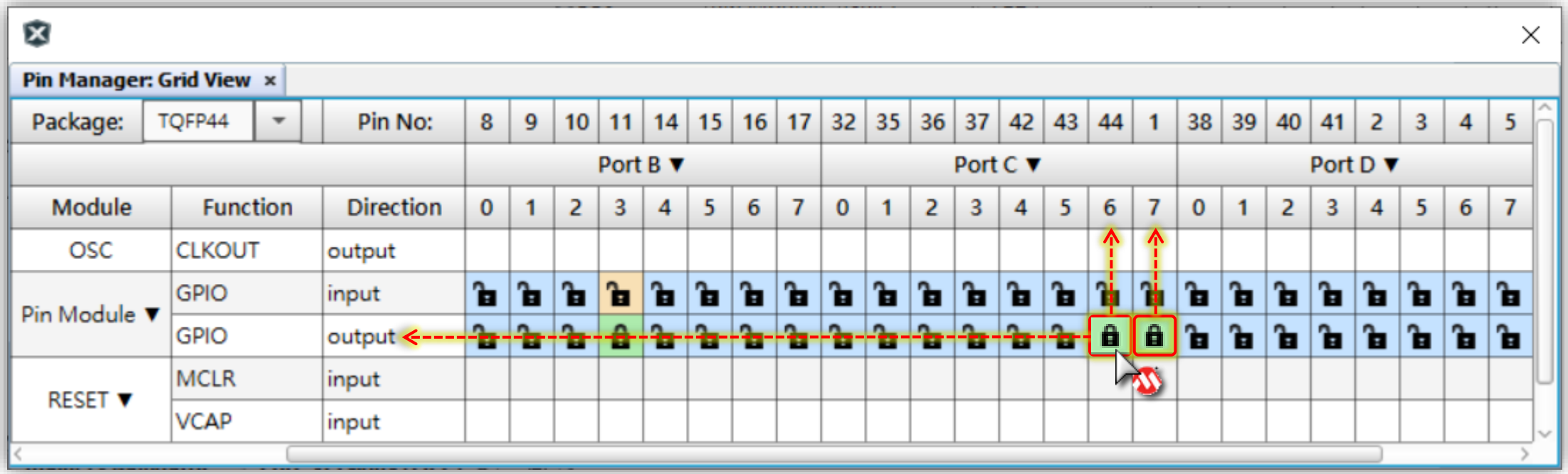
Let’s go!

MICROCHIP

Lab2 GPIO Input

Step 1

- Set PORTC, Pin6 (RC6) , Pin7 (RC7) to digital output mode.
- Pin Manager : Gird View : Click RC6, RC7 lock to GPIO output.

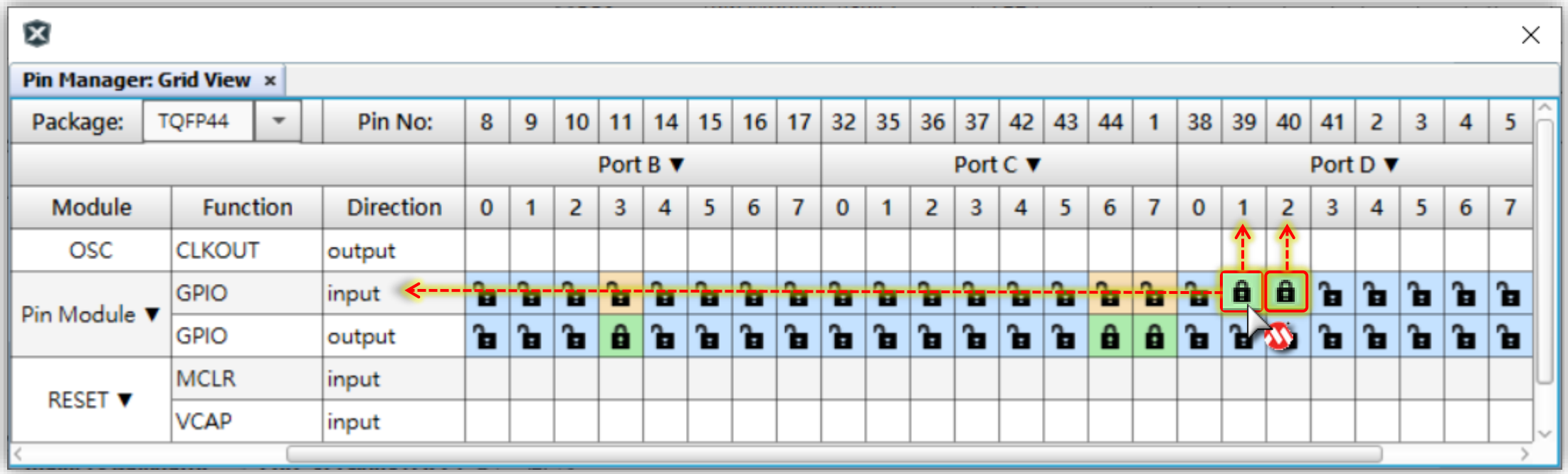


MICROCHIP

Lab2 GPIO Input

Step 2

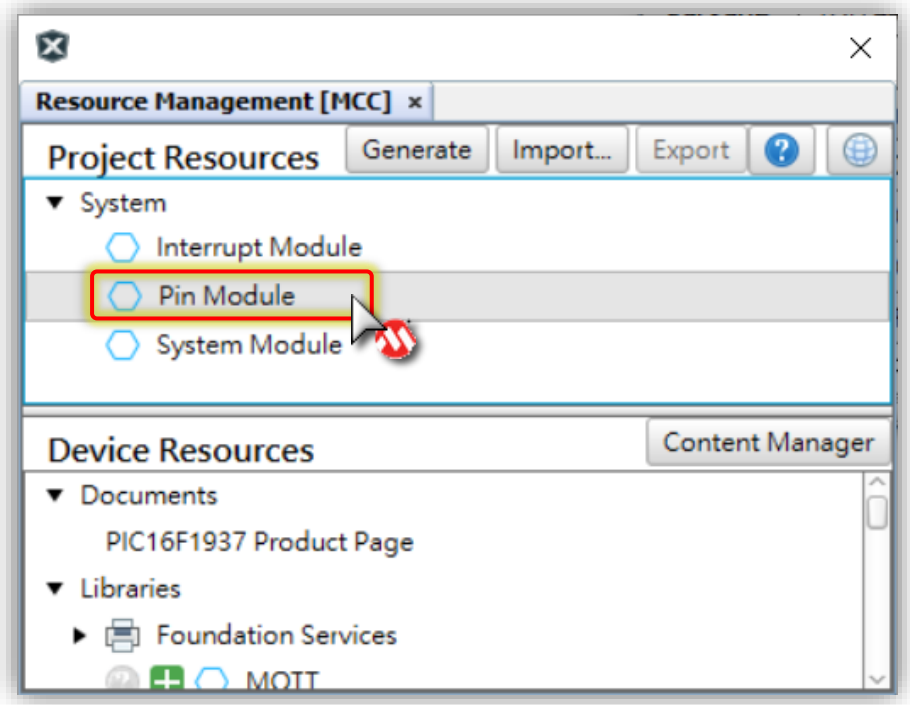
- Set PORTD, Pin1 (RD1) , Pin2 (RD2) to digital input mode.
- Pin Manager : Gird View : Click RD1, RD2 lock to GPIO input.

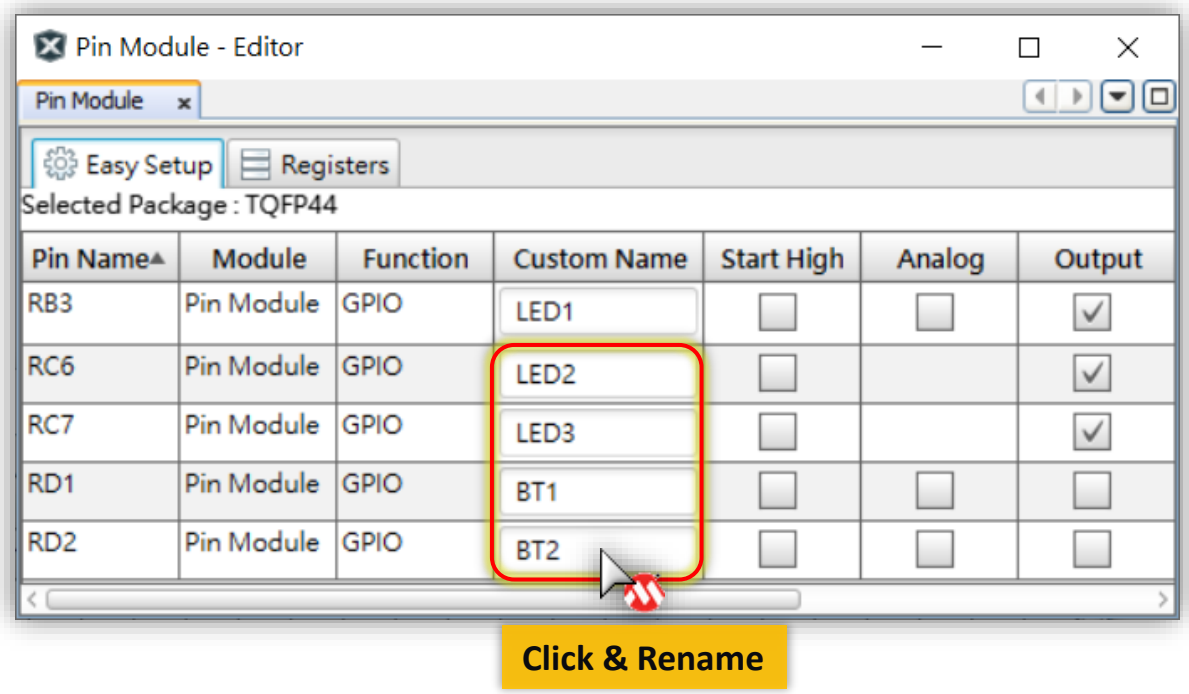


Lab2 GPIO Input

Step 3

- **Set Alias for RC6, RC7, RD1, RD2.**
  - **Pin Module : uncheck Analog, Check Output, Custom Name ▸ LED1.**

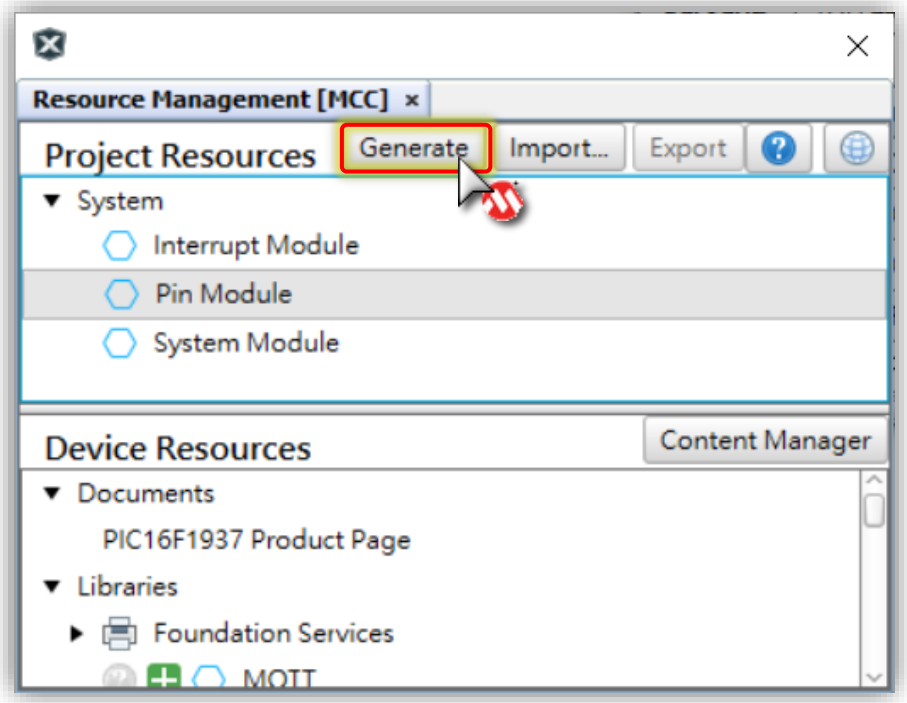


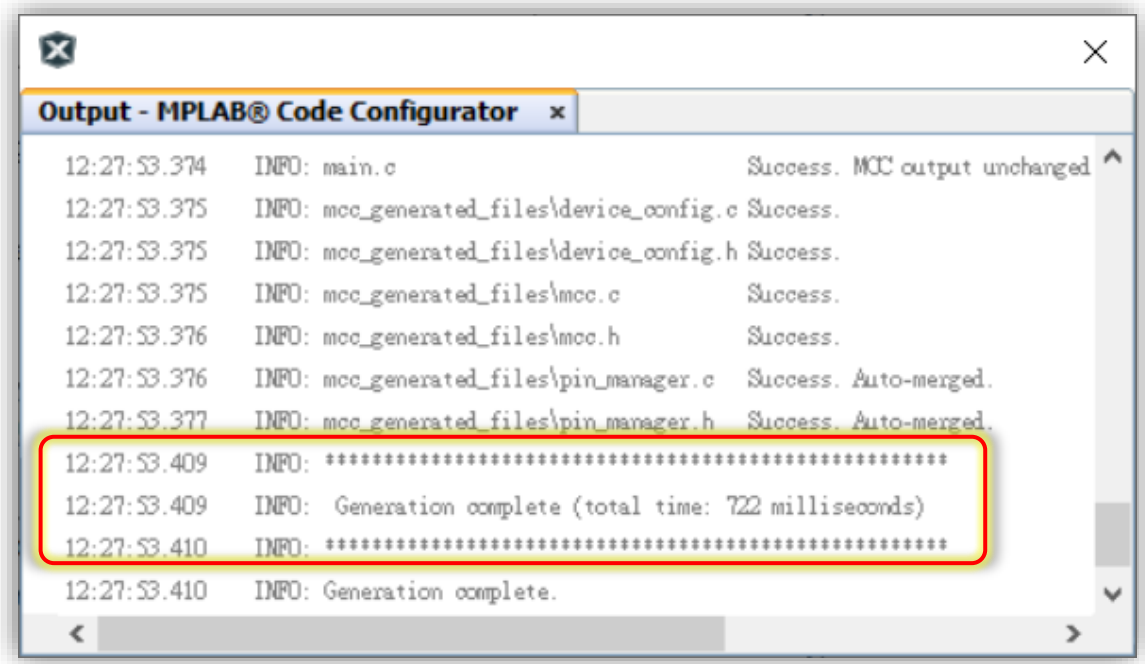


Lab2 GPIO Input

Step 4

- Click **Generate** Button to generate your code.





MICROCHIP

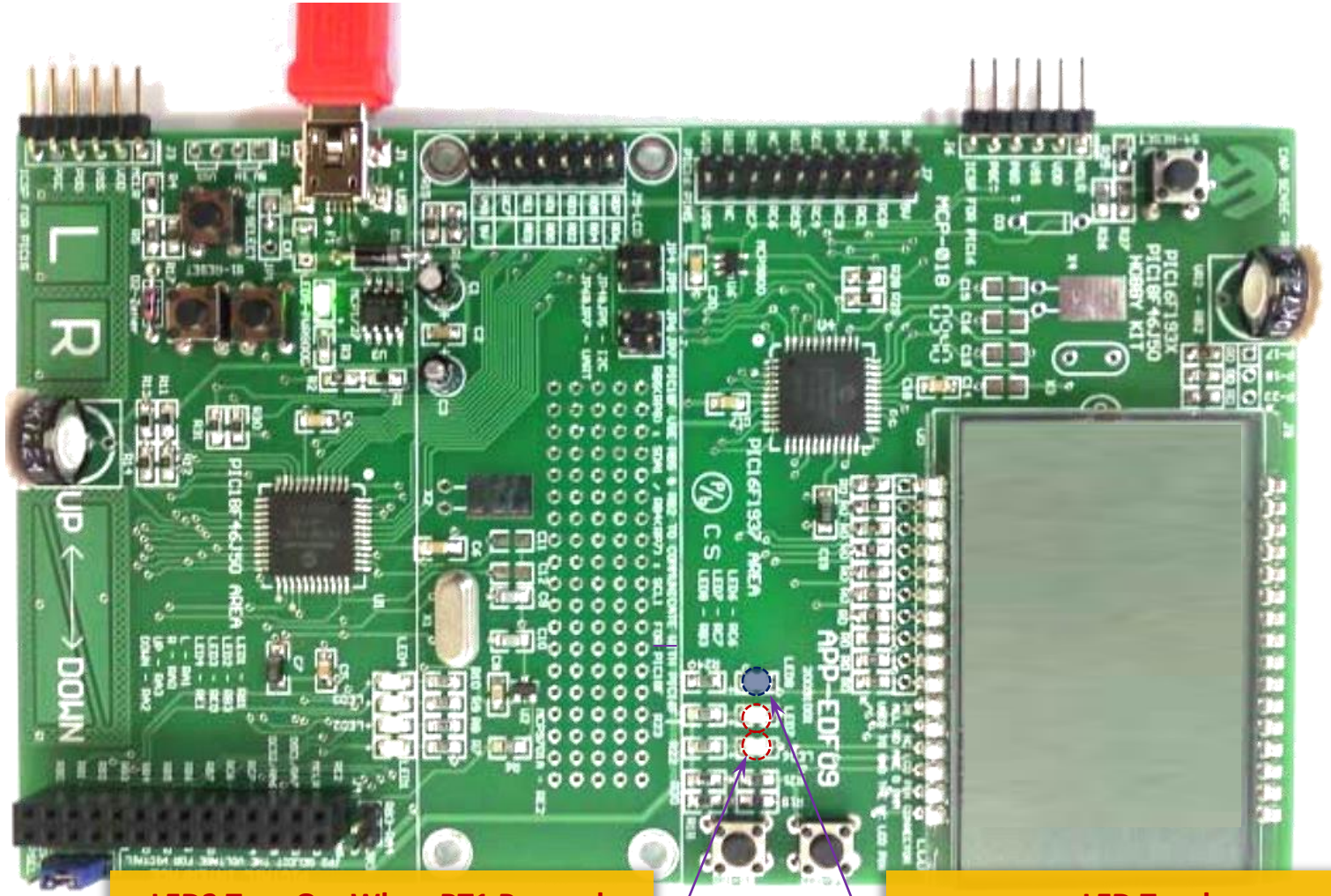
Lab2 GPIO Input

Code Example

```
#include "mcc_generated_files/mcc.h"

uint32_t  i = 0;

int main(void)
{
    …

    // Disable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptDisable();

    while (1)
    {
        // Add your application code
        LED1_Toggle();
        for (i = 0; i < 2800; i++);

        if(!BT1_GetValue())
        {
            LED2_SetLow();
        }
        else if(!BT2_GetValue())
        {
            LED3_SetLow();
        }
        else
        {
            LED2_SetHigh();
            LED3_SetHigh();
        }
    }
}
```

Lab2 GPIO Input

Result



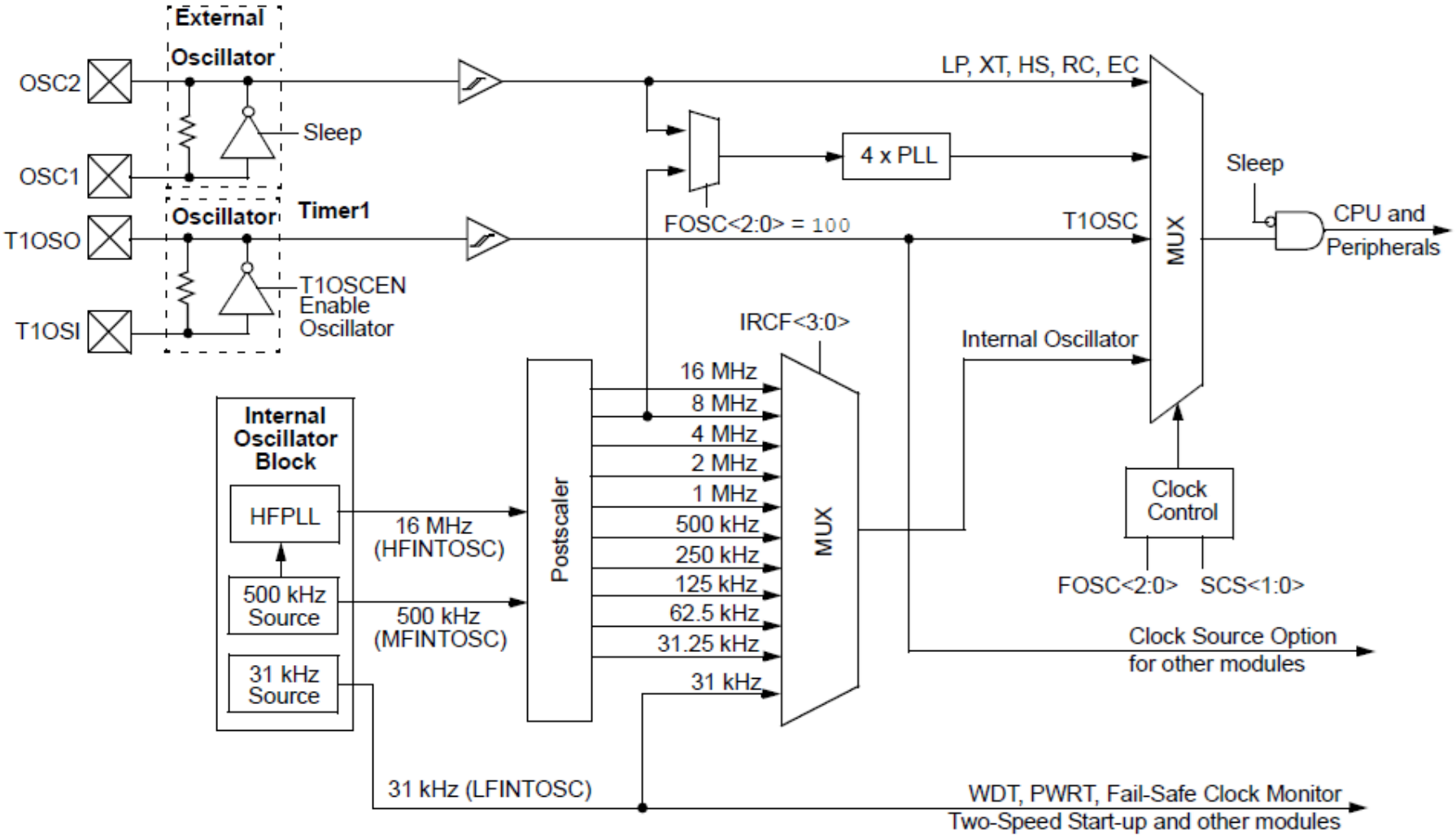
LED2 Turn On, When BT1 Pressed.
LED3 Turn On, When BT2 Pressed.

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

Oscillator Architecture

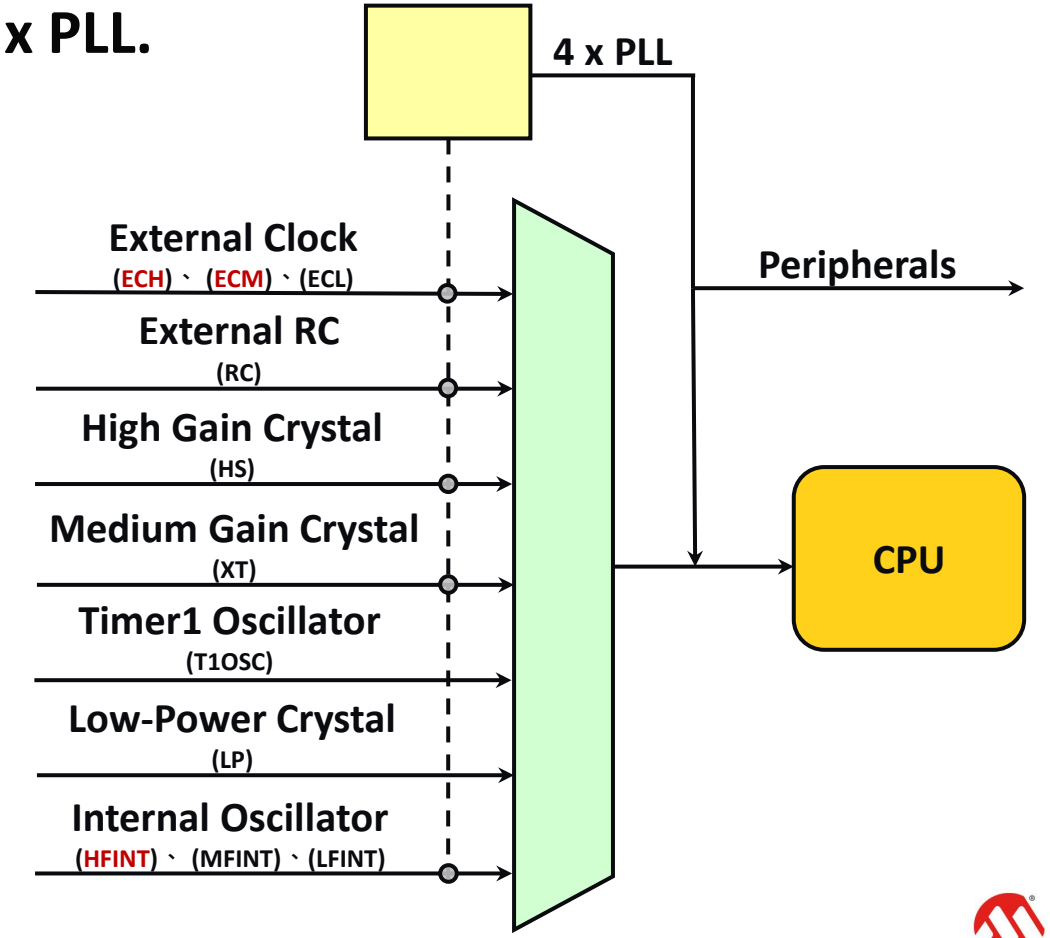
Phase Locked Loop

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



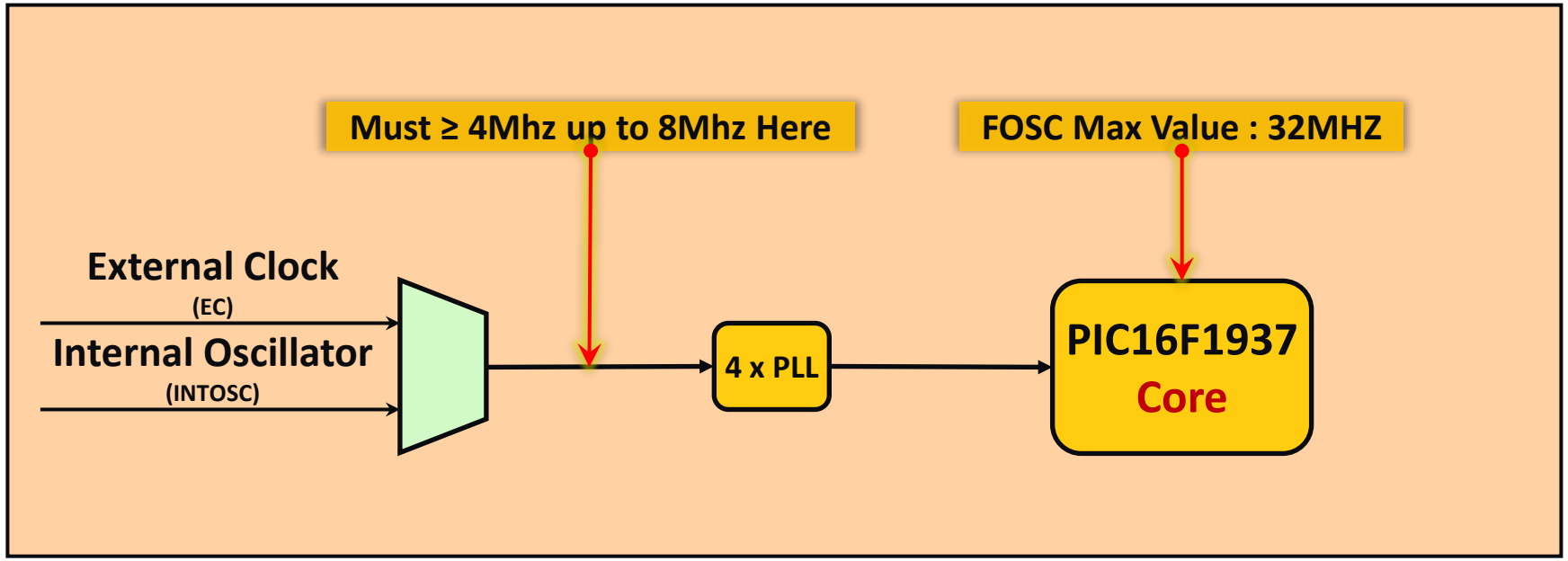
PIC16F Series Clocks

- **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

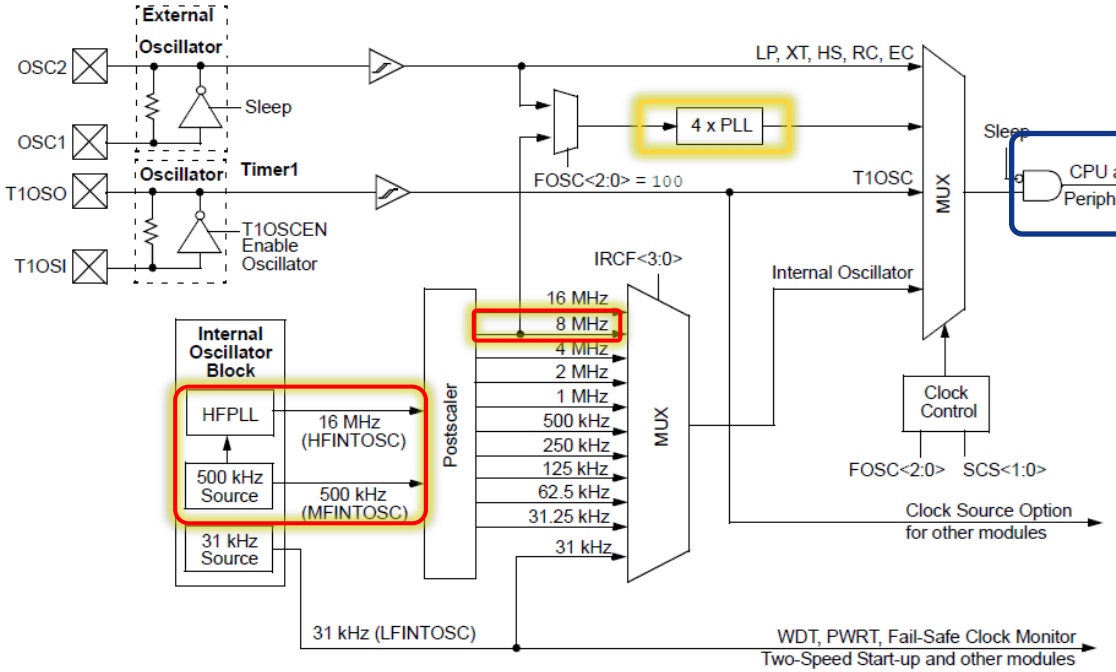


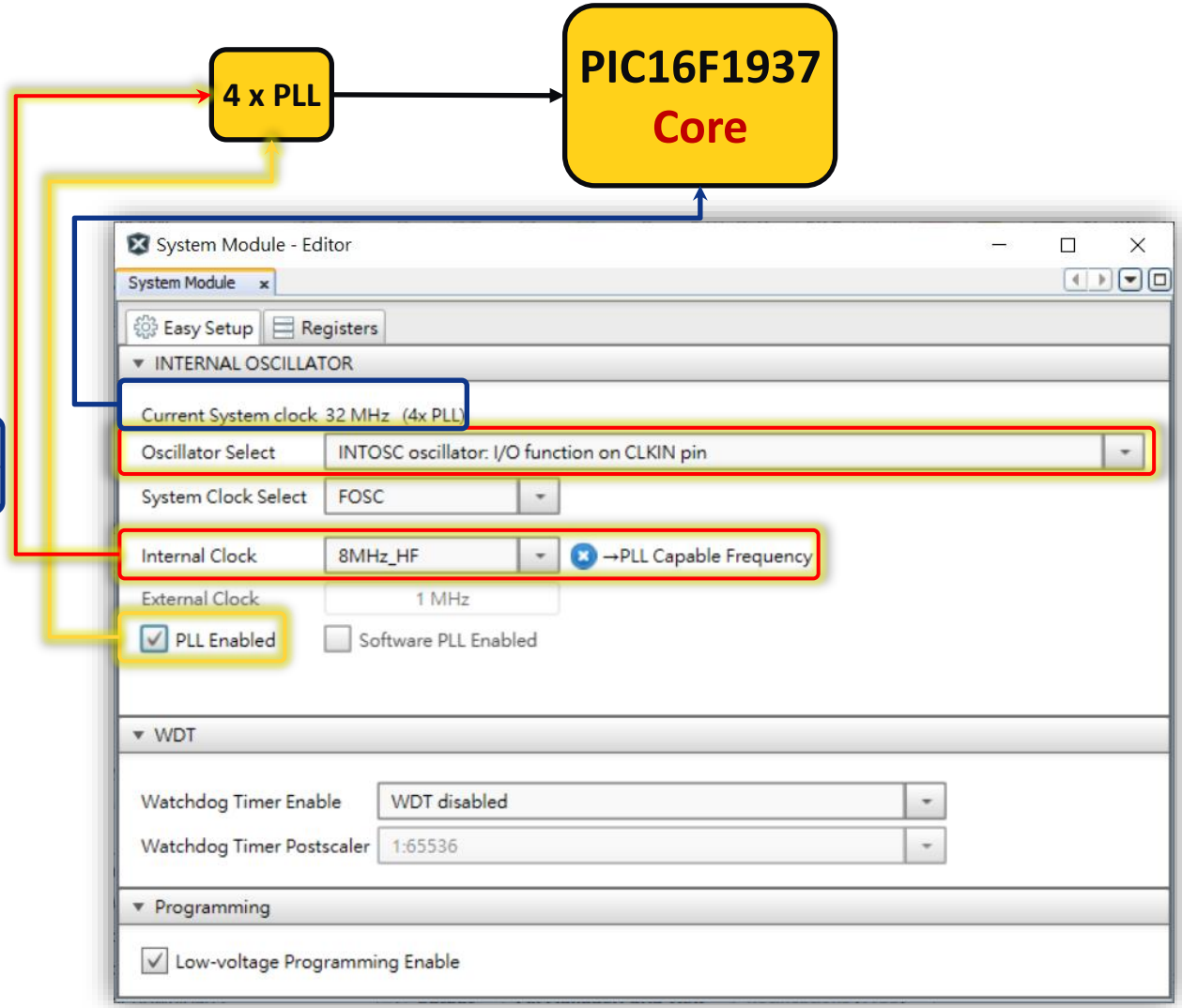
Phase Locked Loop

- 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 Setting of MCC





MICROCHIP

Lab3 Clock PRIPLL

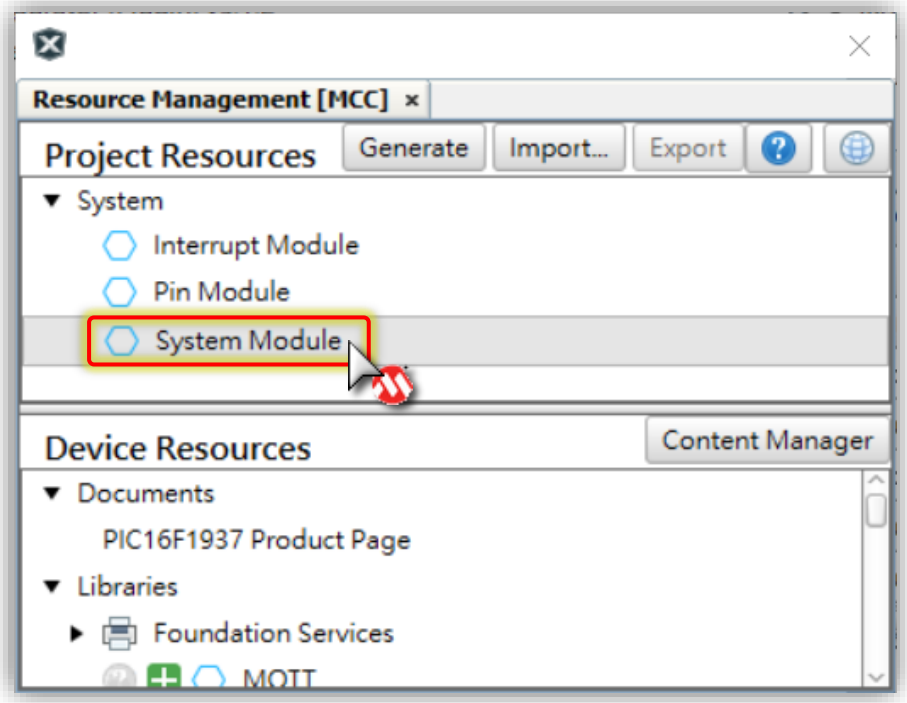
- Base on current project or Open . \Lab3.2_xxx.X to do exercises
- Try provide maximum frequency (4MHz) to MCU.

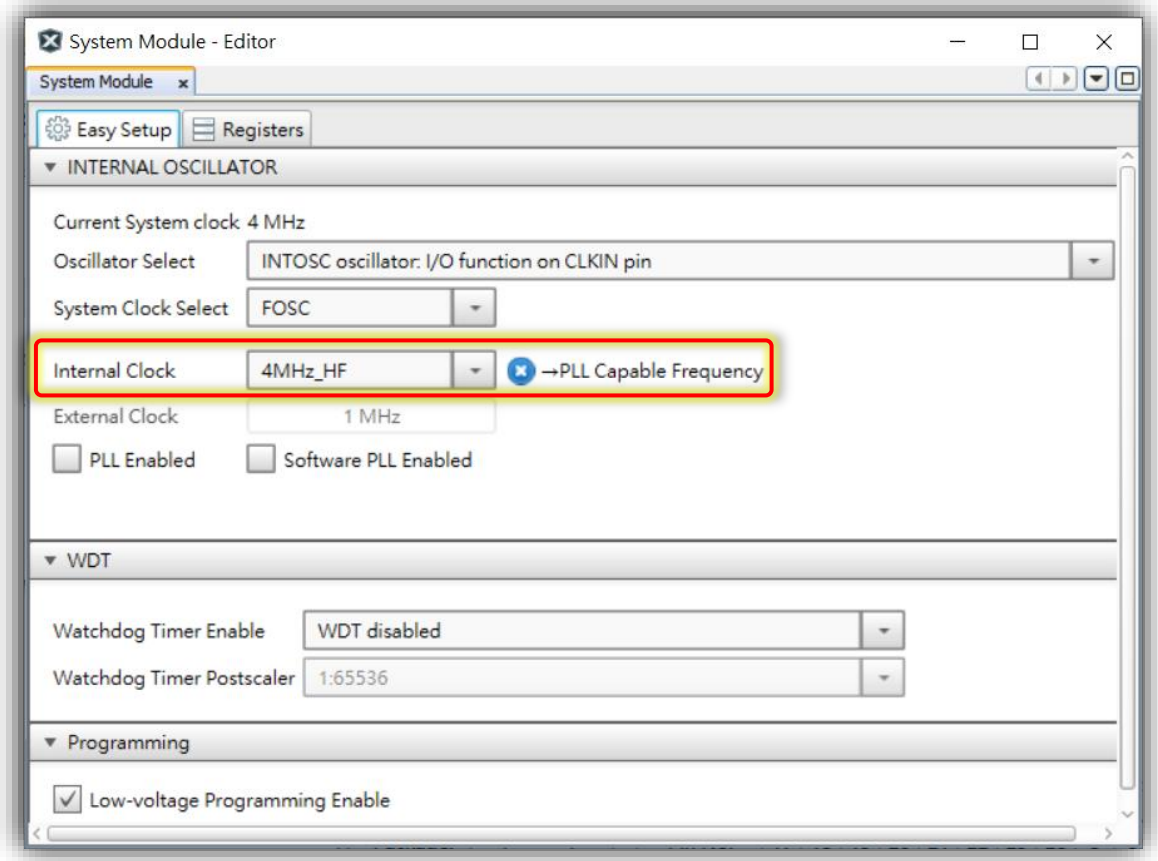
Let’s go!

Lab3 Clock PRIPLL

Step 1

- **Set Internal Clock.**
  - **Internal Clock : 500KHz_MF ▸ 4MHz_HF**



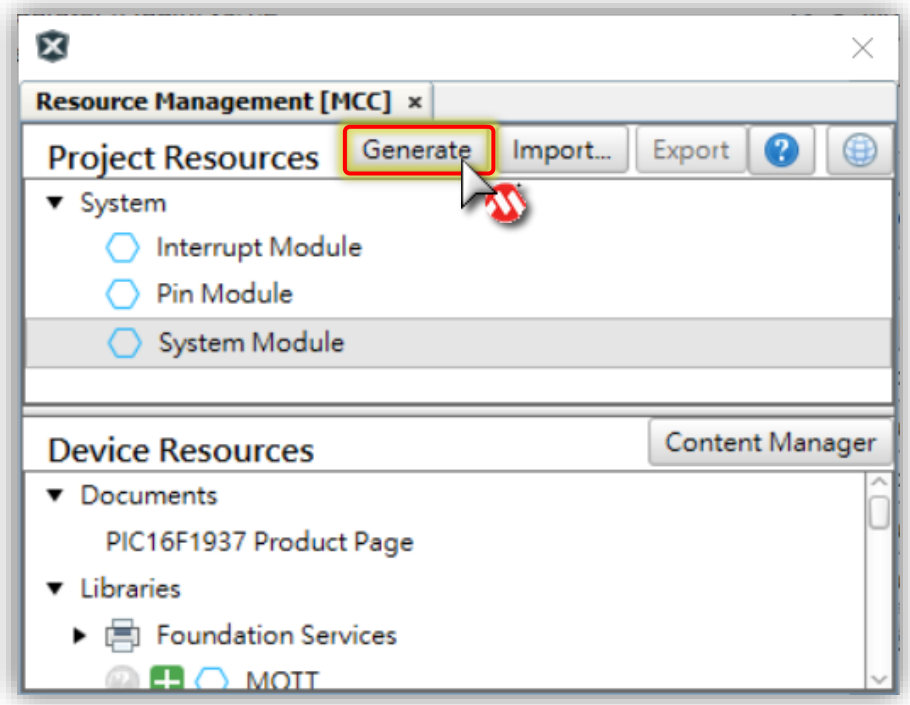


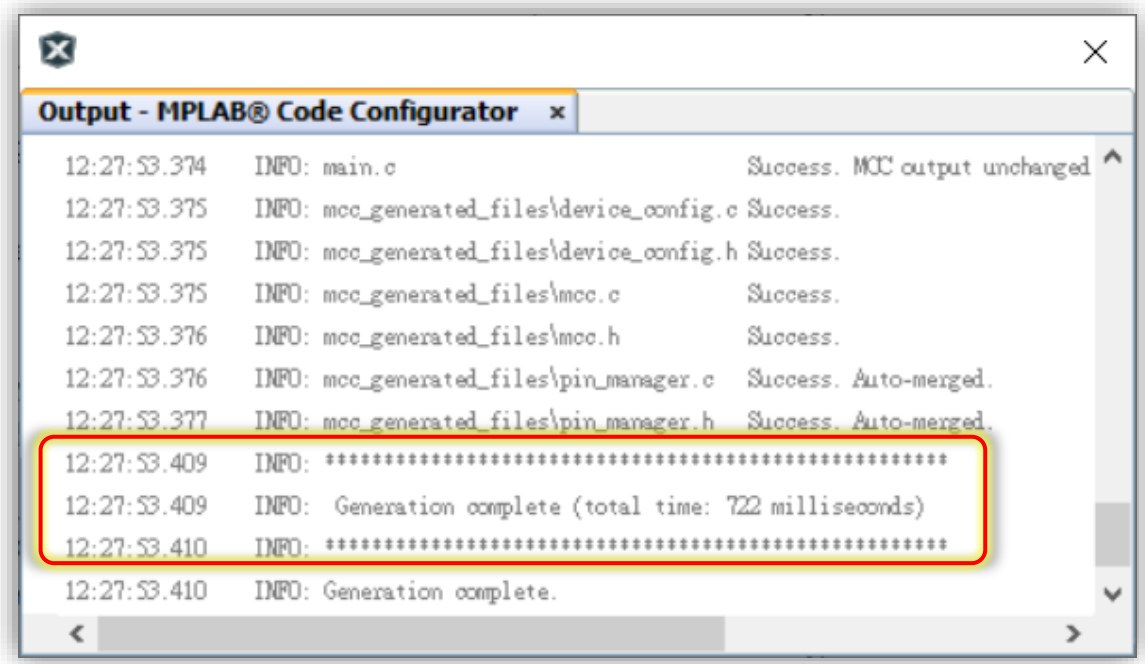
MICROCHIP

Lab3 Clock PRIPLL

Step 2

- Click **Generate** Button to generate your code.

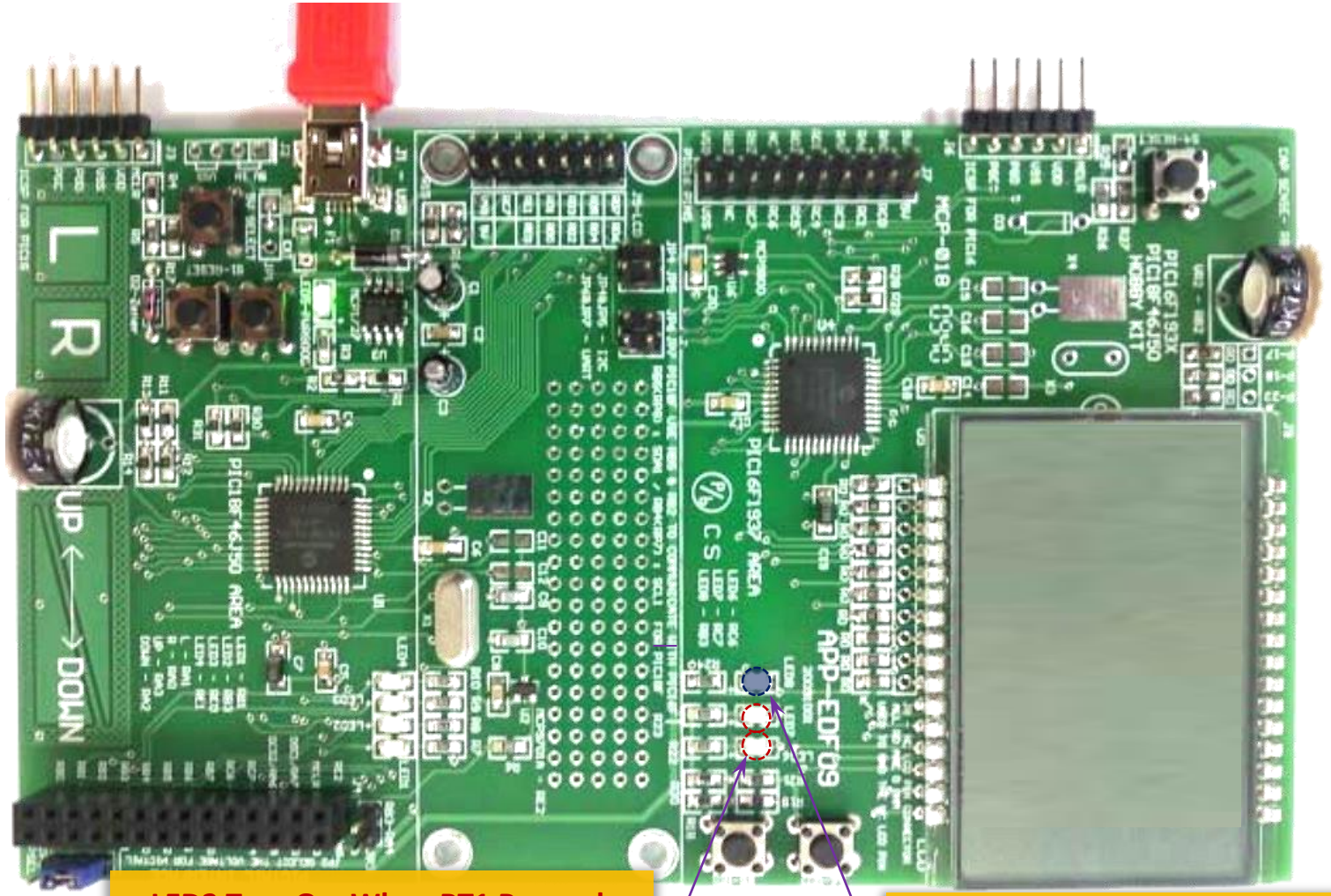




MICROCHIP

Lab3 Clock PRIPLL

Result



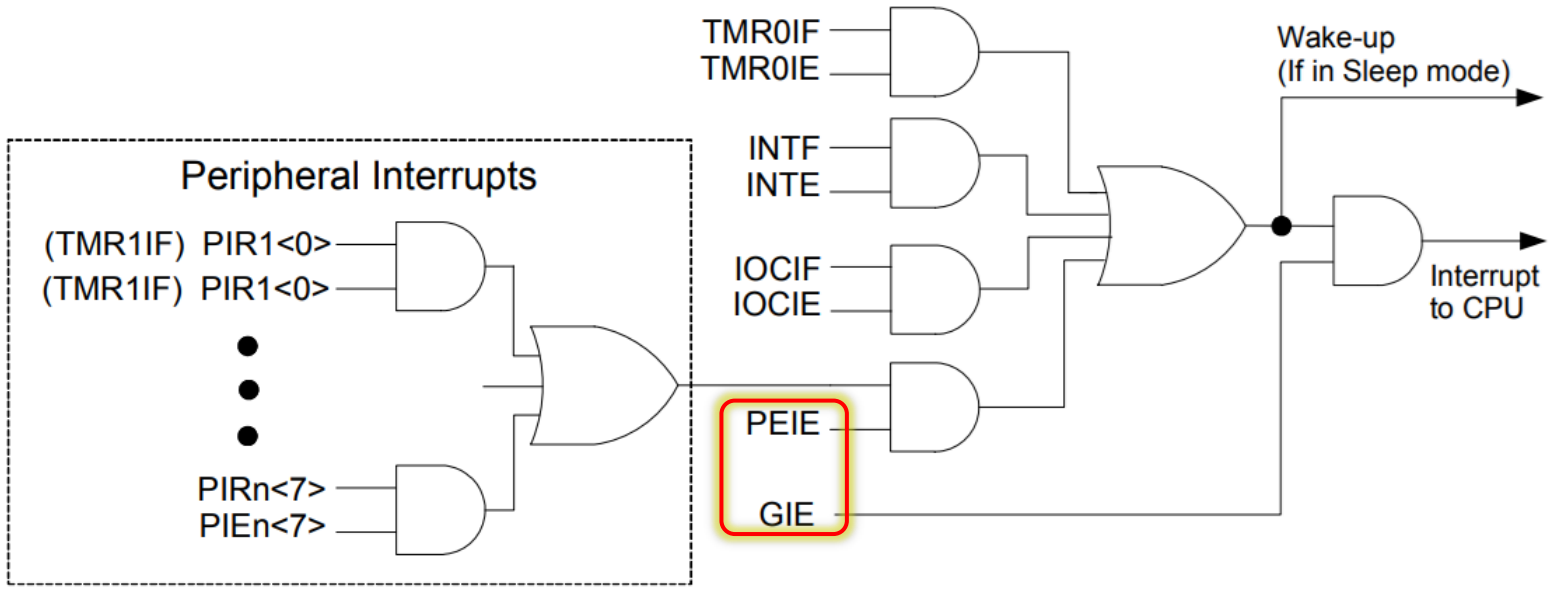
Interrupt Architecture

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

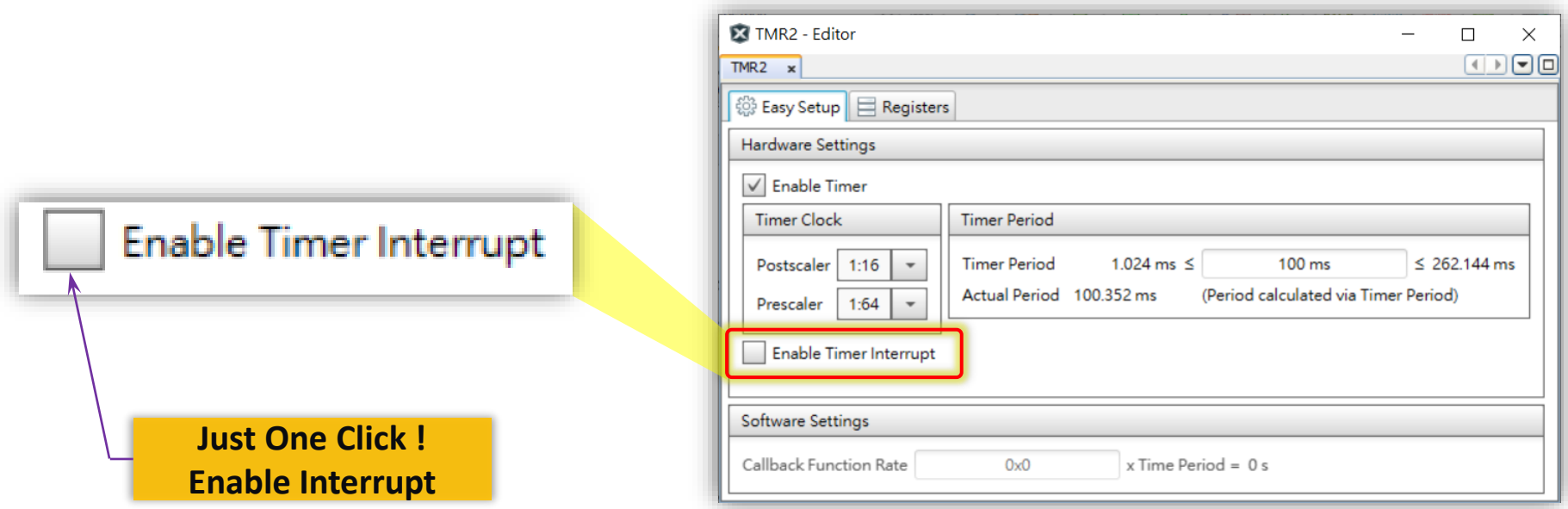
- The interrupt enabled by setting the GIE, PEIE, and Interrupt Enable bit(s).
- PIC16F has only one interrupt entry point, address is 0x0004.
- Support multi-interrupt event processing, but no priority function.
- Interrupt priority is determined by polling in sequence.
- PIC16F background storage adopts Shadow mode.



Interrupt logic

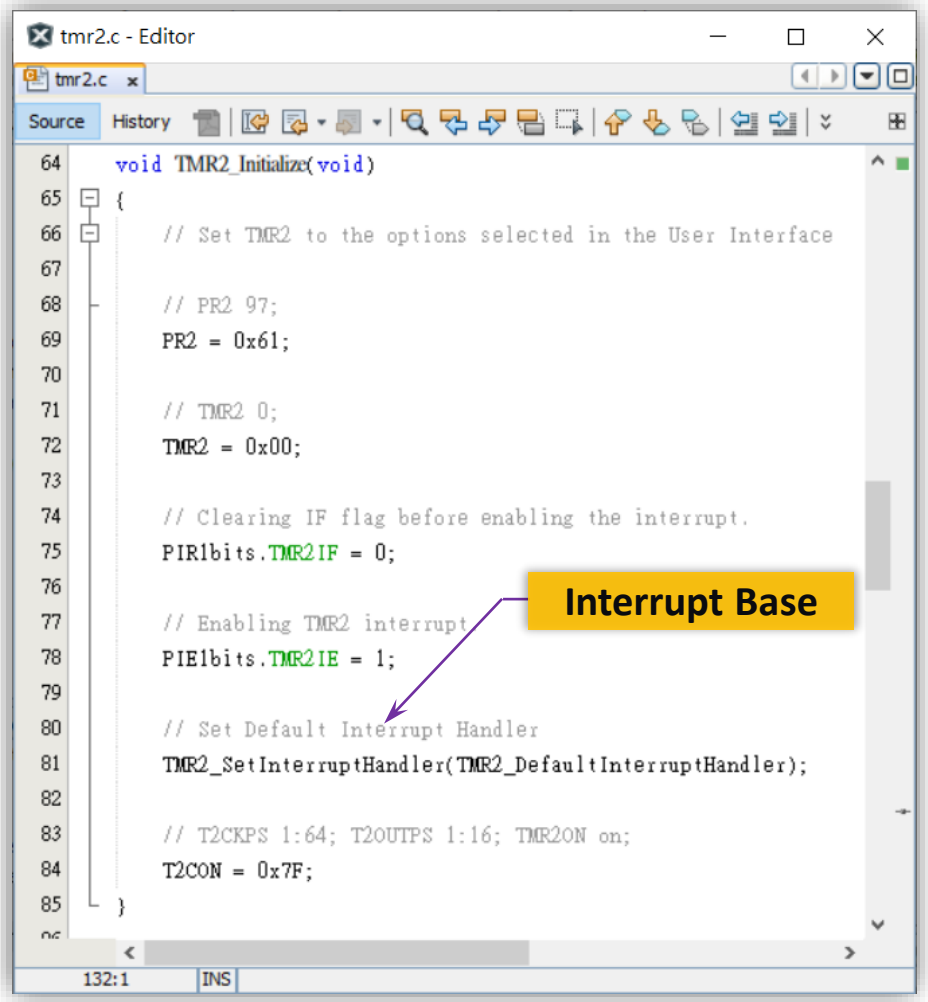
Coding for Interrupt

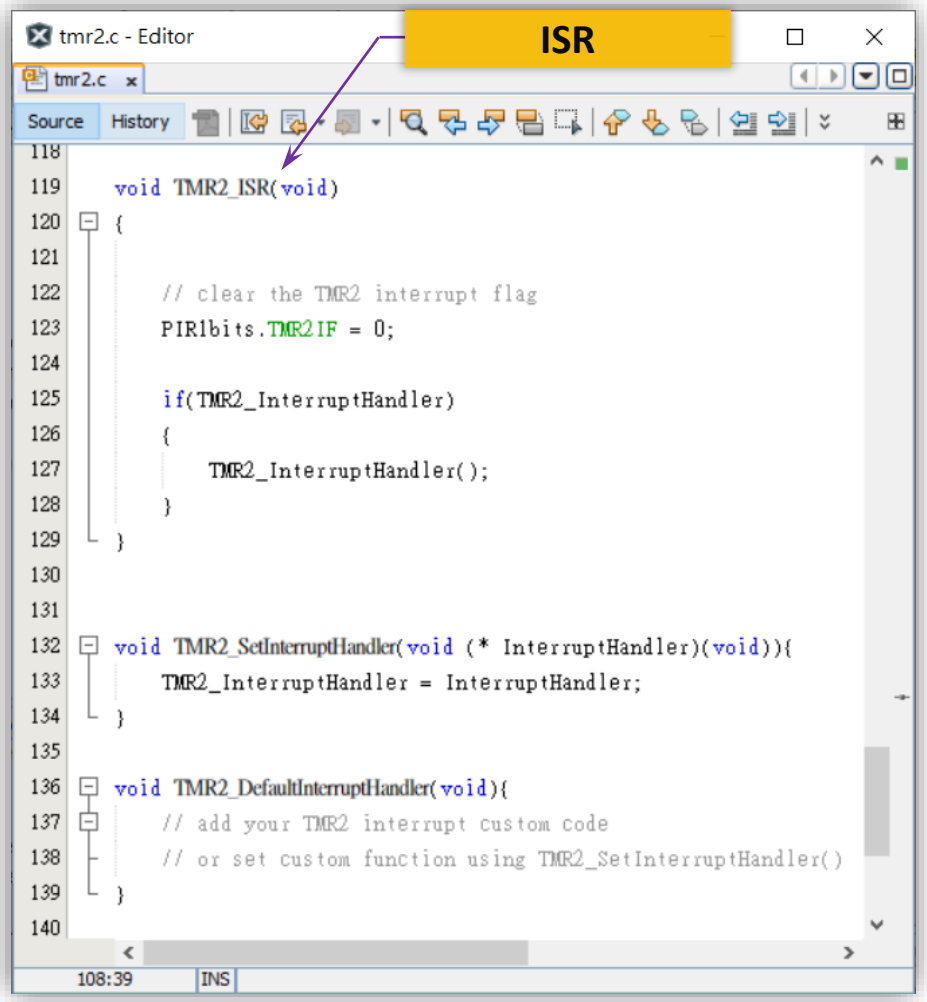
- **A complete interrupt code segment includes below 3 parts.**
  - Interrupt Service Routine (ISR)
  - Enable Interrupt
  - Initial Peripheral to generate Interrupt Request.
- **This is very difficult for beginner, if you use bare metal style to build your code.**
- **MCC could help you to build most all code segment. So, you can get easy to handling interrupts.**
- **Just one click all interrupt relate setting will get ready.**



MCC's Interrupt Function

- **MCC enable interrupt, set priority and create interrupt service routine automatically.**

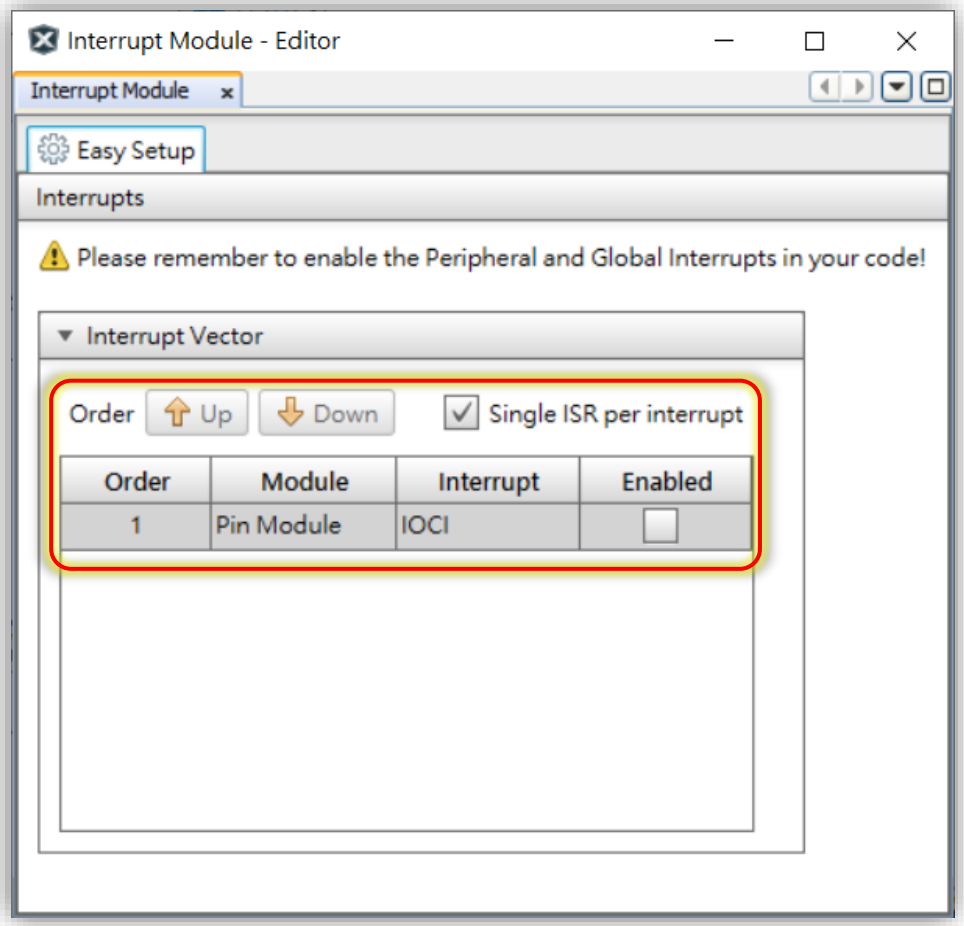




MICROCHIP

MCC’s Interrupt Manager

- **MCC’s Interrupt Module allow user to manager interrupt enable/disable and priority.**



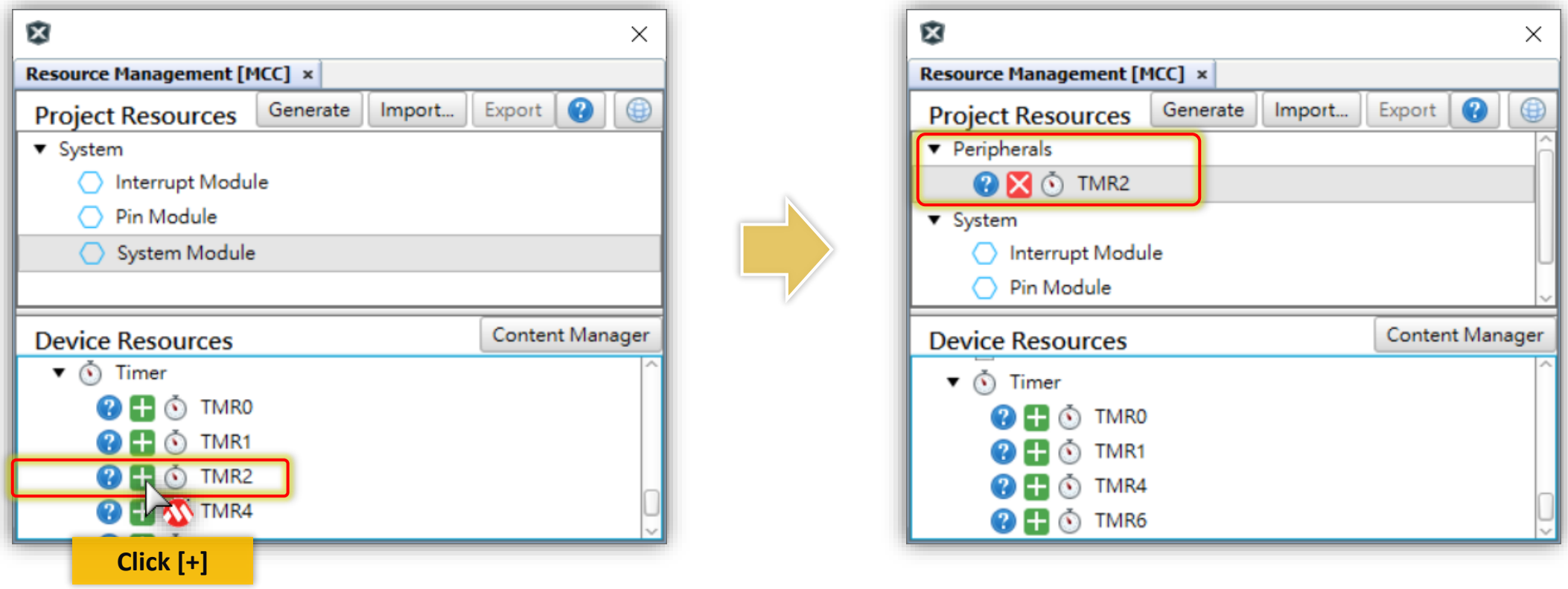
Timer Architecture

Timer or Counter ?

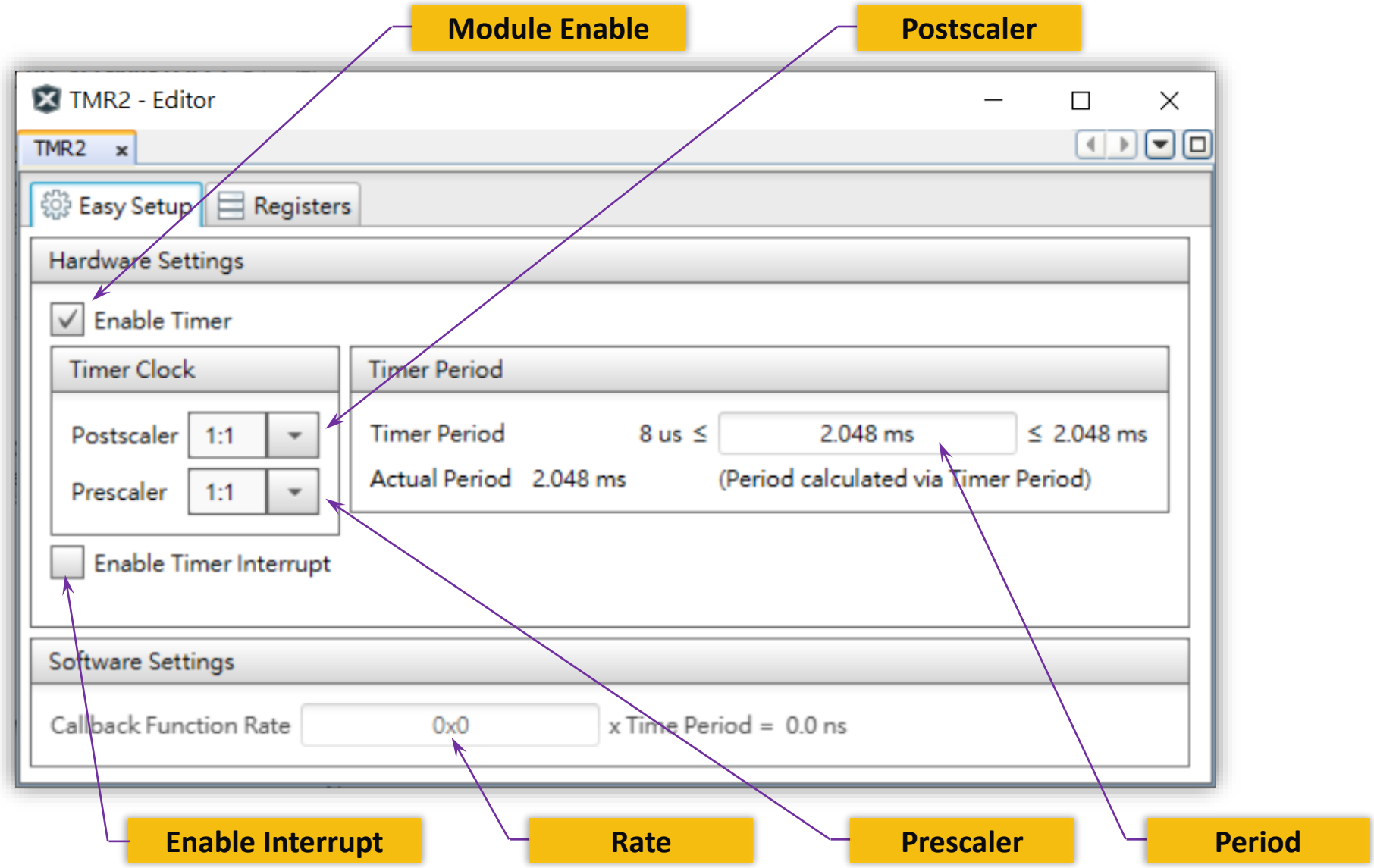
- **Counter or Timer is a clock counter, use to count how many clocks into module after start.**
- **There are two kinds of generally called Timer or Counter. In fact, it's same.**
  - The Input Clock unknown : Just know how many counts, called Counter.
  - The Input Clock known : Can use count and clock 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.**

MCC's Timer Module

- **Add Timer resource from Device Resources.**



MCC's Timer Module



MICROCHIP

Timer Functions of MCC

- **MCC Provide below functions for Timer:**

  Prototype definition : "tmrn.h"

  - void TMRn_Initialize (void);
  - void TMRn_Start (void);
  - void TMRn_Stop (void);
  - void TMRn_WriteTimer (uint8_t timerVal);
  - uint8_t TMRn_ReadTimer (void);
  - void TMRn_LoadPeriodRegister (uint8_t periodVal);
  - void TMRn_SetInterruptHandler (void (* InterruptHandler)(void));

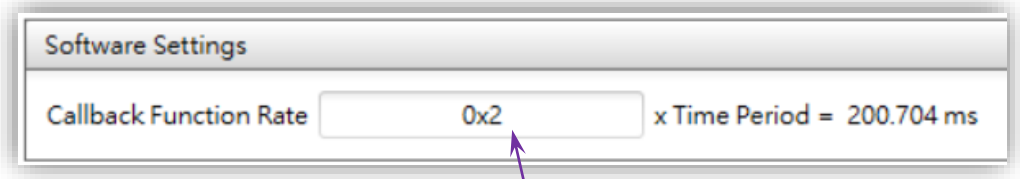
Function Rate

- **Timer's callback function rate is a software counter. Hardware always Limited. It can't set a long period.**
- **The "Rate" will involve software counter to extend your hardware timer period.**

```
void TMR2_ISR(void)
{
    static vaolatile unsigned int CountCallBack = 0;

    PIR1bits.TMR2IF = 0;

    if (++CountCallBack >= TMR2_INTERRUPT_TICKER_FACTOR)
    {
        TMR2_CallBack();
        CountCallBack = 0;
    }
}
```

```
void TMR2_CallBack(void)
{
    // Add your custom callback code here
    if(TMR2_InterruptHandler)
    {
        TMR2_InterruptHandler();
    }
}
```

```
void TMR2_SetInterruptHandler(void (* InterruptHandler)(void))
{
    TMR2_InterruptHandler = InterruptHandler;
}
```



extend your timer period by software artifice.

MICROCHIP

Lab4 Timer2 Interrupt

- Base on current project or Open .\Lab4.2_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 period is 200ms.

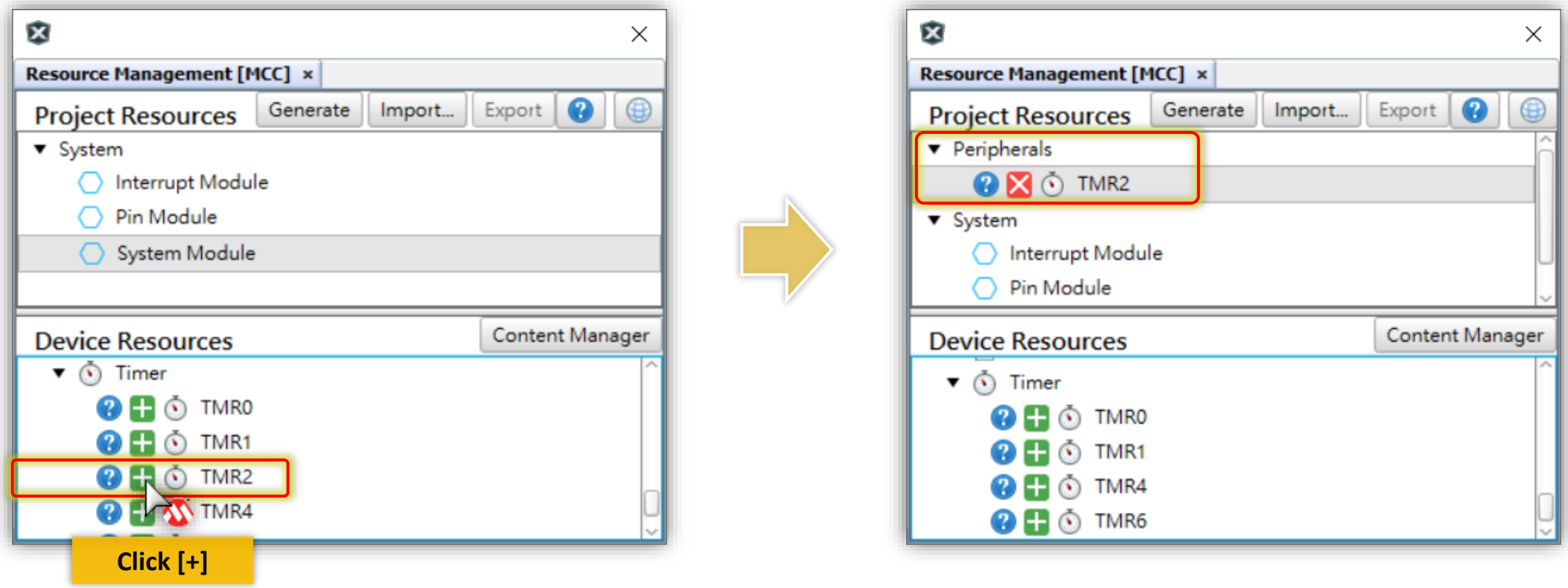
Let’s go!

MICROCHIP

Lab4 Timer2 Interrupt

Step 1

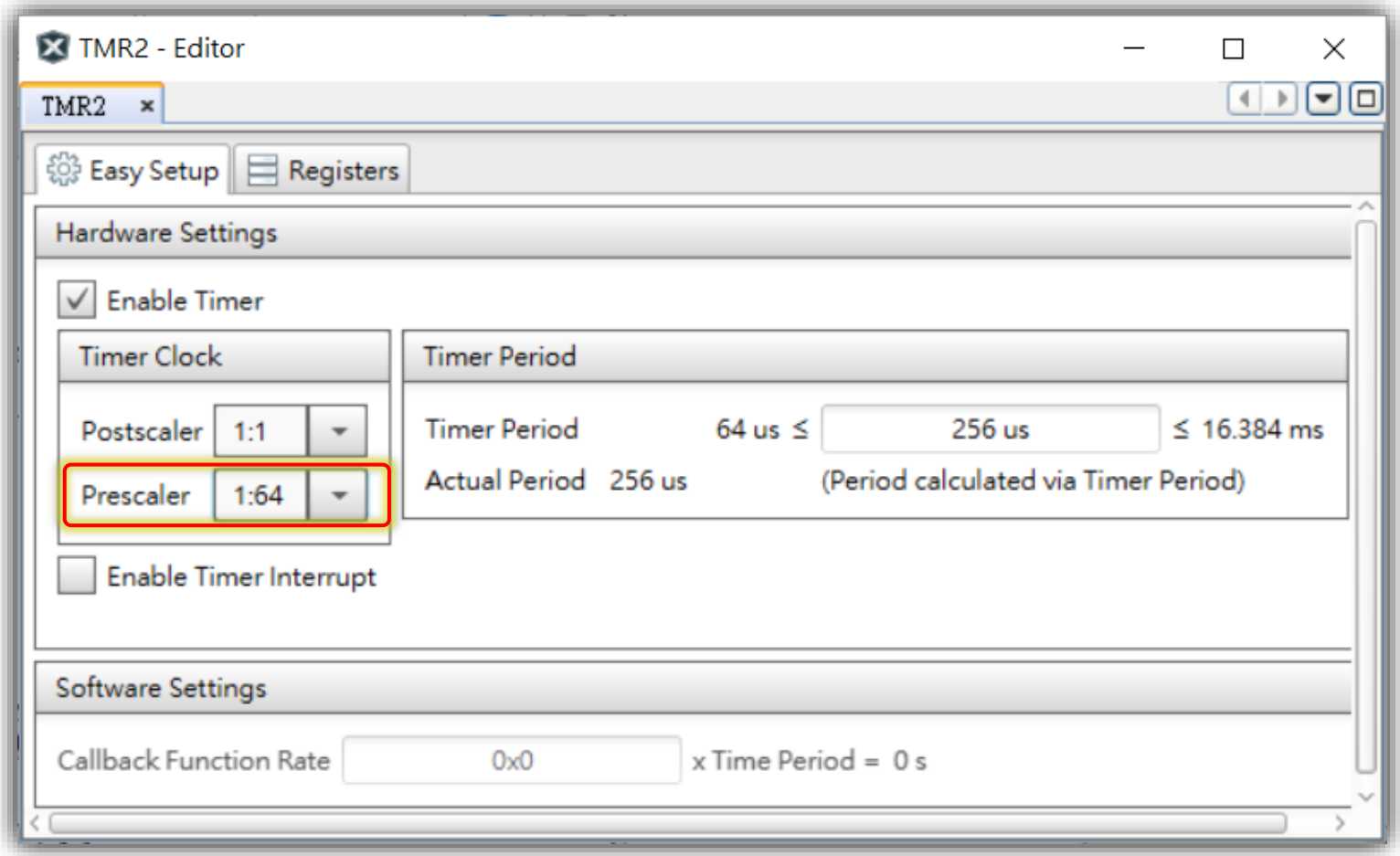
- **Device Resources**
  - **Timer ▸ TMR2**



Lab4 Timer2 Interrupt

Step 2

- **Prescaler**
  - **1:1 ▸ 1:64**

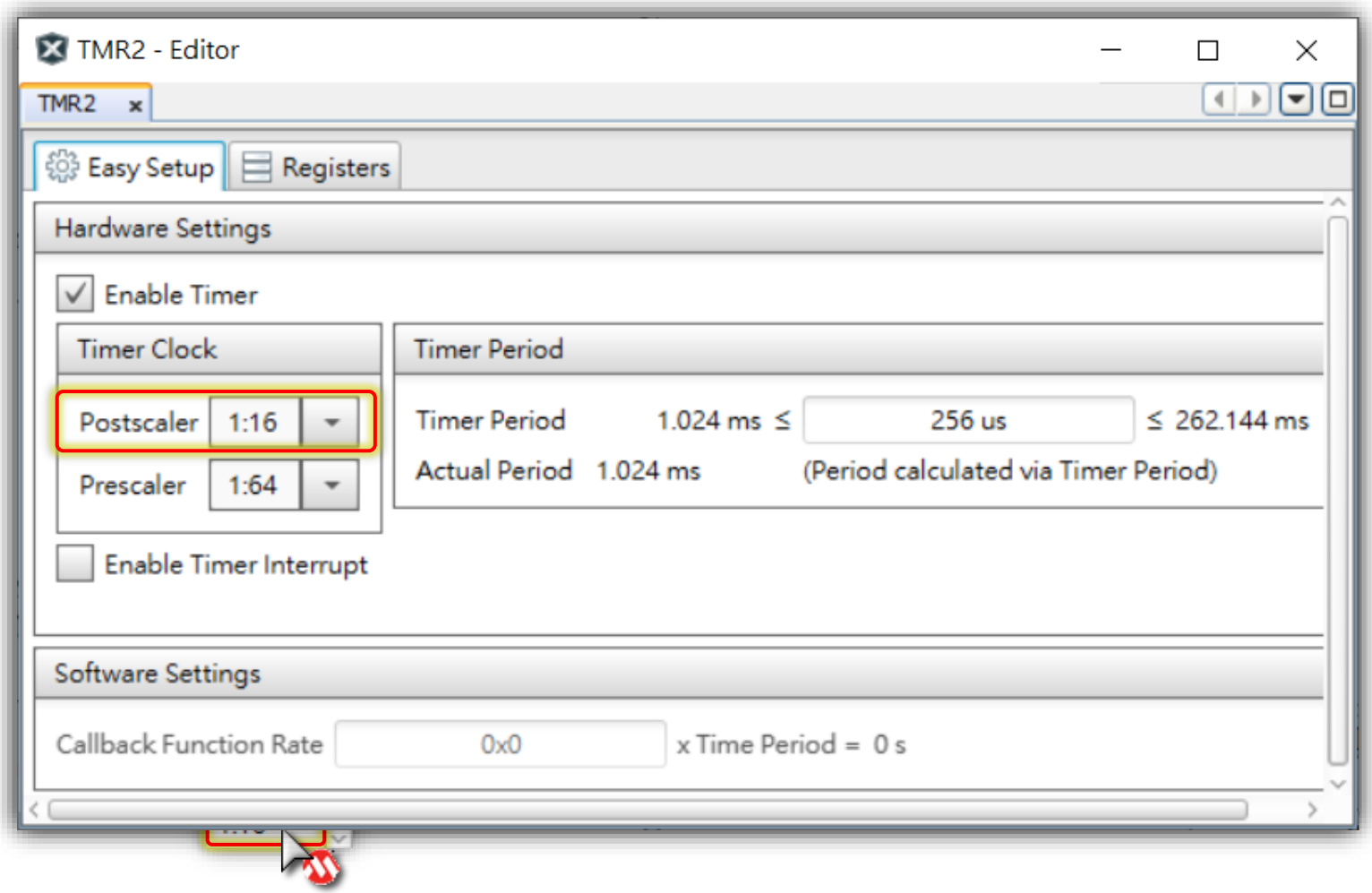


MICROCHIP

Lab4 Timer2 Interrupt

Step 3

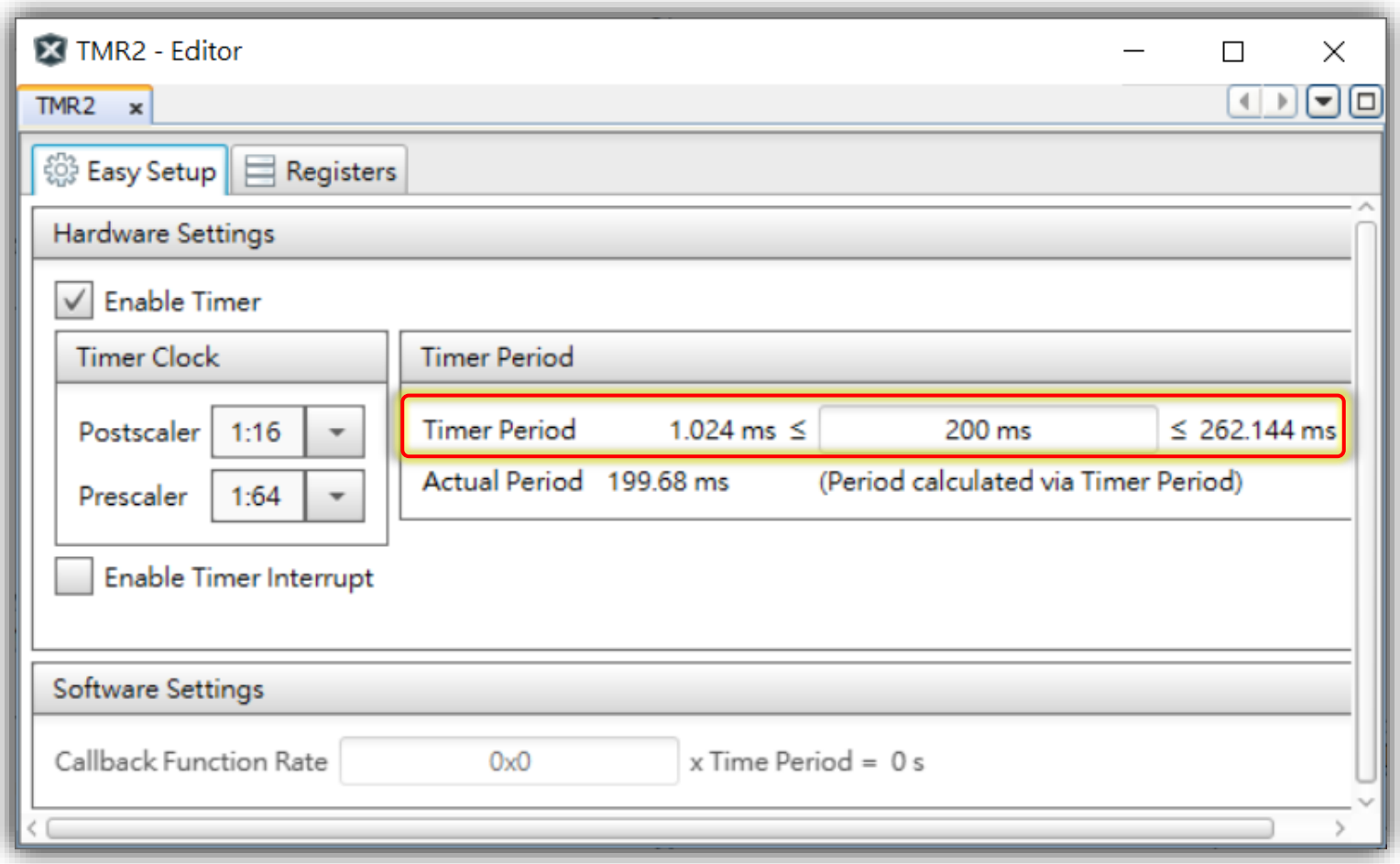
- **Postscaler**
  - **1:1 ▸ 1:16**



Lab4 Timer2 Interrupt

Step 4

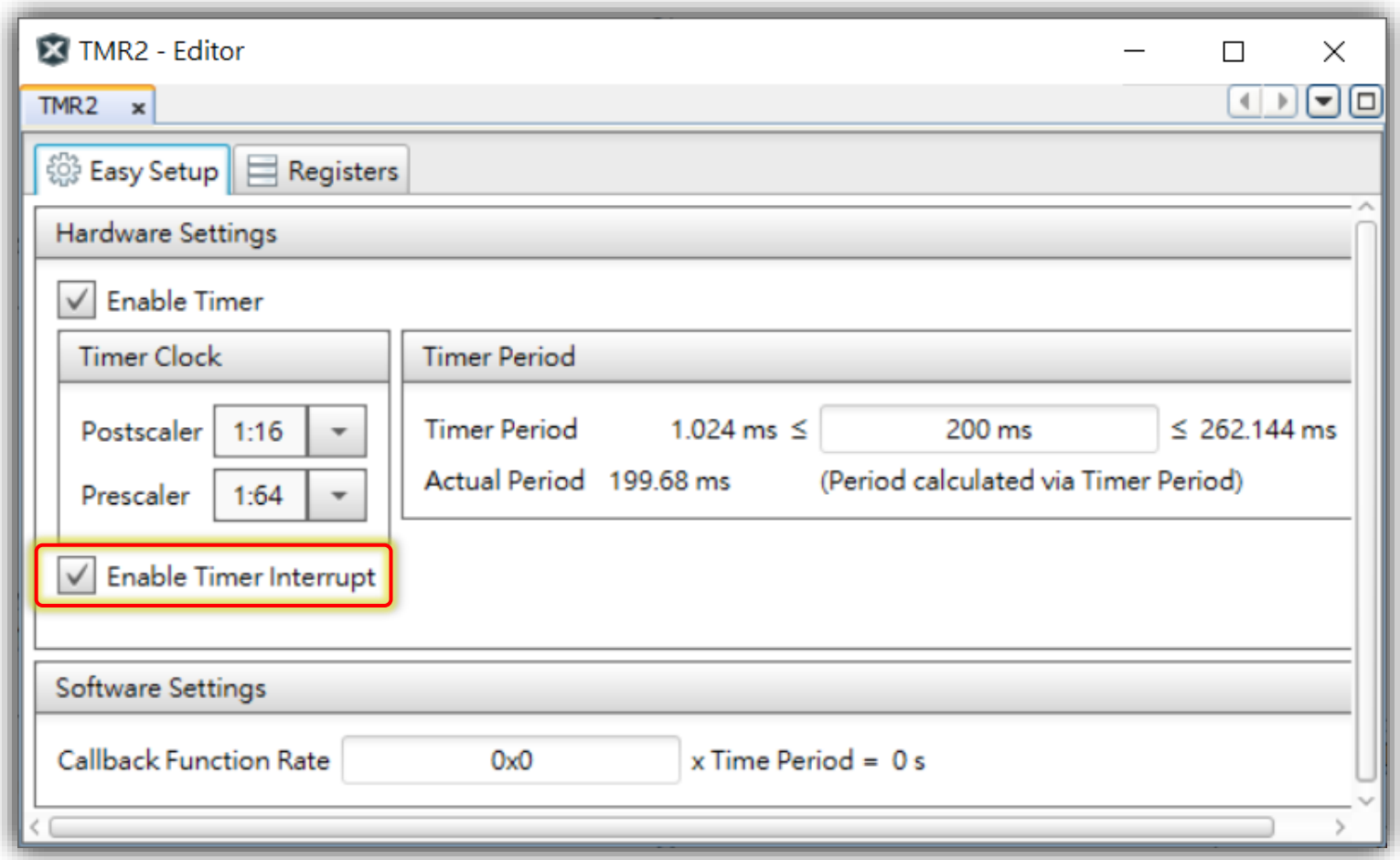
- **Timer Period**
  - **256us ▸ 200ms**



Lab4 Timer2 Interrupt

Step 5

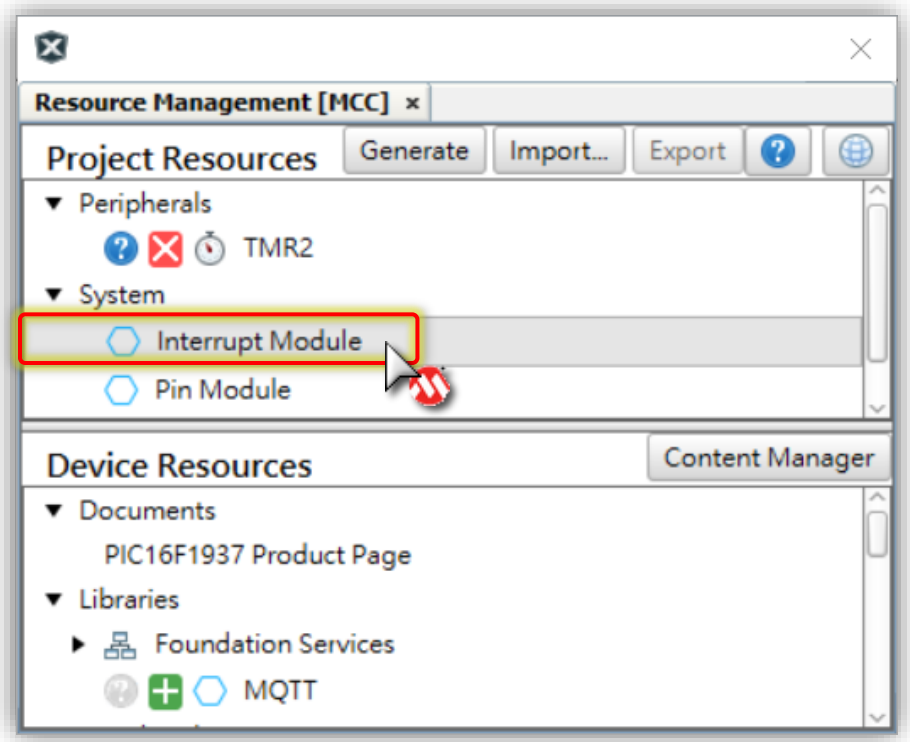
- **Enable Timer Interrupt.**

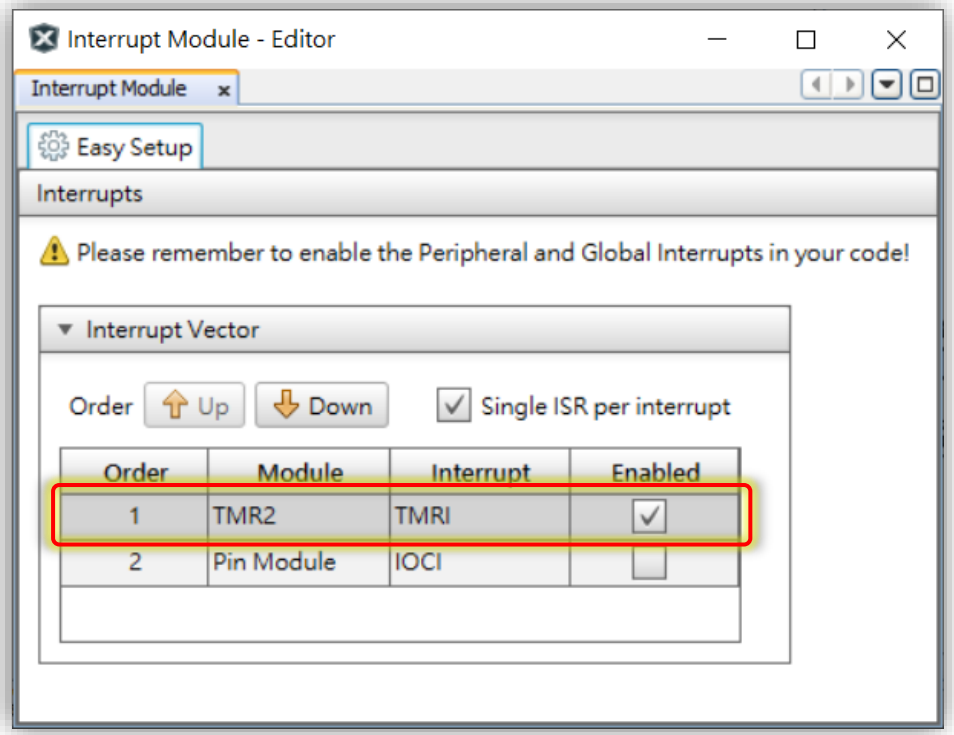


Lab4 Timer2 Interrupt

Step 6

- **Confirm enable TMR2 Interrupt.**

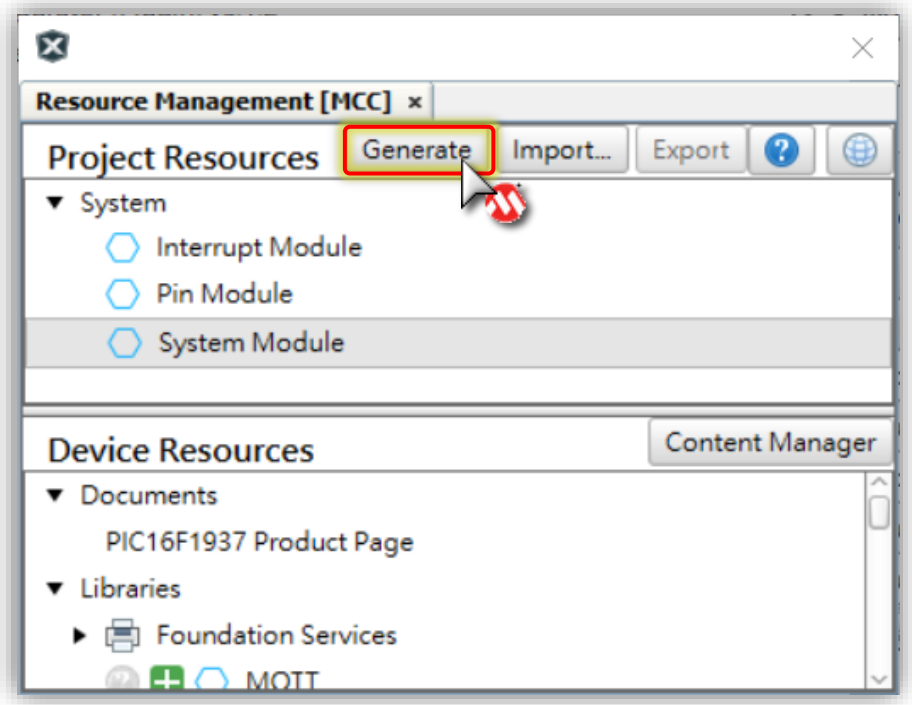


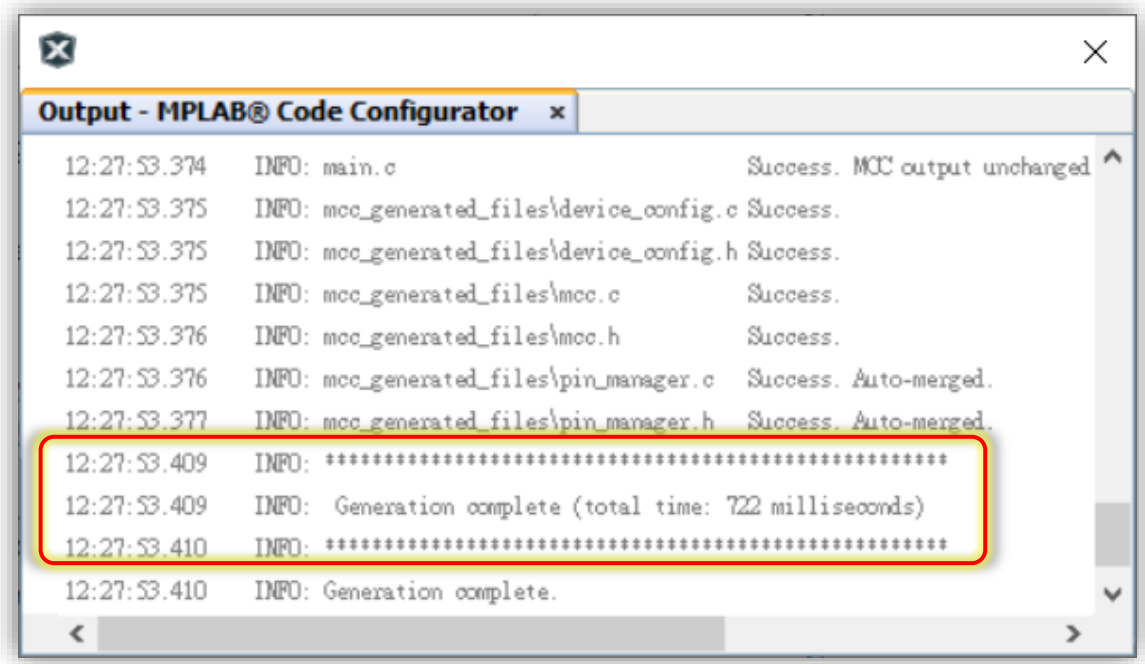


Lab4 Timer2 Interrupt

Step 7

- Click **Generate** Button to generate your code.





Lab4 Timer2 Interrupt

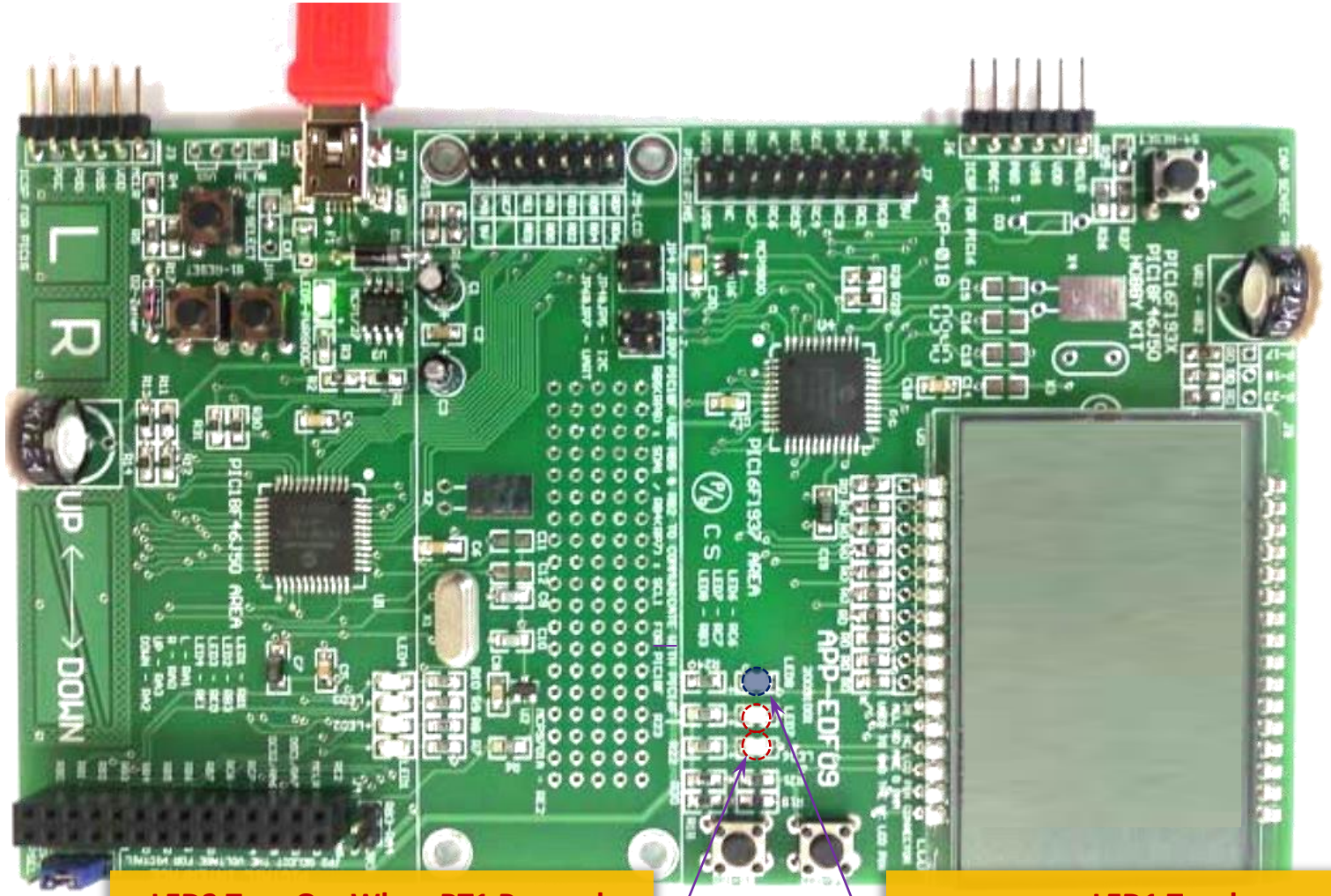
Code Example

```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/tmr2.h"

void TIM2_ISR(void);

uint32_t  i = 0;

int main(void)
{
    …

     // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();
    …
    //INTERRUPT_PeripheralInterruptDisable();

    TMR2_SetInterruptHandler(TIM2_ISR);
    while (1)
    {
        …
    }
}

void TIM2_ISR(void)
{
    LED1_Toggle();
}
```

```
int main(void)
{
    …
    //INTERRUPT_PeripheralInterruptDisable();

    TMR2_SetInterruptHandler(TIM2_ISR);
    while (1)
    {
        // Add your application code
        LED1_Toggle();
        for (i = 0; i < 2800; i++);

        if(!BT1_GetValue())
        {
            LED2_SetLow();
        }
        else if(!BT2_GetValue())
        {
            LED3_SetLow();
        }
        else
        {
            LED2_SetHigh();
            LED3_SetHigh();
        }
    }
}
```

Lab4 Timer2 Interrupt

Result



LED2 Turn On, When BT1 Pressed.
LED3 Turn On, When BT2 Pressed.

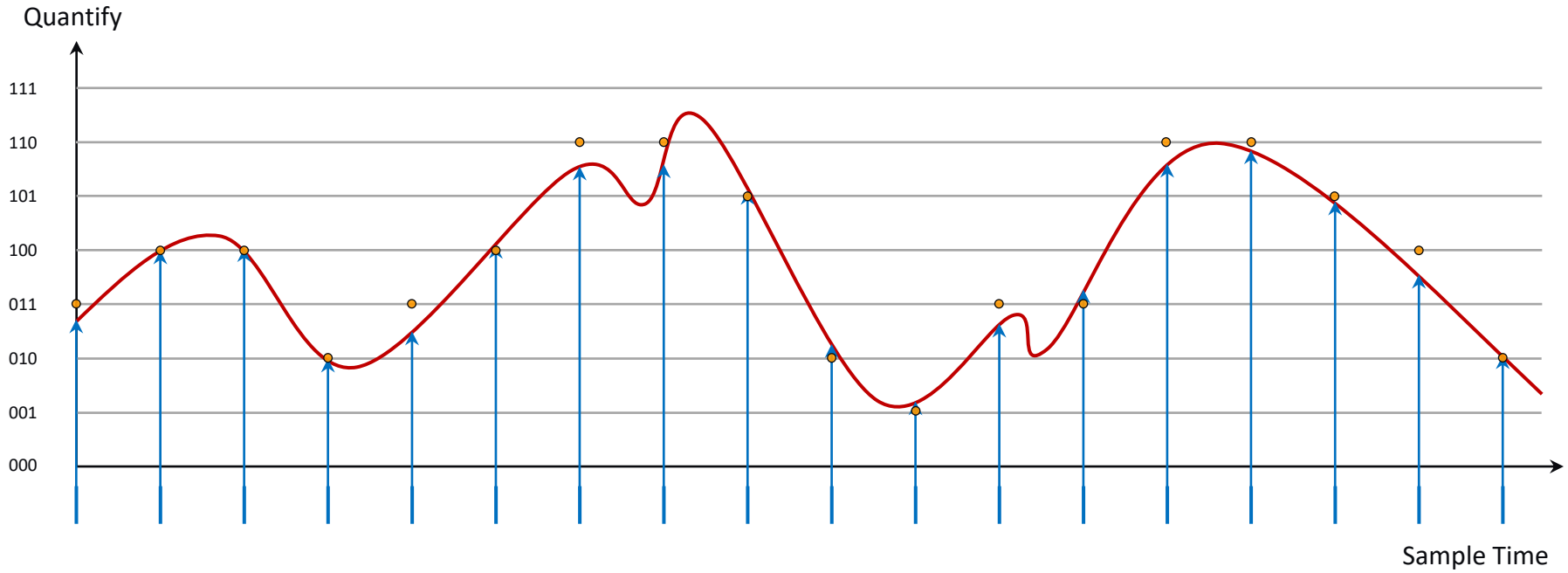
LED1 Toggle,
Interval Period use "Timer2" callback.

MICROCHIP

ADC Architecture

What’s ADC ?

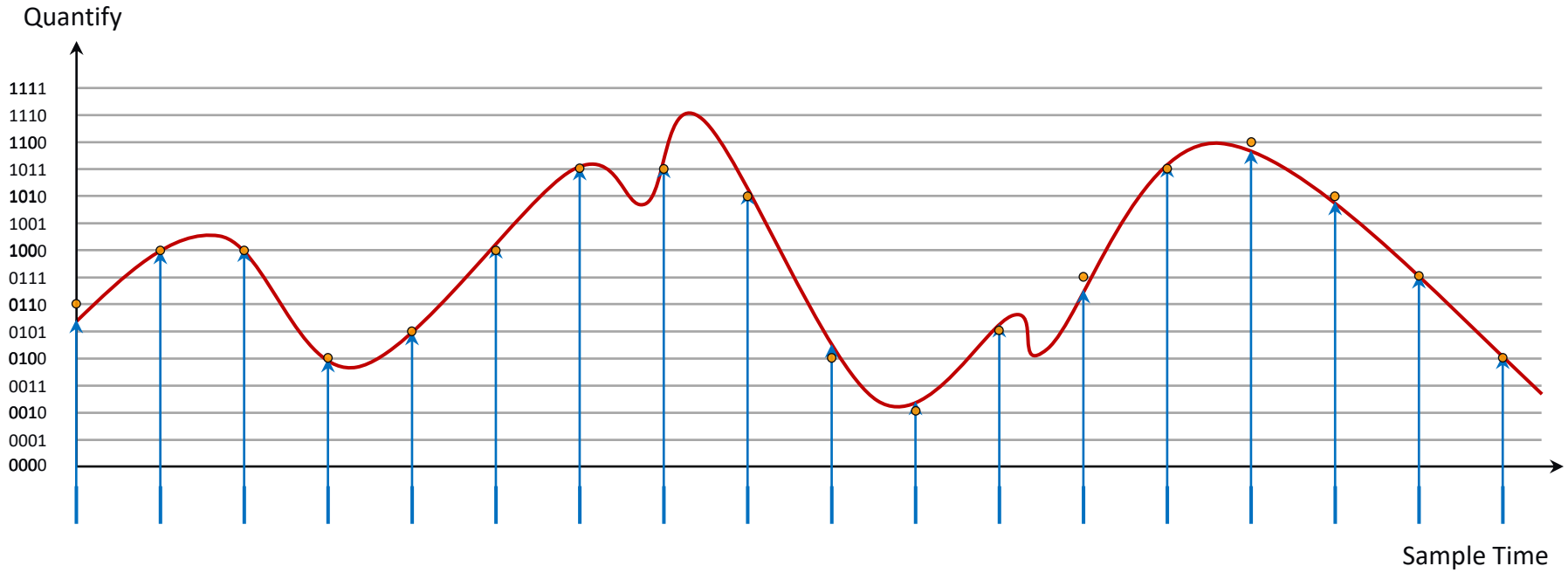
- **The Analog-to-Digital Converter(ADC) converts a signal from the time/amplitude continuous domain into a time/amplitude discrete domain.**



MICROCHIP

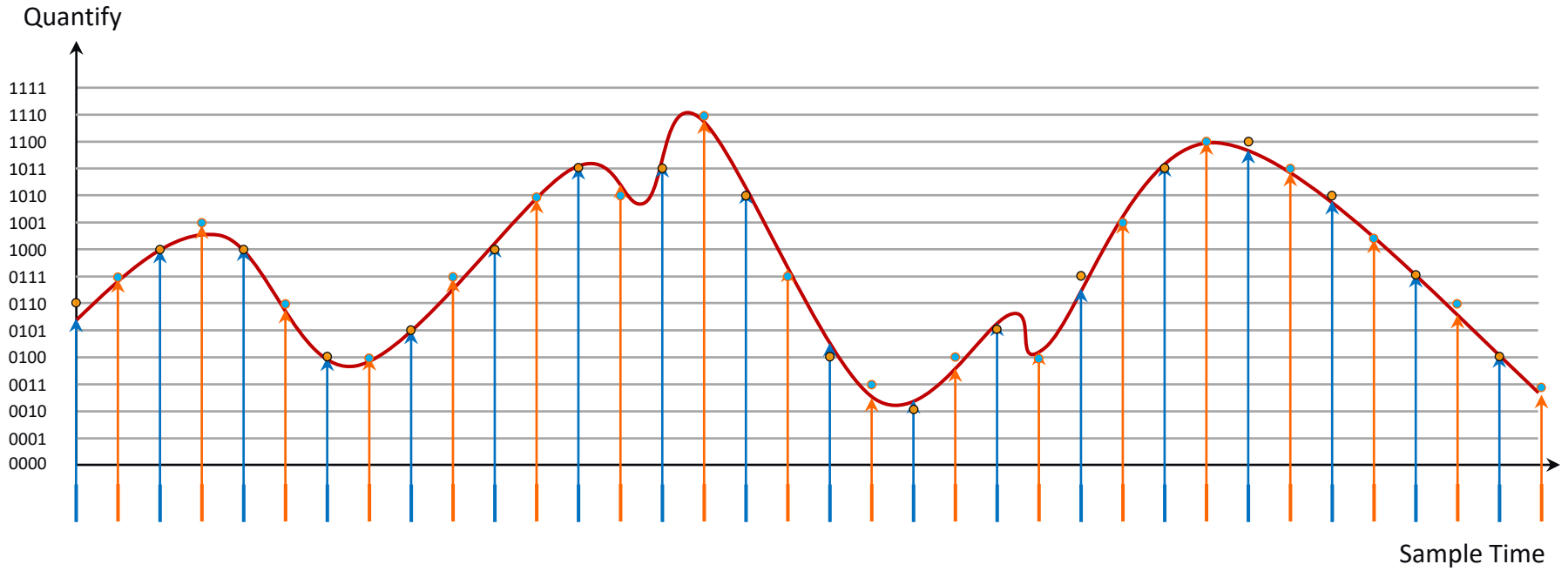
Resolution

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



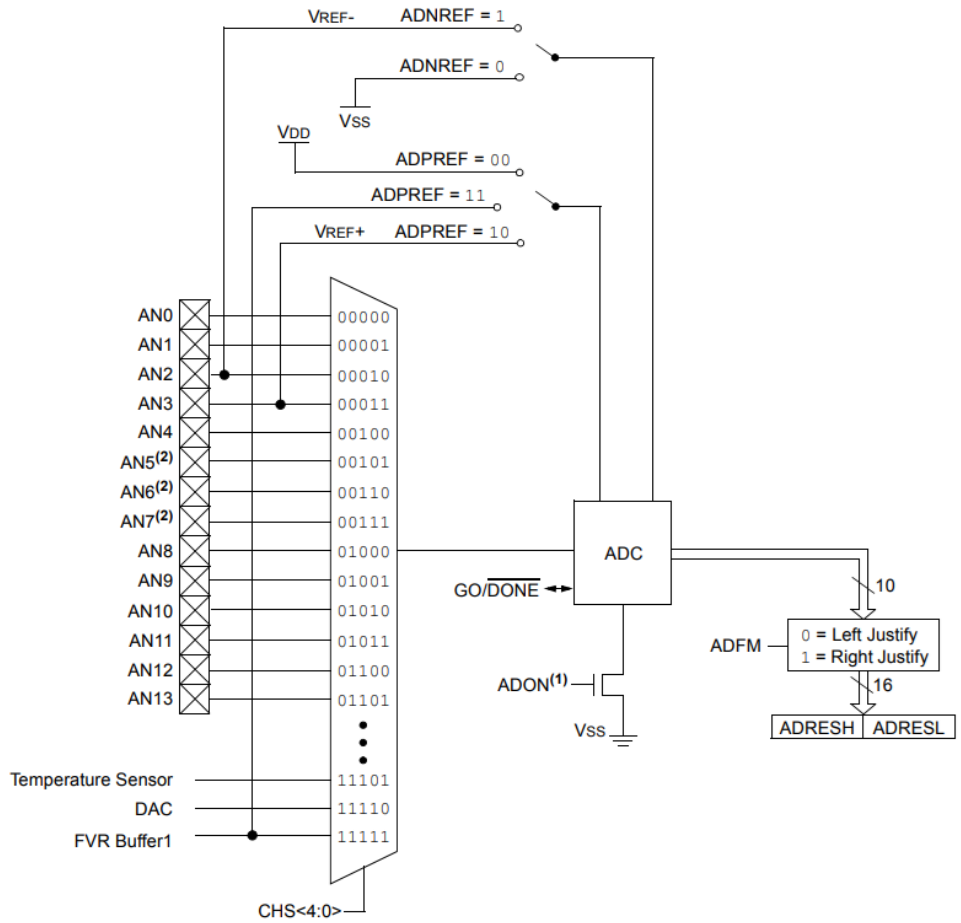
Sample Rate

- 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)



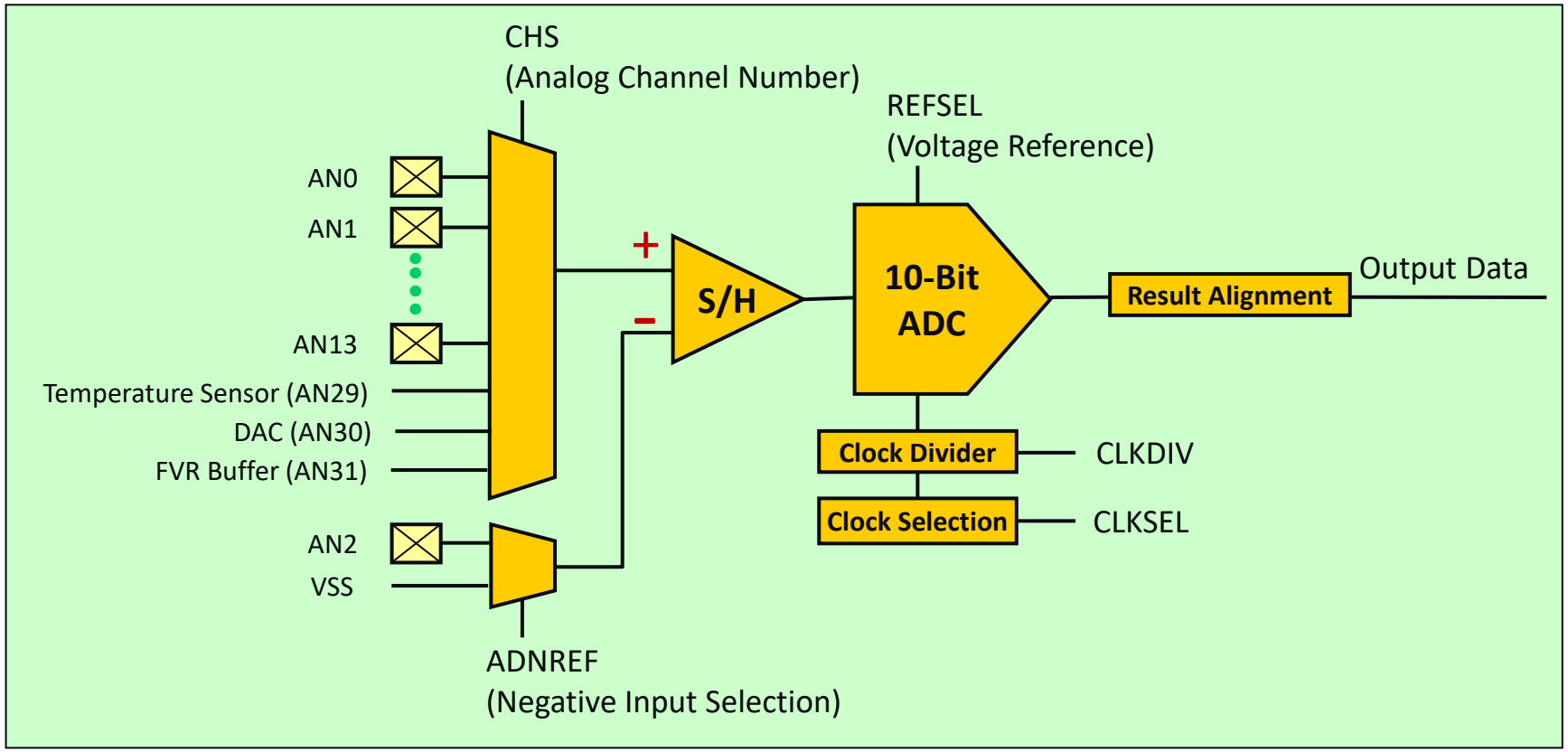
PIC16F Series ADC Features

- **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 Channel Selection

- 14 analog channel inputs to connect as Analog Positive input of S/H.
- AN2 or VSS as Analog Negative input of the S/H.
- The differential input voltage is Positive – Negative.



Sample and Conversion Sequence

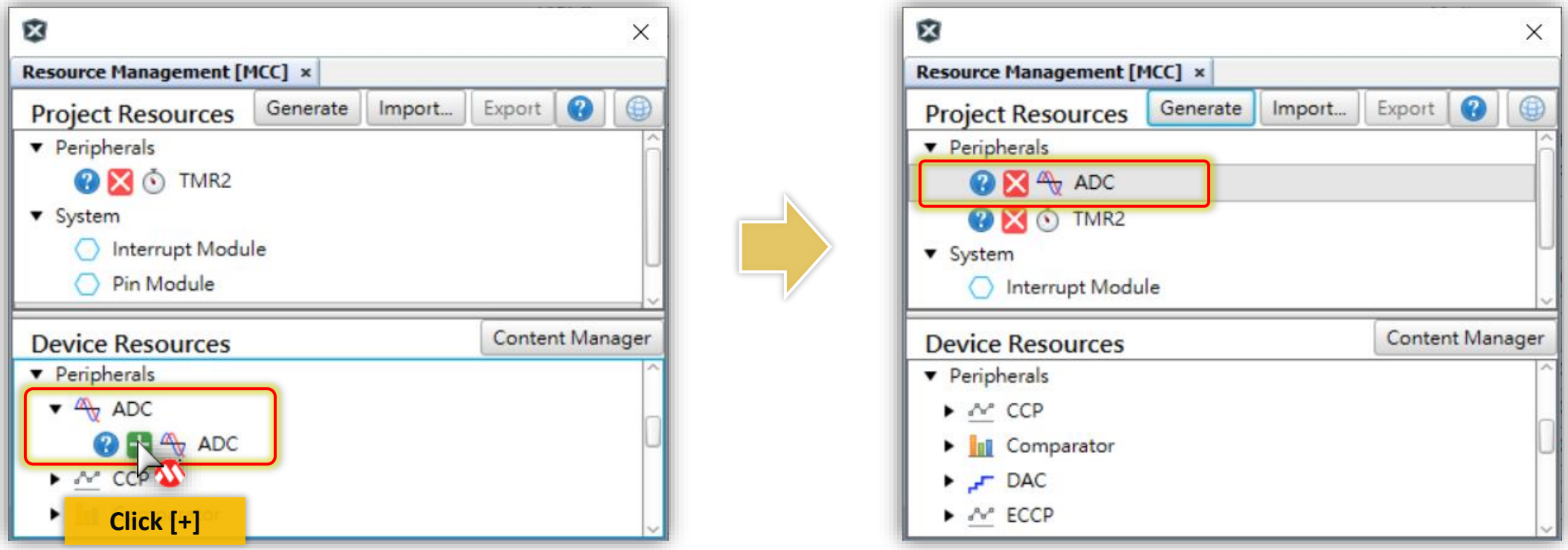
- The ADC sample and conversion time are requirements from the specification.
- The conversion time need 11 TAD, TAD time range 1us to 6us.

ADC Clock Period (TAD)		Device Frequency (Fosc) Device Frequency (Fosc)					
ADC Clock Source	ADCS<2:0>	32 MHz	20 MHz	16 MHz	8 MHz	4 MHz	1 MHz
Fosc/2	000	62.5ns(2)	100 ns(2)	125 ns(2)	250 ns(2)	500 ns(2)	2.0 µs
Fosc/4	100	125 ns(2)	200 ns(2)	250 ns(2)	500 ns(2)	1.0 µs	4.0 µs
Fosc/8	001	0.5 µs(2)	400 ns(2)	0.5 µs(2)	1.0 µs	2.0 µs	8.0 µs(3)
Fosc/16	101	800 ns	800 ns	1.0 µs	2.0 µs	4.0 µs	16.0 µs(3)
Fosc/32	010	1.0 µs	1.6 µs	2.0 µs	4.0 µs	8.0 µs(3)	32.0 µs(3)
Fosc/64	110	2.0 µs	3.2 µs	4.0 µs	8.0 µs(3)	16.0 µs(3)	64.0 µs(3)
FRC	x11	1.0-6.0 µs(1,4)	1.0-6.0 µs(1,4)	1.0-6.0 µs(1,4)	1.0-6.0 µs(1,4)	1.0-6.0 µs(1,4)	1.0-6.0 µs(1,4)

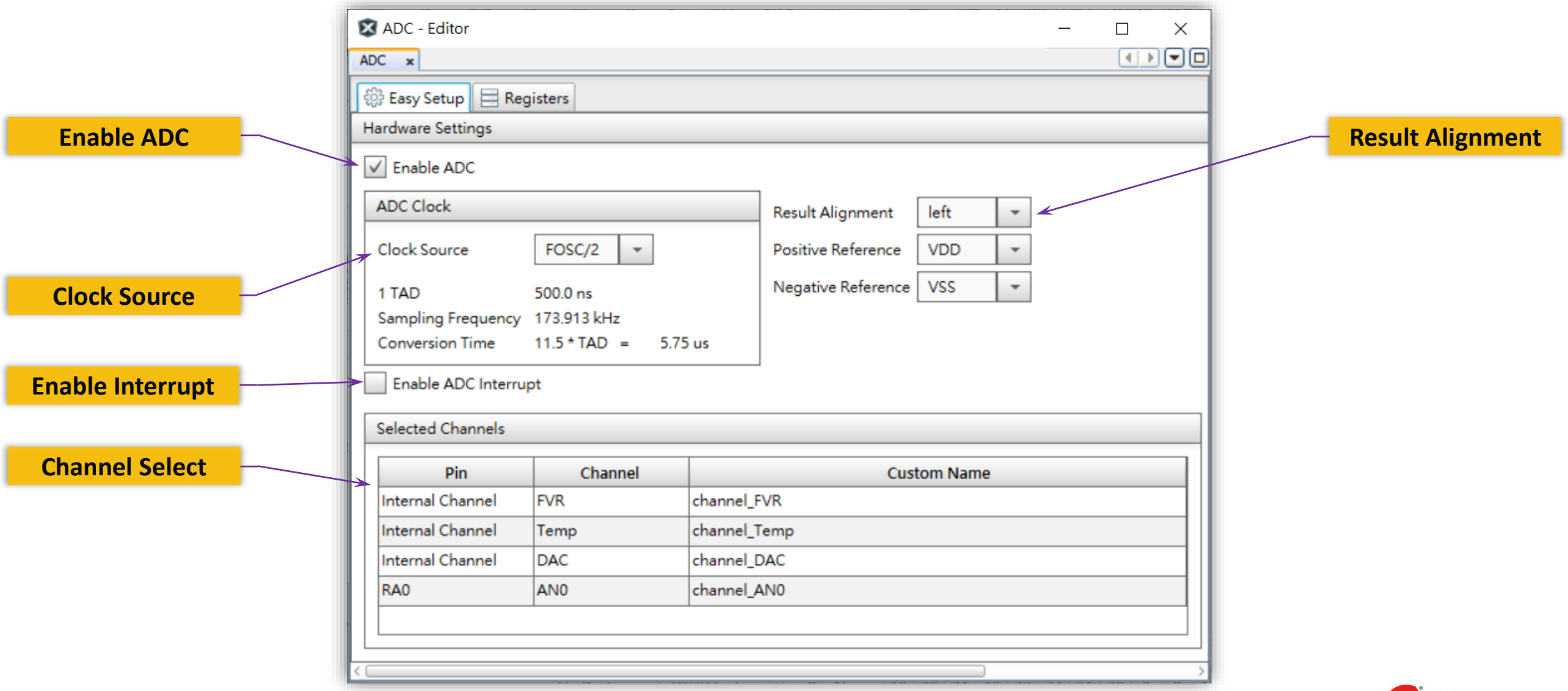
MICROCHIP

MCC's ADC Module

- **Add ADC resource from Device Resources.**



MCC's ADC Module



MICROCHIP

ADC Functions of MCC

- **MCC Provide below functions for ADC:**

  Prototype definition : “adc.h"

  - void ADC_Initialize(void);
  - void ADC_StartConversion(void);
  - void ADC_SelectChannel(adc_channel_t channel);
  - adc_result_t ADC_GetConversion(adc_channel_t channel);
  - adc_result_t ADC_GetConversionResult(void);
  - bool ADC_IsConversionDone(void);

Lab5 ADC Channel Periodically

- Base on current project or Open .\Lab5_xxx.X to do exercises
- Try to add ADC module and conversion Potentiometer (VR2) voltage.
  - ADC Result ≥ 512, turn on LED2.
  - ADC Result < 512, turn off LED2.
- " Clock Source " ► " Fosc/4 "
- " Result Alignment " ► " Right "
- " Channel Select " ► " AN8(RB2) "

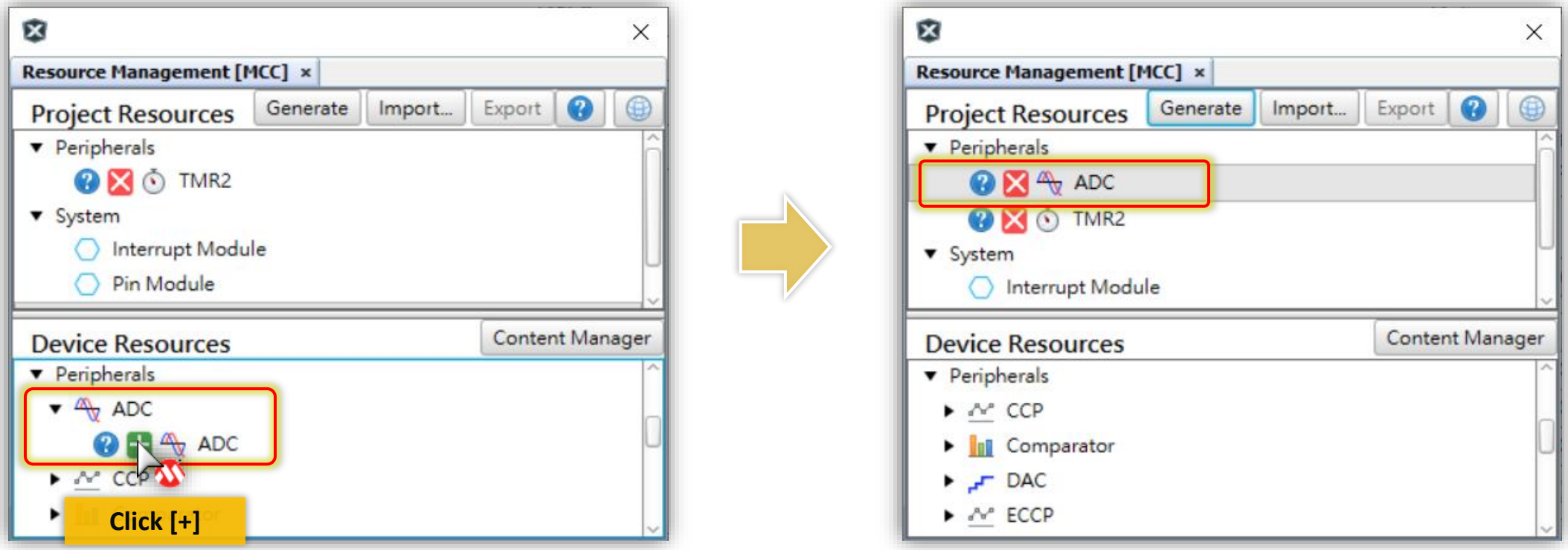
Let’s go!

MICROCHIP

Lab5 ADC Channel Periodically

Step 1

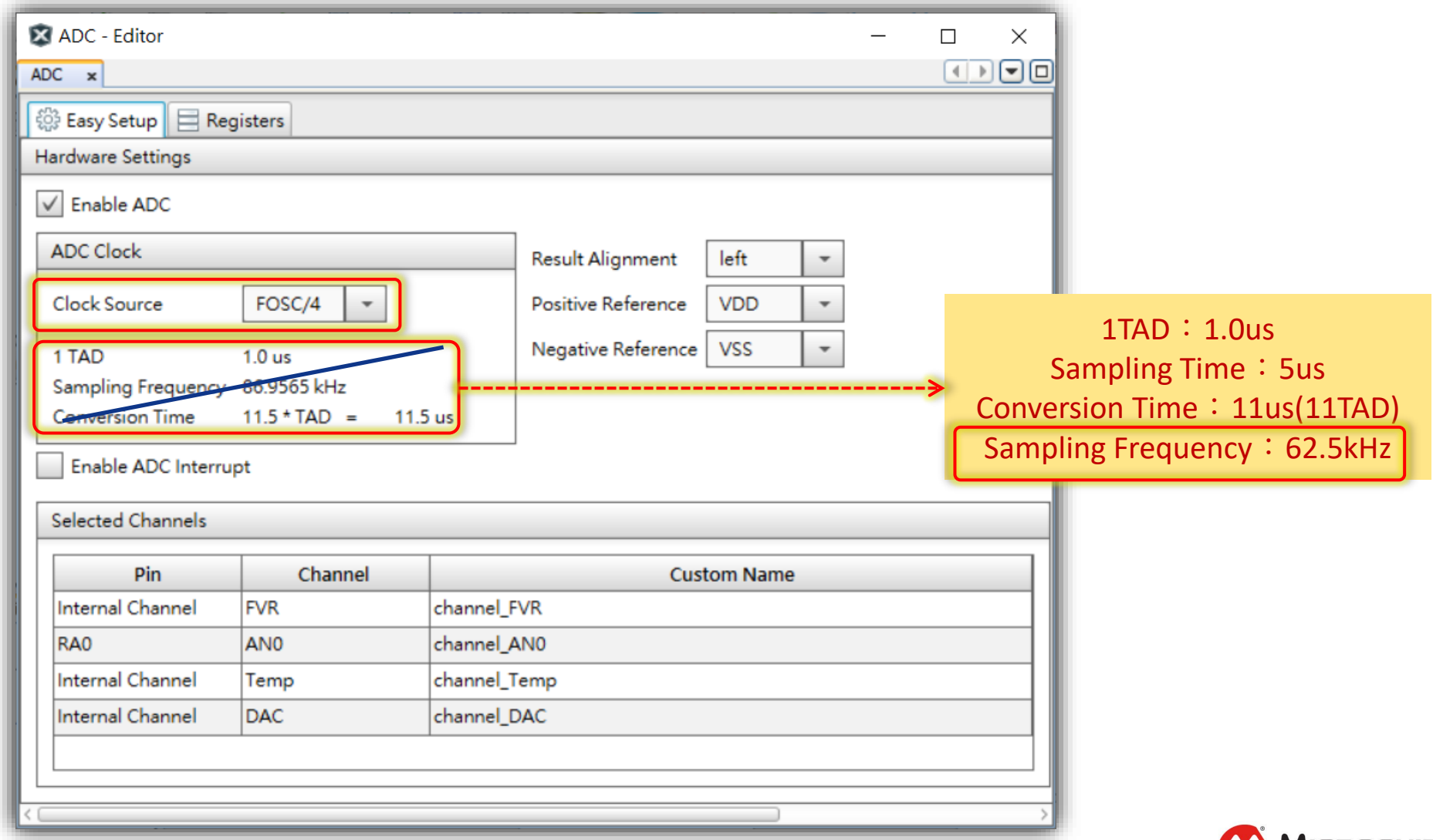
- **Device Resources**
  - **ADC ▸ ADC**



Lab5 ADC Channel Periodically

Step 2

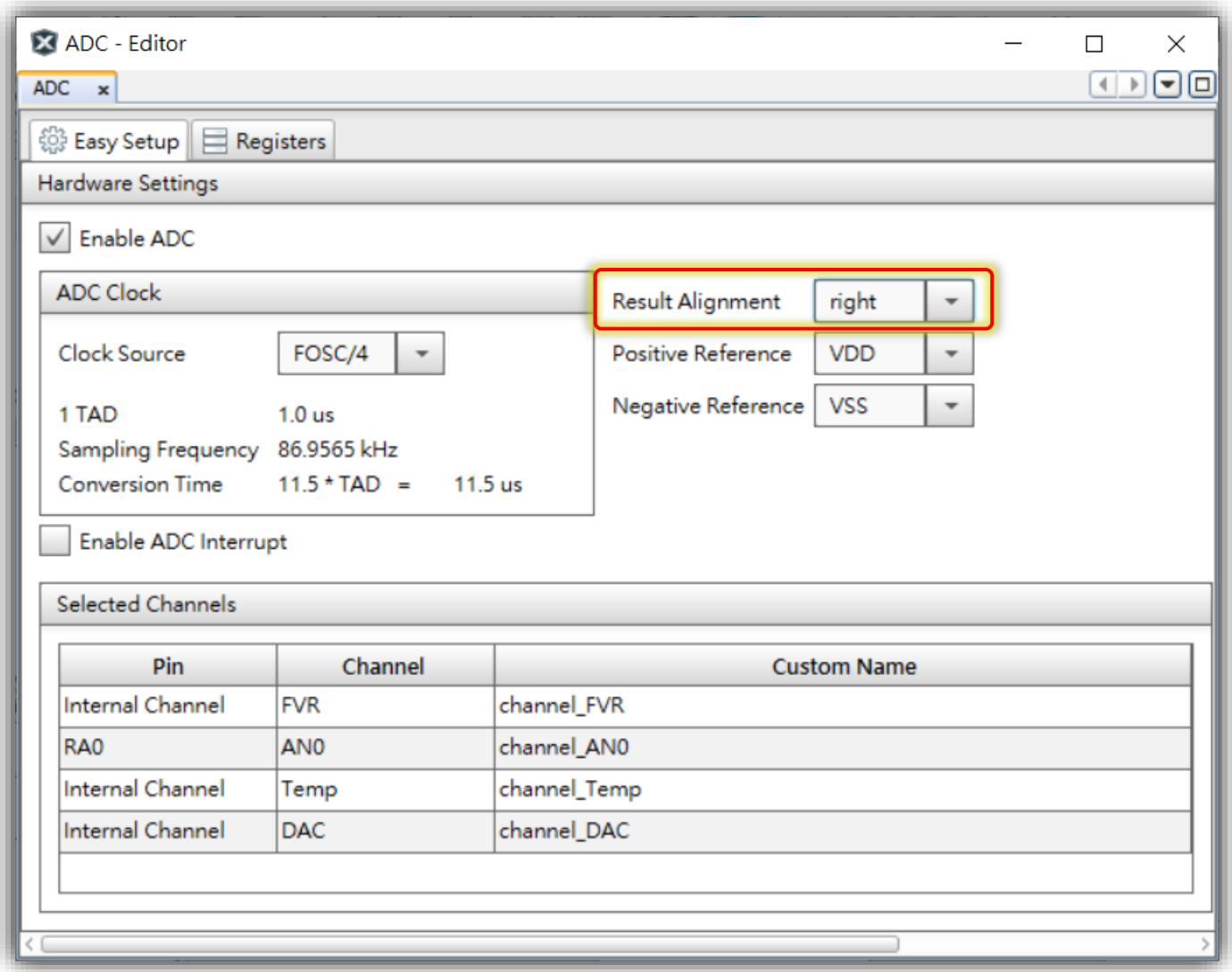
- **Clock Source**
  - **Fosc/2 ▸ Fosc/4**



Lab5 ADC Channel Periodically

Step 3

- **Result alignment**
  - **Left ▸ Right**

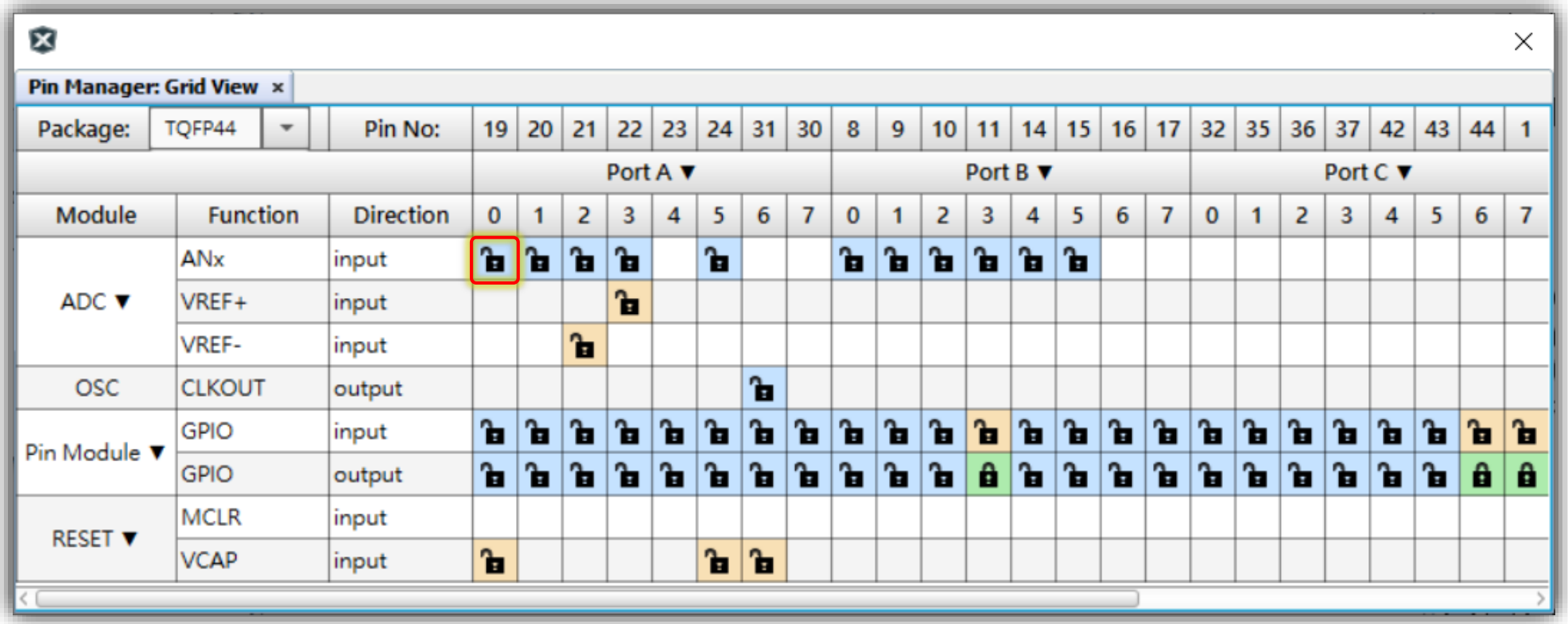


MICROCHIP

Lab5 ADC Channel Periodically

Step 4

- **Pin Manager : Grid View**
  - **AN0(RA0) ▸ disable**

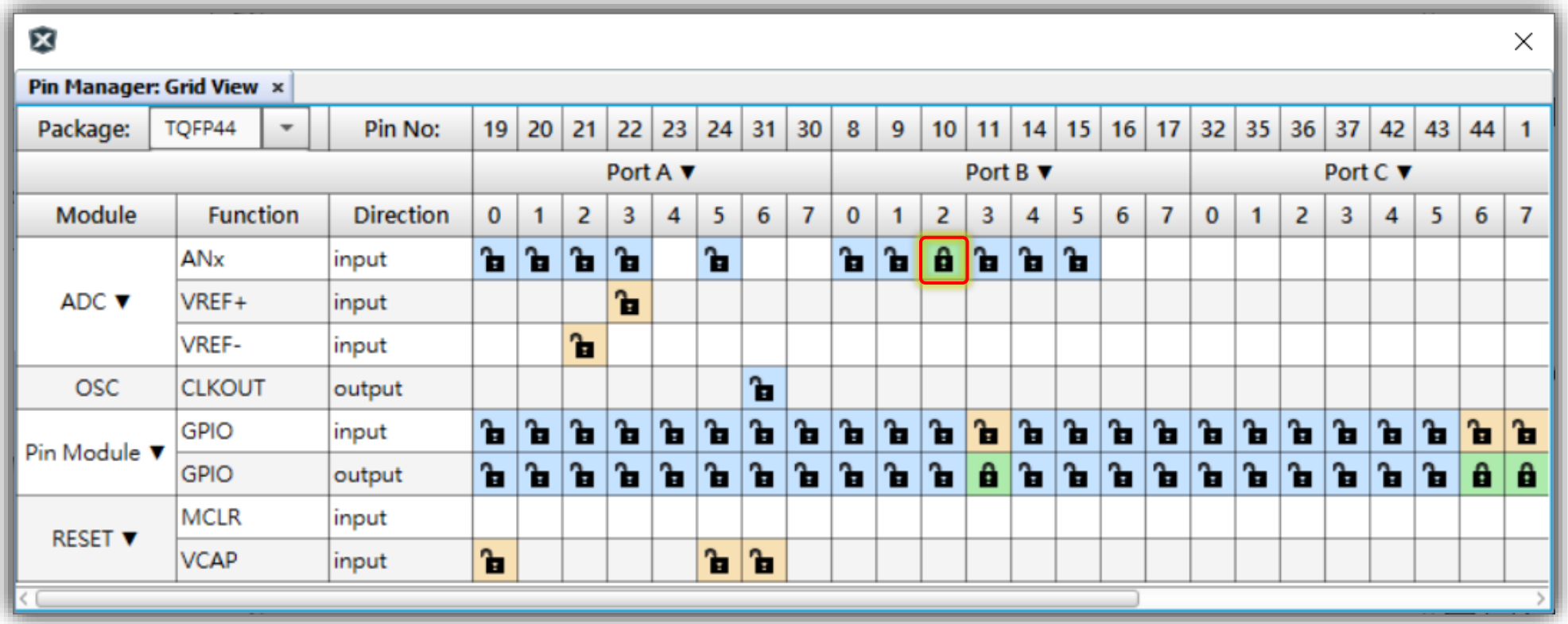


MICROCHIP

Lab5 ADC Channel Periodically

Step 5

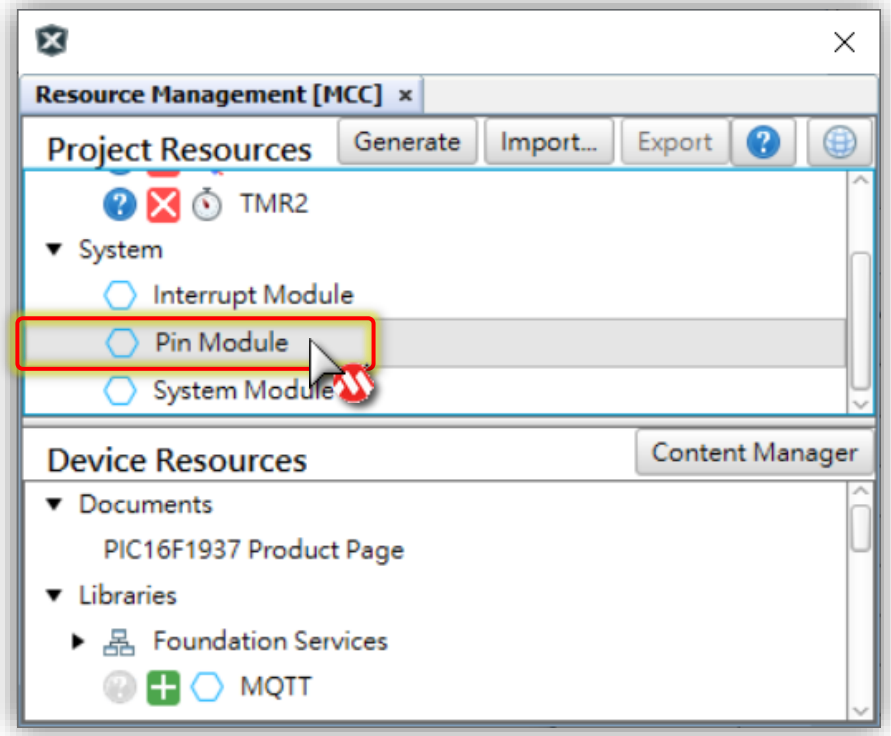
- **Pin Manager : Grid View**
  - **AN8(RB2) ▸ enable**

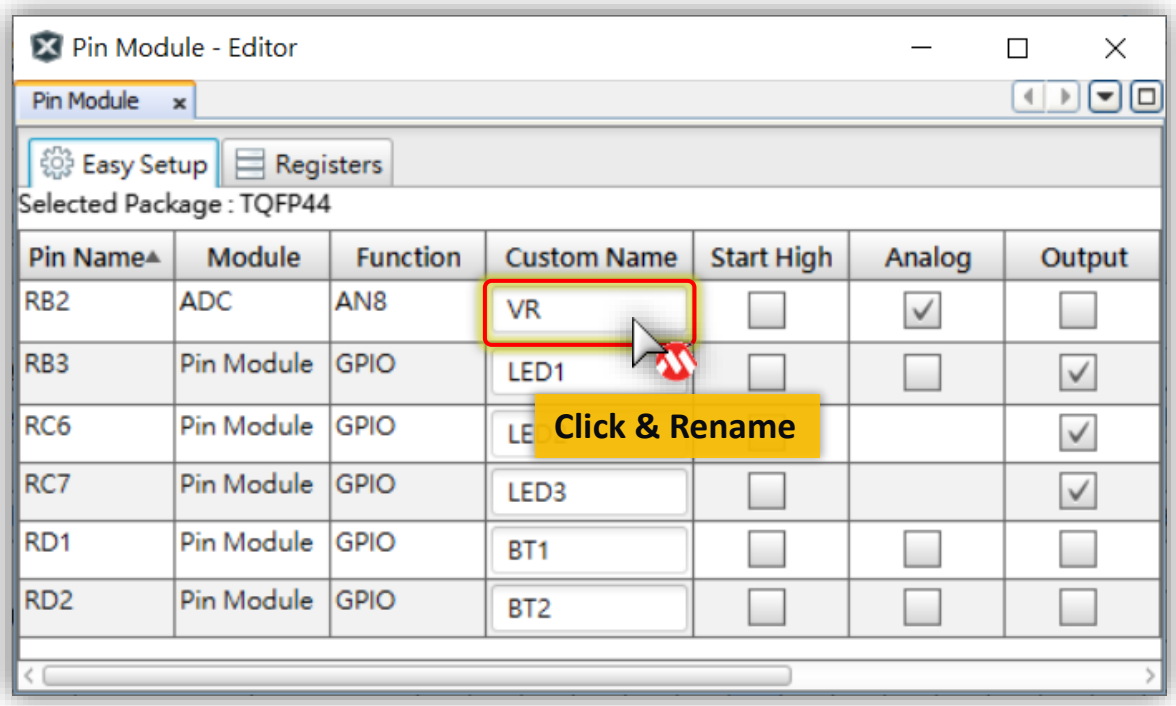


Lab5 ADC Channel Periodically

Step 6

- **Set Alias for RB2.**
  - **Pin Module : Custom Name ▸ VR.**

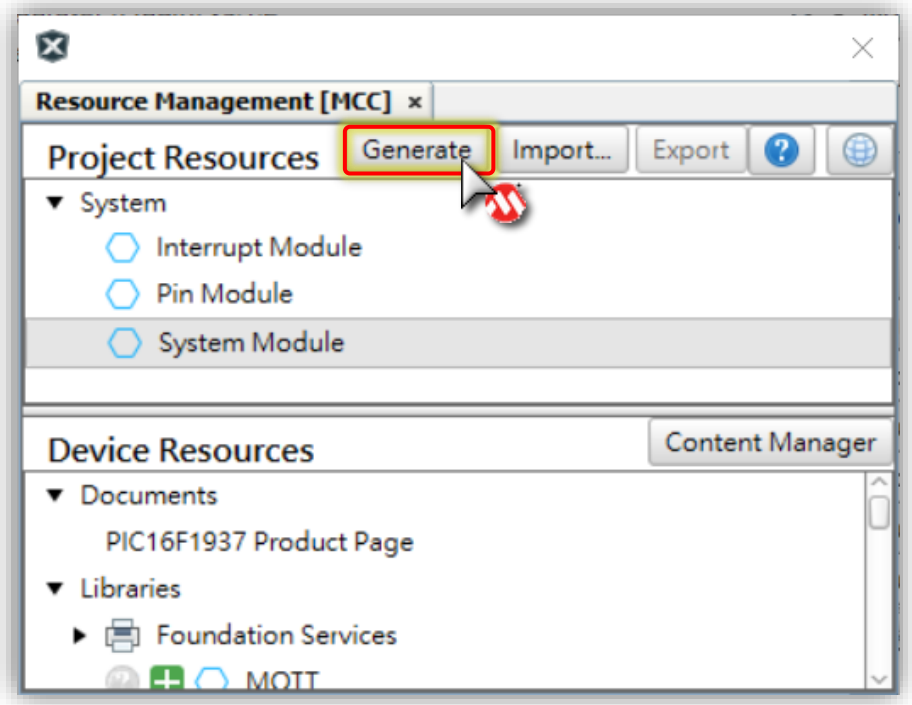


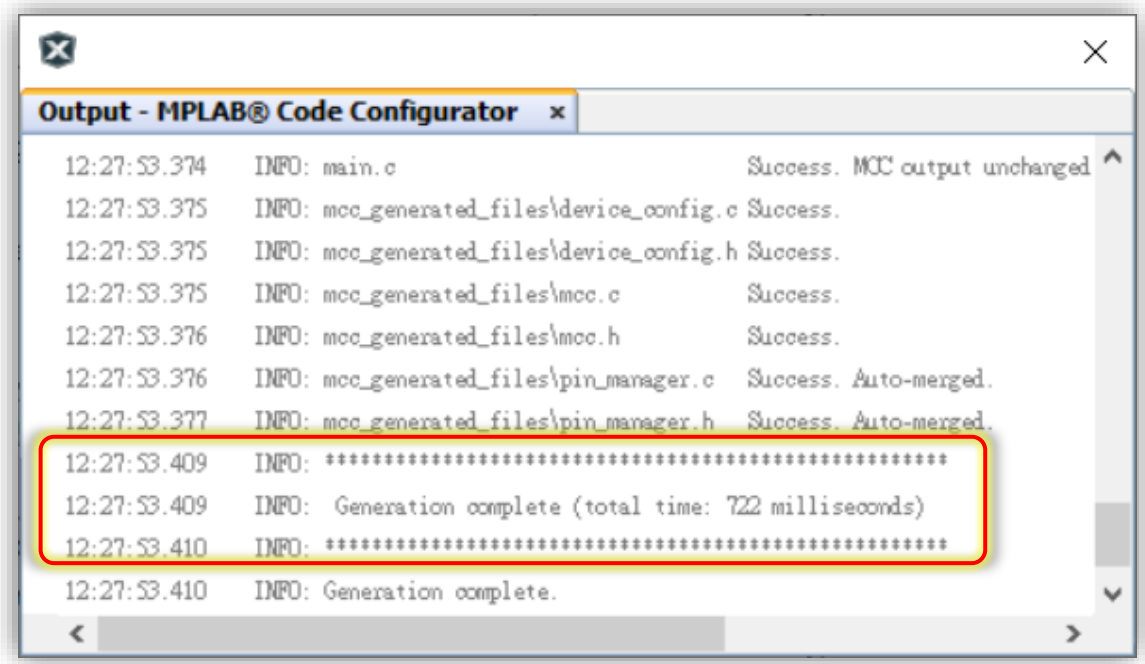


Lab5 ADC Channel Periodically

Step 7

- Click **Generate** Button to generate your code.





Lab5 ADC Channel Periodically

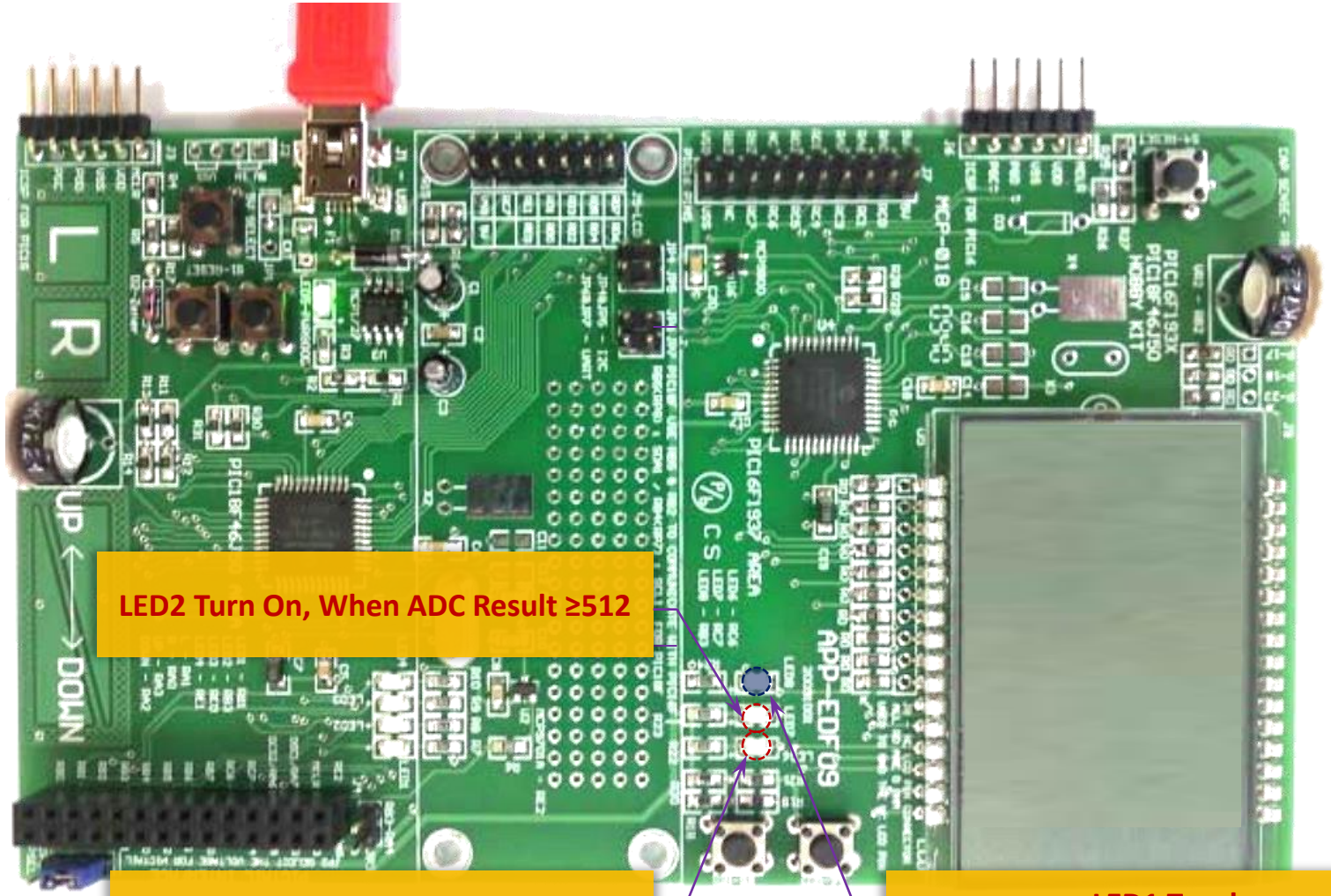
Code Example

```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/tmr2.h"

void TIM2_ISR(void);

uint32_t  i = 0;

uint16_t ADC_Value=0;

int main(void)
{
    ...
    while (1)
    {
        ...
    }
}

void TIM2_ISR(void)
{
    LED1_Toggle();
    ADC_Value = ADC_GetConversion(VR);
}
```

```
int main(void)
{
    ...
    TMR2_SetInterruptHandler(TIM2_ISR);
    while (1)
    {
        // Add your application code

        if(ADC_Value>=512)
        {
            LED2_SetLow();
        }
        else
        {
            LED2_SetHigh();
        }
        if(!BT2_GetValue())
        {
            LED3_SetLow();
        }
        else
        {
            LED3_SetHigh();
        }
    }
}
```

Lab5 ADC Channel Periodically

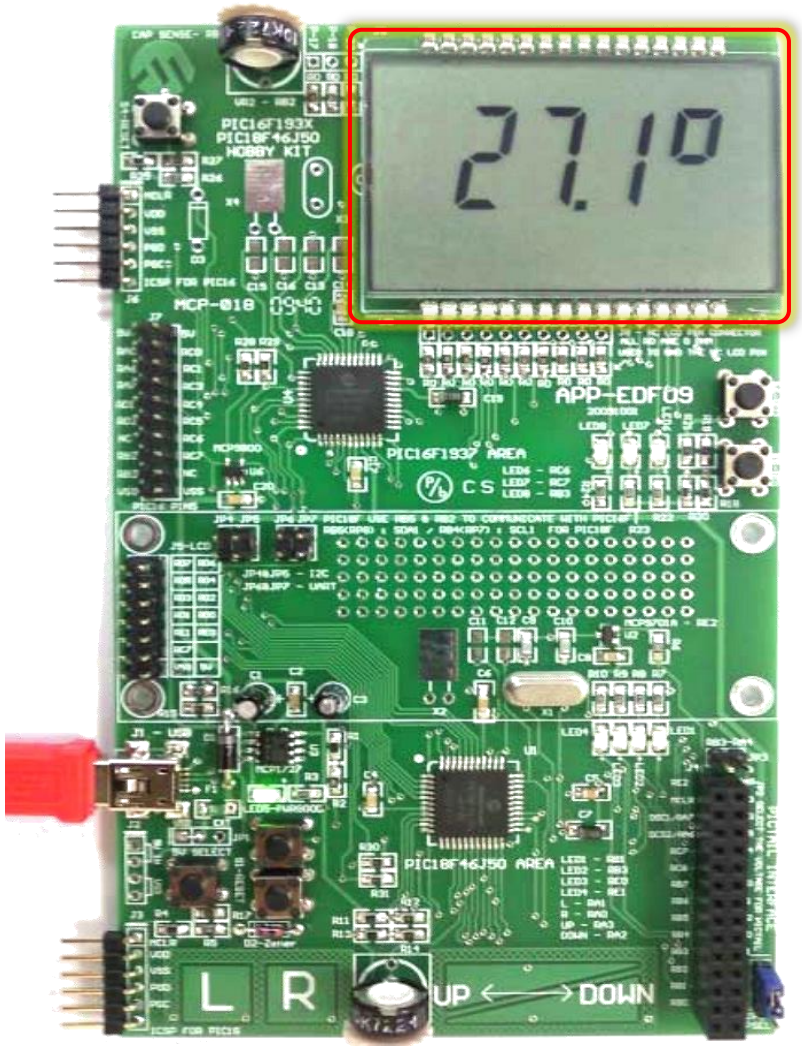
Result



LCD Architecture

PIC16F Series LCD Features

- **Built-in LCD driver module can directly drive LCD panel.**
- **Up to four common pins.**
  - Static (1 common)
  - 1/2 multiplex (2 common)
  - 1/3 multiplex (3 common)
  - 1/4 multiplex (4 common)
- **Segment pins up to.**
  - 28 pins - up to 60 segments.
  - 44 pins - up to 96 segments.
  - 64 pins - up to 184 segments.
  - 80 pins - up to 192 segments.
- **Three LCD clock sources with selectable presale.**
- **Built-in LCD bias generator.**
- **Support 3V and 5V glass panel.**
- **Use software adjusted contrast.**
- **Low power consumption.**



MICROCHIP

LCD Driver Module

- **LCD physical pin total= Common x Segments**

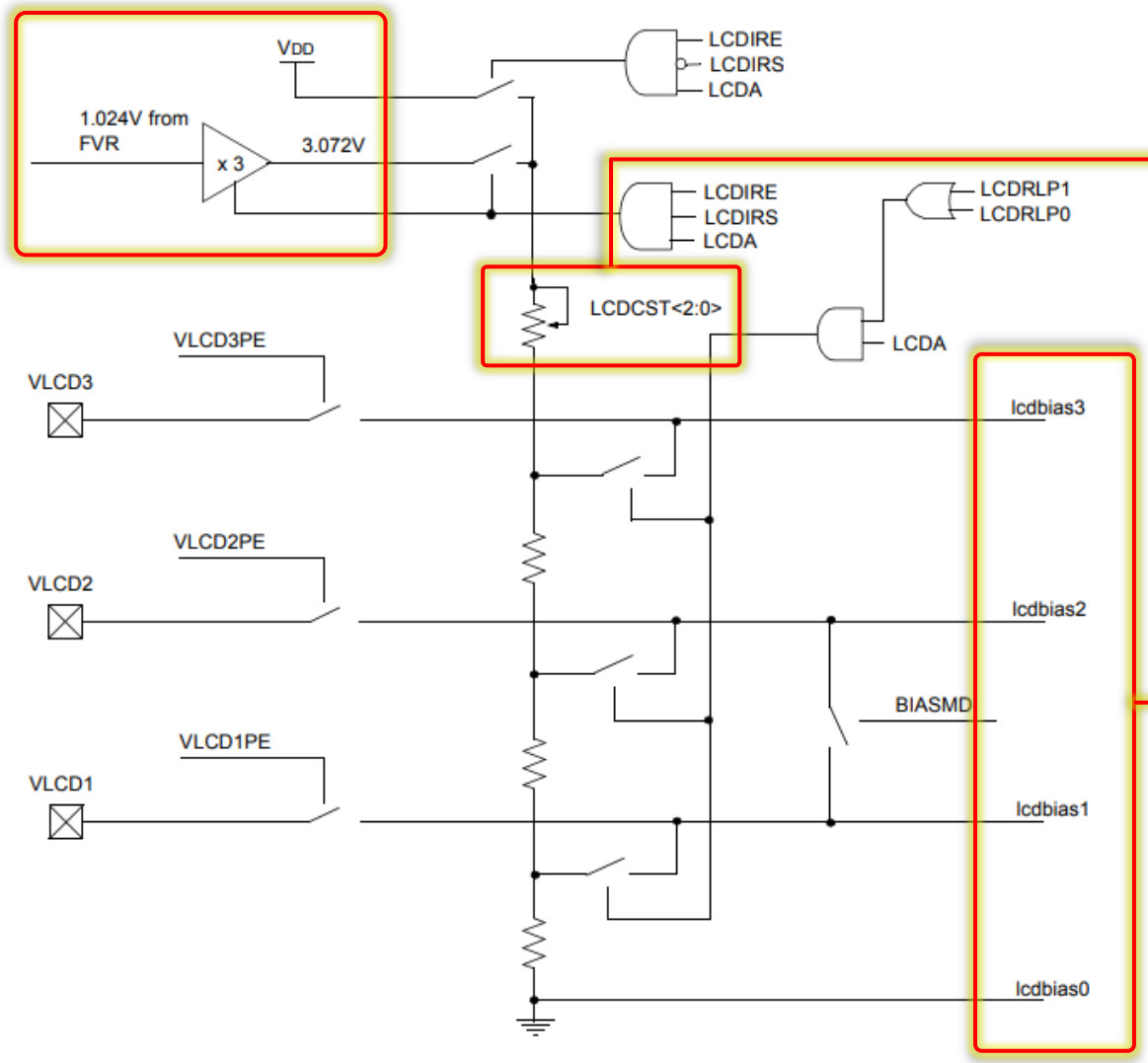
PIC16(L)F 1933/1936/1938	PIC16(L)F 1934/1937/1939	PIC16(L)F 1946/1947
$4 \times 15 = 60$	$4 \times 24 = 96$	$4 \times 46 = 184$

- **Drives a number of LCD types:**
  - Static (1 common)
    - Static Bais
  - 1/2 multiplex (2 common)
    - 1/2 or 1/3 Bais
  - 1/3 multiplex (3 common)
    - 1/2 or 1/3 Bais
  - 1/4 multiplex (4 common)
    - 1/4 Bais

Multiplex	Maximum Number of Pixels		Bias
	PIC16(L)F1936	PIC16(L)F1934/7	
Static (COM0)	16	24	Static
1/2 (COM<1:0>)	32	48	1/2 or 1/3
1/3 (COM<2:0>)	48	72	1/2 or 1/3
1/4 (COM<3:0>)	60(1)	96	1/3

MICROCHIP

LCD Bias Voltage Generation



Power Mode	Nominal Resistance of Entire Ladder	Nominal IDD
Low	3 Mohm	1 µA
Medium	300 kohm	10 µA
High	30 kohm	100 µA

LCD Internal Ladder power Modes(1/3 Bias)

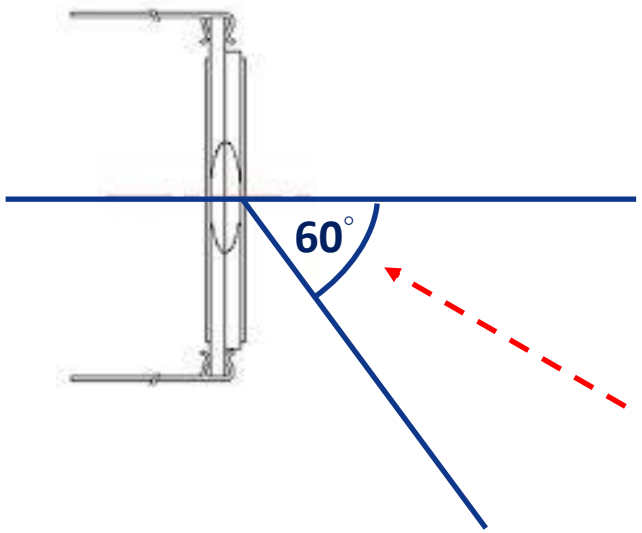
	Static Bias	1/2 Bias	1/3 Bias
LCD Bias 0	VSS	VSS	VSS
LCD Bias 1	—	1/2 VDD	1/3 VDD
LCD Bias 2	—	1/2 VDD	2/3 VDD
LCD Bias 3	VLCD3	VLCD3	VLCD3

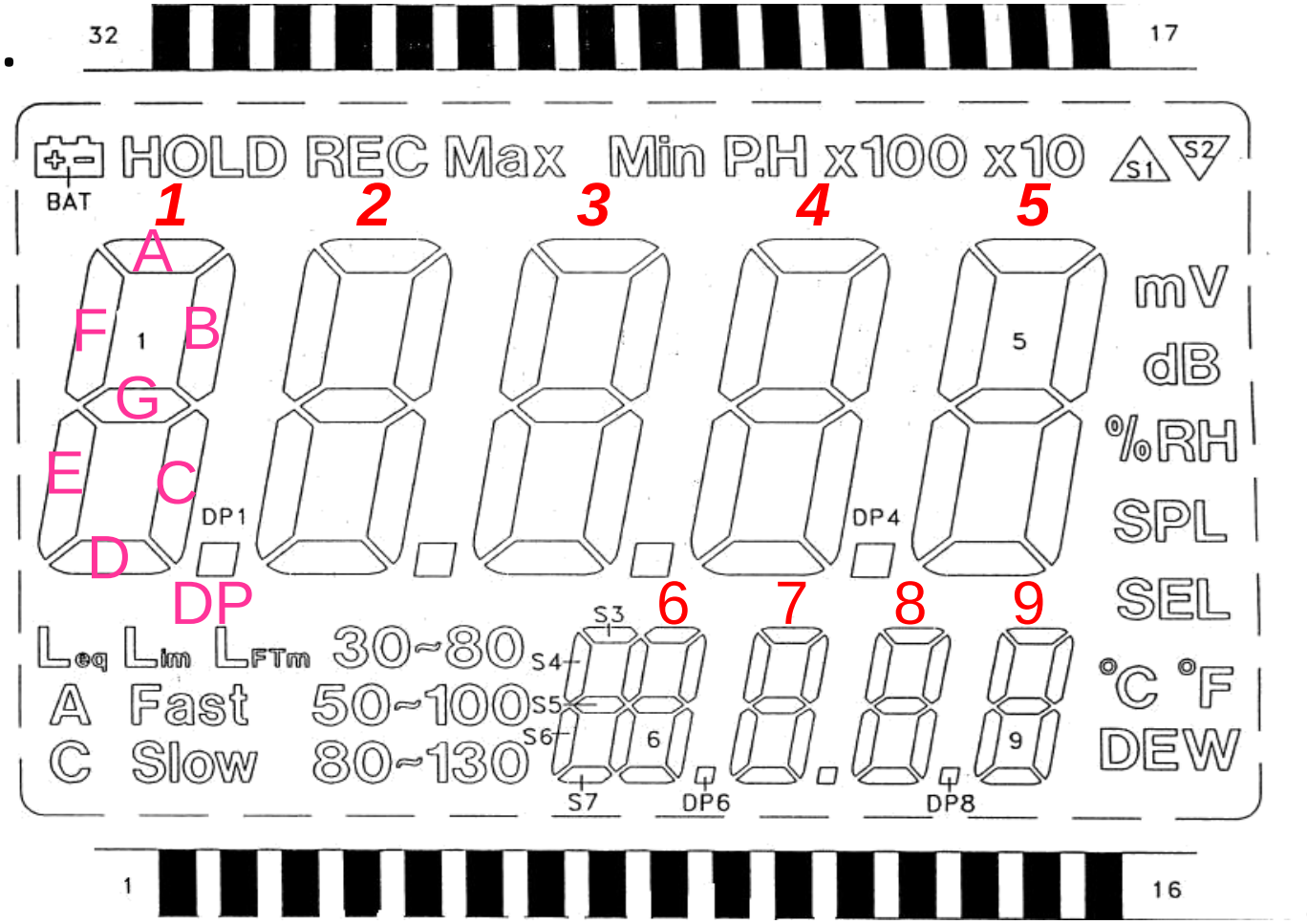
LCD Bias Voltages

MICROCHIP

LCD panel Specifications

- Operating voltage : **3.3V**.
- Operating temperature : -10℃～70℃.
- Drive Scheme : **1/4 Duty**、**1/3Bias**.
- Viewing Direction : 6:00 O'Clock.
- TN(Twisted Nematic) Type.





LCD Pin Configuration

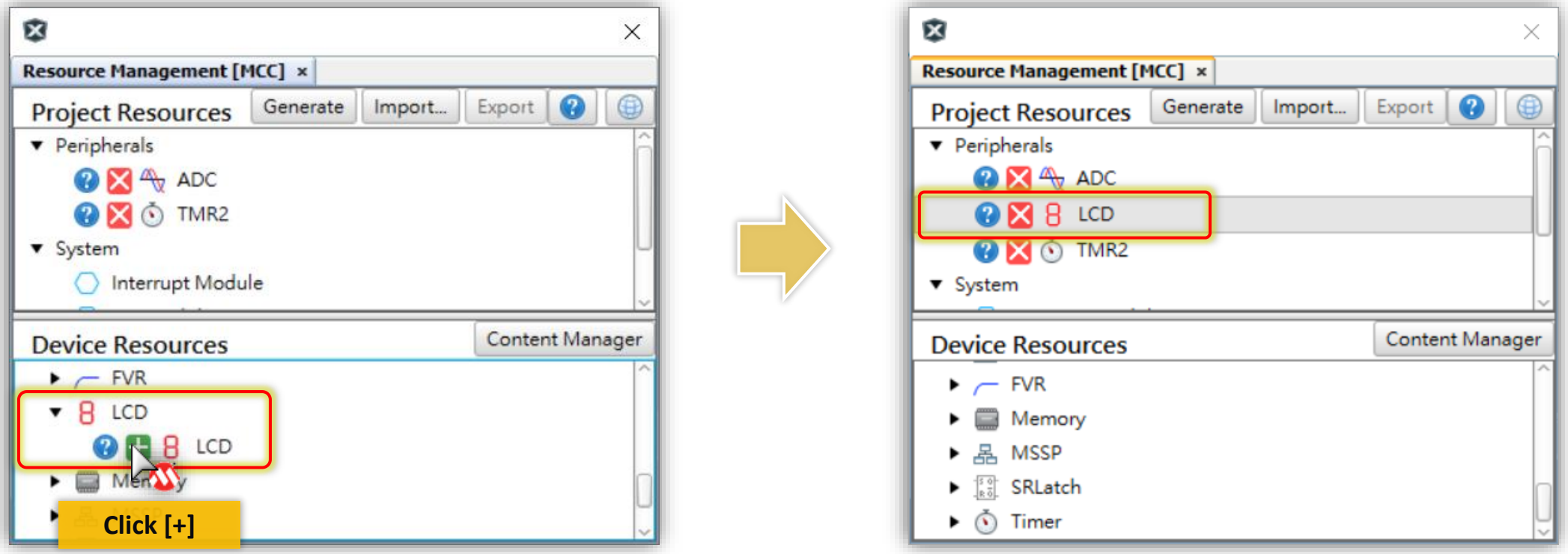
LCD PIN	COM0	COM1	COM2	COM3	Segment
1		L_{eq}	L_{im}	L_{FTm}	N.C.
2	S7	S6	S5	S4	N.C.
3	6D	6E	6G	6F	N.C.
4	6DP	6C	6B	6A	N.C.
5	7D	7E	7G	7F	N.C.
6	7DP	7C	7B	7A	N.C.
7	8D	8E	8G	8F	N.C.
8	8DP	8C	8B	8A	N.C.
9	9D	9E	9G	9F	N.C.
10		9C	9B	9A	N.C.
11		°F	°C	S3	SEG21
12	C	A	Slow	Fast	SEG5
13	COM0				COM0
14		COM1			COM1
15			COM2		COM2
16				COM3	COM3

LCD Pin Configuration

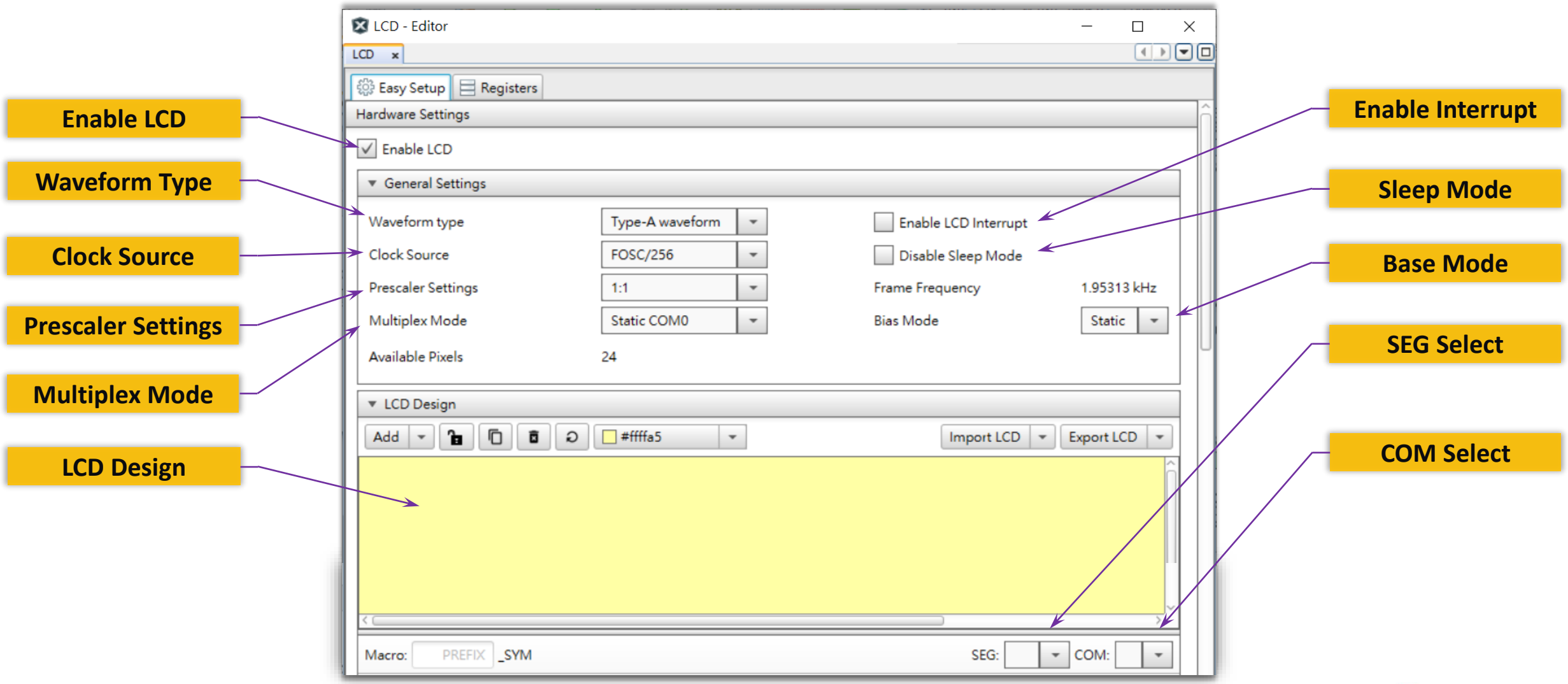
LCD PIN	COM0	COM1	COM2	COM3	Segment
17	DEW	80~130	50~100	30~80	N.C.
18	SEL	SPL	dB	%RH	N.C.
19	m	V	S2	S1	SEG7
20		5C	5B	5A	SEG15
21	5D	5E	5G	5F	SEG4
22					N.C.
23	P.H	x10	x100	Min	N.C.
24	4DP	4C	4B	4A	SEG22
25	4D	4E	4G	4F	SEG23
26	3DP	3C	3B	3A	SEG3
27	3D	3E	3G	3F	SEG16
28	Max	REC	HOLD	BAT	SEG17
29	2DP	2C	2B	2A	SEG18
30	2D	2E	2G	2F	SEG19
31	1DP	1C	1B	1A	SEG20
32	1D	1E	1G	1F	SEG0

MCC's LCD Module

- **Add LCD resource from Device Resources.**

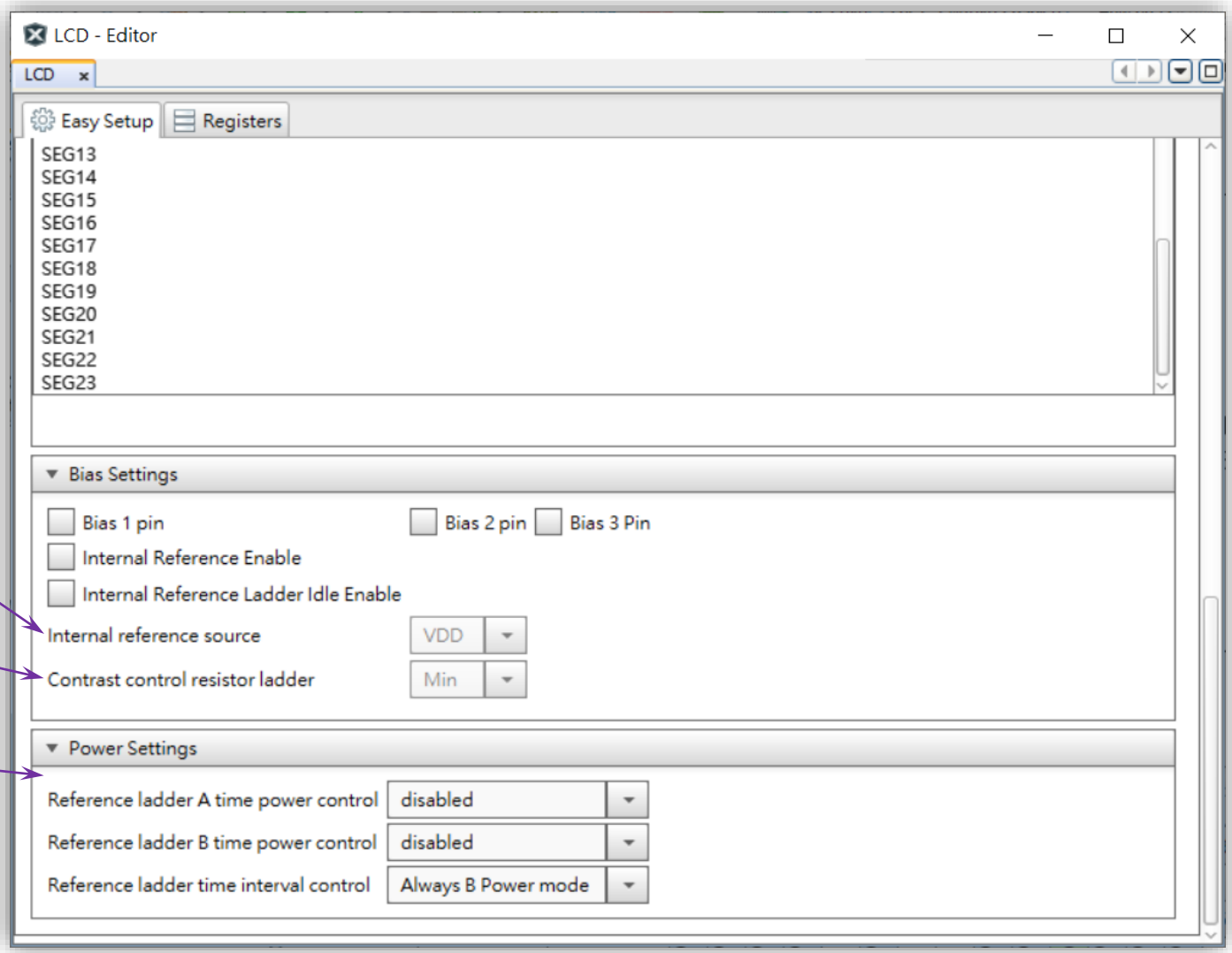


MCC's LCD Module



MICROCHIP

MCC's LCD Module



Voltage reference

Contrast control

Power Setting

MICROCHIP

LCD Functions of MCC

- **MCC Provide below functions for LCD:**

  Prototype definition : "lcd.h"

  - Void LCD_Initialize (void);
  - void LCD_Enable (void);
  - void LCD_EnableSleepMode (void)
  - void LCD_Disable (void);
  - void LCD_DisableSleepMode (void);
  - bool LCD_IsActive (void);
  - void LCD_SetContrast (unsigned int value);
  - void LCD_SetIntervalAPowerMode (unsigned int value);
  - void LCD_SetIntervalBPowerMode (unsigned int value);
  - void LCD_SetPowerDistribution (unsigned int value);

Lab7 LCD Display

- **Base on current project or Open .\Lab7_xxx.X to do exercises**
- **Try to add LCD module and conversion Potentiometer (VR) voltage to shows ADC result on LCD Display.**

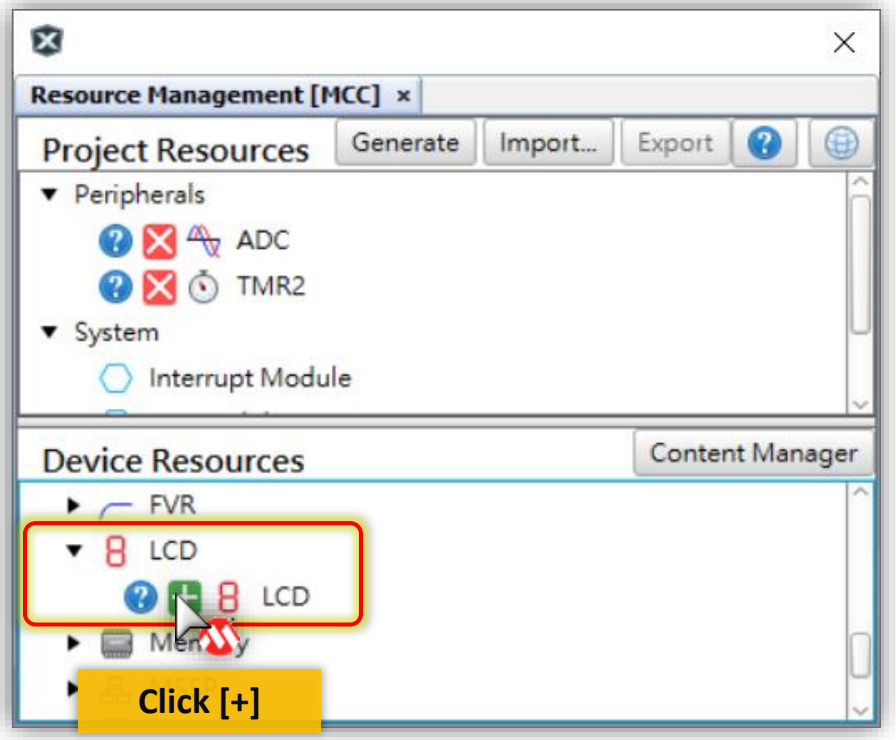
Let’s go!

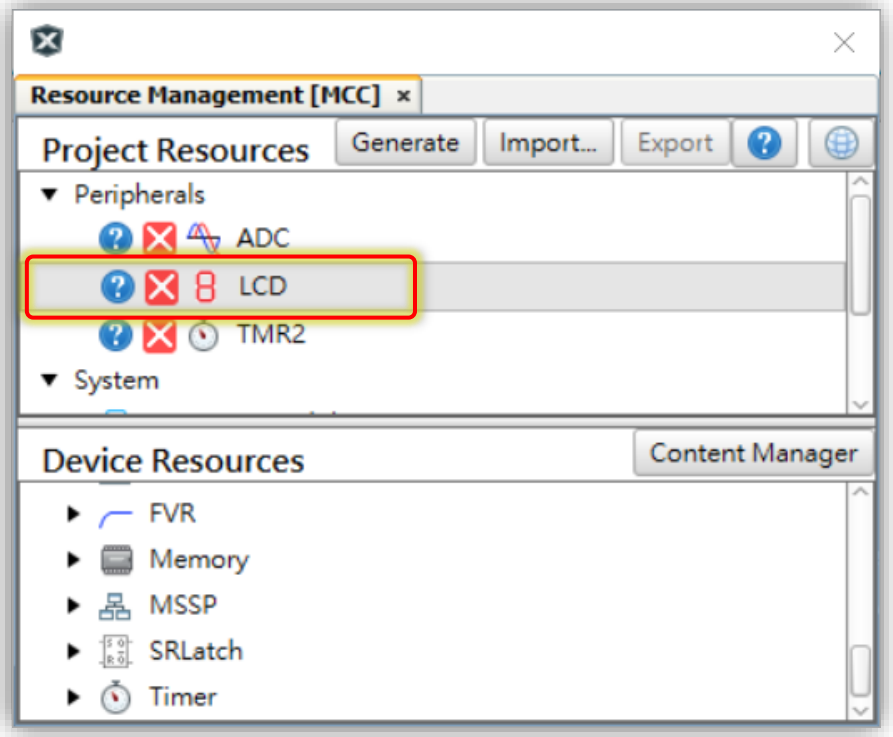
MICROCHIP

Lab7 LCD Display

Step 1

- **Device Resources**
  - **LCD ▸ LCD**

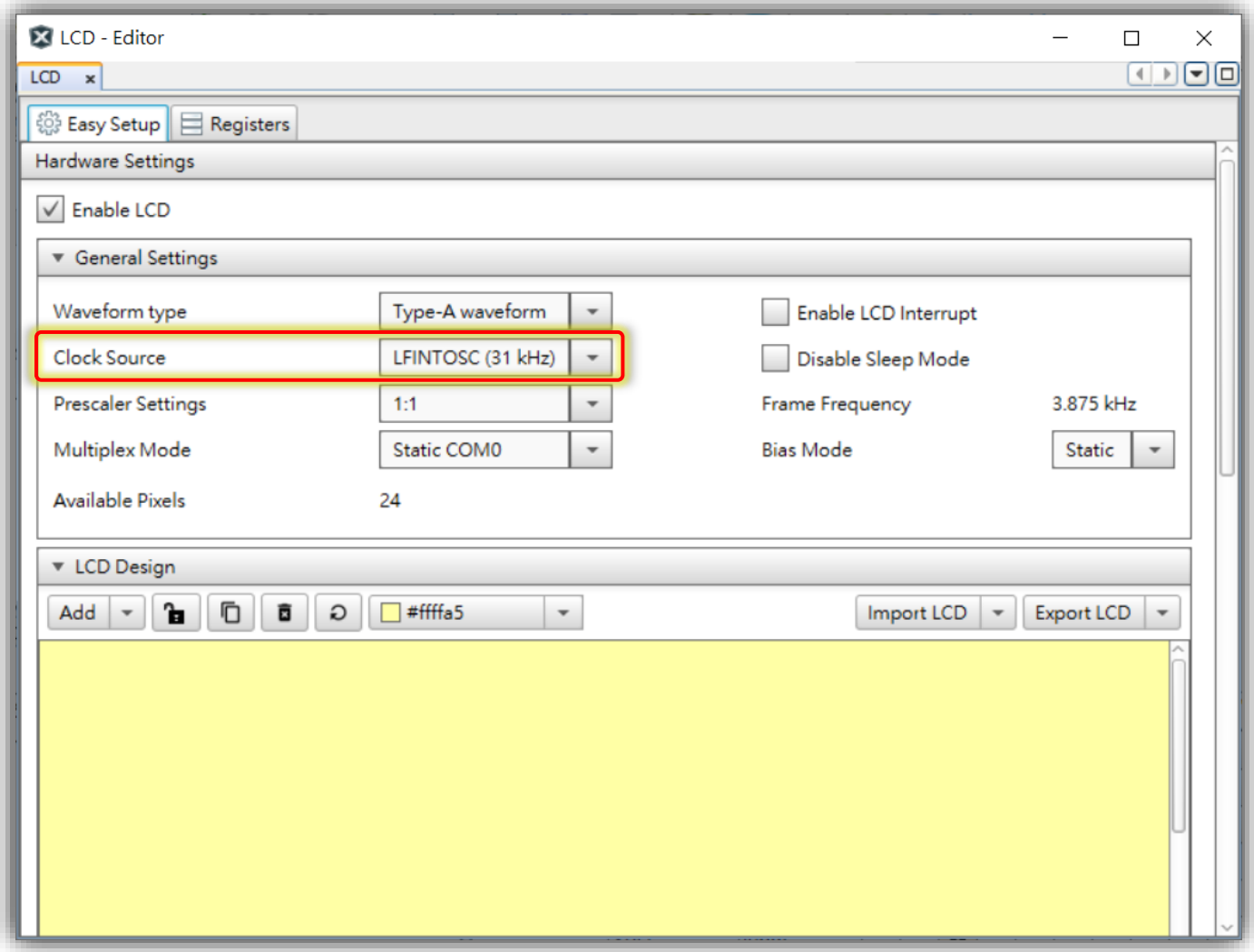




Lab7 LCD Display

Step 2

- **Clock Source**
  - **Fosc/256 ▸ LFINTOSC(31 kHZ)**

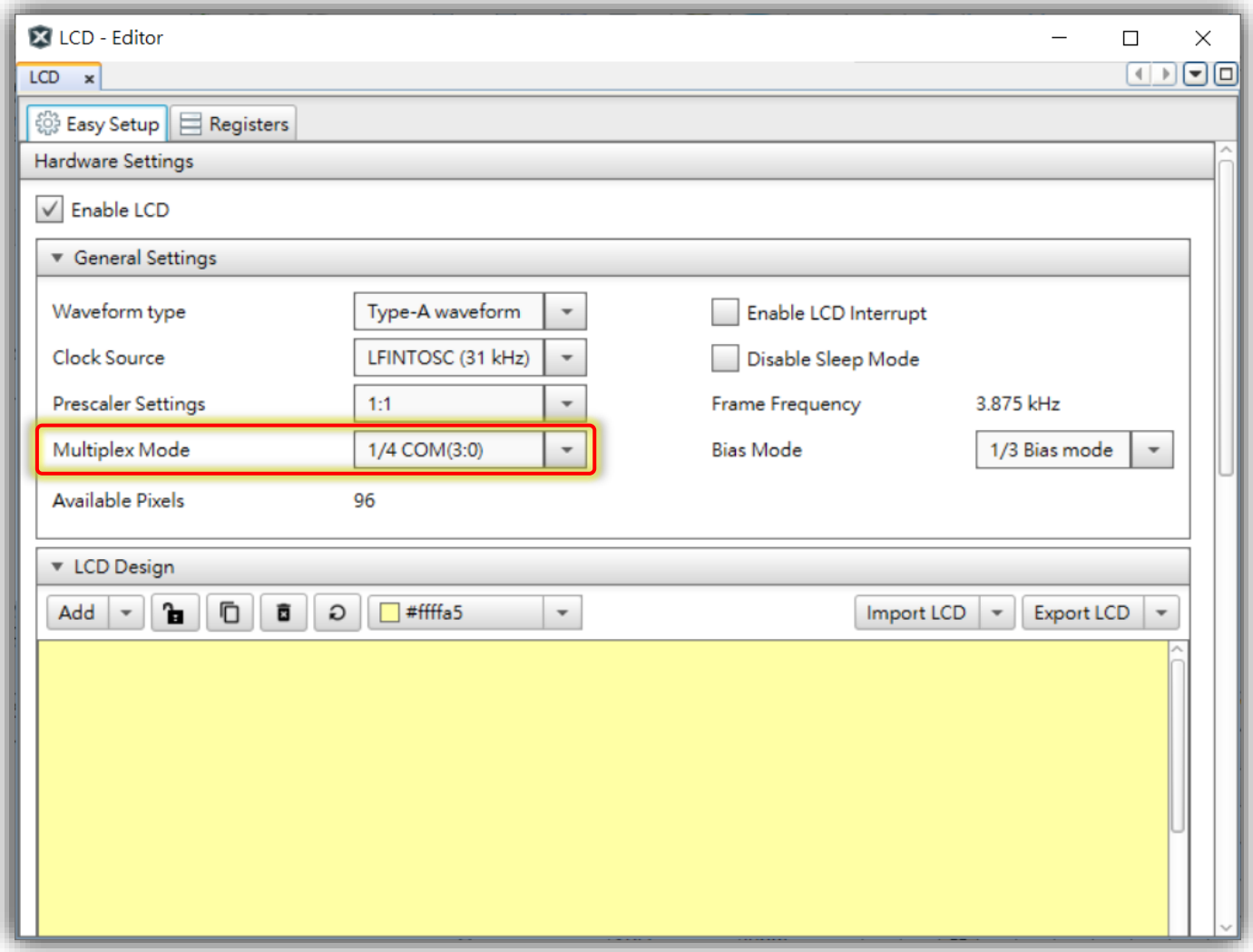


MICROCHIP

Lab7 LCD Display

Step 3

- **Multiplex Mode**
  - **Static COM0 ▸ 1/4 COM(3:0)**

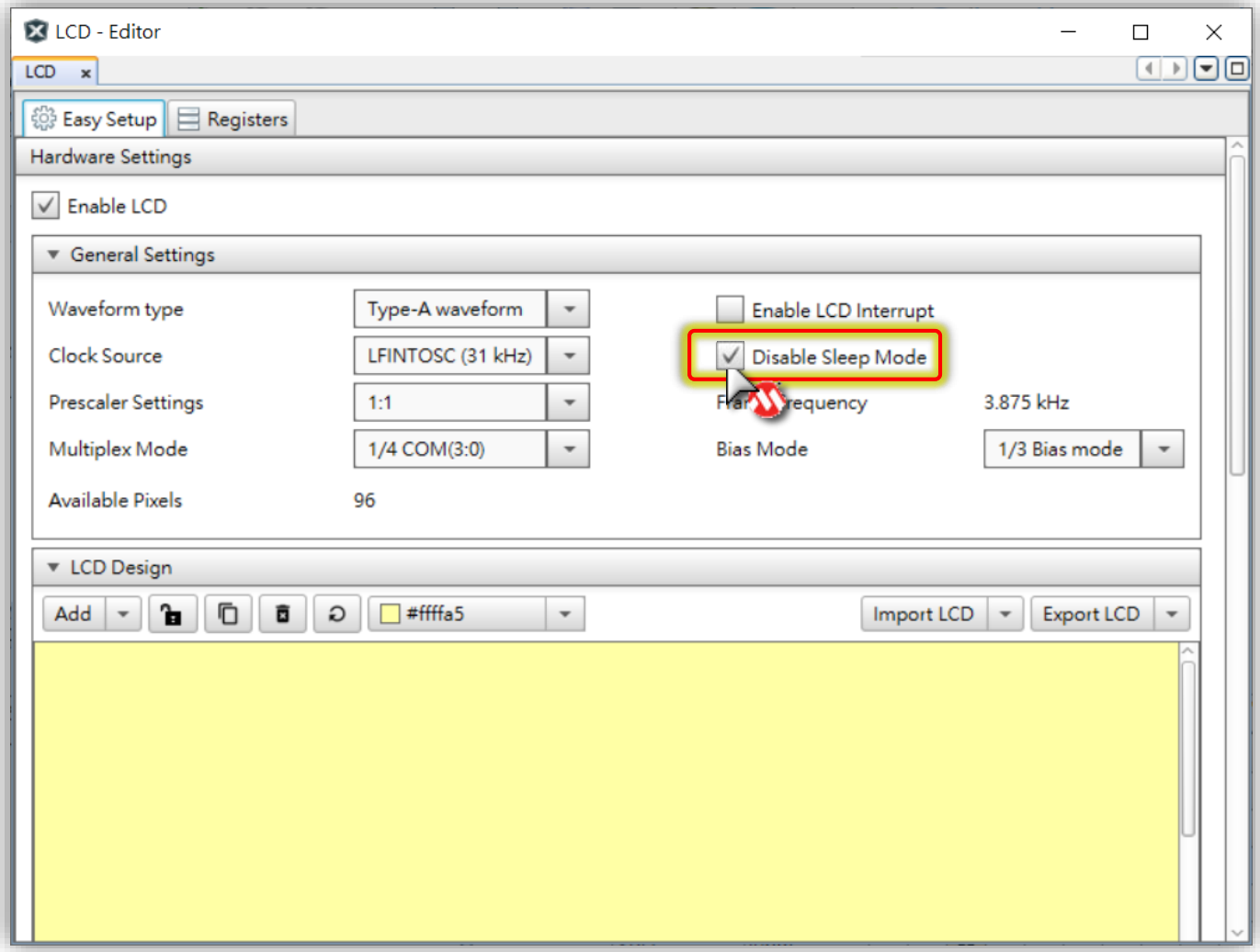


MICROCHIP

Lab7 LCD Display

Step 4

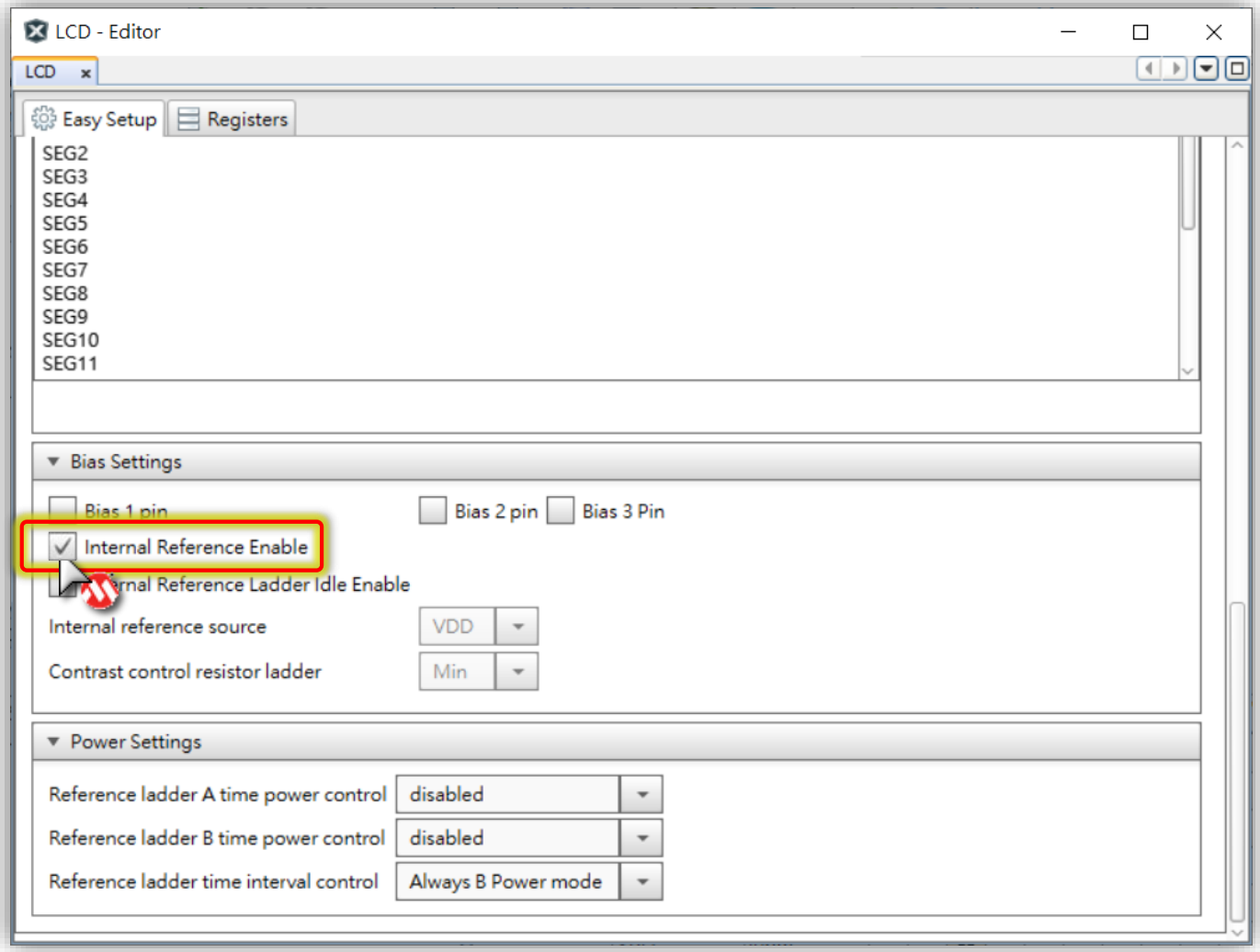
- **Disable Sleep Mode.**



Lab7 LCD Display

Step 5

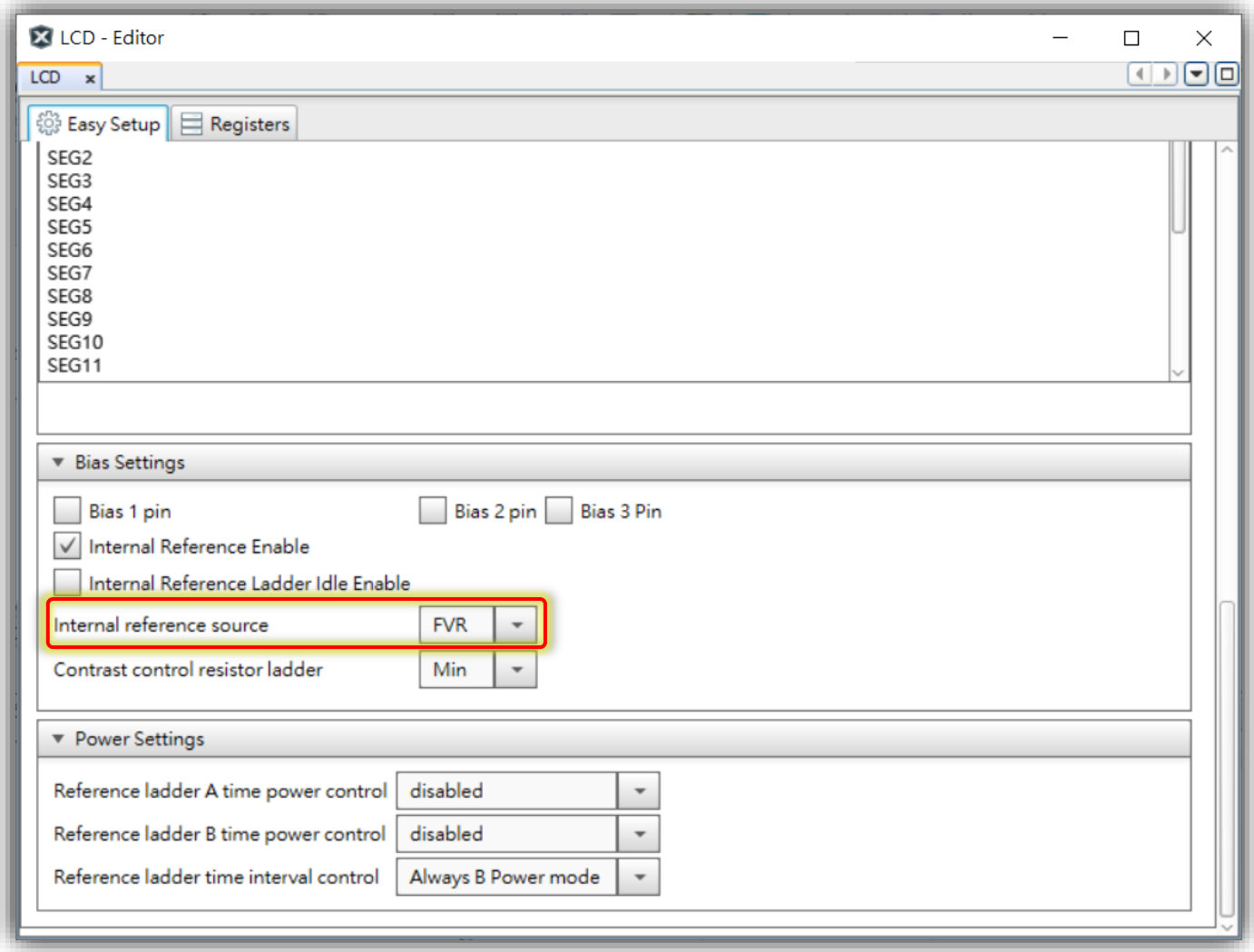
- **Enable Internal Reference.**



Lab7 LCD Display

Step 6

- **Internal reference source**
  - **VDD ▸ FVR**

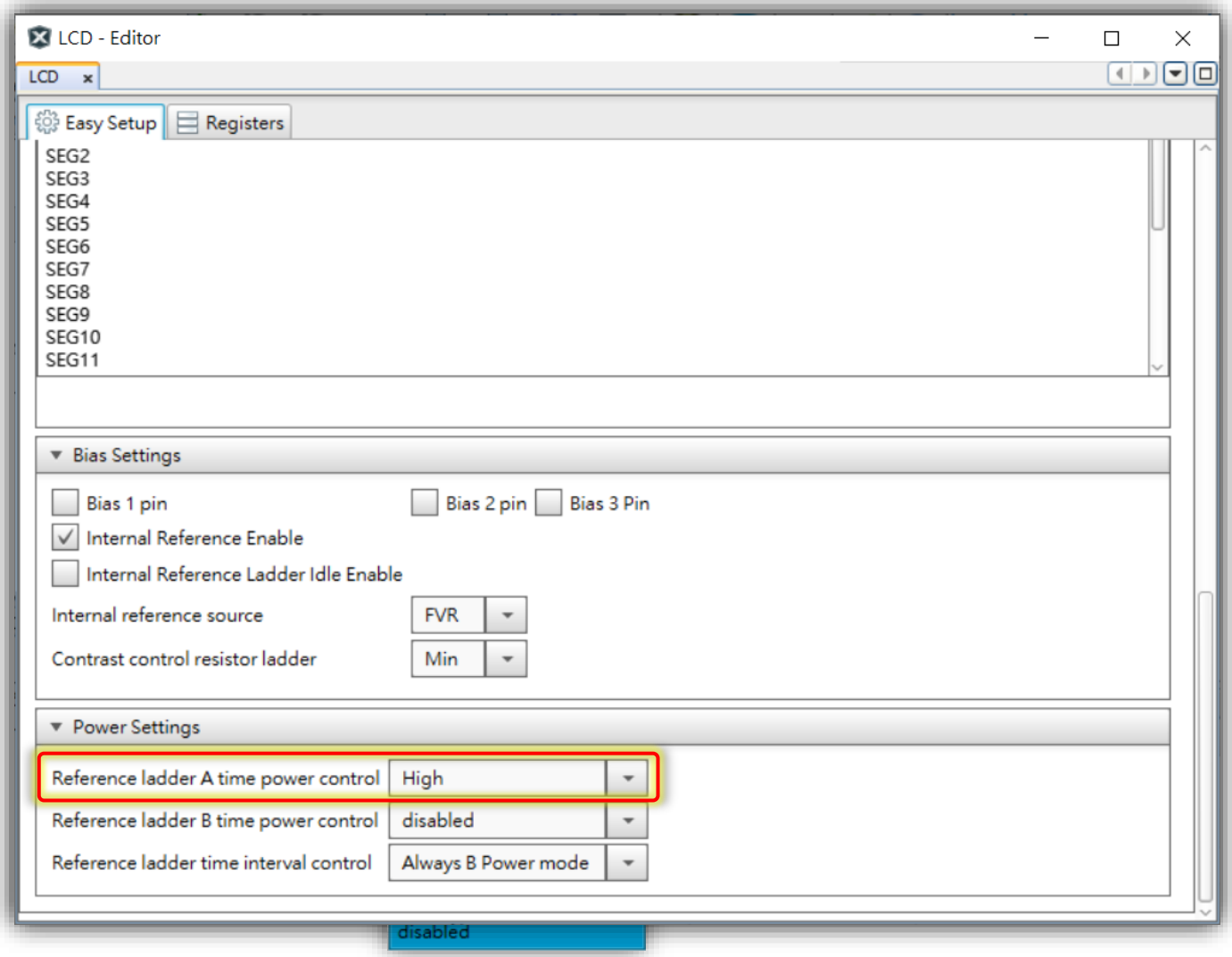


MICROCHIP

Lab7 LCD Display

Step 7

- **Reference ladder A time …**
  - **disabled ▸ High**

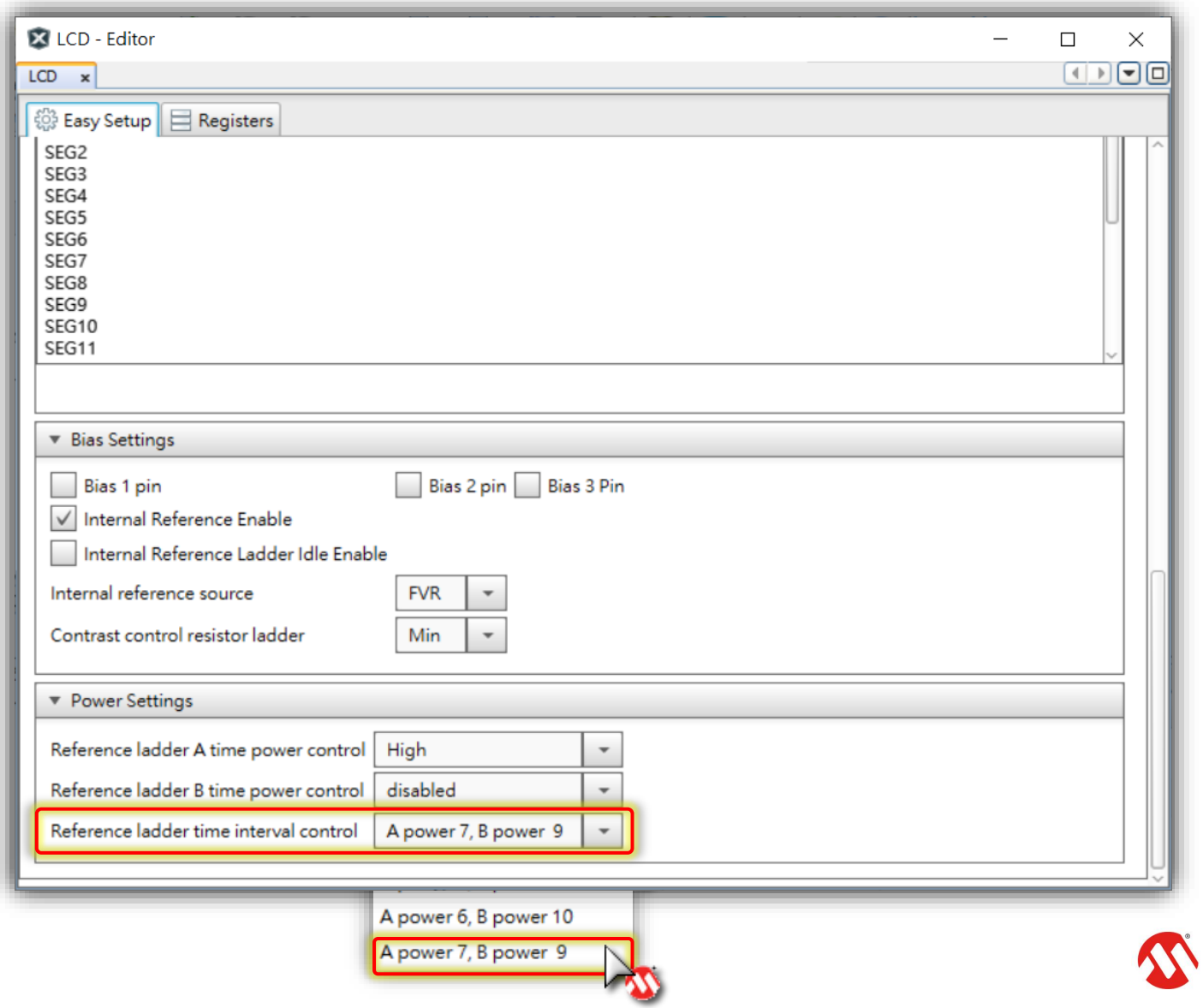


MICROCHIP

Lab7 LCD Display

Step 8

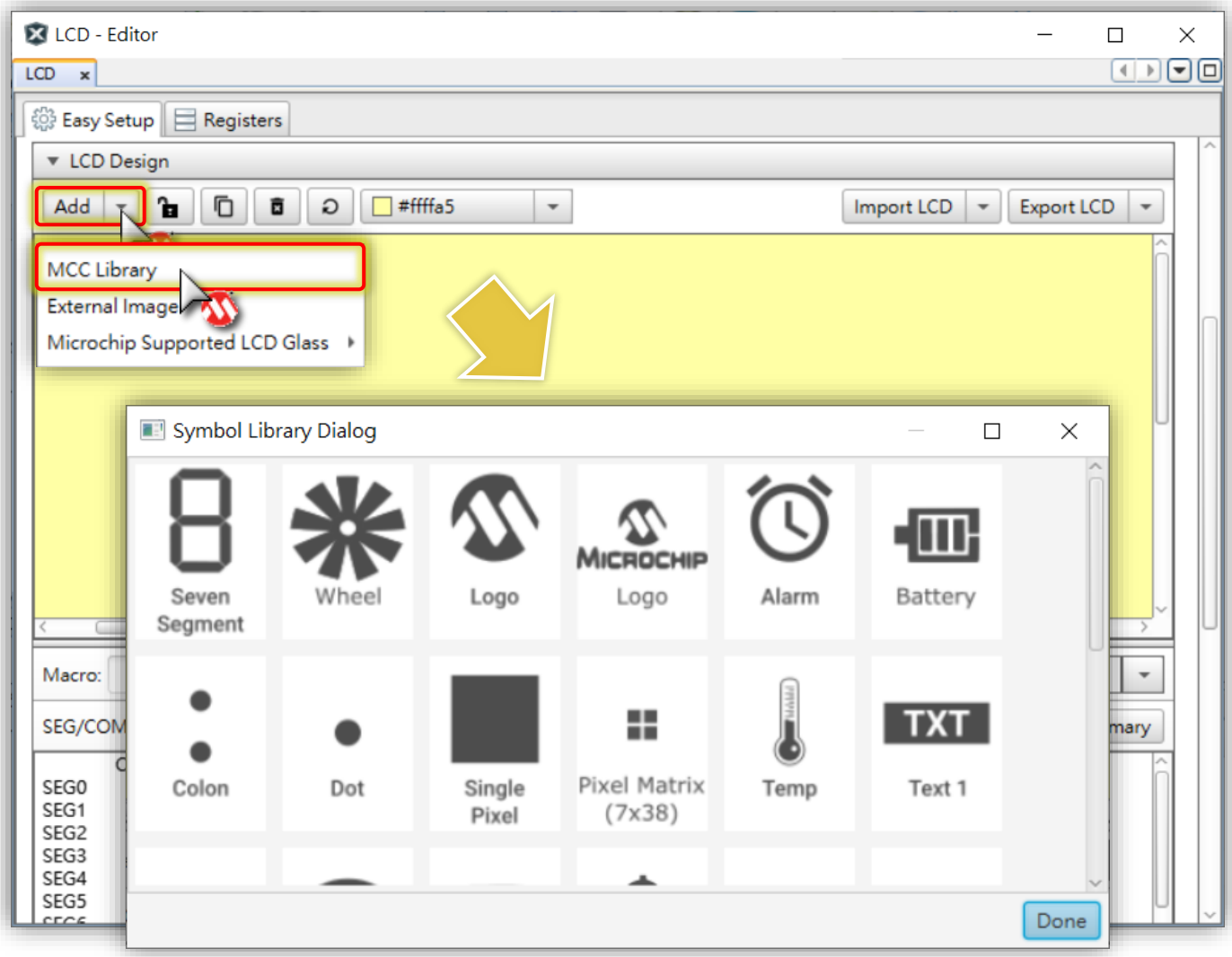
- **Reference ladder time …**
  - **Always B Power mode**
  - ▶
  - **A power 7, B power 9**



Lab7 LCD Display

Step 9

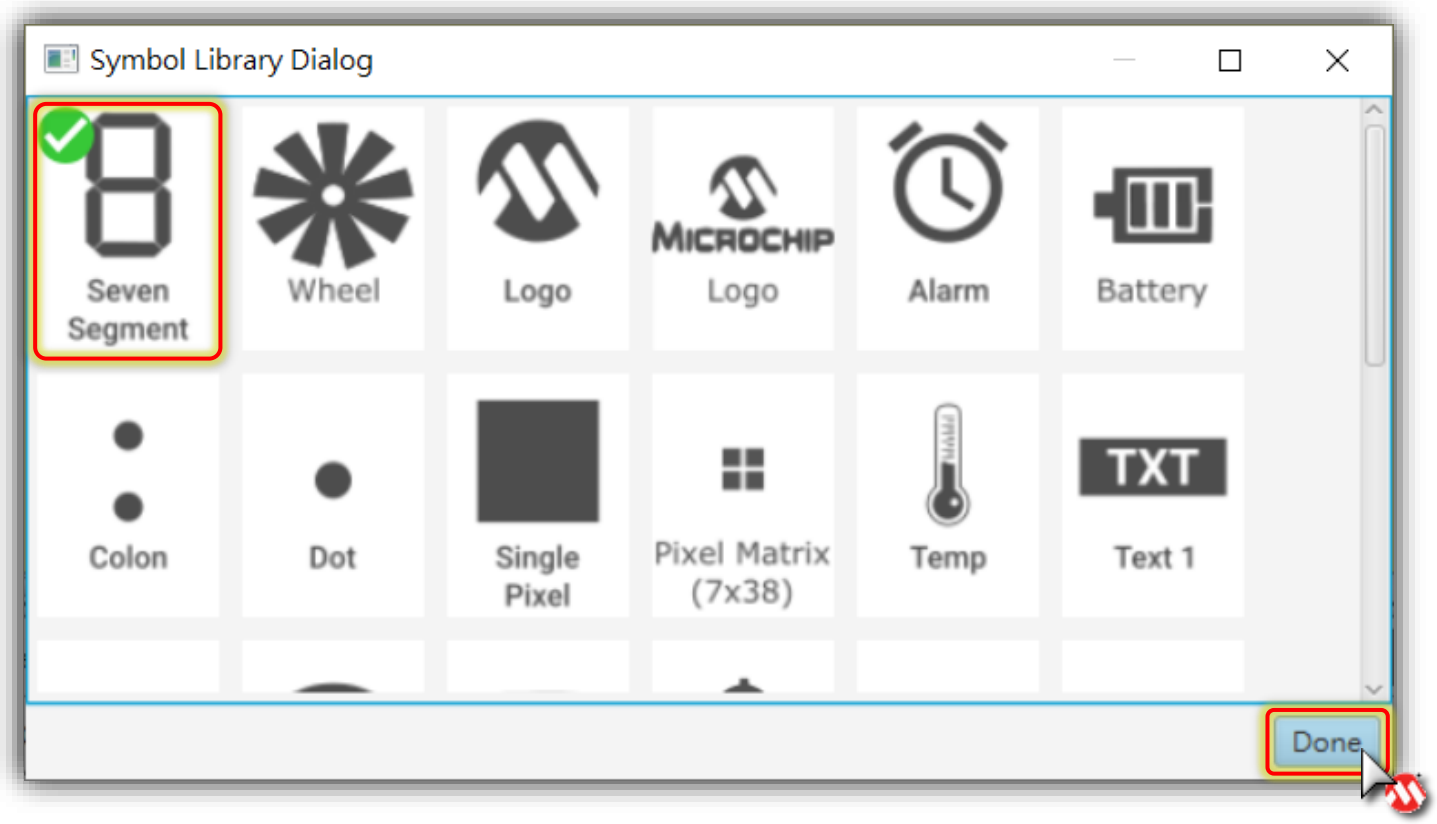
- **LCD Design**
  - **Add ▸ MCC Library**



Lab7 LCD Display

Step 10

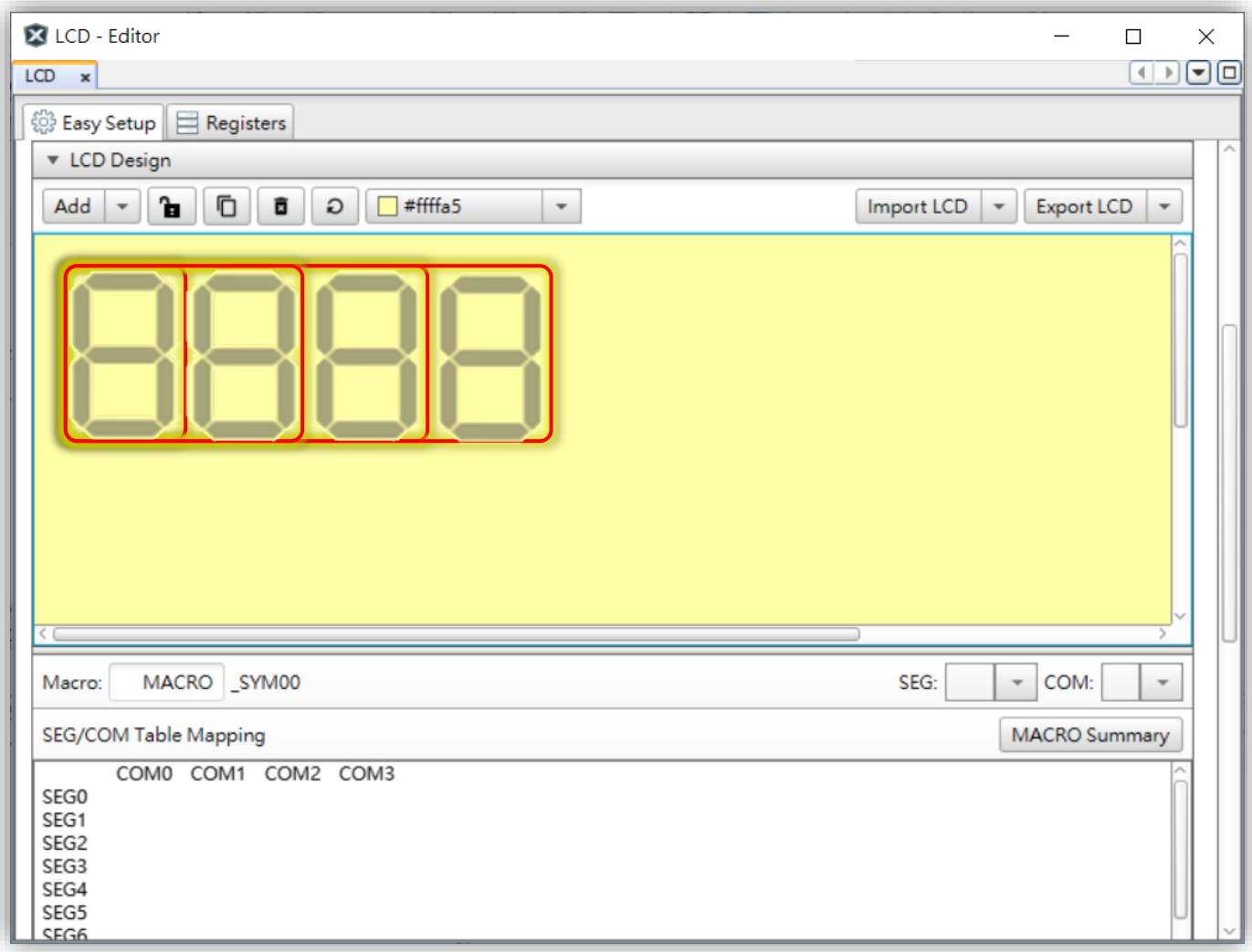
- **LCD Design**
  - **Select Seven Segment**



Lab7 LCD Display

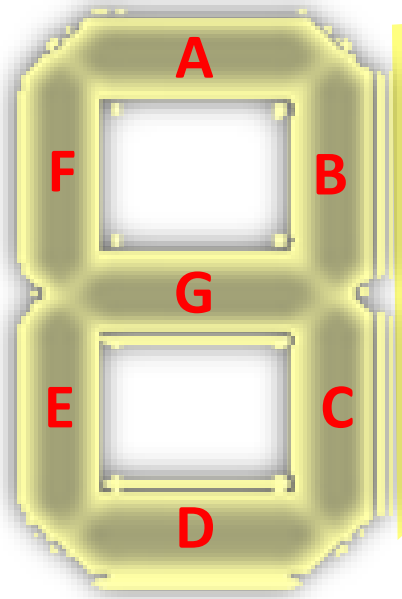
Step 11

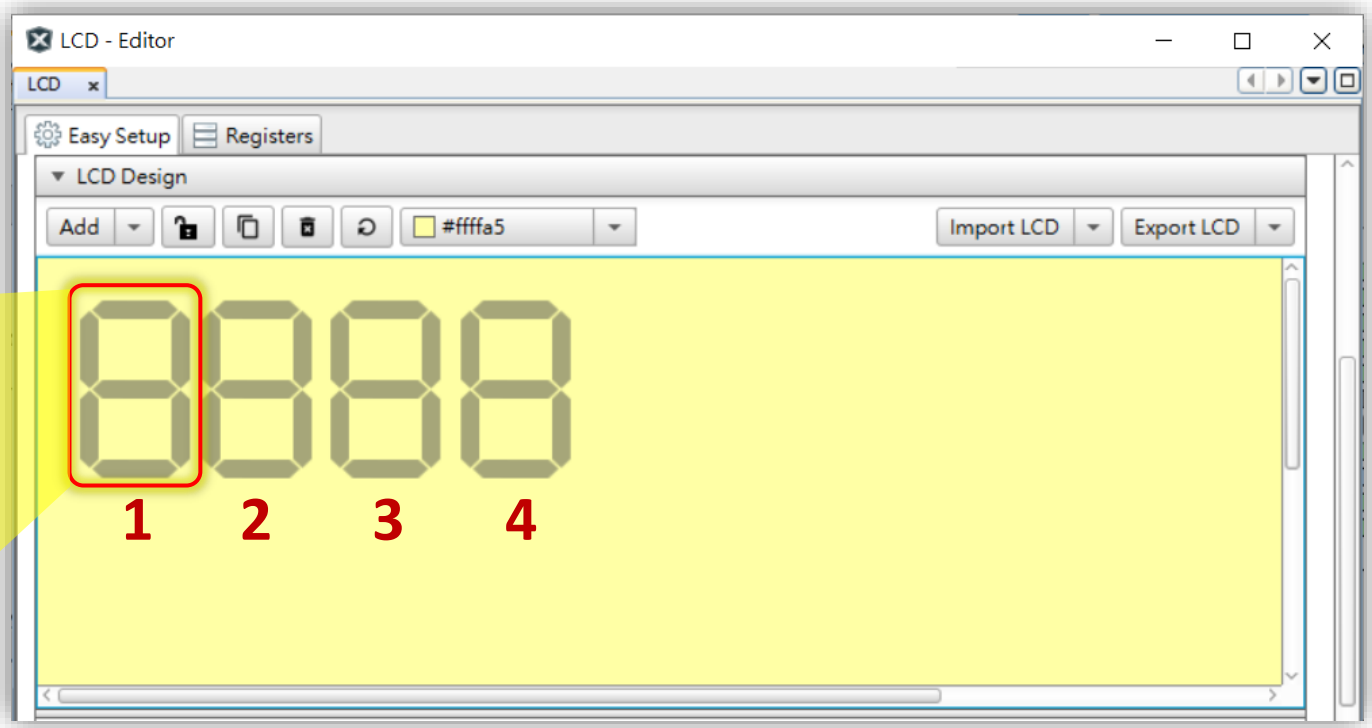
- **LCD Design.**



Lab7 LCD Display

Step 12





LCD PIN	COM0	COM1	COM2	COM3	Segment
31	1DP	1C	1B	1A	SEG20
32	1D	1E	1G	1F	SEG0

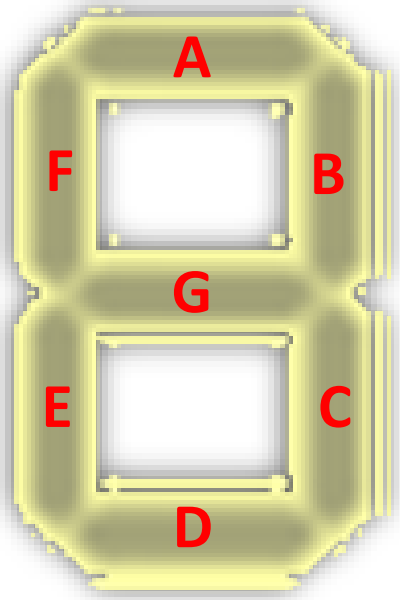
MICROCHIP

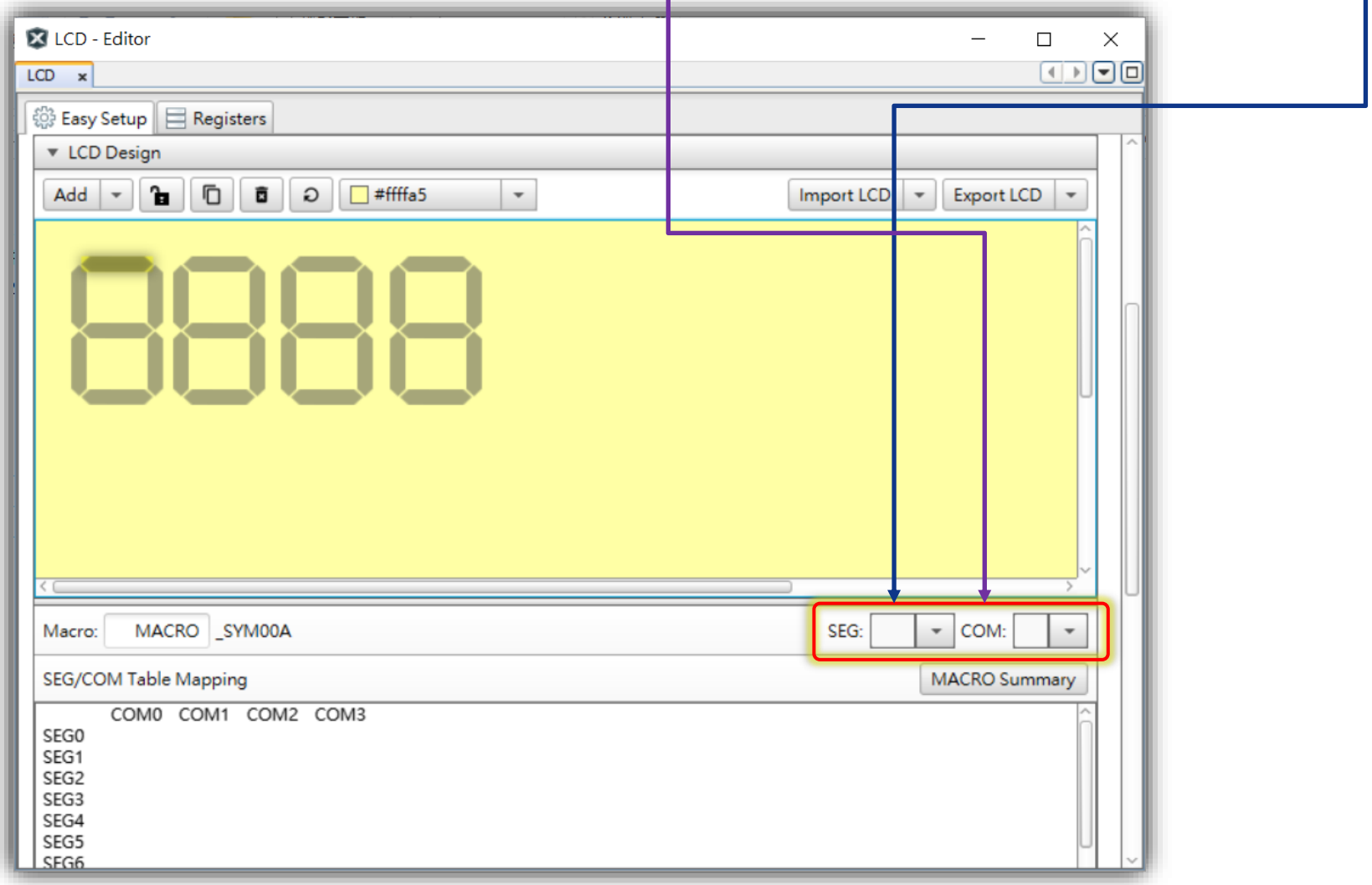
Lab7 LCD Display

Step 13

- **Assign Segment and COM.**

LCD PIN	COM0	COM1	COM2	COM3	Segment
31	1DP	1C	1B	1A	SEG20
32	1D	1E	1G	1F	SEG0



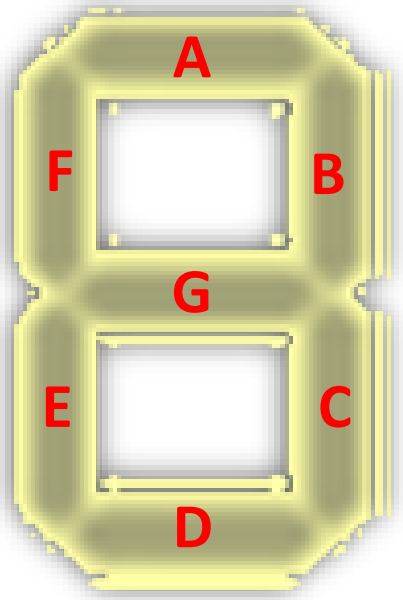


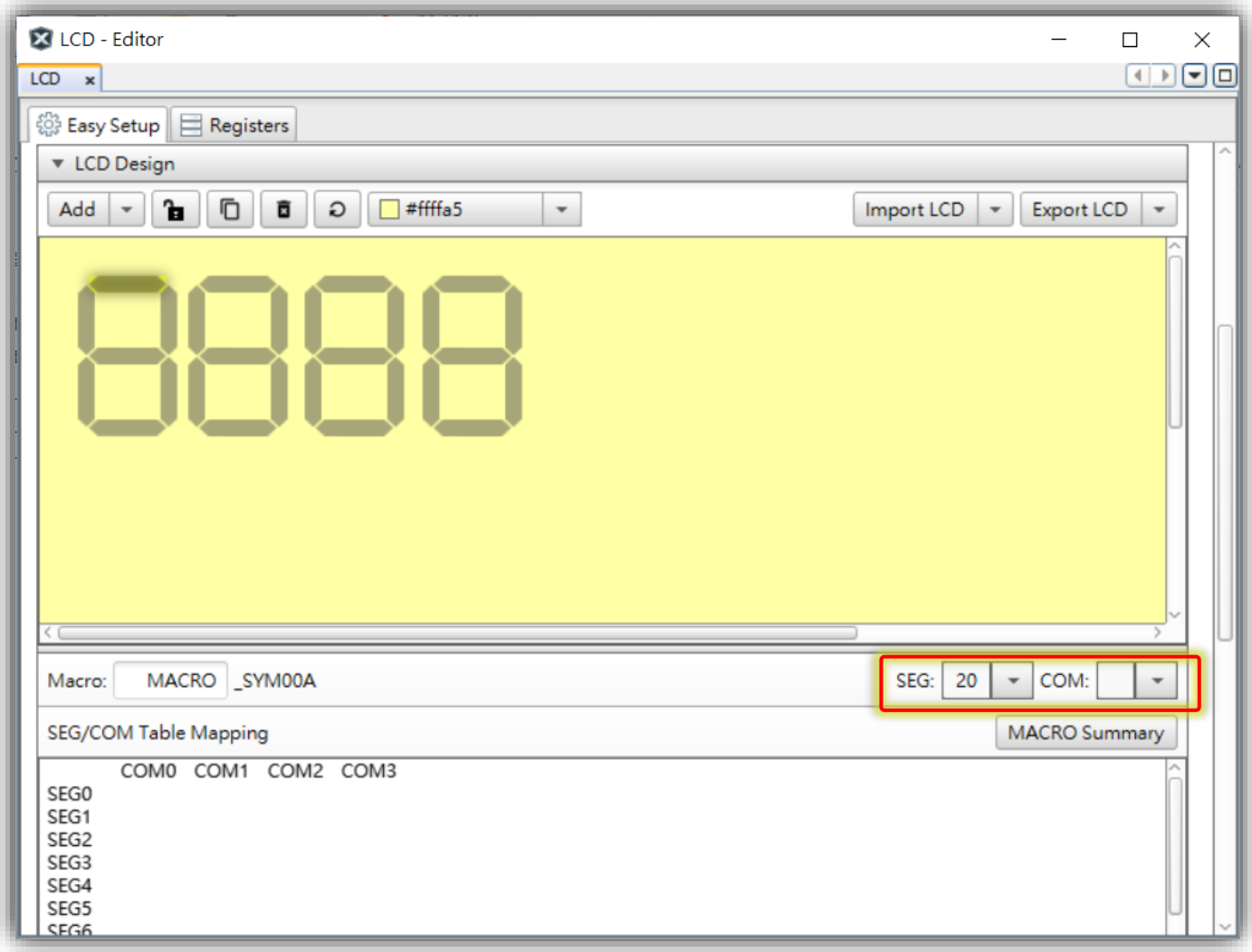
Lab7 LCD Display

Step 14

- **Assign Segment and COM.**
  - **Segment A ▸ SEG20**

LCD PIN	COM0	COM1	COM2	COM3	Segment
31	1DP	1C	1B	1A	SEG20
32	1D	1E	1G	1F	SEG0



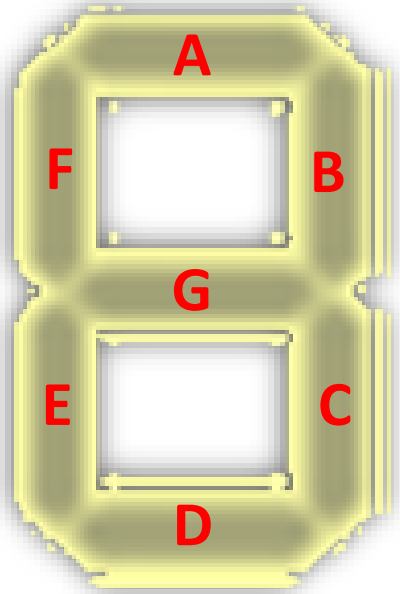


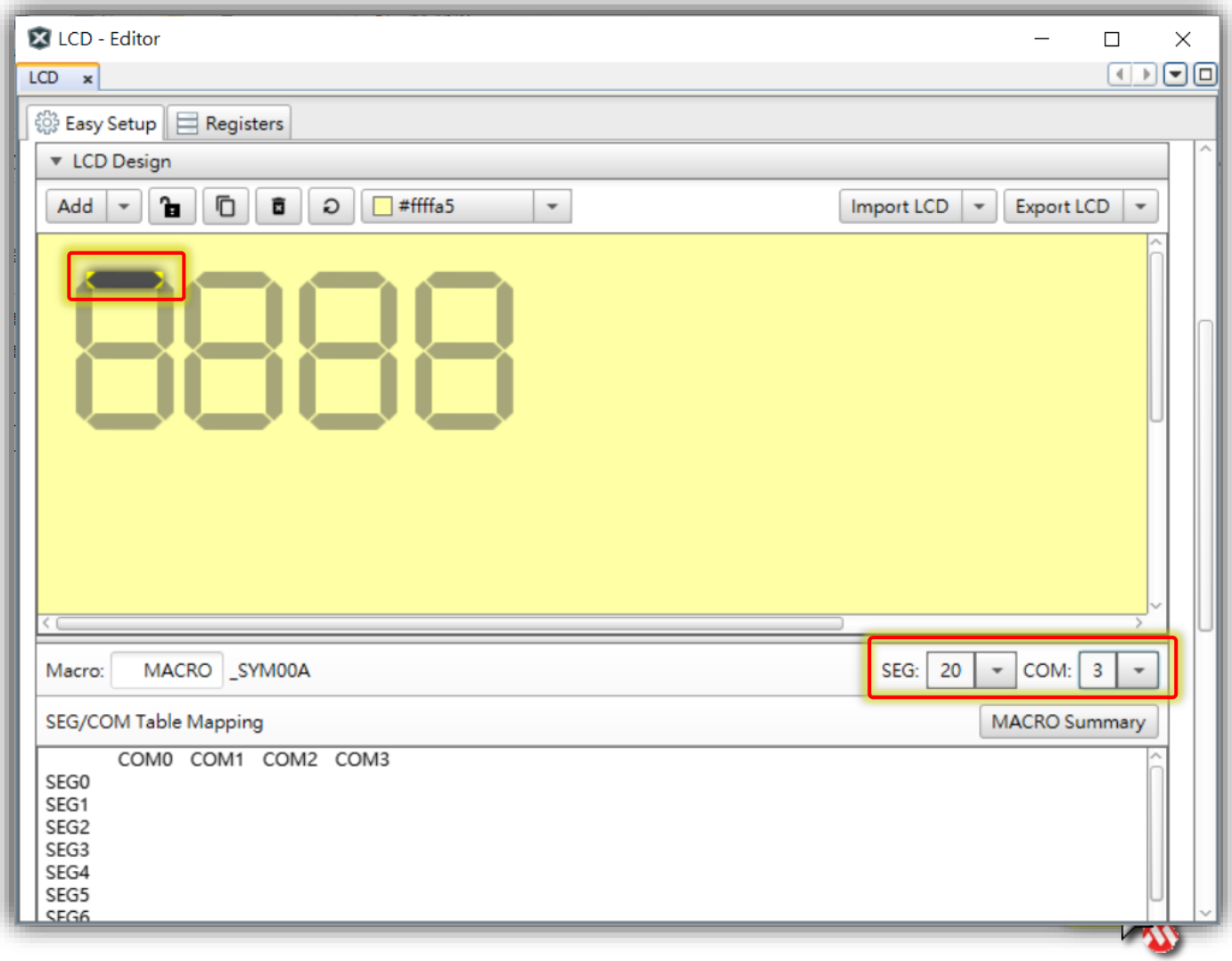
Lab7 LCD Display

Step 15

- **Assign Segment and COM.**
  - **Segment A ▸ COM3**

LCD PIN	COM0	COM1	COM2	COM3	Segment
31	1DP	1C	1B	1A	SEG20
32	1D	1E	1G	1F	SEG0

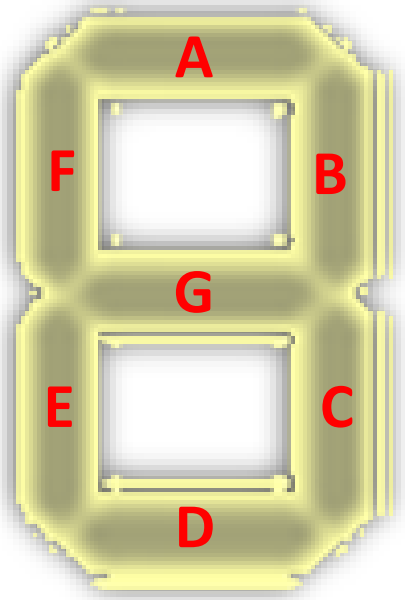


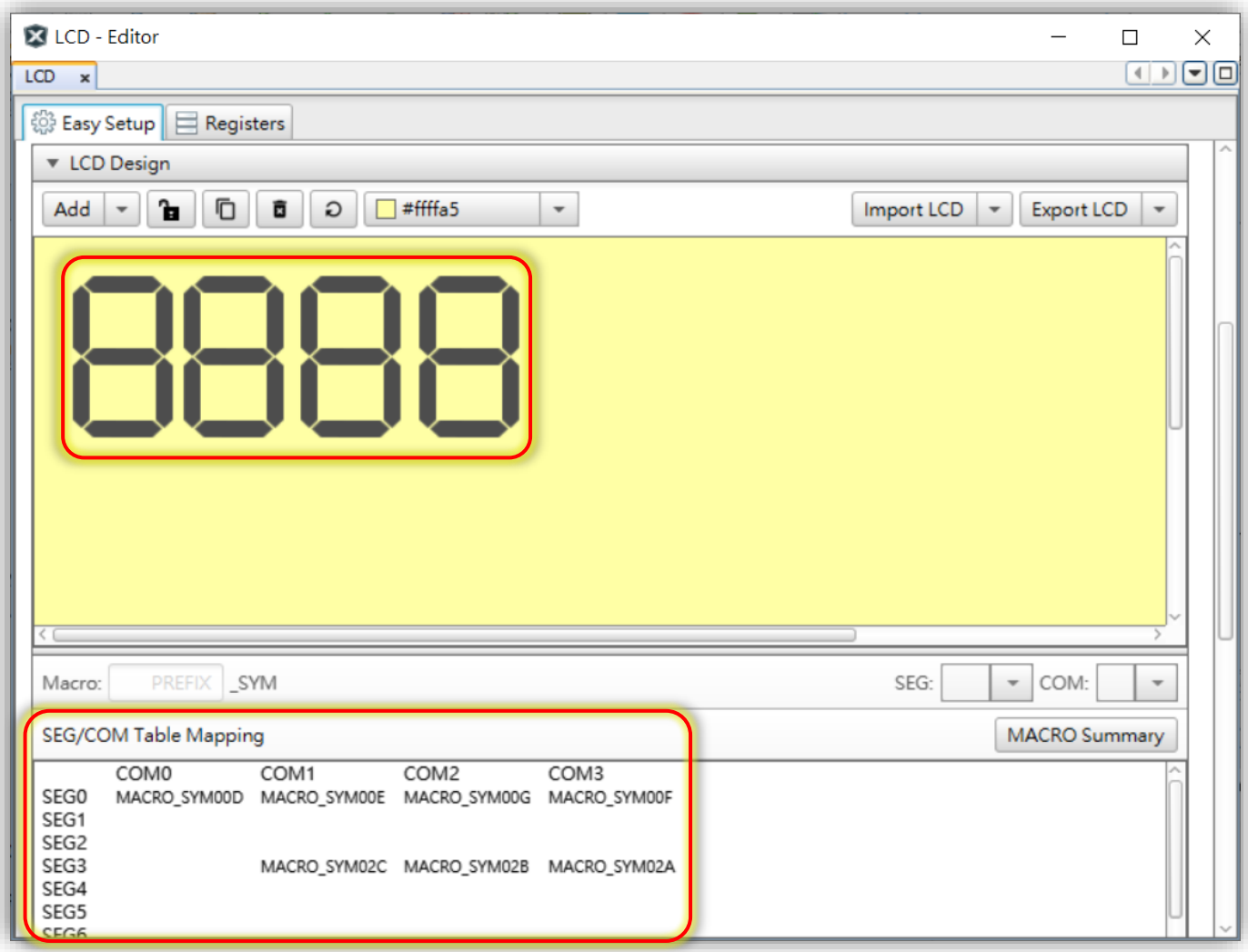


Lab7 LCD Display

Step 16

- **Confirm Segment and COM.**



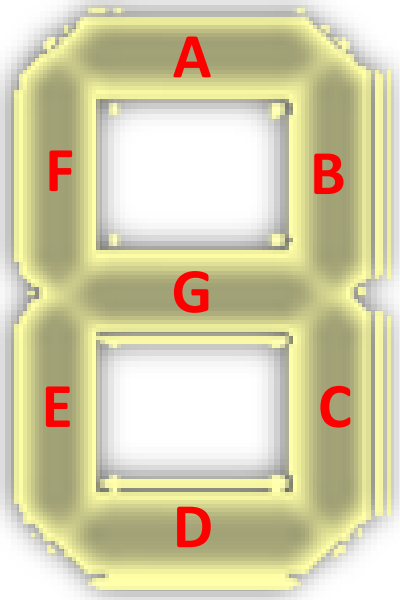


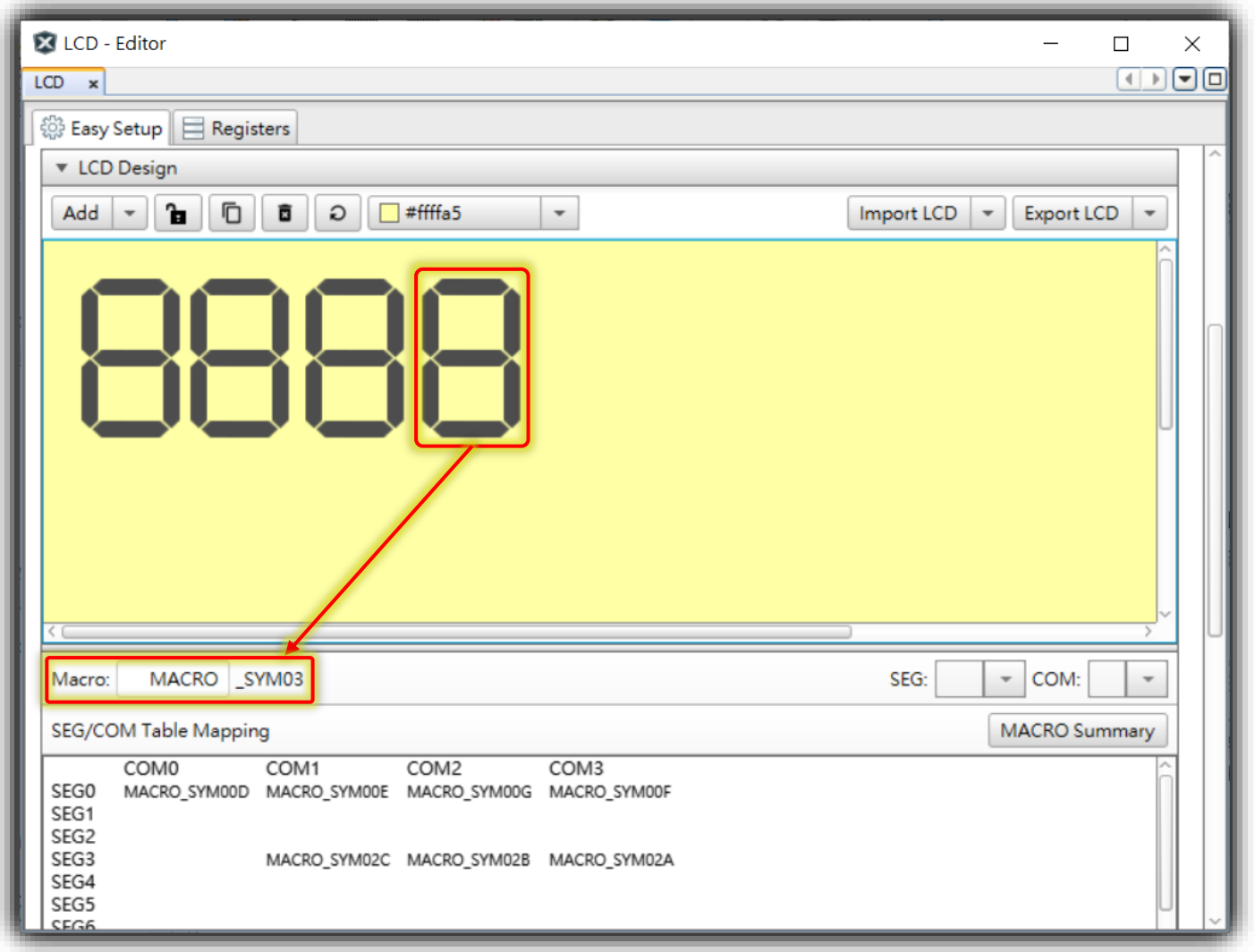
MICROCHIP

Lab7 LCD Display

Step 17

- **Confirm Macro.**



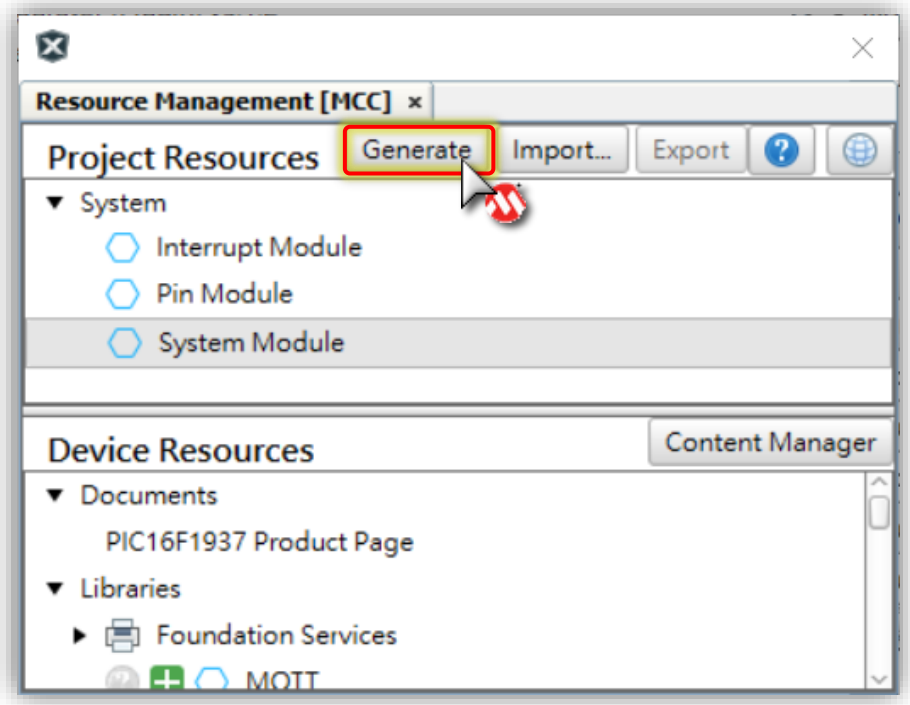


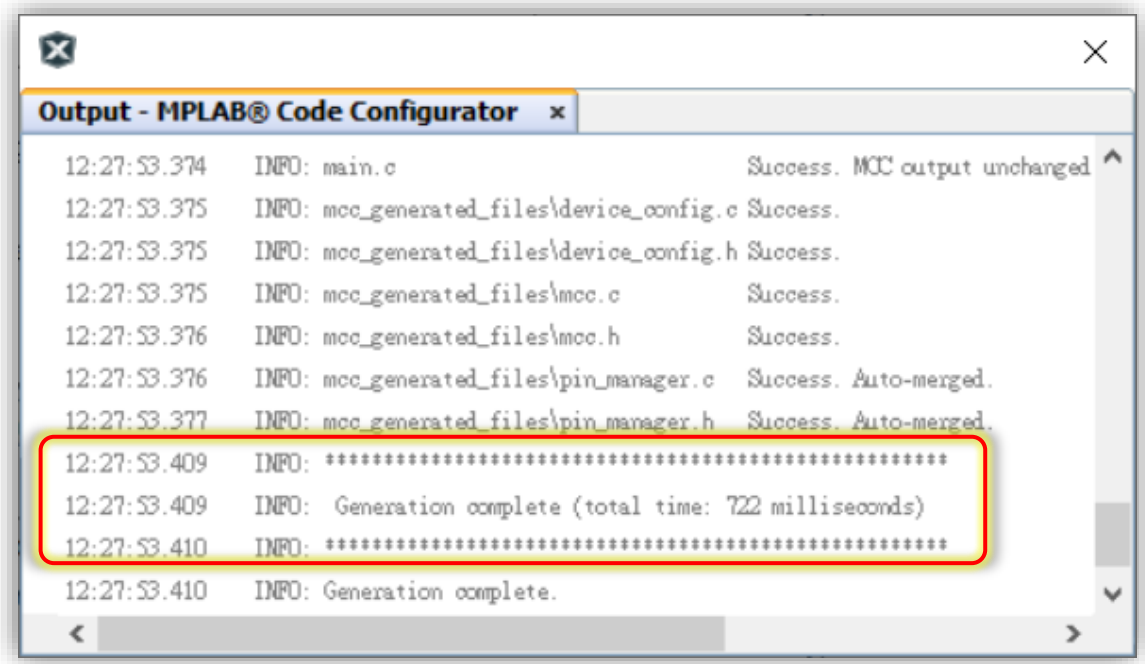
MICROCHIP

Lab7 LCD Display

Step 18

- Click **Generate** Button to generate your code.





Lab7 LCD Display

Code Example

```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/tmr2.h"

void TIM2_ISR(void);
void LCD_Show(uint16_t LCD_Value);

uint16_t ADC_Value=0;
uint8_t Num[4];
volatile uint8_t TMRFlag;

int main(void)
{
   …
   while (1)
   {
      …
   }
}

void TIM2_ISR(void)
{
   TMRFlag=1;
}

void LCD_Show(uint16_t LCD_Value)
{
   …
}
```

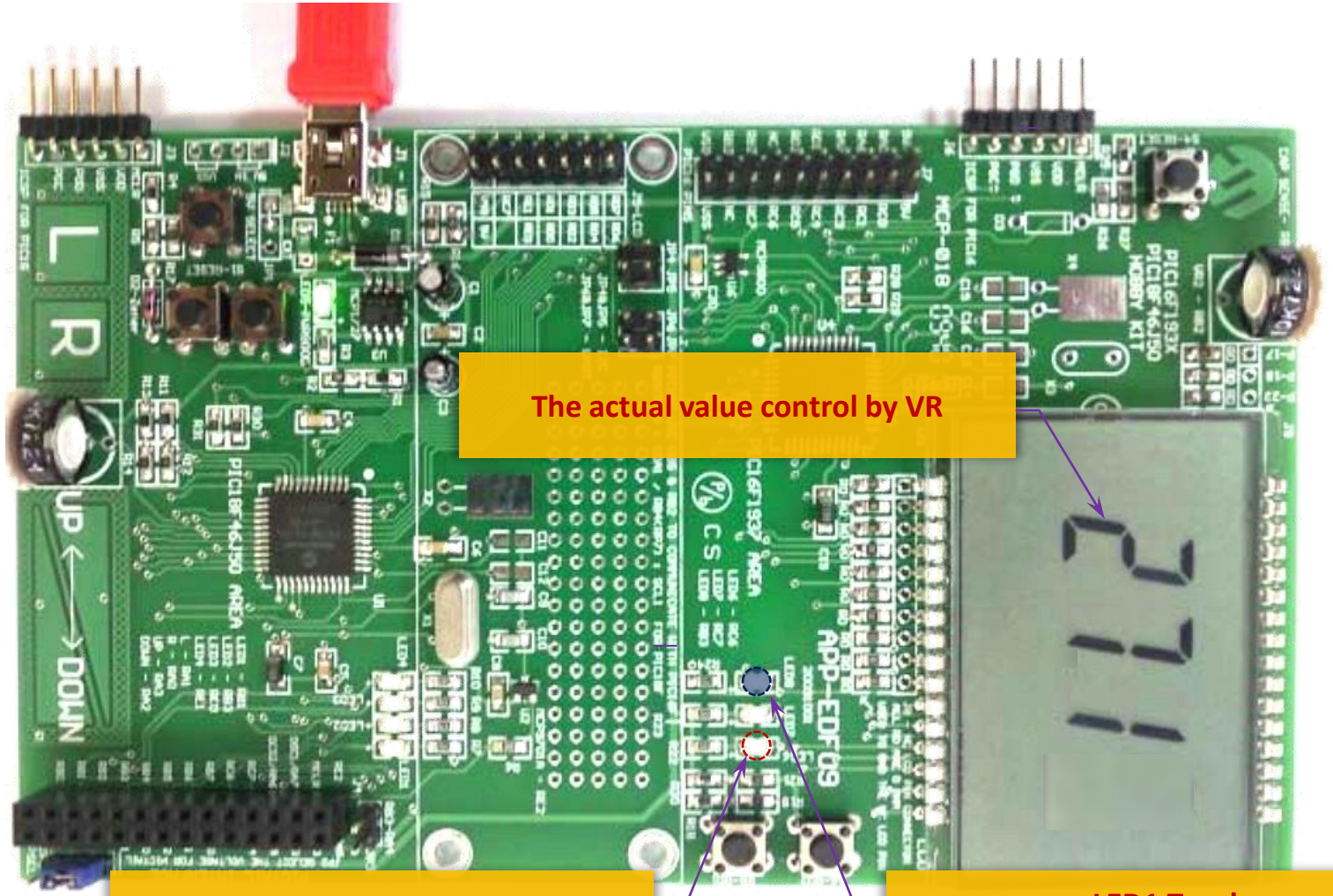
```
void LCD_Show(uint16_t LCD_Value)
{
   Num[0] = (LCD_Value / 1000) % 10;
   Num[1] = (LCD_Value / 100) % 10;
   Num[2] = (LCD_Value / 10) % 10;
   Num[3] = (LCD_Value % 10);
   if (LCD_Value >= 1000)
   {
      LCD_MACRO_SYM00Num(Num[0]);
      LCD_MACRO_SYM01Num(Num[1]);
      LCD_MACRO_SYM02Num(Num[2]);
      LCD_MACRO_SYM03Num(Num[3]);
   }
   else if (LCD_Value >= 100)
   {
      LCD_MACRO_SYM00Num(10);
      LCD_MACRO_SYM01Num(Num[1]);
      LCD_MACRO_SYM02Num(Num[2]);
      LCD_MACRO_SYM03Num(Num[3]);
   }
   else if (LCD_Value >= 10)
   {
      LCD_MACRO_SYM00Num(10);
      LCD_MACRO_SYM01Num(10);
      LCD_MACRO_SYM02Num(Num[2]);
      LCD_MACRO_SYM03Num(Num[3]);
   }
   else
   {
      LCD_MACRO_SYM00Num(10);
      LCD_MACRO_SYM01Num(10);
      LCD_MACRO_SYM02Num(10);
      LCD_MACRO_SYM03Num(Num[3]);
   }
}
```

```
int main(void)
{
   …
   TMR2_SetInterruptHandler(TIM2_ISR);
   while (1)
   {
      // Add your application code

      if(TMRFlag)
      {
         TMRFlag=0;
         LED1_Toggle();
         ADC_Value = ADC_GetConversion(VR);
         LCD_Show(ADC_Value);
      }

      if(!BT2_GetValue())
      {
         LED3_SetLow();
      }
      else
      {
         LED3_SetHigh();
      }
   }
}
```

MICROCHIP

Lab7 LCD Display

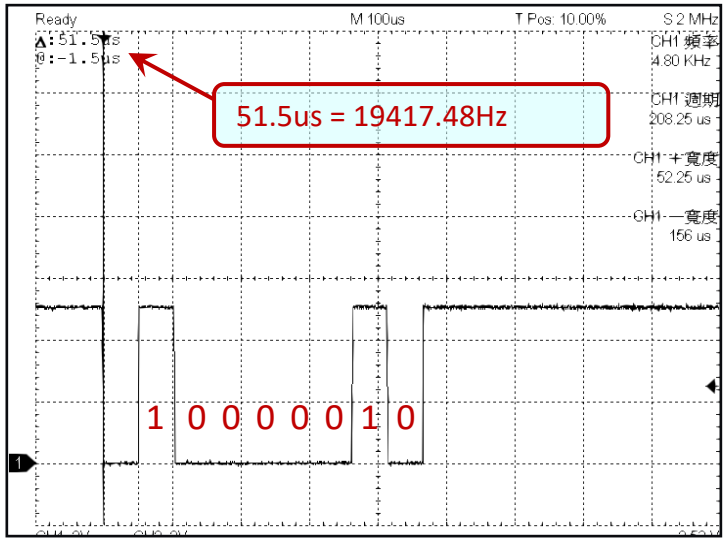
Result



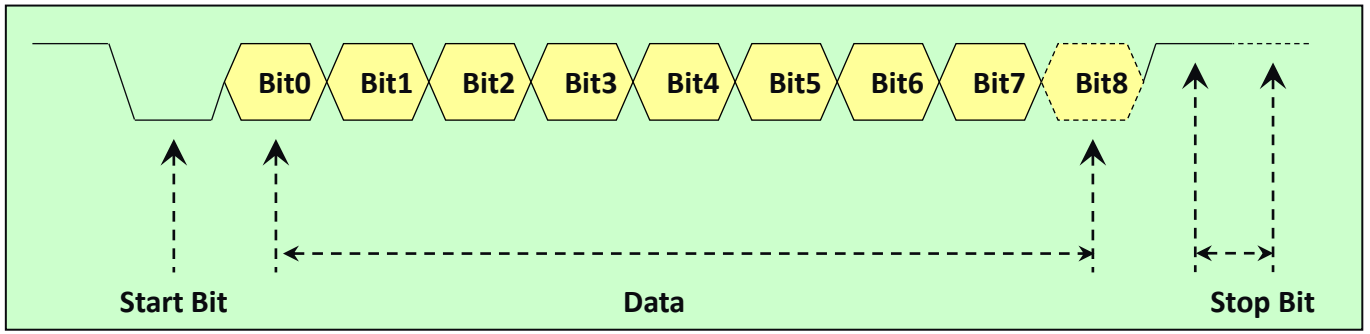
UART Architecture

What's UART

- UART : Universal Asynchronous Receiver Transmitter.
- The module data exchange through serial communication.
- The data formatting includes 1 start bit, 5 ~ 9 data bits and 1 ~ 2 stop bits.



UART Transmit Example :
'A'(0x41), 19,200 bps

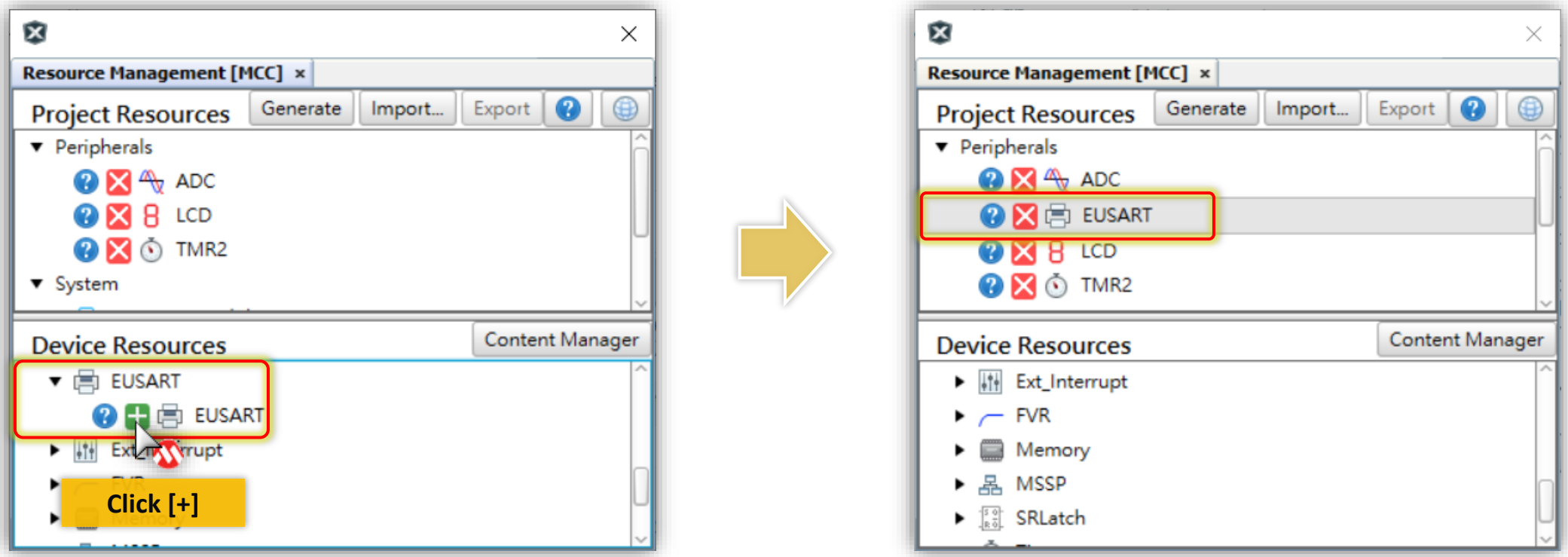


PIC16F Series UART

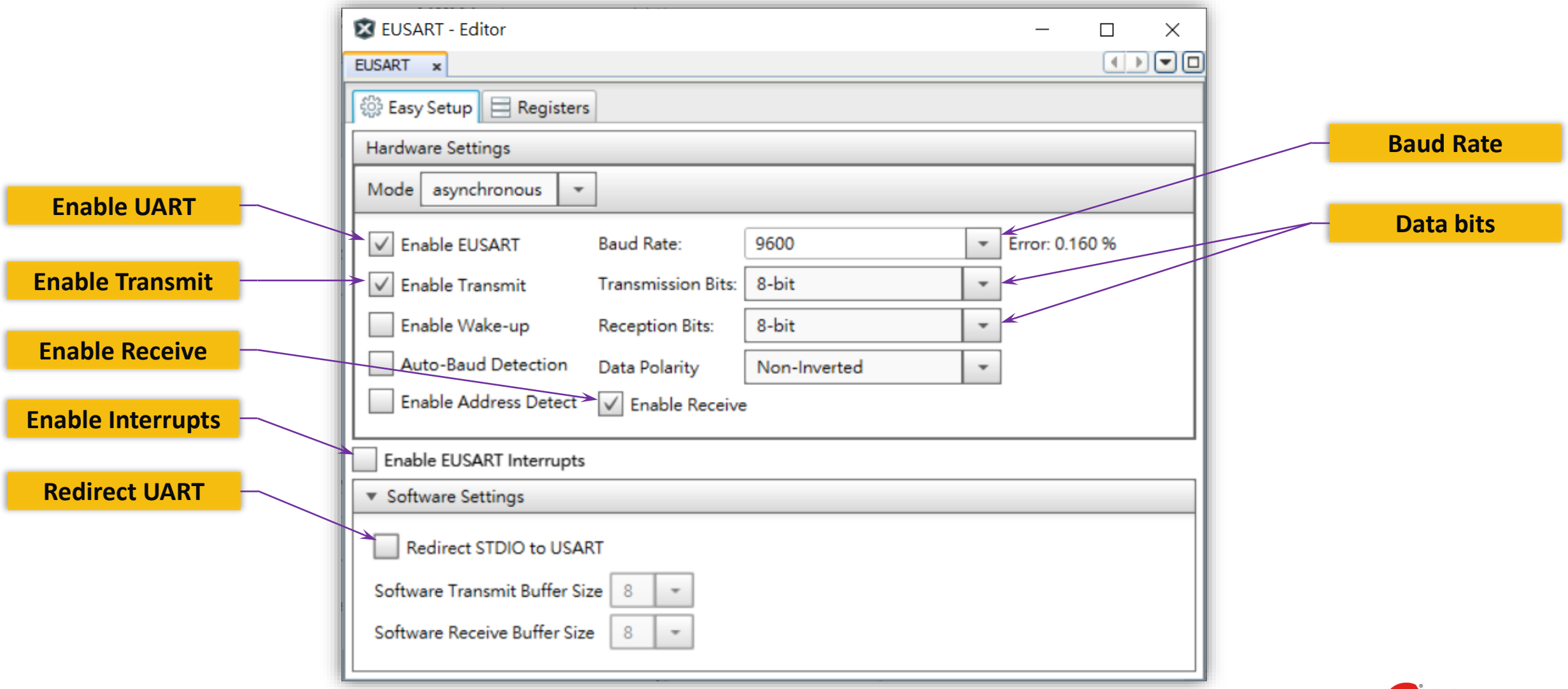
- UART module is full-duplex, asynchronous communication., Support RS-232, RS-485, and LIN.
- The module also supports 8 ～ 9 Data Bits, Half-duplex synchronous master and Slave modes.
- 2-Deep FIFO for RX.
- Provide LIN bus Auto Baud Detect and calibration, 13 Data bit transmit, Wake-up on Break reception.

MCC's UART Module

- **Add UART resource from Device Resources.**



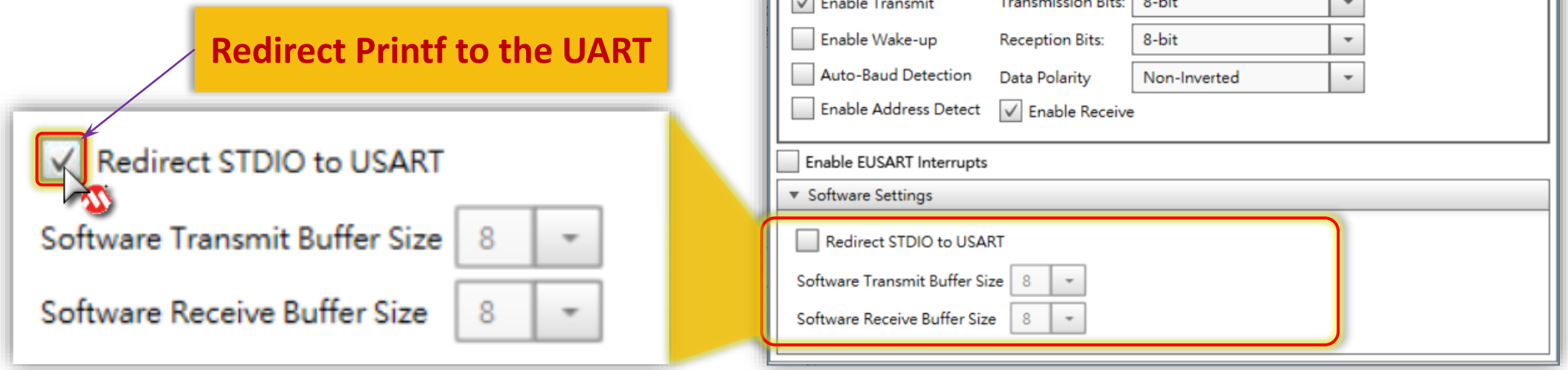
MCC's UART Module



MICROCHIP

STDOUT Redirect

- MCC provide STDOUT redirect function, It can redirect STDOUT to UART.
- You can use printf() to export data by UART, if redirect successfully.
- For Example :
  - UART Transmit Example : printf(" Hello World!! ");



MICROCHIP

Lab Preparation

PC Terminal Software Setting

Lab Preparation

Step 1

- **PL2303TA provides connecting an RS232-like full-duplex asynchronous serial device to any Universal Serial Bus (USB) capable host, include highly compatible drivers could simulate the traditional COM port on most operating systems.**

MICROCHIP

Lab Preparation

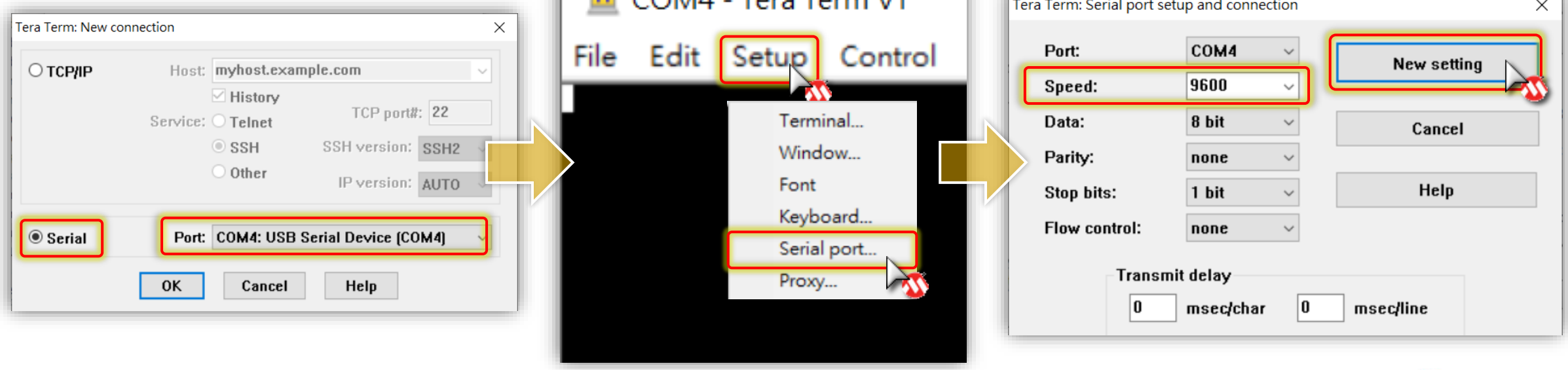
Step 2

- Please confirm Windows Device Manager on your PC.
- For Window 7, you might need install driver before Labs.
  Driver at : \Appendix\Serial Emulator Driver\mchpcdc.inf
- You will see a USB Serial Port (COMx) at Device Manager.

Lab Preparation

Step 3

- Find TeraTerm from your desktop or install it from.
  - http://ttssh2.sourceforge.jp/index.html.en
- Select : Serial → Port: COMx: USB Serial Port (COMx).
- Check : Setup → Serial Port → COMx, 9600, 8 bit, none, 1 bit, none



Lab8 UART printf

- Base on current project or Open .\Lab8_xxx.X to do exercises
- Try to add UART1 module then output VR ADC value .
- UART set to 9600, N, 8, 1, No flow control.
- Disable LED2(RC6) and LED3(RC7) GPIO output function.
- UART has connected by PCB layout.
  - PIC16F1937 (UART-TX, RC6) <-> PL2303TA (RX)
  - PIC16F1937 (UART-RX, RC7) <-> PL2303TA (TX)

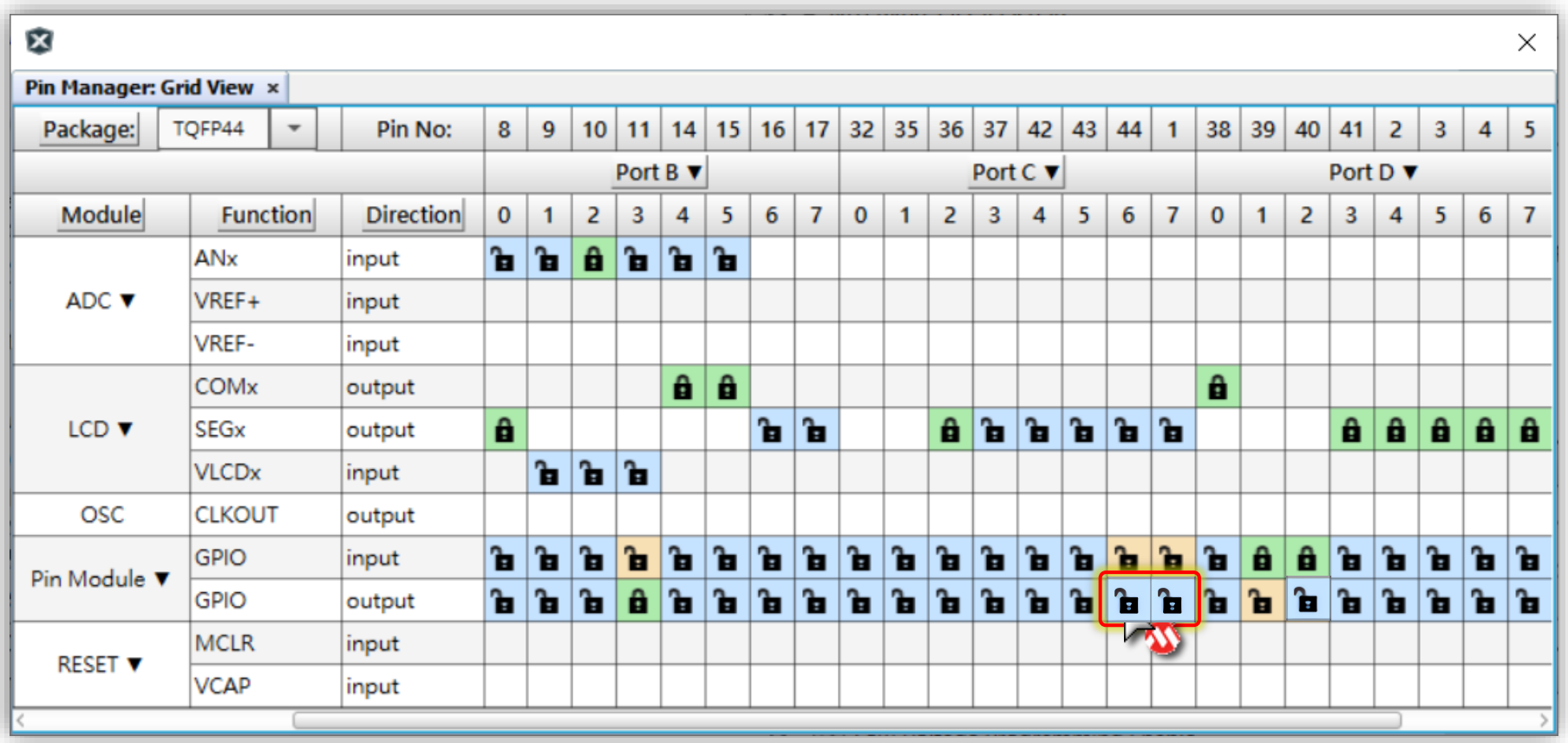
Let’s go!

MICROCHIP

Lab8 UART printf

Step 1

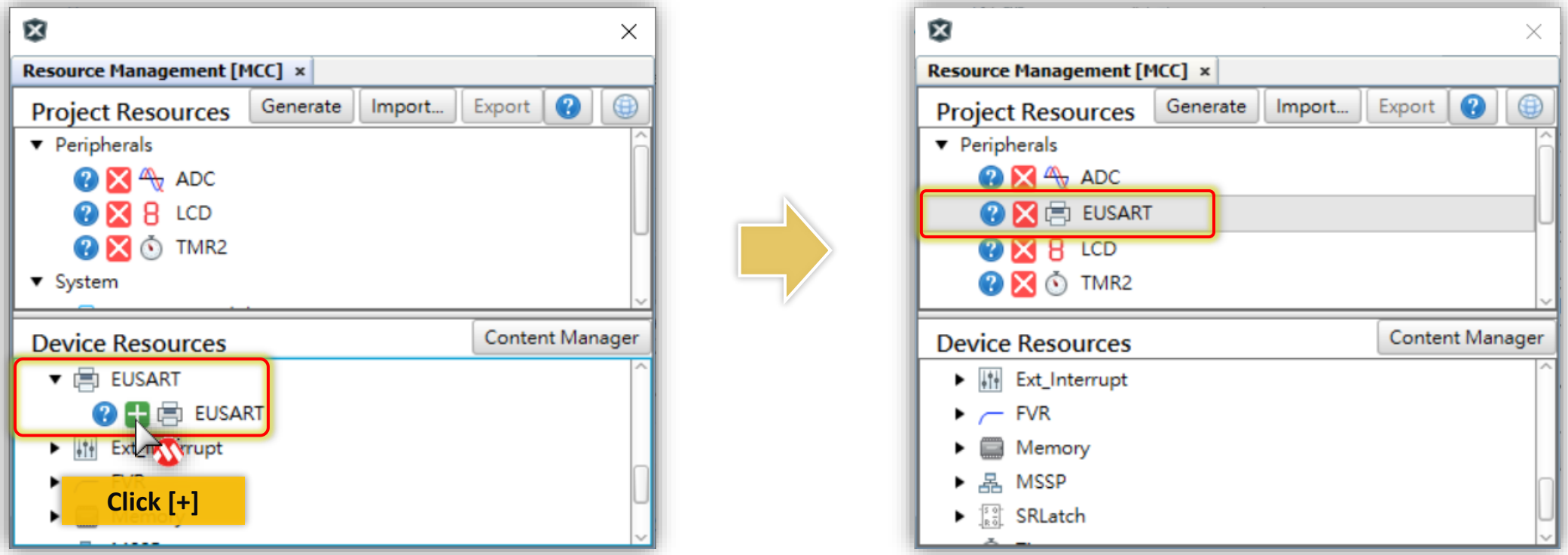
- **Disable RC6, RC7 GPIO output function.**



Lab8 UART printf

Step 2

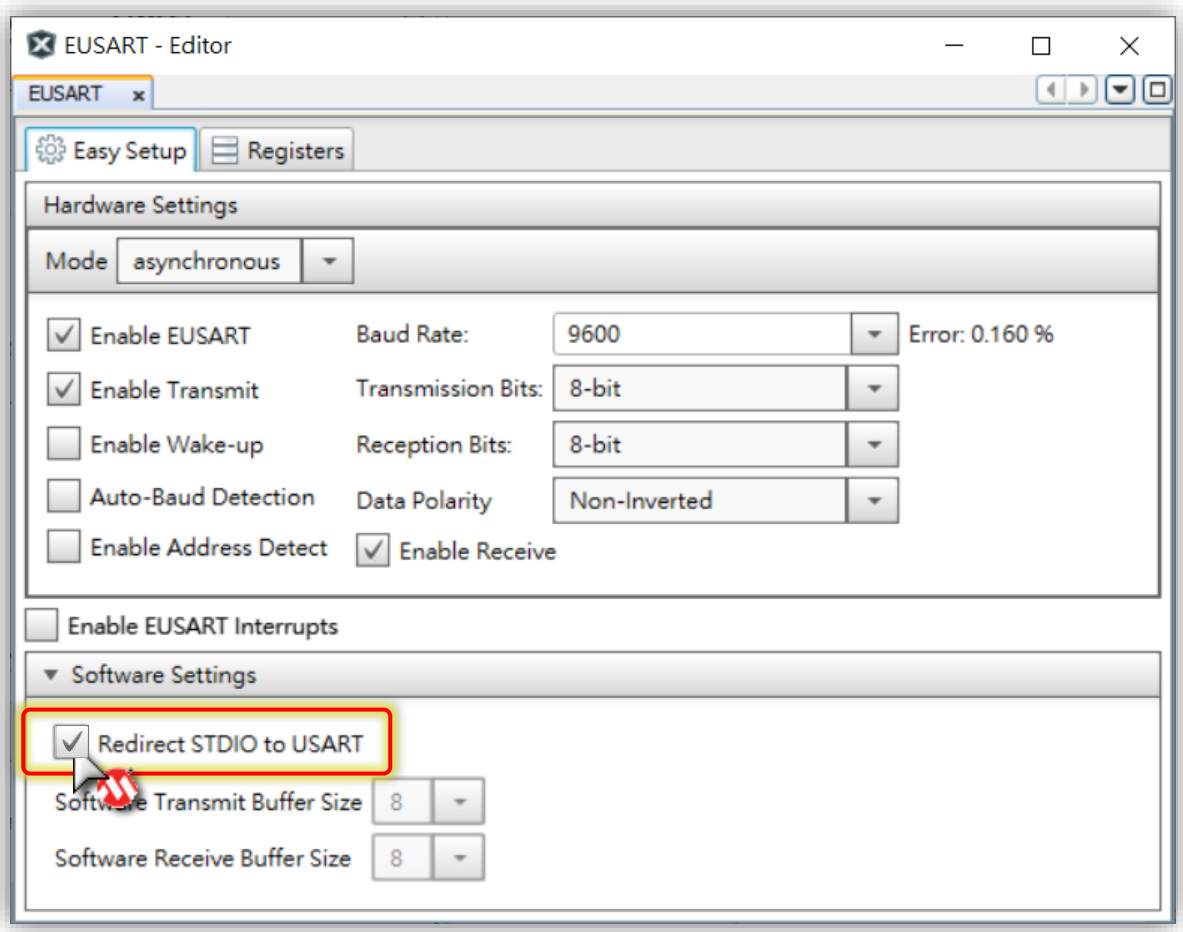
- **Device Resources**
  - **EUSART ▸ EUSART**



Lab8 UART printf

Step 3

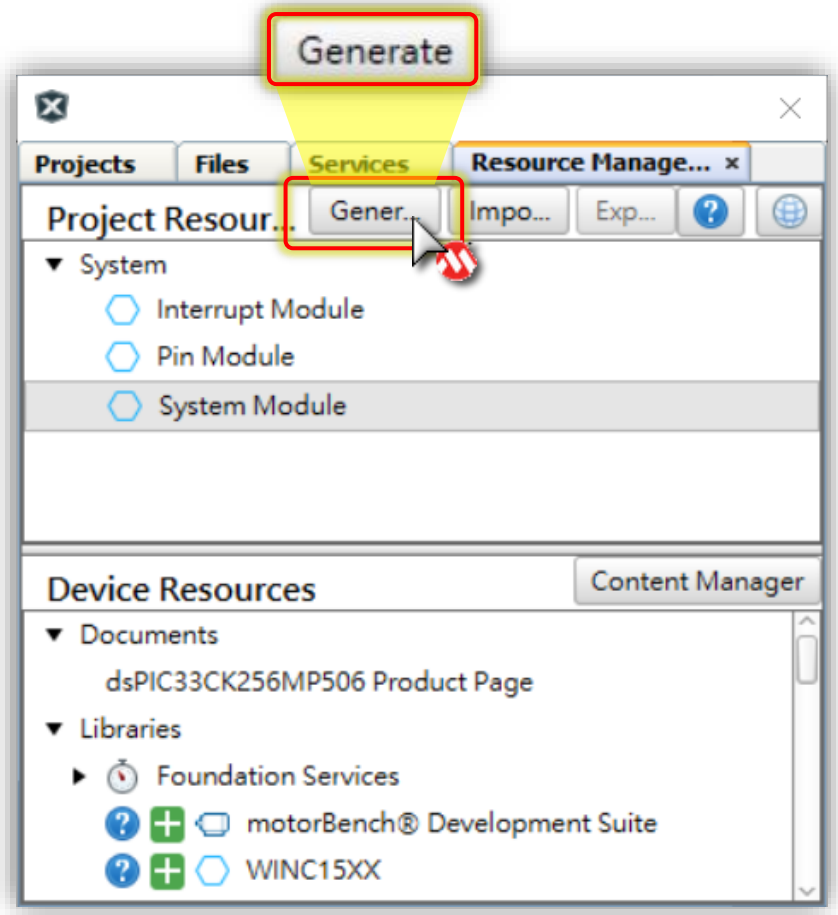
- **Enable Redirect STDIO to USART.**

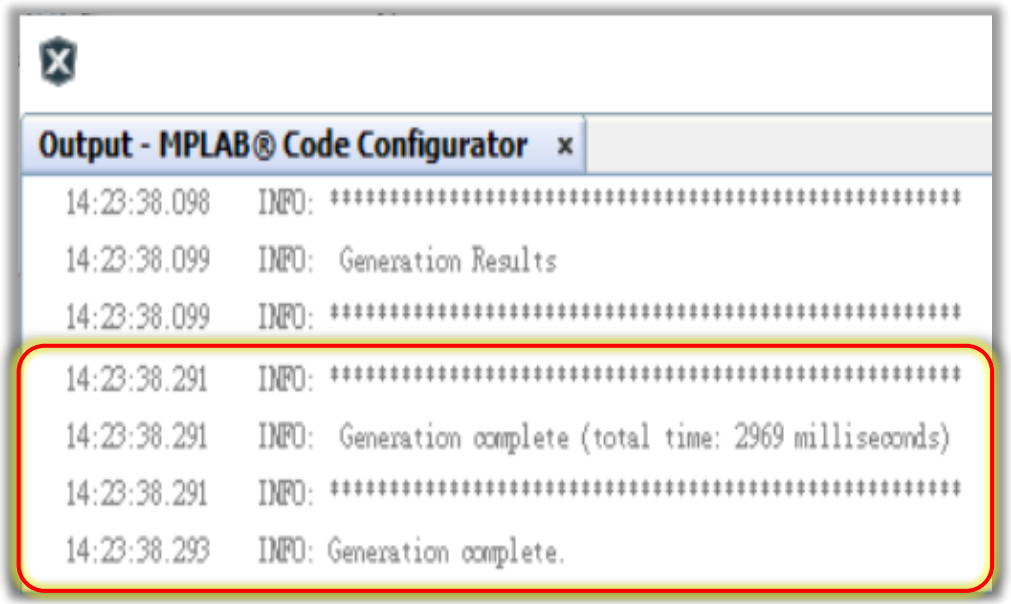


Lab8 UART printf

Step 4

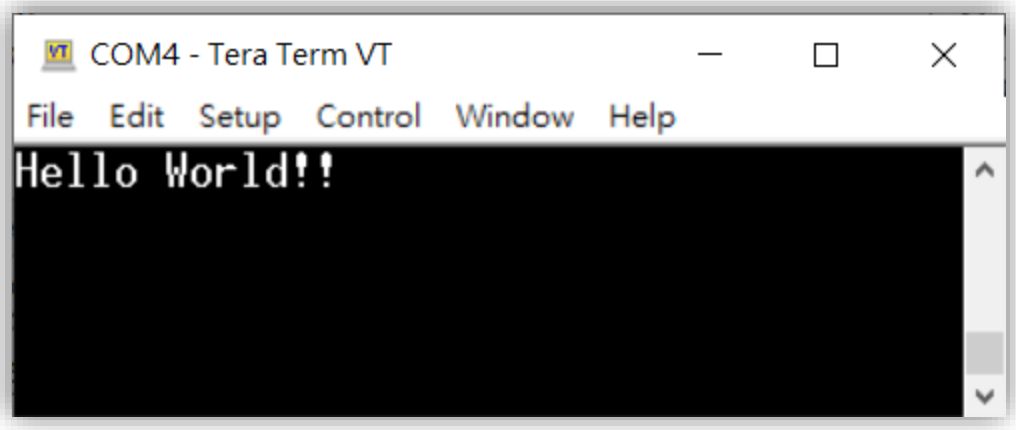
- **Generate Code.**





VT100 Console Control

- Use VT100 command to control UART output as



- printf("\033[2J"); // Clear screen
- printf("\033[?25l"); // Hide cursor
- printf("\033[y;xH"); // Set cursor to (x,y)

Lab8 UART printf

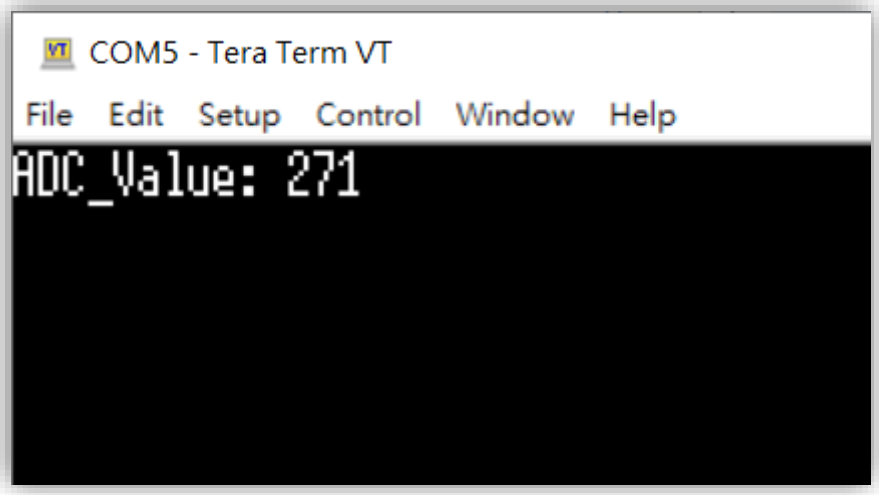
Code Example

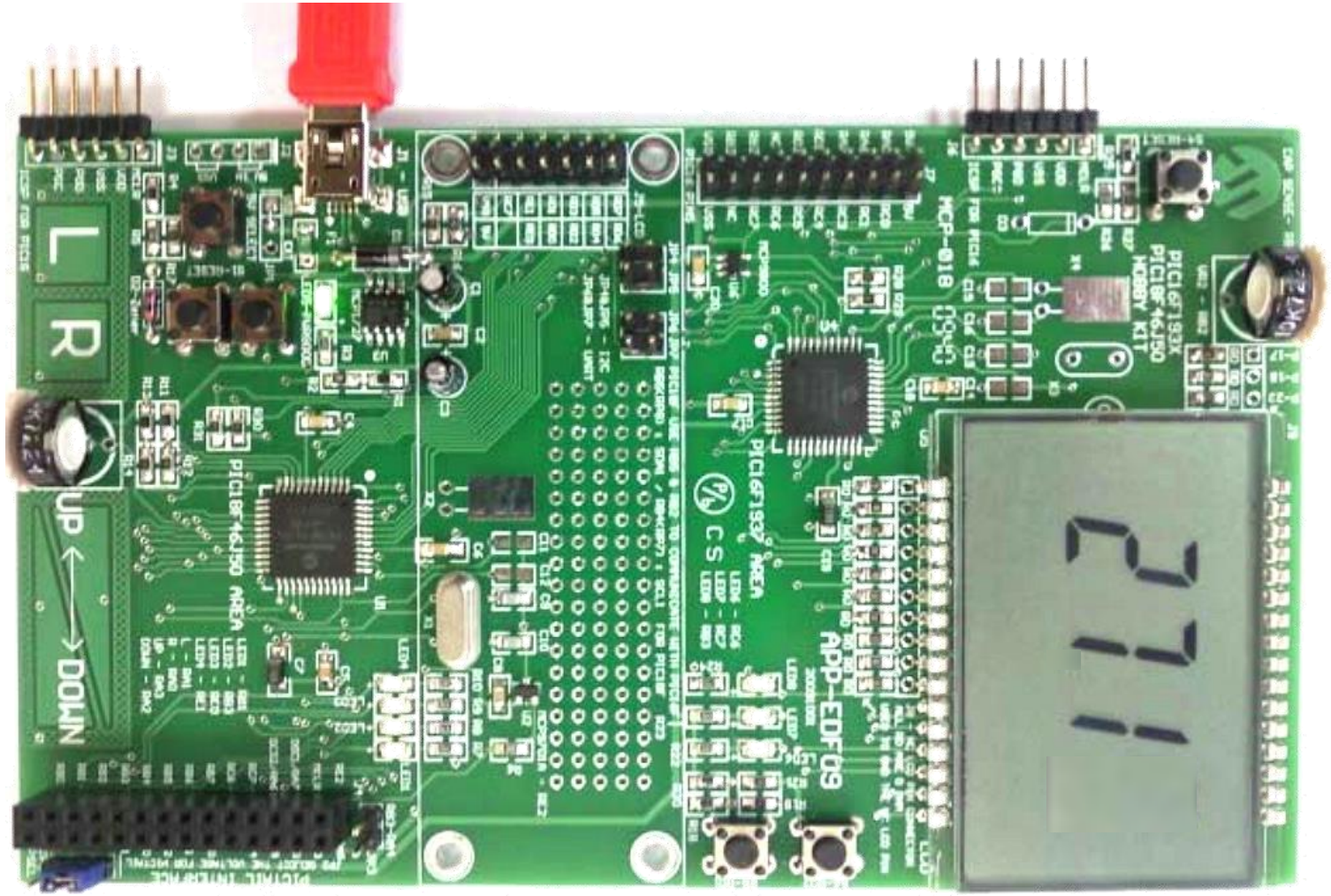
```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/tmr2.h“
#include <stdio.h>

void TIM2_ISR(void);

uint16_t ADC_Value=0;
volatile uint8_t TMRFlag;

int main(void)
{
    …
    while (1)
    {
        …
    }
}

void TIM2_ISR(void)
{
    TMRFlag = 1;
}

void LCD_Show(uint16_t LCD_Value)
{
   …
}
```

```
int main(void)
{
    …
    TMR2_SetInterruptHandler(TIM2_ISR);
    printf("\033[2J");
    printf("\033[?25l");
    while (1)
    {
        // Add your application code

        if (TMRFlag)
        {
            TMRFlag = 0;
            LED1_Toggle();
            ADC_Value = ADC_GetConversion(VR);
            LCD_Show(ADC_Value);
            printf(" \033[1;1HADC_Value:%4d", ADC_Value);
        }
    }
}
```

Lab8 UART printf

Result





MICROCHIP

MCC’s UART Interrupt

- Polling mode not easy to catch that the data from PC to UART.
- This is very easy if enable UART interrupt to be received it.
- Receive and transmit buffer(queue) are implemented ready, so ease to use and don’t worrying data lost.

UART Functions of MCC

- **MCC Provide below functions for UART:**

  Prototype definition : "eusart.h"

  - void EUSART_Initialize(void);
  - bool EUSART_is_tx_ready(void);
  - bool EUSART_is_rx_ready(void);
  - bool EUSART_is_tx_done(void);
  - uint8_t EUSART_Read(void);
  - void EUSART_Write(uint8_t txData);
  - void EUSART_Transmit_ISR(void);
  - void EUSART_Receive_ISR(void);
  - void EUSART_RxDataHandler(void);
  - void EUSART_SetFramingErrorHandler(void (* interruptHandler)(void));
  - void EUSART_SetOverrunErrorHandler(void (* interruptHandler)(void));
  - void EUSART_SetErrorHandler(void (* interruptHandler)(void));

Lab9 UART Communication

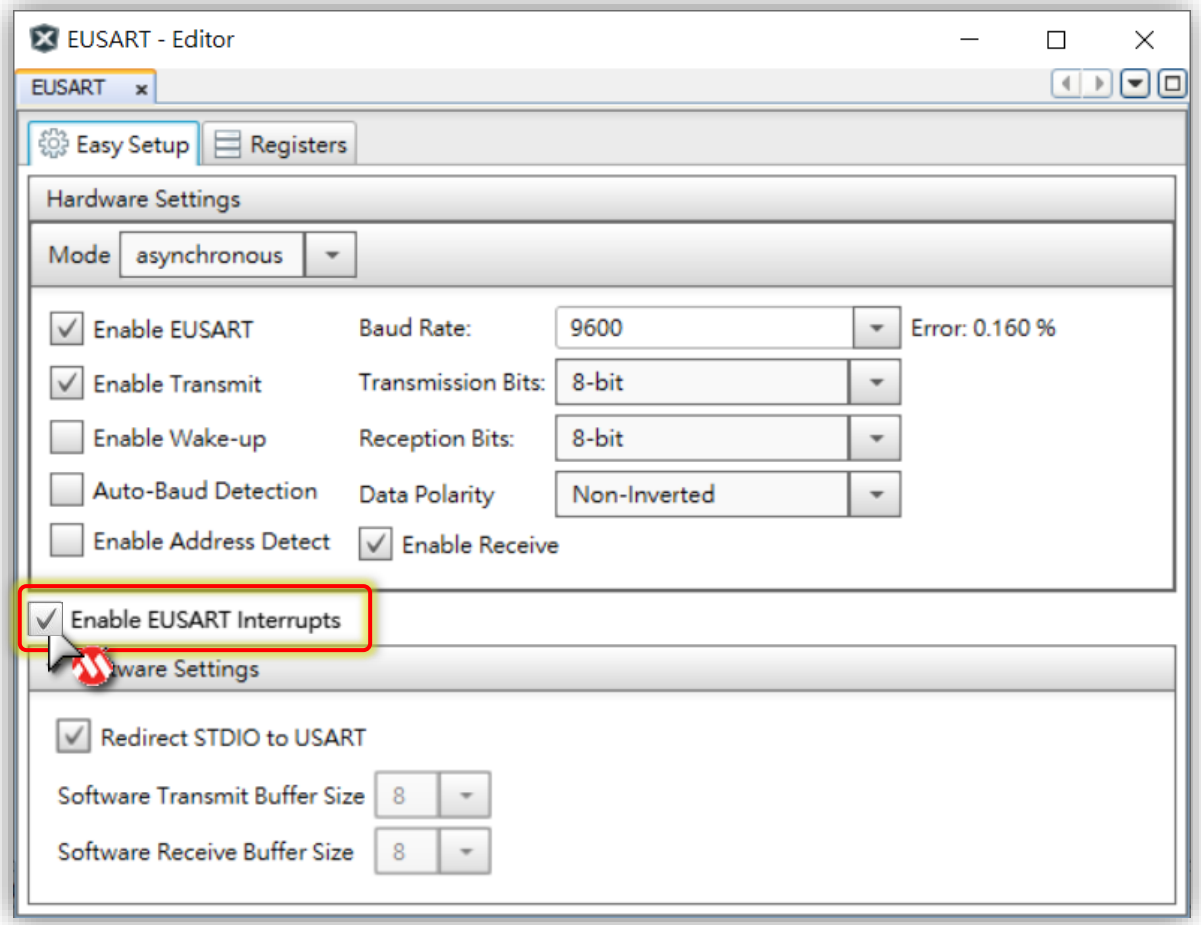
- Base on current project or Open .\Lab9_xxx.X to do exercises
- Try to enable UART interrupt to be received UART data come from PC.
- To set a suitable received buffer size.

Let’s go!

Lab9 UART Communication

Step 1

- **Enable EUSART Interrupts.**

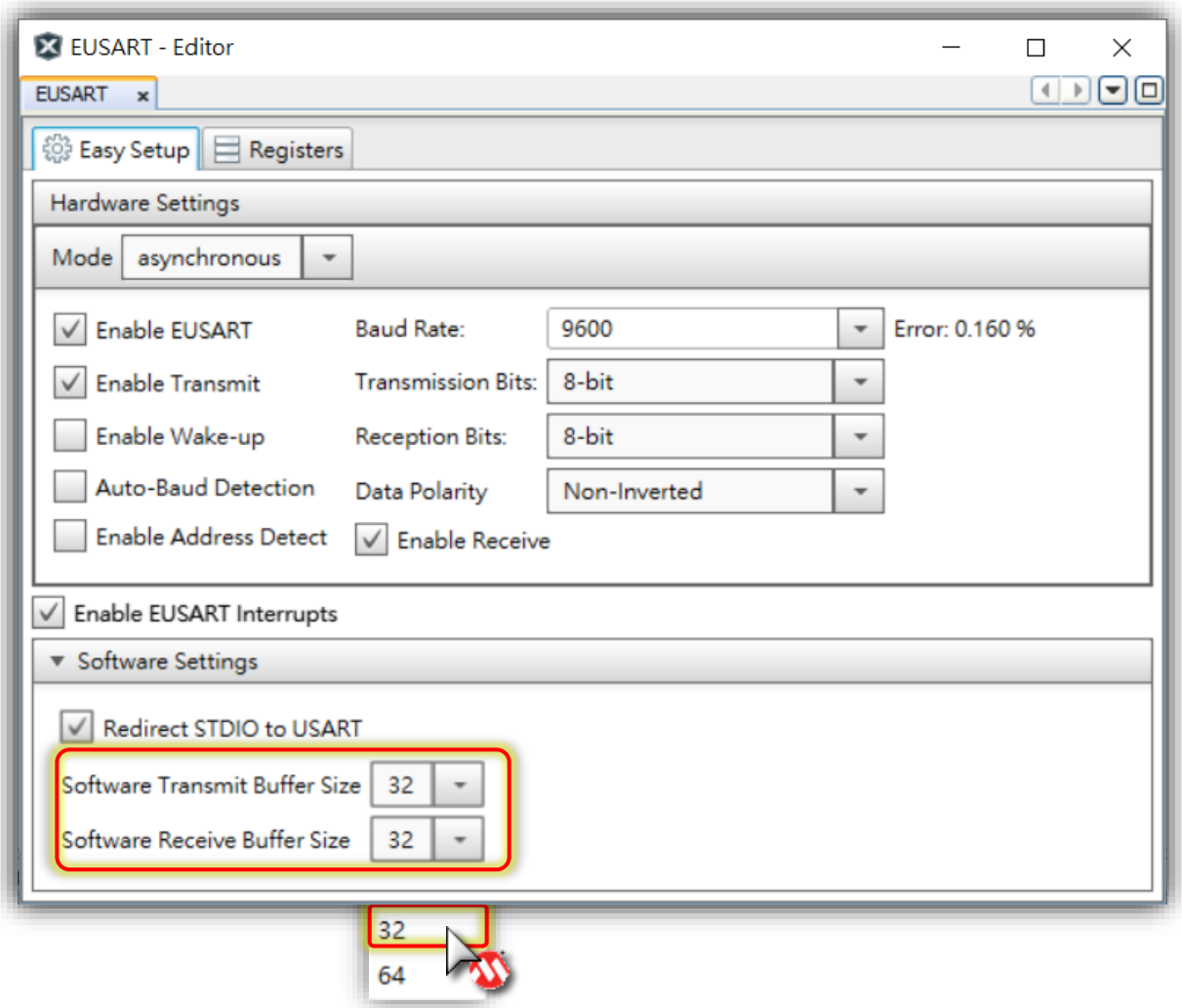


MICROCHIP

Lab9 UART Communication

Step 2

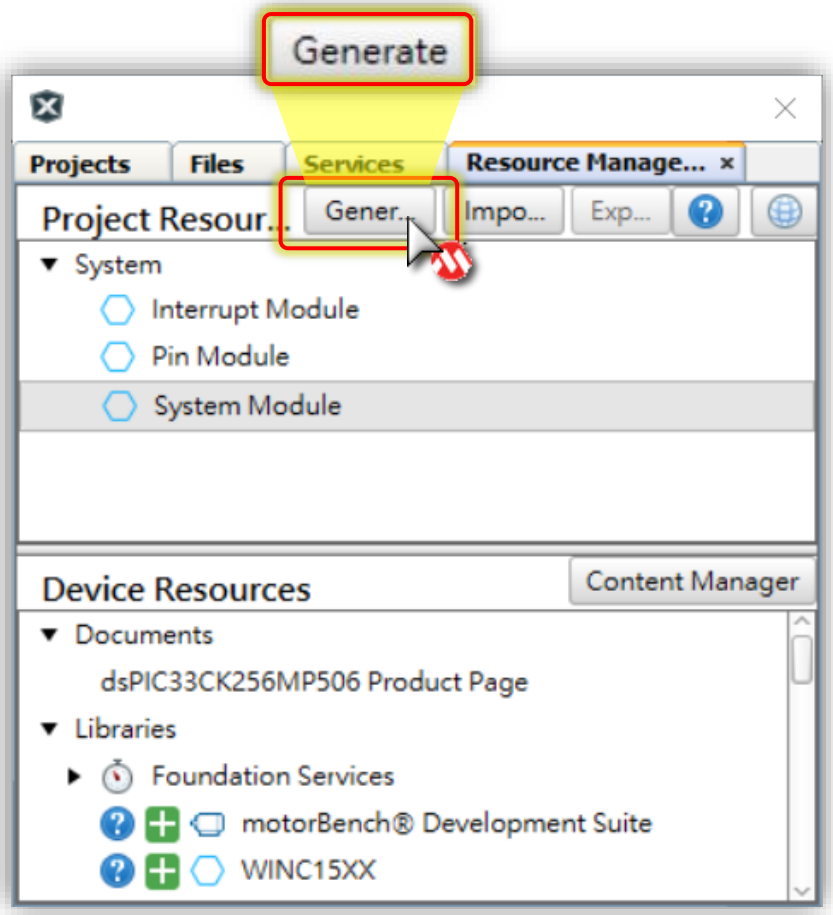
- **Buffer Size.**
  - **8 ▸ 32**

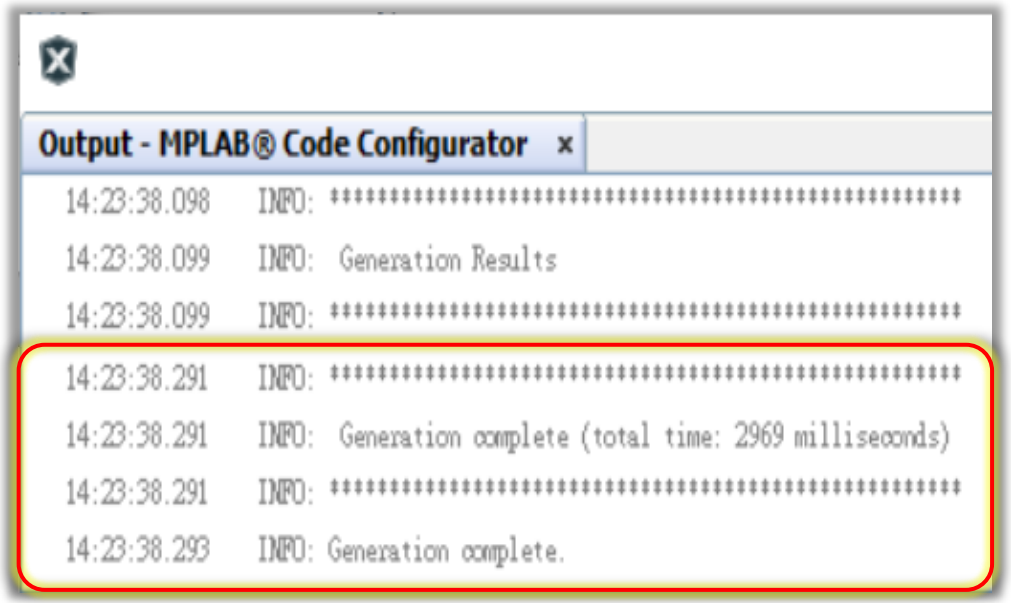


Lab9 UART Communication

Step 3

- **Generate Code.**





Lab9 UART Communication

Code Example

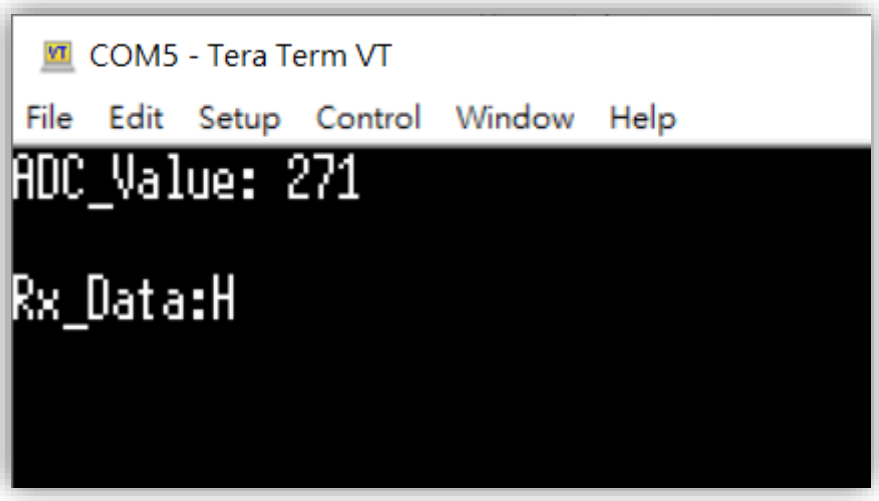
```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/tmr2.h“
#include <stdio.h>

void TIM2_ISR(void);

uint16_t ADC_Value=0;
volatile uint8_t TMRFlag;
uint8_t Rx_Buffer[1];

int main(void)
{
   …
   while (1)
   {
      …
   }
}

void TIM2_ISR(void)
{
   TMRFlag = 1;
}

void LCD_Show(uint16_t LCD_Value)
{
  …
}
```

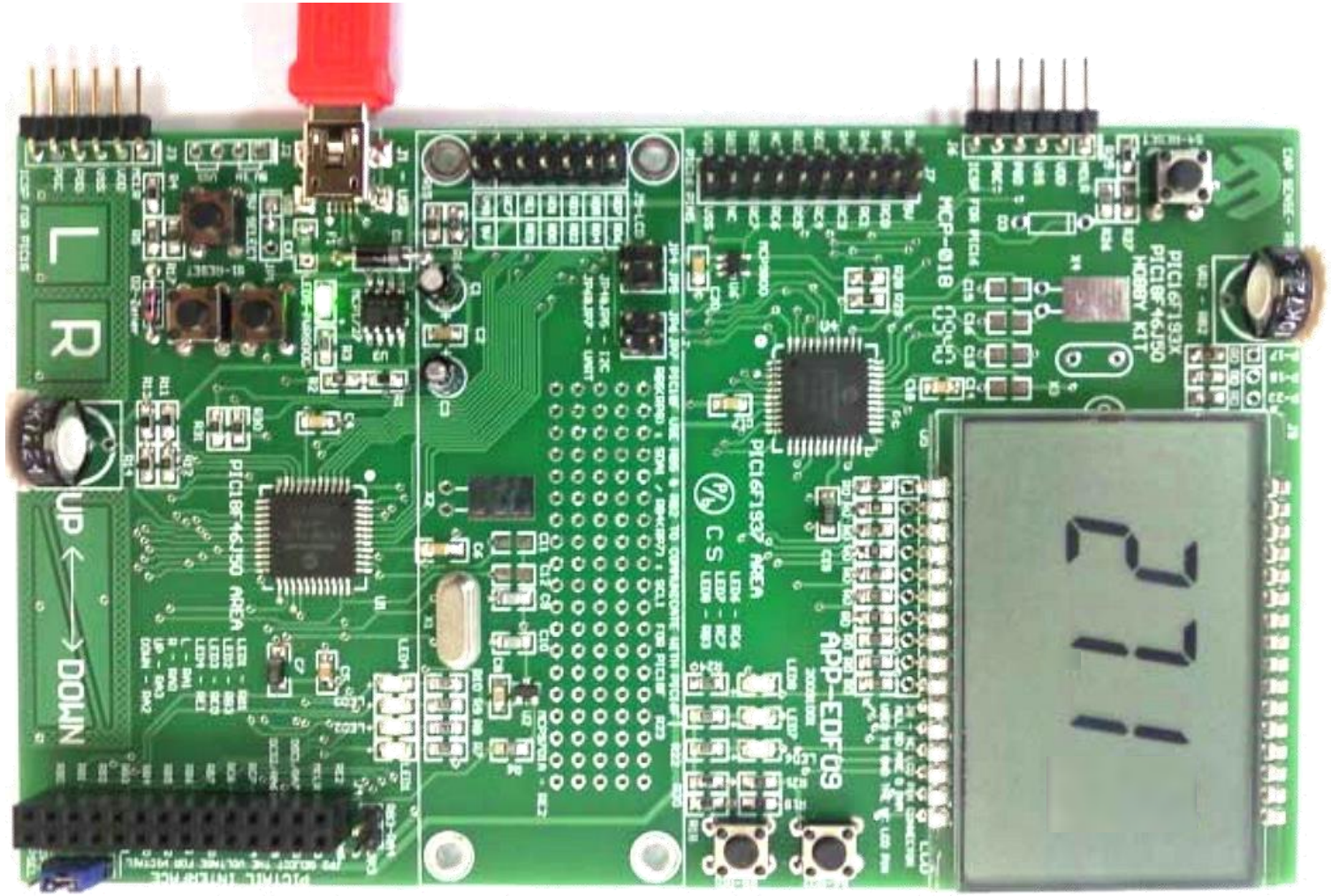
```
int main(void)
{
   ...
   TMR2_SetInterruptHandler(TIM2_ISR);
   printf("\033[2J");
   printf("\033[?25l");
   while (1)
   {
      // Add your application code

      if (TMRFlag)
      {
         TMRFlag = 0;
         LED1_Toggle();
         ADC_Value = ADC_GetConversion(VR);
         LCD_Show(ADC_Value);
         printf(" \033[1;1HADC_Value:%4d", ADC_Value);
      }
      if (EUSART_is_rx_ready())
      {
         Rx_Buffer[0] = EUSART_Read();
         printf(" \033[3;1HRx_Data:%1c", Rx_Buffer[0]);
      }
   }
}
```

MICROCHIP

Lab9 UART Communication

Result



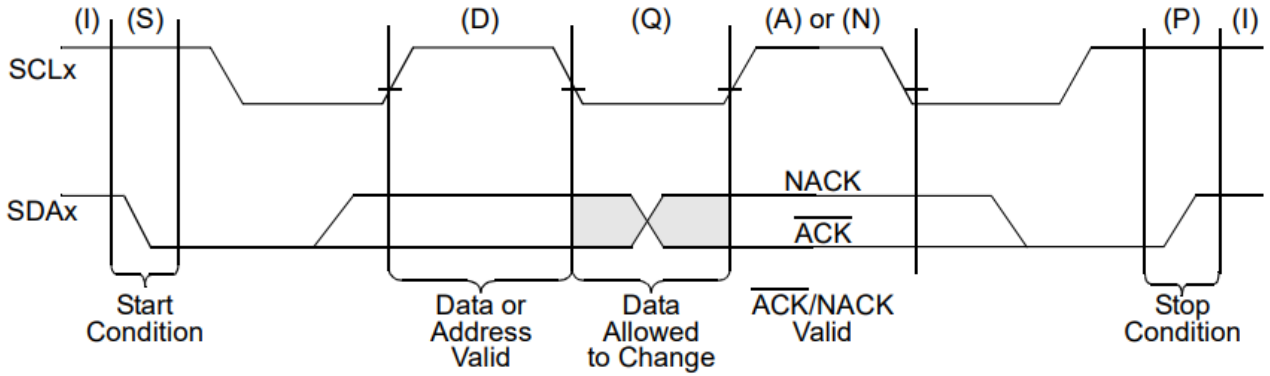


MICROCHIP

I2C Architecture

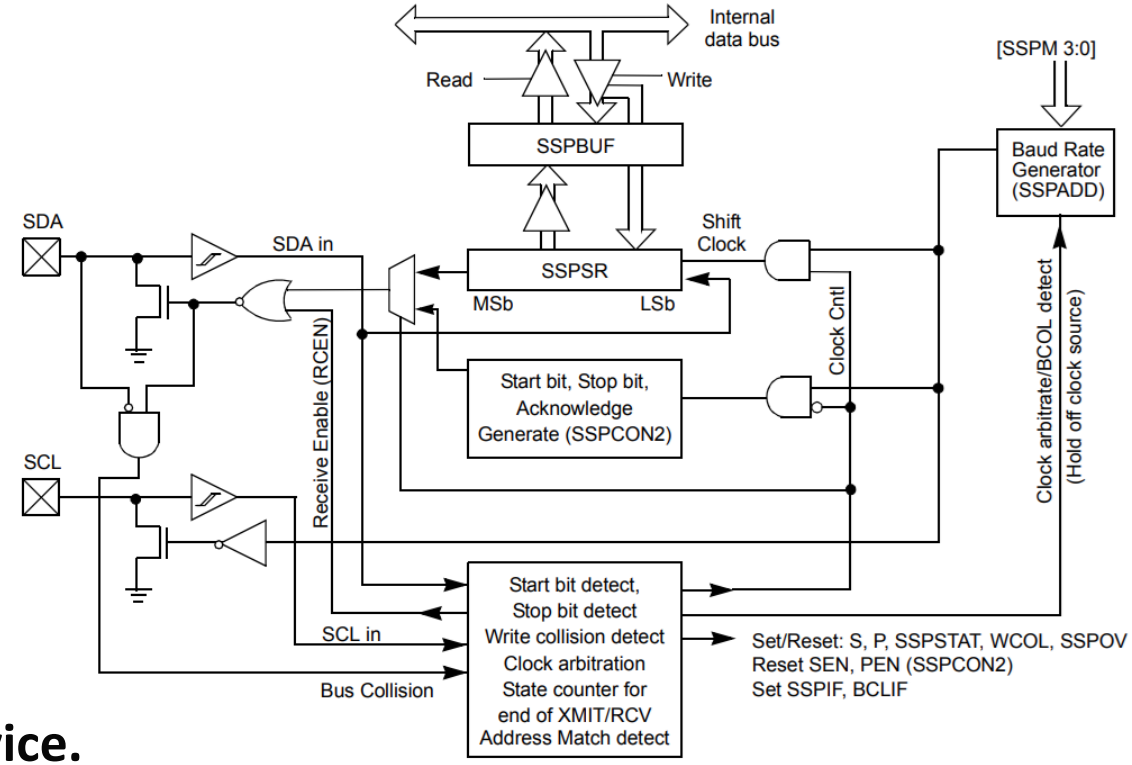
What's I2C

- I2C, Inter-Integrated Circuit
- It's a serial communication interface.
  Synchronous, inside-board level, short distance communication.
- The data is clocked along with a clock signal(SCL).
- The clock signal controls when data is changed and when it should be read.
- Since I2C is synchronous, the clock rate can vary, unlike asynchronous (RS-232 style) communications.



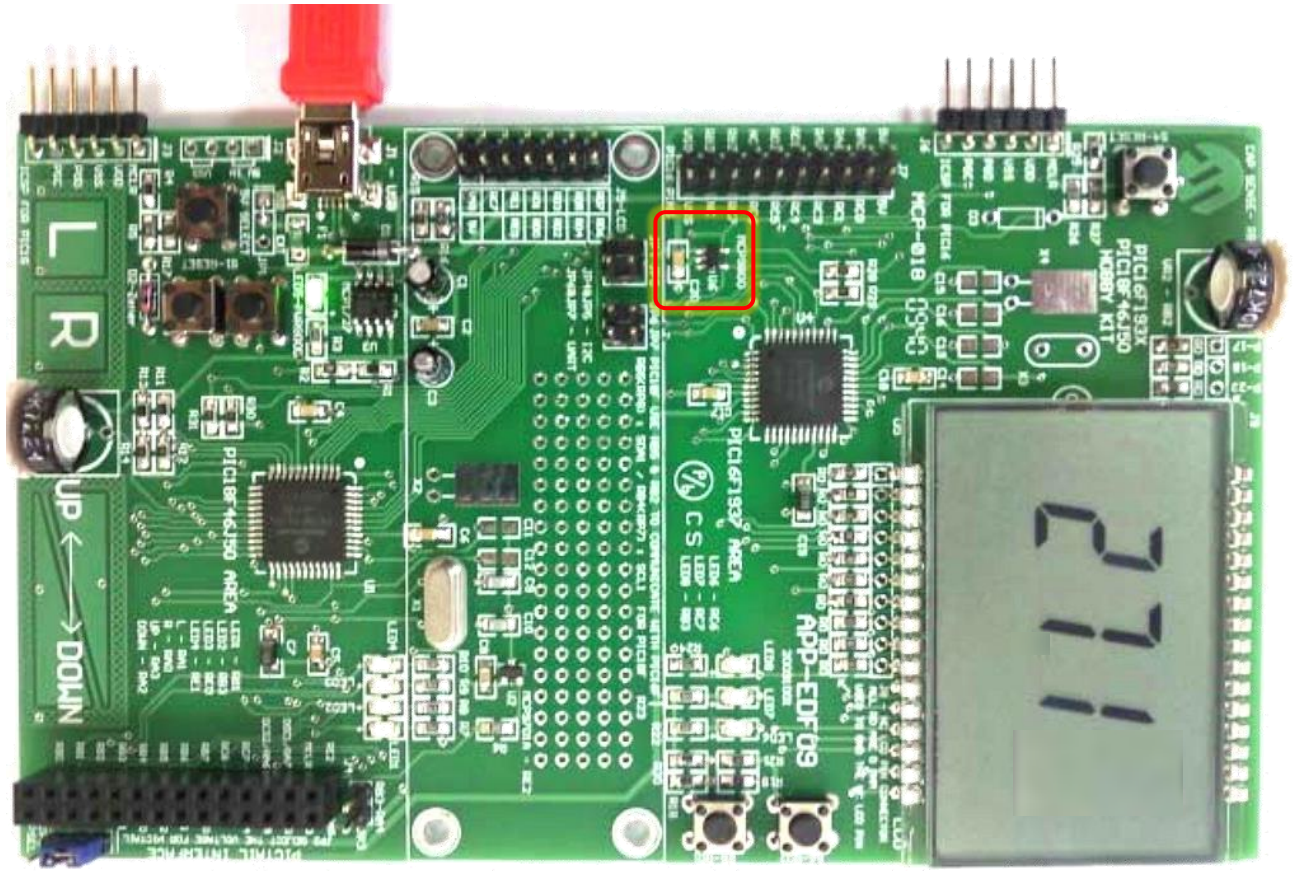
PIC16F Series MSSP Module

- **The Master Synchronous Serial Port (MSSP) module can operate in one of two modes:**
  - Serial Peripheral Interface (SPI).
  - **Inter-Integrated Circuit (I2C).**
- **Serial Peripheral Interface (SPI)**
  - Master and Slave mode.
  - Clock Parity.
  - Slave Select Synchronization (Slave mode only).
- **Inter-Integrated Circuit (I2)**
  - Master and Slave mode.
  - Support limited Multi-Master.
  - Support 7-bit and 10-bit addressing.
- **Pin Definition :**
  - **SCL (Serial Clock) : Clock, PIC16F -> Device.**
  - **SDA (Serial Data) : Data In/Out, PIC16F <-> Device.**



I2C Device

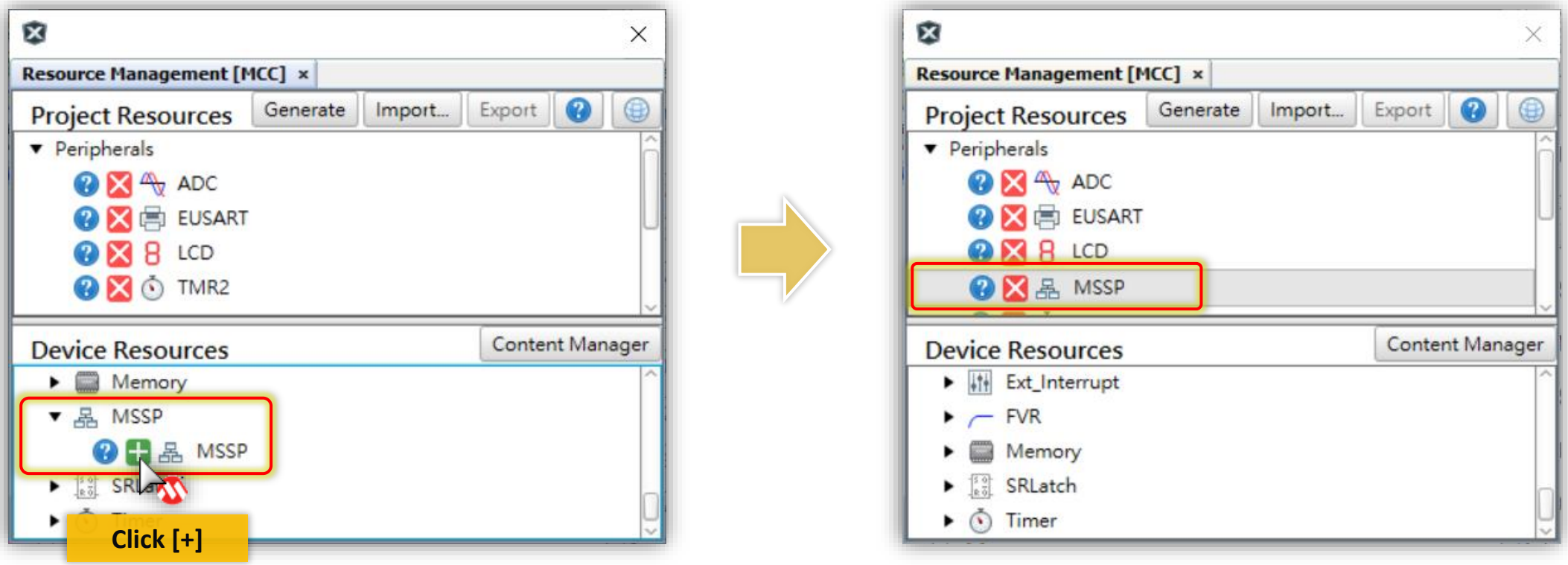
- **Temperature Sensor (MCP9800)：**
  - **MCP9800 is a digital temperature sensor converted to digital word with a user selectable 9 to 12-bit Sigma-Delta Analog to Digital Converter. ±0.5°C (typical) at +25°C**



MICROCHIP

MCC's MSSP Module

- **Add MSSP resource from Device Resources.**



MCC's MSSP Module

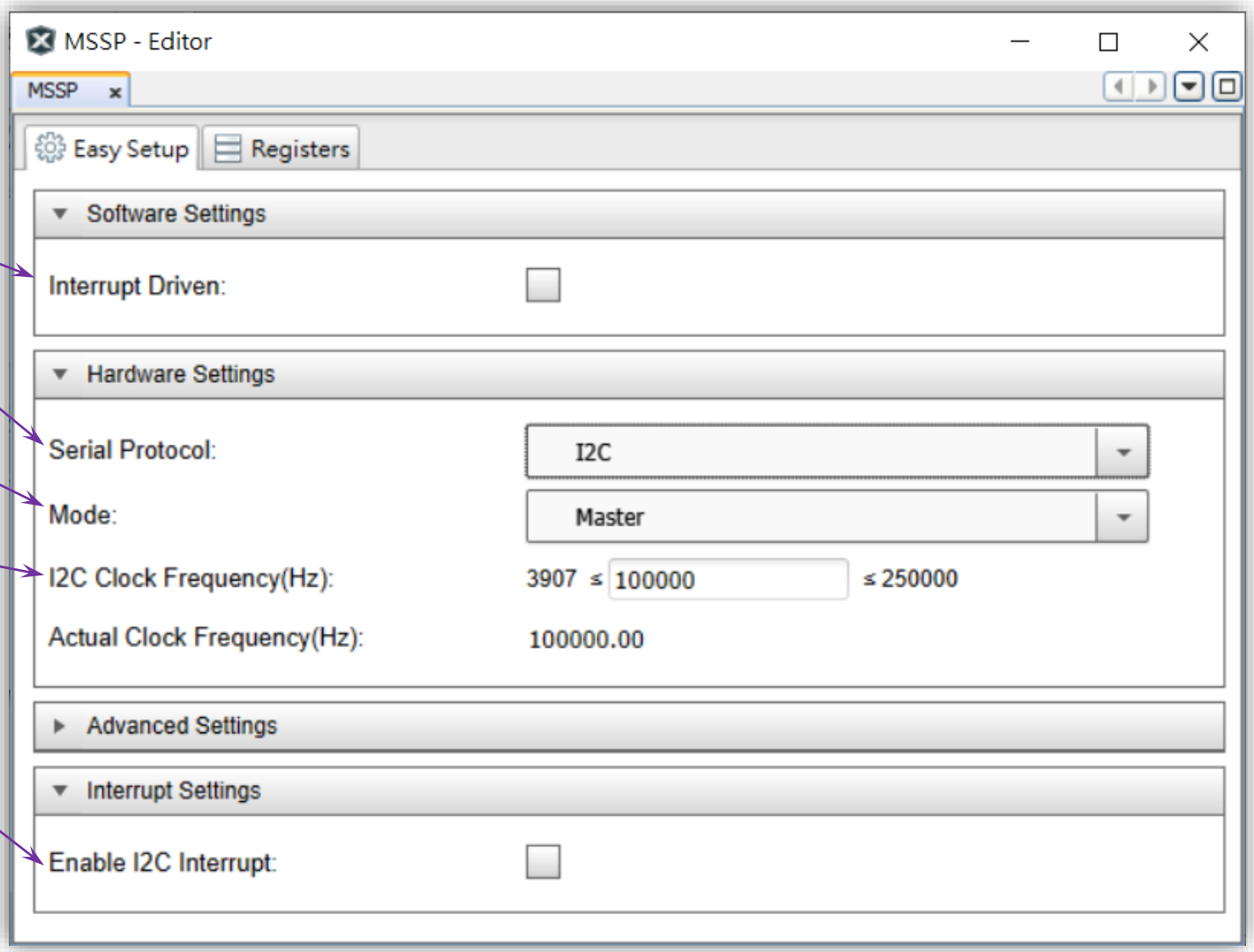
Enable Interrupt

Select Protocol

Select Mode

Clock Frequency

Enable Interrupt



MICROCHIP

I2C Functions of MCC

- **MCC Provide below functions for I2C:**

  Prototype definition : "i2c_master_example.h"

  - uint8_t I2C_Read1ByteRegister(i2c_address_t address, uint8_t reg);
  - uint16_t I2C_Read2ByteRegister(i2c_address_t address, uint8 _t reg);
  - void I2C_Write1ByteRegister(i2c_address_t address, uint8_t reg, uint8_t data);
  - void I2C_Write2ByteRegister(i2c_address_t address, uint8_t reg, uint16_t data);
  - void I2C_WriteNBytes(i2c_address_t address, uint8_t *data, size_t len);
  - void I2C_ReadNBytes(i2c_address_t address, uint8_t *data, size_t len);
  - void I2C_ReadDataBlock(i2c_address_t address, uint8_t reg, uint8_t *data, size_t len);

Lab10 Temperature Sensor

- Base on current project or Open .\Lab10_xxx.X to do exercises
- Try to set MCP9800 ADC resolution to 12 bits.
- Read the value from Temperature Sensor and use UART to output to PC.

Let’s go!

MCP9800 Read & Configuration

- First, Set ADC resolution & resolution & other config you want.
- Secondly, Read ADC value maximum valid 12 bits. (2 Bytes).

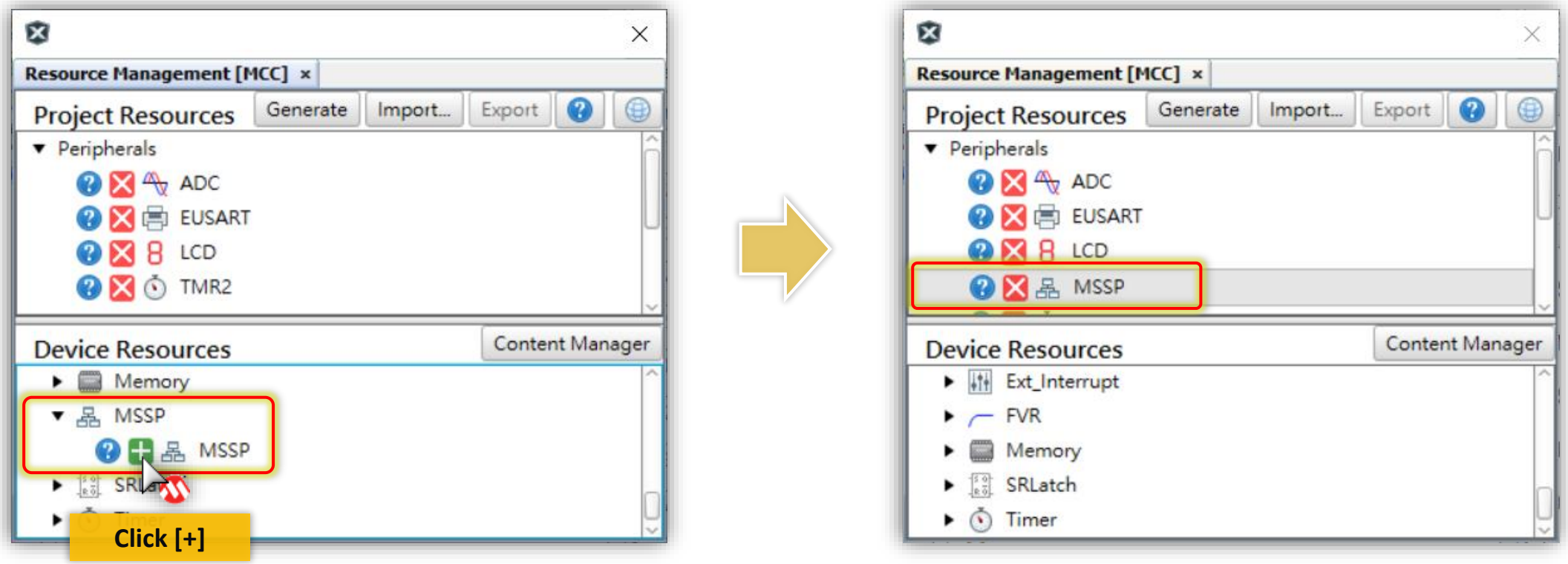
TABLE 5-1: BIT ASSIGNMENT SUMMARY FOR ALL REGISTERS

Register Pointer P1 P0	MSB/ LSB	Bit Assignment							
		7	6	5	4	3	2	1	0
Ambient Temperature Register (T_A)									
0 0	MSB	Sign	2^6°C	2^5°C	2^4°C	2^3°C	2^2°C	2^1°C	2^0°C
	LSB	2^{-1}°C	2^{-2}°C	2^{-3}°C	2^{-4}°C	0	0	0	0
Sensor Configuration Register (CONFIG)									
0 1	LSB	One-Shot	Resolution		Fault Queue		ALERT Polarity	COMP/INT	Shutdown
Temperature Hysteresis Register (T_{HYST})									
1 0	MSB	Sign	2^6°C	2^5°C	2^4°C	2^3°C	2^2°C	2^1°C	2^0°C
	LSB	2^{-1}°C	0	0	0	0	0	0	0
Temperature Limit-Set Register (T_{SET})									
1 1	MSB	Sign	2^6°C	2^5°C	2^4°C	2^3°C	2^2°C	2^1°C	2^0°C
	LSB	2^{-1}°C	0	0	0	0	0	0	0

Lab10 Temperature Sensor

Step 1

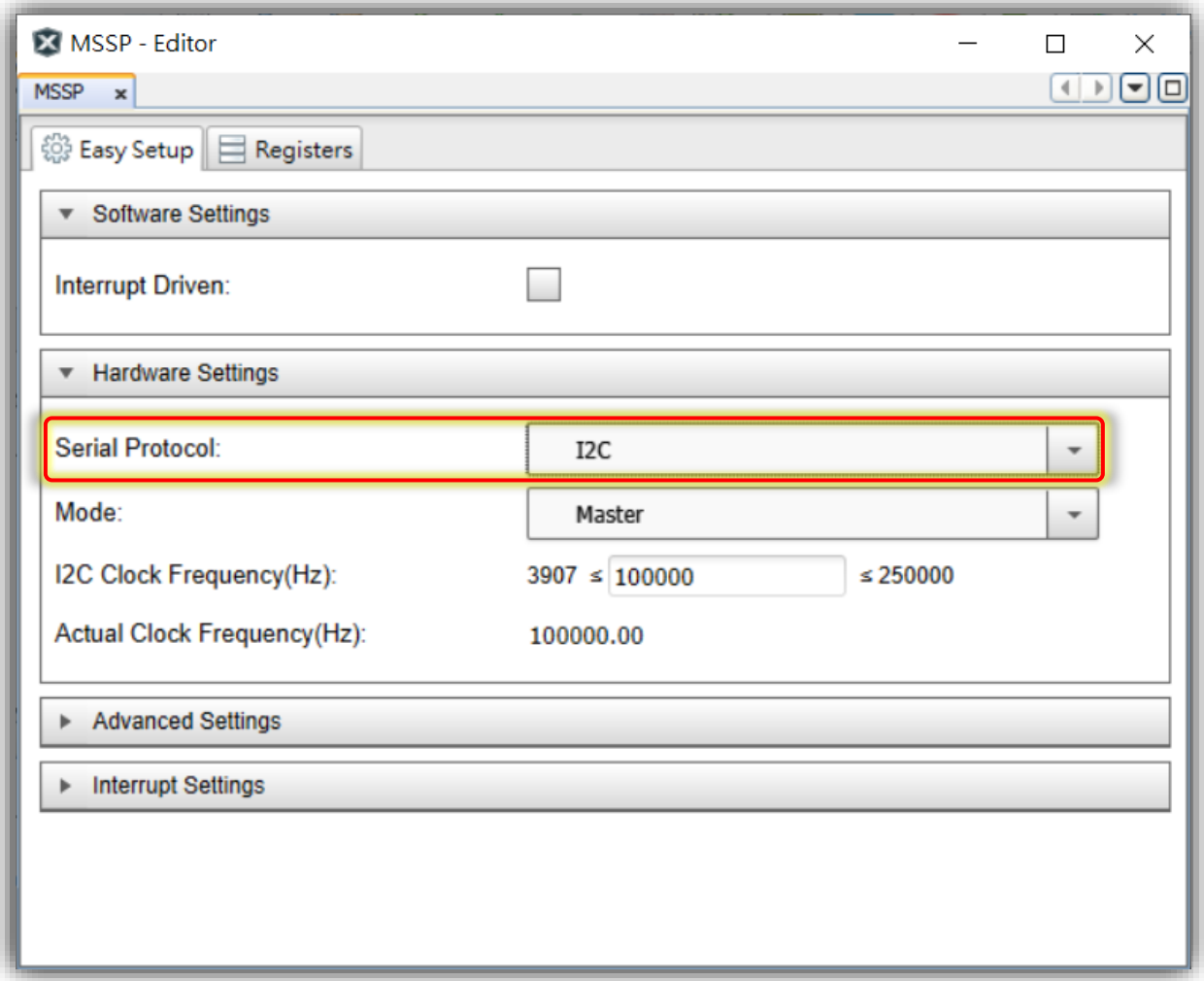
- **Device Resources**
  - **MSSP ▸ MSSP**



Lab10 Temperature Sensor

Step 2

- **Serial Protocol.**
  - **SPI ▸ I2C**

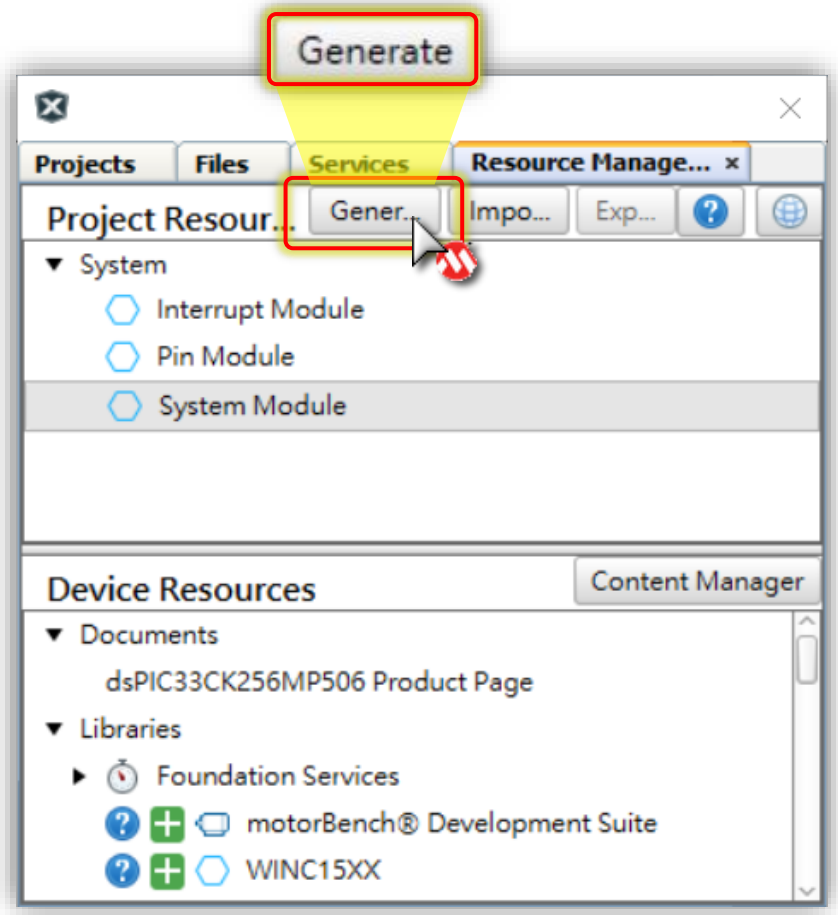


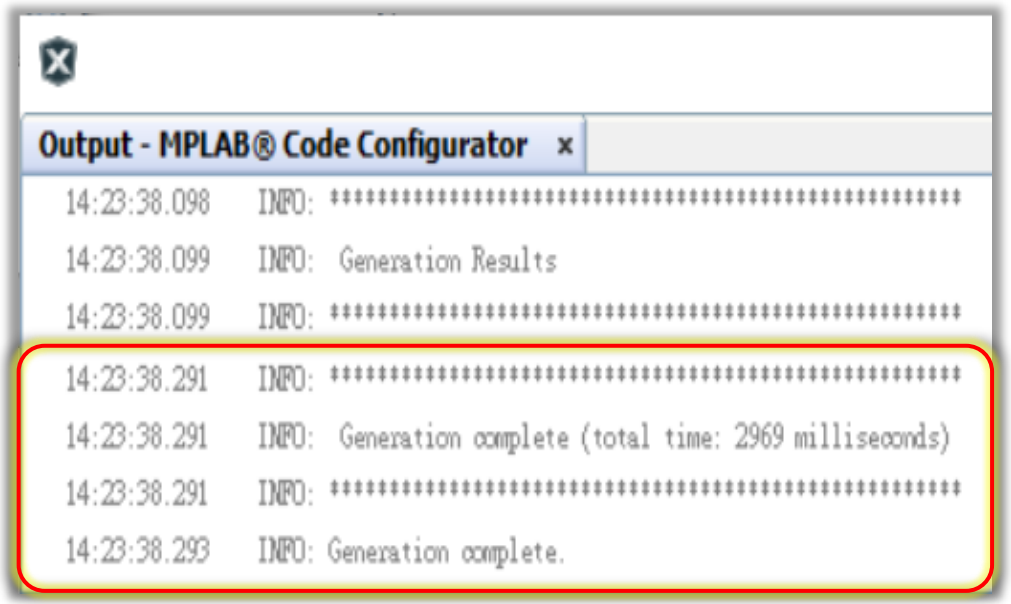
MICROCHIP

Lab10 Temperature Sensor

Step 3

- **Generate Code.**





Lab10 Temperature Sensor

Code Example

```
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/tmr2.h"
#include "mcc_generated_files/examples/i2c_master_example.h"
#include <stdio.h>

#define MCP9800_Address 0x4D
#define MCP9800_REG_TA 0x01
#define MCP9800_Register 0x60
#define MCP9800_Start 0x00

...
volatile uint8_t TMRFlag;
uint16_t MCP9800_Read_Data;

int main(void)
{
    ...
    while (1)
    {
        ...
    }
}

...

void LCD_Show(uint16_t LCD_Value)
{
   ...
}
```

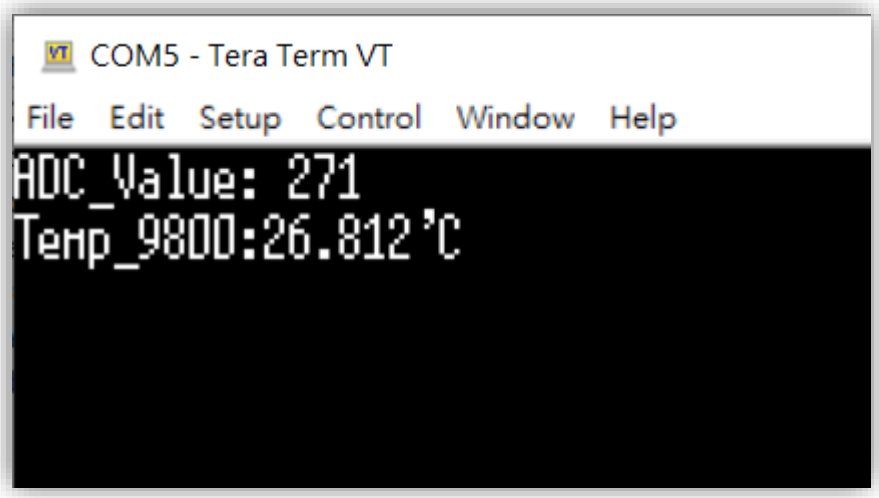
Lab10 Temperature Sensor

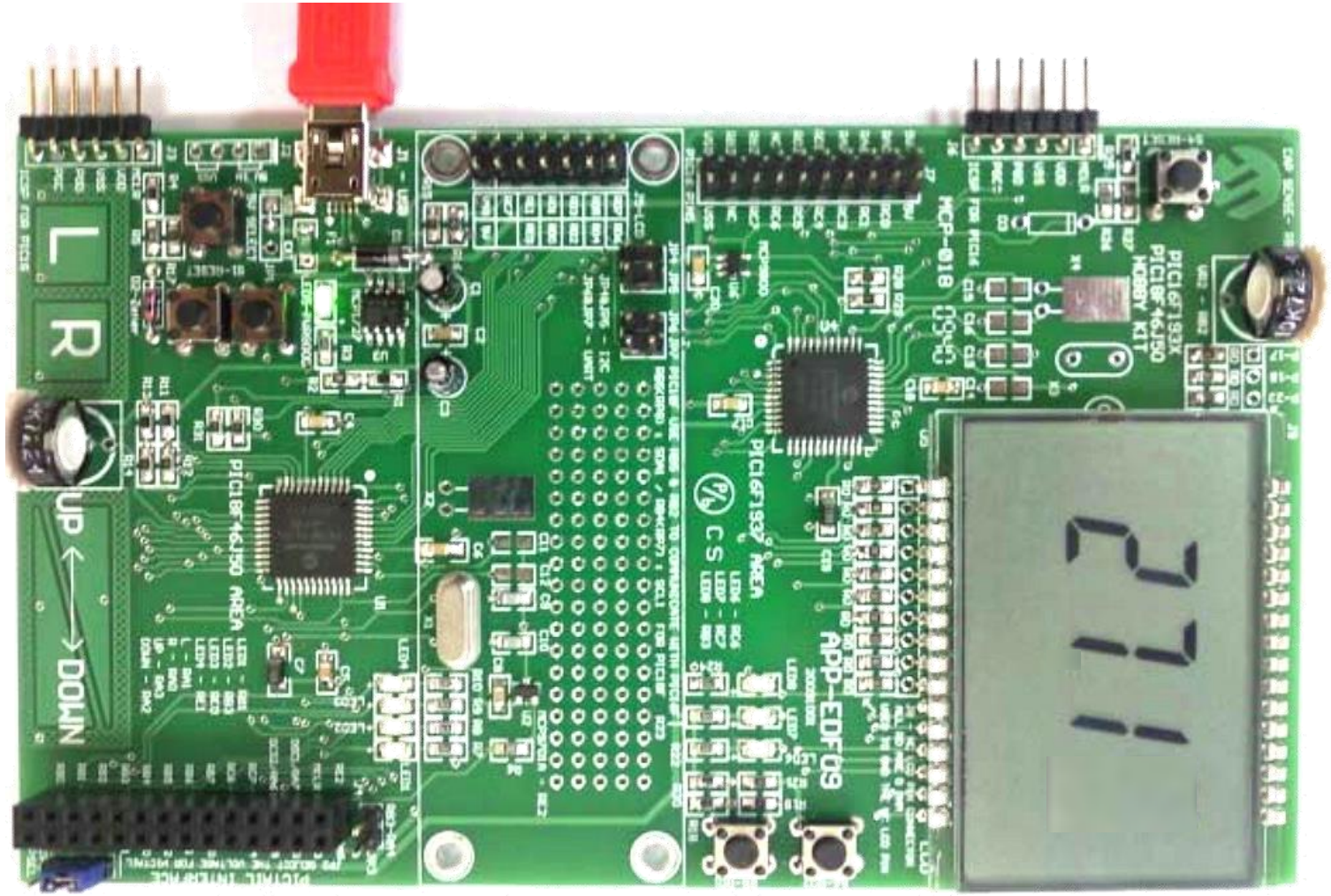
Code Example

```
int main(void)
{
   ...
   TMR2_SetInterruptHandler(TIM2_ISR);
   I2C_Write1ByteRegister(MCP9800_Address, MCP9800_REG_TA, MCP9800_Register);
   __delay_us(100);
   I2C_Write1ByteRegister(MCP9800_Address, MCP9800_Start, 0x00);
   __delay_us(100);

   ...
   while (1)
   {
      // Add your application code

      if (TMRFlag)
      {
         TMRFlag = 0;
         LED1_Toggle();
         ADC_Value = ADC_GetConversion(VR);
         LCD_Show(ADC_Value);
         MCP9800_Read_Data = I2C_Read2ByteRegister (MCP9800_Address, NULL);
         printf(" \033[1;1HADC_Value:%4d\r\n", ADC_Value);
         printf(" \033[2;1HTemp_9800:%2d.%3d'C", (int) ((MCP9800_Read_Data >> 8)&0x7F), (int) ((MCP9800_Read_Data&0xFF) >> 4)*1000 / 16);
      }
   }
}
```

Lab10 Temperature Sensor

Result





MICROCHIP

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