CAE空中教室 SAM2001系列
SAMD21 ARM Cortex-M0+ Microcontroller & Harmony v3
SAM2001 v2.00

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

CAE Taiwan Team
Microchip Technology Inc.

2021 April

SMART | CONNECTED | SECURE

Major Course (1/4)

- SAMD21 ARM Cortex-M0+ Microcontroller Introduction
- MPLAB X IDE and MHC Development Tools Introduction
- SAMD21 EVB APP045 v2.00 Introduction
- Getting Started with First Project
  - Lab0 First Project
- MPLAB Harmony Configurator (MHC) Interface introduction
  - Study Advanced Features
- PORT I/O Architecture
  - Lab1 PORT Output
  - Lab2 PORT Input and Output
- Timer/Counter, TC Architecture
  - Lab3 TC MFRQ Polling
  - Lab4 TCs MFRQ Polling

Major Course (2/4)

- NVIC Architecture
  - Lab5 TC MFRQ Interrupt Callback
  - Lab6 TCs MFRQ Interrupt Callback
  - Study Advanced Features
- Clock System Architecture
  - Lab7 System Clock XOSC32K
  - Lab8 System Clock DFLL48M
  - Study Advance Features
- 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
  - Study Advanced Features

Major Course (3/4)

- High Speed ADC Architecture
  - 💻 Lab12 ADC Polling Software Trigger
  - 🎯 Study Advanced Features
- ADC Interrupt Callback and Pin Scan
  - ➤ ADC Pin Scan
  - 💻 Lab13 ADC Pin Scan
  - 🎯 Study Advanced Features
  - 🎯 ADC Advanced Application
    (Event System / DMA / MPLAB Data Visualizer)
- TCC – PWM Architecture
  - 💻 Lab14 TCC NPWM
  - 💻 Lab15 TCC NPWM ADC Duty Control
  - 💻 Lab16 TCC NPWM Auto Duty Control
  - 🎯 Study Advanced Features

Major Course (4/4)

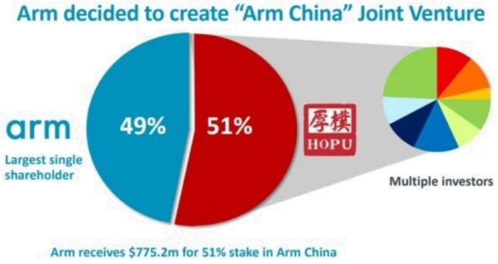
- SERCOM-SPI (OLED SSD1306)
  - 💻 Lab17 SERCOM - SPI SSD1306 OLED

SAMD21 ARM Cortex-M0+ Microcontroller Introduction

ARM's history

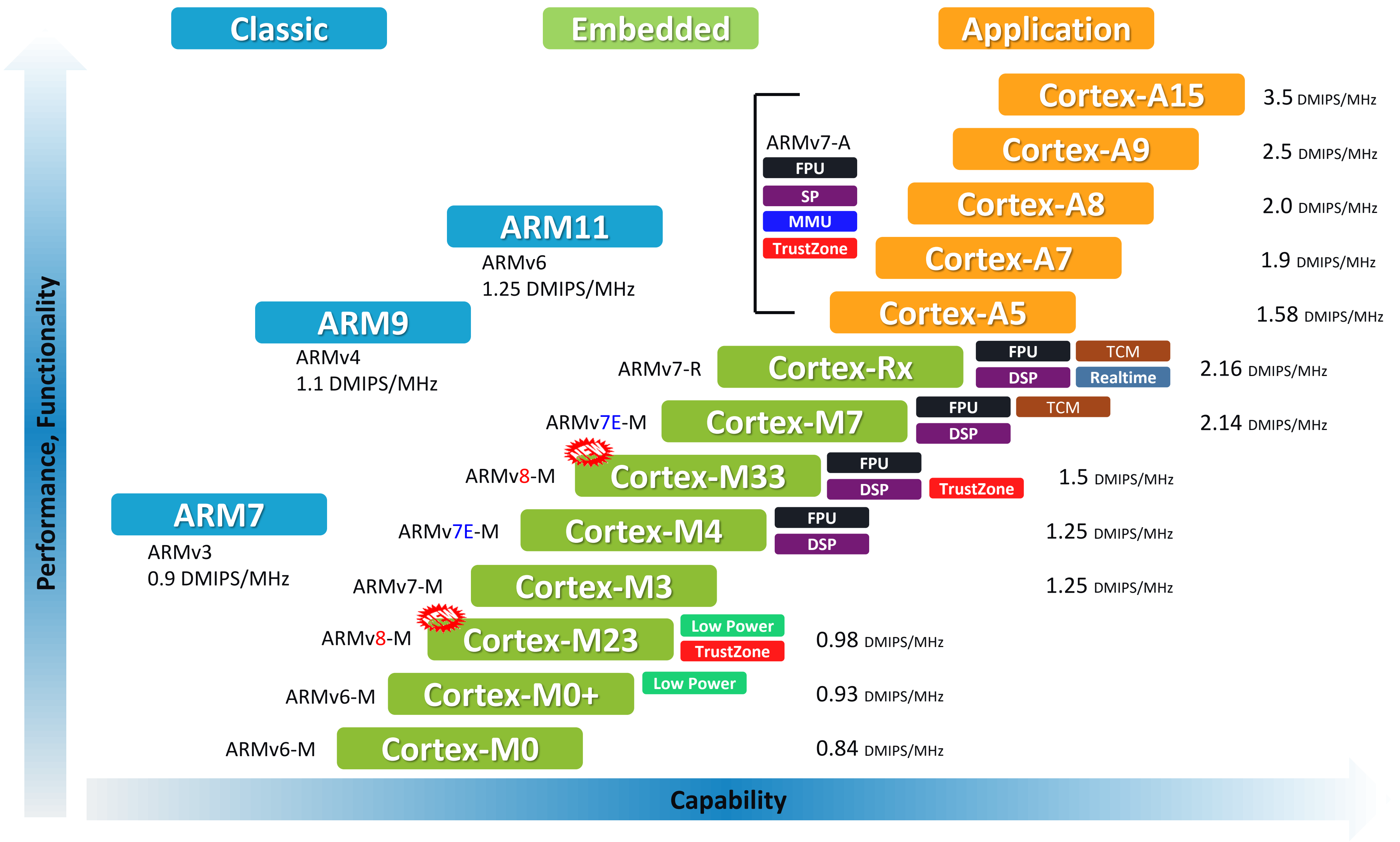
- **ARM**, previously **A**dvanced **R**ISC **M**achine, is a family of reduced instruction set computing (RISC) architectures for computer processors. Arm Holdings develops the architecture and licenses it to other companies, who design their own products that implement systems-on-chips (SoC) and systems-on-modules (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.

ARM® | arm = SoftBank



ARM in Partnership

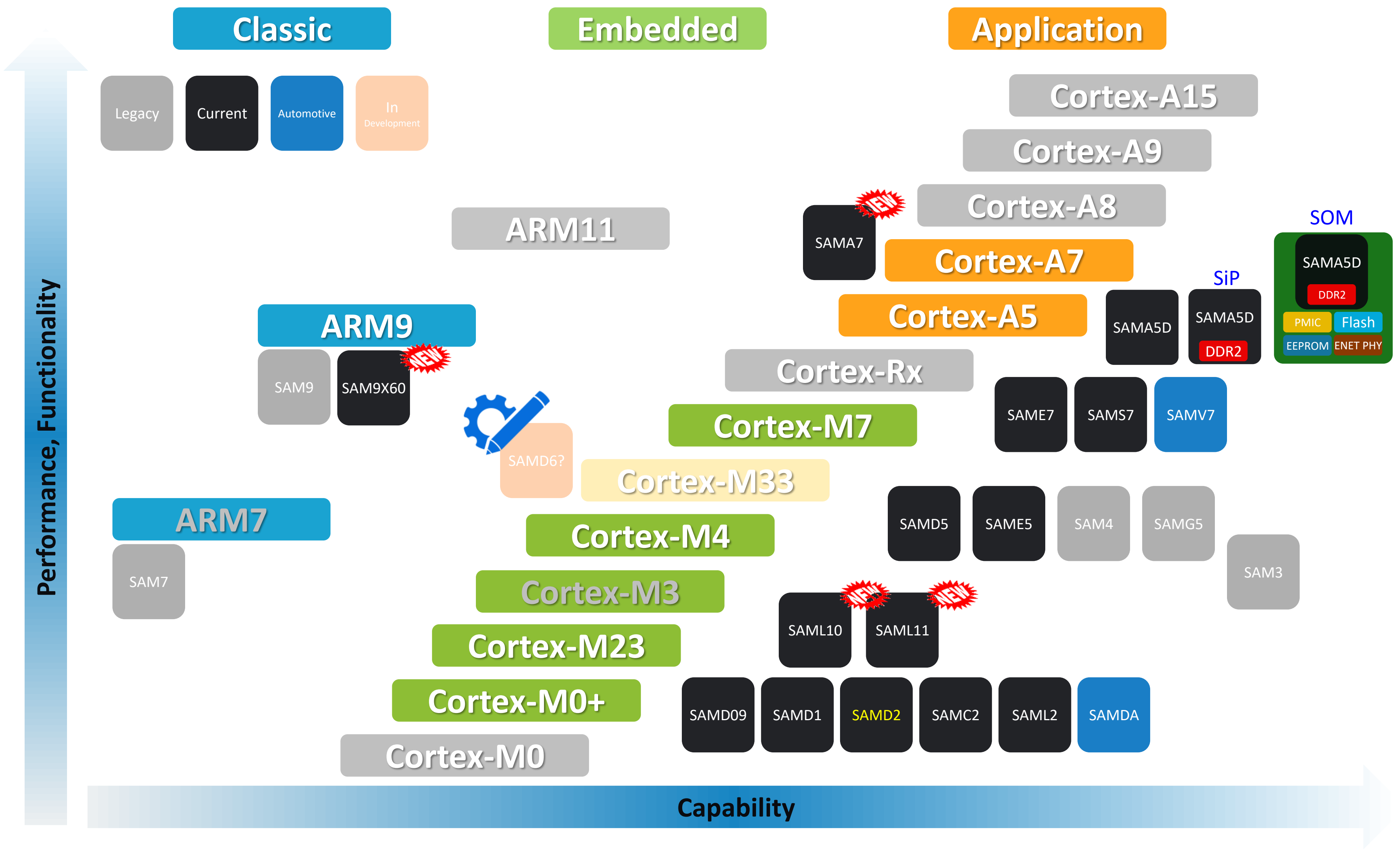
ARM core MCU/MPU brief



MICROCHIP

ARM core MCU/MPU success

Microchip ARM Core Products



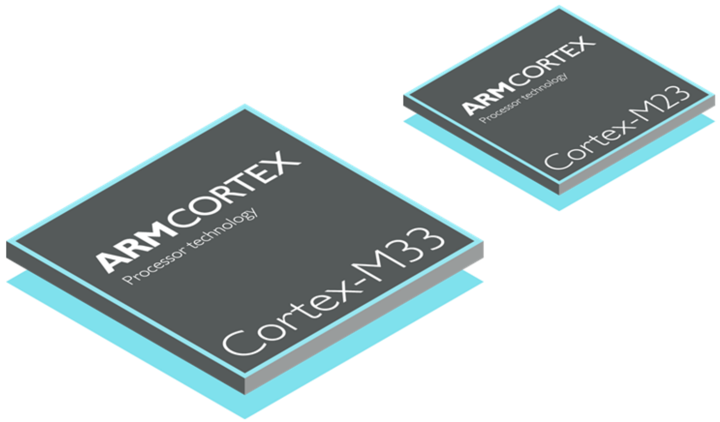
MICROCHIP

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.)

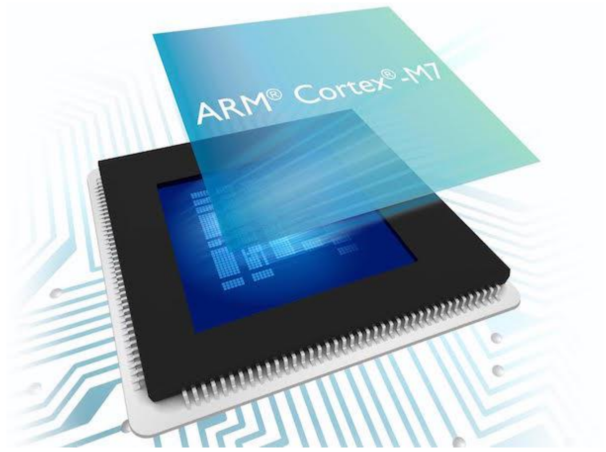
- **Cortex-M4 (New generation)**
  - SAMD5x
    - Support float point FPU, crypto engine, USB , low power.
  - SAME5x
    - FPU, crypto engine, USB, Ethernet, CAN-FD, Low power.
- **Cortex-M23**
  - SAML1x
    - Very low power, support TrustZone.
- **Cortex-M33 (Coming soon)**
  - SAMD6x
    - High performance,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



MICROCHIP

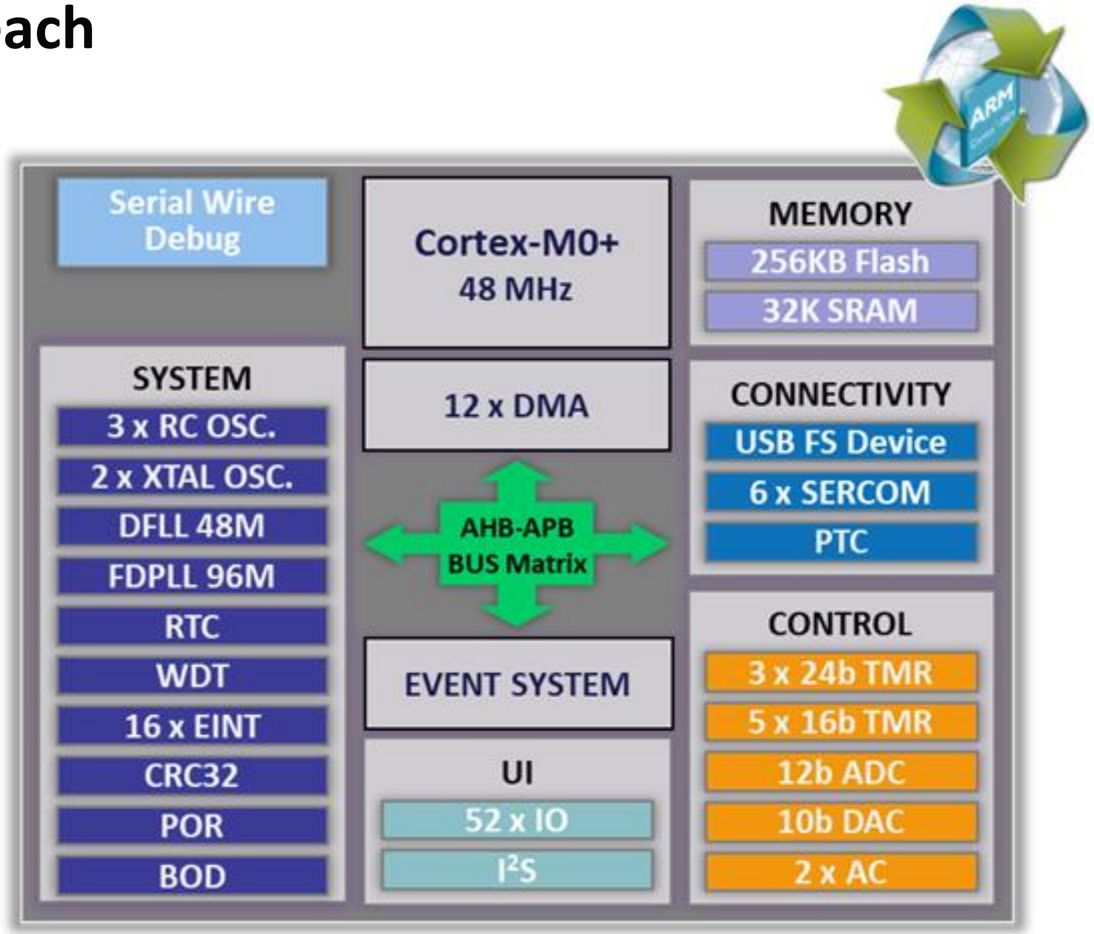
Microchip Cortex-M MCU Series (cont.)

- **Cortex-M7**
  - SAMS7x
    - High performance, rich peripheral.
  - SAME7x
    - Ethernet support, CAN-FD support.
  - SAMV7x
    - Ethernet support, CAN-FD support, automotive certificated.



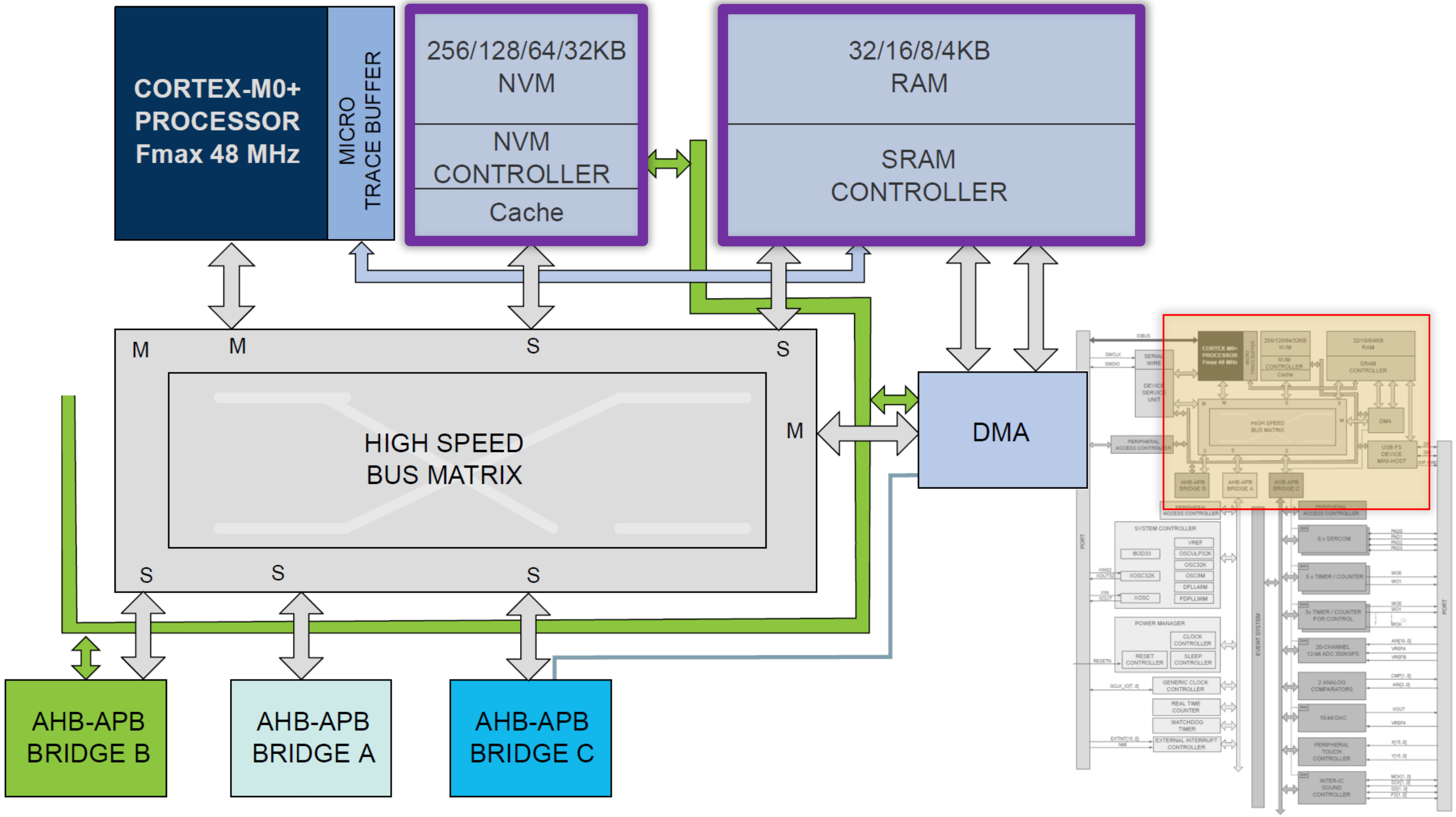
SAMD21 Block Diagram

- 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**

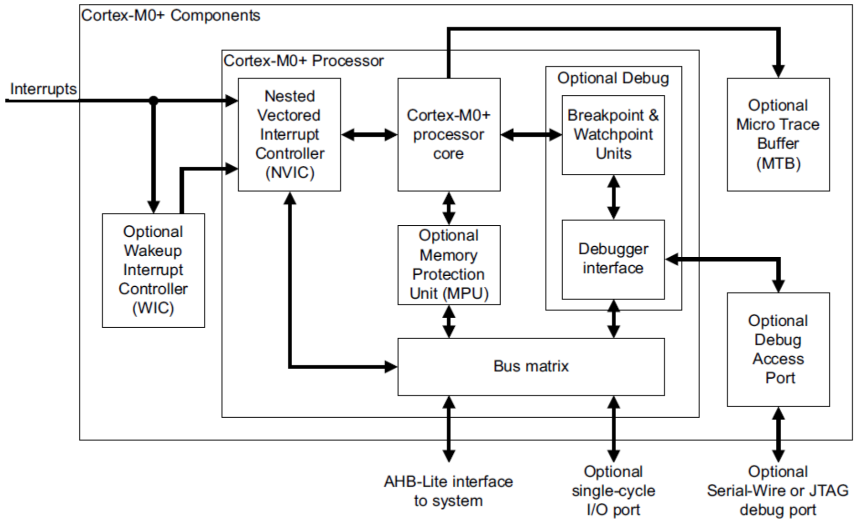


Figure 1-1 Cortex-M0+ implementation

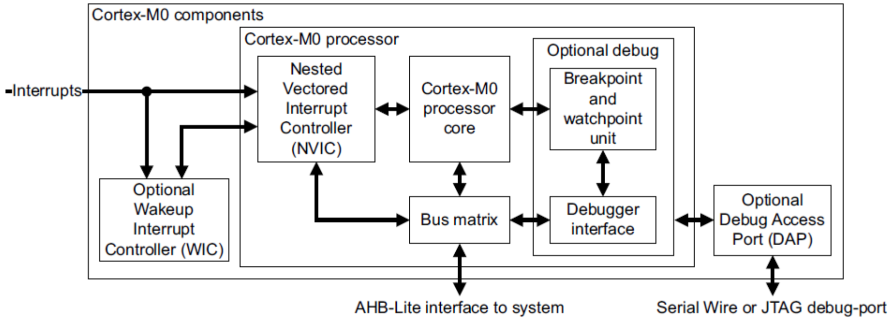


Figure 1-1 Cortex-M0 implementation

Architecture

- **Cortex M3 Harvard**

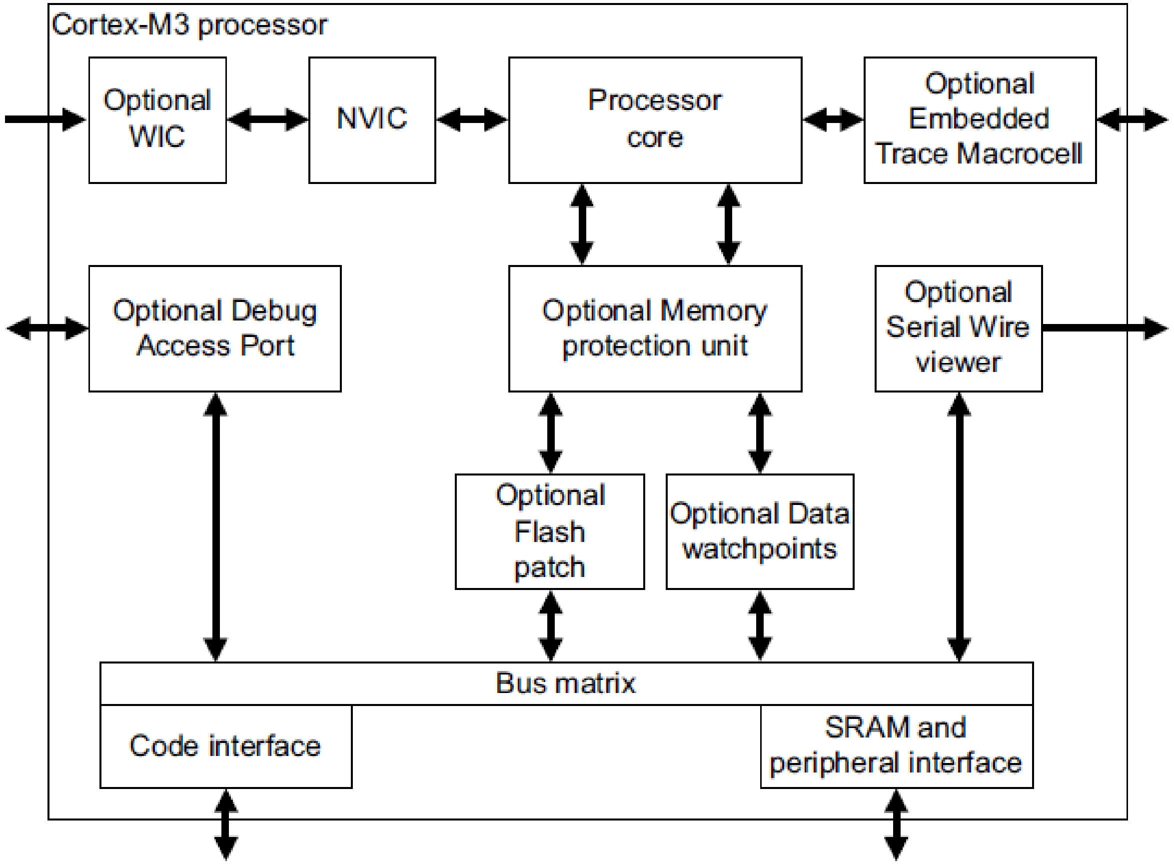


Figure 1-1 Cortex-M3 implementation

Architecture

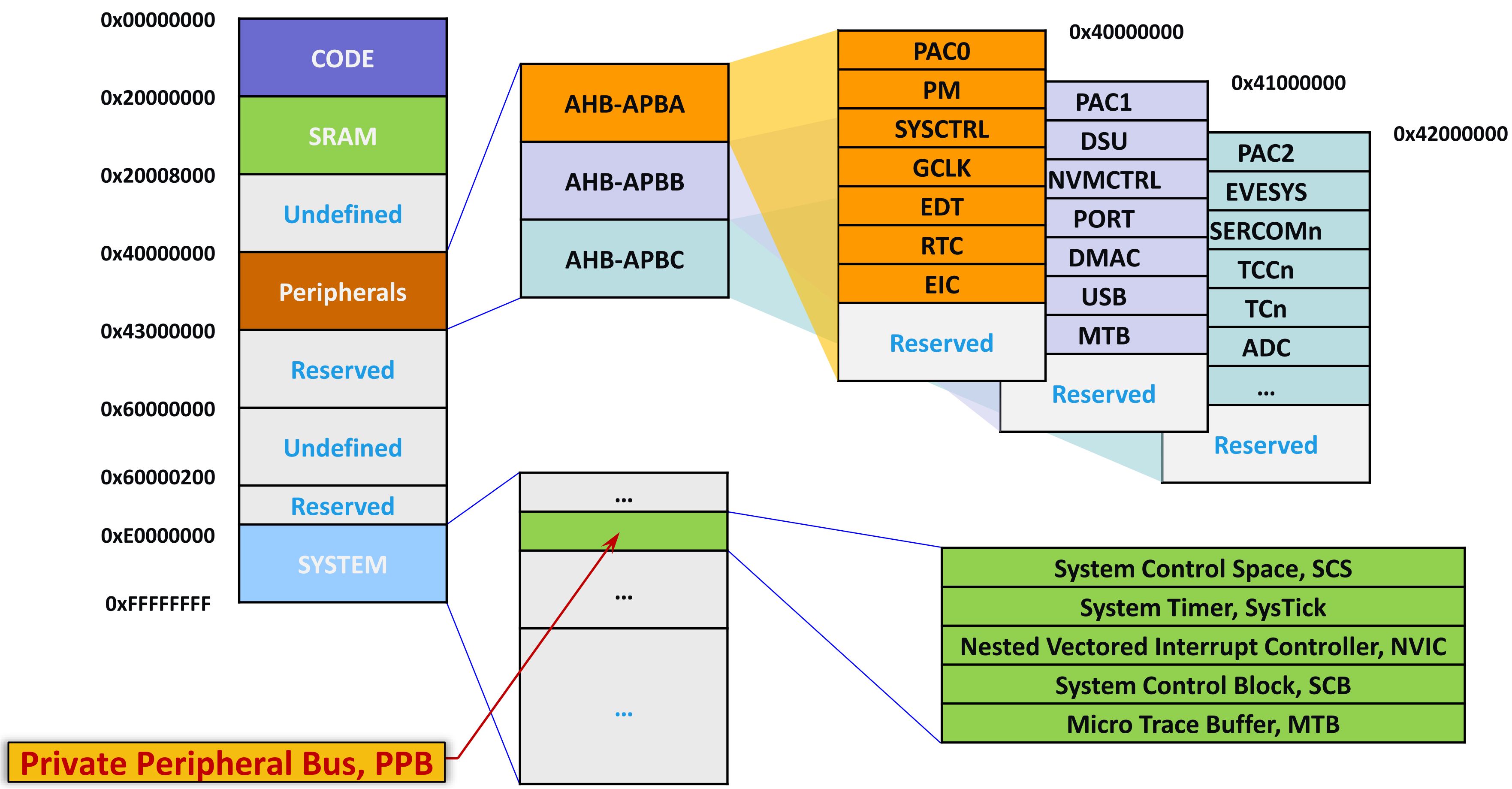
ARM Cortex-M instruction variations

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]	Armv8.1-M
Computer architecture	Von Neuman	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	15 no security ext 27 security ext	TBD	TBD	TBD

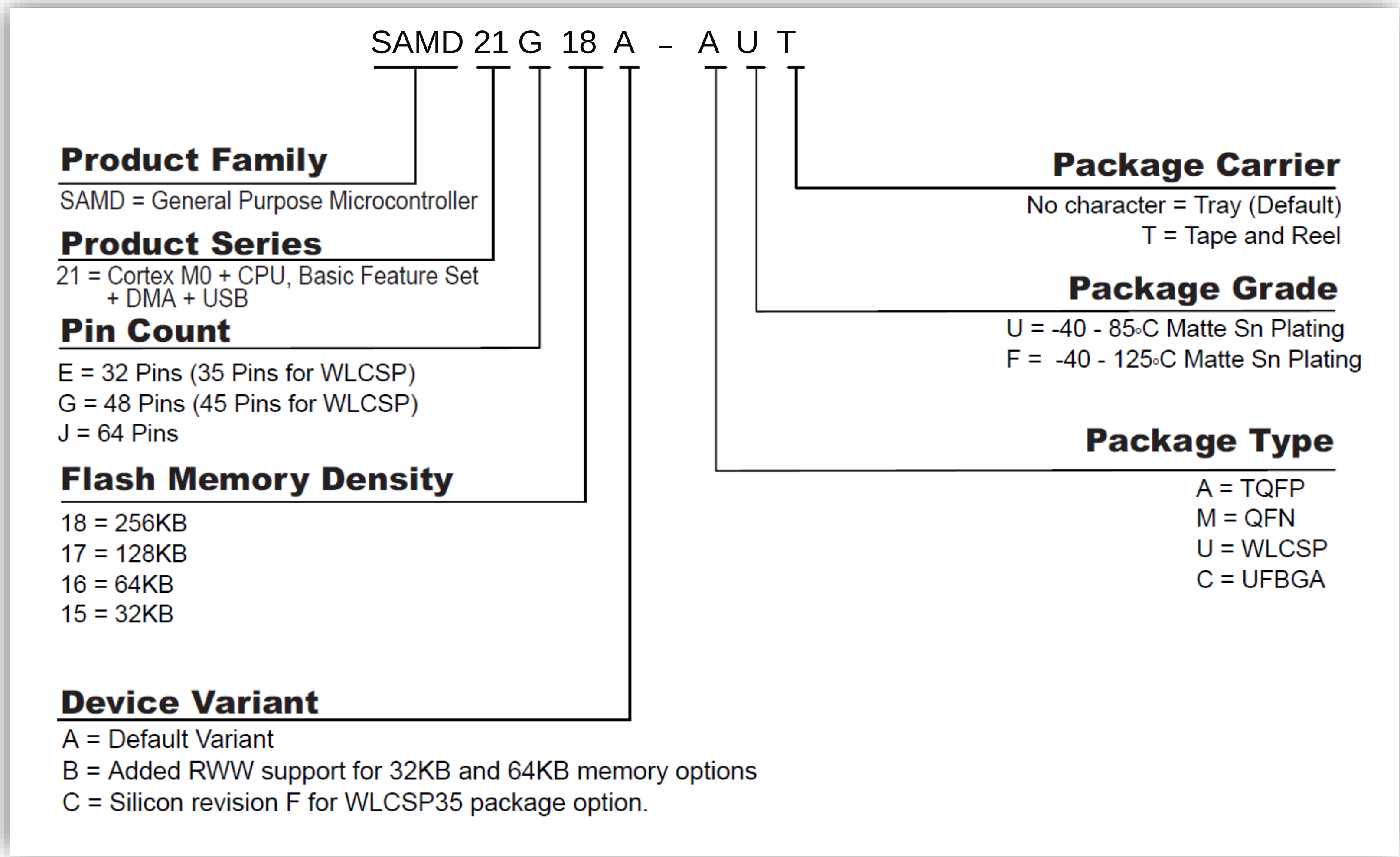
https://en.wikipedia.org/wiki/ARM_Cortex-M

SAMD21 Memory

- Up to 256KB Flash and 32KB SRAM



Part Number

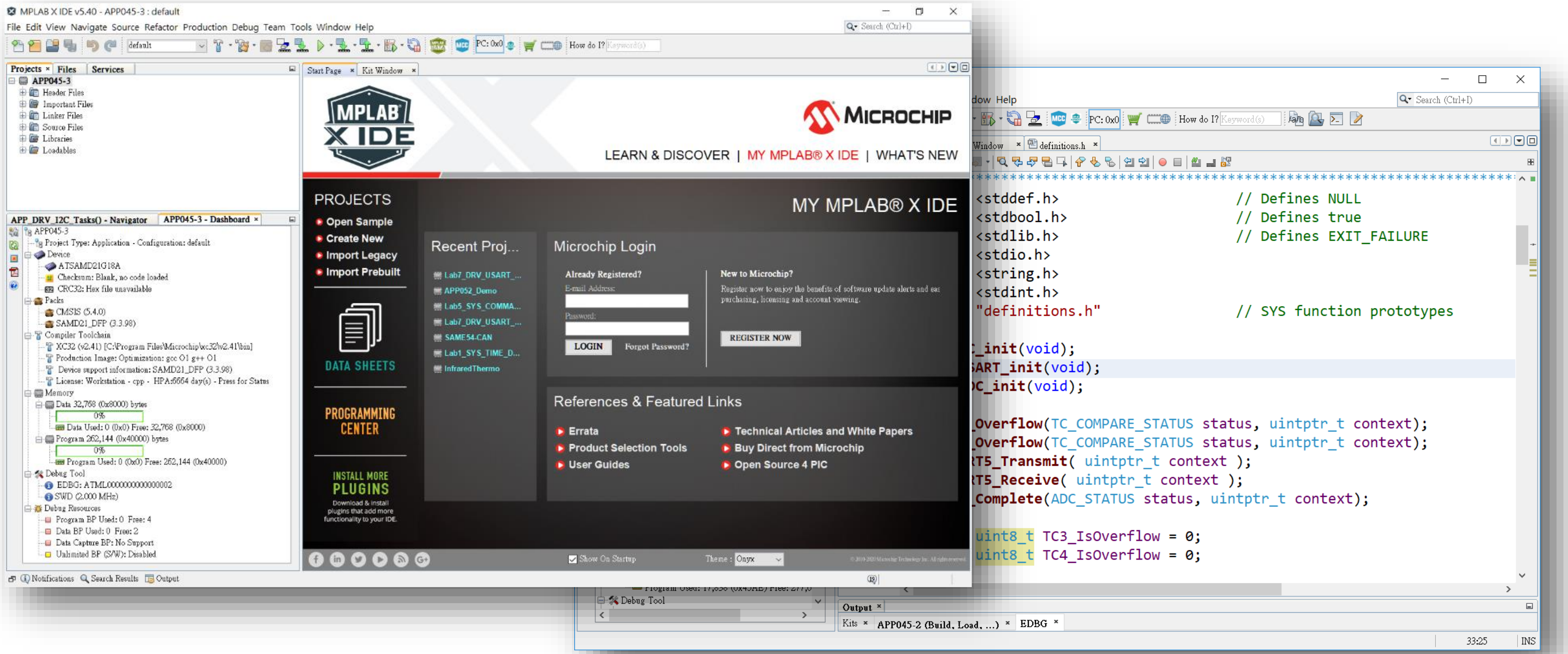

SAMD 21 G 18 A – A U T
Product Family
SAMD = General Purpose Microcontroller
Product Series
21 = Cortex M0 + CPU, Basic Feature Set
+ DMA + USB
Pin Count
E = 32 Pins (35 Pins for WLCSP)
G = 48 Pins (45 Pins for WLCSP)
J = 64 Pins
Flash Memory Density
18 = 256KB
17 = 128KB
16 = 64KB
15 = 32KB
Device Variant
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

MPLAB X IDE and MHC
Development Tools Introduction

All in One Development

MPLAB® X IDE

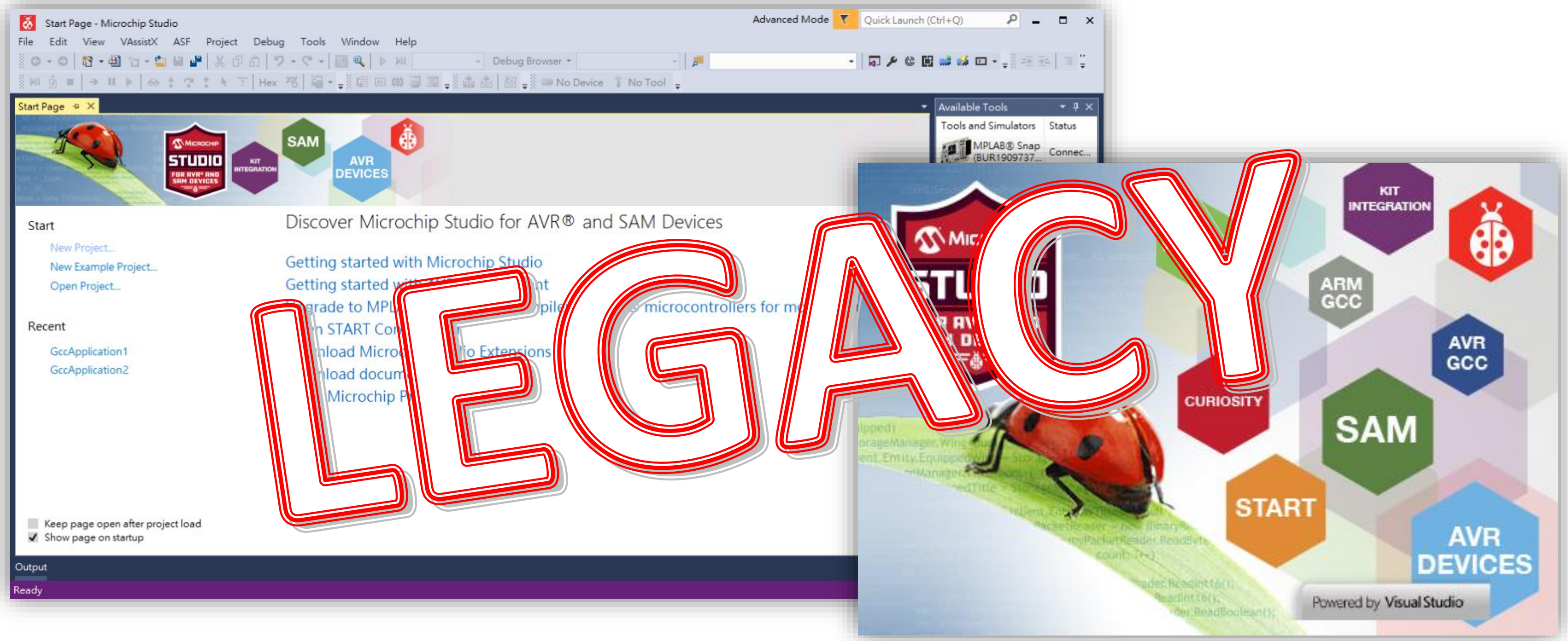
- New generation integrated Development Tools, Support PIC, dsPIC, AVR, CEC and SAM Series MCU/MPU, Provide Plug-in function to extend more advance function. Java Based, Cross platform, Current version is v5.45.



AVR/SAM Series Development

Atmel Studio 7 → Microchip Studio

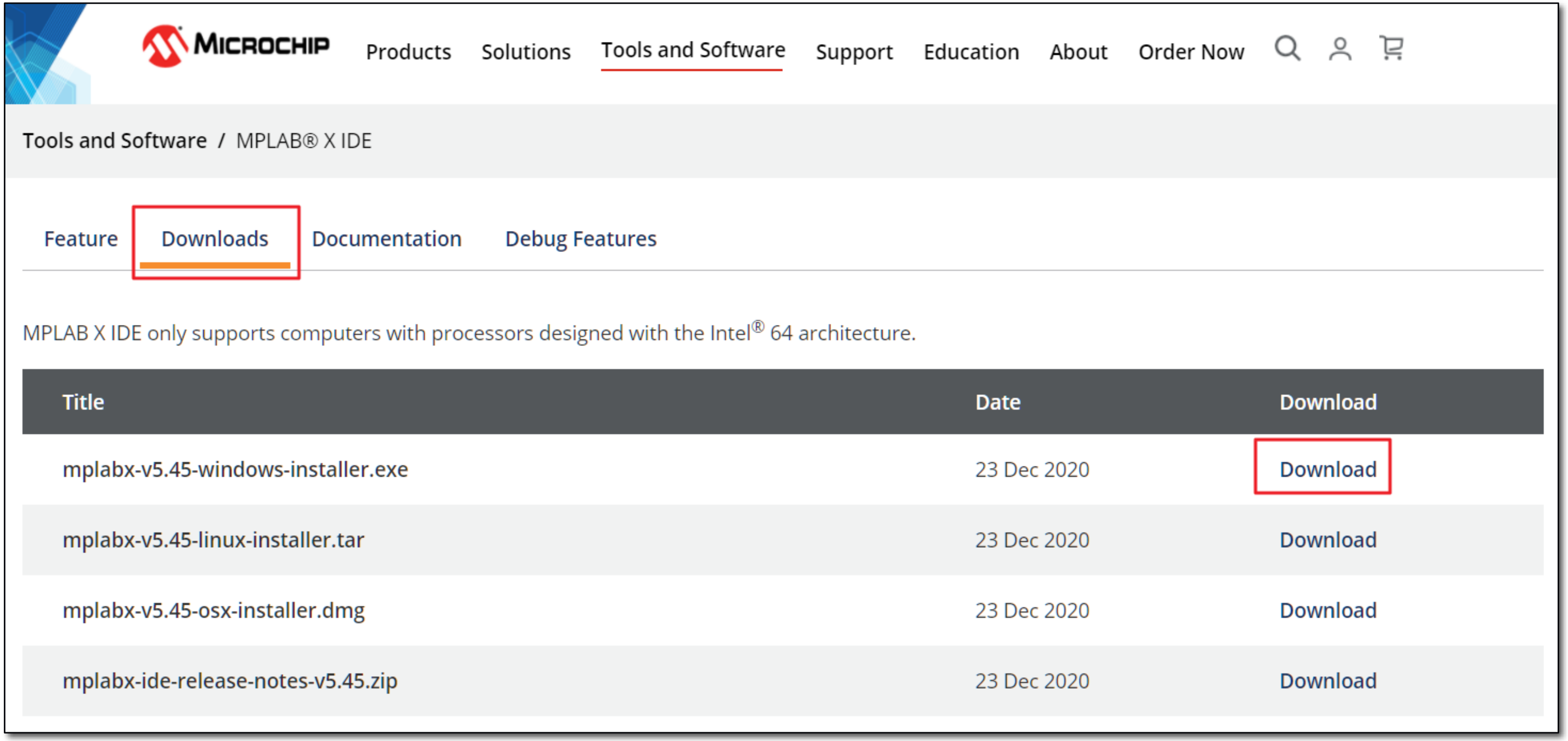
- New generation integrated Development Tools, Support AVR, SAM Series MCU/MPU, Current version is v7.0.2542 。
- The GCC compilers already include support AVR/SAM series.



Microchip

Official Site Download Page

www.microchip.com/mplabX



Tools Installation

- **You can download development tools from below link.**

http://www.microchip.com.tw/RTC/RTC_DVD/

MICROCHIP

Microchip 產品、教育訓練及實驗板資料

Dec. 11, 2020

Microchip Focus Product Selector Guide (July 2019) *Link*
Analog and Interface Product Selector Guide (June 2019) *Link*
Treelink Analog and Interface Products Presentation (June 19 2019) *Link*

技術支援與相關連結

Microchip台灣 (www.microchip.com.tw)
Microchip美國 (www.microchip.com)
Microchip Product Selector and Advanced Parts Selector (www.microchip.com/maps)
Microchip 線上Sample申請 (www.microchip.com/samples)
Microchip線上元件及開發工具購買(**MicrochipDIRECT**) (www.microchipdirect.com)
Microchip線上Code Examples (www.microchip.com/CodeExamplesByFunc.aspx)
Microchip On Line Application Design Center
Microchip On Line Application Notes

開發軟體, 編譯器
Archive Download

MPLAB® X IDE	v5.45
MPLAB® IDE	v8.92

Programmer/Debugger

ICD4

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

$259.95

Programmer/Debugger

PICkit 4

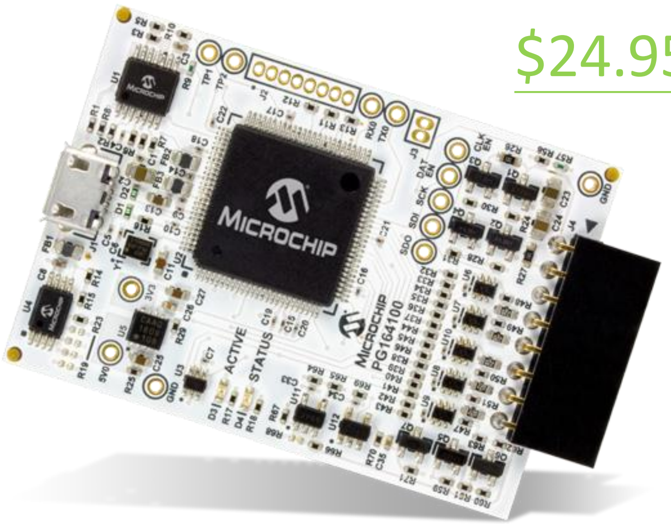
- **The MPLAB® PICkit™ 4 In-Circuit Debugger/Programmer allows fast and easy debugging and programming of all PIC®, dsPIC®, AVR, SAM and CEC flash microcontrollers.**
- **Features**
  - Matches silicon clocking speed.
  - Target voltage of 1.20V to 5.5V.
  - Can supply up to 50mA of power to the target.
  - 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).

$57.95

Programmer/Debugger

MPLAB SNAP

- **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

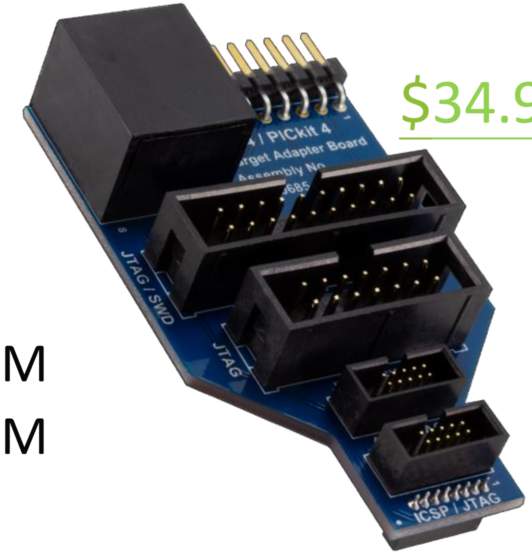


$24.95

Programmer/Debugger

Debug Adapter Board

- **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"



$34.99

Programmer/Debugger

Atmel-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 60 - Low cost PCBA only
  - USD 100 - Basic edition with most common cables
  - USD 140 - Including all cables and adapters

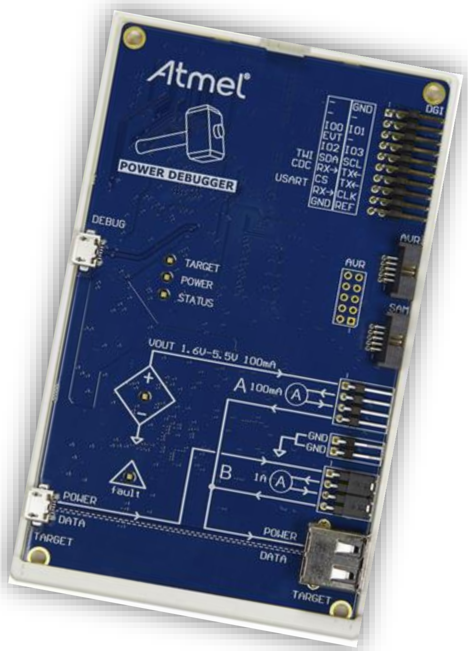
$140

Programmer/Debugger

Power Debugger

- **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 100mAStatus LEDs for debugging & target supply

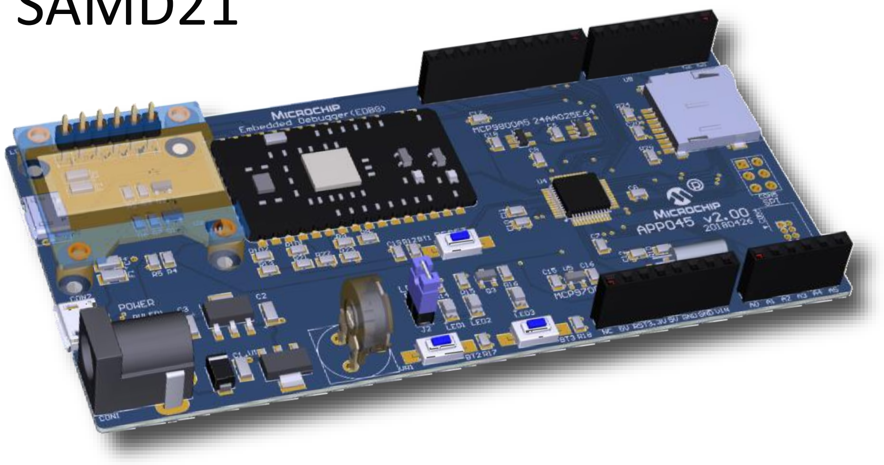
$200



SAMD21 EVB APP045 v2.00 Introduction

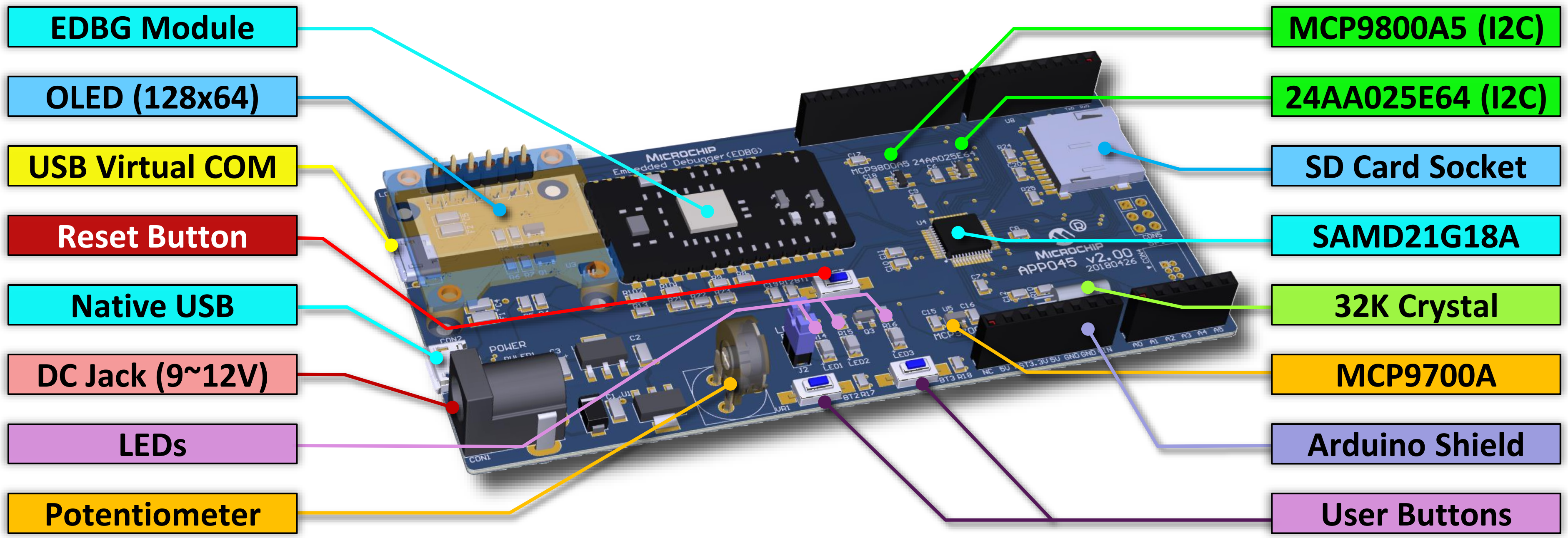
APP045 v2.00 EVB Features

- **APP045 v2.00**
  - **On board SAMD21G18A 32Bit ARM Cortex M0+ Microcontroller**
  - **Built-in Embedded Debugger (EDBG) module**
    Provide Programming & Debugging function for SAMD21
    Provide Virtual COM port (CDC bridge)
  - **Multiple Power Supply Source**
    Programming USB Port
    Native USB Port
    DC Jack (DC 9 ~ 12V)
  - **2 x Buttons, 3 x LEDs**
  - **1 x Potentiometer, 1 x Analog Temperature Sensor**
  - **1 x I2C Temperature Sensor, 1 x I2C EEPROM with Unique ID**
  - **1 x SPI OLED (128 x 64, SSD1306 Compliant)**
  - **1 x MicroSD Socket (use SPI interface)**
  - **Arduino M0 Pro board Compliant with Arduino Shield Connector**

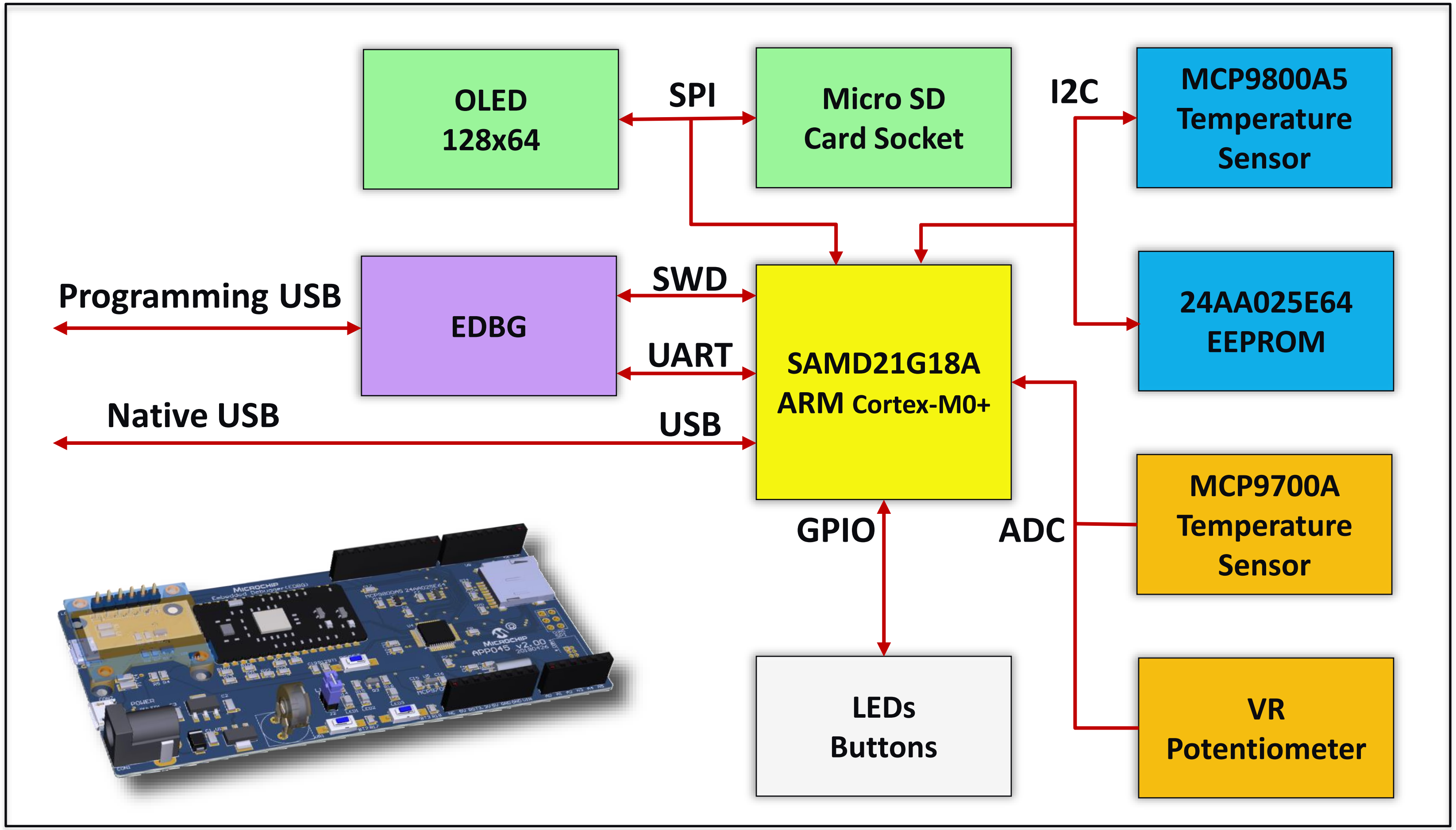


MICROCHIP

APP045 v2.00 Overview

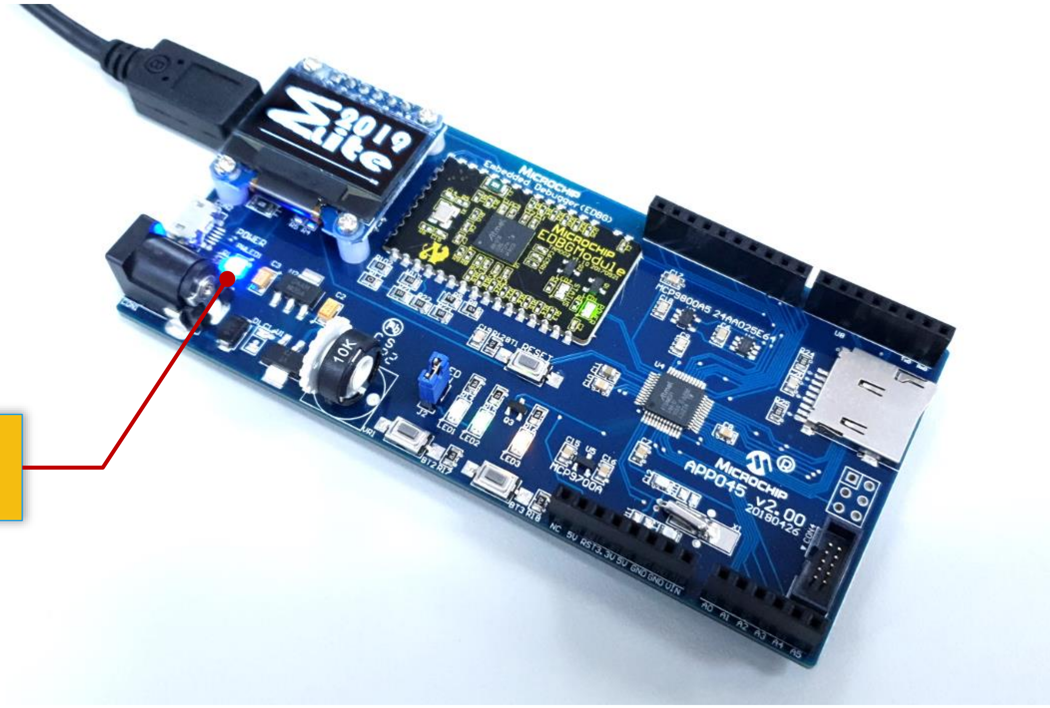


APP045 v2.00 Block Diagram



Power On & Getting started

- **Just plug-in microUSB cable to Programming USB port and confirm the Power LED (PWLED1) has lighting.**
- **OLED & LEDs will show preloaded demo firmware.**



Blue power LED must light.

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.

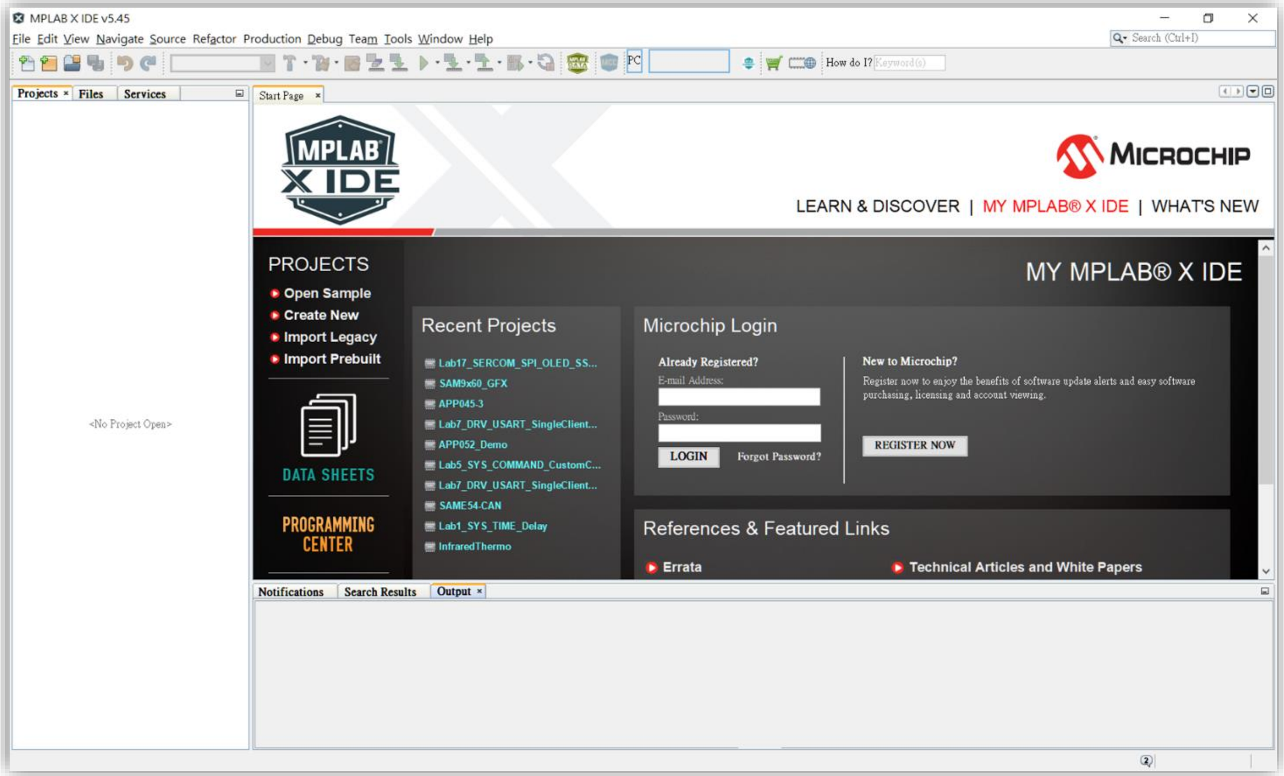
- **How to Start ?**

Lab0 Create Project

Step 1

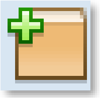
- **Open MPLAB X IDE, firstly.**

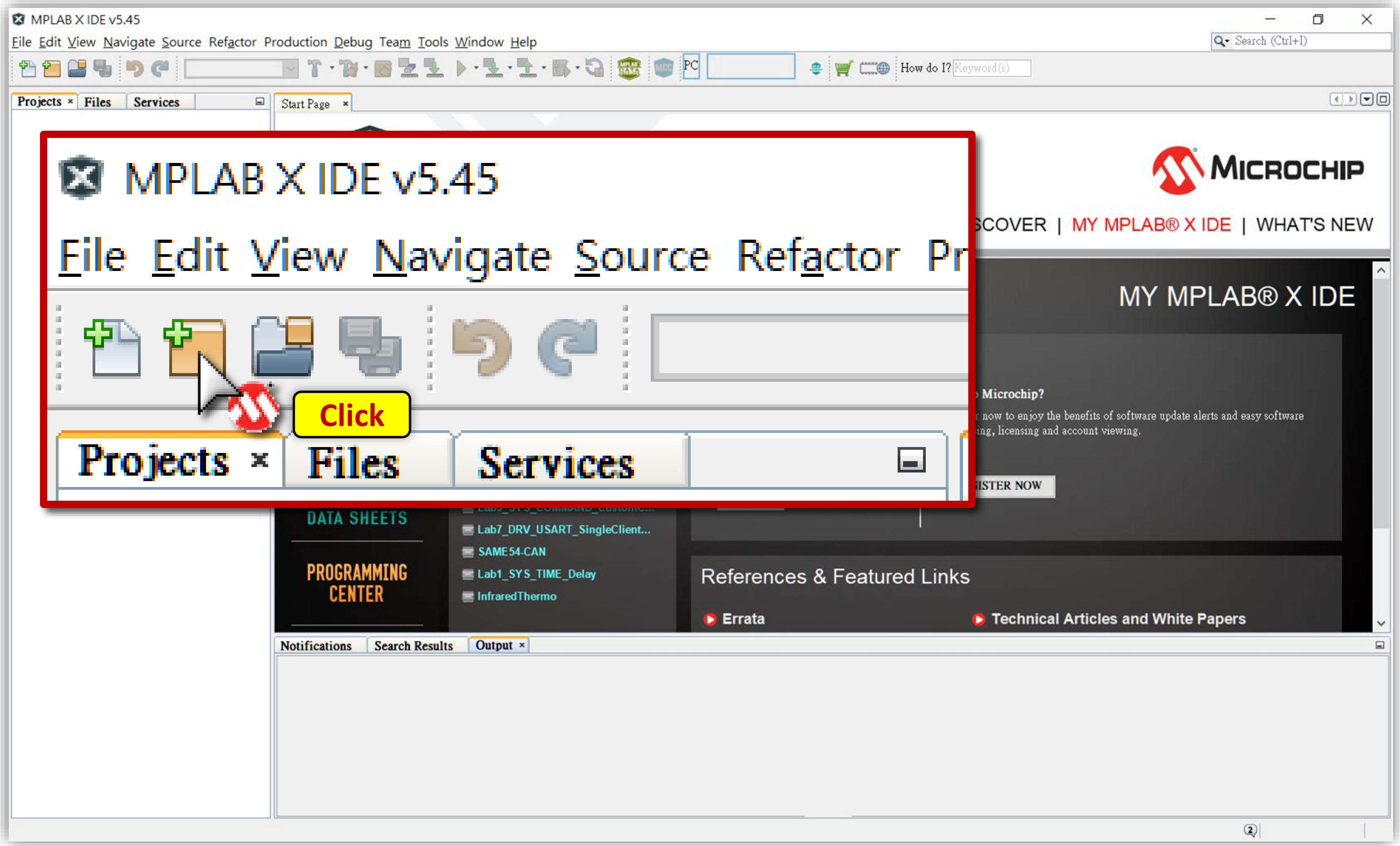
Double Click!



Lab0 Create Project

Step 2

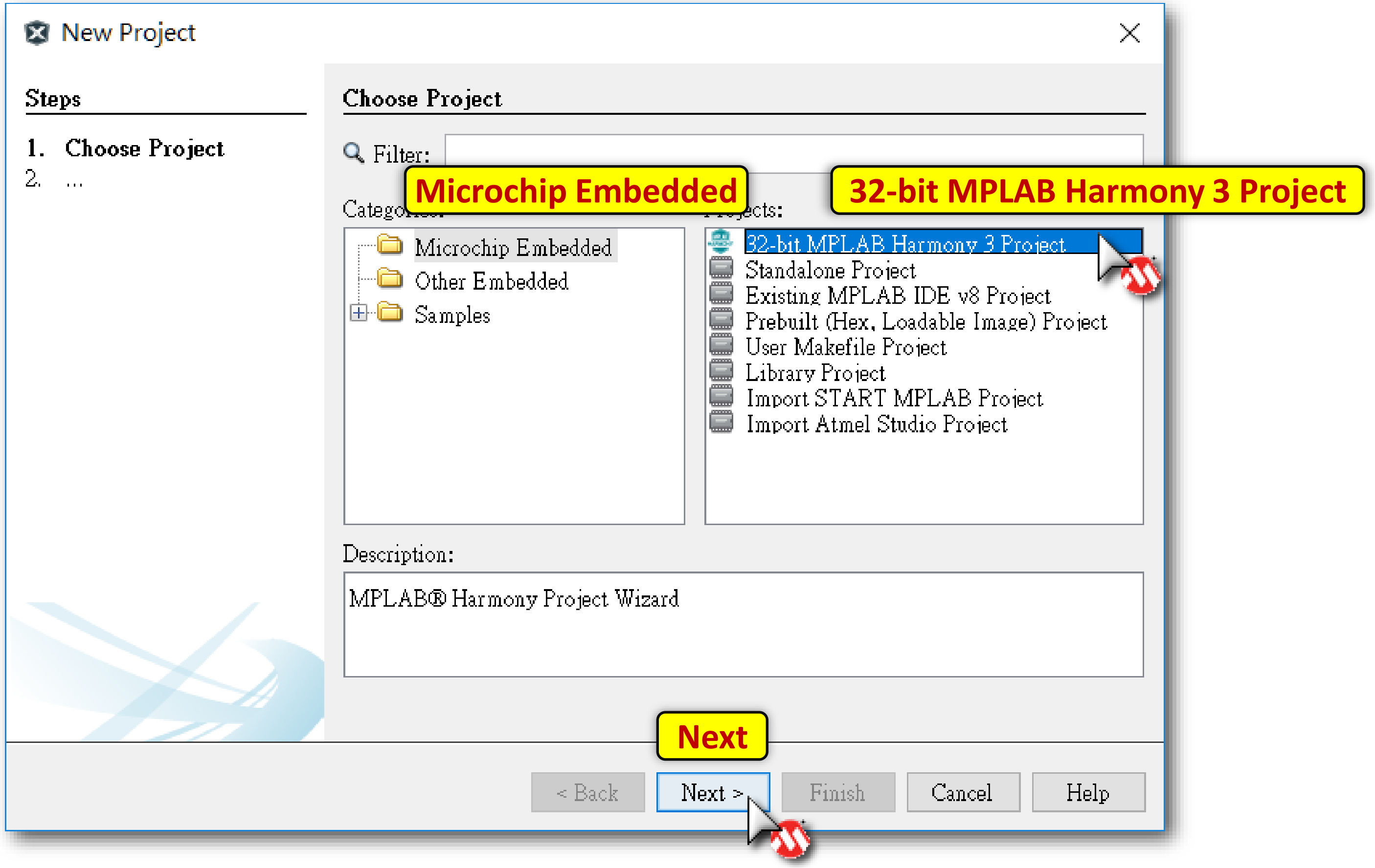
- Select **File ▸ New ▸ Project** or **Click icon** 



Lab0 Create Project

Step 3

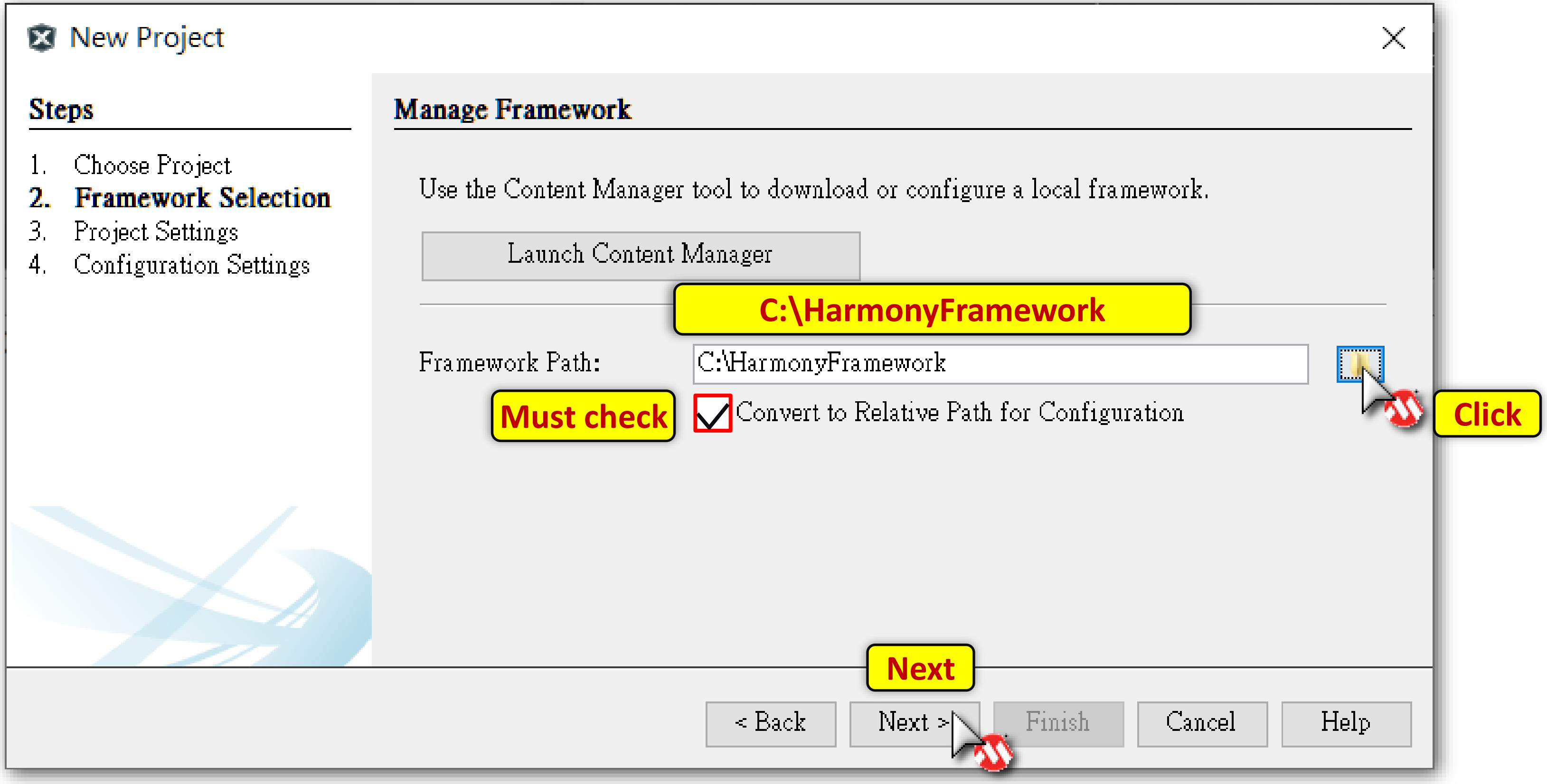
- **Categories :** **Microchip Embedded**
- **Projects :** **32-bit MPLAB Harmony 3 Project**



Lab0 Create Project

Step 4

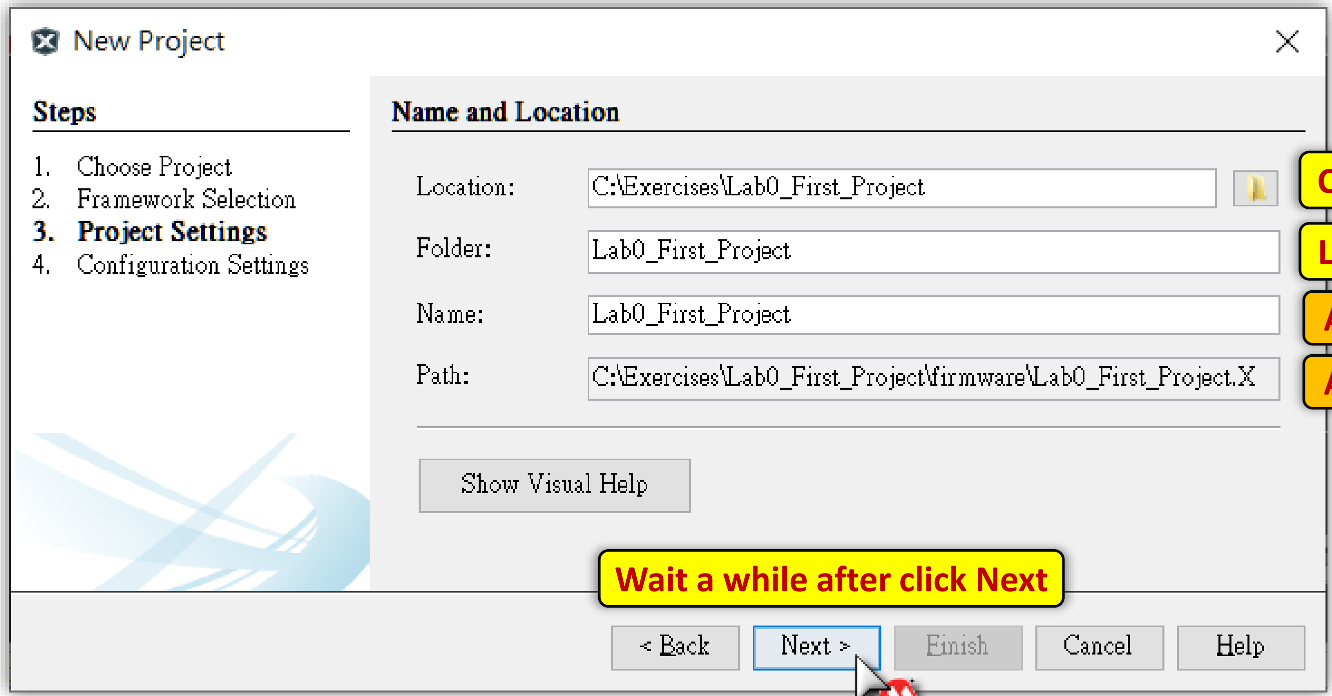
- Assign Harmony 3 framework directory
- Framework Path : C:\HarmonyFramework



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



C:\Exercises\Lab0_First_Project

Lab0_First_Project

Auto generated

Auto generated

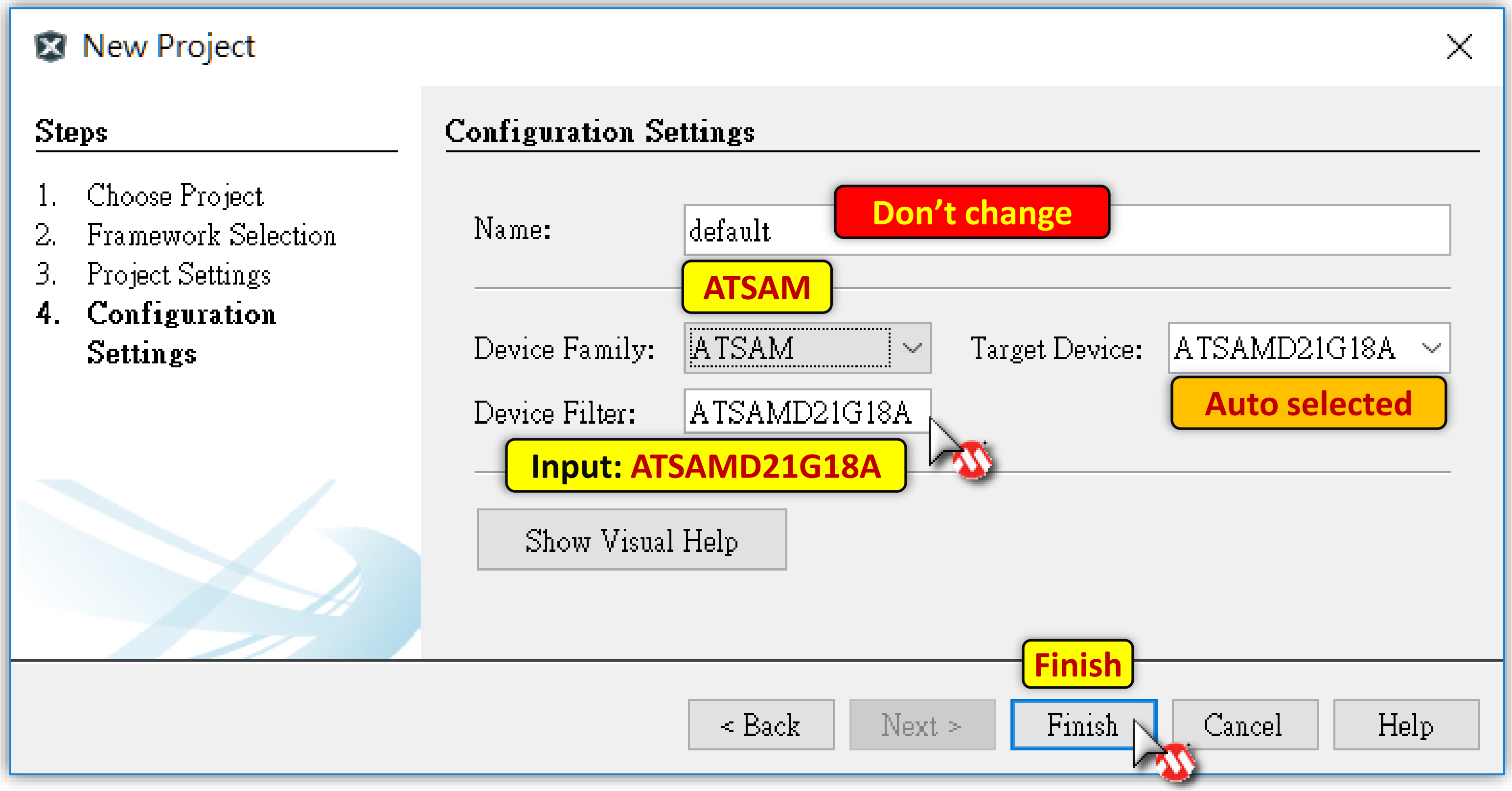
Wait a while after click Next

MICROCHIP

Lab0 Create Project

Step 6

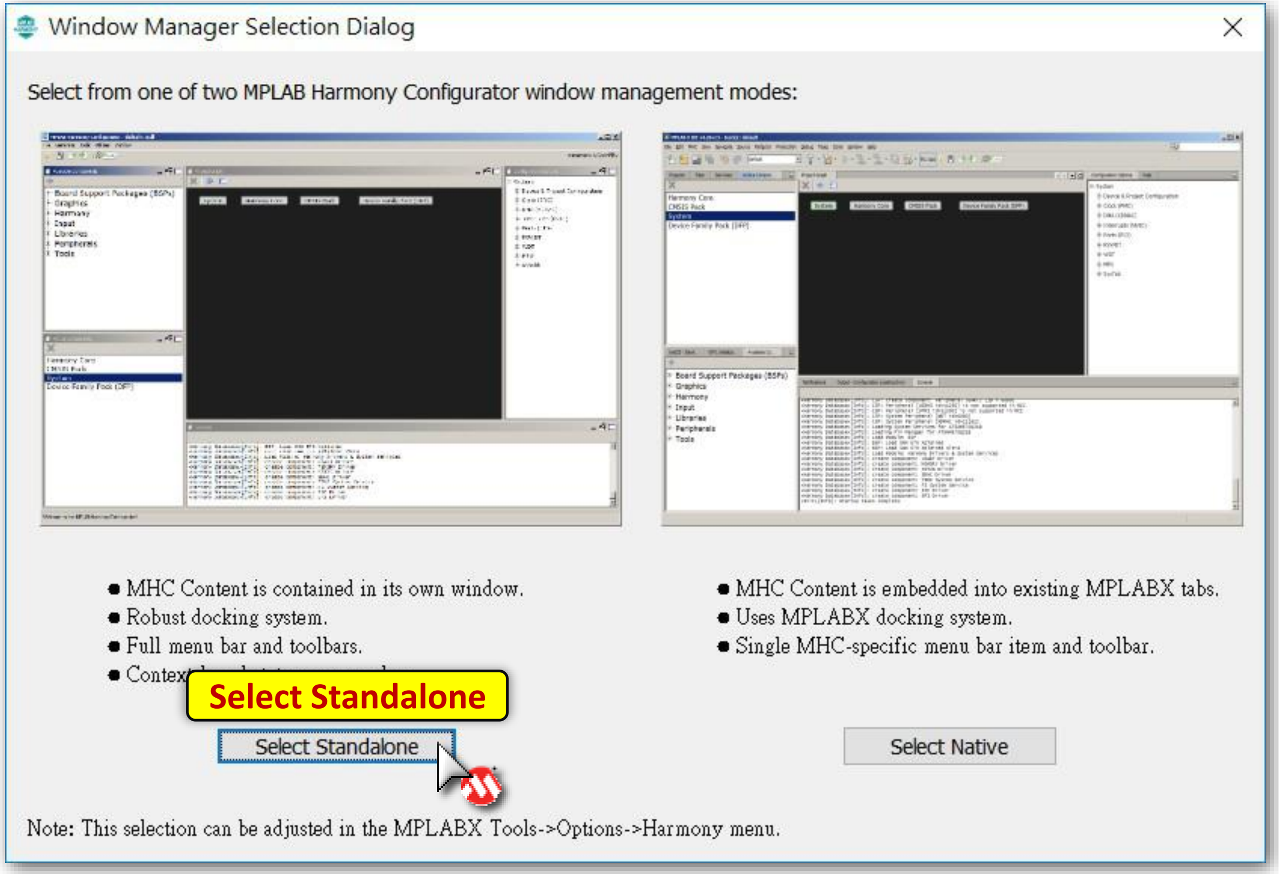
- **Name : default**
- **Device Family : ATSAM Target Device : ATSAMD21G18A**
- **Device Filter : ATSAMD21G18A** **Don't miss any character**



Lab0 Create Project

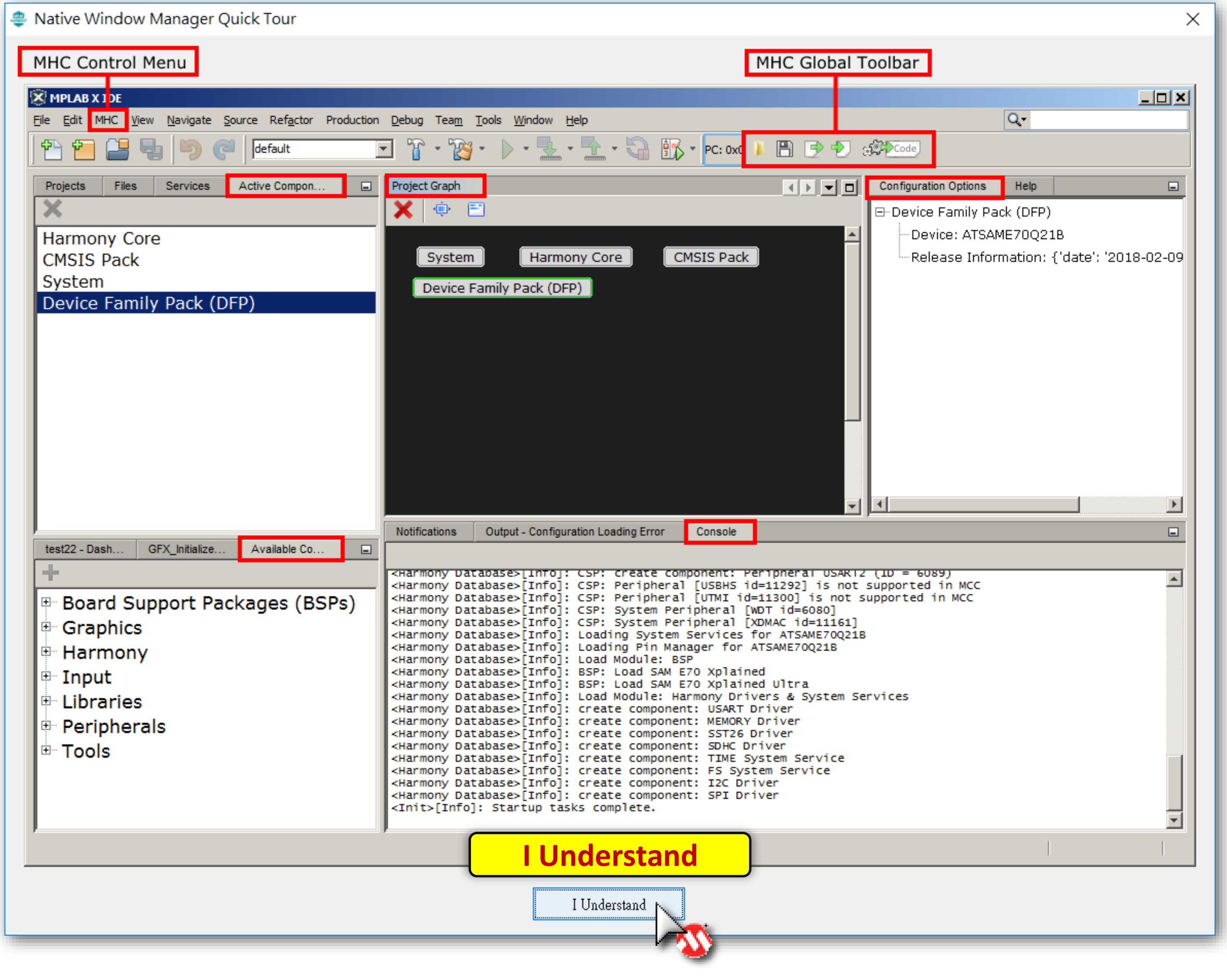
Step 7

- **Select Harmony Configurator window type if first time to execute MHC after install. Please select to Standalone mode**



Lab0 Create Project

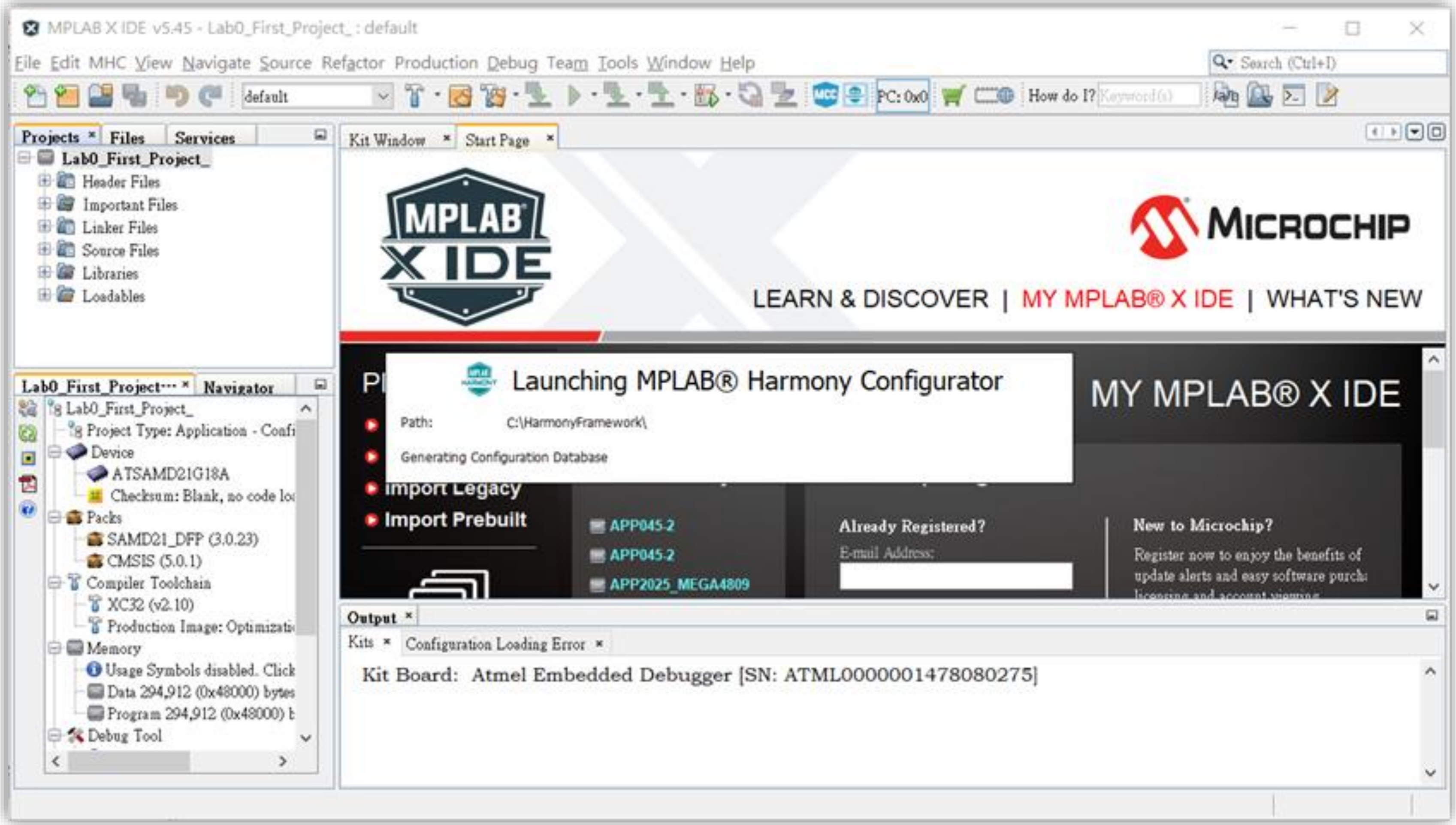
Help for Select Native Mode



Lab0 Create Project

Step 8

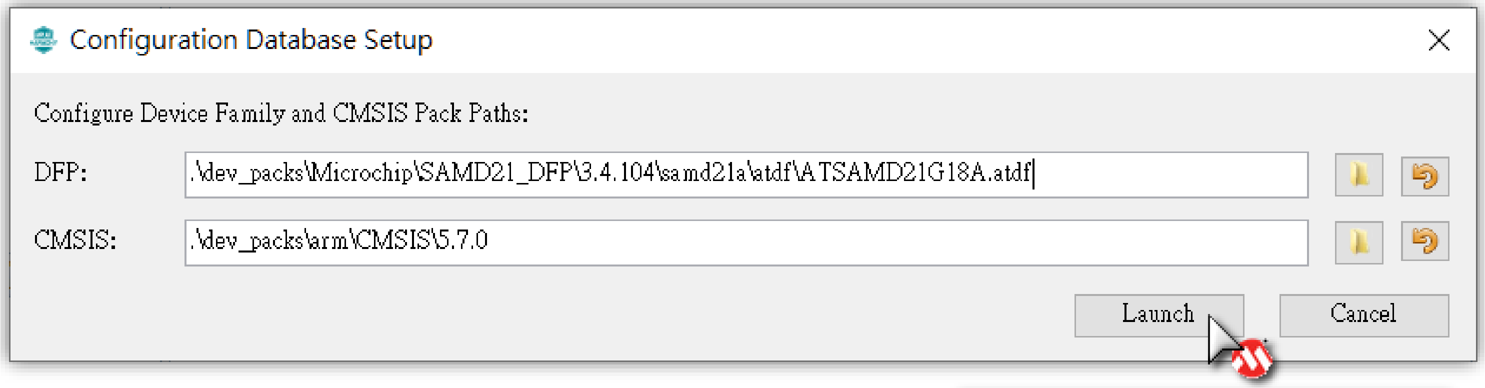
- **Wait a while to launch Harmony Configurator.**



Lab0 Create Project

Step 9

- **Select DFP and CMSIS. (Use default is fine)**

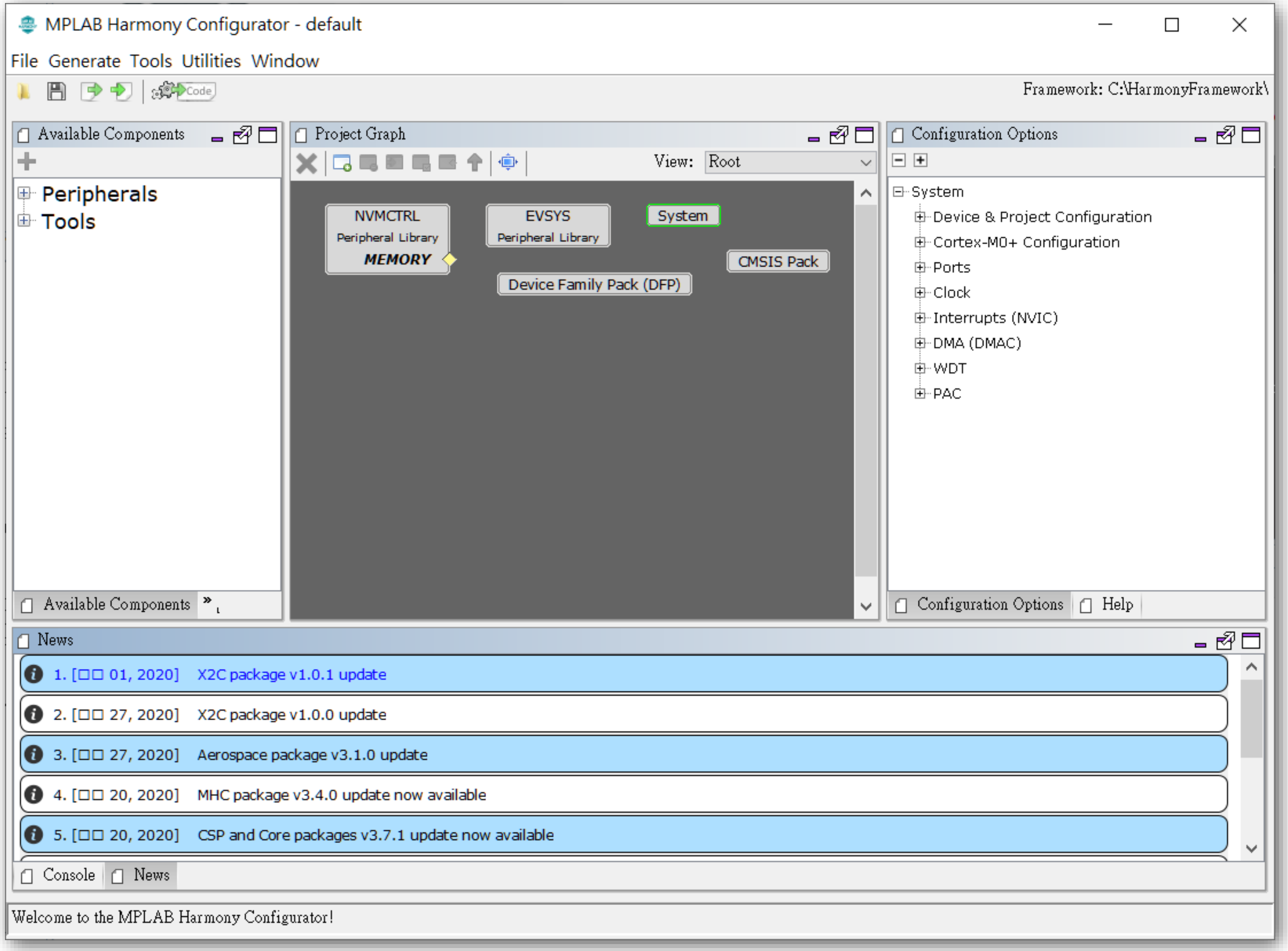


Click launch and wait a while

Lab0 Create Project

Step 10

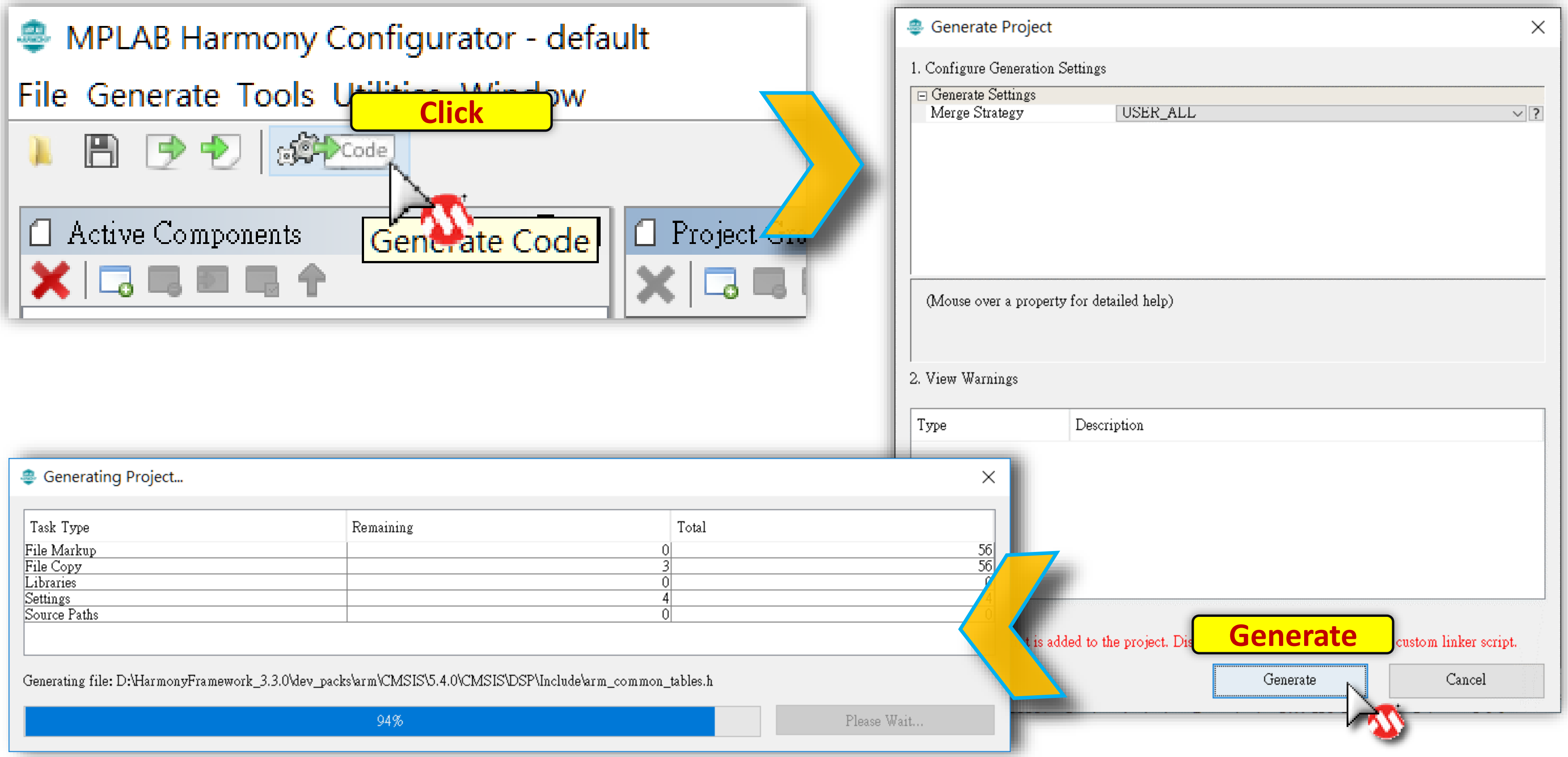
- **Harmony Configurator interface show up, let’s come back later.**



Lab0 Generate Code

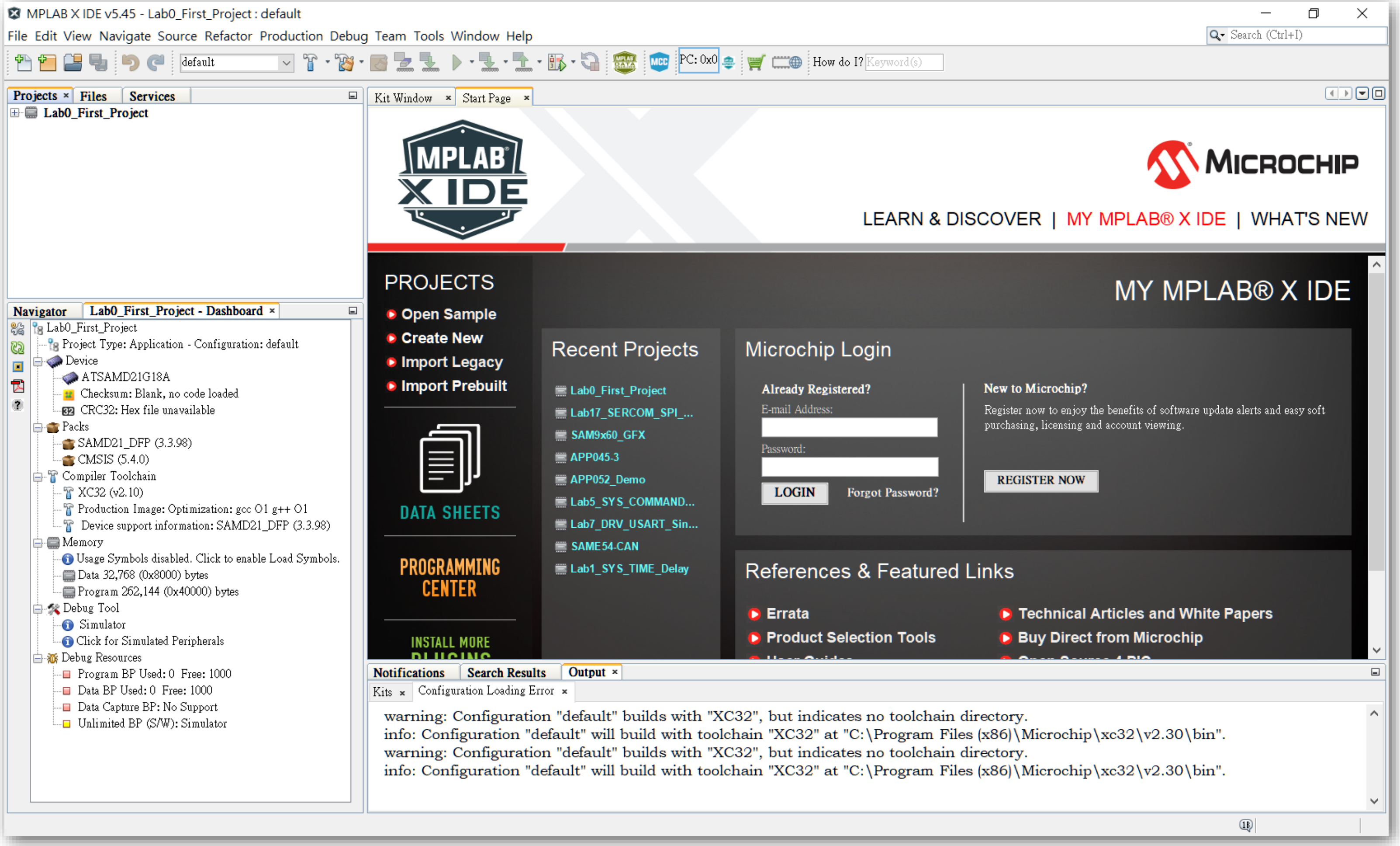
Step 11

- **Click  to Generate Code.**



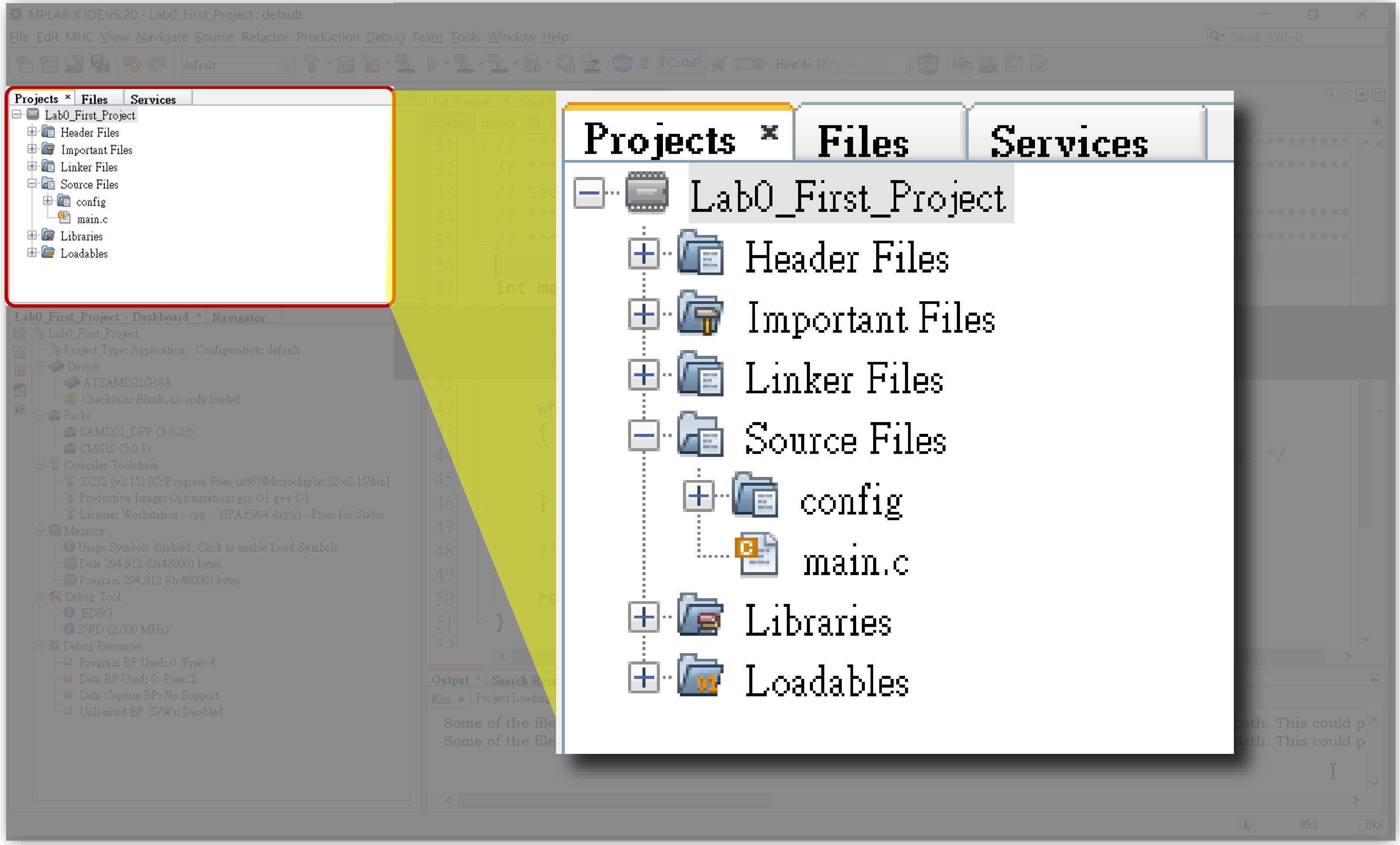
MPLAB X IDE Interface

- Return to MPLAB X IDE, let’s go through its interface.



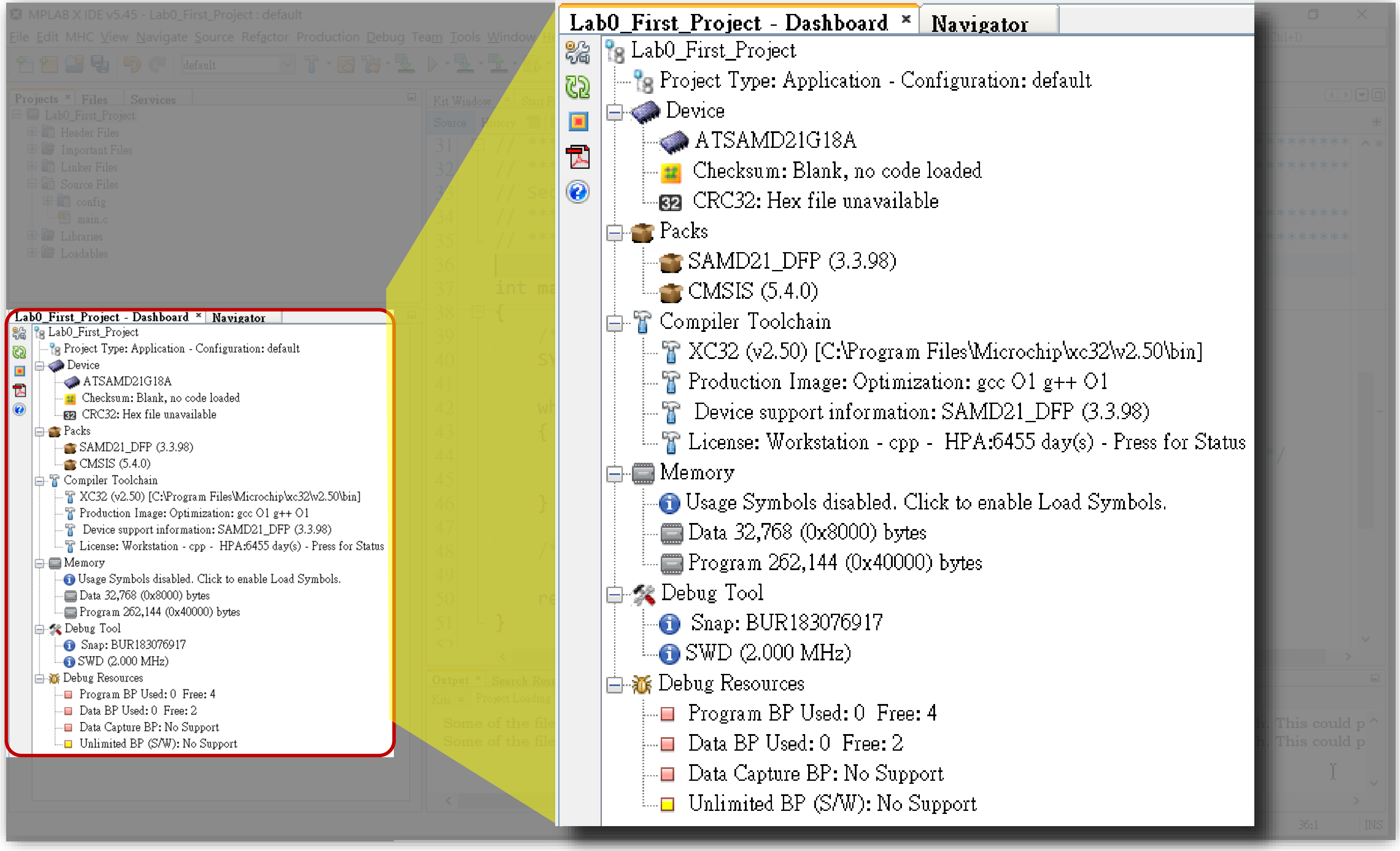
MPLAB X IDE Interface

- **Project window.**



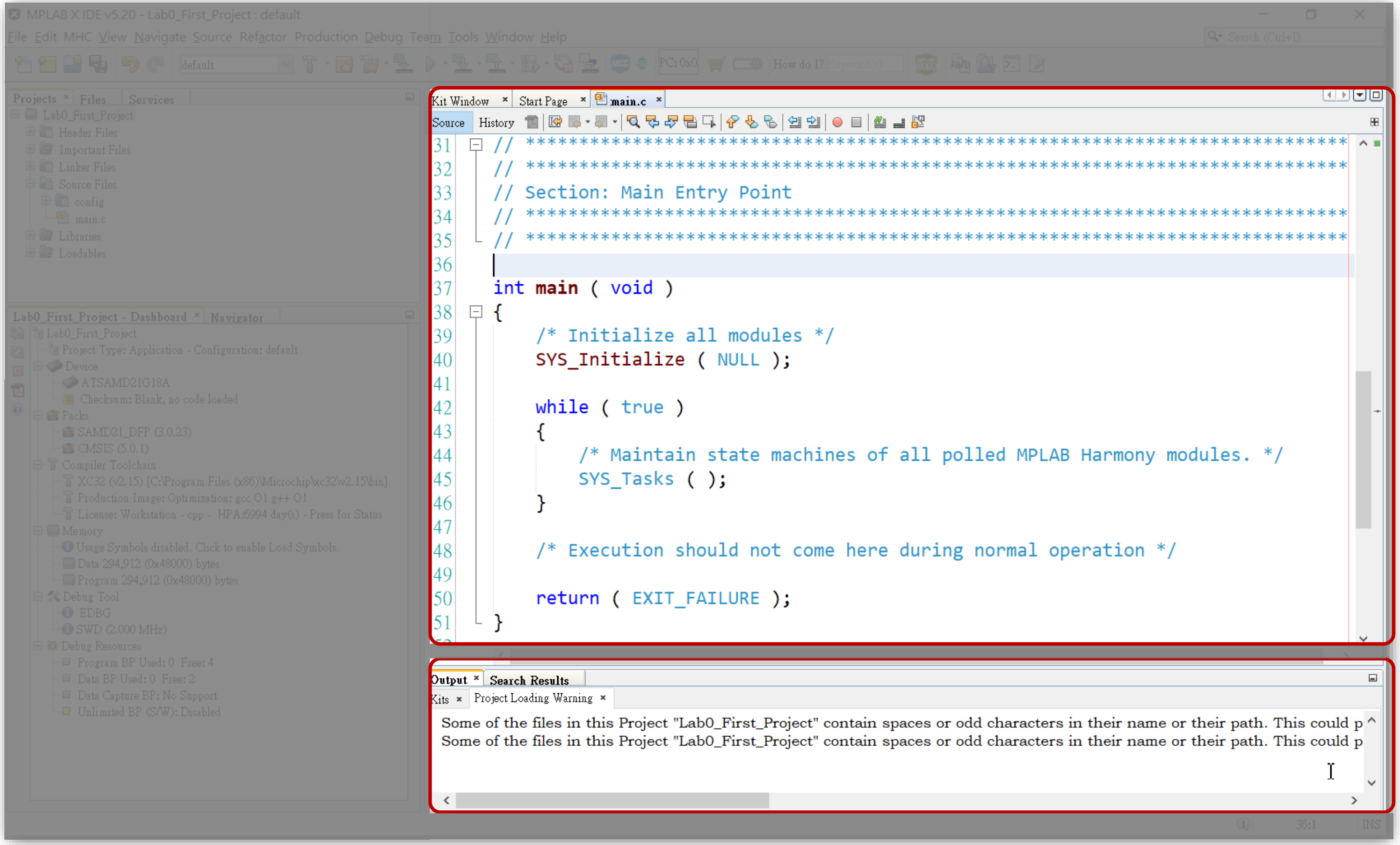
MPLAB X IDE Interface

- **Dashboard window.**



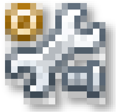
MPLAB X IDE Interface

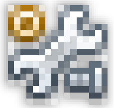
- **Source code and Output window.**



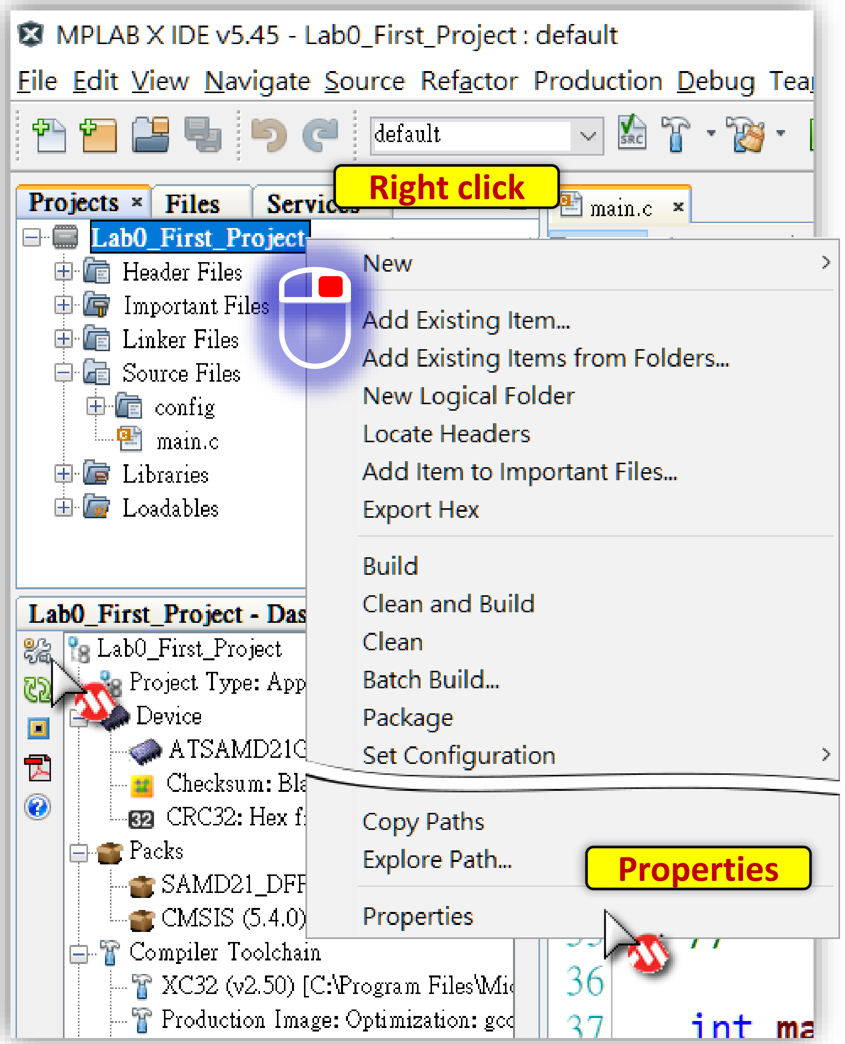
Lab0 Project Properties

Step 12

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



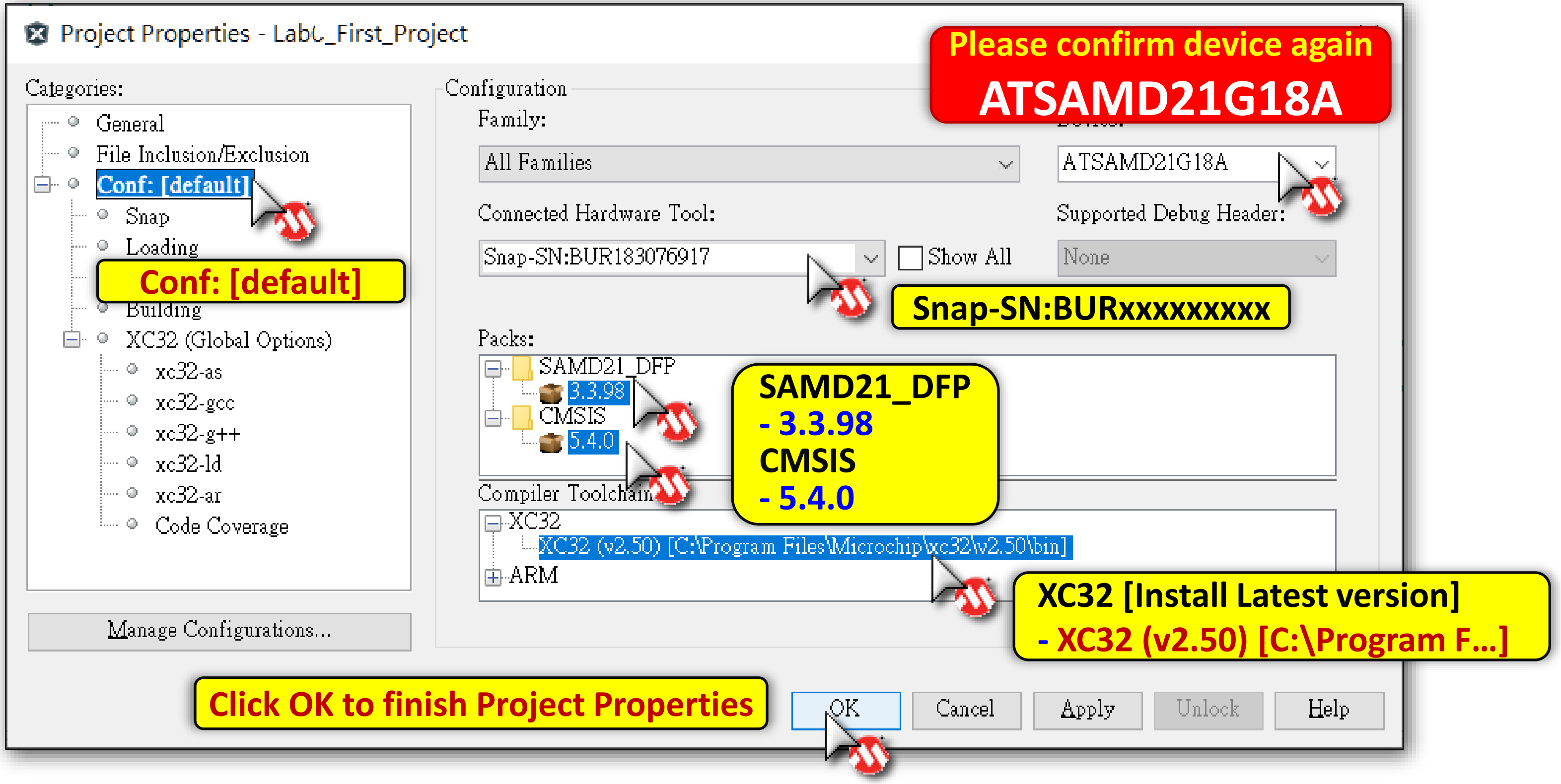
Click



Lab0 Project Properties

Step 13

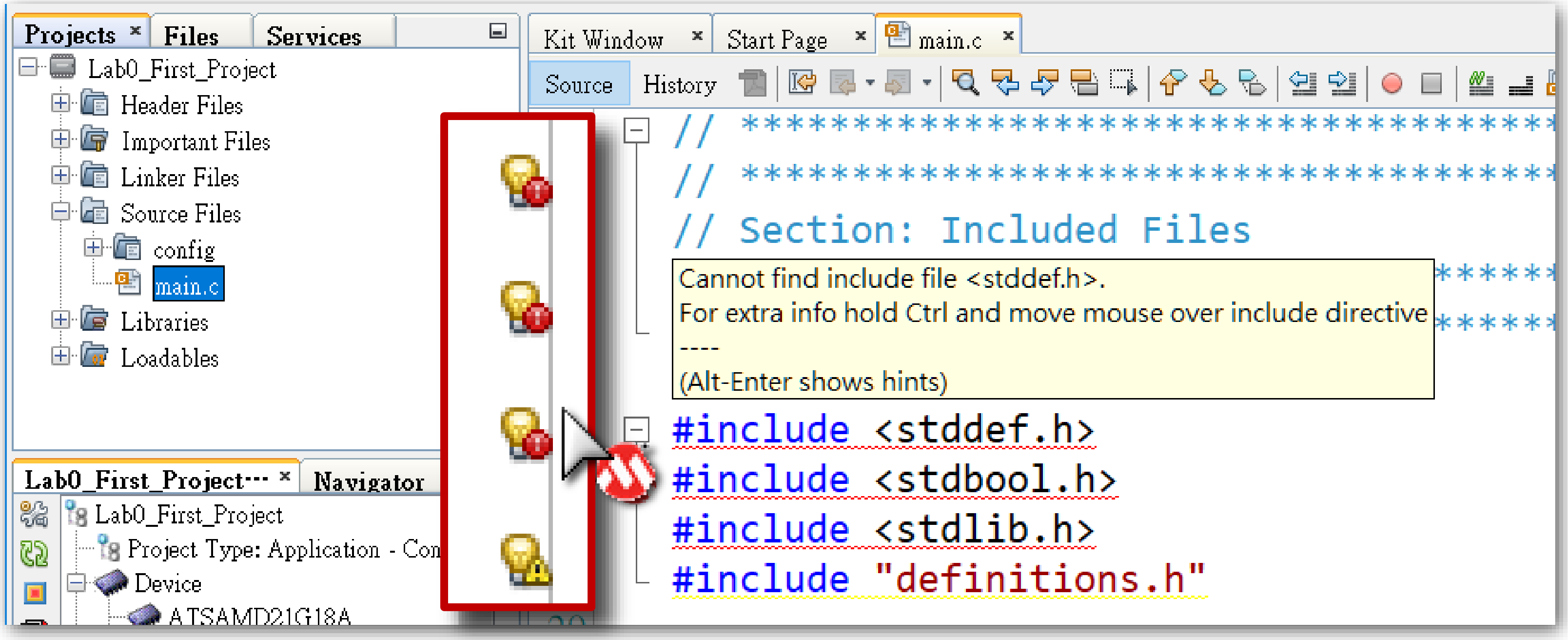
- Select Connected Hardware Tool : Snap-SN: BURxxxxxxxxx
- SAMD21_DFP : 3.3.98
- CMSIS : 5.4.0
- Compiler Toolchain : XC32 (v2.50) [C:\Program Files (x86)…]



MICROCHIP

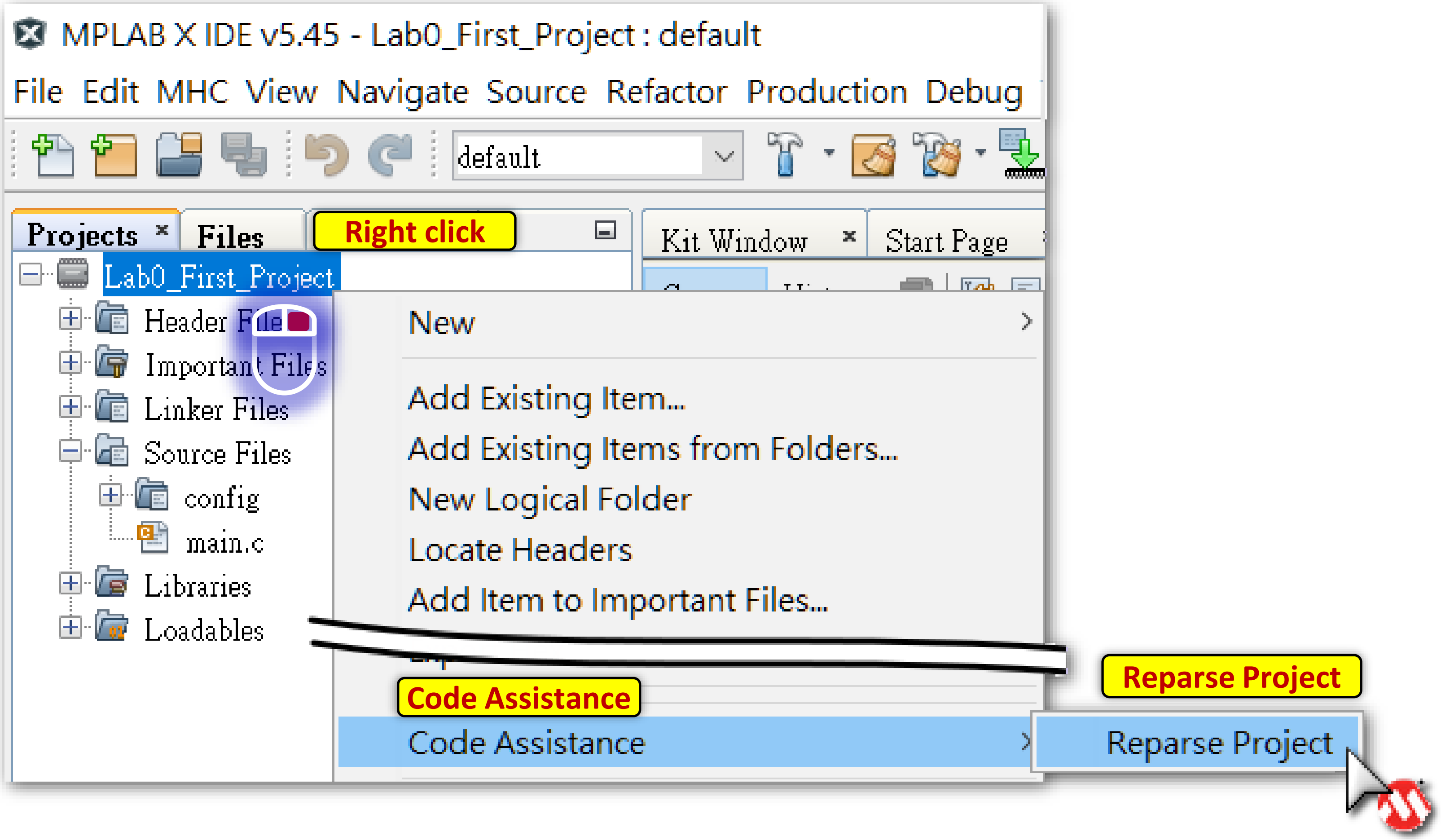
Project Reparse

- MPLAB X IDE build-in project parse function to provide code completion and track function etc..
- In some case, project unable to parse immediately, you must manually reparse project.



Project Reparse

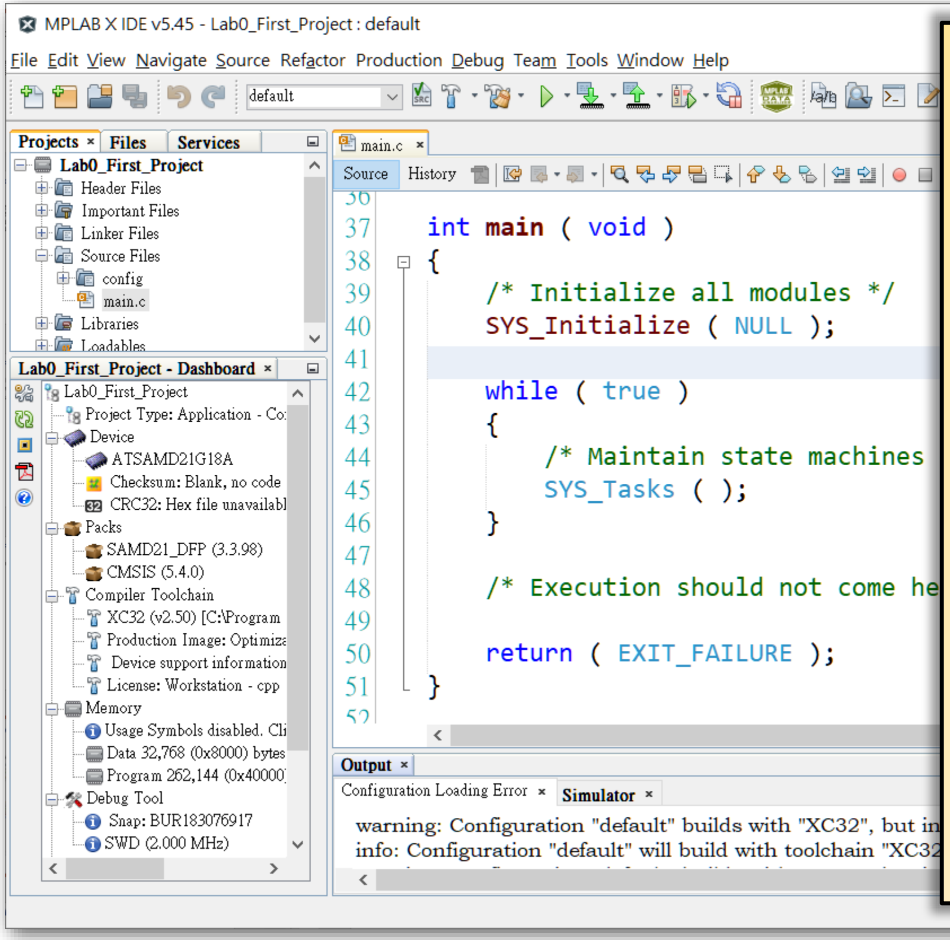
- **Right click on project, select Code Assistance ▸ Reparse Project.**



Lab0 Add Code

Step 14

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

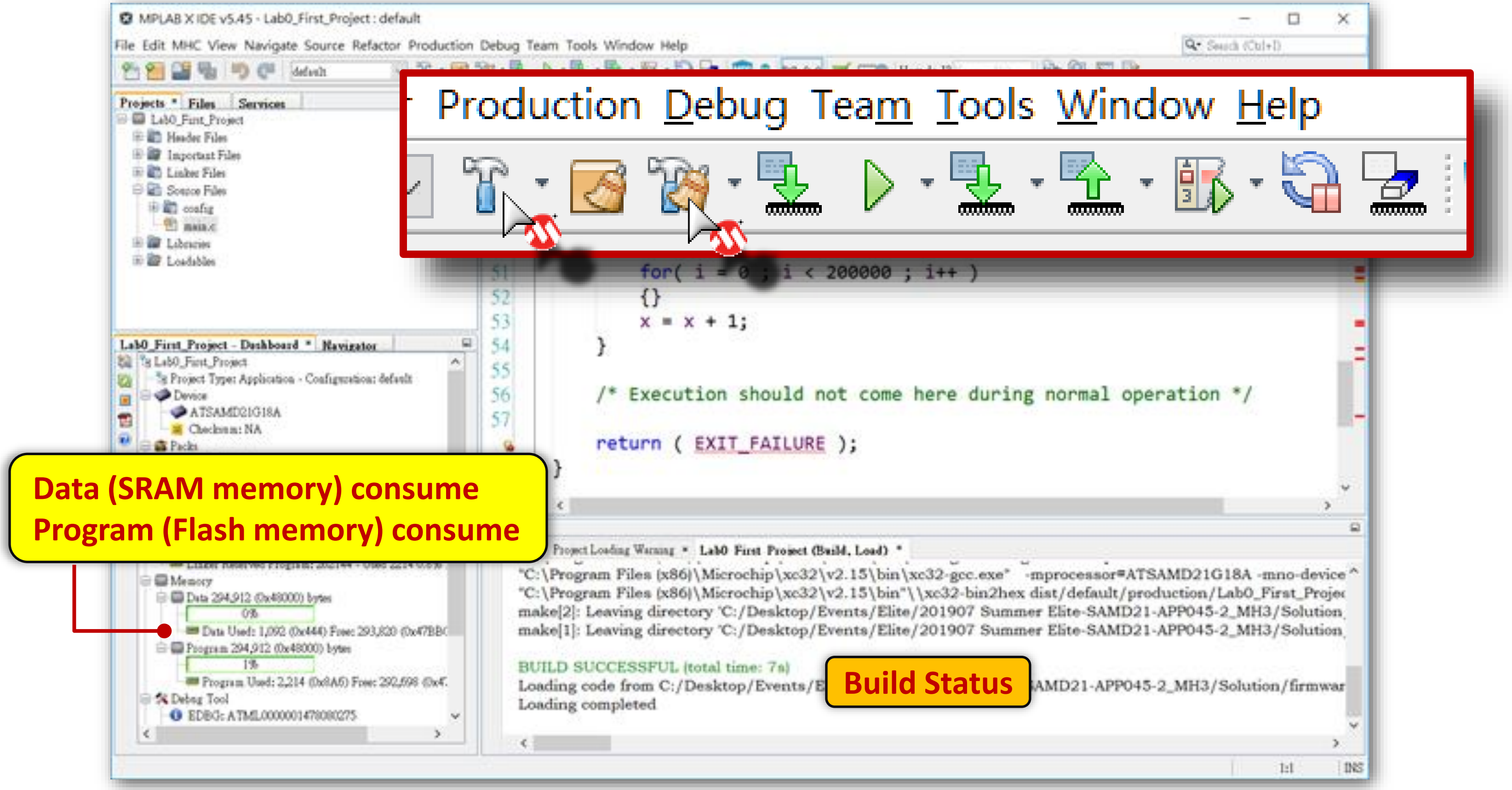



```
// TODO 0.01
uint32_t i = 0;
uint32_t x = 0;

int main( void )
{
   SYS_Iinitialize(NULL);

   while( true )
   {
       SYS_Tasks();

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

   return ( EXIT_FAILURE);
}
```

Lab0 Build Project

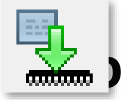
Step 15

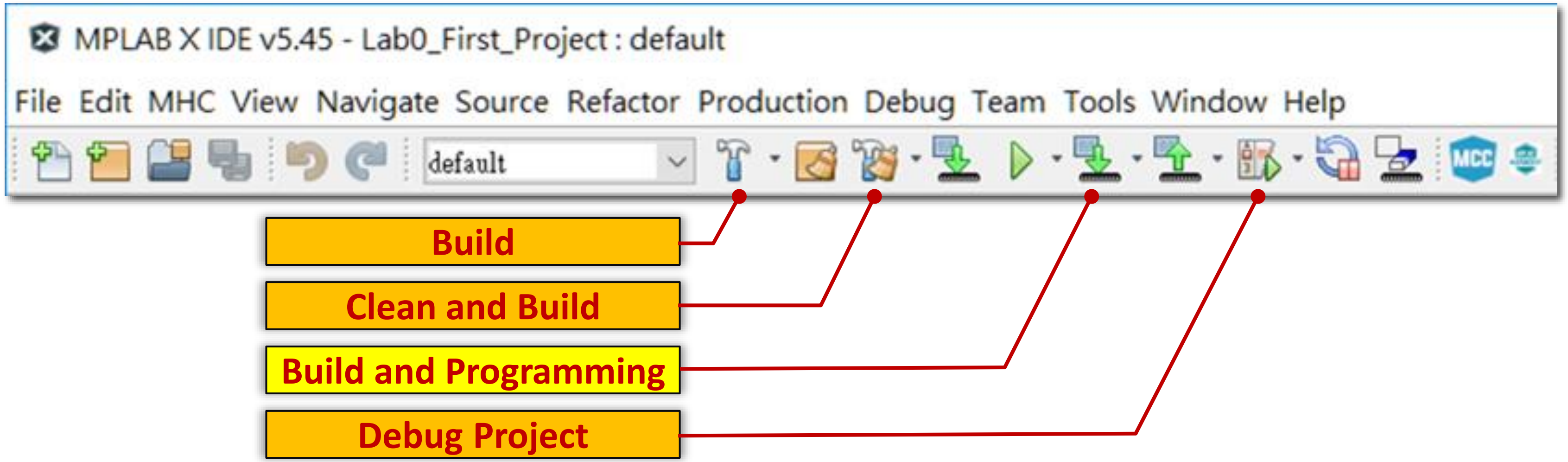
- Click Icon  ,  or press **F11** to build your project.
- Compile result message will show to output windows.



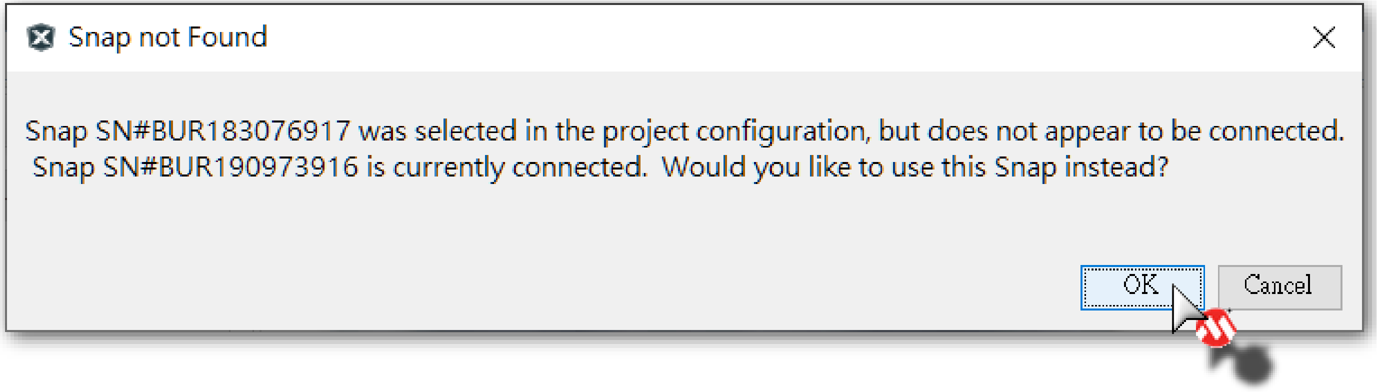
Lab0 Programming Project

Step 16

- Click Icon  build and program firmware to EVB.



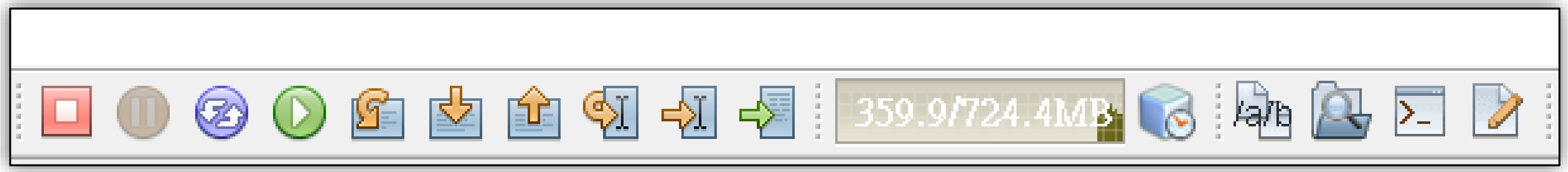
- Different Programmer SN of your project while programming.



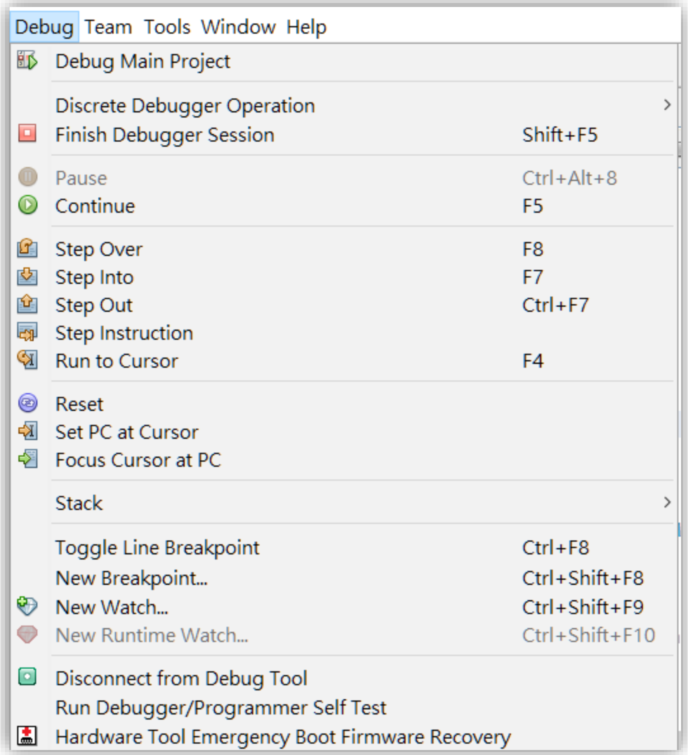
Lab0 Debug Project

Step 17

- **MPLAB X IDE provide a lot of debug window**



- **Select Debug**



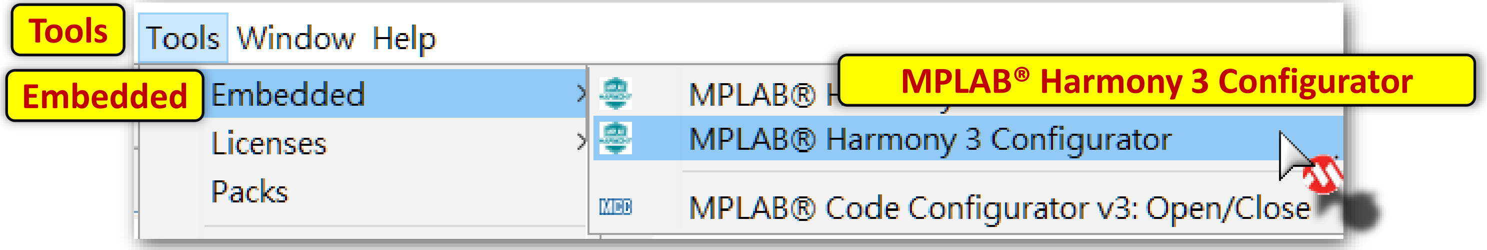
MICROCHIP

MPLAB Harmony Configurator (MHC) Interface introduction

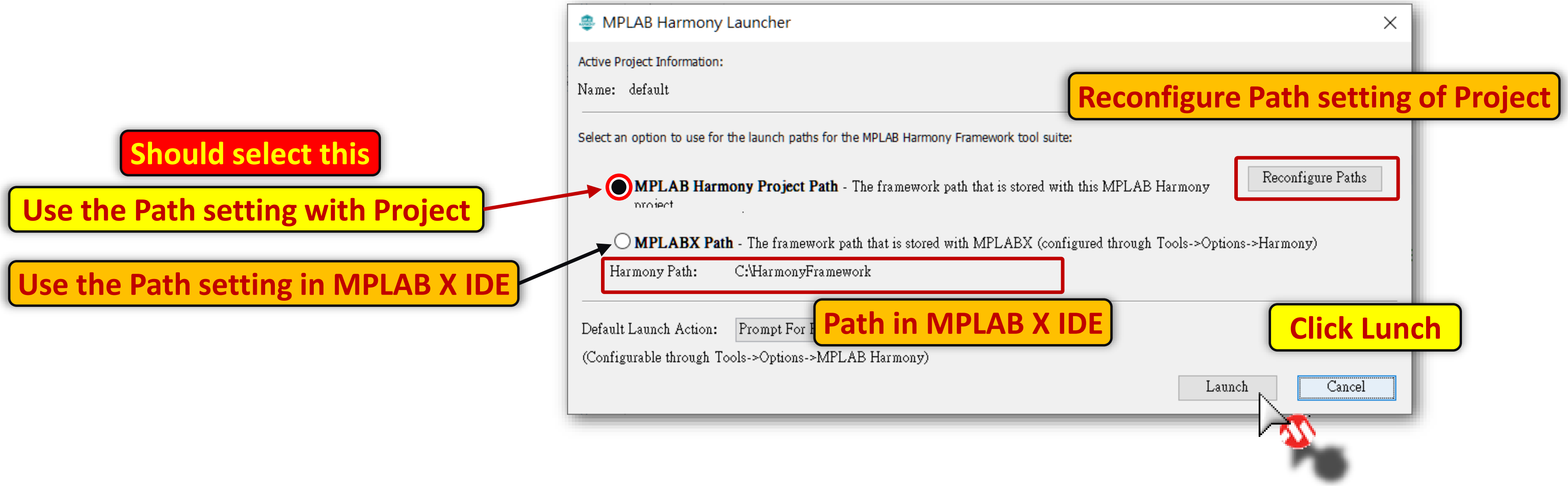
MICROCHIP

MHC interface

- **Notice！ If you already close MHC, please open it through below steps.**
- **Select Tools ▸ Embedded ▸ MPLAB® Harmony 3 Configurator**

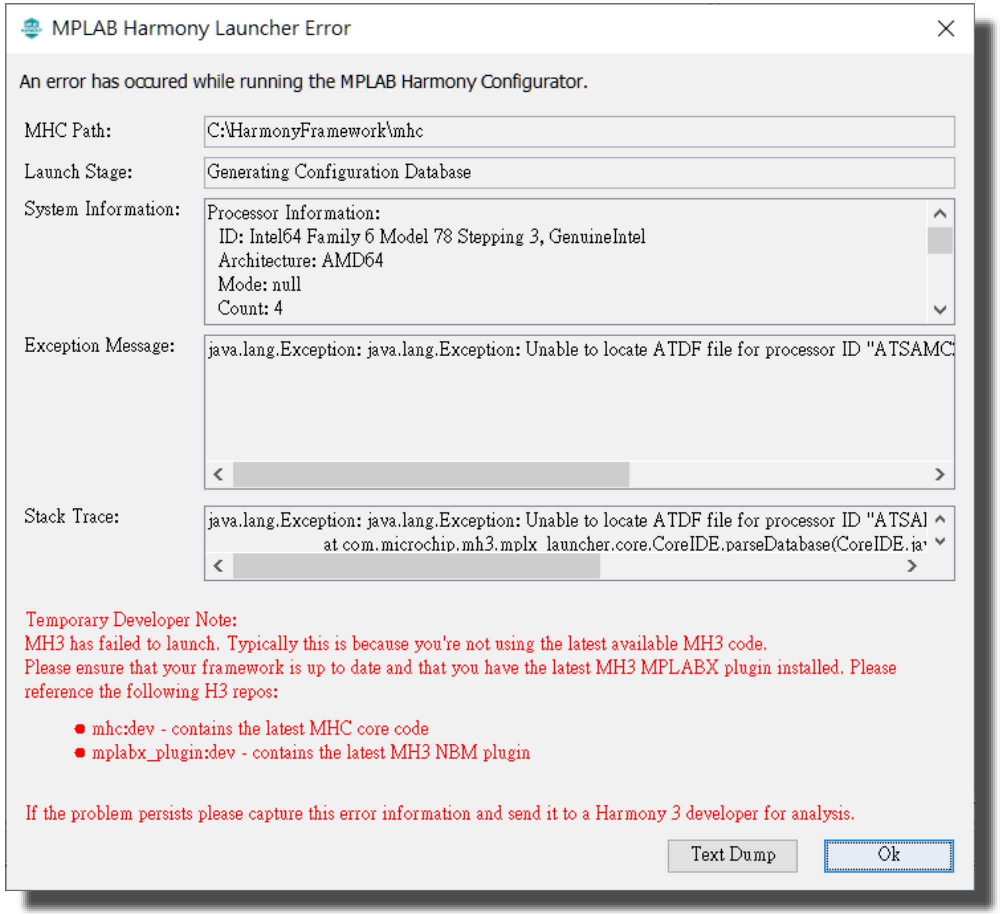


- **Select Harmony Framework location path.**



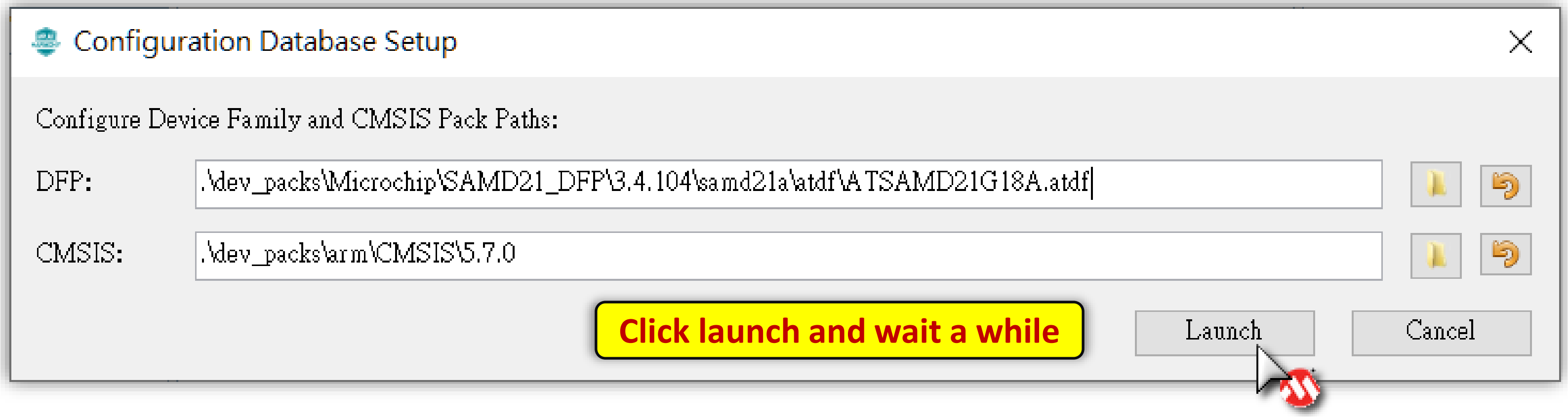
MHC interface

- If something wrong while opening the Harmony 3 Configurator just as below, please make sure your Harmony Framework had download completed.
- For example,
  if SAMC21/SAML21 is the devices to develop but to use the reduced version Framework of this training, you will get failure because the reduced version only reserve the SAMD21 related component in Framework.



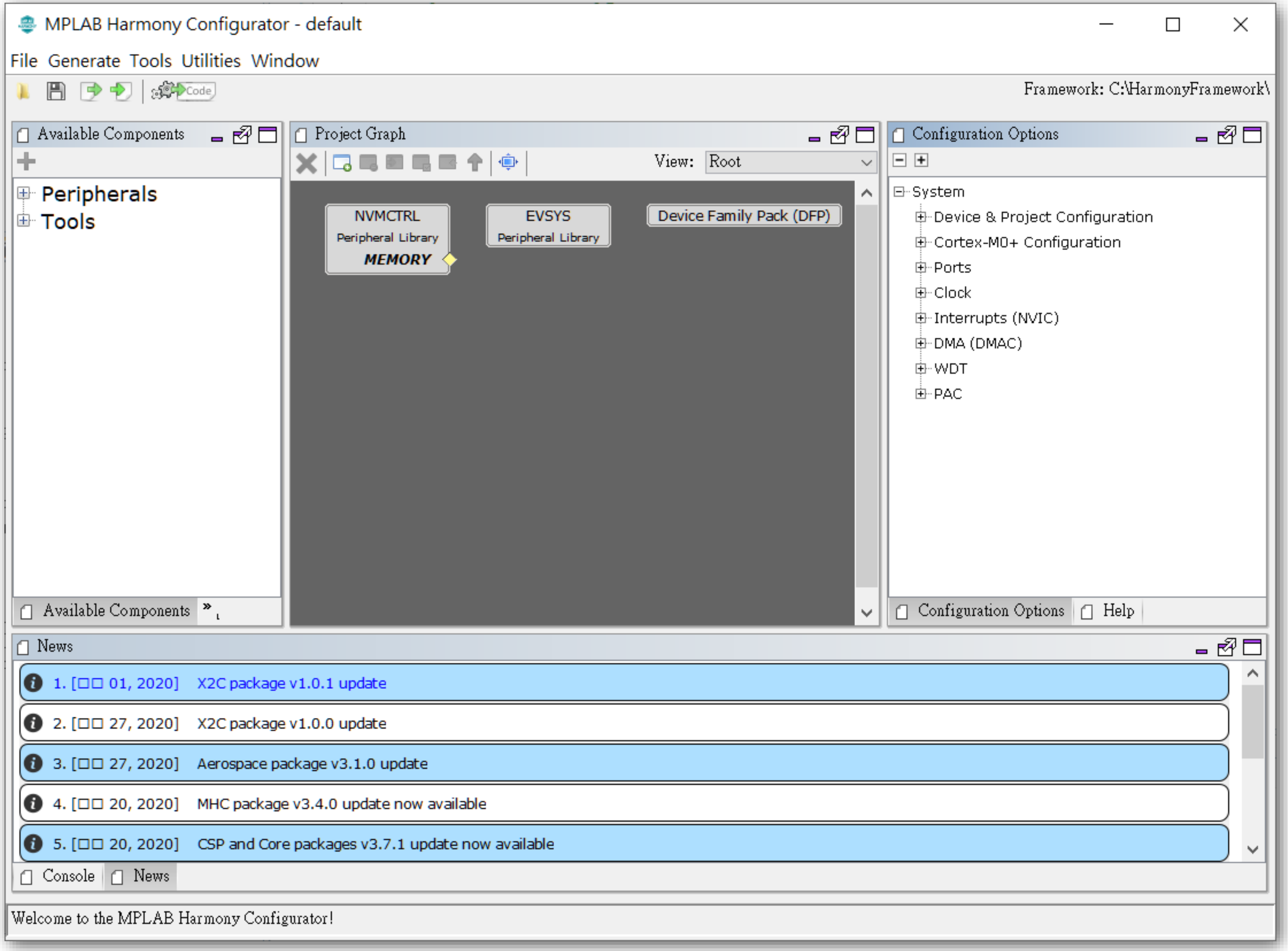
MHC interface

- **Select DFP and CMSIS.** **(Use default is fine)**



MHC interface

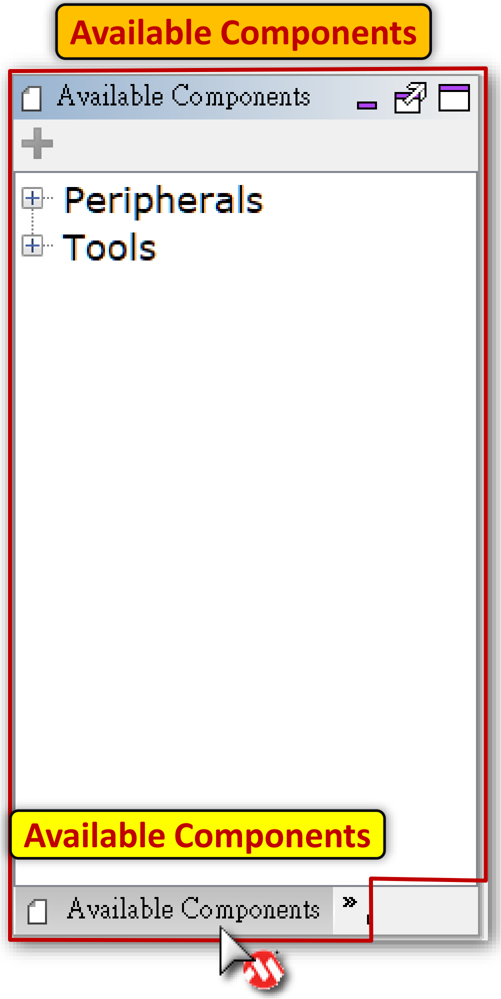
- Now let’s walk through MPLAB Harmony Configurator (MHC).

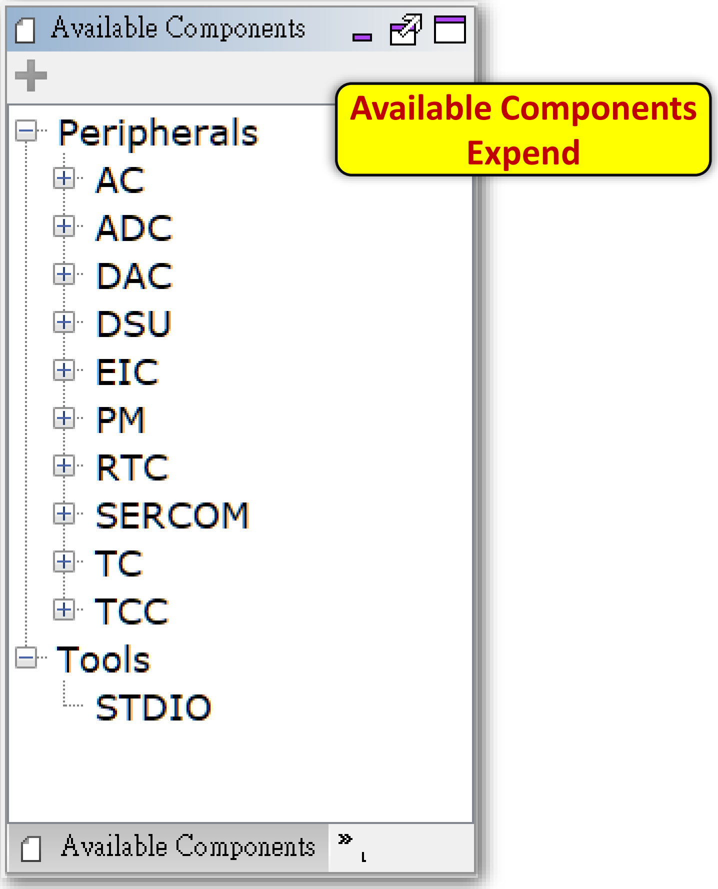


MHC interface

- **Available Components. (Modules you could add-in project)**

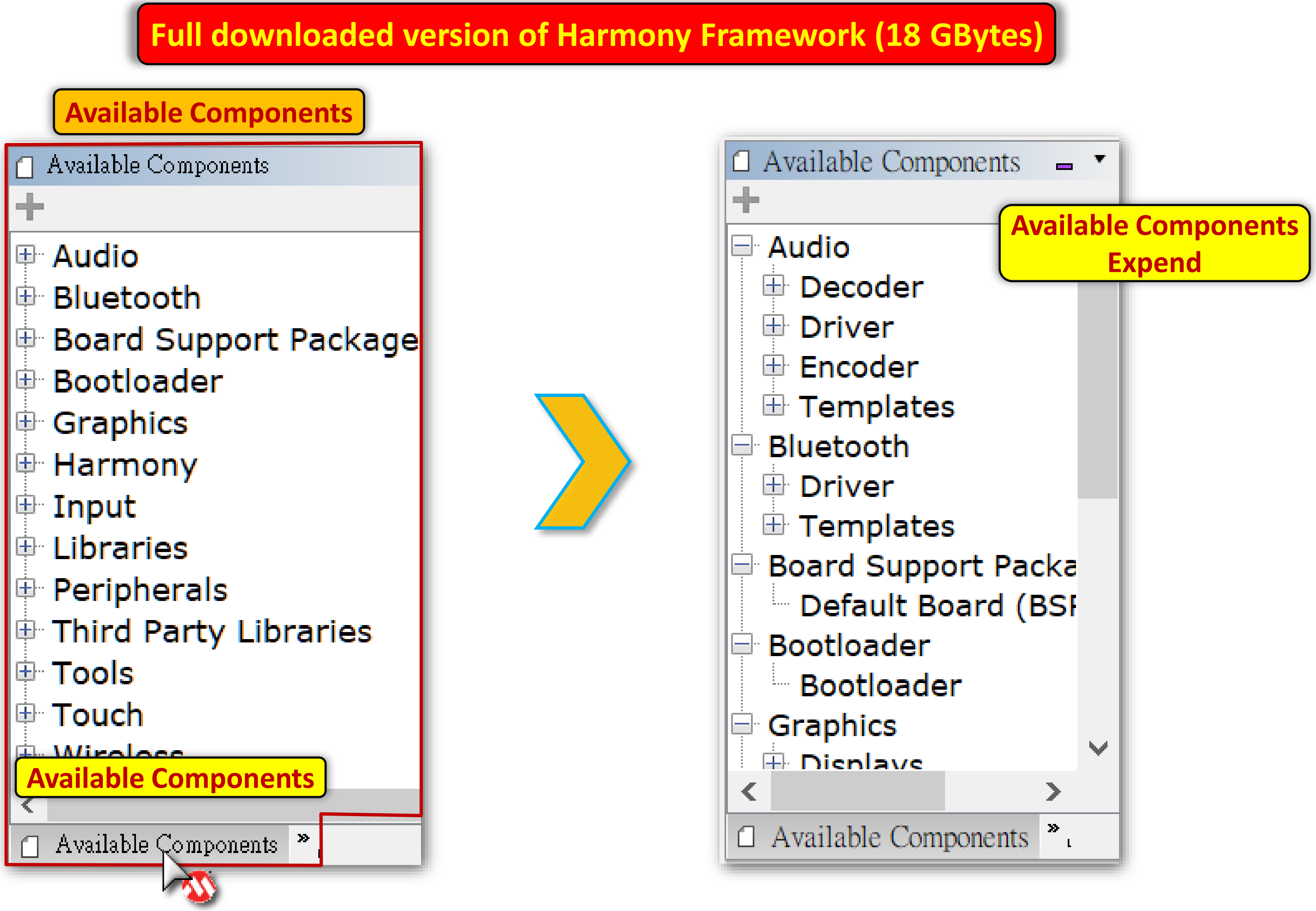
Class use simplify version Harmony Framework (1GBytes)





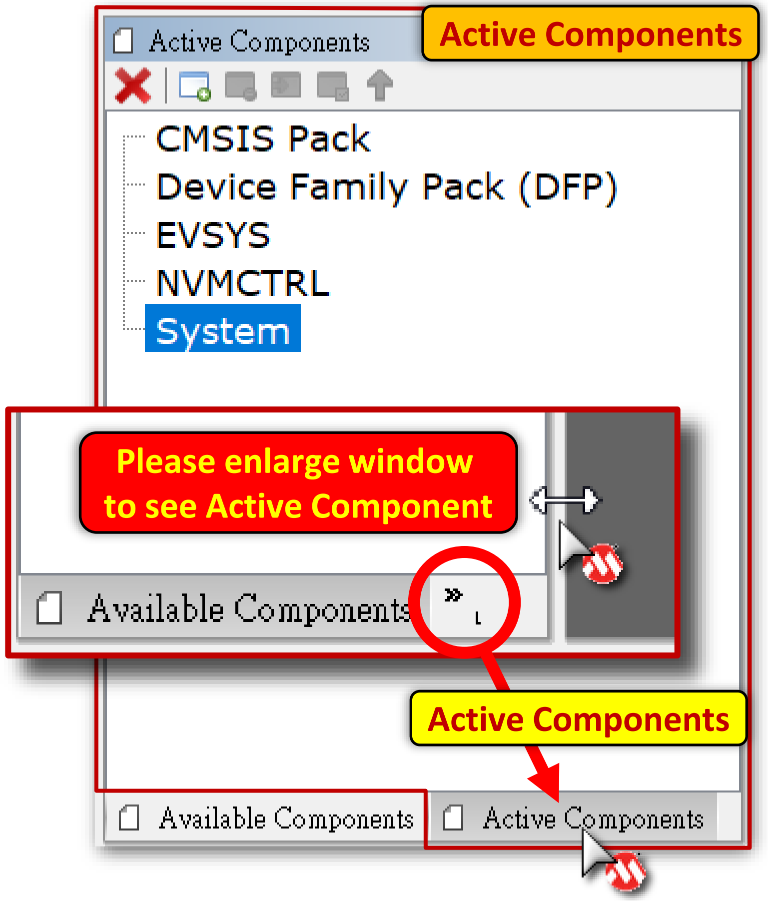
MHC interface

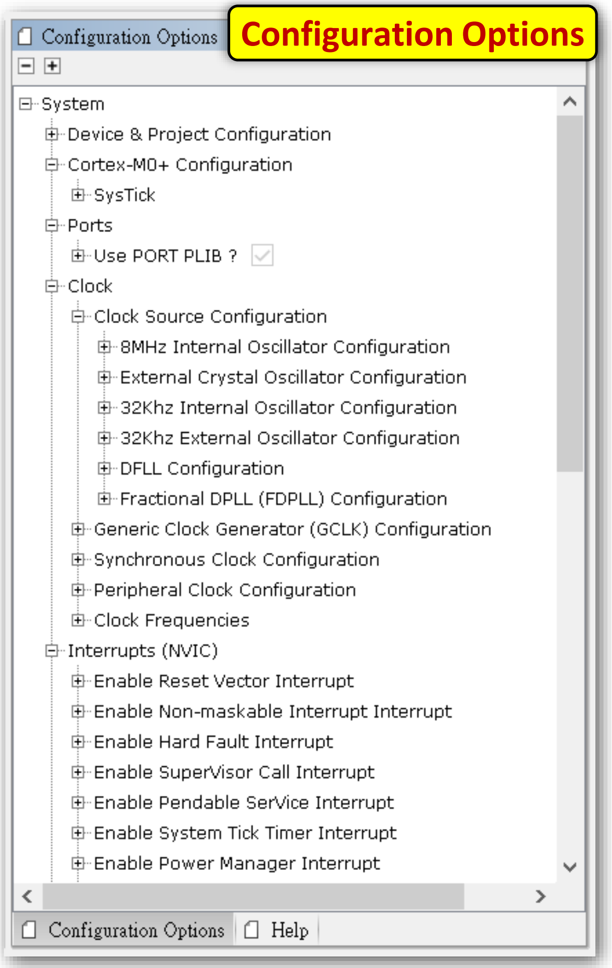
- Available Components. (Modules you could add-in project)



MHC interface

- Active Components. (Current modules in project)
- Configuration Options

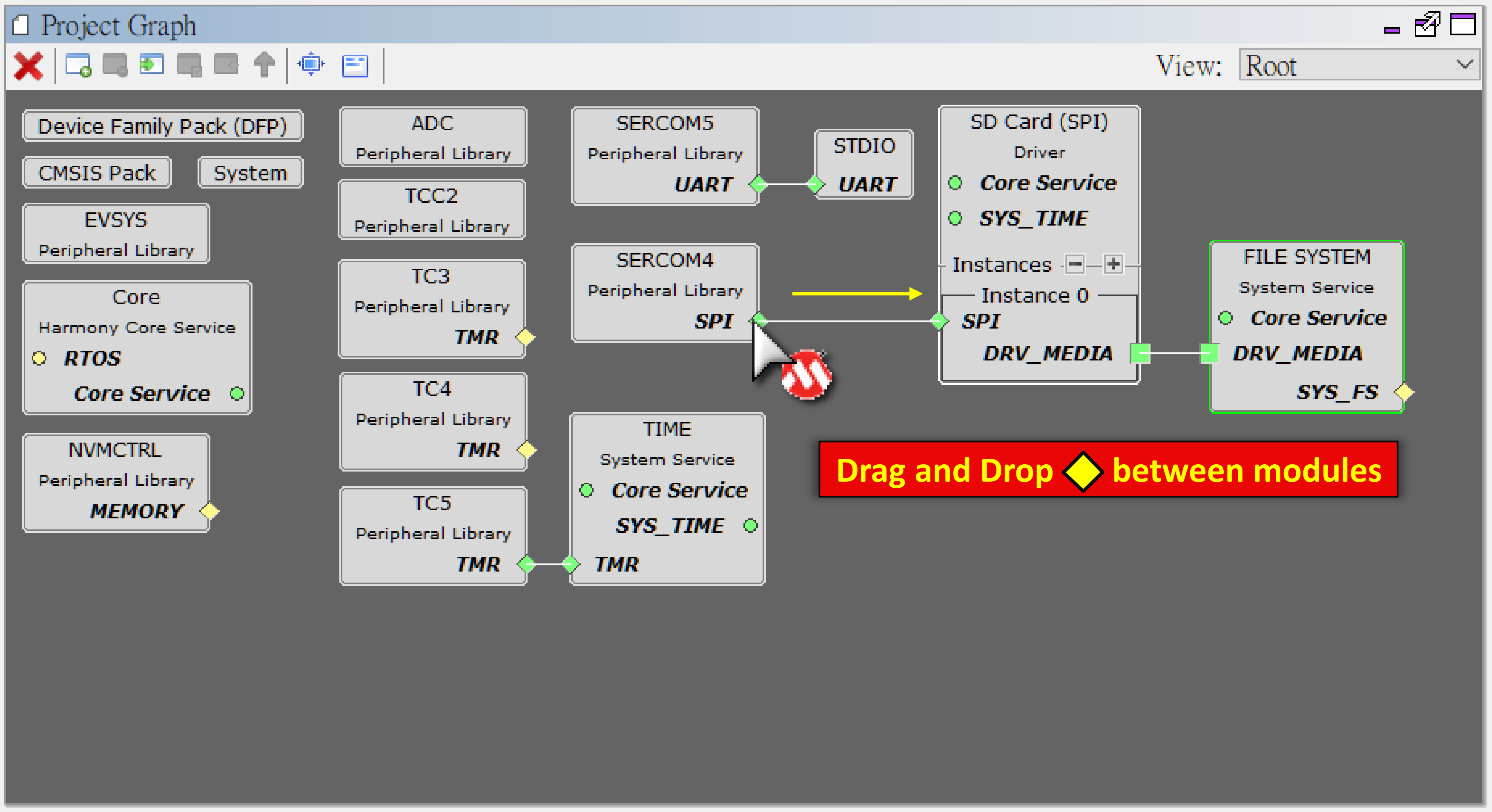




MICROCHIP

MHC interface

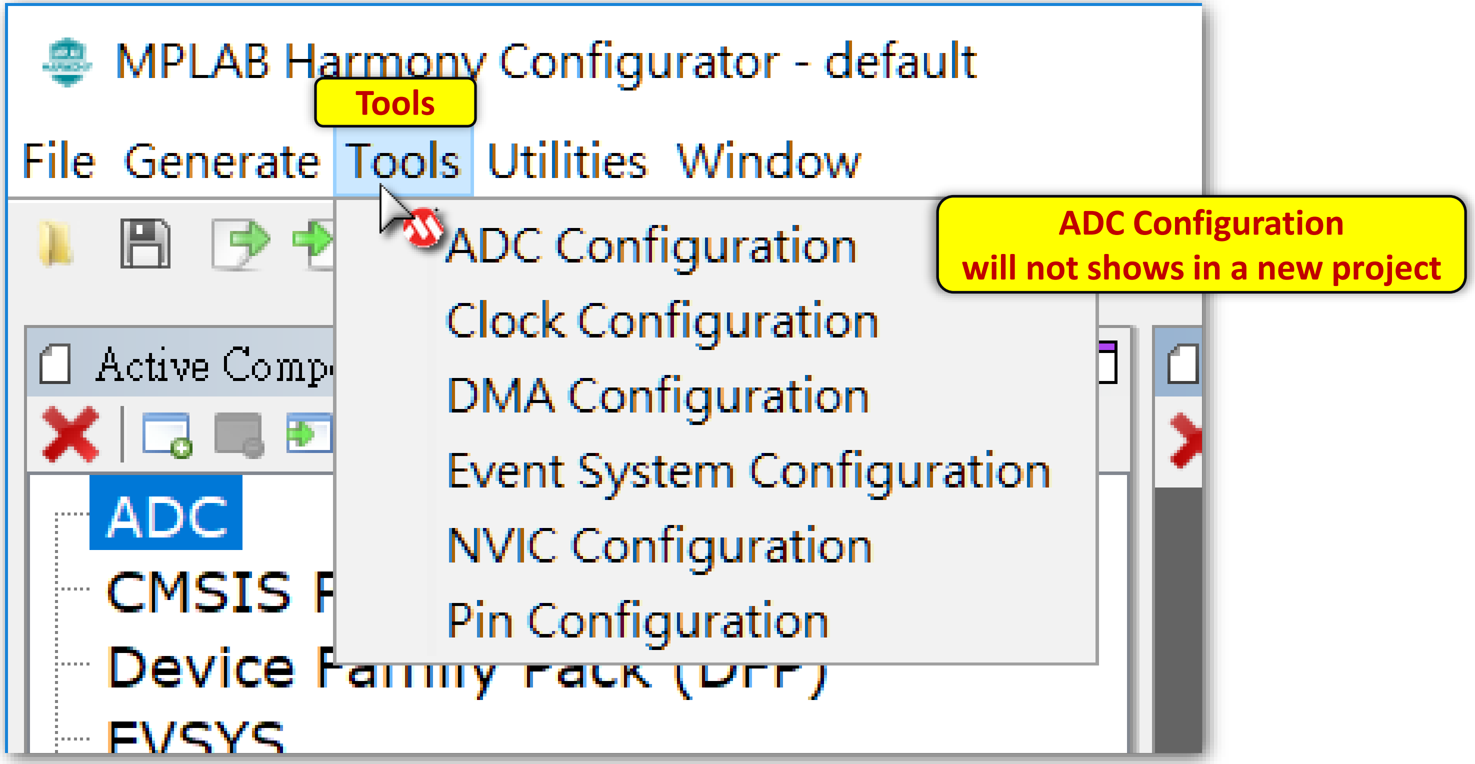
- **Project Graphic. (Project function block diagram)**



MHC interface

Easy View Configuration

- Tools (Easy View Configuration)

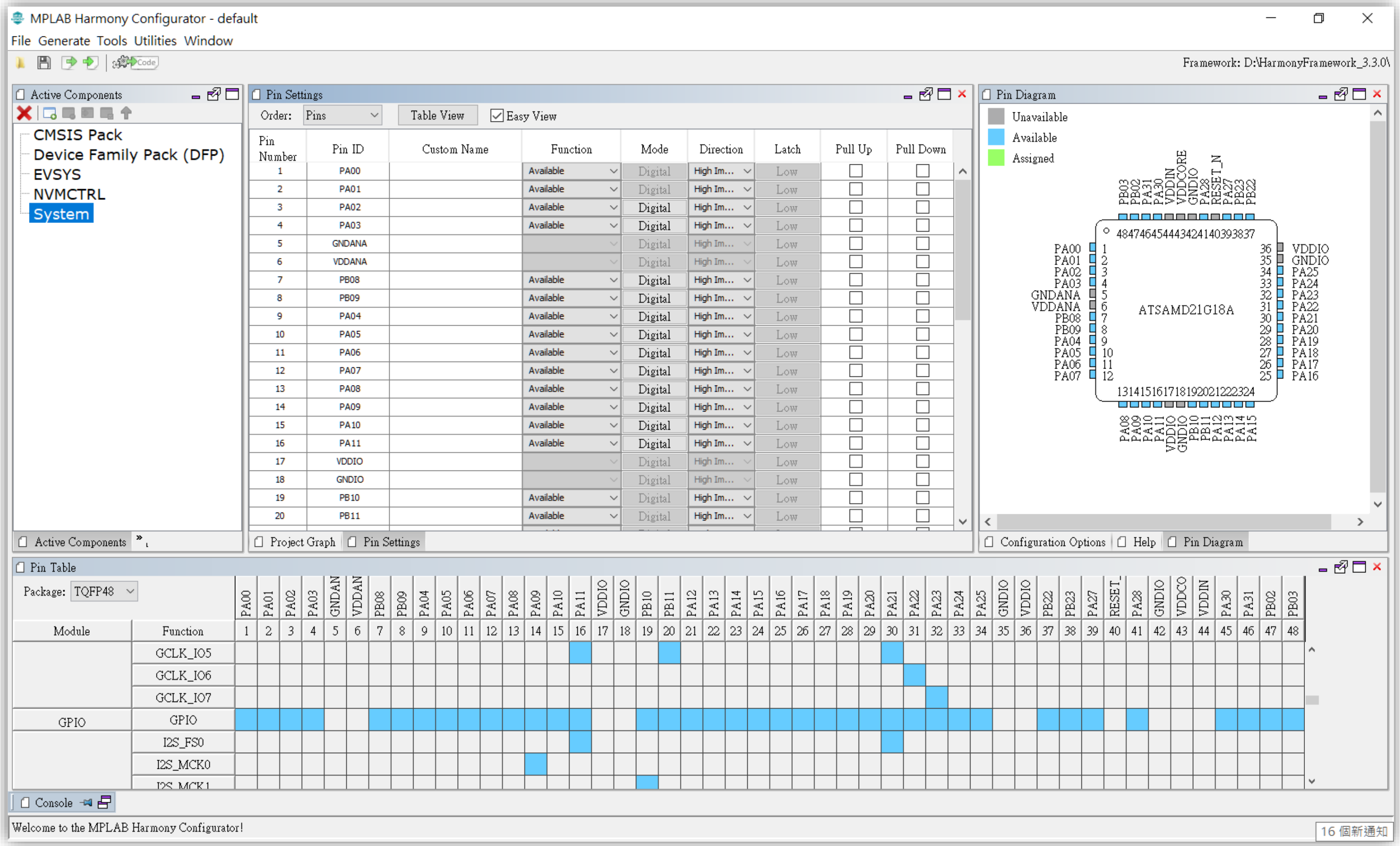


Easy View Configuration Tools will shows depend on what peripheral you already added-in.

MHC interface

Easy View Configuration

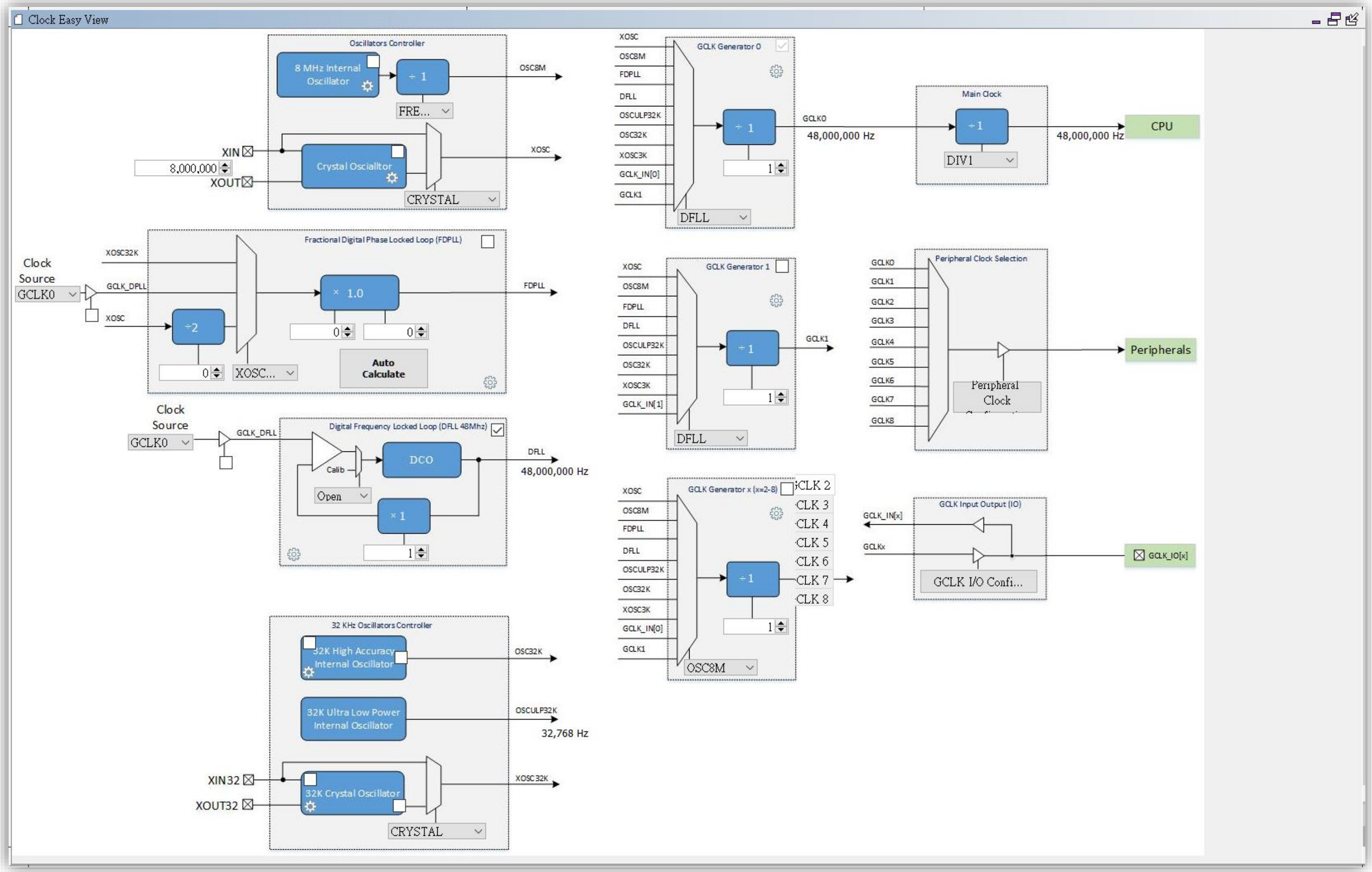
- Tools : Pin Configuration



MHC interface

Easy View Configuration

- Tools : Clock Configuration



MHC interface

Easy View Configuration

- Tools : NVIC Configuration

NVIC Settings

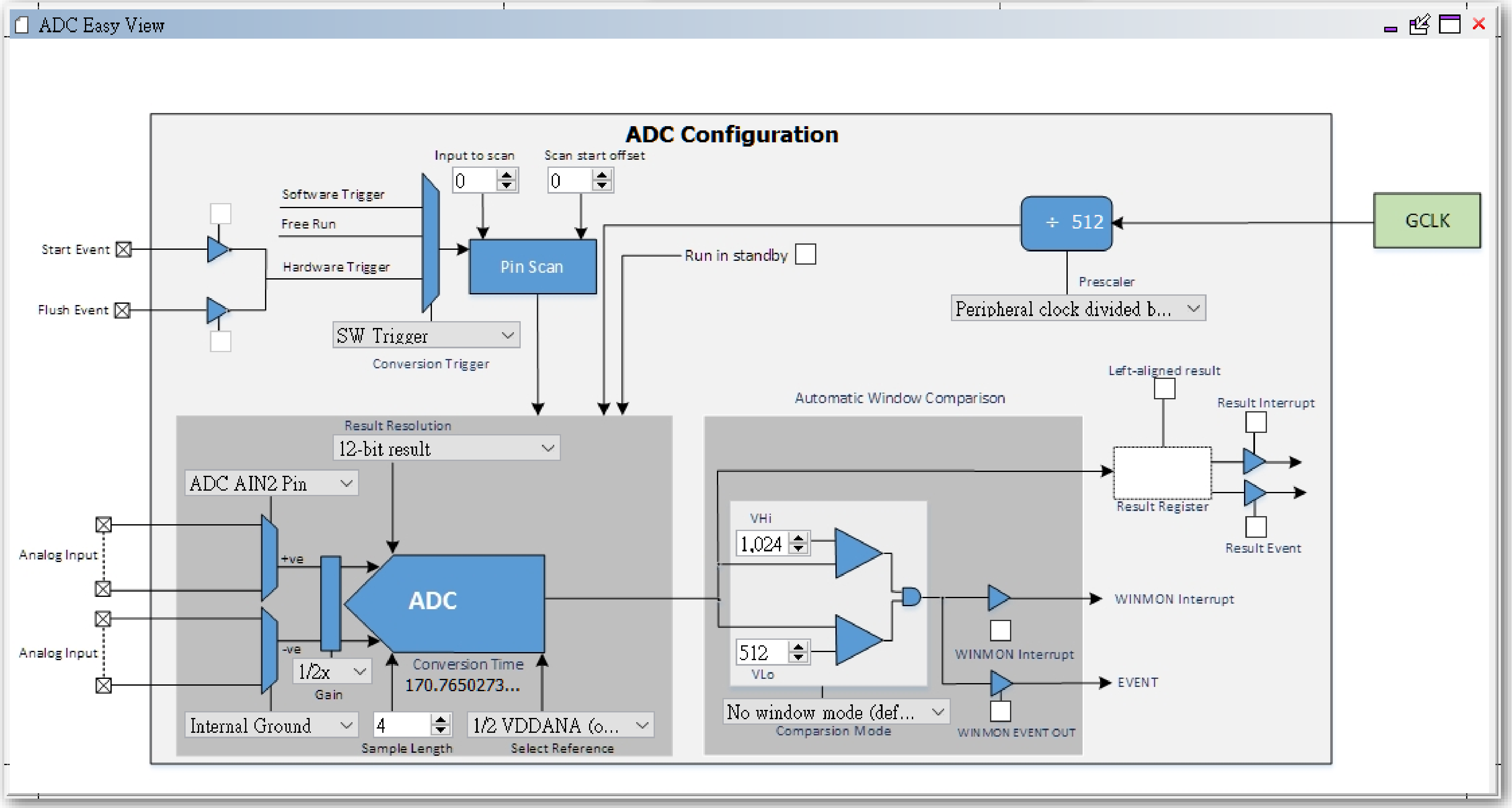
Vector Number	Vector	Enable	Priority (0 =	Handler Name
-15	Reset (Reset Vector)	☑	-3	Reset_Handler
-14	NonMaskableInt (Non-maskable Interrupt)	☑	-2	NonMaskableInt_Handler
-13	HardFault (Hard Fault)	☑	-1	HardFault_Handler
-5	SVCall (SuperVisor Call)	☑	0	SVCall_Handler
-2	PendSV (Pendable SerVice)	☑	0	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_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

MHC interface

Easy View Configuration

- Tools : ADC Configuration

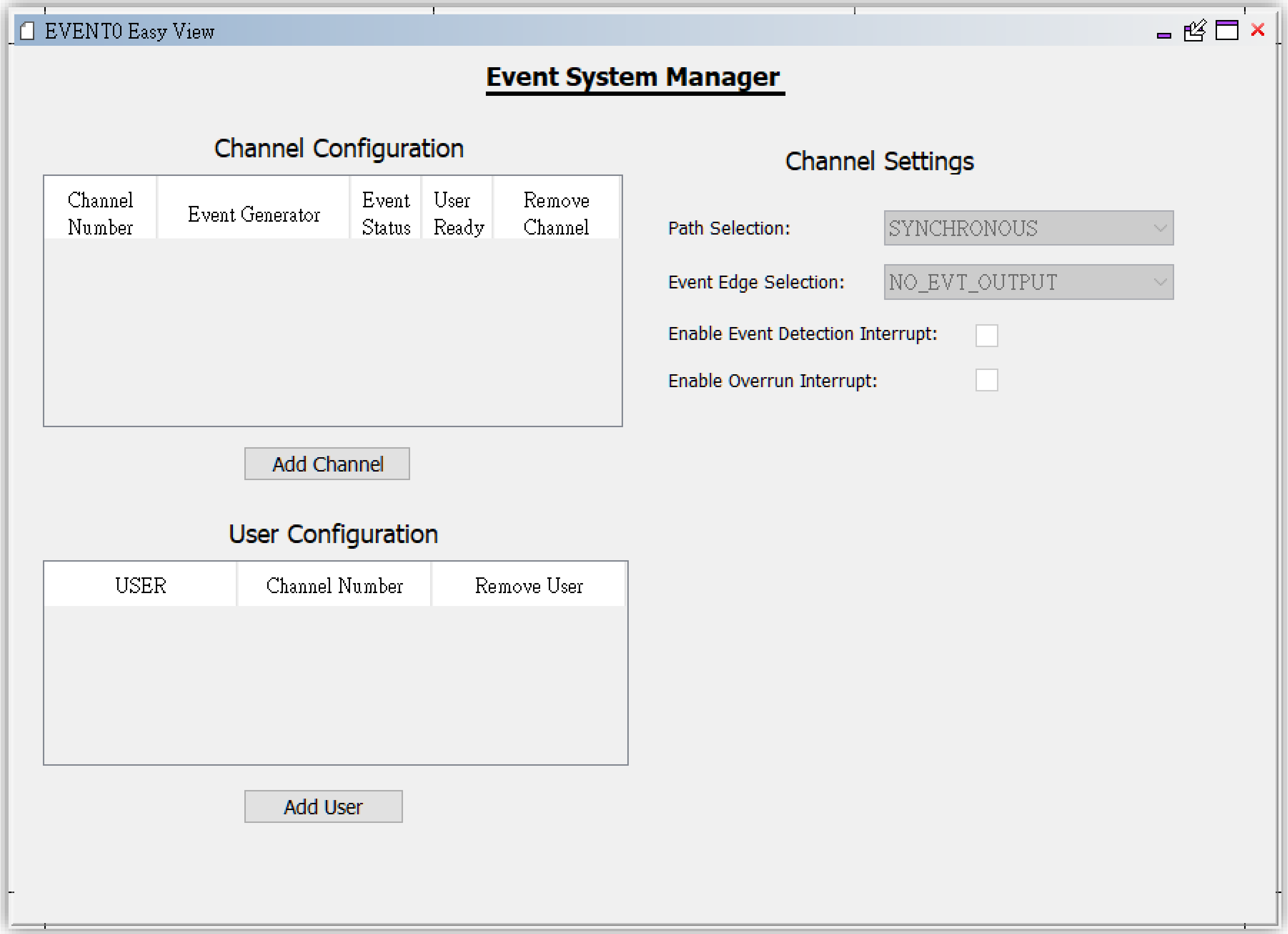
ADC Configuration
will not shows in a new project



MHC interface

Easy View Configuration

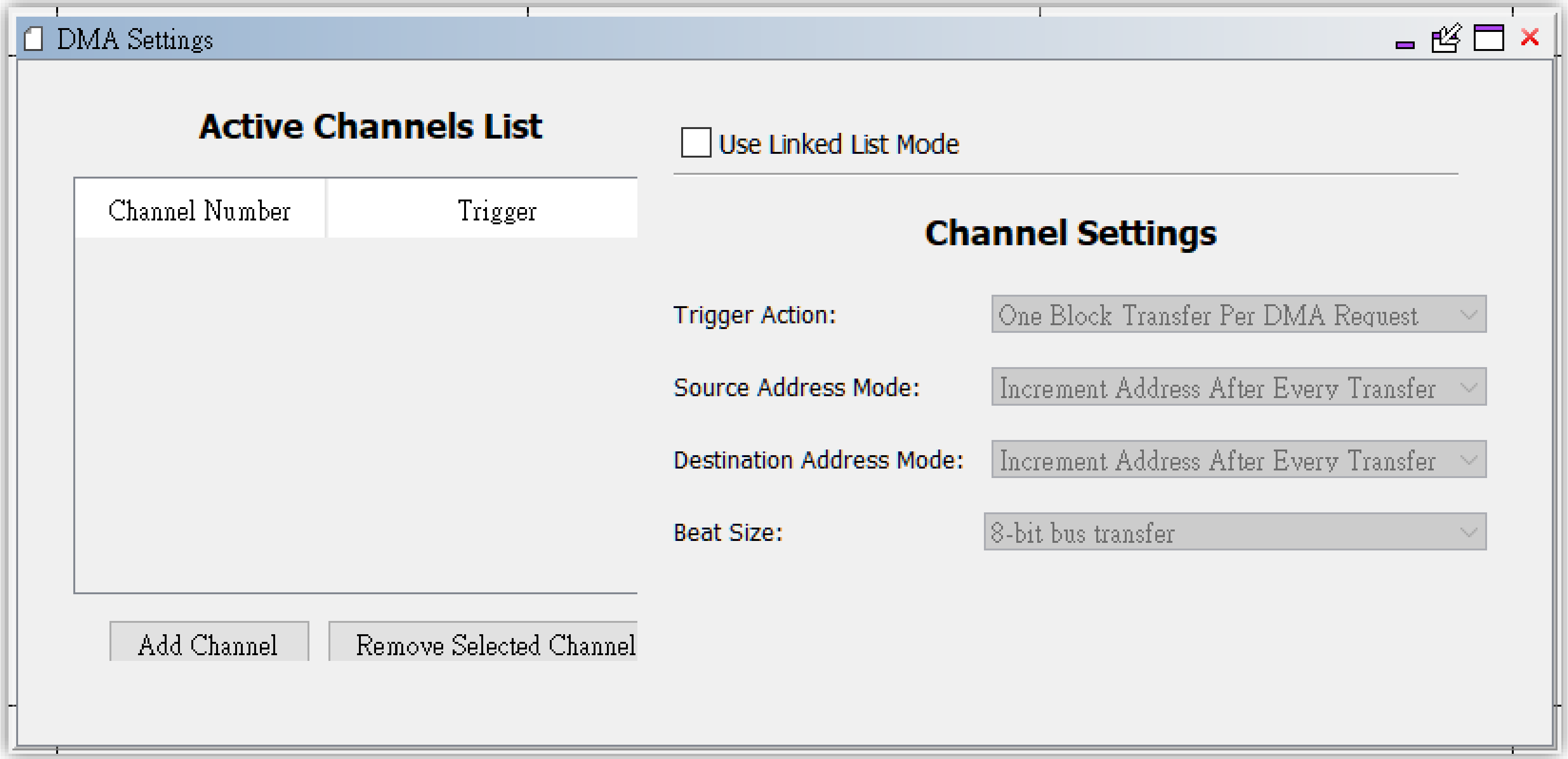
- **Tools : Event System Configuration**



MHC interface

Easy View Configuration

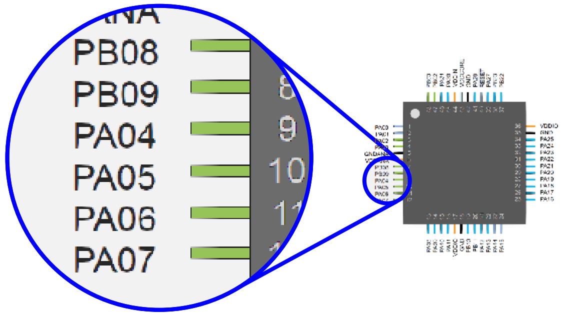
- Tools : DMA Configuration

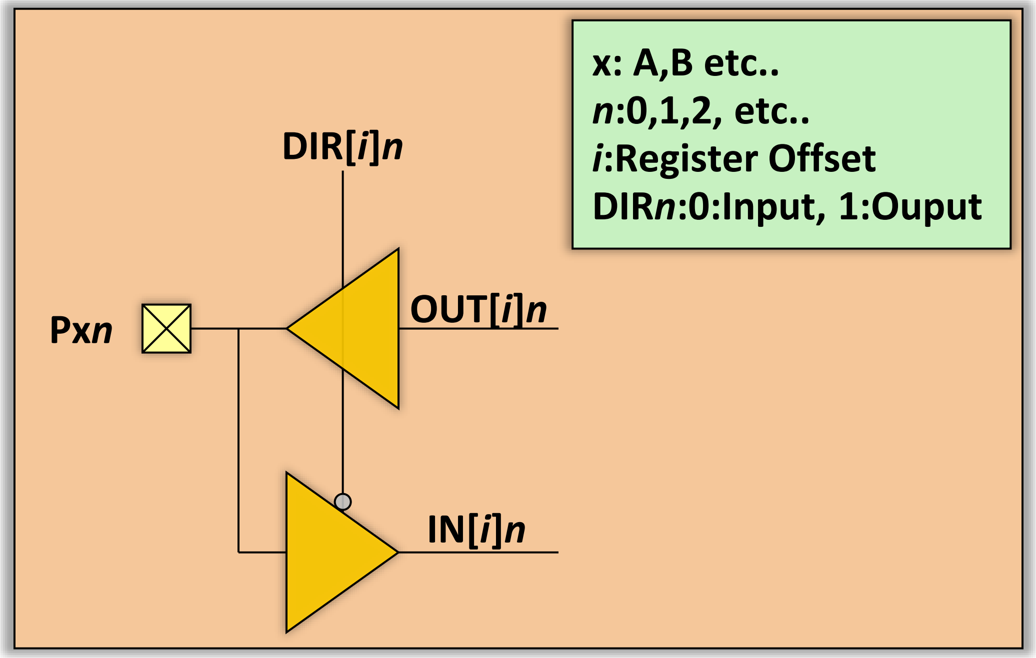


PORT I/O Architecture

PORT Block Diagram

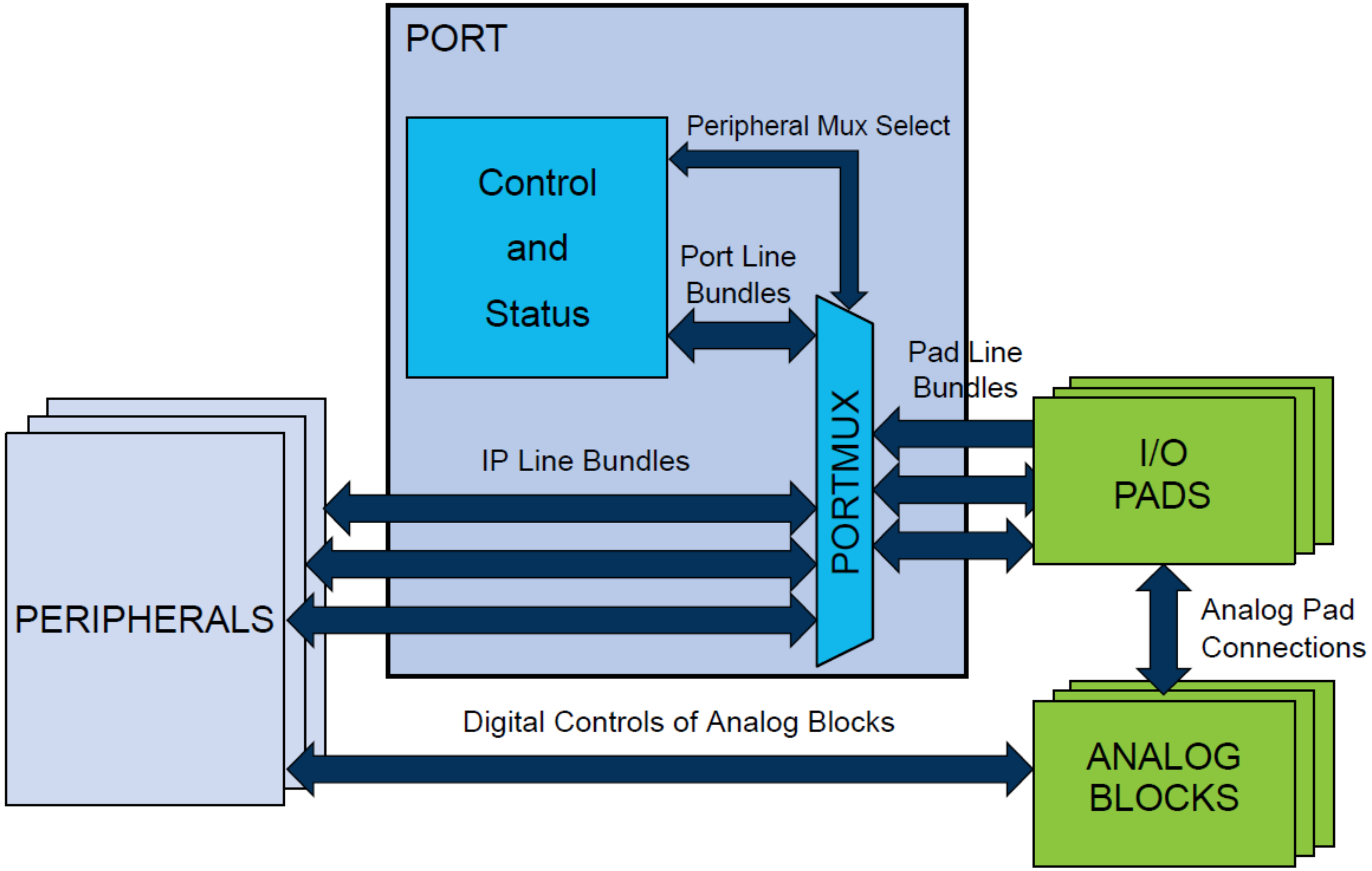
- Please refer to the block diagram, this is the concept for programming model. The real and detail block diagram please refer to the following slide.
- DIR: use to determine in or out. 0 for input, 1 for output.
- OUT: set output drive value for Output pins.
- IN: reaction logical level from Input pins.



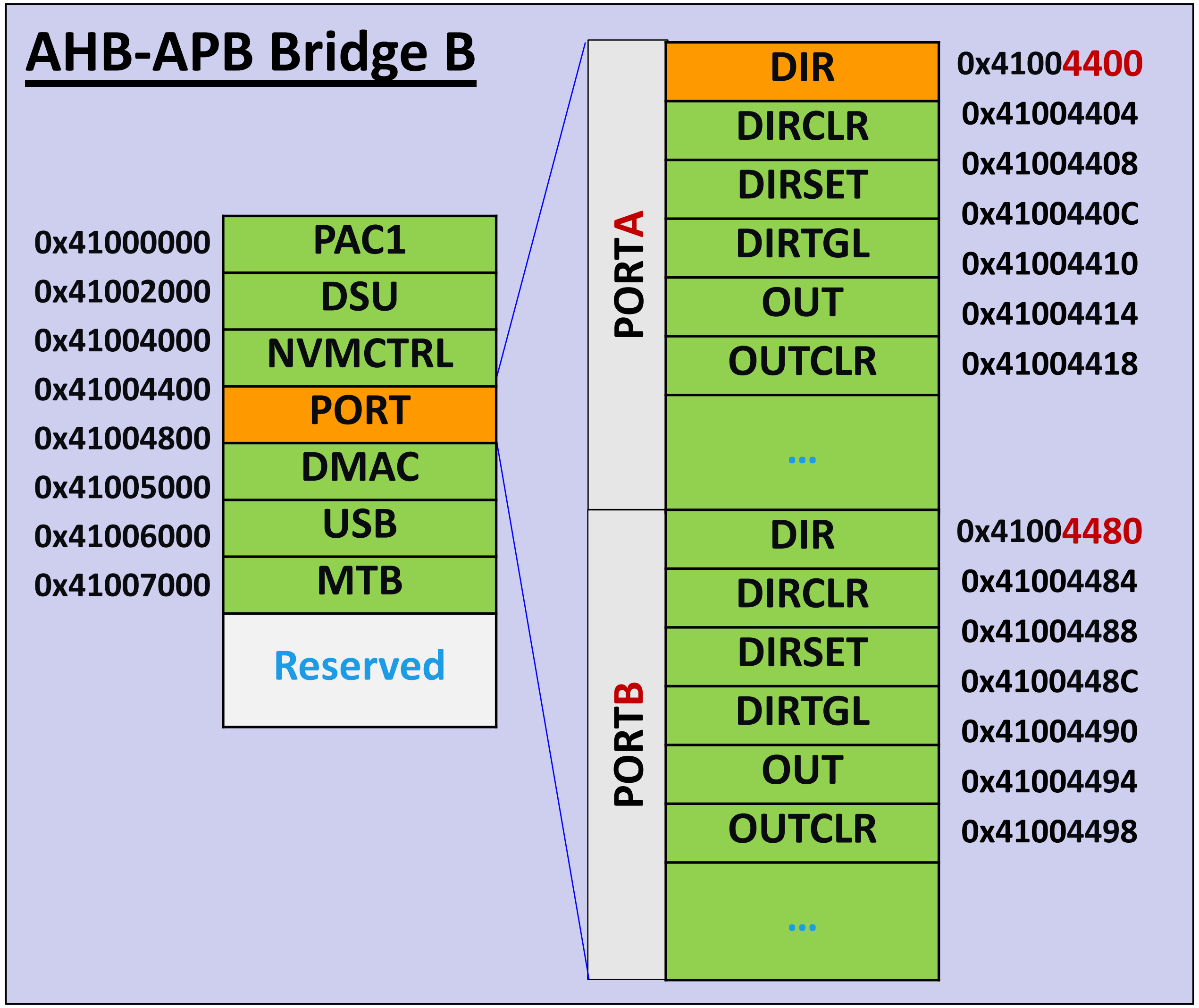


PORT Block Diagram

- **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;
```

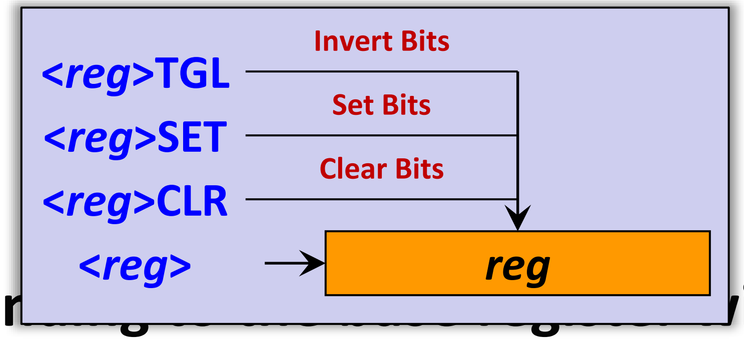
port.h

```
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

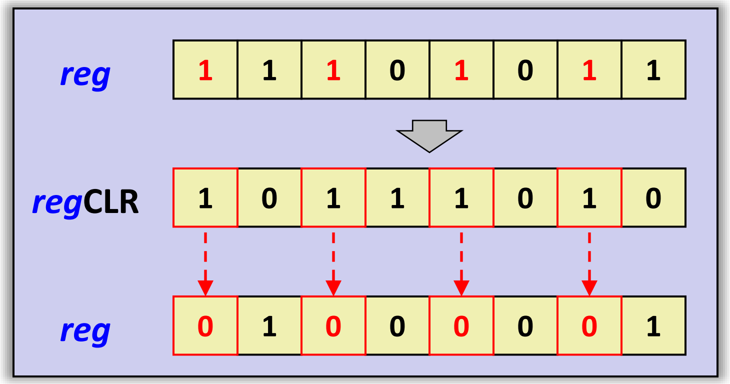
- **Some register provide *<reg>*CLR, *<reg>*SET , *<reg>*TGL register for atomic bits manipulation.**

- **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)

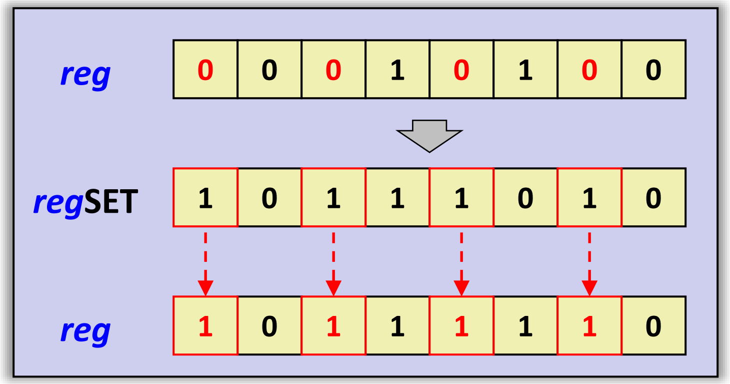


- **Setting the manipulation register bits correspon… will set the bits to 1, clear to 0 or toggle.**

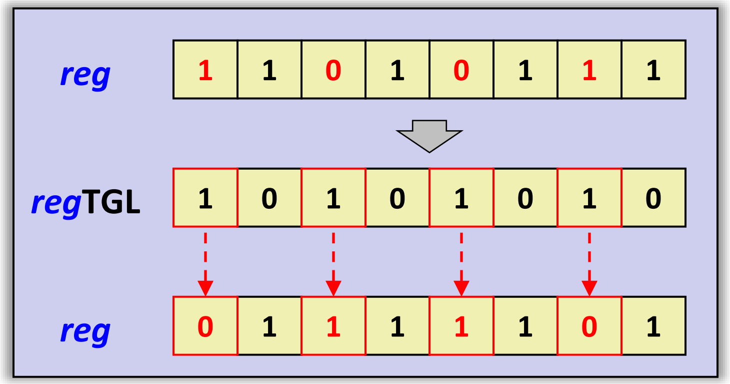
Atomic Bits Manipulation



*reg*CLR : corresponding bits clear to 0



*reg*SET : corresponding bits set to 1



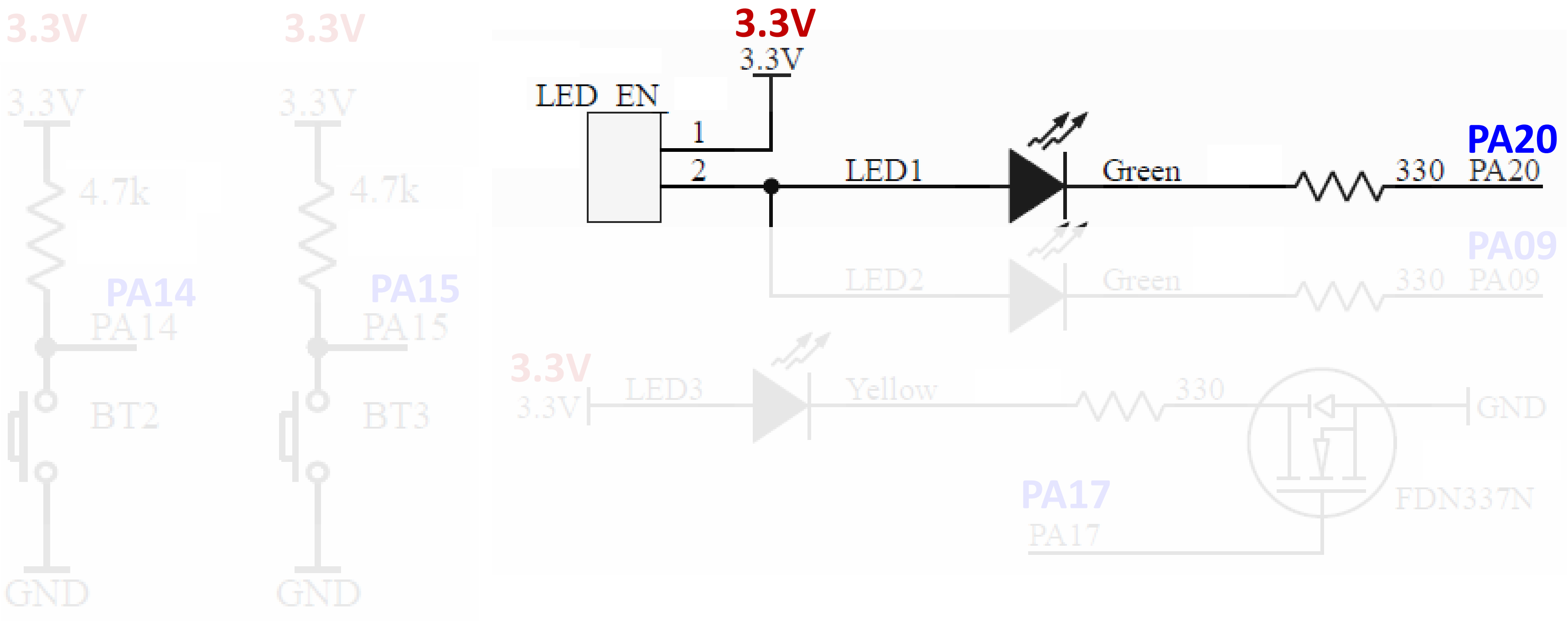
*reg*TGL : corresponding bits toggled
(0 -> 1,   1 -> 0)

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



LED1	PA20
LED2	PA09
LED3	PA17
BT2	PA14
BT3	PA15

Lab1 PORT Output

Step 1

a Add code segment to your main loop

Access by Register Pointer

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

int main (void)
{
   SYS_Initialize ( NULL );

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

   while(1)
   {
      for ( i = 0 ; i < 2000000 ; i++ )
      {}
      // TODO 1.03
      *(__IO uint32_t *)(0x41004400U + 0x1CU) = 0x00100000;
   }
}
```

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(1)
   {
      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	OUTTGL
0x1D	
0x1E	
0x1F	

Lab1 PORT Output

Step 2

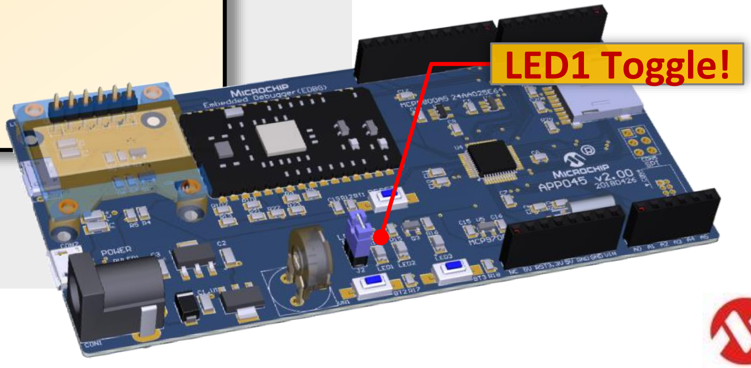
a Modify code segment to your main loop

Access by Pre-defined Struct

Period Control

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

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

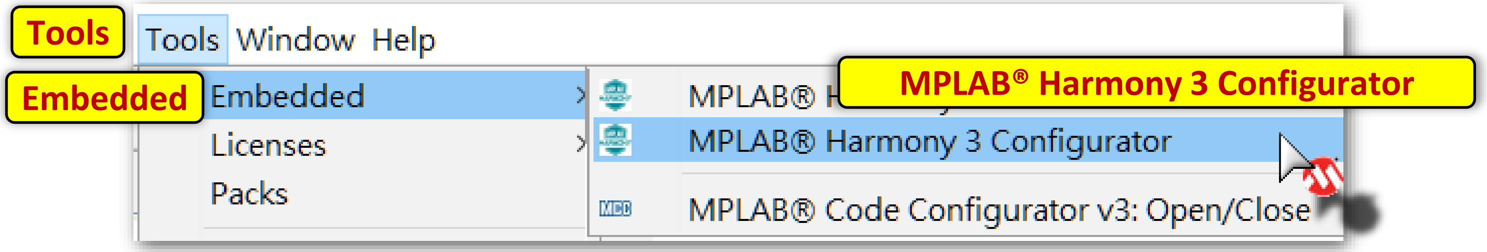
b Program firmware to target board
then observe result.



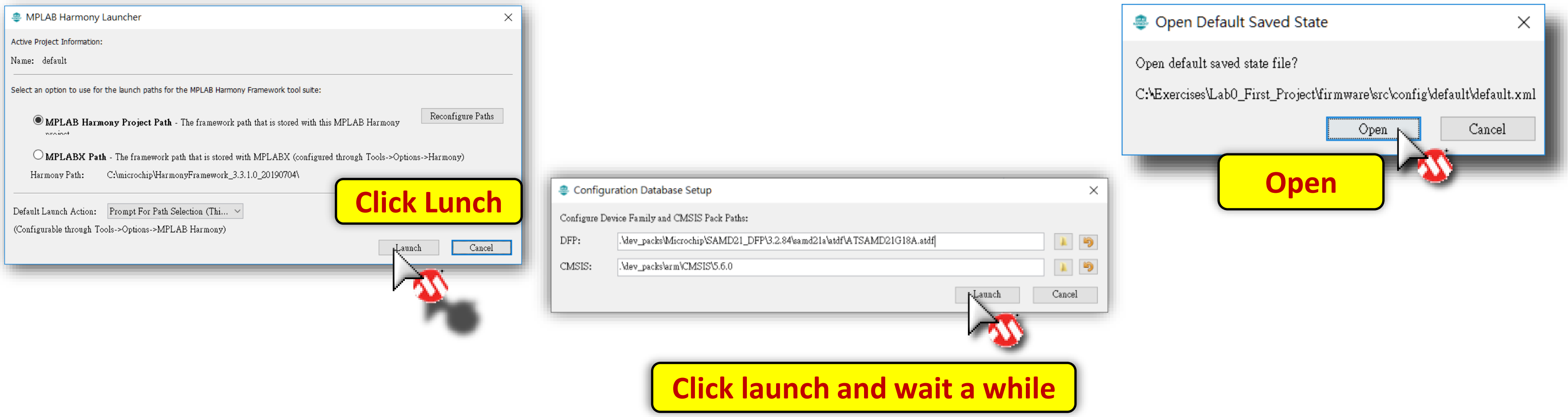
MICROCHIP

Pin Configuration using MHC

- If you already close MHC, please open it through below steps.
- Select Tools ▸ Embedded ▸ MPLAB® Harmony 3 Configurator

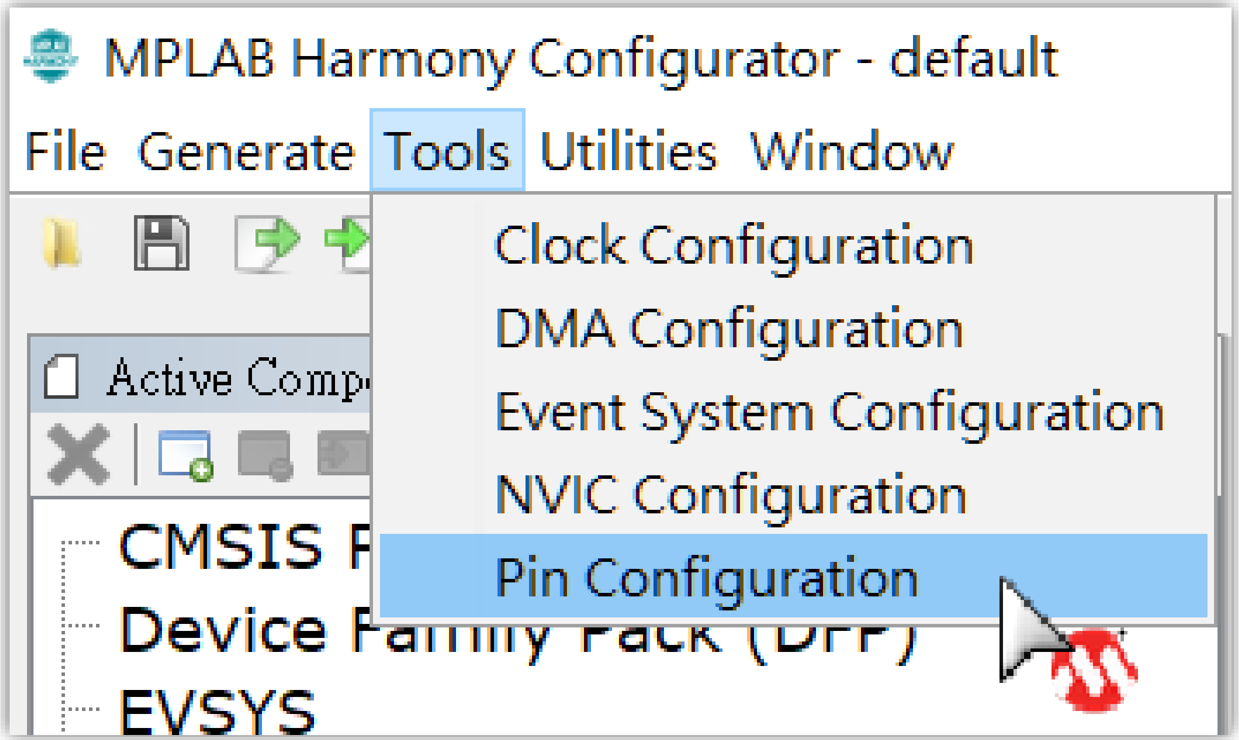


- Following MHC reopen steps reviews.



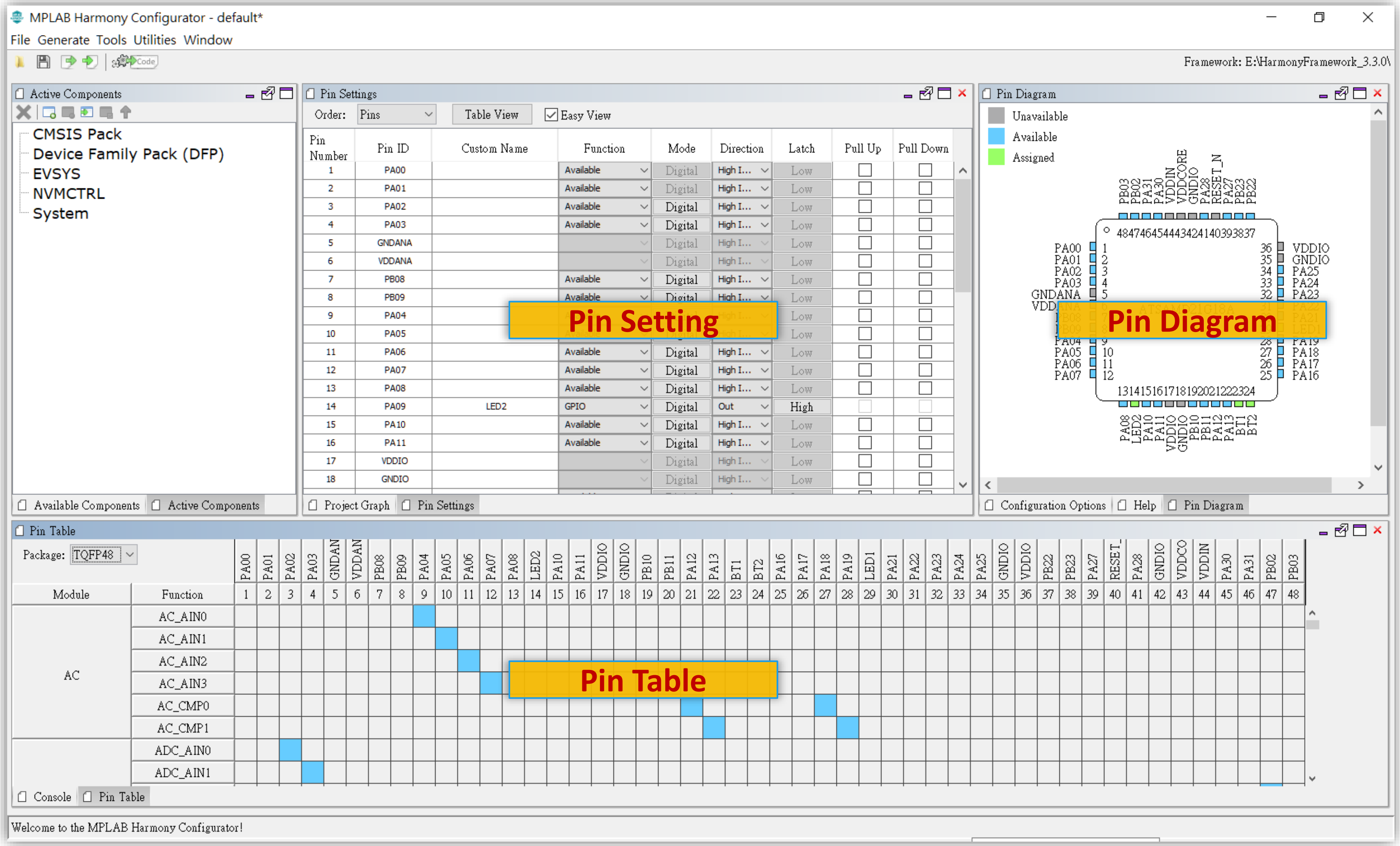
Pin Configuration using MHC

- Return to Harmony Configurator.
- Select Tools ▸ Pin Configuration



Pin Configuration using MHC

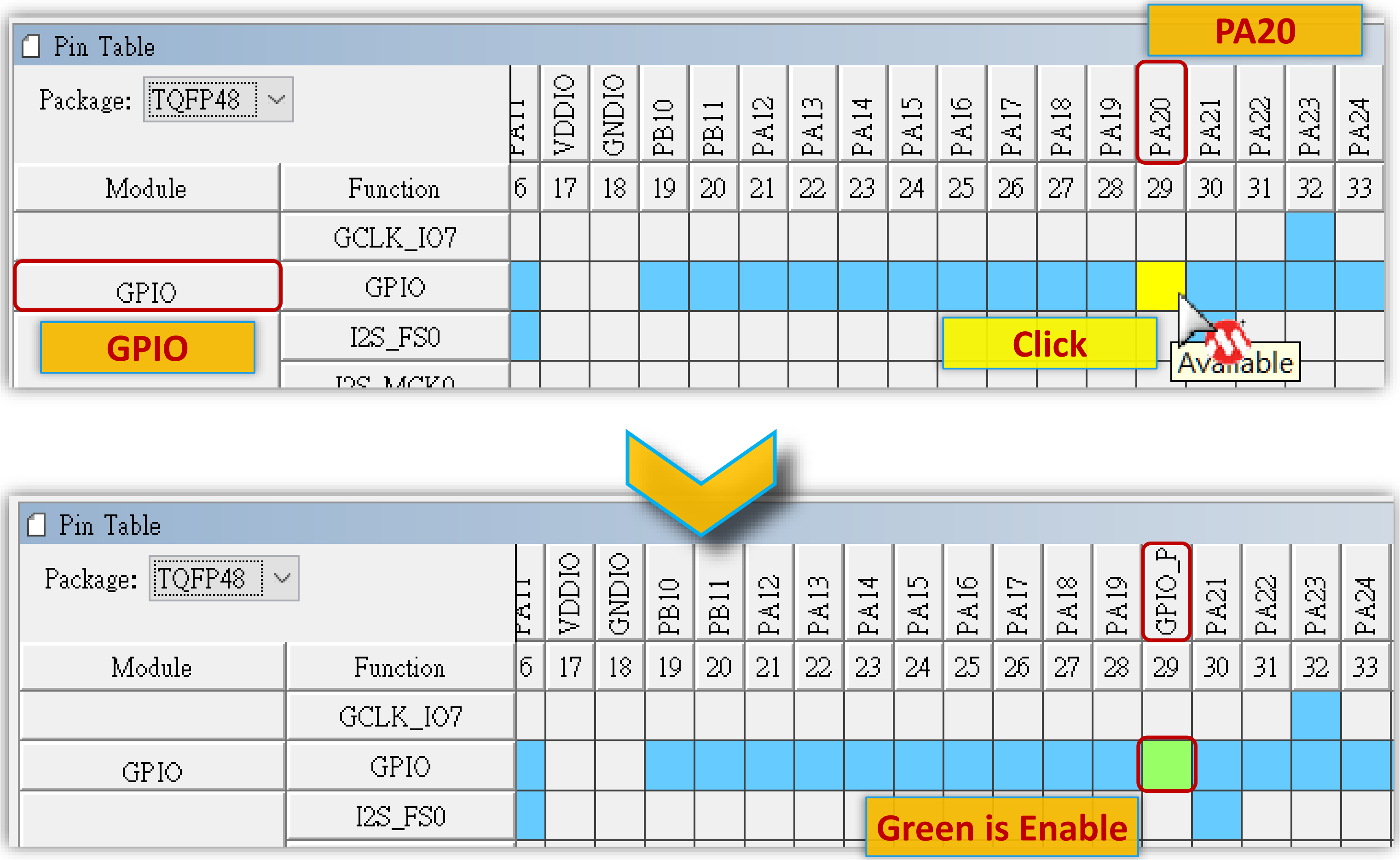
- **Docking three Pin Configuration windows as below.**



Lab1 PORT Output

Step 3

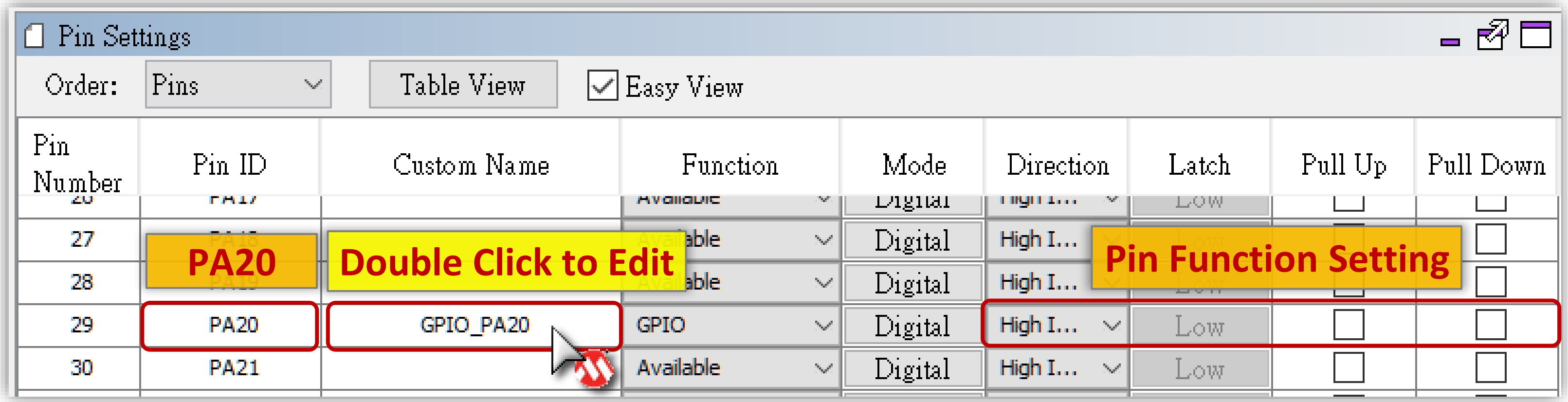
- **Pin Table configure. (Enable or Disable Pin)**

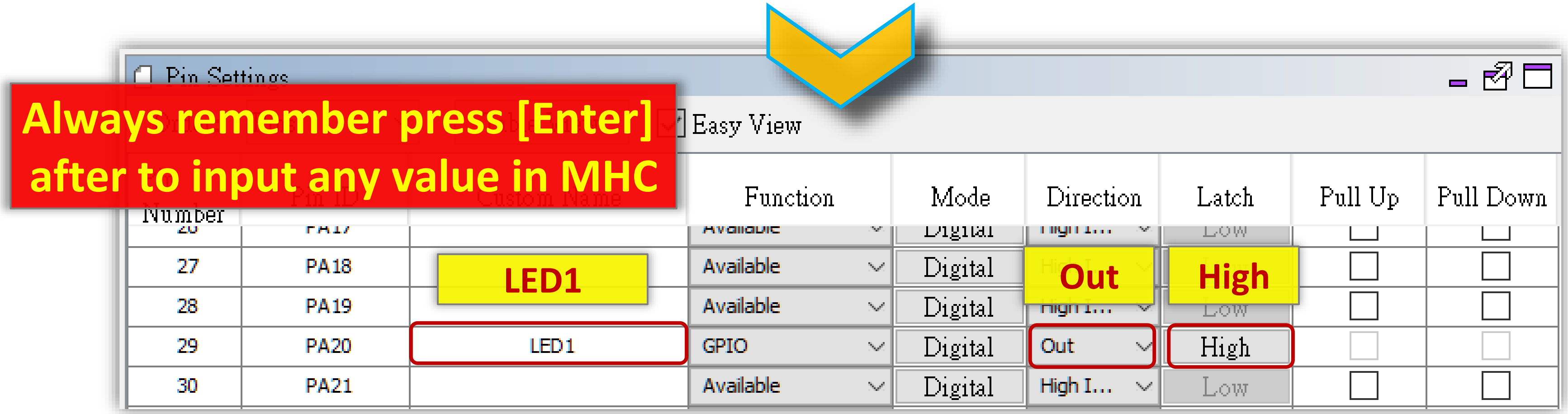


Lab1 PORT Output

Step 4

- **Pin Setting. (Pin Function Setting)**

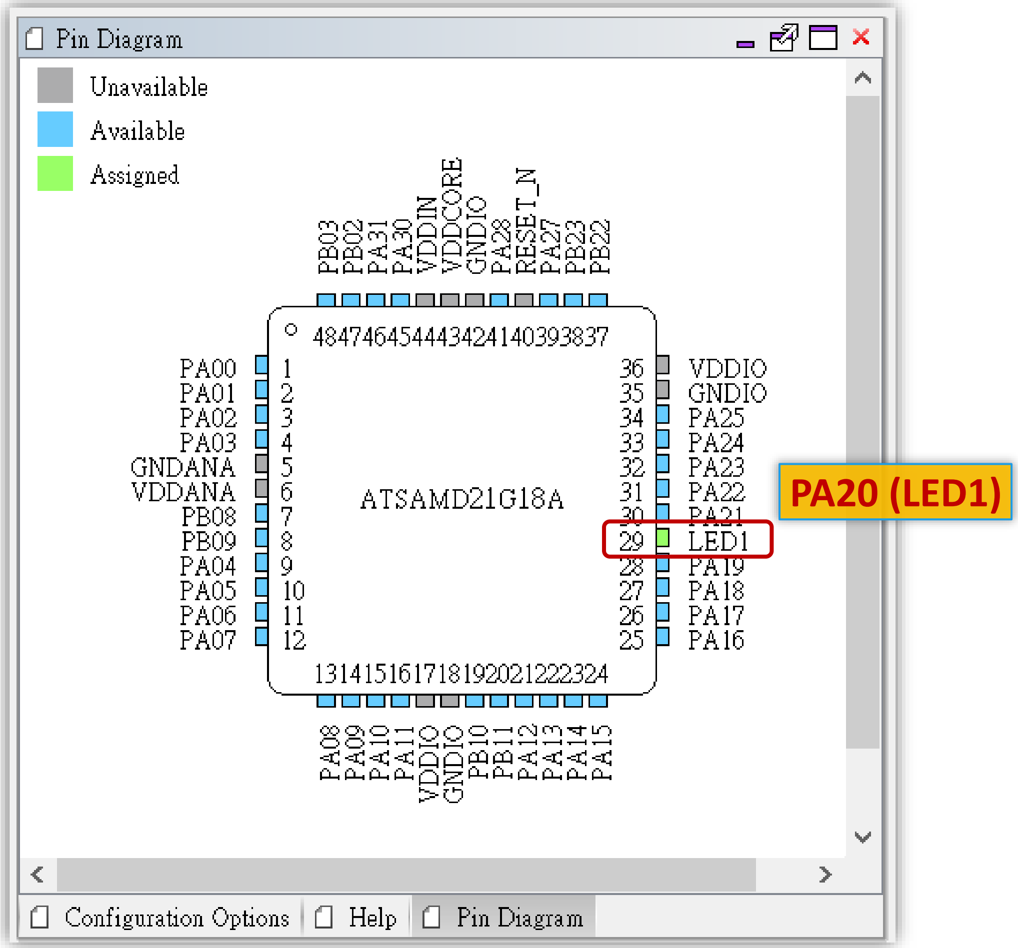




Lab1 PORT Output

Step 5

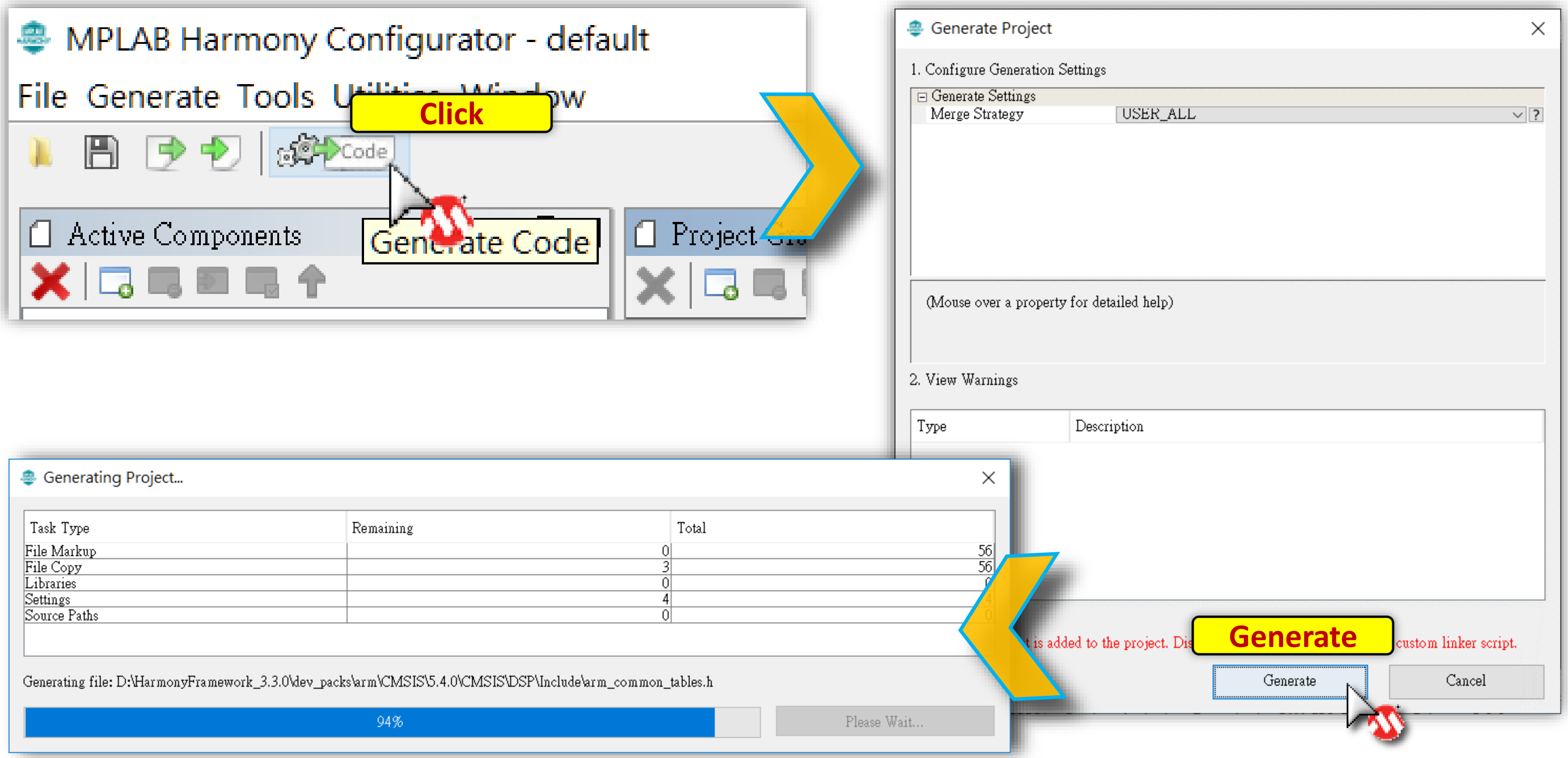
- **Pin Diagram. (Hardware design reference)**



Lab1 PORT Output

Step 6

- Click  to Generate Code and save changes to MHC configure file.



MHC 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

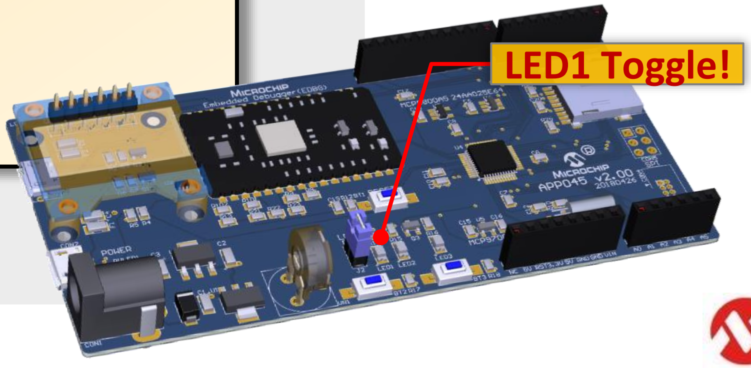
a Modify code segment to your main loop

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

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

PORT initialization already done here

Comment out this line

b Program firmware to target board then observe result.

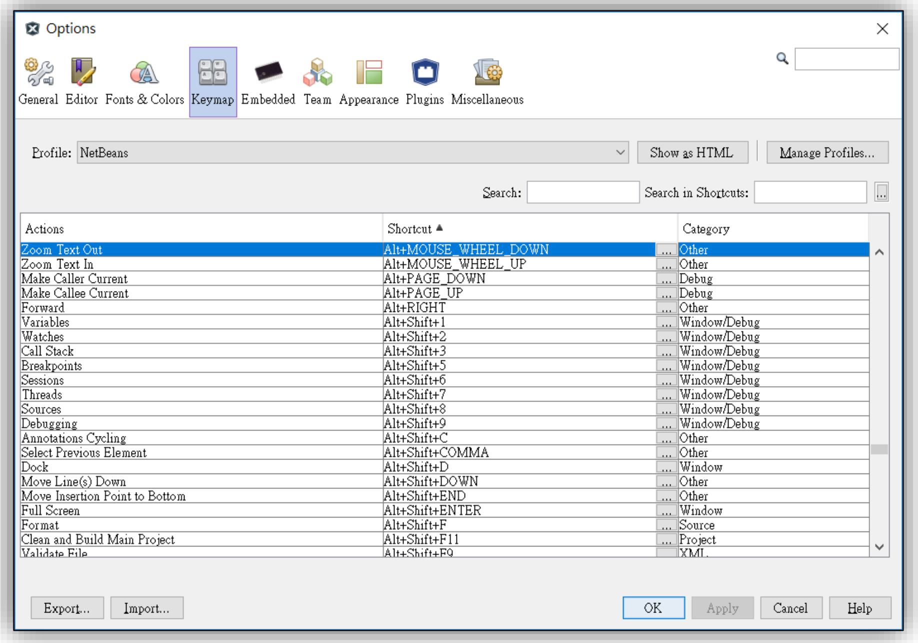


MICROCHIP

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

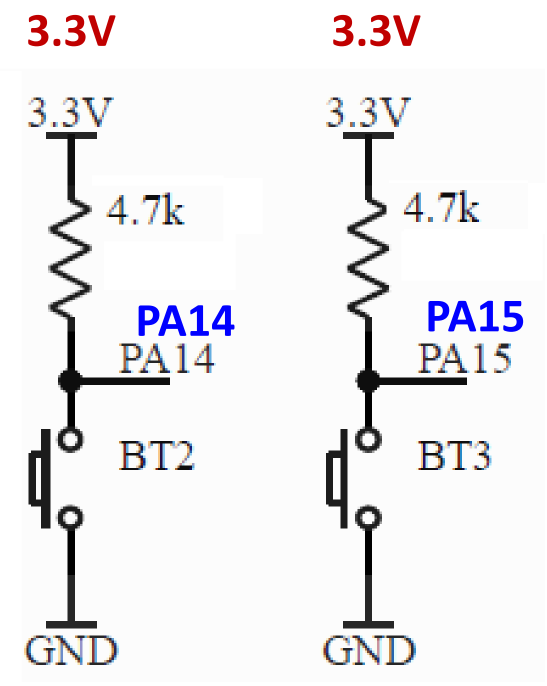


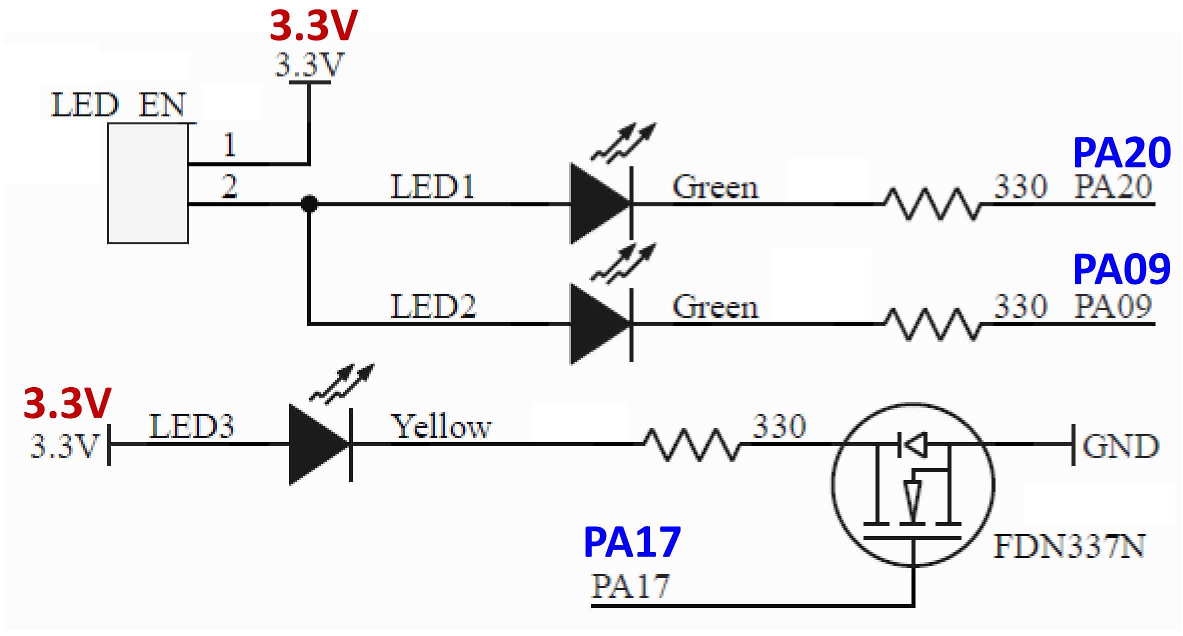
Lab2 PORT Input and Output

- **Base On Lab1, Try to add LED2(PA09) continues toggle (Period around 500ms ~ 1S).**
- **Get BT2(PA14) status then use to control LED3(PA17).**
  Pressed -> LED3 Light
  Released -> LED3 Dark

- **Let’s go!**

Check Schematic



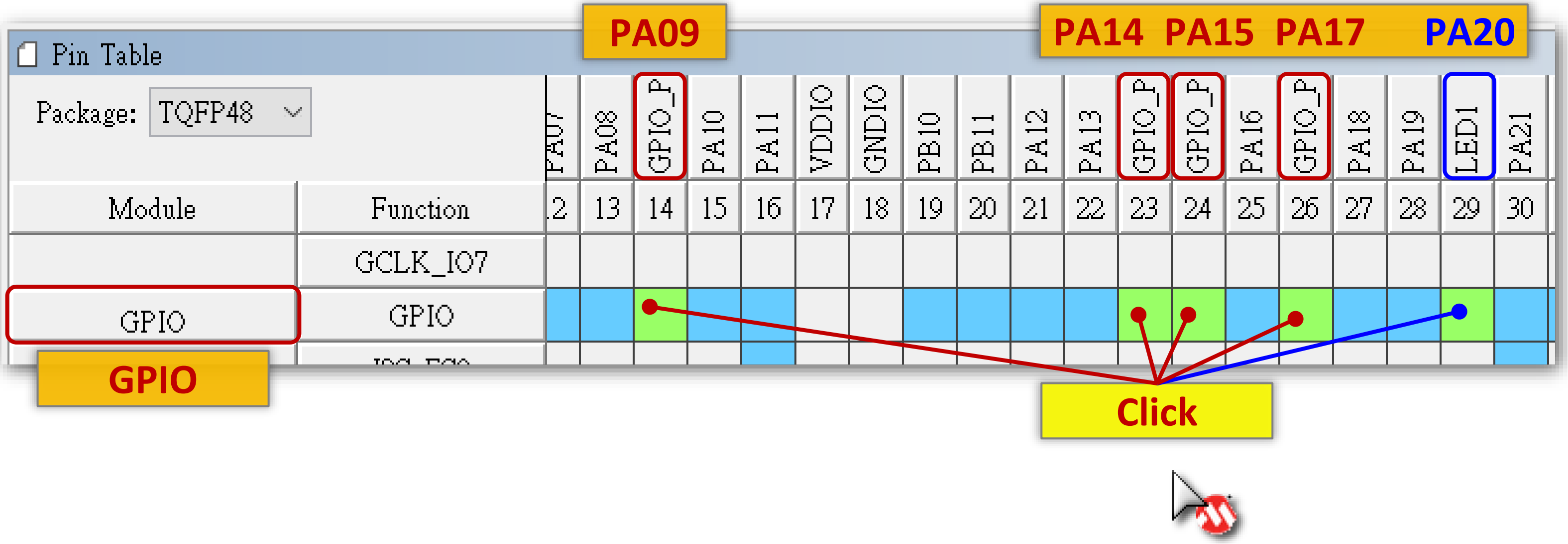


LED1	PA20
LED2	PA09
LED3	PA17
BT2	PA14
BT3	PA15

Lab2 PORT Input and Output

Step 1

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



Lab2 PORT Input and Output

Step 2

- **Pin Setting. (Pin Function Setting)**

Pin Settings

Order: Pins | Table View | ☑ Easy View

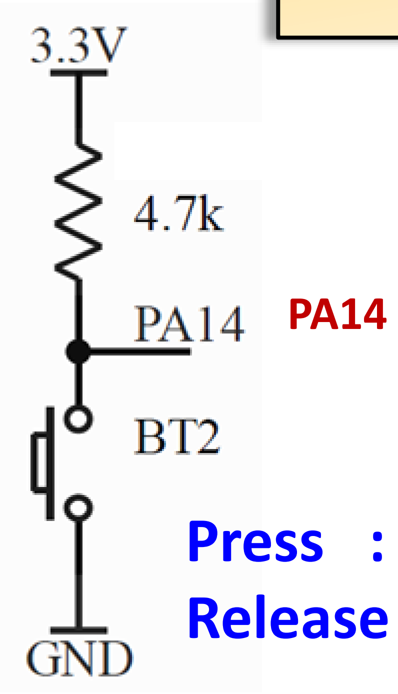
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down
PA09	PA09	LED2	GPIO LED2	Digital	Out	Low	Out	Low
15	PA10		Available	Digital	High I...	Low	☐	☐
16	PA11		Available	Digital	High I...	Low	☐	☐
17	VDDIO			Digital	High I...	Low	☐	☐
18	GNDIO			Digital	High I...	Low	☐	☐
19	PB10		Available	Digital	High I...	Low	☐	☐
20	PB11		Available	Digital	High I...	Low	☐	☐
21	PA12		Available	Digital	High I...	Low	☐	☐
22	PA13		Available	Digital	High I...	Low	☐	☐
PA14	PA14	BT2	GPIO BT2	Digital	In	Low	In	☐
PA15	PA15	BT3	GPIO BT3	Digital	In	Low	In	☐
25	PA16		Available	Digital	High I...	Low	☐	☐
PA17	PA17	LED3	GPIO LED3	Digital	Out	Low	Out	Low
27	PA18		Available	Digital	High I...	Low	☐	☐
28	PA19		Available	Digital	High I...	Low	☐	☐
PA20	PA20	LED1	GPIO LED1	Digital	Out	High	Out	High

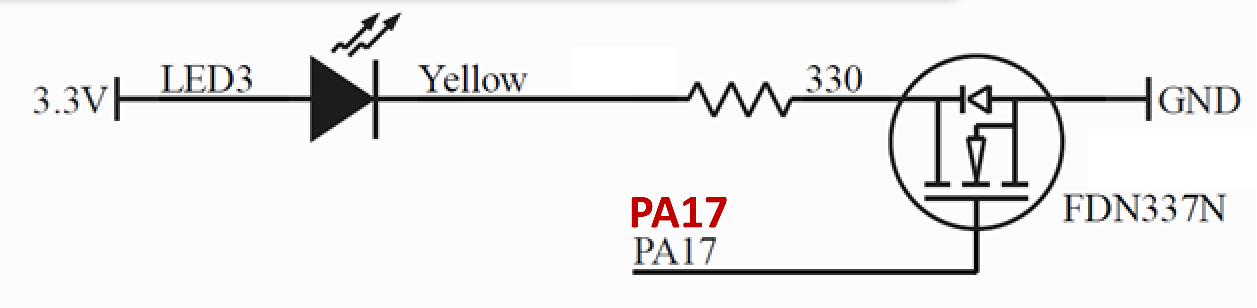
Lab2 PORT Input and Output

Schematic Describe

- Get BT2(PA14) status then use to control LED3(PA17).
  Pressed -> LED3 Light , Released -> LED3 Dark

```
if ( BT2_Get() ) LED3_Clear();
else             LED3_Set();
```





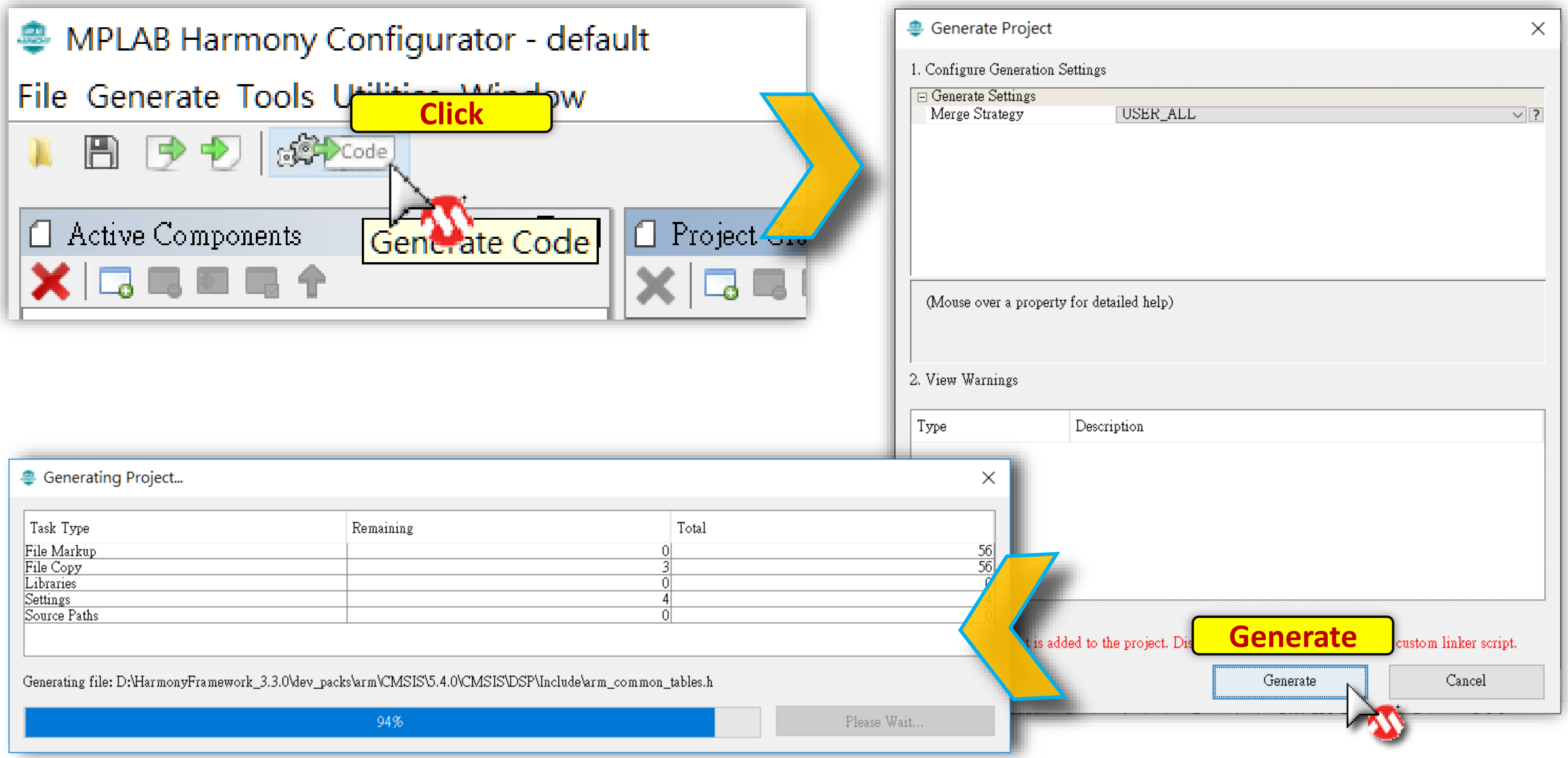
LED3 Light : PA17 = 1
LED3 Dark : PA17 = 0

Press : PA14 = 0
Release : PA14 = 1

Lab2 PORT Input and Output

Step 3

- Click  to Generate Code and save changes to MHC configure file.



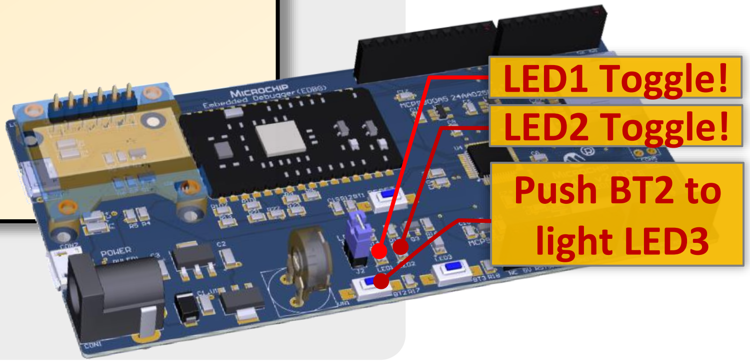
Lab2 PORT Input and Output

Step 4

a Add code segment to your main loop

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

   while(1)
   {
      for ( i = 0 ; i < 2000000 ; i++ )
      {}

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

      // TODO 2.02
      if ( BT2_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(1)
{
   for ( i = 0; i < 2000000 ; i++ )
   {}

   LEDs_Toggle();

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

```
while(1)
{
   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(1)
   {
      if ( i++ > 2000000 )
      {
         i = 0;

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

      // TODO 2.02
      if ( BT2_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

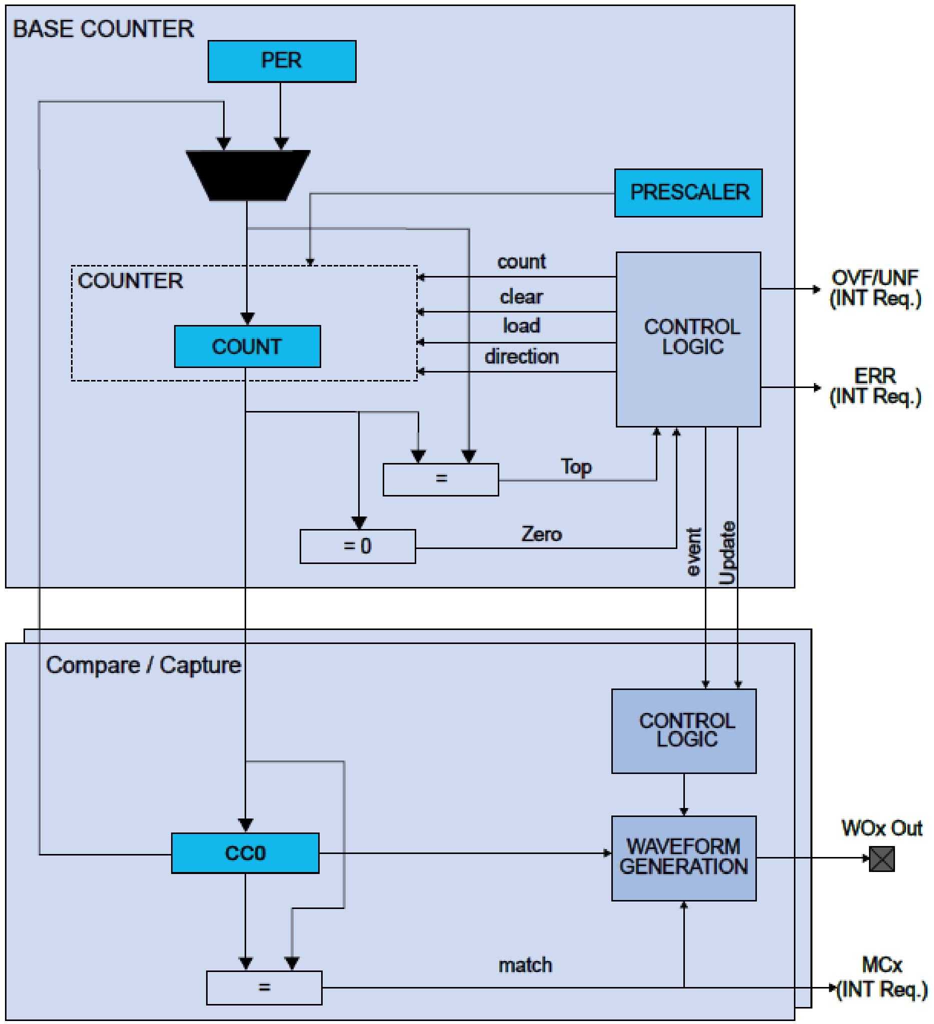
- The SAMD21 Timer/Counter, TC module consists 3 function Counter, Compare and Capture.

- 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**

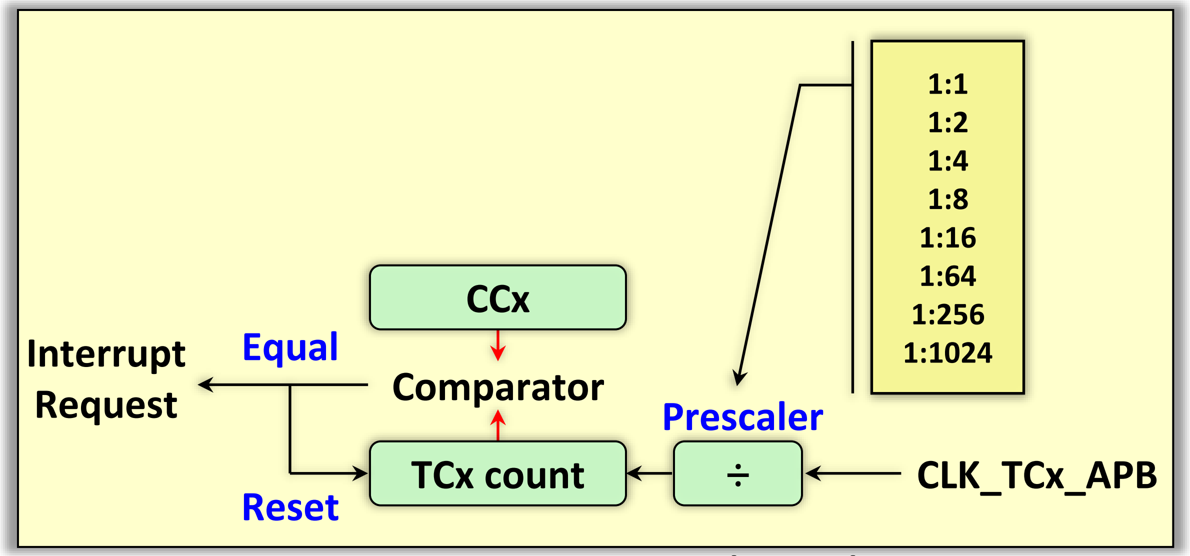


SAMD21 Timer/Counter

- 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.



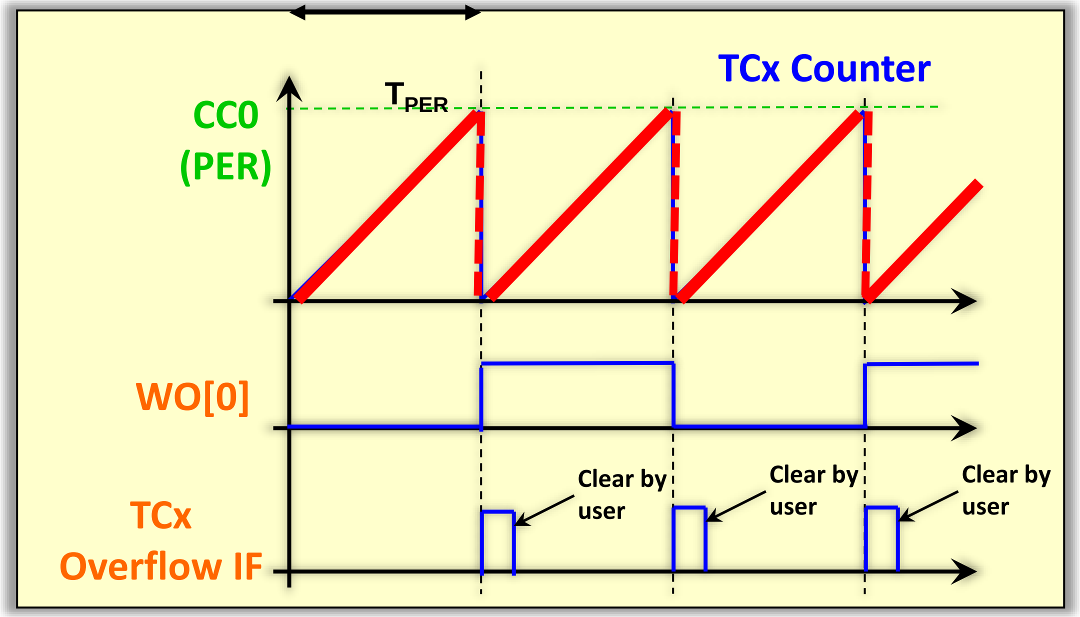
Match Frequency Generation (MFRQ), 16Bit, Up count

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

Match Frequency Generation (MFRQ) Mode

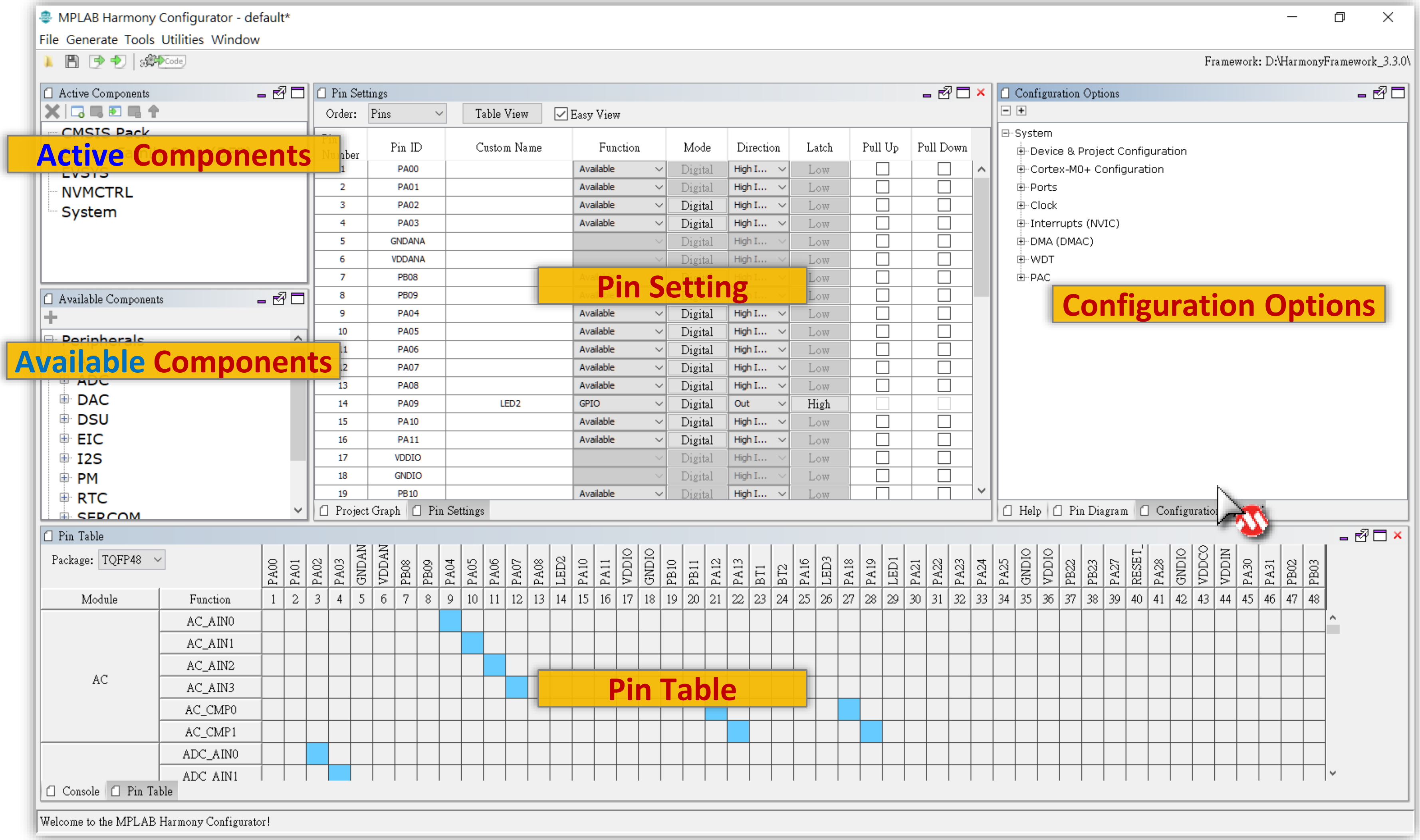
- For Match Frequency Generation, the period time (T_{PER}) is controlled by the CC0 register instead of PER.
  WO[0] toggles on each update condition.



Rule
TCx Counter == CC0 -> WO[x] Toggle

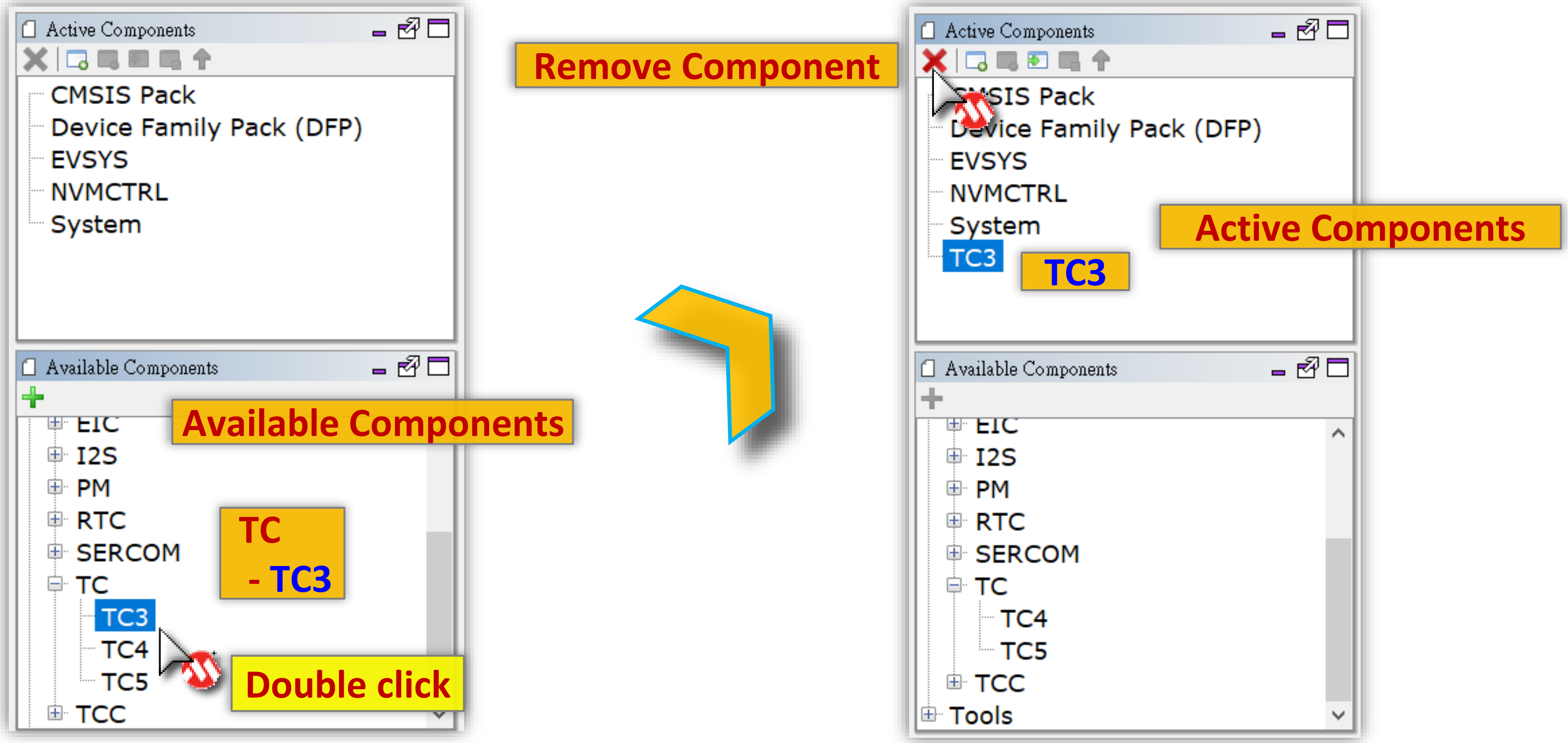
Docking MHC windows

- **Return to Harmony and docking windows as below.**

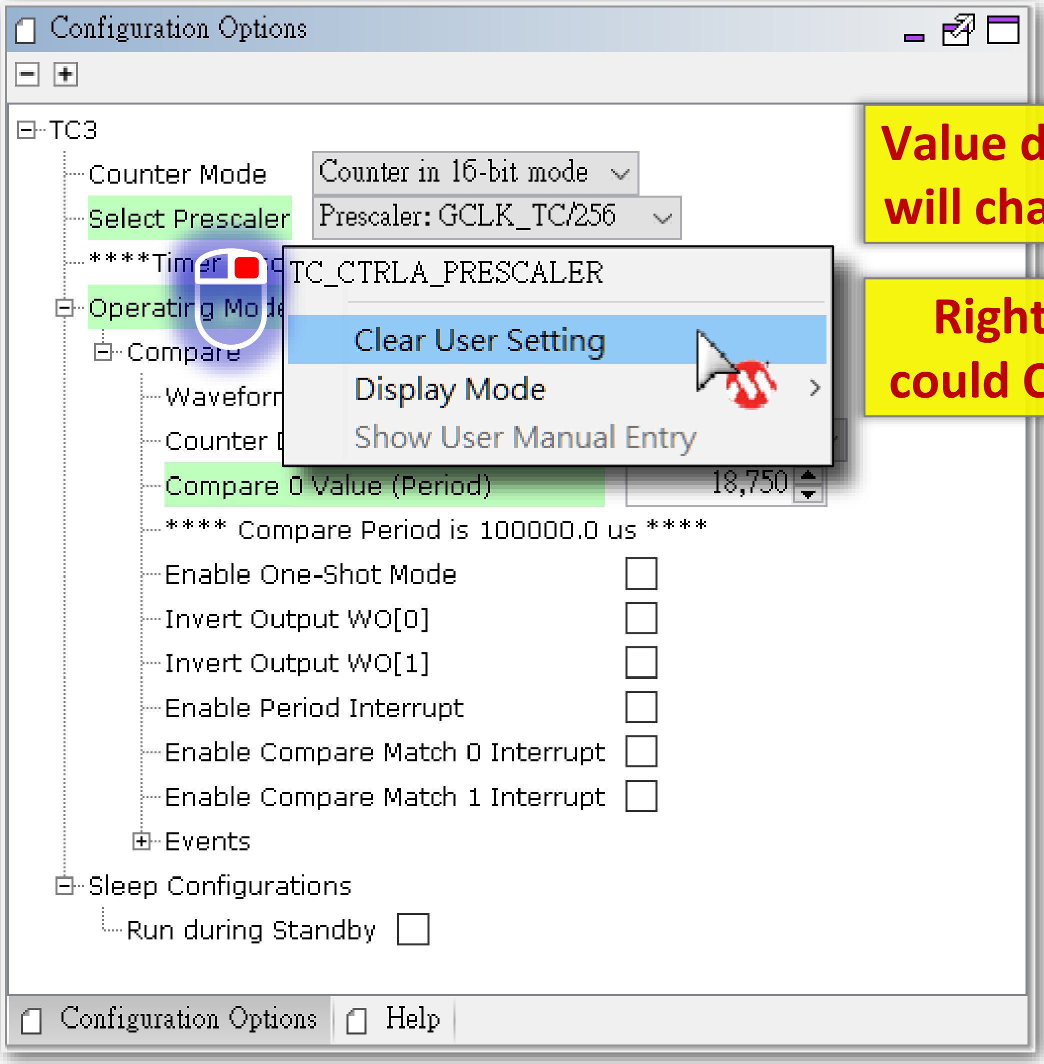


Add TC Function using MHC

- Find TC3 component in Available Components window.
- Double click TC3 to add to Active Components window.
- Click could remove component from Active 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_CompareInitialize( void );
void TC3_CompareStart( void );
void TC3_CompareStop( void );
uint32_t TC3_CompareFrequencyGet( void );
void TC3_Compare16bitPeriodSet( uint16_t period );
uint16_t TC3_Compare16bitPeriodGet( void );
uint16_t TC3_Compare16bitCounterGet( void );
void TC3_Compare16bitCounterSet( uint16_t count );
TC_COMPARE_STATUS TC3_CompareStatusGet( 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 MFRQ Mode Overflow Polling Example

Bit 0 – OVF Overflow Interrupt Flag

Writing a '0' to this bit has no effect.

Writing a '1' to this bit clears the Overflow interrupt flag.

```
...
TC3_CompareStart();

while(1)
{
   if ( TC3_REGS->COUNT16.TC_INTFLAG & TC_COMPARE_STATUS_OVERFLOW )
   {
      TC3_REGS->COUNT16.TC_INTFLAG = TC_COMPARE_STATUS_OVERFLOW;
      LED1_Toggle();
   }
}
```

OR

```
...
TC3_CompareStart();

while(1)
{
   if ( TC3_CompareStatusGet() & TC_COMPARE_STATUS_OVERFLOW )
   {
      LED1_Toggle();
   }
}
```

TC_INTFLAG will get and clear in function

Lab3 TC MFRQ Polling

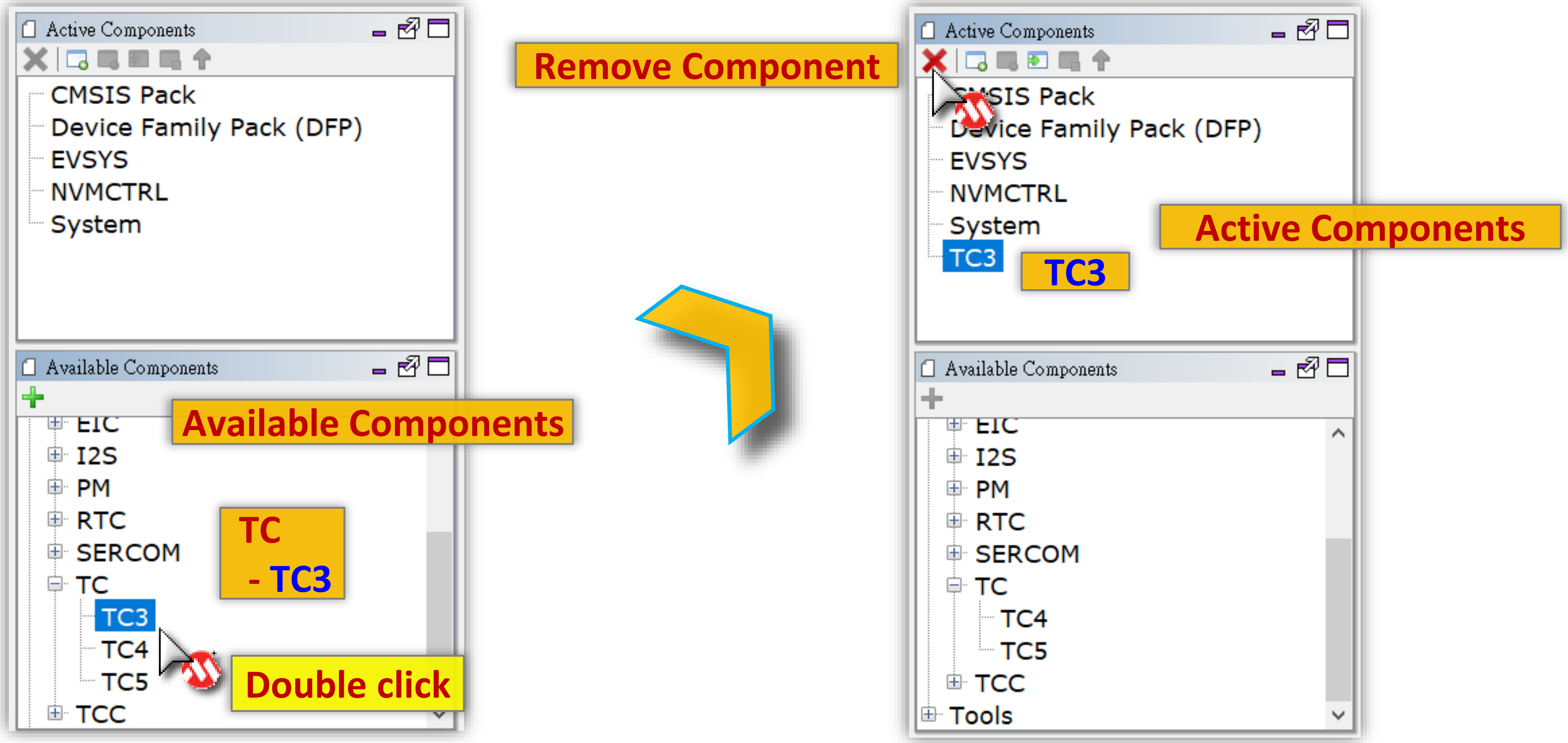
- **Try to use TC3 to replace software delay to control LED1(PA20) continues toggle.**
- **The TC3 setup to MFRQ, 16 bit mode,**
  The Timer period is 100mS.
- **Enable waveform output of TC3 on pin PA18 and PA19.**

- **Let's go!**

Lab3 TC MFRQ Polling

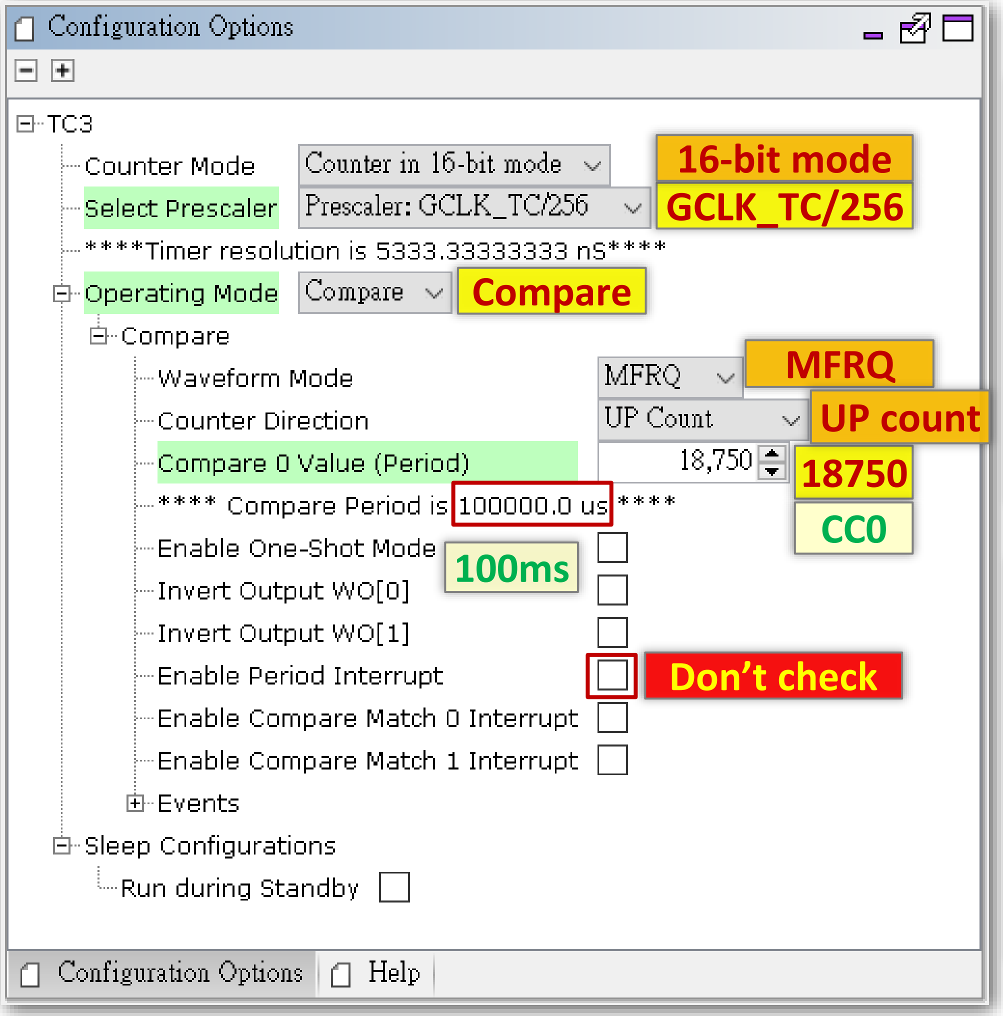
Step 0

- Find **TC3** component in Available Components window.
- Double click **TC3** to add to Active Components window.
- Click could remove component from Active window.



Lab3 TC MFRQ Polling

Step 1



System Clock = 48MHz

One second counts of system clock :

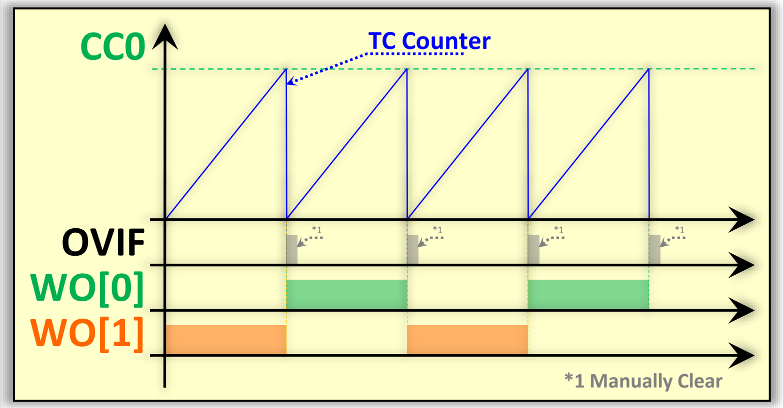
48,000,000 counts

One second counts after Prescaler :

48000000 / 256 187,500 counts

100ms counts for LAB3 target :

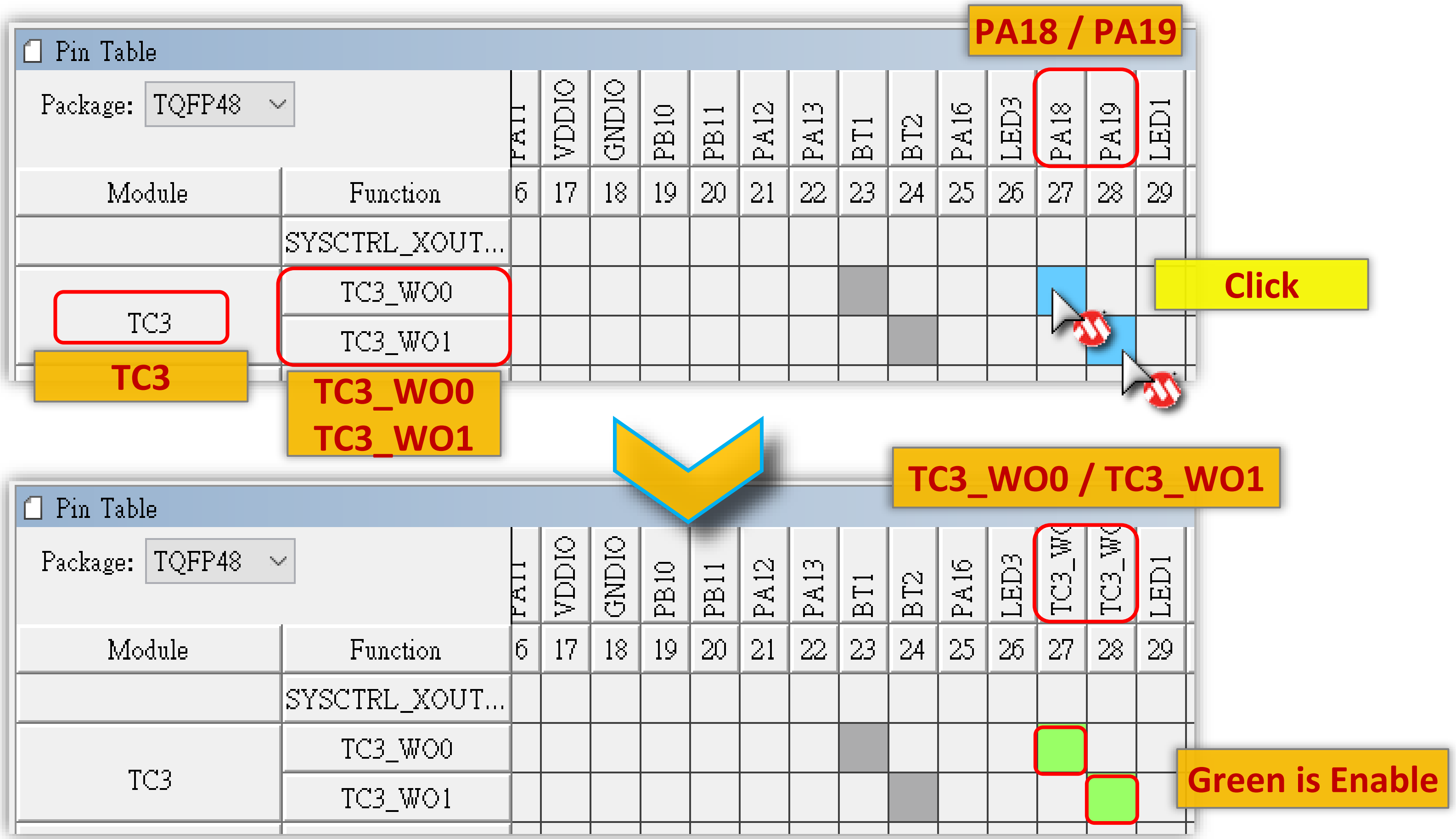
187500 / 10 18,750 counts



Lab3 TC MFRQ Polling

Step 2

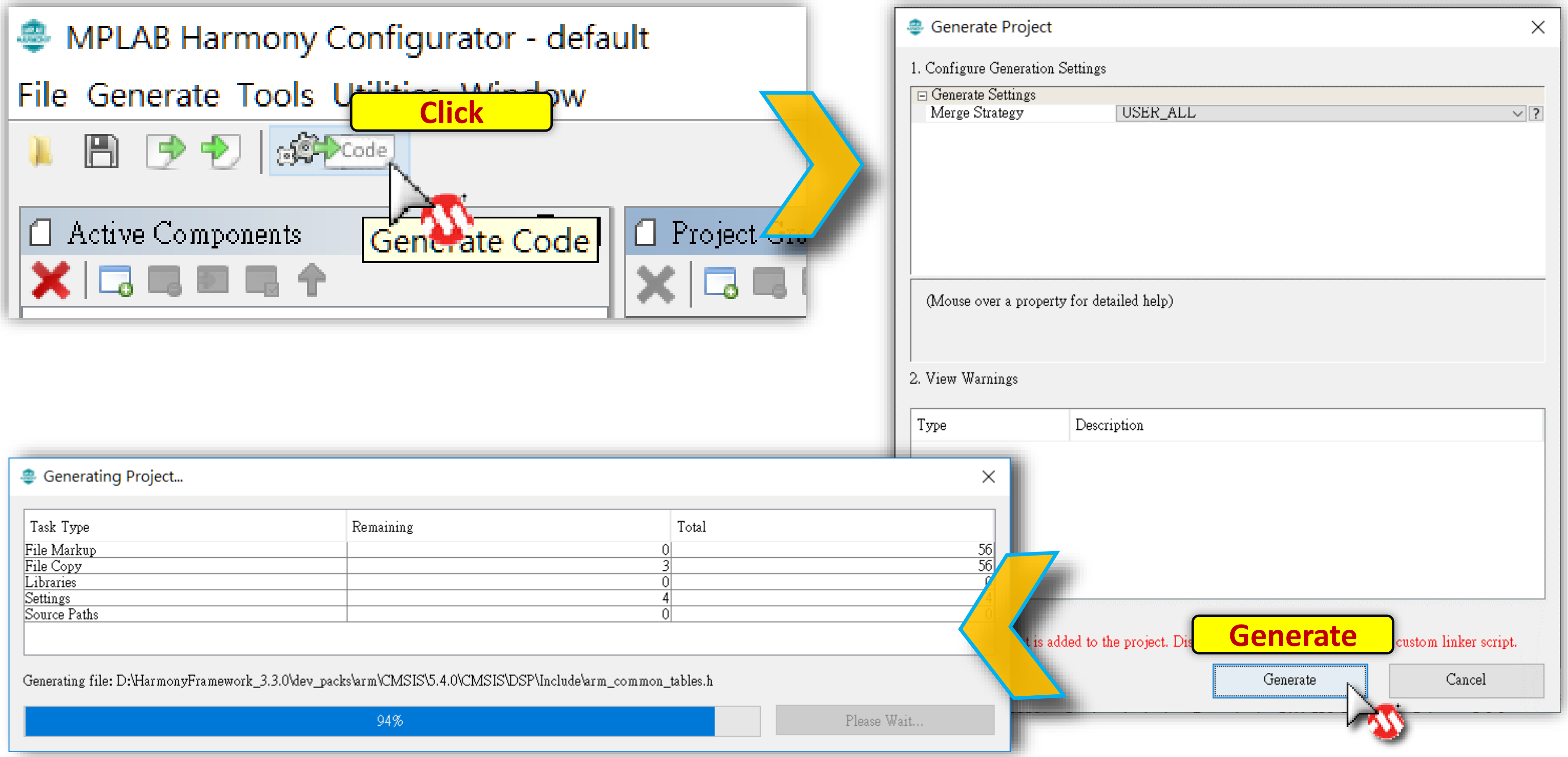
- **TC support waveform output through WO[x] pin depend on different mode.**



Lab3 TC MFRQ Polling

Step 3

- Click  to Generate Code and save changes to MHC configure file.



Lab3 TC MFRQ Polling

Step 4

a Add code segment to your main loop

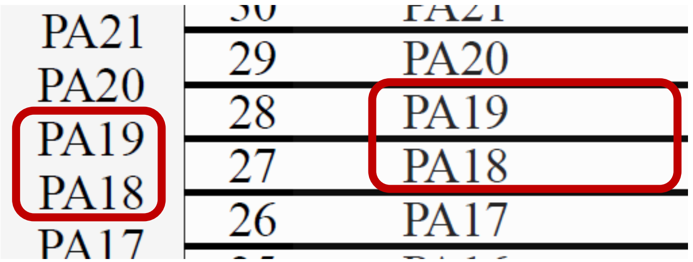
```
…
int main (void)
{
   SYS_Initialize ( NULL );

   // TODO 3.01
   TC3_CompareStart();

   while(1)
   {
      // TODO 3.02
      if ( TC3_CompareStatusGet() &
               TC_COMPARE_STATUS_OVERFLOW )
      {
            LED1_Toggle();
      }
      …
   }
}
```

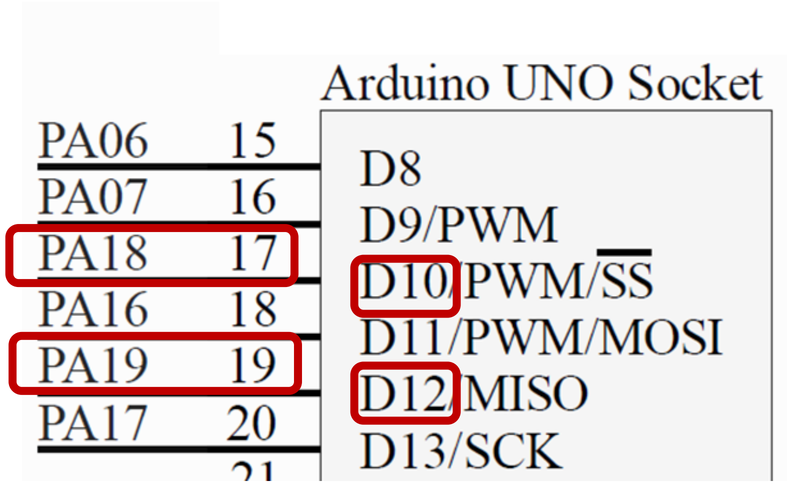
Period Control

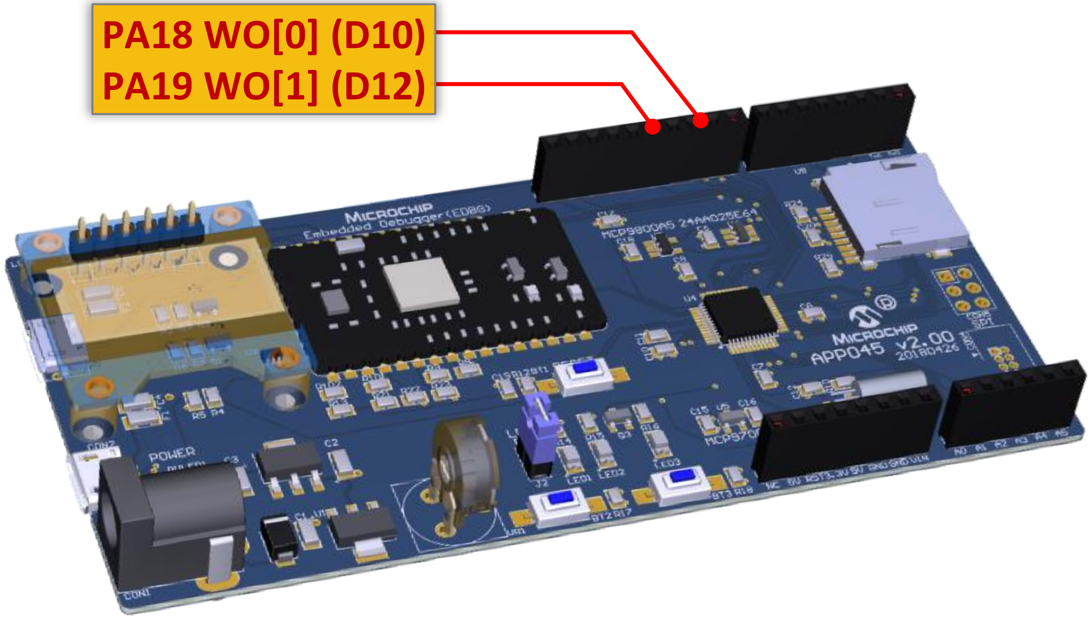
b Program firmware to target board then observe result.

Lab3 TC MFRQ Polling

Waveform Output

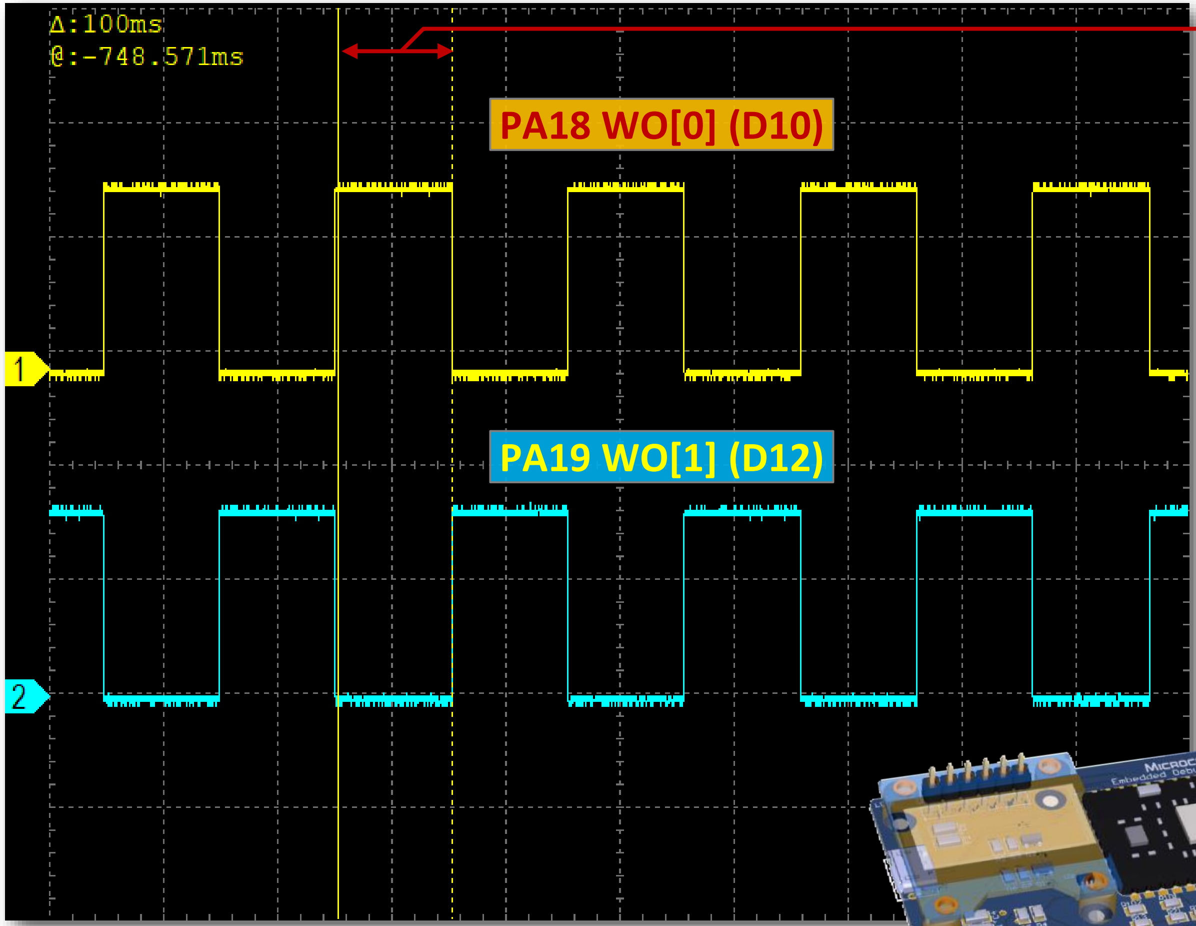


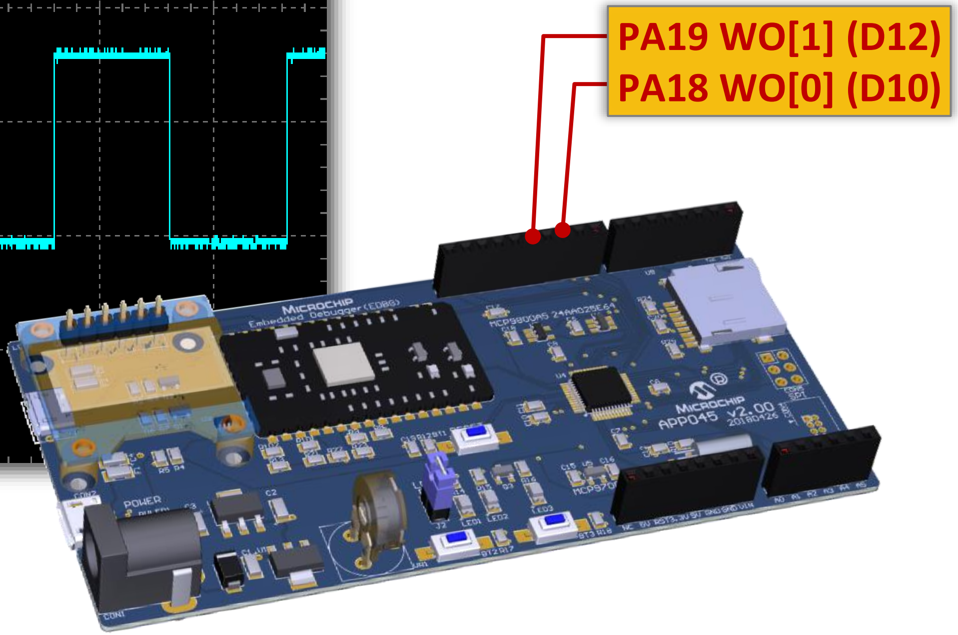




Lab3 TC MFRQ Polling

Waveform Output





Lab4 TCs MFRQ Polling

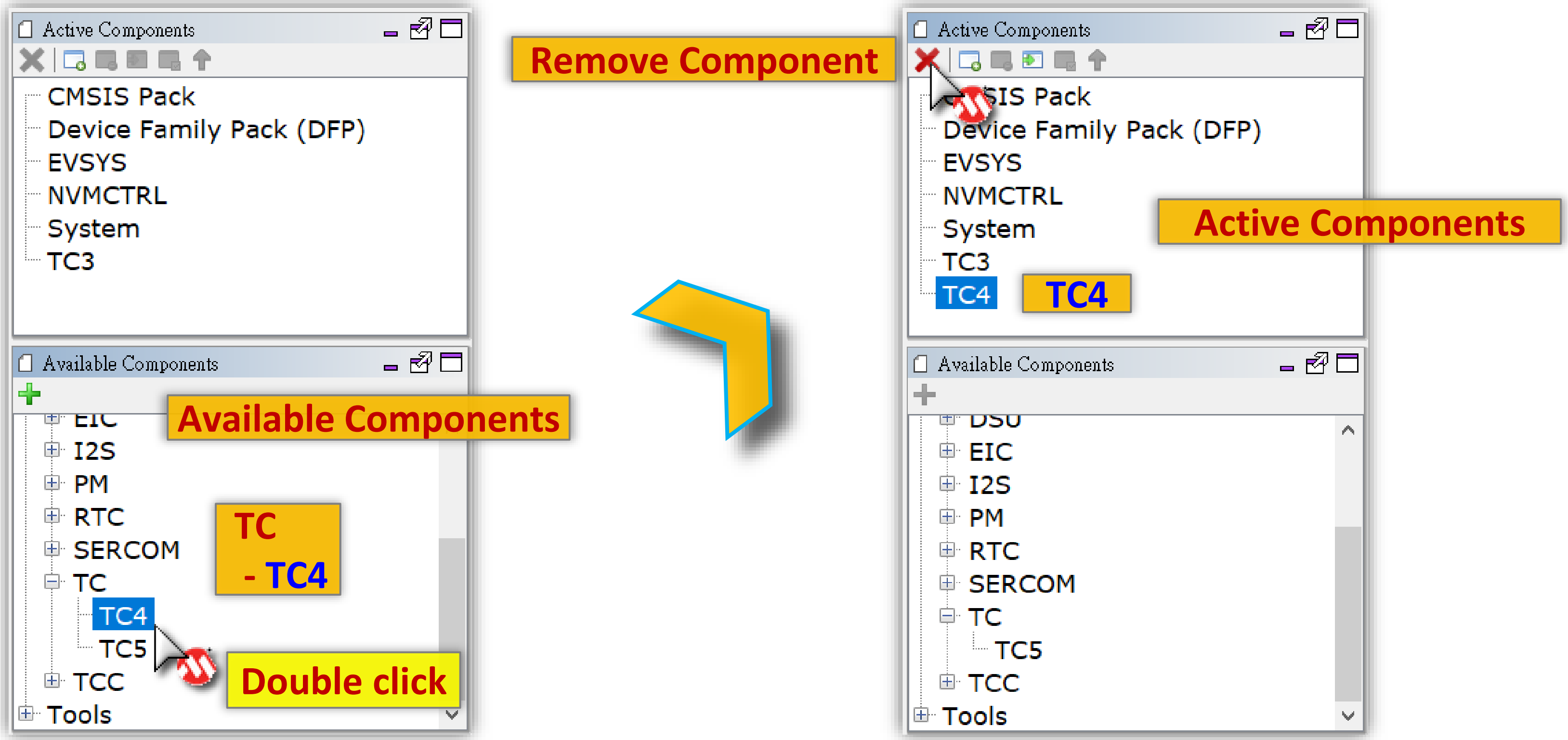
- Try to add TC4 to replace software delay to control LED2(PA09) continues toggle.
- All setting same as TC3 except period.
- The TC4 set timer period to 50ms.
- Enable waveform output of TC4 on pin PA22 and PA23, then you can measure the waveform signal from PA22 and PA23 also.

- **Let's go!**

Lab4 TCs MFRQ Polling

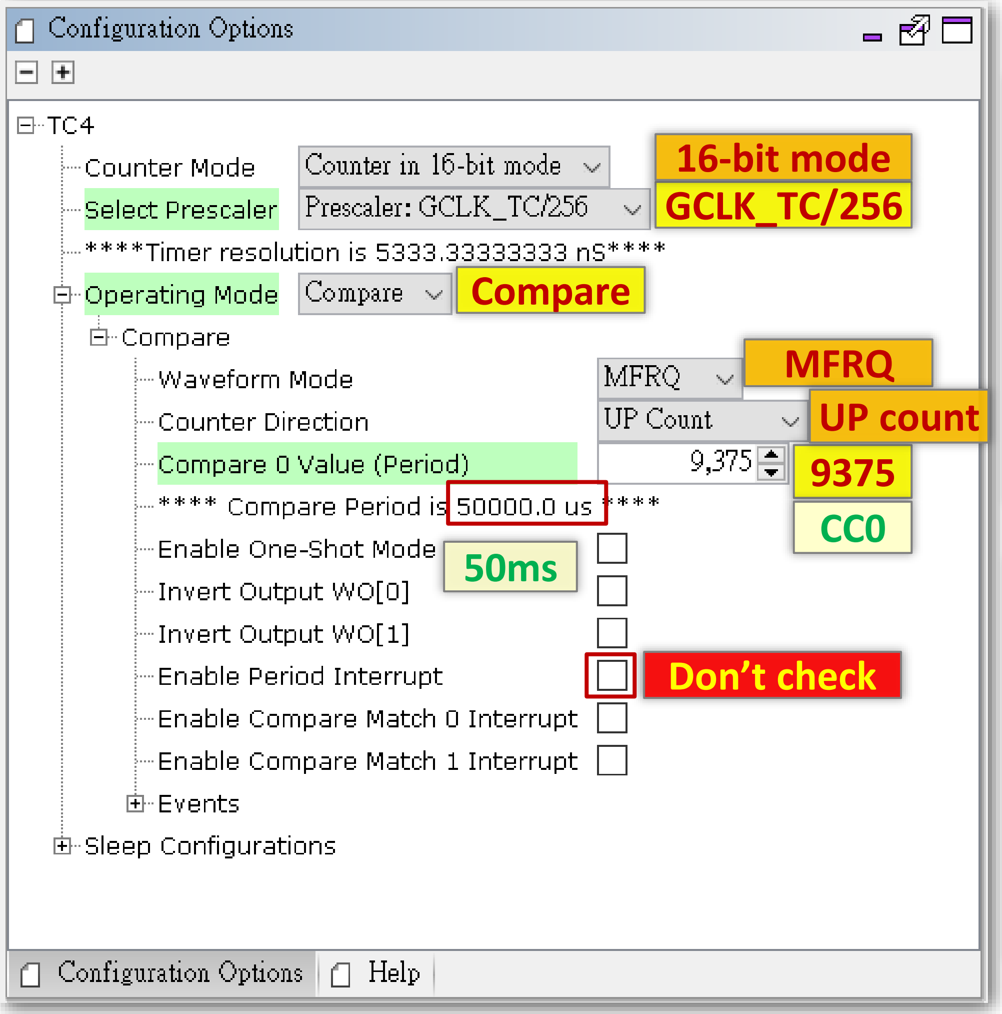
Step 1

- Find **TC4** component in Available Components window.
- Double click **TC4** to add to Active Components window.
- Click could remove component from Active window.

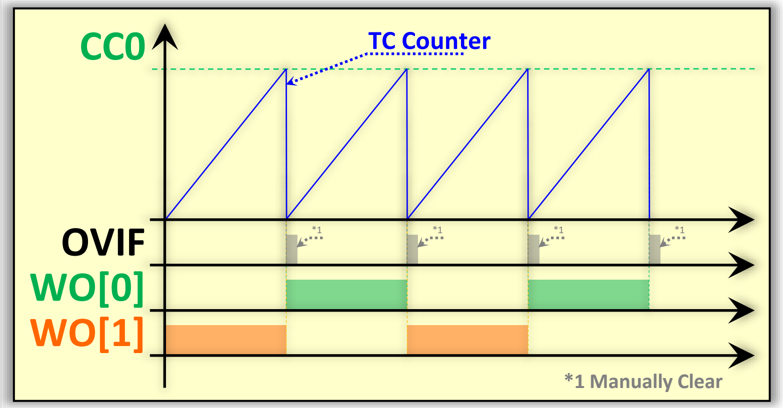


Lab4 TCs MFRQ Polling

Step 2



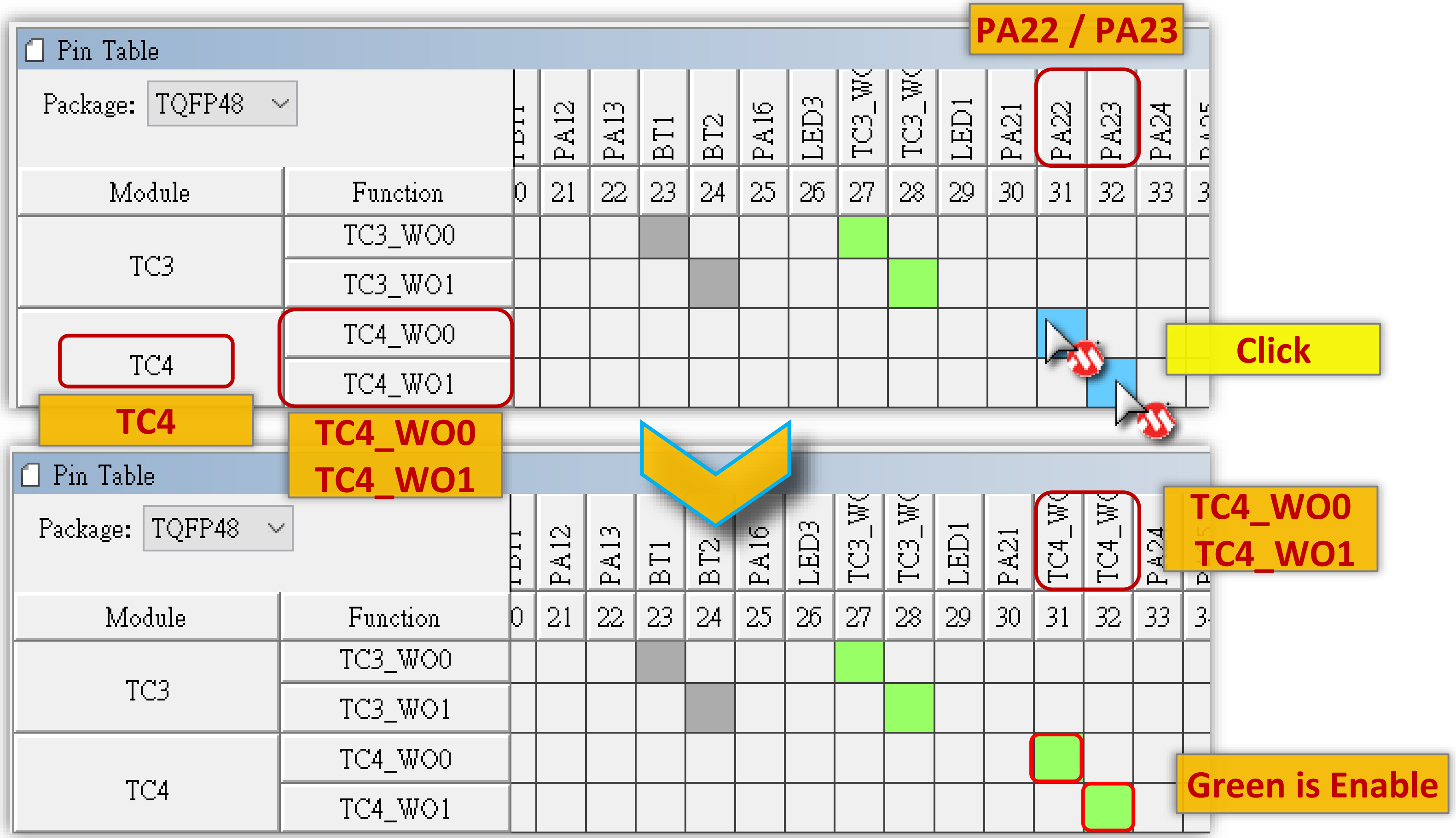
System Clock = 48MHz
One second counts of system clock :
48,000,000 counts
One second counts after Prescaler :
48000000 / 256 187,500 counts
50ms counts for LAB4 target :
187500 / 20 9,375 counts



Lab4 TCs MFRQ Polling

Step 3

- **TC support waveform output through WO[x] pin depend on different mode.**

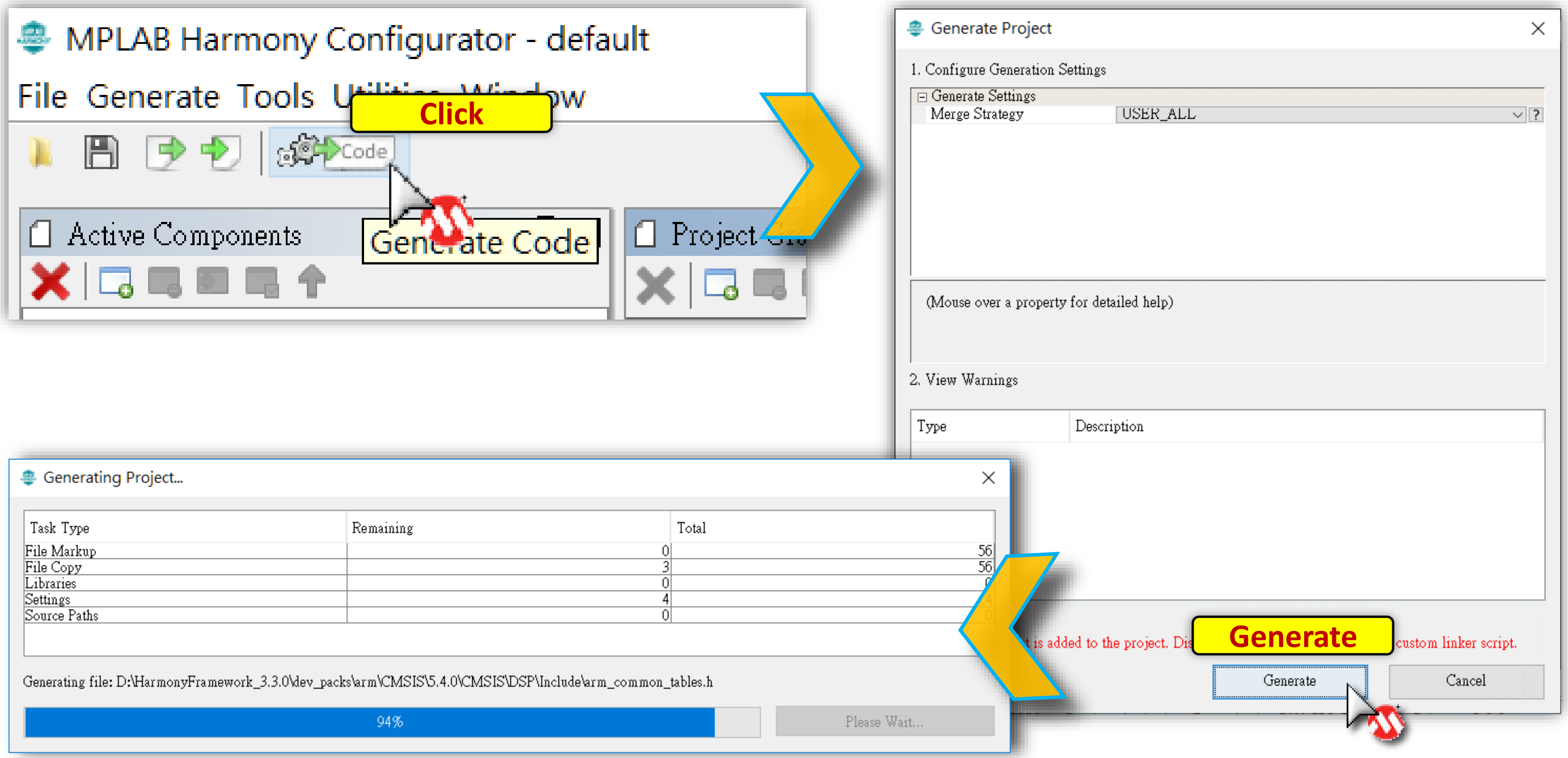


Lab4 TCs MFRQ Polling

Step 4

- Click 

 to Generate Code and save changes to MHC configure file.



Lab4 TCs MFRQ Polling

Step 5

a Add code segment to your main loop

```
int main (void)
{
   SYS_Initialize ( NULL );
   …
   TC3_CompareStart();
   // TODO 4.01
   TC4_CompareStart();

   while(1)
   { …
      // TODO 4.02
      if ( TC4_CompareStatusGet() &
              TC_COMPARE_STATUS_OVERFLOW )
      {
          LED2_Toggle();
      }
      …
   }
}
```

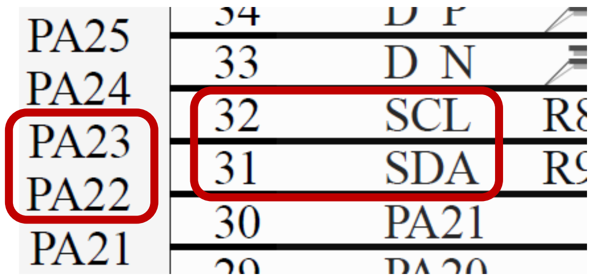
Period Control

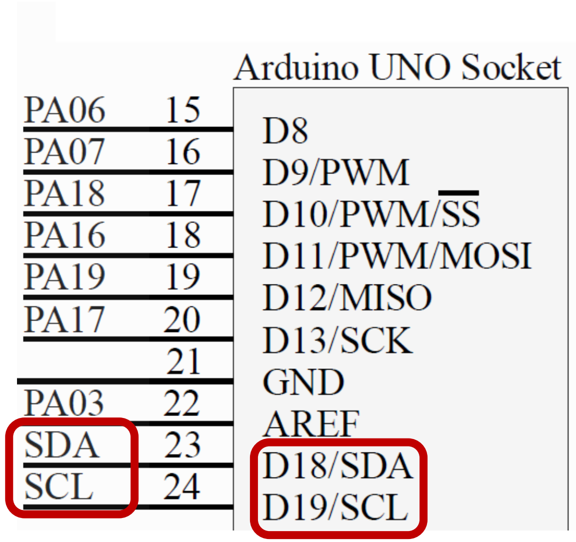
b Program firmware to target board then observe result.

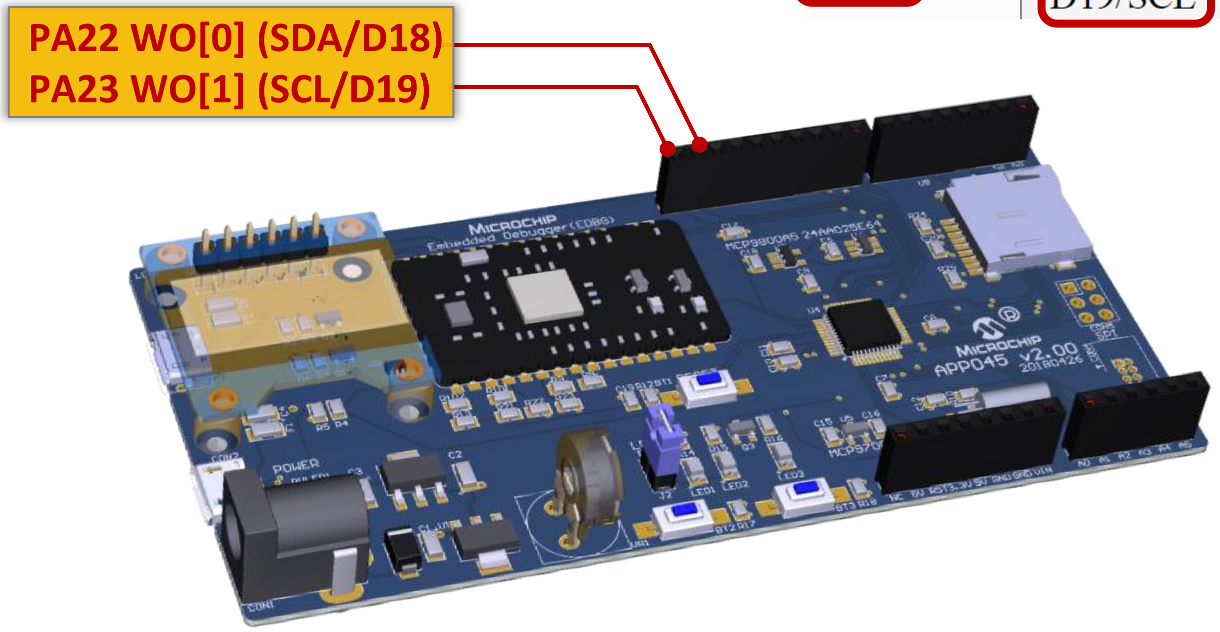
MICROCHIP

Lab4 TCs MFRQ Polling

Waveform Output

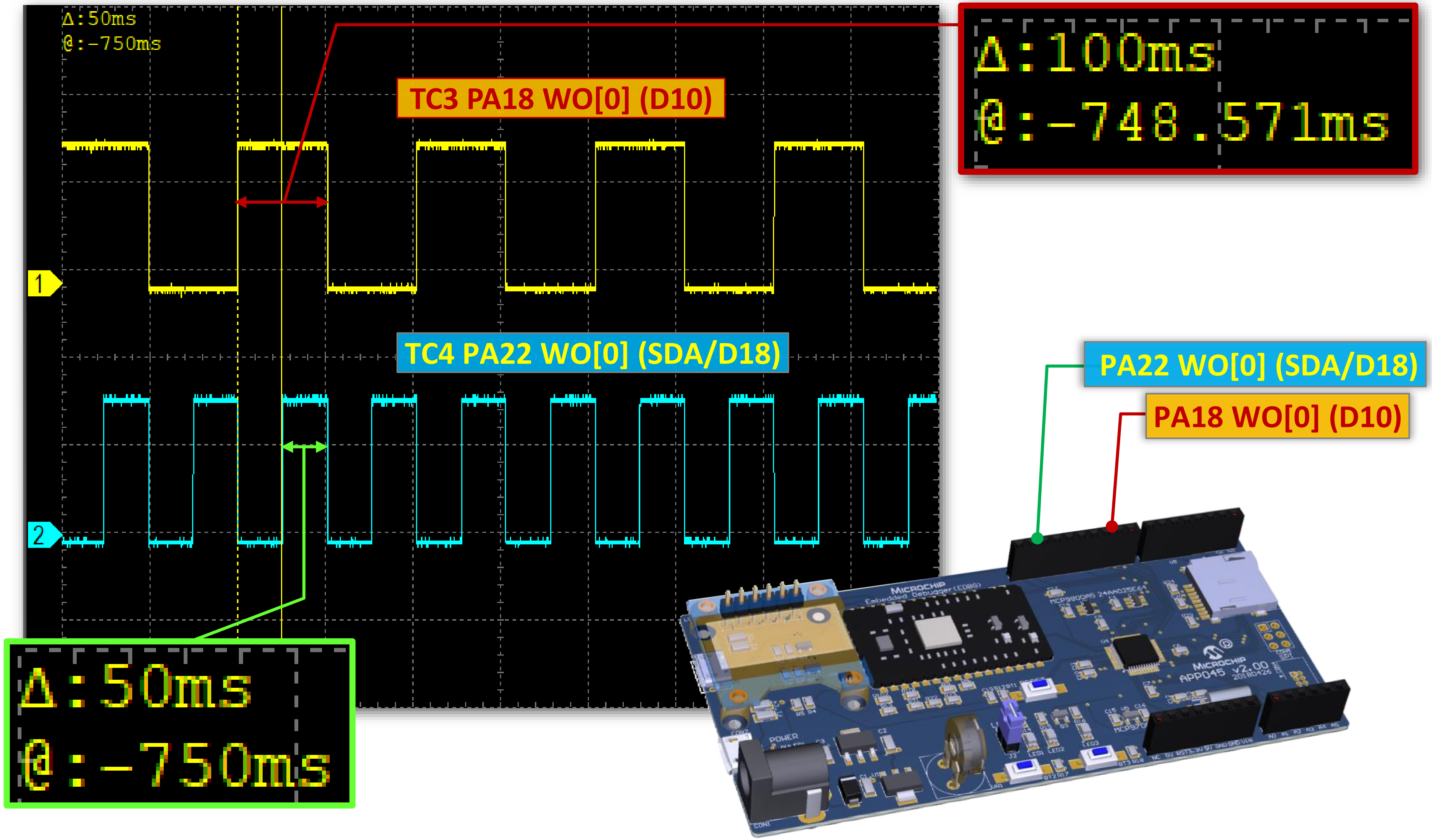






Lab4 TCs MFRQ Polling

Waveform Output



NVIC Architecture

Nested Vector Interrupt Controller Architecture

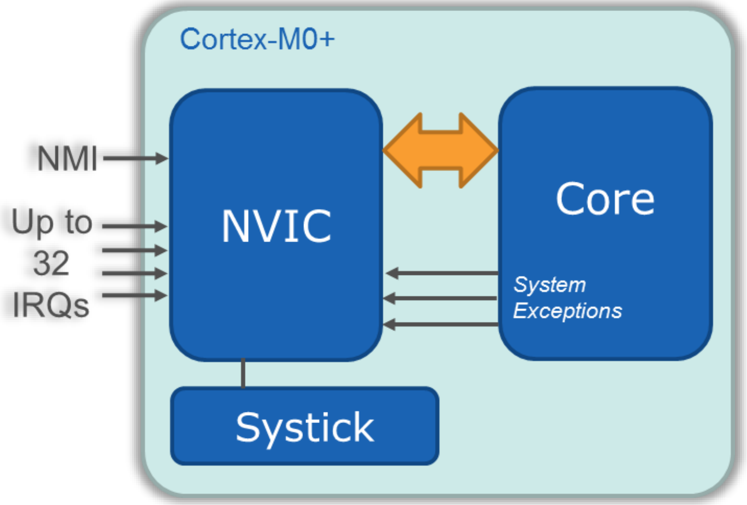
Nested Vector Interrupt Controller

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.

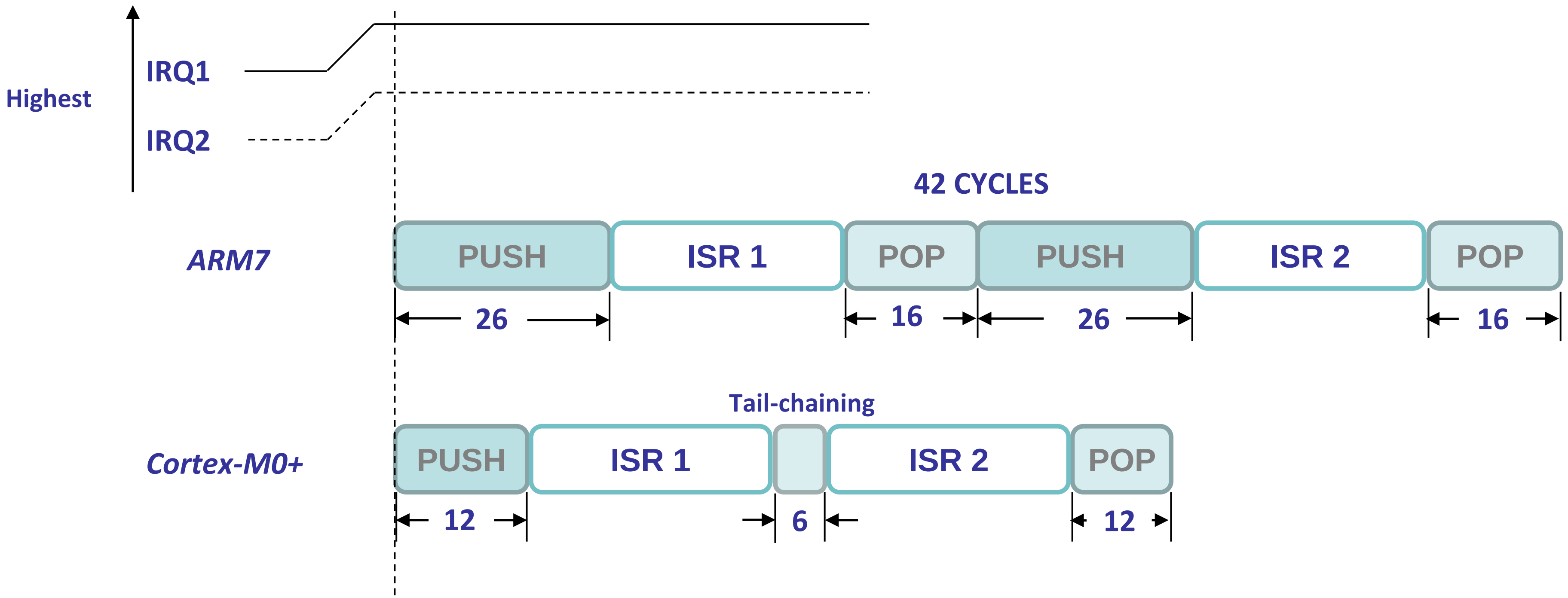
- Each interrupt lines has connected to one peripheral instance and have one or more interrupt flags.



- An interrupt request is generated from the peripheral when the interrupt flag has been set and the corresponding interrupt is enabled.

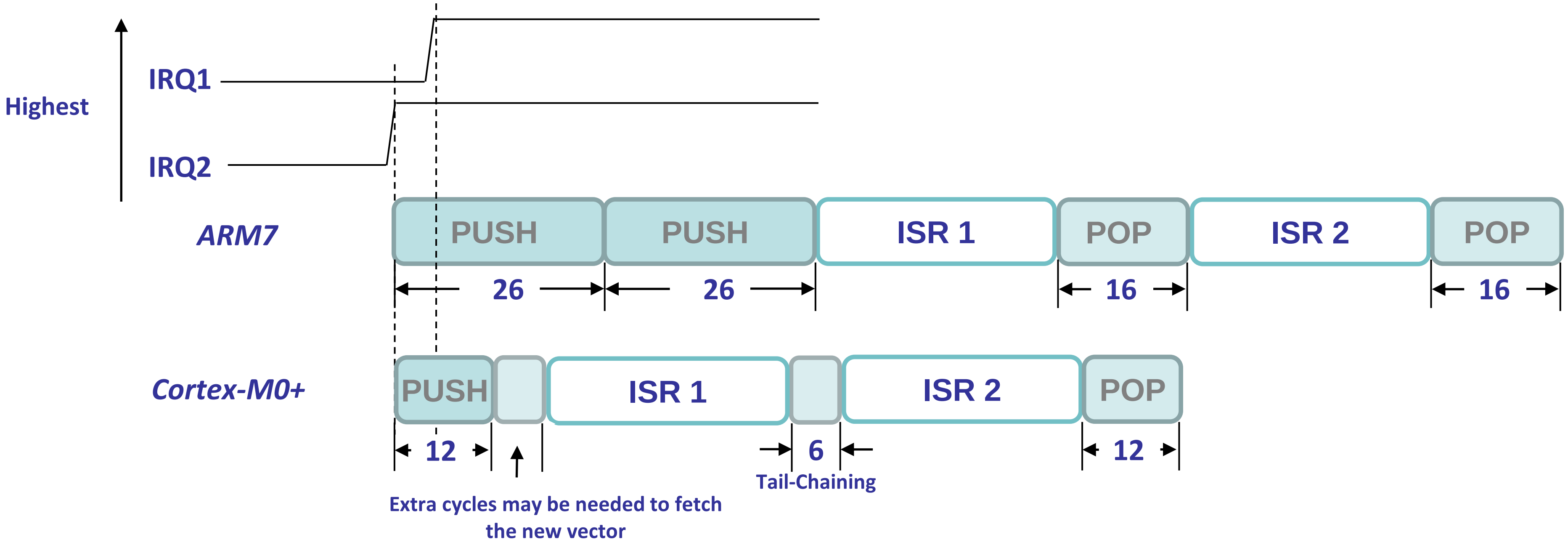
Tail Chaining in the NVIC

- **Tail-chaining is back-to-back processing of exceptions. it's can reduce unnecessary POP and PUSH. when exiting one ISR and entering another because this has no effect on the stack contents.**



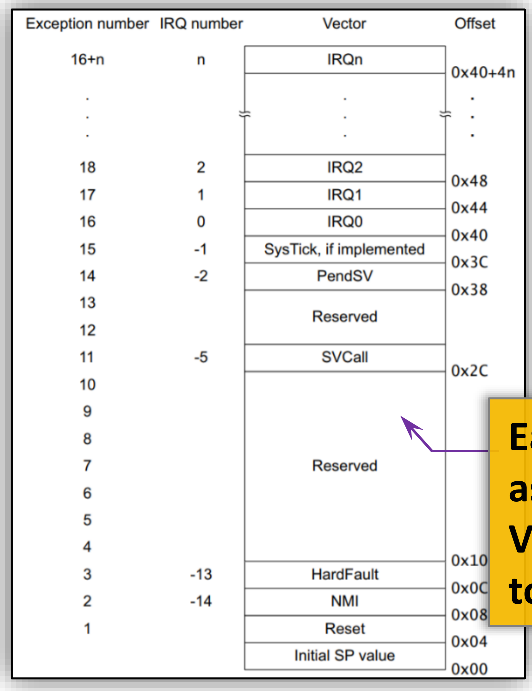
Late Arriving in the NVIC

- **Late-arriving speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception.**



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).**



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

TC3 (18)

TC4 (19)

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_CompareInterruptHandler,
    .pfnTC4_Handler               = ( void * ) TC4_CompareInterruptHandler,
    …
    .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_CompareInterruptHandler( void )
{
    TC_COMPARE_STATUS status;

    status = TC3_REGS->COUNT16.TC_INTFLAG;

    /* clear period interrupt */
    TC3_REGS->COUNT16.TC_INTFLAG = TC_INTFLAG_OVF_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

```
TCn_CompareCallbackRegister( TCn_Overflow, (uintptr_t )NULL );
TCn_CompareStart();
```

Hook (Register)

```
void TCn_CompareInterruptHandler( void )
{
   TCn_COMPARE_STATUS status;
   status = TCn_REGS->COUNT16.TC_INTFLAG;
   /* clear period interrupt */
   TCn_REGS->COUNT16.TC_INTFLAG = TC_INTFLAG_OVF_Msk;
   if( TCn_CallbackObject.callback != NULL )
   {
      TCn_CallbackObject.callback( status, TCn_CallbackObject.context );
   }
}
```

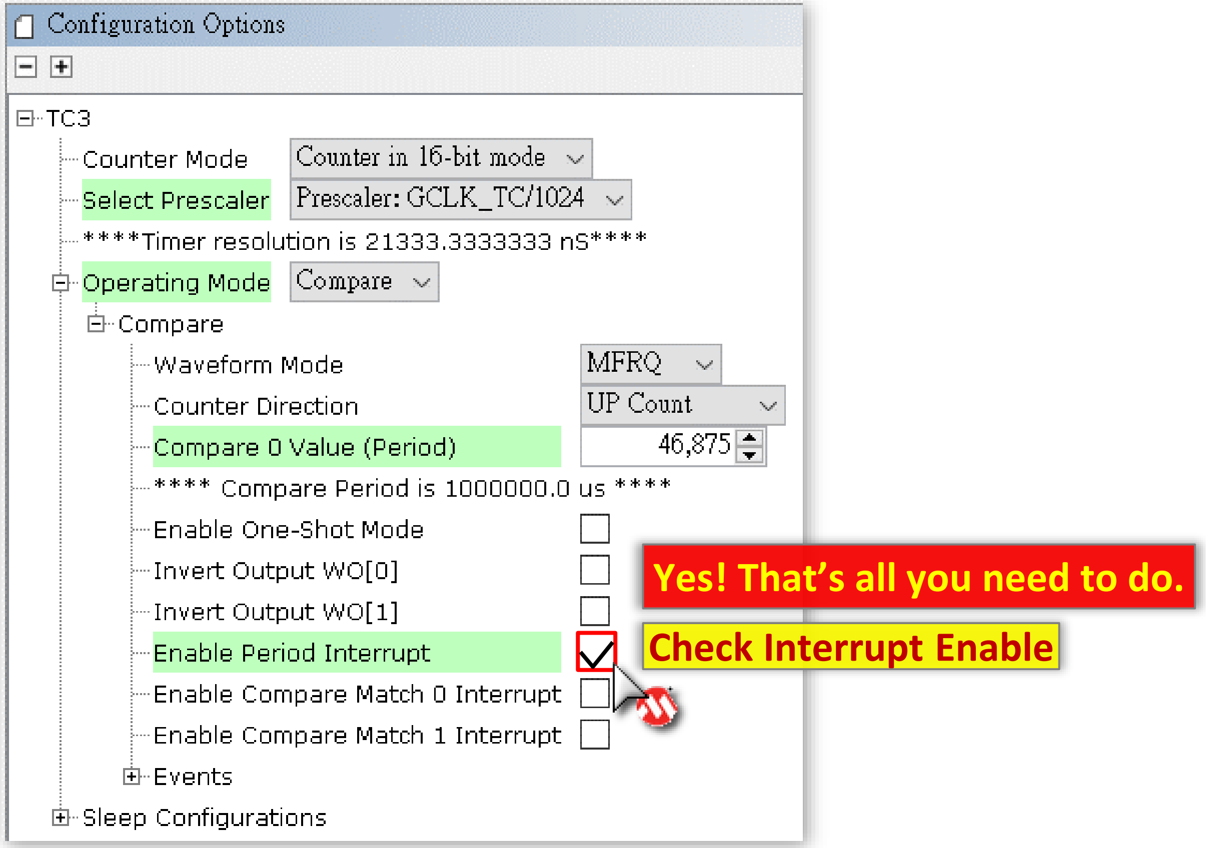
Callback

```
void TCn_Overflow(TC_COMPARE_STATUS status, uintptr_t context)
{
   if( status & TC_INTFLAG_OVF_Msk )
      LED_Toggle();
}
```

MICROCHIP

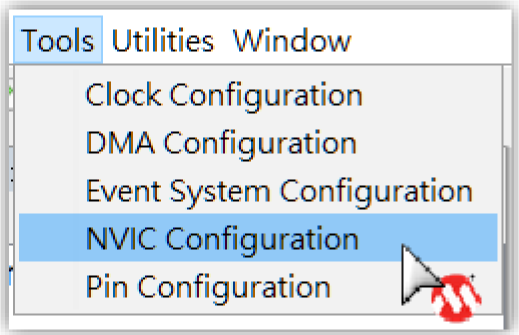
MH3 Interrupt Example

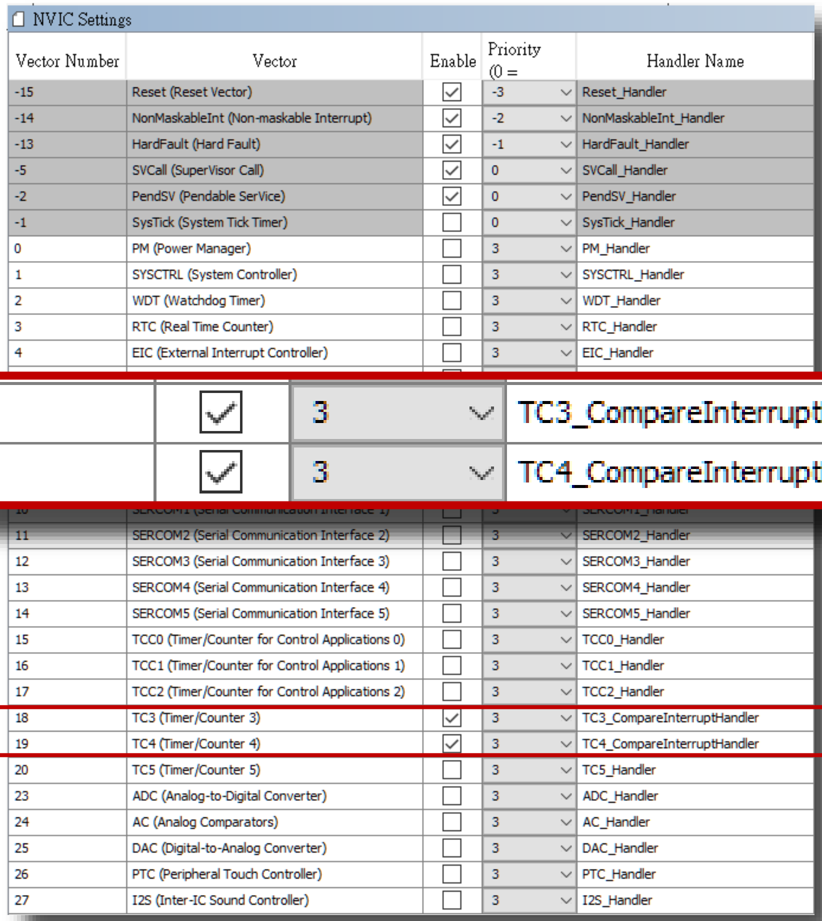
- **Enable Interrupt in TC configuration windows.**
  Only one click !



NVIC Priority Settings in MH3

- **Open NVIC Settings windows in Tools ▸ NVIC Configuration.**





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
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_CompareInterruptHandler
19	TC4 (Timer/Counter 4)	☑	3	TC4_CompareInterruptHandler
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

18	TC3 (Timer/Counter 3)	☑	3	TC3_CompareInterruptHandler
19	TC4 (Timer/Counter 4)	☑	3	TC4_CompareInterruptHandler

TC Interrupt

Interface Routines

```
void TC3_CompareInitialize( void );
void TC3_CompareStart( void );
void TC3_CompareStop( void );
uint32_t TC3_CompareFrequencyGet( void );
void TC3_Compare16bitPeriodSet( uint16_t period );
uint16_t TC3_Compare16bitPeriodGet( void );
uint16_t TC3_Compare16bitCounterGet( void );
void TC3_Compare16bitCounterSet( uint16_t count );
void TC3_CompareCallbackRegister
                ( TC_COMPARE_CALLBACK callback, uintptr_t context );
```

One new function than polling mode.

Lab5 TC MFRQ Interrupt Callback

- **Try to configure code style to interrupt callback style. LED1 toggle in callback while TC3 overflow interrupt.**
- **Modify TC3 period to 1 second in MH3 for easy observe.**
- **Modify code segment in main() follow below sequence.**
  - Firstly, remove polling LED1 toggle from main loop.
  - Secondary, Create a callback function for LED1 toggle
  - Finally, hook up callback function to TC3 overflow event.

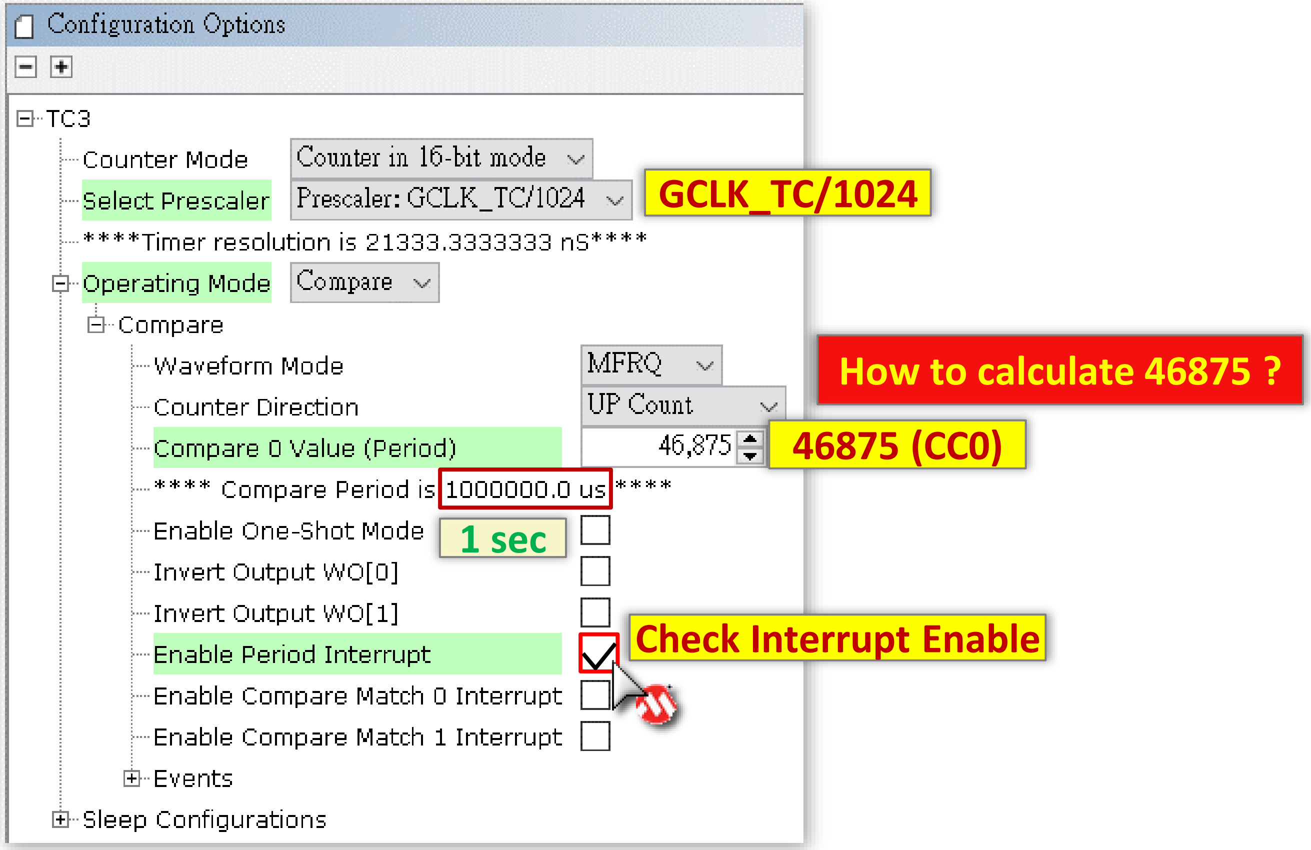
- **Let’s go!**

MICROCHIP

Lab5 TC MFRQ Interrupt Callback

Step 1

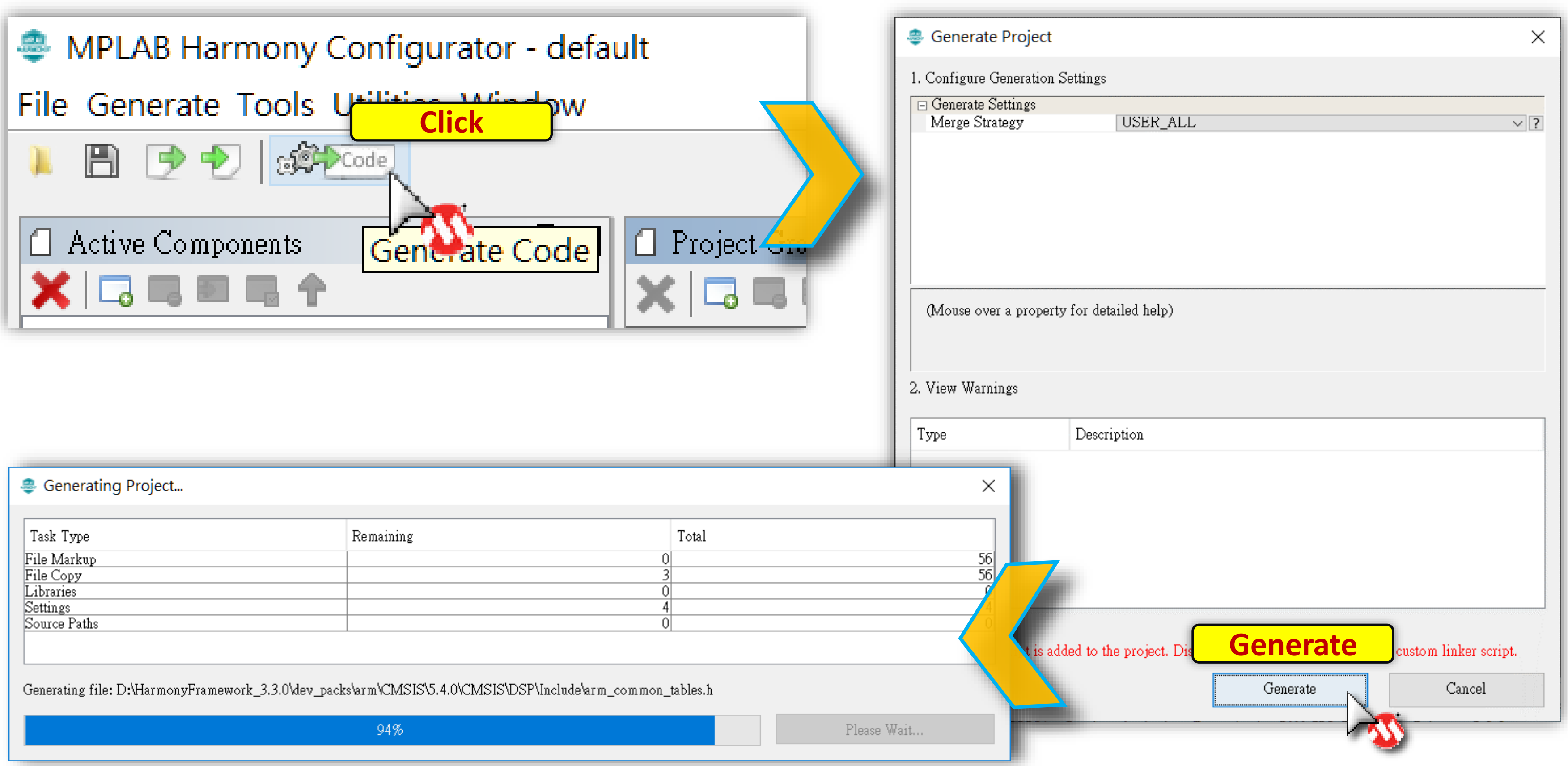
- Configure TC3 Period to 1 second.
- Enable Period Interrupt of TC3 in configuration windows.



Lab5 TC MFRQ Interrupt Callback

Step 2

- Click  to Generate Code and save changes to MHC configure file.



Lab5 TC MFRQ Interrupt Callback

Step 3

a Add code segment to your main loop

```
// TODO 5.01
void TC3_Overflow(TC_COMPARE_STATUS status, uintptr_t context)
{
  if( status & TC_INTFLAG_OVF_Msk )
    LED1_Toggle();
}

int main (void)
{
  SYS_Initialize ( NULL );
  …
  // TODO 5.02
  TC3_CompareCallbackRegister( TC3_Overflow, (uintptr_t )NULL );
  TC3_CompareStart();

  while(1)
  { …
  }
}
```

b Program firmware to target board then observe result.

Lab6 TCs MFRQ Interrupt Callback

- Try to change TC4, LED2 toggle function to interrupt callback style, also.

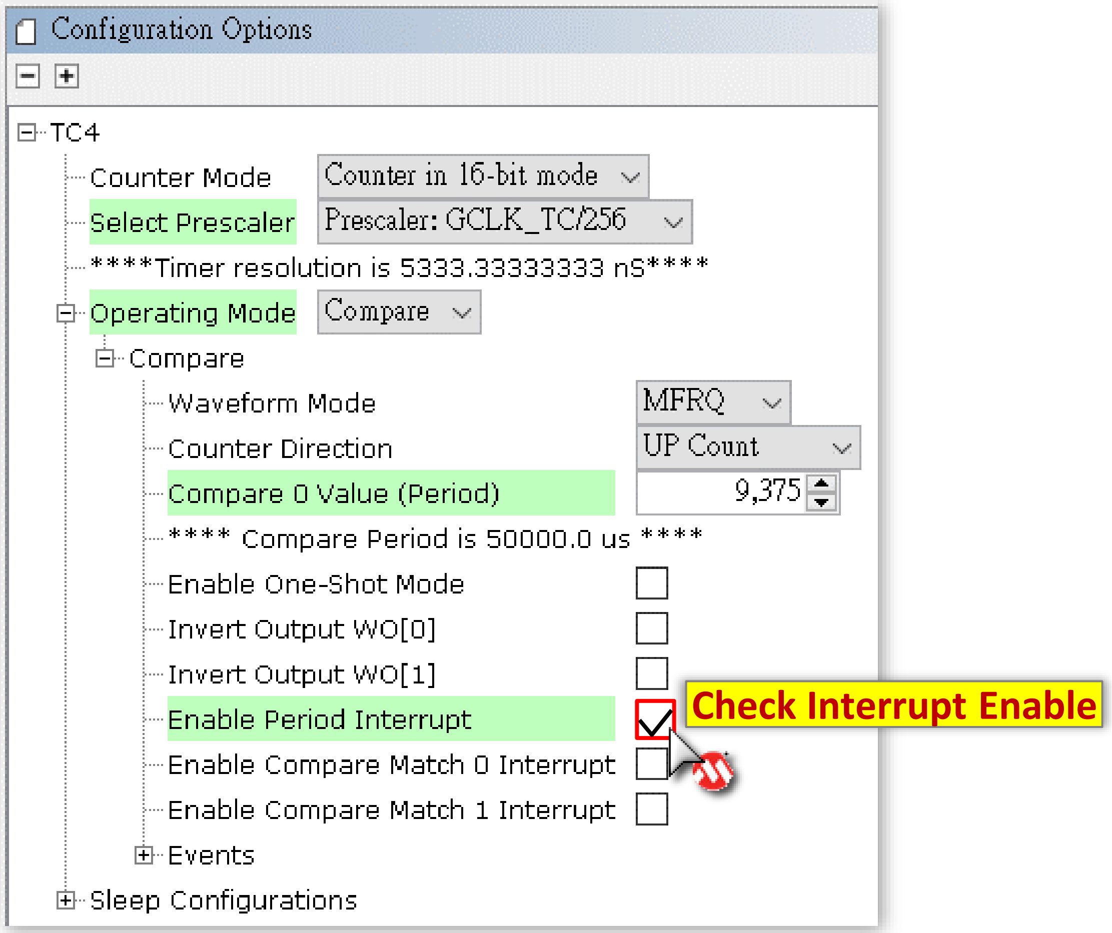
- Let’s go！

MICROCHIP

Lab6 TCs MFRQ Interrupt Callback

Step 1

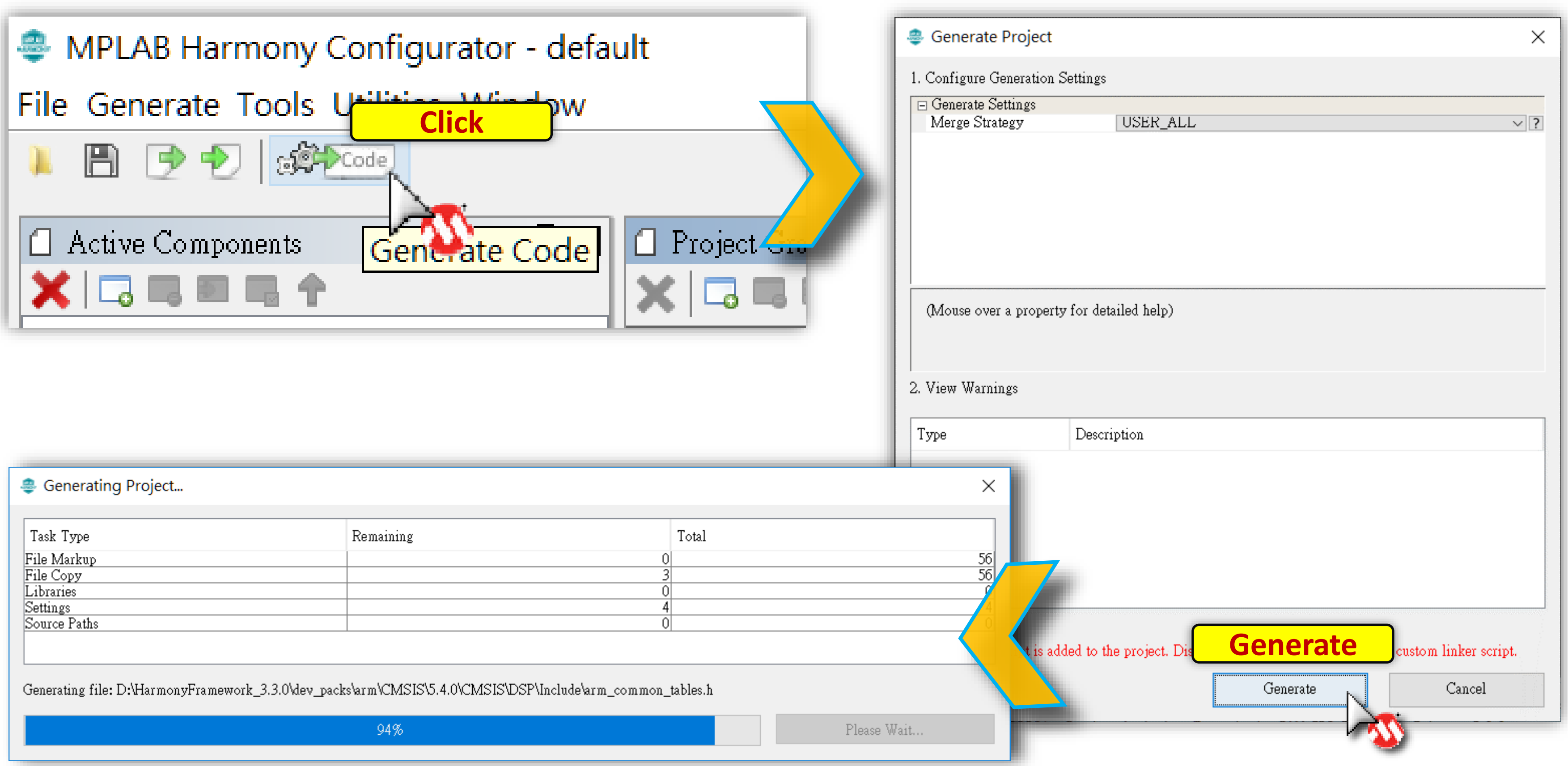
- **Enable Period Interrupt** of TC4 in configuration windows.



Lab6 TCs MFRQ Interrupt Callback

Step 2

- Click  to Generate Code and save changes to MHC configure file.



Lab6 TCs MFRQ Interrupt Callback

Step 3

a Add code segment to your main loop

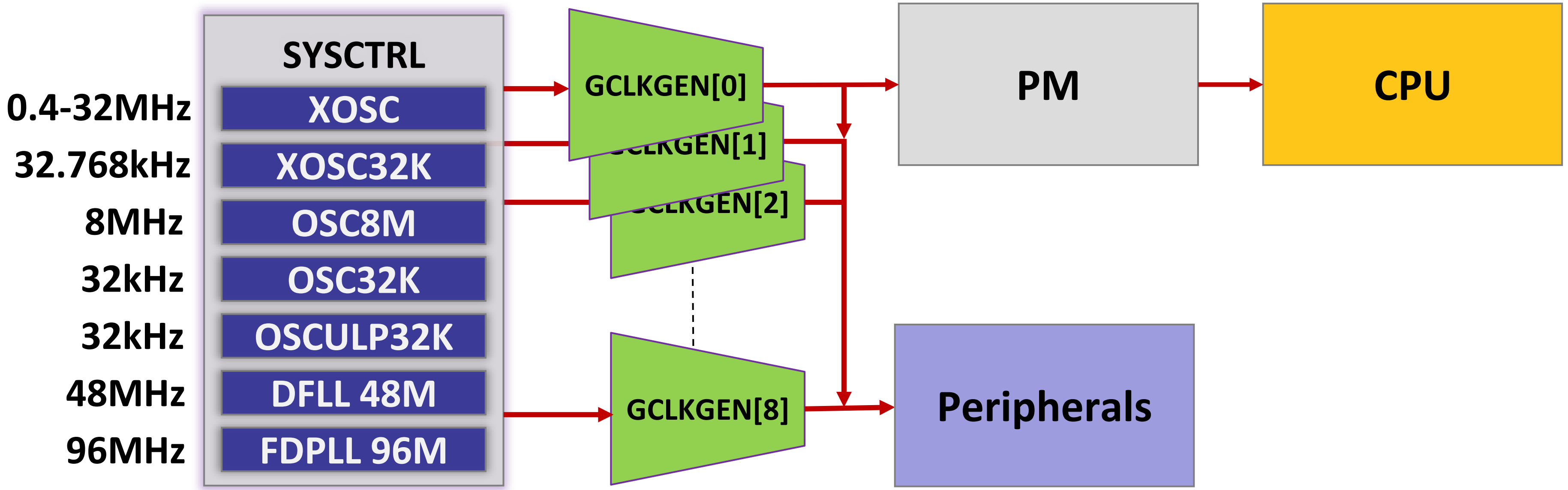
```
// TODO 6.01
void TC4_Overflow(TC_COMPARE_STATUS status, uintptr_t context)
{
  if( status & TC_INTFLAG_OVF_Msk )
    LED2_Toggle();
}

int main (void)
{
  SYS_Initialize ( NULL );
  …
  // TODO 6.02
  TC4_CompareCallbackRegister( TC4_Overflow, (uintptr_t )NULL );
  TC4_CompareStart();

  while(1)
  { …
  }
}
```

b Program firmware to target board then observe result.

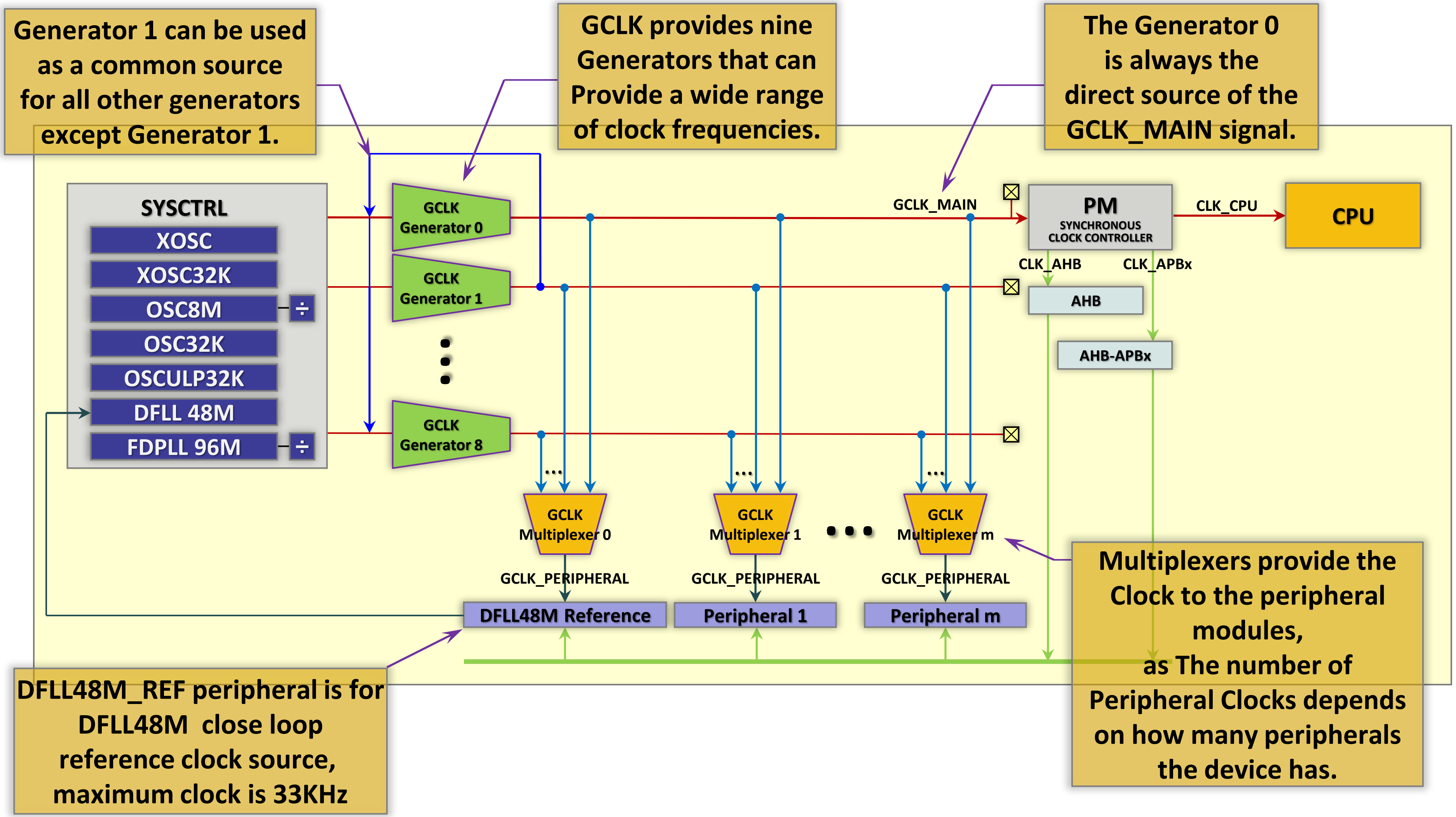
Clock System Architecture

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)。

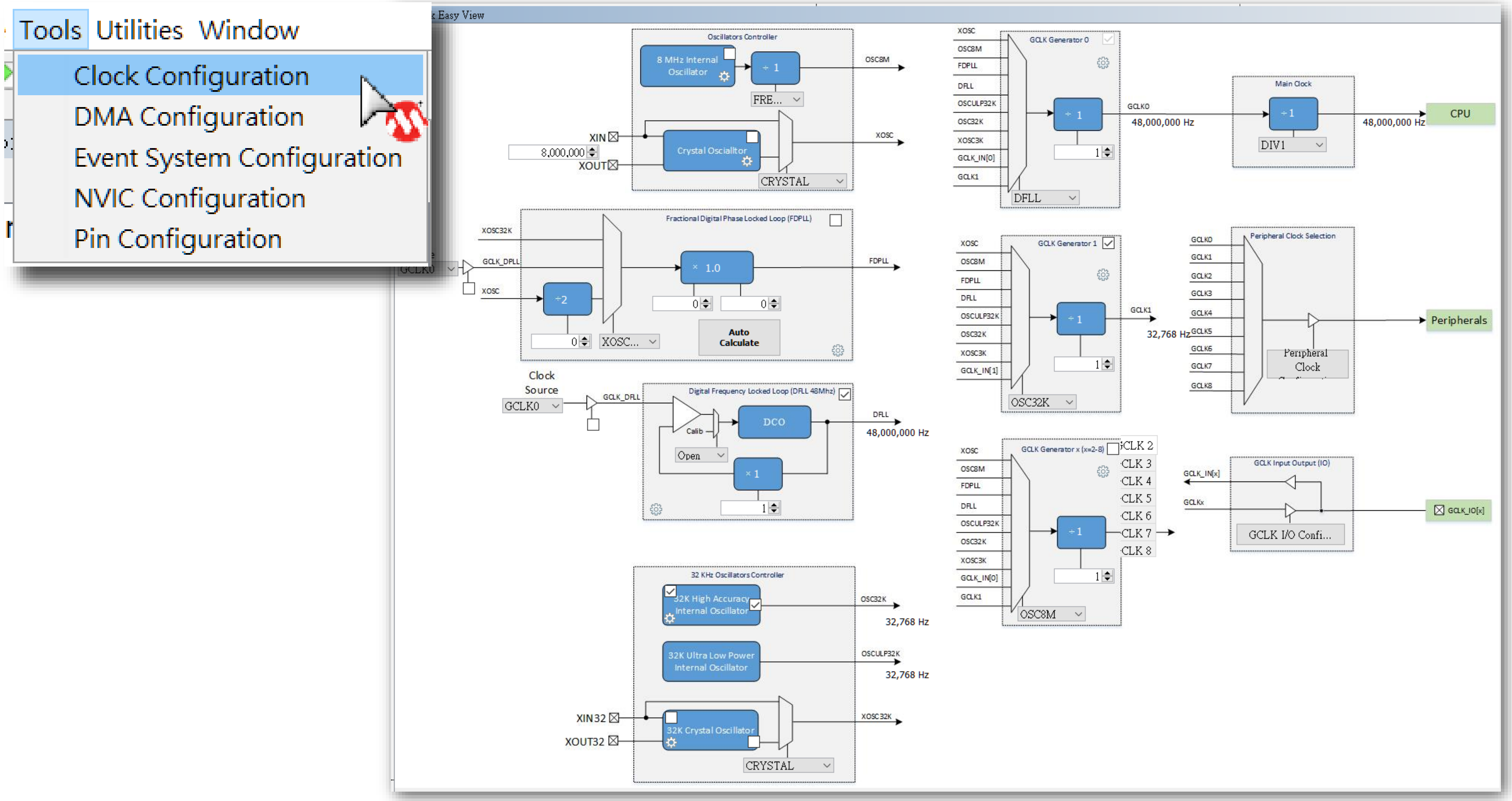


Clock System Block Diagram



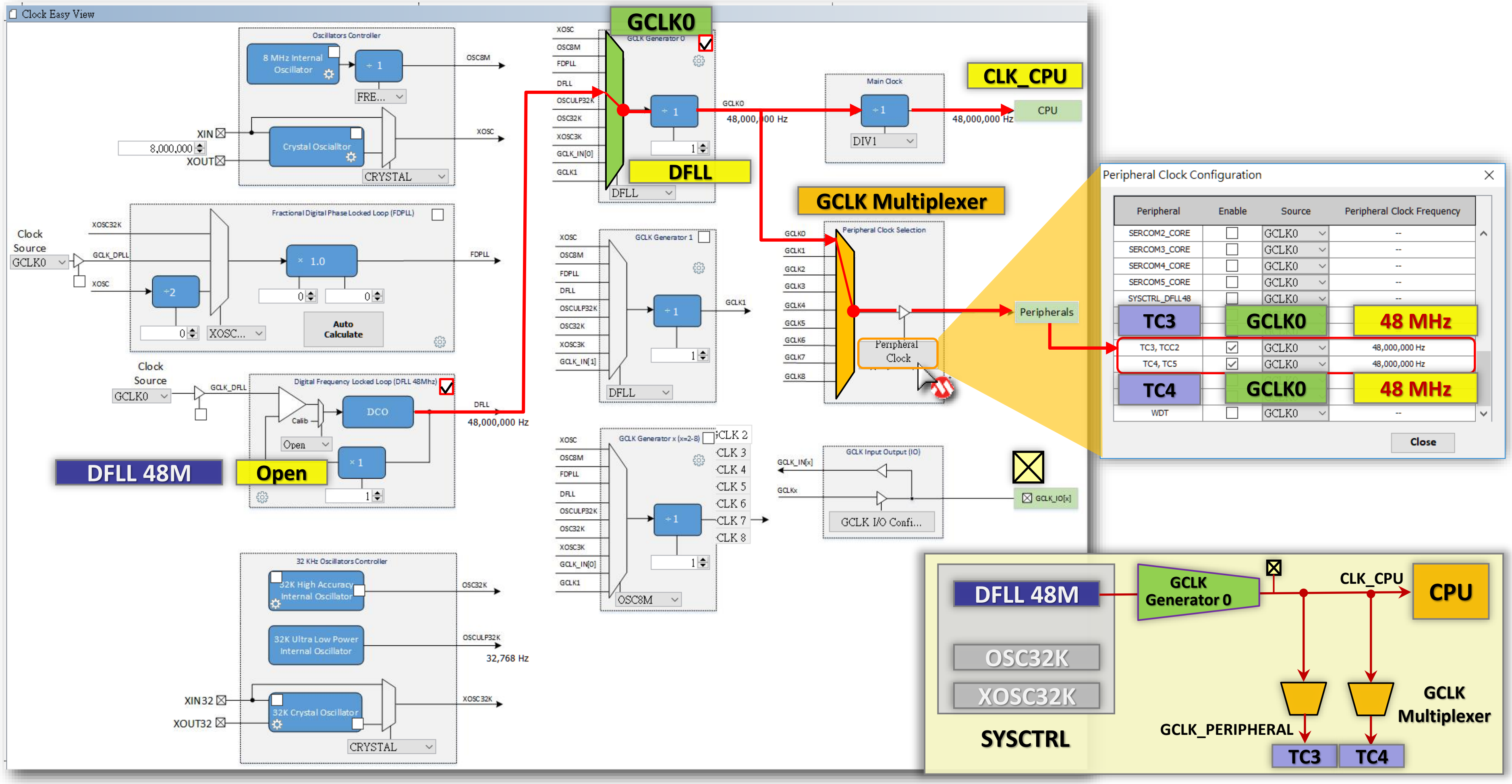
Clock Setting Use MH3

- **Open Clock Easy View windows in Tools ▸ Clock Configuration.**



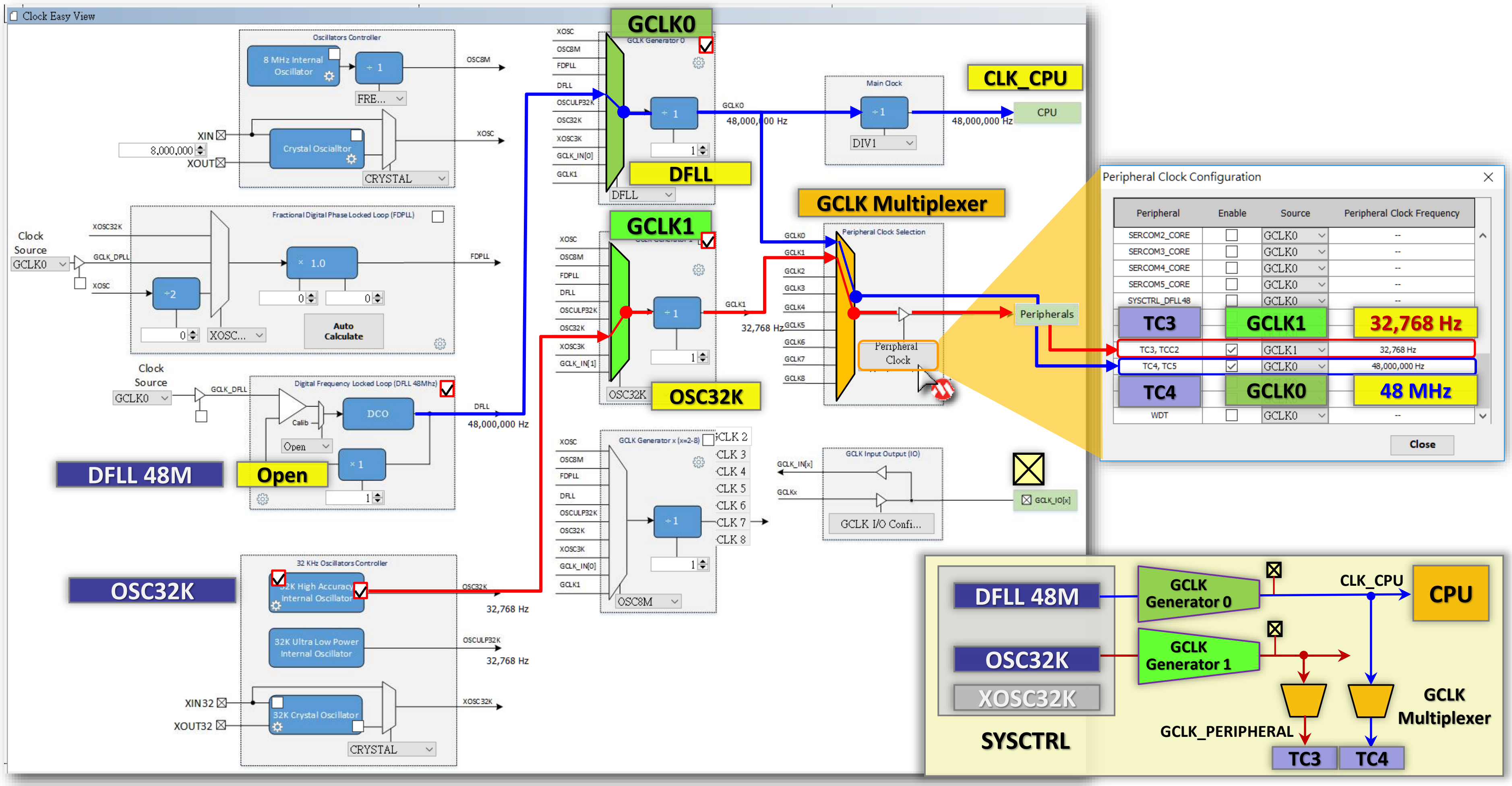
Clock Setting Use MH3

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



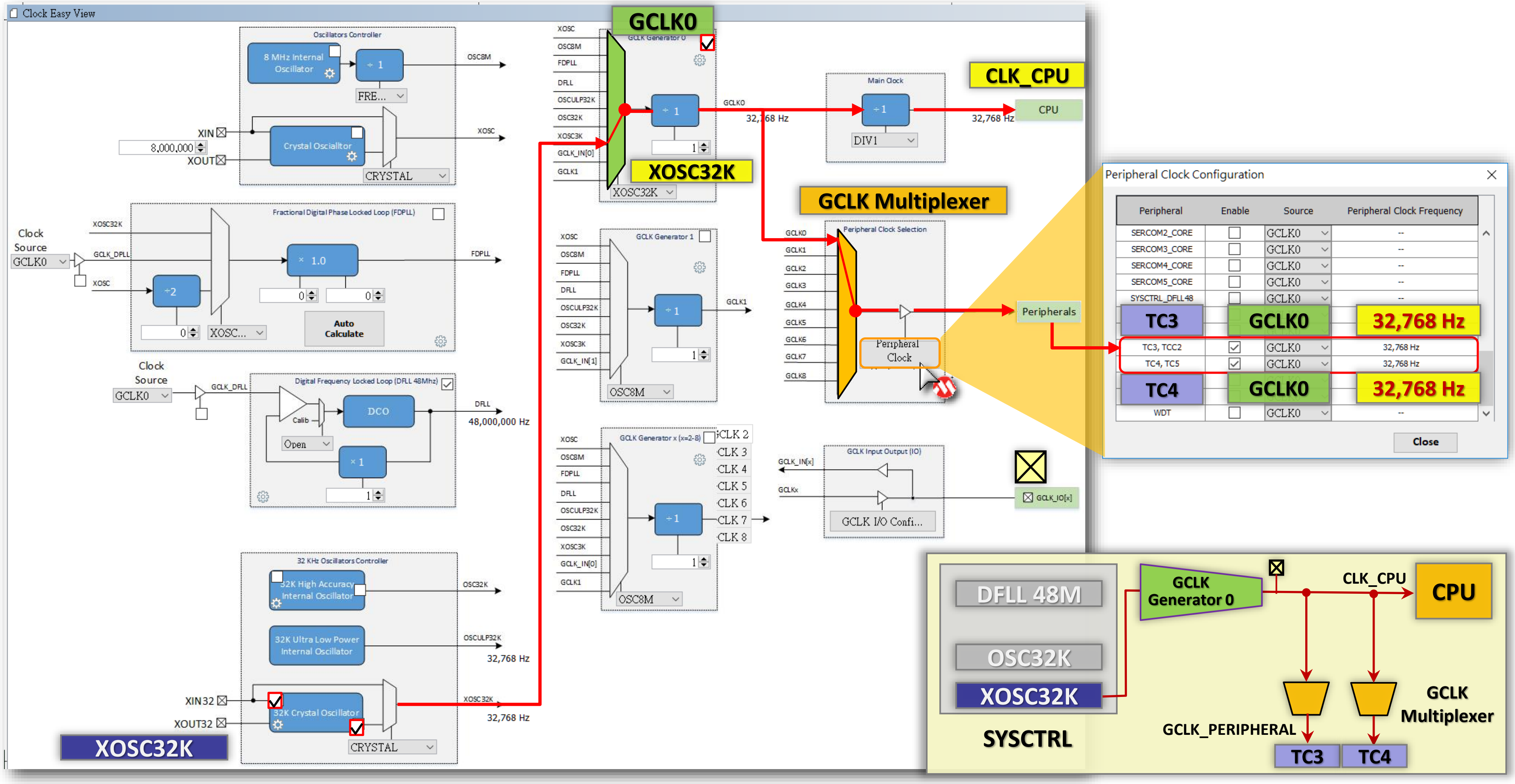
Clock Setting Use MH3

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



Clock Setting Use MH3

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



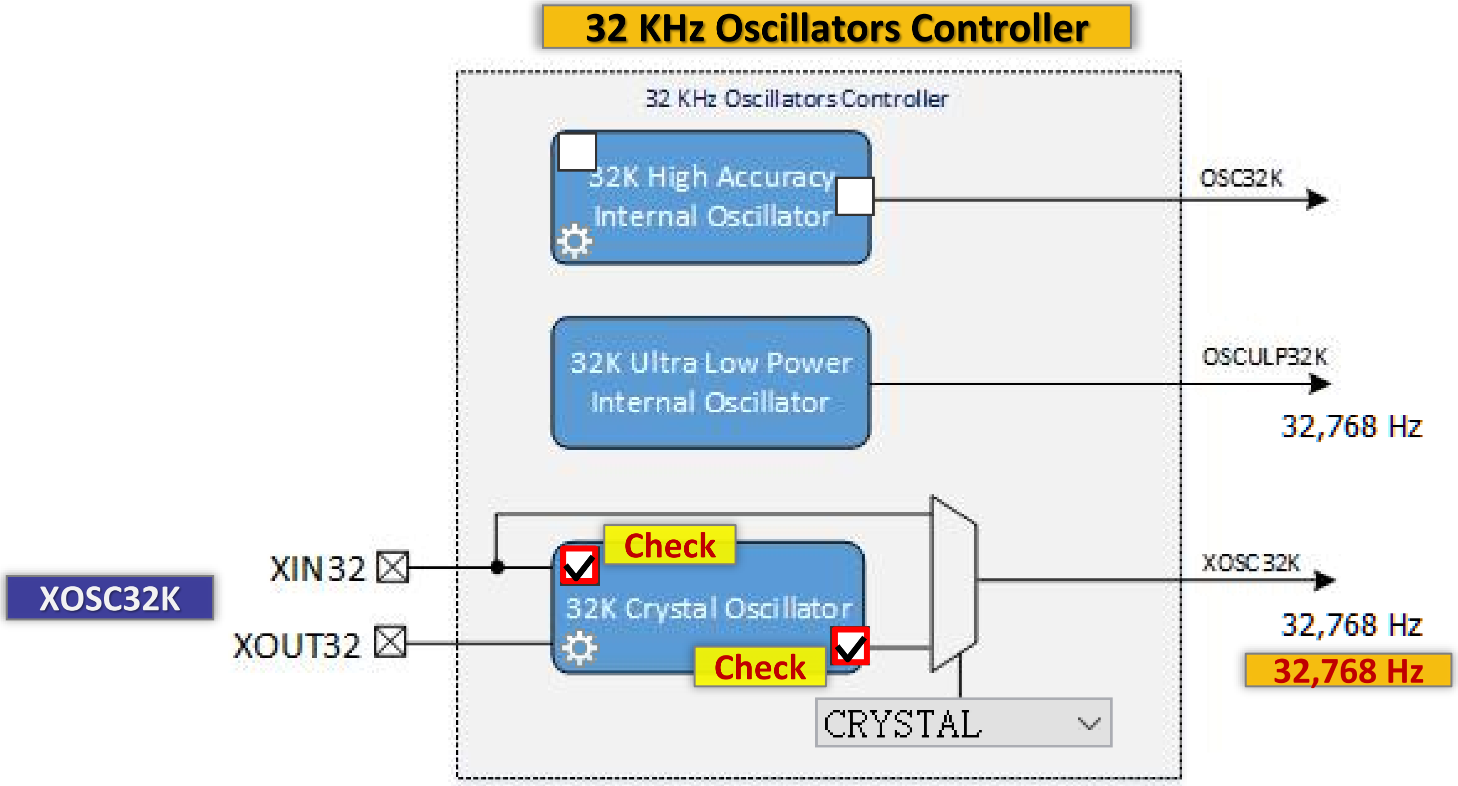
Lab7 System Clock XOSC32K

- 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!

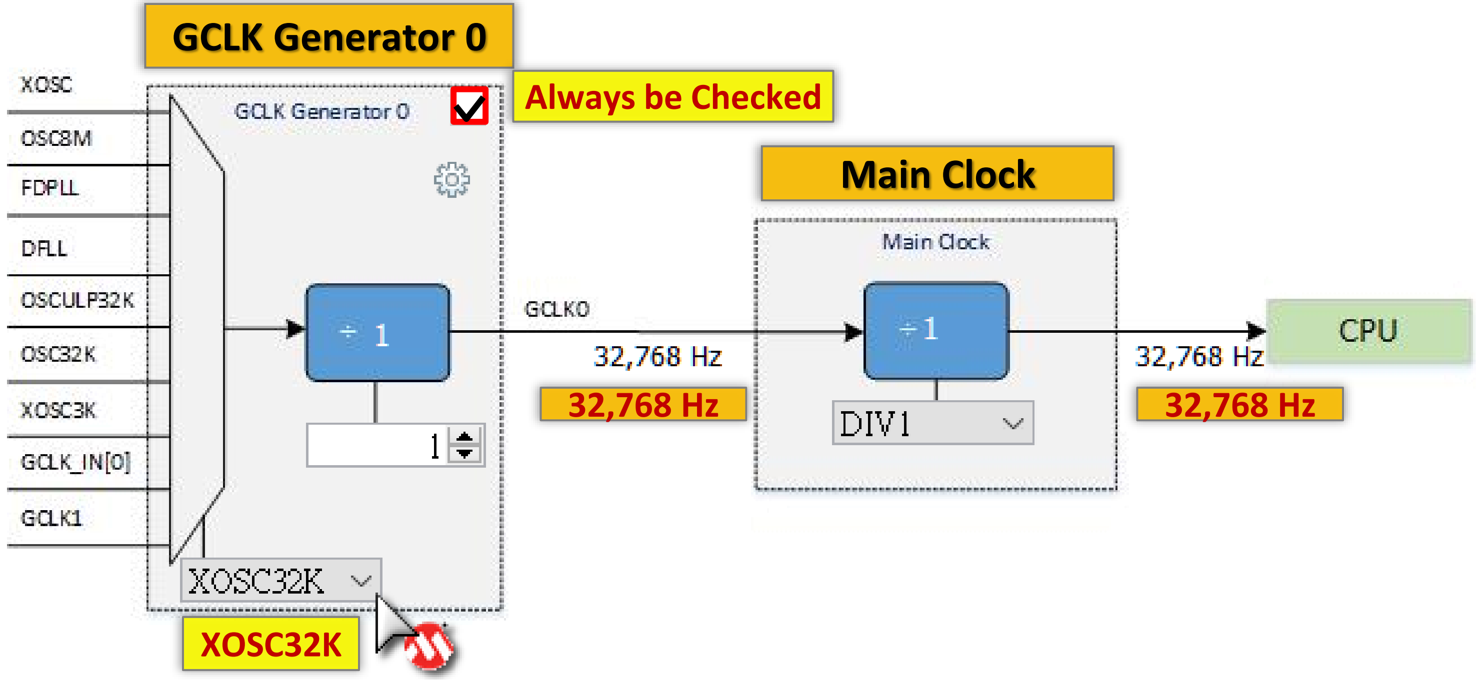
Lab7 System Clock XOSC32K

Step 1



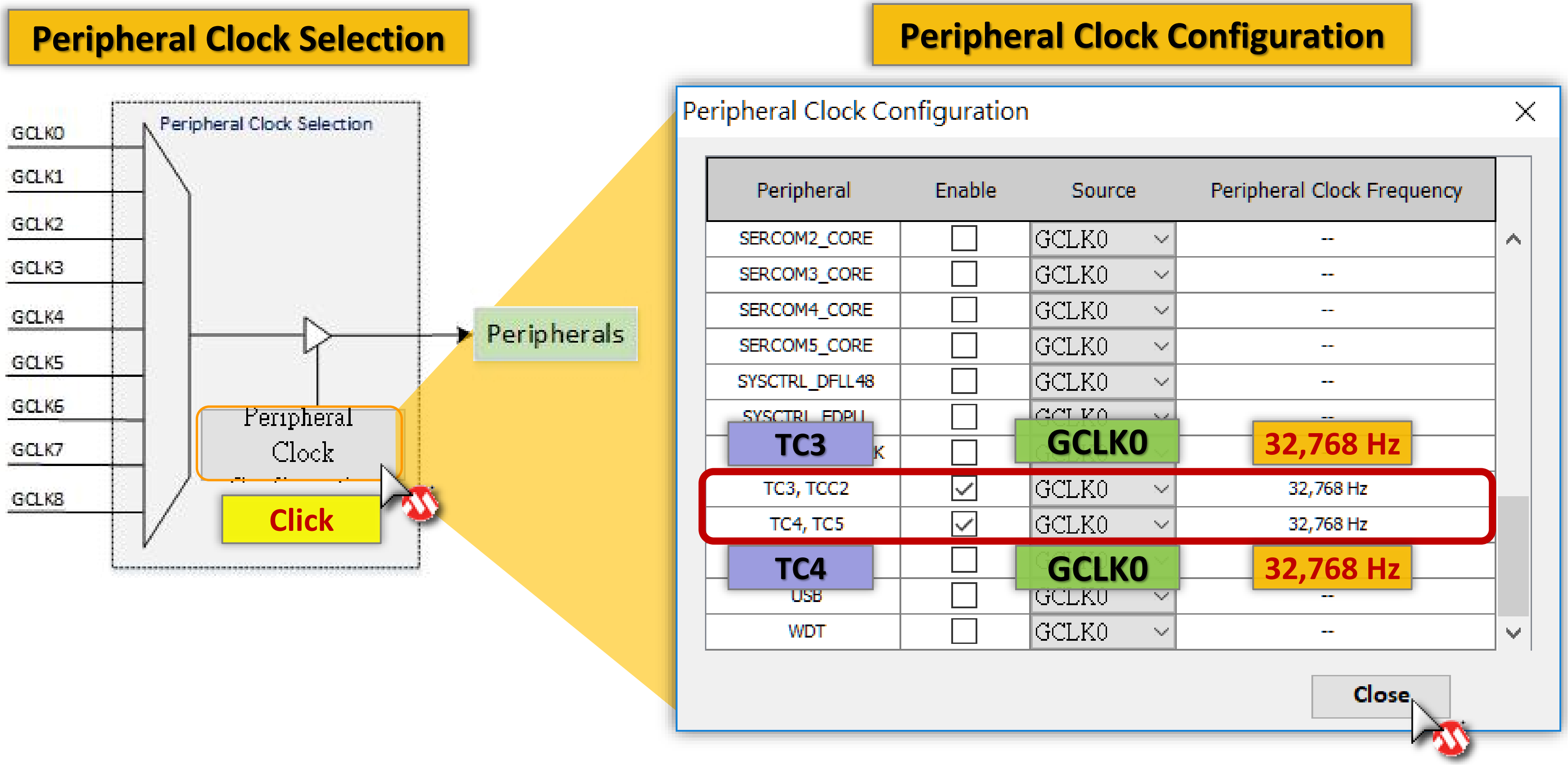
Lab7 System Clock XOSC32K

Step 2



Lab7 System Clock XOSC32K

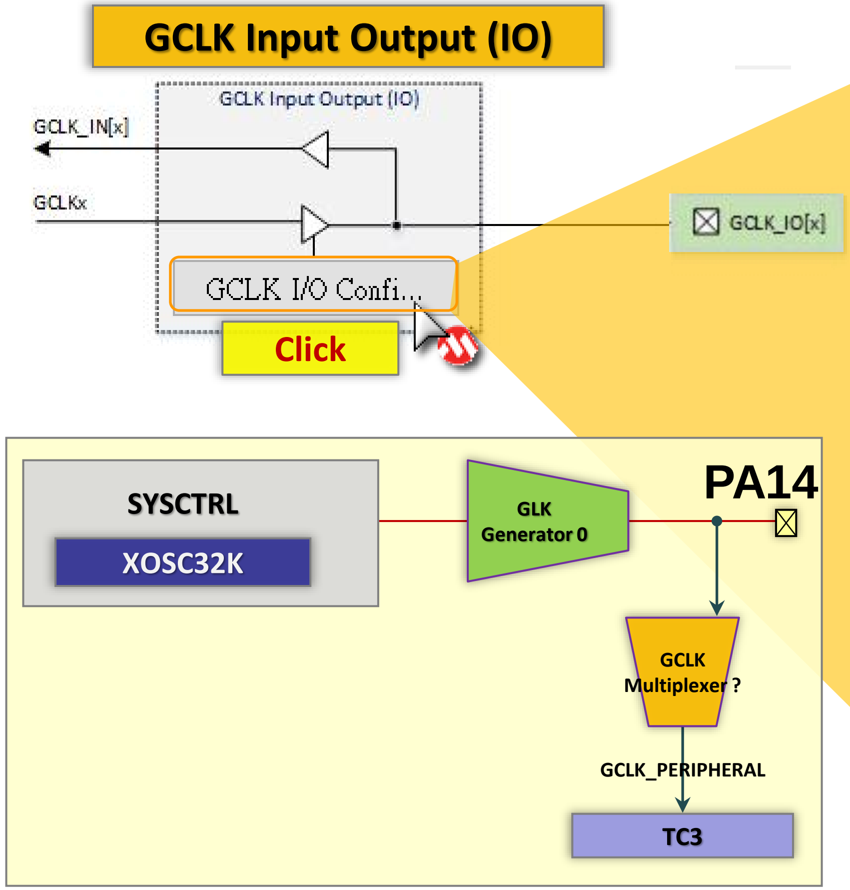
Step 3

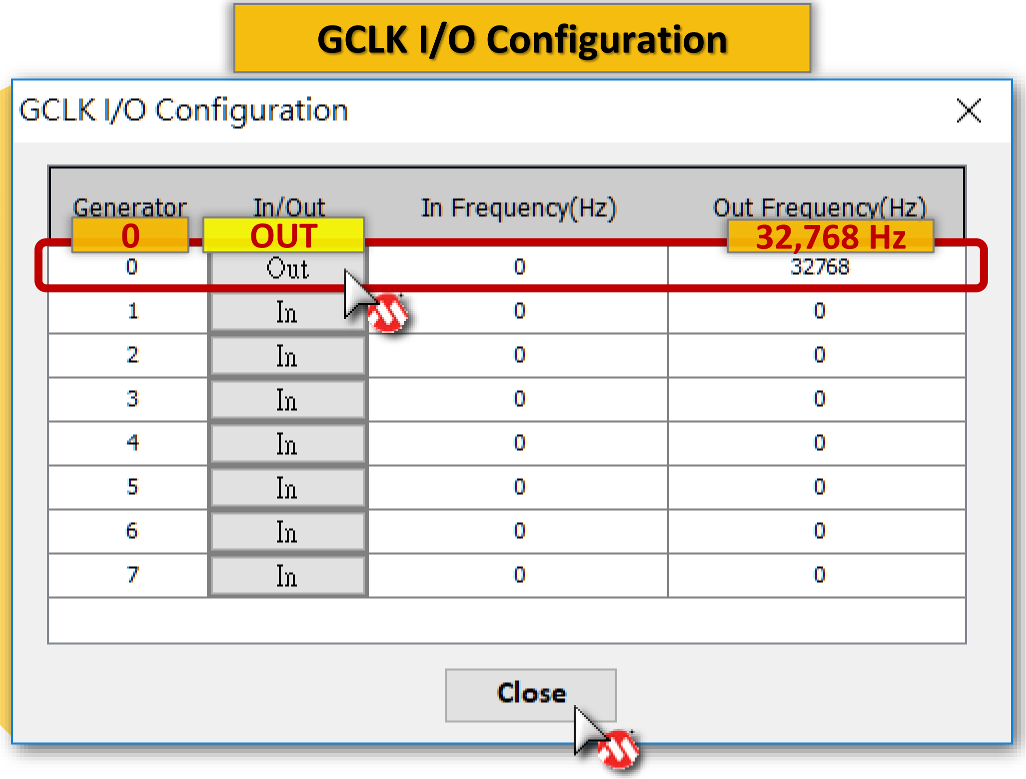


Lab7 System Clock XOSC32K

Step 4

- **Configure GCLK Generator 0 to output clock.**



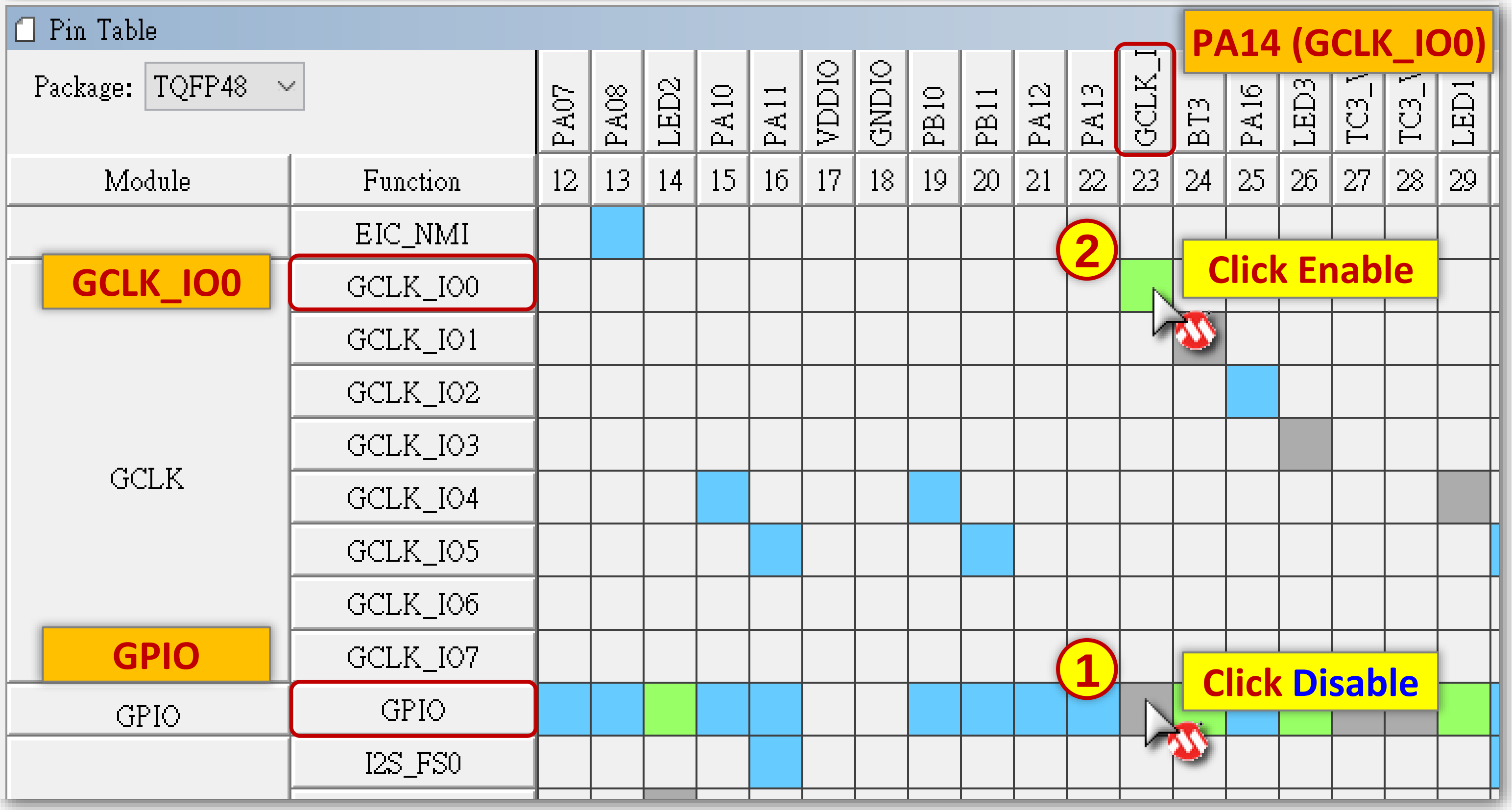


Generator	In/Out	In Frequency(Hz)	Out Frequency(Hz)
0	Out	0	32768
1	In	0	0
2	In	0	0
3	In	0	0
4	In	0	0
5	In	0	0
6	In	0	0
7	In	0	0

Lab7 System Clock XOSC32K

Step 5

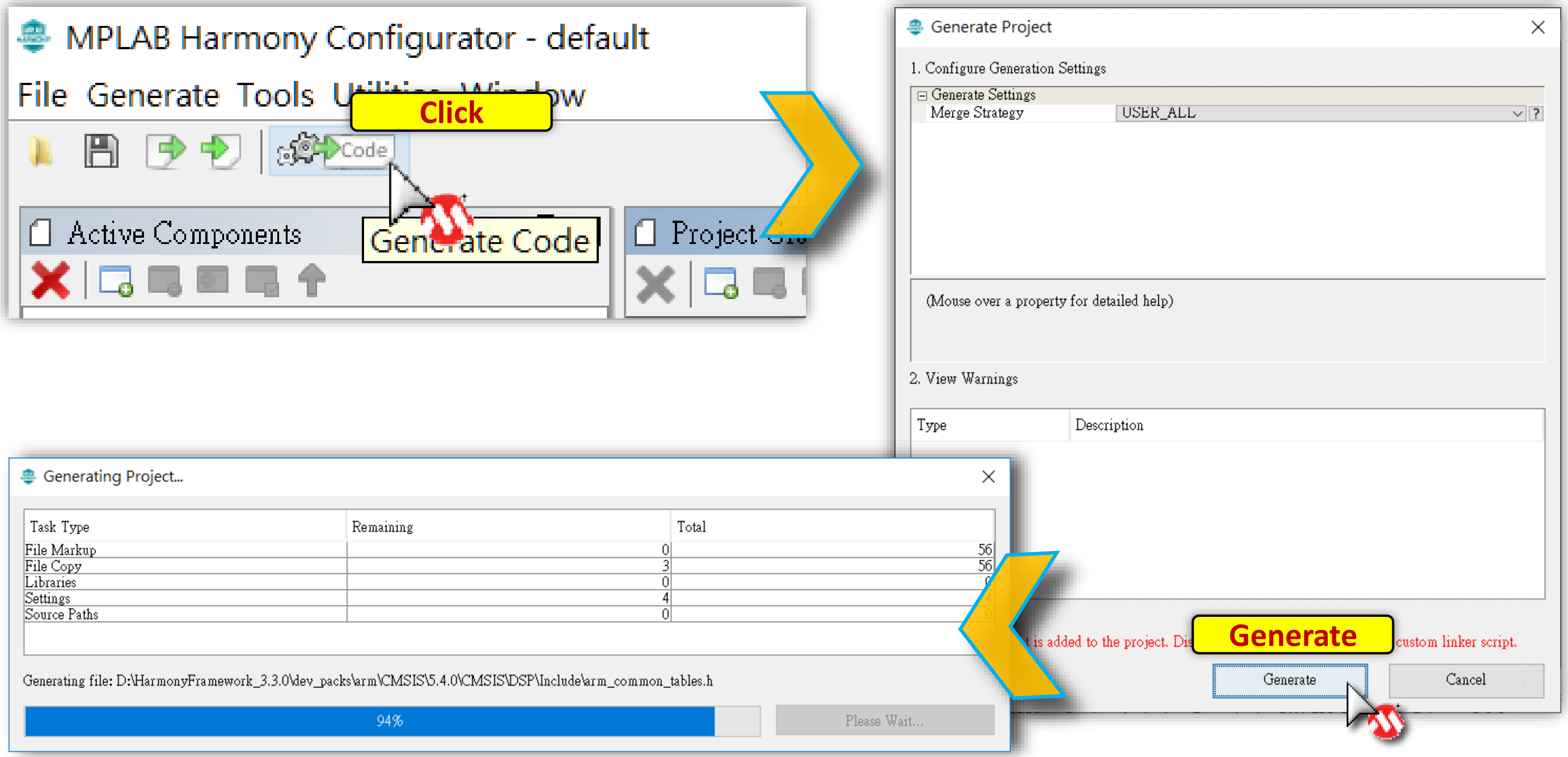
- Please **disable BT2(PA14) GPIO** pin function firstly and then reassign PA14 as **GCLK_IO0** output pin.



Lab7 System Clock XOSC32K

Step 6

- Click [Code] to Generate Code and save changes to MHC configure file.



Lab7 System Clock XOSC32K

Step 7

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

```
...

int main (void)
{
   SYS_Initialize ( NULL );

   while(1)
   {
      ...

      // TODO 7.01
      // if ( BT2_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	

CPU_CLK = 32.768KHz

VDD = 3.3V

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

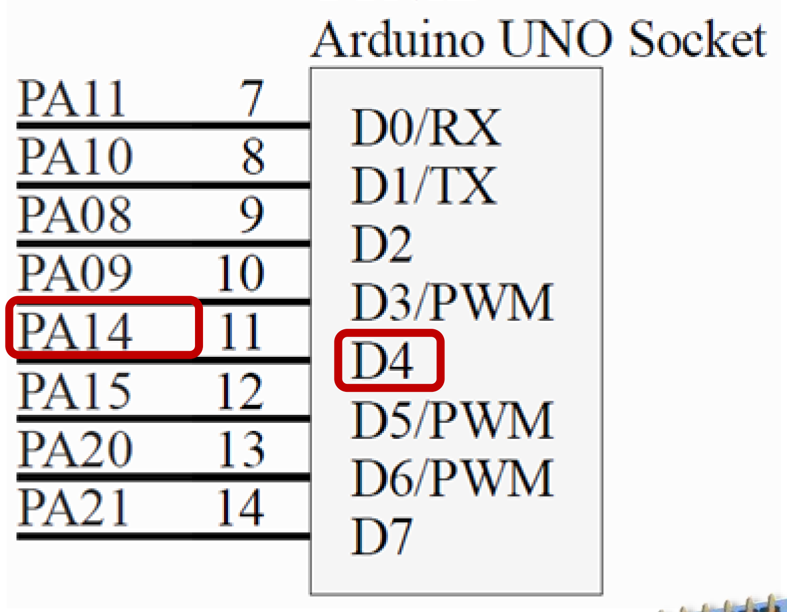
plib_nvmctrl.c

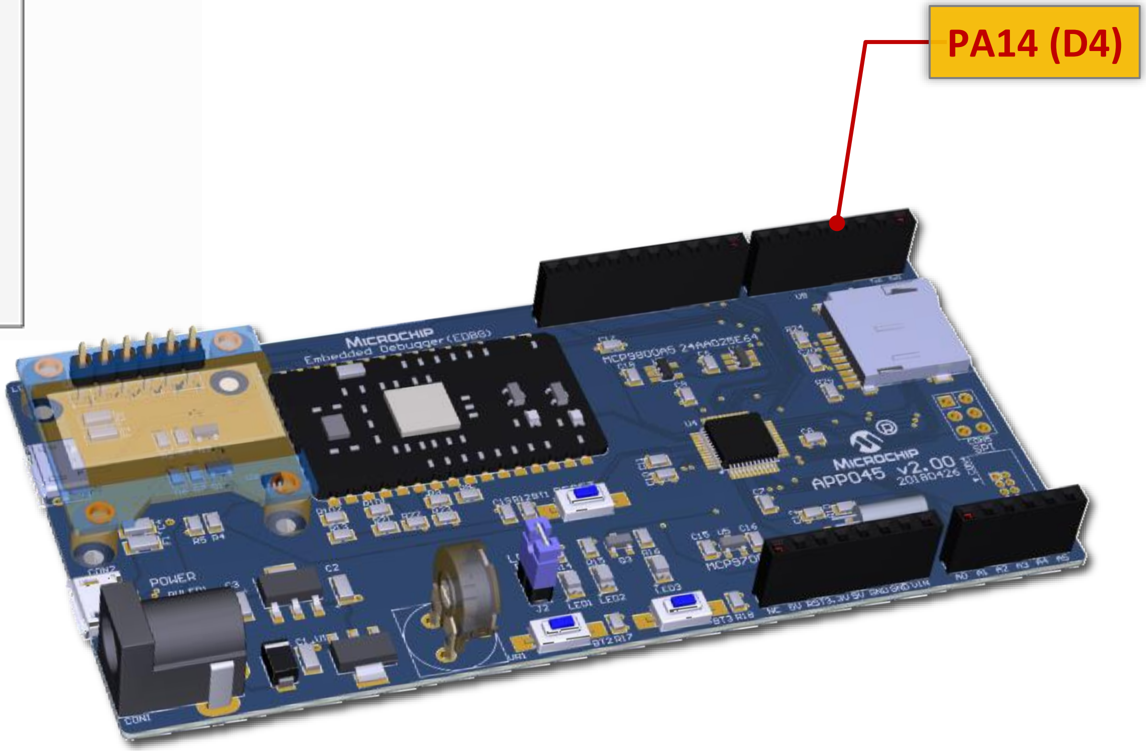
```
void NVMCTRL_Initialize(void)
{
    NVMCTRL_REGS->NVMCTRL_CTRLB =
        NVMCTRL_CTRLB_READMODE_NO_MISS_PENALTY |
        NVMCTRL_CTRLB_SLEEPPRM_WAKEONACCESS |
        NVMCTRL_CTRLB_RWS(0) |
        NVMCTRL_CTRLB_MANW_Msk;
}
```

MICROCHIP

Lab7 System Clock XOSC32K

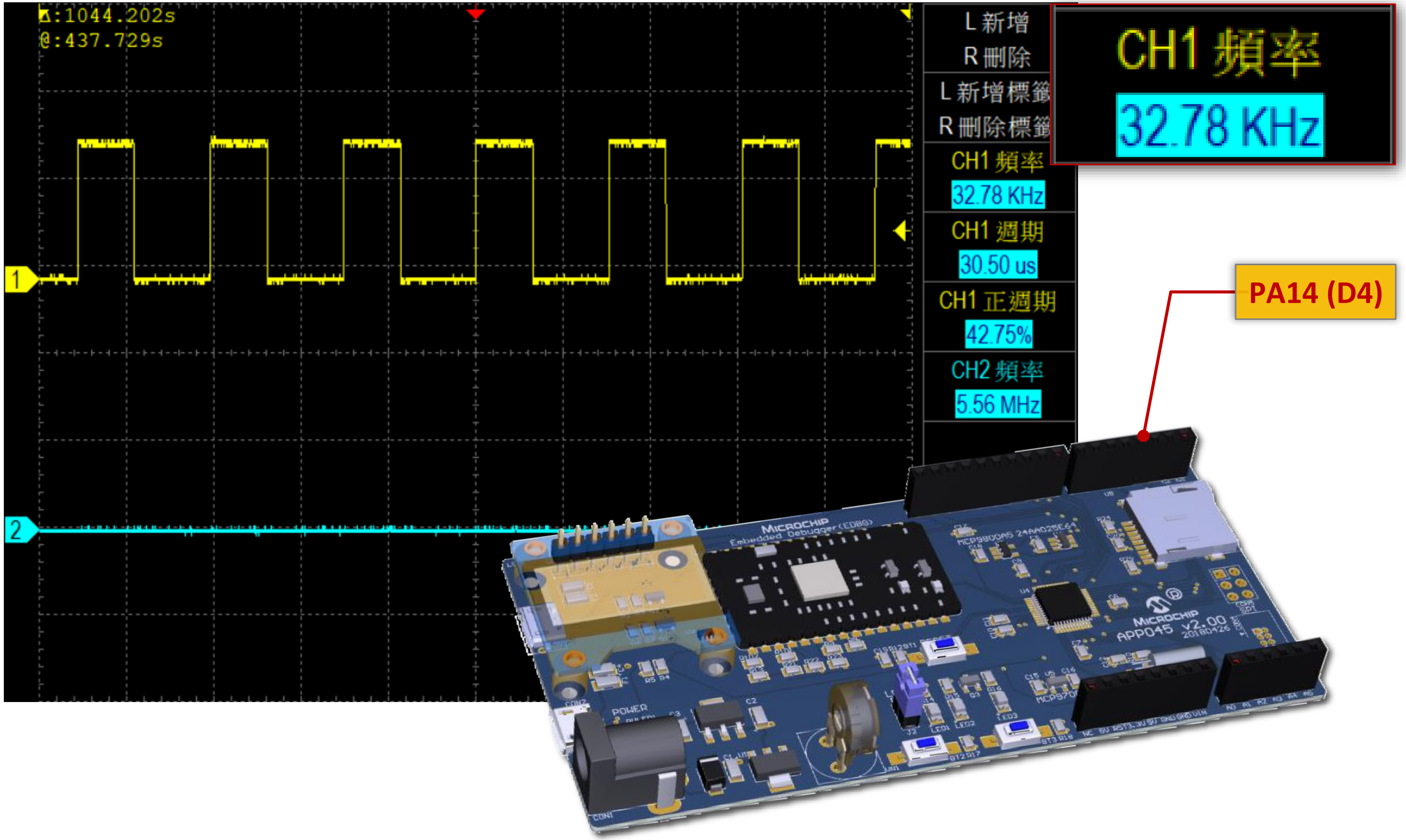
Clock Output





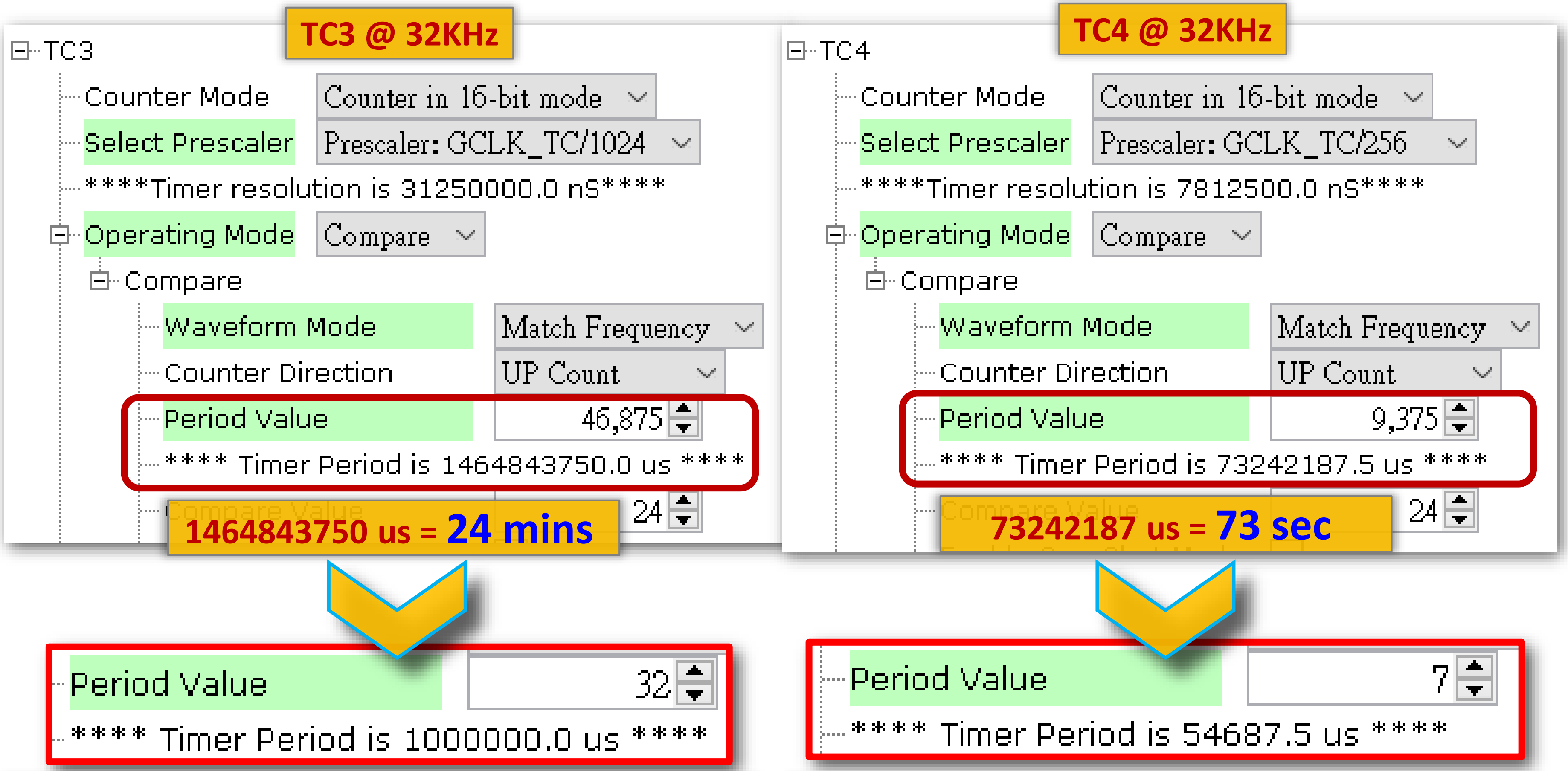
Lab7 System Clock XOSC32K

Clock Output



Lab7 System Clock XOSC32K

Step 8 LED toggle period correct



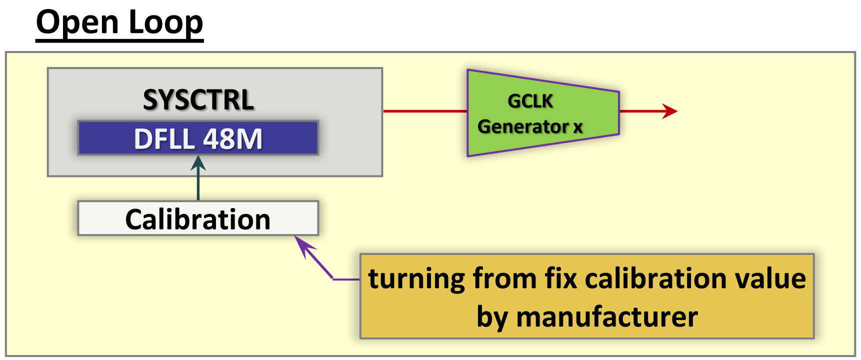
Correct Period Value to **32** to get ≈ 1sec period

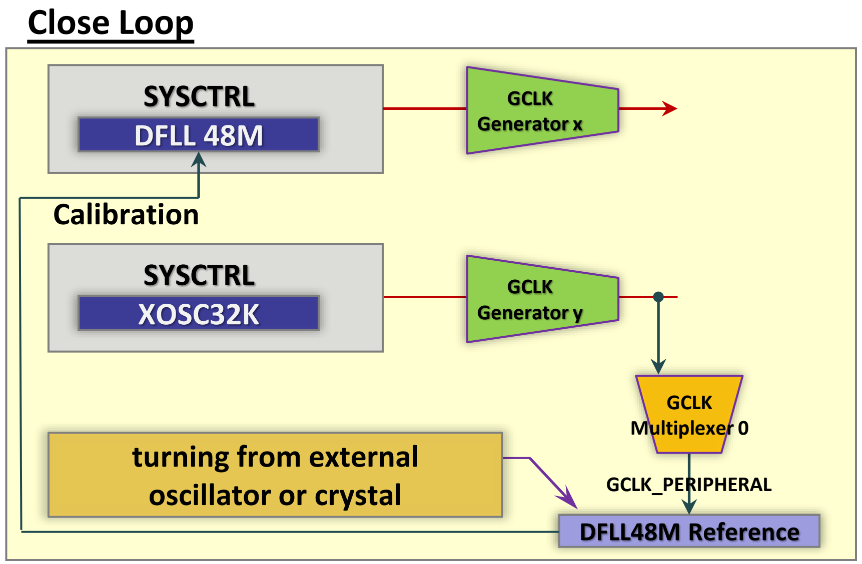
Correct Period Value to **7** to get ≈ 50ms period

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.





Lab8 System Clock DFLL48M

Close Loop

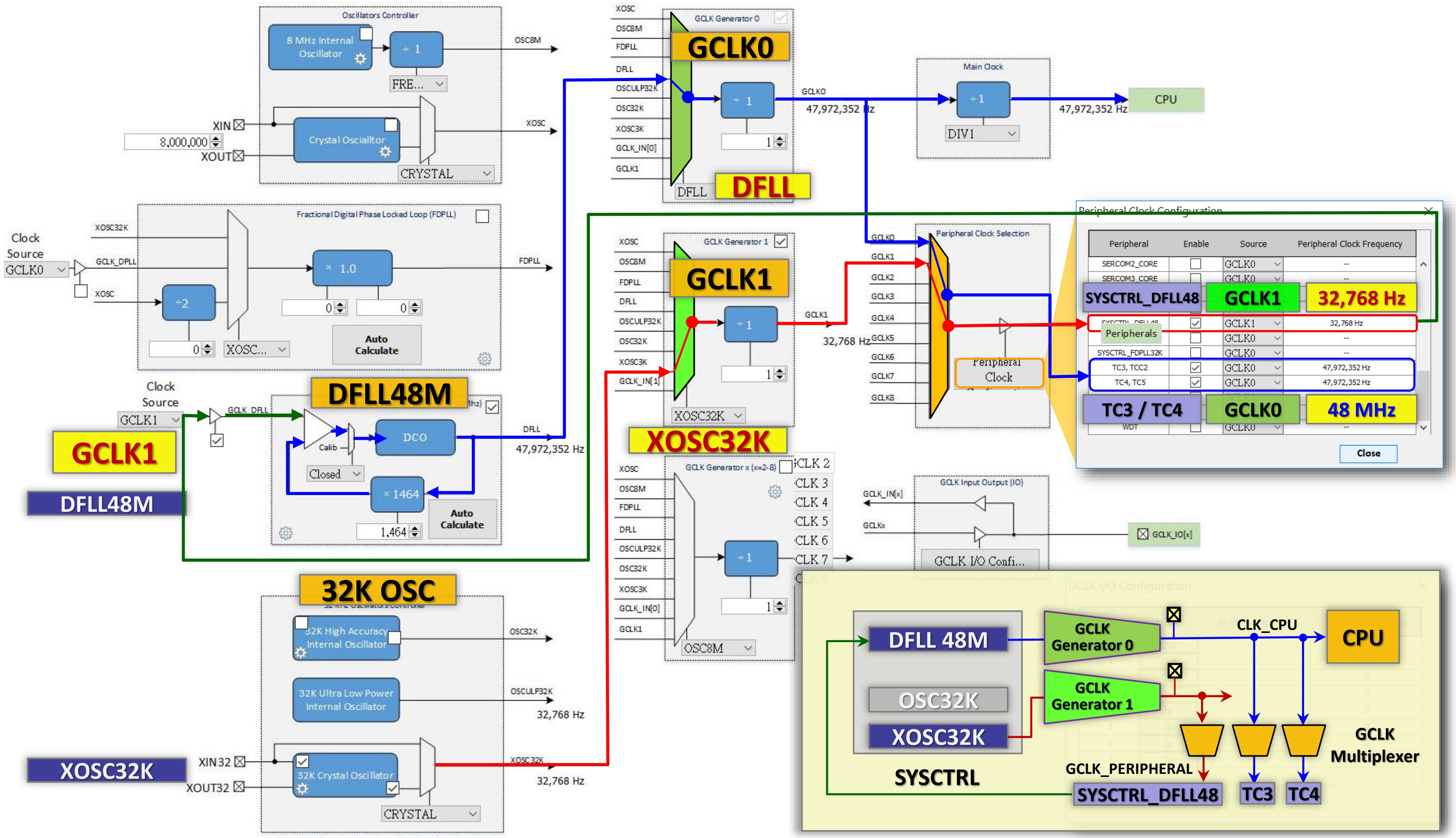
Configure DFLL48M(Close loop) as source of main clock(GCLK0).

- 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 MHC.

- **Let's go!**

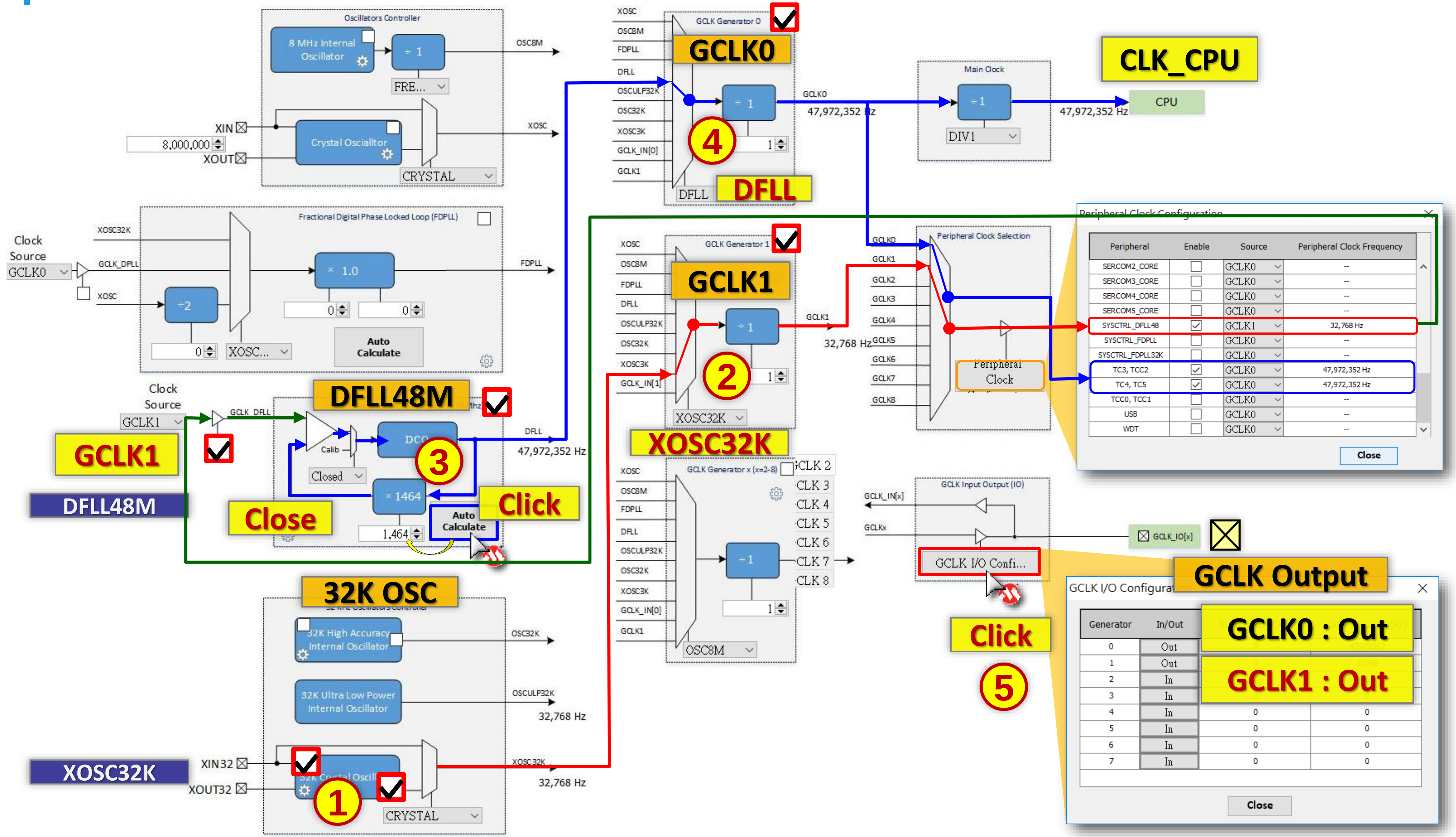
Lab8 System Clock DFLL48M

Overview



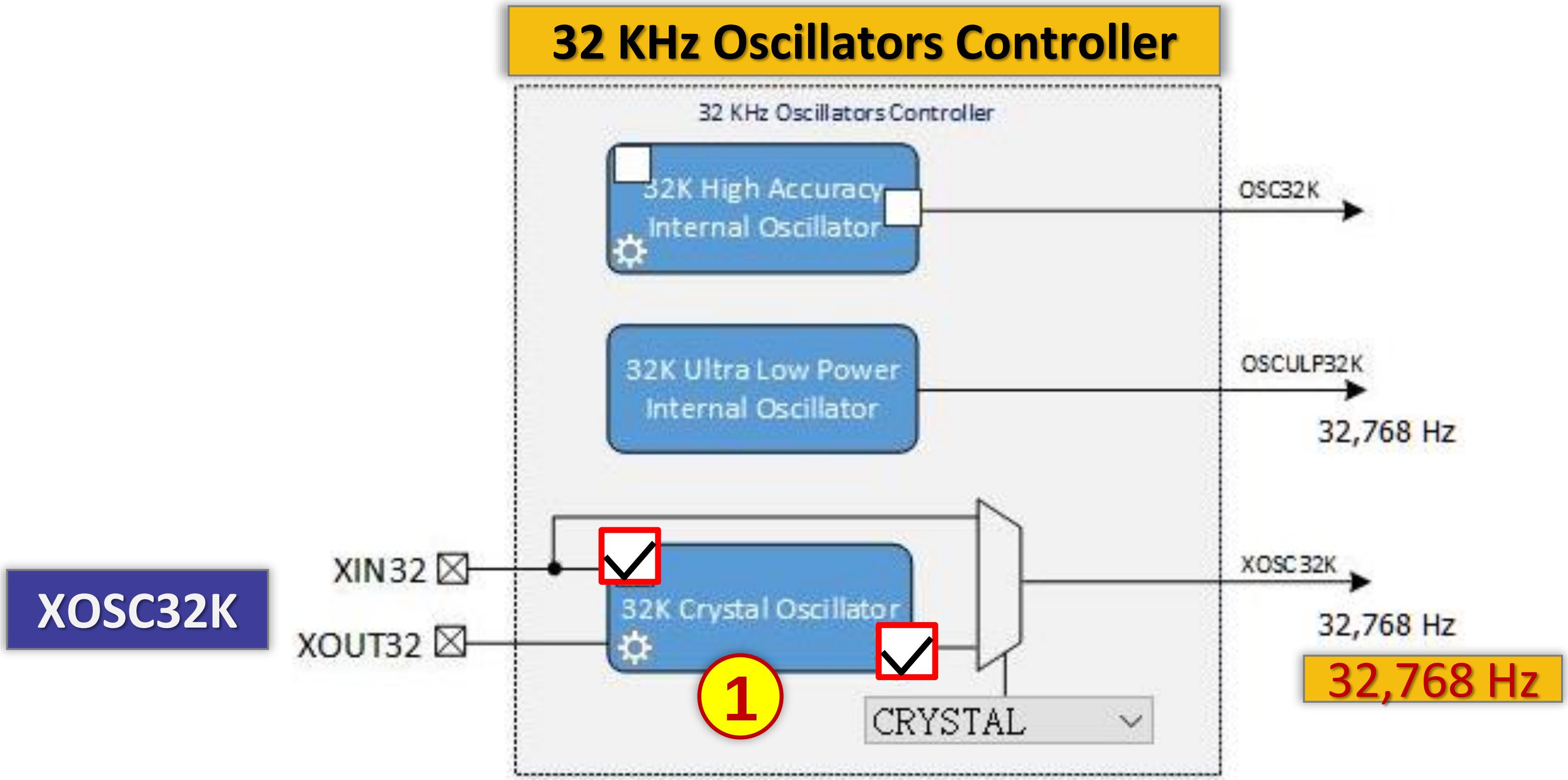
Lab8 System Clock DFLL48M

Setup Steps Overview



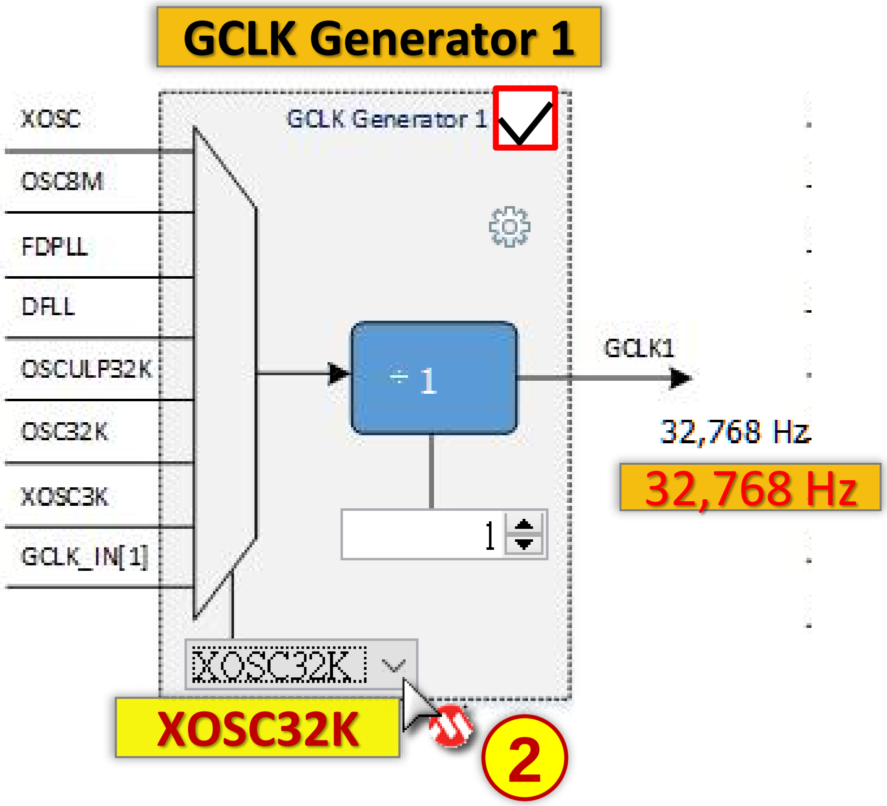
Lab8 System Clock DFLL48M

Step 1



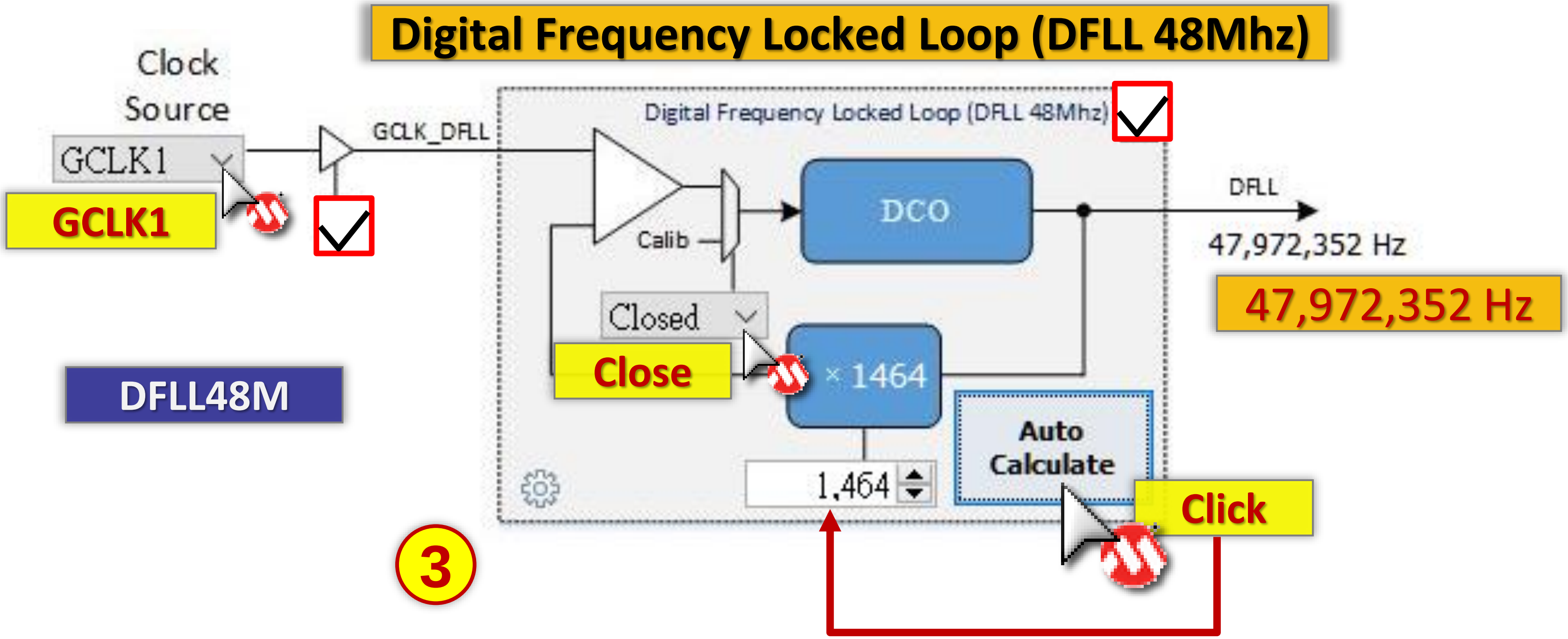
Lab8 System Clock DFLL48M

Step 2



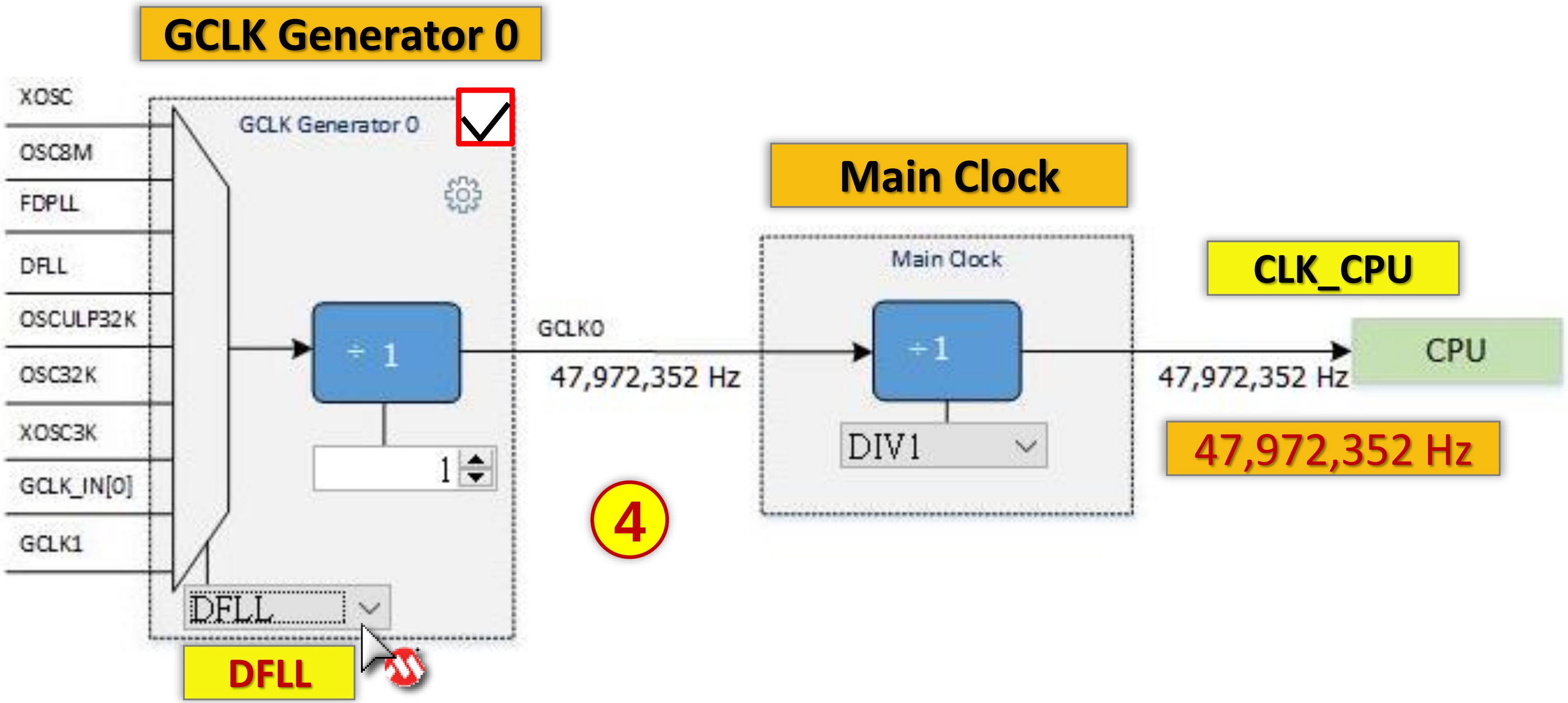
Lab8 System Clock DFLL48M

Step 3



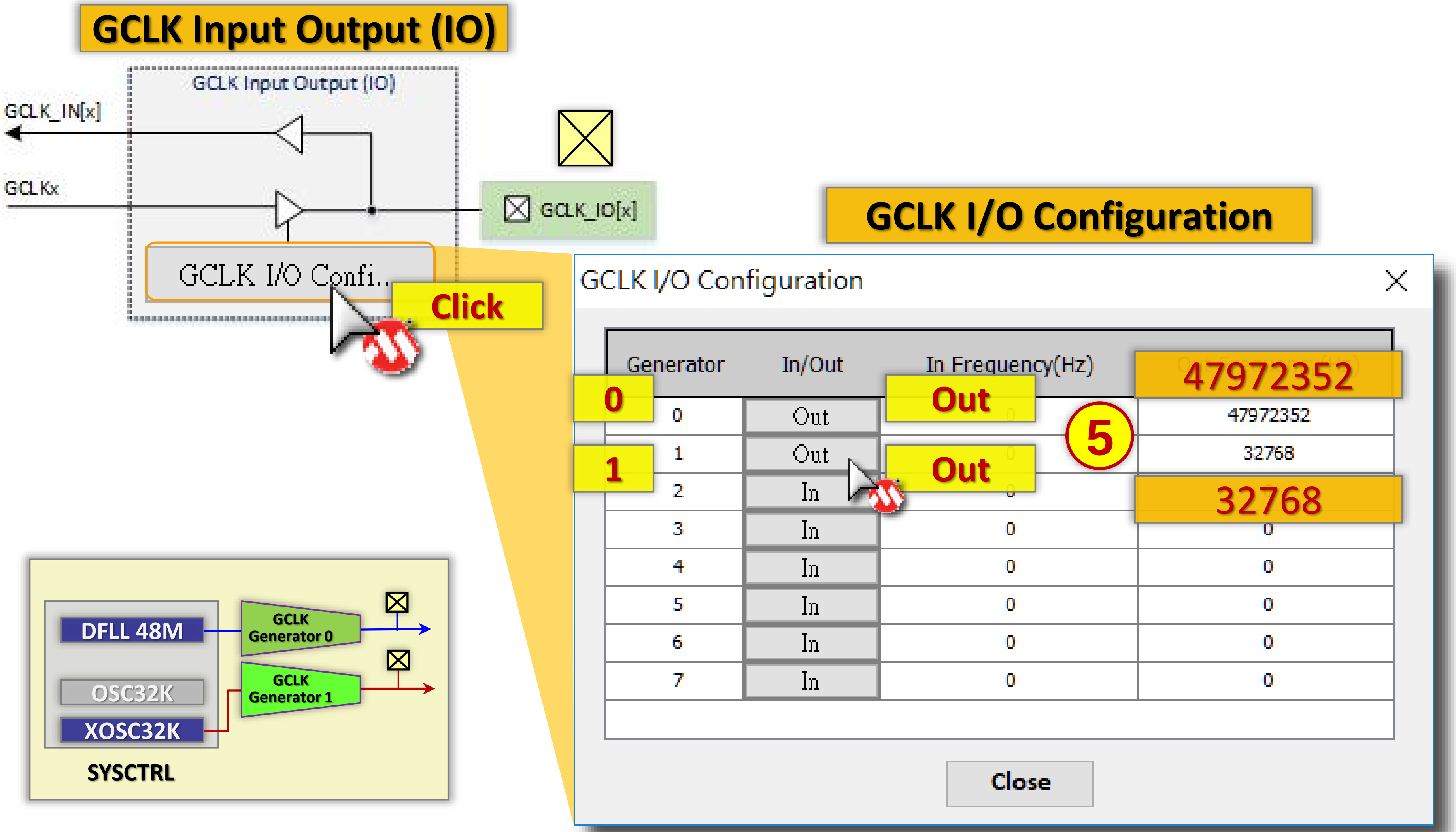
Lab8 System Clock DFLL48M

Step 4



Lab8 System Clock DFLL48M

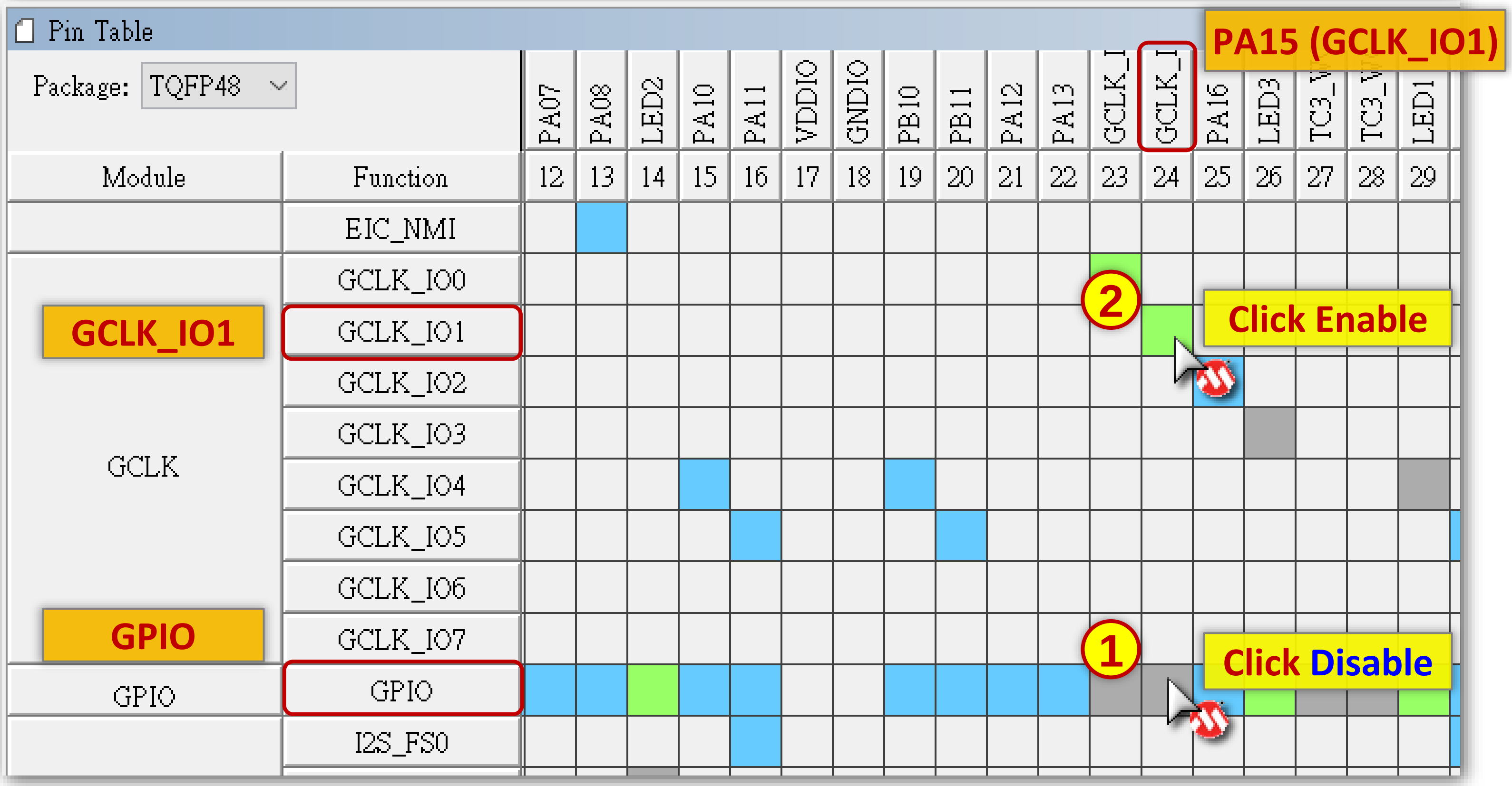
Step 5



Lab8 System Clock DFLL48M

Step 6

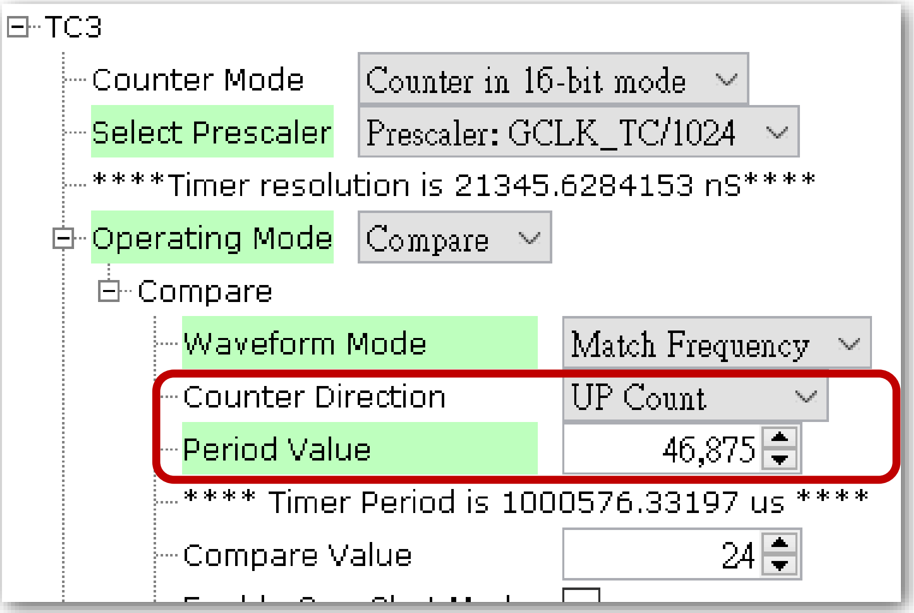
- Please disable BT3(PA15) GPIO pin function firstly and then reassign PA15 as GCLK_IO1 output pin.



Lab8 System Clock DFLL48M

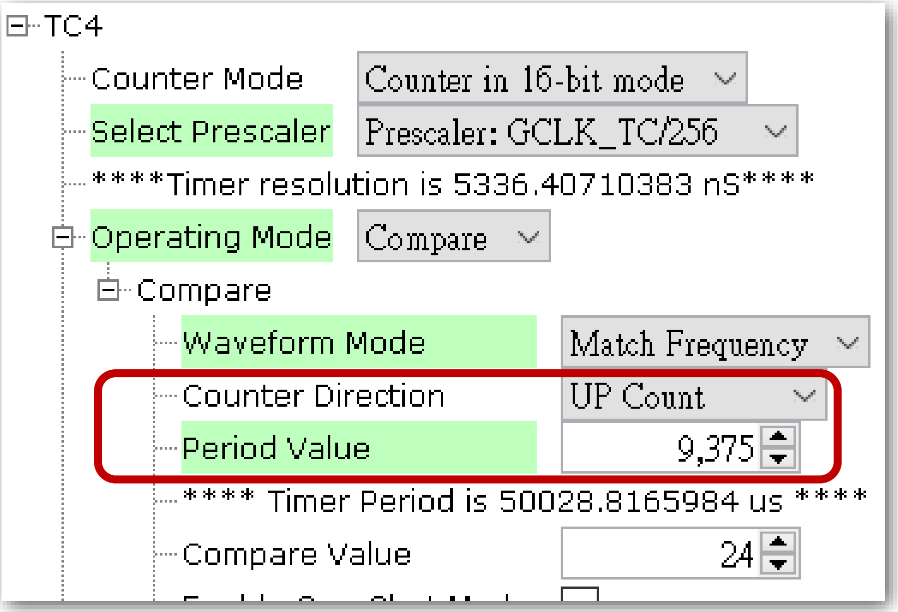
Step 7 LED toggle period correct

TC3 @ 48MHz



Correct Period Value to 46875 to get ≈ 1sec period

TC4 @ 48MHz

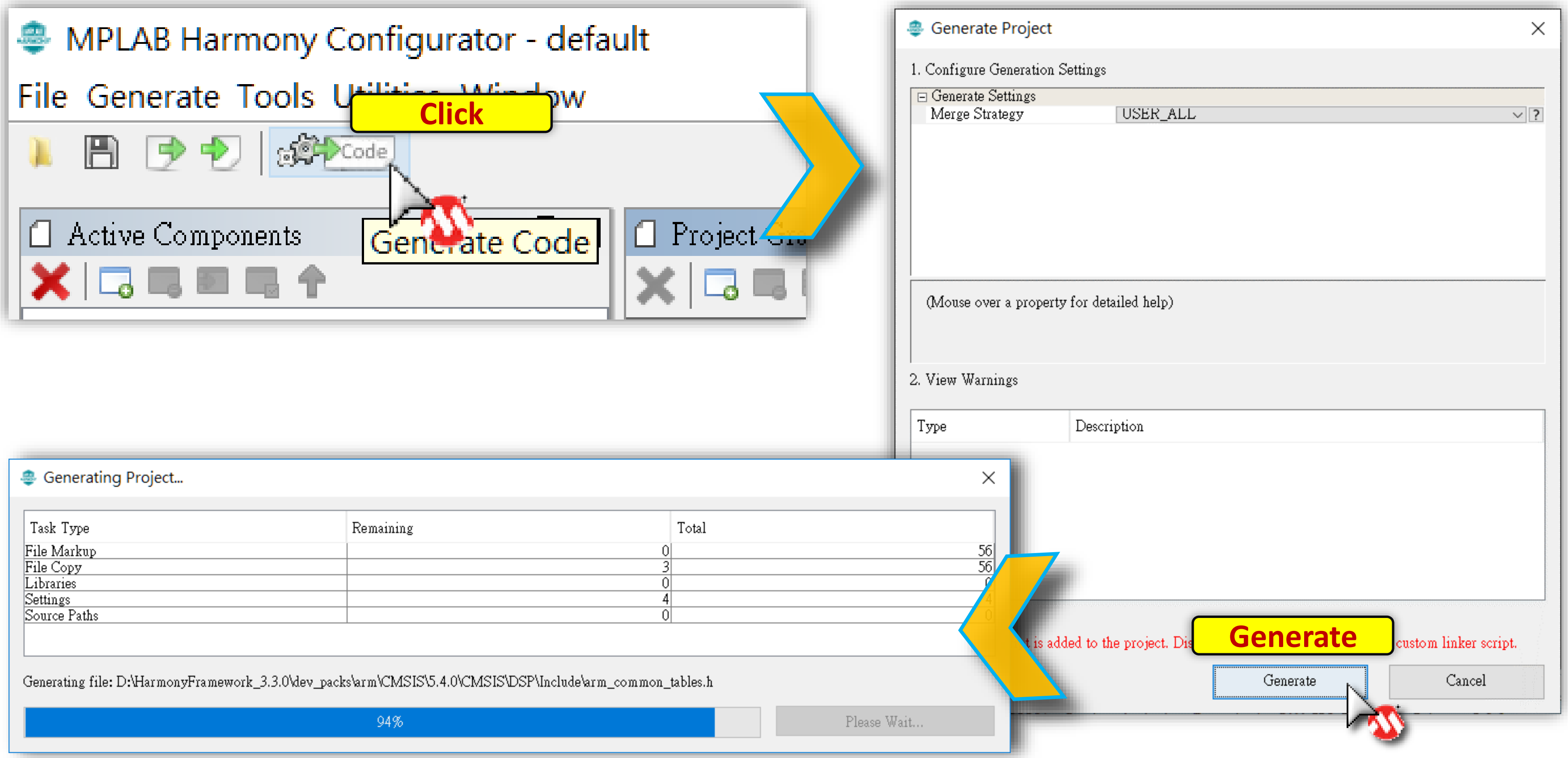


Correct Period Value to 9375 to get ≈ 50ms period

Lab8 System Clock DFLL48M

Step 8

- Click  to Generate Code and save changes to MHC configure file.



MICROCHIP

Flash Access Time and Wait State

- MHC 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;
}
```

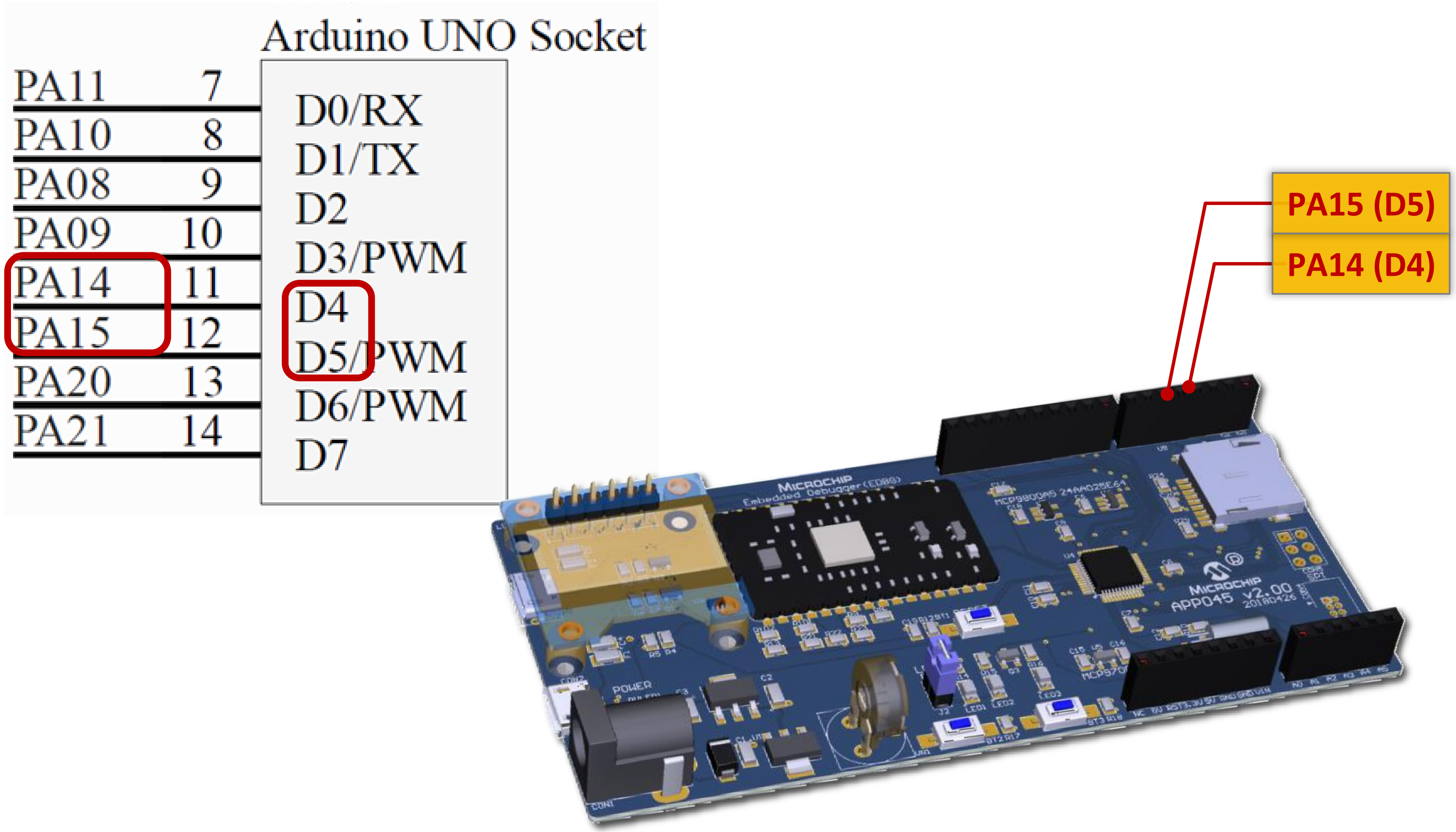
V_{DD} range	NVM Wait States	Maximum Operating Frequency	Units
1.62V to 2.7V	0	14	MHz
	1	28	
	2	42	
	3	48	
2.7V to 3.63V	0	24	
	1	48	

VDD = 3.3V

CPU_CLK = 48MHz

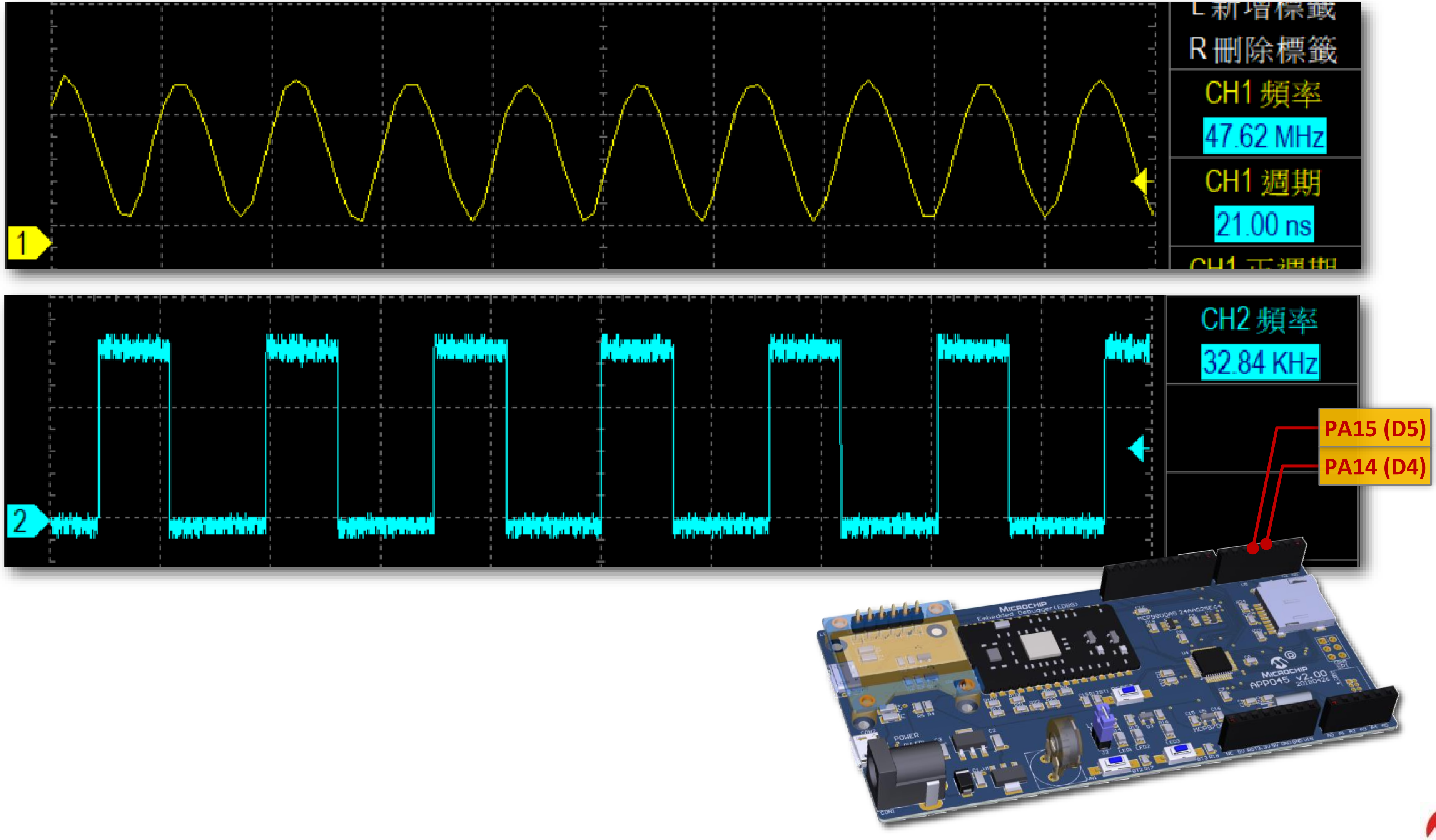
Lab8 System Clock DFLL48M

Clock Output



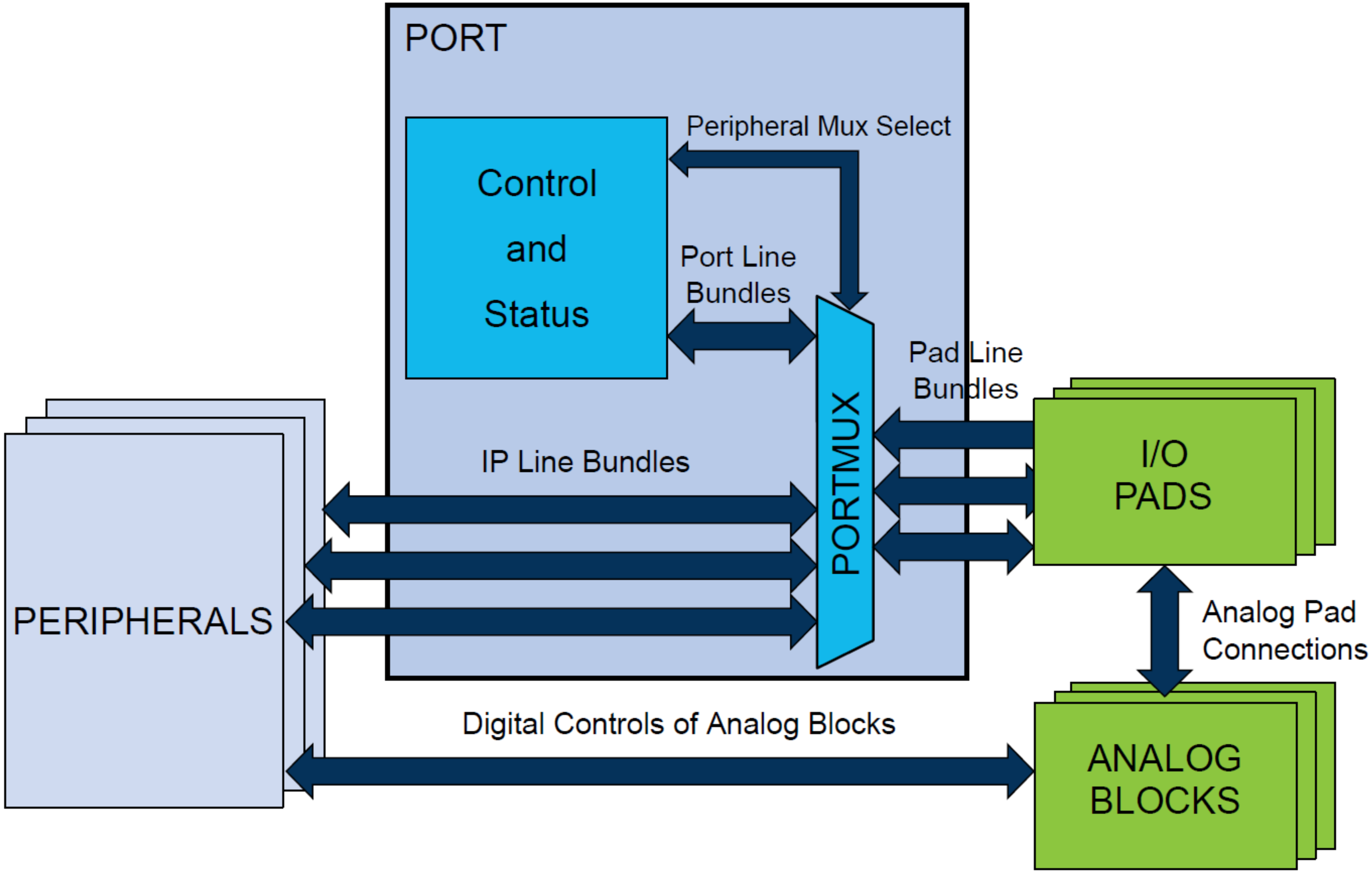
Lab8 System Clock DFLL48M

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]
19	27	37	PA18	VDDIO	EXTINT[2]				X[6]		SERCOM1/ PAD[2]	SERCOM3/ PAD[2]	TC3/WO[0]	TCC0/ WO[2]		AC/CMP[0]
20	28	38	PA19	VDDIO	EXTINT[3]				X[7]		SERCOM1/ PAD[3]	SERCOM3/ PAD[3]	TC3/WO[1]	TCC0/ WO[3]	I2S/SD[0]	AC/CMP[1]
21	31	43	PA22	VDDIO	EXTINT[6]				X[10]		SERCOM3/ PAD[0]	SERCOM5/ PAD[0]	TC4/WO[0]	TCC0/ WO[4]		GCLK_IO[6]
22	32	44	PA23	VDDIO	EXTINT[7]				X[11]		SERCOM3/ PAD[1]	SERCOM5/ PAD[1]	TC4/WO[1]	TCC0/ WO[5]	USB/SOF 1kHz	GCLK_IO[7]

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

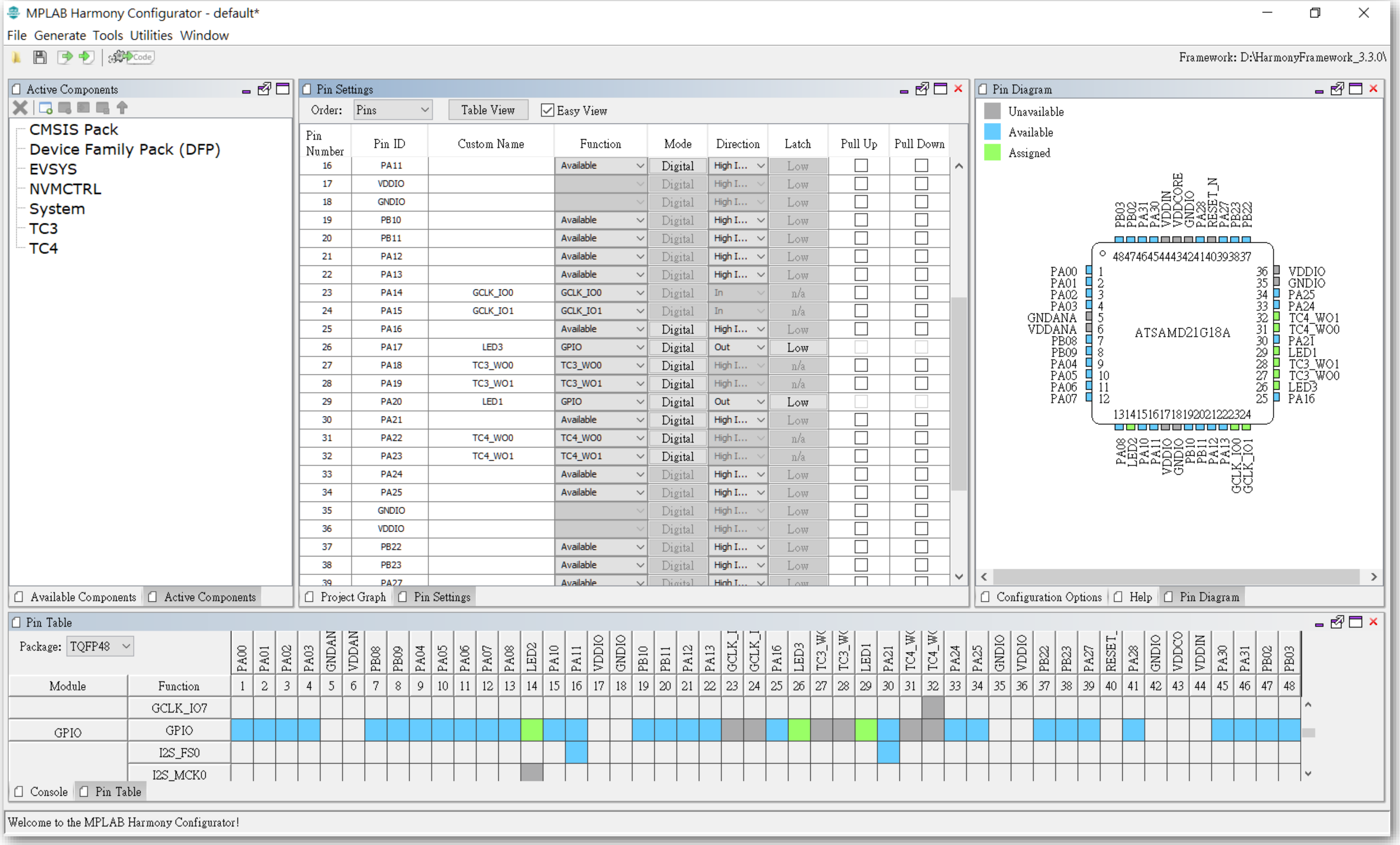
PORT_Initialize() @ plib_port.c

```
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_PINCFG[18] = 0x1;
PORT_REGS->GROUP[0].PORT_PINCFG[19] = 0x1;
PORT_REGS->GROUP[0].PORT_PINCFG[22] = 0x1;
PORT_REGS->GROUP[0].PORT_PINCFG[23] = 0x1;

PORT_REGS->GROUP[0].PORT_PMUX[7] = 0x77;
PORT_REGS->GROUP[0].PORT_PMUX[9] = 0x44;
PORT_REGS->GROUP[0].PORT_PMUX[11] = 0x44;
```

PA14 (GCLK_IO0) — **INEN = 1 (Button)**
PA15 (GCLK_IO1) — **INEN = 1 (Button)**
PA18 (TC3_WO0)
PA19 (TC3_WO1)
PA22 (TC4_WO0)
PA23 (TC4_WO1)

PMUX[7] : PA15 , PA14
PMUX[9] : PA19 , PA18
PMUX[11] : PA23 , PA22

PINMUX Configuration Example



MICROCHIP

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)	1	PA14	VDDIO	EXTINT[14]						SERCOM2/ PAD[2]	SERCOM4/ PAD[2]	TC3/WO[0]	GCLK_IO[0]		GCLK_IO[0]
16	PA15 (7)	2	PA15	VDDIO	EXTINT[15]						SERCOM2/ PAD[3]	SERCOM4/ PAD[3]	TC3/WO[1]	GCLK_IO[1]		GCLK_IO[1]
19	PA18 (4)	7	PA18	VDDIO	EXTINT[2]				X[6]		TC3/WO[0]	3/	TC3/WO[0]	TCC0/ WO[2]		AC/CMP[0]
20	PA19 (4)	8	PA19	VDDIO	EXTINT[3]				X[7]		TC3/WO[1]	3/	TC3/WO[1]	TCC0/ WO[3]	I2S/SD[0]	AC/CMP[1]
21	PA22 (4)	3	PA22	VDDIO	EXTINT[6]				X[10]		TC4/WO[0]	5/	TC4/WO[0]	TCC0/ WO[4]		GCLK_IO[6]
22	PA23 (4)	4	PA23	VDDIO	EXTINT[7]				X[11]		TC4/WO[1]	5/	TC4/WO[1]	TCC0/ WO[5]	USB/SOF 1kHz	GCLK_IO[7]

```
PORT_REGS->GROUP[0].PORT_PMUX[7] = 0x77;
PORT_REGS->GROUP[0].PORT_PMUX[9] = 0x44;
PORT_REGS->GROUP[0].PORT_PMUX[11] = 0x44;
```

PA15 (7) , PA14 (7)

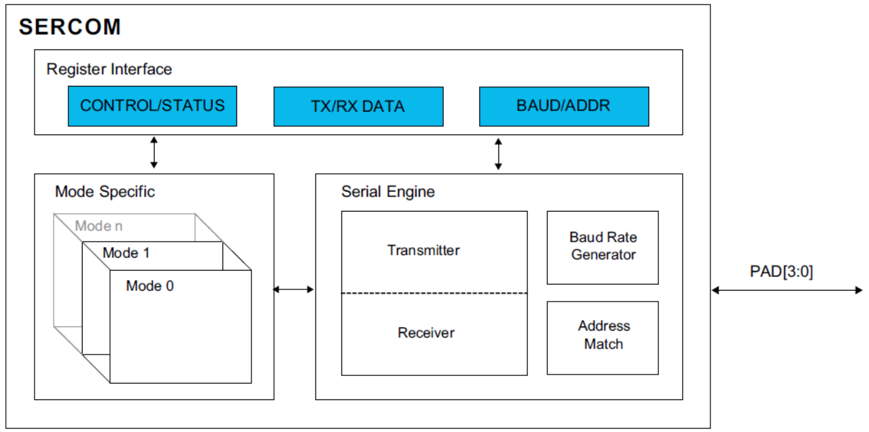
PA19 (4) , PA18 (4)

PA23 (4) , PA22 (4)

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.**

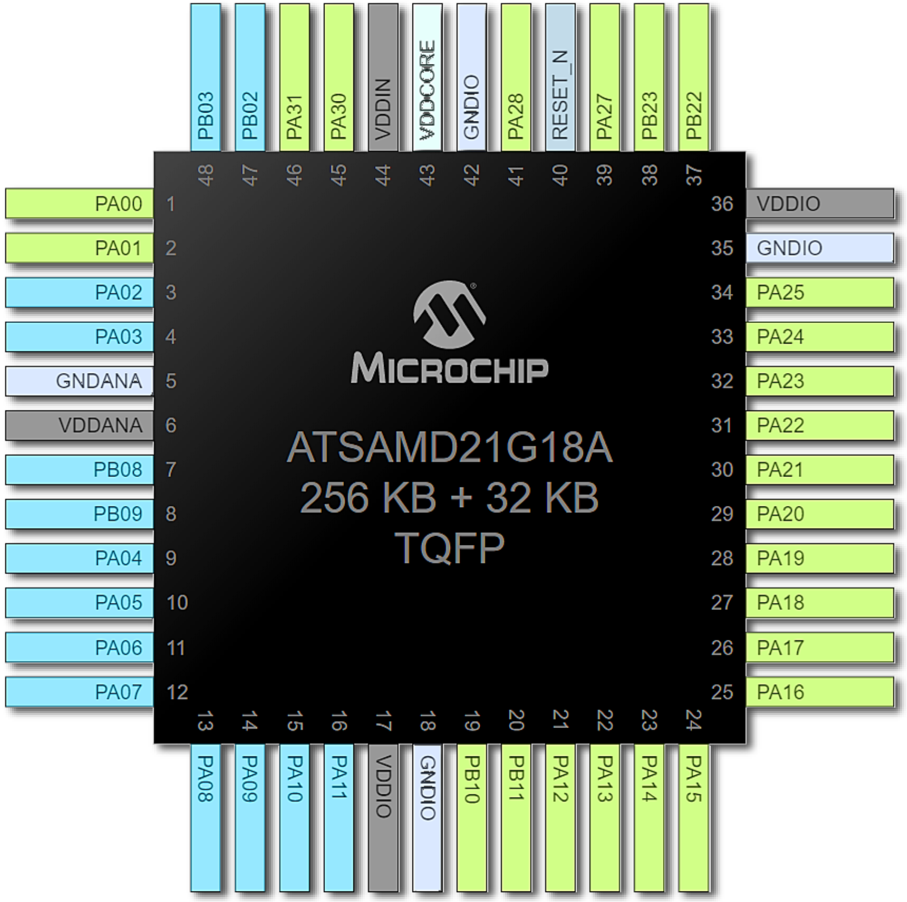


- **The SERCOM serial engine consists of a transmitter and receiver, baud-rate generator and address matching functionality. It can use the internal generic clock or an external clock to operate in all sleep.**

SERCOM Introduction

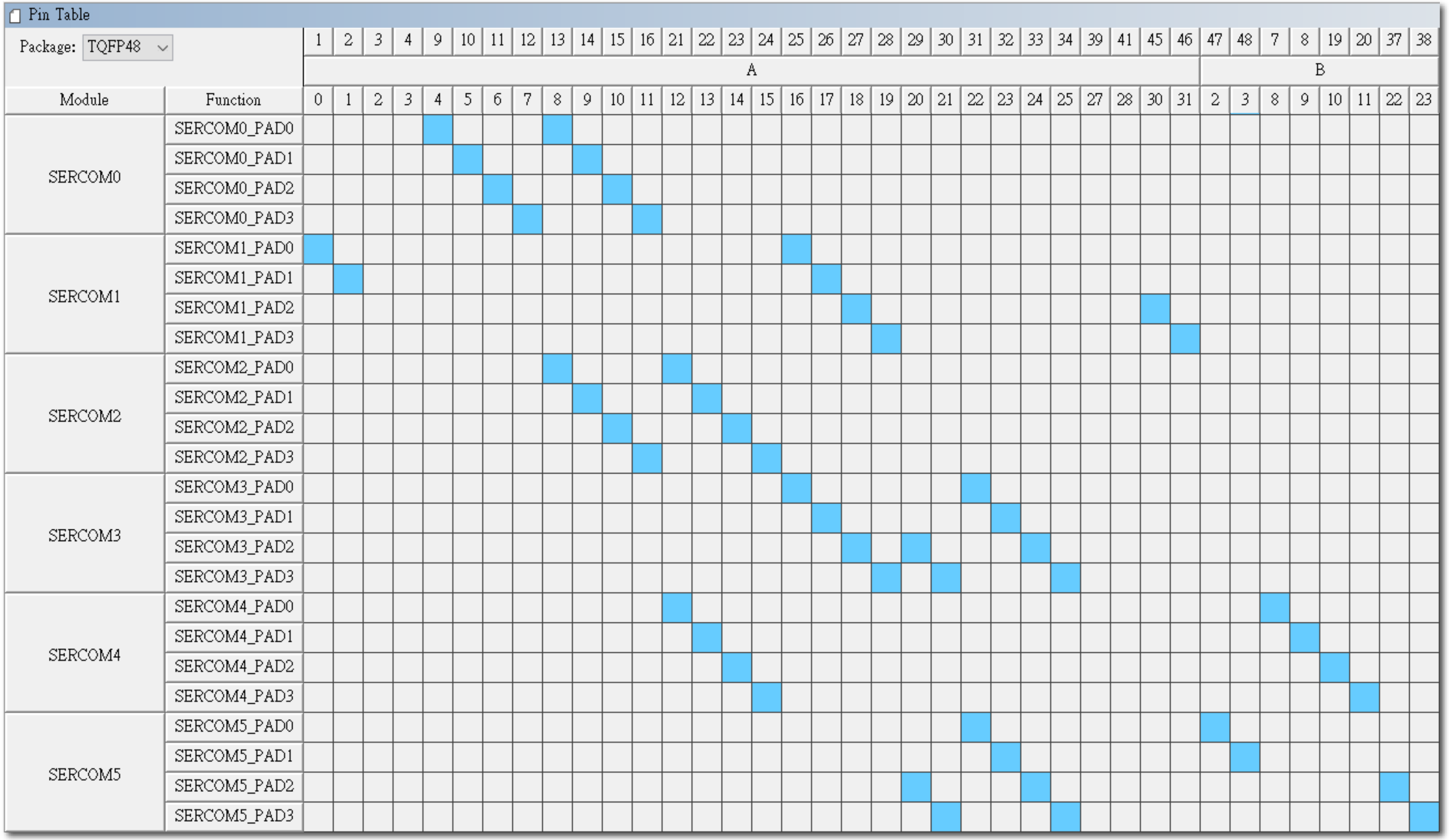
- **Pin Multiplex**

	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)**

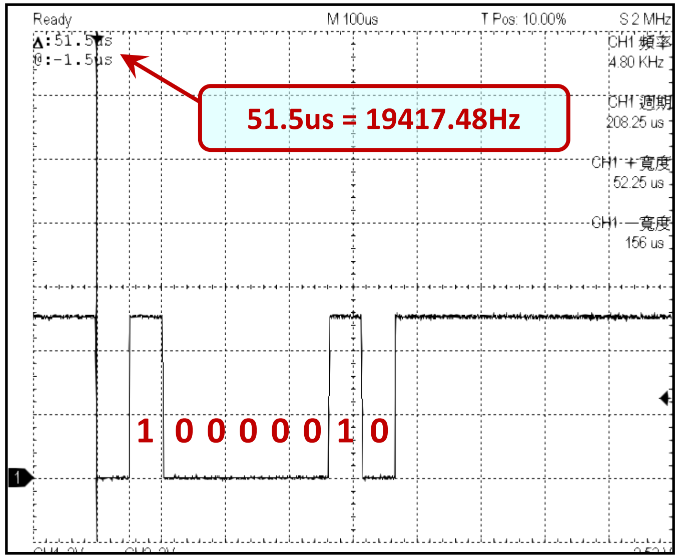


SERCOM Introduction

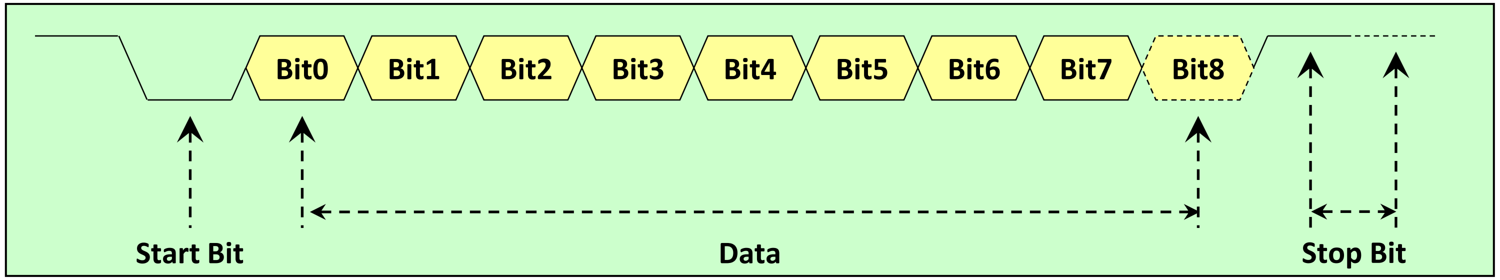
- **For I2C**
  - Fast Mode (400kHz)
  - Master or slave operation
  - SMBUS compatible
  - Wake on address match
- **For SPI**
  - Up to 24Mb/s
  - Supports all four SPI modes
  - Full-duplex operation
  - Single data direction mode frees unused MISO or MOSI pin
- **For USART**
  - Up to 24Mb/s
  -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

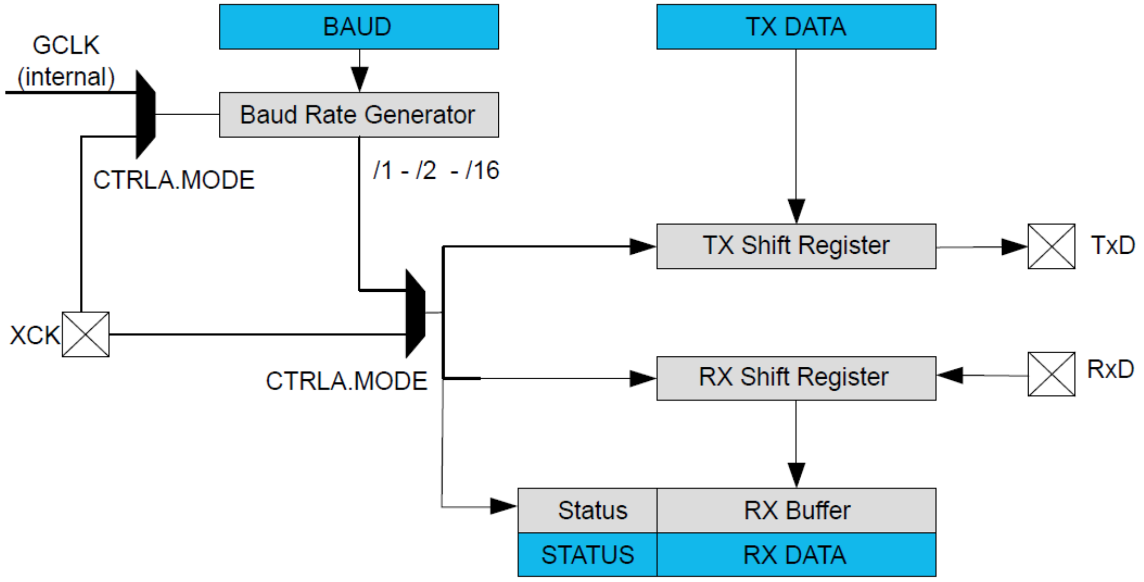


UART Transmit Example :
'A'(0x41), 19200 bps



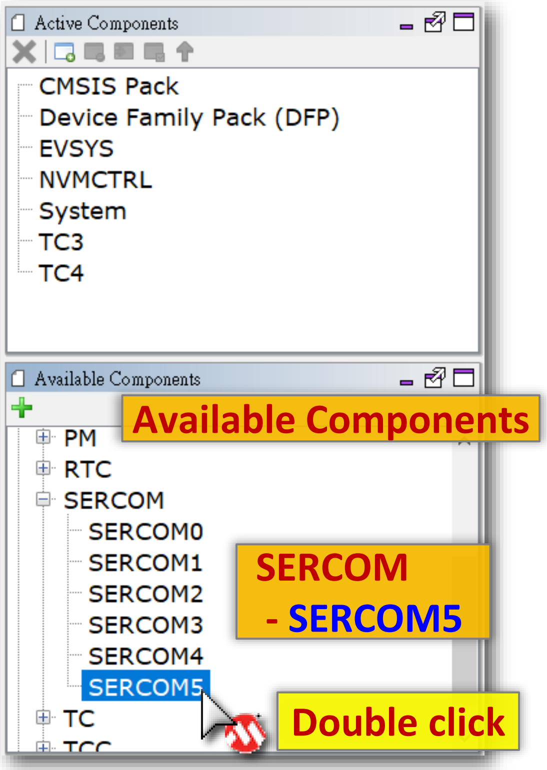
SERCOM - USART

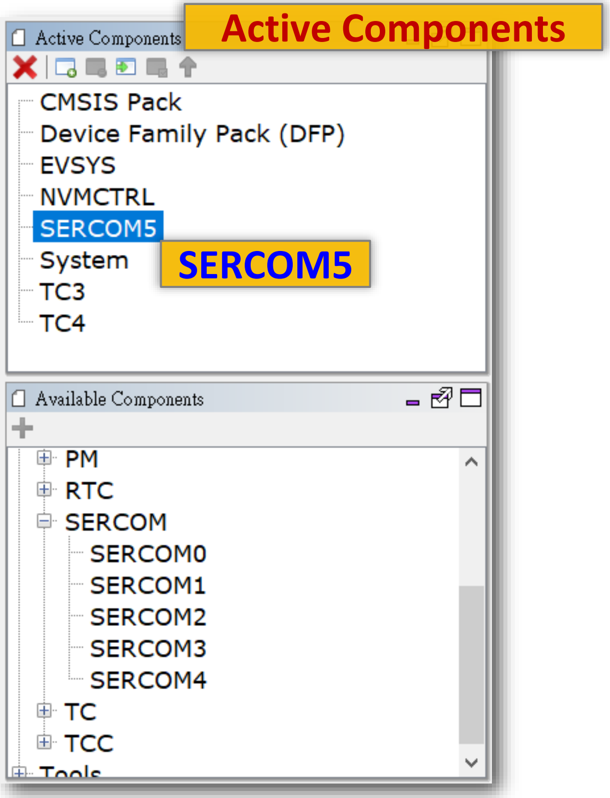
- Supports serial frames with 5, 6, 7, 8 or 9 data bits and 1 or 2 stop bits, Parity Check, RTS and CTS flow control.
- Full-duplex operation, RS-232, RS-485 and IrDA Supported.
- Buffer overflow and frame error detection.
- Internal or external clock source for asynchronous and synchronous operation.



Add SERCOM Function using MHC

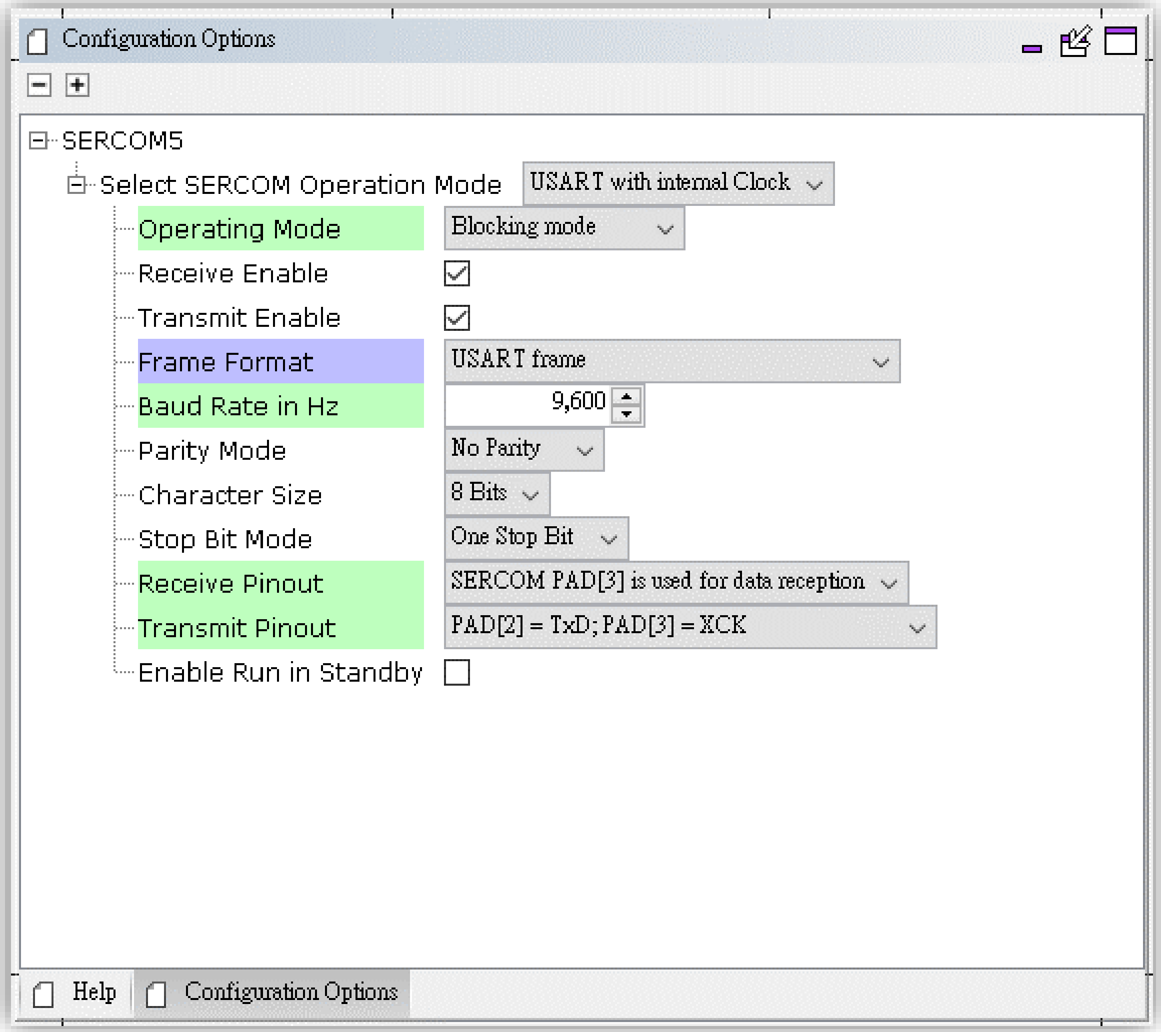
- Find SERCOM component in Available Components window.
- Double click SERCOM5 to add to Active Components window.



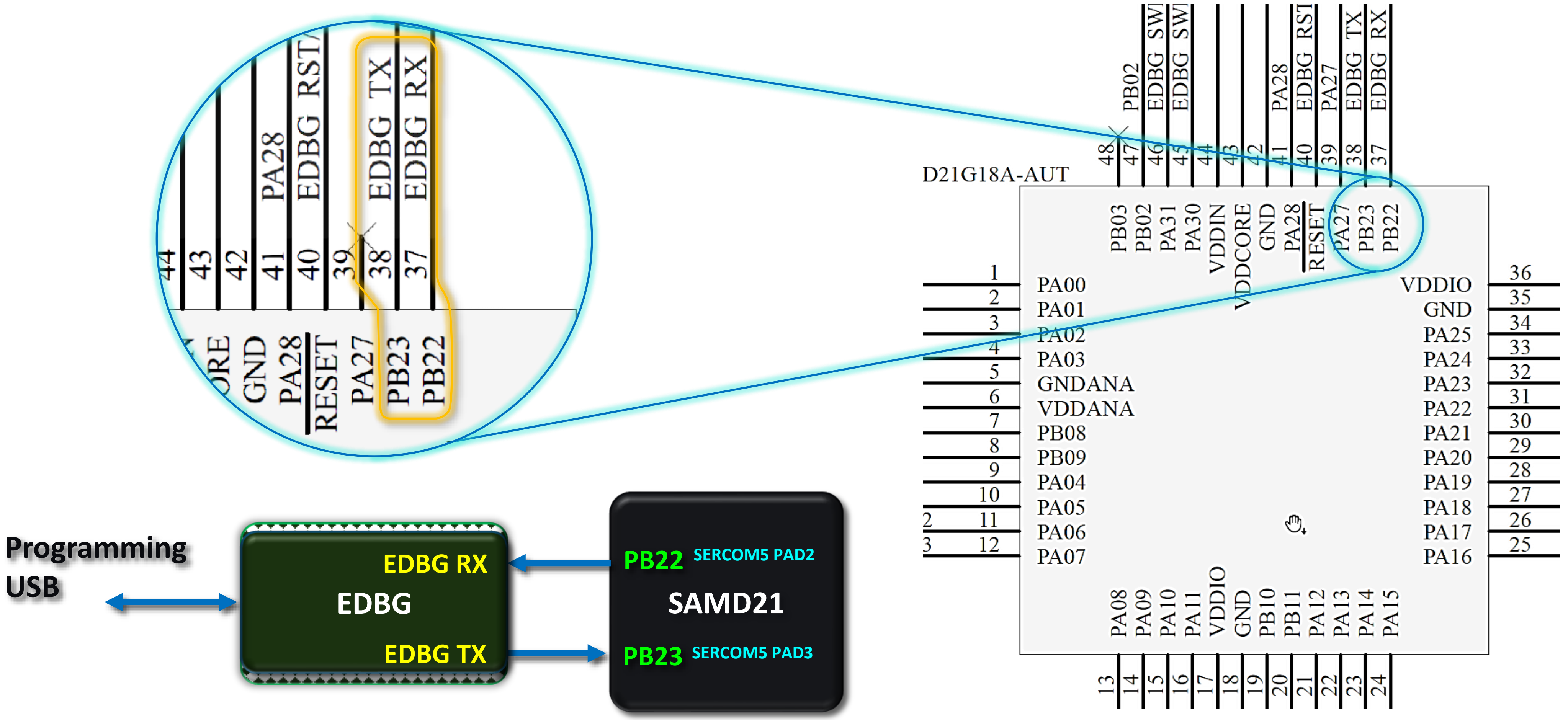


SERCOM USART Polling

Configuration Options Example



APP045 SERCOM USART Schematic



SERCOM USART

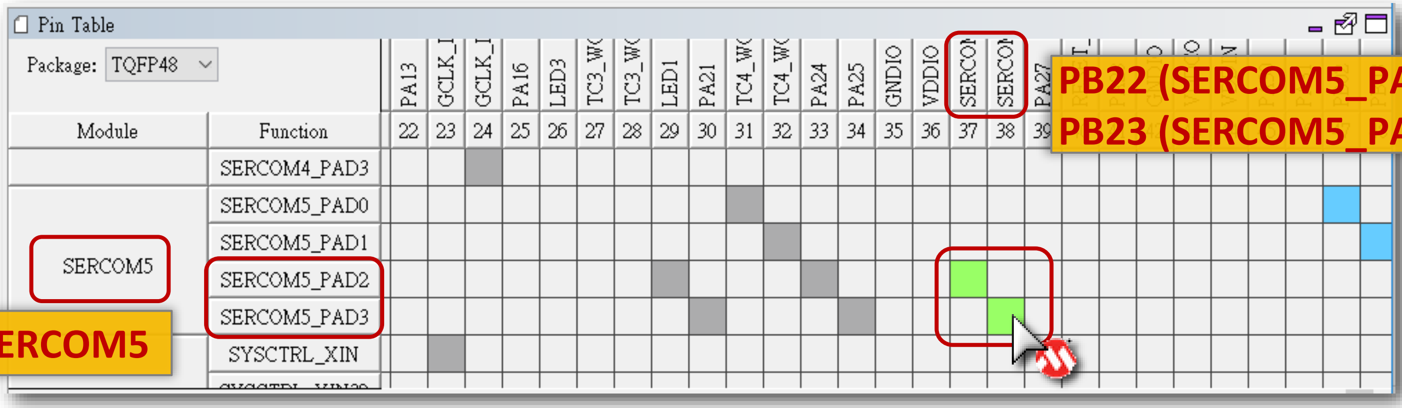
PINMUX Configuration Example

- **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]	TC7/WO[0]	TCC3/		GCLK_IO[0]
	38	[illegible]	PB23	VDDIO	EXTINT[7]							SERCOM5/ PAD[3]	T[illegible]	WO[1]		GCLK_IO[1]

PB22
PB23

SERCOM5/PAD[2]
SERCOM5/PAD[3]



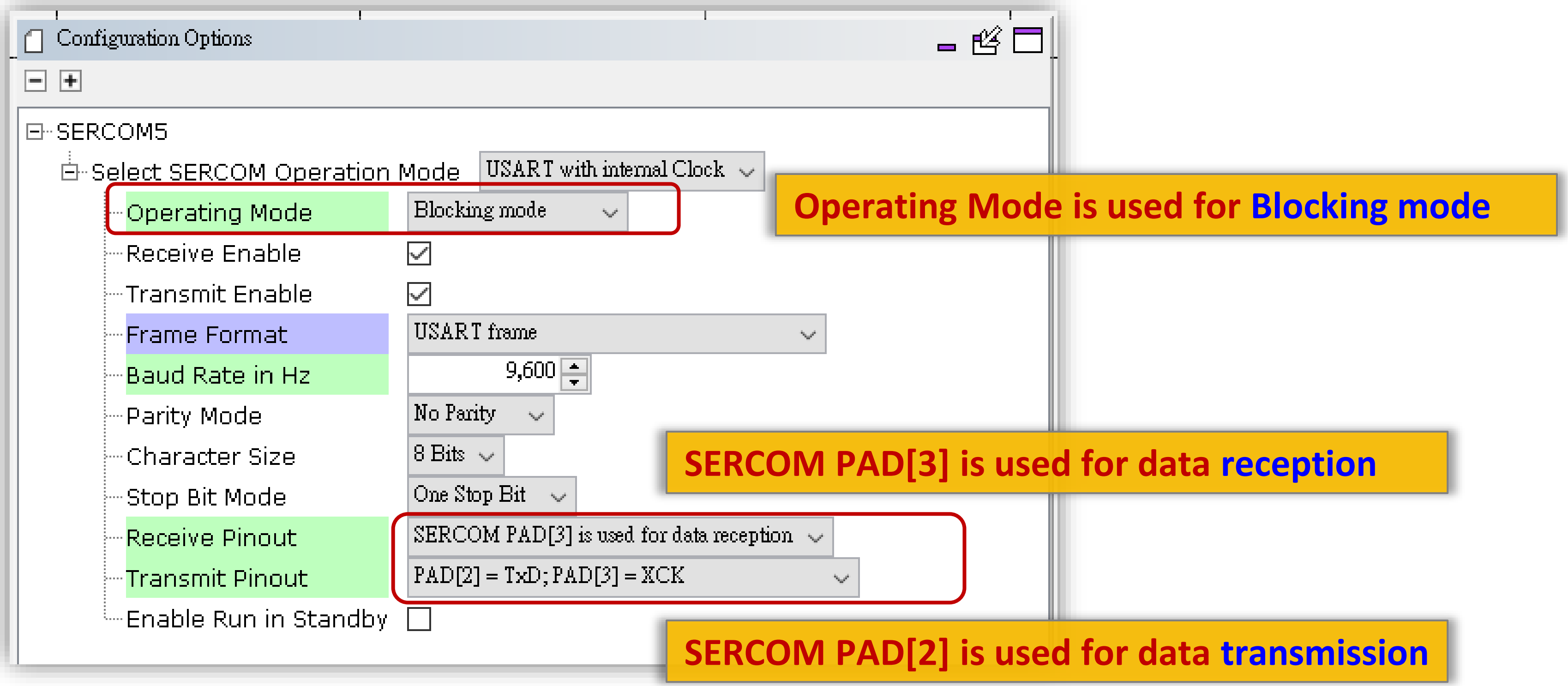
PB22 (SERCOM5_PAD2)
PB23 (SERCOM5_PAD3)

SERCOM5

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)



SERCOM USART Polling

Interface Routines

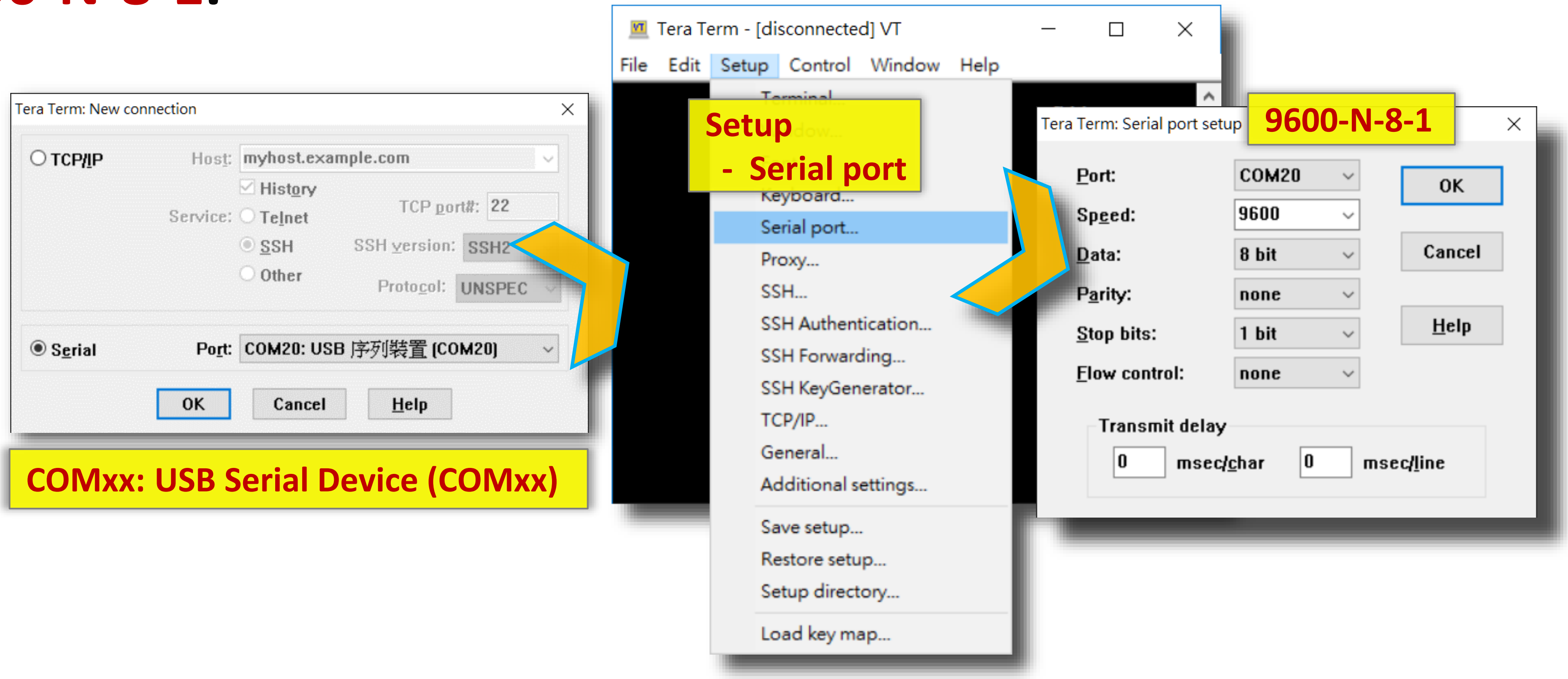
```
void SERCOM5_USART_Initialize( void );
bool SERCOM5_USART_SerialSetup( USART_SERIAL_SETUP * serialSetup,
                                uint32_t clkFrequency );
bool SERCOM5_USART_Write( void *buffer, const size_t size );
bool SERCOM5_USART_TransmitterIsReady( void );
bool SERCOM5_USART_TransmitComplete( 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(1)
   {
      …
      SERCOM5_USART_Write( "Hello world!\r\n", 14 );
   }
}
```

Terminal Console Software

- Tera Term is a terminal console software. (http://ttssh2.sourceforge.jp/index.html.en)
- Select COMxx: USB Serial Device (COMxx)
- Set 9600-N-8-1.



MICROCHIP

Lab9 SERCOM - UART Tx Polling

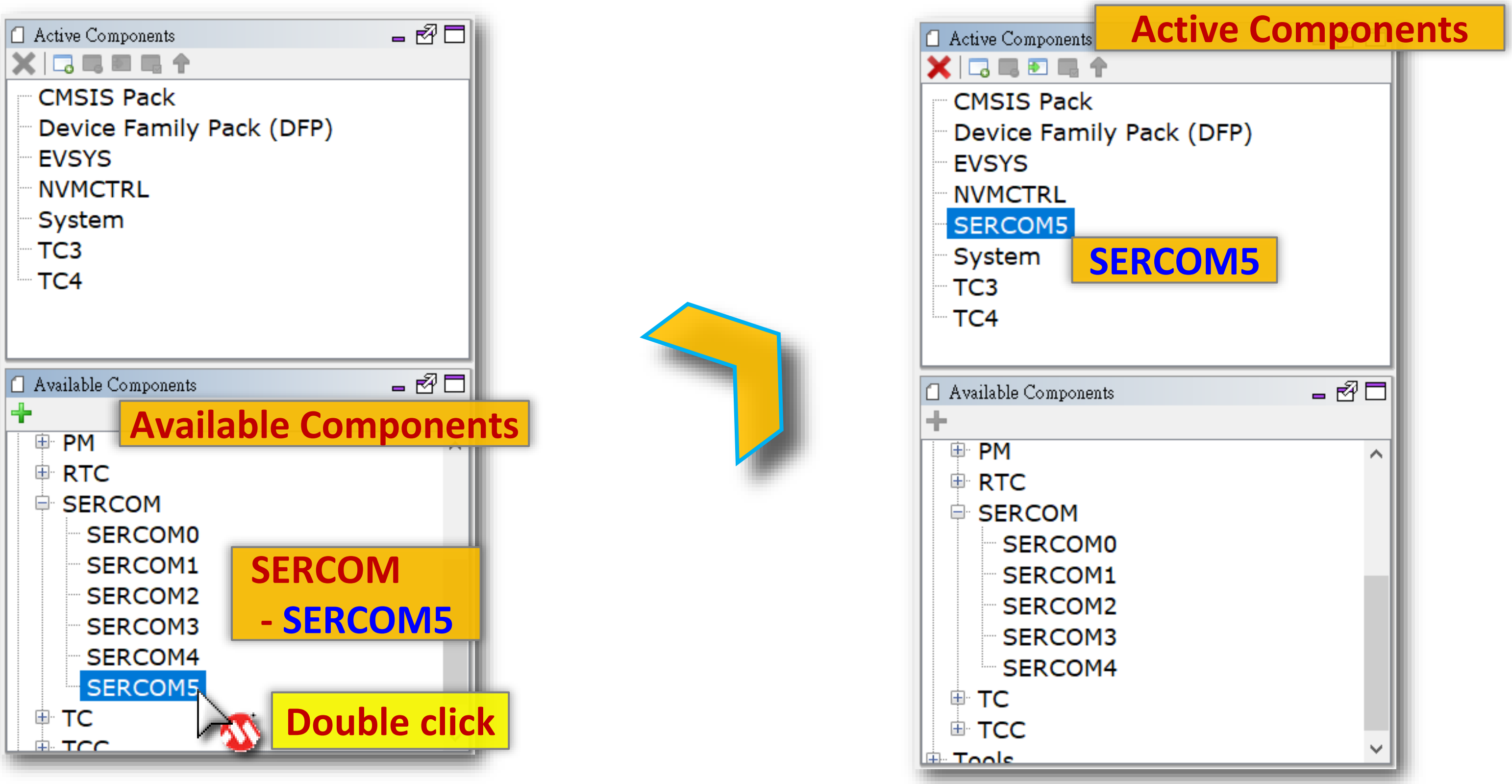
- Try to add SERCOM-USART function to project.
- UART set to 9600-N-8-1, No flow control.
- Send string from UART to PC of one second interval repeated. (Hello World!)
- Character \r for carriage Return (0x0D)
- Character \n for New line (0x0A)

- **Let's go！**

Lab9 SERCOM - UART Tx Polling

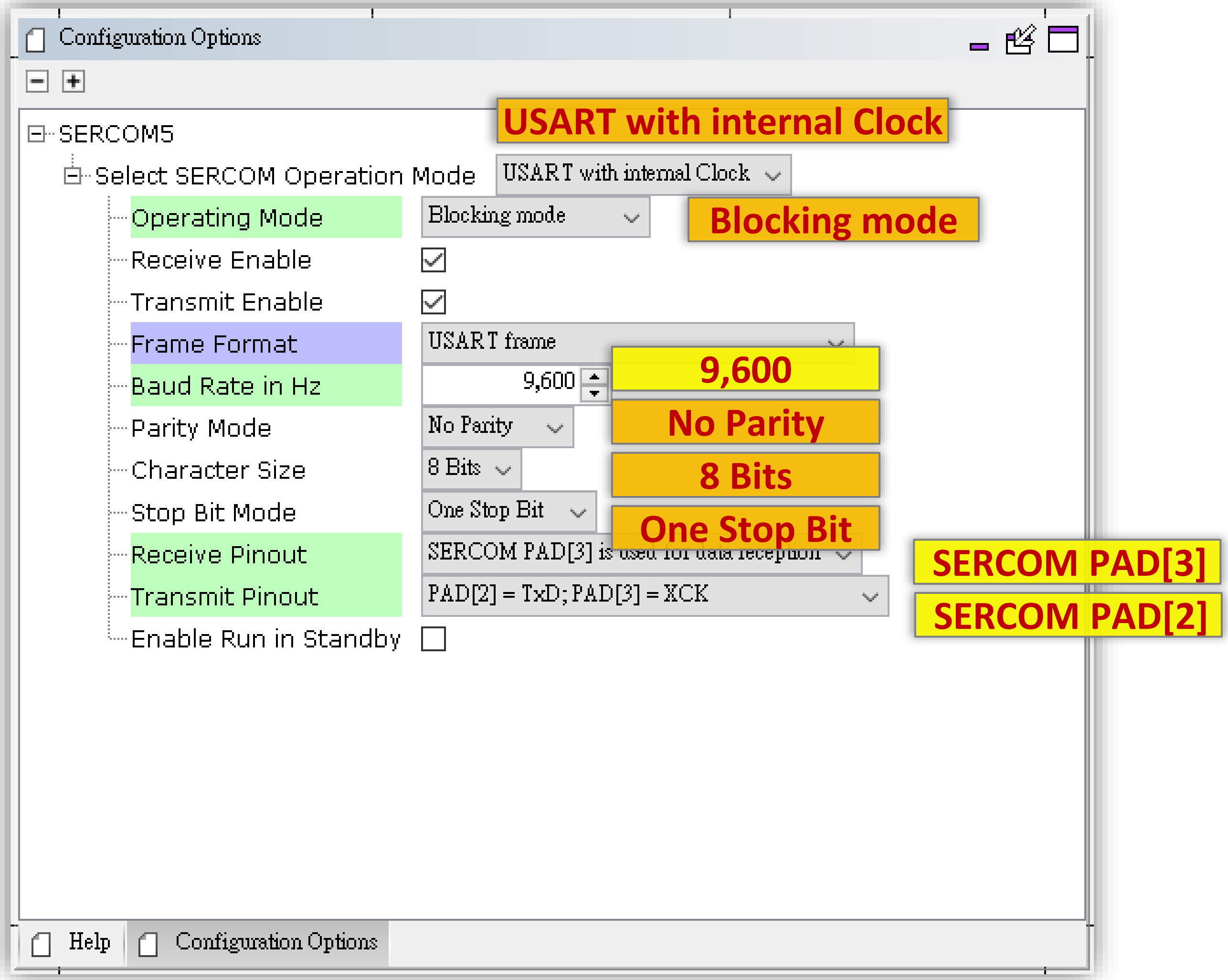
Step 0

- Find **SERCOM** component in Available Components window.
- Double click **SERCOM5** to add to Active Components window.



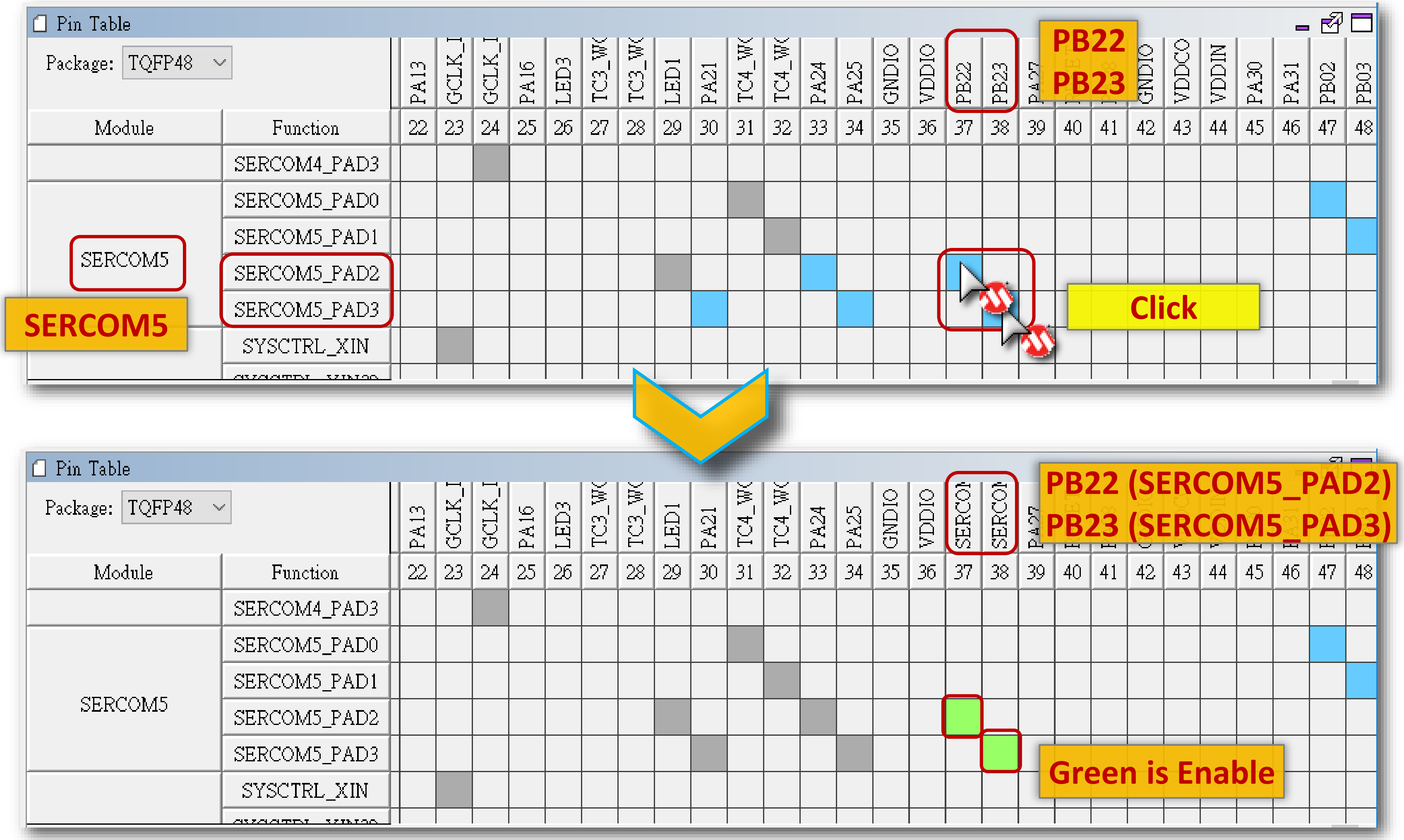
Lab9 SERCOM - UART Tx Polling

Step 1



Lab9 SERCOM - UART Tx Polling

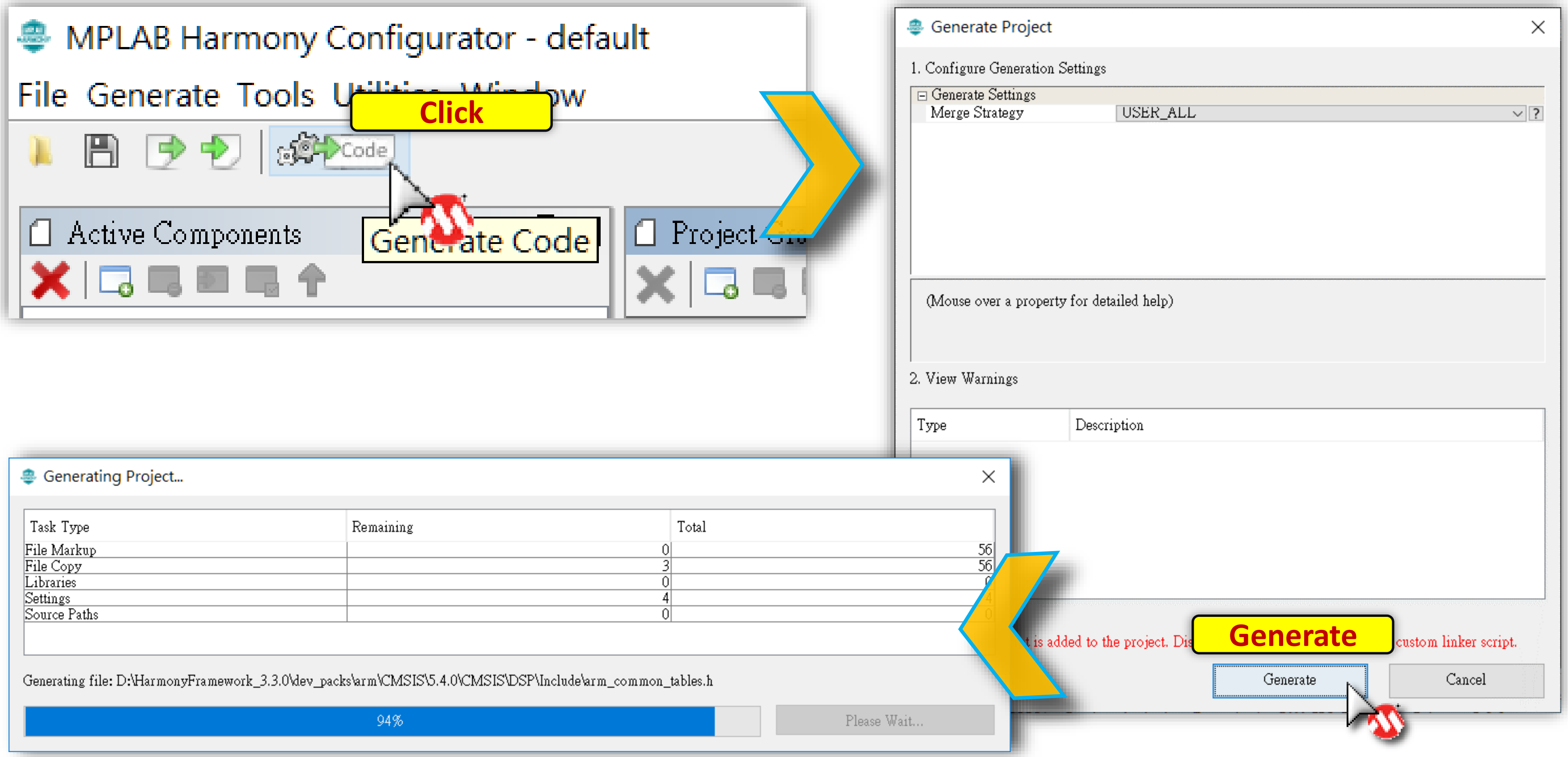
Step 2



Lab9 SERCOM - UART Tx Polling

Step 3

- **Click  to Generate Code and save changes to MHC configure file.**



Lab9 SERCOM - UART Tx Polling

Step 4

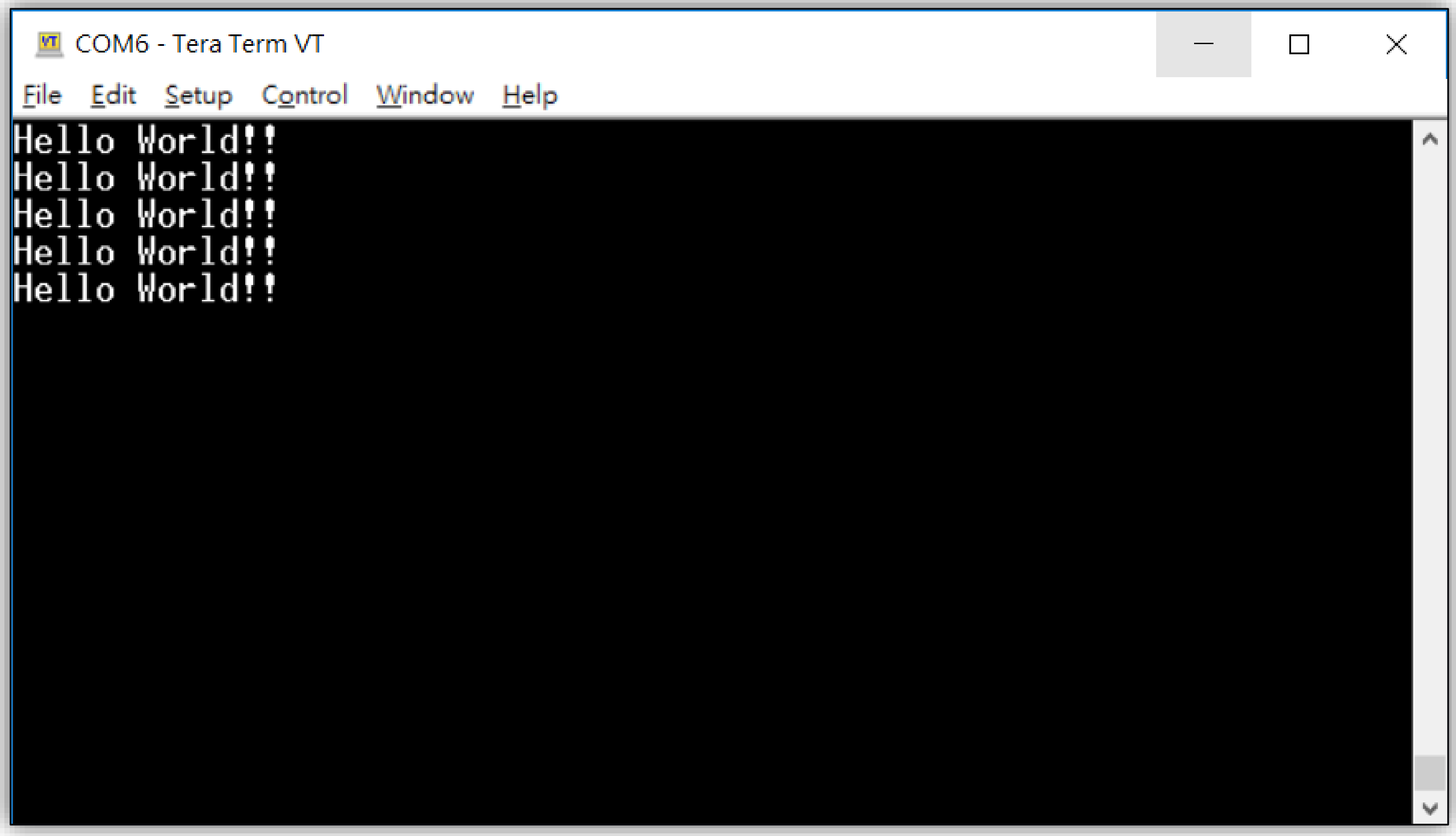
a **Add code segment to your main loop**

```
void TC3_Overflow(TC_COMPARE_STATUS status, uintptr_t context)
{
        if( status & TC_INTFLAG_OVF_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

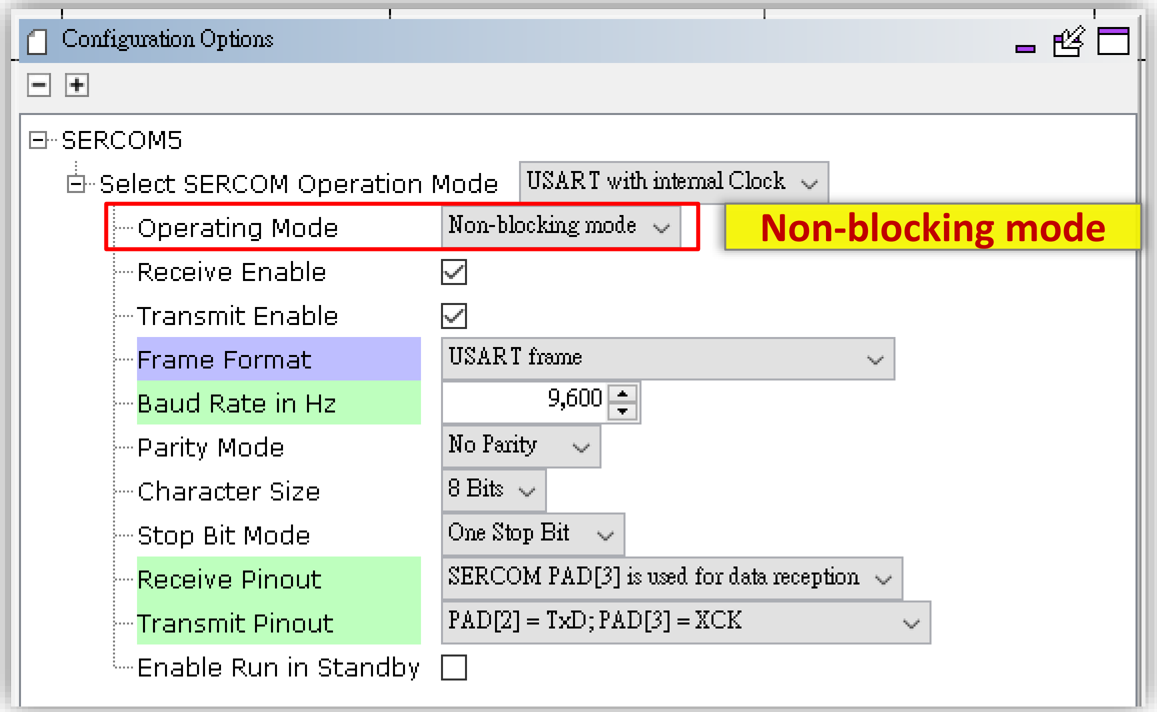
Result

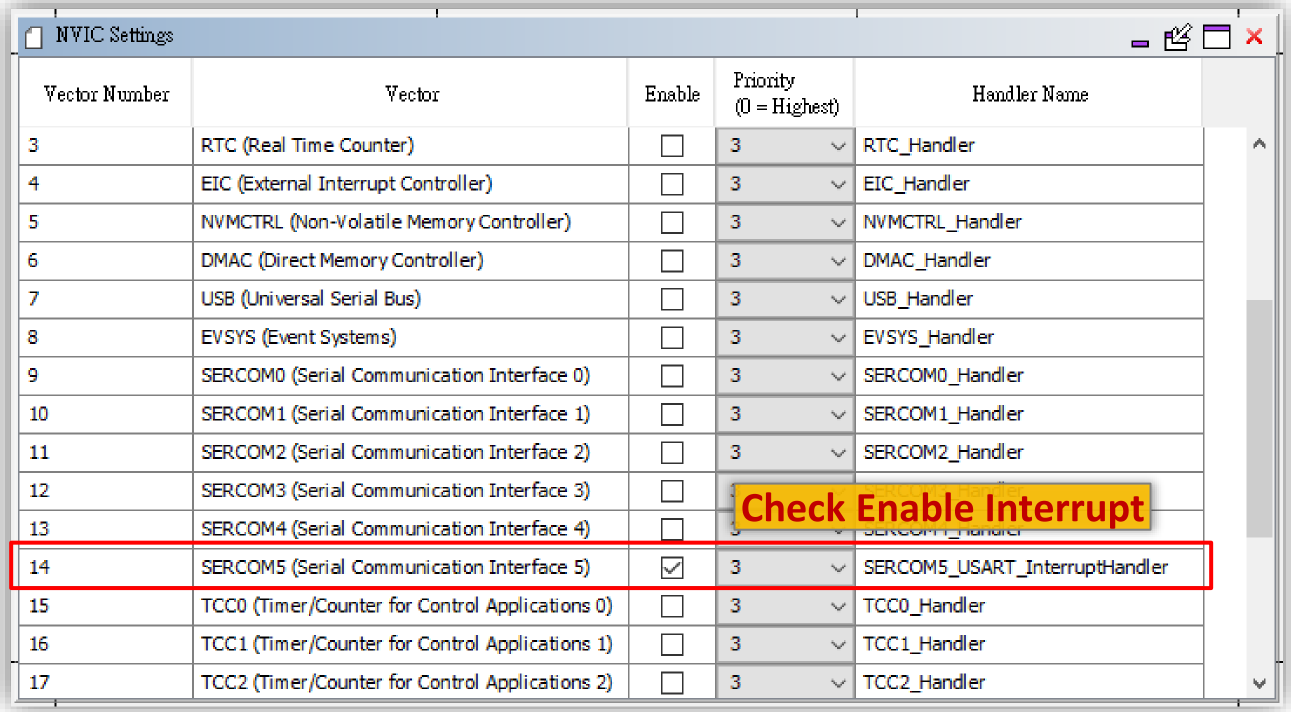


SERCOM USART Interrupt

Configuration Options Example

- **SERCOM provide callback(interrupt) mode, also.**





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)	☐	[illegible]	[illegible]
13	SERCOM4 (Serial Communication Interface 4)	☐	[illegible]	[illegible]
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(1)
   { …
     SERCOM5_USART_Write( "Hello world!\r\n", 14 );
   }
}
```

One-Shot Event

Callback Register

MICROCHIP

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));
```

MICROCHIP

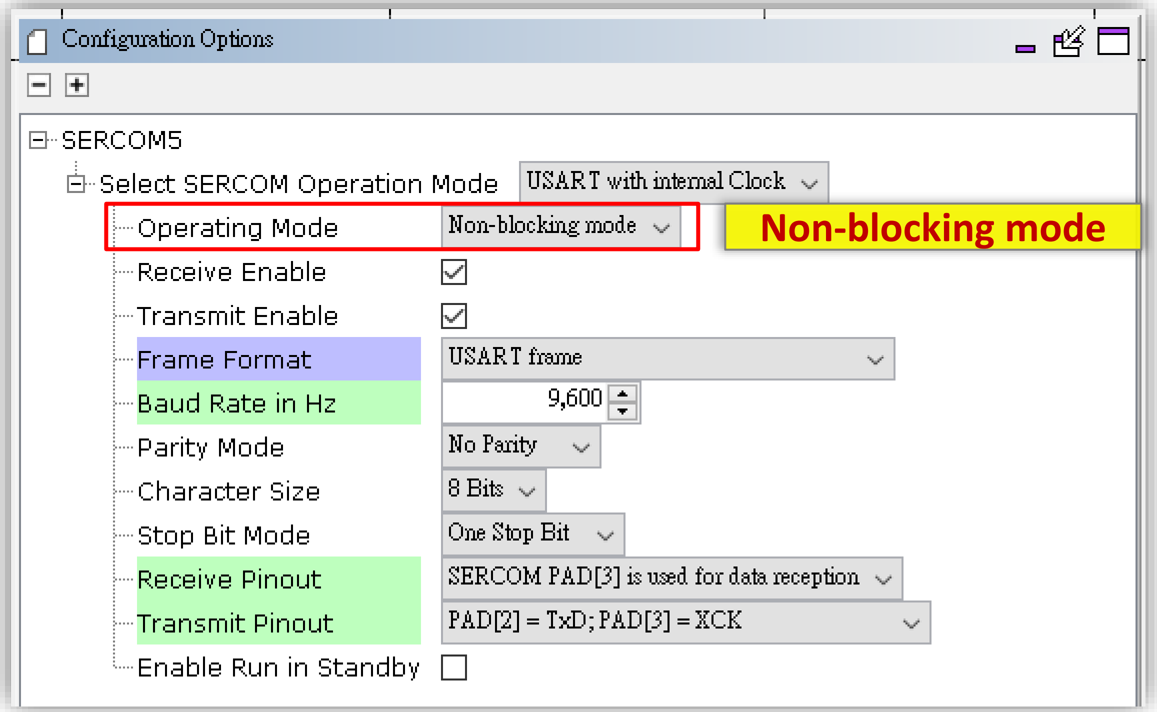
Lab10 SERCOM - UART TxRx Interrupt Callback

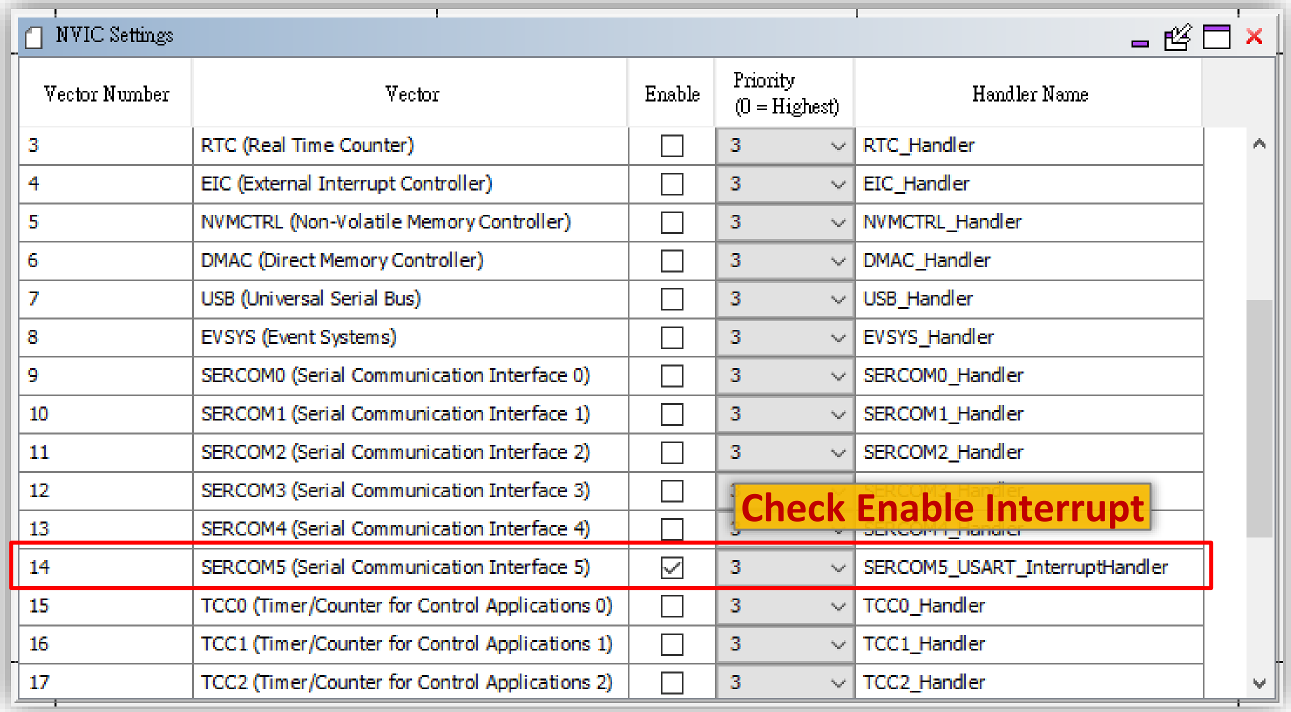
- Try to enable SERCOM-USART Interrupt Callback driver.
- Modify polling transmit function to asynchronous function.
- Add new USART receive function base on interrupt callback function.
- Try to receive character from PC and loopback to TeraTerm as string format "Received Data : x" .
- SERCOM - UART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

- **Let's go!**

Lab10 SERCOM - UART TxRx Interrupt Callback

Step 1

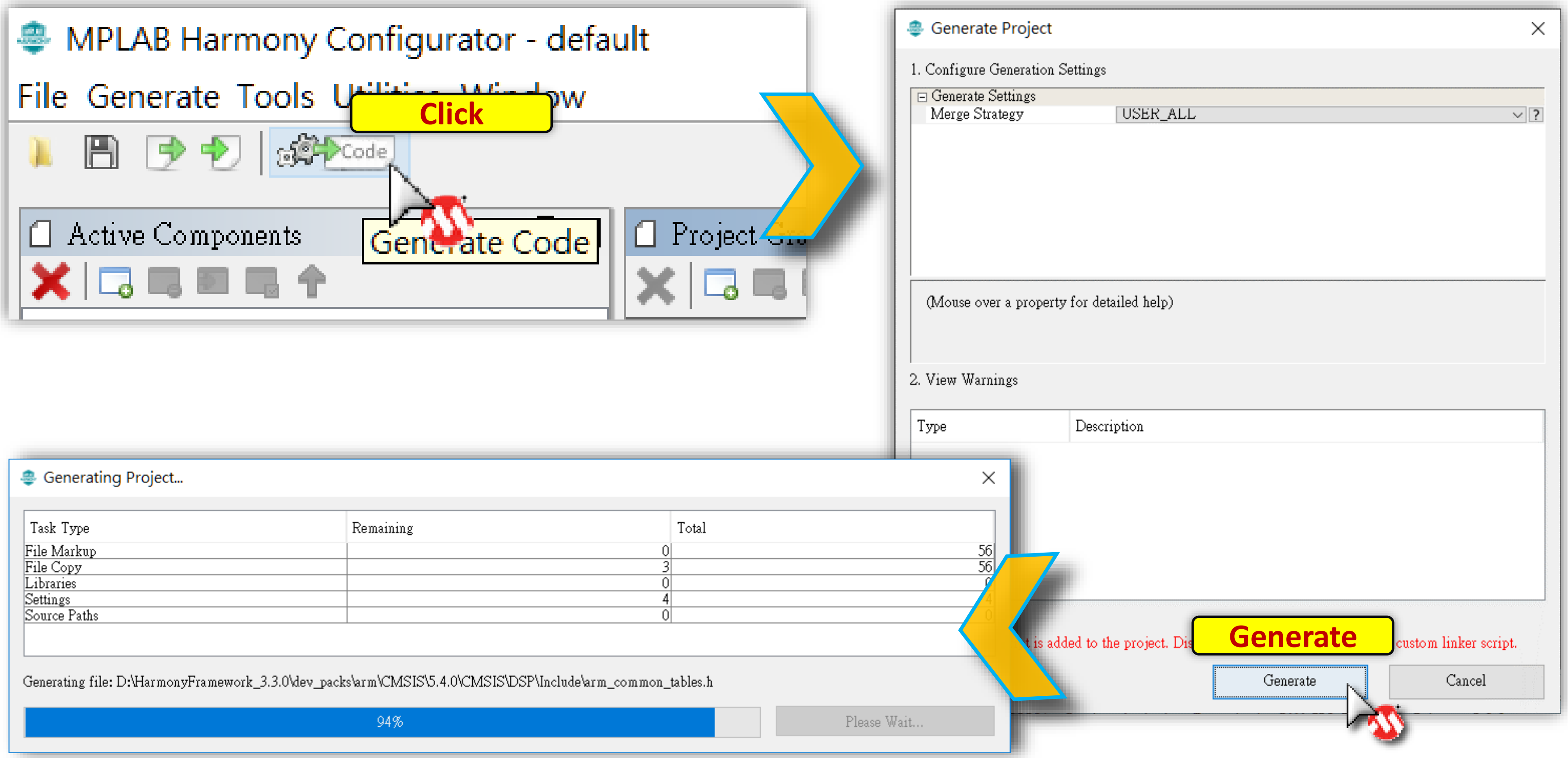




Lab10 SERCOM - UART TxRx Interrupt Callback

Step 2

- Click  to Generate Code and save changes to MHC configure file.



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)
{   …   }
```

MICROCHIP

Lab10 SERCOM - UART TxRx Interrupt Callback

Step 4

a Add code segment to your main loop

```
int main (void)
{ …
  // TODO 10.03
  SERCOM5_USART_ReadCallbackRegister( USART5_Receive, (uintptr_t )NULL );
  SERCOM5_USART_WriteCallbackRegister( USART5_Transmit, (uintptr_t )NULL );
  SERCOM5_USART_Read( USART5_ReceiveData, 1 );

  while(1)
  { …
    // TODO 10.04
    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

Step 5

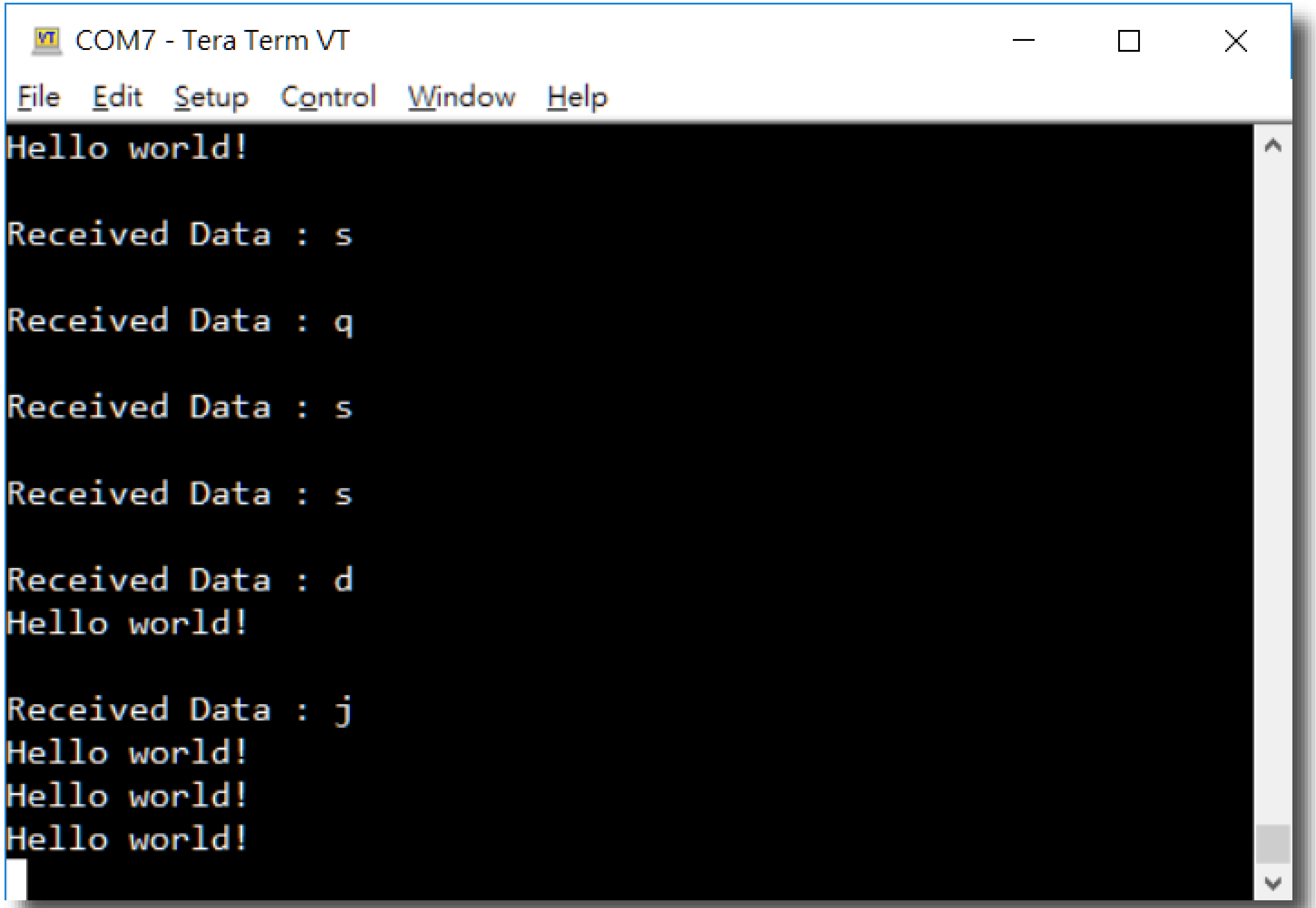
a. **Modify TC3 Callback for UART Tx**

```
void TC3_Overflow(TC_COMPARE_STATUS status, uintptr_t context)
{
        if( status & TC_INTFLAG_OVF_Msk )
        {
           LED1_Toggle();

           // TODO 10.05
           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

Result



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>
#define SYS_CONSOLE_PRINT_BUFFER_SIZE 200
static char consolePrintBuffer[SYS_CONSOLE_PRINT_BUFFER_SIZE];
// TODO 11.02
volatile uint8_t TC3_IsOverflow = 0;
volatile uint8_t TC4_IsOverflow = 0;
...
void TC3_Overflow(TC_COMPARE_STATUS status, uintptr_t context) /* CallBack */
{
   if( status & TC_INTFLAG_OVF_Msk )
   {
      // TODO 11.03
      TC3_IsOverflow = 1;
   }
}

void TC4_Overflow(TC_COMPARE_STATUS status, uintptr_t context) /* CallBack */
{
   if( status & TC_INTFLAG_OVF_Msk )
   {
      // TODO 11.04
      TC4_IsOverflow = 1;
   }
}
```

MICROCHIP

Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

Step 2

a Add code segment to user code.

```
void TC4_Overflow(TC_COMPARE_STATUS status, uintptr_t context) /* CallBack */
{
  …
}
// 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

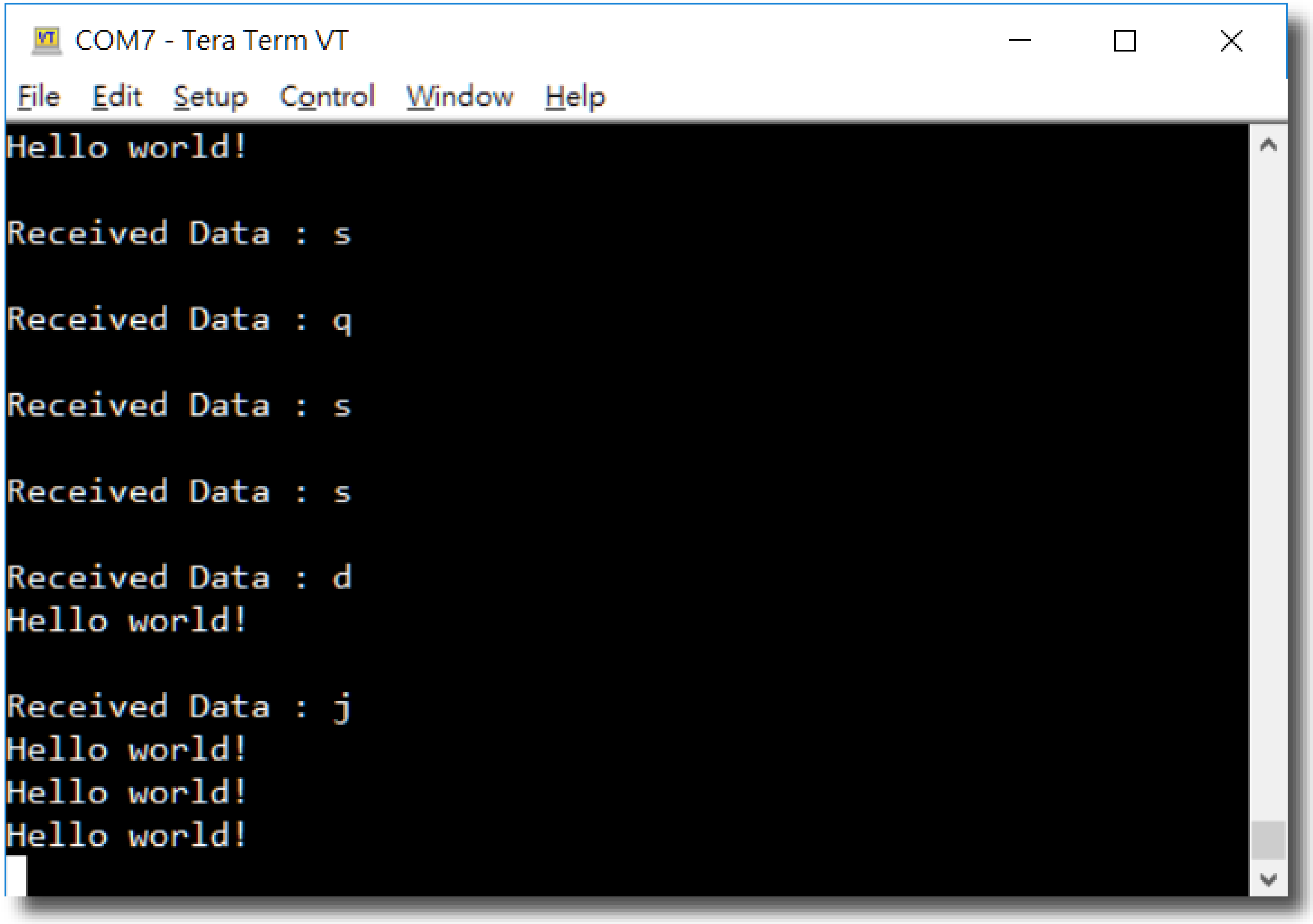
Step 3

a Add code segment to your main while loop.

```
while(1)
{
   // TODO 11.06
   if (TC3_IsOverflow)
   {
     TC3_IsOverflow = 0;
     LED1_Toggle();
     myprintf( "Hello World!!\r\n" );
   }

   if (TC4_IsOverflow)
   {
     TC4_IsOverflow = 0;
     LED2_Toggle();
   }

   if( USART5_IsReceived )
   {
      // TODO 11.07
     myprintf( "\r\nReceived Data : %1c\r\n", USART5_ReceiveData[0] );
     …
   }
```

Lab11 SERCOM - UART TxRx Interrupt Callback with myprintf

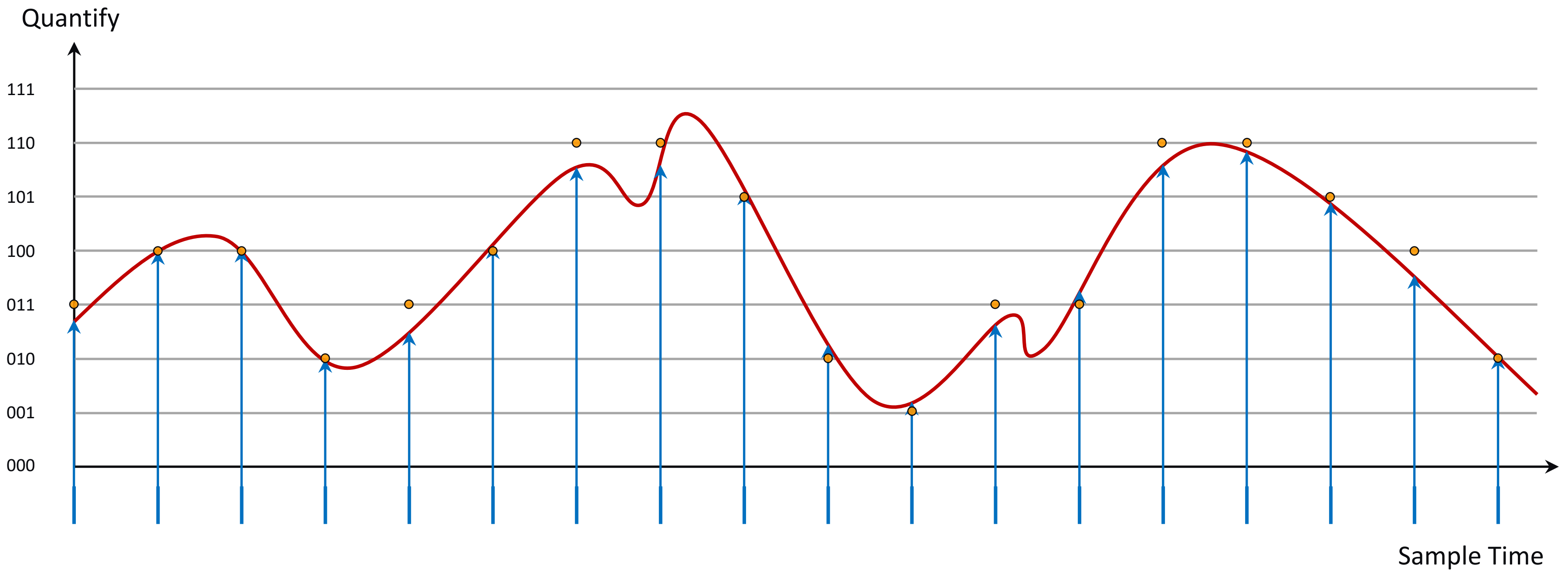
Result



High Speed ADC Architecture

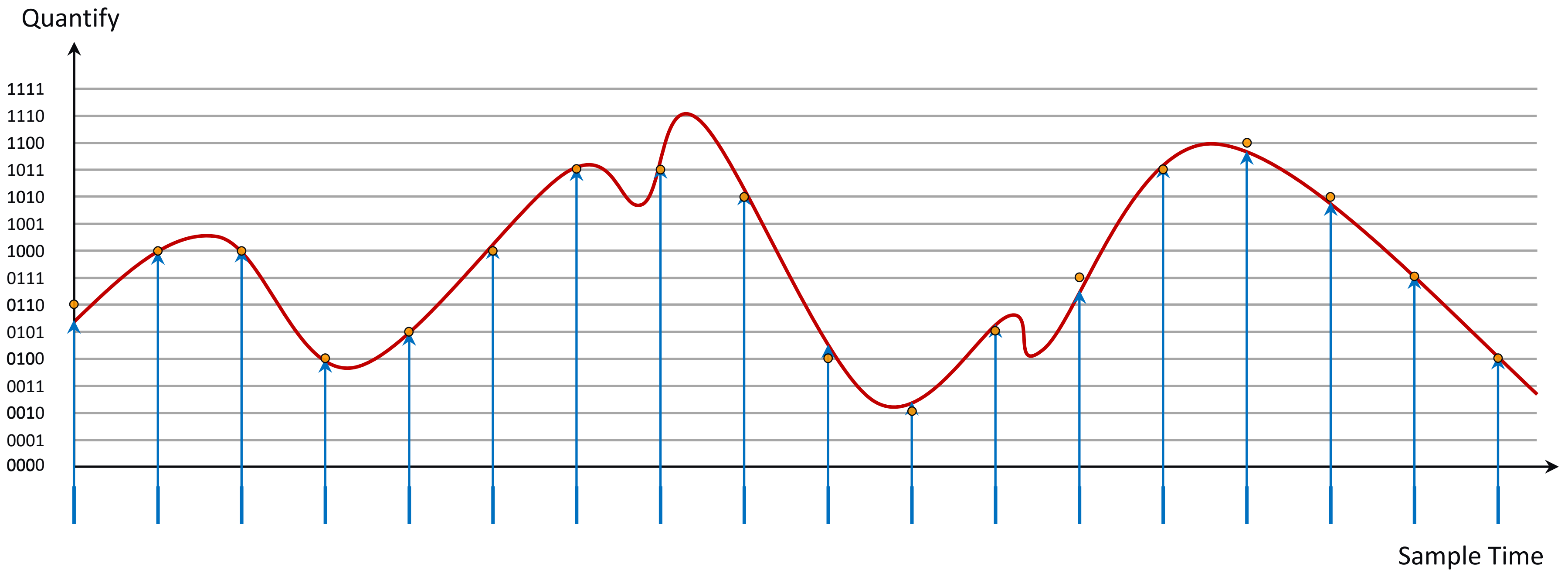
What’s ADC ?

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



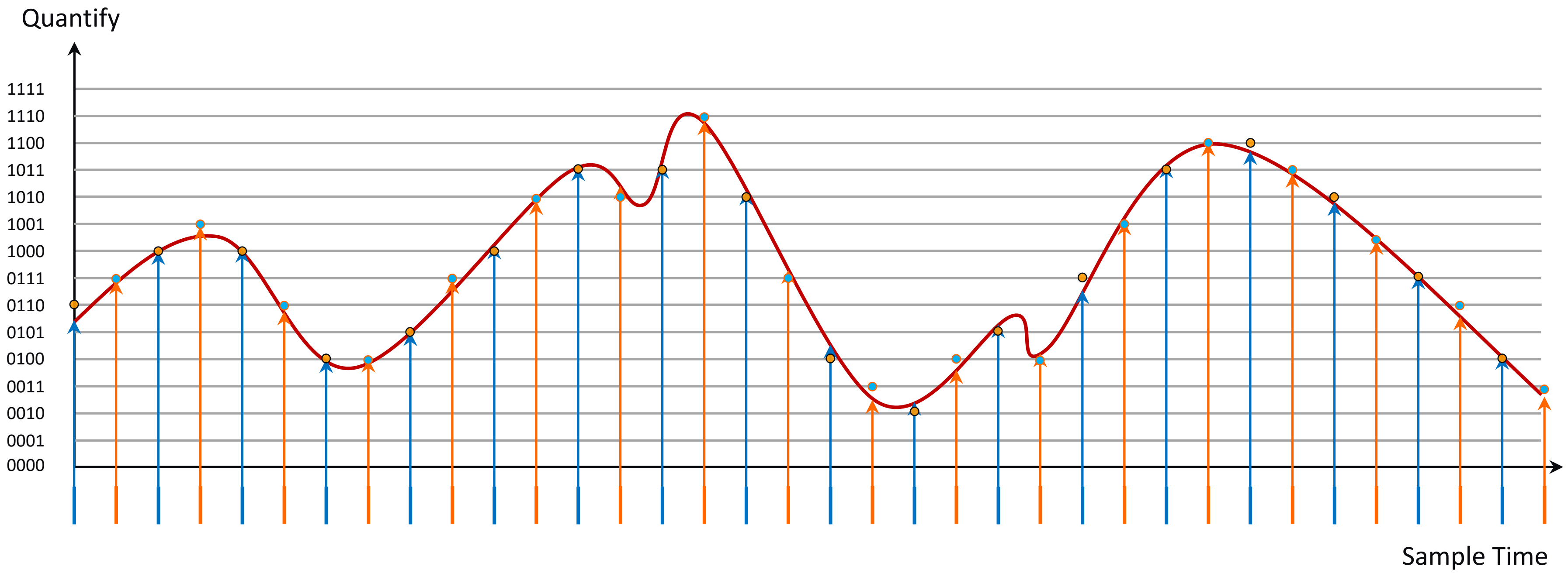
Resolution

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



Sample Rate

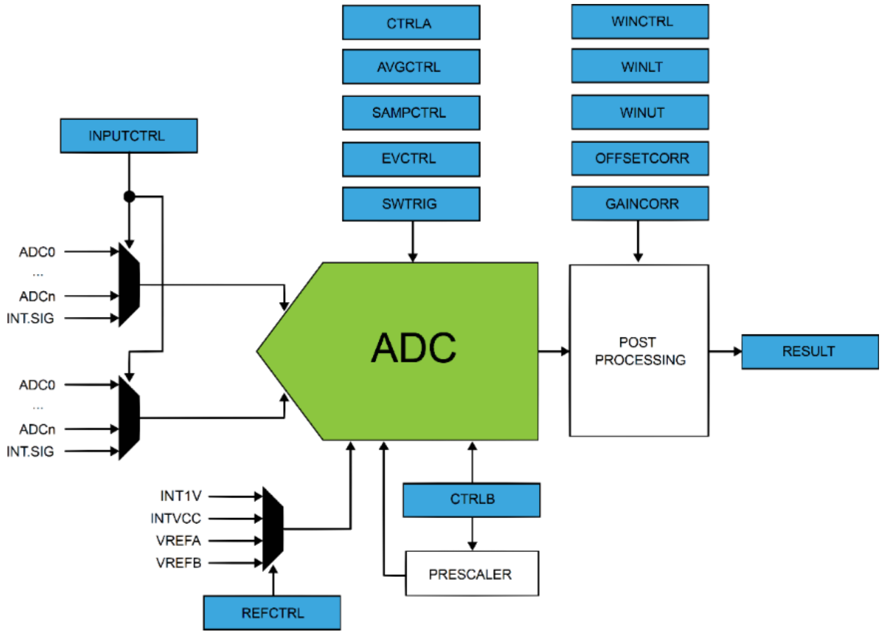
- **The analog signal is required to define the rate at which new digital values are sampled from the analog signal.**
- **This faithful reproduction is only possible if the sampling rate is higher than twice the highest frequency of the signal. (wiki)**



ADC Block Diagram

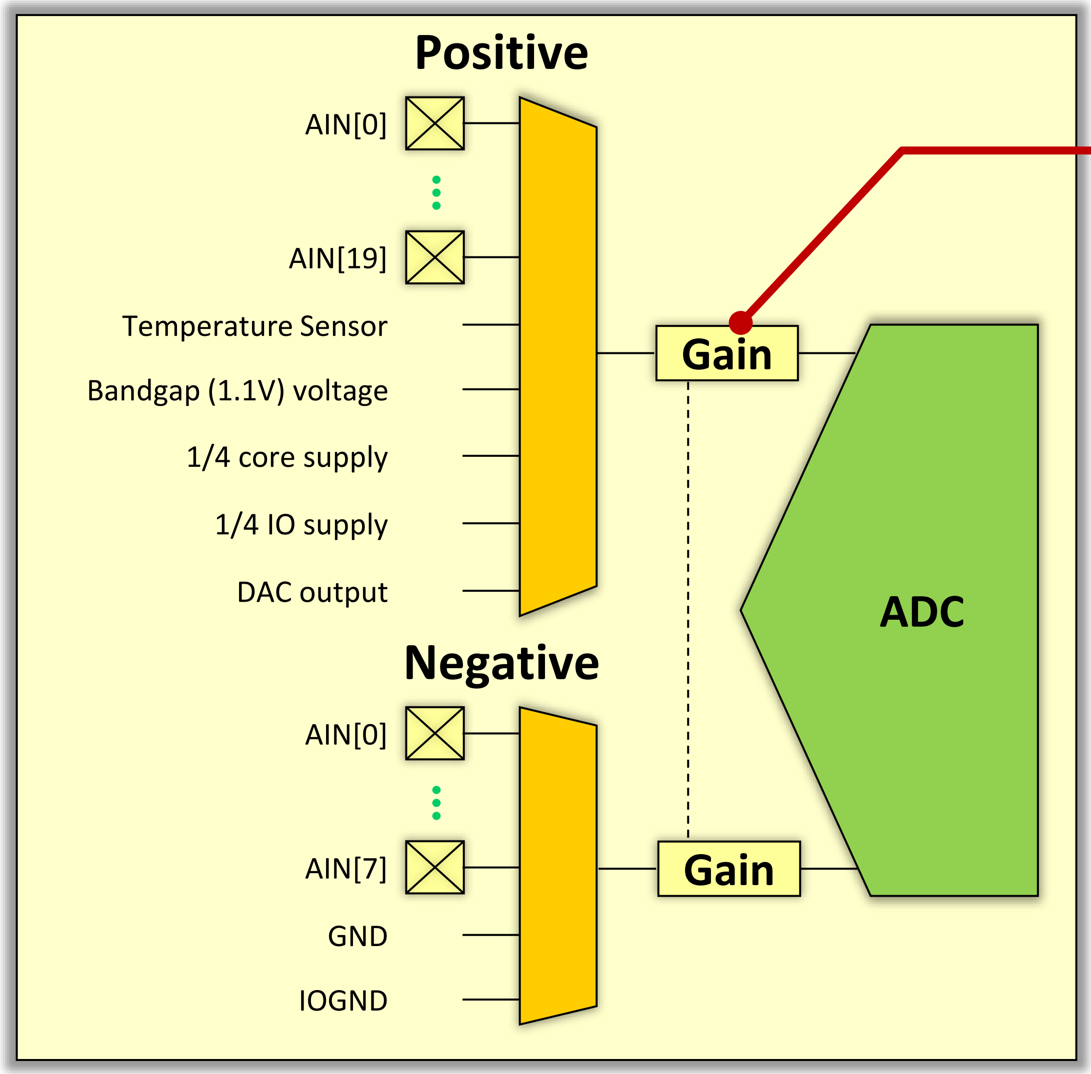
- Up to 32 analog input, 8, 10 or 12 bits resolution.
- Differential and single-ended inputs.
- 1/2x to 16x gain. Up to 350ksps.
- Support Event-triggered and pin scan.
- Support averaging and oversampling.
- External or Internal reference voltage.
- Oversample accumulation average and window monitor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

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



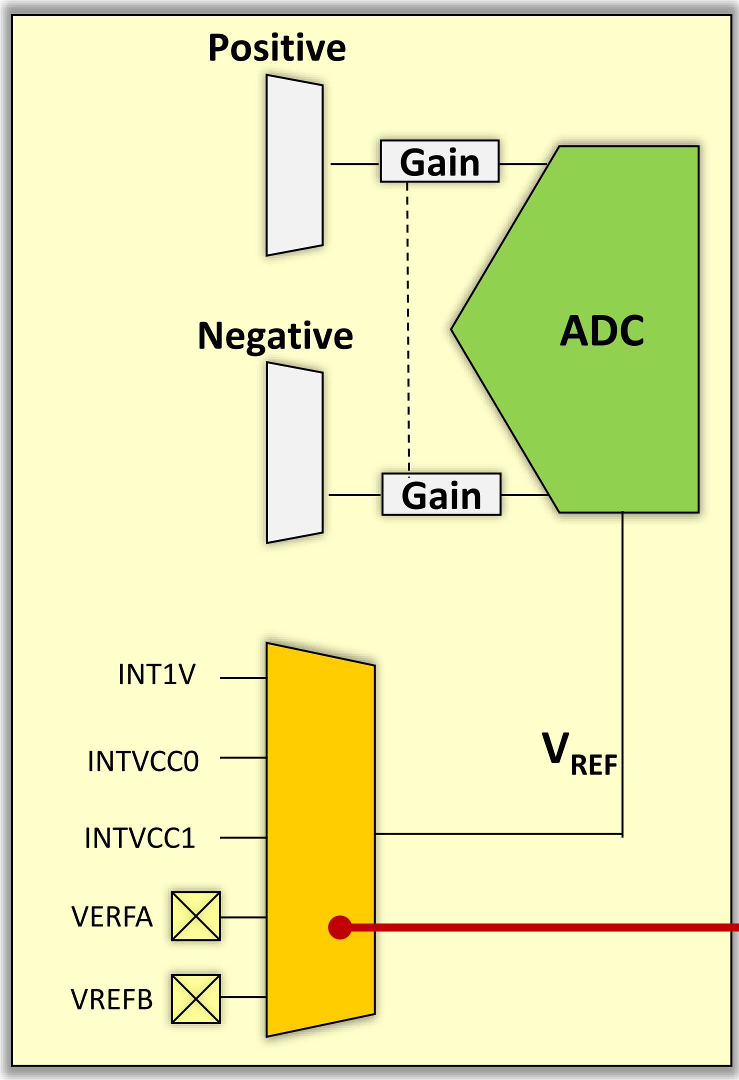
MICROCHIP

ADC Channel Selection & Gain



GAIN[3:0]	Name
0x0	1X
0x1	2X
0x2	4X
0x3	8X
0x4	16X
0x5-0xE	Reserved
0xF	1/2x

ADC Reference Voltage Selection



VREFA and VREFB range

$1.0V \sim (V_{DDANA} - 0.6V)$

Conversion range:

- **Single-ended mode**

  $0V \sim V_{REF}/Gain$

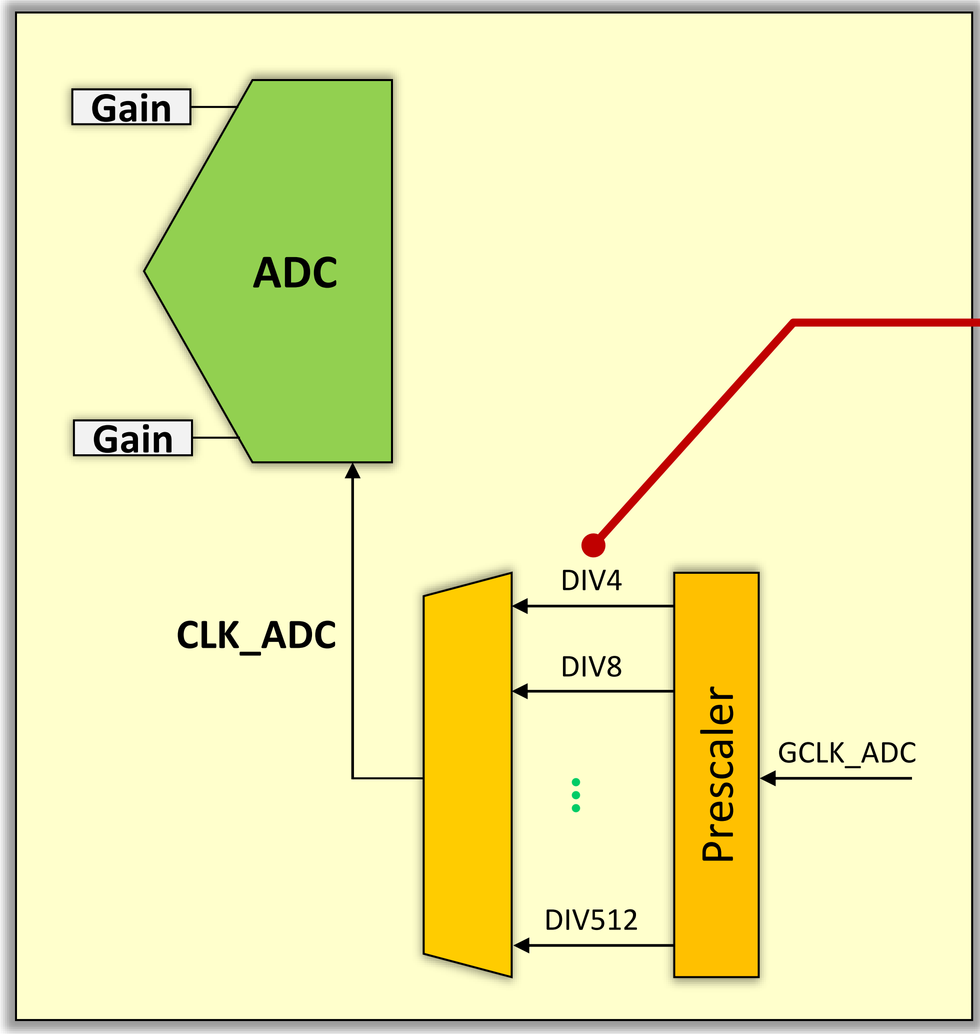
- **Differential mode**

  $-V_{REF}/Gain \sim V_{REF}/Gain$

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



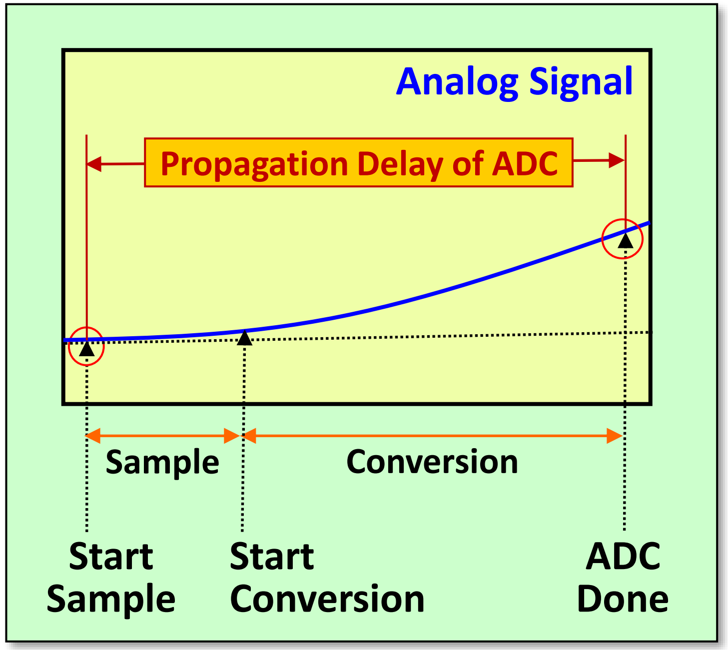
PRESCALER[2:0]	Name
0x0	DIV4
0x1	DIV8
0x2	DIV16
0x3	DIV32
0x4	DIV64
0x5	DIV128
0x6	DIV256
0x7	DIV512

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

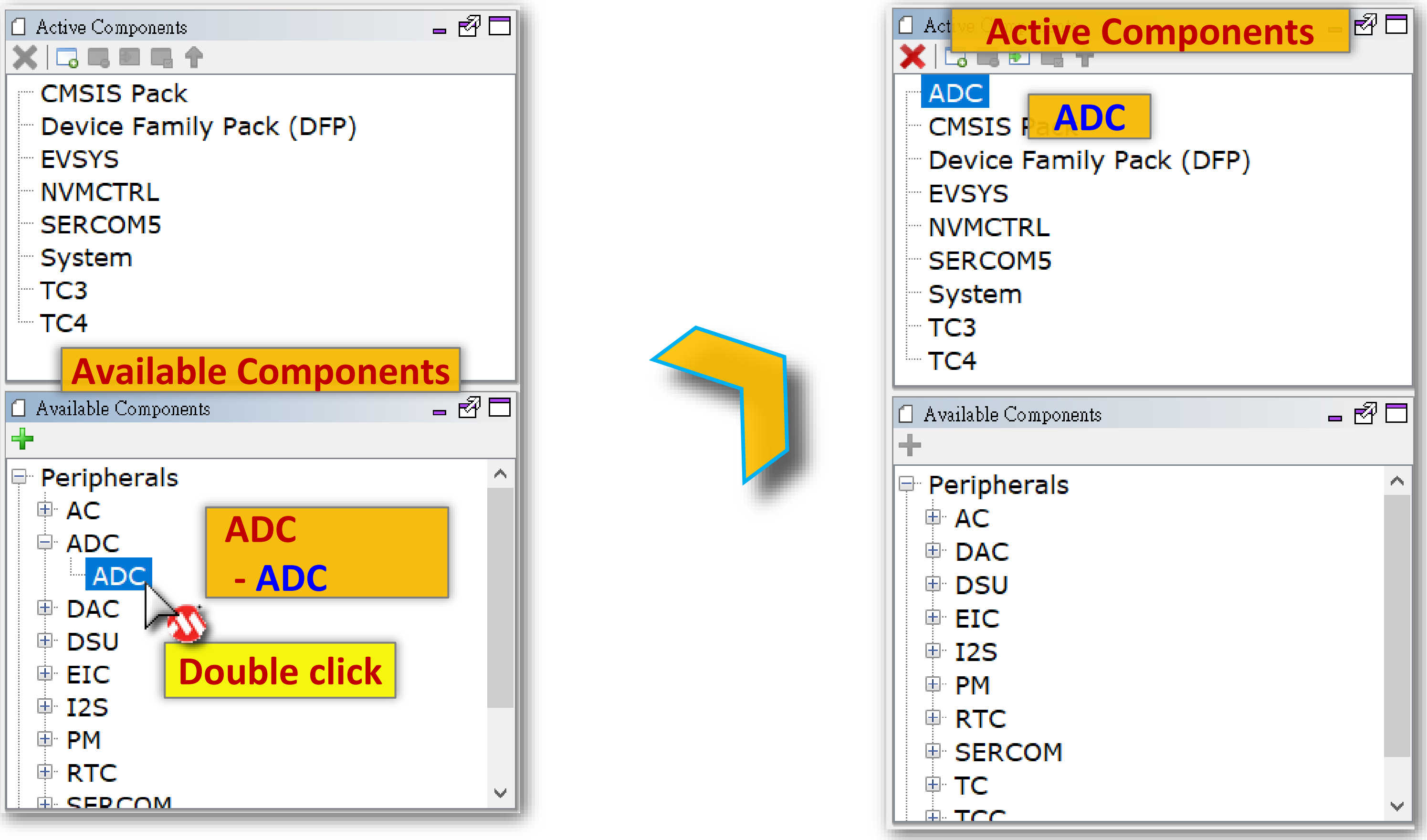
- **Maximum Sampling Rate :**
  Single-shot : 2.1M/Round(6+0.5) = 300k samples/second.
  Free-running : 2.1M/6 = up to 350k samples/second.



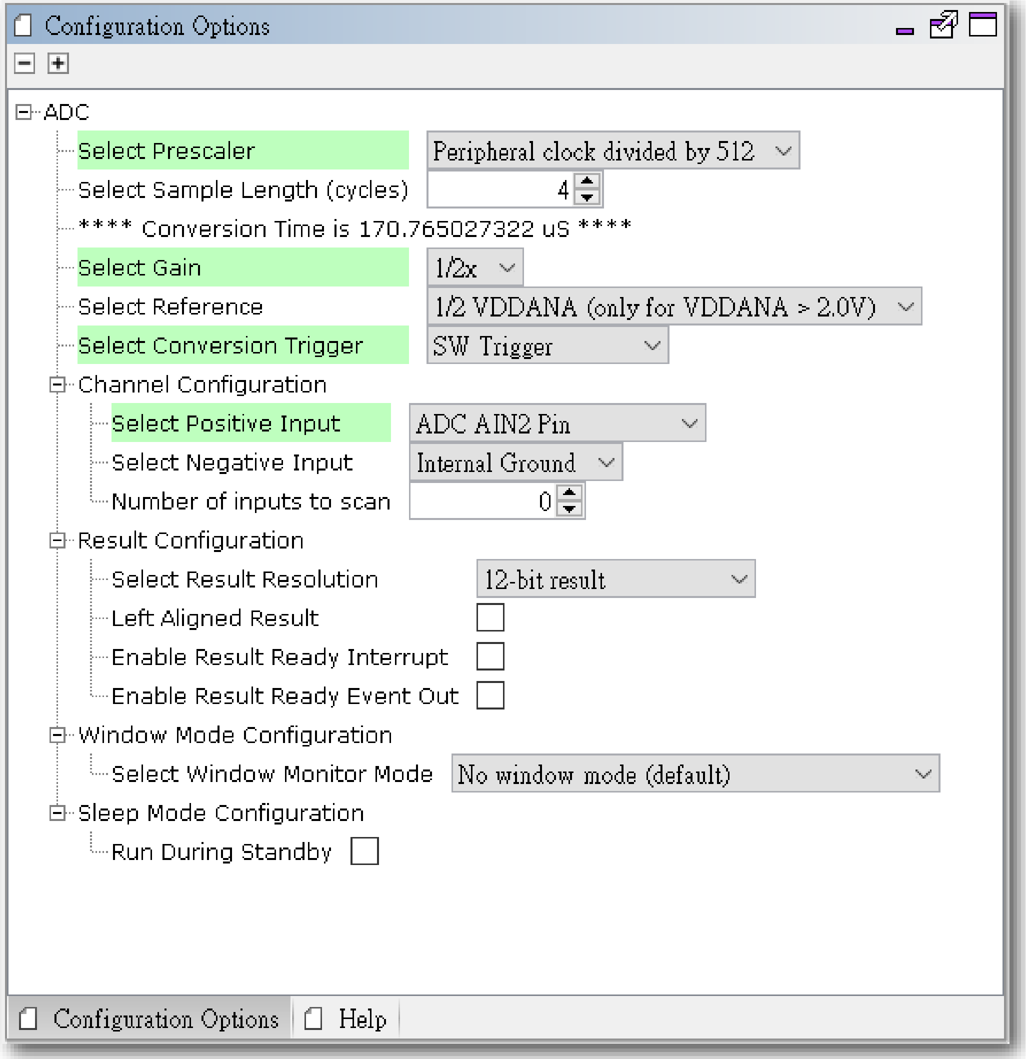
Symbol	Parameter	Conditions	Min.	Max.	Units
RES	Resolution		8	12	bits
f_{CLK_ADC}	ADC Clock frequency		30	2100	kHz
	Sample rate	Single shot	5	300	ksps
		Free running	5	350	ksps
	Sampling time		0.5	-	cycles
	Conversion time	1x Gain	6	-	cycles

Add ADC Function using MHC

- Find ADC component in Available Components window.
- Double click ADC to add to Active Components window.

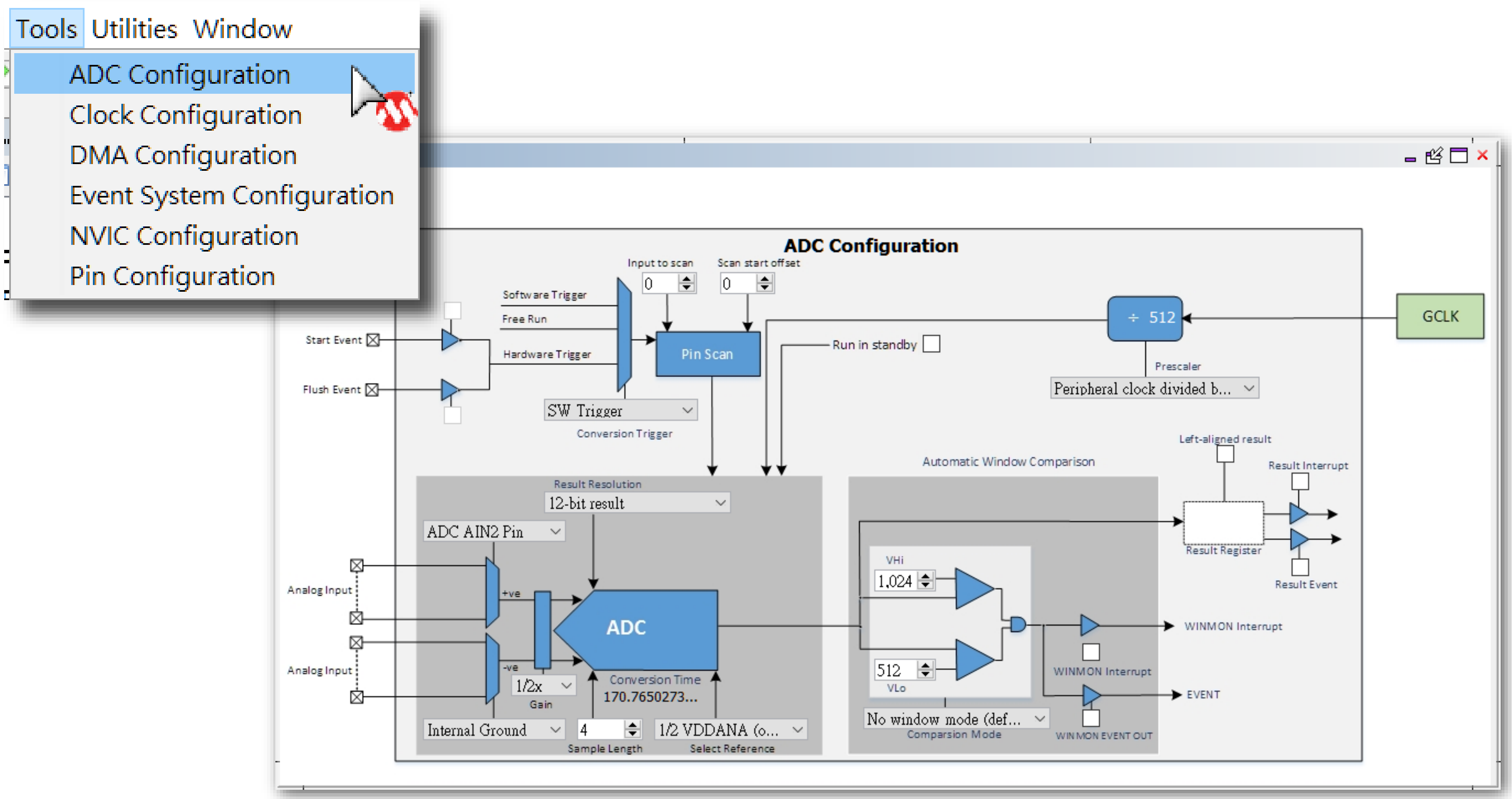


ADC Polling
Configuration Options Example



ADC Polling

Easy View Configuration Example



ADC Polling

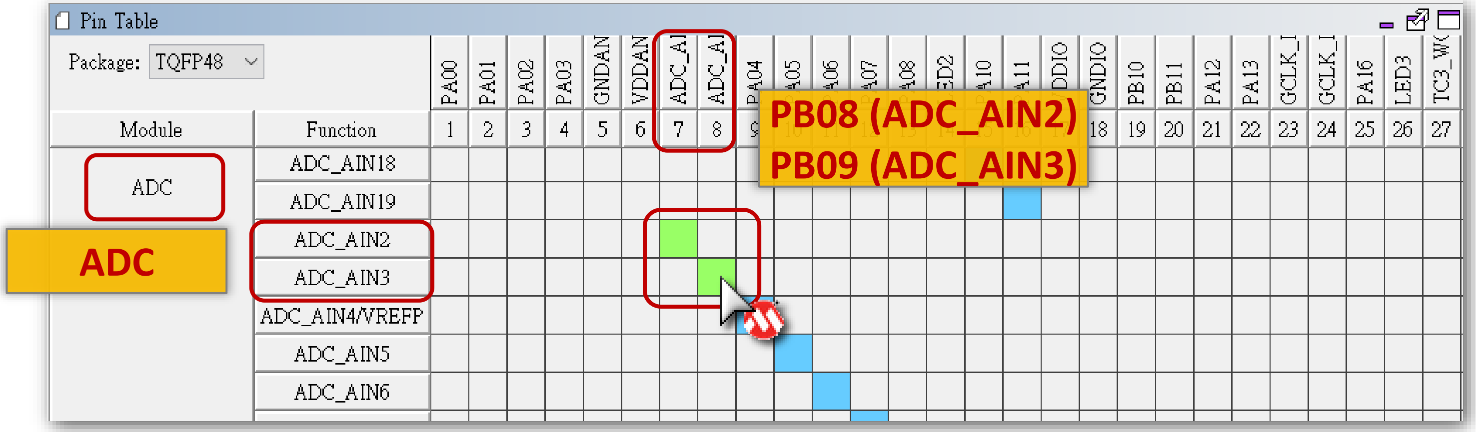
PINMUX Configuration Example

- **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	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
	7	11	PB08	VDDANA	EXTINT[8]		AIN[2]		Y[14]			SERCOM4/ PAD[0]	TC4/WO[0]	TCC3/ WO[6]		
	8	[illegible]	PB09	VDDANA	EXTINT[9]		AIN[3]		[illegible]			SERCOM4/ PAD[1]	TC4/WO[1]	TCC3/ WO[7]		

PB08
PB09

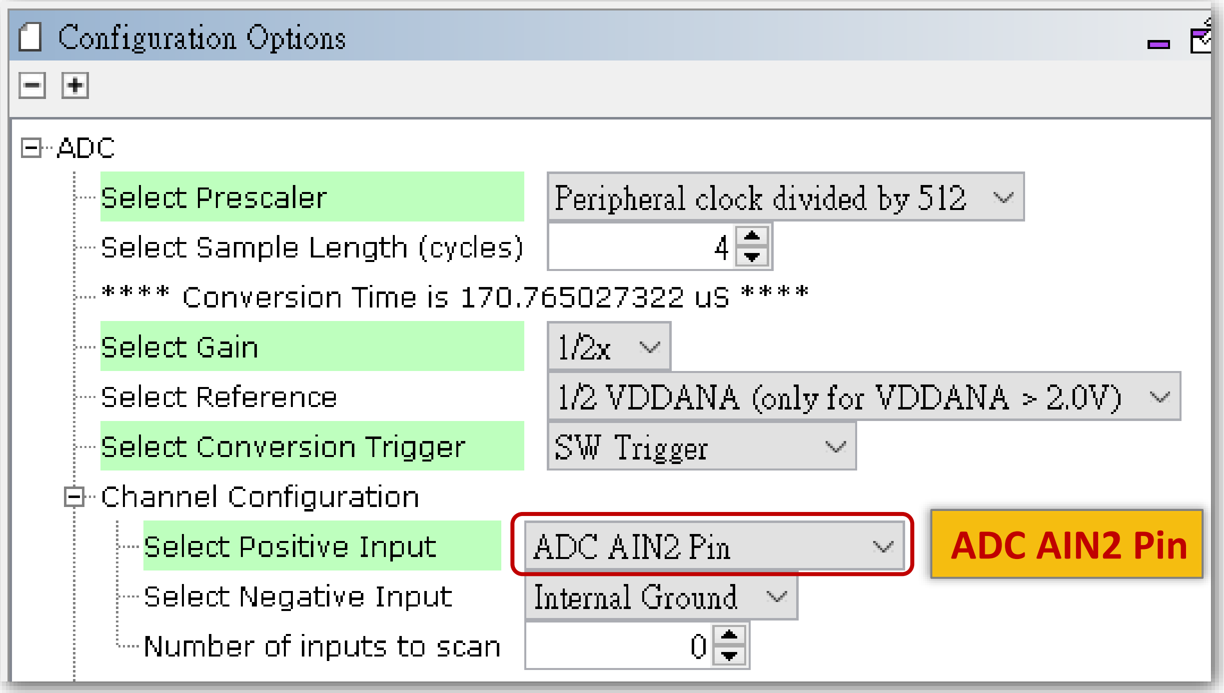
AIN[2]
AIN[3]



ADC Polling

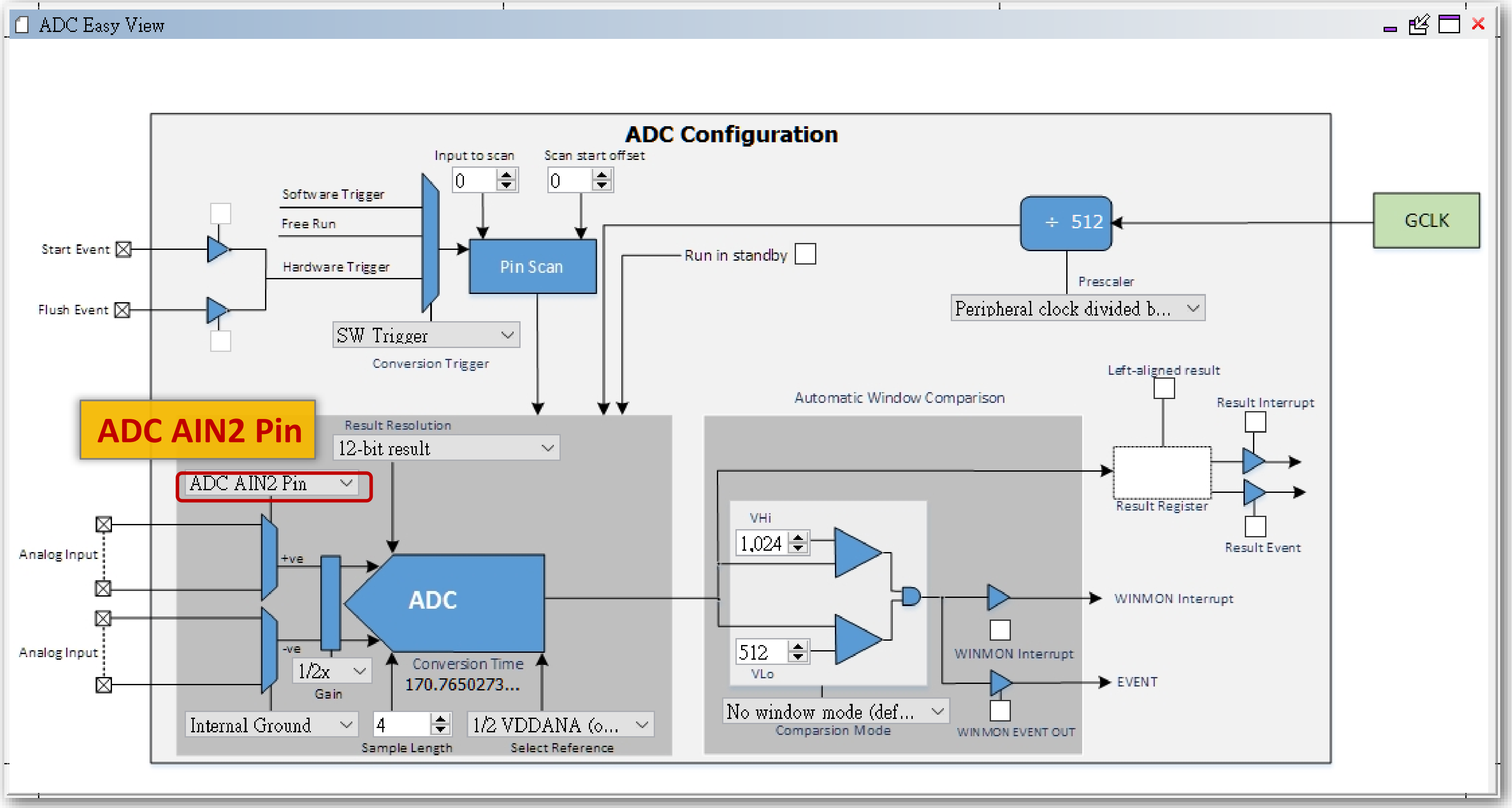
PINMUX Configuration Example

- 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(1)
    {
        …
        ADC_ConversionStart();
        while( !ADC_ConversionStatusGet() );
        myprintf( "%d\r\n", ADC_ConversionResultGet() );
    }
}
```

ADC Factory Calibration

- **ADC Factory calibration value will auto load and applicate in MHC 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) );
}
```

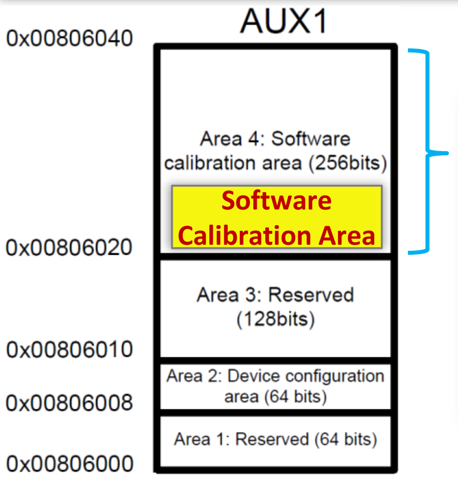


Table 10-5. NVM Software Calibration Area Mapping

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.

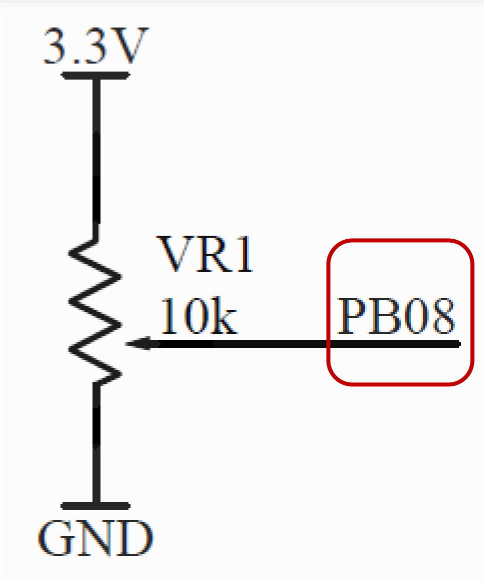
Lab12 ADC Polling Software Trigger

- 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

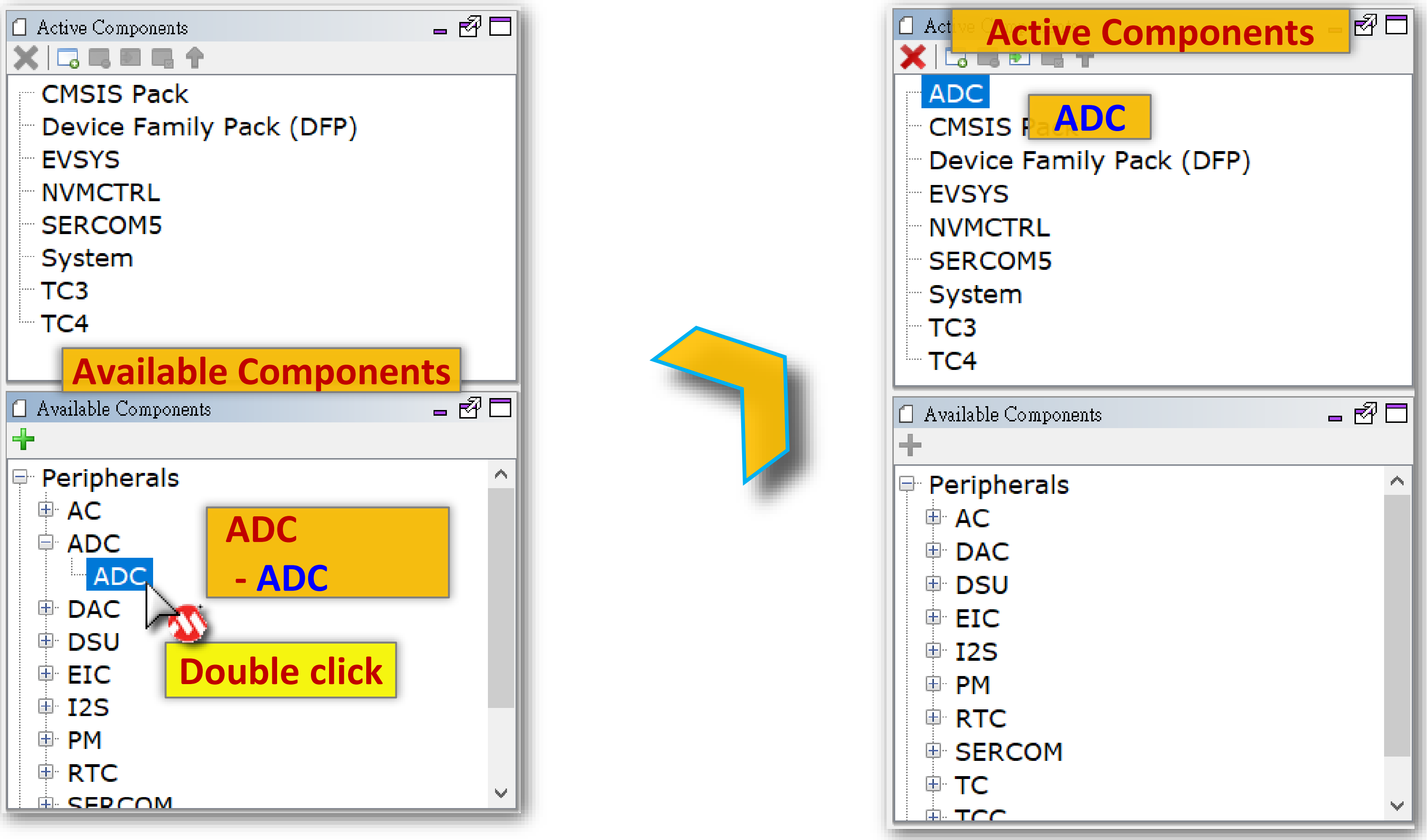
Pin(1)			I/O Pin	Supply	A	B(2)(3)				
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC
	7	11	PB08	VDDANA	EXTINT[8]		AIN[2]		Y[14]	
	8	12	PB09	VDDANA	EXTINT[9]		AIN[3]		Y[15]	



Lab12 ADC Polling Software Trigger

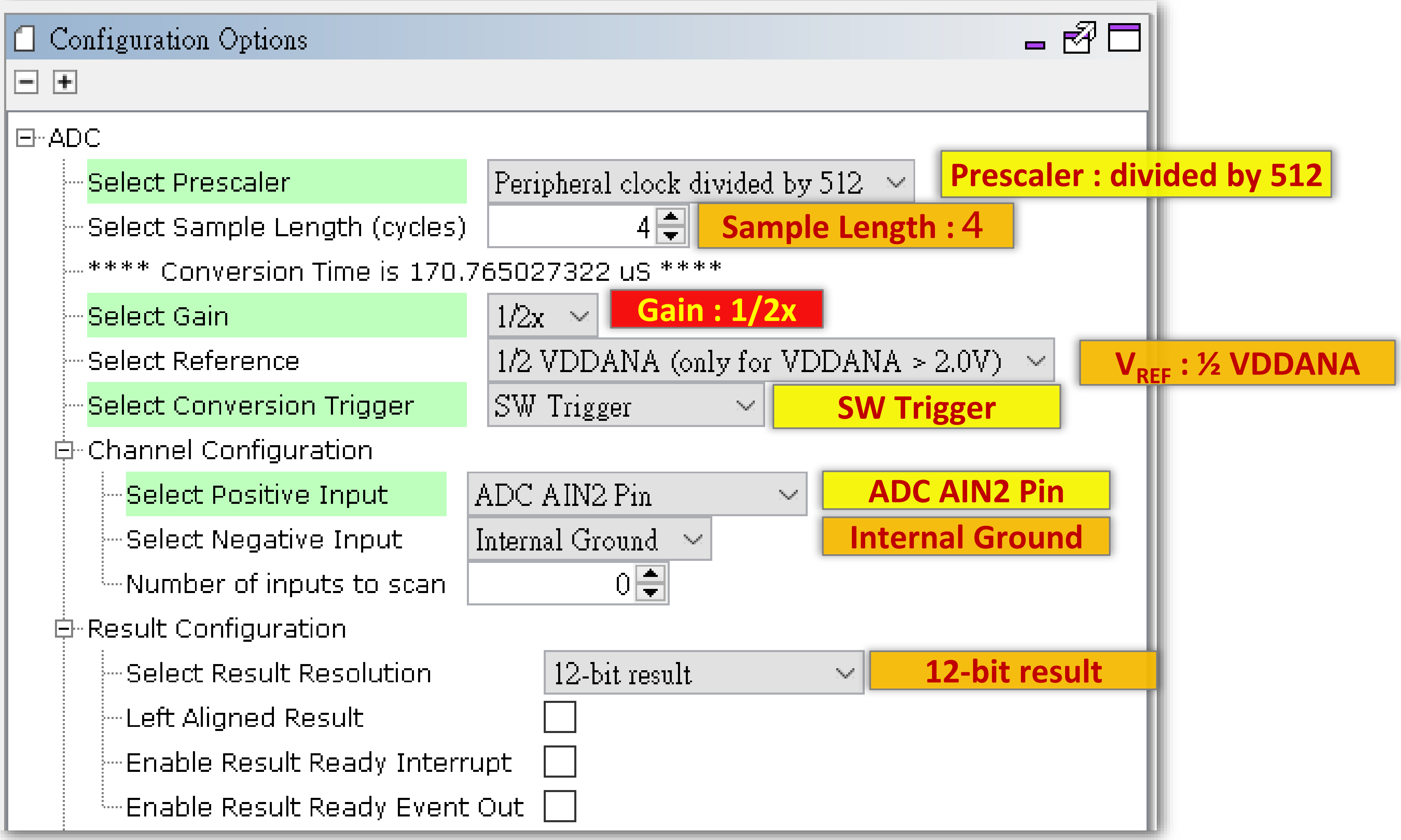
Step 0

- Find ADC component in Available Components window.
- Double click ADC to add to Active Components window.



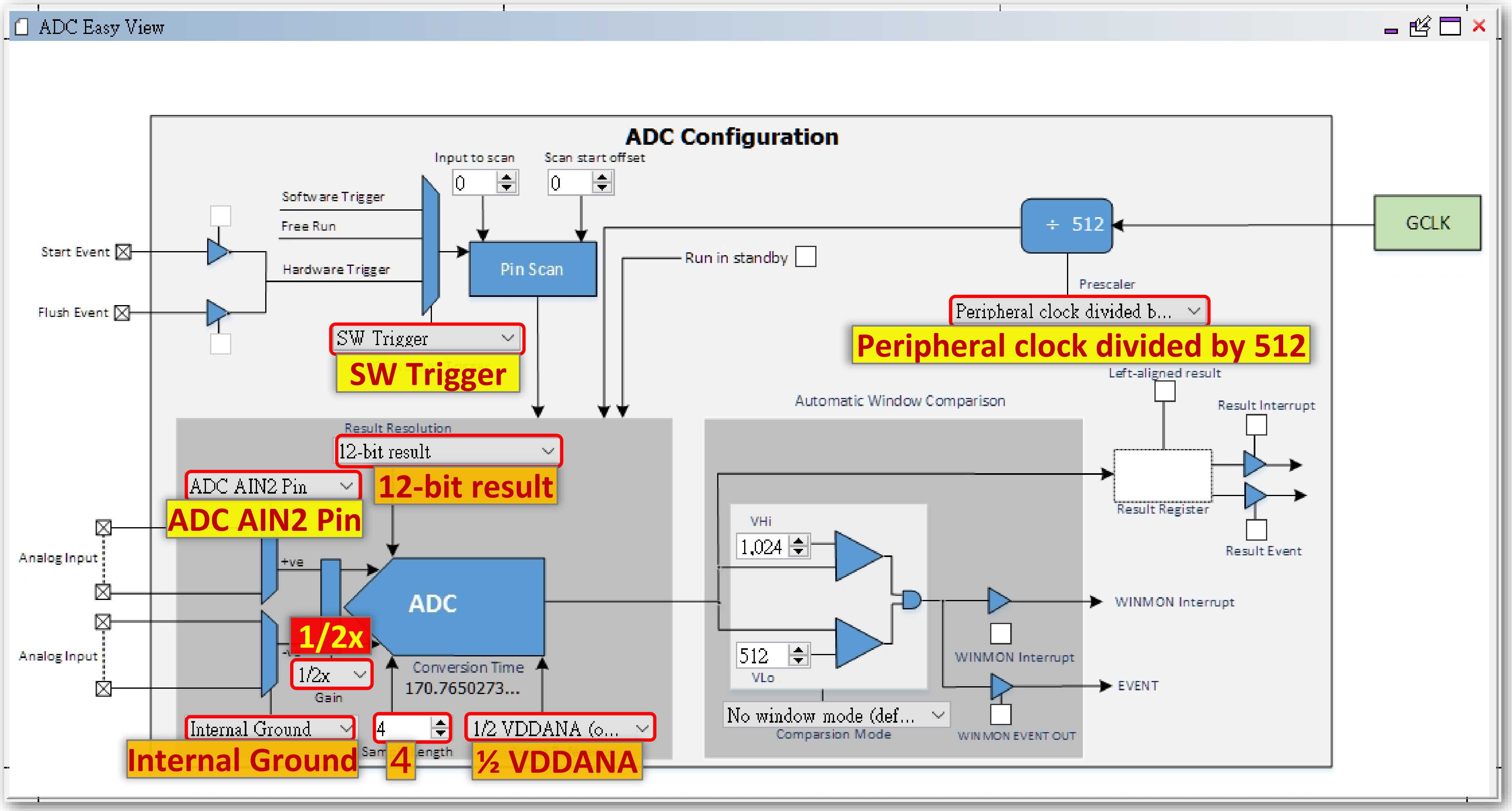
Lab12 ADC Polling Software Trigger

Step 1 (Use Configuration Options)



Lab12 ADC Polling Software Trigger

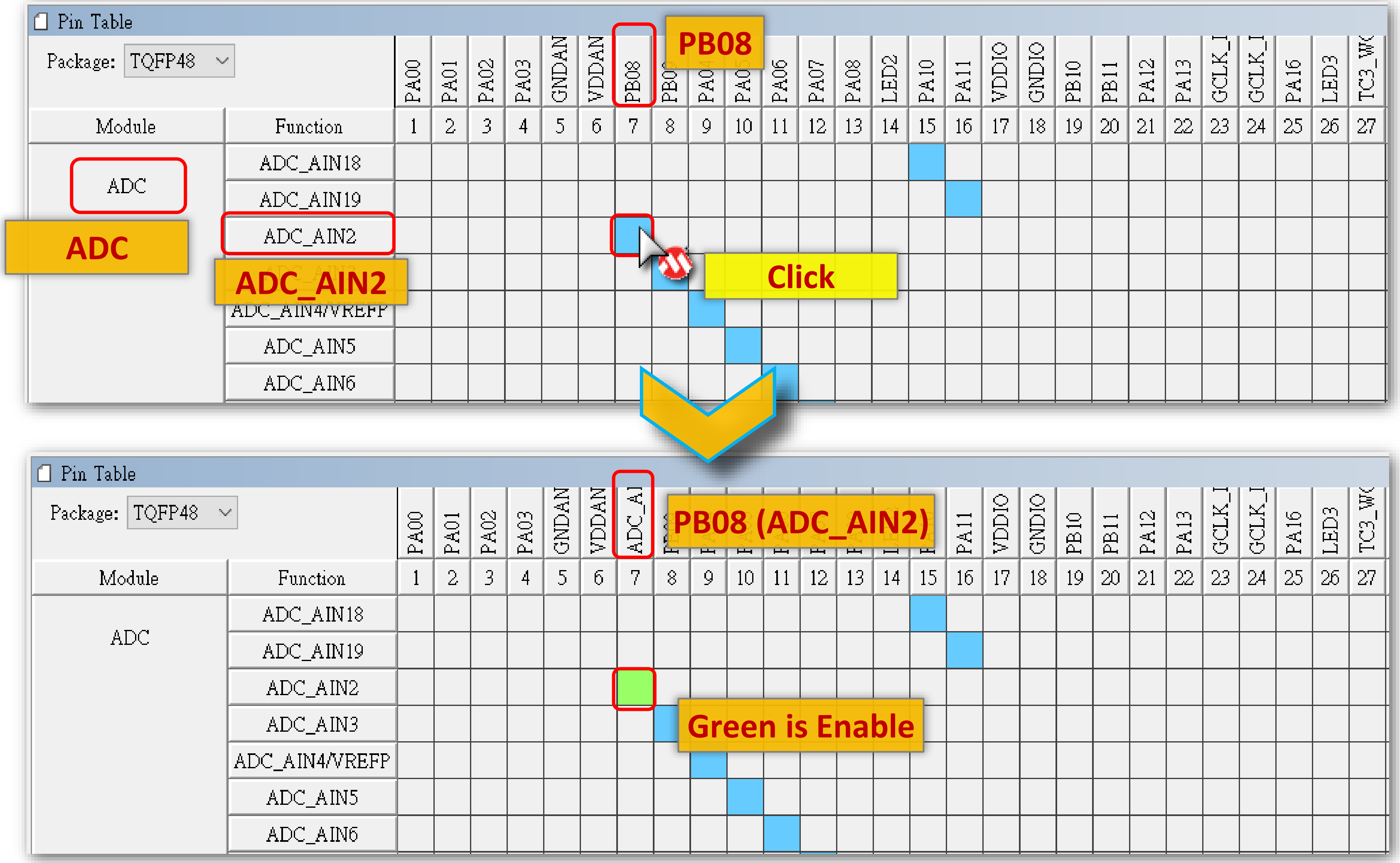
Step 1 (Use Easy View)



MICROCHIP

Lab12 ADC Polling Software Trigger

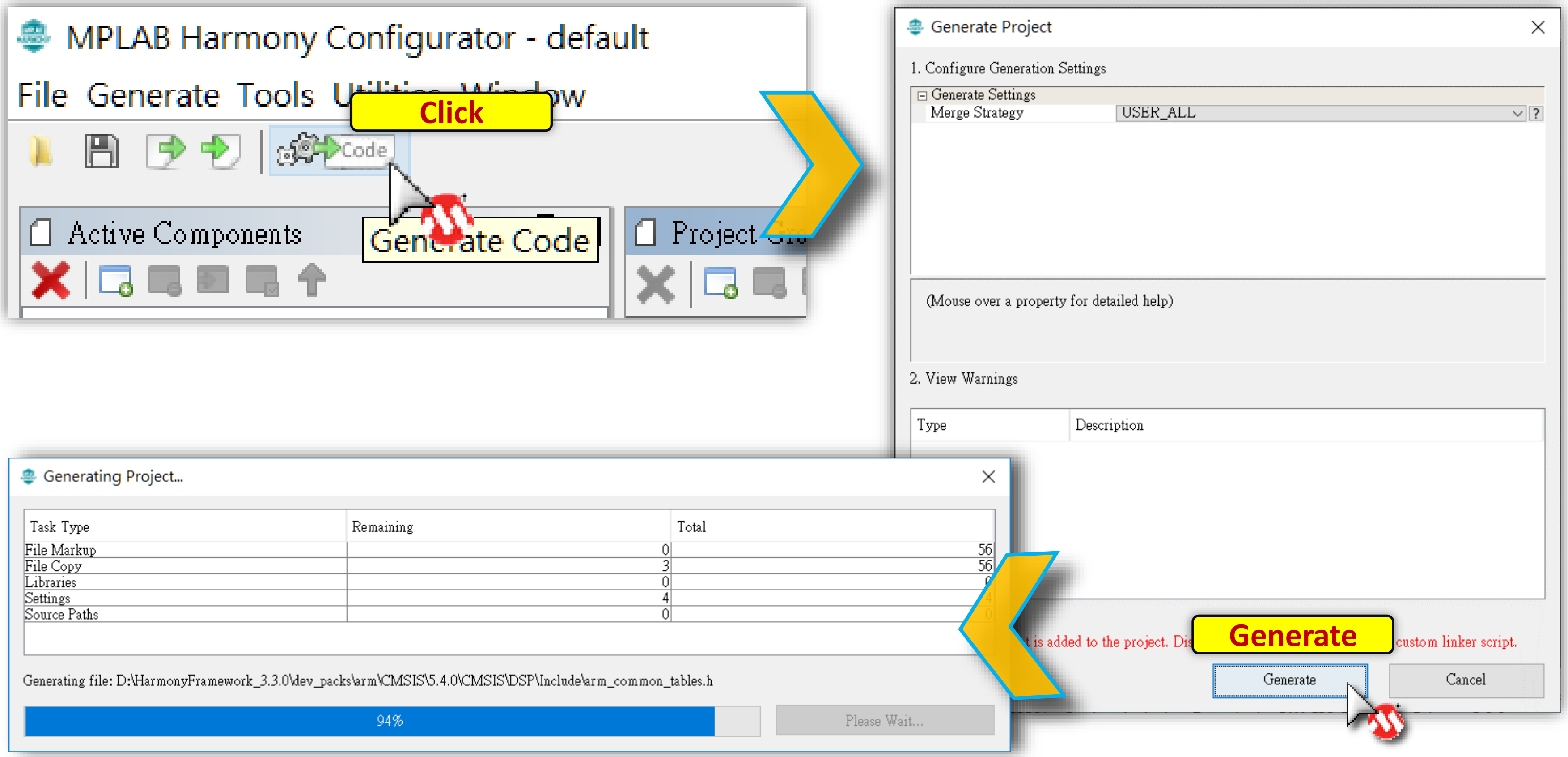
Step 2



Lab12 ADC Polling Software Trigger

Step 3

- Click  to Generate Code and save changes to MHC configure file.



Lab12 ADC Polling Software Trigger

Step 4

a Add code segment to your main.c

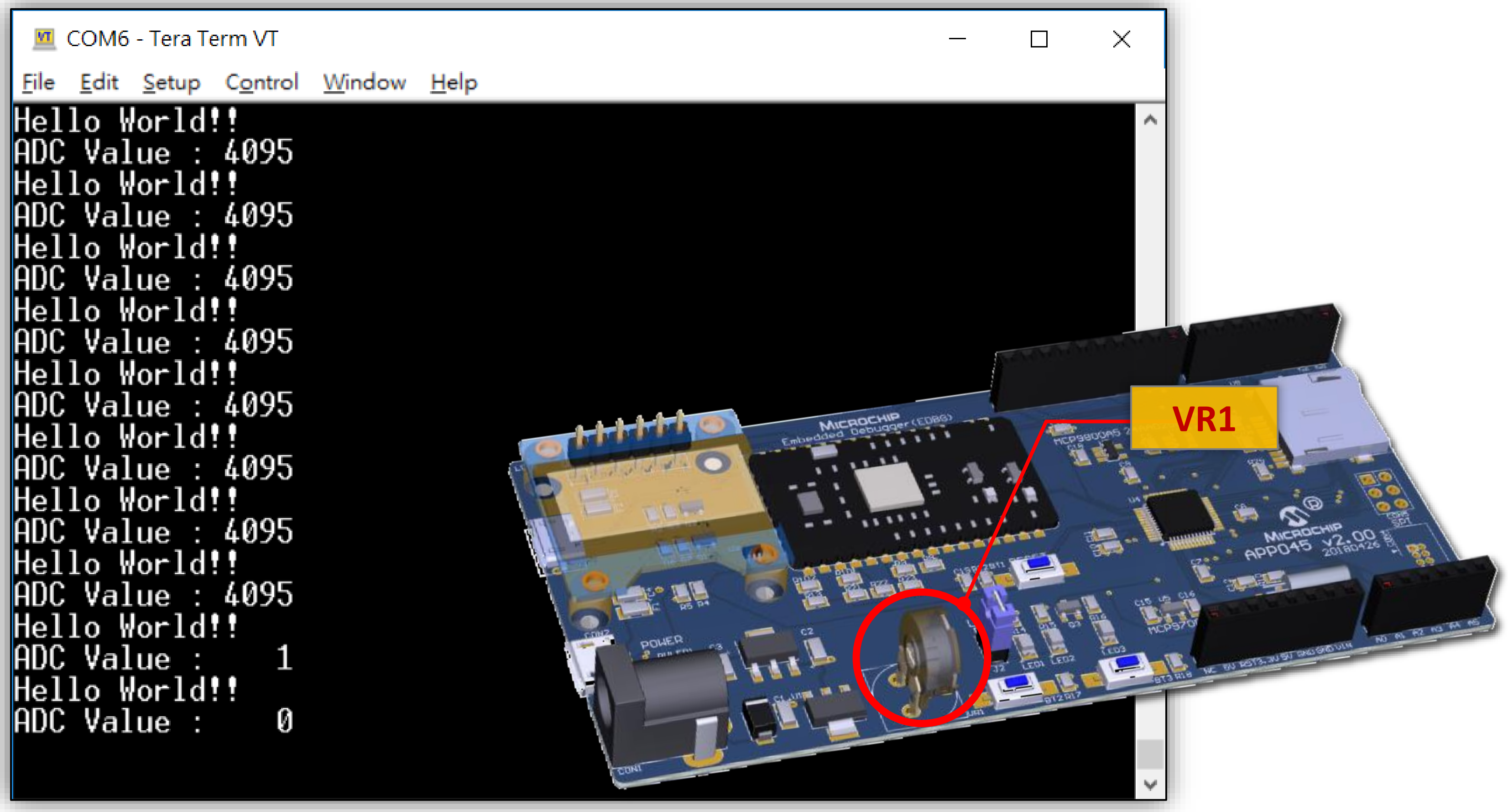
```
// TODO 12.01
uint16_t ADC_Result;

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

   while(1)
   { …
      if( TC3_IsOverflow )
      {
         // TODO 12.03
         ADC_ConversionStart();
         while( !ADC_ConversionStatusGet() );
         ADC_Result = ADC_ConversionResultGet();
         myprintf( "ADC Data : %d\r\n", 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

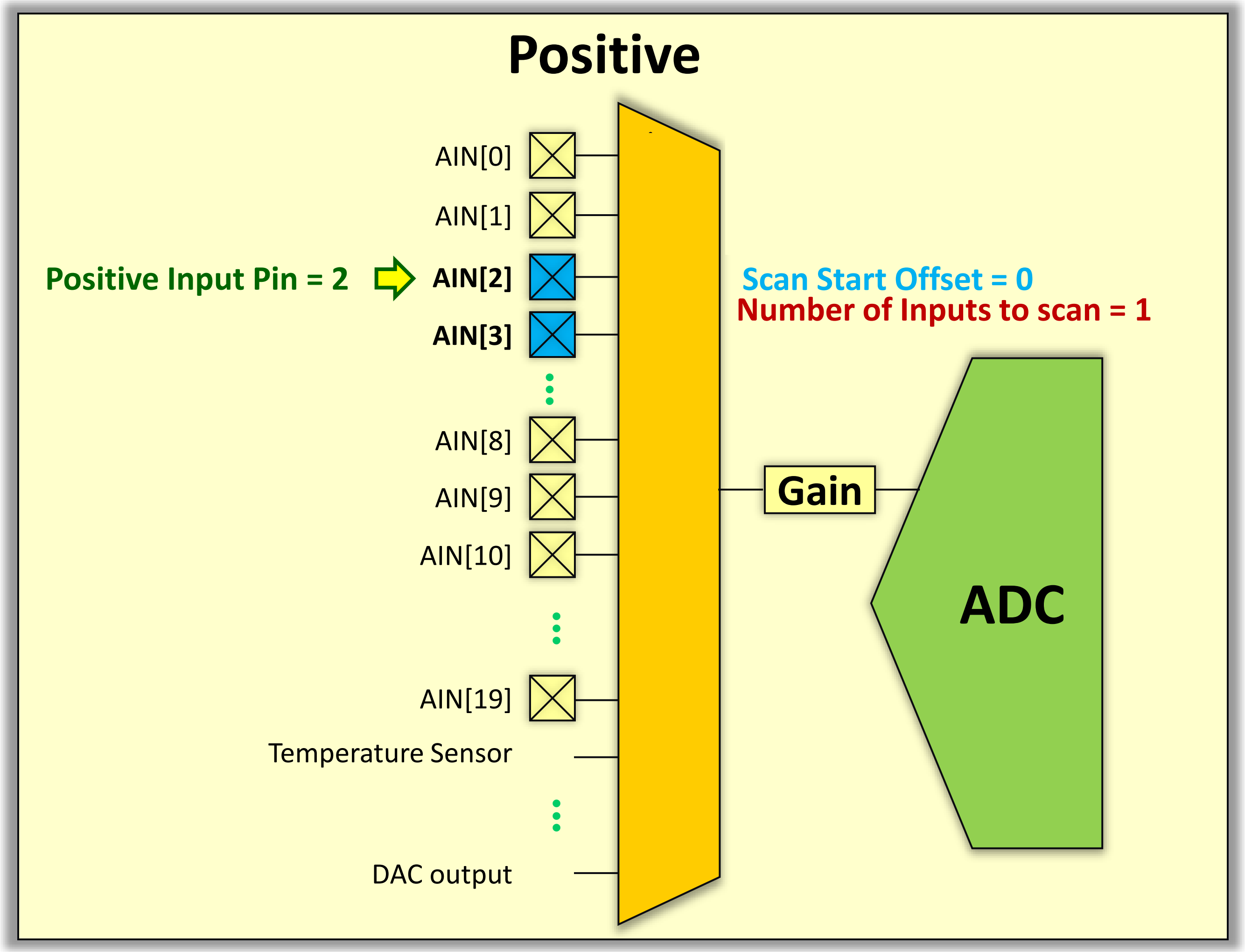
For Example : To do ADC Pin Scan from AIN[2] to AIN[3].

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

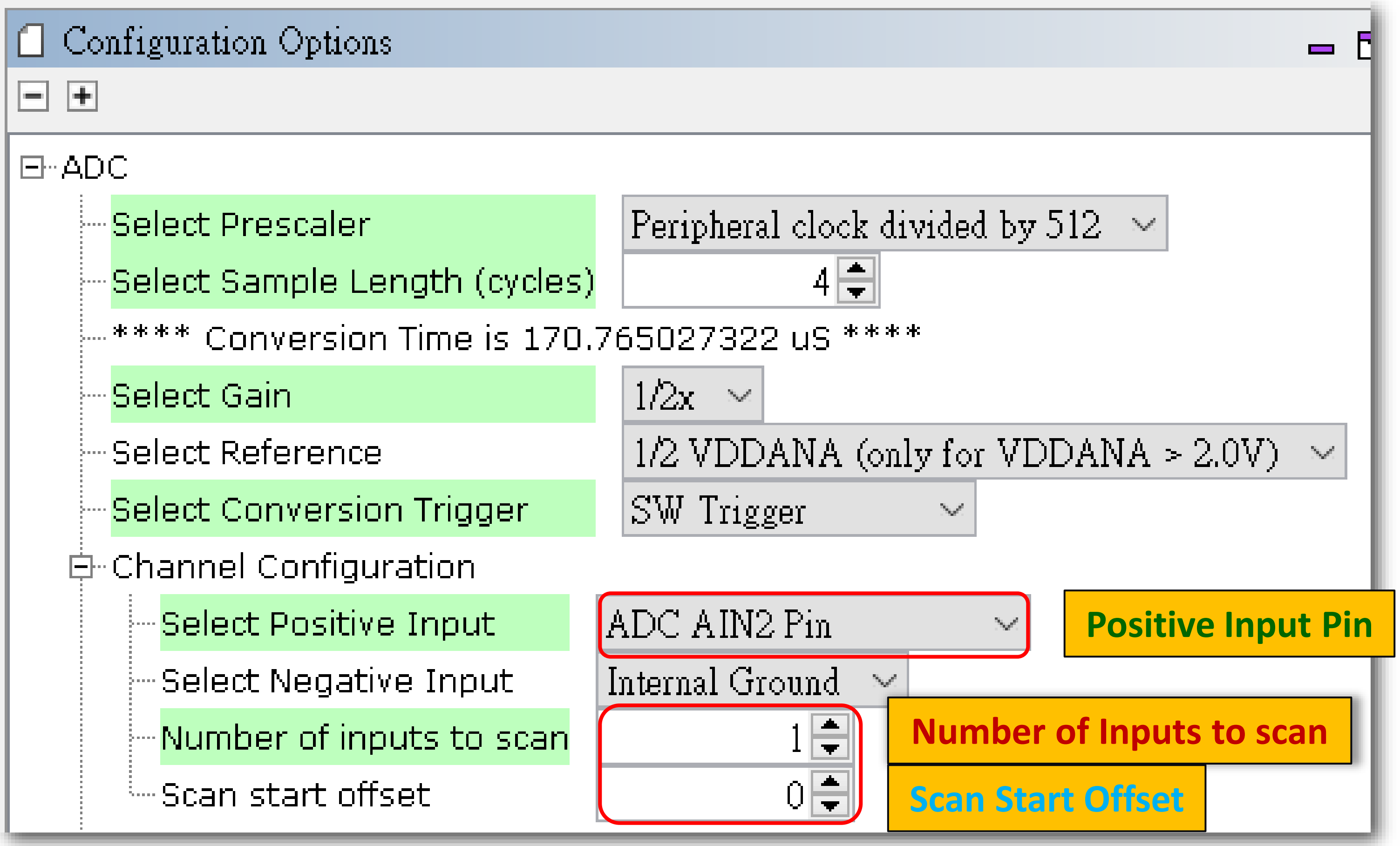
End Pin = AIN[2+1] = AIN[3]

ADC Pin Scan


Positive
AIN[0]
AIN[1]
Positive Input Pin = 2
AIN[2]
AIN[3]
AIN[8]
AIN[9]
AIN[10]
AIN[19]
Temperature Sensor
DAC output
Scan Start Offset = 0
Number of Inputs to scan = 1
Gain
ADC

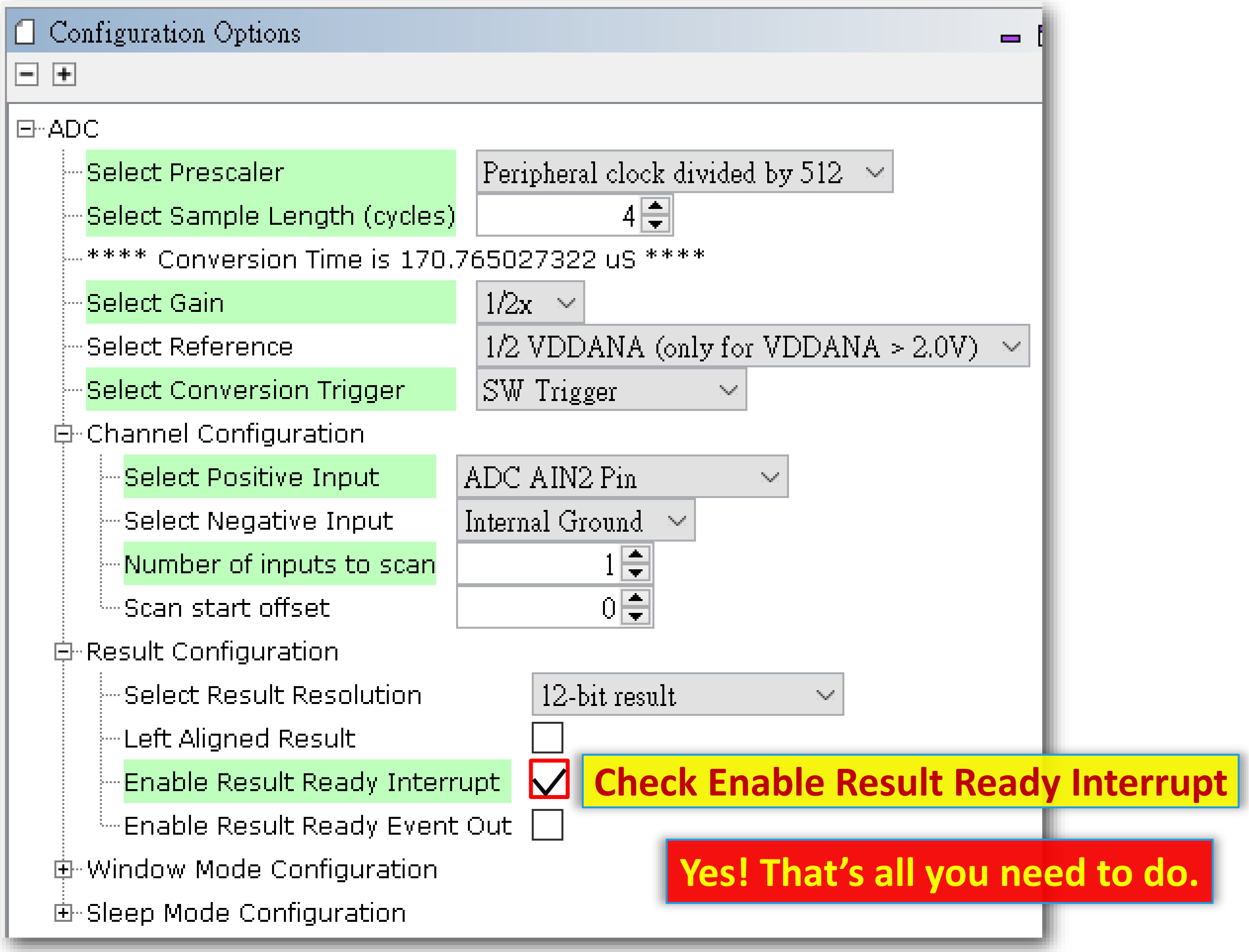
ADC Pin Scan

Configuration Options Example



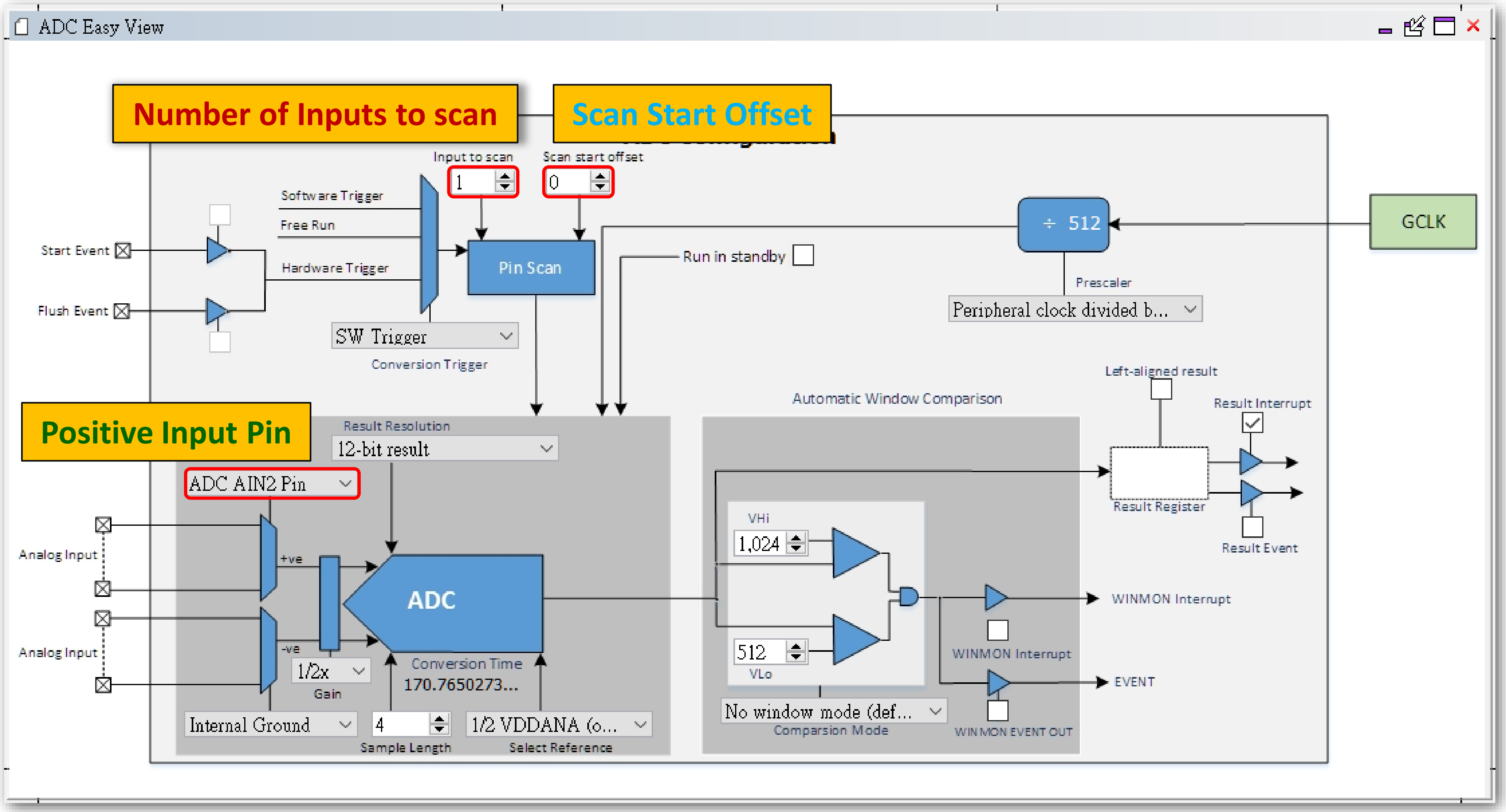
ADC Interrupt

Configuration Options Example



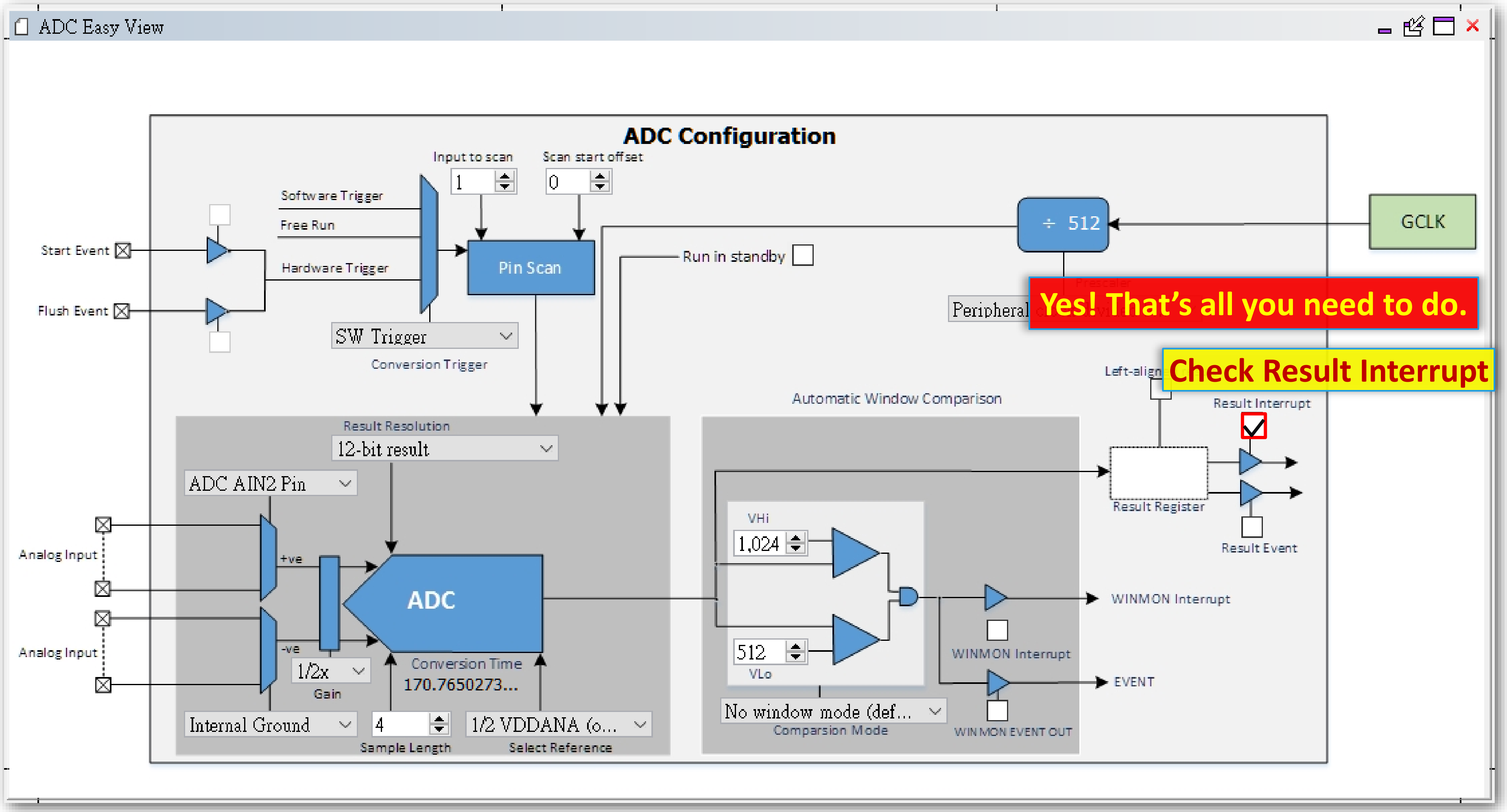
ADC Pin Scan

Easy View Configuration Example



ADC Interrupt

Easy View Configuration Example



MICROCHIP

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(1)
   {   …
   }
}
```

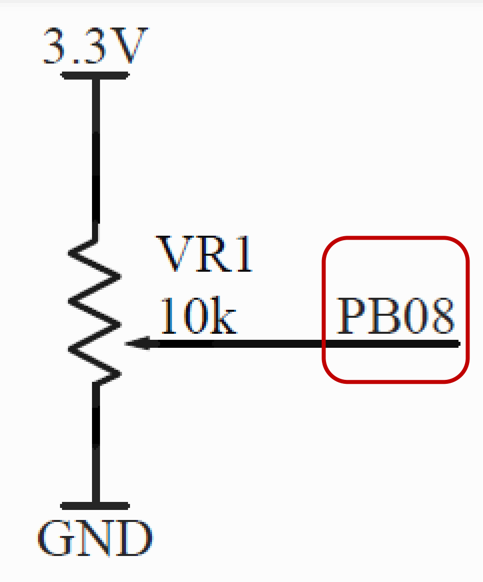
Lab13 ADC Pin Scan

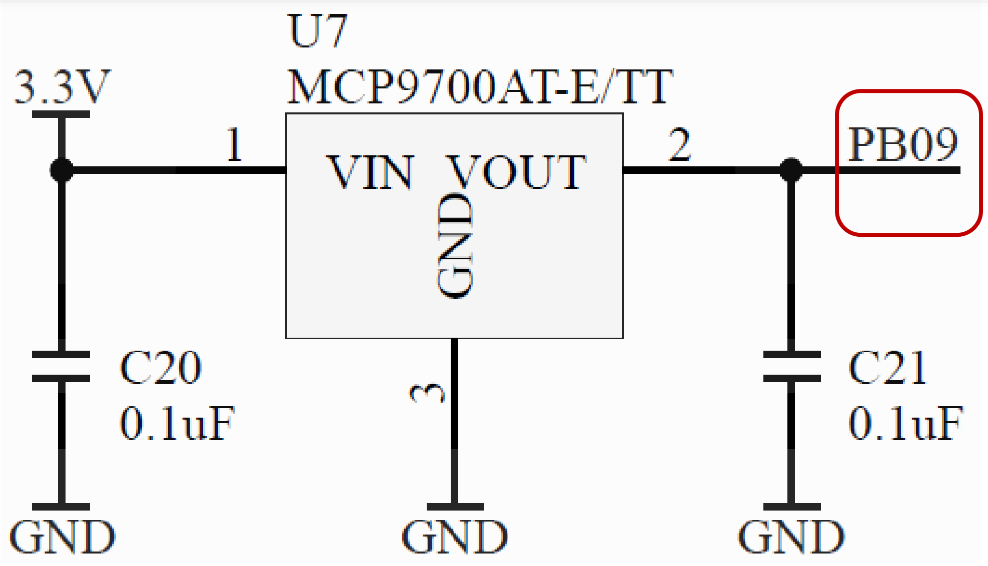
- Please start this Lab from Lab12 (do not from Lab12-1)
- Change ADC polling to Interrupt callback mode and enable pin scan function.
- Get the ADC conversion value from VR1(AIN[2]) and Analog Temperature Sensor(AIN[3]) with one second period and then use UART to output ADC values to PC.

- **Let's go!**

Lab13 ADC Pin Scan

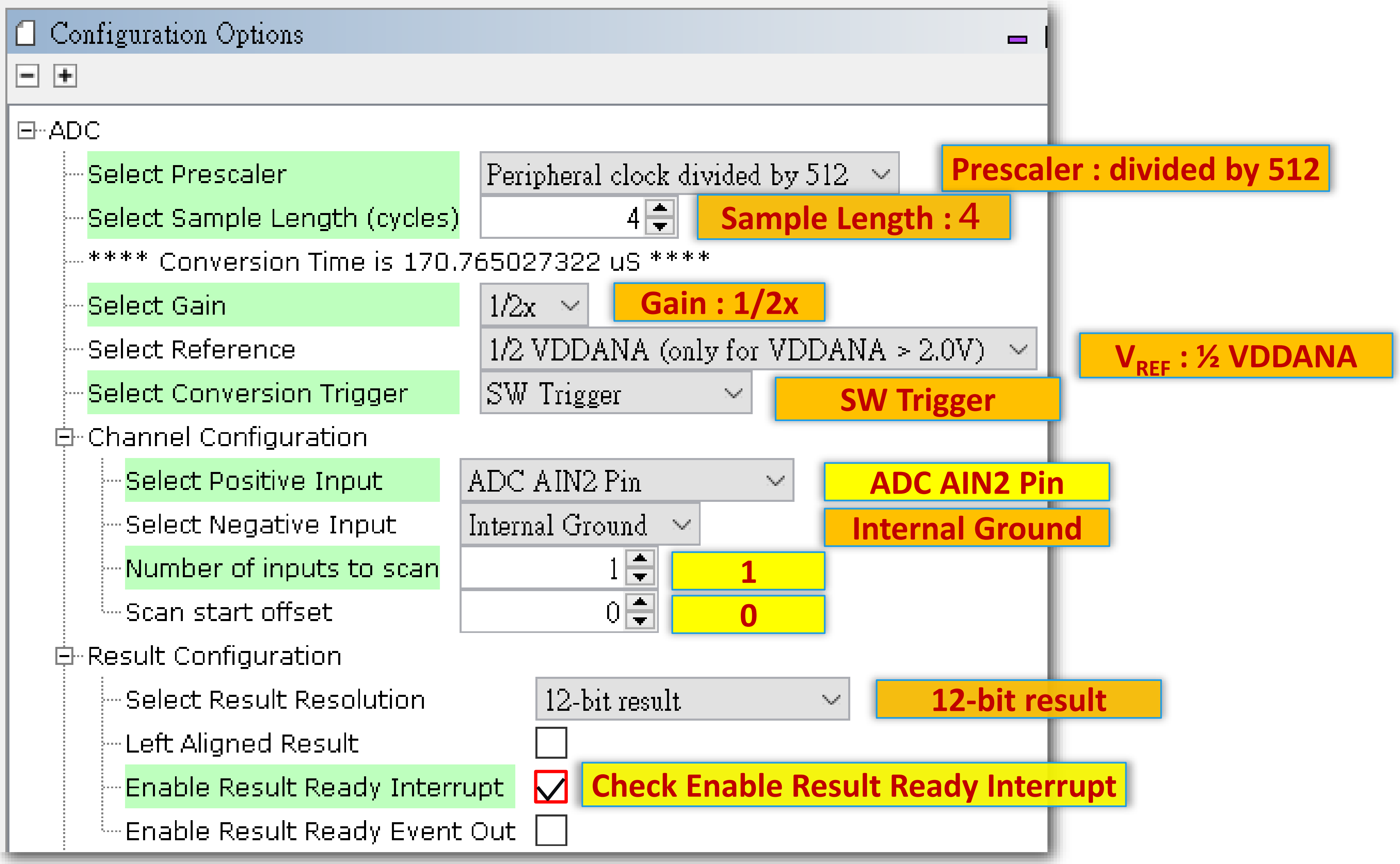
Pin(1)			I/O Pin	Supply	A	B(2)(3)				
SAMD21E	SAMD21G	SAMD21J			EIC	REF	ADC	AC	PTC	DAC
	7	11	PB08	VDDANA	EXTINT[8]		AIN[2]		Y[14]	
	8	12	PB09	VDDANA	EXTINT[9]		AIN[3]		Y[15]	





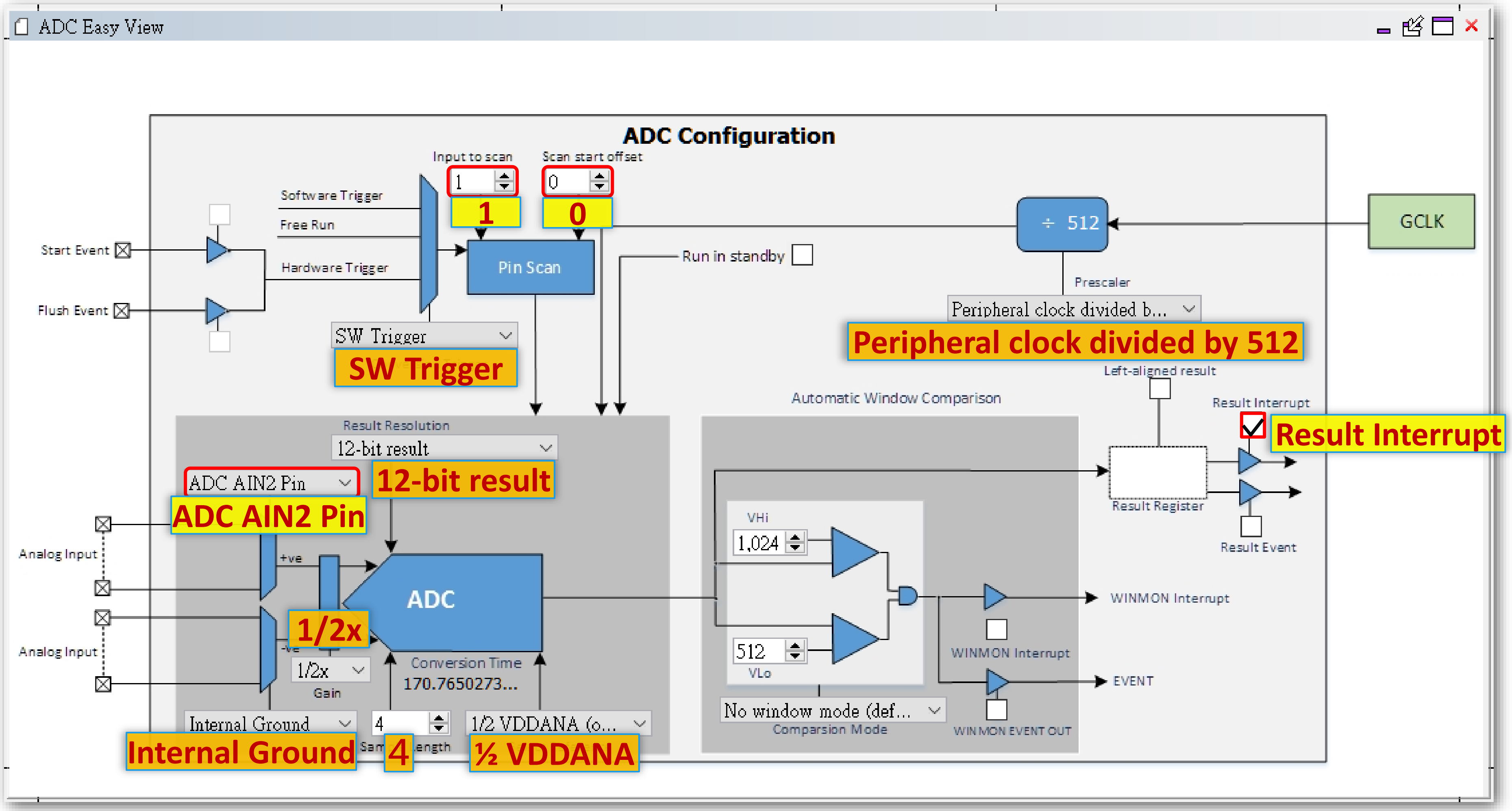
Lab13 ADC Pin Scan

Step 1 (Use Configuration Options)



Lab13 ADC Pin Scan

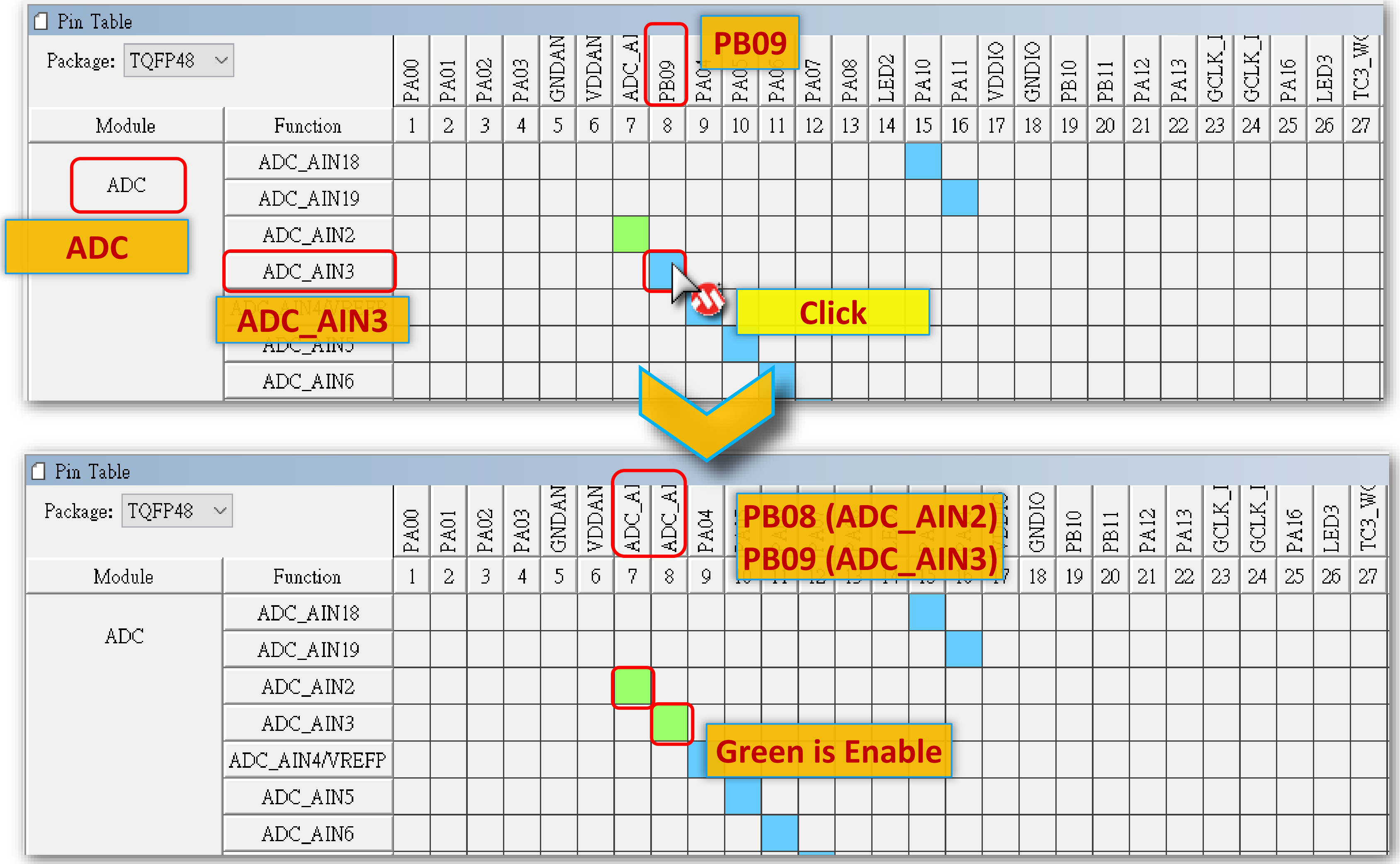
Step 1 (Use Easy View)



MICROCHIP

Lab13 ADC Pin Scan

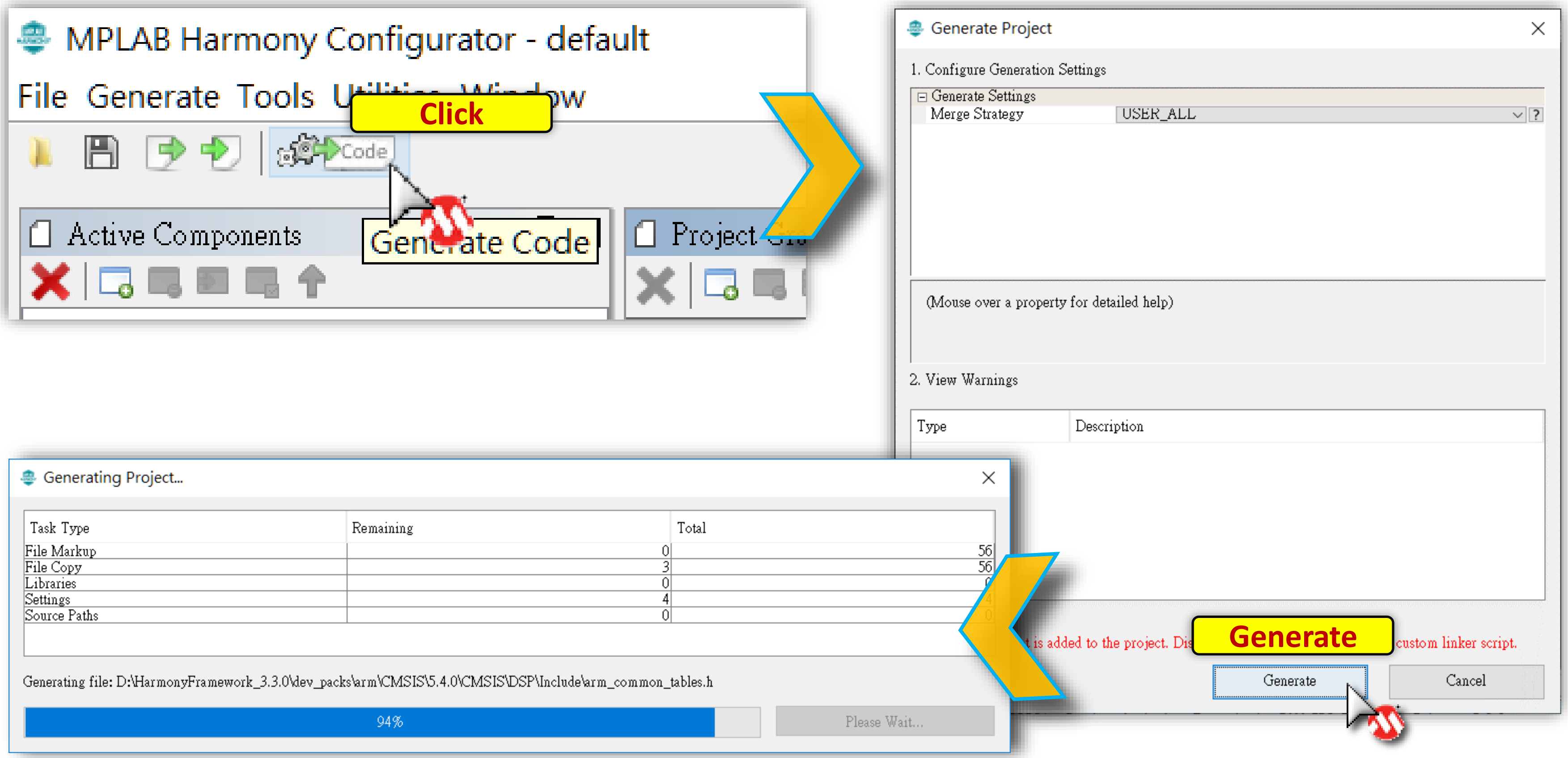
Step 2



Lab13 ADC Pin Scan

Step 3

- **Click  to Generate Code and save changes to MHC configure file.**



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(1)
   {
      …
      if( TC3_IsOverflow )
      {
         // TODO 13.04
         ADC_ConversionStart();
      }
      …
   }
}
```

Lab13 ADC Pin Scan

Step 6

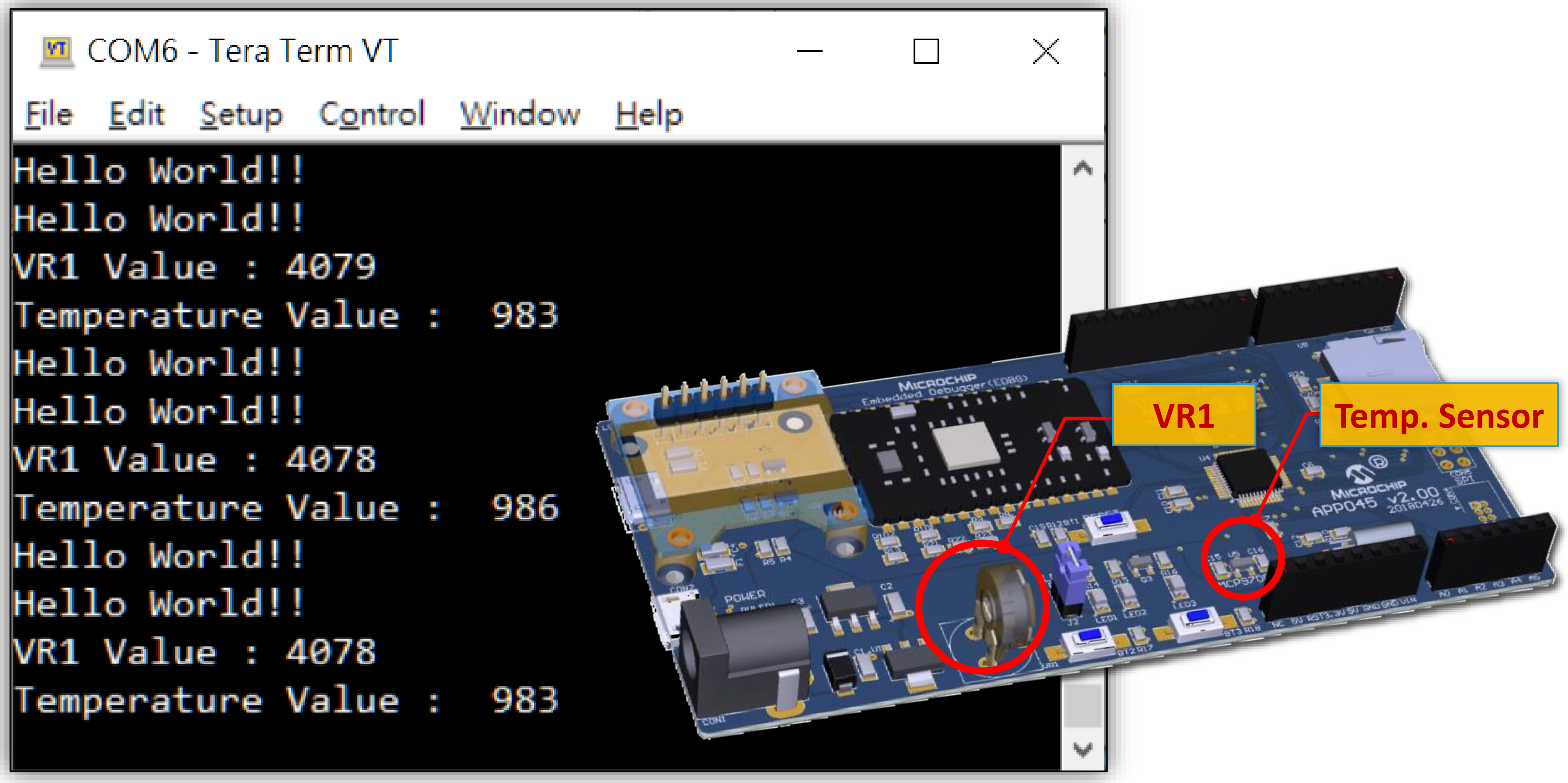
a Add code segment to your main.c

```
…

int main (void)
{
   …

   while(1)
   {
      …
      // TODO 13.05
      if( ADC_IsCompleted )
      {
         ADC_IsCompleted = 0;

         myprintf( "VR1 Value : %4d\r\n", ADC_Result[0] );
         myprintf( "Temperature Value : %4d\r\n", ADC_Result[1] );
      }

      …
   }
}
```

Lab13 ADC Pin Scan

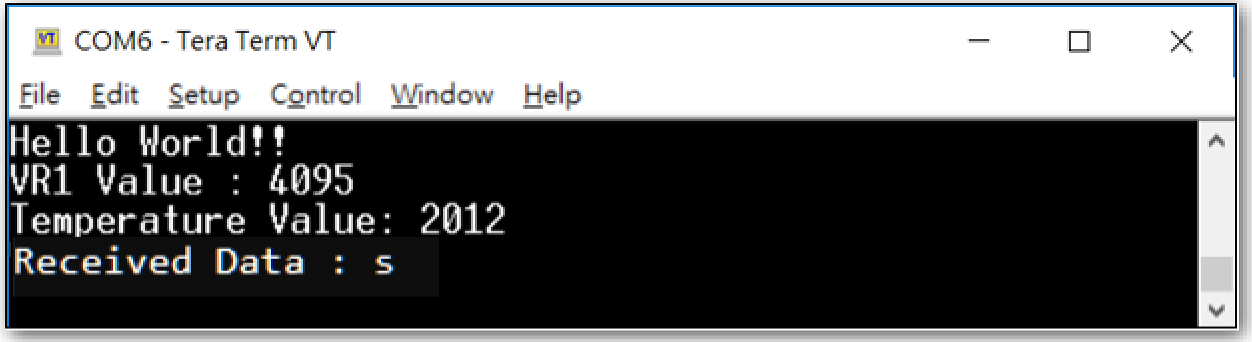
Result

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



VT100 Console Control

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



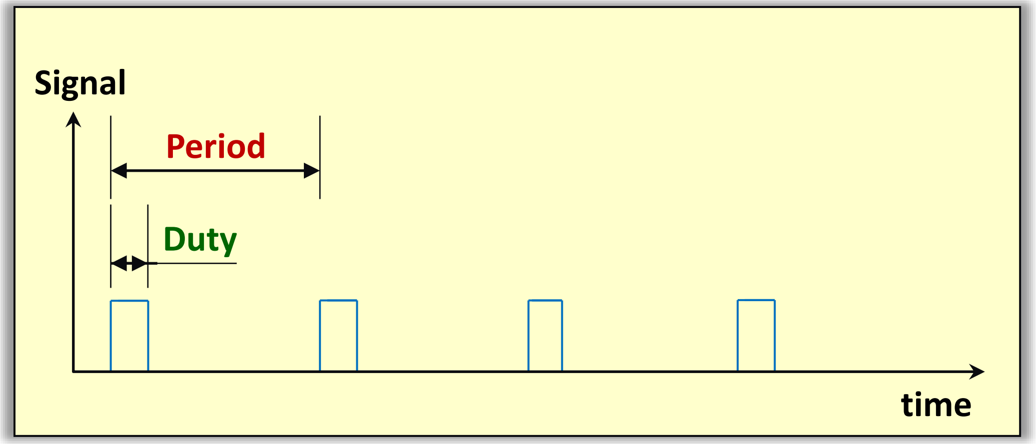

```
myprintf("\033[2J");     // Clear screen
myprintf("\033[?25l");   // Hide cursor
myprintf("\033[y;xH");   // Set cursor to (x,y)
                         // Left-up corner is from (1,1)
```

- **Let's go!**

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

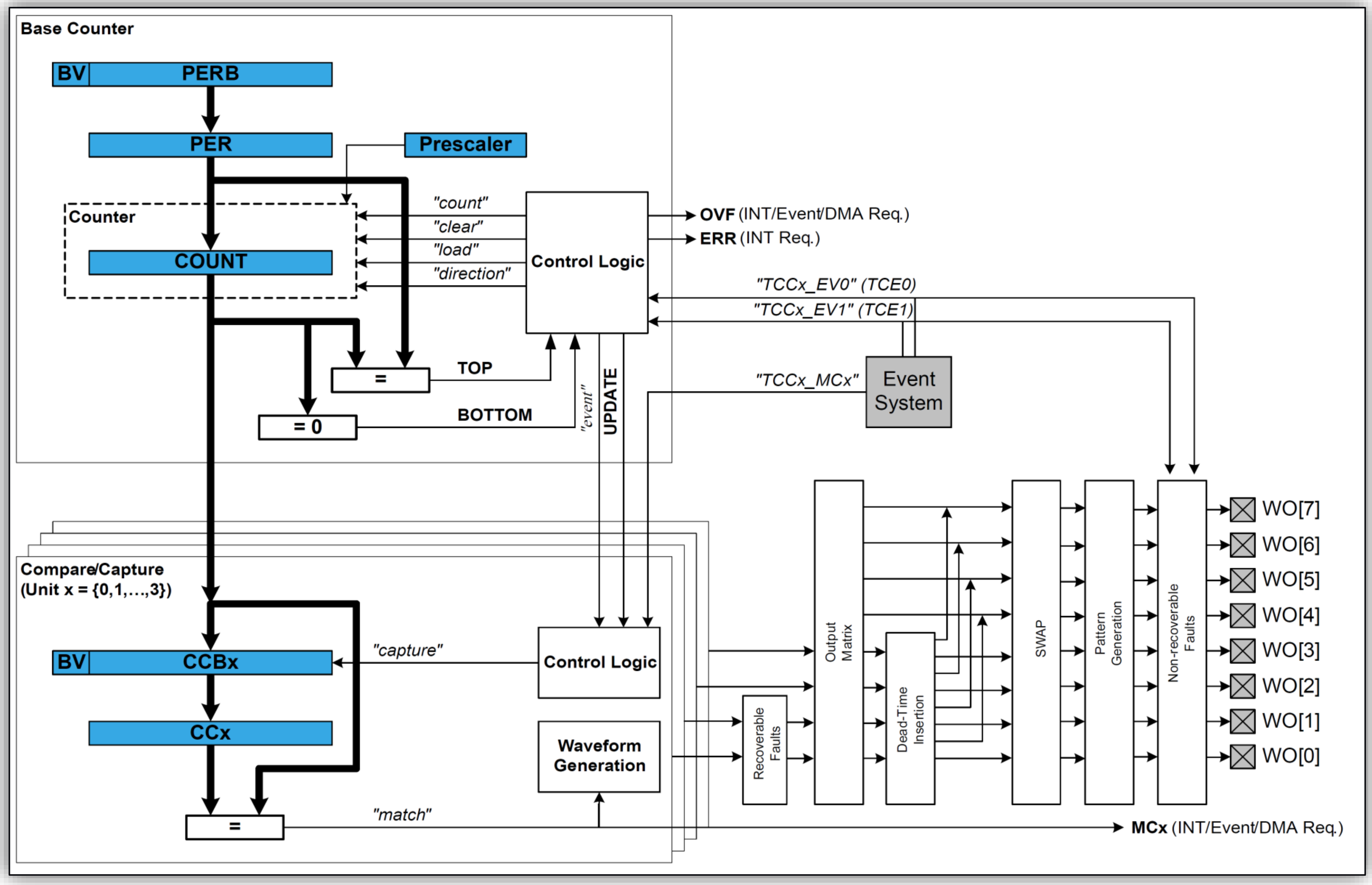


MICROCHIP

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



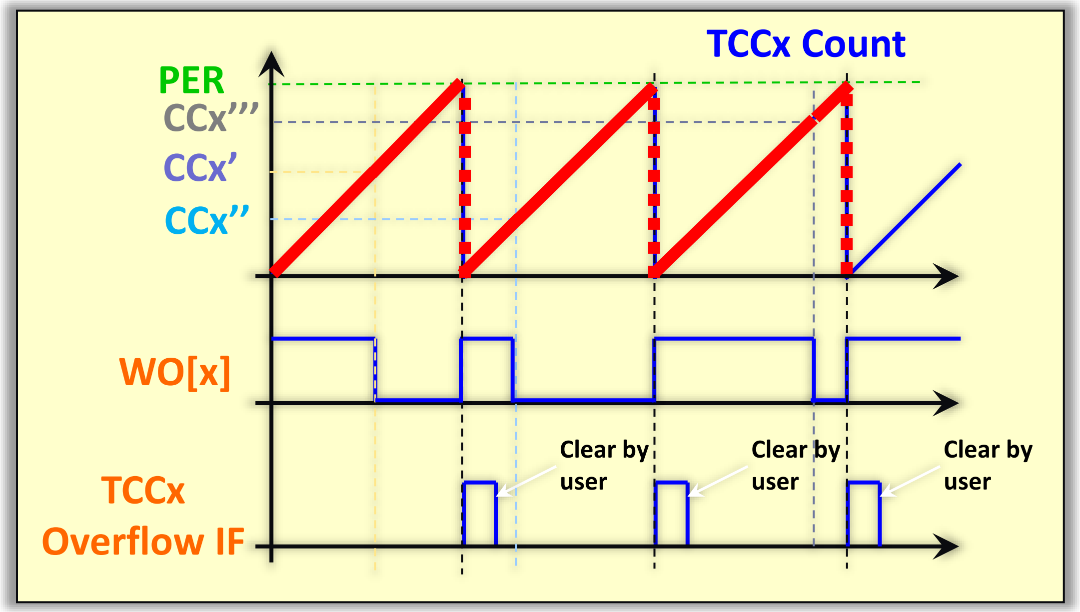
TCCx, Channel and WO[x]

- **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.



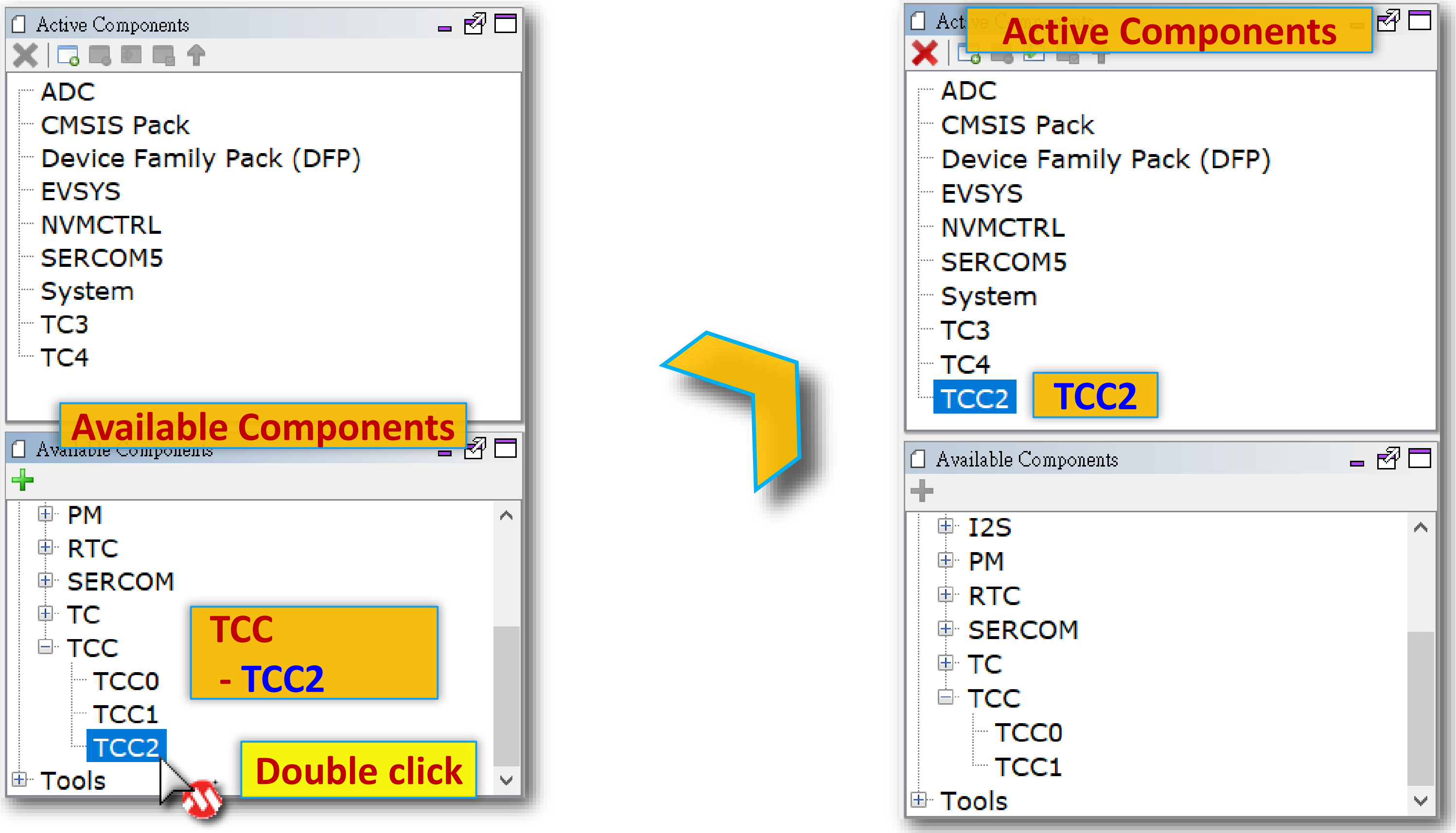
Rule

1.TCCx Counter == CCx -> WO[x] Clear to low

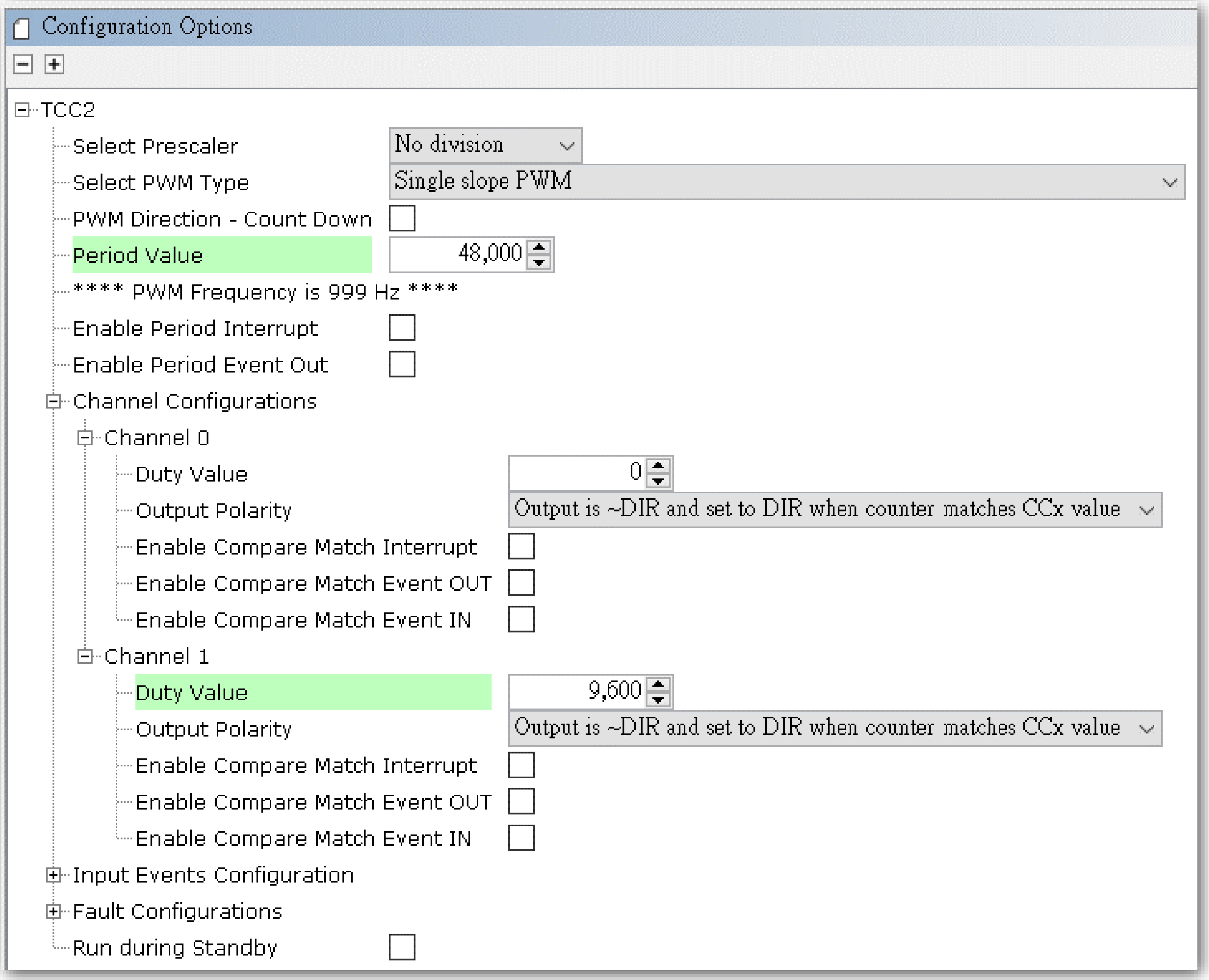
2.TCCx Counter == PER -> WO[x] Set to high

Add TCC Function using MHC

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



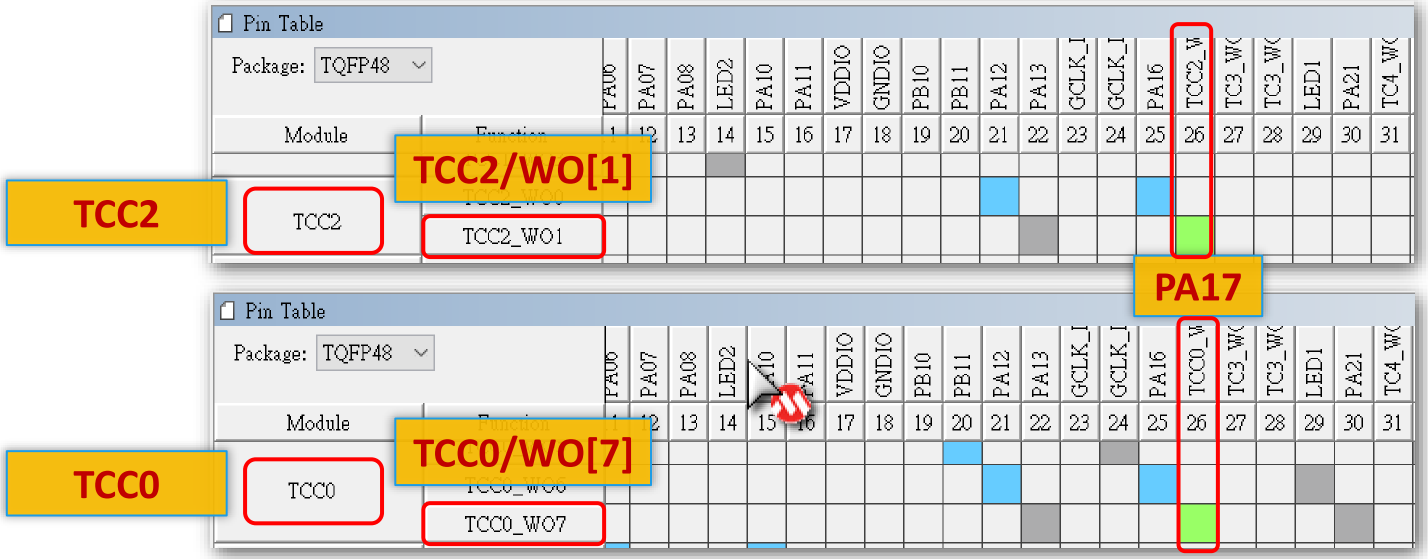
TCC Configuration Options Example



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			EIC	REF	ADC	AC	PTC	DAC	SERCOM[2][3]	SERCOM-ALT	TC[4] /TCC	TCC	COM	AC/ GCLK
18	26	PA17	PA17	VDDIO	EXTINT[1]				X[5]		TCC2/WO[1]		TCC2/WO[1]	TCC0/WO[7]	TCC0/WO[7]	



TCC Polling

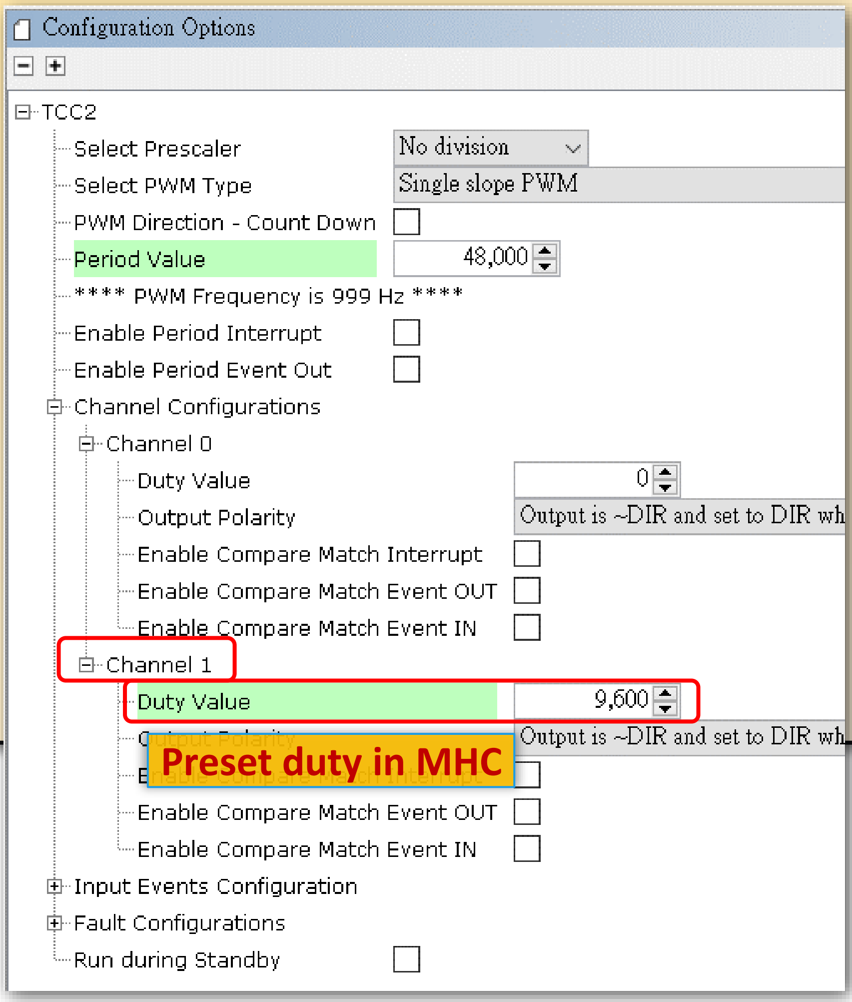
Interface Routines

```
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(1)
   {
      …
   }
}
```

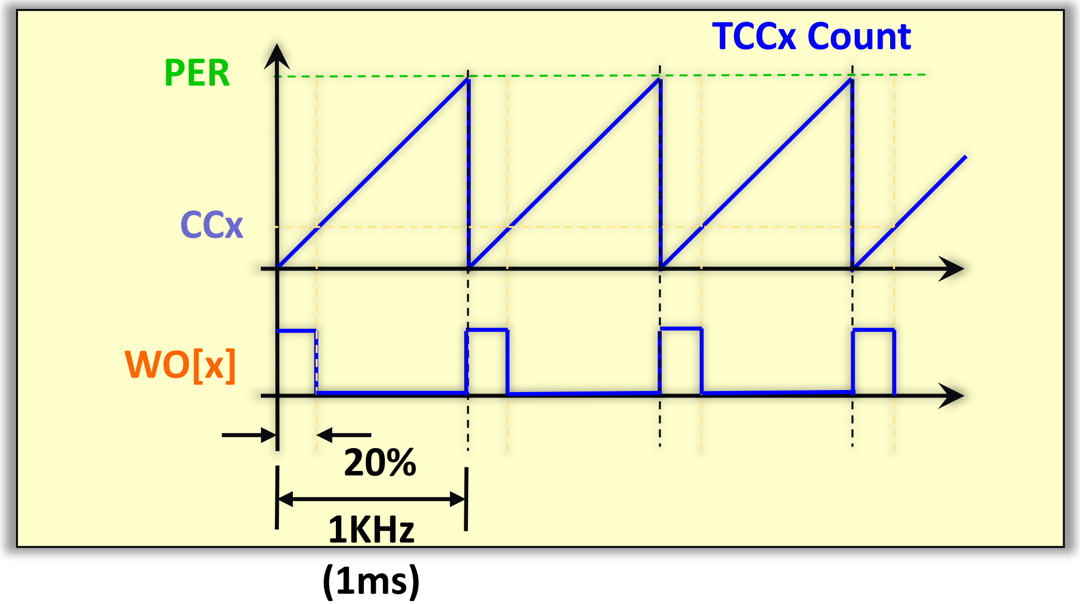
Runtime update duty
Channel 1, Duty 9600



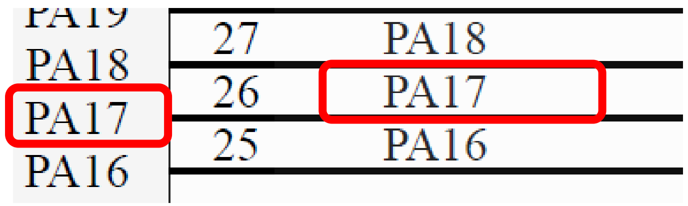
Lab14 TCC NPWM

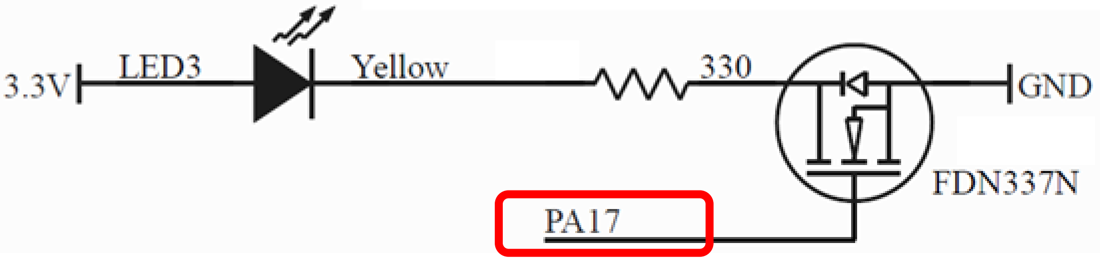
- Add TCC2 NPWM Mode (Single Slope PWM) function to your project.
- Disable LED3(PA17) GPIO function firstly.
- Configure TCC2 to output PWM waveform to LED3(PA17).
- PWM frequency is 1KHz with 20% Duty.

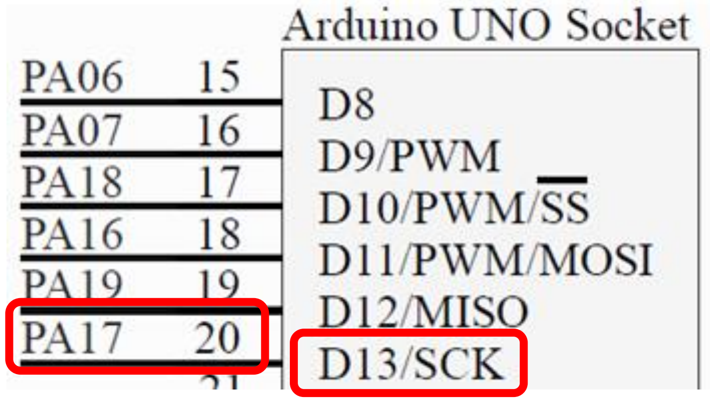
- **Let's go!**

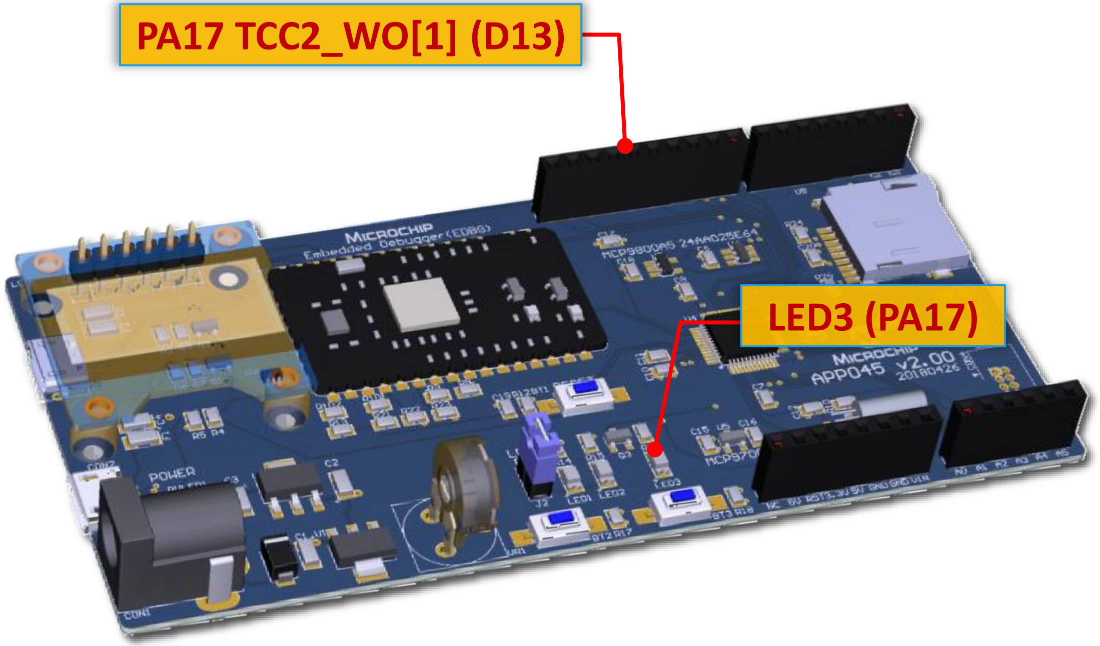


Lab14 TCC NPWM





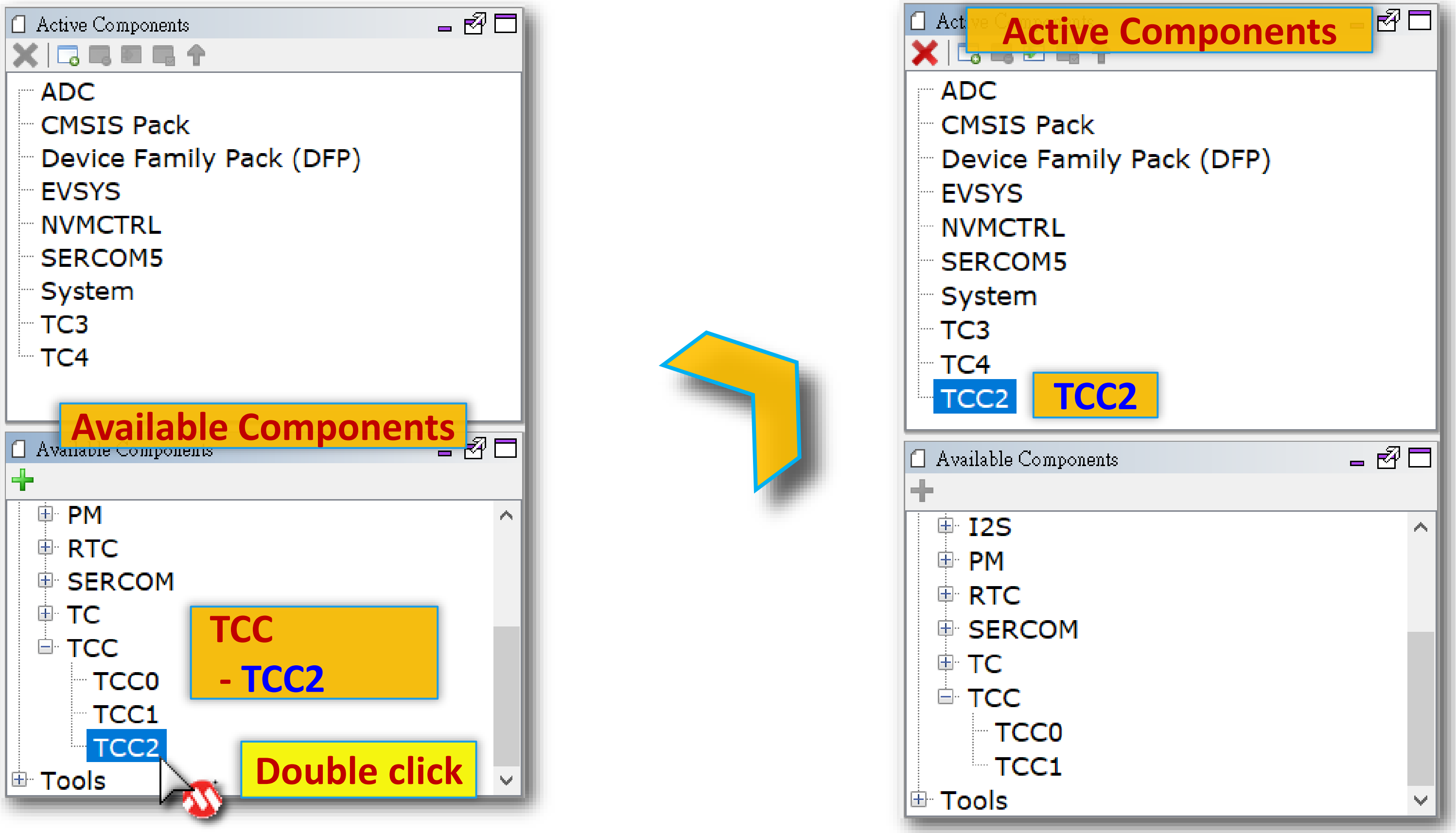




Lab14 TCC NPWM

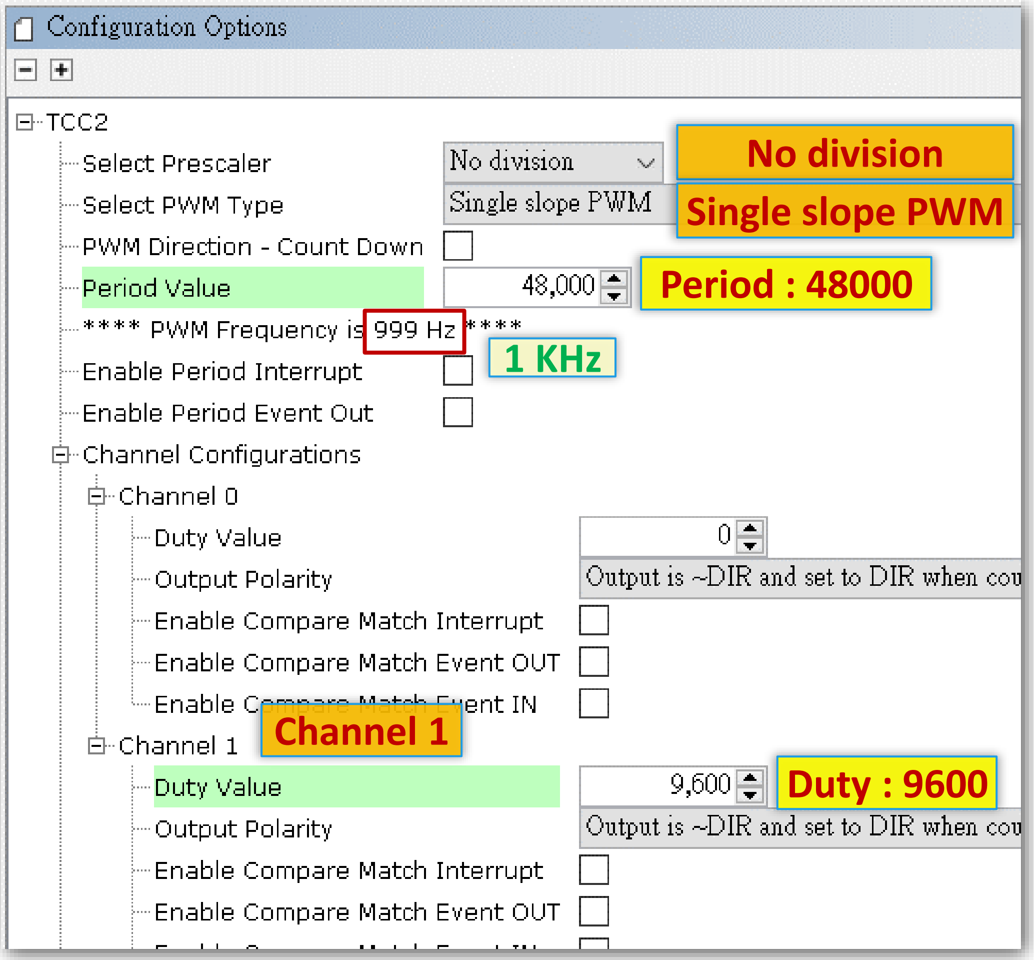
Step 0

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



Lab14 TCC NPWM

Step 1



System Clock = 48MHz

One second counts of system clock :

48,000,000 counts

One second counts after Prescaler :

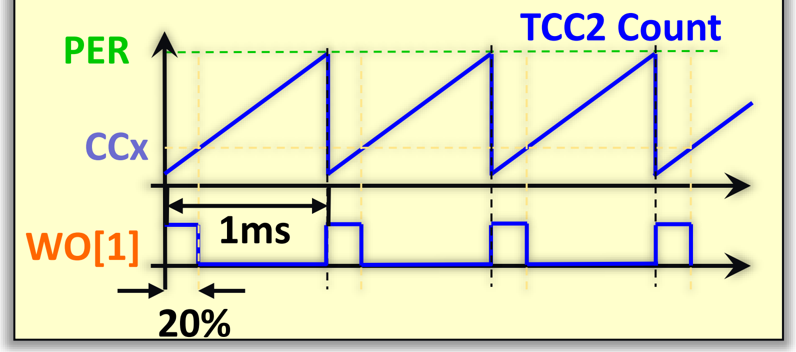
48000000 / 1 48,000,000 counts

1ms counts for LAB14 period target :

48000000 / 1000 48,000 counts

1/5ms counts for LAB14 duty target :

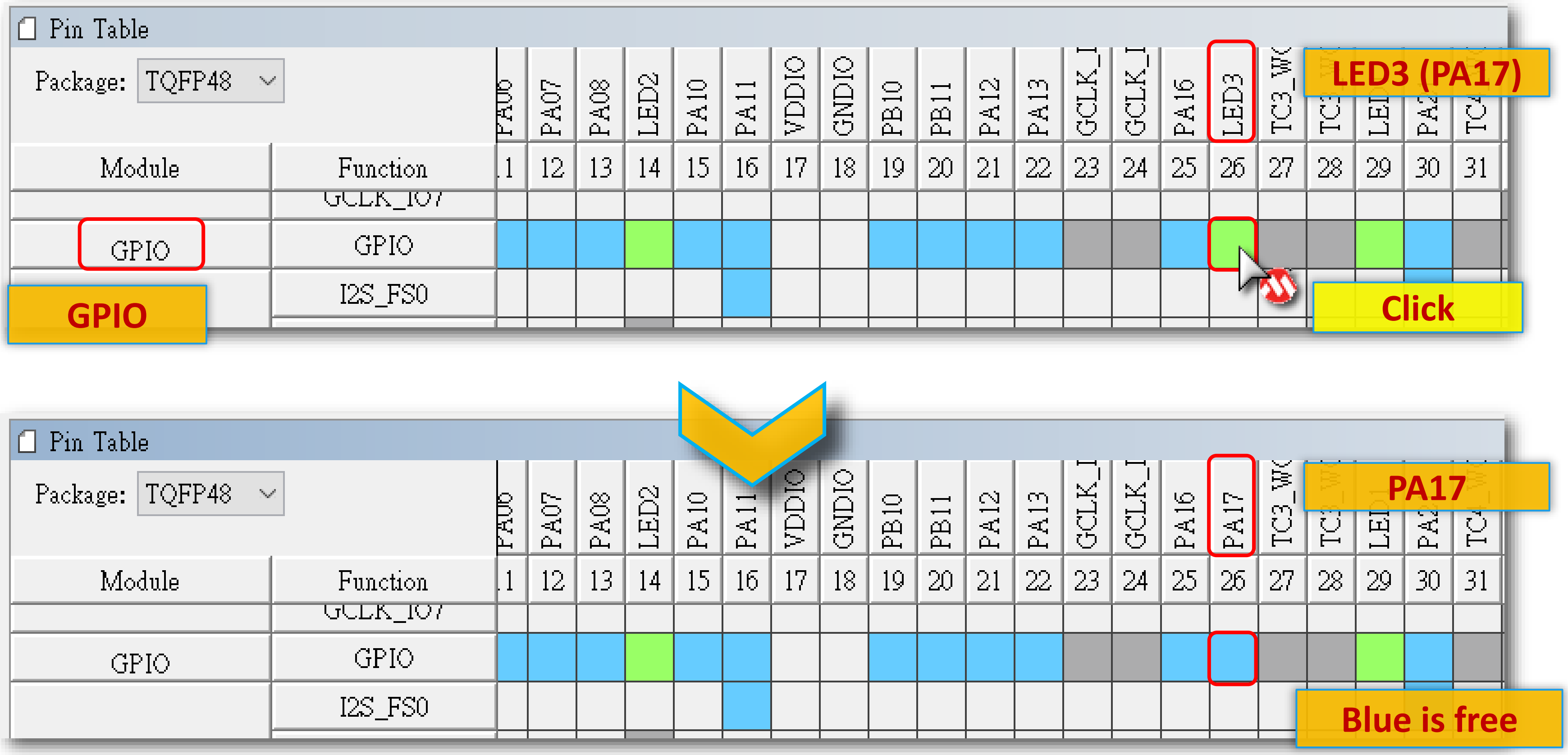
48000 / 5 9,600 counts



Lab14 TCC NPWM

Step 2

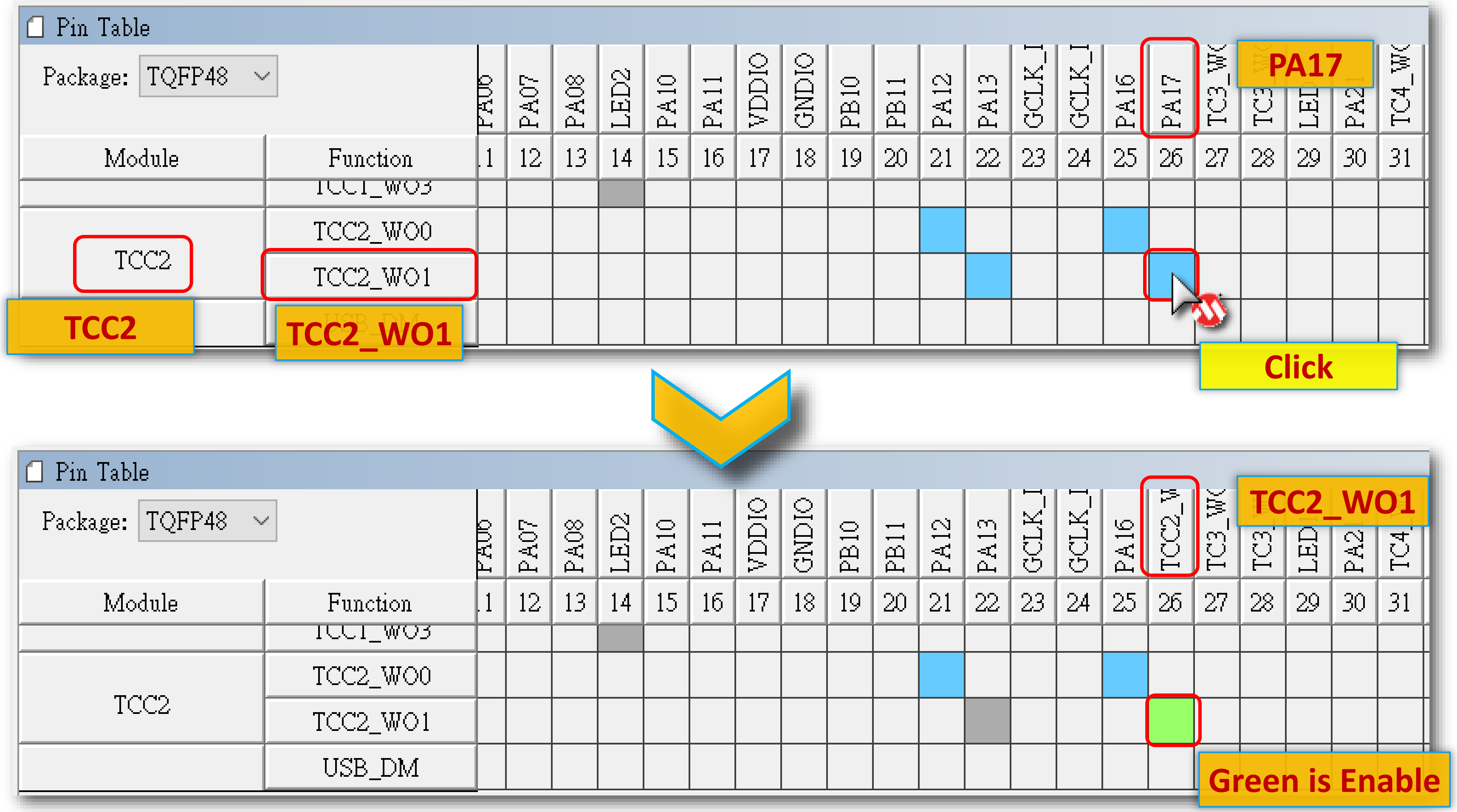
- **Disable LED3(PA17) GPIO function firstly.**



Lab14 TCC NPWM

Step 3

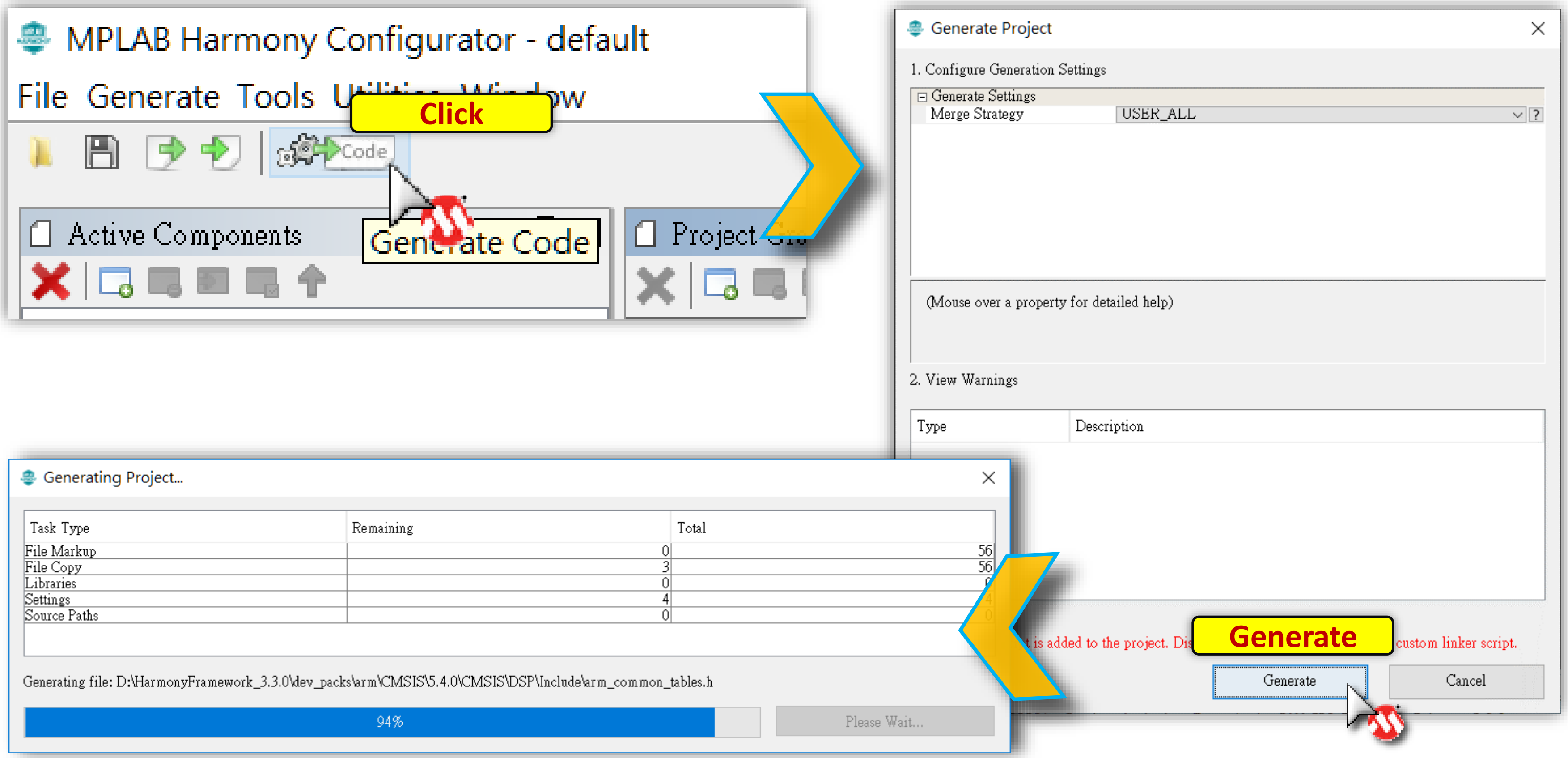
- **Enable TCC2 Waveform Output 1 (TCC2_WO1).**



Lab14 TCC NPWM

Step 4

- Click  to Generate Code and save changes to MHC configure file.



Lab14 TCC NPWM

Step 5

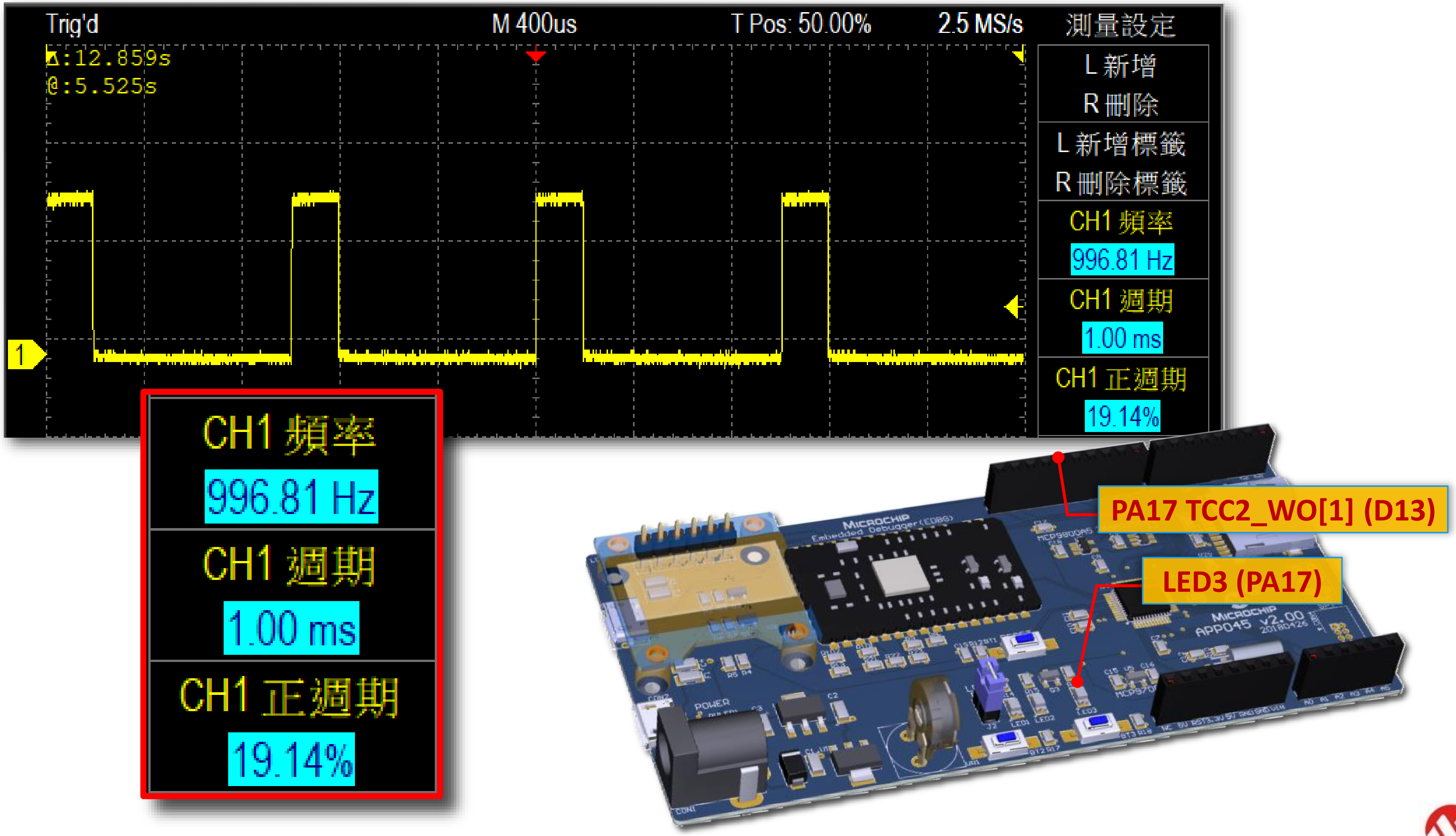
a Add code segment to your main.c

```
…

int main (void)
{
    …

    // TODO 14.01
    TCC2_PWMStart();

    while(1)
    {
        …
    }

}
```

b Program firmware to target board then observe result.

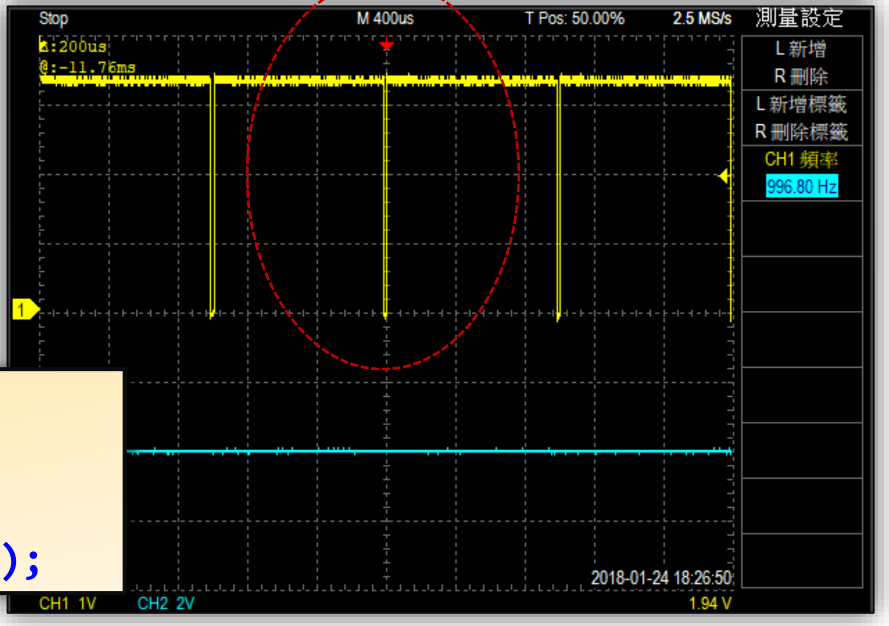
Lab14 TCC NPWM

Result



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 )
{
  // 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

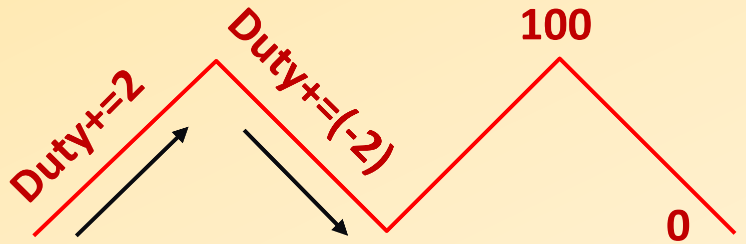
ADC_Result[0] / 4095 = 0.0 ~ 1.0

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)

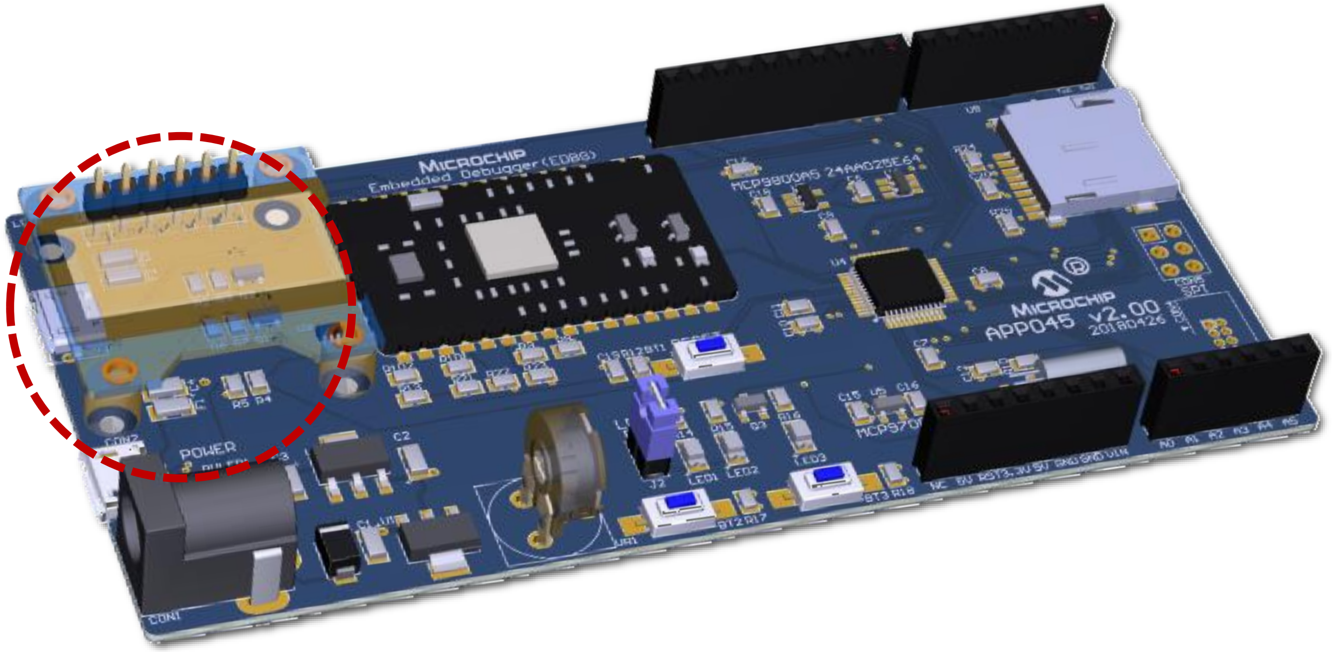
```
// TODO 16.01
uint8_t Duty = 50;
int8_t DutyDistance = 2;

int main (void)
{ …
   if (TC4_IsOverflow)
   { …
     // 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)

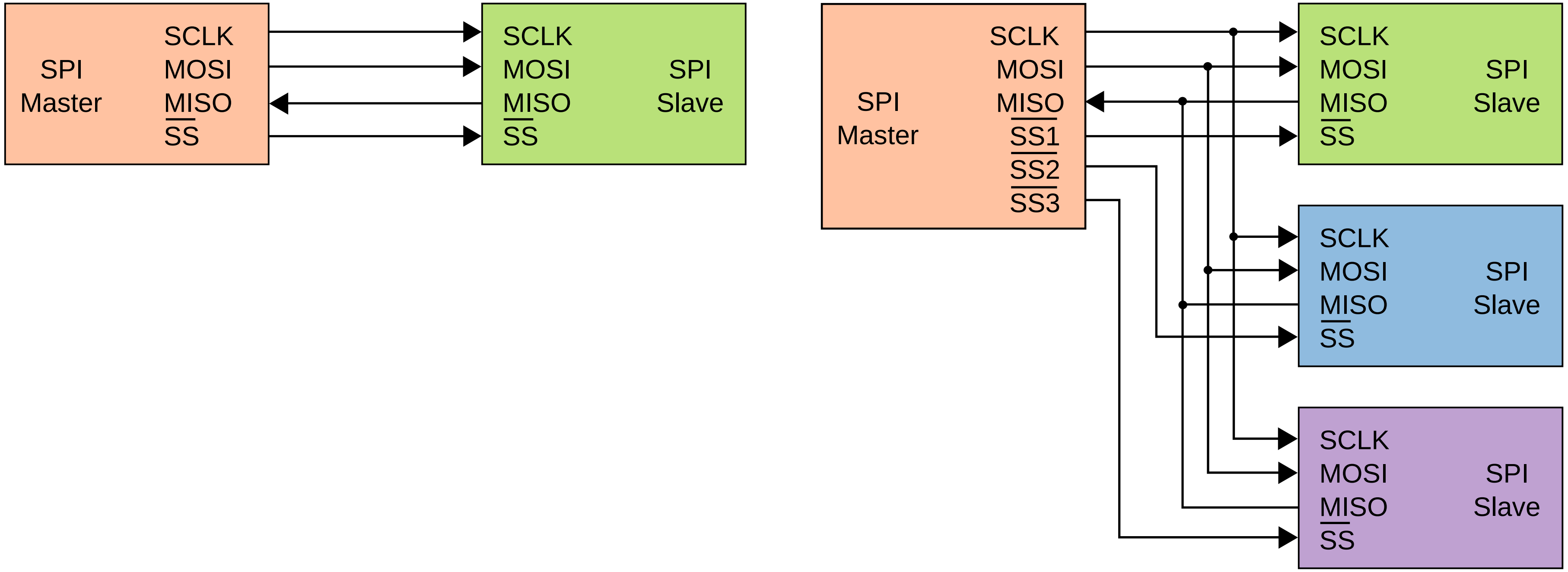
OLED Graphic Display

- **SSD1306**
  128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller and SPI communication interface.



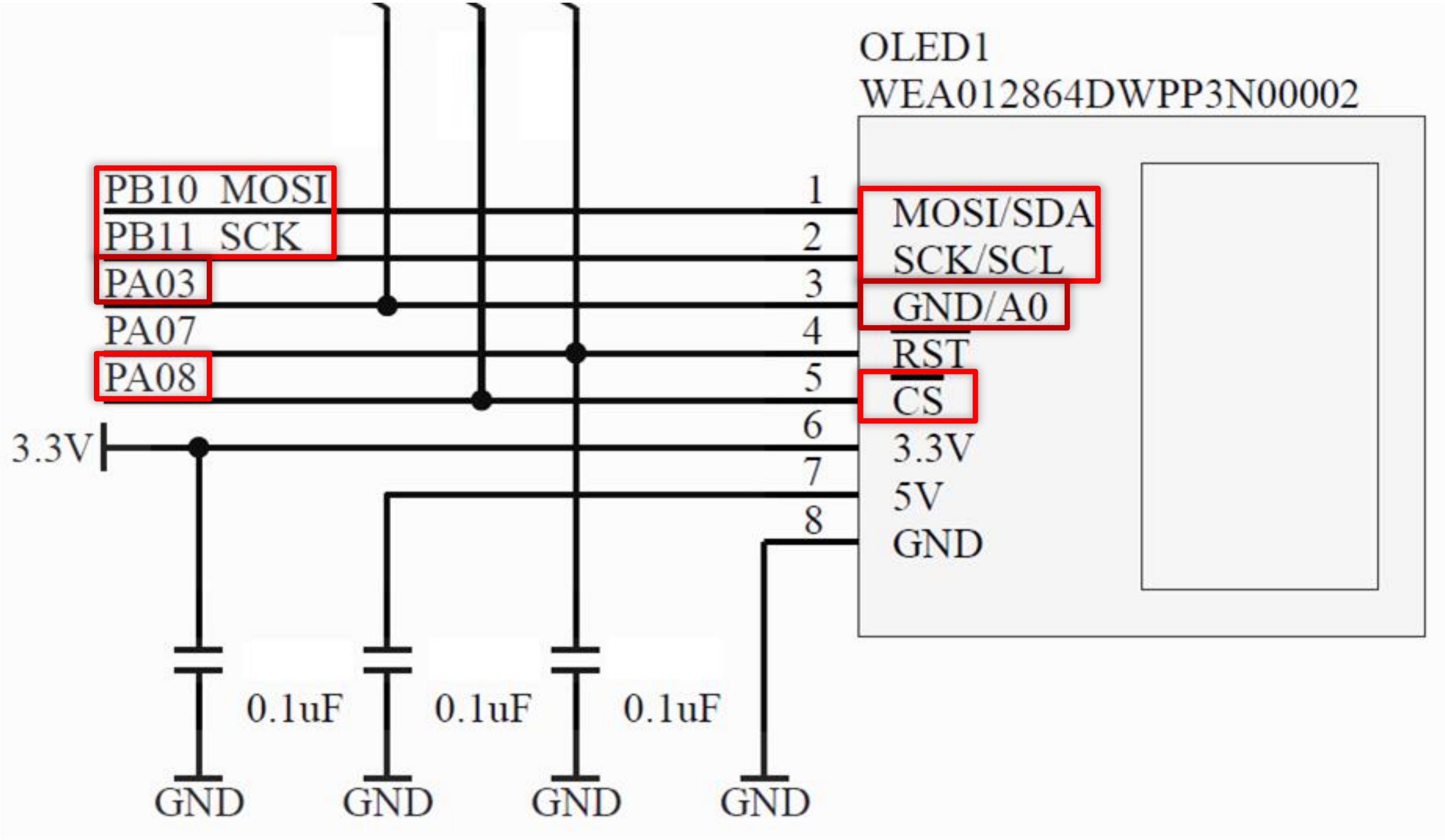
What's SPI

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



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

APP045 OLED SSD1306 Schematic



Lab17 SERCOM - SPI (OLED SSD1306)

- 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

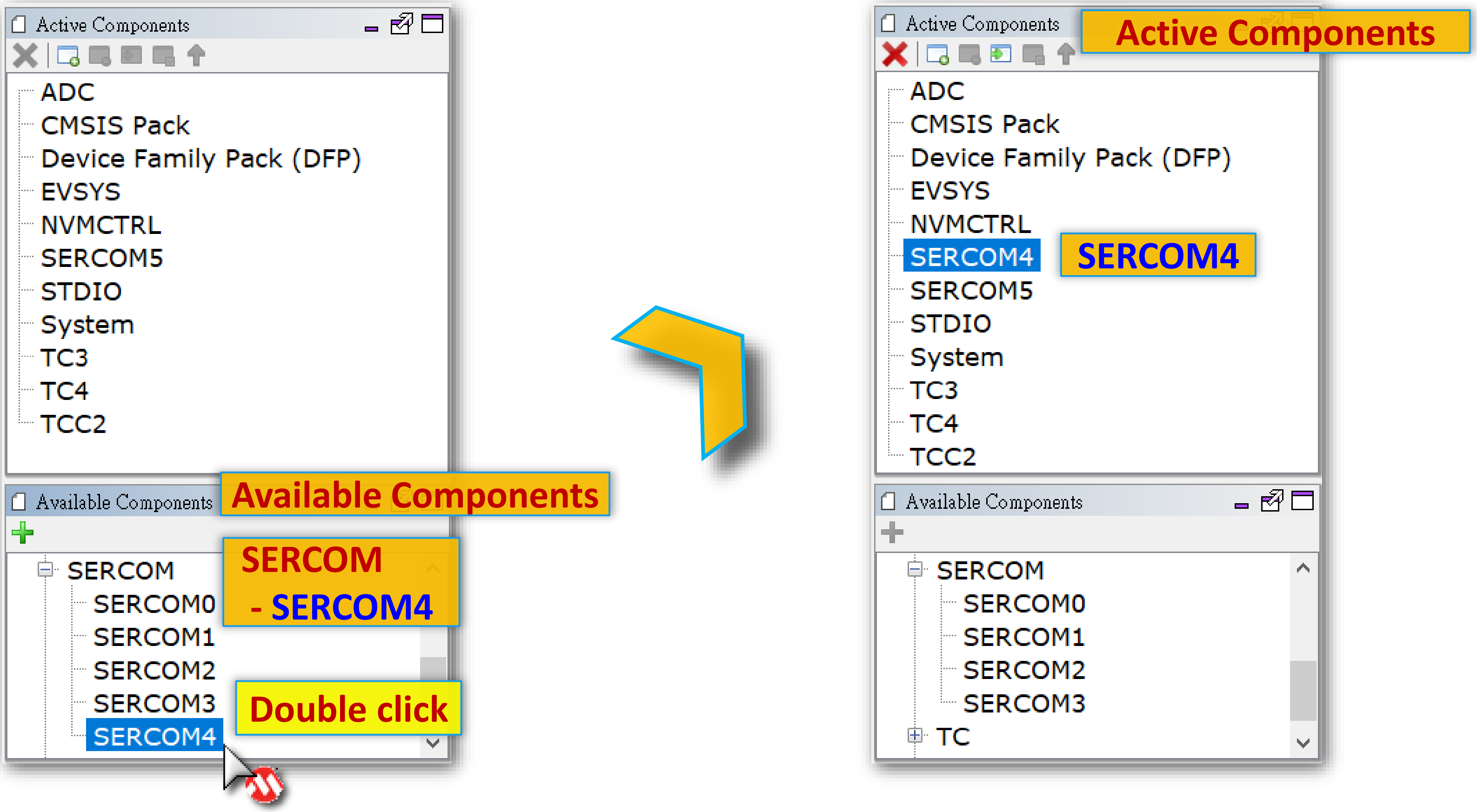
- Let’s go!

MICROCHIP

Lab17 SERCOM - SPI (OLED SSD1306)

Step 1

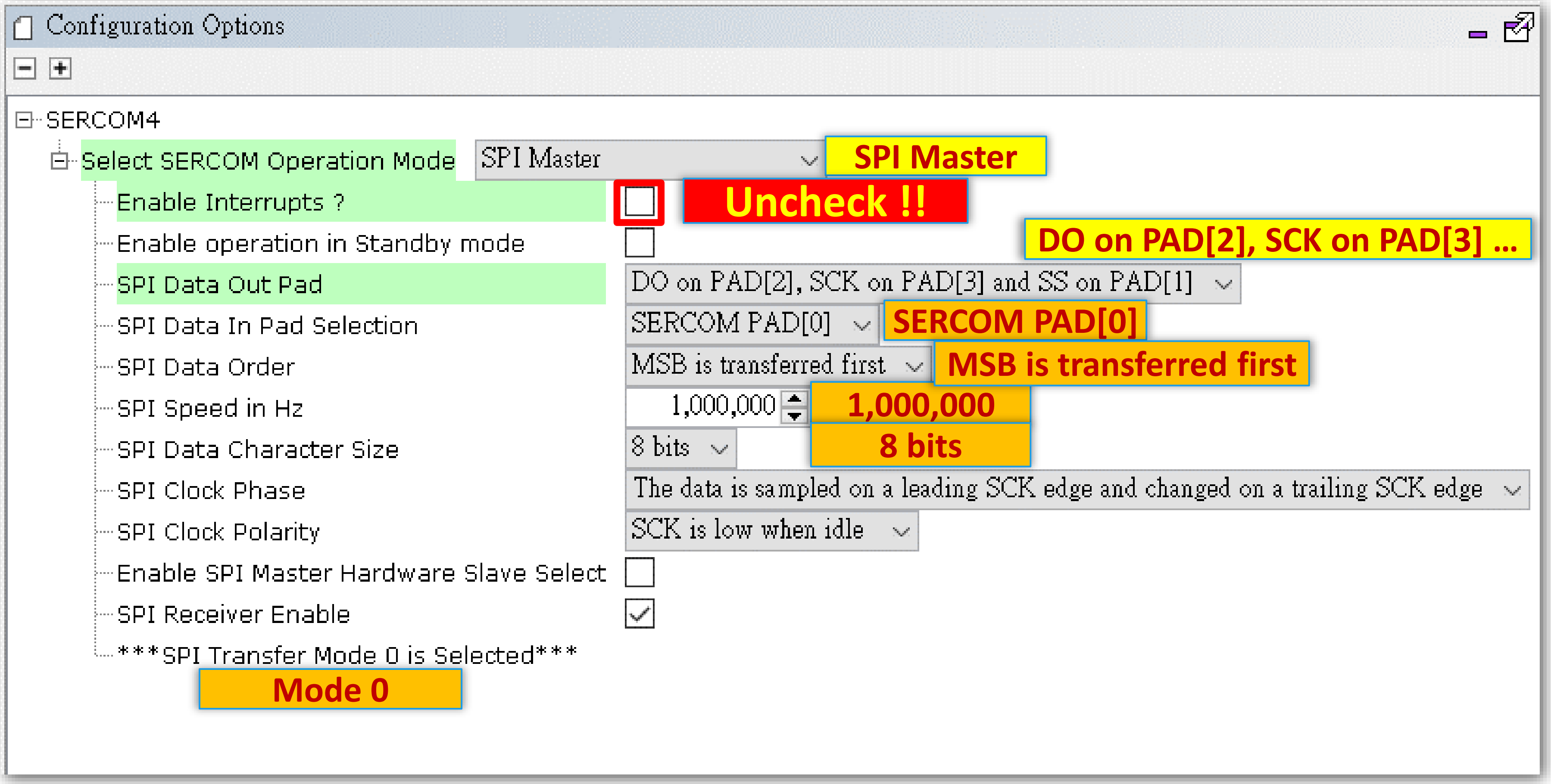
- Find **SERCOM** component in Available Components window.
- Double click **SERCOM4** to add to Active Components window.



Lab17 SERCOM - SPI SSD1306 OLED

Step 2

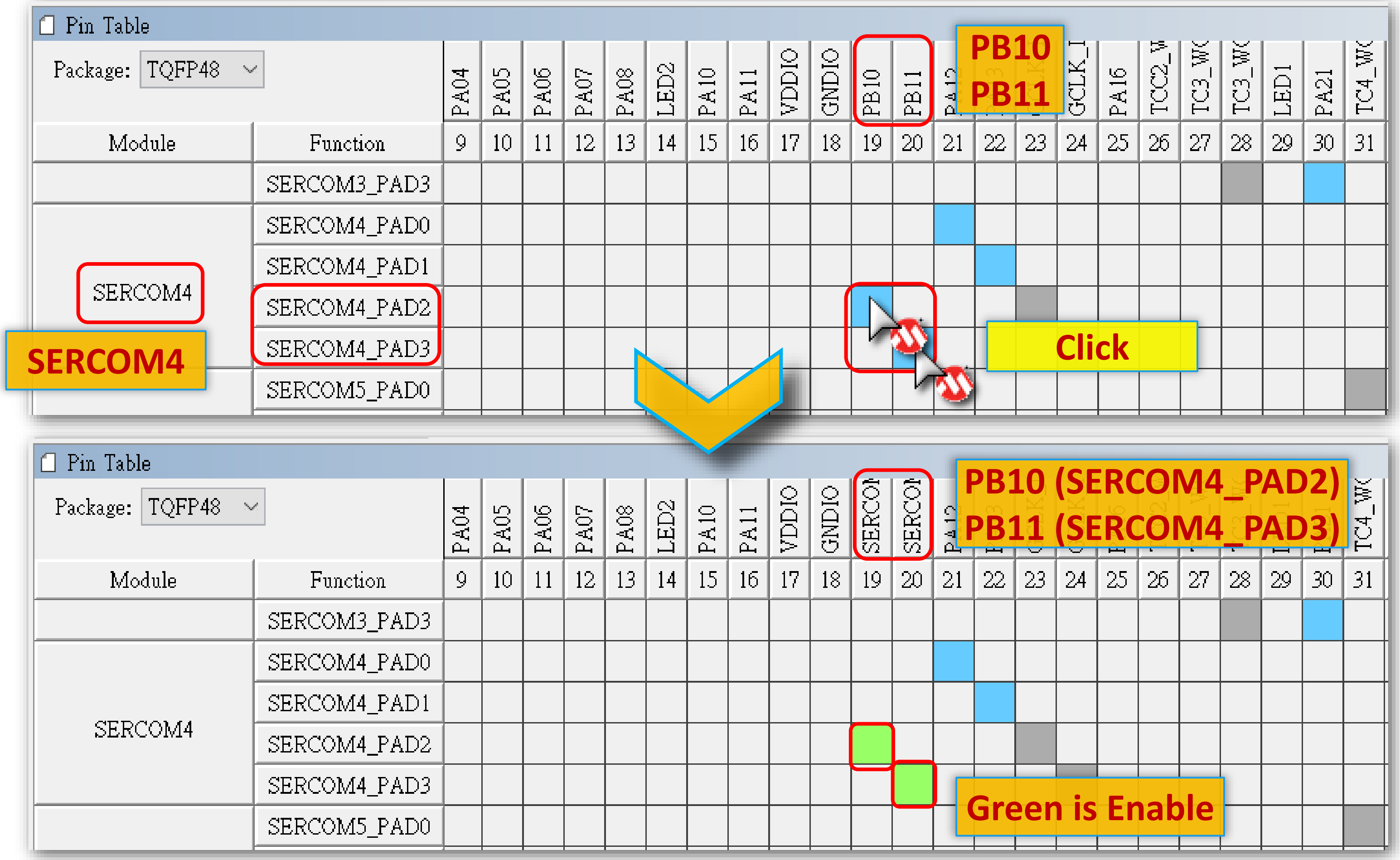
- **Setup SERCOM4 SPI Configuration.**



Lab17 SERCOM - SPI SSD1306 OLED

Step 3

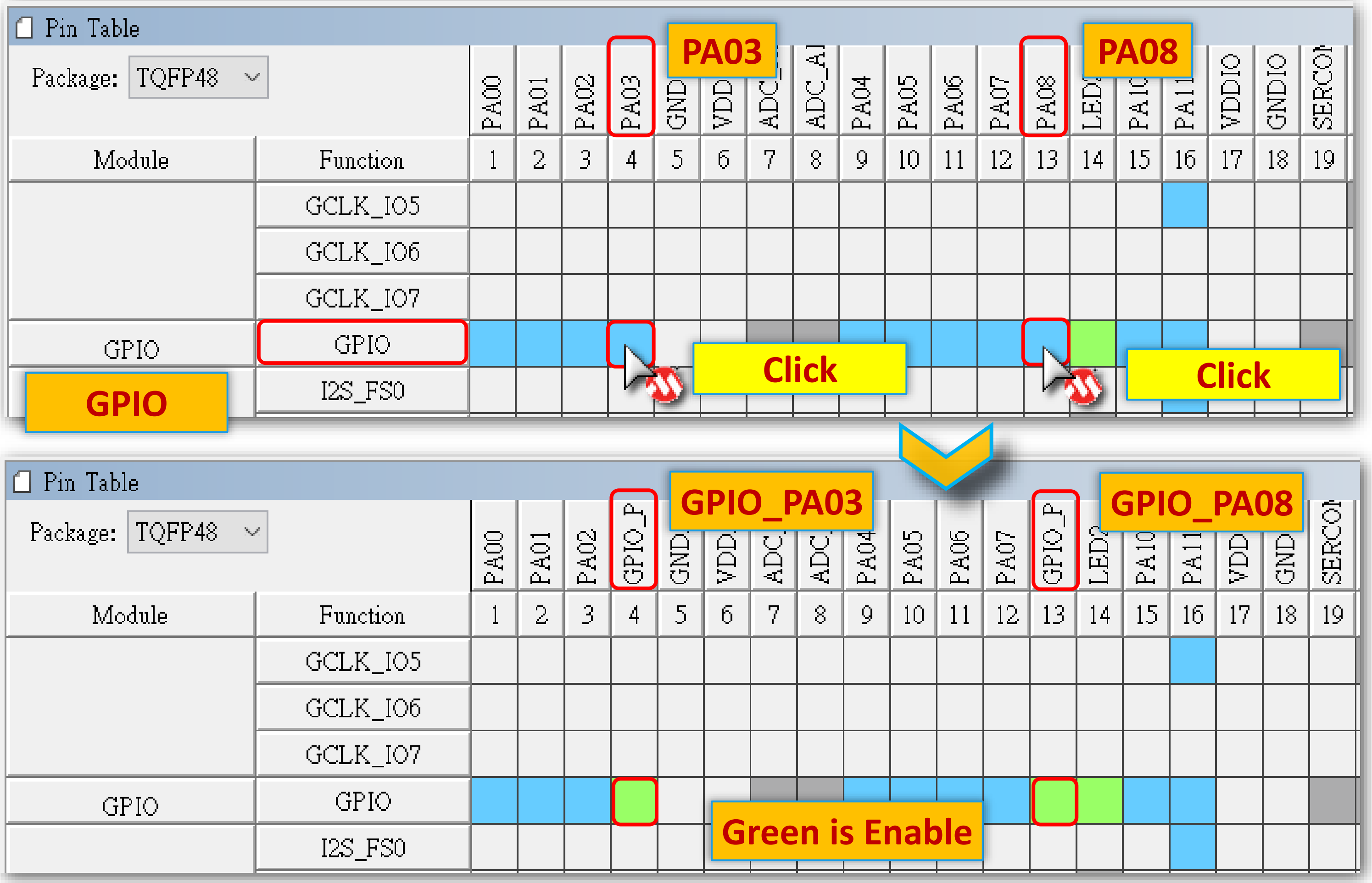
- **Pin Table Configuration of SERCOM4**



Lab17 SERCOM - SPI SSD1306 OLED

Step 4

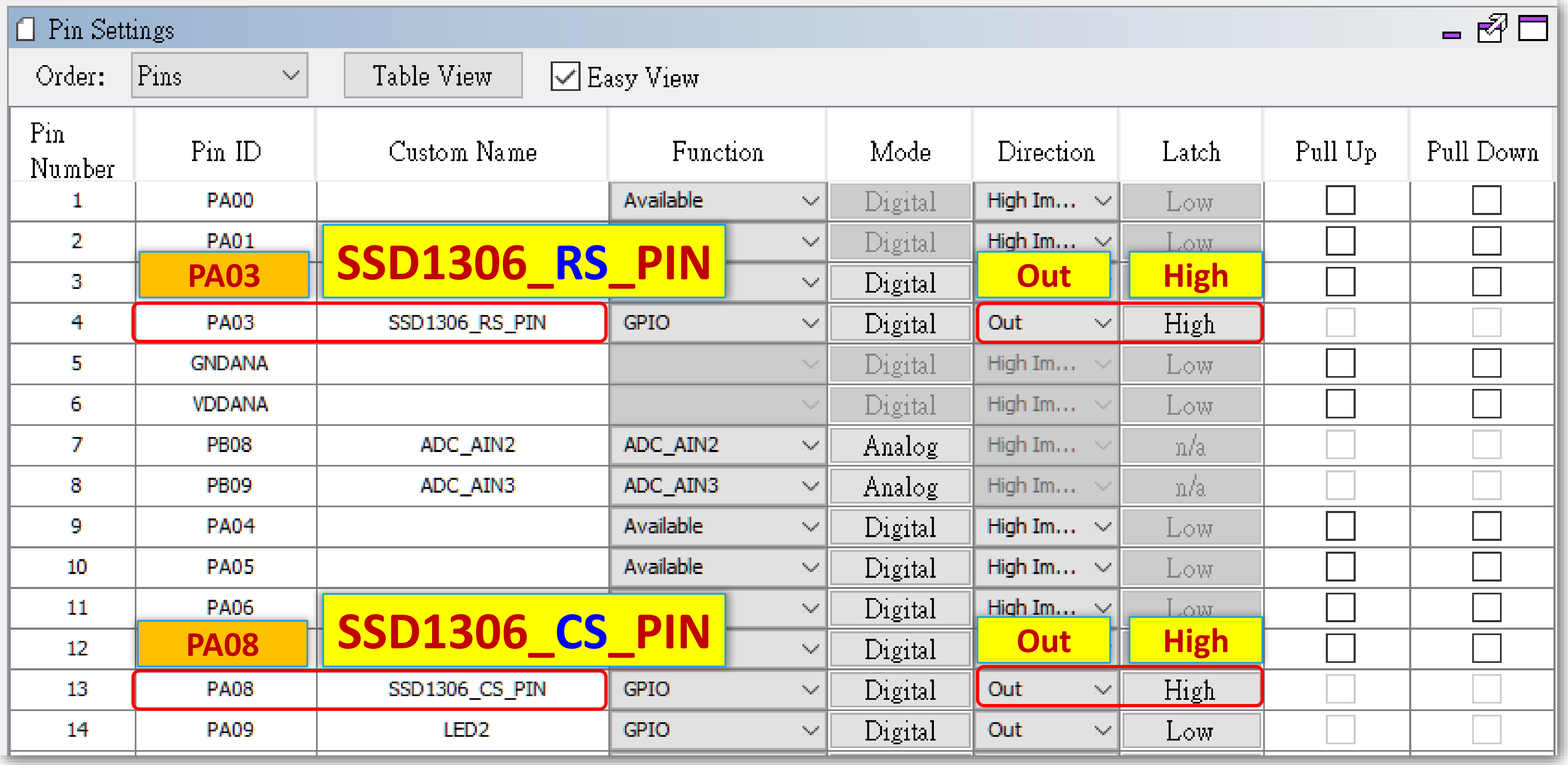
- **Pin Table Configuration of GPIO**



Lab17 SERCOM - SPI SSD1306 OLED

Step 5

- **Pin Setting of GPIO for direction and custom name.**

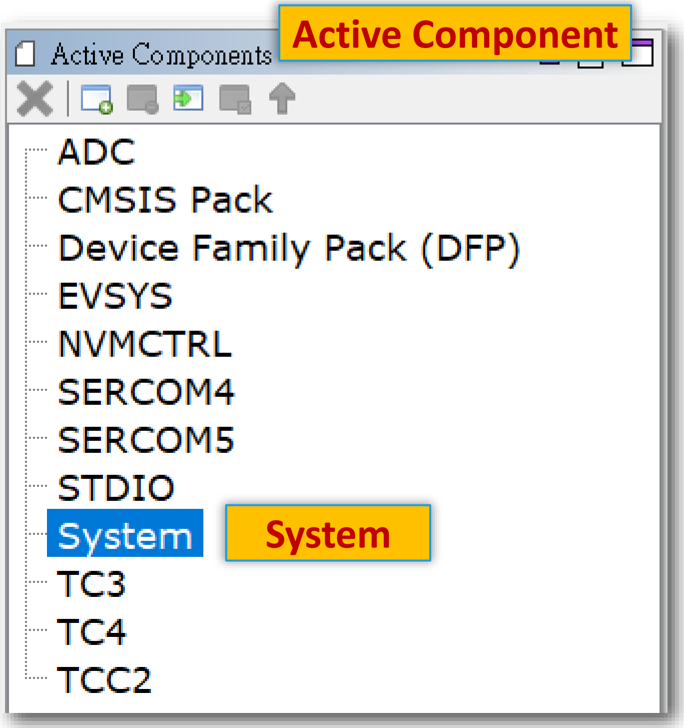


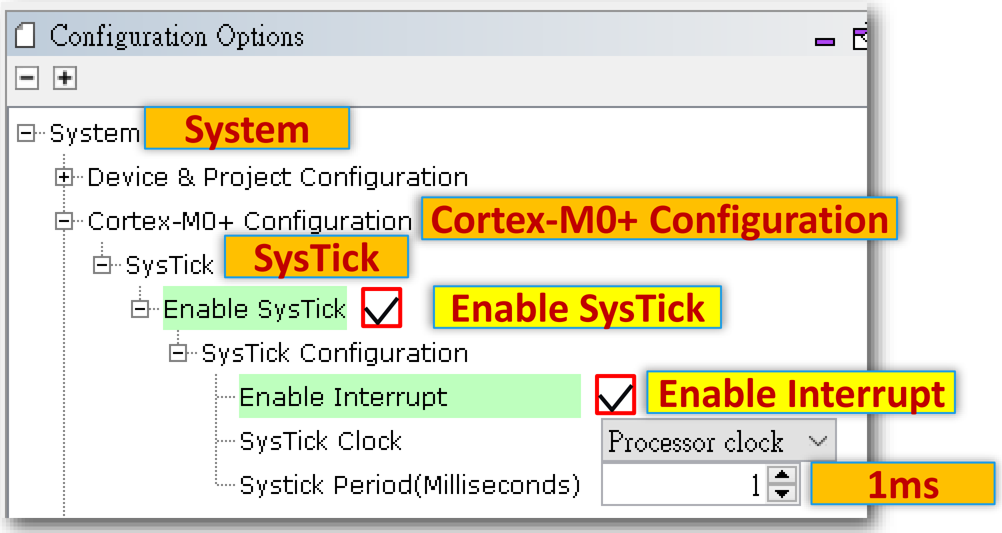
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down
1	PA00		Available	Digital	High Im...	Low		
2	PA01			Digital	High Im...	Low		
3				Digital				
4	PA03	SSD1306_RS_PIN	GPIO	Digital	Out	High		
5	GNDANA			Digital	High Im...	Low		
6	VDDANA			Digital	High Im...	Low		
7	PB08	ADC_AIN2	ADC_AIN2	Analog	High Im...	n/a		
8	PB09	ADC_AIN3	ADC_AIN3	Analog	High Im...	n/a		
9	PA04		Available	Digital	High Im...	Low		
10	PA05		Available	Digital	High Im...	Low		
11	PA06			Digital	High Im...	Low		
12				Digital				
13	PA08	SSD1306_CS_PIN	GPIO	Digital	Out	High		
14	PA09	LED2	GPIO	Digital	Out	Low		

Lab17 SERCOM - SPI SSD1306 OLED

Step 6

- Select exist module **System** in **Active Component**.
- Enable **SysTick** for **Delay Function**.

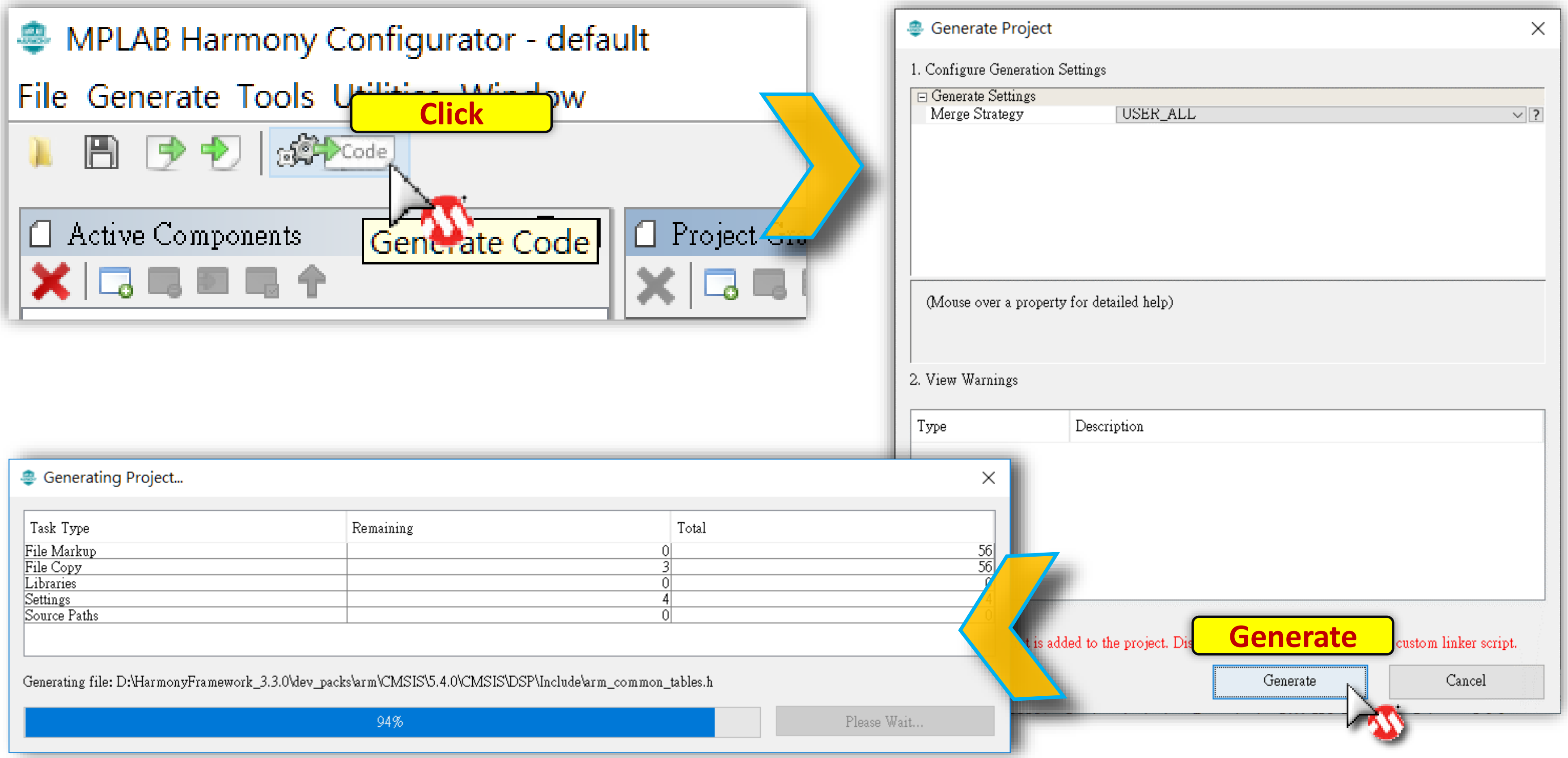




Lab17 SERCOM - SPI SSD1306 OLED

Step 7

- **Click  to Generate Code and save changes to MHC configure file.**



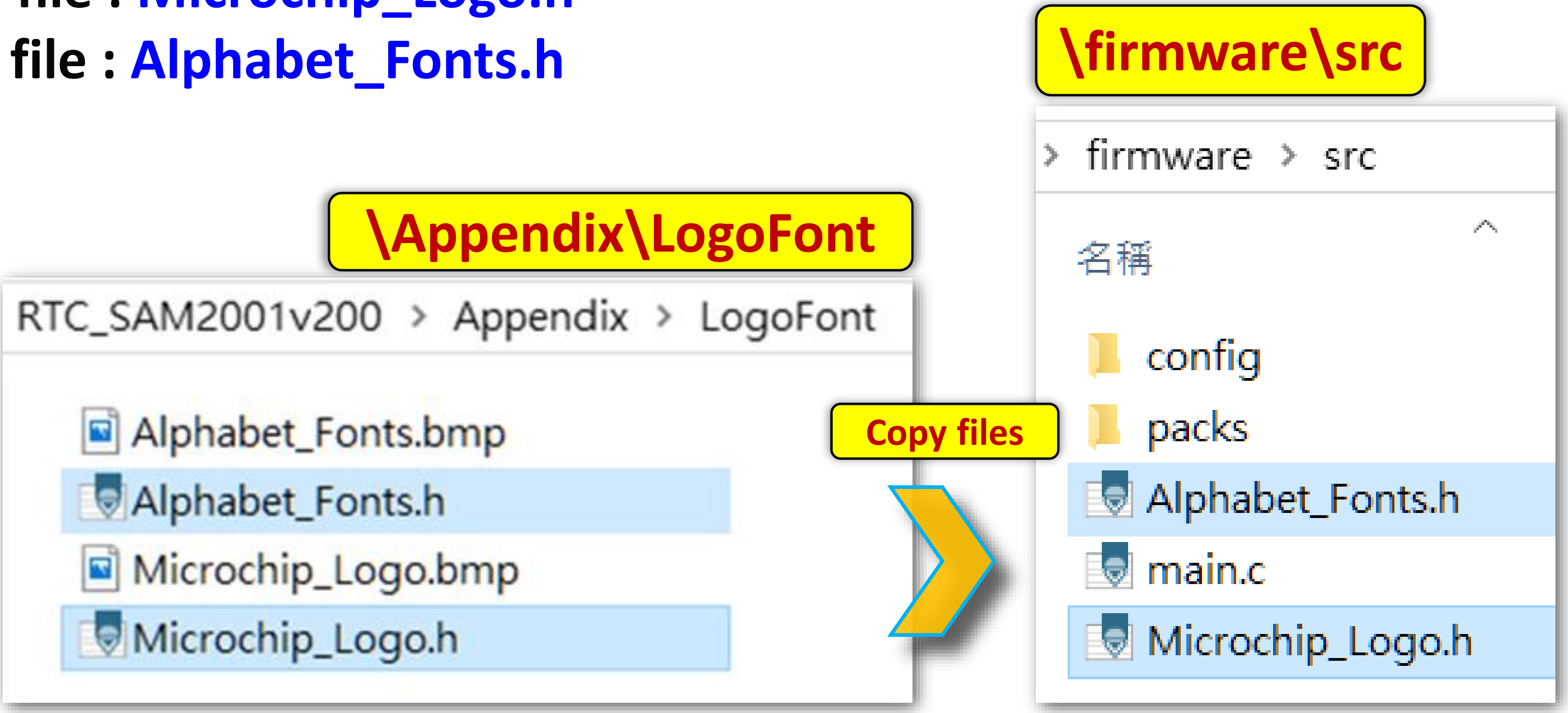
Lab17 SERCOM - SPI SSD1306 OLED

Step 8

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

  Logo image file : Microchip_Logo.h

  Font image file : Alphabet_Fonts.h



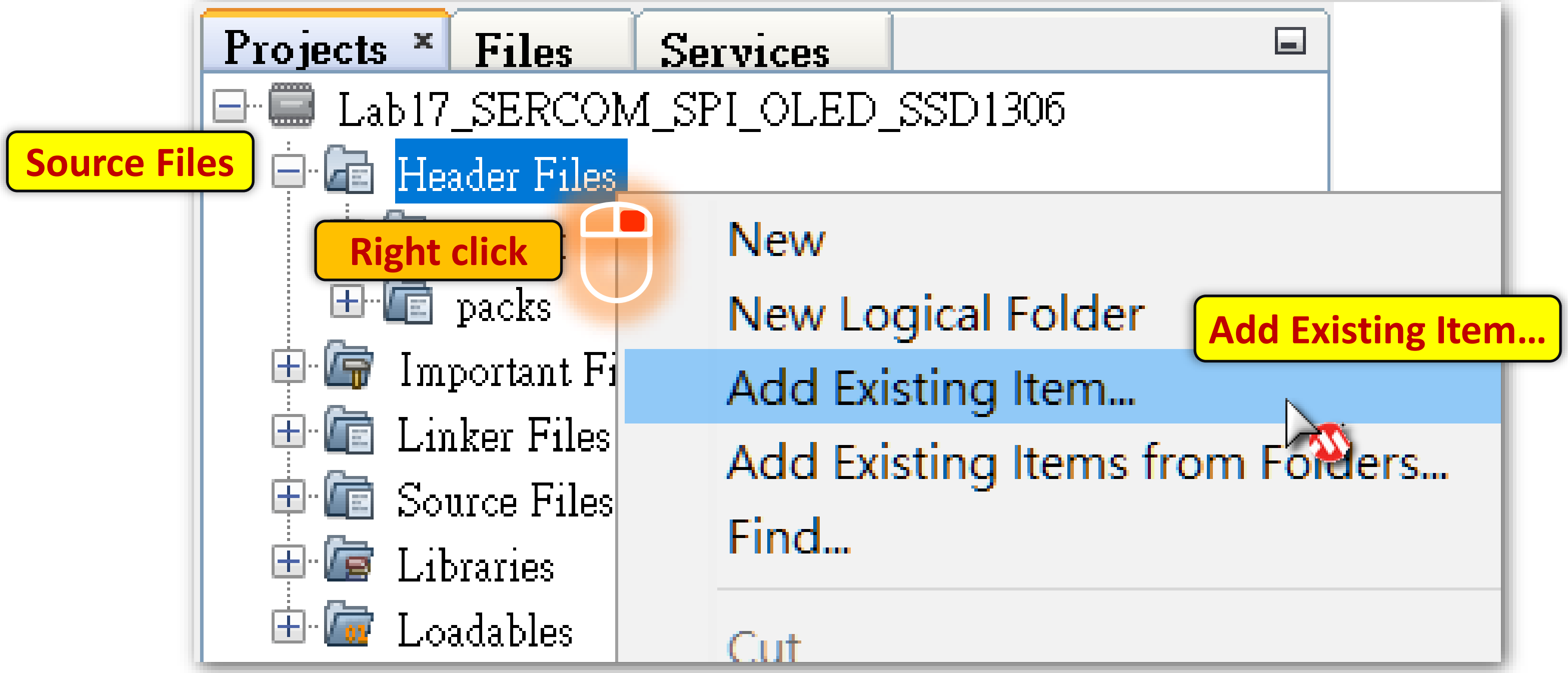
MICROCHIP

Lab17 SERCOM - SPI SSD1306 OLED

Step 9

- **Add pre-build logo image header files into project**

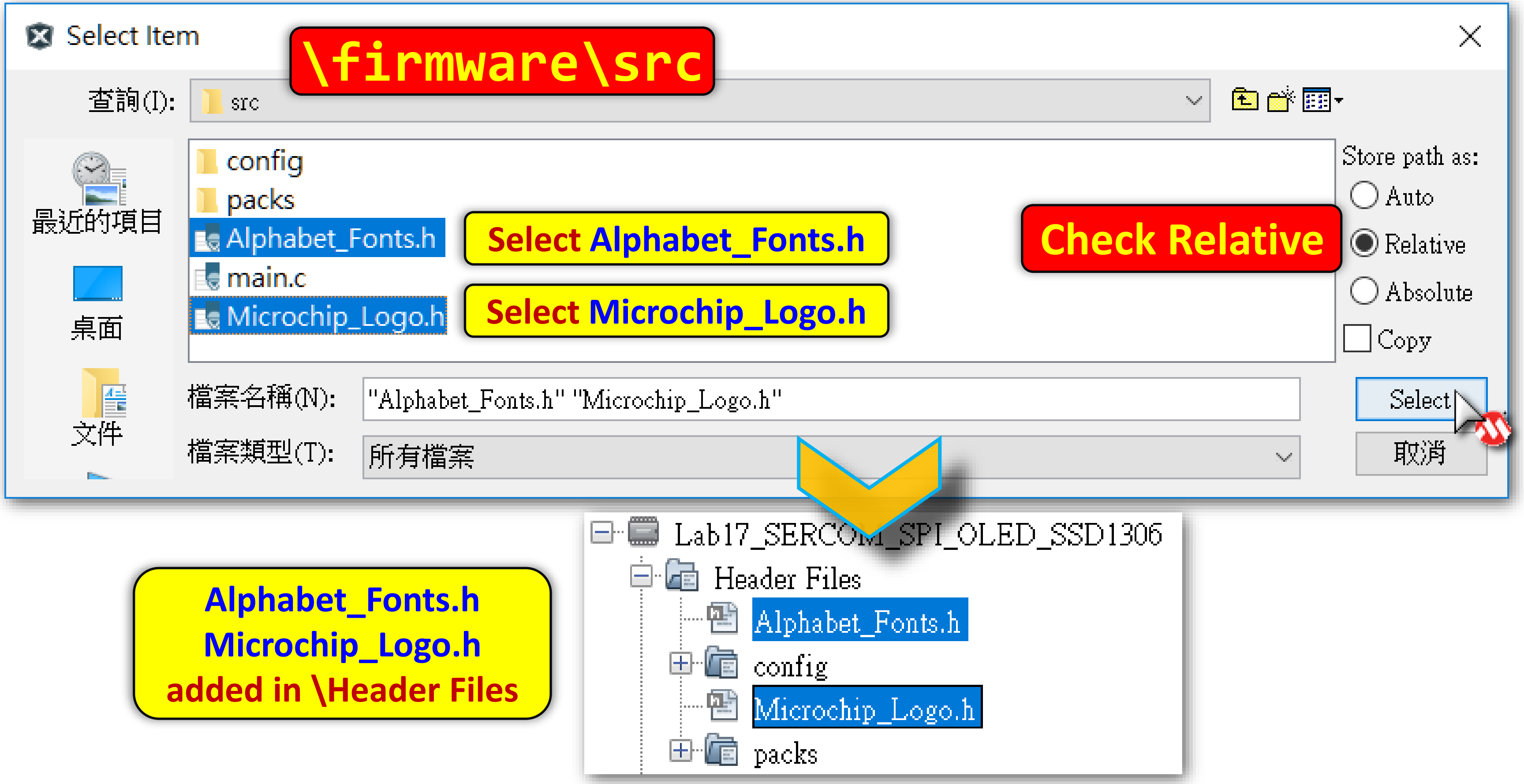
  Right click Header Files ▸ Add Existing item...



Lab17 SERCOM - SPI SSD1306 OLED

Step 10

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



MICROCHIP

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] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijk\
                     lmnopqrstuvwxyz~!@#$%^&()[]{}_\\|;:,.`'\"<>+-*/=? ";

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_Fonts+(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)
{ …
  // 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_Compare16bitPeriodGet()*1000/TC3_CompareFrequencyGet() );
  LCM_DrawString( 0, 1, (char *)USARTRTxBuffer );
  sprintf( (char *)USARTRTxBuffer, "TC4 period %ldms",
                TC4_Compare16bitPeriodGet()*1000/TC4_CompareFrequencyGet() );
  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

```
int main (void)
{  …
   while ( true )
   { …
      if (TC3_IsOverflow)
      { …
         // TODO 17.04
         sprintf( (char *)USARTRTxBuffer, "Hello World!!");
         LCM_DrawString( 0, 3, (char *)USARTRTxBuffer );
      }
      …
       if (TC4_IsOverflow)
      { …
         // TODO 17.05
         sprintf( (char *)USARTRTxBuffer, "PWM Duty : %3d%%", Duty);
         LCM_DrawString( 0, 4, (char *)USARTRTxBuffer );
      }
      …
   }
```

Lab17 SERCOM - SPI SSD1306 OLED

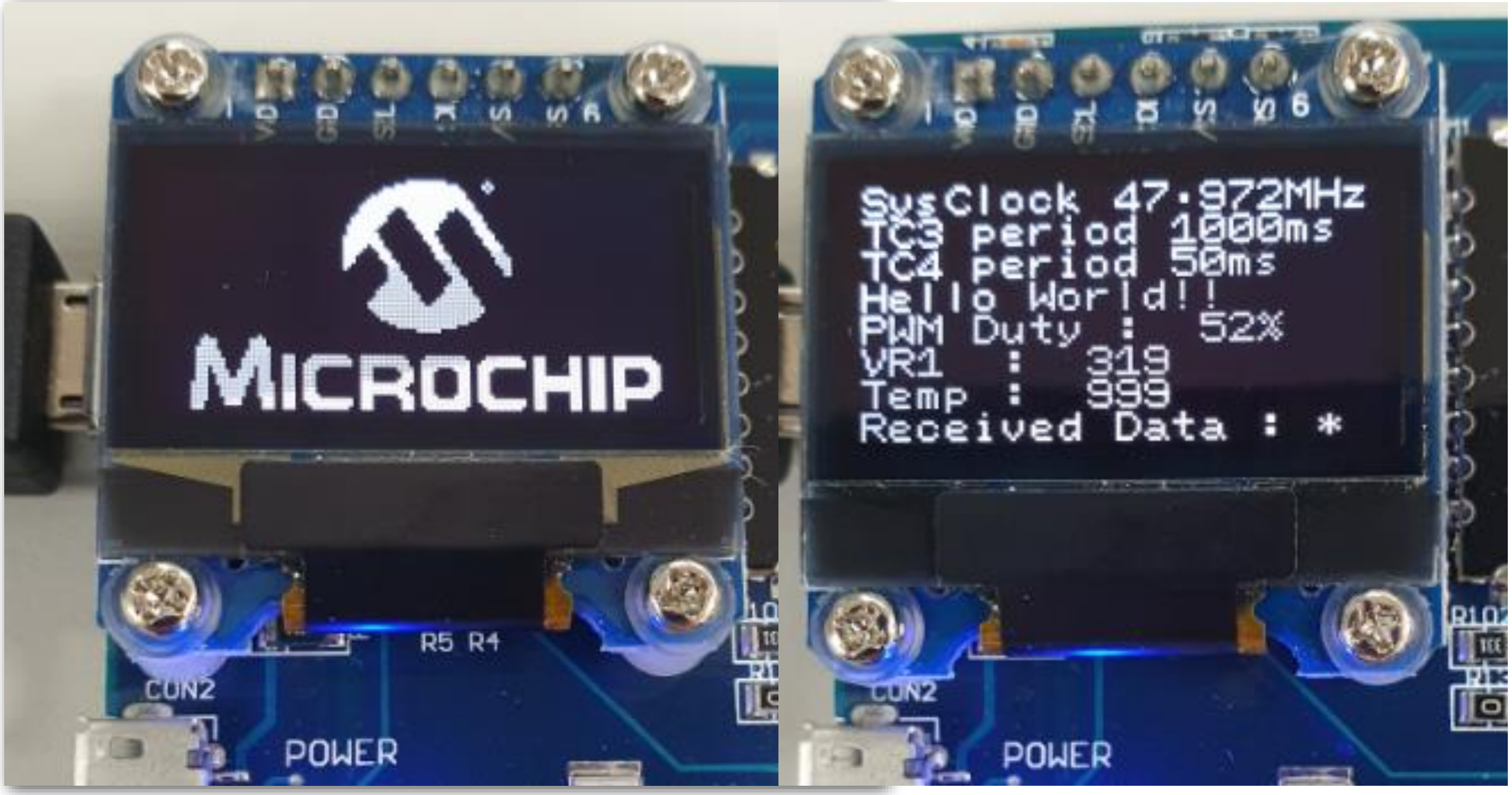
Step 17

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



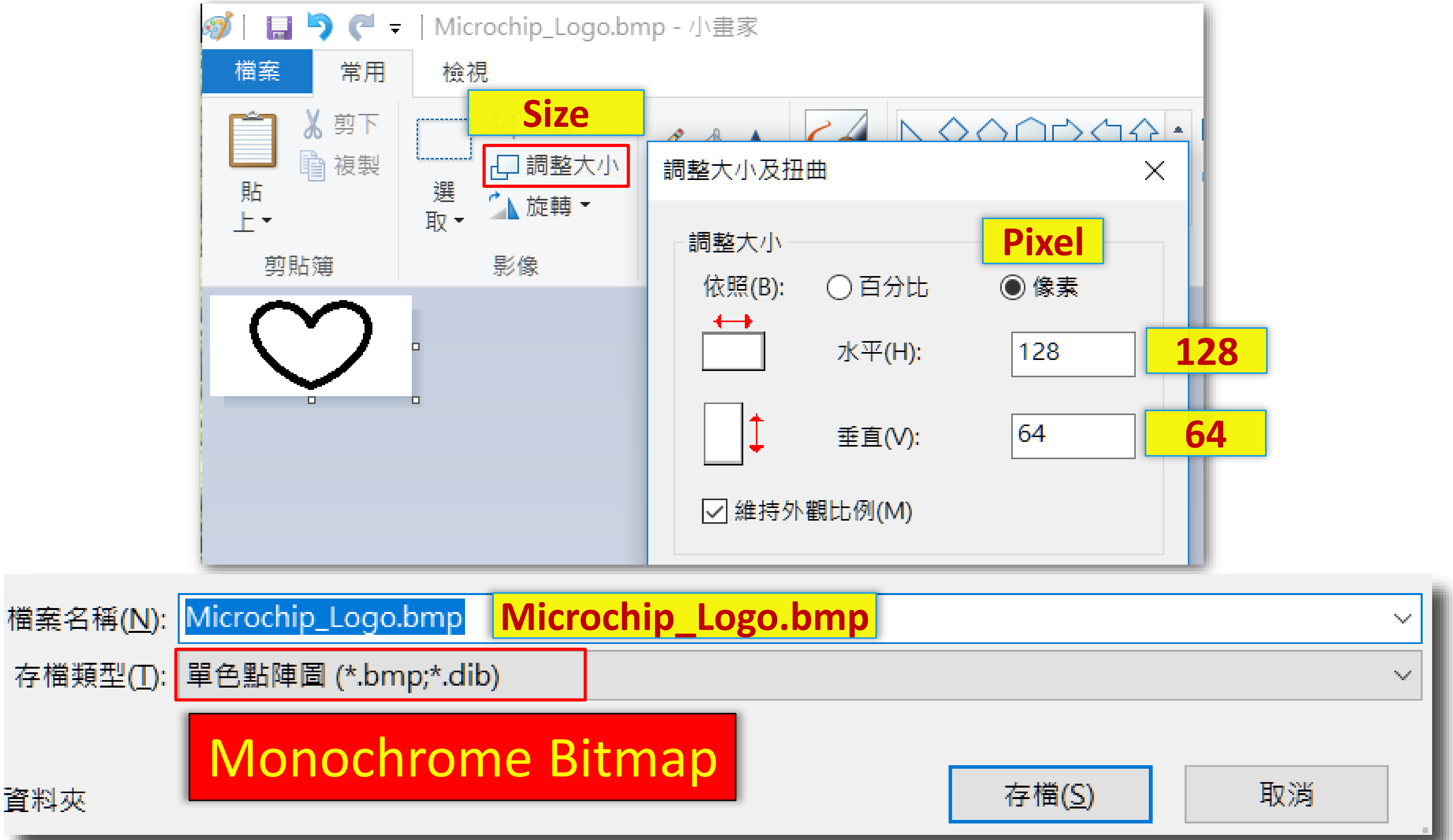
Lab17 SERCOM - SPI SSD1306 OLED

- To create your personally welcome page (logo) !

- **Let’s go!**

Lab17 SERCOM - SPI SSD1306 OLED

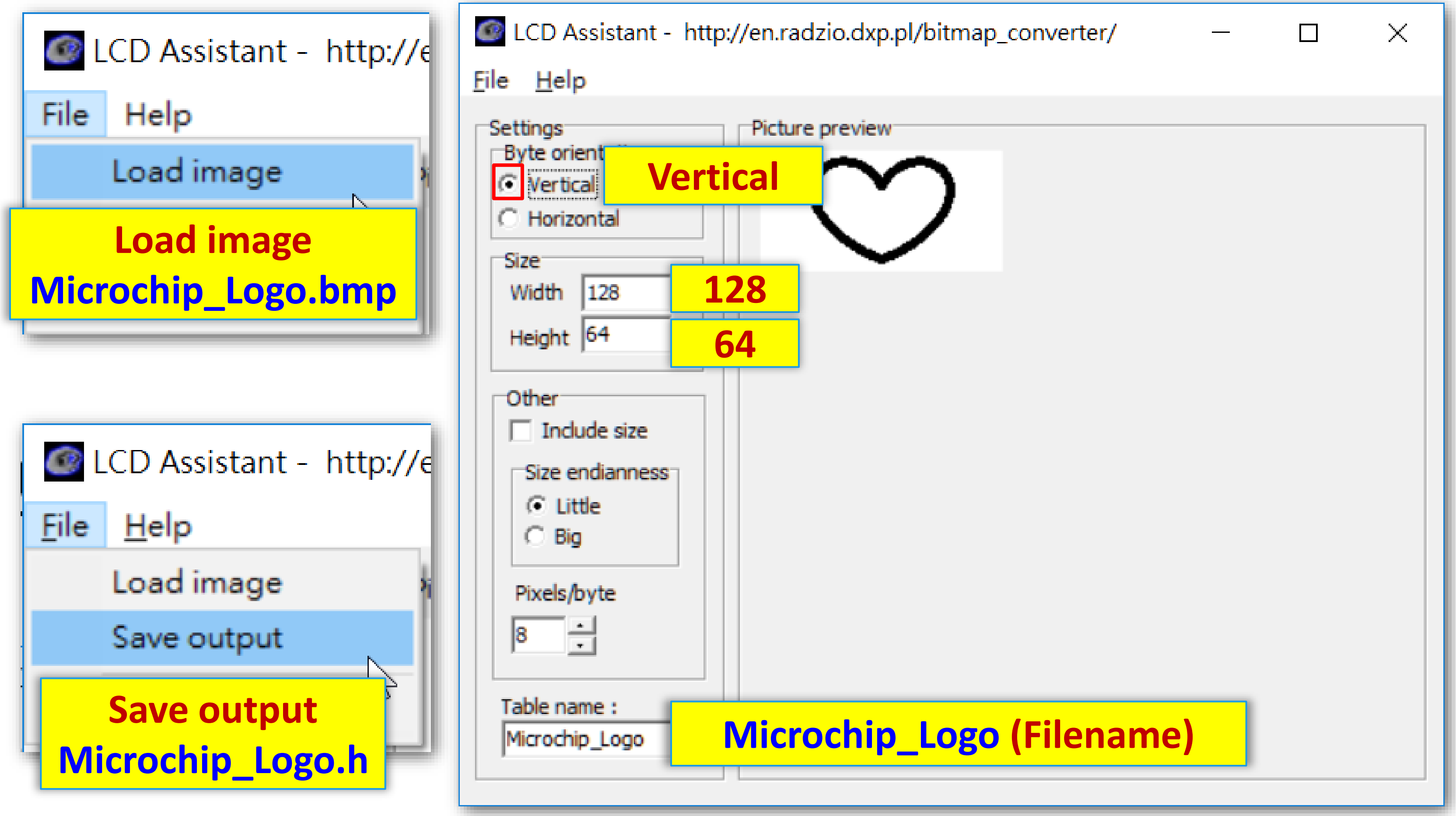
Create your own logo



Lab17 SERCOM - SPI SSD1306 OLED

Create your own logo

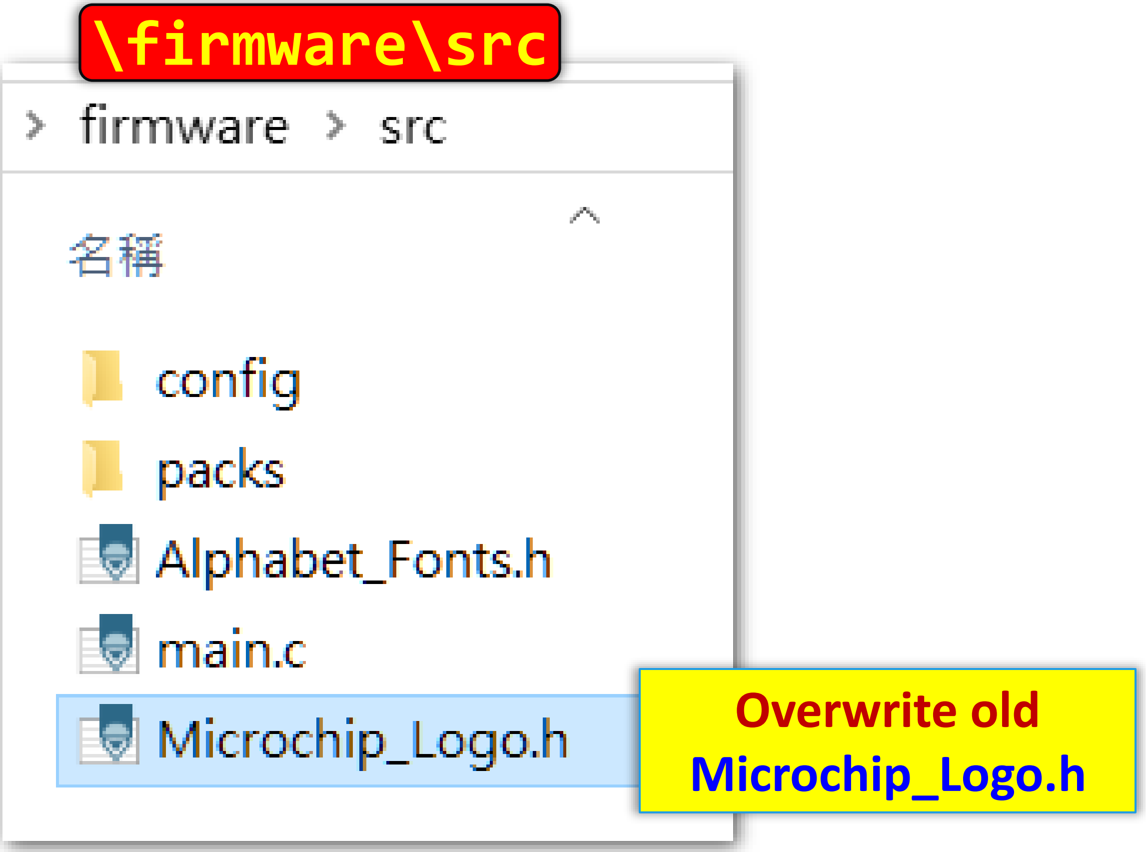
- Execute \Tools\LCDAssistant.exe
  Load Microchip_Logo.bmp and Save output to Microchip_Logo.h



Lab17 SERCOM - SPI SSD1306 OLED

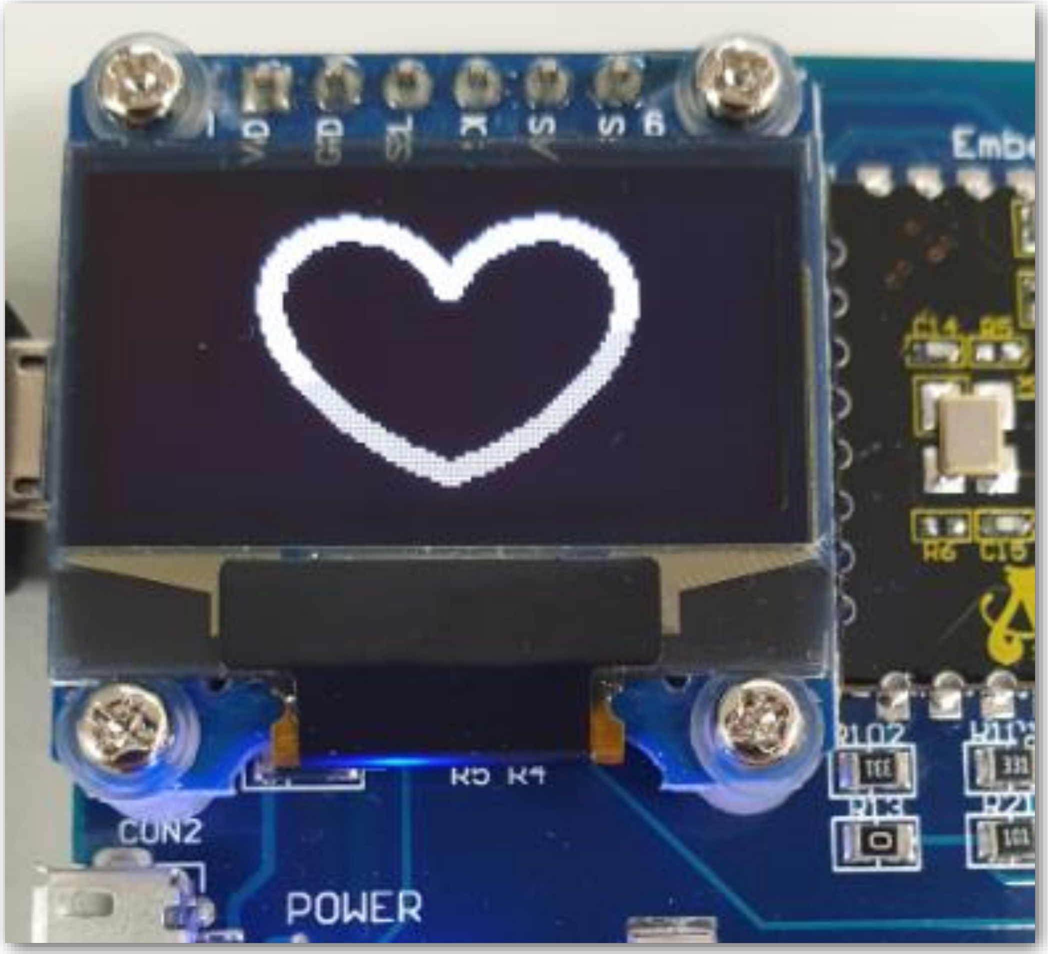
Create your own logo

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



Lab17 SERCOM - SPI SSD1306 OLED

Create your own logo result



Trademarks

MICROCHIP

Thank You !!