

SAMD21 ARM Cortex-M0+ Microcontroller & MCC Harmony SMA2001-MCC



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



SMART | CONNECTED | SECURE

**CAE Taiwan
Microchip Technology Inc.**

March 31, 2023

Class Objectives

- **When you finish this course, you will be able to:**
 - Be familiar with MPLAB X IDE and MCC Harmony
 - Understand how to configure and use the SAMD21 series peripherals.
- **There have two level course included.**
 - Major course : General study of basic features.
 - Appendix : Advance features and appendix.

Major Course (1/3)

- SAMD21 ARM Cortex-M0+ Microcontroller Introduction
- MPLAB® X IDE, MCC Harmony and Development Tools Introduction
- SAMD21 EVB APP045 v4.20 Introduction
- Getting Started with First Project
 -  Lab0 First Project
- MPLAB® Code Configurator Harmony Interface introduction
 -  Study Advanced Features
- PORT I/O Architecture
 -  Lab1 PORT Output
 -  Lab2 PORT Input and Output
- Timer/Counter, TC Architecture
 -  Lab3 TC Timer Method Polling
 -  Lab4 TCs Timer Method Polling

Major Course (2/3)

● NVIC Architecture

 Lab5 TC Timer Method Interrupt Callback

 Lab6 TCs Timer Method Interrupt Callback

 Study Advanced Features

● Clock System Architecture

 Lab7 System Clock XOSC32K

 Lab8 System Clock DFLL48M

● Pin Multiplexer (PINMUX)

● SERCOM - UART Architecture

 Lab9 SERCOM - UART Tx Polling

 Lab10 SERCOM - UART TxRx Interrupt Callback

 Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

Major Course (3/3)

- High Speed ADC Architecture

 -  Lab12 ADC Polling Software Trigger

- ADC Interrupt Callback and Pin Scan

 -  Lab13 ADC Pin Scan

- TCC – PWM Architecture

 -  Lab14 TCC NPWM

 -  Lab15 TCC NPWM ADC Duty Control

 -  Lab16 TCC NPWM Auto Duty Control

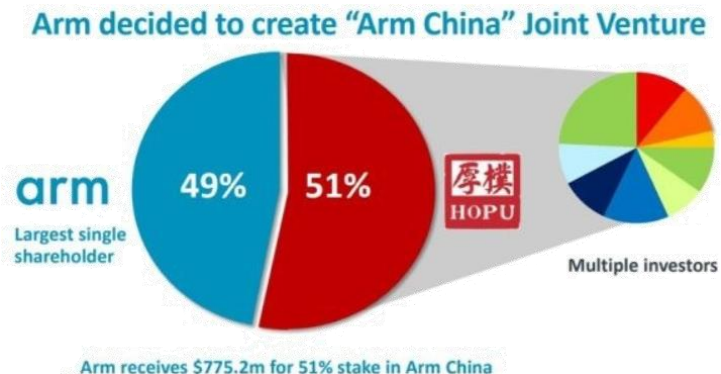
- SERCOM-SPI (OLED SSD1306)

 -  Lab17 SERCOM - SPI SSD1306 OLED

SAMD21 ARM Cortex-M0+ Microcontroller Introduction

ARM's history

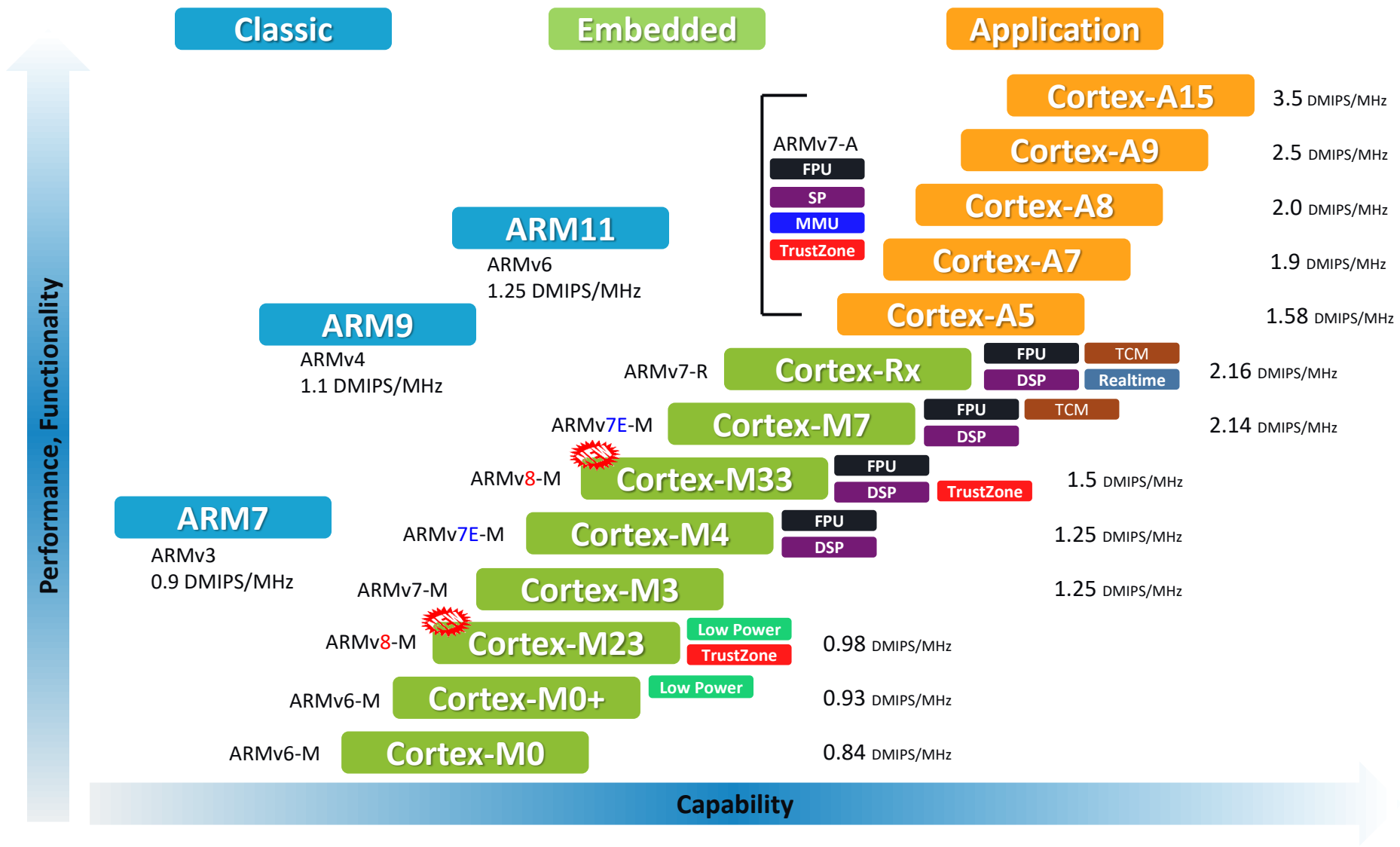
- **ARM**, previously **Advanced RISC Machine**, is a family of [reduced instruction set computing](#) (RISC) [architectures](#) for [computer processors](#). [Arm Holdings](#) develops the architecture and licenses it to other companies, who design their own products that implement [systems-on-chips](#) (SoC) and [systems-on-modules](#) (SoM).
- SoftBank complete acquisition of ARM Holdings on September 5, 2016 (GMT) for a total acquisition price approximately USD 31.0 billion or JPY 3.3 trillion.
- SoftBank confirmed on June 4, 2018 that Arm Limited agreed sell a majority stake in its Chinese subsidiary Arm Technology (China) Co., Ltd. Under the terms of the Transaction, Arm will sell 51% of its equity interest in Arm China for USD 775.2 million to the Direct Investors.
- On 13 September 2020, it was announced that Nvidia would buy Arm from SoftBank for \$40 billion, subject to regulatory approval, with the latter acquiring a 10% share in Nvidia.



ARM in Partnership



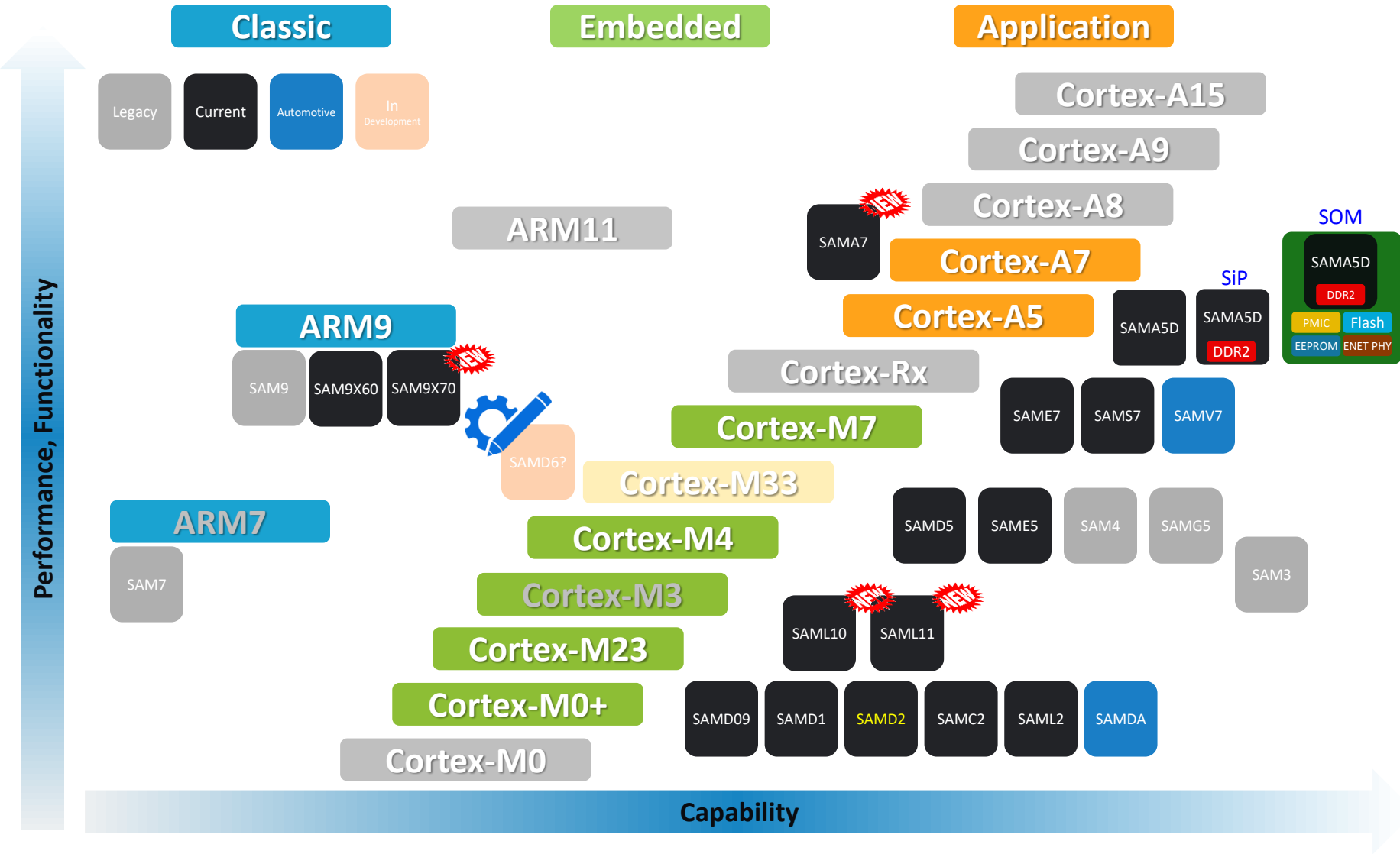
ARM core MCU/MPU brief



ARM core MCU/MPU success



Microchip ARM Core Products



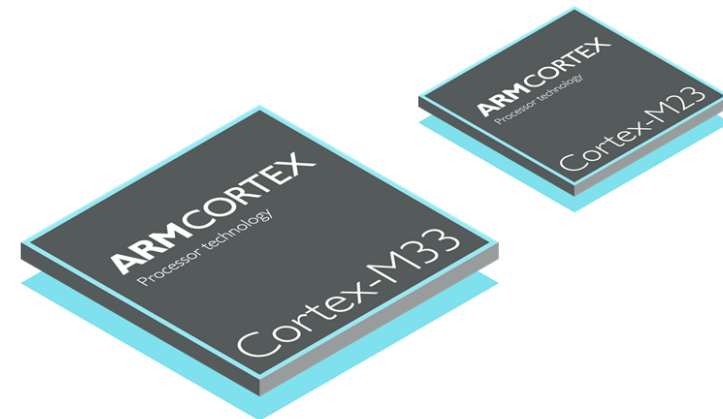
Microchip Cortex-M MCU Series

- **Microchip 32 Bits (ARM core) MCU include below series :**
- **Cortex-M0+**
 - SAMD09/SAMD1x
 - Low pin count, high c/p.
 - SAMD2x
 - Rich peripheral.
 - SAMC2x
 - Support 5V and CAN bus.
 - SAML2x
 - Lowest power consumption and peripheral.
 - SAMDAx
 - Automotive certificated.
- **Cortex-M3, Cortex-M4 (Legacy)**
 - SAM3x, SAM4x, SAMG5x.



Microchip Cortex-M MCU Series (cont.)

- **Cortex-M4 (New generation)**
 - SAMD5x
 - Support float point FPU, crypto engine, USB , low power.
 - SAME5x
 - FPU, crypto engine, USB, Ethernet, CAN-FD, Low power.
- **Cortex-M23**
 - SAML1x
 - Very low power, support TrustZone.
- **Cortex-M33 (Coming soon)**
 - SAMD6x
 - High performance, support TrustZone.



Microchip Cortex-M MCU Series (cont.)

- **Cortex-M7**

- SAMS7x

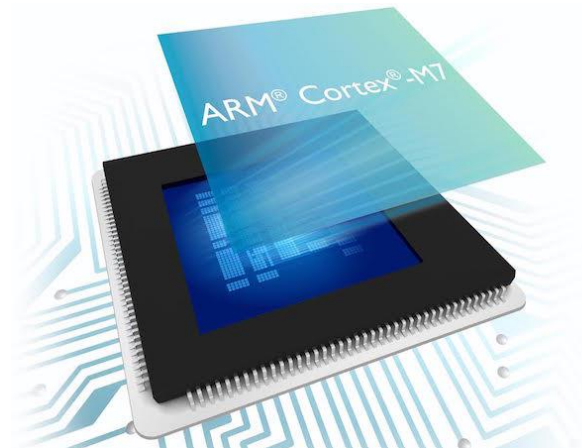
- High performance, rich peripheral.

- SAME7x

- Ethernet support, CAN-FD support.

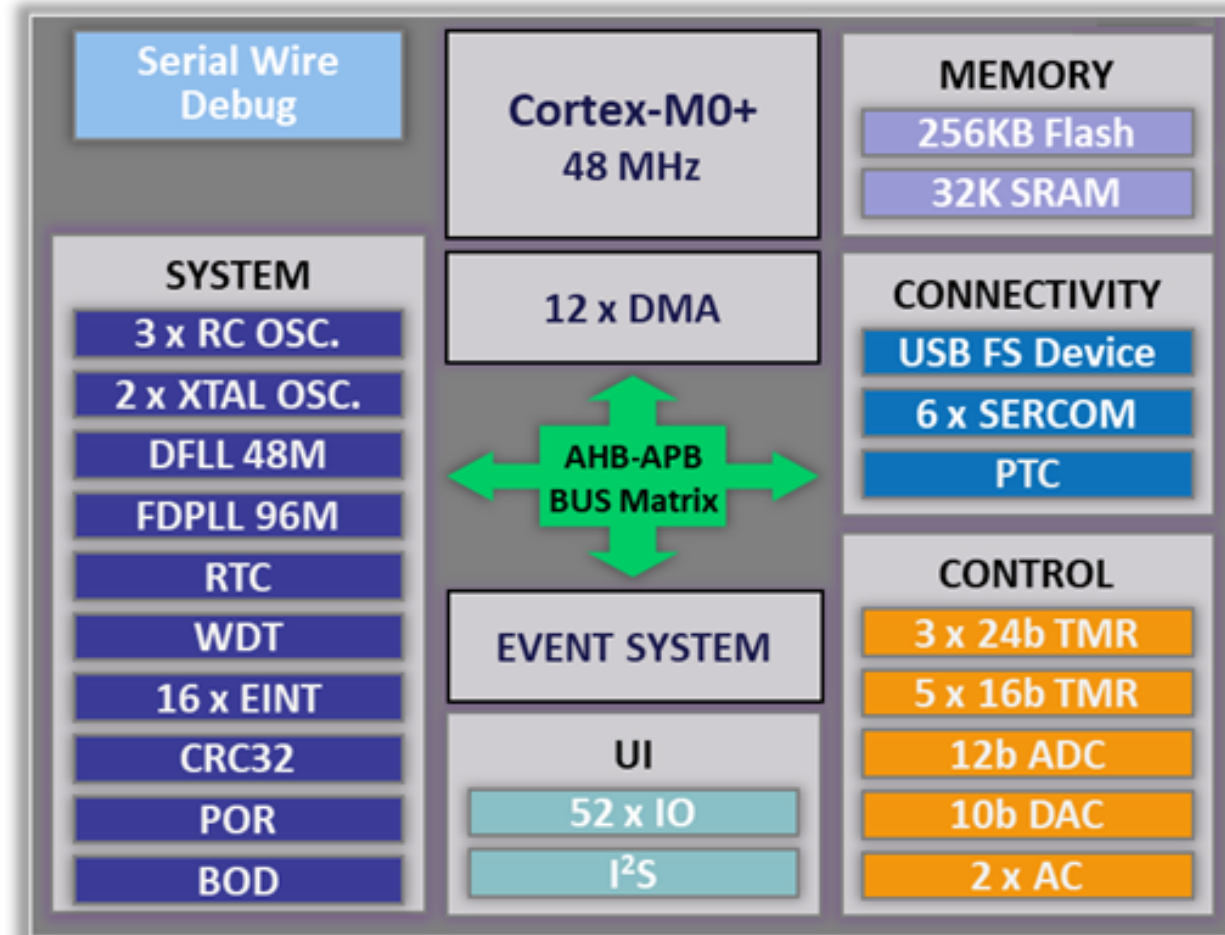
- SAMV7x

- Ethernet support, CAN-FD support, automotive certificated.



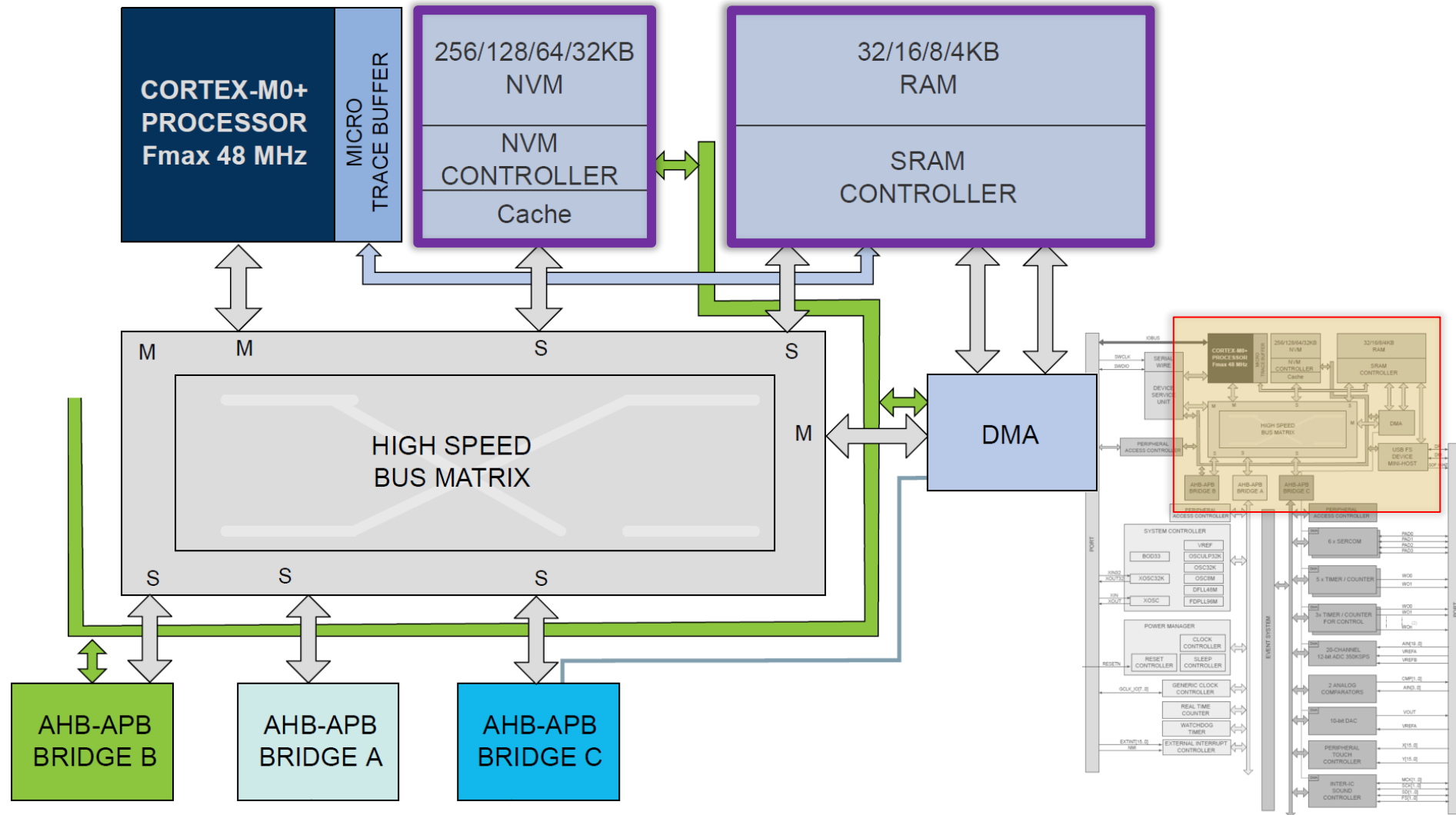
SAMD21 Block Diagram

- The SAMD21 is a series of low-power microcontrollers using the ARM® Cortex® M0+ processor (Cortexv6-M) ◦
- Maximum frequency of 48MHz and reach 2.46 CoreMark®/ MHz ◦
- All devices include intelligent and flexible peripherals, Event System for inter-peripheral signaling, and support for capacitive touch button, slider and wheel user interfaces ◦



Architecture

- Cortex M0+ is **Von Neumann** architecture, RISC



Architecture

- Cortex M0/M0+ **Von Neumann**

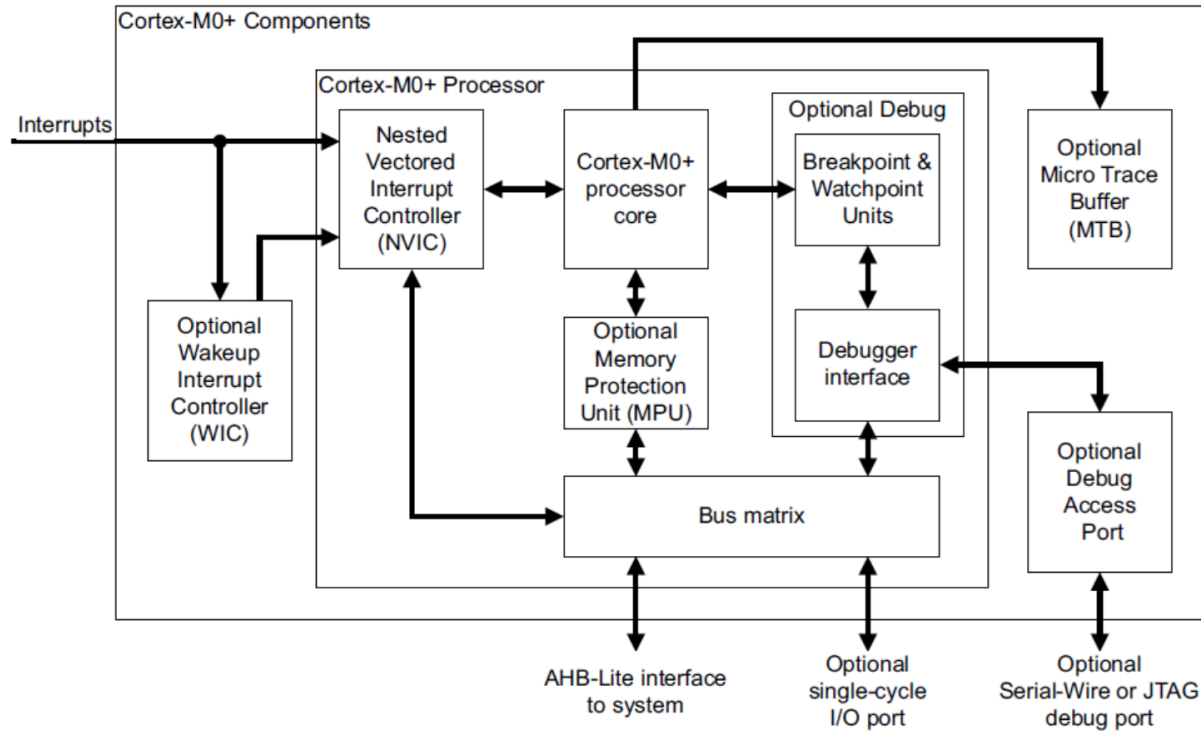


Figure 1-1 Cortex-M0+ implementation

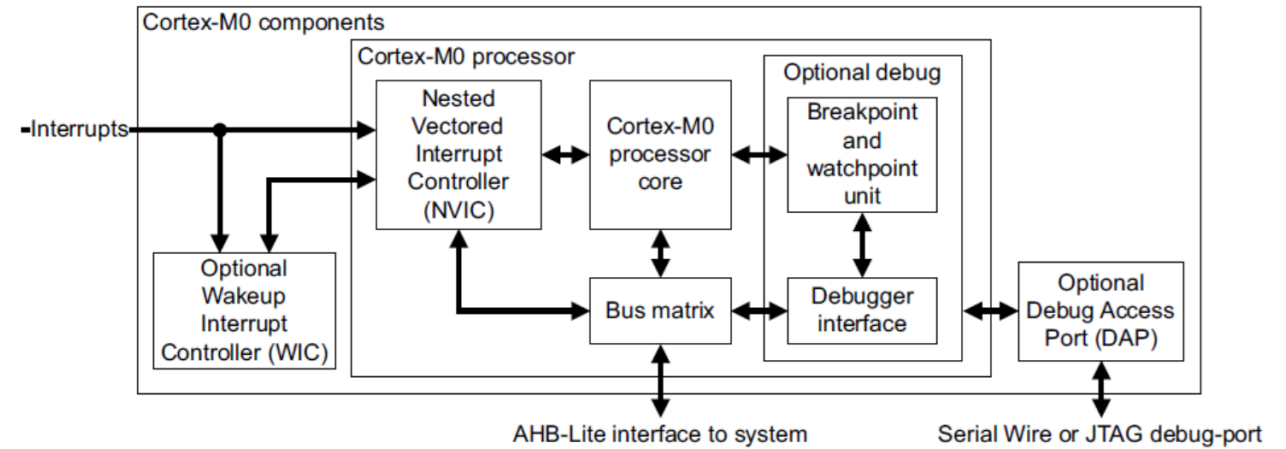


Figure 1-1 Cortex-M0 implementation

Architecture

- Cortex M3 **Harvard**

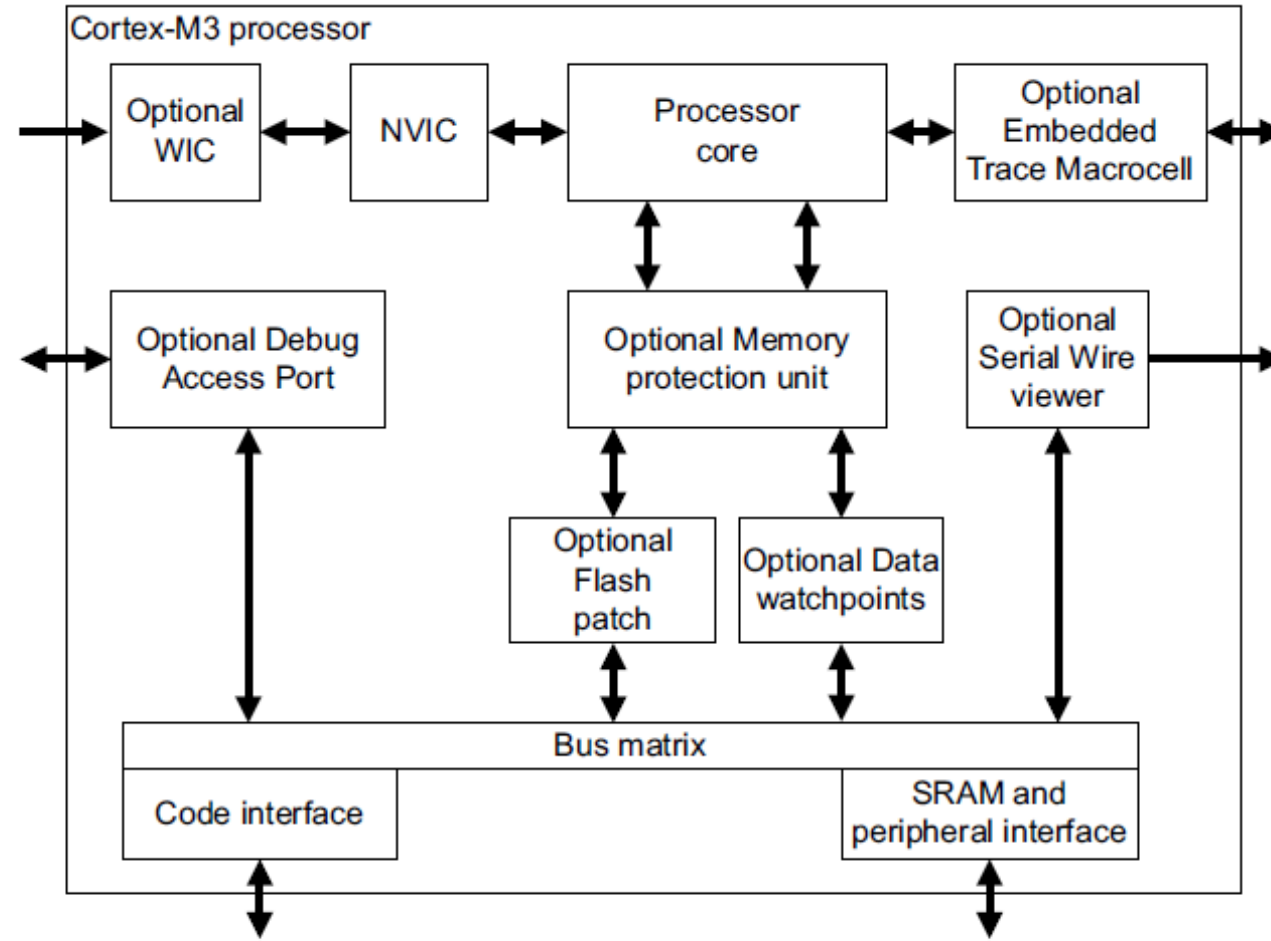


Figure 1-1 Cortex-M3 implementation

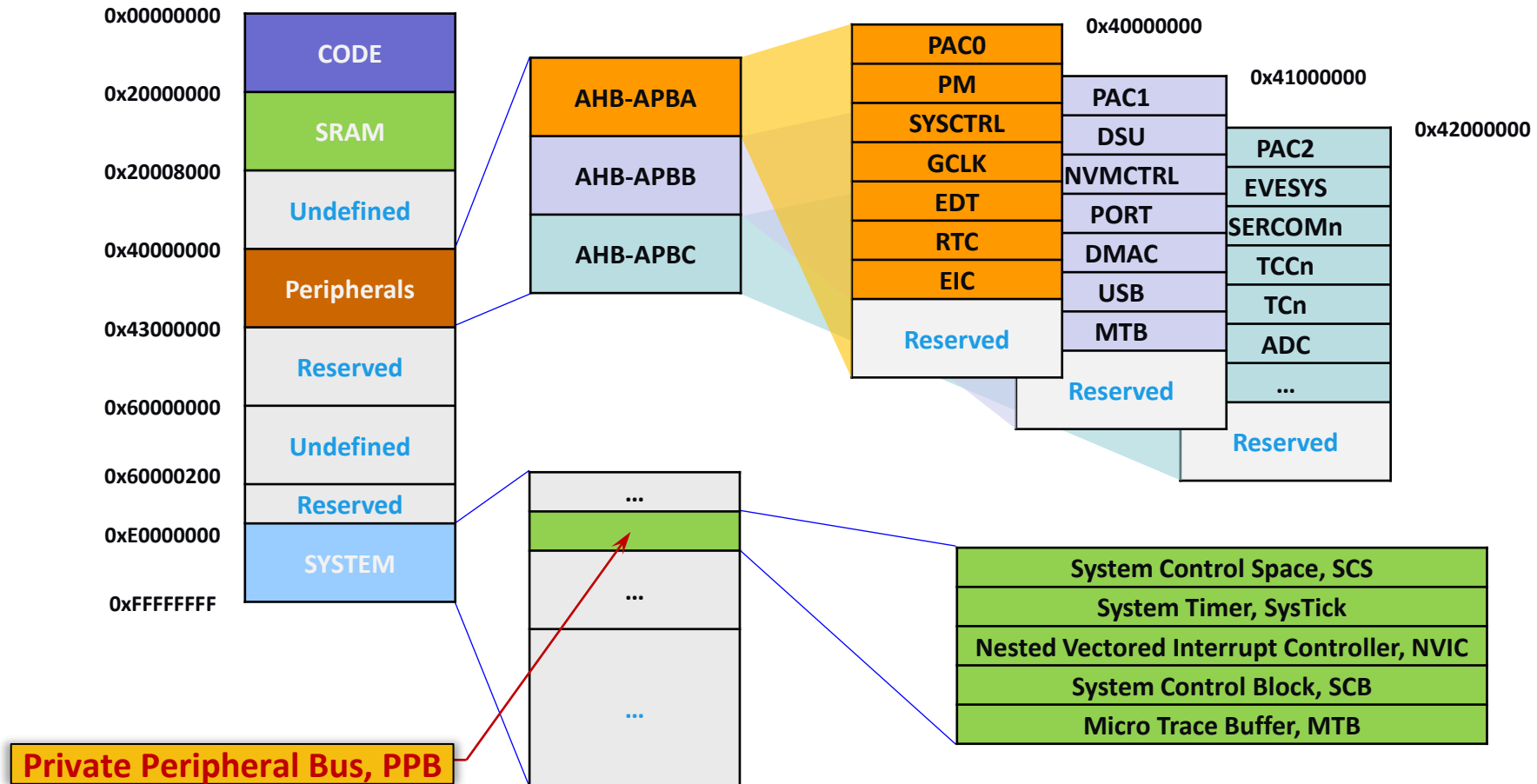
Architecture

ARM Cortex-M instruction variations										
Arm Core	Cortex M0 ^[2]	Cortex M0+ ^[3]	Cortex M1 ^[4]	Cortex M3 ^[5]	Cortex M4 ^[6]	Cortex M7 ^[7]	Cortex M23 ^[8]	Cortex M33 ^[12]	Cortex M35P	Cortex M55
ARM architecture	ARMv6-M ^[9]	ARMv6-M ^[9]	ARMv6-M ^[9]	ARMv7-M ^[10]	ARMv7E-M ^[10]	ARMv7E-M ^[10]	ARMv8-M Baseline ^[15]	ARMv8-M Mainline ^[15]	ARMv8-M Mainline ^[15]	Arm v8.1-M
Computer architecture	Von Neumann	Von Neumann	Von Neumann	Harvard	Harvard	Harvard	Von Neumann	Harvard	Harvard	Harvard
Instruction pipeline	3 stages	2 stages	3 stages	3 stages	3 stages	6 stages	2 stages	3 stages	3 stages	4 to 5 stages
Thumb-1 instructions	Most	Most	Most	Entire	Entire	Entire	Most	Entire	Entire	Entire
Thumb-2 instructions	Some	Some	Some	Entire	Entire	Entire	Some	Entire	Entire	Entire
Multiply instructions 32x32 = 32-bit result	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multiply instructions 32x32 = 64-bit result	No	No	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Divide instructions 32/32 = 32-bit quotient	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Saturated instructions	No	No	No	Some	Yes	Yes	No	Yes	Yes	Yes
DSP instructions	No	No	No	No	Yes	Yes	No	Optional	Optional	Optional
Single-Precision (SP) Floating-point instructions	No	No	No	No	Optional	Optional	No	Optional	Optional	Optional
Double-Precision (DP) Floating-point instructions	No	No	No	No	No	Optional	No	No	No	Optional
Half-Precisions (HP)	No	No	No	No	No	No	No	No	No	Optional
TrustZone instructions	No	No	No	No	No	No	Optional	Optional	Optional	Optional
Co-processor instructions	No	No	No	No	No	No	No	Optional	Optional	Optional
Helium technology	No	No	No	No	No	No	No	No	No	Optional
Interrupt latency (if zero-wait state RAM)	16 cycles	15 cycles	23 for NMI 26 for IRQ	12 cycles	12 cycles	12 cycles 14 worst case	15 no security ext 27 security ext	12 no security ext ?? security ext	TBD	TBD

Link : https://en.wikipedia.org/wiki/ARM_Cortex-M

SAMD21 Memory

- Up to 256KB Flash and 32KB SRAM



Part Number

SAMD 21 G 18 A - A U T

Product Family

SAMD = General Purpose Microcontroller

Product Series

21 = Cortex M0 + CPU, Basic Feature Set
+ DMA + USB

Pin Count

E = 32 Pins (35 Pins for WLCSP)
G = 48 Pins (45 Pins for WLCSP)
J = 64 Pins

Flash Memory Density

18 = 256KB
17 = 128KB
16 = 64KB
15 = 32KB

Device Variant

A = Default Variant
B = Added RWW support for 32KB and 64KB memory options
C = Silicon revision F for WLCSP35 package option.

Package Carrier

No character = Tray (Default)
T = Tape and Reel

Package Grade

U = -40 - 85°C Matte Sn Plating
F = -40 - 125°C Matte Sn Plating

Package Type

A = TQFP
M = QFN
U = WLCSP
C = UFBGA

Soft Request In this Course

- **MPLAB® X IDE v6.05** <https://www.microchip.com/mplab/mplab-x-ide>
- **MPLAB® XC32 v4.21** <https://www.microchip.com/mplab/compilers>
- **MPLAB® Code Configurator v5.3.0**
- **MPLAB® MCC Harmony Framework (From X IDE MCC Harmony Content Manager)**
 - csp v3.16.0
 - Harmony-services v1.2.0
 - Quick_docs v1.6.0
 - dev_packs v3.15.1
- **SAMD21 DFP v3.6.144 (Device Family Pack) (From X IDE > Tools > Packs)**
- **CMSIS v5.8.0 (Common Microcontroller Software Interface Standard)**
- **SNAP tool packs v1.14.1052**

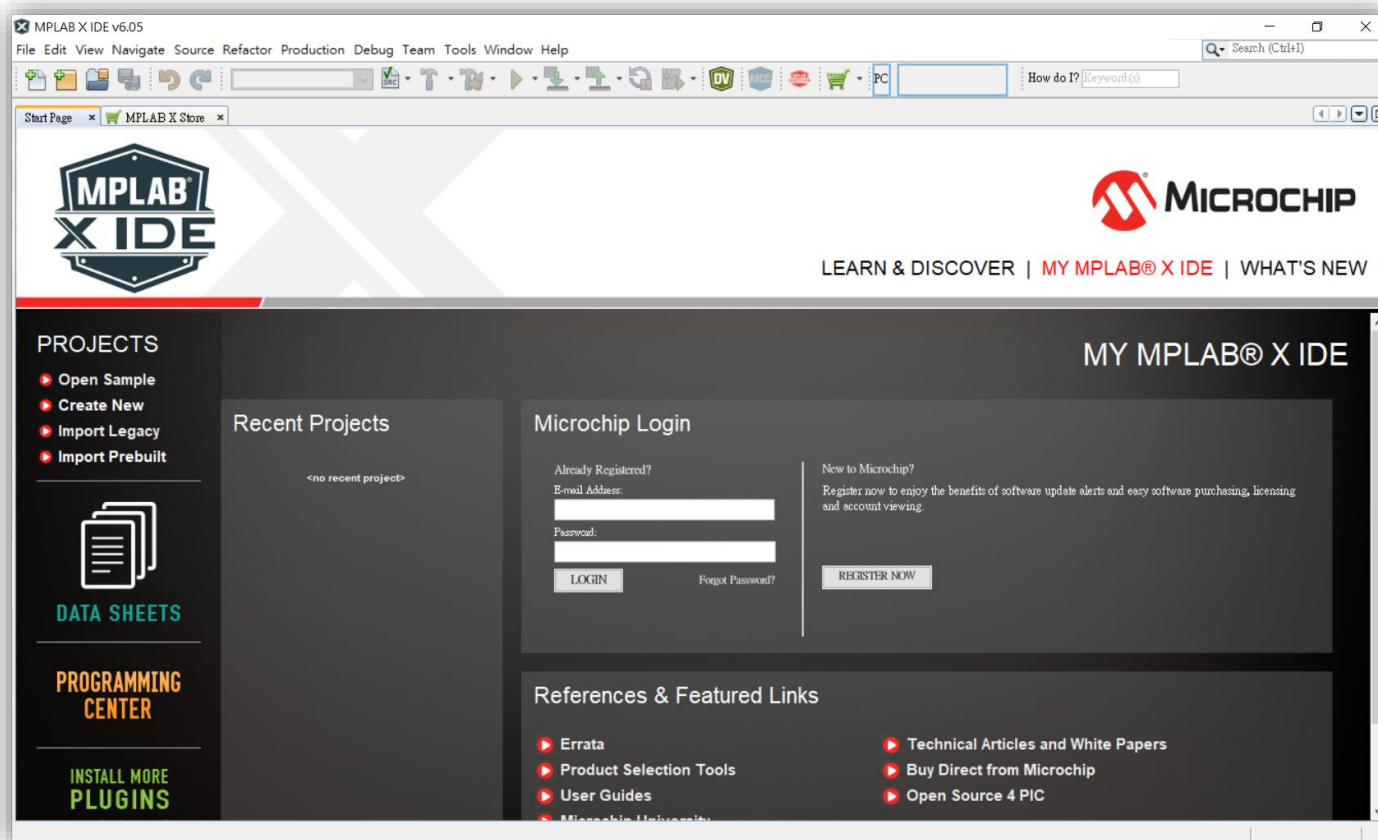
(Lasted Update : 2023/03/30)

MPLAB® X IDE, MCC Harmony and Development Tools Introduction

All in One Development

MPLAB® X IDE

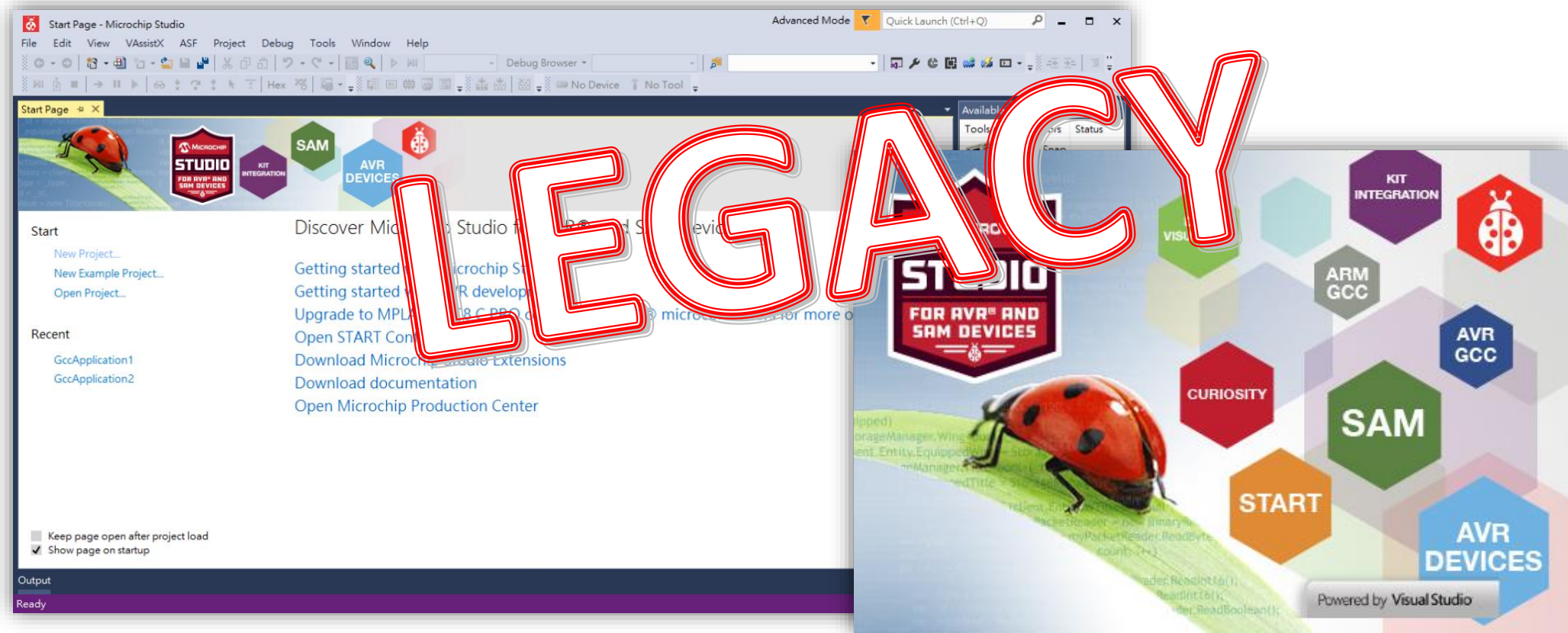
- New generation integrated Development Tools, Support PIC, dsPIC, AVR, CEC and SAM Series MCU/MPU, Provide Plug-in function to extend more advance function. Java Based, Cross platform, Current version is v6.05.



AVR/SAM Series Development

Atmel Studio 7 → Microchip Studio

- New generation integrated Development Tools, Support AVR, SAM Series MCU/MPU, Current version is v7.0.2594.
- The GCC compilers already include support AVR/SAM series.



Official Site Download Page

www.microchip.com/mplabX

 [Products](#) [Solutions](#) [Tools and Resources](#) [Support](#) [Education](#) [About](#) [Order Now](#)   

Tools and Resources / Develop / MPLAB® X IDE

Downloads and Documentation Ecosystem

Downloads, Documentation and Other Resources

Downloads

Features

Documentation

Debug Features

MPLAB X IDE v6.05

MPLAB X IDE only supports computers with processors designed with Intel® 64 or AMD® 64-bit architectures.

Title		Version Number	Date	Download
MPLAB X IDE (Windows)	  3241f56b... 9501	6.05	31 Oct 2022	Download
MPLAB X IDE (Linux)	  7bd2060a... dadd	6.05	31 Oct 2022	Download
MPLAB X IDE (macOS)	  13fd0da9... d696	6.05	31 Oct 2022	Download
MPLAB X IDE Release Notes		6.05	31 Oct 2022	Download

Go to Downloads Archive

Programmer/Debugger

PICkit 4 (Part No. PG164140)

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

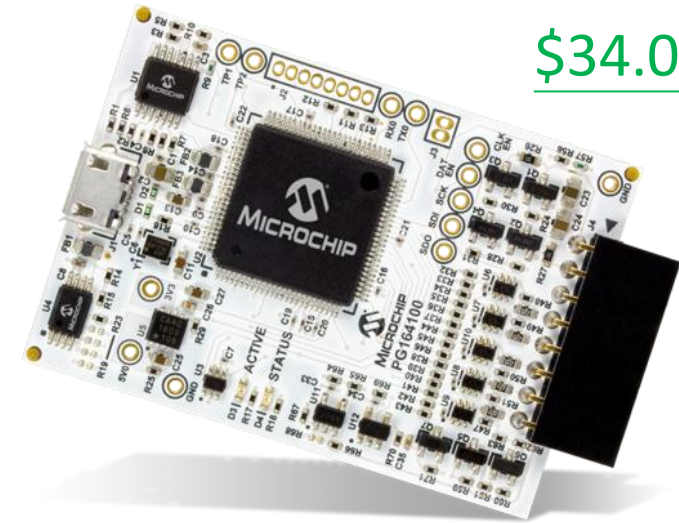


\$76.99

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



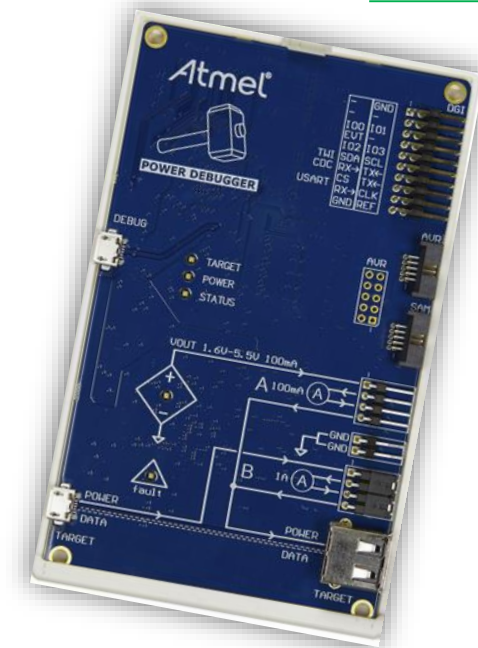
\$34.09

Programmer/Debugger

Power Debugger (Part No. atpowerdebugger)

- **Simultaneous debug & power measurement**
 - Debugging over CMSIS-DAP
 - Extended to support SWO trace, same as Atmel-ICE
- **Two independent channels for power measurements**
 - Channel A for high accuracy, lower currents
 - Channel B for higher currents with lower resolution
- **CDC virtual COM port**
- **DGI for streaming application & power data**
 - Open specification, docs provided
- **Provides target power, 1.6 – 5.5V, up to 100mA**
- **Status LEDs for debugging & target supply**

\$252.99



Programmer/Debugger

Atmel-ICE (Part No. ATATMEL-ICE)

- **Supports programming and debugging all Cortex-M based SAM, AVR and UC3 based parts**
 - Programming/ Debugging AVR/SAM MCU/CPU over JTAG, SD, PDI, SPI, TPI and aWire etc.
 - Target operating voltage range 1.62V to 5.5V
 - ITM serial trace support on Cortex-M based SAM MCUs
- **Supported in Atmel Studio, IAR, Keil and more**
 - Any IDE that supports the CMSIS-DAP protocol can support Atmel-ICE
- **3 different editions**
 - USD 71.99 - Low cost PCBA only
 - USD 114.99 - Basic edition with most common cables
 - USD 160.99 - Including all cables and adapters



Programmer/Debugger

Debug Adapter Board (Part No. AC102015)

- The Debug Adapter Board Gives ICD 4 / PICkit 4 cable compatibility with Atmel-ICE, Power Debugger and J-Link style connectors. It supports JTAG, SWD and ICSP protocols in multiple connector formats.
- **Package Contents**
 - Debugger Adapter Board
 - Cable, FEMALE FLAT RIBBON IDC CONNECTOR, 15CM
 - Cable, FEMALE FLAT RIBBON IDC CONNECTOR, 12CM
 - 8 CONDUCTOR MODULAR CABLE, 6"
 - Cable, HCSD Dual Row Socket IDC Assembly 14pin (7 ea) 10"

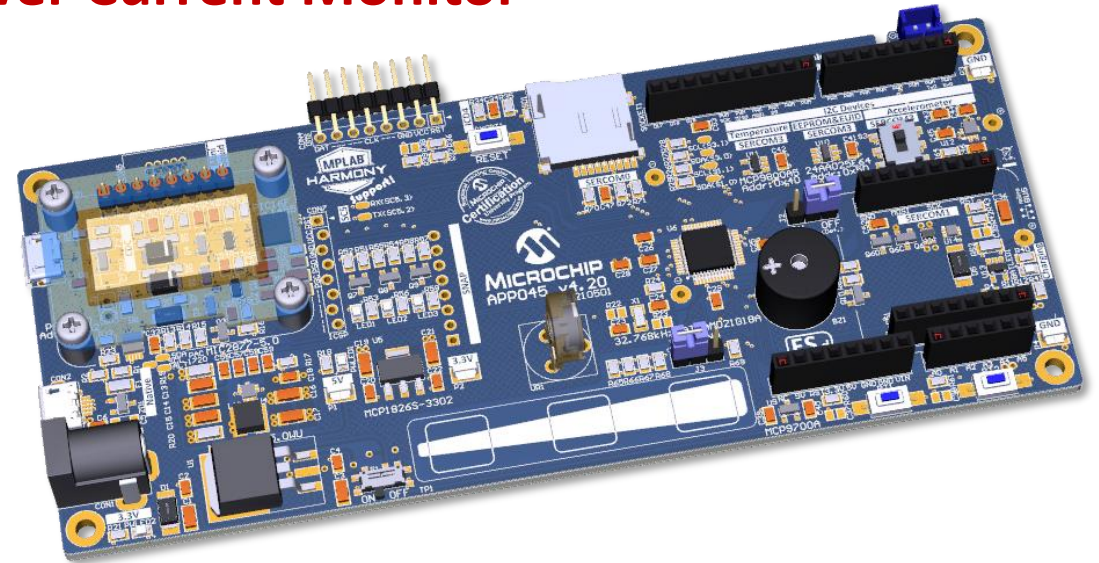


SAMD21 EVB

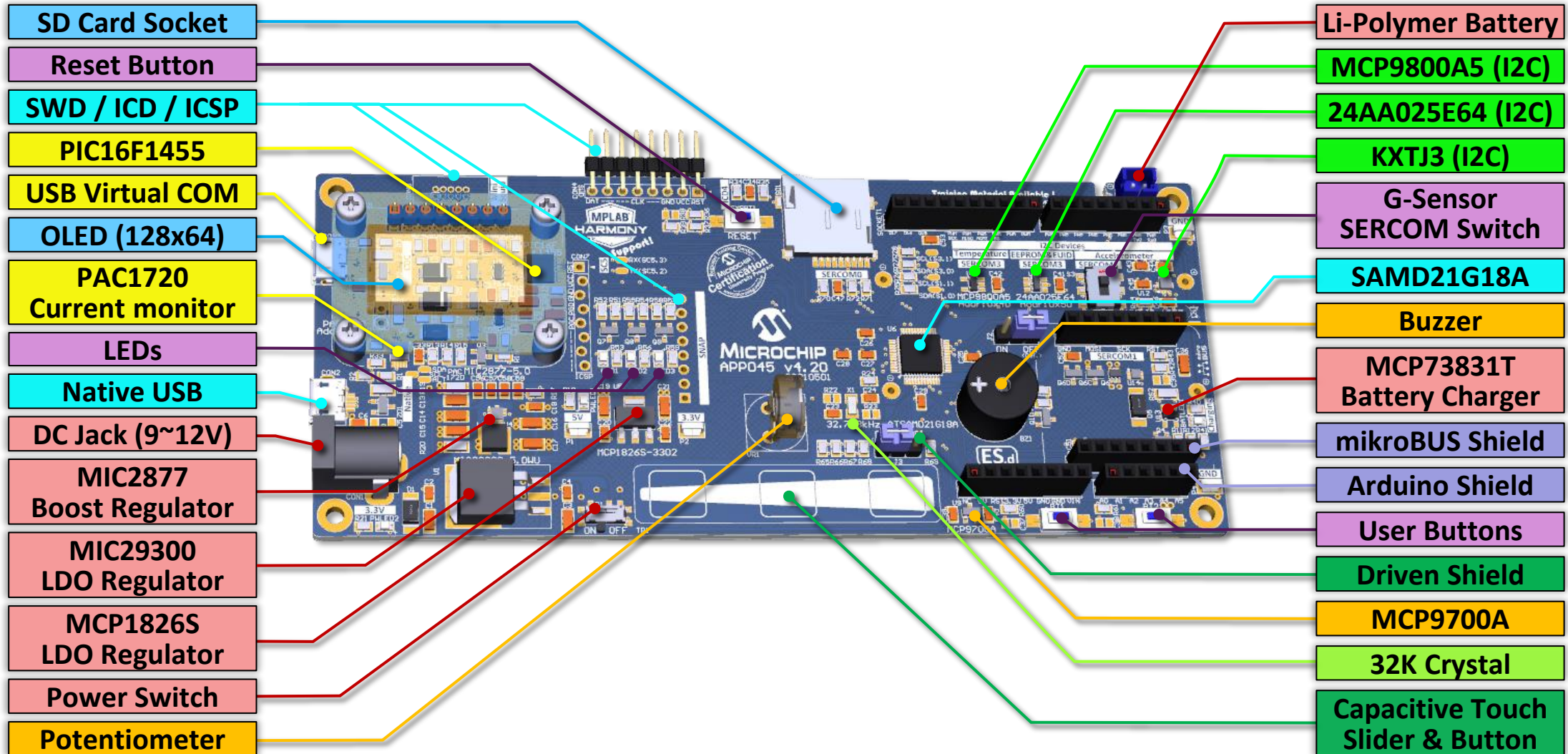
APP045 v4.2 ES.d Introduction

APP045 v4.2 ES.d Features

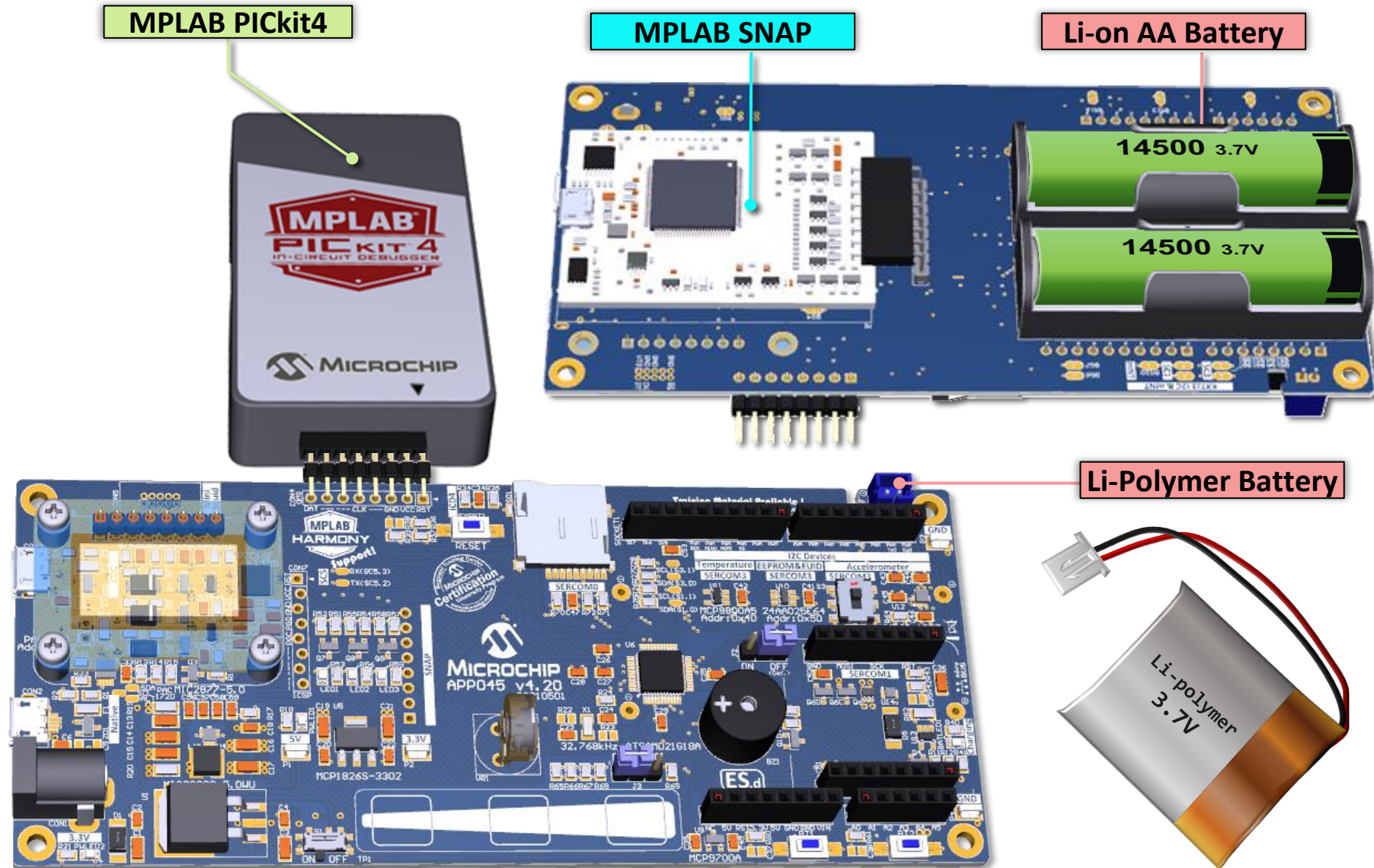
- On board **SAMD21G18A** 32Bit ARM Cortex M0+ Microcontroller
- Support plugged **MPLAB SNAP** external programmer/debugger
- Onboard **USB CDC Virtual COM Port** and **Power Current Monitor**
- Multiple Power Supply Source
- **Support Li-on Battery, boost and charger**
- 2 x Buttons, 3 x LEDs
- 1 x I2C Temperature Sensor
- 1 x I2C EEPROM with Unique ID
- **1 x I2C Accelerometer**
- 1 x MicroSD Socket (SPI interface)
- 1 x SPI OLED (128 x 64, SSD1306 Compliant)
- 1 x Potentiometer, Analog Temperature Sensor and **Buzzer**
- **1 x Capacitive QTouch® Slider & Buttons with Driven Shield**
- **1 x mikroBUS™ Click Board Socket**
- 1 x Arduino® Shield Socket (Arduino M0 Pro board Compliant)



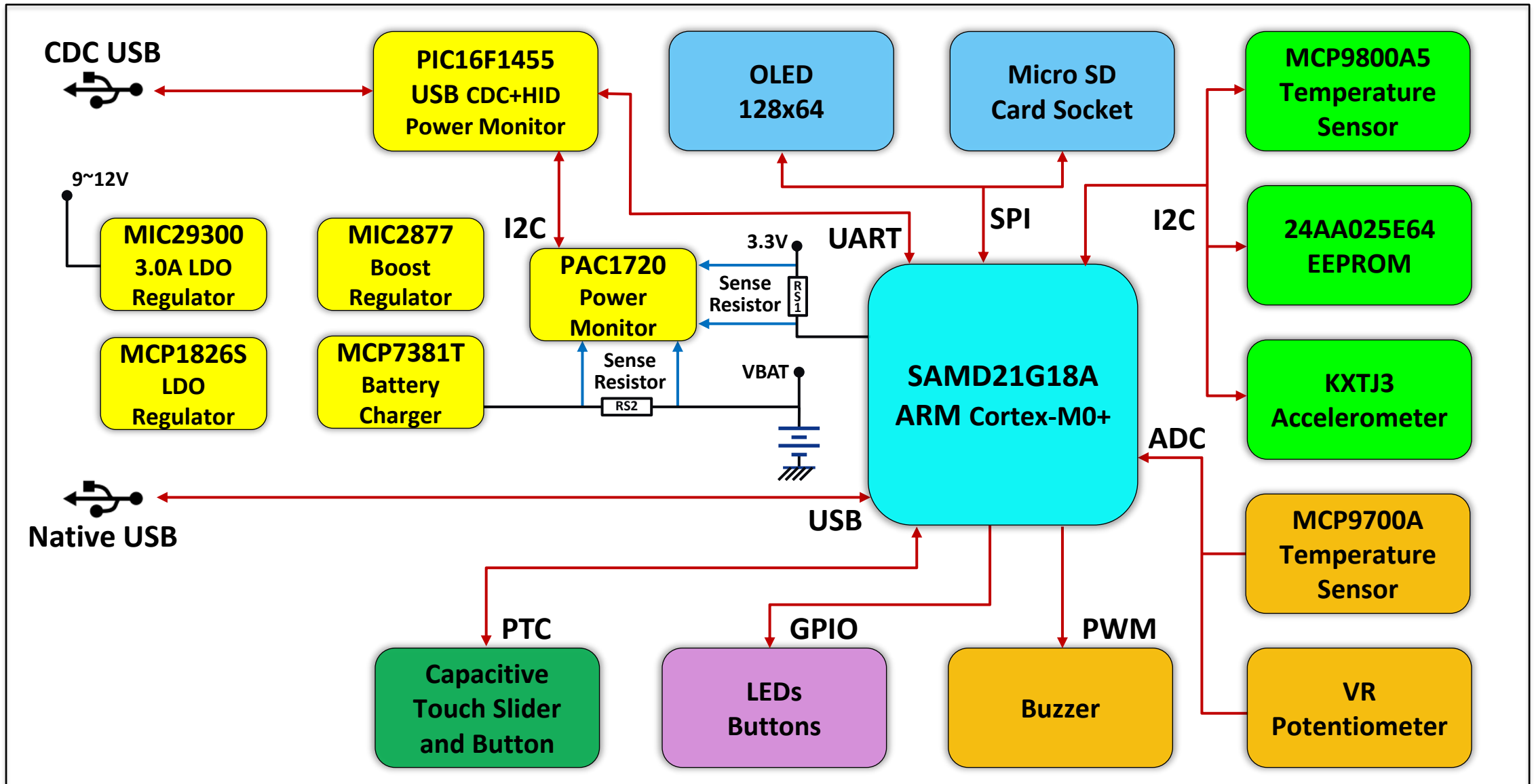
APP045 v4.2 ES.d Overview



APP045 v4.2 ES.d Programmer and Debugger



APP045 v4.2 ES.d Block Diagram



Getting Started with First Project

Lab0 First Project

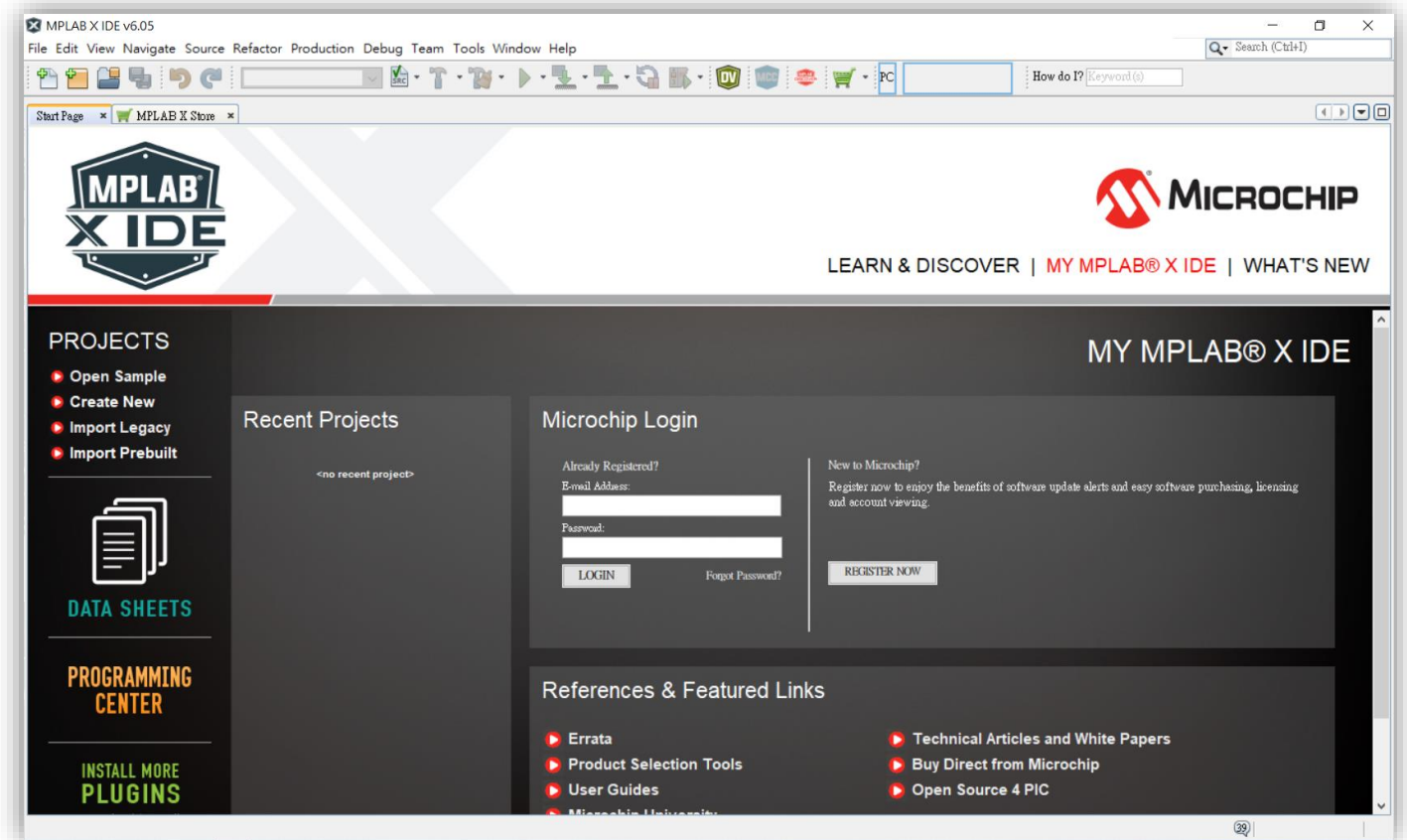
- Try to create your first project to use MPLAB X IDE.
- Learn how to use edit, build, project manager, debug and programming functions.

• **How to Start ?**

Lab0 Create Project

Step 1

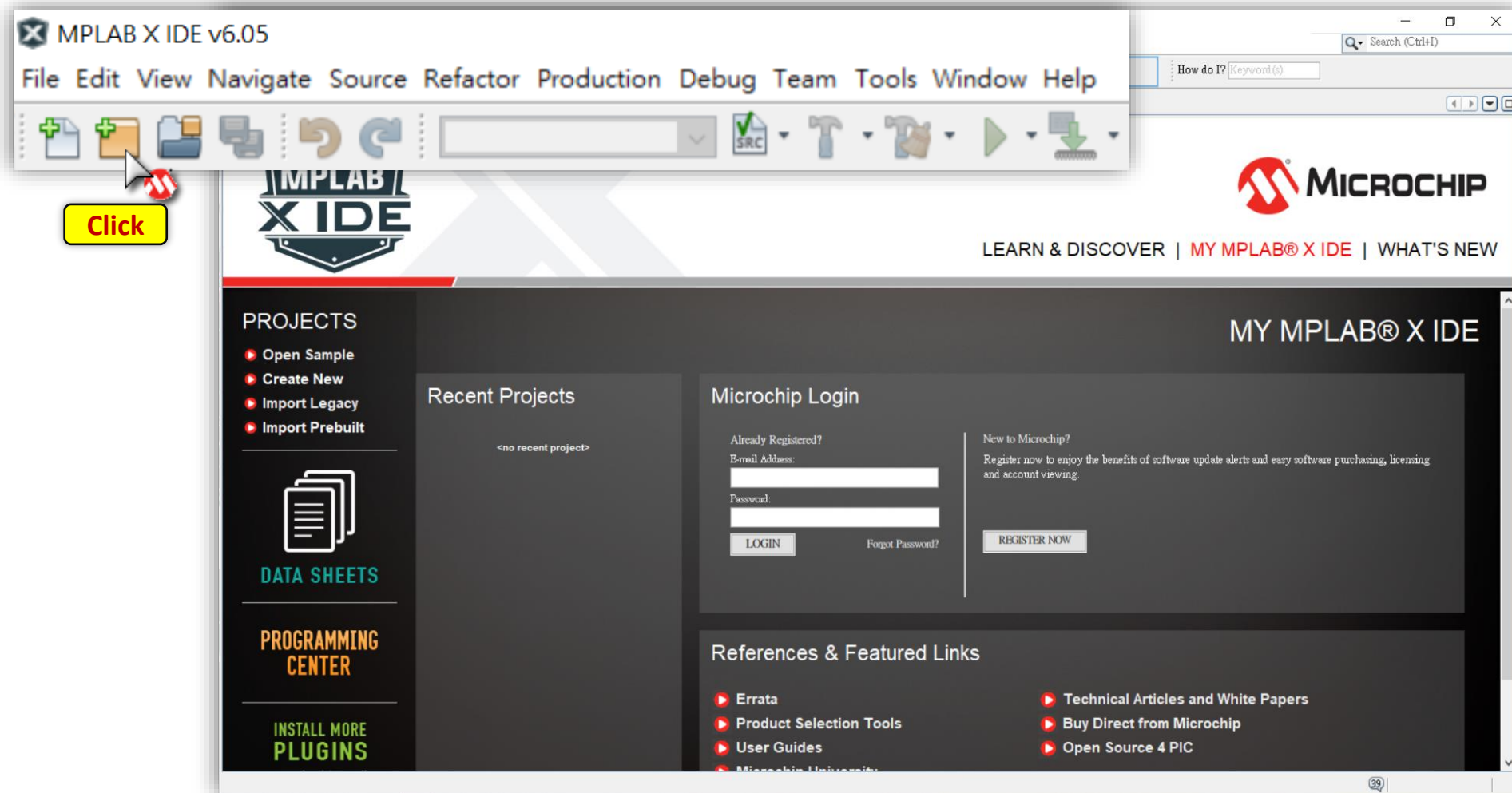
- Open **MPLAB X IDE v6.05** firstly.



Lab0 Create Project

Step 2

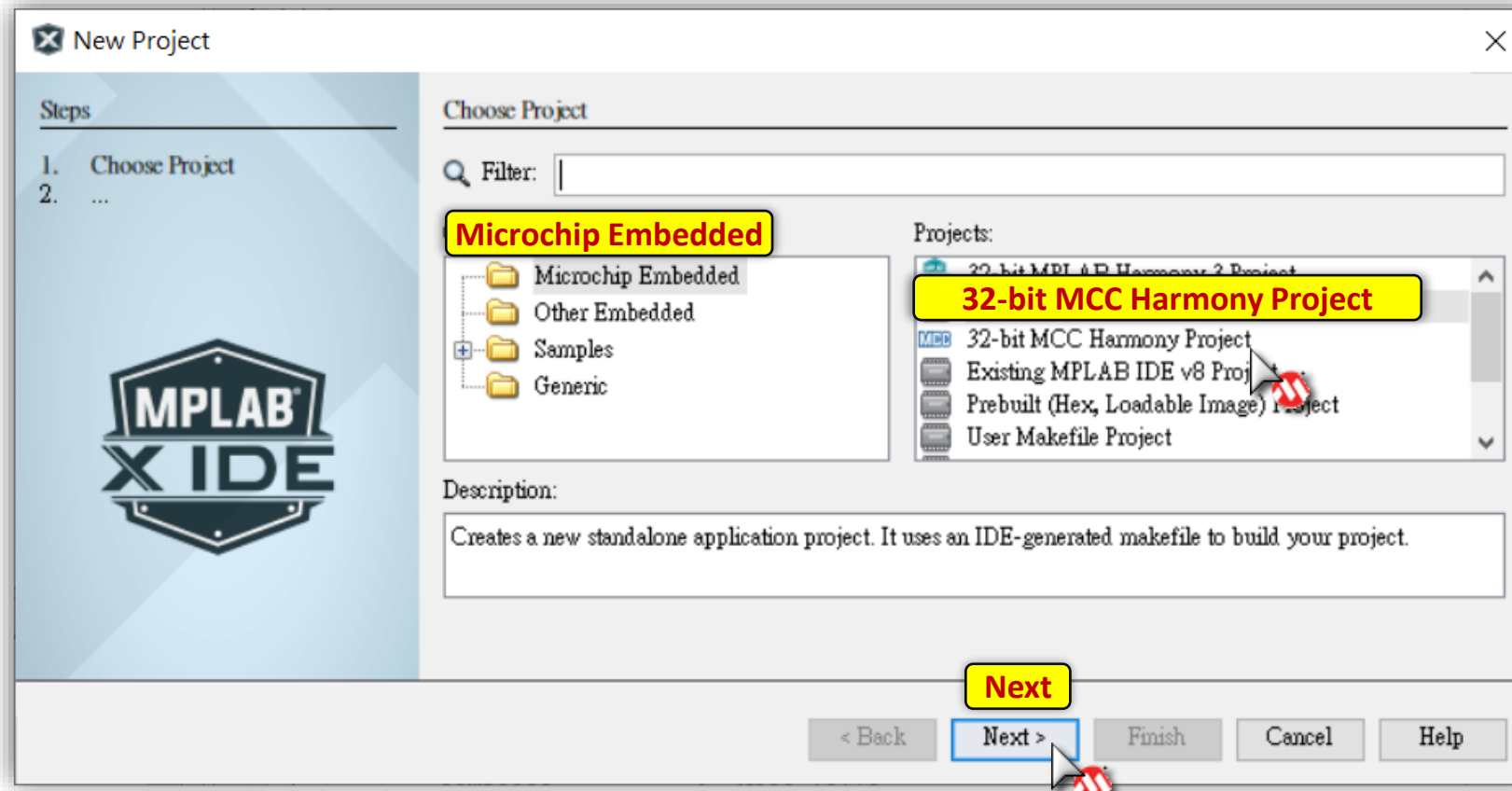
- Select to **File > New > Project** or Click icon 



Lab0 First Project

Step 3

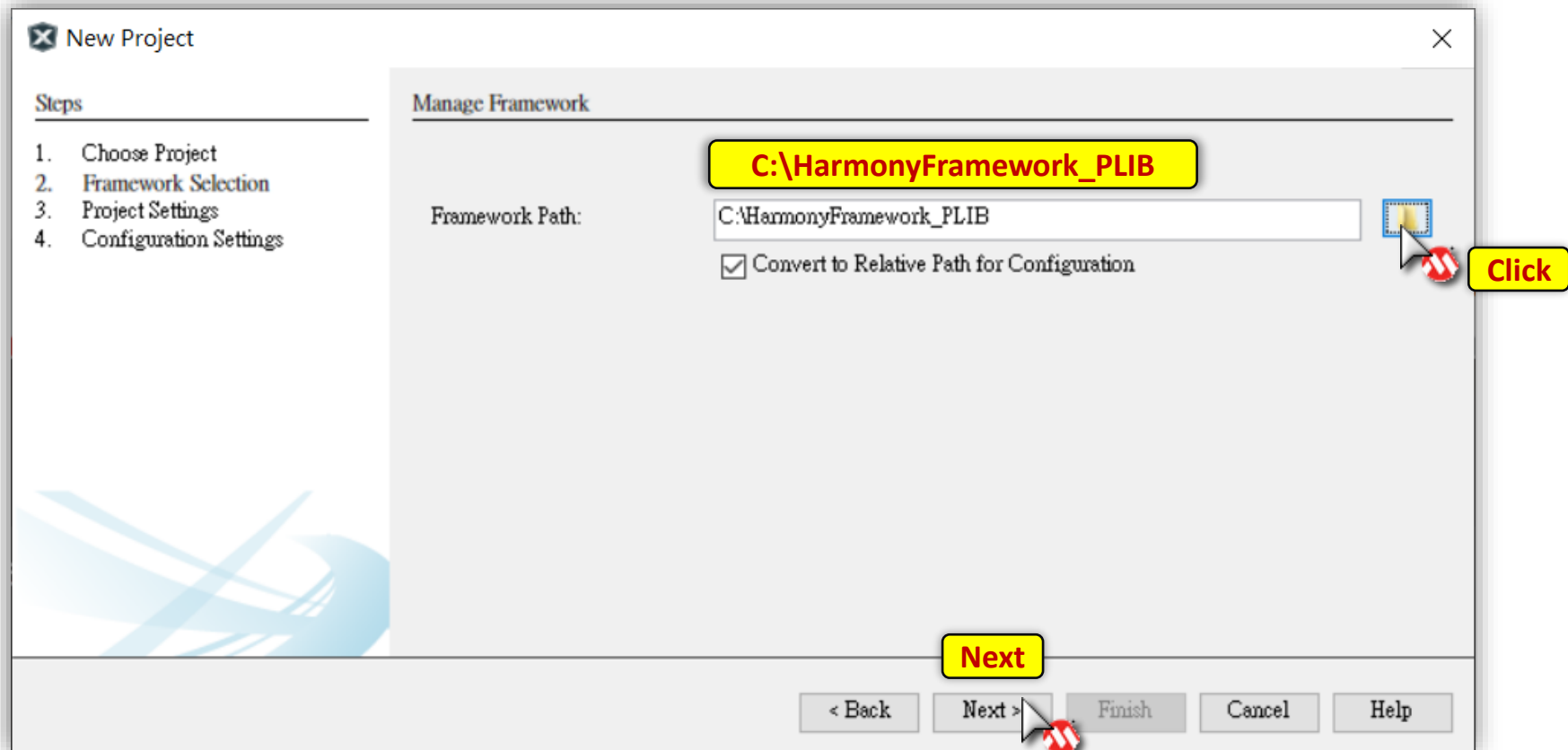
- Categories : **Microchip Embedded**
- Projects : **32-bit MCC Harmony Project**



Lab0 First Project

Step 4

- Assign MCC Harmony framework directory
 - Framework Path : **C:\HarmonyFramework_PLIB**



Lab0 Create Project

Step 5

- Location : **C:\Exercises\Lab0_First_Project** **Don't forget C:**
- Folder : **Lab0_First_Project** **No white space of project folder**
- Name : **Lab0_First_Project**
- Path : **C:\Exercises\Lab0_First_Project.X\firmware\Lab0_First_Project.X**

New Project

Steps

1. Choose Project
2. Framework Selection
3. Project Settings
4. Configuration Settings

Name and Location

Location: C:\Exercises\Lab0_First_Project

Folder: Lab0_First_Project

Name: Lab0_First_Project

Path: C:\Exercises\Lab0_First_Project\firmware\Lab0_First_Project.X

Show Visual Help

< Back Next > Finish Cancel Help

Wait a while after click Next

Lab0 Create Project

Step 6

- Name : default
- Device Family : **ATSAM** Target Device : ATSAM21G18A
- Device Filter : **ATSAMD21G18A** **Don't miss any character**

New Project

Steps

1. Choose Project
2. Framework Selection
3. Project Settings
4. **Configuration Settings**

Configuration Settings

Name: default **Don't change**

Device Family: **ATSAM** Target Device: **Auto selected** ATSAM21G18A

Device Filter: ATSAM21G18A **Input: ATSAM21G18A**

Show Visual Help

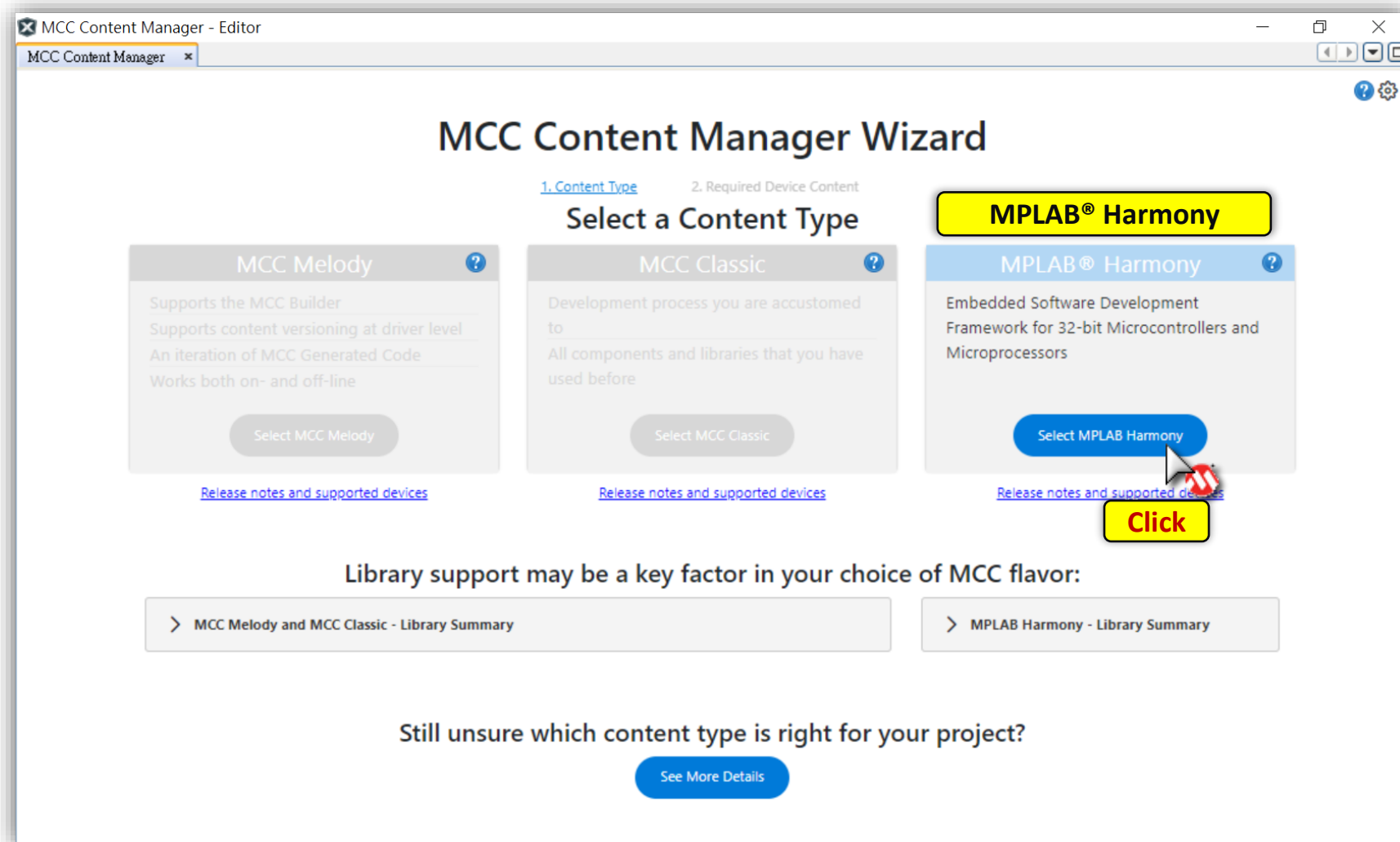
Finish

< Back Next > Finish Cancel Help

Lab0 Create Project

Step 7

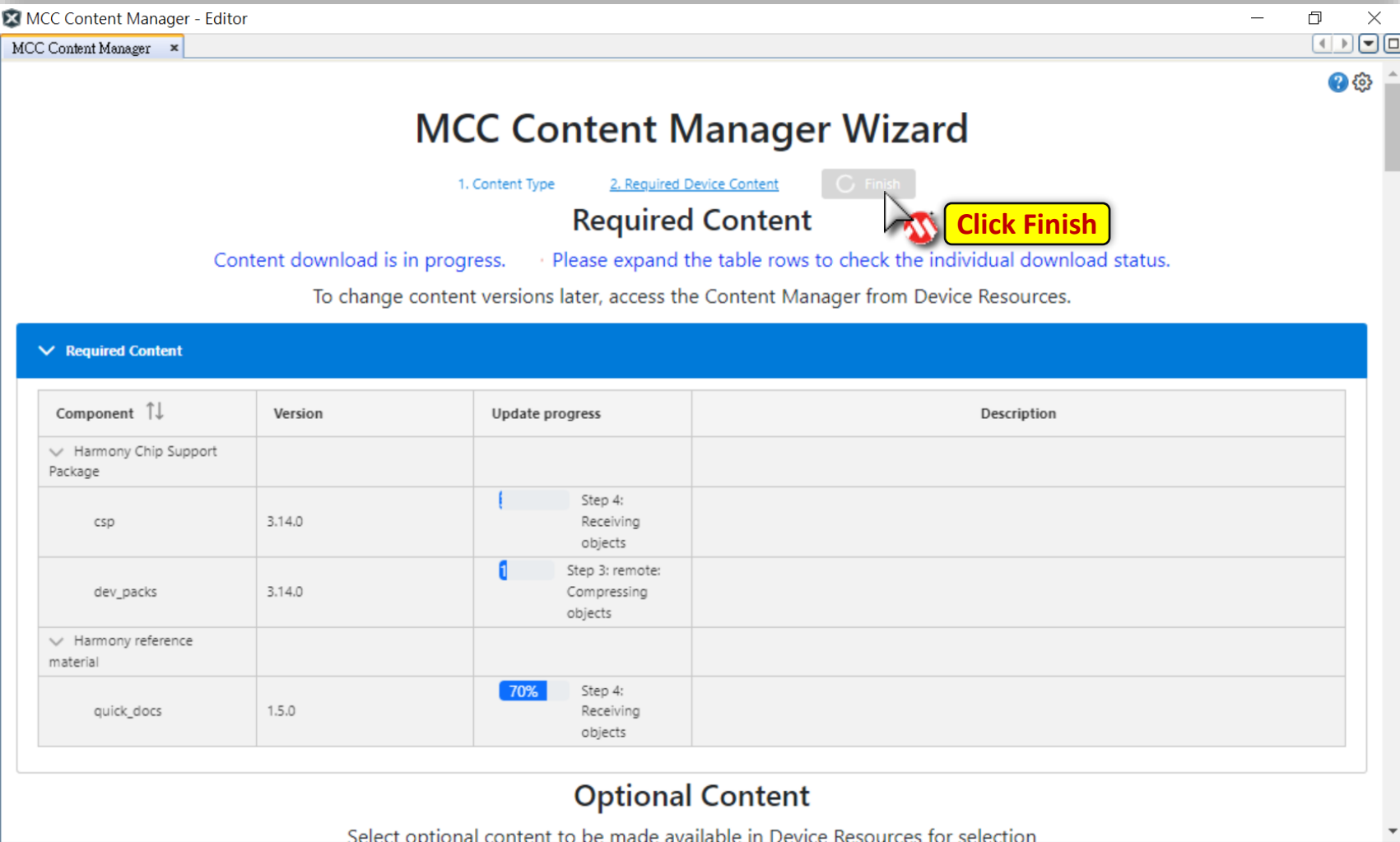
- Select **MPLAB Harmony**.



Lab0 Create Project

Step 8

- Click **Finish** and **Wait Download Framework**.



The screenshot shows the 'MCC Content Manager Wizard' window. The title bar reads 'MCC Content Manager - Editor'. The main heading is 'MCC Content Manager Wizard'. Below the heading, there are two steps: '1. Content Type' and '2. Required Device Content'. A 'Finish' button is visible next to step 2. A yellow callout box with a red arrow points to the 'Finish' button, containing the text 'Click Finish'. Below the steps, the text 'Required Content' is displayed. A status message says 'Content download is in progress. Please expand the table rows to check the individual download status.' Below this, a note states 'To change content versions later, access the Content Manager from Device Resources.'

The 'Required Content' section is expanded, showing a table with the following data:

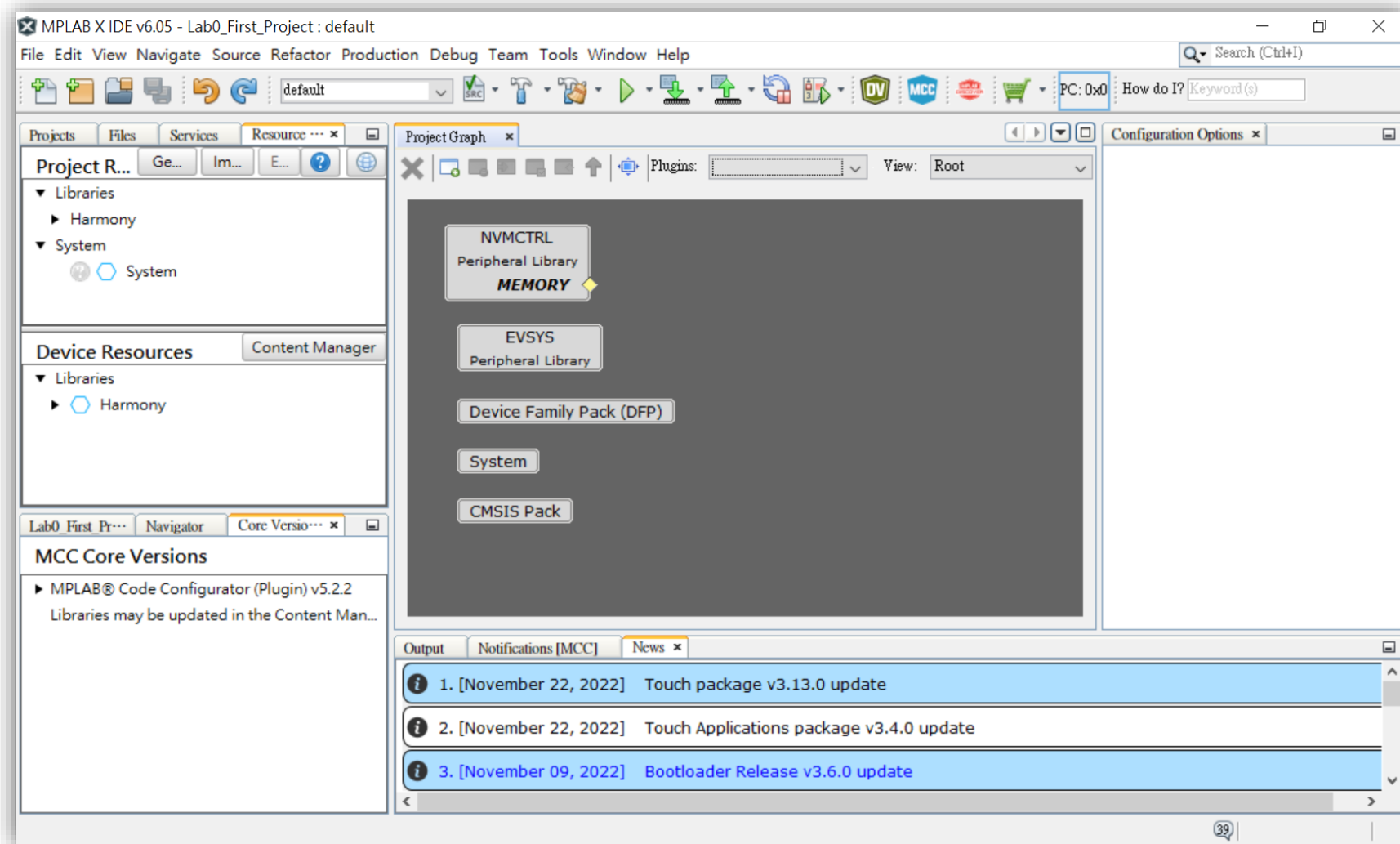
Component ↑↓	Version	Update progress	Description
Harmony Chip Support Package			
csp	3.14.0	Step 4: Receiving objects	
dev_packs	3.14.0	Step 3: remote: Compressing objects	
Harmony reference material			
quick_docs	1.5.0	70% Step 4: Receiving objects	

Below the table, the 'Optional Content' section is visible, with the text 'Select optional content to be made available in Device Resources for selection'.

Lab0 Create Project

Step 9

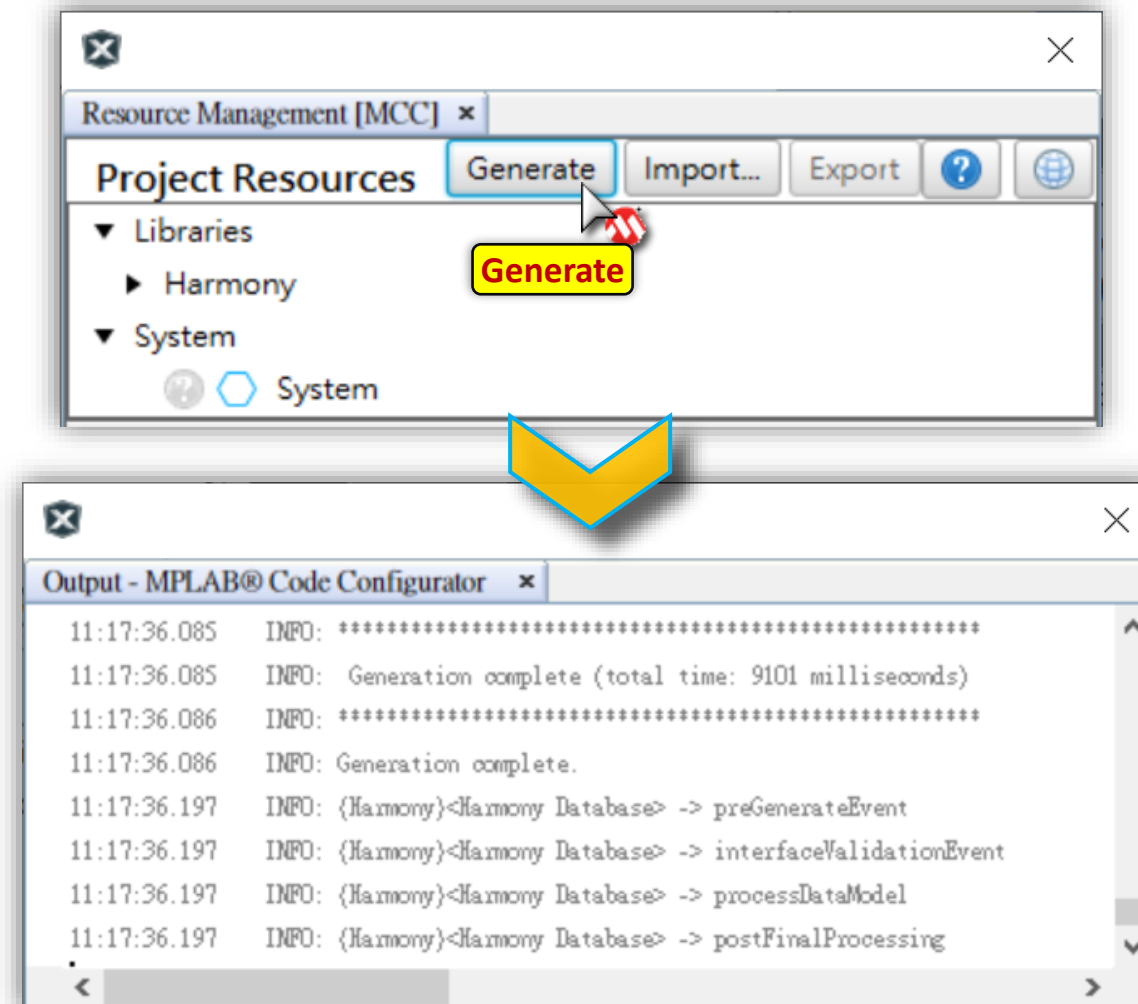
- Harmony Configurator interface show up, let's come back later.



Lab0 Generate Code

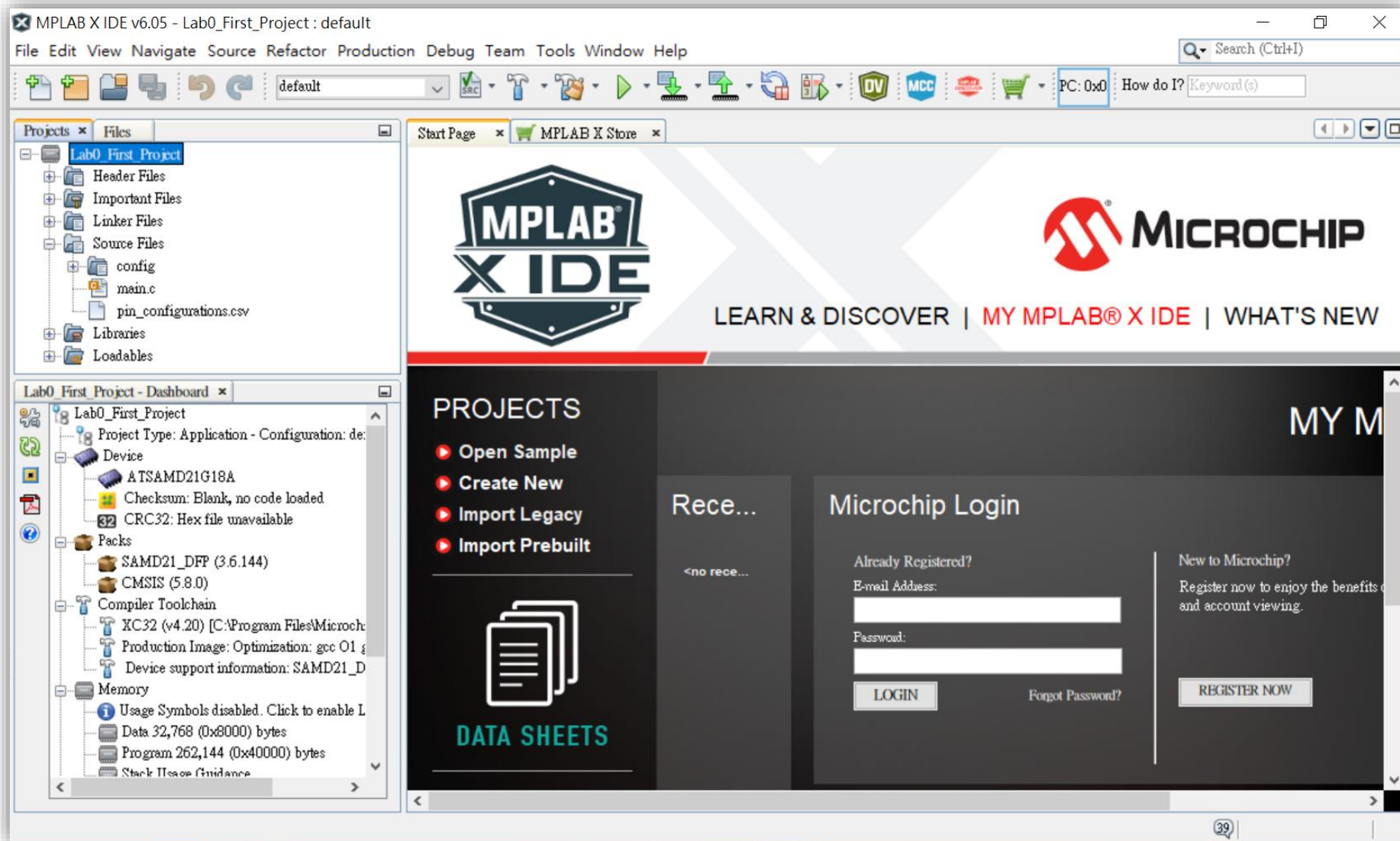
Step 10

- Click "**Generate**" Button to Generate your Code.



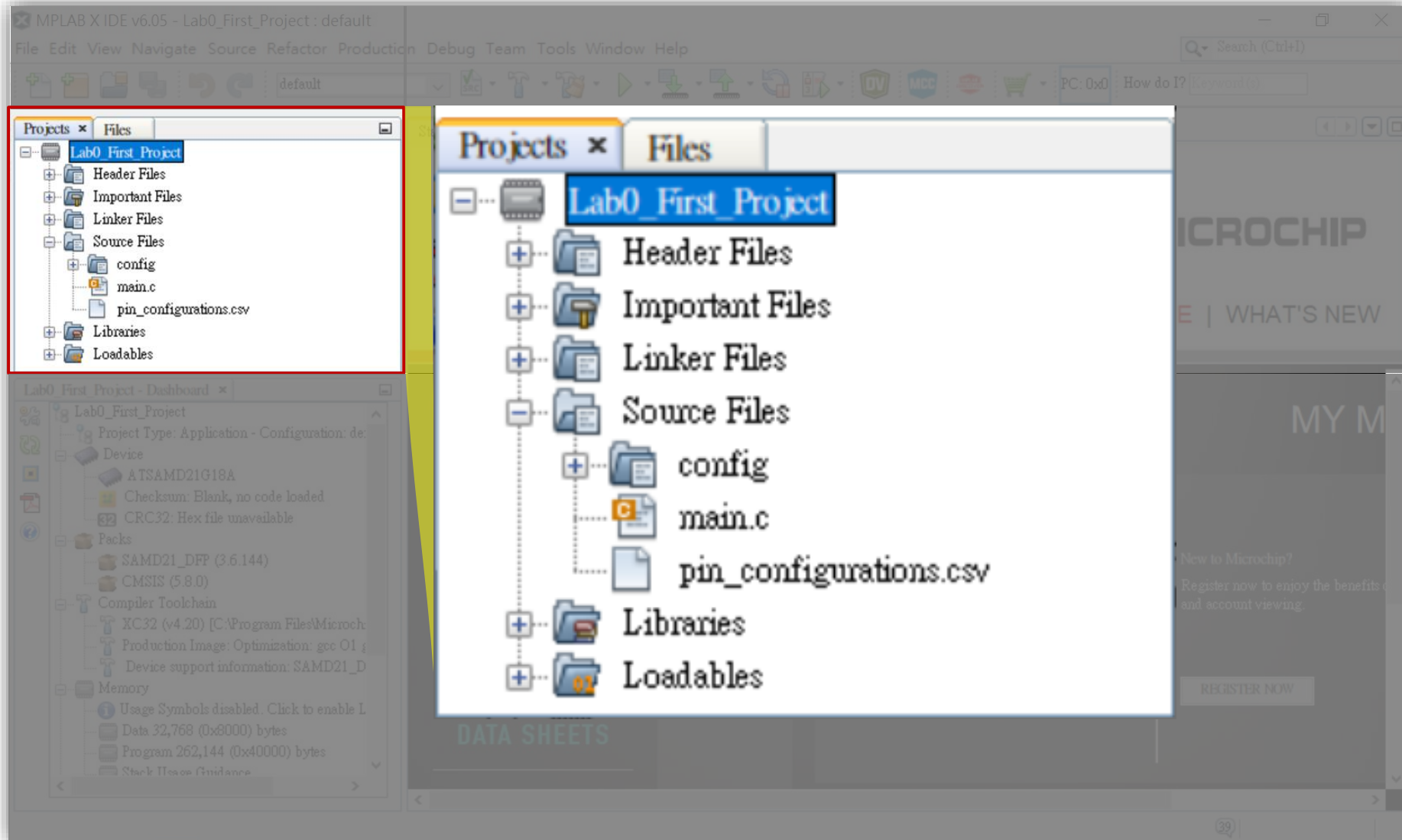
MPLAB X IDE Interface

- Return to MPLAB X IDE, let's go through its interface. (The version of modules might be different to yours.)



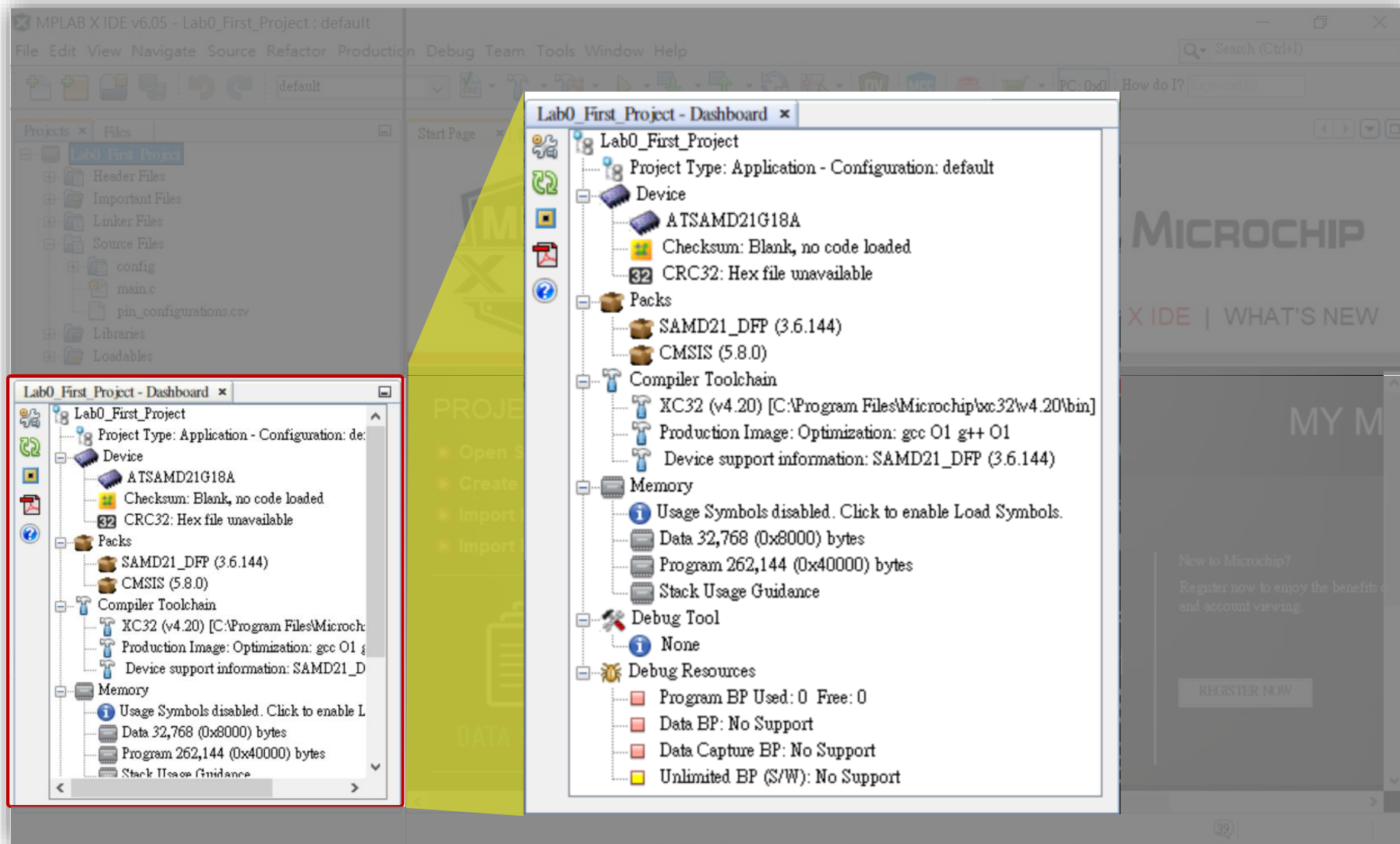
MPLAB X IDE Interface

- Project window.



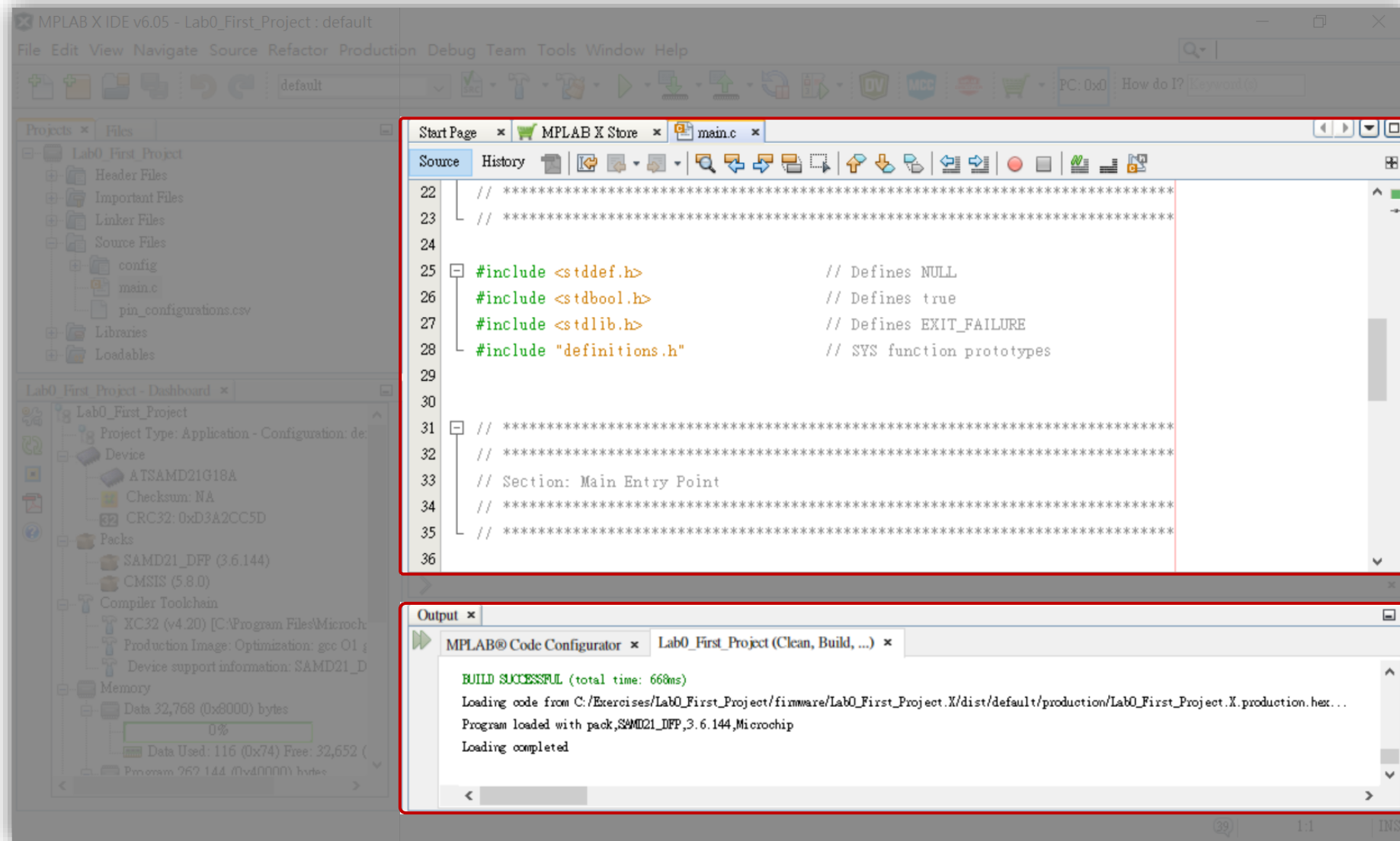
MPLAB X IDE Interface

- **Dashboard window.** (The version of modules might be different to yours.)



MPLAB X IDE Interface

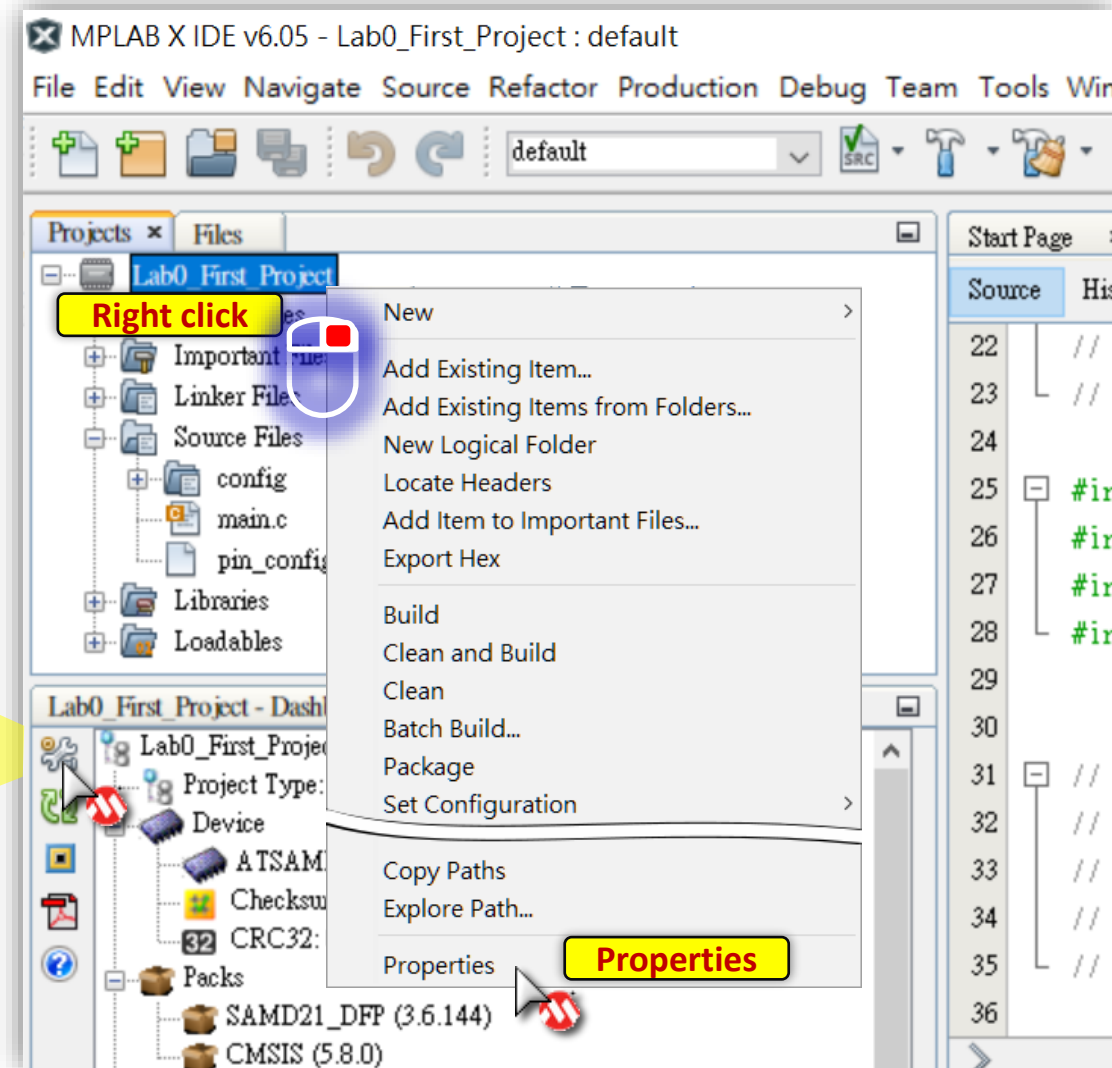
- Source code and Output window.



Lab0 Project Properties

Step 11

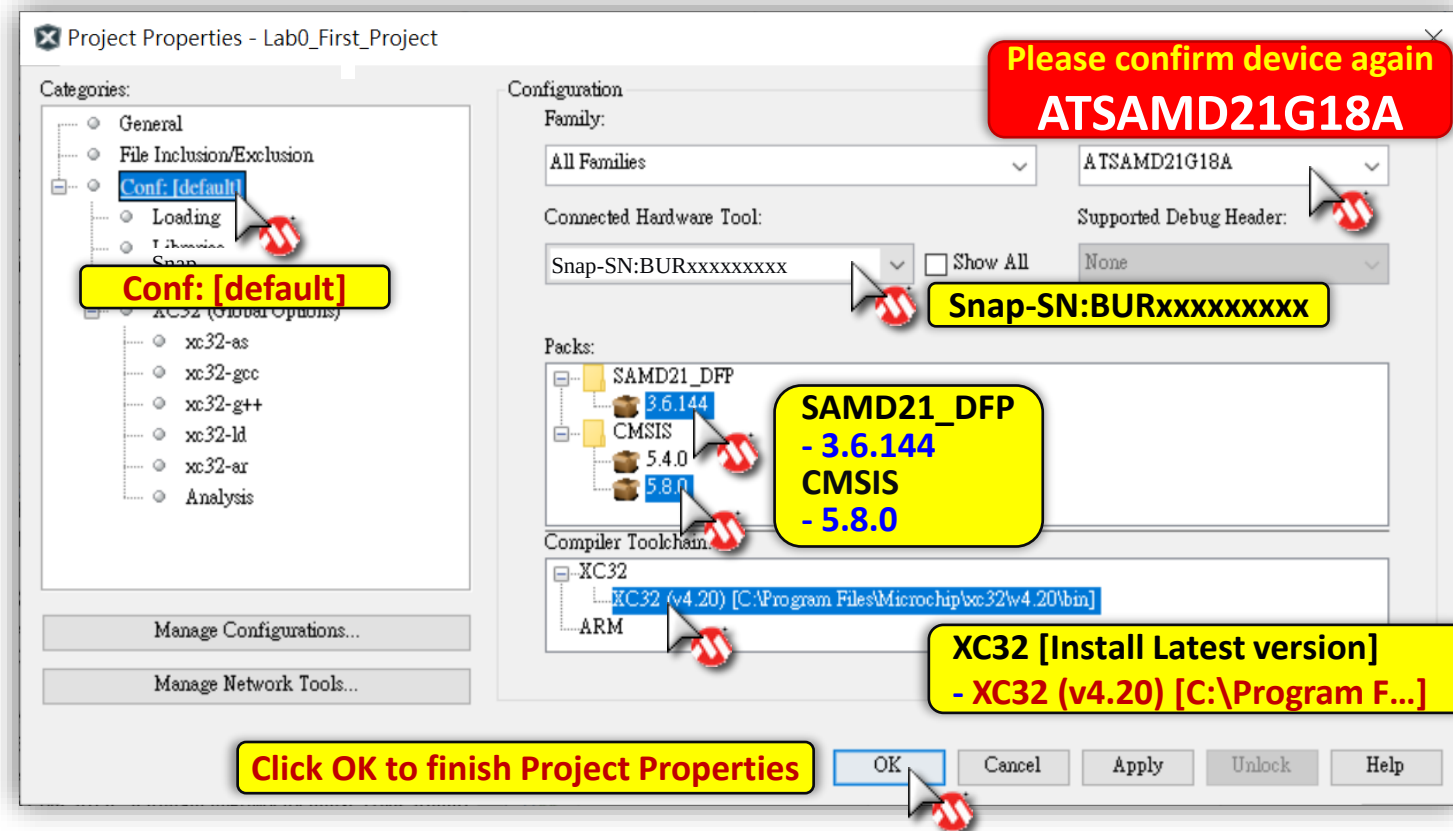
- Now we need do some settings in project properties :
- Click  on Dashboard or Right click on project and select to Properties.



Lab0 Project Properties

Step 12

- Select Connected Hardware Tool : **Snap-SN: BURxxxxxxxxxx**
- SAMD21_DFP : **3.6.144**
- CMSIS : **5.8.0**
- Compiler Toolchain : **XC32 (v4.20) [C:\Program Files\Microchip...]**



Lab0 Add Code

Step 13

- Add below **code** segment to main.c

```
// TODO 0.01
uint32_t i = 0;
uint32_t x = 0;

int main( void )
{
    SYS_Initialize(NULL);

    while( true )
    {
        SYS_Tasks();

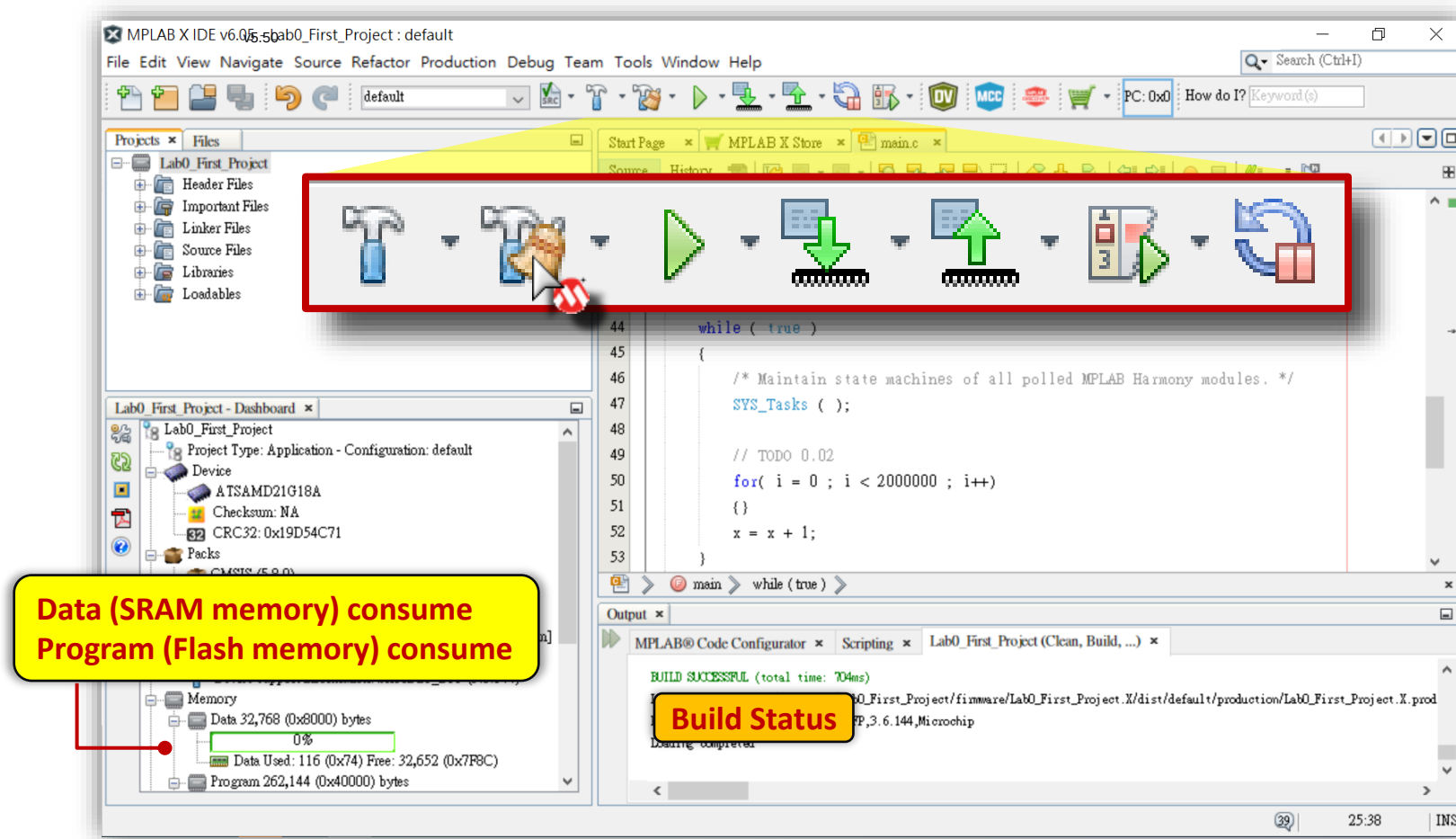
        // TODO 0.02
        for( i = 0 ; i < 2000000 ; i++);
        x = x + 1;
    }

    return ( EXIT_FAILURE);
}
```

Lab0 Build Project

Step 14

- Click Icon  or press **F11** to build your project.
- Compile result message will show to output windows.



The screenshot displays the MPLAB X IDE v6.0.5.5000 interface for the 'Lab0_First_Project'. The toolbar at the top features several icons, with a red box highlighting the build icon (a hammer and a box) and the 'F11' key. The 'Lab0_First_Project - Dashboard' window shows the project configuration, including the device 'ATSAMD21G18A' and the checksum 'NA'. The 'Memory' section indicates that data (SRAM memory) and program (Flash memory) are being consumed. The 'Output' window shows the build status as 'BUILD SUCCESSFUL' with a total time of 704ms. A yellow callout box points to the memory usage section, stating 'Data (SRAM memory) consume' and 'Program (Flash memory) consume'. The 'Build Status' window shows the build details, including the file path and the version of the MPLAB X IDE.

Data (SRAM memory) consume
Program (Flash memory) consume

Build Status

```
while ( true )
{
    /* Maintain state machines of all polled MPLAB Harmony modules. */
    SYS_Tasks ( );

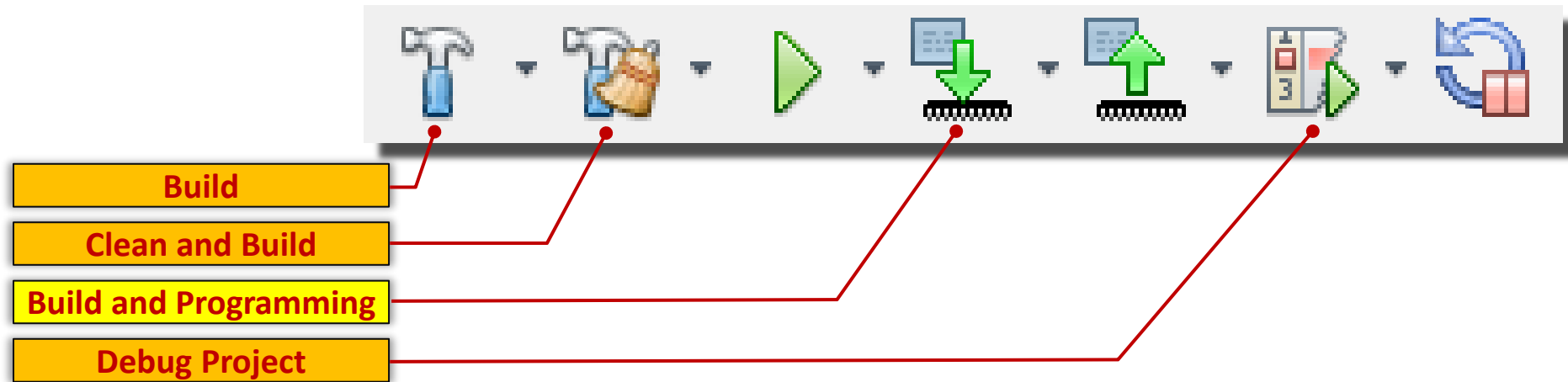
    // TODO 0.02
    for( i = 0 ; i < 2000000 ; i++)
    {
        x = x + 1;
    }
}
```

BUILD SUCCESSFUL (total time: 704ms)
Lab0_First_Project/finware/Lab0_First_Project.X/dist/default/production/Lab0_First_Project.X.prod
FP,3.6.144, Microchip
Loading completed

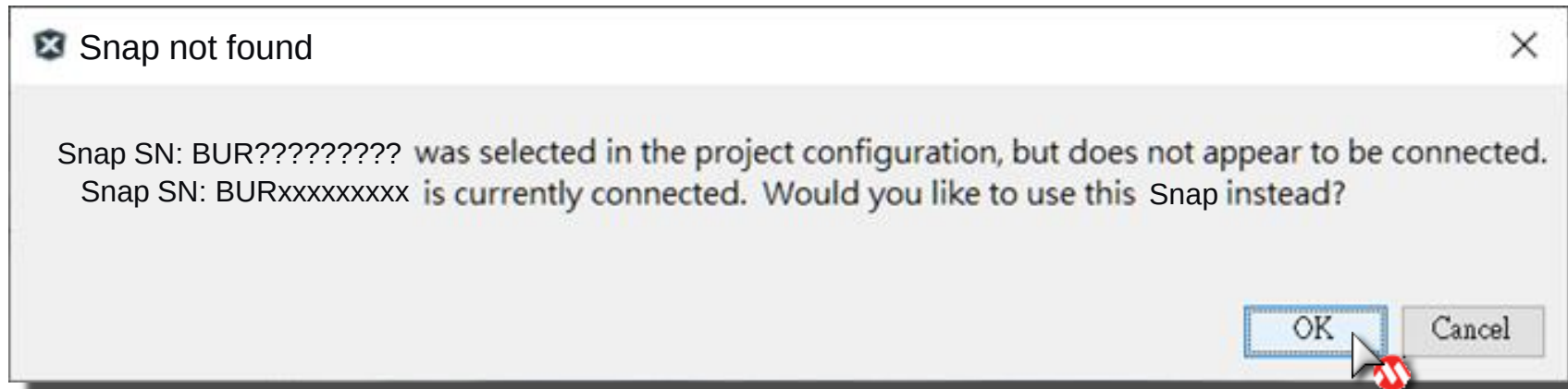
Lab0 Programming Project

Step 15

- Click Icon  to build and program firmware to EVB.



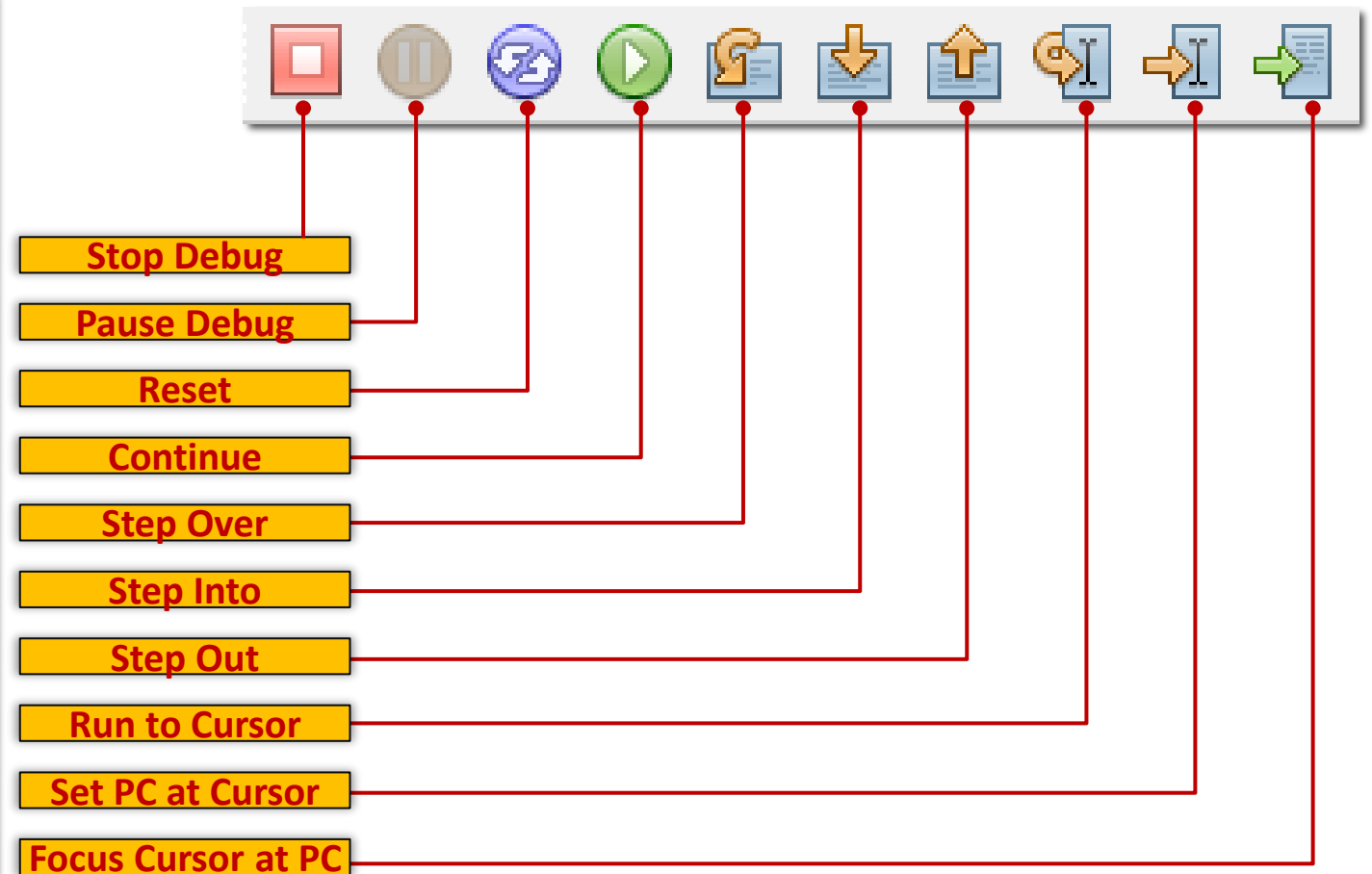
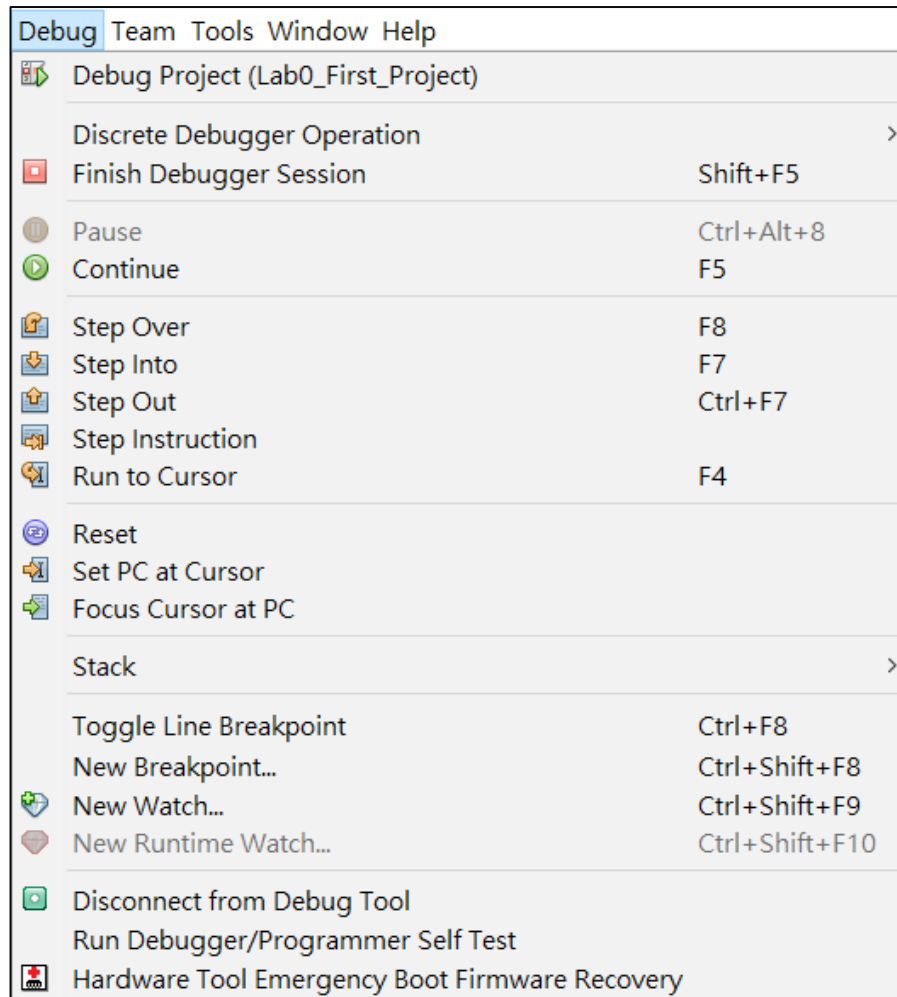
- Different Programmer SN of your project while programming.



Lab0 Debug Project

Step 16

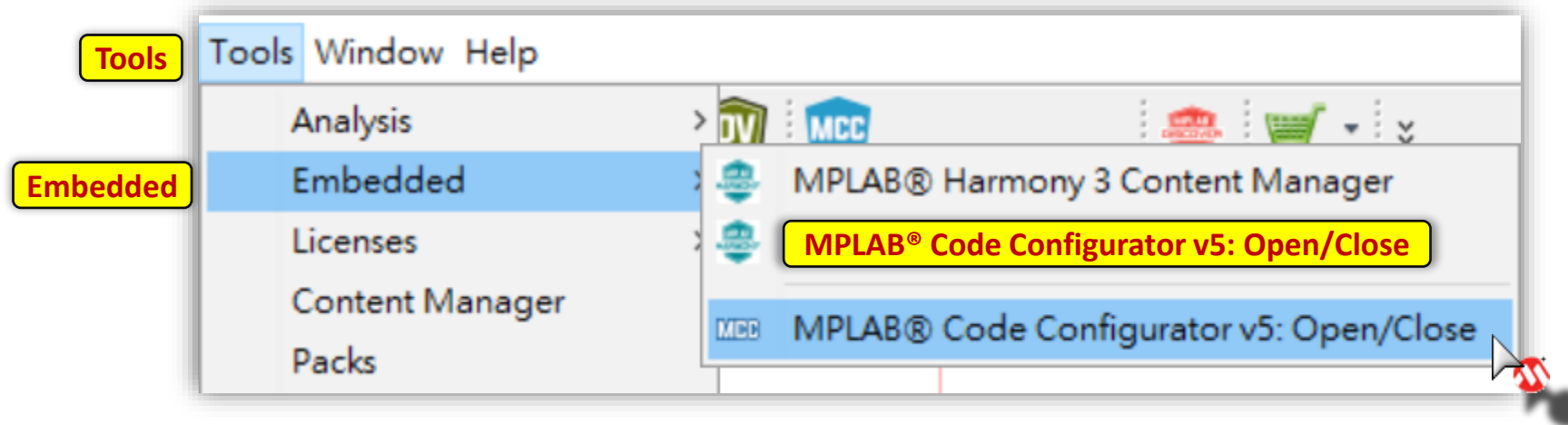
- MPLAB X IDE provide a lot of debug window, select the **Debug**



MPLAB® MCC Harmony Interface introduction

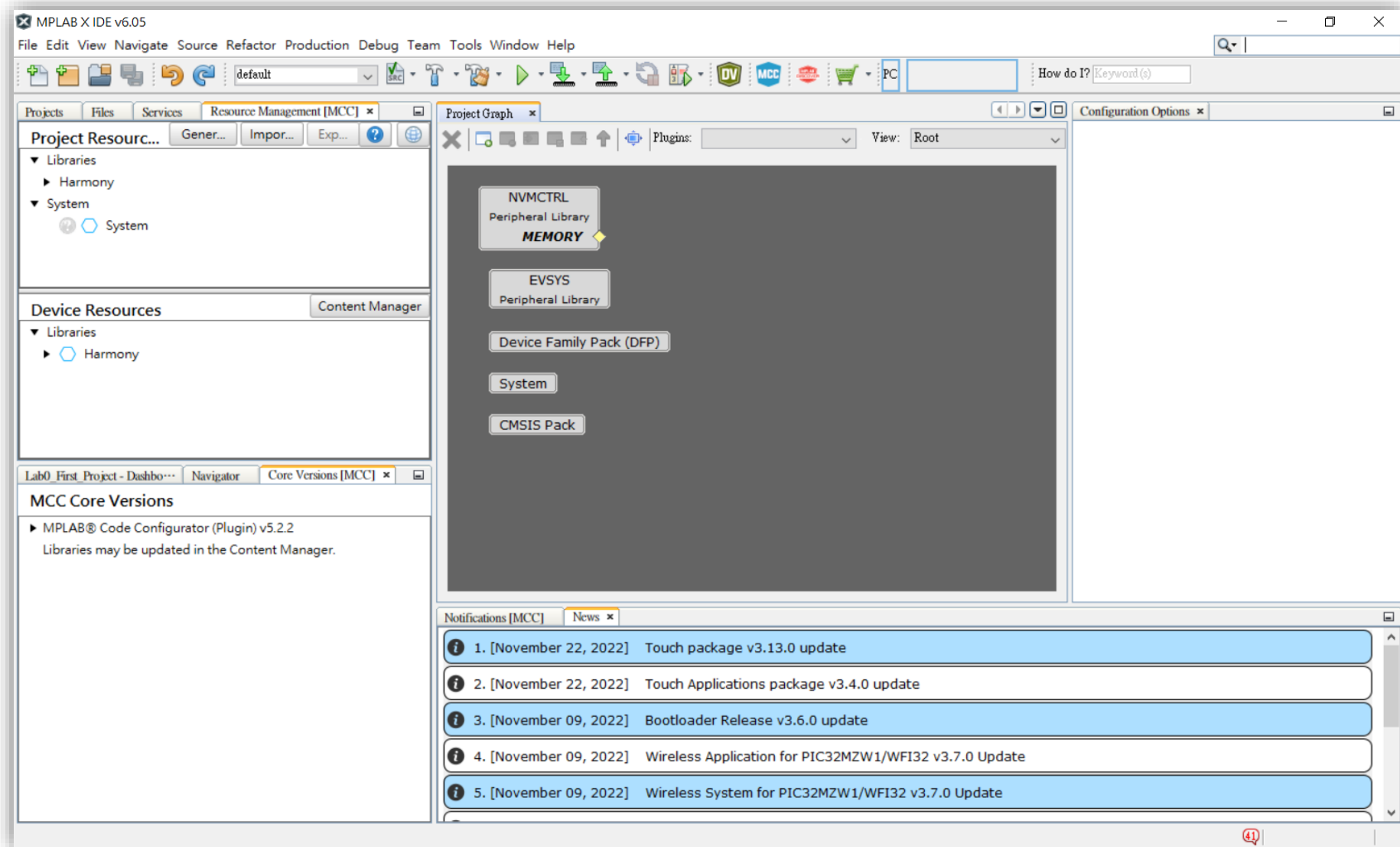
MCC Harmony Interface

- **Notice !** If you already close MCC Harmony, please open it through below steps.
- Select to **Tools > Embedded > MPLAB® Code Configurator v5: Open/Close**



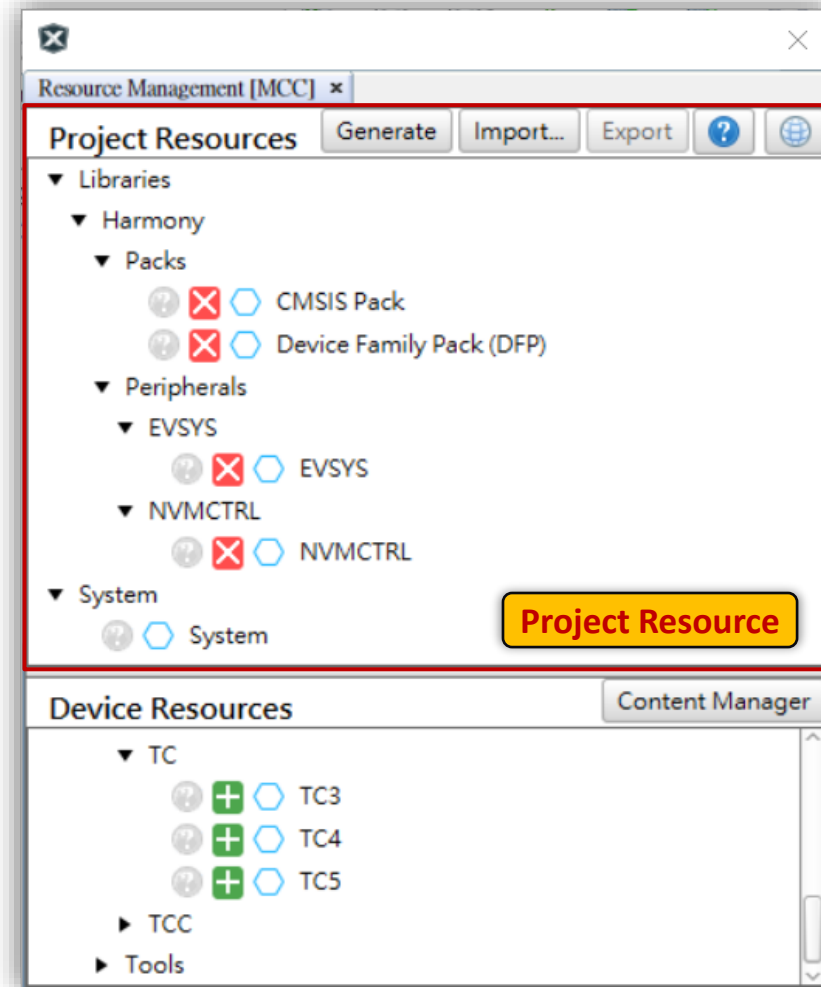
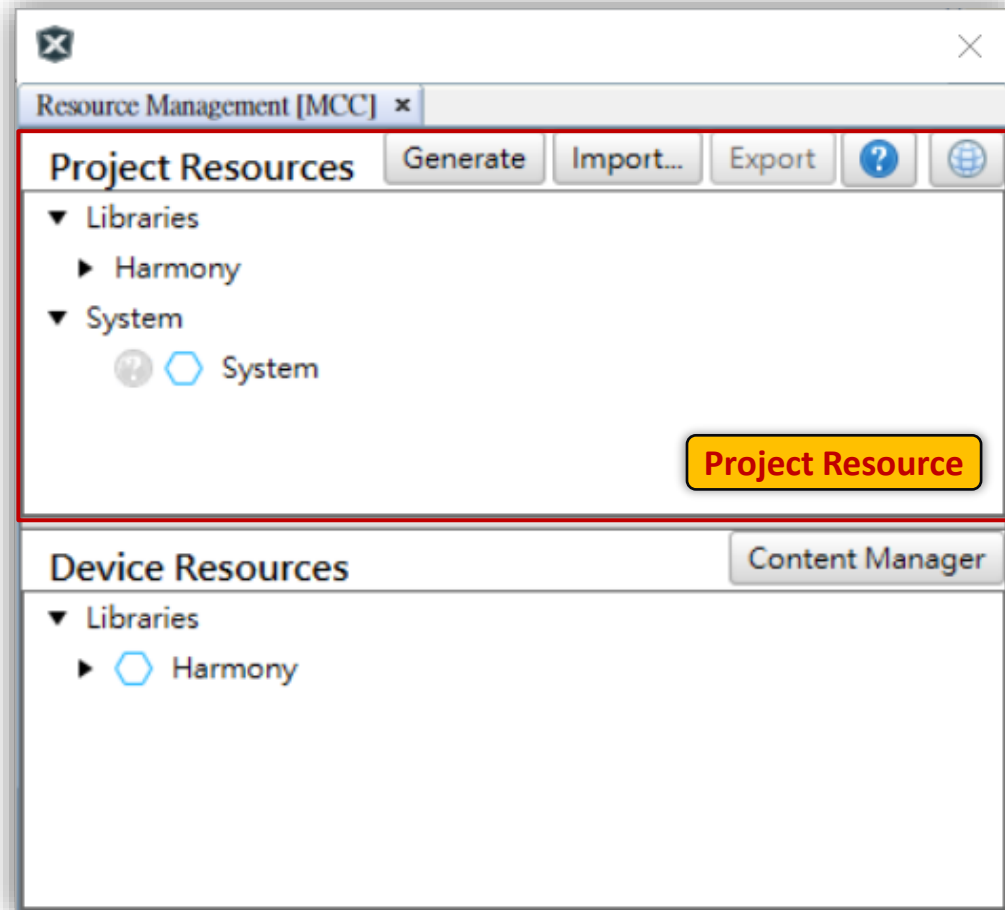
MCC Harmony Interface

- Now let's walk through MCC Harmony.



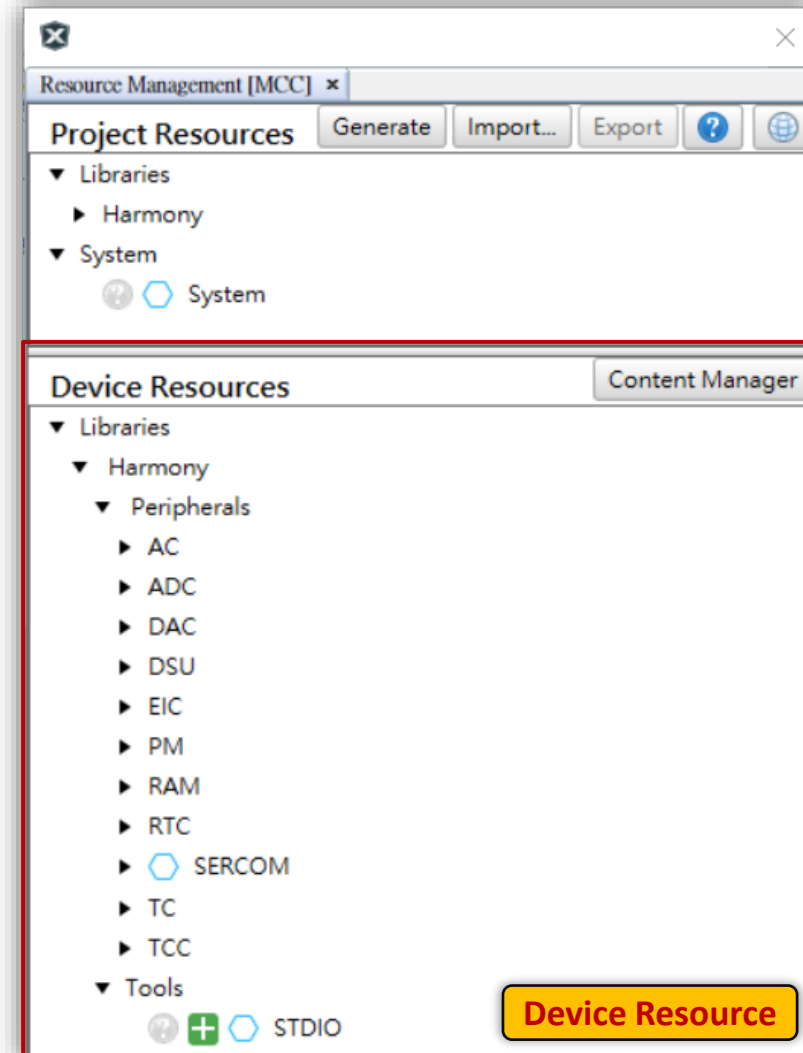
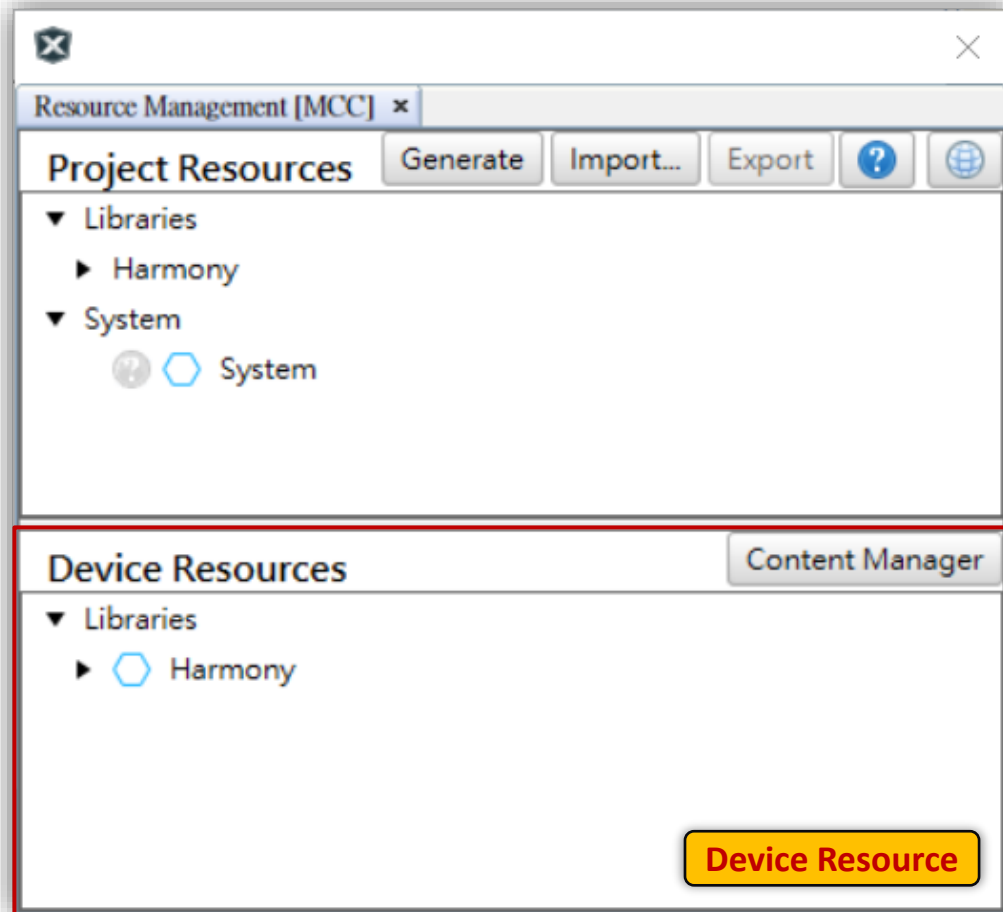
MCC Harmony Interface

- Project Resource. (Current modules in project)



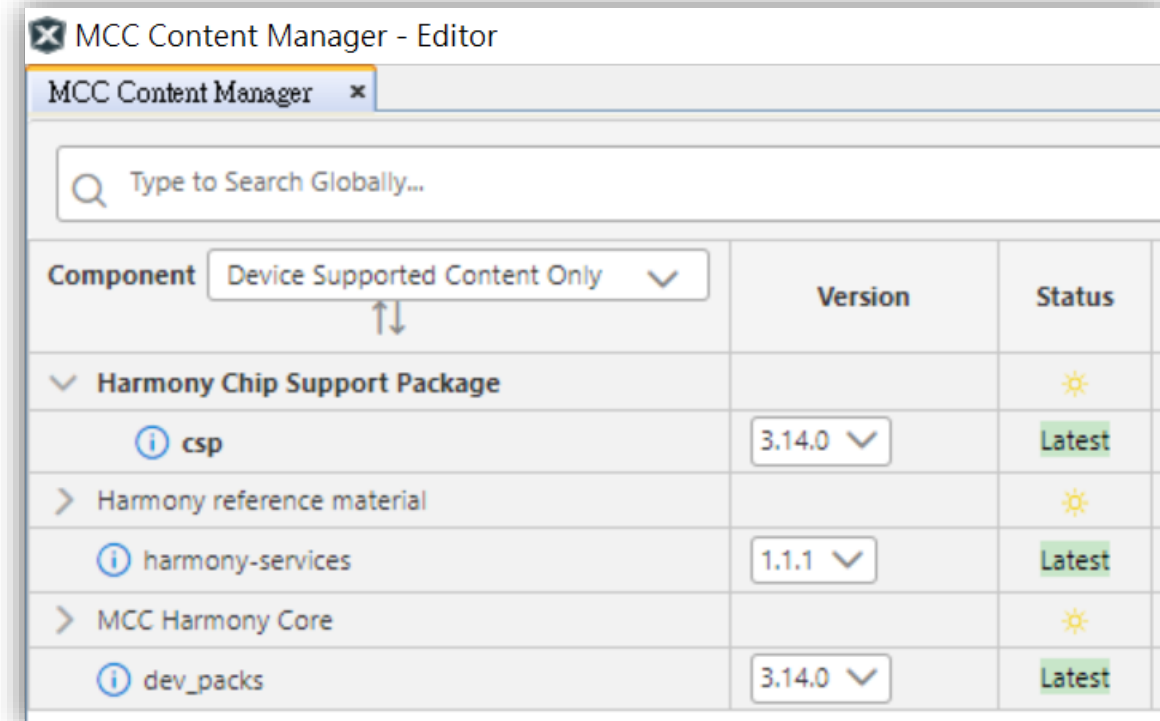
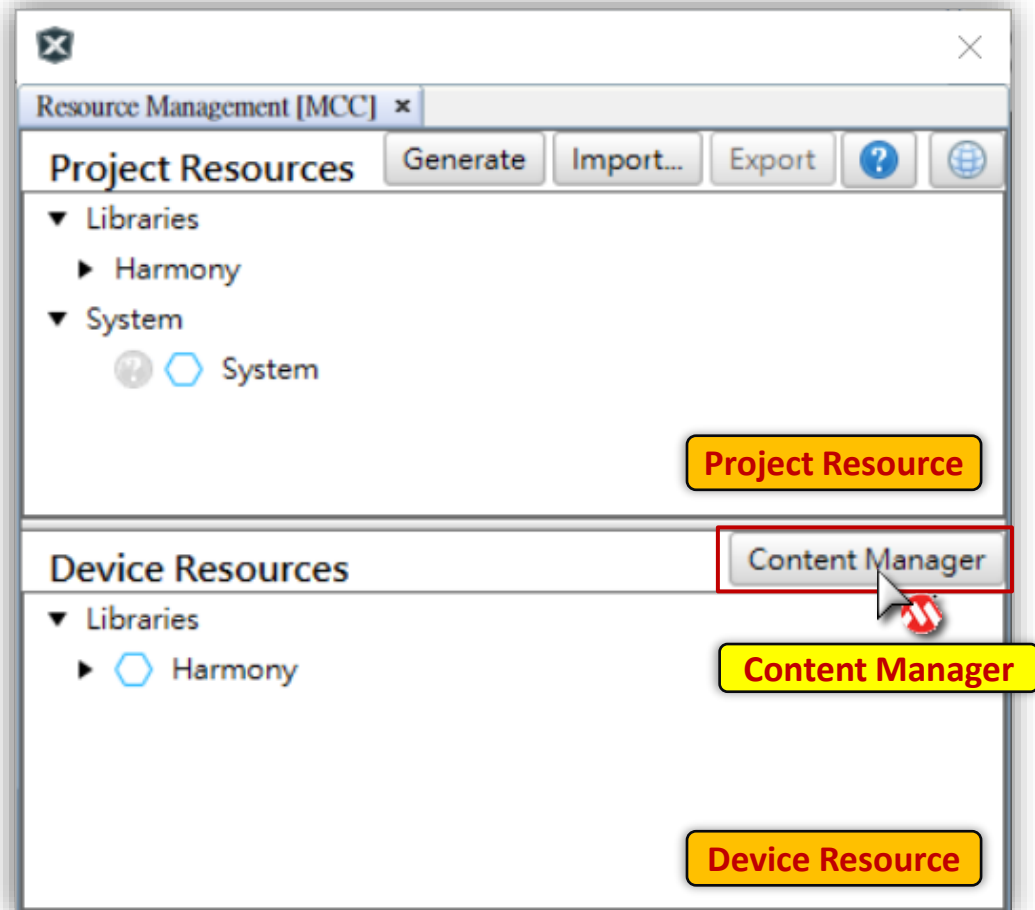
MCC Harmony Interface

- **Device Resource.** (Modules you could add-in project)



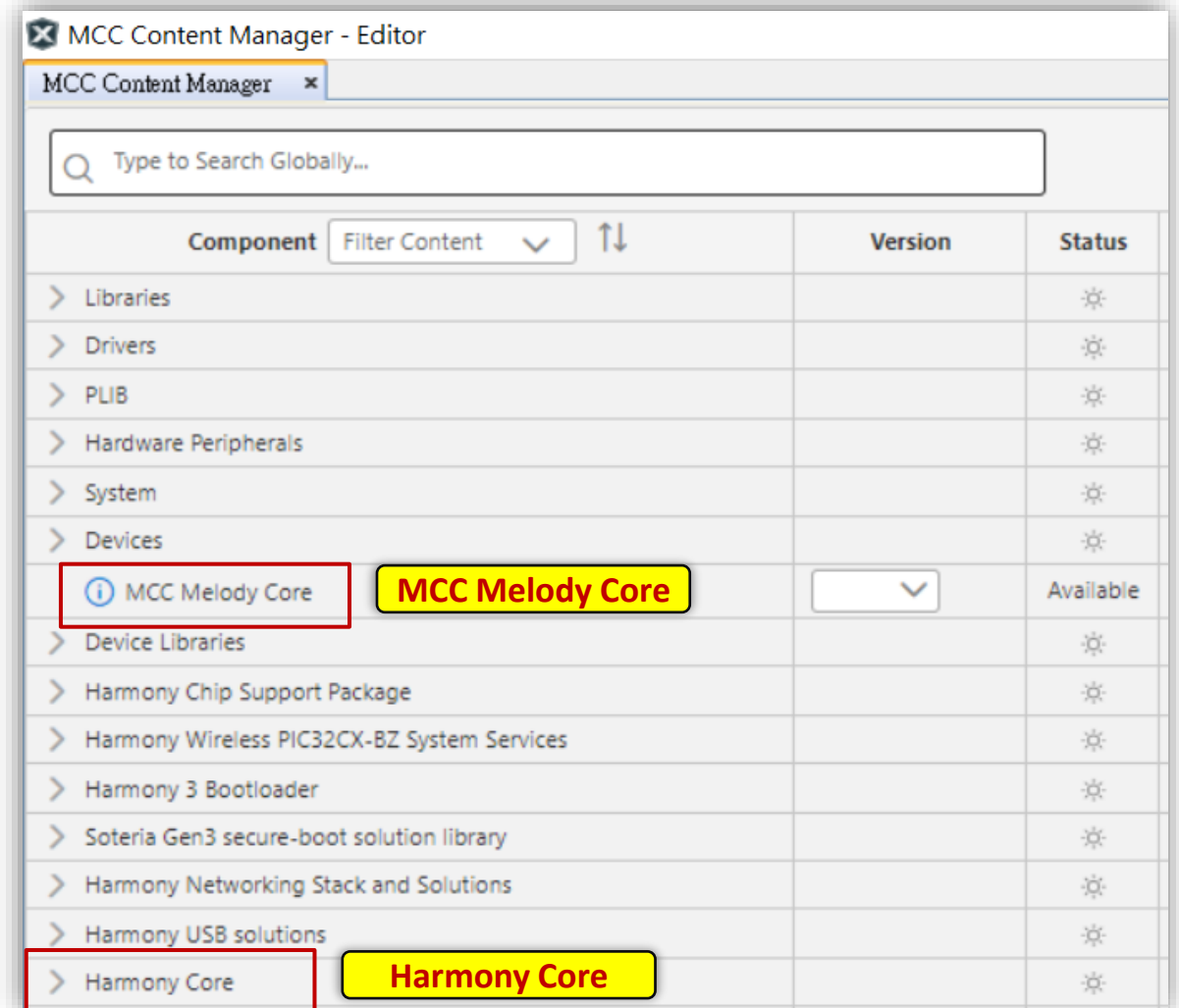
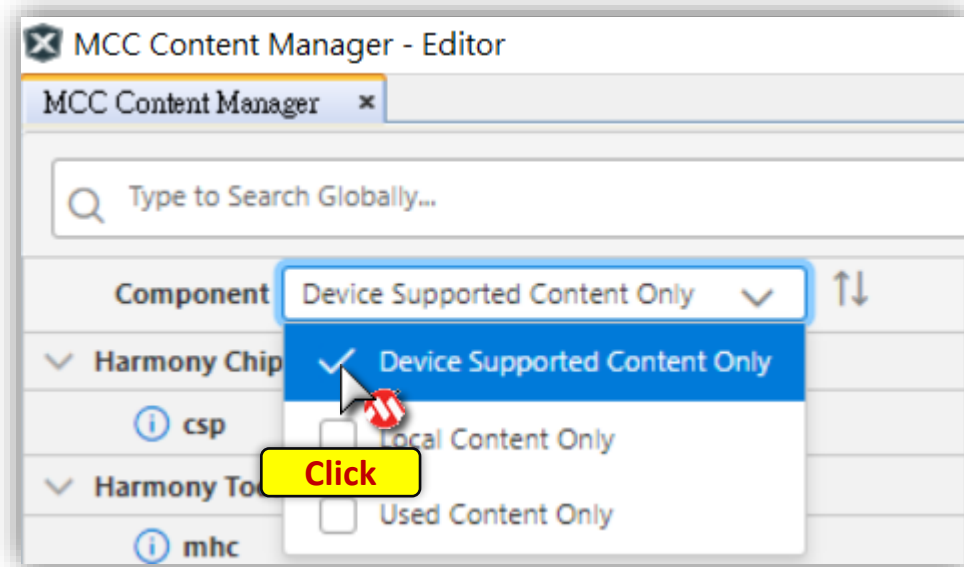
MCC Harmony Interface

- Content Manager.(Existing Framework)



MCC Harmony Interface

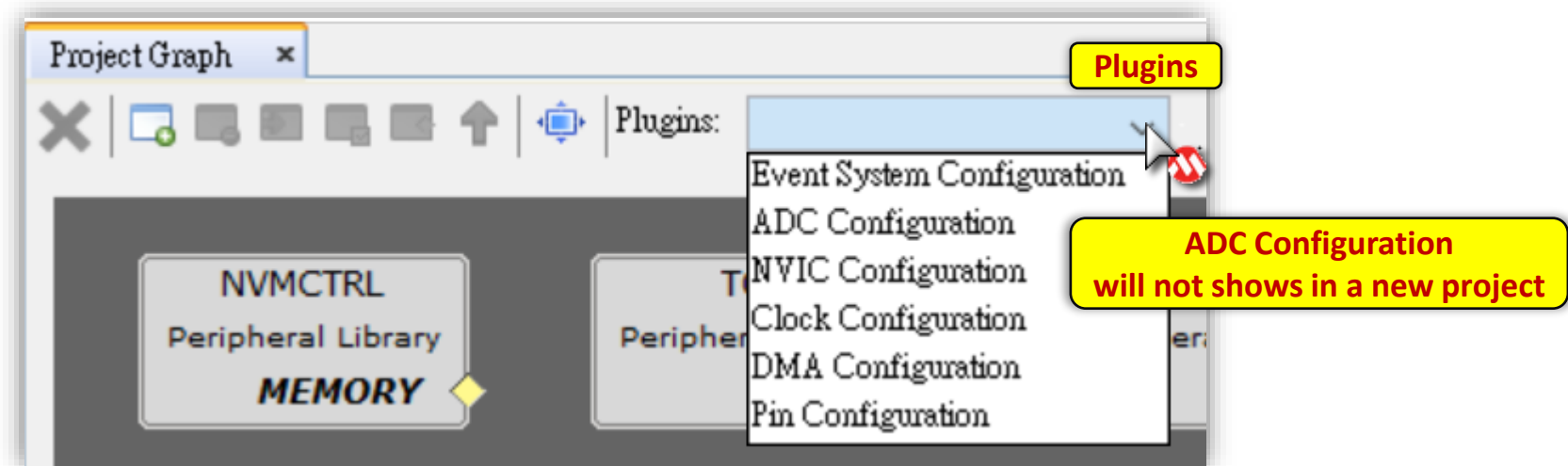
- View All Framework.(Include MCC Harmony and MCC Melody)
 - **Disable Device Supported Content Only.**



MCC Harmony Interface

Easy View Configuration

- Plugins (**Easy View Configuration**)



Easy View Configuration Tools will shows depend on what peripheral you already added-in.

MCC Harmony Interface

Easy View Configuration

- Plugins : **Pin Configuration**

The screenshot displays the MPLAB X IDE v6.05 interface with the MCC Harmony interface for pin configuration. The interface is divided into several panes:

- Project Resource Management [MCC]:** Shows the project structure with Libraries (Harmony, System) and Device Resources (Harmony).
- Project Graph:** Displays the project graph with tabs for Pins, Table View, and Easy View. The Pins tab is active, showing a list of pins and their configurations.
- Pin Settings:** A table showing the configuration for each pin. The table has columns for Pin Number, Pin ID, Custom Name, Function, Mode, and Direction.
- Pin Diagram:** A diagram of the ATSAM21G18A microcontroller showing the pin locations and their configurations.
- Pin Table:** A table showing the pin configurations for the selected package (TQFP48).

The Pin Settings table is as follows:

Pin Number	Pin ID	Custom Name	Function	Mode	Direction
1	PA00		Available	Digital	High Impedance
2	PA01		Available	Digital	High Impedance
3	PA02		Available	Digital	High Impedance
4	PA03		Available	Digital	High Impedance
5	GNDANA			Digital	High Impedance
6	VDDANA			Digital	High Impedance
7	PB08		Available	Digital	High Impedance
8	PB09		Available	Digital	High Impedance
9	PA04		Available	Digital	High Impedance
10	PA05		Available	Digital	High Impedance
11	PA06		Available	Digital	High Impedance
12	PA07		Available	Digital	High Impedance
13	PA08		Available	Digital	High Impedance
14	PA09		Available	Digital	High Impedance
15	PA10		Available	Digital	High Impedance
16	PA11		Available	Digital	High Impedance
17	VDDIO			Digital	High Impedance

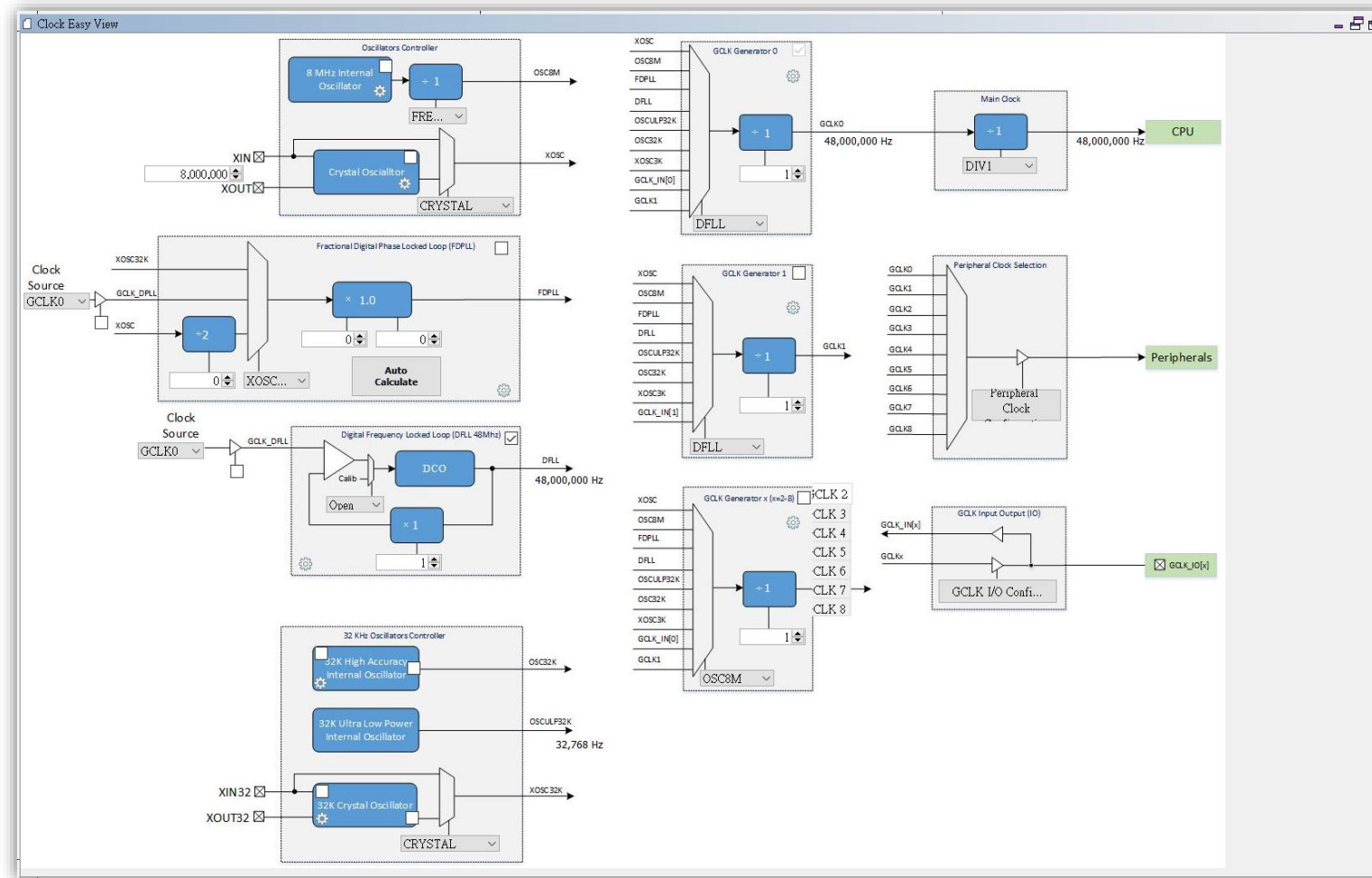
The Pin Table shows the pin configurations for the selected package (TQFP48). The table has columns for Module, Function, and Pin Number (1 to 24).

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
AC	AC_AIN0																								
	AC_AIN1																								
	AC_AIN2																								
	AC_AIN3																								

MCC Harmony Interface

Easy View Configuration

- Plugins : **Clock Configuration**



MCC Harmony Interface

Easy View Configuration

- Plugins : **NVIC Configuration**

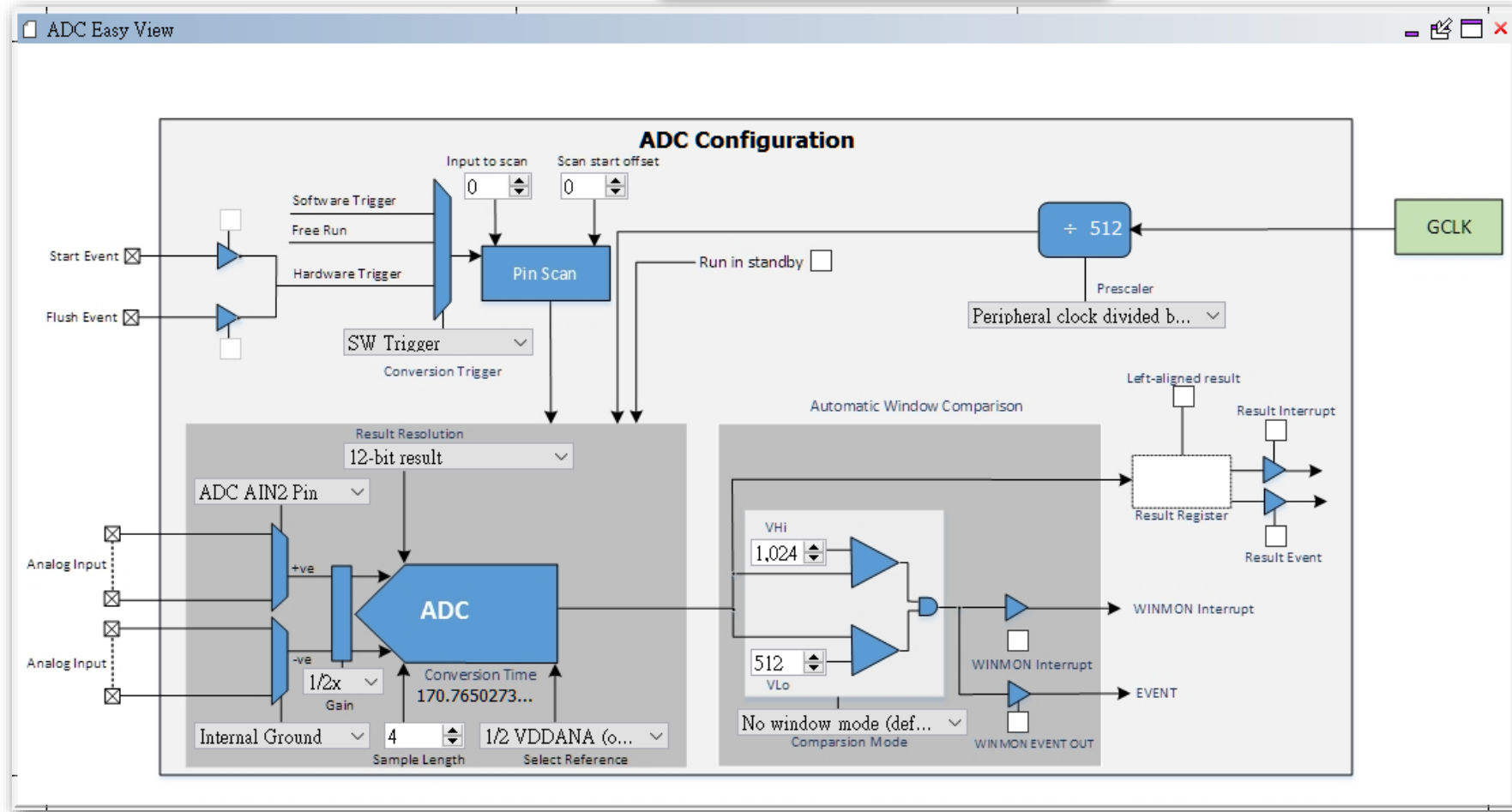
NVIC Settings				
Vector Number	Vector	Enable	Priority (0 =	Handler Name
-15	Reset (Reset Vector)	☒	-3	Reset_Handler
-14	NonMaskableInt (Non-maskable Interrupt)	☒	-2	NonMaskableInt_Handler
-13	HardFault (Hard Fault)	☒	-1	HardFault_Handler
-5	SVCall (SuperVisor Call)	☒	0	SVCall_Handler
-2	PendSV (Pendable Service)	☒	0	PendSV_Handler
-1	SysTick (System Tick Timer)	☐	0	SysTick_Handler
0	PM (Power Manager)	☐	3	PM_Handler
1	SYSCCTRL (System Controller)	☐	3	SYSCCTRL_Handler
2	WDT (Watchdog Timer)	☐	3	WDT_Handler
3	RTC (Real Time Counter)	☐	3	RTC_Handler
4	EIC (External Interrupt Controller)	☐	3	EIC_Handler
5	NVMCTRL (Non-Volatile Memory Controller)	☐	3	NVMCTRL_Handler
6	DMAC (Direct Memory Controller)	☐	3	DMAC_Handler
7	USB (Universal Serial Bus)	☐	3	USB_Handler
8	EVSYS (Event Systems)	☐	3	EVSYS_Handler
9	SERCOM0 (Serial Communication Interface 0)	☐	3	SERCOM0_Handler
10	SERCOM1 (Serial Communication Interface 1)	☐	3	SERCOM1_Handler
11	SERCOM2 (Serial Communication Interface 2)	☐	3	SERCOM2_Handler
12	SERCOM3 (Serial Communication Interface 3)	☐	3	SERCOM3_Handler
13	SERCOM4 (Serial Communication Interface 4)	☐	3	SERCOM4_Handler
14	SERCOM5 (Serial Communication Interface 5)	☐	3	SERCOM5_Handler
15	TCC0 (Timer/Counter for Control Applications 0)	☐	3	TCC0_Handler
16	TCC1 (Timer/Counter for Control Applications 1)	☐	3	TCC1_Handler
17	TCC2 (Timer/Counter for Control Applications 2)	☐	3	TCC2_Handler
18	TC3 (Timer/Counter 3)	☐	3	TC3_Handler
19	TC4 (Timer/Counter 4)	☐	3	TC4_Handler
20	TC5 (Timer/Counter 5)	☐	3	TC5_Handler
23	ADC (Analog-to-Digital Converter)	☐	3	ADC_Handler
24	AC (Analog Comparators)	☐	3	AC_Handler
25	DAC (Digital-to-Analog Converter)	☐	3	DAC_Handler
26	PTC (Peripheral Touch Controller)	☐	3	PTC_Handler
27	I2S (Inter-IC Sound Controller)	☐	3	I2S_Handler

MCC Harmony Interface

Easy View Configuration

- Plugins : **ADC Configuration**

ADC Configuration
will not show in a new project



MCC Harmony Interface

Easy View Configuration

- Plugins : **Event System Configuration**

The screenshot shows the 'EVENT0 Easy View' window with the 'Event System Manager' tab selected. The interface is divided into three main sections: Channel Configuration, Channel Settings, and User Configuration.

Channel Configuration

Channel Number	Event Generator	Event Status	User Ready	Remove Channel
----------------	-----------------	--------------	------------	----------------

Channel Settings

Path Selection:

Event Edge Selection:

Enable Event Detection Interrupt: ☐

Enable Overrun Interrupt: ☐

User Configuration

USER	Channel Number	Remove User
------	----------------	-------------

Buttons: 'Add Channel' and 'Add User' are located below their respective tables.

MCC Harmony Interface

Easy View Configuration

- Plugins : **DMA Configuration**

DMA Settings

Active Channels List

Channel Number	Trigger
----------------	---------

Channel Settings

☐ Use Linked List Mode

Trigger Action: One Block Transfer Per DMA Request

Source Address Mode: Increment Address After Every Transfer

Destination Address Mode: Increment Address After Every Transfer

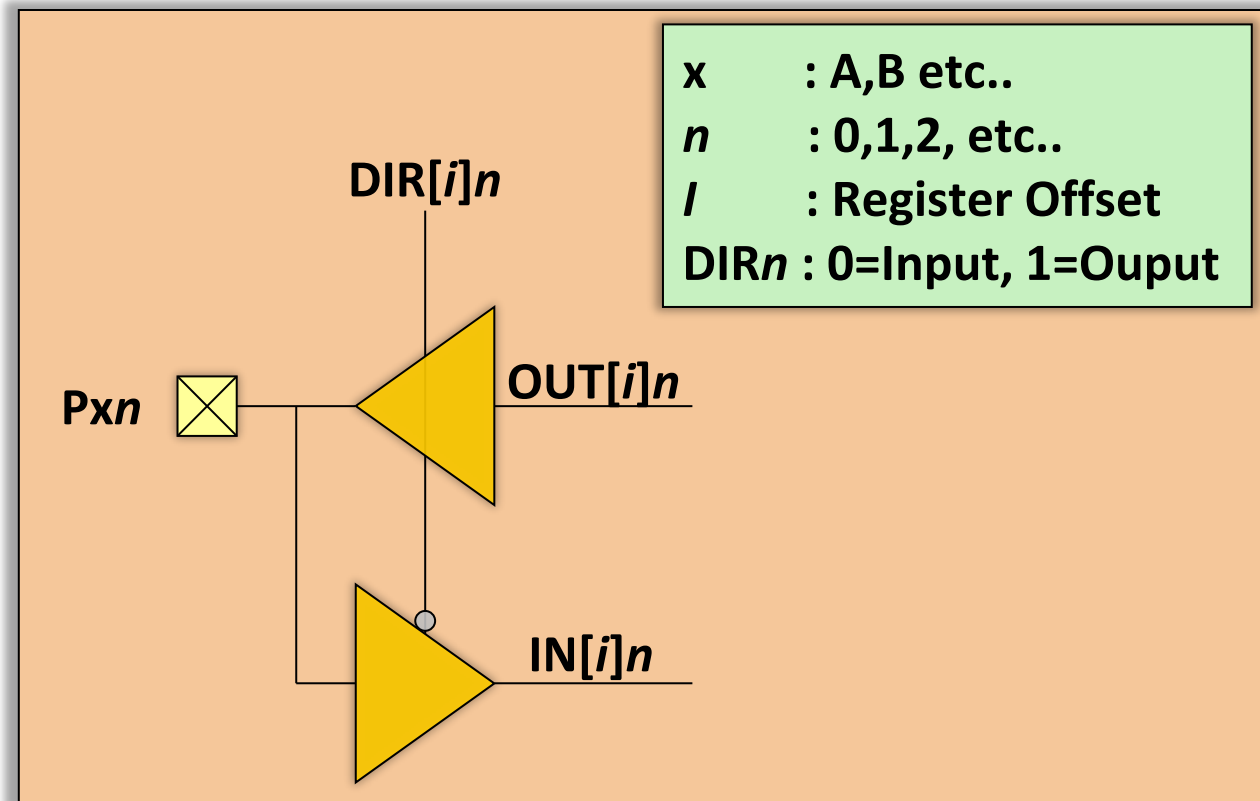
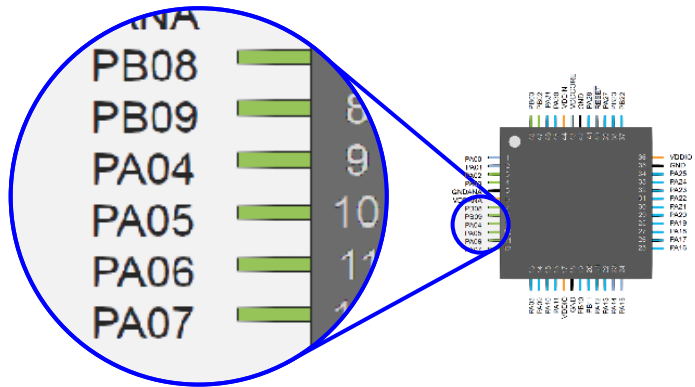
Beat Size: 8-bit bus transfer

Add Channel Remove Selected Channel

PORT I/O Architecture

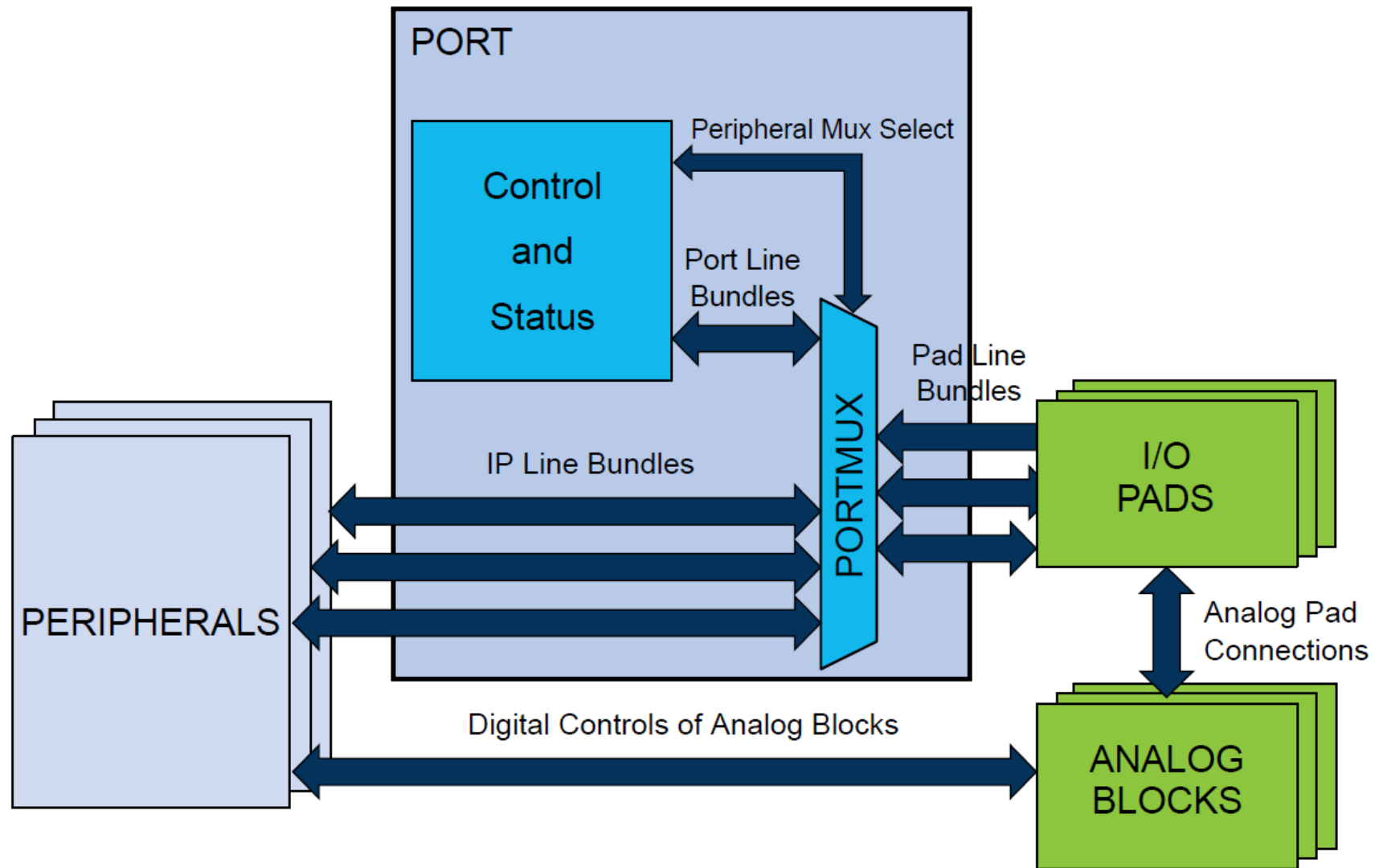
PORT Block Diagram

- 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.
- **DIR:** use to determine in or out. 0 for input, 1 for output.
- **OUT:** set output drive value for Output pins.
- **IN:** reaction logical level from Input pins.

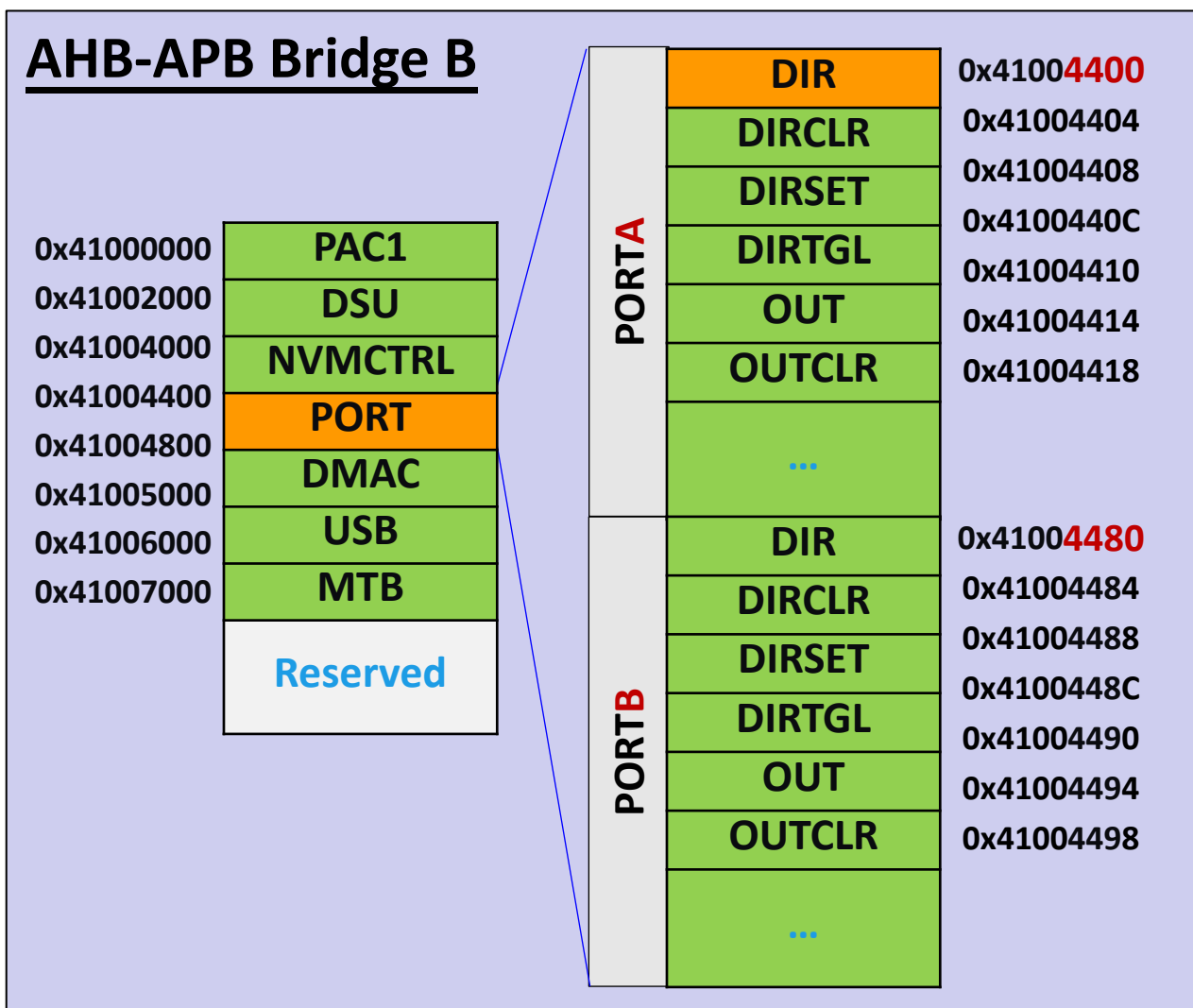


PORT Block Diagram

- SAMD21G18A PORT Block Diagram



Register Access



Register Summary
from datasheet

Offset	Name	Bit Pos.
0x00	DIR	7:0
0x01		15:8
0x02		23:16
0x03		31:24
0x04	DIRCLR	7:0
0x05		15:8
0x06		23:16
0x07		31:24
0x08	DIRSET	7:0
0x09		15:8
0x0A		23:16
0x0B		31:24
0x0C	DIRTGL	7:0
0x0D		15:8
0x0E		23:16
0x0F		31:24
0x10	OUT	7:0
0x11		15:8
0x12		23:16
0x13		31:24
0x14	OUTCLR	7:0
0x15		15:8
0x16		23:16
0x17		31:24

Register Access

- There are two way to access Register:

- Access by Register Pointer

```
*(__IO uint32_t *) (0x41004400U) = 0x55; /* 0x41004400U : PORTA DIR */
```

```
*(__IO uint32_t *) (0x41004480U) = 0x55; /* 0x41004480U : PORTB DIR */
```

- Access by Pre-defined Struct

```
PORT_REGS->GROUP[0].PORT_DIR = 0x55; /* Group[0] : PORTA Register Set */
```

```
PORT_REGS->GROUP[1].PORT_DIR = 0x55; /* Group[1] : PORTB Register Set */
```

atsamd21g18a.h

```
#define PORT_REGS ((port_registers_t*)0x41004400)
```

port.h

```
typedef struct  
{ /* Port Module */  
    port_group_registers_t GROUP[GROUP_NUMBER];  
} port_registers_t;
```

port.h

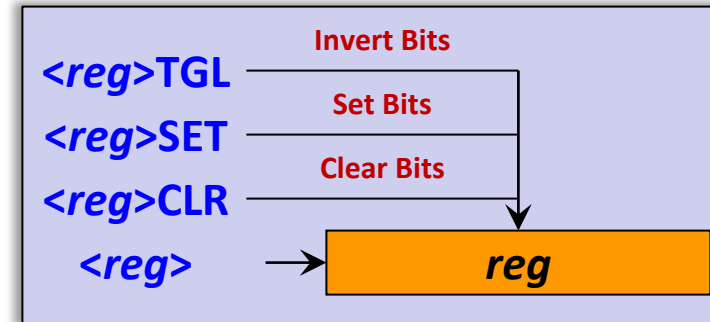
```
typedef struct  
{  
    __IO uint32_t PORT_DIR;  
    __IO uint32_t PORT_DIRCLR;  
    __IO uint32_t PORT_DIRSET;  
    __IO uint32_t PORT_DIRTGL;  
    ...  
} port_group_registers_t;
```


Atomic Bits Manipulation

- Some register provide **<reg>CLR**, **<reg>SET** , **<reg>TGL** register for atomic bits manipulation.

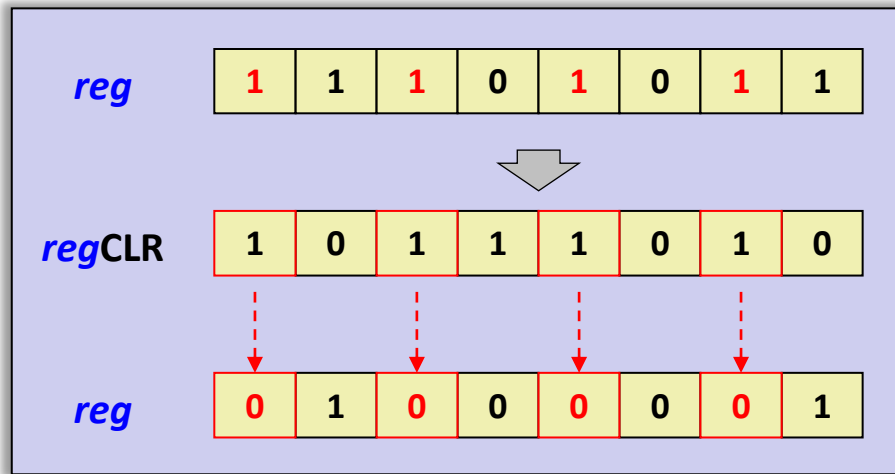
- This register manipulated without doing a read-modify-write operation by using below

- **<reg> Toggle (<reg>TGL)**
- **<reg> Clear (<reg>CLR)**
- **<reg> Set (<reg>SET)**

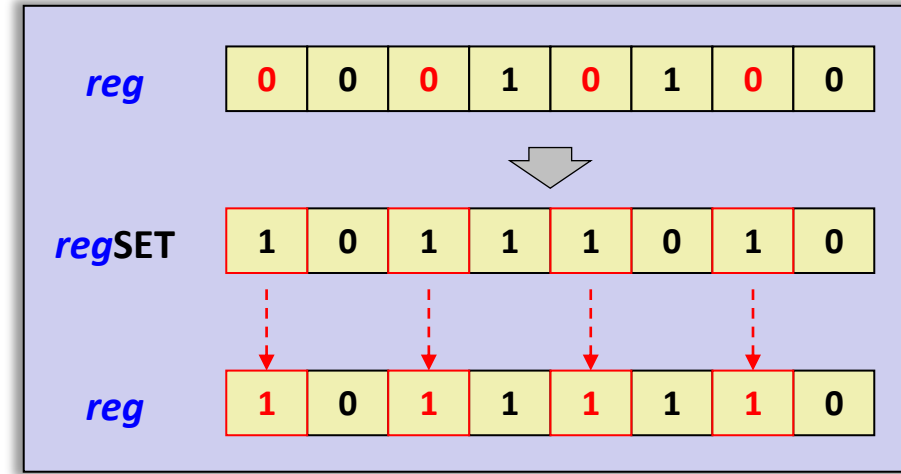


- Setting the manipulation register bits corresponding to the base register will set the bits to 1, clear to 0 or toggle.

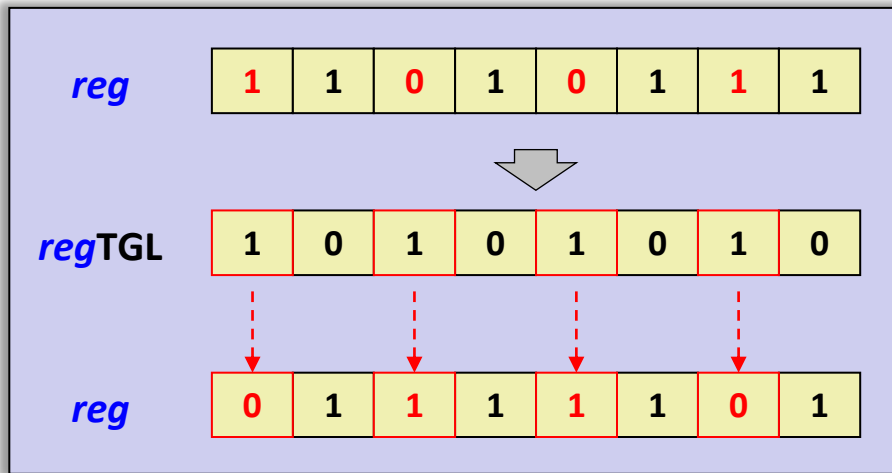
Atomic Bits Manipulation



regCLR : corresponding bits clear to 0



regSET : corresponding bits set to 1



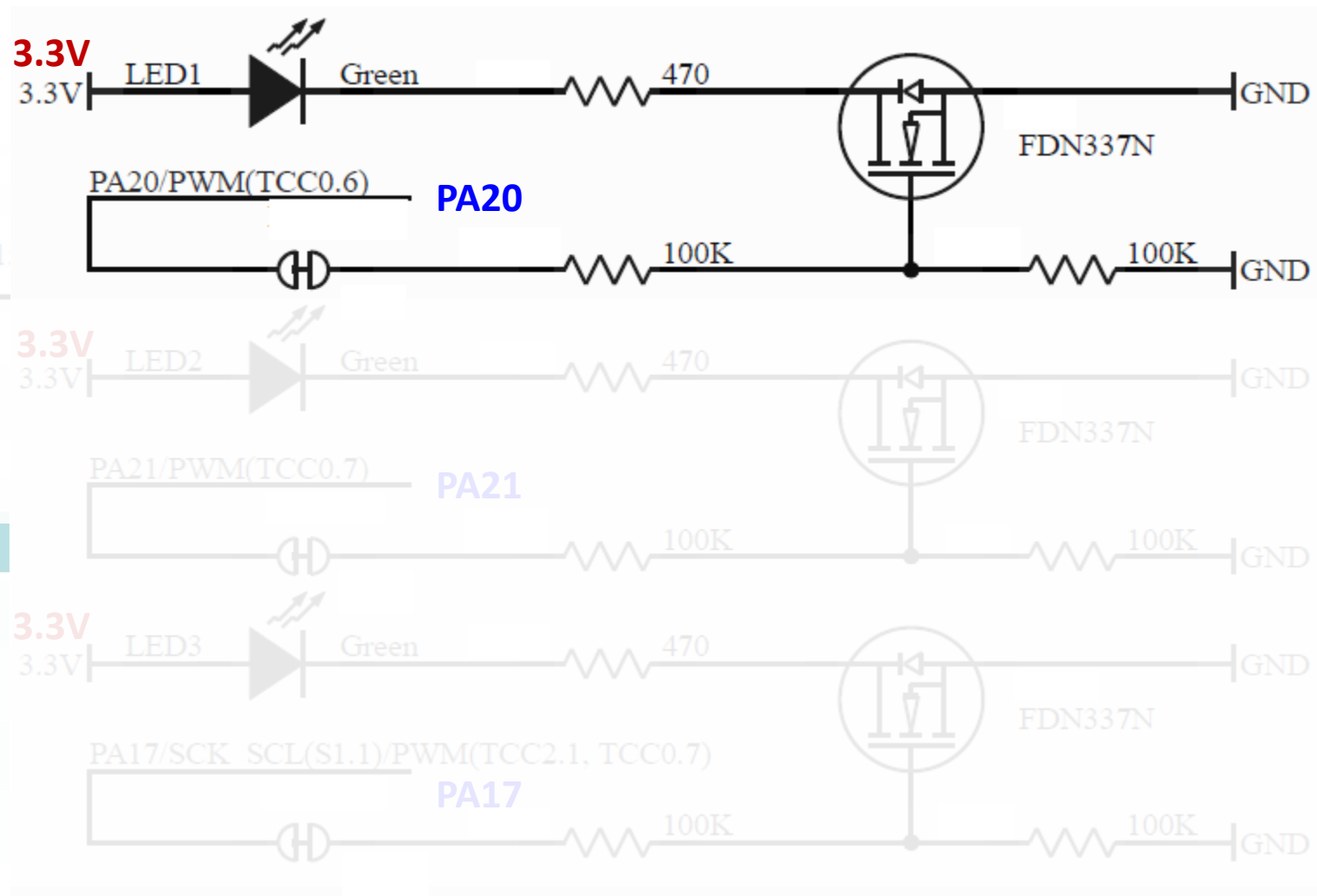
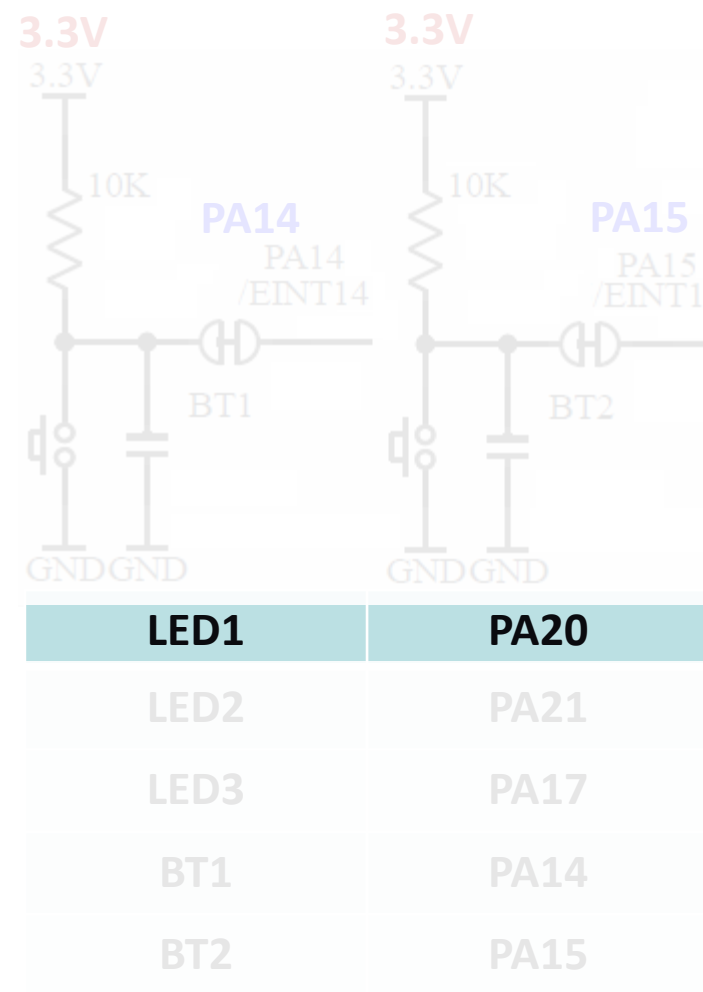
regTGL : corresponding bits toggled
(0 -> 1, 1 -> 0)

Lab1 PORT Output

- Try to control LED1(PA20) continues toggle.
(period around 500ms ~ 1S).
- You can use **register pointer** or **pre-defined struct** to access PORT.

• **How to start ?**

Check Schematic



Lab1 PORT Output

Step 1

a Add code segment to your main loop

```
// TODO 1.01
uint32_t i = 0;

int main (void)
{
    SYS_Initialize ( NULL );

    // TODO 1.02
    *(__IO uint32_t *) (0x41004400U + 0x08U) = 0x00100000;

    while( true )
    {
        for ( i = 0 ; i < 2000000 ; i++ )
        {}
        // TODO 1.03
        *(__IO uint32_t *) (0x41004400U + 0x1CU) = 0x00100000;
    }
}
```

Access by Register Pointer

Period Control

b Program firmware to target board then observe result.

Lab1 PORT Output

Register Description

0x00100000 (Hex)
0000 0000 0001 0000 0000 0000 0000 0000 b (Binary)
3322 2222 2222 1111 1111 11
1098 7654 3210 9876 5432 1098 7654 3210

```
// TODO 1.01
uint32_t i = 0;

int main (void)
{
    SYS_Initialize ( NULL );

    // TODO 1.02
    *(__IO uint32_t *) (0x41004400U + 0x08U) = 0x00100000;

    while( true )
    {
        for ( i = 0 ; i < 2000000 ; i++ );

        // TODO 1.03
        *(__IO uint32_t *) (0x41004400U + 0x1CU) = 0x00100000;
    }
}
```

LED1 Pin PA20

Offset	Name
0x00	DIR
0x01	
0x02	
0x03	
0x04	DIRCLR
0x05	
0x06	
0x07	
0x08	DIRSET
0x09	
0x0A	
0x0B	
0x1C	OUTGL
0x1D	
0x1E	
0x1F	

Lab1 PORT Output

Step 2

a Modify code segment to your main loop

```
// TODO 1.01
uint32_t i = 0;

int main (void)
{
    SYS_Initialize ( NULL );

    // TODO 1.02
    PORT_REGS->GROUP[0].PORT_DIRSET = 1<<20;

    while( true )
    {
        for ( i = 0 ; i < 2000000 ; i++ );

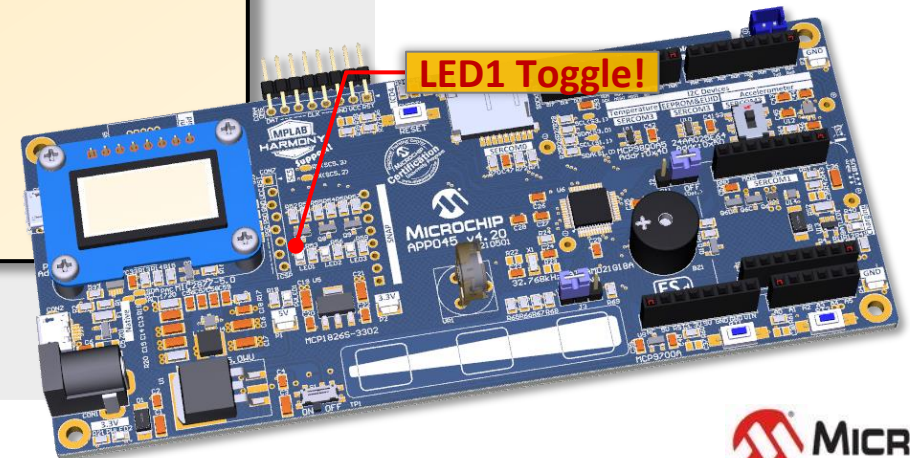
        // TODO 1.03
        PORT_REGS->GROUP[0].PORT_OUTTGL = 1<<20;
    }
}
```

Access by Pre-defined Struct

Period Control

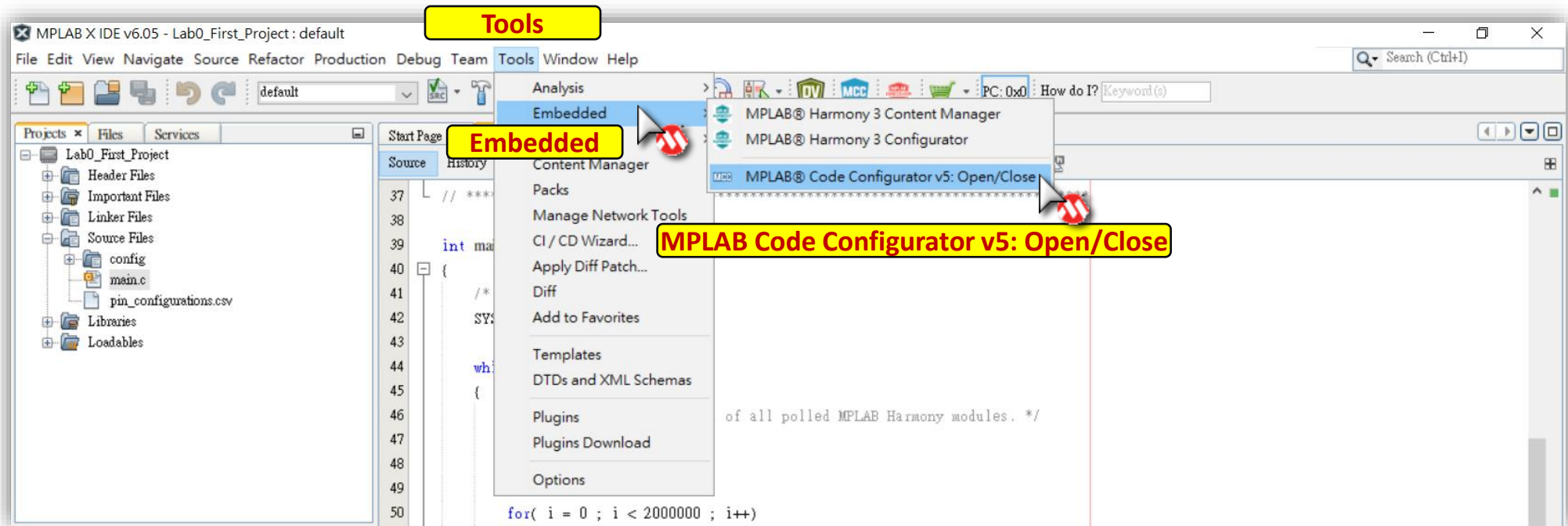
LED1 Toggle!

b Program firmware to target board then observe result.



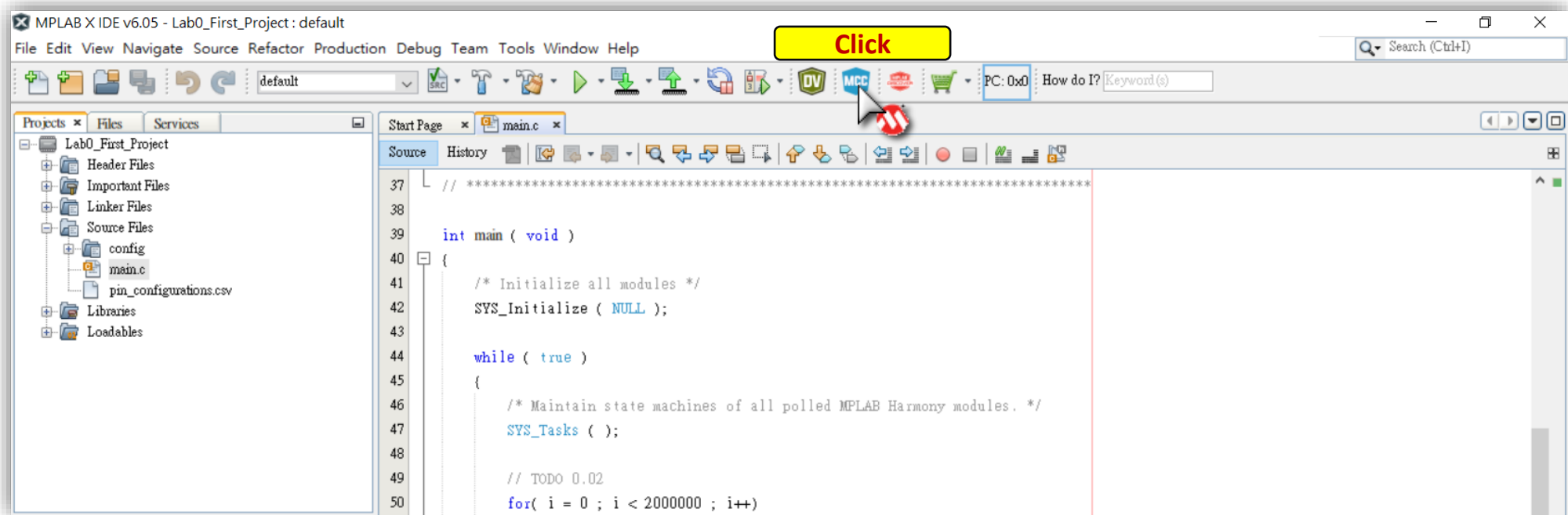
Pin Configuration using MCC Harmony

- If you already close MCC Harmony, please open it through below steps.
- Menu **Tools > Embedded > MPLAB Code Configurator v5: Open/Close** or Click 



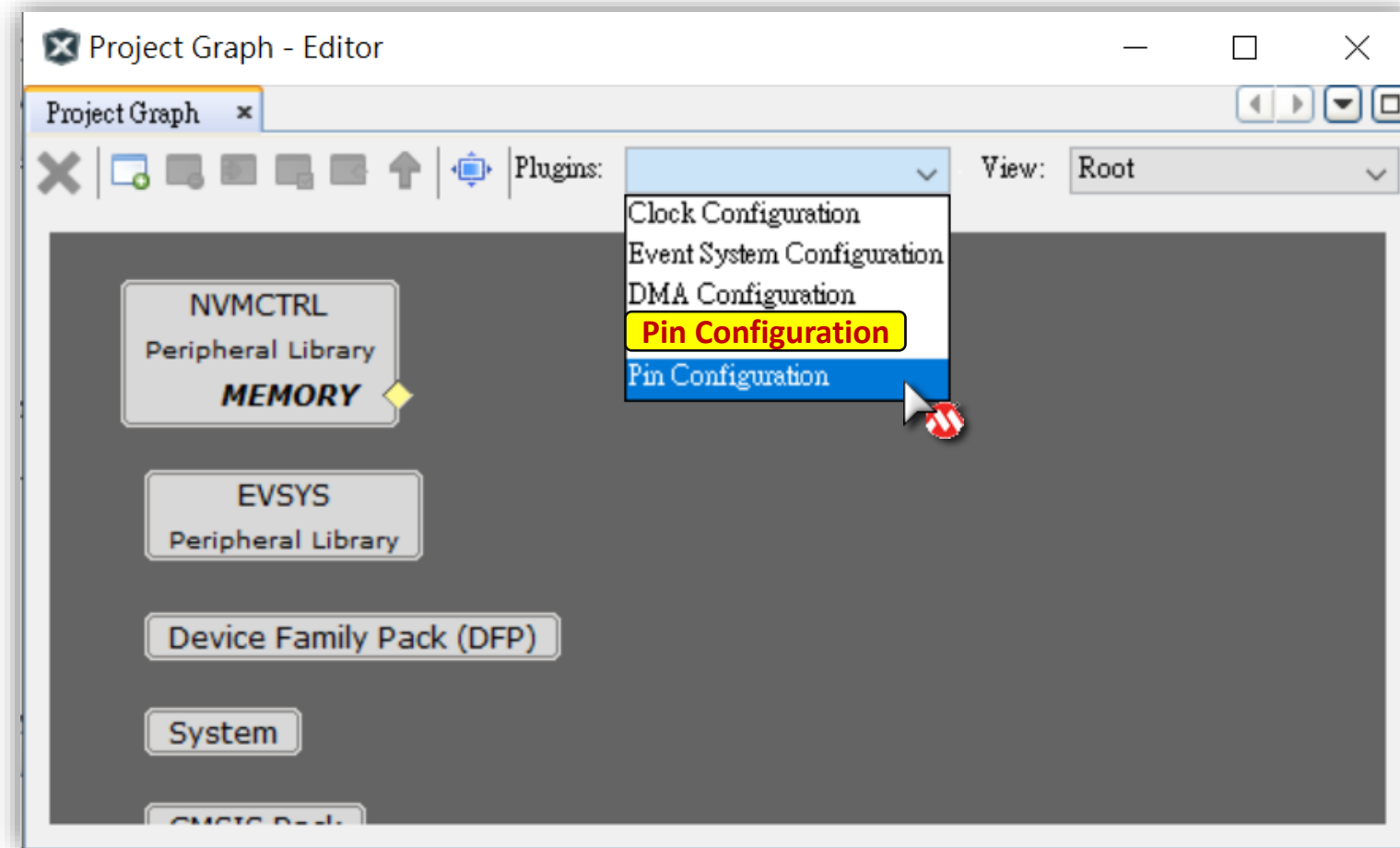
Pin Configuration using MCC Harmony

- Click  Icon.



Pin Configuration using MCC Harmony

- Return to Harmony Configurator.
- Select **Project Graph > Plugins > Pin Configuration**



Pin Configuration using MCC Harmony

- Docking three **Pin Configuration** windows as below.

MPLAB X IDE v6.05 - Lab0_First_Project : default

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

default

PC: 0x0 How do I? Keyword(s)

Projects Files Services Resource Ma... x

Project Re... Gen... Im... E... ?

Libraries

- Harmony

System

Device Resources Content Manager

Libraries

- Harmony

Lab0_First_Pro... Navigator Core Versions... x

MCC Core Versions

- MPLAB® Code Configurator (Plugin) v5.2.2

Libraries may be updated in the Content Manager.

Project Graph x Pin Settings x

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Dire
1	PA00		Available	Digital	High Impe
2	PA01		Available	Digital	High Impe
3	PA02		Available	Digital	High Impe
4	PA03		Available	Digital	High Impe
5	GNDANA			Digital	High Impe
6	VDDANA			Digital	High Impe
7	PB08		Available	Digital	High Impe
8	PB09			Digital	High Impe
9	PA04		Available	Digital	High Impe
10	PA05		Available	Digital	High Impe
11	PA06		Available	Digital	High Impe
12	PA07		Available	Digital	High Impe

Pin Setting

Pin Diagram

ATSAM21G18A

Pin Table

Package: TQFP48

Module	Function	PA00	PA01	PA02	PA03	GNDANA	VDDANA	PB08	PB09	PA04	PA05	PA06	PA07	PA08	PA09	PA10	PA11	VDDIO	GNDIO	PB10
DAC	DAC_VOUT																			
DAC	DAC_VREFP																			

Pin Table

Lab1 PORT Output

Step 3

- Pin Table configure. (Enable or Disable Pin)

Pin Table

Package: TQFP48

		PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	PA14	PA15	PA16	PA17	PA18	PA19	PA20	PA21	PA22	PA23	PA24
Module	Function	6	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
	GCLK_IO7																		
GPIO	GPIO																		
GPIO	I2S_FS0																		
	I2S_MCK0																		

PA20

Click

Available

Pin Table

Package: TQFP48

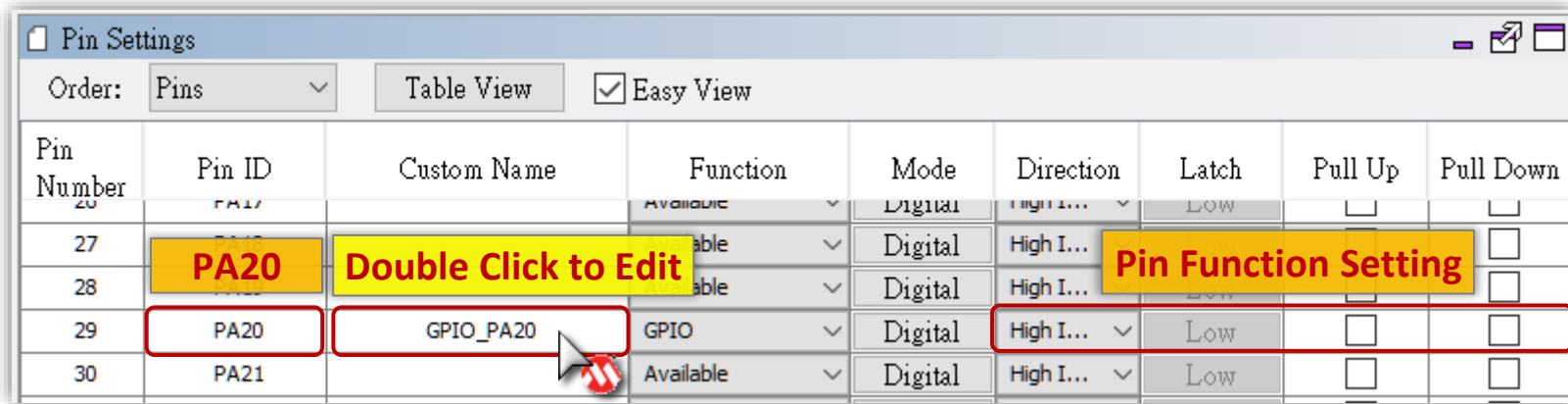
		PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	PA14	PA15	PA16	PA17	PA18	PA19	GPIO_P	PA21	PA22	PA23	PA24
Module	Function	6	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
	GCLK_IO7																		
GPIO	GPIO																		
	I2S_FS0																		

Green is Enable

Lab1 PORT Output

Step 4

- Pin Setting. (Pin Function Setting)

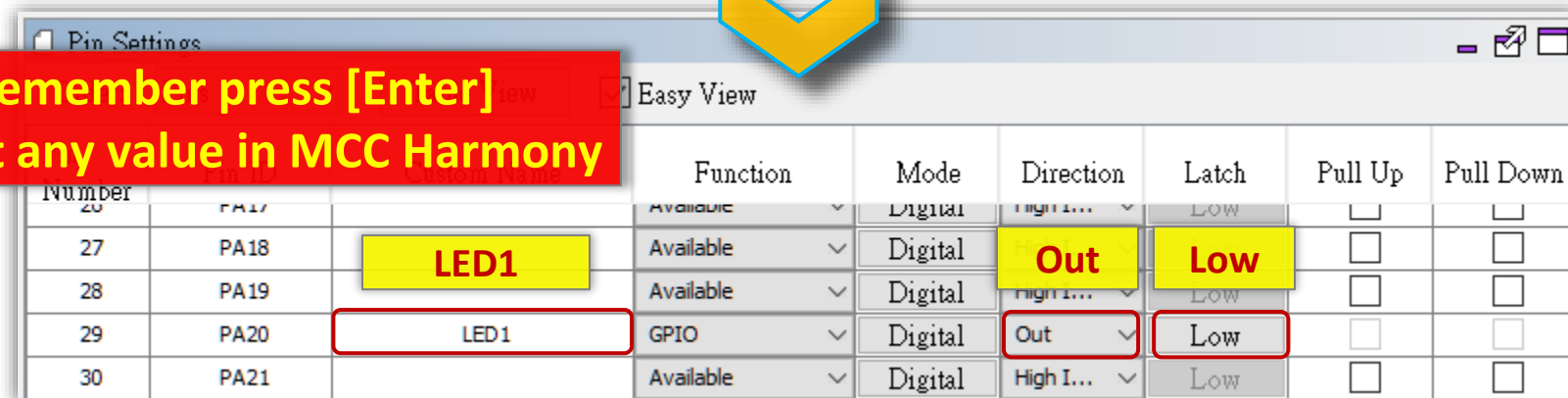


Pin Settings

Order: Pins Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down
20	PA17		Available	Digital	High I...	Low	☐	☐
27	PA18		Available	Digital	High I...	Low	☐	☐
28	PA19		Available	Digital	High I...	Low	☐	☐
29	PA20	GPIO_PA20	GPIO	Digital	High I...	Low	☐	☐
30	PA21		Available	Digital	High I...	Low	☐	☐

Always remember press [Enter]
after to input any value in MCC Harmony



Pin Settings

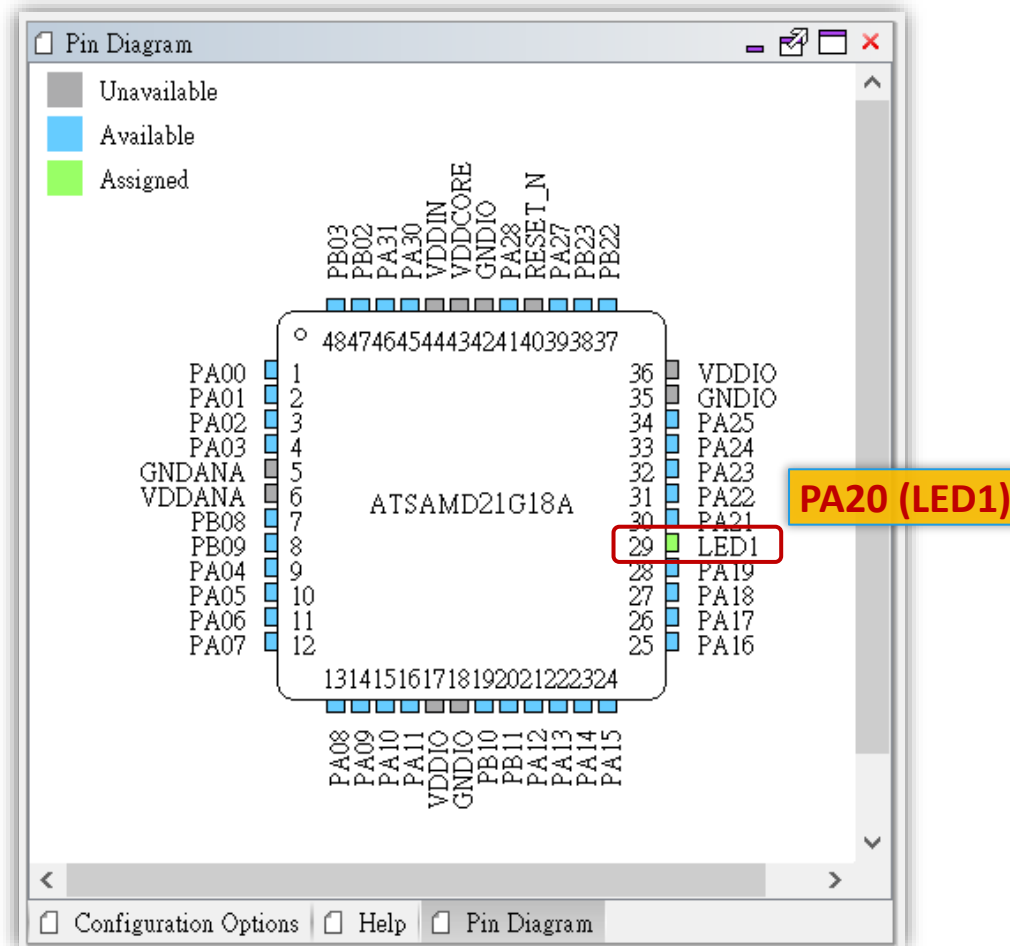
Order: Pins Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down
20	PA17		Available	Digital	High I...	Low	☐	☐
27	PA18	LED1	Available	Digital	High I...	Low	☐	☐
28	PA19		Available	Digital	High I...	Low	☐	☐
29	PA20	LED1	GPIO	Digital	Out	Low	☐	☐
30	PA21		Available	Digital	High I...	Low	☐	☐

Lab1 PORT Output

Step 5

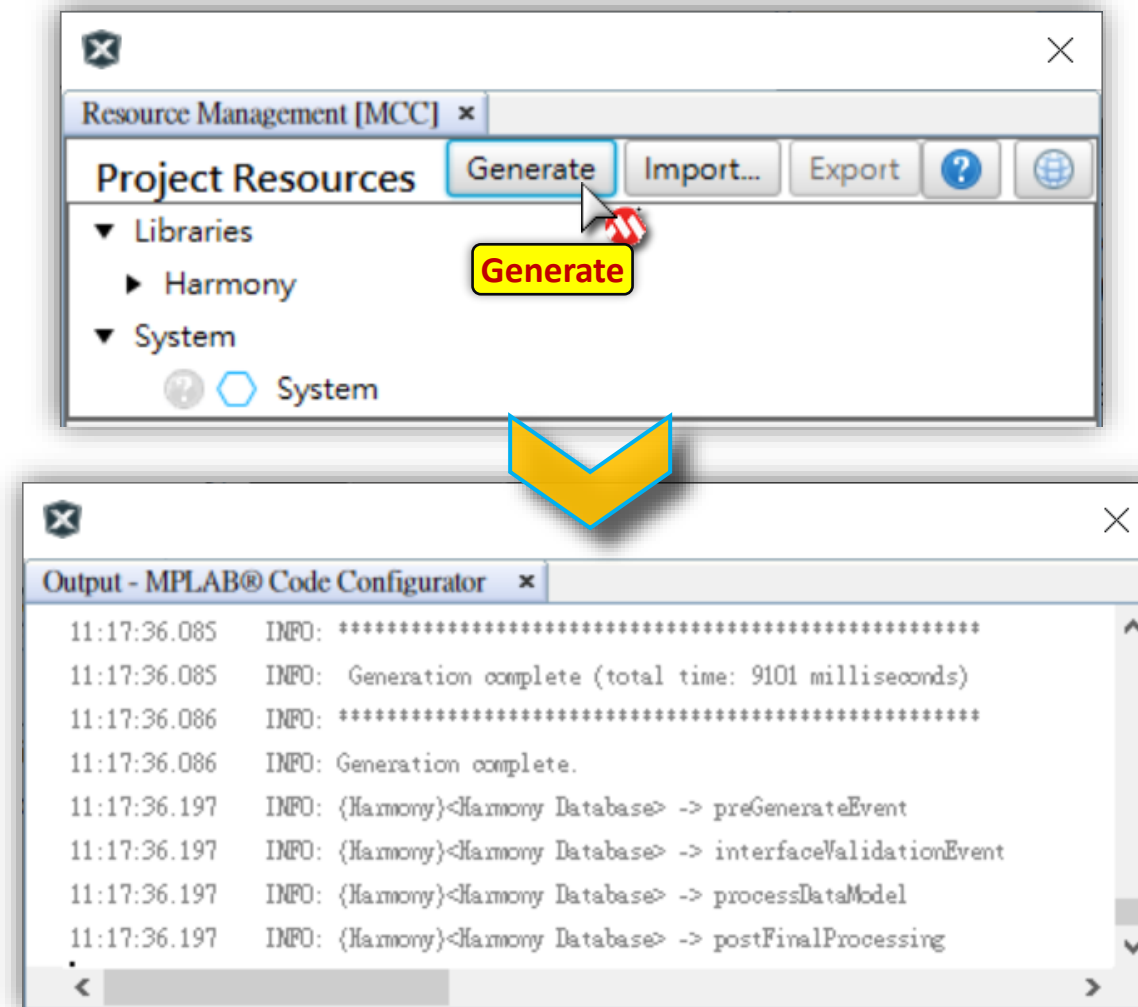
- Pin Diagram. (Hardware design reference)



Lab1 PORT Output

Step 6

- Click "**Generate**" Button to Generate your Code.



MCC Harmony GPIO Macros

Plib_port.h

/** Macros for LED1 pin **/

```
#define LED1_Set()          (PORT_REGS->GROUP[0].PORT_OUTSET = 1 << 20)
#define LED1_Clear()       (PORT_REGS->GROUP[0].PORT_OUTCLR = 1 << 20)
#define LED1_Toggle()      (PORT_REGS->GROUP[0].PORT_OUTTGL = 1 << 20)
#define LED1_Get()         (((PORT_REGS->GROUP[0].PORT_IN >> 20)) & 0x01)
#define LED1_OutputEnable() (PORT_REGS->GROUP[0].PORT_DIRSET = 1 << 20)
#define LED1_InputEnable() (PORT_REGS->GROUP[0].PORT_DIRCLR = 1 << 20)
#define LED1_PIN           PORT_PIN_PA20
```

Plib_port.c

void **PORT_Initialize**(void)

```
{
    /******* GROUP 0 Initialization *****/
    PORT_REGS->GROUP[0].PORT_DIR = 0x100000;
    PORT_REGS->GROUP[0].PORT_OUT = 0x100000;

    /******* GROUP 1 Initialization *****/
}
```

Lab1 PORT Output

Step 7

a Modify code segment to your main loop

```
// TODO 1.01
uint32_t i = 0;

int main (void)
{
    SYS_Initialize ( NULL );

    // TODO 1.02
    //PORT_REGS->GROUP[0].PORT_DIRSET = 1<<20;

    while( true )
    {
        for ( i = 0 ; i < 2000000 ; i++ );

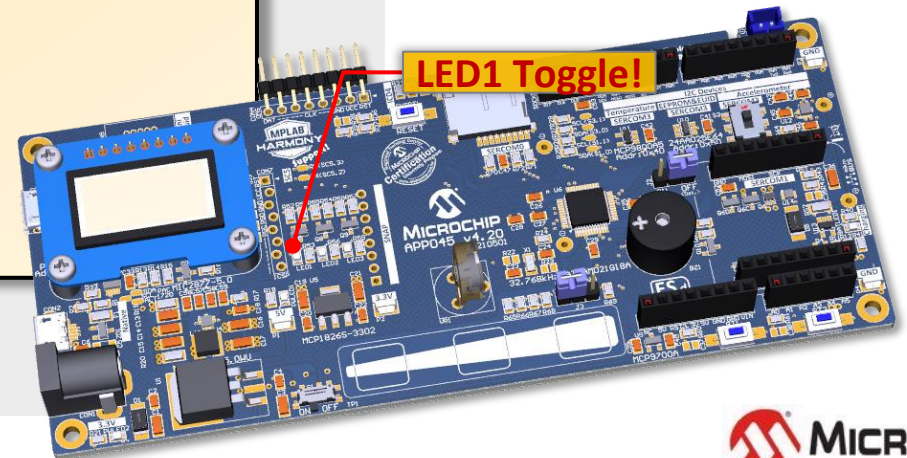
        // TODO 1.03
        LED1_Toggle();
    }
}
```

PORT initialization already done here

Comment out this line

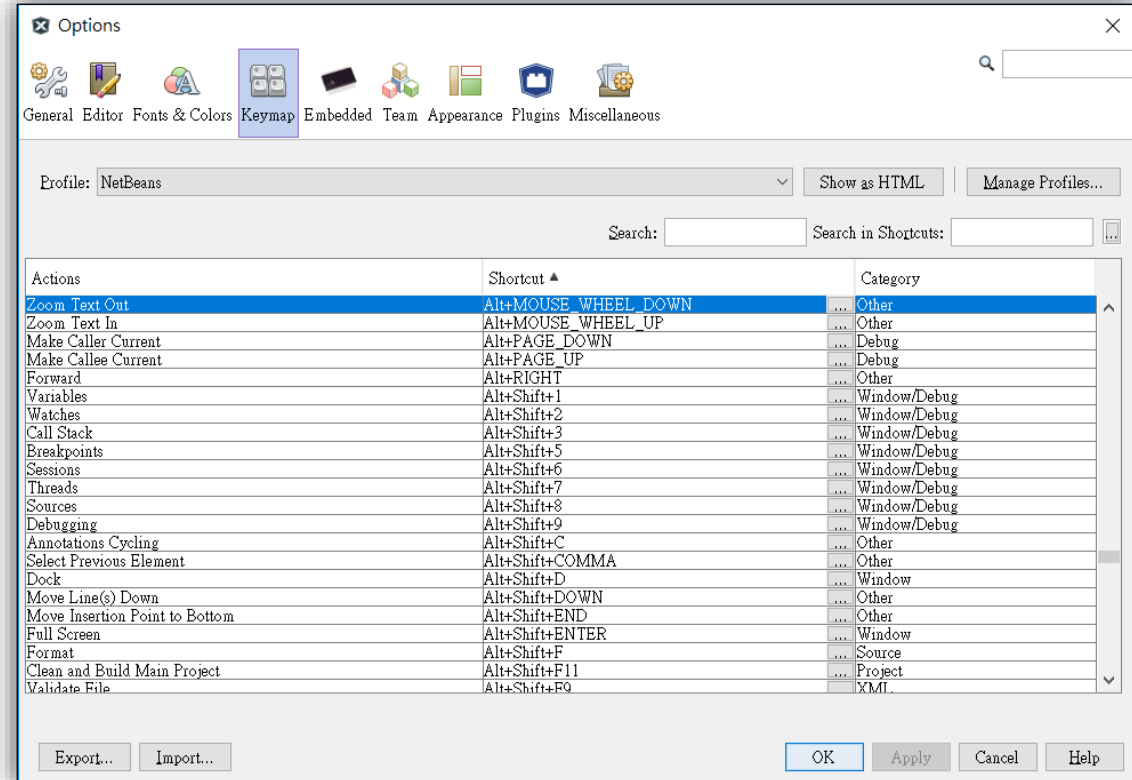
LED1 Toggle!

b Program firmware to target board then observe result.



MPLAB X IDE Shortcut key

- **MPLAB X IDE Provide a few shortcut key to enhance your performance.**
 - **Ctrl + Alt + ** : Show All Code Completion
(Alternative shortcut : Ctrl + \ , Ctrl + Alt + Space)
 - **Ctrl + Mouse Left Click** : Open Macro (Code Trace)
 - **Alt + Left** : Backward (Code Trace)
 - **Alt + Shift + F** : Code Format
 - **Ctrl + Shift + C** : Toggle Comment
- **More ?**
 - Tools > Options > Keymap

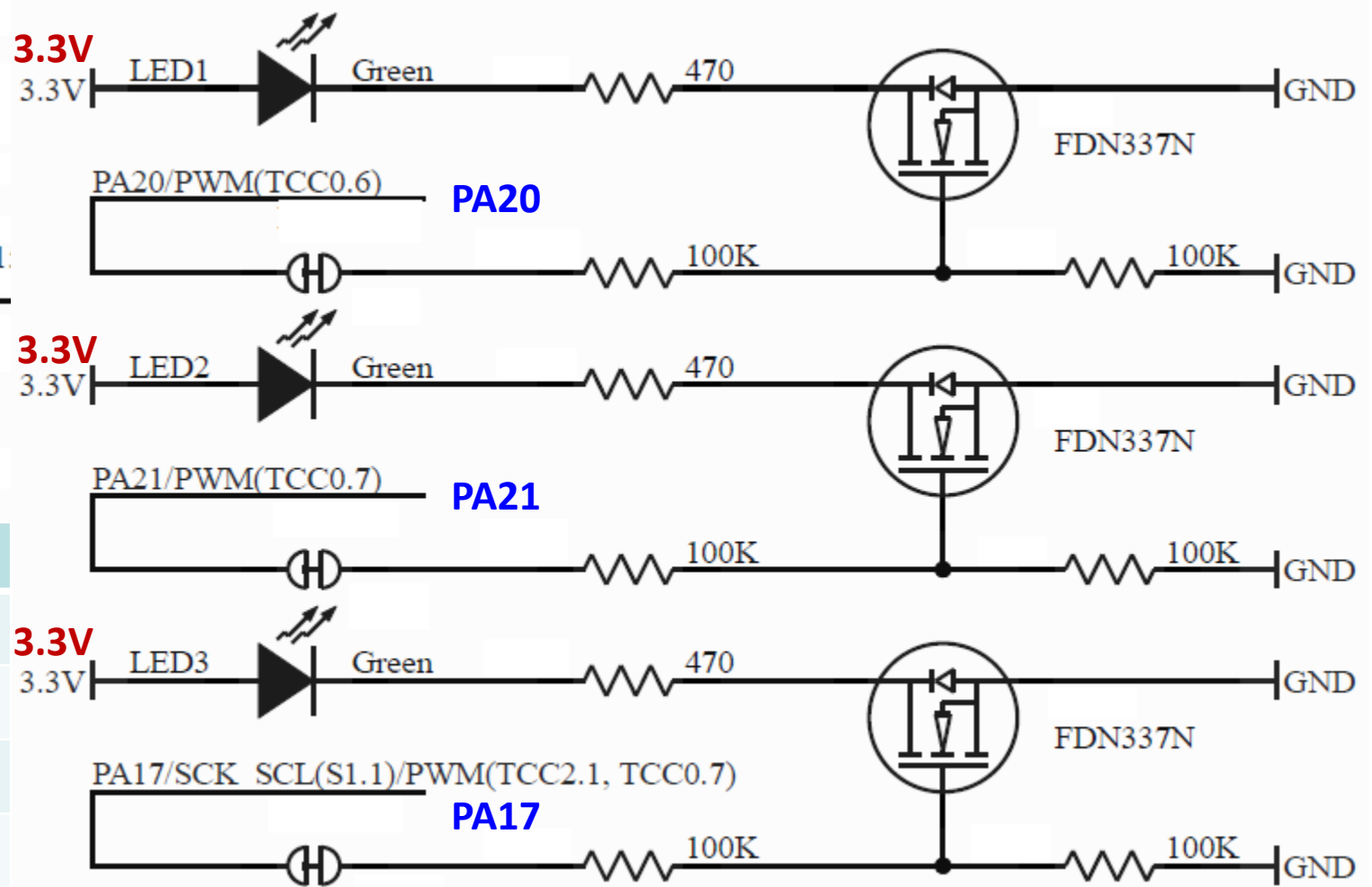
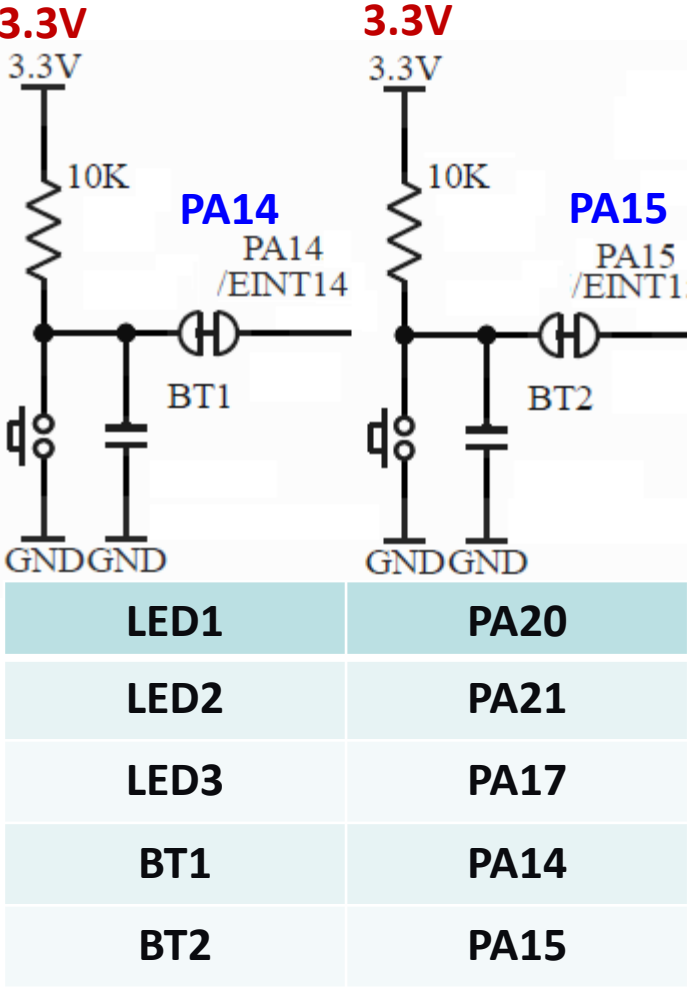


Lab2 PORT Input and Output

- Based on Lab1, Try to add LED2(PA21) continues toggle (Period around 500ms ~ 1S).
- Get BT1(PA14) status then use to control LED3(PA17).
Pressed -> LED3 Light
Released -> LED3 Dark

• **Let's go!**

Check Schematic



Lab2 PORT Input and Output

Step 1

- **Pin Table configure. (Enable or Disable Pin)**

[illegible]

Lab2 PORT Input and Output

Step 2

- Pin Setting. (Pin Function Setting)

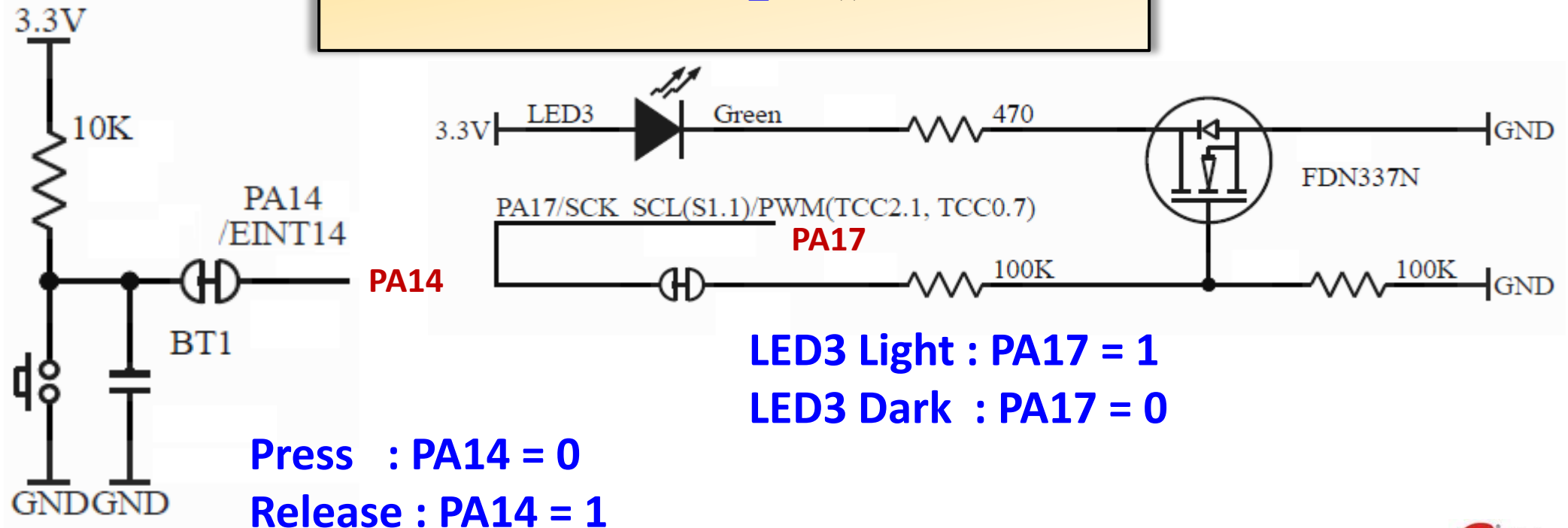
Pin Settings									
Order: Pins		Table View		☒ Easy View					
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
22	PA13		Available	Digital	High Impedance	Low	☐	☐	NORMAL
PA14	PA14	BT1	GPIO	Digital	In	Low	☐	☐	NORMAL
PA15	PA15	BT2	GPIO	Digital	In	Low	☐	☐	NORMAL
25	PA16		Available	Digital	High Impedance	Low	☐	☐	NORMAL
PA17	PA17	LED3	GPIO	Digital	Out	Low	☐	☐	NORMAL
27	PA18		Available	Digital	High Impedance	Low	☐	☐	NORMAL
28	PA19		Available	Digital	High Impedance	Low	☐	☐	NORMAL
PA20	PA20	LED1	GPIO	Digital	Out	Low	☐	☐	NORMAL
PA21	PA21	LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL
31	PA22		Available	Digital	High Impedance	Low	☐	☐	NORMAL

Lab2 PORT Input and Output

Schematic Describe

- Get **BT1(PA14)** status then use to control **LED3(PA17)**.
Pressed -> LED3 Light , Released -> LED3 Dark

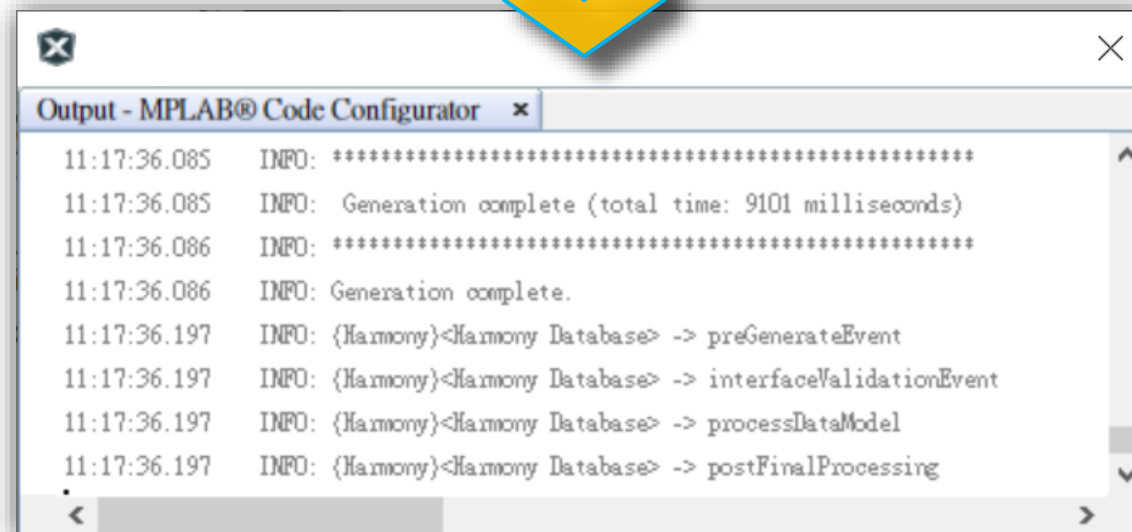
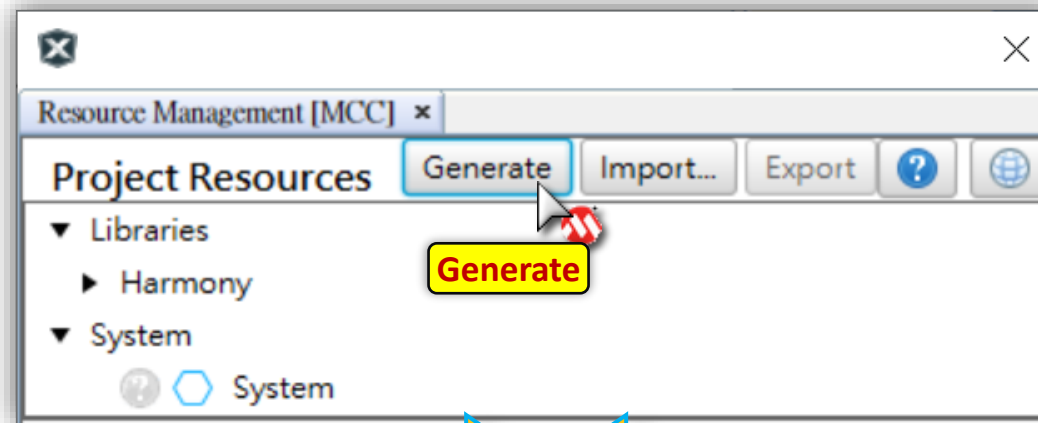
```
if ( BT1_Get() ) LED3_Clear();  
else  
    LED3_Set();
```



Lab2 PORT Input and Output

Step 3

- Click "**Generate**" Button to Generate your Code.



Lab2 PORT Input and Output

Step 4

a Add code segment to your main loop

```
uint32_t i = 0;
int main (void)
{
    SYS_Initialize ( NULL );

    while( true )
    {
        for ( i = 0 ; i < 2000000 ; i++ );

        // TODO 2.01
        LED1_Toggle();
        LED2_Toggle();

        // TODO 2.02
        if ( BT1_Get() ) LED3_Clear();
        else             LED3_Set();
    }
}
```

b Program firmware to target board then observe result.



Lab2 PORT Input and Output

Button Response Discuss

- Button Response Time Issue ?

The software delay will influence the polling interval.

Try to change coding style to state machine mode.

```
while( true )
{
    for ( i = 0; i < 2000000 ; i++ )
    {}

    LEDs_Toggle();

    if ( Button_Detected() )
        LED_Light();
    else
        LED_Dark();
}
```



```
while( true )
{
    if ( i++ > 2000000 )
    {
        i = 0;

        LEDs_Toggle();
    }

    if ( Button_Detected() )
        LED_Light();
    else
        LED_Dark();
}
```

Lab2 PORT Input and Output

Step 5

a change coding style to state machine mode

```
uint32_t i = 0;
int main (void)
{
    SYS_Initialize ( NULL );

    while( true )
    {
        if ( i++ > 2000000 )
        {
            i = 0;

            // TODO 2.01
            LED1_Toggle();
            LED2_Toggle();
        }

        // TODO 2.02
        if ( BT1_Get() ) LED3_Clear();
        else             LED3_Set();
    }
}
```

**How about current LED toggle speed after
change to state machine flow control ?
Why ?**

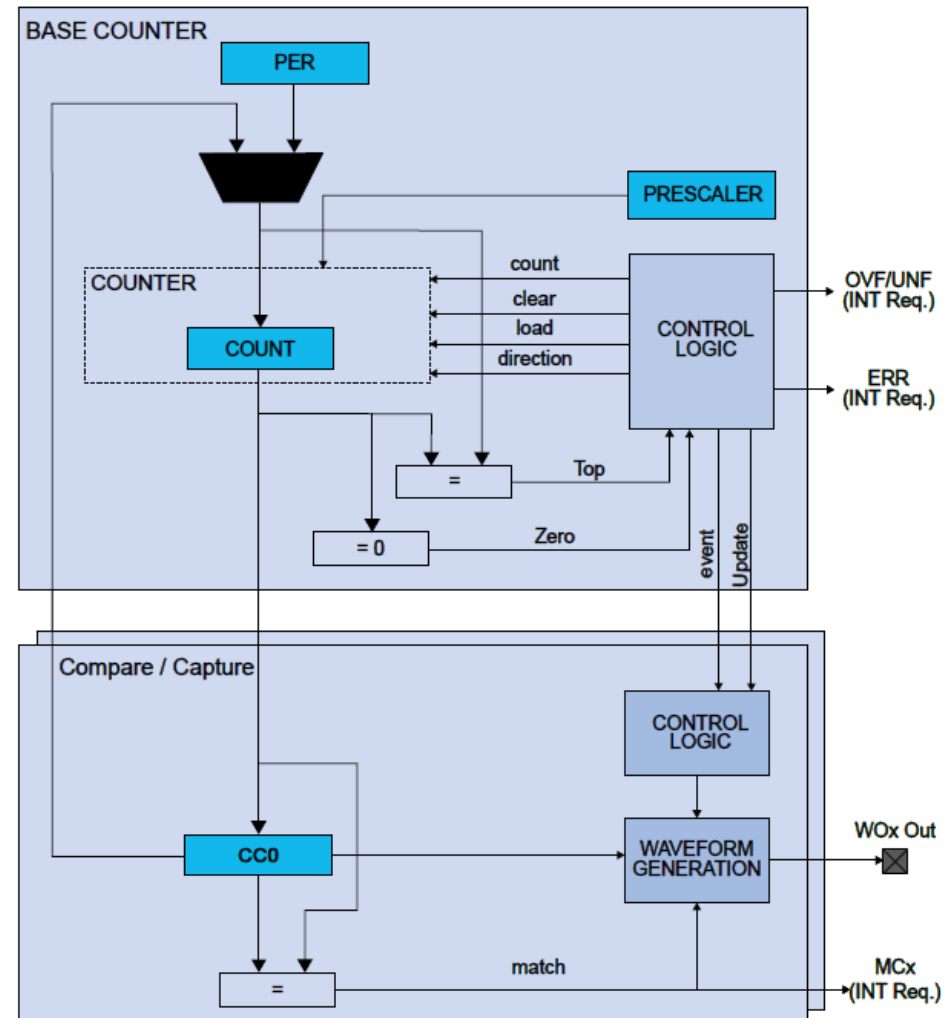
Timer/Counter, TC Architecture

SAMD21 Timer/Counter

- The SAMD21 Timer/Counter, TC module consists 3 function Counter, Compare and Capture.
- The counter can be set to count events, or it can be configured to count clock pulses. The counter, together with the compare/capture channels, can be configured to timestamp input events, allowing capture of frequency and pulse width.
- It can also perform waveform generation, such as frequency generation and pulse-width modulation (PWM).

TC Block Diagram

- SAMD21G18A TC Block Diagram

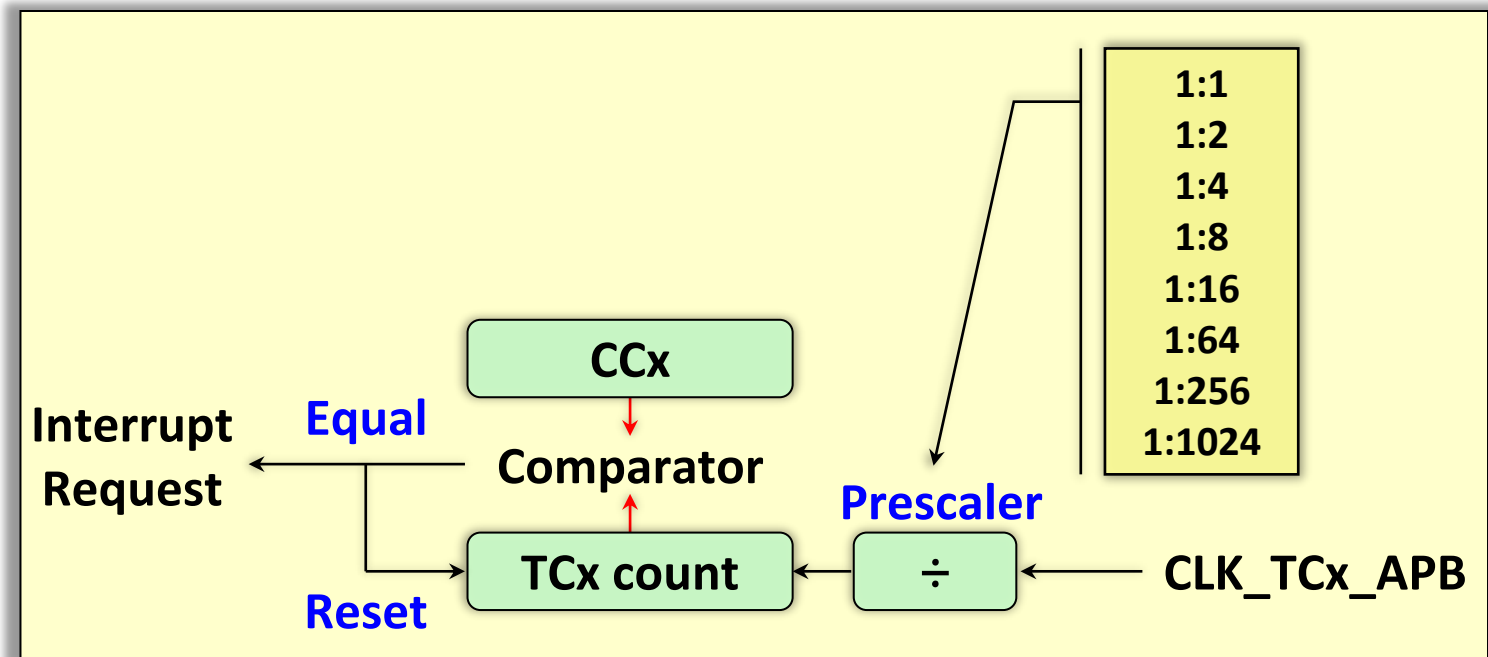


SAMD21 Timer/Counter

- Please refer to the block diagram, Just a concept. The real and detail block diagram please refer to the following slide.
- Clock into TCx after prescaler. Comparator continuously compares the values of **TCx** and **CCx**.

Asserts a into the interrupt request when equal, and resets TCx to zero.

- You can fill a value to CCx to determine period you want.



Match Pulse-Width Modulation (MPWM), 16Bit, Up count

Timer/Counter

- **TC provide 4 difference operation mode for Timer, Counter and Compare.**

This training only force on MFRQ mode. It's easy to use.

if you want to know more, please refer appendix [Timer/Counter Operation Mode](#).

- **Normal Frequency Generation (NFRQ)**

toggled on each compare match

- **Match Frequency Generation (MFRQ)**

toggles on each update condition

- **Normal Pulse-Width Modulation Operation (NPWM)**

toggles on each match & update condition

- **Match Pulse-Width Modulation Operation (MPWM)**

toggles on each match & update condition

Docking MCC Harmony windows

- Return to Harmony and docking windows as below.

The screenshot displays the MPLAB X IDE v6.05 interface for the Lab0_First_Project. The MCC Harmony configuration windows are docked as follows:

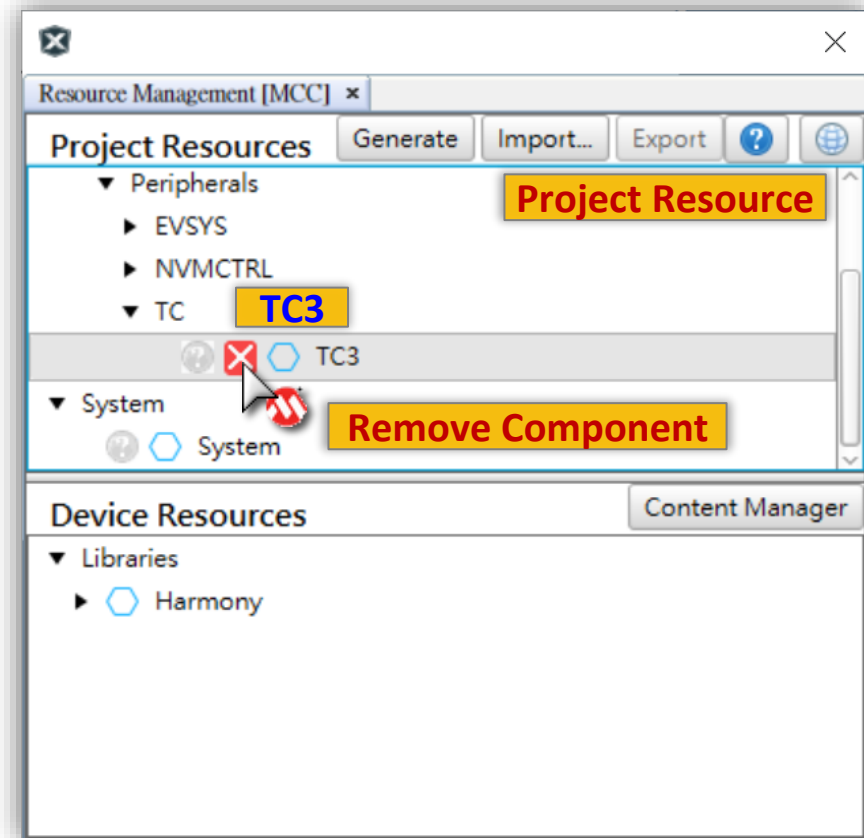
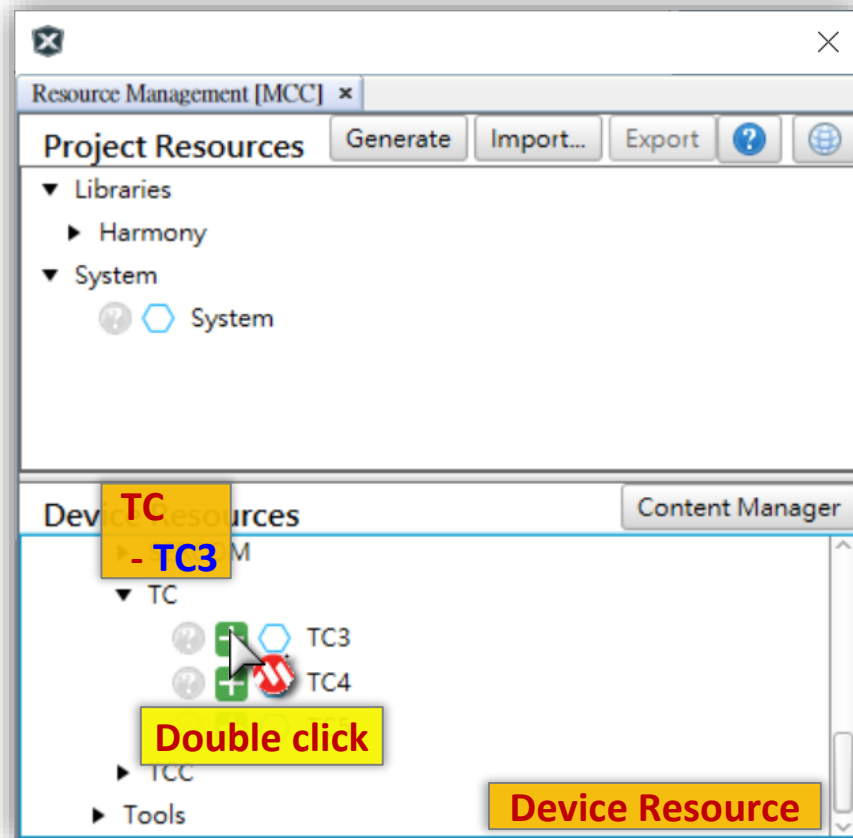
- Project Resource:** Located in the top-left pane, showing a tree view of project resources including Libraries (Harmony, Packs, Peripherals, EVSYS) and Device Resources (Harmony).
- Device Resource:** Located in the bottom-left pane, showing the Content Manager for the selected device.
- MCC Core Versions:** Located in the bottom-left pane, showing the installed version of the MPLAB® Code Configurator (Plugin) v5.2.2.
- Pin Settings:** A table in the center pane showing the configuration for pins. The table has columns for Pin Number, Pin ID, Custom Name, and Function. Pins 1 through 12 are listed, with PA00 through PA07 and GNDANA, VDDANA, PB08.
- Configuration Options:** A pane on the right showing the configuration for the TC3 timer. It includes settings for Counter Mode (Counter in 16-bit mode), Select Prescaler (Prescaler: GCLK_TC), Operating Mode (Timer), and Sleep Configurations.
- Pin Table:** A table at the bottom showing the pin configuration for the TQFP48 package. It lists the Module (DAC), Function (DAC_VOUT, DAC_VREFP), and the corresponding Pin Number (1 through 18).

Pin Number	Pin ID	Custom Name	Function
1	PA00		Available
2	PA01		Available
3	PA02		Available
4	PA03		Available
5	GNDANA		
6	VDDANA		
7	PB08		Available
8			Available
9	PA04		Available
10	PA05		Available
11	PA06		Available
12	PA07		Available

Module	Function	PA00	PA01	PA02	PA03	GNDANA	VDDANA	PB08	PB09	PA04	PA05	PA06	PA07	PA08	PA09	PA10	PA11	VDDIO	GNDIO
DAC	DAC_VOUT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
DAC	DAC_VREFP																		

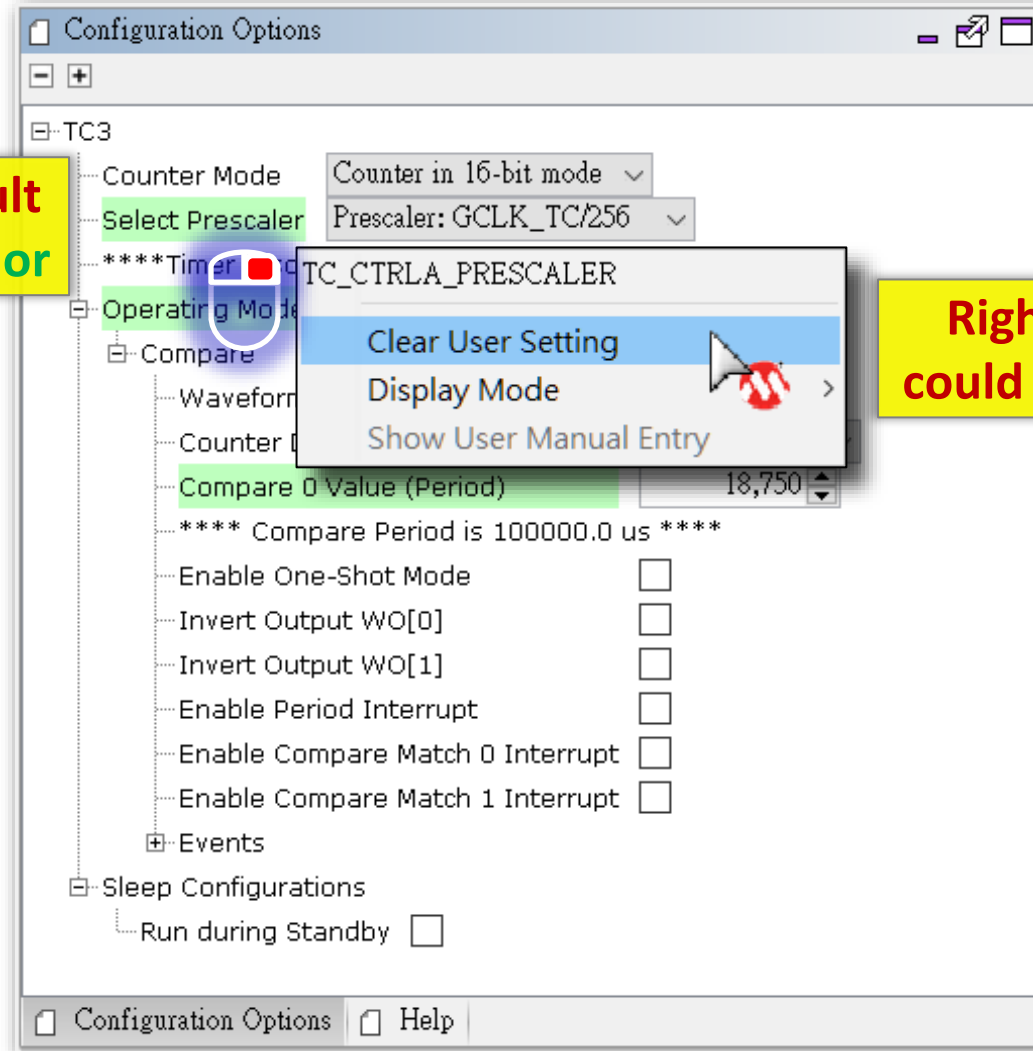
Add TC Function using MCC Harmony

- Find **TC3** component in Device Resource window.
- Double click **TC3** to add to Project Resource window.
- Click  could remove component from Project Resource window.



TC Configuration Options Example

Value different to default
will change to green color



Right click the value
could Clear User Setting

TC polling

Interface Routines

```
void TC3_TimerInitialize( void );  
void TC3_TimerStart( void );  
void TC3_TimerStop( void );  
uint32_t TC3_TimerFrequencyGet( void );  
void TC3_Timer16bitPeriodSet( uint16_t period );  
uint16_t TC3_Timer16bitPeriodGet( void );  
uint16_t TC3_Timer16bitCounterGet( void );  
void TC3_Timer16bitCounterSet( uint16_t count );  
bool TC3_TimerPeriodHasExpired ( void );
```

TC Interrupt Flag Event

- **The TC has the following interrupt sources:**
 - **Overflow/Underflow (OVF)**
Counter overflow/underflow
 - **Match or Capture Channel x (MCx)**
Compare match or capture
 - **Capture Overflow Error (ERR)**
New capture interrupt occurs while interrupt flag is set.
 - **Synchronization Ready (SYNCRDY)**
Read/Write synchronization ready.
- **How to polling TC overflow interrupt flag status ?**

TC Polling Code Example

TC Timer Method (MPWM Mode) Overflow Polling Example

```
...
TC3_TimerStart();

while( true )
{
    if (TC3_TimerPeriodHasExpired ())
    {
        LED1_Toggle();
    }
}
```



```
bool TC3_TimerPeriodHasExpired( void )
{
    bool timer_status;
    timer_status = (bool) ((TC3_REGS->COUNT16.TC_INTFLAG) & TC_INTFLAG_OVF_Msk);
    TC3_REGS->COUNT16.TC_INTFLAG = timer_status;
    return timer_status;
}
```

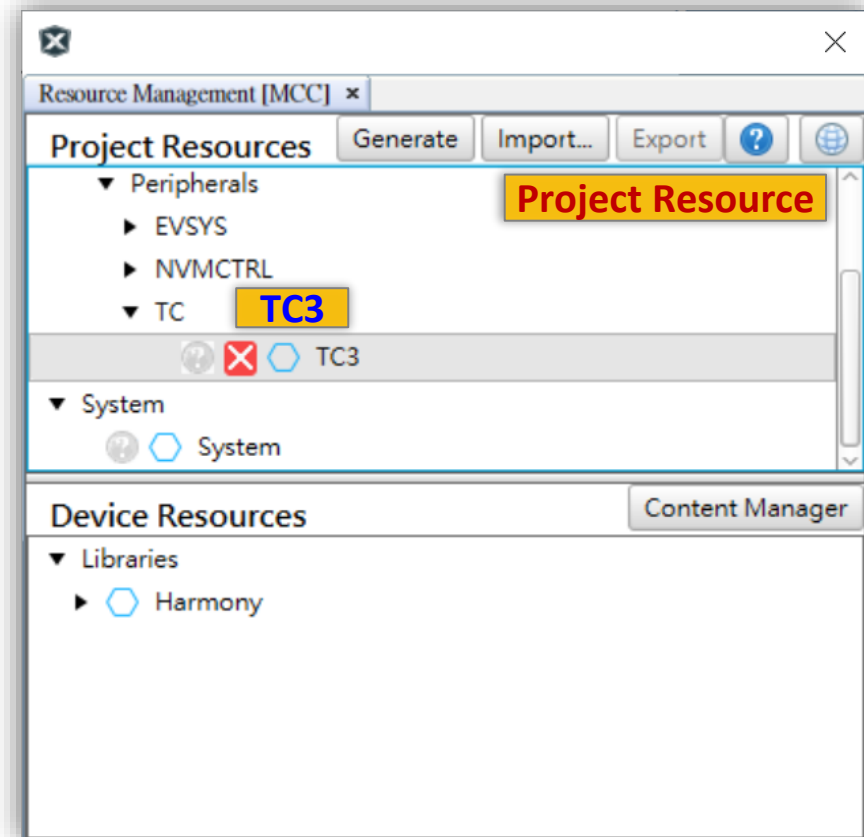
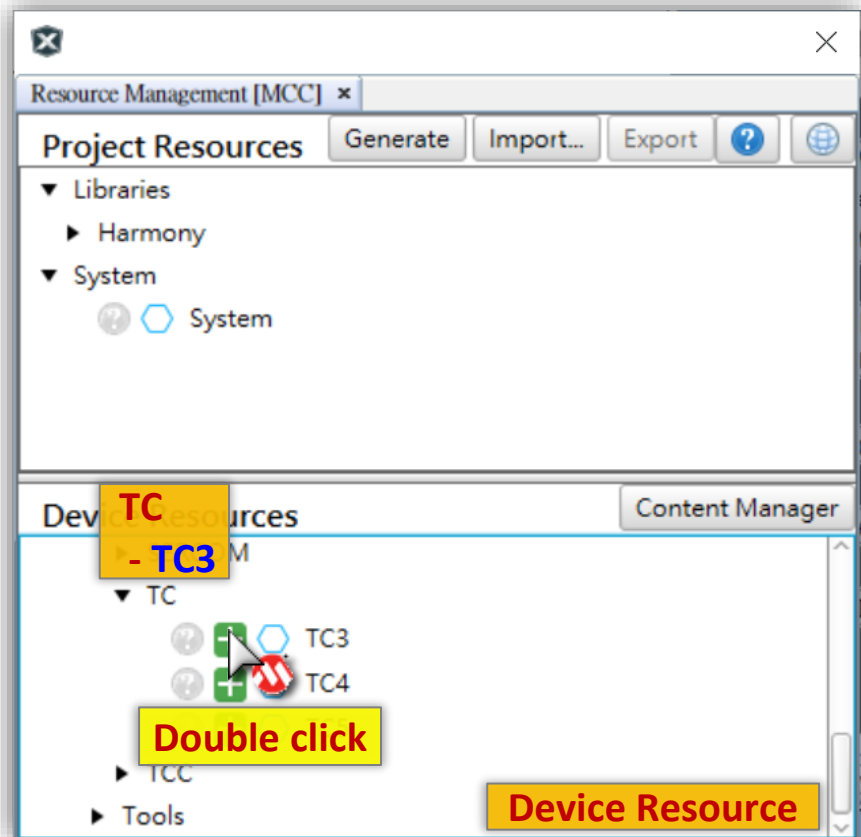
Lab3 TC Timer Method Polling

- Try add **TC3** to replace software delay for control **LED1(PA20)** to continues toggle.
- The **TC3** setup to **Timer** Mode (It's implemented by **MPWM** 16-bit mode), and setup the Timer period to **100ms**.
- **Let's go!**

Lab3 TC Timer Method Polling

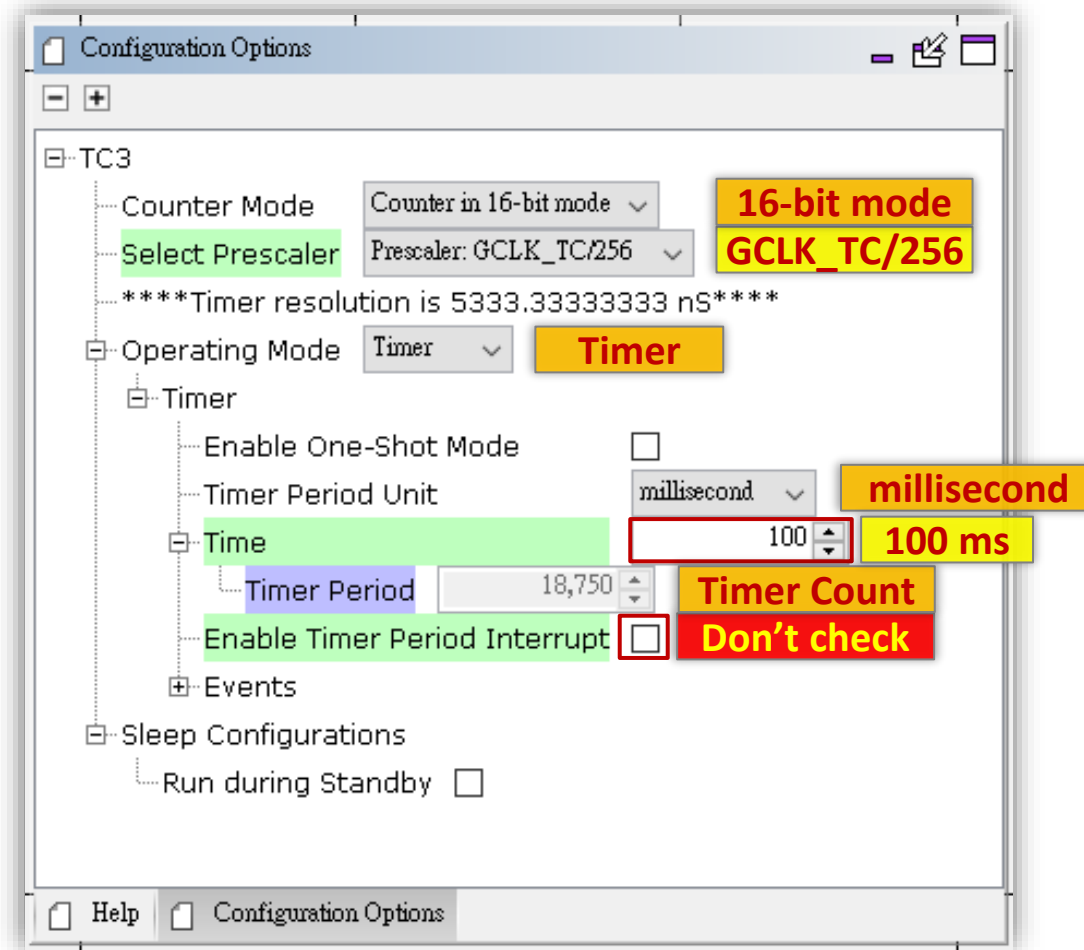
Step 1

- Find **TC3** component in Device Resource window.
- Double click **TC3** to add to Project Resource window.



Lab3 TC Timer Method Polling

Step 2



System Clock = 48MHz

One second counts of system clock :
48,000,000 counts

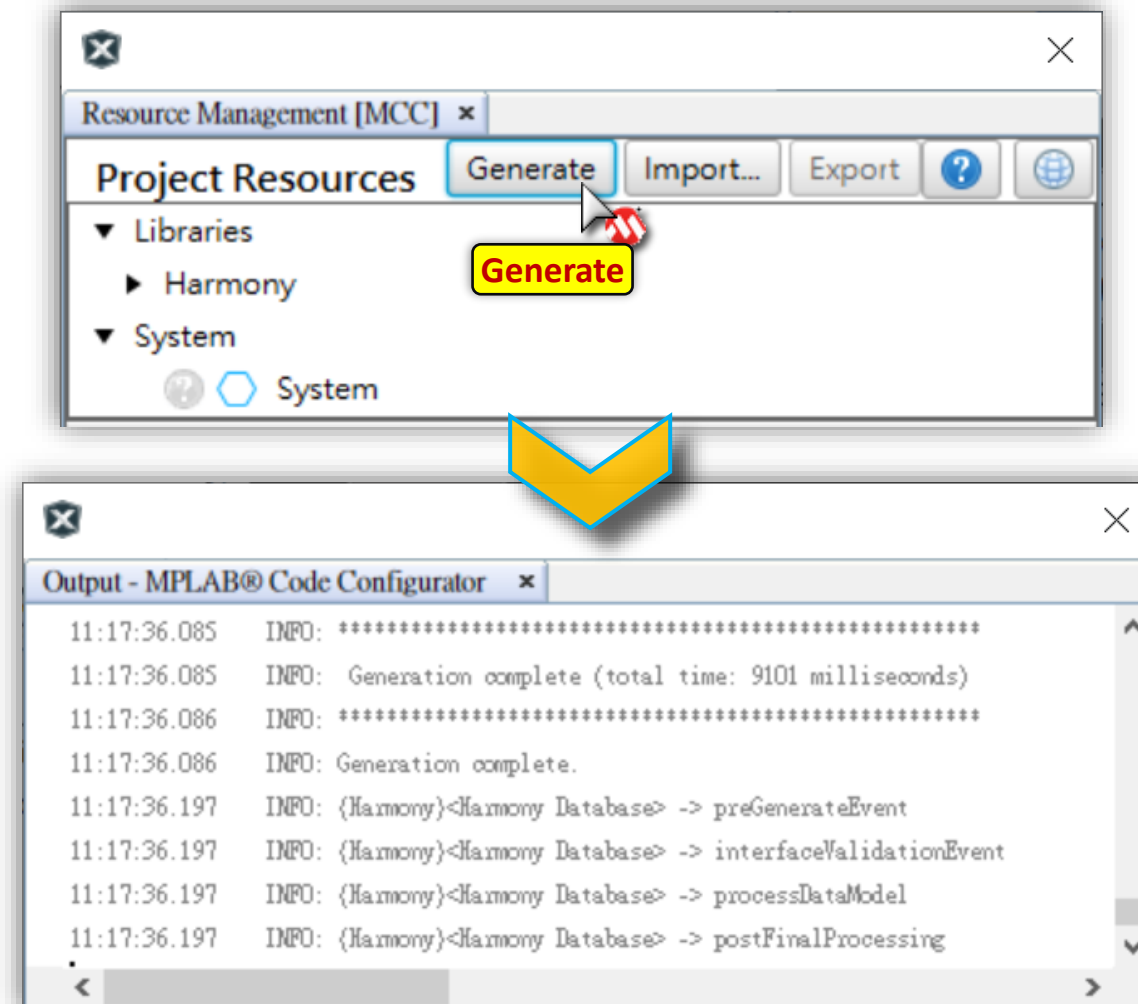
One second counts after Prescaler :
 $48000000 / 256$ 187,500 counts

100ms counts for LAB3 target :
 $187500 / 10$ 18,750 counts

Lab3 TC Timer Method Polling

Step 3

- Click "**Generate**" Button to Generate your Code.



Lab3 TC Timer Method Polling

Step 4

a Add code segment to your main loop

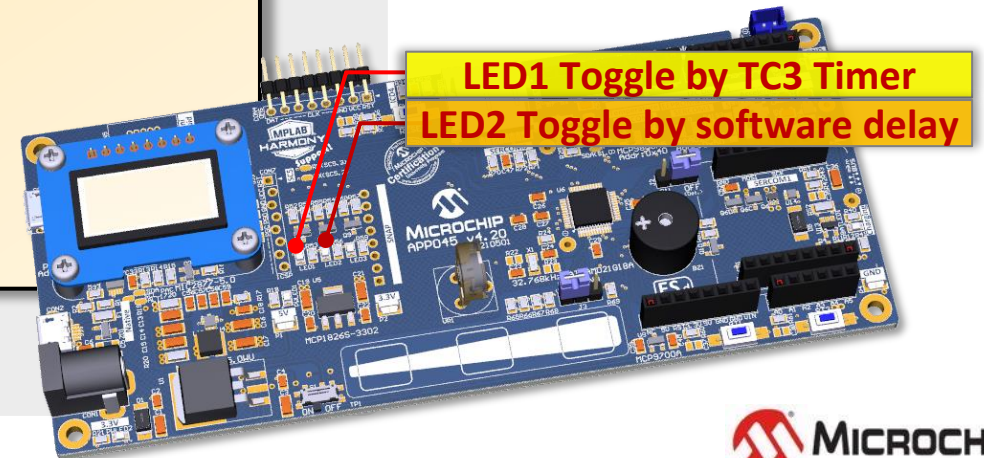
```
...
int main (void)
{
    SYS_Initialize ( NULL );

    // TODO 3.01
    TC3_TimerStart();

    while( true )
    {
        // TODO 3.02
        if ( TC3_TimerPeriodHasExpired() )
        {
            LED1_Toggle();
        }
        ...
    }
}
```

Period Control

b Program firmware to target board then observe result.



LED1 Toggle by TC3 Timer
LED2 Toggle by software delay

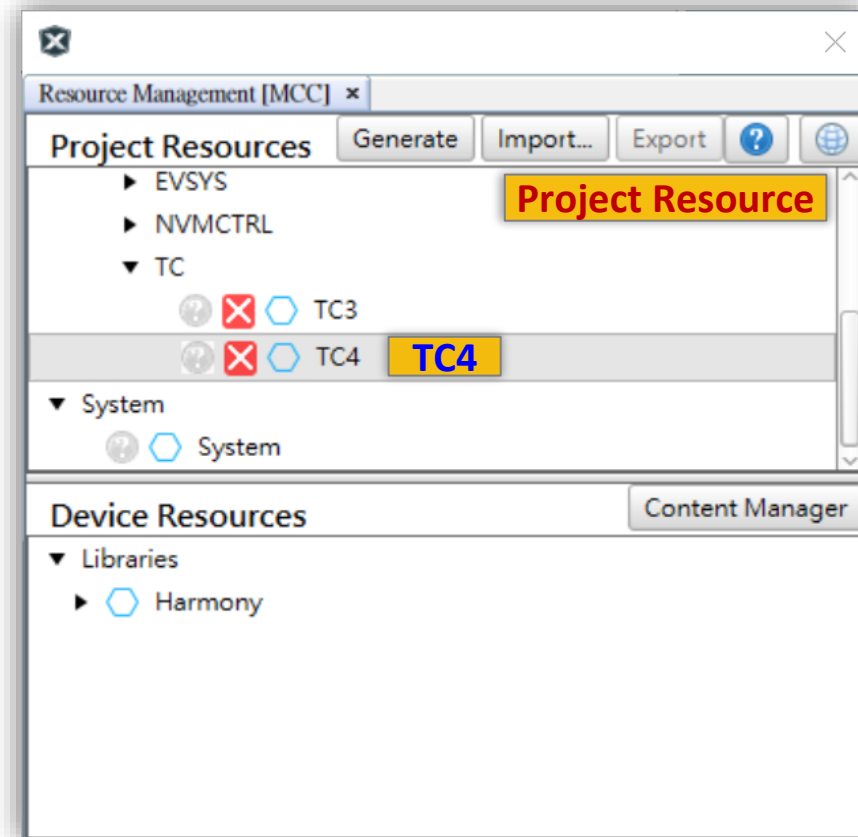
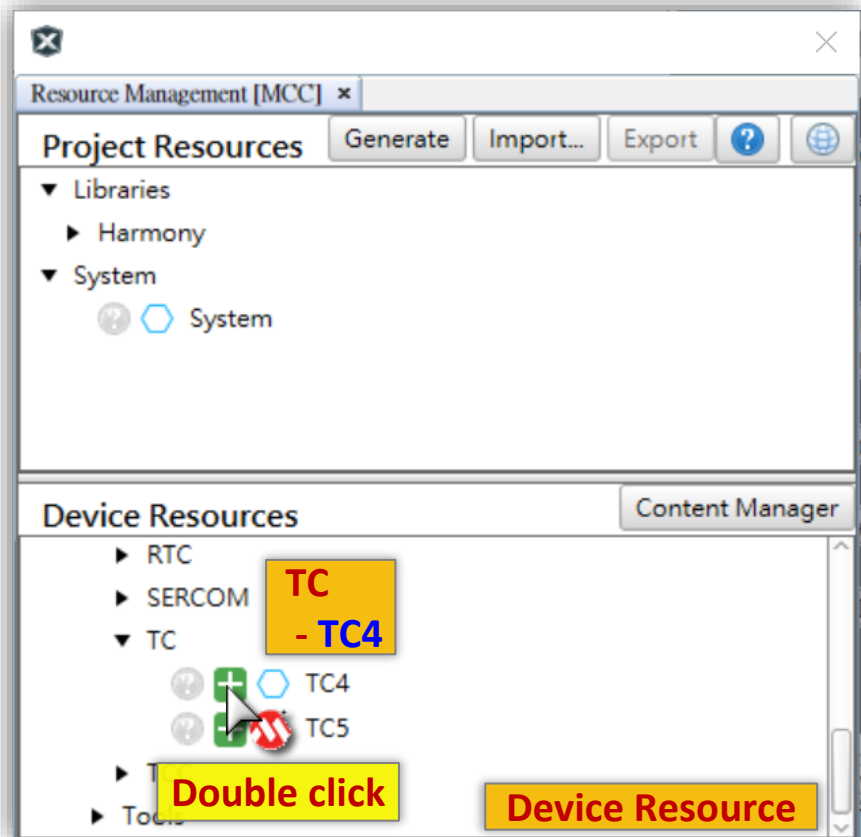
Lab4 TCs Timer Method Polling

- Try add **TC4** to replace software delay for control **LED2(PA21)** to continues toggle.
- The **TC4** setup to **Timer** Mode (It's implemented by **MPWM** 16-bit mode), and setup the Timer period to **50ms**.
- **Let's go!**

Lab4 TCs Timer Method Polling

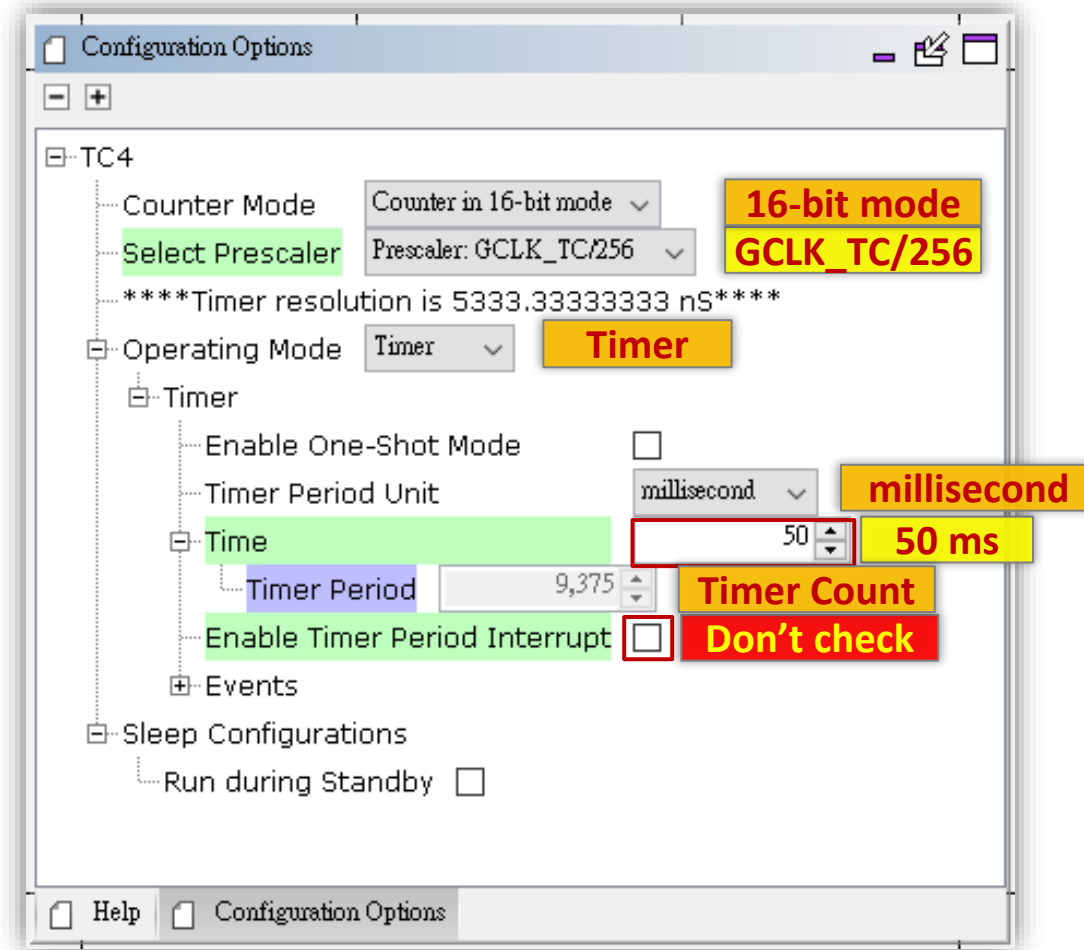
Step 1

- Find **TC4** component in Device Resource window.
- Double click **TC4** to add to Project Resource window.



Lab4 TCs Timer Method Polling

Step 2



System Clock = 48MHz

One second counts of system clock :
48,000,000 counts

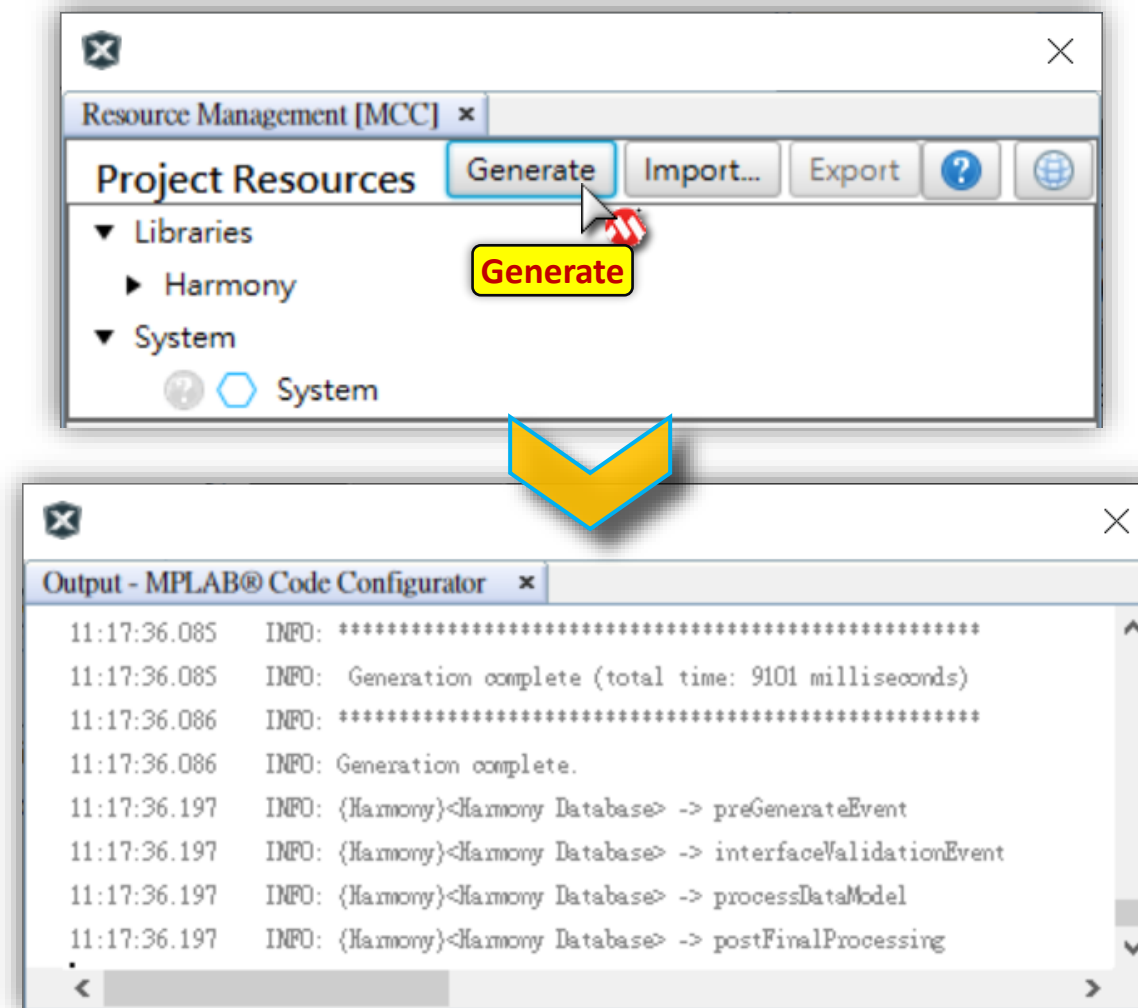
One second counts after Prescaler :
 $48000000 / 256$ 187,500 counts

50ms counts for LAB4 target :
 $187500 / 20$ 9,375 counts

Lab4 TCs Timer Method Polling

Step 3

- Click "**Generate**" Button to Generate your Code.



Lab4 TCs Timer Method Polling

Step 4

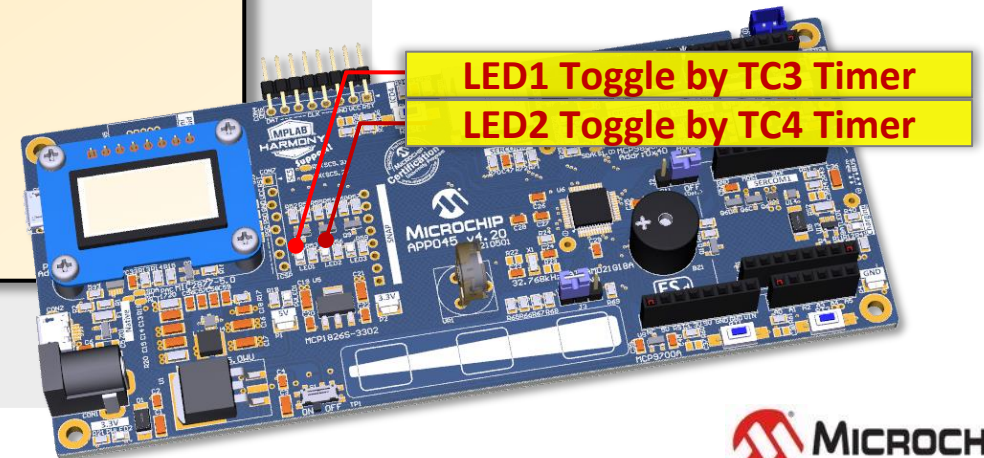
a Add code segment to your main loop

```
int main (void)
{
    SYS_Initialize ( NULL );
    ...
    TC3_TimerStart();
    // TODO 4.01
    TC4_TimerStart ();

    while( true )
    { ...
      // TODO 4.02
      if ( TC4_TimerPeriodHasExpired() )
      {
          LED2_Toggle();
      }
      ...
    }
}
```

Period Control

b Program firmware to target board then observe result.



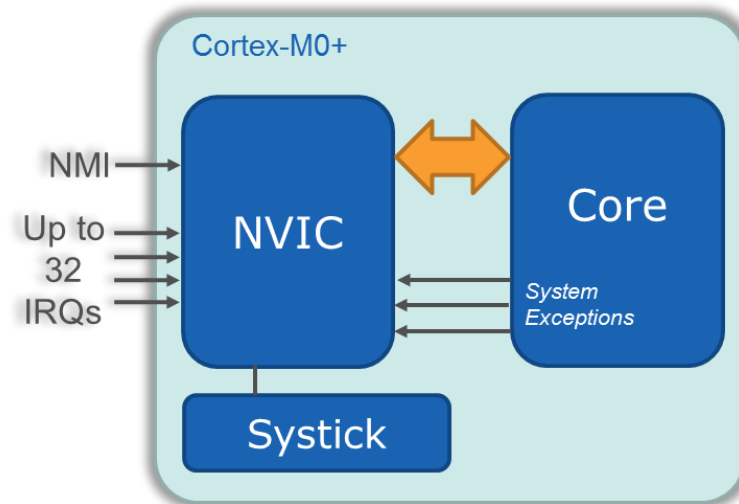
NVIC Architecture

Nested Vector Interrupt Controller Architecture

Nested Vector Interrupt Controller

ARM® Cortex® M0+

- NVIC provides an interface between interrupt sources (peripherals and external pins) and the core.
- The **priority (Level 0 ~ 3)** for each interrupt source is programmable, **Level 0 is highest priority**.
- Each interrupt lines has connected to one peripheral instance and **have one or more interrupt flags**.
- An interrupt request is generated from the peripheral when the **interrupt flag** has been set and the corresponding **interrupt is enabled**.



Natural Priority

- If two pending interrupts share the same priority, priority is given to the interrupt with the lowest exception number (lowest interrupt vector address).

Exception number	IRQ number	Vector	Offset
16+n	n	IRQn	0x40+4n
.	.	.	.
.	.	.	.
.	.	.	.
18	2	IRQ2	0x48
17	1	IRQ1	0x44
16	0	IRQ0	0x40
15	-1	SysTick, if implemented	0x3C
14	-2	PendSV	0x38
13	.	Reserved	.
12	.	.	.
11	-5	SVCall	0x2C
10	.	.	.
9	.	.	.
8	.	.	.
7	.	.	.
6	.	.	.
5	.	.	.
4	.	.	.
3	-13	HardFault	0x10
2	-14	NMI	0x0C
1	.	Reset	0x08
.	.	Initial SP value	0x04
.	.	.	0x00

TC3 (18)
TC4 (19)

Each peripheral has been assigned to one of Vector table and based to address 0x00000000

Peripheral Source	NVIC Line
EIC NMI – External Interrupt Controller	NMI
PM – Power Manager	0
SYSCTRL – System Control	1
WDT – Watchdog Timer	2
RTC – Real Time Counter	3
EIC – External Interrupt Controller	4
NVMCTRL – Non-Volatile Memory Controller	5
DMAC – Direct Memory Access Controller	6
USB – Universal Serial Bus	7
EVSYS – Event System	8
SERCOM0 – Serial Communication Interface 0	9
SERCOM1 – Serial Communication Interface 1	10
SERCOM2 – Serial Communication Interface 2	11
SERCOM3 – Serial Communication Interface 3	12
SERCOM4 – Serial Communication Interface 4	13
SERCOM5 – Serial Communication Interface 5	14
TCC0 – Timer Counter for Control 0	15
TCC1 – Timer Counter for Control 1	16
TCC2 – Timer Counter for Control 2	17
TC3 – Timer Counter 3	18
TC4 – Timer Counter 4	19
TC5 – Timer Counter 5	20
TC6 – Timer Counter 6	21
TC7 – Timer Counter 7	22
ADC – Analog-to-Digital Converter	23
AC – Analog Comparator	24
DAC – Digital-to-Analog Converter	25
PTC – Peripheral Touch Controller	26
I2S – Inter IC Sound	27

SAMD21G18A IVT Define

Interrupt.c

```
__attribute__((section(".vectors")))
const DeviceVectors exception_table=
{
    /* Configure Initial Stack Pointer, using linker-generated symbols */
    .pvStack = (void*) (&_stack),

    .pfnReset_Handler      = ( void * ) Reset_Handler,
    .pfnNonMaskableInt_Handler = ( void * ) NonMaskableInt_Handler,
    .pfnHardFault_Handler  = ( void * ) HardFault_Handler,
    .pfnSVCall_Handler     = ( void * ) SVCall_Handler,
    .pfnPendSV_Handler     = ( void * ) PendSV_Handler,
    .pfnSysTick_Handler    = ( void * ) SysTick_Handler,
    .pfnNVMCTRL_Handler    = ( void * ) NVMCTRL_Handler,
    ...
    .pfnSERCOM5_Handler    = ( void * ) SERCOM0_Handler,
    ...
    .pfnTCC2_Handler       = ( void * ) TCC2_Handler,
    .pfnTC3_Handler        = ( void * ) TC3_TimerInterruptHandler,
    .pfnTC4_Handler        = ( void * ) TC4_TimerInterruptHandler,
    ...
    .pfnADC_Handler        = ( void * ) ADC_Handler,
    ...
};
```

ISR Function Example

- So, You need Implement ISR (Interrupt Service Routine) by yourself like below ?

plib_tc3.c

```
#include "plib_tc3.h"
...

void TC3_TimerInterruptHandler( void )
{
    TC_TIMER_STATUS status;
    status = (TC_TIMER_STATUS) (TC3_REGS->COUNT16.TC_INTFLAG);
    /* Clear interrupt flags */
    TC3_REGS->COUNT16.TC_INTFLAG = TC_INTFLAG_Msk;

    // Getting starter add your code here now. !?

}
```

Now ? Hands-on Lab ?

No!

Callback Function

- MH3 module provide the **interrupt** mode driver.
- **All flag and event handling by MH3 interrupt interface routine automatically, user just focus on application.** It's more powerful and easy to understand than operating a NVIC ISR function directly.
- **The interrupt mode provide "callback" function to hook up your customer function to specific interrupt source and event.**
- **For Example,**
You can hook the LED toggle callback function in TC overflow event. **Callback function will be executed automatically** while TC overflow event has occurred.

TC Callback Hook Example

```
void TCn_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if( status & TC_INTFLAG_OVF_Msk )
        LED_Toggle();
}

TCn_TimerCallbackRegister (TCn_TimerExpired, (uintptr_t) NULL );
TCn_TimerStart ();
```

Hook (Register)

Callback

Application

System

```
/* Register callback function */
void TCn_TimerCallbackRegister( TC_TIMER_CALLBACK callback, uintptr_t context )
{
    TCn_CallbackObject.callback = callback;
    TCn_CallbackObject.context = context;
}
```

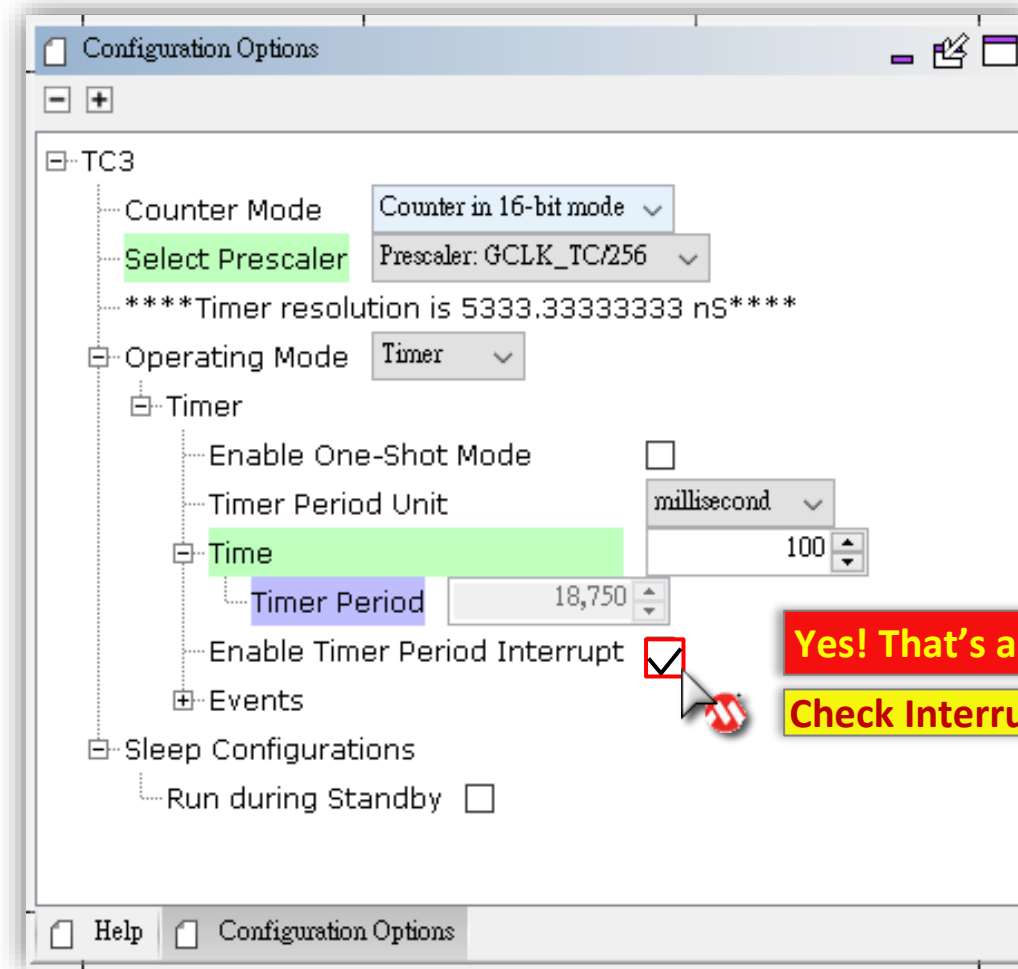
Function Pointer

```
/* Timer Interrupt handler */
void TCn_TimerInterruptHandler ( void )
{
    TC_TIMER_STATUS Sstatus;
    status = (TC_TIMER_STATUS) TCn_REGS->COUNT16.TC_INTFLAG;
    /* Clear interrupt flags */
    TCn_REGS->COUNT16.TC_INTFLAG = TC_INTFLAG_Msk;
    if( TCn_CallbackObject.callback != NULL )
    {
        TCn_CallbackObject.callback( status, TCn_CallbackObject.context );
    }
}
```

Interrupt

MCC Harmony Interrupt Example

- Enable Interrupt in TC configuration windows.
 - **Only one click !**

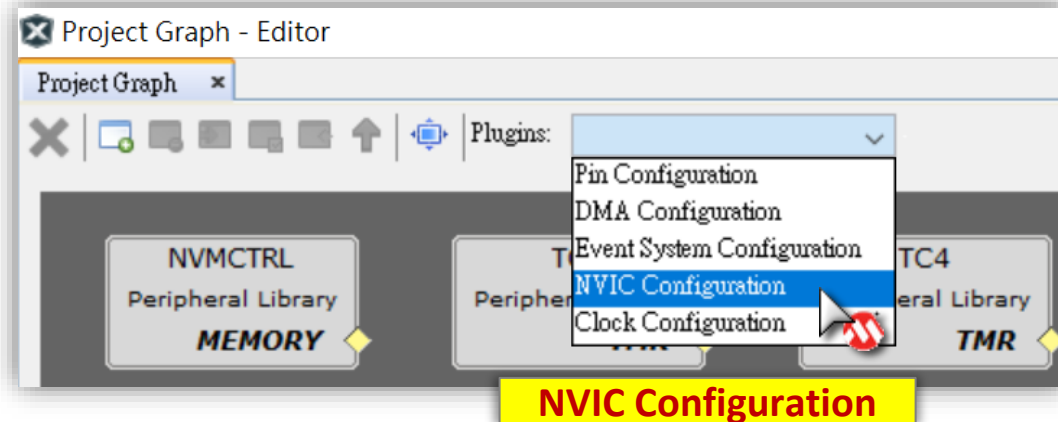


Yes! That's all you need to do.

Check Interrupt Enable

NVIC Priority Settings in MCC Harmony

- Open NVIC Settings windows in **Project Graph > Plugins > NVIC Configuration**.



NVIC Settings				
Vector Number	Vector	Enable	Priority (0 =	Handler Name
-15	Reset (Reset Vector)	☒	-3	Reset_Handler
-14	NonMaskableInt (Non-maskable Interrupt)	☒	-2	NonMaskableInt_Handler
-13	HardFault (Hard Fault)	☒	-1	HardFault_Handler
-5	SVCall (SuperVisor Call)	☒	0	SVCall_Handler
-2	PendSV (Pendable Service)	☒	0	PendSV_Handler
-1	SysTick (System Tick Timer)	☐	0	SysTick_Handler
0	PM (Power Manager)	☐	3	PM_Handler
1	SYSCTRL (System Controller)	☐	3	SYSCTRL_Handler
2	WDT (Watchdog Timer)	☐	3	WDT_Handler
3	RTC (Real Time Counter)	☐	3	RTC_Handler
4	EIC (External Interrupt Controller)	☐	3	EIC_Handler
5	NVMCTRL (Non-Volatile Memory Controller)	☐	3	NVMCTRL_Handler
6	DMAC (Direct Memory Controller)	☐	3	DMAC_Handler
7	USB (Universal Serial Bus)	☐	3	USB_Handler
8	EVSYS (Event Systems)	☐	3	EVSYS_Handler
9	SERCOM0 (Serial Communication Interface 0)	☐	3	SERCOM0_Handler
10	SERCOM1 (Serial Communication Interface 1)	☐	3	SERCOM1_Handler
11	SERCOM2 (Serial Communication Interface 2)	☐	3	SERCOM2_Handler
12	SERCOM3 (Serial Communication Interface 3)	☐	3	SERCOM3_Handler
13	SERCOM4 (Serial Communication Interface 4)	☐	3	SERCOM4_Handler
14	SERCOM5 (Serial Communication Interface 5)	☐	3	SERCOM5_Handler
15	TCC0 (Timer/Counter for Control Applications 0)	☐	3	TCC0_Handler
16	TCC1 (Timer/Counter for Control Applications 1)	☐	3	TCC1_Handler
17	TCC2 (Timer/Counter for Control Applications 2)	☐	3	TCC2_Handler
18	TC3 (Timer/Counter 3)	☒	3	TC3_TimerInterruptHandler
19	TC4 (Timer/Counter 4)	☒	3	TC4_TimerInterruptHandler
20	TC5 (Timer/Counter 5)	☐	3	TC5_Handler
23	ADC (Analog-to-Digital Converter)	☐	3	ADC_Handler
24	AC (Analog Comparators)	☐	3	AC_Handler
25	DAC (Digital-to-Analog Converter)	☐	3	DAC_Handler
26	PTC (Peripheral Touch Controller)	☐	3	PTC_Handler
27	I2S (Inter-IC Sound Controller)	☐	3	I2S_Handler

TC Interrupt

Interface Routines

void TC3_TimerInitialize(void);

void TC3_TimerStart(void);

void TC3_TimerStop(void);

uint32_t TC3_TimerFrequencyGet(void);

void TC3_Timer16bitPeriodSet(uint16_t period);

uint16_t TC3_Timer16bitPeriodGet(void);

uint16_t TC3_Timer16bitCounterGet(void);

void TC3_Timer16bitCounterSet(uint16_t count);

void TC3_TimerCallbackRegister(TC_Timer_CALLBACK callback, uintptr_t context);

One new function than polling mode.

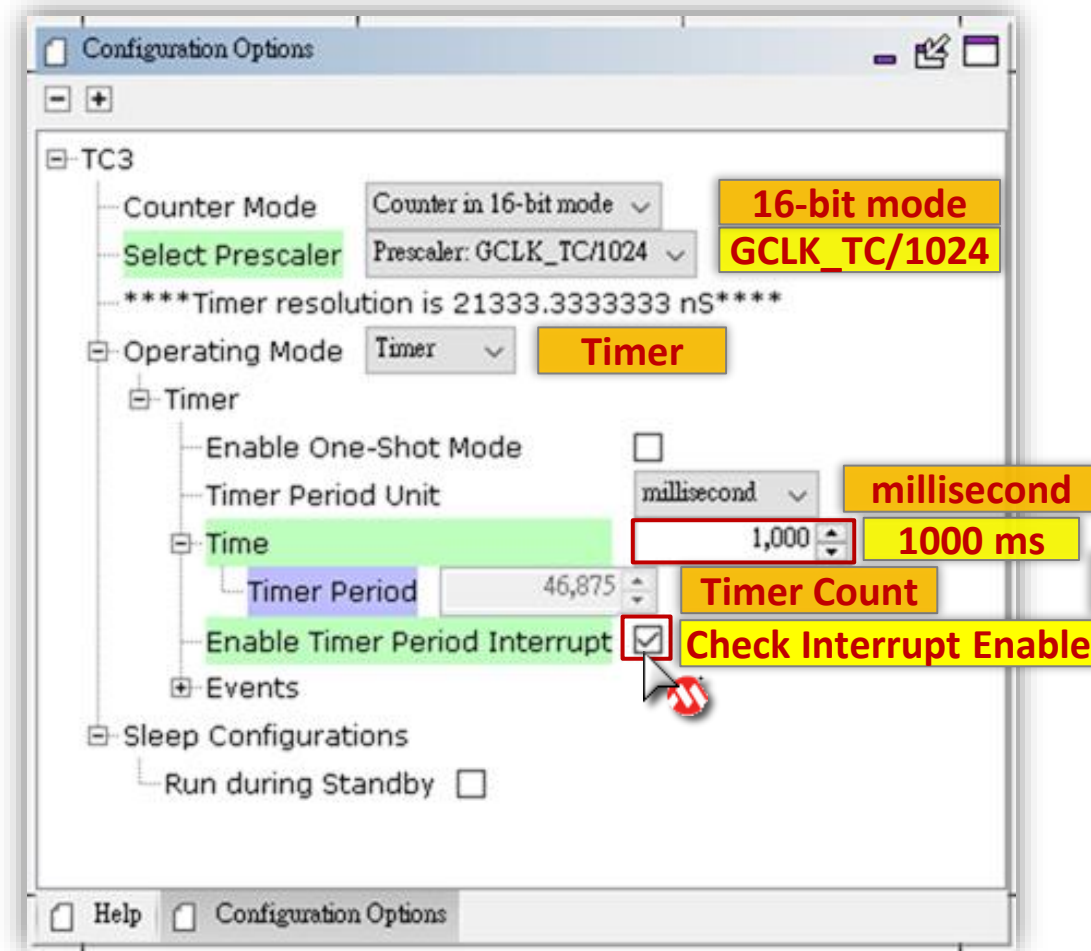
Lab5 TC Timer Method Interrupt Callback

- Try to configure TC3 to use **interrupt callback** method.
LED1 will toggle in **Timer Callback** while **Timer Period Expired** interrupt.
- Modify **TC3 Timer Period** to **1 second** in MH3 for easy observe the toggle.
- Modify code segment in **main()** and follow below sequence.
 - Firstly, remove Timer polling LED1 toggle from main loop.
 - Secondary, Create a TC3 Timer Callback function for LED1 toggle
 - Finally, hook(register) up TC3 Timer Callback function to TC3 Timer Interrupt event.
- **Let's go!**

Lab5 TC Timer Method Interrupt Callback

Step 1

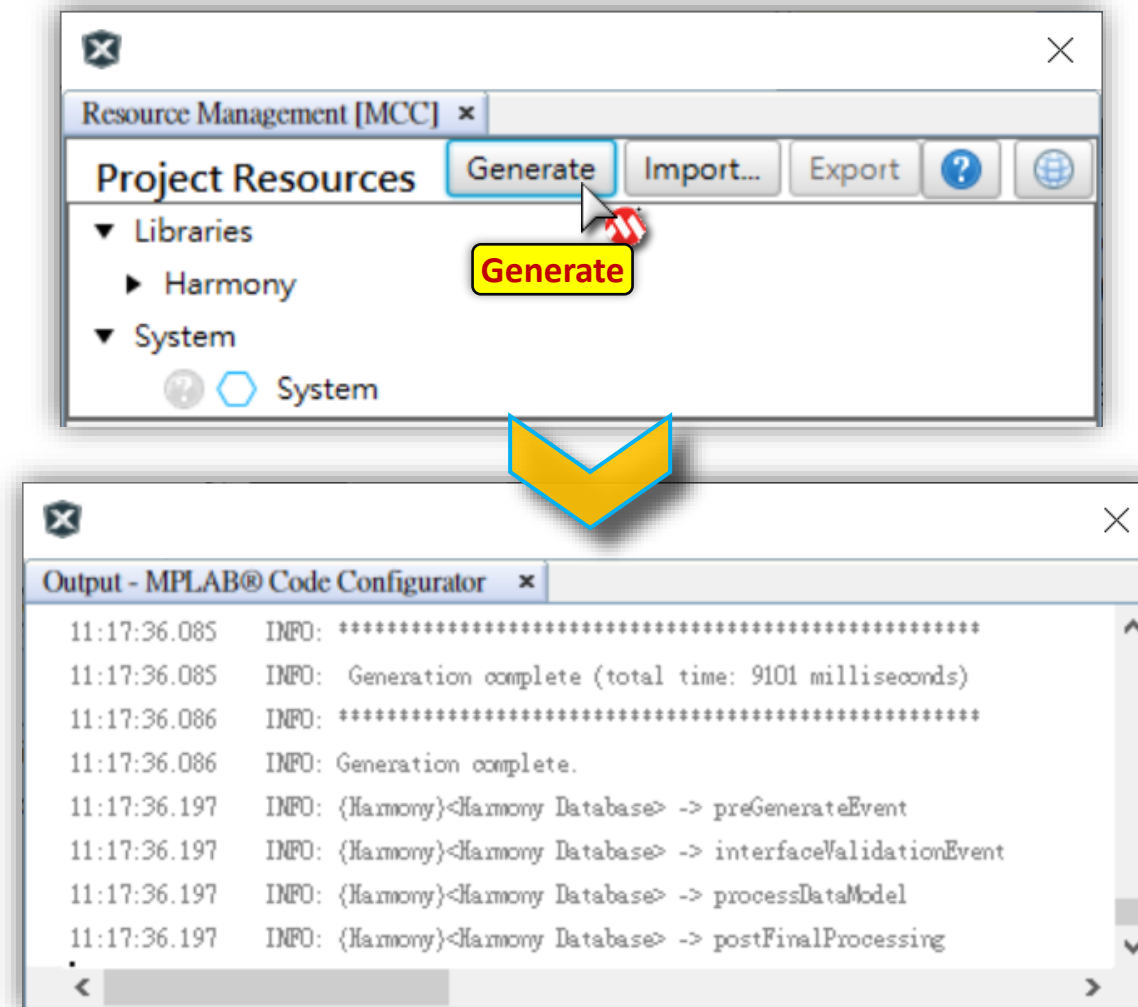
- Configure **TC3 Period to 1 second** and **Enable Period Interrupt** in configuration.



Lab5 TC Timer Method Interrupt Callback

Step 2

- Click "**Generate**" Button to Generate your Code.



Lab5 TC Timer Method Interrupt Callback

Step 3

a Add code segment to your main loop

```
// TODO 5.01
void TC3_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if( status & TC_INTFLAG_OVF_Msk )
        LED1_Toggle();
}

int main (void)
{
    SYS_Initialize ( NULL );
    ...
    // TODO 5.02
    TC3_TimerCallbackRegister( TC3_TimerExpired, (uintptr_t )NULL );
    TC3_TimerStart();

    while( true )
    { ...
    }
}
```

b Program firmware to target board then observe result.

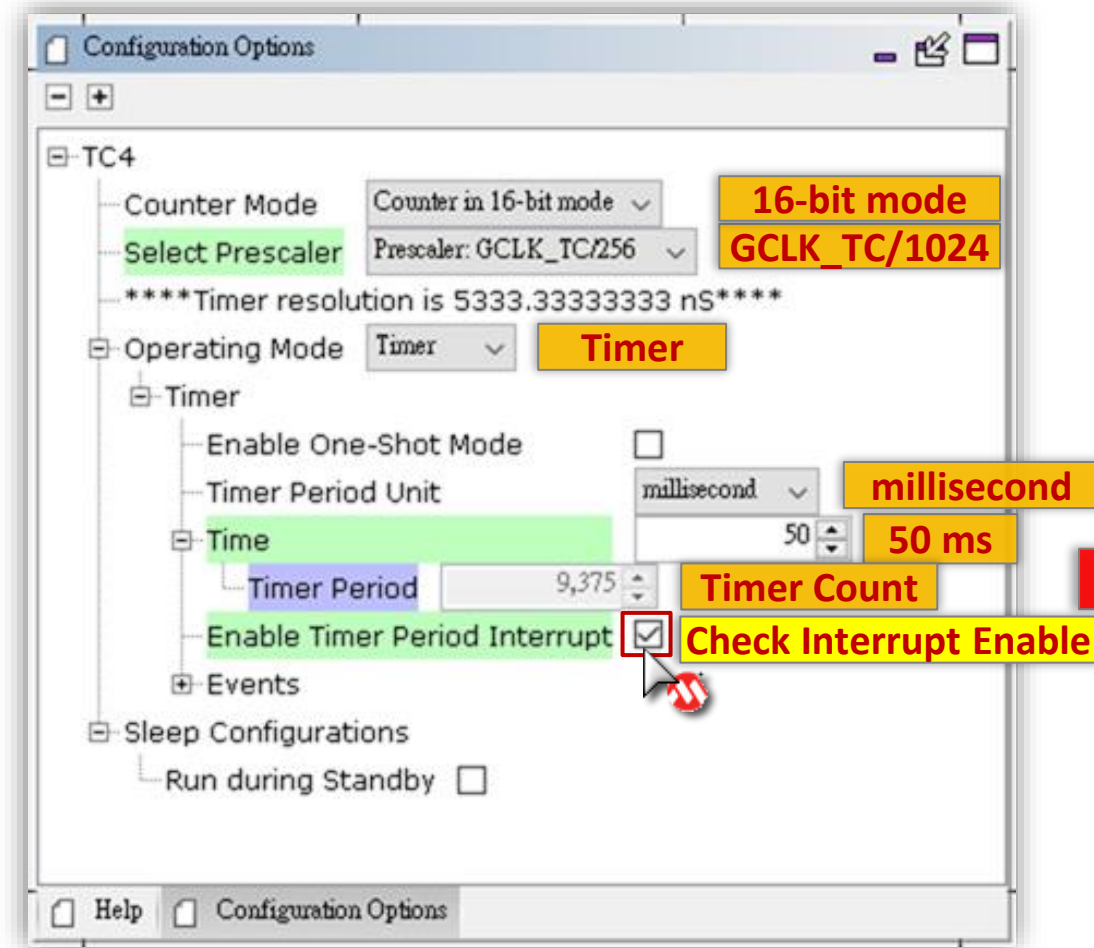
Lab6 TCs Timer Method Interrupt Callback

- Try to configure TC4 to use **interrupt callback** method.
LED2 will toggle in **Timer Callback** while **Timer Period Expired** interrupt.
- Keep **TC4 Timer Period** as **50ms** in MH3 that we did in previous Lab.
- **Modify code segment in main() and follow below sequence.**
 - Firstly, remove Timer polling LED2 toggle from main loop.
 - Secondary, Create a TC4 Timer Callback function for LED2 toggle
 - Finally, hook(register) up TC4 Timer Callback function to TC4 Timer Interrupt event.
- **Let's go!**

Lab6 TCs Timer Method Interrupt Callback

Step 1

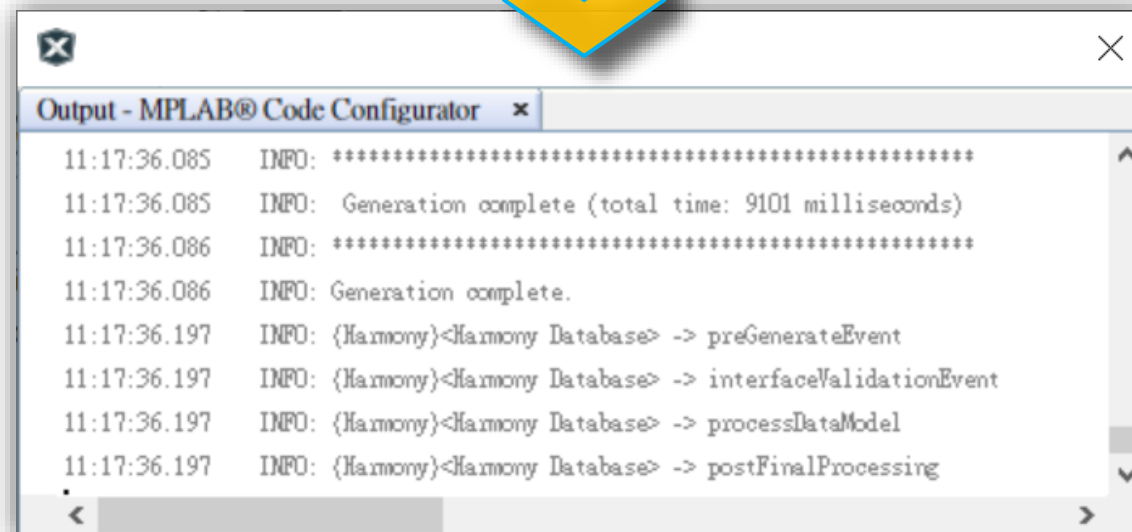
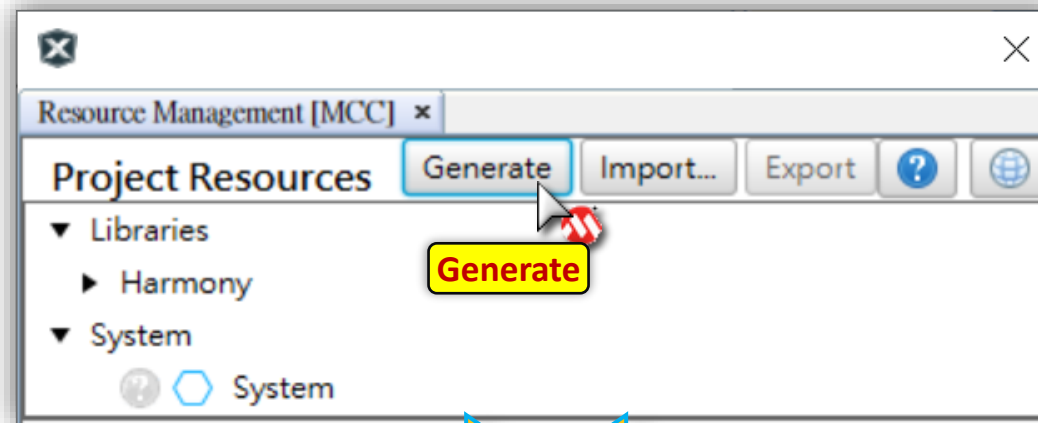
- **Enable Period Interrupt** of TC4 in configuration windows.



Lab6 TCs Timer Method Interrupt Callback

Step 2

- Click "**Generate**" Button to Generate your Code.



Lab6 TCs Timer Method Interrupt Callback

Step 3

a Add code segment to your main loop

```
// TODO 6.01
void TC4_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if( status & TC_INTFLAG_OVF_Msk )
        LED2_Toggle();
}

int main (void)
{
    SYS_Initialize ( NULL );
    ...
    // TODO 6.02
    TC4_TimerCallbackRegister( TC4_TimerExpired, (uintptr_t )NULL );
    TC4_TimerStart();

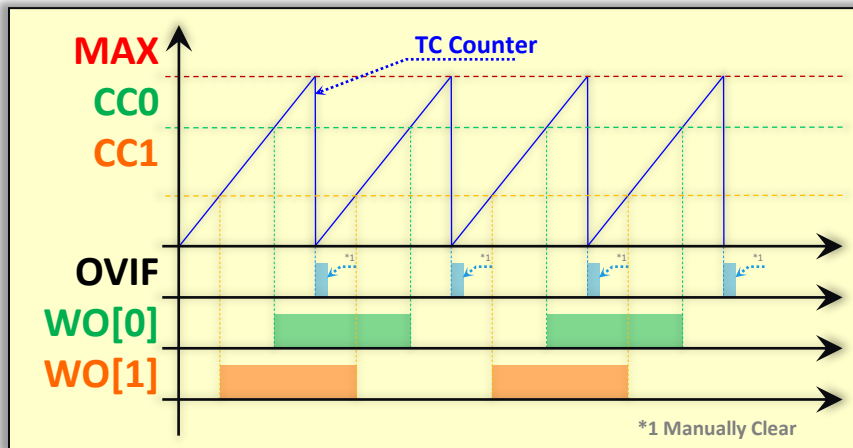
    while( true )
    { ...
    }
}
```

b Program firmware to target board then observe result.

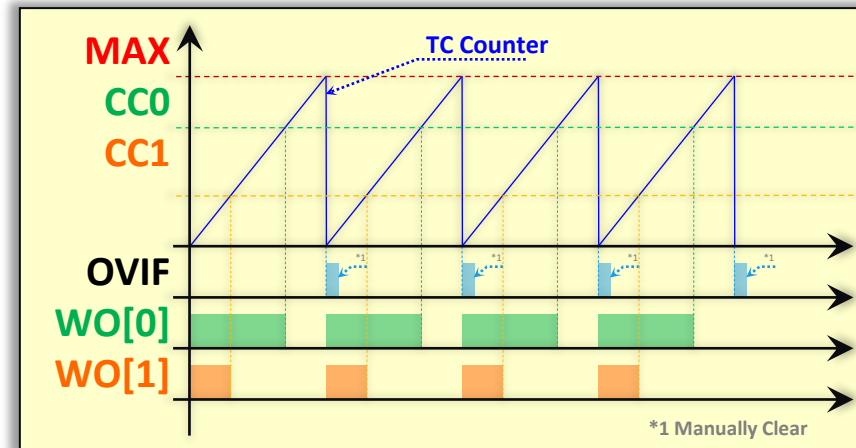
Study Advanced Features

The main course of this section will stop here, If you want to know more about TC/TCC Timer, you can refer to the SAM2001ADV course..

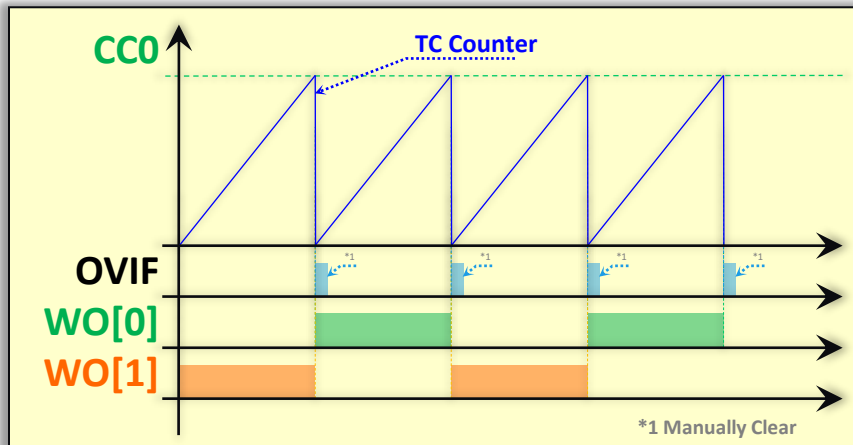
NFRQ, 16Bit, Up Count



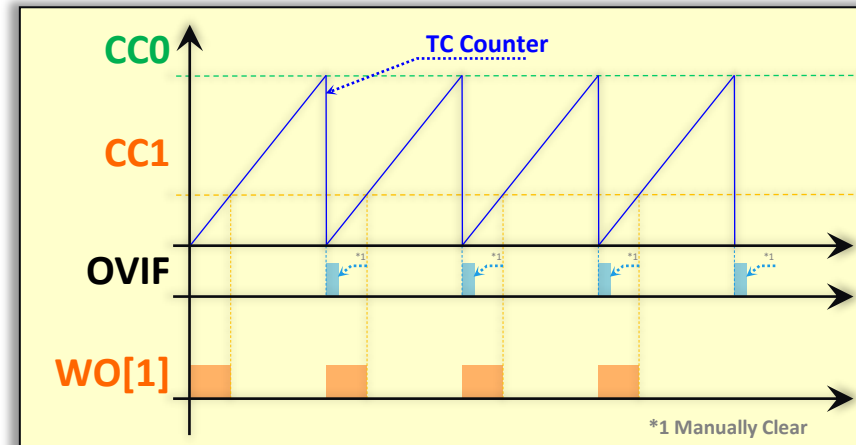
NPWM, 16Bit, Up Count



MFRQ, 16Bit, Up Count



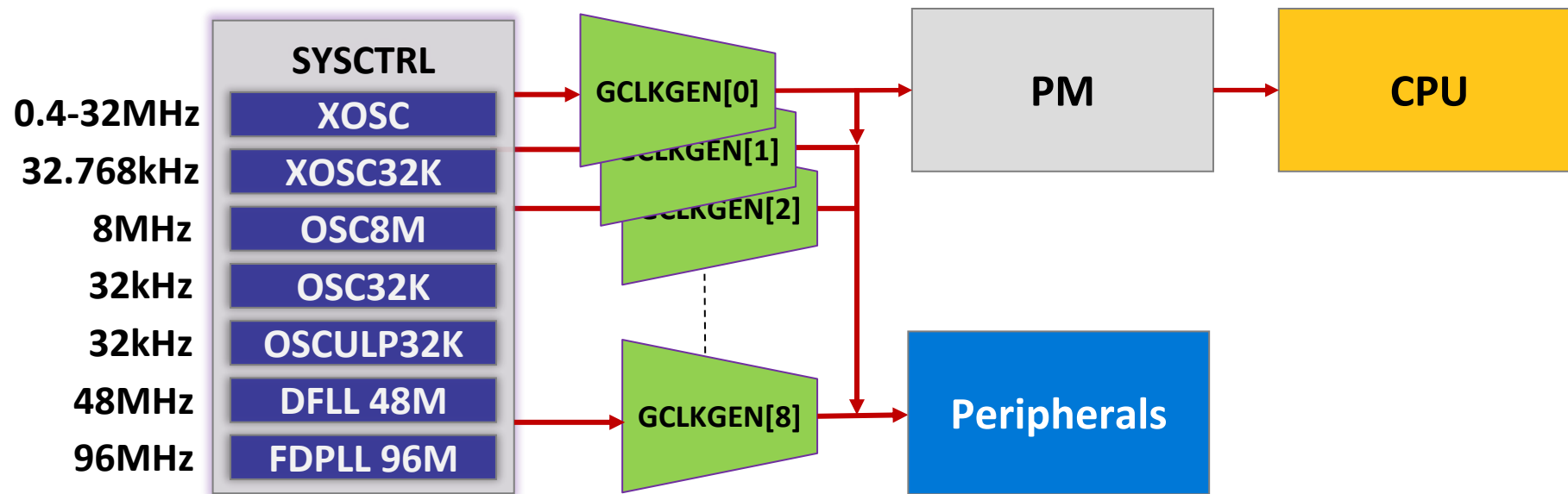
MPWM, 16Bit, Up Count



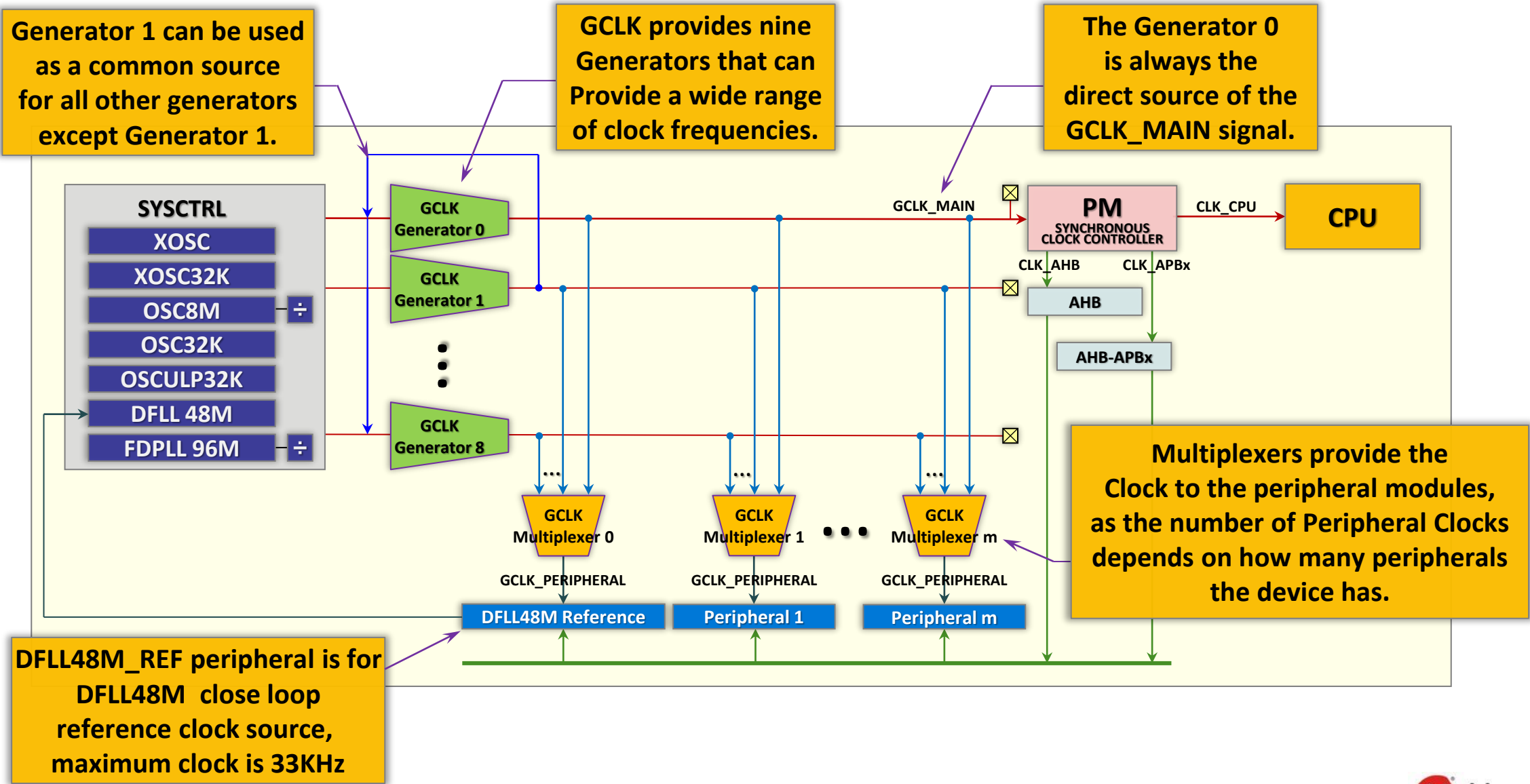
Clock System Architecture

Clock System

- Review previously section, SAMD21's clock system provides clock sources to the Generic Clock Controller ◦ The available clock sources are XOSC, XOSC32K, OSC32K, OSCULP32K, OSC8M,DFLL48M and FDPLL96M ◦
- The bus clock can be enabled and disabled in the Power Manager(PM) include CPU Clock(GCLK_MAIN) ◦

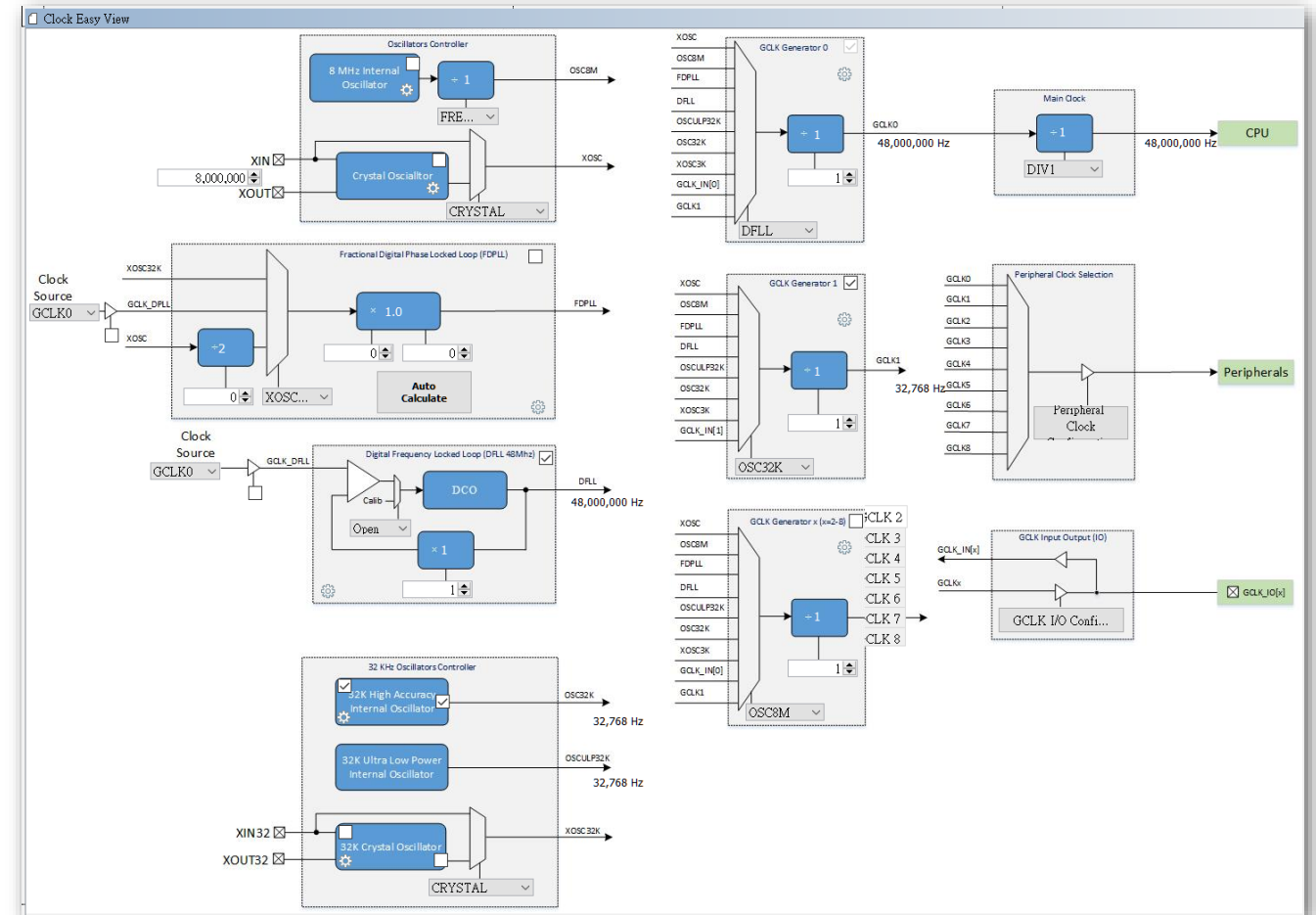
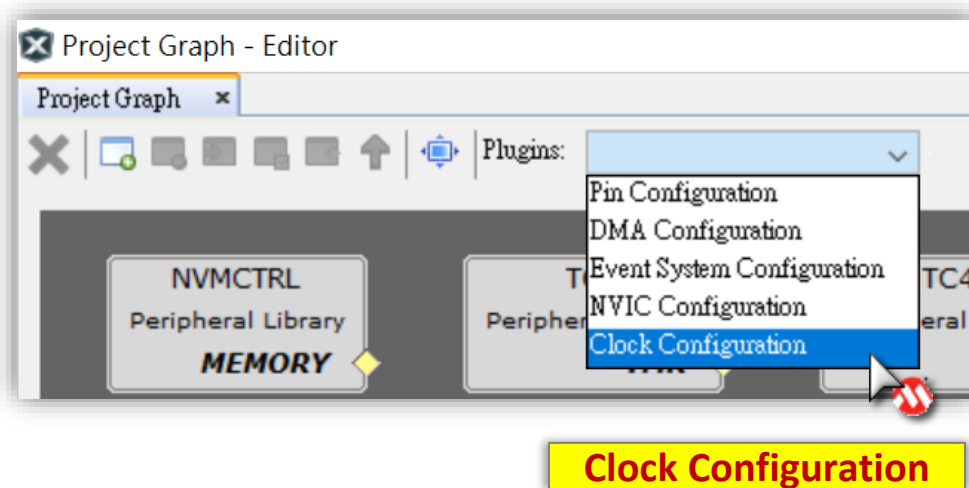


Clock System Block Diagram



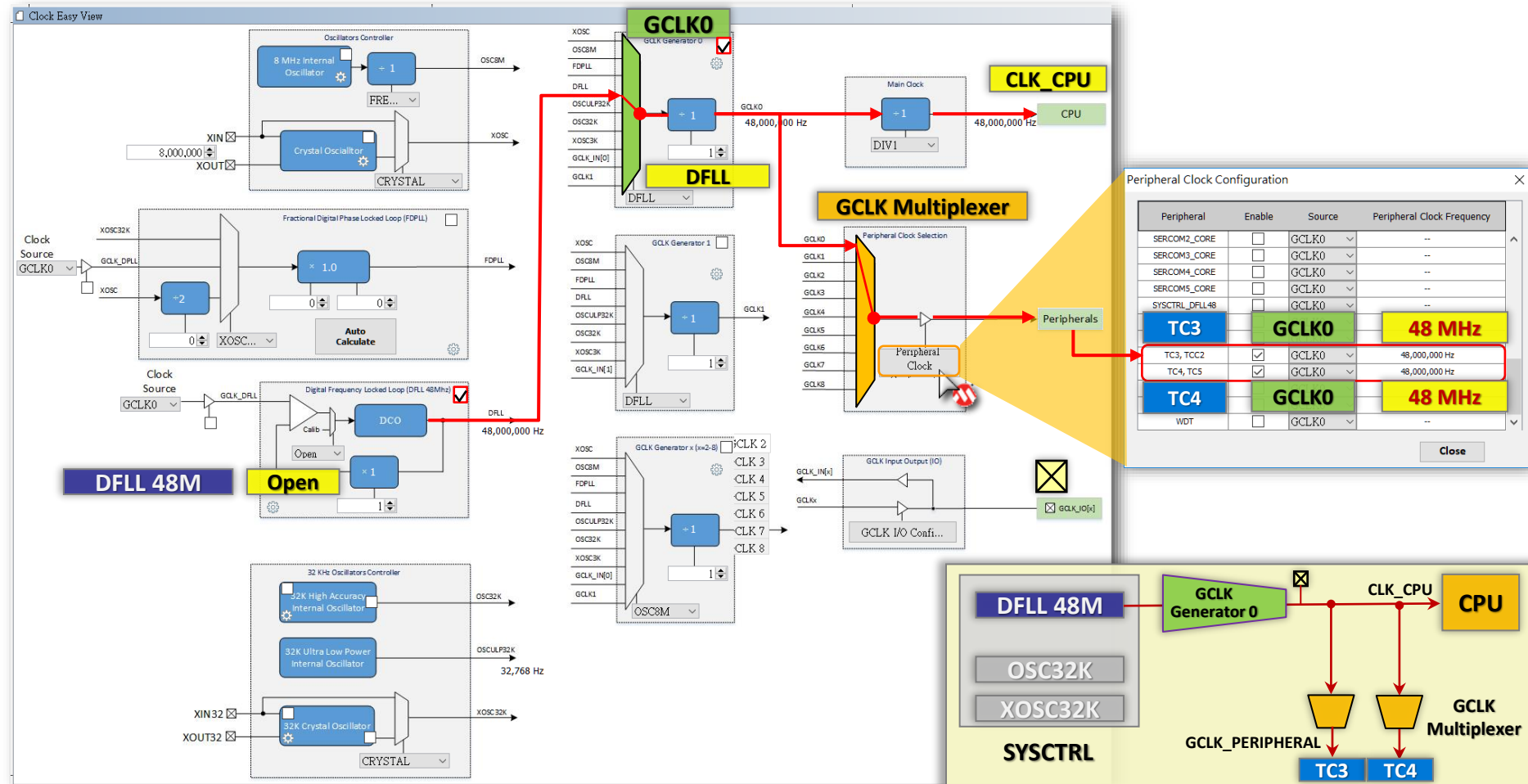
Clock Setting Use MCC Harmony

- Open Clock Easy View windows in **Project Graph > Plugins > Clock Configuration**.



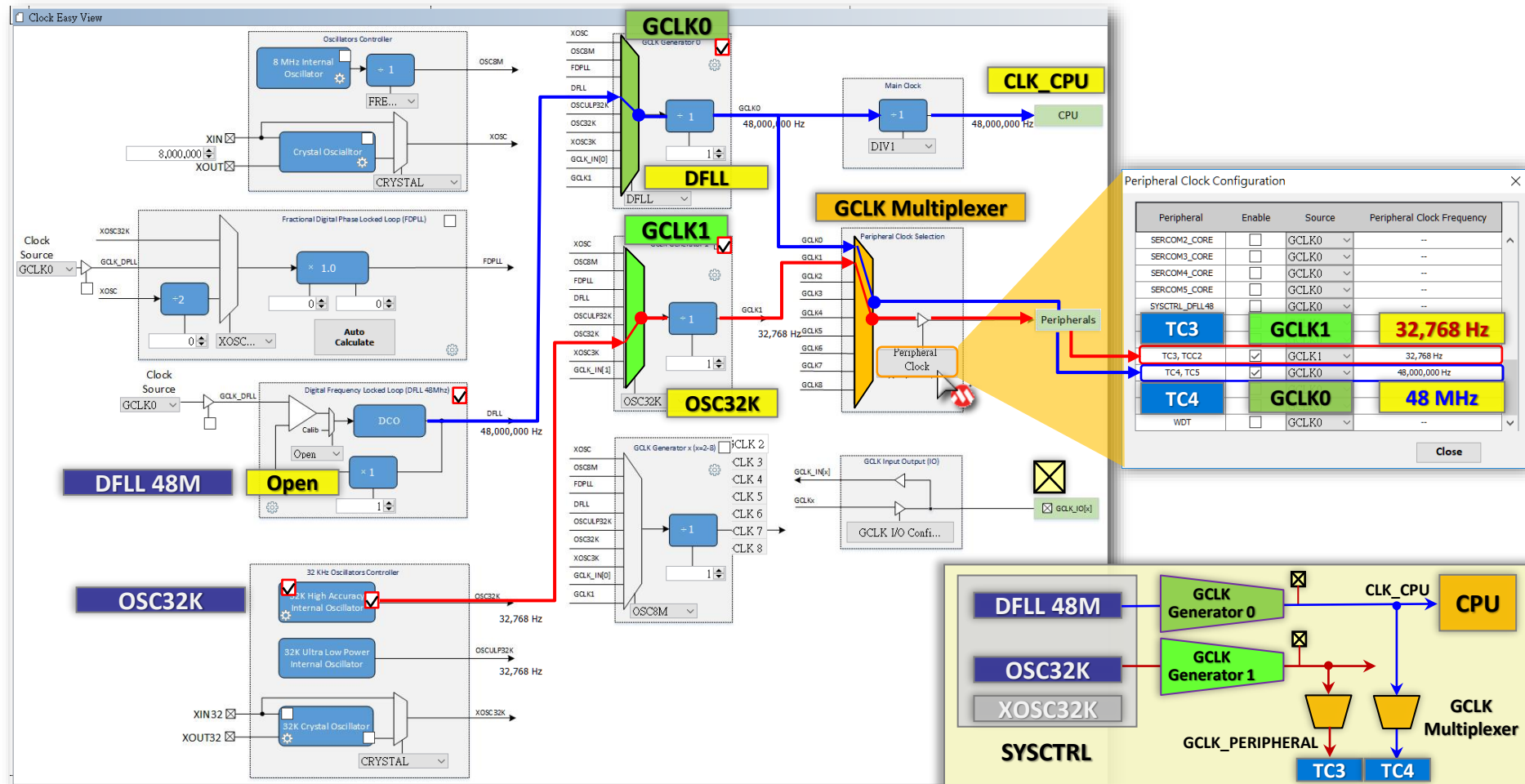
Clock Setting Use MCC Harmony

- Default clock setting of new project (DFLL48M open loop).



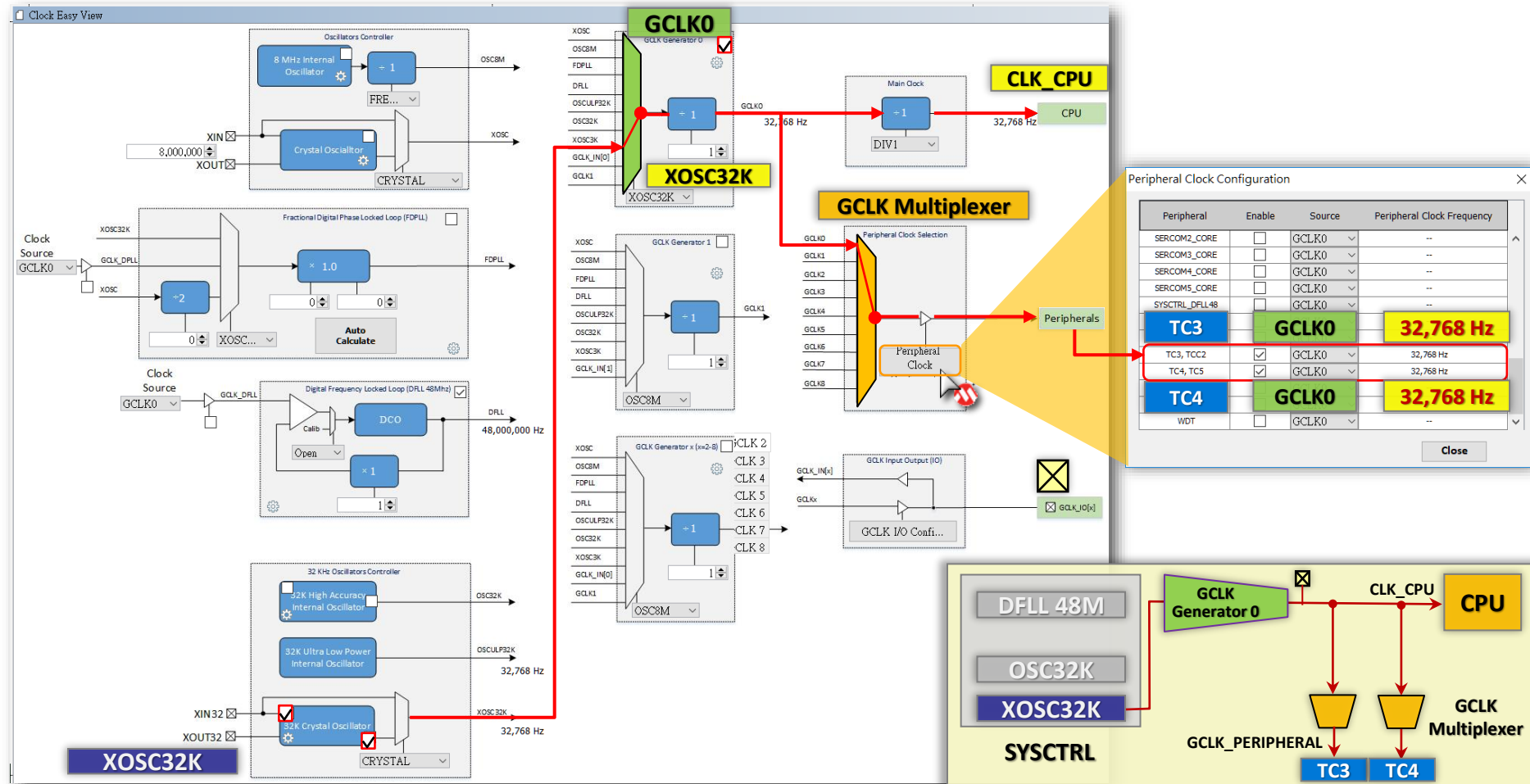
Clock Setting Use MCC Harmony

- For Example : Internal 32k RC -> GCLK Generator 1 -> TC3



Clock Setting Use MCC Harmony

- For Example : External 32.768K -> Generator 0 -> CPU & TC3



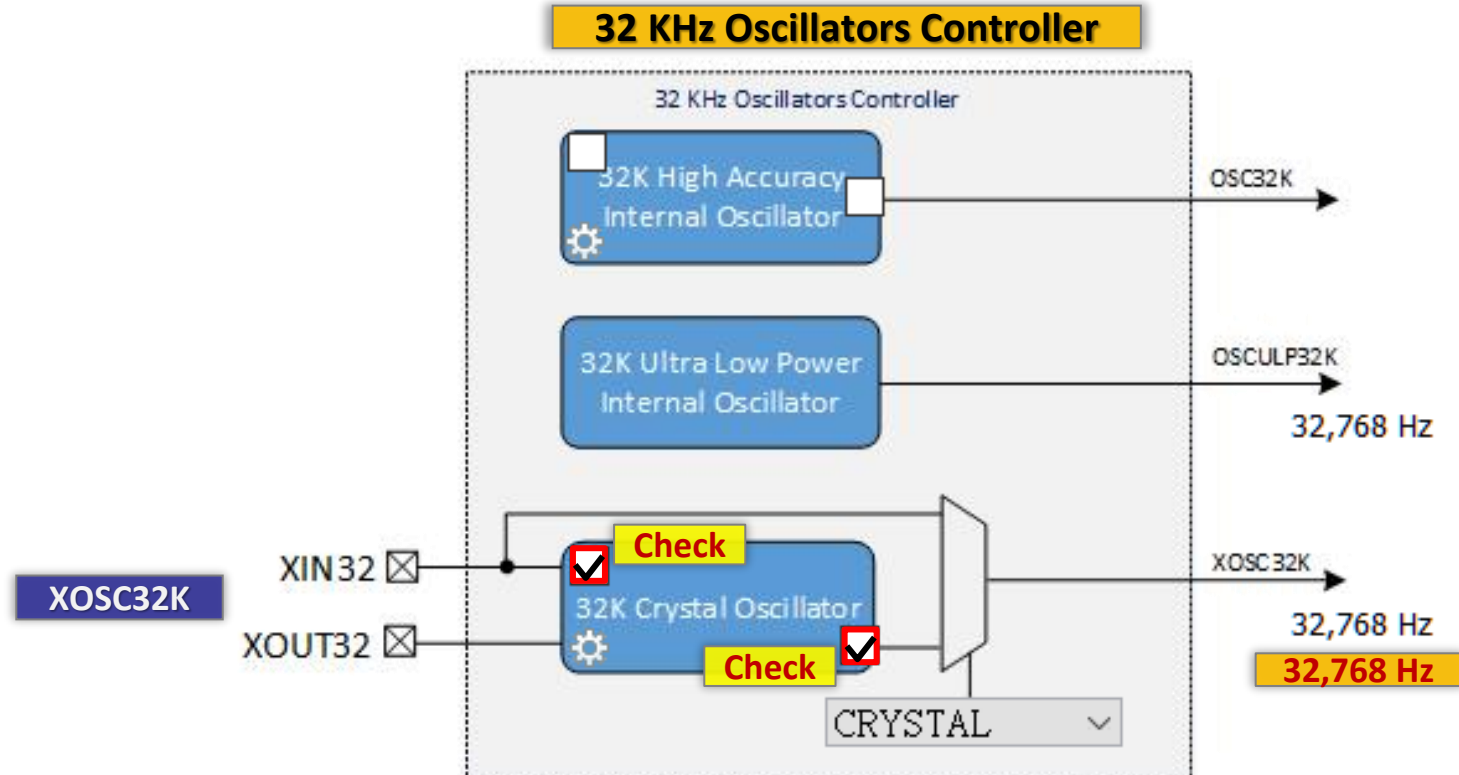
Lab7 System Clock XOSC32K

- Try to change main clock from default source to external 32.768K.
- Setting XOSC32K then enable it, first.
- Connect XOSC32K to Generator 0.
- Connect TC3, TC4 to Generator 0.

• **Let's go!**

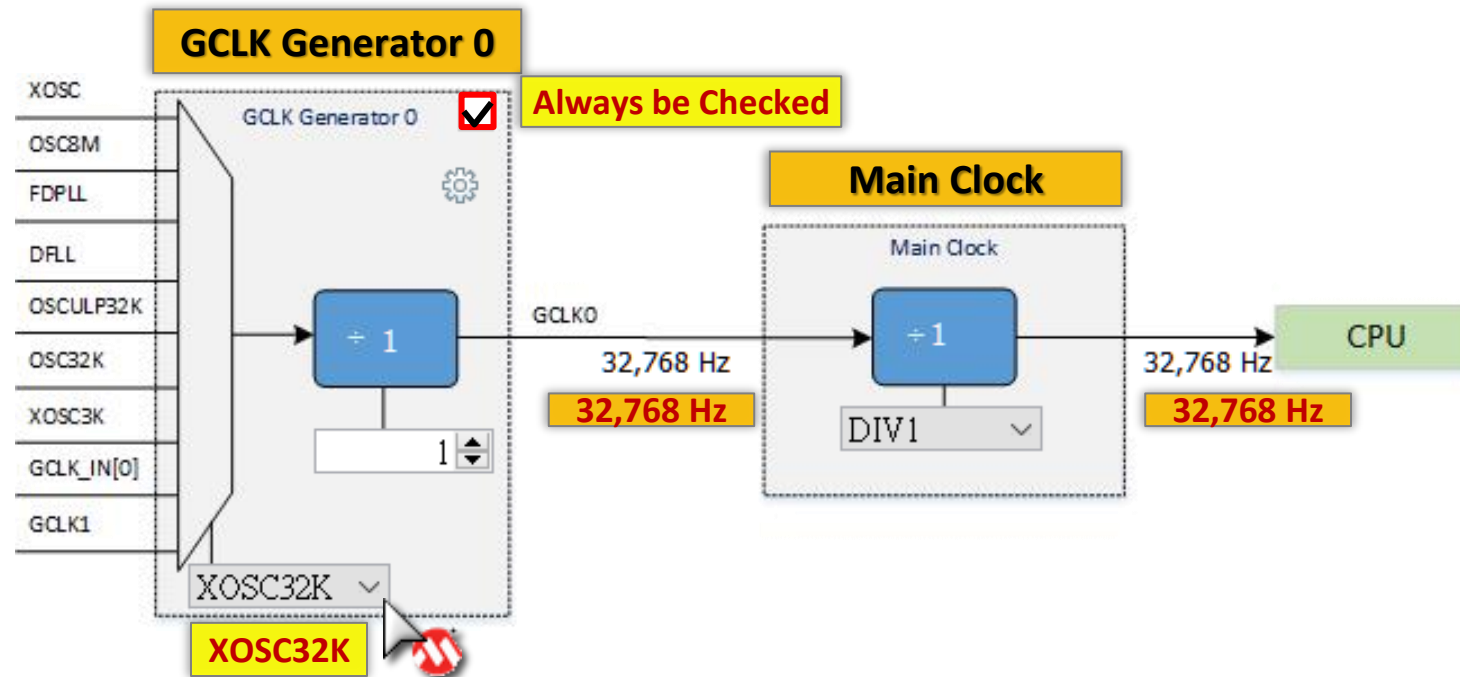
Lab7 System Clock XOSC32K

Step 1



Lab7 System Clock XOSC32K

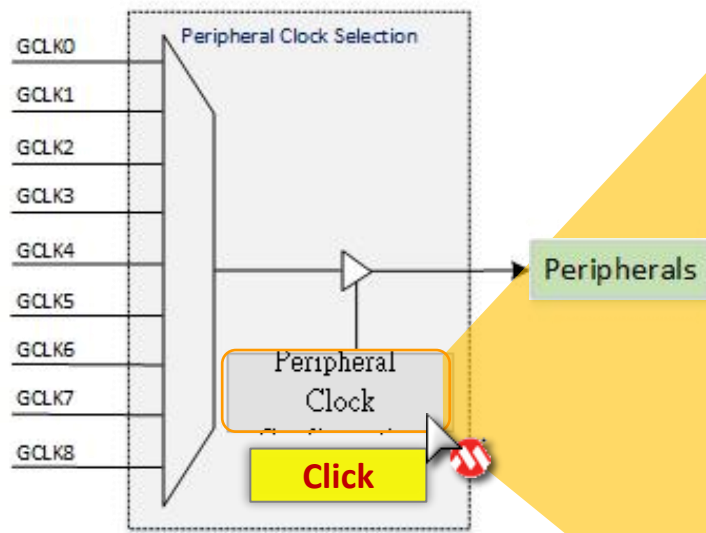
Step 2



Lab7 System Clock XOSC32K

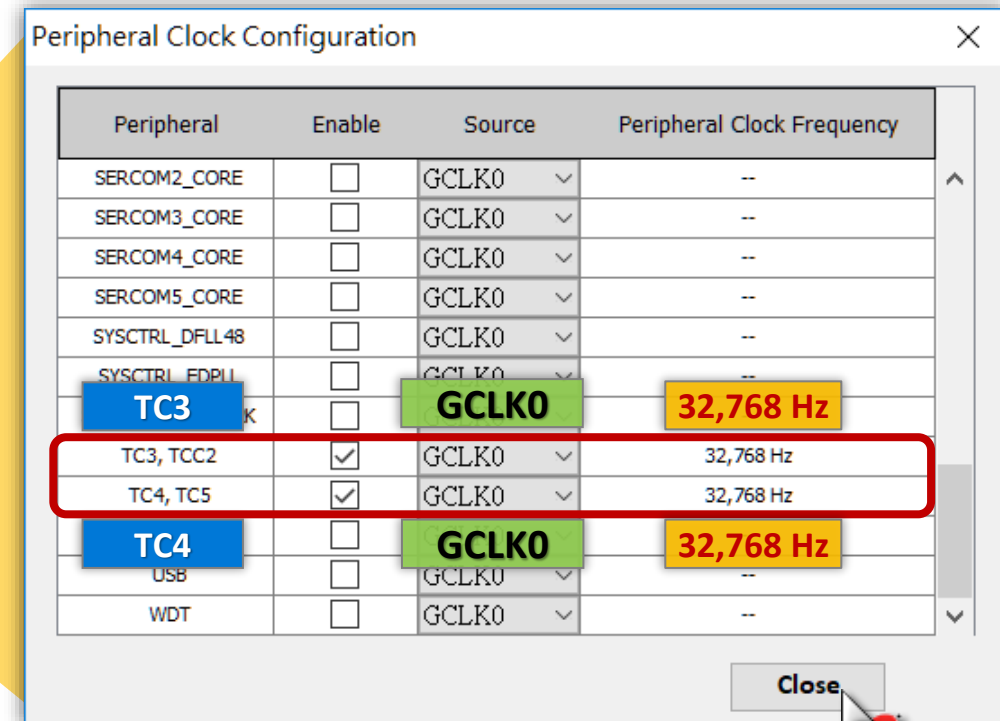
Step 3

Peripheral Clock Selection



The diagram shows a 'Peripheral Clock Selection' block with inputs GCLK0 through GCLK8. A 'Peripheral Clock' button is highlighted with a yellow 'Click' label and a mouse cursor. An arrow points from this button to a 'Peripherals' block.

Peripheral Clock Configuration



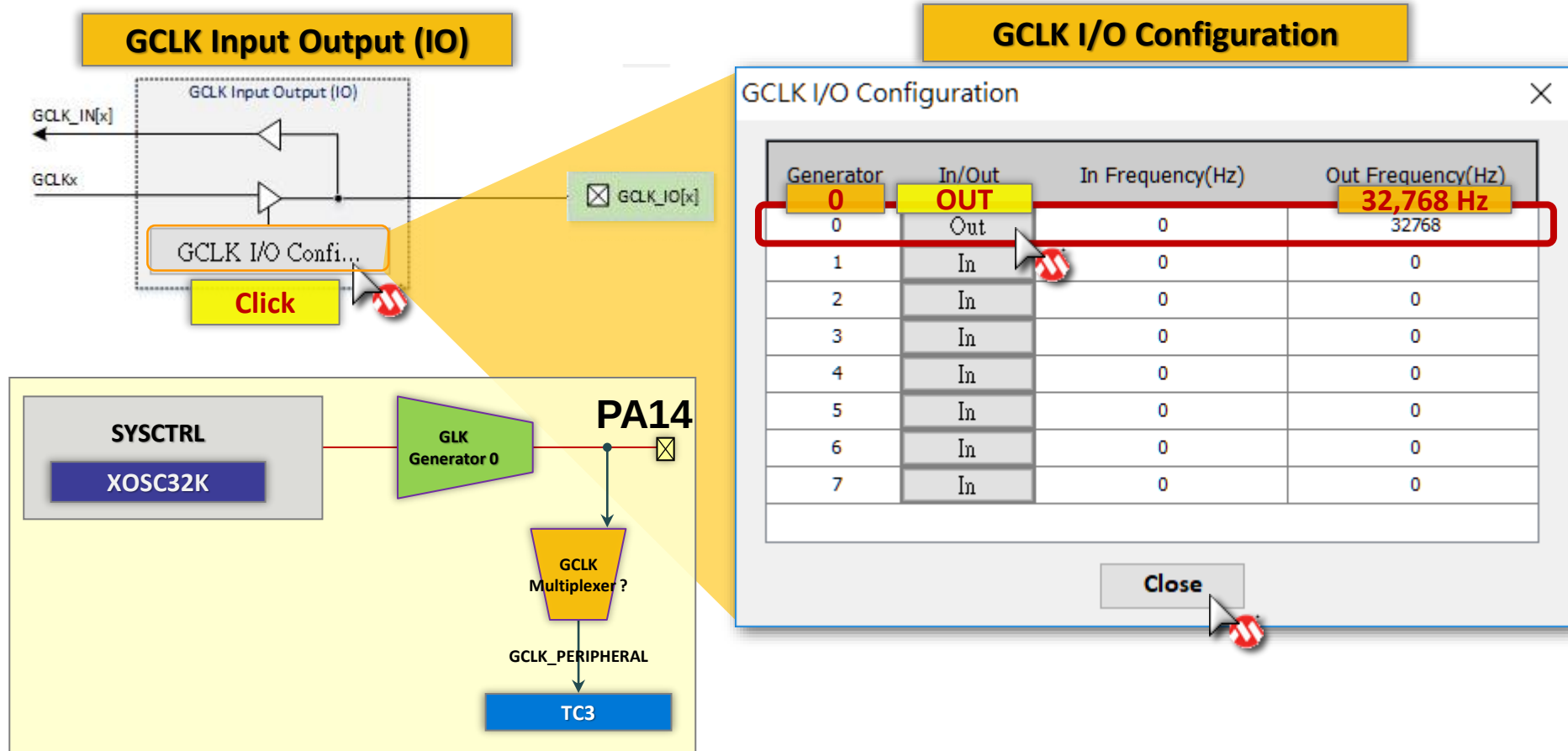
The 'Peripheral Clock Configuration' window displays a table of peripheral clock settings. A red box highlights the configuration for TC3, TCC2 and TC4, TC5, both set to GCLK0 at 32,768 Hz. A blue box highlights 'TC3' and another blue box highlights 'TC4'. A green box highlights 'GCLK0' and an orange box highlights '32,768 Hz' for both entries. A 'Close' button is at the bottom right.

Peripheral	Enable	Source	Peripheral Clock Frequency
SERCOM2_CORE	☐	GCLK0	--
SERCOM3_CORE	☐	GCLK0	--
SERCOM4_CORE	☐	GCLK0	--
SERCOM5_CORE	☐	GCLK0	--
SYSCTRL_DFLL48	☐	GCLK0	--
SYSCTRL_FDPLL	☐	GCLK0	--
TC3	☐	GCLK0	32,768 Hz
TC3, TCC2	☒	GCLK0	32,768 Hz
TC4, TC5	☒	GCLK0	32,768 Hz
TC4	☐	GCLK0	32,768 Hz
USB	☐	GCLK0	--
WDT	☐	GCLK0	--

Lab7 System Clock XOSC32K

Step 4

- Configure GCLK Generator 0 to output clock.



Lab7 System Clock XOSC32K

Step 5

- Please **disable BT1(PA14) GPIO** pin function firstly and then reassign **PA14** as **GCLK_IO0** output pin.

Pin Table

Package: TQFP48

Module	Function	PA07	PA08	PA09	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	GCLK_I	BT3	PA16	LED3	PA18	PA19	LED1
		12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
	EIC_NMI																		
GCLK_IO0	GCLK_IO0																		
	GCLK_IO1																		
	GCLK_IO2																		
	GCLK_IO3																		
GCLK	GCLK_IO4																		
	GCLK_IO5																		
	GCLK_IO6																		
GPIO	GCLK_IO7																		
GPIO	GPIO																		
	I2S_FS0																		

PA14 (GCLK_IO0)

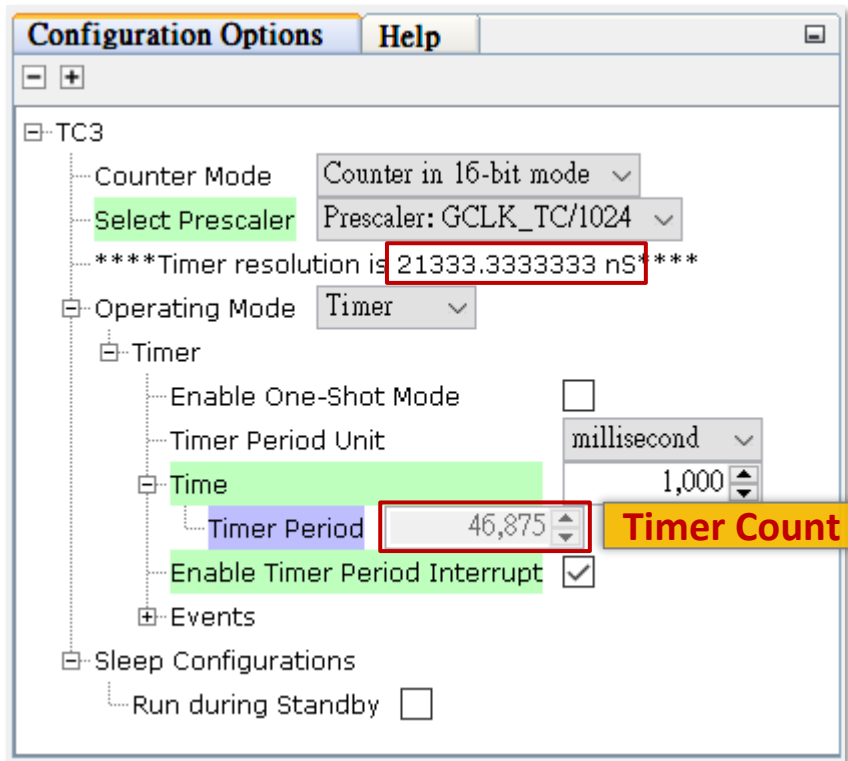
Click Enable

Click Disable

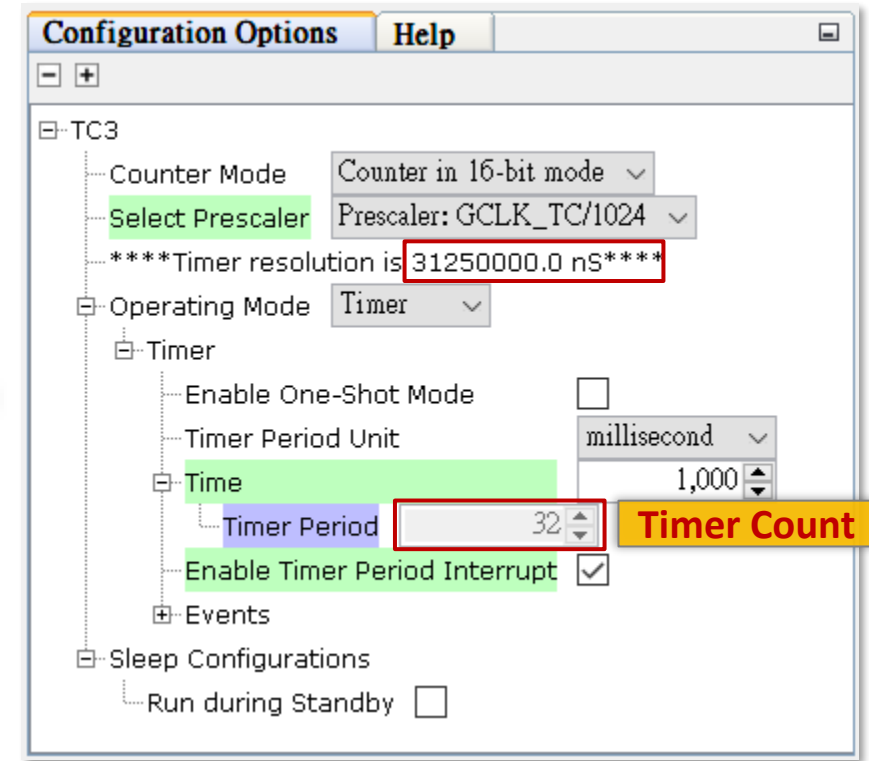
Lab7 System Clock XOSC32K

What's changed if to change main clock from 48MHz to 32.768KHz ?

- Let's check **TC3** configuration to see the difference after change the clock source.



Clock Source : DFL48M

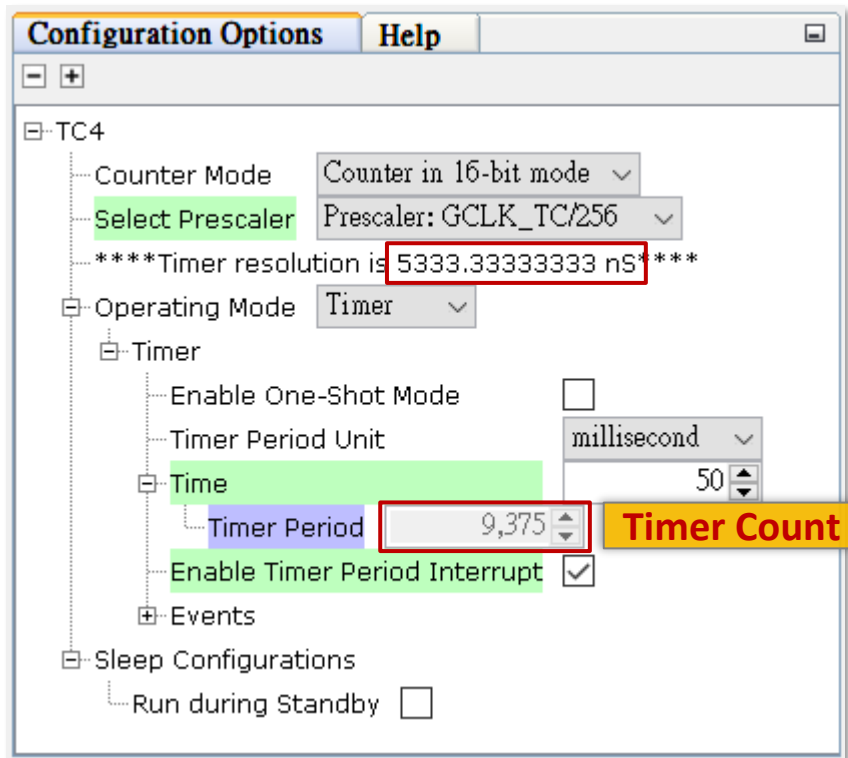


Clock Source : XOSC32K

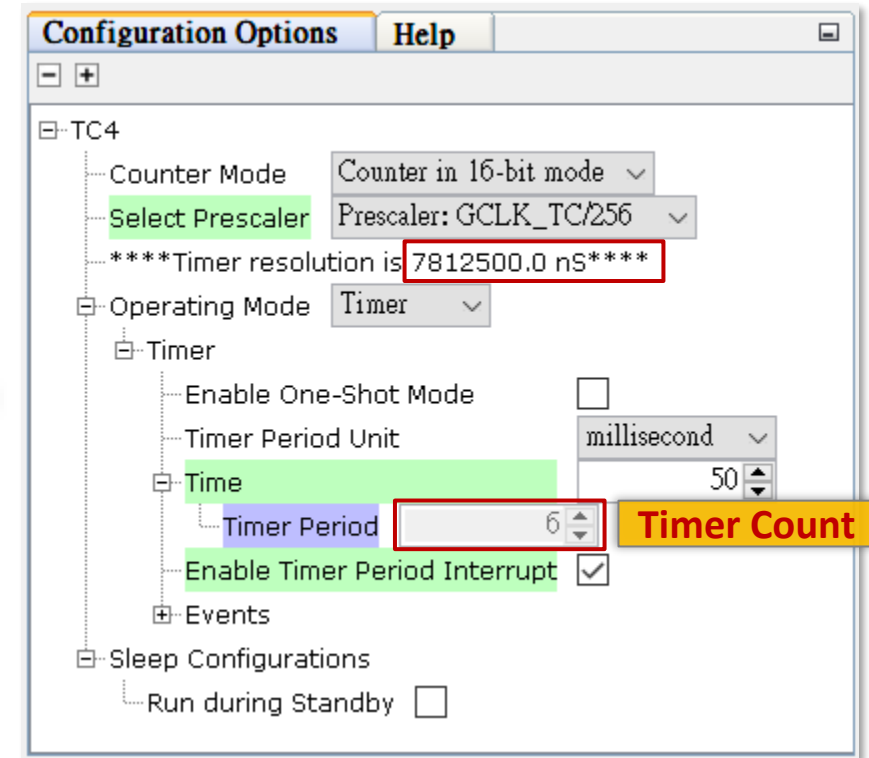
Lab7 System Clock XOSC32K

What's changed if to change main clock from 48MHz to 32.768KHz ?

- Let's check **TC4** configuration to see the difference after change the clock source.



Clock Source : DFL48M

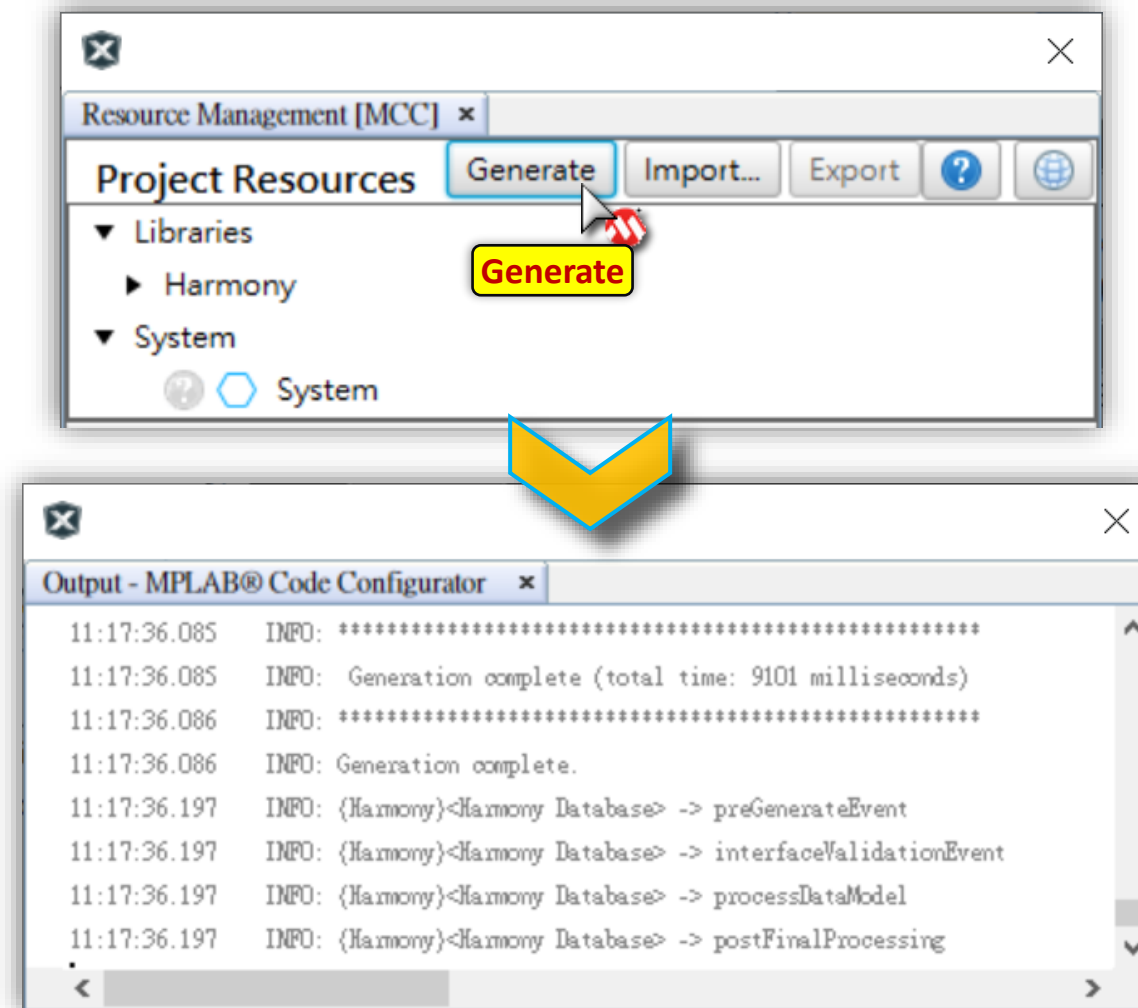


Clock Source : XOSC32K

Lab7 System Clock XOSC32K

Step 6

- Click "**Generate**" Button to Generate your Code.



Lab7 System Clock XOSC32K

Step 7

a Comment out BT2(PA14) related GPIO code segment.

```
...

int main (void)
{
    SYS_Initialize ( NULL );

    while( true )
    {
        ...

        // TODO 7.01
        // if ( BT1_Get() ) LED3_Clear();
        // else             LED3_Set();
    }
}
```

Flash Access Time and Wait State

- The SAMD21 main flash Random Read time around 40nS. That means **you must adjust Wait State to proper value if main clock over than 24MHz**. The actual value shows in below, it will depend on system supply voltage.

V _{DD} range	NVM Wait States	Maximum Operating Frequency	Units
1.62V to 2.7V	0	14	MHz
	1	28	
	2	42	
	3	48	
2.7V to 3.63V	0	24	
	1	48	

VDD = 3.3V

CPU_CLK = 32.768KHz

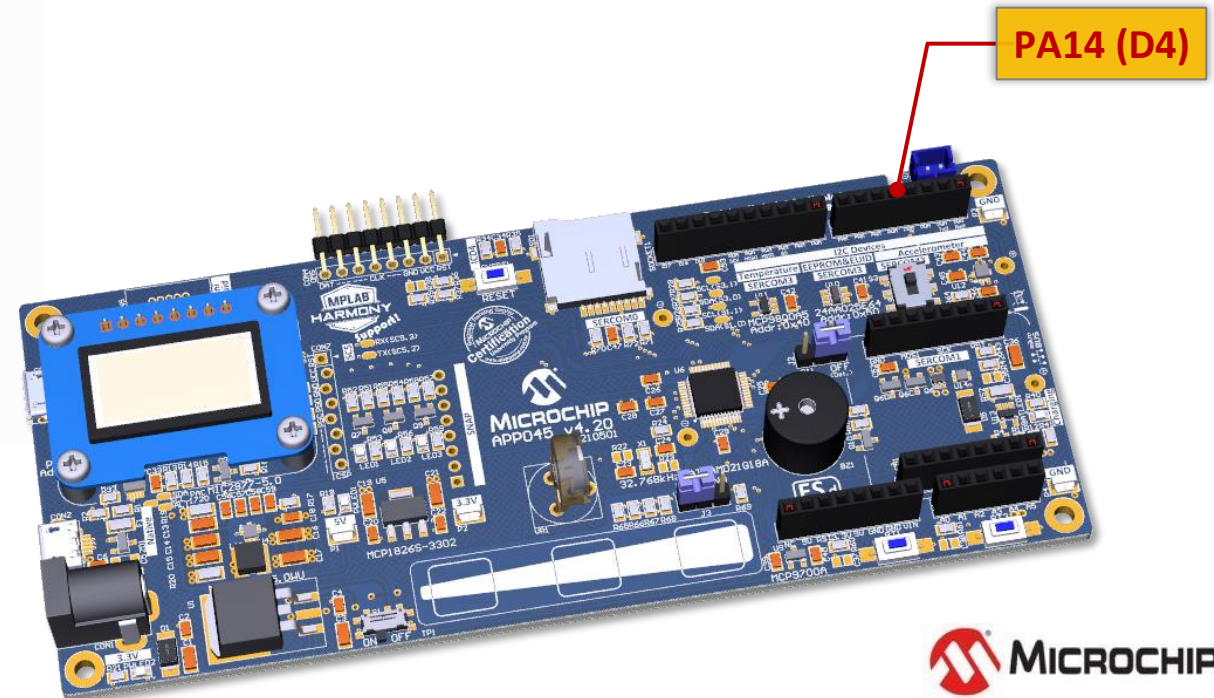
MCC Harmony will auto adjust proper
NVM Wait State.
(In case of 3.3V voltage supply only).

```
plib_nvmctrl.c
void NVMCTRL_Initialize(void)
{
    NVMCTRL_REGS->NVMCTRL_CTRLB =
        NVMCTRL_CTRLB_READMODE_NO_MISS_PENALTY |
        NVMCTRL_CTRLB_SLEEPPRM_WAKEONACCESS |
        NVMCTRL_CTRLB_RWS(0) |
        NVMCTRL_CTRLB_MANW_Msk;
}
```

Lab7 System Clock XOSC32K

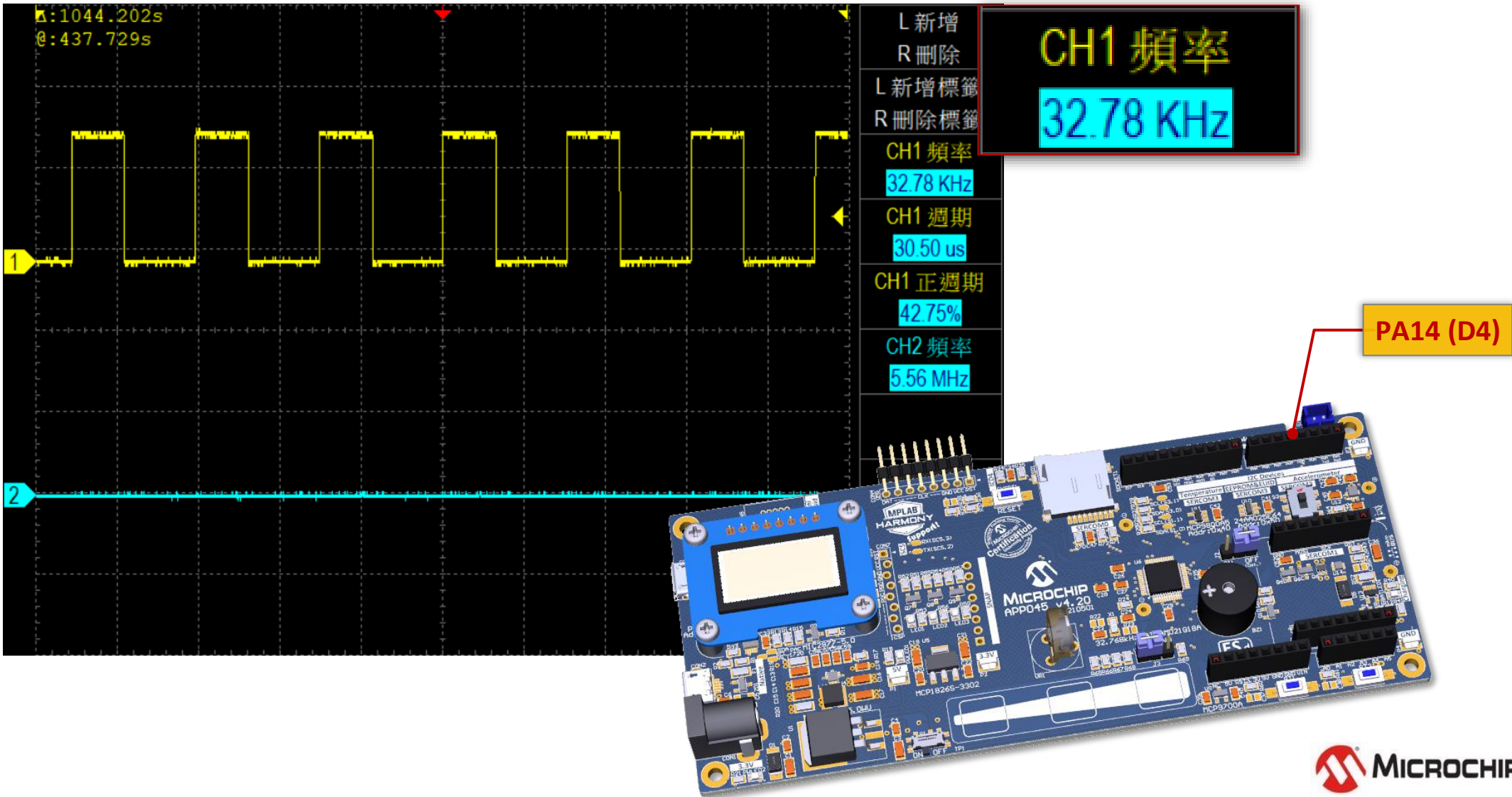
Clock Output

Arduino UNO Socket		
PA11	7	D0/RX
PA10	8	D1/TX
PA08	9	D2
PA09	10	D3/PWM
PA14	11	D4
PA15	12	D5/PWM
PA20	13	D6/PWM
PA21	14	D7



Lab7 System Clock XOSC32K

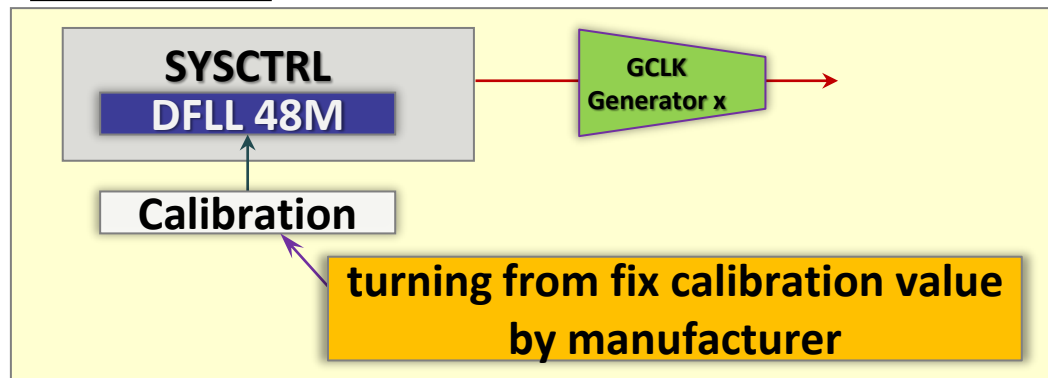
Clock Output



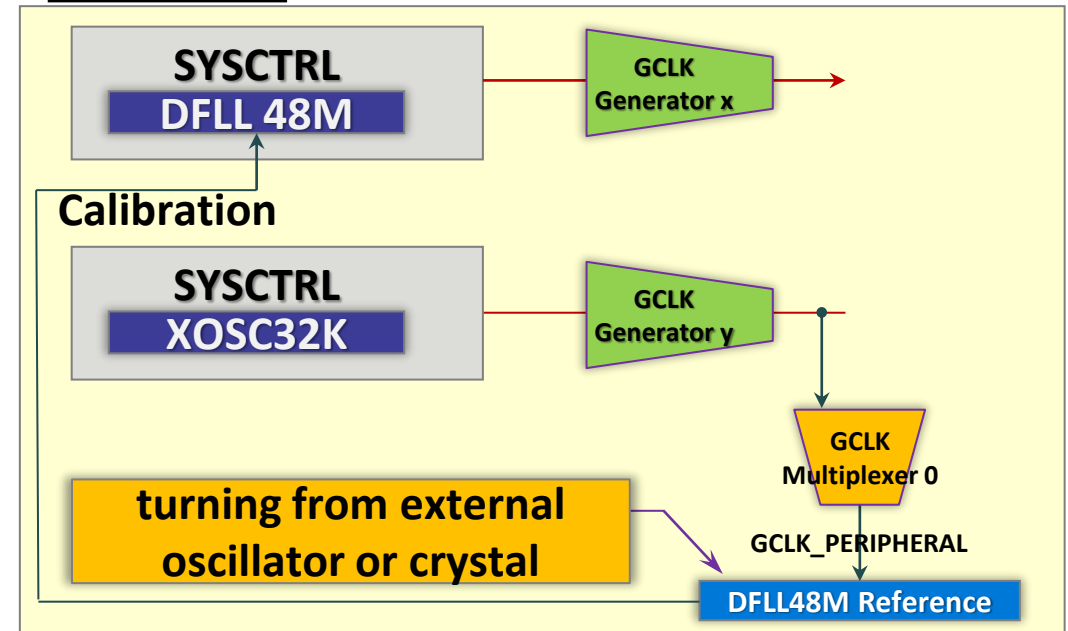
Clock source from DFLL48M

- There are 2 different mode for DFLL48M clock source
 - Open Loop : **Project Default Use**
Clock turning from fix calibration value by manufacturer.
 - Close Loop :
Clock turning from external oscillator or crystal.

Open Loop



Close Loop



Lab8 System Clock DFLL48M

Close Loop

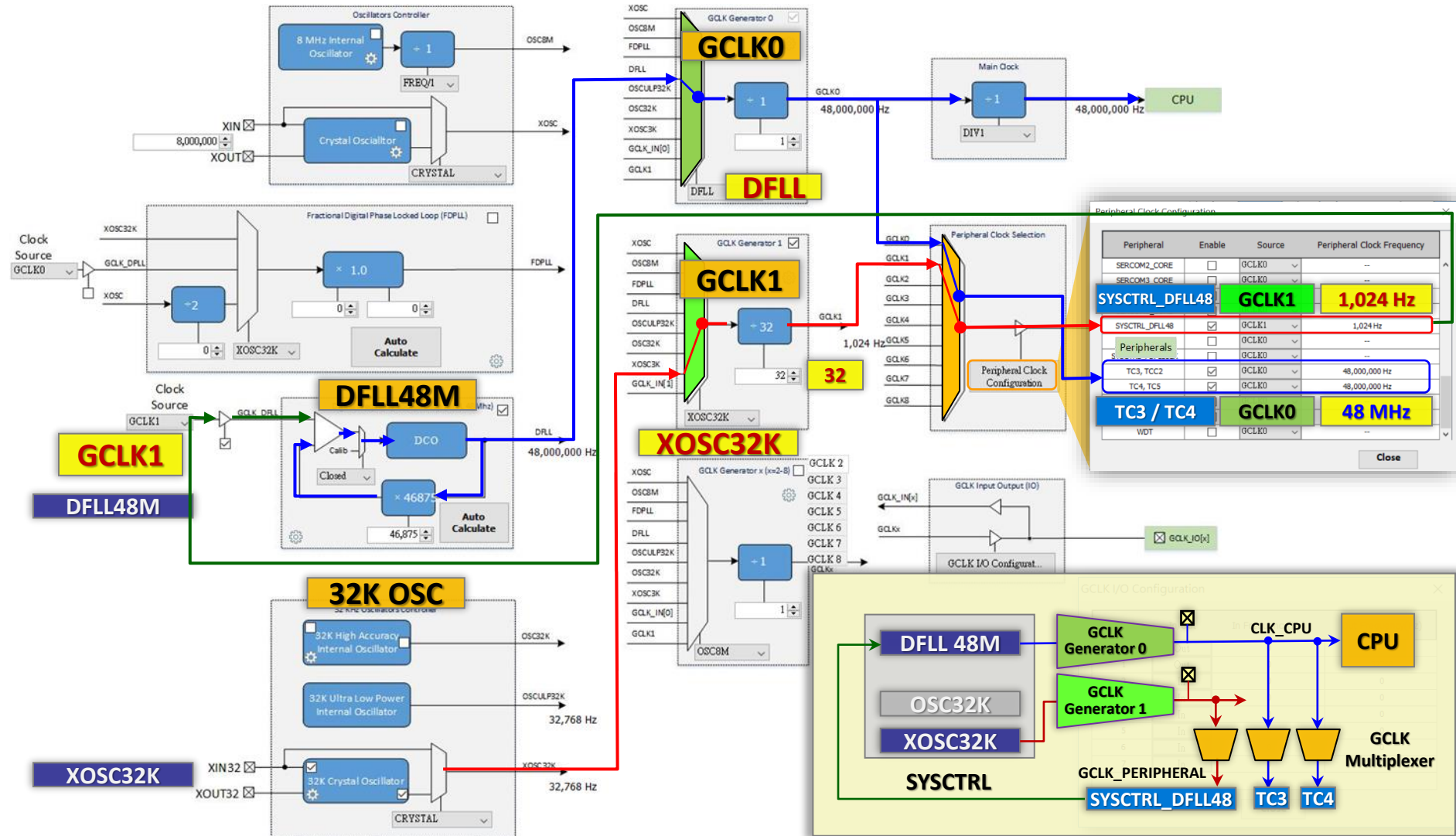
Configure DFLL48M(Close loop) as source of main clock(GCLK0).

- Enable **XOSC32K** clock source.
- Enable **GCLK Generator 1** and assign **XOSC32K** as source of it.
- Enable **DFLL48M** and assign **GCLK1** as source or it.
- Select **Close** Loop calibration of **DFLL48M**.
- Click **Auto Calculate** to generate multiplier factor of **DFLL48M**.
- Assign **DFLL48M** as clock source of **GCLK Generator 0**.
- CPU wait state will auto judge by MCC Harmony.

• **Let's go!**

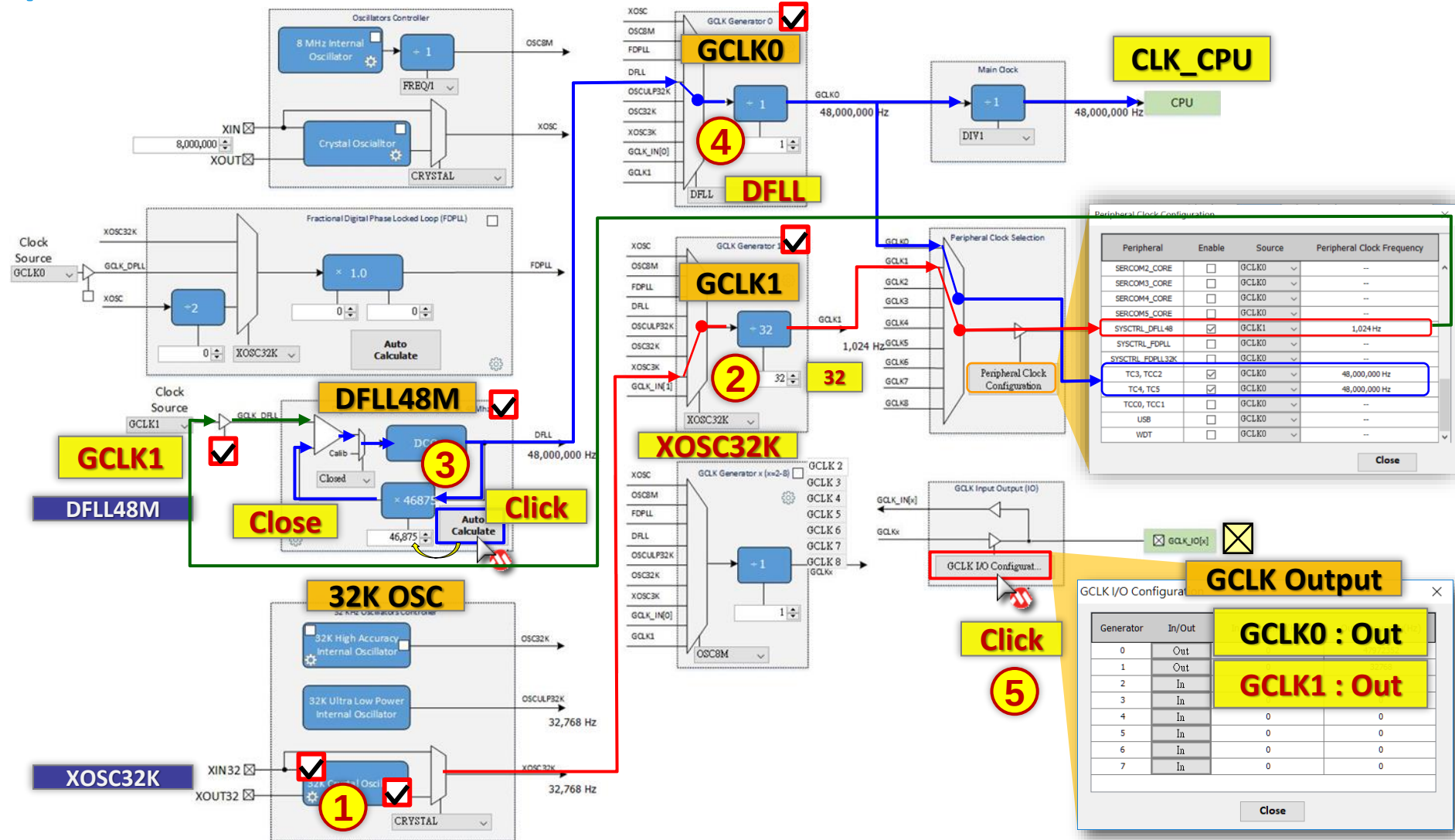
Lab8 System Clock DFL48M

Overview



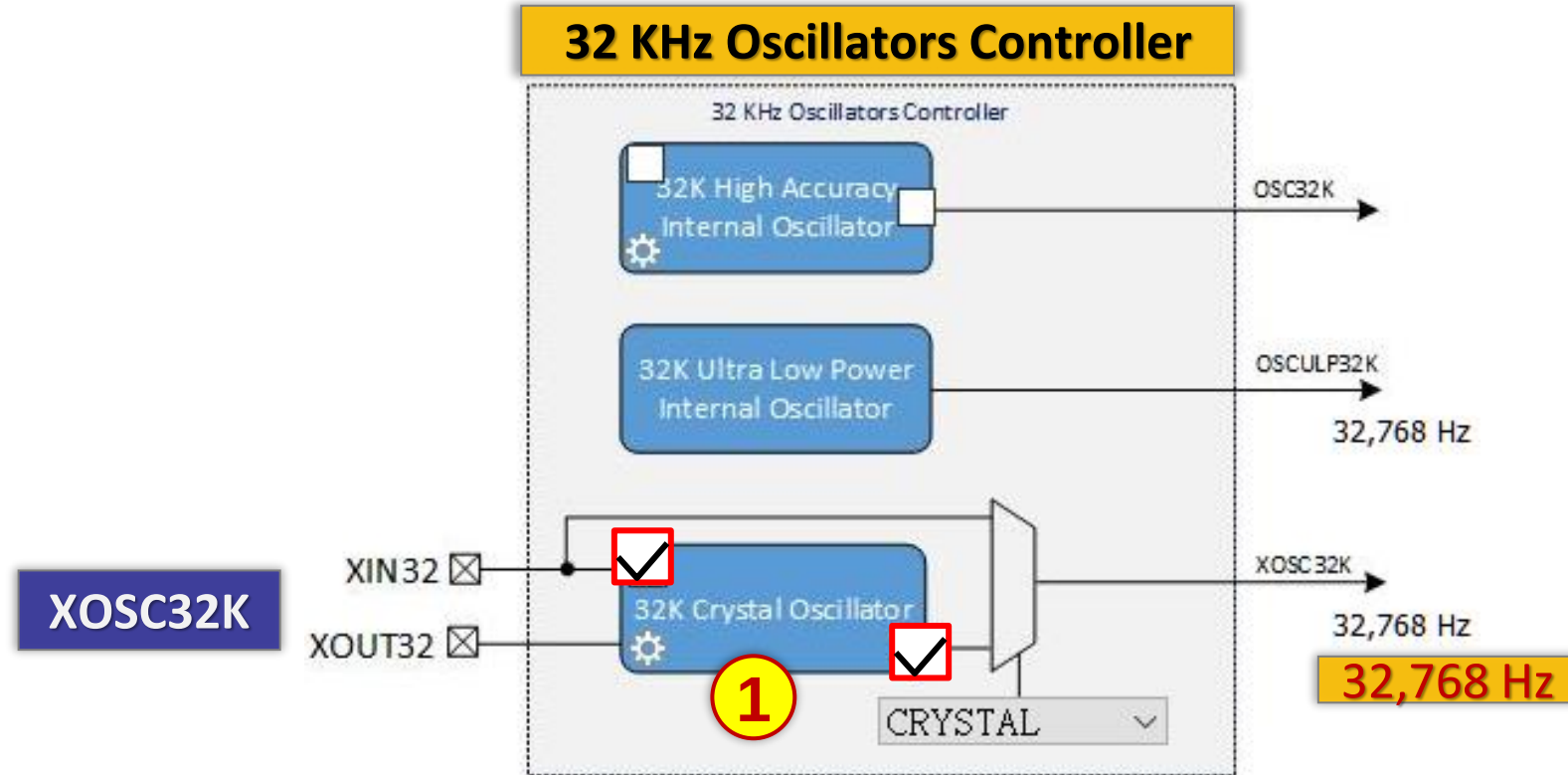
Lab8 System Clock DFL48M

Setup Steps Overview



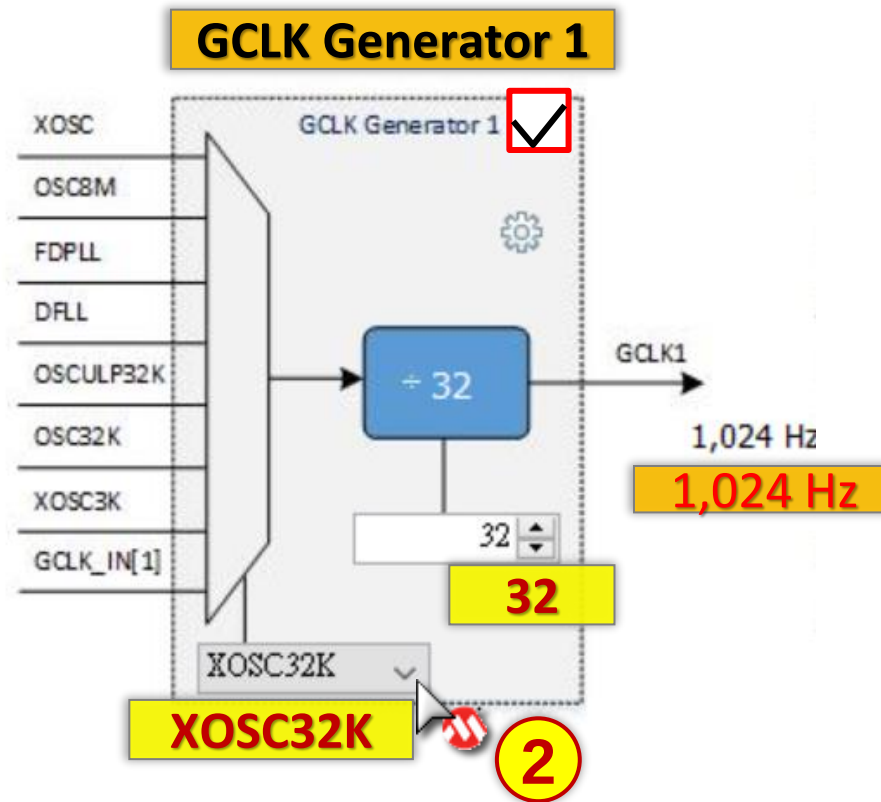
Lab8 System Clock DFLL48M

Step 1



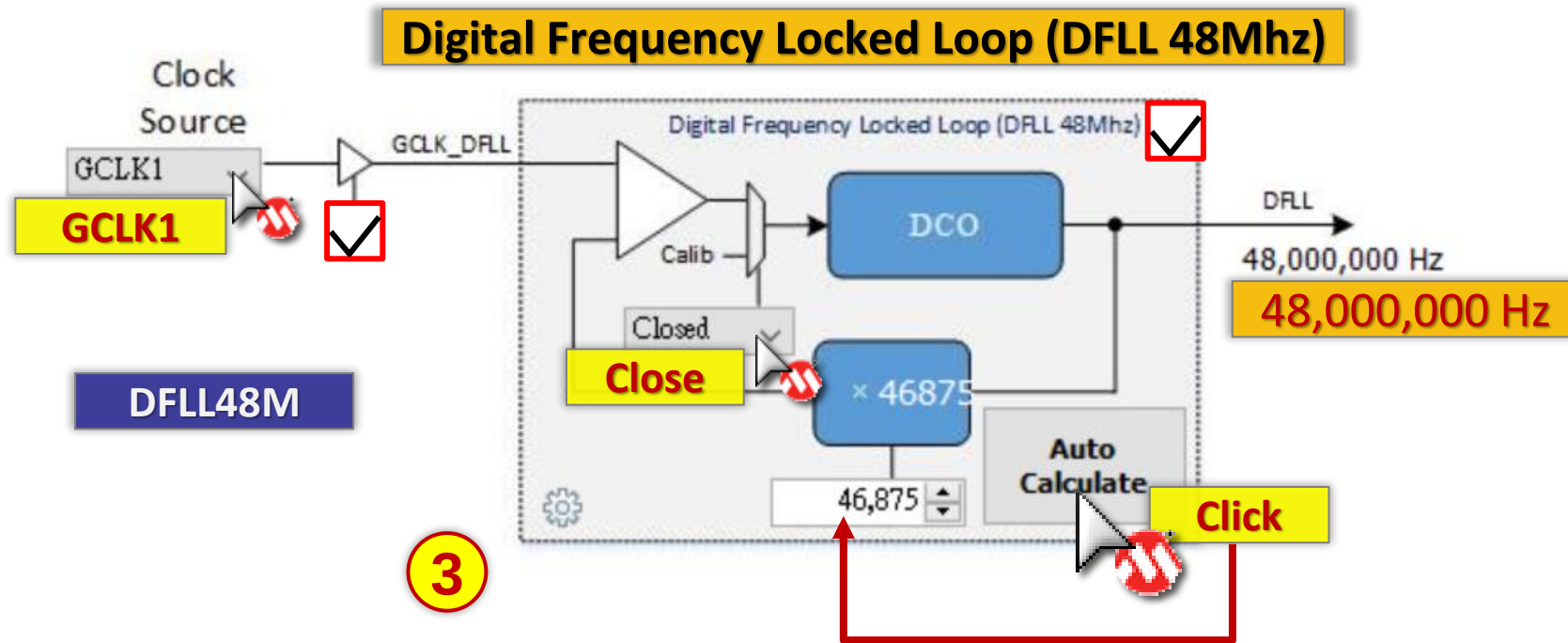
Lab8 System Clock DFLL48M

Step 2



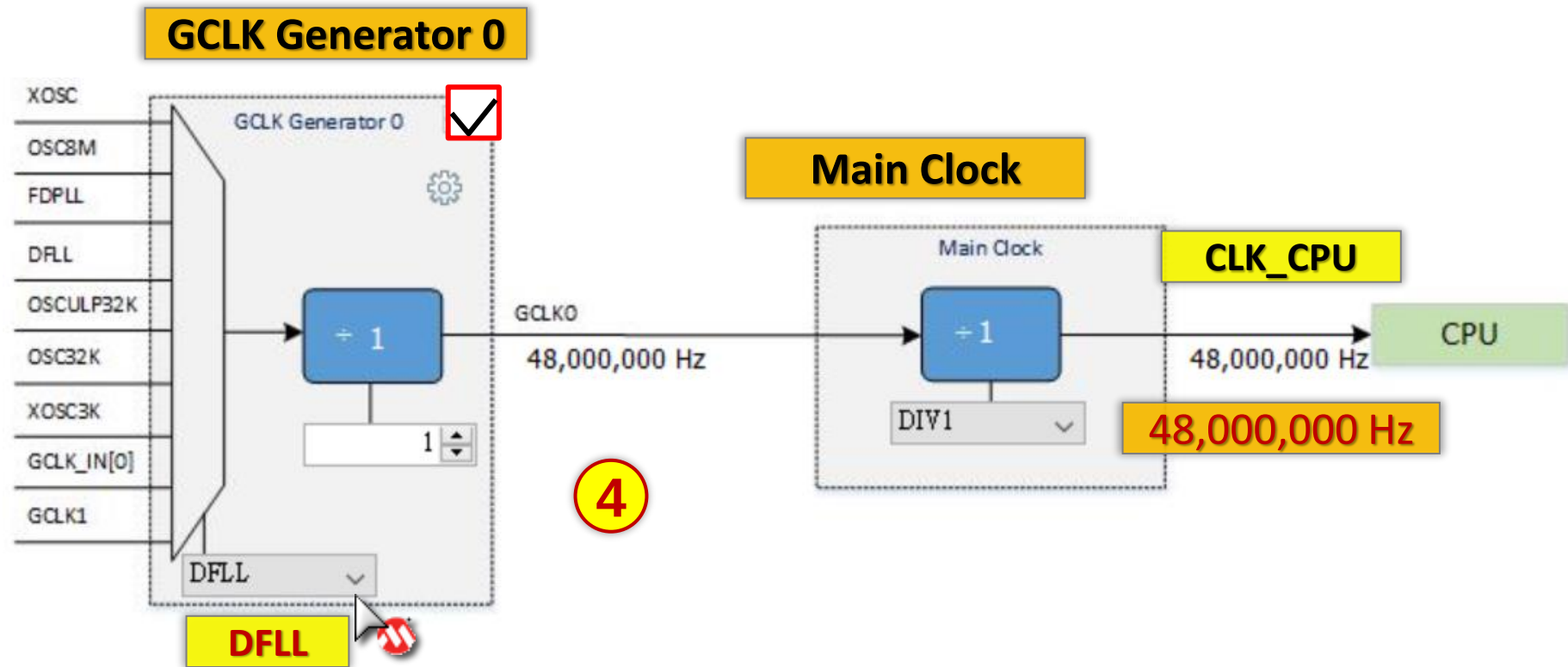
Lab8 System Clock DFL48M

Step 3



Lab8 System Clock DFLL48M

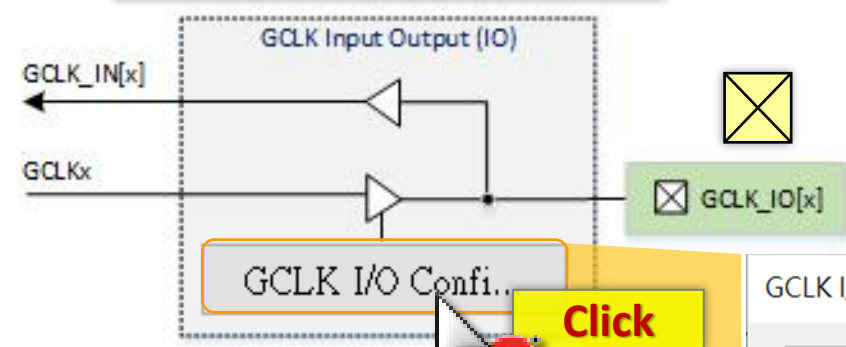
Step 4



Lab8 System Clock DFLL48M

Step 5

GCLK Input Output (IO)

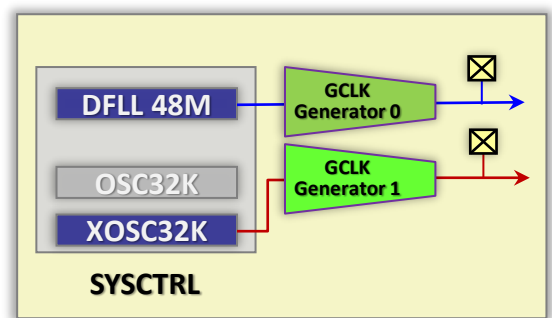


GCLK I/O Configuration

GCLK I/O Configuration

Generator	In/Out	In Frequency(Hz)	
0	Out	48000000	48000000
1	Out	1024	1024
2	In	0	0
3	In	0	0
4	In	0	0
5	In	0	0
6	In	0	0
7	In	0	0

Close



Lab8 System Clock DFLL48M

Step 6

- Please **disable GPIO PA15** pin function firstly and then reassign **PA15** as **GCLK_IO1** output pin.

Pin Table

Package: TQFP48

Module	Function	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	GCLK_I	GCLK_I	PA16	LED3	PA18	PA19	LED1
	EIC_NMI															
GCLK	GCLK_IO0															
	GCLK_IO1															
	GCLK_IO2															
	GCLK_IO3															
	GCLK_IO4															
	GCLK_IO5															
	GCLK_IO6															
GPIO	GCLK_IO7															
GPIO	GPIO															
	I2S_FS0															

PA15 (GCLK_IO1)

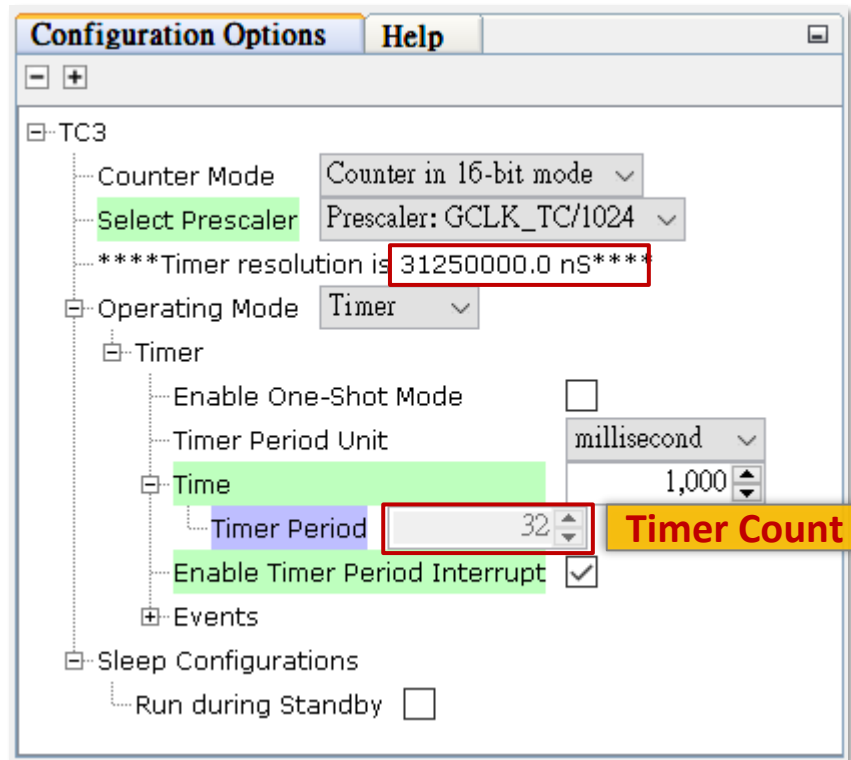
Click Enable

Click Disable

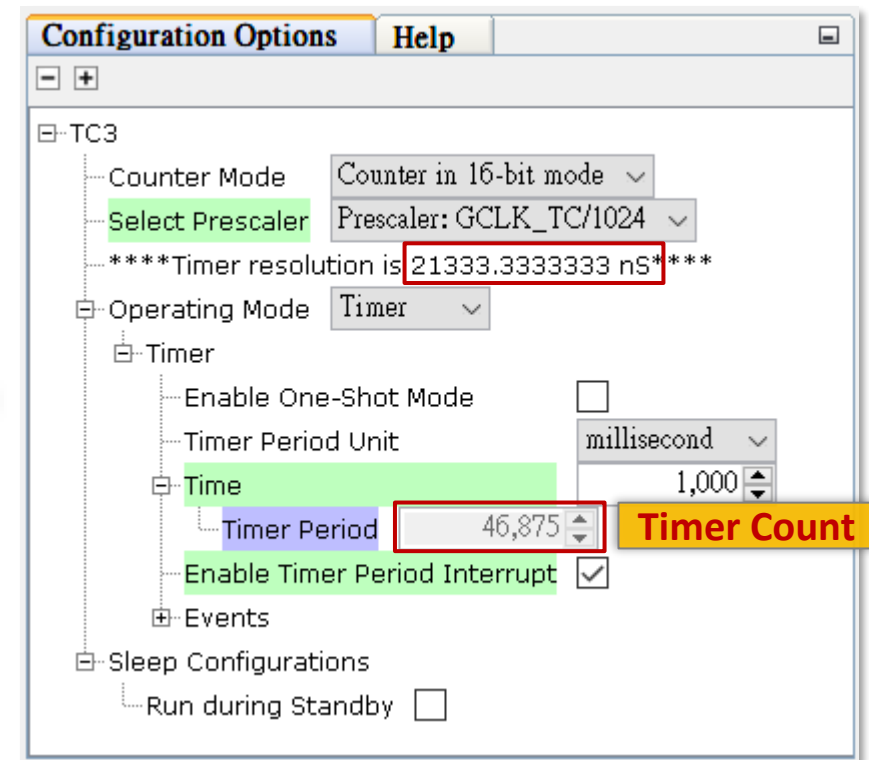
Lab8 System Clock DFLL48M

What's changed if to change main clock from 32.768KHz to DFLL Close Loop ?

- Let's check **TC3** configuration to see the difference after change the clock source.



Clock Source : XOSC32K

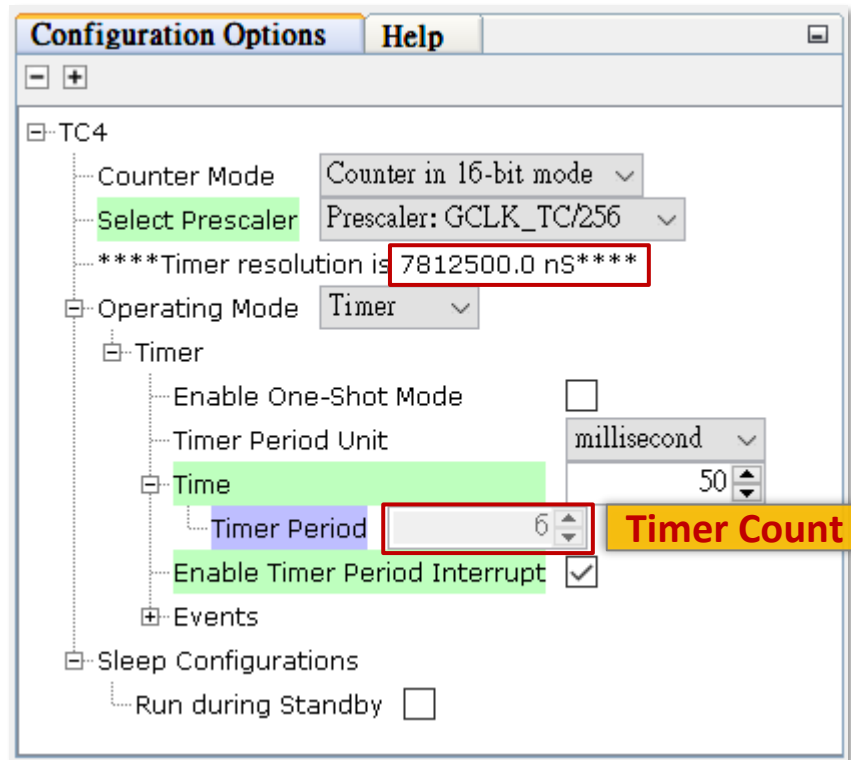


Clock Source : DFLL48M

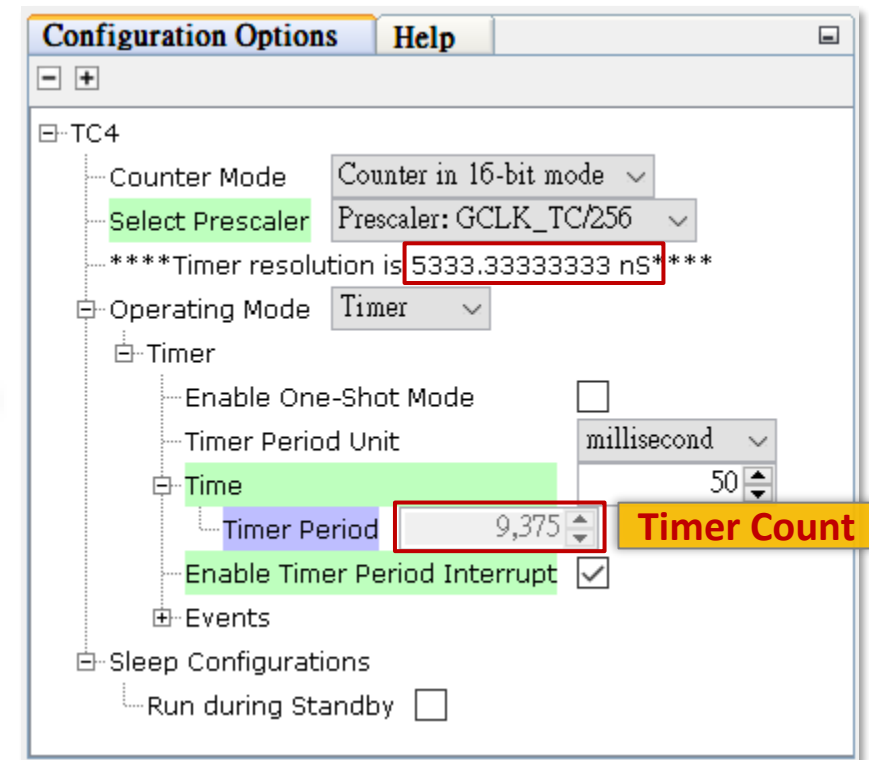
Lab8 System Clock DFLL48M

What's changed if to change main clock from 32.768KHz to DFLL Close Loop ?

- Let's check **TC4** configuration to see the difference after change the clock source.



Clock Source : XOSC32K

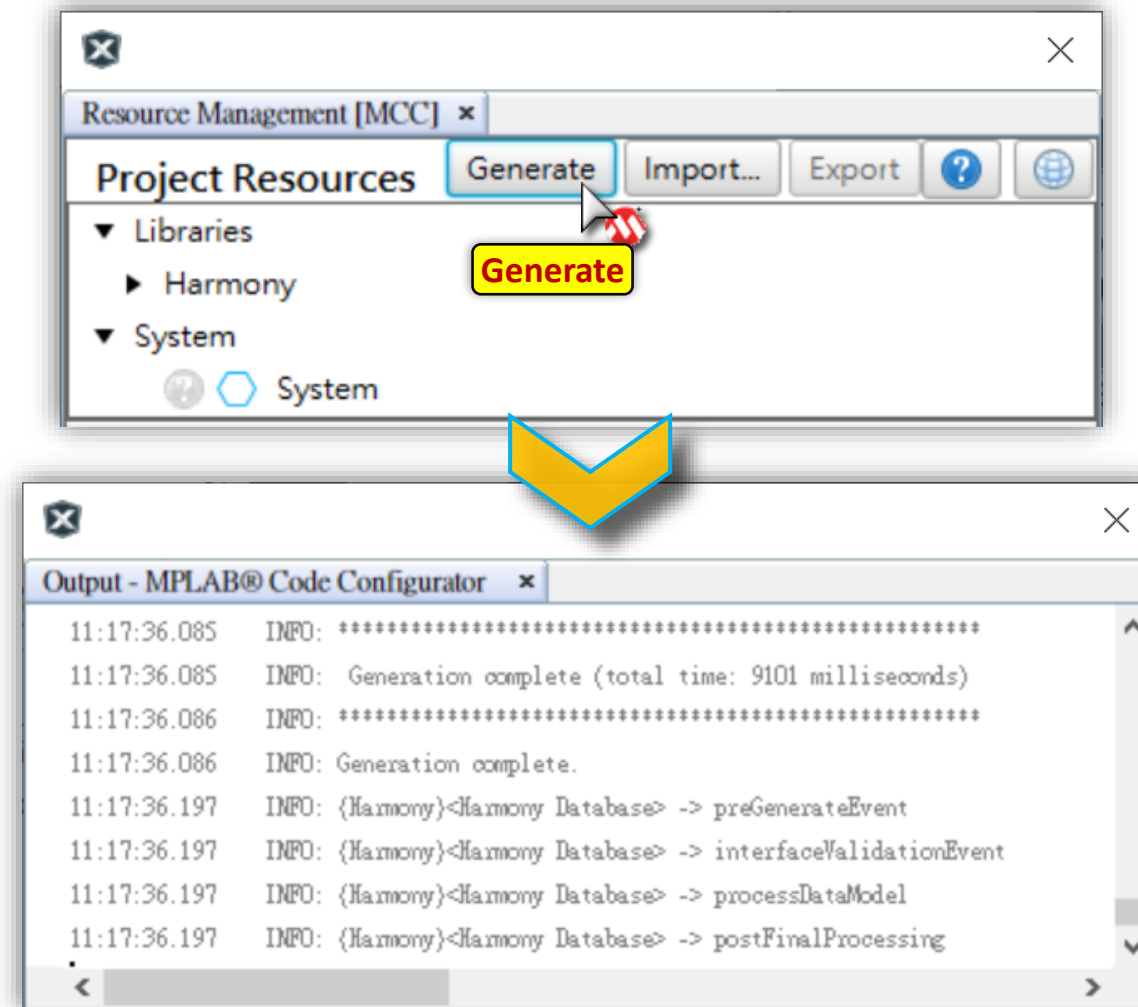


Clock Source : DFLL48M

Lab8 System Clock DFLL48M

Step 7

- Click "**Generate**" Button to Generate your Code.



Lab8 System Clock DFLL48M

Step 8

a Compiler and programming to your EVB to check the result.

```
...  
  
int main (void)  
{  
    SYS_Initialize ( NULL );  
  
    while (1)  
    {  
        ...  
  
        // TODO 7.01  
        // if ( BT2_Get() ) LED3_Clear();  
        // else          LED3_Set();  
    }  
}
```

No code need to change or modify.

Flash Access Time and Wait State

- MCC Harmony will auto adjust proper **NVM Wait State**, also.

```
plib_nvmctrl.c  
void NVMCTRL_Initialize(void)  
{  
    NVMCTRL_REGS->NVMCTRL_CTRLB =  
        NVMCTRL_CTRLB_READMODE_NO_MISS_PENALTY |  
        NVMCTRL_CTRLB_SLEEPPRM_WAKEONACCESS |  
        NVMCTRL_CTRLB_RWS(1) |  
        NVMCTRL_CTRLB_MANW_Msk;  
}
```

V _{DD} range	NVM Wait States	Maximum Operating Frequency	Units
1.62V to 2.7V	0	14	MHz
	1	28	
	2	42	
	3	48	
2.7V to 3.63V	0	24	
	1	48	

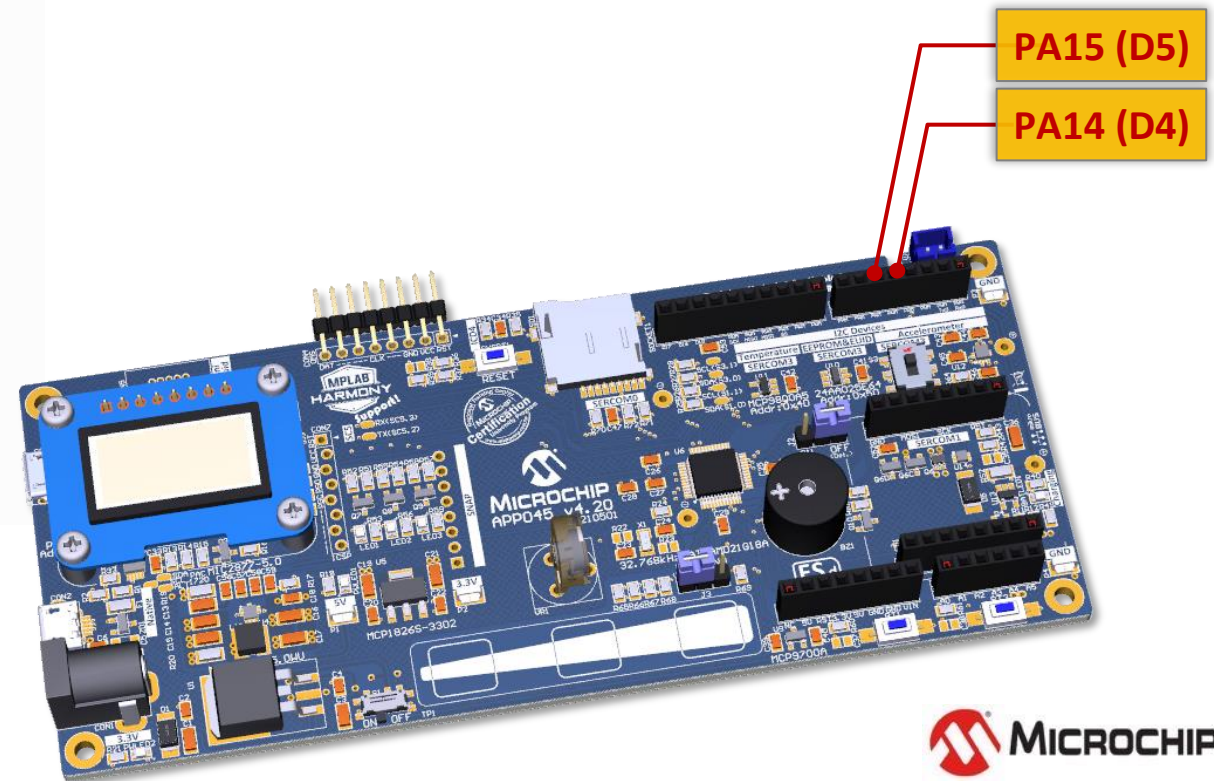
VDD = 3.3V

CPU_CLK = 48MHz

Lab8 System Clock DFLL48M

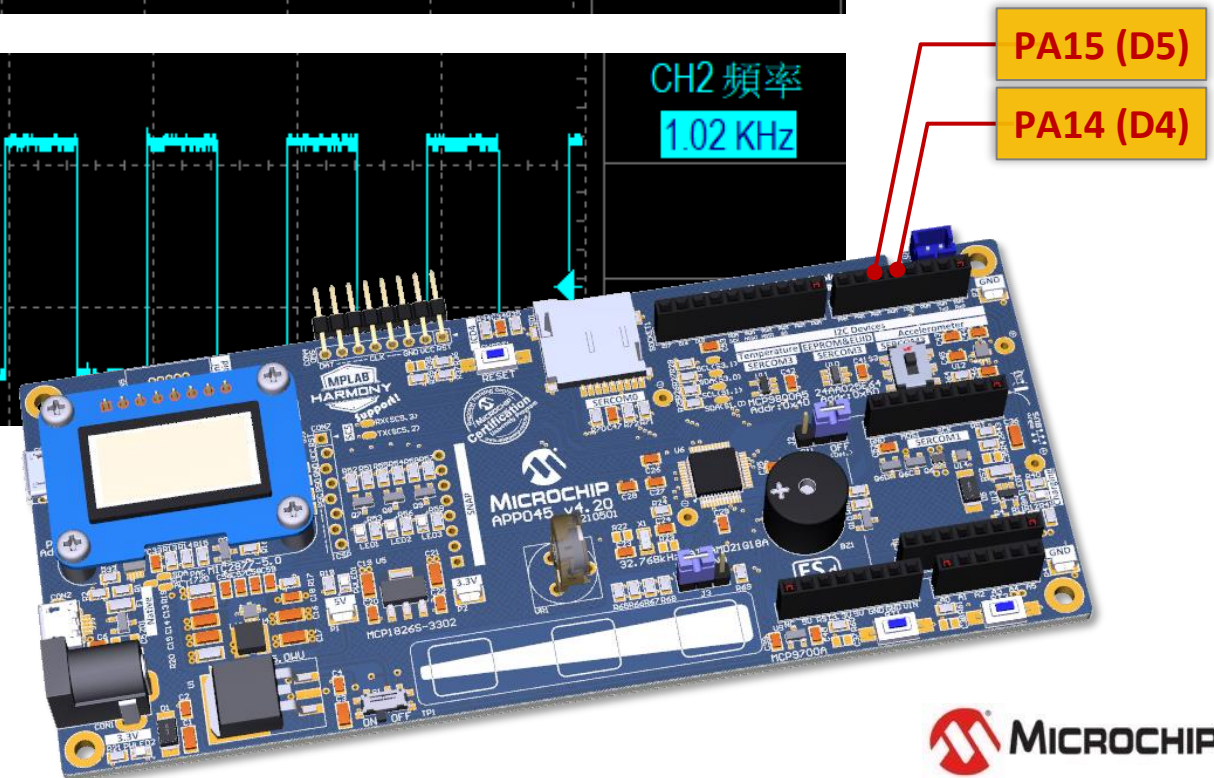
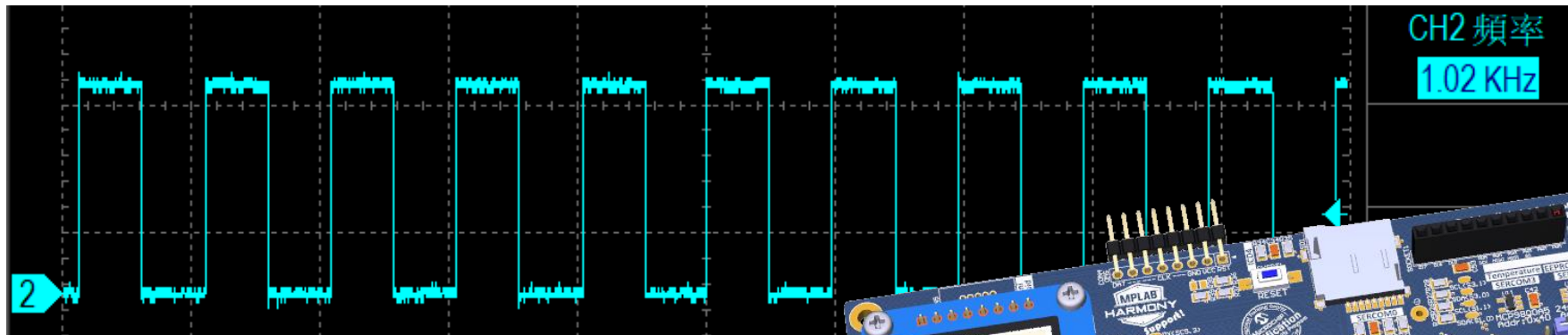
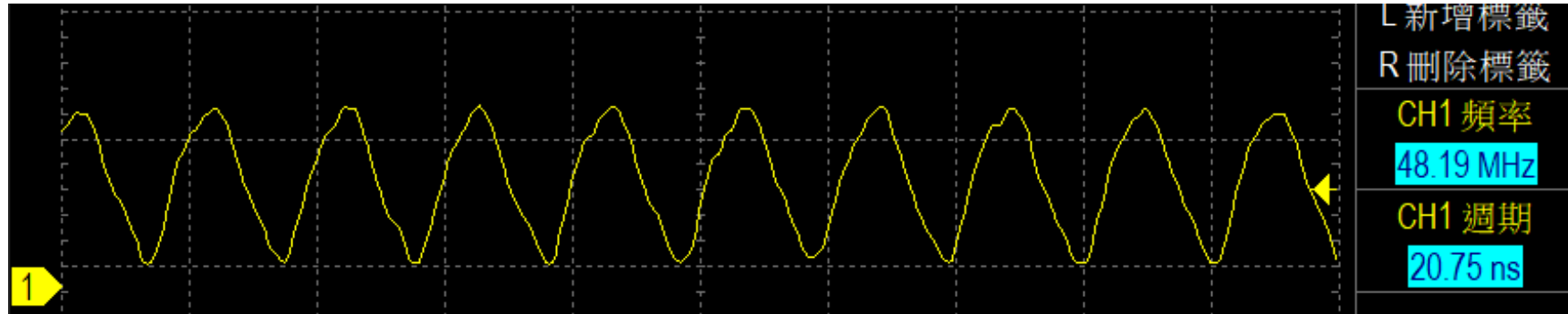
Clock Output

		Arduino UNO Socket
PA11	7	D0/RX
PA10	8	D1/TX
PA08	9	D2
PA09	10	D3/PWM
PA14	11	D4
PA15	12	D5/PWM
PA20	13	D6/PWM
PA21	14	D7



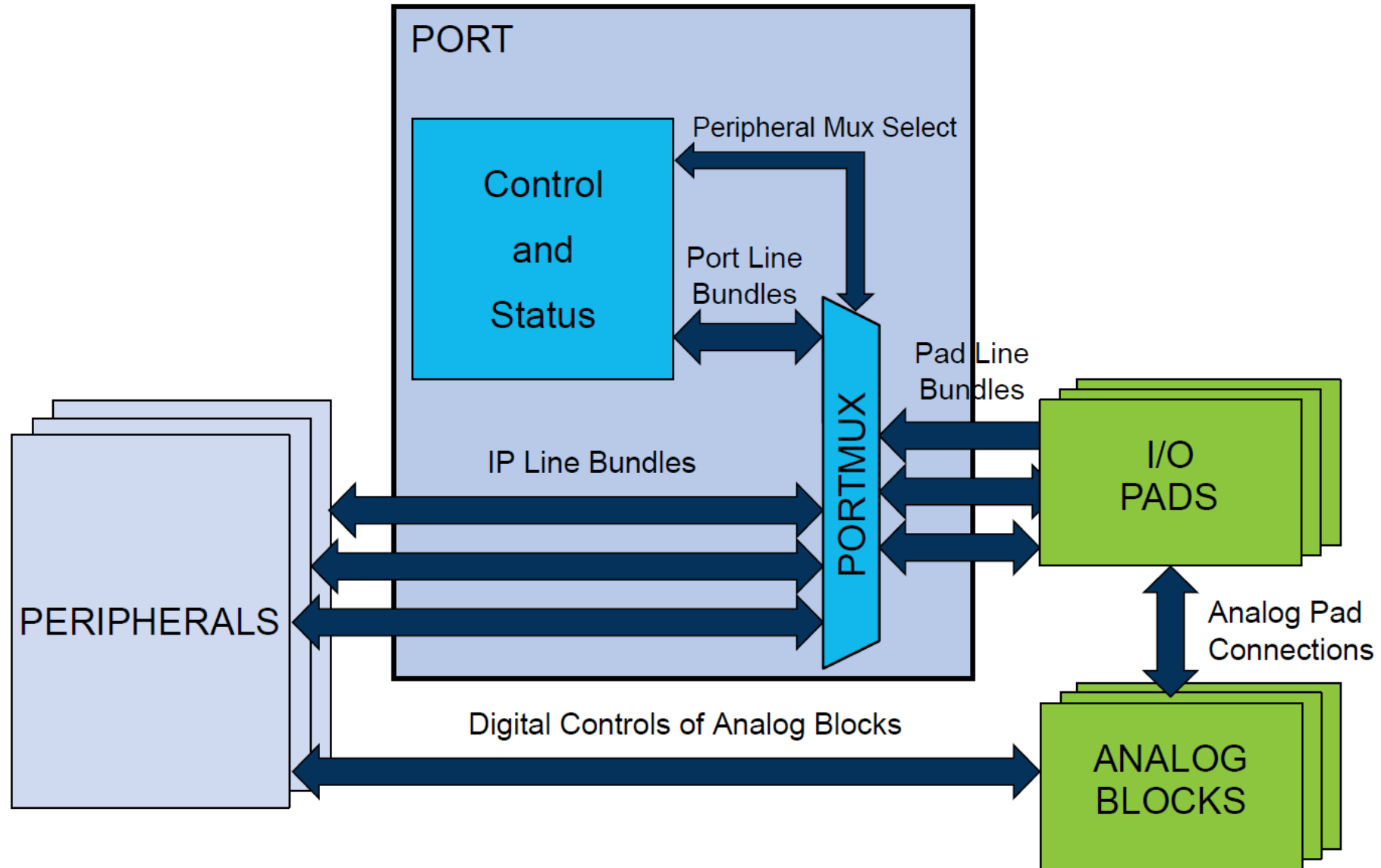
Lab8 System Clock DFLL48M

Clock Output



Pin Multiplexer (PINMUX)

What's PINMUX ?



What's PINMUX ?

- Each pin is by default controlled by the PORT as a general purpose I/O and alternatively. It can be assigned to one of the peripheral functions group.

Function Groups (A ~ H)

			PMUX		0	1					2	3	4	5	6	7
Pin ⁽¹⁾			I/O Pin	Supply	A	B ⁽²⁾⁽³⁾					C	D	E	F	G	H
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC	SERCOM ⁽²⁾⁽³⁾	SERCOM-ALT	TC ⁽⁴⁾ /TCC	TCC	COM	AC/ GCLK
15	23	31	PA14	VDDIO	EXTINT[14]						SERCOM2/ PAD[2]	SERCOM4/ PAD[2]	TC3/WO[0]	TCC0/ WO[4]		GCLK_IO[0]
16	24	32	PA15	VDDIO	EXTINT[15]						SERCOM2/ PAD[3]	SERCOM4/ PAD[3]	TC3/WO[1]	TCC0/ WO[5]		GCLK_IO[1]

Pin Name

PINMUX PLIB Analysis

main() @ main.c

```
SYS_Initialize ( NULL );
```

SYS_Initialize() @ initialization.c

```
PORT_Initialize();
```

Offset	Name	7	6	5	4	3	2	1	0
0x30	PMUX0	PMUXO[3:0]				PMUXE[3:0]			
	PMUX	PMUXO (ODD)				PMUXE (EVEN)			
0x3F	PMUX15	PMUXO[3:0]				PMUXE[3:0]			
0x40	PINCFG0		DRVSTR				PULLEN	INEN	PMUXEN
	PINCFG								
0x5F	PINCFG31		DRVSTR				PULLEN	INEN	PMUXEN

INEN PMUXEN

PORT_Initialize() @ plib_port.c

```
PORT_REGS->GROUP[0].PORT_DIR = 0x120200;  
PORT_REGS->GROUP[0].PORT_OUT = 0x200;  
PORT_REGS->GROUP[0].PORT_PINCFG[14] = 0x3;  
PORT_REGS->GROUP[0].PORT_PINCFG[15] = 0x3;  
  
PORT_REGS->GROUP[0].PORT_PMUX[7] = 0x77;
```

PA14 (GCLK_IO0)

PA15 (GCLK_IO1)

INEN = 1 (Button)

INEN = 1 (Button)

PMUX[7] : PA15 , PA14

PINMUX PLIB Analysis

			PMUX	0	1					2	3	4	5	6	7	
Pin ⁽¹⁾			I/O Pin	Supply	A	B ⁽²⁾⁽³⁾					C	D	E	F	G	H
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC	SERCOM ⁽²⁾⁽³⁾	SERCOM-ALT	TC ⁽⁴⁾ /TCC	TCC	COM	AC/ GCLK
15	PA14 (7)		PA14	VDDIO	EXTINT[14]						SERCOM2/ PAD[2]	SERCOM4/ PAD[2]	TC3/WO[0]	TCC WO[4]	GCLK_IO[0]	GCLK_IO[0]
16	PA15 (7)		PA15	VDDIO	EXTINT[15]						SERCOM2/ PAD[3]	SERCOM4/ PAD[3]	TC3/WO[1]	TCC WO[5]	GCLK_IO[1]	GCLK_IO[1]

PORT_Initialize() @ plib_port.c

```
PORT_REGS->GROUP[0].PORT_DIR = 0x120200;
PORT_REGS->GROUP[0].PORT_OUT = 0x200;
PORT_REGS->GROUP[0].PORT_PINCFG[14] = 0x3;
PORT_REGS->GROUP[0].PORT_PINCFG[15] = 0x3;
```

```
PORT_REGS->GROUP[0].PORT_PMUX[7] = 0x77;
```

PMUX[7] : PA15 , PA14

PINMUX Configuration Example

MPLAB X IDE v6.05 - Lab1_PORT_Output : default

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

default PC: 0x0 How do I? Keyword(s)

Projects Files Services Resource Management [MCC] x

Project Resource... Gener... Impor... Exp... ?

Libraries

- Harmony
- System
 - System

Device Resources Content Manager

Libraries

- Harmony

Lab1_PORT_Output - Dash... Navigator Core Versions [MCC] x

MCC Core Versions

- MPLAB® Code Configurator (Plugin) v5.2.2
- Libraries may be updated in the Content Manager.

Project Graph x Pin Settings x

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch
1	PA00		Available	Digital	High Impedance	Low
2	PA01		Available	Digital	High Impedance	Low
3	PA02		Available	Digital	High Impedance	Low
4	PA03		Available	Digital	High Impedance	Low
5	GNDANA			Digital	High Impedance	Low
6	VDDANA			Digital	High Impedance	Low
7	PB08		Available	Digital	High Impedance	Low
8	PB09		Available	Digital	High Impedance	Low
9	PA04		Available	Digital	High Impedance	Low
10	PA05		Available	Digital	High Impedance	Low
11	PA06		Available	Digital	High Impedance	Low
12	PA07		Available	Digital	High Impedance	Low
13	PA08		Available	Digital	High Impedance	Low
14	PA09		Available	Digital	High Impedance	Low

Pin Diagram x

ATSAM21G18A

Pin Table x

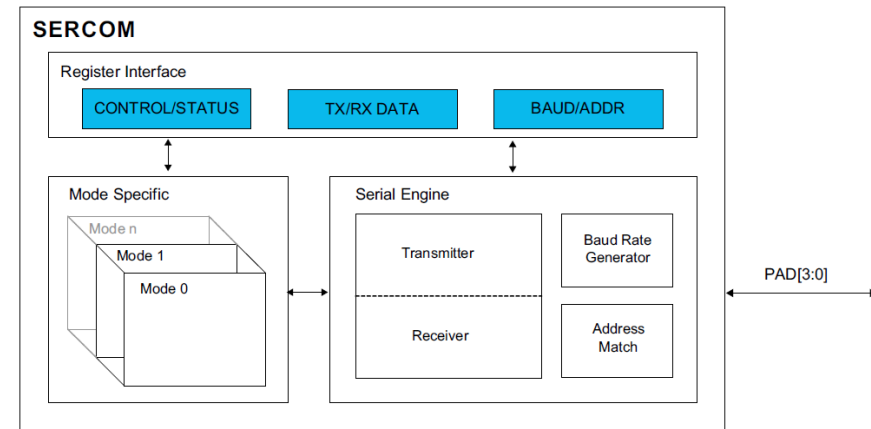
Package: TQFP48

Module	Function	PA00	PA01	PA02	PA03	GNDANA	VDDANA	PB08	PB09	PA04	PA05	PA06	PA07	PA08	PA09	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	PA14	PA15
AC	AC_AIN0																								
	AC_AIN1																								
	AC_AIN2																								
	AC_AIN3																								
	AC_CMP0																								
	AC_CMP1																								

SERCOM – UART Architecture

SERCOM Introduction

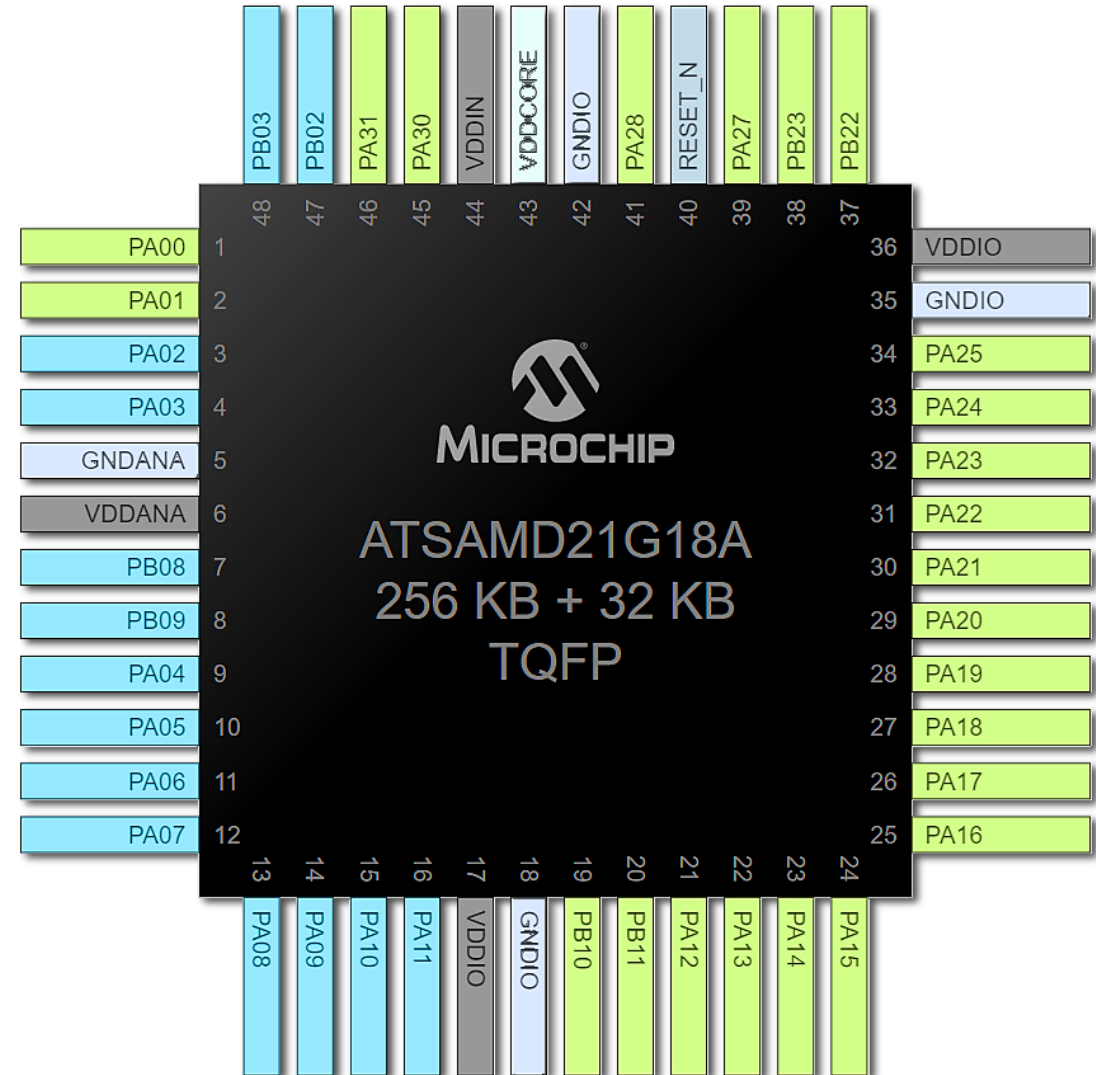
- Serial Communication Interface, SERCOM be configured to support a number of modes :
 - I2C
 - SPI
 - USART : RS232, RS485, IrDA
- There are up to six instances of the serial communication interface SERCOM peripheral.
- The SERCOM serial engine consists of a transmitter and receiver, baud-rate generator and address matching functionality. It can use the internal generic clock or an external clock to operate in all sleep.



SERCOM Introduction

- Pin Multiplex

	PAD[0]	PAD[1]	PAD[2]	PAD[3]	PMUX
SERCOM0	PA04	PA05	PA06	PA07	D
	PA08	PA09	PA10	PA11	C
SERCOM1	PA00	PA01	PA30	PA31	D
	PA16	PA17	PA18	PA19	C
SERCOM2	PA08	PA09	PA10	PA11	D
	PA12	PA13	PA14	PA15	C
SERCOM3	PA16	PA17	PA18	PA19	D
			PA20	PA21	D
	PA22	PA23	PA24	PA25	C
SERCOM4	PA12	PA13	PA14	PA15	D
	PB08	PB09	PB10	PB11	D
SERCOM5			PA20	PA21	C
	PA22	PA23	PA24	PA25	D
	PB02	PB03	PB22	PB23	D



SERCOM Introduction

- Pin Multiplex in Harmony (Clear)

Pin Table																																																	
Package: TQFP48		1	2	3	4	9	10	11	12	13	14	15	16	21	22	23	24	25	26	27	28	29	30	31	32	33	34	39	41	45	46	47	48	7	8	19	20	37	38										
		A																															B																
Module	Function	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	27	28	30	31	2	3	8	9	10	11	22	23										
SERCOM0	SERCOM0_PAD0																																																
	SERCOM0_PAD1																																																
	SERCOM0_PAD2																																																
	SERCOM0_PAD3																																																
SERCOM1	SERCOM1_PAD0																																																
	SERCOM1_PAD1																																																
	SERCOM1_PAD2																																																
	SERCOM1_PAD3																																																
SERCOM2	SERCOM2_PAD0																																																
	SERCOM2_PAD1																																																
	SERCOM2_PAD2																																																
	SERCOM2_PAD3																																																
SERCOM3	SERCOM3_PAD0																																																
	SERCOM3_PAD1																																																
	SERCOM3_PAD2																																																
	SERCOM3_PAD3																																																
SERCOM4	SERCOM4_PAD0																																																
	SERCOM4_PAD1																																																
	SERCOM4_PAD2																																																
	SERCOM4_PAD3																																																
SERCOM5	SERCOM5_PAD0																																																
	SERCOM5_PAD1																																																
	SERCOM5_PAD2																																																
	SERCOM5_PAD3																																																

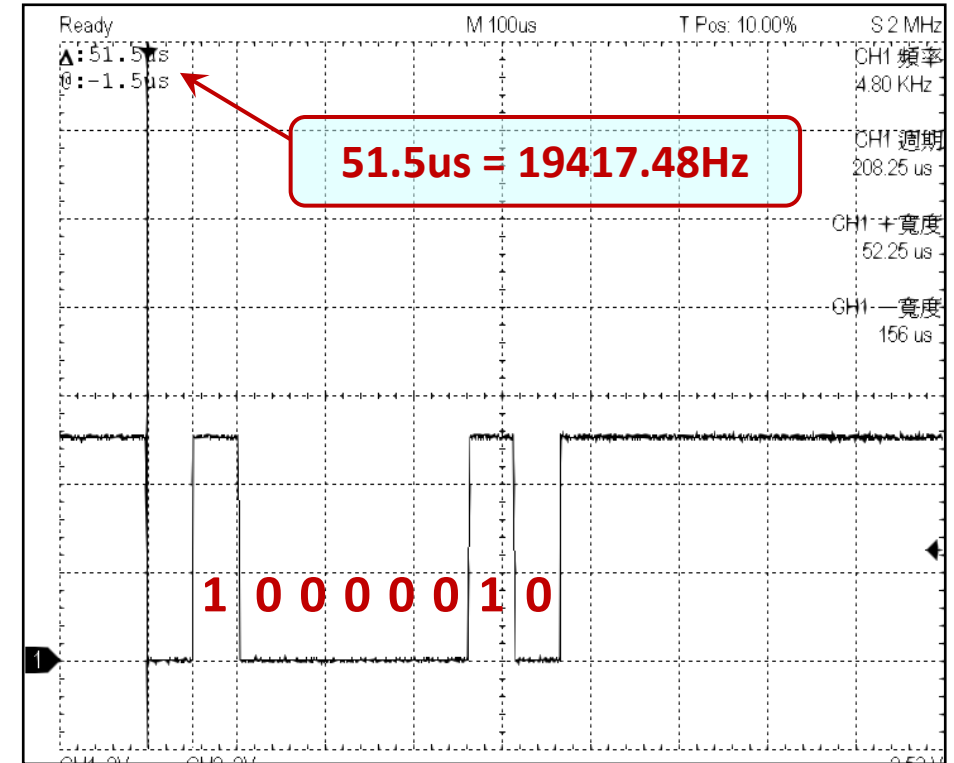
SERCOM Introduction

- **For I2C**
 - Fast Mode (400kHz)
 - Master or slave operation
 - SMBUS compatible
 - Wake on address match
- **For SPI**
 - Up to 24Mb/s
 - Supports all four SPI modes
 - Full-duplex operation
 - Single data direction mode frees unused MISO or MOSI pin
- **For USART**
 - Up to 24Mb/s
 - Half-/Full-duplex operation
 - Sync and Async operation

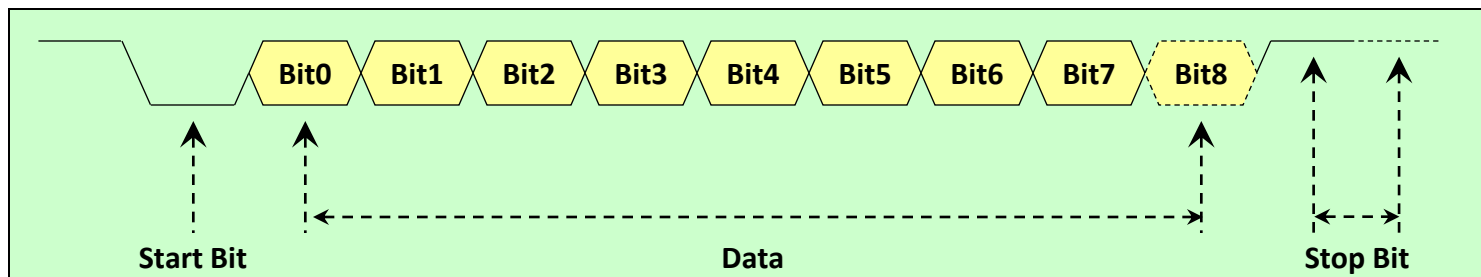


What's UART ?

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

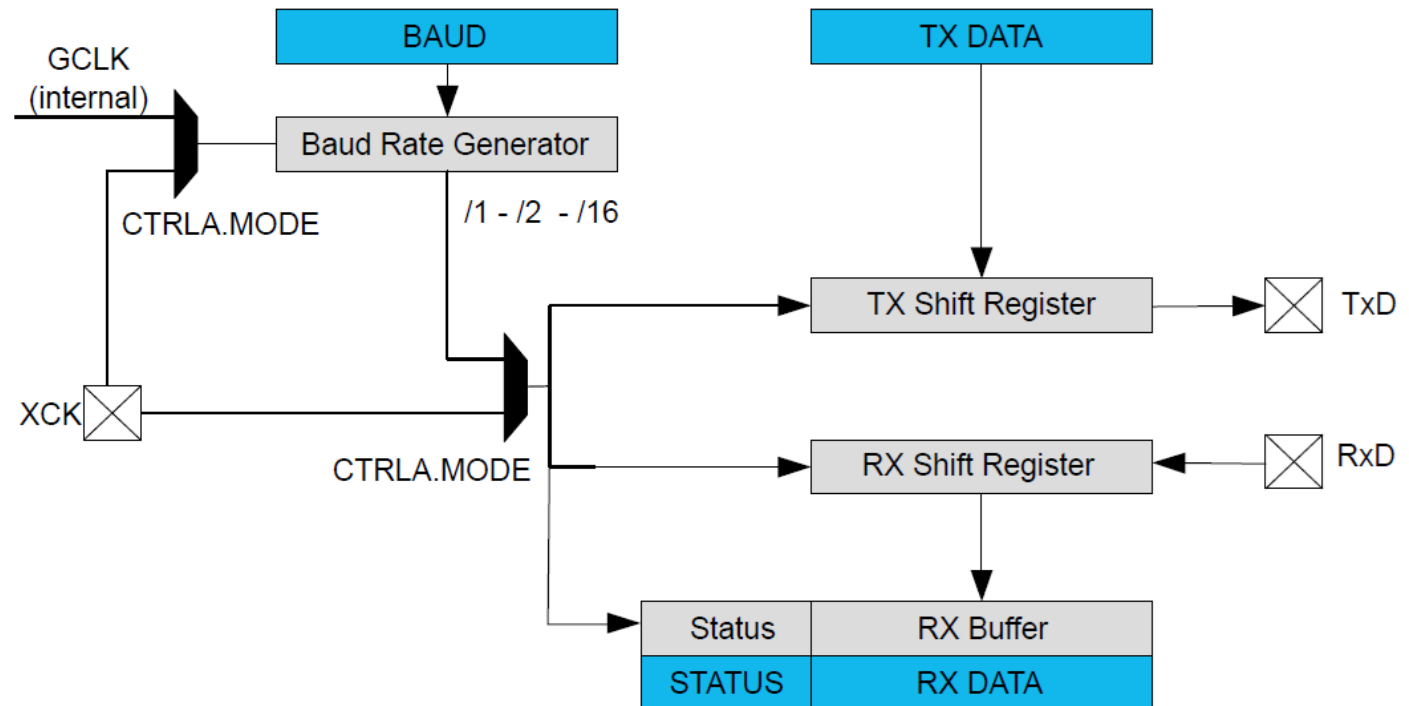


**UART Transmit Example :
'A'(0x41), 19200 bps**



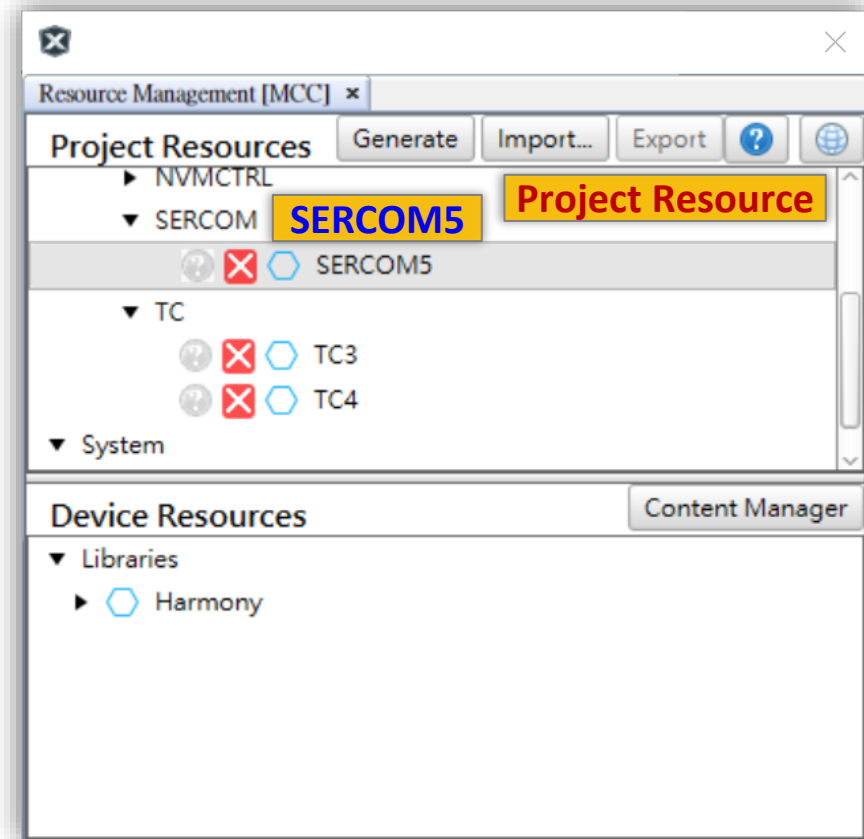
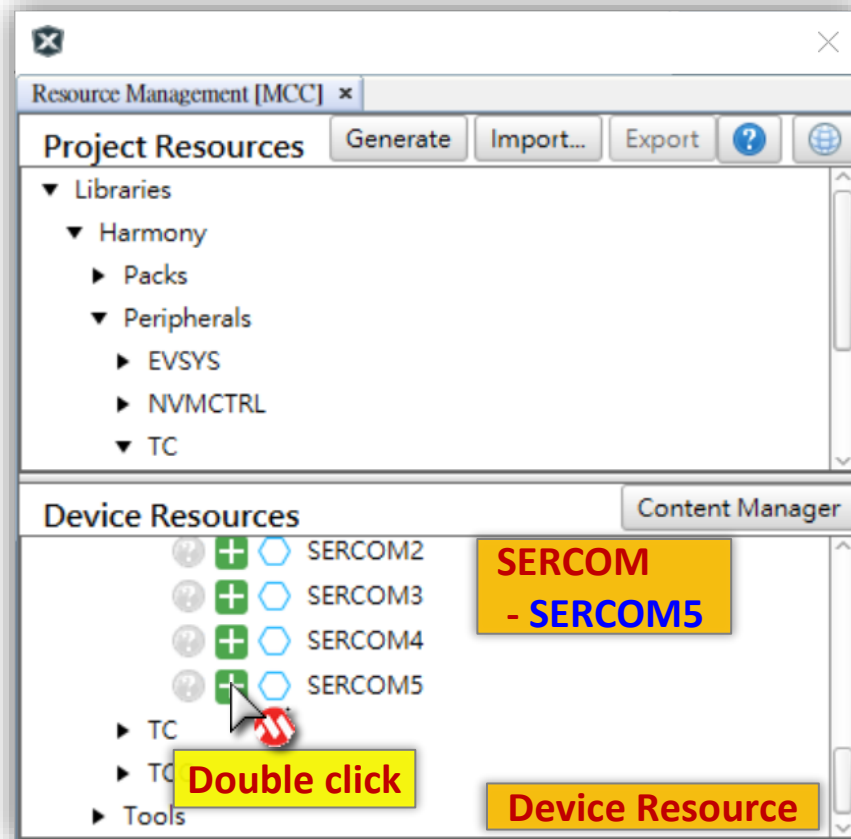
SERCOM - USART

- Supports serial frames with 5, 6, 7, 8 or 9 data bits and 1 or 2 stop bits, Parity Check, RTS and CTS flow control.
- Full-duplex operation, RS-232, RS-485 and IrDA Supported.
- Buffer overflow and frame error detection.
- Internal or external clock source for asynchronous and synchronous operation.



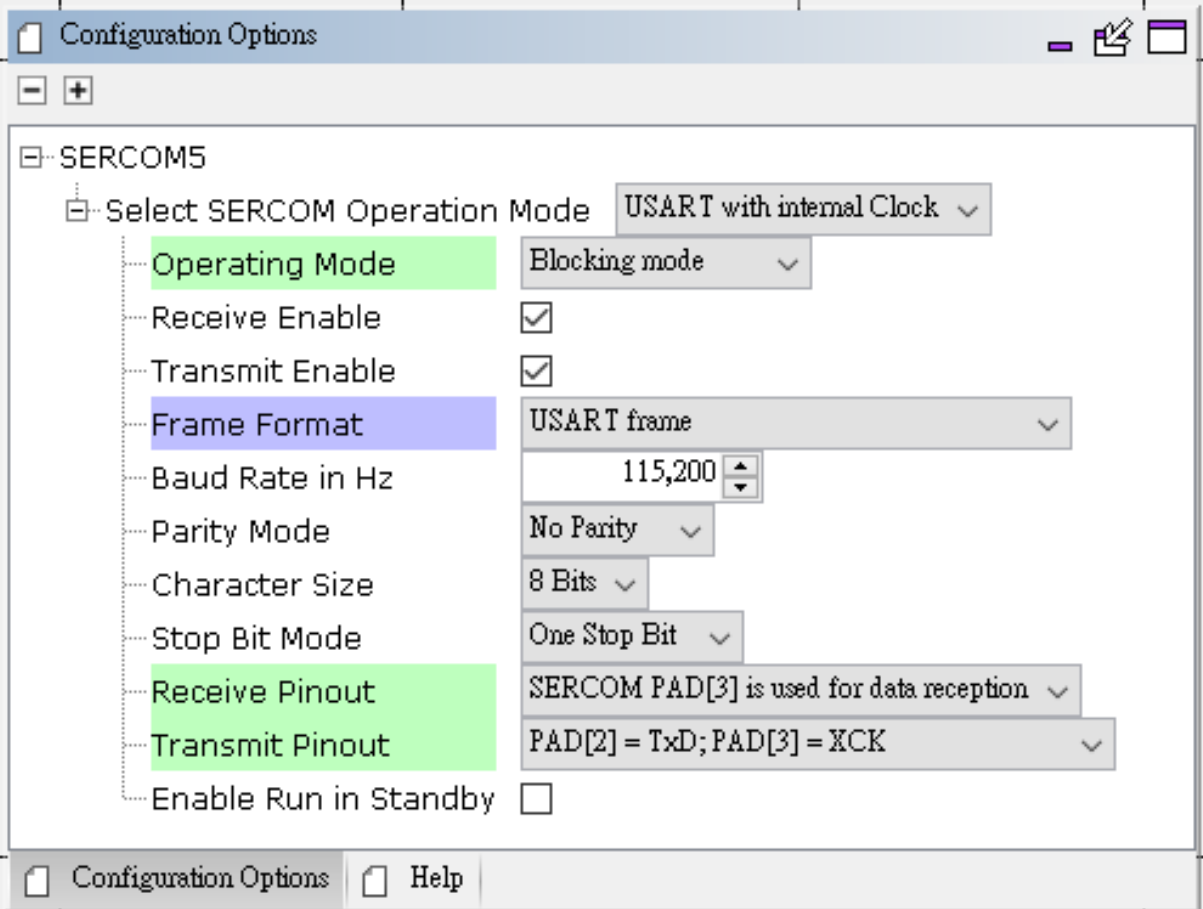
Add SERCOM Function using MCC Harmony

- Find **SERCOM5** component in Device Resource window.
- Double click **SERCOM5** to add to Project Resource window.



SERCOM USART Polling

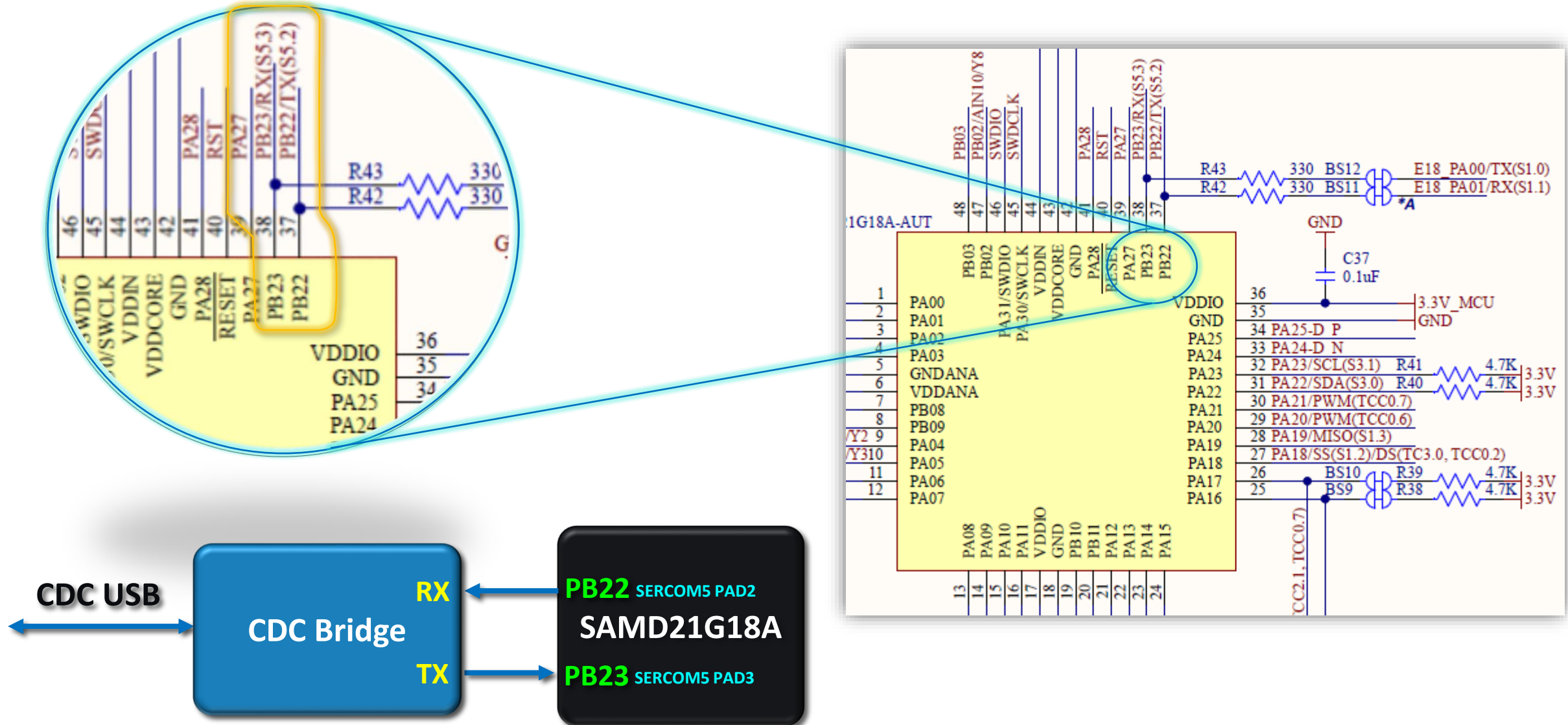
Configuration Options Example



The screenshot shows the 'Configuration Options' dialog box for the SERCOM5 peripheral. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is 'Blocking mode'. Both 'Receive Enable' and 'Transmit Enable' are checked. The 'Frame Format' is 'USART frame'. The 'Baud Rate in Hz' is 115,200. The 'Parity Mode' is 'No Parity'. The 'Character Size' is '8 Bits'. The 'Stop Bit Mode' is 'One Stop Bit'. The 'Receive Pinout' is 'SERCOM PAD[3] is used for data reception'. The 'Transmit Pinout' is 'PAD[2] = TxD; PAD[3] = XCK'. The 'Enable Run in Standby' checkbox is unchecked.

Option	Value
Select SERCOM Operation Mode	USART with internal Clock
Operating Mode	Blocking mode
Receive Enable	☒
Transmit Enable	☒
Frame Format	USART frame
Baud Rate in Hz	115,200
Parity Mode	No Parity
Character Size	8 Bits
Stop Bit Mode	One Stop Bit
Receive Pinout	SERCOM PAD[3] is used for data reception
Transmit Pinout	PAD[2] = TxD; PAD[3] = XCK
Enable Run in Standby	☐

APP045 SERCOM USART Schematic



SERCOM USART

PINMUX Configuration Example

- SERCOM allow limited alternate pad index assignment.
- For example : PB22 as SERCOM5_PAD2
PB23 as SERCOM5_PAD3

Pin ⁽¹⁾			I/O Pin	Supply	A	B ⁽²⁾⁽³⁾					C	D	E	F	G	H
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC	SERCOM ⁽²⁾⁽³⁾	SERCOM-ALT	Tc ⁽⁴⁾ /TCC	TCC	COM	AC/ GCLK
	37	49	PB22	VDDIO	EXTINT[6]							SERCOM5/ PAD[2]	TCC4/WO[0]	TCC3/ WO[0]		GCLK_IO[0]
	38		PB23	VDDIO	EXTINT[7]							SERCOM5/ PAD[3]	TCC4/WO[1]	TCC3/ WO[1]		GCLK_IO[1]

Pin Table

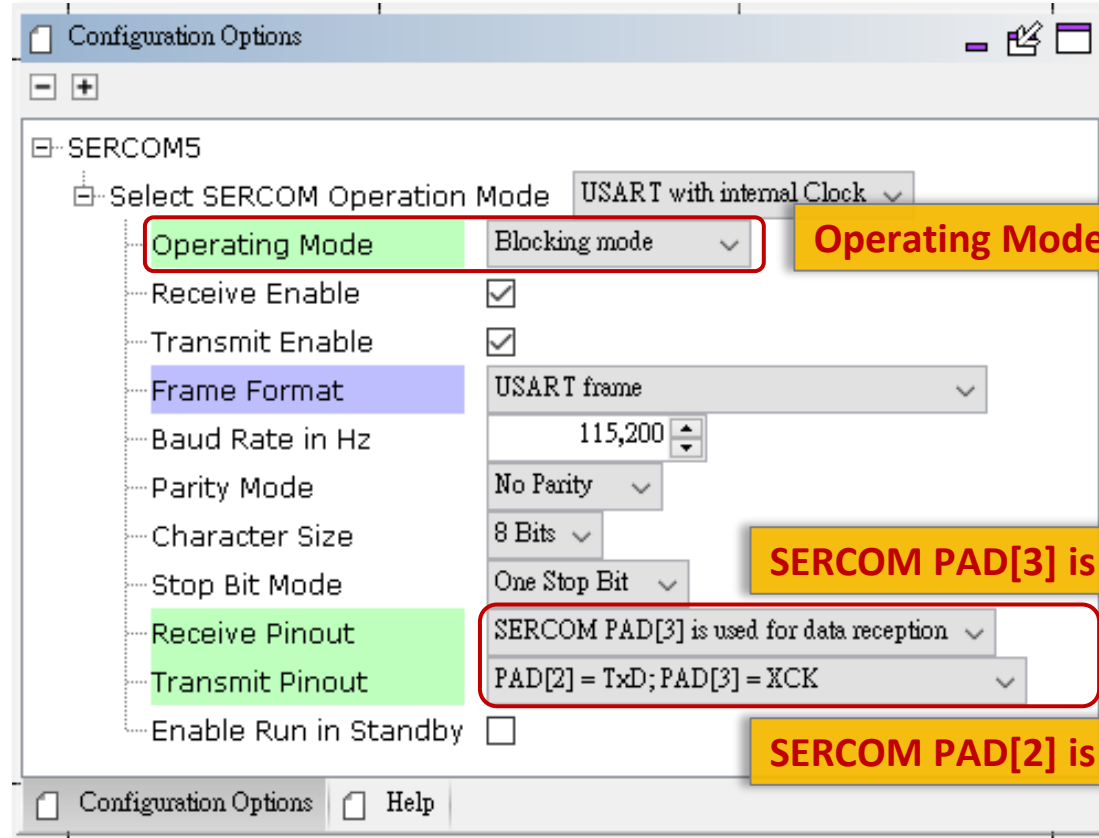
Package: TQFP48

Module	Function	PA22	PA23	PA24	PA25	GNDDIO	VDDIO	SERCOM	SERCOM	PA27
	SERCOM4_PAD3									
	SERCOM5_PAD0									
	SERCOM5_PAD1									
SERCOM5	SERCOM5_PAD2									
	SERCOM5_PAD3									
	SYSCTRL_XIN									

SERCOM USART

PINMUX Configuration Example

- Remember arrange pads function in SERCOM module.
- For example : SERCOM5 PAD[2] as USART TXD (PB22)
SERCOM5 PAD[3] as USART RXD (PB23)



The screenshot shows the 'Configuration Options' dialog for the SERCOM5 module. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is set to 'Blocking mode'. The 'Receive Enable' and 'Transmit Enable' checkboxes are checked. The 'Frame Format' is set to 'USART frame'. The 'Baud Rate in Hz' is set to 115,200. The 'Parity Mode' is set to 'No Parity'. The 'Character Size' is set to '8 Bits'. The 'Stop Bit Mode' is set to 'One Stop Bit'. The 'Receive Pinout' is set to 'SERCOM PAD[3] is used for data reception'. The 'Transmit Pinout' is set to 'PAD[2] = TxD; PAD[3] = XCK'. The 'Enable Run in Standby' checkbox is unchecked.

Operating Mode is used for Blocking mode

SERCOM PAD[3] is used for data reception

SERCOM PAD[2] is used for data transmission

SERCOM USART Polling

Interface Routines

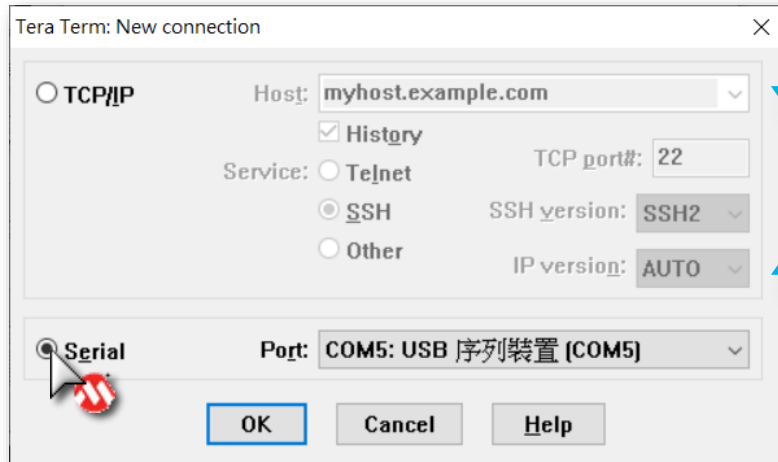
```
void SERCOM5_USART_Initialize( void );  
bool SERCOM5_USART_SerialSetup( USART_SERIAL_SETUP * serialSetup,  
                                uint32_t clkFrequency );  
bool SERCOM5_USART_Write( void *buffer, const size_t size );  
bool SERCOM5_USART_TransmitterIsReady( void );  
bool SERCOM5_USART_TransmitComplete( void );  
void SERCOM5_USART_WriteByte( int data );  
bool SERCOM5_USART_Read( void *buffer, const size_t size );  
bool SERCOM5_USART_ReceiverIsReady( void );  
int SERCOM5_USART_ReadByte( void );  
USART_ERROR SERCOM5_USART_ErrorGet( void );  
uint32_t SERCOM5_USART_FrequencyGet( void );
```

SERCOM USART Tx Polling Code Example

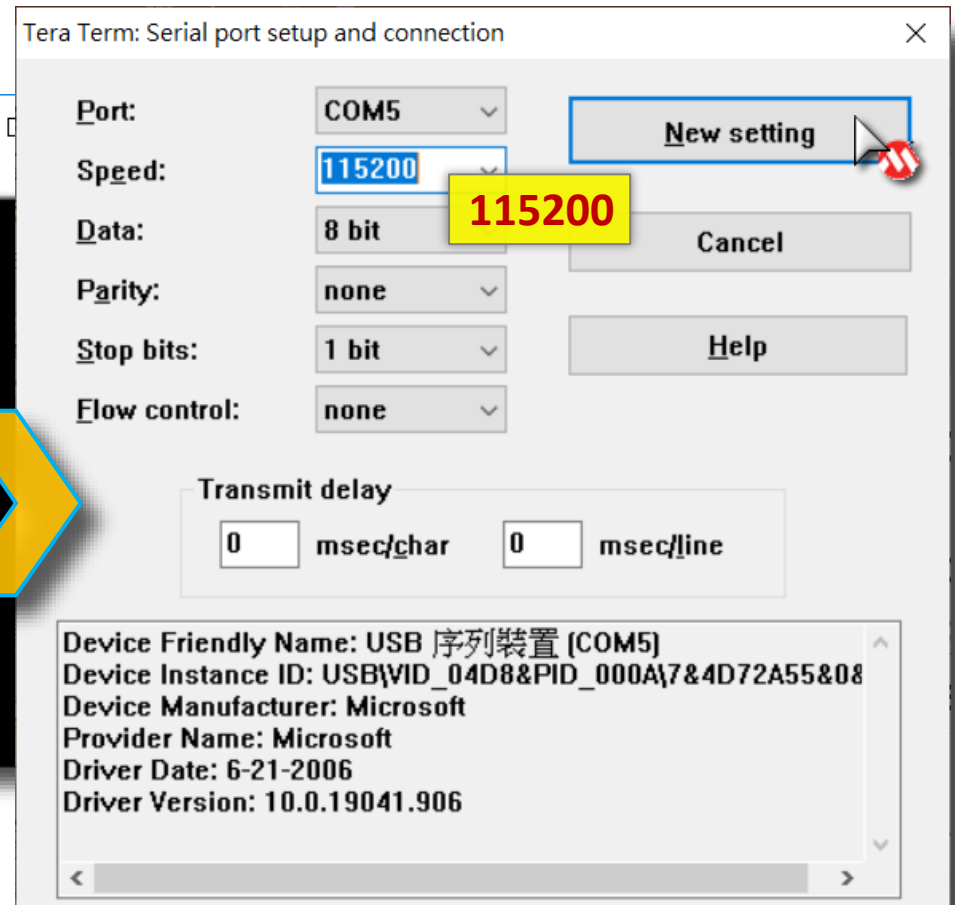
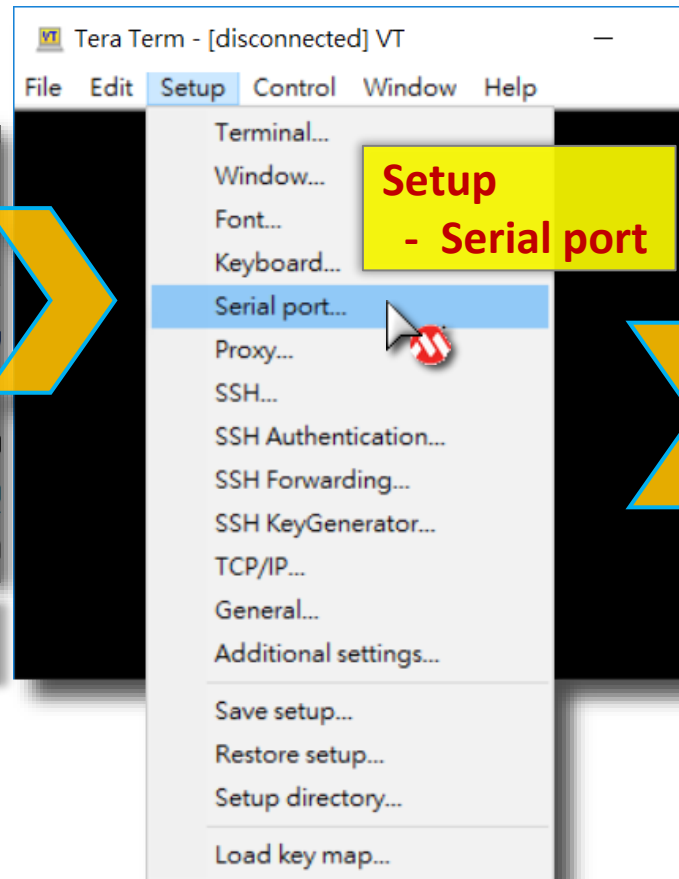
```
...  
  
int main (void)  
{  
    SYS_Initialize ( NULL );  
    ...  
  
    while( true )  
    {  
        ...  
        SERCOM5_USART_Write( "Hello world!\r\n", 14 );  
    }  
}
```

Terminal Console Software

- Tera Term is a terminal console software.
(<https://ttssh2.osdn.jp/index.html.en>)
- Select COMxx: **USB Serial Device (COMxx)**
- Set **115200-N-8-1**.



COMxx: USB Serial Device (COMxx)



Lab9 SERCOM - UART Tx Polling

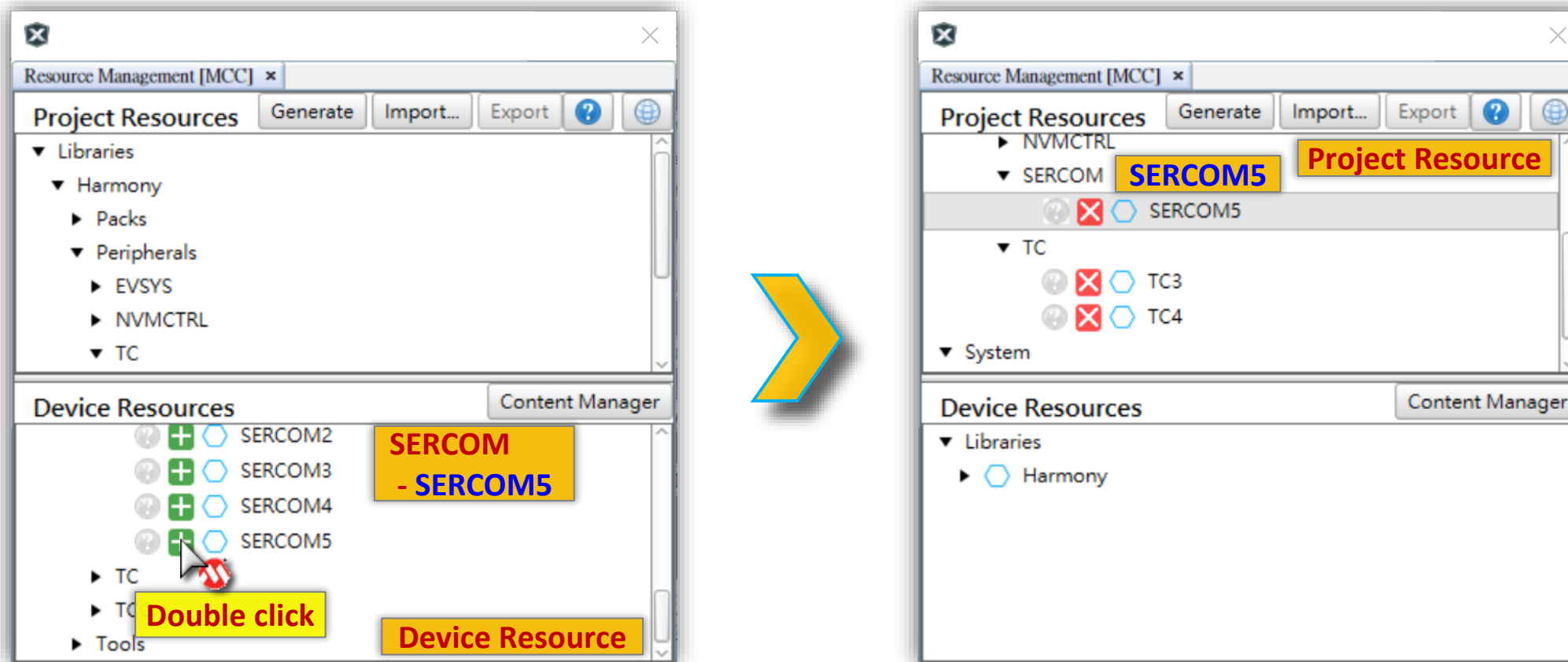
- Try to add SERCOM-USART function to project.
- UART set to **115200-N-8-1**, No flow control.
- Send string from UART to PC of **one second interval** repeated.
(Hello World!)
- Character **\r** for carriage **R**eturn (0x0D)
- Character **\n** for **N**ew line (0x0A)

- **Let's go!**

Lab9 SERCOM - UART Tx Polling

Step 1

- Find **SERCOM5** component in Device Resource window.
- Double click **SERCOM5** to add to Project Resource window.



Lab9 SERCOM - UART Tx Polling

Step 2

The screenshot shows the 'Configuration Options' dialog for the SERCOM5 peripheral. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is set to 'Blocking mode'. The 'Receive Enable' and 'Transmit Enable' checkboxes are checked. The 'Frame Format' is set to 'USART frame'. The 'Baud Rate in Hz' is set to 115,200. The 'Parity Mode' is set to 'No Parity'. The 'Character Size' is set to '8 Bits'. The 'Stop Bit Mode' is set to 'One Stop Bit'. The 'Receive Pinout' is set to 'SERCOM PAD[3] is used for data reception'. The 'Transmit Pinout' is set to 'PAD[2] = TxD; PAD[3] = XCK'. The 'Enable Run in Standby' checkbox is unchecked.

Configuration Options

SERCOM5

Select SERCOM Operation Mode: USART with internal Clock

Operating Mode: Blocking mode

Receive Enable: ☒

Transmit Enable: ☒

Frame Format: USART frame

Baud Rate in Hz: 115,200

Parity Mode: No Parity

Character Size: 8 Bits

Stop Bit Mode: One Stop Bit

Receive Pinout: SERCOM PAD[3] is used for data reception

Transmit Pinout: PAD[2] = TxD; PAD[3] = XCK

Enable Run in Standby: ☐

Configuration Options Help

Lab9 SERCOM - UART Tx Polling

Step 3

Pin Table

Package: TQFP48

Module	Function	PA22	PA23	PA24	PA25	GNDDIO	VDDIO	PB22	PB23	PA27	PA28	PA29	PA30	PA31	PB02	PB03
	SERCOM4_PAD3															
SERCOM5	SERCOM5_PAD0															
	SERCOM5_PAD1															
	SERCOM5_PAD2															
	SERCOM5_PAD3															
	SYSCTRL_XIN															

SERCOM5

PB22
PB23

Click

Pin Table

Package: TQFP48

Module	Function	PA22	PA23	PA24	PA25	GNDDIO	VDDIO	SERCOM5_PAD2	SERCOM5_PAD3	PA27	PA28	PA29	PA30	PA31	PB02	PB03
	SERCOM4_PAD3															
SERCOM5	SERCOM5_PAD0															
	SERCOM5_PAD1															
	SERCOM5_PAD2															
	SERCOM5_PAD3															
	SYSCTRL_XIN															

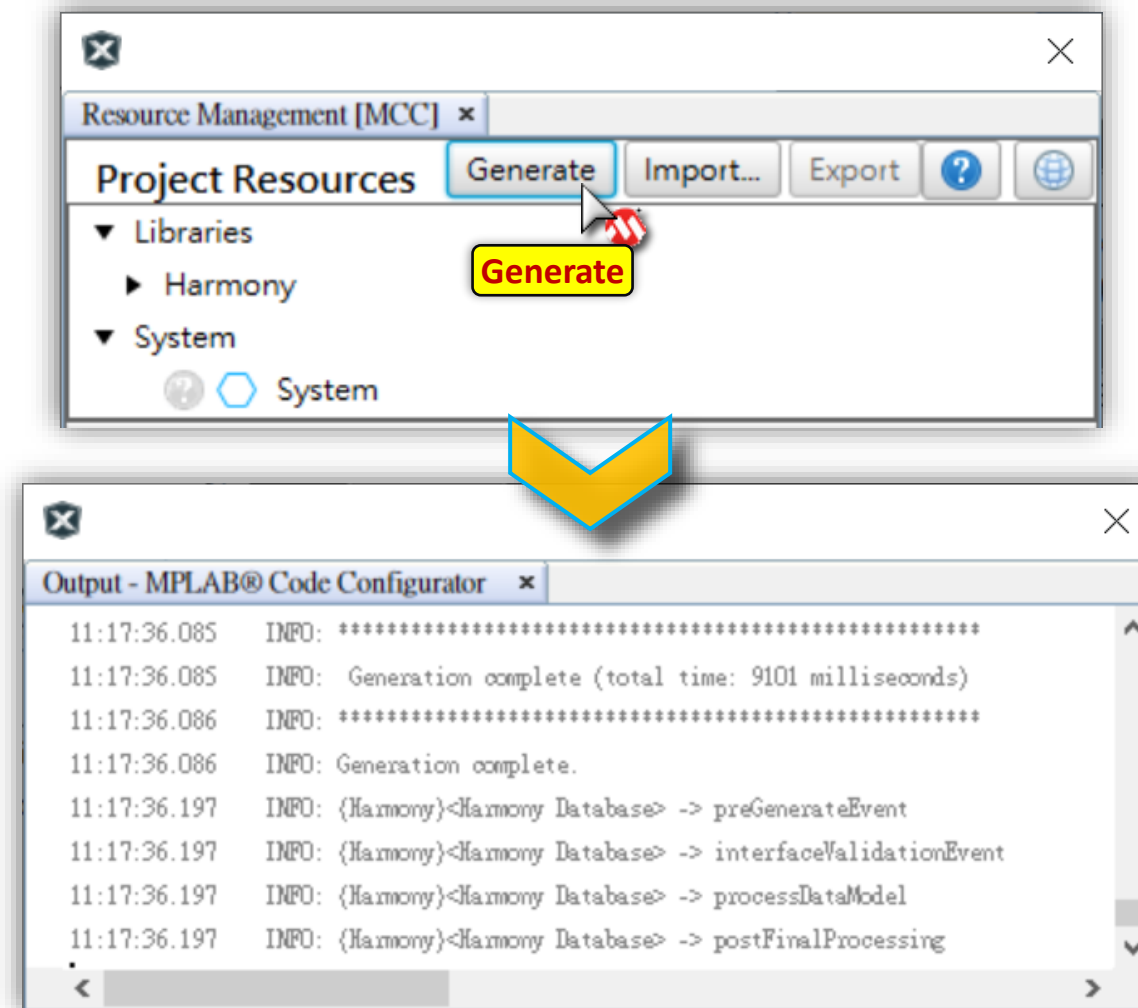
PB22 (SERCOM5_PAD2)
PB23 (SERCOM5_PAD3)

Green is Enable

Lab9 SERCOM - UART Tx Polling

Step 4

- Click "**Generate**" Button to Generate your Code.



Lab9 SERCOM - UART Tx Polling

Step 5

a Add code segment to your main loop

```
void TC3_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if( status & TC_INTFLAG_Msk )
    {
        LED1_Toggle();

        // TODO 9.01
        SERCOM5_USART_Write( "Hello world!\r\n", 14 );
    }
}

int main (void)
{
    SYS_Initialize ( NULL );

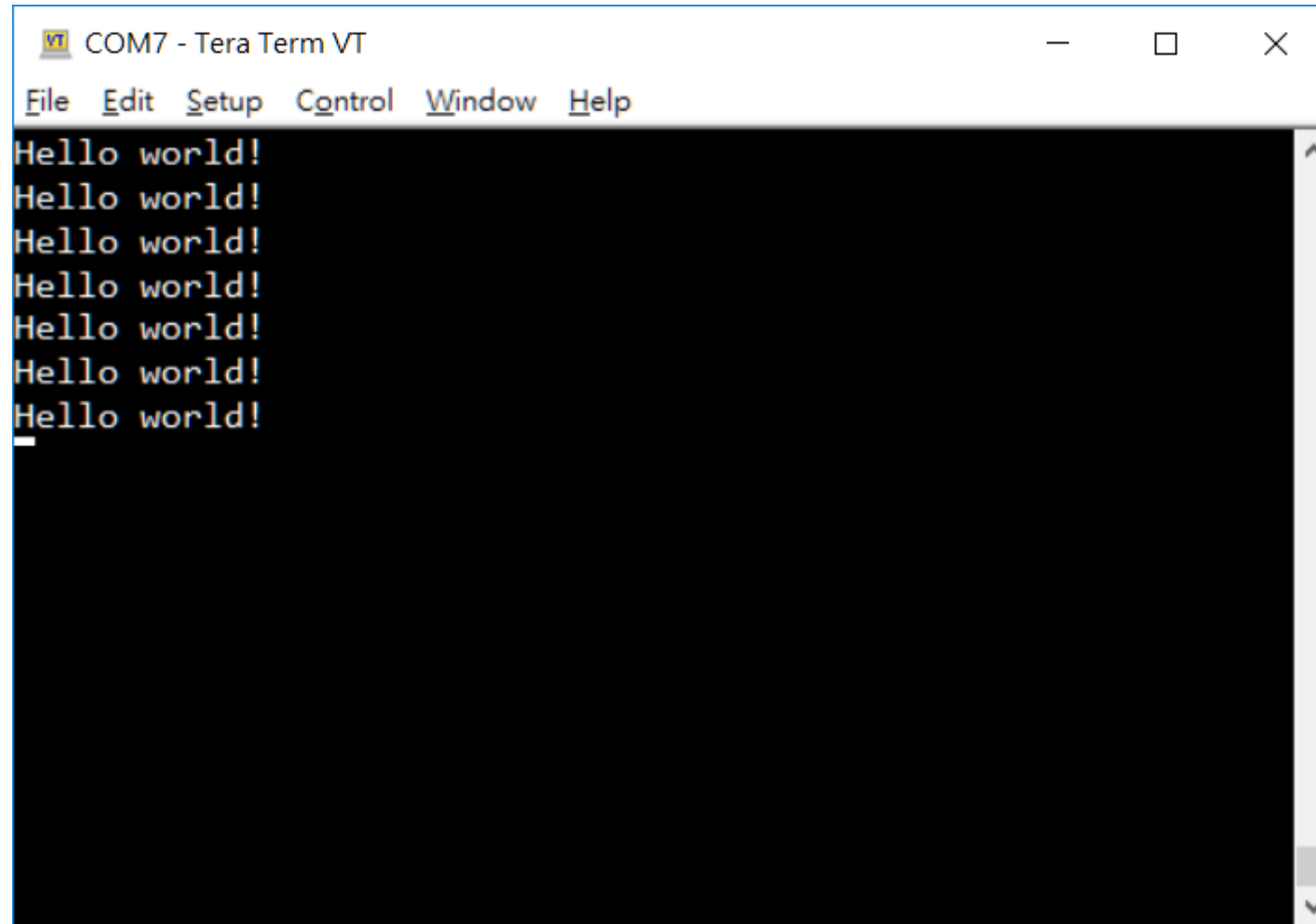
    ...

}
```

b Program firmware to target board then observe result.

Lab9 SERCOM - UART Tx Polling

Result



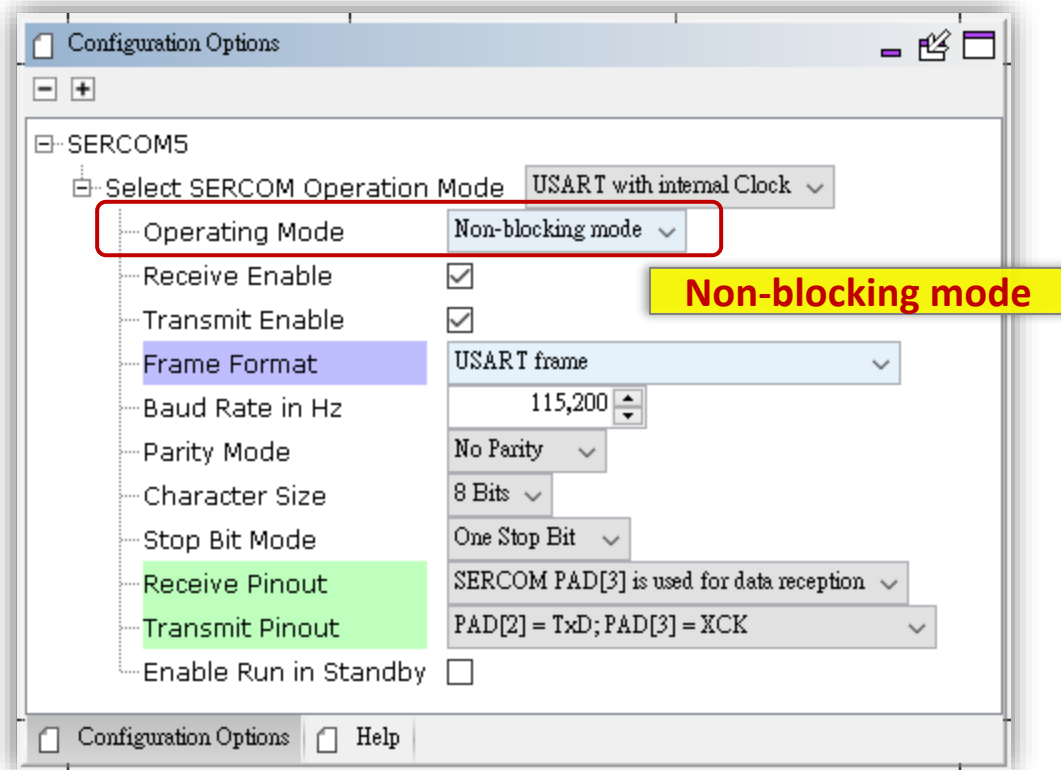
A screenshot of a Tera Term VT window titled "COM7 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main area is black with white text displaying "Hello world!" repeated seven times, one per line. A white cursor is visible at the end of the seventh line. The window has standard minimize, maximize, and close buttons in the top right corner.

```
COM7 - Tera Term VT
File Edit Setup Control Window Help
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
_
```

SERCOM USART Interrupt

Configuration Options Example

- SERCOM provide callback(interrupt) mode, also.



The screenshot shows the 'NVIC Settings' dialog. It contains a table with columns: Vector Number, Vector, Enable, Priority (0 = Highest), and Handler Name. The row for SERCOM5 (Vector 14) is highlighted with a red box, and the 'Enable' checkbox is checked. A yellow callout box labeled 'Check Enable Interrupt' points to the 'Enable' checkbox.

Vector Number	Vector	Enable	Priority (0 = Highest)	Handler Name
3	RTC (Real Time Counter)	☐	3	RTC_Handler
4	EIC (External Interrupt Controller)	☐	3	EIC_Handler
5	NVMCTRL (Non-Volatile Memory Controller)	☐	3	NVMCTRL_Handler
6	DMAC (Direct Memory Controller)	☐	3	DMAC_Handler
7	USB (Universal Serial Bus)	☐	3	USB_Handler
8	EVSYS (Event Systems)	☐	3	EVSYS_Handler
9	SERCOM0 (Serial Communication Interface 0)	☐	3	SERCOM0_Handler
10	SERCOM1 (Serial Communication Interface 1)	☐	3	SERCOM1_Handler
11	SERCOM2 (Serial Communication Interface 2)	☐	3	SERCOM2_Handler
12	SERCOM3 (Serial Communication Interface 3)	☐	3	SERCOM3_Handler
13	SERCOM4 (Serial Communication Interface 4)	☐	3	SERCOM4_Handler
14	SERCOM5 (Serial Communication Interface 5)	☒	3	SERCOM5_USART_InterruptHandler
15	TCC0 (Timer/Counter for Control Applications 0)	☐	3	TCC0_Handler
16	TCC1 (Timer/Counter for Control Applications 1)	☐	3	TCC1_Handler
17	TCC2 (Timer/Counter for Control Applications 2)	☐	3	TCC2_Handler

SERCOM USART Interrupt Callback

Interface Routines

```
void SERCOM5_USART_Initialize( void );  
bool SERCOM5_USART_SerialSetup( USART_SERIAL_SETUP * serialSetup, uint32_t clkFrequency );  
bool SERCOM5_USART_Write( void *buffer, const size_t size );  
bool SERCOM5_USART_WriteIsBusy( void );  
size_t SERCOM5_USART_WriteCountGet( void );  
void SERCOM5_USART_WriteCallbackRegister( SERCOM_USART_CALLBACK callback, uintptr_t context );  
bool SERCOM5_USART_Read( void *buffer, const size_t size );  
bool SERCOM5_USART_ReadIsBusy( void );  
size_t SERCOM5_USART_ReadCountGet( void );  
void SERCOM5_USART_ReadCallbackRegister( SERCOM_USART_CALLBACK callback, uintptr_t context );  
USART_ERROR SERCOM5_USART_ErrorGet( void );  
uint32_t SERCOM5_USART_FrequencyGet( void );
```


USART TxRx Interrupt Code Example

```
uint8_t USART5_ReceiveData[1];
void USART5_Receive( uintptr_t context )
{
    SERCOM5_USART_Write( USART5_ReceiveData, 1 );
    SERCOM5_USART_Read( USART5_ReceiveData, 1 );
}

void USART5_Transmit( uintptr_t context ) { }

int main (void)
{
    SYS_Initialize ( NULL );
    ...
    SERCOM5_USART_ReadCallbackRegister( USART5_Receive, (uintptr_t) NULL );
    SERCOM5_USART_WriteCallbackRegister( USART5_Transmit, (uintptr_t) NULL );
    SERCOM5_USART_Read( USART5_ReceiveData, 1 );
    while( true )
    { ...
        SERCOM5_USART_Write( "Hello world!\r\n", 14 );
    }
}
```

One-Shot Event

Callback Register

About String Functions

- UART generally to used for text string communication,
We can use C provided string functions to convert multi
valuables to an output string.
Here to introduce two string functions `sprintf( )` and `strlen( )`.
- `int sprintf(char *s, const char *format, ...);`
Formatting values to target char type string buffer.
* **Must include <stdio.h>**
- `size_t strlen(const char *s);`
Calculate string length, not include the end of string `0x00('\0')`.
* **Must include <string.h>**
- For example

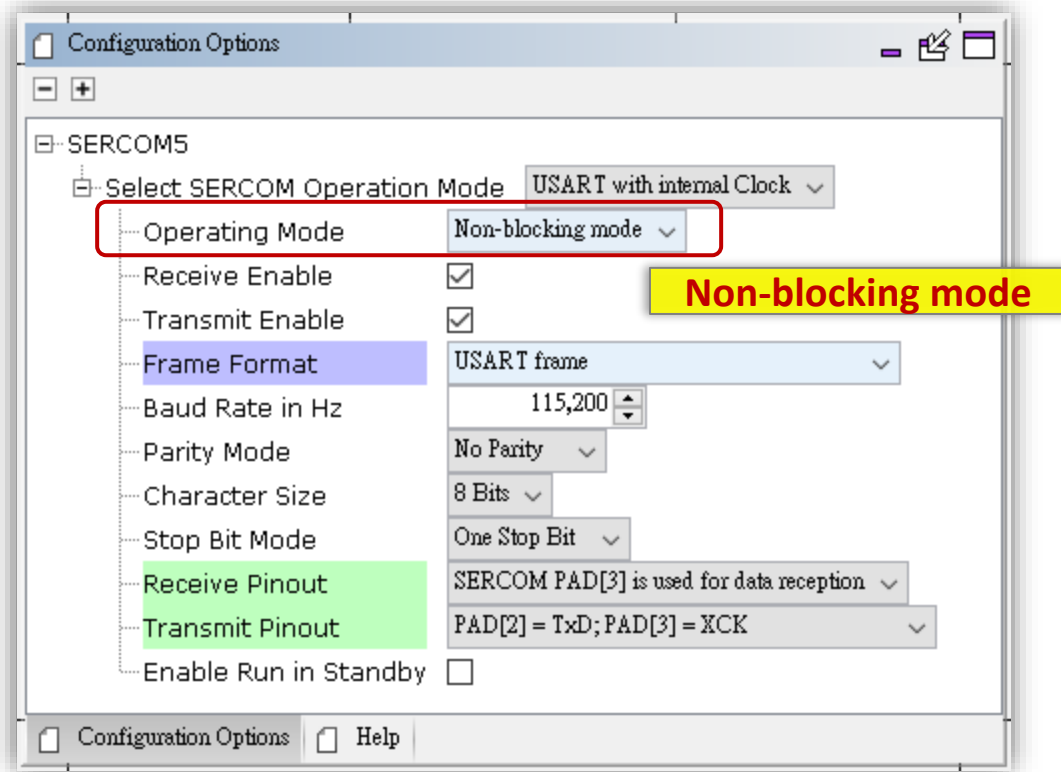
```
uint8_t USARTRTxBuffer[100];  
uint16_t USART5_ReceiveData;  
sprintf((char *) USARTRTxBuffer, "Received Data : %1c\r\n", USART5_ReceiveData);  
SERCOM5_USART_Write( USARTRTxBuffer, strlen((char *)USARTRTxBuffer));
```

Lab10 SERCOM - UART TxRx Interrupt Callback

- Try to enable SERCOM-USART **Interrupt** Callback driver.
- Modify polling transmit function to asynchronous function.
- Add new USART receive function base on interrupt callback function.
- Try to receive character from PC and loopback to TeraTerm as string format "**Received Data : x**".
- SERCOM - UART setting same as Lab9
- **Let's go!**

Lab10 SERCOM - UART TxRx Interrupt Callback

Step 1



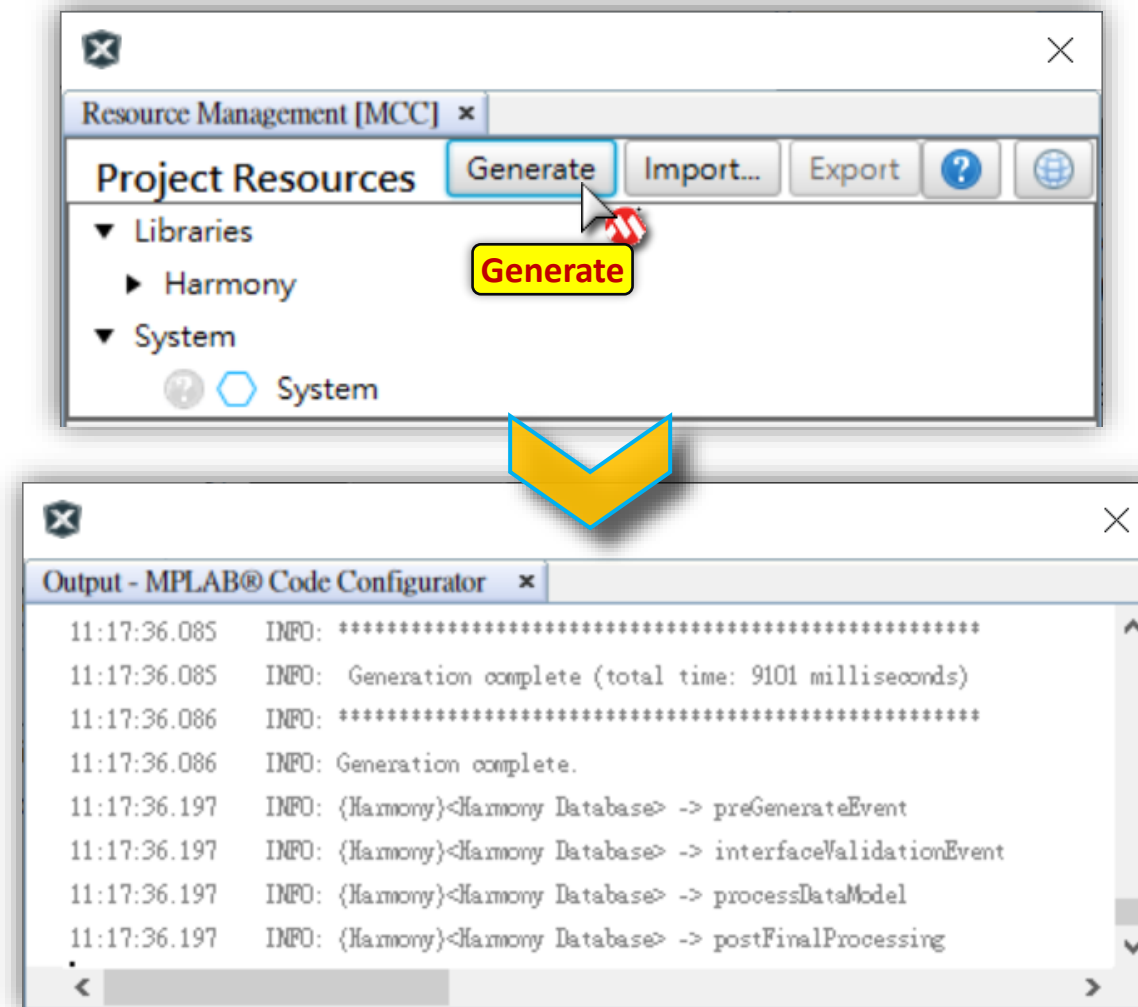
The screenshot shows the 'NVIC Settings' dialog. The table below lists the interrupt settings for various peripherals. The row for SERCOM5 (Vector Number 14) is highlighted with a red box, and a yellow callout box labeled 'Check Enable Interrupt' points to the 'Enable' checkbox, which is checked.

Vector Number	Vector	Enable	Priority (0 = Highest)	Handler Name
3	RTC (Real Time Counter)	☐	3	RTC_Handler
4	EIC (External Interrupt Controller)	☐	3	EIC_Handler
5	NVMCTRL (Non-Volatile Memory Controller)	☐	3	NVMCTRL_Handler
6	DMAC (Direct Memory Controller)	☐	3	DMAC_Handler
7	USB (Universal Serial Bus)	☐	3	USB_Handler
8	EVSYS (Event Systems)	☐	3	EVSYS_Handler
9	SERCOM0 (Serial Communication Interface 0)	☐	3	SERCOM0_Handler
10	SERCOM1 (Serial Communication Interface 1)	☐	3	SERCOM1_Handler
11	SERCOM2 (Serial Communication Interface 2)	☐	3	SERCOM2_Handler
12	SERCOM3 (Serial Communication Interface 3)	☐	3	SERCOM3_Handler
13	SERCOM4 (Serial Communication Interface 4)	☐	3	SERCOM4_Handler
14	SERCOM5 (Serial Communication Interface 5)	☒	3	SERCOM5_USART_InterruptHandler
15	TCC0 (Timer/Counter for Control Applications 0)	☐	3	TCC0_Handler
16	TCC1 (Timer/Counter for Control Applications 1)	☐	3	TCC1_Handler
17	TCC2 (Timer/Counter for Control Applications 2)	☐	3	TCC2_Handler

Lab10 SERCOM - UART TxRx Interrupt Callback

Step 2

- Click "**Generate**" Button to Generate your Code.



Lab10 SERCOM - UART TxRx Interrupt Callback

Step 3

a Create UART TxRx Callback function, related buffer and

```
// TODO 10.01
#include <stdio.h>
#include <string.h>

// TODO 10.02
uint8_t USARTRTxBuffer[100];
uint8_t USART5_ReceiveData[1];
volatile uint8_t USART5_IsReceived = 0;
volatile uint8_t USART5_IsTransmitted = 1;

void USART5_Transmit( uintptr_t context )
{
    USART5_IsTransmitted = 1;
}

void USART5_Receive( uintptr_t context )
{
    USART5_IsReceived = 1;
}

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

Lab10 SERCOM - UART TxRx Interrupt Callback

Step 4

a Modify TC3 Callback for UART Tx

```
void TC3_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if( status & TC_INTFLAG_OVF_Msk )
    {
        LED1_Toggle();

        // TODO 10.03
        sprintf((char *) USARTRTxBuffer, "Hello world!\r\n" );
        SERCOM5_USART_Write( USARTRTxBuffer, strlen((char*)USARTRTxBuffer) );
    }
}

int main (void)
{
    SYS_Initialize ( NULL );

    ...
}
```

b Program firmware to target board then observe result.

Lab10 SERCOM - UART TxRx Interrupt Callback

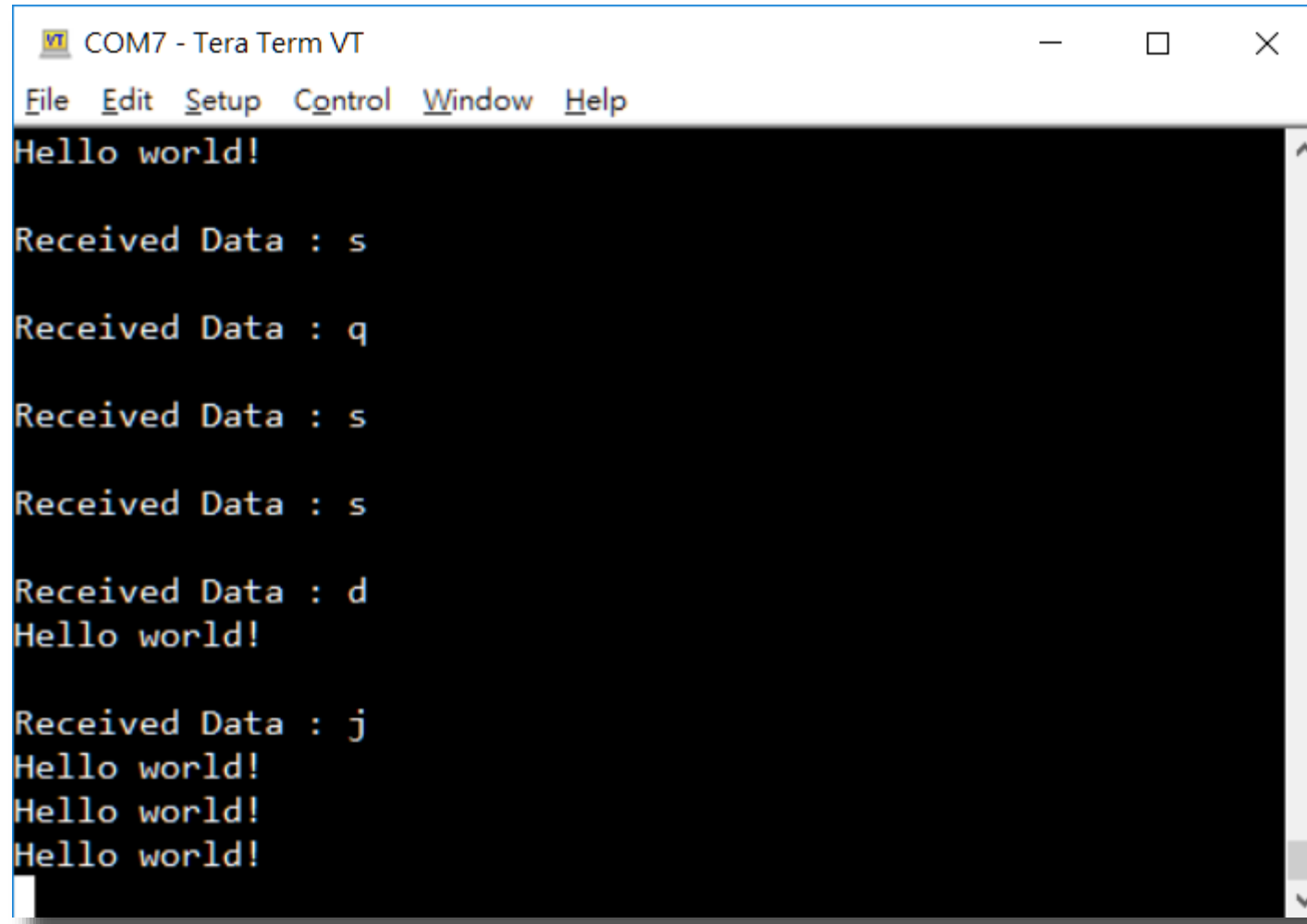
Step 5

a Add code segment to your main loop

```
int main (void)
{
    ...
    // TODO 10.04
    SERCOM5_USART_ReadCallbackRegister( USART5_Receive, (uintptr_t )NULL );
    SERCOM5_USART_WriteCallbackRegister( USART5_Transmit, (uintptr_t )NULL );
    SERCOM5_USART_Read( USART5_ReceiveData, 1 );
    while( true )
    {
        ...
        // TODO 10.05
        if( USART5_IsReceived )
        {
            USART5_IsReceived = 0;
            sprintf((char *) USARTRTxBuffer,
                    "\r\nReceived Data : %1c\r\n", USART5_ReceiveData[0] );
            SERCOM5_USART_Write( USARTRTxBuffer, strlen((char*)USARTRTxBuffer) );
            while( SERCOM5_USART_WriteIsBusy() );
            SERCOM5_USART_Read( USART5_ReceiveData, 1 );
        }
    }
}
```


Lab10 SERCOM - UART TxRx Interrupt Callback

Result



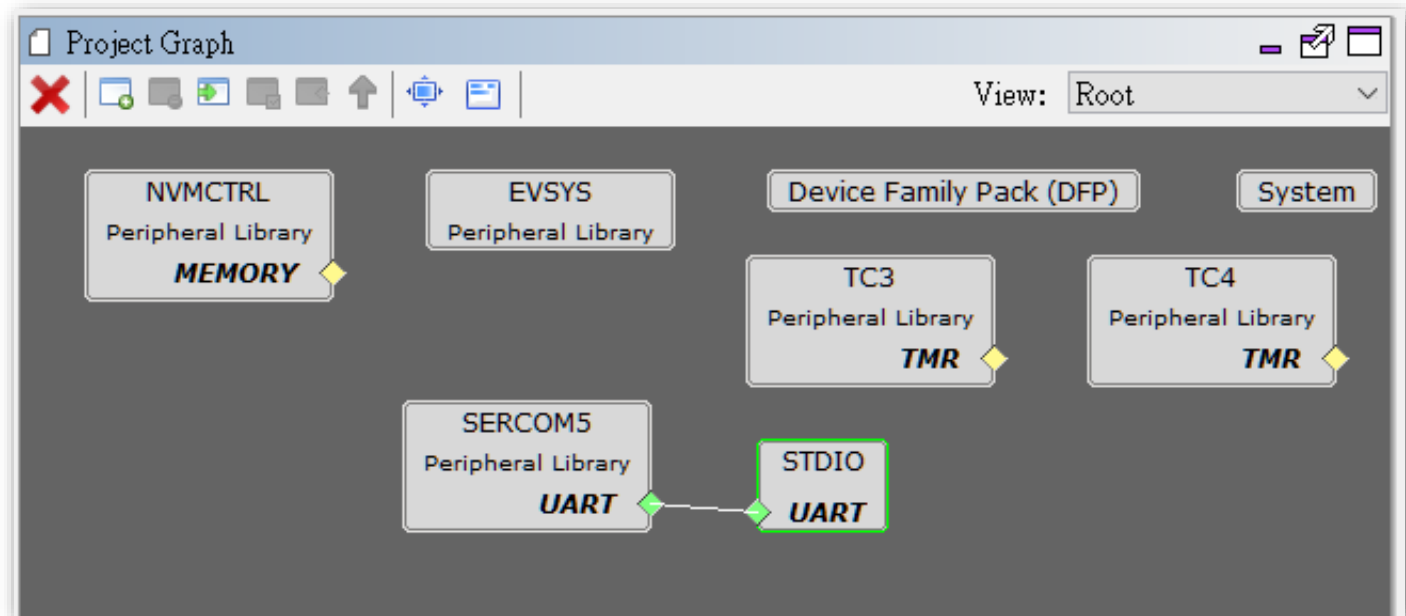
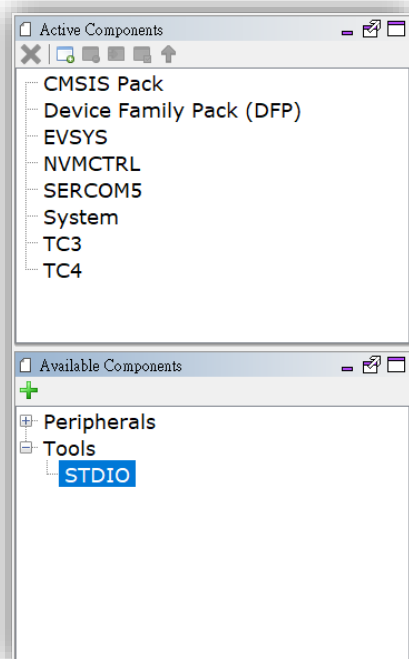
The screenshot shows a Tera Term VT window titled "COM7 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main text area displays the following sequence of messages:

```
Hello world!  
  
Received Data : s  
  
Received Data : q  
  
Received Data : s  
  
Received Data : s  
  
Received Data : d  
Hello world!  
  
Received Data : j  
Hello world!  
Hello world!  
Hello world!
```

The text is displayed in a monospaced font on a black background. A vertical scrollbar is visible on the right side of the text area.

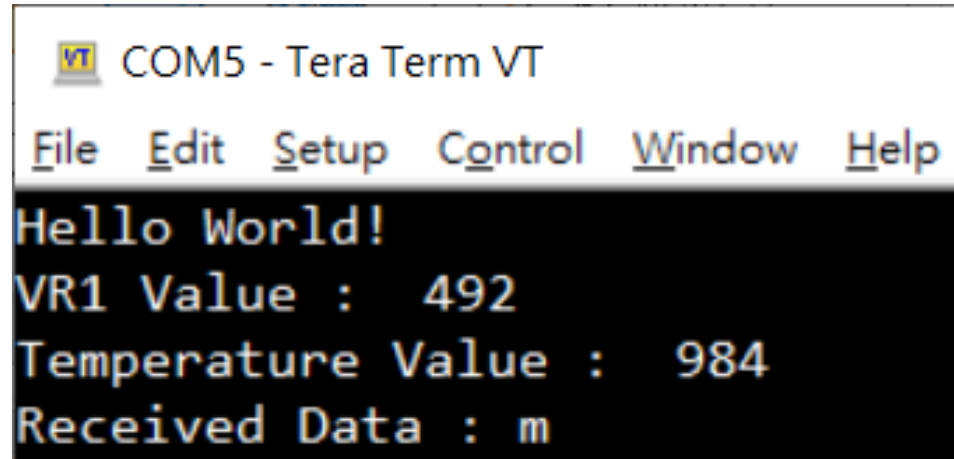
The STDIO limitation

- Harmony support the standard input/output (STDIO) could be easy to use printf() function to output UART message. But the mechanism of STDIO module is based on polling mode (blocking mode). It will conflict when selected SERCOM-UART Non-blocking mode and try to use printf().
- Let 's try implement our myprintf() function to use interrupt method but have convenient use as the C printf() function.



VT100 Console Control

- Use VT100 command to control Tera Term UART output as

A screenshot of a Tera Term window titled 'COM5 - Tera Term VT'. The window has a menu bar with 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The main display area shows the following text: 'Hello World!', 'VR1 Value : 492', 'Temperature Value : 984', and 'Received Data : m'. The text is displayed in a monospaced font with a yellow background and black text.

```
VT COM5 - Tera Term VT
File Edit Setup Control Window Help
Hello World!
VR1 Value : 492
Temperature Value : 984
Received Data : m
```

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

- **Let's go!**

Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

- Try add **myprintf** code for user code.
- After done this, you could use **myprintf()** function to instead of USART interface routines.
- SERCOM - UART setting same as Lab10
- **Let's go!**

Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

Step 1

a Add software flag control of TC3 and TC4.

```
// TODO 11.01
#include <stdarg.h>

// TODO 11.02
#define SYS_CONSOLE_PRINT_BUFFER_SIZE 200
static char consolePrintBuffer[SYS_CONSOLE_PRINT_BUFFER_SIZE];
volatile uint8_t TC3_HasExpired= 0;
volatile uint8_t TC4_HasExpired= 0;

...
void TC3_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if( status & TC_INTFLAG_OVF_Msk )
    {
        // TODO 11.03
        TC3_HasExpired= 1;
    }
}
void TC4_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if( status & TC_INTFLAG_OVF_Msk )
    {
        // TODO 11.04
        TC4_HasExpired= 1;
    }
}
```

Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

Step 2

a Add code segment to user code.

```
void TC4_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    ...
}

// TODO 11.05
void myprintf(const char *format, ...)
{
    size_t len = 0;
    va_list args = {0};

    va_start(args, format);
    len = vsnprintf(consolePrintBuffer, SYS_CONSOLE_PRINT_BUFFER_SIZE, format , args);
    va_end(args);

    if ((len > 0) && (len < SYS_CONSOLE_PRINT_BUFFER_SIZE))
    {
        consolePrintBuffer[len] = '\0';
        SERCOM5_USART_Write(consolePrintBuffer, len);
        while (SERCOM5_USART_WriteIsBusy());
    }
}
```

Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

Step 3

a Add code segment to your main while loop.

```
int main ( void )
{
    ...
    // TODO 11.06
    myprintf("\033[2J");
    myprintf("\033[?251");
    while( true )
    {
        // TODO 11.07
        if (TC3_HasExpired)
        {
            TC3_HasExpired= 0;
            LED1_Toggle();
            myprintf( "\033[1;1HHello World!" );
        }

        if (TC4_HasExpired)
        {
            TC4_HasExpired= 0;
            LED2_Toggle();
        }
    }
}
```

Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

Step 4

a Add code segment to your main while loop.

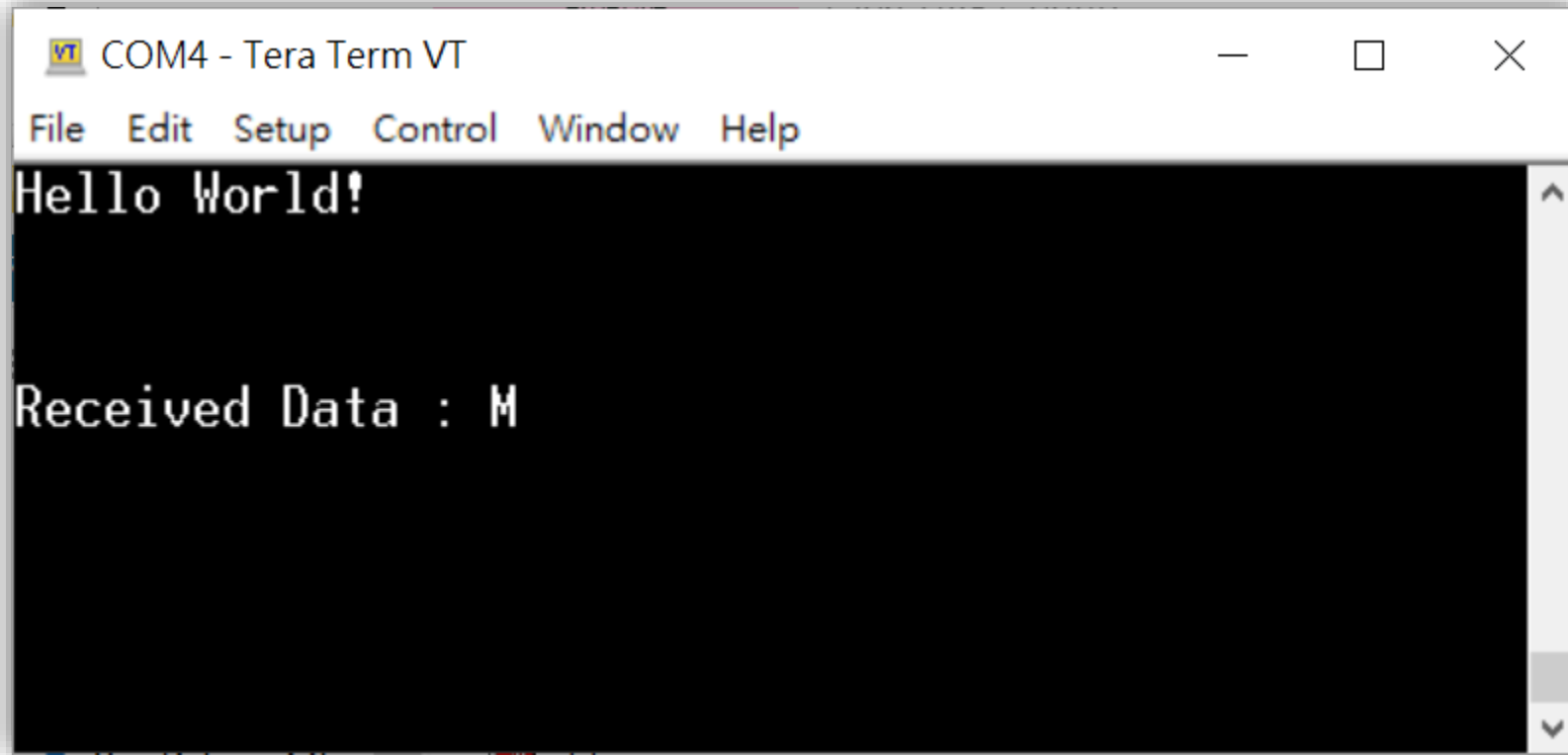
```
int main ( void )
{
    ...
    while( true )
    {
        if (TC4_HasExpired)
        {
            TC4_HasExpired= 0;
            LED2_Toggle();
        }

        if( USART5_IsReceived )
        {
            USART5_IsReceived = 0;

            // TODO 11.08
            myprintf("\033[6;1HReceived Data : %1c", USART5_ReceiveData[0] );
            SERCOM5_USART_Read( USART5_ReceiveData, 1 );
        }
    }
}
```


Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

Result



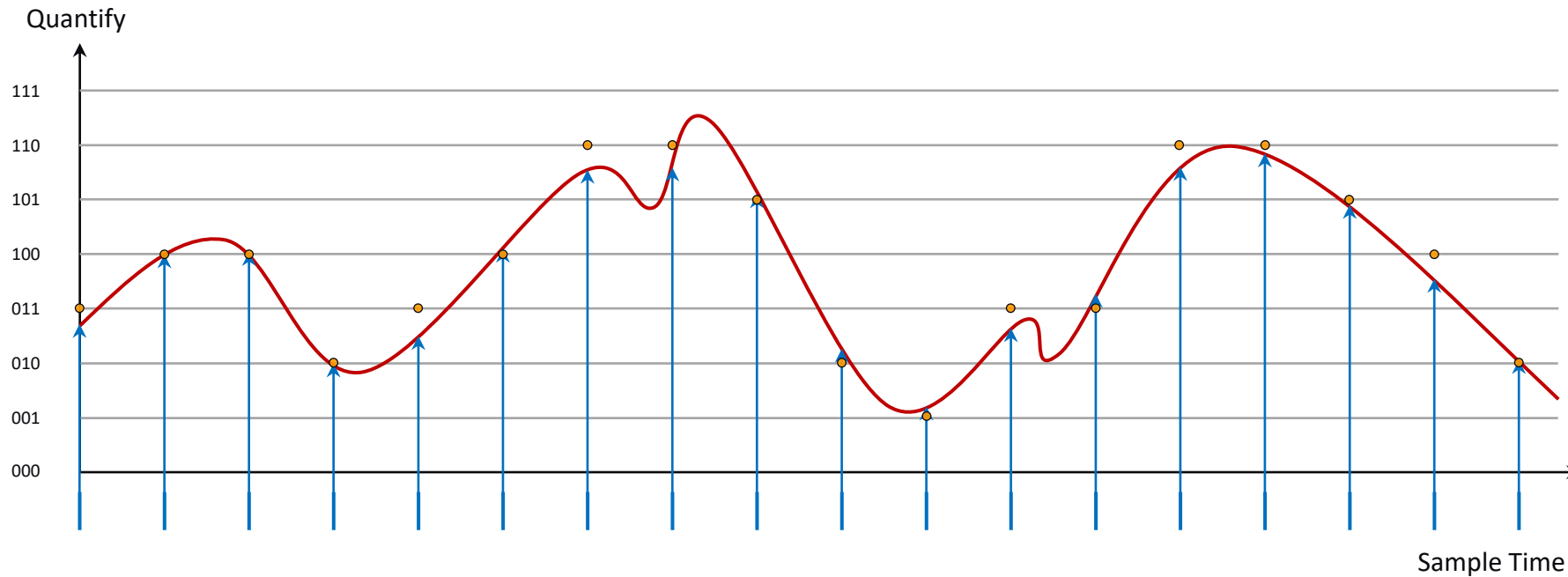
A screenshot of a Tera Term VT window titled "COM4 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main text area is black with white text. It displays "Hello World!" on the first line and "Received Data : M" on the second line. A vertical scrollbar is visible on the right side of the text area.

```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Hello World!
Received Data : M
```

High Speed ADC Architecture

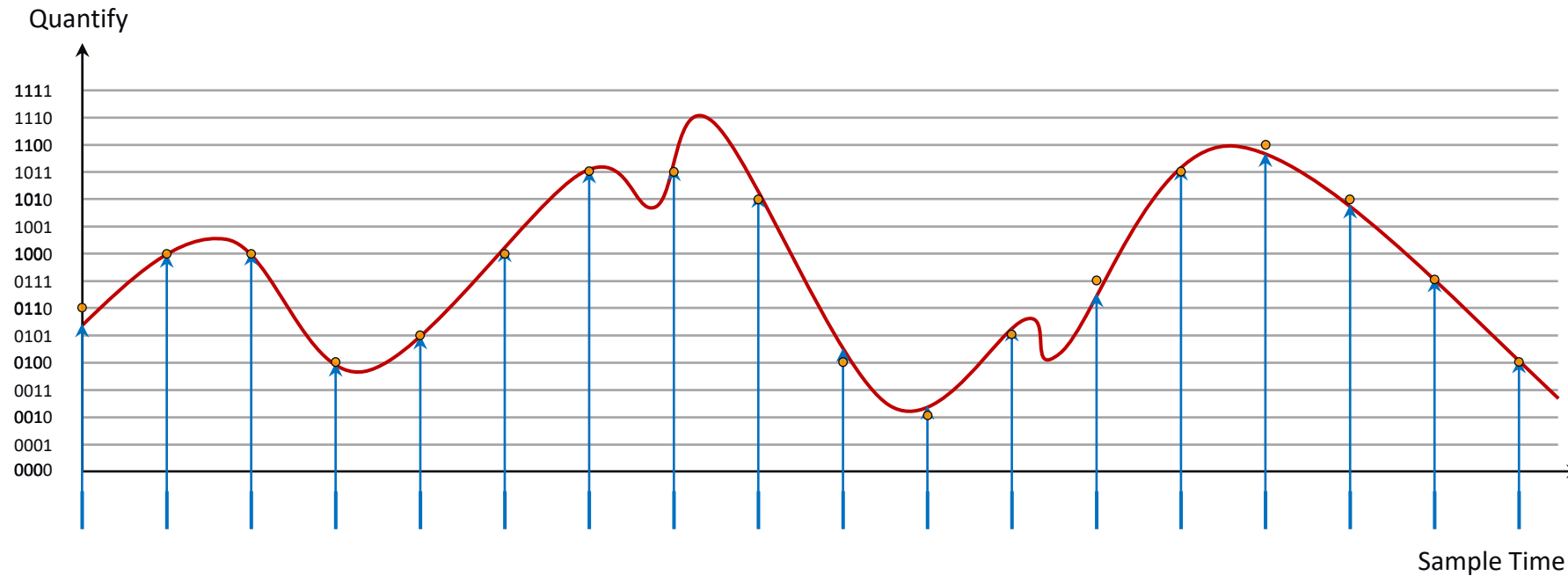
What's ADC ?

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



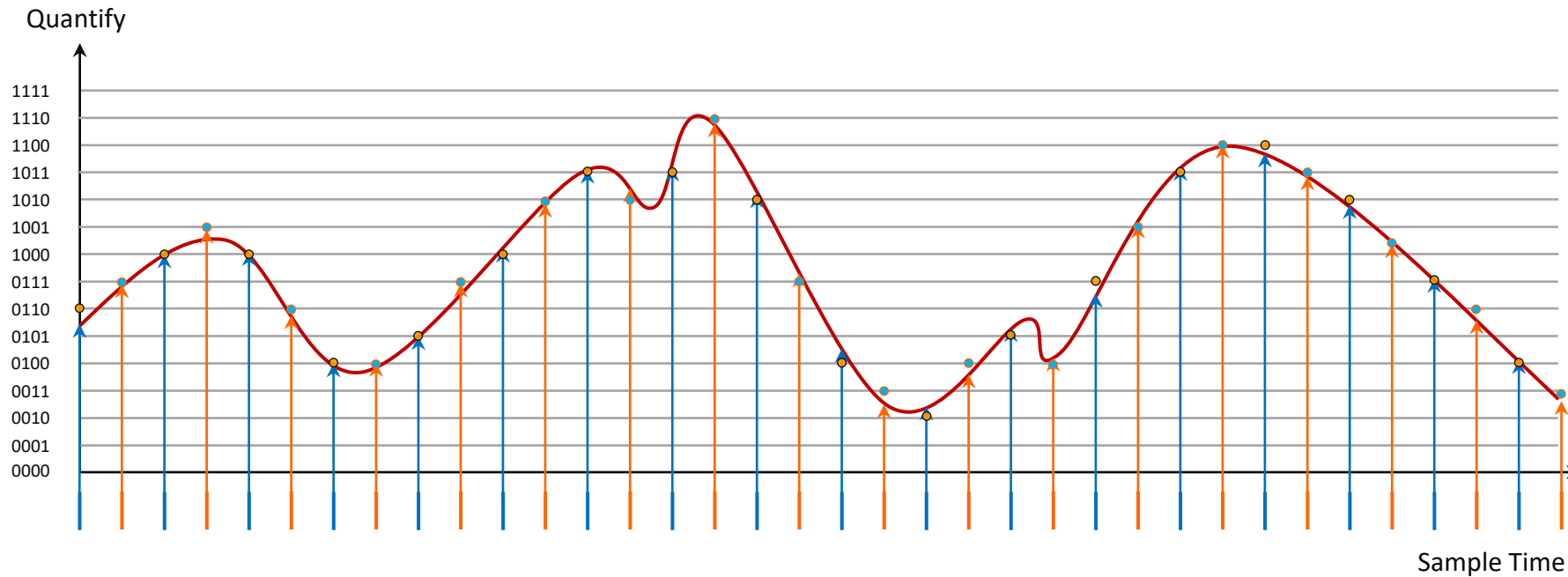
Resolution

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



Sample Rate

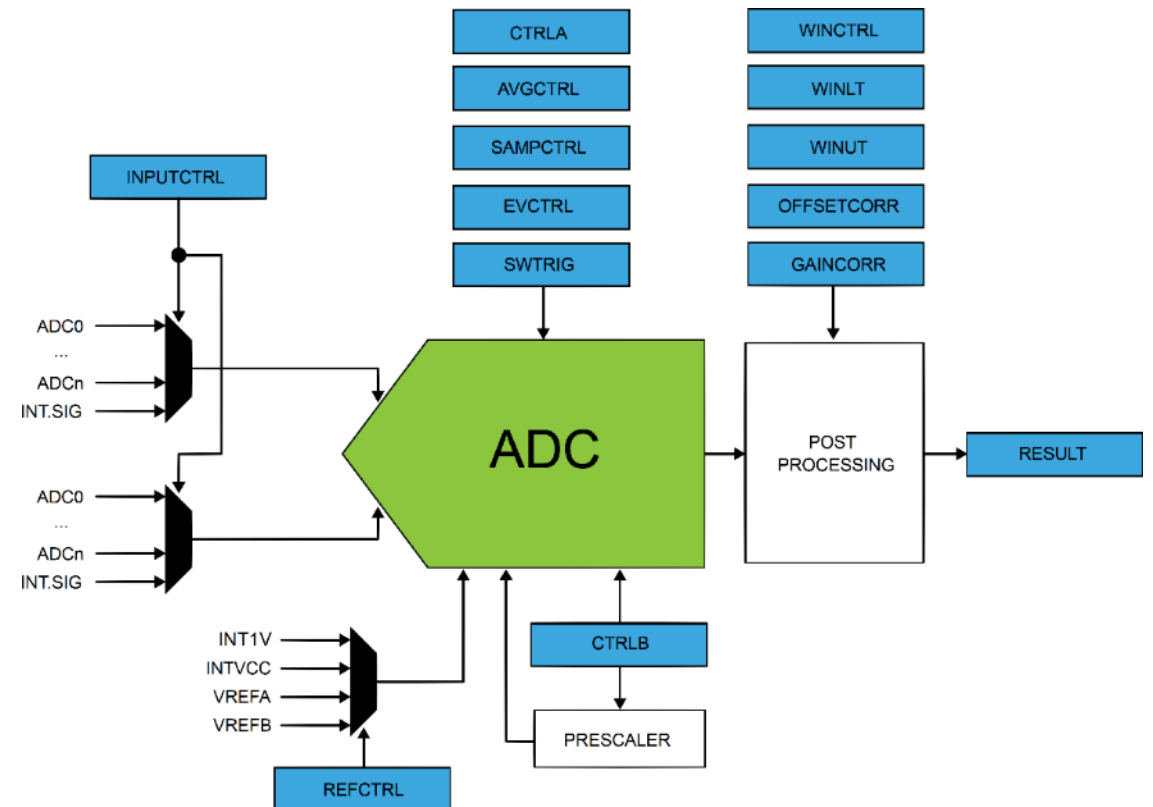
- The analog signal is required to define the rate at which new digital values are sampled from the analog signal.
- This faithful reproduction is only possible if the sampling rate is higher than twice the highest frequency of the signal. (wiki)



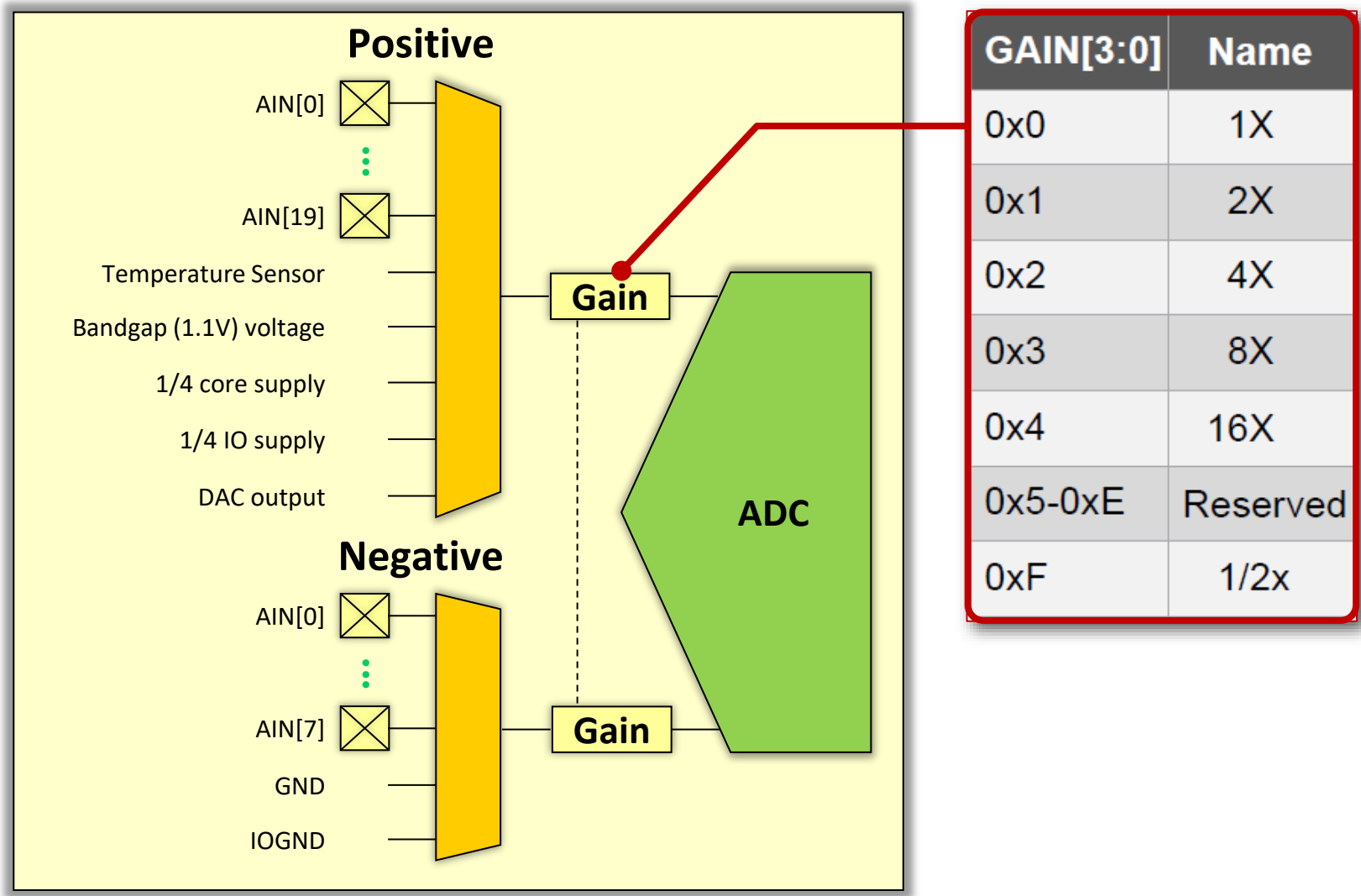
ADC Block Diagram

- Up to 32 analog input, 8, 10 or 12 bits resolution.
- Differential and single-ended inputs.
- 1/2x to 16x gain. Up to 350ksps.
- Support Event-triggered and pin scan.
- Support averaging and oversampling.
- External or Internal reference voltage.
- Oversample accumulation average and window monitor feature.

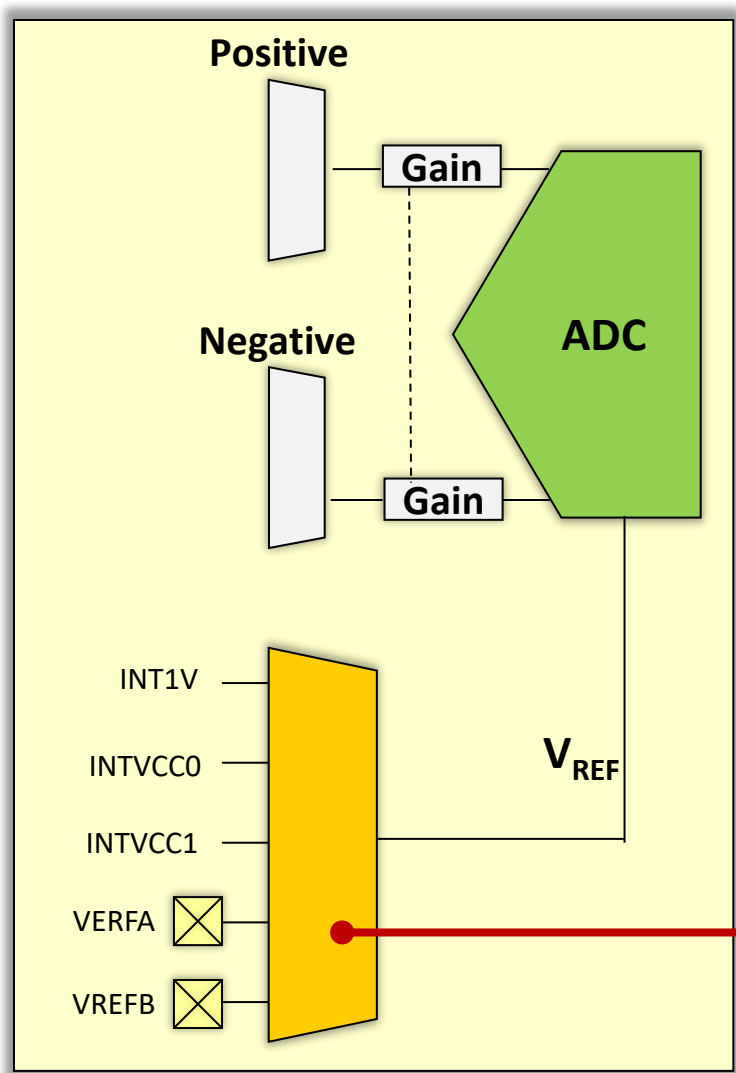
$$Result_{AD} = \left(\frac{AN_+ - V_{Ref-}}{V_{Ref+} - V_{Ref-}} \times 2^n \right) - 1$$



ADC Channel Selection & Gain



ADC Reference Voltage Selection

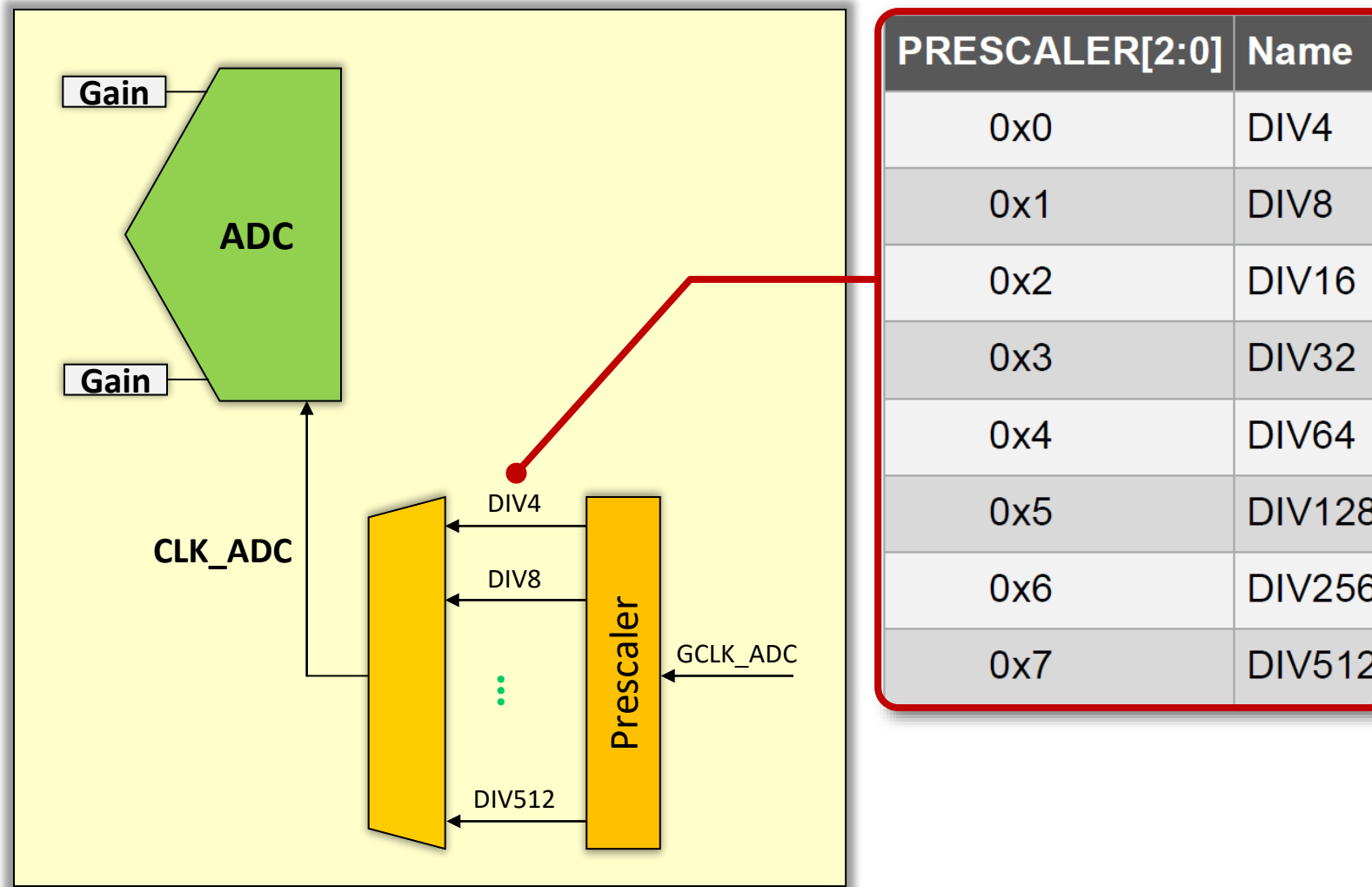


VREFA and VREFB range
 $1.0V \sim (V_{DDANA} - 0.6V)$
Conversion range:
– **Single-ended mode**
 $0V \sim V_{REF}/Gain$
– **Differential mode**
 $-V_{REF}/Gain \sim V_{REF}/Gain$

REFSEL[3:0]	Name	Description
0x0	INT1V	1.0V voltage reference
0x1	INTVCC0	1/1.48 VDDANA
0x2	INTVCC1	1/2 VDDANA (only for VDDANA > 2.0V)
0x3	VREFA	External reference
0x4	VREFB	External reference
0x5-0xF		Reserved

VDDANA = 3.3V
½ VDDANA = 1.65V

ADC Prescaler



ADC Trigger Source

- **There are three way to be started data conversion.**

- **Software Trigger**

One shot conversion control via software.

- **Free Run**

A free-running mode can be used to continuously convert an input channel. When using free-running mode the first conversion must be started, while subsequent conversions will start automatically at the end of previous conversions.

- **HW Event Trigger**

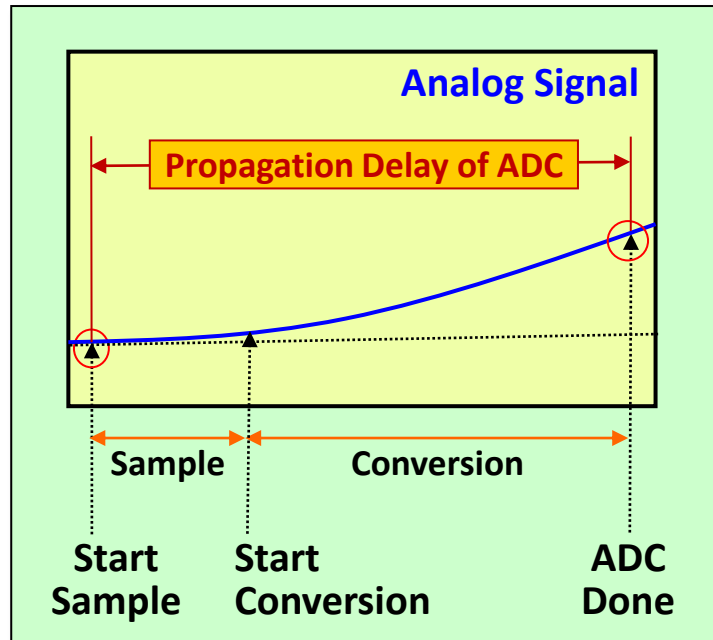
Through Event System peripheral to automatically conversion.

ADC Characteristics

- Maximum Sampling Rate :

Single-shot : $2.1\text{M}/\text{Round}(6+0.5) = 300\text{k samples/second.}$

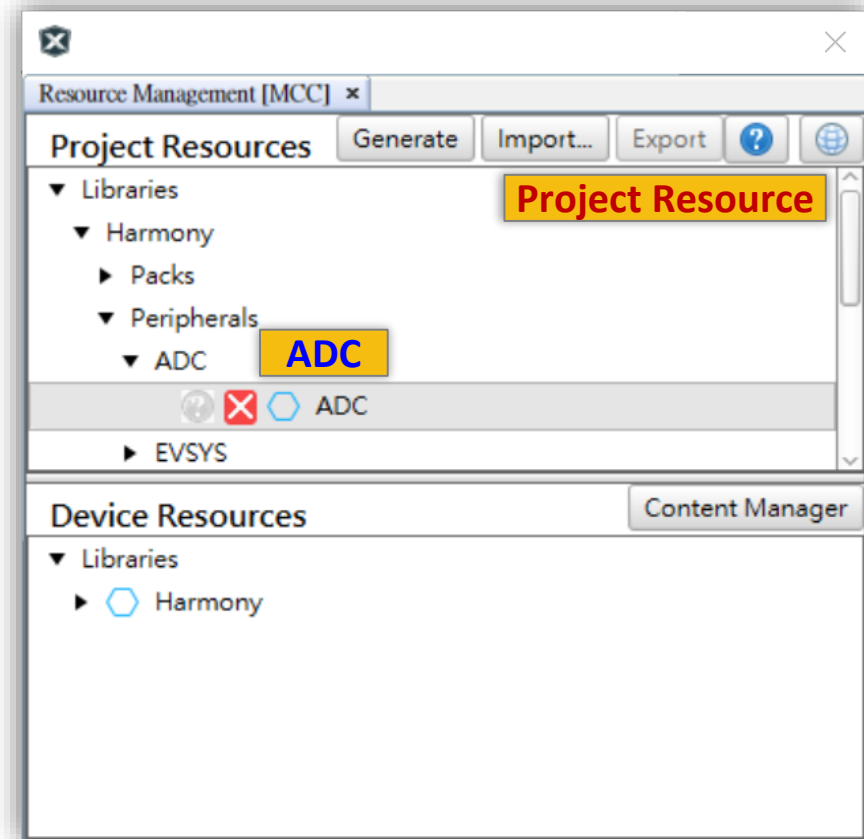
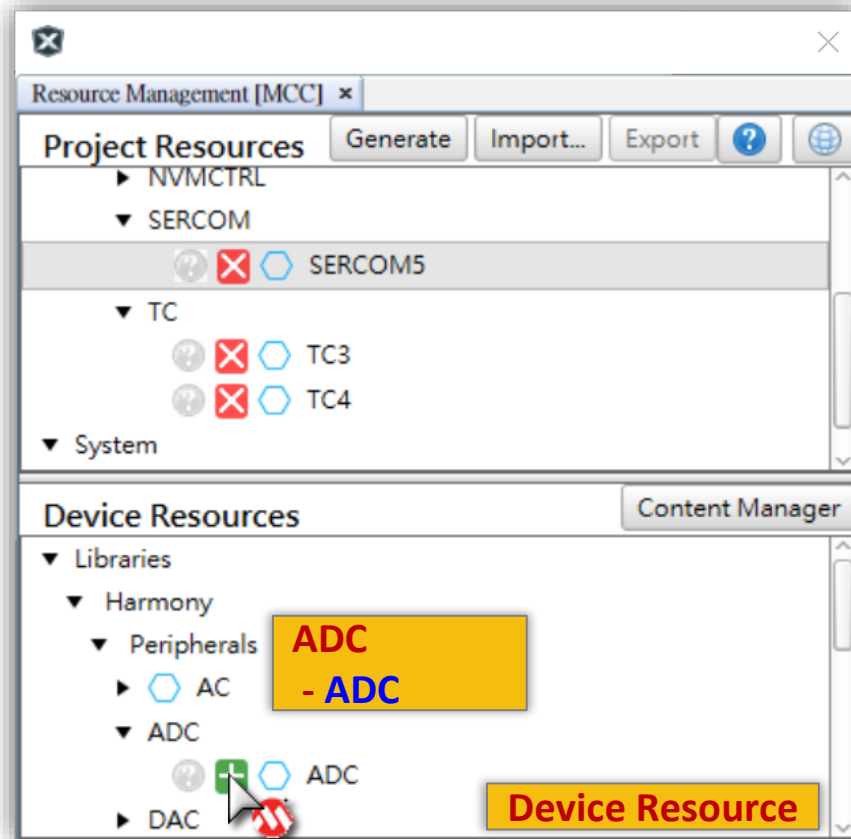
Free-running : $2.1\text{M}/6 = \text{up to } 350\text{k samples/second.}$



Symbol	Parameter	Conditions	Min.	Max.	Units
RES	Resolution		8	12	bits
f _{CLK_ADC}	ADC Clock frequency		30	2100	kHz
	Sample rate	Single shot	5	300	ksps
		Free running	5	350	ksps
	Sampling time		0.5	-	cycles
	Conversion time	1x Gain	6	-	cycles

Add ADC Function using MCC Harmony

- Find **ADC** component in Device Resource window.
- Double click **ADC** to add to Project Resource window.



ADC Polling

Configuration Options Example

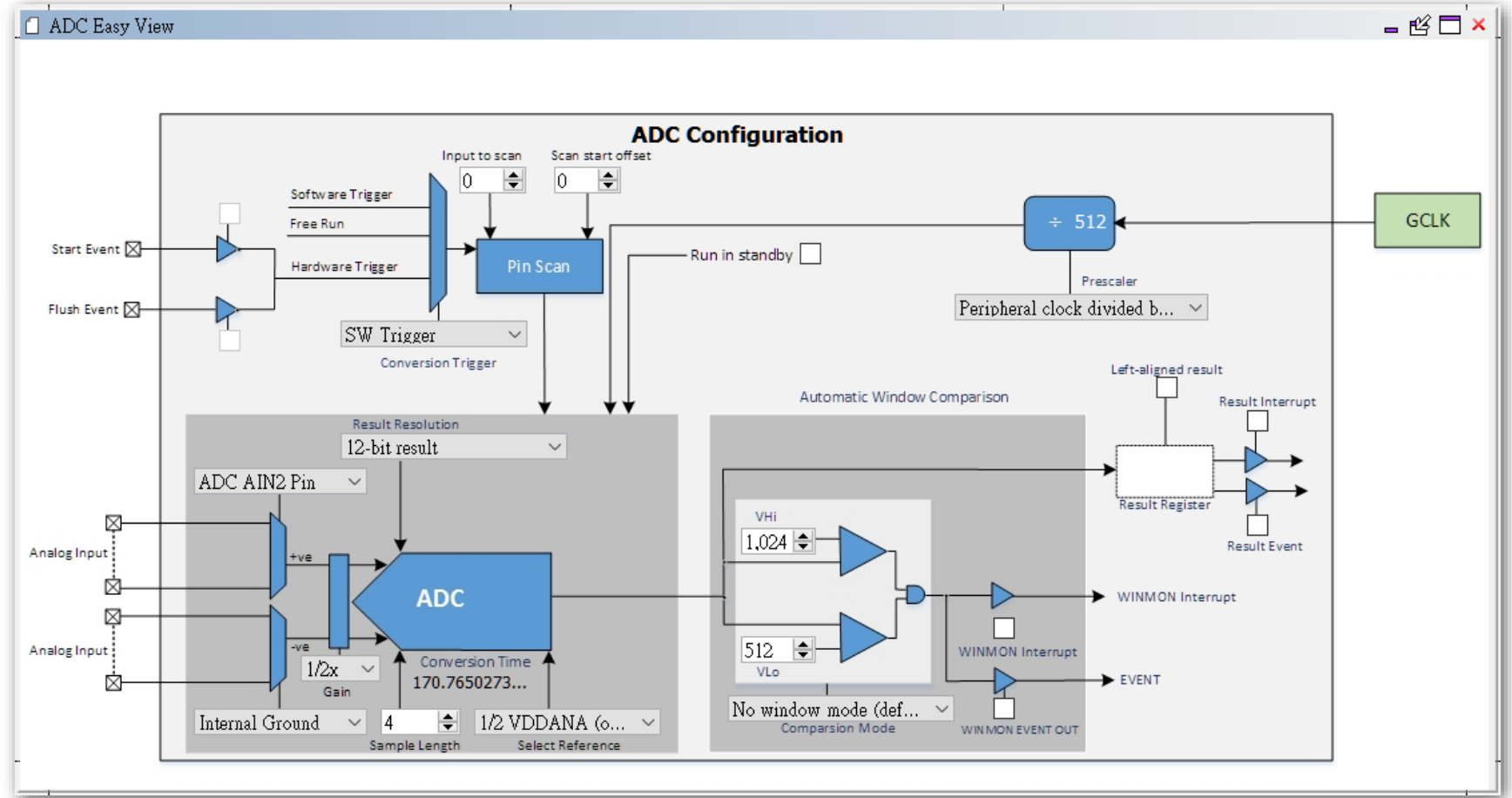
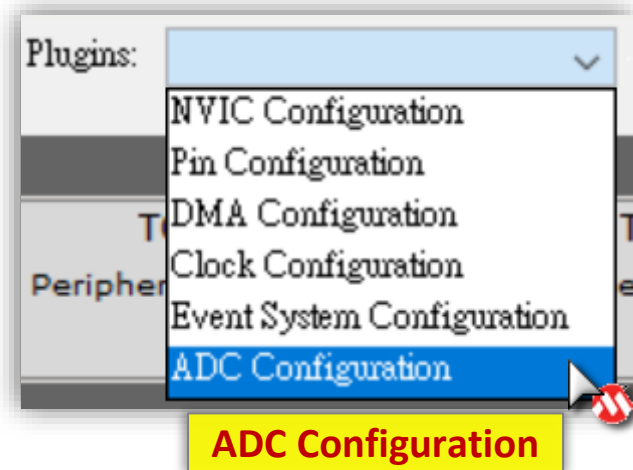
The screenshot shows a software configuration window titled "Configuration Options" with a tree view on the left and configuration fields on the right. The tree view includes sections for ADC, Channel Configuration, Result Configuration, Window Mode Configuration, and Sleep Mode Configuration. Several options are highlighted in green.

Configuration Options

- ADC**
 - Select Prescaler: Peripheral clock divided by 512
 - Select Sample Length (half ADC clock cycles): 4
 - **** Conversion Time is 106.666666667 uS ****
 - Select Gain: 1/2x
 - Select Reference: 1/2 VDDANA (only for VDDANA > 2.0V)
 - Select Conversion Trigger: SW Trigger
- Channel Configuration**
 - Select Positive Input: ADC AIN2 Pin
 - Select Negative Input: Internal Ground
 - Number of inputs to scan: 0
- Result Configuration**
 - Select Result Resolution: 12-bit result
 - Left Aligned Result: ☐
 - Enable Result Ready Interrupt: ☐
 - Enable Result Ready Event Out: ☐
- Window Mode Configuration**
 - Select Window Monitor Mode: No window mode (default)
- Sleep Mode Configuration**
 - Run During Standby: ☐

ADC Polling

Easy View Configuration Example



ADC Polling

PINMUX Configuration Example

- Analog pins will all belong to **PINMUX function group B**.
- For example : **PB08** as **AIN[2]** (Analog input Channel 2)
PB09 as **AIN[3]** (Analog input Channel 3)

Pin ⁽¹⁾			I/O Pin	Supply	A	REF	B ²⁾⁽³⁾				C	D	E	F	G	H
SAMD21E	SAMD21G	SAMD21J					ADC	AC	PTC	DAC						
	7	11	PB08	VDDANA	EXTINT[8]		AIN[2]		Y[14]			SERCOM4/ PAD[0]	TC4/WO[0]	TCC3/ WO[6]		
	8		PB09	VDDANA	EXTINT[9]		AIN[3]					SERCOM4/ PAD[1]	TC4/WO[1]	TCC3/ WO[7]		

Pin Table

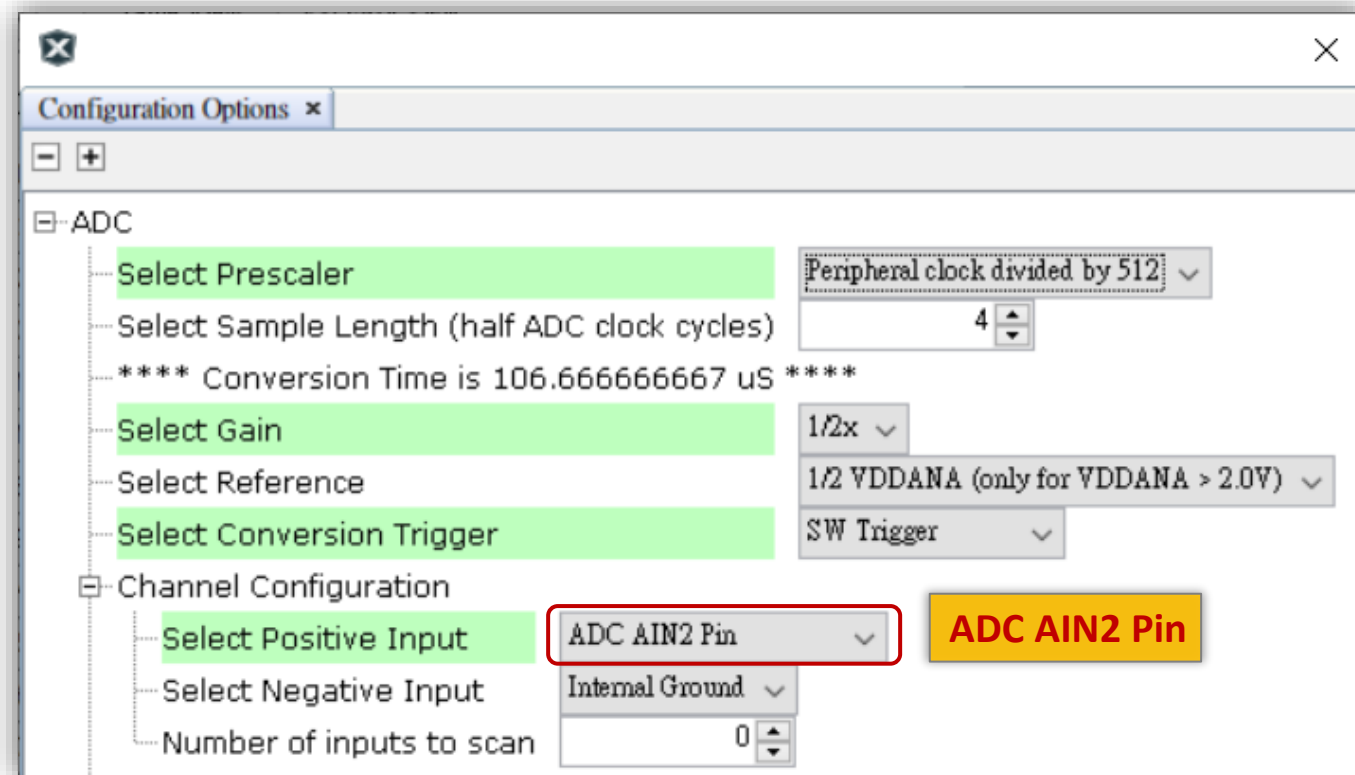
Package: TQFP48

		PA00	PA01	PA02	PA03	GNDAN	VDDAN	ADC_AI	ADC_AI	PB08 (ADC_AIN2) PB09 (ADC_AIN3)				
Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13
ADC	ADC_AIN18													
	ADC_AIN19													
	ADC_AIN2													
	ADC_AIN3													
	ADC_AIN4/VREFP													
	ADC_AIN5													
	ADC_AIN6													

ADC Polling

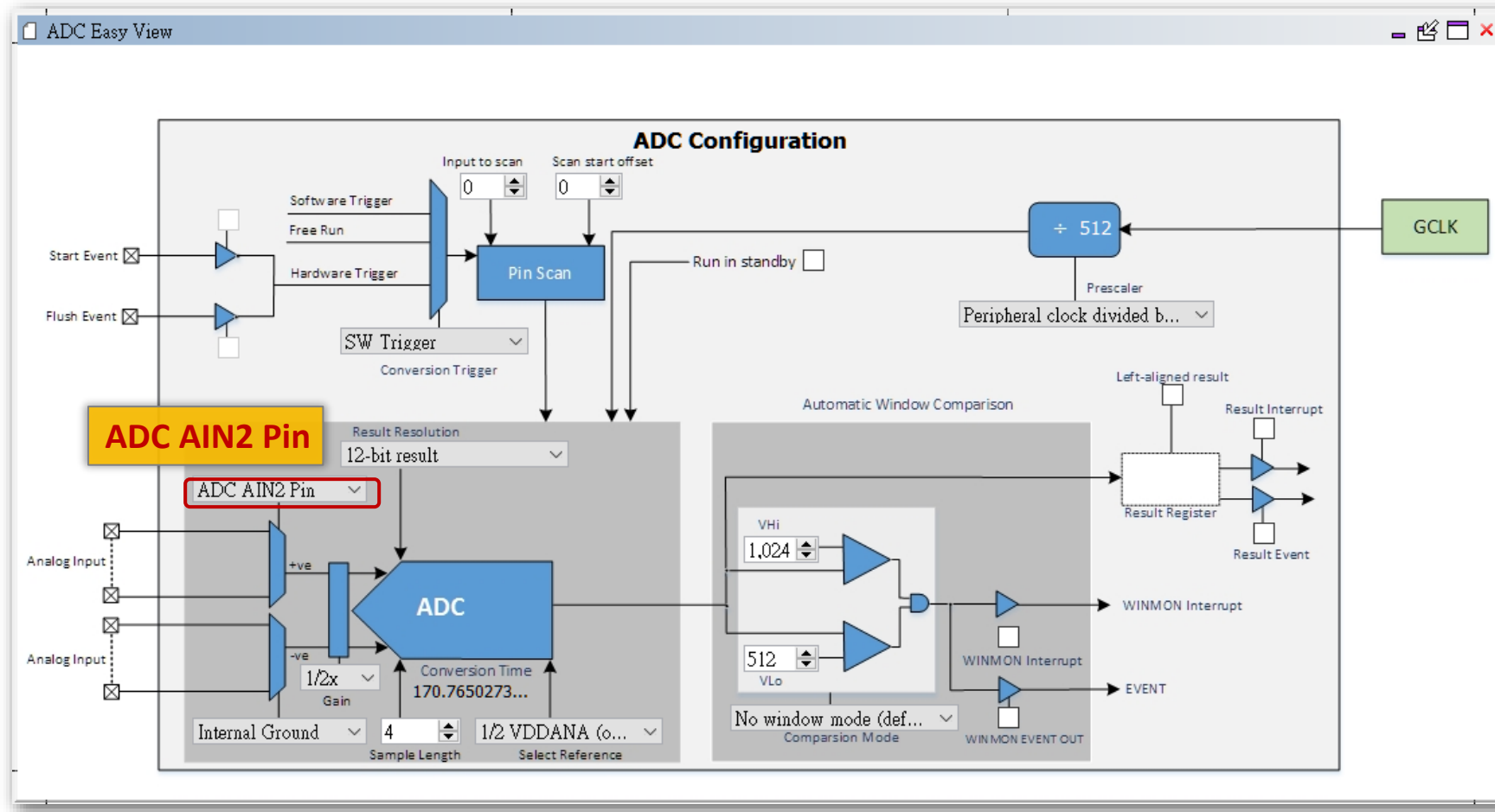
PINMUX Configuration Example

- Remember assign **Positive Input** pin in ADC module.
- For example : **ADC AIN2 Pin** as Positive Input Pin (**PB08**)



ADC Polling

Easy View PINMUX Configuration Example



ADC Polling

Interface Routines

```
void ADC_Initialize( void );  
void ADC_Enable( void );  
void ADC_Disable( void );  
void ADC_ChannelSelect( ADC_POSINPUT positiveInput, ADC_NEGINPUT negativeInput );  
void ADC_ConversionStart( void );  
uint16_t ADC_ConversionResultGet( void );  
bool ADC_ConversionStatusGet( void );  
void ADC_ComparisonWindowSet(uint16_t low_threshold, uint16_t high_threshold);  
void ADC_WindowModeSet(ADC_WINMODE mode);
```

ADC Polling

Software Trigger Code Example

```
...  
  
int main (void)  
{  
    SYS_Initialize ( NULL );  
    ...  
    ADC_Enable();  
    while( true )  
    {  
        ...  
        ADC_ConversionStart();  
        while( !ADC_ConversionStatusGet() );  
        myprintf( "%d\r\n", ADC_ConversionResultGet() );  
    }  
}
```

ADC Factory Calibration

- ADC Factory calibration value will auto load and applicate in MCC Harmony generated ADC initial function.

plib_adc.c

```
void ADC_Initialize( void )
{
    ...
    /* Write linearity calibration and bias calibration */
    ADC_REGS->ADC_CALIB =
        (uint32_t)( ADC_CALIB_LINEARITY_CAL( adc_linearity0 | (adc_linearity1 << 5) ) ) |
        ADC_CALIB_BIAS_CAL( ( ( (*(uint64_t*)OTP4_ADDR + 1) & ADC_BIASCAL_Msk ) >> ADC_BIASCAL_POS) );
}
```

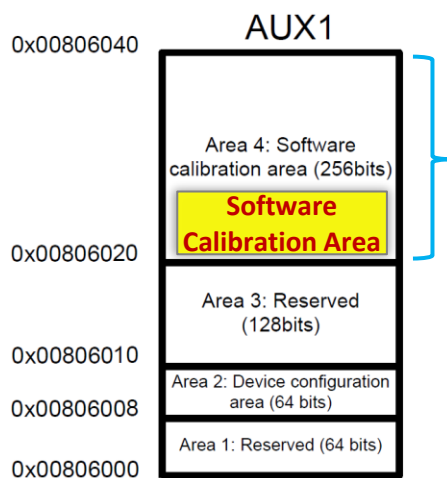


Table 10-5. NVM Software Calibration Area Mapping

Bit Position	Name	Description
34:27	ADC LINEARITY ADC_LINEARITY	ADC Linearity Calibration. Should be written to ADC CALIB register.
37:35	ADC BIASCAL ADC_BIASCAL	ADC Bias Calibration. Should be written to ADC CALIB register.

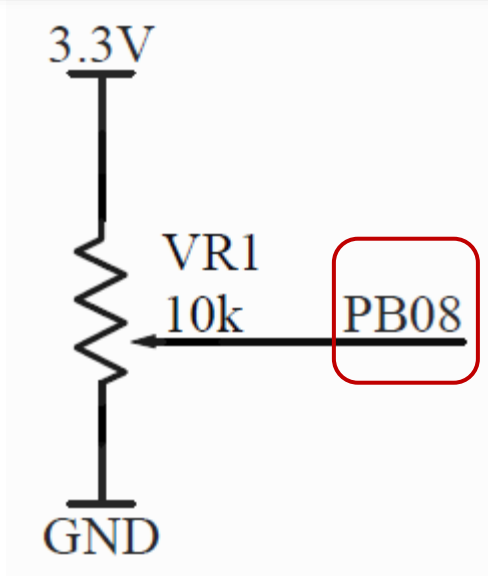
Lab12 ADC Polling Software Trigger

- Try to add ADC function to project.
- Initial ADC and get the conversion value from **VR1 (AIN[2])** with one second period, then use UART to output to PC.
- ADC set to **Software Trigger, 12 bits resolution, Single-End** mode.

• **Let's go!**

Lab12 ADC Polling Software Trigger

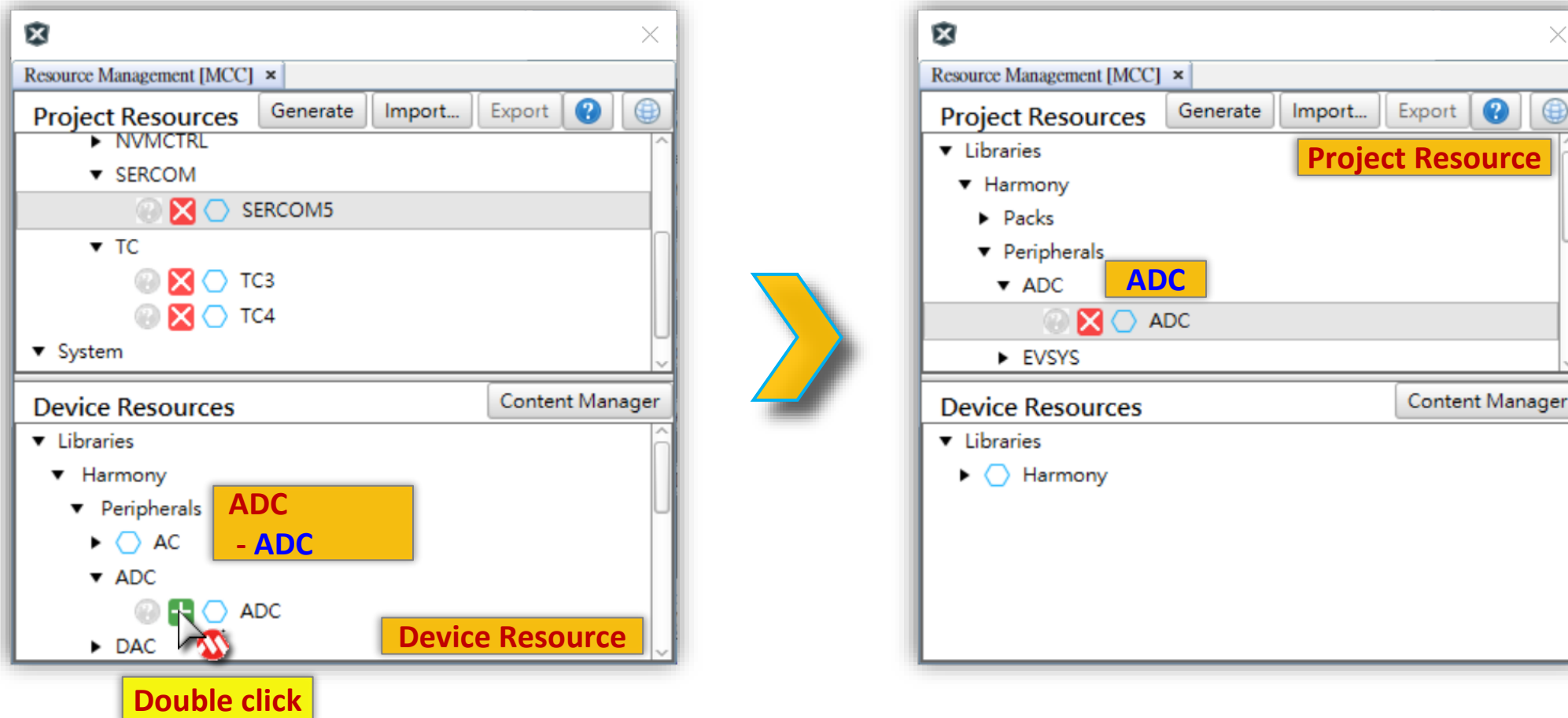
Pin ⁽¹⁾			I/O Pin	Supply	A	B ⁽²⁾⁽³⁾				
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC
	7	11	PB08	VDDANA	EXTINT[8]		AIN[2]		Y[14]	
	8	12	PB09	VDDANA	EXTINT[9]		AIN[3]		Y[15]	



Lab12 ADC Polling Software Trigger

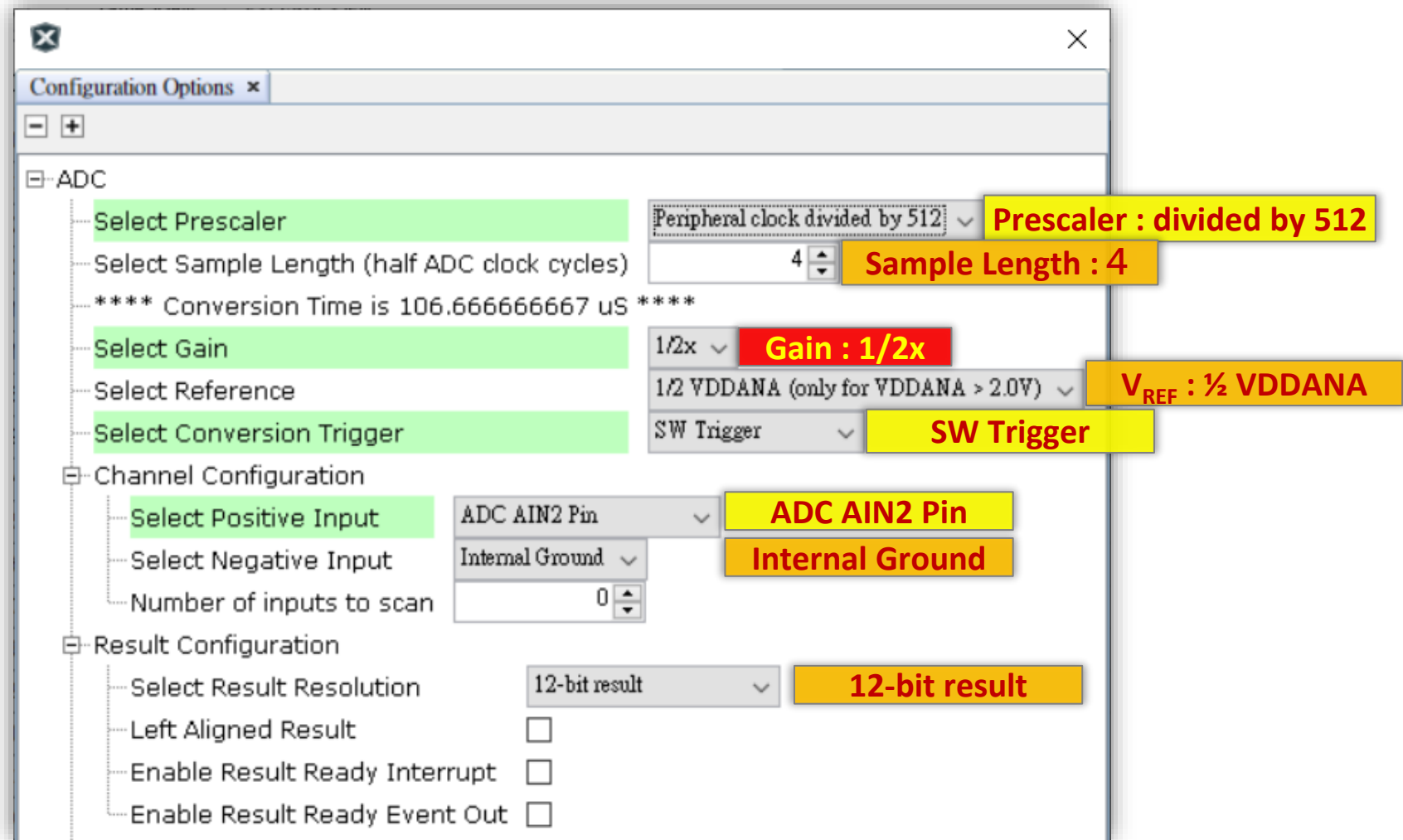
Step 1

- Find **ADC** component in Device Resource window.
- Double click **ADC** to add to Project Resource window.



Lab12 ADC Polling Software Trigger

Step 2 (Use Configuration Options)

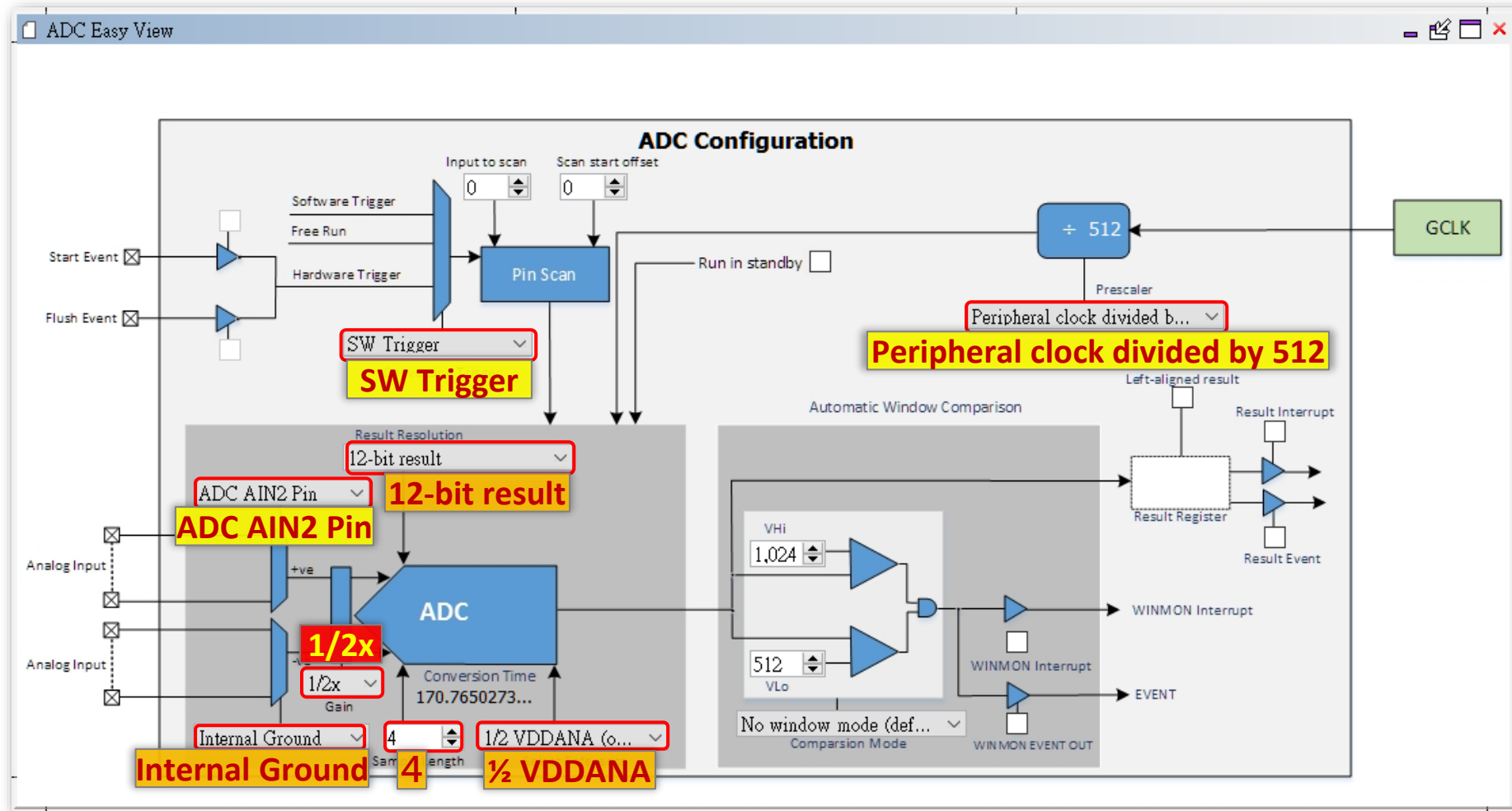


The screenshot shows the 'Configuration Options' dialog for the ADC module. The settings are as follows:

- Select Prescaler:** Peripheral clock divided by 512. **Prescaler : divided by 512**
- Select Sample Length (half ADC clock cycles):** 4. **Sample Length : 4**
- **** Conversion Time is 106.666666667 uS ******
- Select Gain:** 1/2x. **Gain : 1/2x**
- Select Reference:** 1/2 VDDANA (only for VDDANA > 2.0V). **V_{REF} : ½ VDDANA**
- Select Conversion Trigger:** SW Trigger. **SW Trigger**
- Channel Configuration**
 - Select Positive Input:** ADC AIN2 Pin. **ADC AIN2 Pin**
 - Select Negative Input:** Internal Ground. **Internal Ground**
 - Number of inputs to scan:** 0
- Result Configuration**
 - Select Result Resolution:** 12-bit result. **12-bit result**
 - Left Aligned Result:** ☐
 - Enable Result Ready Interrupt:** ☐
 - Enable Result Ready Event Out:** ☐

Lab12 ADC Polling Software Trigger

Step 2 (Use Easy View)



Lab12 ADC Polling Software Trigger

Step 3

Pin Table

Package: TQFP48

Module	Function	PA00	PA01	PA02	PA03	GNDAN	VDDAN	PB08	PB09	PA04	PA05	PA06	PA07	PA08
ADC	ADC_AIN18													
	ADC_AIN19													
	ADC_AIN2													
	ADC_AIN4/VREFP													
	ADC_AIN5													
	ADC_AIN6													

ADC

ADC_AIN2

Click

PB08

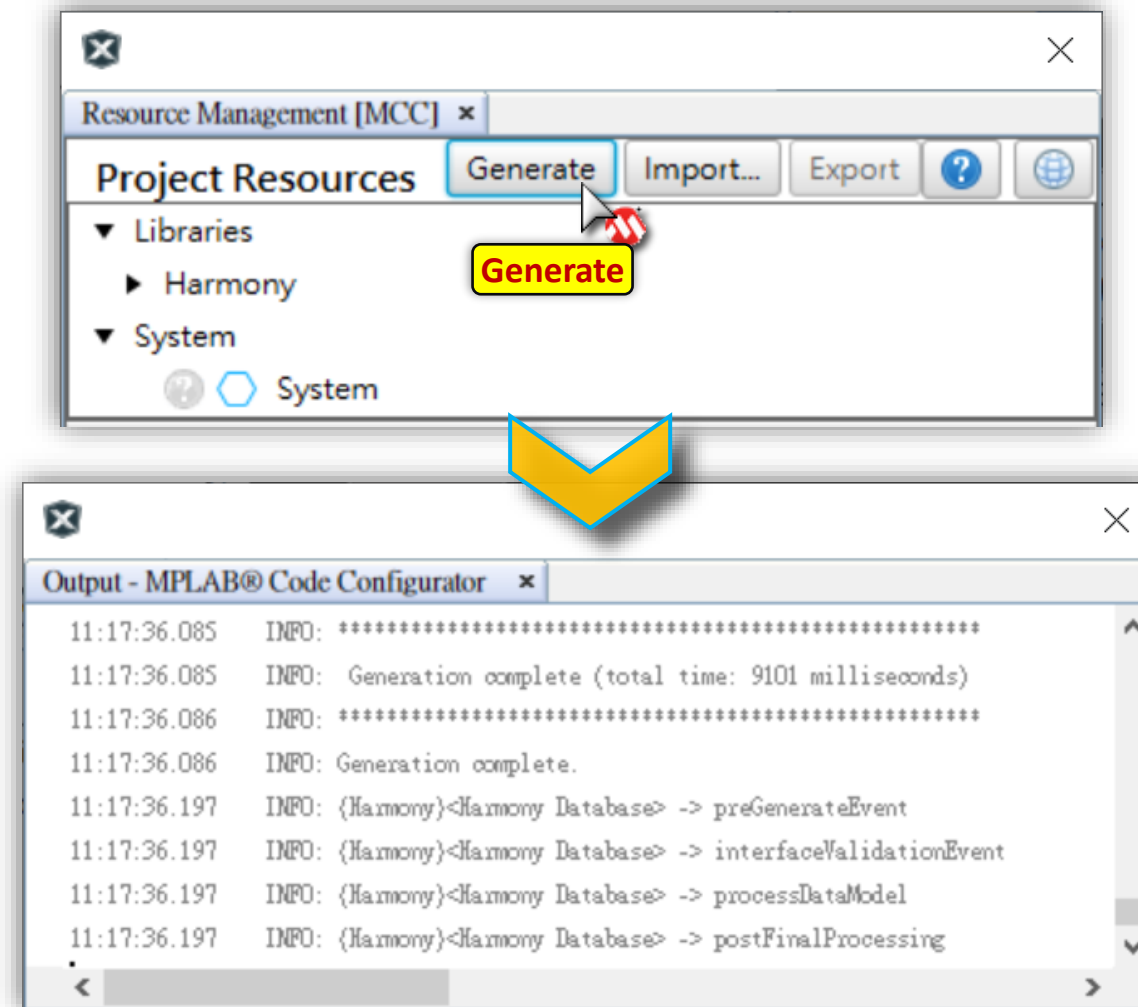
PB08 (ADC_AIN2)

Green is Enable

Lab12 ADC Polling Software Trigger

Step 4

- Click "**Generate**" Button to Generate your Code.



Lab12 ADC Polling Software Trigger

Step 5

a Add code segment to your main.c

```
// TODO 12.01
uint16_t ADC_Result;

int main (void)
{
    ...
    // TODO 12.02
    ADC_Enable();

    while( true )
    {
        ...
        if( TC3_HasExpired)
        {
            ...
            myprintf( "\033[1;1HHello World!\r\n" );

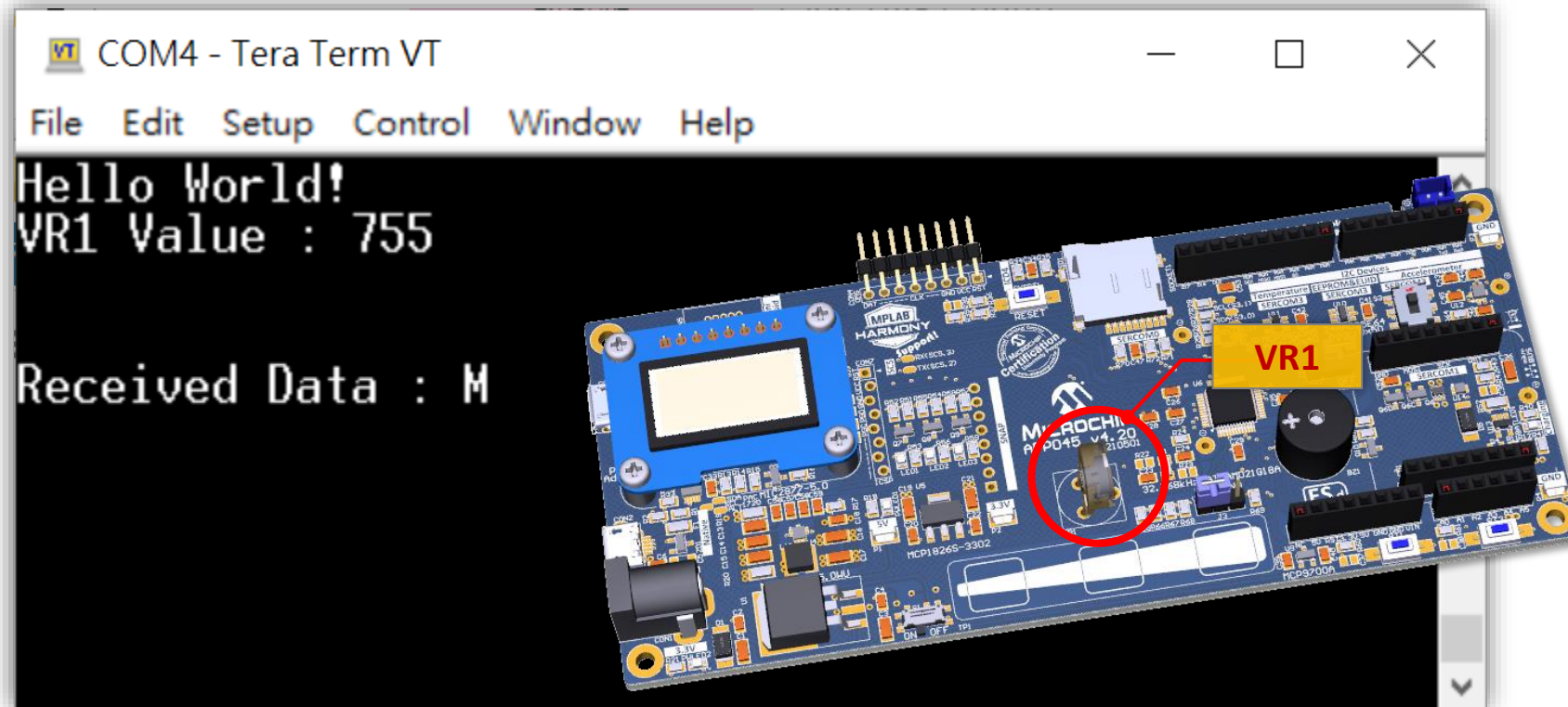
            // TODO 12.03
            ADC_ConversionStart();
            while( !ADC_ConversionStatusGet() );
            ADC_Result = ADC_ConversionResultGet();
            myprintf("\033[2;1HVR : %4d", ADC_Result );
        }
        ...
    }
}
```

b Program firmware to target board then observe result.

Lab12 ADC Polling Software Trigger

Result

- "Hello World!" string and VR1 ADC value will output to terminal with one second period.



ADC Interrupt Callback and Pin Scan

ADC Pin Scan

- The ADC support analog channel switch call Pin Scan function.
- Pin scan switch analog channel sequential after a conversion automatically.
- **Start Pin** : **Positive Input** + **Scan Start Offset**
- **End Pin** : **Start Pin** + **Number of Inputs to Scan**

For Example : To do ADC Pin Scan from AIN[2] to AIN[3].

Positive Input Pin = AIN[0]
Scan Start Offset = 2
Start Pin = AIN[0+2] = AIN[2]

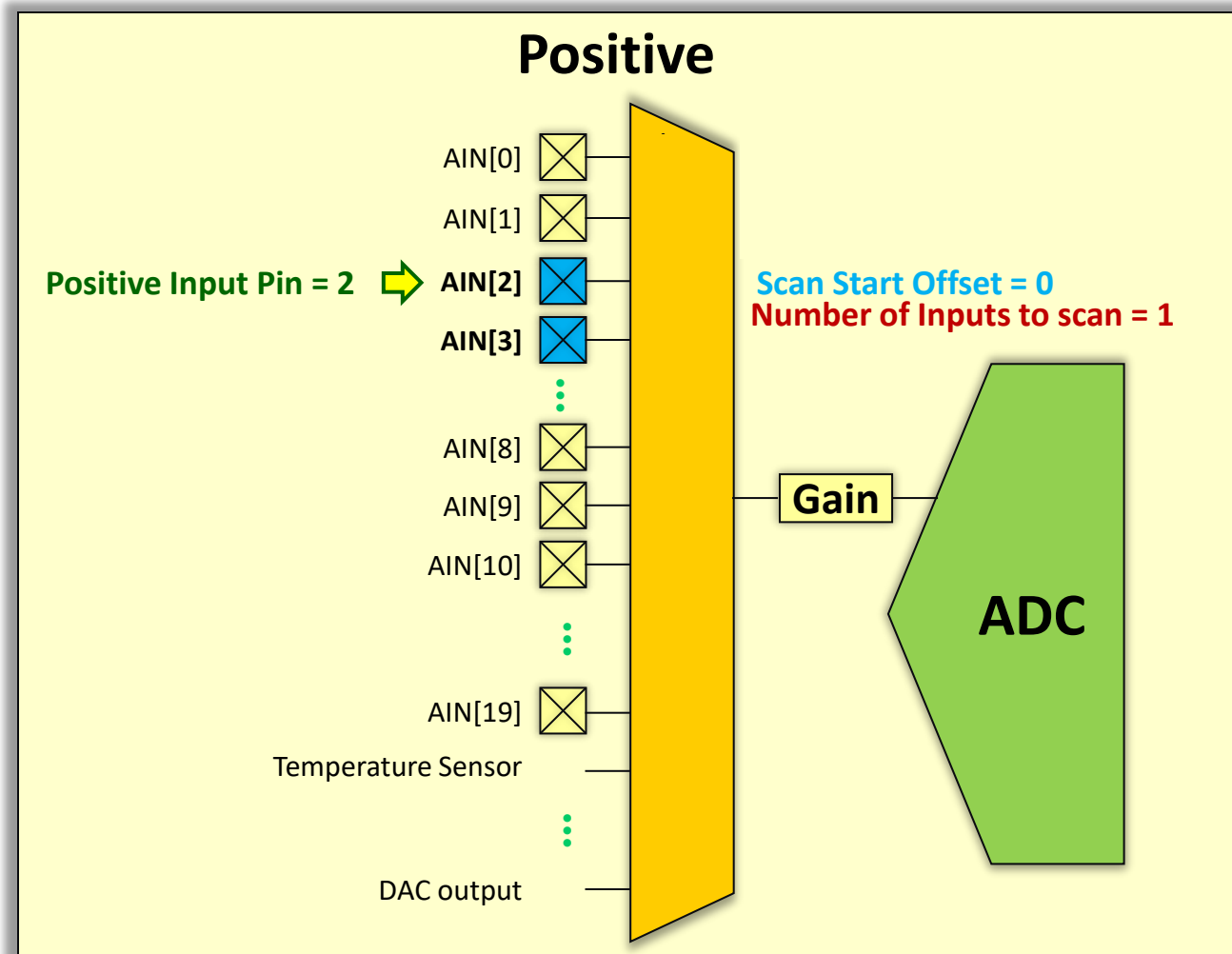
OR

Positive Input Pin = AIN[2]
Scan Start Offset = 0
Start Pin = AIN[2+0] = AIN[2]

Number of Inputs to scan = 1

End Pin = AIN[2+1] = AIN[3]

ADC Pin Scan



ADC Pin Scan

Configuration Options Example

The screenshot shows the 'Configuration Options' dialog box for an ADC. The 'ADC' section is expanded, showing various configuration parameters. Annotations include green highlights on labels, red boxes around specific values, and yellow boxes with text labels.

Configuration Option	Value	Annotation
Select Prescaler	Peripheral clock divided by 512	
Select Sample Length (half ADC clock cycles)	4	
**** Conversion Time is 106.666666667 uS ****		
Select Gain	1/2x	
Select Reference	1/2 VDDANA (only for VDDANA > 2.0V)	
Select Conversion Trigger	SW Trigger	
Channel Configuration		
Select Positive Input	ADC AIN2 Pin	Positive Input Pin
Select Negative Input	Internal Ground	
Number of inputs to scan	1	Number of Inputs to scan
Scan start offset	0	Scan Start Offset

ADC Interrupt

Configuration Options Example

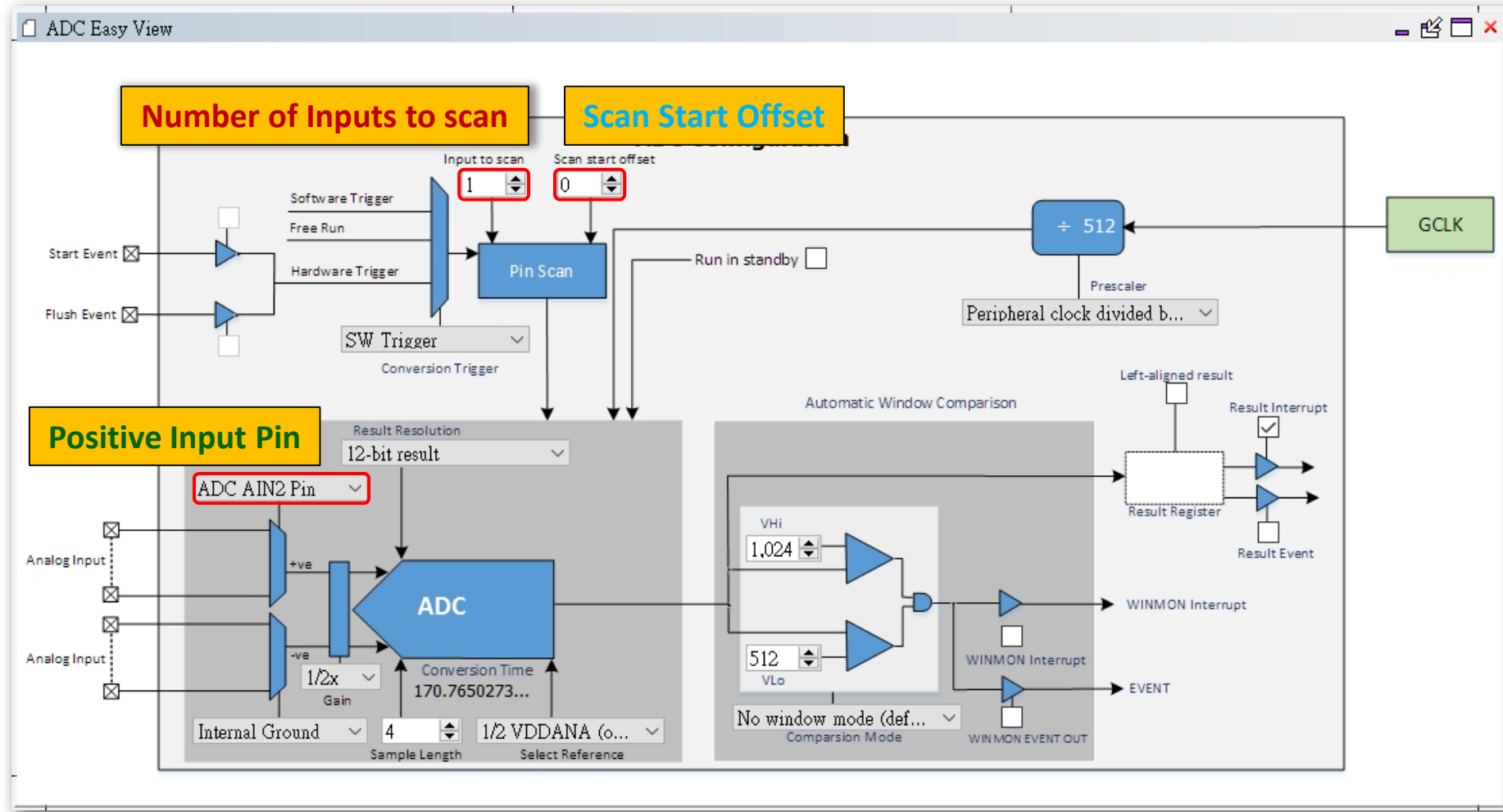
The screenshot shows the 'Configuration Options' dialog box for an ADC. The 'ADC' section is expanded, showing the following settings:

- Select Prescaler:** Peripheral clock divided by 512
- Select Sample Length (half ADC clock cycles):** 4
- **** Conversion Time is 106.666666667 uS ******
- Select Gain:** 1/2x
- Select Reference:** 1/2 VDDANA (only for VDDANA > 2.0V)
- Select Conversion Trigger:** SW Trigger
- Channel Configuration:**
 - Select Positive Input:** ADC AIN2 Pin
 - Select Negative Input:** Internal Ground
 - Number of inputs to scan:** 1
 - Scan start offset:** 0
- Result Configuration:**
 - Select Result Resolution:** 12-bit result
 - Left Aligned Result:** ☐
 - Enable Result Ready Interrupt:** ☒ **Check Enable Result Ready Interrupt**
 - Enable Result Ready Event Out:** ☐

Yes! That's all you need to do.

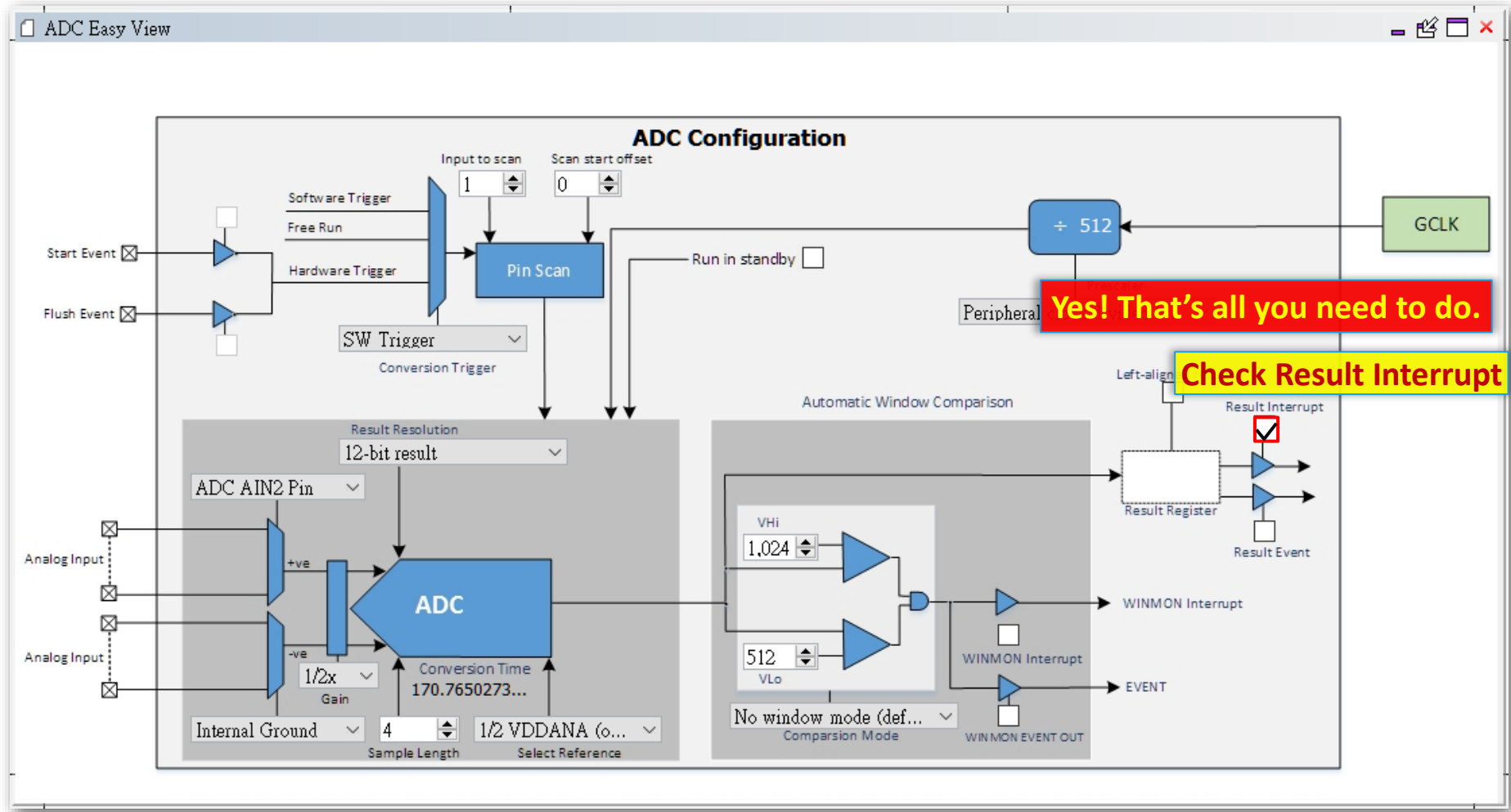
ADC Pin Scan

Easy View Configuration Example



ADC Interrupt

Easy View Configuration Example



ADC Interrupt Callback

Interface Routines

/ ADC Interface Routines**/**

void ADC_Initialize(void);

void ADC_Enable(void);

void ADC_Disable(void);

void ADC_ChannelSelect(ADC_POSINPUT positiveInput, ADC_NEGINPUT negativeInput);

void ADC_ConversionStart(void);

uint16_t ADC_ConversionResultGet(void);

void ADC_ComparisonWindowSet(uint16_t low_threshold, uint16_t high_threshold);

void ADC_WindowModeSet(ADC_WINMODE mode);

void ADC_CallbackRegister(ADC_CALLBACK callback, uintptr_t context);

ADC Interrupt Callback

Software Trigger Code Example

```
...
void ADC_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_INTFLAG_RESRDY_Msk )
    {
        myprintf( "%d\r\n", ADC_ConversionResultGet() );
    }
    ADC_ConversionStart();
}

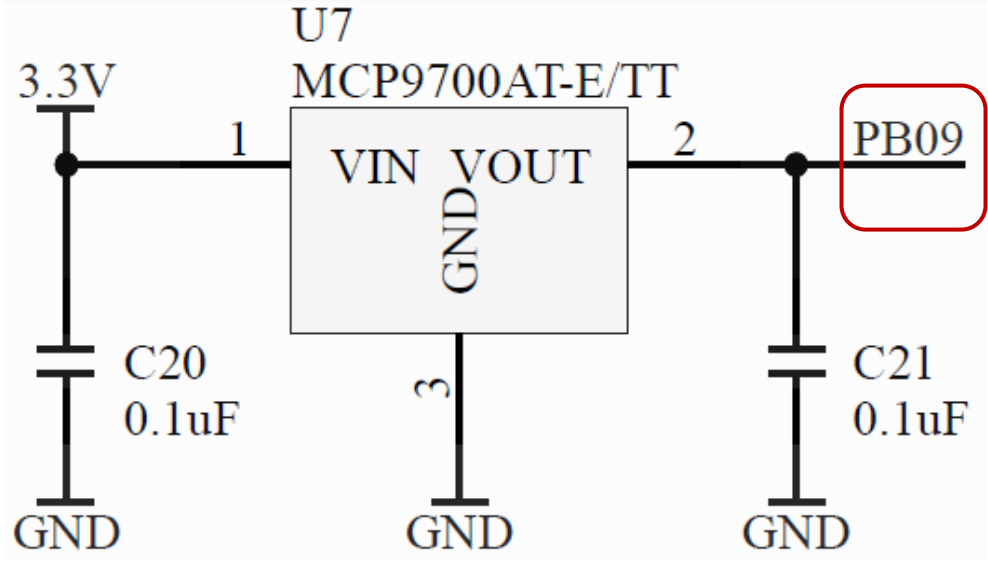
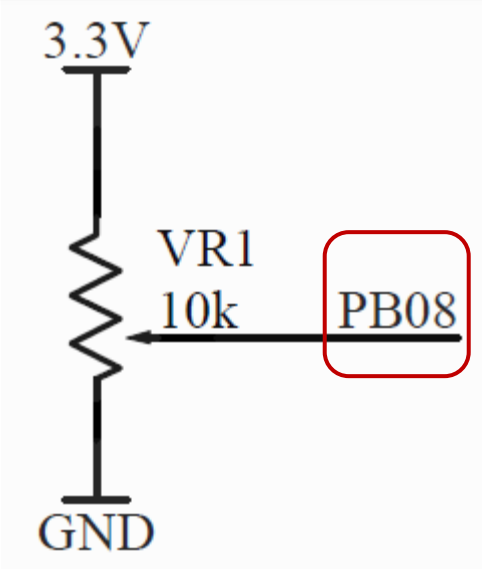
int main (void)
{ ...
    ADC_CallbackRegister( ADC_Complete, (uintptr_t )NULL );
    ADC_Enable();
    ADC_ConversionStart();
    while( true )
    { ...
    }
}
```

Lab13 ADC Pin Scan

- Change ADC polling to Interrupt callback mode and enable pin scan function.
- Get the ADC conversion value from **VR1(AIN[2])** and **Analog Temperature Sensor(AIN[3])** with one second period and then use UART to output ADC values to PC.
- **Let's go!**

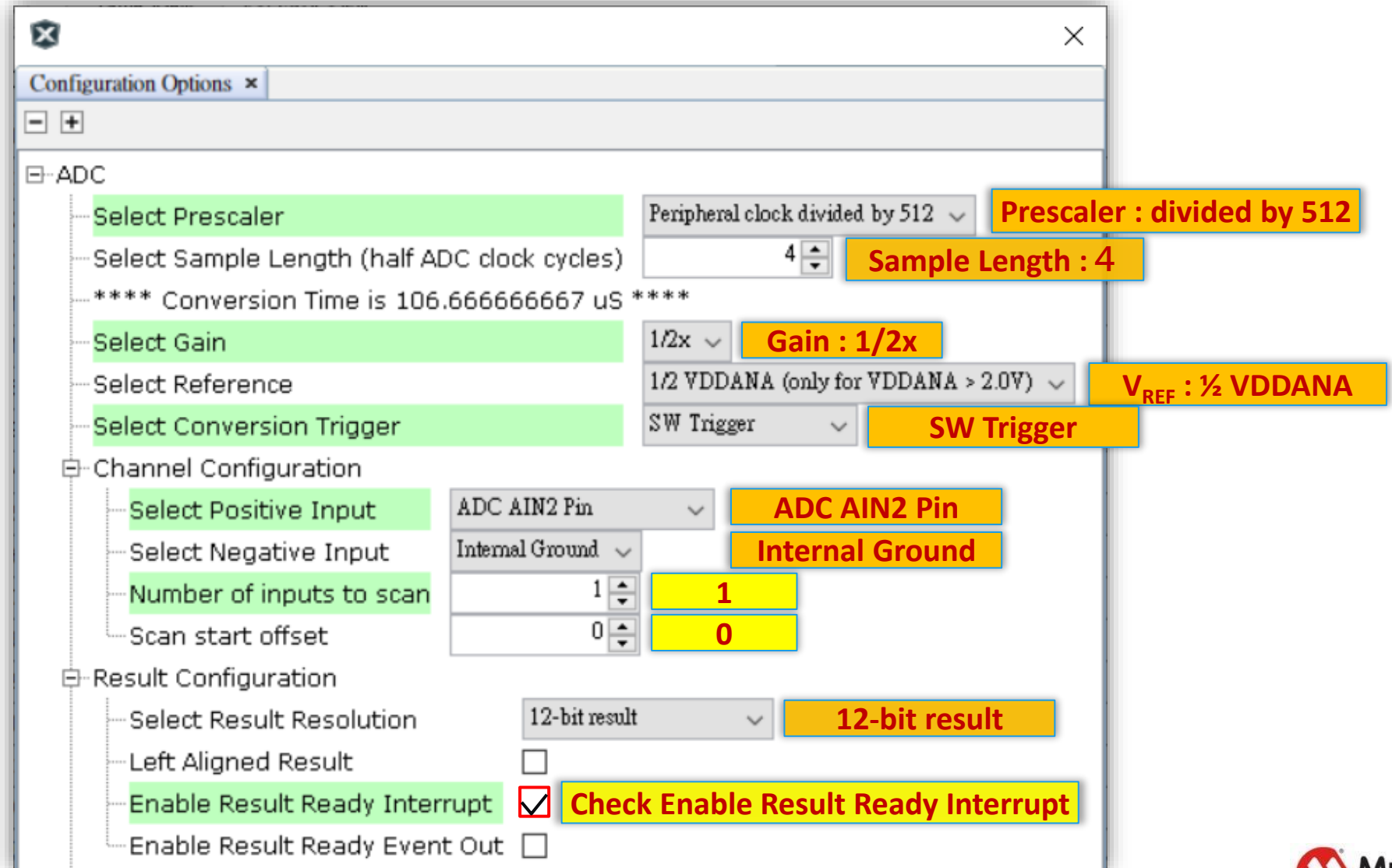
Lab13 ADC Pin Scan

Pin ⁽¹⁾			I/O Pin	Supply	A	B ⁽²⁾⁽³⁾				
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC
	7	11	PB08	VDDANA	EXTINT[8]		AIN[2]		Y[14]	
	8	12	PB09	VDDANA	EXTINT[9]		AIN[3]		Y[15]	



Lab13 ADC Pin Scan

Step 1 (Use Configuration Options)



The screenshot shows the 'Configuration Options' dialog box for the ADC. The 'ADC' section is expanded, showing various configuration options. The 'Channel Configuration' section is also expanded. The 'Result Configuration' section is expanded. The 'Select Prescaler' option is set to 'Peripheral clock divided by 512'. The 'Select Sample Length (half ADC clock cycles)' option is set to '4'. The 'Select Gain' option is set to '1/2x'. The 'Select Reference' option is set to '1/2 VDDANA (only for VDDANA > 2.0V)'. The 'Select Conversion Trigger' option is set to 'SW Trigger'. The 'Select Positive Input' option is set to 'ADC AIN2 Pin'. The 'Select Negative Input' option is set to 'Internal Ground'. The 'Number of inputs to scan' option is set to '1'. The 'Scan start offset' option is set to '0'. The 'Select Result Resolution' option is set to '12-bit result'. The 'Left Aligned Result' option is unchecked. The 'Enable Result Ready Interrupt' option is checked. The 'Enable Result Ready Event Out' option is unchecked. The conversion time is displayed as '**** Conversion Time is 106.666666667 uS ****'.

Configuration Options

ADC

- Select Prescaler: Peripheral clock divided by 512 **Prescaler : divided by 512**
- Select Sample Length (half ADC clock cycles): 4 **Sample Length : 4**
- **** Conversion Time is 106.666666667 uS ****
- Select Gain: 1/2x **Gain : 1/2x**
- Select Reference: 1/2 VDDANA (only for VDDANA > 2.0V) **V_{REF} : ½ VDDANA**
- Select Conversion Trigger: SW Trigger **SW Trigger**

Channel Configuration

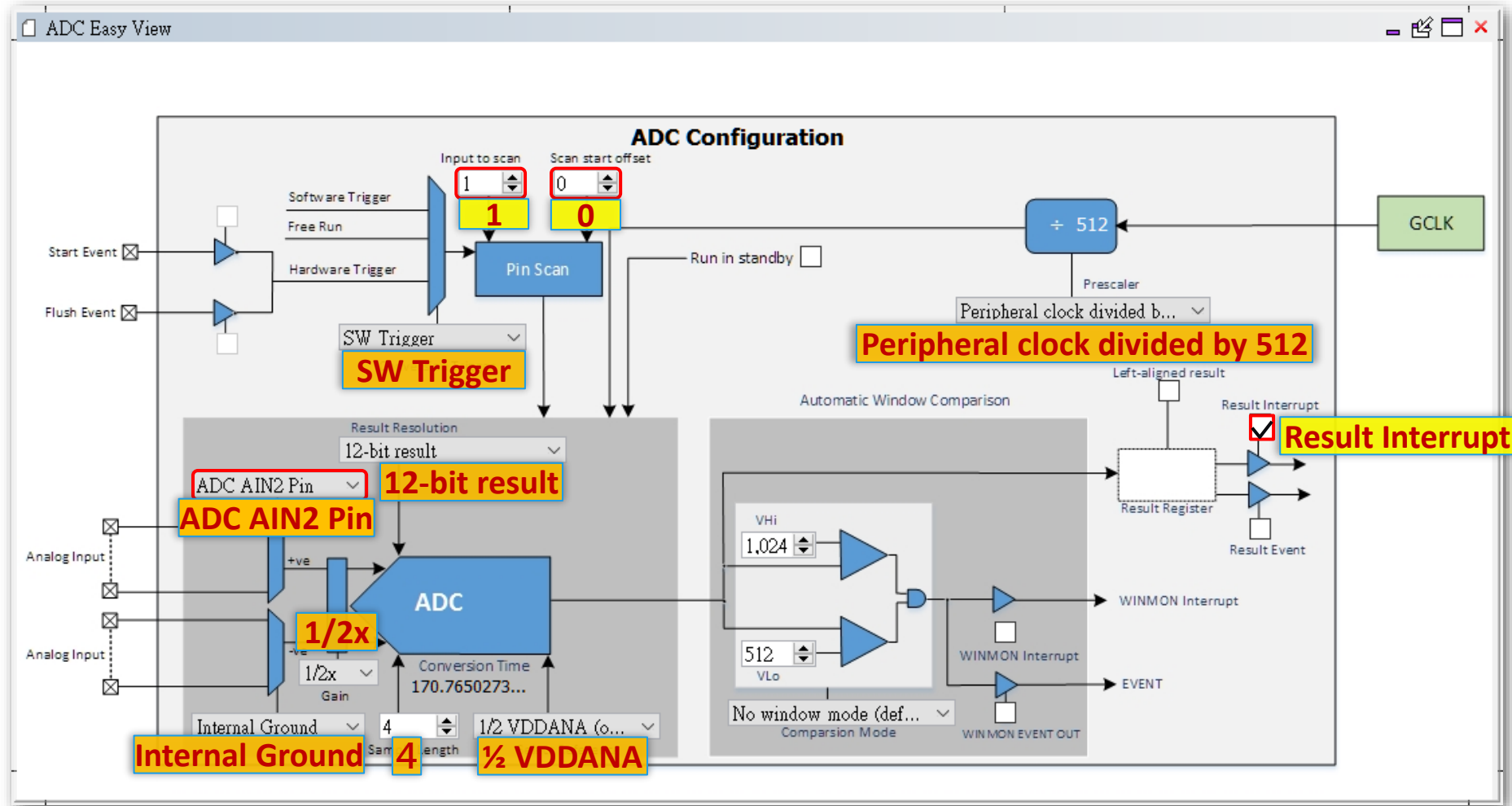
- Select Positive Input: ADC AIN2 Pin **ADC AIN2 Pin**
- Select Negative Input: Internal Ground **Internal Ground**
- Number of inputs to scan: 1 **1**
- Scan start offset: 0 **0**

Result Configuration

- Select Result Resolution: 12-bit result **12-bit result**
- Left Aligned Result: ☐
- Enable Result Ready Interrupt: ☒ **Check Enable Result Ready Interrupt**
- Enable Result Ready Event Out: ☐

Lab13 ADC Pin Scan

Step 1 (Use Easy View)



Lab13 ADC Pin Scan

Step 2

Pin Table

Package: TQFP48

Module	Function	PA00	PA01	PA02	PA03	GNDAN	VDDAN	ADC_AI	PB09	PA04	PA05	PA06	PA07	PA08
		1	2	3	4	5	6	7	8	9	10	11	12	13
ADC	ADC_AIN18													
	ADC_AIN19													
	ADC_AIN2													
	ADC_AIN3													
	ADC_AIN5													
	ADC_AIN6													

ADC

ADC_AIN3

Click

Pin Table

Package: TQFP48

Module	Function	PA00	PA01	PA02	PA03	GNDAN	VDDAN	ADC_AI	ADC_AI	PA04	PA05	PA06	PA07	PA08
		1	2	3	4	5	6	7	8	9	10	11	12	13
ADC	ADC_AIN18													
	ADC_AIN19													
	ADC_AIN2													
	ADC_AIN3													
	ADC_AIN4/VREFP													
	ADC_AIN5													
ADC_AIN6														

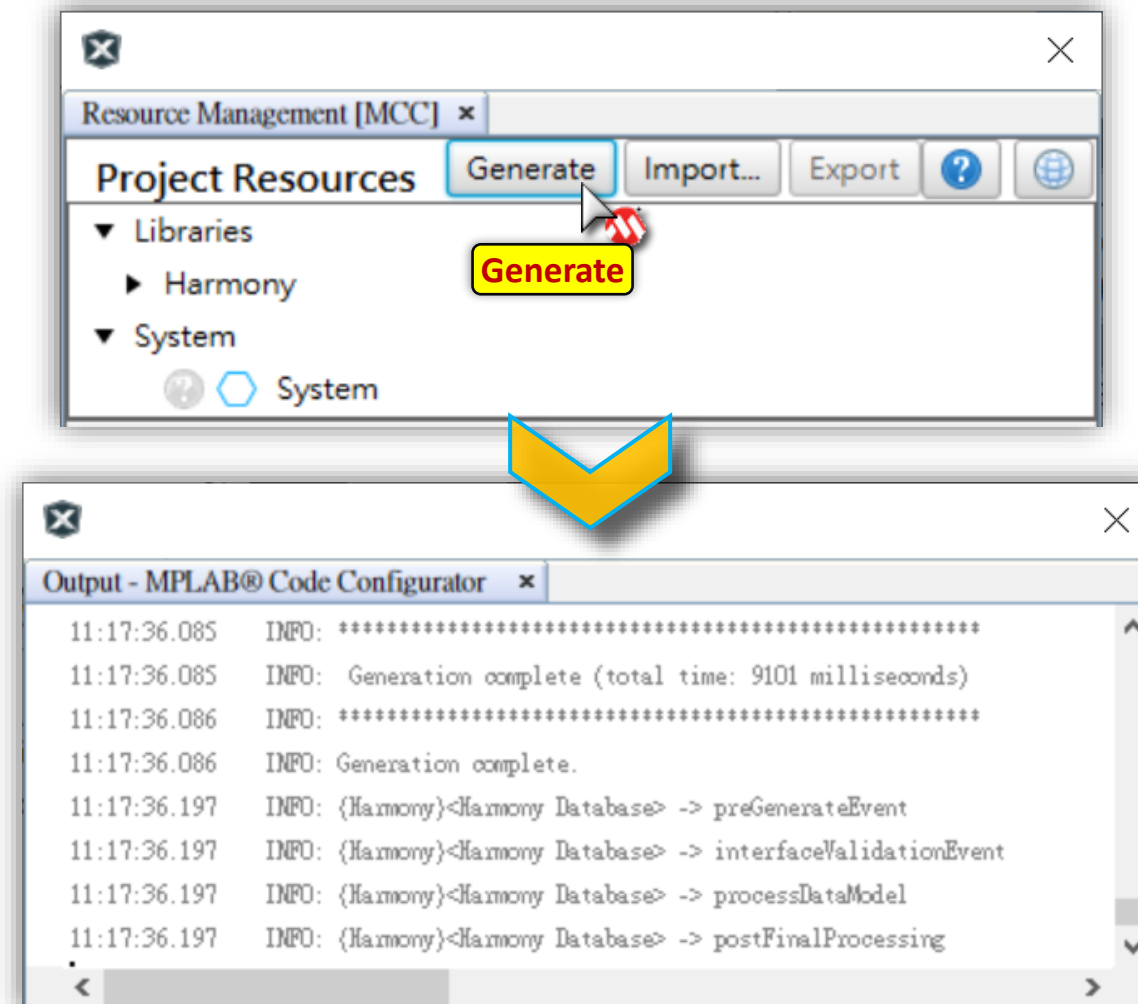
PB08 (ADC_AIN2)
PB09 (ADC_AIN3)

Green is Enable

Lab13 ADC Pin Scan

Step 3

- Click "**Generate**" Button to Generate your Code.



Lab13 ADC Pin Scan

Step 4

a Add code segment to your main.c

```
// TODO 13.01
uint16_t ADC_Result[2];
volatile uint8_t ADC_IsCompleted = 0;
volatile int8_t ADC_ChannelIdx = 0;

// TODO 13.02
void ADC_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_INTFLAG_RESRDY_Msk )
    {
        ADC_Result[ADC_ChannelIdx] = ADC_ConversionResultGet();
        ADC_ChannelIdx = ( ADC_ChannelIdx + 1 ) % 2;
        if( ADC_ChannelIdx == 0 )
        {
            ADC_IsCompleted = 1;
        }
    }
}
...

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

Lab13 ADC Pin Scan

Step 5

a Add code segment to your main.c

```
...

int main (void)
{
    ...

    // TODO 13.03
    ADC_CallbackRegister( ADC_Complete, (uintptr_t )NULL );
    ADC_Enable();

    while( true )
    {
        ...
        if( TC3_HasExpired)
        {
            // TODO 13.04
            ADC_ConversionStart();
        }
        ...
    }
}
```

Lab13 ADC Pin Scan

Step 6

a Add code segment to your main.c

```
...

int main (void)
{
    ...

    while( true )
    { ...

        if( USART5_IsReceived )
        { ... }

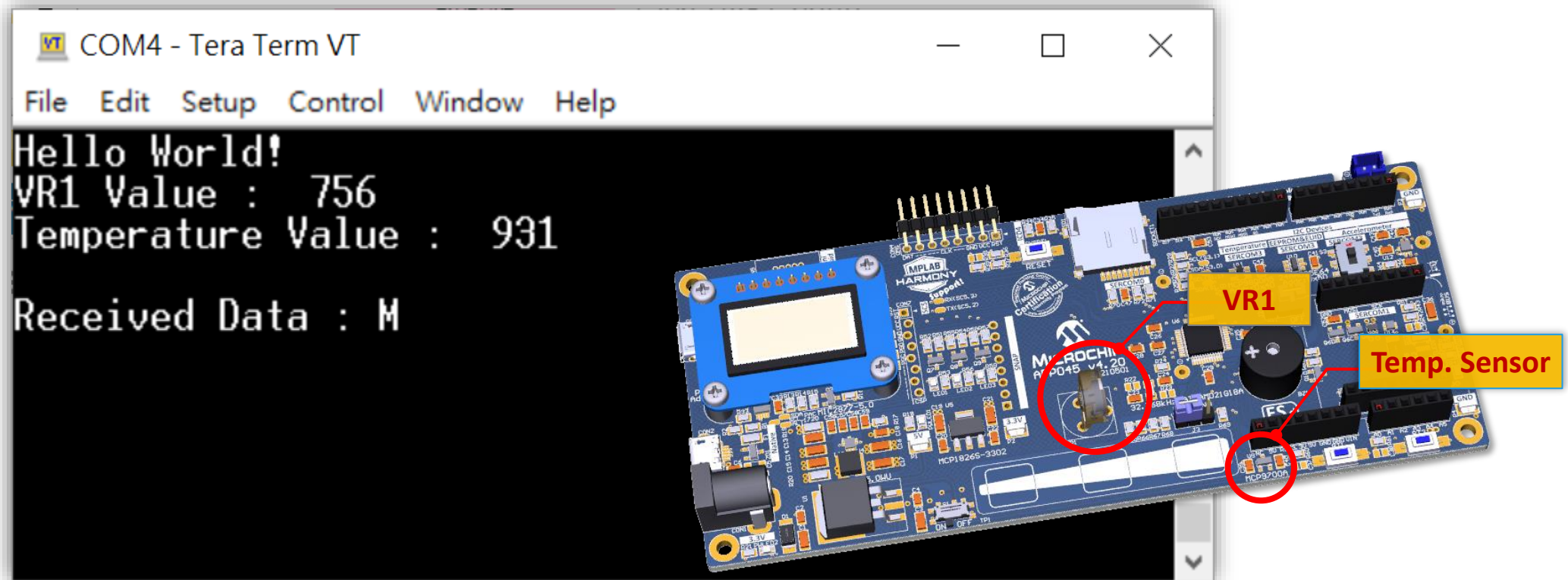
        // TODO 13.05
        if( ADC_IsCompleted )
        {
            ADC_IsCompleted = 0;

            myprintf( "\033[2;1HVR1 Value : %4d", ADC_Result[0] );
            myprintf( "\033[3;1HTemperature Value : %4d", ADC_Result[1] );
        }
    }
}
```

Lab13 ADC Pin Scan

Result

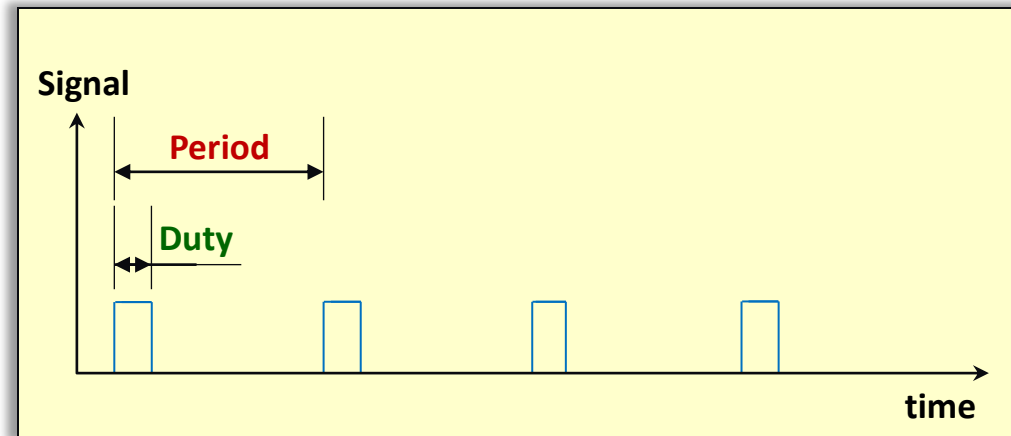
- "Hello World!!" string and VR1 / Temperature Sensor ADC value will output to terminal with one second period.



TCC – PWM Architecture

What's PWM

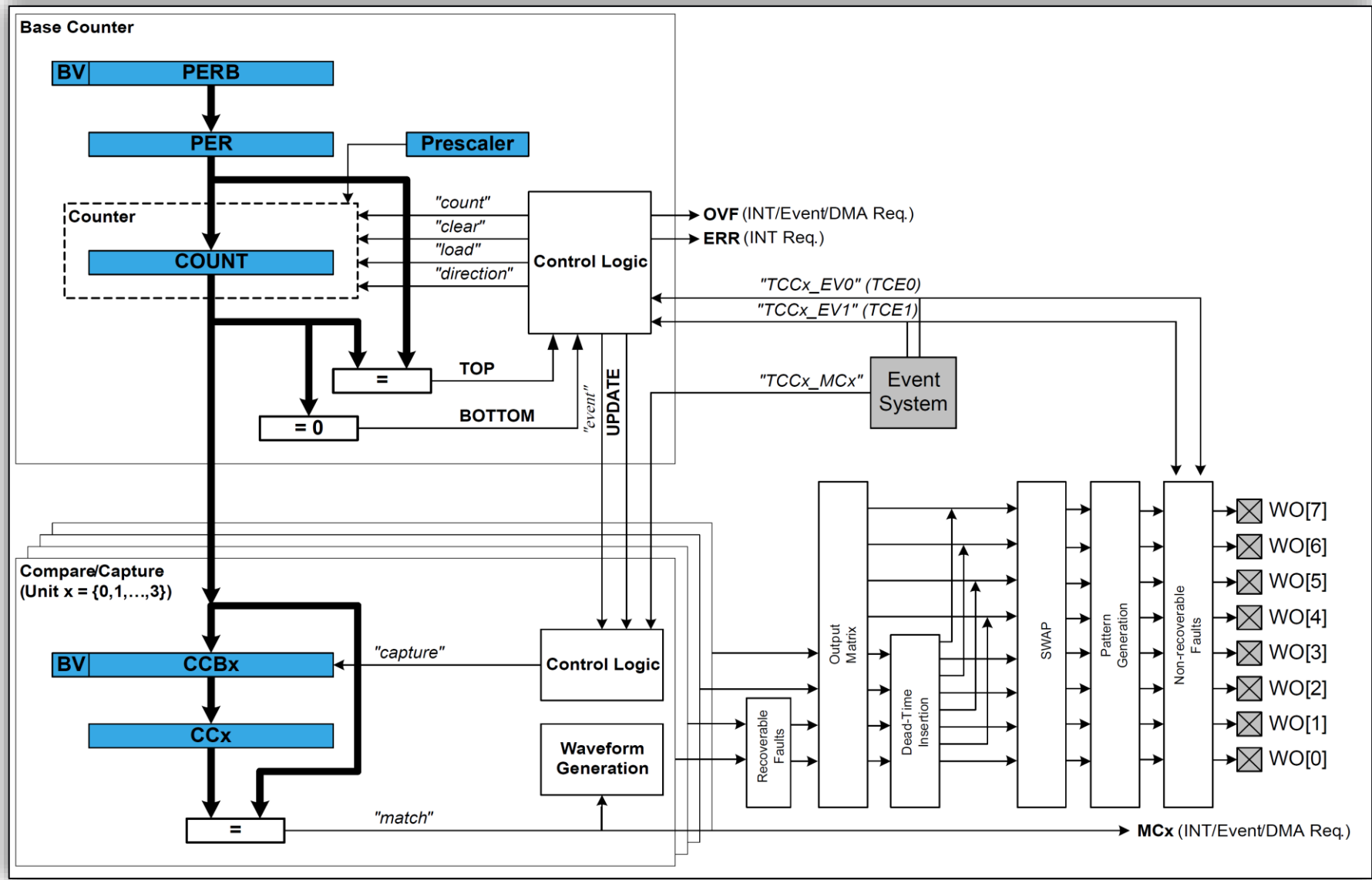
- The Pulse-width modulation (PWM) Waveform is a modulation technique used to encode a message into a pulsing signal.
- Its main use is to allow the control of the power supplied to electrical devices, ex. motor control, ballast, LED, H-bridge, power converters, and other types of power control applications.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



SAMD21 TCC

- **Timer/Counter for Control, TCC is the TC Enhanced version.**
- **TCC provide below feature**
 - Up to four compare/capture channels
 - Waveform (PWM, Frequency generation)
 - Input capture (Event, Frequency, PWM Capture)
 - Fault protection for safe disabling
 - 3 input Event, 3 output Event
 - Support Interrupt, DMA, Event

TCC Block Diagram



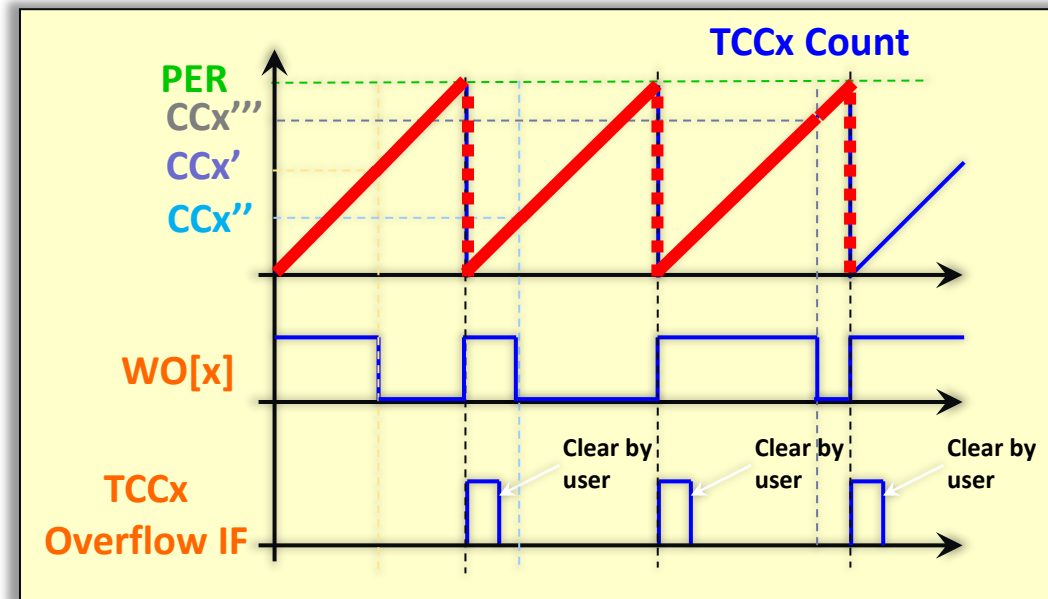
TCCx, Channel and WO[x]

- **TCCx : Timer Counter for Control peripheral**
 - TCC0, TCC1, TCC2
- **Channel : Compare Channel CCx**
 - TCC0 : 4 channel
 - TCC1 : 2 channel
 - TCC2 : 2 channel
- **WO[x] : Waveform Output Pins**
 - TCC0 : 8 pins → WO[0] ~ WO[7] → Channel [0, 1, 2, 3, **0, 1, 2, 3**]
 - TCC1 : 4 pins → WO[0] ~ WO[3] → Channel [0, 1, **0, 1**]
 - TCC2 : 2 pins → WO[0] ~ WO[1] → Channel [0, 1]

Normal Pulse-Width Modulation

NPWM Mode

- For Normal Pulse-Width Modulation, the period time (T) is controlled by the PER register. WO[x] is set at start and cleared on compare match.

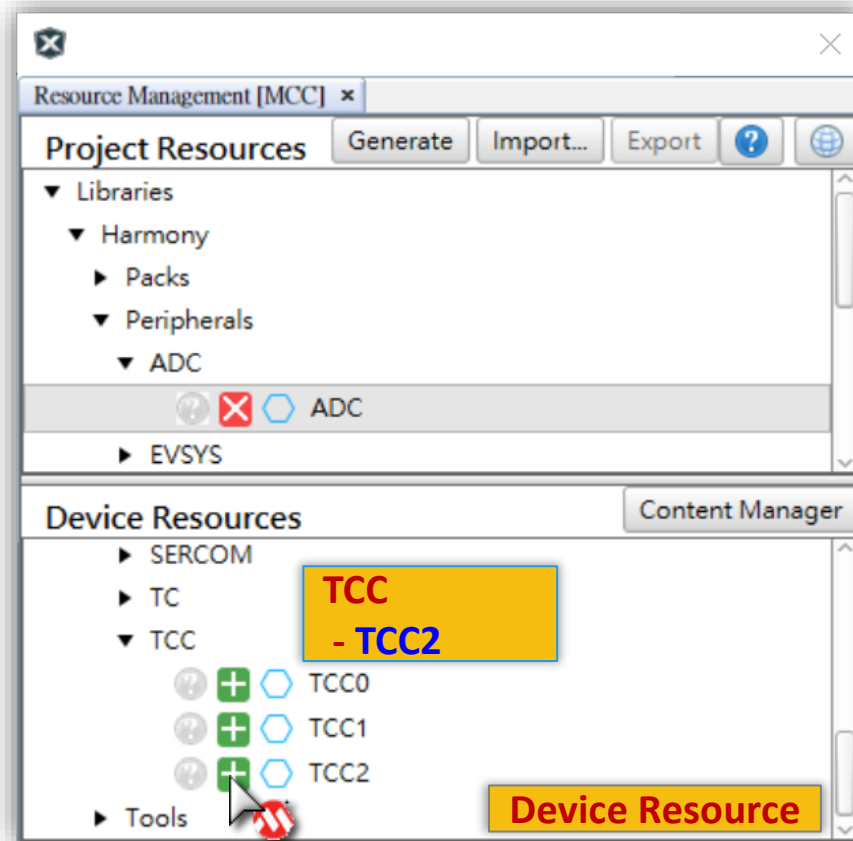


Rule

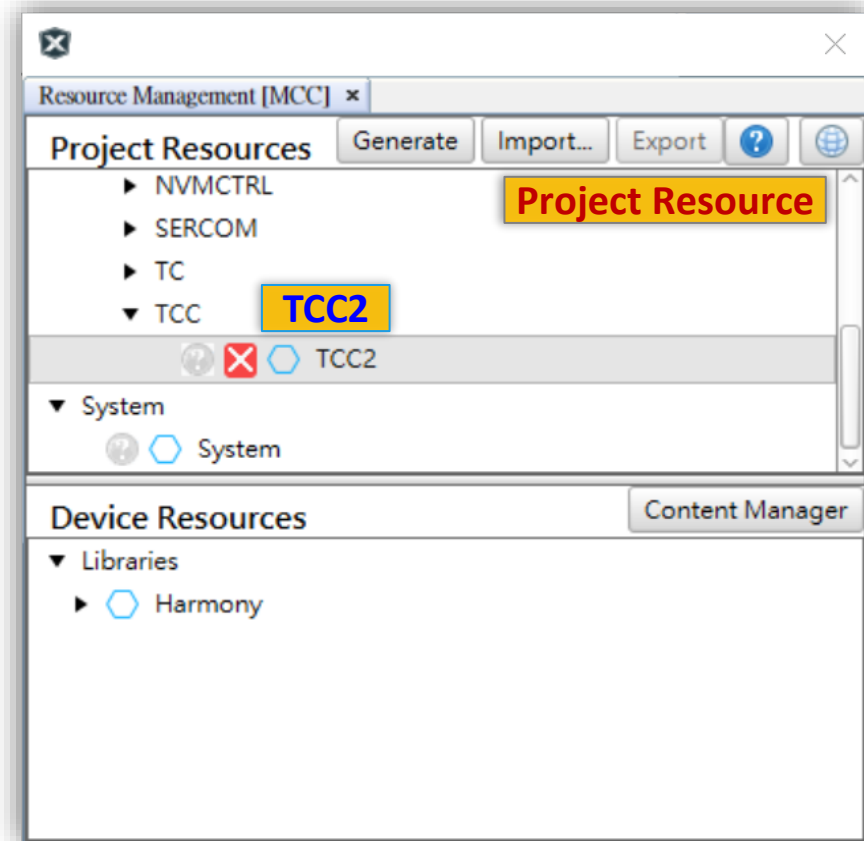
1. TCCx Counter == CCx -> WO[x] Clear to low
2. TCCx Counter == PER -> WO[x] Set to high

Add TCC Function using MCC Harmony

- Find **TCC** component in Device Resource window.
- Double click **TCC2** to add to Project Resource window.



Double click



TCC Configuration Options Example

Configuration Options

TCC2

- Run during Standby ☐
- Select Prescaler No division
- Operating Mode PWM
 - PWM
 - Select PWM Type NPWM
 - PWM Direction - Count Down ☐
 - Period Value 48,000
 - **** PWM Frequency is 999 Hz ****
 - Enable Period Interrupt ☐
 - Enable Period Event Out ☐
 - Channel Configurations
 - Channel 0
 - Duty Value 0
 - Output Polarity Output is ~DIR and set to DIR when counter matches CCx value
 - Enable Compare Match Interrupt ☐
 - Enable Compare Match Event OUT ☐
 - Enable Compare Match Event IN ☐
 - Channel 1
 - Duty Value 9,600
 - Output Polarity Output is ~DIR and set to DIR when counter matches CCx value
 - Enable Compare Match Interrupt ☐
 - Enable Compare Match Event OUT ☐
 - Enable Compare Match Event IN ☐
 - Outputs
 - Input Events Configuration
 - Fault Configurations

TCC PMUX Configuration Example

- TCC could output waveform to corresponds pins of different channel.
- For example :
PA17 as TCC2/WO[1] (TCC2 Channel 1) or
TCC0/WO[7] (TCC0 Channel 3)

Pin(1)			I/O Pin	Supply	A	B(2)(3)					C	D	E	F	G	H
SAMD21E	SAMD21G	SAMD21J				REF	ADC	AC	PTC	DAC						
18	26	PA17	PA17	VDDIO	EXTINT[1]				X[5]		TCC2/WO[1]		TCC2/WO[1]	TCC0/WO[7]	TCC0/WO[7]	

Pin Table

Package: TQFP48

Module	Function	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	GCLK_I	GCLK_I	PA16	TCC2_WO
	TCC1_WO3												
TCC2	TCC2_WO0												
	TCC2_WO1												

TCC2

TCC2/WO[1]

Pin Table

Package: TQFP48

Module	Function	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	GCLK_I	GCLK_I	PA16	TCC0_WO
	TCC0_WO5												
TCC0	TCC0_WO6												
	TCC0_WO7												

TCC0

TCC0/WO[7]

TCC Polling

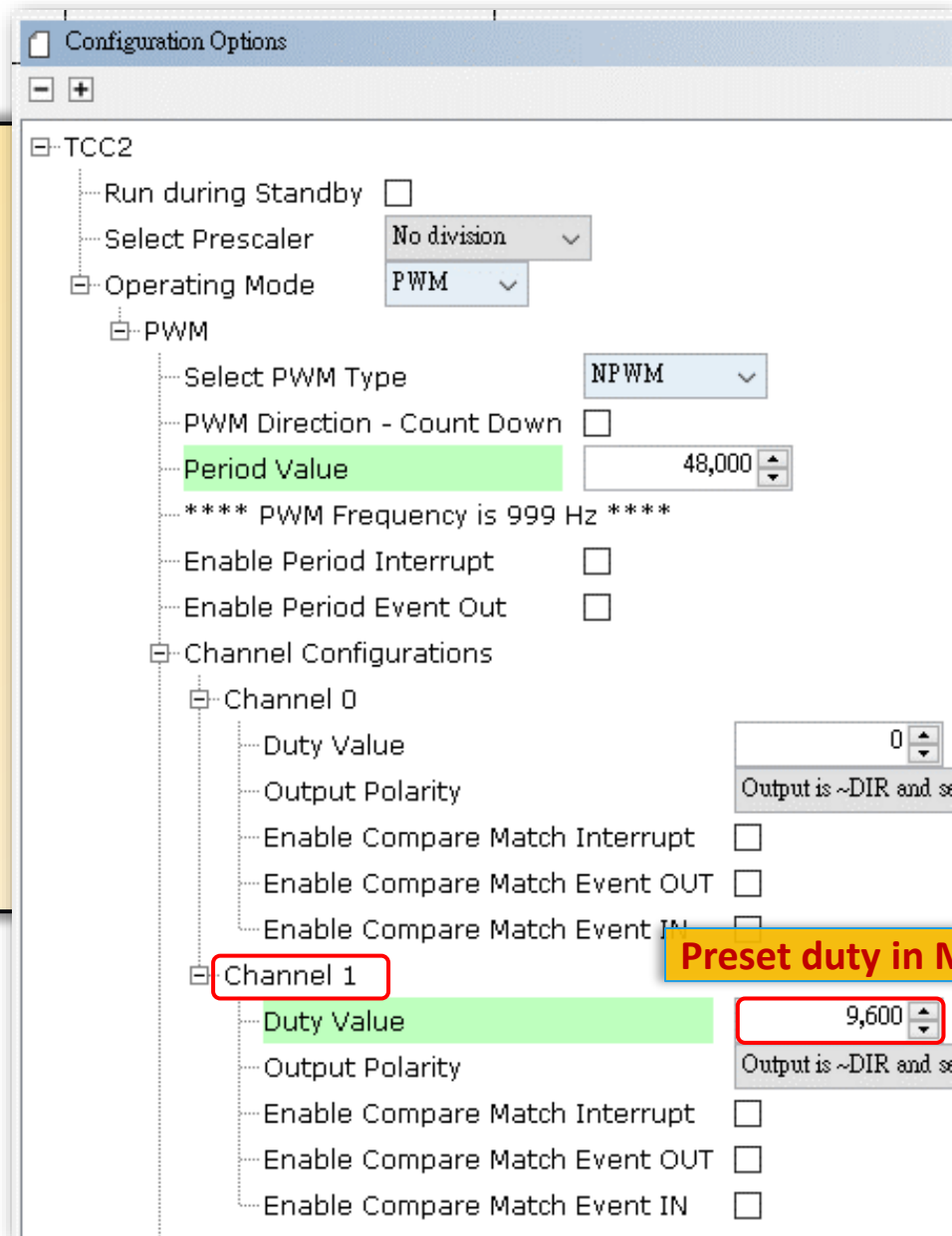
Interface Routines

```
void TCC2_PWMInitialize(void);  
void TCC2_PWMStart(void);  
void TCC2_PWMStop(void);  
void TCC2_PWMForceUpdate(void);  
void TCC2_PWMPeriodInterruptEnable(void);  
void TCC2_PWMPeriodInterruptDisable(void);  
uint32_t TCC2_PWMInterruptStatusGet(void);  
void TCC2_PWM16bitPeriodSet(uint16_t period);  
uint16_t TCC2_PWM16bitPeriodGet(void);  
void TCC2_PWM16bitCounterSet(uint16_t count);  
__STATIC_INLINE void TCC2_PWM16bitDutySet(TCC2_CHANNEL_NUM channel, uint16_t duty);
```

TCC Polling Code Example

```
...  
  
int main (void)  
{  
    SYS_Initialize ( NULL );  
    ...  
    TCC2_PWMStart();  
    TCC2_PWM16bitDutySet( 1, 9600 );  
  
    while( true )  
    {  
        ...  
    }  
}
```

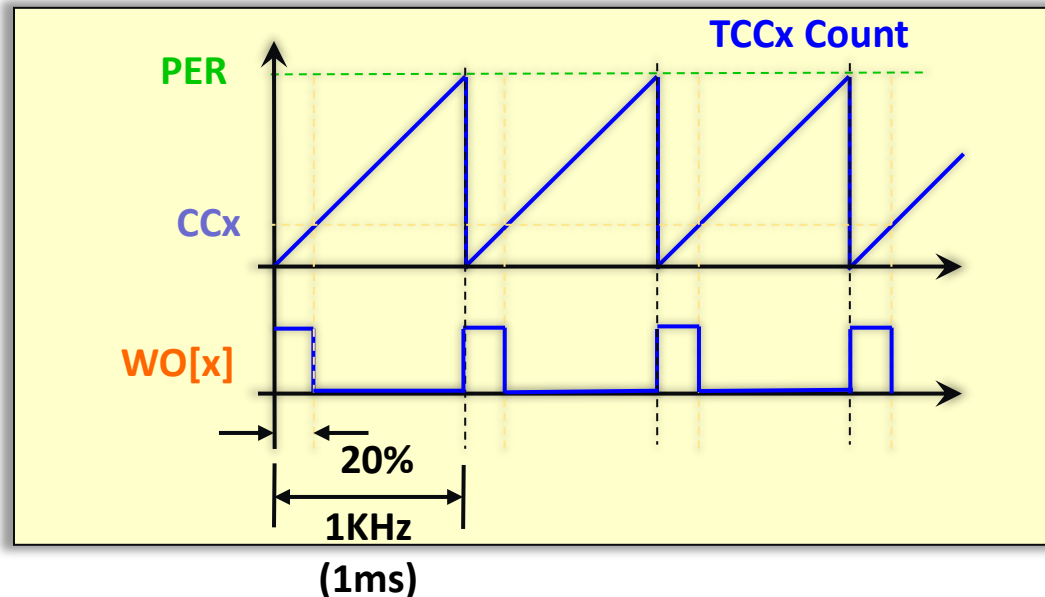
**Runtime update duty
Channel 1, Duty 9600**



Lab14 TCC NPWM

- Add **TCC2 NPWM Mode (Single Slope PWM)** function to your project.
- **Disable LED3(PA17) GPIO function firstly.**
- Configure **TCC2** to output PWM waveform to **LED3(PA17)**.
- PWM frequency is **1KHz** with **20% Duty**.

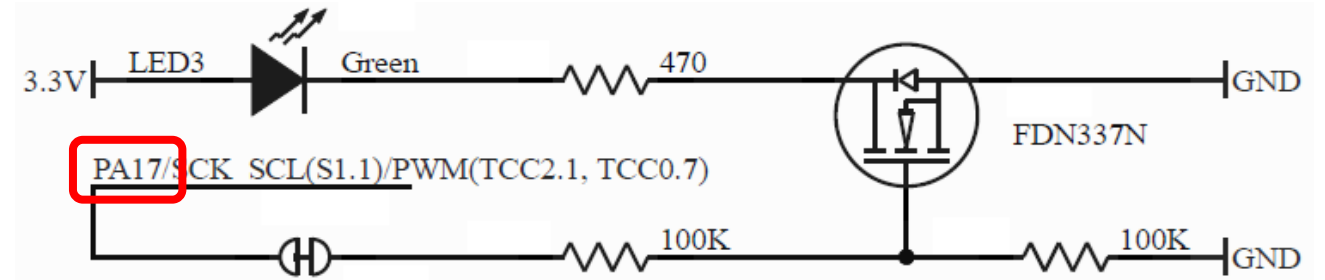
• **Let's go!**



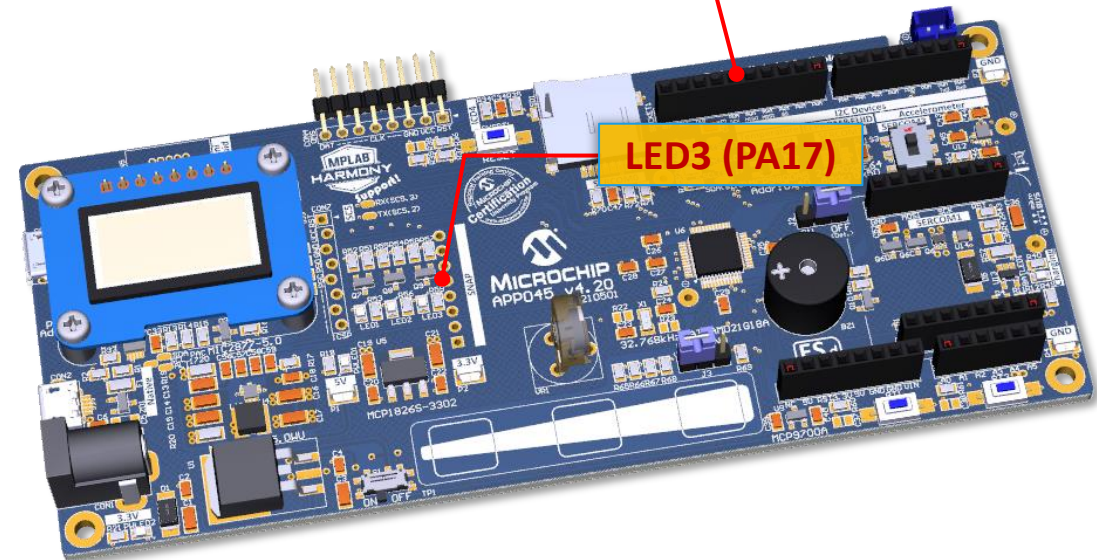
Lab14 TCC NPWM

PA19	27	PA18
PA18	26	PA17
PA17	25	PA16
PA16		

		Arduino UNO Socket
PA06	15	D8
PA07	16	D9/PWM
PA18	17	D10/PWM/SS
PA16	18	D11/PWM/MOSI
PA19	19	D12/MISO
PA17	20	D13/SCK



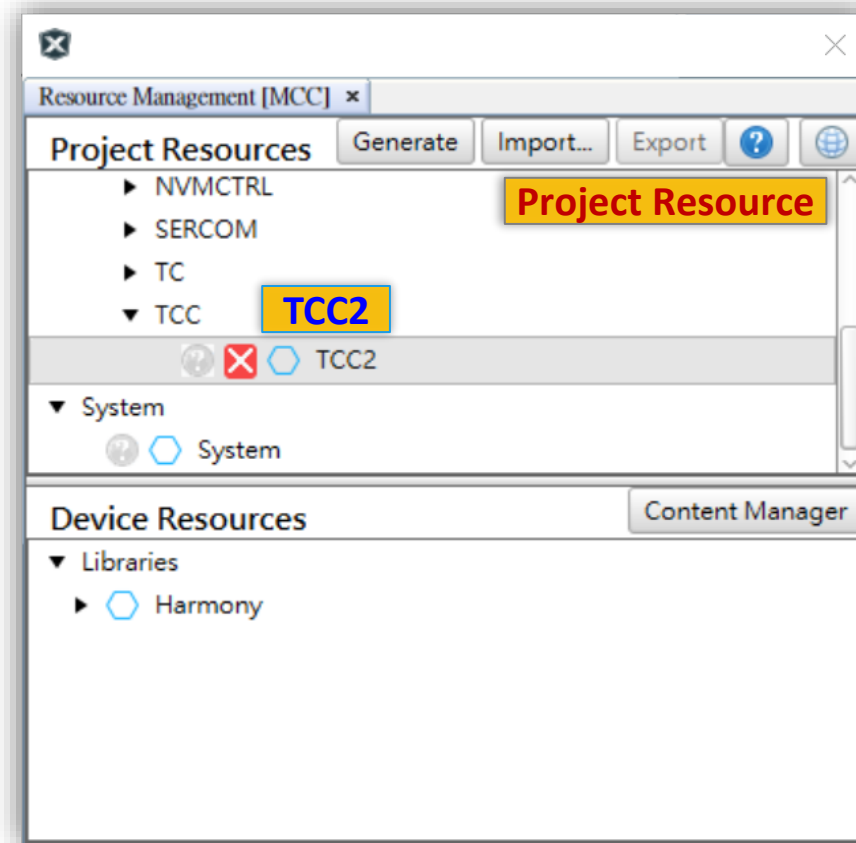
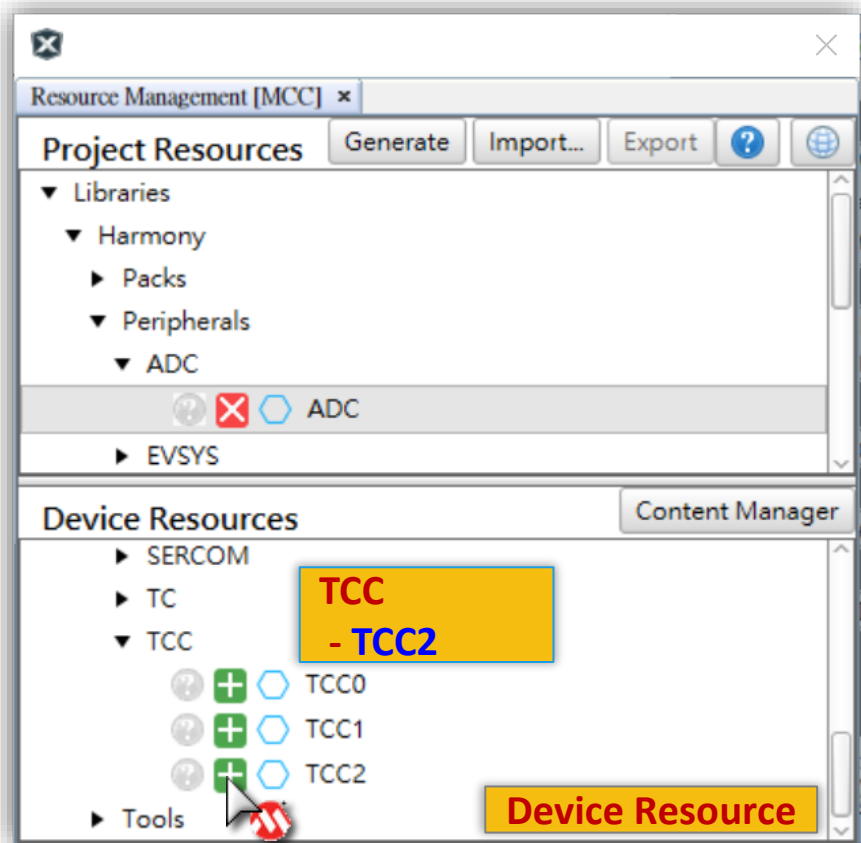
PA17 TCC2_WO[1] (D13)



Lab14 TCC NPWM

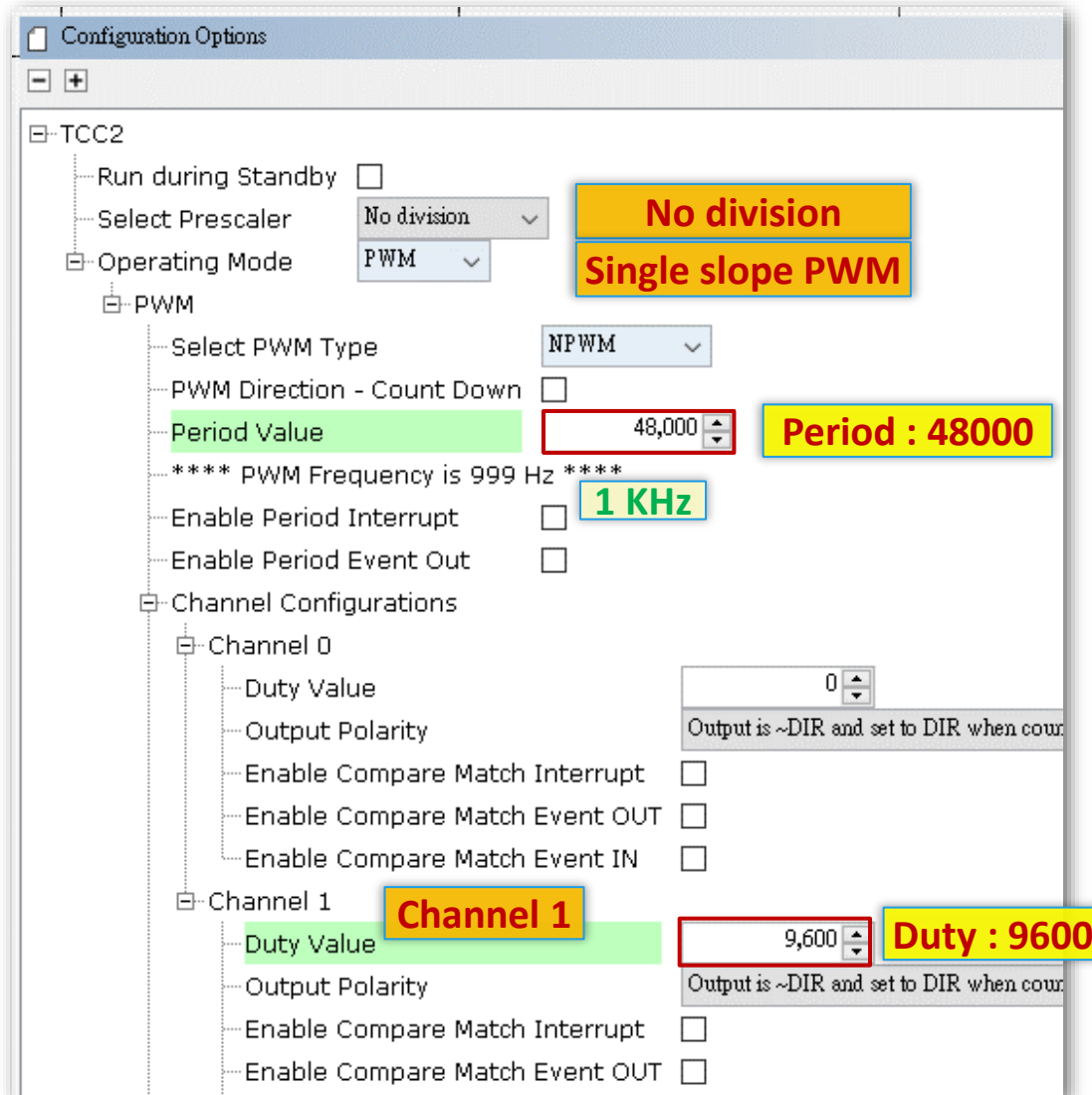
Step 1

- Find **TCC** component in Available Components window.
- Double click **TCC2** to add to Active Components window.



Lab14 TCC NPWM

Step 2



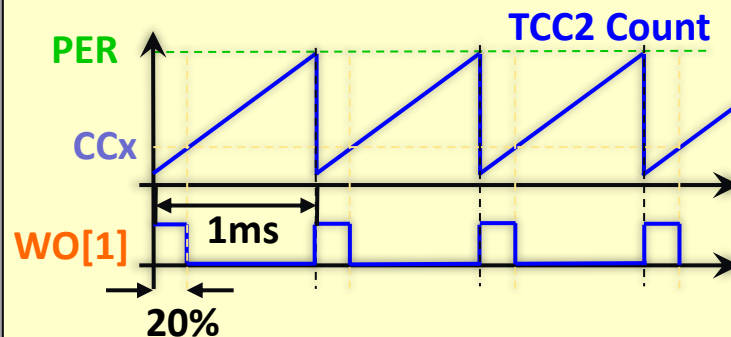
System Clock = 48MHz

One second counts of system clock :
48,000,000 counts

One second counts after Prescaler :
 $48000000 / 1$ 48,000,000 counts

1ms counts for LAB14 period target :
 $48000000 / 1000$ 48,000 counts

1/5ms counts for LAB14 duty target :
 $48000 / 5$ 9,600 counts



Lab14 TCC NPWM

Step 3

- Disable LED3(PA17) GPIO function firstly.
- Enable TCC2 Waveform Output 1 (TCC2_WO1).

Pin Table Configuration - GPIO

Module	Function	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	GCLK_I	GCLK_I	PA16	PA17
		15	16	17	18	19	20	21	22	23	24	25	26
GPIO	GCLK_IO7												
	GPIO												
	I2S_FS0												

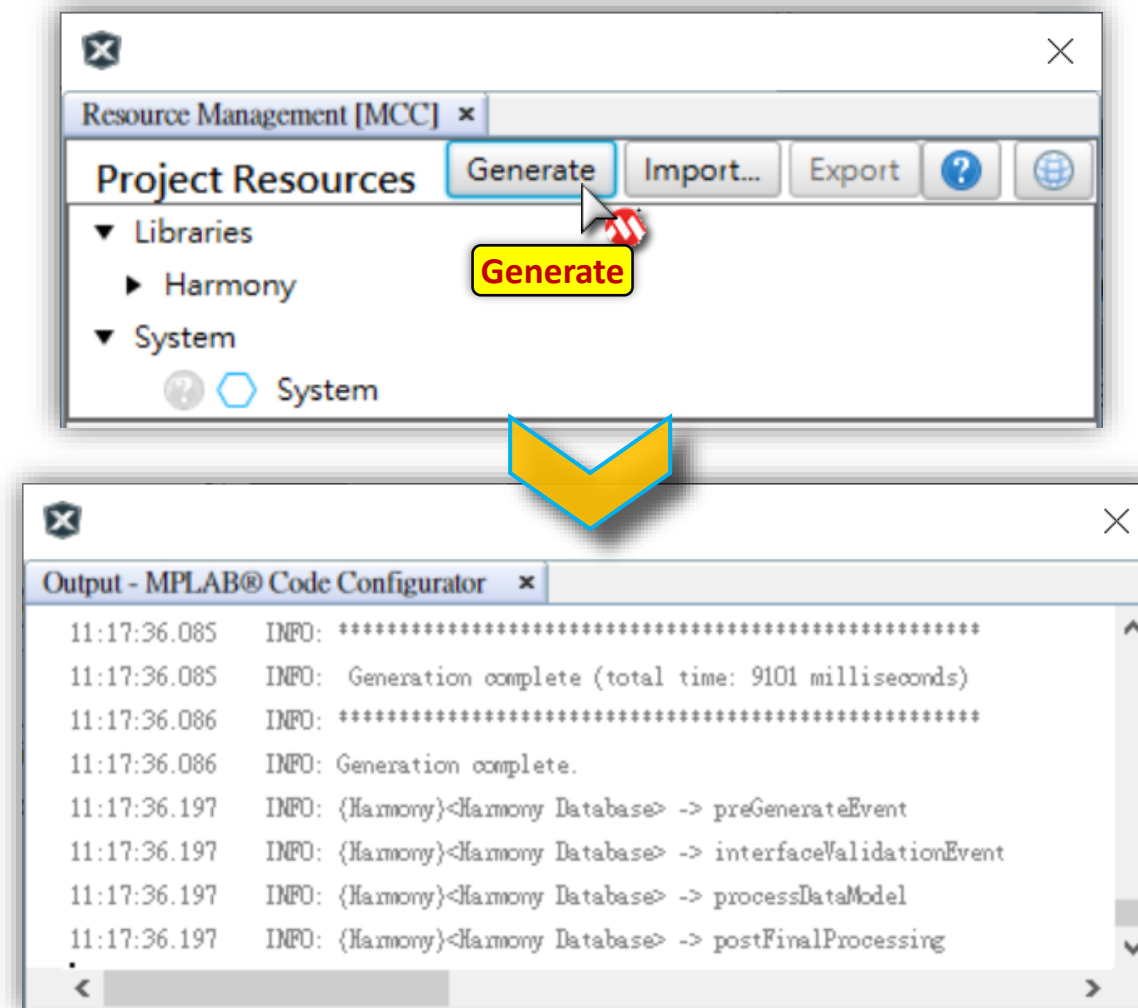
Pin Table Configuration - TCC2

Module	Function	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PA13	GCLK_I	GCLK_I	PA16	TCC2_W
		15	16	17	18	19	20	21	22	23	24	25	26
TCC2	TCC1_WO3												
	TCC2_WO0												
	TCC2_WO1												
	TCC2_WO1												

Lab14 TCC NPWM

Step 4

- Click "**Generate**" Button to Generate your Code.



Lab14 TCC NPWM

Step 5

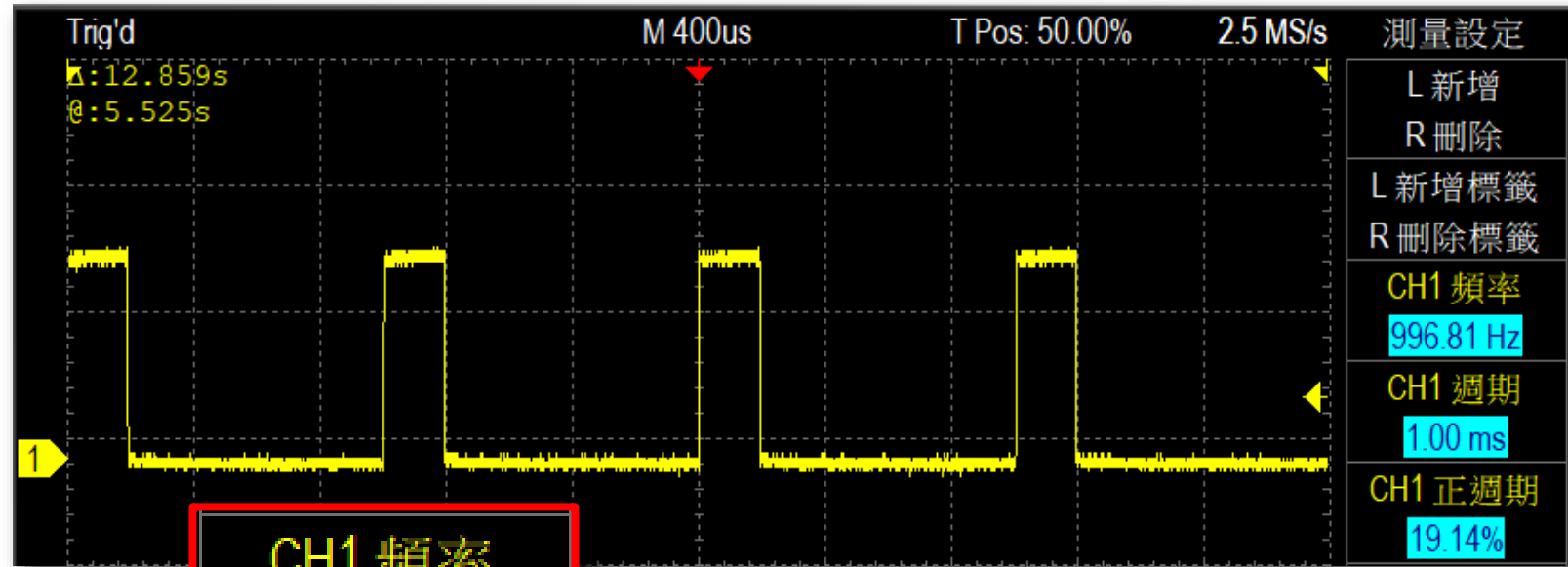
a Add code segment to your main.c

```
...  
  
int main (void)  
{  
    ...  
  
    // TODO 14.01  
    TCC2_PWMStart();  
  
    while( true )  
    {  
        ...  
    }  
  
}
```

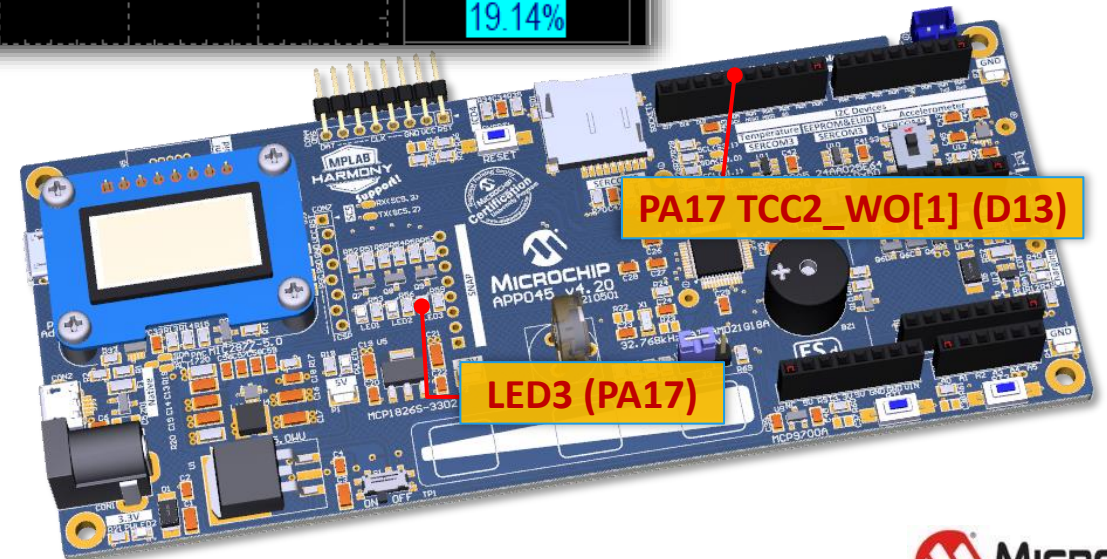
b Program firmware to target board then observe result.

Lab14 TCC NPWM

Result



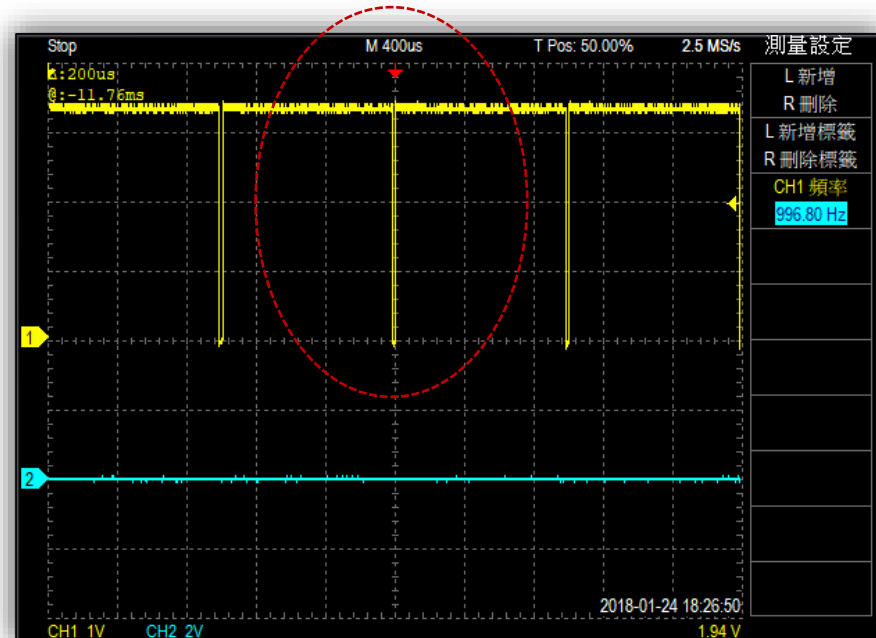
CH1 頻率
996.81 Hz
CH1 週期
1.00 ms
CH1 正週期
19.14%



100% Duty Issue

- How to generate 100% duty PWM ?
 - Rule Review
 - 1.TCCx Counter = CCx -> Output Set to low
 - 2.TCCx Counter = PER -> Output Set to high
- This means CCx = PER must be set for 100% duty output.
Which rule be executed ?
 - Result : Glitch !
- To prevent this issue,
set CCx > PER for 100% duty.

```
if( DutyValue >= PER ) /* 100% duty */  
    TCC2_PWM16bitDutySet( 1, PER + 1 );  
else  
    TCC2_PWM16bitDutySet( 1, DutyValue );
```



Lab15 TCC NPWM ADC Duty Control

- Use **VR1** to adjust **TCC2** PWM duty for **LED3** dimming control.
 - VR1 ADC Value = 0 -> TCC2 PWM Duty 0%
 - VR1 ADC Value = 4095 -> TCC2 PWM Duty 100%

```
if( ADC_IsCompleted )
{
    ADC_IsCompleted = 0;

    // TODO 15.01
    if (ADC_Result[0] >= 4095)
        TCC2_PWM16bitDutySet( 1, TCC2_PWM16bitPeriodGet()+1 );
    else
        TCC2_PWM16bitDutySet( 1, (uint32_t)ADC_Result[0] * TCC2_PWM16bitPeriodGet()/4095 );
    ...
}
```

TCC2_PWM_Peroid = 48000

ADC_Result[0] = 0 ~ 4095

ADC Result[0] / 4095 = 0.0 ~ 1.0

ADC_Result[0] * TCC2_PWM_Peroid / 4095 = 0 ~ 48000

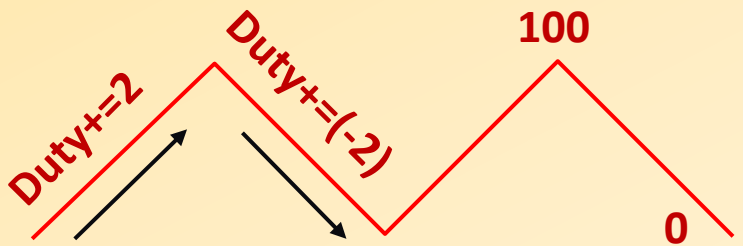
Lab16 TCC NPWM Auto Duty Control

- Use **Timer(TC4)** to control PWM duty for **breathing light effect** on **LED3**.
Duty change interval 50ms with stepping 2% ~ 5%.
(Duty 0% -> Duty 100% -> Duty 0% -> ... repeat)

```
// TODO 16.01
uint8_t Duty = 50;
int8_t DutyDistance = 2;

int main (void)
{
    ...
    if (TC4_HasExpired)
    {
        TC4_HasExpired = 0;
        // TODO 16.02
        Duty += DutyDistance;
        if (Duty >= 100 || Duty <= 0)
            DutyDistance = -DutyDistance;

        if (Duty >= 100)
            TCC2_PWM16bitDutySet( 1, TCC2_PWM16bitPeriodGet() + 1 );
        else
            TCC2_PWM16bitDutySet( 1, ((uint32_t)Duty * TCC2_PWM16bitPeriodGet()) / 100 );
        ...
    }
}
```



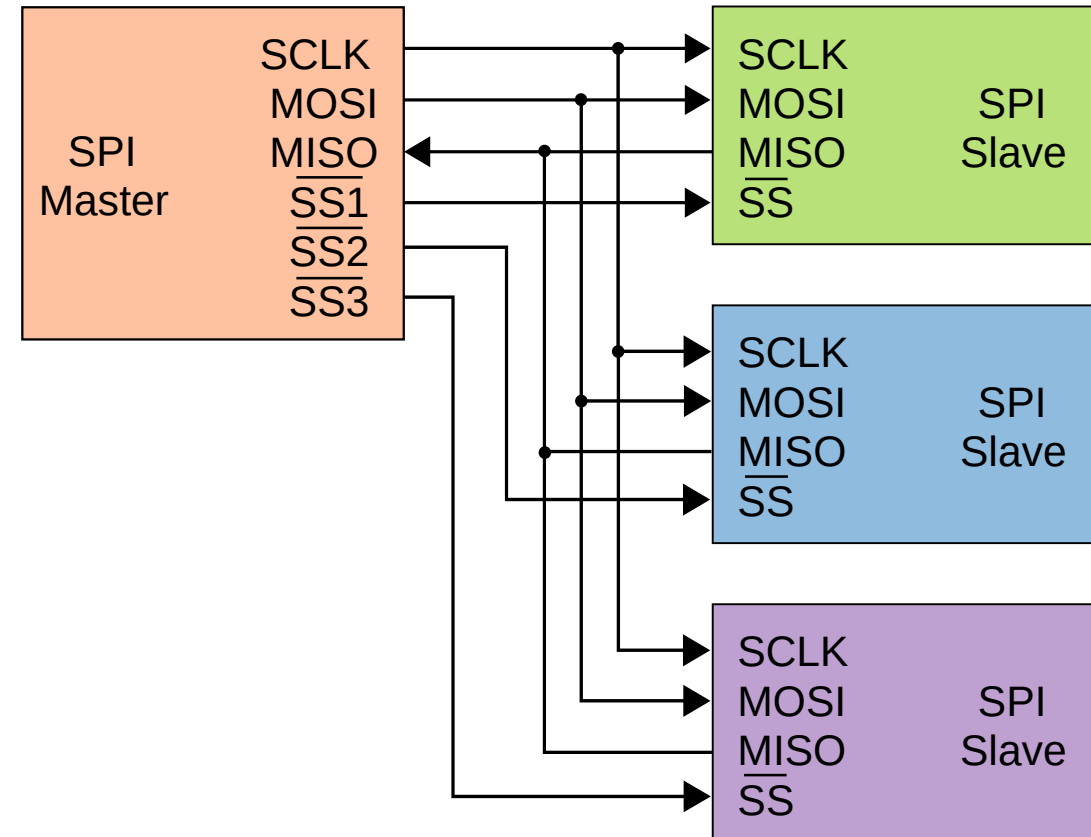
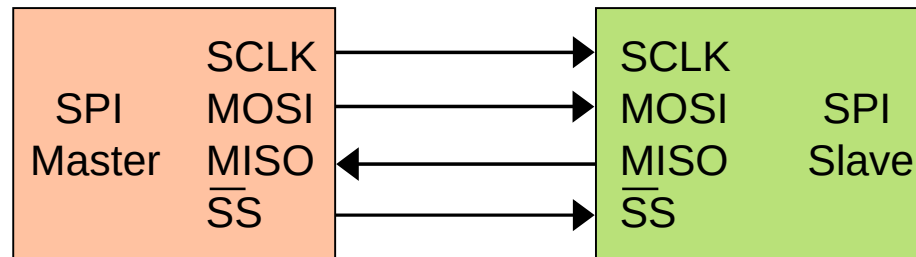
SERCOM-SPI (OLED SSD1306)

- **SSD1306**

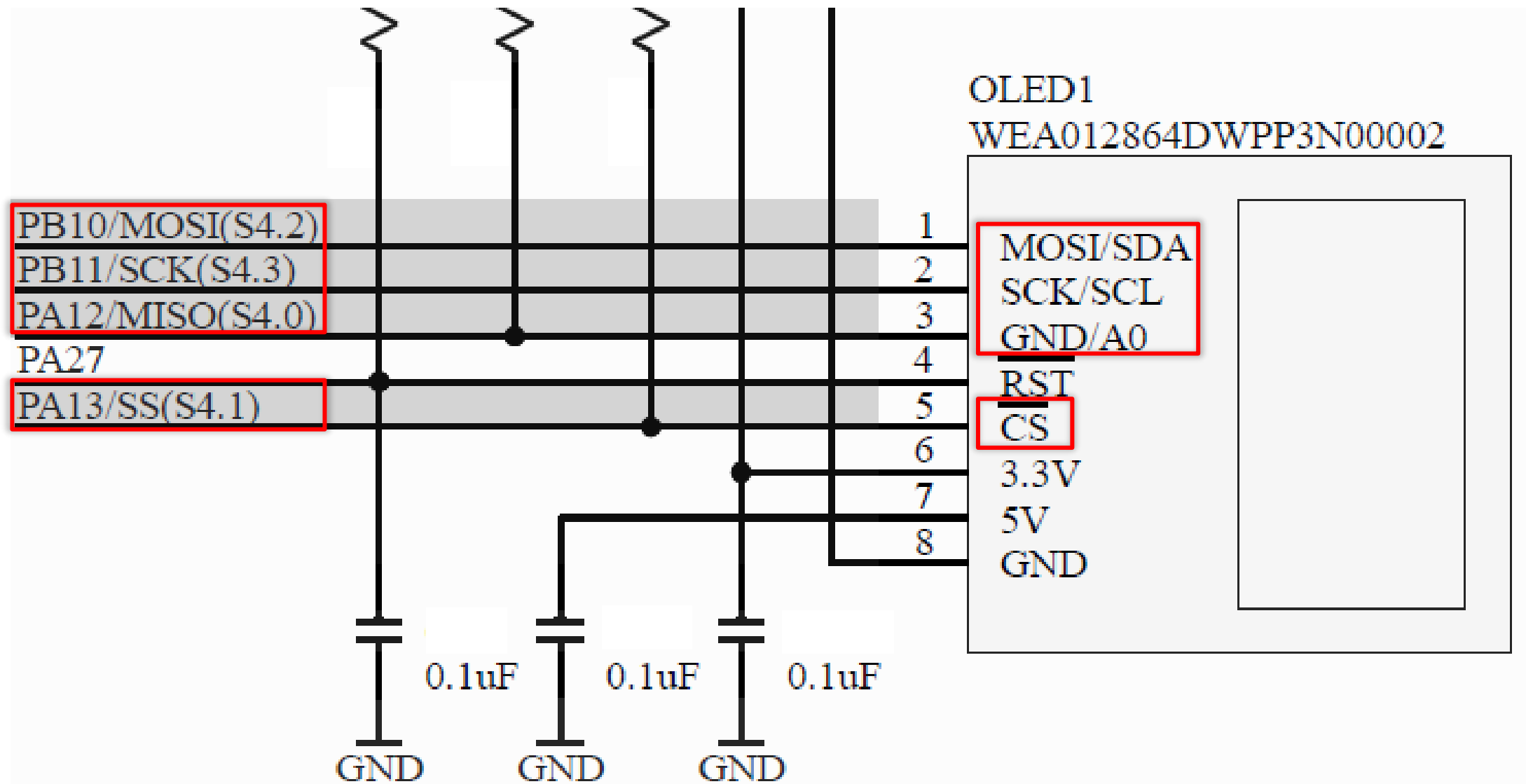
What's SPI

- SPI, Serial Peripheral Interface
- It's a serial communication interface.

Synchronous, inside-board level, short distance communication.



APP045 OLED SSD1306 Schematic



Lab17 SERCOM - SPI SSD1306 OLED

- Add SERCOM4-SPI module to your project.
- Add OLED control function to shows bitmap logo on your OLED display.
- Create your own Logo and replace original one.
- Pre-build Logo Image and Font header file at

.\Appendix\LogoFont\Microchip_Logo.h

.\Appendix\LogoFont\Alphabet_Fonts.h

- Logo image create application at

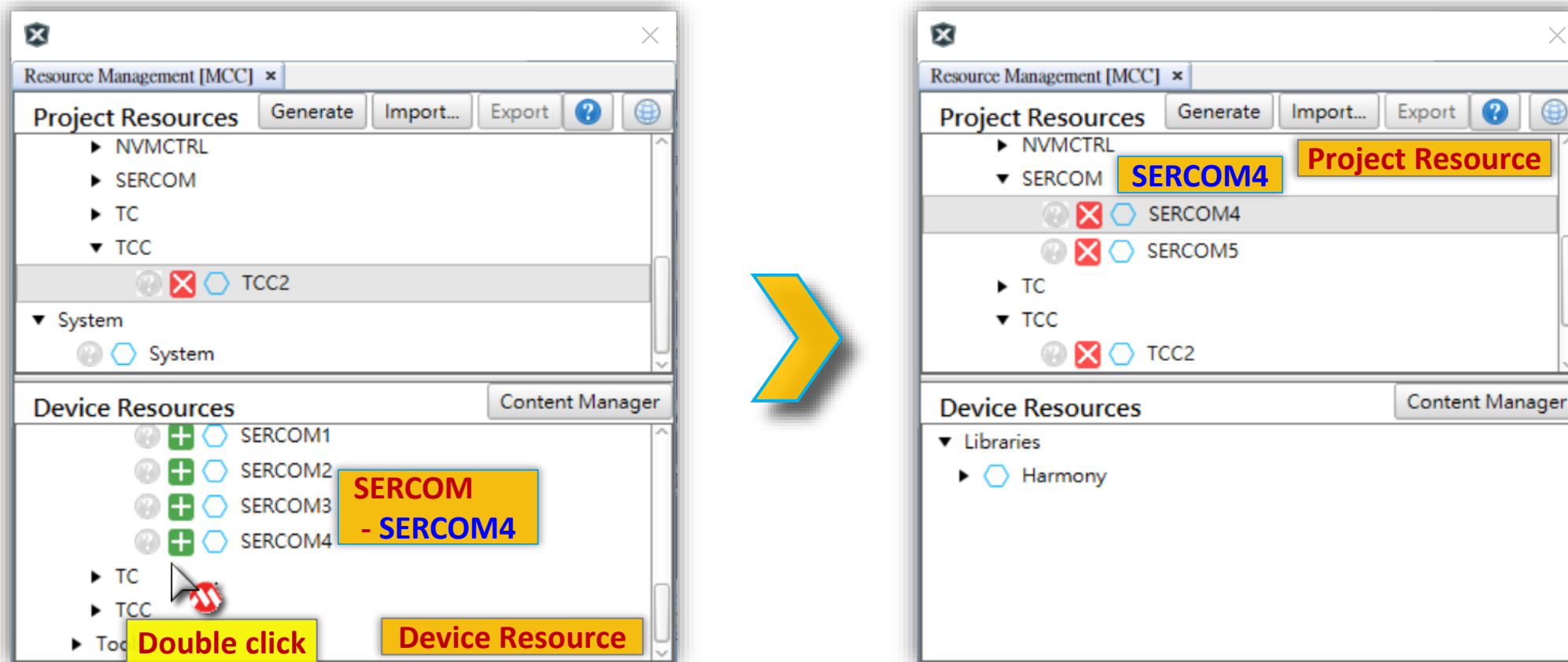
.\Tools\LCDAssistant.exe

- **Let's go!**

Lab17 SERCOM - SPI SSD1306 OLED

Step 1

- Find **SERCOM4** component in Device Resource window.
- Double click **SERCOM4** to add to Project Resource window.



Lab17 SERCOM - SPI SSD1306 OLED

Step 2

- Setup SERCOM4 SPI Configuration.

The screenshot shows the 'Configuration Options' dialog for the SERCOM4 peripheral. The 'SERCOM4' section is expanded, and the 'Select SERCOM Operation Mode' is set to 'SPI Master'. The following options are visible:

- Select SERCOM Operation Mode:** SPI Master (Annotated with 'SPI Master' in a yellow box)
- Enable Interrupts ?** ☐ (Annotated with 'Uncheck !!' in a red box)
- Enable operation in Standby mode** ☐ (Annotated with 'DO on PAD[2], SCK on PAD[3] ...' in a yellow box)
- SPI Data Out Pad:** DO on PAD[2], SCK on PAD[3] and SS on PAD[1] (Annotated with 'DO on PAD[2], SCK on PAD[3] ...' in a yellow box)
- SPI Data In Pad Selection:** SERCOM PAD[0] (Annotated with 'SERCOM PAD[0]' in a yellow box)
- SPI Data Order:** MSB is transferred first (Annotated with 'MSB is transferred first' in a yellow box)
- SPI Speed in Hz:** 1,000,000 (Annotated with '1,000,000' in a yellow box)
- SPI Data Character Size:** 8 bits (Annotated with '8 bits' in a yellow box)
- SPI Clock Phase:** The data is sampled on a leading SCK edge and changed on a trailing SCK edge (Annotated with 'The data is sampled on a leading SCK edge and changed on a trailing SCK edge' in a yellow box)
- SPI Clock Polarity:** SCK is low when idle (Annotated with 'SCK is low when idle' in a yellow box)
- Enable SPI Master Hardware Slave Select** ☐
- SPI Receiver Enable** ☒
- ***SPI Transfer Mode 0 is Selected***** (Annotated with 'Mode 0' in a yellow box)

Lab17 SERCOM - SPI SSD1306 OLED

Step 3

- Pin Table Configuration of SERCOM4

Pin Table

Package: TQFP48

Module	Function	PA10	PA11	VDDIO	GNDIO	PB10	PB11	PA12	PB12	PA13	PB13	PA14	PB14	PA15	PB15	PA16	TCC2_W
	SERCOM3_PAD3																
	SERCOM4_PAD0																
	SERCOM4_PAD1																
SERCOM4	SERCOM4_PAD2																
	SERCOM4_PAD3																
	SERCOM5_PAD0																

Click

Green is Enable

PB10 (SERCOM4_PAD2)
PB11 (SERCOM4_PAD3)

Lab17 SERCOM - SPI SSD1306 OLED

Step 4

- Pin Table Configuration of GPIO

Pin Table

Package: TQFP48

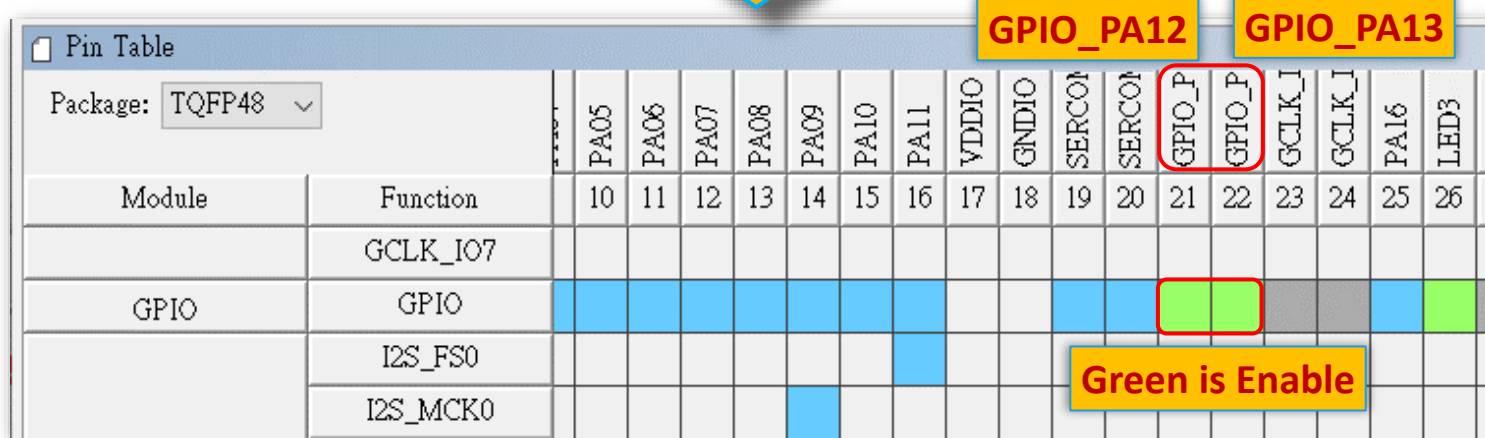
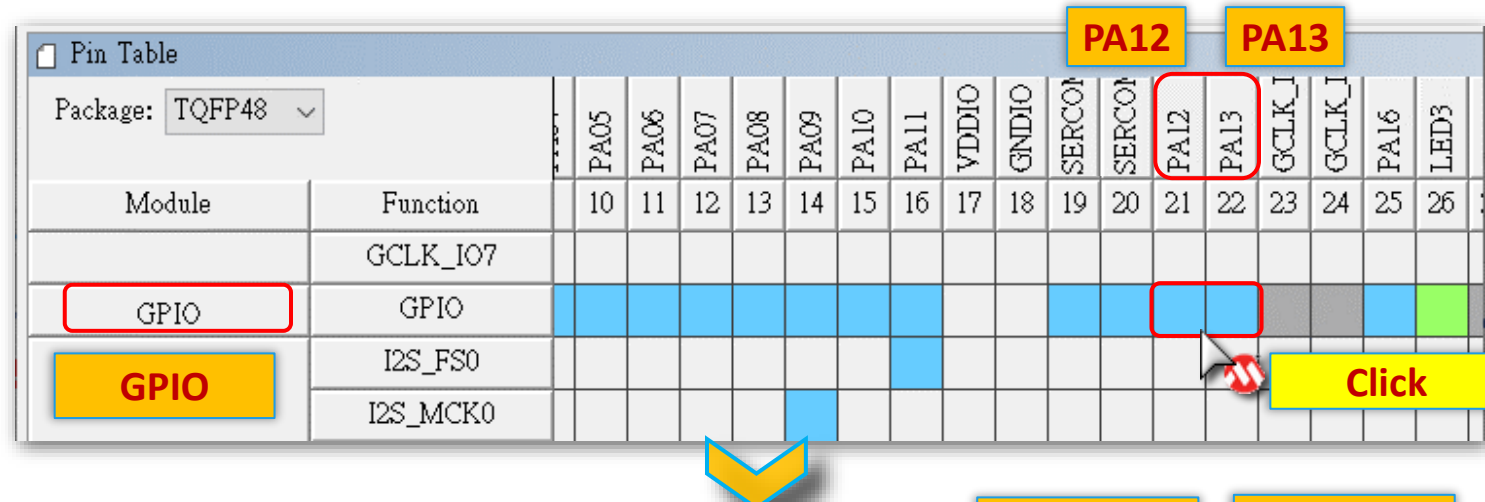
Module	Function	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
	GCLK_IO7																	
GPIO	GPIO																	
GPIO	I2S_FS0																	
	I2S_MCK0																	

PA12 PA13

Click

GPIO_PA12 GPIO_PA13

Green is Enable



Lab17 SERCOM - SPI SSD1306 OLED

Step 5

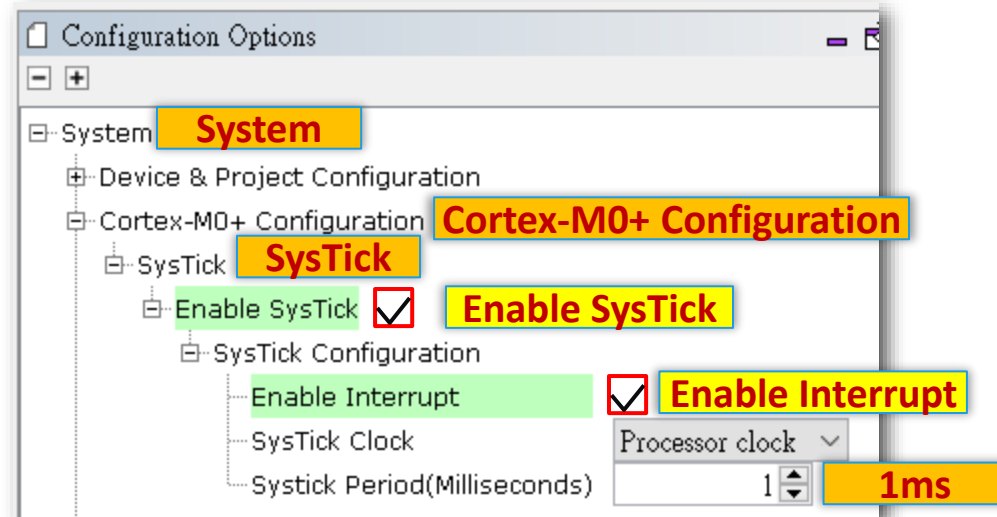
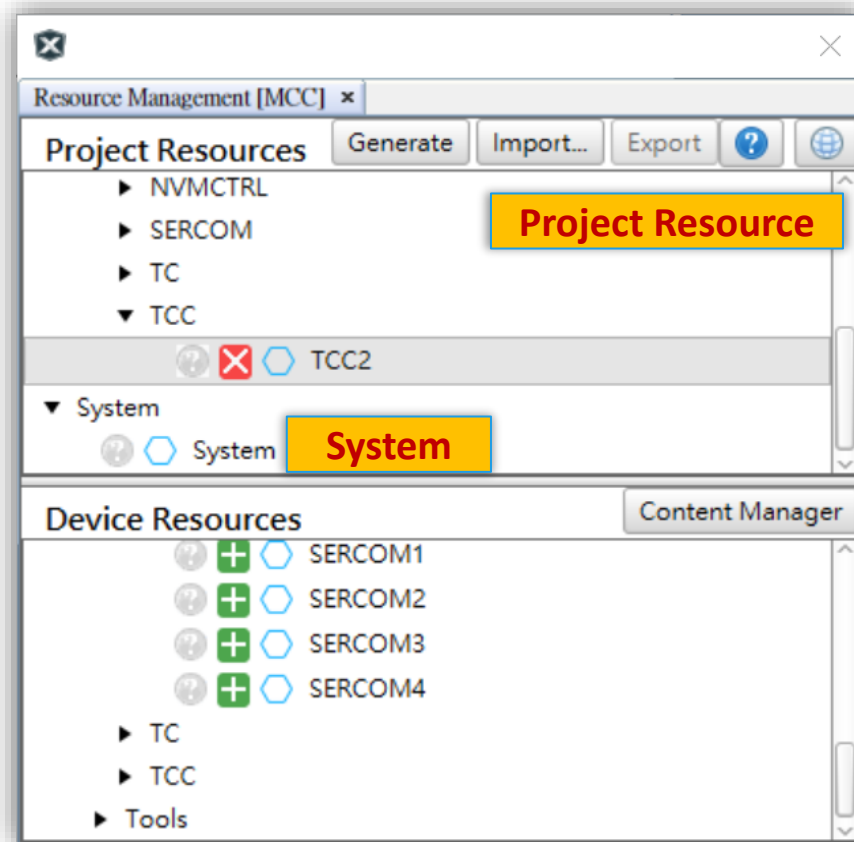
- Pin Setting of GPIO for direction and custom name.

Pin Settings									
Order: Pins		Table View		☒ Easy View					
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
20	PA12	SSD1306_RS_PIN		Digital	Out	High	☐	☐	NORMAL
21	PA12	SSD1306_RS_PIN	GPIO	Digital	Out	High	☐	☐	NORMAL
22	PA13	SSD1306_CS_PIN	GPIO	Digital	Out	High	☐	☐	NORMAL
23	PA13	SSD1306_CS_PIN		Digital	Out	High	☐	☐	NORMAL
24	PA15			Digital	High Impedance	n/a	☐	☐	NORMAL
25	PA16		Available	Digital	High Impedance	Low	☐	☐	NORMAL
26	PA17	LED3	GPIO	Digital	Out	Low	☐	☐	NORMAL
27	PA18		TC3_WO0	Digital	High Impedance	n/a	☐	☐	NORMAL
28	PA19		TC3_WO1	Digital	High Impedance	n/a	☐	☐	NORMAL
29	PA20	LED1	GPIO	Digital	Out	Low	☐	☐	NORMAL
30	PA21	LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL

Lab17 SERCOM - SPI SSD1306 OLED

Step 6

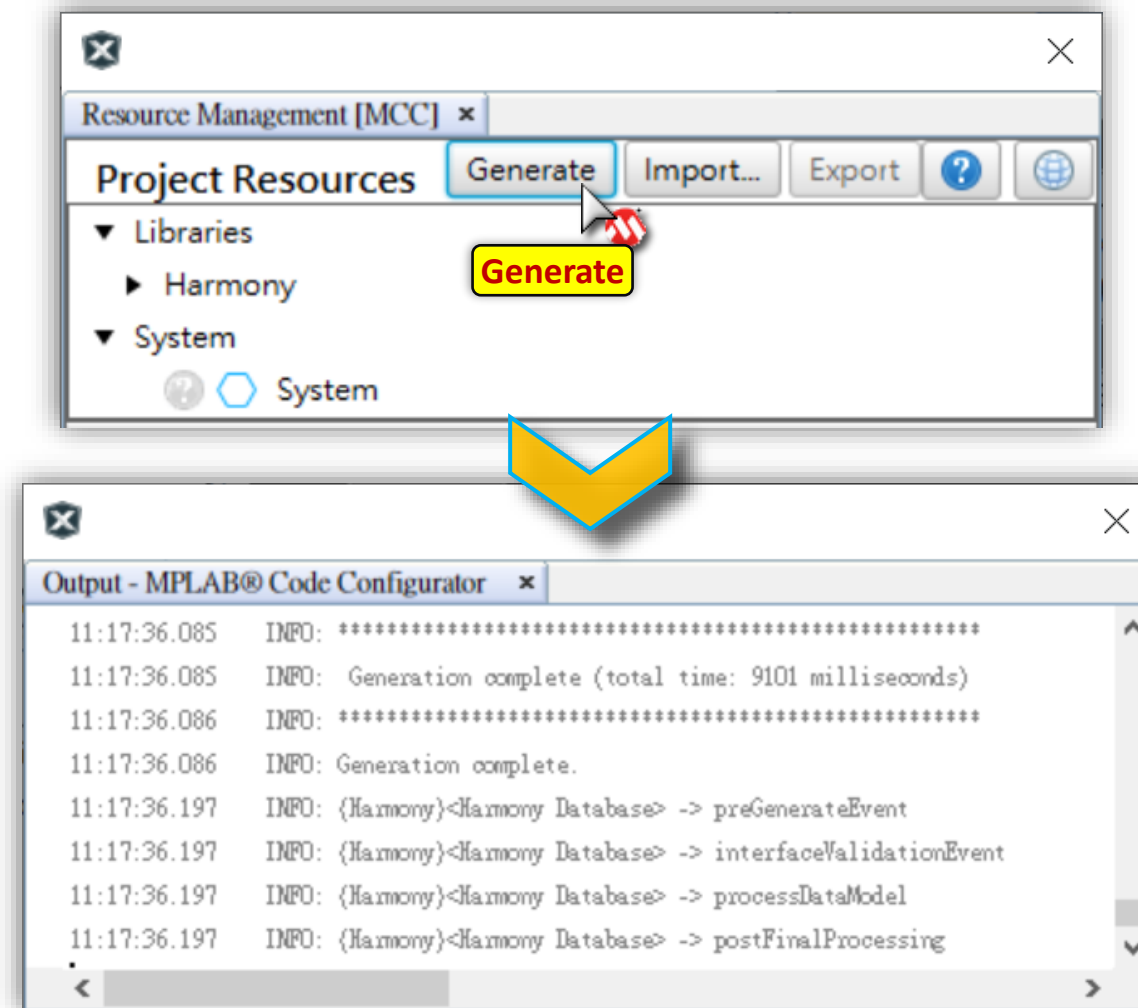
- Select exist module **System** in **Project Resource**.
- Enable **SysTick** for **Delay Function**.



Lab17 SERCOM - SPI SSD1306 OLED

Step 7

- Click "**Generate**" Button to Generate your Code.



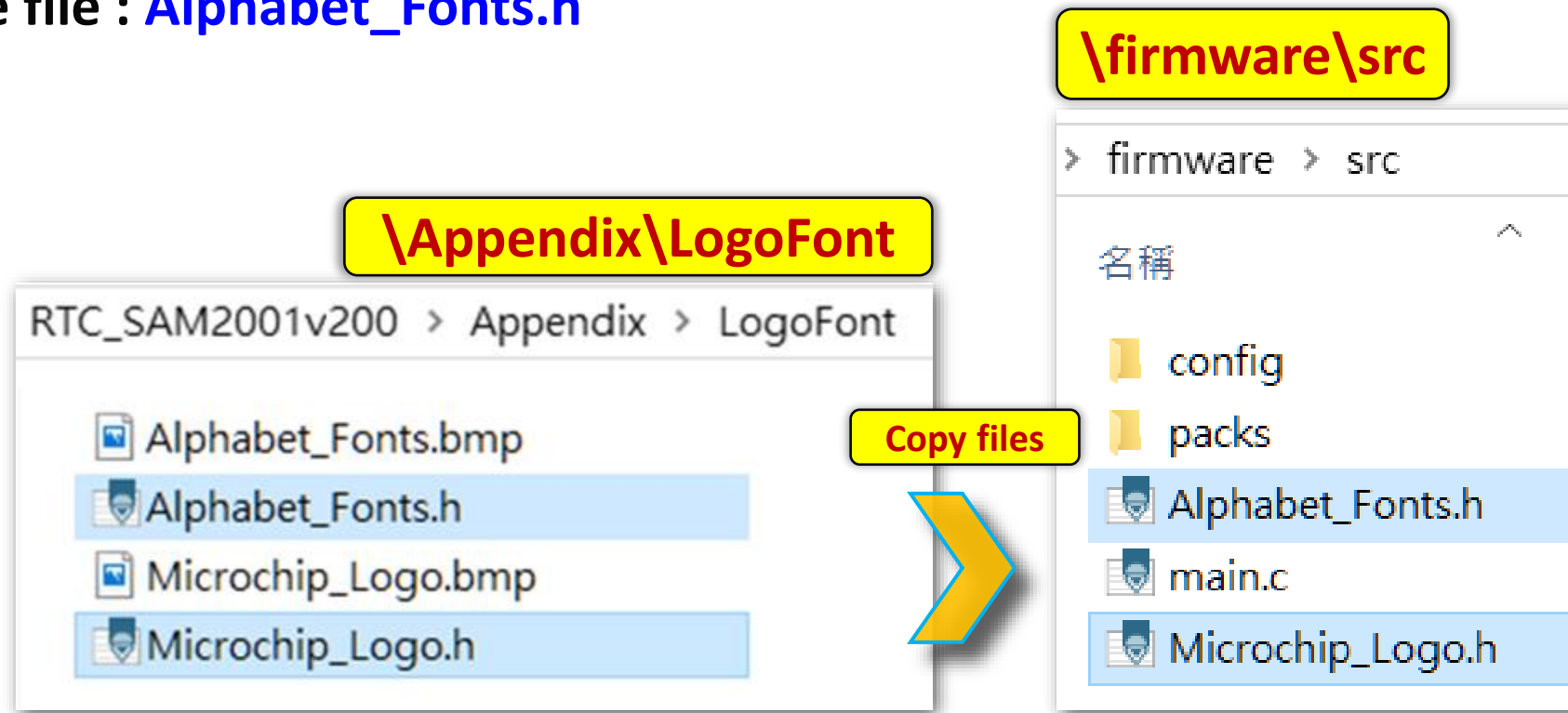
Lab17 SERCOM - SPI SSD1306 OLED

Step 8

- Copy below files to your project's `\firmware\src` folder

Logo image file : `Microchip_Logo.h`

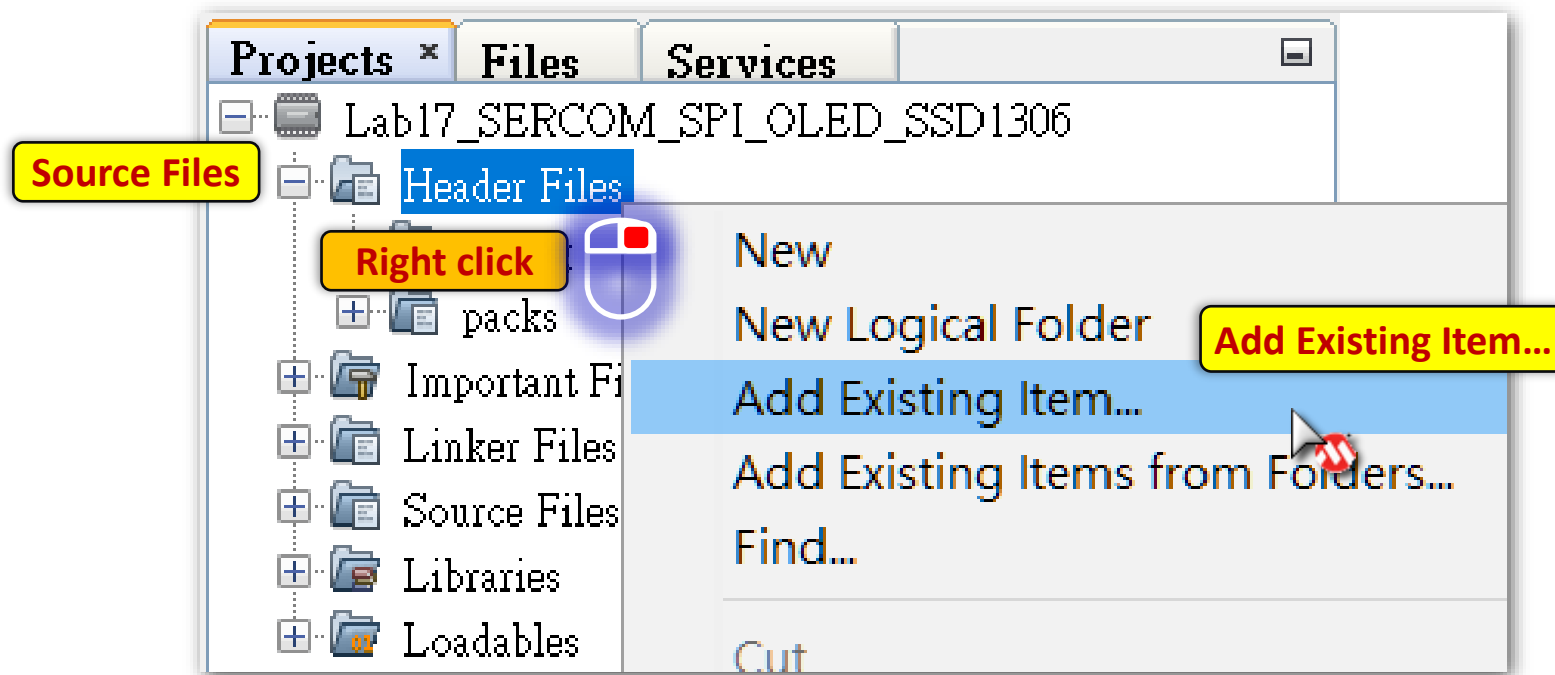
Font image file : `Alphabet_Fonts.h`



Lab17 SERCOM - SPI SSD1306 OLED

Step 9

- Add pre-build logo image header files into project
Right click **Header Files** > **Add Existing item...**



Lab17 SERCOM - SPI SSD1306 OLED

Step 10

- Select folder in **\firmware\src**
- Add **Alphabet_Fonts.h** and **Microchip_Logo.h** into project.



Alphabet_Fonts.h
Microchip_Logo.h
added in \Header Files



Lab17 SERCOM - SPI SSD1306 OLED

Step 11

a Add code segment to your project

```
// TODO 17.01
#include "Microchip_Logo.h"
#include "Alphabet_Fonts.h"
...
// TODO 17.02
#define LCM_WIDTH 128
#define LCM_HEIGHT 64
#define FONT_WIDTH 7
#define FONT_HEIGHT 8
uint8_t LogoX, LogoY;
unsigned char LCM_Blank[1024];
char Alphabet[95] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz~!@#$%^&()[]{}_\\";
                    |;:,.`'\ "<>+-*./=? ";

const uint8_t LCM_InitCMD[] = {
    0xAE, 0xD5, 0x80, 0xA8, 0x3F, 0xD3, 0x00, 0x40, 0x8D, 0x14, 0x20, 0x02, 0xA1,
    0xC8, 0xDA, 0x12, 0x81, 0xCF, 0xD9, 0xF1, 0xDB, 0x40, 0xA4, 0xA6, 0xAF
};
```

Lab17 SERCOM - SPI SSD1306 OLED

Step 12

a Add code segment to your project

```
// TODO 17.02
...
void LCM_SetCursor( uint8_t x, uint8_t y )
{
    uint8_t OUT[3] = { 0x0F & x, 0x10 + (x >> 4), 0xB0 + y };
    SSD1306_RS_PIN_Clear();
    SERCOM4_SPI_Write(( void* )OUT, 3);
}

void LCM_DrawBitmap( uint8_t sx, uint8_t sy, uint8_t width,
                    uint8_t height, const unsigned char *byte )
{
    for( int y = 0; y < (height/8); y++ )
    {
        LCM_SetCursor(sx, sy + y);
        SSD1306_RS_PIN_Set();
        SERCOM4_SPI_Write(( void* )byte, (width - sx));
        byte += width;
    }
}
```

Lab17 SERCOM - SPI SSD1306 OLED

Step 13

a Add code segment to your project

```
// TODO 17.02
...
void LCM_DrawFont( uint8_t sx, uint8_t row, char ch )
{
    int chIdx = 0;

    while( ch != Alphabet[chIdx] ) { chIdx++; }
    LCM_SetCursor(sx, row);
    SSD1306_RS_PIN_Set();
    SERCOM4_SPI_Write( ( void* )(Alphabet_Font+(chIdx*FONT_WIDTH)),
                        FONT_WIDTH );
}

void LCM_DrawString( uint8_t sx, uint8_t row, char *str )
{
    for( int x=0 ; x<strlen(str); x++ )
    {
        LCM_DrawFont( sx+(x*FONT_WIDTH), row, str[x] );
    }
}
```


Lab17 SERCOM - SPI SSD1306 OLED

Step 14

a Add code segment to your project

```
...
int main (void)
{ ...
    TCC2_PWMStart();

    // TODO 17.03
    SYSTICK_TimerStart();
    SSD1306_CS_PIN_Clear();
    SSD1306_RS_PIN_Clear();
    SERCOM4_SPI_Write(( void* )LCM_InitCMD, sizeof (LCM_InitCMD));
    memset(LCM_Blank, 0, 1024);
    LCM_DrawBitmap(0, 0, LCM_WIDTH, LCM_HEIGHT, LCM_Blank);
    LogoX = 127;
    LogoY = 0;
    for( i = 0; i < 128; i += 2 )
    {
        LCM_DrawBitmap(LogoX - i, LogoY, LCM_WIDTH, LCM_HEIGHT, Microchip_Logo);
        SYSTICK_DelayMs((i * i) / 400);
    }
    ...
}
```

Lab17 SERCOM - SPI SSD1306 OLED

Step 15

a Add code segment to your project

```
int main (void)
{ ...
  // TODO 17.03
  ...
  SYSTICK_DelayMs(1000);
  memset(LCM_Blank, 0, 1024);
  LCM_DrawBitmap(0, 0, LCM_WIDTH, LCM_HEIGHT, LCM_Blank);
  sprintf( (char *)USARTRTxBuffer, "SysClock %ld.-%ldMHz",
          SYSTICK_TimerFrequencyGet()/1000000,
          (SYSTICK_TimerFrequencyGet()%1000000)/1000 );
  LCM_DrawString( 0, 0, (char *)USARTRTxBuffer );
  sprintf( (char *)USARTRTxBuffer, "TC3 period %ldms",
          TC3_Timer16bitPeriodGet()*1000/TC3_TimerFrequencyGet() );
  LCM_DrawString( 0, 1, (char *)USARTRTxBuffer );
  sprintf( (char *)USARTRTxBuffer, "TC4 period %ldms",
          TC4_Timer16bitPeriodGet()*1000/TC4_TimerFrequencyGet() );
  LCM_DrawString( 0, 2, (char *)USARTRTxBuffer );
  sprintf( (char *)USARTRTxBuffer, "Received Data :" );
  LCM_DrawString( 0, 7, (char *)USARTRTxBuffer );
  ...
}
```

Lab17 SERCOM - SPI SSD1306 OLED

Step 16

a Add code segment to your project

```
while ( true )
{ ...
  if (TC3_ HasExpired)
  { ...
    LED1_Toggle();

    // TODO 17.04
    sprintf( (char *)USARTRTxBuffer, "Hello World!");
    LCM_DrawString( 0, 3, (char *)USARTRTxBuffer );
    ...
  }

  if (TC4_ HasExpired)
  { ...
    // TODO 17.05
    sprintf( (char *)USARTRTxBuffer, "PWM Duty : %3d%%", Duty);
    LCM_DrawString( 0, 4, (char *)USARTRTxBuffer );

    LED2_Toggle();
  }
}
```

Lab17 SERCOM - SPI SSD1306 OLED

Step 17

a Add code segment to your project

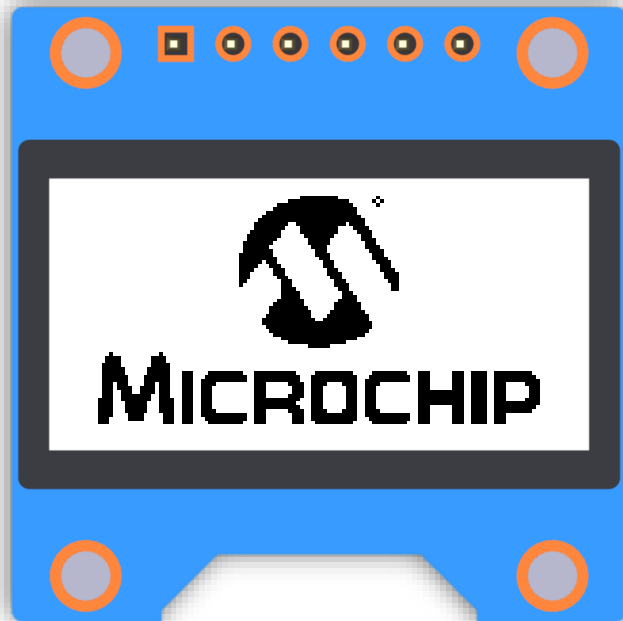
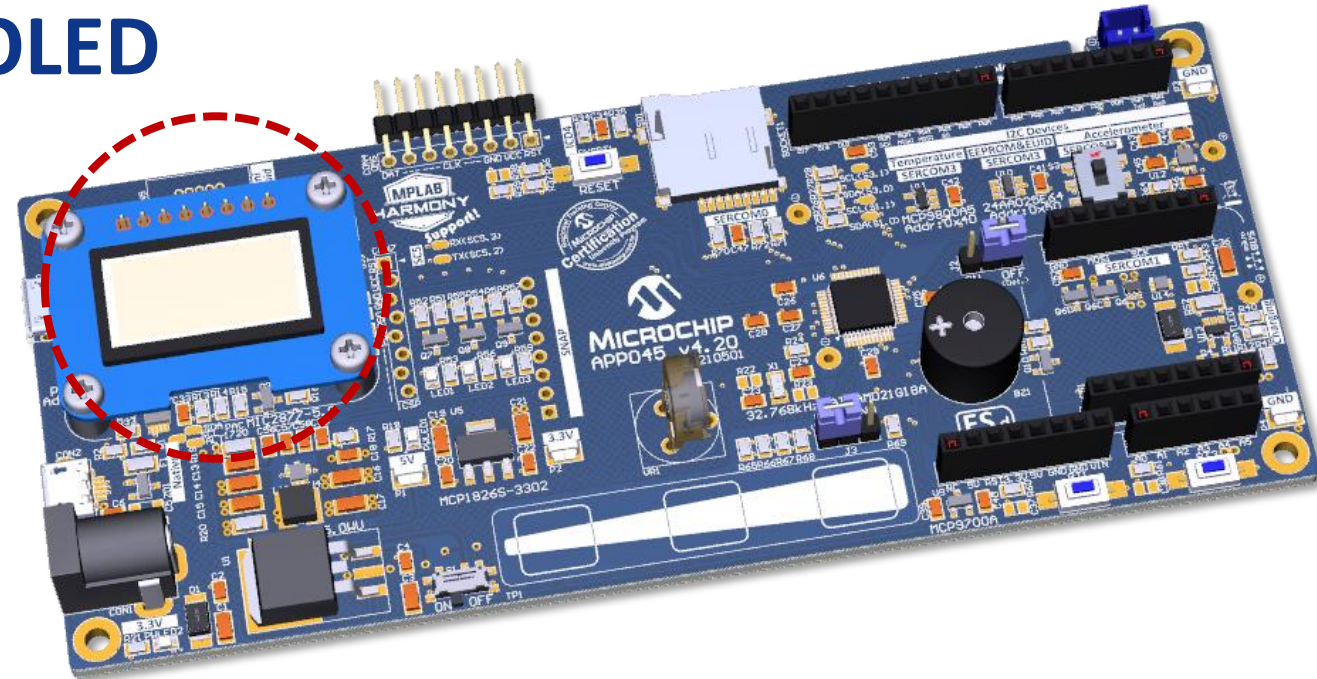
```
while ( true )
{ ...
  if (USART5_IsReceived)
  { ...
    // TODO 17.06
    sprintf( (char *)USARTRTxBuffer, "Received Data : %1c",
                                                    USART5_ReceiveData[0] );

    LCM_DrawString( 0, 7, (char *)USARTRTxBuffer );
    ...
  }

  if (ADC_IsCompleted)
  { ...
    // TODO 17.07
    sprintf( (char *)USARTRTxBuffer, "VR1  : %4d", ADC_Result[0]);
    LCM_DrawString( 0, 5, (char *)USARTRTxBuffer );
    sprintf( (char *)USARTRTxBuffer, "Temp : %4d", ADC_Result[1]);
    LCM_DrawString( 0, 6, (char *)USARTRTxBuffer );
    ...
  }
}
```

Lab17 SERCOM - SPI SSD1306 OLED

Result



1 Sec.

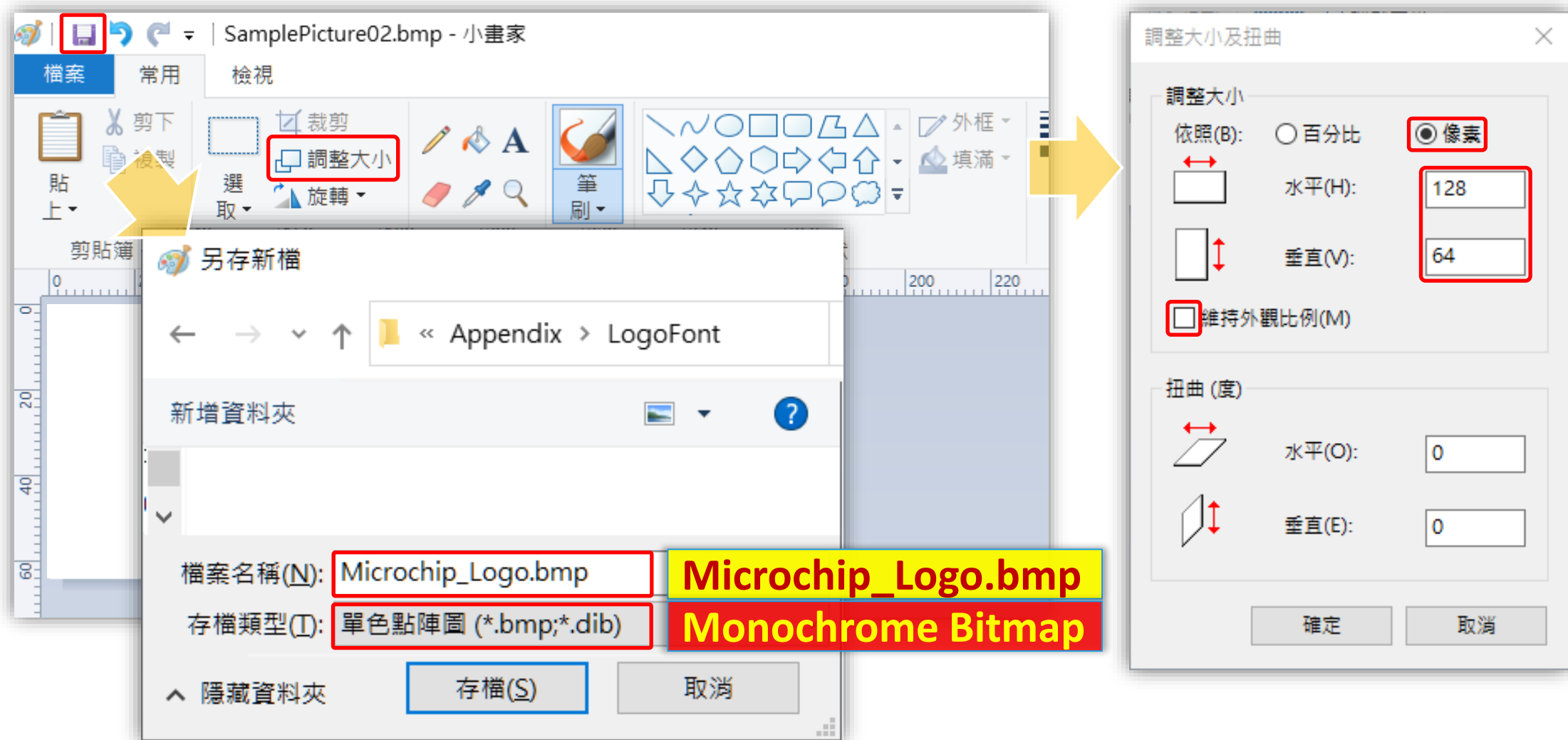


Lab17 SERCOM - SPI SSD1306 OLED

- To create your personally welcome page (logo) !
- **Let's go!**

Lab17 SERCOM - SPI SSD1306 OLED

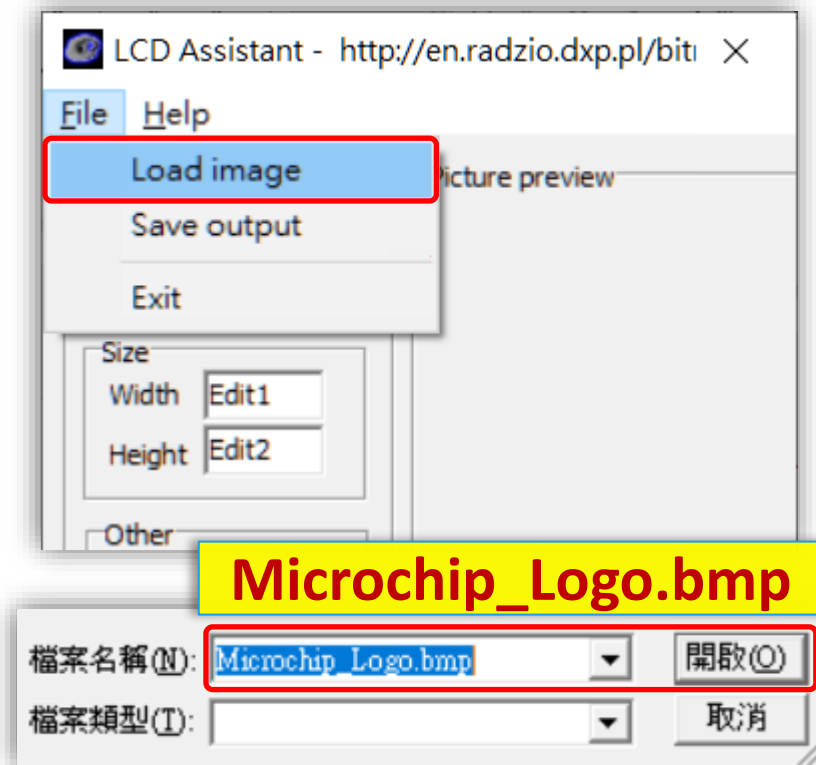
Create your own logo



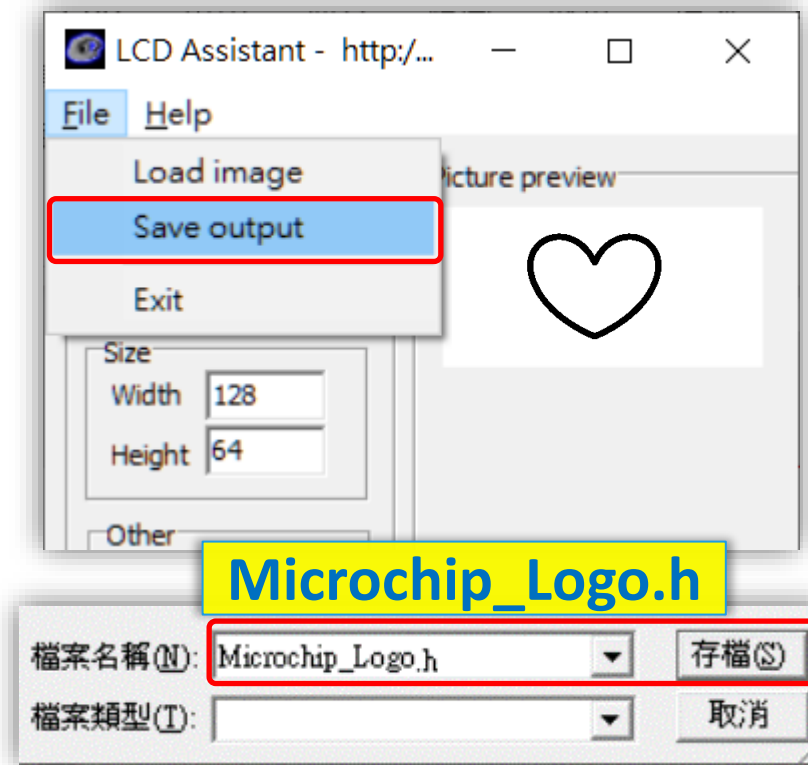
Lab17 SERCOM - SPI SSD1306 OLED

Create your own logo

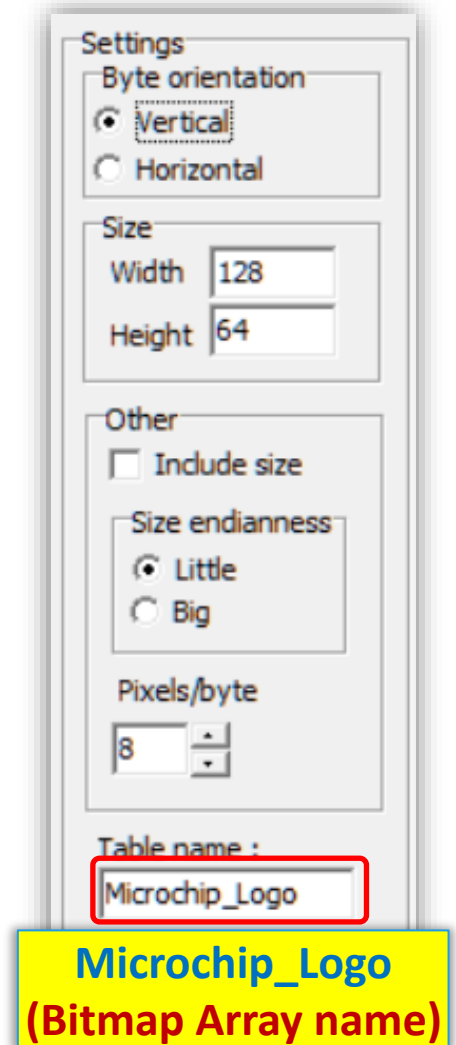
- Execute `\Tools\LCDAssistant.exe`
Load `Microchip_Logo.bmp` and Save output to `Microchip_Logo.h`



Microchip_Logo.bmp



Microchip_Logo.h

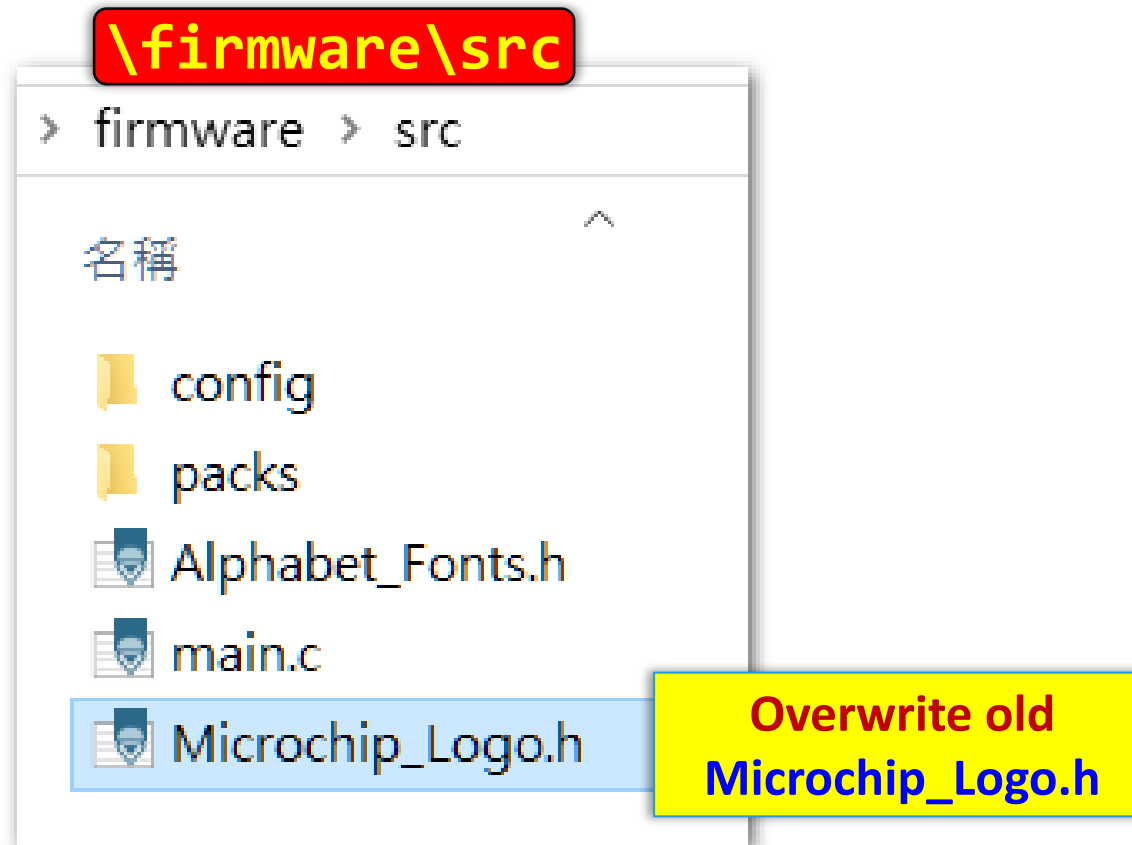


Microchip_Logo
(Bitmap Array name)

Lab17 SERCOM - SPI SSD1306 OLED

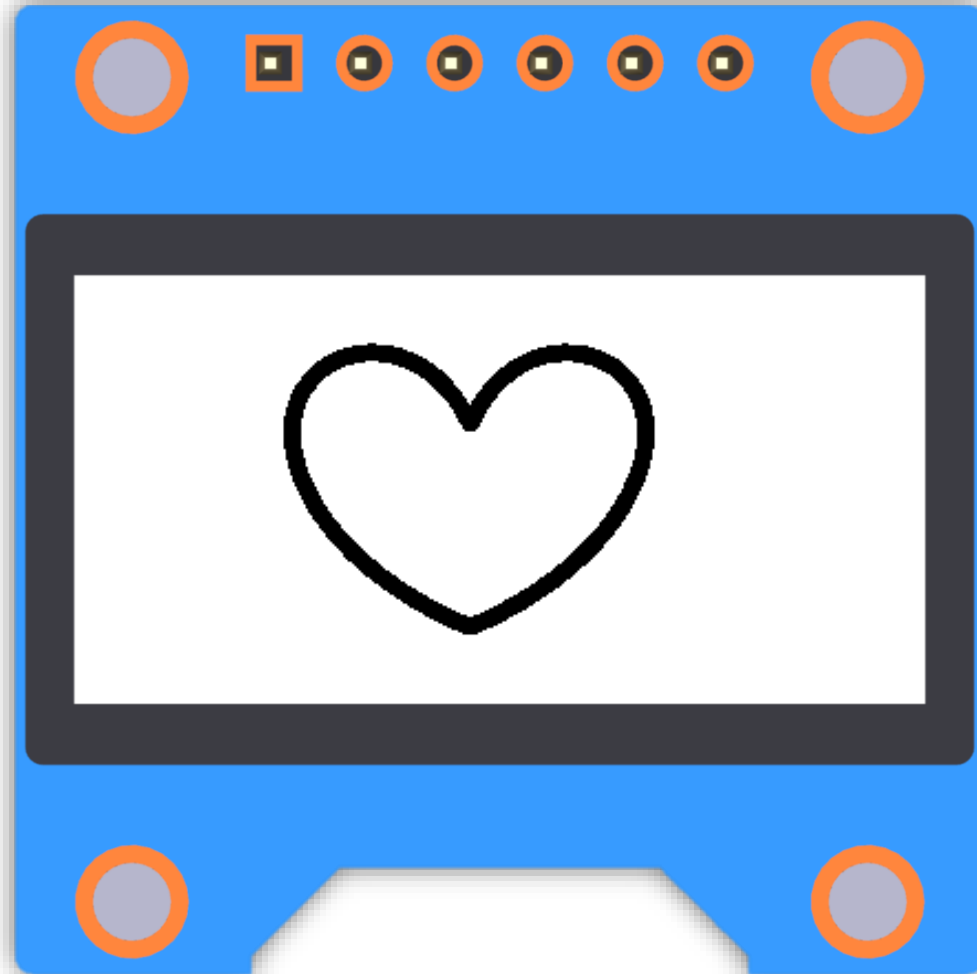
Create your own logo

- Copy new **Microchip_Logo.h** to overwrite old one in folder **\firmware\src**



Lab17 SERCOM - SPI SSD1306 OLED

Create your own logo result



Trademarks

- **The following are registered trademarks of Microchip in the U.S.A. and other countries:** The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA.
- **The following are registered trademarks of Microchip in the U.S.A:** AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL.
- **The following are registered trademarks of Microchip in other countries:** BeaconThings, GestIC, Silicon Storage Technology.
- **The following are trademarks of Microchip in the U.S.A. and other countries:** Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA.
- **The following is a service mark of Microchip in the U.S.A.:** SQTP.
- **The following are logos of Microchip in the U.S.A. and other countries:** The Adaptec logo, Frequency on Demand, Silicon Storage Technology, Symmcom, and Trusted Time
- **The following is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries:** GestIC