Microcontroller & MCC Classic
PIC1001 v1.10

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

CAE Taiwan
Microchip Technology Inc.

March 31, 2023

Major Course (1/3)

- Microchip 16-bits Microcontroller Introduction
- IDE, Compilers, MCC and Development Tools Introduction
- APP041 8, 16, 32-bits General Purpose EVM Introduction
- Getting Started with First Project
  - Lab0 First Project
- GPIO Architecture
  - Lab1 GPIO Output
  - Lab2 Multi-GPIO Output
  - Lab3 GPIO Input and Output
- Oscillator Architecture
  - Lab4 Clock PRIPLL

Major Course (2/3)

- Interrupt Architecture
- Timer Architecture
  - 💻Lab5 Timer1 Interrupt
- MCCP/SCCP Architecture
  - 💻Lab6 SCCP Timer Interrupt
- OLED Architecture (PPS&SPI)
  - 💻Lab7 OLED Display
  - 💻Lab8 Customize Screen
- 12 Bits High Speed ADC Architecture
  - 💻Lab9 ADC Single Channel
  - 💻Lab10 ADC Multiple Channel

Major Course (3/3)

- High Resolution PWM Architecture
  - 💻Lab11 PWM Output
  - 💻Lab12 PWM Duty Adj By ADC
  - 💻Lab13 PWM Buzzer
- UART Architecture
  - 💻Lab14 UART Printf
  - 💻Lab15 UART Communication
- I2C Architecture
  - 💻Lab16 EEPROM Operation
  - 💻Lab17 Temperature Sensor (MCP9800) Operation

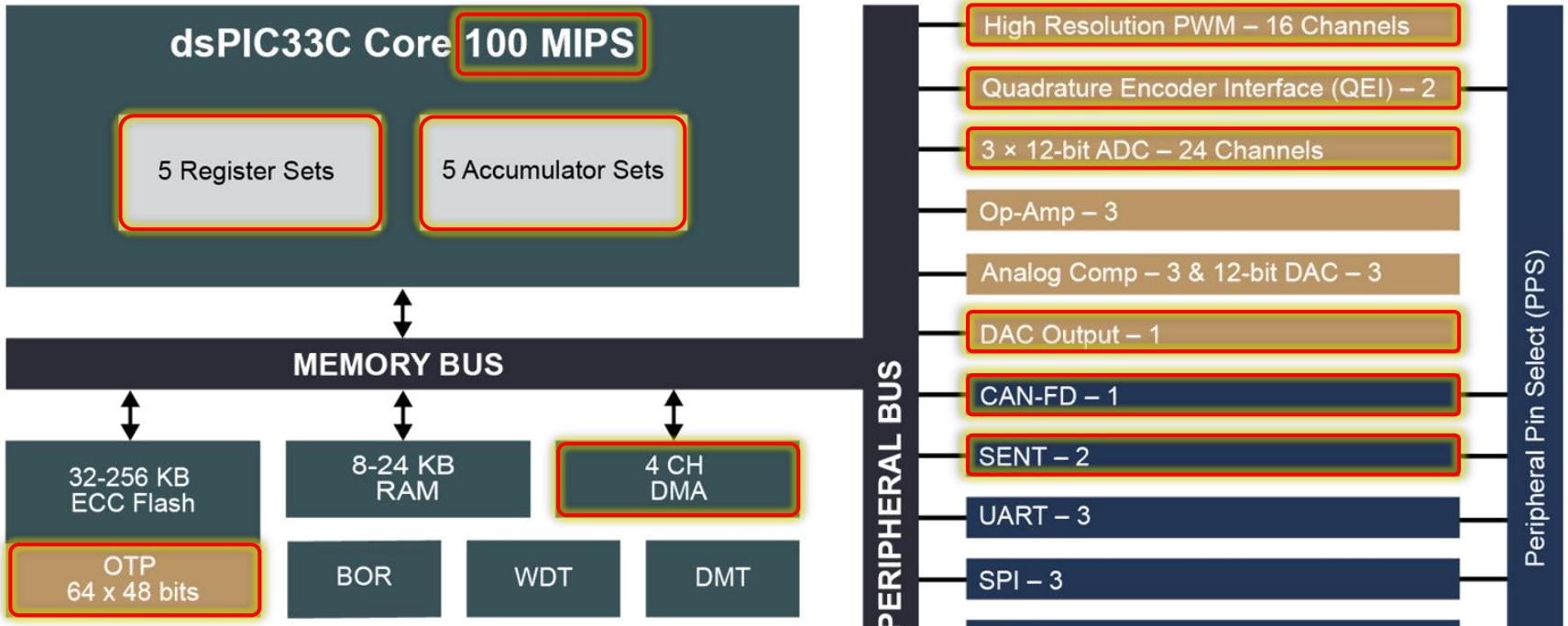
Microchip 16-bits Microcontroller Architecture

MICROCHIP

16-bits PIC/dsPIC® MCU series

- **Microchip 16-bits MCU families include below,**
- PIC24FJ (16 MIPS)
  - Very high CP. Low price and high performance.
- PIC24HJ (40 MIPS)
  - Built-in DMA, upgrade performance to 40 MIPS.
- PIC24EP (70 MIPS)
  - Latest generation of PIC24 series, upgrade performance to 70 MIPS.
- dsPIC30F (30 MIPS)
  - Base on PIC24FJ, Built-in DSP.
- dsPIC33FJ (40 MIPS)
  - Base on PIC24HJ, Built-in DSP.
- dsPIC33EP (60 MIPS)
  - Built-in DSP & USB OTG, upgrade performance to 60 MIPS.
- **dsPIC33CK (100 MIPS)**
  - High Performance, 12-bits High Speed ADC & High-Resolution PWM.

dsPIC33CK256MP508 Family



Operating Voltage: 3.0-3.6 V

Operating Temperature: -40°C to 125°C

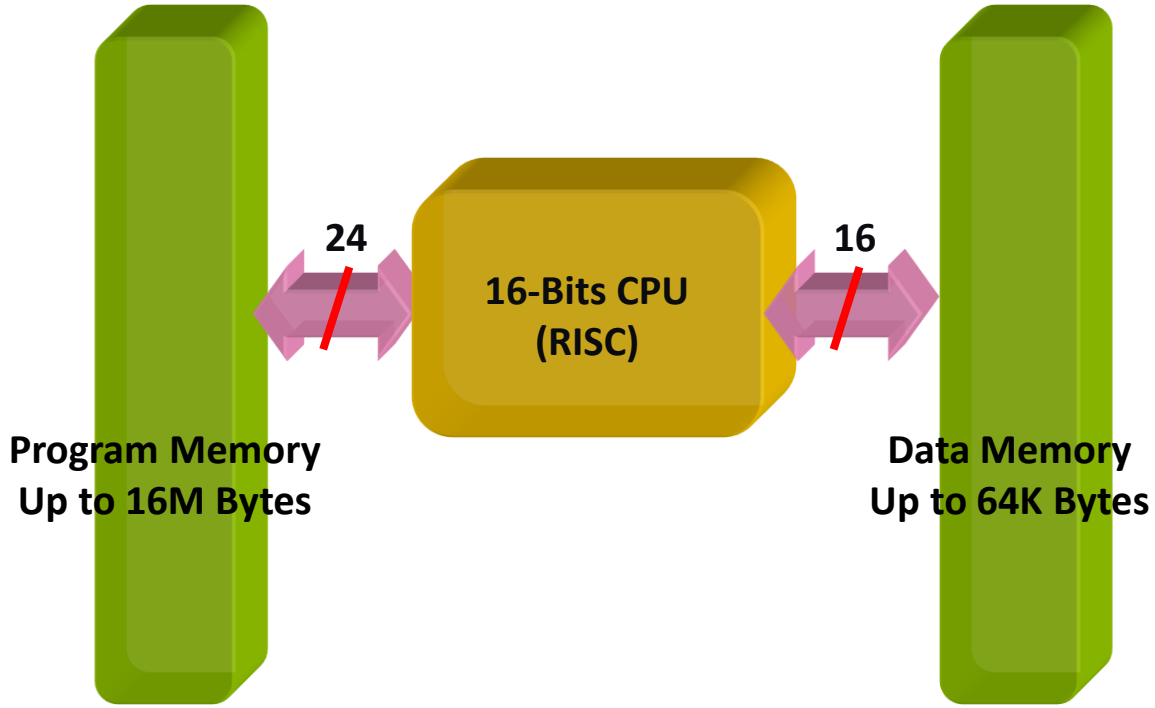
AEC-Q100 Grade 1 Qualification

28-pin: SSOP, uQFN (6 x 6 mm)
36-pin: uQFN (5 x 5 mm)
48-pin: uQFN, TQFP (6 x 6 mm, 7 x 7 mm)
64-pin: QFN, TQFP (9 x 9 mm, 10 mm x 10 mm)
80-pin: TQFP (12 x 12 mm)

MICROCHIP

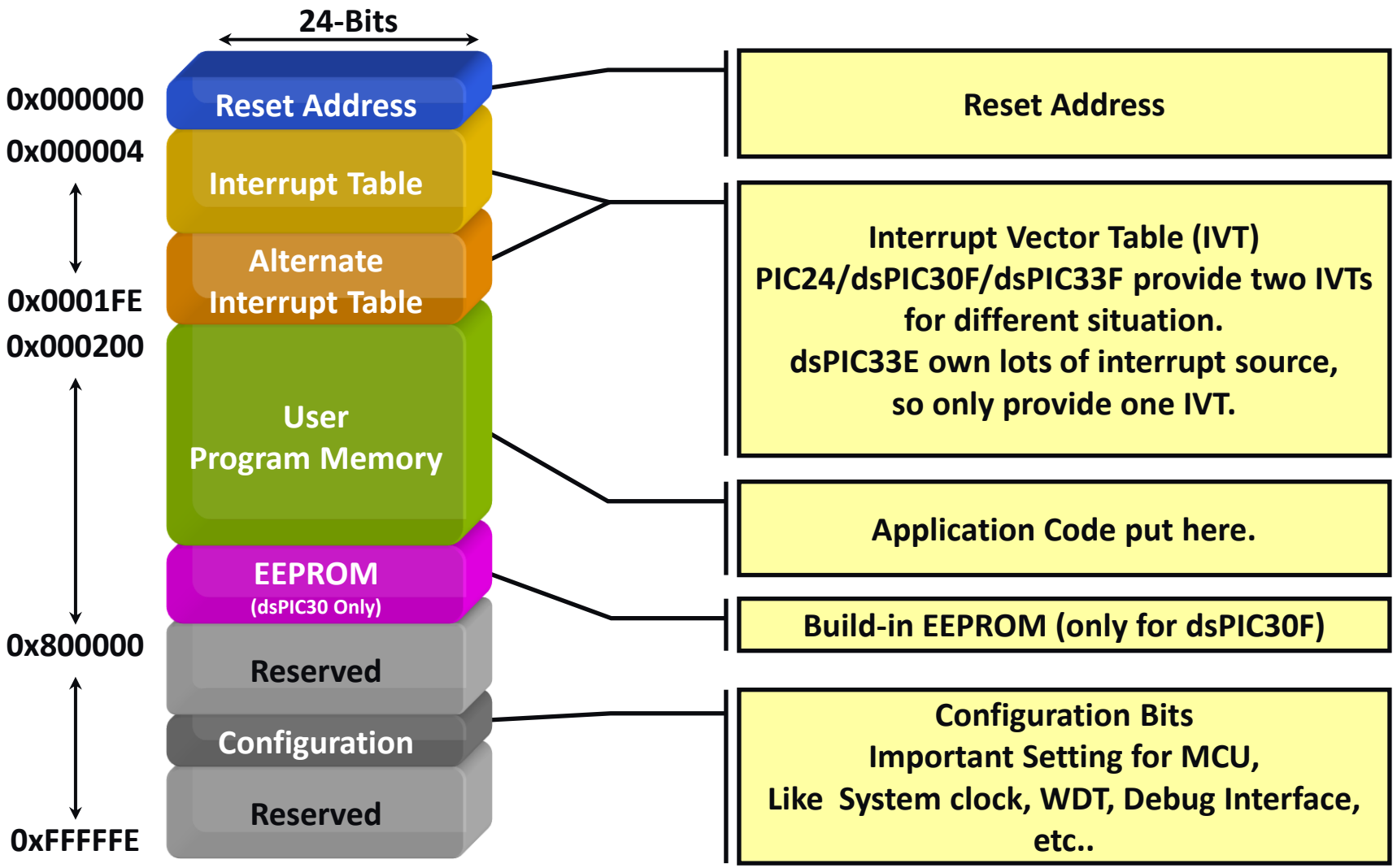
Core Architecture

- Microchip 16-bits PIC/dsPIC® MCU is Harvard architecture.
- This means program and data memory bus are independently.
- Data Memory is 16-bits, Up to 64K Bytes.
- Program Memory is 24-bits, Up to 16 M Bytes.

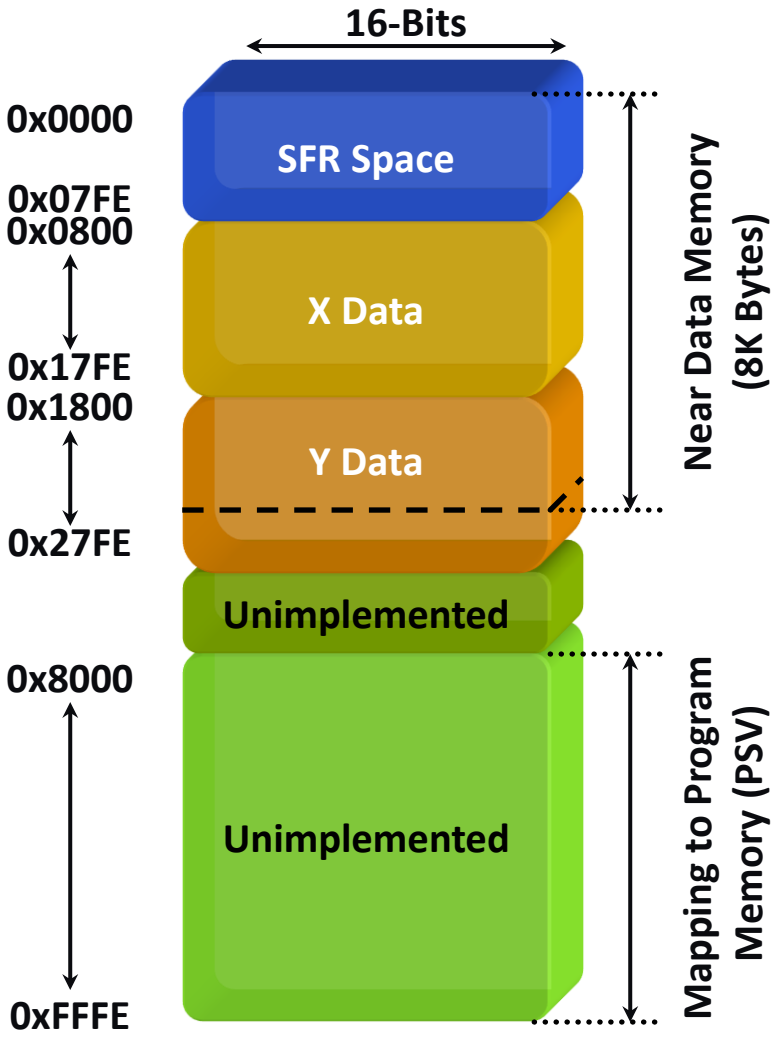


MICROCHIP

Program Memory Mapping



Data Memory Mapping



The 8-Kbyte area is referred as the near data space.
The 64-Kbyte area is referred as the far data space.

The first 2 Kbytes are primarily occupied with Special Function Registers (SFRs).

X Data, Y Data area only for DSP Engine For Dual Access Operation.
All Data is X Data area for PIC24 series.

0x8000 ~ 0xFFFF is mapping area for PSV use.

MICROCHIP

Machine & Instruction Cycle

- **T_{OSC}(Machine Cycle) :**
  - A machine cycle consists of the steps that a computer's processor executes whenever it receives a machine language instruction. (techopedia)
- **T_{CY}(Instruction Cycle) :**
  - The basic operational process of a computer system.(wiki)
- **dsPIC30 Family**
  - 1 T_{CY} = 4 T_{OSC} . (F_{CY} = F_{OSC} /4)
    Ex:120 MHz > 30 MIPS, T_{CY} = 33nS.
- **PIC24F/PIC24H/dsPIC33 Families**
  - 1 T_{CY} = 2 T_{OSC} . (F_{CY} = F_{OSC} /2)
    Ex:32 MHz > 16 MIPS, T_{CY} = 62.5nS.

X IDE, C Compiler, MCC & Development Tools Introduction

MICROCHIP

Soft Request In this Course

- **MPLAB® X IDE v6.05** https://www.microchip.com/mplab/mplab-x-ide
- **MPLAB® XC16 v2.00** https://www.microchip.com/mplab/compilers
- **MPLAB® Code Configurator (MCC) v5.3.0**
- **MCC Core v5.5.0**
- **MCC Classic Libraries (From X IDE MCC Content Manager)**
  - PIC24 / dsPIC33 / PIC32MM MCUs v1.171.1
  - Foundation Services v0.2.3
- **MPLAB® X IDE Packs**
  - dsPIC33CK-MP DFP v1.11.346 (Device Family Pack)
  - SNAP tool packs v1.14.1052 (Tool Firmware Pack)

➢ **MCC Core and Libraries Update from X IDE > MCC > Content Manager**

➢ **dsPIC33CK-MP DFP and SNAP TP Update from X IDE > Tools > Packs**

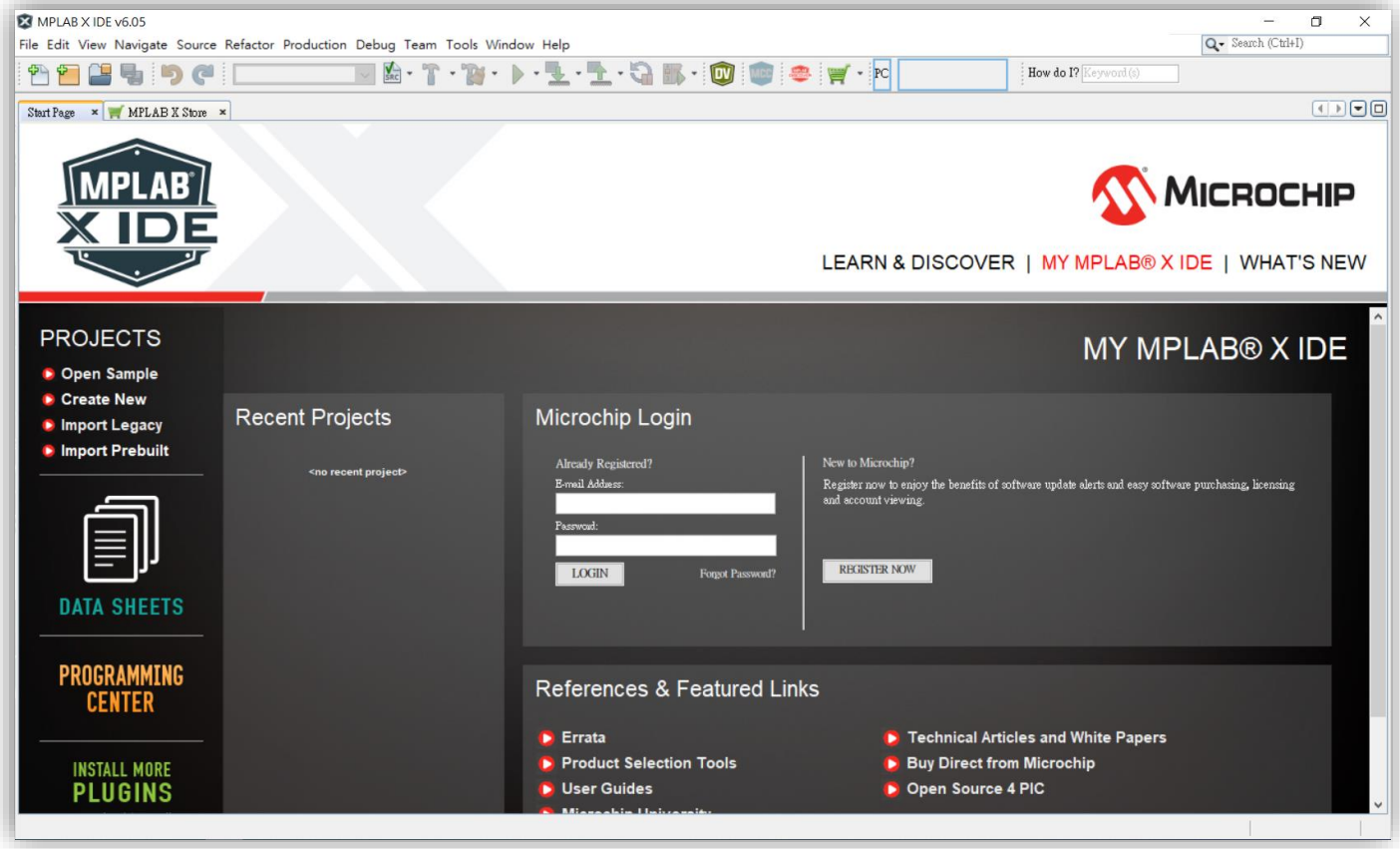
(Lasted Update : 2023/03/27)

MICROCHIP

All in One Development Tools

MPLAB® X IDE

- **New generation integrated Development Tools, Support PIC, dsPIC, AVR, CEC and SAM Series MCU/MPU, Provide Plug-in function to extend more advance function. Java Based, Cross platform, Current version is v6.05.**



MICROCHIP

C Compiler for 16-bits PIC/dsPIC® MCU

MPLAB® XC Compilers

- **XC16 is new generation C compiler, it's supported all 16-bits PIC/dsPIC® MCU (PIC24F/PIC24H/PIC24E, dsPIC).**
- **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.**

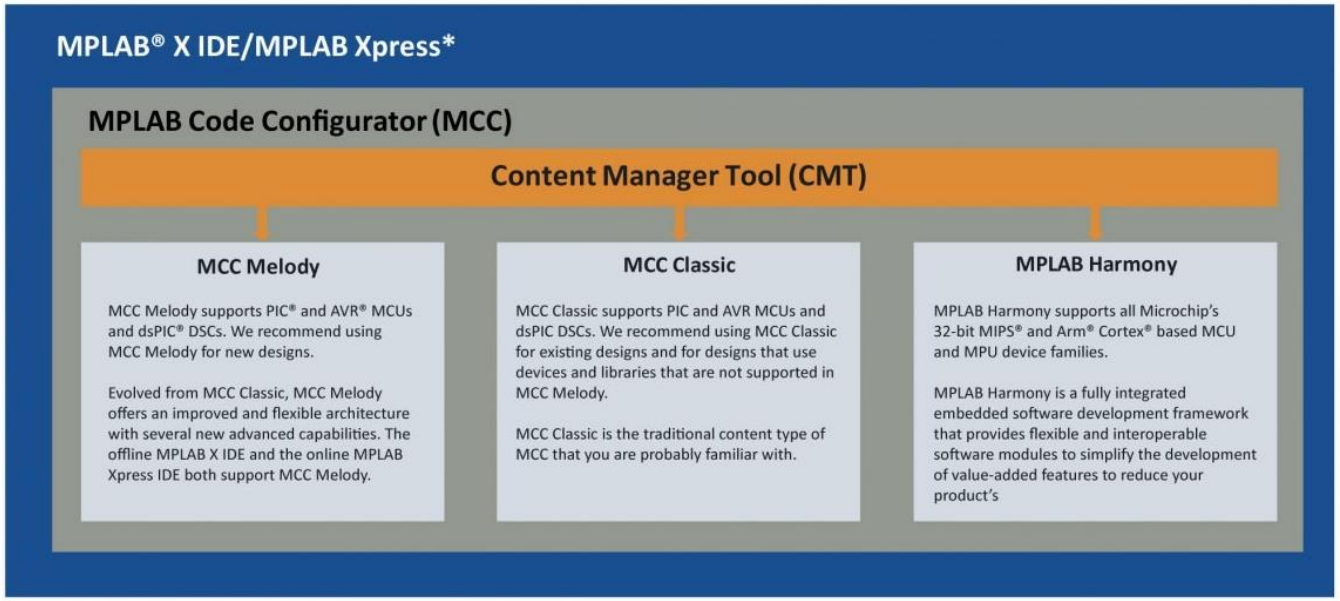
License Type	Installs On	# of Activations	# of Users	Wait Time Between Users	HPA
Workstation License	Workstation	3	1	None	Yes
Subscription License	Workstation	1	1	None	No
Site License	Network	1	Varies by Seat	None	Yes
Network Server License	Network	1	Unlimited	One Hour	Yes
Virtual Machine *	Network	1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

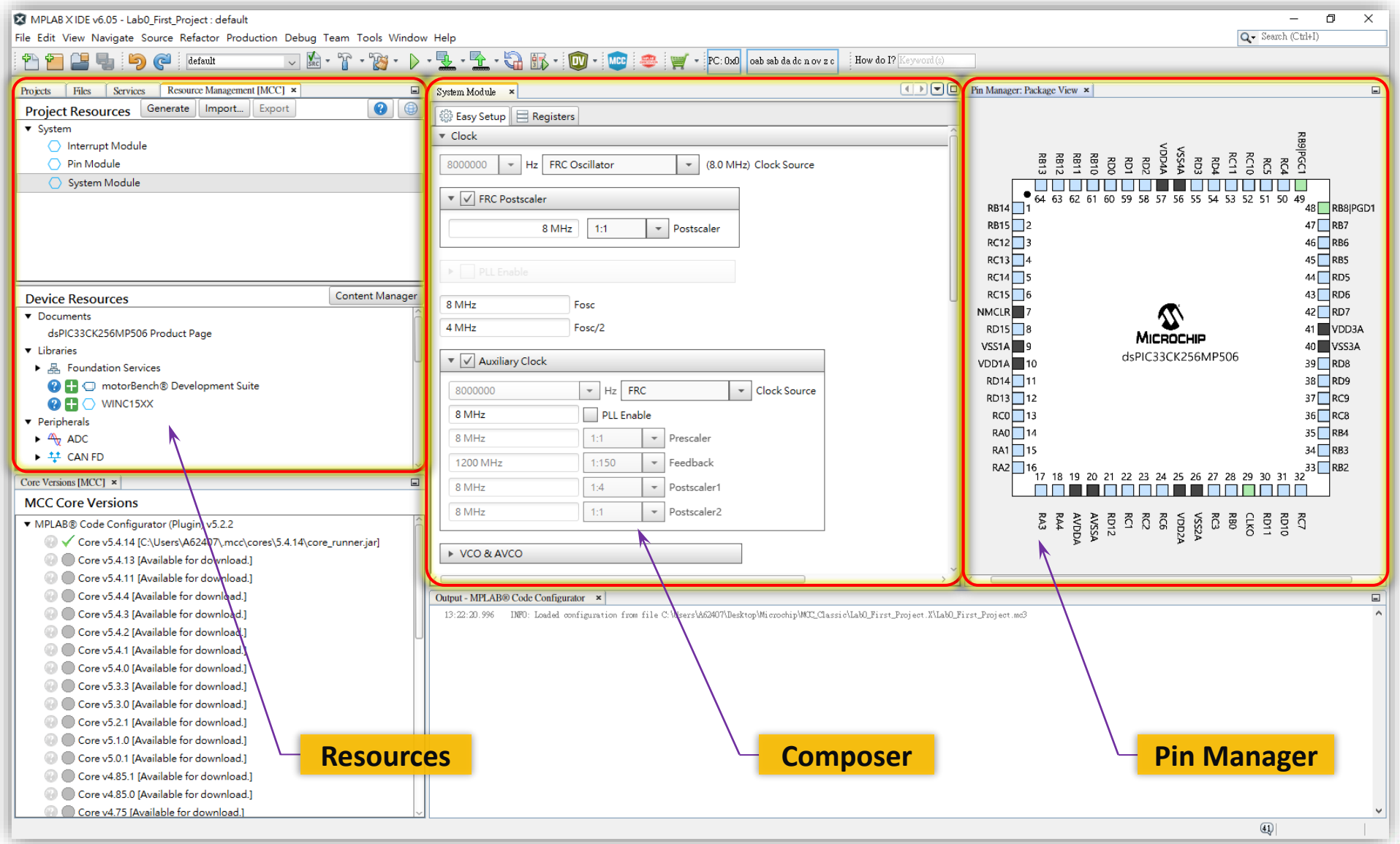
- **MPLAB® Code Configurator - MCC is a free, graphical programming environment that generates seamless, easy-to-understand C code to be inserted into your project.**
- **Supports 8-bits, 16-bits and limited 32-bits PIC® microcontrollers.**
- **MCC consists of three content types: MCC Melody, MCC Classic and MPLAB Harmony. It offers application libraries and system and peripheral drivers for the development of embedded software.**



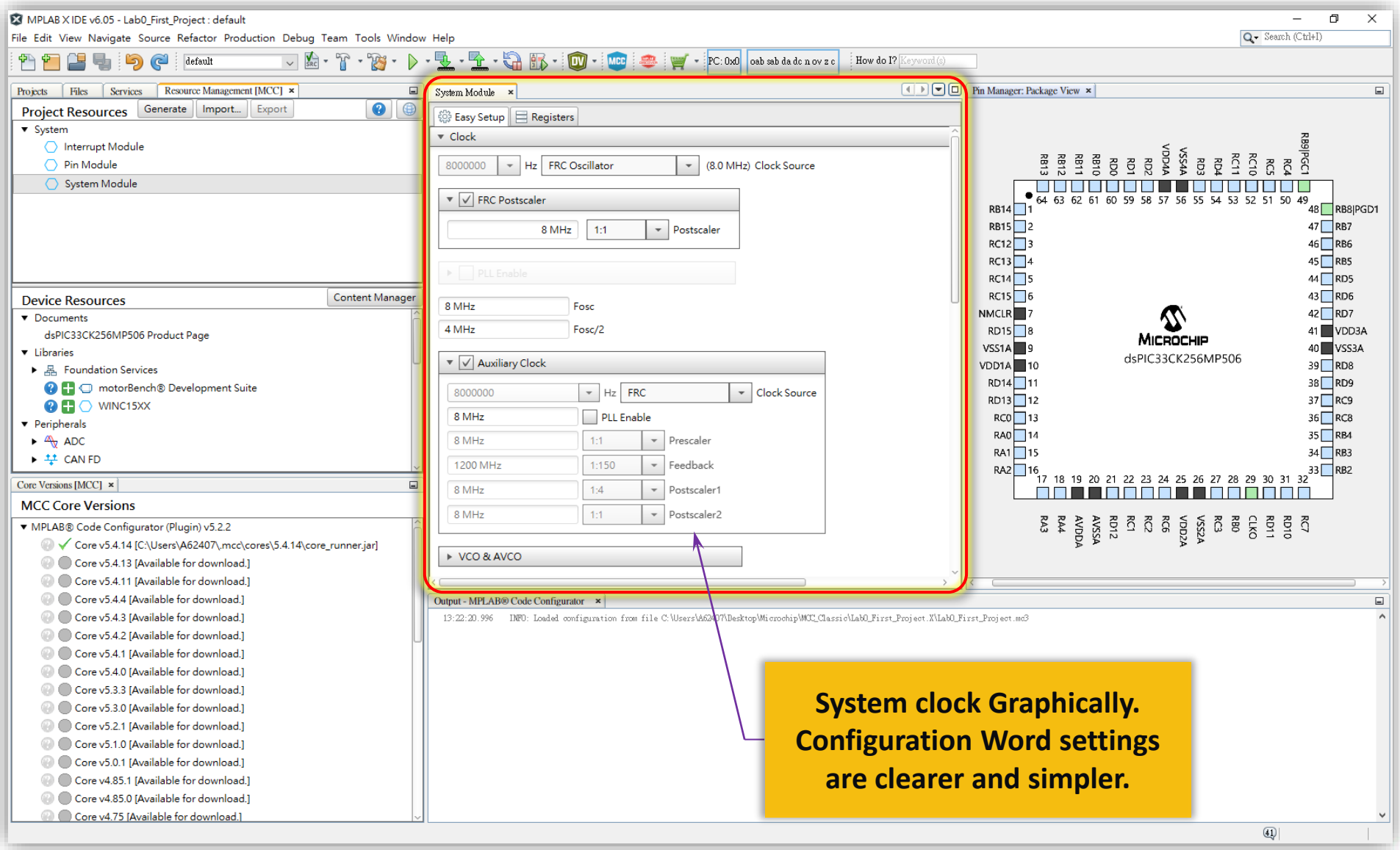
MPLAB® CODE CONFIGURATOR

MICROCHIP

MCC Classic GUI Quick View

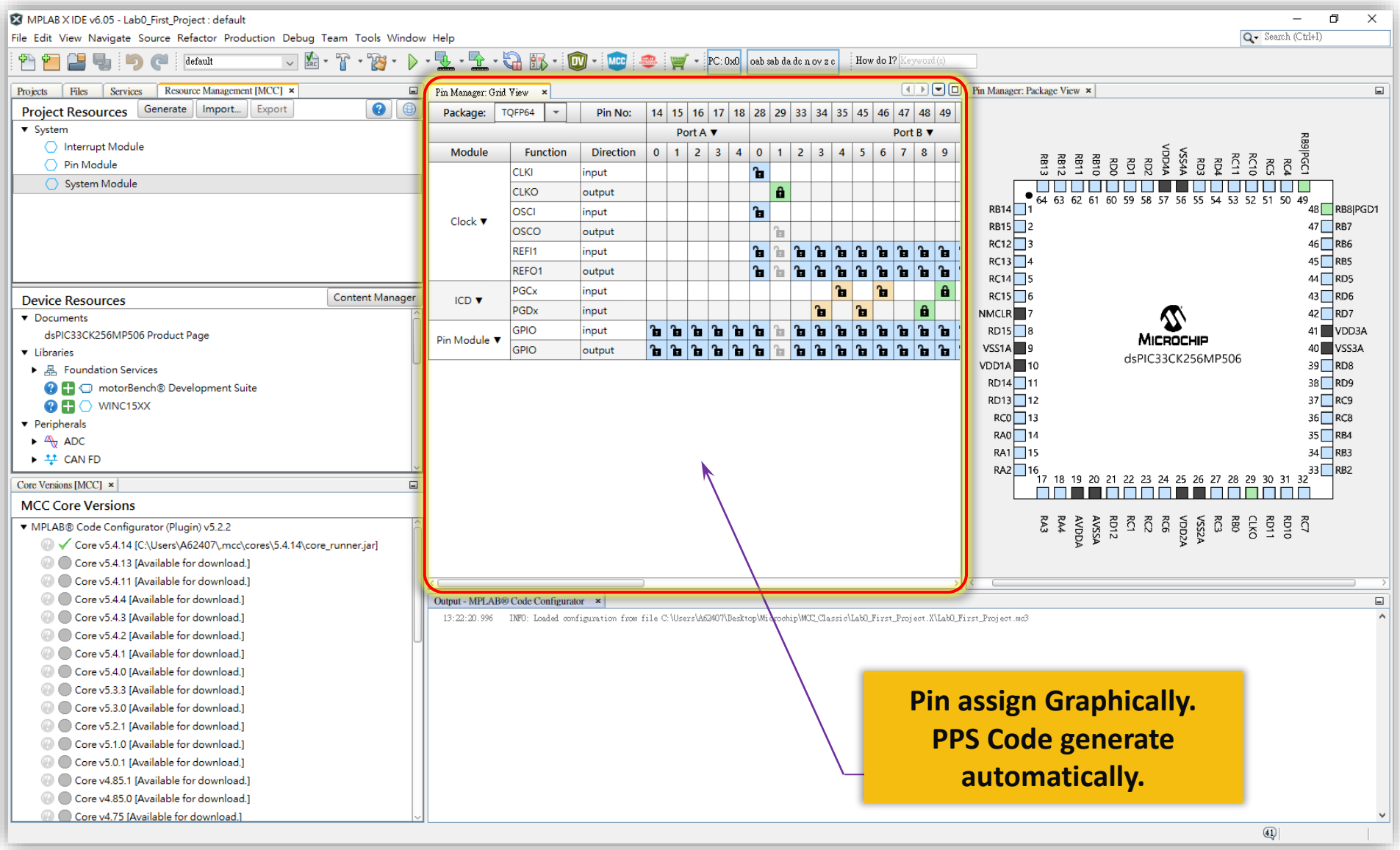


MCC Classic GUI Quick View

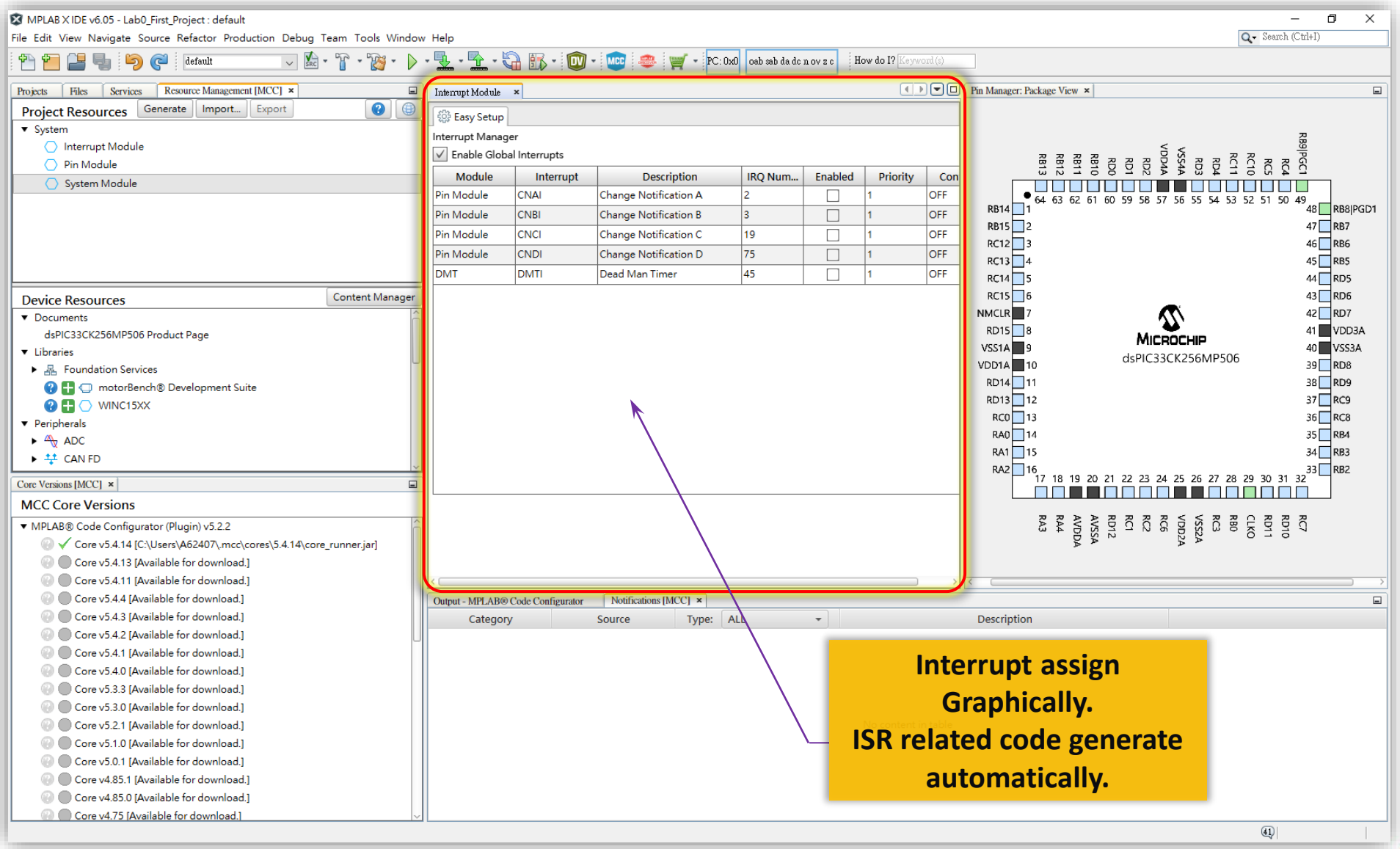


MICROCHIP

MCC Classic GUI Quick View



MCC Classic GUI Quick View



MCC Classic GUI Quick View

More ?

www.microchip.com/mcc

MICROCHIP

Programmer/Debugger

MPLAB® PICkit 4 (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.**
- **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).

$88.54

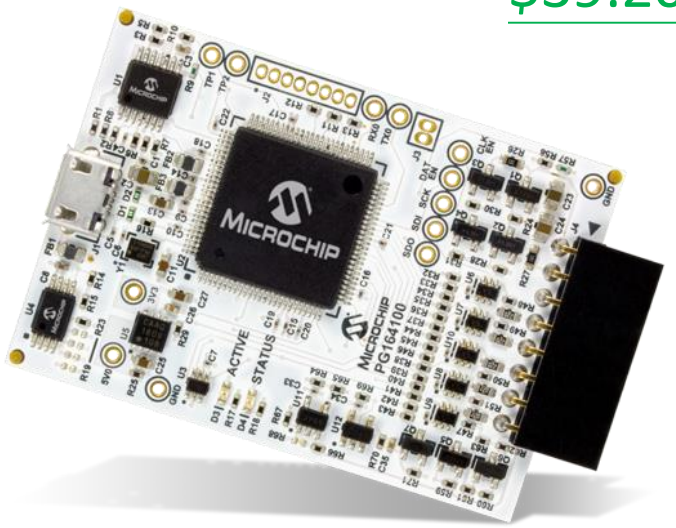
MICROCHIP

Programmer/Debugger

MPLAB® 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

$39.20



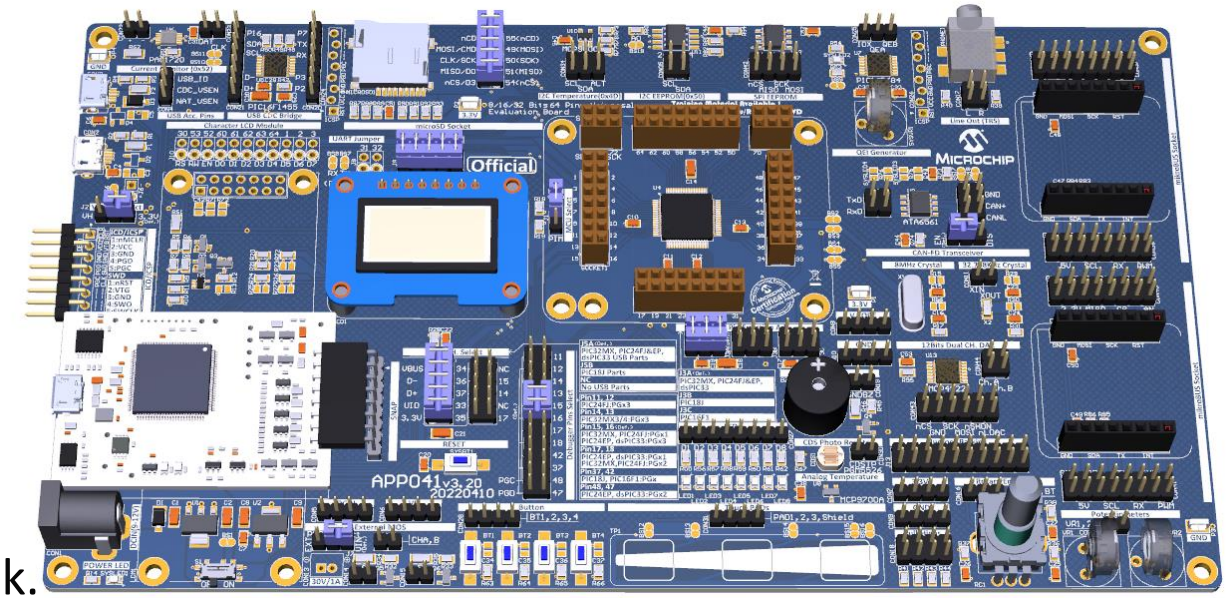
MICROCHIP

APP041 8, 16, 32-bits
General Purpose EVM Introduction

MICROCHIP

8, 16, 32-bits General Purpose EVM

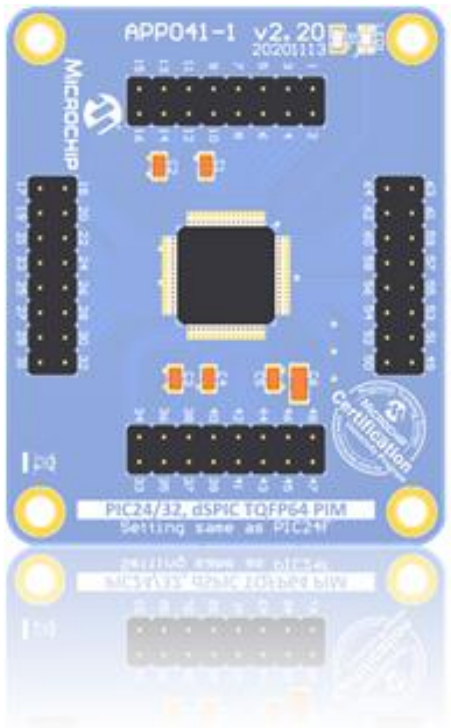
- **APP041 v3.10 is general purpose EVM, it's support 8, 16, and 32-bits 64-pins microcontroller. Built-in PIC24FJ128GB106-I/PT microcontroller at factory.**
  - **Built-in SNAP Debugger/Programmer.**
  - **PIM for easy device swapping.**
  - **Multiply power supply available,**
    - Serial USB Port
    - Native USB Port
    - DC Jack (DC 9 ~ 12V)
  - **Built-on rich components,**
    - LEDs, Buttons, Potentiometer.
    - Temperature and Light Sensor, Buzzer, Phone Jack.
    - Graphic OLED Module, Character LCD Module(Optional).
    - I2C & SPI EEPROM, DAC, ENCODE, QEI, CAN-FD Transceiver.
    - MicroSD Socket, Touch PADs, USB Serial Emulator, MikroBUS Socket.

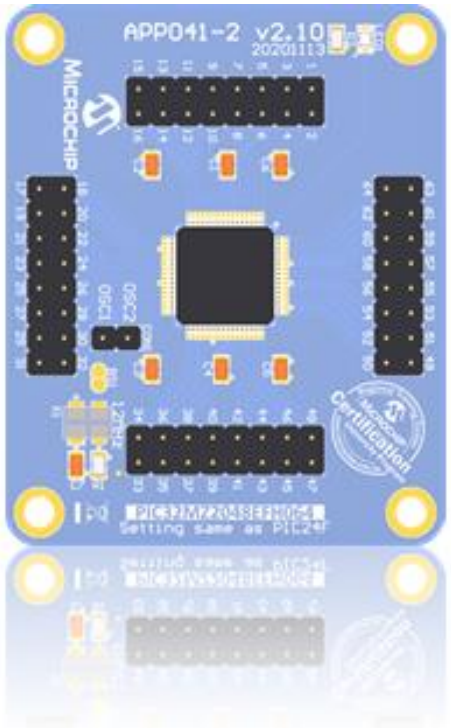


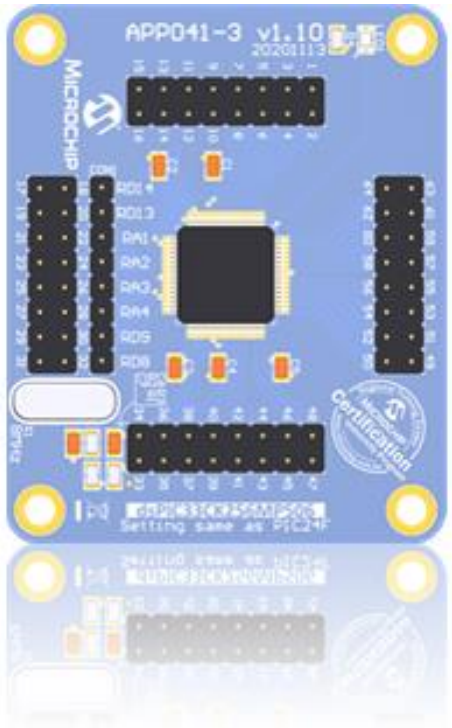
MICROCHIP

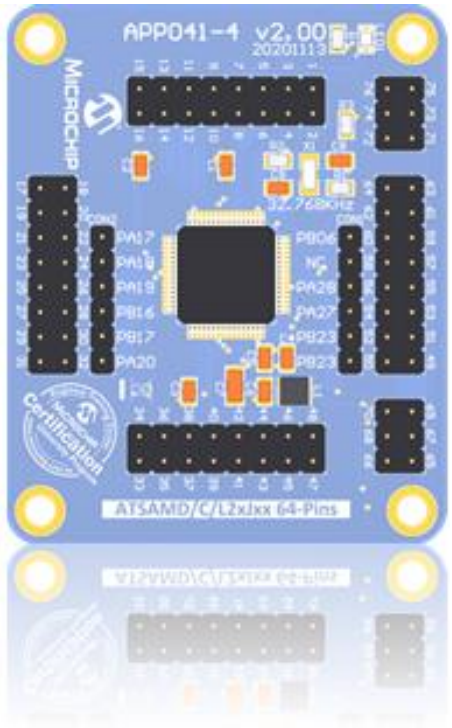
PIM, Plug-In Module

- **Rich PIMs to support different microcontroller.**
- **It’s included PIC18, PIC24, dsPIC33, PIC32MX, PIC32MZ, SAMD21, SAMD51, SAML and SAMC series.**



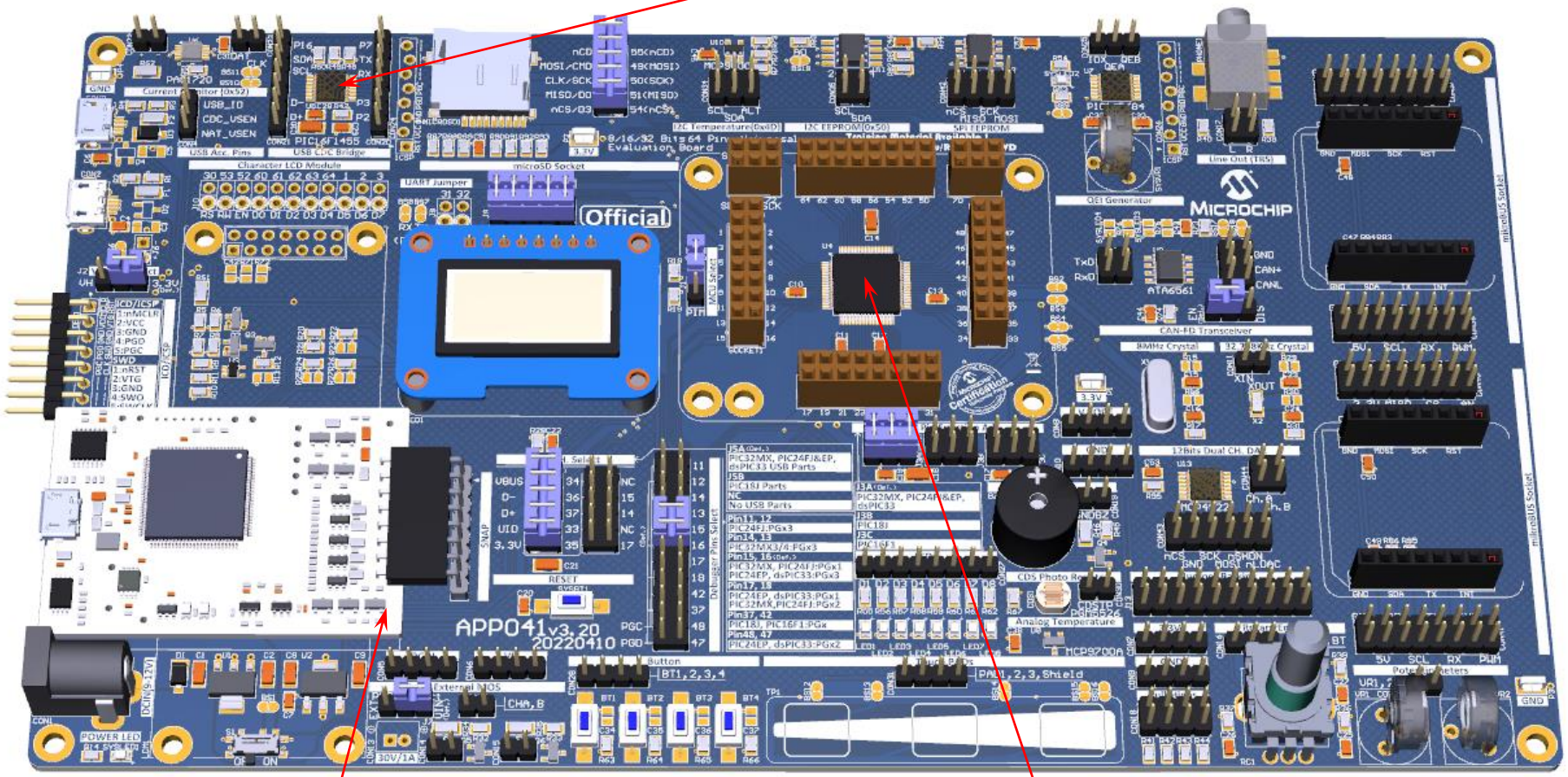






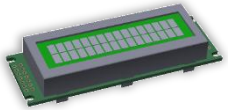
MICROCHIP

Main Controller



Components-1

MicroSD Card Socket



Character LCD Socket

Serial Port USB

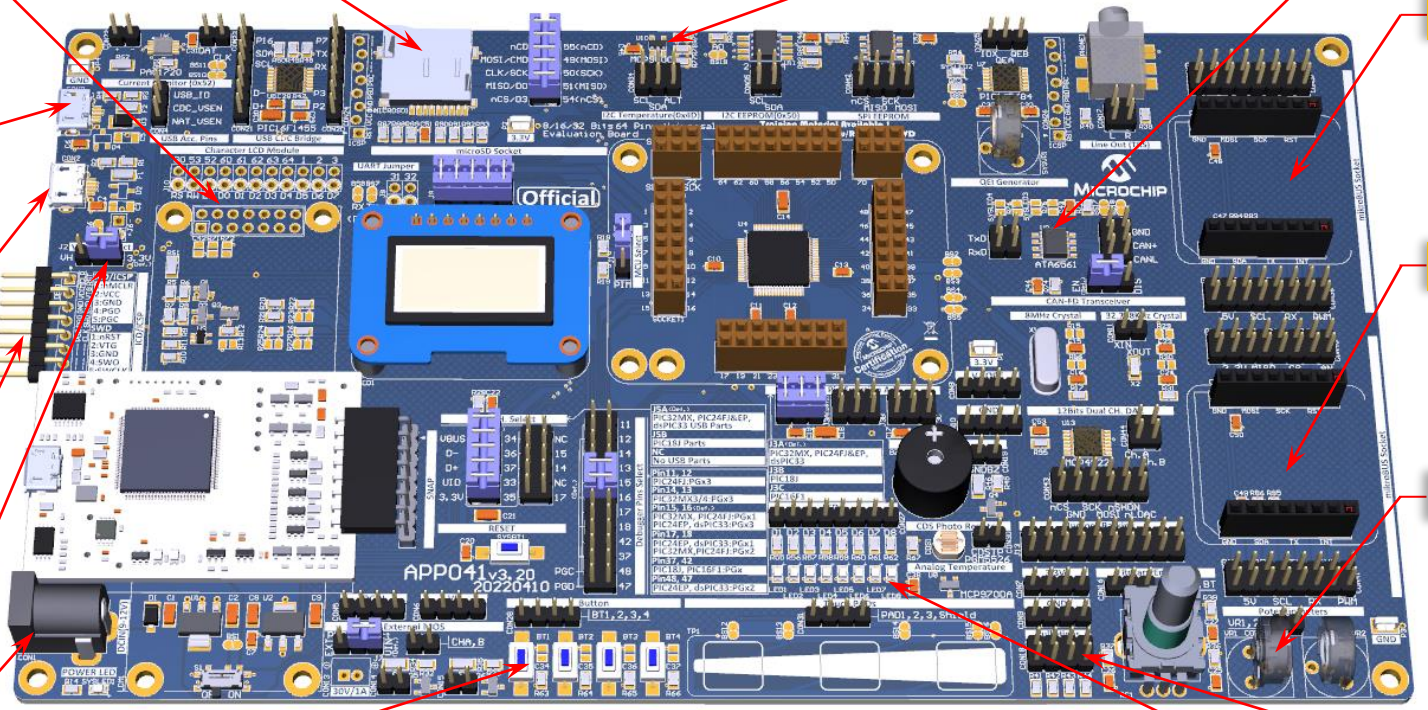
Native USB

ICD/ICSP Connector

Main Voltage Select

DC Jack (9~12V)

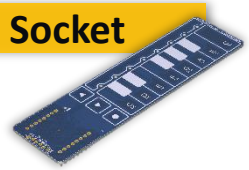
Buttons



I2C Temperature Sensor

CAN-FD Transceiver

MikroBUS Socket



MikroBUS Socket

Potentiometer

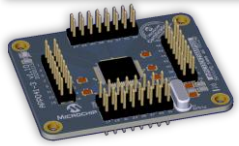
Pull High Res.

LEDs

MICROCHIP

Components-2

MUC PIM Socket



MCU Select

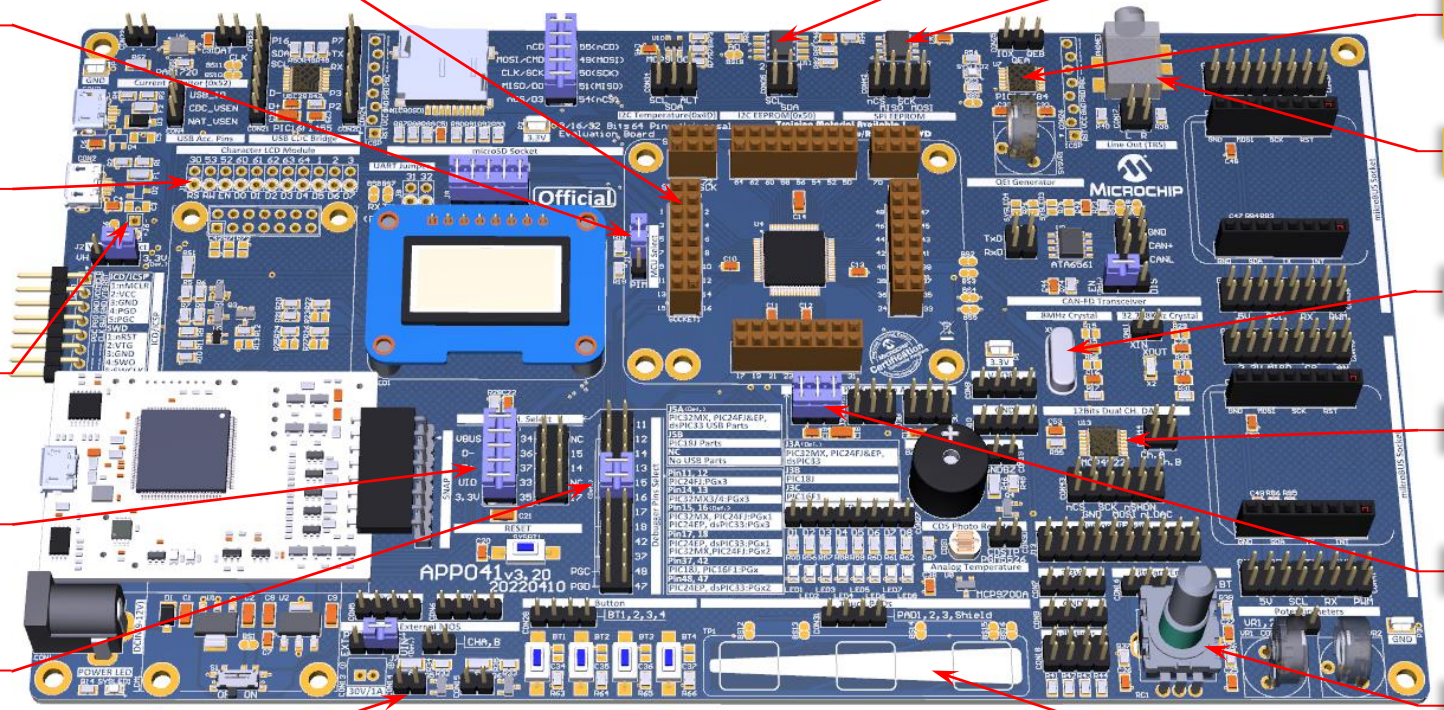
Character LCD Pin Select

USBID Connector

USB Pin Select

Debug Pin Select

Ext. MOSFET



I2C EEPROM

SPI EEPROM

PIC16F684 (Encoder)

Phone Jack

8MHz Crystal

SPI DAC

MCU Pin Function

Encoder

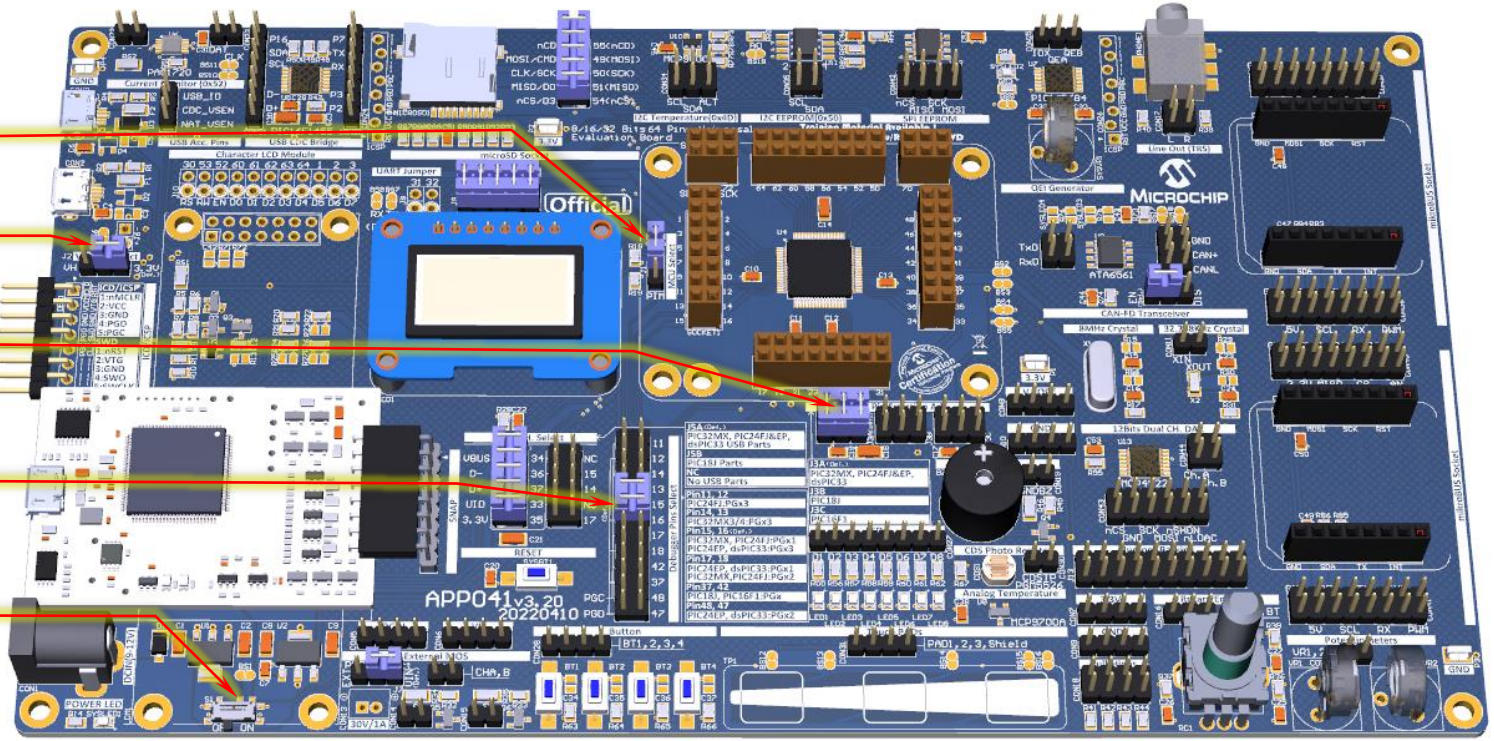
Touch

MICROCHIP

Caution !

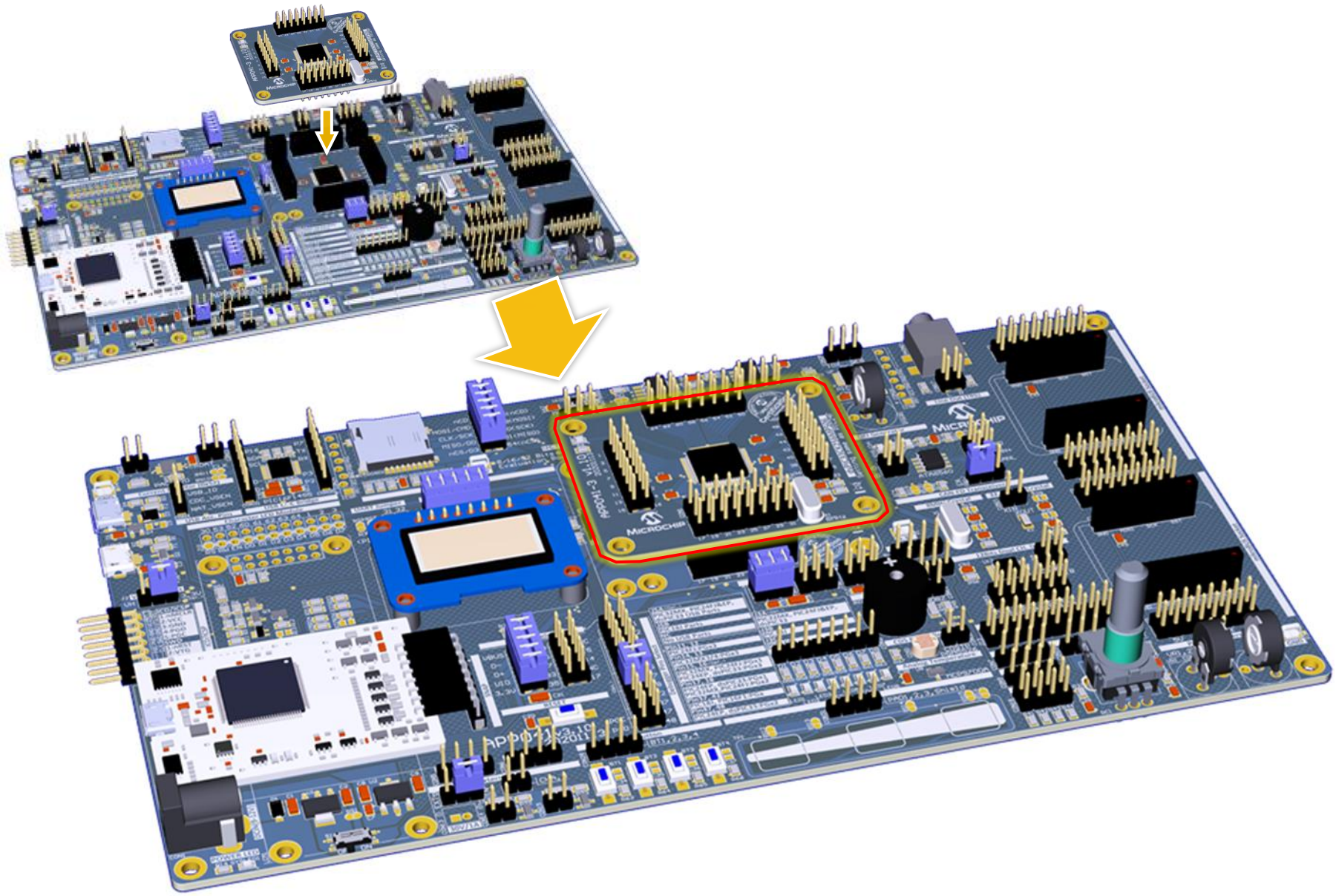
- You must check all jumpers position before power on. Wrong setting will damage silicon and your power source.
- Please confirm all setting shows below, It’s for dsPIC33CK256MP506.

J1	PIM
J2	3.3V
J3	J3A
J4	Pin17, 18 (PGx2)
S1	ON



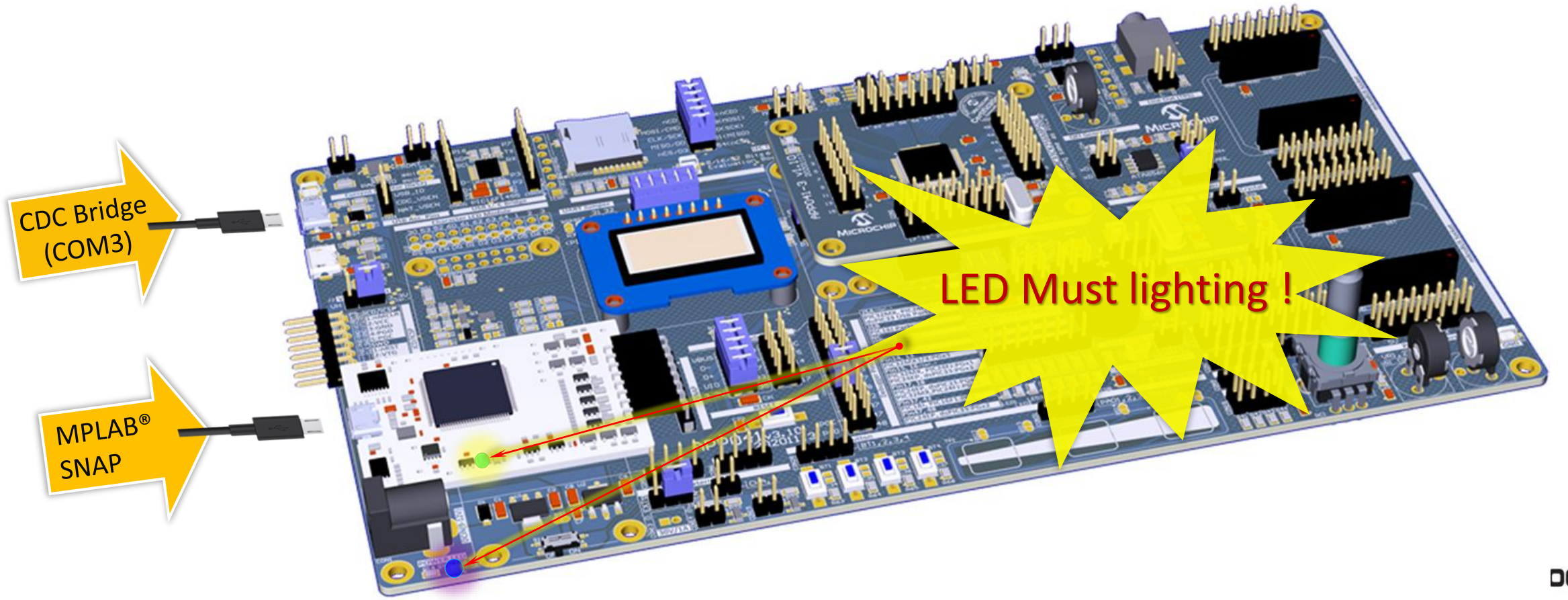
MICROCHIP

Plug-In Module Assembly



Power On & Getting started

- **Just plug-in micro-USB cable to COM3 and MPLAB® SNAP then confirm power LED (SYSLED1) and ACTIVE LED on MPLAB® SNAP are light or not.**
- **OLED/LCM will show preloaded demo firmware.**



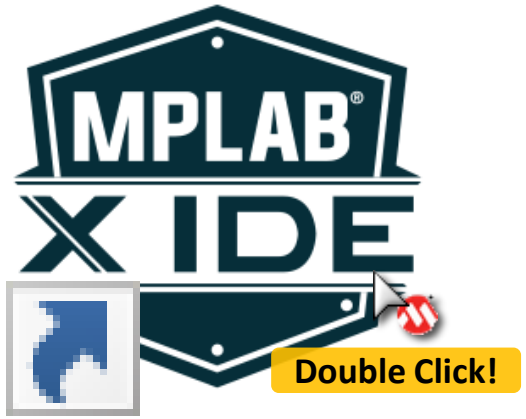
Lab Preparation

Check Compiler & MCC version

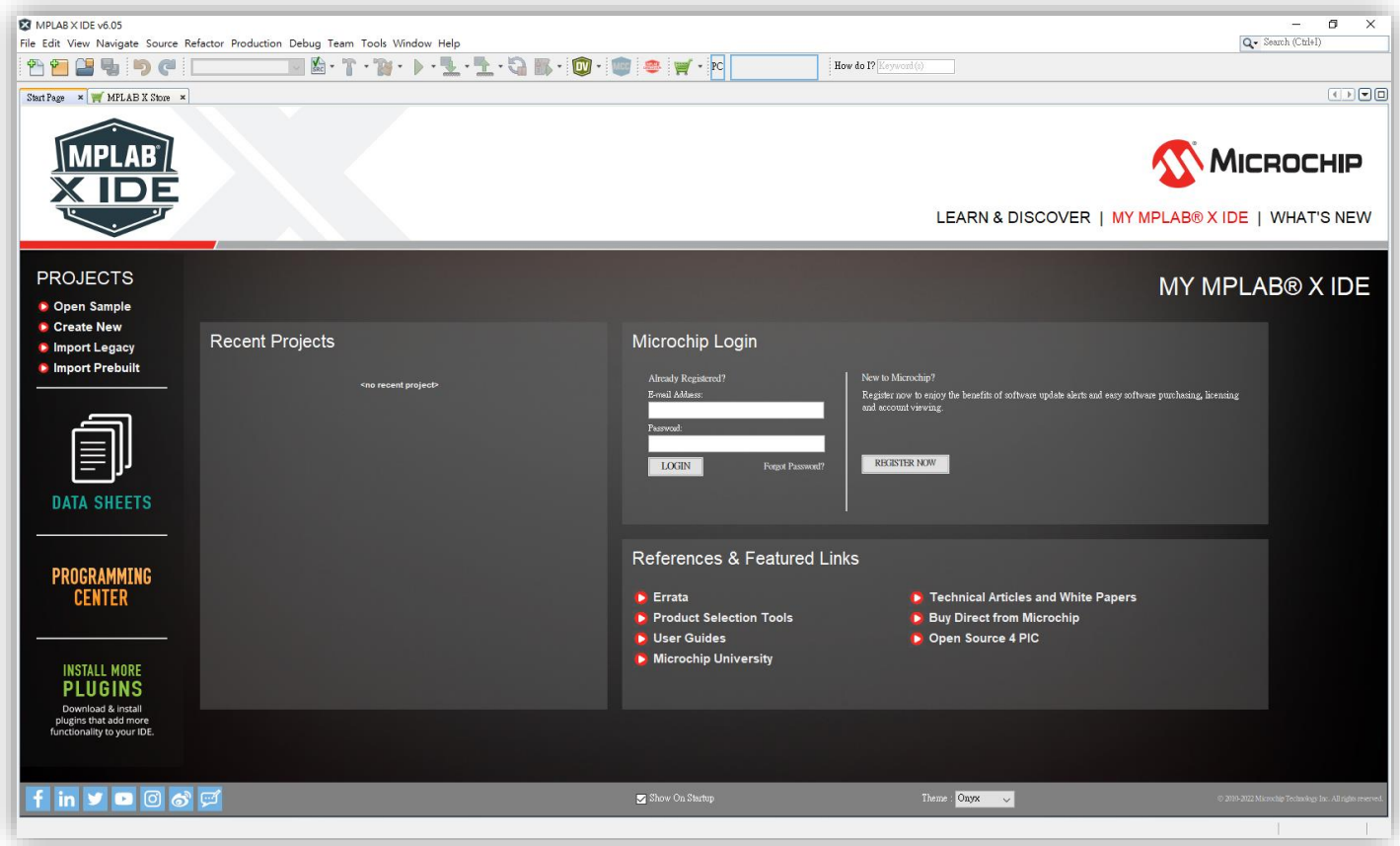
Lab Preparation

Step 1

- Open **MPLAB X IDE v6.05** firstly.



MPLAB X IDE
v6.05

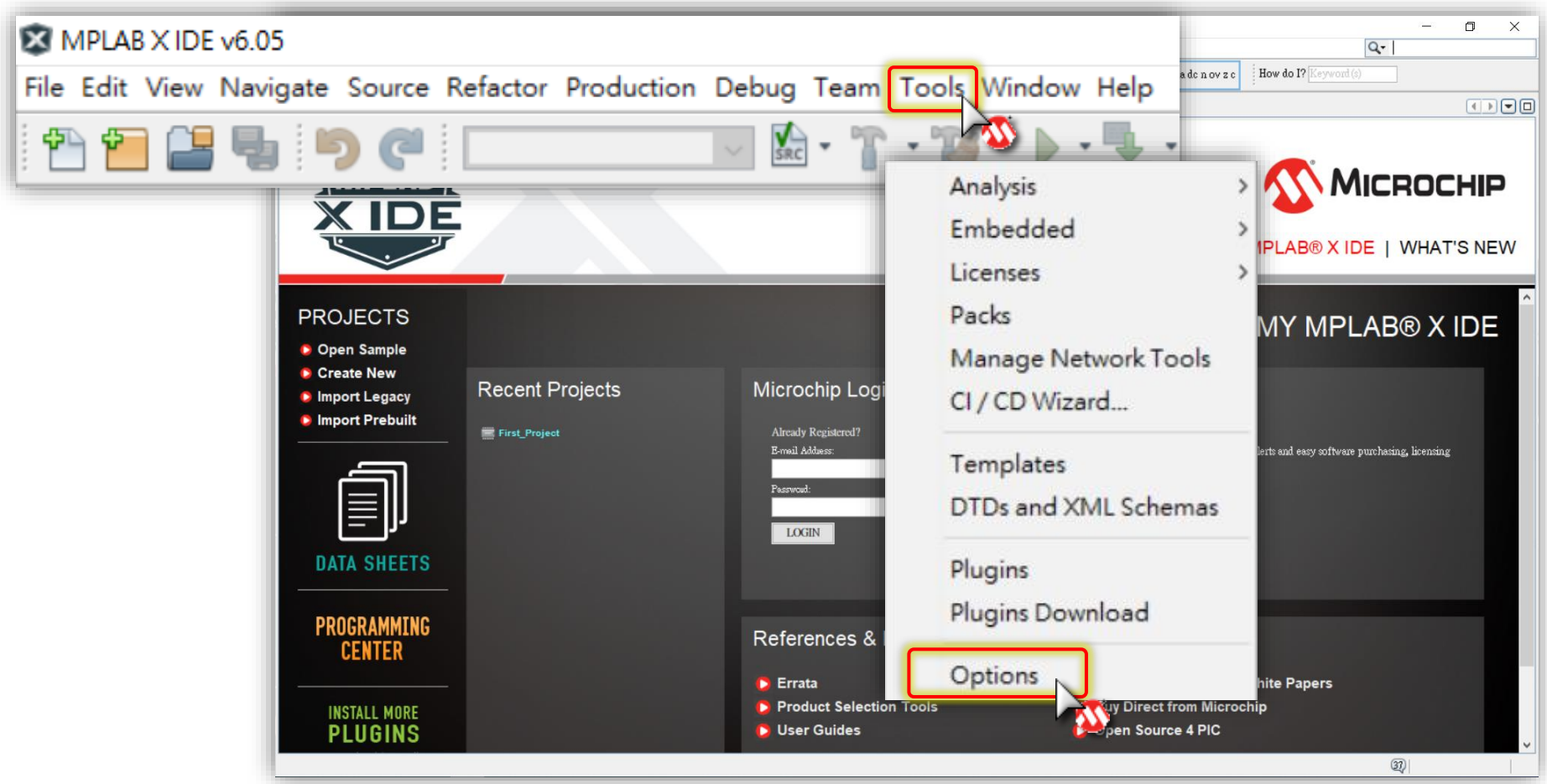


MICROCHIP

Lab Preparation

Step 2

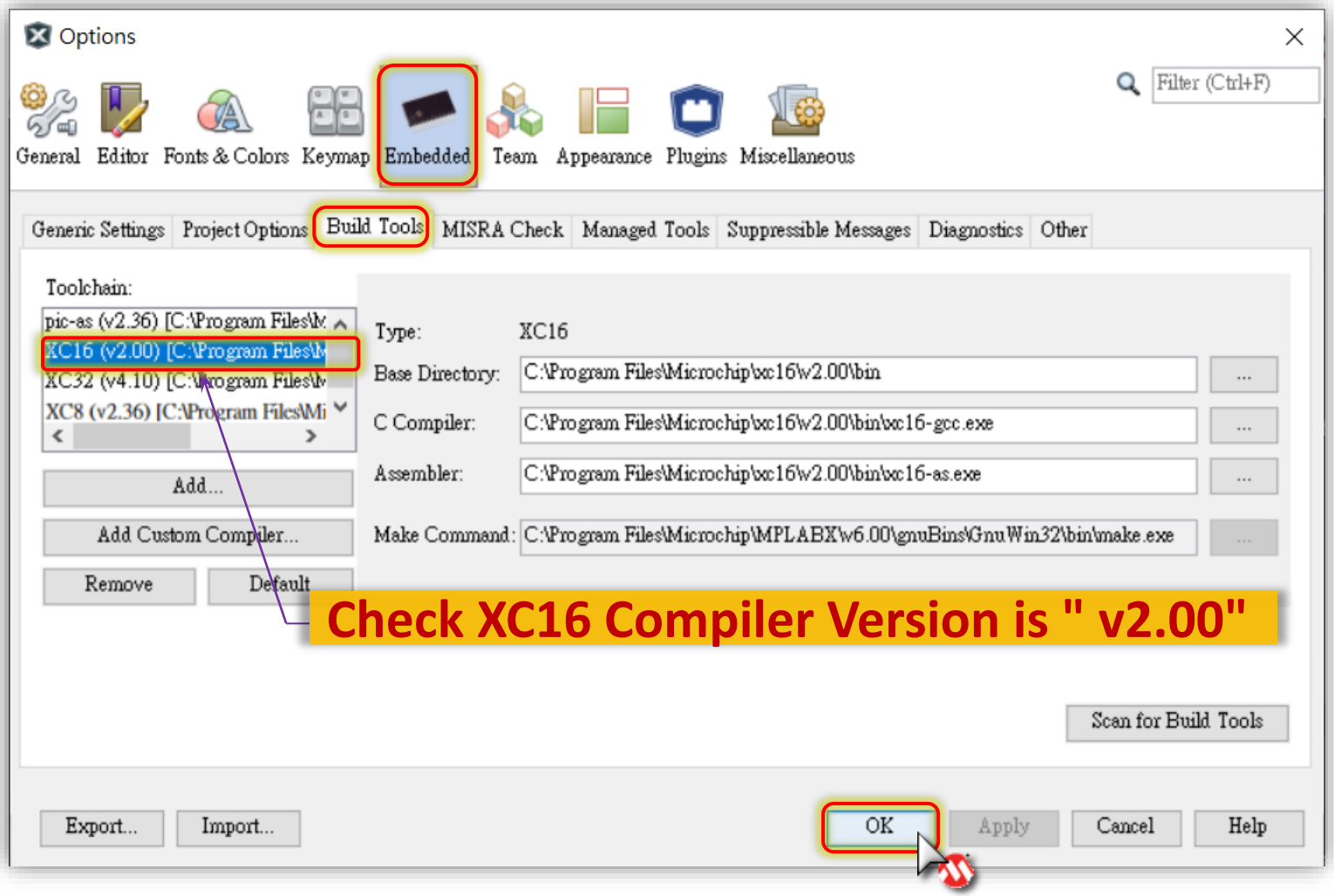
- **Select : Tools > Options**



Lab Preparation

Step 3

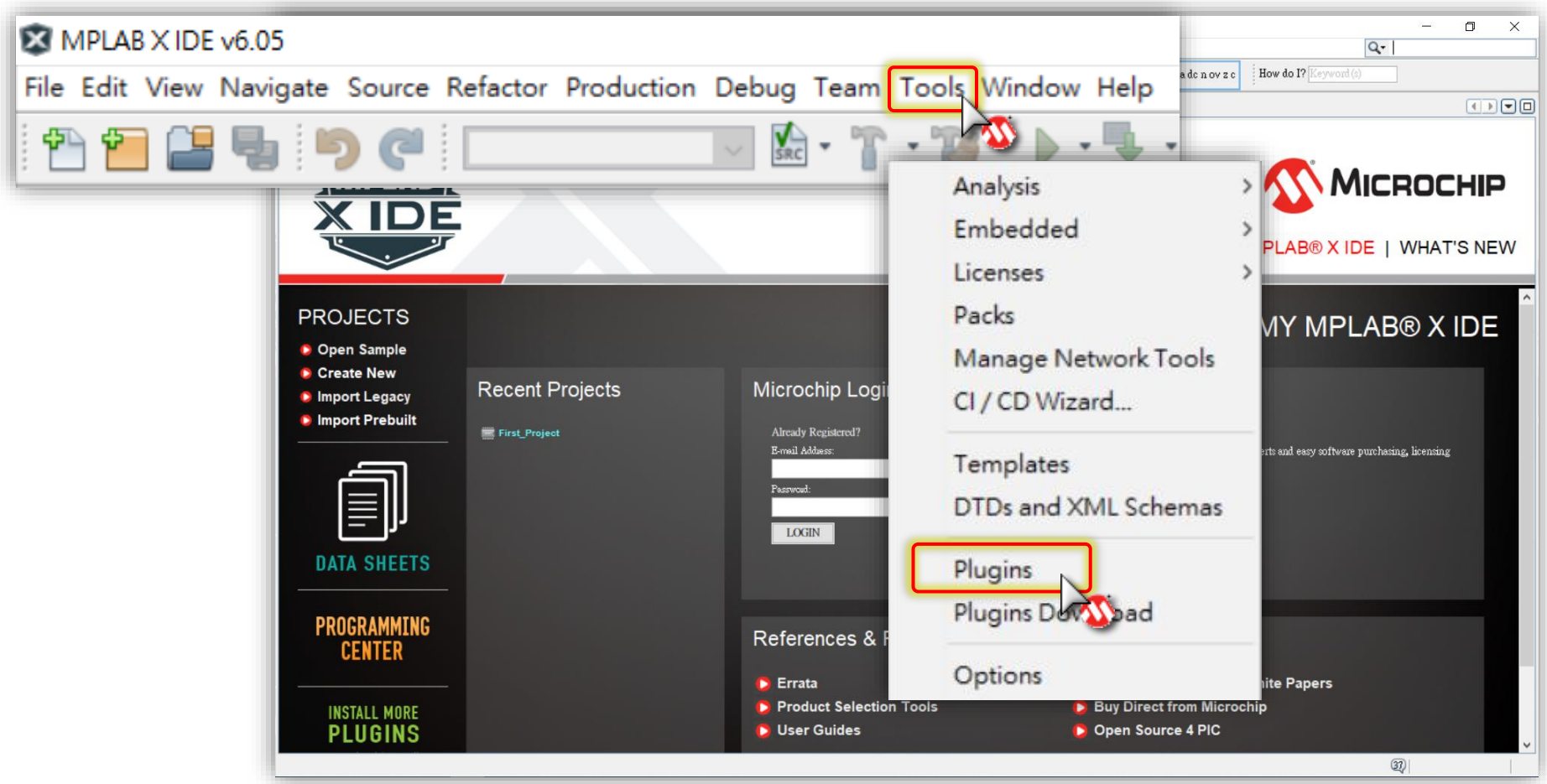
- **Select : Embedded > Build Tools**



Lab Preparation

Step 4

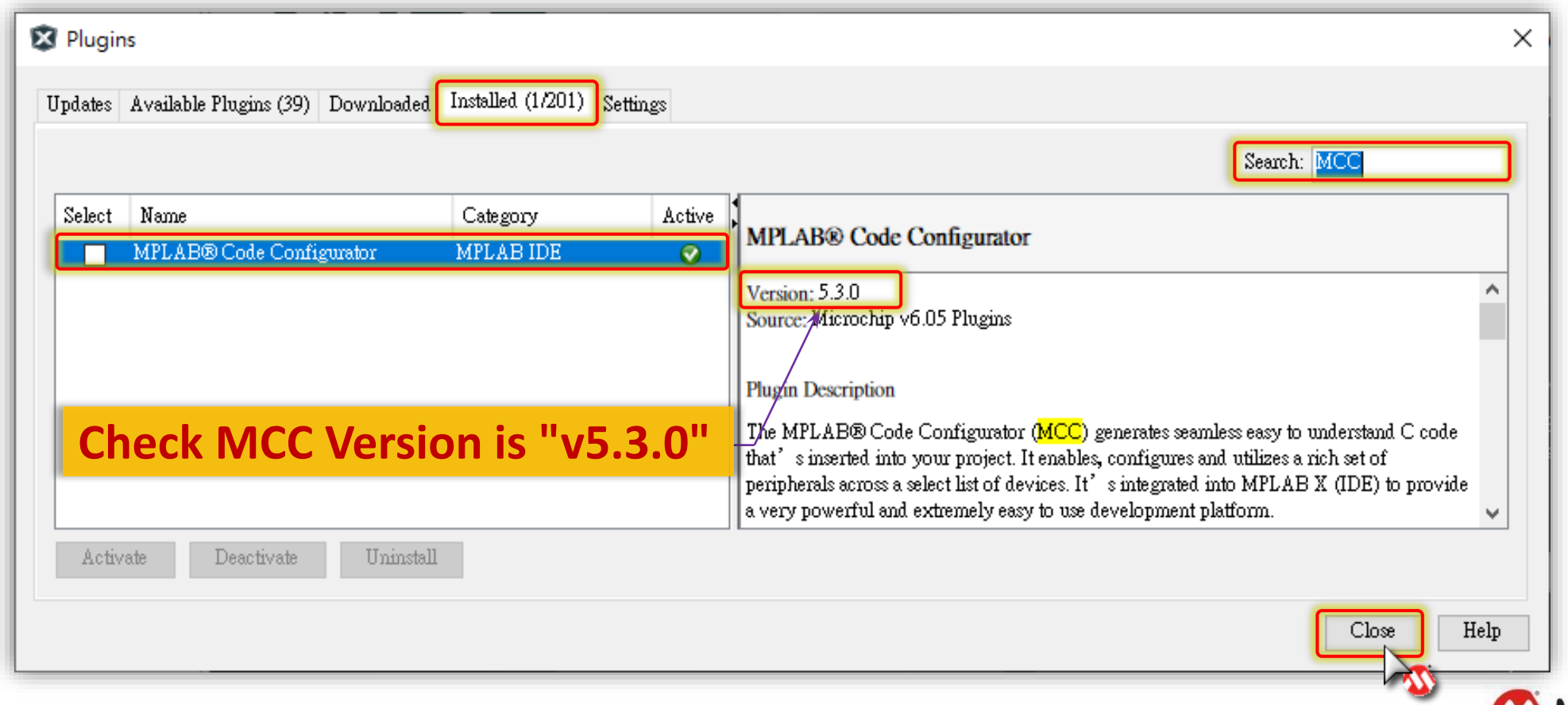
- **Select : Tools > Plugins**



Lab Preparation

Step 5

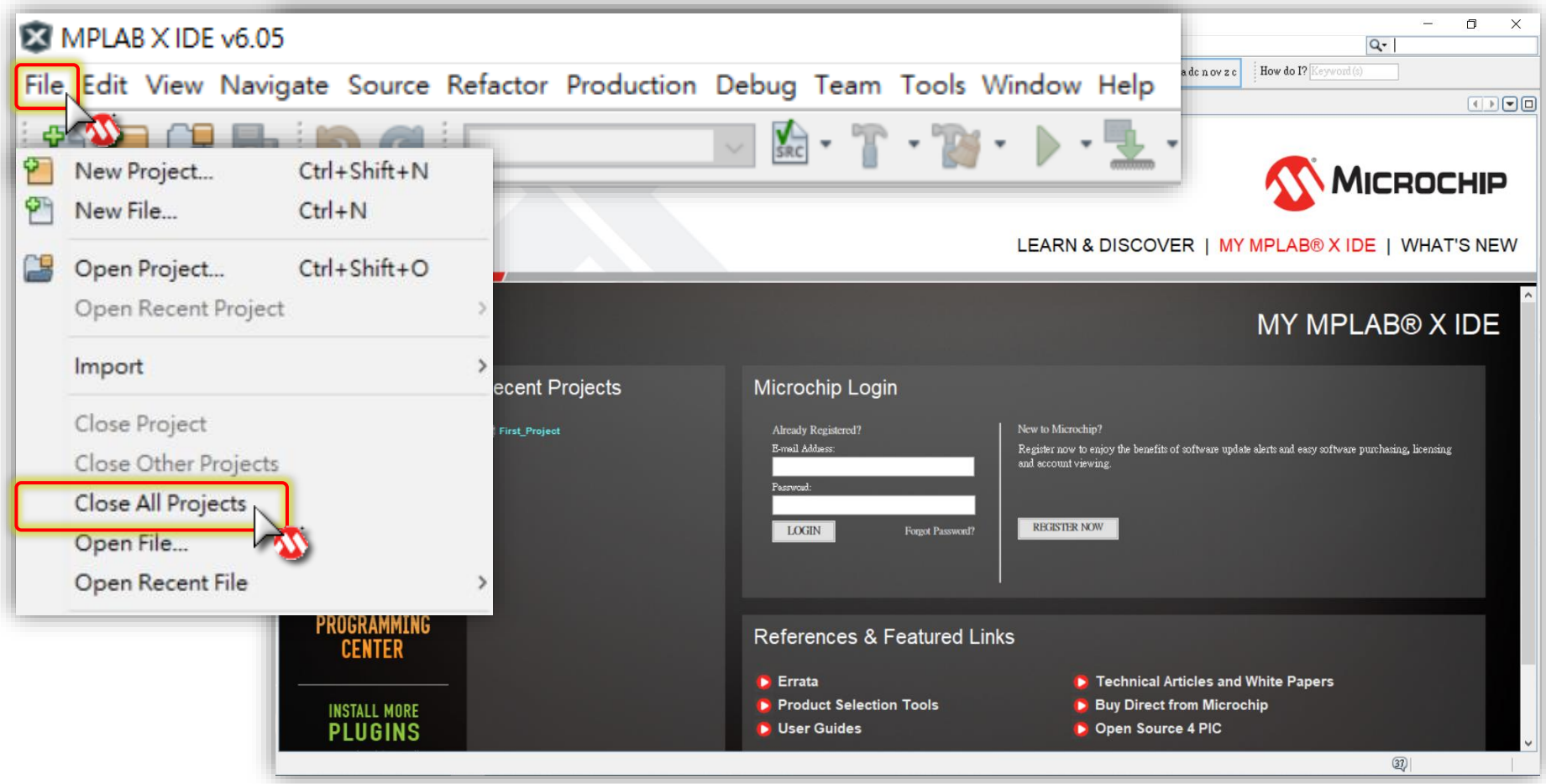
- **Select : Installed**
  - **Search : "MCC"**



Lab Preparation

Step 6

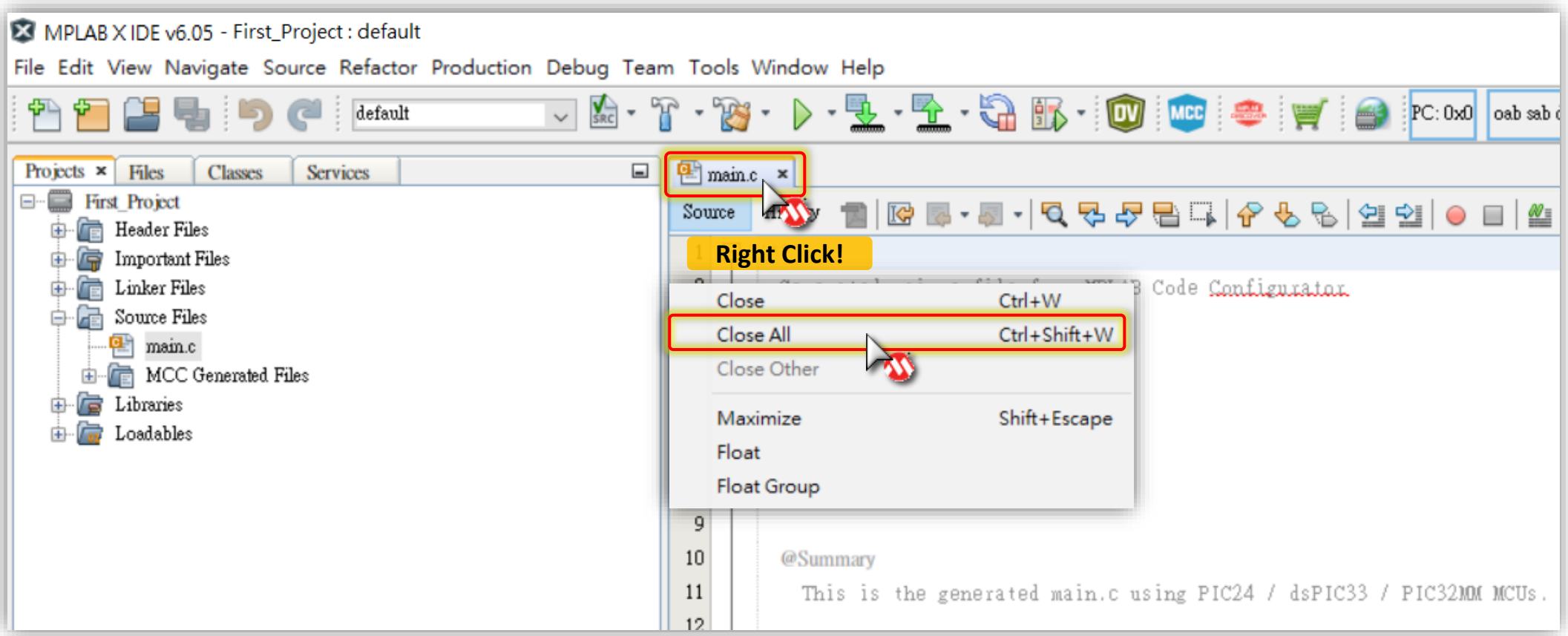
- To prevent opened projects to confuse our main project.
- Please select File > Close All Projects.



Lab Preparation

Step 7

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



Lab0 First Project

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.

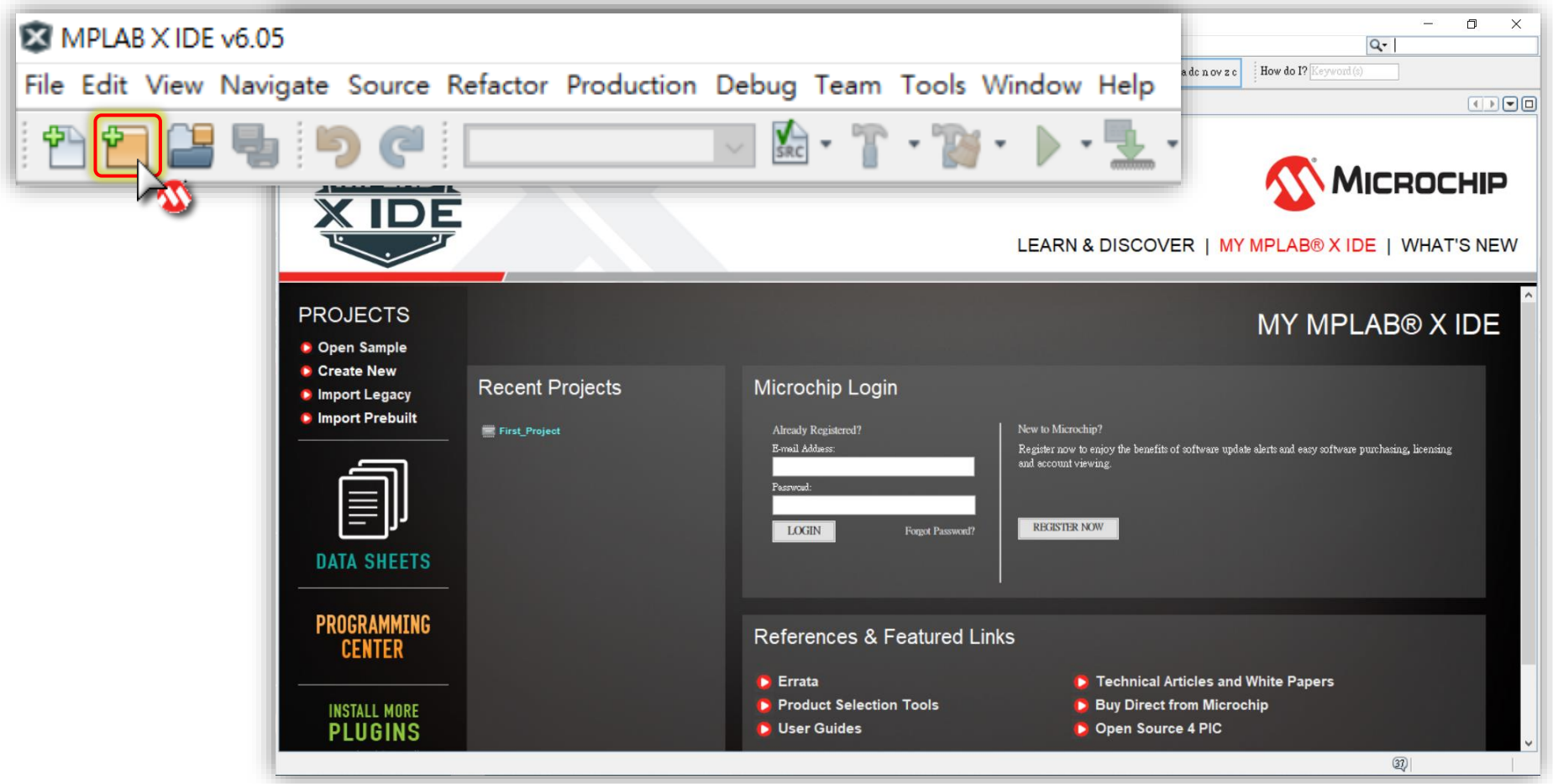
How to start ?

MICROCHIP

Lab0 First Project

Step 1

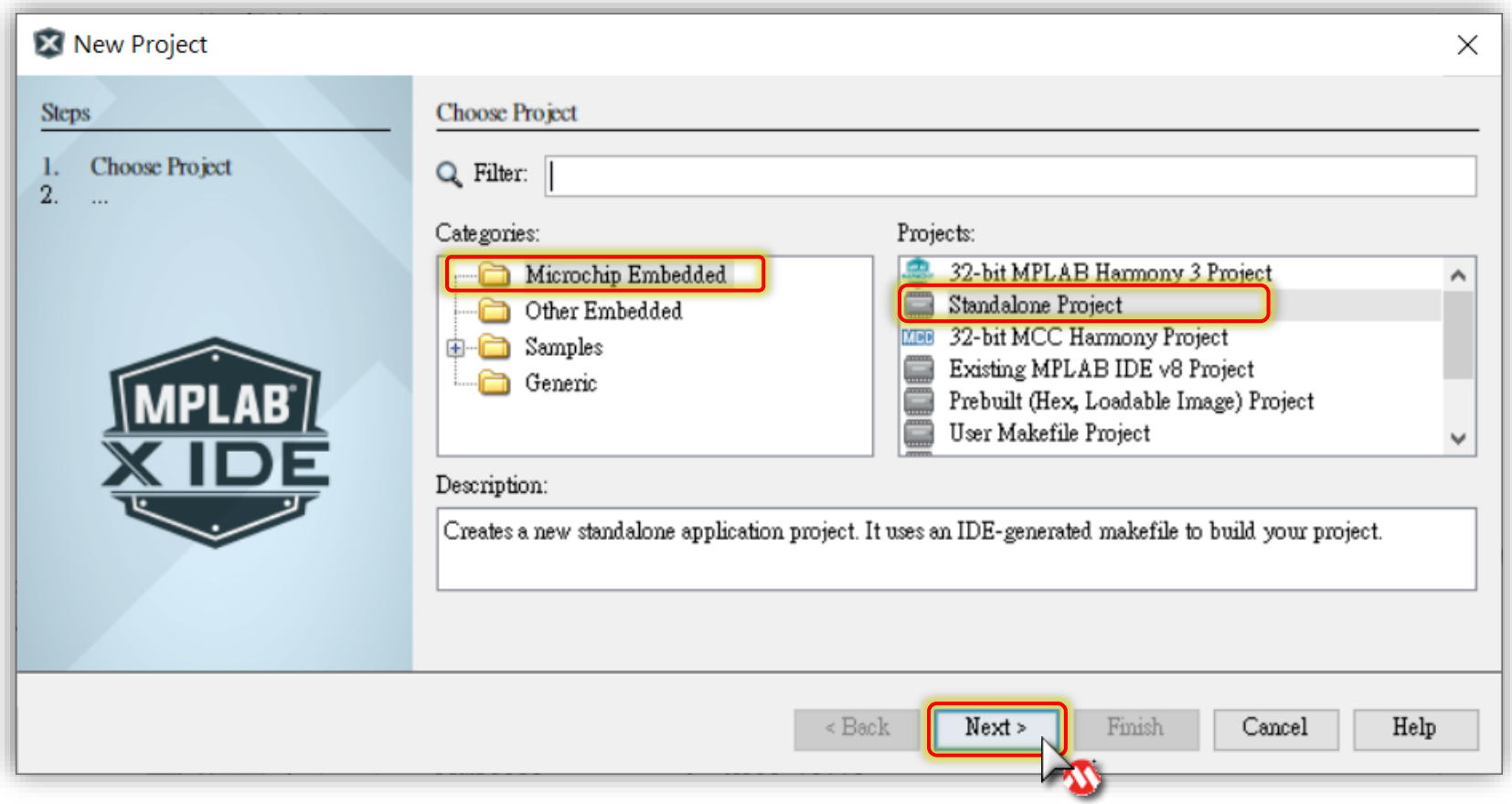
- **Execute Project Wizard**
- **Select : File > New Project or Click icon**



Lab0 First Project

Step 2

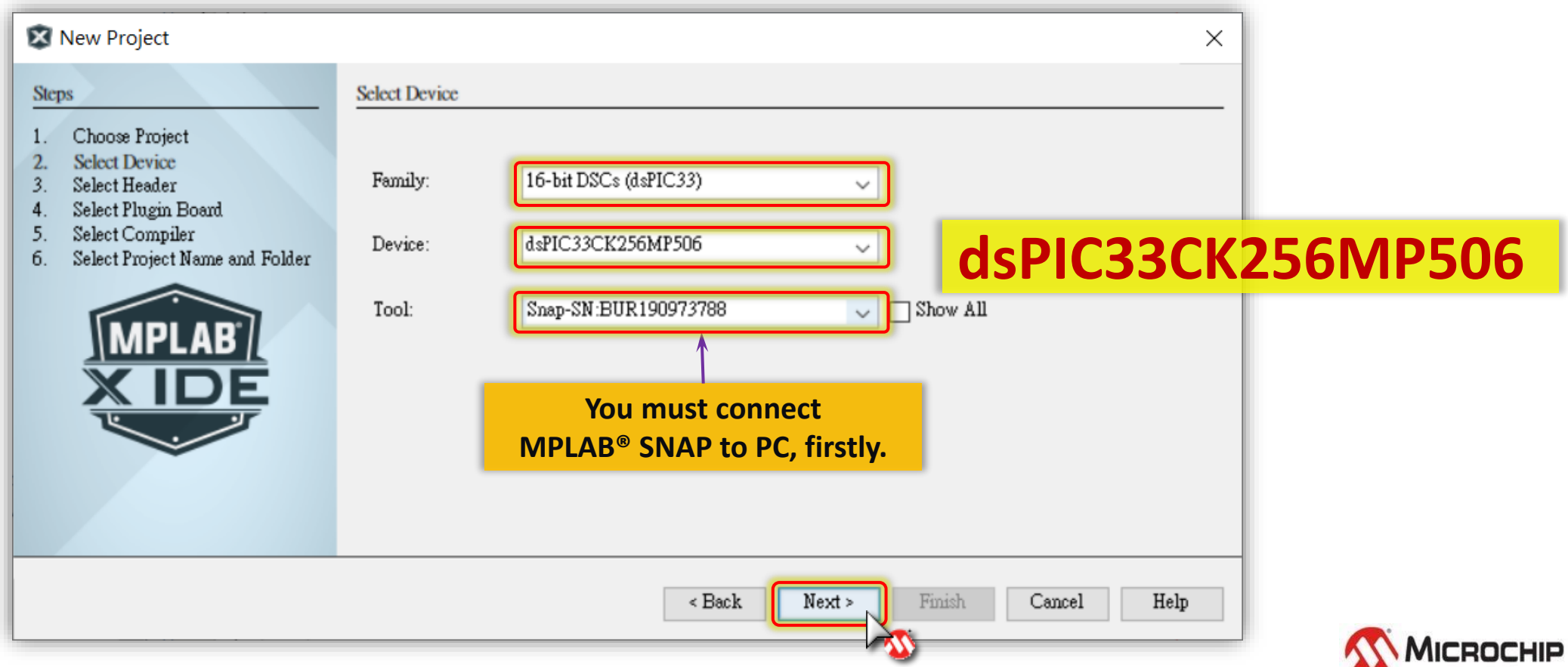
- **Chooses Project :**
  - **Categories : Microchip Embedded , Projects : Standalone Project**



Lab0 First Project

Step 3

- **Select Target Device :**
  - **Family: 16-bit DSCs (dsPIC33) , Device: dsPIC33CK256MP506 , Tool : Snap-SN : BURXXXXXXXXX**

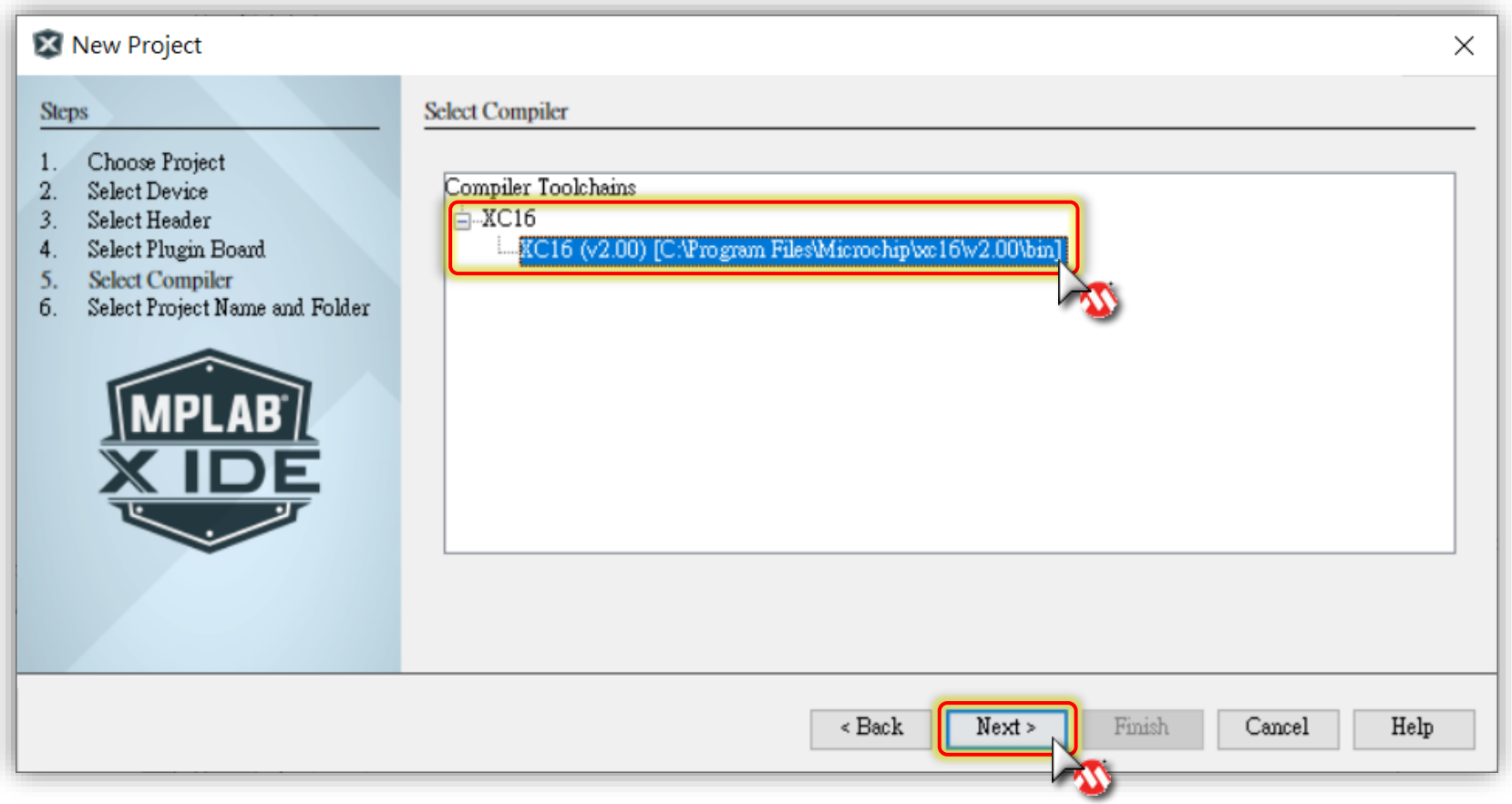


MICROCHIP

Lab0 First Project

Step 4

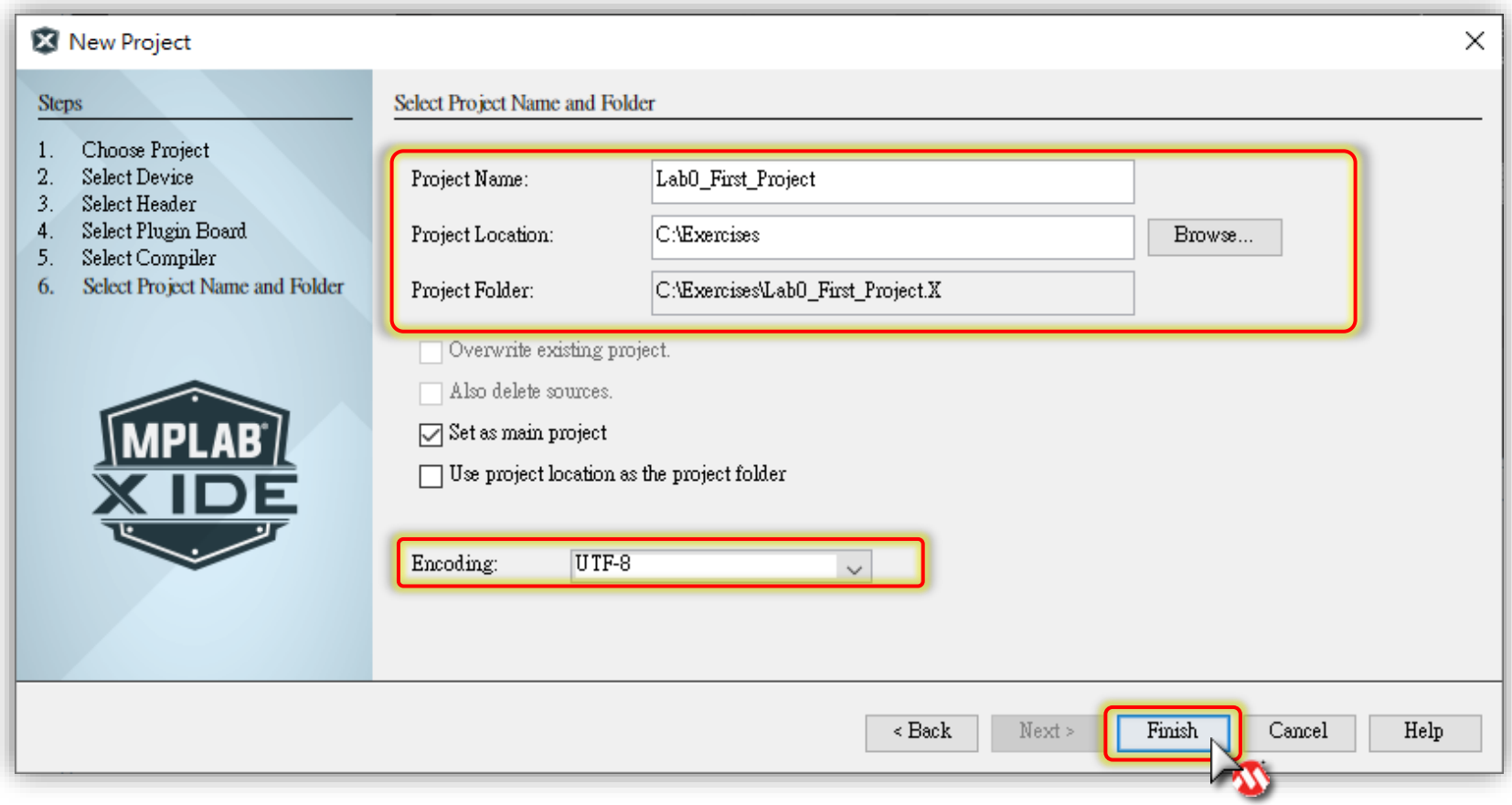
- **Select Compiler :**
  - **Compiler Toolchains : XC16 (v2.00) [C:\Program Files\Microchip\xc16\v2.00\bin**



Lab0 First Project

Step 5

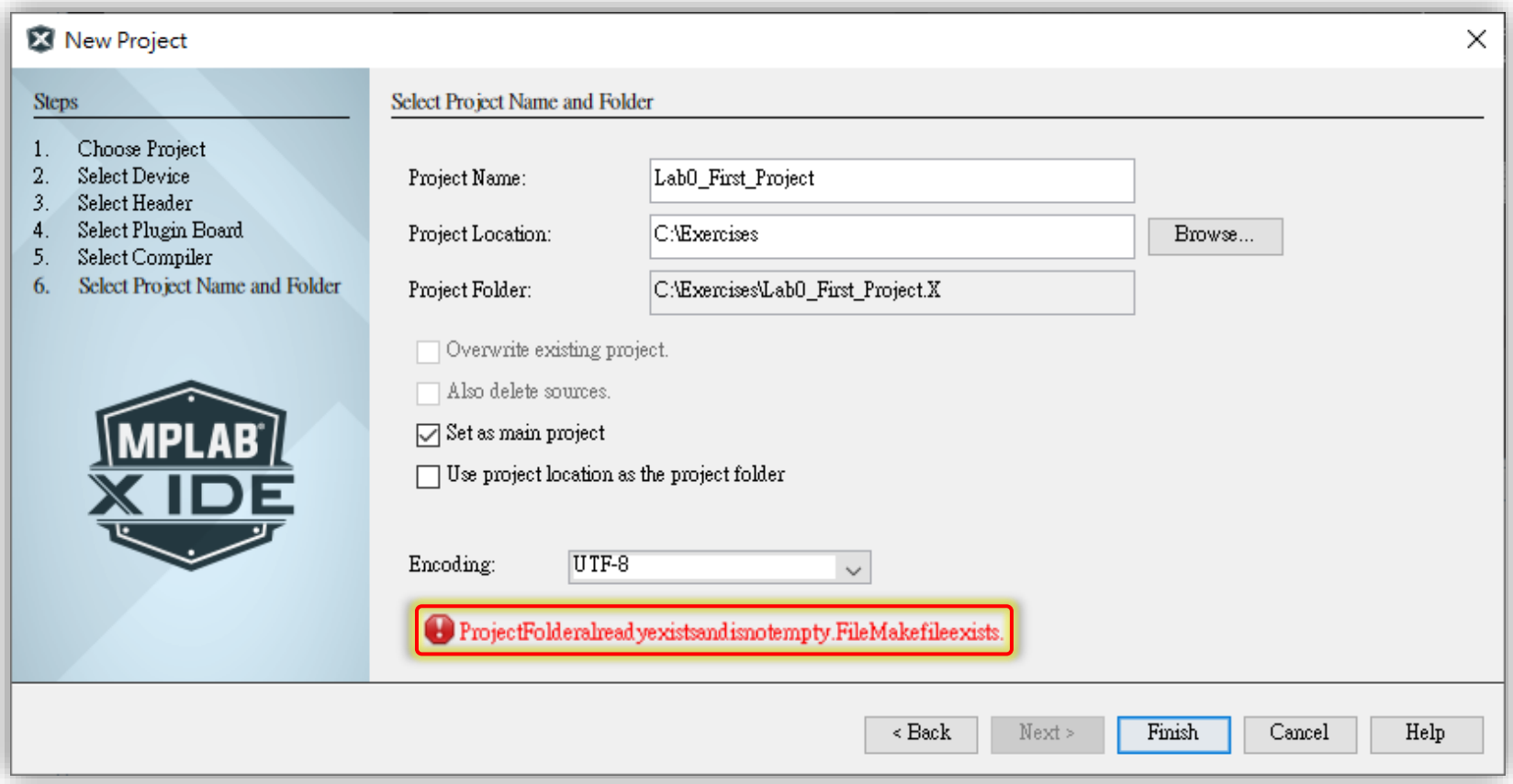
- **Select Project Name and Folder :**
  - **Project Name : Lab0_First_Project , Project Location : C:\Exercises**
  - **Encoding : UTF-8**



Lab0 First Project

Notice

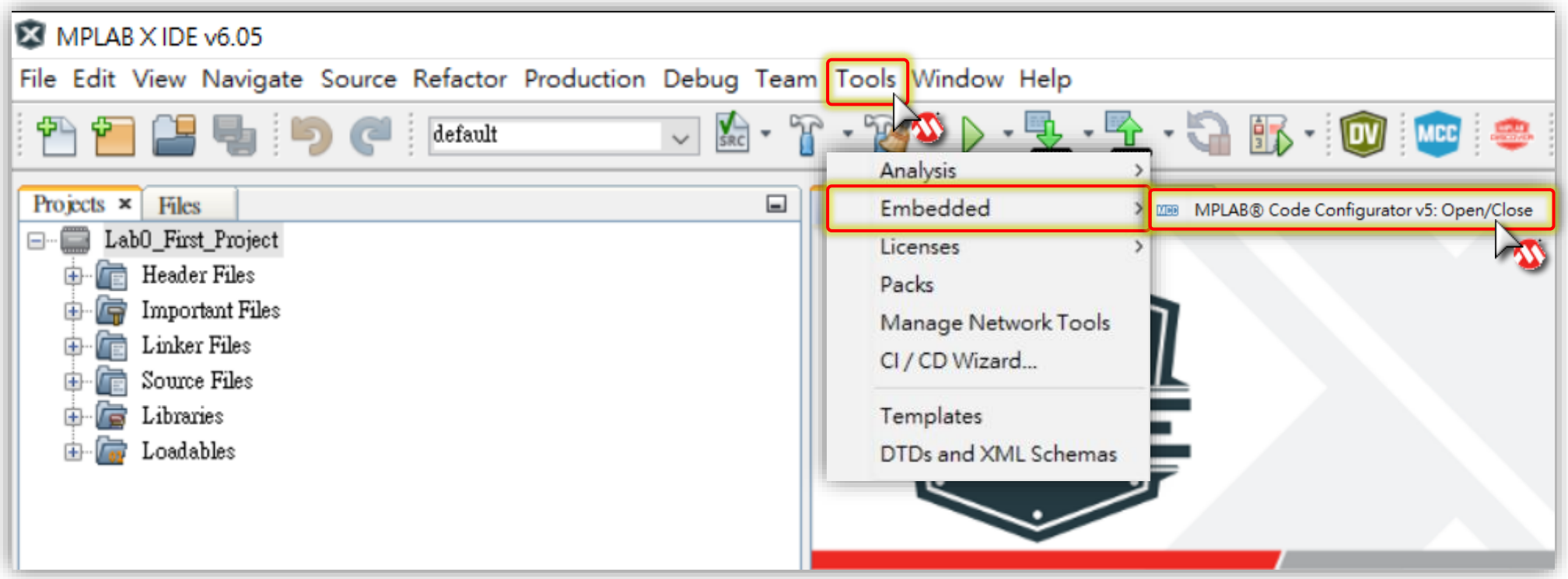
- **If an old project already exist at the same location, you must delete it firstly or change to new project folder.**



Lab0 First Project

Step 6 (Menu)

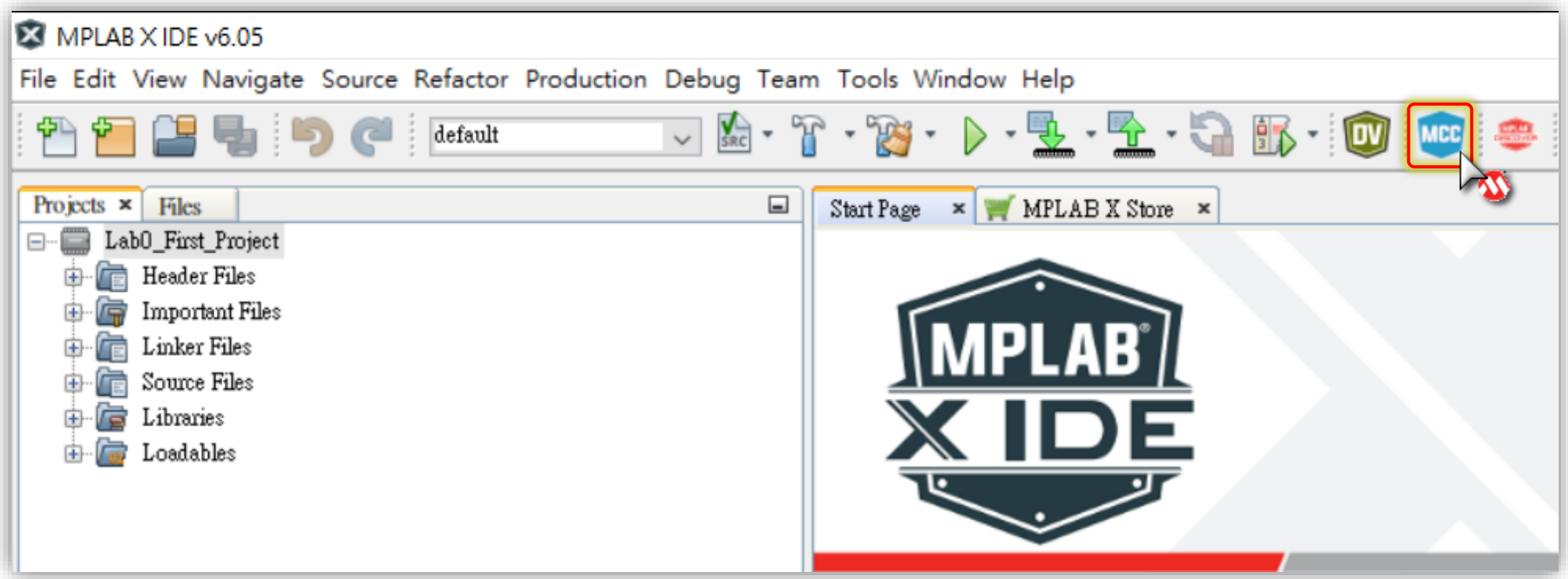
- **Execute MCC.**
  - **Tools > Embedded > MPLAB Code Configurator v5: Open/Close**



Lab0 First Project

Step 6 (Icon)

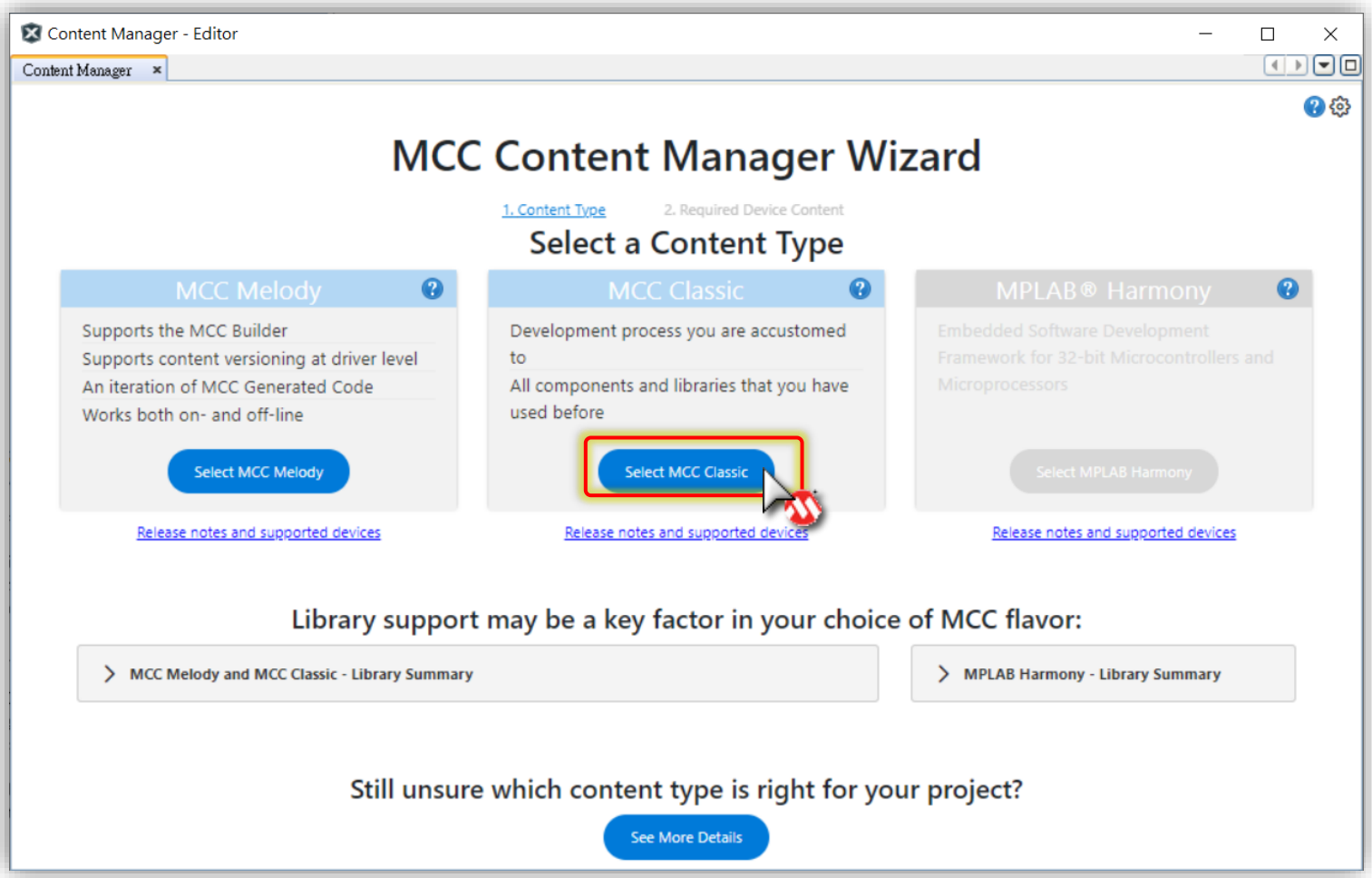
- **Execute MCC.**
  - **Click** 



Lab0 First Project

Step 7

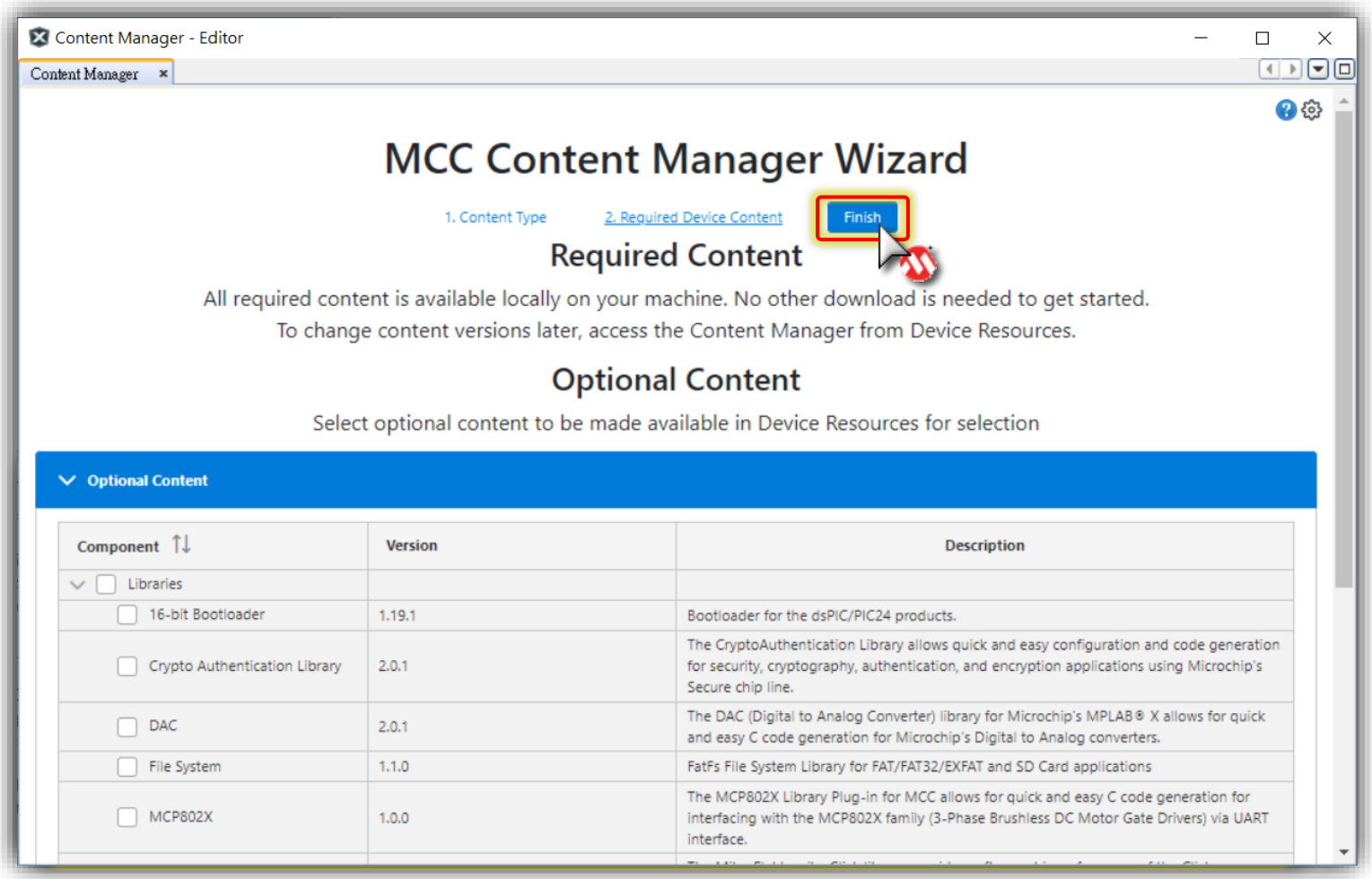
- "**Select MCC Classic**"



Lab0 First Project

Step 8

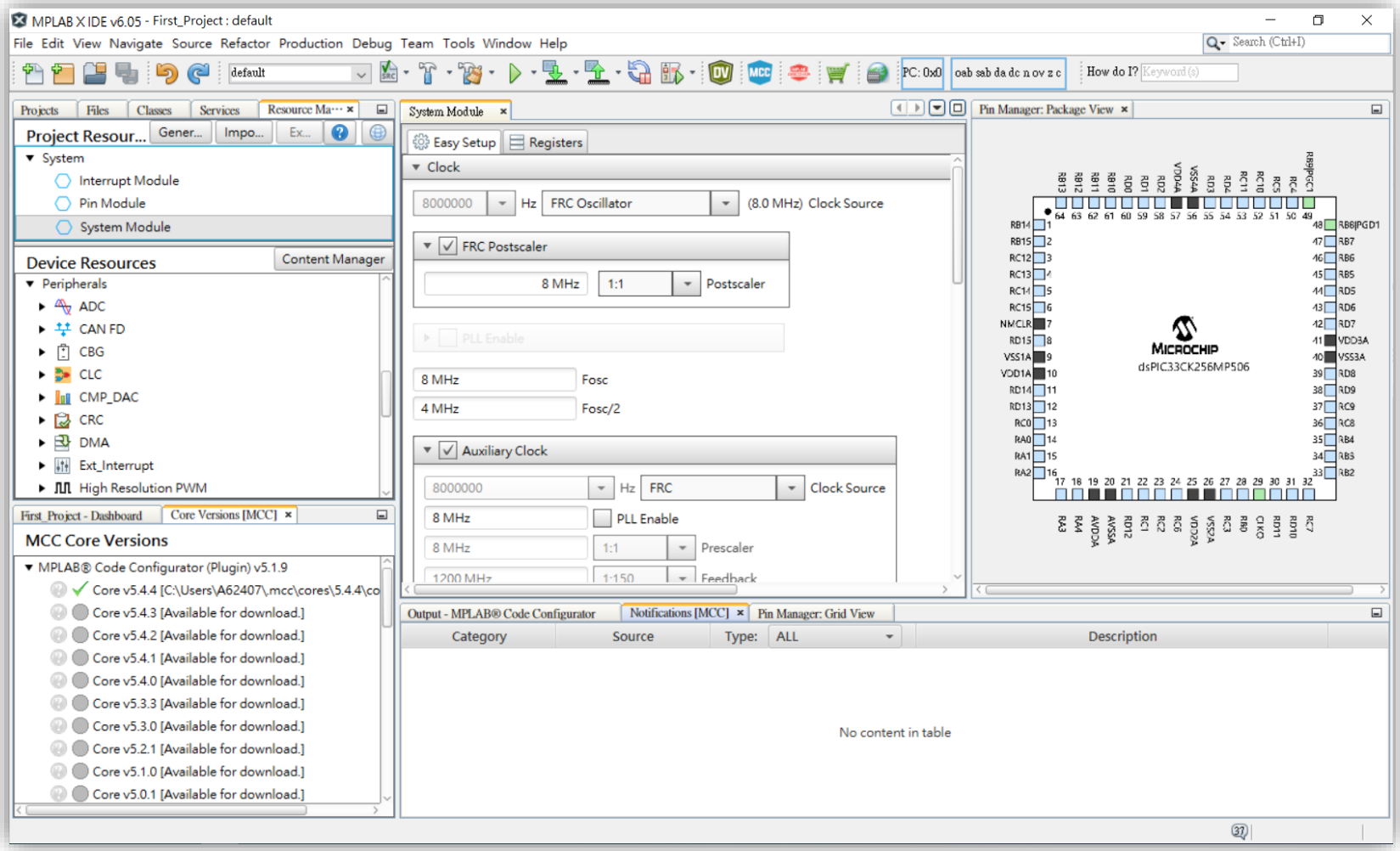
- You can add the available Libraries provided in the Device resource to the project or click "Finish".



Lab0 First Project

Step 9

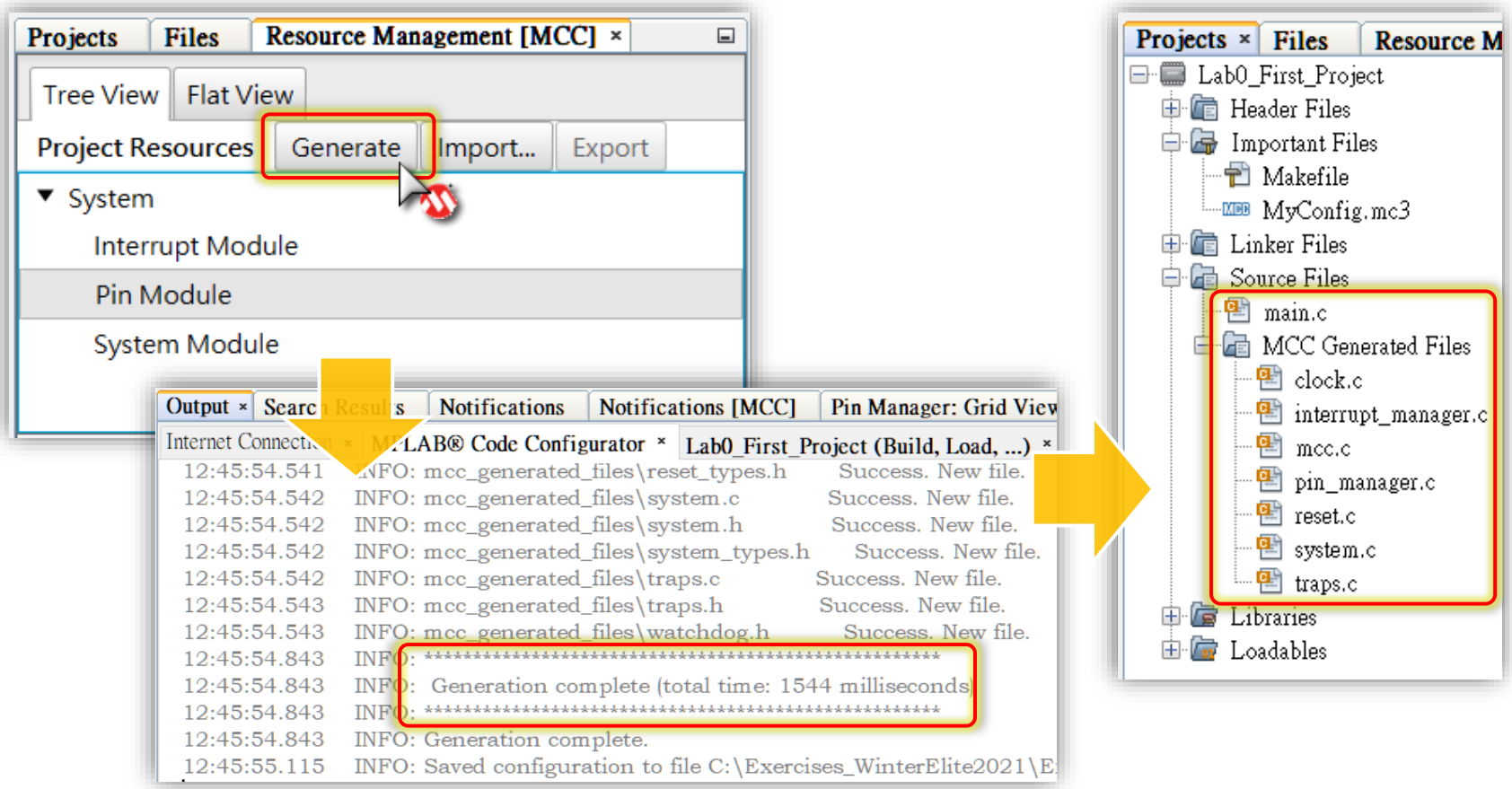
- **MCC Configurator interface show up.**



Lab0 First Project

Step 10

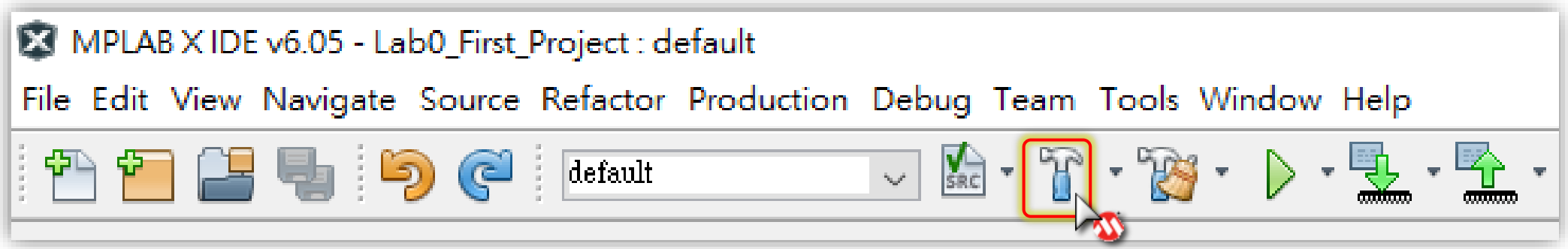
- Click "Generate" Button to get your first MCC Project.
- MCC to generate files include "main.c" automatically.

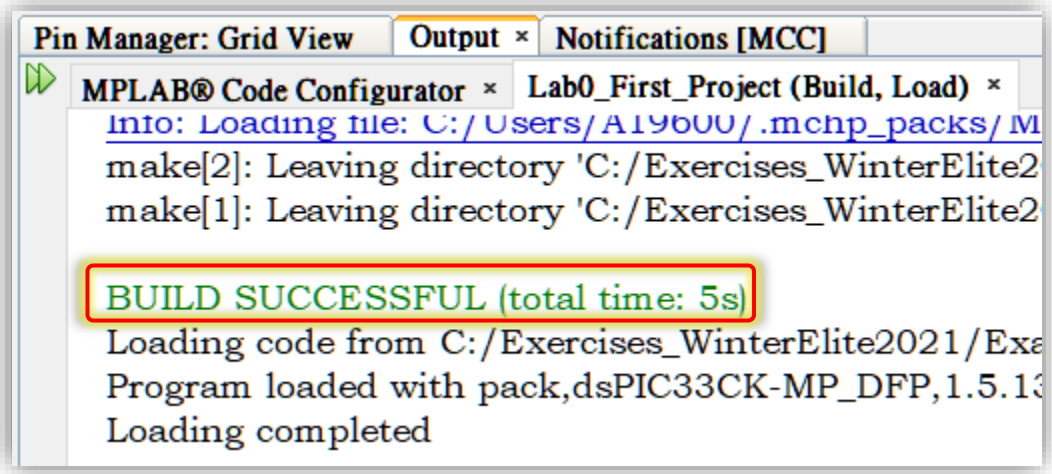


Lab0 First Project

Step 11

- **Build your first MCC Project.**
  - **Select "Build Main Project" icon**
  - **Make sure compiled result is BUILD SUCCESSFUL**



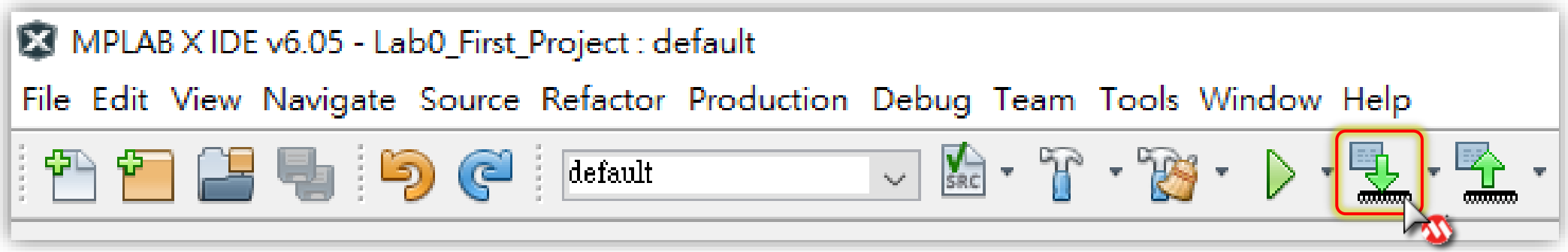


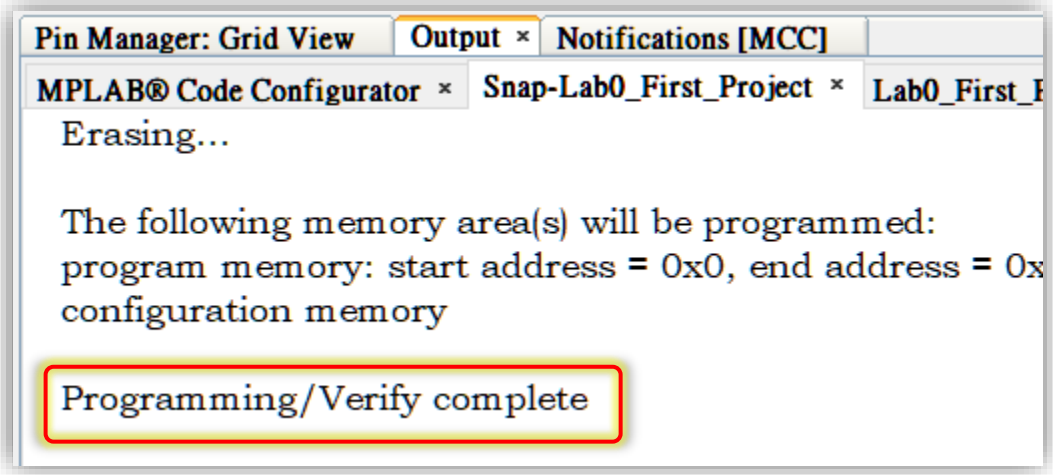
MICROCHIP

Lab0 First Project

Step 12

- Try program compiled firmware to your target board.
  - Select "Make & Program Device Main Project" icon
  - Make sure the result is Programming/Verify complete



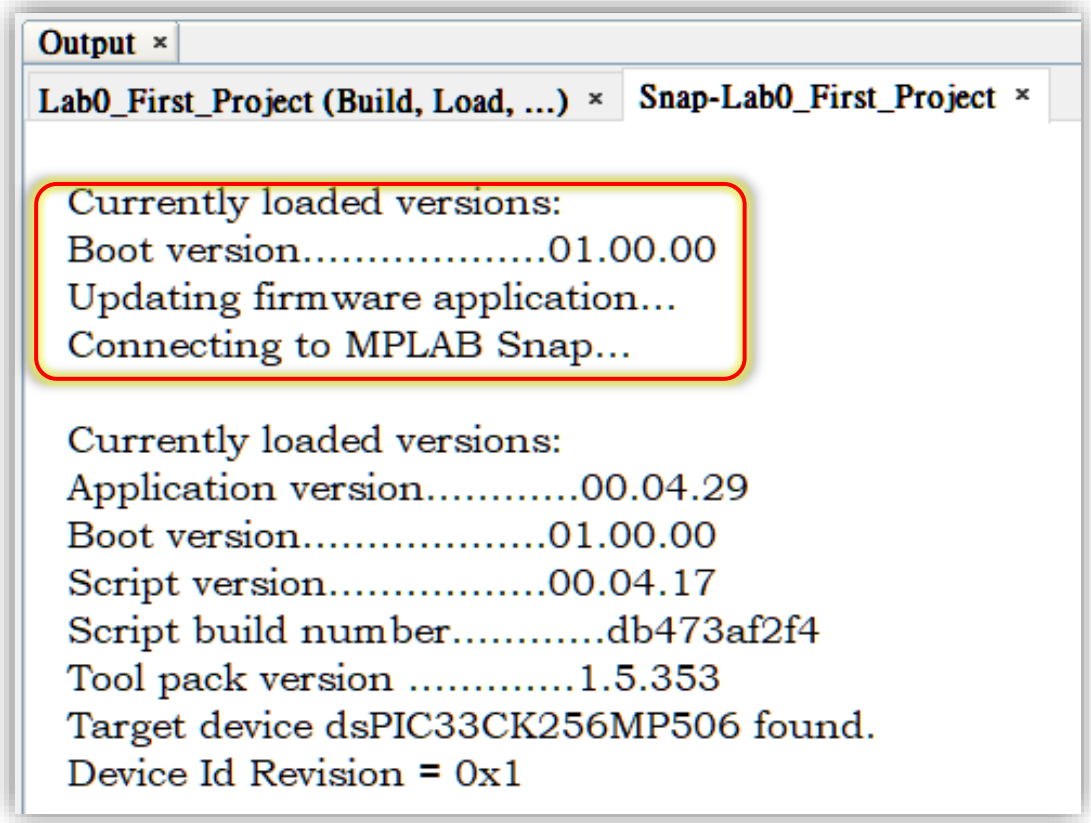


MICROCHIP

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.**

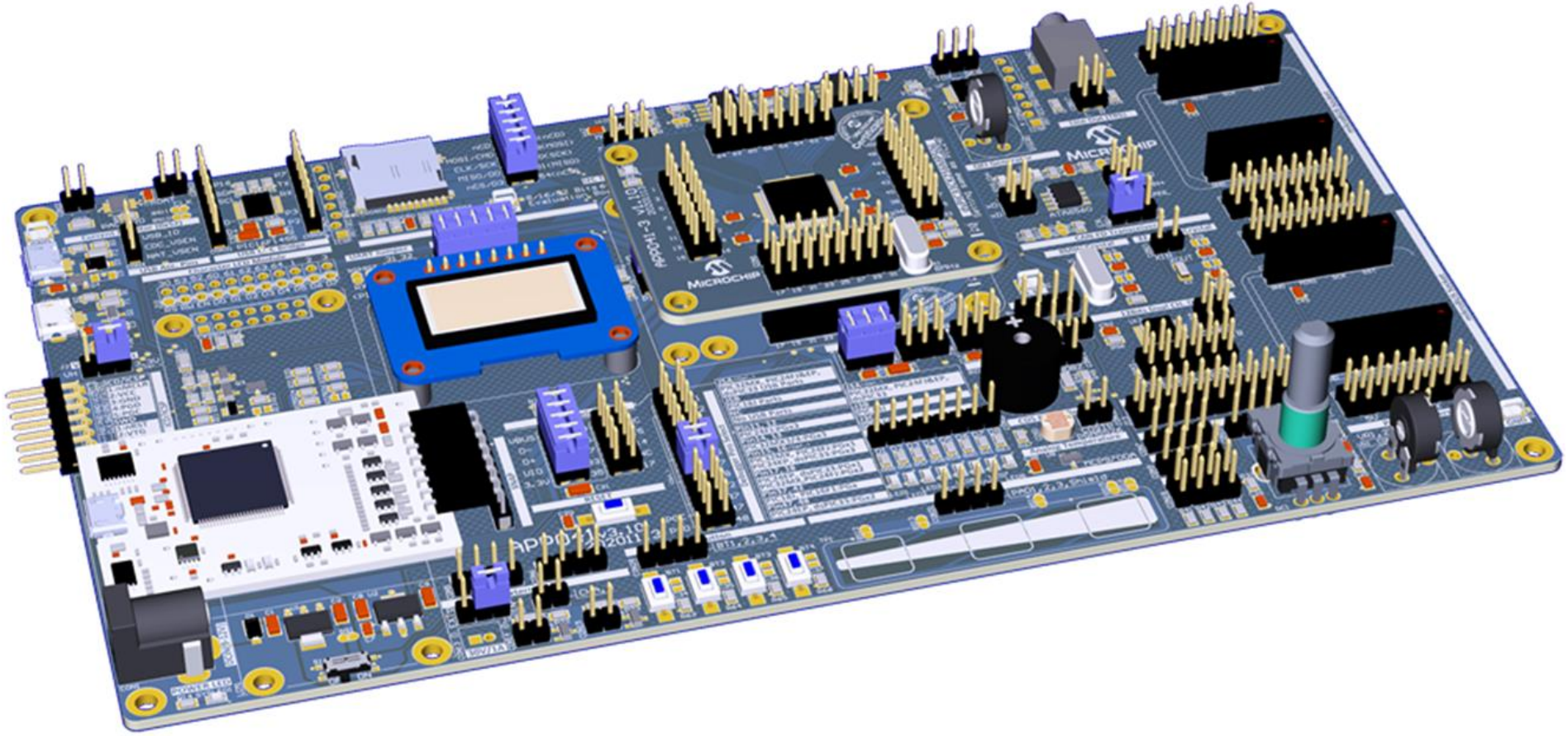


MICROCHIP

Lab0 First Project

Check Point

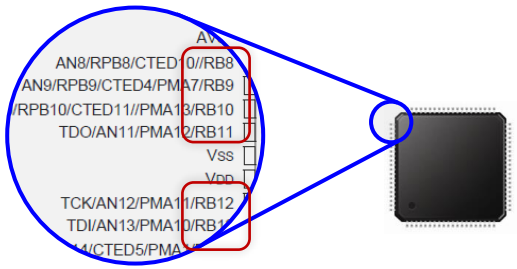
Nothing happen ????

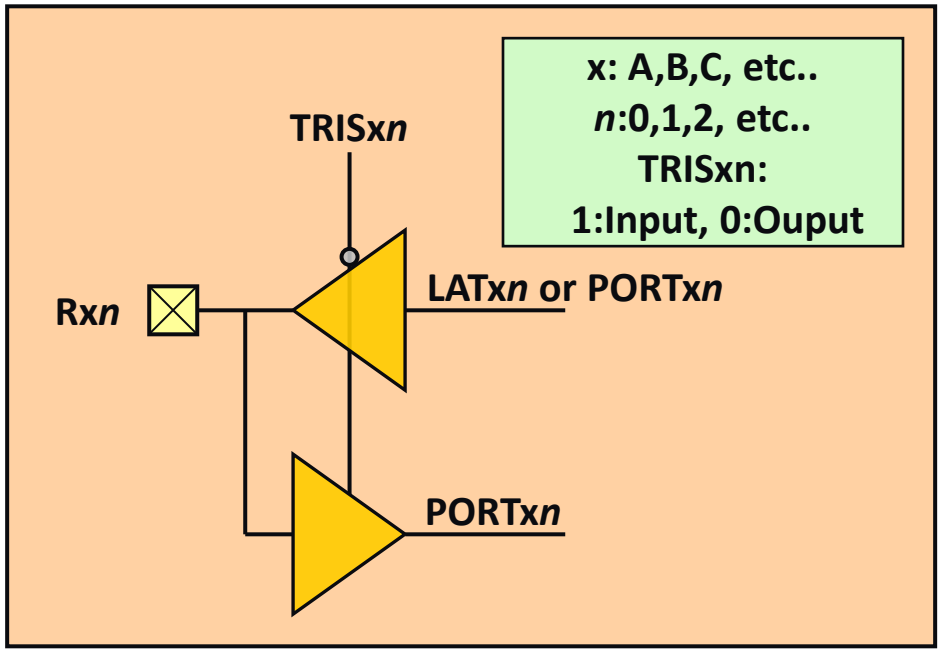


GPIO Architecture

GPIO Block Diagram

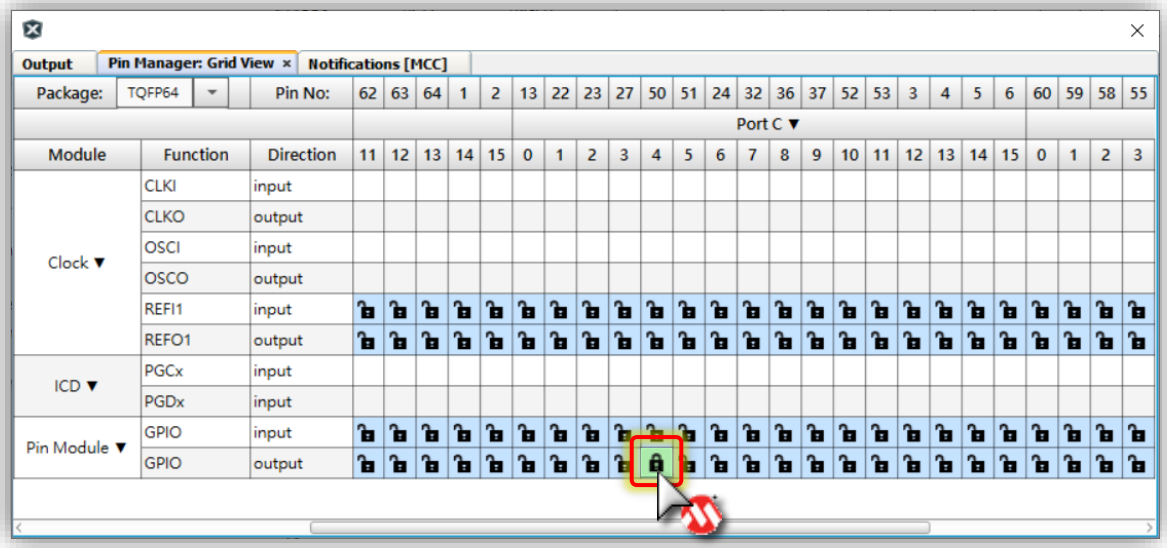
- Please refer to the block diagram, this is the concept for programming model. The real and detail block diagram please refer to the following slide.
- TRIS: To determine Input or Output.
  1 for Input, 0 for Output.
- LAT: set output drive
  value for Output pins.
- PORT: reaction logical
  level from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

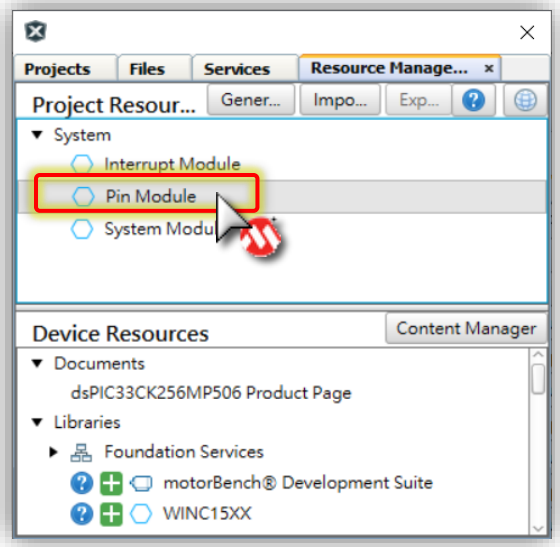


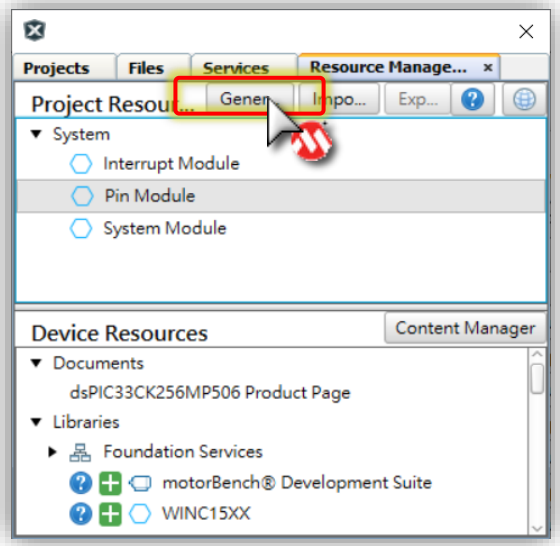


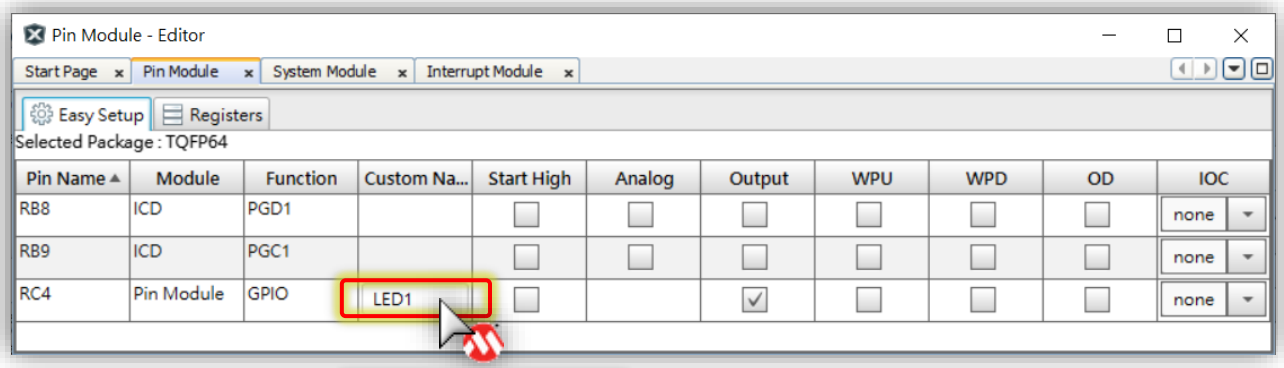
GPIO Setting of MCC

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









Click & Rename

GPIO Functions of MCC

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

  Prototype definition : "pin_manager.h"

  - **void PIN_MANAGER_Initialize (void);**
  - **OUT operation**
    - ***CustomName*_SetHigh();**
    - ***CustomName* _SetLow();**
    - ***CustomName* _Toggle();**
    - ***CustomName* _SetDigitalOutput();**
  - **IN operation**
    - ***CustomName* _GetValue();**
    - ***CustomName* _SetDigitalInput();**

MICROCHIP

Lab1 GPIO Output

- Base on current project or Open .\Exams\Lab1_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 RB6 to digital output mode and toggle output level in period 500ms ~ 1S roughly.
- Please connect RB6 (Pin11) to LED (LED1) to observe pin status.

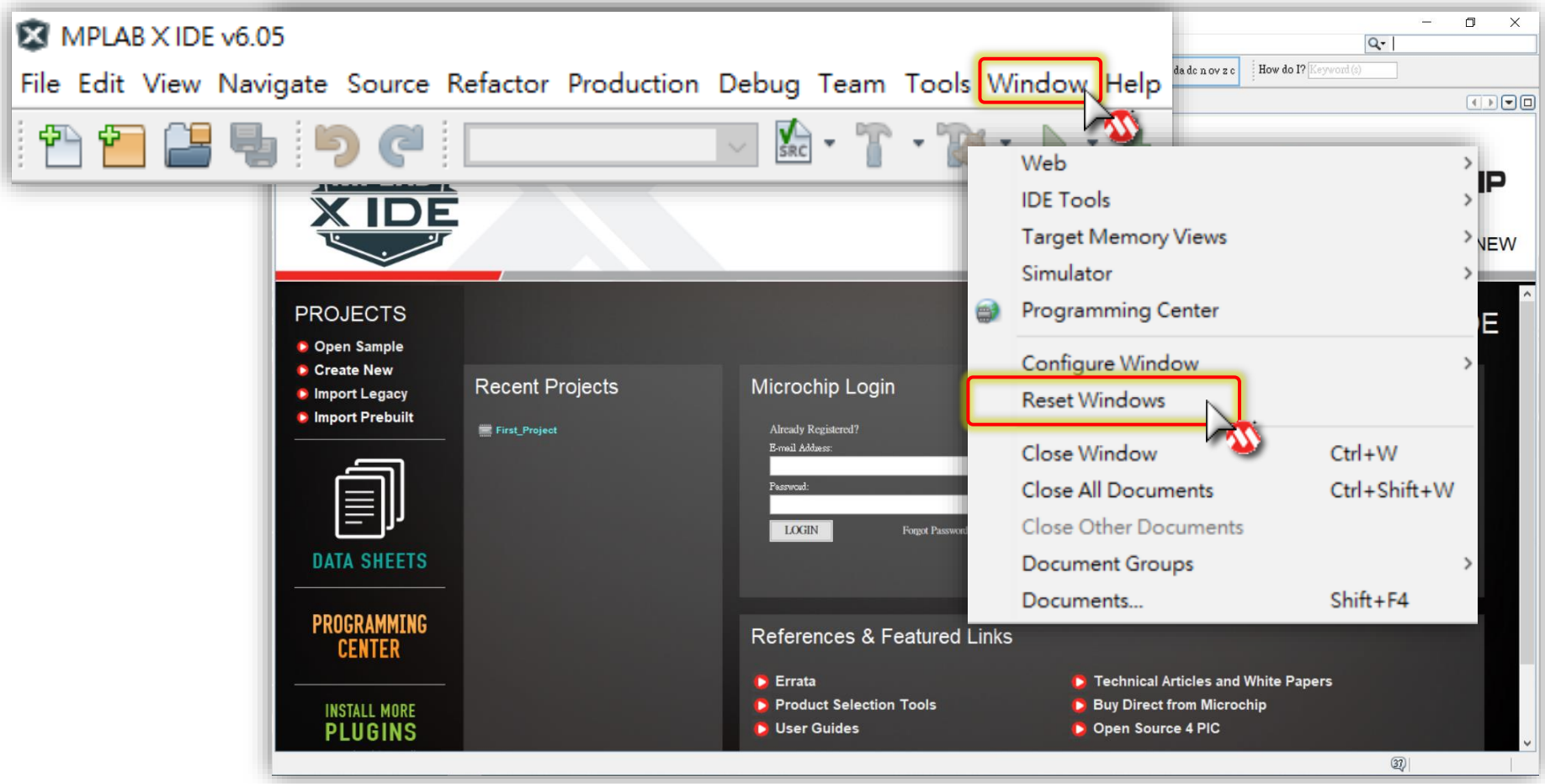
How to start ?

MICROCHIP

MPLAB X IDE

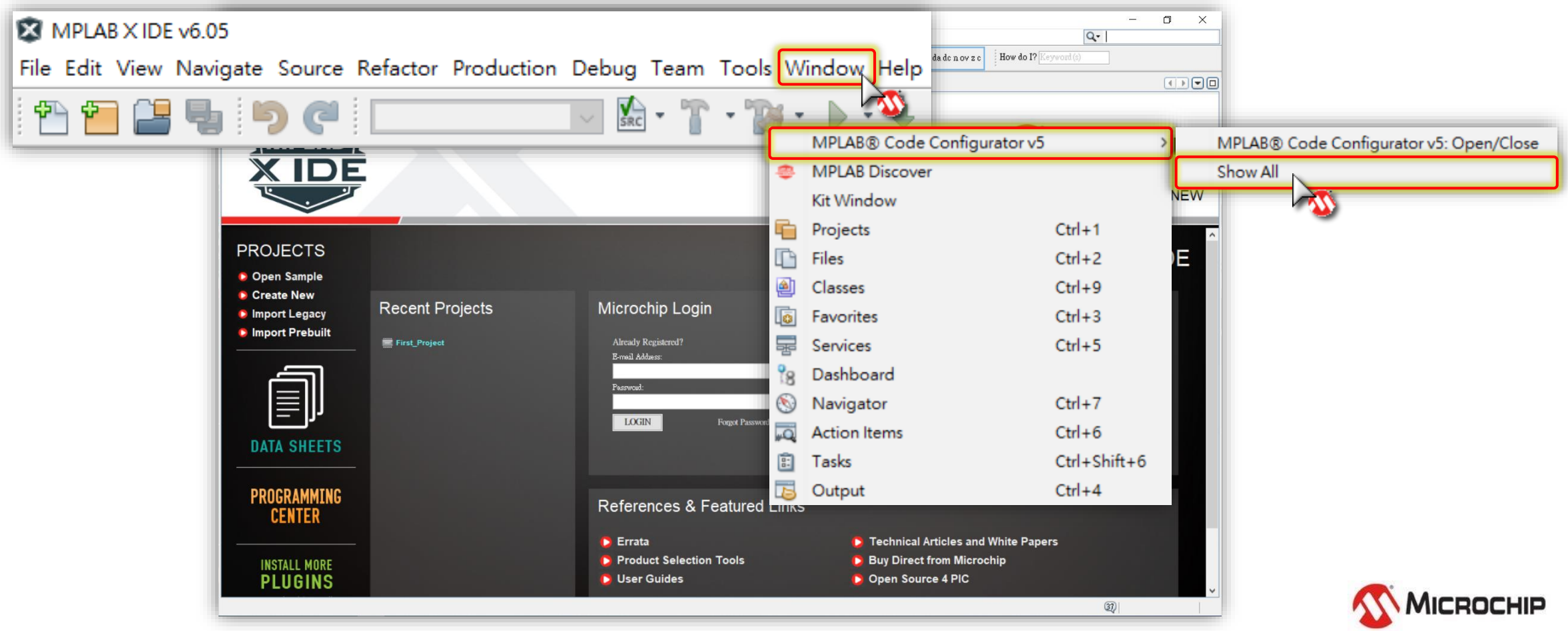
Tips

- **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

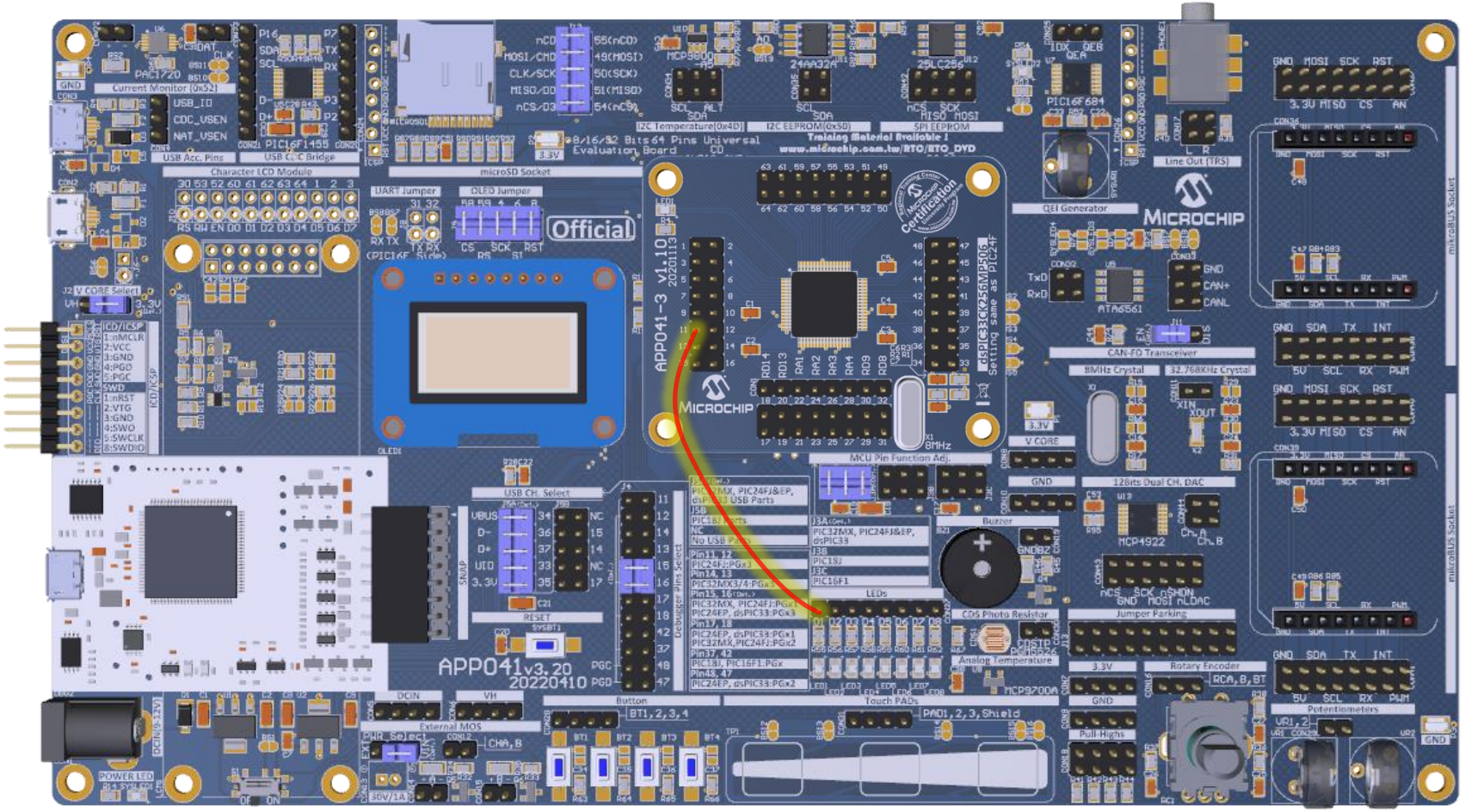
- **You may can’t find the MCC window you want also. MCC provide named "Show All" to show all MCC windows.**



Lab1 GPIO Output

Connection

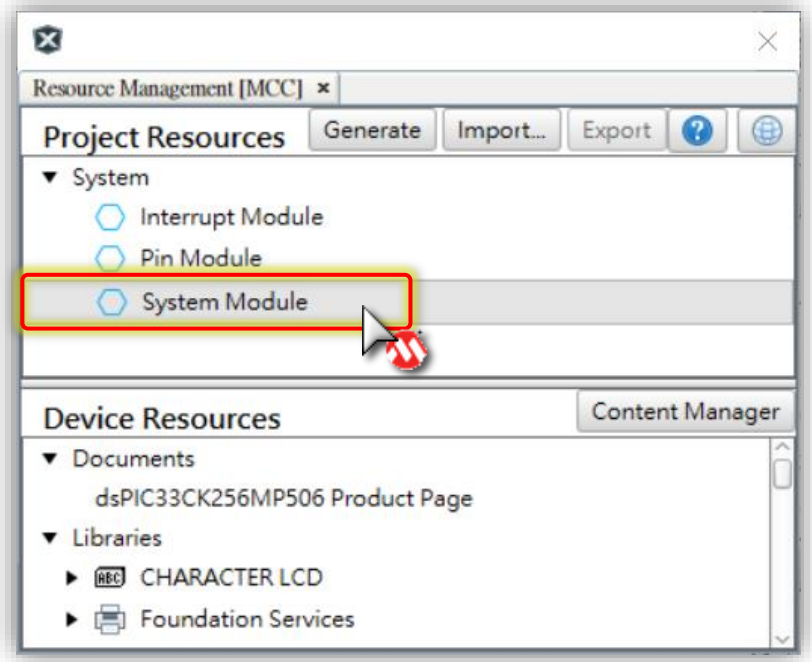
- Please connect RB6 (Pin11) to LED (D1).

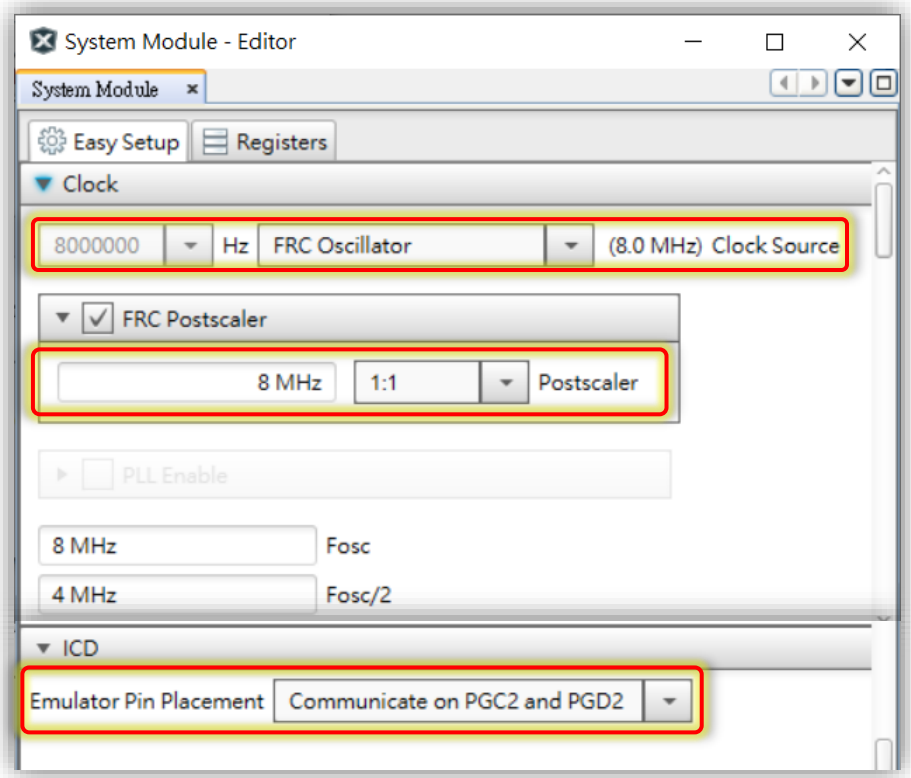


Lab1 GPIO Output

Step 1

- **Set System Clock & Debug Interface.**
  - **System Module : Clock : FRC Oscillator, Postscaler 1:1.**
  - **System Module : ICD : PGC2 and PGD2.**

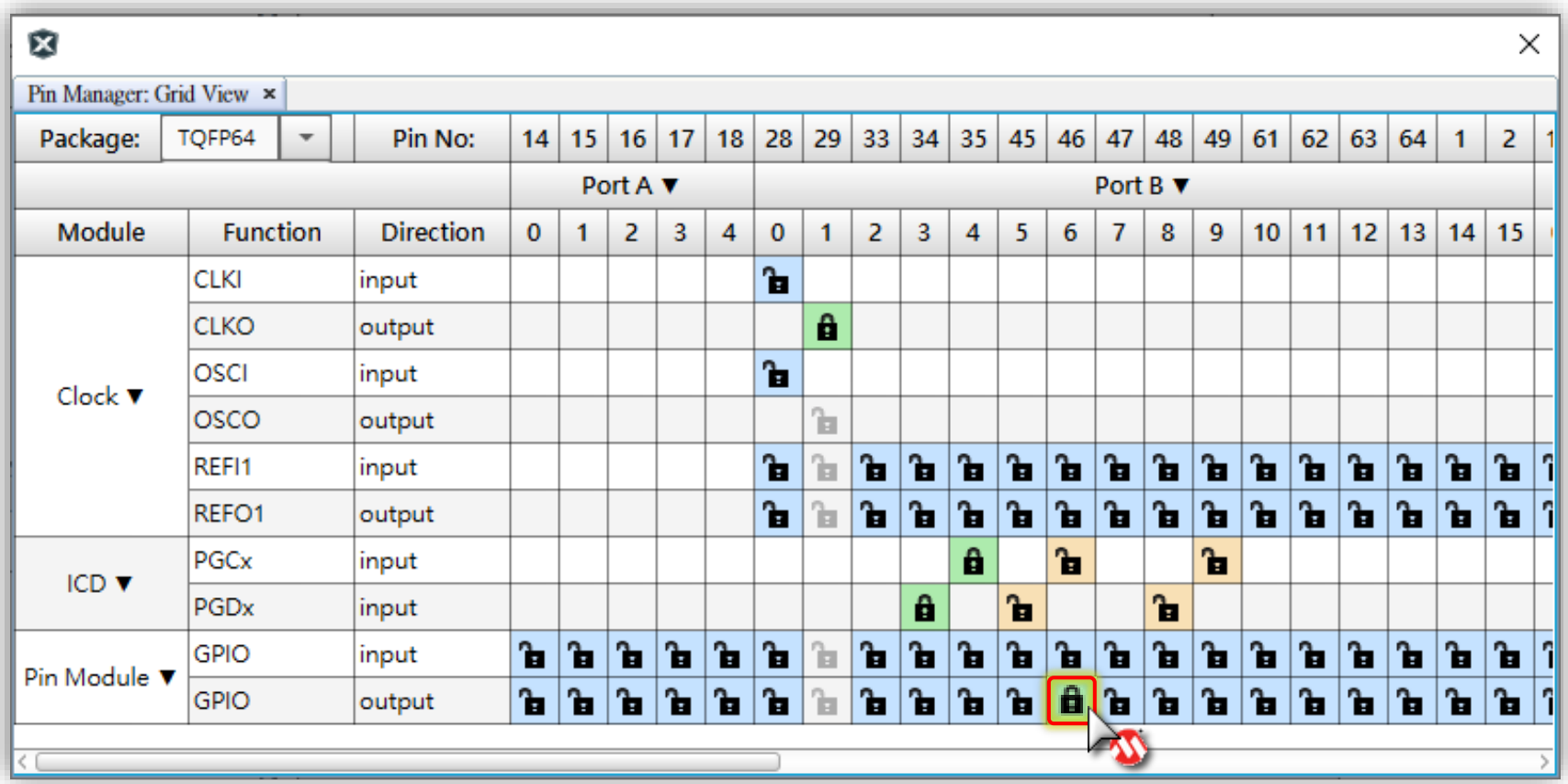




Lab1 GPIO Output

Step 2

- Set PORTB : Pin6 (RB6) to digital output mode.
  - Pin Manager Gird View : Click RB6 to lock to GPIO output.

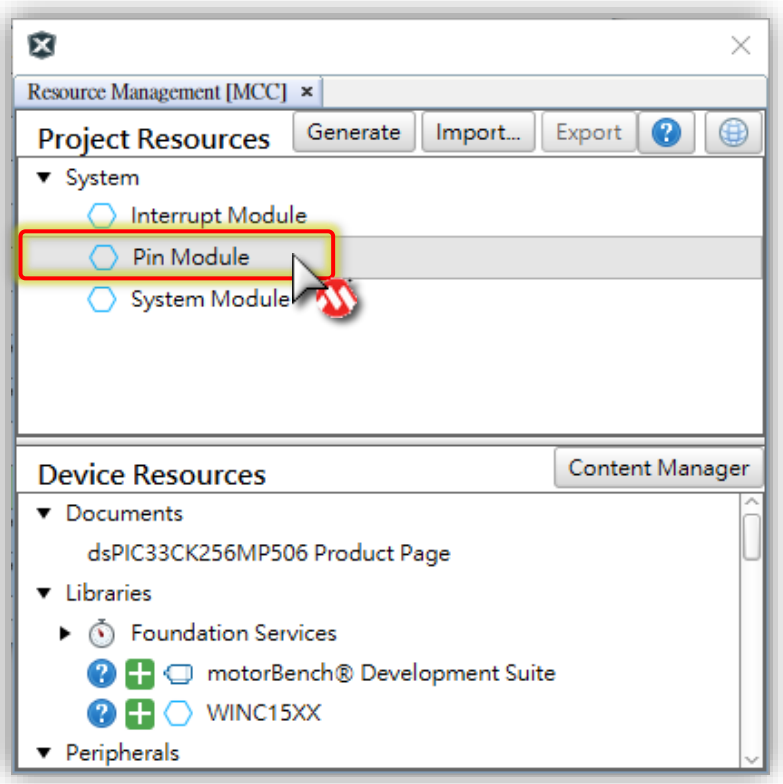


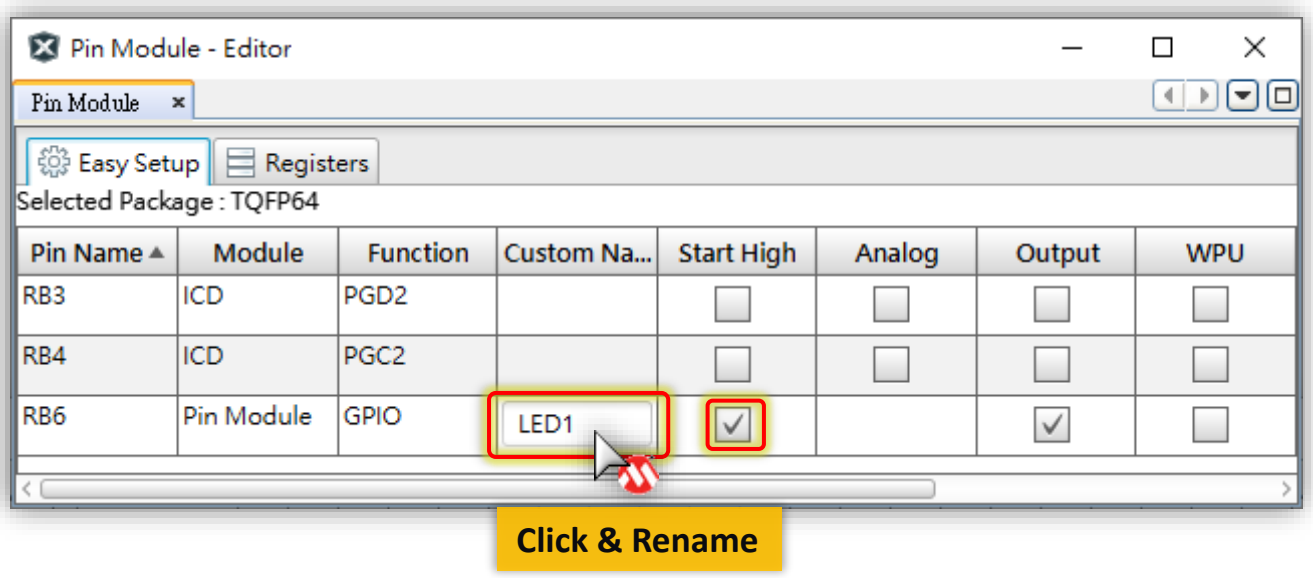
MICROCHIP

Lab1 GPIO Output

Step 3

- **Set Alias for RB6.**
  - **Pin Module : RB6 , Custom Name > LED1 , Start High , Check Output.**

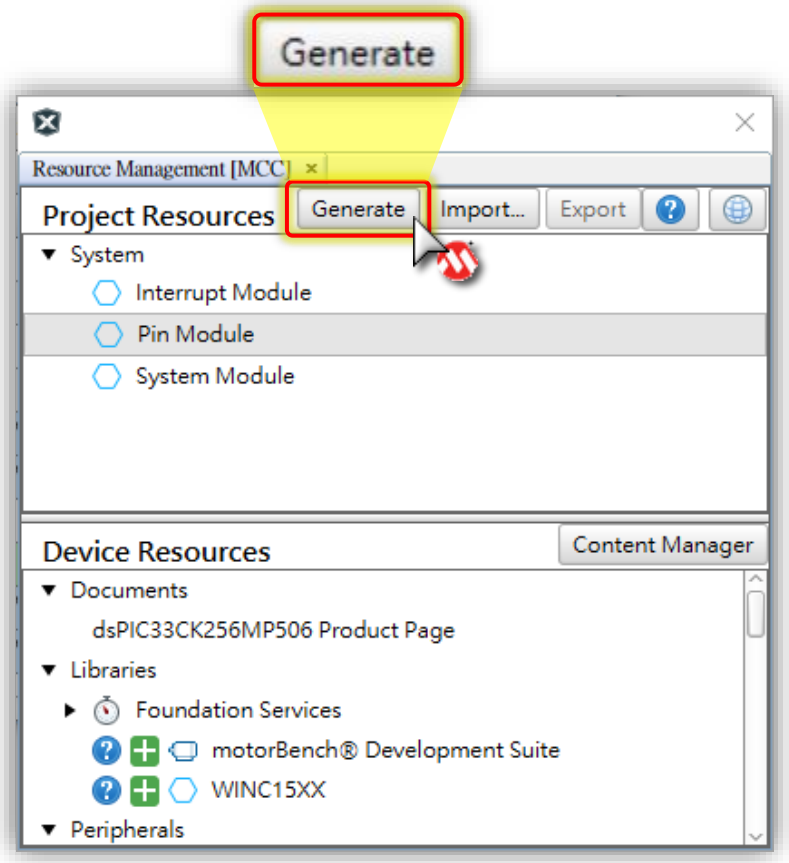


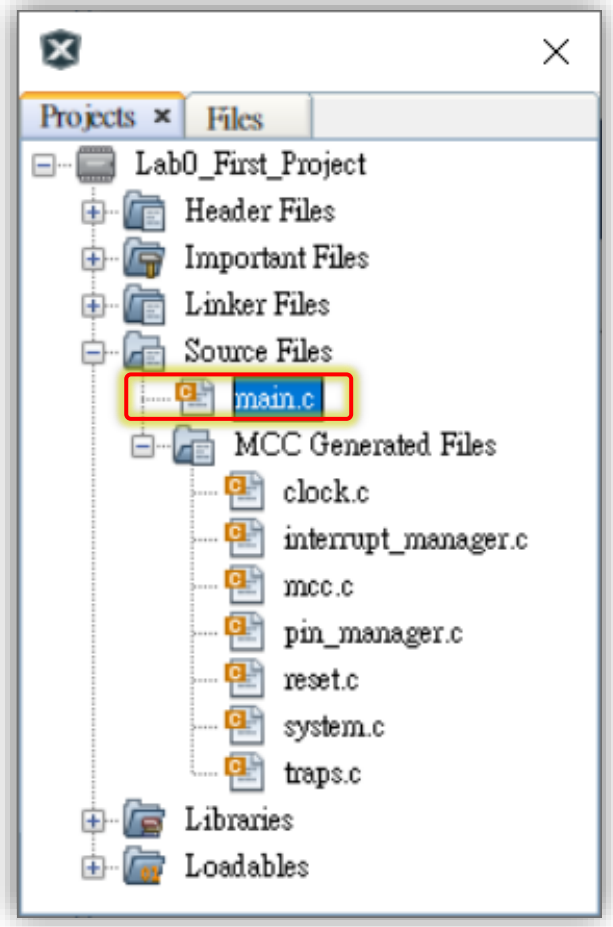


Lab1 GPIO Output

Step 4

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

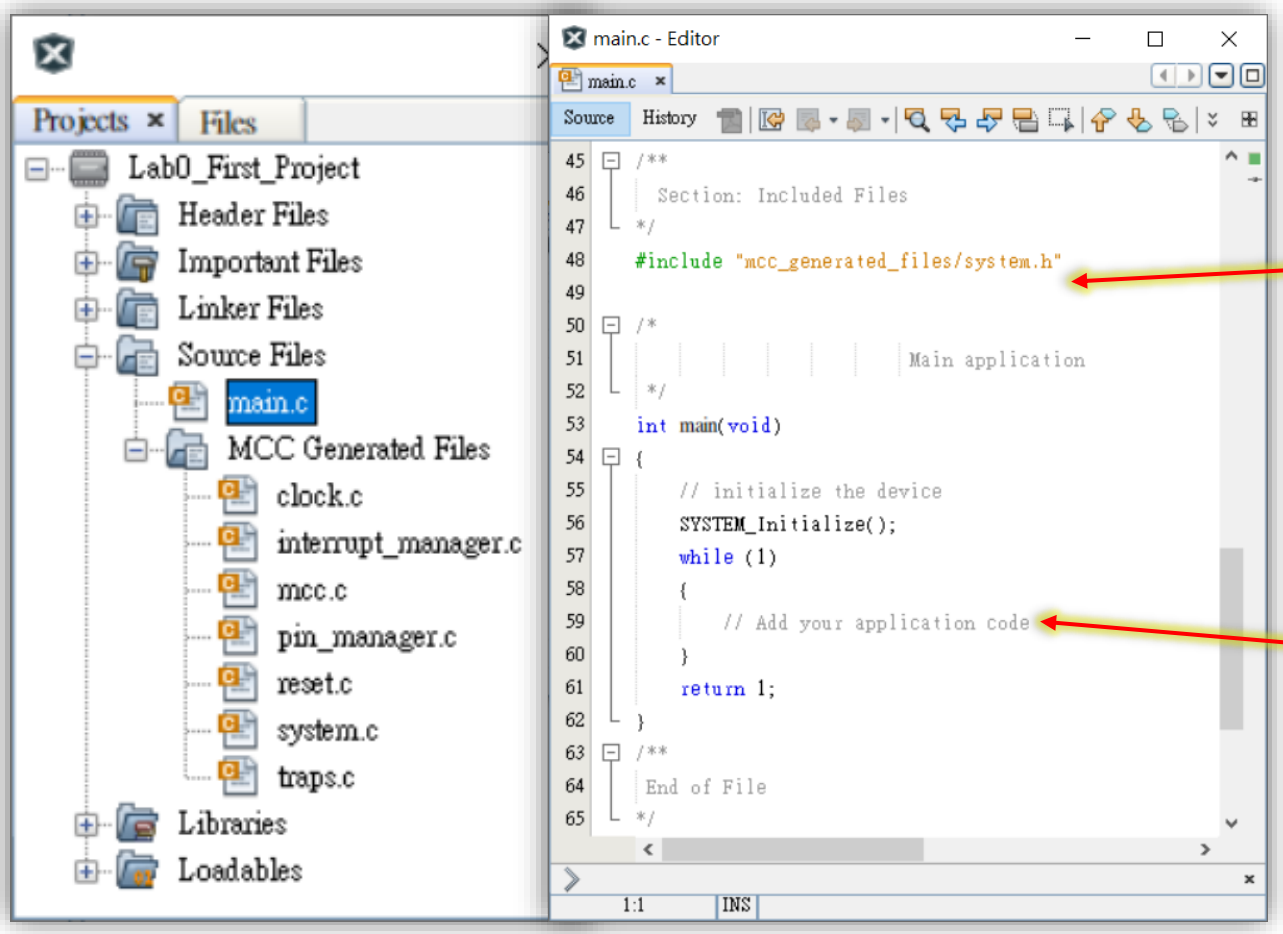




Lab1 GPIO Output

Step 5

- **Try to add code to main function double click "main.c" to view & and add below code segment in it.**

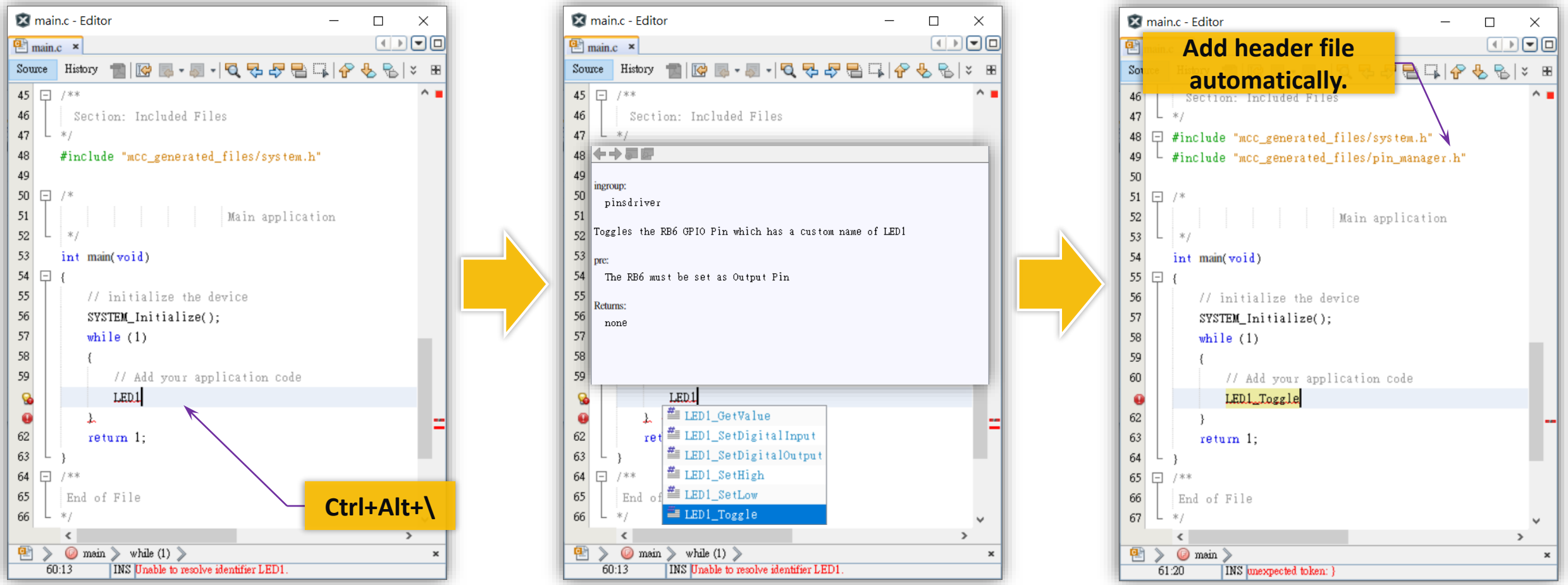


Add below code to main()

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/pin_manager.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 < 80000; i++);
    }
    return 1;
}
```

MPLAB X IDE Hints

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



Lab1 GPIO Output

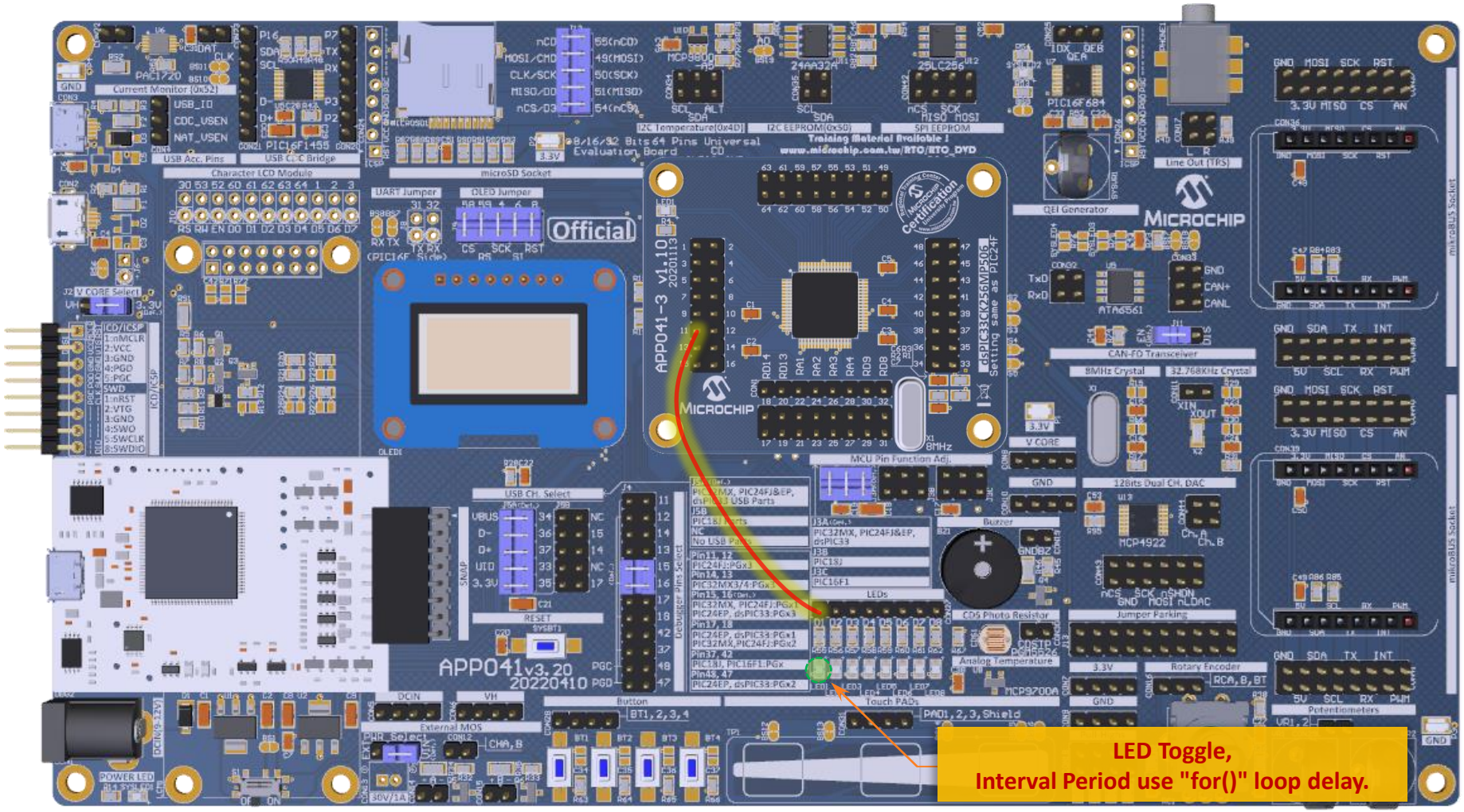
Code Example

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/pin_manager.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 < 80000; i++);
    }
    return 1;
}
```

MICROCHIP

Lab1 GPIO Output

Result



Lab2 Multi-GPIO Output

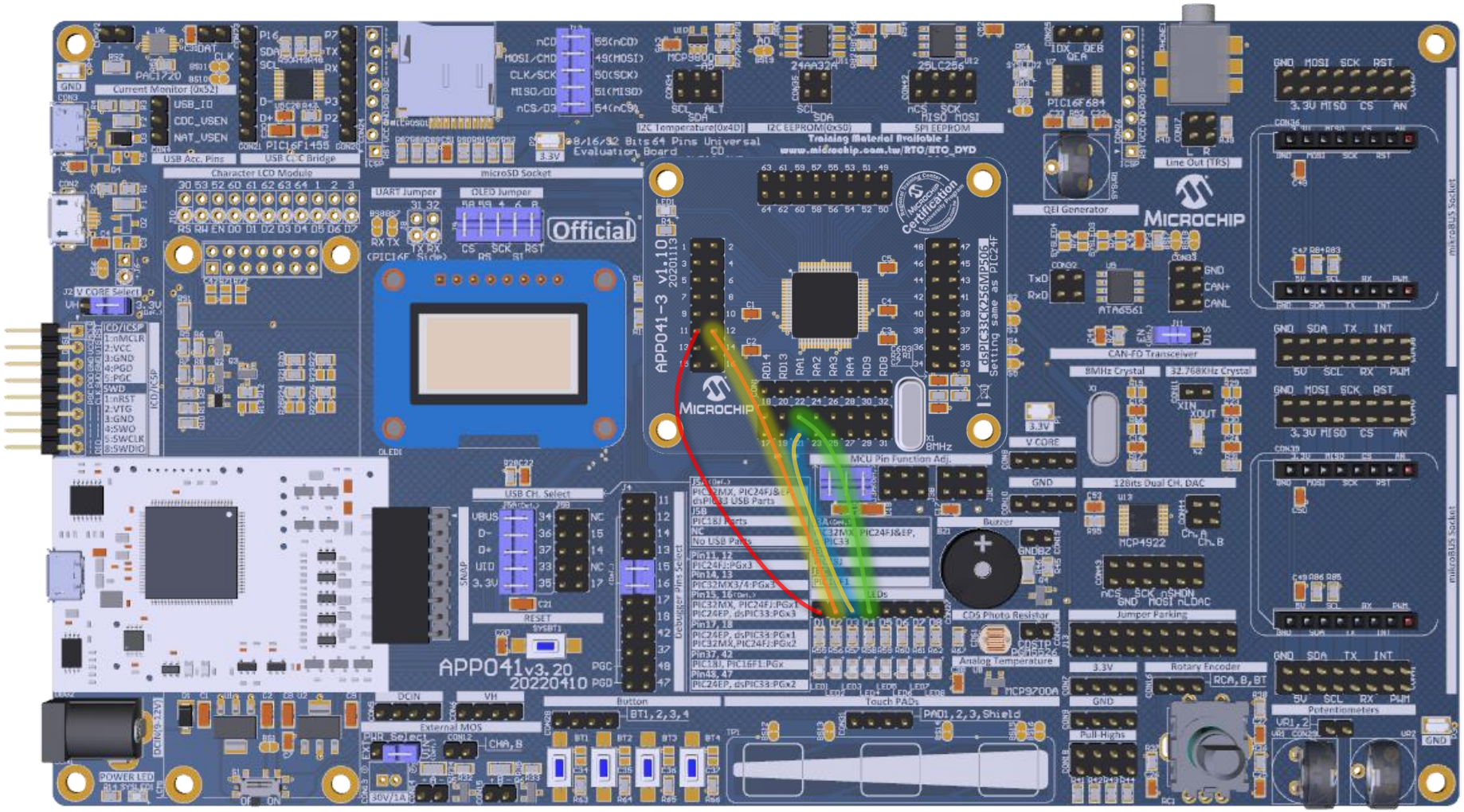
- Base on current project or Open .\Exams\Lab2_xxx.X to do exercises.
- Try to set more GPIO pins to digital output mode and toggle levels (period around 500ms ~ 1S), individually.
- Please connect
  - RB5 (Pin12) to LED2 (LED2)
  - RD12 (Pin21) to LED3 (LED3) , Start High
  - RC1 (Pin22) to LED4 (LED4)

Let’s go!

MICROCHIP

Lab2 Multi-GPIO Output

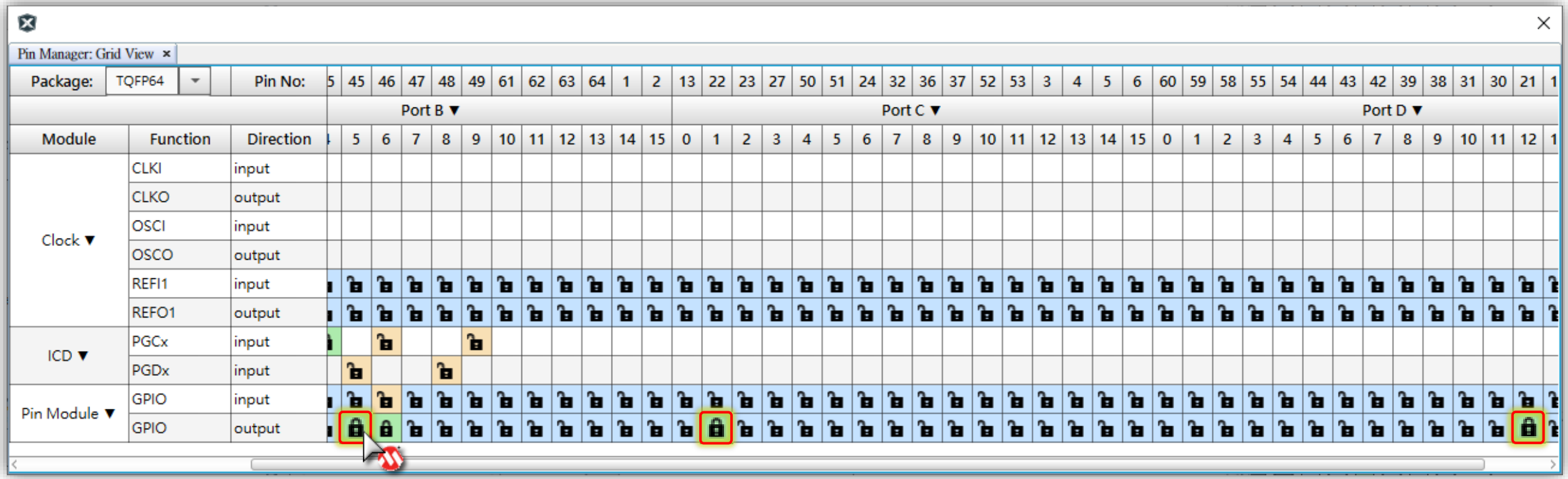
Connection



Lab2 Multi-GPIO Output

Step 1

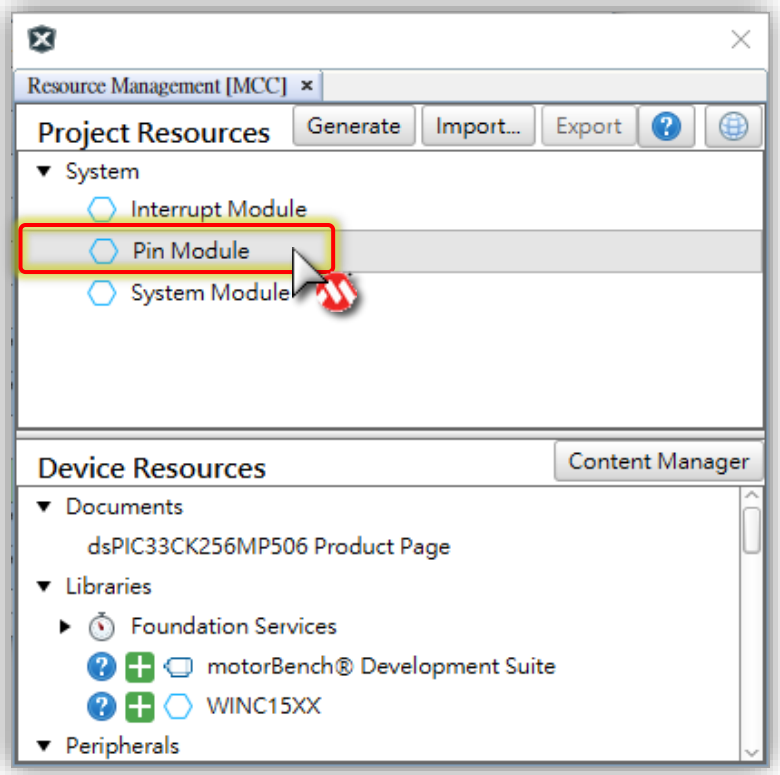
- **Set PORTB : Pin5 (RB5), PORTC : Pin1 (RC1), PORTD : Pin12 (RD12), to digital output mode.**

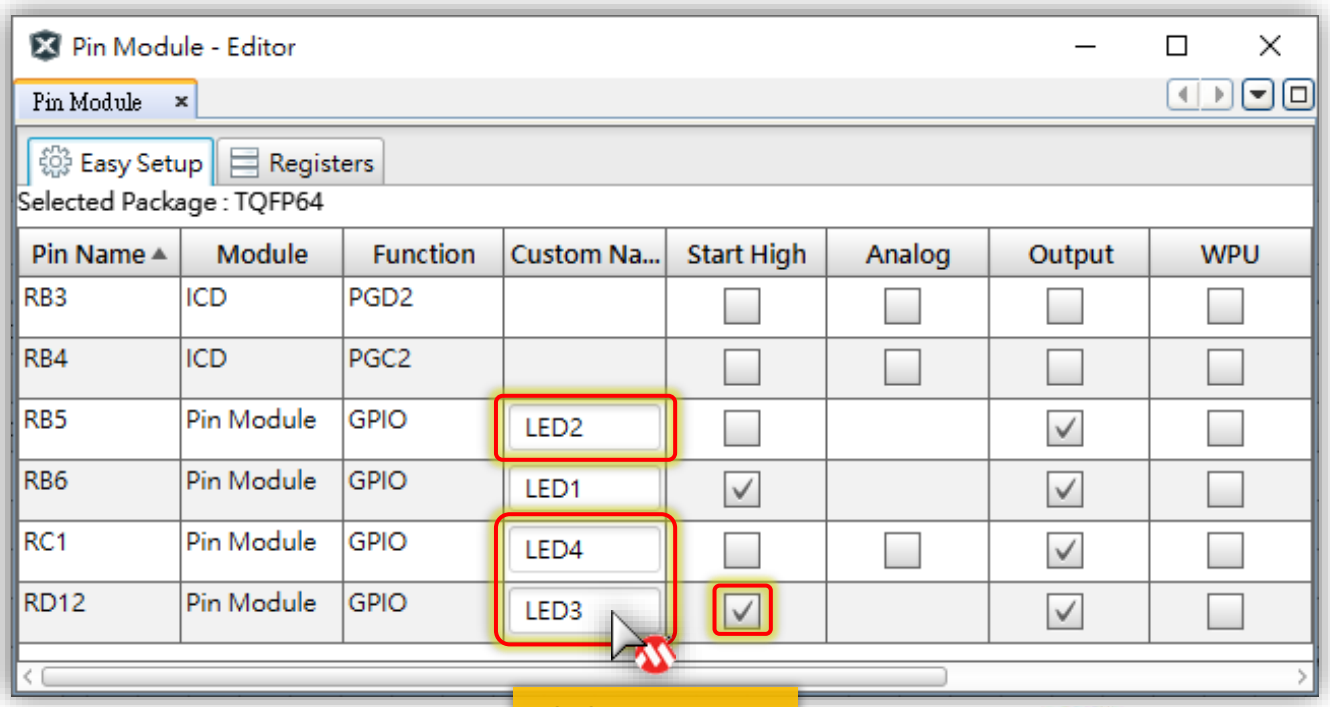


Lab2 Multi-GPIO Output

Step 2

- **Set Alias for RB5 , RD12 and RC1.**
  - **Pin Module : RB5 , Custom Name > LED2 , Check Output.**
  - **Pin Module : RD12 , Custom Name > LED3 , Start High, Check Output.**
  - **Pin Module : RC1 , Custom Name > LED4 , Check Output.**

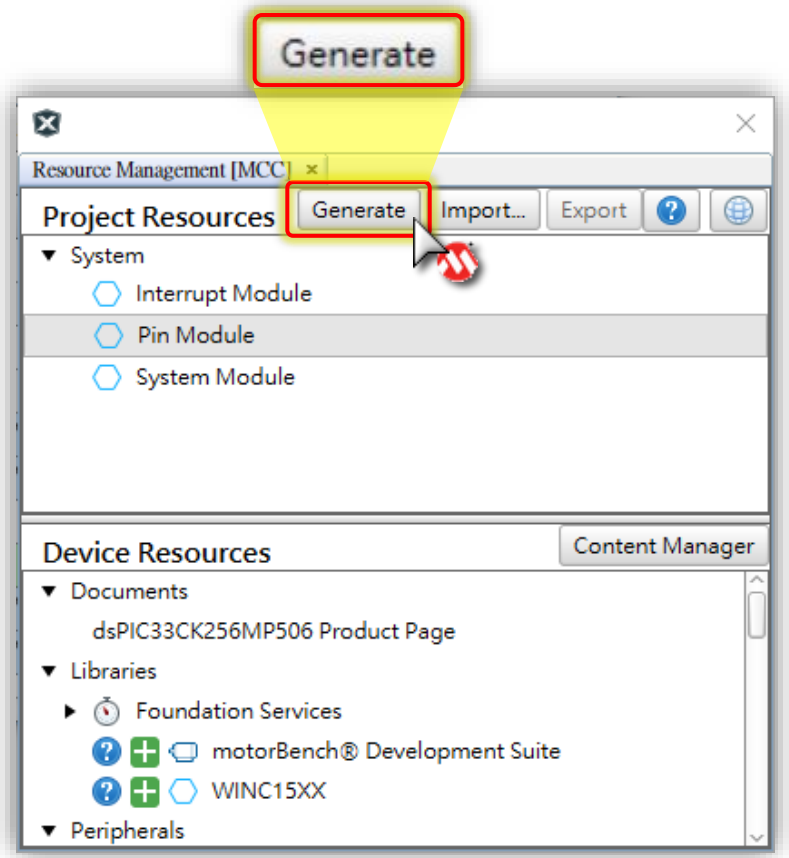


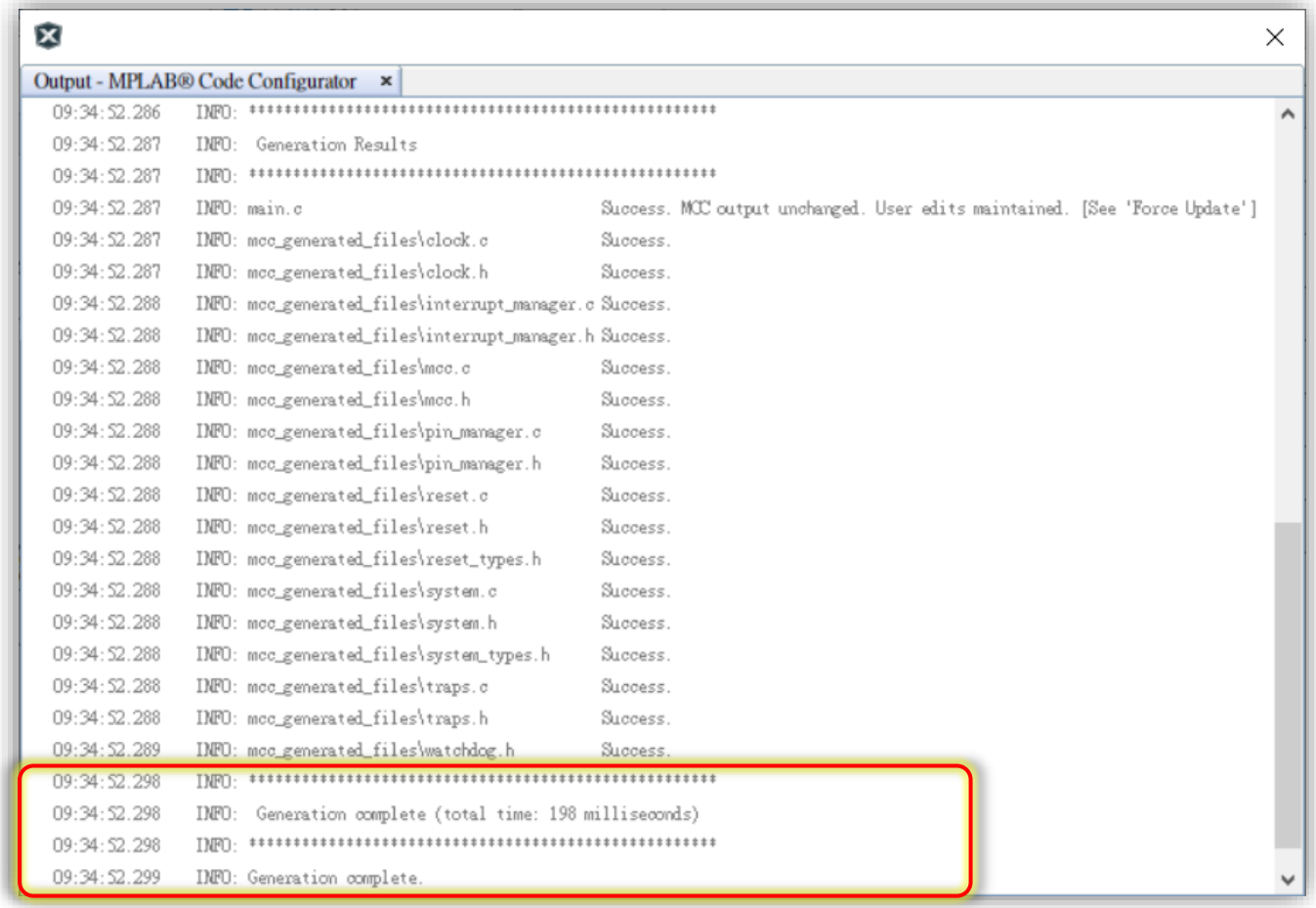


Lab2 Multi-GPIO Output

Step 3

- Click "Generate" Button to generate your code.





Lab2 Multi-GPIO Output

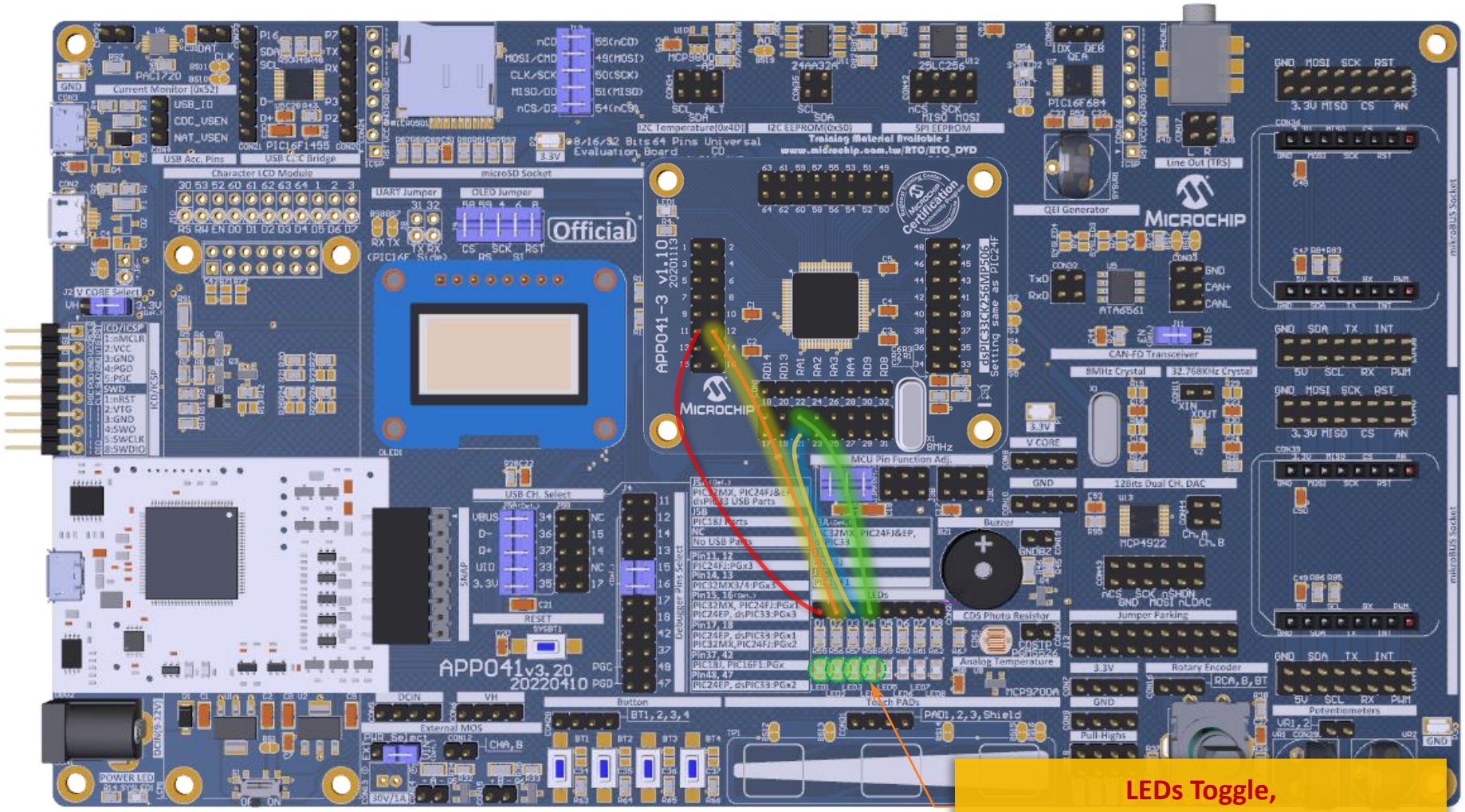
Code Example

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/pin_manager.h"

uint32_t  i = 0;

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

    while (1)
    {
        // Add your application code
        LED1_Toggle();
        LED2_Toggle();
        LED3_Toggle();
        LED4_Toggle();

        for (i = 0; i < 80000; i++);
    }
    return 1;
}
```

MICROCHIP

Lab2 Multi-GPIO Output

Result



Lab3 GPIO Input

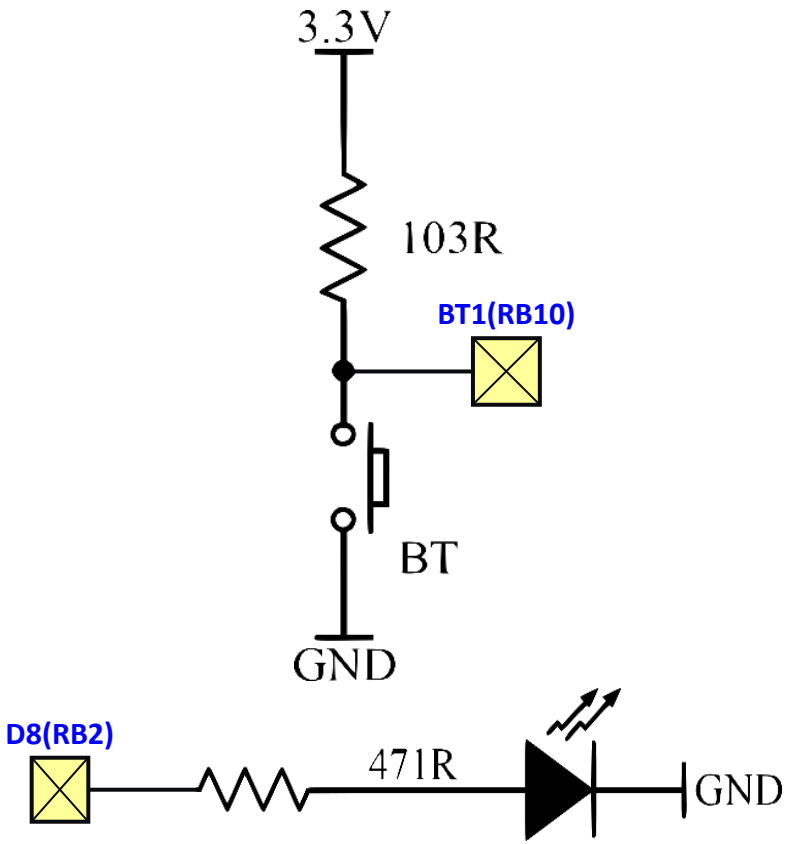
- **Base on current project or Open .\Exams\Lab3_xxx.X to do exercises.**
- **Try to set RB10 to digital input as get button status.**
- **Try use button to control LED light or dark.**
  - BT1 Pressed > D8 Light.
  - BT1 Released > D8 Dark.
- **Please connect**
  - **RB10 (Pin61) to Button (BT1),**
  - **RB2 (Pin33) to LED (LED8).**

Let’s go!

MICROCHIP

Lab3 GPIO Input

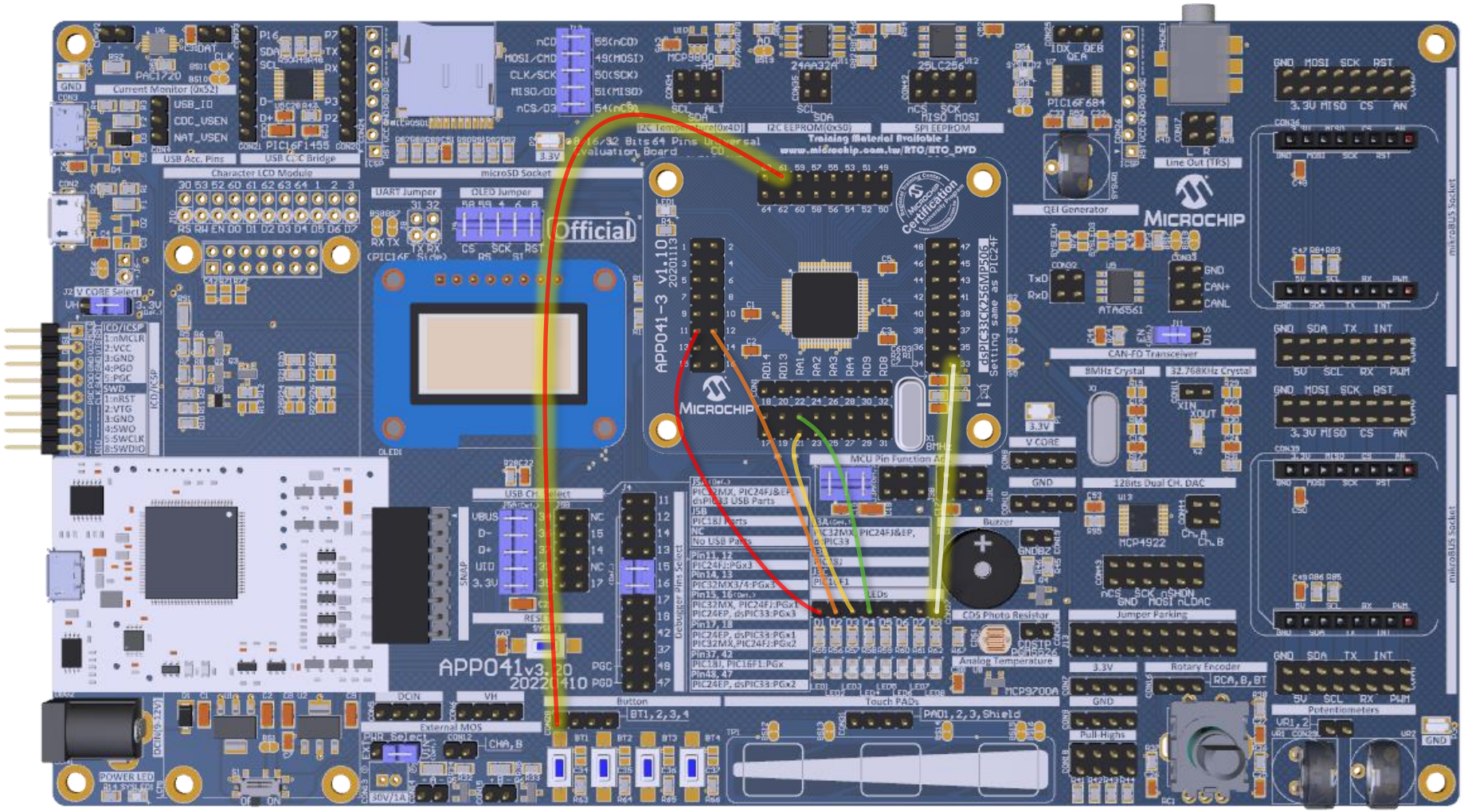
- *CustomName*_GetValue(); function is used for get digital input level from outside. The simple example code shows below:




```
if(BT1_GetValue())
{
   LED8_SetLow();
}
else
{
   LED8_SetHigh();
}
```

Lab3 GPIO Input

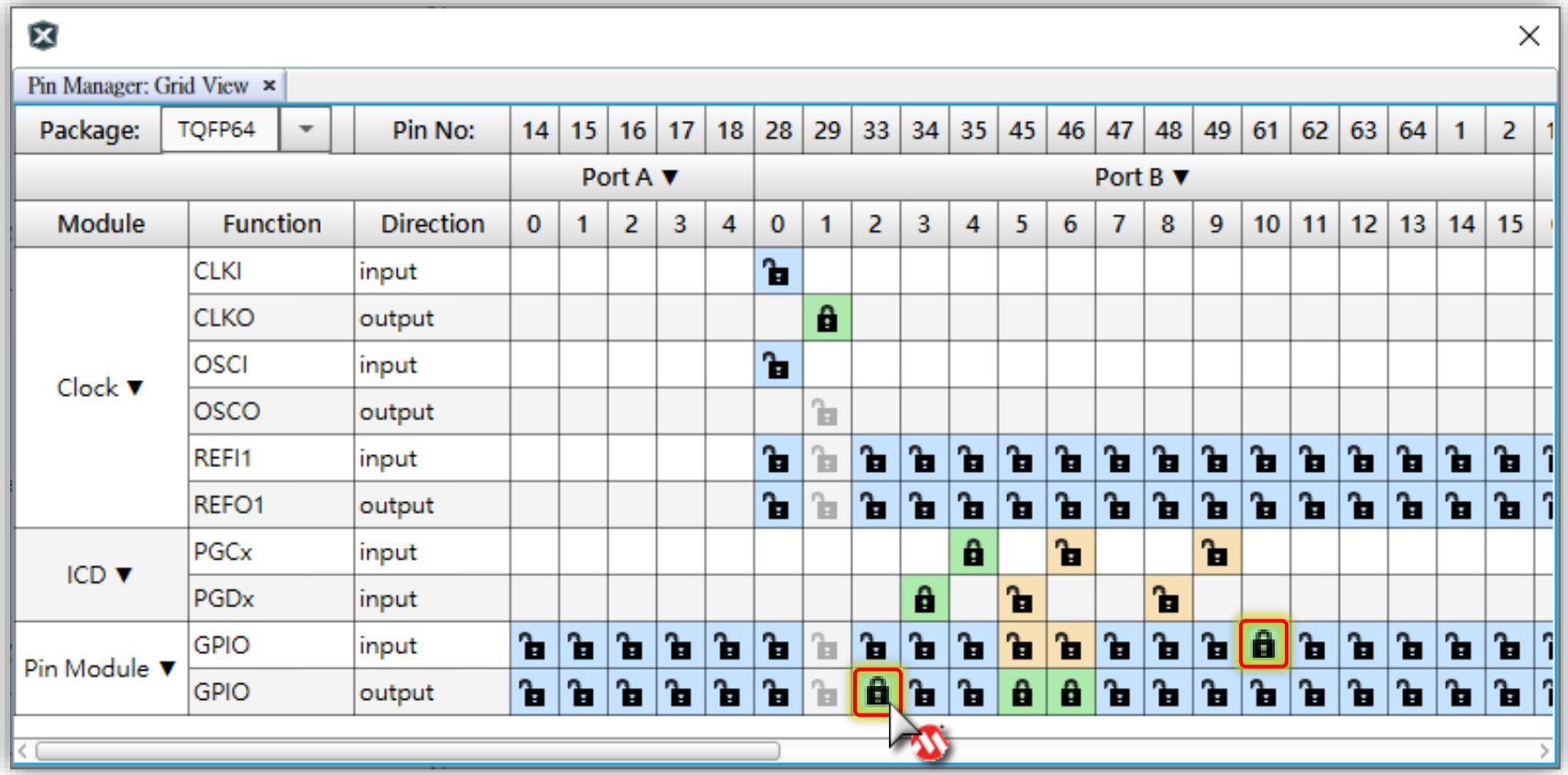
Connection



Lab3 GPIO Input

Step 1

- Set PORTB : Pin2 (RB2) to digital output mode.
- Set PORTB : Pin10 (RB10) to digital input mode.

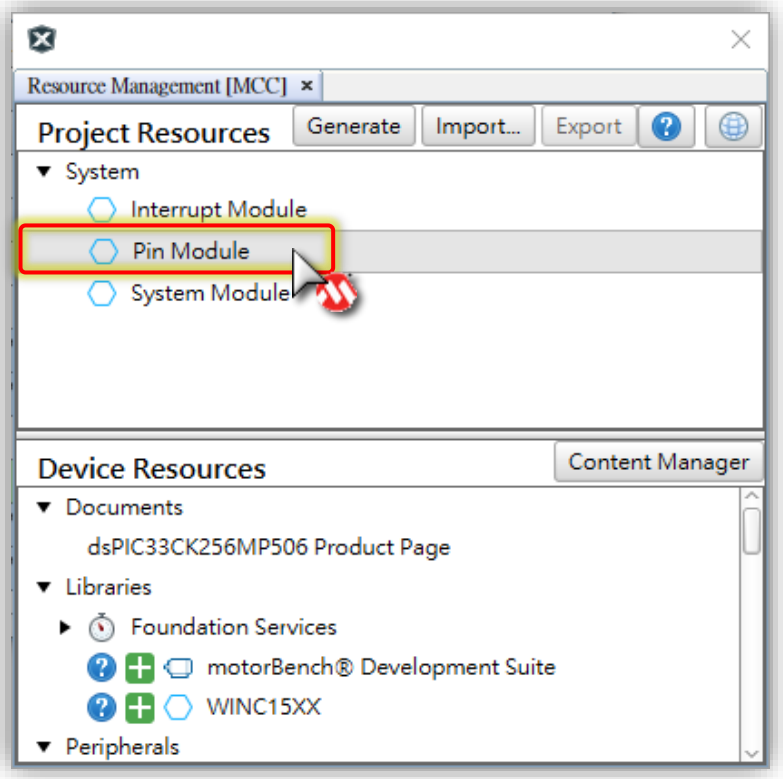


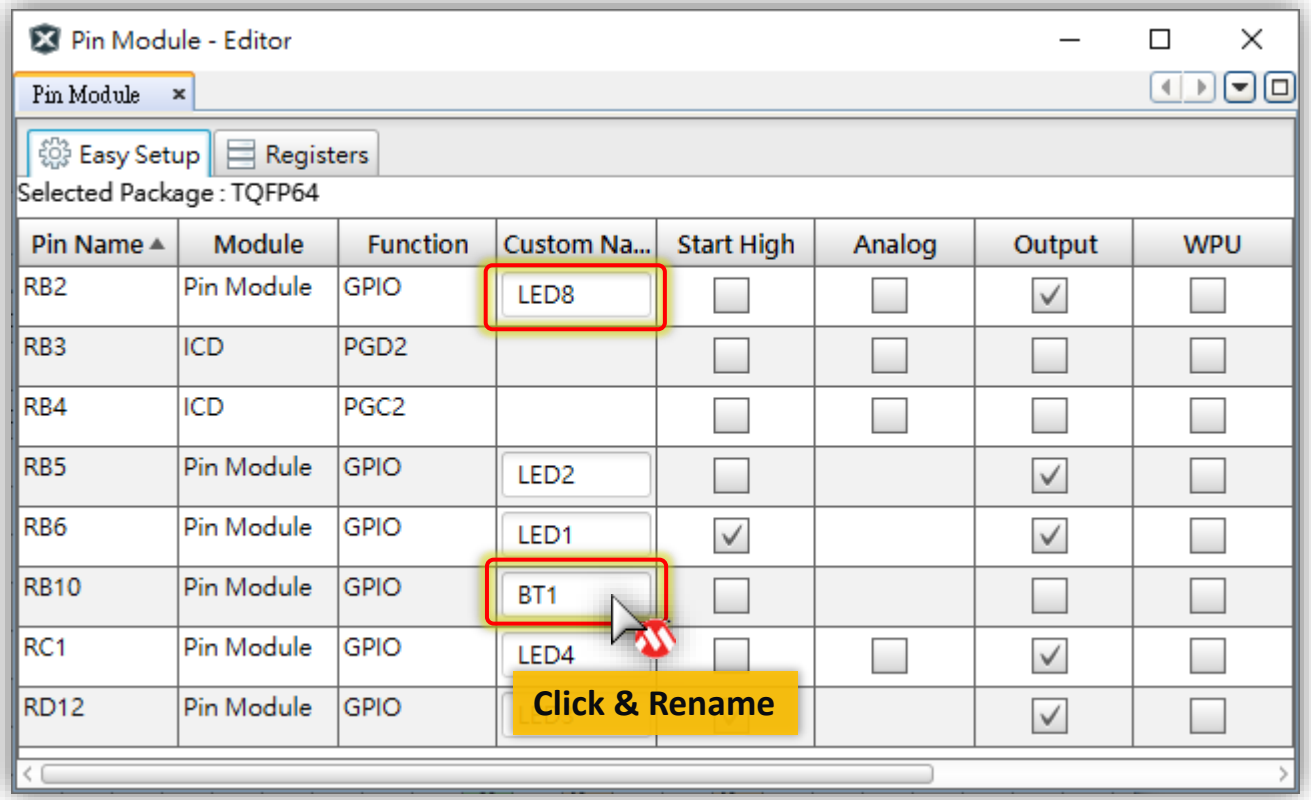
MICROCHIP

Lab3 GPIO Input

Step 2

- **Set Alias for RB2 and RB10.**
  - **Pin Module : RB2 , Custom Name > LED8 , Check Output.**
  - **Pin Module : RB10 , Custom Name > BT1.**

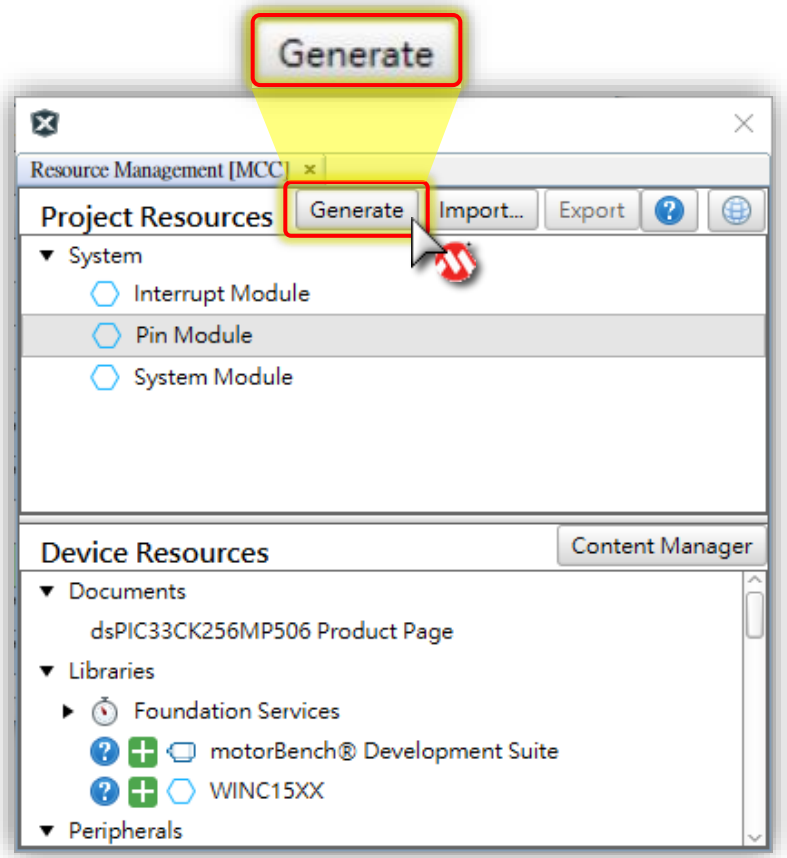


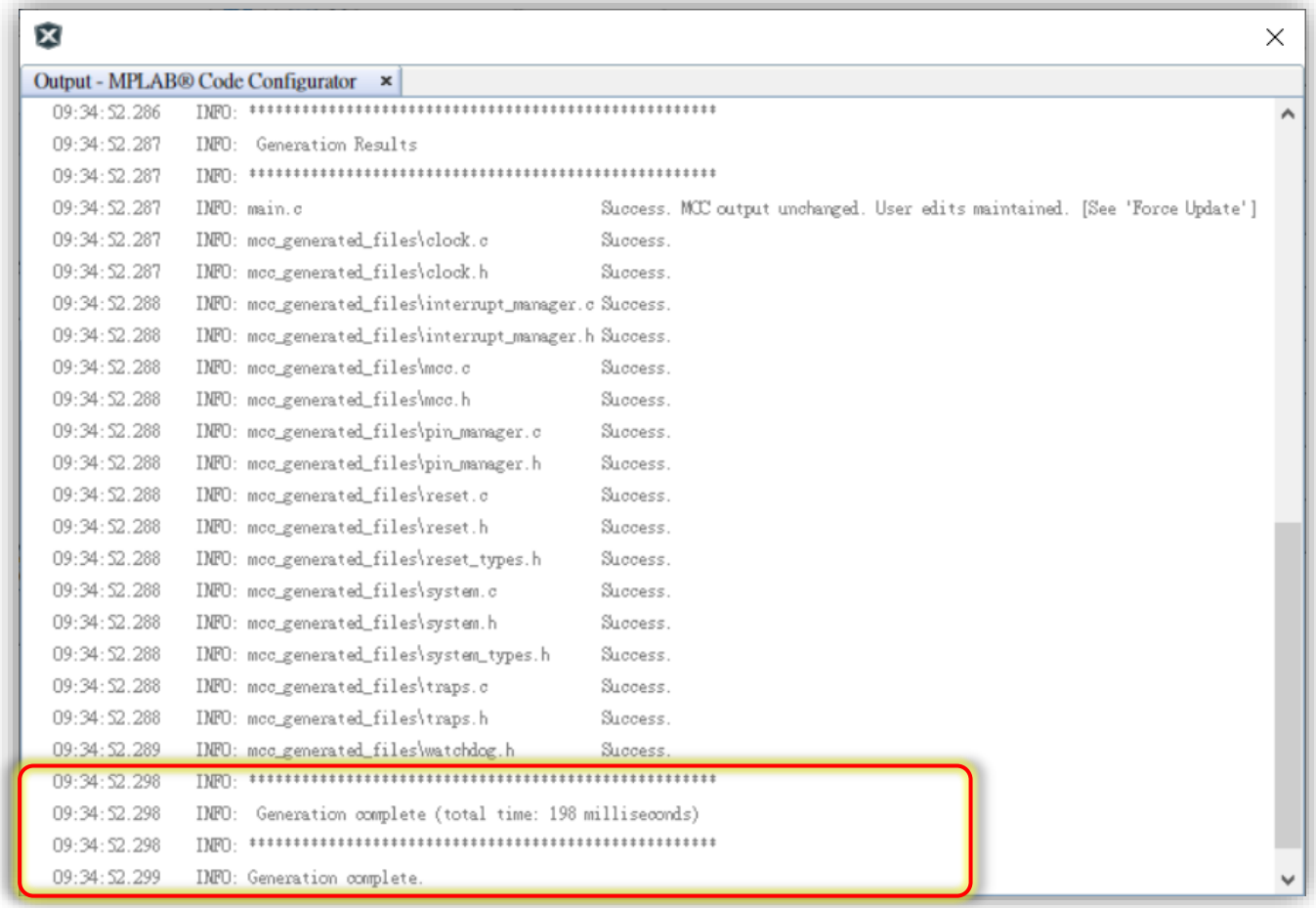


Lab3 GPIO Input

Step 3

- Click "Generate" Button to generate your code.





Lab3 GPIO Input

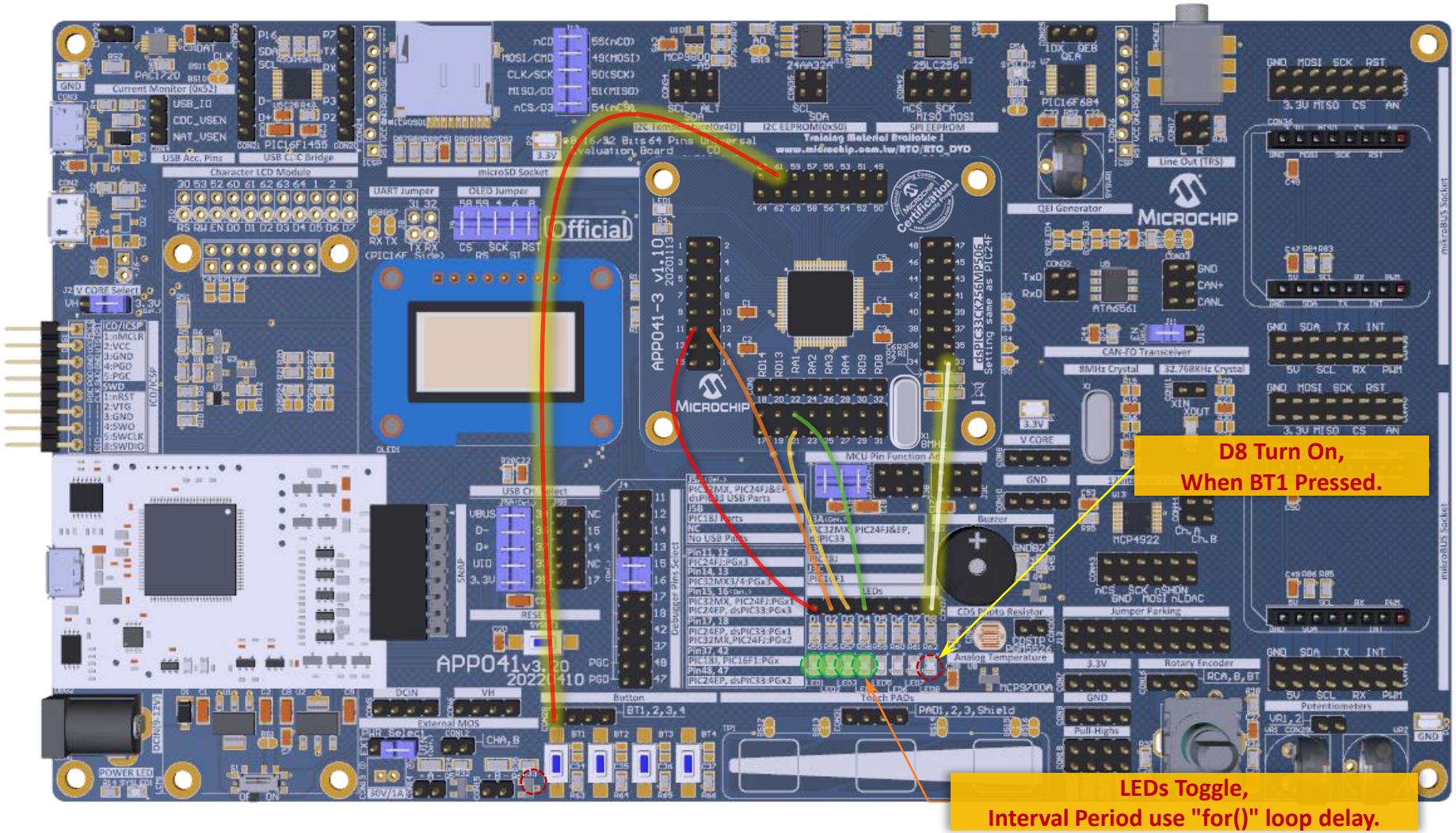
Code Example

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/pin_manager.h"

uint32_t  i = 0;

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

    while (1)
    {
        // Add your application code
        if( i++ > 80000 )
        {
            LED1_Toggle();
            LED2_Toggle();
            LED3_Toggle();
            LED4_Toggle();
            i = 0;
        }

        if(BT1_GetValue())
            LED8_SetLow();
        else
            LED8_SetHigh();
    }
    return 1;
}
```

MICROCHIP

Lab3 GPIO Input

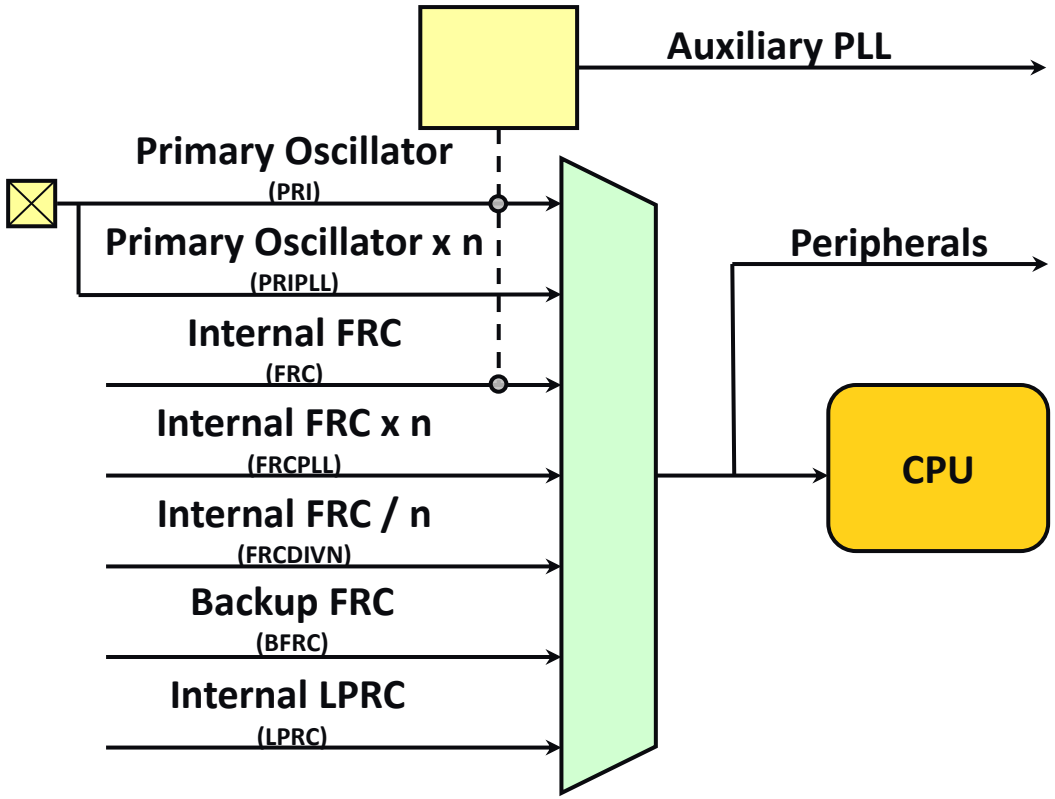
Result



Oscillator Architecture

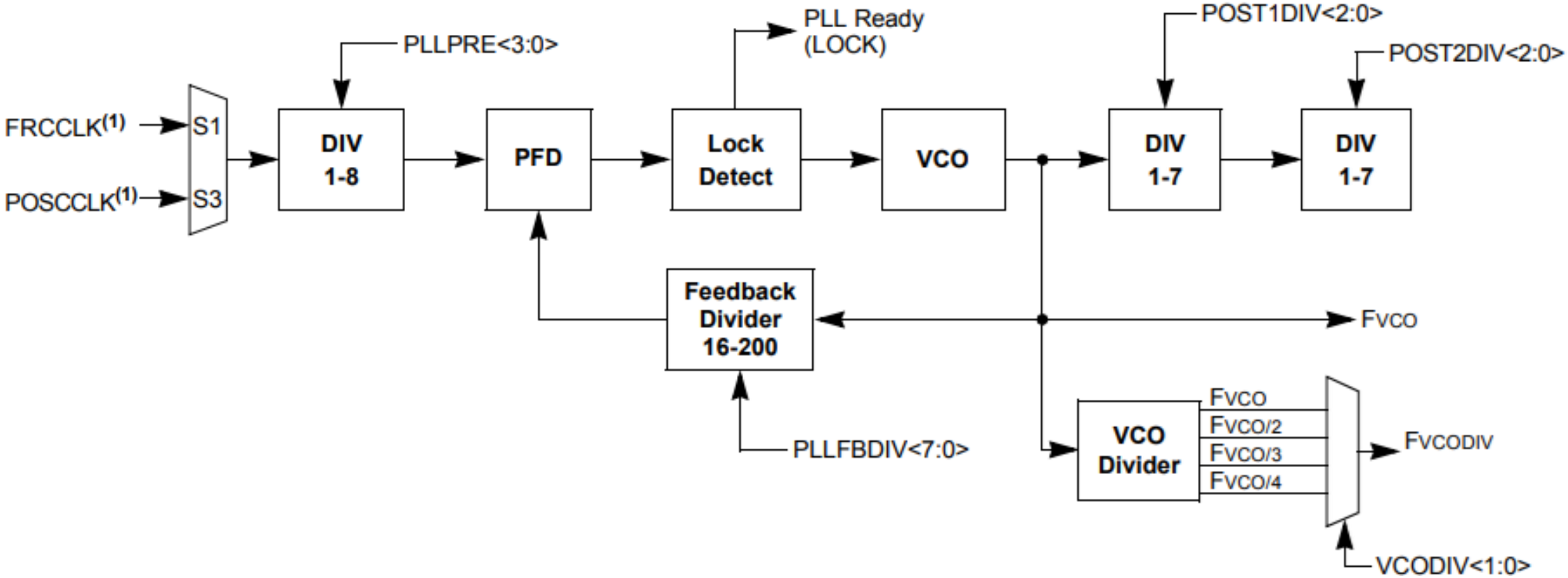
dsPIC33CK Series Clocks

- Microchip dsPIC33CK Series MCU available clock sources are internal Fast RC, Low Power RC and External Crystal or Oscillator, etc..
- The clocks can be divided or multiply by PLL.
- Auxiliary PLL (APLL) Clock Generator to Boost Operating Frequency for Peripherals
- Software-controllable switching between various clock sources.



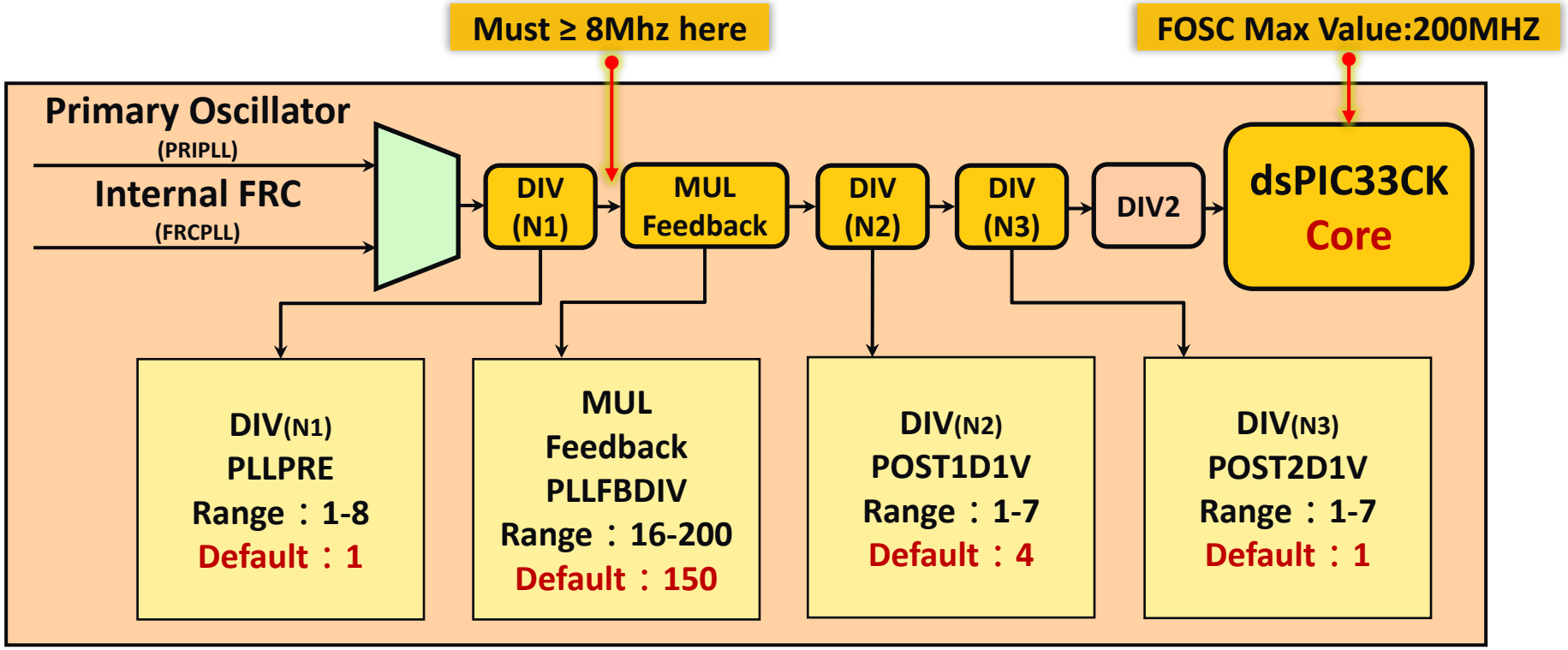
Phase Locked Loop

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



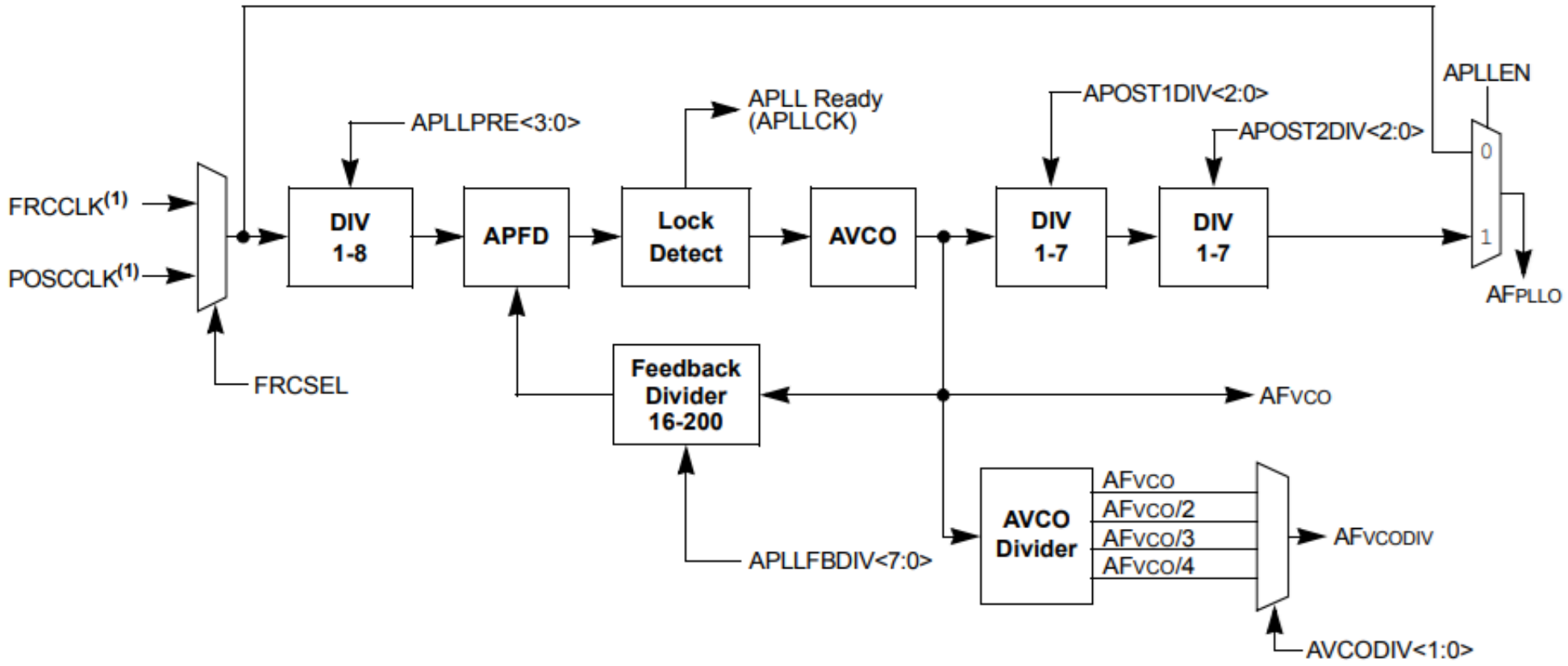
Phase Locked Loop

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



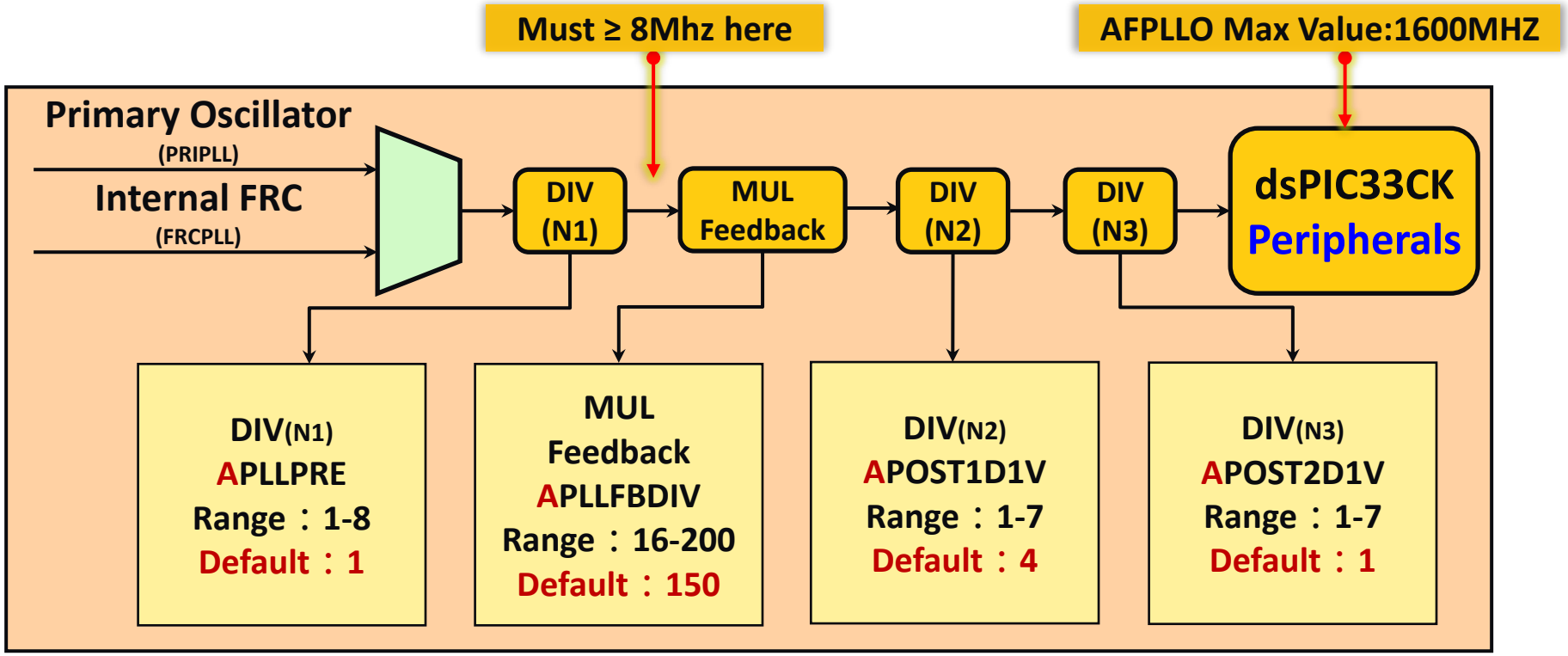
Phase Locked Loop

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

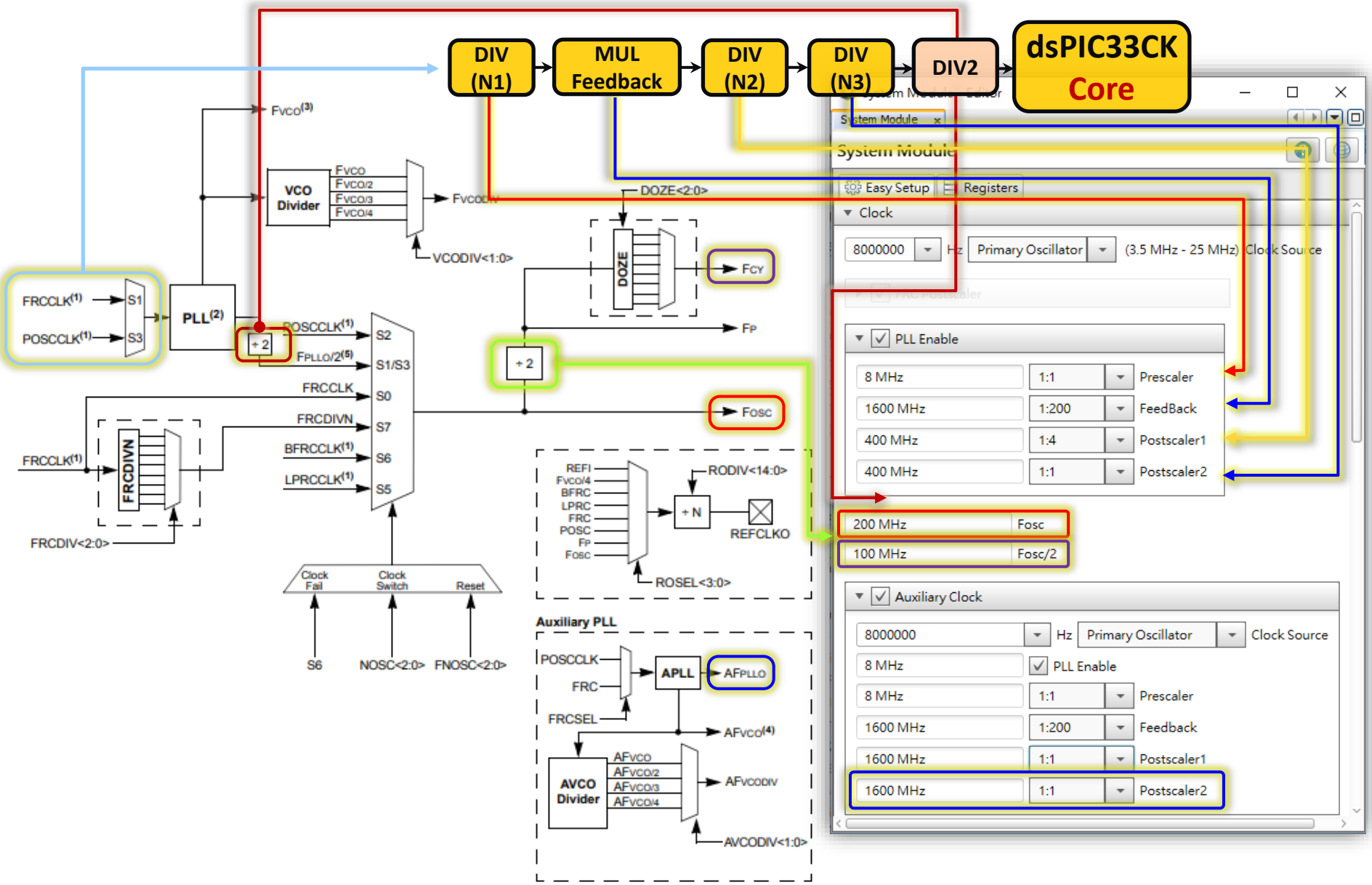


Auxiliary Phase Locked Loop

- Clock input must 8MHz after APLLPRE, which can use Primary Oscillators (Crystal or External Clock) or Internal Fast RC as clock source.



Oscillator Setting of MCC



Lab4 Clock PRIPLL

- **Base on current project or Open .\Exams\Lab4_xxx.X to do exercises.**
- **Try to change system clock from internal Fast RC to External Crystal (8MHz) and enable PLL.**
- **Try to provide maximum frequency (16MHz) to MCU.**

Let’s go!

MICROCHIP

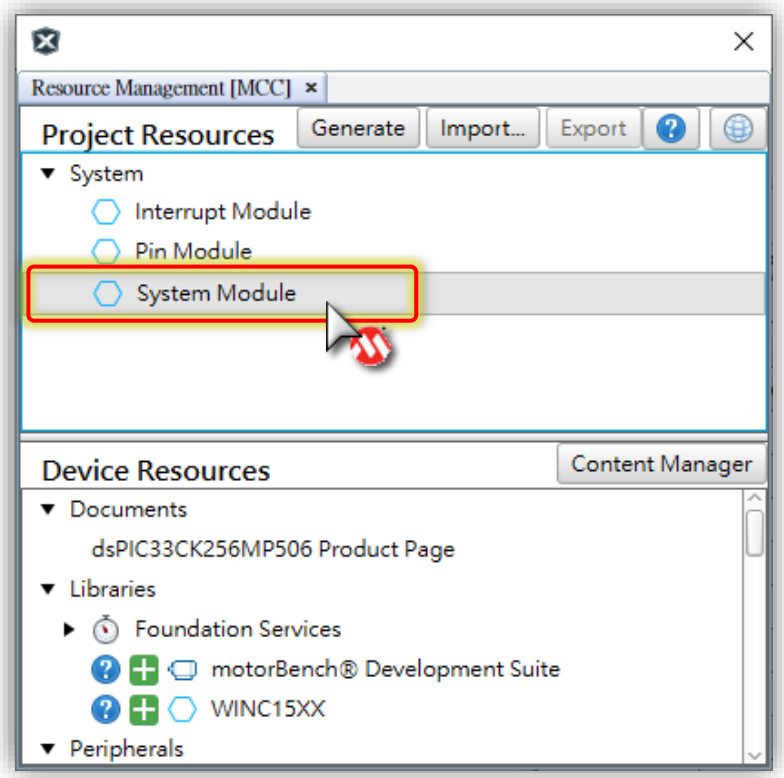
Lab4 Clock PRIPLL

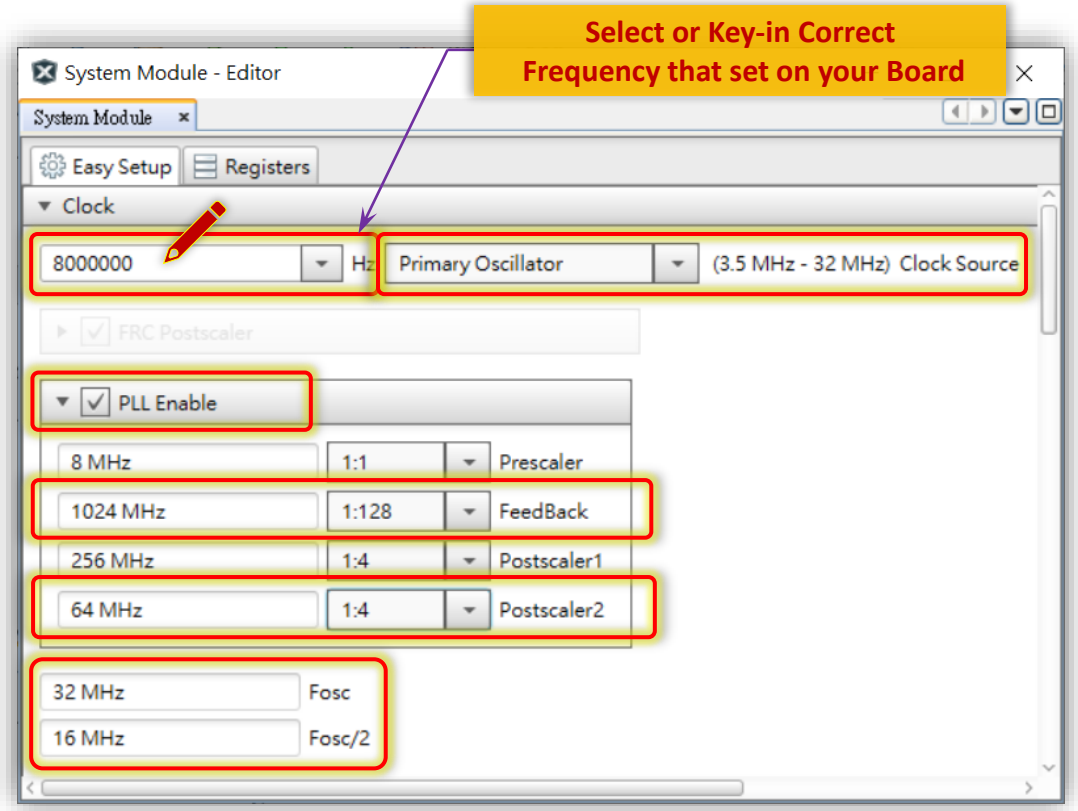
Connection

Lab4 Clock PRIPLL

Step 1

- **Set System Module**
  - **Set Primary Oscillator 8MHz , enable PLL , FeedBack > 1:128 , Postscaler2 > 1:4.**

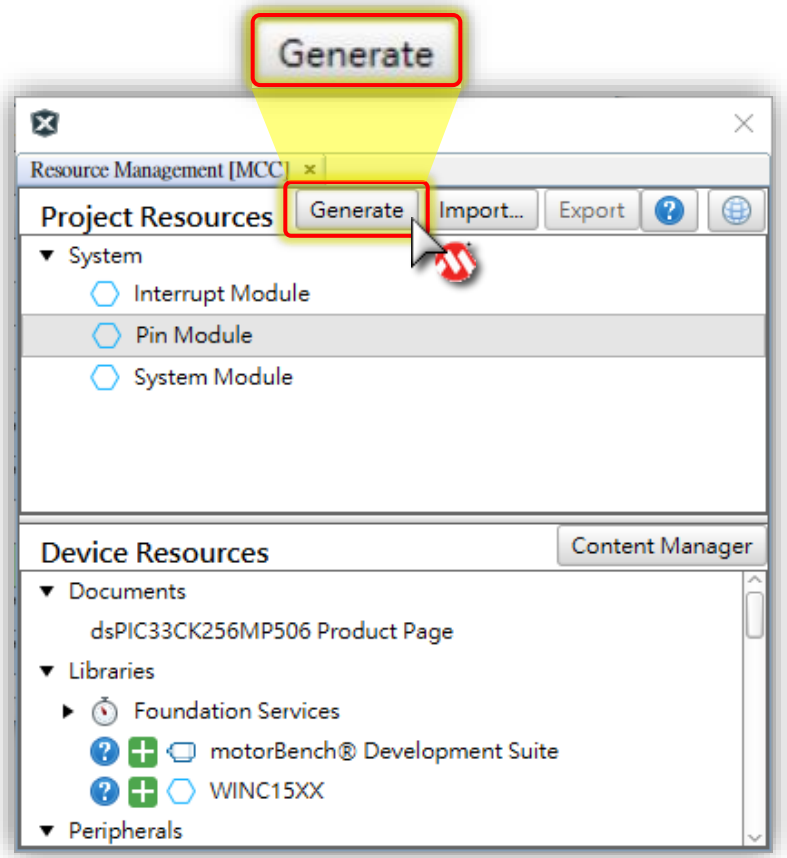


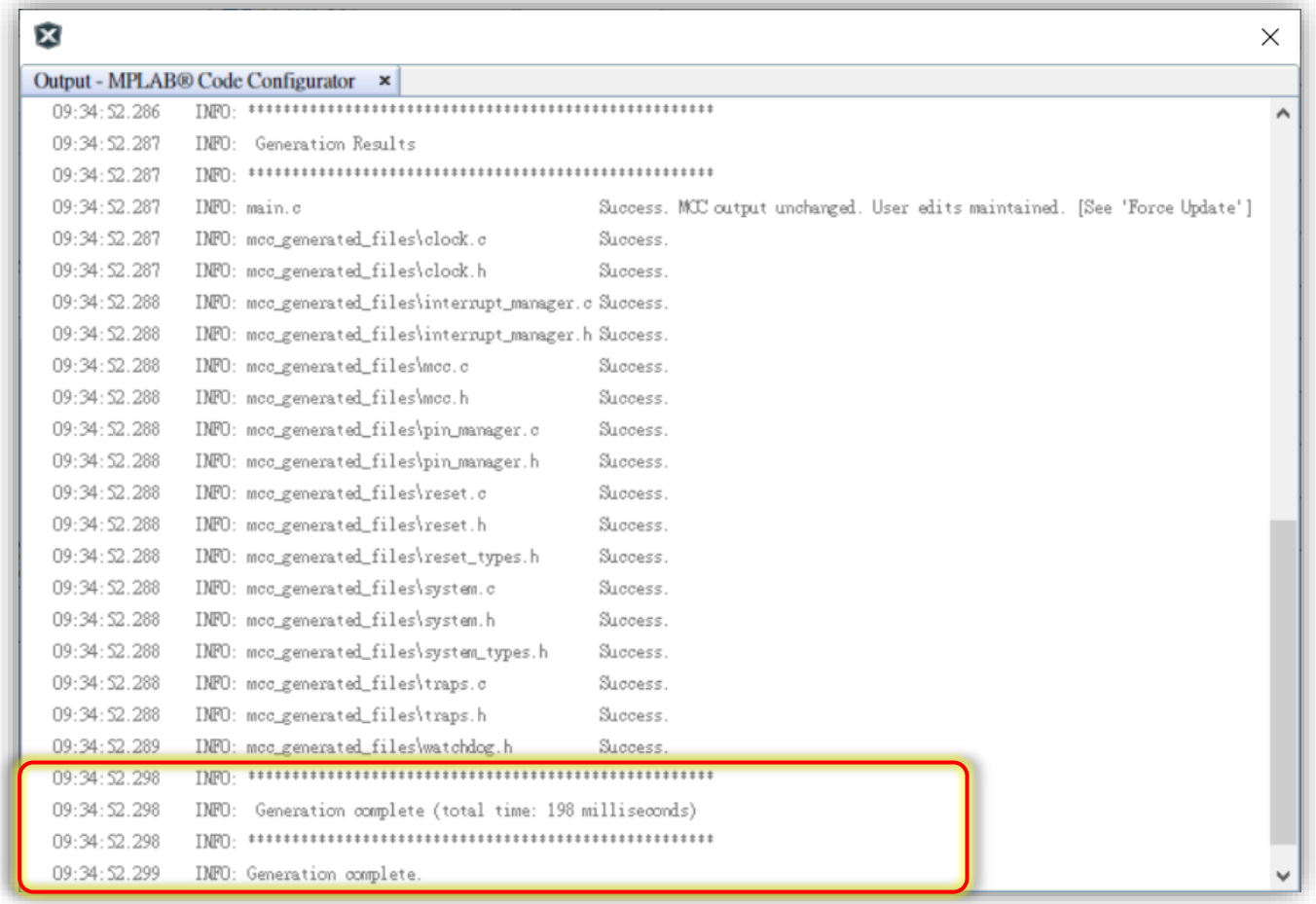


Lab4 Clock PRIPLL

Step 2

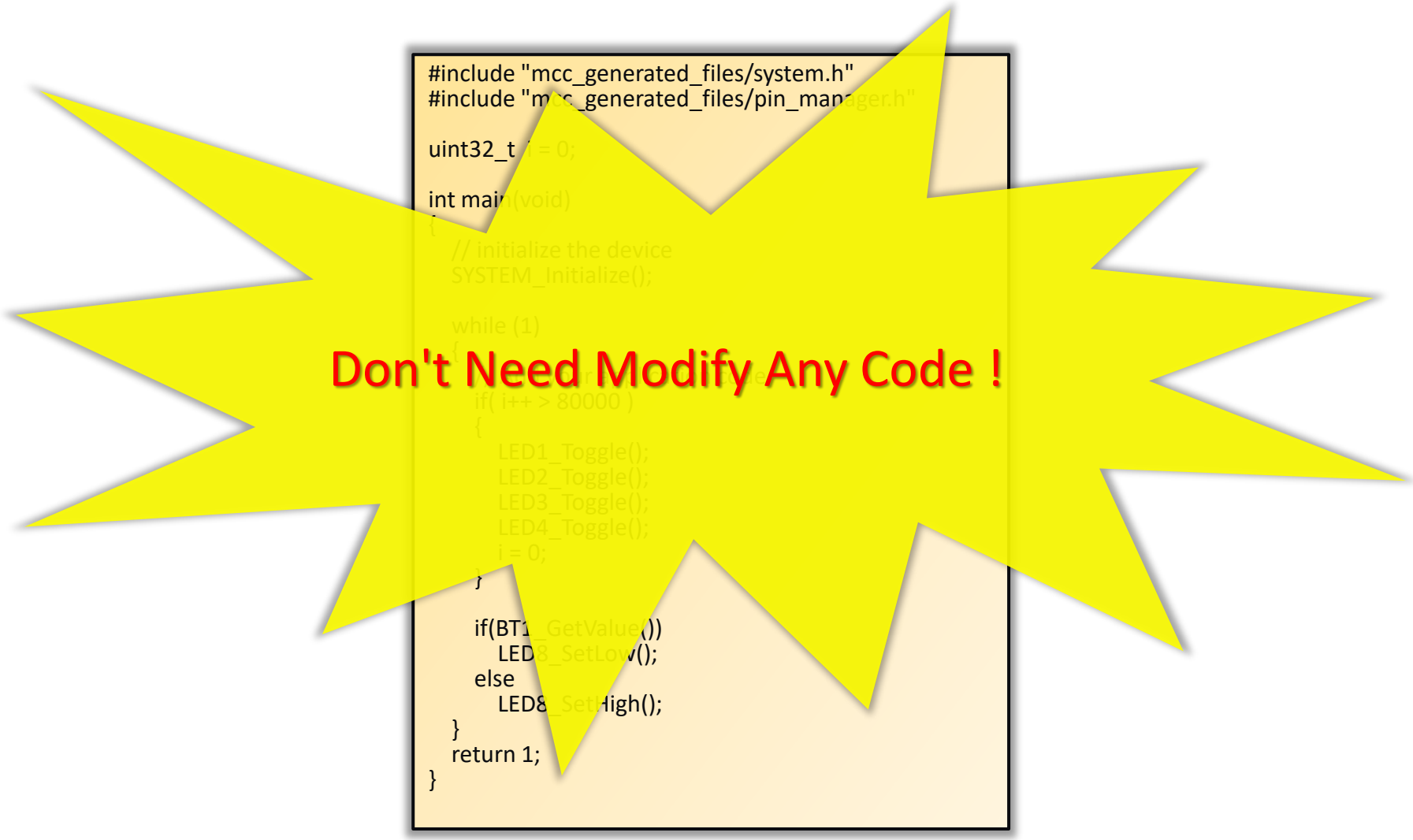
- Click "Generate" Button to generate your code.





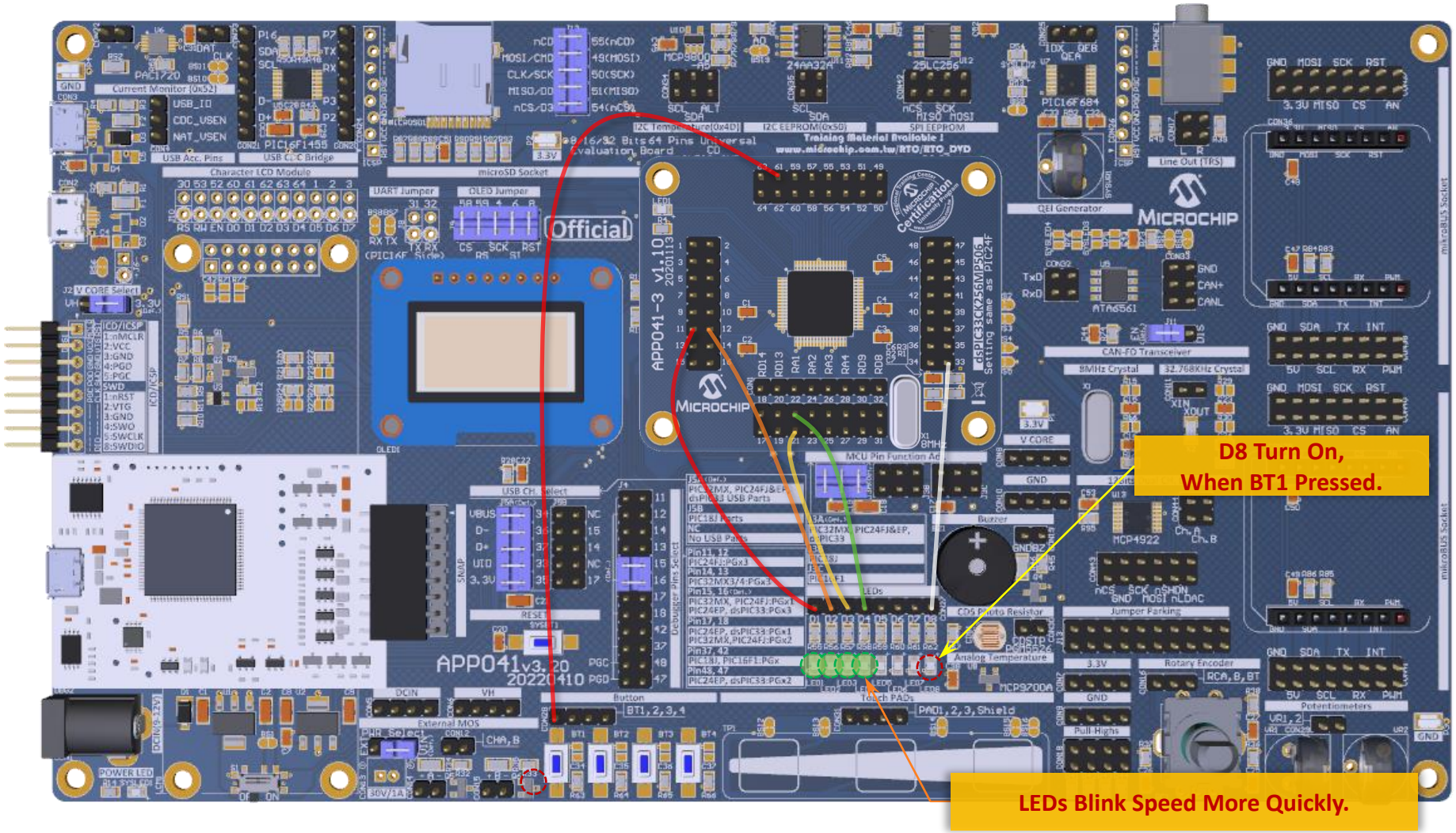
Lab4 Clock PRIPLL

Code Example



Lab4 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.

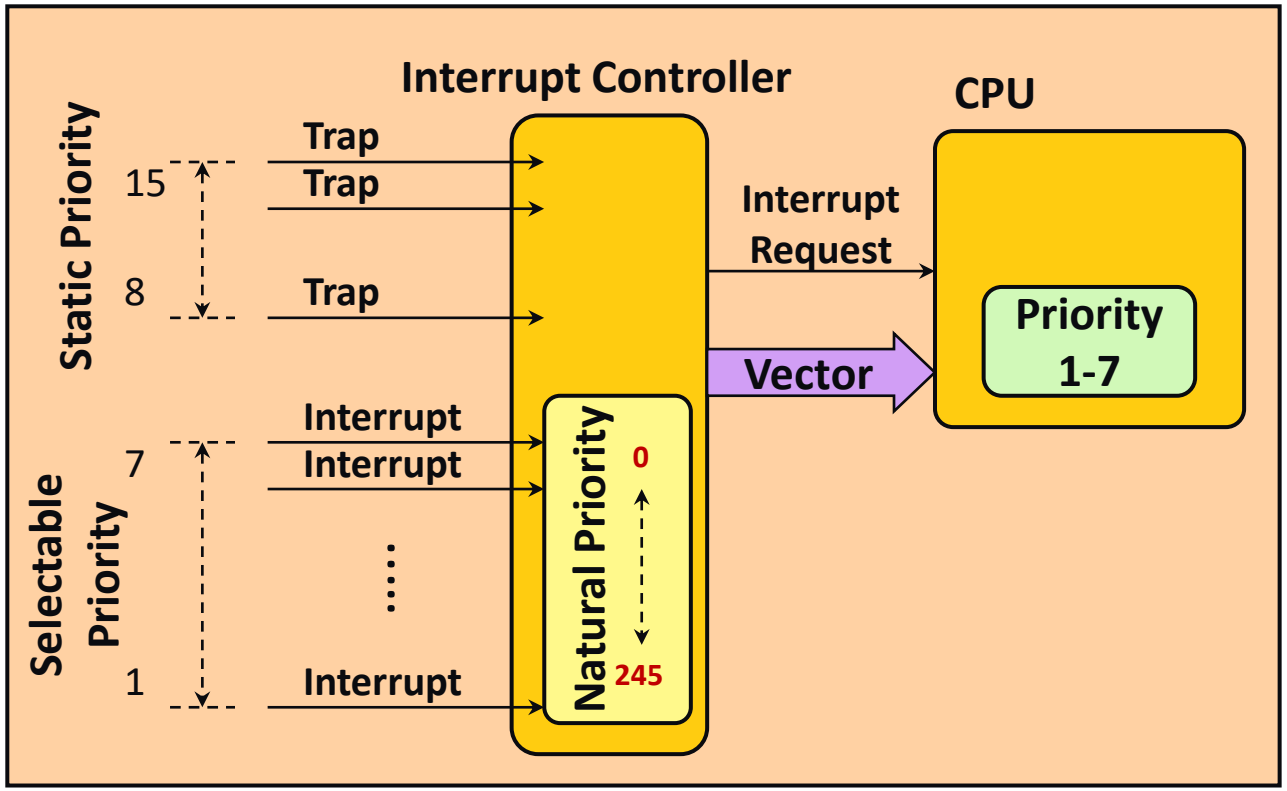
Which events need Interrupt ?

- **For General (maskable)**
  - Time critical event
    As soon as possible, when event occurred.
  - Unpredicted event
    Polling too waste computing time.

- **For Emergency (non maskable)**
  - Stack Overflow/Underflow
  - Math Error
  - Address Error
  - etc..

dsPIC33CK Series Interrupt Architecture

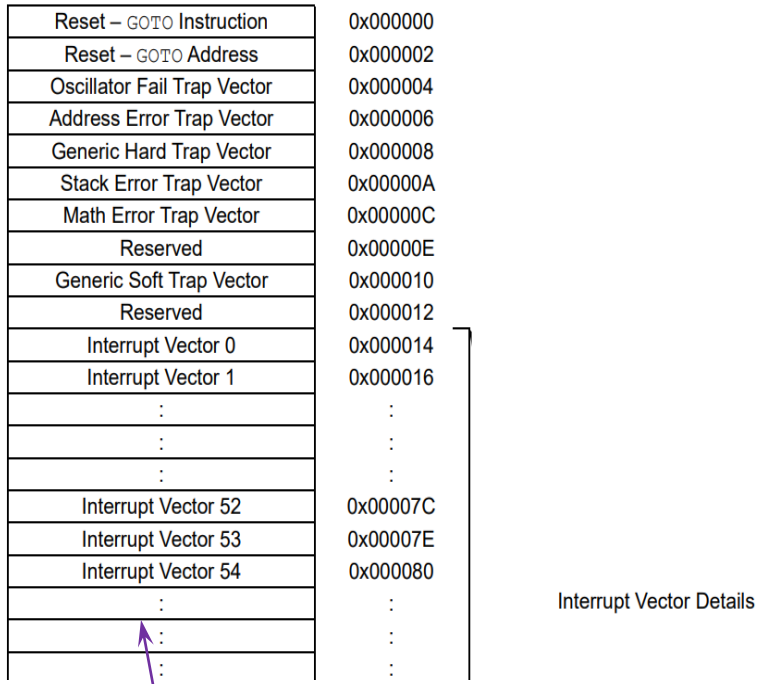
- dsPIC33's Interrupt is Nested Vector Interrupt Controller.
- Provide up to 8 processor exceptions and software traps
- Provide 7 (1~7) user-selectable priority levels. 0 is disable, 7 is the Highest priority.
- IVT/AIVT provide 246 vectors.
- Fixed interrupt entry and return latencies.



Natural Priority

- **If two pending interrupts share the same priority, priority is given to the interrupt with the lowest exception number.** **(.\Microchip\xc16\v2.00\docs\vector_docs\PIC33CK256MP506)**

(lowest interrupt vector address).



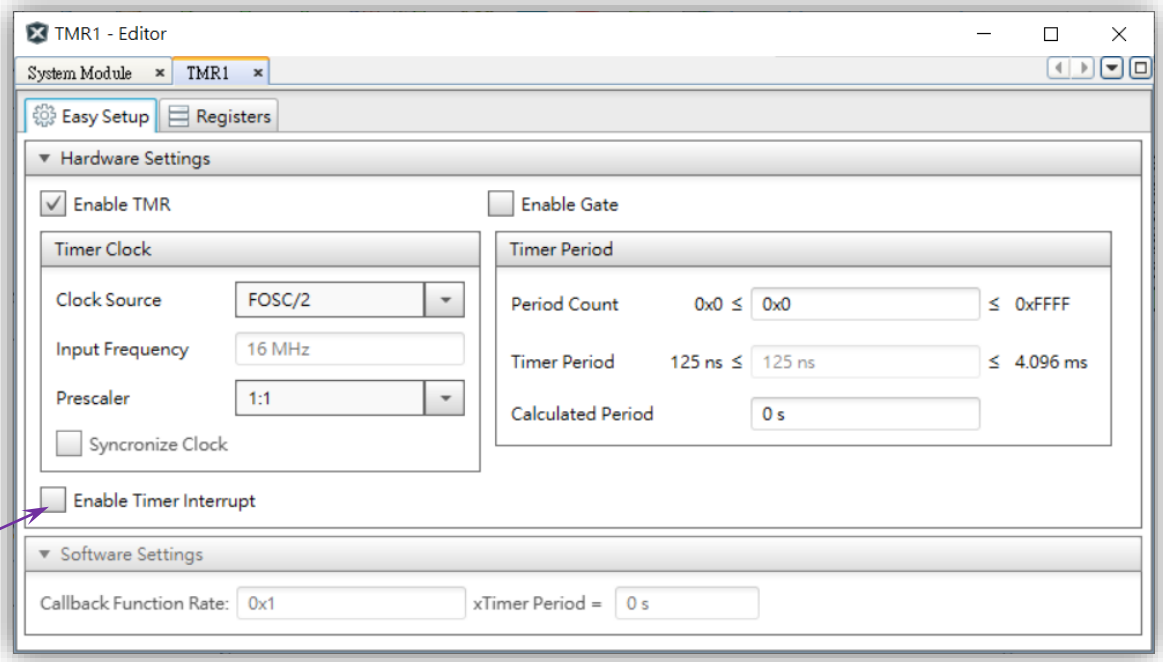
IRQ#	Primary Name	Alternate Name	Vector Function
N/A	_OscillatorFail	_AltOscillatorFail	Oscillator Fail Trap vector
N/A	_AddressError	_AltAddressError	Address error Trap vector
N/A	_HardTrapError	_AltHardTrapError	Generic Hard Trap vector
N/A	_StackError	_AltStackError	Stack Error Trap Vector
N/A	_MathError	_AltMathError	Math Error Trap vector
N/A	_SoftTrapError	_AltSoftTrapError	Generic Soft Trap vector
0	_INT0Interrupt	_AltINT0Interrupt	External Interrupt 0
1	_T1Interrupt	_AltT1Interrupt	Timer 1
2	_CNAInterrupt	_AltCNAInterrupt	Change Notification A
3	_CNBInterrupt	_AltCNBInterrupt	Change Notification B
4	_DMA0Interrupt	_AltDMA0Interrupt	DMA Channel 0
6	_CCP1Interrupt	_AltCCP1Interrupt	CCP1 Capture/Compare Event
7	_CCT1Interrupt	_AltCCT1Interrupt	CCP1 Timer Event
8	_DMA1Interrupt	_AltDMA1Interrupt	DMA Channel 1
9	_SPI1RXInterrupt	_AltSPI1RXInterrupt	SPI1 RX
10	_SPI1TXInterrupt	_AltSPI1TXInterrupt	SPI1 TX
11	_U1RXInterrupt	_AltU1RXInterrupt	UART1 RX
12	_U1TXInterrupt	_AltU1TXInterrupt	UART1 TX

Each peripheral has an assigned.
Vector table is based at address 0x00000000

MICROCHIP

Coding for Interrupt

- **A complete interrupt code segment includes below 3 parts.**
  - Interrupt Service Routine (ISR)
  - Enable Interrupt and Set Interrupt Priority
  - 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, jone click all interrupt relate setting will get ready.**

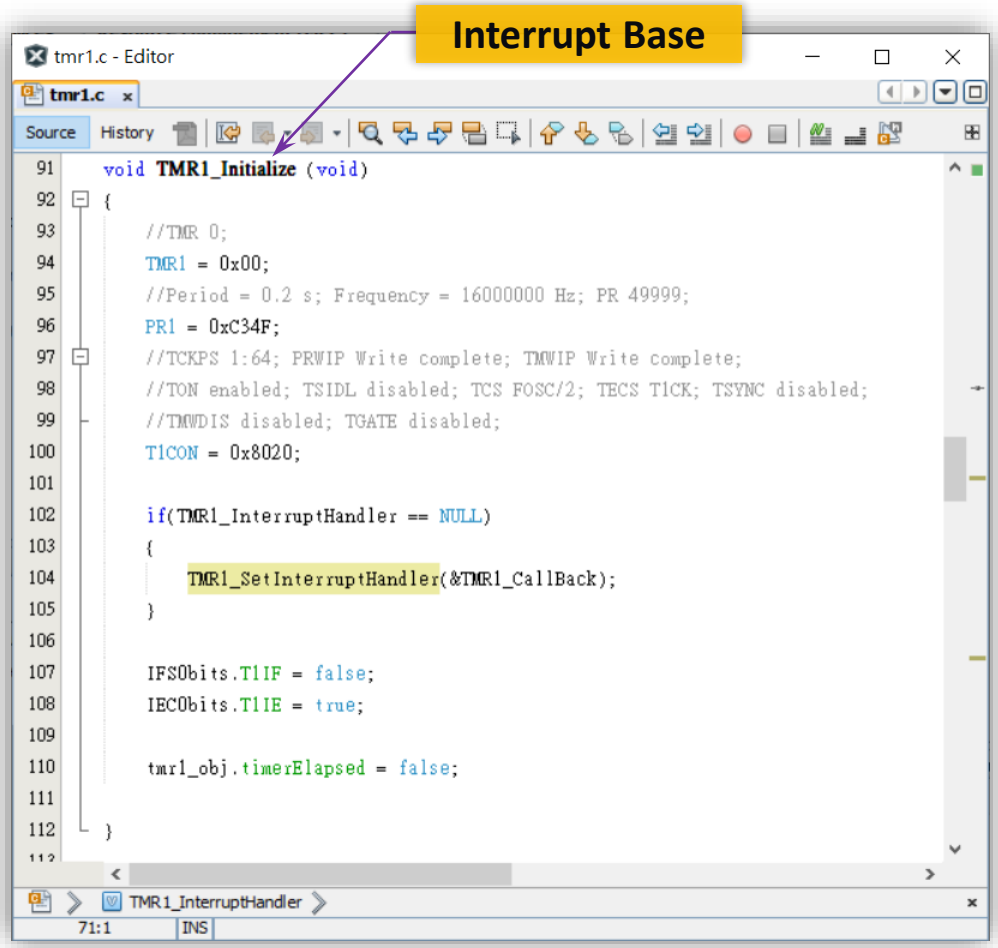


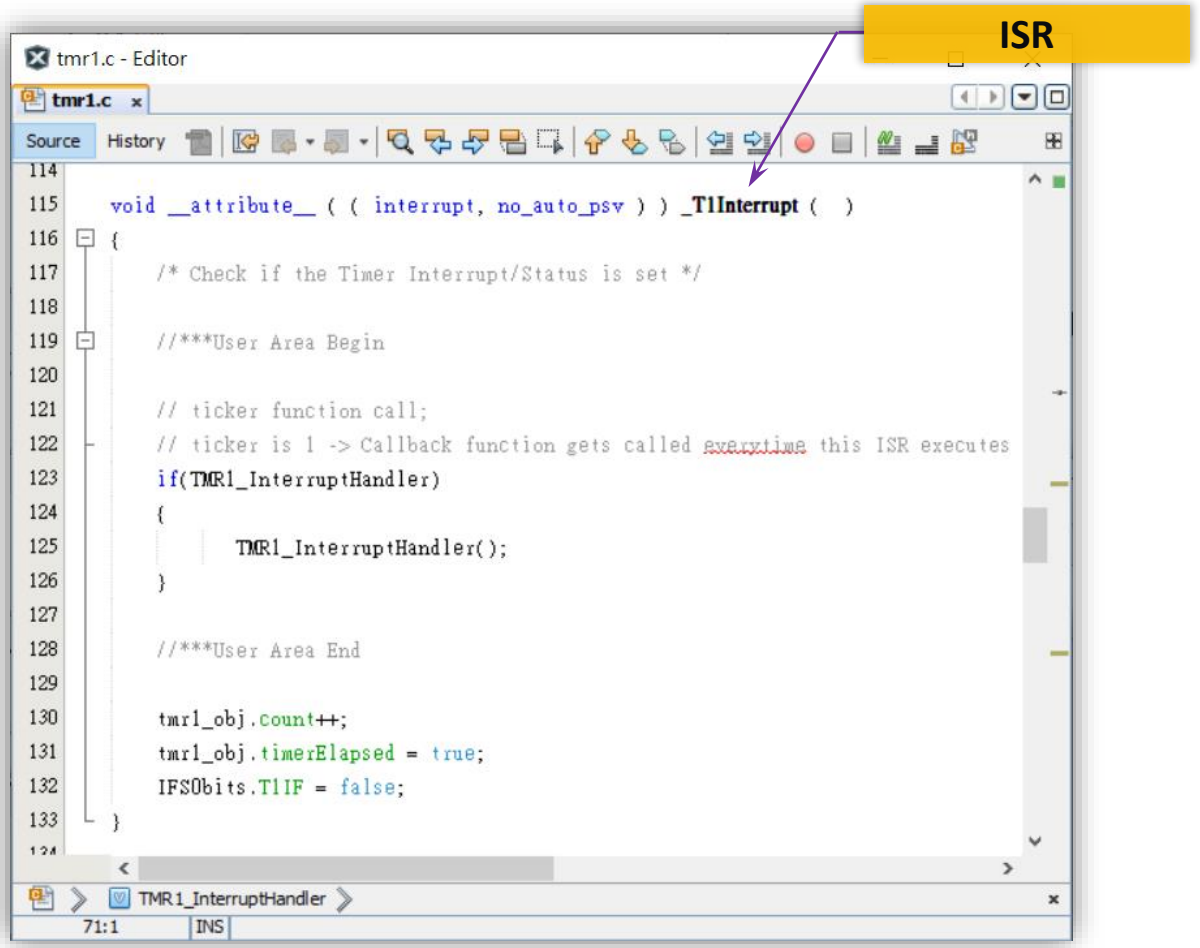
Just One Click !
Enable Interrupt

MICROCHIP

MCC's Interrupt Function

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

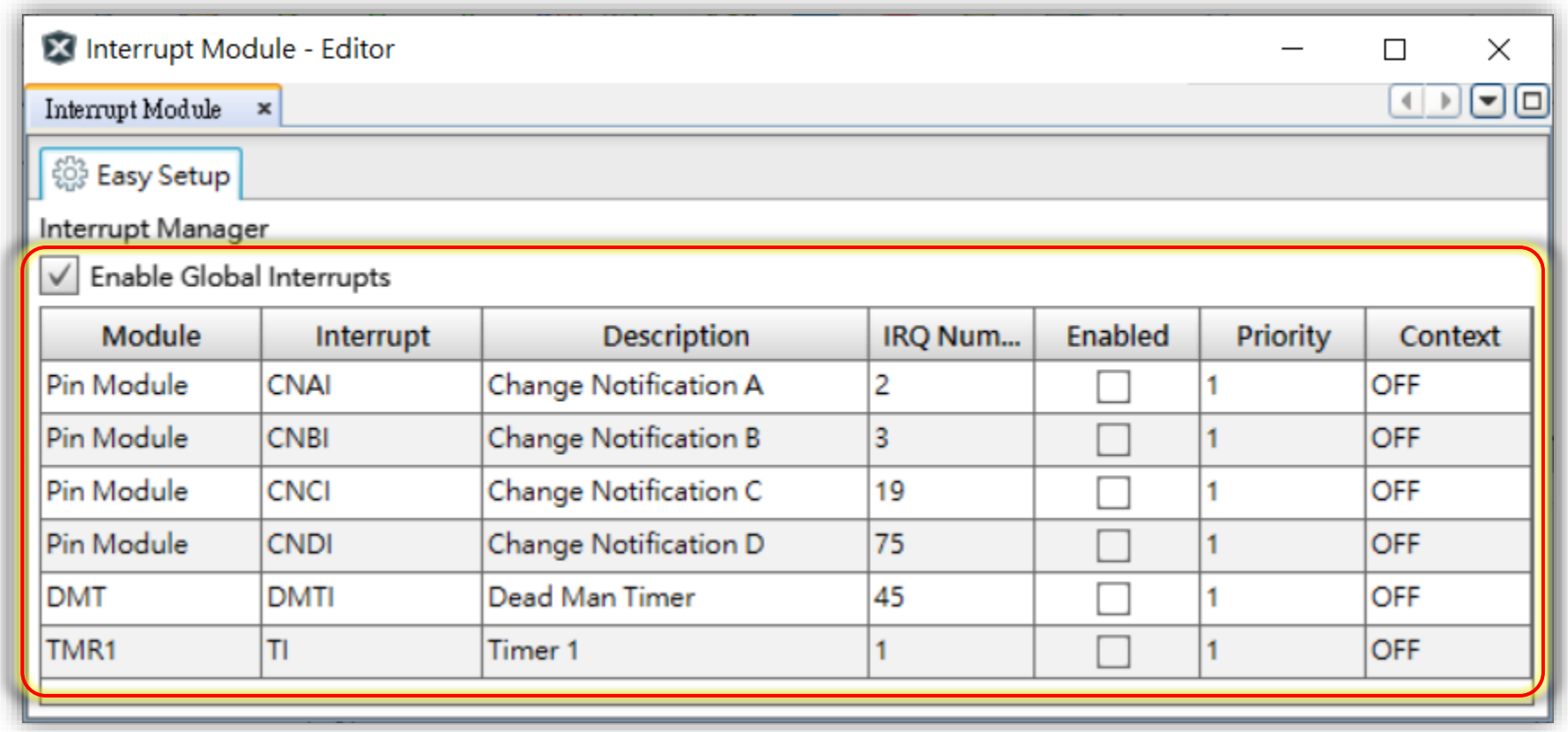




MICROCHIP

MCC’s Interrupt Manager

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



Module	Interrupt	Description	IRQ Num...	Enabled	Priority	Context
Pin Module	CNAI	Change Notification A	2	☐	1	OFF
Pin Module	CNBI	Change Notification B	3	☐	1	OFF
Pin Module	CNCI	Change Notification C	19	☐	1	OFF
Pin Module	CNDI	Change Notification D	75	☐	1	OFF
DMT	DMTI	Dead Man Timer	45	☐	1	OFF
TMR1	TI	Timer 1	1	☐	1	OFF

Callback Function

- Most interrupt handler function provide named "callback" function for user.
- Flag and event handle by handler function automatically, user focus at application only. It’s more powerful and easy to understand than operating a bare metal style.
- The callback function own named "weak" qualification. This is mean user can redefine your same name function in your code to replace the default function.

```
void __attribute__ ((weak)) XXX_CallBack(void)
{
   // Add your custom callback code here
}
```

MICROCHIP

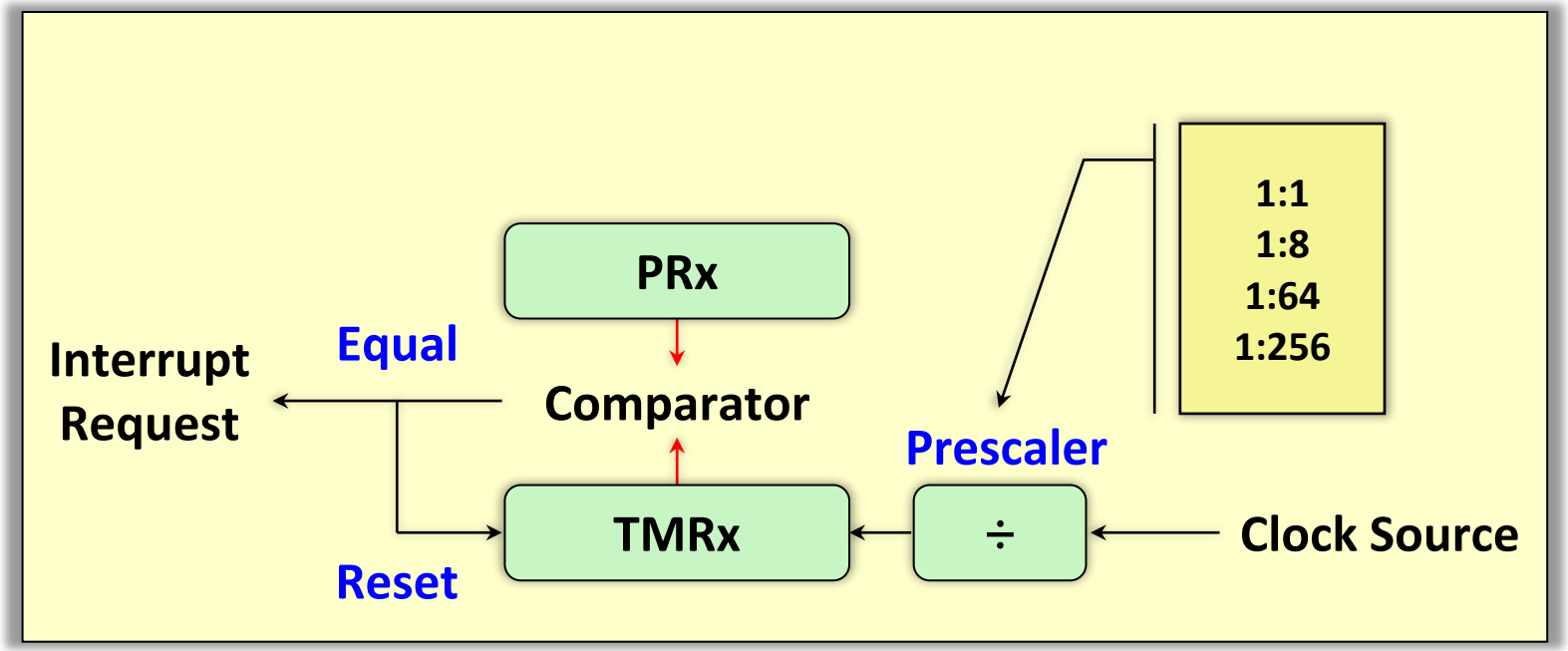
Timer Architecture

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.
- **dsPIC33CKMP506 provide 1 timer named "TMR1" only.**

dsPIC33CK Timer/Counter

- Please refer to the block diagram, Just a concept. The real and detail block diagram please refer to the following slide.
- Clock into TMRx after prescaler. Comparator continuously compares the values of TMRx and PRx. Asserts an interrupt request while TMRx and PRx equal and then clear TMRx to zero.
- You can fill a value to PRx to determine period you want.



MICROCHIP

Timer Functions of MCC

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

  Prototype definition : "tmr1.h"

  - void TMR1_Initialize (void);
  - void TMR1_Period16BitSet(uint16_t value);
  - uint16_t TMR1_Period16BitGet(void);
  - void TMR1_Counter16BitSet (uint16_t value);
  - uint16_t TMR1_Counter16BitGet(void);
  - void TMR1_CallBack(void);
  - void TMR1_Start(void);
  - void TMR1_Stop(void);
  - bool TMR1_GetElapsedThenClear(void);
  - int TMR1_SoftwareCounterGet(void);
  - void TMR1_SoftwareCounterClear(void);

MICROCHIP

Callback and Function Rate

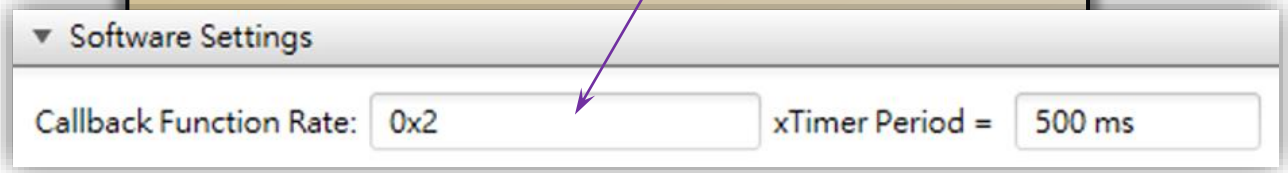
- **Timer defined named "void TMR1_CallBack(void)" weak function. Just define same name function to replace it.**

```
void __attribute__ ((weak)) TMR1_CallBack(void)
{
   // Add your custom callback code here
}
```

- **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 __attribute__((interrupt, no_auto_psv)) _T1Interrupt()
{
   static volatile unsigned int CountCallBack = 0;
   if (++CountCallBack >= TMR1_INTERRUPT_TICKER_FACTOR)
   {
      TMR1_CallBack();
      CountCallBack = 0;
   }
   ....
}
```

extend your timer period by software artifice.



MICROCHIP

Lab5 Timer1 Interrupt

- Base on current project or Open .\Exams\Lab5_xxx.X to do exercises.
- Try to add TMR1 to replace software delay to control LEDs toggle period.
- The Timer1 setup to timer mode, timer period is 500ms (250ms x 2).
- Timer1 interrupt priority set to level 1

Let’s go!

MICROCHIP

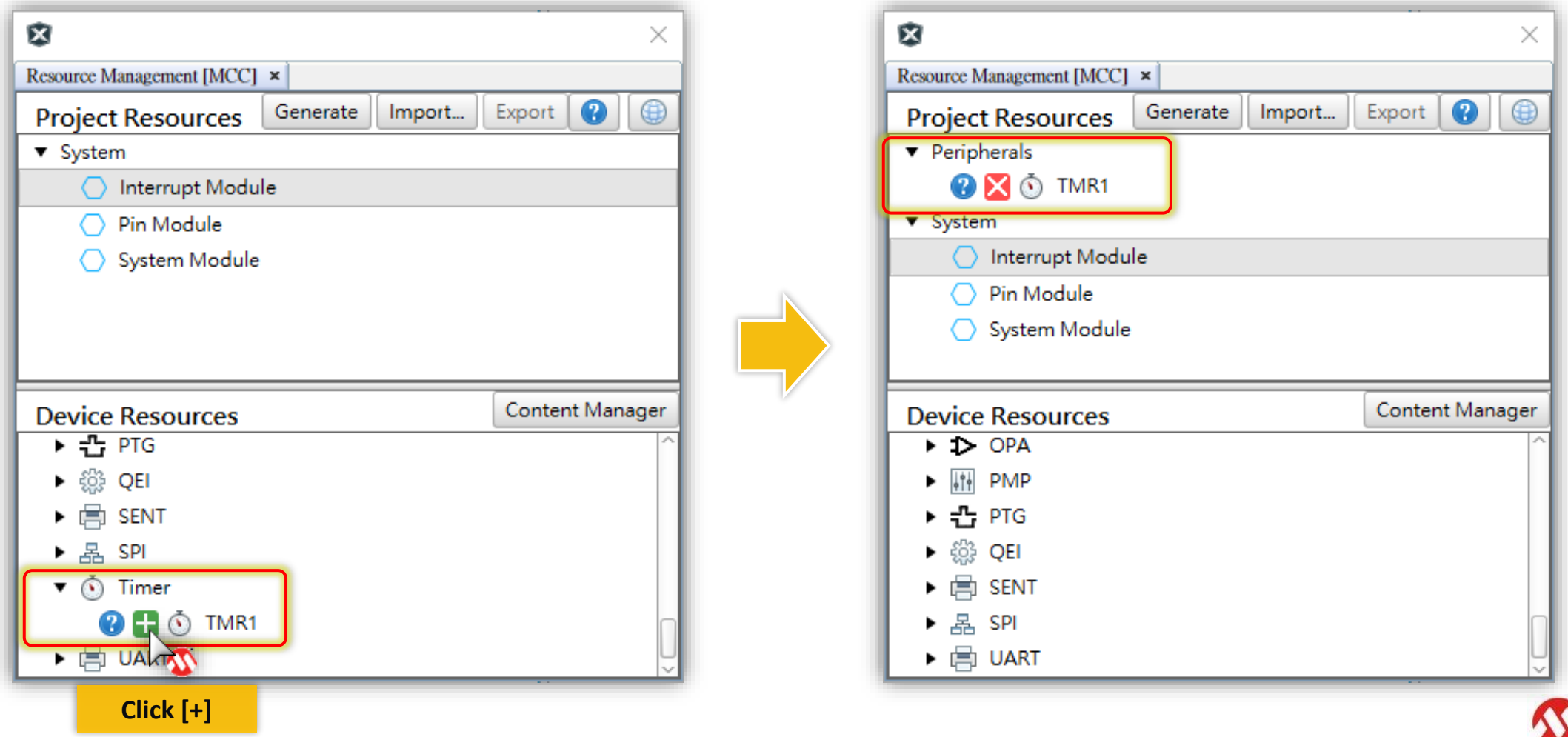
Lab5 Timer1 Interrupt

Connection

Lab5 Timer1 Interrupt

Step 1

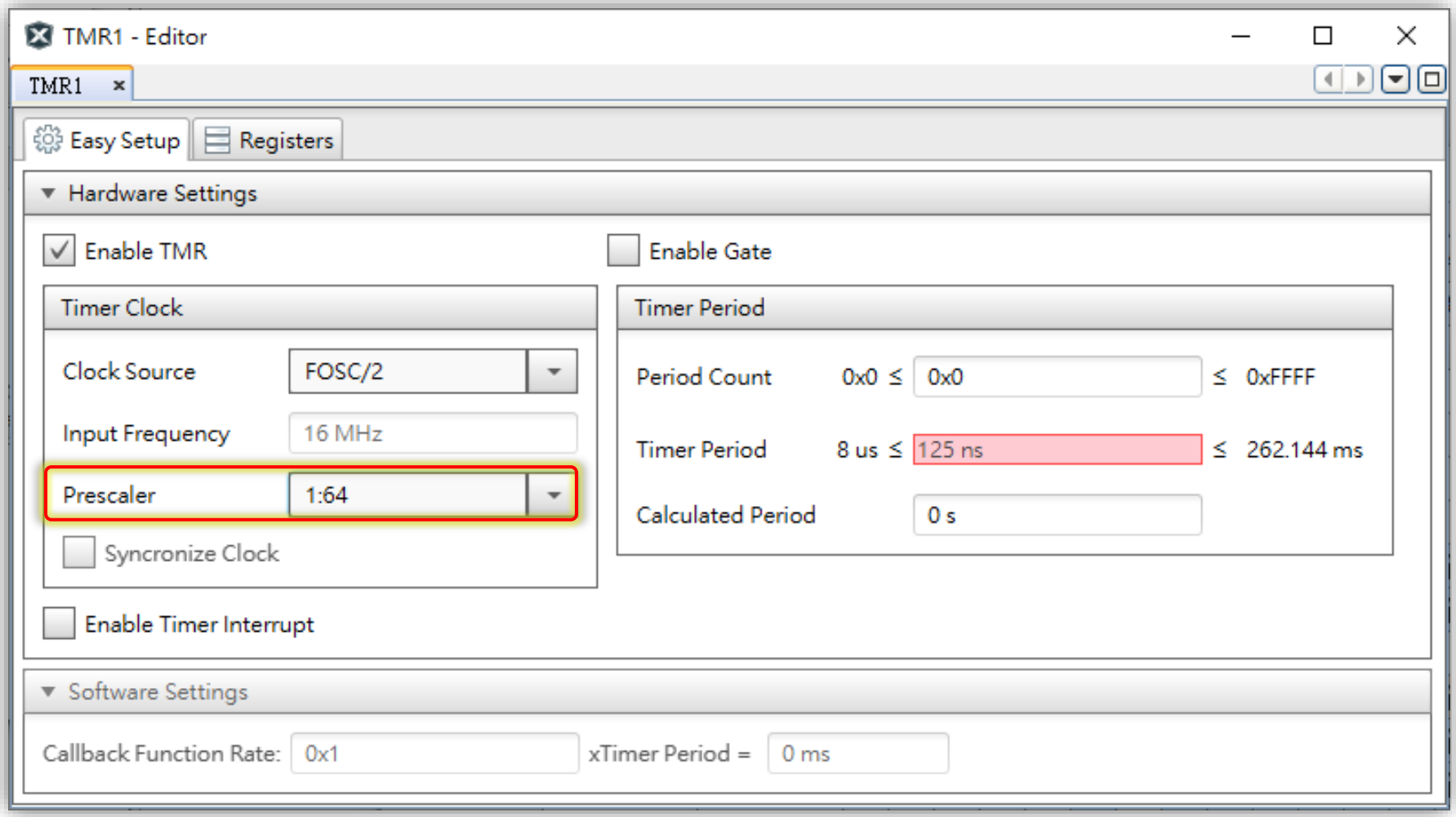
- **Click [+] to add Timer > TMR1 Module.**



Lab5 Timer1 Interrupt

Step 2

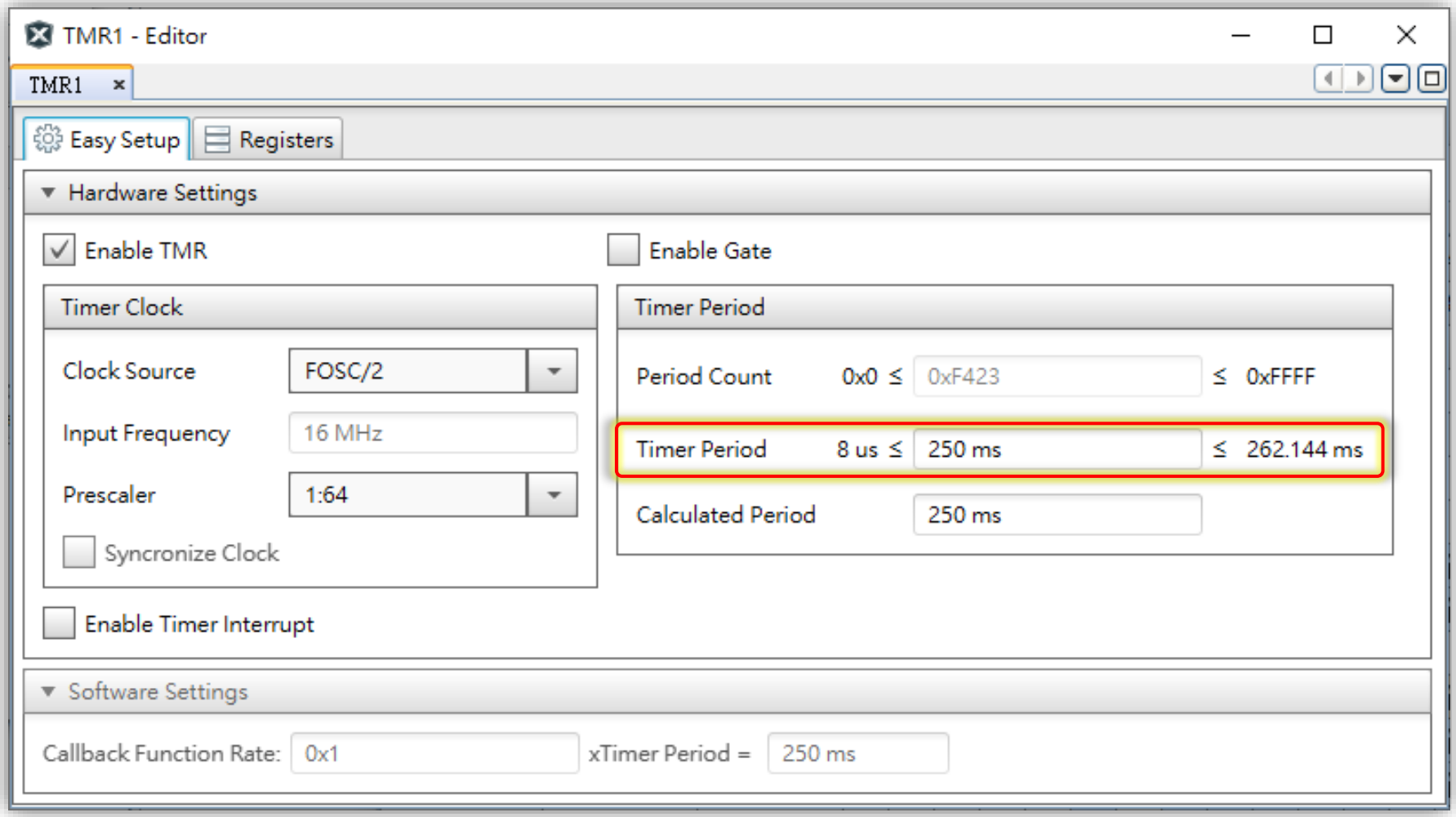
- Set **Prescaler** : **1:1** > **1:64**.



Lab5 Timer1 Interrupt

Step 3

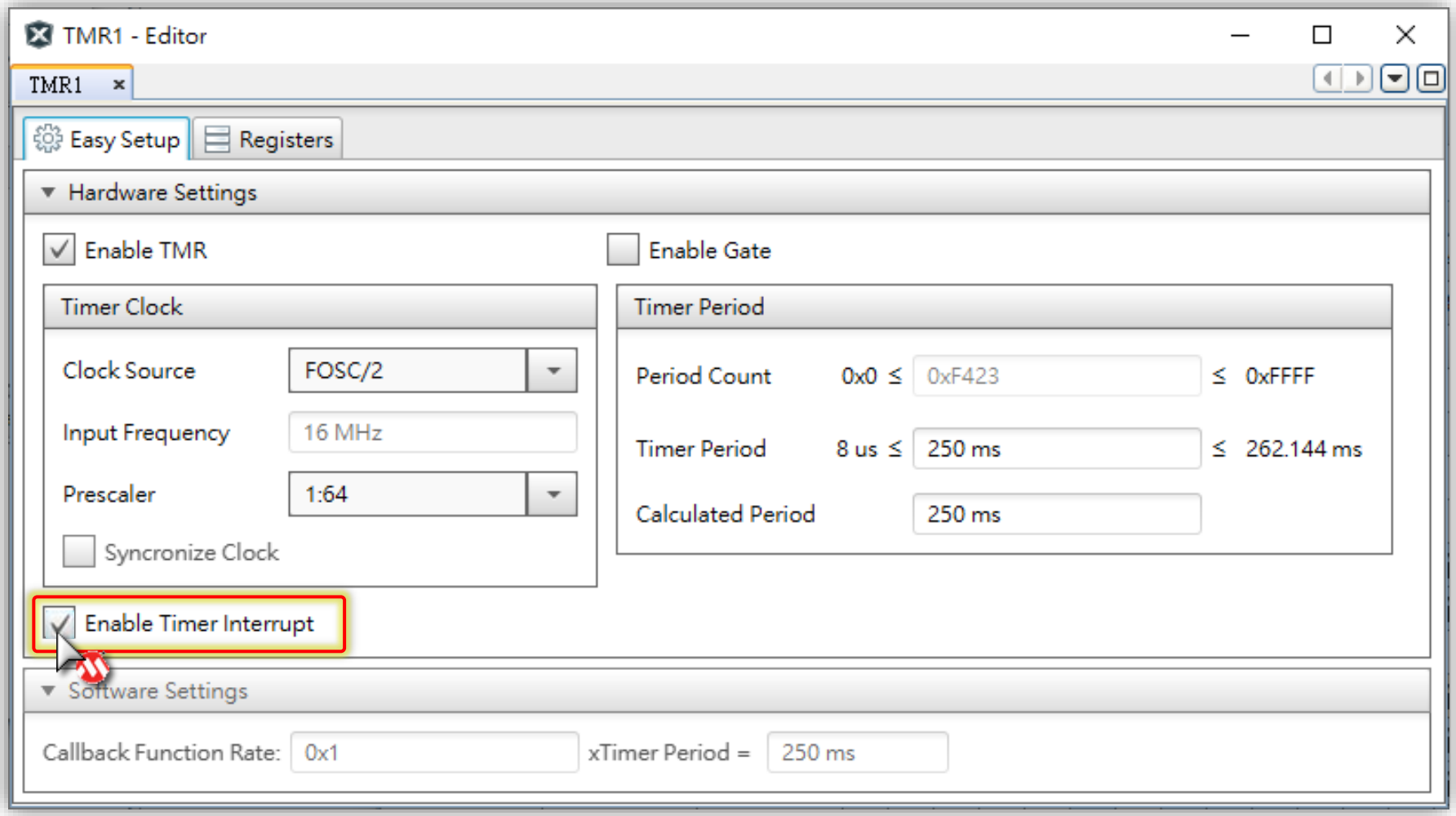
- Set **Time Period** : **125ns** > **250ms**.



Lab5 Timer1 Interrupt

Step 4

- Set **Enable Timer Interrupt**.

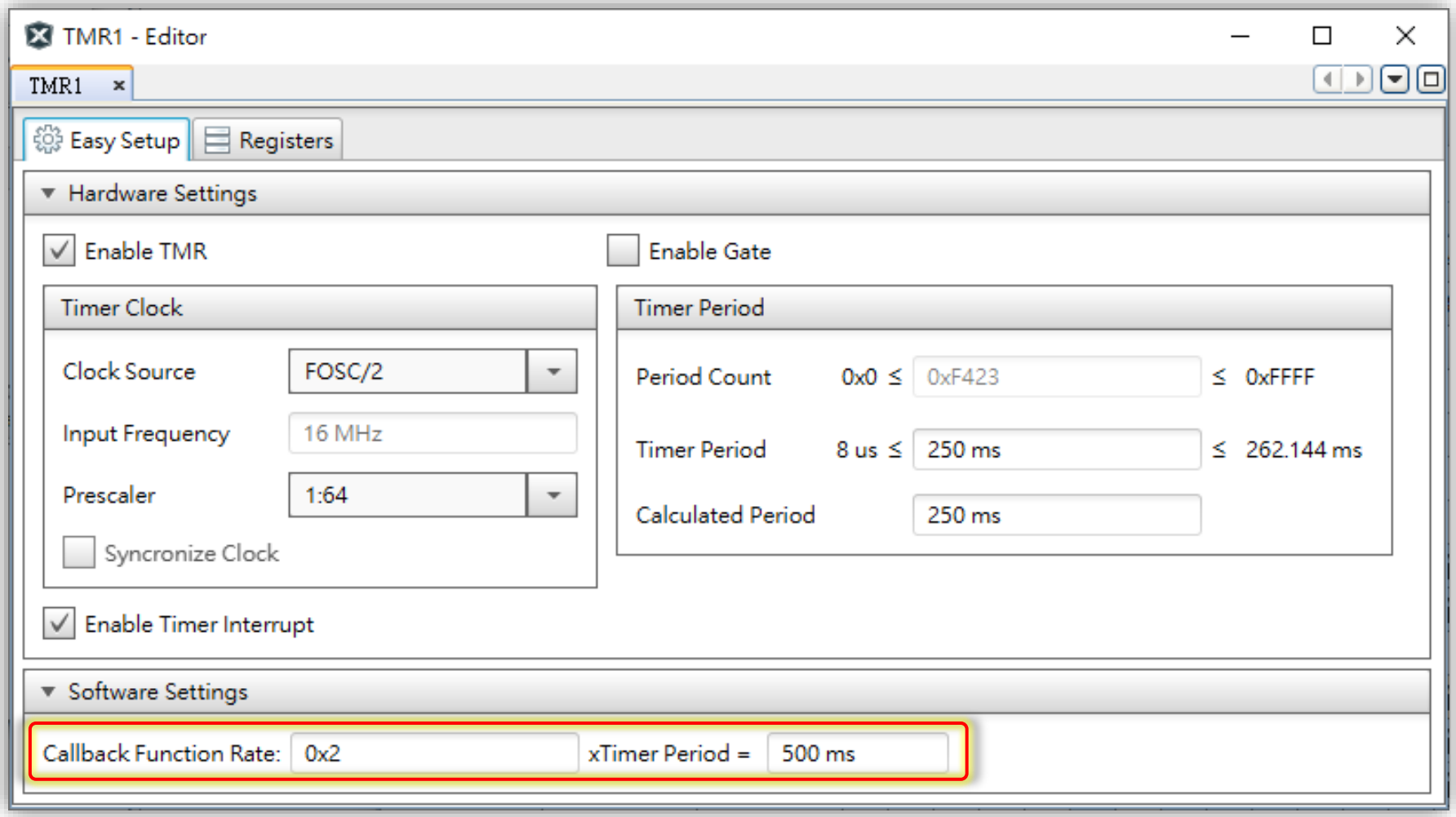


MICROCHIP

Lab5 Timer1 Interrupt

Step 5

- Set **Function Rate** : **0x01** > **0x02**

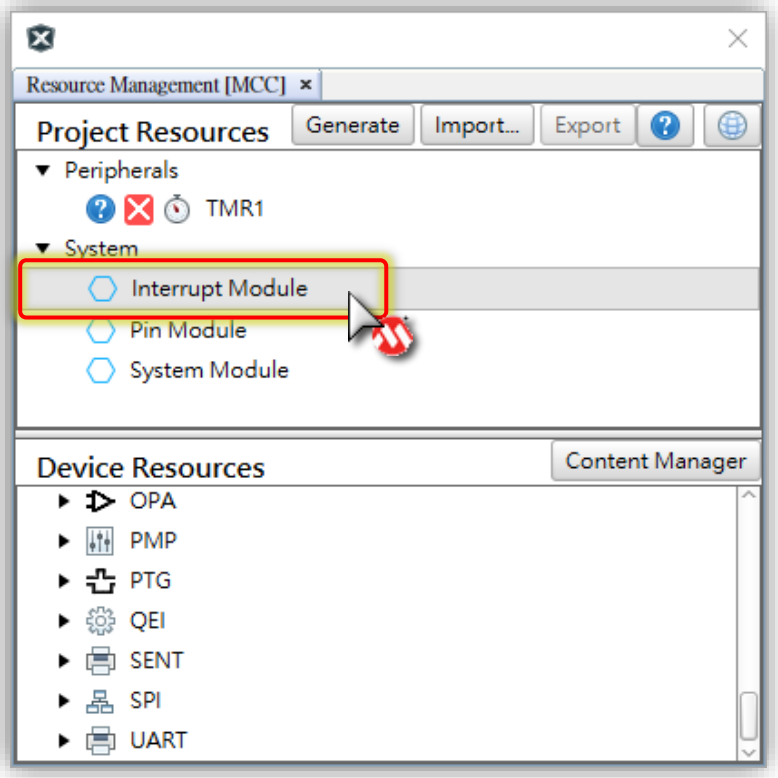


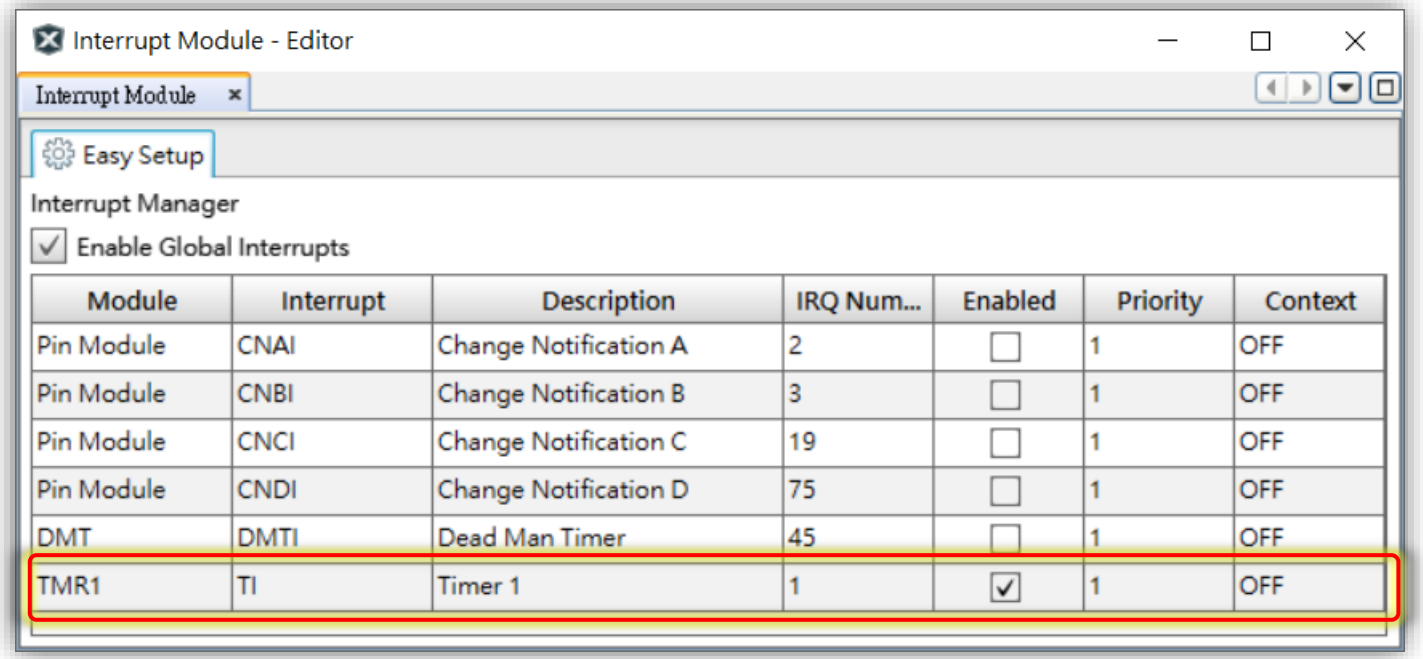
MICROCHIP

Lab5 Timer1 Interrupt

Step 6

- Check **Enable TMR1 Interrupt.**





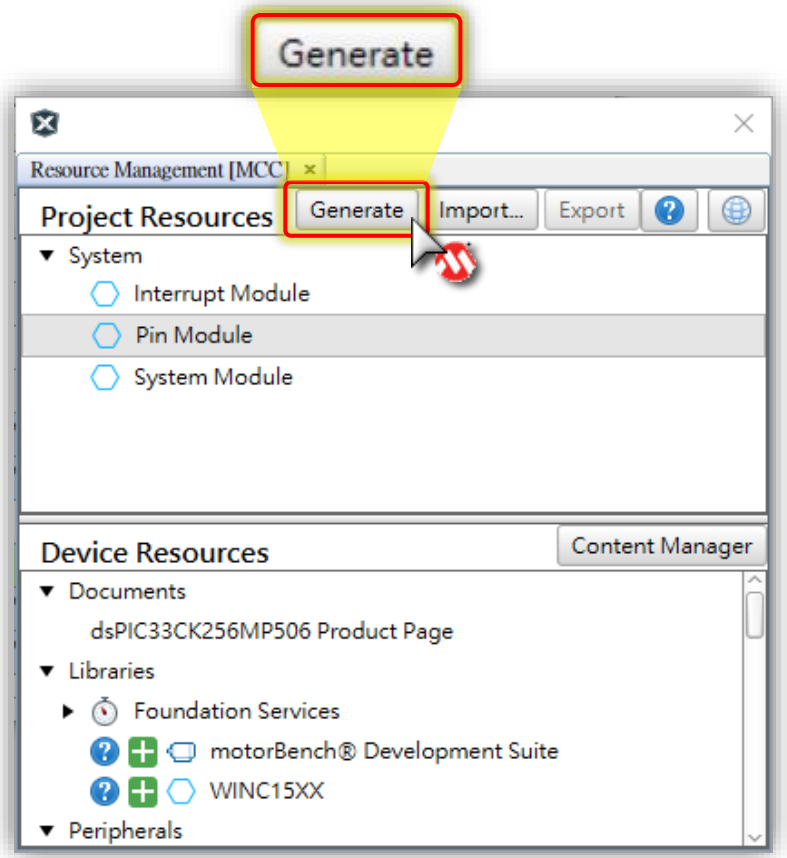
Module	Interrupt	Description	IRQ Num...	Enabled	Priority	Context
Pin Module	CNAI	Change Notification A	2	☐	1	OFF
Pin Module	CNBI	Change Notification B	3	☐	1	OFF
Pin Module	CNCI	Change Notification C	19	☐	1	OFF
Pin Module	CNDI	Change Notification D	75	☐	1	OFF
DMT	DMTI	Dead Man Timer	45	☐	1	OFF
TMR1	TI	Timer 1	1	☑	1	OFF

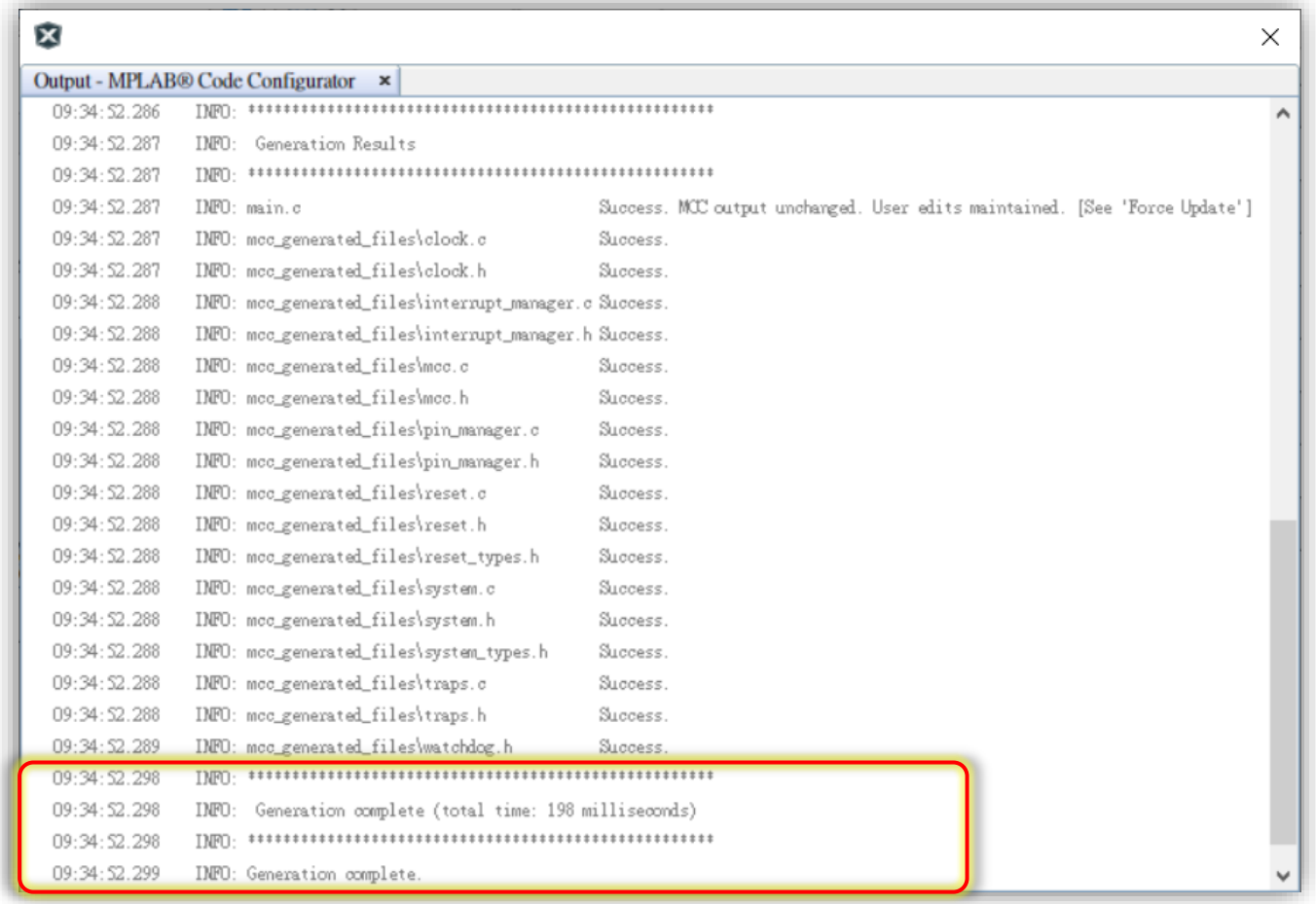
MICROCHIP

Lab5 Timer1 Interrupt

Step 6

- Click "Generate" Button to generate your code.





Lab5 Timer1 Interrupt

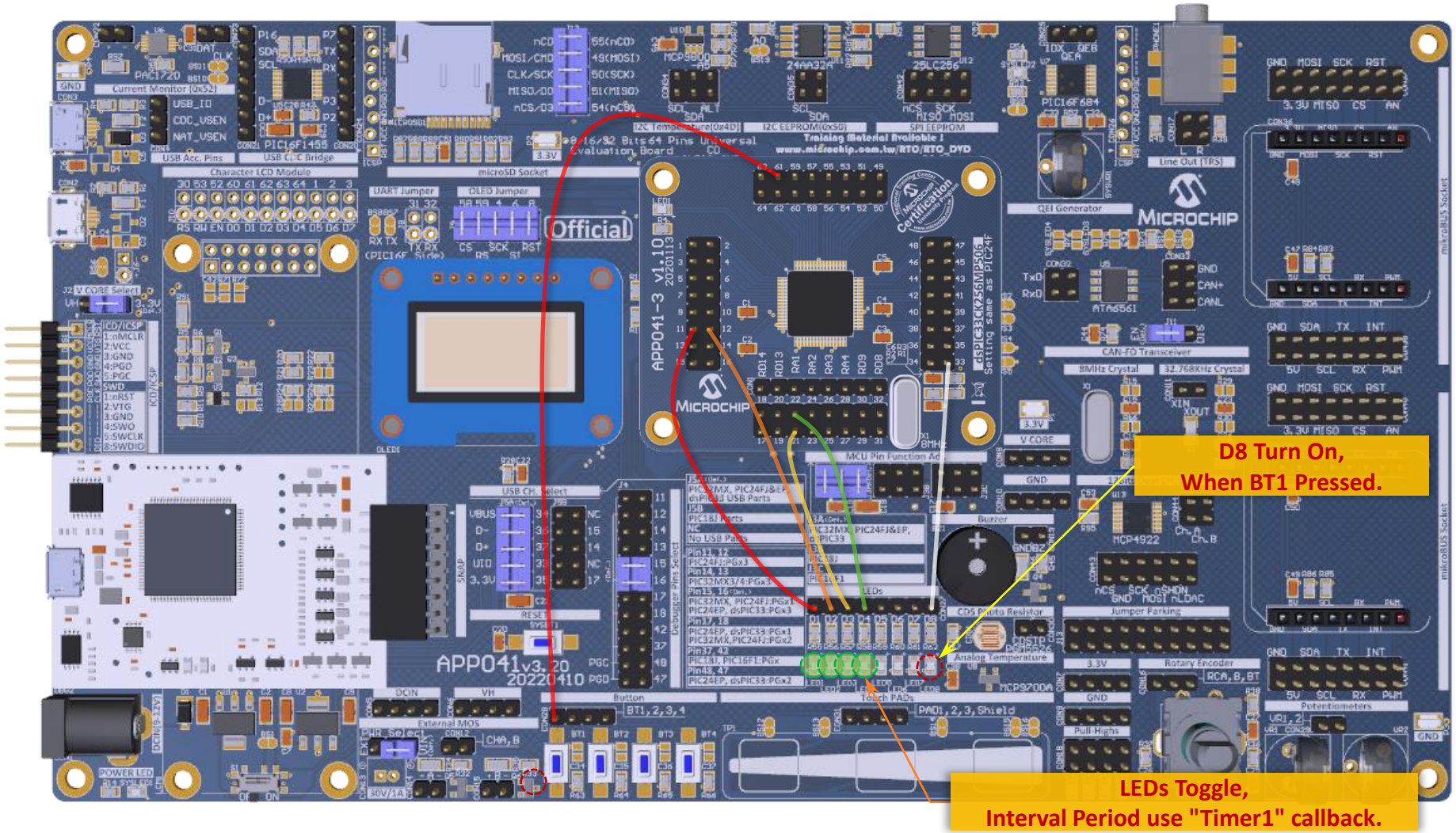
Code Example

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/pin_manager.h"

uint32_t  i = 0;

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

    while (1)
    {
        // Add your application code

        if(BT1_GetValue())
            LED8_SetLow();
        else
            LED8_SetHigh();
    }
    return 1;
}

void TMR1_CallBack(void)
{
    LED1_Toggle();
    LED2_Toggle();
    LED3_Toggle();
    LED4_Toggle();
}
```

MICROCHIP

Lab5 Timer1 Interrupt

Result

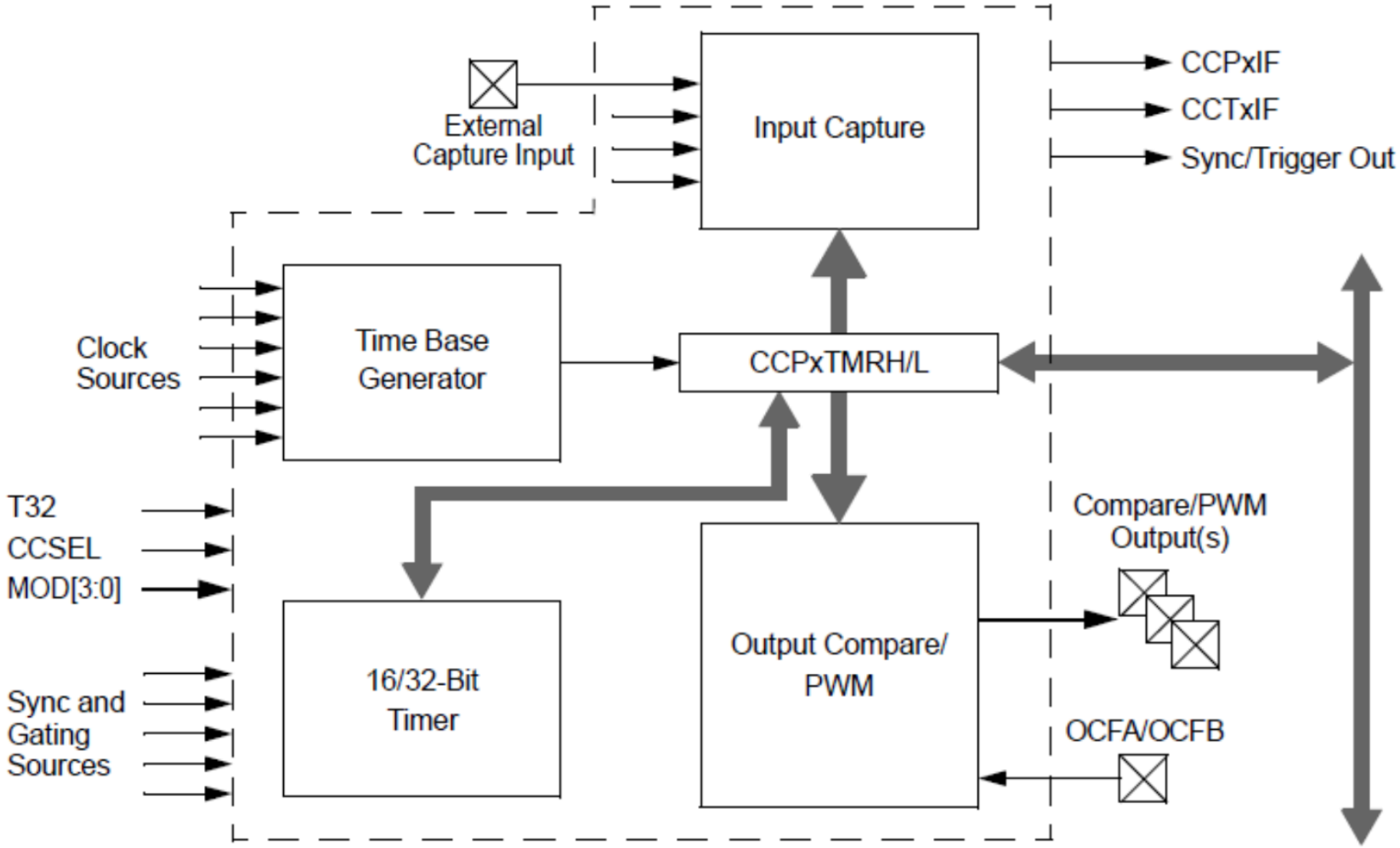


MCCP & SCCP Architecture

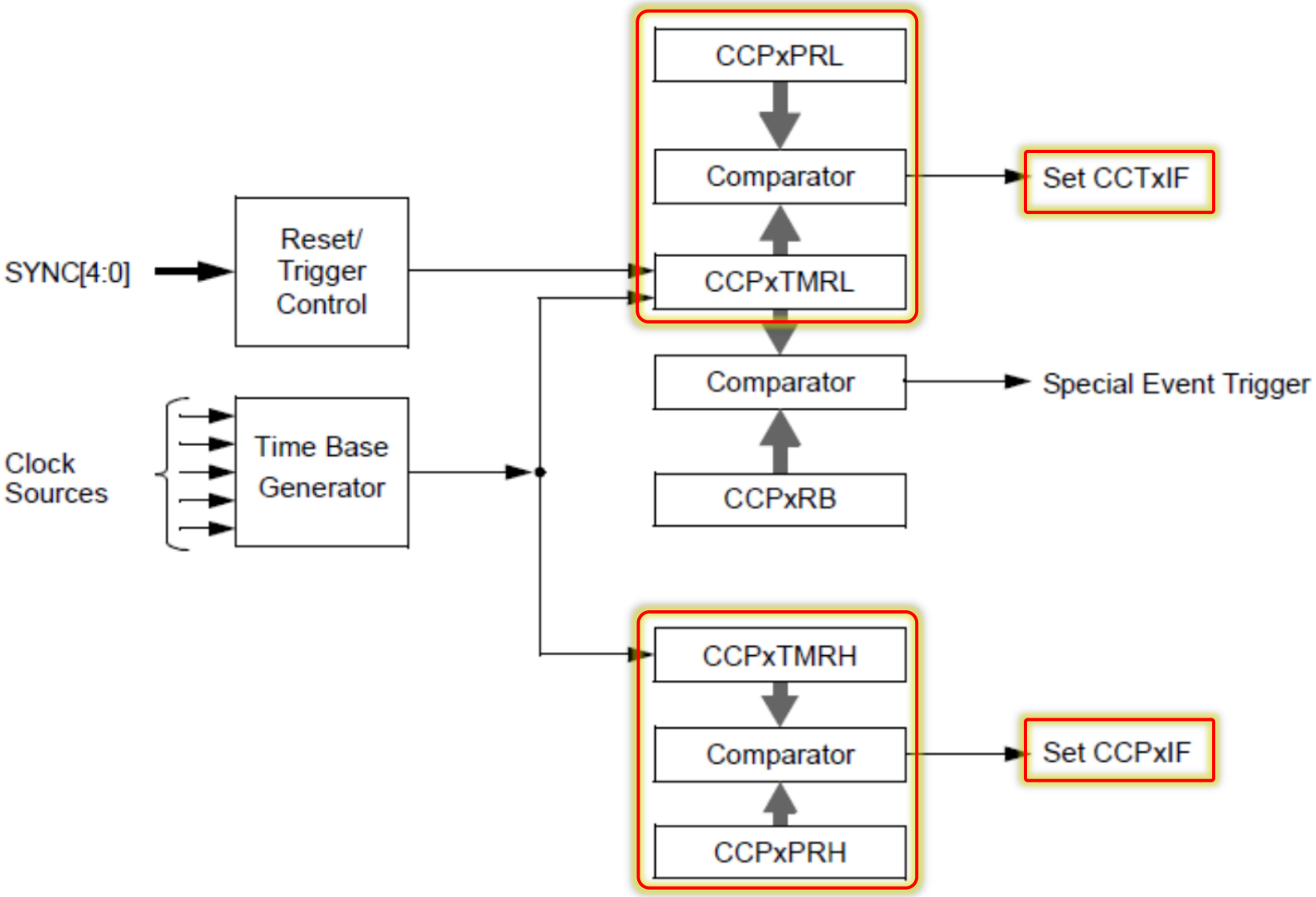
SCCP Module

- **SCCP can operate in one of three major modes:**
  - General Purpose Timer.
  - Input Capture.
  - Output Compare & PWM.
- **Output Compare/PWM Modes:**
  - Single Edge and Dual Edge Compare modes.
  - Center-Aligned Compare mode.
  - Variable Frequency Pulse mode.
  - External Input mode.
- **dsPIC33CK256MP506 include 8 SCCP and 1 MCCP.**
  - SCCP > 1 x PWM Channel.
  - MCCP > 6 x PWM Channels.
- **In this section, we focus on Output Compare & PWM mode.**

Block diagram of SCCP

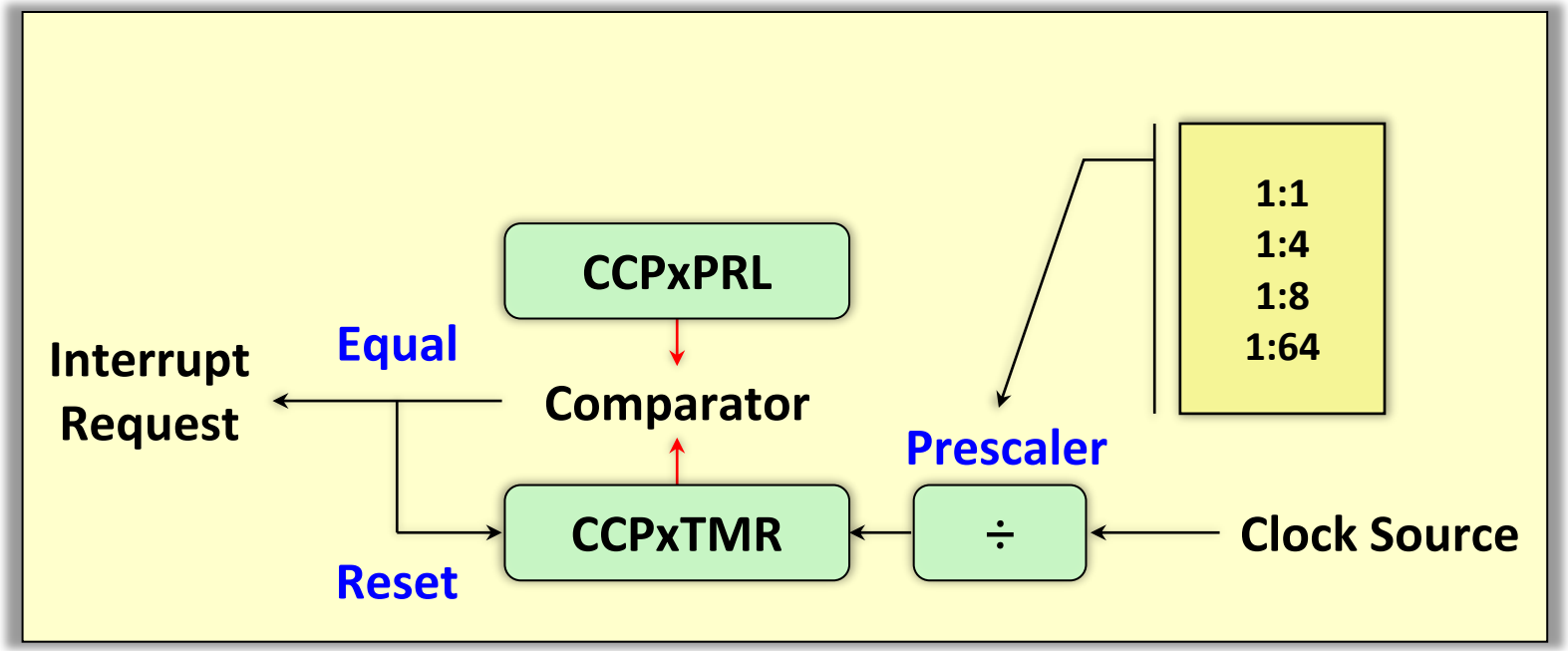


Block diagram of SCCP - Timer



Timer/Counter Operation Mode

- Please refer to the block diagram, Just a concept. The real and detail block diagram please refer to the following slide.
- Clock into CCPxTMR after prescaler. Comparator continuously compares the values of CCPxTMR and CCPxPRL. Asserts an interrupt request while CCPxTMR and CCPxPRL equal and then clear CCPxTMR to zero.
- You can fill a value to CCPxPRL to determine period you want.



Lab6 SCCP Timer Interrupt

- Base on current project or Open .\Exams\Lab6_xxx.X to do exercises
- Try to add SCCP1 to your project to control LEDs toggle period.
- The SCCP1 setup to timer mode, timer period is 100ms (50ms x 2).
- To separate D1, D2 & D3, D4 blink period.
  - D1 and D2 blink by TMR1
    Period 500mS(250ms x 2), Priority : 1.
  - D3 and D4 blink by SCCP1
    Period 100ms(50ms x 2), Priority : 2.

Let’s go!

MICROCHIP

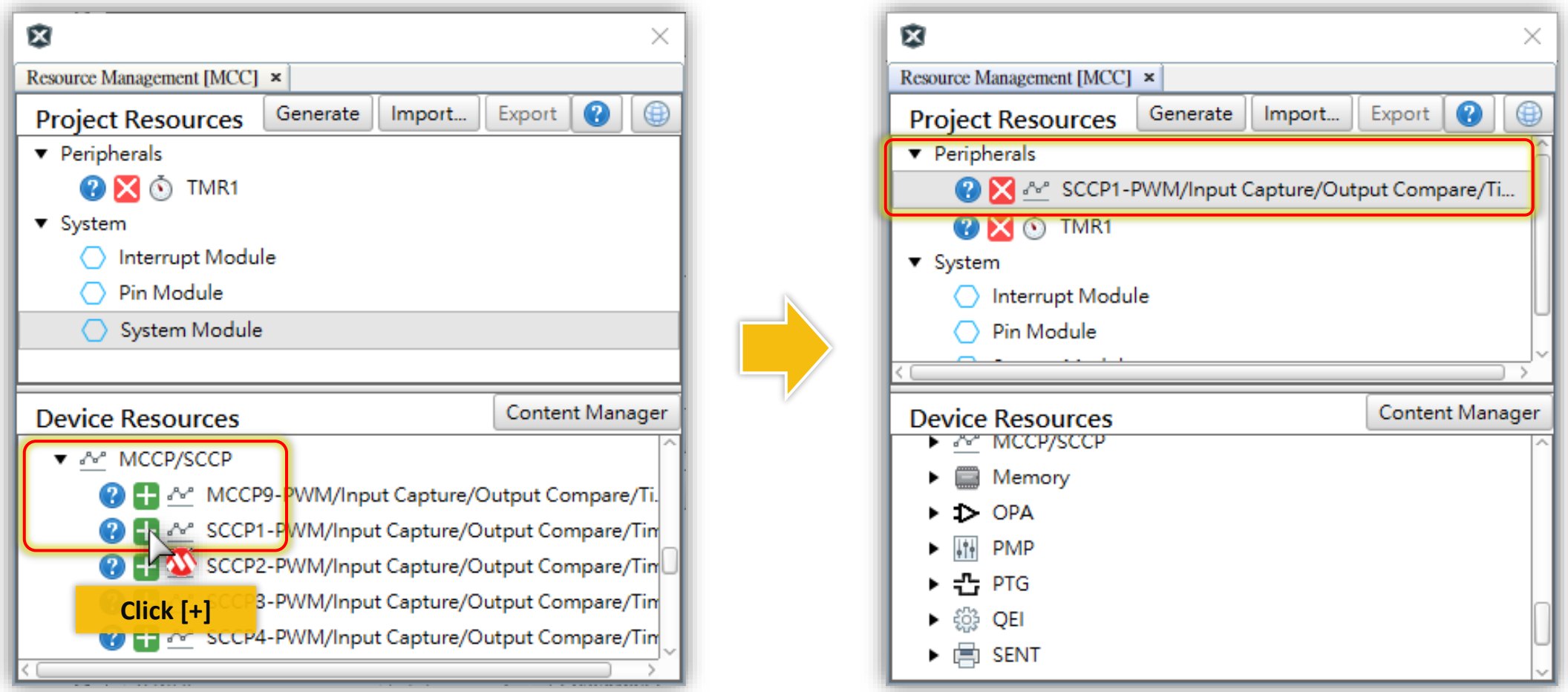
Lab6 SCCP Timer Interrupt

Connection

Lab6 SCCP Timer Interrupt

Step 1

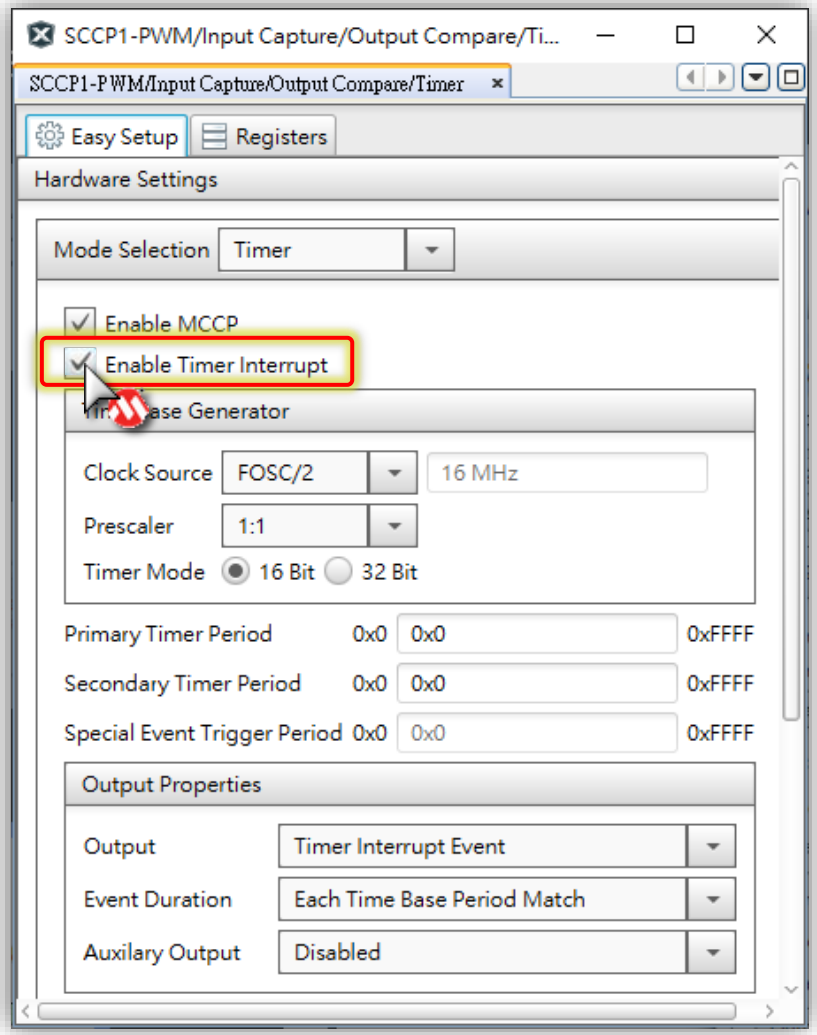
- **Click [+] to add MCCP/SCCP > SCCP1 Module.**



Lab6 SCCP Timer Interrupt

Step 2

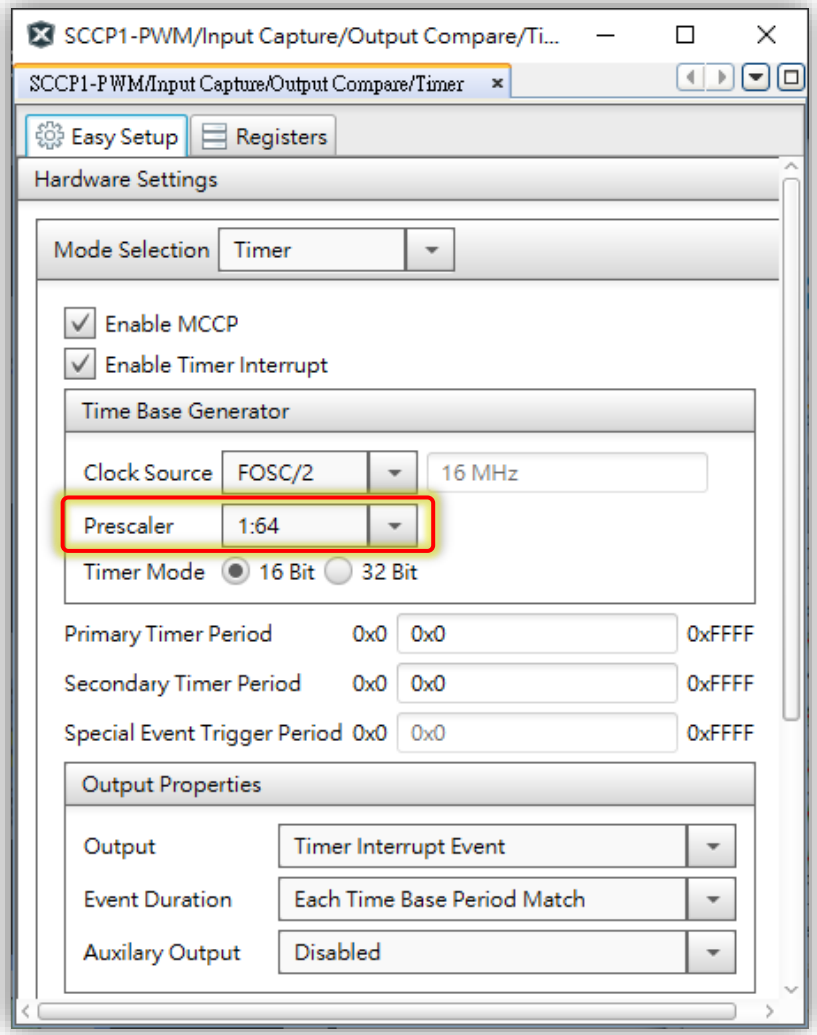
- Set **Enable MCCP Interrupt**.



Lab6 SCCP Timer Interrupt

Step 3

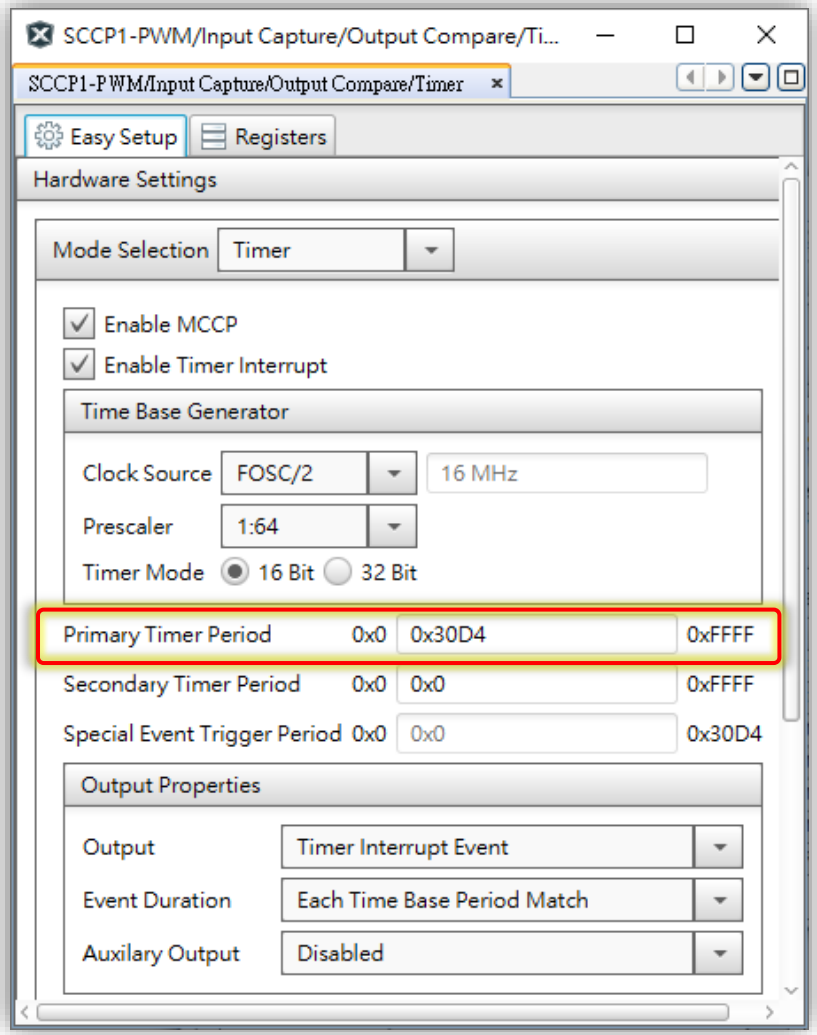
- Set **Prescaler** : **1:1** > **1:64**.



Lab6 SCCP Timer Interrupt

Step 4

- Set **Primary Timer Period** : **0x00** > **0x30D4 (12500)**.



$$CCPxPRL = \left(\frac{Clock\ Source}{Prescaler} \times Peroid\right)$$

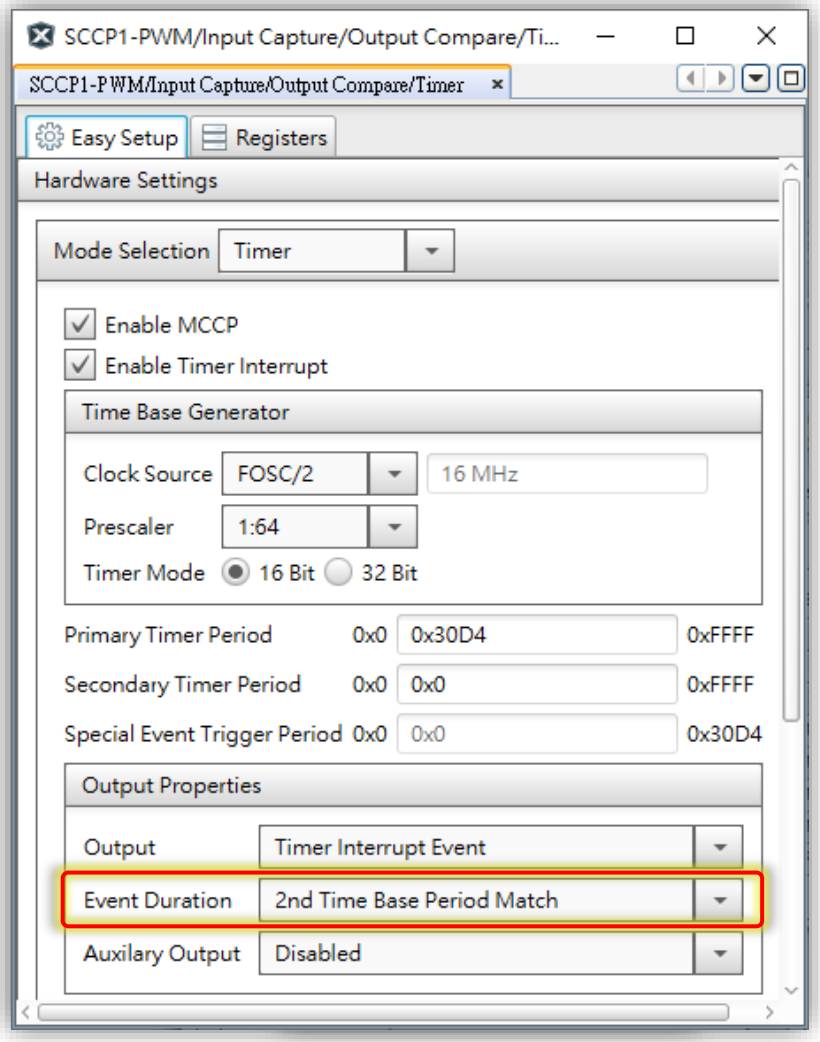
$$0x30D4 = 12500 = \left(\frac{16M}{64} \times 50ms\right)$$

MICROCHIP

Lab6 SCCP Timer Interrupt

Step 5

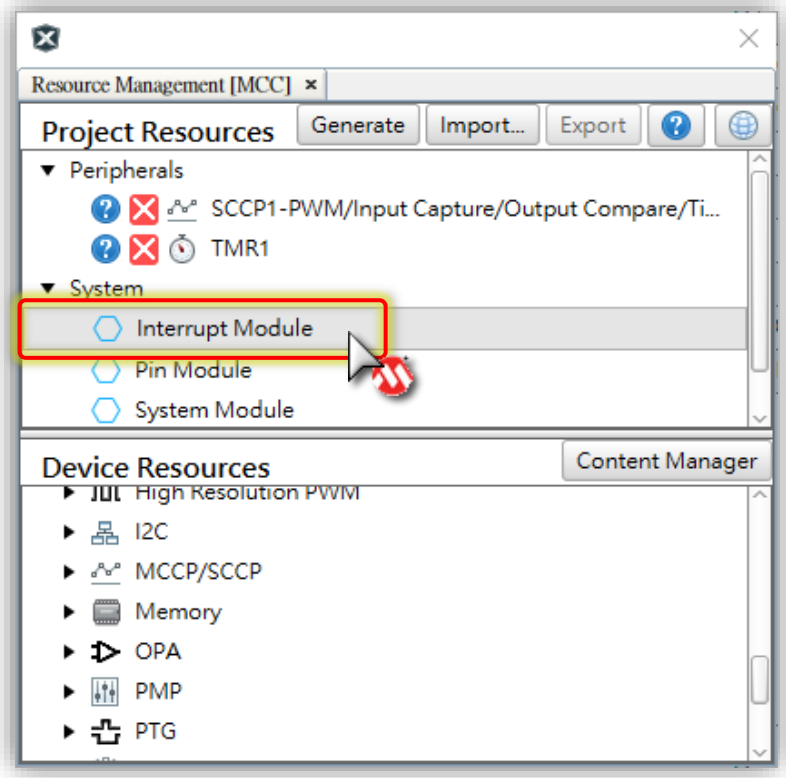
- Set **Event Duration** : **Each Time Base Period Match** > **2nd Time Base Period Match**.

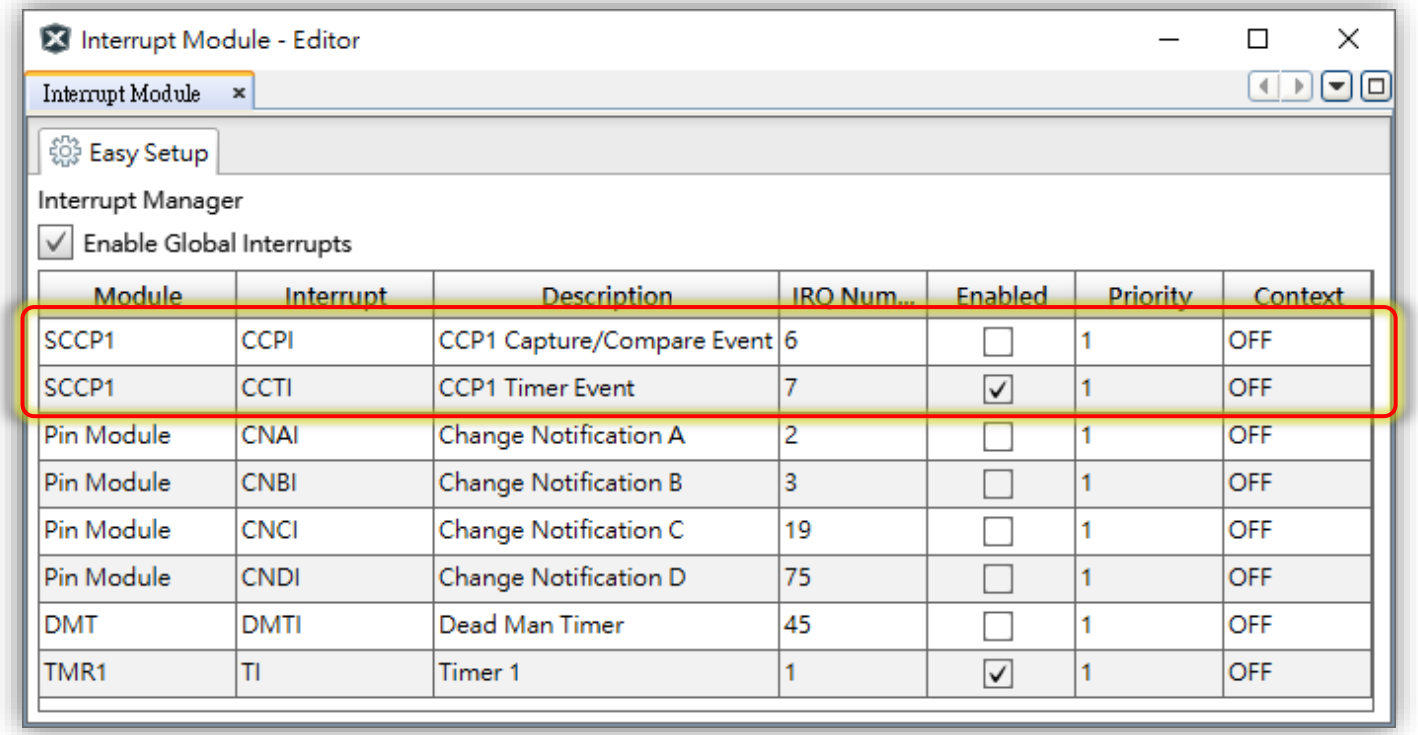


Lab6 SCCP Timer Interrupt

Step 6

- Open **Interrupt Module**.



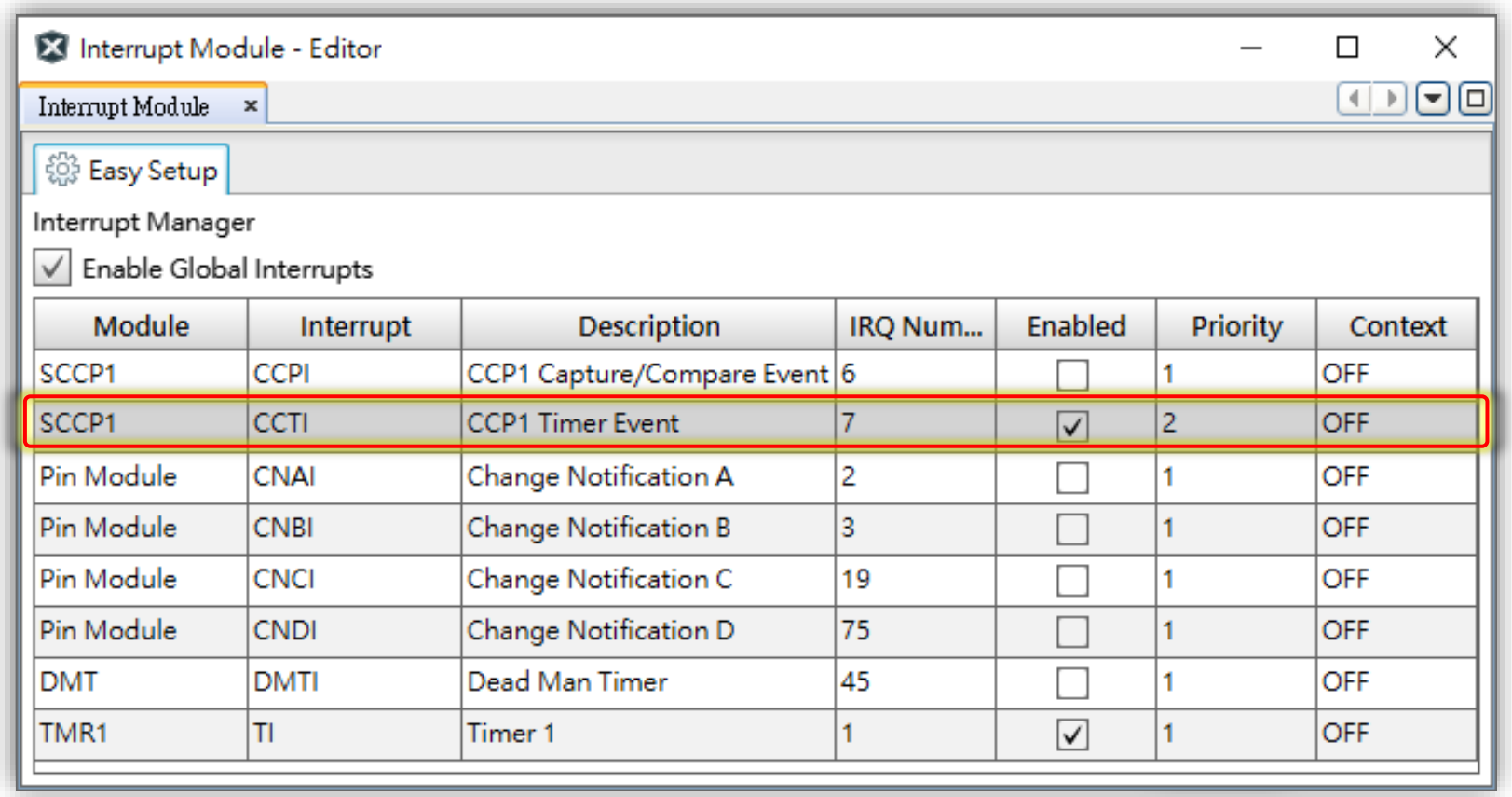


Module	Interrupt	Description	IRQ Num...	Enabled	Priority	Context
SCCP1	CCPI	CCP1 Capture/Compare Event	6	☐	1	OFF
SCCP1	CCTI	CCP1 Timer Event	7	☑	1	OFF
Pin Module	CNAI	Change Notification A	2	☐	1	OFF
Pin Module	CNBI	Change Notification B	3	☐	1	OFF
Pin Module	CNCI	Change Notification C	19	☐	1	OFF
Pin Module	CNDI	Change Notification D	75	☐	1	OFF
DMT	DMTI	Dead Man Timer	45	☐	1	OFF
TMR1	TI	Timer 1	1	☑	1	OFF

Lab6 SCCP Timer Interrupt

Step 7

- **Modify CCT1 Interrupt Priority : 1 > 2.**



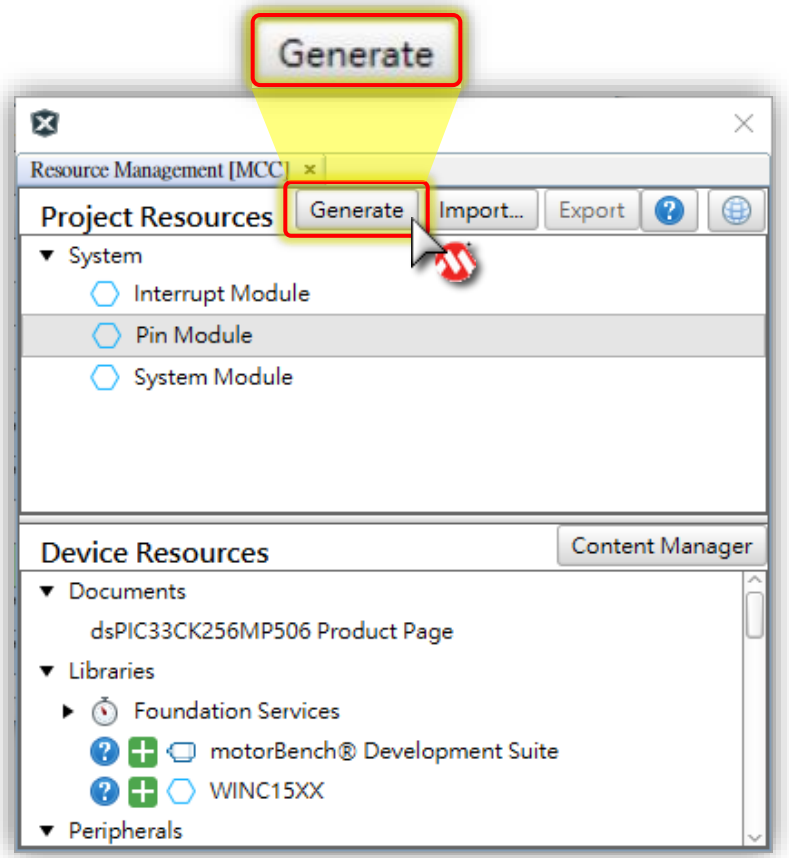
Module	Interrupt	Description	IRQ Num...	Enabled	Priority	Context
SCCP1	CCPI	CCP1 Capture/Compare Event	6	☐	1	OFF
SCCP1	CCTI	CCP1 Timer Event	7	☑	2	OFF
Pin Module	CNAI	Change Notification A	2	☐	1	OFF
Pin Module	CNBI	Change Notification B	3	☐	1	OFF
Pin Module	CNCI	Change Notification C	19	☐	1	OFF
Pin Module	CNDI	Change Notification D	75	☐	1	OFF
DMT	DMTI	Dead Man Timer	45	☐	1	OFF
TMR1	TI	Timer 1	1	☑	1	OFF

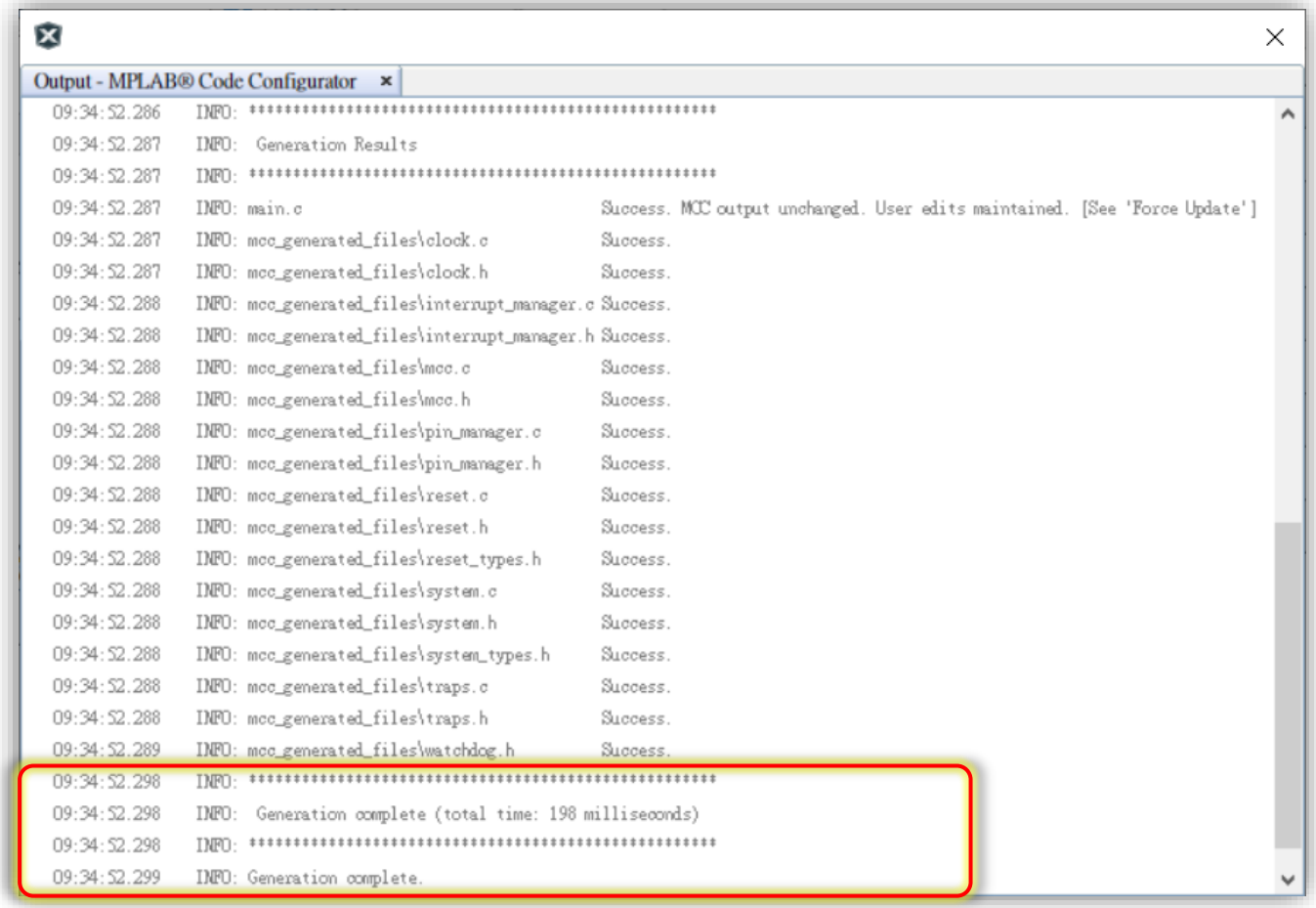
MICROCHIP

Lab6 SCCP Timer Interrupt

Step 8

- Click "Generate" Button to generate your code.





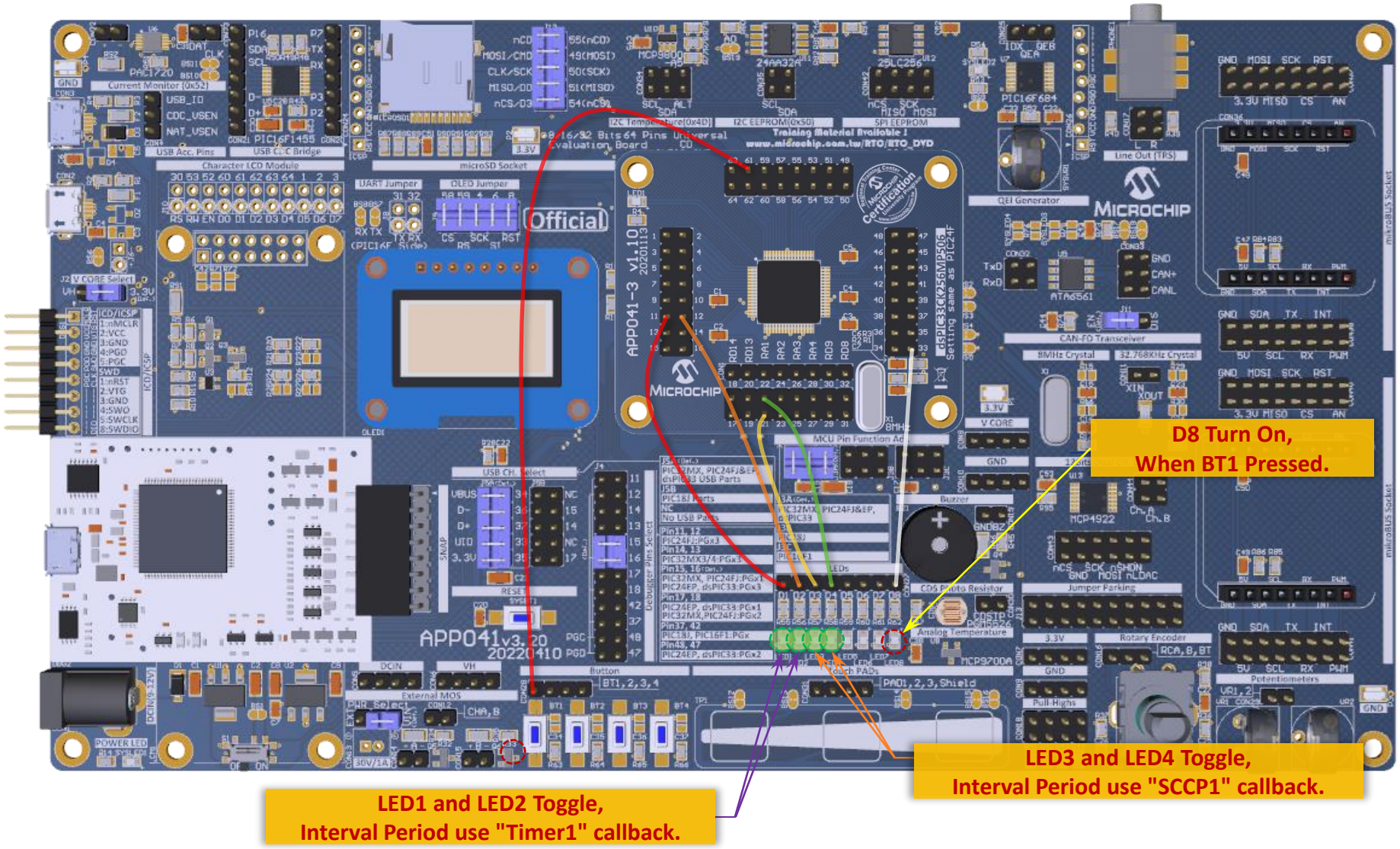
Lab6 SCCP Timer Interrupt

Code Example

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/pin_manager.h"
#include "mcc_generated_files/sccp1_tmr.h"

uint32_t  i = 0;

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

    while (1)
    {
        // Add your application code
        …
    }
    return 1;
}

void TMR1_CallBack(void)
{
    D1_Toggle();
    D2_Toggle();
}

void SCCP1_TMR_PrimaryTimerCallBack (void)
{
    D3_Toggle();
    D4_Toggle();
}
```

Lab6 SCCP Timer Interrupt

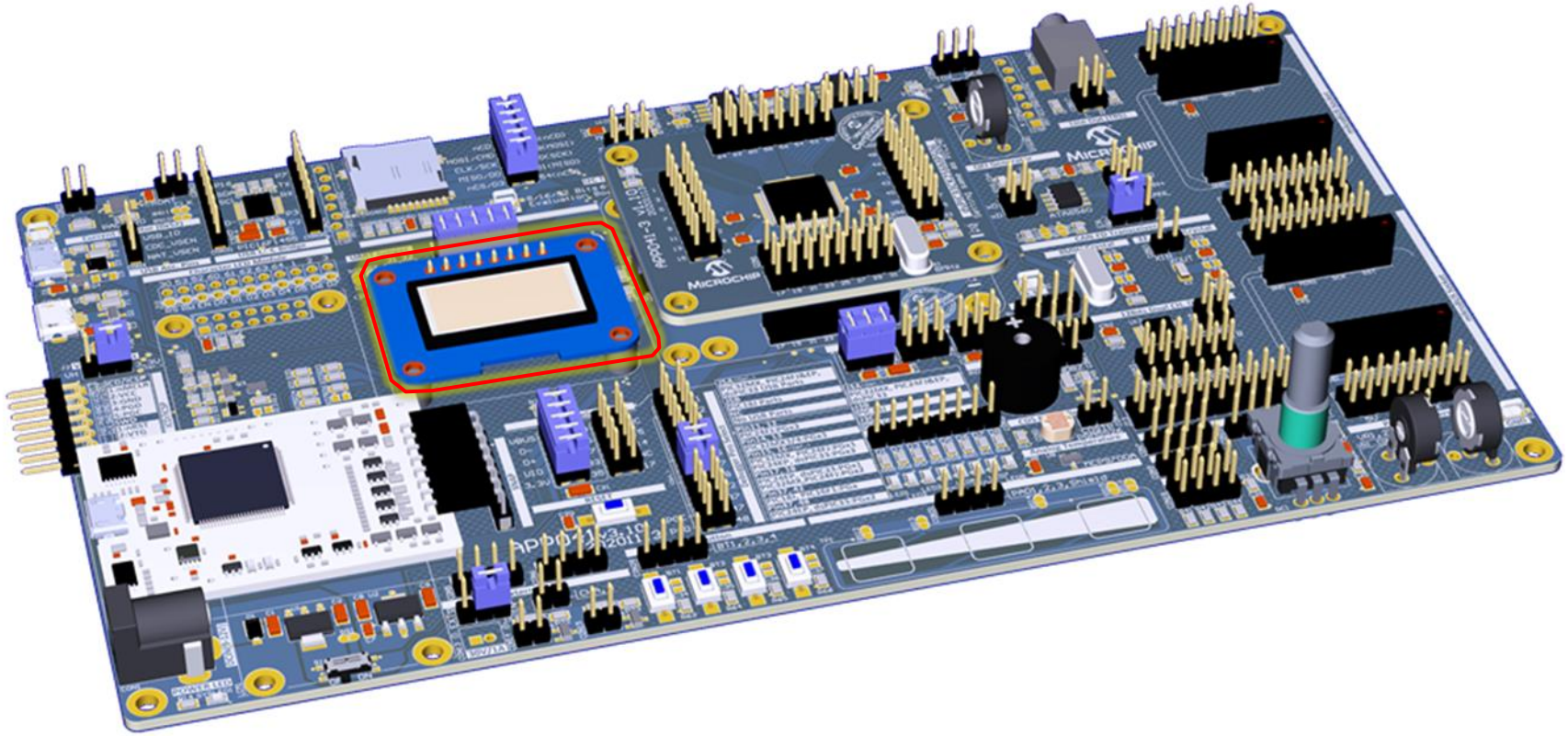
Result



OLED Application (PPS & SPI)

OLED Graphic Display

- OLED (Organic Light-Emitting Diode) with controller
  - 128 x 64 Dot matrix, Monochrome, 3 Wire SPI with 3 External control pins.
- Controller : SSD1306
  - 128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller and SPI communication interface.



MICROCHIP

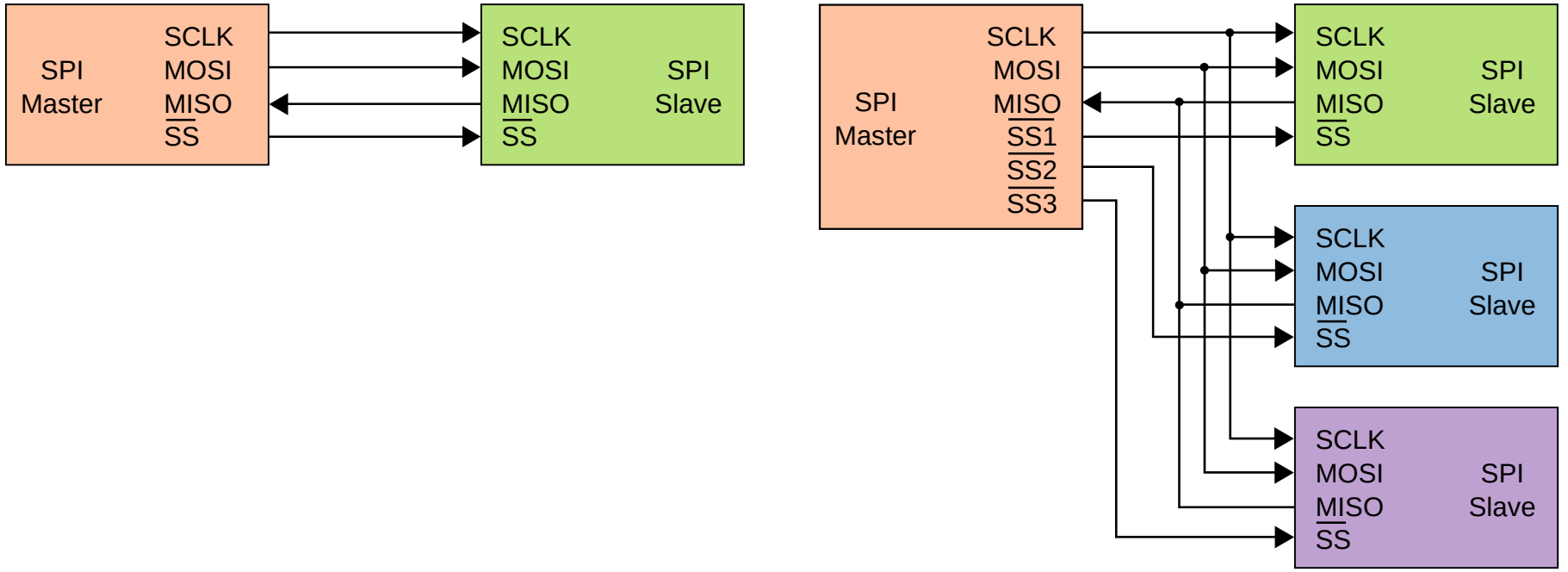
Recourses Requirement for OLED

- **Peripherals : SPI**
  - **3 Wire SPI, Mode 0, SCLK = 8 MHz**
    - Chip Select
    - SCK
    - SDO(MOSI)
- **Peripherals : GPIO**
  - **CS : Chip Select (Enable Chip & Initial Communication)**
    - High : Disable.
    - Low : Enable.
  - **RS : Data / Command select**
    - High : Data.
    - Low : Command.
  - **RESET : Module Reset**
    - High : Normal.
    - Low : Hold Reset.
- **Libraries: Delay**
  - **Timing Control (Block)**

MICROCHIP

What's SPI

- **SPI, Serial Peripheral Interface**
- **It's a serial communication interface.**
  Synchronous, inside-board level, short distance communication.



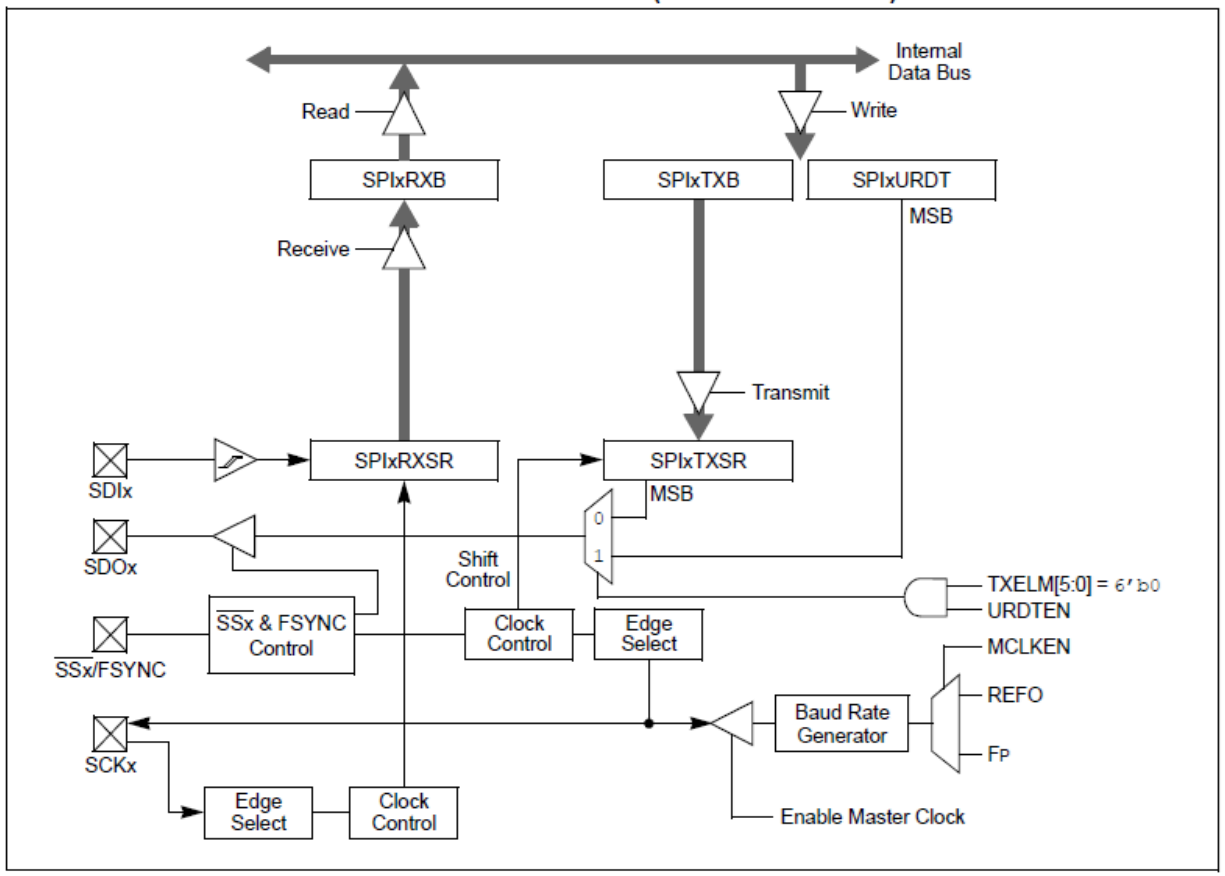
https://zh.wikipedia.org/wiki/%E5%BA%8F%E5%88%97%E5%91%A8%E9%82%8A%E4%BB%8B%E9%9D%A2

MICROCHIP

dsPIC33's SPI Module

- **dsPIC33CK256MP506 Provide 3 SPI module, support limited pin remapping.**
- **Pin Definition :**
  - **SDO (MOSI) :**
    - **Data Out, dsPIC33 to Device.**
  - **SDI (MISO) :**
    - **Data In, Device to dsPIC33.**
  - **SCK (SCLK) :**
    - **Clock Out, dsPIC33 to Device.**
  - **SS (FSYNC) :**
    - **Active-Low Slave Select or Frame Synchronization I/O Pulse**

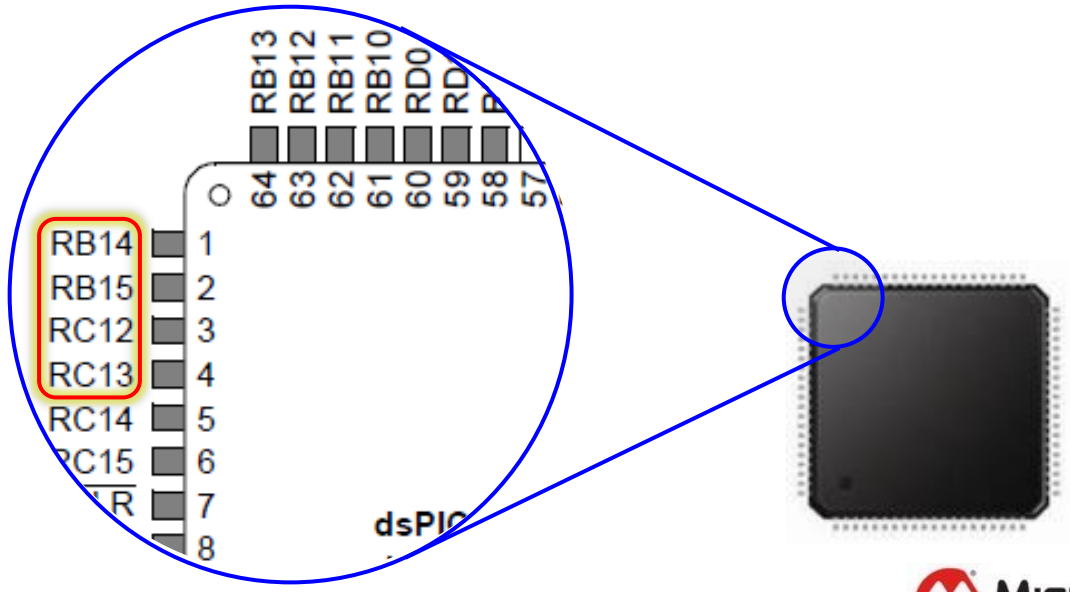
FIGURE 17-1: SPIx MODULE BLOCK DIAGRAM (STANDARD MODE)



What's PPS

- PPS, Peripheral Pin Select
- Provides a flexible peripheral pins remapping mechanism users can better tailor the microcontroller to their entire application.
- PPS pin definition
  - RP*n* : Input & Output
  - RPI*n* : Input Only
- The peripherals managed by the peripheral pin select are all digital only peripherals.

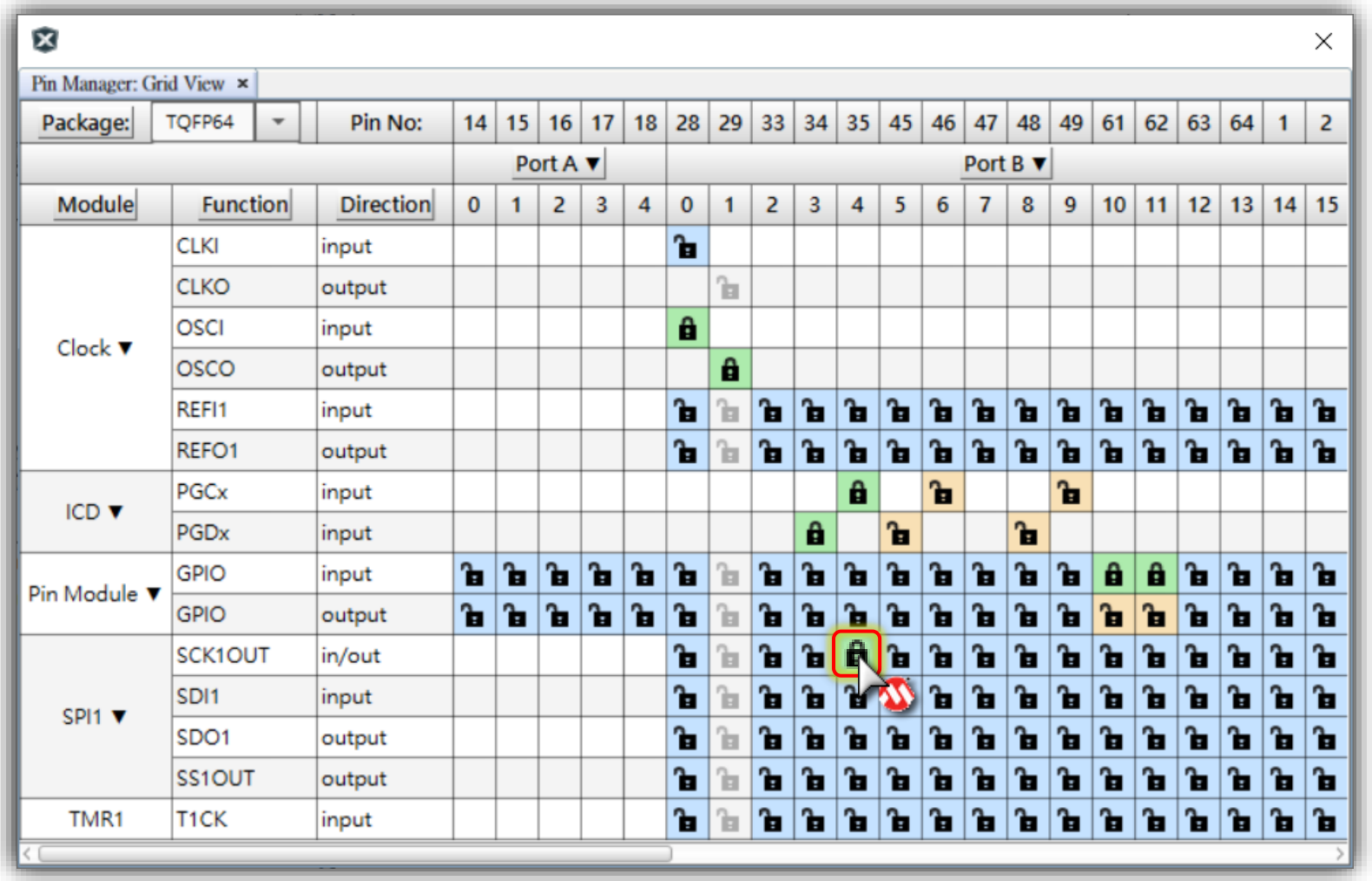
Pin #	Function
1	**RP46**/PWM1H/PMD5/RB14
2	**RP47**/PWM1L/PMD6/RB15
3	**RP60**/PWM8H/PMD7/RC12
4	**RP61**/PWM8L/PMA5/RC13



MICROCHIP

PPS Assign Pin at MCC

- **Just one Click on that pin you want to config in :**
  - **Pin Manager : Gird View**



OLED Functions of MCC

- **We Provide below functions for OLED:**
  "myOLED_ssd1306.c", "myOLED_ssd1306.h"
  - `void myOLED_Init(void);` /* Initial OLED Module */
  - `void myOLED_ClearScreen(void);` /* Clear Screen */
  - `void myOLED_DrawBitmap(uint8_t column, uint8_t page, uint8_t width, uint8_t height, const uint8_t *byte);` /* Draw Raw Bitmap Picture */
  - `void myOLED_DrawChar(uint8_t column, uint8_t page, uint8_t byte);` /* Draw Single Character */
  - `void myOLED_DrawStr(uint8_t column, uint8_t page, const uint8_t *byte);` /* Draw String */

MICROCHIP

Lab Preparation

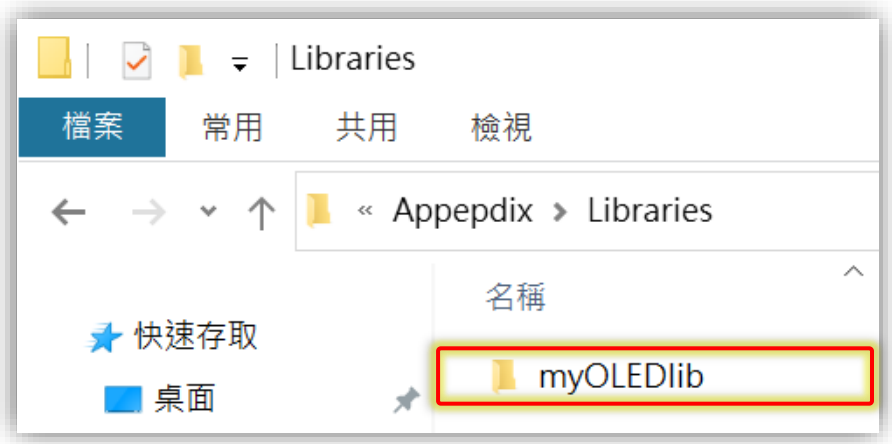
Add OLED Libraries

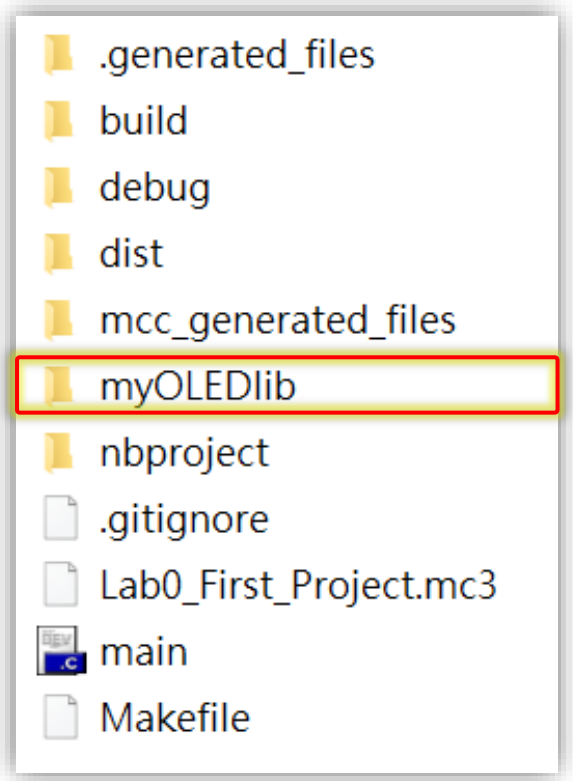
MICROCHIP

Lab Preparation

Step 1

- **File Copy paste on folder.**
  - **(\Appepdix\Libraries\myOLEDlib** to **\Labn_xxx.X)**

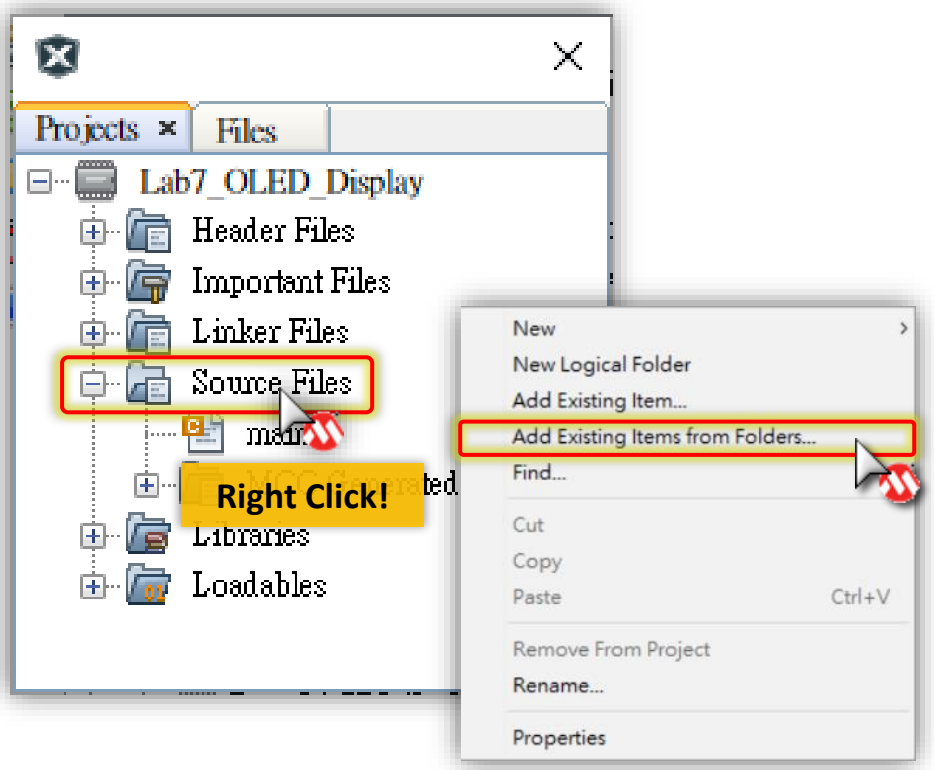


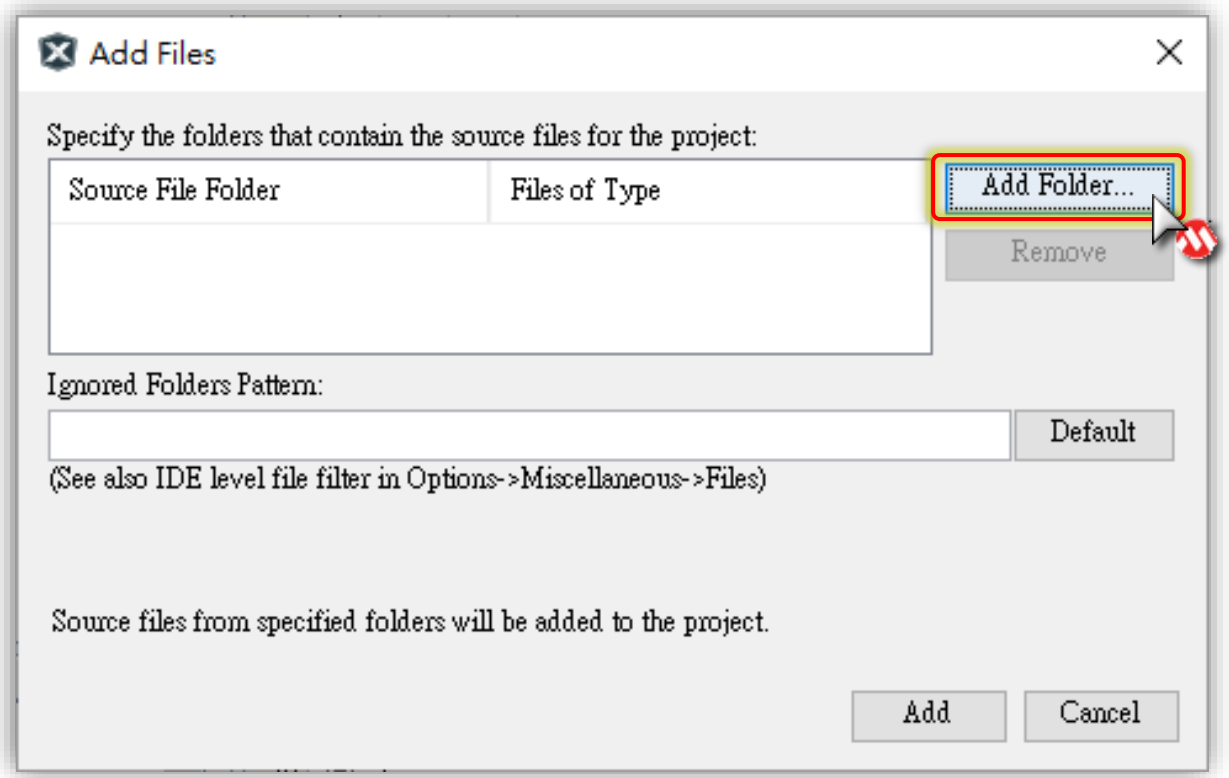


Lab Preparation

Step 2

- **Add Existing Folder to Project**
  - Right click on "**Source Files**".
  - Select "**Add Existing Items from Folders**" at Projects Windows.
  - Select "**Add Folder…**".

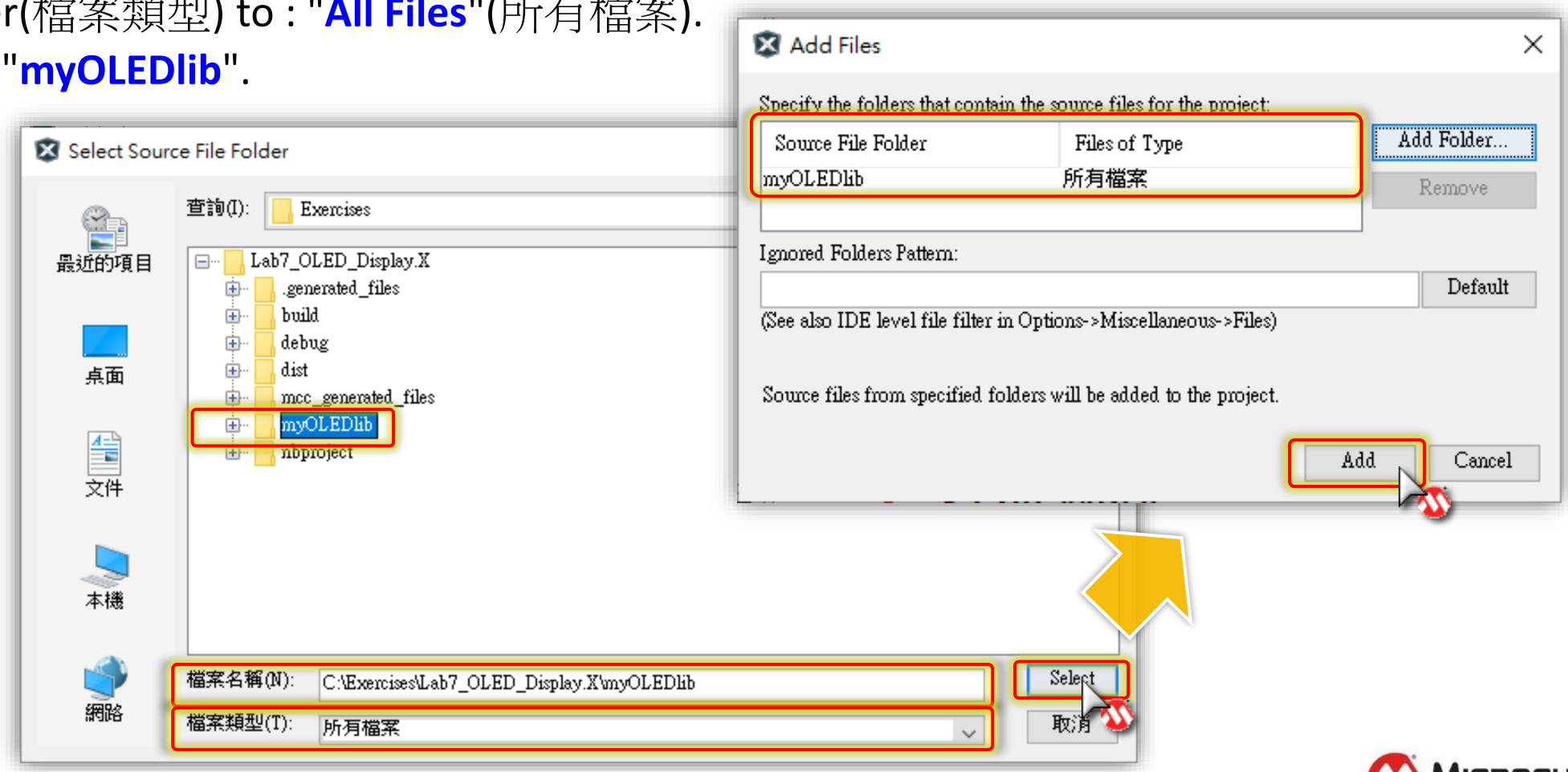




Lab Preparation

Step 3

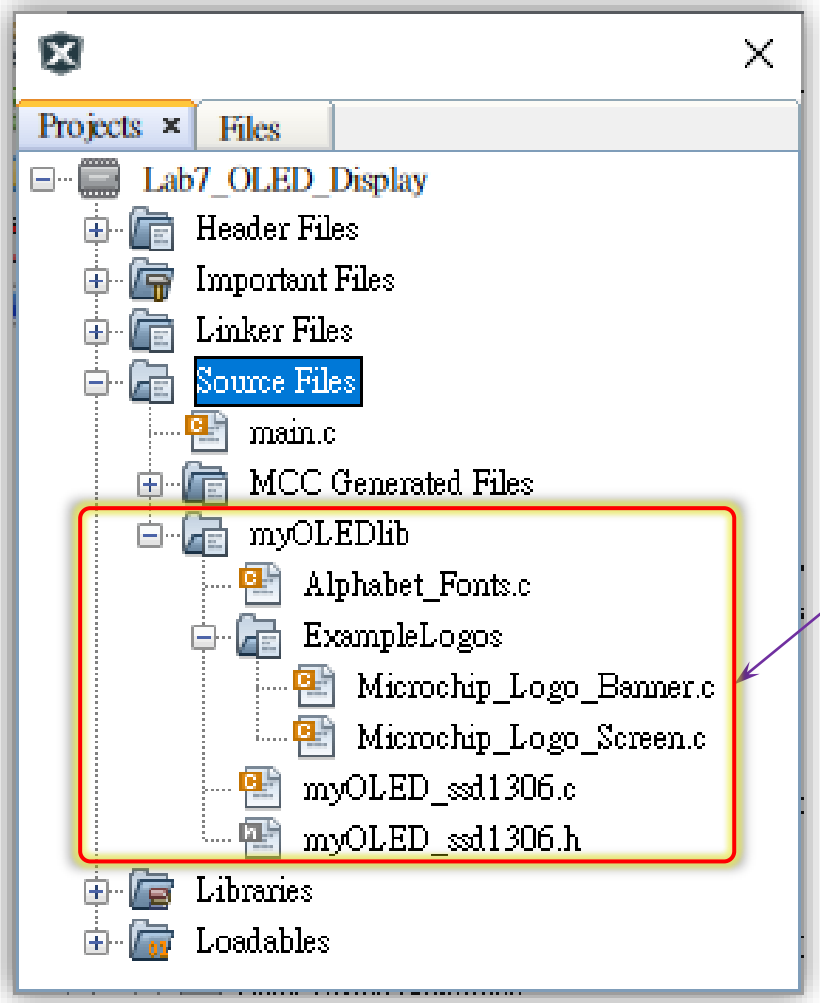
- **Select myOLEDlib folder to Add**
  - Select File Filter(檔案類型) to : "**All Files**"(所有檔案).
  - Click on folder "**myOLEDlib**".
  - Click "**Select**".
  - Click "**Add**"



Lab Preparation

Step 4

- **Success to add "myOLEDlib" libraries folder in your Project.**



Libraries include :
OLED control functions
Raw Data of Fonts
Raw Data of Demo logos

MICROCHIP

What’s Delay

- MCC provide various software function and protocol stack libraries.

  e.g., Software UART, I2C, Delay, LIN Master/Salve etc..

- Delay libraries is an easy, simple and block functions.

  Use real instruction cycle (Tcy) to generate time latency to ensure accurate delay.

Lab7 OLED Display

- **Base on current project or Open .\Exams\Lab7_xxx.X to do exercises**
- **Try to add OLED Library and related resources to project then initial OLED display.**
- **Try to display some information and bitmap logo on OLED display.**

Let’s go!

MICROCHIP

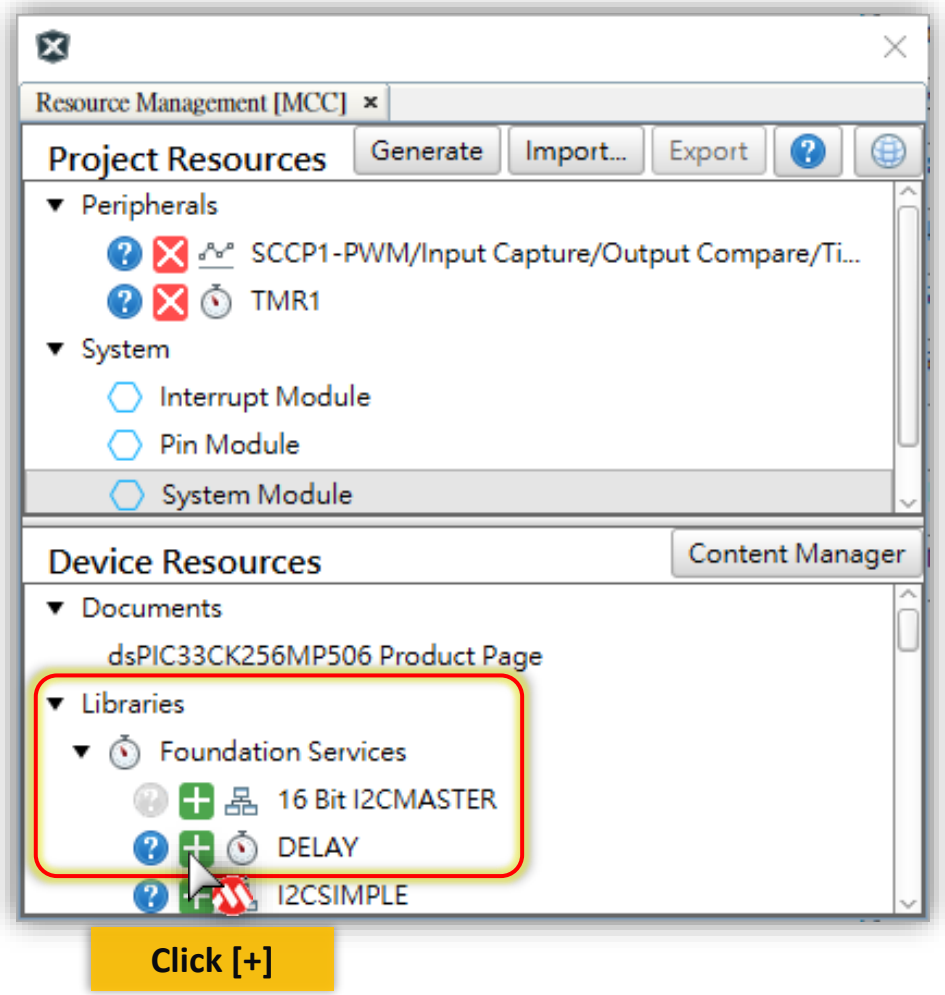
Lab7 OLED Display

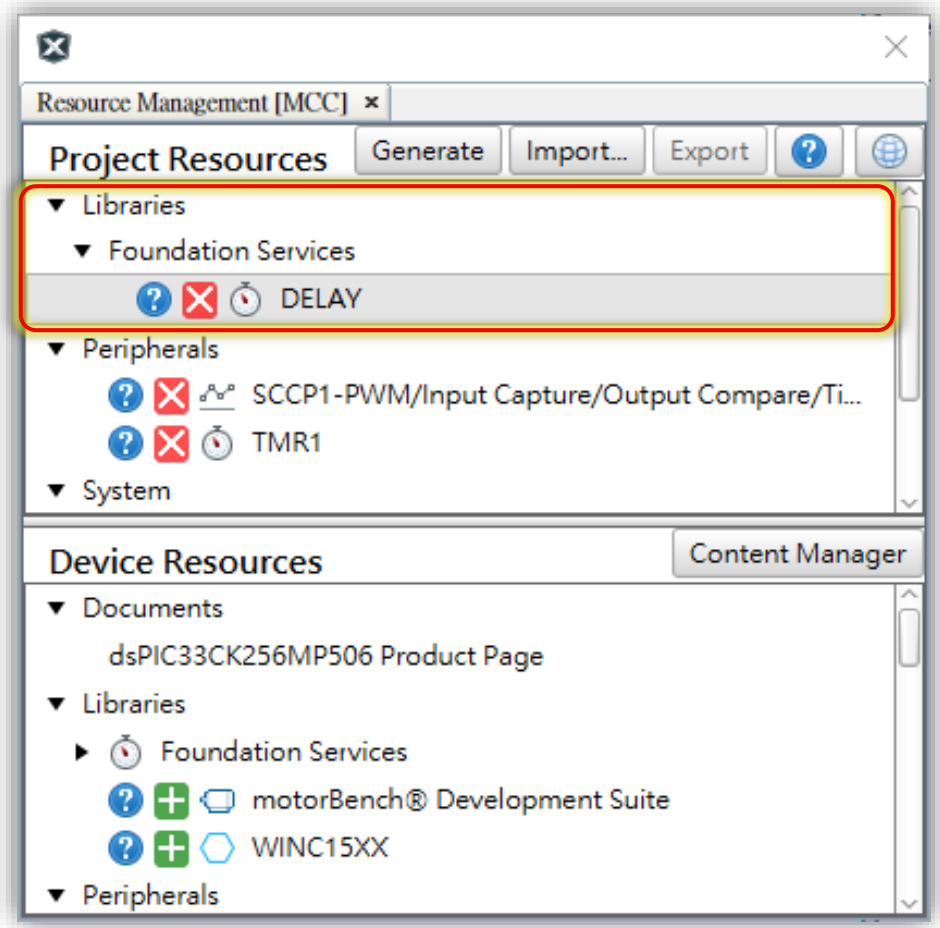
Connection

Lab7 OLED Display

Step 1

- **Click [+] to add DELAY Module.**

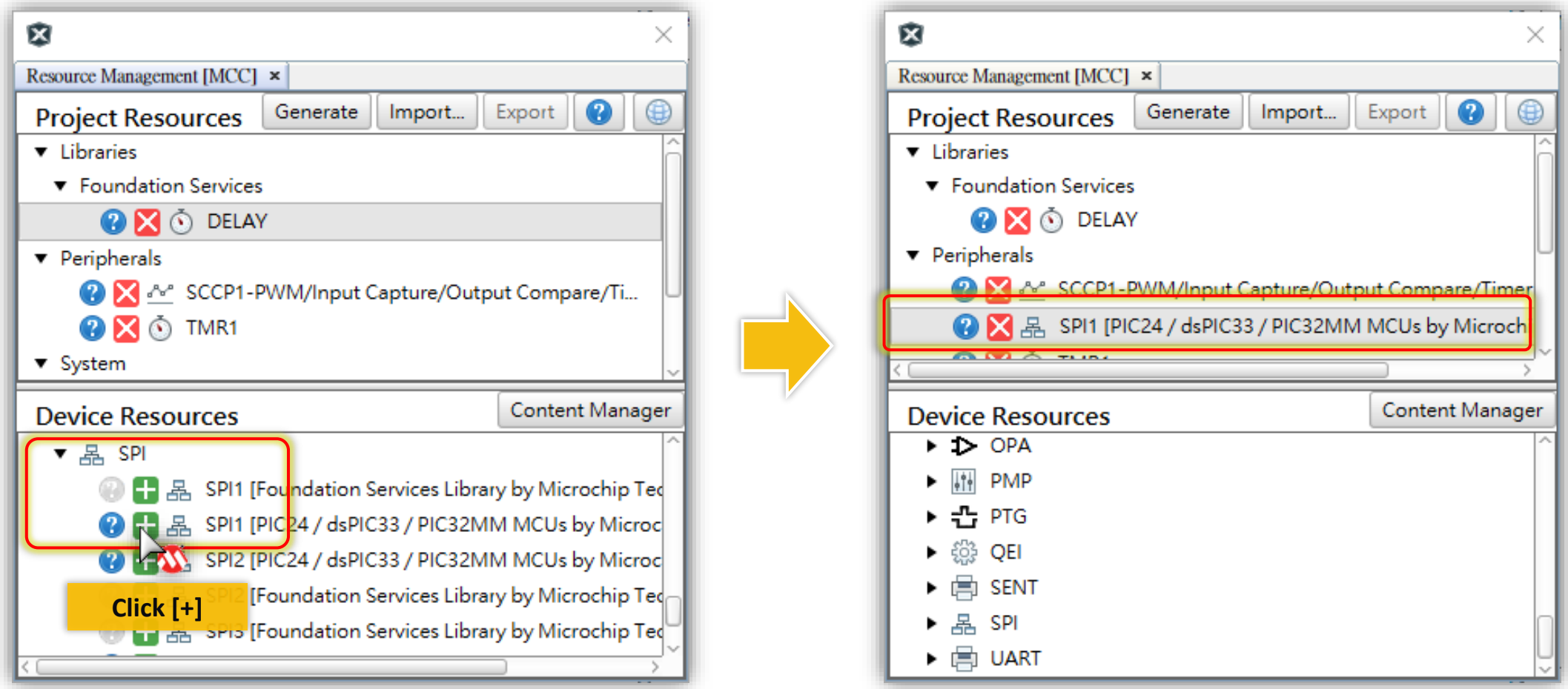




Lab7 OLED Display

Step 2

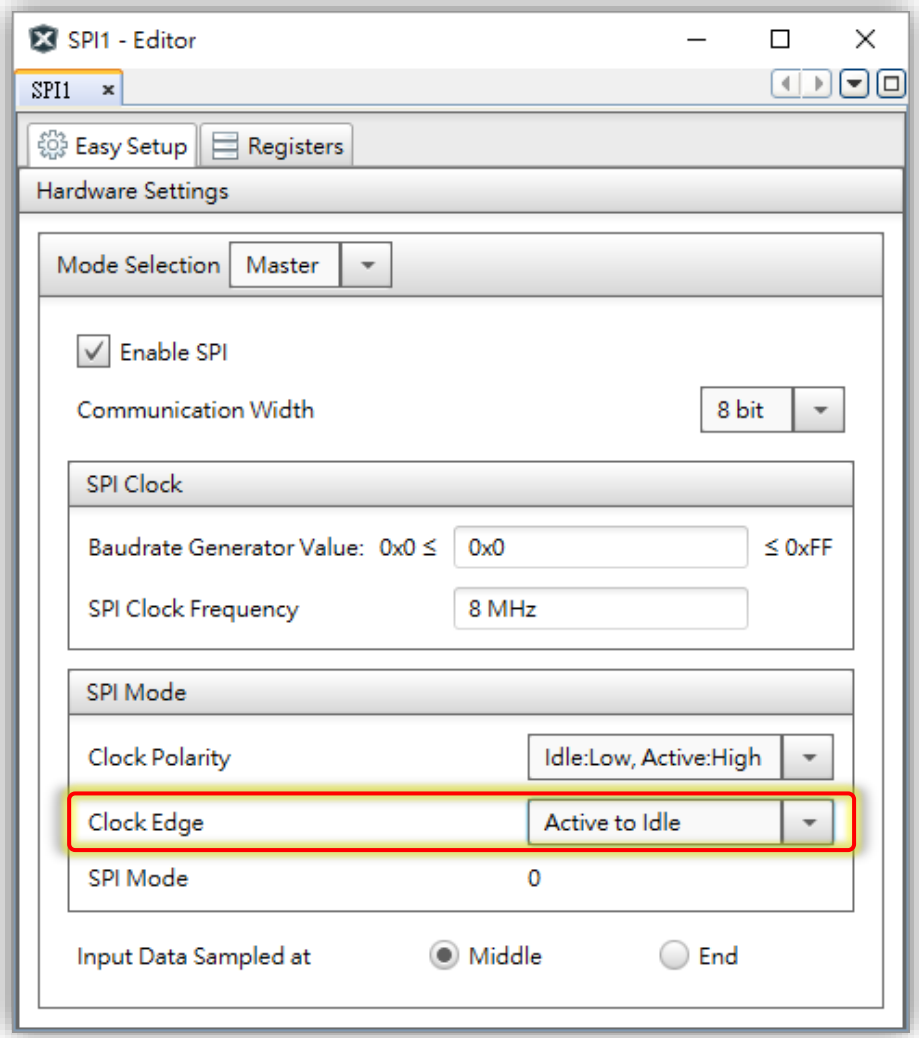
- **Click [+] to add SPI > SPI1[PIC24 / dsPIC33 / PIC32MM MCUs by Microchip…] Module.**



Lab7 OLED Display

Step 3

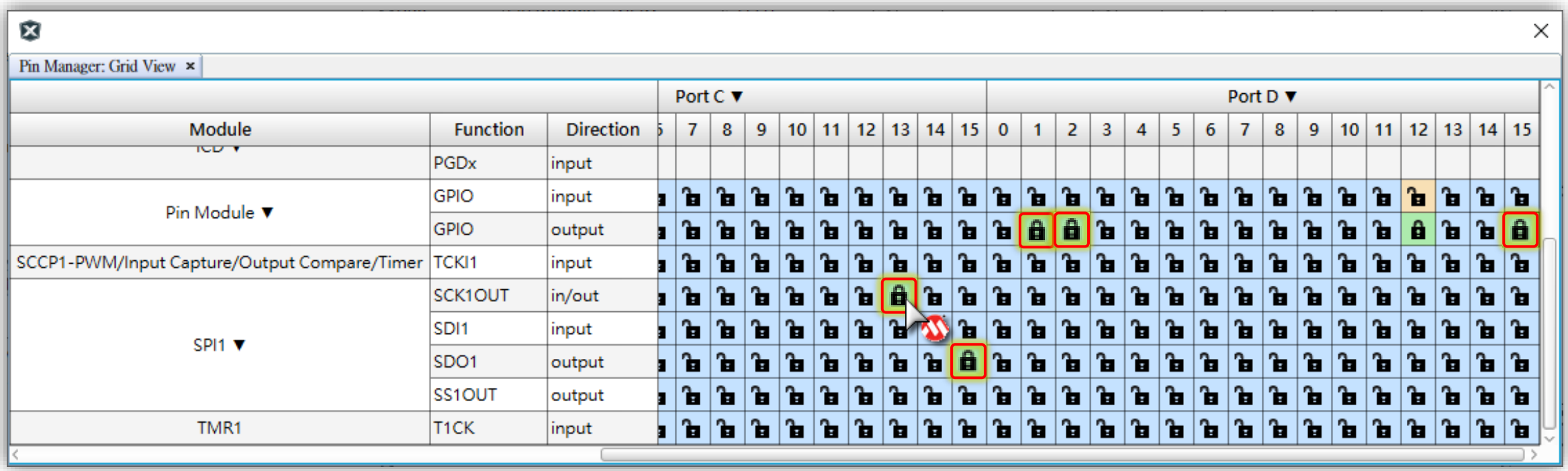
- Set Clock Edge : Idle to Active > Active to Idle.



Lab7 OLED Display

Step 4

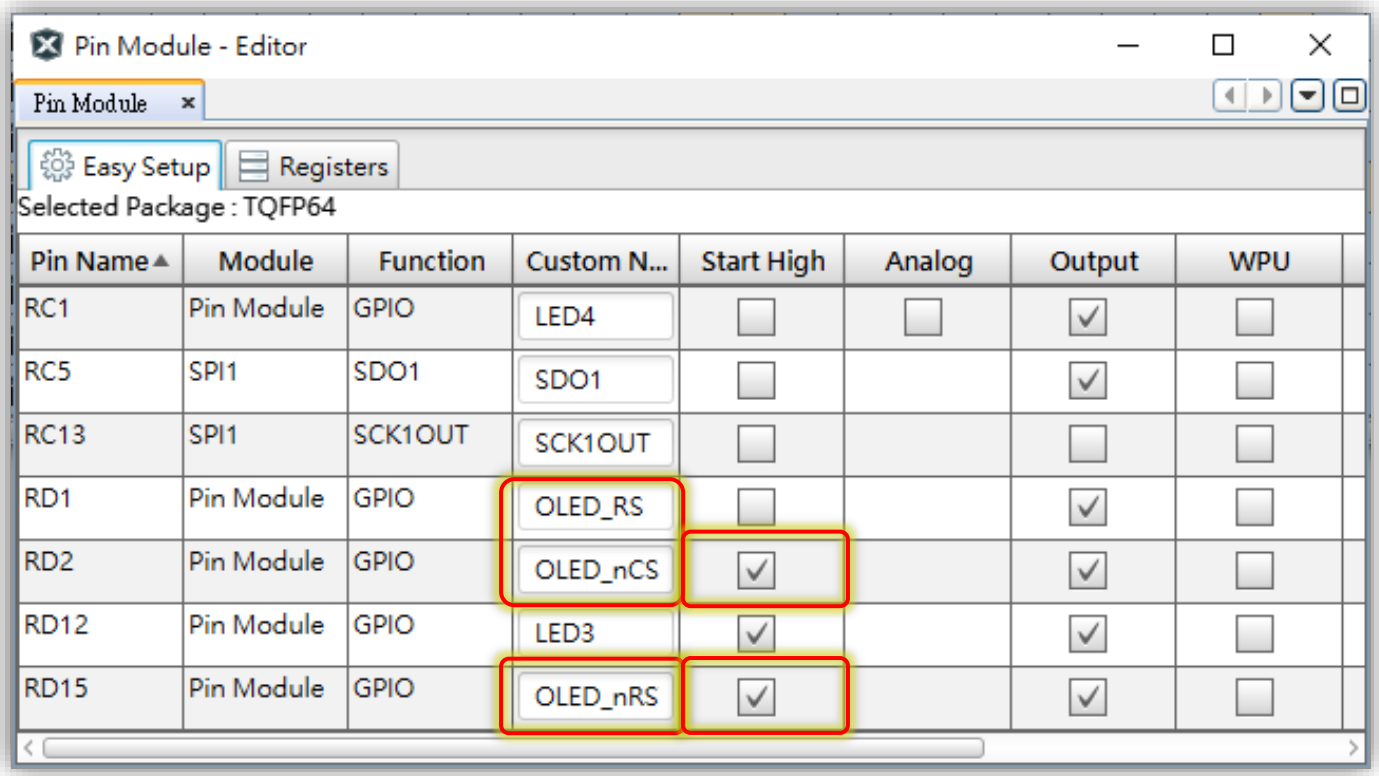
- Set PORTC : RC13 (SCK1OUT) , RC15 (SDO1) , to PPS SPI Function.
- Set PORTD : Pin1 (RD1), Pin2 (RD2), Pin15 (RD15), to digital output mode.



Lab7 OLED Display

Step 5

- **Set Alias for RD1 , RD2 and RD15.**
  - **Pin Module : RD1 , Custom Name > OLED_RS , Check Output.**
  - **Pin Module : RD2 , Custom Name > OLED_nCS , Start High , Check Output.**
  - **Pin Module : RD15 , Custom Name > OLED_nRST , Start High , Check Output.**

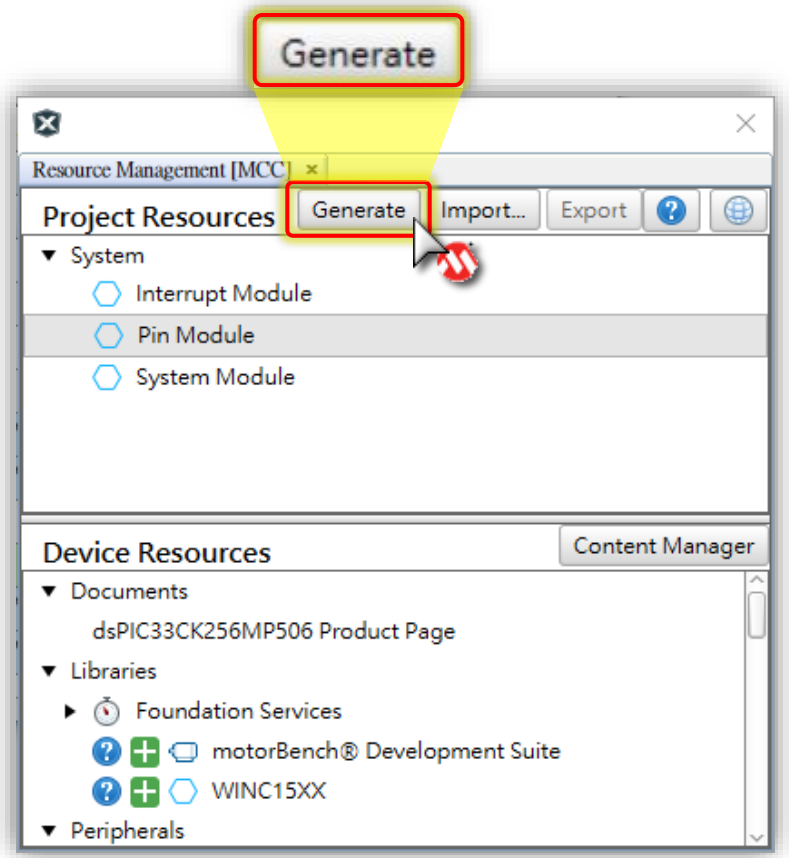


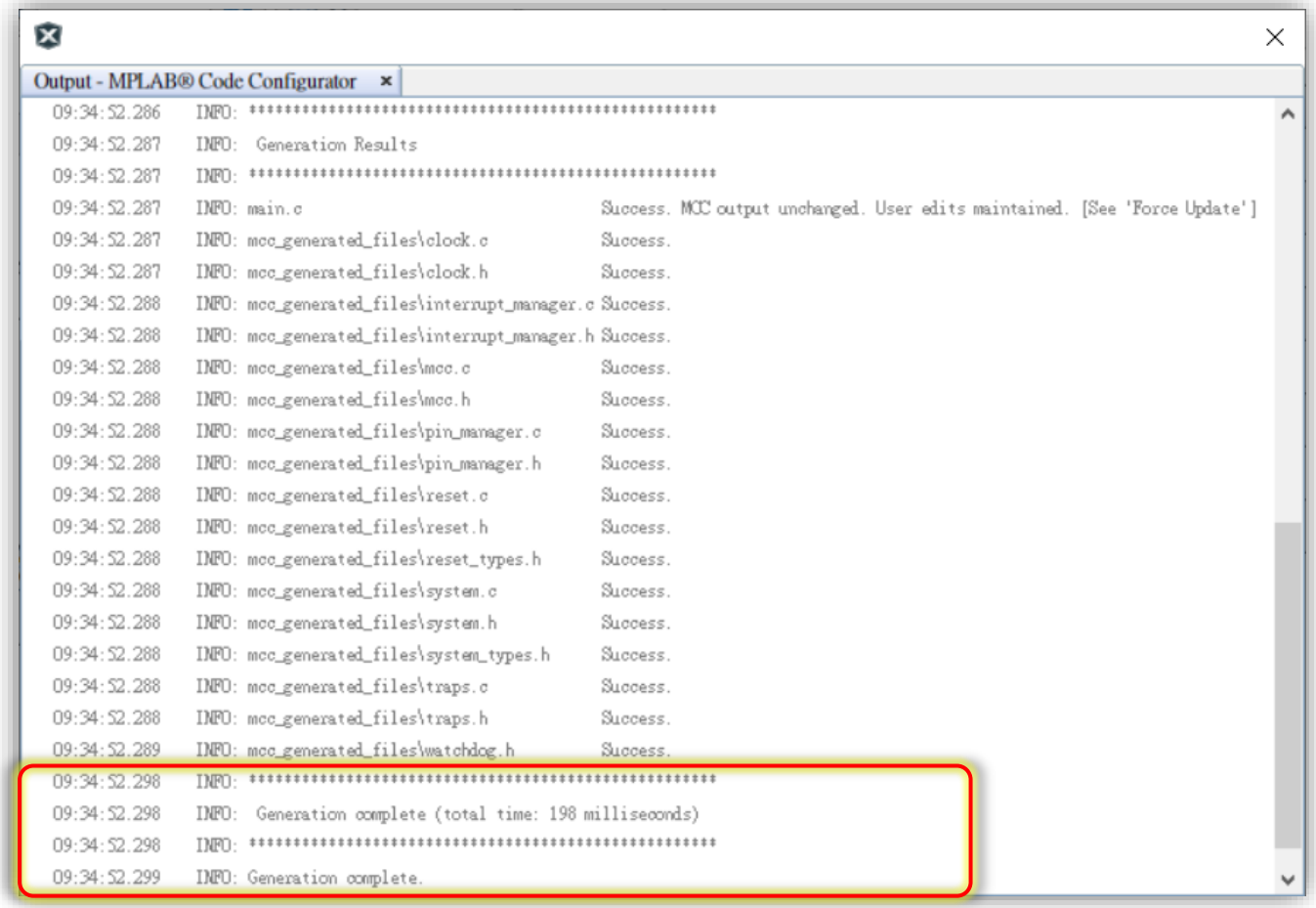
MICROCHIP

Lab7 OLED Display

Step 6

- Click "Generate" Button to generate your code.





Lab7 OLED Display

Code Example

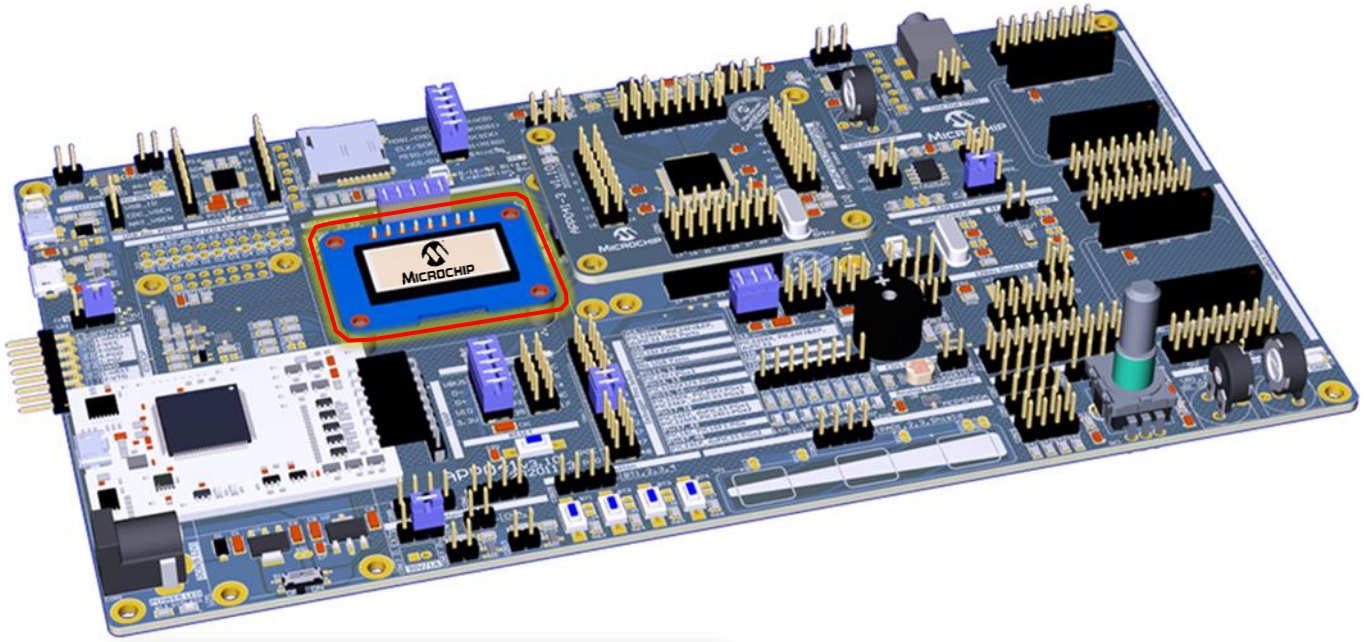
```
#include "mcc_generated_files/sccp1_tmr.h"
#include <stdio.h>
#include "mcc_generated_files/delay.h"
#include "myOLEDlib/myOLED_ssd1306.h"

uint32_t  i = 0;
extern const uint8_t Microchip_Logo_Screen[];
extern const uint8_t Microchip_Logo_Banner[];

uint8_t StringBuffer[40];

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

   while (1)
   {
      // Add your application code
      …
   }
   return 1;
}
```

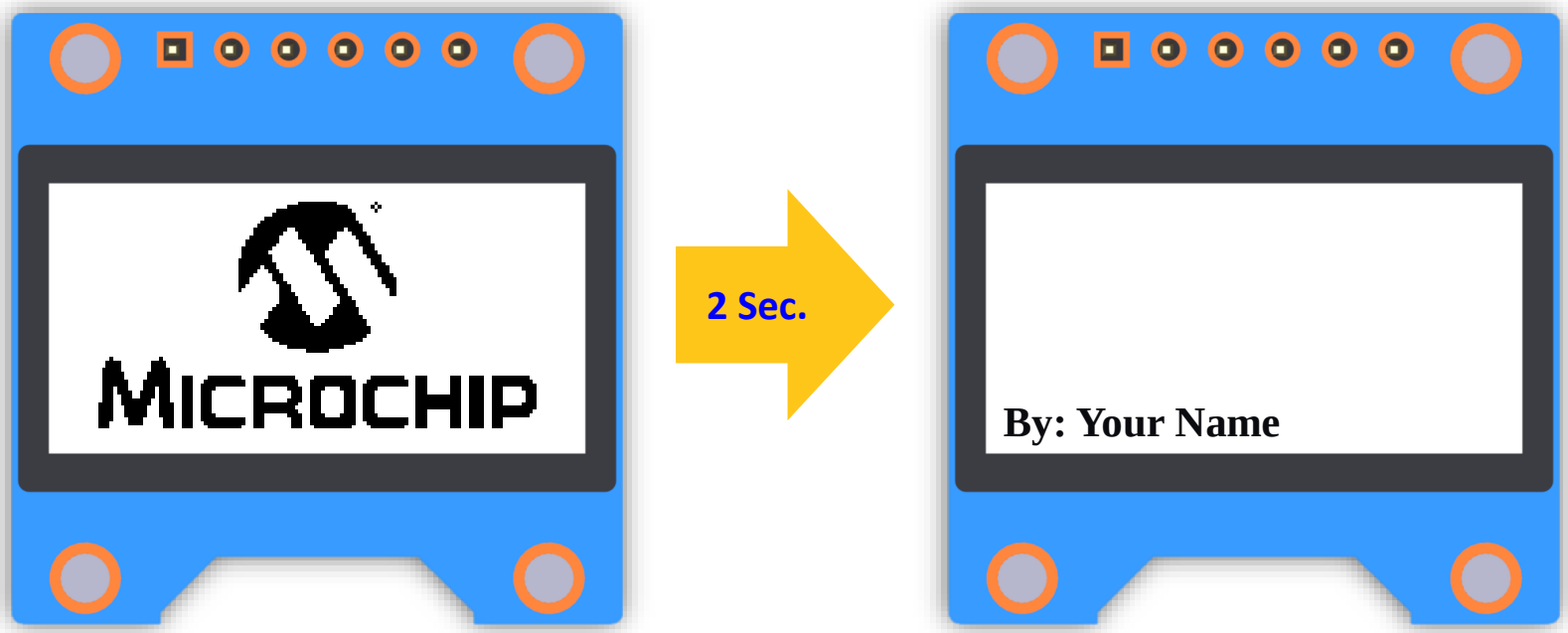
```
int main(void)
{
   // initialize the device
   SYSTEM_Initialize();

   myOLED_Init();

   myOLED_ClearScreen();
   myOLED_DrawBitmap(0, 0, 128, 64, Microchip_Logo_Screen);

   DELAY_milliseconds(2000);

   myOLED_ClearScreen();
   myOLED_DrawStr(0, 7, (const uint8_t *) "By: Your Name");

   while (1)
   {
      // Add your application code
      …
   }
   return 1;
}
```

MICROCHIP

Lab7 OLED Display

Result





MICROCHIP

Lab8 Customize Screen

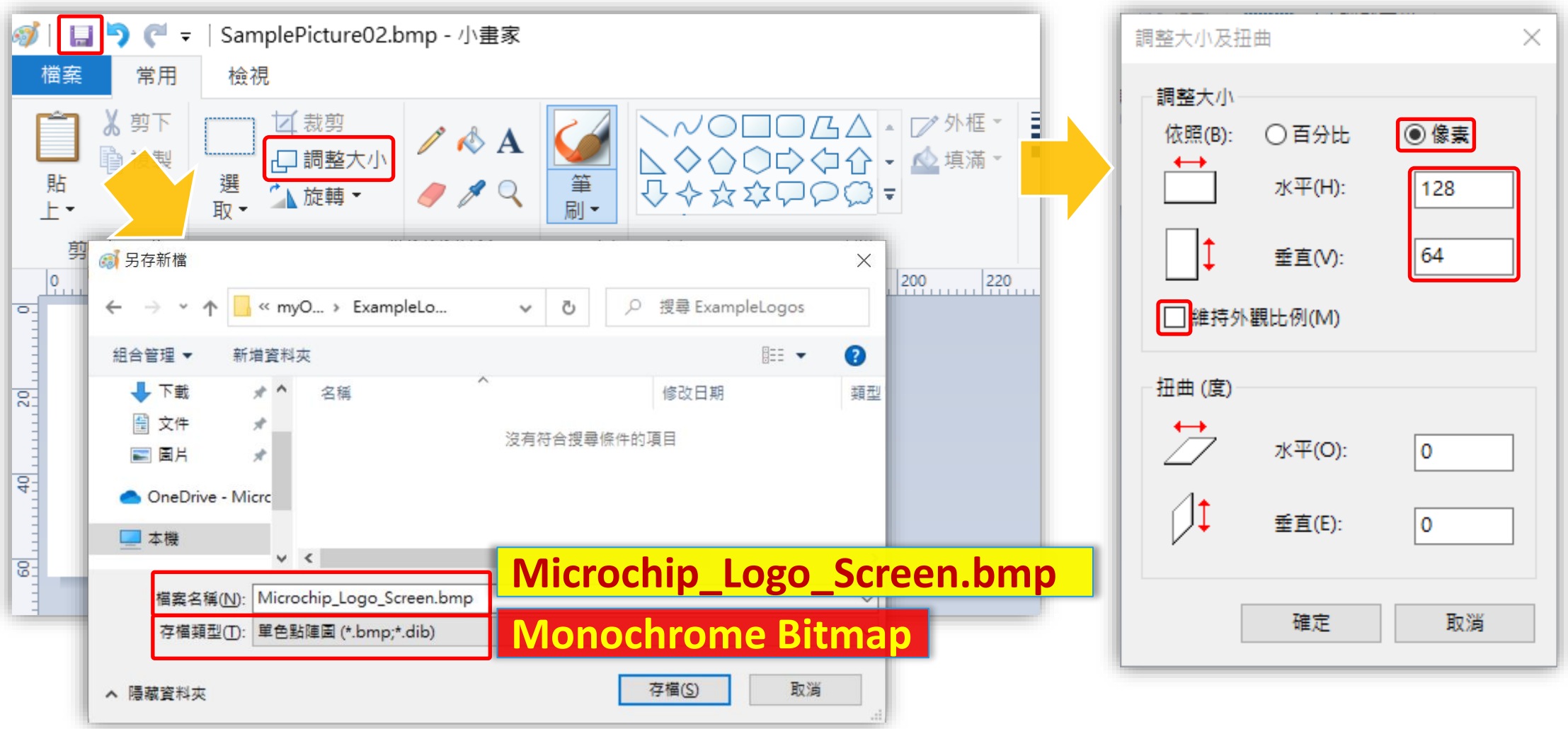
- Base on current project or Open .\Exams\Lab8_xxx.X to do exercises
- Try to change default Boot Screen to you want to.

Let’s go!

MICROCHIP

Lab8 Customize Screen

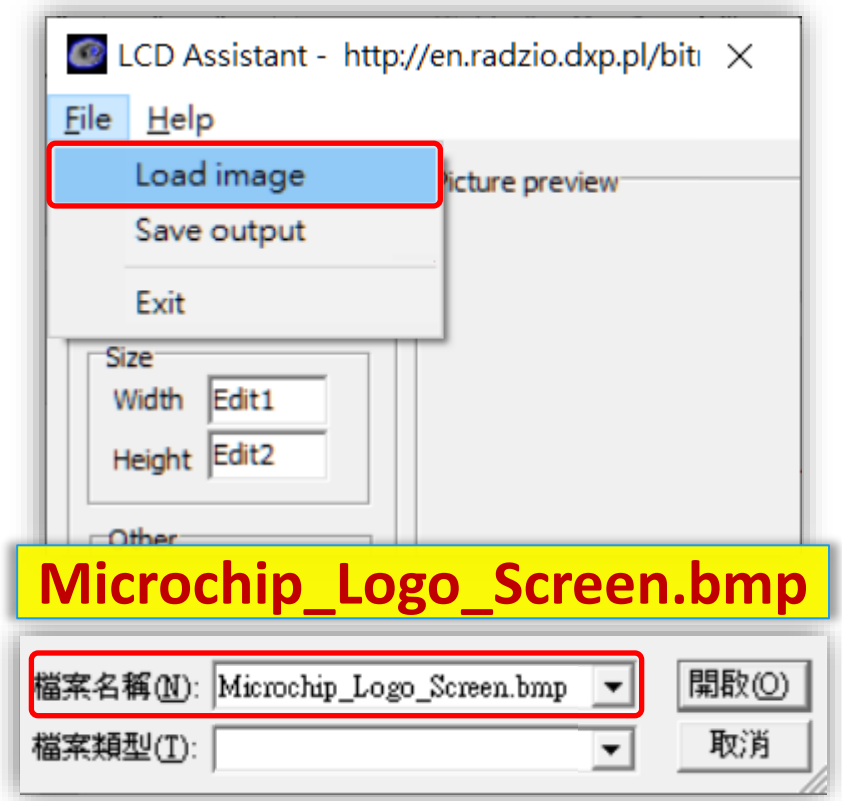
Create your own logo



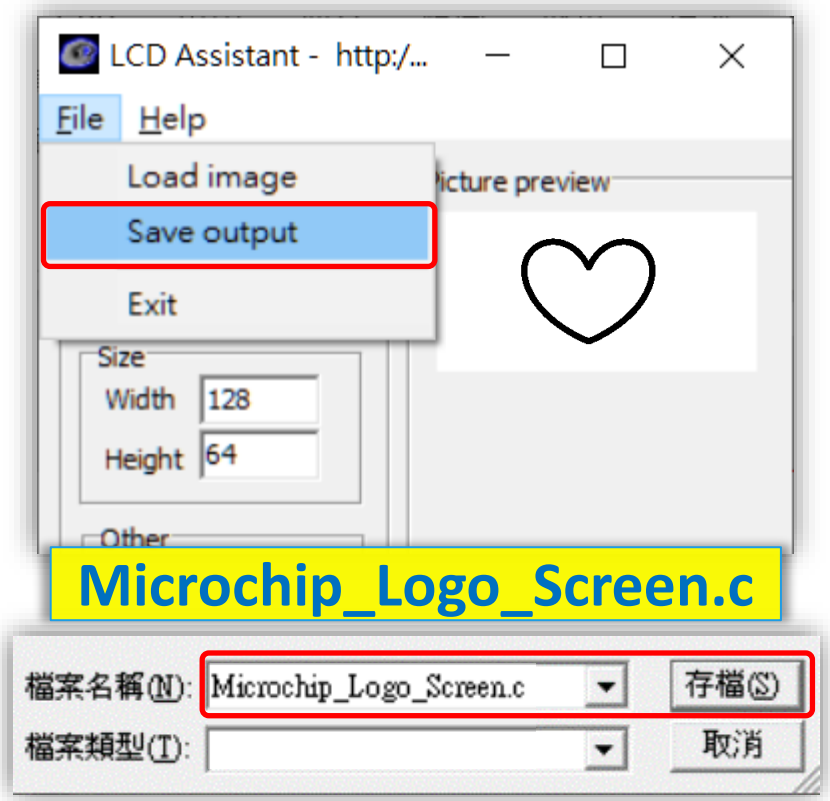
Lab8 Customize Screen

Create your own logo

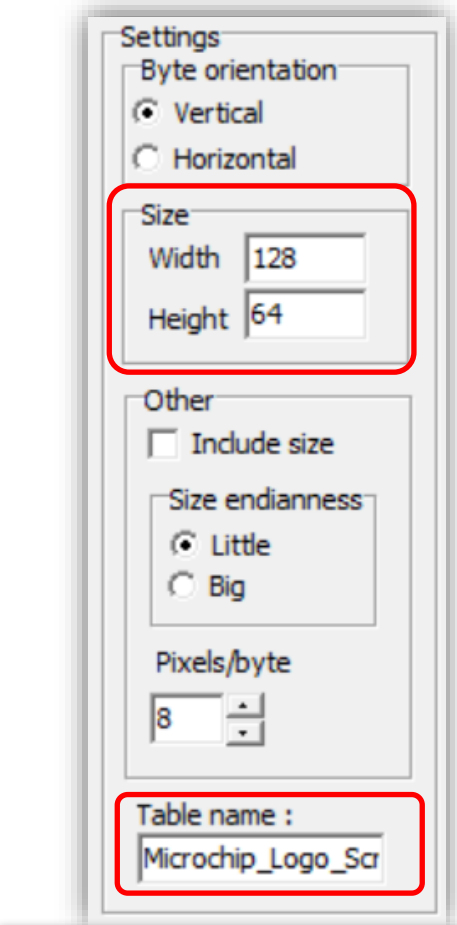
- **Execute \Tools\LCDAssistant.exe**
  Load Microchip_Logo.bmp and Save output to Microchip_Logo.h



Microchip_Logo_Screen.bmp



Microchip_Logo_Screen.c

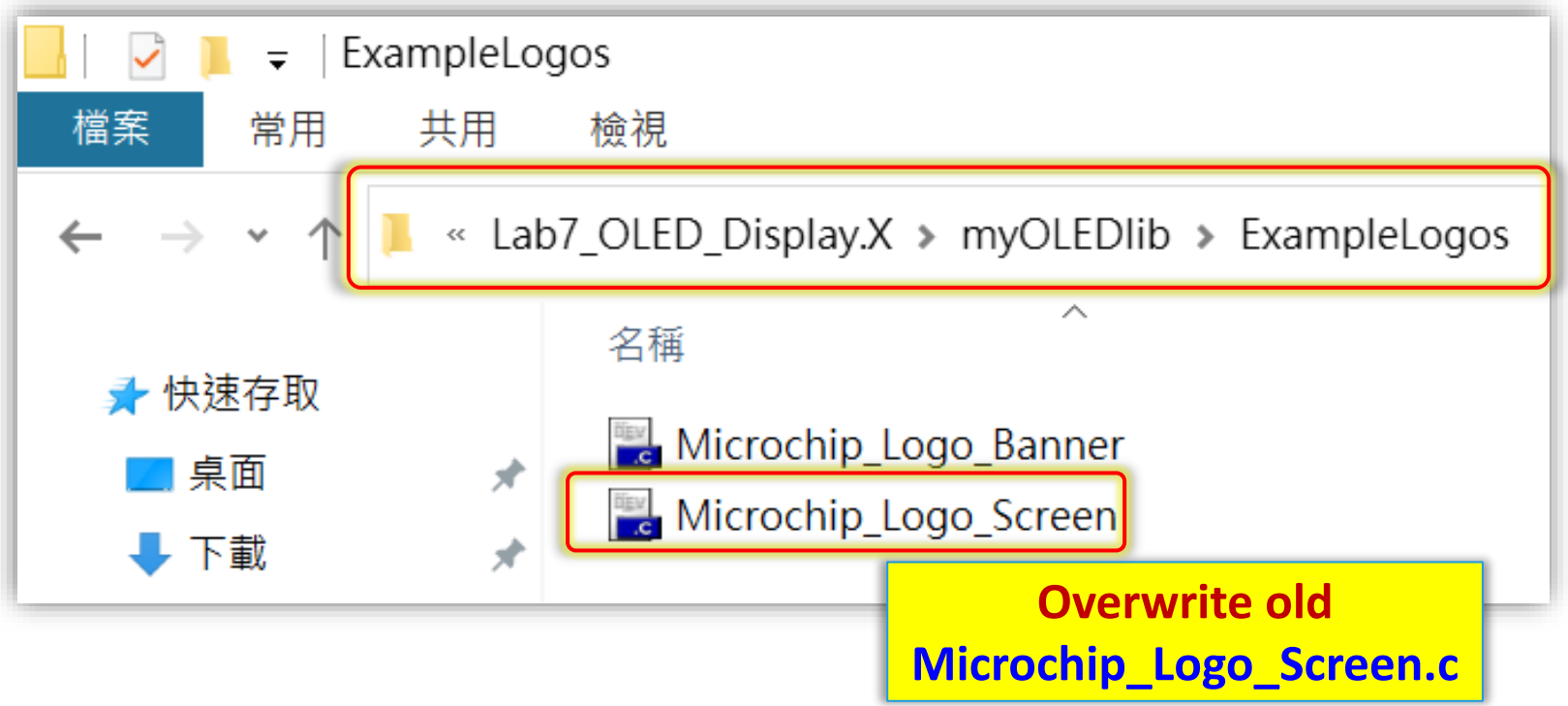


Microchip_Logo_Screen
(Bitmap Array name)

Lab8 Customize Screen

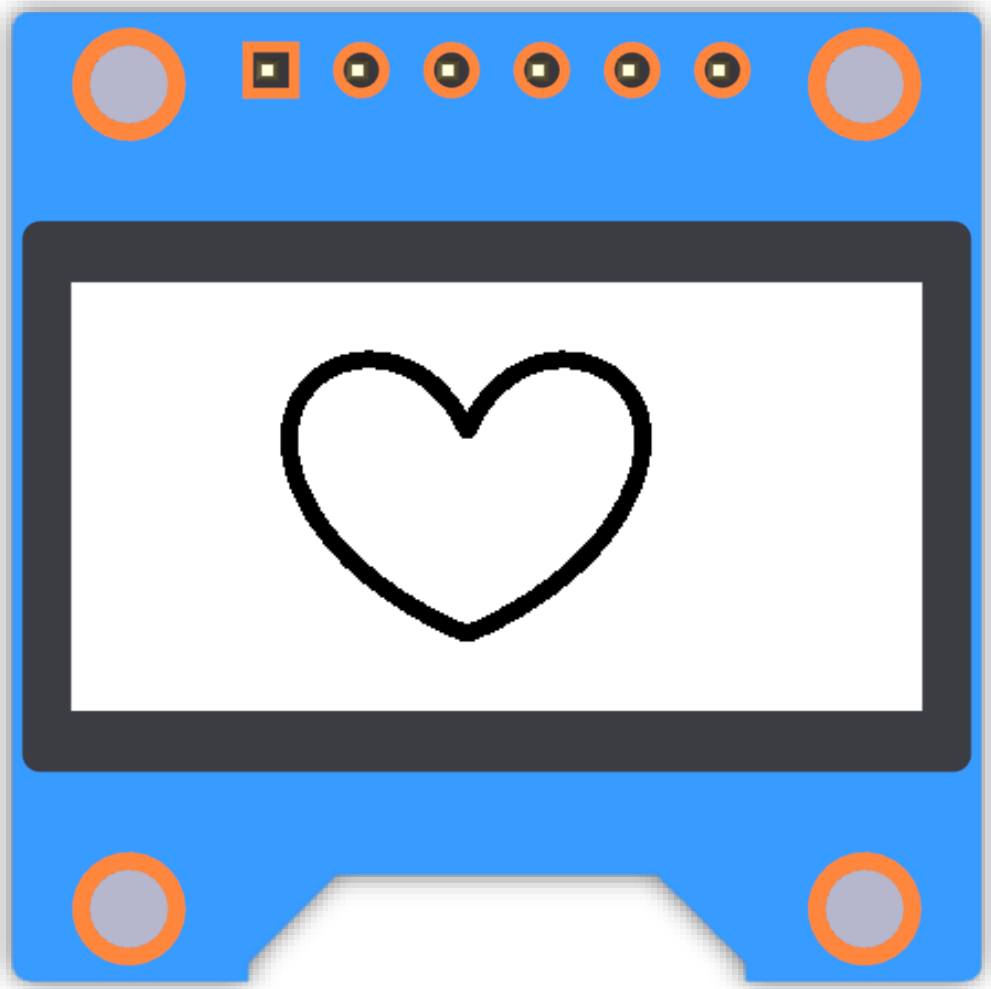
Create your own logo

- **Copy new Microchip_Logo.h to overwrite old one in folder.**
  - **\Labn_xxx.X\myOLEDlib\ExampleLogos**



Lab8 Customize Screen

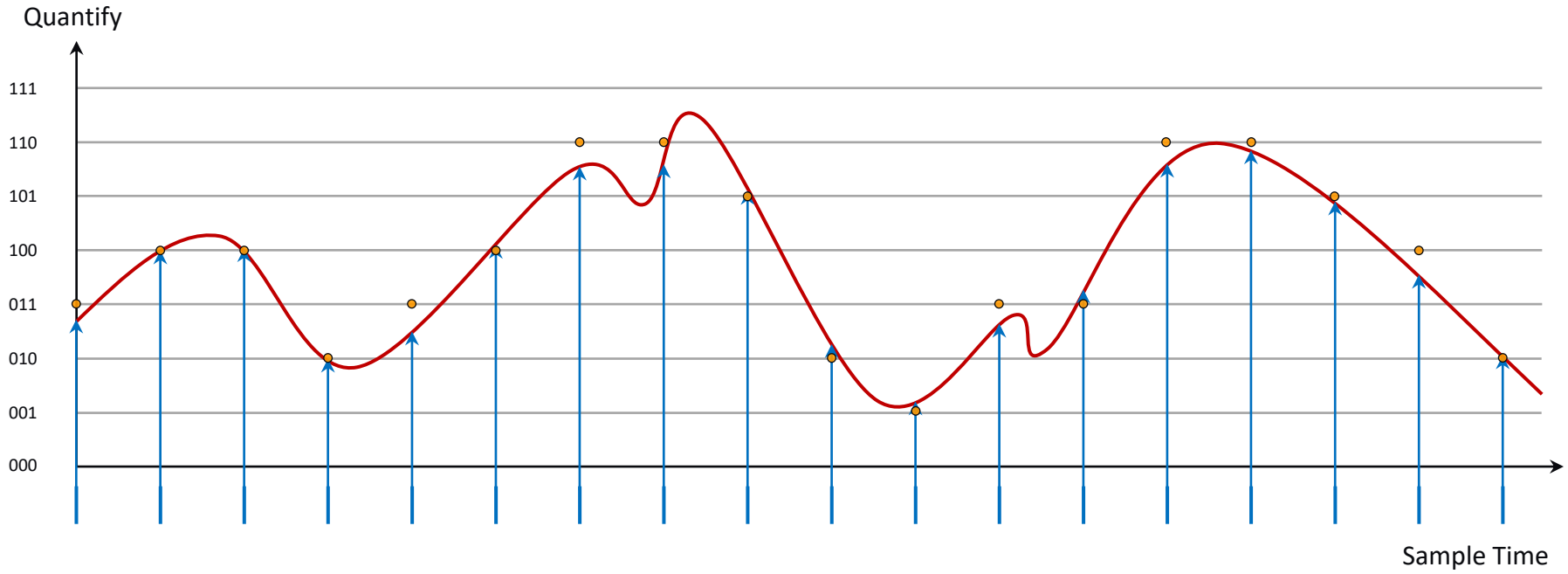
Create your own logo result



12 Bits High Speed ADC Application

What’s ADC ?

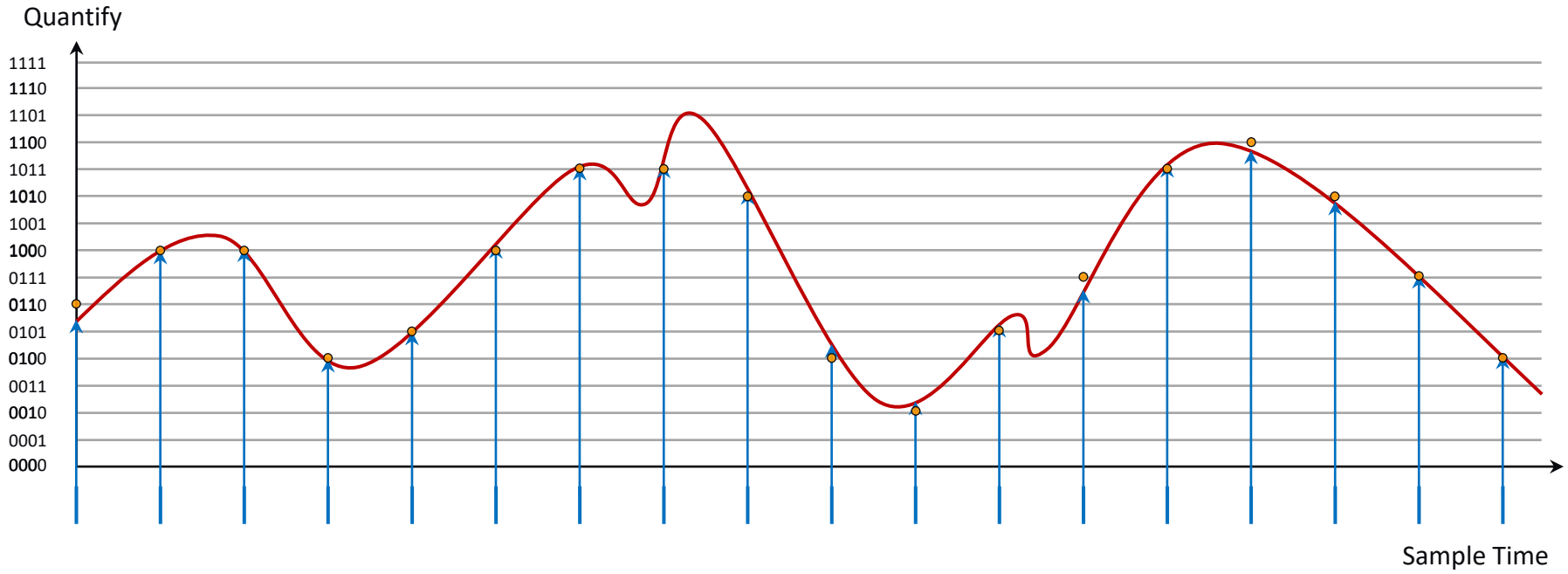
- **The Analog-to-Digital Converter(ADC) converts a signal from the time/amplitude continuous domain into.**
  a time/amplitude discrete domain.



MICROCHIP

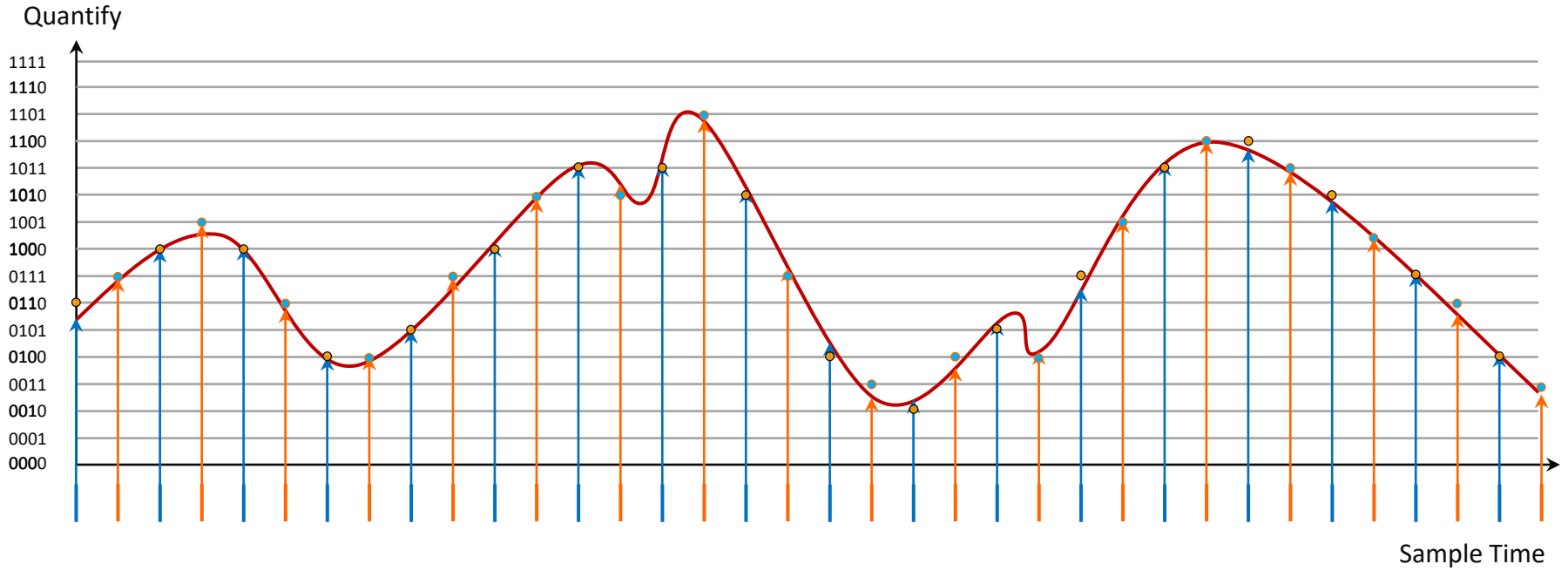
Resolution

- **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)

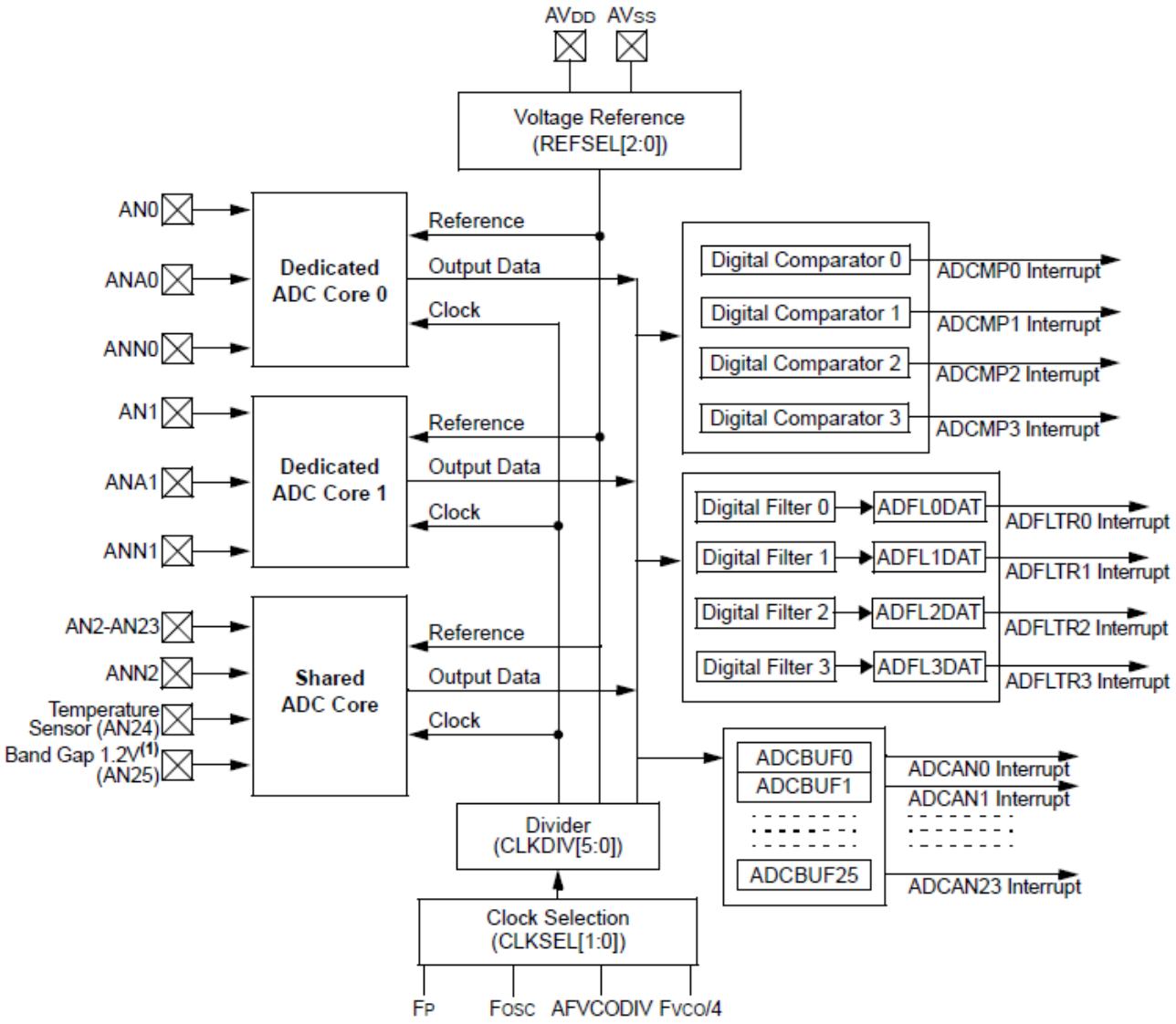


MICROCHIP

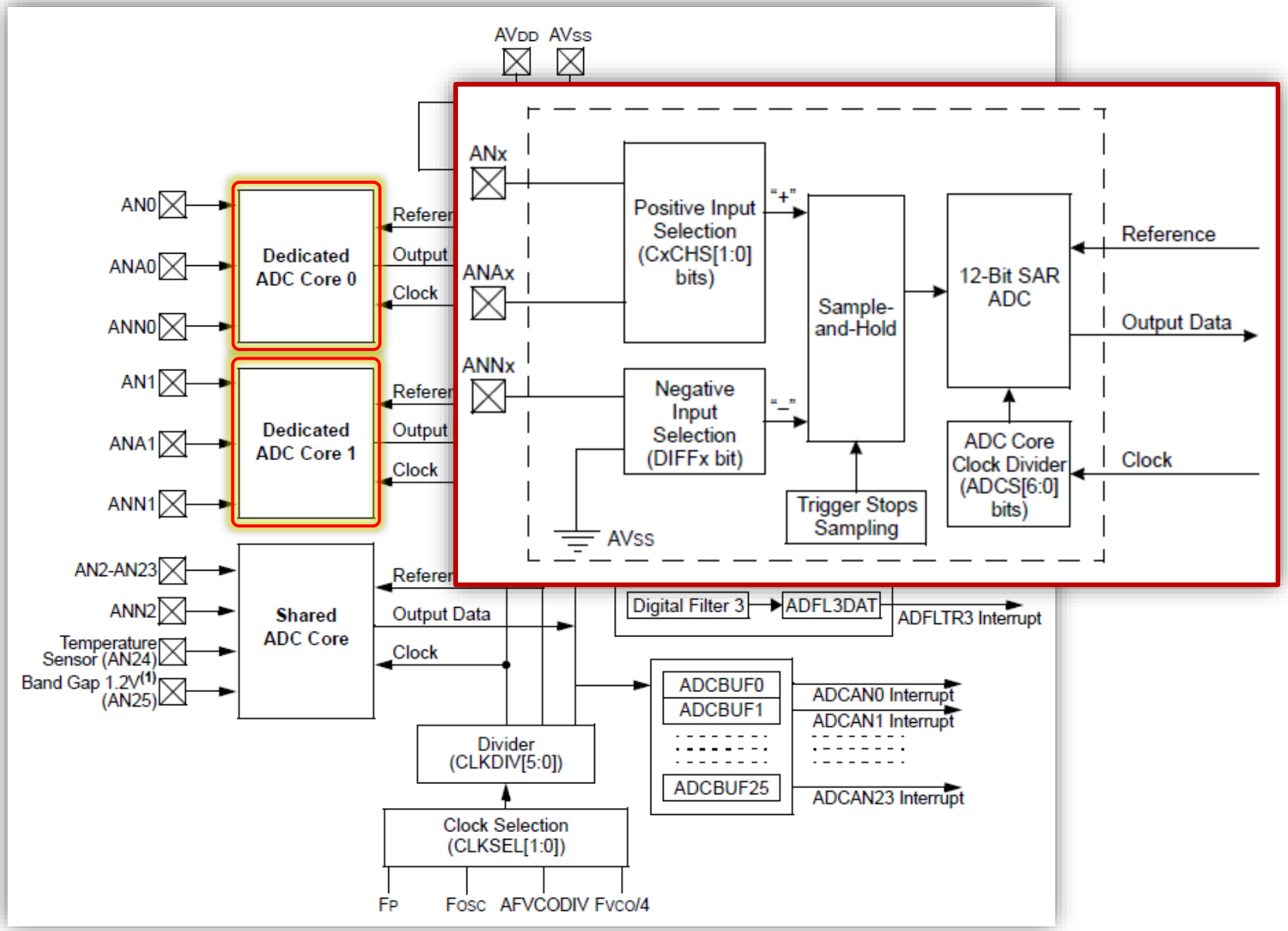
dsPIC33CK ADC Features

- **The High-Speed, 12-Bit Multiple SARs Analog-to-Digital Converter (ADC) includes the following features:**
  - Three ADC Cores: **Two Dedicated Cores** and **One Shared (common) Core**
  - User-Configurable **Resolution of up to 12 Bits**, **Up to 3.5 Msps Conversion Rate** per Channel at 12-Bit Resolution
  - Up to **24 Analog Input Channels**, with a Separate 16-Bit Conversion Result Register for each Input
  - Conversion Result can be Formatted as Unsigned or Signed Data
  - Simultaneous Sampling of up to Three Analog Inputs, Channel Scan Capability
  - Multiple Conversion Trigger Options for each Core, including:
    - PWM triggers from CPU cores
    - MCCP/SCCP modules triggers
    - CLC modules triggers
    - External pin trigger event (ADTRG31)
    - **Software trigger**
  - Four Integrated Digital Comparators with Dedicated Interrupts
  - Four Oversampling Filters with Dedicated Interrupts

dsPIC33CK ADC Module Block Diagram

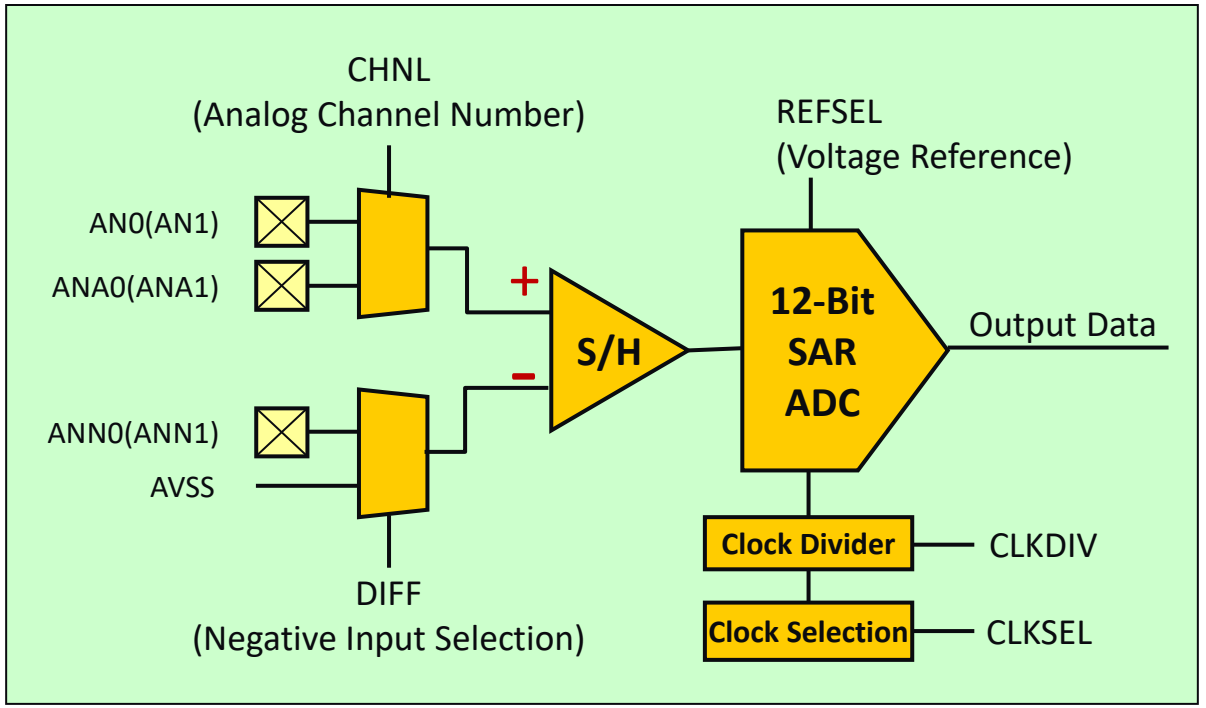


ADC Dedicate Core Block Diagram

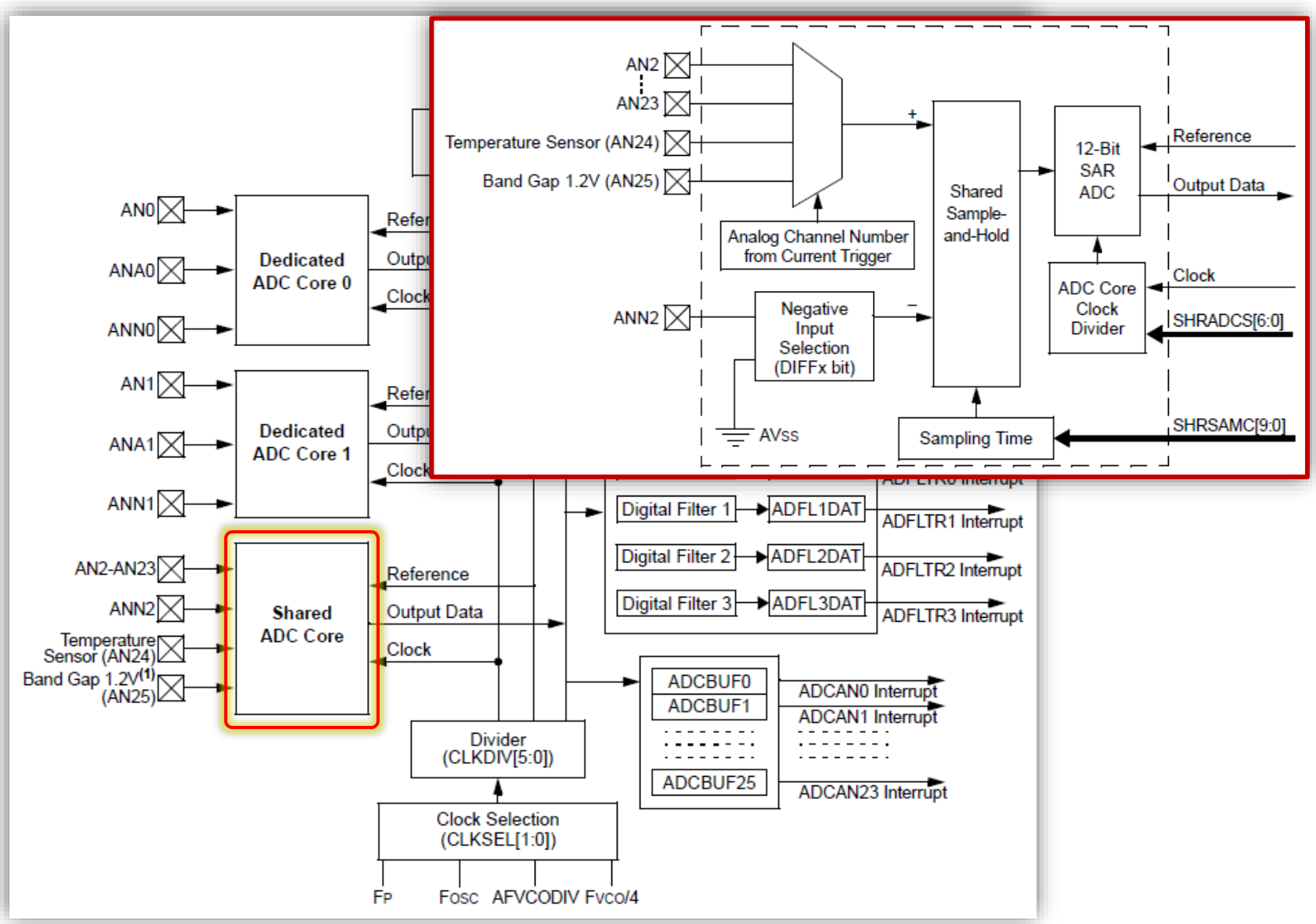


ADC Dedicate Core Channel Selection

- One analog input AN0(AN1) and one alternate input ANA0(ANA1) to connect as Analog Positive input of S/H.
- ANN0(ANN1) or AVSS as Analog Negative input of the S/H.
- The differential input voltage is Positive – Negative.

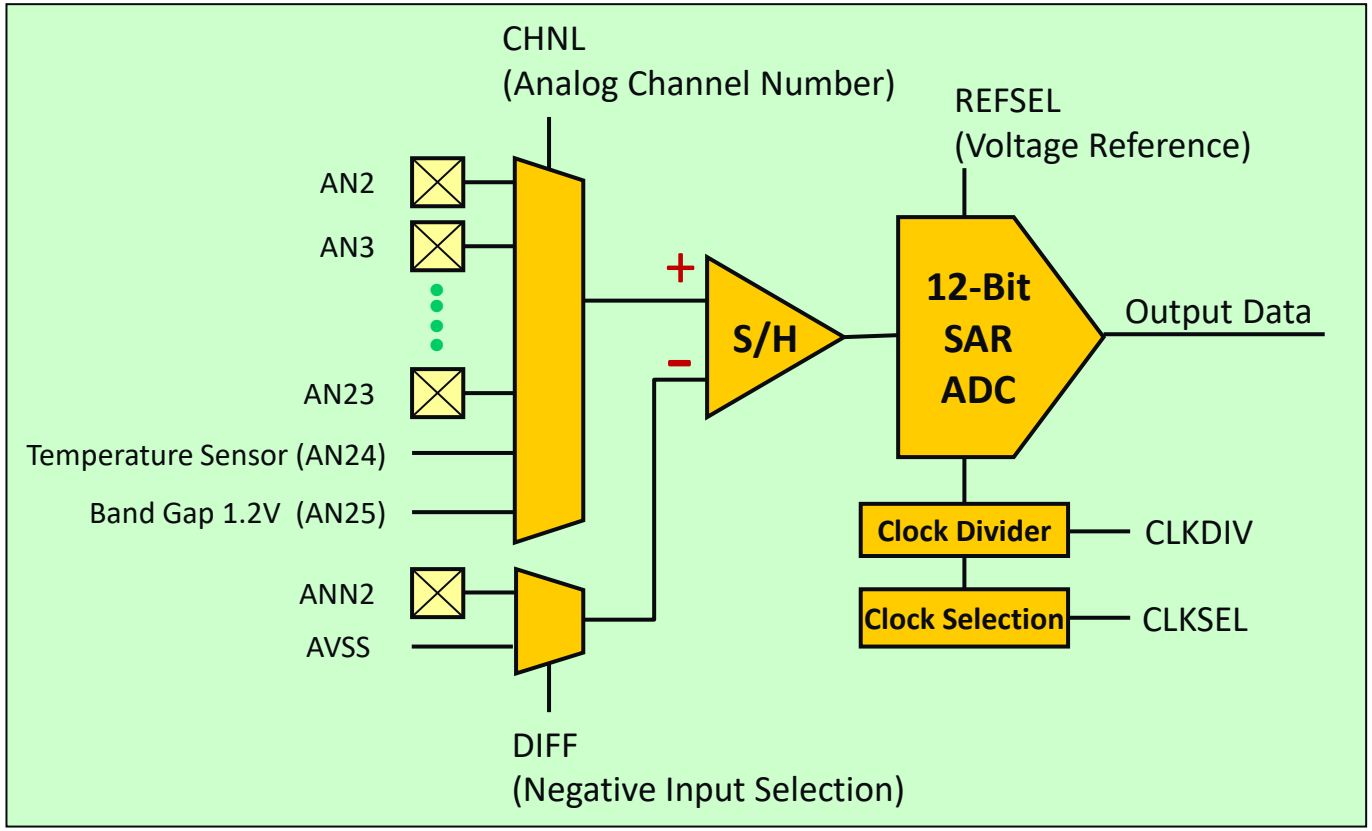


ADC Shared Core Block Diagram



ADC Shared Core Channel Selection

- 24 analog channel inputs to connect as Analog Positive input of S/H.
- ANN2 or AVSS as Analog Negative input of the S/H.
- The differential input voltage is Positive – Negative.



ADC Functions of MCC

- **MCC Provide below functions for ADC (Interrupt Enabled):**

  Prototype definition : "adc1.h"

  - void ADC1_Initialize(void);
  - void ADC1_Core0PowerEnable ( ) ;
  - void ADC1_Core1PowerEnable ( ) ;
  - void ADC1_SharedCorePowerEnable ( );
  - void ADC1_Set*CustomName*InterruptHandler(void* handler);
  - ADC1_*CustomName*_CallBack(uint16_t adcVal);
  - void ADC1_Setchannel_AN24InterruptHandler(void* handler);
  - void ADC1_Setchannel_AN25InterruptHandler(void* handler);
  - ADC1_channel_AN24_CallBack(uint16_t adcVal);
  - ADC1_channel_AN25_CallBack(uint16_t adcVal);

Lab9 ADC Single Channel Periodically

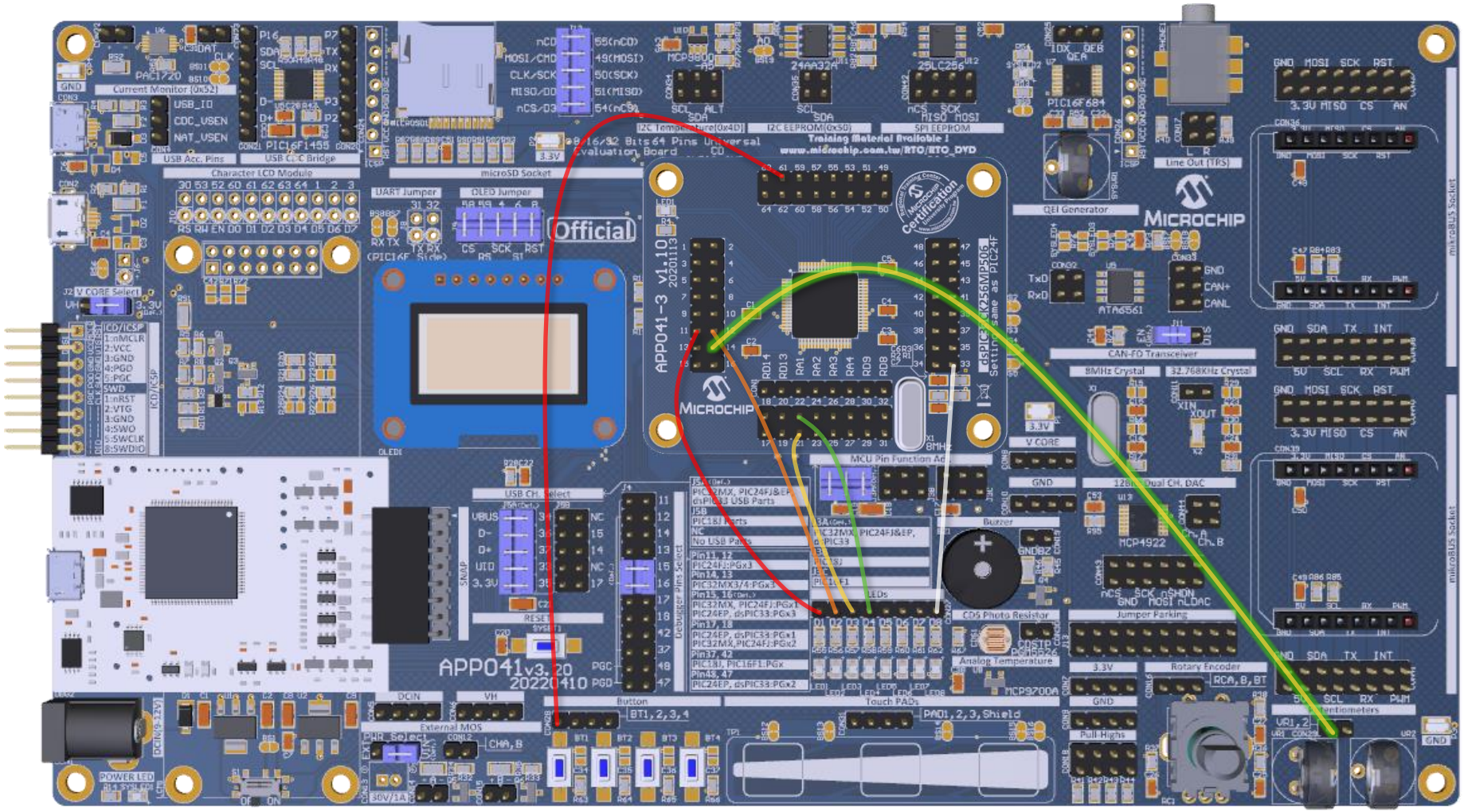
- Base on current project or Open .\Exams\Lab9_xxx.X to do exercises
- Try to add ADC1 module and conversion Potentiometer (VR1) voltage to shows ADC result on OLED Display.
- Use SCCP1 Timer to software trigger of ADC conversion periodically.
  - "ADC Core" > "Core0"
  - "Core Channel" > "AN0"
  - "Custom Name" > "VR1"
  - "Trigger Source" > "Common Software Trigger"
  - "Interrupt" > "Enable"
- Please connect RA0 (AN0, Pin14) to Potentiometer (VR1) to get analog voltage of potentiometer.

Let's go!

MICROCHIP

Lab9 Single Channel Periodically

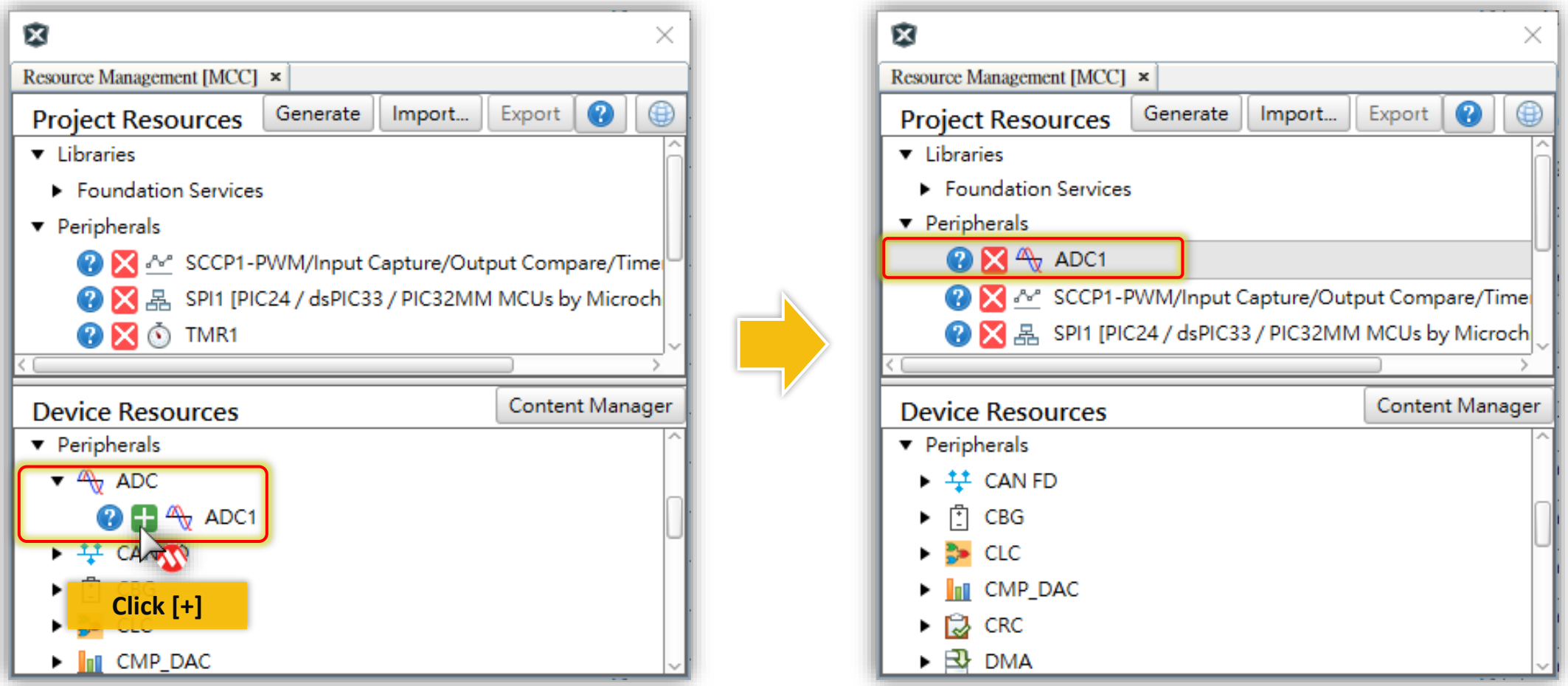
Connection



Lab9 Single Channel Periodically

Step 1

- **Click [+] to add ADC > ADC1 Module.**



Lab9 Single Channel Periodically

Notice

- **For the dedicated core, the throughput includes $4T_{ADCORE}$ sampling time and $13T_{ADCORE}$ conversion time.**
- **For the shared core, the throughput includes $10T_{ADCORE}$ sampling time and $13T_{ADCORE}$ conversion time.**
  - Clock Source : $F_{SRC} = F_{OSC}/2 = 32MHz/2 = 16MHz$, $T_{SRC} = 1/16MHz = 62.5ns$
  - $1T_{ADCORE} = T_{SRC} \times CLKDIV(1) \times ADCS(2) = 62.5ns \times 1 \times 2 = 125ns$
  - $1T_{ADCORE} = T_{SRC} \times CLKDIV(1) \times SHRADCS(2) = 62.5ns \times 1 \times 2 = 125ns$
  - **Dedicated core sampling time:** $4T_{ADCORE} = (125ns \times 4) = 500ns$
  - **Share core sampling time:** $10T_{ADCORE} = (125ns \times 10) = 1250ns$
  - **Dedicated core sampling time is preset to $4T_{ADCORE}$ (500ns) , no setting is required.**
  - **Share core sampling time needs to be manually set to $10T_{ADCORE}$ (1250ns) to match the standard.**

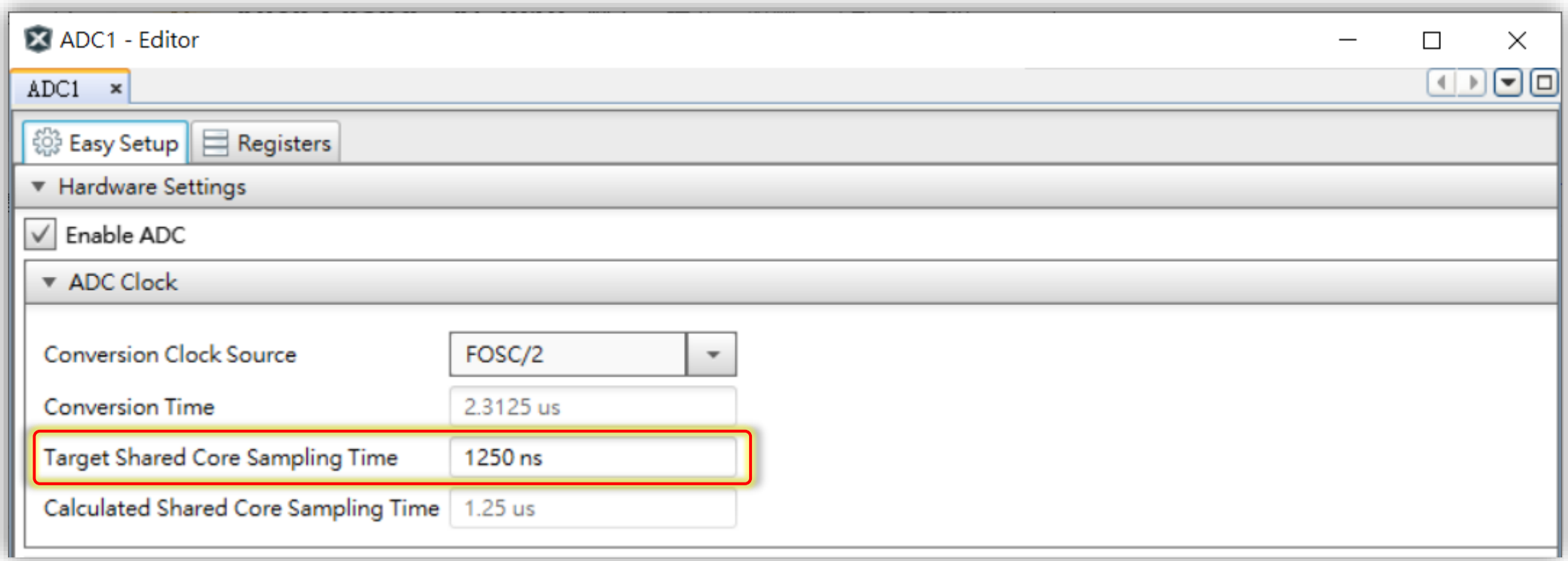
Reference Datasheet : TABLE 33-39: ADC MODULE SPECIFICATIONS

MICROCHIP

Lab9 Single Channel Periodically

Step 2

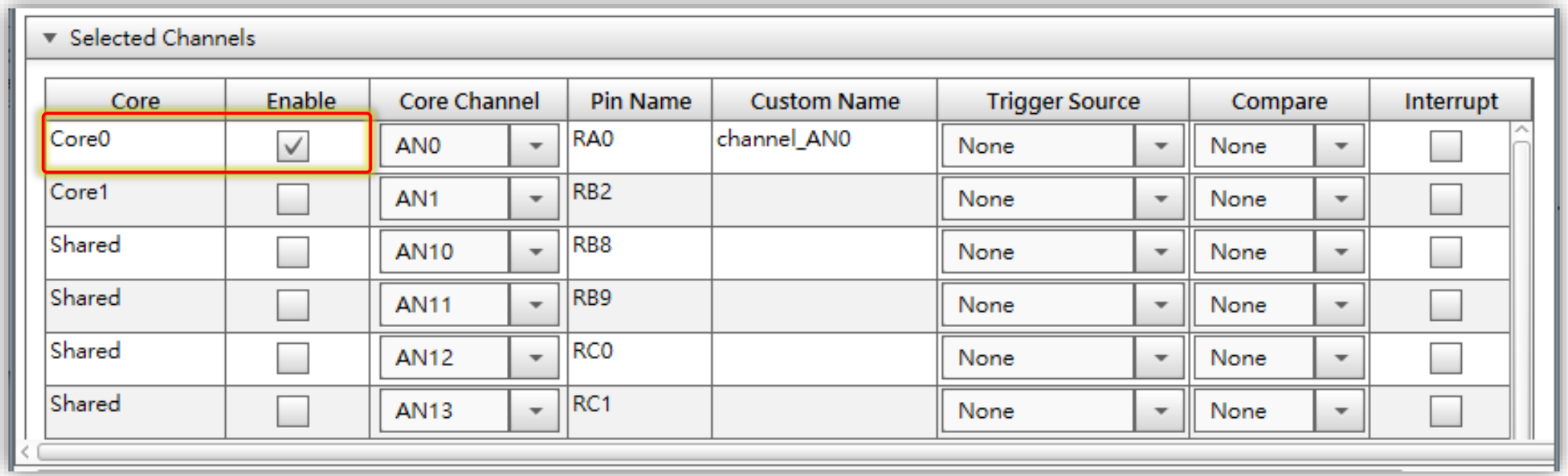
- Set **Target Shared Core Sampling Time** : **250ns** > **1,250ns**.



Lab9 Single Channel Periodically

Step 3

- Set Enable AN0 dedicate Core0.

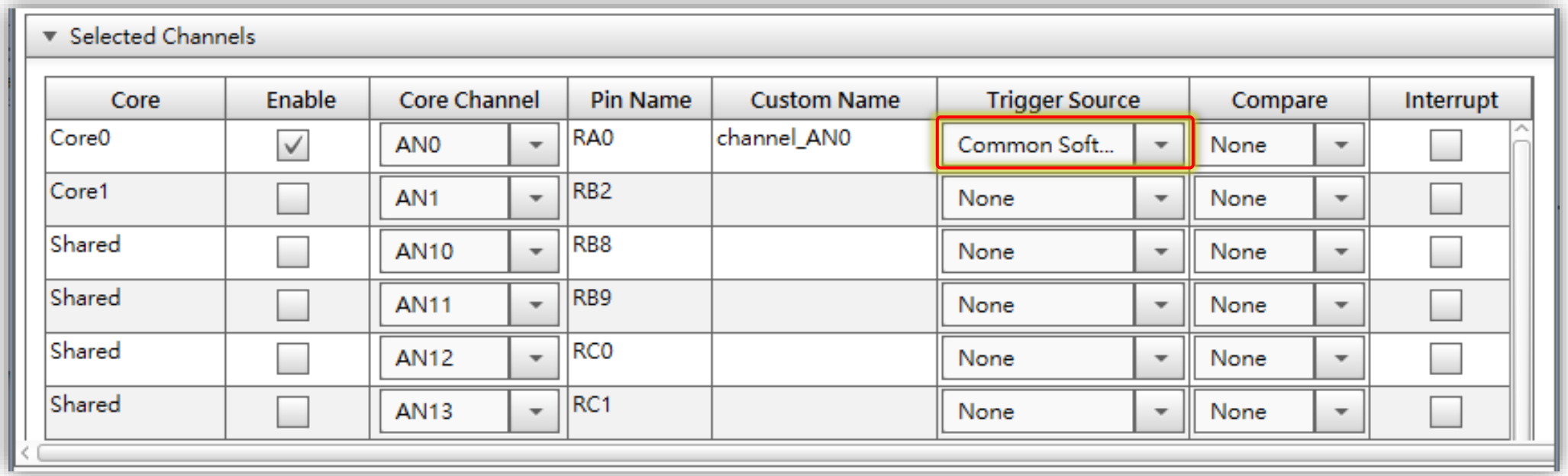


MICROCHIP

Lab9 Single Channel Periodically

Step 4

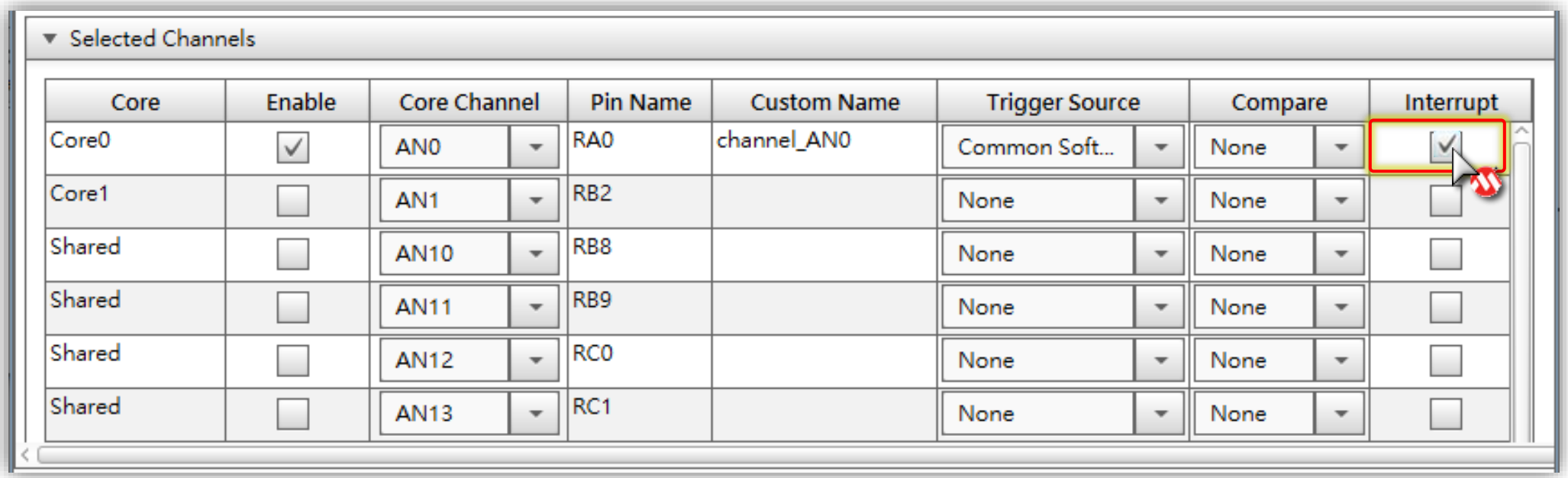
- Set Trigger Source : None > Common Software Trigger.



Lab9 Single Channel Periodically

Step 5

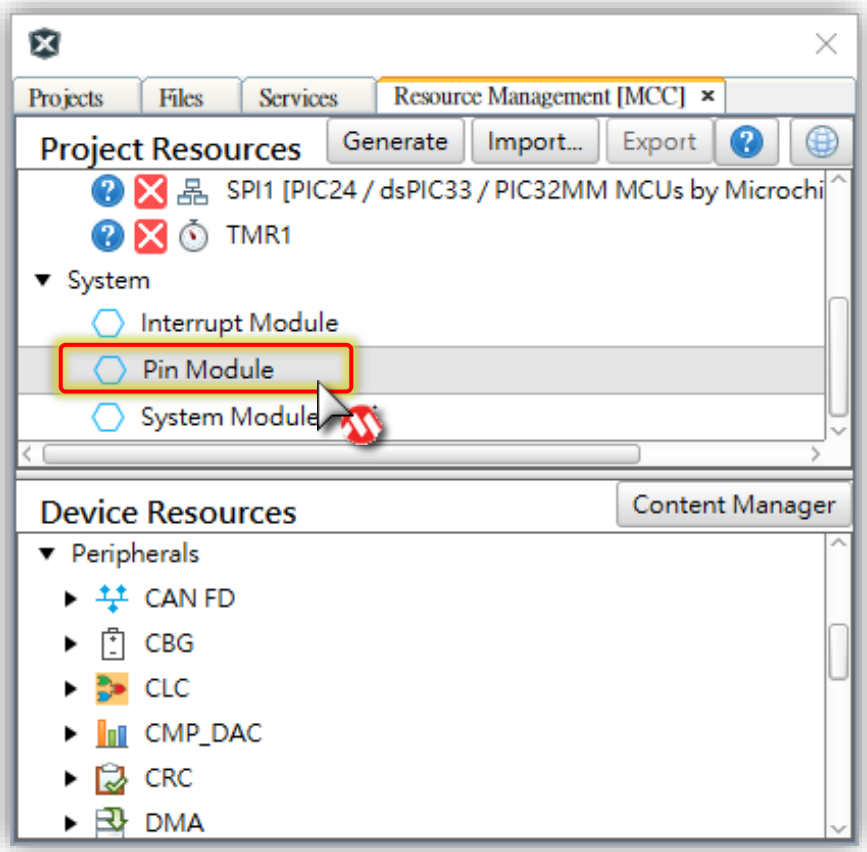
- Set **Enable Interrupt**.

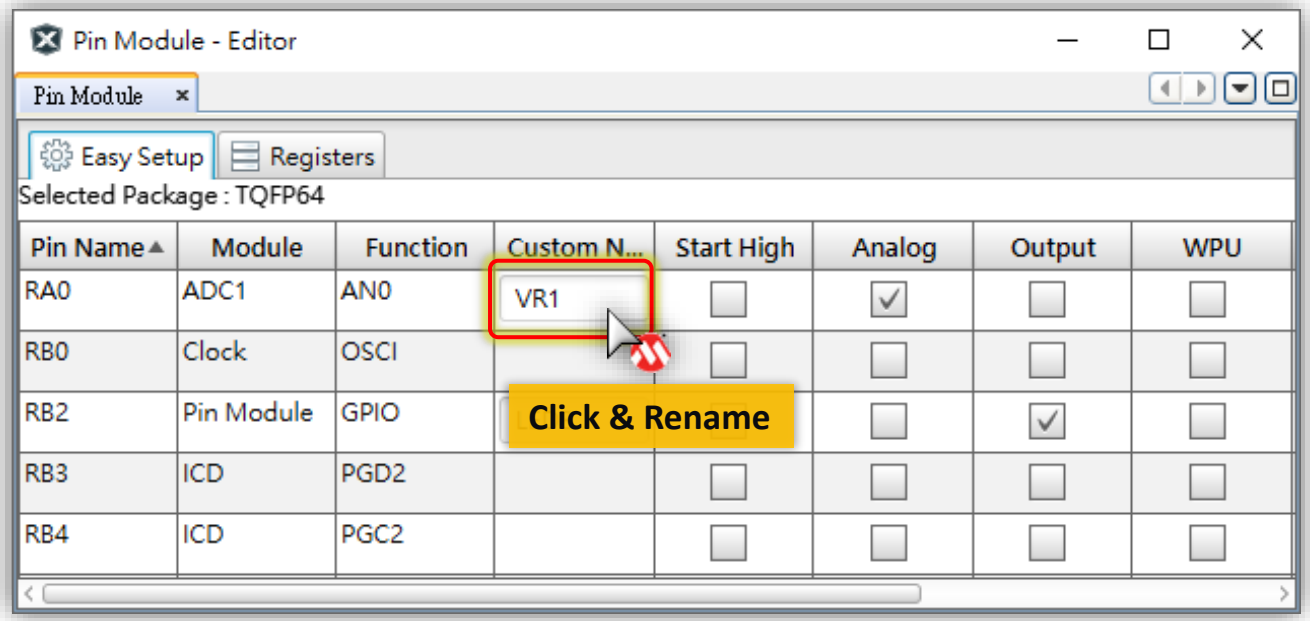


Lab9 Single Channel Periodically

Step 6

- **Set Alias for RA0.**
  - **Pin Module : RA0 (AN0) Custom Name > VR1.**

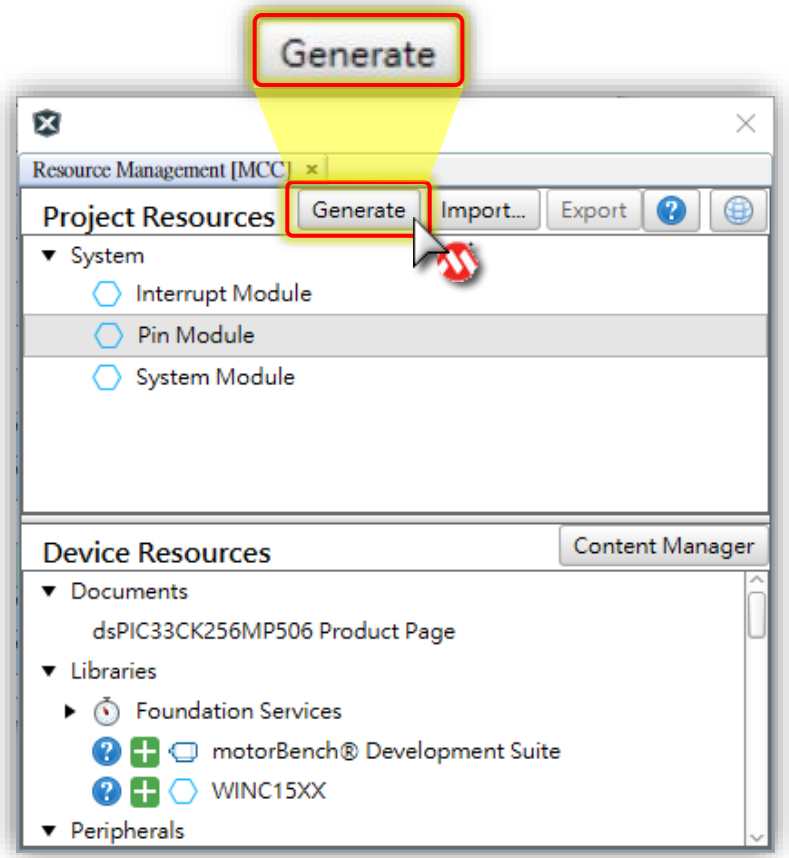


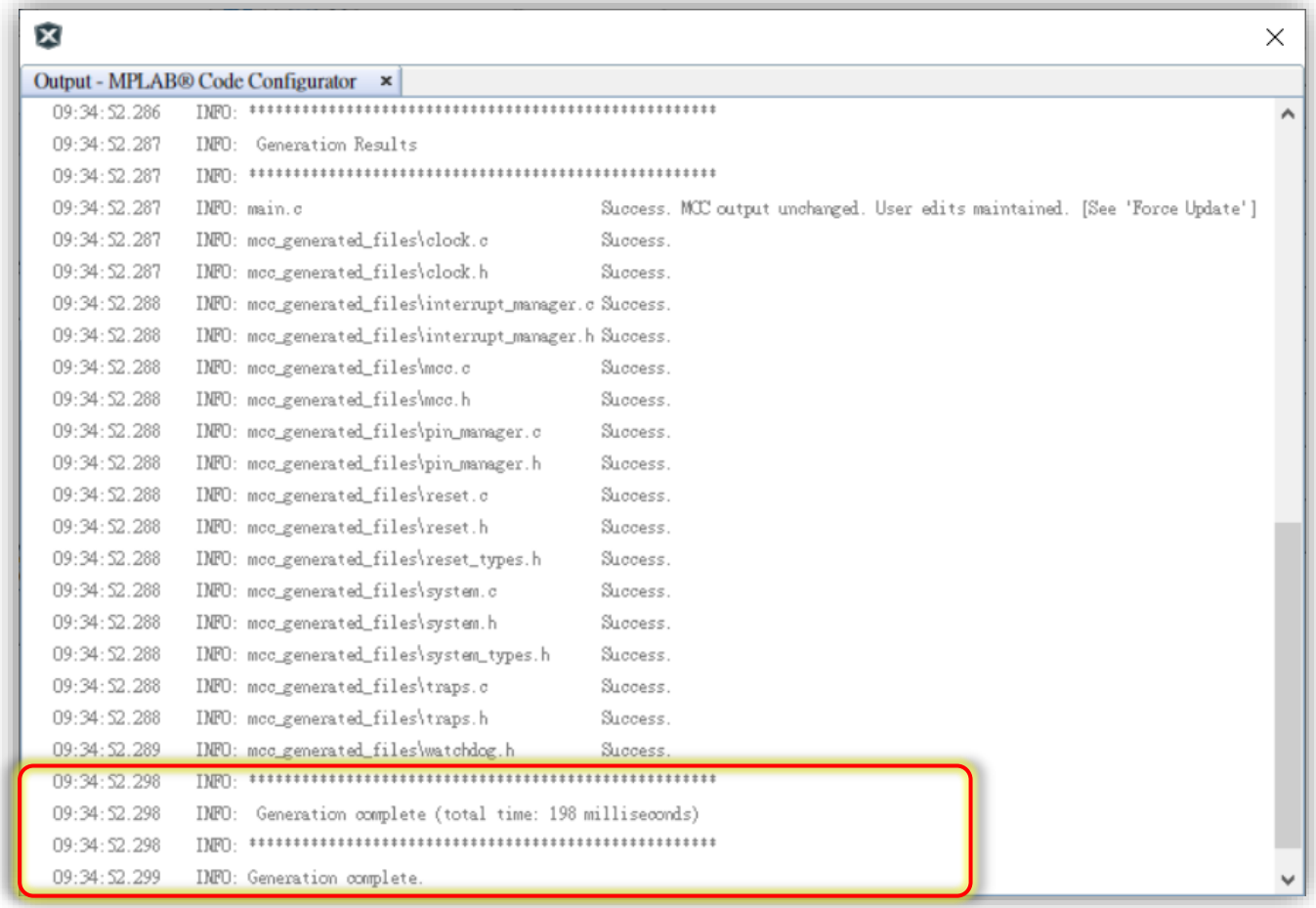


Lab9 Single Channel Periodically

Step 7

- Click "Generate" Button to generate your code.





Lab9 Single Channel Periodically

Code Example

```
#include "myOLEDlib/myOLED_ssd1306.h"
#include "mcc_generated_files/adc1.h"

…
volatile uint8_t SCCP1Flag = 0;
volatile uint8_t VR1Flag = 0;
volatile uint16_t ADC1_VR1_Buffer = 0;

int main(void)
{
    …
}

…
void SCCP1_TMR_PrimaryTimerCallBack (void)
{
    SCCP1Flag=1;
    D3_Toggle();
    D4_Toggle();
}

void ADC1_VR1_CallBack (uint16_t adcVal)
{
    VR1Flag = 1;
    ADC1_VR1_Buffer = adcVal;
}
```

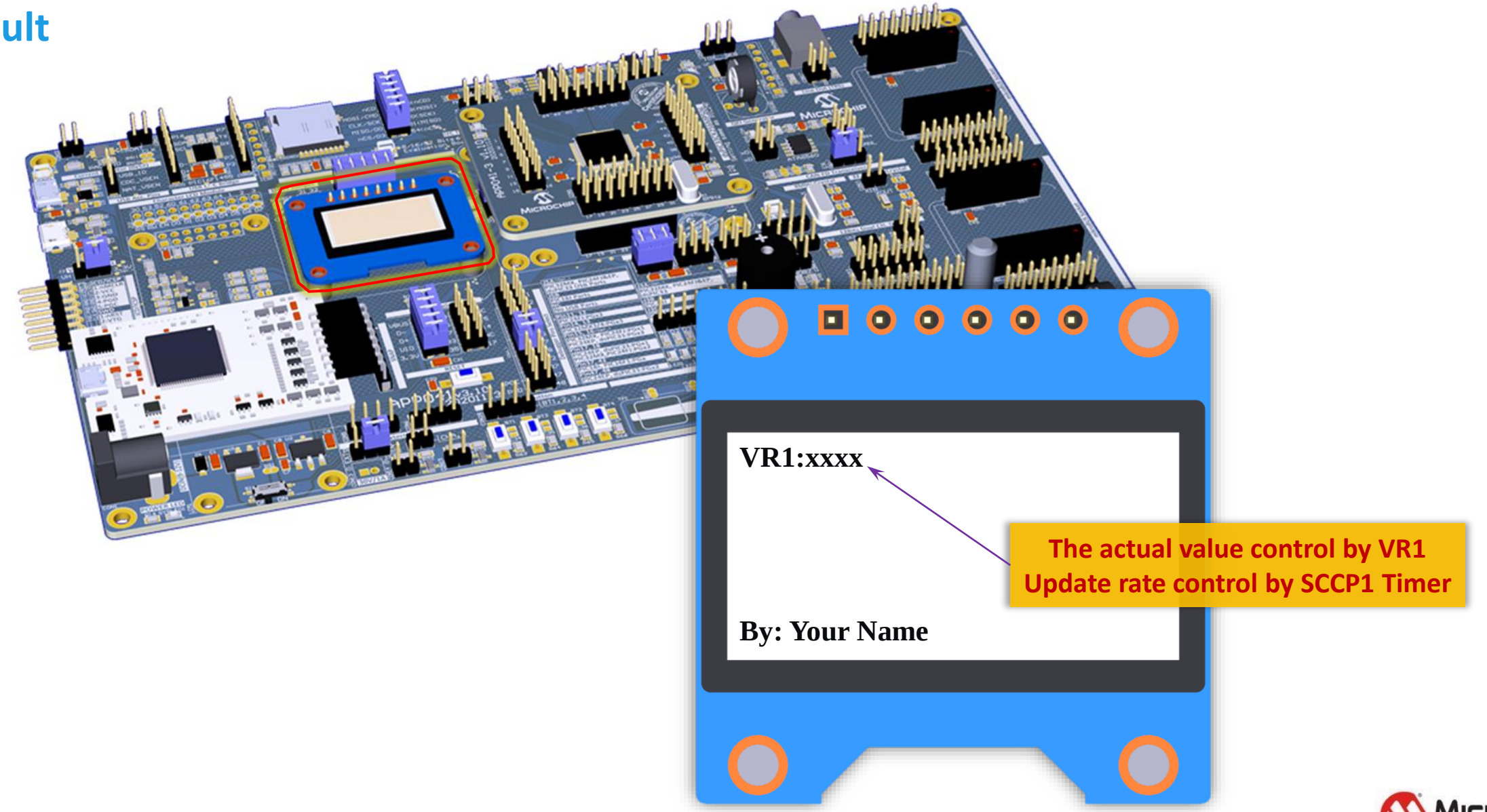
```
int main(void)
{
    // initialize the device
    SYSTEM_Initialize();
    …

    while (1)
    {
        // Add your application code
        …
        if (SCCP1Flag)
        {
            SCCP1Flag=0;
            ADC1_SoftwareTriggerEnable();
        }

        if (VR1Flag)
        {
            VR1Flag = 0;
            sprintf((char *) StringBuffer, "VR1:%4d", ADC1_VR1_Buffer);
            myOLED_DrawStr(0, 0, StringBuffer);
        }

    }
    return 1;
}
```

SCCP1 Timer for ADC Software Trigger periodically

MICROCHIP

Lab9 Single Channel Periodically

Result



Lab10 ADC Multiple Channel Periodically

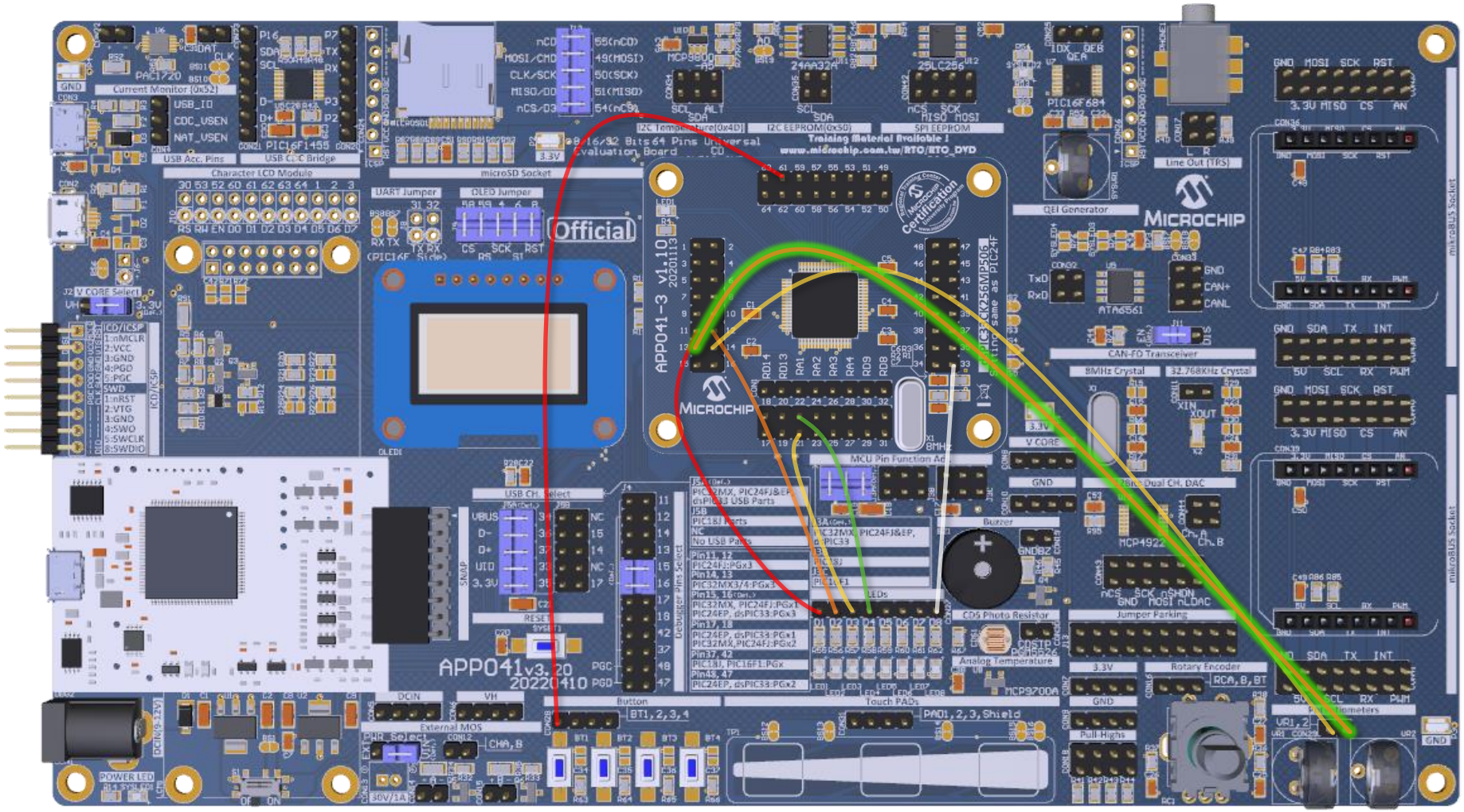
- Base on current project or Open .\Exams\Lab10_xxx.X to do exercises
- Try to add extra channel of ADC1 to get Potentiometer VR2 voltage.
- Use SCCP1 Timer to software trigger of ADC conversion periodically.
  - "ADC Core" -> "Shared"
  - "Core Channel" -> "AN12"
  - "Custom Name" -> "VR2 "
  - "Trigger Source" -> "Common Software Trigger"
  - "Interrupt" -> "Enable"
- Please connect RC0 (AN12, Pin13) to Potentiometer (VR2) to get analog voltage of potentiometer.
- Show both VR1 and VR2 result on OLED Display.

Let's go!

MICROCHIP

Lab10 ADC Multiple Channel Periodically

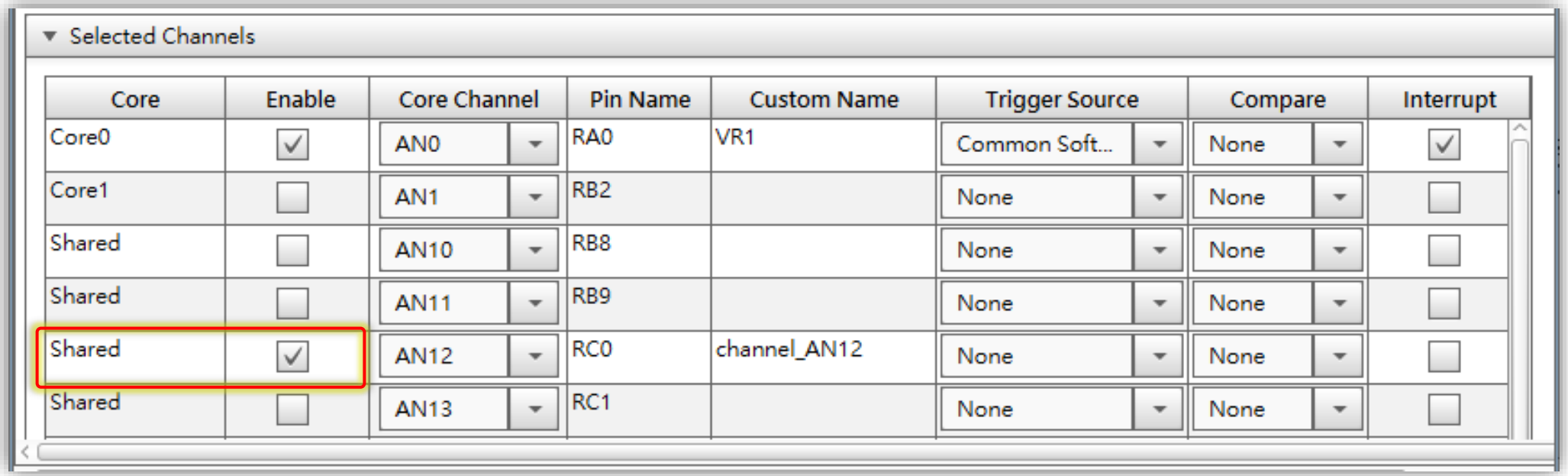
Connection



Lab10 ADC Multiple Channel Periodically

Step 1

- Set Enable AN12 shared Core.

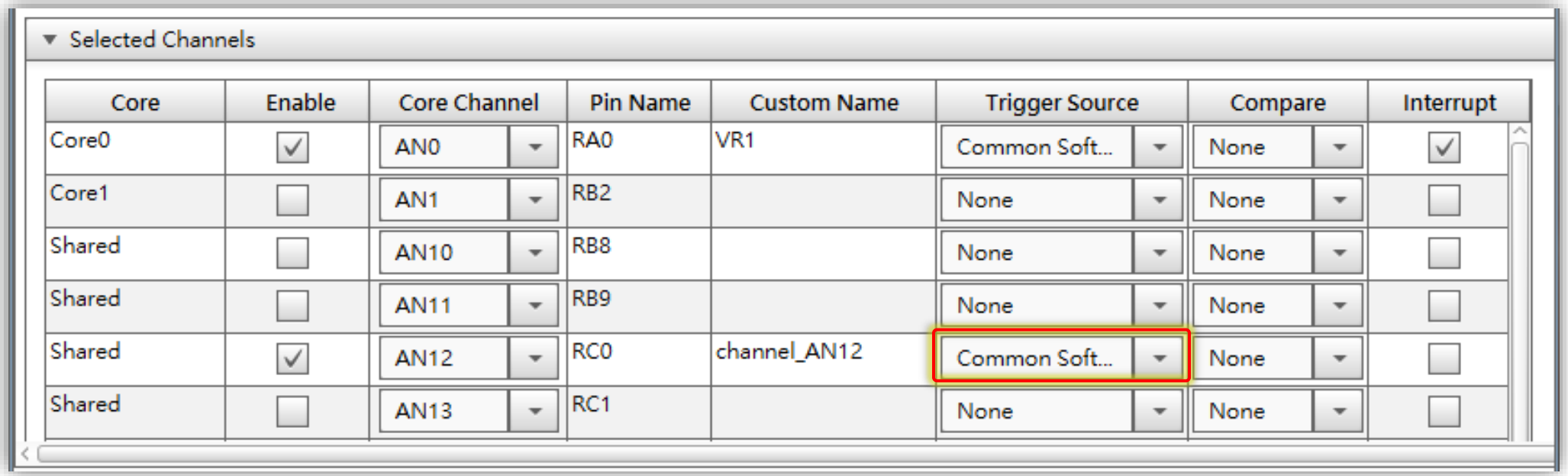


MICROCHIP

Lab10 ADC Multiple Channel Periodically

Step 2

- Set Trigger Source : None > Common Software Trigger.

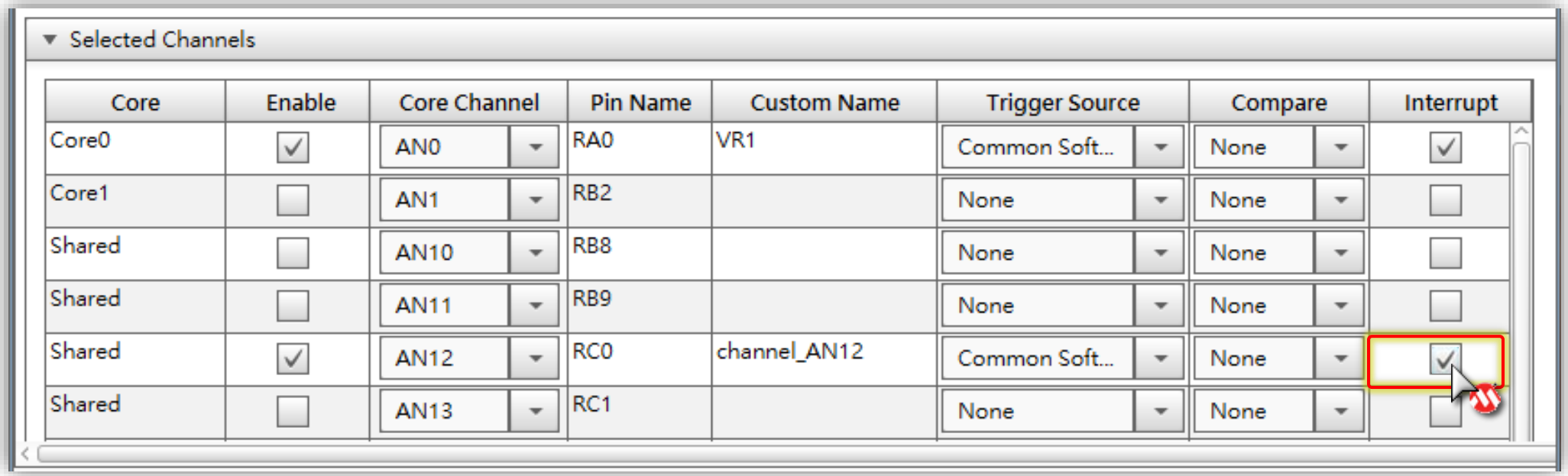


MICROCHIP

Lab10 ADC Multiple Channel Periodically

Step 3

- Set **Enable Interrupt**.

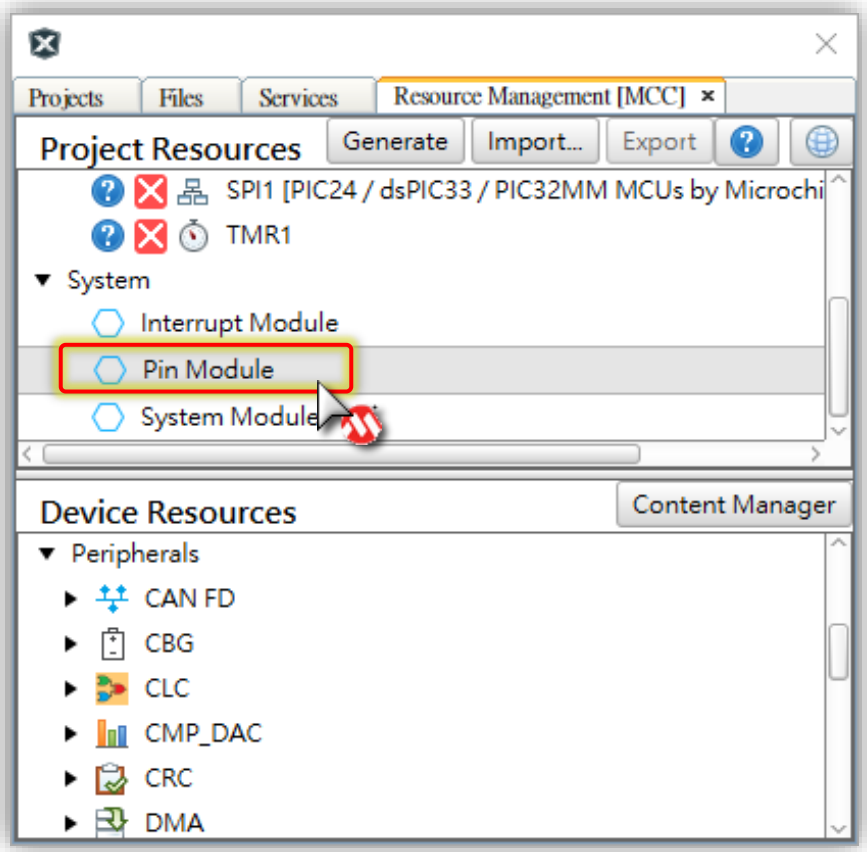


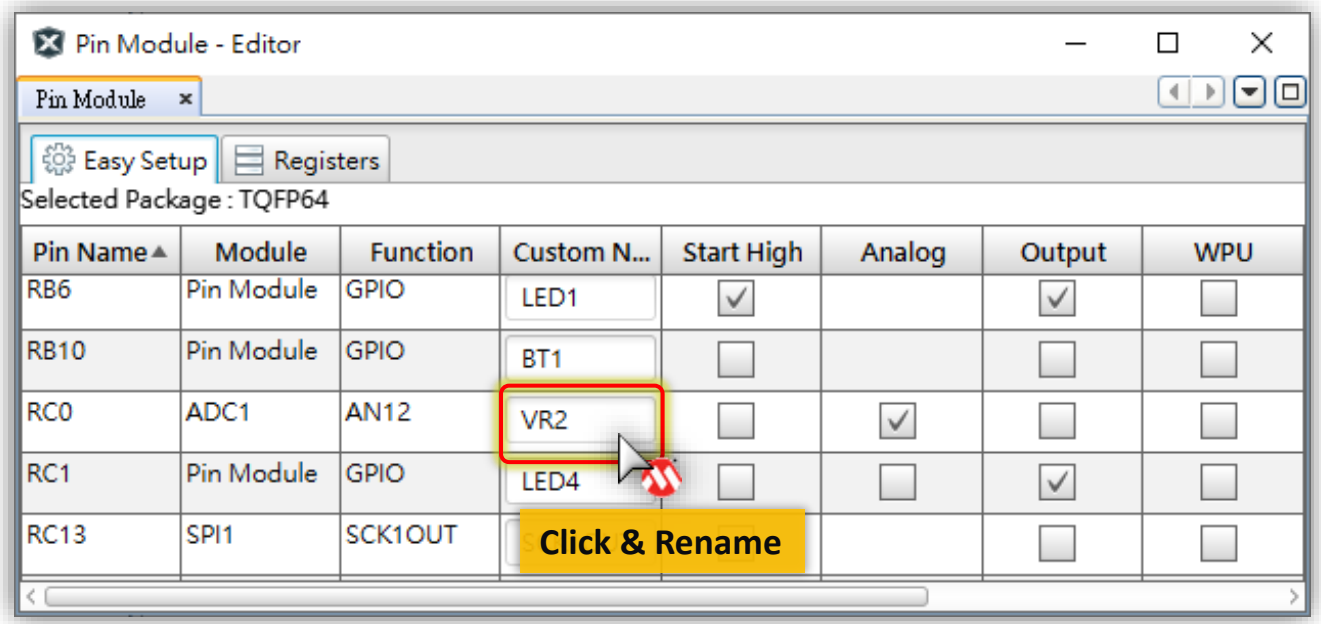
MICROCHIP

Lab10 ADC Multiple Channel Periodically

Step 4

- **Set Alias for RA0.**
  - **Pin Module : RC0 (AN12) Custom Name > VR2.**

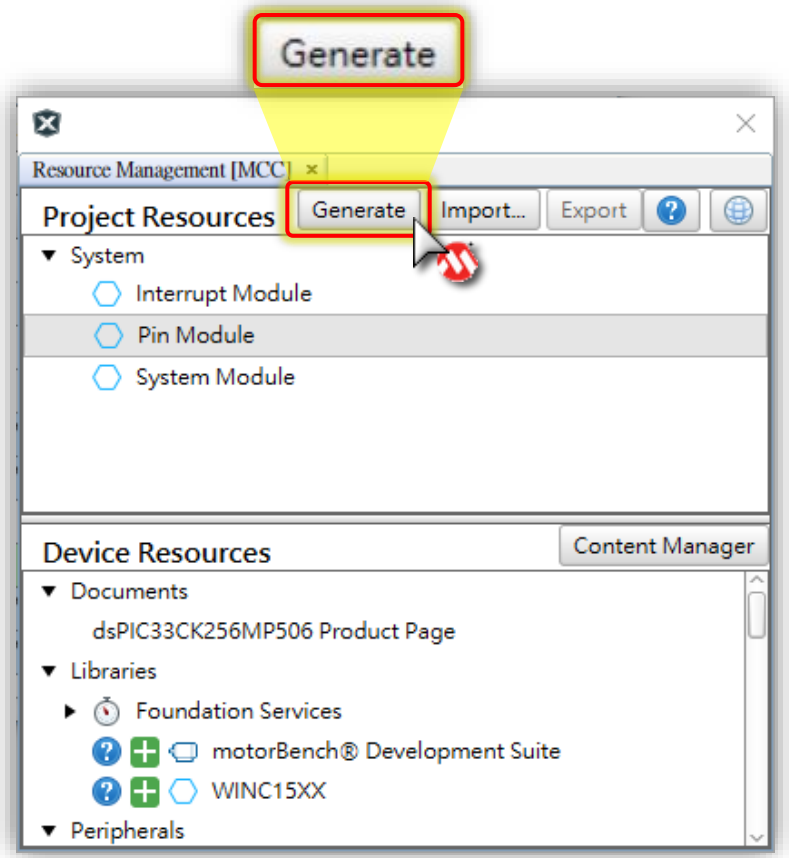


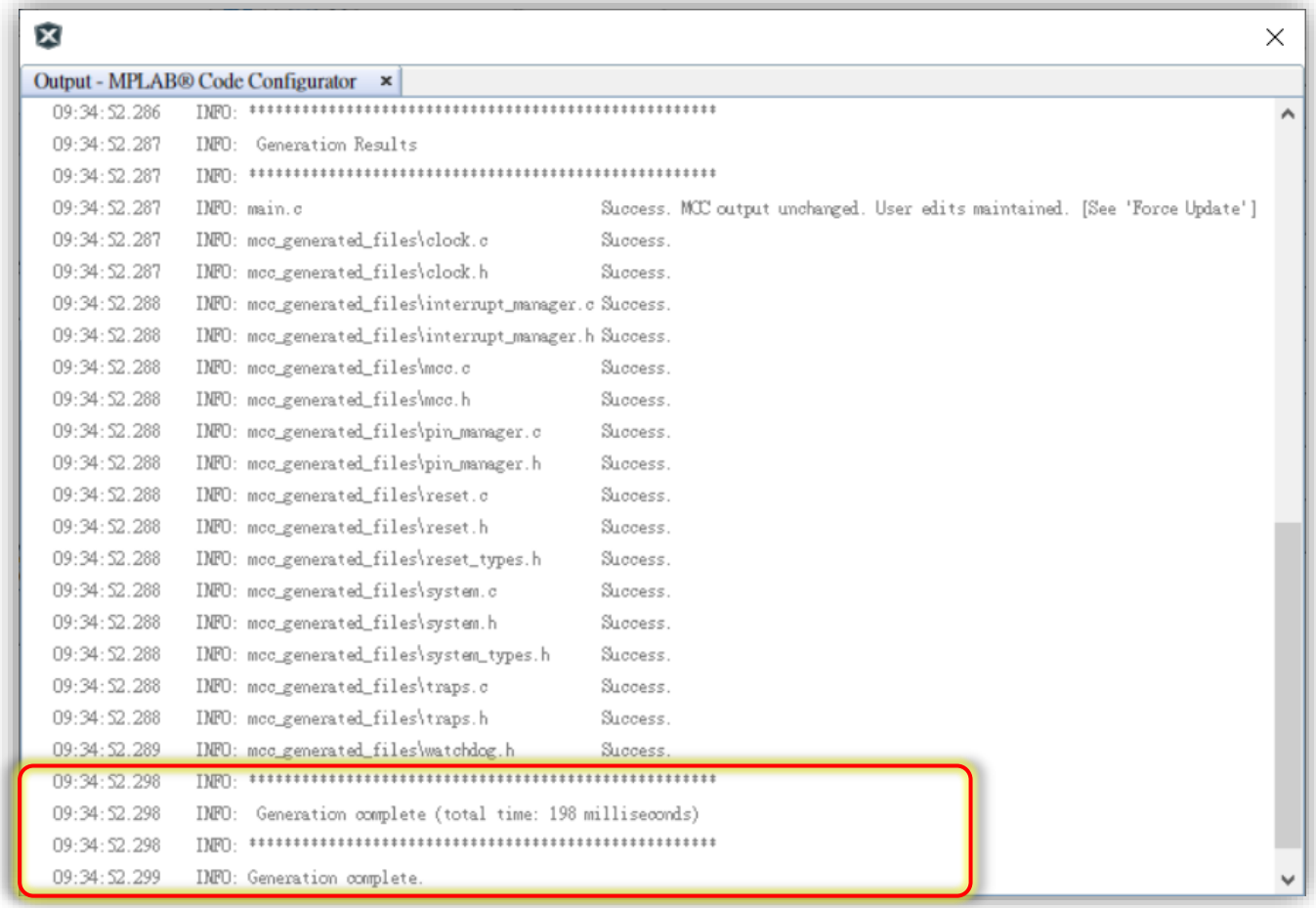


Lab10 ADC Multiple Channel Periodically

Step 5

- Click "Generate" Button to generate your code.





Lab10 ADC Multiple Channel Periodically

Code Example

```
#include "myOLEDlib/myOLED_ssd1306.h"
#include "mcc_generated_files/adc1.h"

...
volatile uint8_t SCCP1Flag = 0;
volatile uint8_t VR1Flag = 0;
volatile uint16_t ADC1_VR1_Buffer = 0;
volatile uint8_t VR2Flag = 0;
volatile uint16_t ADC1_VR2_Buffer;

int main(void)
{
   ...
}

...

void ADC1_VR1_CallBack (uint16_t adcVal)
{
   VR1Flag = 1;
   ADC1_VR1_Buffer = adcVal;
}

void ADC1_VR2_CallBack (uint16_t adcVal)
{
   VR2Flag = 1;
   ADC1_VR2_Buffer = adcVal;
}
```

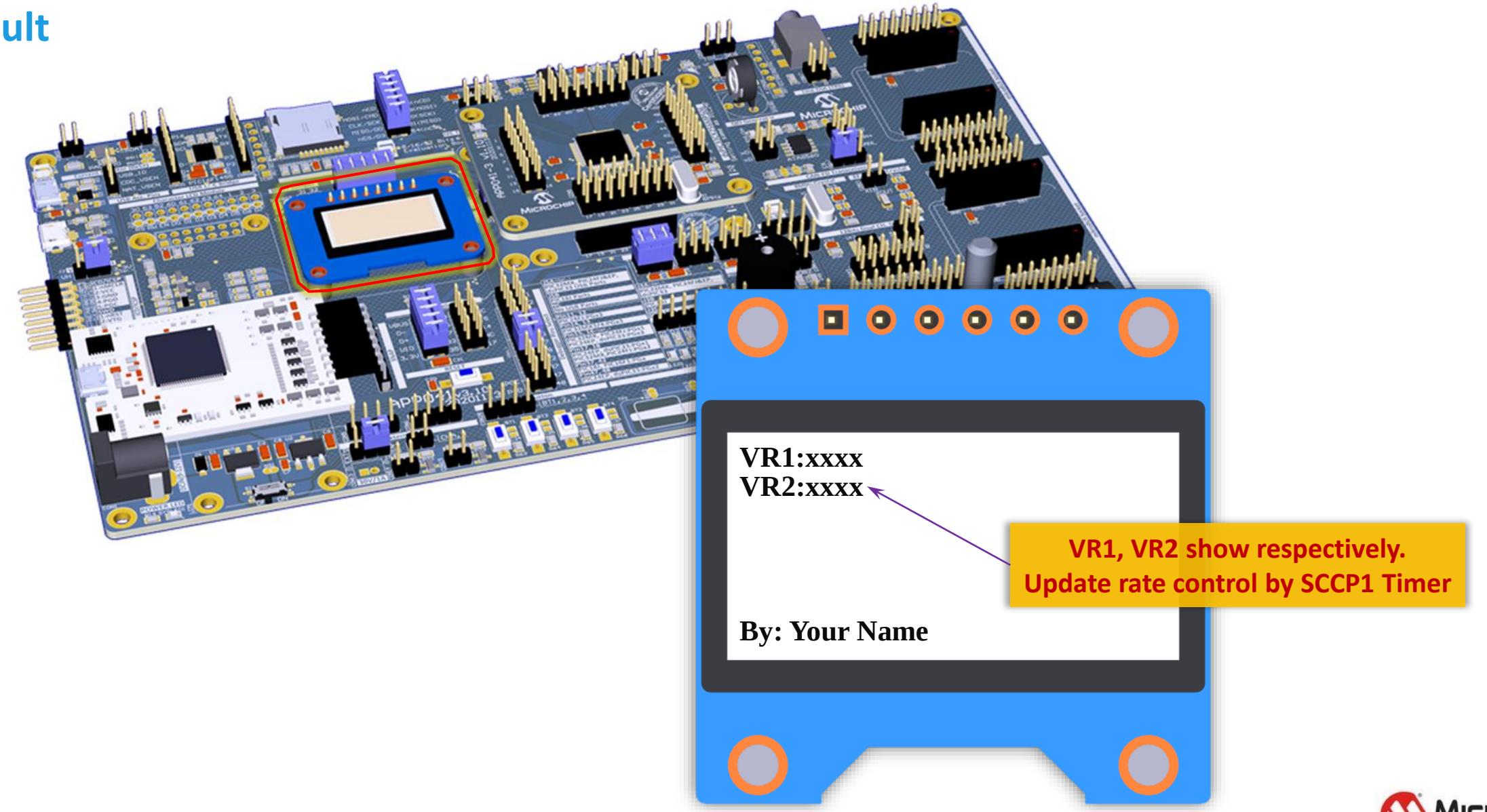
```
int main(void)
{
   ...
   while (1)
   {
      // Add your application code
      ...
      if (SCCP1Flag)
      {
         ...
         ADC1_SoftwareTriggerEnable();
      }

      if (VR1Flag)
      {
         VR1Flag = 0;
         sprintf((char *) StringBuffer, "VR1:%4d", ADC1_VR1_Buffer);
         myOLED_DrawStr(0, 0, StringBuffer);
      }

      if (VR2Flag)
      {
         VR2Flag = 0;
         sprintf((char *) StringBuffer, "VR2:%4d", ADC1_VR2_Buffer);
         myOLED_DrawStr(0, 1, StringBuffer);
      }

   }
   return 1;
}
```

SCCP1 Timer for ADC Software Trigger periodically

MICROCHIP

Lab10 ADC Multiple Channel Periodically

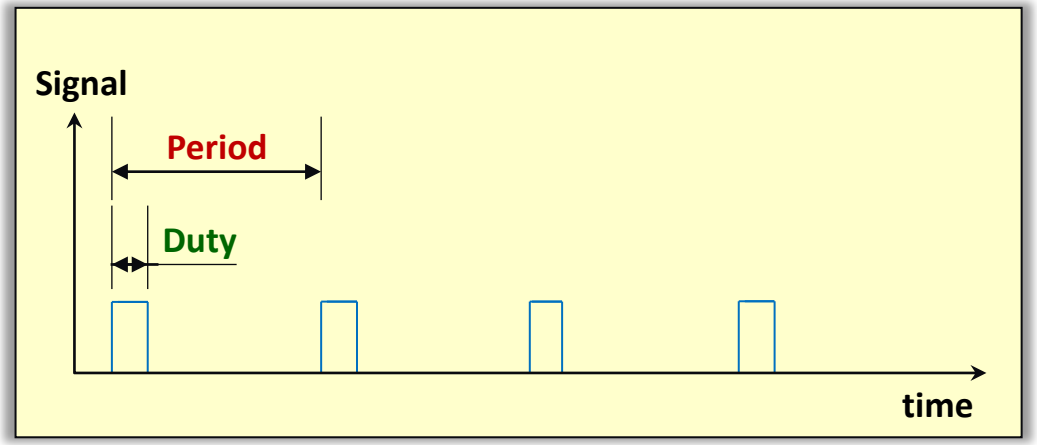
Result



High Resolution PWM Application

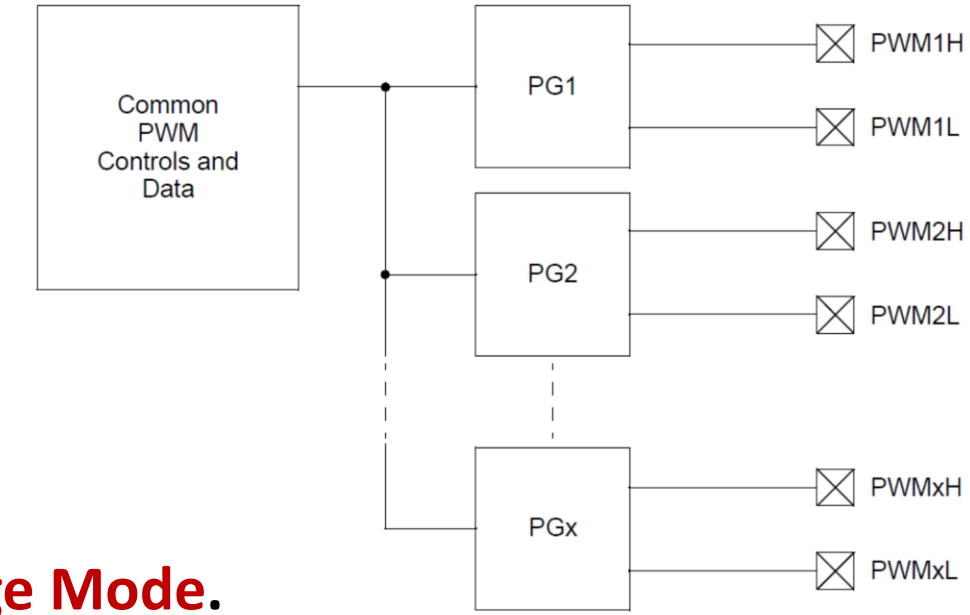
What's PWM

- **The Pulse-width modulation (PWM) Waveform is a modulation technique used to encode a message into a pulsing signal.**
- **Its main use is to allow the control of the power supplied to electrical devices, ex. motor control, ballast, LED, H-bridge, power converters, and other types of power control applications.**
- **There have two important terms**
  - **Period :**
    The single square wave duration.
  - **Duty :**
    The proportion of active time to the regular interval.



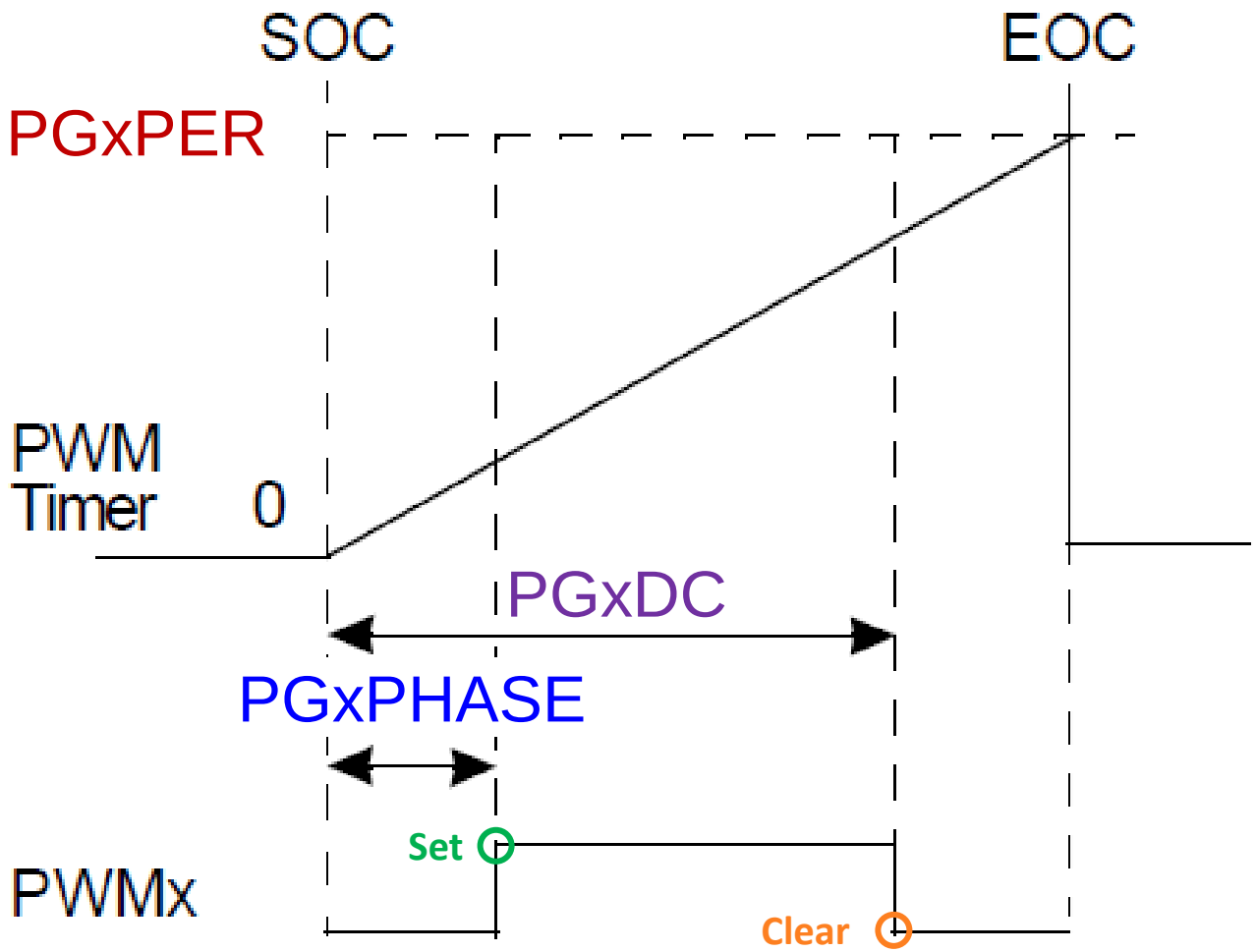
High Resolution PWM

- **This flexible module provides features to support many types of Motor Control (MC) and Power Control (PC).**
- **dsPIC33CK256MP506 include one PWM module provide maximum eight pairs of PWM outputs available.**
- **All PWM has dedicated pins except PWM4, PWM4 could has flexibility of Peripheral Select Pin(PPS).**
- **Support lot of operating modes:**
  - **Independent Edge mode.**
  - Variable Phase PWM mode.
  - Center-Aligned mode.
  - Double Update Center-Aligned mode.
  - Dual Edge Center-Aligned mode.
  - Dual PWM mode.
- **At this section, we focus on Independent Edge Mode.**



Independent Edge Mode of High Resolution PWM

- For **Independent Edge Mode**, the period time (T) has controlled by the **PGxPER** register.
- PWMx to "**Set**" after shift of **PGxPHASE** and "**Clear**" after **PGxDC**.



MICROCHIP

Lab11 PWM Output

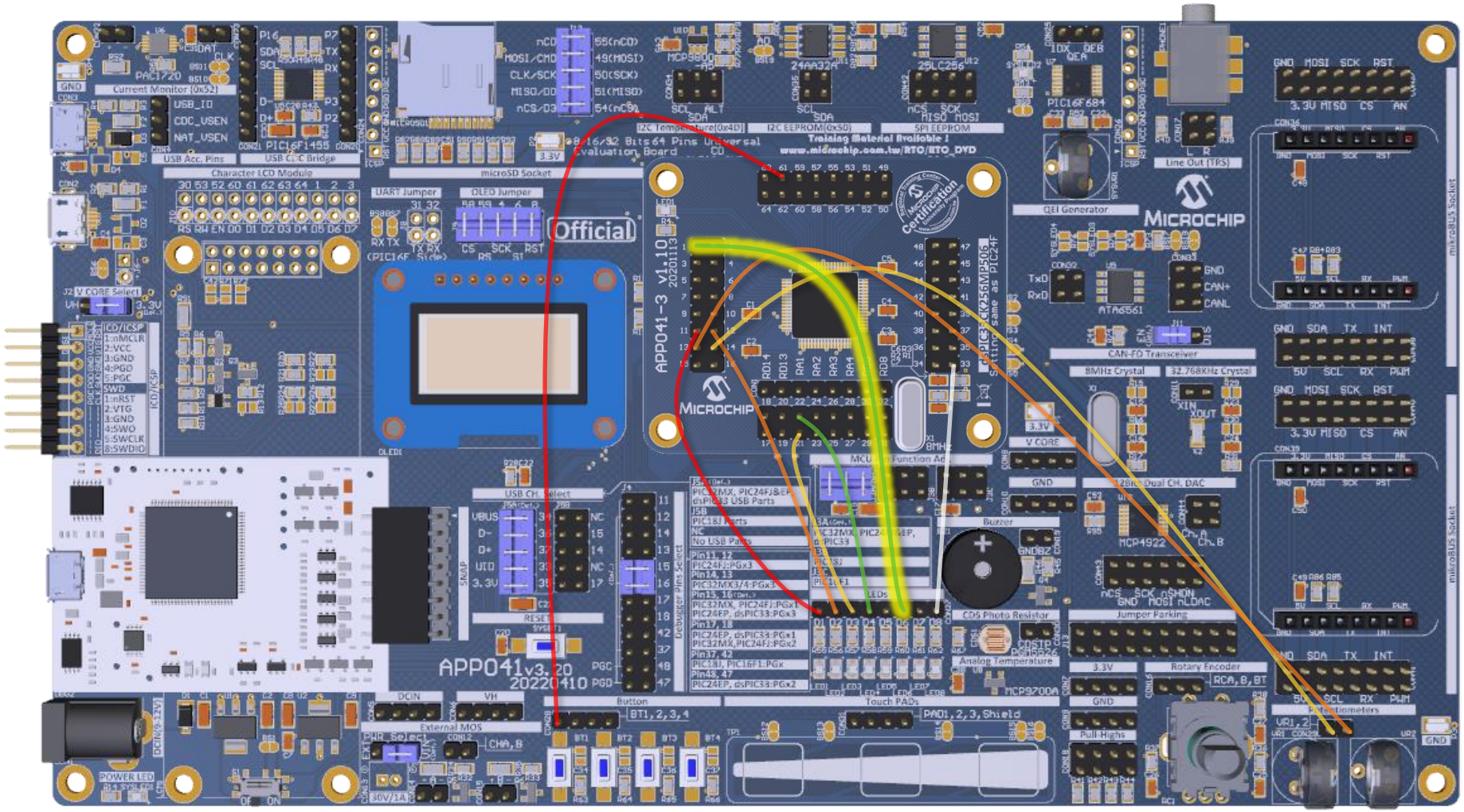
- Base on current project or Open .\Exams\Lab11_xxx.X to do exercises
- Try to initial H.R. PWM-PG1 to Independent Edge Mode, then use PWM to control LED brightness.
- H.R. PWM-PG1 module clock source set to 16MHz, Frequency is 1KHz, Duty cycle 20%.
- Please connect RB14 (PWM1_H, Pin1) to LED (D6) to observe LED brightness.

Let’s go!

MICROCHIP

Lab11 PWM Output

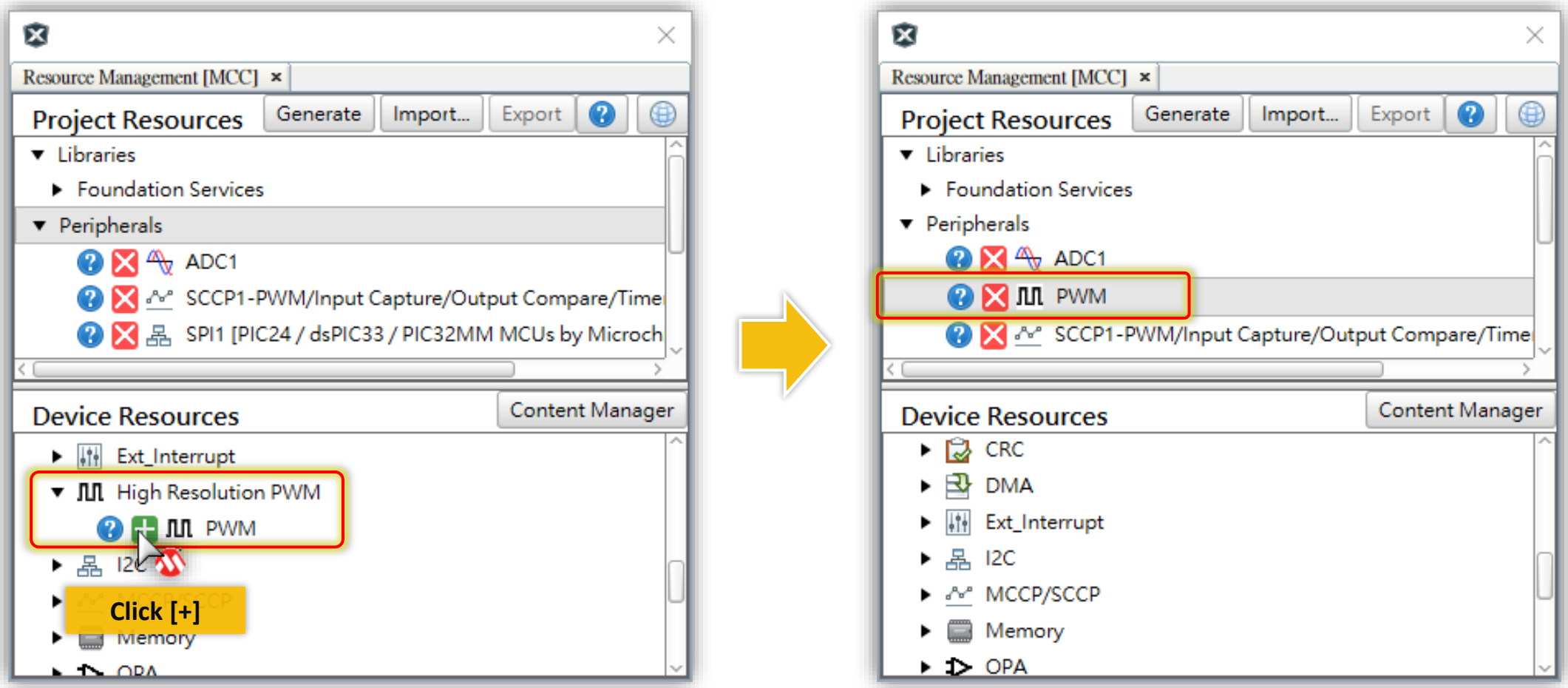
Connection



Lab11 PWM Output

Step 1

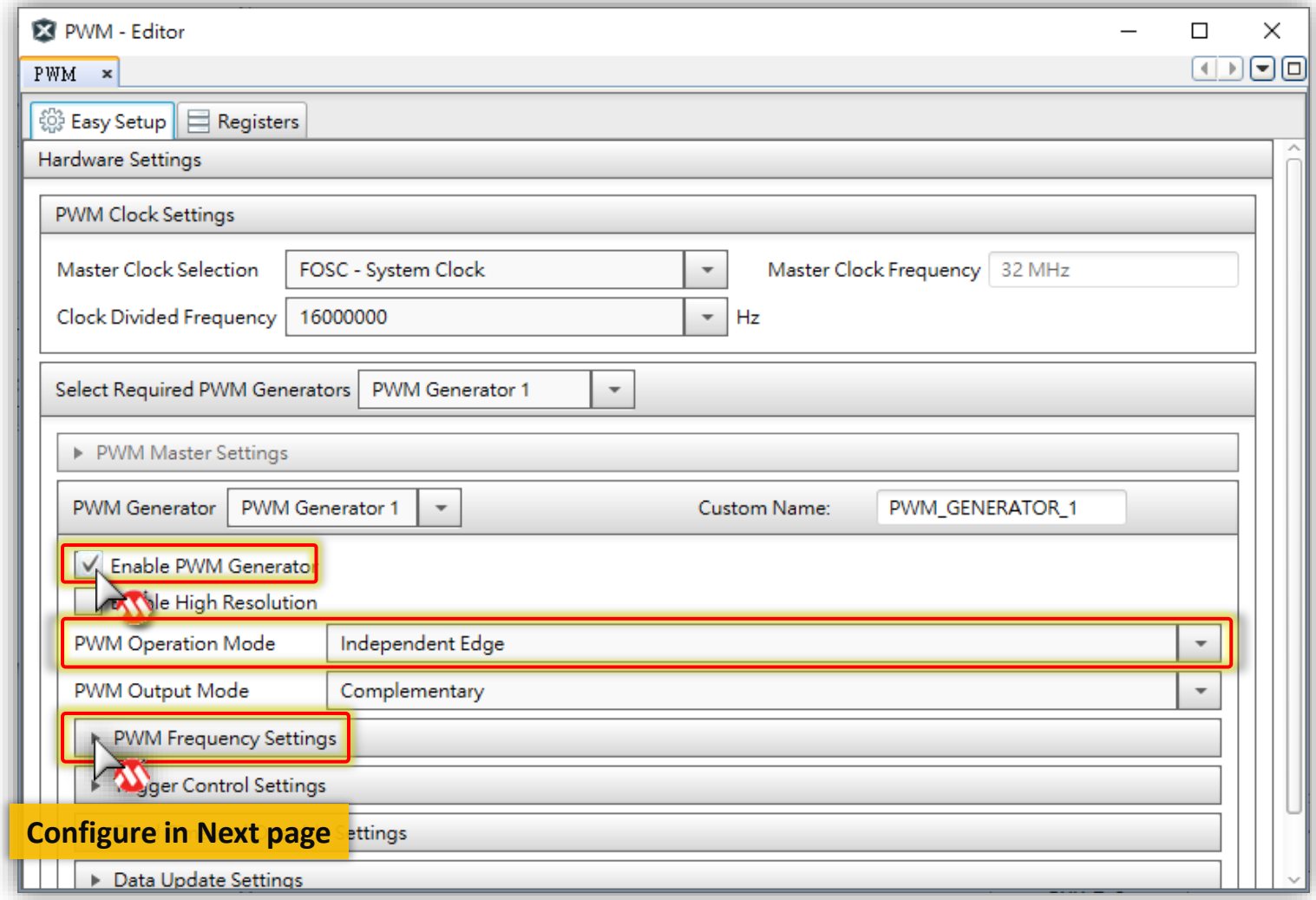
- Click [+] to add High Resolution PWM > PWM module.



Lab11 PWM Output

Step 2

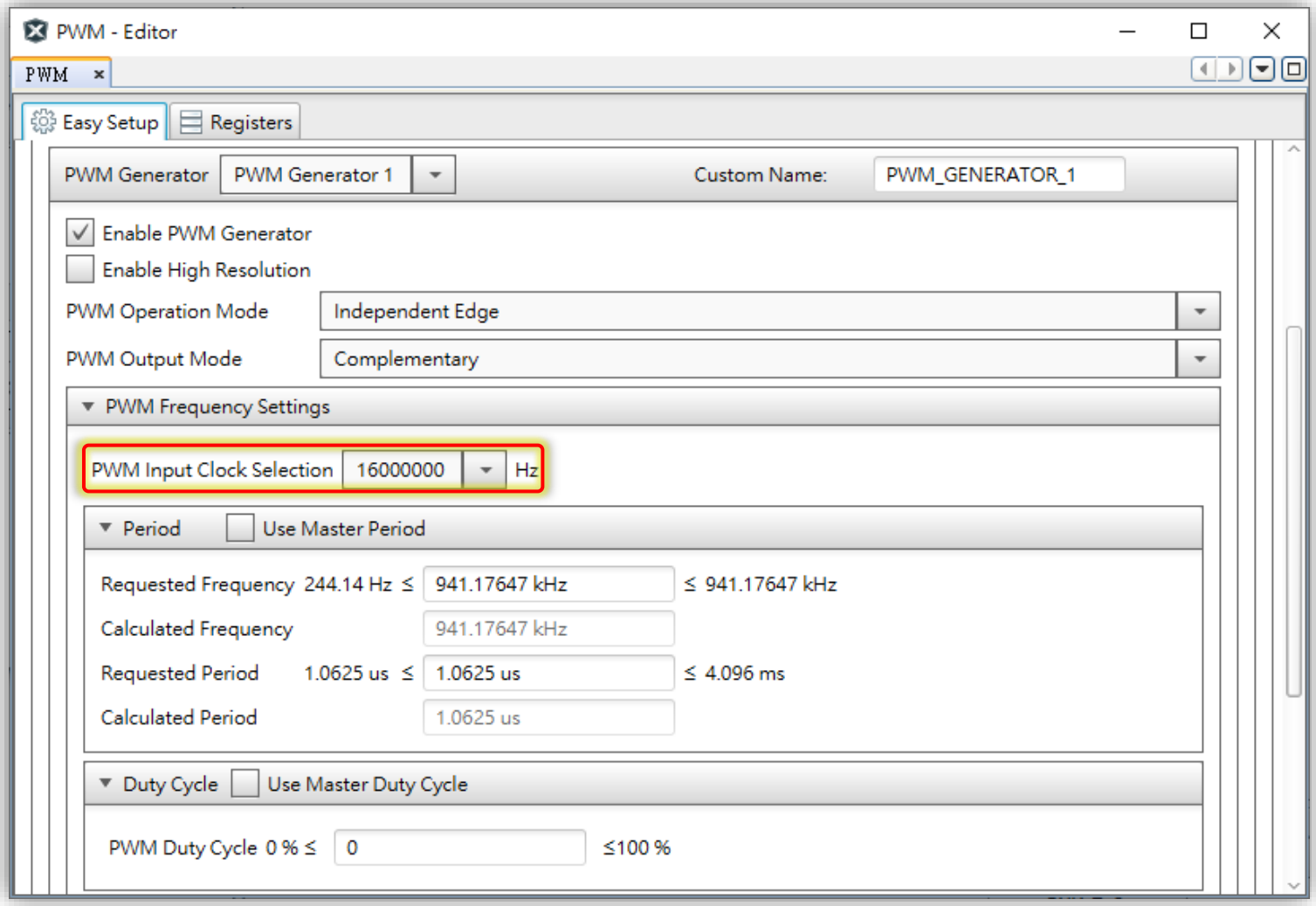
- Set Enable PWM Generator 1 and Confirm PWM Mode : Independent Edge.



Lab11 PWM Output

Step 3

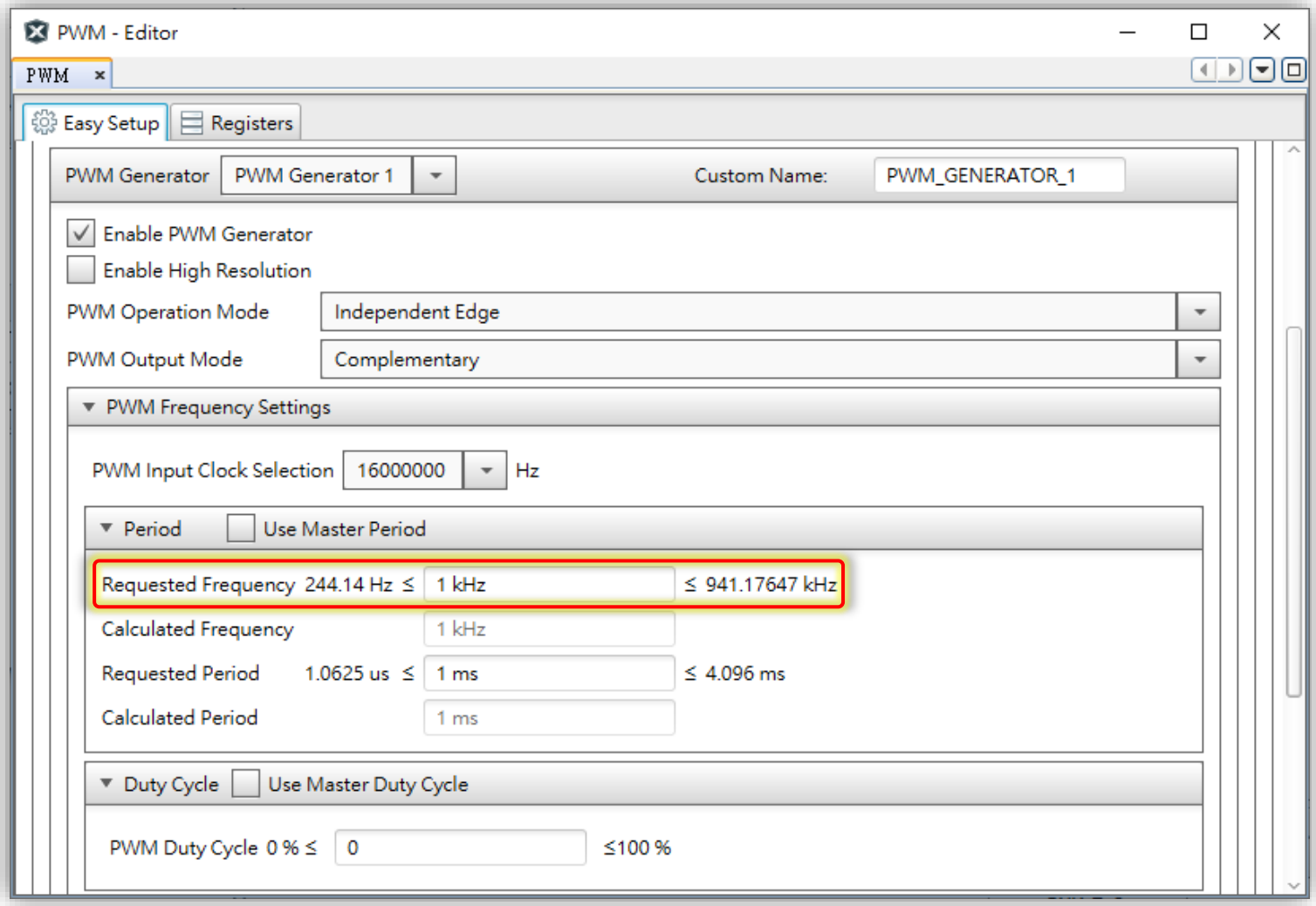
- Set PWM1 Clock Selection : 32000000 > 16000000.



Lab11 PWM Output

Step 4

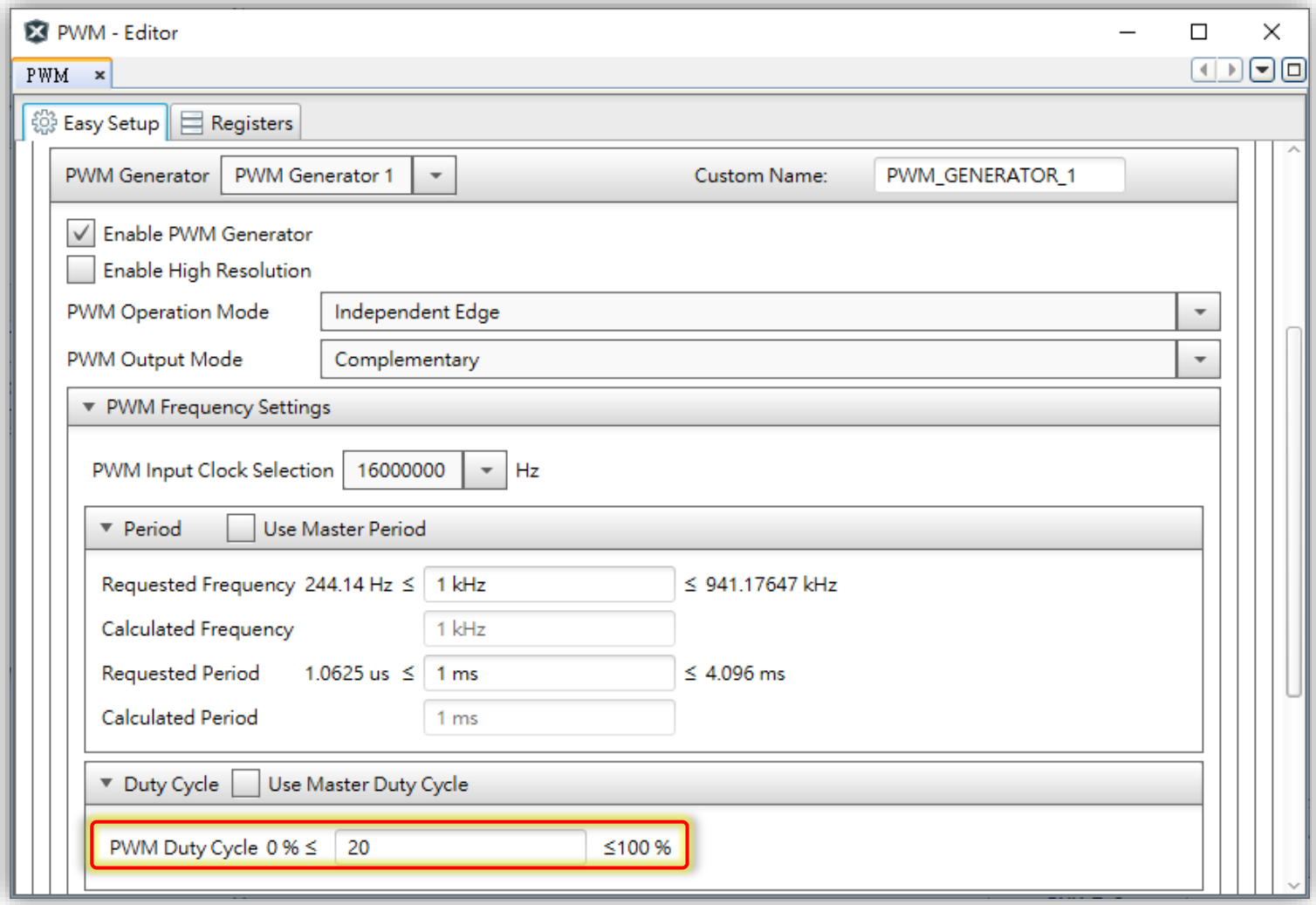
- Set PWM1 Frequency : 941.17647kHz > 1KHz.



Lab11 PWM Output

Step 5

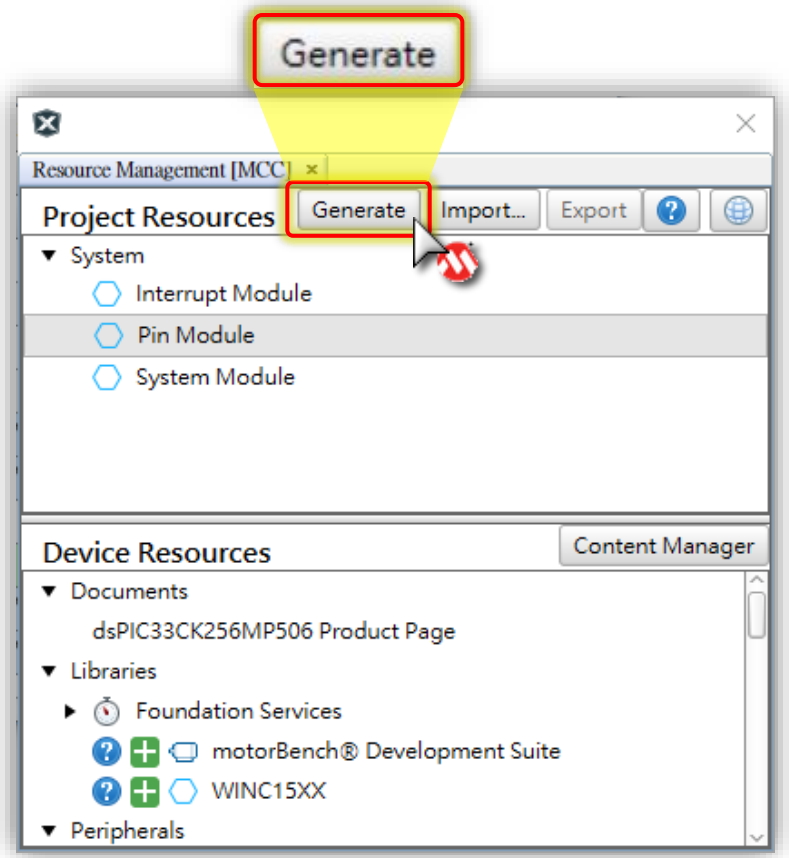
- Set PWM1 Duty Cycle : 0 > 20.

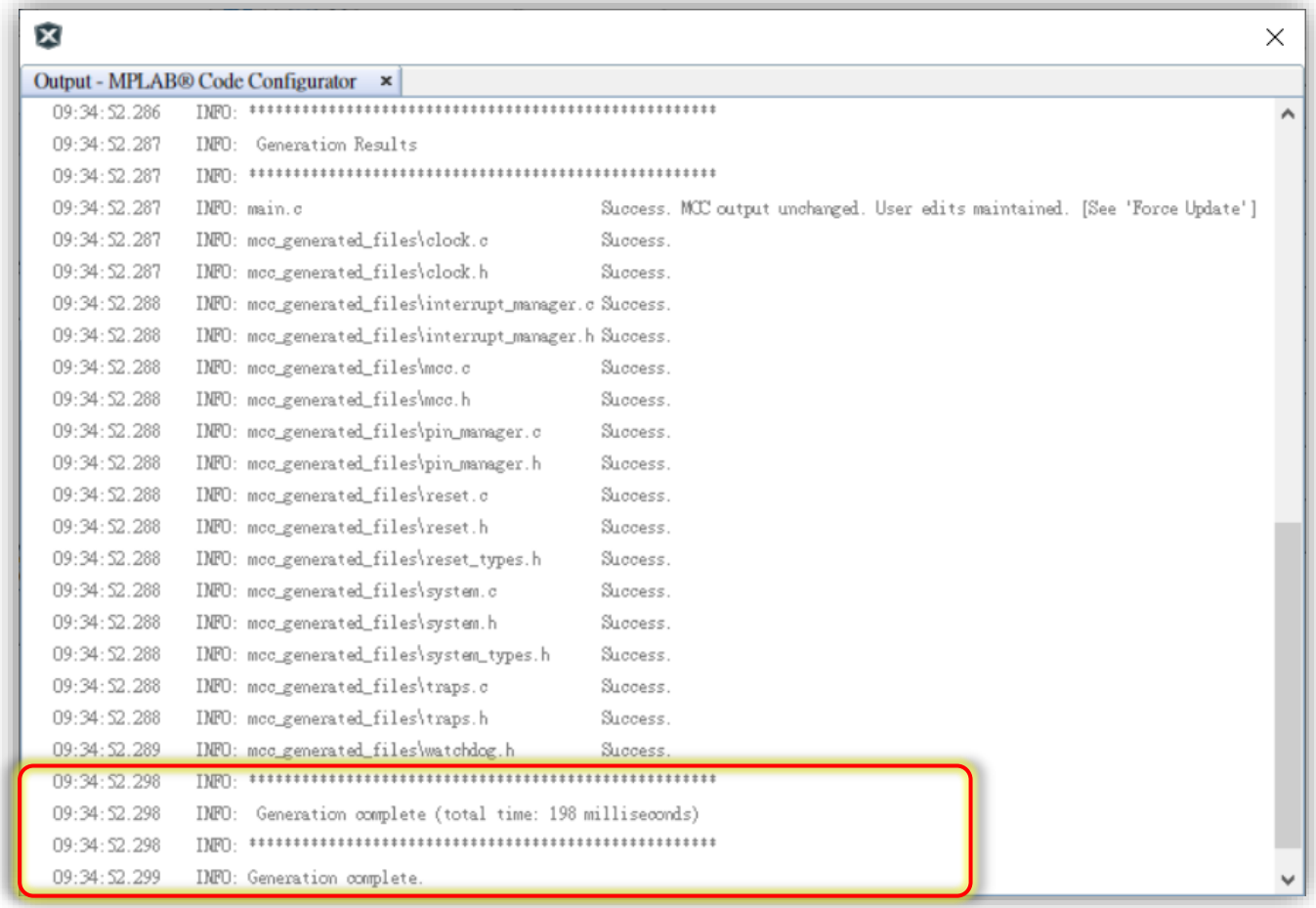


Lab11 PWM Output

Step 6

- Click "Generate" Button to generate your code.





Lab11 PWM Output

Code Example

```
#include "myOLEDlib/myOLED_ssd1306.h"
#include "mcc_generated_files/adc1.h"

...
volatile uint8_t SCCP1Flag = 0;
volatile uint8_t VR1Flag = 0;
volatile uint16_t ADC1_VR1_Buffer = 0;
volatile uint8_t VR2Flag = 0;
volatile uint16_t ADC1_VR2_Buffer;

int main(void)
{
    ...
}

...

void ADC1_VR1_CallBack (uint16_t adcVal)
{
    VR1Flag = 1;
    ADC1_VR1_Buffer = adcVal;
}

void ADC1_VR2_CallBack (uint16_t adcVal)
{
    VR2Flag = 1;
    ADC1_VR2_Buffer = adcVal;
}
```

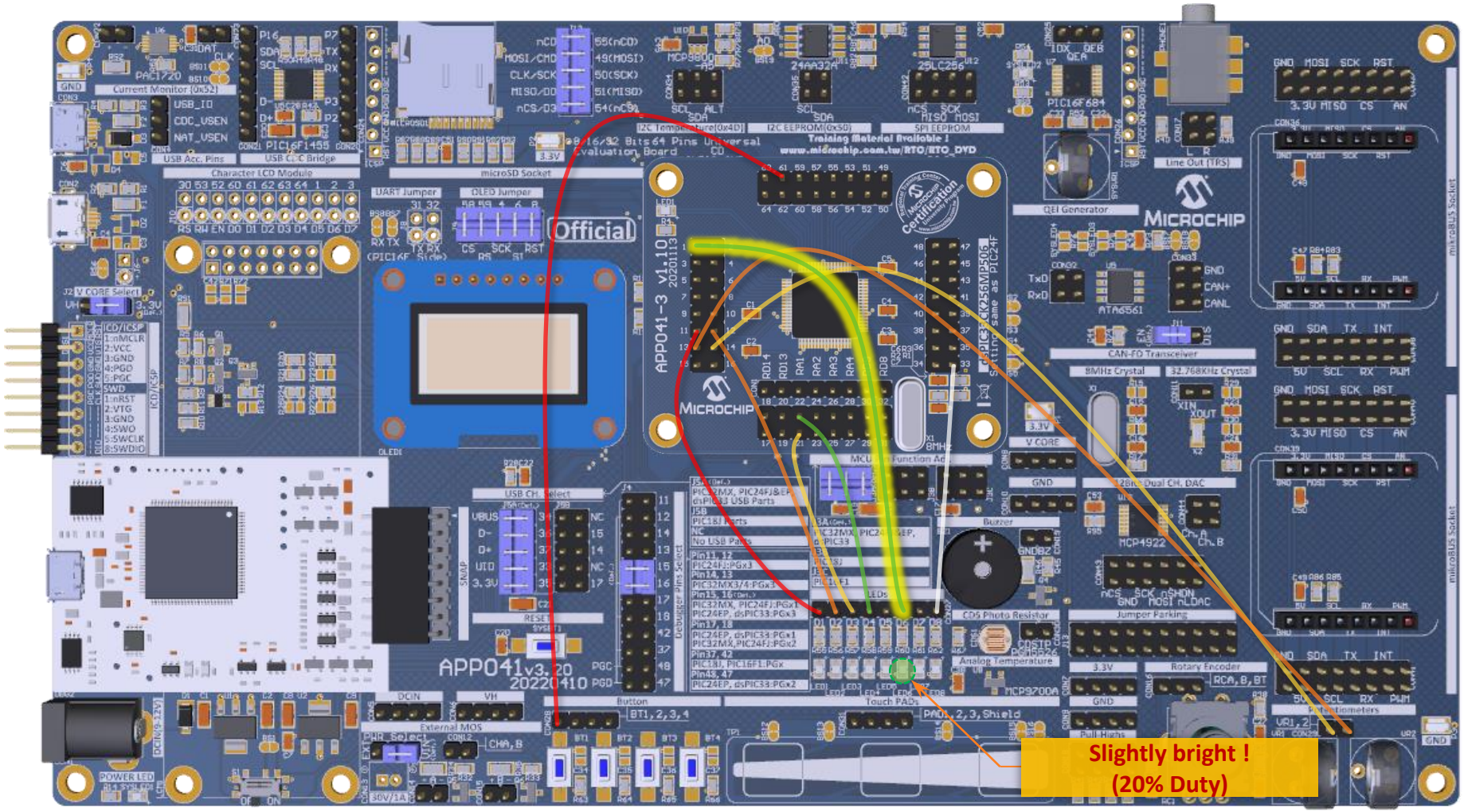
```
int main(void)
{
    ...
    while (1)
    {
        // Add your application code
        ...
        if (SCCP1Flag)
        {
            ...
            ADC1_SoftwareTriggerEnable();
        }
        if (VR1Flag)
        {
            VR1Flag = 0;
            sprintf((char *) StringBuffer, "VR1:%4d", ADC1_VR1_Buffer);
            myOLED_DrawStr(0, 0, StringBuffer);
        }

        if (VR2Flag)
        {
            VR2Flag = 0;
            sprintf((char *) StringBuffer, "VR2:%4d", ADC1_VR2_Buffer);
            myOLED_DrawStr(0, 1, StringBuffer);
        }

    }
    return 1;
}
```

Don't Need Modify Any Code !

MICROCHIP

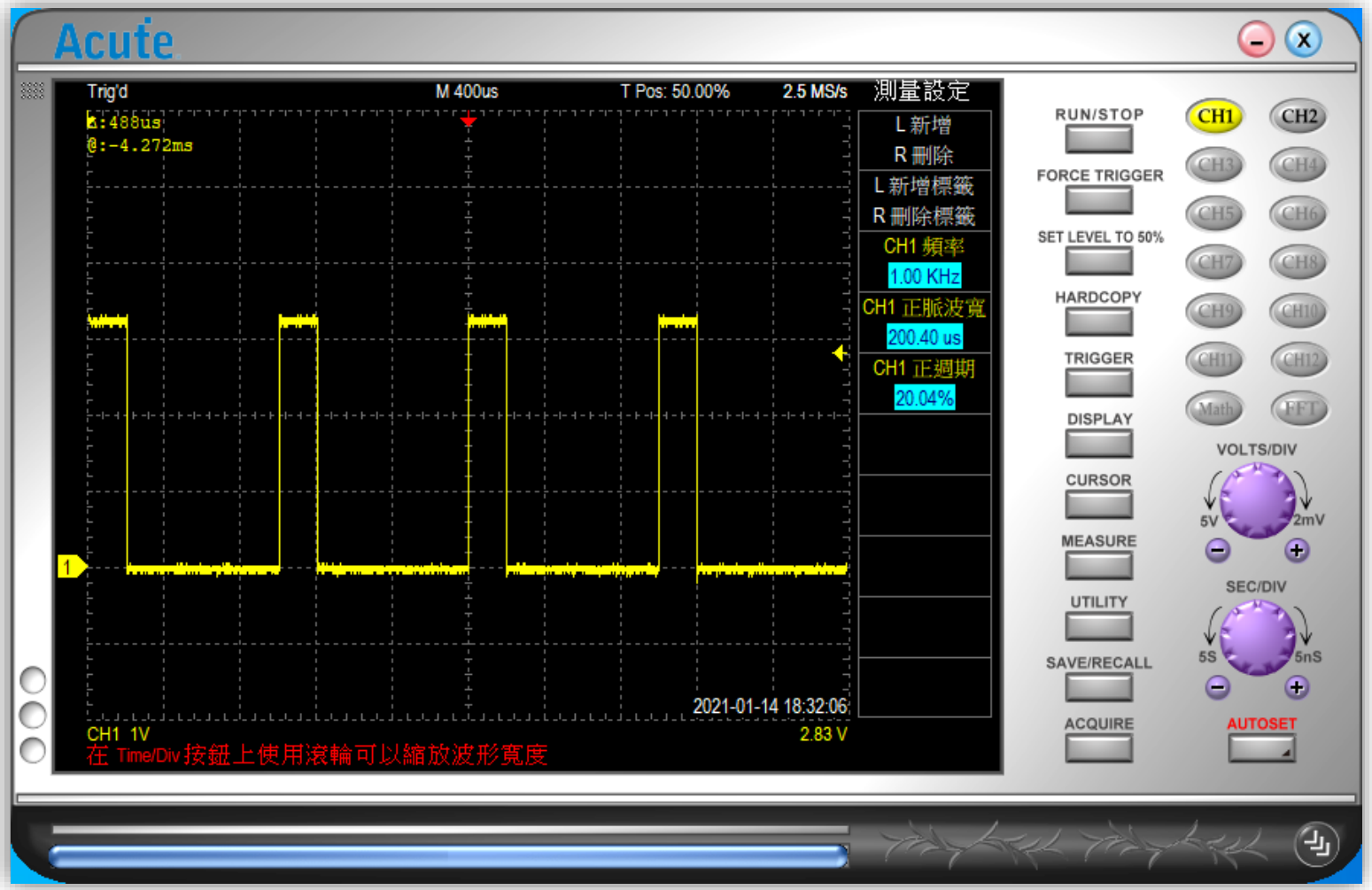
Lab11 PWM Output

Result



Lab11 PWM Output

Result



H.R. PWM Functions of MCC

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

  Prototype definition : "pwm.h"

  - void PWM_Initialize (void);
  - void PWM_GeneratorEnable(PWM_GENERATOR genNum);
  - void PWM_GeneratorDisable(PWM_GENERATOR genNum);
  - void PWM_Enable(void);
  - void PWM_DIsable(void);
  - PWM_MasterPeriodSet(uint16_t masterPeriod);
  - void PWM_MasterDutyCycleSet(uint16_t masterDutyCycle);
  - void PWM_MasterPhaseSet(uint16_t masterPhase);
  - void PWM_PeriodSet(PWM_GENERATOR genNum,uint16_t period);
  - void PWM_DutyCycleSet(PWM_GENERATOR genNum,uint16_t dutyCycle);
  - void PWM_PhaseSet(PWM_GENERATOR genNum,uint16_t phase);
  - void PWM_Generator*n*_CallBack(void);

Lab12 PWM Duty Adj By ADC

- Base on current project or Open .\Exams\Lab12_xxx.X to do exercises
- Try to use Potentiometer (VR1) to control LED (D6) brightness.
- VR Value (0 ~4095) <-> PWM Duty (0% ~ 100%)
- H.R. PWM Setting same as Lab11.

Let’s go!

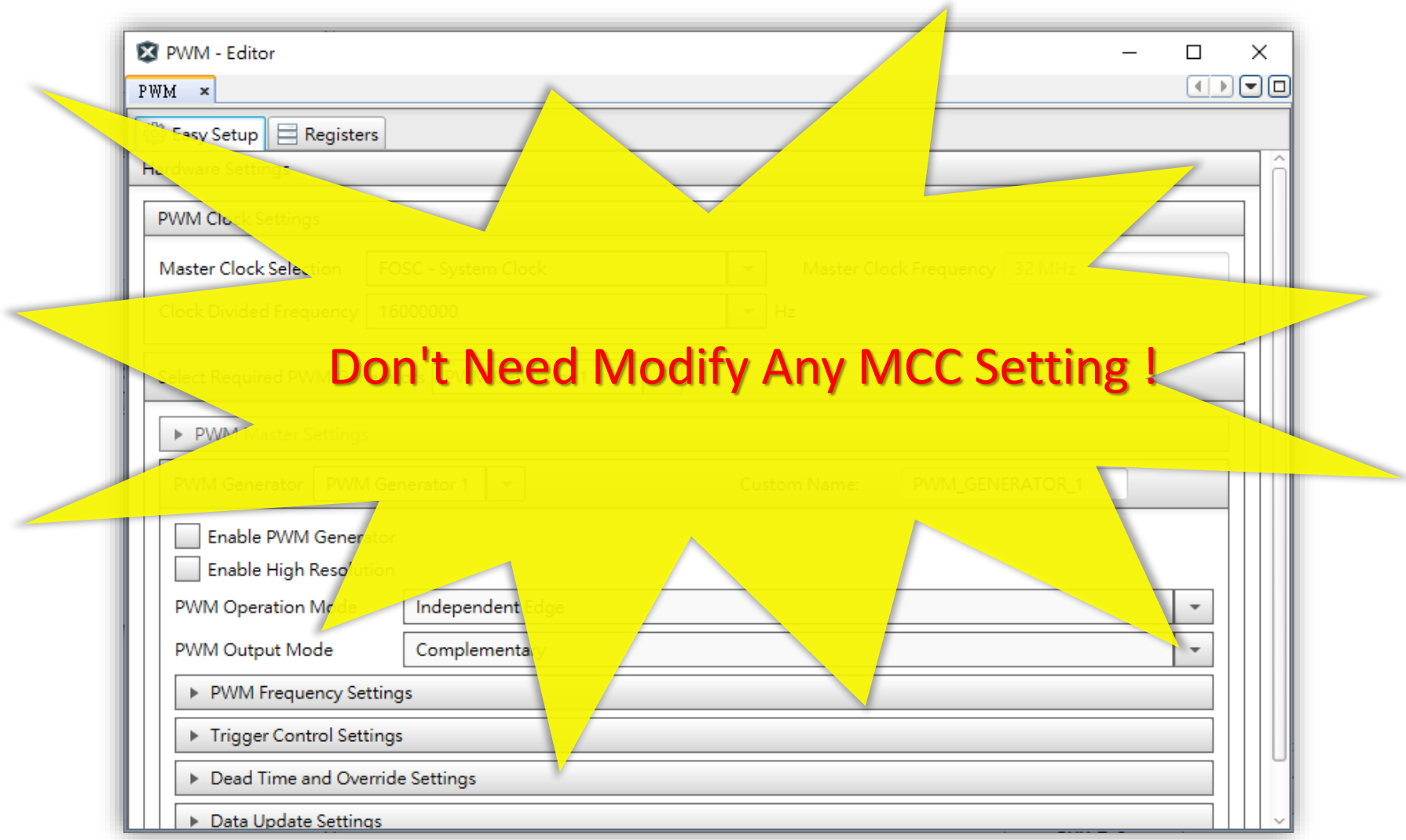
MICROCHIP

Lab12 PWM Duty Adj By ADC

Connection

Lab12 PWM Duty Adj By ADC

Notice



Lab12 PWM Duty Adj By ADC

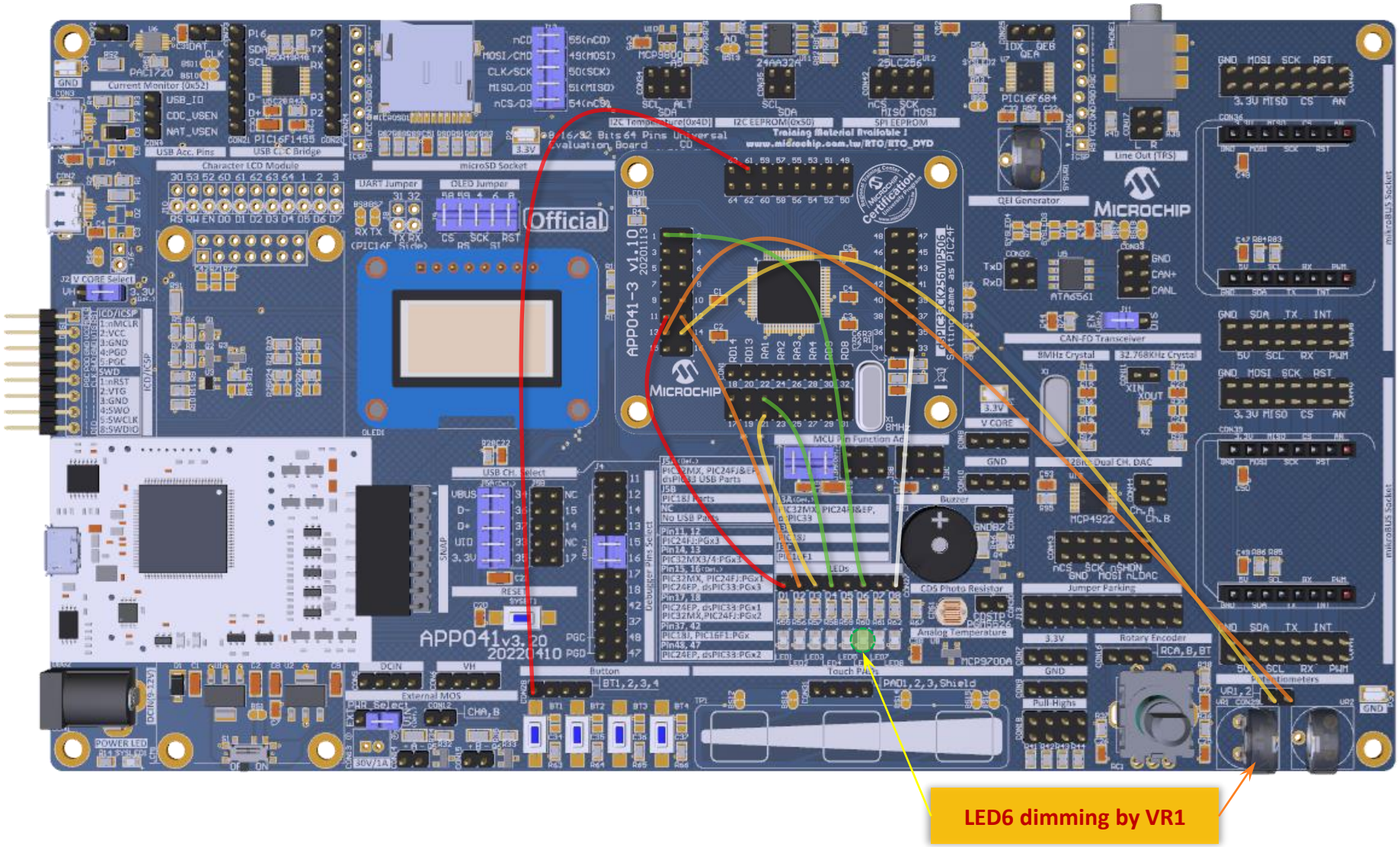
Code Example

```
#include "mcc_generated_files/adc1.h"
#include "mcc_generated_files/pwm.h"
…

int main(void)
{
    …
    while (1)
    {
        // Add your application code
        …

        if (VR1Flag)
        {
            VR1Flag = 0;

            sprintf((char *) StringBuffer, "VR1:%4d", ADC1_VR1_Buffer);
            myOLED_DrawStr(0, 0, StringBuffer);

            PWM_DutyCycleSet(PWM_GENERATOR_1,((uint32_t) ADC1_VR1_Buffer * PG1PER) / 4095);
            PWM_SoftwareUpdateRequest(PWM_GENERATOR_1);

        }
    }
    return 1;
}
```

MICROCHIP

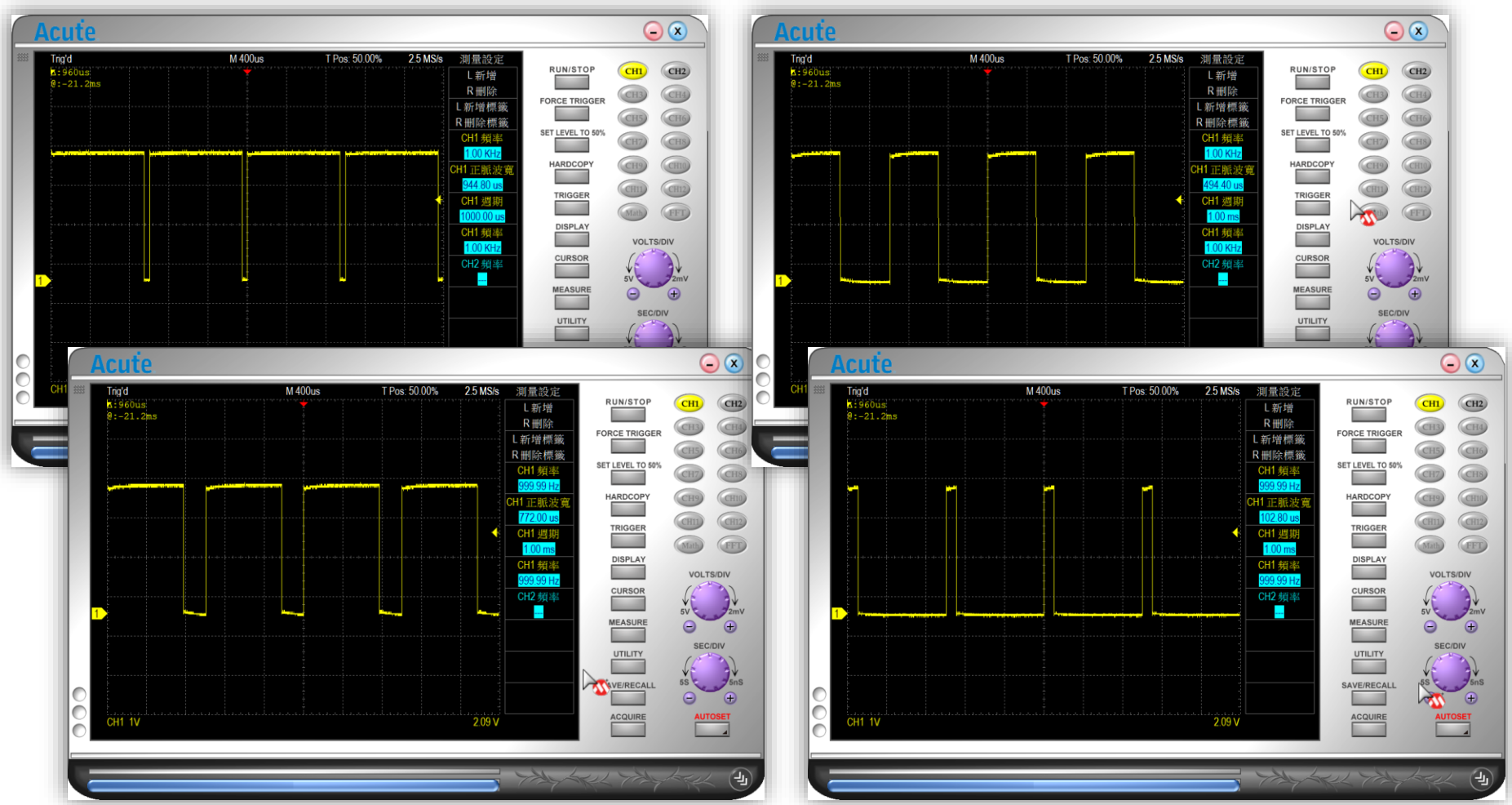
Lab12 PWM Duty Adj By ADC

Result



Lab12 PWM Duty Adj By ADC

Result



Lab13 PWM Buzzer Tone

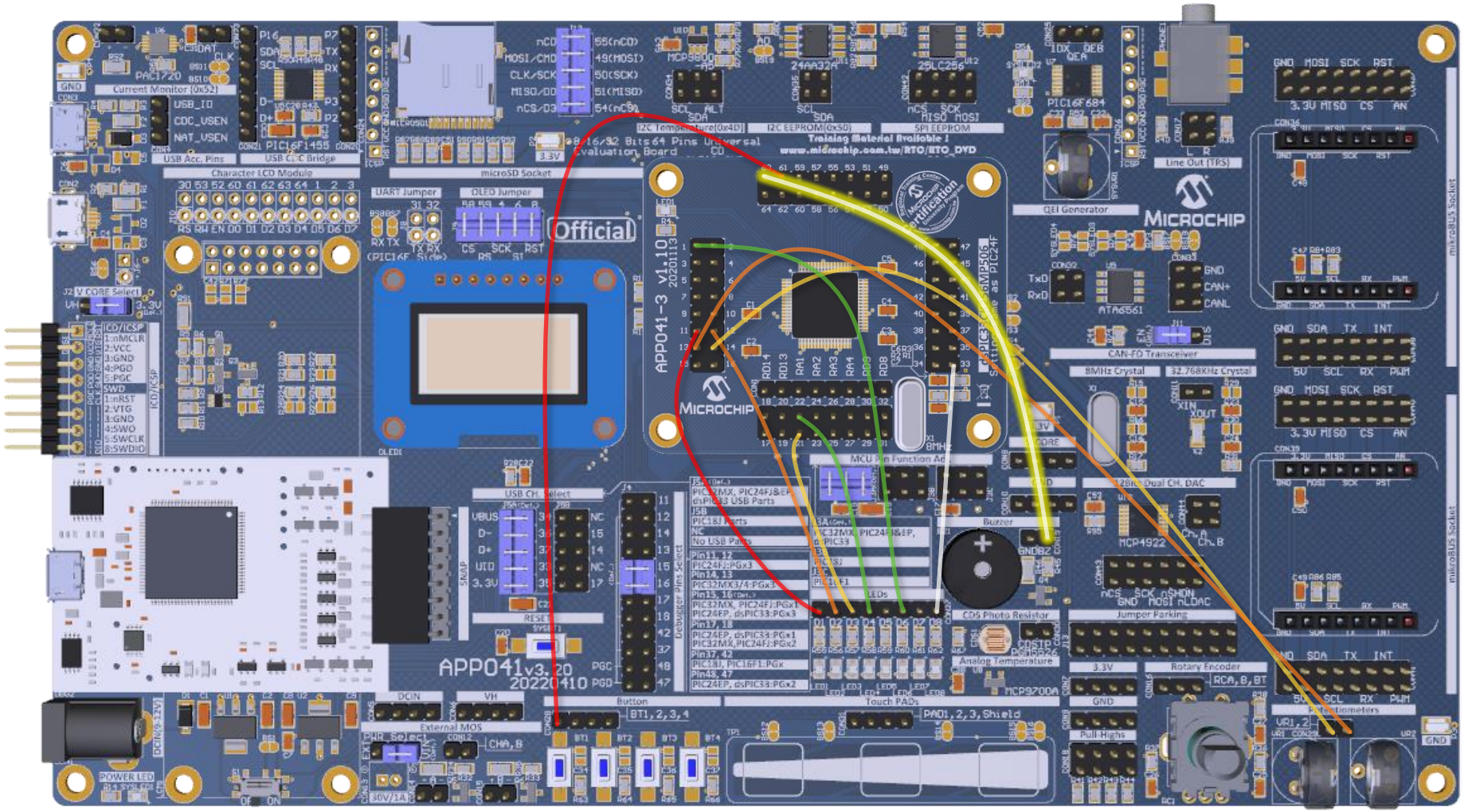
- Base on current project or Open .\Exams\Lab13_xxx.X to do exercises
- Try to initial H.R. PWM - PG2 to output special frequency (Tone) to Buzzer by VR2.
- VR2 Value (0 ~4,095) <-> PWM Frequency (0Hz ~ 4,095Hz)
  - Target Frequency = PWM Source Frequency / 16bits Period Count
- Enable H.R. PWM – PG2 interrupt & connect.
  - RB12(PWM2_H, Pin63) to Buzzer(BZ)

Let’s go!

MICROCHIP

Lab13 PWM Buzzer Tone

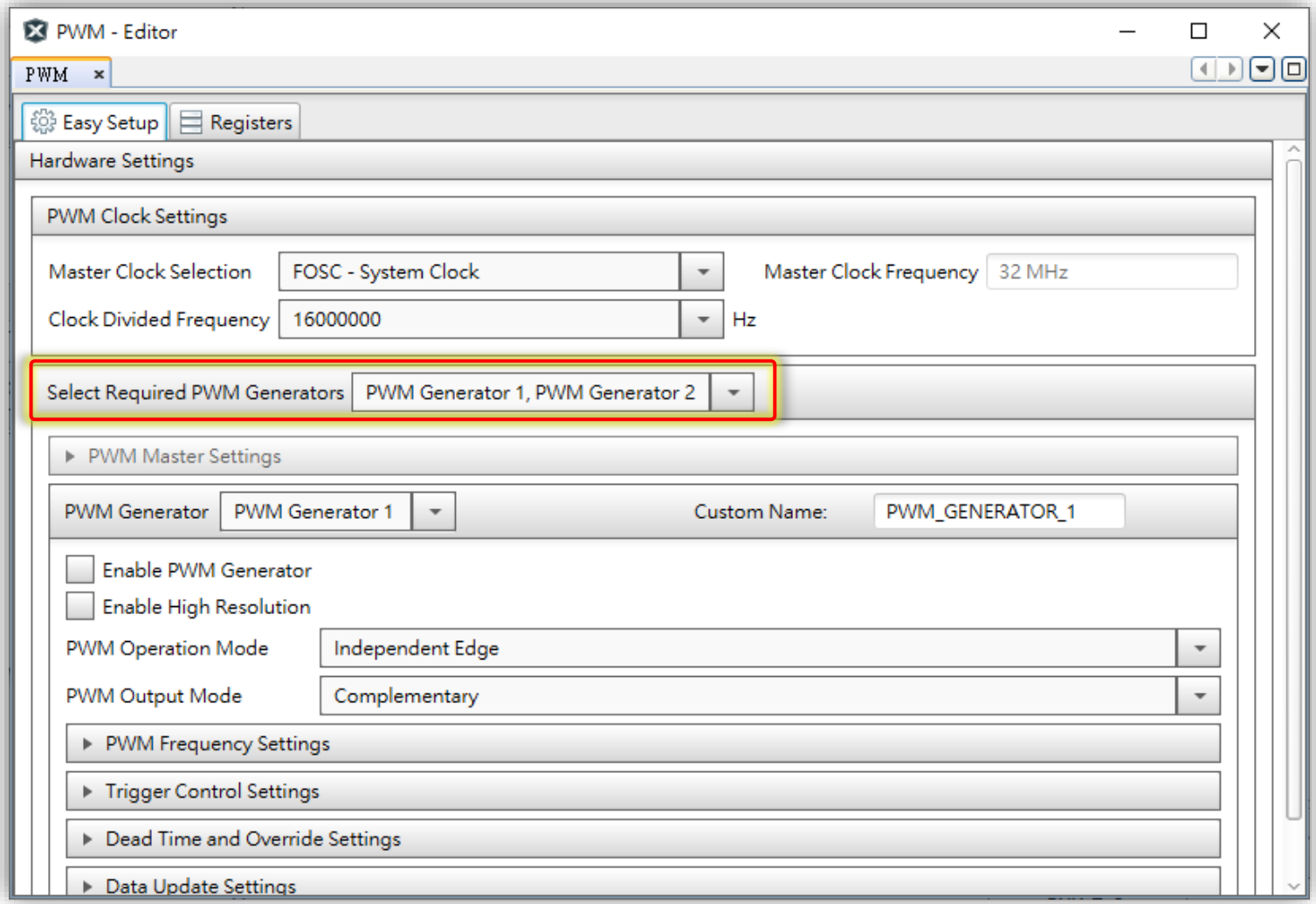
Connection



Lab13 PWM Buzzer Tone

Step 1

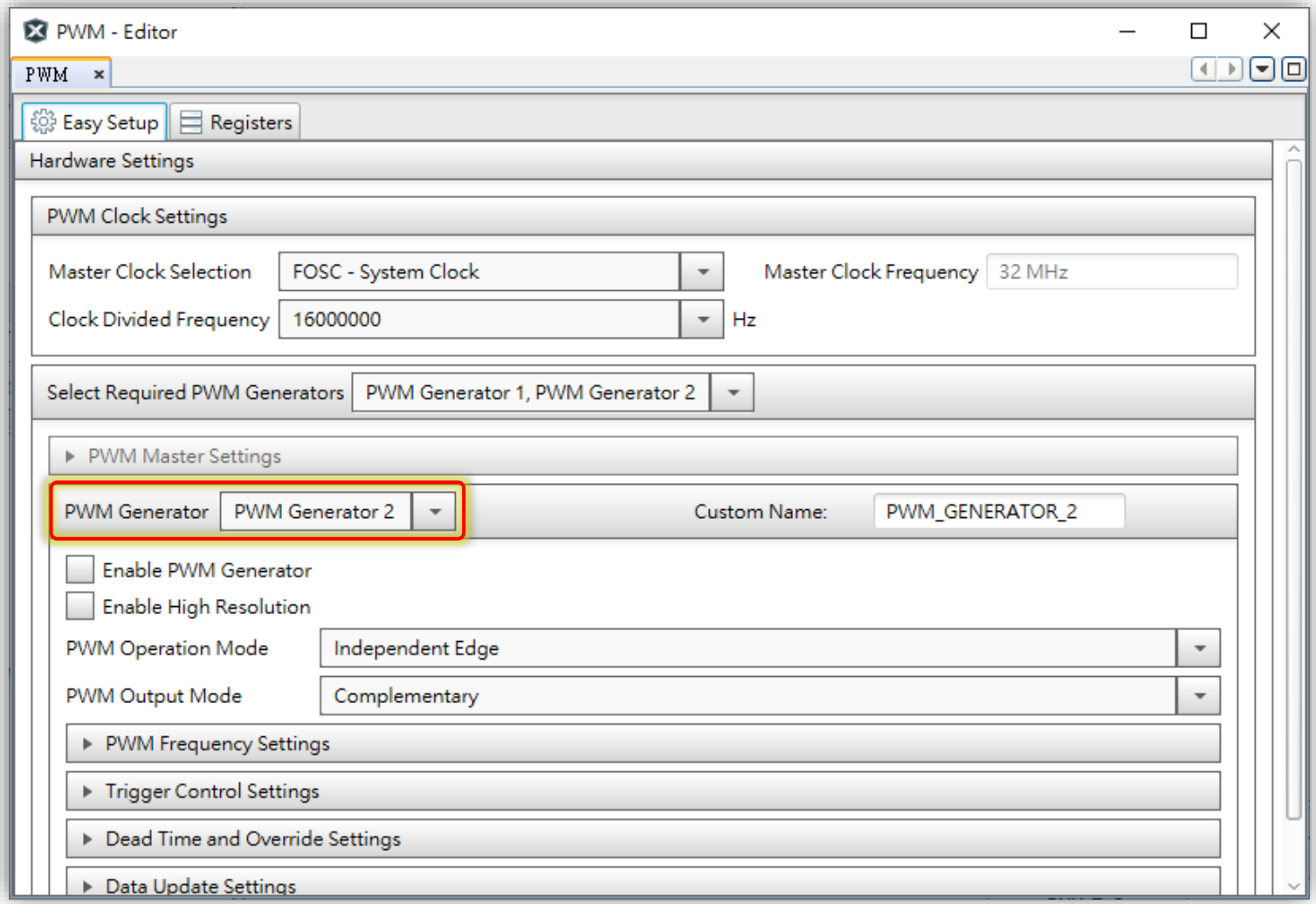
- Set Enable PWM Generator 2.



Lab13 PWM Buzzer Tone

Step 2

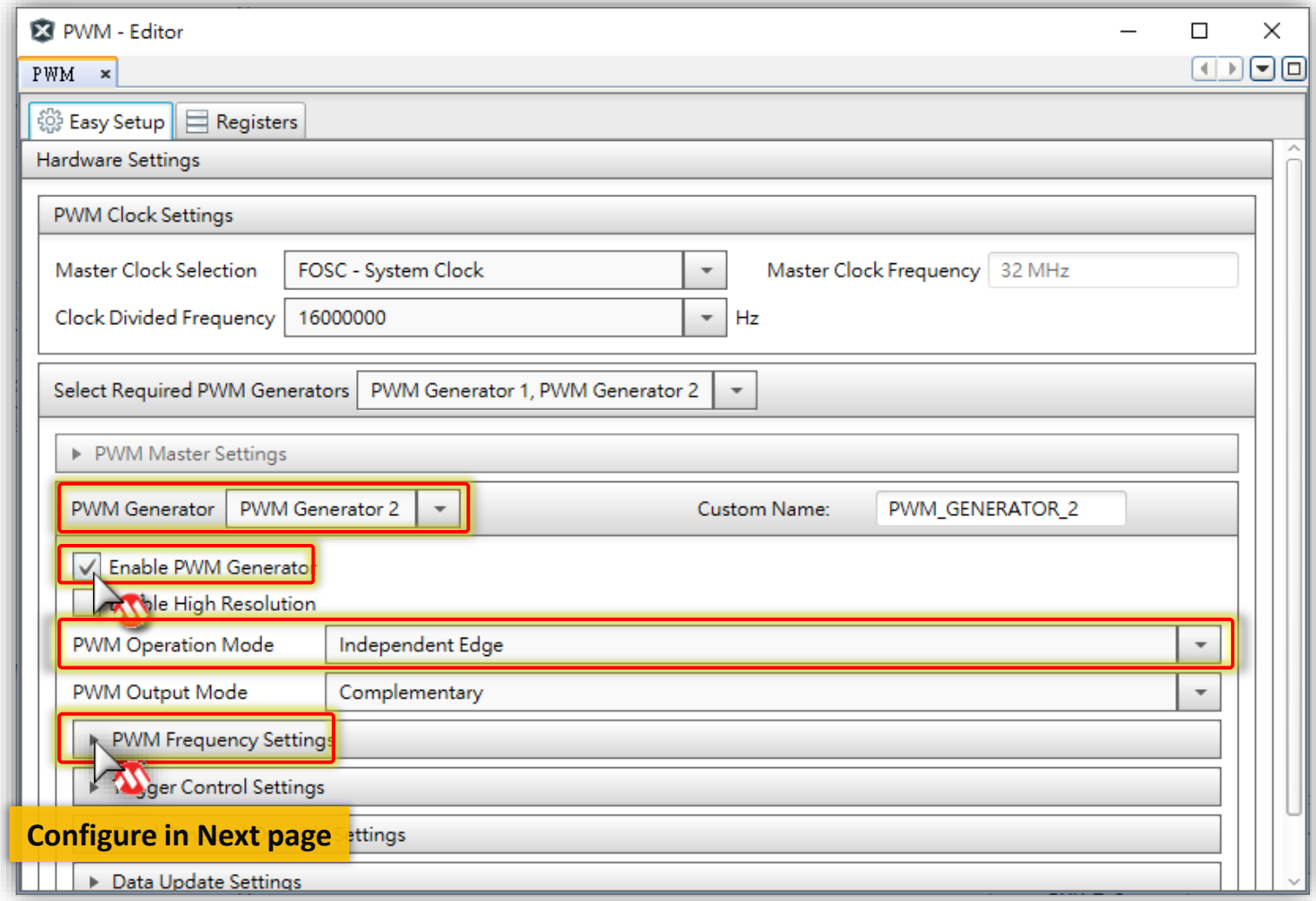
- Set PWM Generator : PWM Generator 1 > PWM Generator 2.



Lab13 PWM Buzzer Tone

Step 3

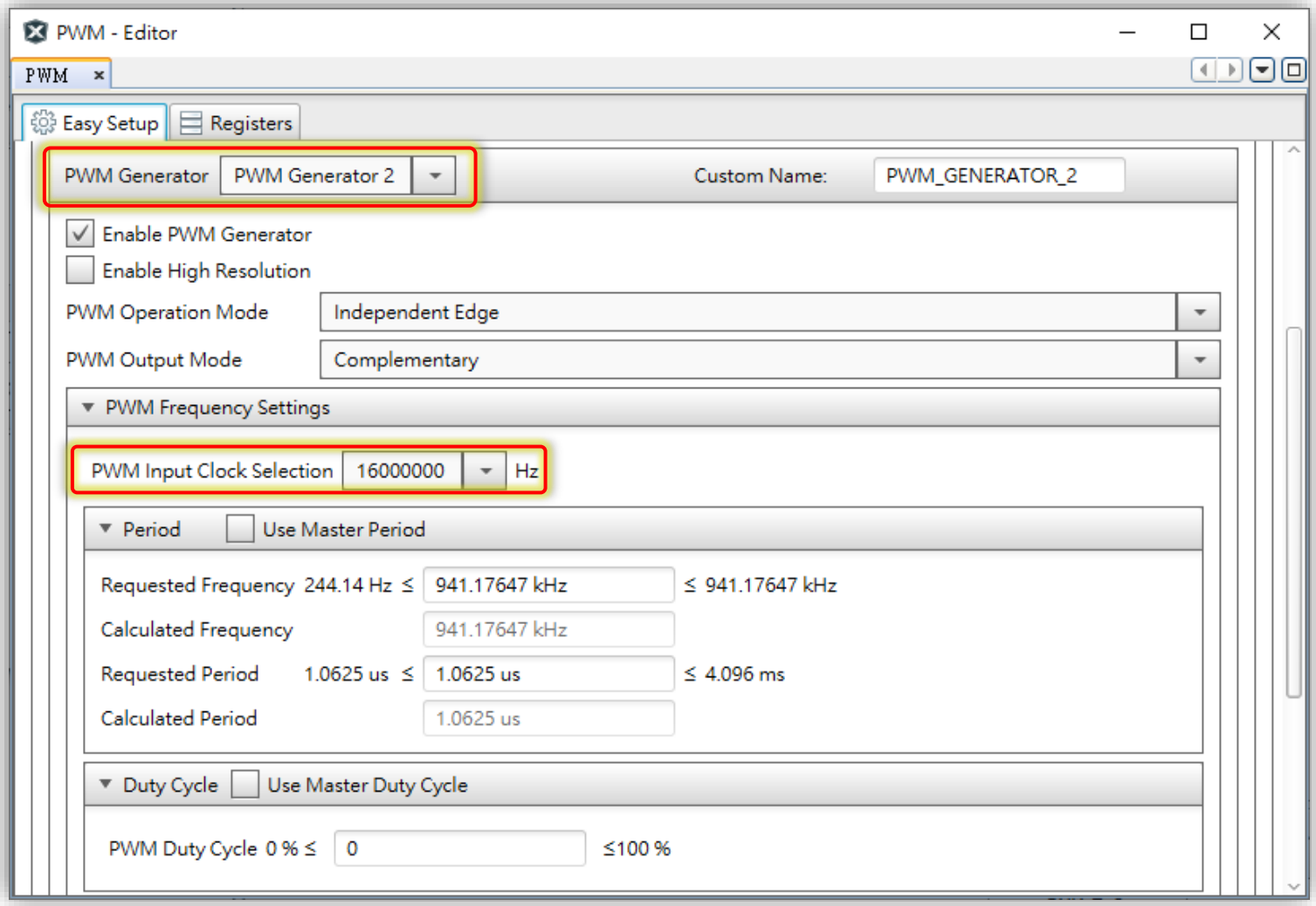
- Set Enable PWM Generator 2 and Confirm PWM Mode : Independent Edge.



Lab13 PWM Buzzer Tone

Step 4

- Set PWM2 Clock Selection : 32000000 > 16000000.

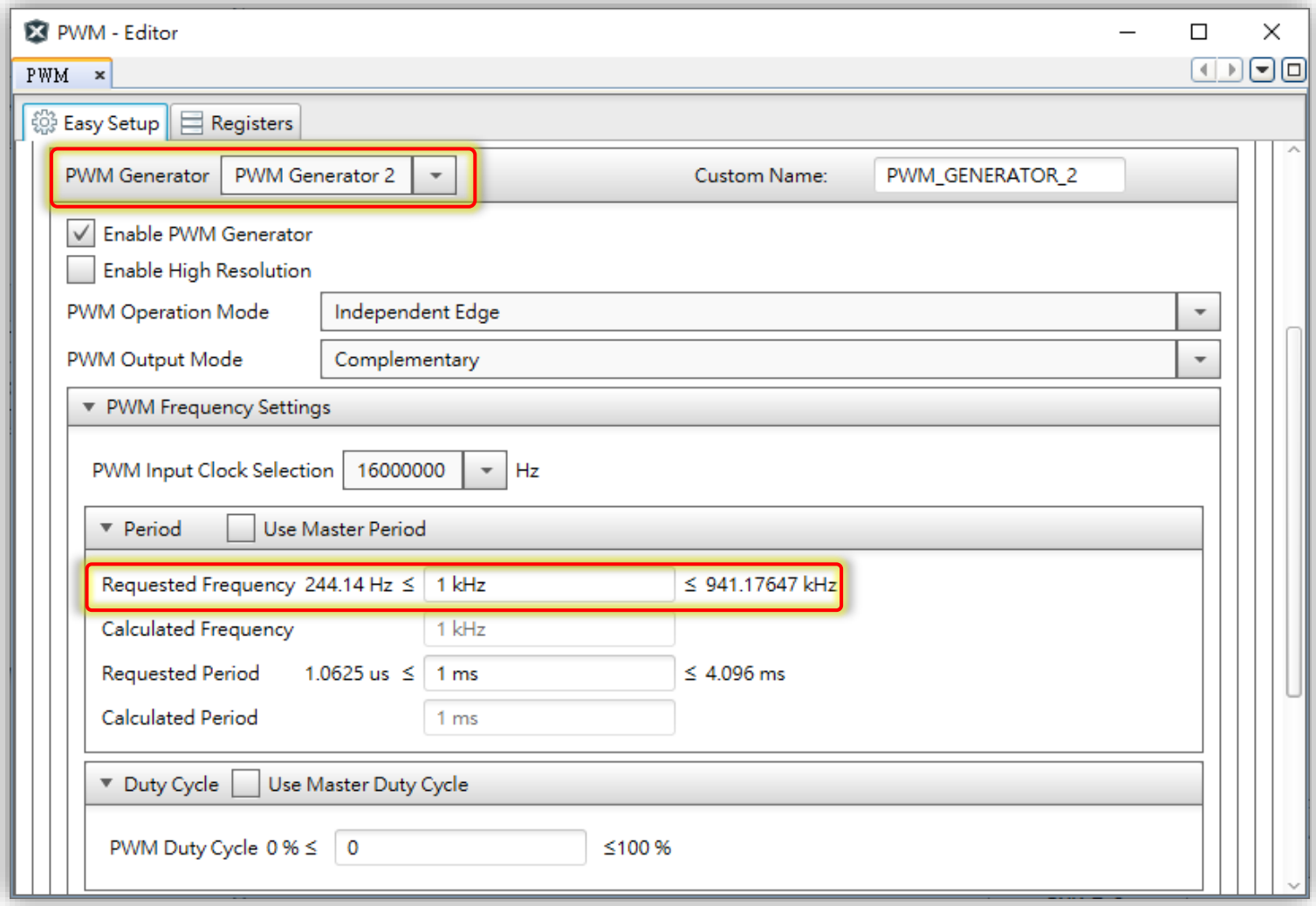


MICROCHIP

Lab13 PWM Buzzer Tone

Step 5

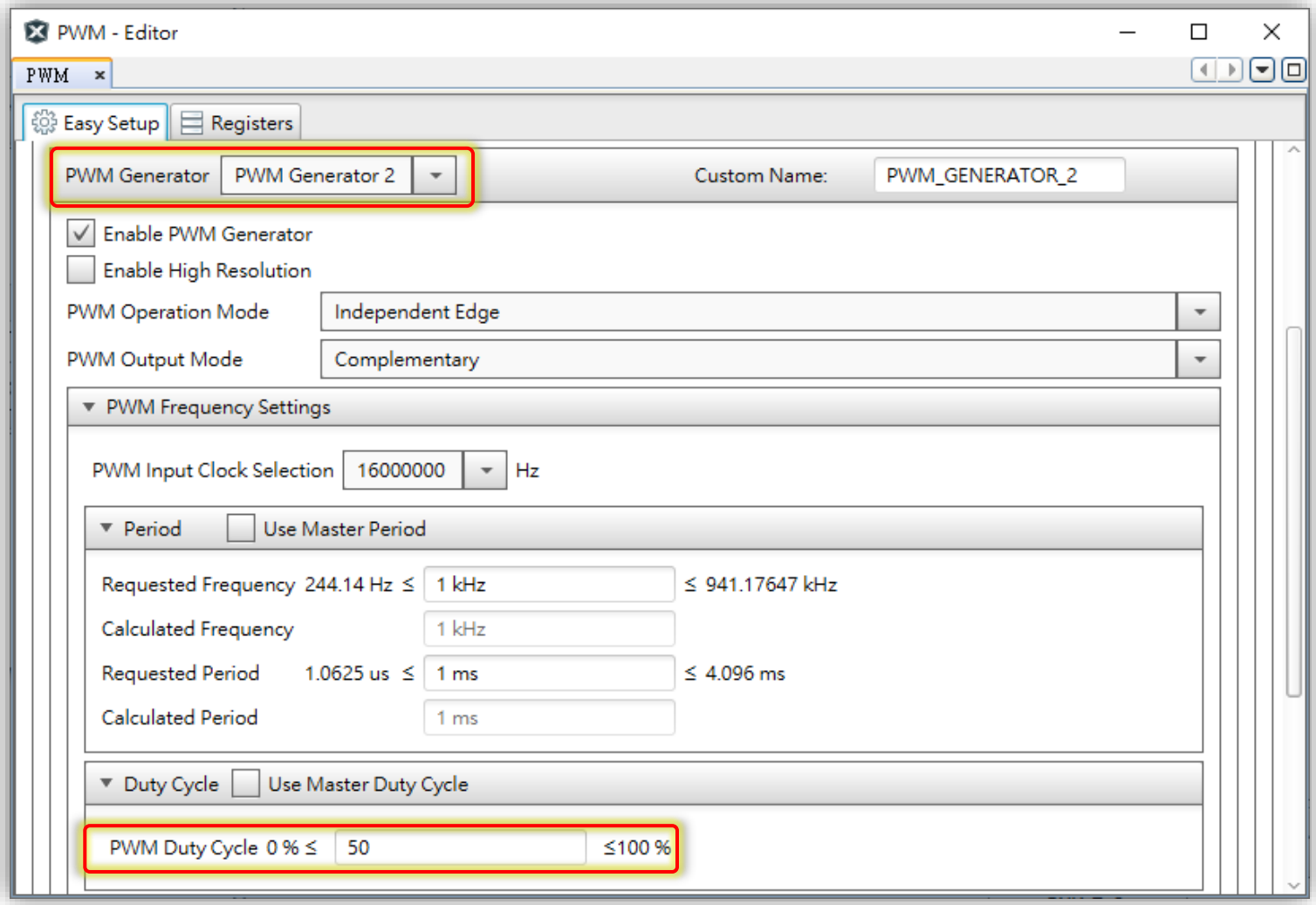
- Set PWM2 Frequency : 941.17647kHz > 1kHz.



Lab13 PWM Buzzer Tone

Step 6

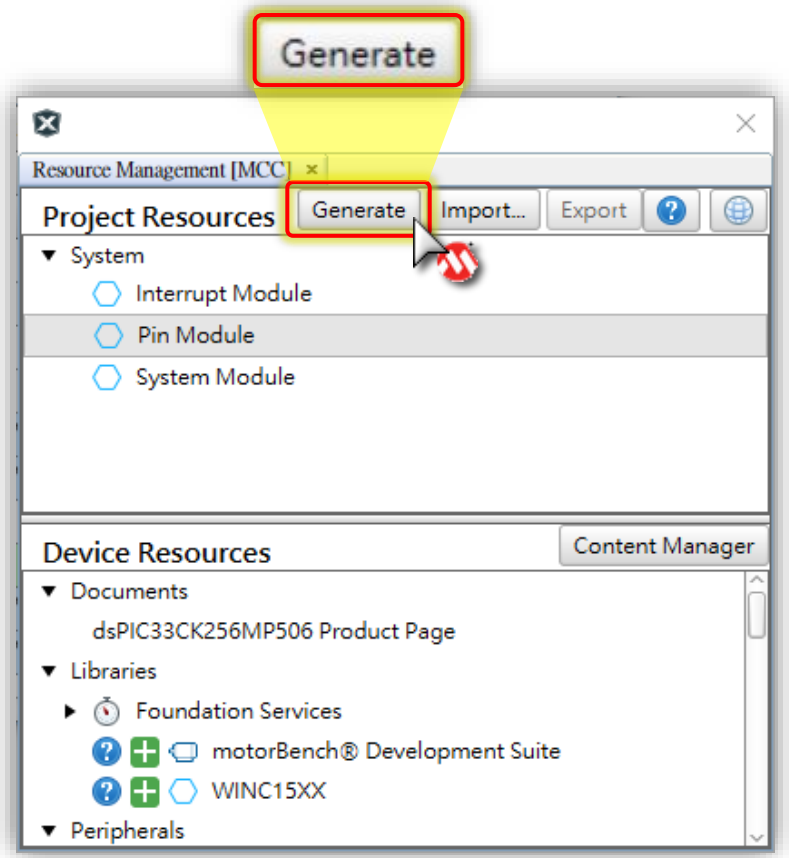
- Set PWM2 Duty Cycle : 0 > 50.

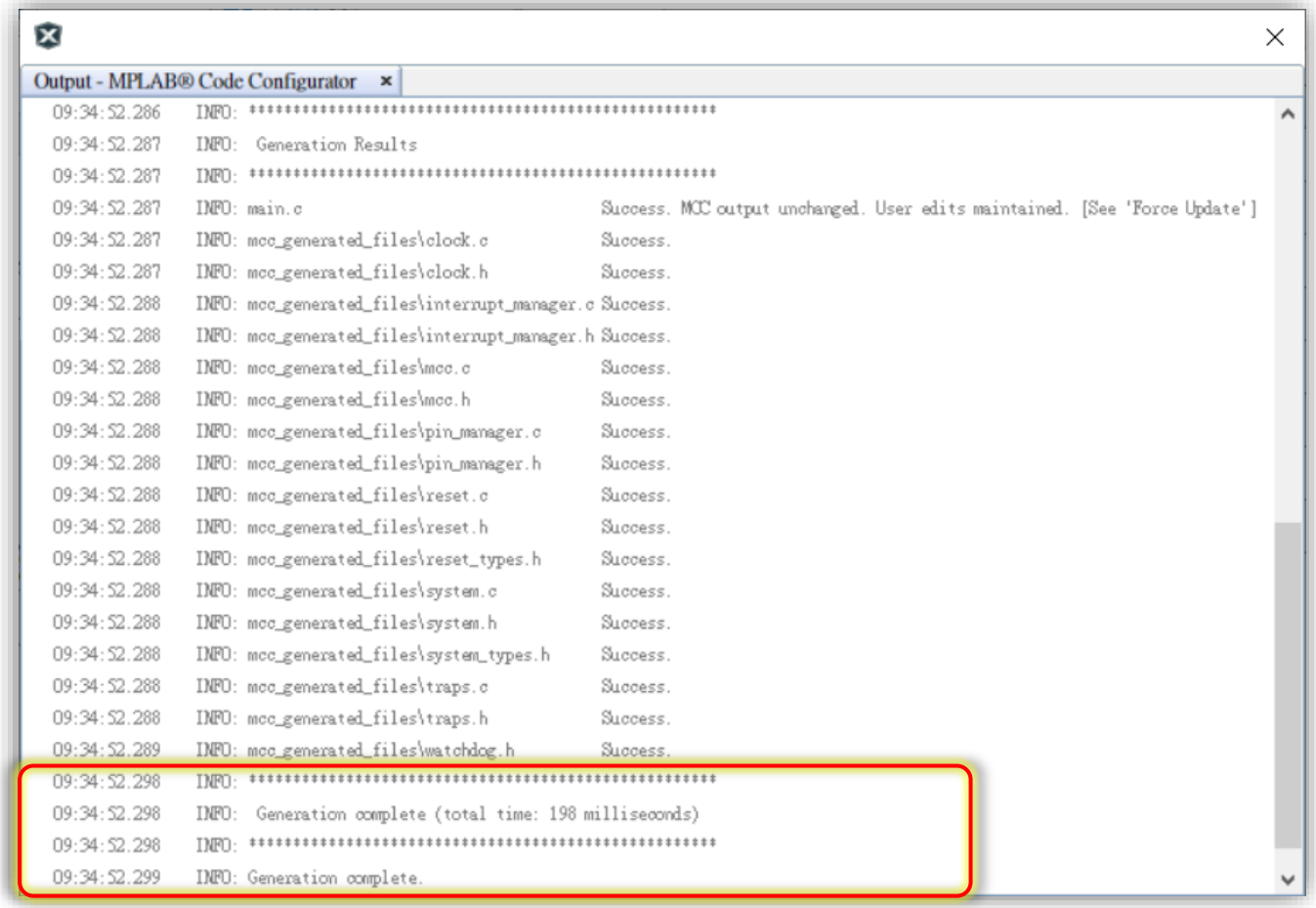


Lab13 PWM Buzzer Tone

Step 7

- Click "Generate" Button to generate your code.





Lab13 PWM Buzzer Tone

Code Example

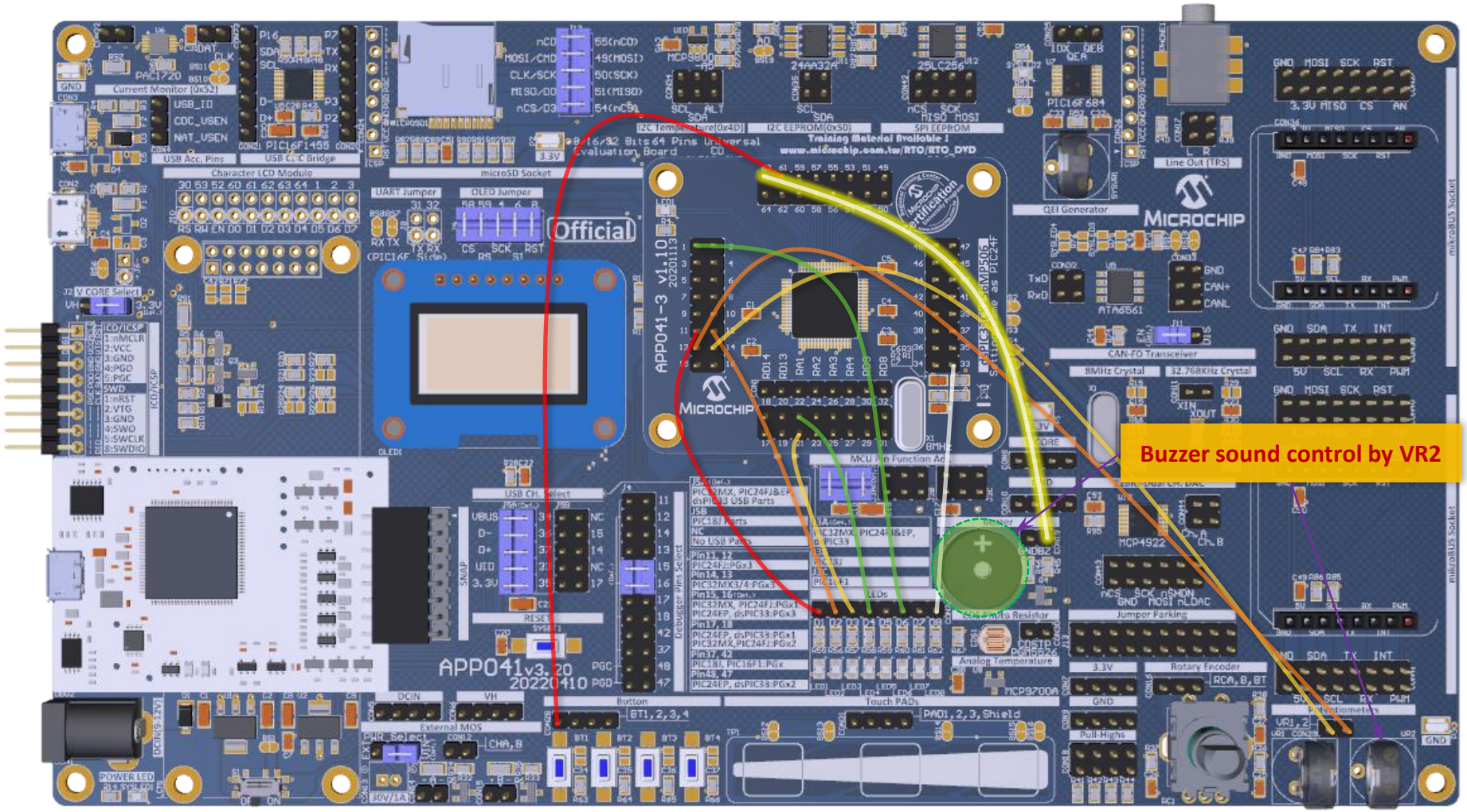
- **VR2 Value(0 ~ 4,095) <-> PWM Frequency (244.14Hz ~ 4,095Hz)**
  - **PWM Frequency (244.14Hz) : PGxPER = 0xFFFF**
  - **PWM Frequency (4095Hz) : PGxPER = 0x0F43 (16M/4095)**

```
#include "mcc_generated_files/adc1.h"
#include "mcc_generated_files/pwm.h"
…

int main(void)
{
    …
    while (1)
    {
        // Add your application code
        …

        if (VR2Flag)
        {
            VR2Flag = 0;
            sprintf((char *) StringBuffer, "VR2:%4d", ADC1_VR2_Buffer);
            myOLED_DrawStr(0, 2, StringBuffer);

            PWM_PeriodSet(PWM_GENERATOR_2, 0xF43+((uint32_t)(0xFFFF-0xF43)*ADC1_VR2_Buffer/4095));
            PWM_DutyCycleSet(PWM_GENERATOR_2, PG2PER >> 1);
            PWM_SoftwareUpdateRequest(PWM_GENERATOR_2);
        }
    }
    return 1;
}
```

MICROCHIP

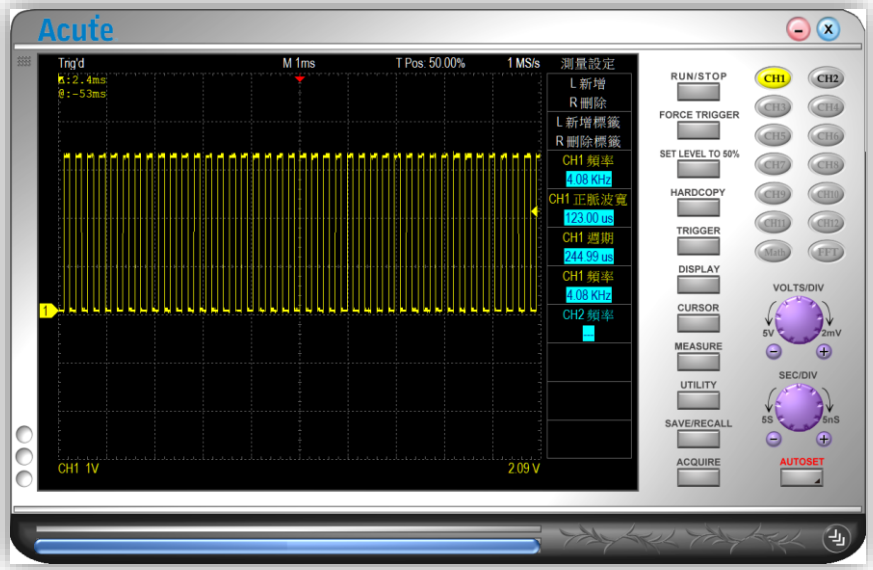
Lab13 PWM Buzzer Tone

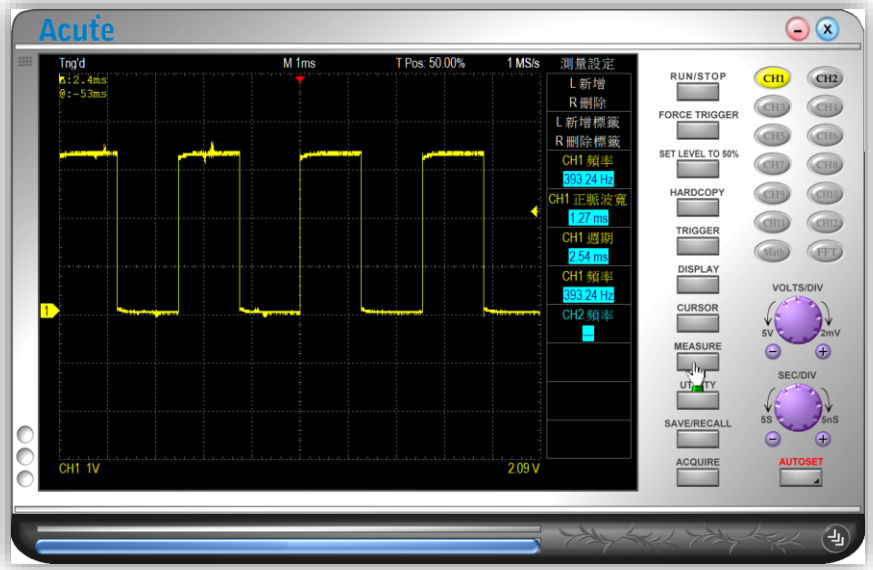
Result

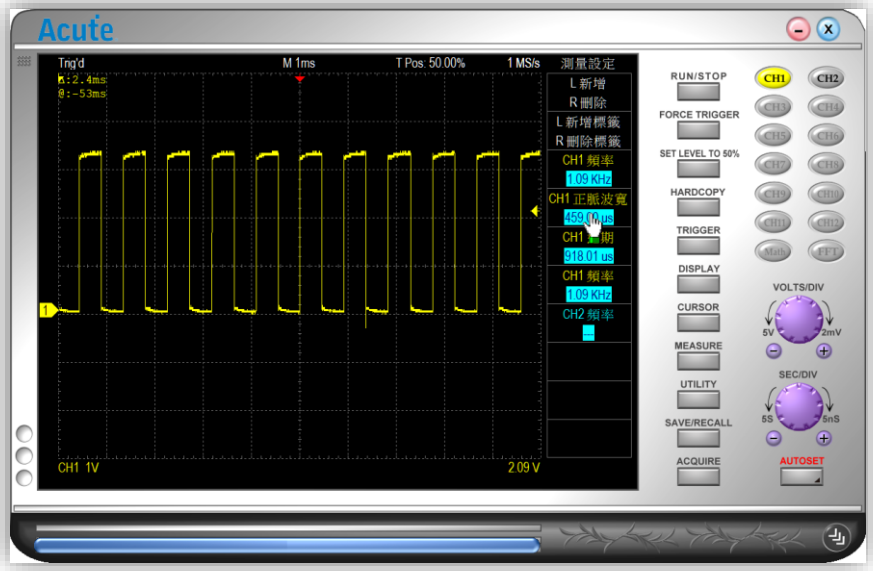


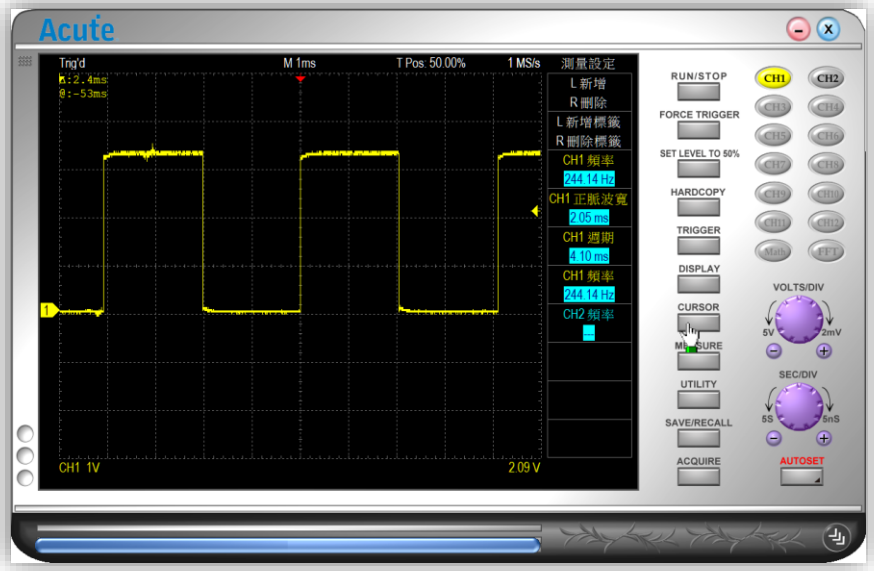
Lab13 PWM Buzzer Tone

Result







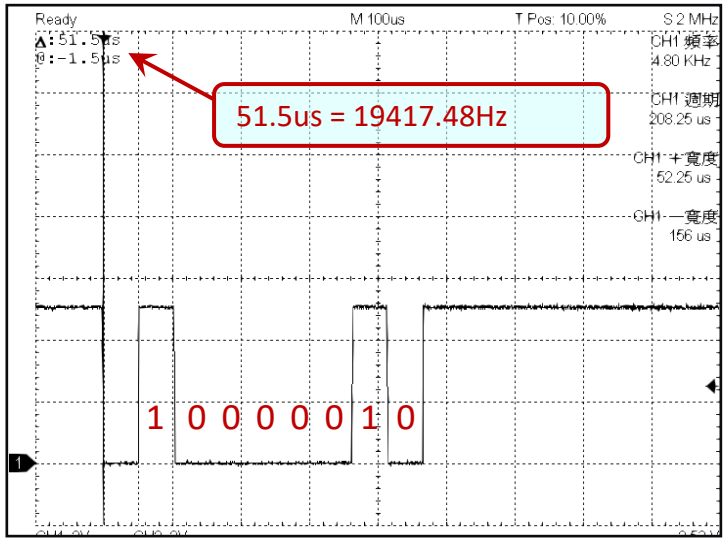


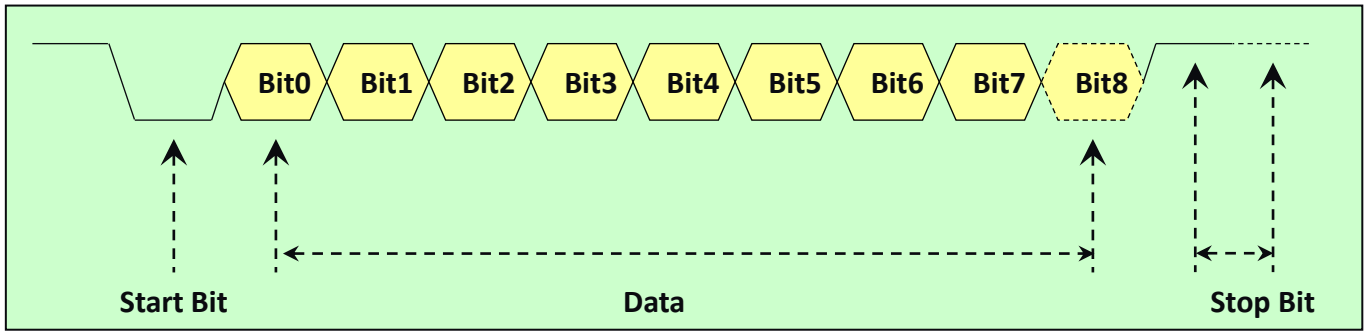
MICROCHIP

UART Application

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.

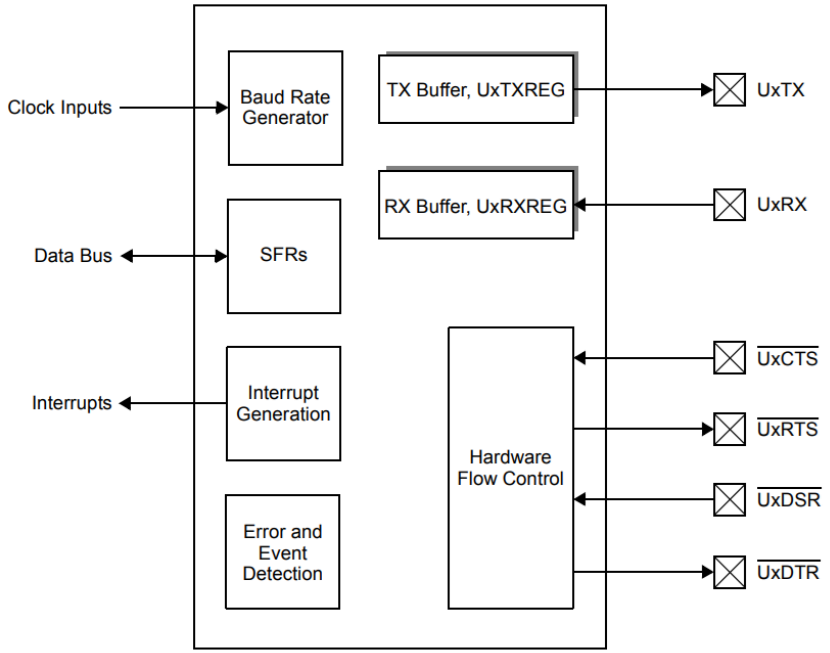




UART Transmit Example : 'A' (0x41), 19,200 bps

dsPIC33CK's UART

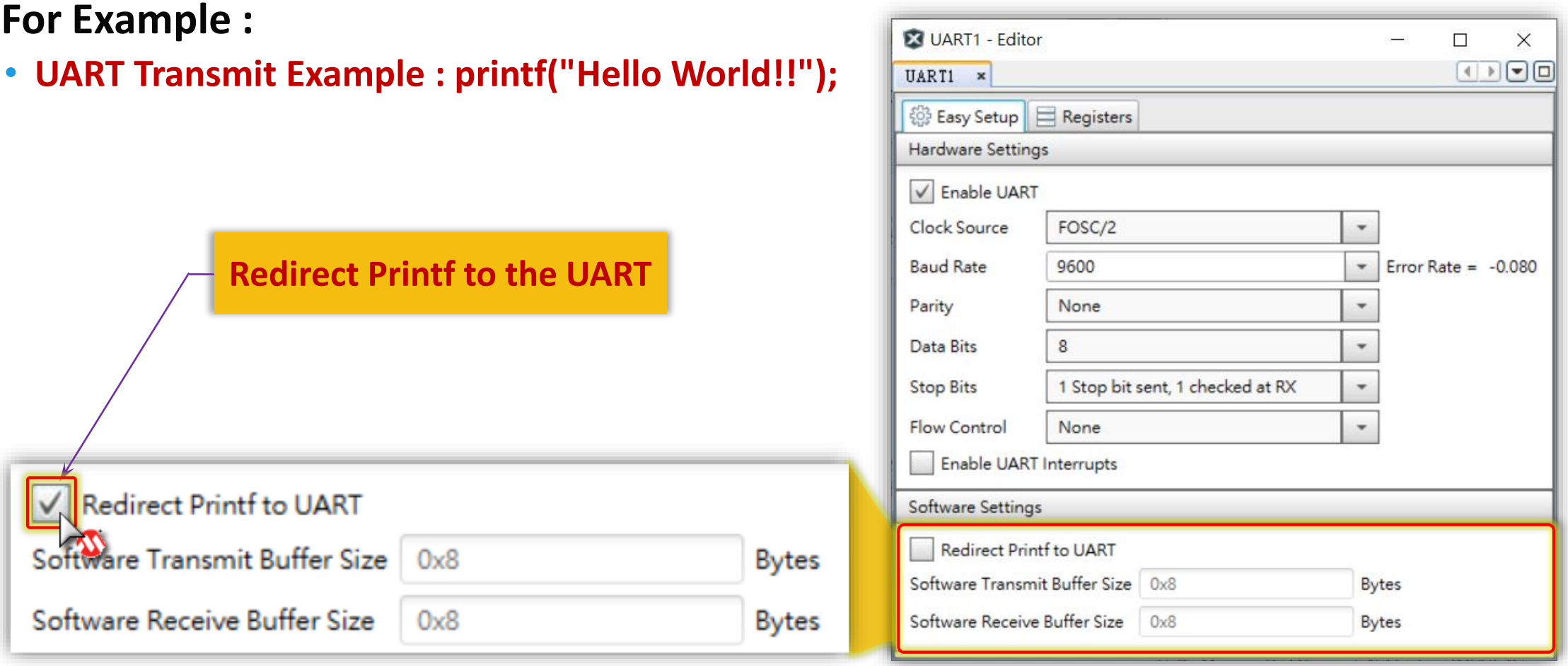
- UART module is full-duplex, asynchronous communication, Support RS-232, RS-485, IrDA and LIN.
- The module also supports hardware flow control, Parity check, Frame Error, Overflow 8 ~ 9 Data Bits.
- 8-Deep FIFO for TX & RX.
- Provide Loopback, Address Detect(RS-485), Auto Baud Detect(LIN bus).



MICROCHIP

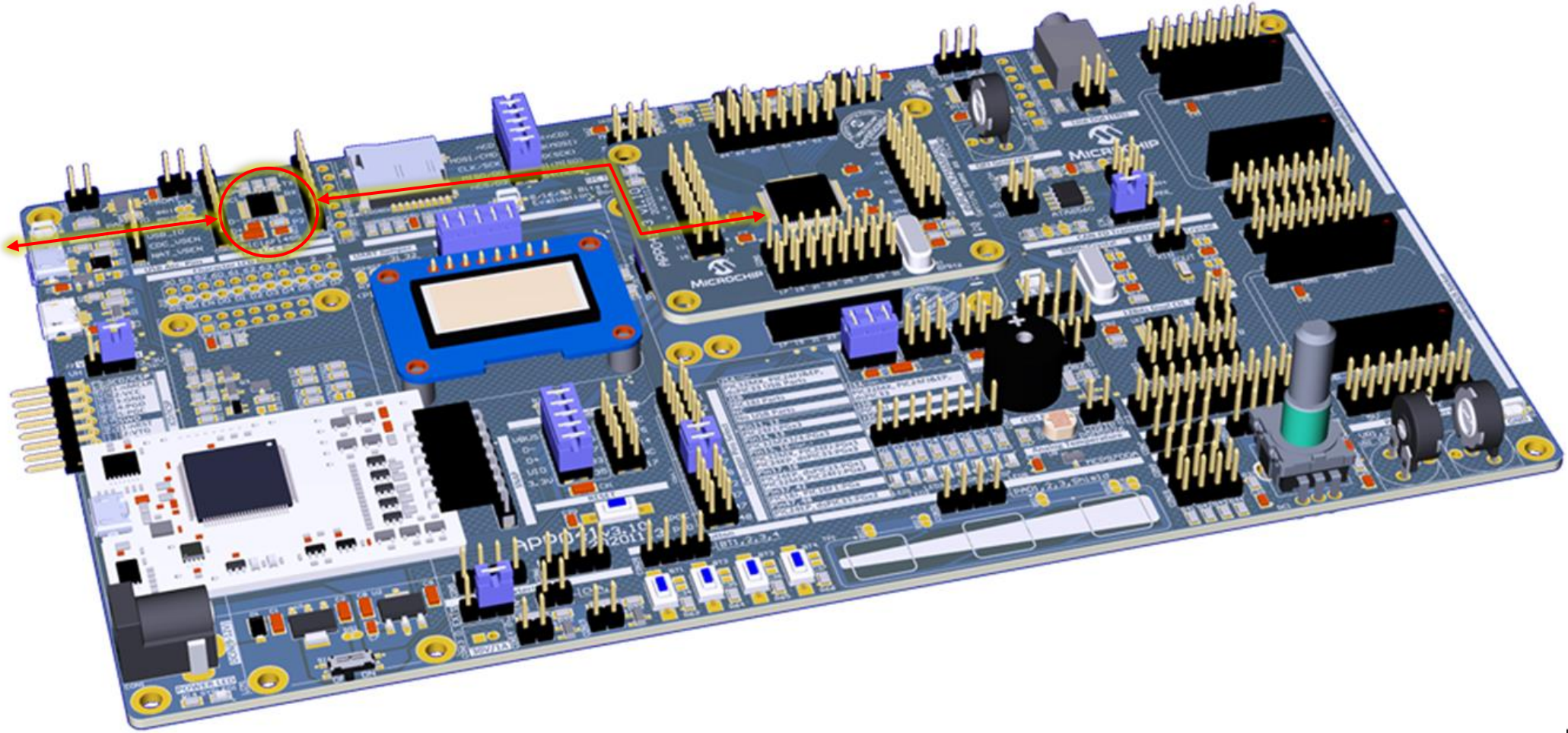
STDOUT Redirect

- 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!!");



USB CDC Emulator

- **The PIC16F1455 already preload USB Serial Emulator firmware (USB Virtual COM Port).**



Lab Preparation

PC Terminal Software Setting

MICROCHIP

Lab Preparation

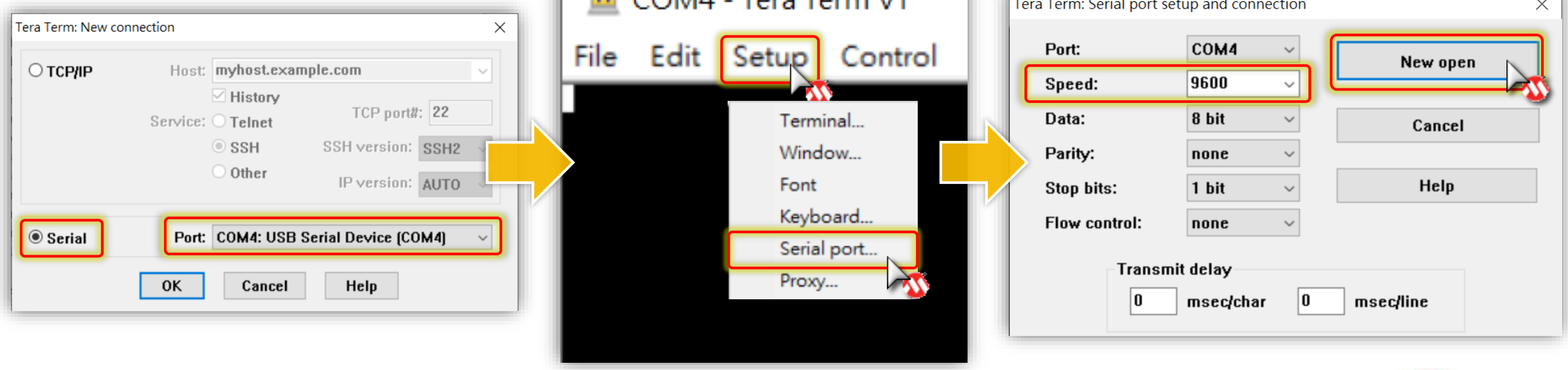
Step 1

- 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

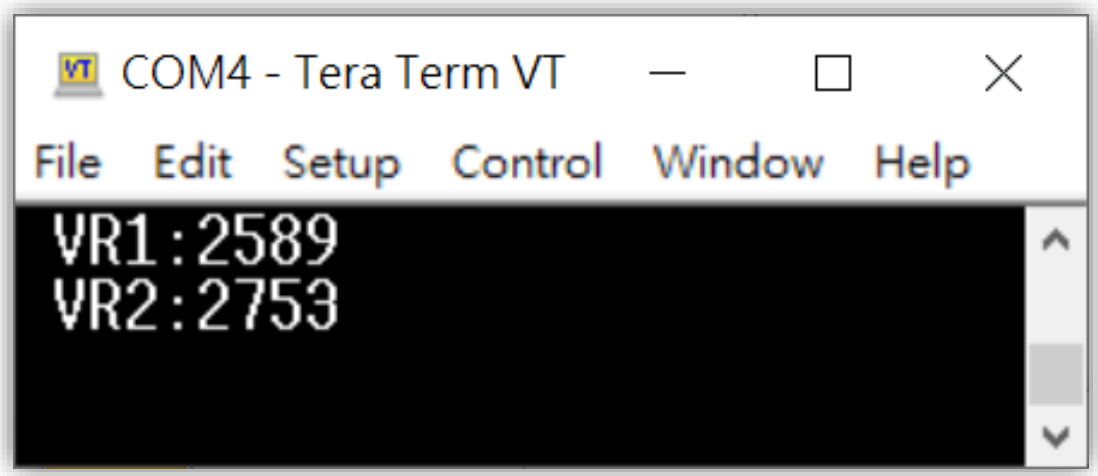
Step 2

- 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.



VT100 Console Control

- **Use VT100 command to control Tera Term UART output as**




```
myprintf("\033[2J");    // Clear screen
myprintf("\033[?25l");  // Hide cursor
myprintf("\033[y;xH");  // Set cursor to (x,y)
                        // Left-up corner is from (1,1)
```

Let's go!

MICROCHIP

Lab14 UART printf

- Base on current project or Open .\Exams\Lab14_xxx.X to do exercises
- Try to add UART1 module then output "Boot Elapsed:xx" and VR1, VR2 ADC value to PC via UART1 module as we did on OLED display.
- UART1 set to 9600 , N , 8 , 1, No flow control.
- UART has connected by PCB layout.
  dsPIC33CK RC7 (UART1-U1TX, Pin32) <-> PIC16 RC5 (RX, Pin5)
  dsPIC33CK RD10 (UART1-U1RX, Pin31) <-> PIC16 RC4 (TX, Pin6)

Let’s go!

MICROCHIP

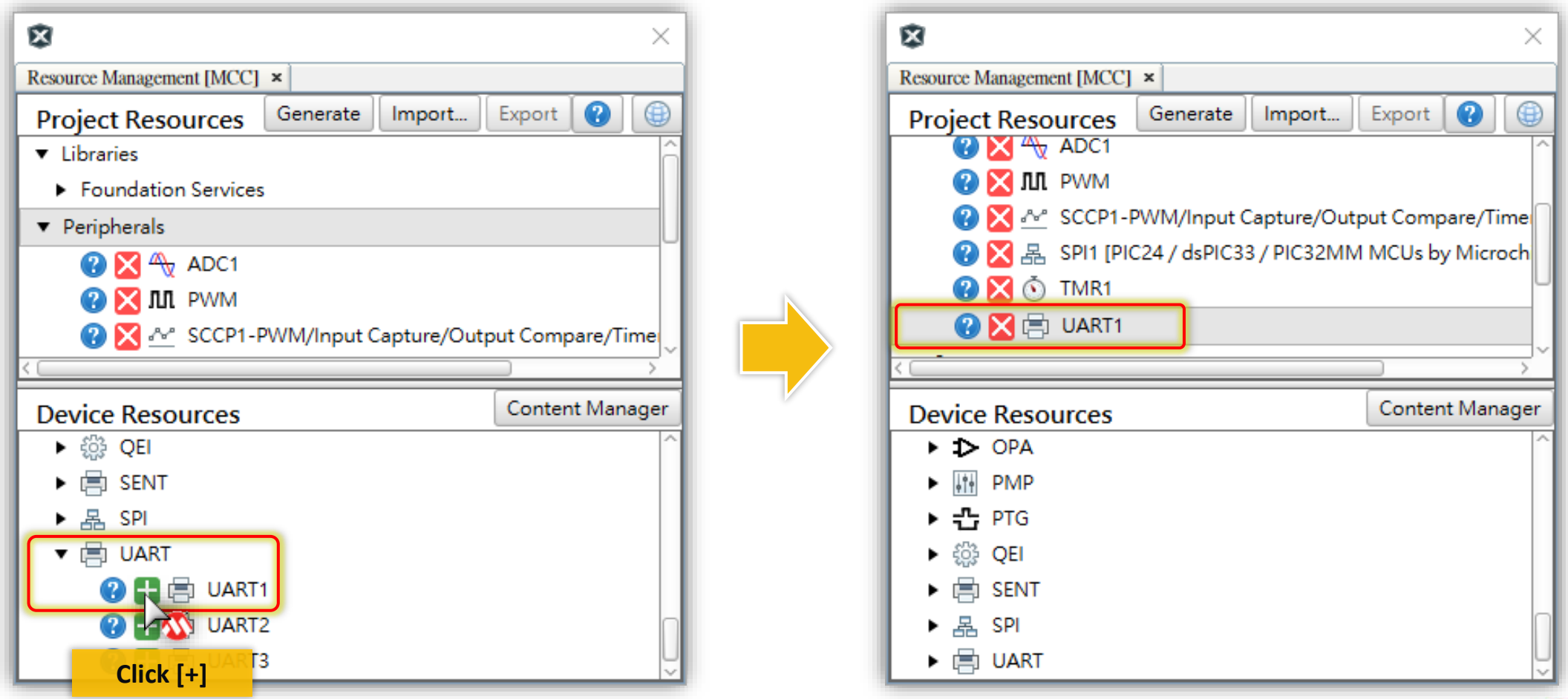
Lab14 UART printf

Connection

Lab14 UART printf

Step 1

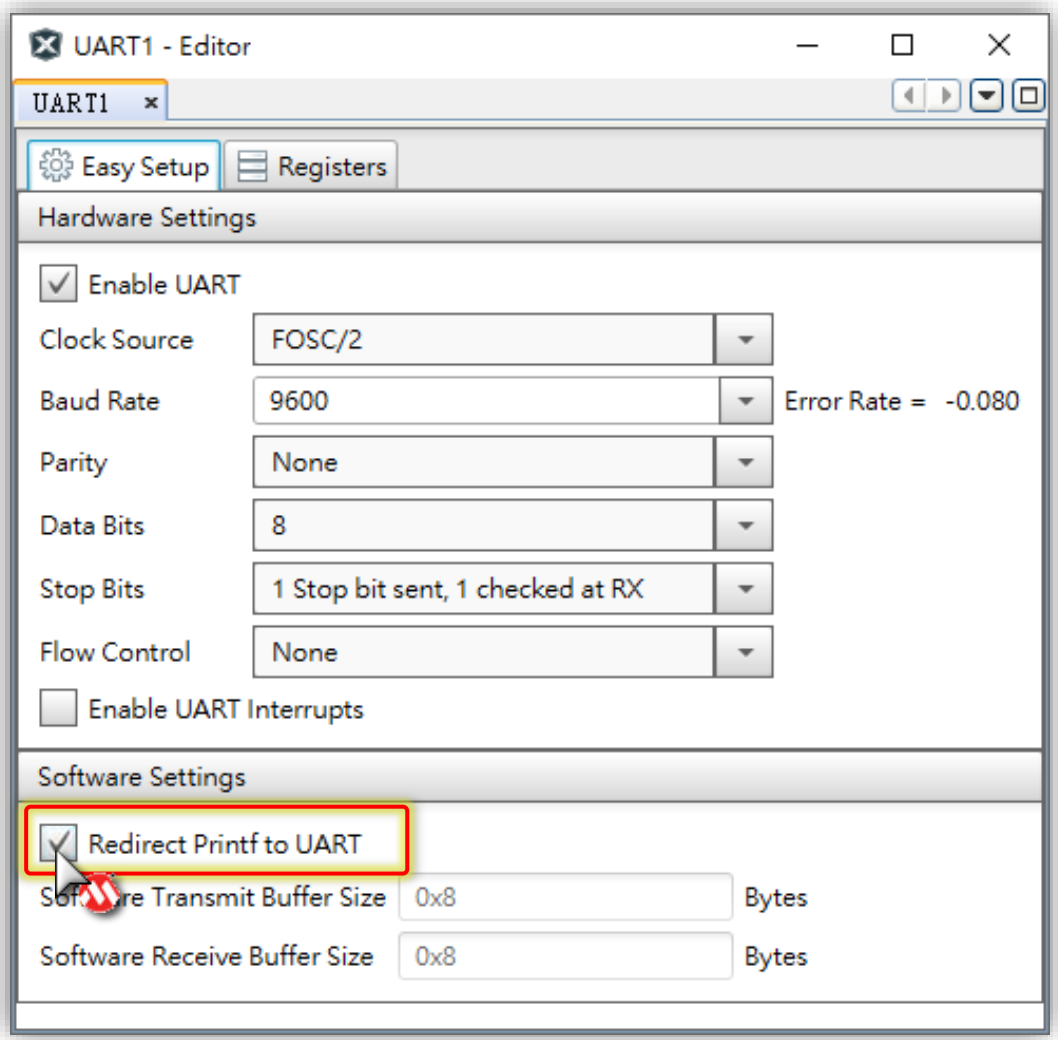
- **Click [+] to add UART > UART1 Module.**



Lab14 UART printf

Step 2

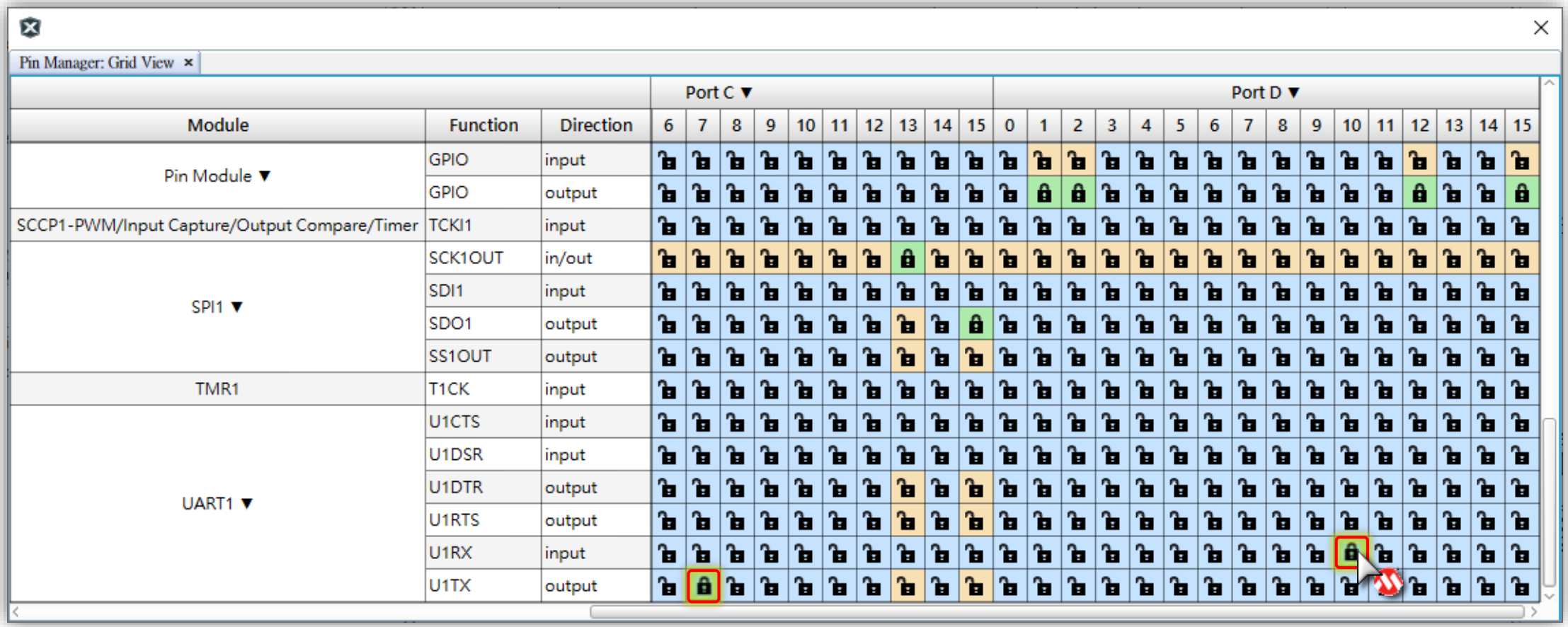
- Set Enable Redirect Printf to UART.



Lab14 UART printf

Step 3

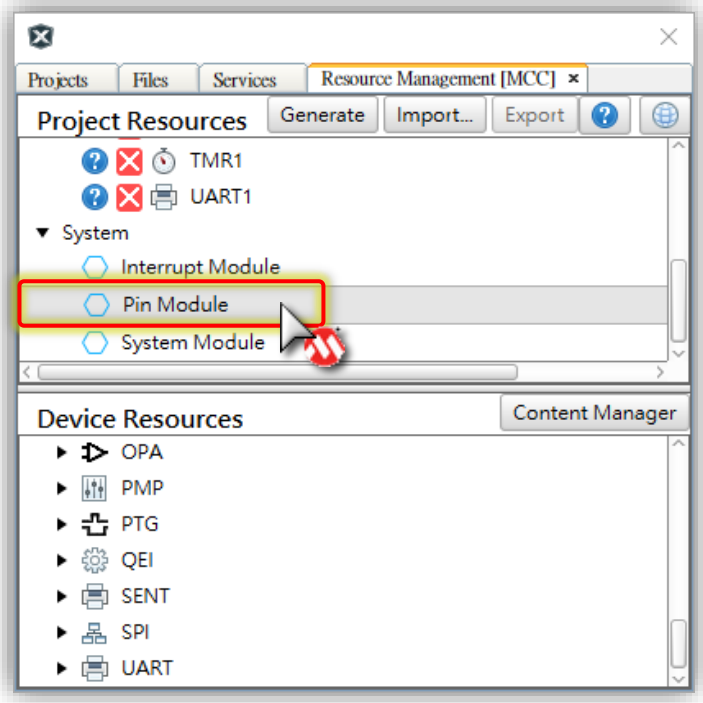
- **Assign UART1 TX and RX Pin.**
  - **Pin Manager : RC7 > U1TX , RD10 > U1RX.**

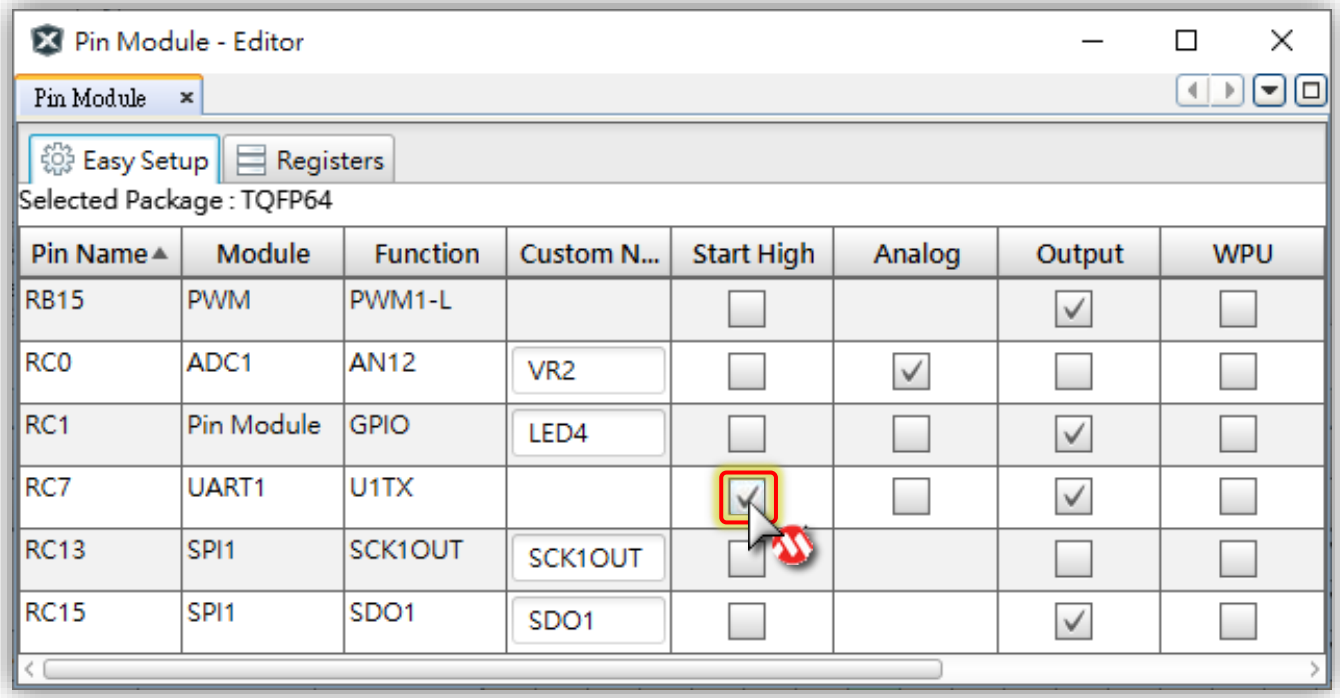


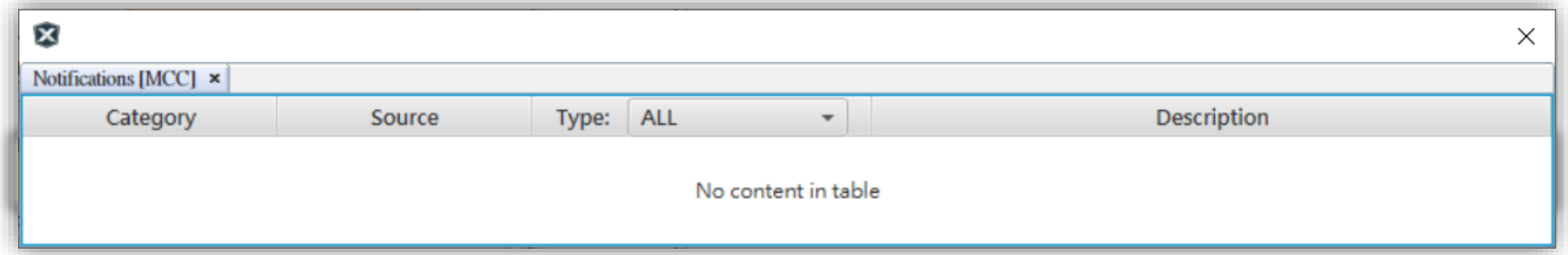
Lab14 UART printf

Step 4

- Set RC7 Start High.





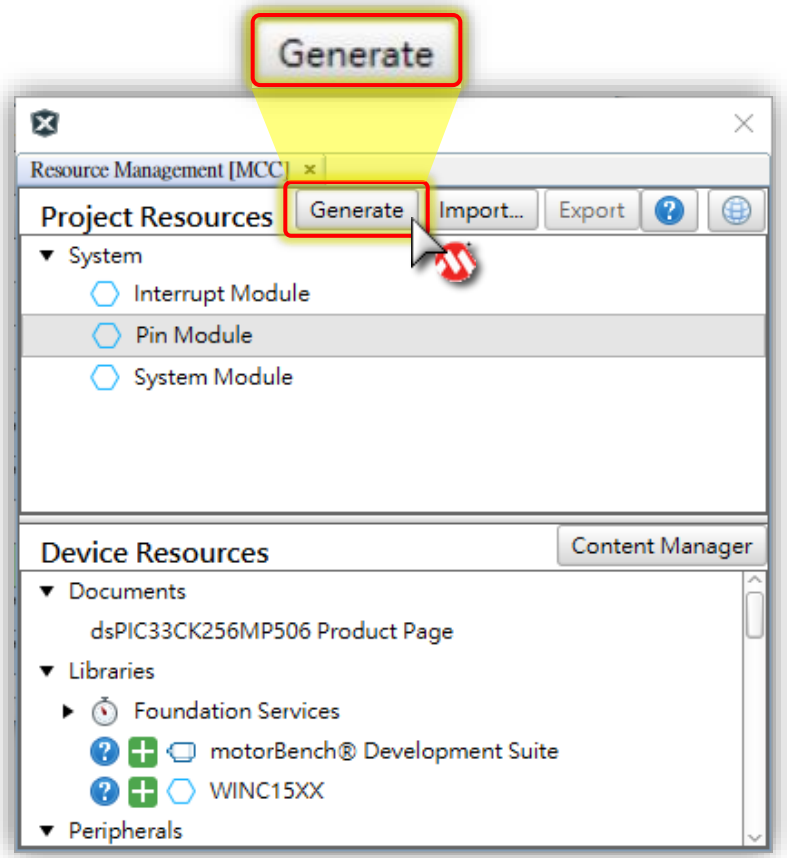


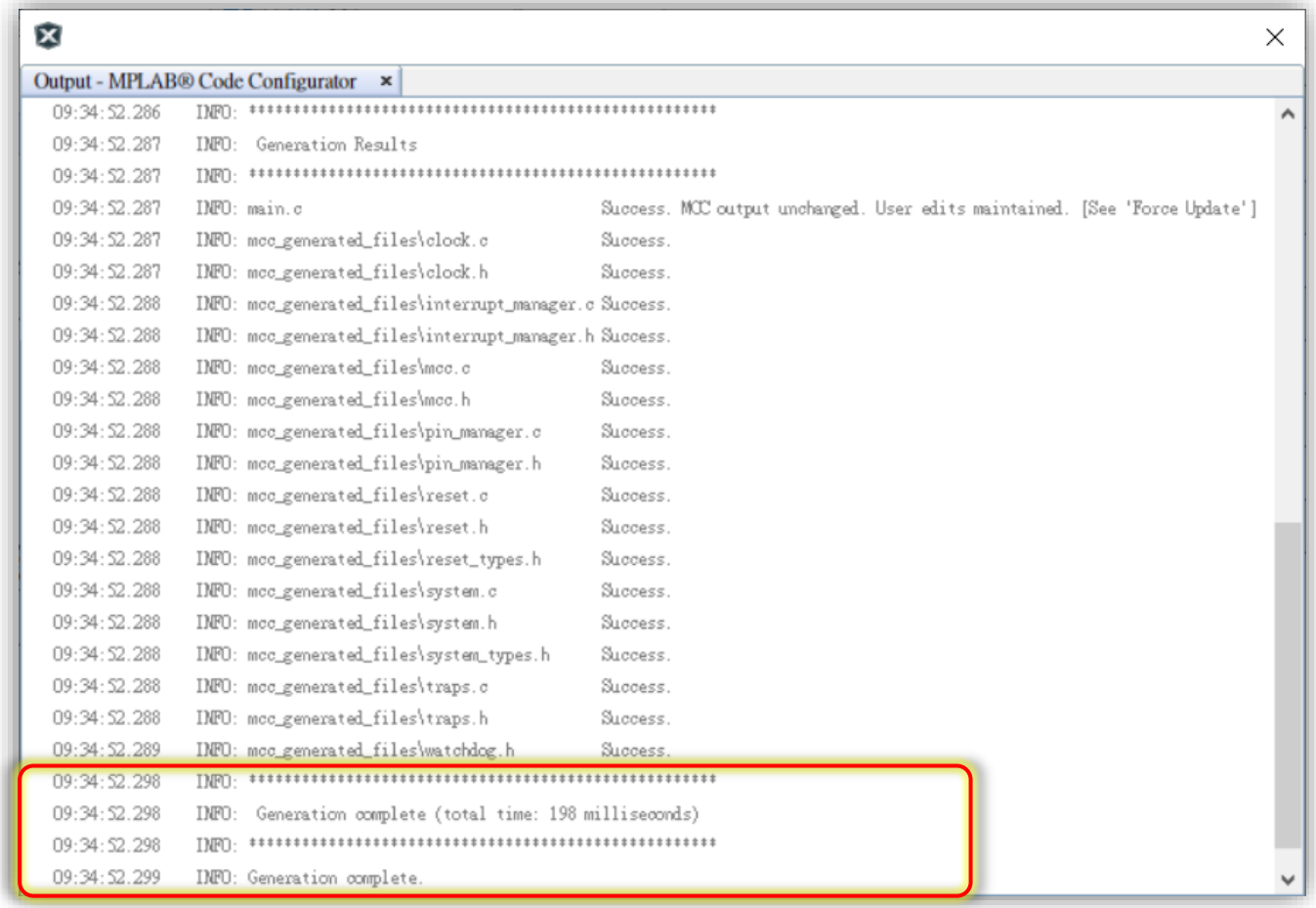
MICROCHIP

Lab14 UART printf

Step 5

- Click "Generate" Button to generate your code.





Lab14 UART printf

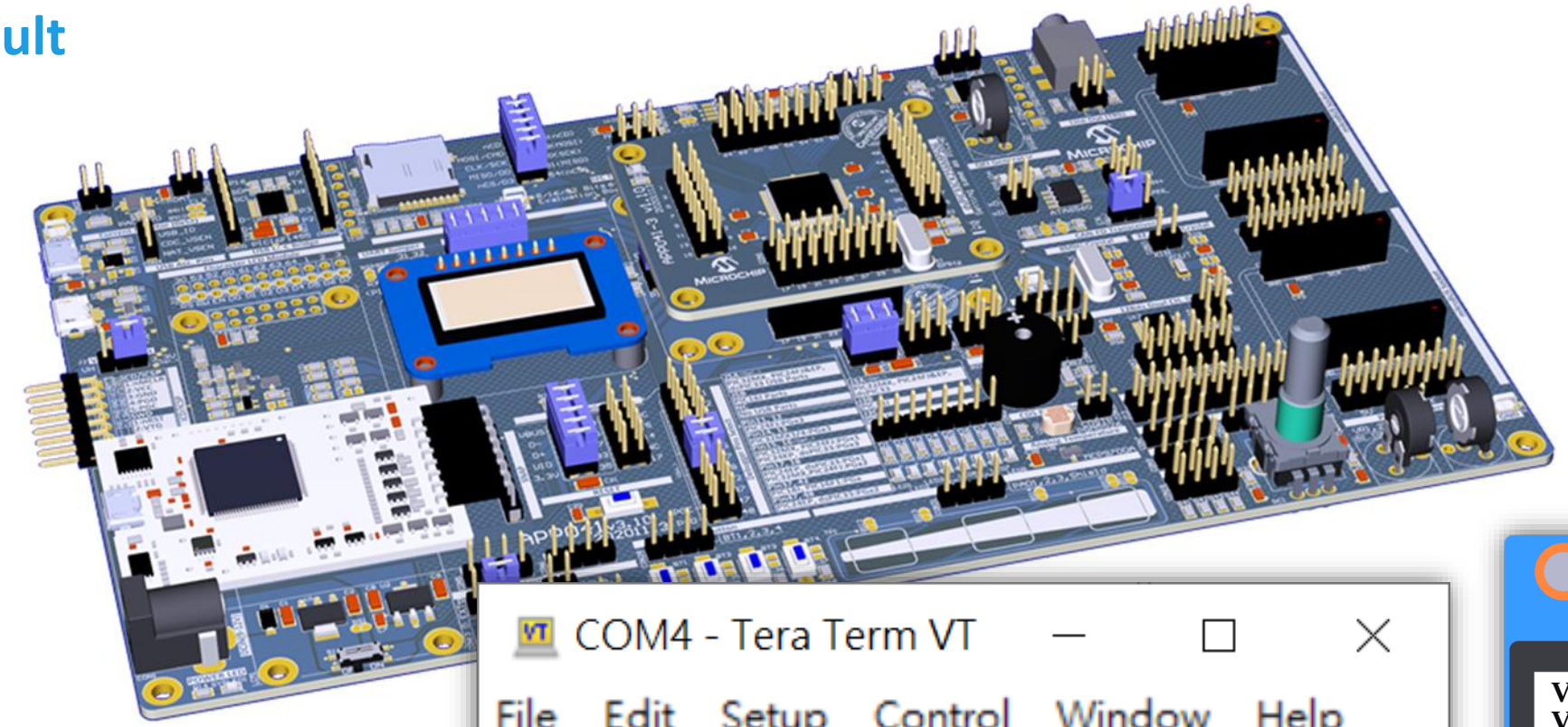
Code Example

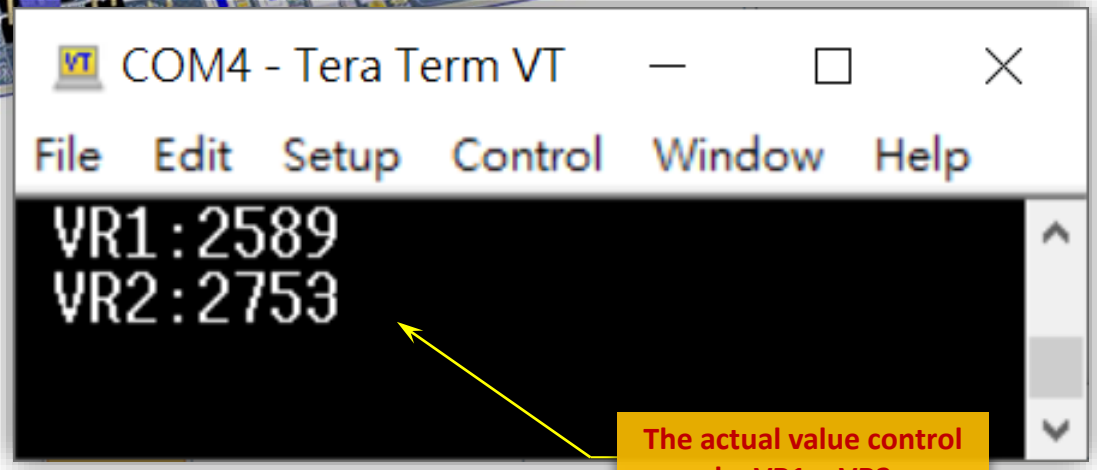
```
int main(void)
{
    ...
    printf("\033[2J");
    printf("\033[?25l");
    while (1)
    {
        // Add your application code
        ...
        if (SCCP1Flag)
        {
            ...
        }
        if (VR1Flag)
        {
            VR1Flag = 0;
            sprintf((char *) StringBuffer, "VR1:%4d", ADC1_VR1_Buffer);
            printf(" \033[1;1H%s", StringBuffer);
            myOLED_DrawStr(0, 0, StringBuffer);
        }
        if (VR2Flag)
        {
            VR2Flag = 0;
            sprintf((char *) StringBuffer, "VR2:%4d", ADC1_VR2_Buffer);
            printf(" \033[2;1H%s", StringBuffer);
            myOLED_DrawStr(0, 1, StringBuffer);
        }
    }
    return 1;
}
```

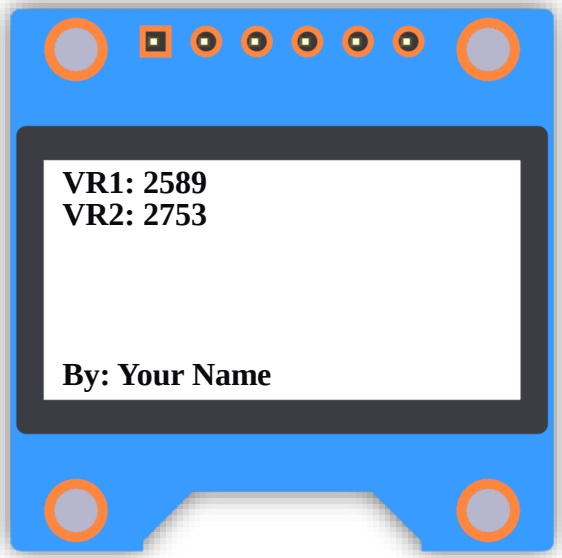
MICROCHIP

Lab14 UART printf

Result







MICROCHIP

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 : "uartn.h"

  - **void UART*n*_Initialize(void);**
  - **uint8_t UART*n*_Read(void);**
  - **void UART*n*_Write(uint8_t byte);**
  - **bool UART*n*_IsRxReady(void);**
  - **bool UART*n*_IsTxReady(void);**
  - **bool UART*n*_IsTxDone(void);**

MICROCHIP

Lab15 UART Communication

- Base on current project or Open .\Exams\Lab15_xxx.X to do exercises
- Try to enable UART1 interrupt to be received UART data come from PC.
- To set a suitable received buffer size.
- Shows received character on OLED display and echo it to PC.

Let’s go!

MICROCHIP

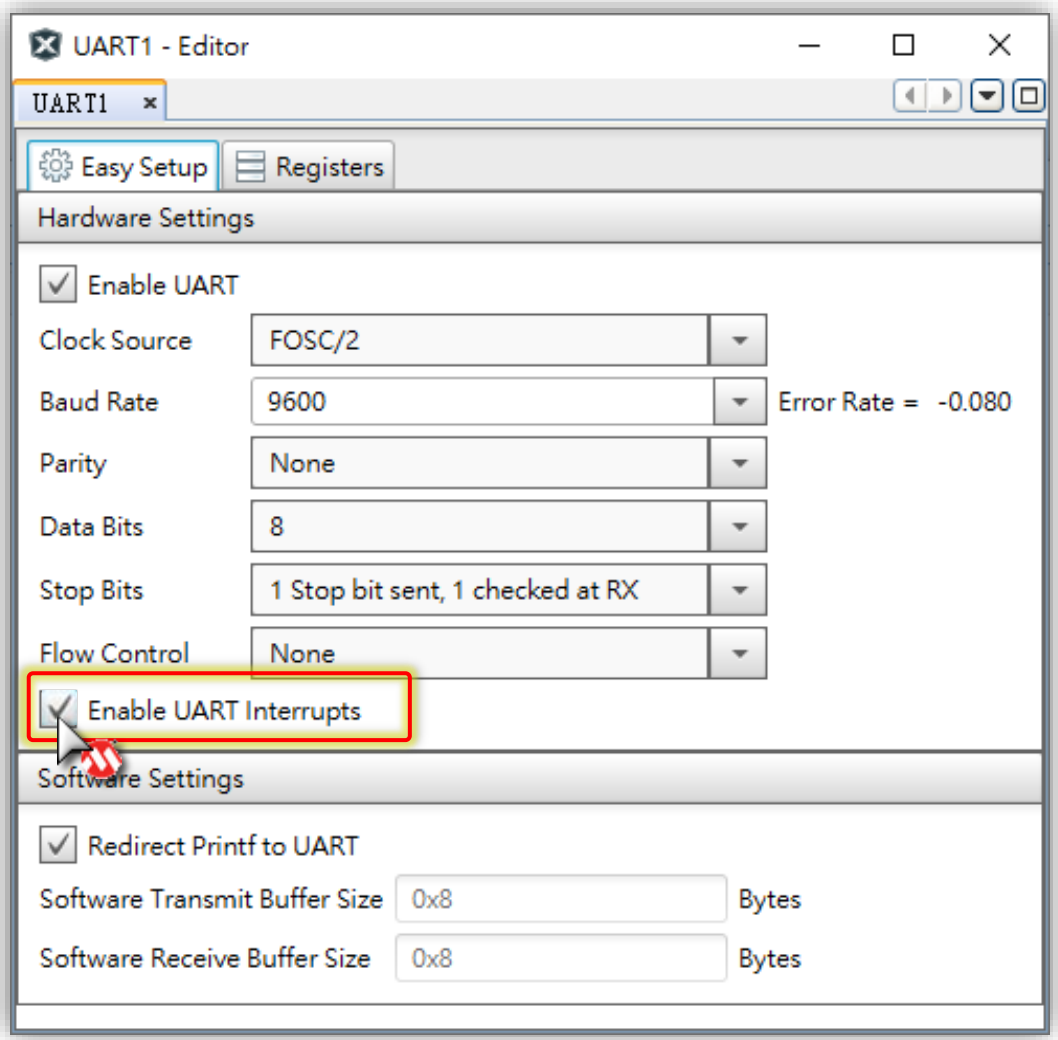
Lab15 UART Communication

Connection

Lab15 UART Communication

Step 1

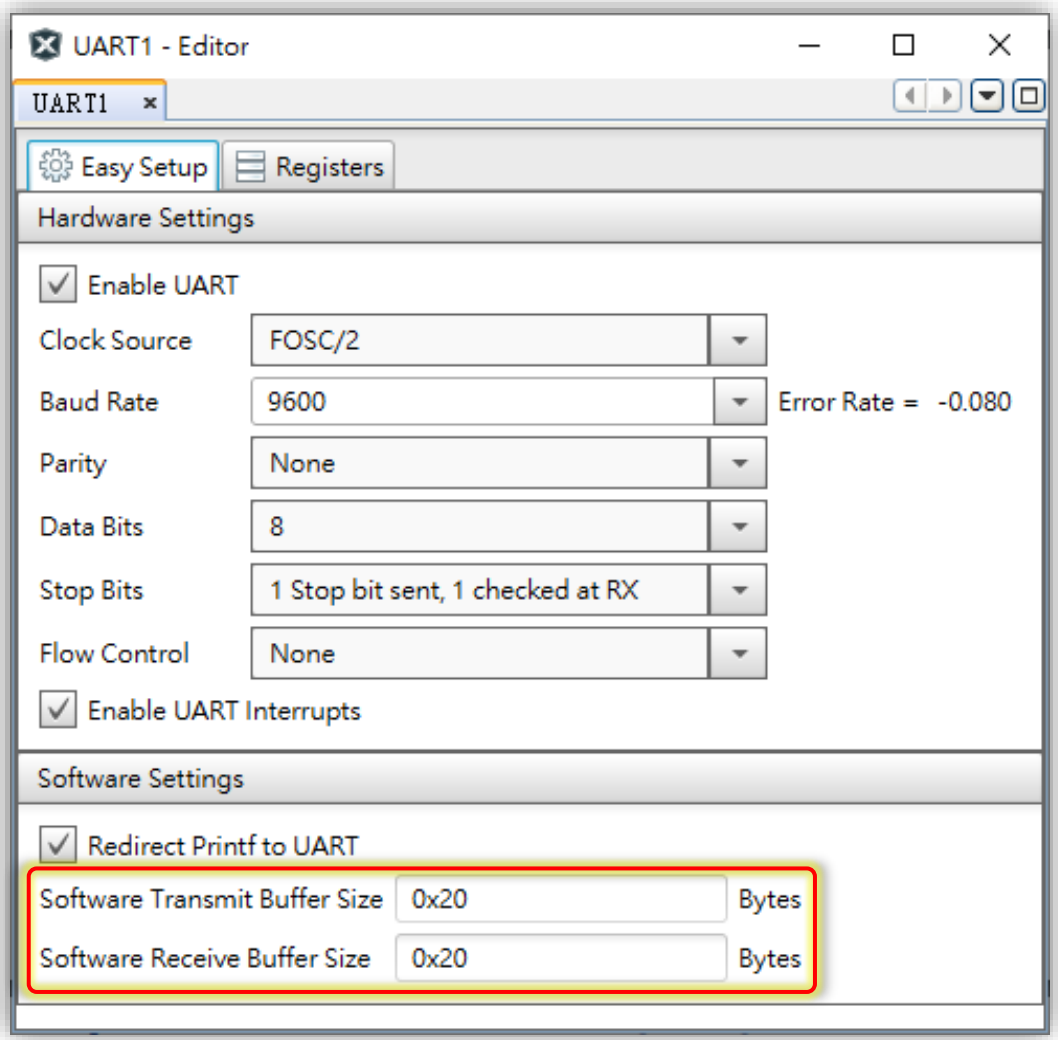
- Set Enable UART Interrupts.



Lab15 UART Communication

Step 2

- Set Software Transmit/Receive Buffer Size : 0x8 > 020.

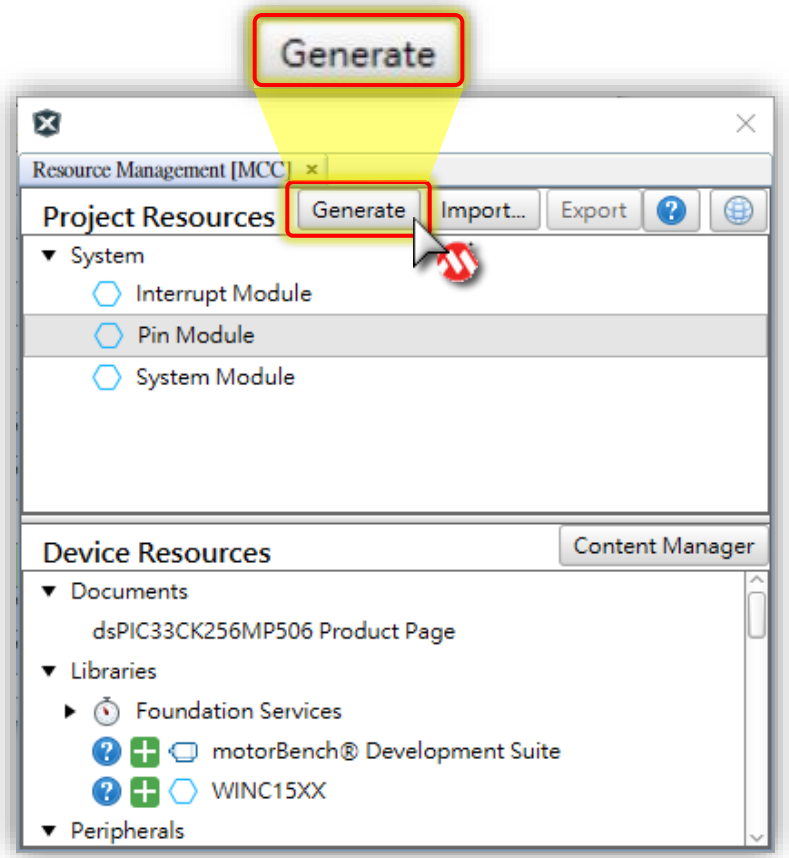


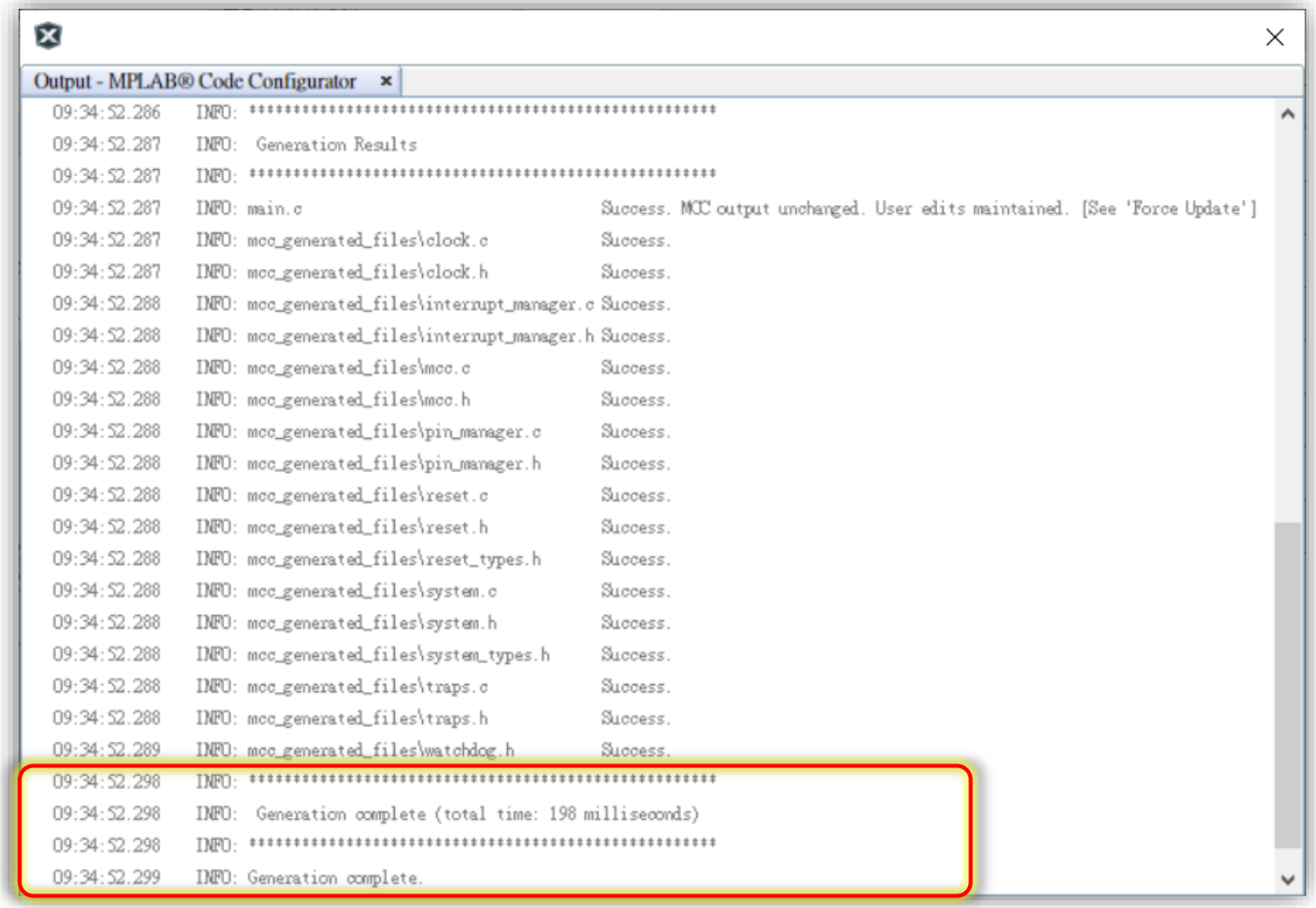
MICROCHIP

Lab15 UART Communication

Step 3

- Click "Generate" Button to generate your code.





Lab15 UART Communication

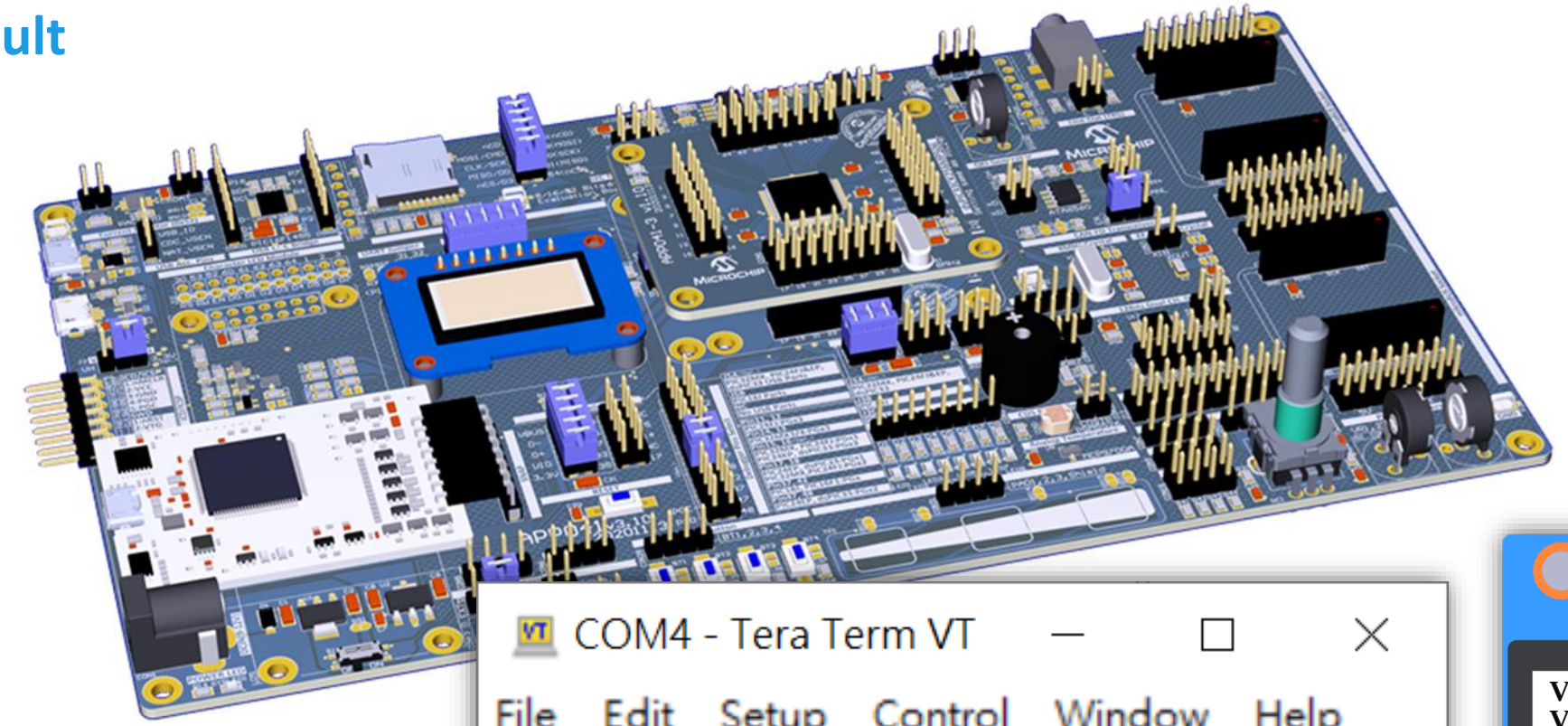
Code Example

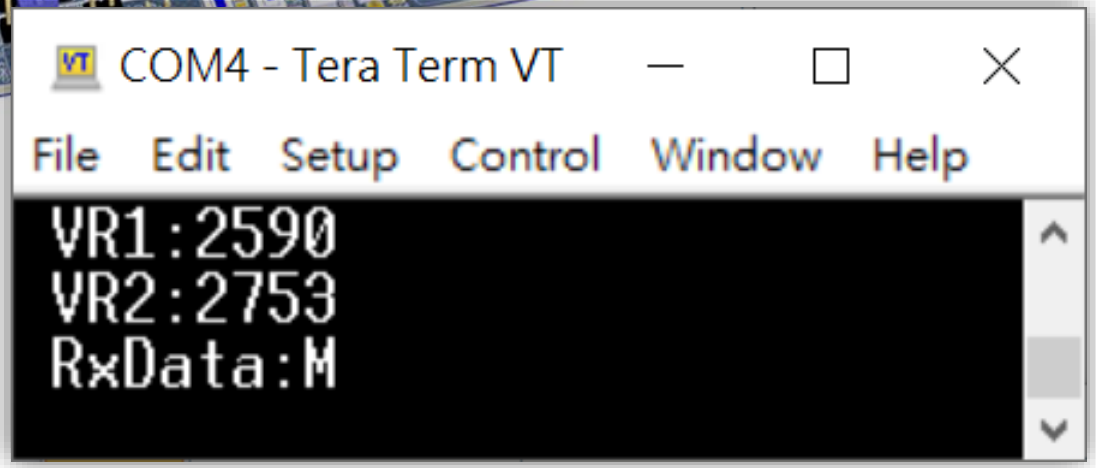
```
#include "mcc_generated_files/pwm.h"
#include "mcc_generated_files/uart1.h"

…

int main(void)
{
    …
    while (1)
    {
        // Add your application code
        …
        if (UART1_IsRxReady())
        {
            sprintf((char *) StringBuffer, "RxData:%1c", UART1_Read());
            printf(" \033[6;1H%s", StringBuffer);
            myOLED_DrawStr(0, 6, StringBuffer);
        }
     }
    return 1;
}
```

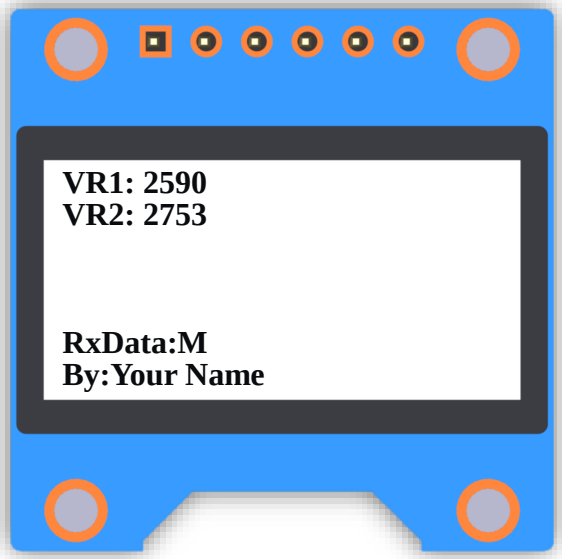
MICROCHIP

Lab15 UART Communication

Result





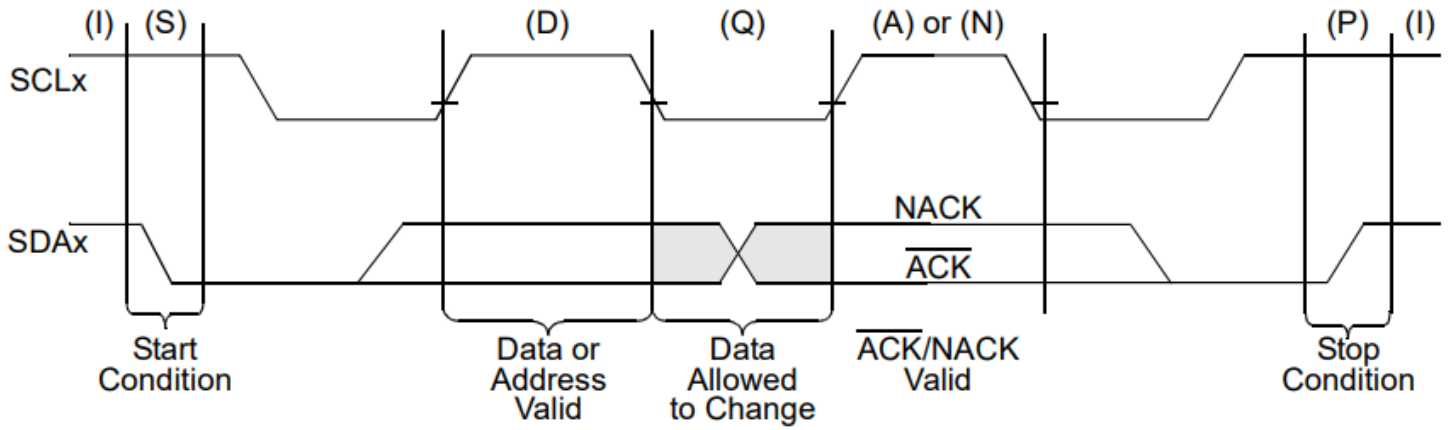


MICROCHIP

I2C Application

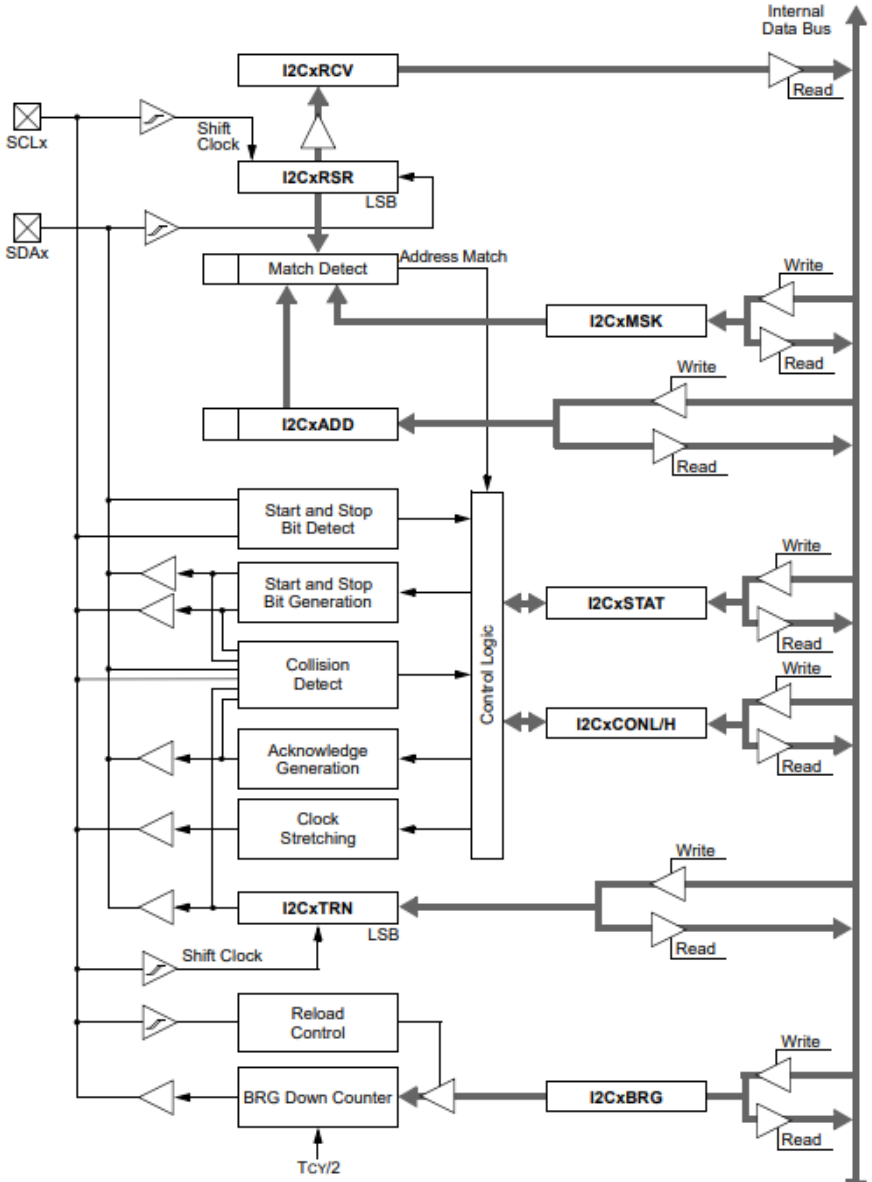
What's I2C

- 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.



dsPIC33's I2C Module

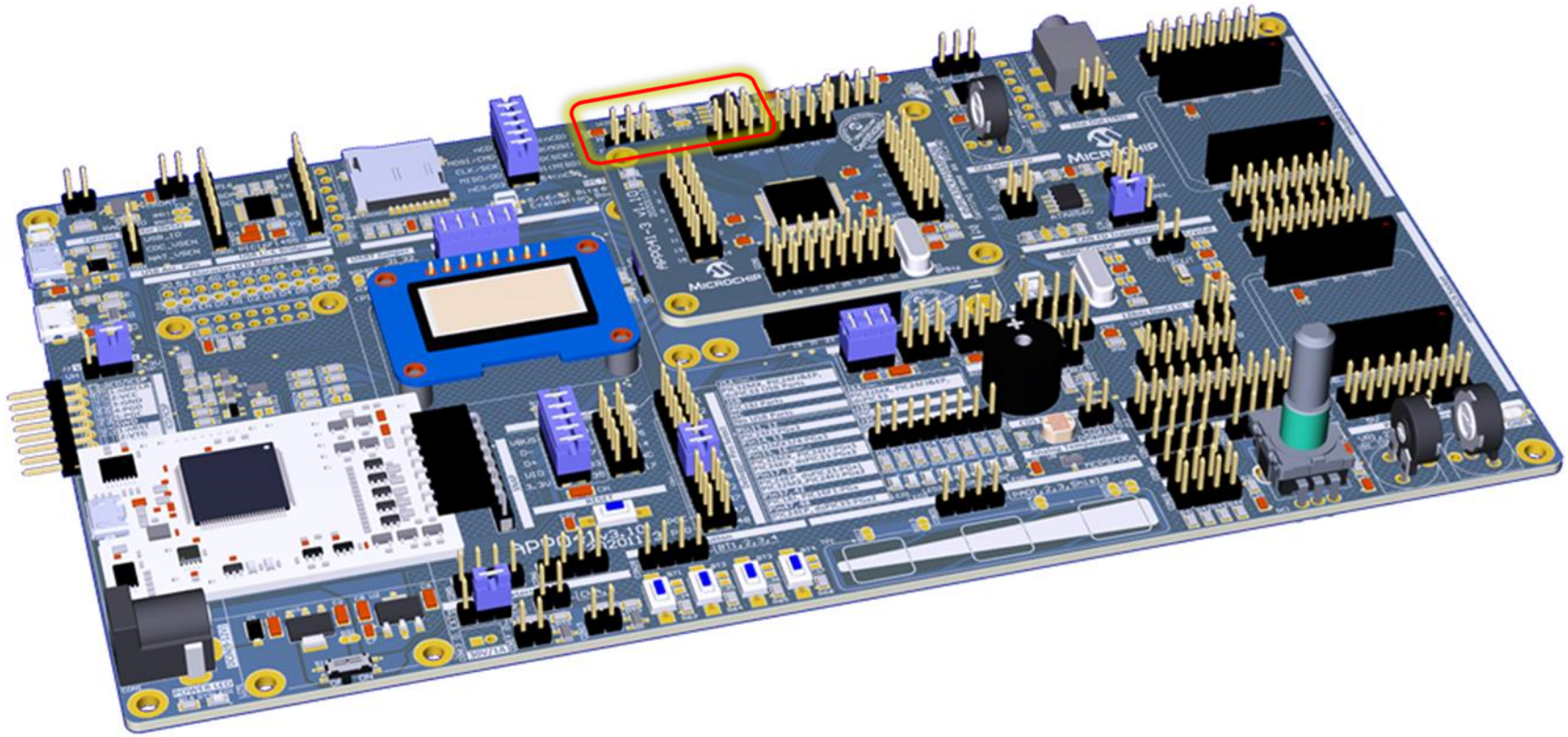
- **dsPIC33CK256MP506 Provide three I2C modules.**
- **Support 7, 10 Bits Device Addresses and General Call Address.**
- **Provide Clock Stretching.**
- **Both 100 kHz and 400 kHz Bus Specifications.**
- **SMBus Compatible Voltage Thresholds.**
- **Pin Definition :**
  - **SCL (Serial Clock)**：Clock, dsPIC33 <-> Device.
  - **SDA (Serial Data)** ：Data In/Out, dsPIC33 <-> Device.



MICROCHIP

I2C Devices

- **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
- **EEPROM (24AA32A)：**
  - 24AA32A is a 32Kb I2C EEPROM has a Page Write capability for up to 32 bytes of data.



I2C Functions of MCC

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

  Prototype definition : "i2cn.h"

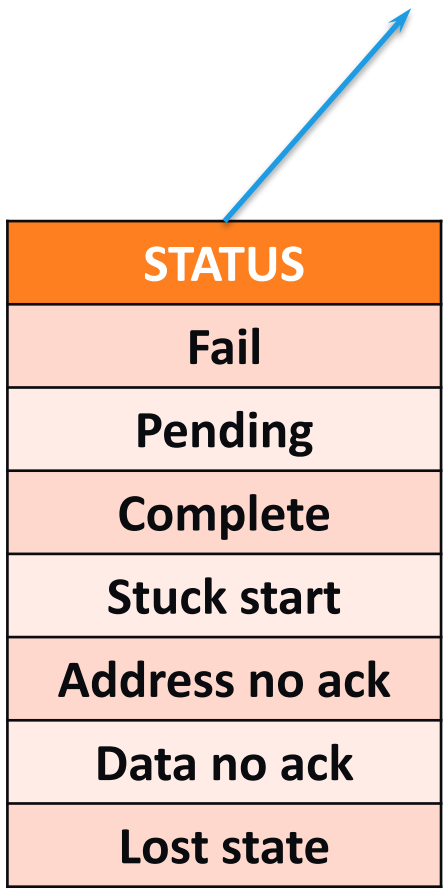
  - void I2C*n*_Initialize(void);
  - bool I2C*n*_MasterQueueIsEmpty(void);
  - bool I2C*n*_MasterQueueIsFull(void);
  - void I2C*n*_MasterRead(uint8_t *pdata, uint8_t length, uint16_t address,
    I2C*n*_MESSAGE_STATUS *pstatus);
  - void I2C*n*_MasterWrite(uint8_t *pdata, uint8_t length, uint16_t address,
    I2C*n*_MESSAGE_STATUS *pstatus);

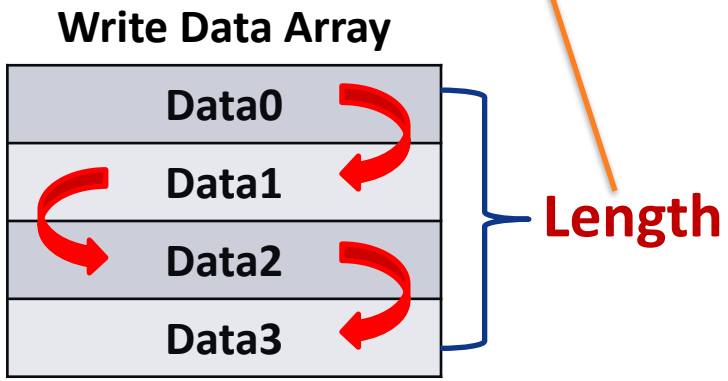
- **I2C Function is non-block function. This means all function running by interrupt. Operation not execute immediate after function call.**

MICROCHIP

I2C Functions Introduction

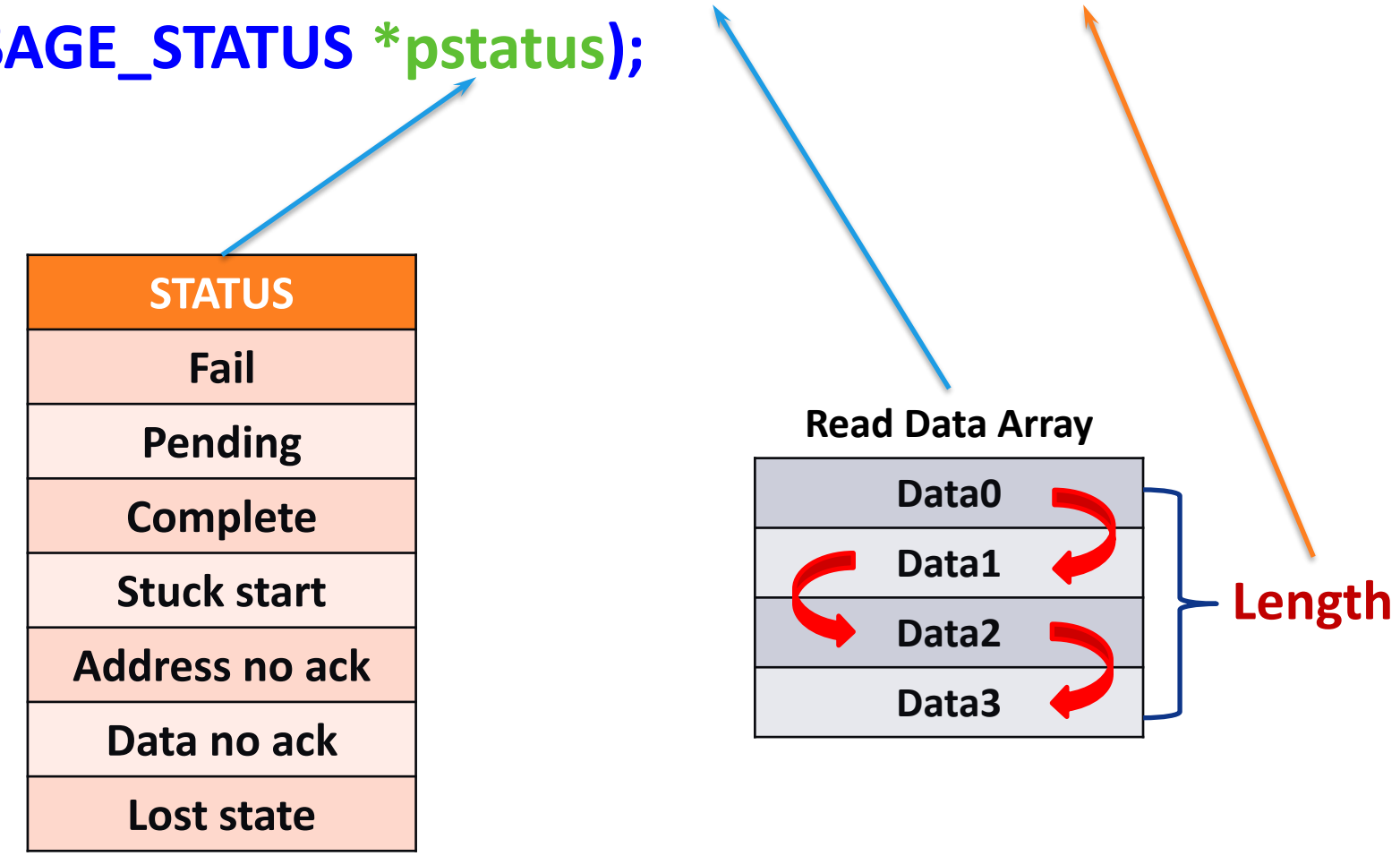
- void I2Cn_MasterWrite(uint8_t *pdata, uint8_t length, uint16_t address, I2Cn_MESSAGE_STATUS *pstatus);





I2C Functions Introduction (cont.)

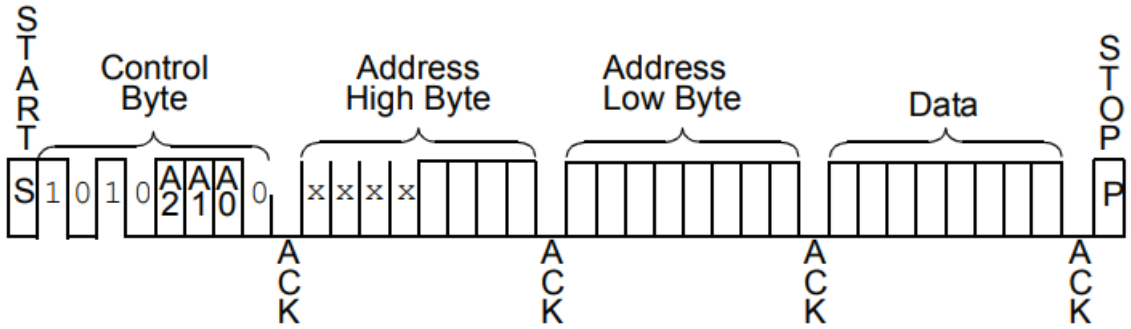
- void I2Cn_MasterRead(uint8_t *pdata, uint8_t length, uint16_t address, I2Cn_MESSAGE_STATUS *pstatus);



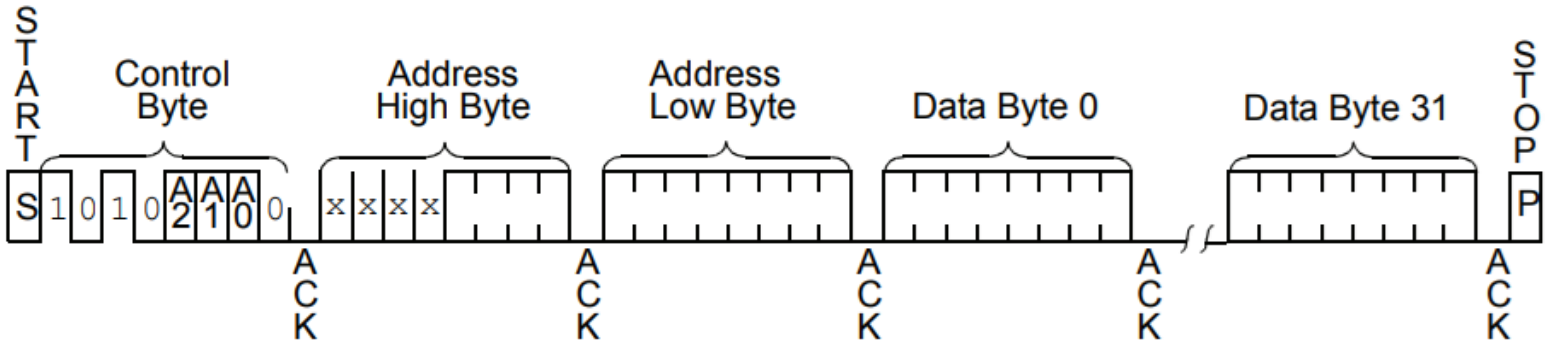
MICROCHIP

EEPROM Write (24AA32A)

- **BYTE WRITE: I2Cn_MasterWrite(); //Data Array Length = 2+1**



- **PAGE WRIYE: I2Cn_MasterWrite(); //Data Array Length = 2+n**

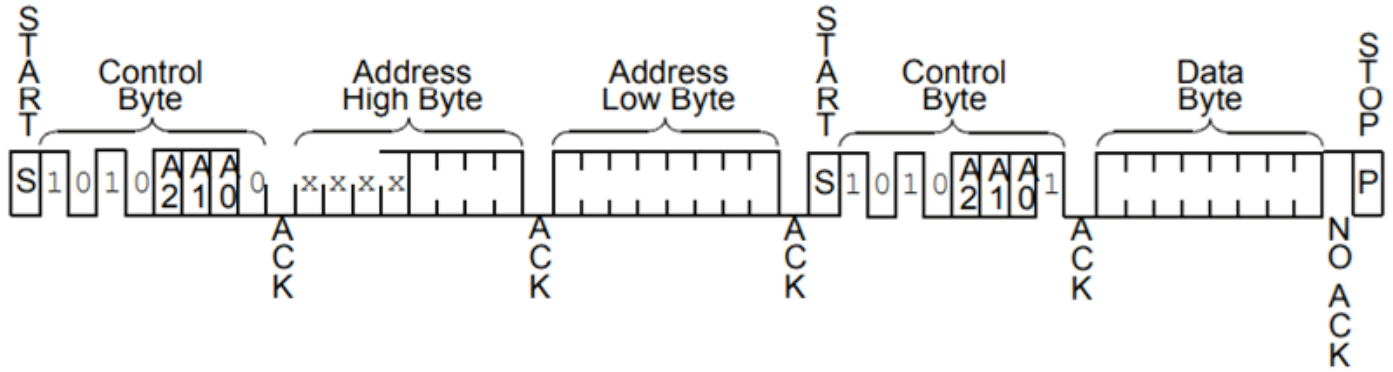


- **At this section, we focus on PAGE WRITE.**

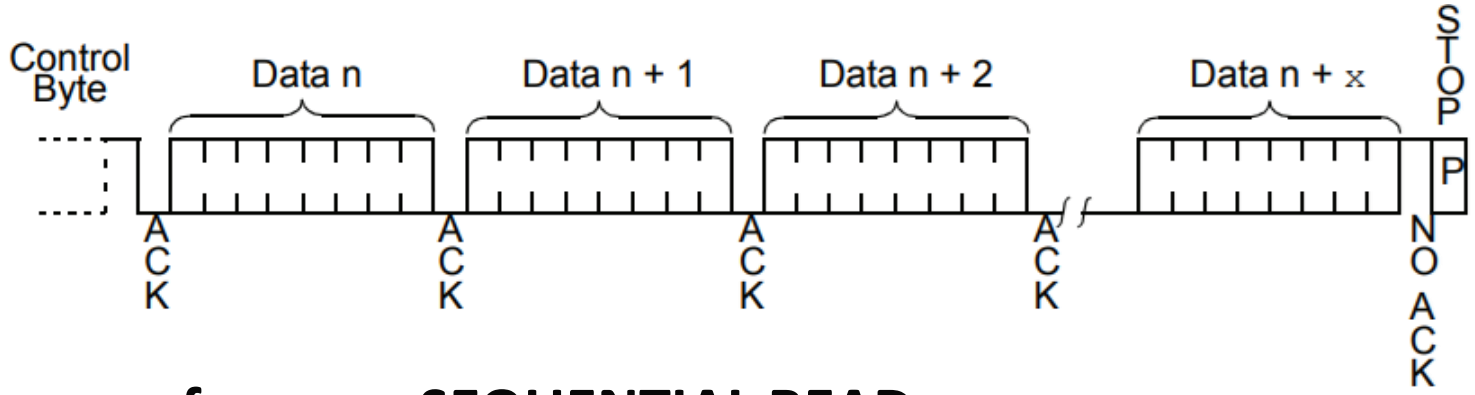
MICROCHIP

EEPROM Read (24AA32A)

- **RANDOM READ: I2Cn_MasterWrite() + I2Cn_MasterRead();**



- **SEQUENTIAL READ: I2Cn_MasterWrite() + I2Cn_MasterRead();**



- **At this section, we focus on SEQUENTIAL READ.**

MICROCHIP

Lab16 EEPROM

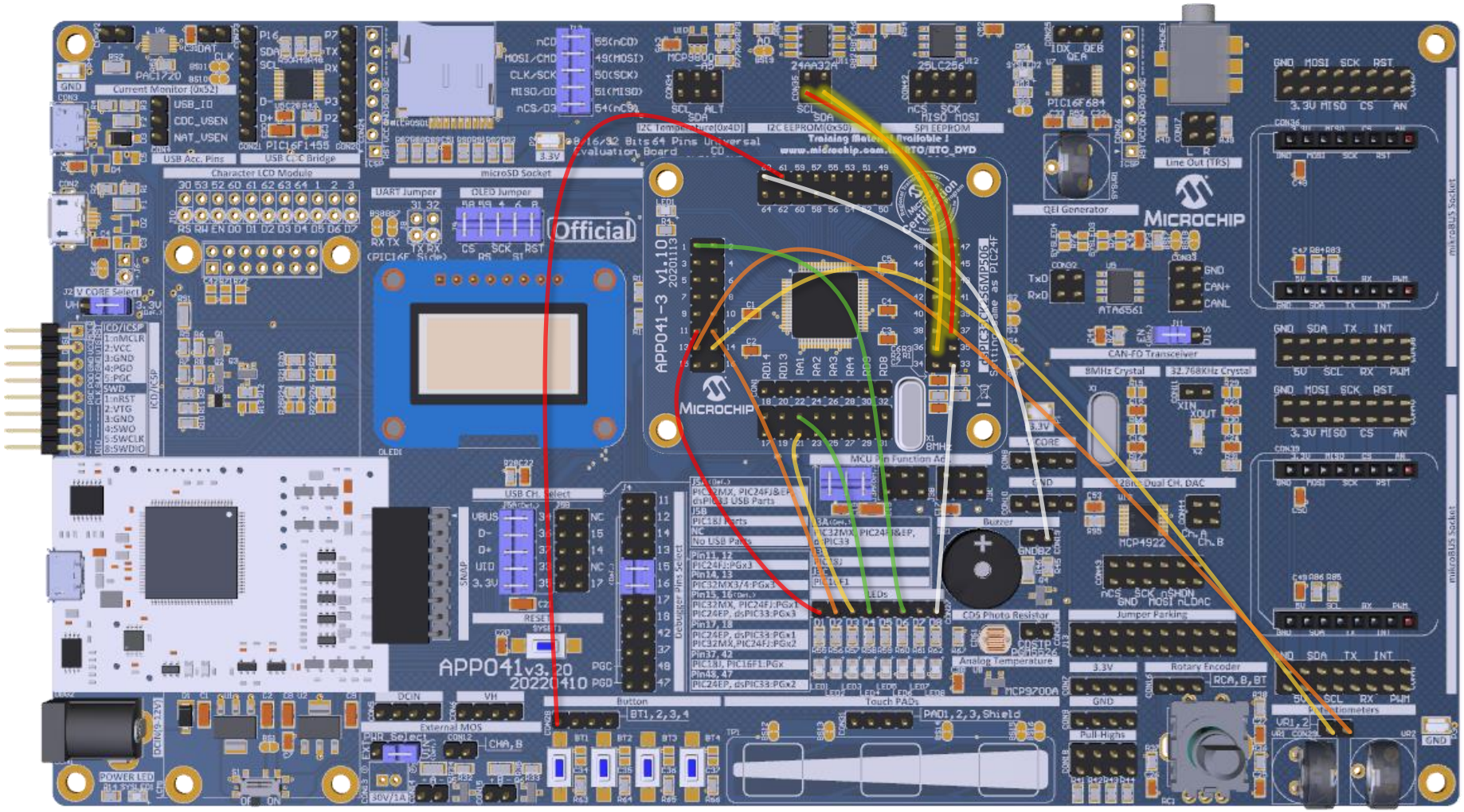
- Base on current project or Open .\Exams\Lab16_xxx.X to do exercises
- Try to add I2C1 module then write the value into the EEPROM and Show on OLED display.
- I2C1 clock set to 100kHz,
- Read the value from EEPROM and Show on OLED display.

Let’s go!

MICROCHIP

Lab16 EEPROM

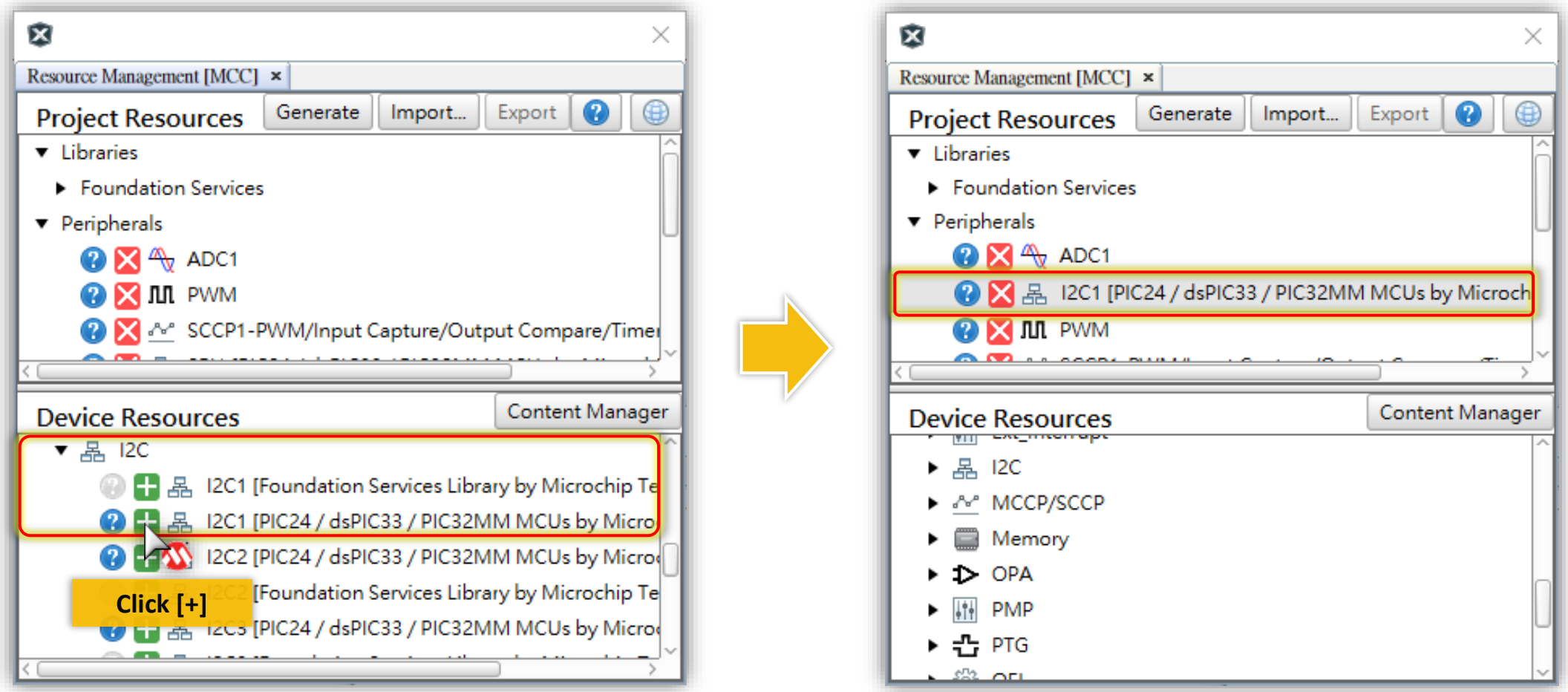
Connection



Lab16 EEPROM

Step 1

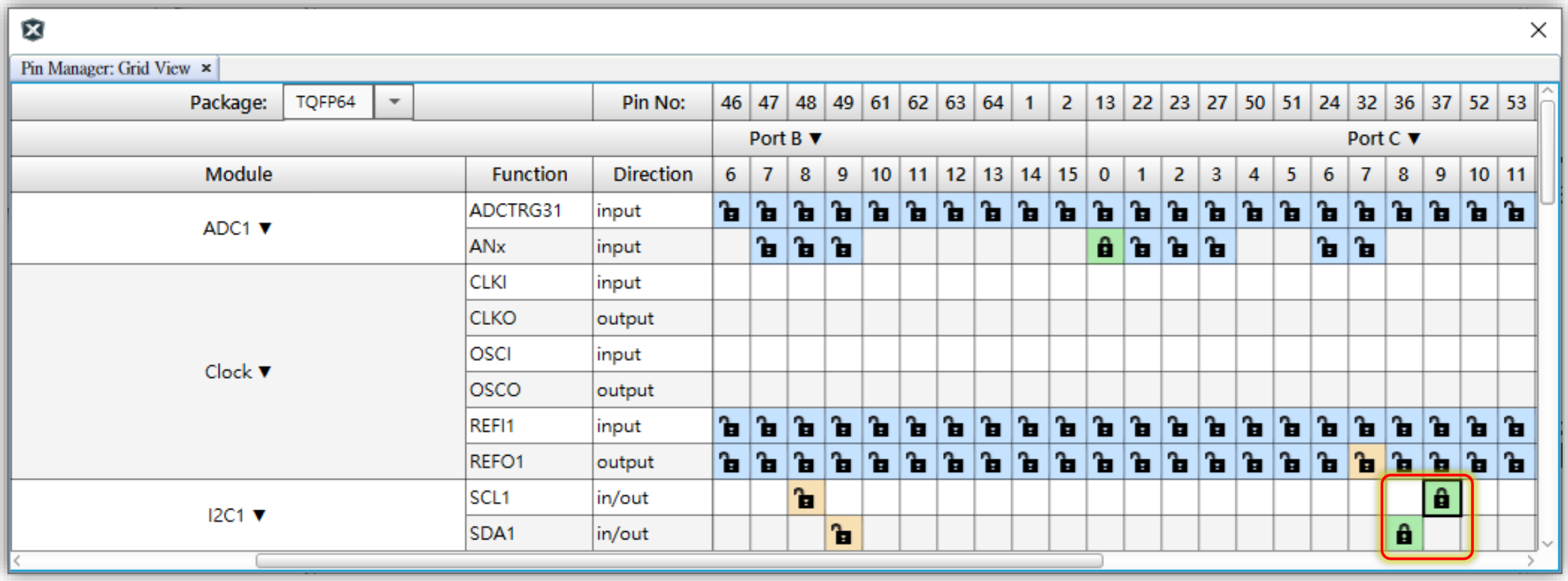
- **Click [+] to add I2C > I2C1[PIC24 / dsPIC33 / PIC32MM MCUs by Microchip…] Module.**



Lab16 EEPROM

Step 2

- **Change I2C SCL1 and SDA1 Pin.**
  - **Pin Manager : RB8 > RC9 (SCL1) , RB9 > RC8 (SDA1).**

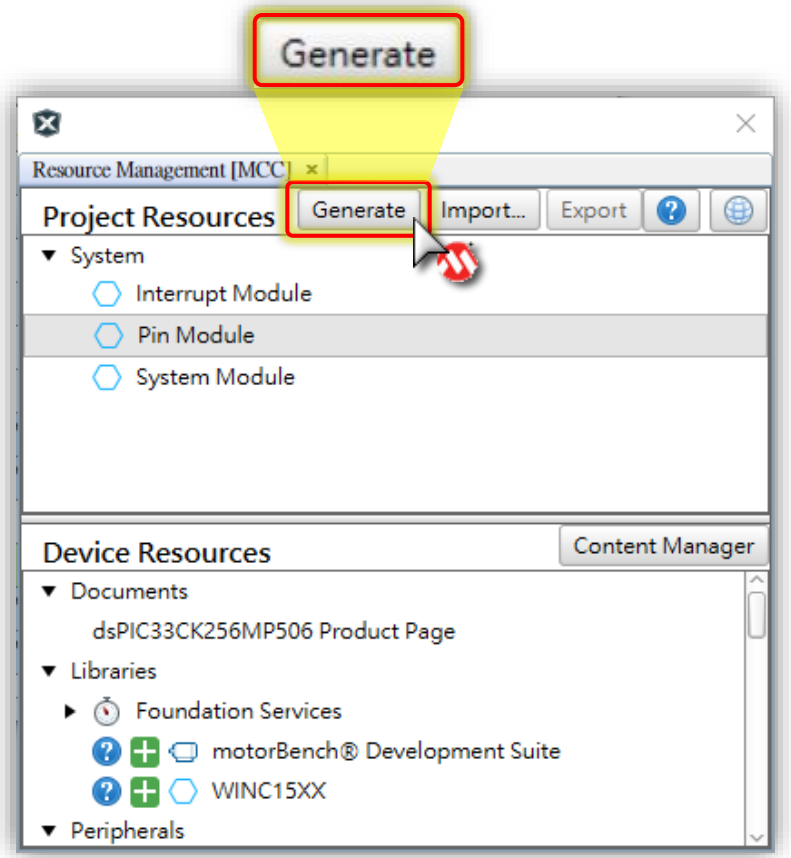


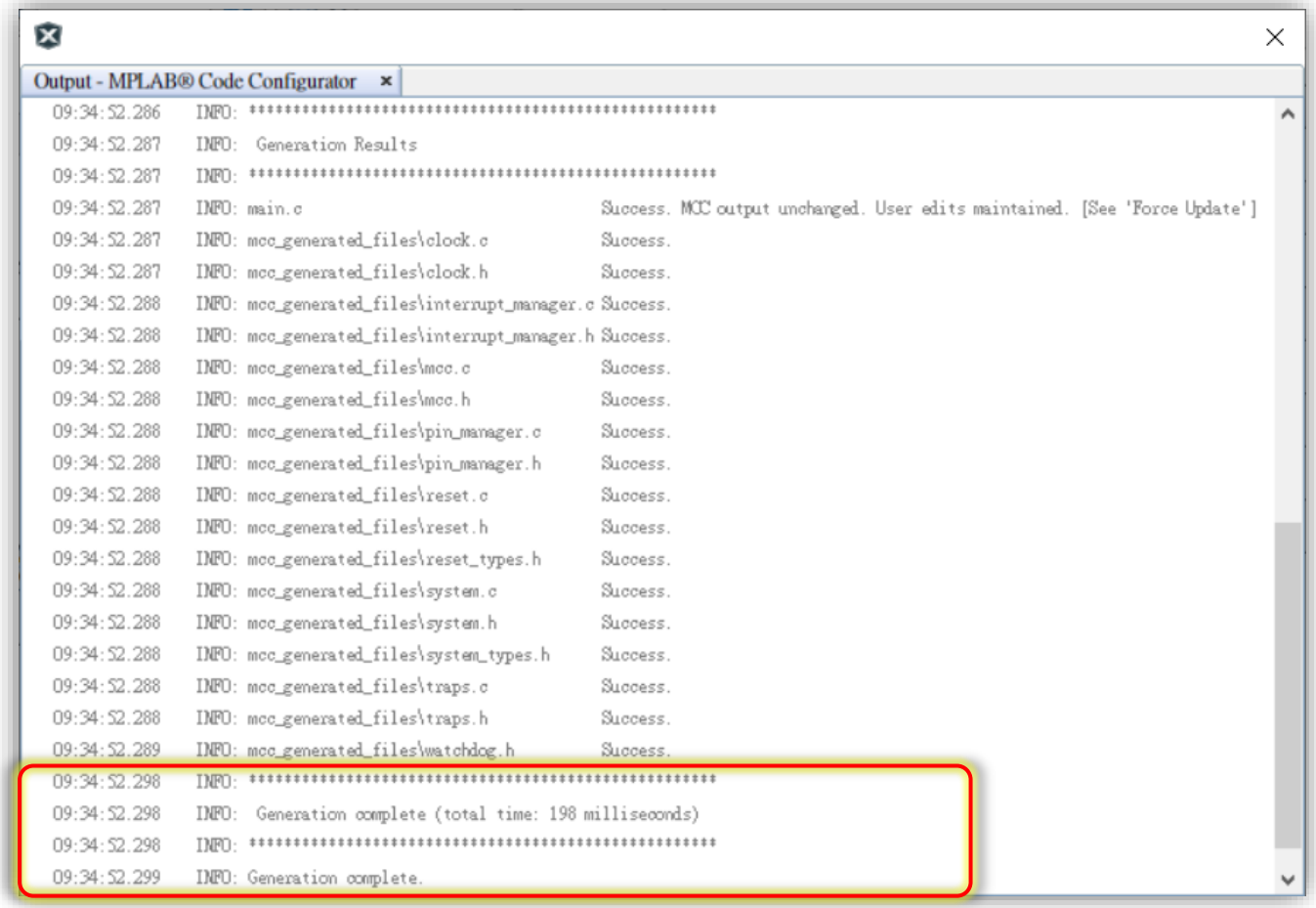
MICROCHIP

Lab16 EEPROM

Step 3

- Click "Generate" Button to generate your code.





Lab16 EEPROM

Code Example

```
#include "mcc_generated_files/uart1.h"
#include "mcc_generated_files/i2c1.h"

#define EEPROM_Address 0x50

...
uint8_t Button_Flag=0;
uint8_t I2C_EEPROM_Write_Data[4];
uint8_t I2C_EEPROM_Read_Data[2];

int main(void)
{
   while (1)
   {
      // Add your application code
      if(BT1_GetValue() && Button_Flag==0)
      {
         LED8_SetLow();
         Button_Flag=1;
      }
      else if(!BT1_GetValue()&&Button_Flag==1)
      {
         static uint8_t i = 0;
         if (++i > 100) i = 0;

         I2C_EEPROM_Write_Data[0] = 0x00;
         I2C_EEPROM_Write_Data[1] = 0x00;
         I2C_EEPROM_Write_Data[2] = i;
         I2C_EEPROM_Write_Data[3] = i << 1;
         I2C1_MasterWrite(I2C_EEPROM_Write_Data, 4, EEPROM_Address, NULL);
         DELAY_microseconds(100);
         D8_SetHigh();
         Button_Flag=0;
      }
   }
}
```

MICROCHIP

Lab16 EEPROM

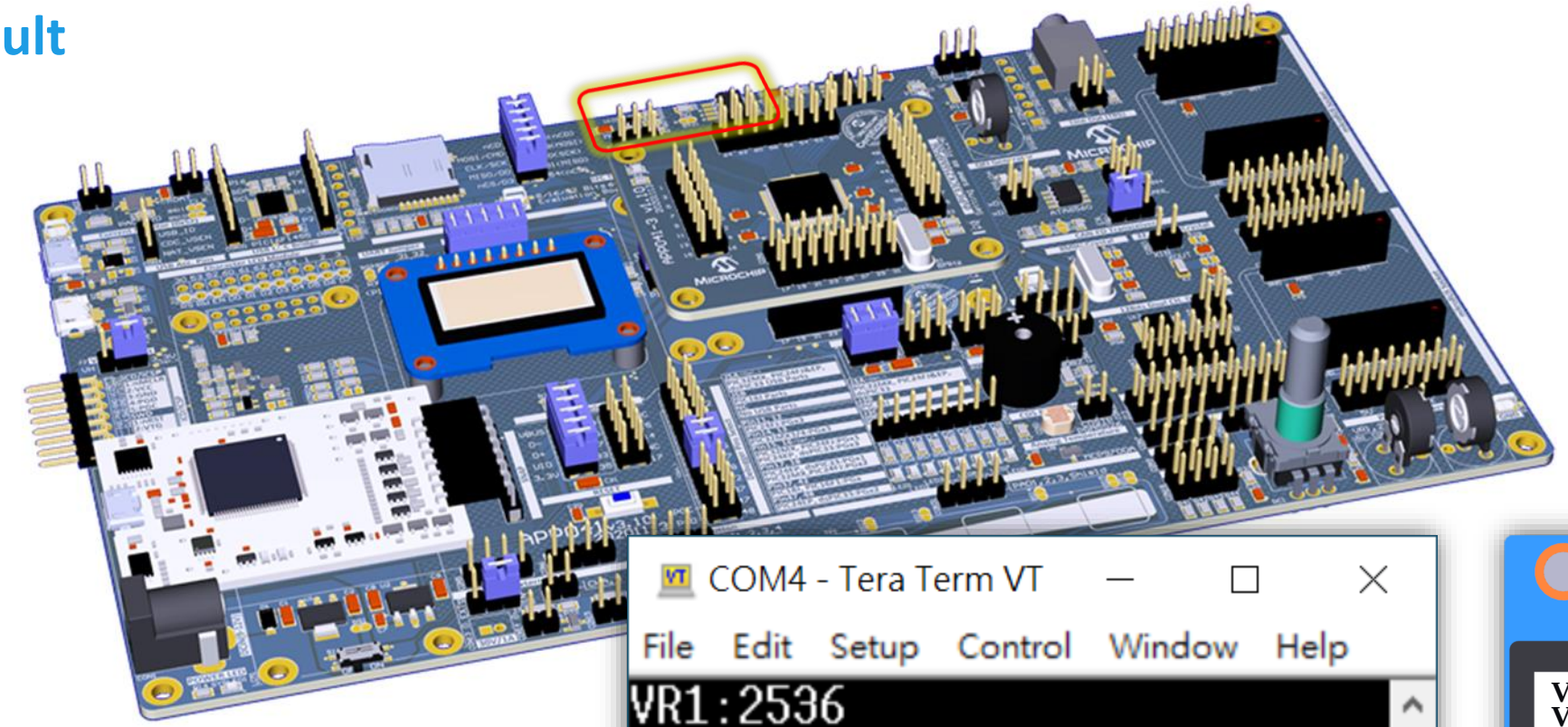
Code Example

```
....
   while (1)
   {
      // Add your application code
      ....
      if (SCCP1Flag)
      {
         SCCP1Flag = 0;

         I2C_EEPROM_Write_Data[0] = 0x00;
         I2C_EEPROM_Write_Data[1] = 0x00;
         I2C1_MasterWrite(I2C_EEPROM_Write_Data, 2, EEPROM_Address, NULL);
         DELAY_microseconds(100);
         I2C1_MasterRead(I2C_EEPROM_Read_Data, 2, EEPROM_Address, NULL);
         DELAY_microseconds(100);
         …

        sprintf((char *) StringBuffer, "I2C_W_EE:0x%02X,0x%02X", I2C_EEPROM_Write_Data[2], I2C_EEPROM_Write_Data[3]);
         printf("\033[3;1H%s", StringBuffer);
         myOLED_DrawStr(0, 2, StringBuffer);

         sprintf((char *) StringBuffer, "I2C_R_EE:0x%02X,0x%02X", I2C_EEPROM_Read_Data[0], I2C_EEPROM_Read_Data[1]);
         printf("\033[4;1H%s", StringBuffer);
         myOLED_DrawStr(0, 3, StringBuffer);

         ADC1_SoftwareTriggerEnable();
      }
    }
```

MICROCHIP

Lab16 EEPROM

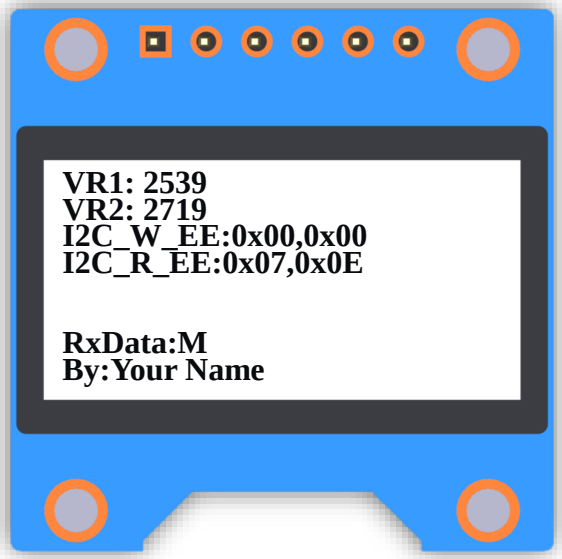
Result



COM4 - Tera Term VT

File Edit Setup Control Window Help

```
VR1:2536
VR2:2718
I2C_W_EE:0x00,0x00
I2C_R_EE:0x01,0x02

RxData:M
```



MICROCHIP

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

Lab17 Temperature Sensor

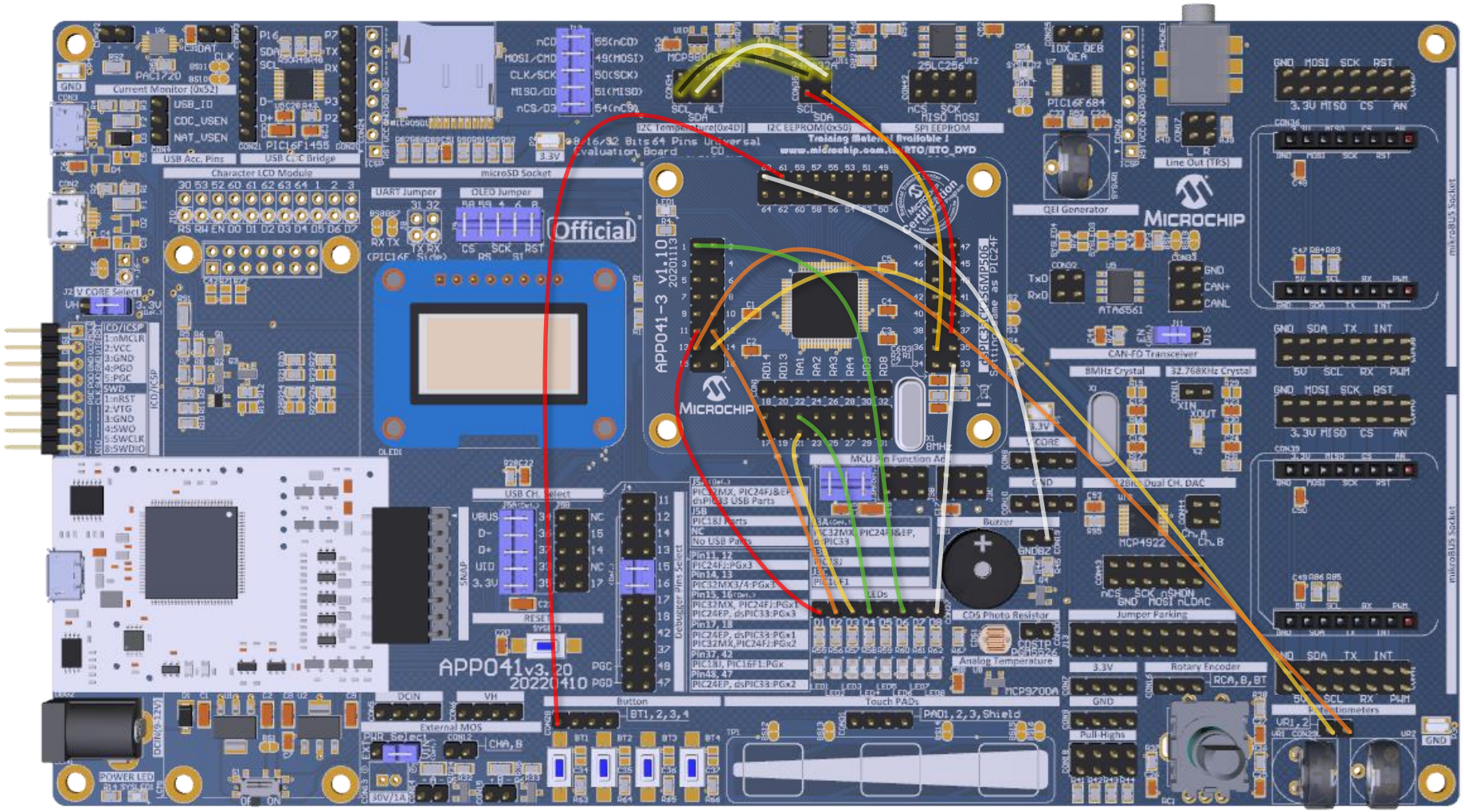
- Base on current project or Open .\Exams\Lab17_xxx.X to do exercises
- Try to set MCP9800 ADC resolution to 12 bits.
- Read the value from Temperature Sensor and Show on OLED display.

Let’s go!

MICROCHIP

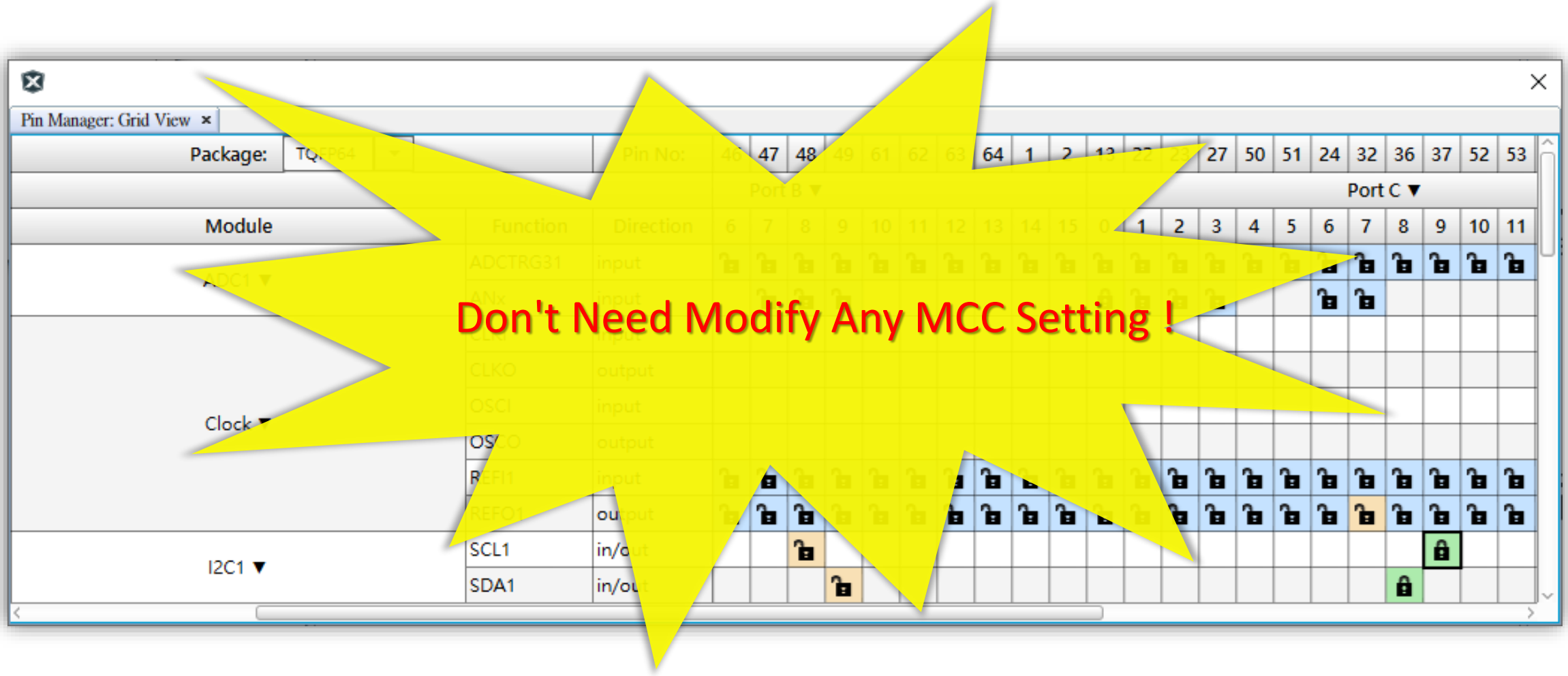
Lab17 Temperature Sensor

Connection



Lab17 Temperature Sensor

MCC Setting



MICROCHIP

Lab17 Temperature Sensor

Code Example

```
#include "mcc_generated_files/i2c1.h"

#define EEPROM_Address 0x50

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

…
uint8_t MCP9800_Write_Data[2];
uint8_t MCP9800_Read_Data[2];

int main(void)
{
    …
    myOLED_DrawStr(0, 7, (const uint8_t *) "By: Your Name");

    MCP9800_Write_Data[0] = MCP9800_REG_TA;
    MCP9800_Write_Data[1] = MCP9800_Register;
    I2C1_MasterWrite(MCP9800_Write_Data, 2, MCP9800_Address, NULL);
    DELAY_microseconds(100);
    MCP9800_Write_Data[0] = MCP9800_Start;
    I2C1_MasterWrite(MCP9800_Write_Data, 1, MCP9800_Address, NULL);
    DELAY_microseconds(100);
    while (1)
    {
        // Add your application code
        …
   }
}
```

MICROCHIP

Lab17 Temperature Sensor

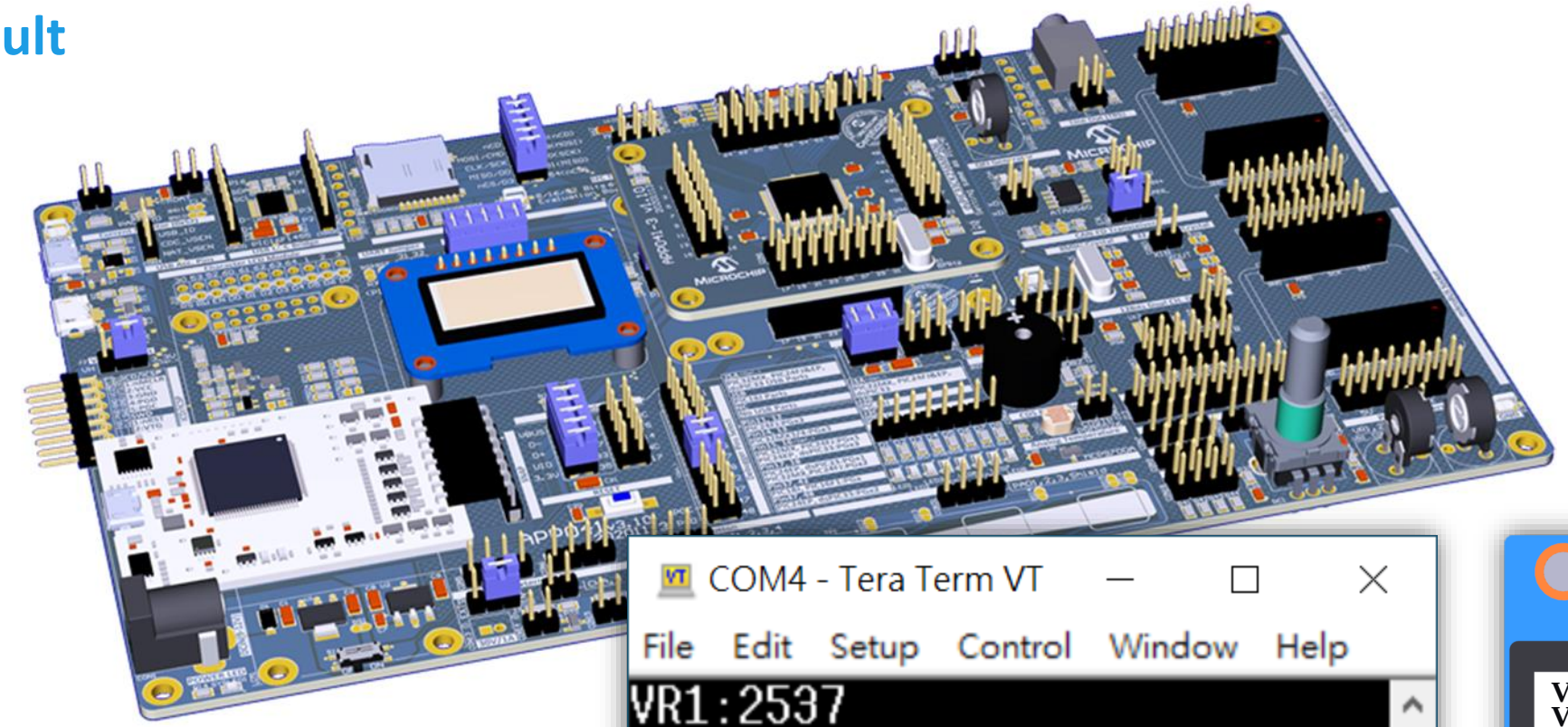
Code Example

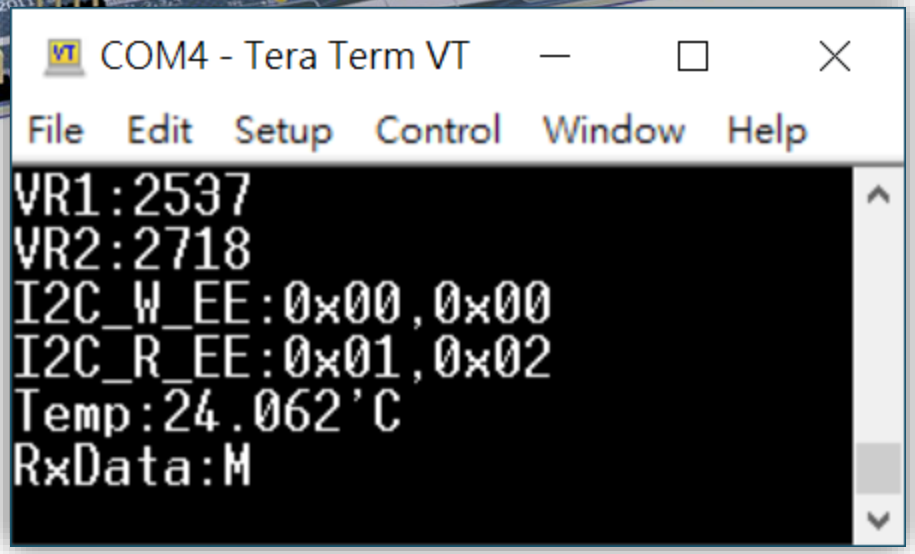
```
int main(void)
{
    ....
    while (1)
    {
        // Add your application code
        ....
        if (SCCP1Flag)
        {
            SCCP1Flag = 0;

            ...
            myOLED_DrawStr(0, 6, StringBuffer);

            I2C1_MasterRead(MCP9800_Read_Data, 2, MCP9800_Address, NULL);
            DELAY_microseconds(100);


            sprintf((char *) StringBuffer, "Temp.:%2d.%03d'C", (int) (MCP9800_Read_Data[0]&0x7F), (int) (MCP9800_Read_Data[1] >> 4)*1000 / 16);
            printf("\033[5;1H%s", StringBuffer);
            myOLED_DrawStr(0, 4, StringBuffer);

            ADC1_SoftwareTriggerEnable();
        }
    }
}
```

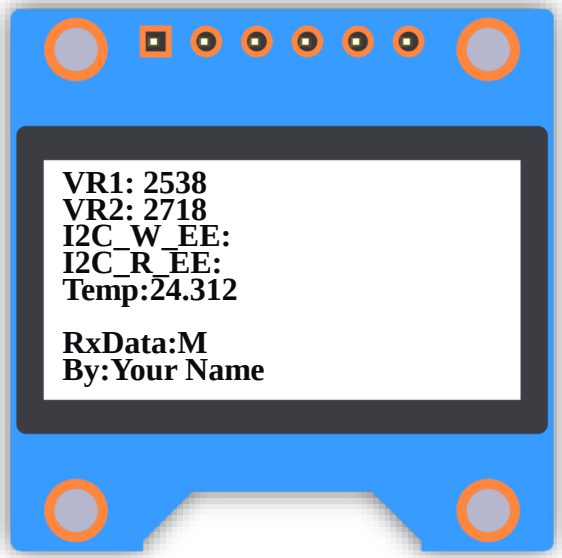
MICROCHIP

Lab17 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