

Microcontroller & MCC Melody PIC1001-Melody v1.00



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



SMART | CONNECTED | SECURE

CAE Taiwan
Microchip Technology Inc.
2023 September

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 Driver Architecture

 -  Lab5 Timer1 Interrupt

 -  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

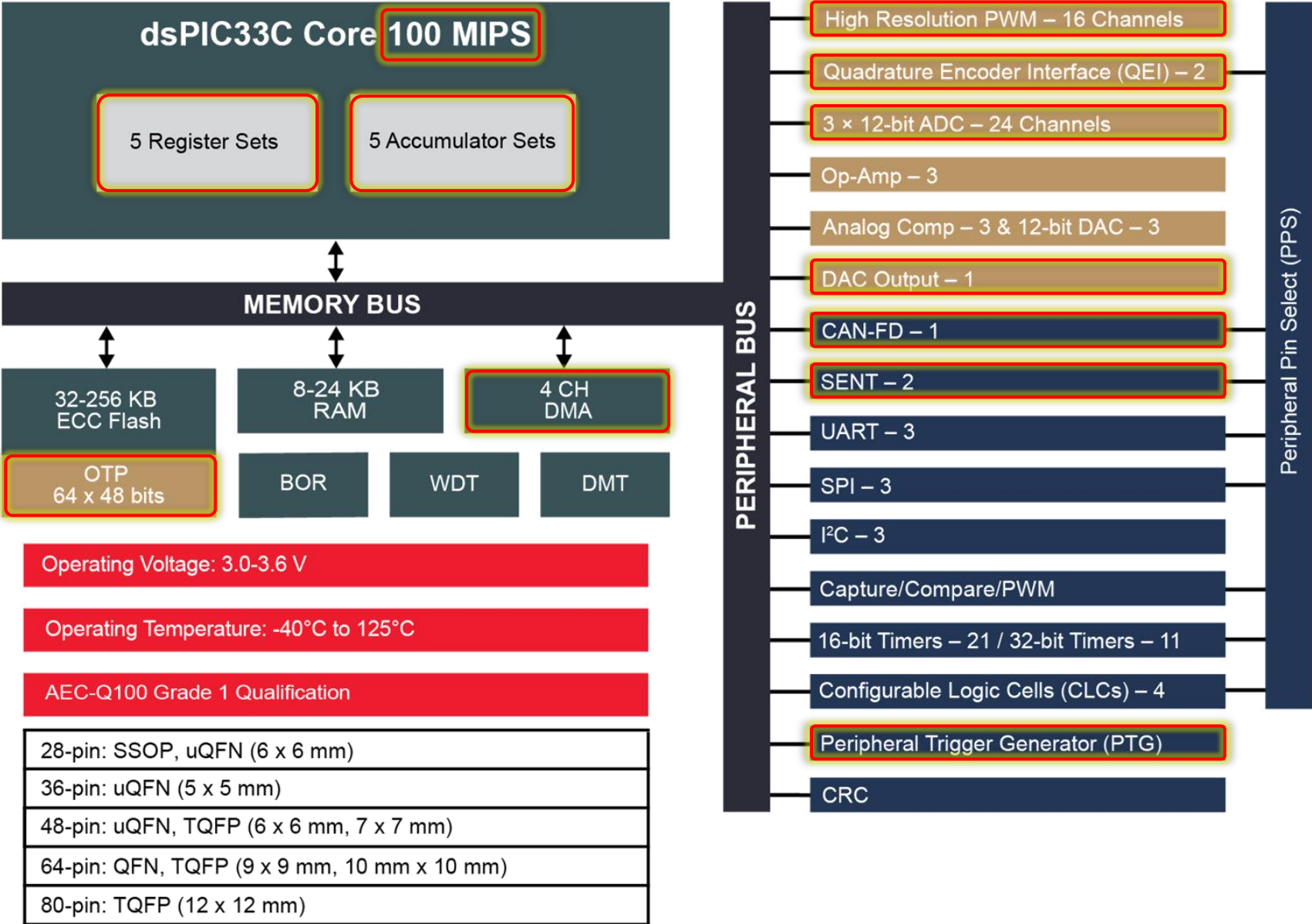
 Lab17 Temperature Sensor (MCP9800) Operation

Microchip 16-bits Microcontroller Architecture

16-bits PIC/dsPIC® MCU series

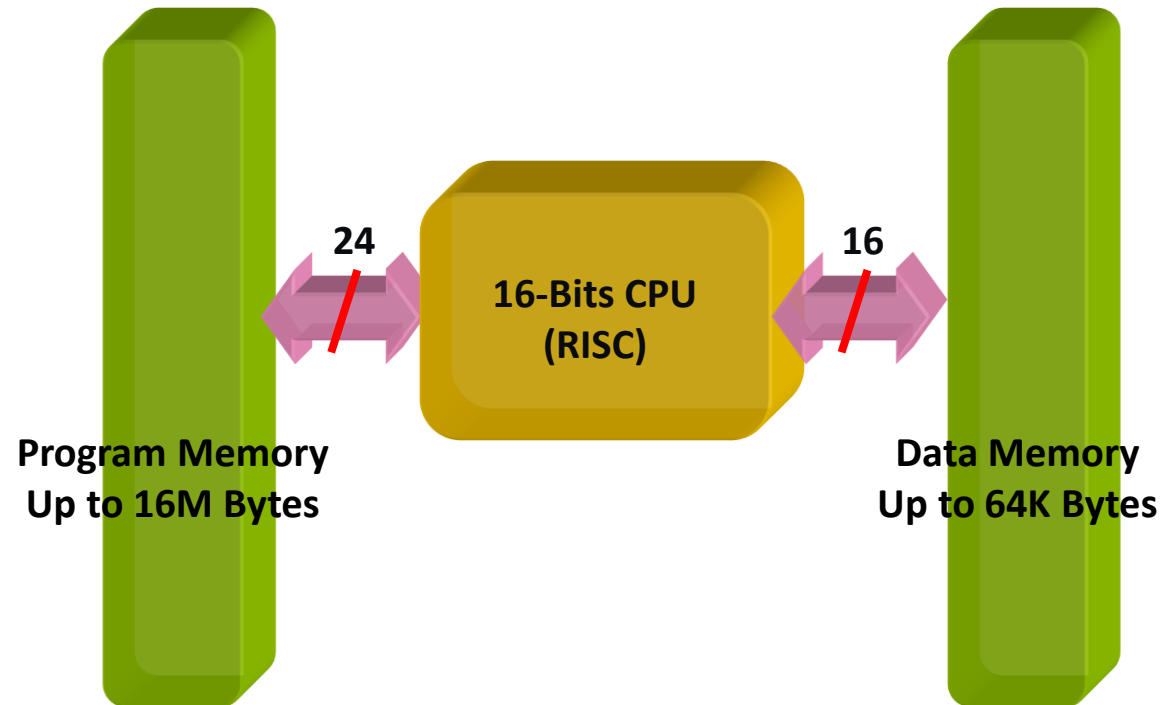
- **Microchip 16-bits MCU families include below,**
- PIC24FJ (16 MIPS)
 - Very high CP. Low price and high performance.
- PIC24HJ (40 MIPS)
 - Built-in DMA, upgrade performance to 40 MIPS.
- PIC24EP (70 MIPS)
 - Latest generation of PIC24 series, upgrade performance to 70 MIPS.
- dsPIC33FJ (40 MIPS)
 - 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.
- dsPIC33CH (90 MIPS & 100 MIPS)
 - Dual core with shared resources.

dsPIC33CK256MP508 Family

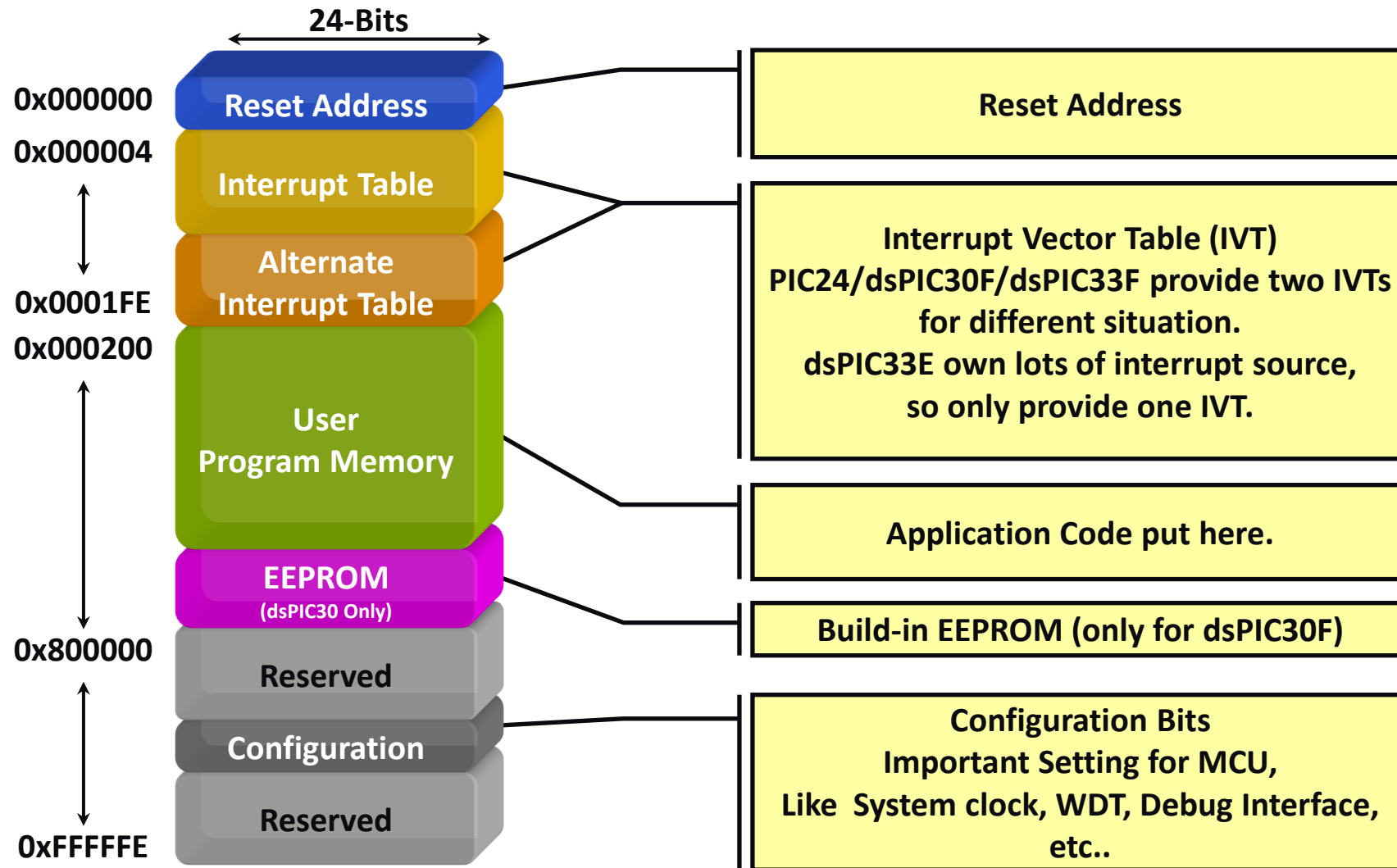


Core Architecture

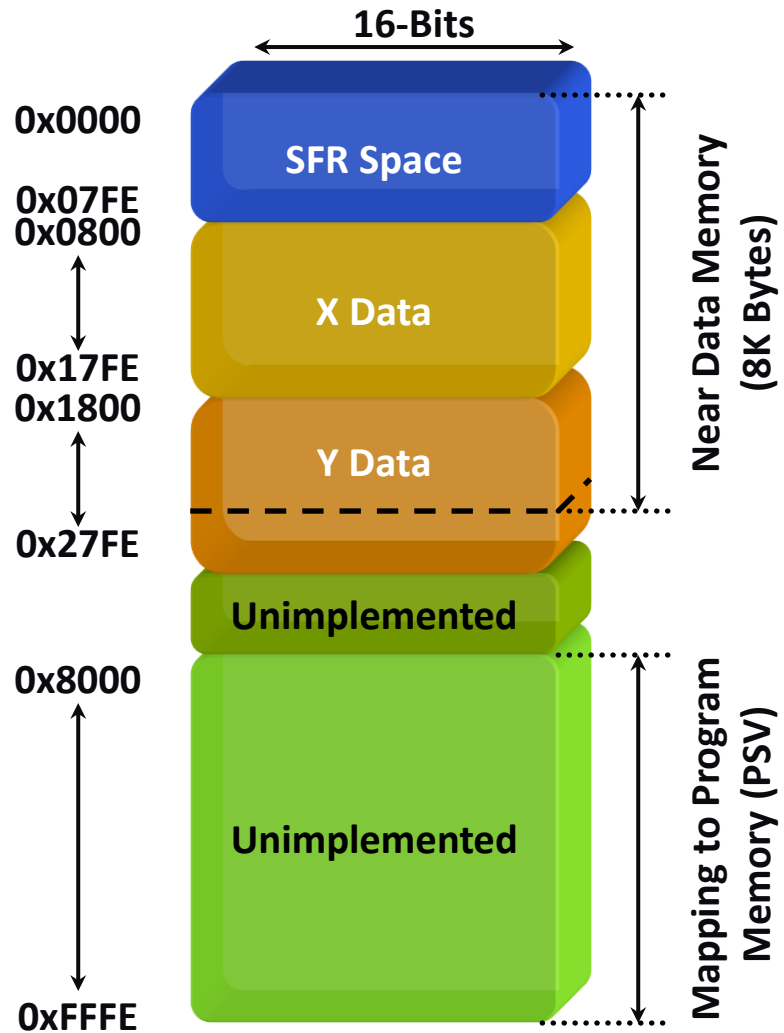
- Microchip 16-bits PIC/dsPIC® MCU is **Harvard architecture**.
- This means program and data memory bus are independently.
- Data Memory is **16-bits, Up to 64K Bytes**.
- Program Memory is **24-bits, Up to 16 M Bytes**.



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.

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**

- $1 T_{CY} = 4 T_{OSC} \cdot (F_{CY} = F_{OSC} / 4)$
Ex: 120 MHz > 30 MIPS, $T_{CY} = 33\text{nS}$.

- **PIC24F/PIC24H/dsPIC33 Families**

- $1 T_{CY} = 2 T_{OSC} \cdot (F_{CY} = F_{OSC} / 2)$
Ex: 32 MHz > 16 MIPS, $T_{CY} = 62.5\text{nS}$.

X IDE, C Compiler, MCC & Development Tools Introduction

Soft Request In this Course

- MPLAB® X IDE v6.15 <https://www.microchip.com/mplab/mplab-x-ide>
- MPLAB® XC16 v2.10 <https://www.microchip.com/mplab/compilers>
- MPLAB® Code Configurator (MCC) v5.3.7
- MCC Core v5.5.7
- MCC Melody (From X IDE MCC Content Manager)
 - MCC Melody Core v2.6.1
 - PIC24/dsPIC devices v5.10.1
 - PIC24/dsPIC Hardware Peripheral Initializer v1.0.0
 - Libraries, Drivers, PLIB and System (reference to table)
- MPLAB® X IDE Packs
 - dsPIC33CK-MP DFP v1.12.354 (Device Family Pack)
 - SNAP tool packs v2.1.1120 (Tool Firmware Pack)

Drivers		PLIB		System	
ADC Multicore	v2.1.2	ADC Multiple SARs	v2.4.2	Clock	v1.2.0
External Interrupt	v1.0.2	EXT INTERRUPT	v2.2.0	Clock PLIB	v1.4.2
I2C Host	v1.0.4	I2C	2.2.2	Configuration Bits	v1.2.1
PWM HS	v2.1.1	MCCP/SCCP	v1.6.2	ICD	v1.0.2
SPI_Host	v1.1.0	PWM	v2.3.1	Interrupt	v1.3.1
Timer	v1.1.0	SPI	v1.3.0	Main	v1.1.2
UART	v1.8.0	Timer	v1.5.2	Pins	v1.3.0
		UART	v1.4.1	Pins View	v3.7.0
				Reset	v1.1.0

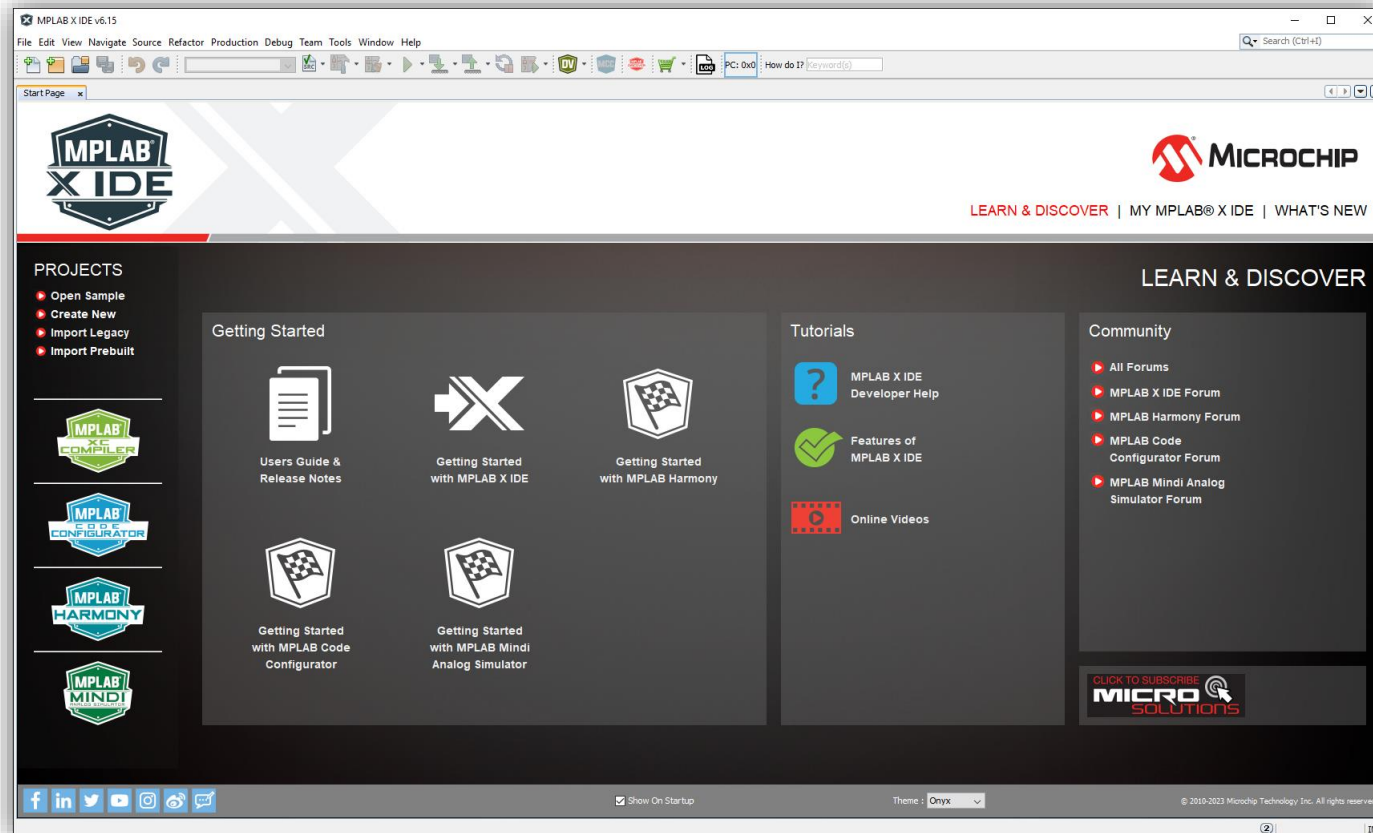
- MCC Core and Libraries Update from X IDE > MCC > Content Manager
- dsPIC33CK-MP DFP and SNAP TP Update from X IDE > Tools > Packs

(Lasted Update : 2023/08/31)

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 v6.15.



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.

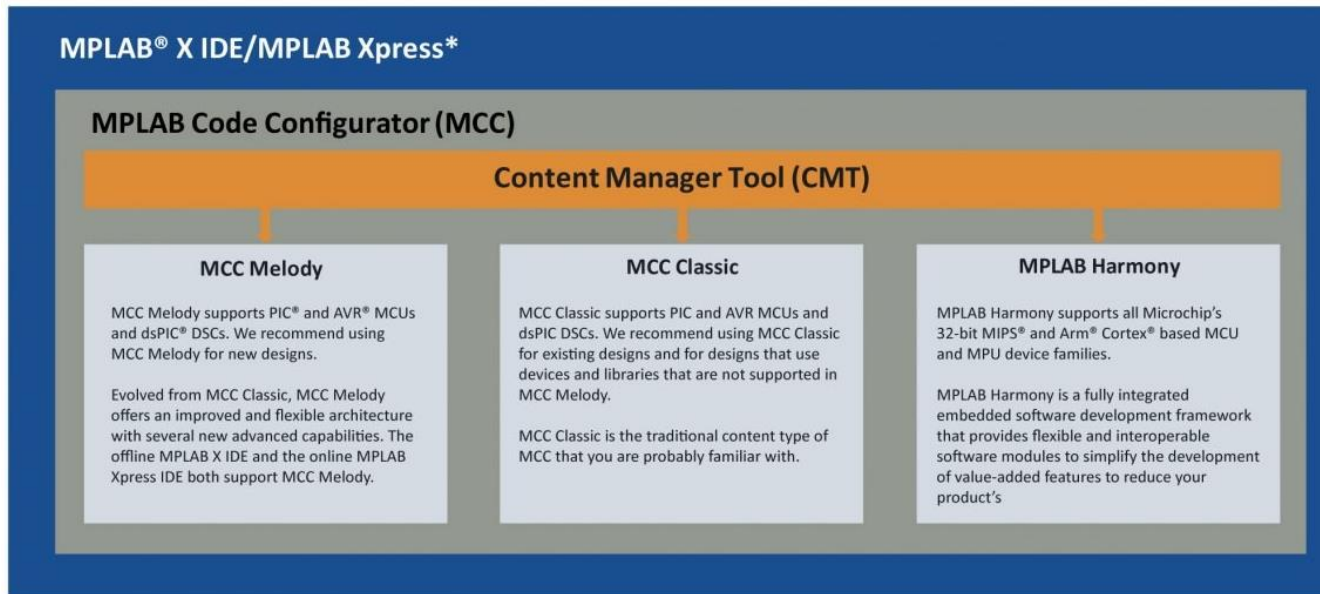
License Type	Installs On	# of Activations	# of Users	Wait Time Between Users	HPA
Workstation License	Workstation	3	1	None	Yes
Subscription License	Workstation	1	1	None	No
Site License	Network	1	Varies by Seat	None	Yes
Network Server License	Network	1	Unlimited	One Hour	Yes
Virtual Machine *	Network	1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



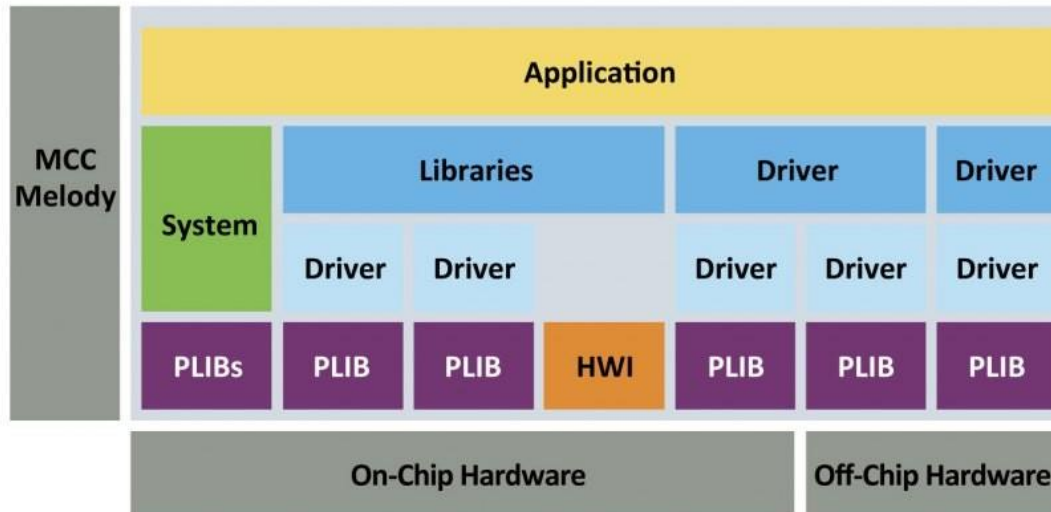
MPLAB® Code Configurator

- 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 Melody

- Evolved from MCC Classic, MCC Melody offers an improved and flexible architecture to easily configure devices, peripherals and libraries and generate code.
- MCC Melody provides libraries, drivers, Peripheral Libraries (PLIBs) and Hardware Initializers (HWIs) for the development of embedded software for our PIC® and AVR® MCUs and dsPIC® Digital Signal Controllers (DSCs).



MCC Melody GUI Quick View

Resources

Graphical

Pin Manager

Pin Assign

Builder

Icons

- Libraries/Driver: ?
- PLIB: ?
- Hardware Initializers: ?
- Hardware: ?

System Firmware

Microchip dsPIC33CK256MP506

Lab0_First_Project - Dashboard Navigator Pin Package View

Package: TQFP64

Export Image

Microchip dsPIC33CK256MP506

Pin Grid View

Package:	TQFP64	Pin No:	14	15	16	17	18	28	29	33	34	35	45	46	47	48	49	61	62	63	64	1	2	13	22	23	27	50	51	24	32	36	37	52	53	3	4	5	6	60	55	
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	
Clock	CLKO	output																																								
	REFI	input																																								
	REFO	output																																								
ICD	PGCx	input																																								
	PGDx	input																																								
Pins	GPIO	input																																								

MCC Melody GUI Quick View

Project Resources

- System
 - Clock**
 - Configuration Bits
 - DMT
 - ICD
 - Interrupt
 - Pins
 - Watchdog Timer

Device Resources

- Libraries
 - 16-Bit Data EEPROM Emulation (DEE)
 - CAN-TP
 - MCP802X
- Drivers
 - ADC
 - CAN FD

Lab0_First_Project - Dashboard

Package: TQFP64

Pin Package View

Microchip dsPIC33CK256MP506

System Clock

- System Clock Source: FRC Oscillator
- Set System(FOSC) Frequency to Maximum: ☐
- Calculated System Frequency(FOSC): 8.00 MHz
- Calculated Peripheral Frequency(FOSC/2): 4.00 MHz
- PLL Output Frequency(FPLO): 1200.00 MHz
- VCO Frequency(FVCO): 1200.00 MHz
- FVCO Divider: 4
- VCO Divider Frequency(FVCO DIV): 300.00 MHz

Auxiliary Clock

- Auxiliary Clock Source: FRC Oscillator
- Auxiliary PLL Output Frequency(AFPLLO): 8.00 MHz

Reference Clock

- Reference Clock Enable: ☐
- REFO Clock Source: FOSC
- Requested REFO Frequency(in Hz): 8000000
- Calculated REFO Frequency: 0.00 kHz

Output - MPLAB® Code Configurator

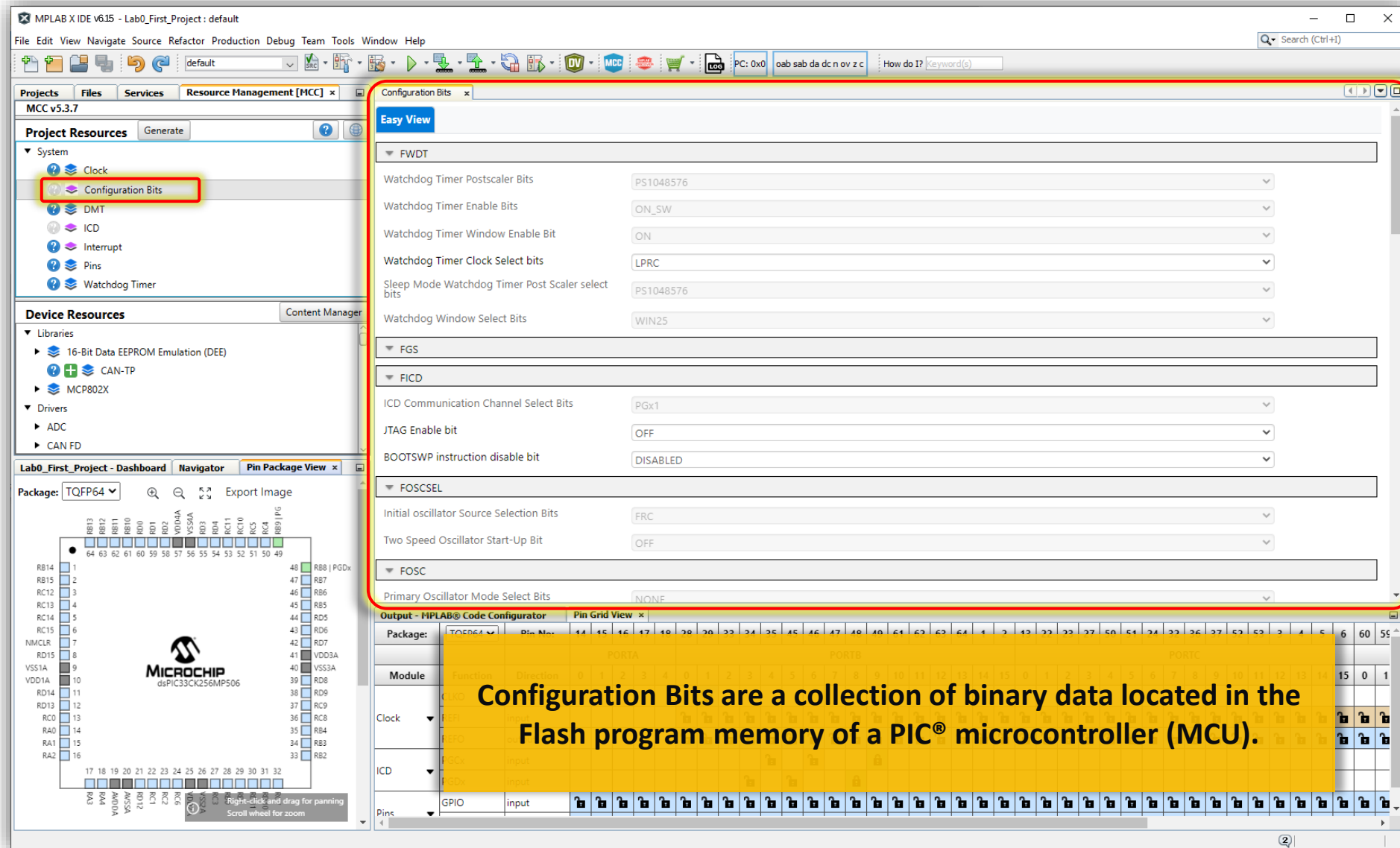
Package: TQFP64

Pin Grid View

Module	Function	Direction	Pin
Clock	CLKO	output	14
	REFI	input	15
ICD	PGCx	input	16
	PGDx	input	17
GPIO	GPIO	input	18
	GPIO	input	19

System clock Graphically.
Configuration Word settings are clearer and simpler.

MCC Melody GUI Quick View



MCC Melody GUI Quick View

The screenshot displays the MPLAB X IDE v6.15 interface with the MCC v5.3.7 configuration tool. The 'Resource Management [MCC]' tab is active, showing 'Project Resources' on the left. The 'DMT' resource is highlighted with a red box. The 'DMT' configuration window is open, showing the 'Easy View' tab. The 'Settings' section is expanded, displaying the following configuration:

- DMT Enable Type: In Software mode, DMT can only be enabled in Software and cannot be disabled. In Hardware mode DMT is always enabled. (Dropdown menu set to Software)
- DMT Fetch Counter: 0x0 <= 0x0 (Range: 0x0 to 0xFFFFFFFF)
- DMT Window Interval: 0x0 <= 0x0 (Range: 0x0 to 0xFFFFFFFF)

A yellow text box is overlaid on the right side of the interface, containing the text:

Deadman Timer Graphically.
Interrupt the processor in the event of a software malfunction.

The bottom of the interface shows the 'Lab0_First_Project - Dashboard' and 'Pin Package View' tabs. The 'Pin Package View' tab is active, displaying a pin package diagram for the dsPIC33CK256MP506. The package is labeled 'TQFP64' and shows the pin connections for the device.

Output - MPLAB® Code Configurator

Pin Grid View

Package:	TQFP64	Pin No:	14	15	16	17	18	28	29	33	34	35	45	46	47	48	49	61	62	63	64	1	2	13	22	23	27	50	51	24	32	36	37	52	53	3	4	5	6	60	55	
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	
Clock	CLKO	output																																								
	REFI	input																																								
ICD	PGCx	input																																								
	PGDx	input																																								
GPIO	PGDx	input																																								

MCC Melody GUI Quick View

The screenshot displays the MPLAB X IDE v6.15 interface with the MCC (Melody Code Configurator) window open. The 'Resource Management [MCC]' tab is active, showing a tree view of project resources. The 'ICD' resource is highlighted with a red box. The 'Easy View' tab is selected, showing the 'Software Settings' section with 'Emulator Pin Placement' set to 'Communicate on PGC1 and PGD1'. A yellow box highlights the 'ICD' resource in the tree and the 'Easy View' tab. A large yellow box with black text is overlaid on the center of the screen, stating: 'ICD assign Graphically. Debugger and in-circuit debugging functionality pin.'

**ICD assign Graphically.
Debugger and in-circuit debugging functionality pin.**

The bottom of the screenshot shows the 'Pin Package View' for the dsPIC33CK256MP506 package. It displays a pin grid with various pins and their functions. The 'Output - MPLAB® Code Configurator' window is also visible, showing the 'Pin Grid View' with a table of pin assignments.

Module	Function	Direction	Pin No.
Clock	CLKO	output	1
	REFI	input	1
	REFO	output	1
ICD	PGCx	input	1
	PGDx	input	1
Pins	GPIO	input	1

MCC Melody GUI Quick View

Project Resources

- System
 - Clock
 - Configuration Bits
 - DMT
 - ICD
 - Interrupt**
 - Pins
 - Watchdog Timer

Device Resources

- Libraries
 - 16-Bit Data EEPROM Emulation (DEE)
 - CAN-TP
 - MCP802X
- Drivers
 - ADC
 - CAN FD

Interrupt Settings

Global Interrupt Enable: ☒

CTXT1: IPL for Alternate Working Register 1: OFF

CTXT2: IPL for Alternate Working Register 2: OFF

CTXT3: IPL for Alternate Working Register 3: OFF

CTXT4: IPL for Alternate Working Register 4: OFF

Module	Interrupt	Description	IRQ Number	Enabled	Priority(IPL)	Context(CTXTn)
DMT	DMT	Dead Man Timer	45	☒	1	☐
Pins	CNA	Change Notification A	2	☐	1	☐
Pins	CNB	Change Notification B	3	☐	1	☐
Pins	CNC	Change Notification C	19	☐	1	☐
Pins	CND	Change Notification D	75	☐	1	☐

Pin Package View

Package: TQFP64

Export Image

Microchip dsPIC33CK256MP506

Right-click and drag for panning
Scroll wheel for zoom

Output - MPLAB® Code Configurator

Package: TQFP64

Module: GPIO

Function: input

Direction: input

Interrupt assign Graphically.
ISR related code generate automatically.

MCC Melody GUI Quick View

MPLAB X IDE v6.15 - Lab0_First_Project : default

File Edit View Navigate Source Refactor Production Debug Team Tools Window Help

default PC: 0x0 oab sab da dc n ov z c How do I? Keyword(s)

Projects **Files** **Services** **Resource Management [MCC]**

MCC v5.3.7

Project Resources Generate

- System
 - Clock
 - Configuration Bits
 - DMT
 - ICD
 - Interrupt
 - Pins**
 - Watchdog Timer

Device Resources Content Manager

- Libraries
 - 16-Bit Data EEPROM Emulation (DEE)
 - CAN-TP
 - MCP802X
- Drivers
 - ADC
 - CAN FD

Lab0_First_Project - Dashboard Navigator **Pin Package View**

Package: TQFP64 Export Image

Microchip dsPIC33CK256MP506

Right-click and drag for panning
Scroll wheel for zoom

Pins

Location	Pin Name	Module	Function	Direction	Custom Name	Analog	Start High	Weak Pulldown	Weak Pullup	Open Drain	Interrupt on Change
49	RB9	ICD	PGCx	input		☐	☐	☐	☐	☐	none
48	RB8	ICD	PGDx	input		☐	☐	☐	☐	☐	none

Rename or change settings for selected pins.

**Pin assign Graphically.
PPS Code generate automatically.**

Output - MPLAB® Code Configurator **Pin Grid View**

Package: TQFP64 Pin No: 14 15 16 17 18 28 29 33 34 35 45 46 47 48 49 61 62 63 64 1 2 13 22 23 27 50 51 24 32 36 37 52 53 3 4 5 6 60 55

Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	
Clock	CLKO	output																																								
	REF1	input																																								
	REFO	output																																								
ICD	PGCx	input																																								
	PGDx	input																																								
Pins	GPIO	input																																								

MCC Melody GUI Quick View

Watchdog Timer Graphically.
Used to detect system software malfunctions.

Output - MPLAB® Code Configurator

Package:	Pin No:	14	15	16	17	18	28	29	33	34	35	45	46	47	48	49	61	62	63	64	1	2	13	22	23	27	50	51	24	32	36	37	52	53	3	4	5	6	60	55	
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1
Clock	CLKO	output																																							
	REFI	input																																							
ICD	REFO	output																																							
	PGCx	input																																							
Pins	PGDx	input																																							
	GPIO	input																																							

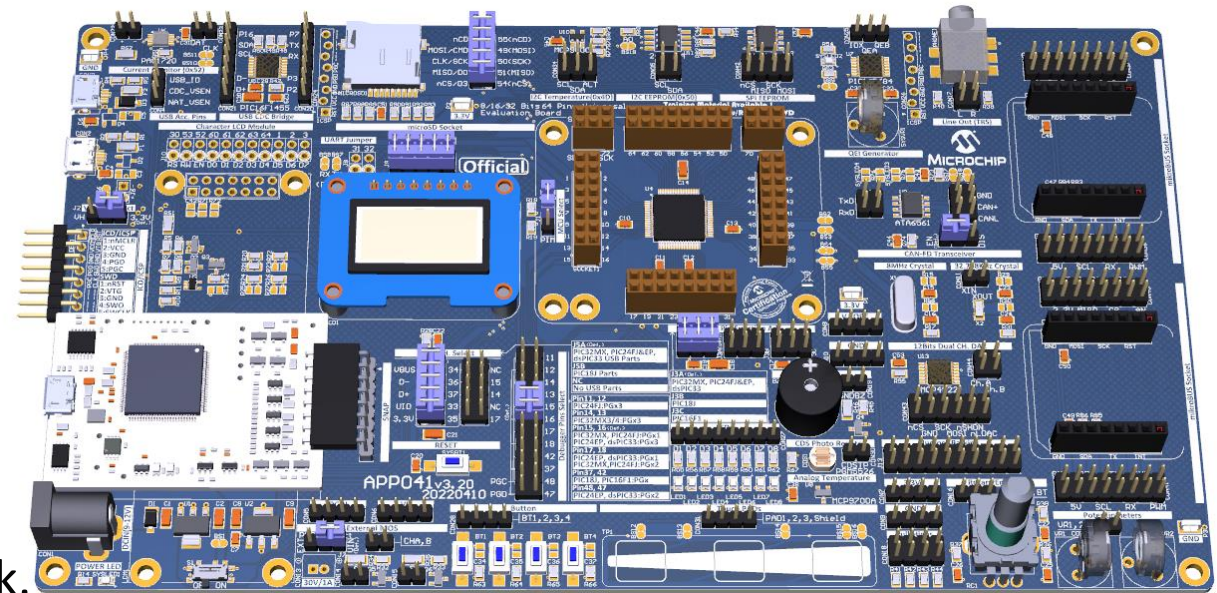
More ?

www.microchip.com/mcc

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

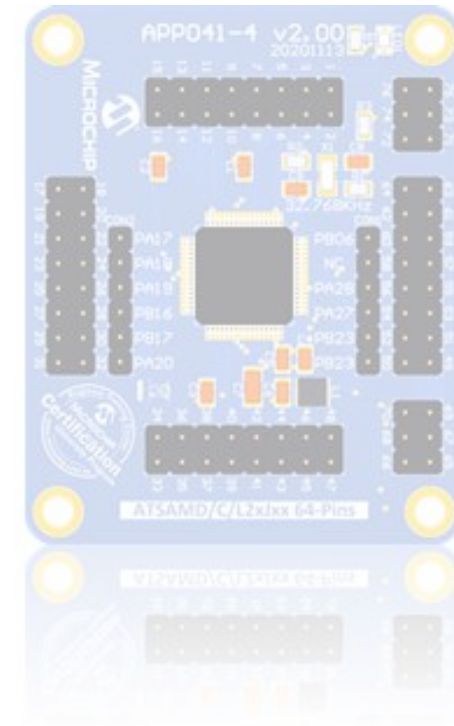
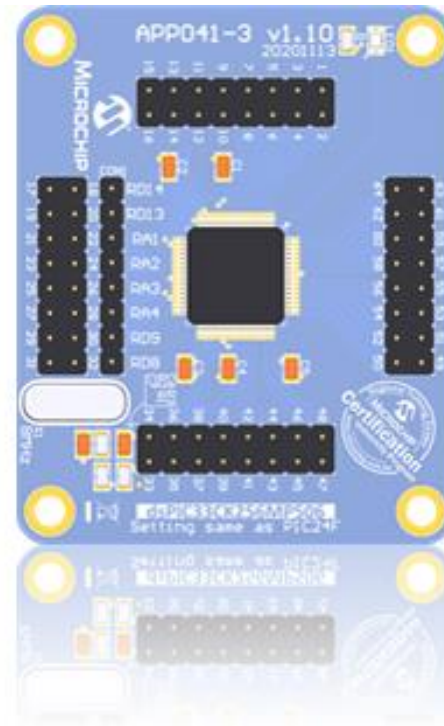
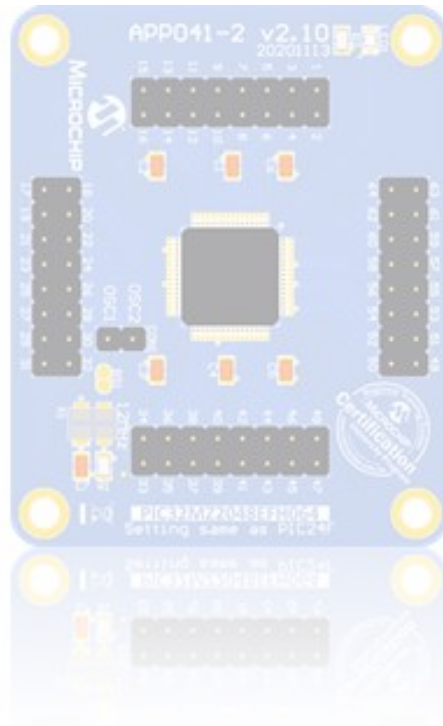
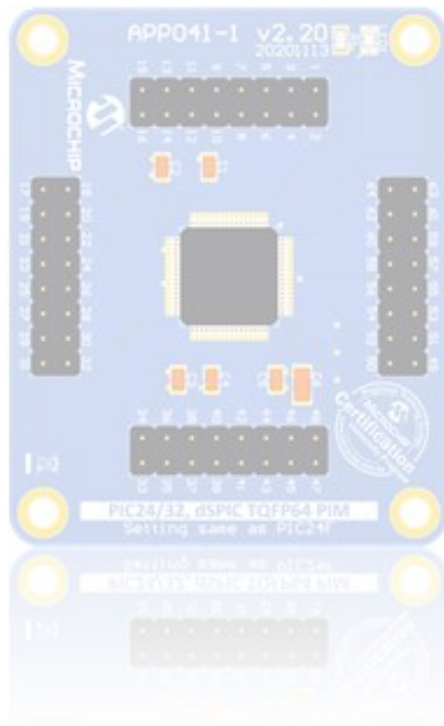
8, 16, 32-bits General Purpose EVM

- **APP041 v3.20** 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.



PIM, Plug-In Module

- Rich PIMs to support different microcontroller.
- It's included PIC18, PIC24, **dsPIC33**, PIC32, SAMD21, SAMD51, SAML and SAMC series.

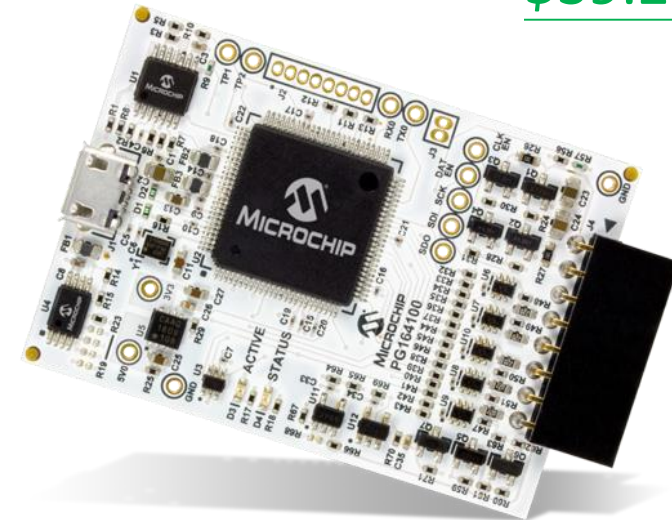


Programmer/Debugger

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



Programmer/Debugger

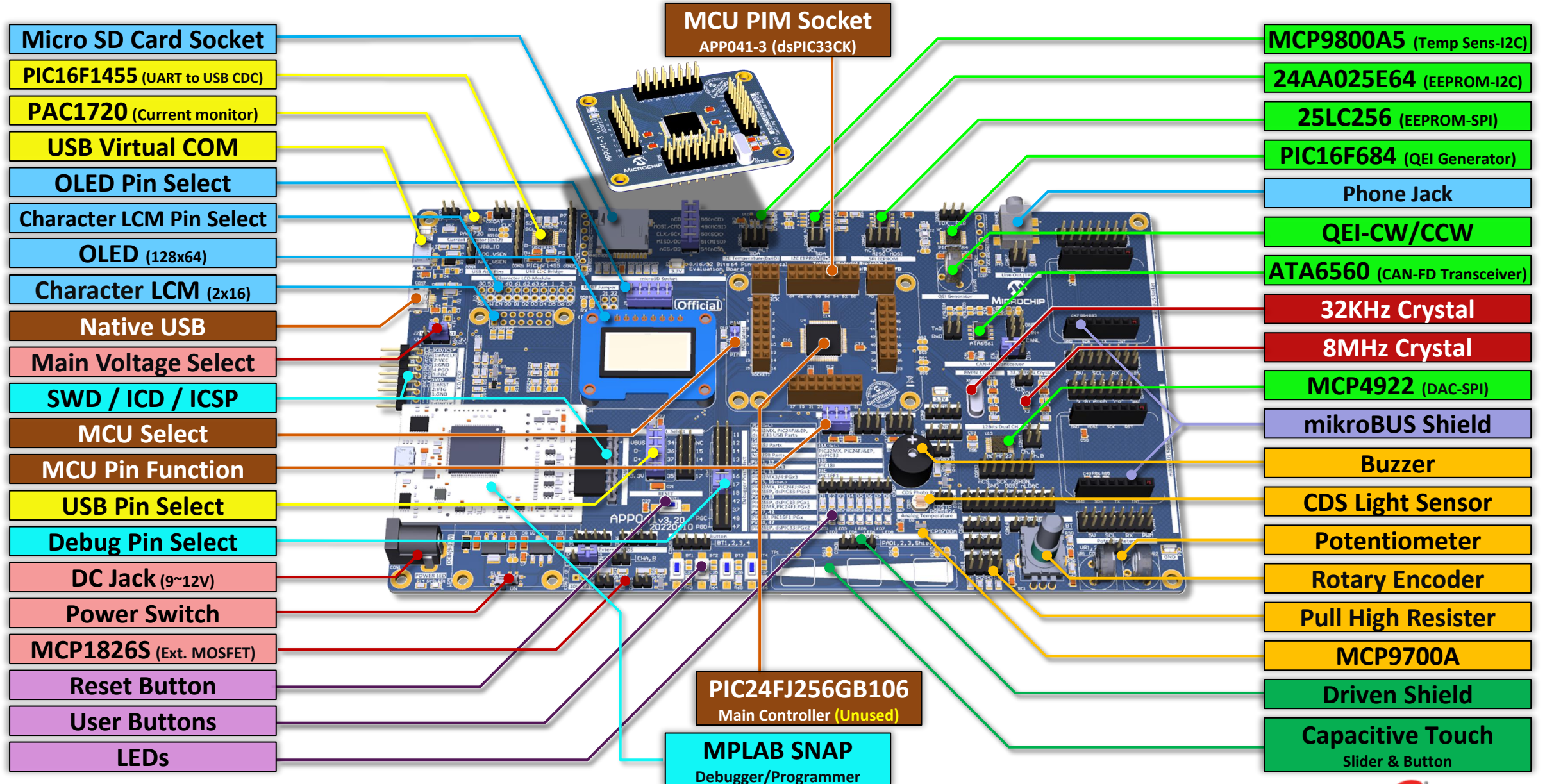
MPLAB® PICKit 5 (Part No. PG164150)

- The MPLAB® PICKit™ 5 In-Circuit Debugger/Programmer allows fast and easy debugging and programming of all PIC®, dsPIC®, AVR, SAM and CEC flash microcontrollers.
- **Features**
 - Improved Programmer-to-Go (PTG) support.
 - Connect wirelessly from your smartphone via Bluetooth®
 - Select from multiple saved program images on the SD Card
 - Start programming from the app or by pressing on the logo
 - Supply 150 mA to the target.
 - Option to be self-powered from the target (2.7V to 5V).
 - USB Type-C® connector and cable.
 - No external power needed when the device is powered.
 - Use the 8-pin single in-line header.



\$94.99

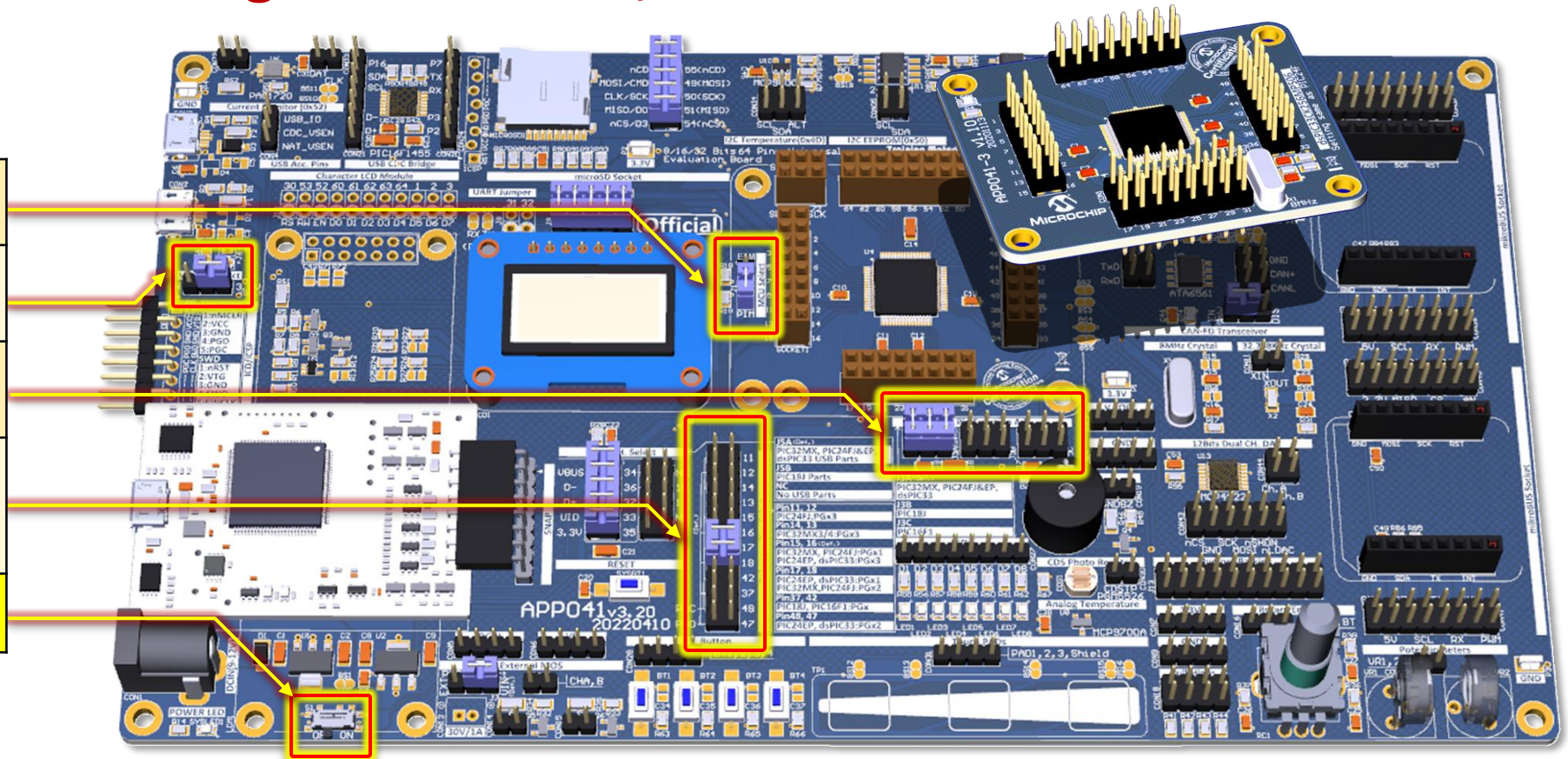
Components



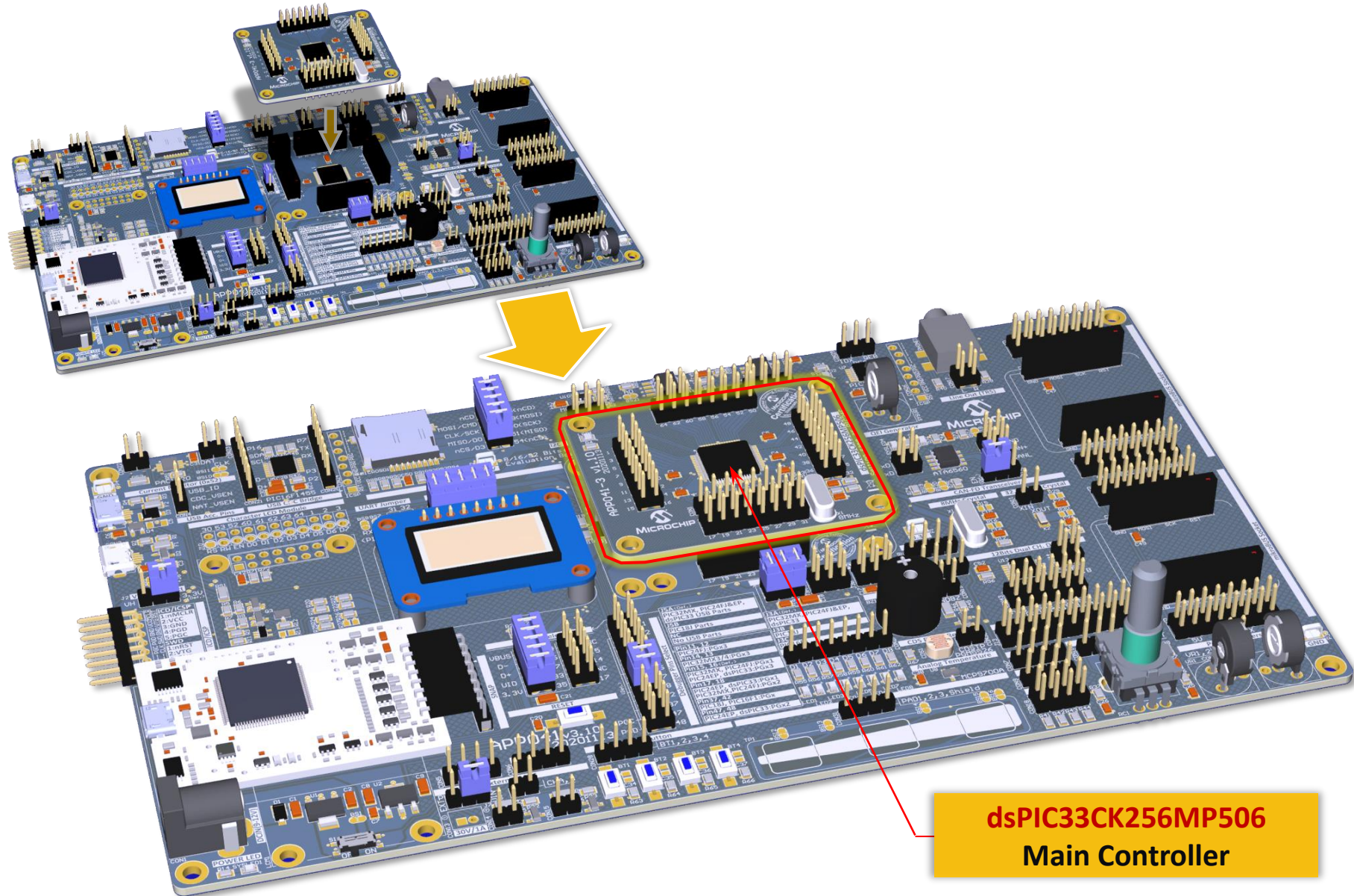
Caution !

- 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

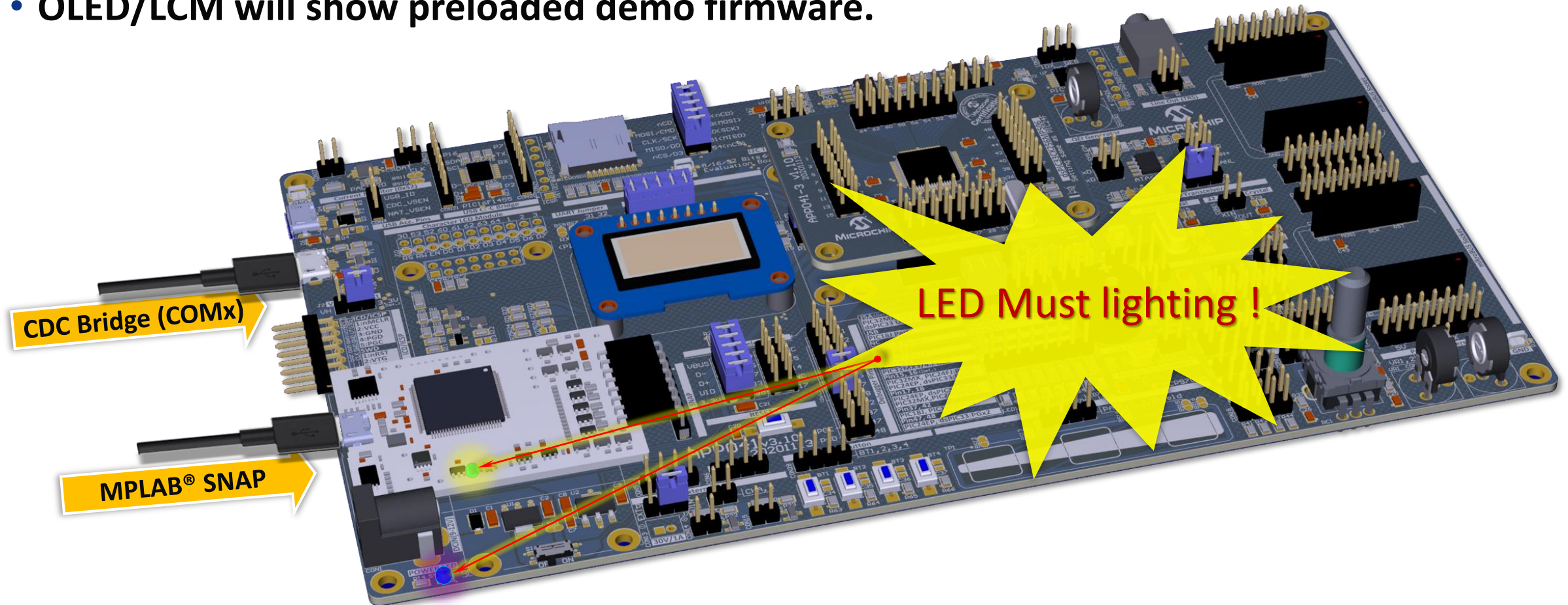


Plug-In Module Assembly



Power On & Getting started

- Just plug-in micro-USB cable to COMx 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.

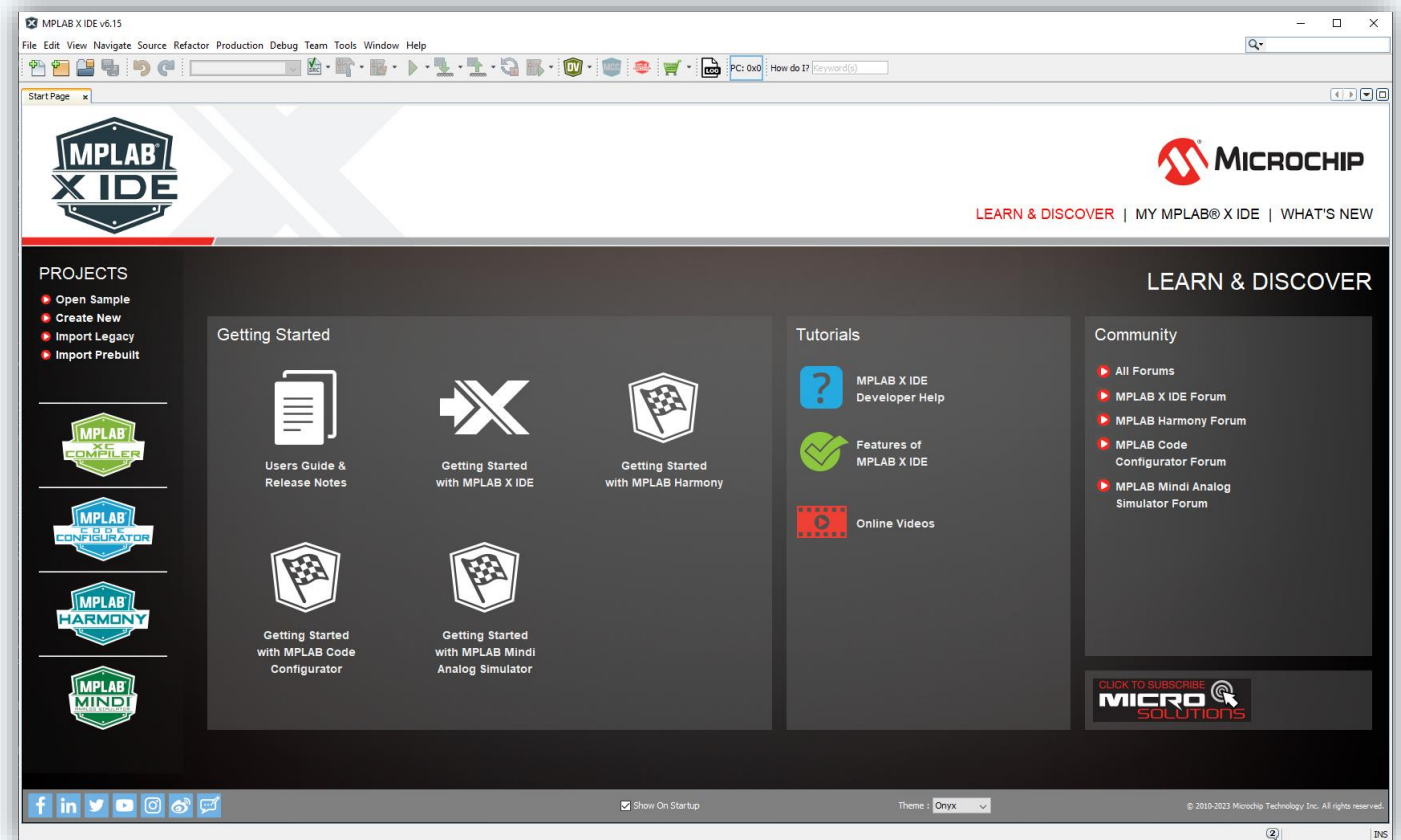


Lab Preparation

Check Compiler & MCC version

Development Tools Check

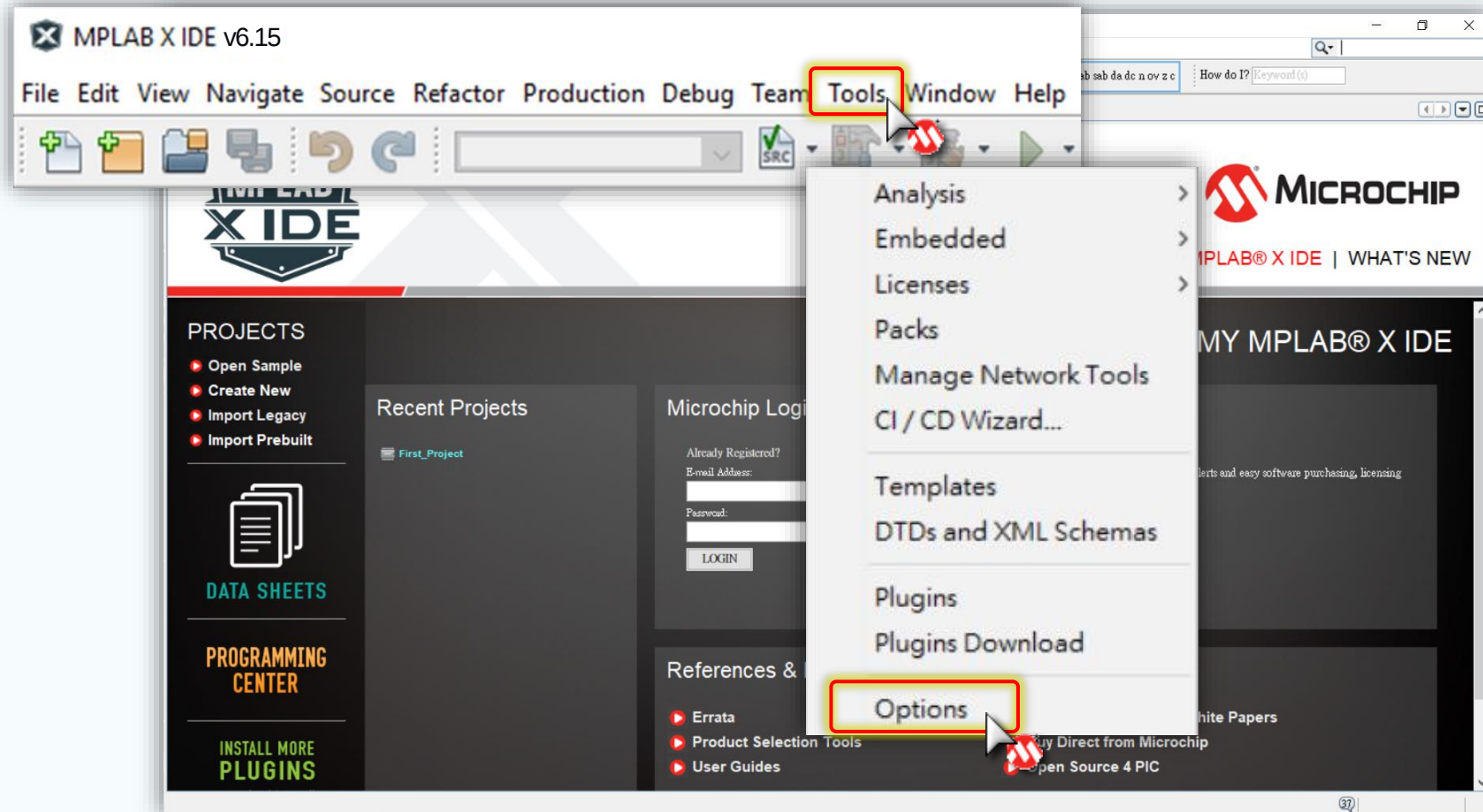
- Open **MPLAB X IDE v6.15** firstly.



Development Tools Check

Check Compiler Version

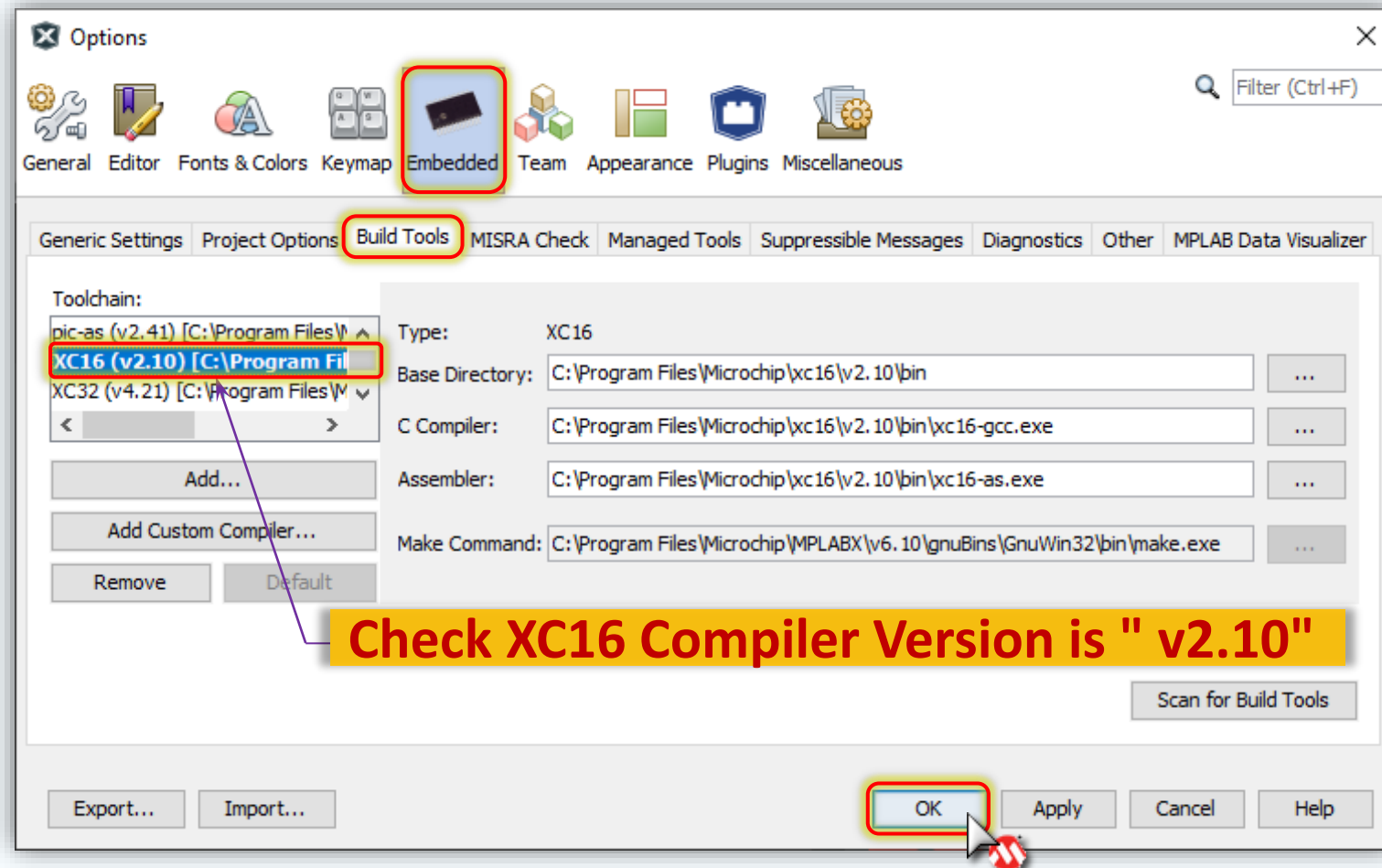
- Select : **Tools > Options**



Development Tools Check

Check Compiler Version

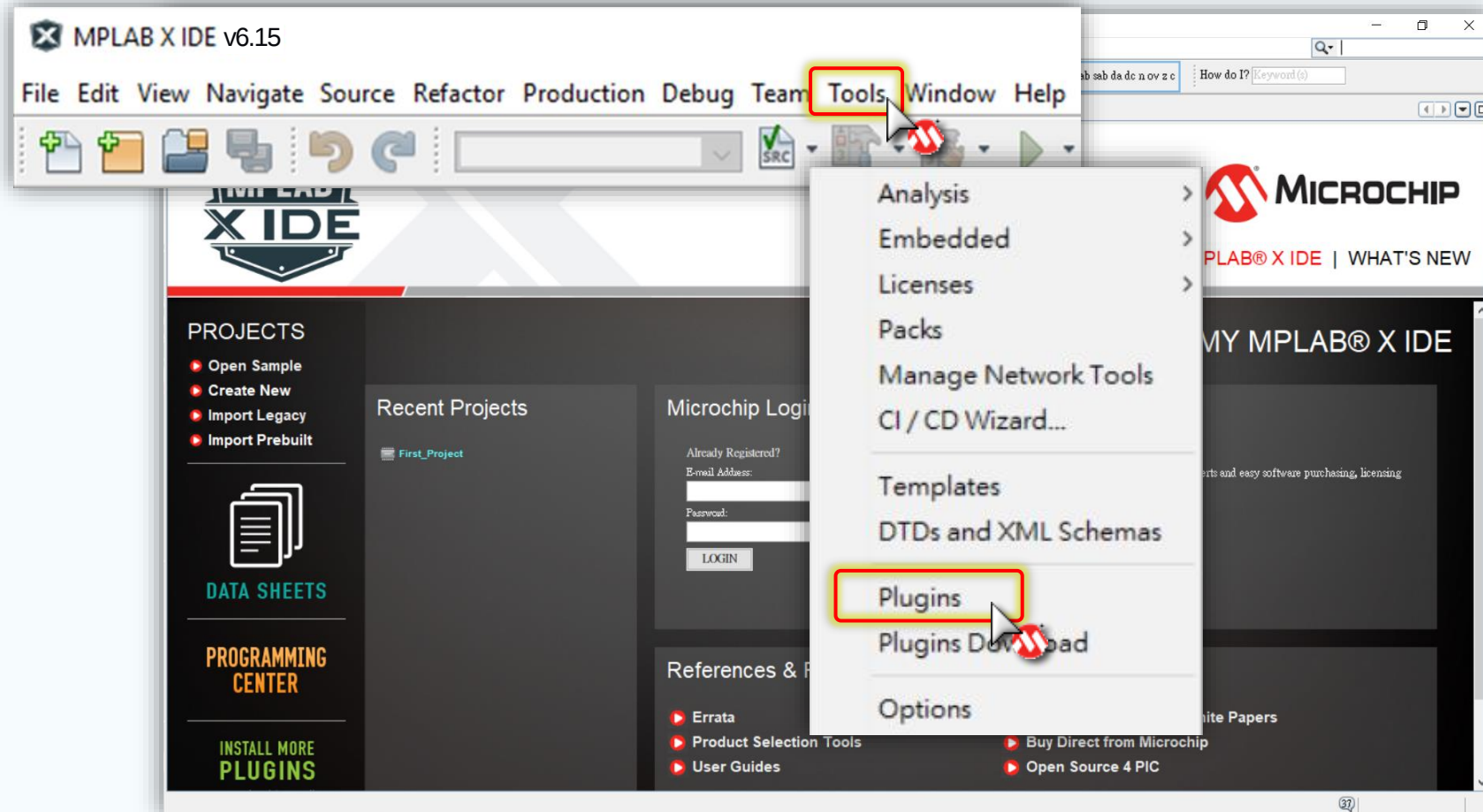
- Select : **Embedded > Build Tools**



Development Tools Check

Check MCC Version

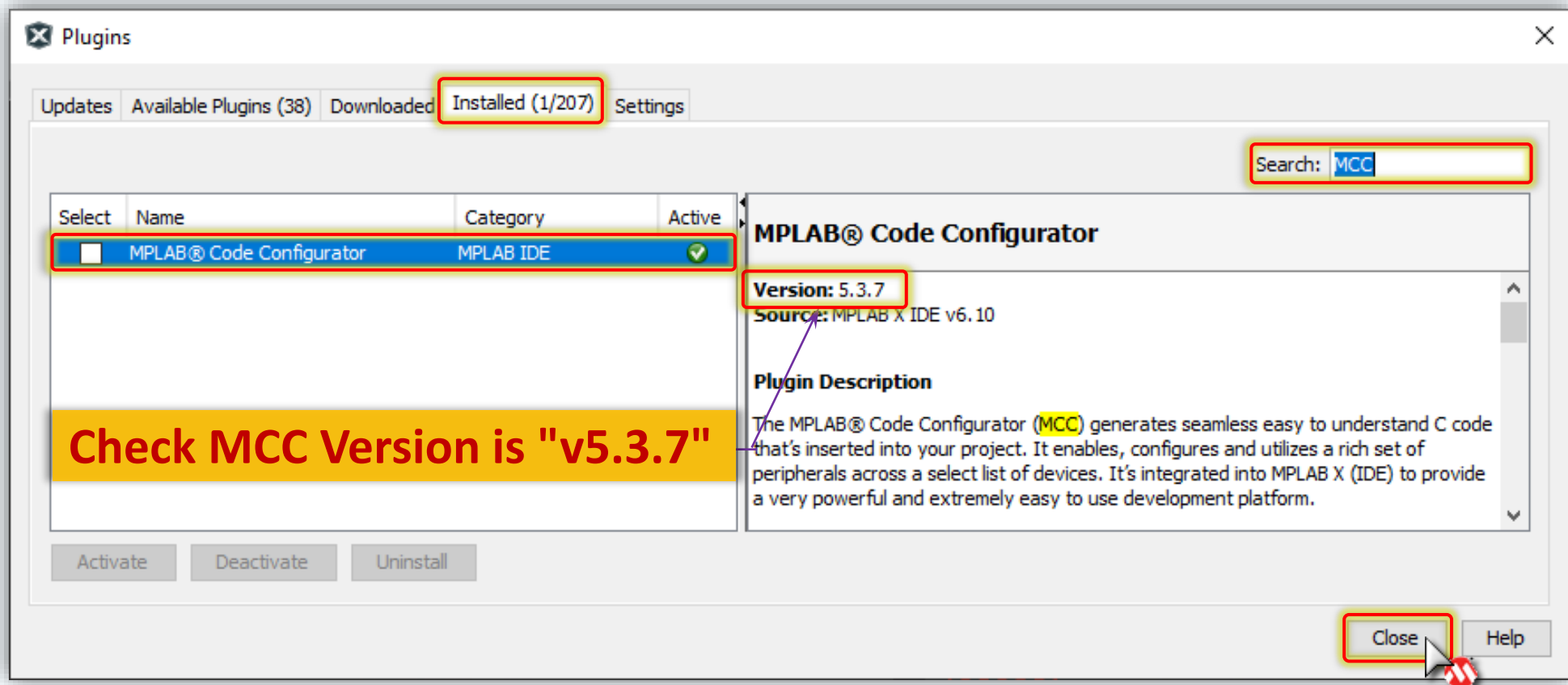
- Select : **Tools > Plugins**



Development Tools Check

Check MCC Version

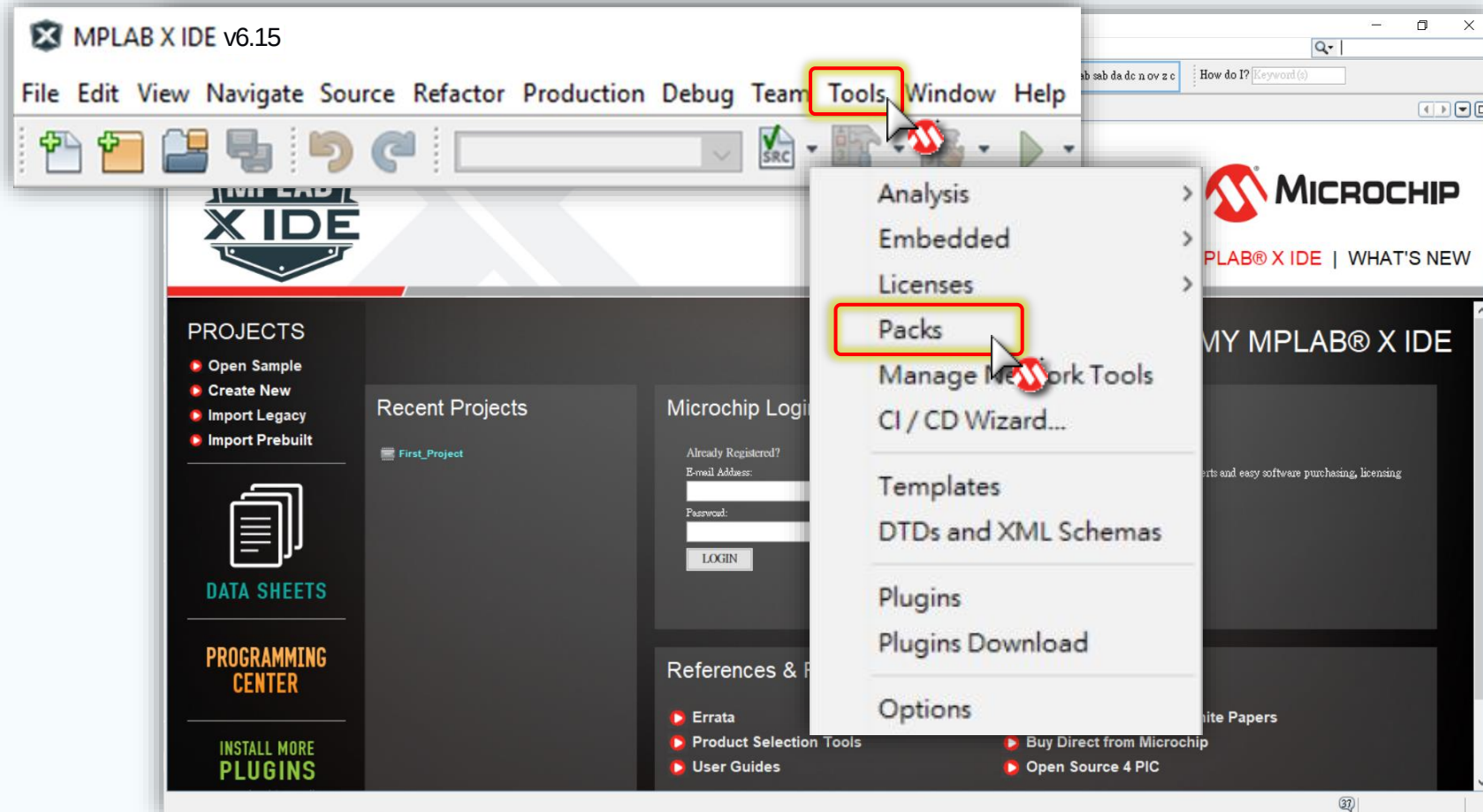
- Select : **Installed**
 - Search : "**MCC**"



Development Tools Check

Check Packs Version

- Select : **Tools > Plugins**



Development Tools Check

Check Packs Version

- Select : **Device Family Packs**
 - Search : "**dsPIC33CK-MP**"

The screenshot shows the 'Packs - Editor' window in MPLAB X IDE. The 'Packs' tab is active, displaying a list of installed packs. The 'Device Family Packs (39)' tab is selected, and the 'dsPIC33CK-MP_DFP' pack is highlighted. The pack details show version '1.12.354' and a status of 'Installed'. A yellow box highlights the pack name and version, and a red box highlights the '1.12.354' version number. A purple arrow points from the yellow box to the red box. A yellow banner at the bottom of the window contains the text: 'Check DFP "v1.12.354" version is Installed.'

Packs - Editor

Pack Manager adds and removes device support for MPLAB X IDE/IPE

3306 devices supported by 121 installed packs
34 pack updates available
5 new packs available
Last checked for updates 2023-09-12T10:15:38.445

Check for Updates Install or Uninstall Packs Show Packs Compatible with MPLAB X IDE 6.15 dsPIC33CK-MP Show Logs

Device Family Packs (39) Tool Packs

dsPIC33CK-MP_DFP

1.12.354 - - Updated DMAINTx mask va... Installed

12 more pack releases

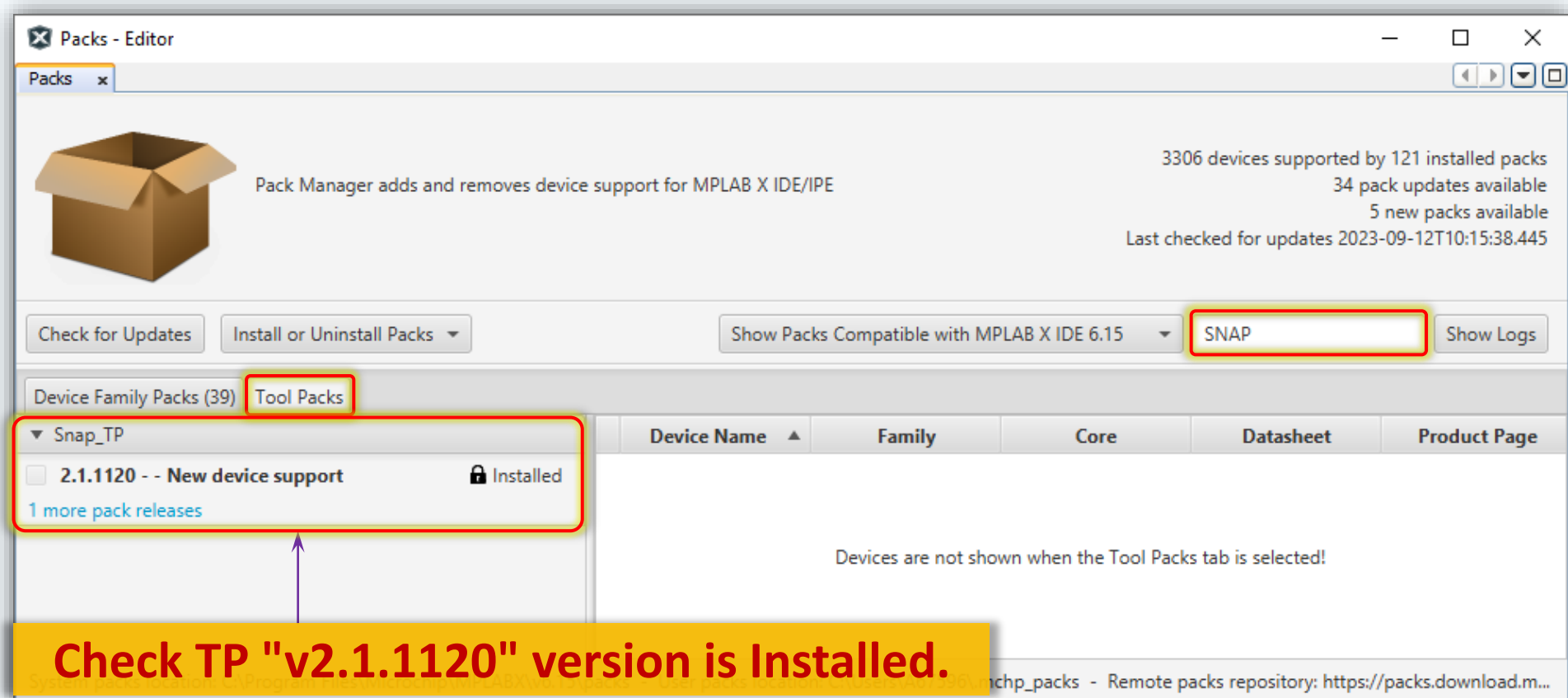
Device Name	Family	Core	Datasheet	Product Page
dsPIC33CK1024MP...	dsPIC33CK-MP	33xxxx	Datasheet	Product Page
dsPIC33CK1024MP...	dsPIC33CK-MP	33xxxx	Datasheet	Product Page
dsPIC33CK1024MP...	dsPIC33CK-MP	33xxxx	Datasheet	Product Page
dsPIC33CK1024MP...	dsPIC33CK-MP	33xxxx	Datasheet	Product Page

chp_packs - Remote packs repository: https://packs.download.m...

Development Tools Check

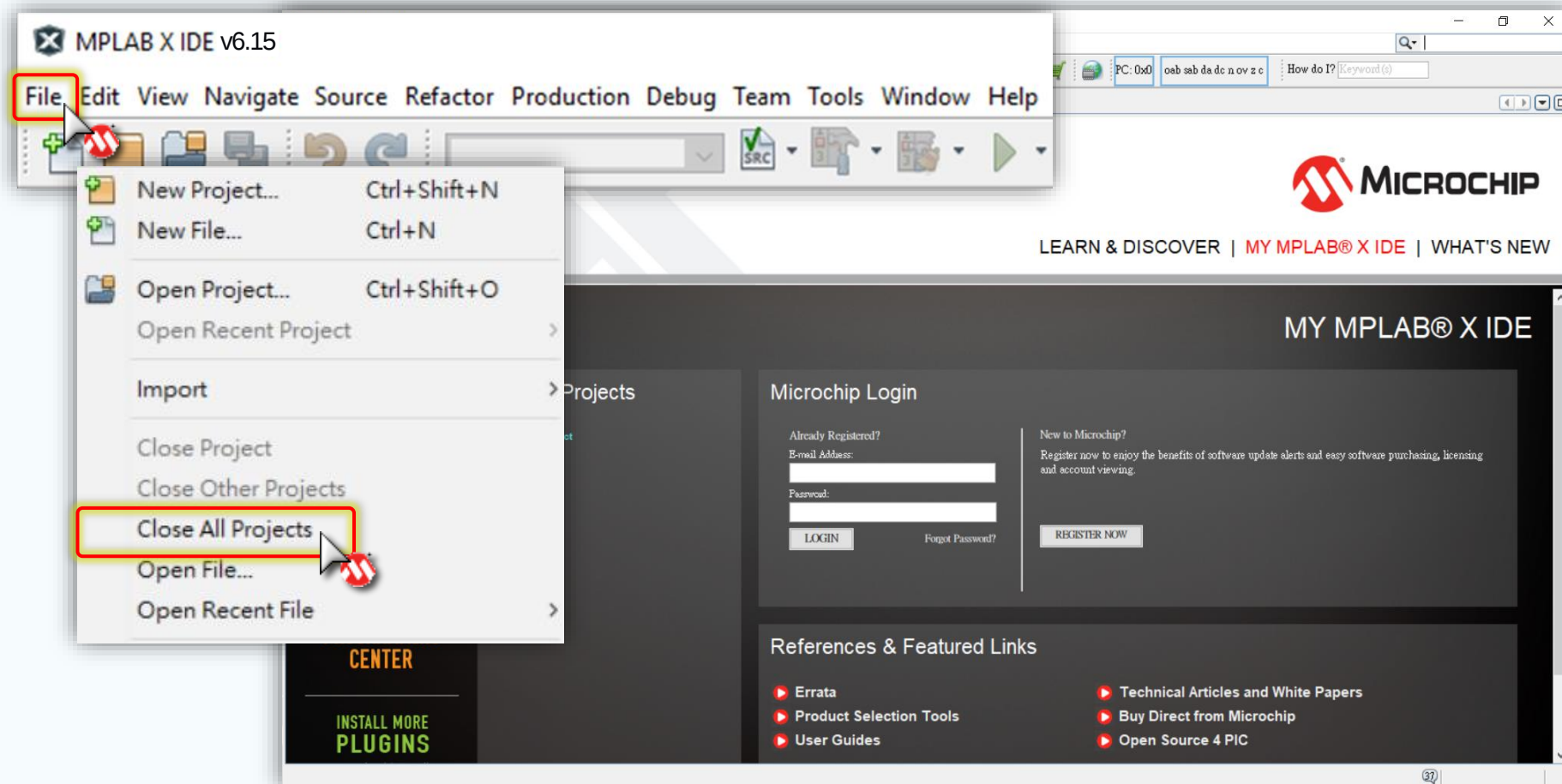
Check Packs Version

- Select : **Tool Packs**
 - Search : "**SNAP**"



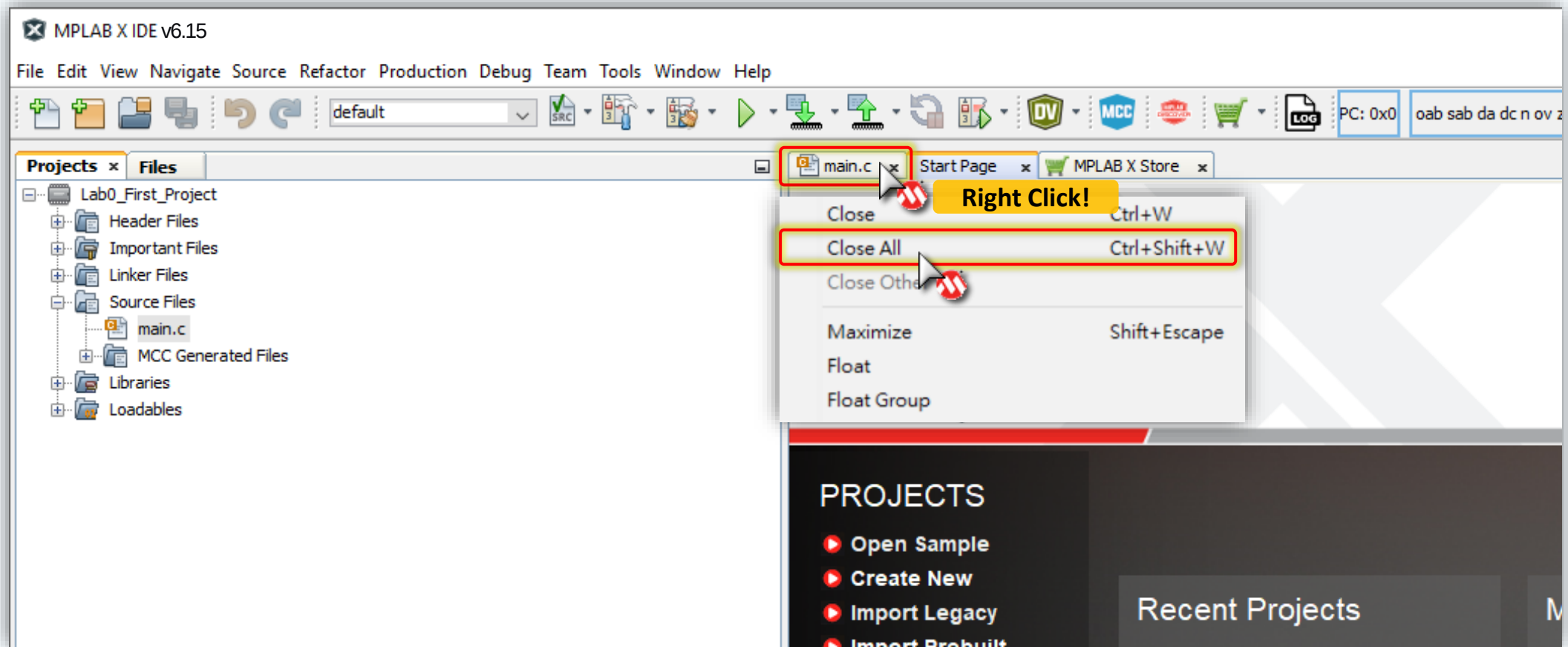
Lab Preparation

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



Lab Preparation

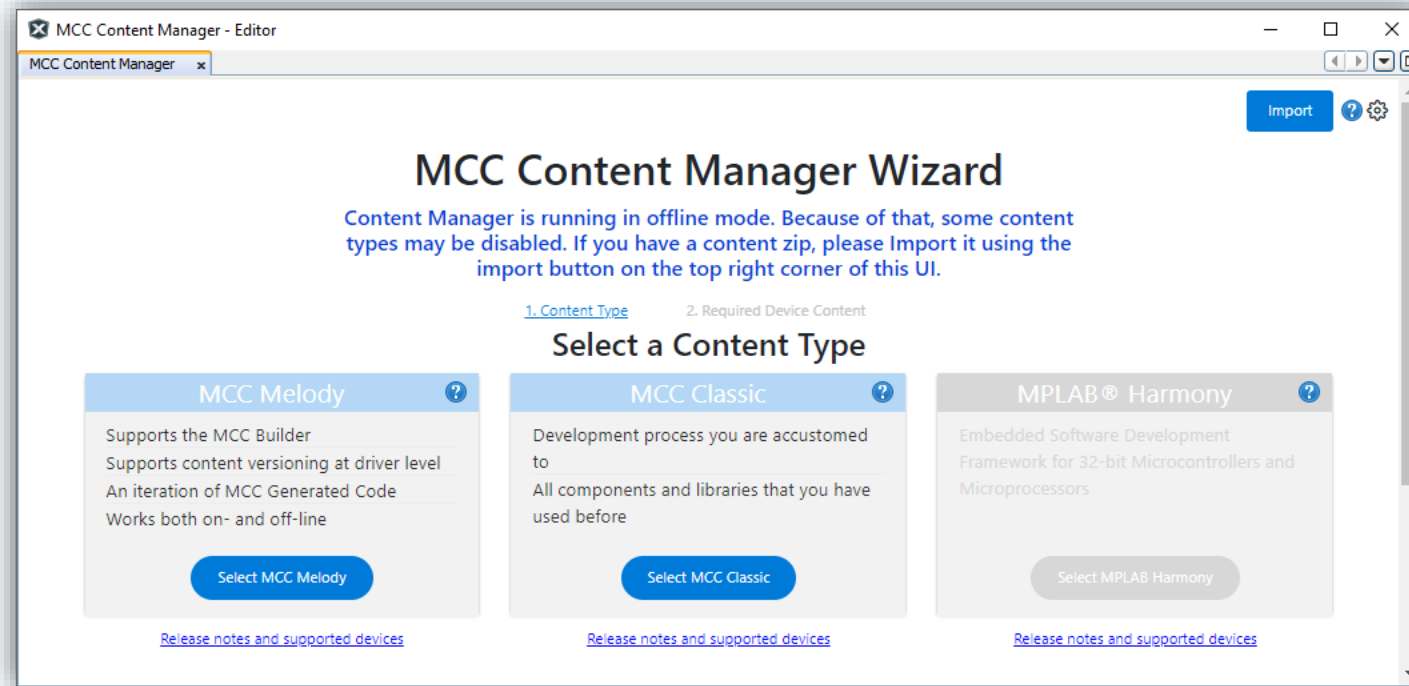
- To prevent opened files to confuse our implementation.
- Please "**Right Click**" on opened file and choose "**Close All**"



Lab Preparation

MCC Offline Mode

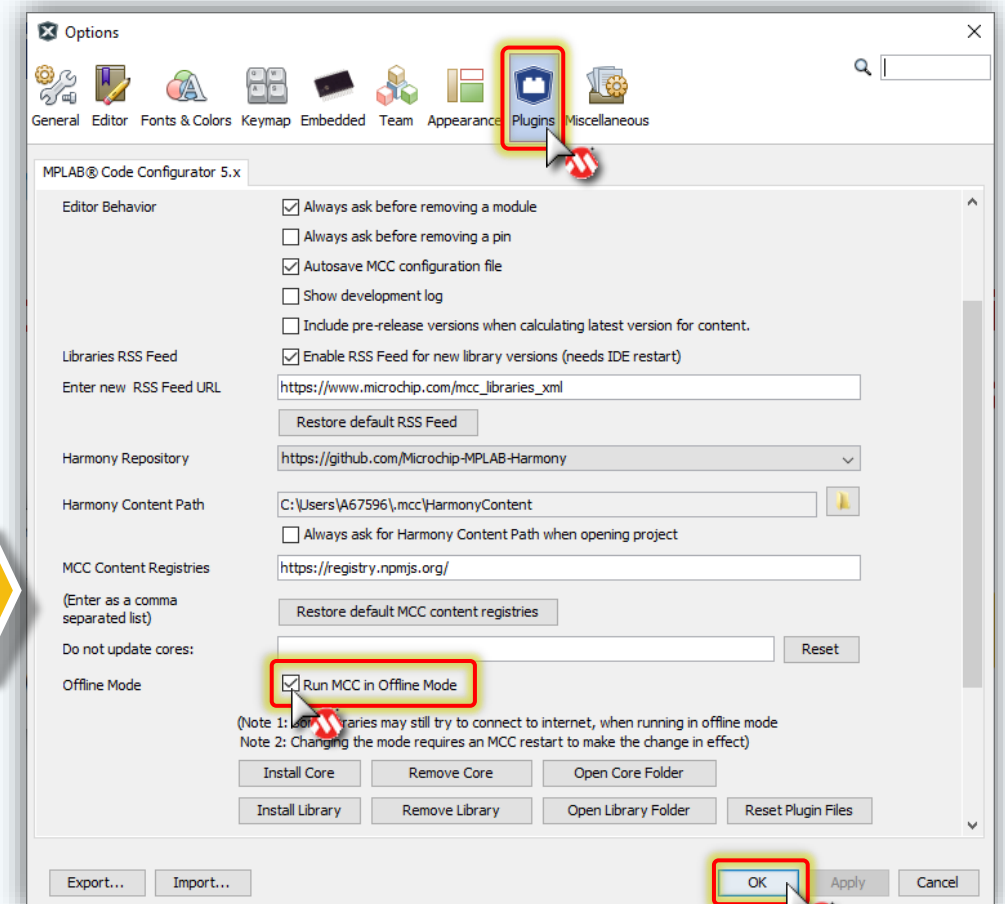
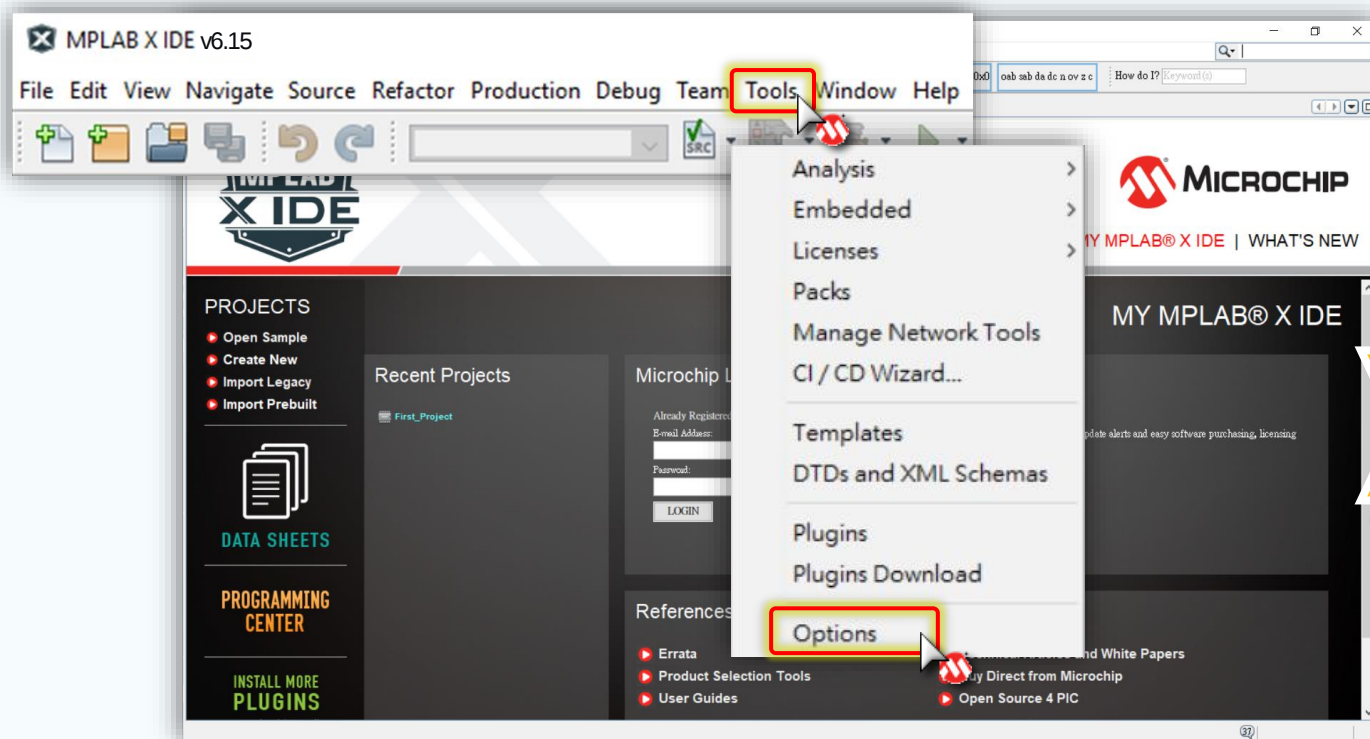
- Generally, using the MCC requires operating in a network environment. If the network environment is not good, there may be problems opening MCC.
- MCC provides a mode that can run without network, named "**Offline Mode**".
- In offline mode, the online content update function will be disabled, user can add or update MCC content through "**import**".



Lab Preparation

MCC Offline Mode (optional feature)

- Select : **Tools > Options**
- Enable the **Plugins > Run MCC in Offline Mode**, then click "**Finish**"



Lab0 First Project

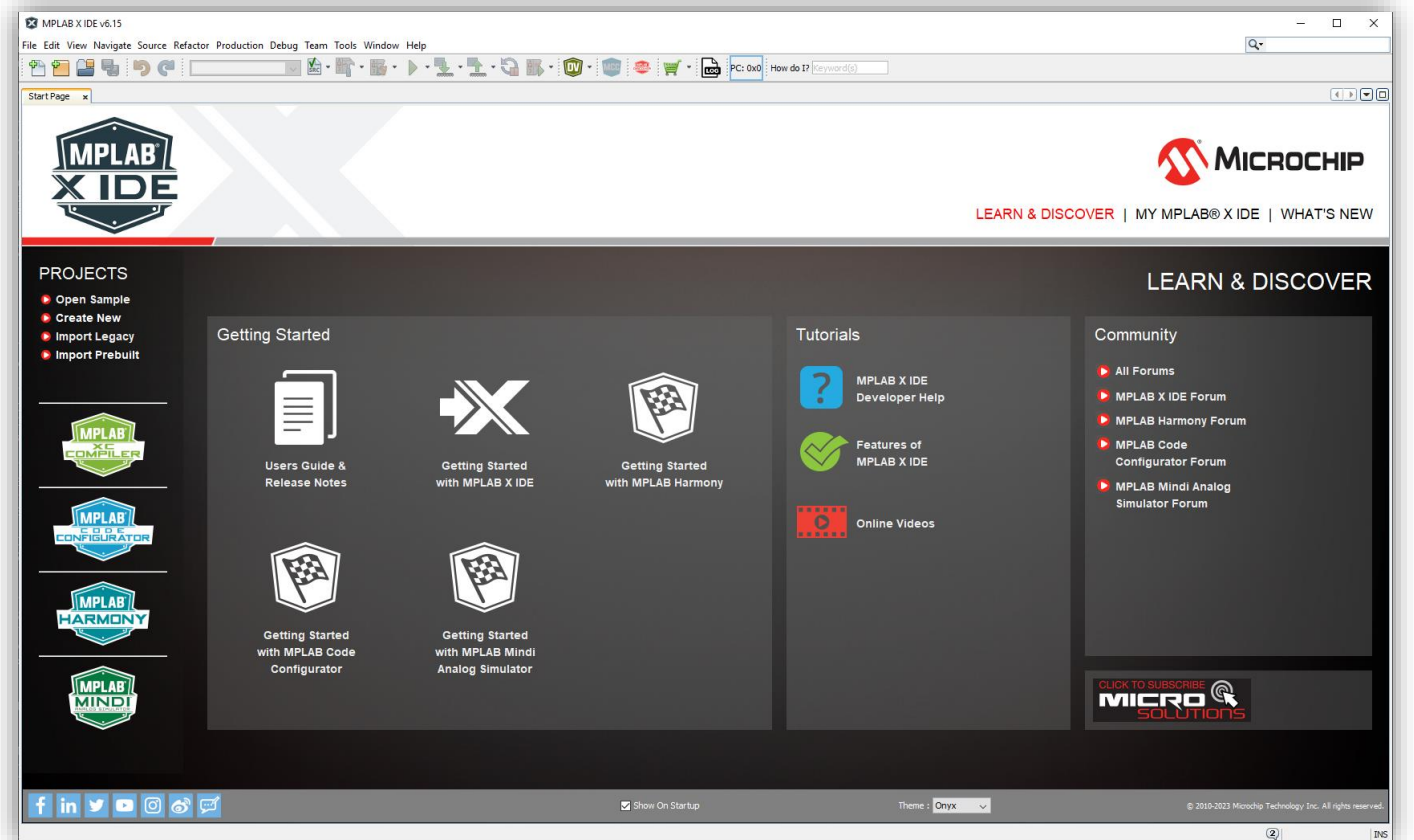
- Try to create your first **MPLAB X IDE** Project.
- Try to create your first **MCC Melody** style Project.
- To understanding MPLAB X IDE basic operation.
- Learn how to use edit, build, project manager and program functions.

How to start ?

Lab0 First Project

Step 1

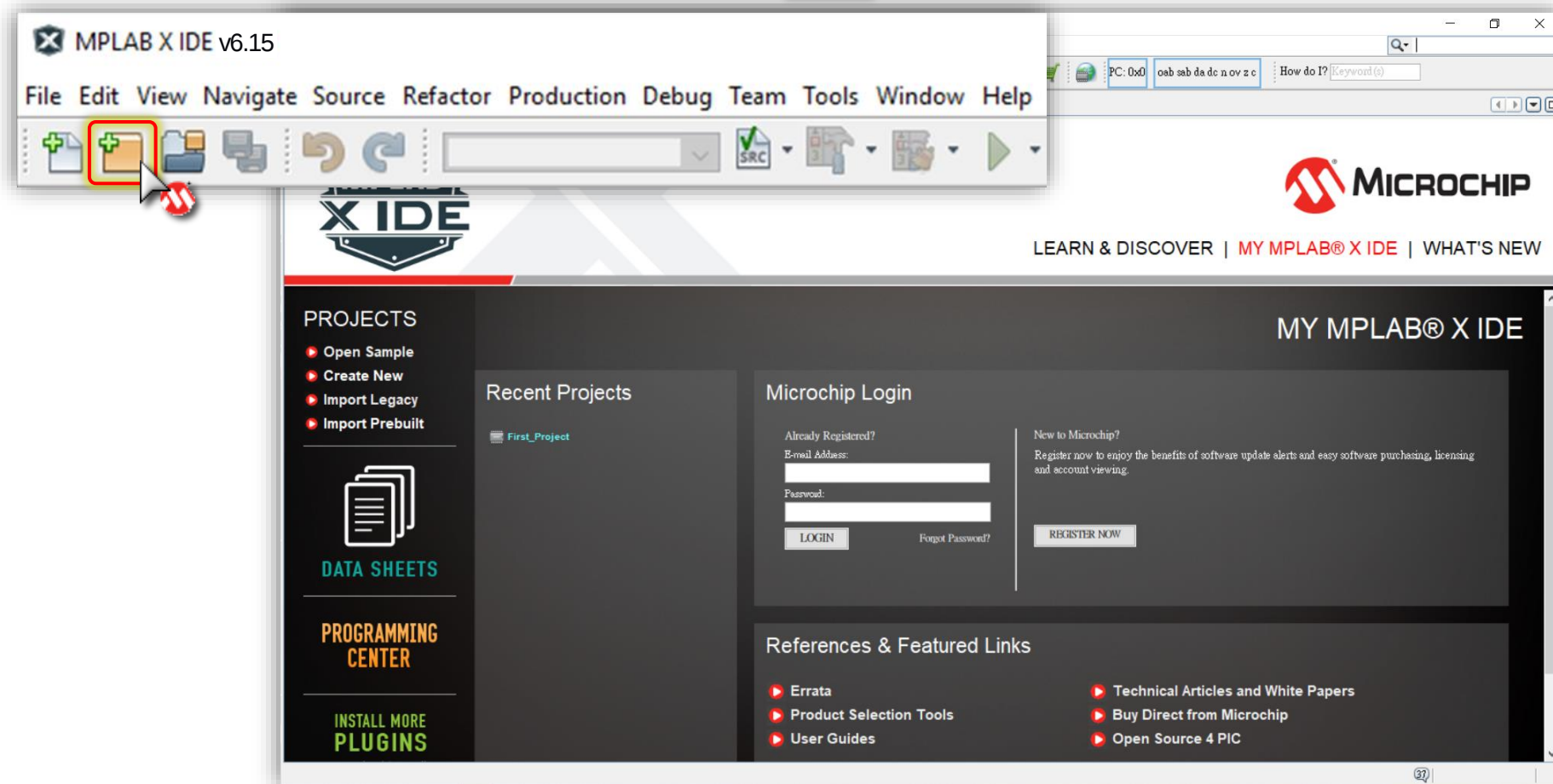
- Open **MPLAB X IDE v6.15** firstly.



Lab0 First Project

Step 2

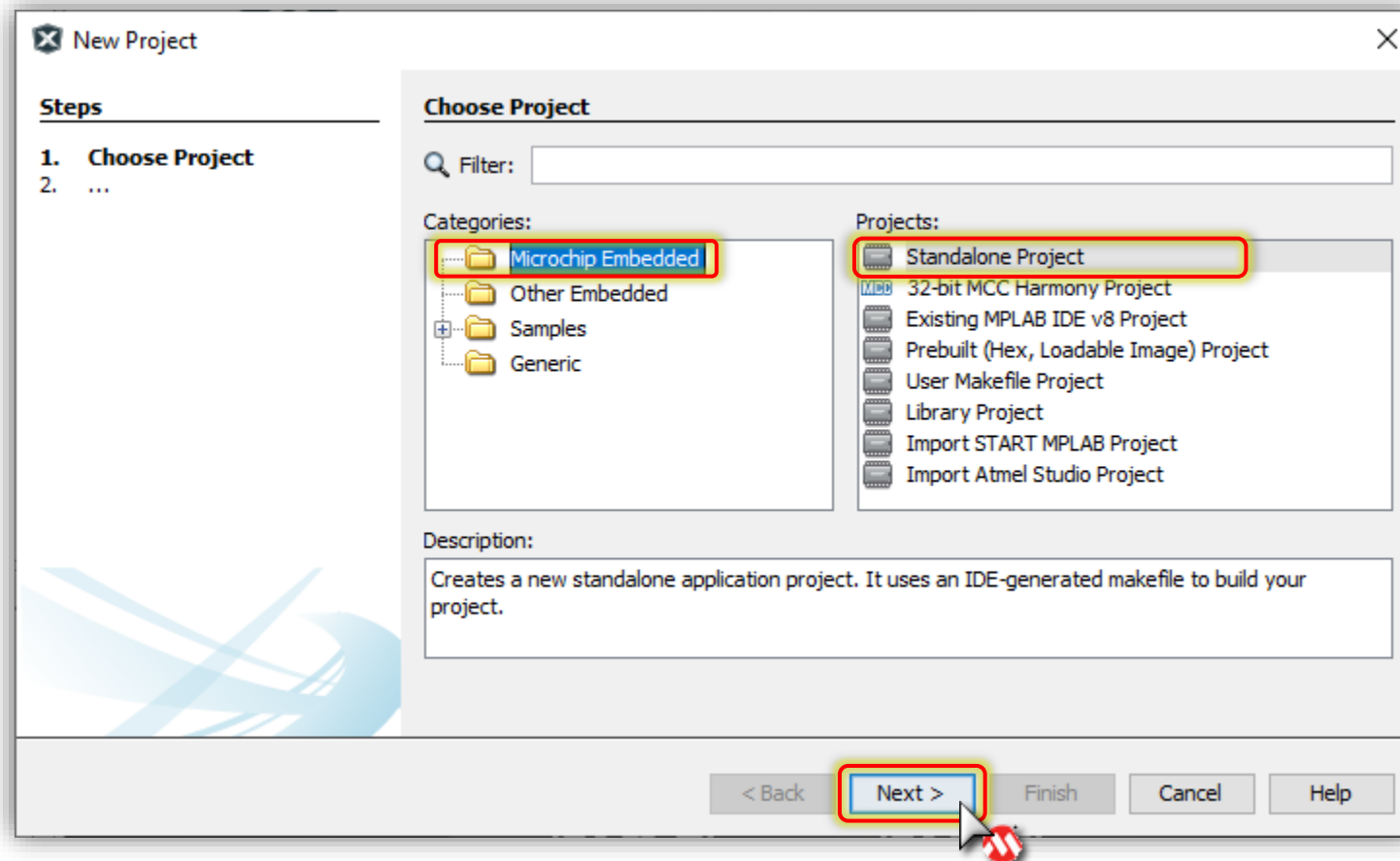
- Execute Project Wizard
 - Select : **File > New Project** or Click icon



Lab0 First Project

Step 3

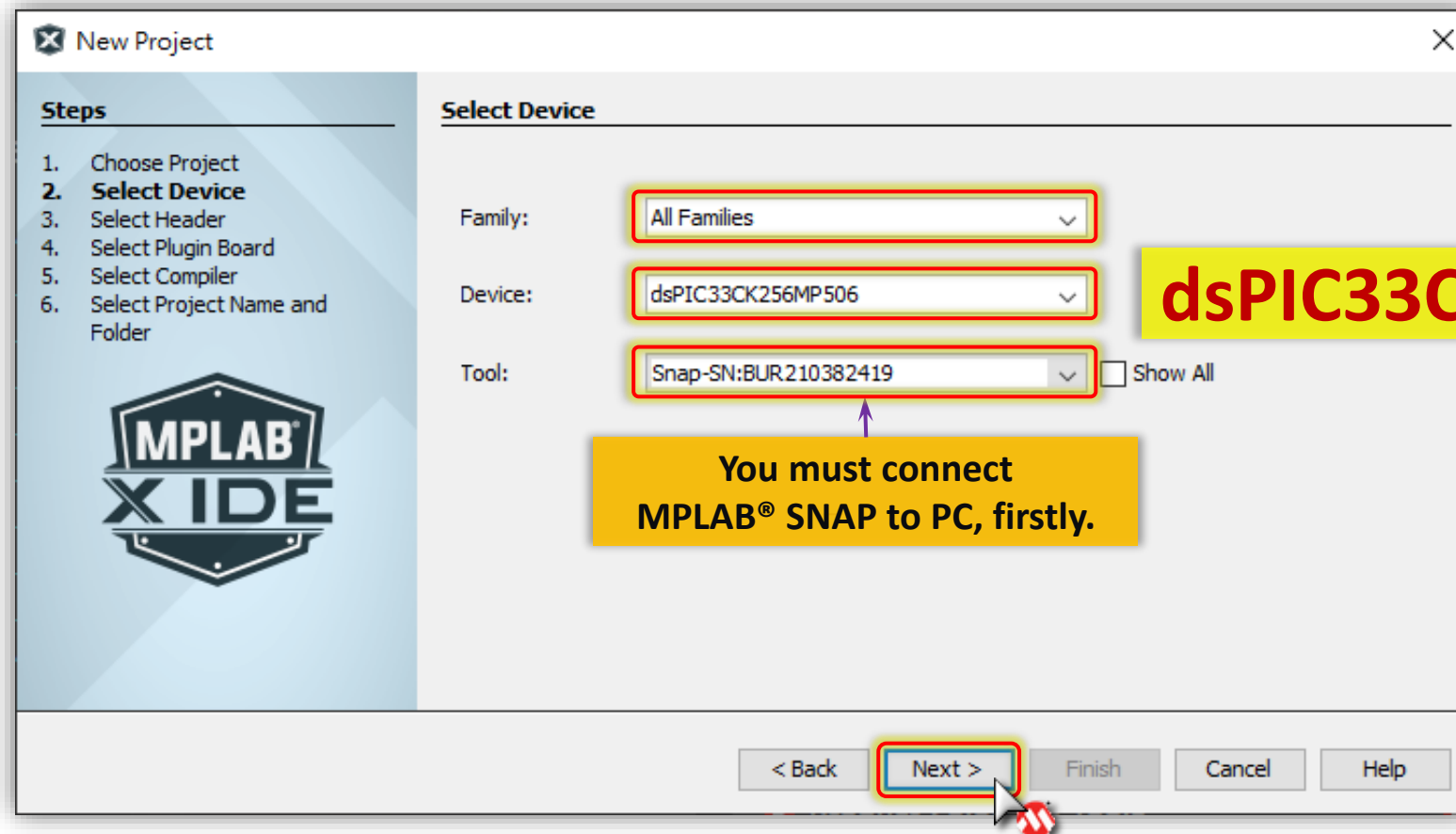
- Chooses Project :
 - Categories: **Microchip Embedded**, Projects: **Standalone Project**



Lab0 First Project

Step 4

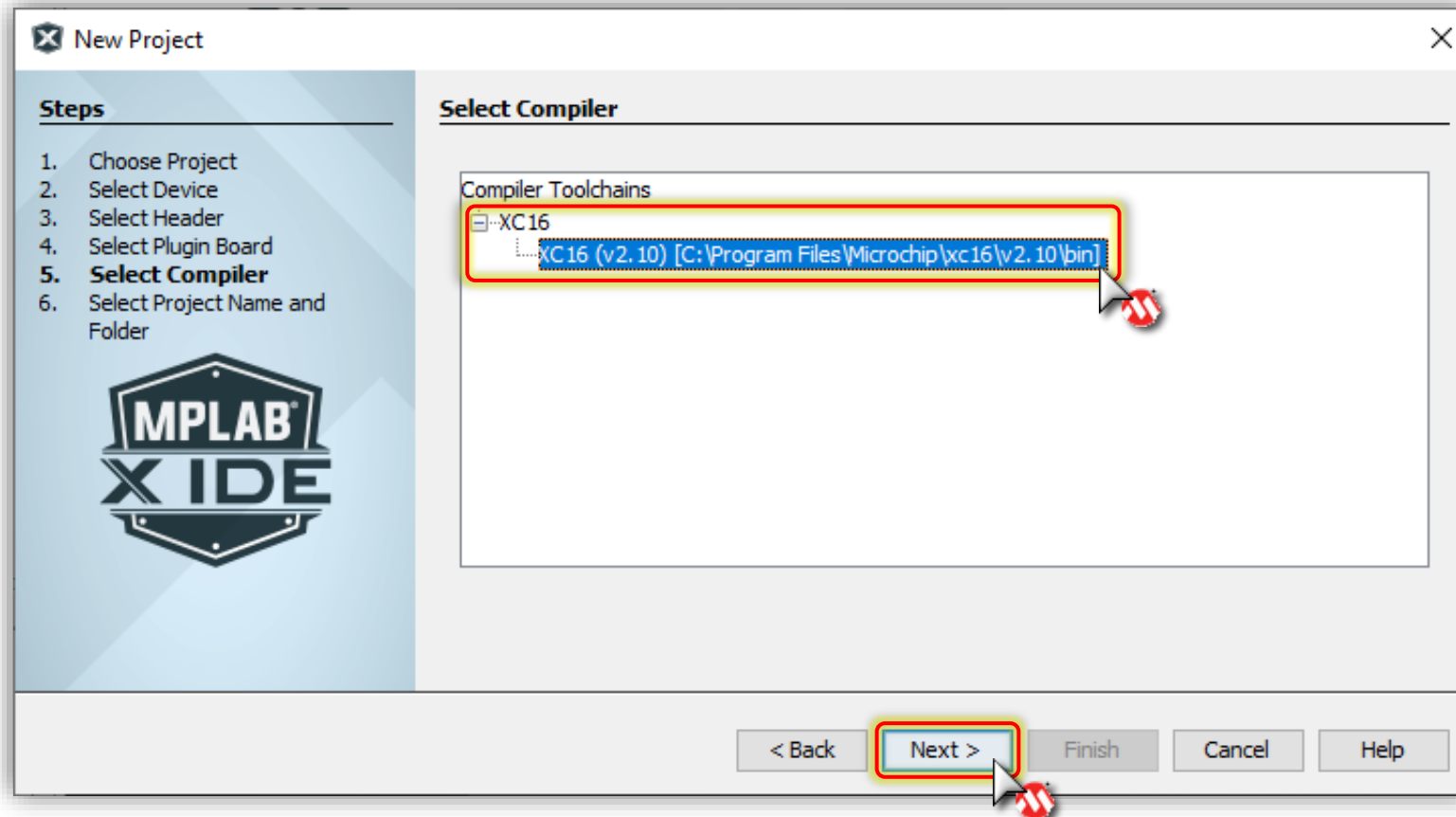
- Select Target Device :
 - Family: **16-bit DSCs (dsPIC33)**, Device: **dsPIC33CK256MP506**, Tool: **Snap-SN : BURXXXXXXXXXX**



Lab0 First Project

Step 5

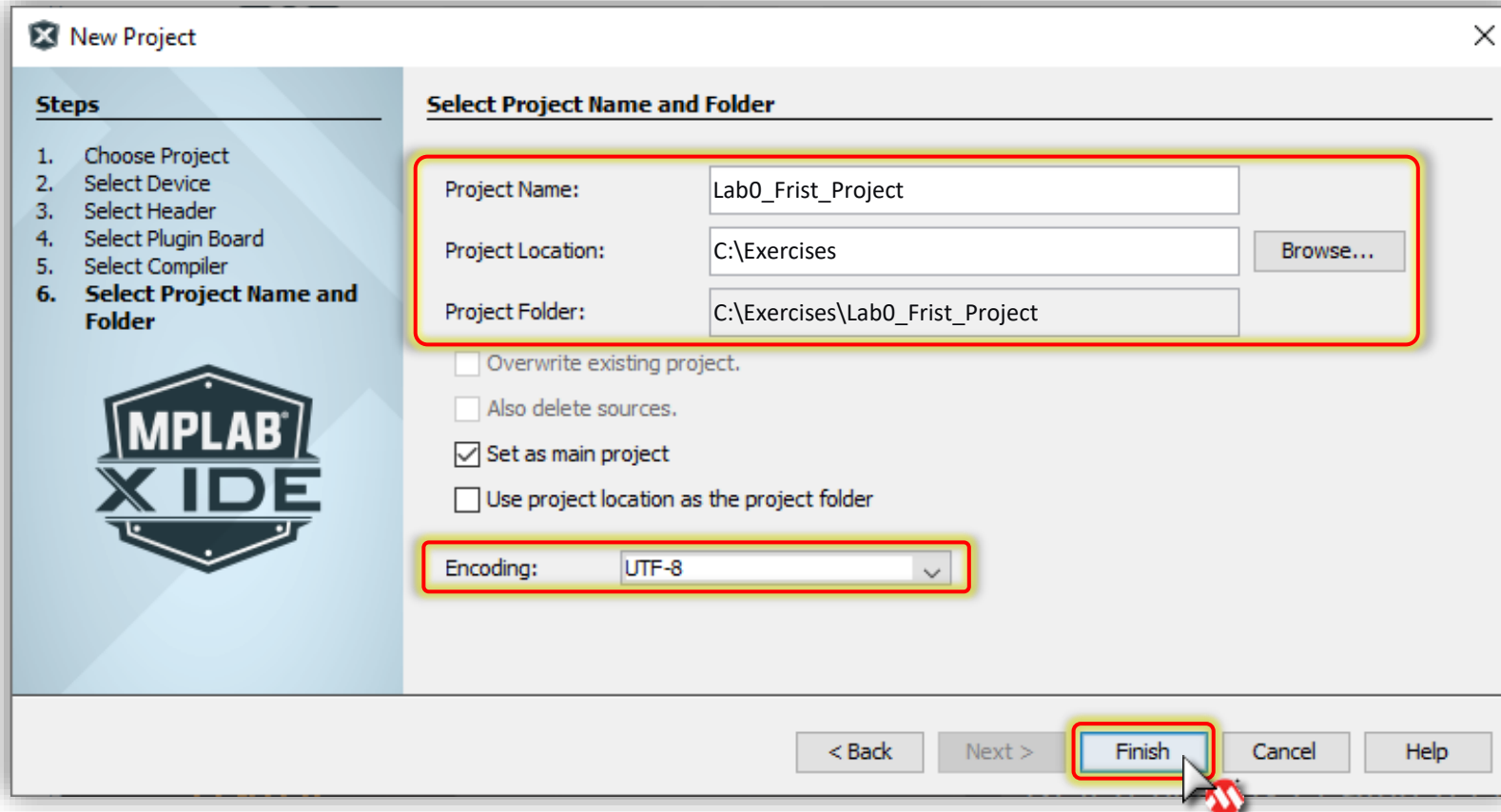
- **Select Compiler :**
 - **Compiler Toolchains:** **XC16 (v2.10) [C:\Program Files\Microchip\xc16\v2.10\bin]**



Lab0 First Project

Step 6

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



The image shows the 'New Project' dialog box in the MPLAB X IDE. The 'Steps' panel on the left lists six steps, with step 6, 'Select Project Name and Folder', highlighted. The main area of the dialog is titled 'Select Project Name and Folder'. It contains three text input fields: 'Project Name' with the value 'Lab0_Frist_Project', 'Project Location' with the value 'C:\Exercises', and 'Project Folder' with the value 'C:\Exercises\Lab0_Frist_Project'. A 'Browse...' button is next to the 'Project Location' field. Below these fields are four checkboxes: 'Overwrite existing project.' (unchecked), 'Also delete sources.' (unchecked), 'Set as main project' (checked), and 'Use project location as the project folder' (unchecked). At the bottom, there is an 'Encoding' dropdown menu set to 'UTF-8'. The bottom of the dialog has five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Finish' button is highlighted with a red box and a mouse cursor is pointing at it.

New Project

Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Plugin Board
5. Select Compiler
6. **Select Project Name and Folder**

MPLAB X IDE

Select Project Name and Folder

Project Name: Lab0_Frist_Project

Project Location: C:\Exercises Browse...

Project Folder: C:\Exercises\Lab0_Frist_Project

☐ Overwrite existing project.

☐ Also delete sources.

☒ Set as main project

☐ Use project location as the project folder

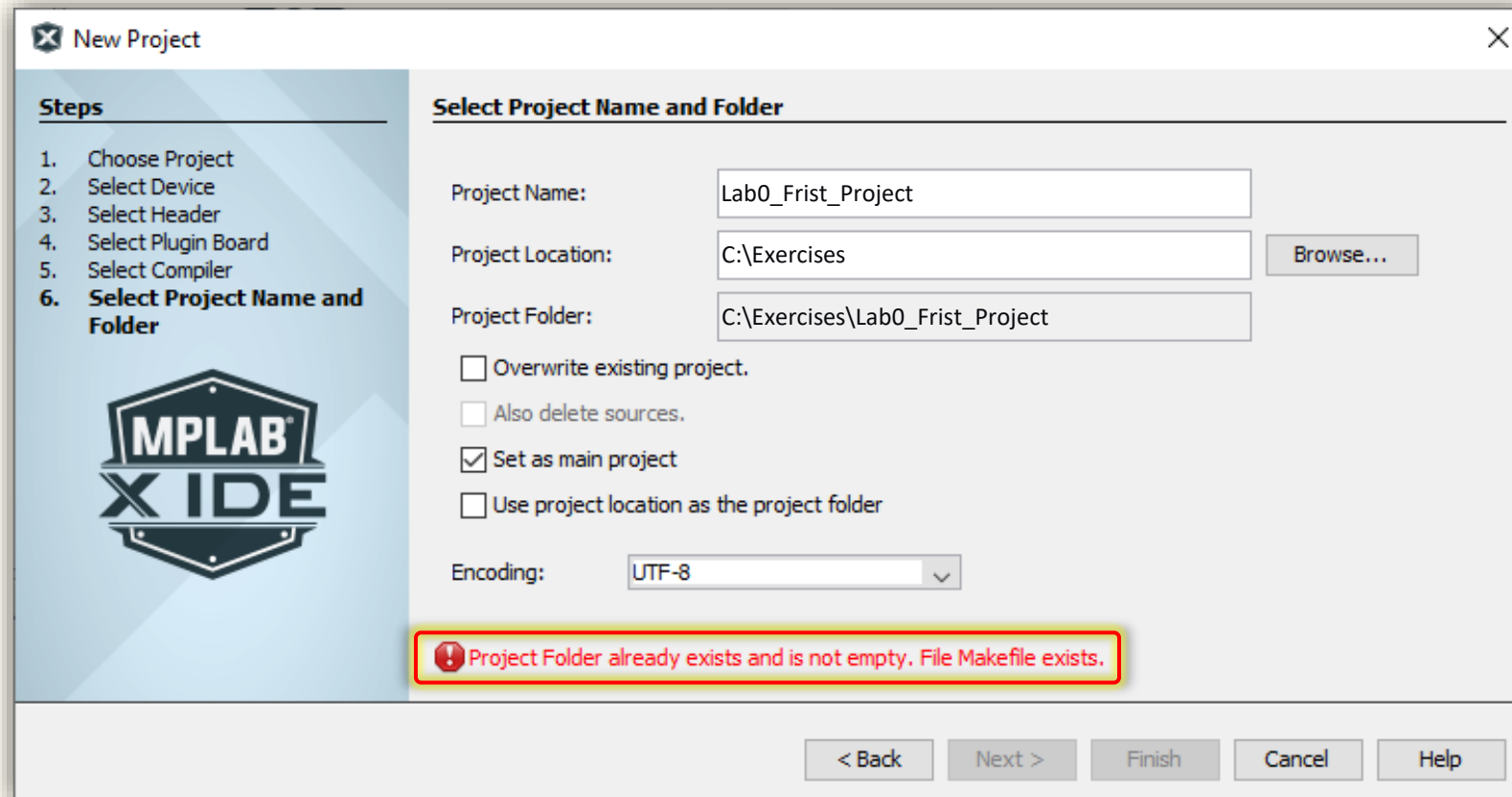
Encoding: UTF-8

< Back Next > Finish Cancel Help

Lab0 First Project

Notice

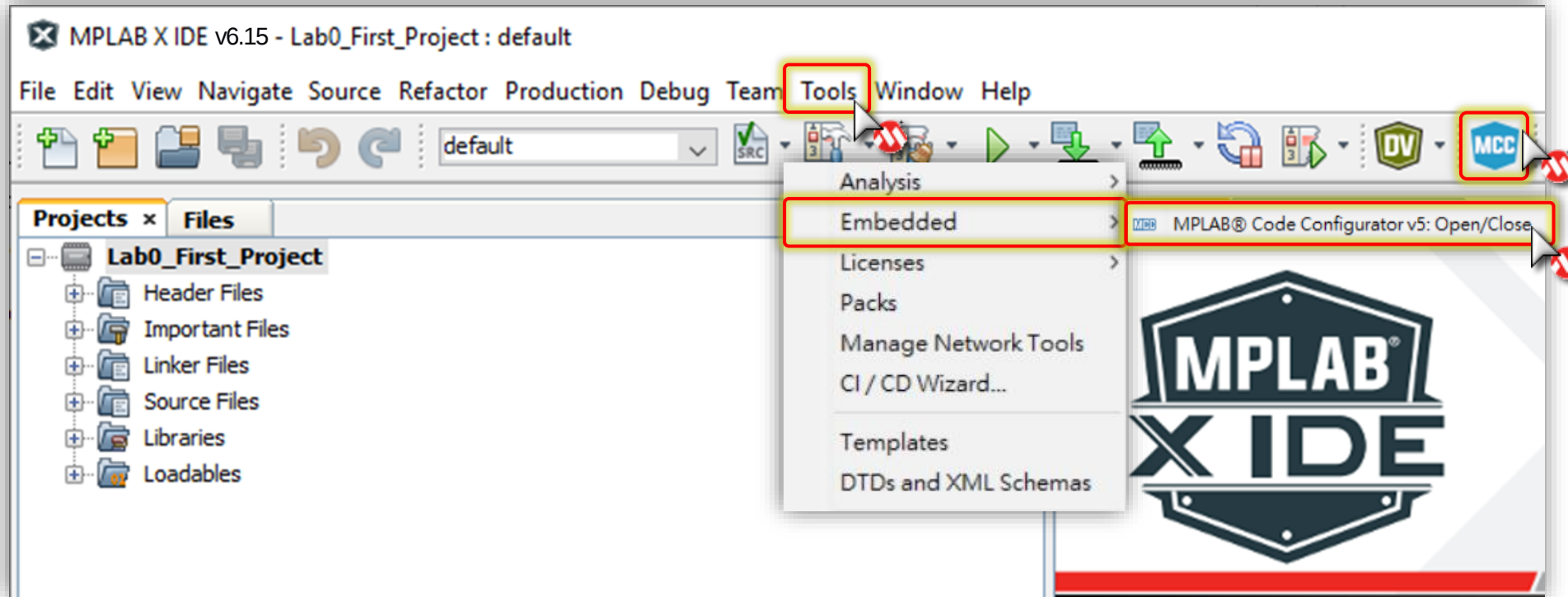
- If an old project already exist at the same location, you must delete it firstly or change to new project folder.



Lab0 First Project

Step 7

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



or

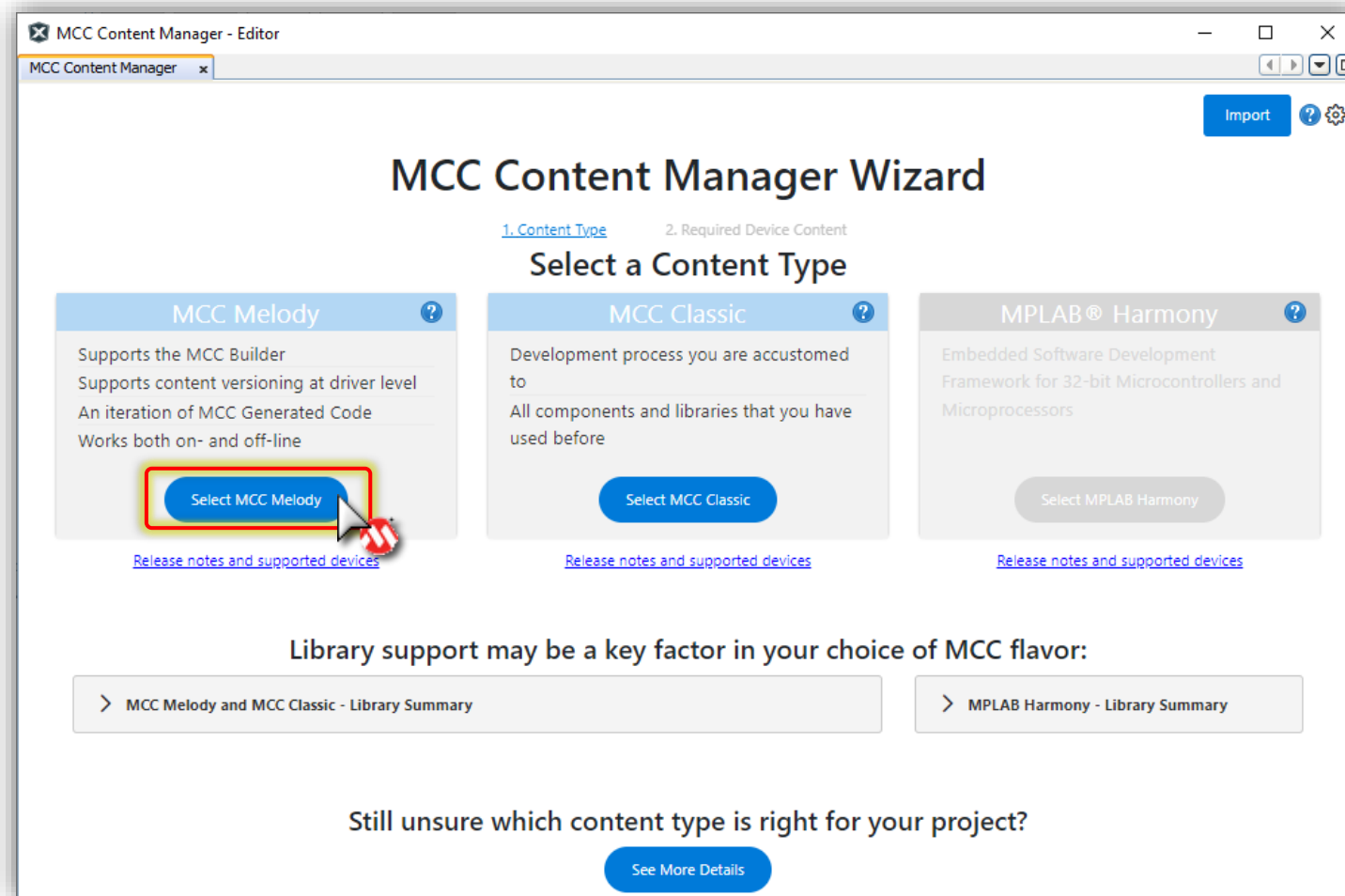
Online Download MCC Framework (Internet Accessible)

MCC Framework Preparation

Lab0 First Project

Step 8 (Online Download MCC Framework)

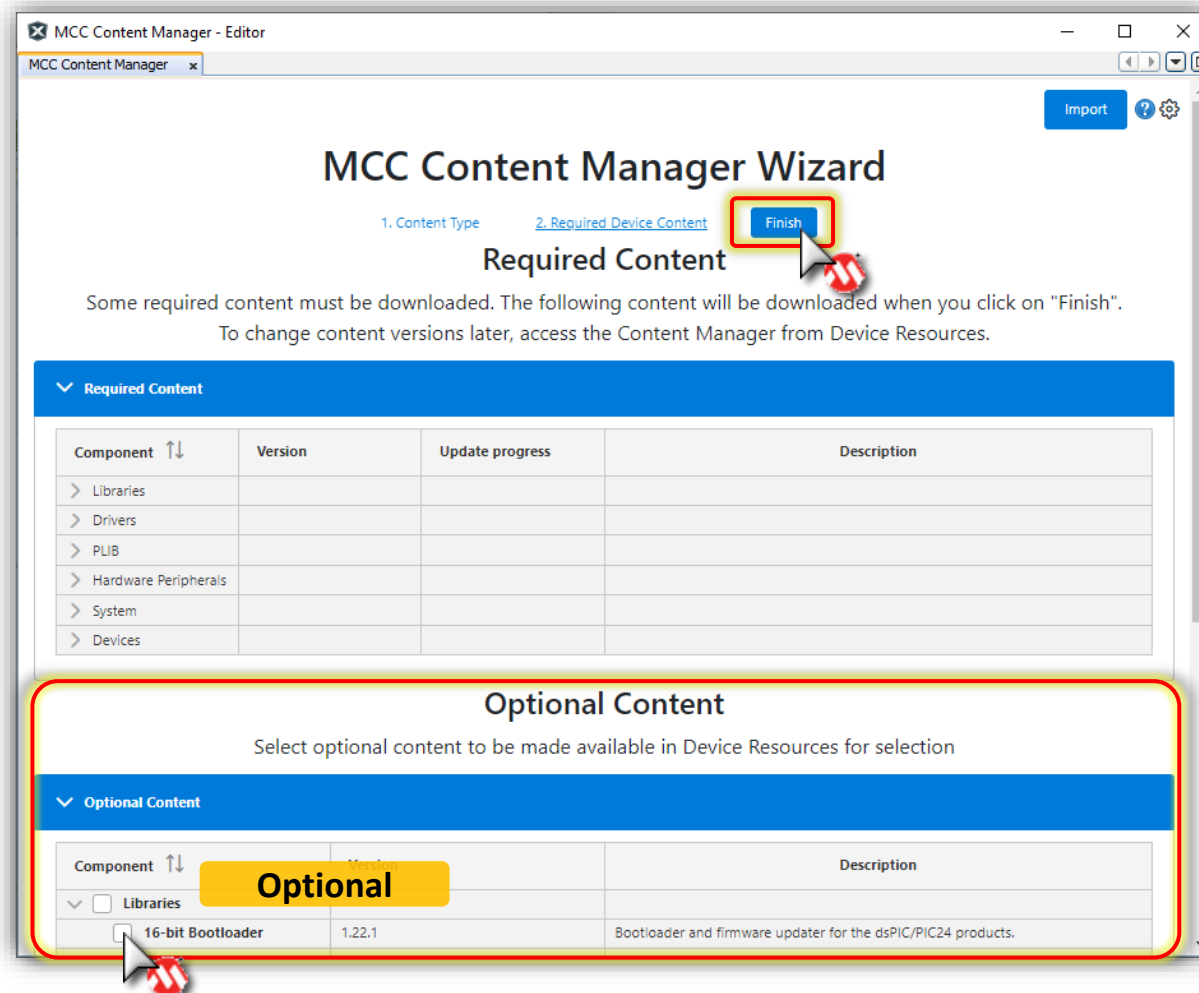
- Click the **"Select MCC Melody"**



Lab0 First Project

Step 9 (Online Download MCC Framework)

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



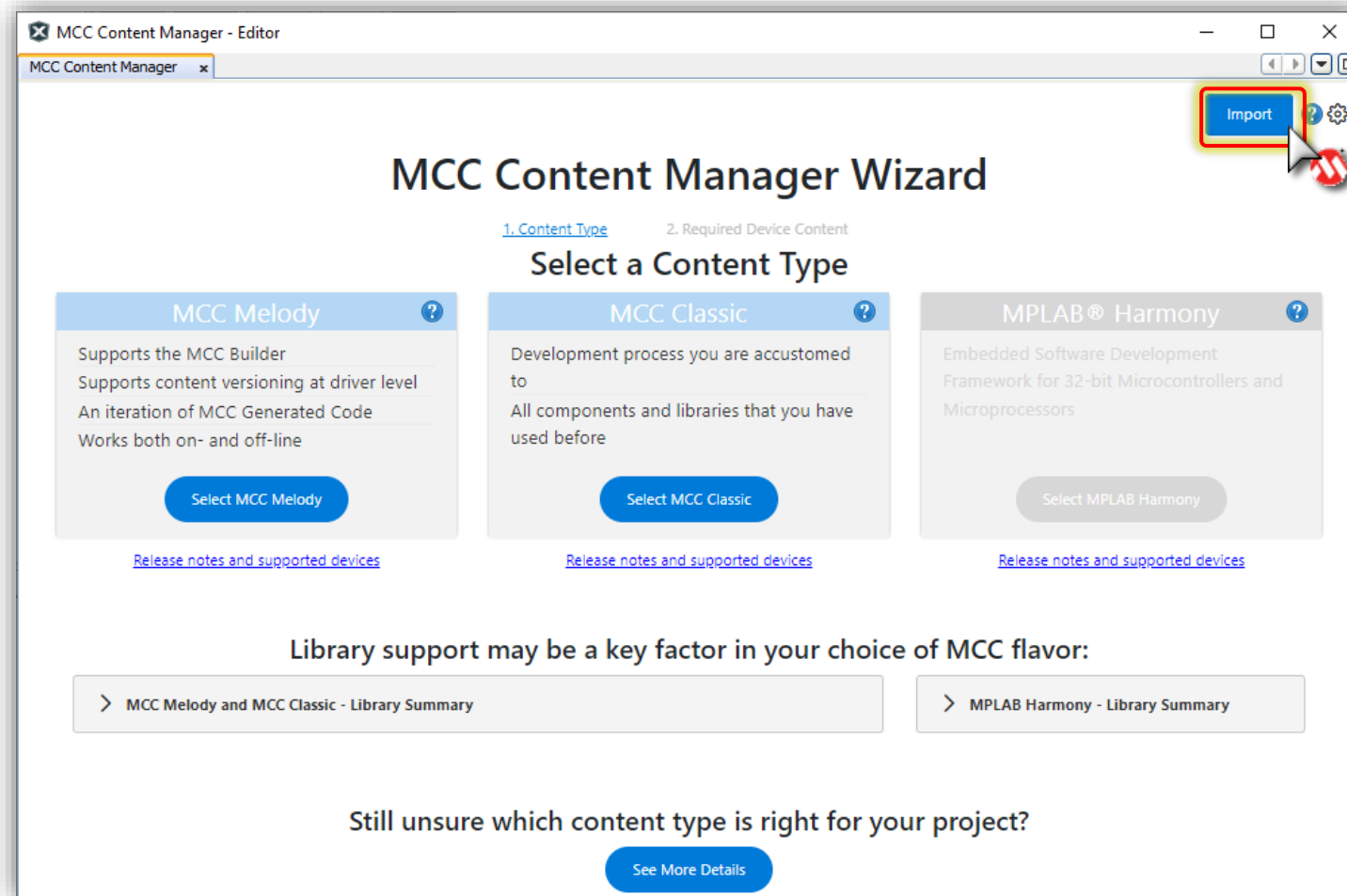
MCC Off-Line Mode Import Exist MCC Framework (Internet not Stable)

MCC Framework Preparation

Lab0 First Project

Step 8 (Import Exist MCC Framework)

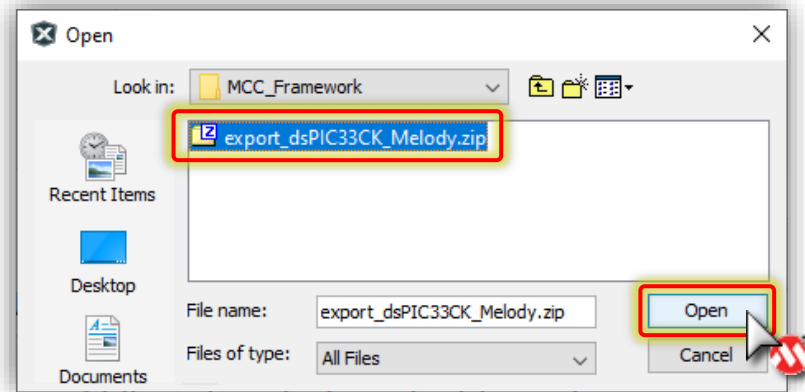
- Click "**Import**".



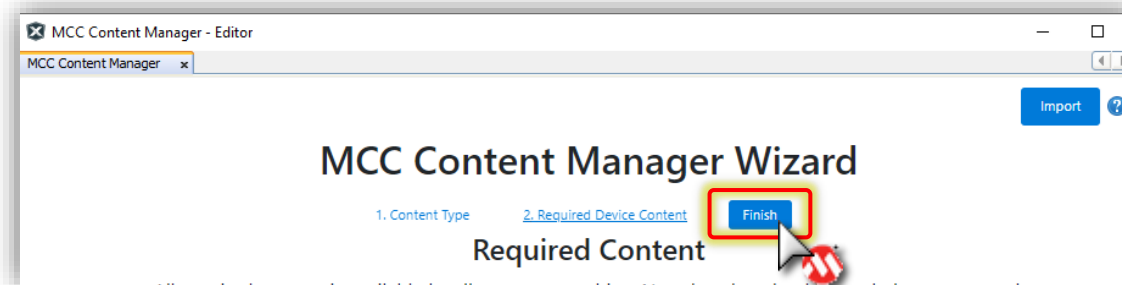
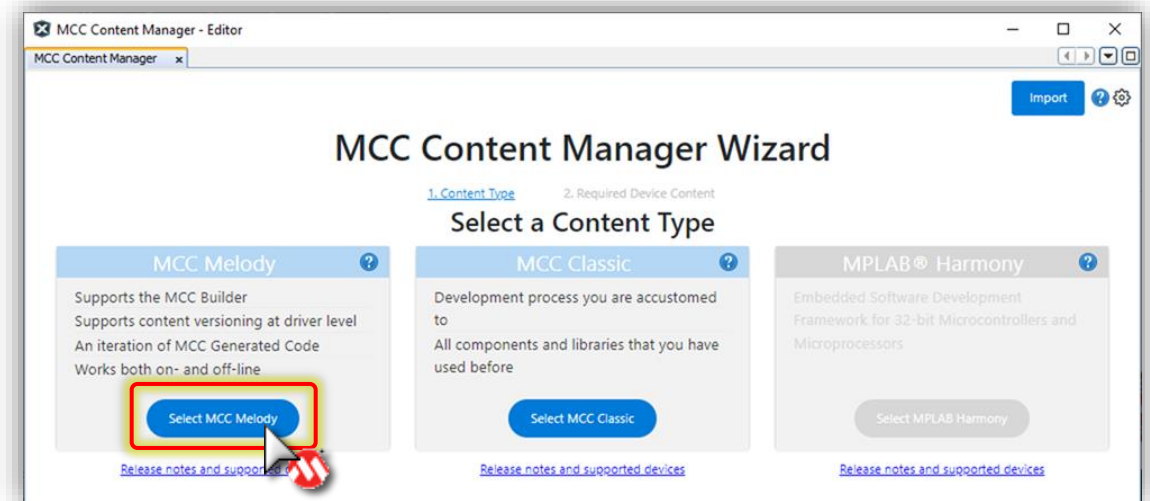
Lab0 First Project

Step 9 (Import Exist MCC Framework)

- Select the exist "**Framework Archive (*.zip)**" and click on **Open**.
- Click the "**Select MCC Melody**", then click "**Finish**"



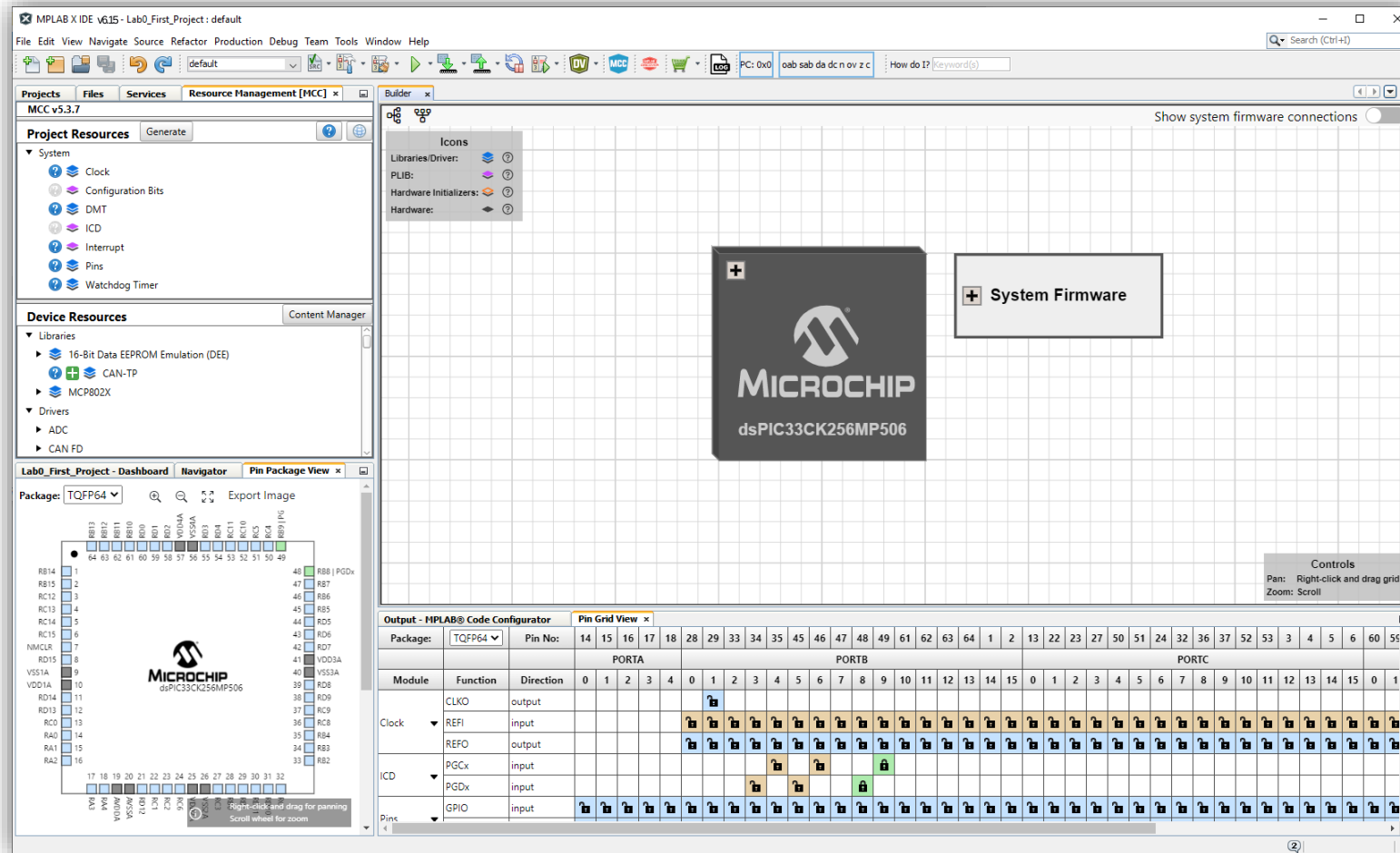
Wait Download
Finished



Lab0 First Project

Step 10

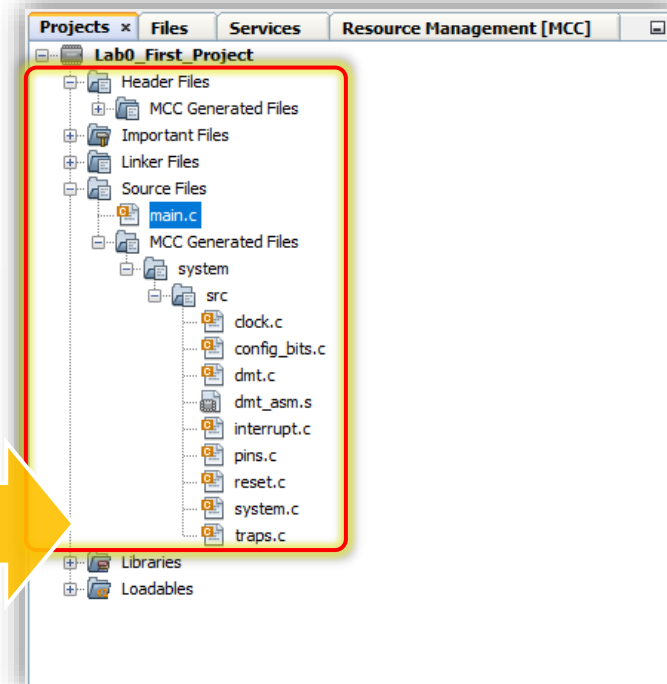
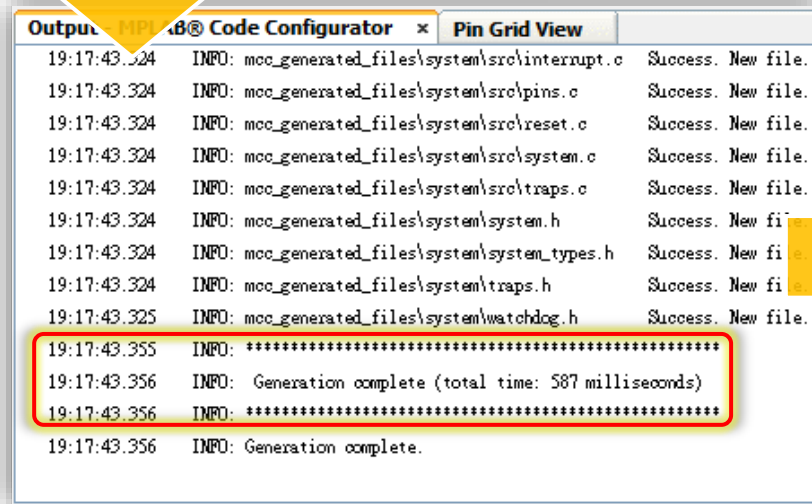
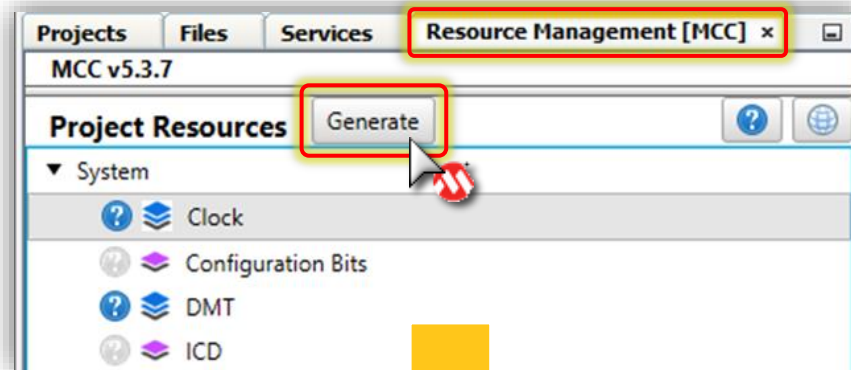
- MCC Configurator interface show up.



Lab0 First Project

Step 11

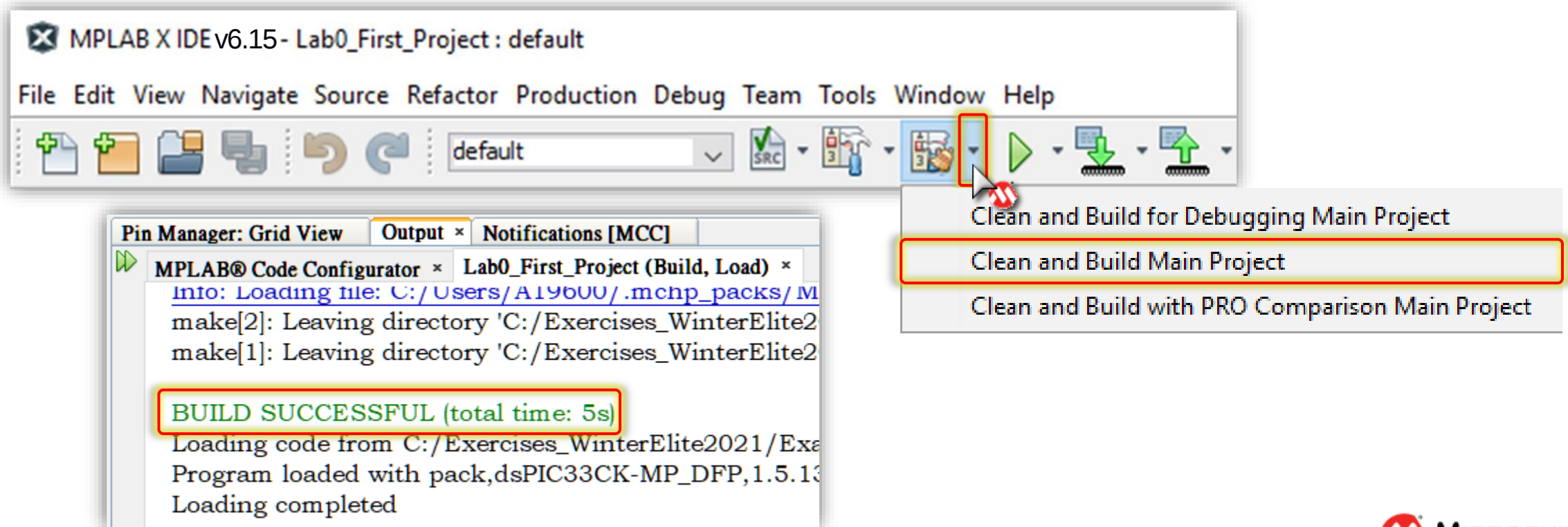
- Click "**Generate**" Button to get your first MCC Project.
- MCC to generate files include "**main.c**" automatically.



Lab0 First Project

Step 12

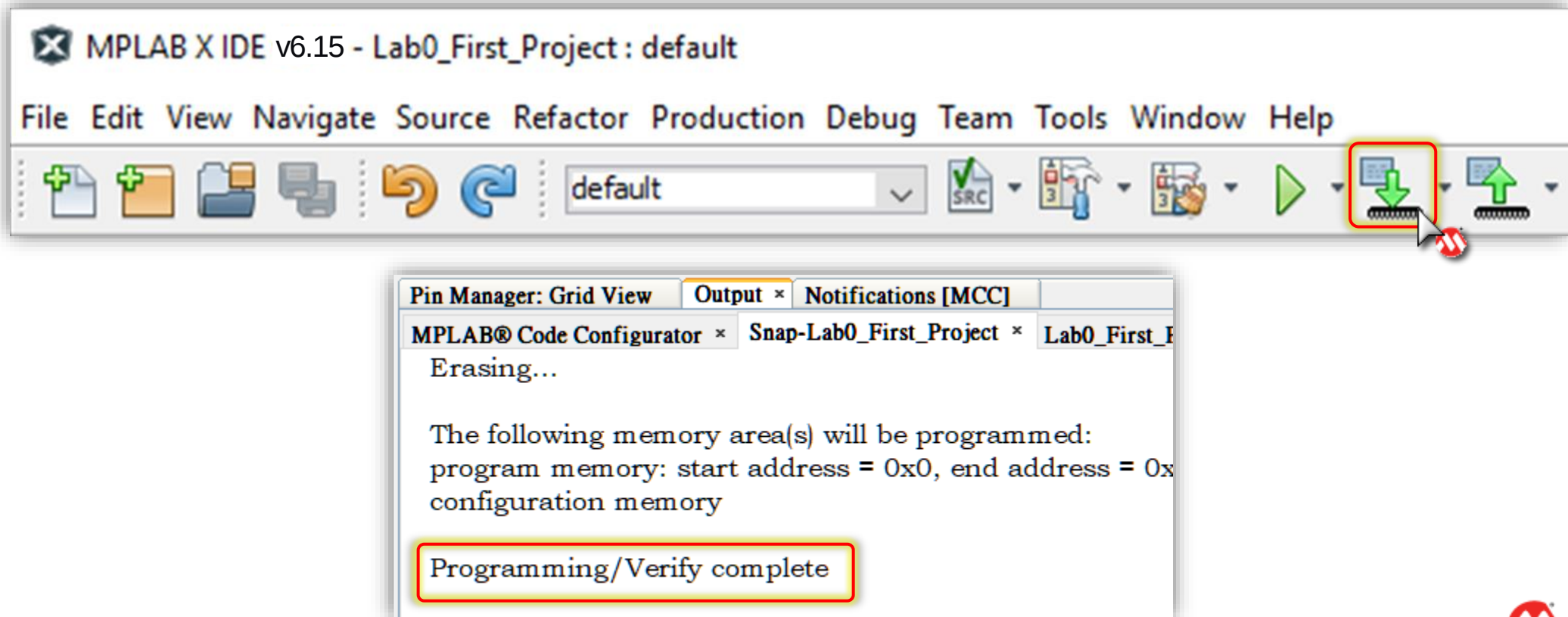
- Build your first MCC Project.
 - Click the "Arrow Button" near the icon
 - Select "**Clean and Build Main Project**"
 - Make sure compiled result is **BUILD SUCCESSFUL**



Lab0 First Project

Step 13

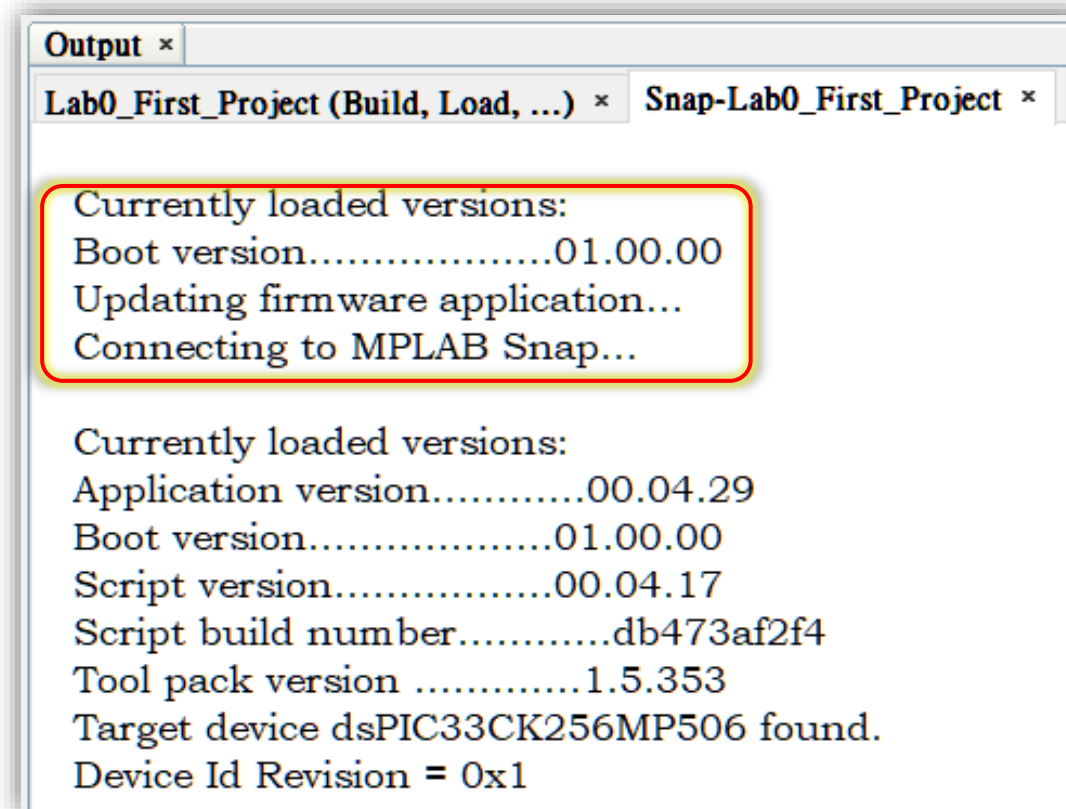
- Try program compiled firmware to your target board.
 - Select "**Make and Program Device Main Project**" icon 
 - Make sure the result is Programming/Verify complete



Lab0 First Project

Notice

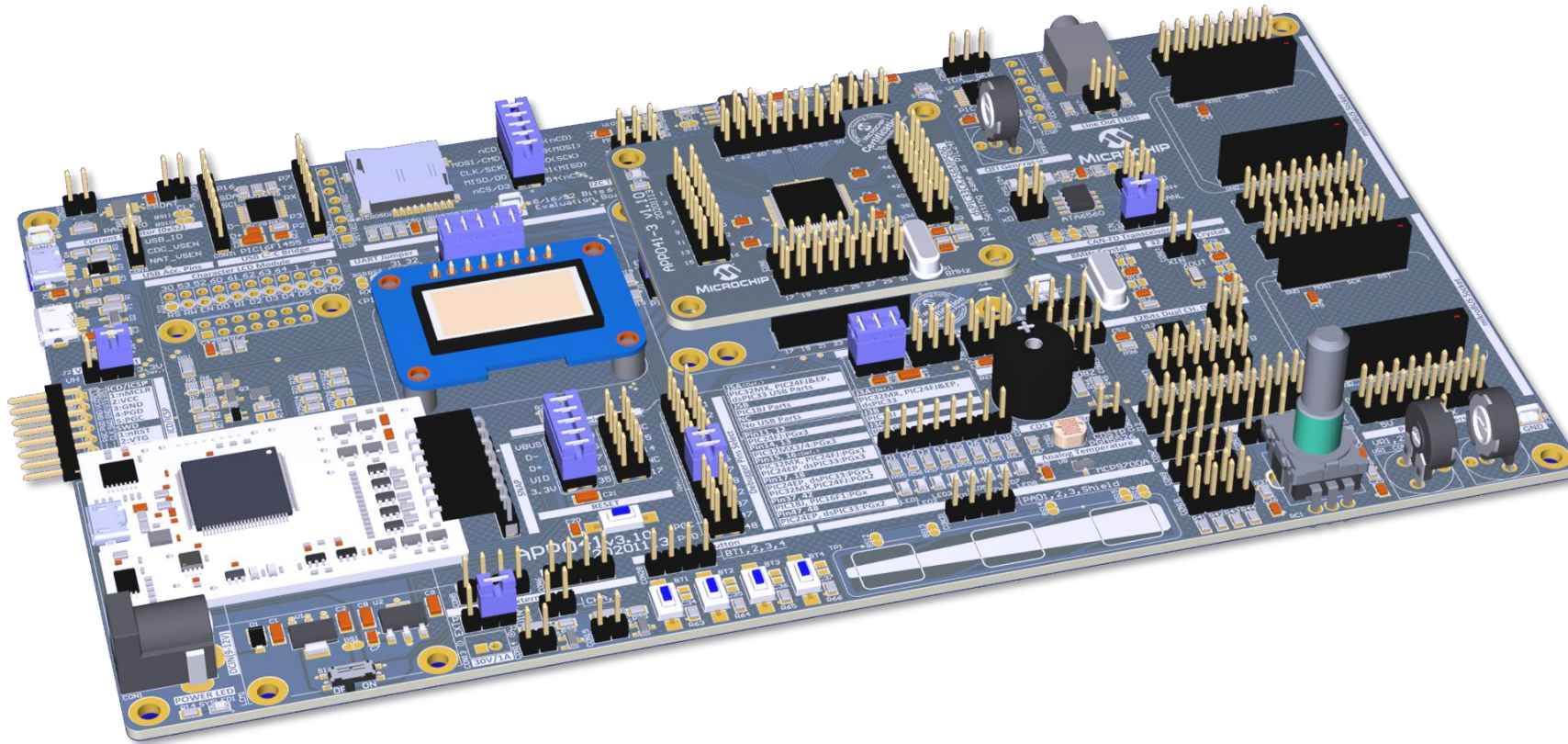
- MPLAB X IDE will auto upgrade the firmware of SNAP to keep it up to date, it might take couple minutes to wait for finish. Your project will start programming after this.



Lab0 First Project

Check Point

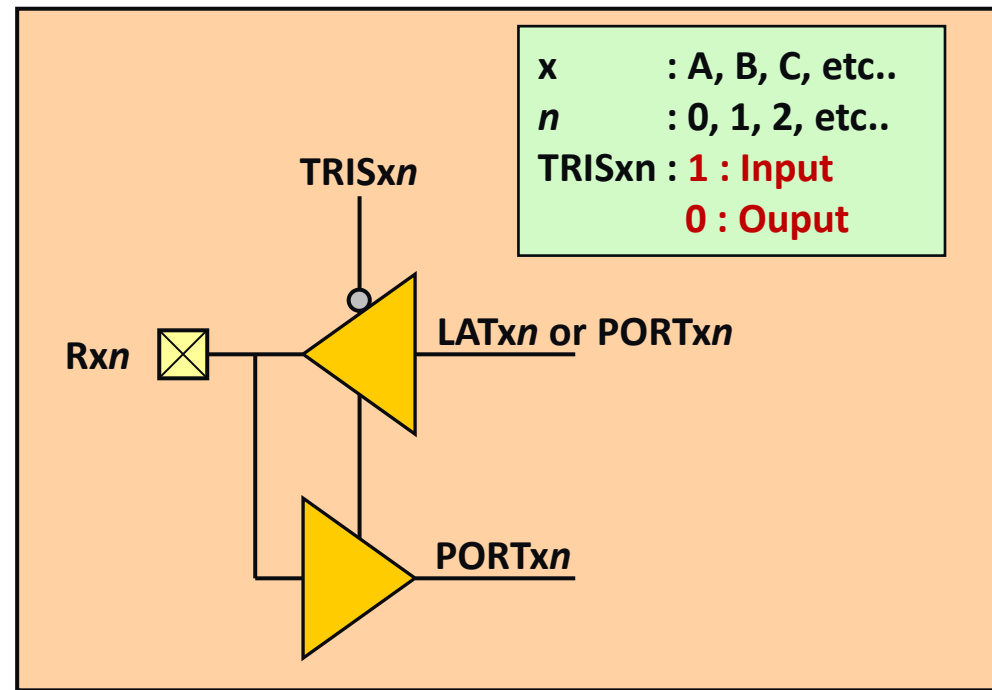
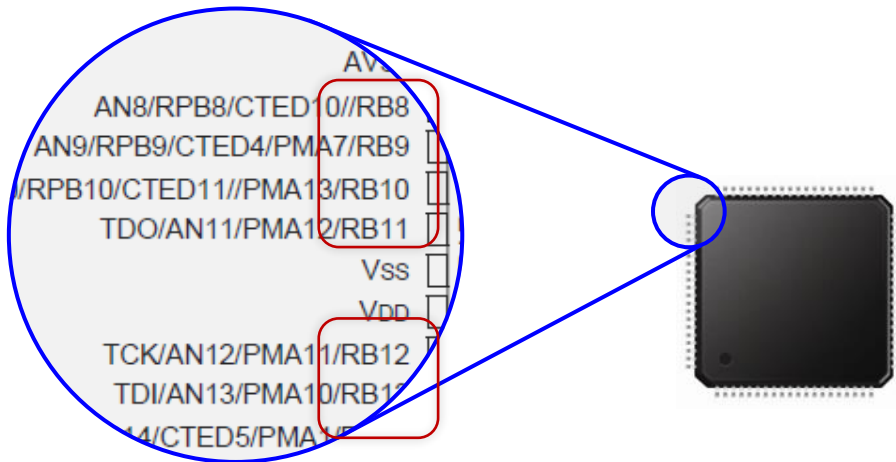
Nothing happen ????



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.

Pin Grid View

Package: TQFP64

Pin No: 14 15 16 17 18 28 29 33 34 35 45 46 47 48 49 61 62 63 64 1 2

Module	Function	Direction	PORTA					PORTB															
			0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Clock	CLKO	output																					
	REFI	input																					
	REFO	output																					
ICD	PGCx	input																					
	PGDx	input																					
Pins	GPIO	input																					
	GPIO	output																					

Resource Management [MCC] x

MCC v5.3.7

Project Resources Generate

System

- Clock
- Configuration Bits
- DMT
- ICD
- Interrupt
- Pins
- Watchdog Timer

Device Resources Content Manager

Drivers

- ADC
- CAN FD
- CBG

Resource Management [MCC] x

MCC v5.3.7

Project Resources Generate

System

- Clock
- Configuration Bits
- DMT
- ICD
- Interrupt
- Pins
- Watchdog Timer

Device Resources Content Manager

Drivers

- ADC
- CAN FD
- CBG

Pins

Location	Pin Name	Module	Function	Direction	Custom Name	Analog	Start High	Weak Pulldown	Weak Pullup	Open Drain	Interrupt on Change
35	RB4	ICD	PGCx	input		☐	☐	☐	☐	☐	none
34	RB3	ICD	PGDx	input		☐	☐	☐	☐	☐	none
46	RB6	Pins	GPIO	output	LED1	☐	☐	☐	☐	☐	none

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();

↑
API Function Name in *Green Italics* means it's customizable.

Lab1 GPIO Output

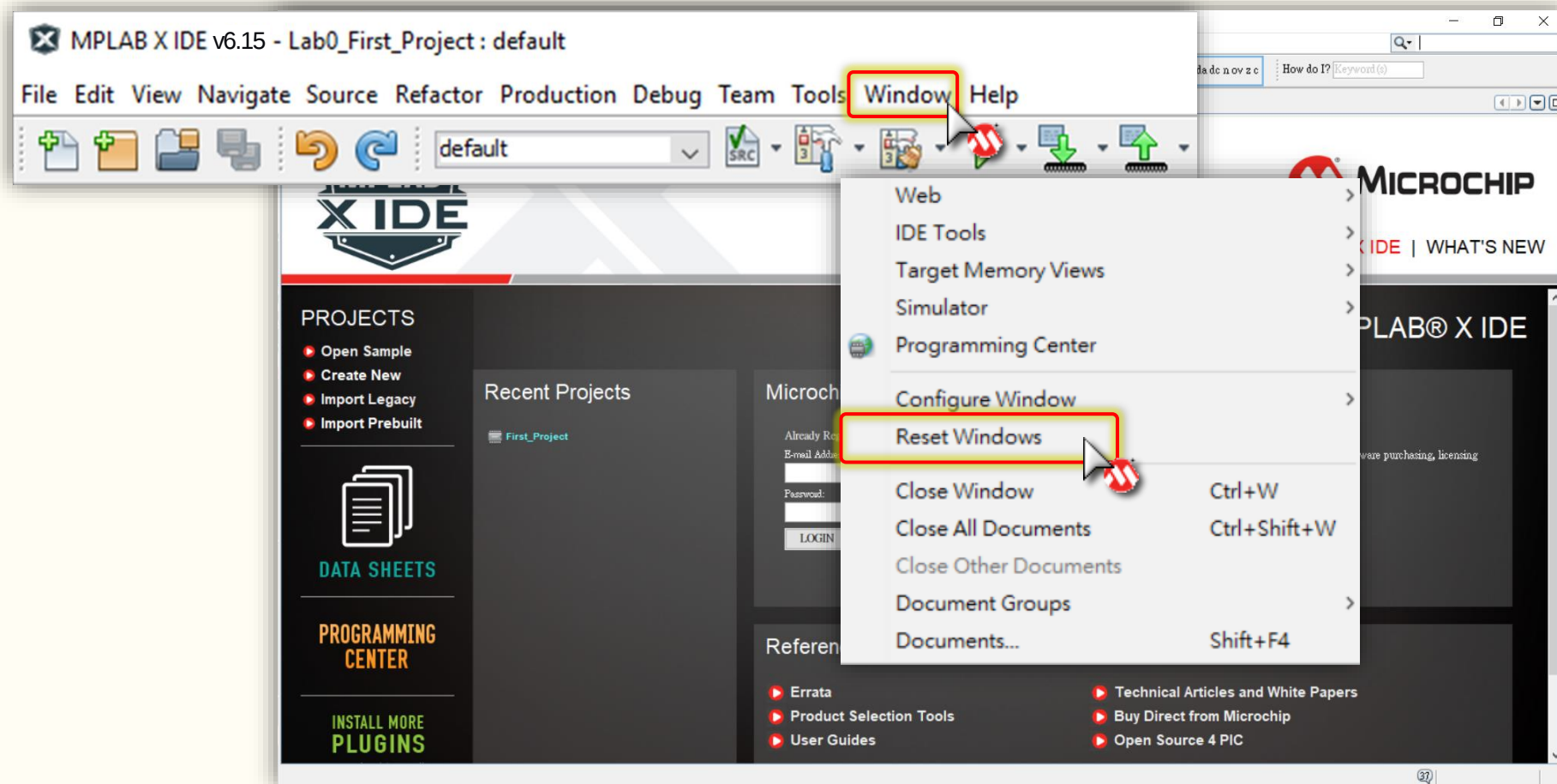
- Base on current project or Open `.\Exercises\Lab1_xxx.X` to do exercise.
- Try to use MCC to generate your first MCC style code and control GPIO base on MCC style.
- Try to set **RB6** to digital output mode, set the initial to **bright**, and toggle output level in period 500ms ~ 1s roughly.
- Please connect **RB6 (Pin11)** to **LED (LED1)** to observe pin status.

How to start ?

MPLAB X IDE Hints

Tips

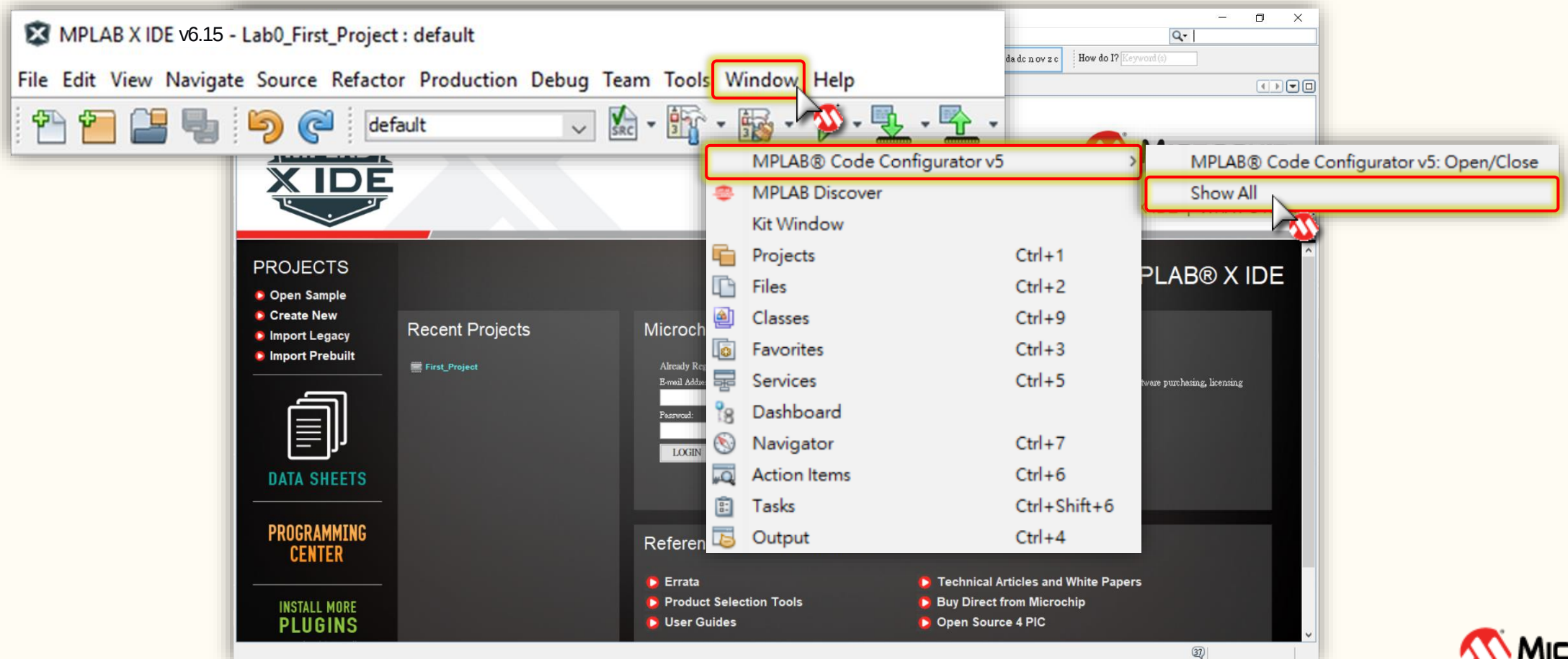
- Too many windows, You may can't find the window you want. MPLAB X IDE provide named "**Reset Windows**" to go back to default layout.



MCC Hints

Tips

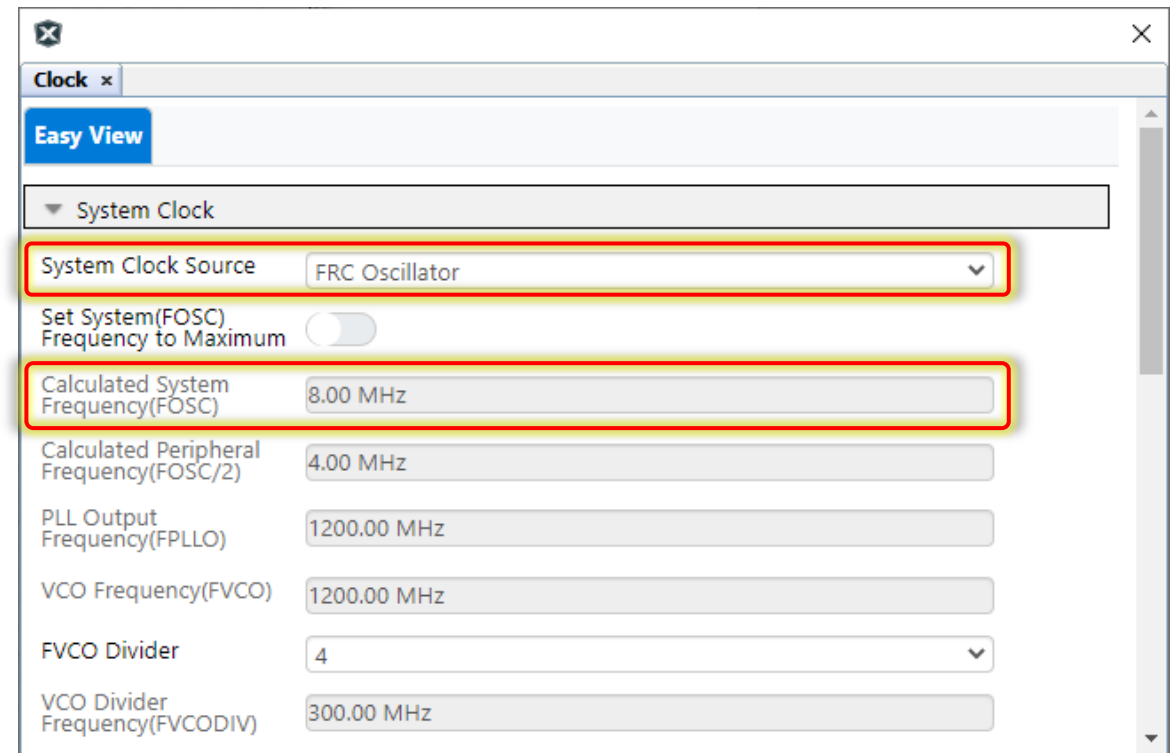
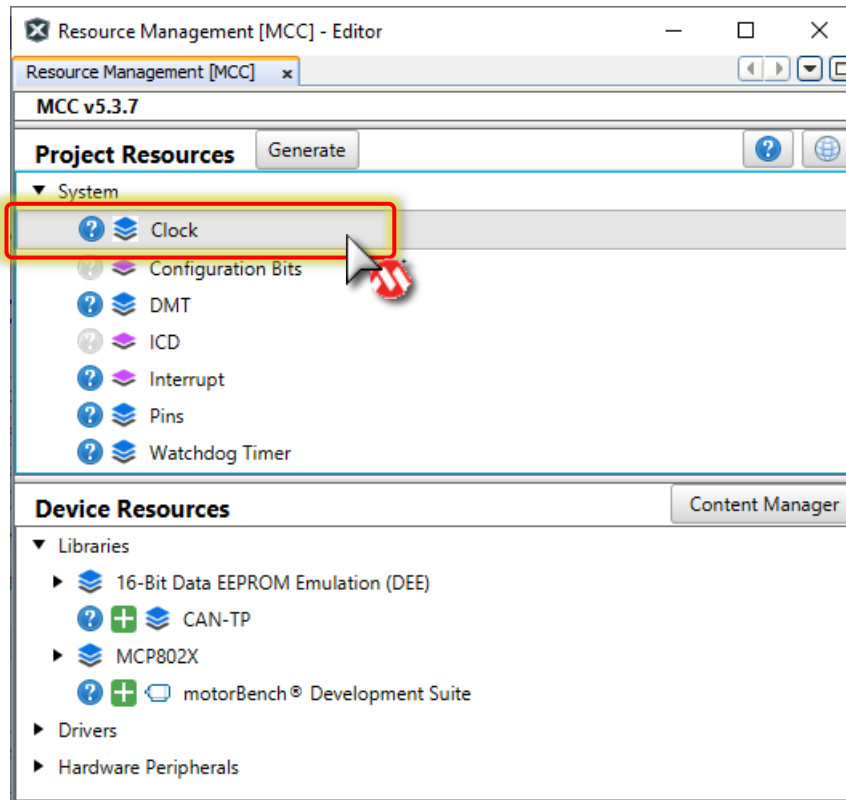
- After Click MCC executed, you may can't find the MCC window you want. MCC provide function named "**Show All**" to show all MCC windows.



Lab1 GPIO Output

Lab Preparation

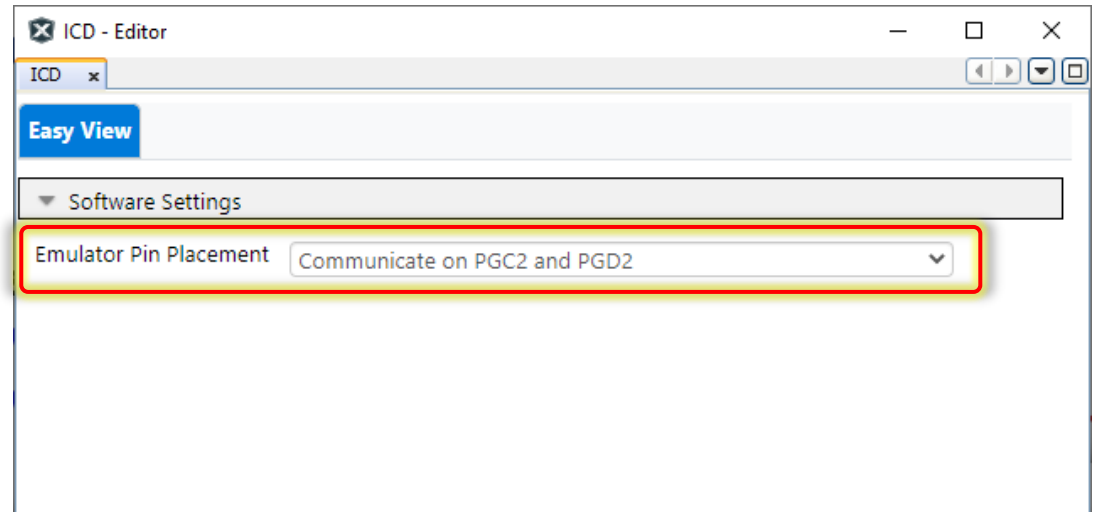
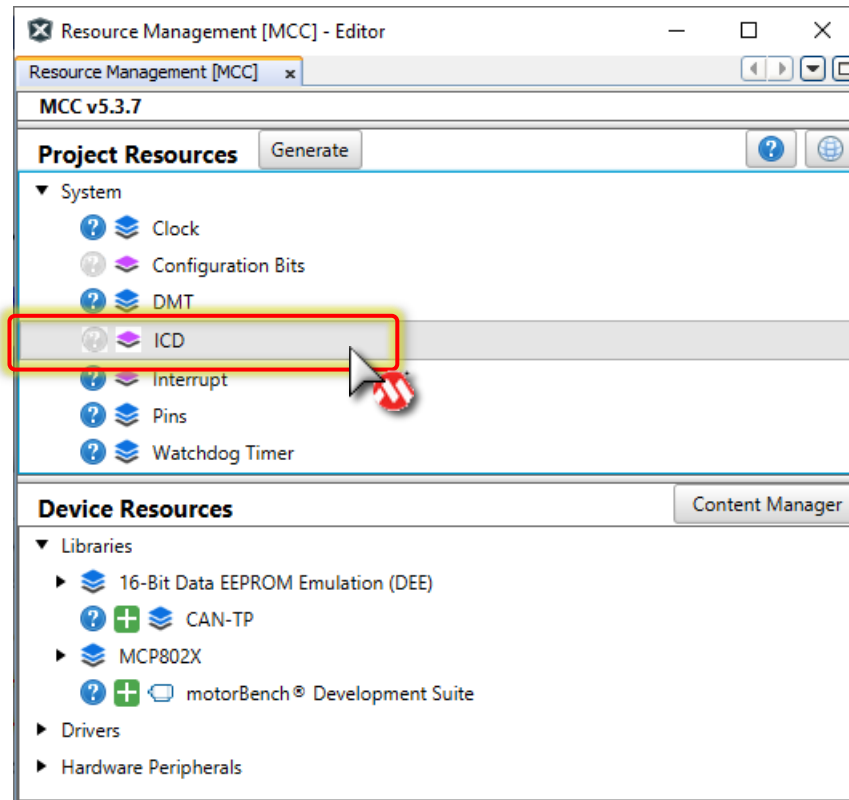
- Check System Clock .
 - **System Clock** : **System Clock Source: FRC Oscillator**, **FOSC: 8.00 MHz**.



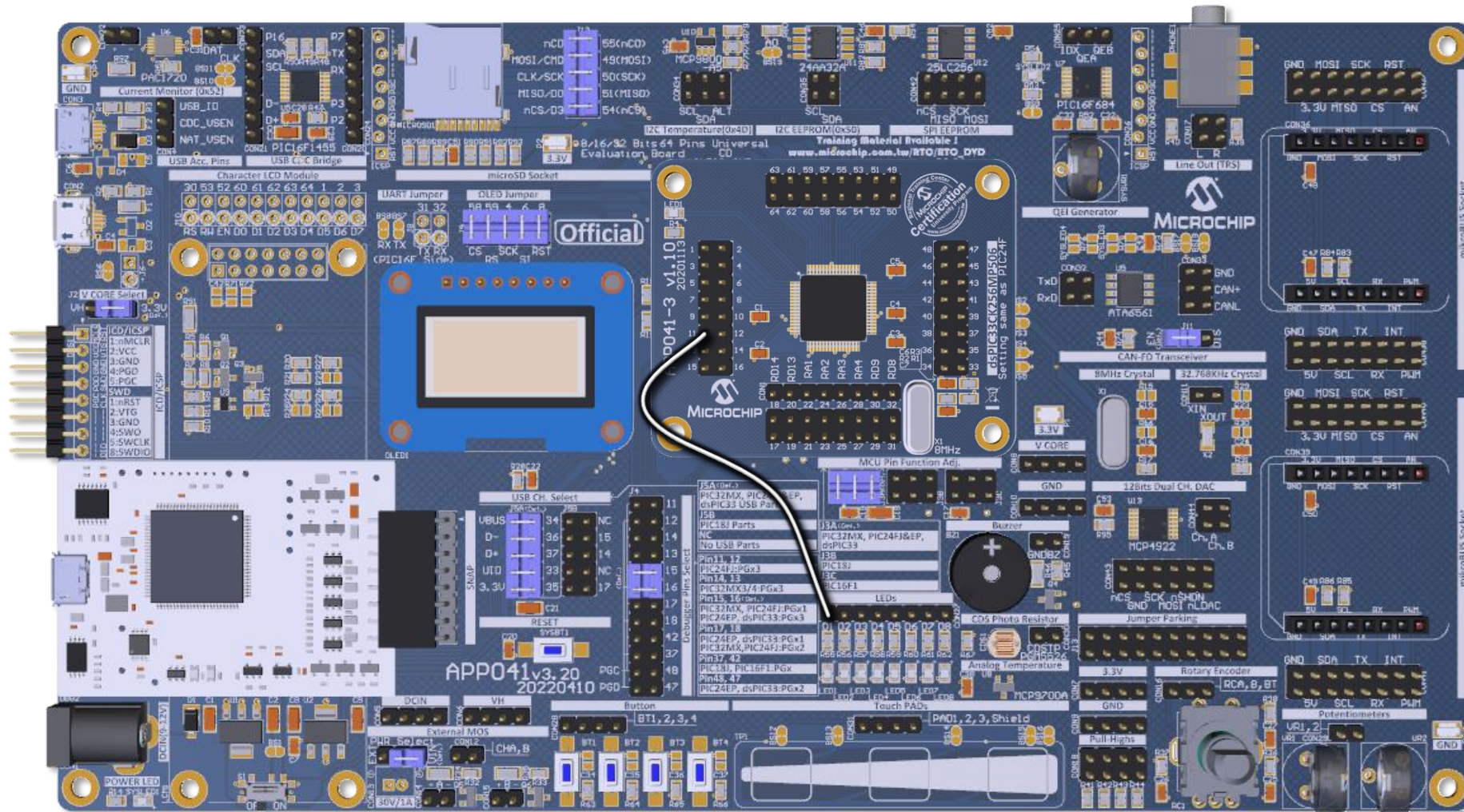
Lab1 GPIO Output

Lab Preparation

- Set Debug Interface.
 - **Software Settings: Emulator Pin Placement: Communicate on PGC2 and PGD2.**



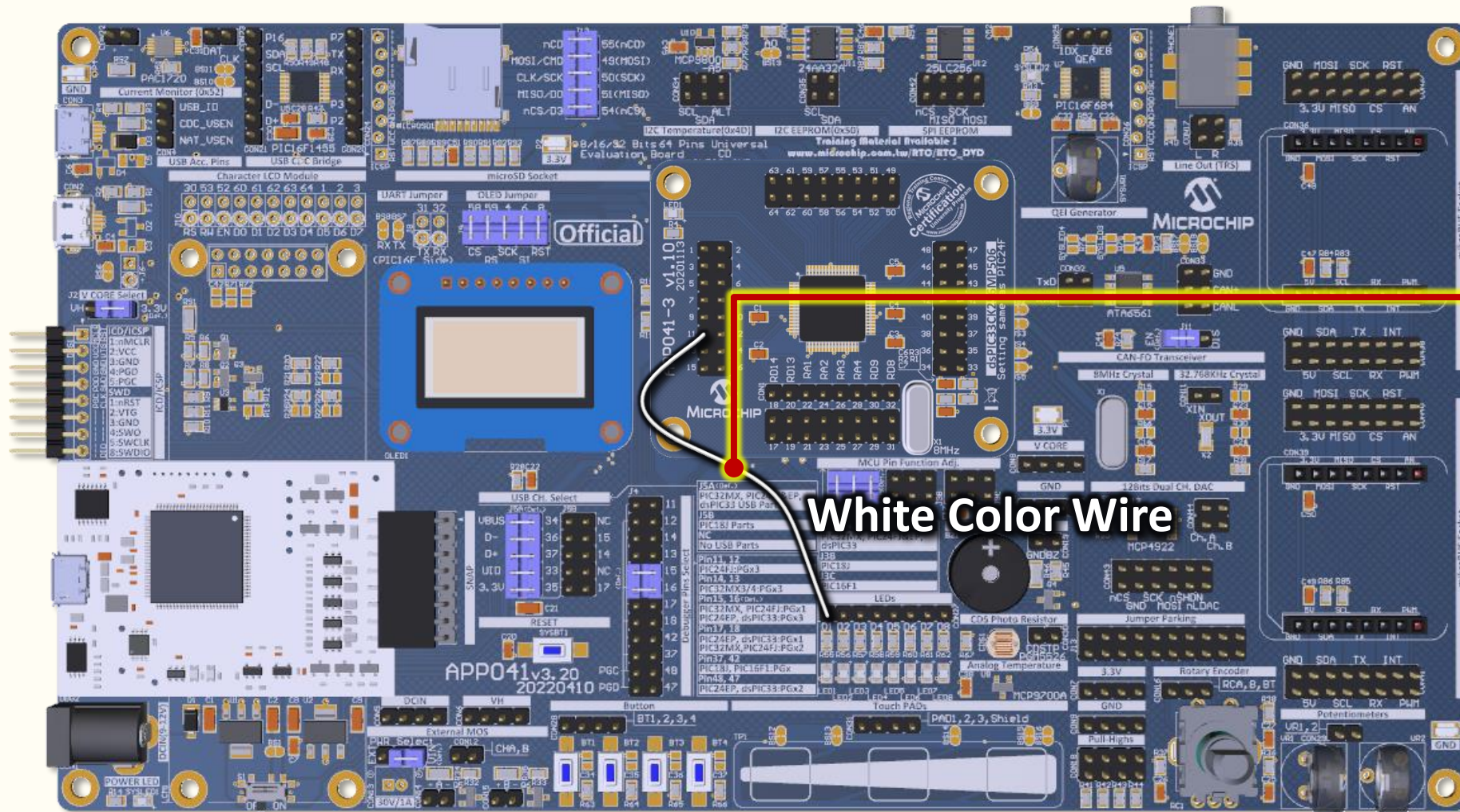
Connection



PIM Pin#		Symbol
11		D1

Lab1 GPIO Output

Note !! Please use the same color wire as presentation slide for easy following debug.



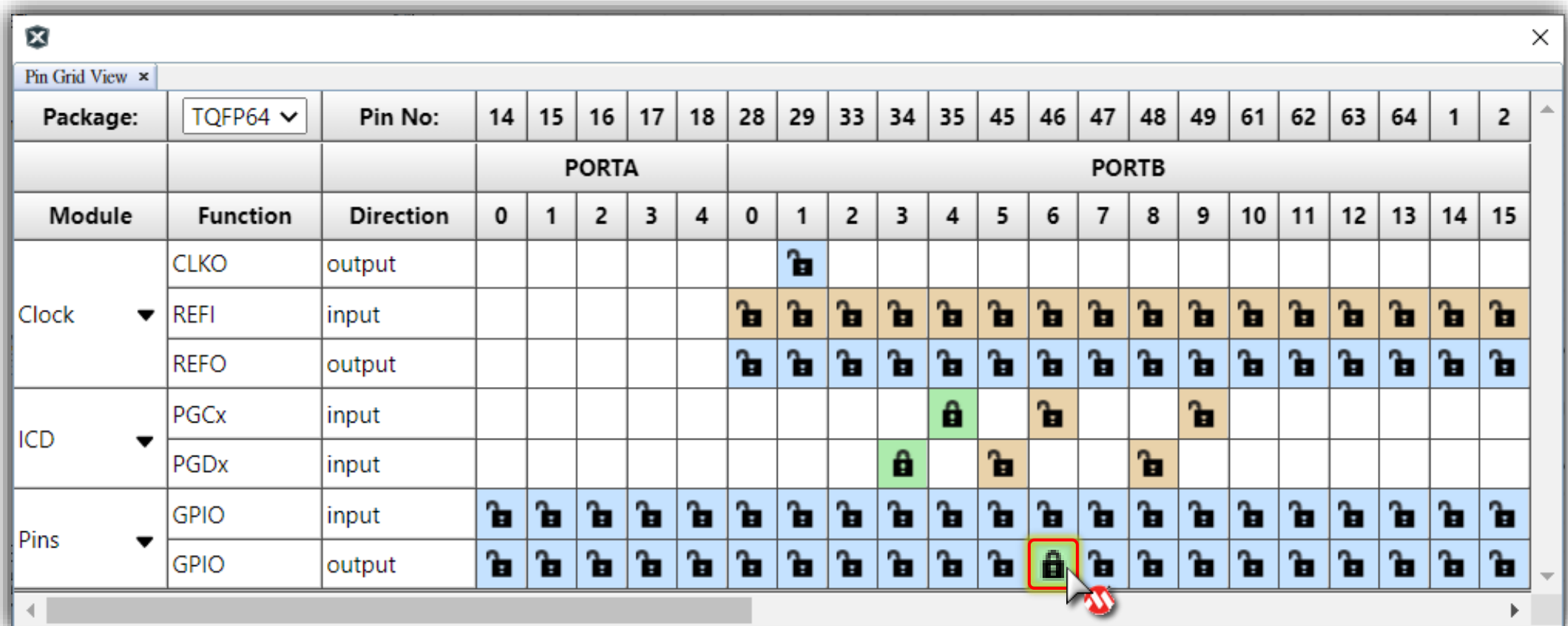
PIM Pin#	Symbol
11	D1

White Color Wire

Lab1 GPIO Output

Step 1

- Set **PORTB : RB6** to digital output mode.
 - **Pin Manager Grid View** : Click **RB6** to lock to **GPIO output**.



Package: TQFP64			Pin No:																							
			14	15	16	17	18	28	29	33	34	35	45	46	47	48	49	61	62	63	64	1	2			
			PORTA					PORTB																		
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Clock	CLKO	output							🔒																	
	REFI	input						🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
	REFO	output						🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
ICD	PGCx	input										🔒		🔒			🔒									
	PGDx	input									🔒		🔒			🔒										
Pins	GPIO	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
	GPIO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				

Lab1 GPIO Output

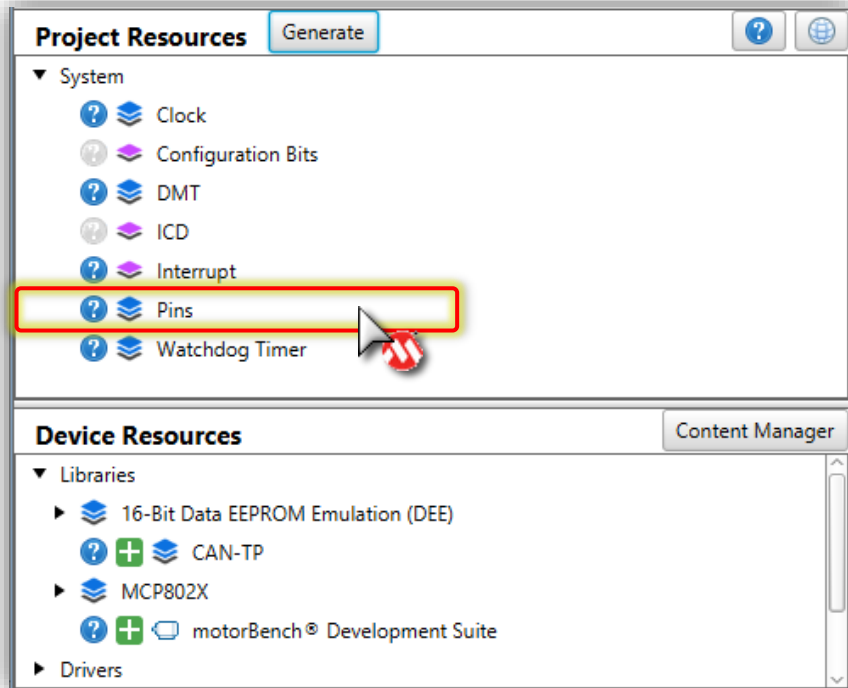
Note !! PIM's Pin number is not equal to real chip Pin number in MCC Pin Grid View.

The screenshot shows the 'Pin Grid View' window in Proteus. The package is set to 'TQFP64'. The pin grid shows various functions like CLKO, REFI, REFO, PGCx, PGDx, GPIO, and PPS. A yellow box highlights pin 46, which is connected to the PIC24FJ064GB010. A red box highlights pin 11, which is connected to the PIC24FJ064GB010. A red box highlights pin 11, which is connected to the PIC24FJ064GB010.

Lab1 GPIO Output

Step 2

- Set Alias for RB6.
 - Pins : **RB6** Custom Name → **LED1**, Check Output, Start High



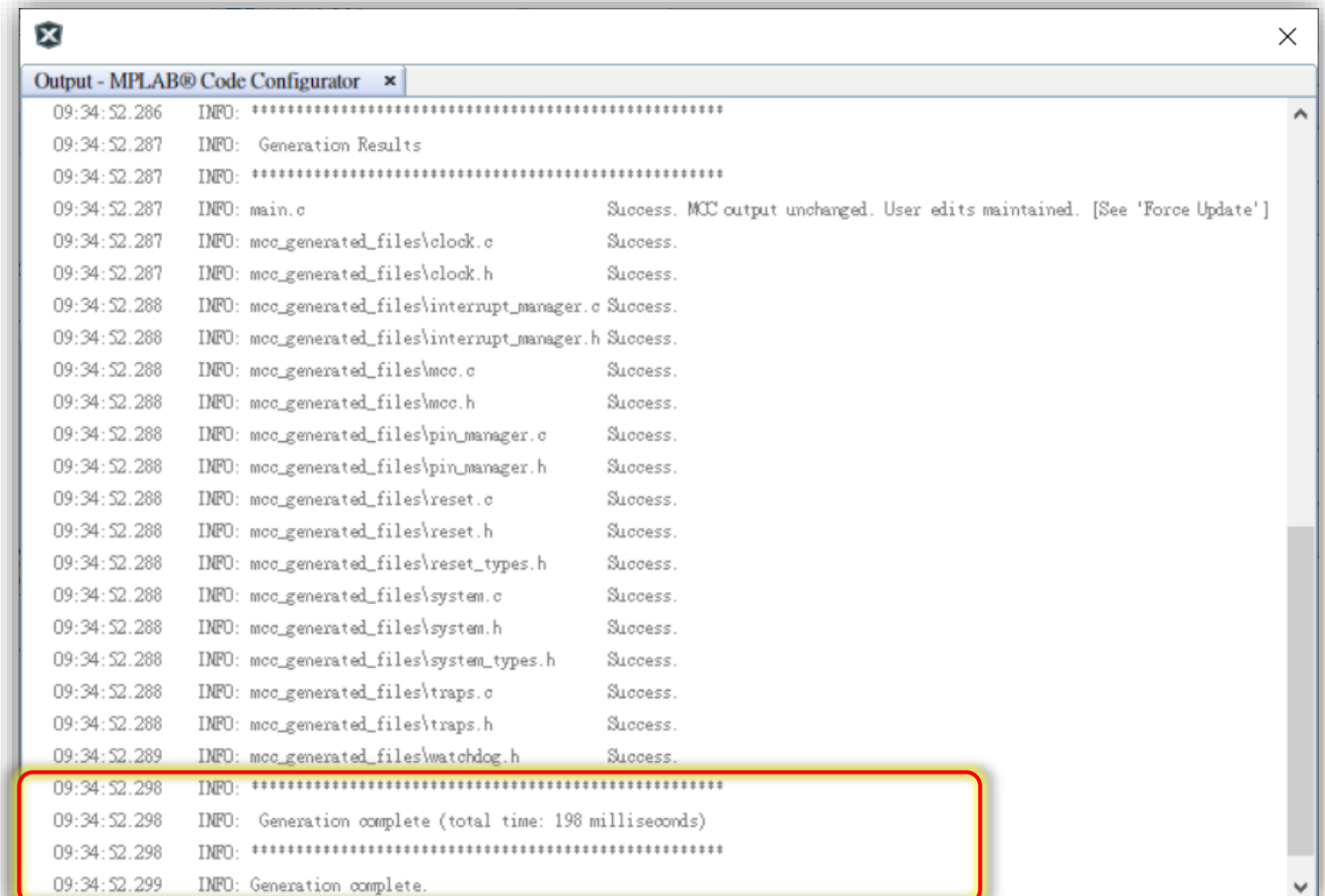
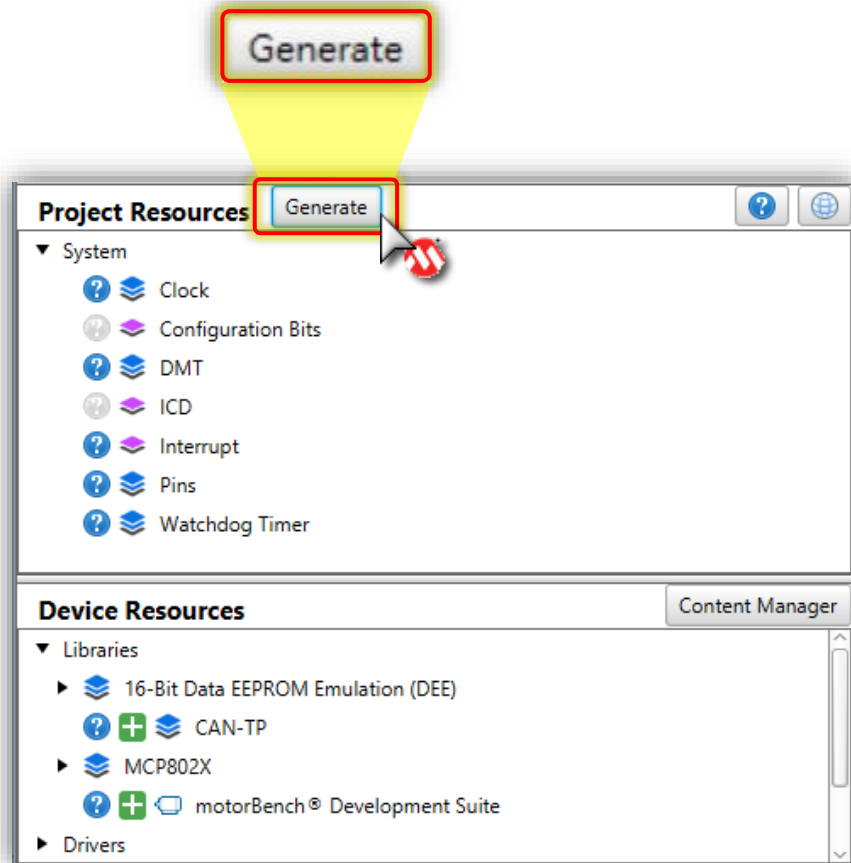
Location	Pin Name	Module	Function	Direction	Custom Name	Analog	Start High	Weak Pulldown	Weak Pullup	Open Drain	Interrupt on Change
35	RB4	ICD	PGCx	input		☐	☐	☐	☐	☐	none ▼
34	RB3	ICD	PGDx	input		☐	☐	☐	☐	☐	none ▼
46	RB6	Pins	GPIO	output	LED1	☐	☒	☐	☐	☐	none ▼

Click & Rename

Lab1 GPIO Output

Step 3

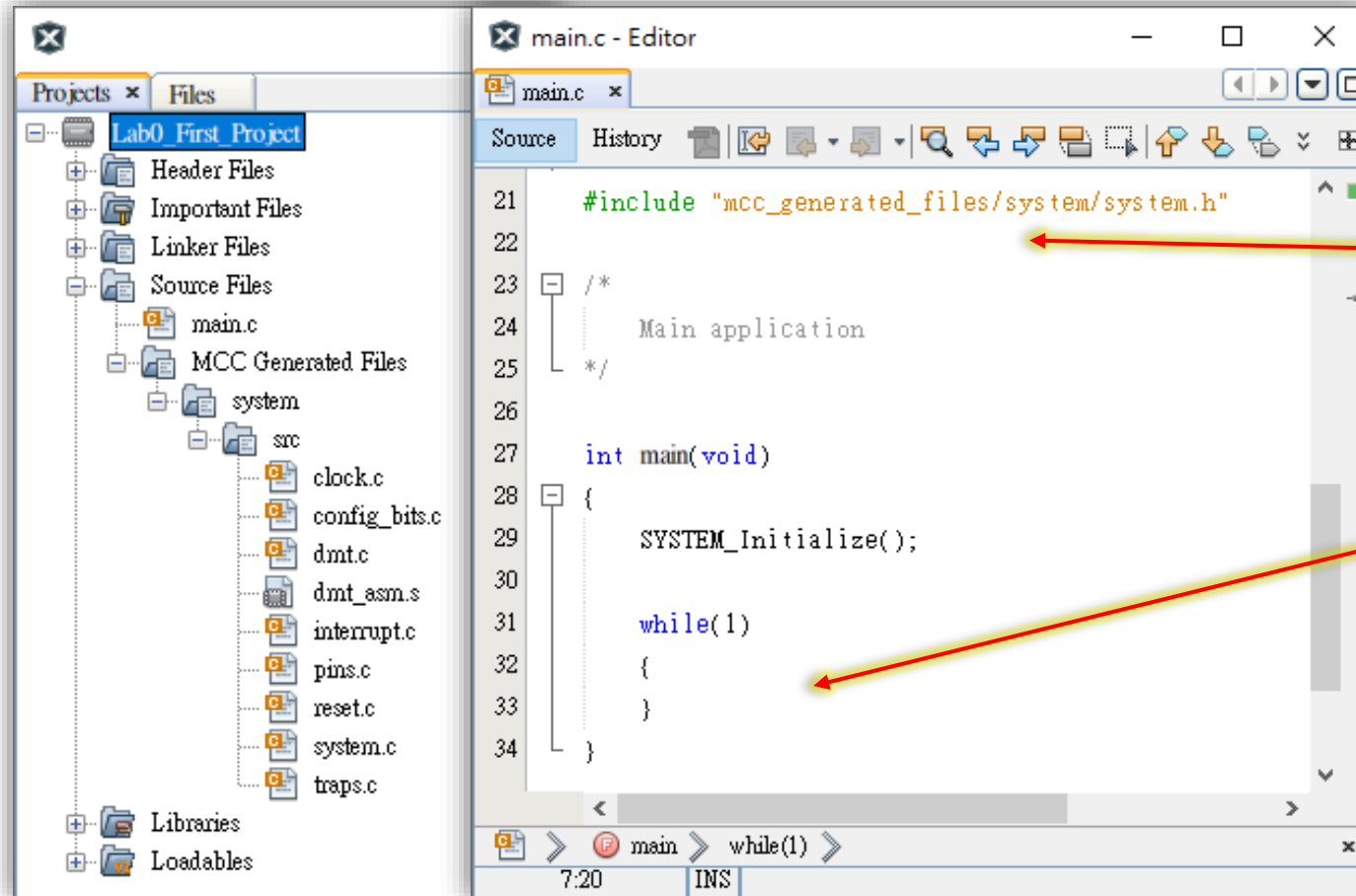
- Click "**Generate**" Button to generate your code.



Lab1 GPIO Output

Step 4

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



Add below code to main()

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

uint32_t i = 0;

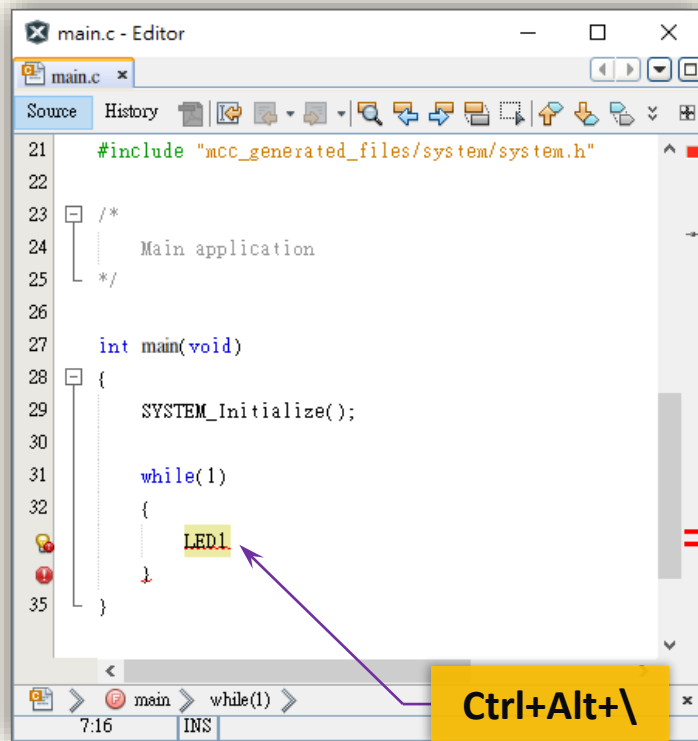
int main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        LED1_Toggle();
        for (i = 0; i < 80000; i++);
    }
}
```

MPLAB X IDE Hints

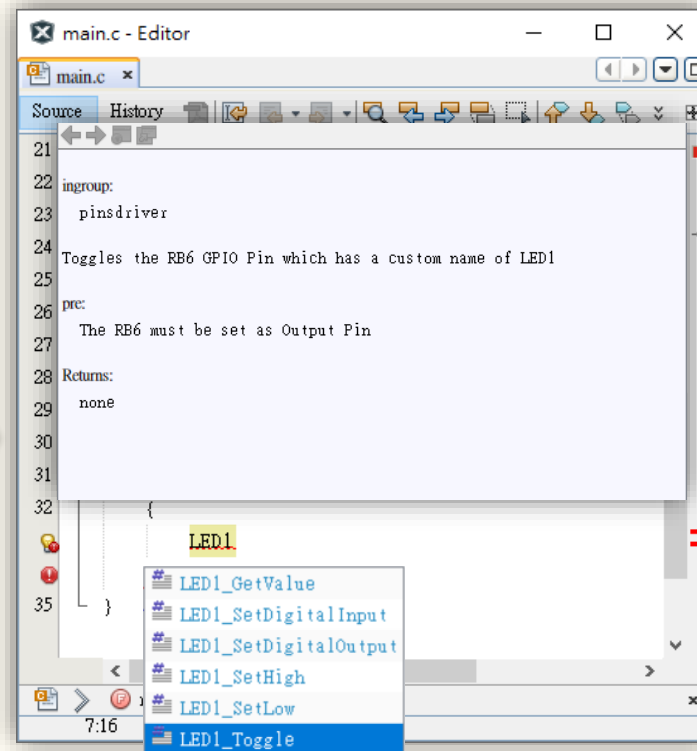
Tips

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



```
21 #include "mcc_generated_files/system/system.h"
22
23 /*
24  * Main application
25  */
26
27 int main(void)
28 {
29     SYSTEM_Initialize();
30
31     while(1)
32     {
33         LED1
34     }
35 }
```

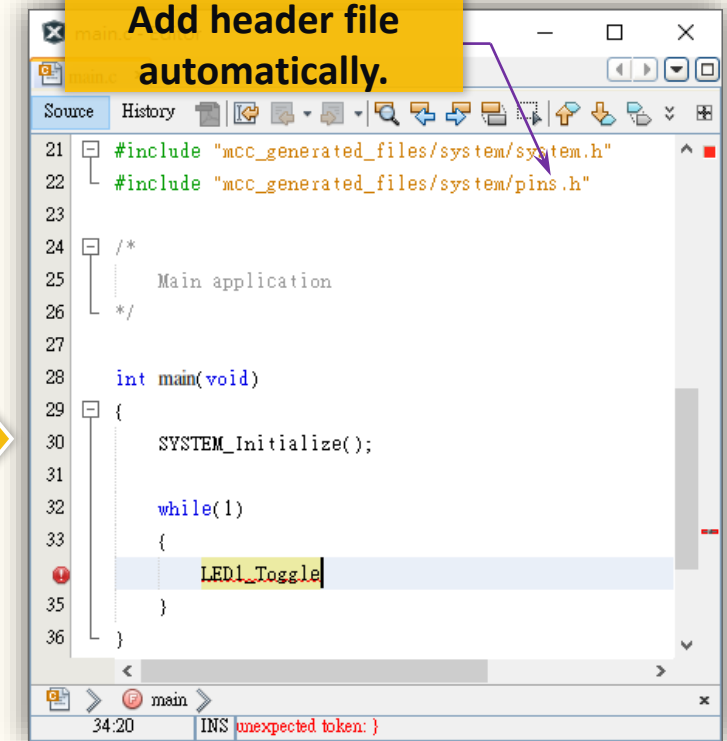
Ctrl+Alt+\'



```
21 ingroup:
22 pinsdriver
23
24 Toggles the RB6 GPIO Pin which has a custom name of LED1
25
26 pre:
27 The RB6 must be set as Output Pin
28
29 Returns:
30 none
31
32 {
33     LED1
34 }
35 }
```

LED1_Toggle

Add header file automatically.



```
21 #include "mcc_generated_files/system/system.h"
22 #include "mcc_generated_files/system/pins.h"
23
24 /*
25  * Main application
26  */
27
28 int main(void)
29 {
30     SYSTEM_Initialize();
31
32     while(1)
33     {
34         LED1_Toggle
35     }
36 }
```

Add header file automatically.

Lab1 GPIO Output

Code Example

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

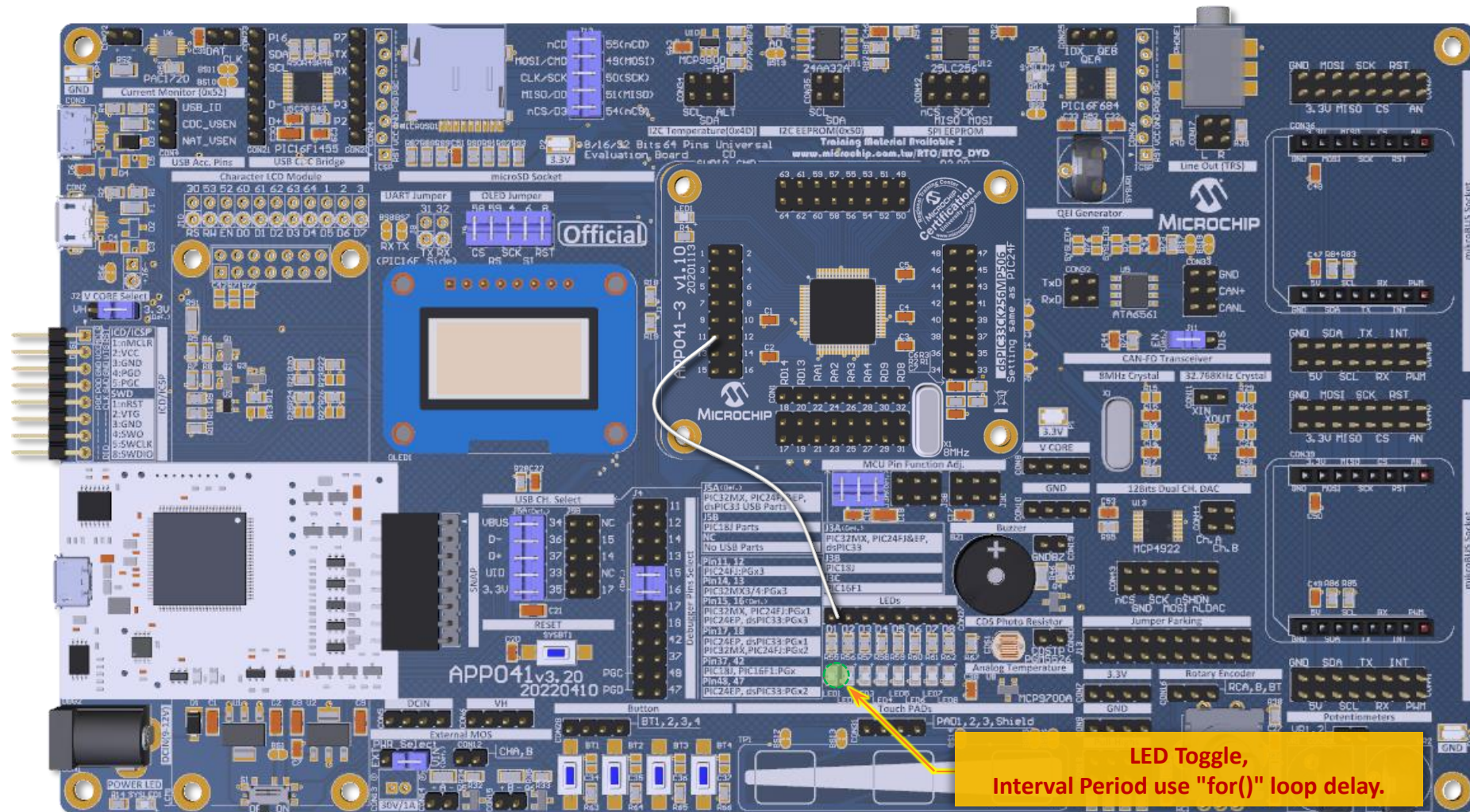
uint32_t i = 0;

int main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        LED1_Toggle();
        for (i = 0; i < 80000; i++);
    }
}
```

Lab1 GPIO Output

Result

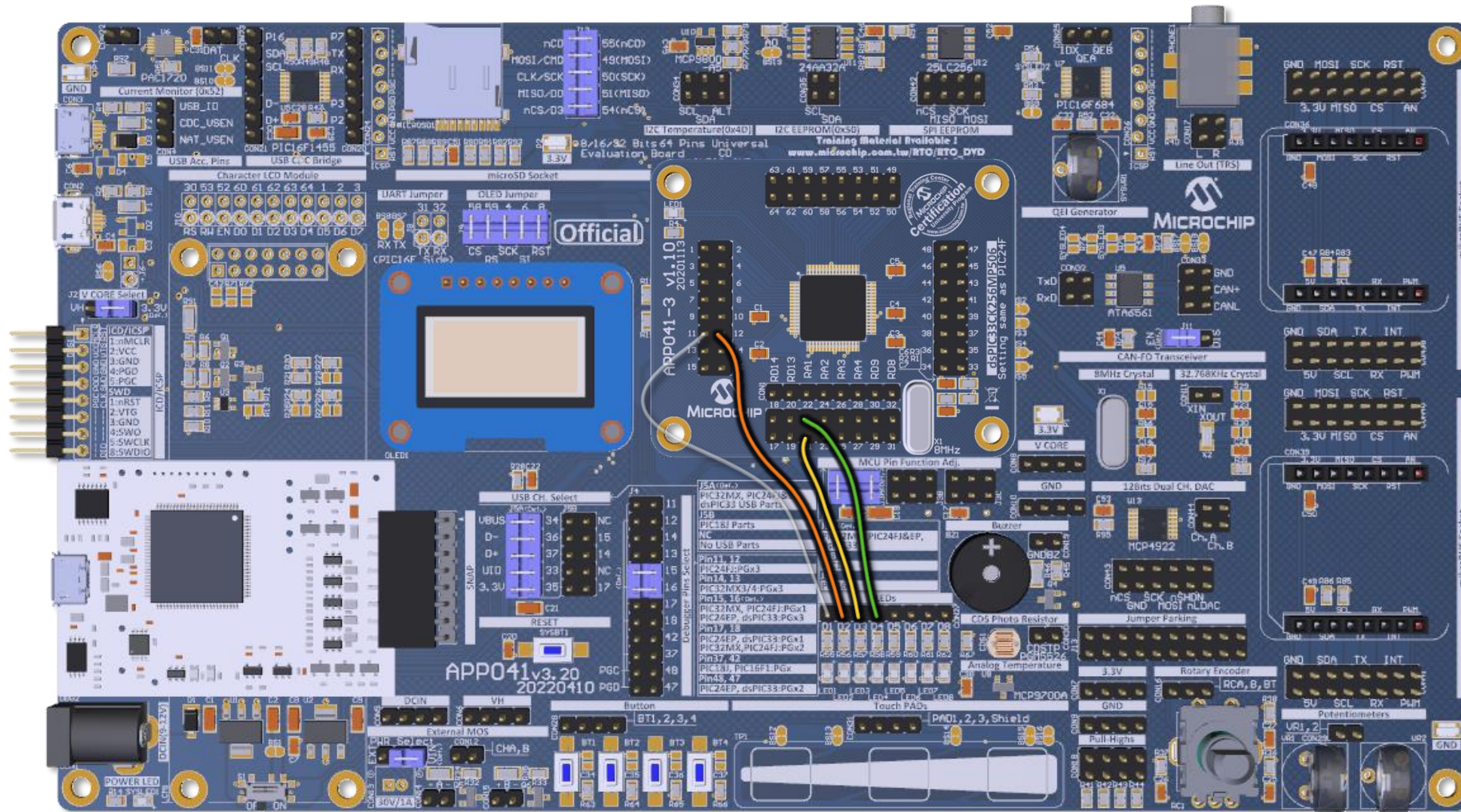


Lab2 Multi-GPIO Output

- Base on current project or Open `.\Exercises\Lab2_xxx.X` to do exercise.
- Try to set **more GPIO** pins to digital output mode and make LED flash alternately in period 500ms ~ 1s roughly.
- Please connect
 - **RB5 (Pin12)** to **LED (LED2)**
 - **RC1 (Pin22)** to **LED (LED4)**
 - **RD12 (Pin21)** to **LED (LED3)**

Let's go!

Connection



PIM Pin#		Symbol
12		D2
21		D3
22		D4

Lab2 Multi-GPIO Output

Step 1

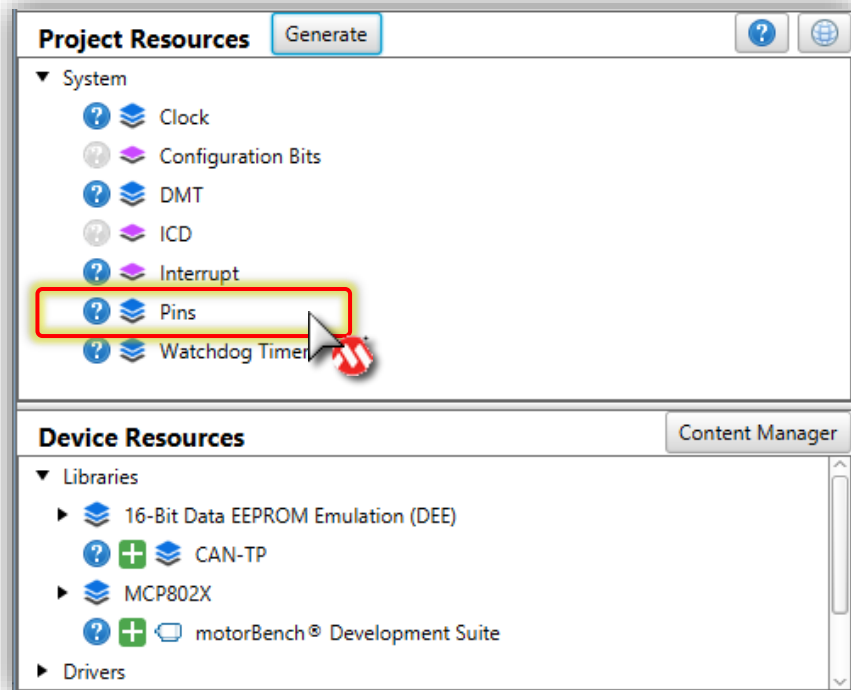
- Set **PORTB** : **RB5**, **PORTC** : **RC1**, **PORTD** : **RD12**, to digital output mode.

Pin Grid View																																																		
Package:	TQFP64	Pin No:	45	46	47	48	49	61	62	63	64	1	2	13	22	23	27	50	51	24	32	36	37	52	53	3	4	5	6	60	59	58	55	54	44	43	42	39	38	31	30	21								
			PORTB																PORTC																PORTD															
Module	Function	Direction	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12								
Clock	CLKO	output																																																
	REFI	input																																																
	REFO	output																																																
ICD	PGCx	input																																																
	PGDx	input																																																
Pins	GPIO	input																																																
	GPIO	output																																																

Lab2 Multi-GPIO Output

Step 2

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



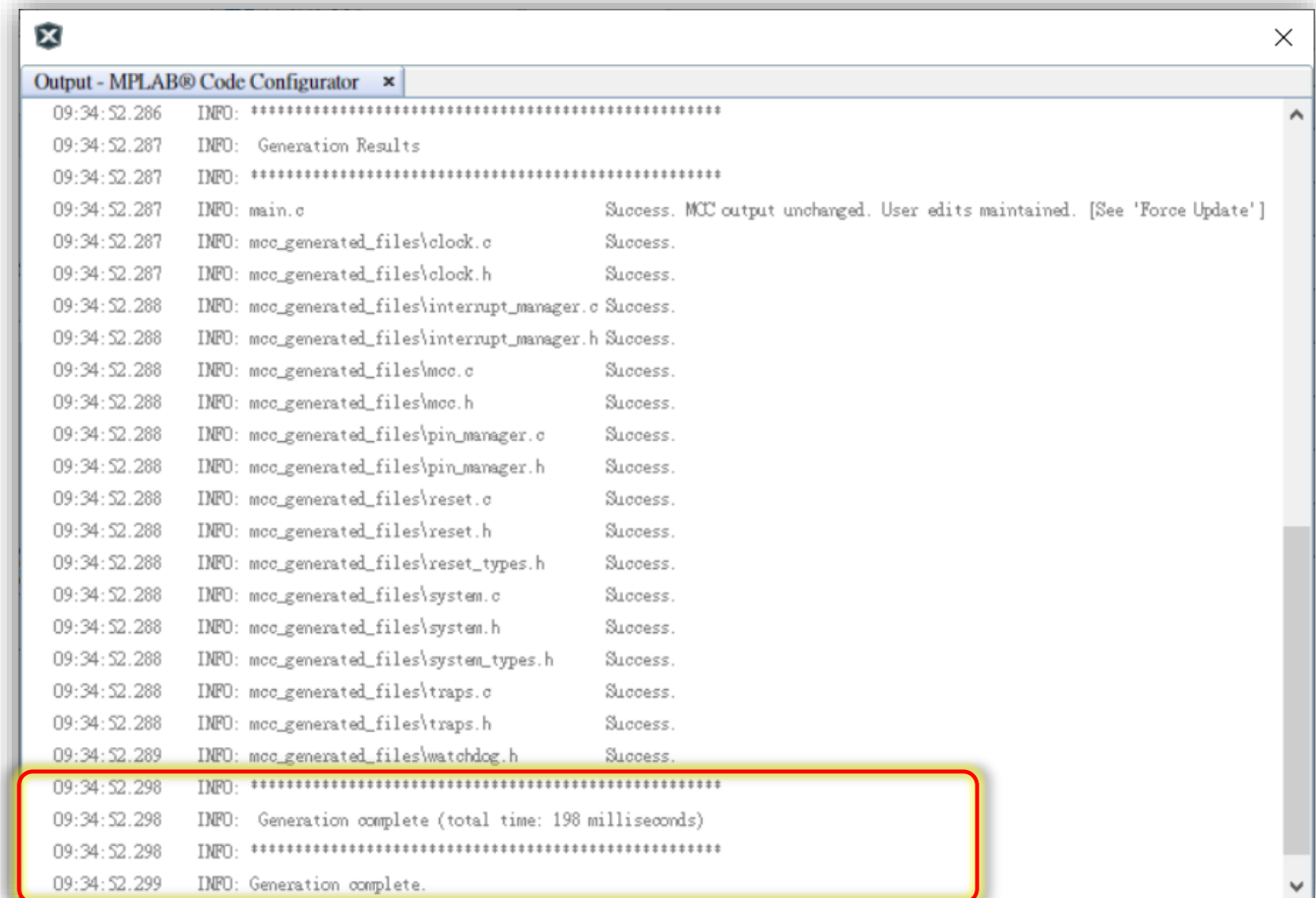
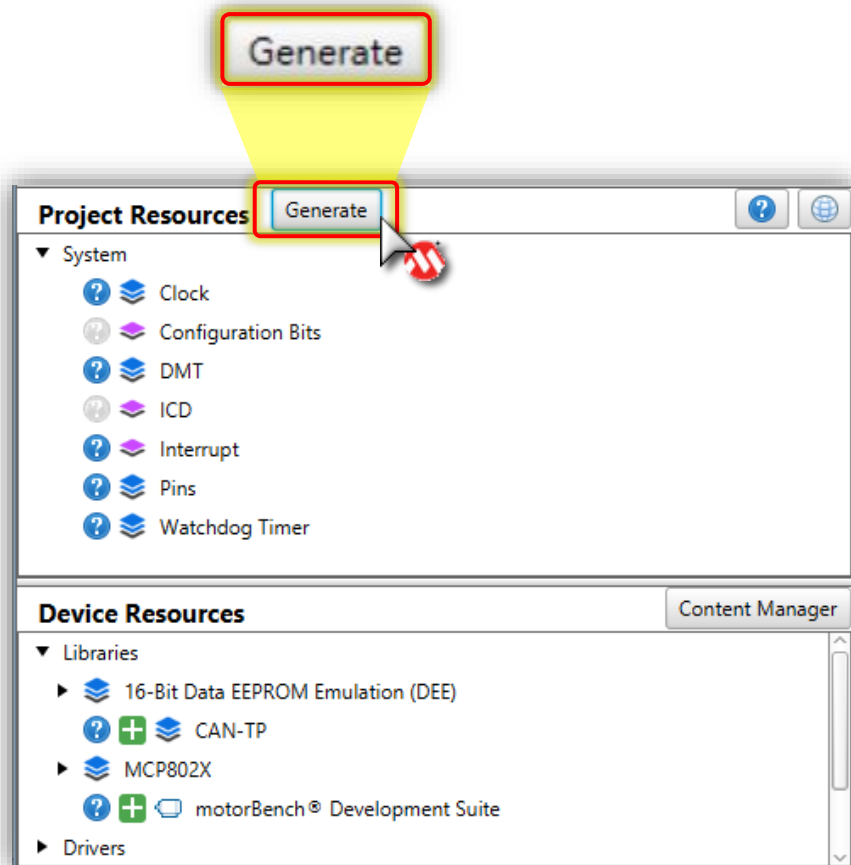
Location	Pin Name	Module	Function	Direction	Custom Name	Analog	Start High	Weak Pulldown	Weak Pullup	Open Drain	Interrupt on Change
35	RB4	ICD	PGCx	input		☐	☐	☐	☐	☐	none ▼
34	RB3	ICD	PGDx	input		☐	☐	☐	☐	☐	none ▼
45	RB5	Pins	GPIO	output	LED2		☐	☐	☐	☐	none ▼
46	RB6	Pins	GPIO	output	LED1		☒	☐	☐	☐	none ▼
22	RC1	Pins	GPIO	output	LED4	☐	☐	☐	☐	☐	none ▼
21	RD12	Pins	GPIO	output	LED3		☒	☐	☐	☐	none ▼

Click & Rename

Lab2 Multi-GPIO Output

Step 3

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



Lab2 Multi-GPIO Output

Code Example

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

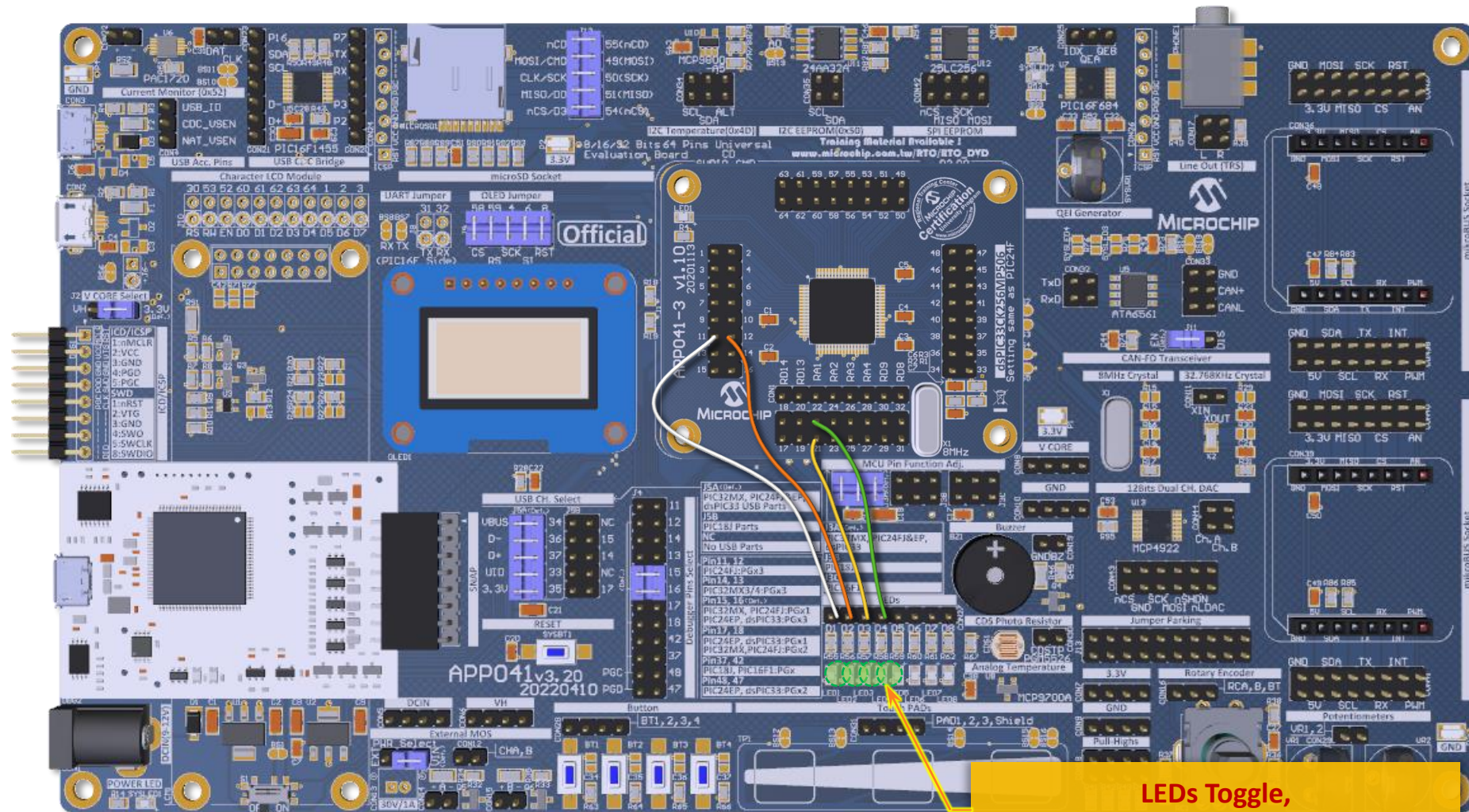
uint32_t i = 0;

int main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        LED1_Toggle();
        LED2_Toggle();
        LED3_Toggle();
        LED4_Toggle();
        for (i = 0; i < 80000; i++);
    }
}
```

Lab2 Multi-GPIO Output

Result



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

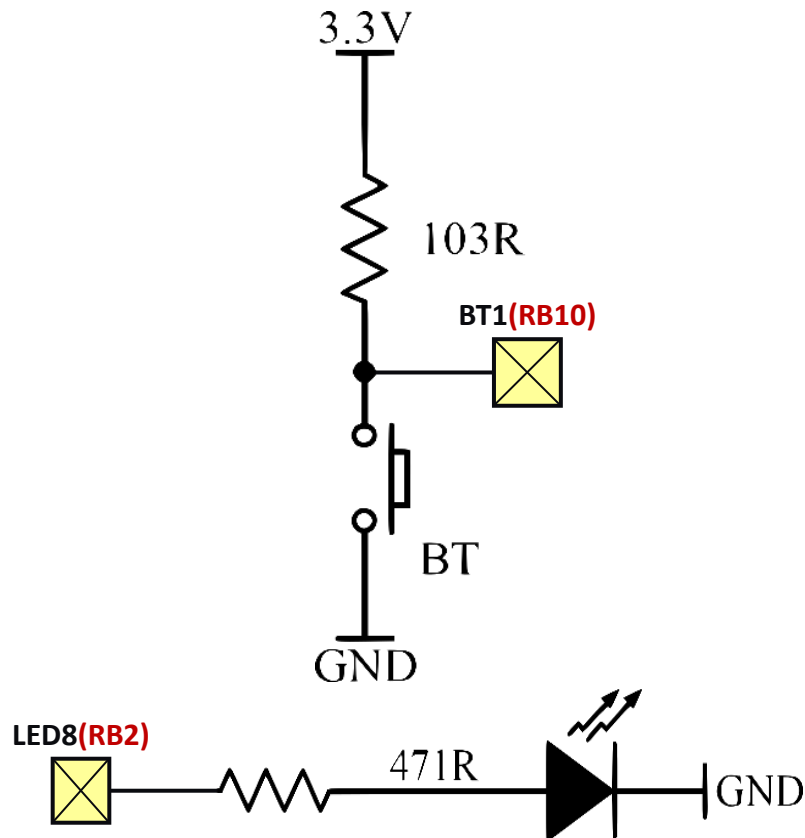
Lab3 GPIO Input

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

Let's go!

Lab3 GPIO Input

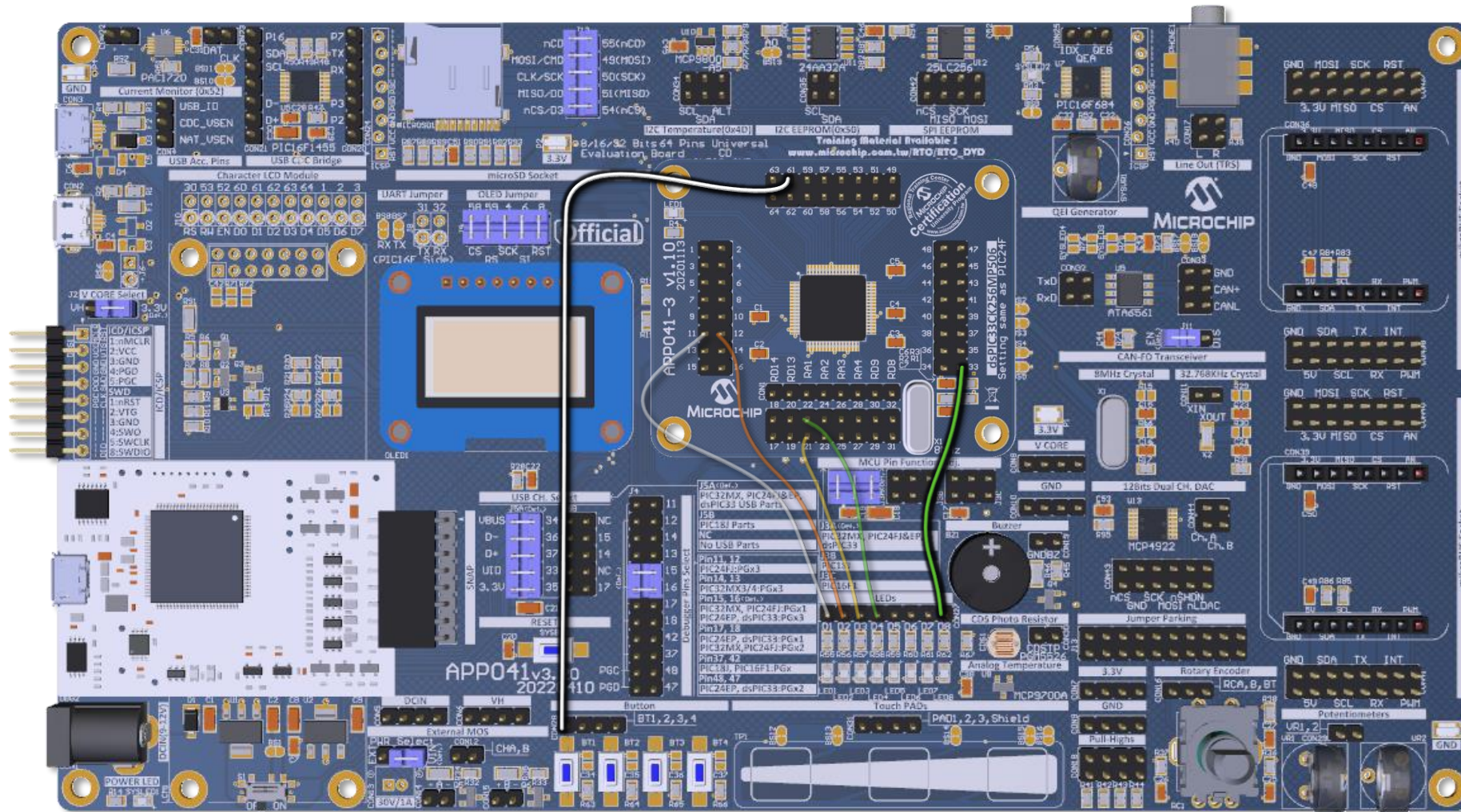
- *CustomName_GetValue();* function is used for get digital input level from outside. The simple example code shows below:



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

Lab3 GPIO Input

Connection



PIM Pin#		Symbol
61		BT1
33		D8

Lab3 GPIO Input

Step 1

- Set **PORTB** : **RB2** to digital **output** mode.
- Set **PORTB** : **RB10** to digital **input** mode.

Pin Grid View

Package:

TQFP64

Pin No:

1415161718282933343545464748496162636412

PORTA

PORTB

Module

Function

Direction

012340123456789101112131415

Clock

▼

CLKO

output

REFI

input

REFO

output

ICD

▼

PGCx

input

PGDx

input

Pins

▼

GPIO

input

GPIO

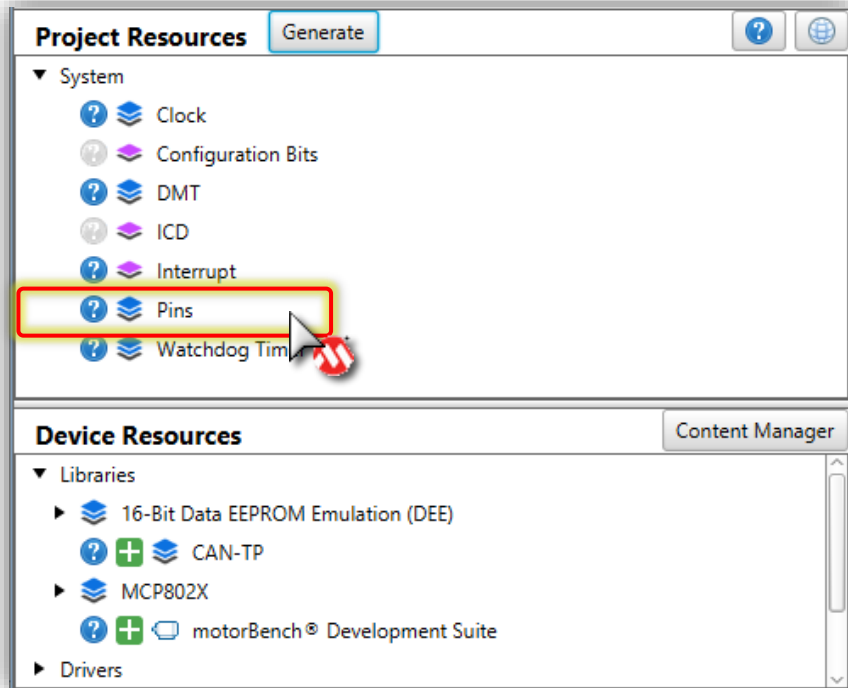
output

<

Lab3 GPIO Input

Step 2

- Set Alias for RB2 and RB10.
 - Pins : **RB2** Custom Name → **LED8**, Check Output
 - Pins : **RB10** Custom Name → **BT1**, Check Input

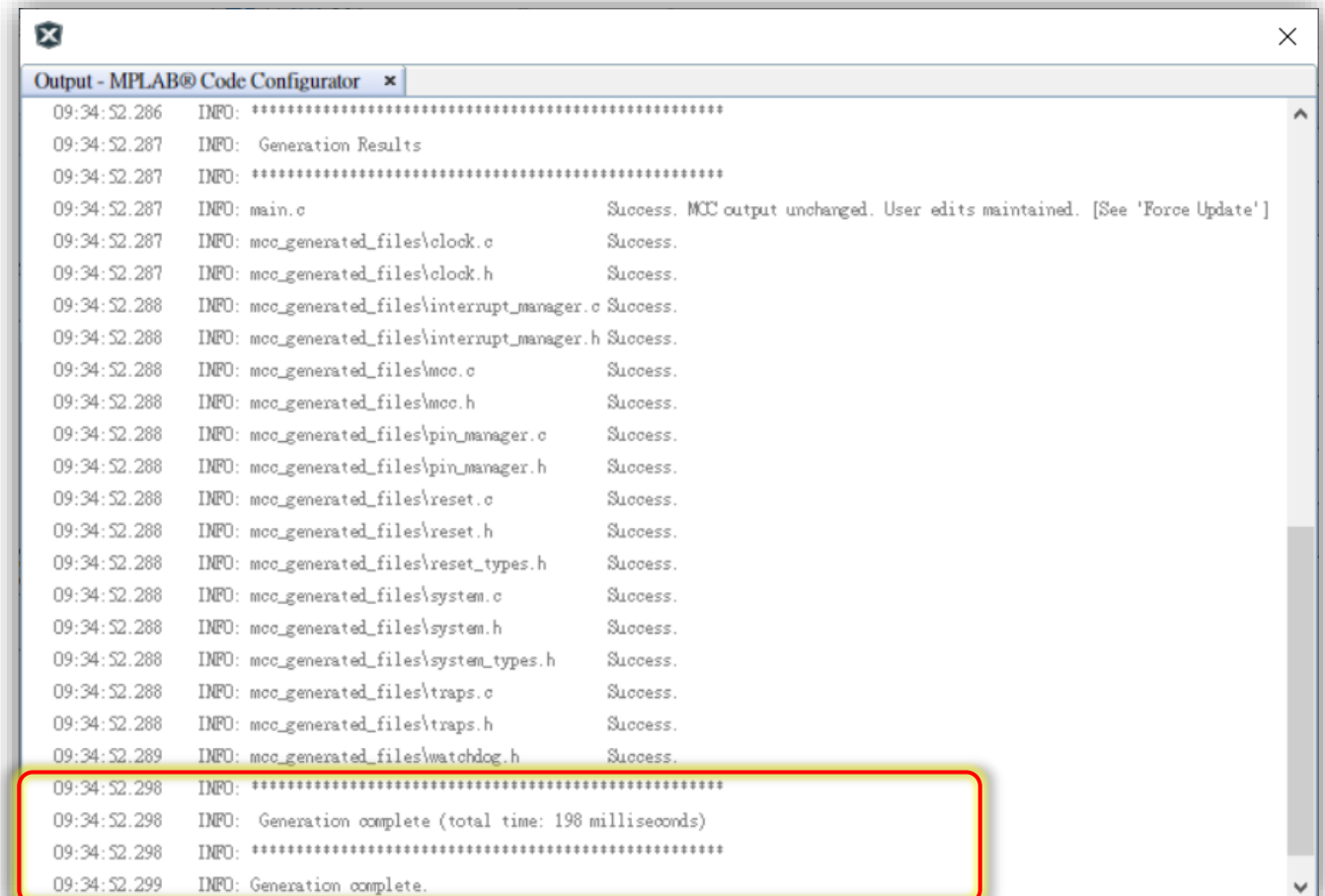
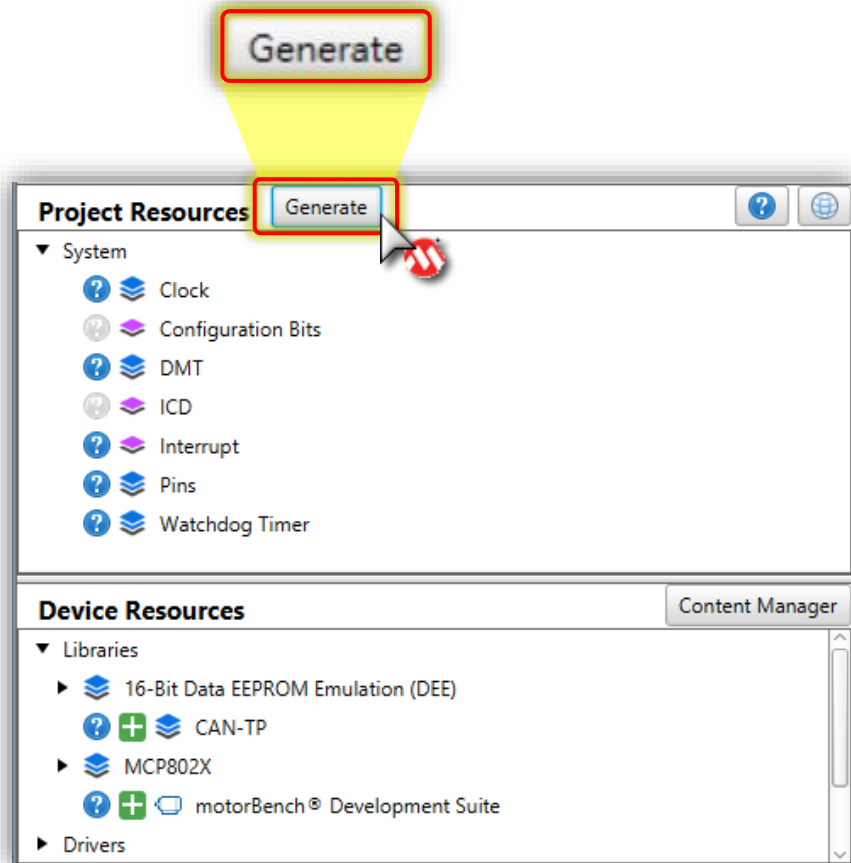


Location	Pin Name	Module	Function	Direction	Custom Name	Analog	Start High	Weak Pulldown	Weak Pullup	Open Drain	Interrupt on Change
35	RB4	ICD	PGCx	input		☐	☐	☐	☐	☐	none ▼
34	RB3	ICD	PGDx	input		☐	☐	☐	☐	☐	none ▼
61	RB10	Pins	GPIO	input	BT1		☐	☐	☐	☐	none ▼
33	RB2	Pins	GPIO	output	LED8	☐	☐	☐	☐	☐	none ▼
45	RB5	Pins	GPIO	output	LED2		☐	☐	☐	☐	none ▼
46	RB6	Pins	GPIO	output				☐	☐	☐	none ▼
22	RC1	Pins	GPIO	output	LED4	☐	☐	☐	☐	☐	none ▼
21	RD12	Pins	GPIO	output	LED3		☒	☐	☐	☐	none ▼

Lab3 GPIO Input

Step 3

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



Lab3 GPIO Input

Code Example

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

uint32_t i = 0;

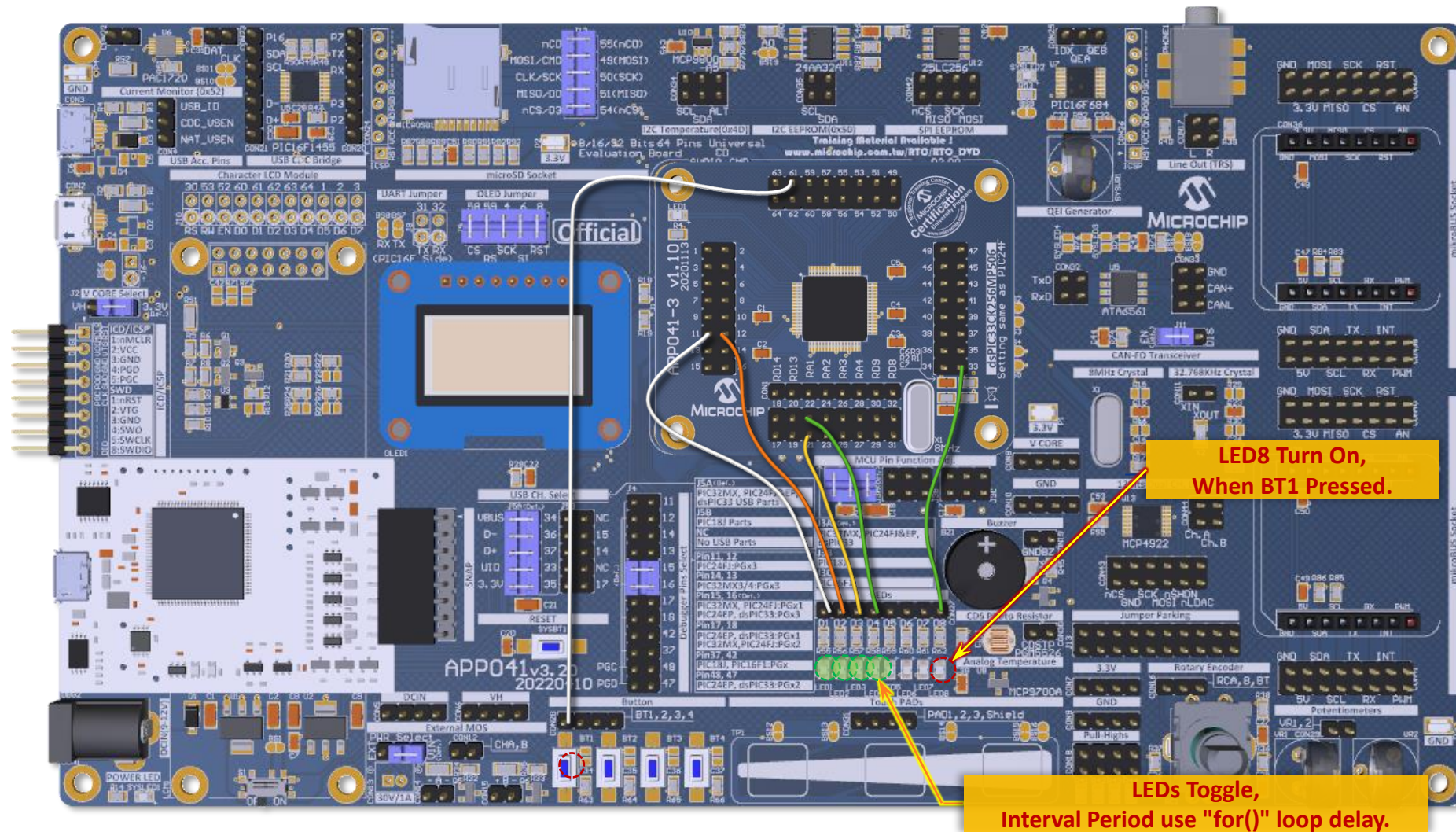
int main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        LED1_Toggle();
        LED2_Toggle();
        LED3_Toggle();
        LED4_Toggle();
        for (i = 0; i < 80000; i++);

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

Lab3 GPIO Input

Result



Lab3 GPIO Input

Button Response Discuss

- Button Response Time Issue ?

The software delay will influence the polling interval.

Try to change coding style to state machine mode.

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

uint32_t i = 0;

int main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        LED1_Toggle();
        LED2_Toggle();
        LED3_Toggle();
        LED4_Toggle();
        for (i = 0; i < 80000; i++);

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



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

uint32_t i = 0;

int main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        if( i++ > 80000 )
        {
            i = 0;
            LED1_Toggle();
            LED2_Toggle();
            LED3_Toggle();
            LED4_Toggle();
        }

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

Lab3 GPIO Input

Code Example (State Machine)

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

uint32_t i = 0;

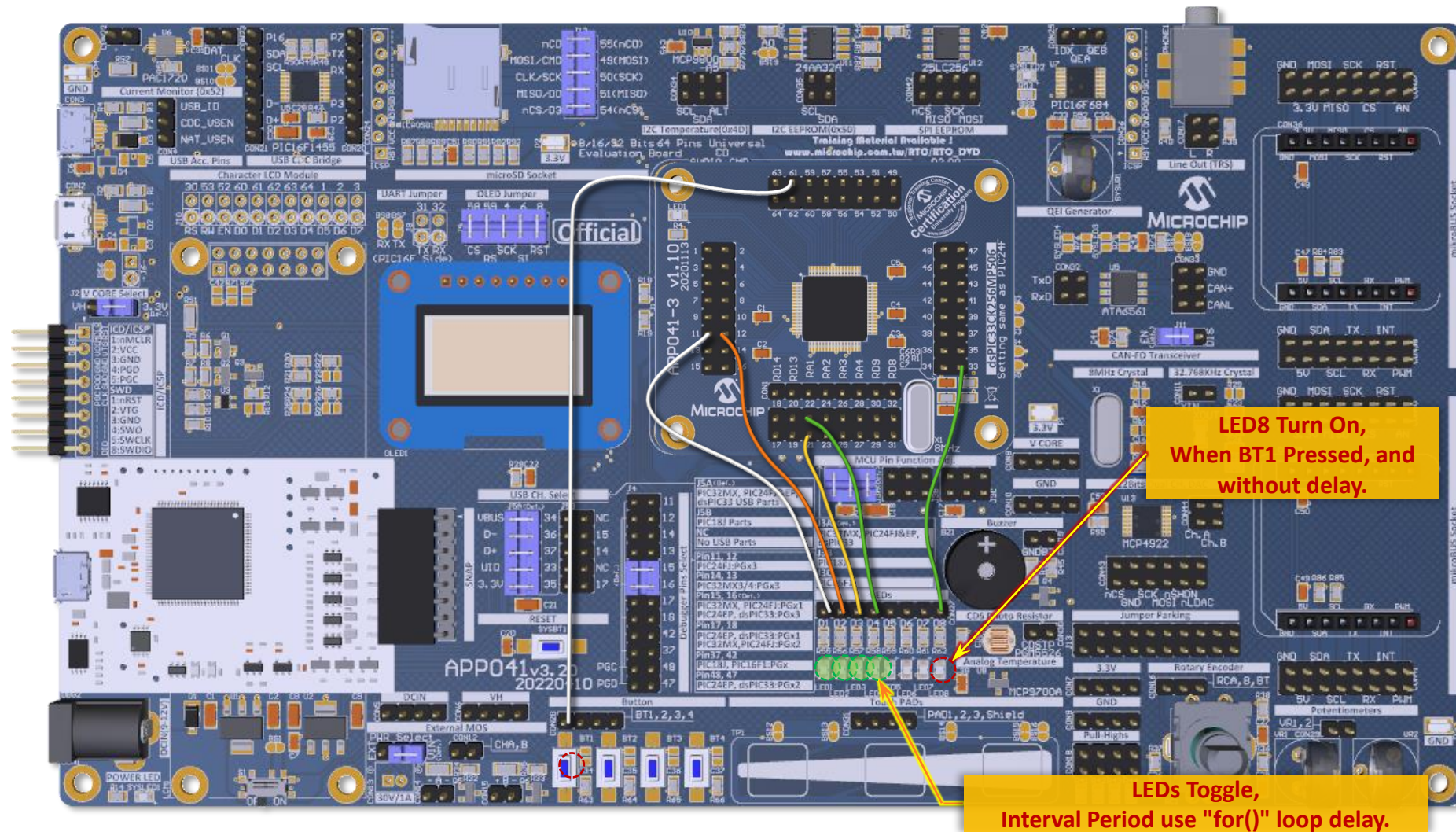
int main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        if( i++ > 80000 )
        {
            i = 0;
            LED1_Toggle();
            LED2_Toggle();
            LED3_Toggle();
            LED4_Toggle();
        }

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

Lab3 GPIO Input

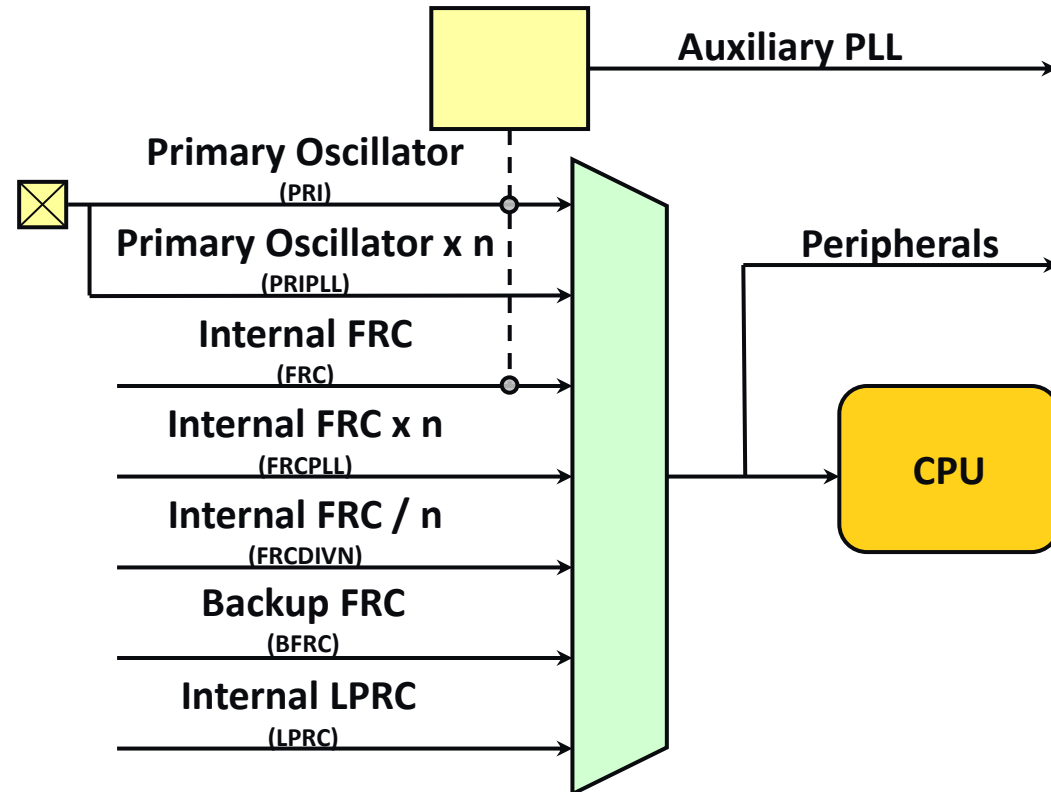
Result (State Machine)



Oscillator Architecture

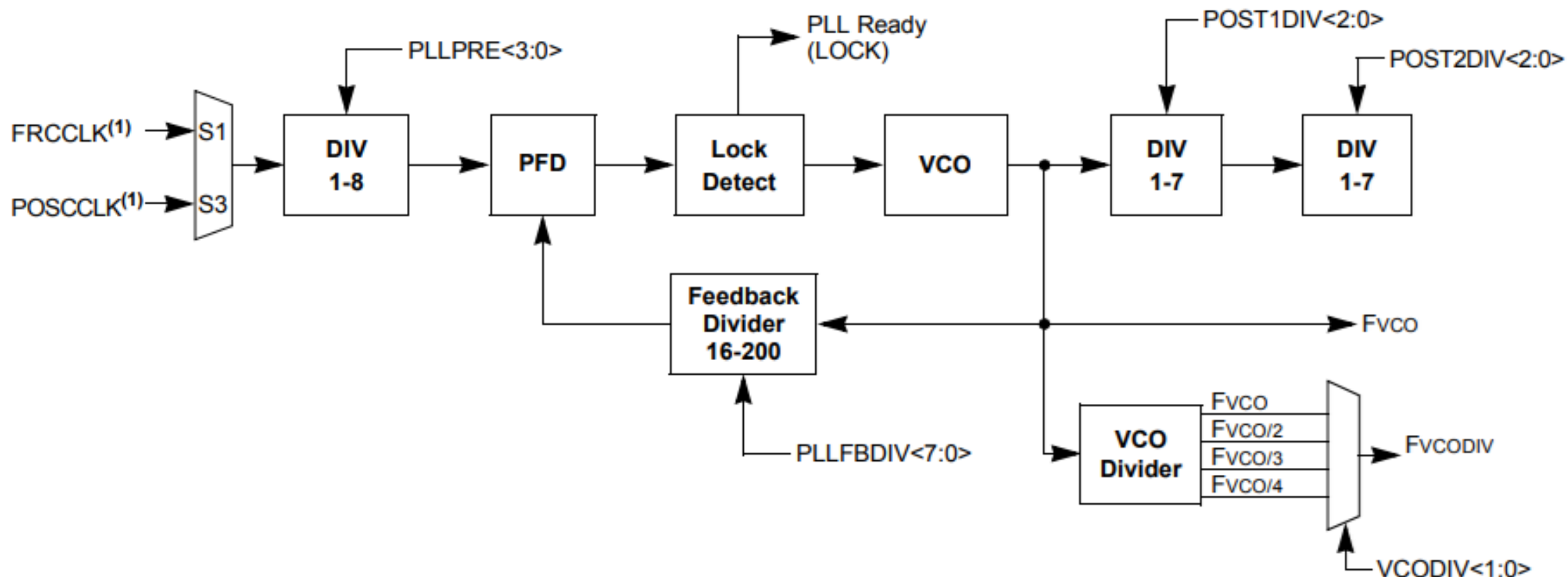
dsPIC33CK Series Clocks

- 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.



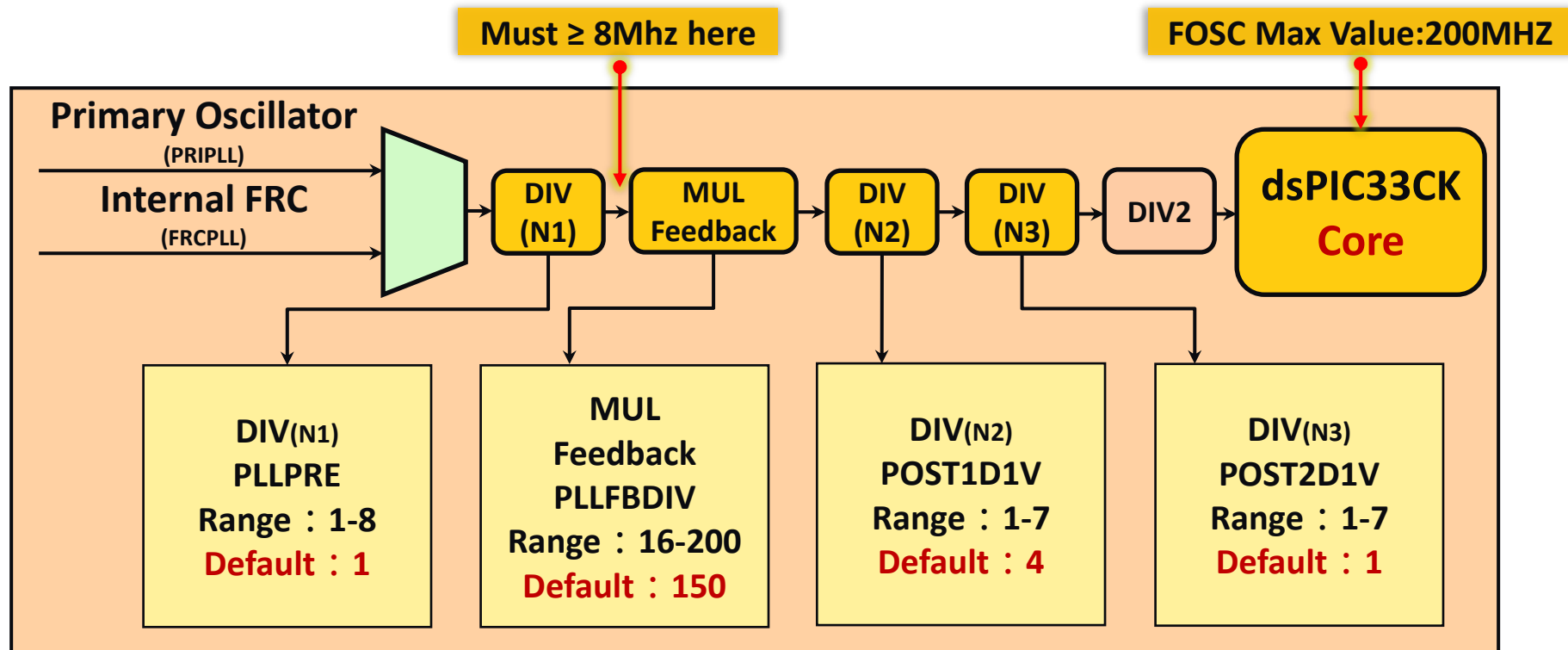
Phase Locked Loop

- A block diagram of the PLL module.



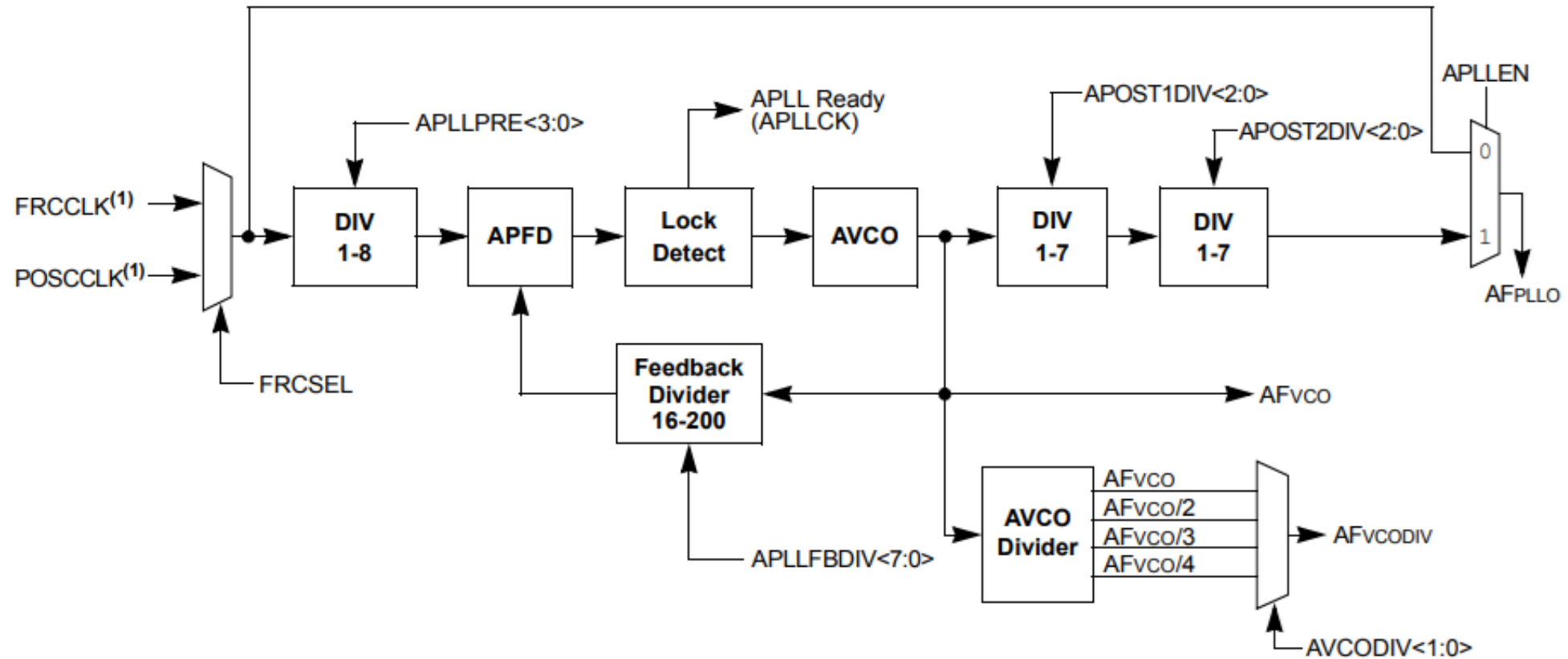
Phase Locked Loop

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



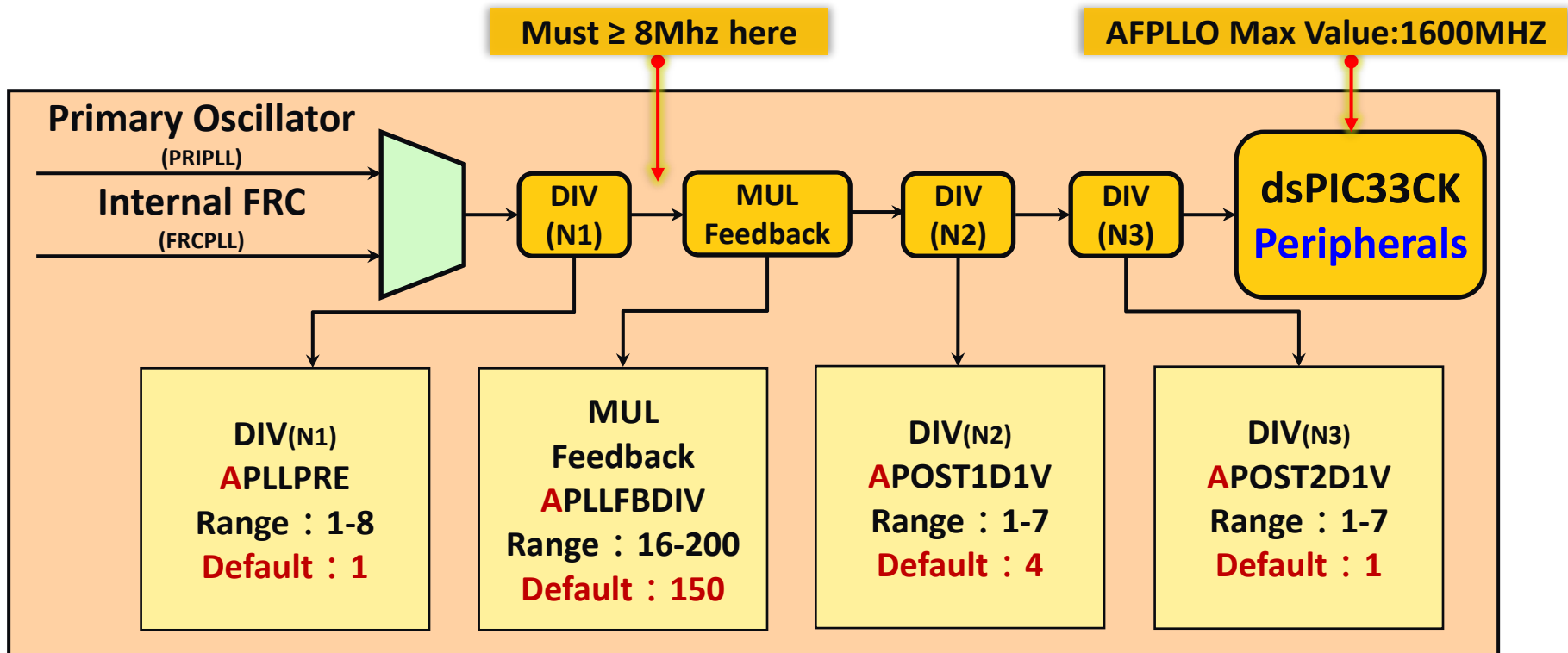
Phase Locked Loop

- A block diagram of the APLL module.

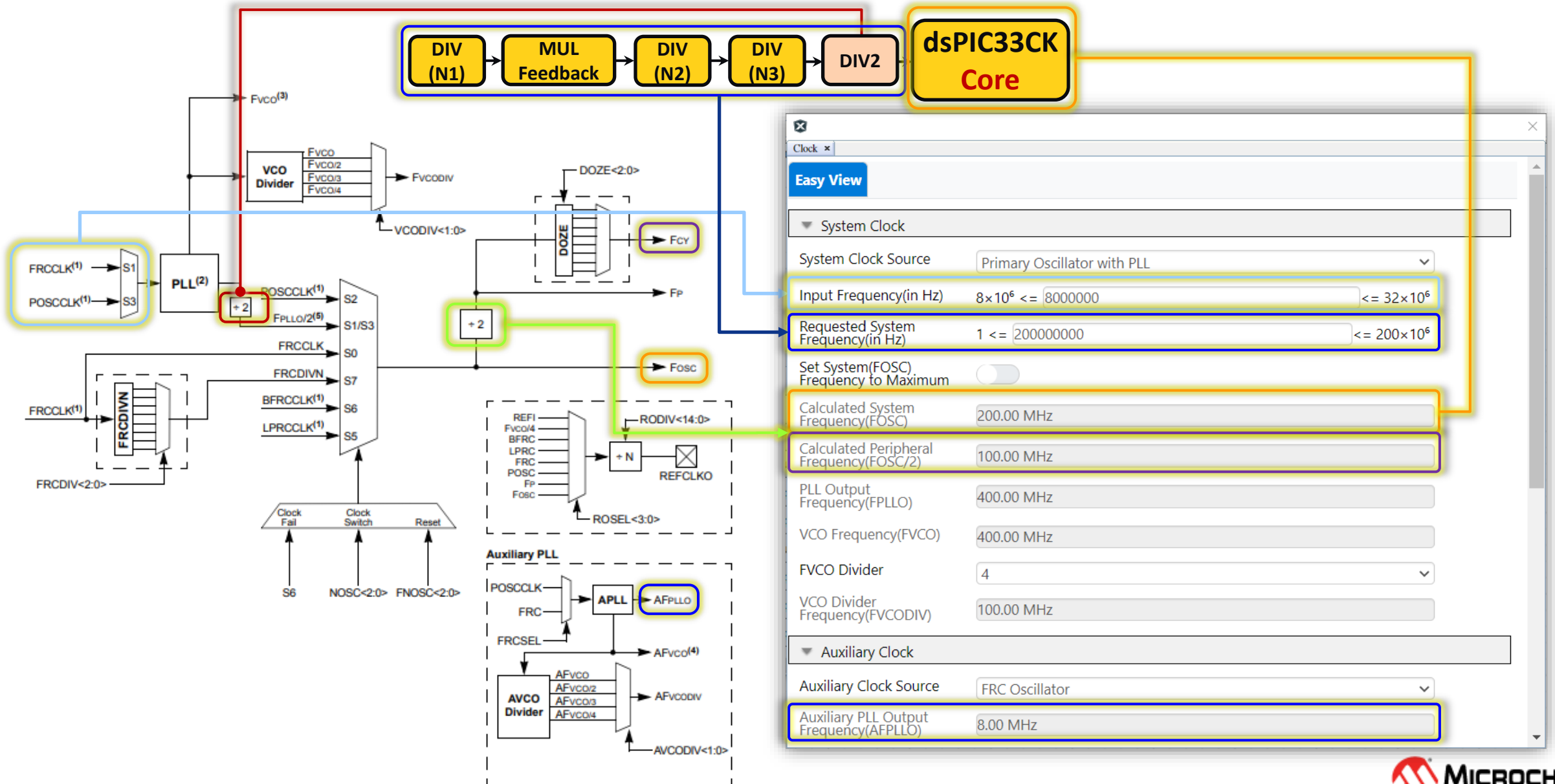


Auxiliary Phase Locked Loop

- 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 **(32MHz)** to MCU.

Let's go!

Lab4 Clock PRIPLL

Connection

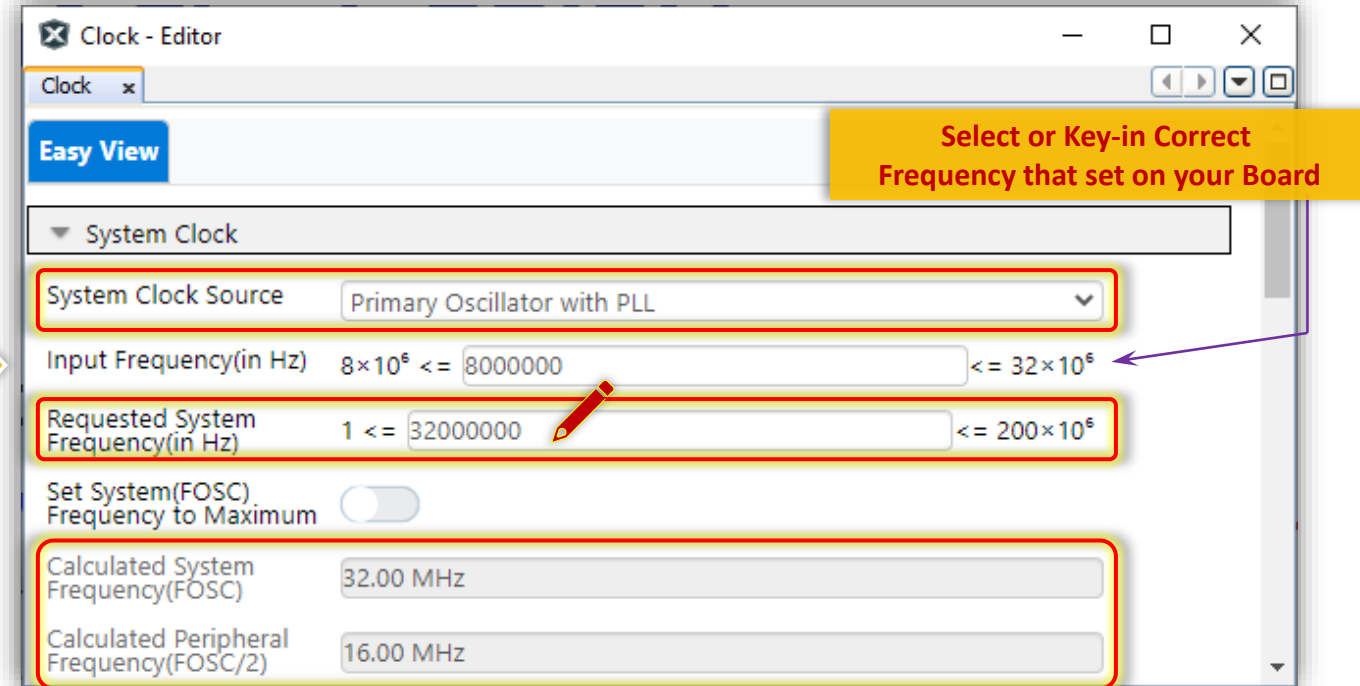
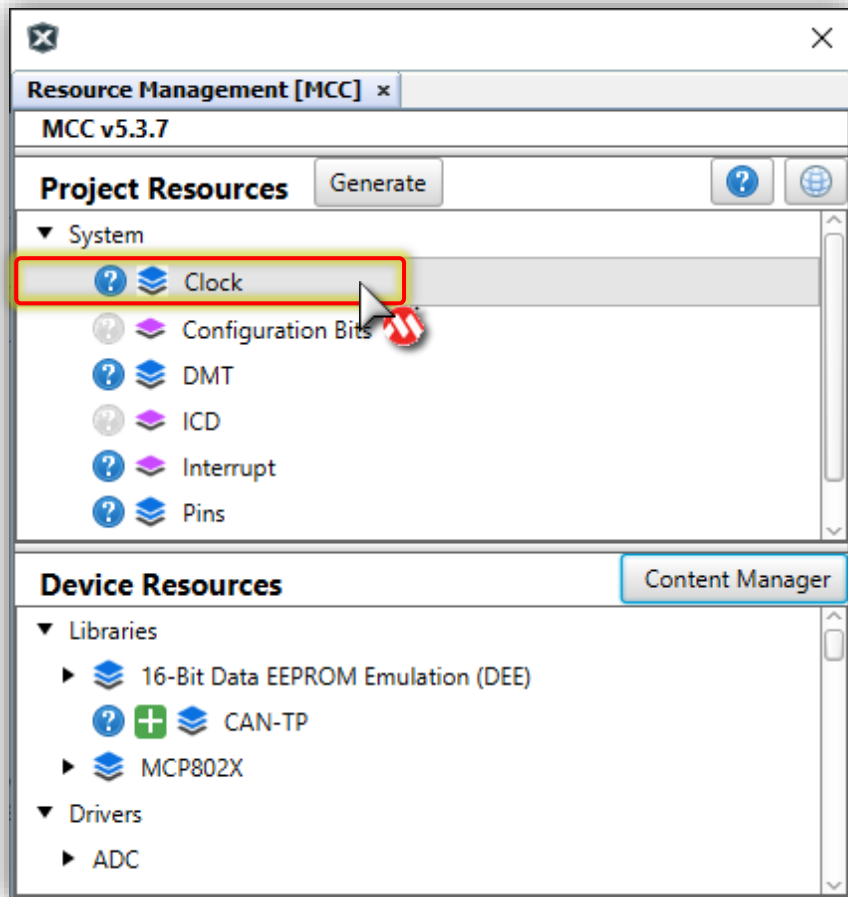


Lab4 Clock PRIPLL

Step 1

- Set System

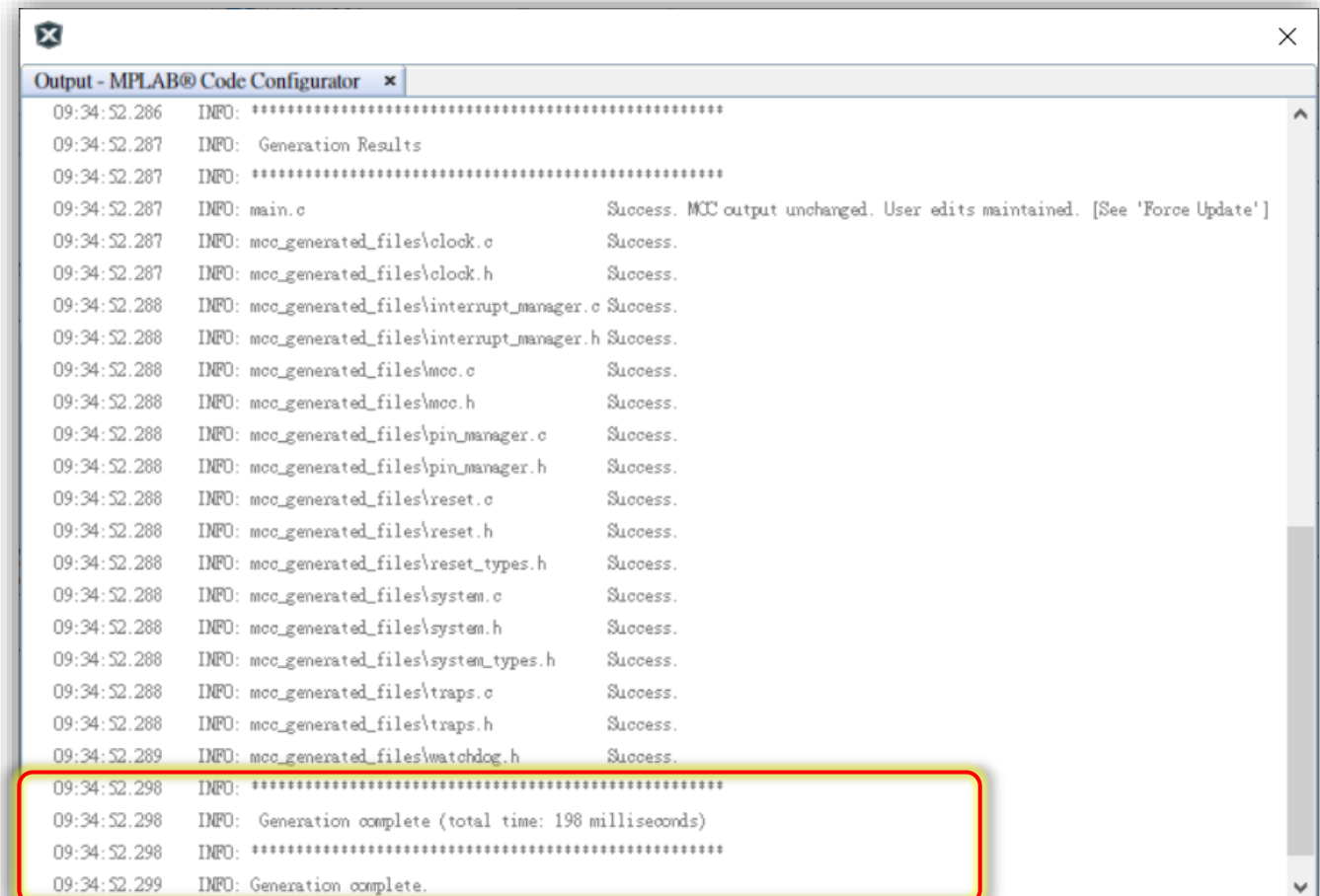
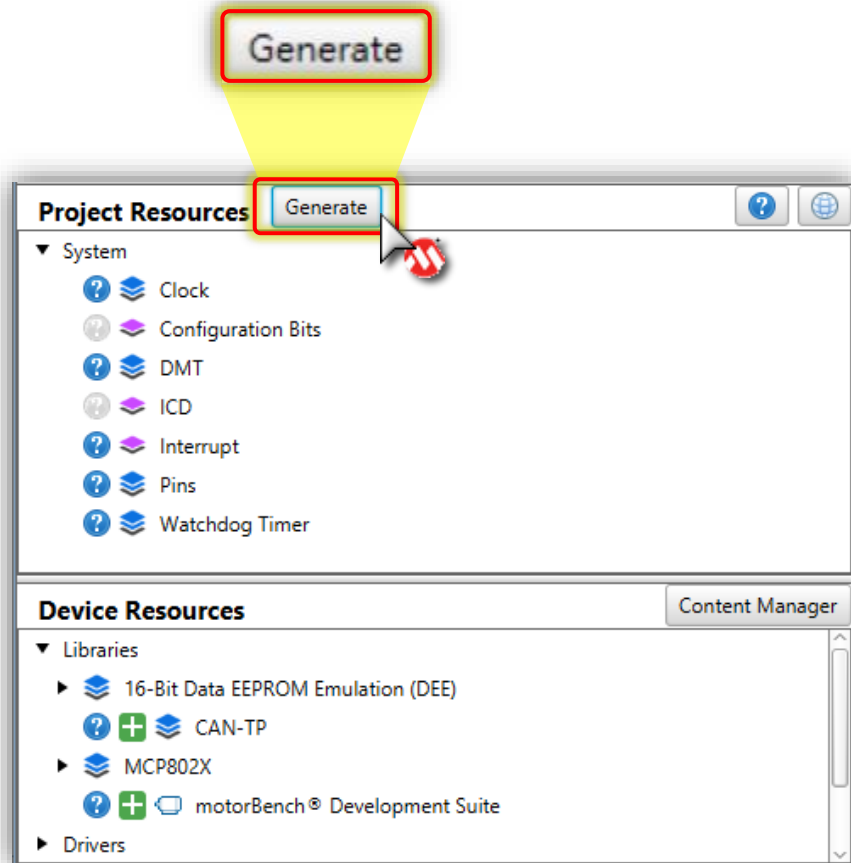
- System Clock: Primary Oscillator with PLL, Requested System Frequency(in Hz) → 32000000.



Lab4 Clock PRIPLL

Step 2

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



Result



Interrupt Architecture

What's Interrupt ?

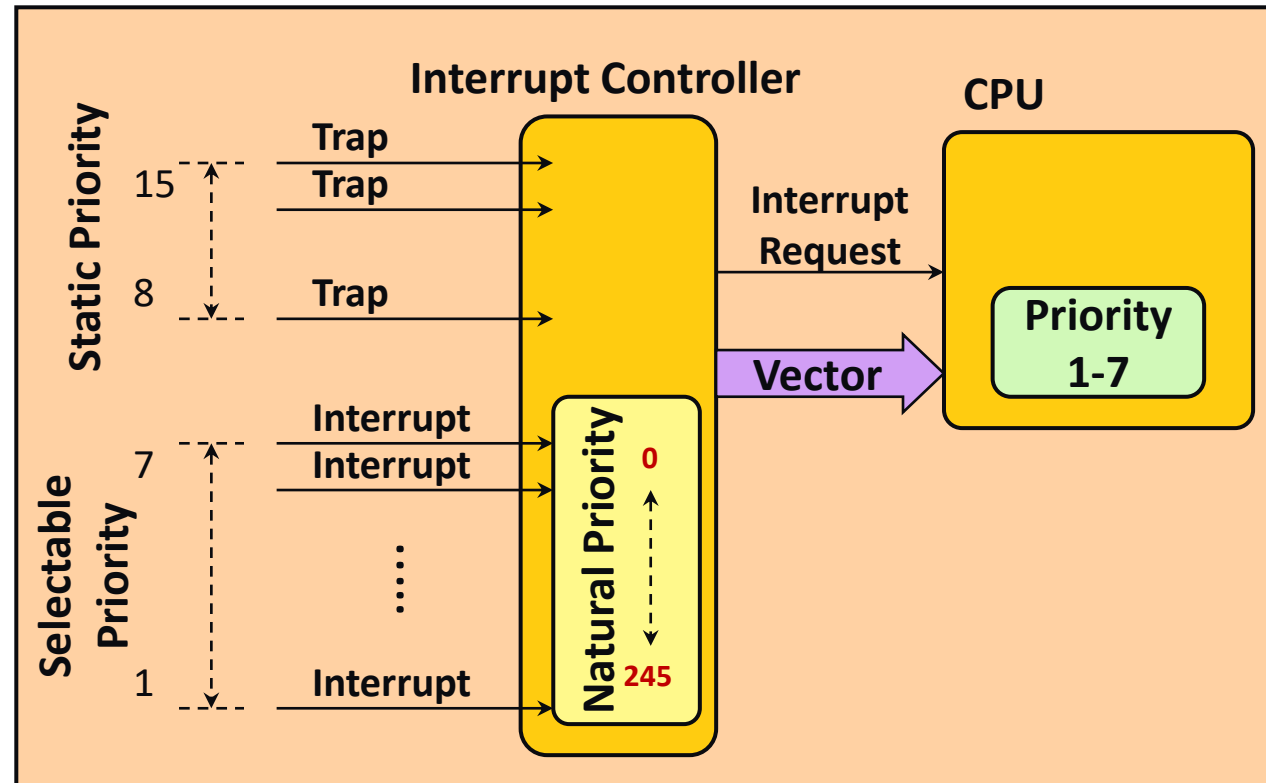
- In general, processor is executed in sequence according to the program's flow.
- However, if some peripherals or event needs immediate attention then an interrupt flag been set as high-priority and requiring interrupting current process to serve it.
- Processor will jump to execute the interrupt service 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\v2.10\docs\vector_docs\PIC33CK256MP506](#))

(lowest interrupt vector address).

Reset – GOTO Instruction	0x000000
Reset – GOTO Address	0x000002
Oscillator Fail Trap Vector	0x000004
Address Error Trap Vector	0x000006
Generic Hard Trap Vector	0x000008
Stack Error Trap Vector	0x00000A
Math Error Trap Vector	0x00000C
Reserved	0x00000E
Generic Soft Trap Vector	0x000010
Reserved	0x000012
Interrupt Vector 0	0x000014
Interrupt Vector 1	0x000016
:	:
:	:
:	:
:	:
Interrupt Vector 52	0x00007C
Interrupt Vector 53	0x00007E
Interrupt Vector 54	0x000080
:	:
:	:
:	:

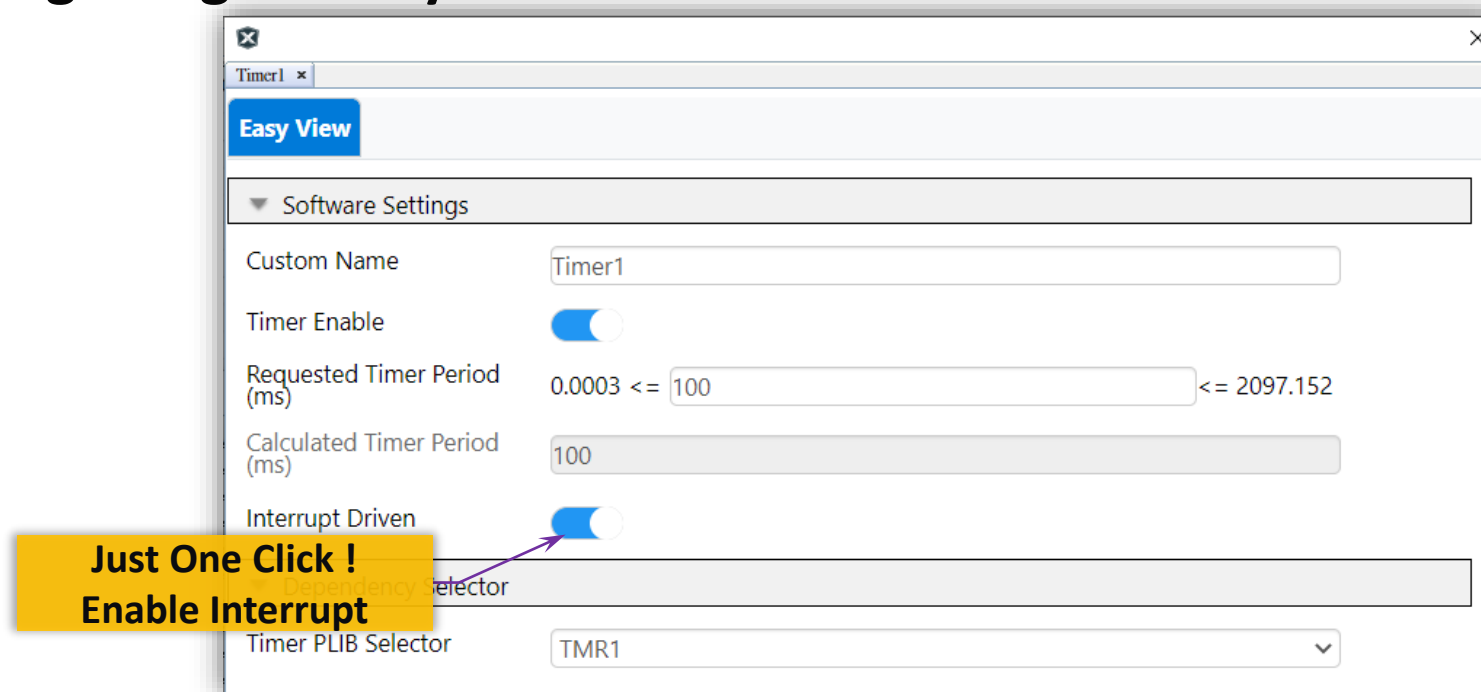
Interrupt Vector Details

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

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

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, just one click all interrupt relate setting will get ready.**



MCC Melody's Interrupt Function

- MCC Melody will **automatically** enable interrupts, set priority and create interrupt service routines, and provide "**Callback Function**" to 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.

Automatically initialize and enable interrupts.

```
void TMR1_Initialize (void)
{
    ...
    TMR1_TimeoutCallbackRegister(TMR1_TimeoutCallback);
    TMR1_Start();
}

void TMR1_Start( void )
{
    IFS0bits.T1IF = 0;           //Clear interrupt flag
    IEC0bits.T1IE = 1;          //Enable the interrupt
    T1CONbits.TON = 1;          // Start the Timer
}
```

Automatically create an Interrupt Service Routine (ISR).

```
void __attribute__ ((interrupt, no_auto_psv)) _T1Interrupt(void)
{
    (*TMR1_TimeoutHandler)(); //event handle
    IFS0bits.T1IF = 0;        //Clear interrupt flag
}
```

Automatically reset after an interrupt occurs.

```
void TMR1_TimeoutCallback( void )
{
    // Add your custom callback code here
}
```

User only need focus at application and add your custom code in "**Callback**" Function.

Callback Function

- MCC Melody provides two ways to use the callback function.
- MCC Melody will automatically generate a callback function with the "**weak**" attribute, user can redefine same name function in your code to replace the default function.
- Or you can use **xxxCallbackRegister()** to register (hook up) your custom callback function.
- We will **focus on using register custom callback function** in the following exercises.

tmr1.c

```
void __attribute__((weak)) TMR1_TimeoutCallback( void )  
{  
}  
}
```

main.c

```
void TMR1_TimeoutCallback( void )  
{  
    // Add your custom callback code here  
}  
  
int main(void)  
{  
    ...  
}
```

main.c

```
void myTimeoutCallback( void )  
{  
    // Add your custom callback code here  
}  
  
int main(void)  
{  
    TMR1_TimeoutCallbackRegister(&myTimeoutCallback);  
    ...  
}
```

Callback Hook Example

```
void myTimer1_Callback(void) {
```

```
// Add your custom callback code here
```

```
}
```

```
void TMRn_Initialize (void) {
```

```
...
```

```
TMR1_TimeoutCallbackRegister (myTimer1_Callback);
```

```
...
```

```
}
```

Hook (Register)

Callback

Application

System

```
void TMR1_TimeoutCallbackRegister(void (*handler)(void)) {
```

```
if(NULL != handler)
```

```
{
```

```
TMR1_TimeoutHandler = handler;
```

```
}
```

```
}
```

Function Pointer

Interrupt Service Routine (ISR)

```
void __attribute__ ( ( interrupt, no_auto_psv ) ) _TnInterrupt ( ) {
```

```
(*TMR1_TimeoutHandler)();
```

```
IFS0bits.T1IF = 0;
```

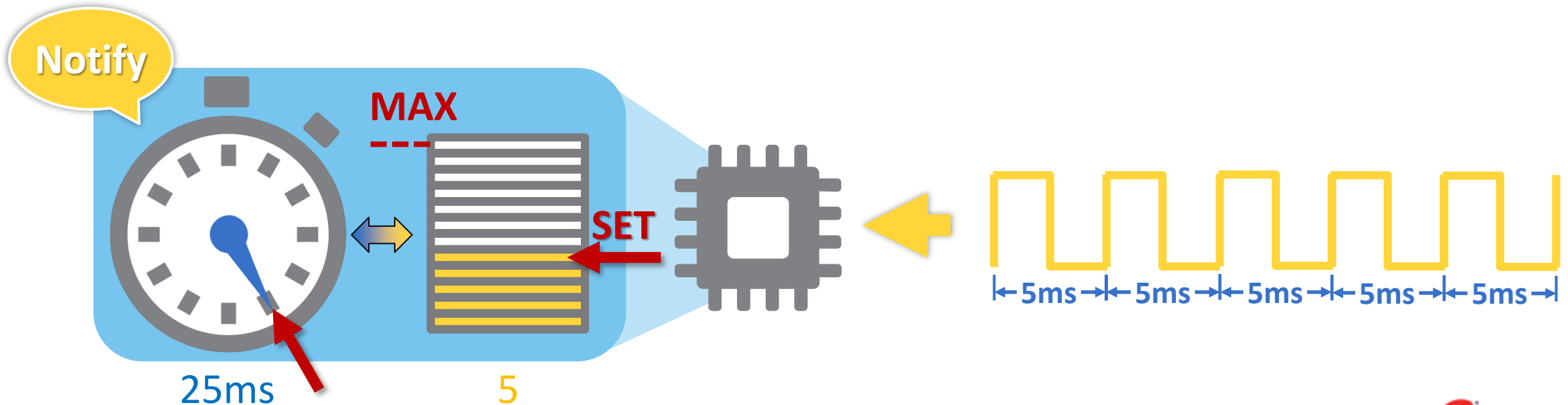
```
}
```

Interrupt

Timer Driver Architecture

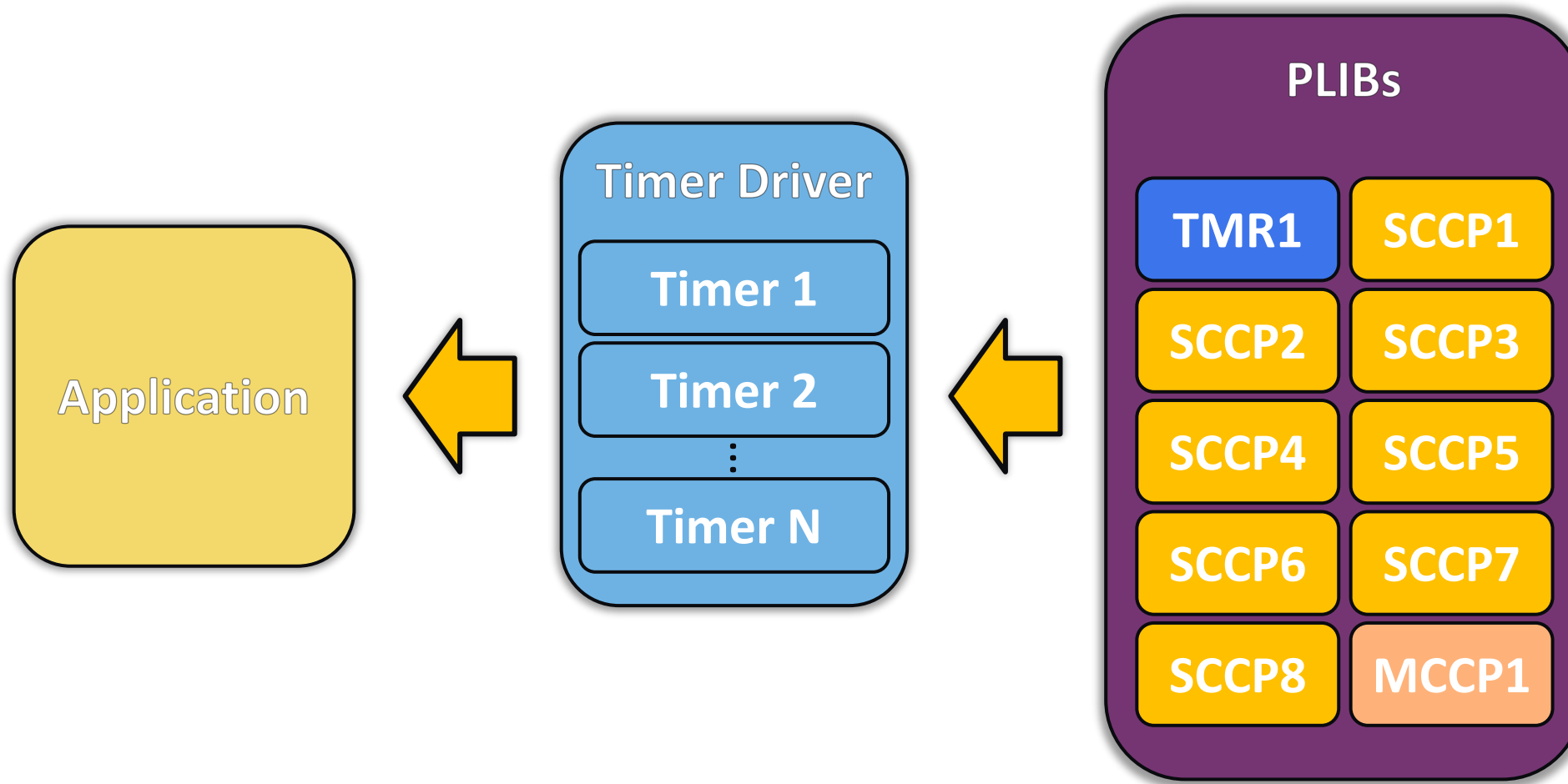
Timer or Counter ?

- Timer is a clock counter, use to count how many clocks into module after start, we can use the **known clock** period to calculate time.
- Timer can set a **notify** condition, like overflow, equal some value etc.
- Timer supports both **polling** and **interrupt** mode.
 - If interrupt disable : User must check flag, as soon as possible.
 - If interrupt enable : Timer will notify CPU, if condition is true.



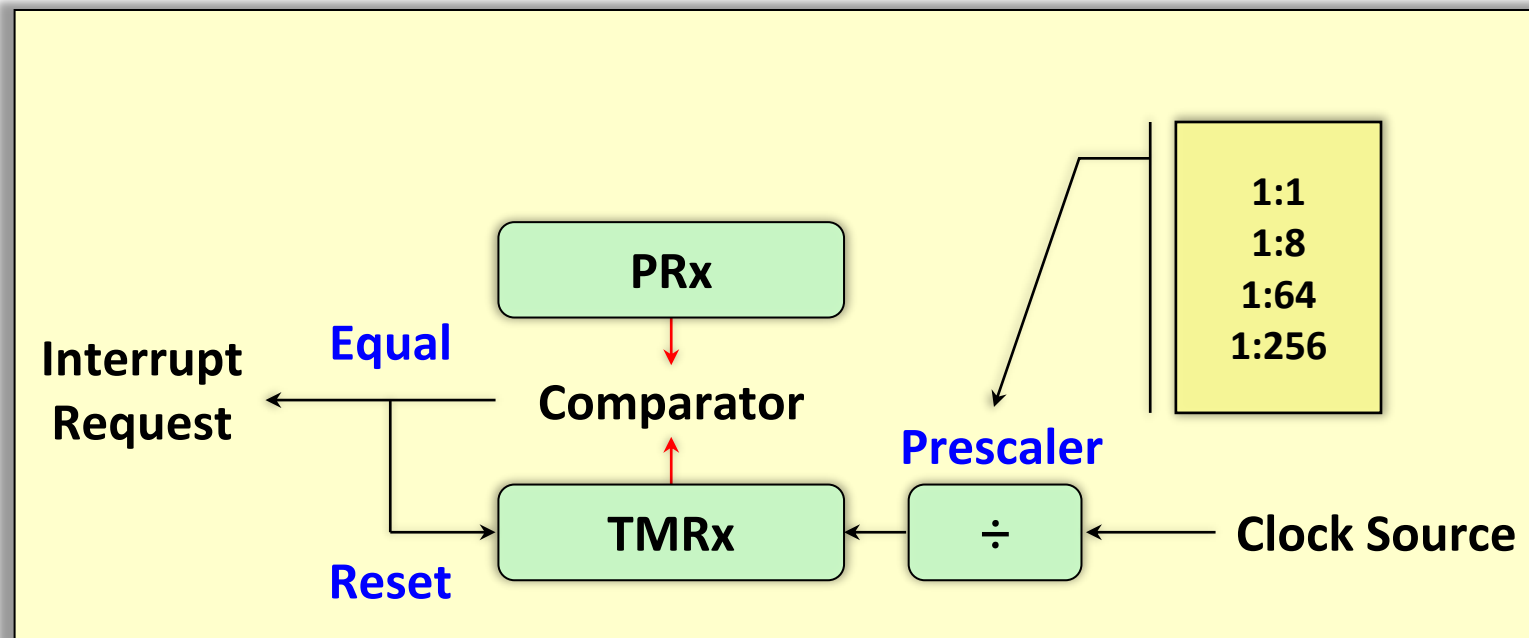
Timer Driver

- Driver provides simple, abstracted, portable interface to peripherals.
- dsPIC33CKMP506 provides **10 sets** of peripherals that can be used for **Timer Driver**, **1 "TMR"**, **8 "SCCP"** and **1 "MCCP"**, in this section we focus on **Timer** mode.



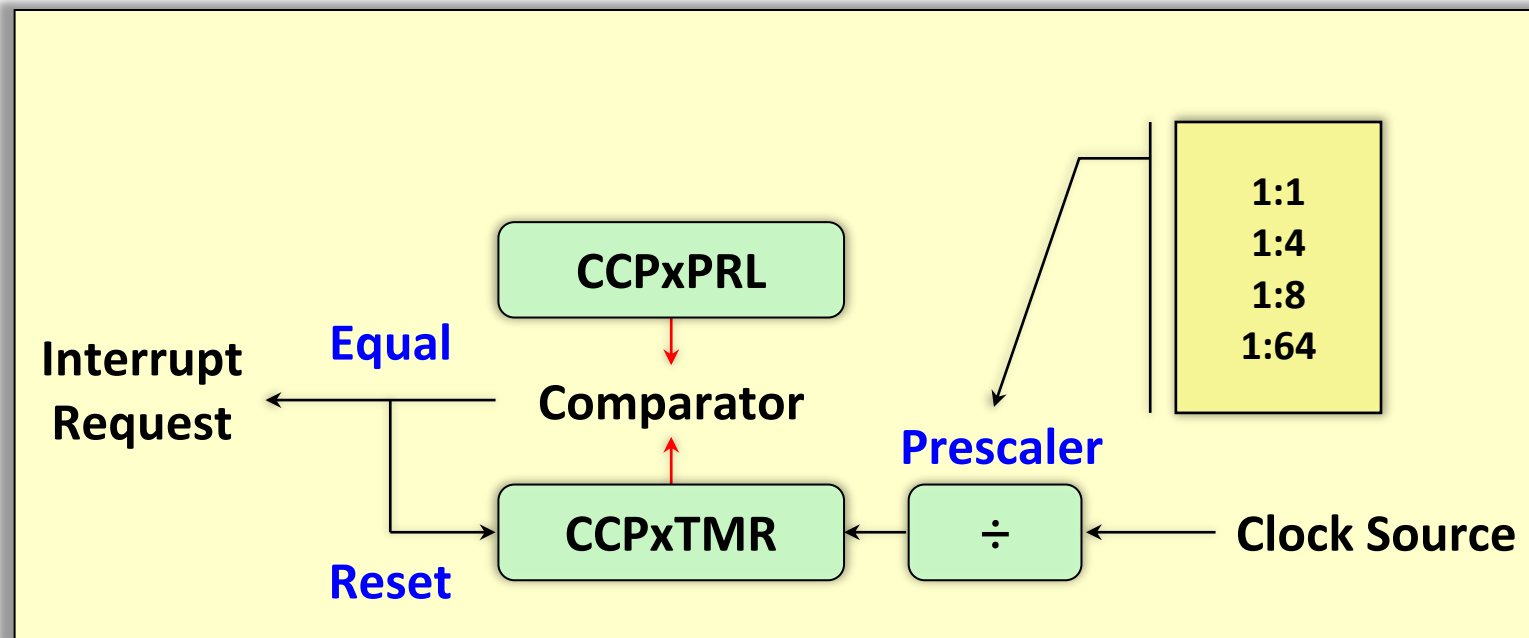
Timer PLIB - TMR

- 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.



Timer PLIB - SCCP/MCCP with Timer 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.



Functions of Melody Timer Driver

- Melody Provide below functions for Timer:

Prototype definition : "tmr1.h" and "sccpn.h"

- **TIMER_INTERFACE**

- (void) *CustomName*.Initialize(void);
 - (void) *CustomName*.Deinitialize(void);
 - (void) *CustomName*.Start(void);
 - (void) *CustomName*.Stop(void);
 - (void) *CustomName*.PeriodSet(uint32_t count);
 - (uint32_t) *CustomName*.PeriodGet(void);
 - (uint32_t) *CustomName*.CounterGet (void);
 - (void) *CustomName*.InterruptPrioritySet(enum INTERRUPT_PRIORITY priority);
 - (void) *CustomName*.TimeoutCallbackRegister(void (*handler)(void));
- Default *CustomName* is *Timern*.

Lab5 Timer1 Interrupt

- Base on current project or Open .\Exercises\Lab5_xxx.X to do exercise.
- Try to add **Timer Driver** using **TMR1** and rename it as "**myTimer1**".
- Try to replace software delay to **myTimer1** to control LEDs to change state every **500ms**.
- **myTimer1** interrupt priority set to level **1**.

Let's go!

Lab5 Timer1 Interrupt

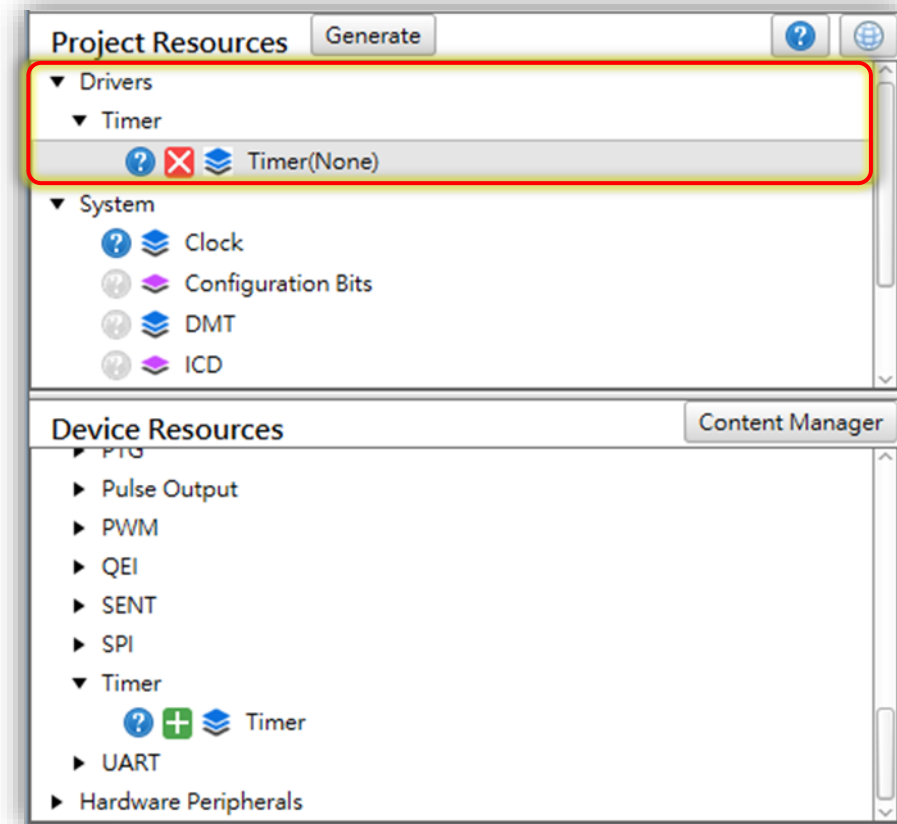
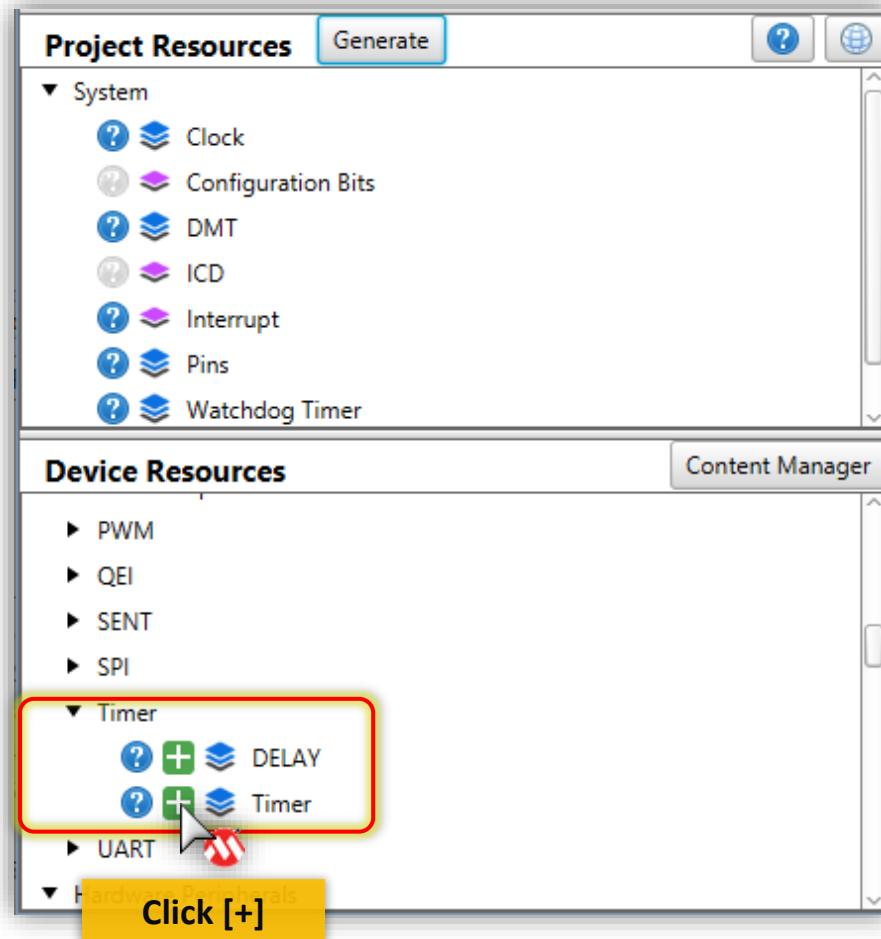
Connection



Lab5 Timer1 Interrupt

Step 1

- Click [+] to add **Timer Driver**.



Lab5 Timer1 Interrupt

Step 2

- Select **Timer PLIB** to **TMR1**.

Easy View

▼ Software Settings

Custom Name

Timer1

Timer Enable

☒

Requested Timer Period (ms)

0.0003 <= 1 <= 2097.152

Calculated Timer Period (ms)

1

Interrupt Driven

☒

▼ Dependency Selector

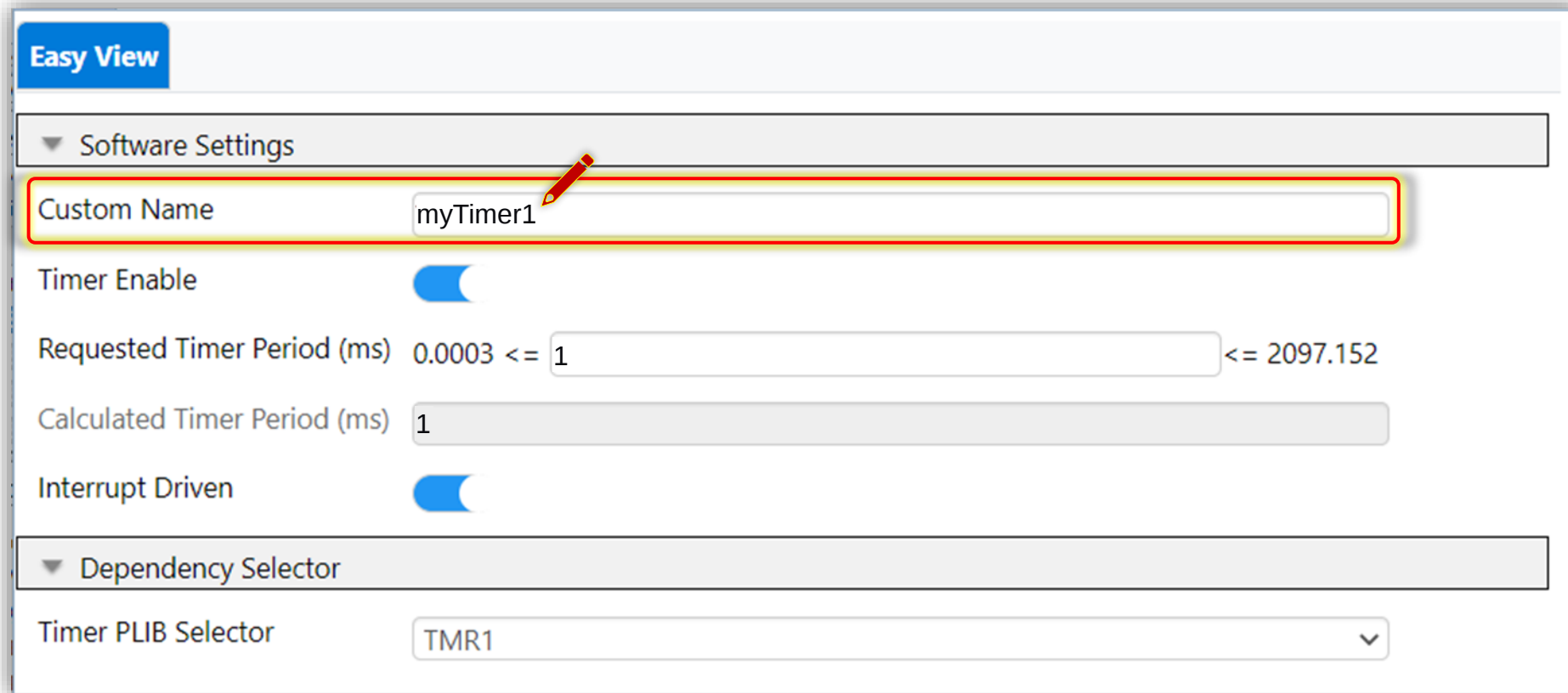
Timer PLIB Selector

TMR1

Lab5 Timer1 Interrupt

Step 3

- Modify the Custom Name to "myTimer1".



Easy View

▼ Software Settings

Custom Name myTimer1

Timer Enable ☒

Requested Timer Period (ms) 0.0003 <= 1 <= 2097.152

Calculated Timer Period (ms) 1

Interrupt Driven ☒

▼ Dependency Selector

Timer PLIB Selector TMR1

Lab5 Timer1 Interrupt

Step 4

- Set **Requested Timer Period (ms)** : **500**.

Easy View

▼ Software Settings

Custom Name

myTimer1

Timer Enable

☒

Requested Timer Period (ms)

0.0003 <= 500 <= 2097.152

Calculated Timer Period (ms)

500

Interrupt Driven

☒

▼ Dependency Selector

Timer PLIB Selector

TMR1

Lab5 Timer1 Interrupt

Step 5

- Check **Timer Interrupt** is **Driven**.

Easy View

▼ Software Settings

Custom Name

myTimer1

Timer Enable

☒

Requested Timer Period (ms)

0.0003 <= 500 <= 2097.152

Calculated Timer Period (ms)

500

Interrupt Driven

☒

▼ Dependency Selector

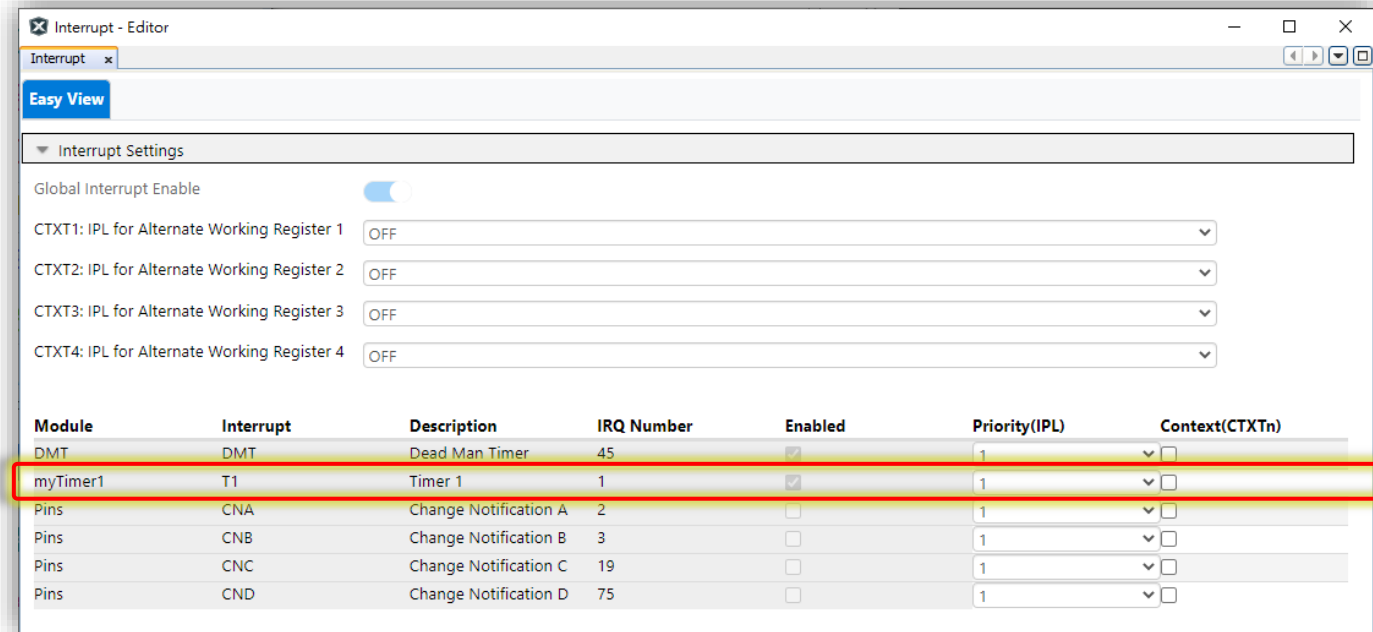
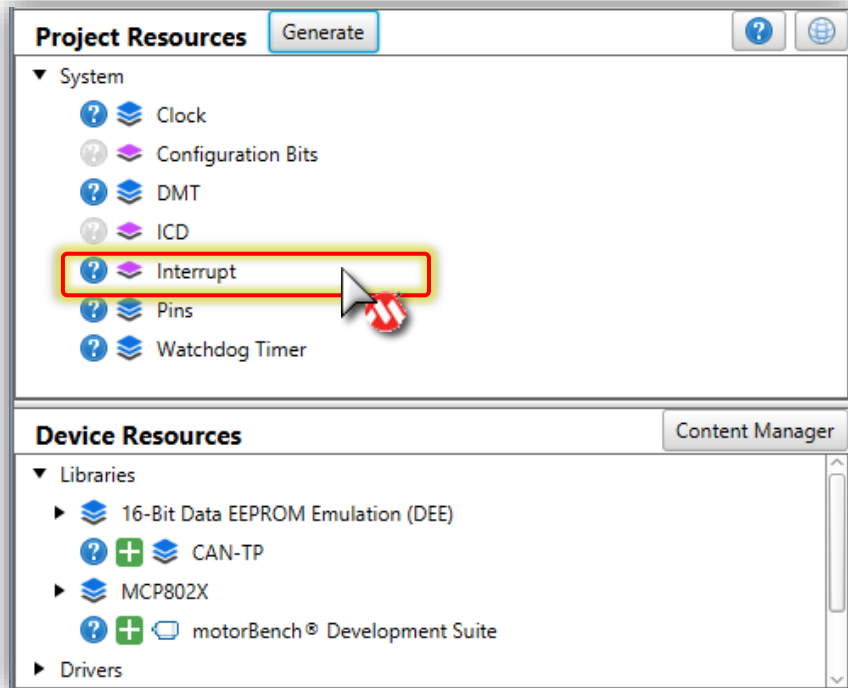
Timer PLIB Selector

TMR1

Lab5 Timer1 Interrupt

Step 6

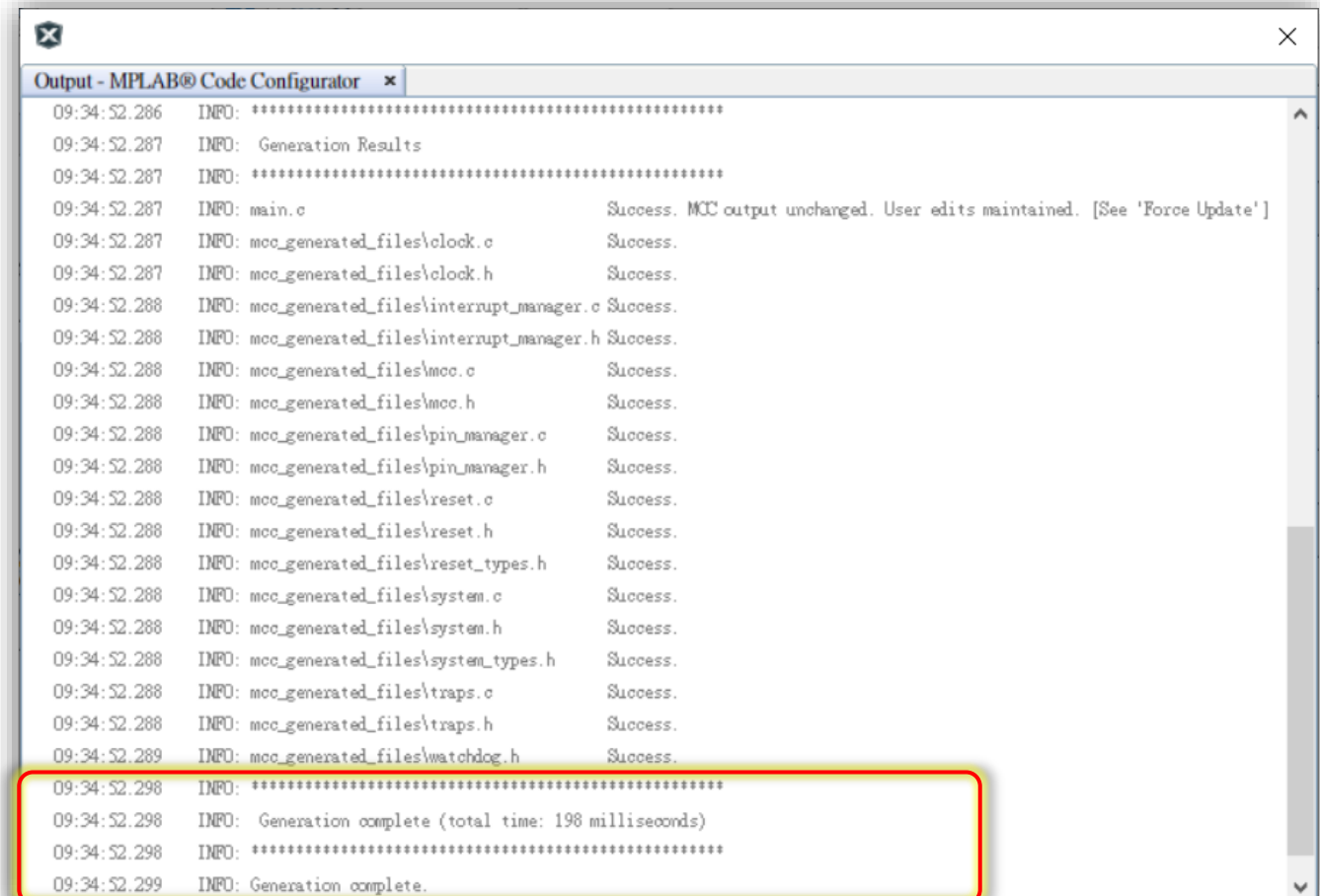
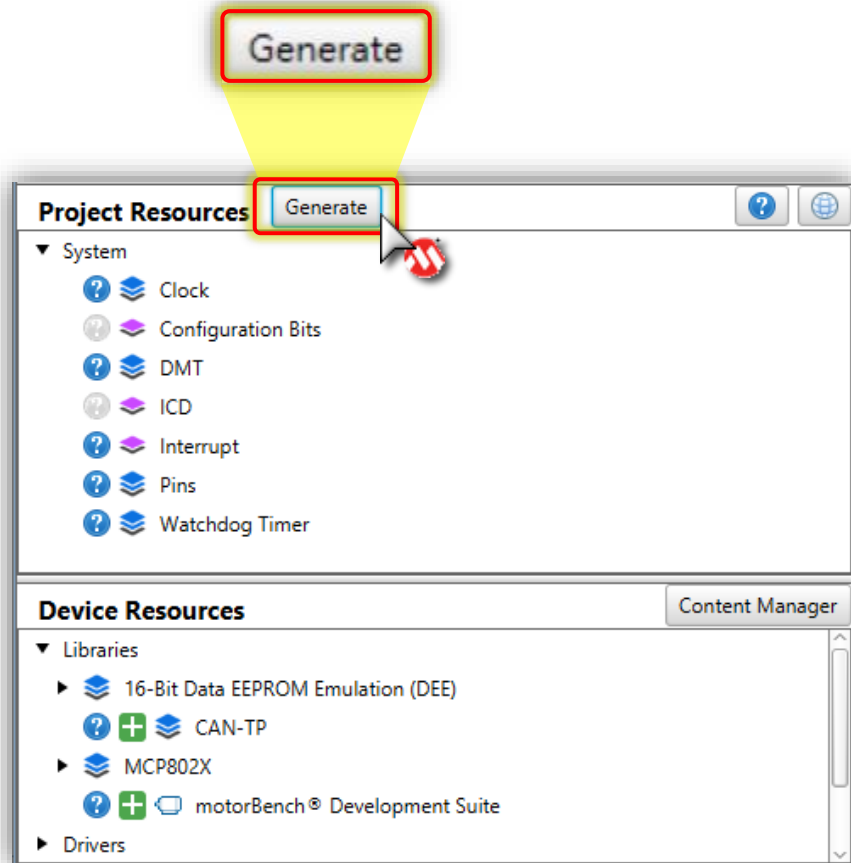
- Check **Interrupt status (Timer1)** for **myTimer1**.



Lab5 Timer1 Interrupt

Step 7

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



Lab5 Timer1 Interrupt

Code Example

```
#include "mcc_generated_files/system/system.h"
#include "mcc_generated_files/system/pins.h"
#include "mcc_generated_files/timer/tmr1.h"

uint32_t i = 0;

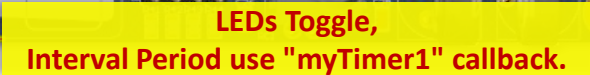
void myTimer1_Callback(void)
{
    LED1_Toggle();
    LED2_Toggle();
    LED3_Toggle();
    LED4_Toggle();
}

int main(void)
{
    SYSTEM_Initialize();

    myTimer1.TimeoutCallbackRegister(myTimer1_Callback);

    while (1)
    {
        if(BT1_GetValue())
            LED8_SetLow();
        else
            LED8_SetHigh();
    }
}
```

Result



Lab6 SCCP Timer Interrupt

- Base on current project or Open .\Exercises\Lab6_xxx.X to do exercise.
- Try to add **Timer Driver** using **SCCP1** and rename it as "**myTimer2**".
- Try to replace software delay to **myTimer2** to control LEDs to change state every **1,000ms (1s)**.
- **myTmier2** interrupt priority set to level **2**.
- To separate LED1 & LED2, LED3 & LED4 to blink different period.
 - **LED1** & **LED2** blink by **myTimer1 (500ms)**
LED3 & **LED4** blink by **myTimer2 (1,000ms)**

Let's go!

Lab6 SCCP Timer Interrupt

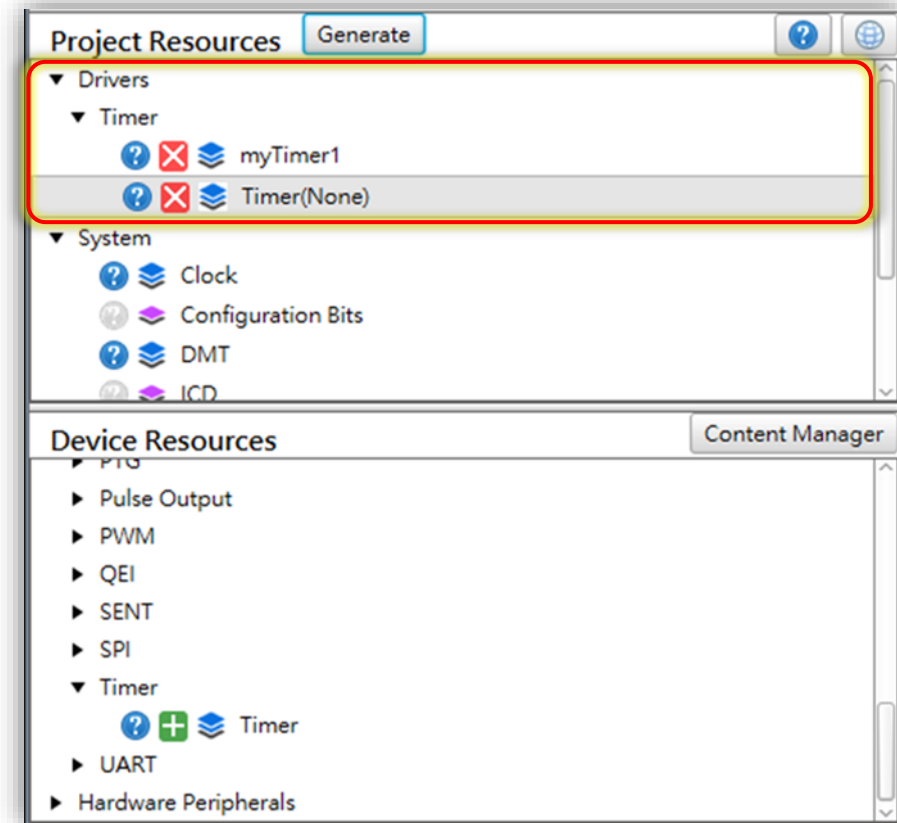
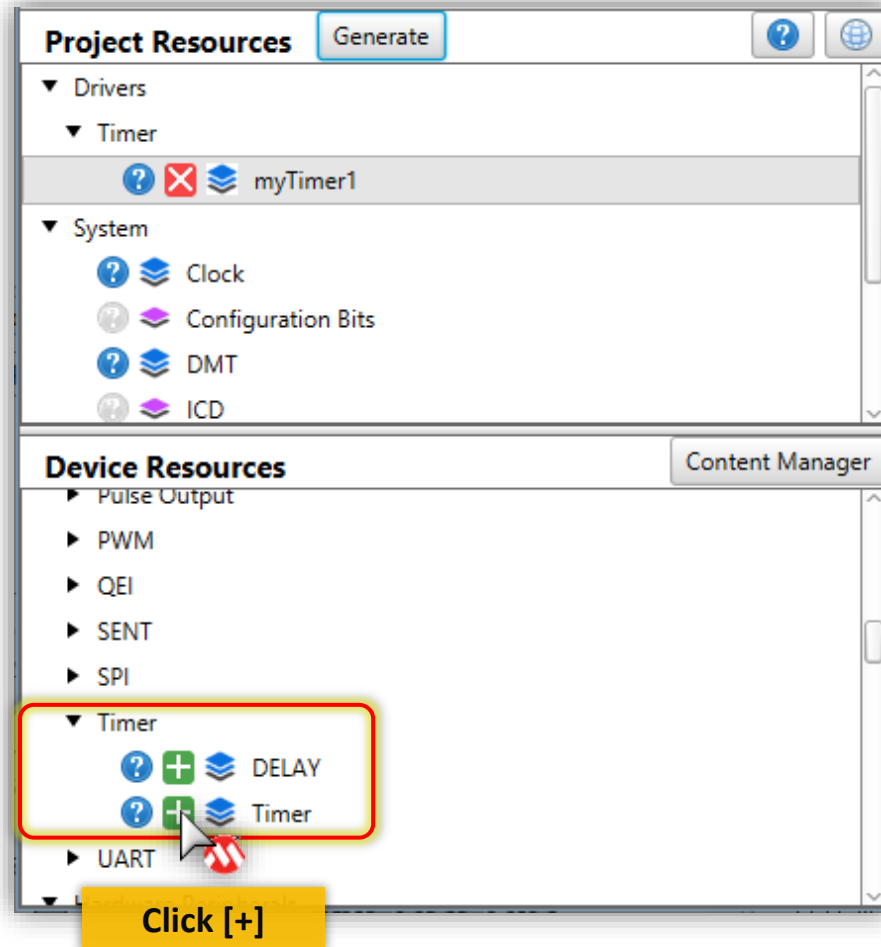
Connection



Lab6 SCCP Timer Interrupt

Step 1

- Click [+] to add **Timer Driver**.



Lab6 SCCP Timer Interrupt

Step 2

- Select **Timer PLIB** to **SCCP1**.

Easy View

▼ Software Settings

Custom Name

Timer1

Timer Enable

☒

Requested Timer Period (ms)

0.0003 <= 1 <= 34.359738368×10⁶

Calculated Timer Period (ms)

1

Interrupt Driven

☒

▼ Dependency Selector

Timer PLIB Selector

SCCP1

Lab6 SCCP Timer Interrupt

Step 3

- Modify the Custom Name to "myTimer2".

Easy View

▼ Software Settings

Custom Name

myTimer2

Timer Enable

☒

Requested Timer Period (ms)

0.0003 <= 1 <= 34.359738368×10⁶

Calculated Timer Period (ms)

1

Interrupt Driven

☒

▼ Dependency Selector

Timer PLIB Selector

SCCP1

Lab6 SCCP Timer Interrupt

Step 4

- Set **Requested Timer Period (ms)** : **1000**.

Easy View

▼ Software Settings

Custom Name

myTimer2

Timer Enable

☒

Requested Timer Period (ms)

0.0003 <= 1000 <= 34.359738368×10⁶

Calculated Timer Period (ms)

1000

Interrupt Driven

☒

▼ Dependency Selector

Timer PLIB Selector

SCCP1

Lab6 SCCP Timer Interrupt

Step 5

- Check **Timer Interrupt** is **Driven**.

Easy View

▼ Software Settings

Custom Name

myTimer2

Timer Enable

☒

Requested Timer Period (ms)

0.0003 <= 1000 <= 34.359738368×10⁶

Calculated Timer Period (ms)

1000

Interrupt Driven

☒

▼ Dependency Selector

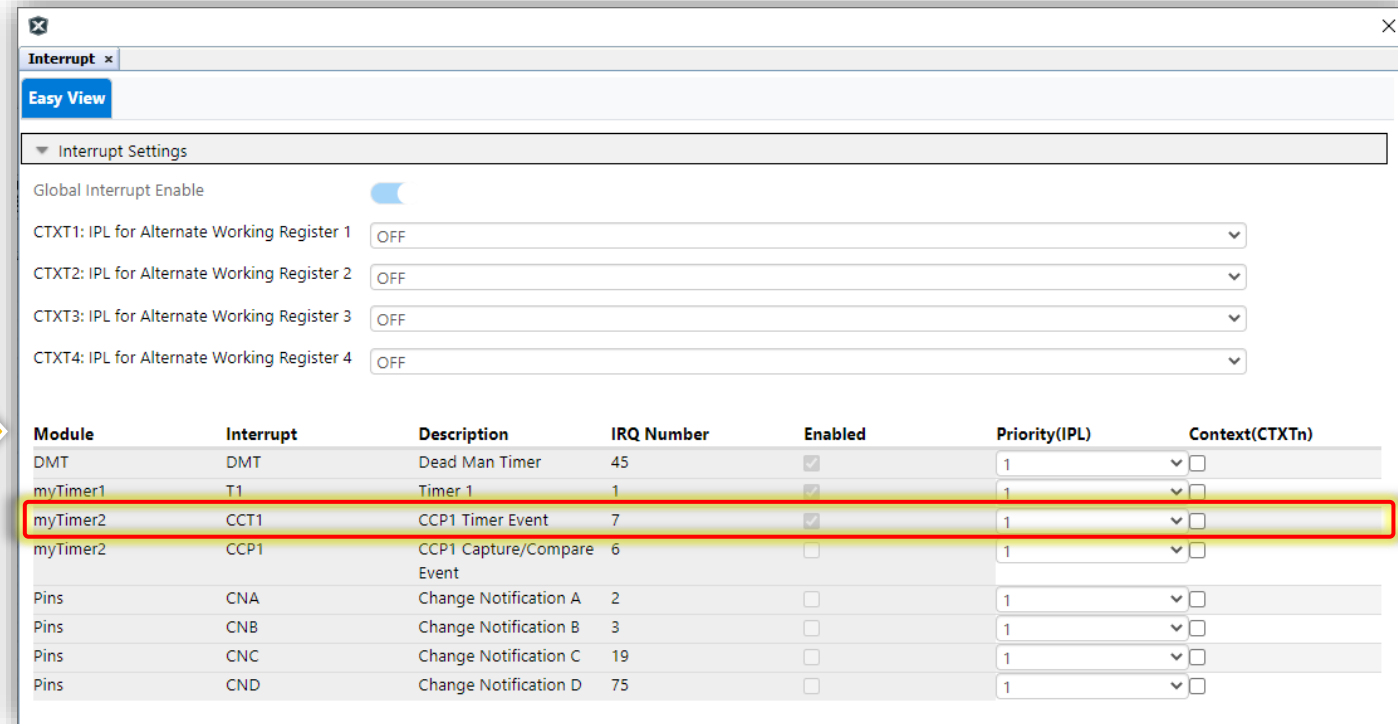
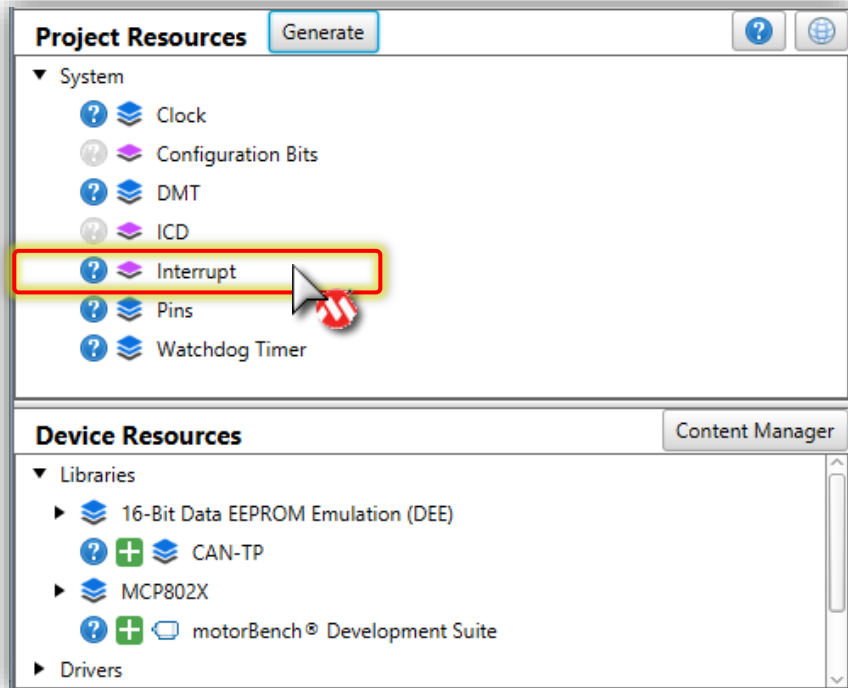
Timer PLIB Selector

SCCP1 ▼

Lab6 SCCP Timer Interrupt

Step 6

- Check **Interrupt status (CCP1 Timer Event)** for **myTimer2**.



Lab6 SCCP Timer Interrupt

Step 7

- Modify Timer2 Module's **Interrupt Priority** for **CCT1 Timer Event** to **2**.

Interrupt x

Easy View

Interrupt Settings

Global Interrupt Enable ☒

CTXT1: IPL for Alternate Working Register 1 OFF

CTXT2: IPL for Alternate Working Register 2 OFF

CTXT3: IPL for Alternate Working Register 3 OFF

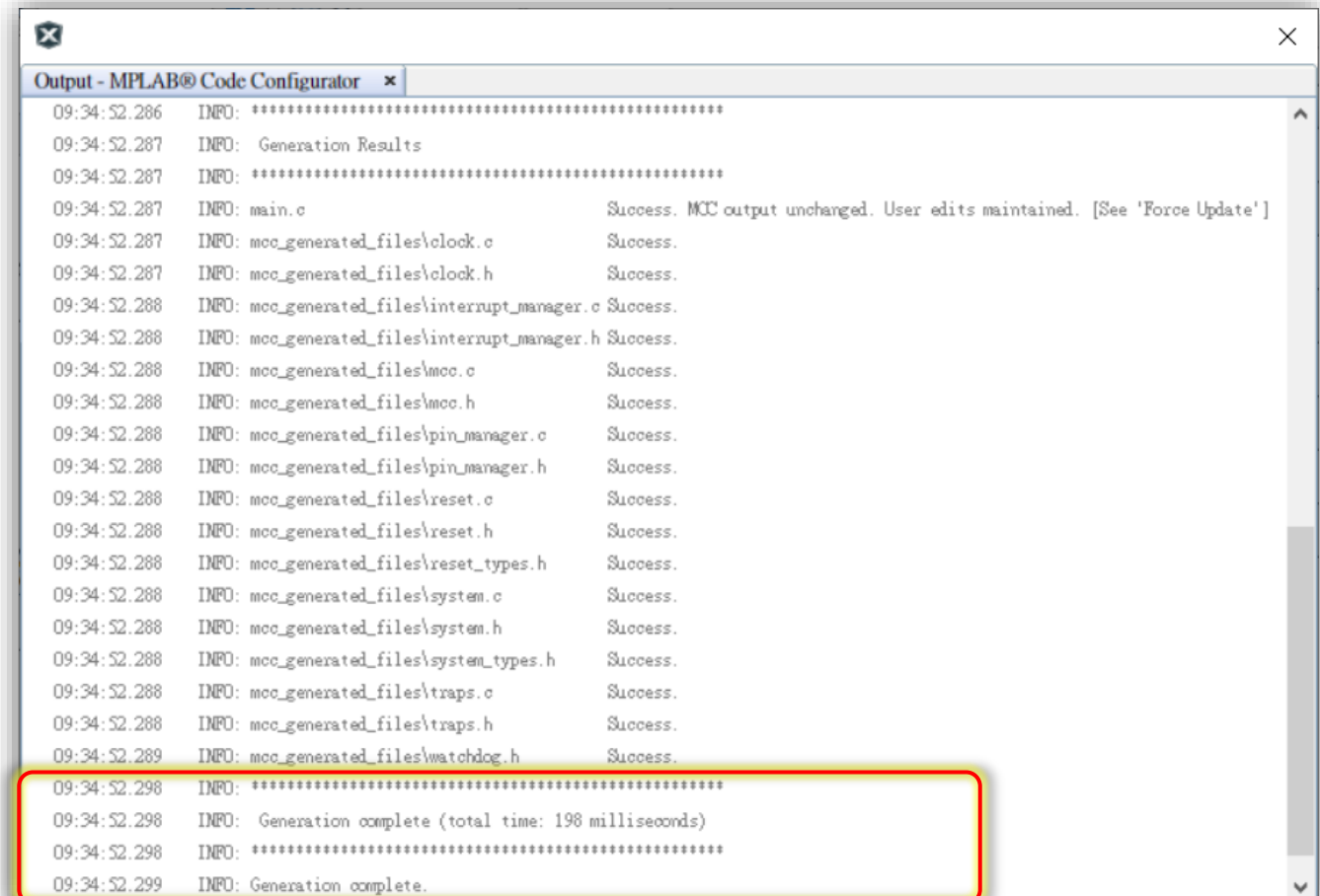
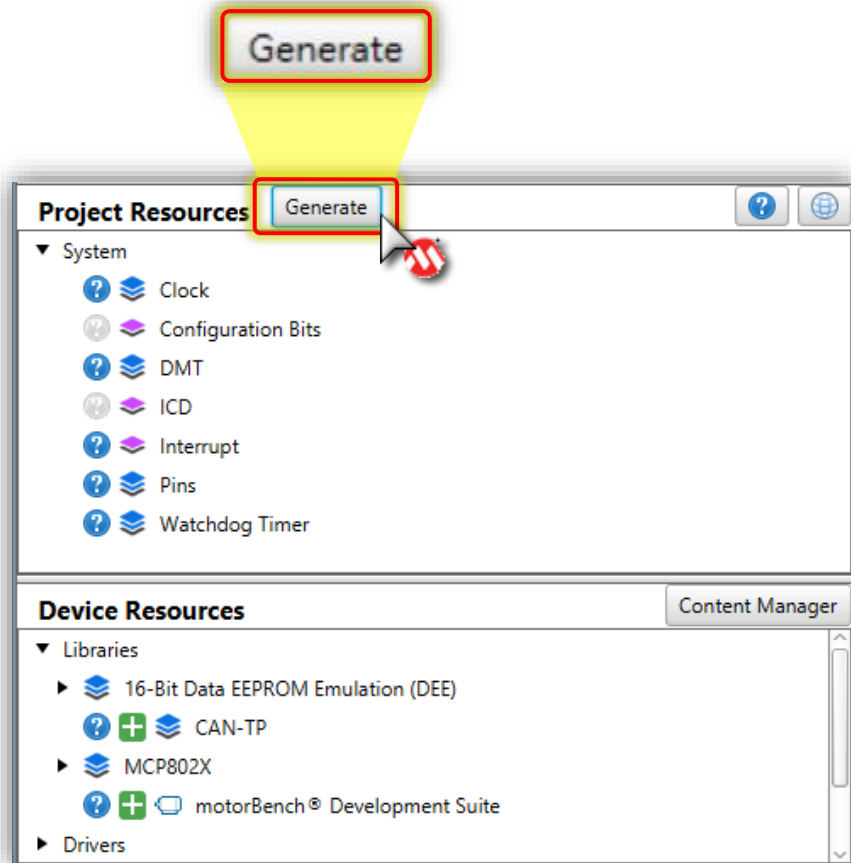
CTXT4: IPL for Alternate Working Register 4 OFF

Module	Interrupt	Description	IRQ Number	Enabled	Priority(IPL)	Context(CTXTn)
DMT	DMT	Dead Man Timer	45	☒	1	☐
myTimer1	T1	Timer 1	1	☒	1	☐
myTimer2	CCT1	CCP1 Timer Event	7	☒	2	☐
myTimer2	CCP1	CCP1 Capture/Compare Event	6	☐	1	☐
Pins	CNA	Change Notification A	2	☐	1	☐
Pins	CNB	Change Notification B	3	☐	1	☐
Pins	CNC	Change Notification C	19	☐	1	☐
Pins	CND	Change Notification D	75	☐	1	☐

Lab6 SCCP Timer Interrupt

Step 8

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



Lab6 SCCP Timer Interrupt

Code Example

```
...
#include "mcc_generated_files/timer/tmr1.h"
#include "mcc_generated_files/timer/sccp1.h"

uint32_t i = 0;

void myTimer1_Callback(void)
{
    LED1_Toggle();
    LED2_Toggle();
}

void myTimer2_Callback(void)
{
    LED3_Toggle();
    LED4_Toggle();
}

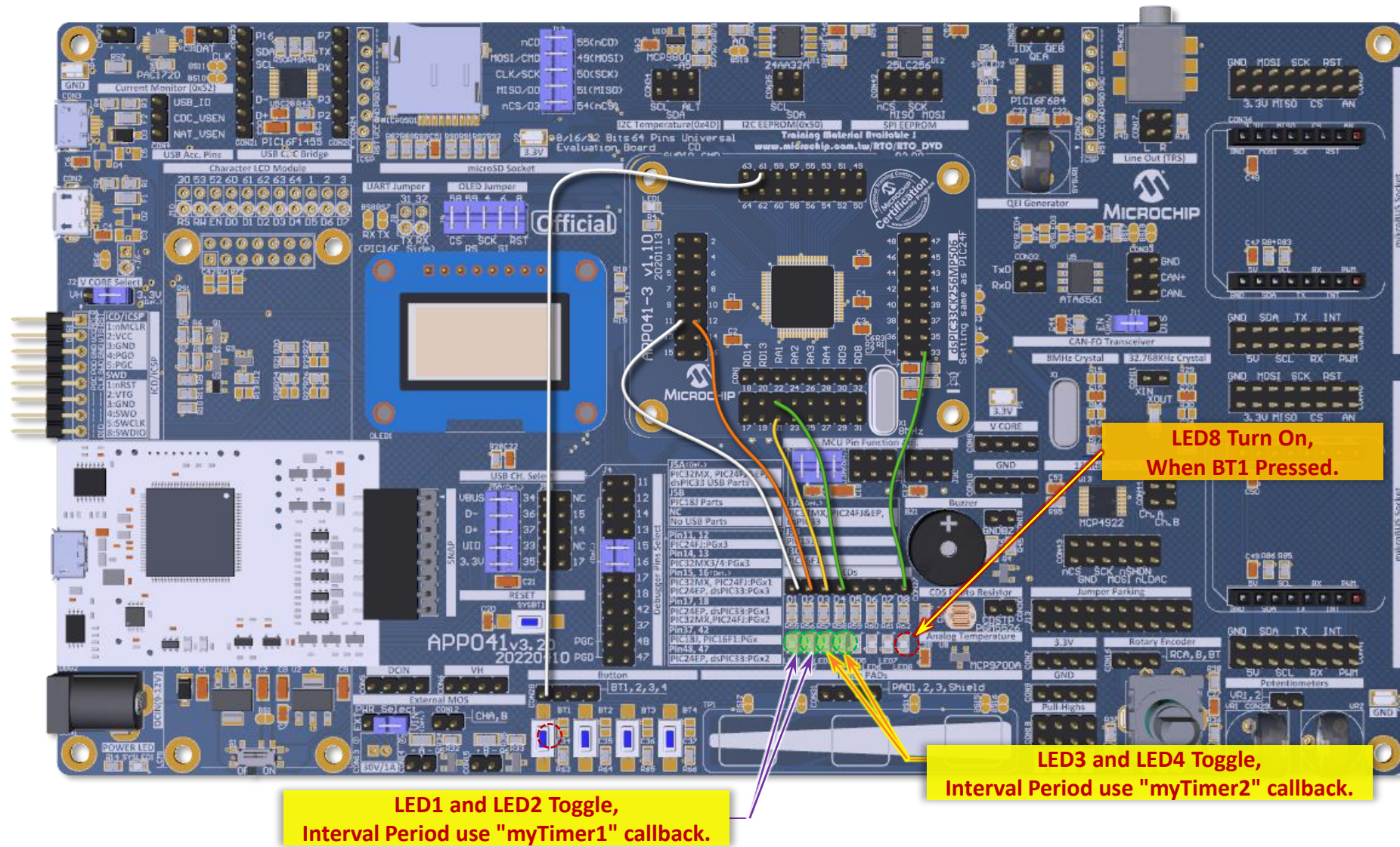
int main(void)
{
    SYSTEM_Initialize();

    myTimer1.TimeoutCallbackRegister(myTimer1_Callback);
    myTimer2.TimeoutCallbackRegister(myTimer2_Callback);

    while (1)
    {
        ...
    }
}
```

Lab6 SCCP Timer Interrupt

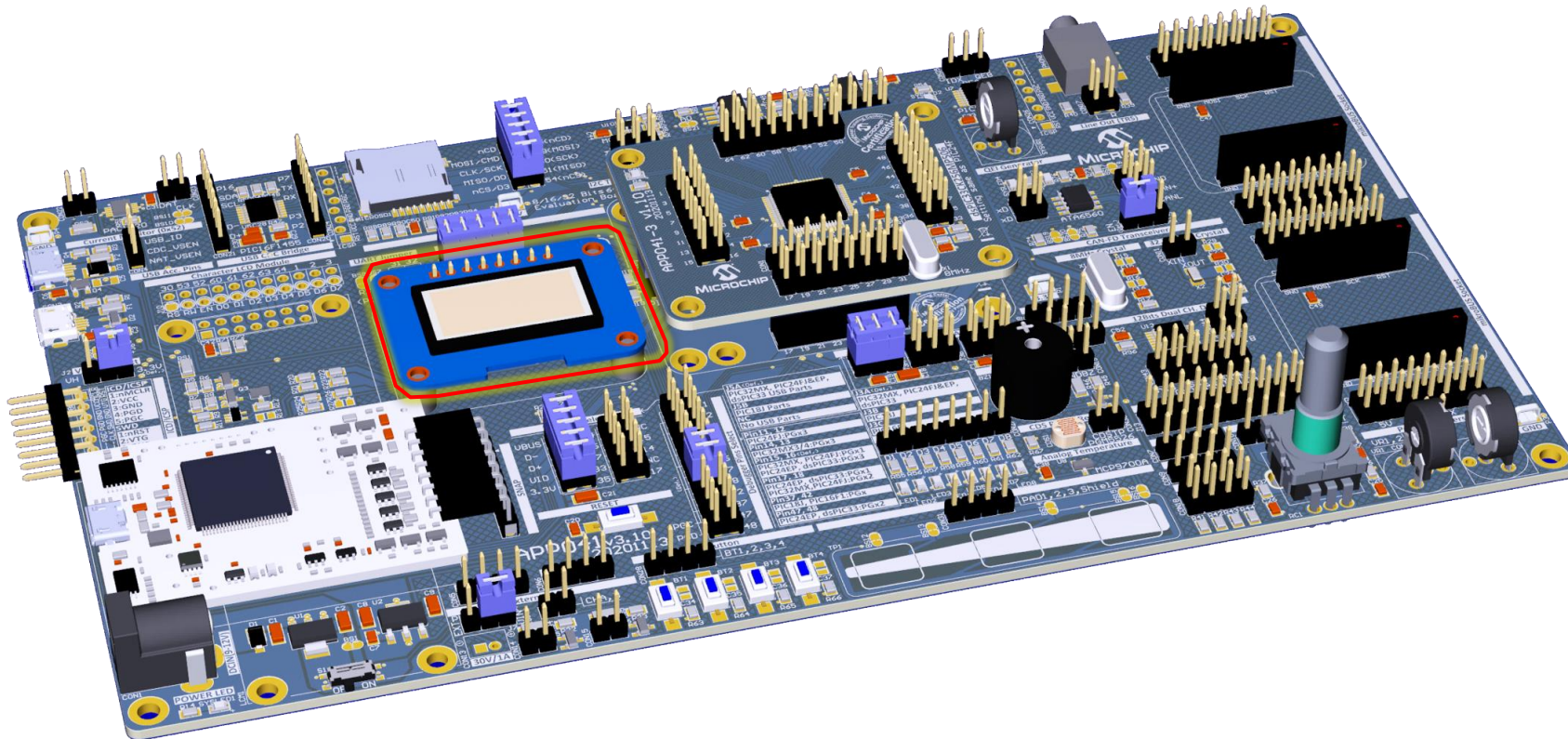
Result



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.

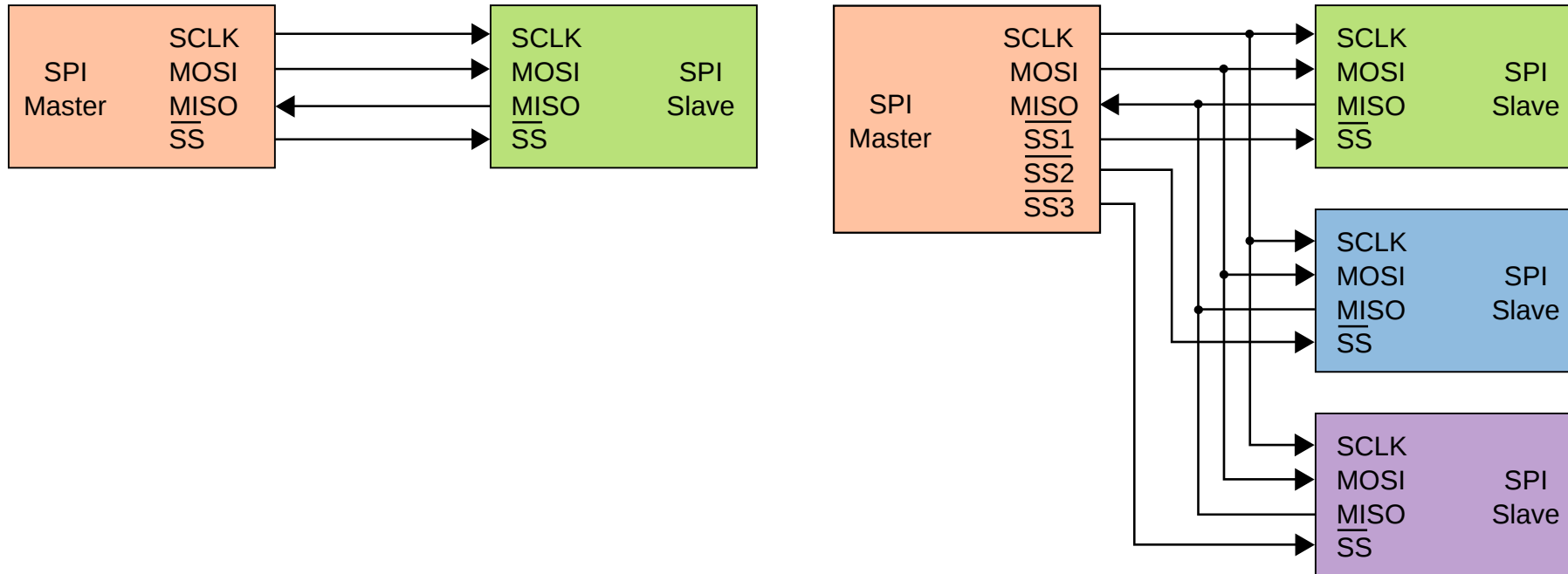


Recourses Requirement for OLED

- **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.
- **Drivers: Delay Driver**
 - **Timing Control (Block)**

What's SPI

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

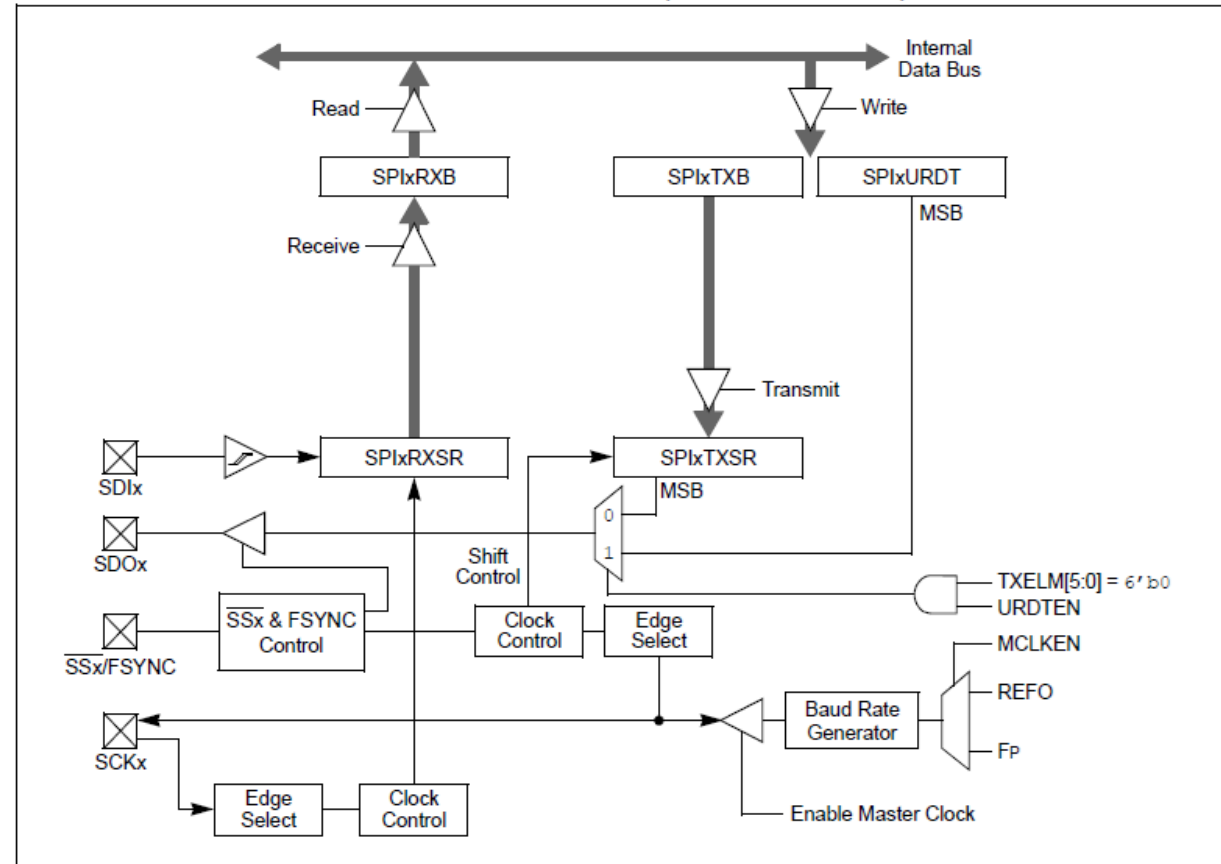


<https://zh.wikipedia.org/wiki/%E5%BA%8F%E5%88%97%E5%91%A8%E9%82%8A%E4%BB%8B%E9%9D%A2>

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

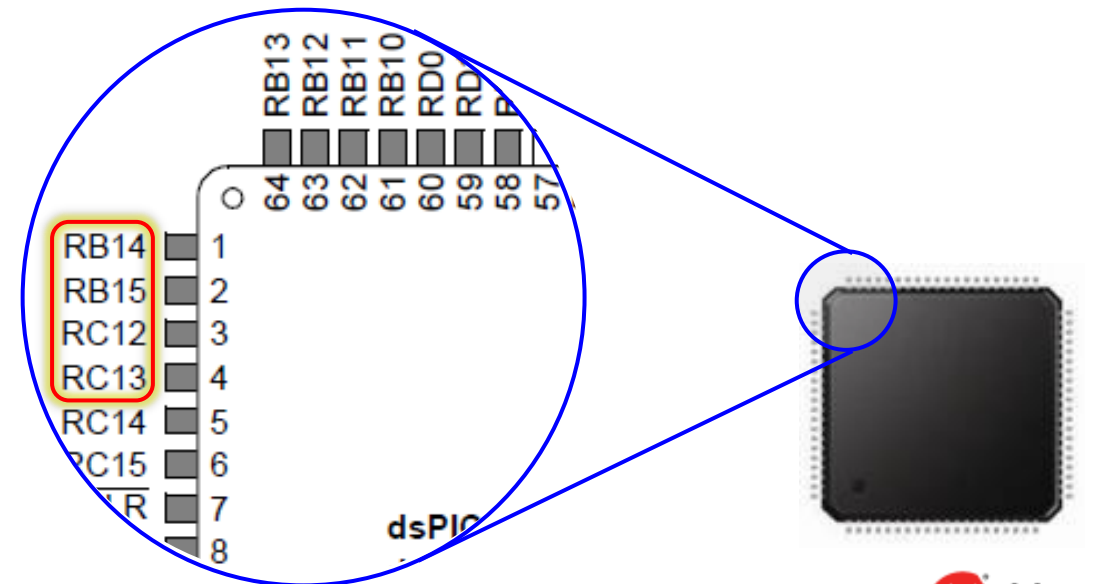
FIGURE 17-1: SPIx MODULE BLOCK DIAGRAM (STANDARD MODE)



What's PPS

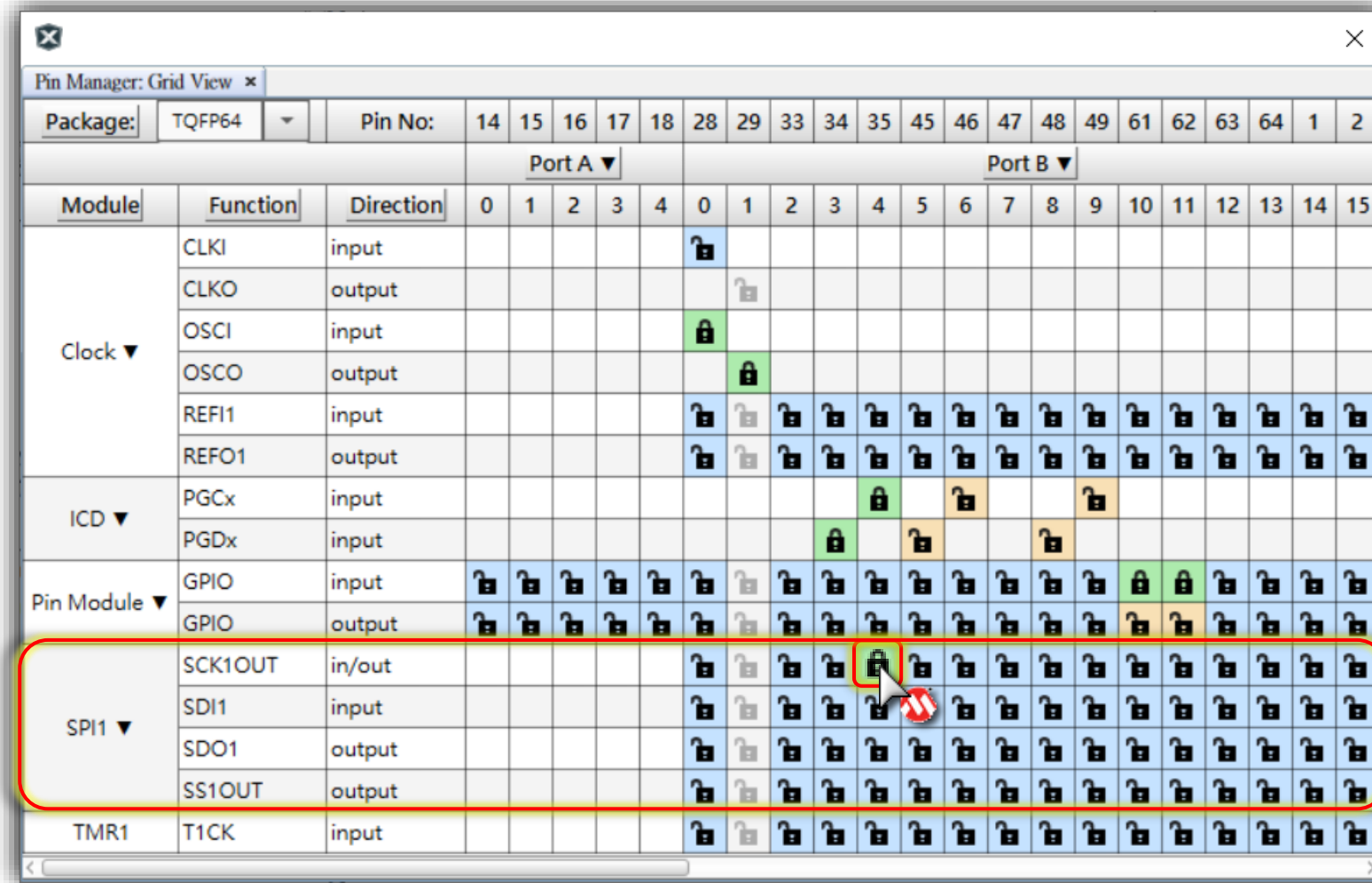
- PPS, Peripheral Pin Select
- Provides a flexible peripheral pins remapping mechanism users can better tailor the microcontroller to their entire application.
- PPS pin definition
 - **RP n** : Input & Output
 - **RP I n** : Input Only
- The peripherals managed by the peripheral pin select are all **digital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

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



PPS Assign Pin at Pin Manager

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



What's Delay Driver

- The Delay Driver aims to provide an abstraction interface for invoking built-in delay macros across device families.
- Flexible APIs for creating blocking delays in millisecond and microsecond time units, use real instruction cycle (T_{cy}) to generate time latency to ensure accurate delay.

OLED Functions of myOLEDlib

- We Provide below functions for OLED:

"myOLED_ssd1306.c", "myOLED_ssd1306.h"

- void myOLED_Init(void);
- void myOLED_ClearScreen(void);
- void myOLED_DrawBitmap(uint8_t column, uint8_t page, uint8_t width, uint8_t height, const uint8_t *byte);
- void myOLED_DrawChar(uint8_t column, uint8_t page, uint8_t byte);
- void myOLED_DrawStr(uint8_t column, uint8_t page, const uint8_t *byte);

/* Initial OLED Module */

/* Clear Screen */

/* Draw Raw Bitmap Picture */

/* Draw Single Character */

/* Draw String */

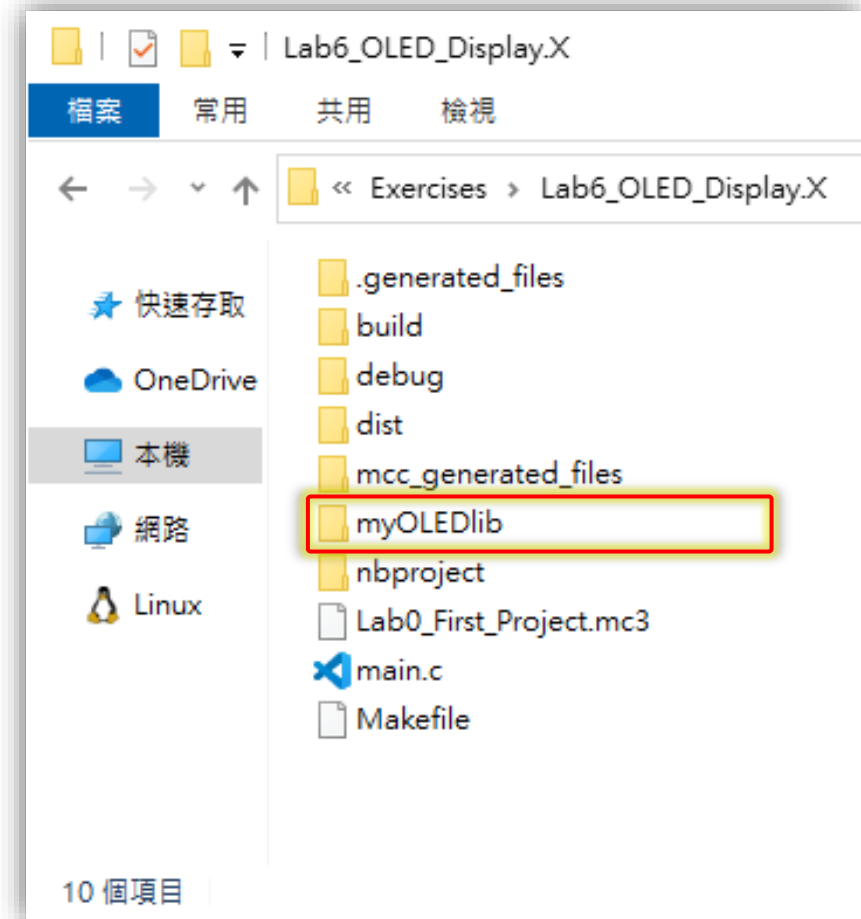
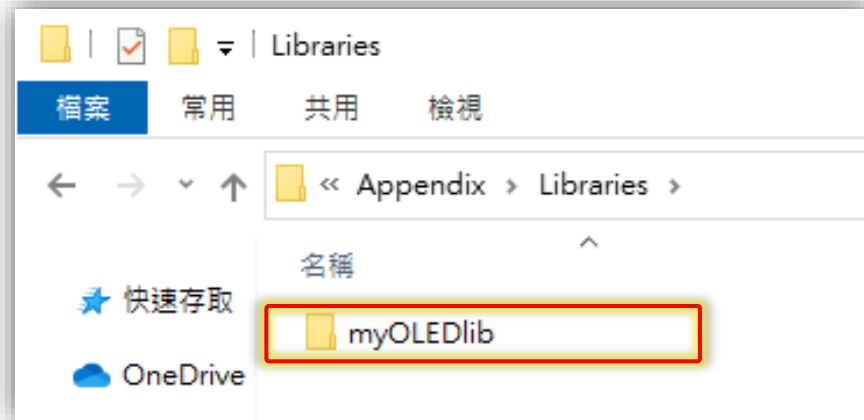
Lab Preparation

[Add OLED Libraries](#)

Lab Preparation

Step 1

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

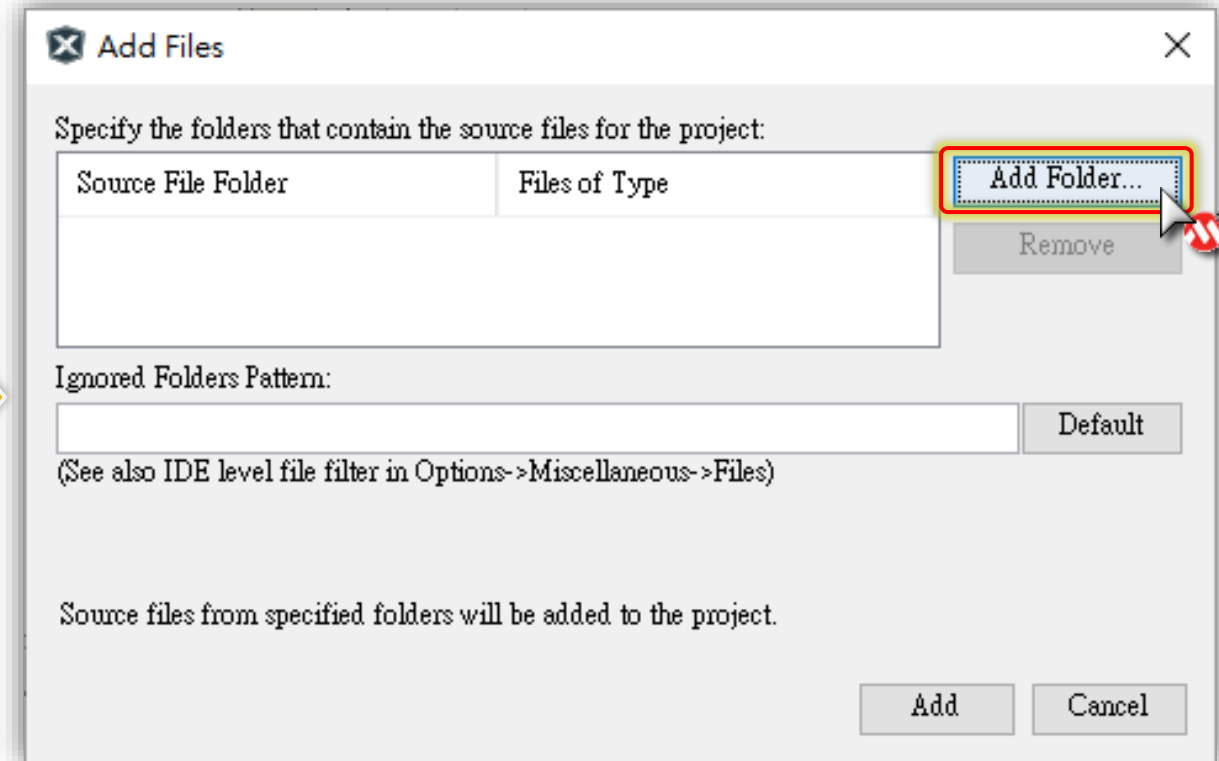
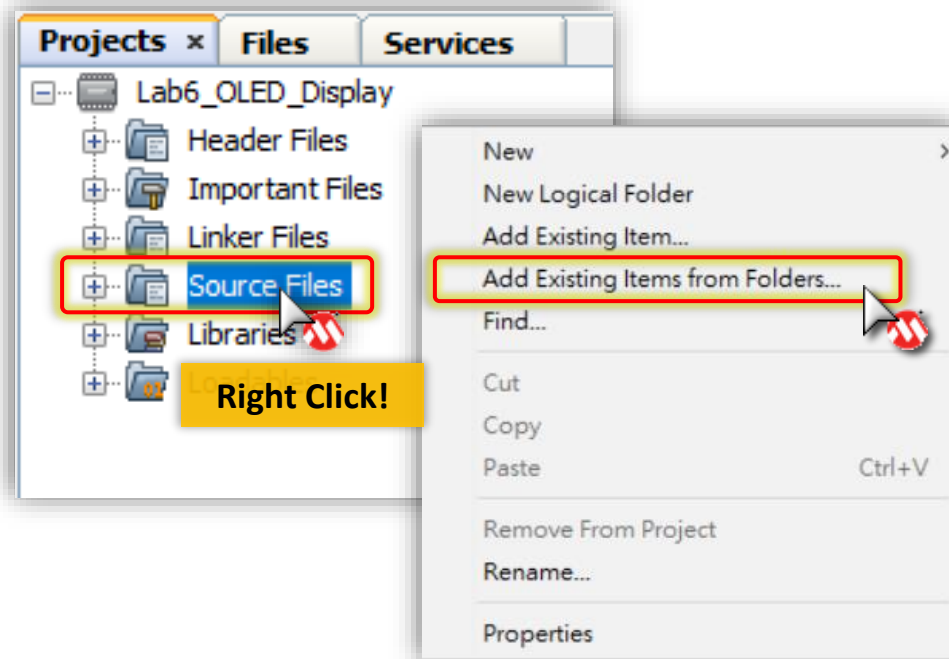


Lab Preparation

Step 2

- **Add Existing Folder to Project**

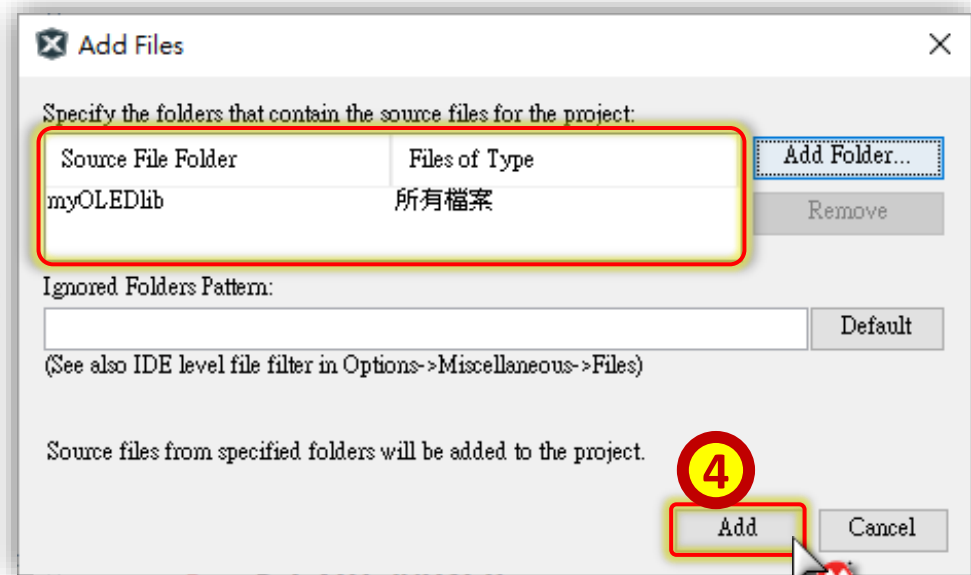
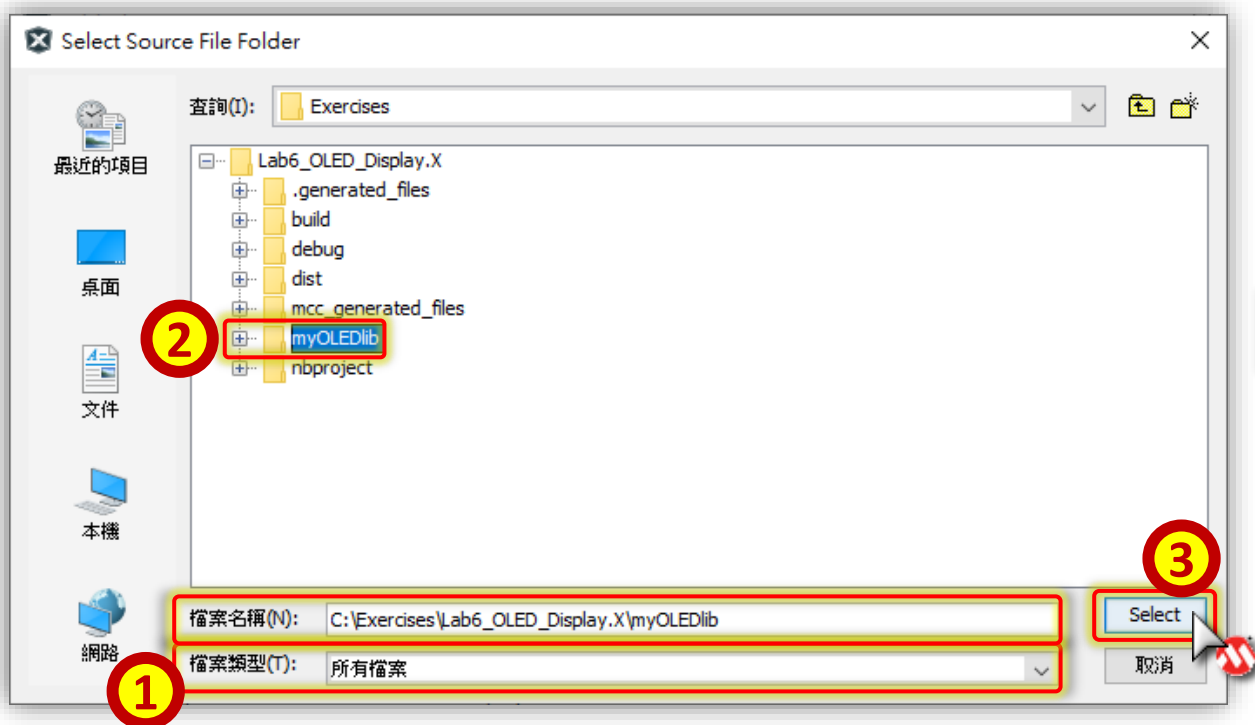
- Right click on "**Source Files**".
- Select "**Add Existing Items from Folders**" at Projects Windows.
- Select "**Add Folder...**".



Lab Preparation

Step 3

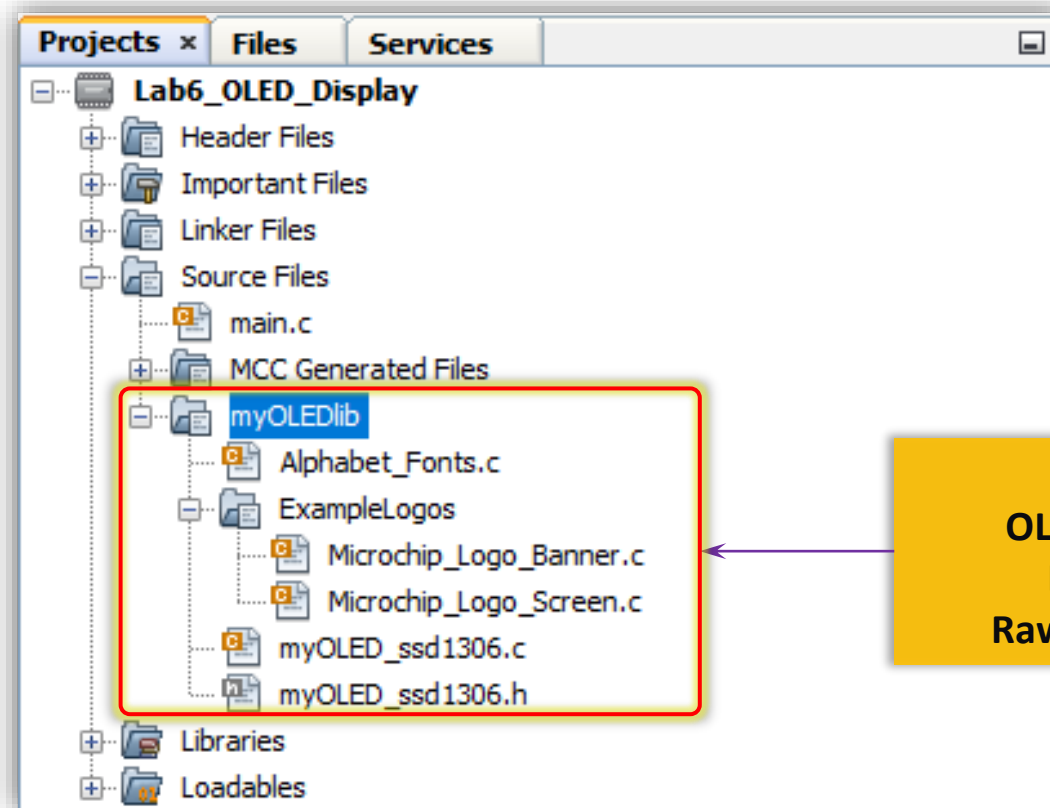
- **Select myOLEDlib folder to Add**
 - Select File Filter(檔案類型) to : "**All Files (所有檔案)**".
 - Click on folder "**myOLEDlib**".
 - Click "**Select**".
 - Click "**Add**".



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

Lab7 OLED Display

- **Base on current project or Open .\Exercises\Lab7_xxx.X to do exercise**
- **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!

Lab7 OLED Display

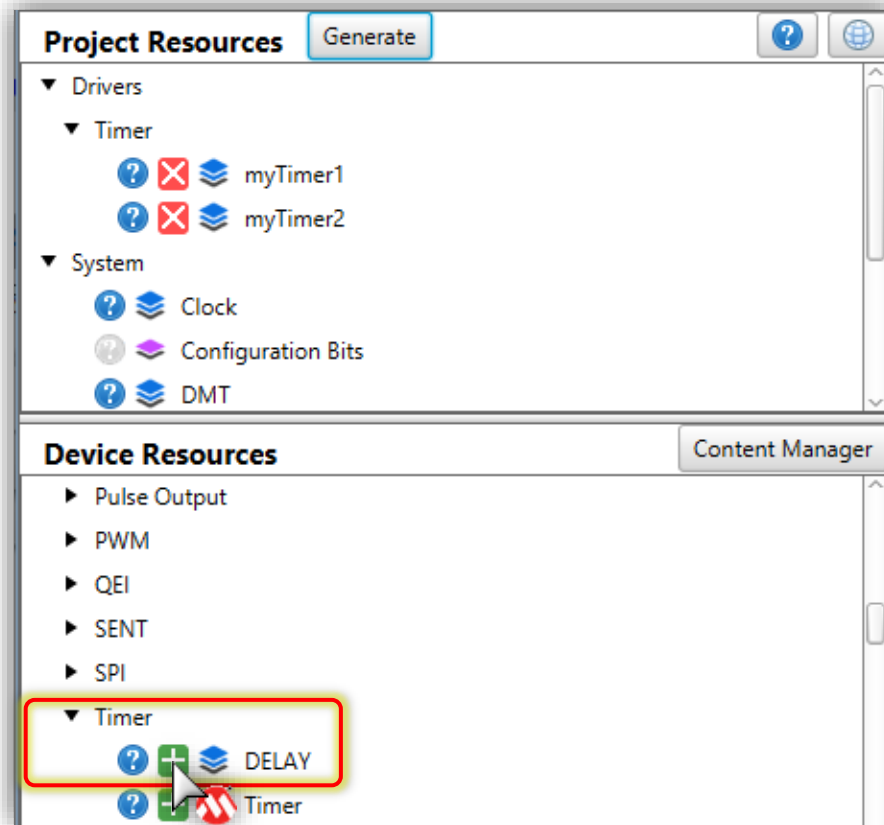
Connection



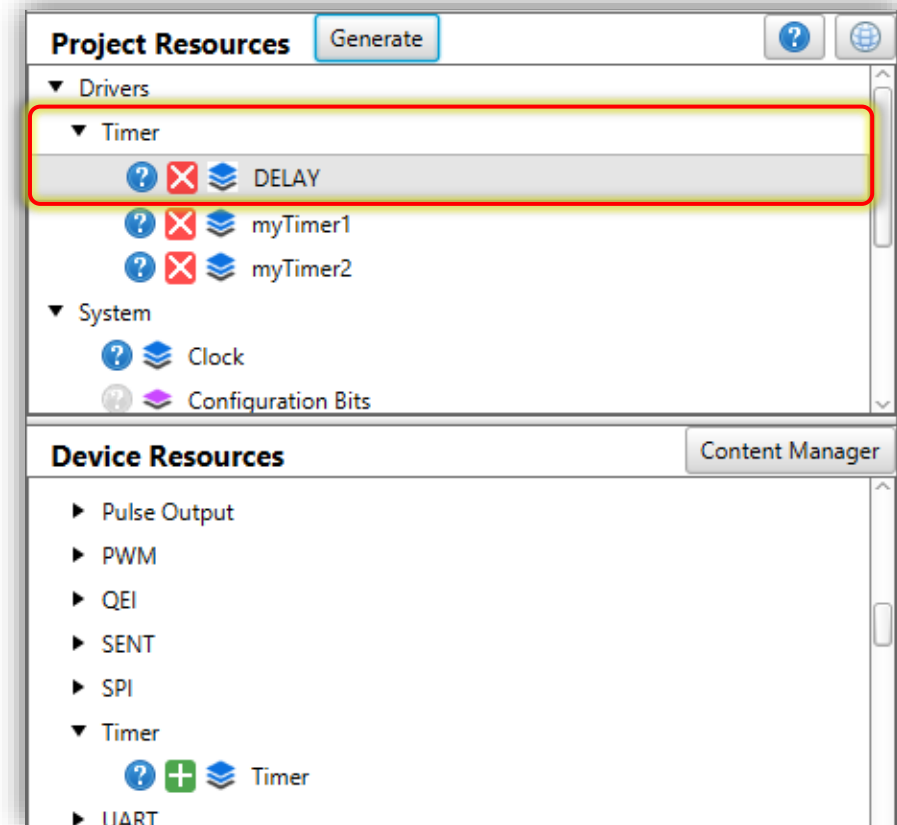
Lab7 OLED Display

Step 1

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



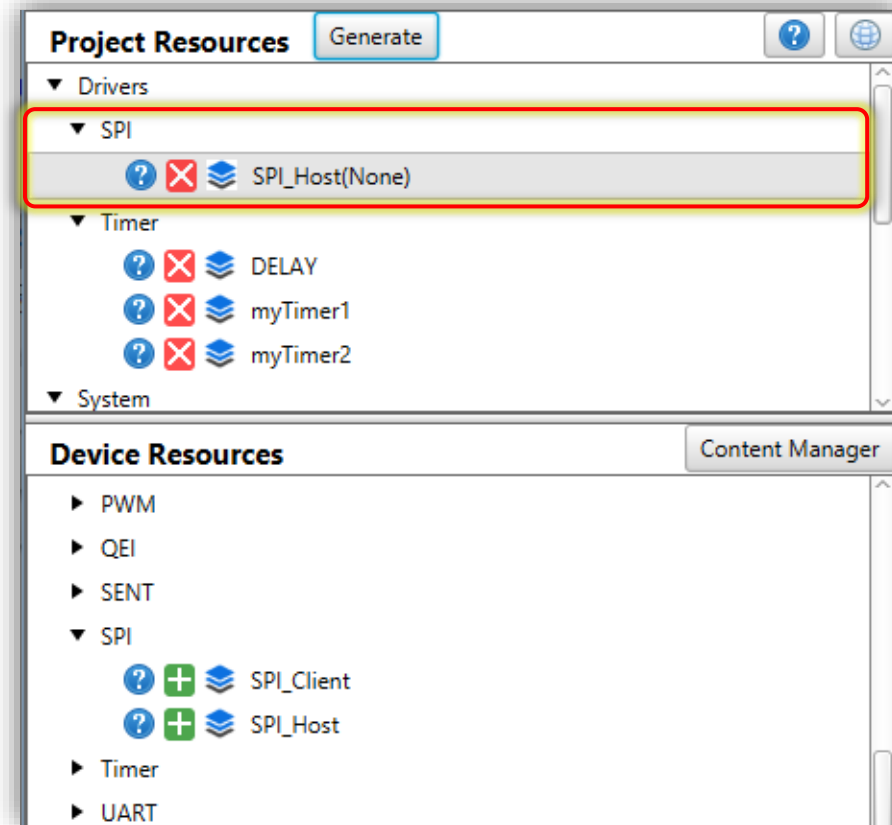
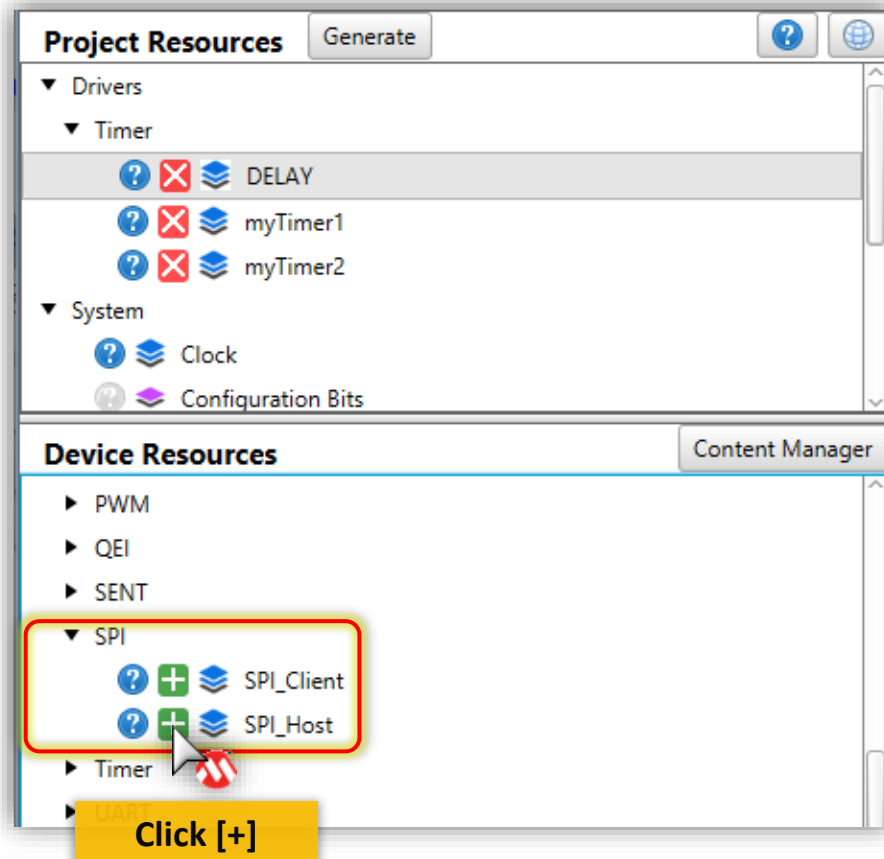
Click [+]



Lab7 OLED Display

Step 2

- Click [+] to add **SPI** > **SPI_Host** Module.



Lab7 OLED Display

Step 3

- Set **SPI Host PLIB Selector Module** → **SPI1**.

SPI1_Host x

Easy View

▼ Software Settings

Custom Name: SPI1_Host

Config Name	Requested Speed (kHz)	Calculated Speed (kHz)	Mode	Data Input Sample At
HOST_CONFIG	125	125	Mode 1	Middle

Add Row

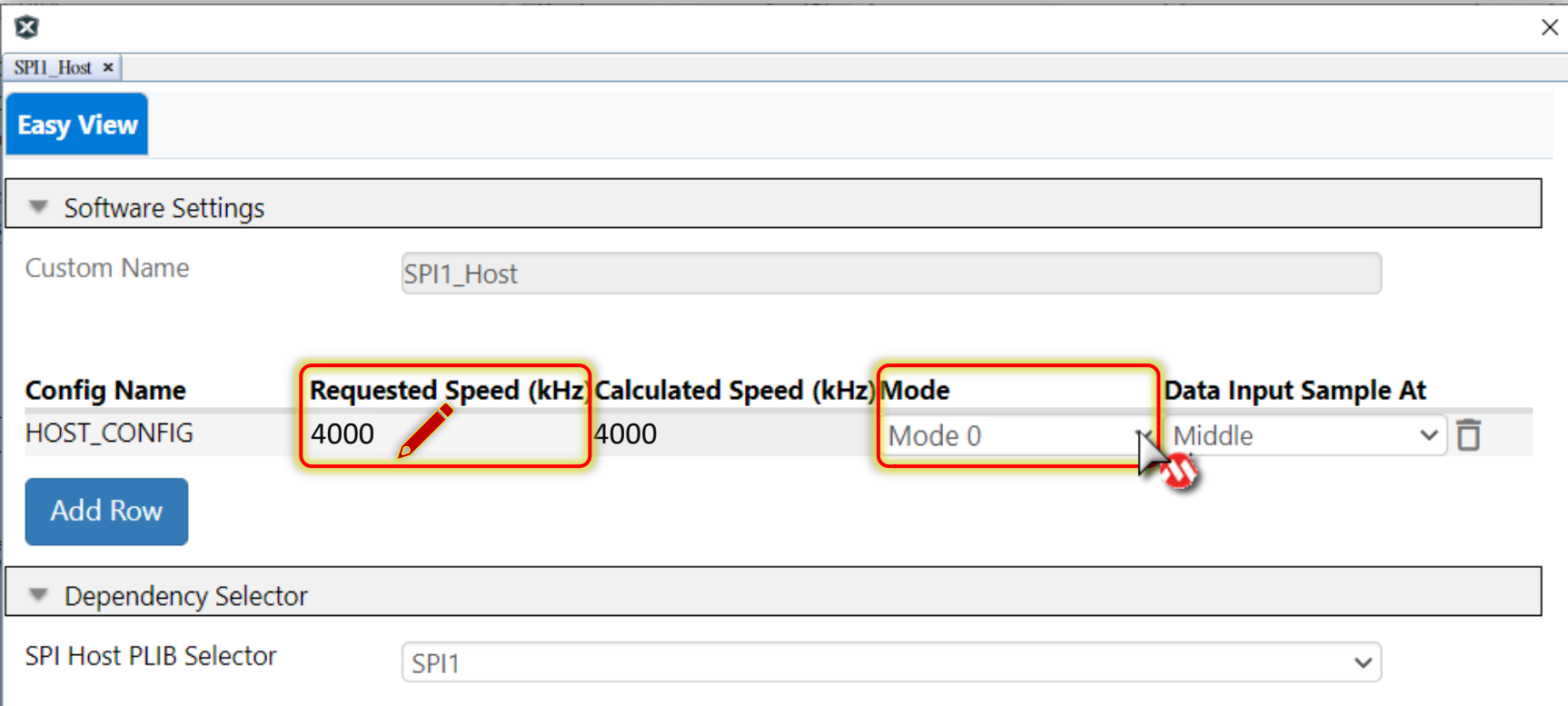
▼ Dependency Selector

SPI Host PLIB Selector: SPI1

Lab7 OLED Display

Step 4

- Set **Requested Speed (kHz)**: 125 → **4000**
- Set **SPI Mode**: Mode 1 → **Mode 0**



The screenshot shows the 'SPI1_Host' configuration window. The 'Easy View' tab is selected. Under 'Software Settings', the 'Custom Name' is 'SPI1_Host'. Below this is a table with the following columns: 'Config Name', 'Requested Speed (kHz)', 'Calculated Speed (kHz)', 'Mode', and 'Data Input Sample At'. The first row has 'HOST_CONFIG' as the config name, '4000' as the requested speed (highlighted with a red box and a pencil icon), '4000' as the calculated speed, 'Mode 0' as the mode (highlighted with a red box), and 'Middle' as the data input sample at. Below the table is an 'Add Row' button. At the bottom, under 'Dependency Selector', the 'SPI Host PLIB Selector' is set to 'SPI1'.

Config Name	Requested Speed (kHz)	Calculated Speed (kHz)	Mode	Data Input Sample At
HOST_CONFIG	4000	4000	Mode 0	Middle

Lab7 OLED Display

Step 5

- Set **PORTC**: **RC13 (SCK1)**, **RC15 (SDO1)**, to PPS SPI Function.
- Set **PORTD**: **RD1 (RS)**, **RD2 (nCS)**, **RD15 (nRST)**, to digital output mode.

Pin Grid View																													
Package:	TQFP64	Pin No:	32	36	37	52	53	3	4	5	6	60	59	58	55	54	44	43	42	39	38	31	30	21	12	11	8		
			PORTC															PORTD											
Module	Function	Direction	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
SPI1_Host	SCK1	in/out	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒		
	SDI1	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒		
	SDO1	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒		
Clock	OSCO	output																											
	OSCI	input																											
	REFI	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒		
	REFO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒		
ICD	PGCx	input																											
	PGDx	input																											
Pins	GPIO	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒		
	GPIO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒		

Lab7 OLED Display

Step 6

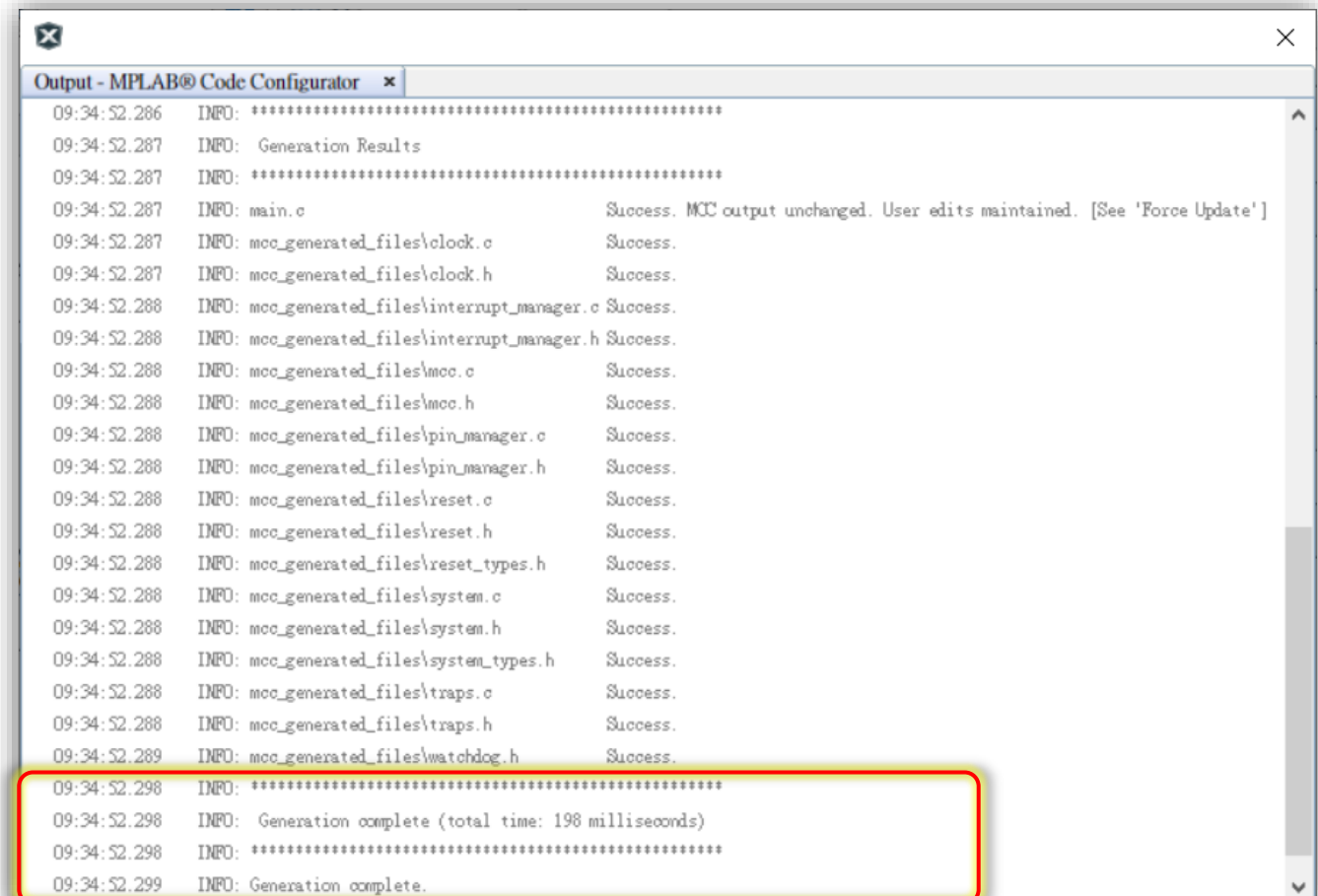
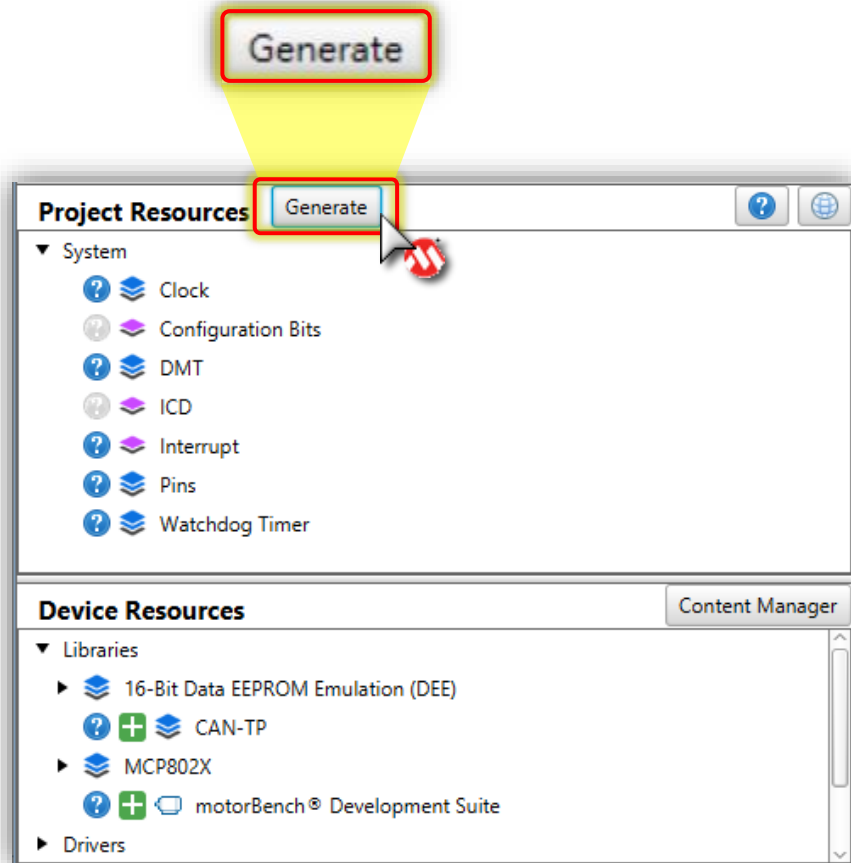
- Set Alias for RD1 , RD2 and RD15.
 - Pins : **RD1** Custom Name → **OLED_RS**, Check Output
 - Pins : **RD2** Custom Name → **OLED_nCS**, Check Output, Start High
 - Pins : **RD15** Custom Name → **OLED_nRST**, Check Output, Start High

Location	Pin Name	Module	Function	Direction	Custom Name	Analog	Start High	Weak Pulldown	Weak Pullup	Open Drain	Interrupt on Change
45	RB5	Pins	GPIO	output	LED2		☐	☐	☐	☐	none ▼
46	RB6	Pins	GPIO	output	LED1		☒	☐	☐	☐	none ▼
22	RC1	Pins	GPIO	output	LED4	☐	☐	☐	☐	☐	none ▼
59	RD1	Pins	GPIO	output	OLED_RS		☐	☐	☐	☐	none ▼
21	RD12	Pins	GPIO	output	LED3		☒	☐	☐	☐	none ▼
8	RD15	Pins	GPIO	output	OLED_nRST		☒	☐	☐	☐	none ▼
58	RD2	Pins	GPIO	output	OLED_nCS		☒	☐	☐	☐	none ▼

Lab7 OLED Display

Step 7

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



Lab7 OLED Display

Code Example

```
...
#include "mcc_generated_files/timer/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[];
...

int main(void)
{
    SYSTEM_Initialize();

    myTimer1.TimeoutCallbackRegister(myTimer1_Callback);
    myTimer2.TimeoutCallbackRegister(myTimer2_Callback);

    myOLED_Init();

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

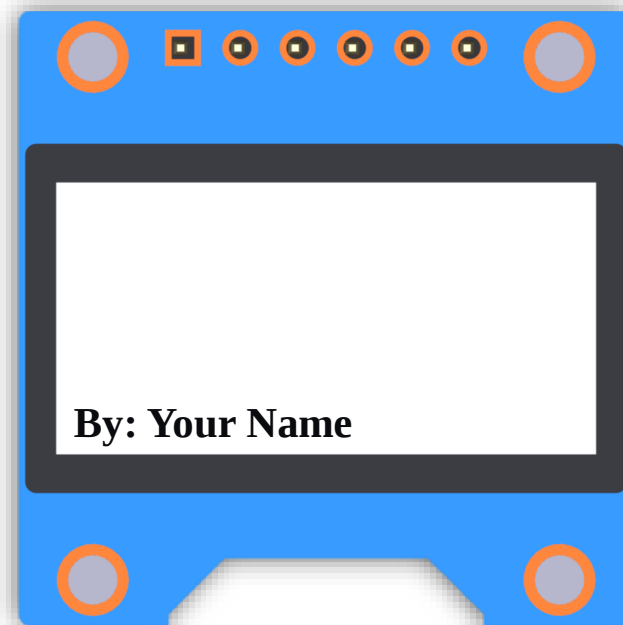
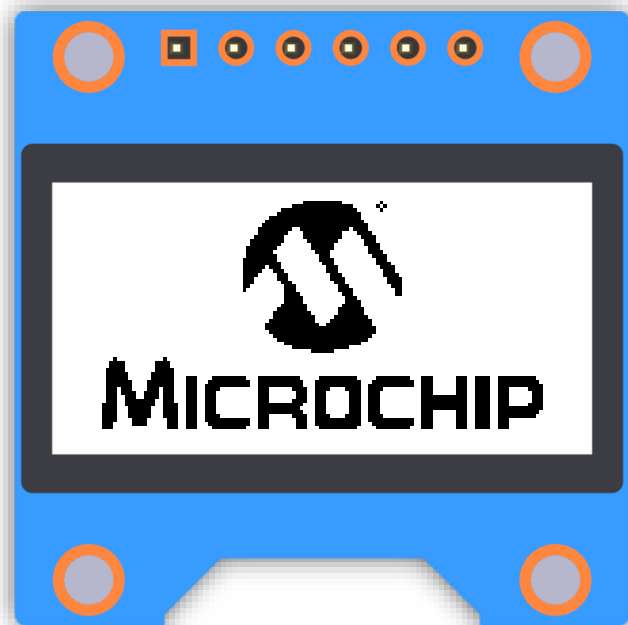
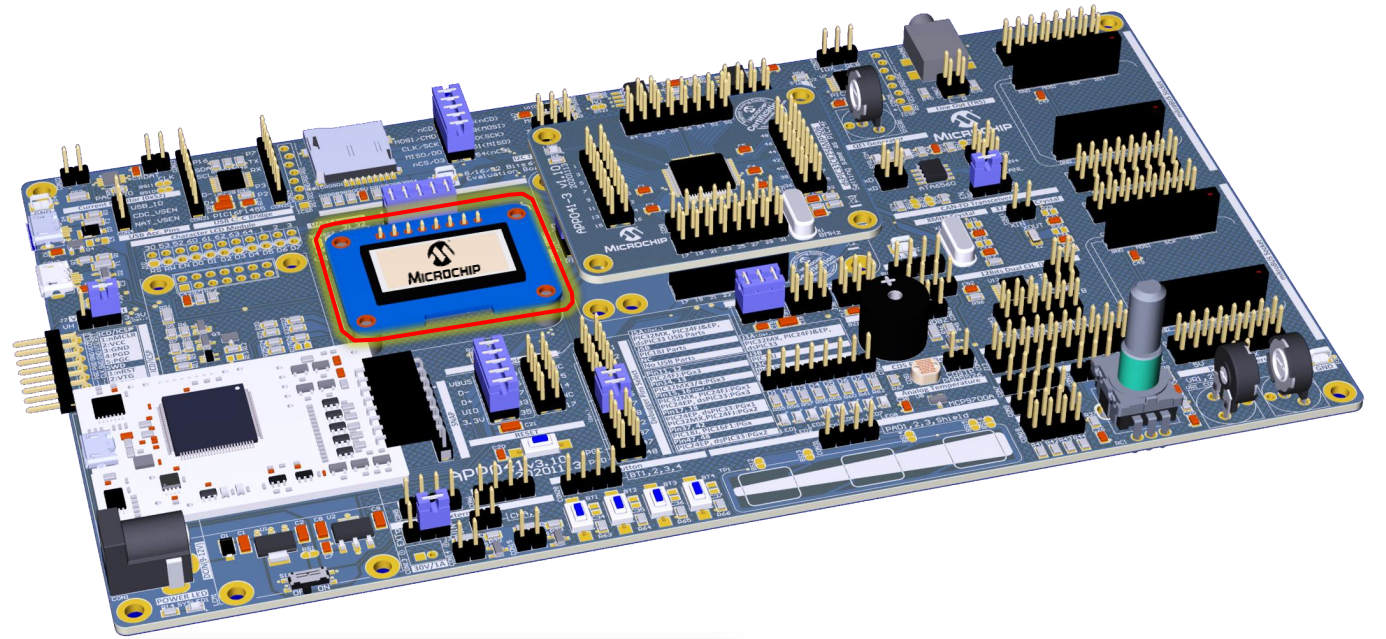
    DELAY_milliseconds(2000);

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

    while (1)
    {
        ...
    }
}
```

Lab7 OLED Display

Result



Lab8 Customize Screen

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

Let's go!

Lab8 Customize Screen

Create your own logo

SamplePicture02.bmp - 小畫家

檔案 常用 檢視

調整大小

調整大小及扭曲

調整大小

依照(B): ☐ 百分比 ☒ 像素

水平(H): 128

垂直(V): 64

☐ 維持外觀比例(M)

扭曲(度)

水平(O): 0

垂直(E): 0

確定 取消

另存新檔

組合管理 新增資料夾

檔案名稱(N): Microchip_Logo_Screen.bmp

存檔類型(T): 單色點陣圖 (*.bmp;*.dib)

存檔(S) 取消

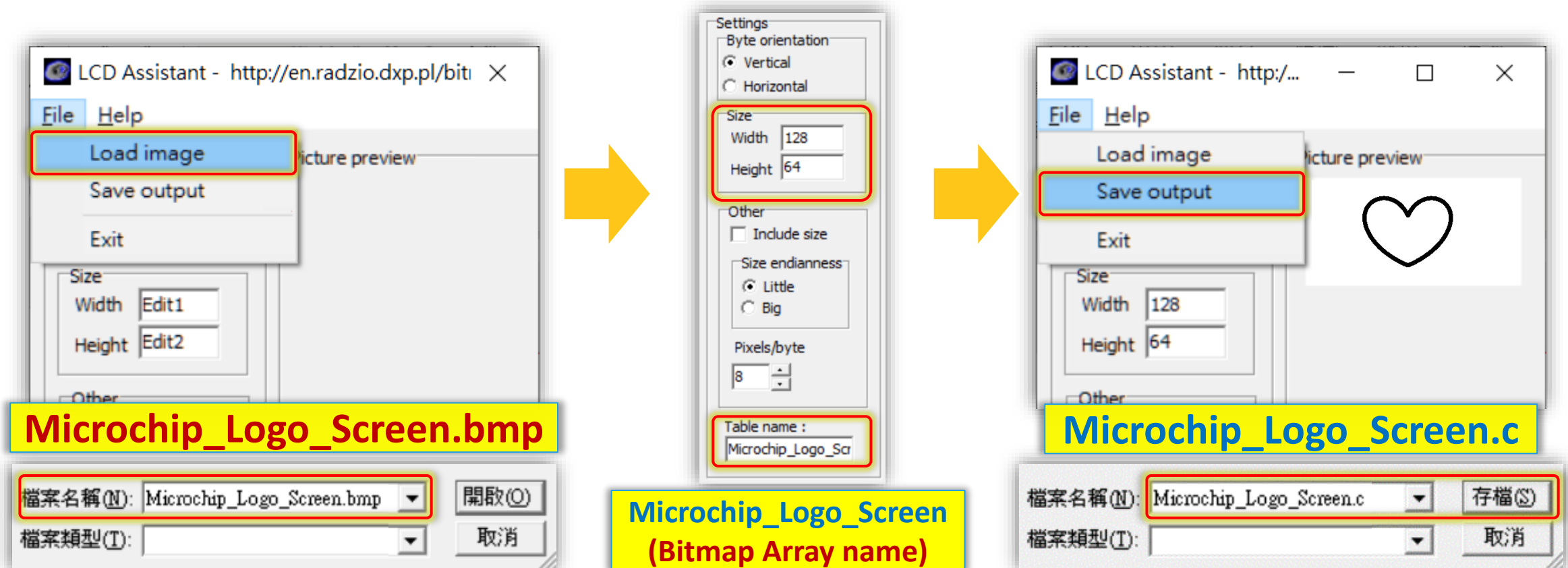
Microchip_Logo_Screen.bmp

Monochrome Bitmap

Lab8 Customize Screen

Create your own logo

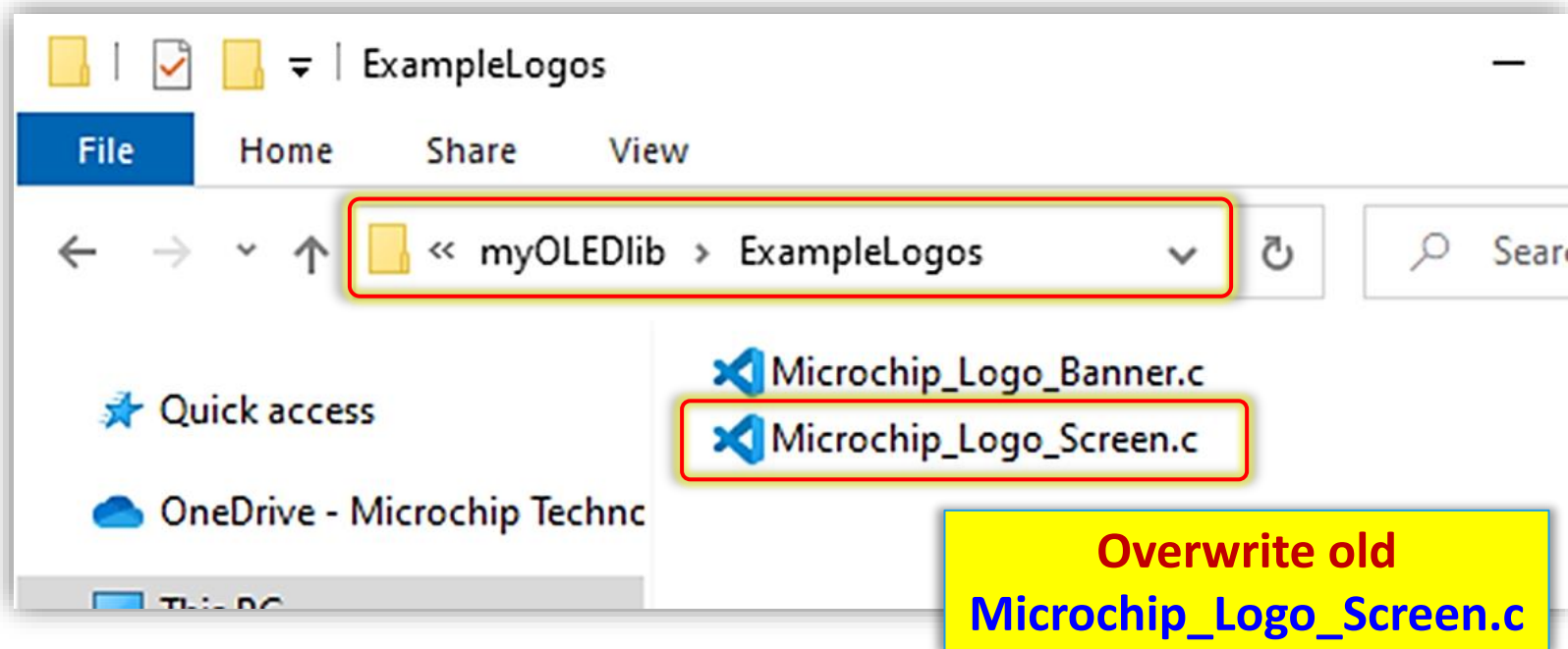
- Execute `\Tools\LCDAssistant.exe`
Load `Microchip_Logo_Screen.bmp` and Save output to `Microchip_Logo_Screen.c`



Lab8 Customize Screen

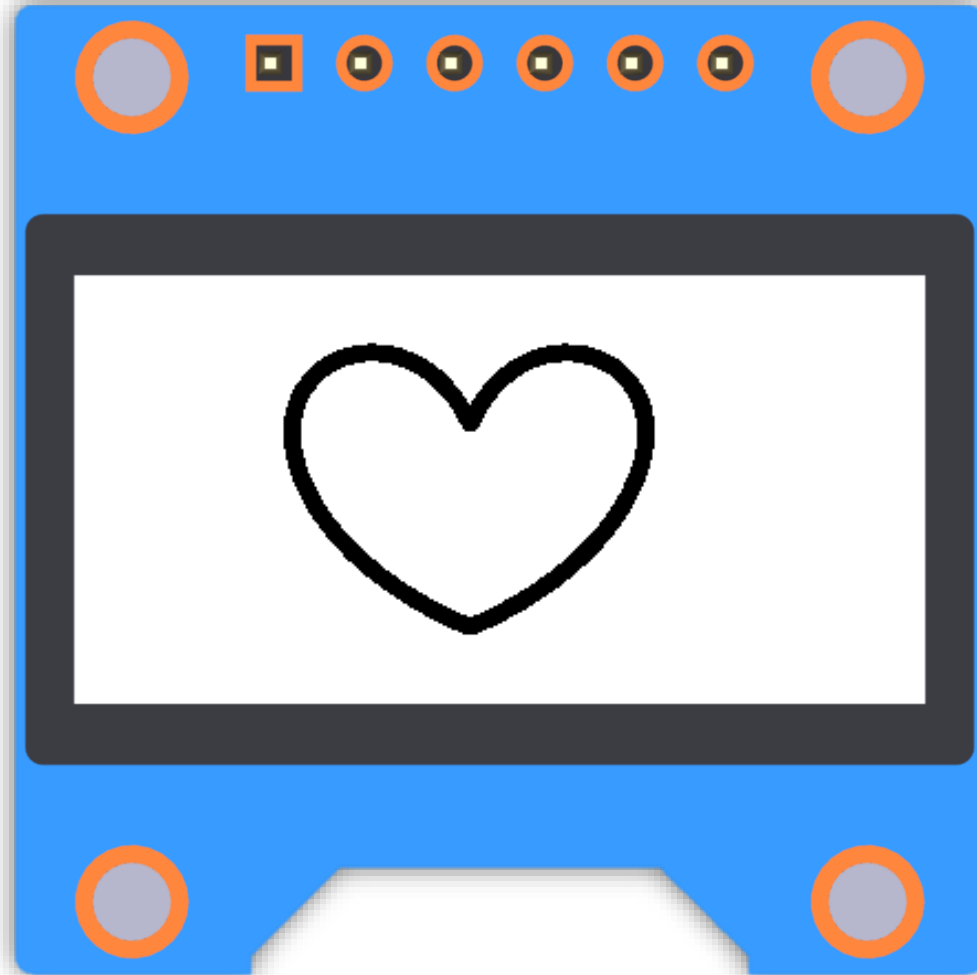
Create your own logo

- Copy new **Microchip_Logo_Screen.c** to overwrite old one in folder.
 - \Labn_xxx.X\myOLEDLib\ExampleLogos



Lab8 Customize Screen

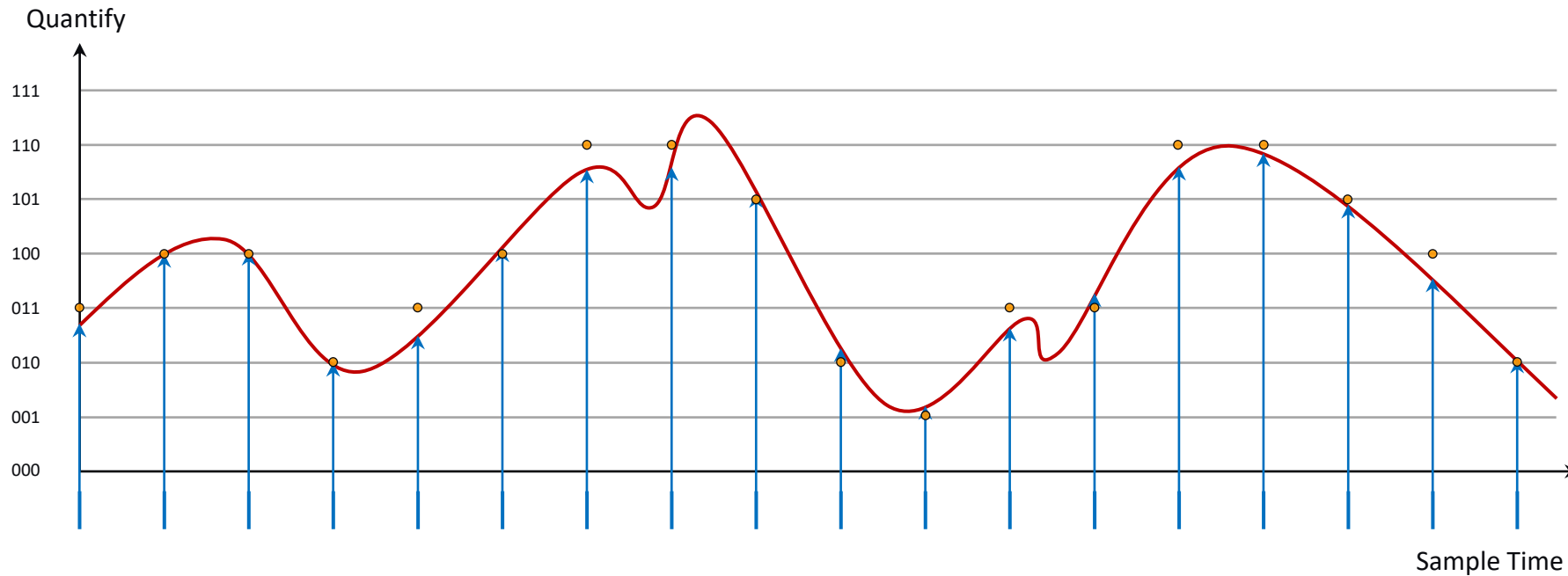
Create your own logo



12 Bits High Speed ADC Application

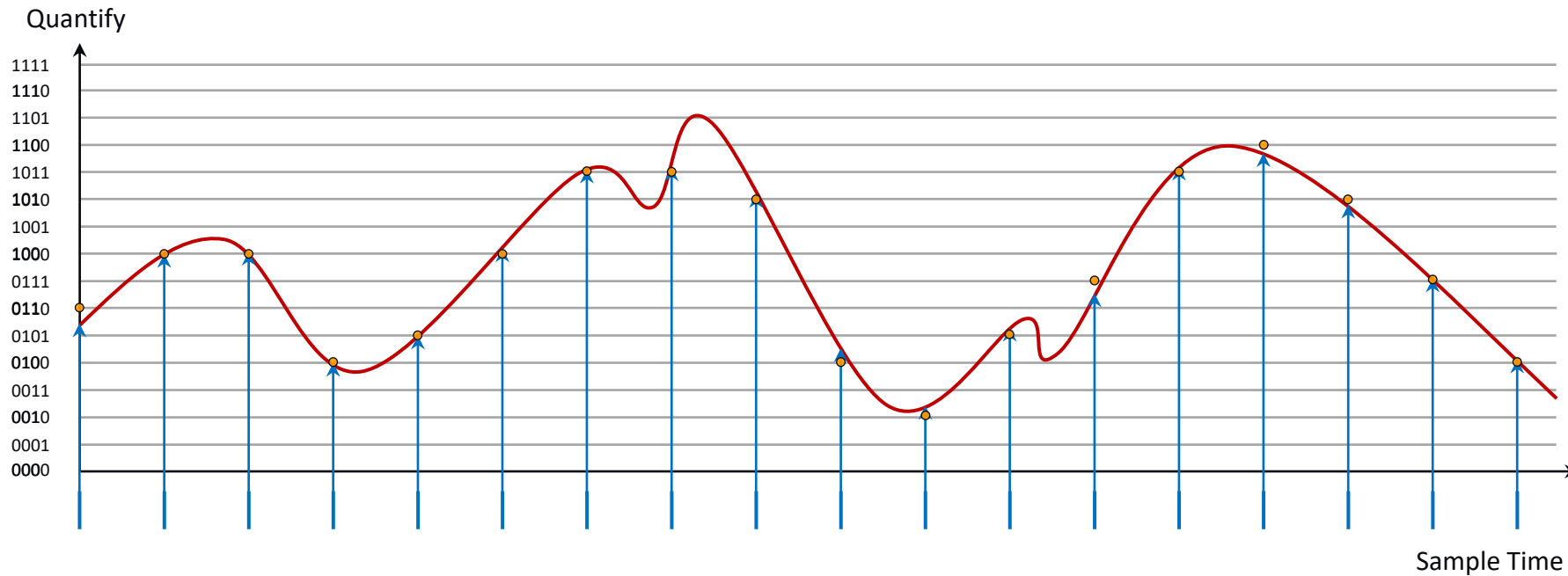
What's ADC ?

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



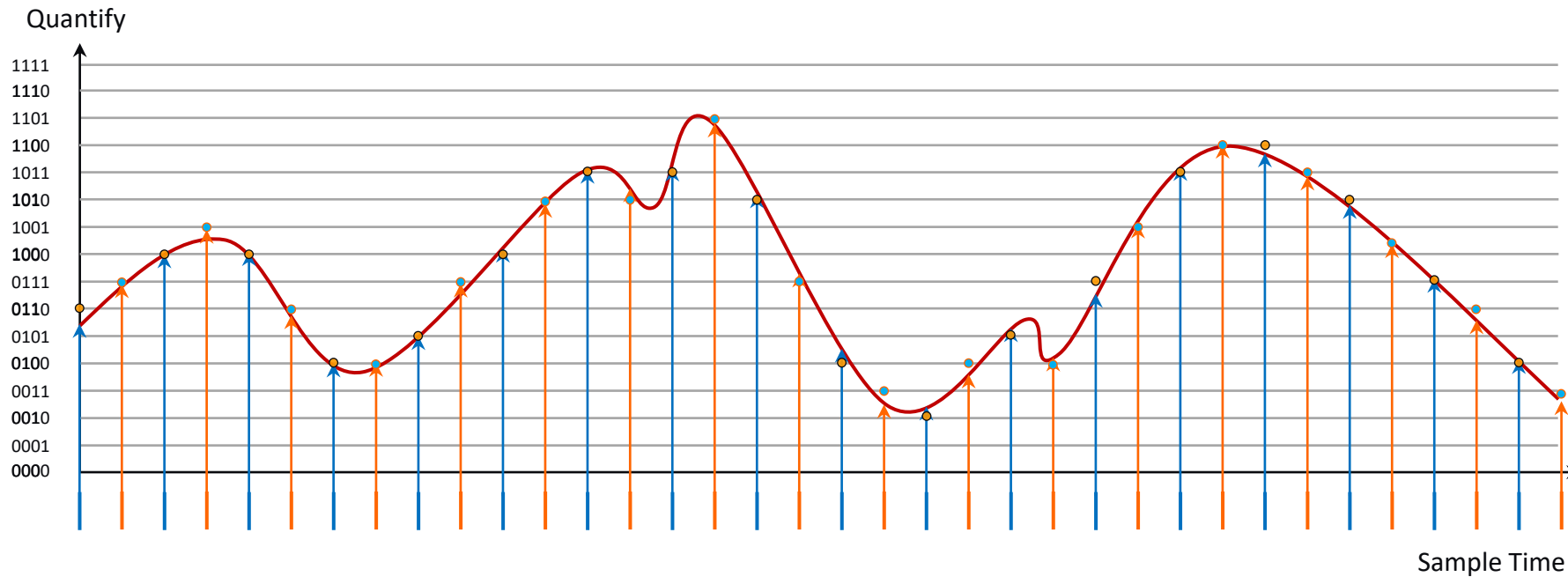
Resolution

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



Sample Rate

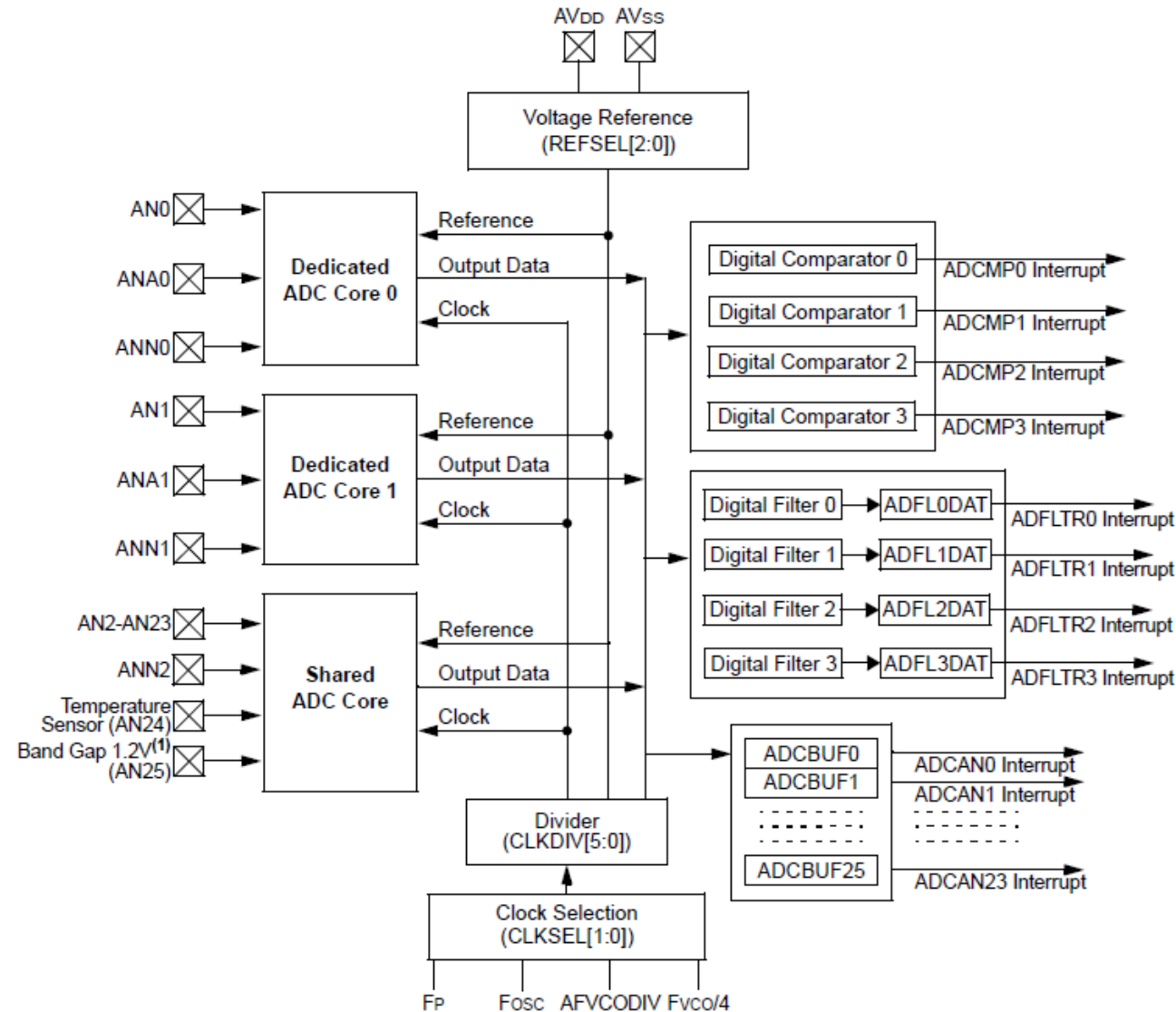
- The analog signal is required to define the rate at which new digital values are sampled from the analog signal.
- This faithful reproduction is only possible if the sampling rate is higher than twice the highest frequency of the signal. (wiki)



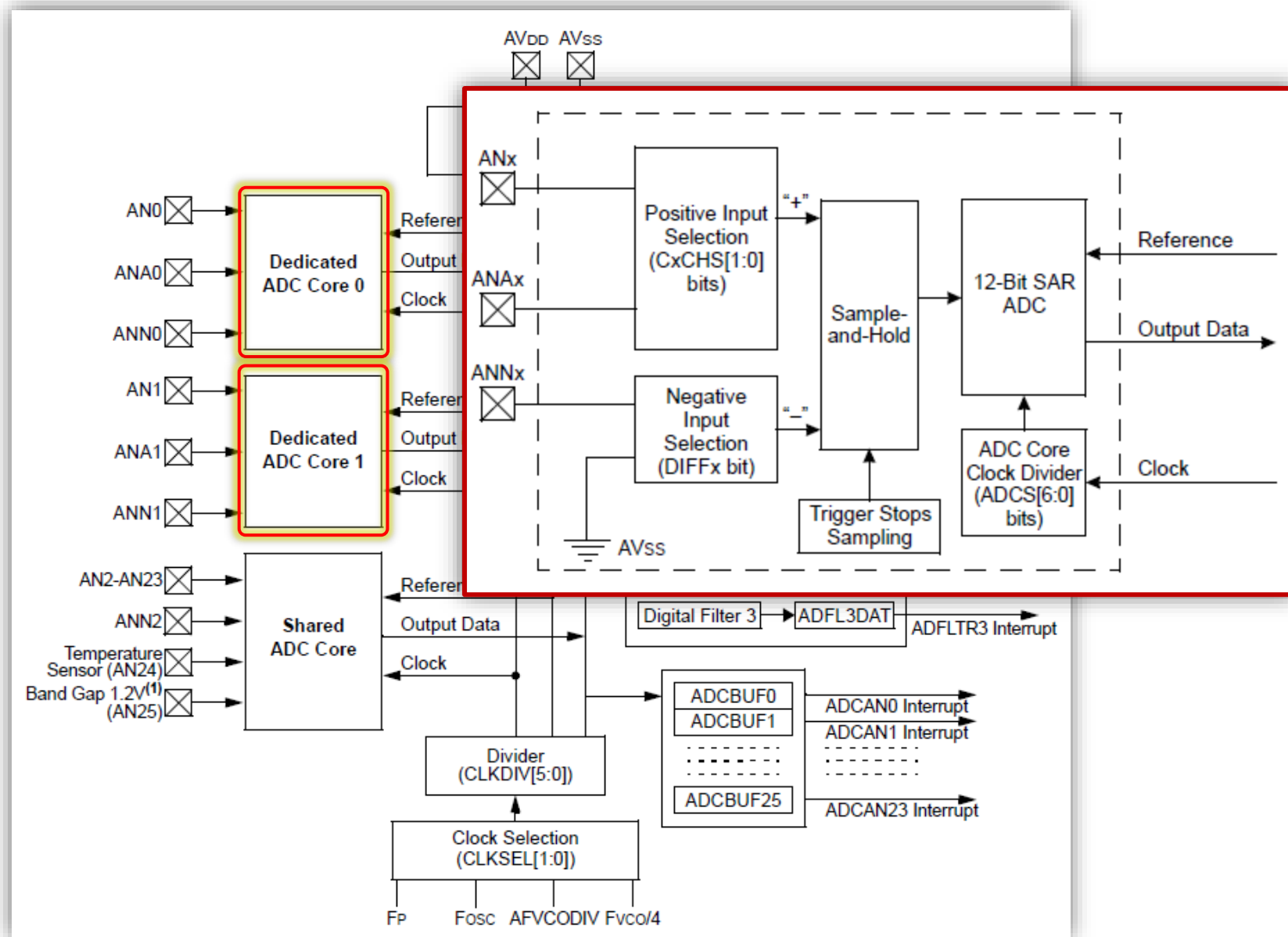
dsPIC33CK ADC Features

- **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

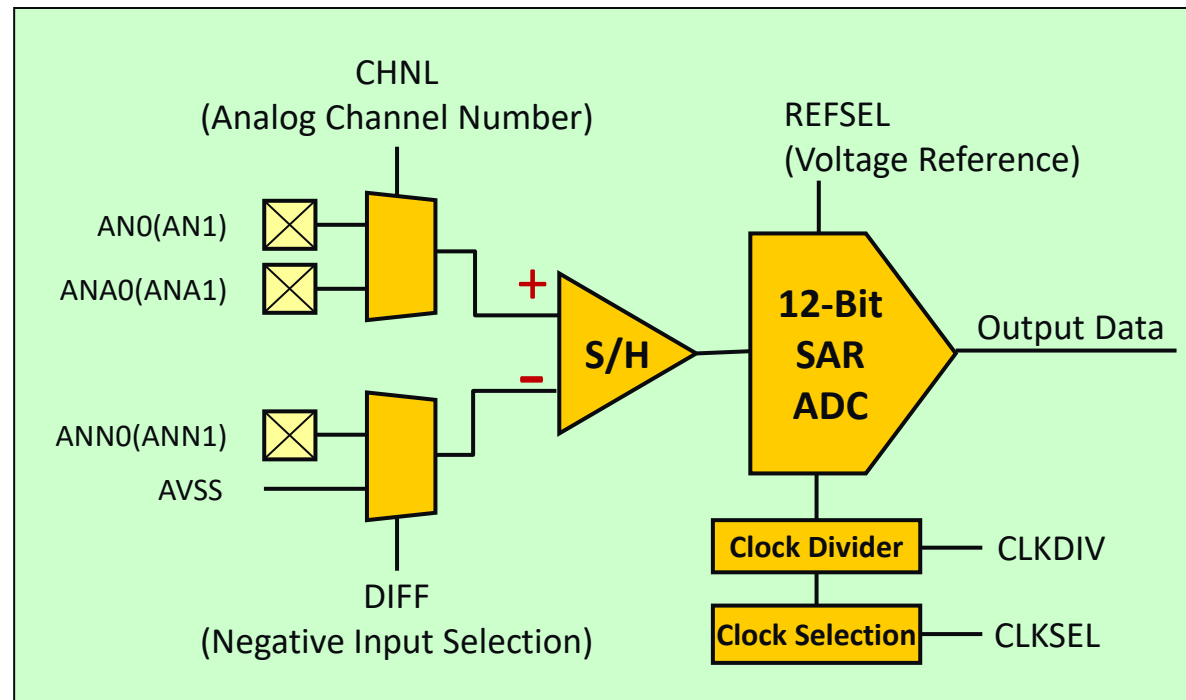


ADC Dedicate Core Block Diagram

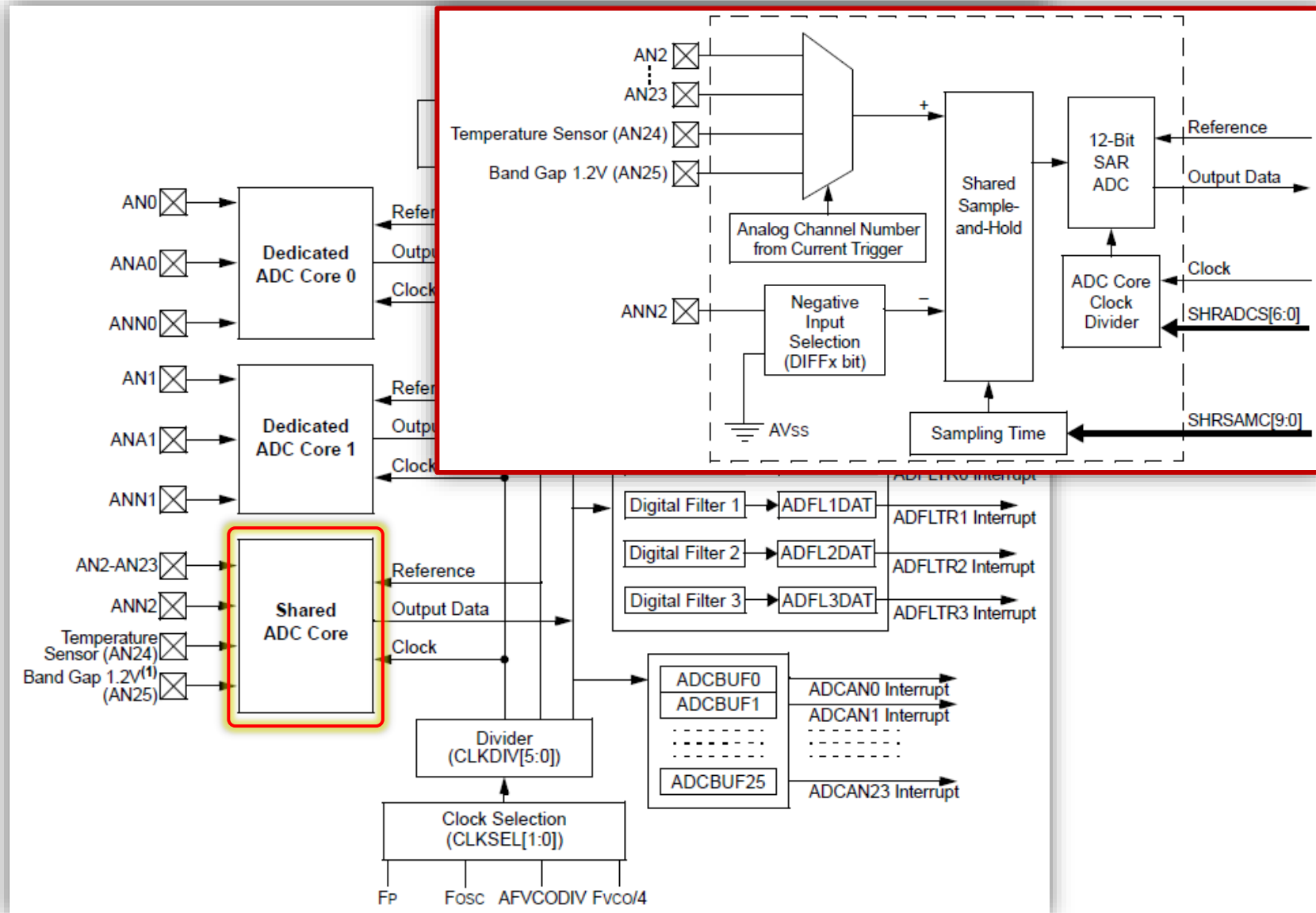


ADC Dedicate Core Channel Selection

- 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**.

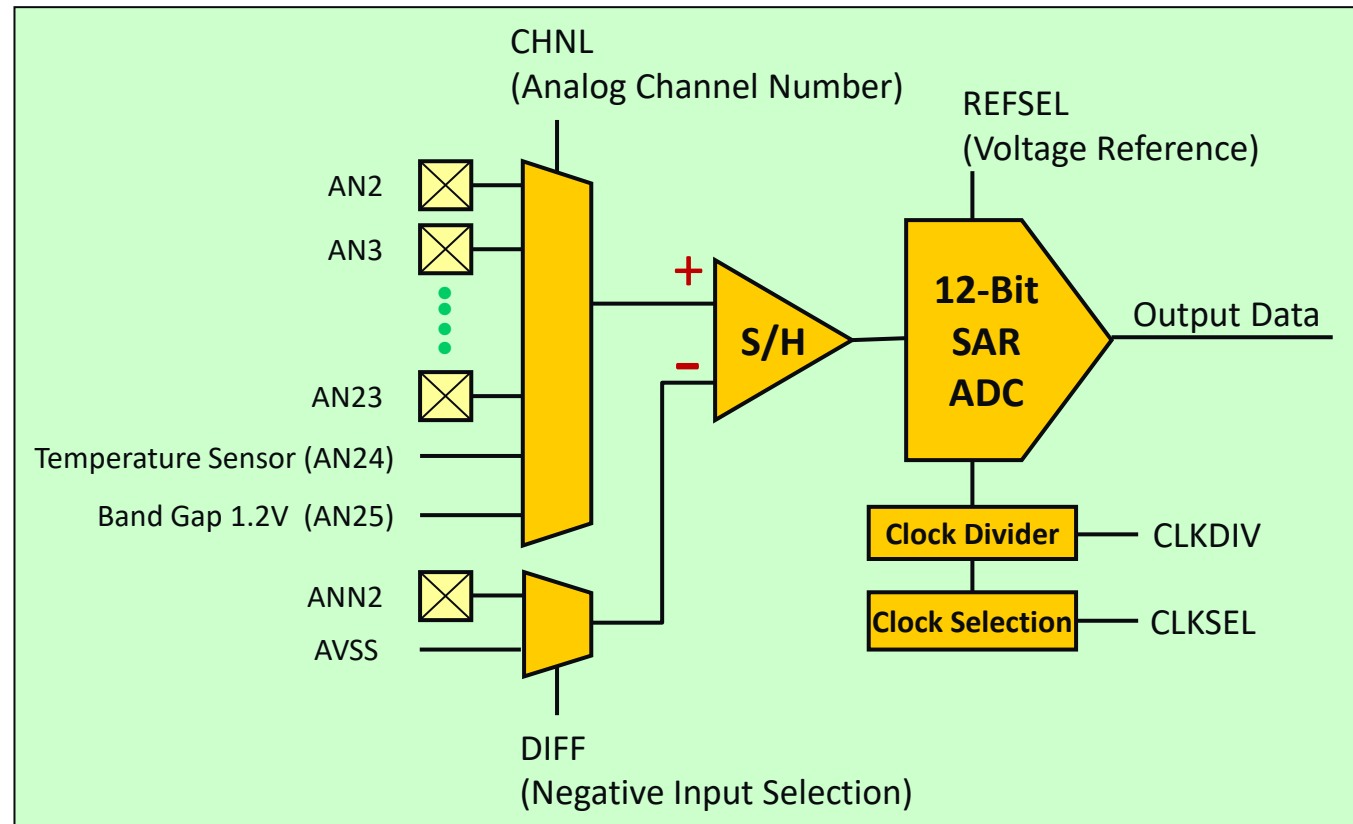


ADC Shared Core Block Diagram



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**.



Functions of Melody ADC Driver

- Melody Provide below functions for Timer:

Prototype definition : "adc1.h"

- **ADC_INTERFACE**

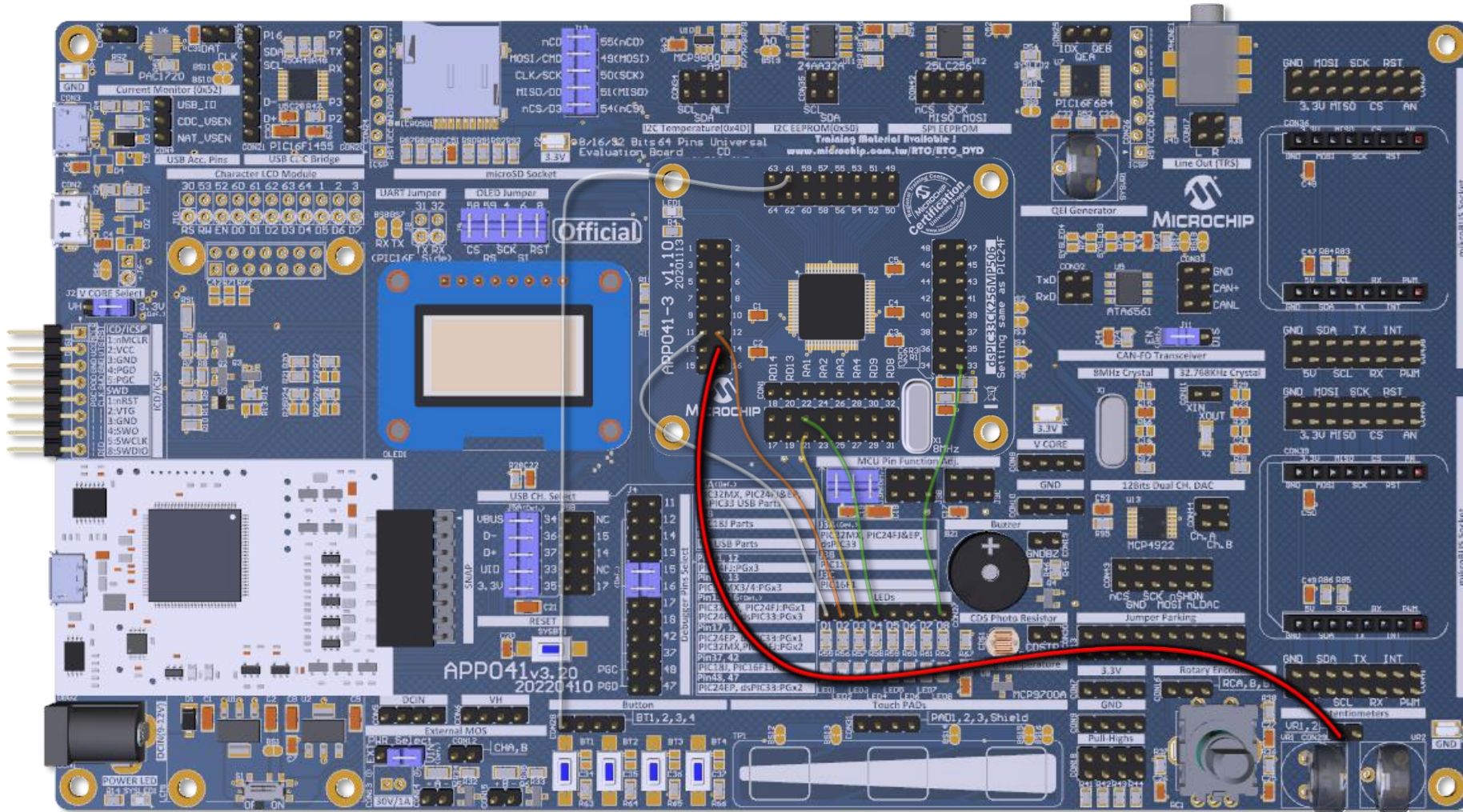
- (void) *ADC1*.Initialize(void);
- (void) *ADC1*.Deinitialize(void);
- (void) *ADC1*.Enable(void);
- (void) *ADC1*.Disable(void);
- (void) *ADC1*.SoftwareTriggerEnable(void);
- (void) *ADC1*.SoftwareTriggerDisable(void);
- (void) *ADC1*.ChannelSelect(enum ADC_CHANNEL channel);
- (uint16_t) *ADC1*.ConversionResultGet(enum ADC_CHANNEL channel);
- (bool) *ADC1*.IsConversionComplete(enum ADC_CHANNEL channel);
- (void) *ADC1*.ResolutionSet(enum ADC_RESOLUTION_TYPE resolution);
- (void) *ADC1*.CommonCallbackRegister(void(*callback)(void));
- (void) *ADC1*.Tasks(void);
- (void) *ADC1*.adcMulticoreInterface->ChannelCallbackRegister(void(*callback)(...));

Lab9 ADC Single Channel Periodically

- Base on current project or Open `.\Exercises\Lab9_xxx.X` to do exercise
- 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!

Connection

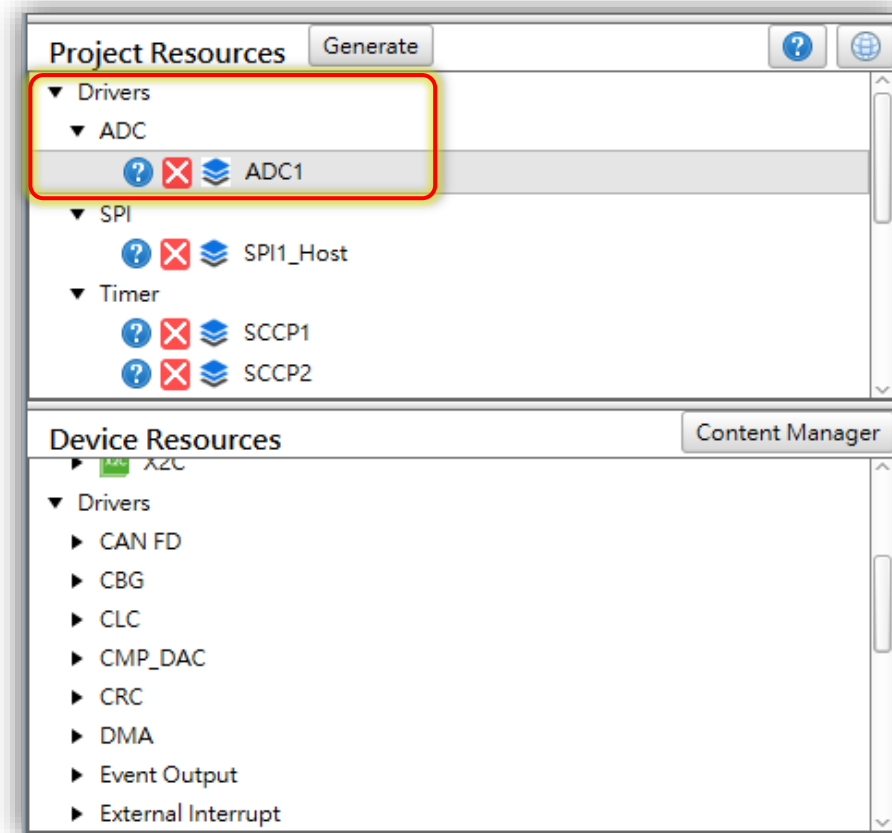
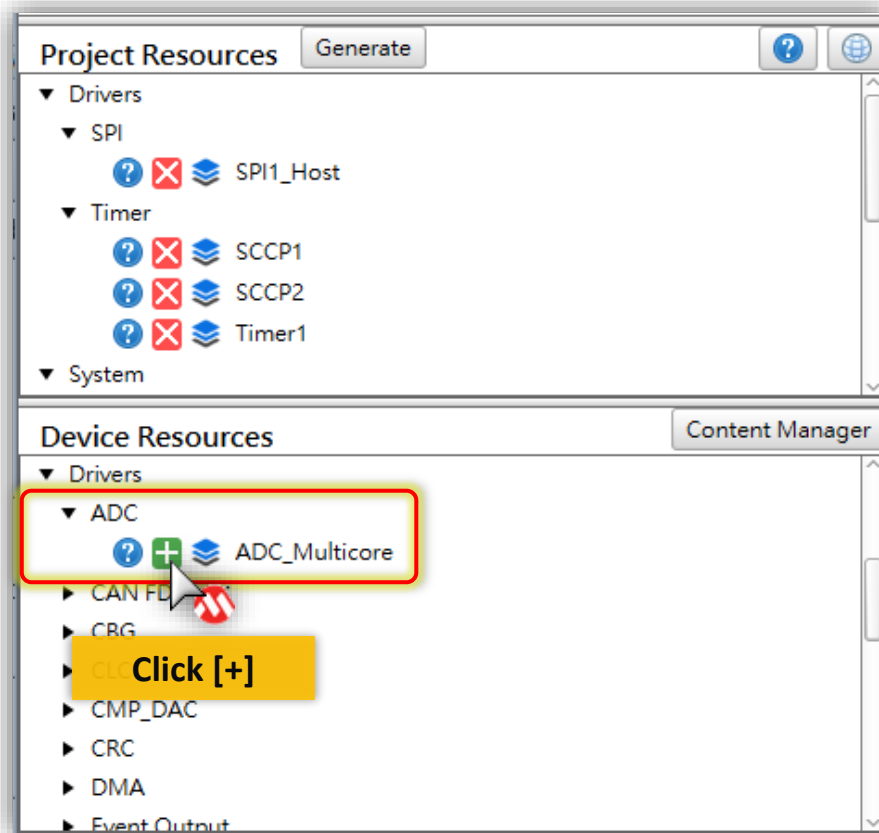


PIM Pin#		Symbol
14		VR1

Lab9 ADC Single Channel Periodically

Step 1

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



Lab9 ADC Single Channel Periodically

Step 2

- Enable **dedicated Core0**.

ADC1 x

▼ Dedicated Core0 Settings

Use Core ☒

Requested Sampling Time (us) 0.25 ≤ 1 ≤ 128.15

Calculated Sampling Time (us) 1

Core0 Configuration

Channel	Pins	Enable	Custom Name	Trigger Source	Interrupts	Signed	CMP0	CMP1	CMP2	CMP3
AN0	RA0	☐	Channel_AN0	None	☐	☐	☐	☐	☐	☐

Lab9 ADC Single Channel Periodically

Step 3

- Check the **Requested Sampling Time (us)** is **1** of **Dedicated Core0**.

ADC1 x

▼ Dedicated Core0 Settings

Use Core ☒

Requested Sampling Time (us) 0.25 <= 1 <= 128.15

Calculated Sampling Time (us) 1

Core0 Configuration

Channel	Pins	Enable	Custom Name	Trigger Source	Interrupts	Signed	CMP0	CMP1	CMP2	CMP3
AN0	RA0	☐	Channel_AN0	None	☐	☐	☐	☐	☐	☐

Lab9 ADC Single Channel Periodically

Step 4

- **Enable AN0** and Modify the **Custom Name** to "**VR1**".
- Select the **Trigger Source** to **Common Software Trigger** and **Enable Interrupt**.

ADC1 x

▼ Dedicated Core0 Settings

Use Core ☒

Requested Sampling Time (us) 0.25 <= 1 <= 128.15

Calculated Sampling Time (us) 1

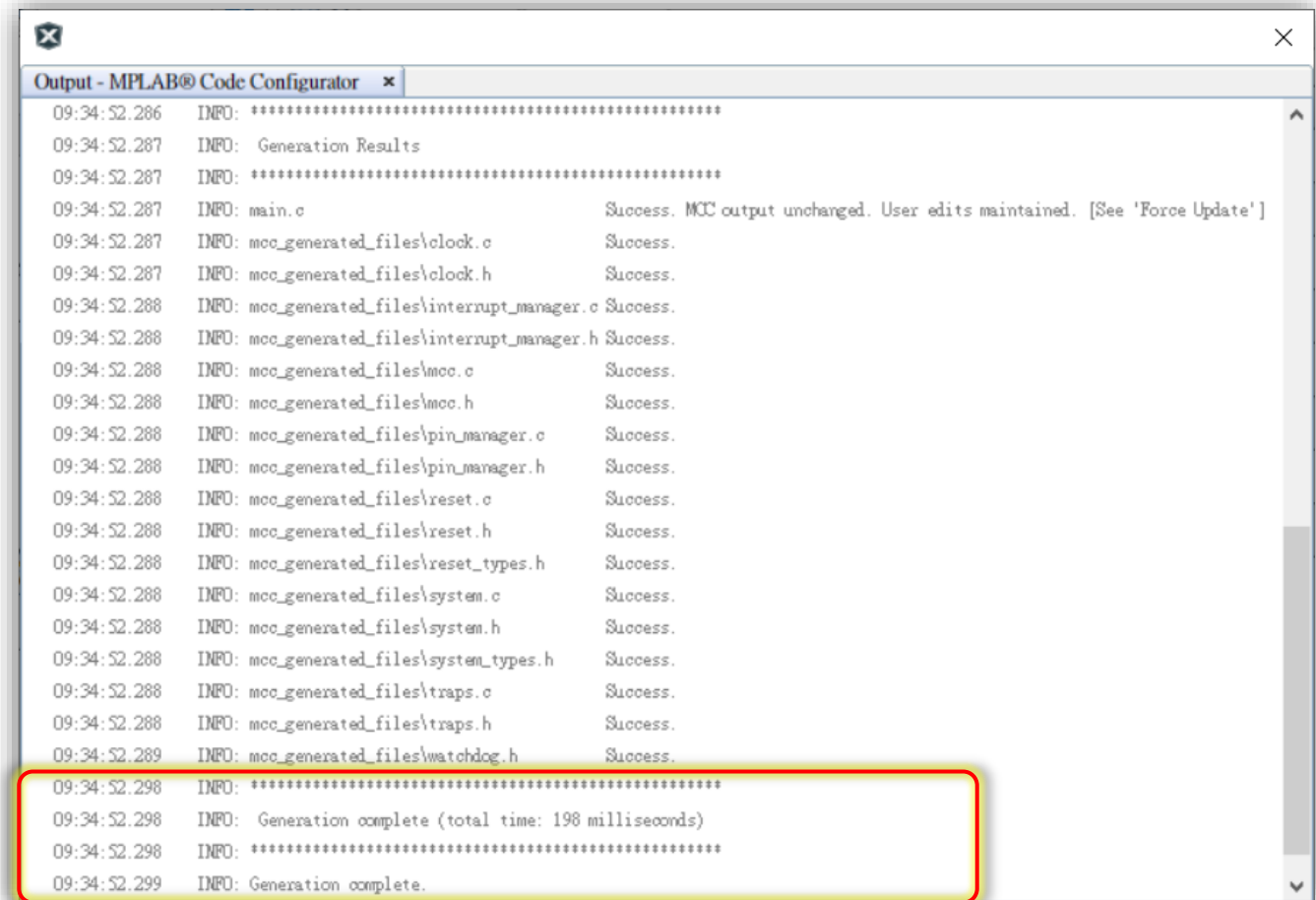
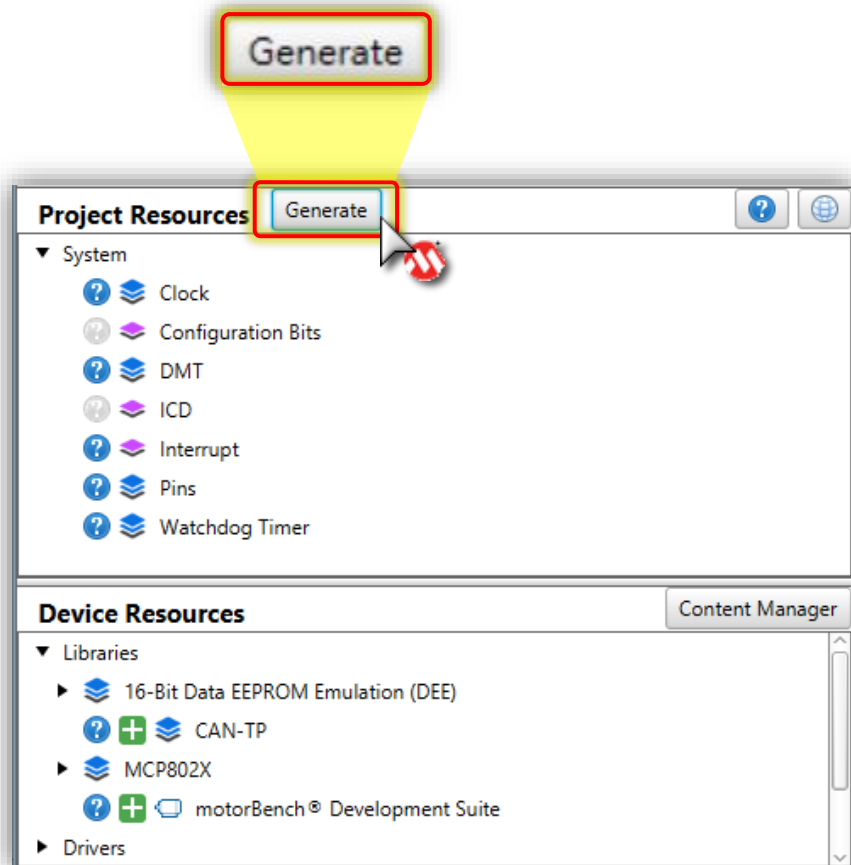
Core0 Configuration

Channel	Pins	Enable	Custom Name	Trigger Source	Interrupts	Signed	CMP0	CMP1	CMP2	CMP3
AN0	RA0	☒	VR1	Common Software	☒	☐	☐	☐	☐	☐

Lab9 ADC Single Channel Periodically

Step 5

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



Lab9 ADC Single Channel Periodically

Code Example

```
#include "myOLEDLIB/myOLED_ssd1306.h"
#include "mcc_generated_files/adc/adc1.h"
#include <stdio.h>

...
volatile uint8_t TMR1Flag = 0;
volatile uint8_t VR1Flag = 0;
volatile uint16_t ADC1Buffer[2];
uint8_t StringBuffer[40];

void myTimer1_Callback(void)
{
    TMR1Flag=1;
    LED1_Toggle();
    LED2_Toggle();
}

...
void ADC1_Callback(enum ADC_CHANNEL channel, uint16_t adcVal)
{
    if (channel == VR1)
    {
        VR1Flag = 1;
        ADC1Buffer[0] = adcVal;
    }
}

int main(void)
{
    ...
}
```

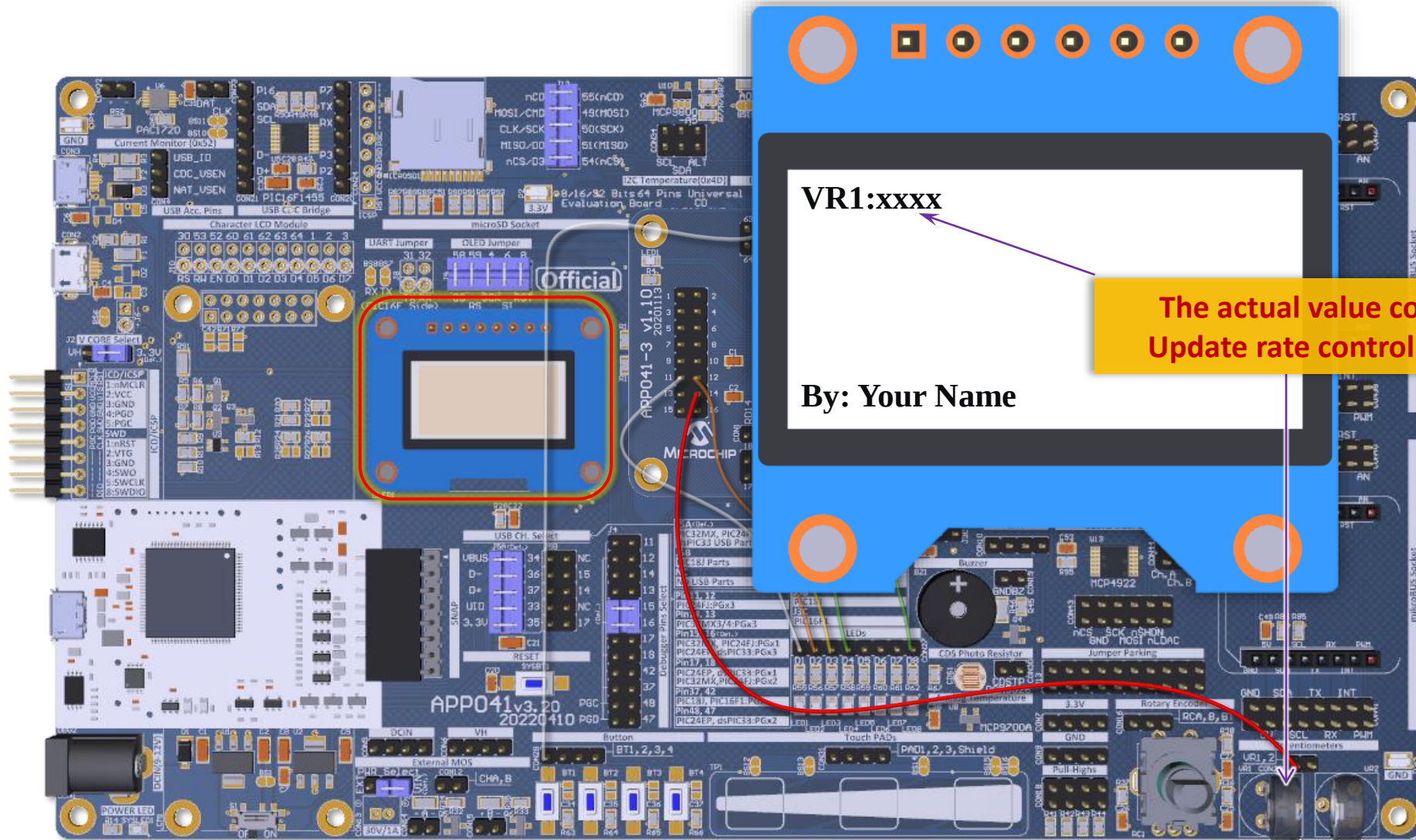
```
int main(void)
{
    SYSTEM_Initialize();
    ...
    myTimer2.TimeoutCallbackRegister(myTimer2_Callback);
    ADC1.adcMulticoreInterface->ChannelCallbackRegister(ADC1_Callback);
    ...

    while (1)
    {
        ...
        if(TMR1Flag)
        {
            TMR1Flag=0;
            ADC1.SoftwareTriggerEnable();
        }

        if (VR1Flag)
        {
            VR1Flag = 0;
            sprintf((char *) StringBuffer, "VR1:%4d", ADC1Buffer[0]);
            myOLED_DrawStr(0, 0, StringBuffer);
        }
    }
}
```

**myTimer1 Timer for ADC
Software Trigger periodically**

Result



Lab10 ADC Multiple Channel Periodically

- Base on current project or Open `.\Exercises\Lab10_xxx.X` to do exercise
- 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!

Connection



Lab10 ADC Multiple Channel Periodically

Step 1

- Check the **Requested Sampling Time (us)** is **1** of **Shared Core**.

ADC1 x

Easy View

Basic Settings

Custom Name: ADC1

Resolution: 12

Conversion Time (us): Refer PLIB for the Calculations. 2.31

Data Output Format: ☒ Integer ☐ Fractional

Common Interrupt: ☐

Pin Help: Pin Selection should only be done using the table. Channel Enable checkbox selects the corresponding analog channel and auto locks the analog pin in the Pins Grid View.

Shared Core Settings

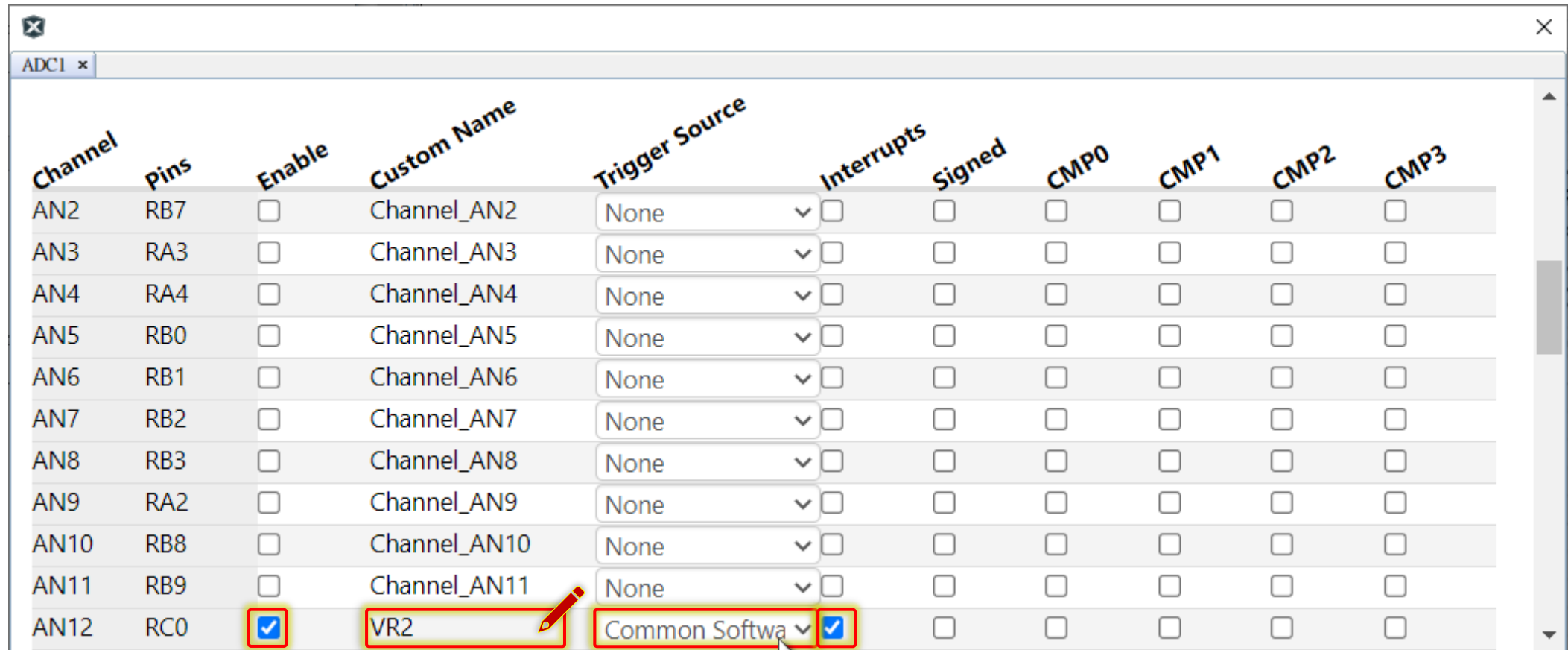
Requested Sampling Time (us): 0.25 <= 1 <= 128.15

Calculated Sampling Time (us): 1

Lab10 ADC Multiple Channel Periodically

Step 2

- **Enable AN12** and Modify the **Custom Name** to "**VR2**".
- Select the **Trigger Source** to **Common Software Trigger** and **Enable Interrupt**.



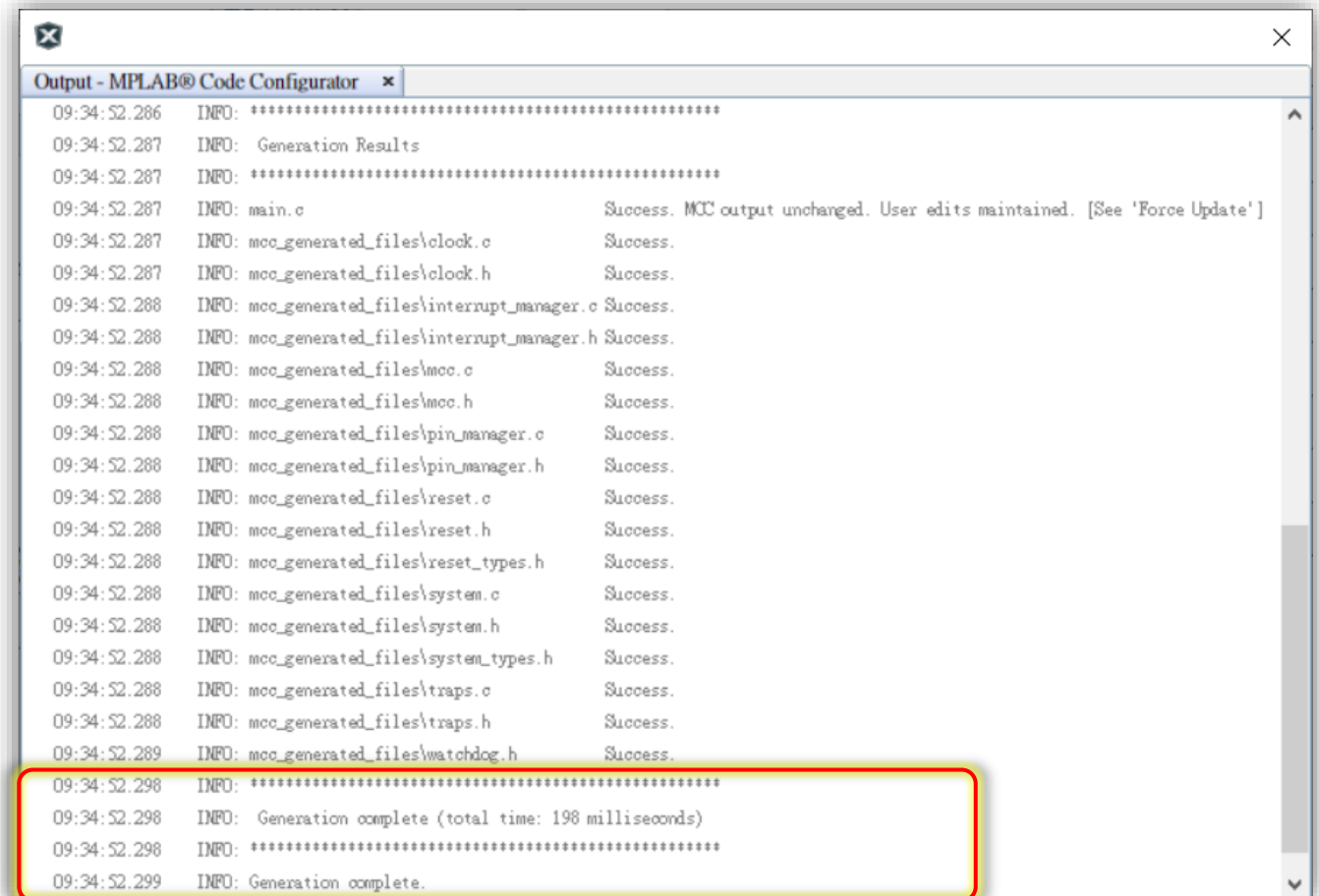
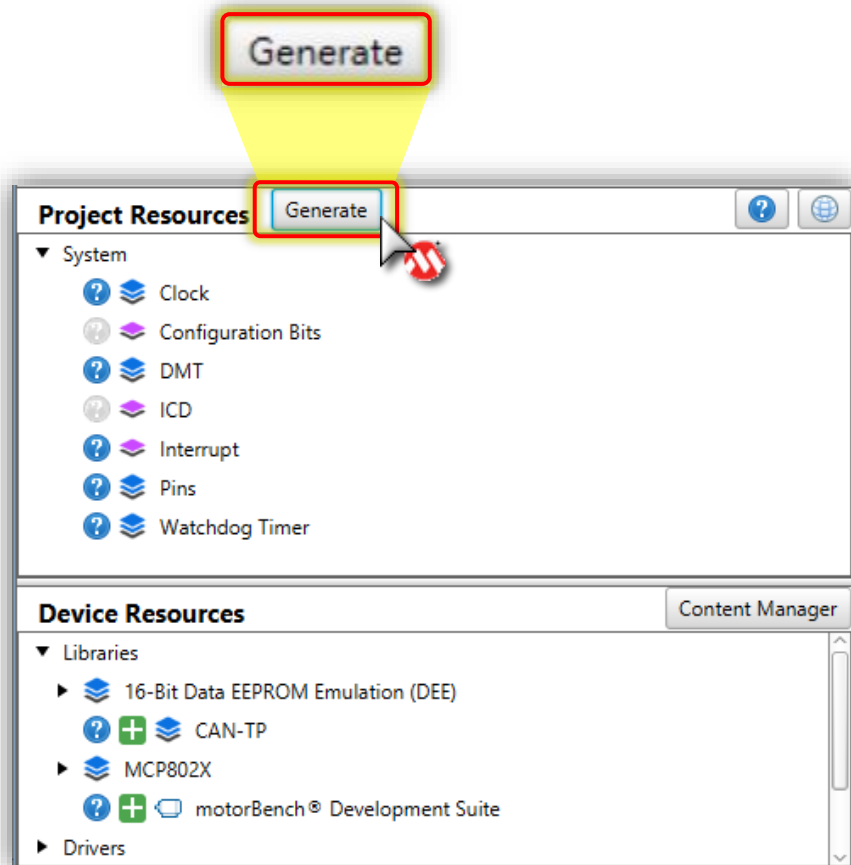
The screenshot shows the ADC1 configuration window with a table of channels. The row for AN12 is highlighted, and its configuration is as follows:

Channel	Pins	Enable	Custom Name	Trigger Source	Interrupts	Signed	CMP0	CMP1	CMP2	CMP3
AN2	RB7	☐	Channel_AN2	None	☐	☐	☐	☐	☐	☐
AN3	RA3	☐	Channel_AN3	None	☐	☐	☐	☐	☐	☐
AN4	RA4	☐	Channel_AN4	None	☐	☐	☐	☐	☐	☐
AN5	RB0	☐	Channel_AN5	None	☐	☐	☐	☐	☐	☐
AN6	RB1	☐	Channel_AN6	None	☐	☐	☐	☐	☐	☐
AN7	RB2	☐	Channel_AN7	None	☐	☐	☐	☐	☐	☐
AN8	RB3	☐	Channel_AN8	None	☐	☐	☐	☐	☐	☐
AN9	RA2	☐	Channel_AN9	None	☐	☐	☐	☐	☐	☐
AN10	RB8	☐	Channel_AN10	None	☐	☐	☐	☐	☐	☐
AN11	RB9	☐	Channel_AN11	None	☐	☐	☐	☐	☐	☐
AN12	RC0	☒	VR2	Common Software	☒	☐	☐	☐	☐	☐

Lab10 ADC Multiple Channel Periodically

Step 3

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



Lab10 ADC Multiple Channel Periodically

Code Example

```
...
volatile uint8_t TMR1Flag = 0;
volatile uint8_t VR1Flag = 0;
volatile uint16_t ADC1Buffer[2];
uint8_t StringBuffer[40];
volatile uint8_t VR2Flag = 0;

...
void ADC1_CallBack(enum ADC_CHANNEL channel, uint16_t adcVal)
{
    if (channel == VR1)
    {
        VR1Flag = 1;
        ADC1Buffer[0] = adcVal;
    }
    else if (channel == VR2)
    {
        VR2Flag = 1;
        ADC1Buffer[1] = adcVal;
    }
}

int main(void)
{
    ...
}
```

```
int main(void)
{
    ...
    ADC1.adcMulticoreInterface->ChannelCallbackRegister(ADC1_CallBack);
    ...

    while (1)
    {
        if(TMR1Flag)
        {
            TMR1Flag=0;
            ADC1.SoftwareTriggerEnable();
        }

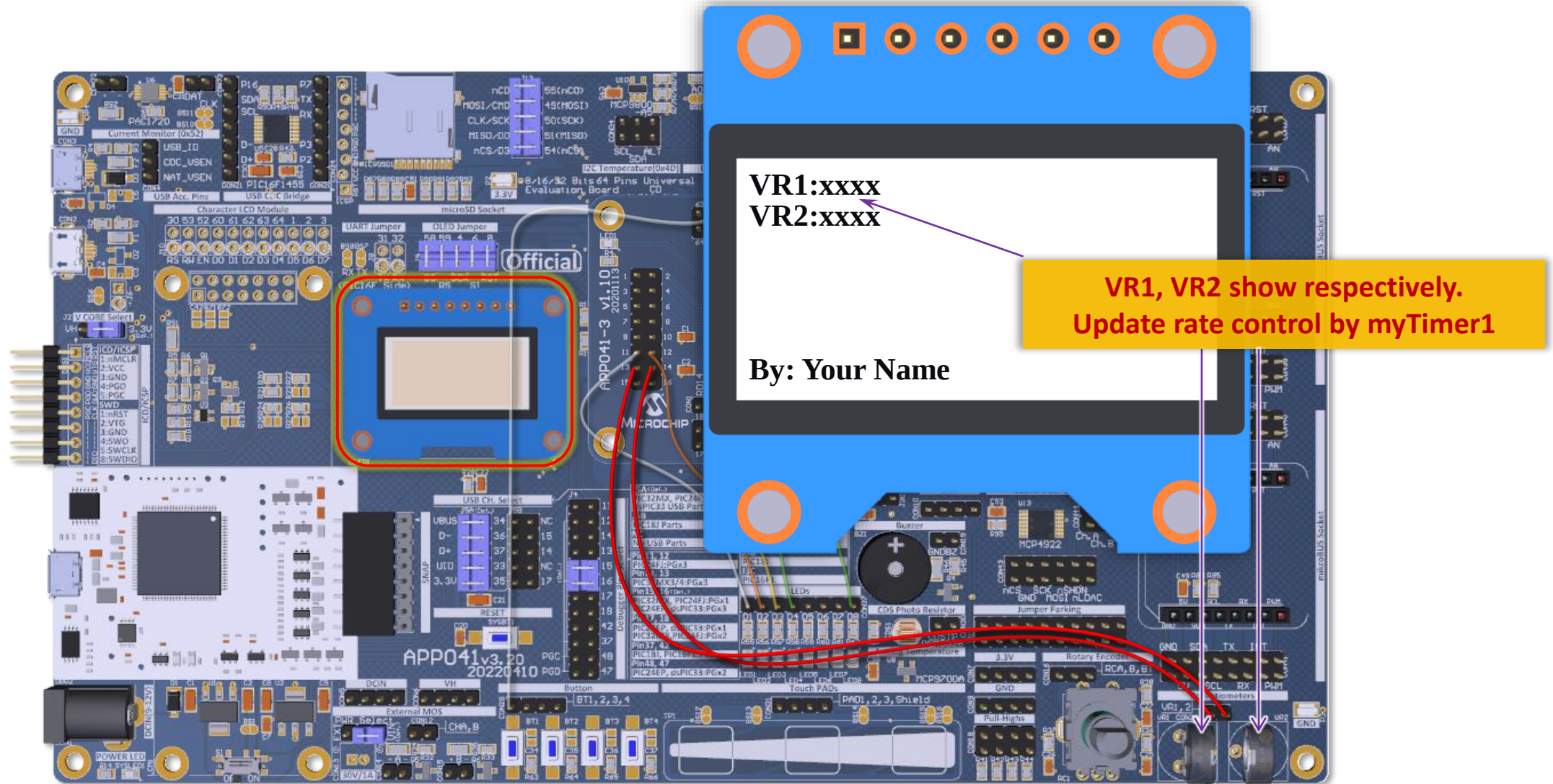
        if (VR1Flag)
        {
            VR1Flag = 0;
            sprintf((char *) StringBuffer, "VR1:%4d", ADC1Buffer[0]);
            myOLED_DrawStr(0, 0, StringBuffer);
        }

        if (VR2Flag)
        {
            VR2Flag = 0;
            sprintf((char *) StringBuffer, "VR2:%4d", ADC1Buffer[1]);
            myOLED_DrawStr(0, 1, StringBuffer);
        }
    }
}
```

**myTimer1 Timer for ADC
Software Trigger periodically**

Lab10 ADC Multiple Channel Periodically

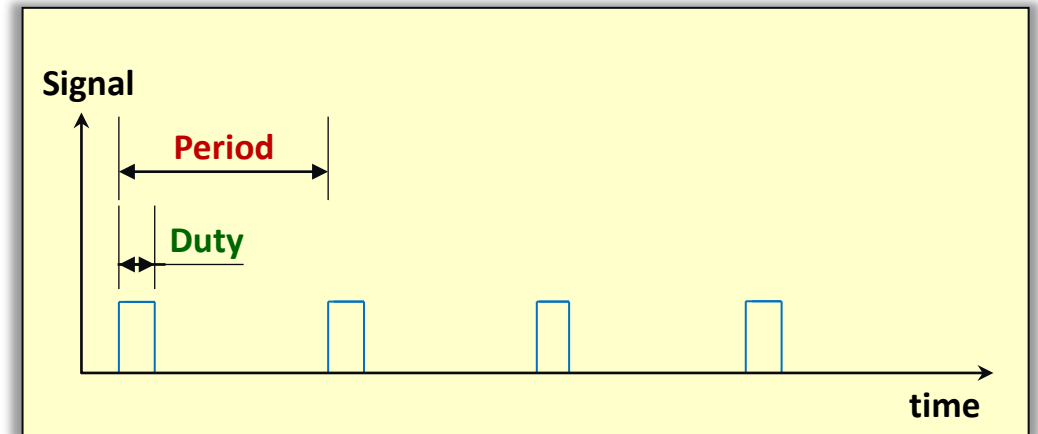
Result



High Resolution PWM Application

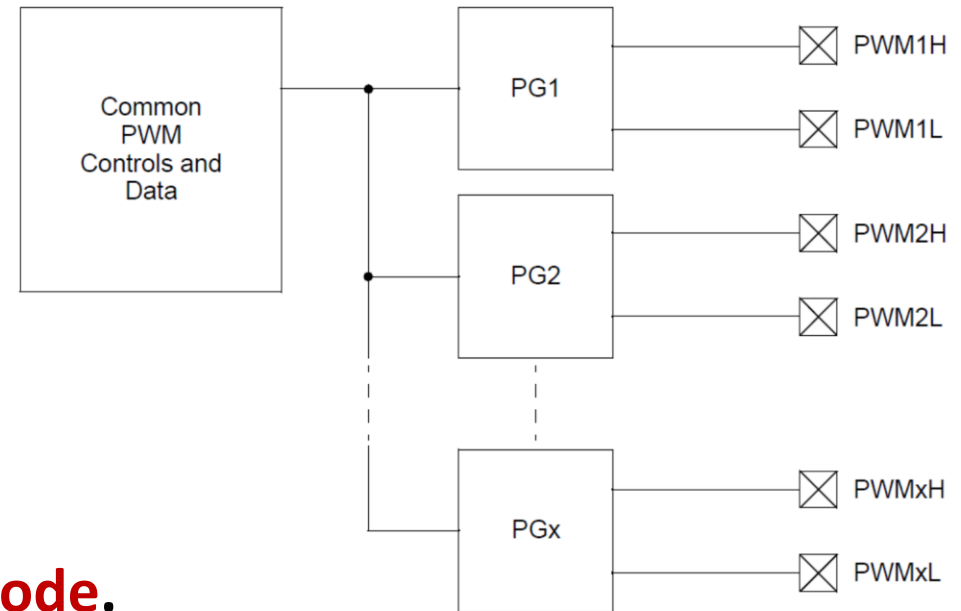
What's PWM

- 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 are two important terms
 - **Period :**
The single square wave duration.
 - **Duty :**
The proportion of active time to the regular interval.



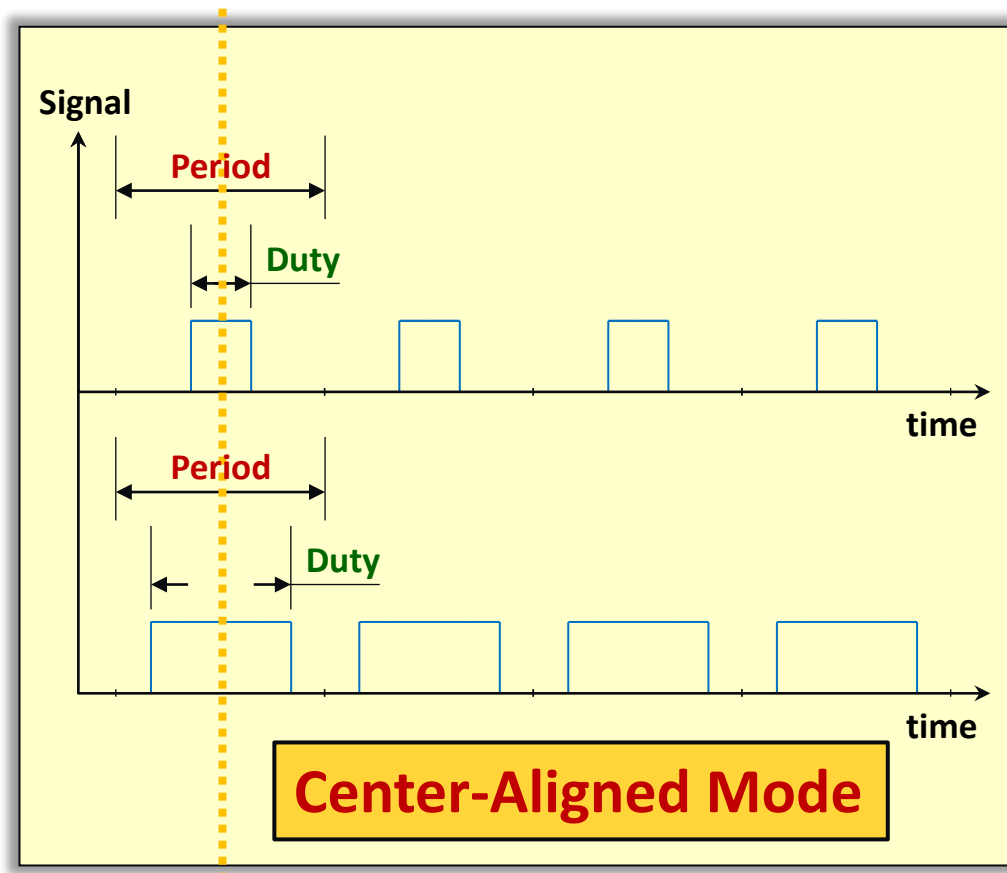
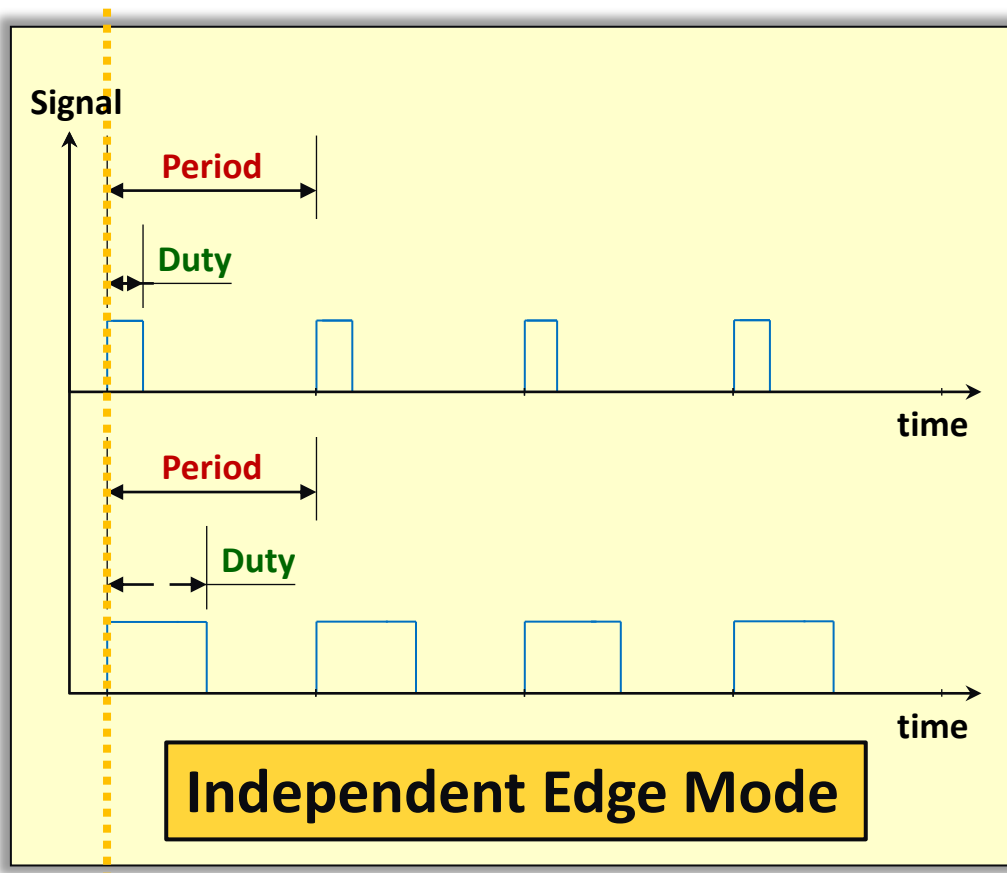
High Resolution PWM

- 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 **Center-Aligned mode.**



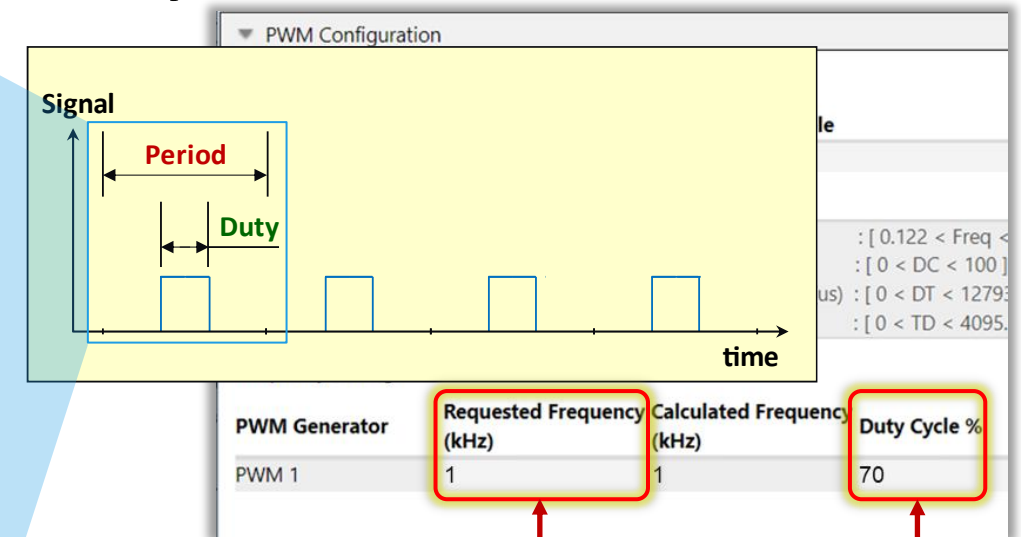
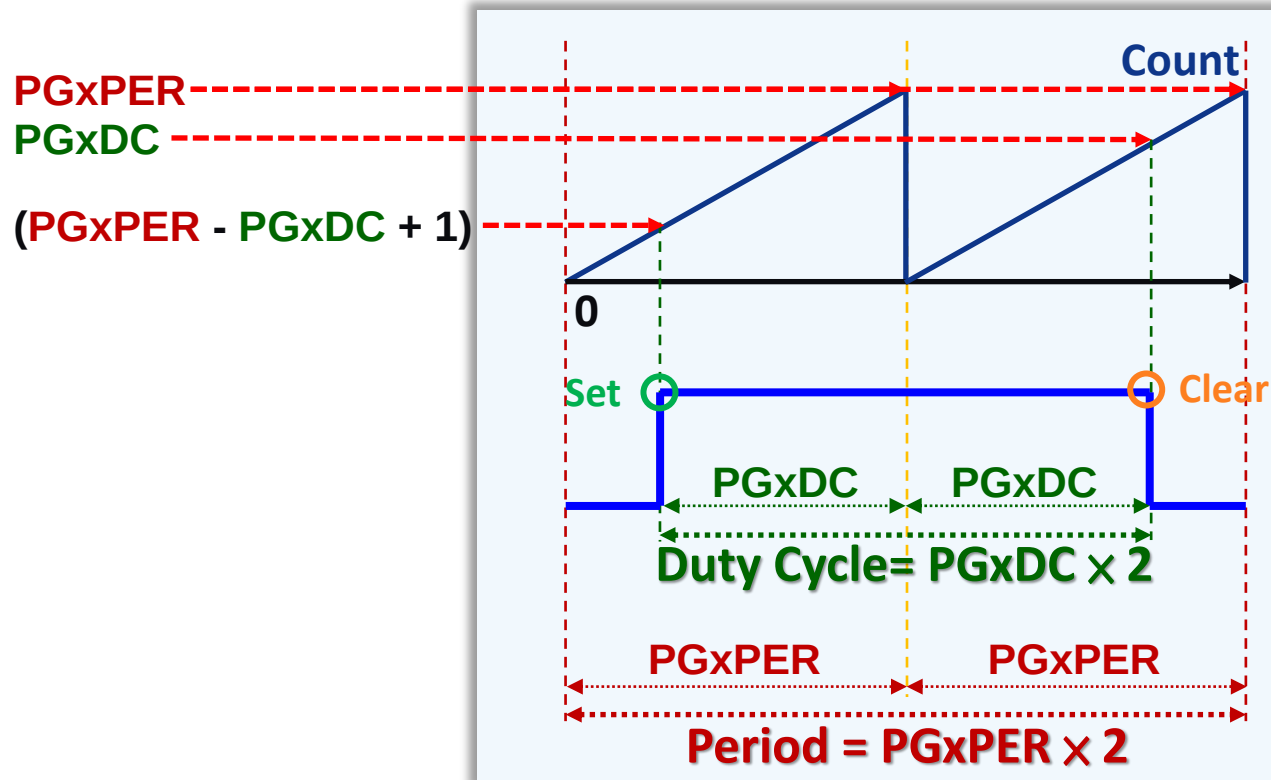
Independent Edge Mode vs Center-Aligned Mode

- Independent Edge Mode is a common PWM mode, which is aligned from the starting point of the cycle.
- Center-Aligned Mode aligns from the center of the cycle, currently MCC Melody only supports this mode.



Center-Aligned Mode of High Resolution PWM

- For **Center-Aligned Mode**, signals avoid coincident rising or falling edges between PWM Generators when the duty.
- **PGxDC** : Determines the width of the PWM pulse from the center of the two timer cycles.
- **PGxPER** : Determines the end of the PWM timer count cycle.



$$\text{Frequency} = 1 / \text{Period}$$

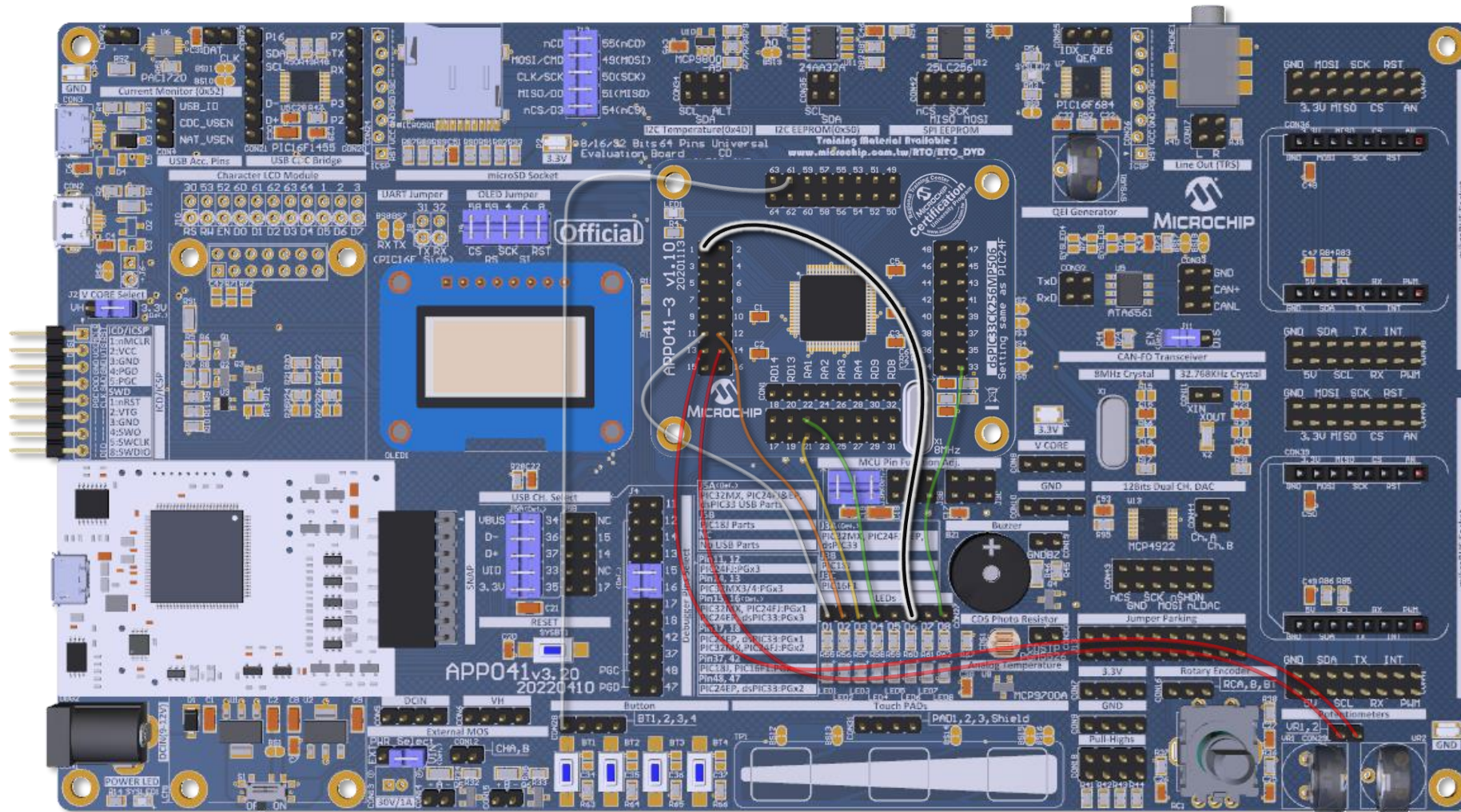
$$\text{Duty Cycle \%} = \text{Duty Cycle} / \text{Period}$$

Lab11 PWM Output

- Base on current project or Open `.\Exercises\Lab11_xxx.X` to do exercise
- Try to initial H.R. PWM-PG1 to Center-Aligned 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** (**LED6**) to observe LED brightness.

Let's go!

Connection

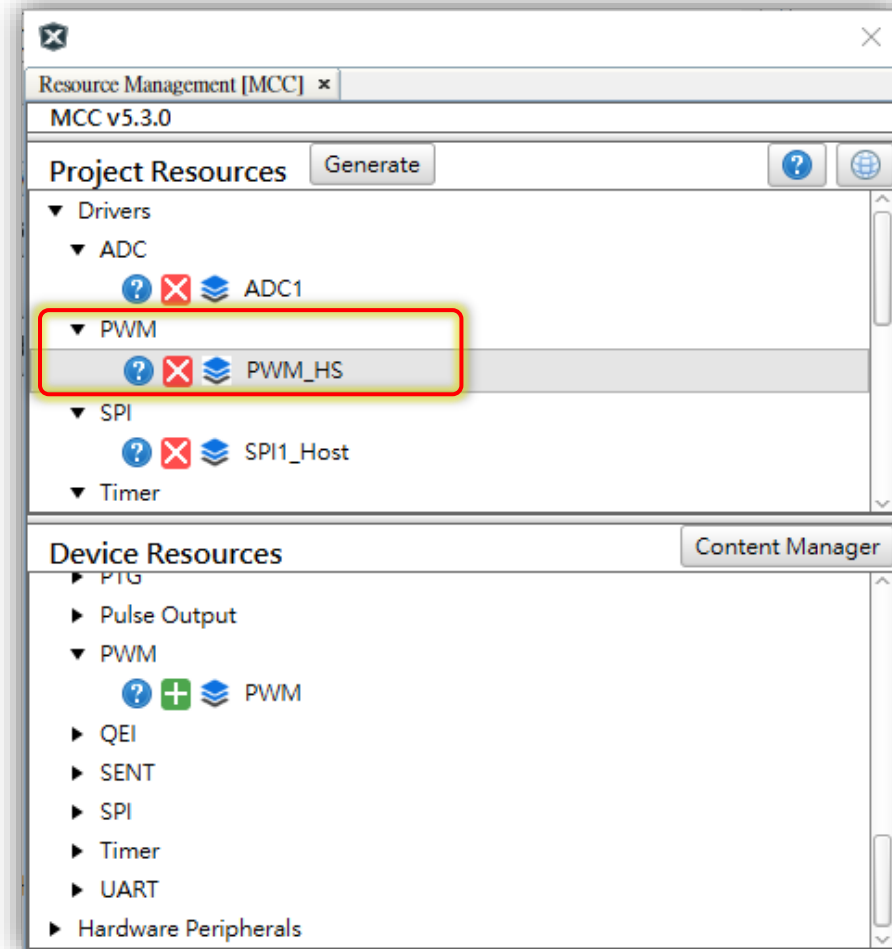
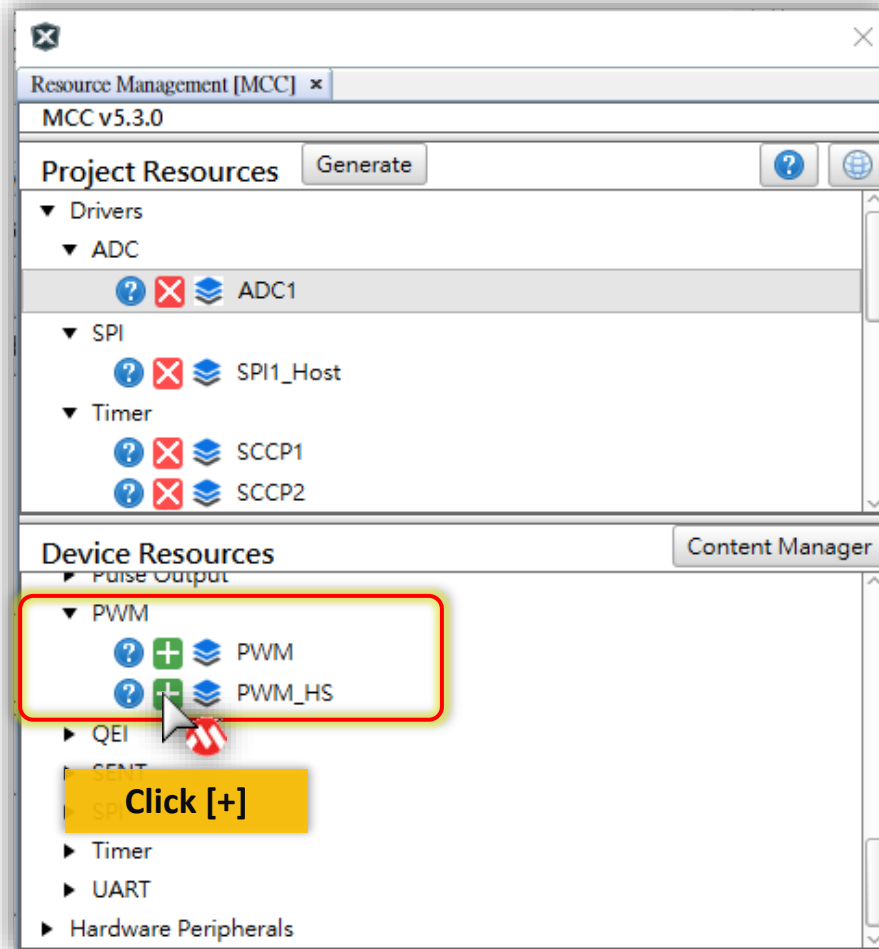


PIM Pin#		Symbol
1		D6

Lab11 PWM Output

Step 1

- Click **[+]** to add **PWM** > **PWM_HS** module.



Lab11 PWM Output

Step 2

- Enable **PWM 1**

PWM_HS - Editor

PWM_HS x

Easy View

Basic Configuration

Custom Name: PWM_HS

PWM Generators Required: 1

Interrupt Driven: ☐

PWM Configuration

Basic Settings

PWM Generator	Enable	Custom Name
PWM 1	☒	PWM_GENERATOR_1

Min and Max Limit

Frequency in (kHz) : [0.244 < Freq < 941.176]
Duty Cycle in (%) : [0 < DC < 100]
Dead Time Delay in (us) : [0 < DT < 63968750.000]
Trigger Delay in (us) : [0 < TD < 2047.969]

Frequency Settings

PWM Generator	Requested Frequency (kHz)	Calculated Frequency (kHz)	Duty Cycle %	Dead Time (us)	Operating Mode	Output Mode	Synchronous Update
PWM 1	941.176	941.176	0	0	Center-Aligned	Complementary	☐

Lab11 PWM Output

Note

- The PWM output pin is connected by default.

Pin Grid View x																													
Package:	TQFP64 ▾	Pin No:	14	15	16	17	18	28	29	33	34	35	45	46	47	48	49	61	62	63	64	1	2	13	22	23	27	50	51
			PORTA					PORTB																					
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5
SPI1_Host ▶	--	--																											
ADC1 ▶	--	--																											
	PWM1-H	output																											
	PWM1-L	output																											
	PCI10	input																											
	PCI11	input																											
	PCI12	input																											
	PCI13	input																											
	PCI14	input																											
PCI15	input																												

Lab11 PWM Output

Step 3

- Modify **PWM1 Requested Frequency (kHz)** to **1** and Set **PWM1 Duty Cycle (%)** to **20**.
- Check the **PWM1 Operating Mode** is "**Center-Aligned**".

The screenshot shows the 'PWM_HS - Editor' window with the 'Easy View' tab selected. The 'Basic Configuration' section includes fields for 'Custom Name' (PWM_HS), 'PWM Generators Required' (1), and 'Interrupt Driven' (disabled). The 'PWM Configuration' section includes 'Basic Settings' and 'Frequency Settings'. In the 'Basic Settings' table, 'PWM 1' is enabled with a custom name 'PWM_GENERATOR_1'. The 'Frequency Settings' table shows 'Requested Frequency (kHz)' set to 1, 'Calculated Frequency (kHz)' set to 1, 'Duty Cycle %' set to 20, 'Dead Time (us)' set to 0, 'Operating Mode' set to 'Center-Aligned', 'Output Mode' set to 'Complementary', and 'Synchronous Update' set to 'Off'. Red boxes and pencil icons highlight the 'Requested Frequency (kHz)', 'Duty Cycle %', and 'Operating Mode' fields.

PWM Generator	Enable	Custom Name
PWM 1	☒	PWM_GENERATOR_1

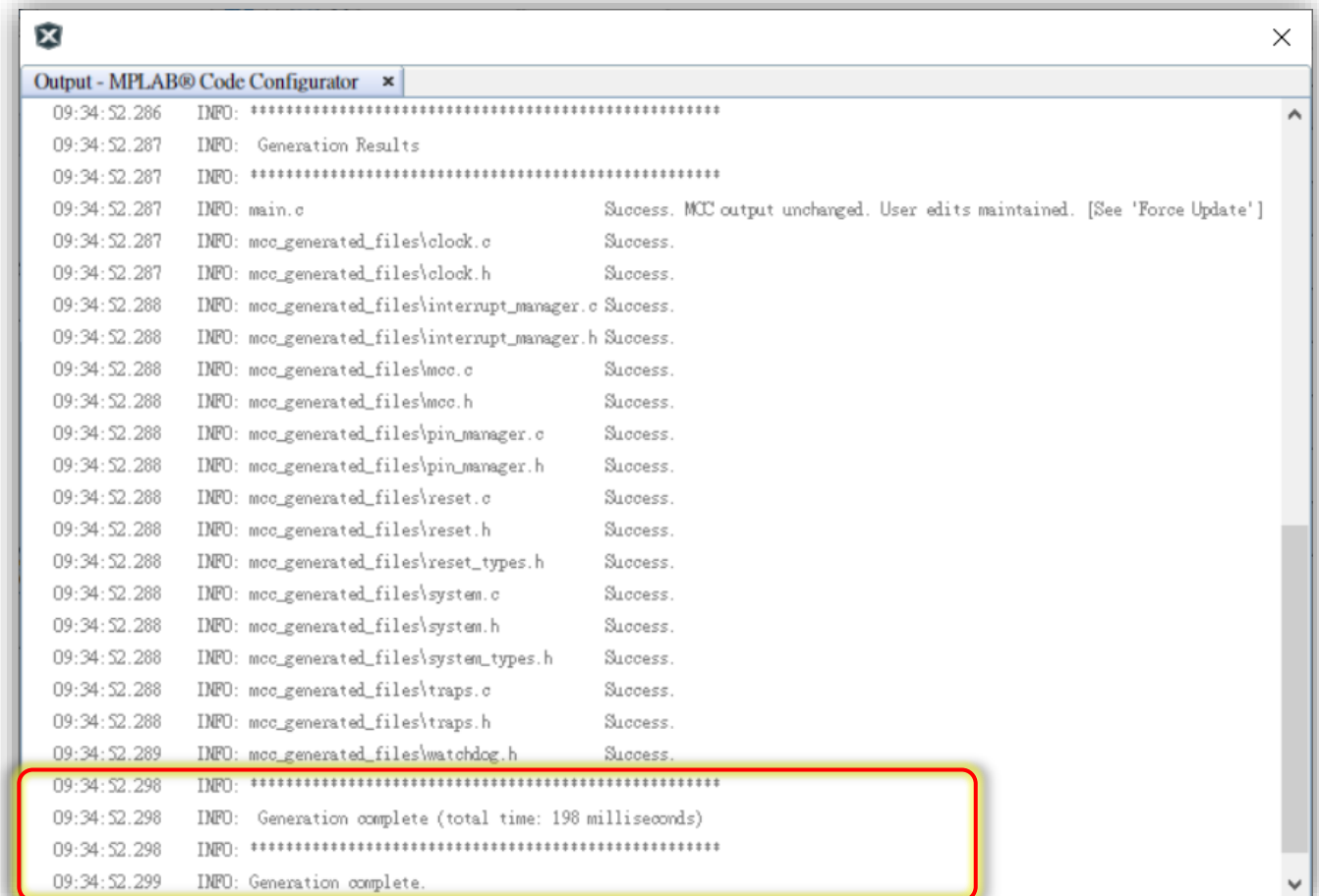
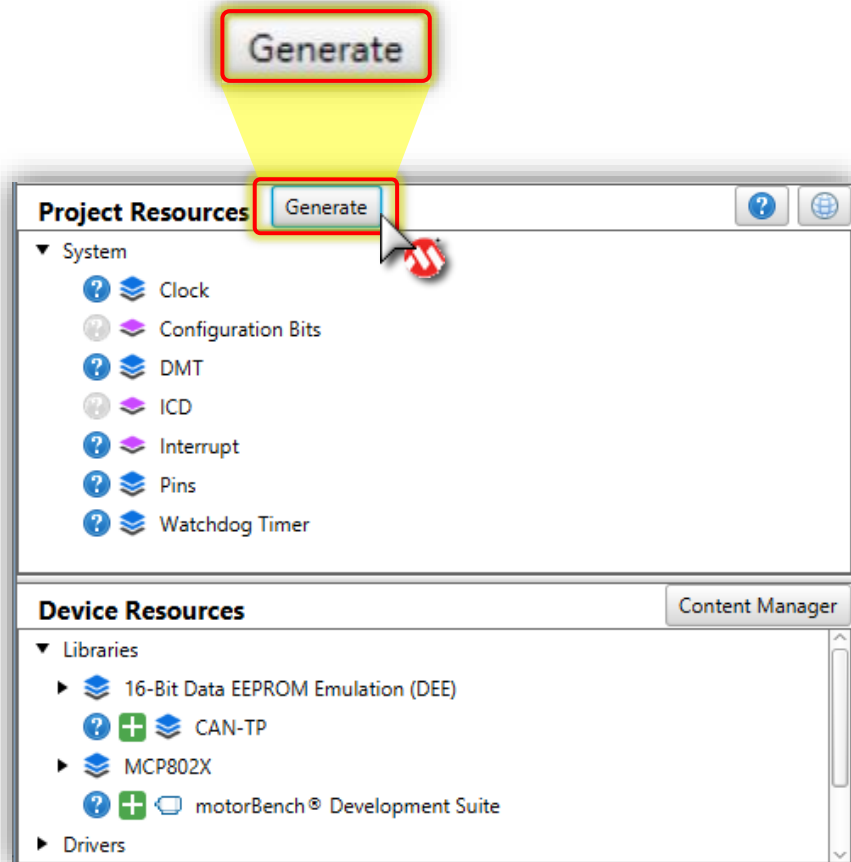
Min and Max Limit	Frequency in (kHz)	Duty Cycle in (%)	Dead Time Delay in (us)	Trigger Delay in (us)
	[0.244 < Freq < 941.176]	[0 < DC < 100]	[0 < DT < 63968750.000]	[0 < TD < 2047.969]

PWM Generator	Requested Frequency (kHz)	Calculated Frequency (kHz)	Duty Cycle %	Dead Time (us)	Operating Mode	Output Mode	Synchronous Update
PWM 1	1	1	20	0	Center-Aligned	Complementary	☐

Lab11 PWM Output

Step 4

- 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;
}
```

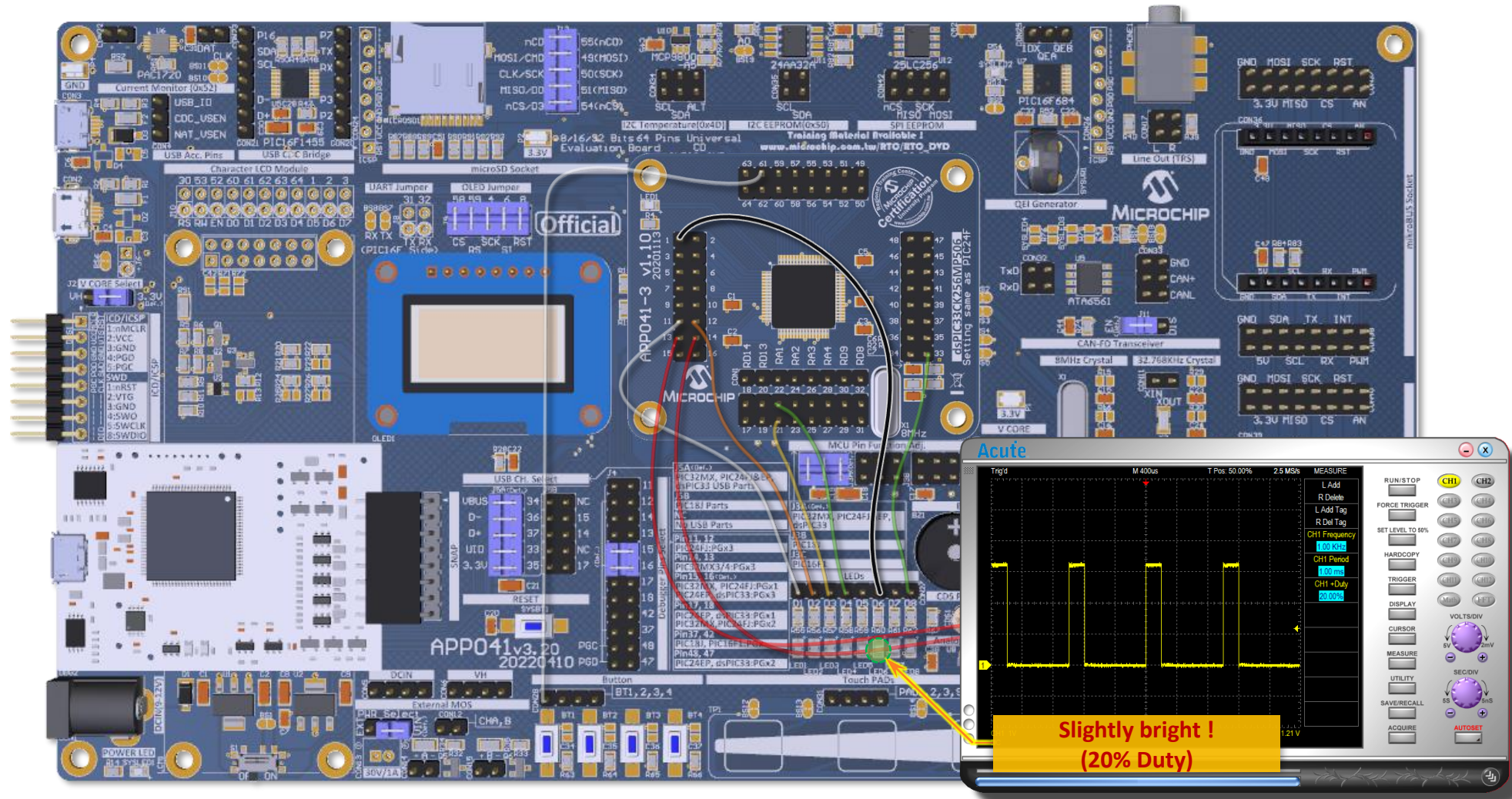
```
void ADC1_VR2_Callback (uint16_t adcVal)
{
    VR2Flag = 1;
    ADC1_VR2_Buffer = adcVal;
}
```

Don't Need Modify Any Code !

```
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;
}
```

Lab11 PWM Output

Result



H.R. PWM Functions of Melody

- Melody Provide below functions for PWM:

Prototype definition : "pwm.h"

- PWM_HS_INTERFACE

- (void) *PWM_HS*.Initialize (void);
- (void) *PWM_HS*.Deinitialize (void);
- (void) *PWM_HS*.Disable(void);
- (void) *PWM_HS*.Enable(void);
- (void) *PWM_HS*.GeneratorDisable(enum PWM_GENERATOR *genNum*);
- (void) *PWM_HS*.GeneratorEnable(enum PWM_GENERATOR *genNum*);
- (void) *PWM_HS*.ModeSet(enum PWM_GENERATOR *genNum*, enum PWM_MODES *mode*);
- (void) *PWM_HS*.PeriodSet(enum PWM_GENERATOR *genNum*,uint16_t *period*);
- (void) *PWM_HS*.DutyCycleSet(enum PWM_GENERATOR *genNum*,uint16_t *dutyCycle*);
- (void) *PWM_HS*.PhaseSelect(enum PWM_GENERATOR *genNum*, enum PWM_SOURCE_SELECT *source*);
- (void) *PWM_HS*.PhaseSet(enum PWM_GENERATOR *genNum*, uint16_t *phase*);
- (void) *PWM_HS*.SoftwareUpdateRequest(enum PWM_GENERATOR *genNum*);
- (void) *PWM_HS*.GeneratorEOCEventCallbackRegister(void (*callback)(...)); /* End Of Cycle */

Lab12 PWM Duty Adj By ADC

- Base on current project or Open .\Exercises\Lab12_xxx.X to do exercise
- Try to use **Potentiometer (VR1)** to control **LED (LED6)** brightness.
- VR Value (0 ~4095) <-> PWM Duty (0% ~ 100%)
- H.R. PWM Setting same as previous Lab.

Let's go!

Connection



Lab12 PWM Duty Adj By ADC

Notice



The screenshot shows the 'PWM_HS' configuration window in the MPLAB IDE. A large yellow starburst is overlaid on the window, containing the text 'Don't Need Modify Any MCC Melody Setting !' in red. The window is divided into sections: 'Basic Configuration' and 'PWM Configuration'. Under 'Basic Configuration', 'Custom Name' is 'PWM_HS', 'PWM Generators Required' is '1', and 'Interrupt Driven' is checked. Under 'PWM Configuration', 'Basic Settings' shows 'PWM Generator' as 'PWM 1' and 'Enable' is checked. 'Min and Max Limit' shows 'Frequency In (kHz)' as '1', 'Duty Cycle In (%)' as '50', 'Dead Time Delay In (us)' as '0', and 'Trigger Delay In (us)' as '0'. 'Frequency Settings' shows a table with columns: 'PWM Generator', 'Requested Frequency (kHz)', 'Calculated Frequency (kHz)', 'Duty Cycle %', 'Dead Time (us)', 'Operating Mode', 'Output Mode', and 'Synchronous Update'. The table has one row for 'PWM 1' with values: '1', '1', '50', '0', 'Center-Aligned', 'Complementary', and 'Synchronous Update' is unchecked. 'Output Control Settings' is at the bottom.

Don't Need Modify Any MCC Melody Setting !

PWM Generator	Requested Frequency (kHz)	Calculated Frequency (kHz)	Duty Cycle %	Dead Time (us)	Operating Mode	Output Mode	Synchronous Update
PWM 1	1	1	50	0	Center-Aligned	Complementary	☐

Lab12 PWM Duty Adj By ADC

Code Example

```
...
#include <stdio.h>
#include "mcc_generated_files/pwm_hs/pwm.h"
...

int main(void)
{
    ...

    while (1)
    {
        ...

        if (VR1Flag)
        {
            VR1Flag = 0;
            sprintf((char *) StringBuffer, "VR1:%4d", ADC1Buffer[0]);
            myOLED_DrawStr(0, 0, StringBuffer);

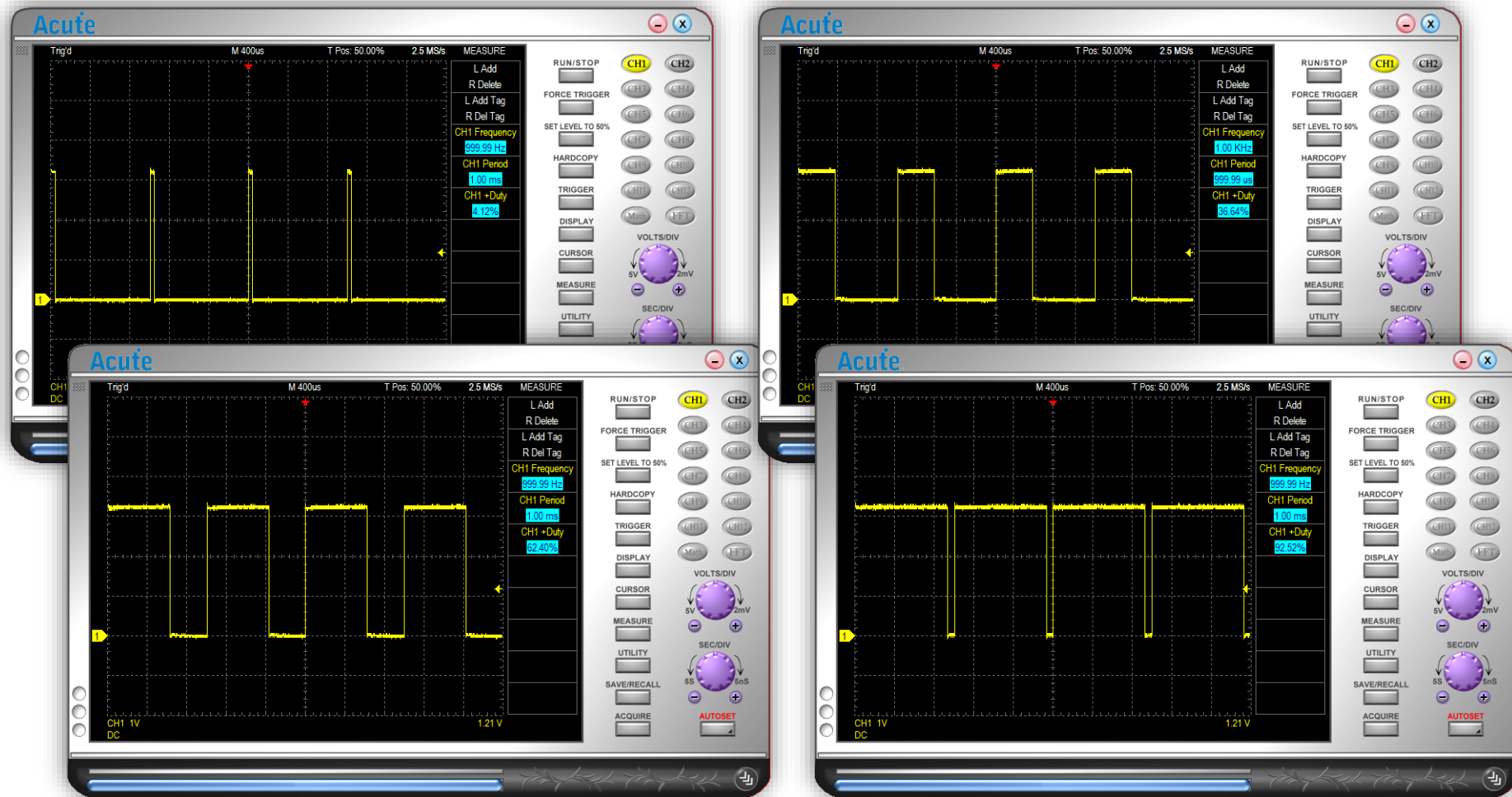
            PWM_HS.DutyCycleSet(PWM_GENERATOR_1, ( (uint32_t) ADC1Buffer[0] * PG1PER) / 4095);
            PWM_HS.SoftwareUpdateRequest(PWM_GENERATOR_1);
        }
    }
}
```

Result



Lab12 PWM Duty Adj By ADC

Result

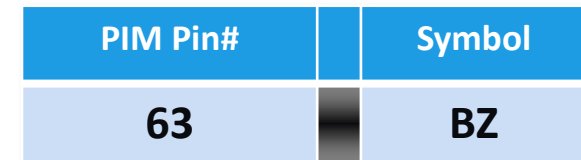


Lab13 PWM Buzzer Tone

- Base on current project or Open .\Exercises\Lab13_xxx.X to do exercise
- Try to initial H.R. PWM - PG2 to output special frequency (Tone) to Buzzer by VR2.
- VR2 Value (0 ~4,095) <-> PWM Frequency (122.07Hz ~ 1kHz)
- Connect **RB12** (**PWM2_H**, **Pin63**) to **Buzzer (BZ)** to produce sound.

Let's go!

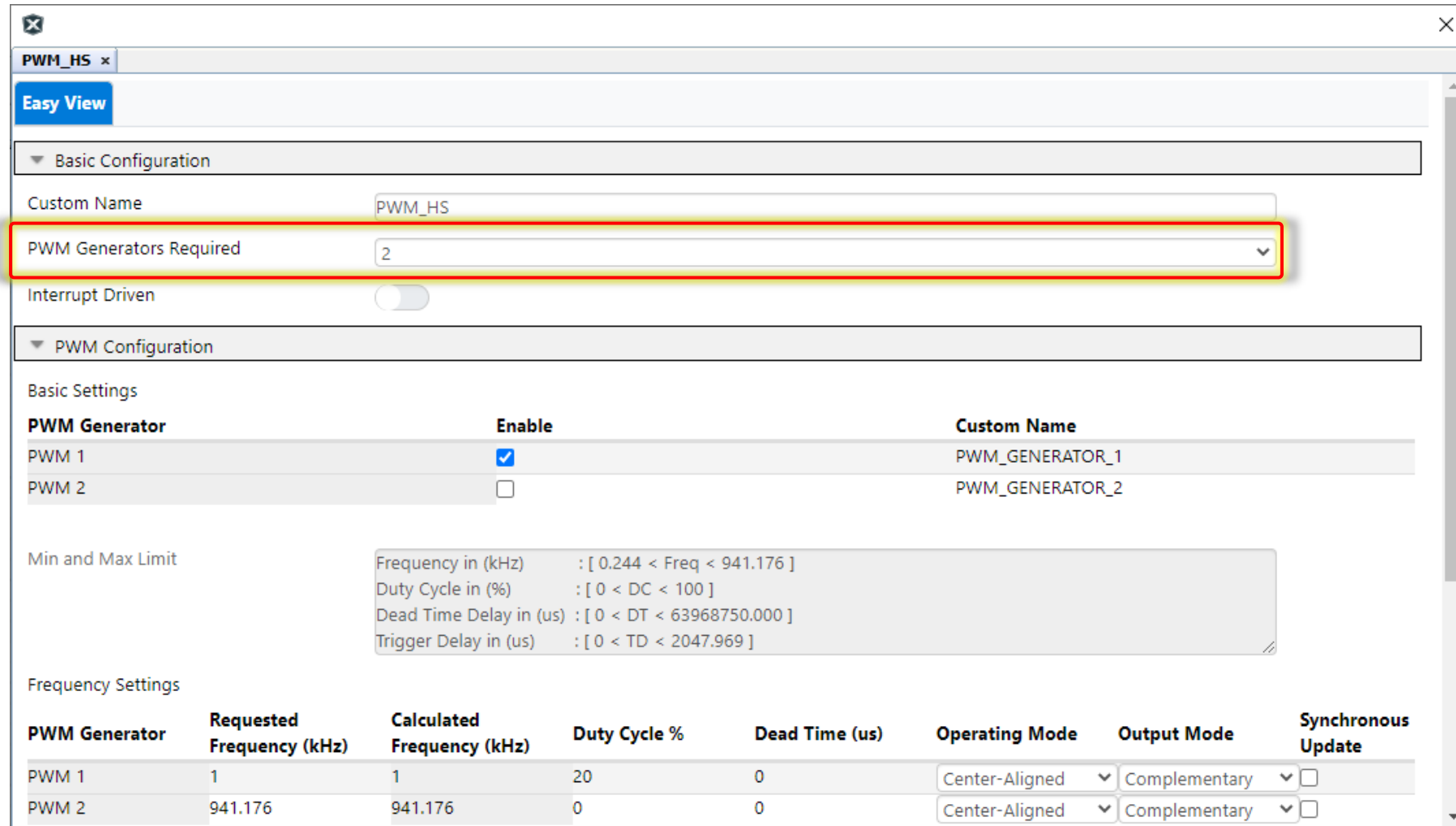
Connection



Lab13 PWM Buzzer Tone

Step 1

- Select **PWM Generators Required**: 1 → 2.



The screenshot shows the 'PWM_HS' configuration window. The 'Basic Configuration' section has a 'Custom Name' field set to 'PWM_HS' and a 'PWM Generators Required' dropdown menu highlighted with a red box, set to '2'. The 'Interrupt Driven' toggle is off. The 'PWM Configuration' section includes 'Basic Settings' and 'Frequency Settings'.

Basic Settings

PWM Generator	Enable	Custom Name
PWM 1	☒	PWM_GENERATOR_1
PWM 2	☐	PWM_GENERATOR_2

Min and Max Limit

Frequency in (kHz)	: [0.244 < Freq < 941.176]
Duty Cycle in (%)	: [0 < DC < 100]
Dead Time Delay in (us)	: [0 < DT < 63968750.000]
Trigger Delay in (us)	: [0 < TD < 2047.969]

Frequency Settings

PWM Generator	Requested Frequency (kHz)	Calculated Frequency (kHz)	Duty Cycle %	Dead Time (us)	Operating Mode	Output Mode	Synchronous Update
PWM 1	1	1	20	0	Center-Aligned	Complementary	☐
PWM 2	941.176	941.176	0	0	Center-Aligned	Complementary	☐

Lab13 PWM Buzzer Tone

Step 2

- Enable **PWM 2**.

PWM_HS x

Easy View

▼ Basic Configuration

Custom Name: PWM_HS

PWM Generators Required: 2

Interrupt Driven: ☐

▼ PWM Configuration

Basic Settings

PWM Generator	Enable	Custom Name
PWM 1	☒	PWM_GENERATOR_1
PWM 2	☒	PWM_GENERATOR_2

Min and Max Limit

Frequency in (kHz) : [0.244 < Freq < 941.176]
Duty Cycle in (%) : [0 < DC < 100]
Dead Time Delay in (us) : [0 < DT < 63968750.000]
Trigger Delay in (us) : [0 < TD < 2047.969]

Frequency Settings

PWM Generator	Requested Frequency (kHz)	Calculated Frequency (kHz)	Duty Cycle %	Dead Time (us)	Operating Mode	Output Mode	Synchronous Update
PWM 1	1	1	20	0	Center-Aligned	Complementary	☐
PWM 2	941.176	941.176	0	0	Center-Aligned	Complementary	☐

Lab13 PWM Buzzer Tone

Step 3

- Modify **PWM 2 Requested Frequency (kHz)** to **1** and Set **Duty Cycle (%)** to **50**.
- Check the **PWM 2 Operating Mode** is "**Center-Aligned**".

Easy View

Basic Configuration

Custom Name

PWM_HS

PWM Generators Required

2

Interrupt Driven

☐

PWM Configuration

Basic Settings

PWM Generator	Enable	Custom Name
PWM 1	☒	PWM_GENERATOR_1
PWM 2	☒	PWM_GENERATOR_2

Min and Max Limit

Frequency in (kHz) : [0.244 < Freq < 941.176]

Duty Cycle in (%) : [0 < DC < 100]

Dead Time Delay in (us) : [0 < DT < 63968750.000]

Trigger Delay in (us) : [0 < TD < 2047.969]

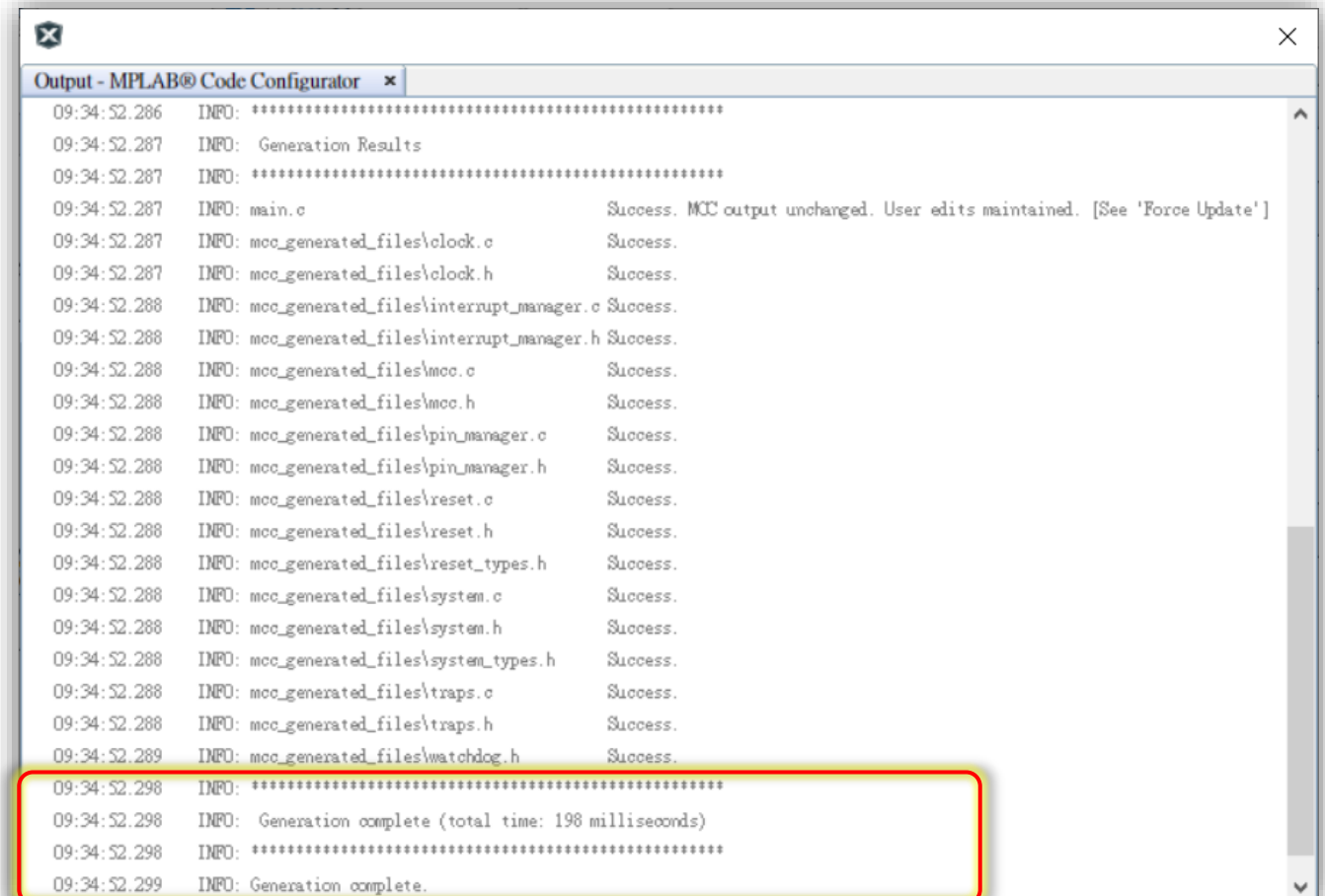
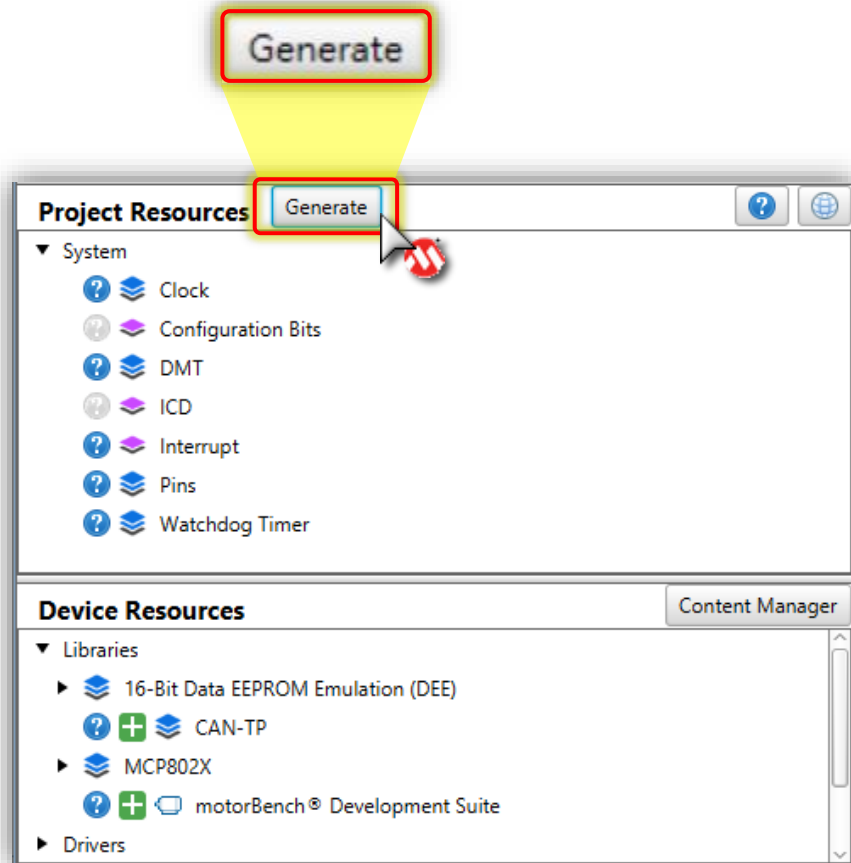
Frequency Settings

PWM Generator	Requested Frequency (kHz)	Calculated Frequency (kHz)	Duty Cycle %	Dead Time (us)	Operating Mode	Output Mode	Synchronous Update
PWM 1	1	1	20	0	Center-Aligned	Complementary	☐
PWM 2	1	1	50	0	Center-Aligned	Complementary	☐

Lab13 PWM Buzzer Tone

Step 4

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



Lab13 PWM Buzzer Tone

Code Example

- VR2 Value(0 ~ 4095) <-> PWM Frequency (122.07Hz ~ 1kHz)
 - PWM Frequency (**244.14Hz**) : **PGxPER** = **65535** (maximum count of 16-bits register)
 - PWM Frequency (**1kHz**) : **PGxPER** = **15999**

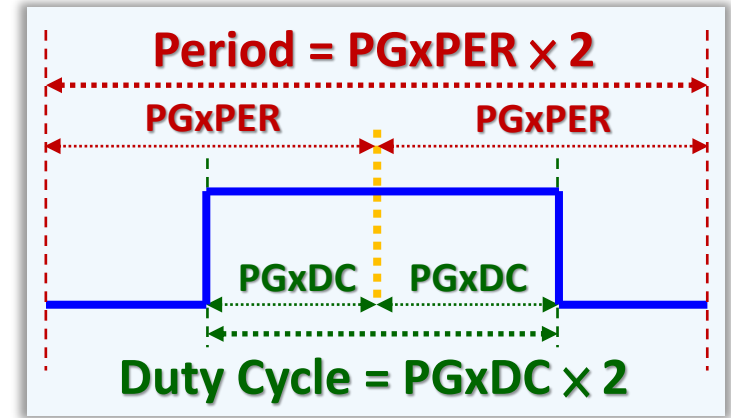
```
#include <stdio.h>
#include "mcc_generated_files/pwm_hs/pwm.h"
...

int main(void)
{
    ...

    while (1)
    {
        ...

        if (VR2Flag)
        {
            VR2Flag = 0;
            sprintf((char *) StringBuffer, "VR2:%4d", ADC1Buffer[1]);
            myOLED_DrawStr(0, 2, StringBuffer);

            PWM_HS.PeriodSet(PWM_GENERATOR_2, 65535 - (uint32_t)ADC1Buffer[1] * (65535 - 15999) / 4095);
            PWM_HS.DutyCycleSet(PWM_GENERATOR_2, PG2PER / 2); //Keep duty cycle at 50%
            PWM_HS.SoftwareUpdateRequest(PWM_GENERATOR_2);
        }
    }
}
```



Center-Aligned Modes

$$F_{PWM} = \left(\frac{F_{PGx_clk}}{2 \times (\text{PGxPER} + 1)} \right)$$

$$244.14\text{Hz} = \left(\frac{32\text{MHz}}{2 \times (65535 + 1)} \right)$$

$$\text{PGxPER} = \left(\frac{F_{PGx_clk}}{2 \times F_{PWM}} \right) - 1$$

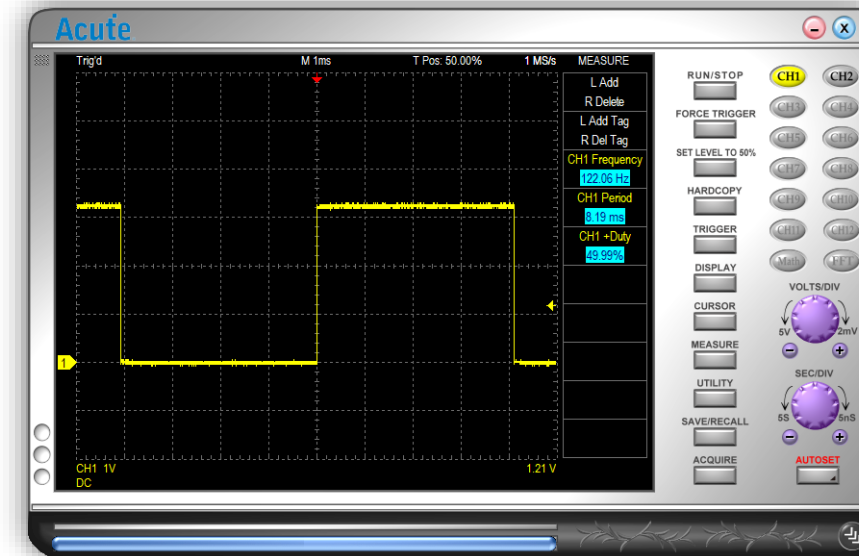
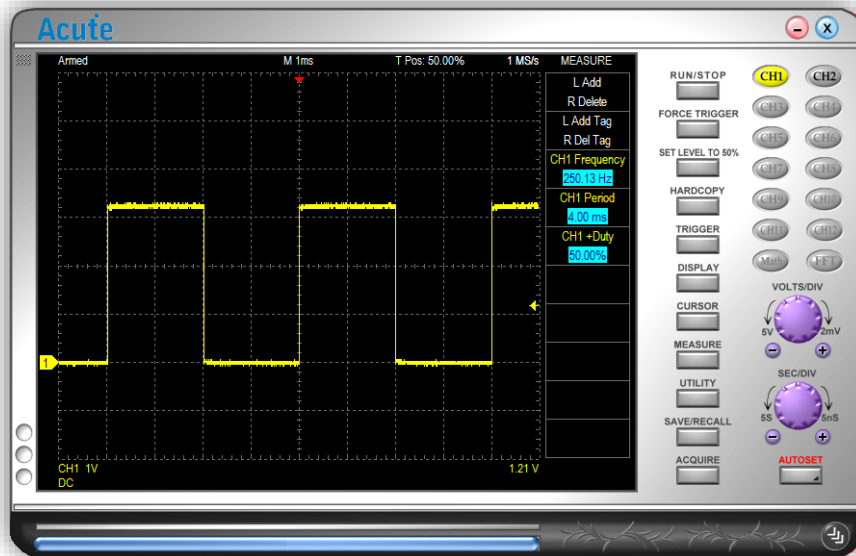
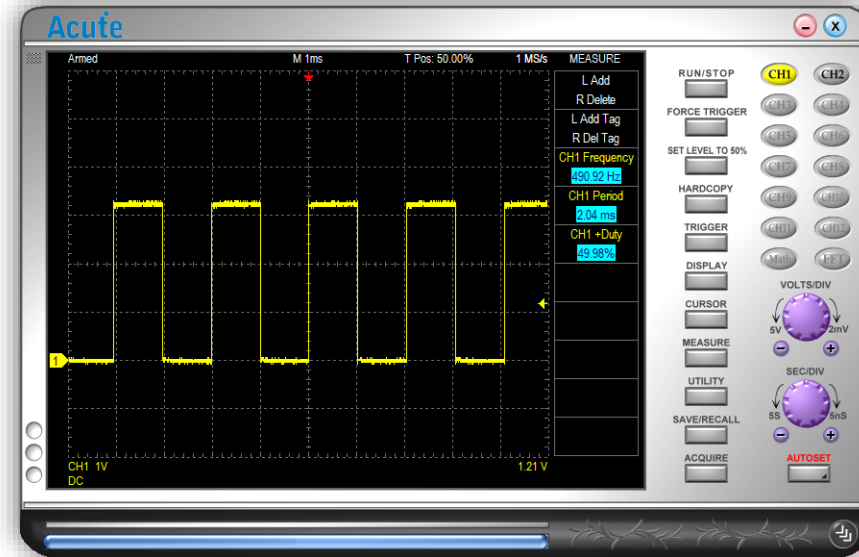
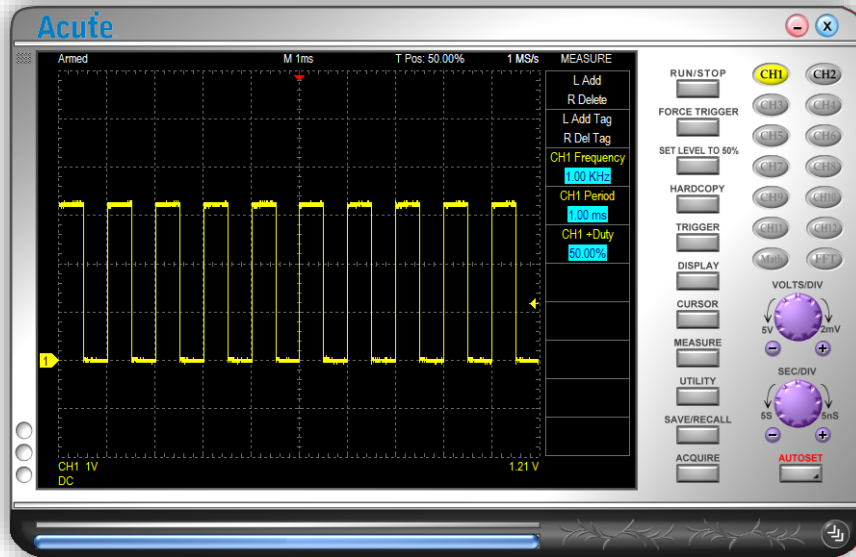
$$15999 = \left(\frac{32\text{MHz}}{2 \times 1\text{kHz}} \right) - 1$$

Result



Lab13 PWM Buzzer Tone

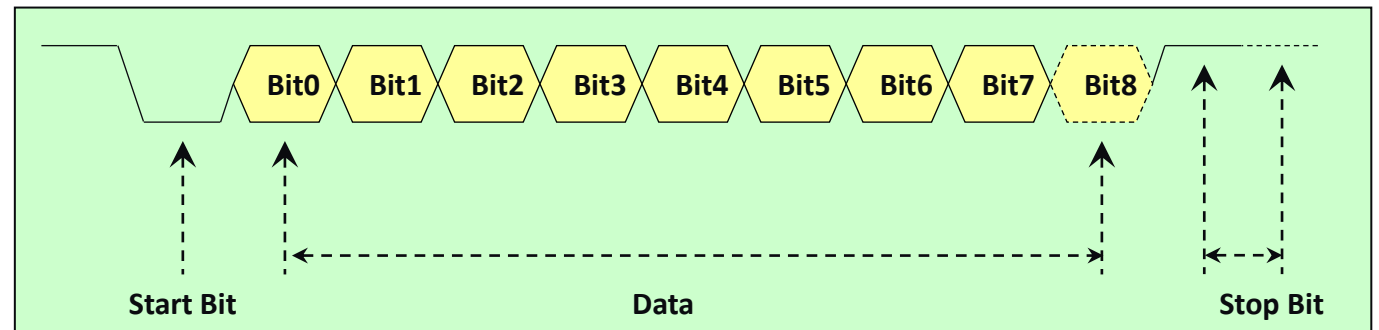
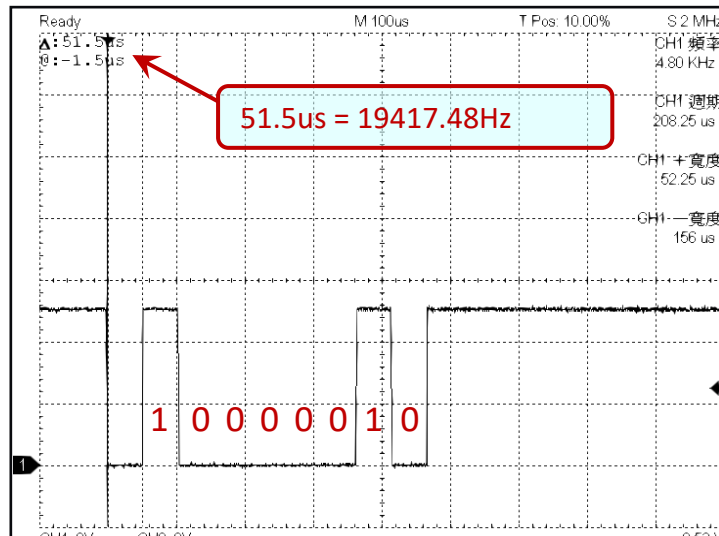
Result



UART Application

What's UART

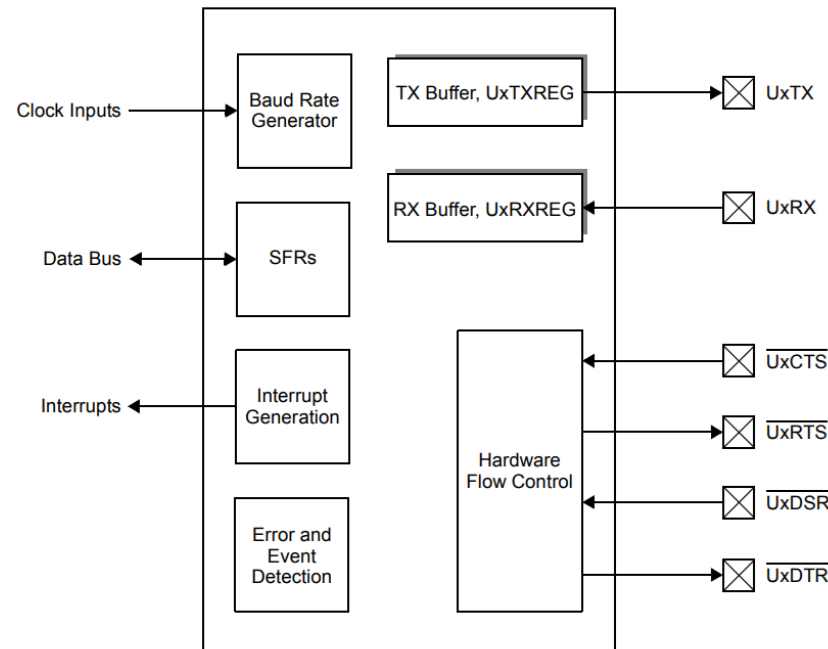
- UART : Universal Asynchronous Receiver Transmitter.
- The module data exchange through serial communication.
- The data formatting includes 1 start bit, 5 ~ 9 data bits and 1 ~ 2 stop bits.



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).



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!!");**

Redirect Printf to the UART

Redirect Printf to UART



Ensure Redirect to Printf is enabled for only one UART driver.

UART1 x

Easy View

Configuration Settings

Custom Name

Requested Baudrate

Calculated Baudrate

Baud Rate Error (%)

Parity

Data Size

Stop Bits

Flow Control Mode

Redirect Printf to UART ☐ Ensure Redirect to Printf is enabled for only one UART driver.

Interrupt Settings

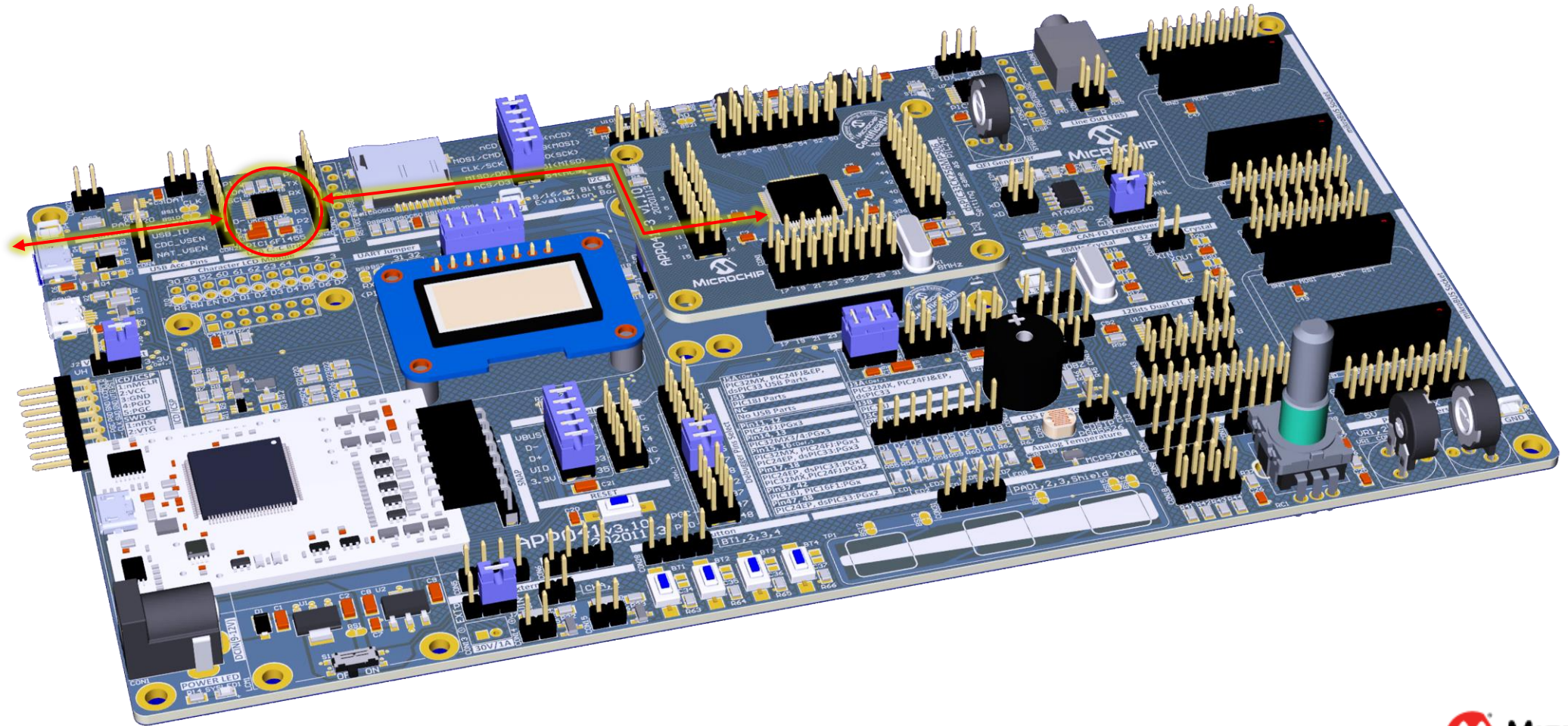
Interrupt Driven ☐

Dependency Selector

UART PLIB Selector

USB CDC Emulator

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



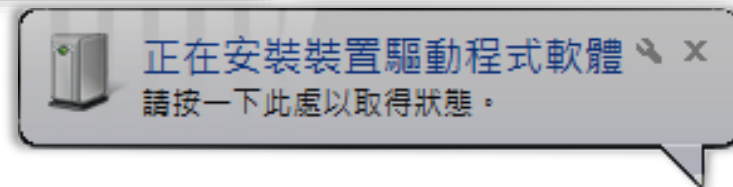
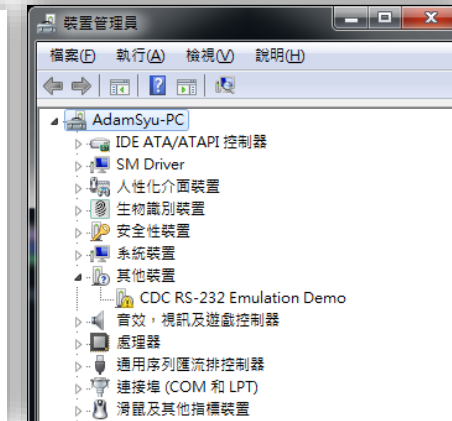
Lab Preparation

PC Terminal Software Setting

Lab Preparation

Step 1

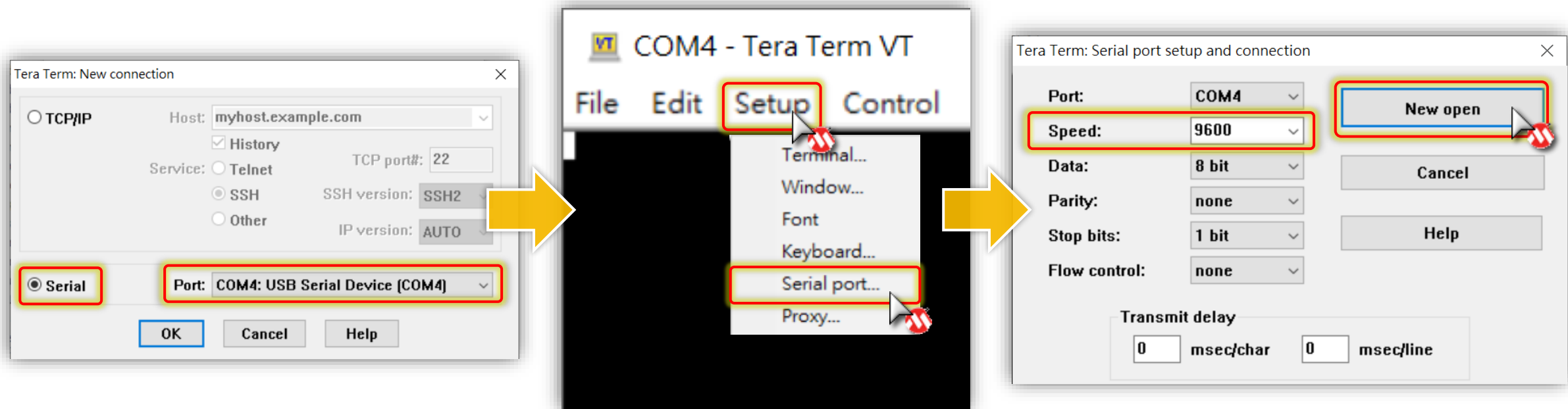
- Please confirm Windows Device Manager on your PC.
- For Window 7, you might need install driver before Labs.
 - Driver at : [\Appendix\Serial Emulator Driver\mchpcdc.inf](#).
- You will see a **USB Serial Port (COMx)** at Device Manager.



Lab Preparation

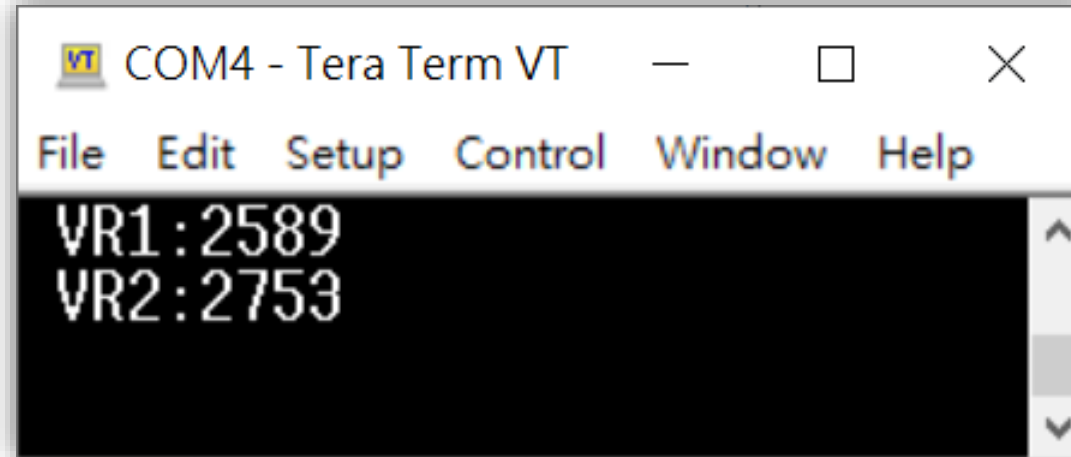
Step 2

- Find TeraTerm from your desktop or install it from.
 - <http://ttssh2.sourceforge.jp/index.html.en>
- Select: Serial → Port: COMx (USB Serial Port).
- 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



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

Let's go!

Lab14 UART printf

- Base on current project or Open `.\Exercises\Lab14_xxx.X` to do exercise
- 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!

Lab14 UART printf

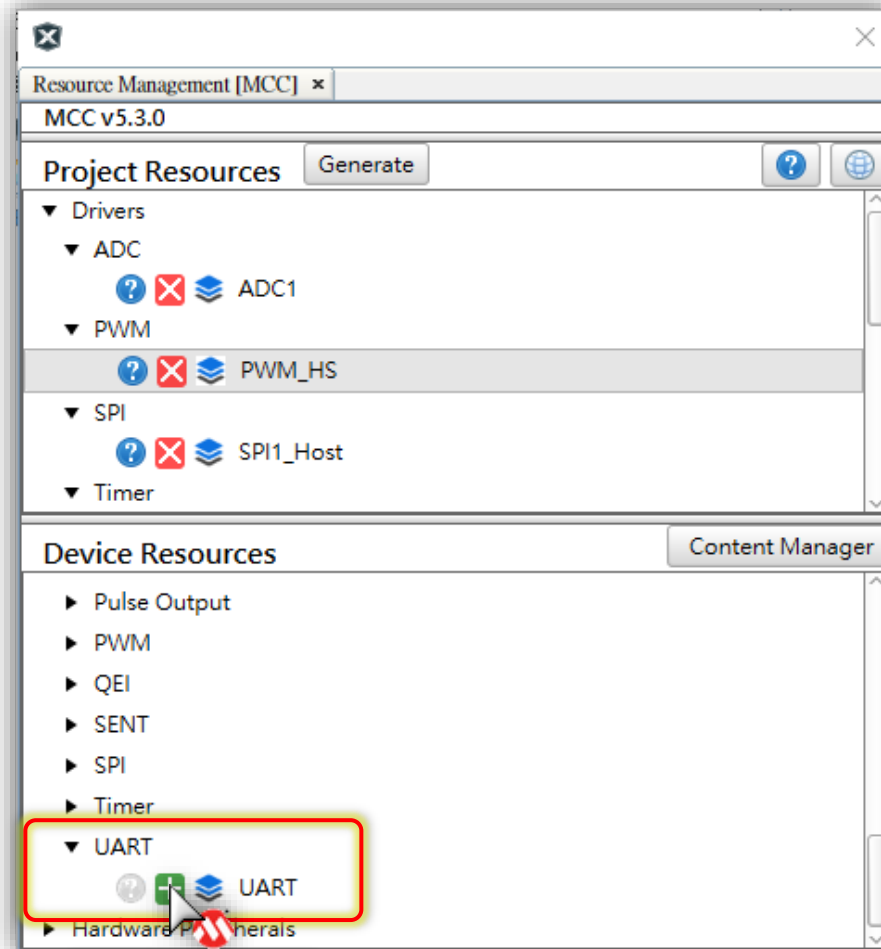
Connection



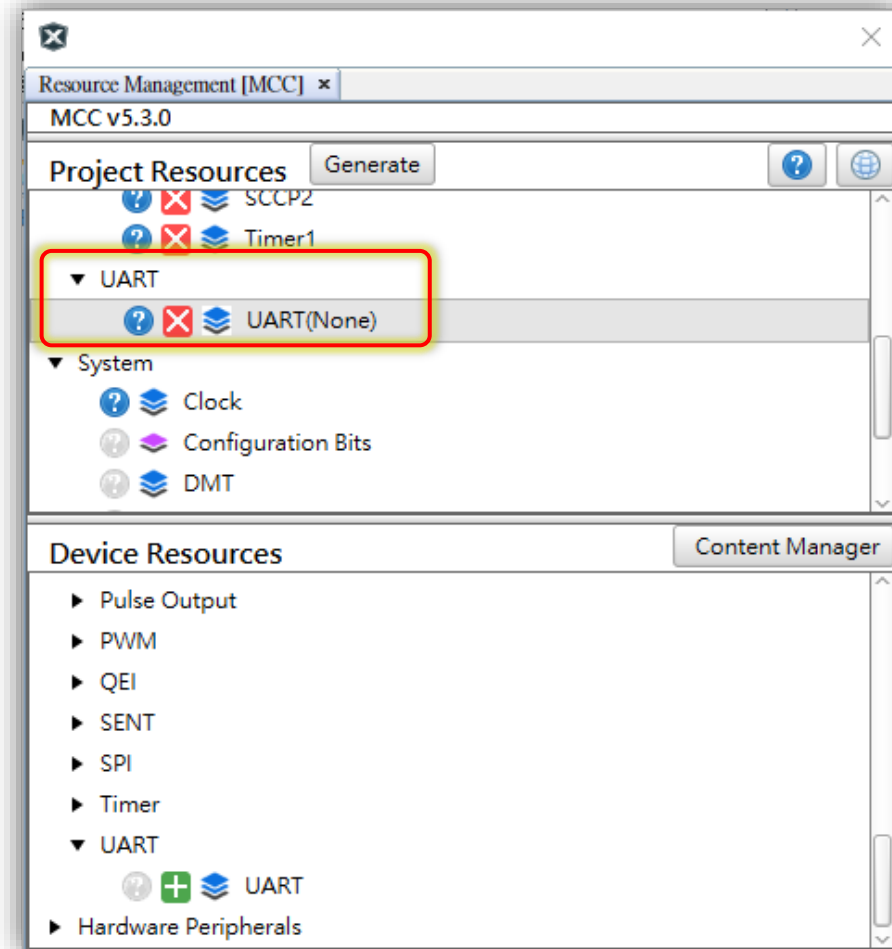
Lab14 UART printf

Step 1

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



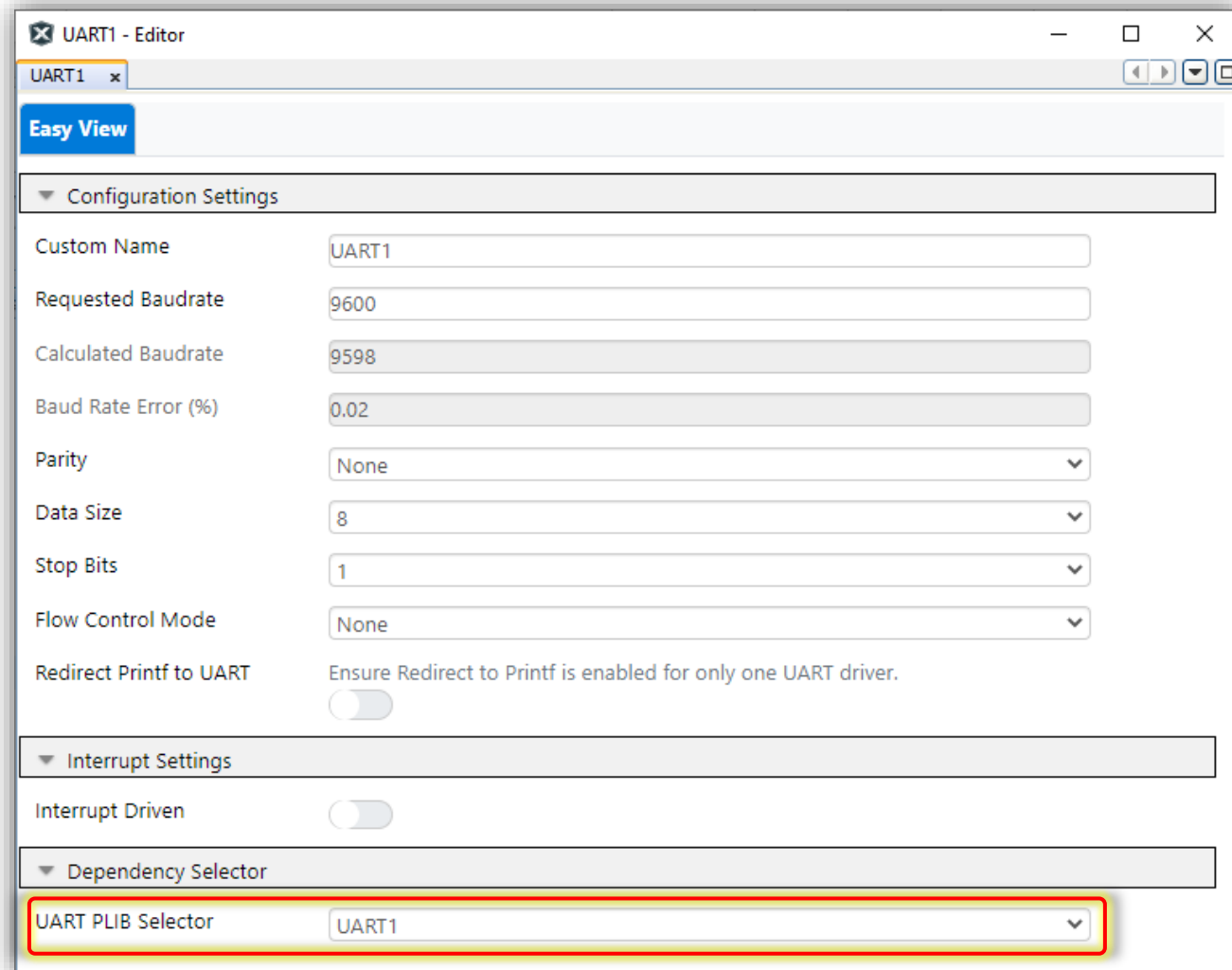
Click [+]



Lab14 UART printf

Step 2

- Set **UART PLIB Selector Module** → **UART1**.



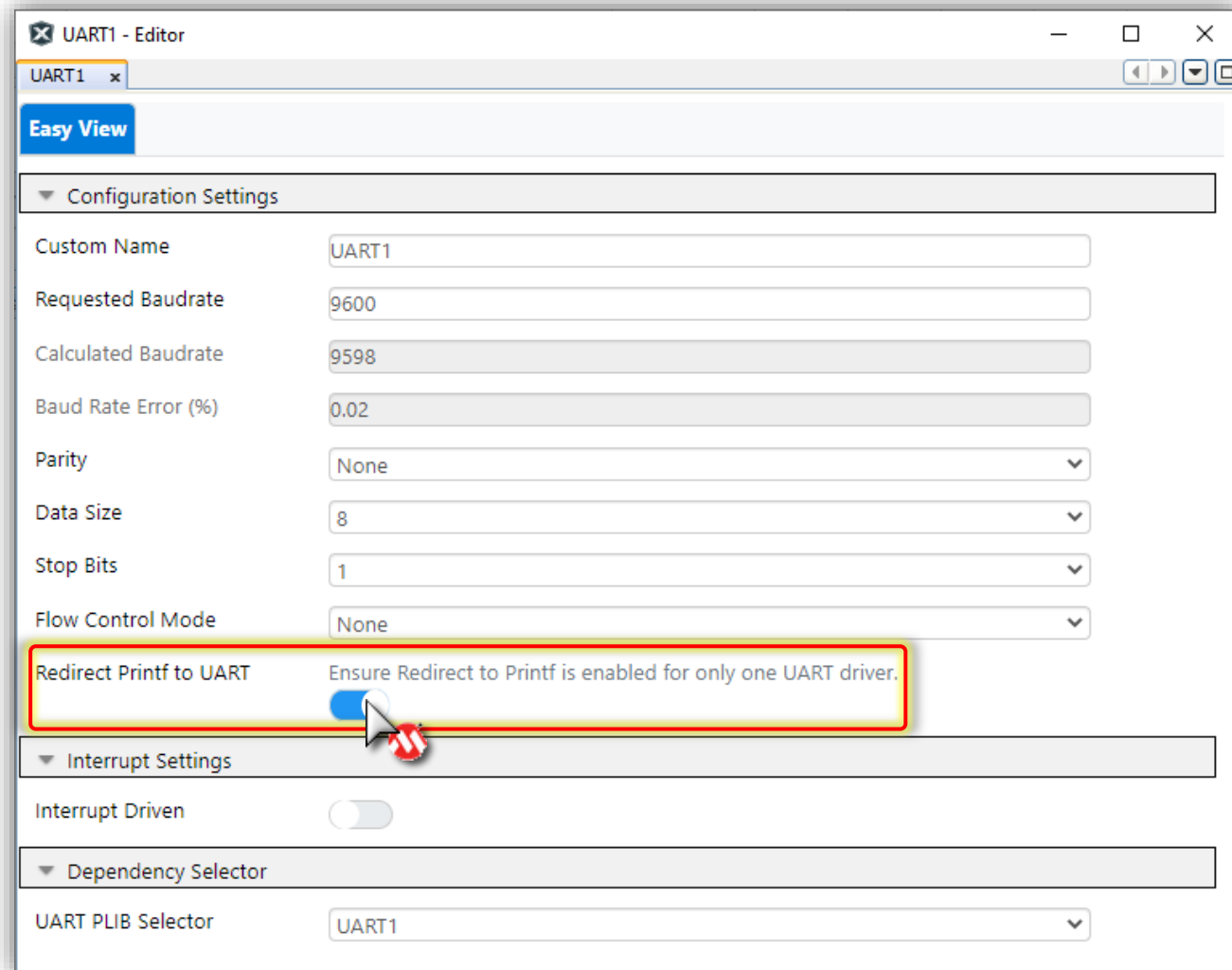
The screenshot shows the 'UART1 - Editor' window with the 'Easy View' tab selected. The 'Configuration Settings' section includes fields for Custom Name (UART1), Requested Baudrate (9600), Calculated Baudrate (9598), Baud Rate Error (%) (0.02), Parity (None), Data Size (8), Stop Bits (1), and Flow Control Mode (None). The 'Redirect Printf to UART' toggle is disabled. The 'Interrupt Settings' section shows 'Interrupt Driven' is disabled. The 'Dependency Selector' section shows 'UART PLIB Selector' set to 'UART1', which is highlighted with a red rectangle.

Setting	Value
Custom Name	UART1
Requested Baudrate	9600
Calculated Baudrate	9598
Baud Rate Error (%)	0.02
Parity	None
Data Size	8
Stop Bits	1
Flow Control Mode	None
Redirect Printf to UART	Disabled
Interrupt Driven	Disabled
UART PLIB Selector	UART1

Lab14 UART printf

Step 3

- Set **Enable Redirect Printf to UART**.



The screenshot shows the 'UART1 - Editor' window with the 'Easy View' tab selected. The 'Configuration Settings' section is expanded, showing various parameters for the UART driver. The 'Redirect Printf to UART' option is highlighted with a red box, and a mouse cursor is pointing at the toggle switch, which is currently turned off. A warning message states: 'Ensure Redirect to Printf is enabled for only one UART driver.'

Parameter	Value
Custom Name	UART1
Requested Baudrate	9600
Calculated Baudrate	9598
Baud Rate Error (%)	0.02
Parity	None
Data Size	8
Stop Bits	1
Flow Control Mode	None
Redirect Printf to UART	☒
Interrupt Driven	☐
UART PLIB Selector	UART1

Lab14 UART printf

Step 4

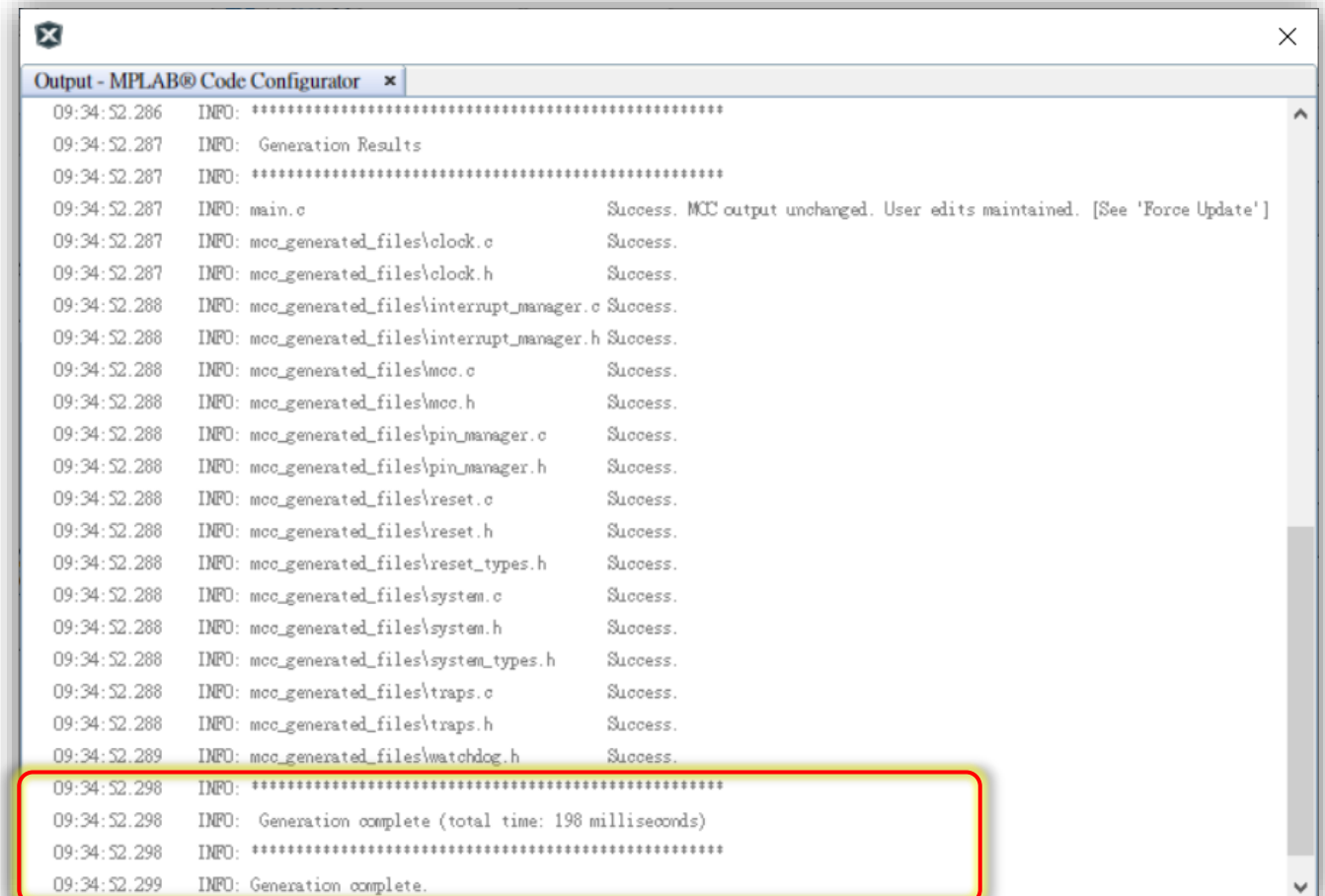
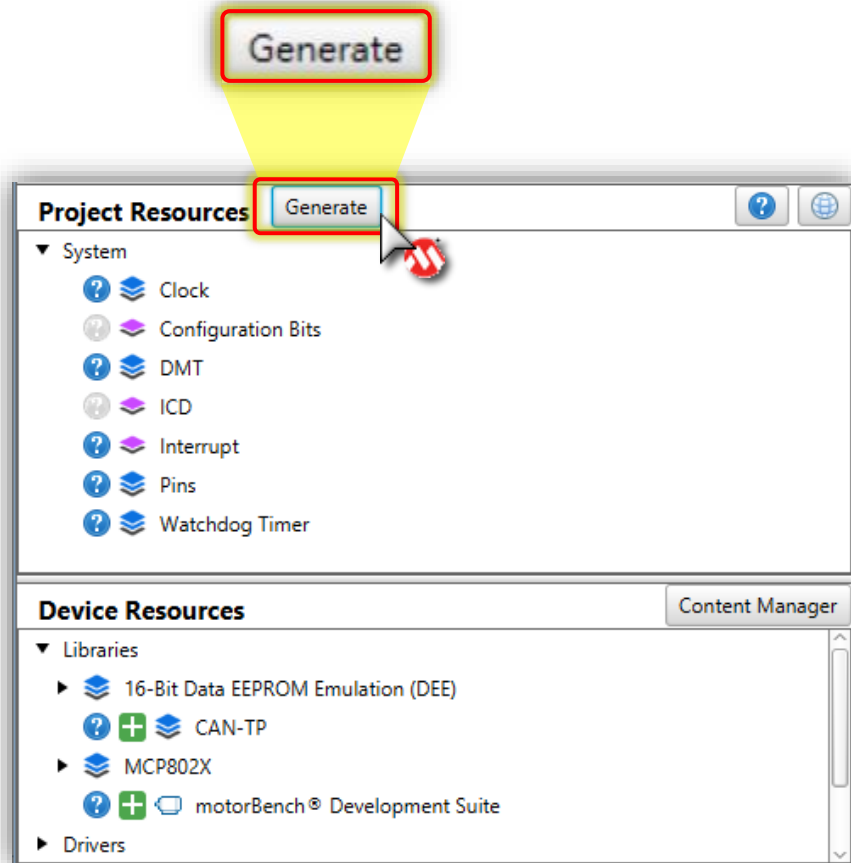
- Set **PORTC**: **RC7 (U1TX)** to UART **TX** Function.
- Set **PORTD**: **RD10 (U1RX)** to UART **RX** Function.

Pin Grid View																											
Package:	TQFP64	Pin No:	32	36	37	52	53	3	4	5	6	60	59	58	55	54	44	43	42	39	38	31	30	21	12	11	8
			PORTC															PORTD									
Module	Function	Direction	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SPI1_Host	SCK1	in/out	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
	SDI1	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
	SDO1	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
ADC1	ANx	input	🔒																		🔒	🔒					
	ADCTRG31	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
ICD	PGCx	input																									
	PGDx	input																									
UART1	U1TX	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
	U1RX	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
Pins	GPIO	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒
	GPIO	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒

Lab14 UART printf

Step 5

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



Lab14 UART printf

Code Example

```
int main(void)
{
    ...

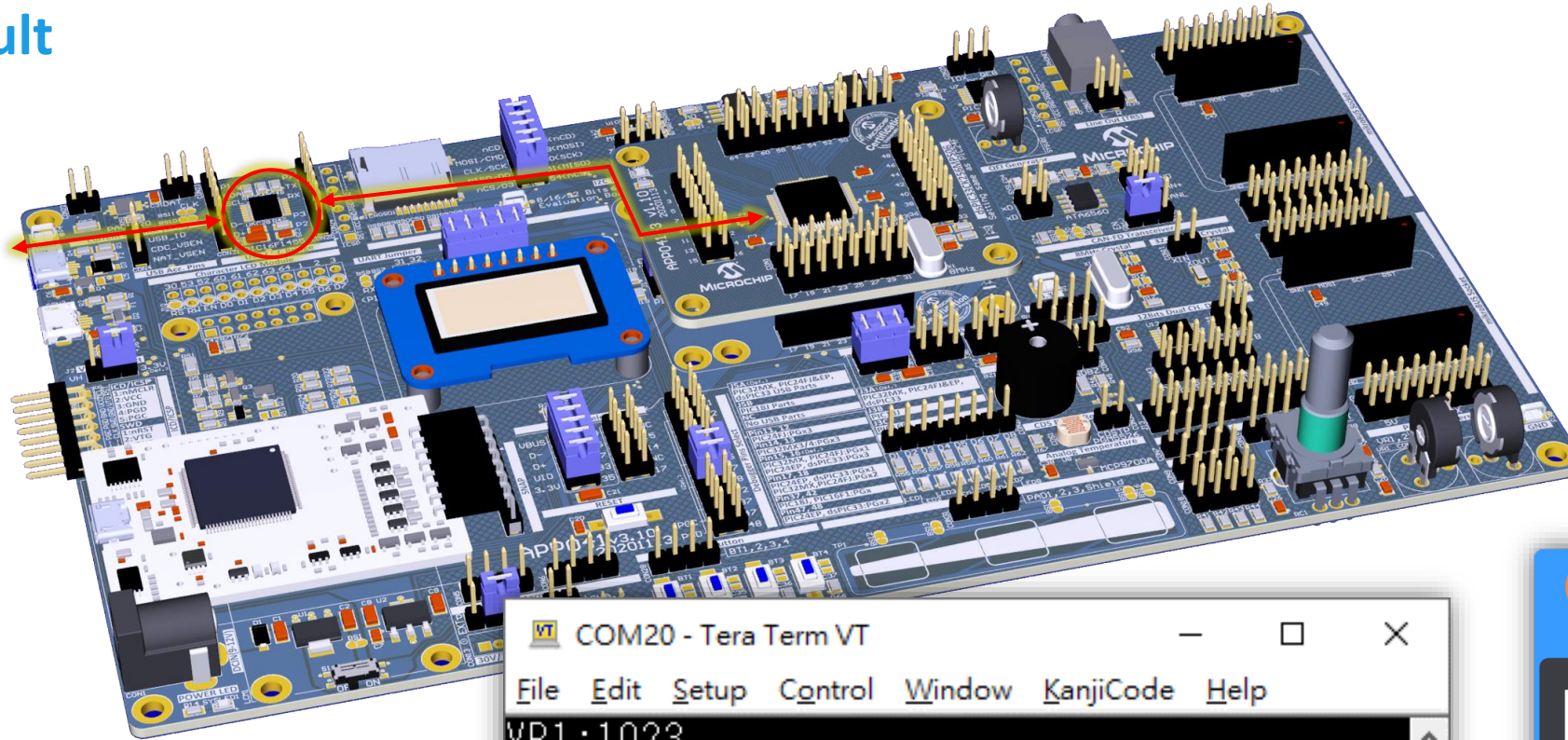
    printf("\033[2J");
    printf("\033[?25l");

    while (1)
    {
        ...
        if (VR1Flag)
        {
            VR1Flag = 0;
            sprintf((char *) StringBuffer, "VR1:%4d", ADC1Buffer[0]);
            myOLED_DrawStr(0, 0, StringBuffer);
            printf(" \033[1;1H%s\r\n", StringBuffer);
            ...
        }

        if (VR2Flag)
        {
            VR2Flag = 0;
            sprintf((char *) StringBuffer, "VR2:%4d", ADC1Buffer[1]);
            myOLED_DrawStr(0, 1, StringBuffer);
            printf(" \033[2;1H%s\r\n", StringBuffer);
            ...
        }
    }
}
```

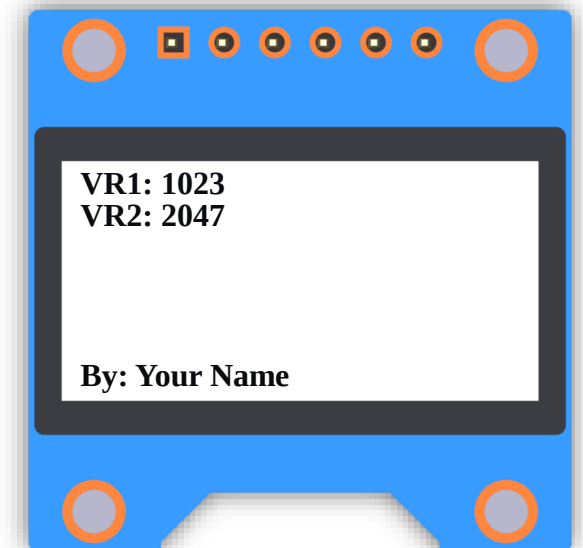
Lab14 UART printf

Result



```
COM20 - Tera Term VT
File Edit Setup Control Window KanjiCode Help
VR1:1023
VR2:2047
```

The actual value control by VR1、VR2



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 Melody

- Melody Provide below functions for UART:

Prototype definition : "uartn.h"

- UART_INTERFACE

- (void) *UARTn_Drv*.Initialize(void);
- (void) *UARTn_Drv*.Deinitialize(void);
- (uint8_t) *UARTn_Drv*.Read(void);
- (void) *UARTn_Drv*.Write(uint8_t txData);
- (bool) *UARTn_Drv*.IsRxReady(void);
- (bool) *UARTn_Drv*.IsTxReady(void);
- (bool) *UARTn_Drv*.IsTxDone(void);
- (void) *UARTn_Drv*.TransmitEnable(void);
- (void) *UARTn_Drv*.TransmitDisable(void);

Lab15 UART Communication

- Base on current project or Open `.\Exercises\Lab15_xxx.X` to do exercise
- 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!

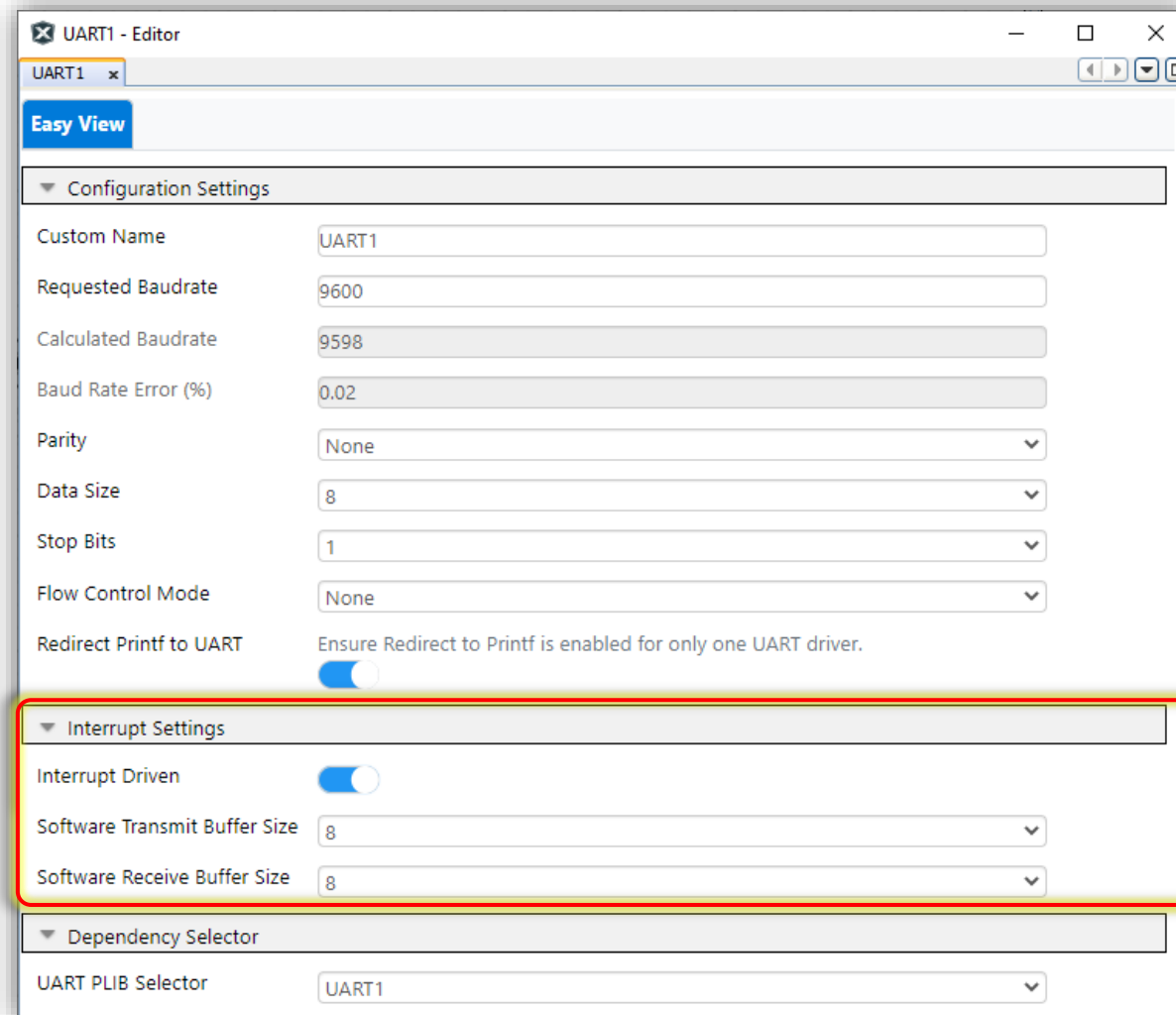
Connection



Lab15 UART Communication

Step 1

- Set **Enable UART Interrupt Driven**.



The screenshot shows the 'UART1 - Editor' window with the following configuration settings:

Configuration Settings	
Custom Name	UART1
Requested Baudrate	9600
Calculated Baudrate	9598
Baud Rate Error (%)	0.02
Parity	None
Data Size	8
Stop Bits	1
Flow Control Mode	None
Redirect Printf to UART	Ensure Redirect to Printf is enabled for only one UART driver. ☒

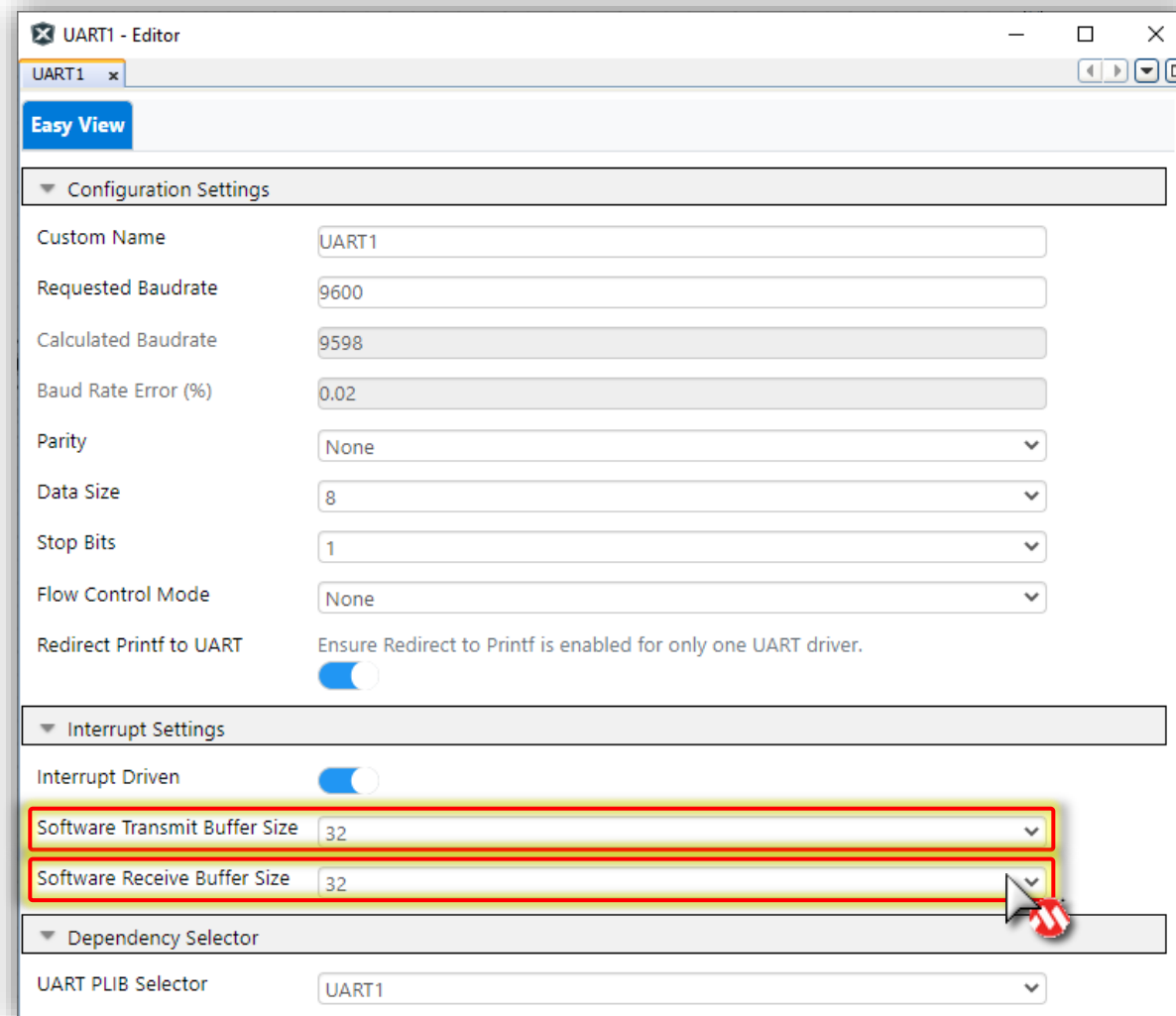
Interrupt Settings	
Interrupt Driven	☒
Software Transmit Buffer Size	8
Software Receive Buffer Size	8

Dependency Selector	
UART PLIB Selector	UART1

Lab15 UART Communication

Step 2

- Select **Software Buffer Size (Transmit and Receive)** to **32**.



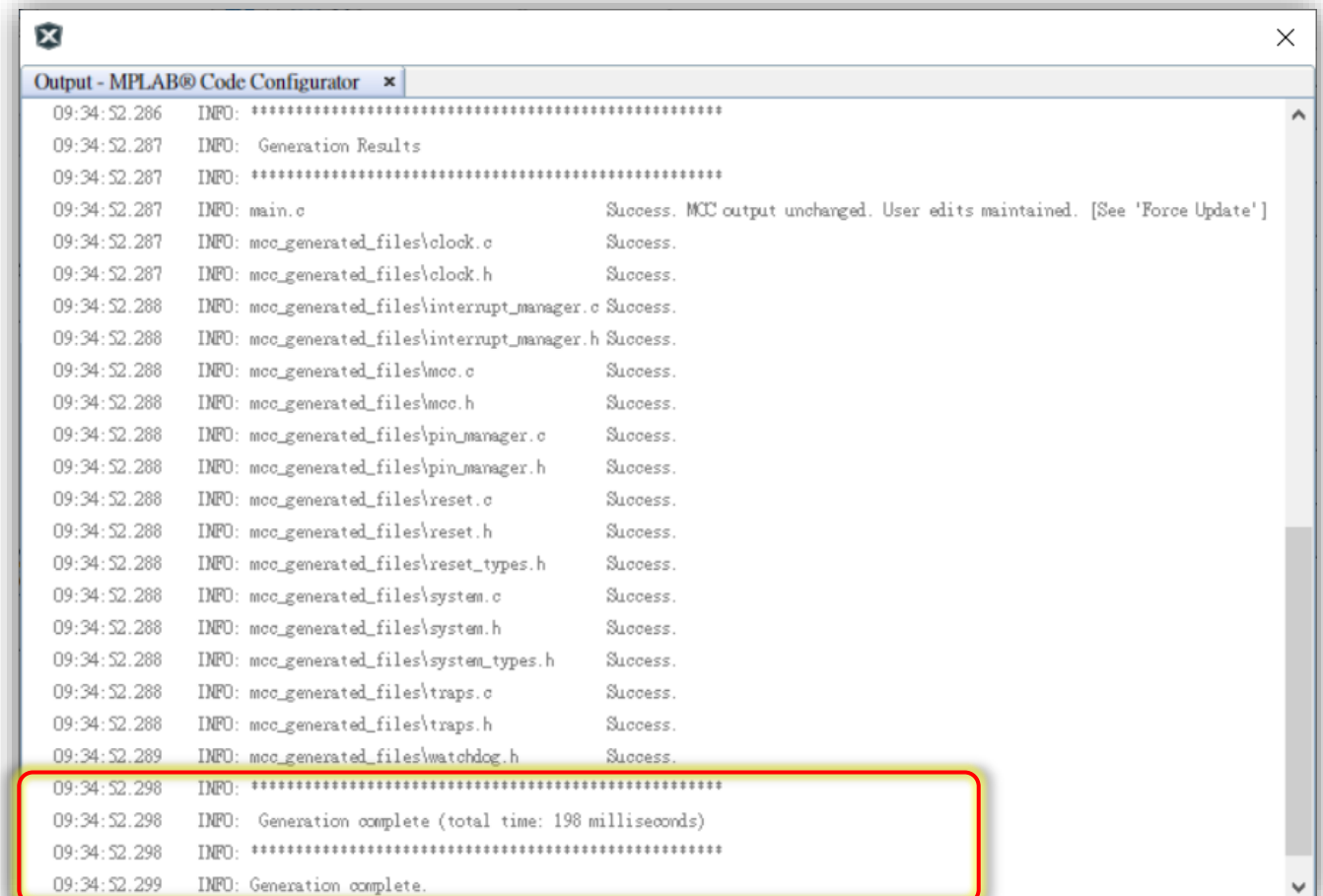
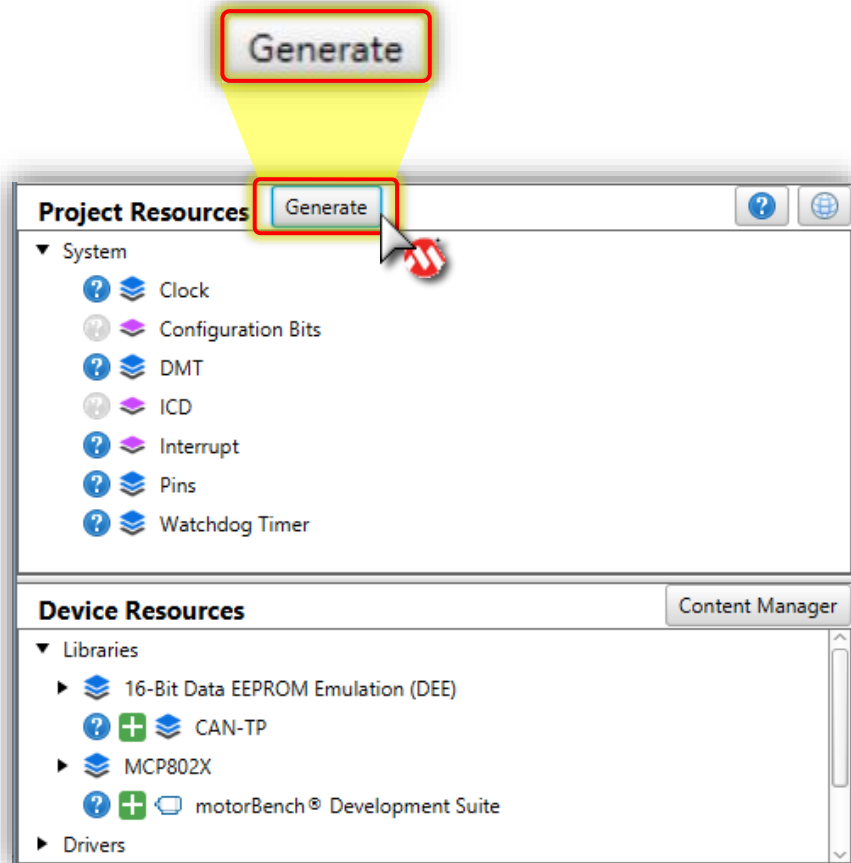
The screenshot shows the 'UART1 - Editor' window with the 'Easy View' tab selected. The 'Configuration Settings' section includes fields for Custom Name (UART1), Requested Baudrate (9600), Calculated Baudrate (9598), Baud Rate Error (%) (0.02), Parity (None), Data Size (8), Stop Bits (1), Flow Control Mode (None), and a toggle for Redirect Printf to UART (enabled). The 'Interrupt Settings' section shows 'Interrupt Driven' as enabled. The 'Software Transmit Buffer Size' and 'Software Receive Buffer Size' are both set to 32, with red boxes highlighting these fields. The 'Dependency Selector' section shows 'UART PLIB Selector' set to UART1.

Section	Parameter	Value
Configuration Settings	Custom Name	UART1
	Requested Baudrate	9600
	Calculated Baudrate	9598
	Baud Rate Error (%)	0.02
	Parity	None
	Data Size	8
	Stop Bits	1
	Flow Control Mode	None
	Redirect Printf to UART	Enabled
Interrupt Settings	Interrupt Driven	Enabled
	Software Transmit Buffer Size	32
	Software Receive Buffer Size	32
Dependency Selector	UART PLIB Selector	UART1

Lab15 UART Communication

Step 3

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



Lab15 UART Communication

Code Example (Use UART Rx Callback)

```
#include "mcc_generated_files/pwm_hs/pwm.h"
#include "mcc_generated_files/uart/uart1.h"
...
volatile uint8_t VR2Flag = 0;
volatile uint8_t UART1_Rx_Done = 0;

void ADC1_CallBack(enum ADC_CHANNEL channel, uint16_t adcVal) { ... }

void UART1_Rx_CallBack(void)
{
    UART1_Rx_Done = 1;
}

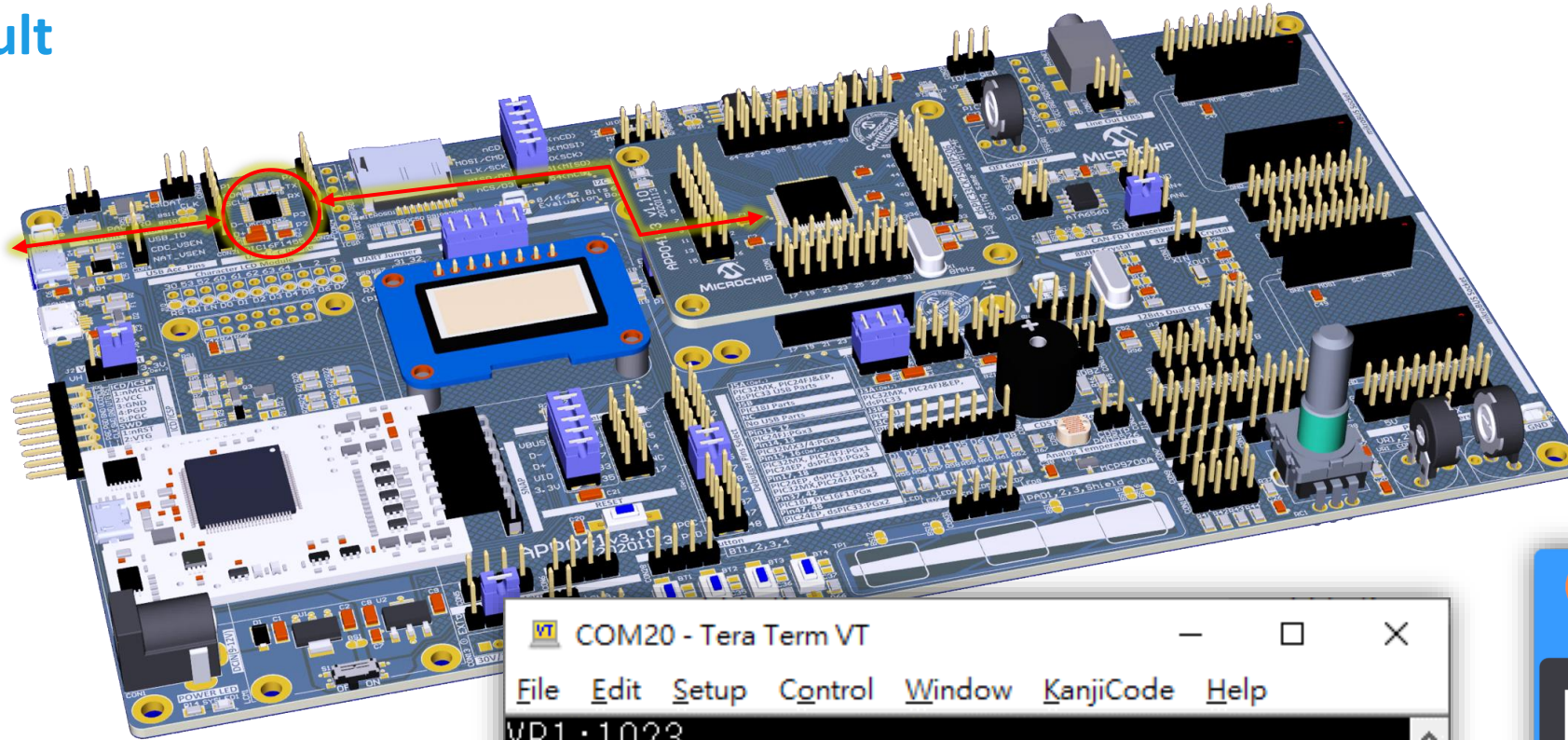
int main(void)
{
    ...
    ADC1.adcMulticoreInterface->ChannelCallbackRegister(ADC1_CallBack);
    UART1_Drv.RxCompleteCallbackRegister(UART1_Rx_CallBack);
    ...

    while (1)
    {
        ...

        if (UART1_Rx_Done)
        {
            UART1_Rx_Done = 0;
            sprintf((char *) StringBuffer, "RxData:%1c", UART1_Drv.Read());
            myOLED_DrawStr(0, 6, StringBuffer);
            printf(" \033[6;1H%s\r\n", StringBuffer);
        }
    }
}
```

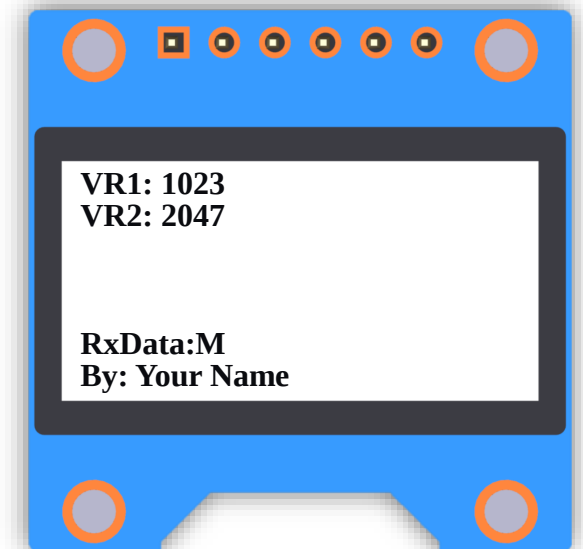
Lab15 UART Communication

Result



```
COM20 - Tera Term VT
File Edit Setup Control Window KanjiCode Help
VR1:1023
VR2:2047

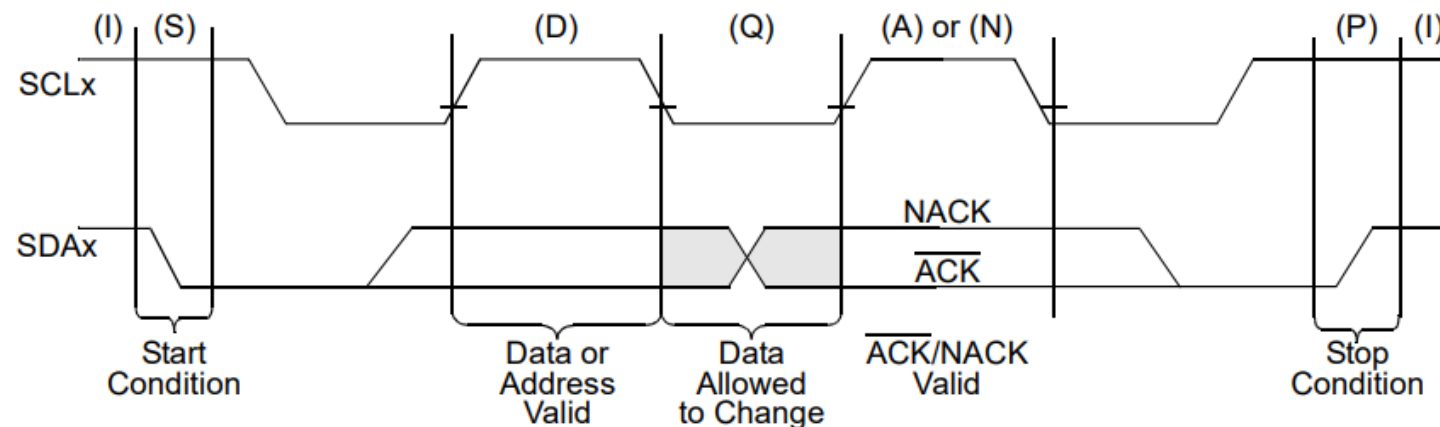
RxData:M
```



I2C Application

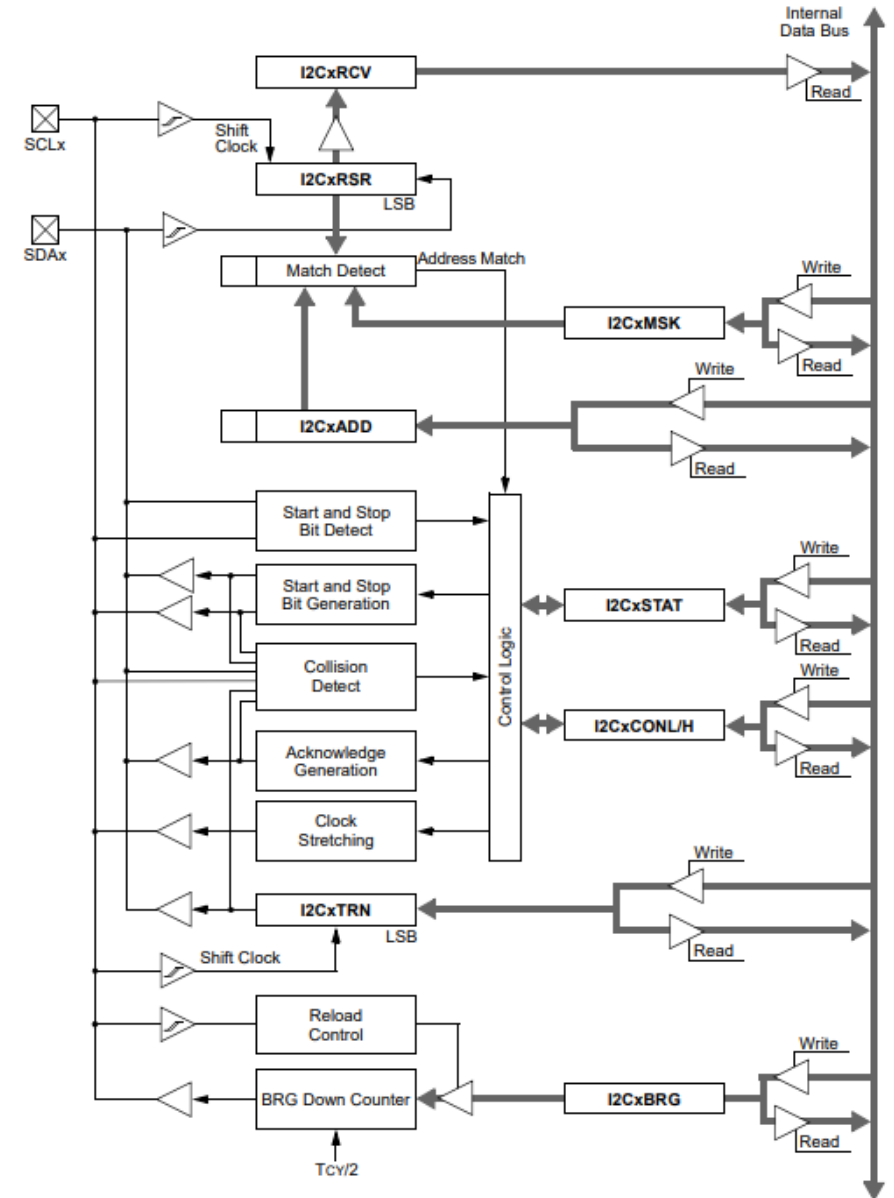
What's I2C

- 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.



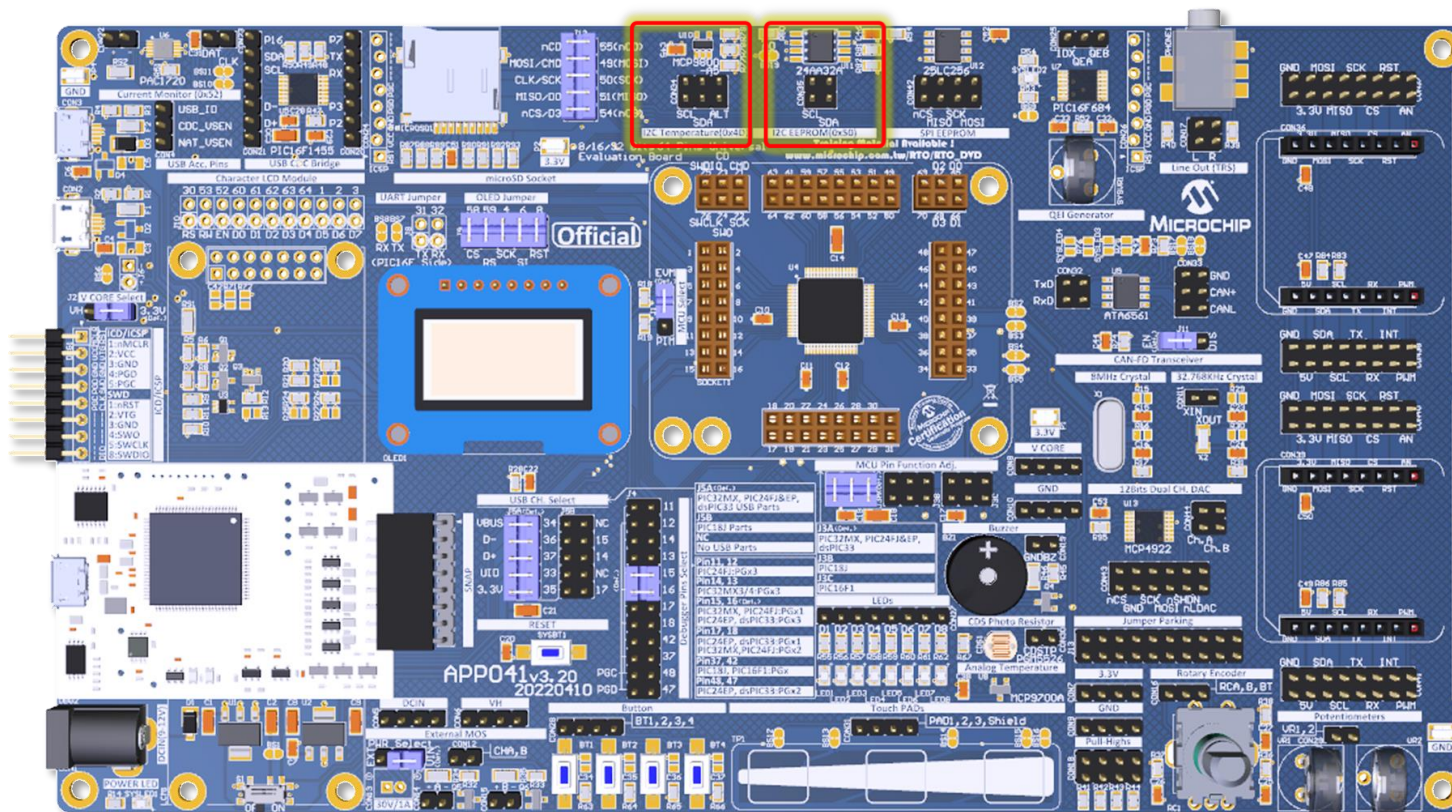
dsPIC33's I2C Module

- 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.



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. $\pm 0.5^{\circ}\text{C}$ (typical) at $+25^{\circ}\text{C}$
- **EEPROM (24AA32A) :**
 - 24AA32A is a 32Kb I2C EEPROM has a Page Write capability for up to 32 bytes of data.



I2C Functions of Melody

- Melody Provide below functions for I2C:

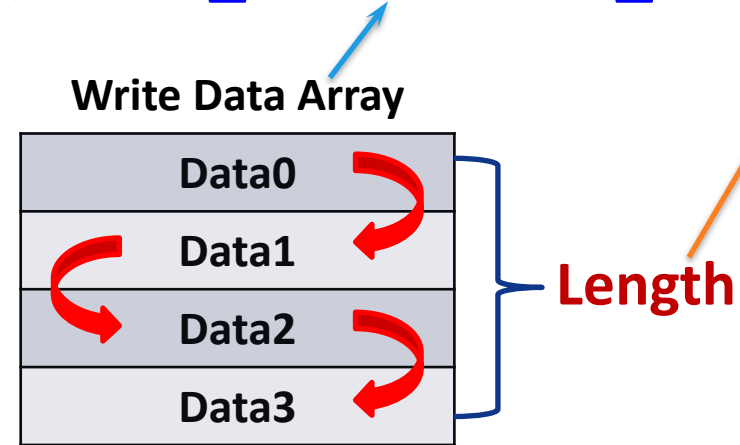
Prototype definition : "i2cn.h"

- UART_INTERFACE

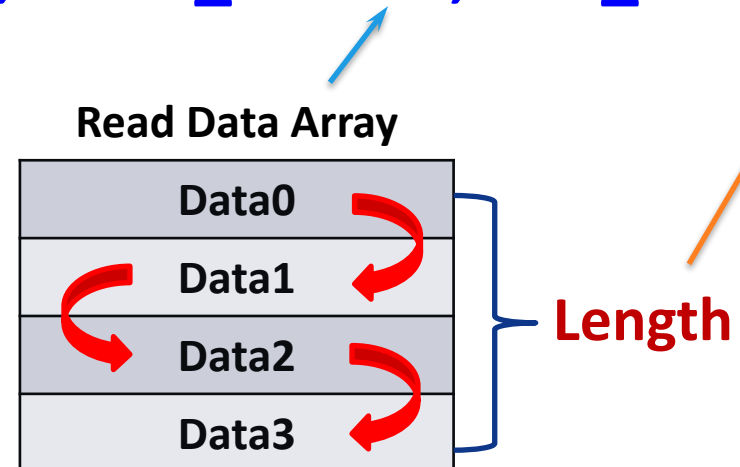
- (void) *I2Cn_Drv*.Initialize(void);
 - (void) *I2Cn_Drv*.Deinitialize(void);
 - (bool) *I2Cn_Drv*.IsBusy(void);
 - (bool) *I2Cn_Drv*.Write(uint16_t address, uint8_t *data, size_t dataLength);
 - (bool) *I2Cn_Drv*.Read(uint16_t address, uint8_t *data, size_t dataLength);
 - (bool) *I2Cn_Drv*.WriteRead(uint16_t address, uint8_t *writeData, size_t writeLength, uint8_t *readData, size_t readLength);
 - (void) *I2Cn_Drv*.HostCallbackRegister(void (*callback)(void));
 - (bool) *I2Cn_Drv*.TransferSetup(struct I2C_TRANSFER_SETUP *setup, uint32_t srcClkFreq);
 - (enum I2C_HOST_ERROR) *I2Cn_Drv*.ErrorGet(void)
- I2C Function is non-block function. This means all function running by interrupt. Operation not execute immediate after function call.

I2C Functions Introduction

- `bool I2Cn_Drv.Write(uint16_t address, uint8_t *data, size_t dataLength);`

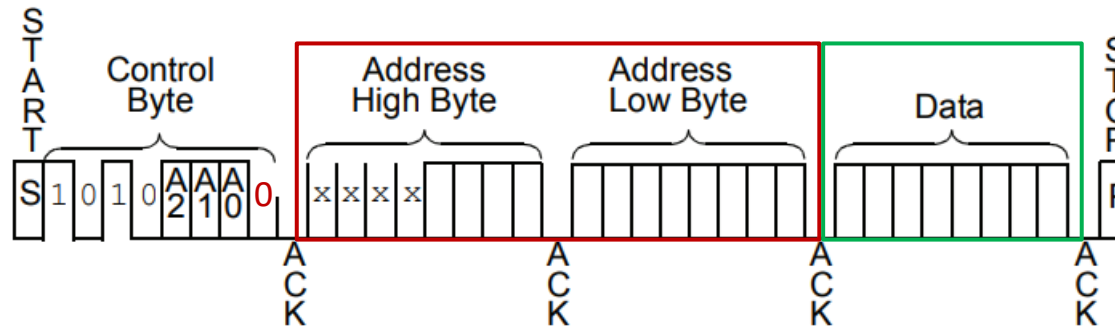


- `bool I2Cn_Drv.Read(uint16_t address, uint8_t *data, size_t dataLength);`

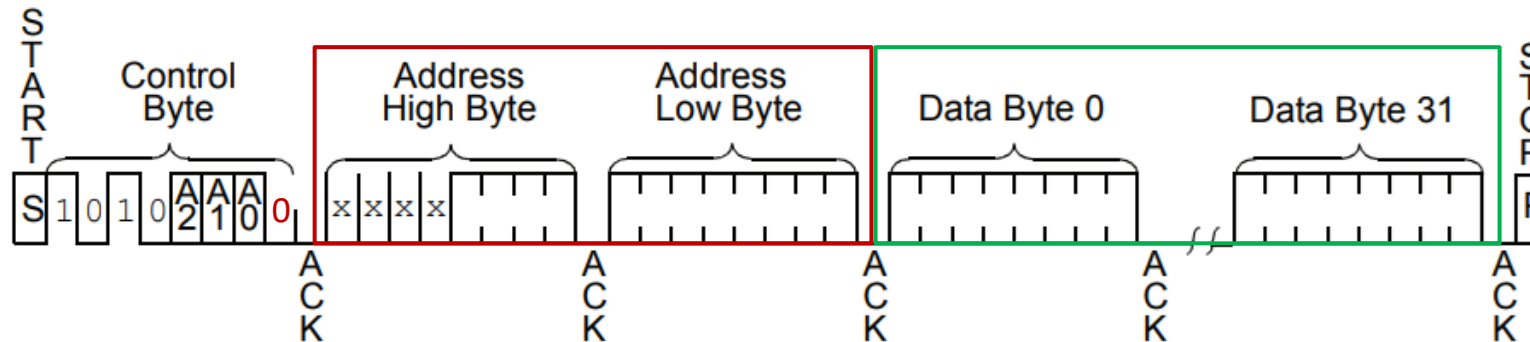


EEPROM Write (24AA32A)

- BYTE WRITE: `I2Cn_Drv.Write();` //Data Array Length = 2+1



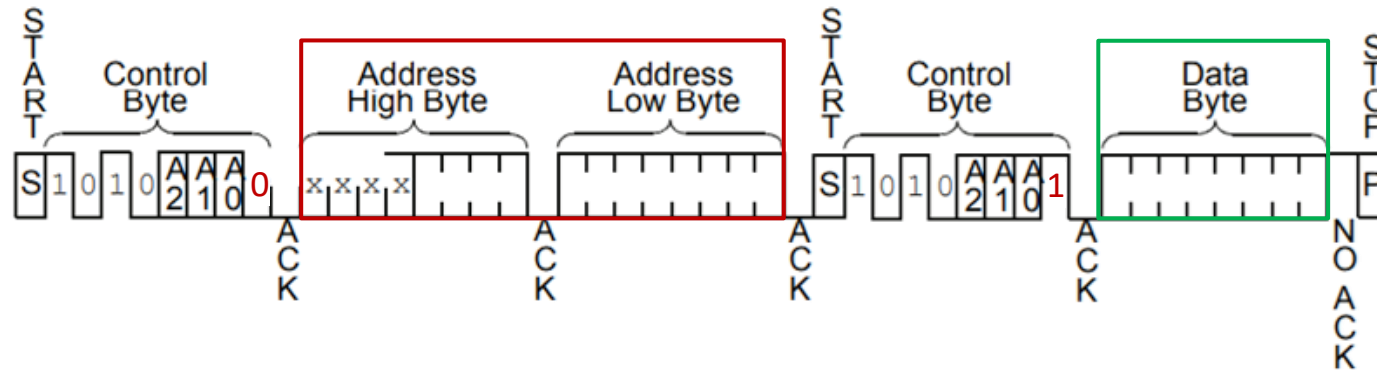
- PAGE WRITE: `I2Cn_Drv.Write();` //Data Array Length = 2+n



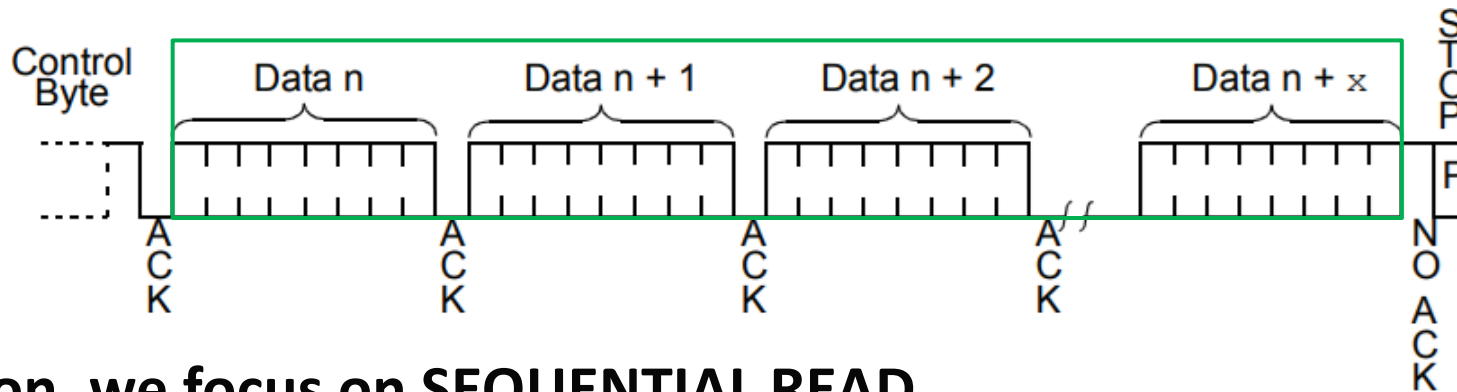
- At this section, we focus on PAGE WRITE.

EEPROM Read (24AA32A)

- RANDOM READ: `I2Cn_Drv.Write() + I2Cn_Drv.Read();` //Data Array Length = 2+1



- SEQUENTIAL READ: `I2Cn_Drv.Write() + I2Cn_Drv.Read();` //Data Array Length = 2+n



- At this section, we focus on SEQUENTIAL READ.

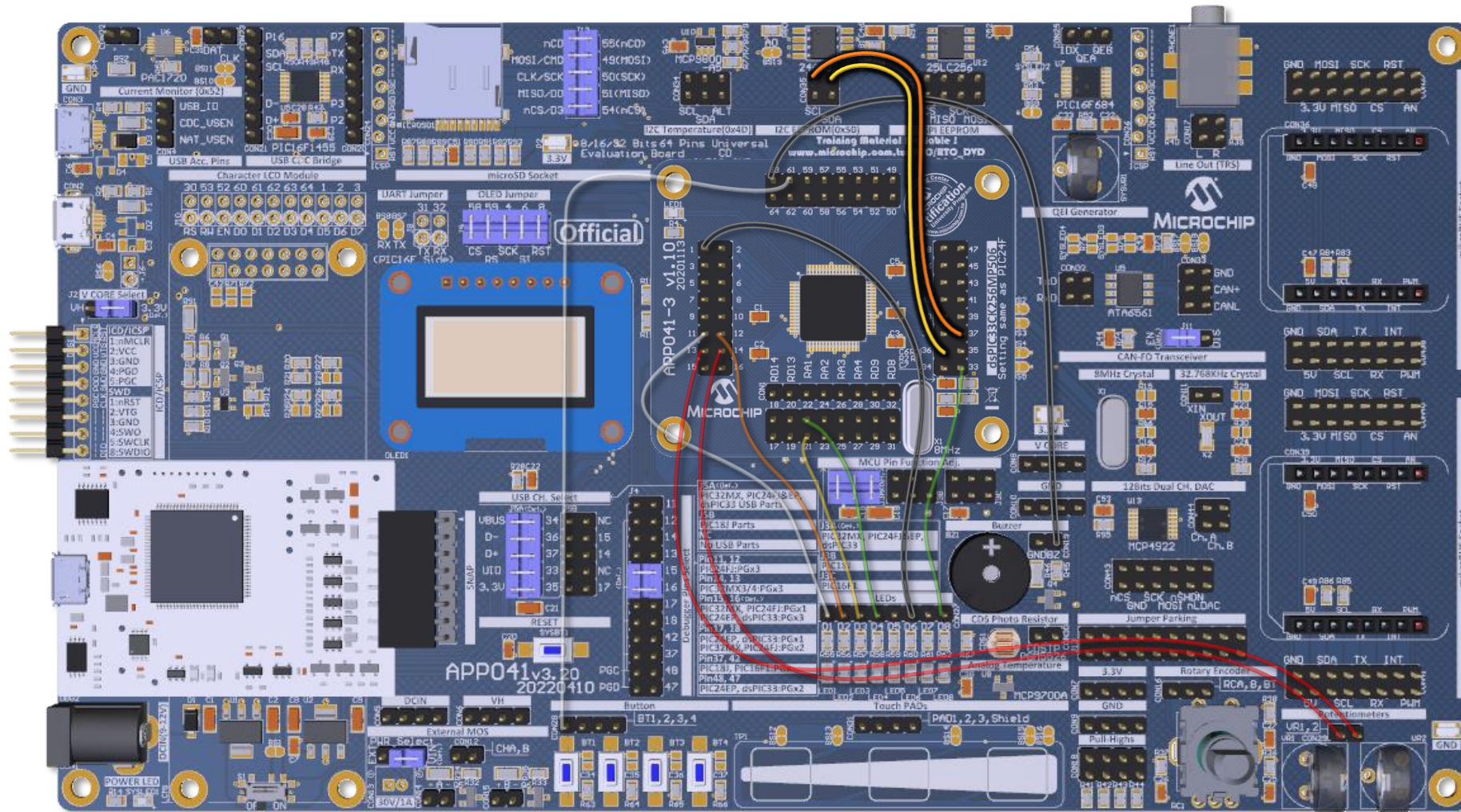
Lab16 EEPROM

- **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!

Lab16 EEPROM

Connection

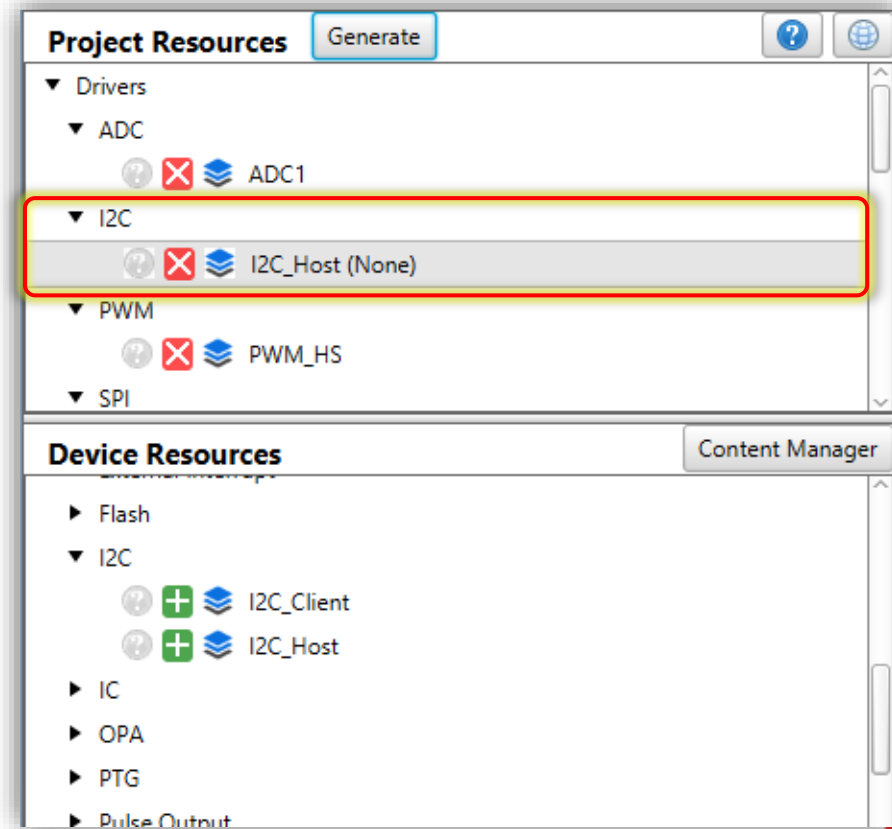
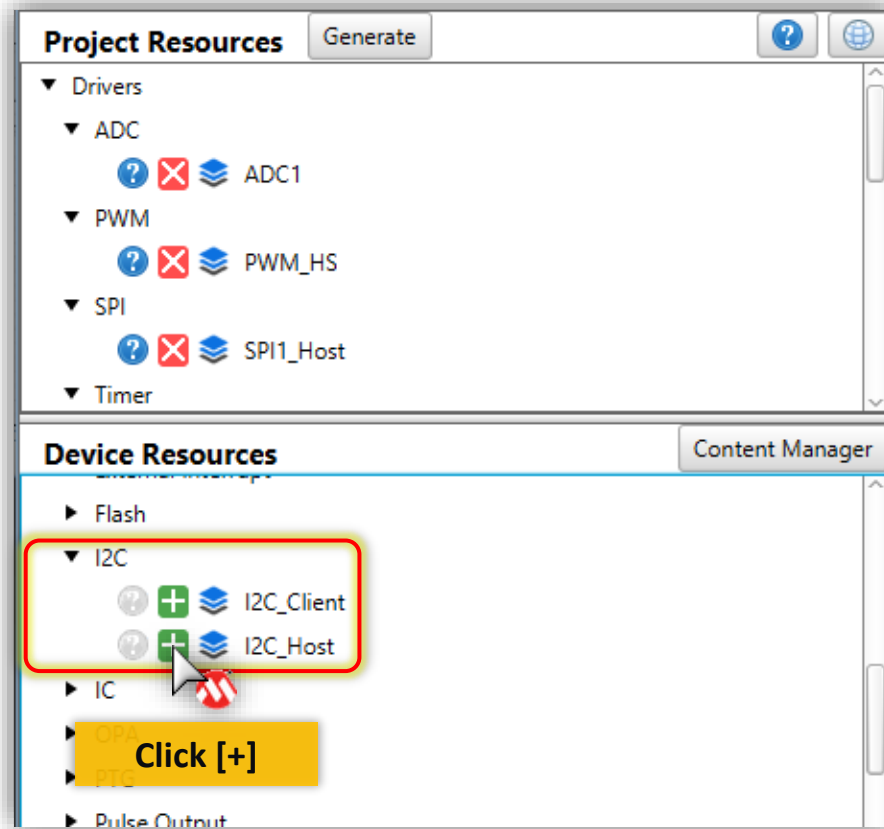


PIM Pin#		Symbol
36		SDA (EEPROM)
37		SCL (EEPROM)

Lab16 EEPROM

Step 1

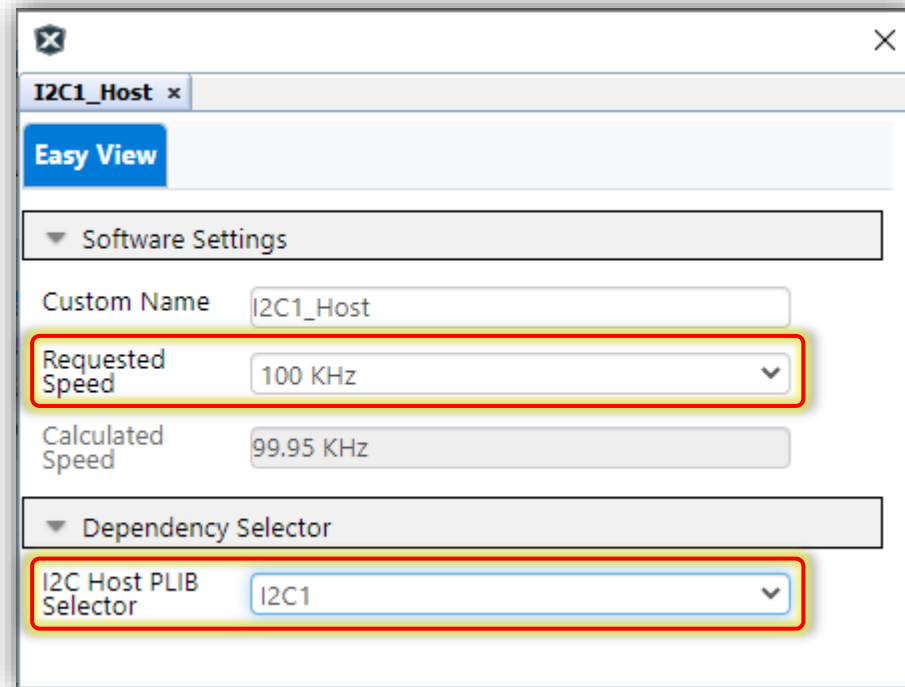
- Click [+] to add I2C > I2C_Host Module.



Lab16 EEPROM

Step 2

- Set **I2C1_Host > Dependency Selector** → **I2C1**.
- Check **Requested Speed** is **100 kHz**.



Lab16 EEPROM

Step 3

- Change **I2C1_Host SCL1** and **SDA1** Pin to **PORTC: RC8 (SDA1), RC9 (SCL1)**.

✕

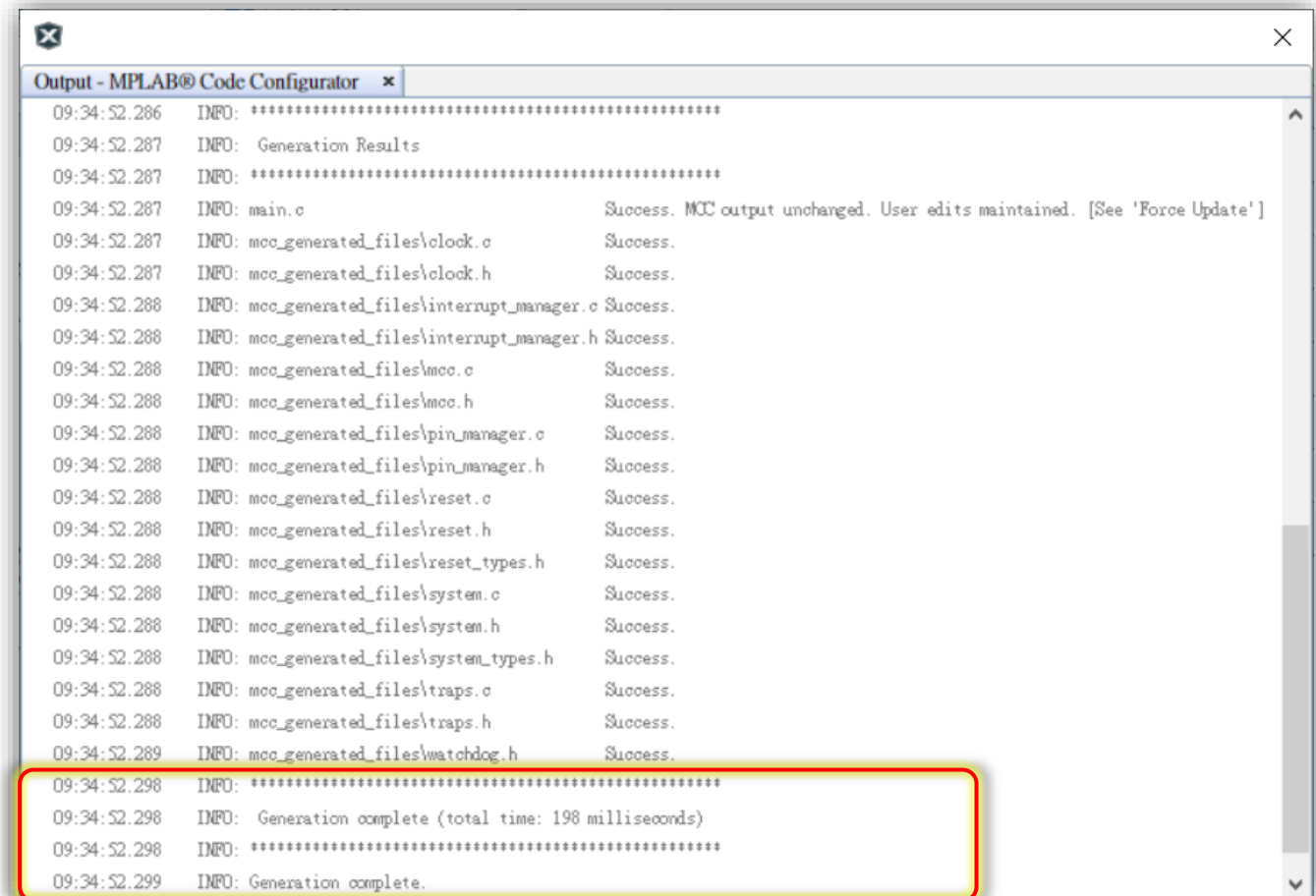
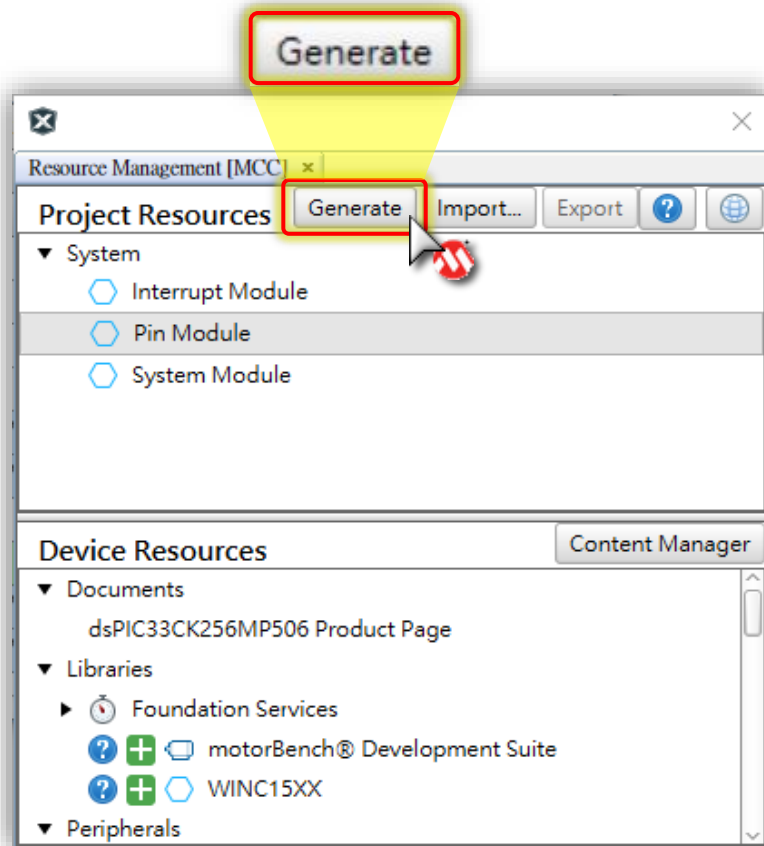
Pin Grid View ✕

Package:	TQFP64 ▾	Pin No:	46	47	48	49	61	62	63	64	1	2	13	22	23	27	50	51	24	32	36	37	52	53	3	4	5	6				
			PORTB															PORTC														
Module	Function	Direction	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
SPI1_Host ▾	SCK1	in/out	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
	SDI1	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
	SDO1	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
UART1 ▾	U1TX	output	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
	U1RX	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				
I2C1_Host ▾	SCL1	in/out			🔒																											
	SDA1	in/out				🔒																										
ADC1 ▾	ANx	input		🔒	🔒	🔒							🔒	🔒	🔒	🔒			🔒	🔒												
	ADCTRG31	input	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒				

Lab16 EEPROM

Step 4

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



Lab16 EEPROM

Code Example

```
...
#include "mcc_generated_files/uart/uart1.h"
#include "mcc_generated_files/i2c_host/i2c1.h"
...
volatile uint8_t UART1_Rx_Done = 0;

#define EEPROM_Address 0x50

uint8_t BT_Flag = 0;
uint8_t I2C_EEPROM_Write_Data[4];
uint8_t I2C_EEPROM_Read_Data[2];
...
int main(void)
{
    while (1)
    {
        if (BT1_GetValue()) {
            if (BT_Flag == 0) {
                BT_Flag = 1;
            }
            LED8_SetLow();
        } else {
            if (BT_Flag == 1) {
                BT_Flag = 0;
                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_Host.Write(EEPROM_Address, I2C_EEPROM_Write_Data, 4);
                while (I2C1_Host.IsBusy());
            }
            LED8_SetHigh();
        }
    }
    ...
}
```

Lab16 EEPROM

Code Example

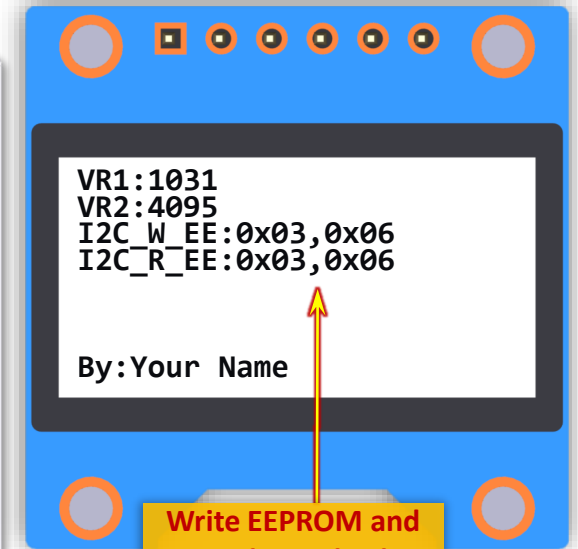
```
int main(void)
{
    ...
    while (1)
    {
        ...
        if (TMR1Flag)
        {
            TMR1Flag=0;
            ADC1.SoftwareTriggerEnable();

            I2C_EEPROM_Write_Data[0] = 0x00;
            I2C_EEPROM_Write_Data[1] = 0x00;
            I2C1_Host.WriteRead(EEPROM_Address, I2C_EEPROM_Write_Data, 2, I2C_EEPROM_Read_Data, 2);
            while (I2C1_Host.IsBusy());

            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);

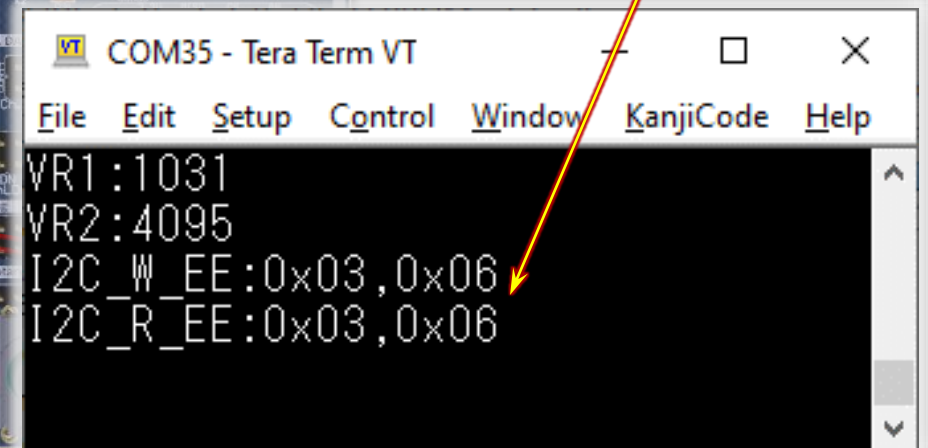
            sprintf((char *) StringBuffer, "I2C_R_EE:0x%02X,0x%02X", I2C_EEPROM_Read_Data[0], I2C_EEPROM_Read_Data[1]);
            printf("\033[4;1H%s", StringBuffer);
            myOLED_DrawStr(0, 3, StringBuffer);
        }
        ...
    }
}
```

Result



By: Your Name

Write EEPROM and Read Data back



MCP9800 Read & Configuration

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

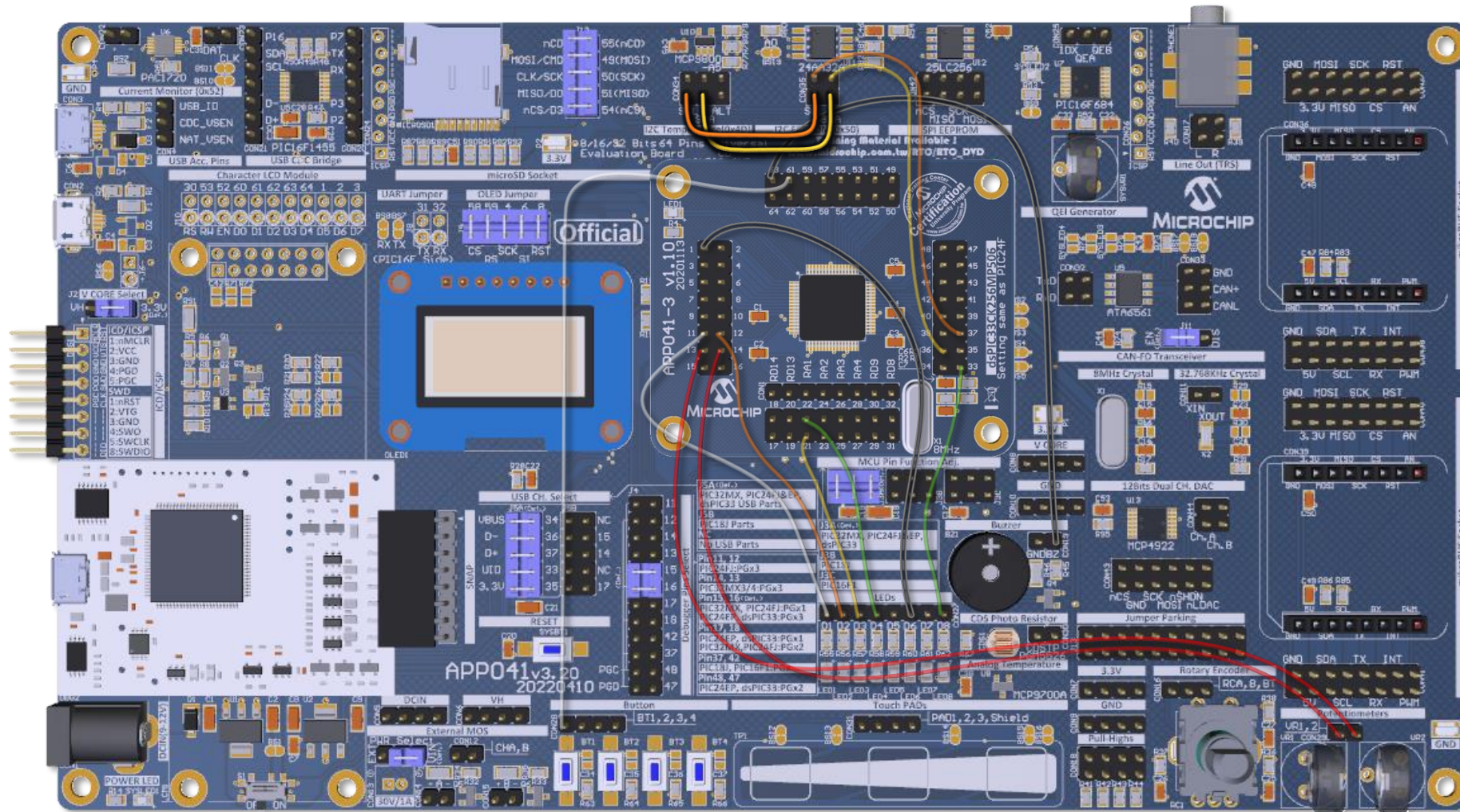
TABLE 5-1: BIT ASSIGNMENT SUMMARY FOR ALL REGISTERS									
Register Pointer P1 P0	MSB/ LSB	Bit Assignment							
		7	6	5	4	3	2	1	0
Ambient Temperature Register (T _A)									
0 0	MSB	Sign	2 ⁶ °C	2 ⁵ °C	2 ⁴ °C	2 ³ °C	2 ² °C	2 ¹ °C	2 ⁰ °C
	LSB	2 ⁻¹ °C	2 ⁻² °C	2 ⁻³ °C	2 ⁻⁴ °C	0	0	0	0
Sensor Configuration 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!

Connection



PIM Pin#		Symbol
SDA (EEPROM)		SDA (Temp.)
SCL (EEPROM)		SCL (Temp.)

Lab17 Temperature Sensor

Notice



The screenshot shows the 'PWM_HS' configuration window in the MPLAB IDE. A large yellow star is overlaid on the window, containing the text 'Don't Need Modify Any MCC Melody Setting !'. The window is divided into two main sections: 'Basic Configuration' and 'PWM Configuration'. The 'Basic Configuration' section includes fields for 'Custom Name' (PWM_HS), 'PWM Generators Required' (1), and 'Interrupt Driven'. The 'PWM Configuration' section includes 'Basic Settings' (Enable, Custom Name: PWM_GENERATOR_1) and 'Min and Max Limit' (Frequency in kHz: [0.122 < Freq < 470.588], Duty Cycle in (%): [0 < DC < 100], Dead Time Delay in (µs): [0 < DT < 12793.50000], Trigger Delay in (µs): [0 < TD < 4095.937]). The 'Frequency Settings' table shows the following data:

PWM Generator	Requested Frequency (kHz)	Calculated Frequency (kHz)	Duty Cycle %	Dead Time (µs)	Operating Mode	Output Mode	Synchronous Update
PWM 1	1	1	20	0	Center-Aligned	Complementary	☐

The 'Output Control Settings' section is partially visible at the bottom.

Lab17 Temperature Sensor

Code Example

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

uint8_t I2C_EEPROM_Read_Data[2];
...

#define MCP9800_Address      0x4D
#define MCP9800_REG_CONFIG   0x01
#define MCP9800_Register     0x60
#define MCP9800_REG_TA      0x00

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

int main(void)
{
    ...
    printf("\033[?25l");

    MCP9800_Write_Data[0] = MCP9800_REG_CONFIG;
    MCP9800_Write_Data[1] = MCP9800_Register;
    I2C1_Host.Write(MCP9800_Address, MCP9800_Write_Data, 2);
    while (I2C1_Host.IsBusy());
    MCP9800_Write_Data[0] = MCP9800_REG_TA;
    I2C1_Host.Write(MCP9800_Address, MCP9800_Write_Data, 1);
    while (I2C1_Host.IsBusy());

    while (1)
    {
        ...
    }
}
```

Lab17 Temperature Sensor

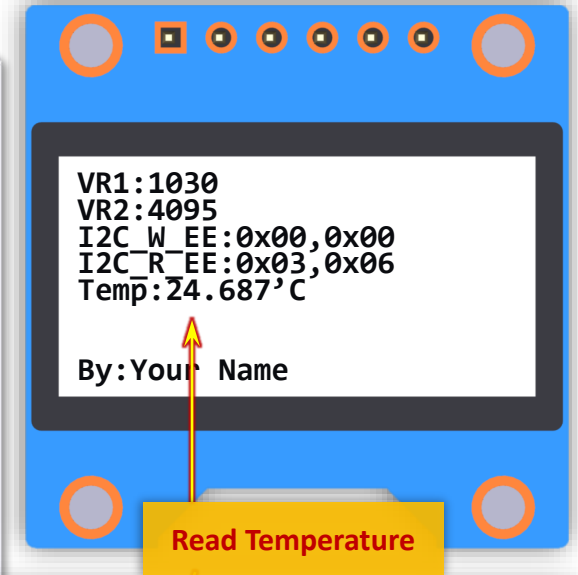
Code Example

```
int main(void)
{
    ...
    while (1)
    {
        ...
        if (TMR1Flag)
        {
            TMR1Flag=0;
            ADC1.SoftwareTriggerEnable();
            ...

            I2C1_Host.Read(MCP9800_Address, MCP9800_Read_Data, 2);
            while (I2C1_Host.IsBusy());

            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);
        }
        ...
    }
}
```

Result



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, Klear, 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, SQL, 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

