

CAE專家教室 PIC32CX1001系列課程
PIC32CXSG ARM Cortex-M4F
Microcontroller & Harmony v3
PIC32CX1001 v1.00



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



SMART | CONNECTED | SECURE

CAE Taiwan
Microchip Technology Inc.

August 2025

SAM series course history and brief

- **SAM1001 – MCU32 Peripheral Library (PLIB)**

- (2017~2018)
 - Peripheral Library (PLIB) on APP045v2.0 (SAMD21 Arm Cortex-M0+)
 - Advance Software Framework (ASF) v3.x + GNU C compiler + Microchip Studio 7
- (2019~2022)
 - Peripheral Library (PLIB) on APP045v2.0/v4.x (SAMD21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
- (2023~2024)
 - Peripheral Library (PLIB) on APP045v4.x/v5.x (SAMD21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE

- **SAM2001ADV – MCU32 Advanced Peripheral Library (PLIB)**

- (2021~2022)
 - Peripheral Library (PLIB) Advanced modules on APP045v4.x (SAMD21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
- (2023~2024)
 - Peripheral Library (PLIB) Advanced modules on APP045v4.x/v5.x (SAMD21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE

SAM series course history and brief

- **SAM2002 – MCU32 System Service and Driver**
 - (2019~2022)
 - System Service and Driver on APP045v4.x (SAMD21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
 - (2023~2024)
 - System Service and Driver on APP045v4.x/v5.x (SAMD21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE
- **SAM2002ADV – MCU32 Touch and USB Device**
 - (2021~2022)
 - QTouch Library and USB Device Driver on APP045v4.x (SAMD21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
 - (2023~2024)
 - QTouch Library and USB Device Driver on APP045v4.x (SAMD21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE
- **PIC32CX1001 – Cortex M4F with Advanced Peripheral Library (PLIB)**
 - (2025)
 - Peripheral Library (PLIB) Advanced modules on APP055v1.01 (PIC32CX-SG41 Arm Cortex-M4F)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE

Class Objectives

- **When you finish this course, you will be able to:**
 - Be familiar with MPLAB X IDE and MCC Harmony
 - Understand how to configure and use the PIC32CX-SG families' peripherals.
- **There have two level course included.**
 - Major course : General study of basic features.
 - Appendix : Advance features and appendix.

Index (1/2)

- [PIC32CX-SG Arm Cortex-M4F Microcontroller Introduction](#)
- [MPLAB X IDE and MCC Harmony Development Tools Introduction](#)
 - § [Soft Request In this Course](#)
- [PIC32CX-SG EVB APP055 v1.10 Introduction](#)
- [Course Materials](#)
- [Getting Started with First Project](#)
 - ✂ [Lab0 First Project](#)
- [MPLAB® Code Configurator \(MCC\) Harmony Interface Introduction](#)
 - § [Appendix : Open MPLAB X IDE Projects](#)
- [I/O – PORT](#)
 - ✂ [Lab1 PORT Input and Output](#)
 - § [Appendix : Code Trace and PORT Register](#)
- [Timer by Polling – TC](#)
 - ✂ [Lab2 TC Timer Method Polling](#)
- [Interrupt – NVIC](#)
- [Timer by Interrupt Callback – TC](#)
 - ✂ [Lab3 Timer Interrupt Callback TC](#)
- [Oscillator and Clock Generator](#)
 - ✂ [Lab4 System Clock](#)
 - § [Appendix : New Clock Configurator GUI](#)
- [SERCOM Introduction](#)
- [SERCOM-USART Architecture](#)
- [UART Transmit by Polling](#)
 - ✂ [Lab5 COM Port TX by polling - SERCOM-USART](#)
- [UART Receive by Interrupt](#)
 - ✂ [Lab6 COM Port RX by interrupt - SERCOM-USART](#)

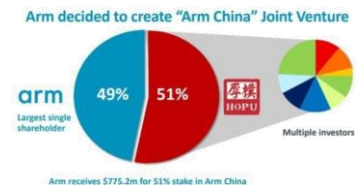
Index (2/2)

- [myprintf](#)
 - ✂ [Lab7 myprintf - SERCOM-USART](#)
 - § [Appendix : Custom VT100 command](#)
- [High Speed ADC Architecture](#)
 - ✂ [Lab8 ADC Polling Software Trigger](#)
- [Dual ADC Instances](#)
 - ✂ [Lab9 Dual ADC Core Polling Software Trigger](#)
- [ADC Interrupt Method](#)
 - ✂ [Lab10 Dual ADC Core Interrupt Software Trigger](#)
 - § [Appendix : ADC DMA Sequencing](#)
 - § [Appendix : ADC Host-Client Operation](#)
 - § [Appendix : ADC Throughput Rate](#)
- [Pulse-Width Modulation - TCC_PWM](#)
 - ✂ [Lab11 LED Dimming TCC PWM](#)
 - ✂ [Lab12 LED Dimming Adjustable TCC PWM](#)
 - ✂ [Lab13 LED Breath TCC PWM](#)
 - ✂ [Appendix : TCC PWM Waveform Output Matrix](#)
- [SERCOM-SPI Architecture](#)
 - ✂ [Lab14 OLED Graphics](#)
 - ✂ [Lab15 Design Your Own Logo](#)
 - § [Appendix : OLED Graphic Library](#)
 - § [Appendix : OLED Graphic Application](#)
- [6-Axis Motion Sensor Gyroscope and Accelerometer](#)
- [SERCOM-I2C Architecture](#)
 - ✂ [Lab16 Motion Sensor Application](#)
 - § [Appendix : Motion Application](#)

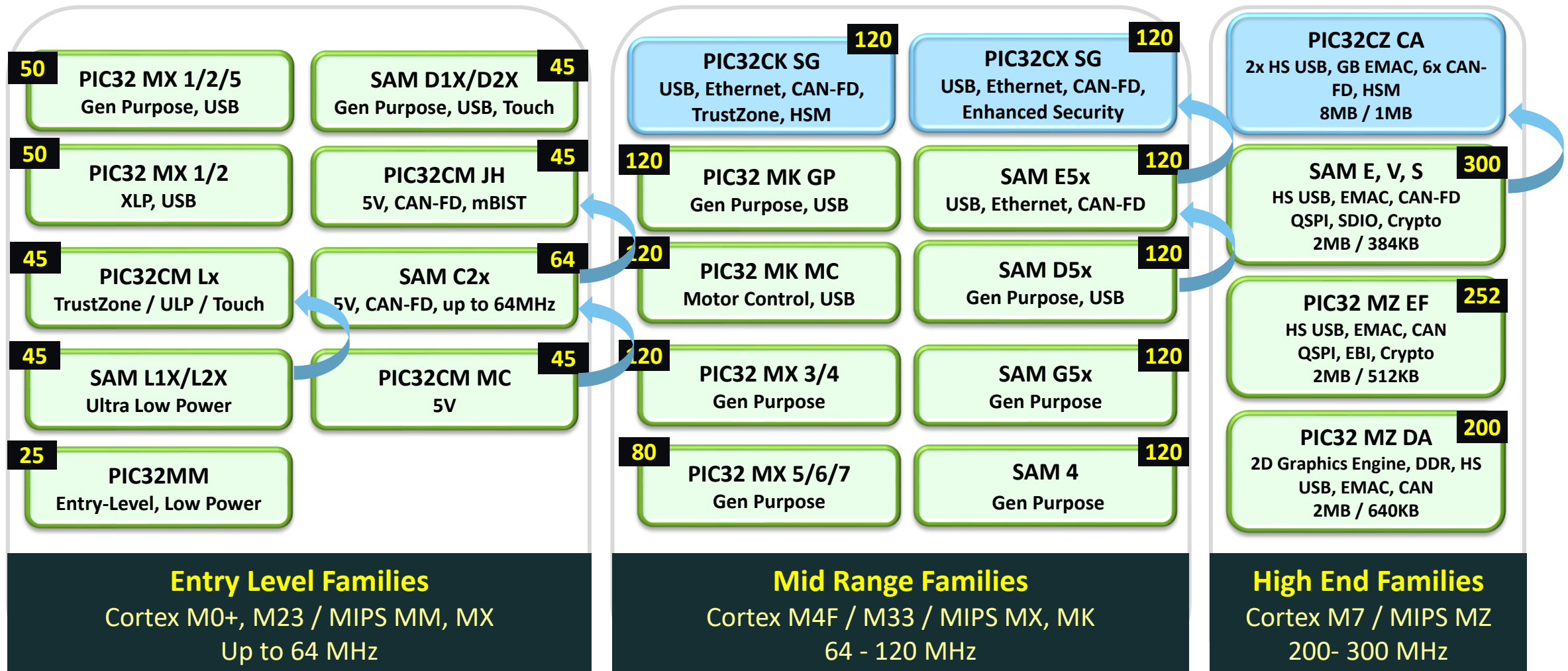
PIC32CX-SG Arm Cortex-M4F Microcontroller Introduction

Arm's history

- Arm, originally known as Advanced RISC Machines, is a leading processor architecture based on RISC (Reduced Instruction Set Computing) principles. Developed by Arm Ltd., the architecture is licensed globally and powers a wide range of devices from smartphones and IoT sensors to servers and AI accelerators. Its success stems from its power efficiency and scalability.
- **SoftBank acquired Arm in 2016 for \$31 billion. In 2018, it sold 51% of Arm China to local investors**, but the venture became mired in a prolonged governance dispute, raising international concern. **NVIDIA's 2020 attempt to acquire Arm for \$40 billion failed in 2022 due to global regulatory pushback.**
- To regain value, SoftBank launched an IPO for Arm on September 14, **2023. Listed on NASDAQ under the symbol Arm**, it debuted with a \$54 billion valuation, making it the biggest tech IPO of the year. As of 2025, Arm continues to lead in energy-efficient computing and is expanding its footprint in AI, automotive, and data center markets, while strengthening collaborations across the global semiconductor ecosystem.



A Powerhouse 32-bit MCU Portfolio



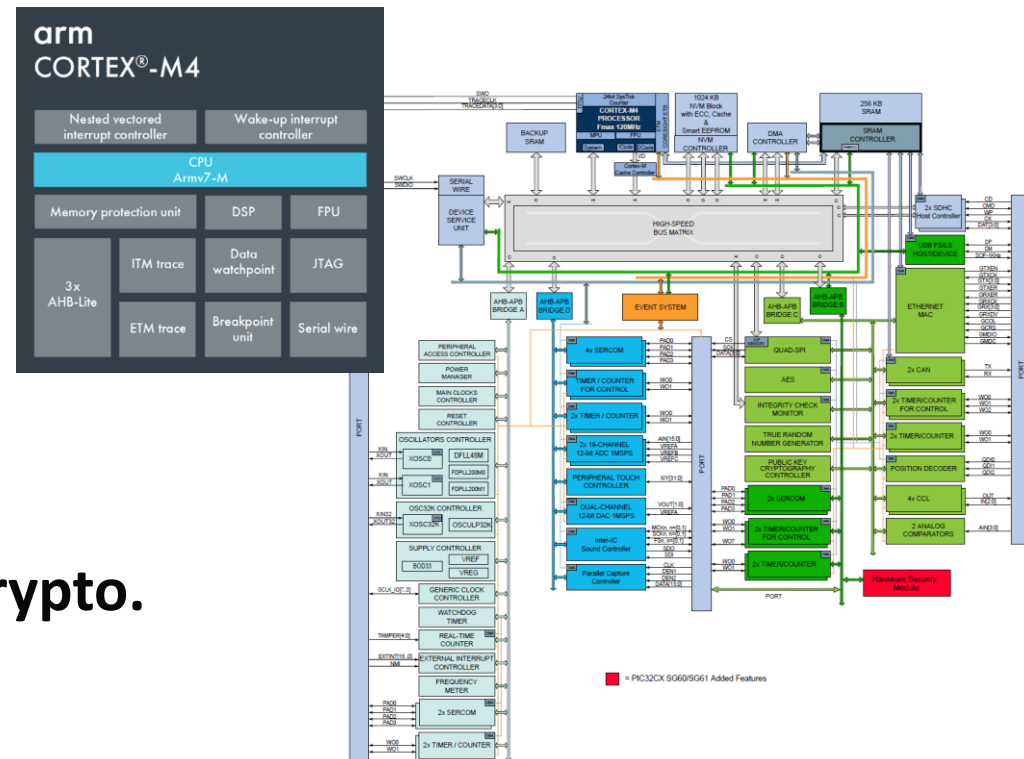
In Production

New Products

Early Access

PIC32CX-SG families Introduction

- **PIC32CX-SG families are mid-performance, that combine connectivity, performance, and security microcontrollers using the Arm® Cortex® M4F processor with Arm v7E-M Harvard architecture.**
- **Maximum frequency of 120 MHz and reach 403.04 CoreMark®.**
https://www.eembc.org/viewer/?benchmark_seq=2560
- **All devices include intelligent and flexible peripherals, Event System for inter-peripheral signaling, and support USB, Ethernet and PTC for capacitive touch button, slider and wheel user interfaces.**
- **Optional Hardware Security Module and Crypto.**
- **Single-precision Floating Point Unit (FPU).**



Architecture

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

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

Instruction Set Architecture, ISA

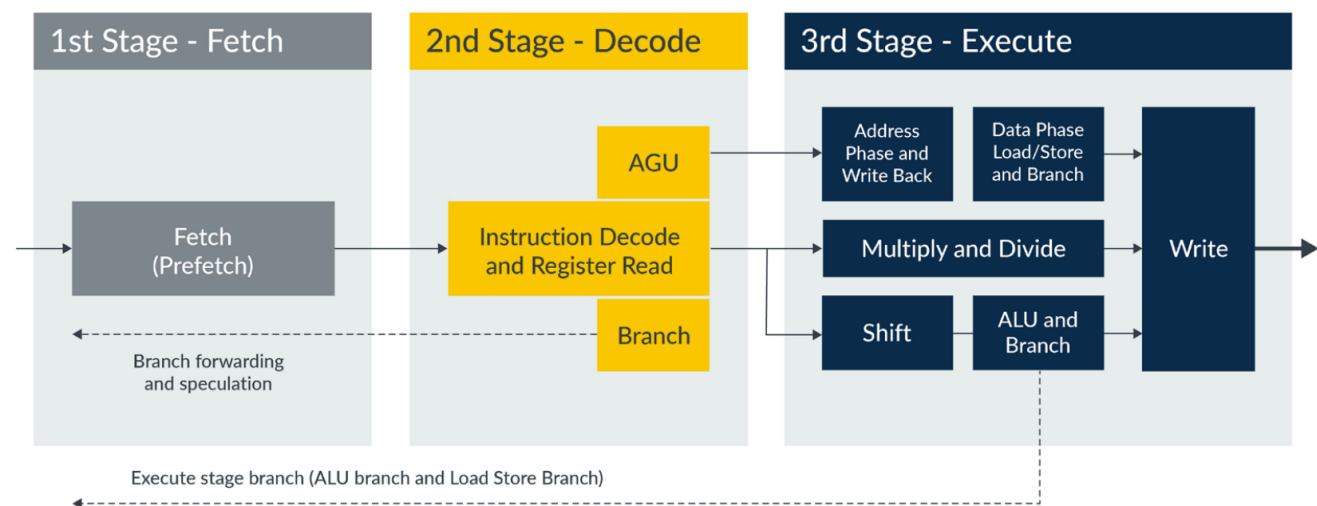
- **Arm v7E-M**

- **98+ instructions**
- **All Thumb1 and Thumb 2 instructions**
- **Multiply adds 32x32 into 64-bits result**
- **Divide instructions**
- **DSP and Saturation instructions**
- **Single precision floating point**



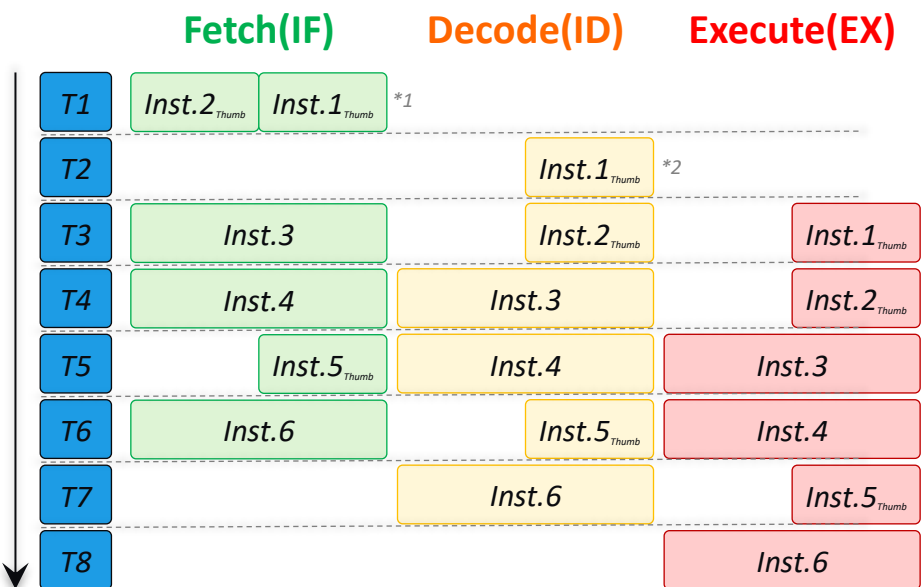
3-Stage Pipeline

- The Cortex-M4F processor with a 3-Stage Pipeline.
 - Fetch, Decode and Execute.



e.g.

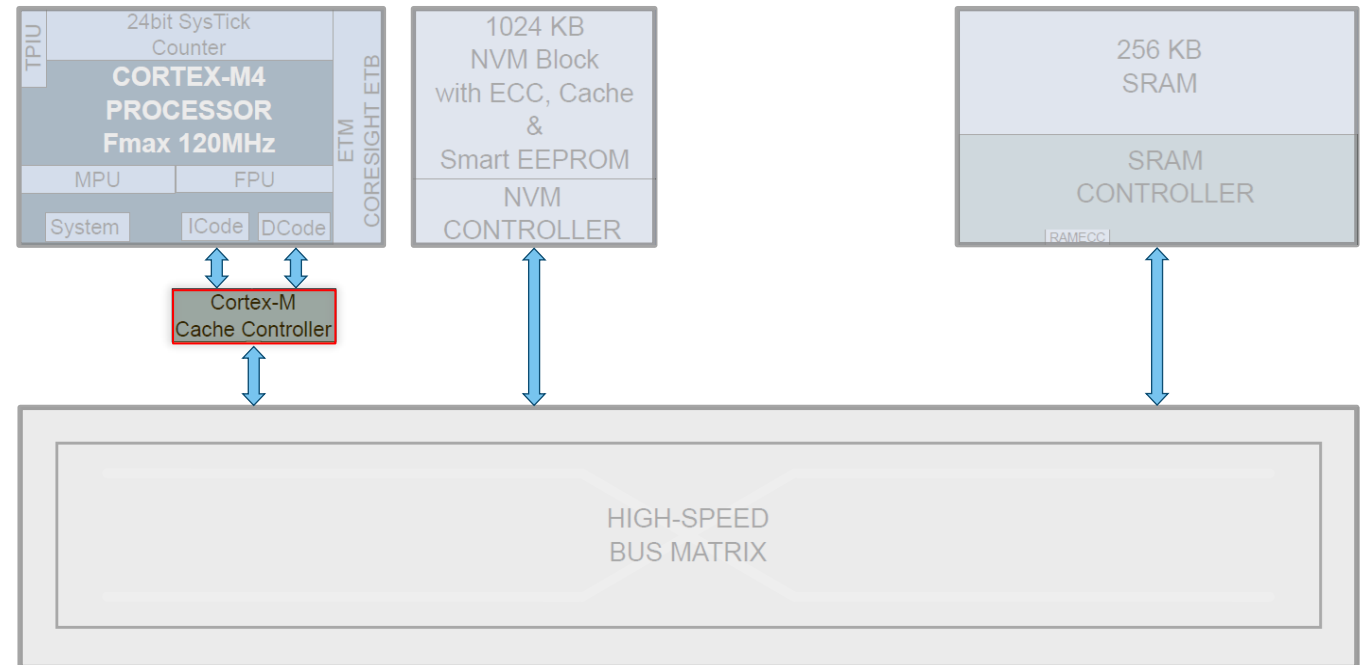
- 1.ADD R0, R0, #1(Thumb)
- 2.SUB R1, R1, #2(Thumb)
- 3.MOVW R2, #0x1234(32-bits)
- 4.MOVT R2, #0x5678(32-bits)
- 5.LDR R3, [R4](Thumb)
- 6.MOVT R3, #0xABCD(32-bits)



*1 Two thumb Instruction be fetched in on transfer.
*2 processing one instruction at a time.

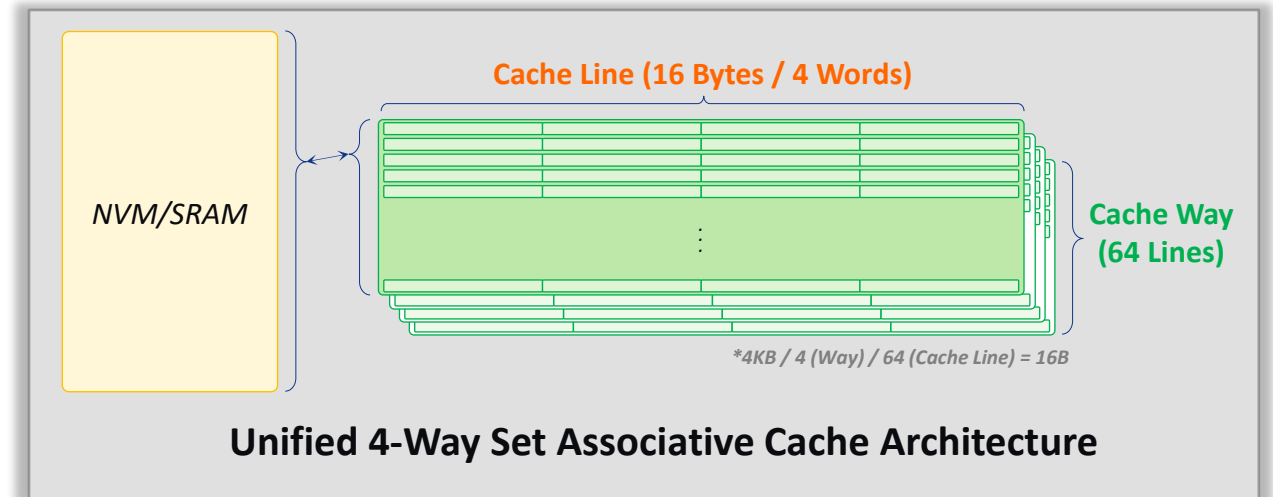
Cortex-M Cache Controller

- The Cortex M4 architectural does not include a cache, a cache was added with a Microchip peripheral called the Cortex-M cache controller, CMCC.
- The key features of the CMCC,
 - 4KB instruction and Data Cache.
 - * Von Neumann Cache Architecture
 - Unified 4-Way Set Associative.
 - 64 Lines totally, 16 Bytes each Line.
 - Round Robin victim selection policy
 - Configurable Tightly Coupled Memory, TCM



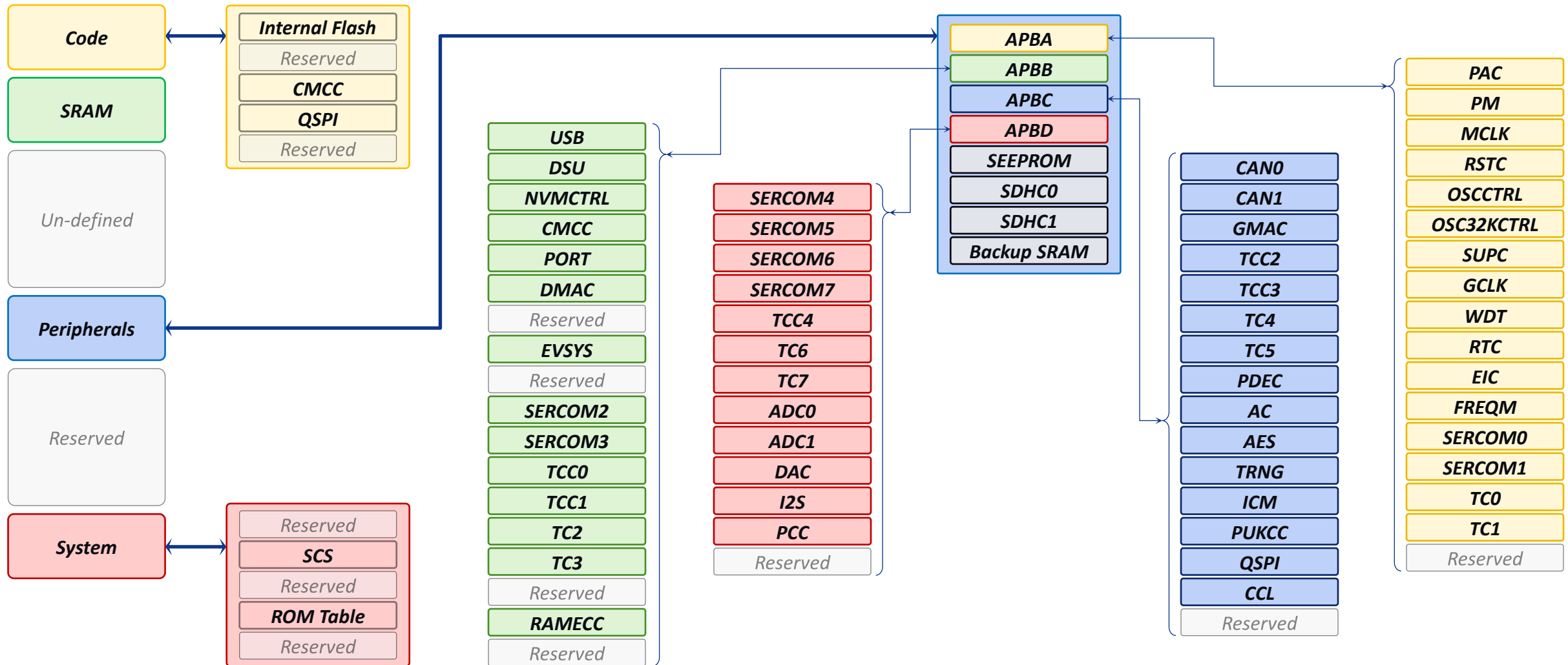
Unified 4-Way Set Associative Cache

- **Unified** means that Instructions and data share the cache. It's more flexible because store instructions or data as needed. But instructions and data may compete for cache space.
- **Unified 4-Way Set Associative Cache,**
 - 16 Bytes each Cache Line. 64 Lines each way, 4 ways totally.
 - The main memory is mapped to 64 Cache Lines and stored in one of the Cache Line within 4-Ways.
 - Multiple main memory addresses may be mapped to the same Cache Line,
 - but they are distinguished by Tags, and the replacement strategy (such as LRU) determines which Way to store them.
 - e.g.,
A, B, C, D, and E all mapped to same Cache Line and is 4-Way Set Associative.
The A, B, C, and D are stored in the Set in sequence. When E is also mapped to same Cache Line, the replacement strategy must be used to replace one of A/B/C/D

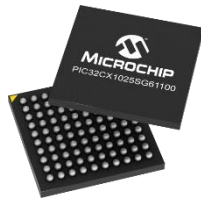
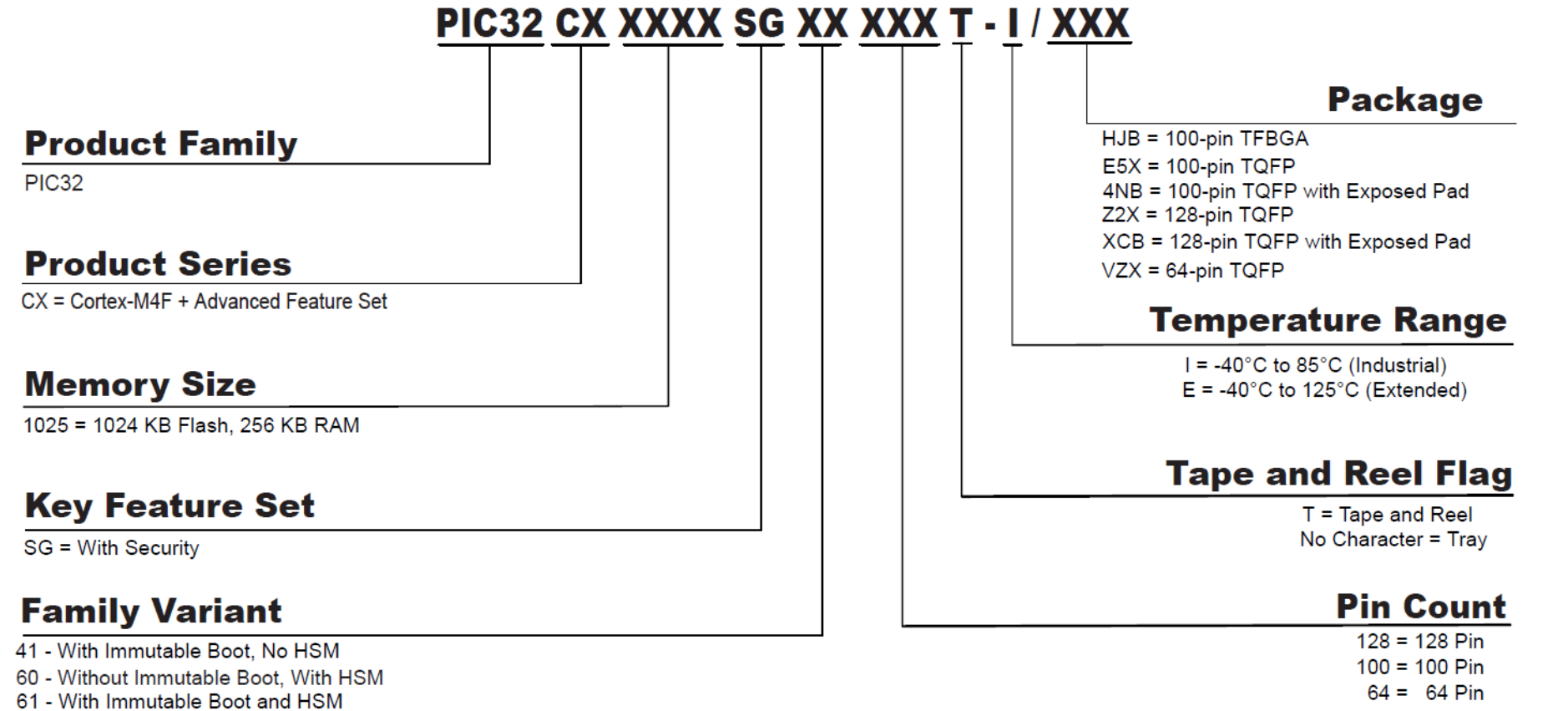


PIC32CX-SG Memory Map

- Up to 1MB NVM(Flash) and 256KB SRAM with 8KB Backup SRAM



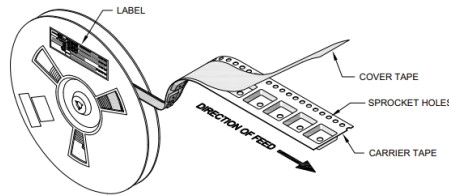
PIC32CX-SG Coding of Part Number



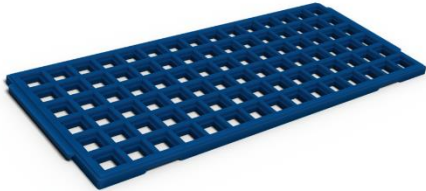
HJB, 100-Pins TFBGA



Z2X, 128 Pins TQFP



Tape and Reel



Tray

MPLAB X IDE and MCC Harmony Development Tools Introduction

Microchip Development Tool Ecosystem

Discover



Feature
Application
Software
Courses



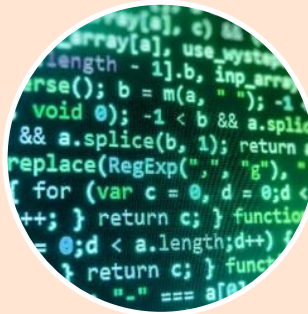
Configure



System init
Device init
Peripheral init



Develop



IDEs, Compilers
Example code
Software Stacks



Debug



Eval boards
Debuggers
Data Visualizer



Qualify



Code coverage
Code profiling
Functional safety



Production



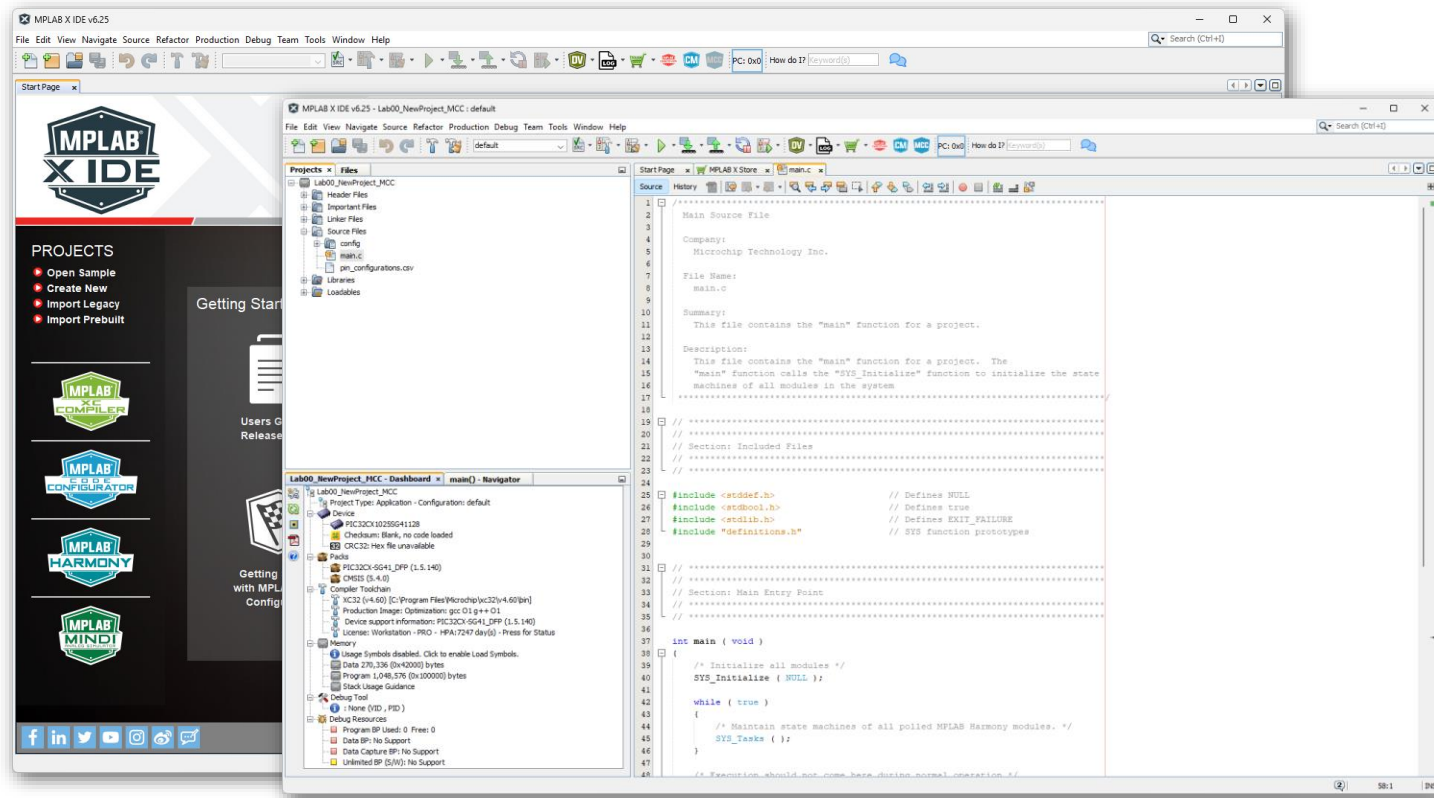
Programmers
Prog Center
3rd Party



All in One Development

MPLAB® X IDE

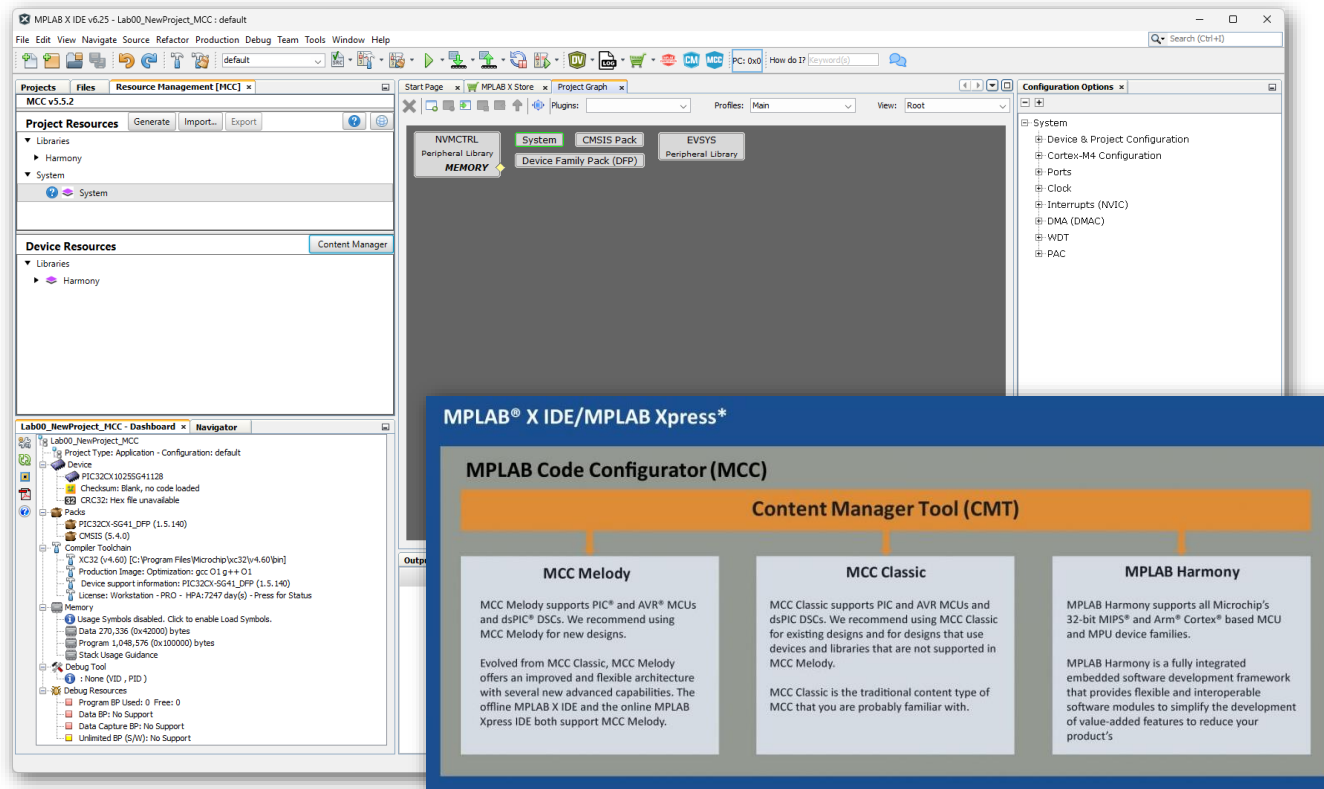
- New generation integrated Development Tools, Support PIC, dsPIC, AVR, CEC and SAM Series MCU/MPU, Provide Plug-in function to extend more advance function. Java Based, Cross platform, Current version is v6.25.



Graphical Code Generation Tool

MPLAB® Code Configurator

- MCC is a free, graphical programming environment that generates seamless,
- easy-to-understand C code to be inserted into your project.
- Supports 8-bits, 16-bits and 32-bits microcontrollers and microprocessors.



Embedded Software Development Framework

MPLAB® MCC Harmony

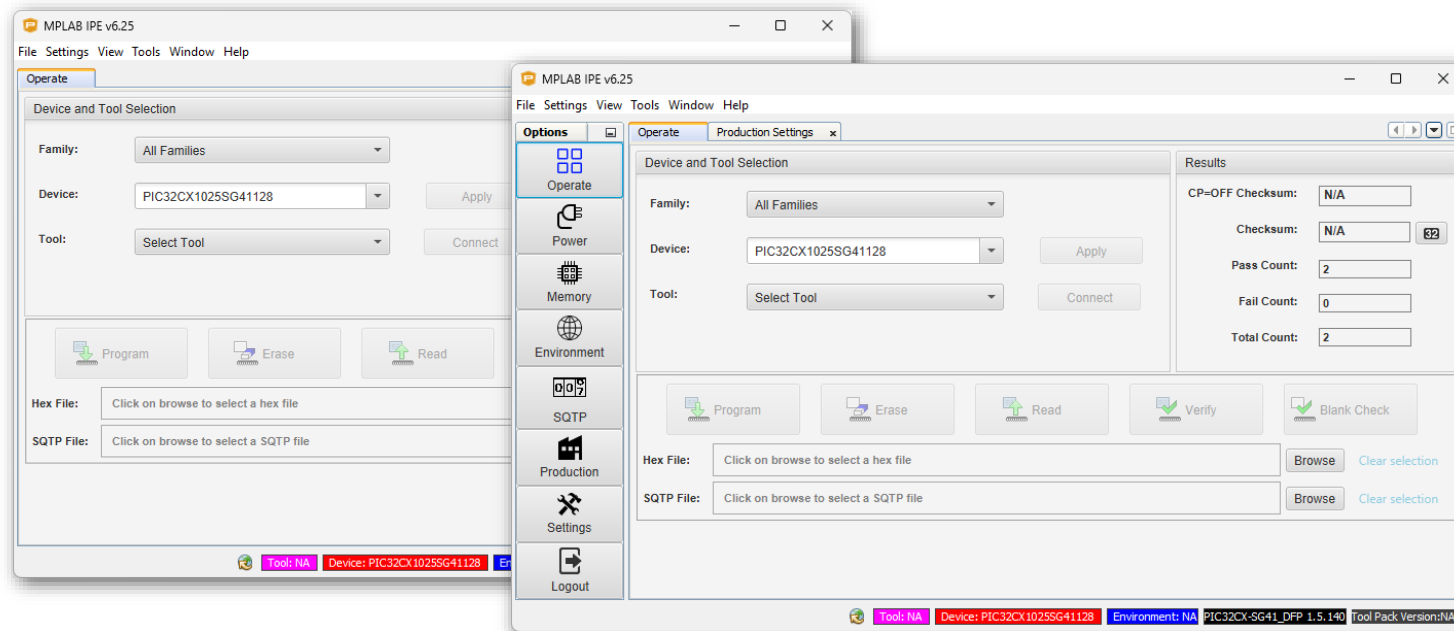
- MPLAB® MCC Harmony is a fully integrated embedded software development framework that provides flexible and interoperable software modules to simplify the development of value-added features and reduce your product's time to market.



Integrated Programming Environment

MPLAB® IPE

- It is a simplified application that provides an interface for production programmers that allows a technician to easily load, program and verify software, that provides quick access to key production programming features.
- MPLAB IPE can operate in Production Mode or Advanced Mode, for different technician-level user.



ISO C Compiler for MCU

MPLAB® XC Compilers

- The MPLAB® XC Compiler is a powerful, optimizing compiler that converts your ISO C programs into high-performance, assembly-language instructions to run bare metal on our MCU/MPU.
- This compiler offers many command-line options and language extensions that enable full access to the device's hardware features.

MPLAB XC8 Compiler

The MPLAB XC8 Compiler supports all 8-bit PIC® and AVR® microcontrollers (MCUs).

[View XC8 Compiler Downloads](#)

MPLAB XC16 Compiler

The MPLAB XC16 Compiler supports all 16-bit PIC microcontrollers (MCUs).

[View XC16 Compiler Downloads](#)

MPLAB XC-DSC Compiler

The MPLAB XC-DSC C Compiler supports all dsPIC DSCs.

[View XC-DSC Compiler Downloads](#)

MPLAB XC32 Compiler

The MPLAB XC32 C/C++ Compiler supports all 32-bit PIC and SAM MCUs and MPUs featuring Arm®, MIPS® and PIC32A cores.

[View XC32 Compiler Downloads](#)



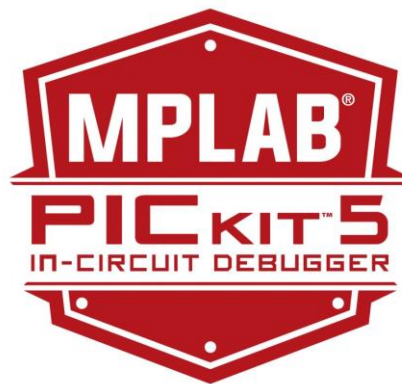
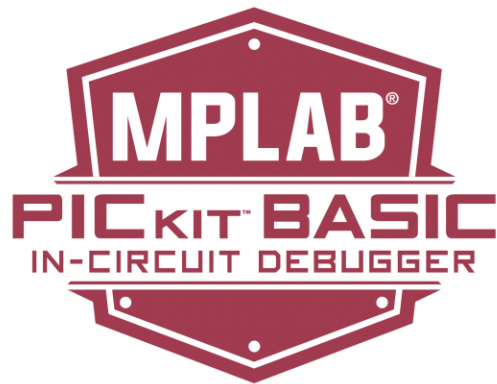
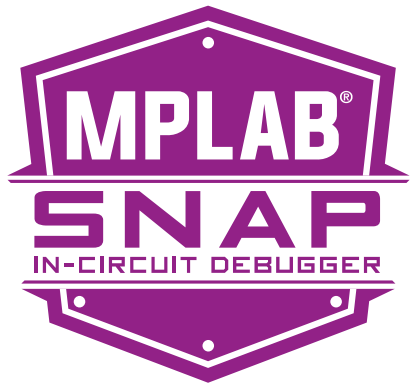
Soft Request In this Course

When the versions of the software and the framework differ from the created project, issues may occur. Therefore, it is recommended to use the following setting in this course.

- **MPLAB® X IDE v6.25** <https://www.microchip.com/mplab/mplab-x-ide>
 - **MPLAB® XC32 Compiler v4.60** <https://www.microchip.com/mplab/compilers>
 - **MPLAB® Code Configurator (MCC) v5.5.2**
 - **MCC Core v5.7.1**
 - **MCC Harmony Framework (From X IDE MCC Content Manager)**
 - MCC Harmony Core v1.5.5
 - Harmony 3 - Chip Support Package (csp) v3.22.4
 - Harmony 3 - Harmony Services v1.5.0
 - Harmony 3 - Quick Documentation Package v1.7.0
 - arm CMSIS v5.9.0
 - **MPLAB® X IDE Packs**
 - PIC32CX-SG41_DFP v1.5.140 (Device Family Pack)
 - CMSIS v5.4.0 or Newest (Device Family Pack)
- **MCC Core and Framework Update from X IDE > MCC > Content Manager**
- **PIC32CX-SG41_DFP Update from X IDE > Tools > Packs**

(Lasted Update : 2025/06/10)

Programmer/Debugger



Programmers and Debuggers

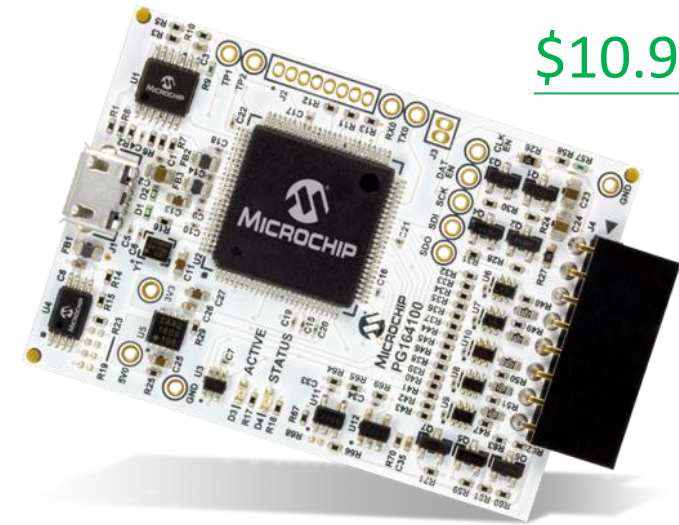
Feature	MPLAB® Snap	MPLAB® PICkit™ Basic	MPLAB® PICkit™ 5	MPLAB® ICD 5	Atmel-ICE	Power Debugger	MPLAB® ICE 4
Devices Supported	All	All	All	All	AVR, SAM	AVR, SAM	All
USB Interface	Micro USB	USB-C	USB-C	USB-C	Micro USB	Micro USB	USB 3.0
Power to Target			✓ (150mA)	✓ (1A)			✓ (1A)
Ethernet				✓ PoE			✓
Bluetooth			✓				
WiFi							✓
Over Voltage/Current Protection			✓	✓	✓	✓	✓
Software Breakpoints	✓	✓	✓	✓	✓	✓	✓
Programmer-to-Go			✓				
Trace				✓*	✓	✓	✓
Power Measurement/Profiling				✓*		✓	✓
Price	\$10.99	\$29.99	\$74.99	399.99	\$99.99	\$290.94	\$1,499.99

*Basic support, see product specifications

Programmer/Debugger

MPLAB® SNAP (Part No. PG164100)

- The MPLAB® SNAP In-Circuit Debugger/Programmer allows fast and easy debugging and programming of most PIC®, dsPIC®, AVR, SAM and CEC flash microcontrollers.
- **Features**
 - Affordable performance
 - Matches silicon clocking speed
 - Target voltage of 1.20V to 5.5V
 - Portable USB-powered
 - CE and RoHS-compliant
 - 8-pin single in-line header
 - Backward compatible for demo boards, headers and target systems using 2-wire JTAG and ICSP



\$10.99

Programmer/Debugger

MPLAB® PICkit™ Basic (Part No. PG164110)

- The MPLAB® PICkit™ Basic In-Circuit Debugger/Programmer allows fast and easy debugging and programming of most PIC® and AVR® MCUs, dsPIC® DSCs and SAM Arm® Cortex®-M based microcontrollers.
- **Features**
 - USB Type-C® interface for easy PC connection
 - 8-pin single in-line header with beginner-friendly flying leads
 - Target voltage of 1.20V to 5.5V
 - Supports 4-wire JTAG, Serial Wire Debug (SWD), UPDI, PDI, SPI, debugWIRE and TPI programming
 - Backward compatible with two-wire JTAG and ICSP
 - Includes an adapter to convert from the 8-pin inline to the standard 10-pin Arm SWD connector
 - Easy firmware recovery with a push button



\$29.99

Programmer/Debugger

MPLAB® PICKit 5 (Part No. PG164150)

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



\$74.99

Programmer/Debugger

MPLAB® ICD 5 (Part No. DV164055)

- The MPLAB® ICD 5 In-Circuit Debugger/Programmer is Microchip's fast, cost-effective debugging and programming tool for PIC®, dsPIC®, AVR, SAM and CEC flash microcontrollers.
- **Features**
 - Target voltage of 1.2V to 5.5V.
 - Can supply up to 1A of power to the target
 - Power monitoring
 - Overvoltage/ Over-current protection
 - Supports multiple breakpoints, stopwatch and source code file debugging
 - Selectable pull-up/pull-down option to the target interface
 - PoE+ (IEEE® 802.3at) offers a convenient
 - Fast Ethernet connectivity supports speeds up to 100 Mbps
 - Continuous Integration/Continuous Delivery (CI/CD) Support

\$399.99



Programmer/Debugger

MPLAB[®] ICE 4 (Part No. DV244140)

- The MPLAB ICE 4 in-circuit emulator system boosts productivity with feature-rich programming and debugging for PIC[®], dsPIC[®], AVR, SAM and CEC flash microcontrollers and microprocessors.
- **Features**
 - Full-Speed Real-Time Emulation
 - Target voltage of 1.2V to 5.5V.
 - Can supply up to 1A of power to the target
 - Power Monitoring (Supported on All Devices)
 - Supports multiple breakpoints, stopwatch and source code file debugging
 - Selectable pull-up/pull-down option to the target interface
 - Ethernet connectivity with speeds up to 100 Mbps
 - Wi-Fi Access Point Connectivity
 - Connection to Wi-Fi using Wireless Station Mode
 - Continuous Integration/Continuous Delivery (CI/CD) Support

\$1499.99



Programmer/Debugger

Atmel-ICE (Part No. ATATMEL-ICE)

- **Supports programming and debugging all Cortex-M based SAM, AVR and UC3 based parts**
 - Programming/ Debugging AVR/SAM MCU/CPU over JTAG, SD, PDI, SPI, TPI and aWire etc.
 - Target operating voltage range 1.62V to 5.5V
 - ITM serial trace support on Cortex-M based SAM MCUs
- **Supported in Atmel Studio, IAR, Keil and more**
 - Any IDE that supports the CMSIS-DAP protocol can support Atmel-ICE



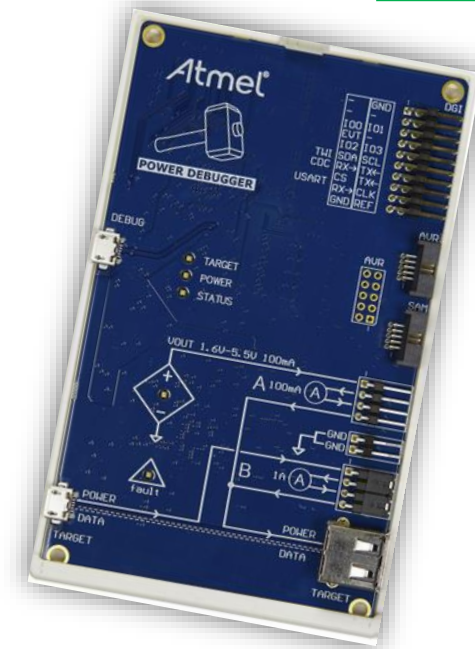
\$99.99

Programmer/Debugger

Power Debugger (Part No. atpowerdebugger)

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

\$290.94

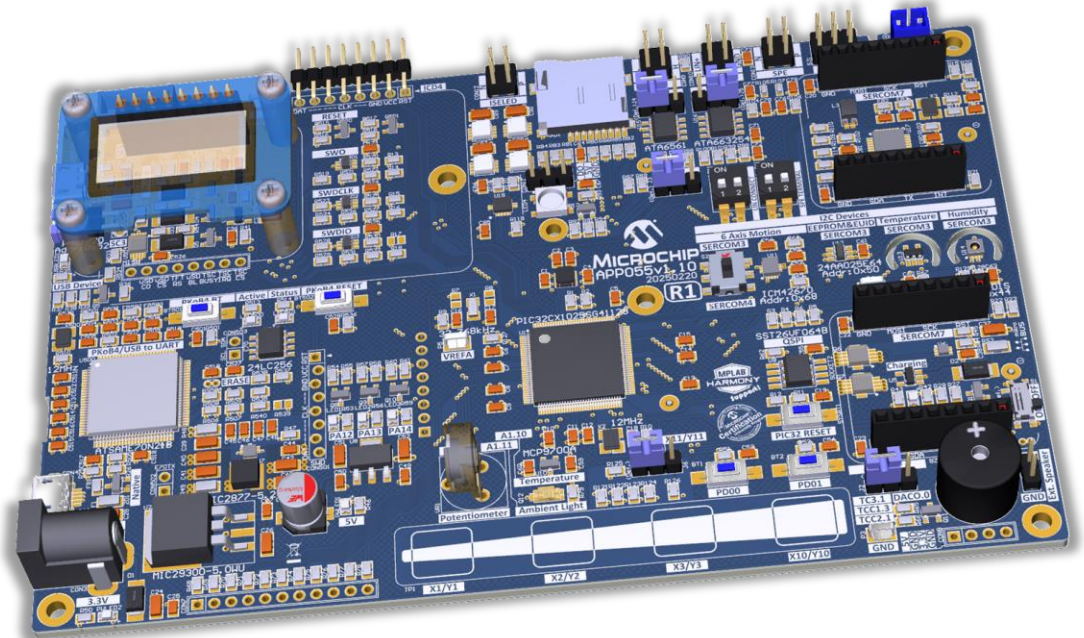


PIC32CX-SG EVB

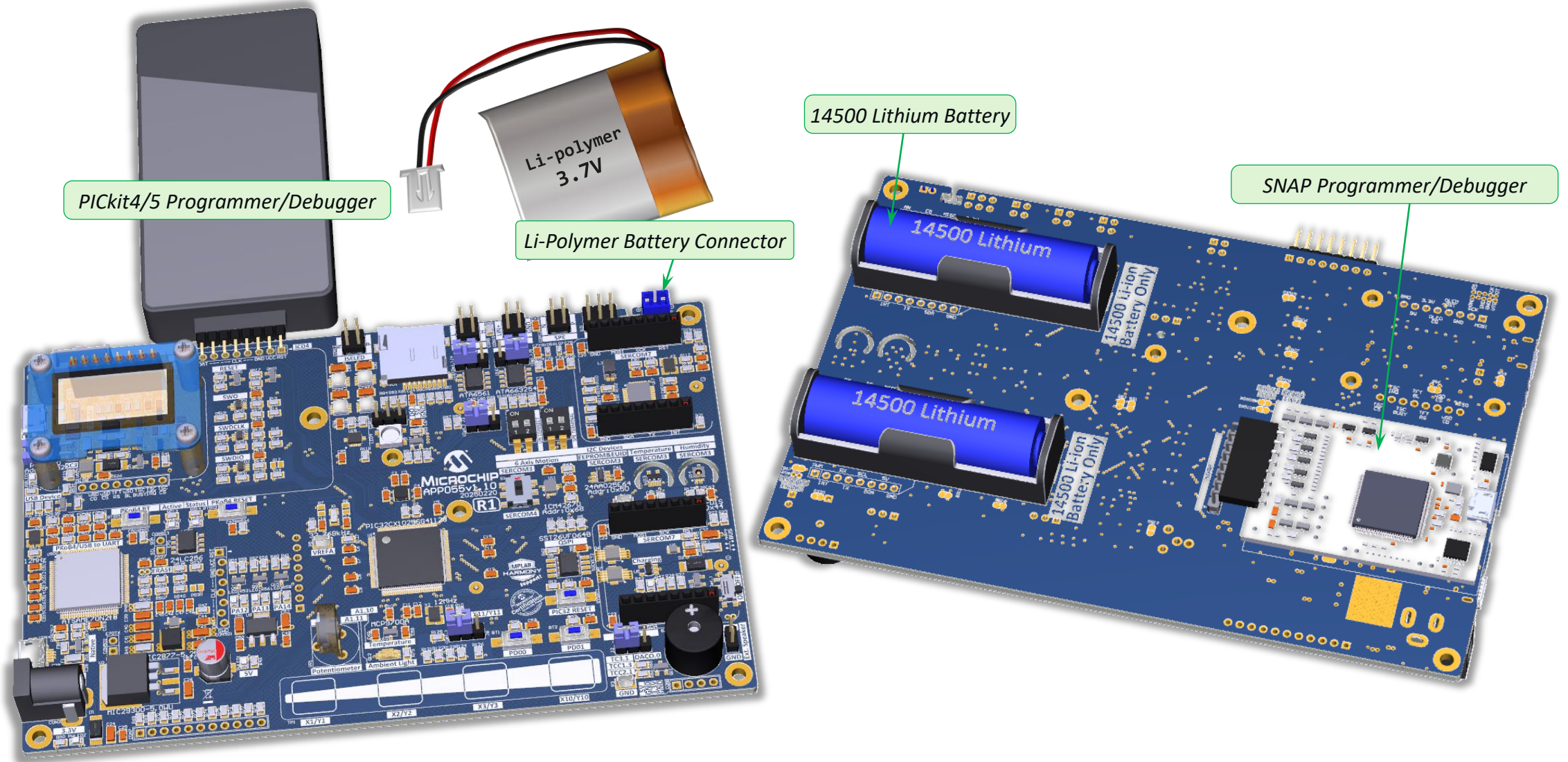
APP055 v1.10_{R1} Introduction

APP055 v1.10_{R1} Features

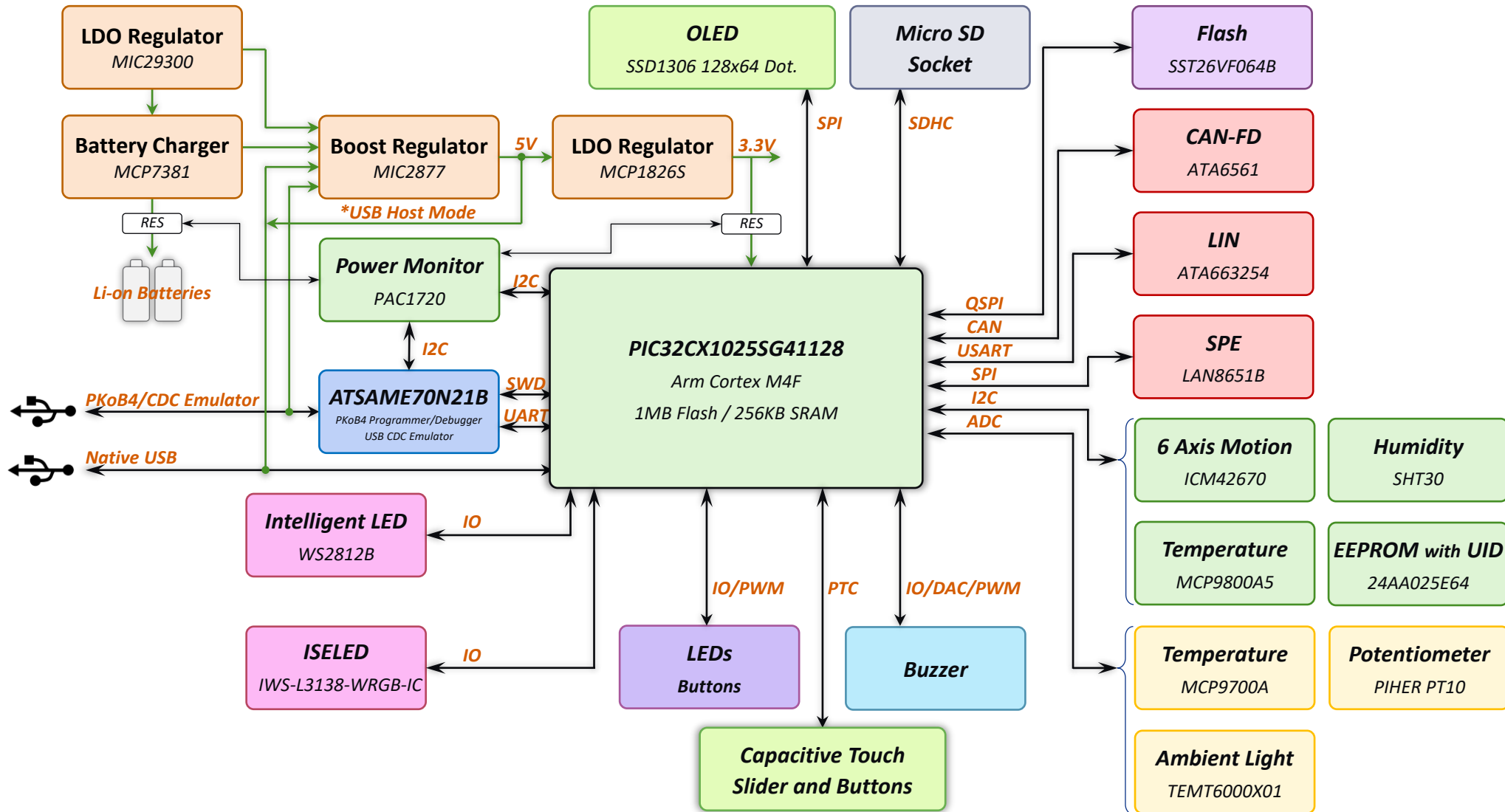
- **PIC32CX1025SG41128** 32Bit Arm Cortex M4F Microcontroller with **FPU**.
- **Built-in PKoB4 Programmer/Debugger with USB CDC Virtual COM.**
- **Multiple Power Source** and support Li-on battery with battery charger.
- **ISELED, Single Pair Ethernet, CAN-FD, LIN** support ready.
- **Capacitive QTouch®** Slider & Buttons with Driven Shield.
- **mikroBUS™** Click Board Socket ready.
- **Digital IO : 2 x Buttons** and **3 x LEDs** and **WS2812B**.
- **Analog : Potentiometer, Temperature Sensor** and **Ambient Light Sensor**.
- **I2C : Temperature, EEPROM, 6 Axis Motion** and **Humidity Sensor**.
- **SPI : OLED (128 x 64, SSD1306 Compliant)**
- **QSPI : Flash Memory.**
- **SDHC : MicroSD Socket.**
- **PWM, DAC : Buzzer.**



APP055 v1.10_{R1} Accessories



APP055 v1.10_{R1} Block Diagram



Course Materials

- The course provides rich materials and will be used continuously in the course. You can check the folder at any time and find the lecture, exercises or tools you need for the current section.

Course Materials	Slides (PDF)	<i>Lecture for courses, Includes the peripheral introduction of the MCU, how to use MCC Harmony and the steps for exercises.</i>
	Labs	<i>Labs for courses, Includes the Copy and Paste Textbook, Resources and Solution for exercises.</i>
Supplementary Documents	Appendix	<i>Appendix for courses, includes additional tools and factory demo.</i>
	Documents	<i>Supplementary documents for courses, Includes the Datasheet and Schematic for APP055 EVB or PIC32CX-SG MCU.</i>
Development Environment Installation Guide	Pre-Work	<i>This is the Pre-Work you need to do before the course begins, including downloading and installing development tools. All installation steps can be found in the manual, be sure to try the steps on your computer before the course begins. You can find the development tools and resources you need in the Tools and MCC_Framework folders.</i>
	Tools	
	MCC Framework	

Getting Started with First Project

Lab0 First Project

Try to create your first project to use MPLAB X IDE and MCC.

Lab0 First Project

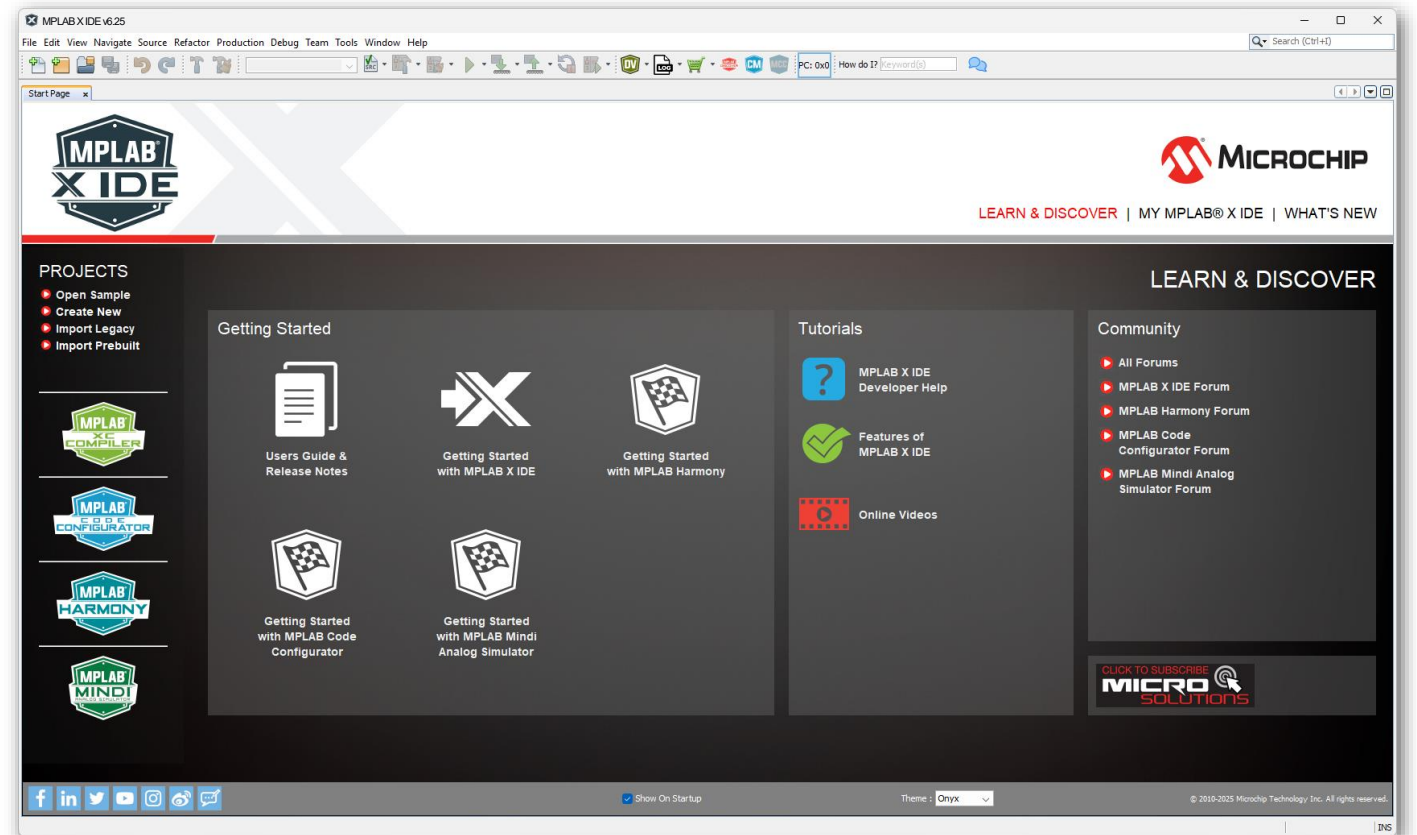
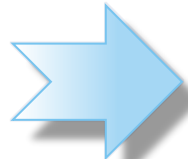
- Try to **create your first project** to use MPLAB X IDE and MCC.
- Learn how to use edit, build, project manager, debug and programming functions.

How to Start ?

⚙️ Lab0 First Project

Step 1

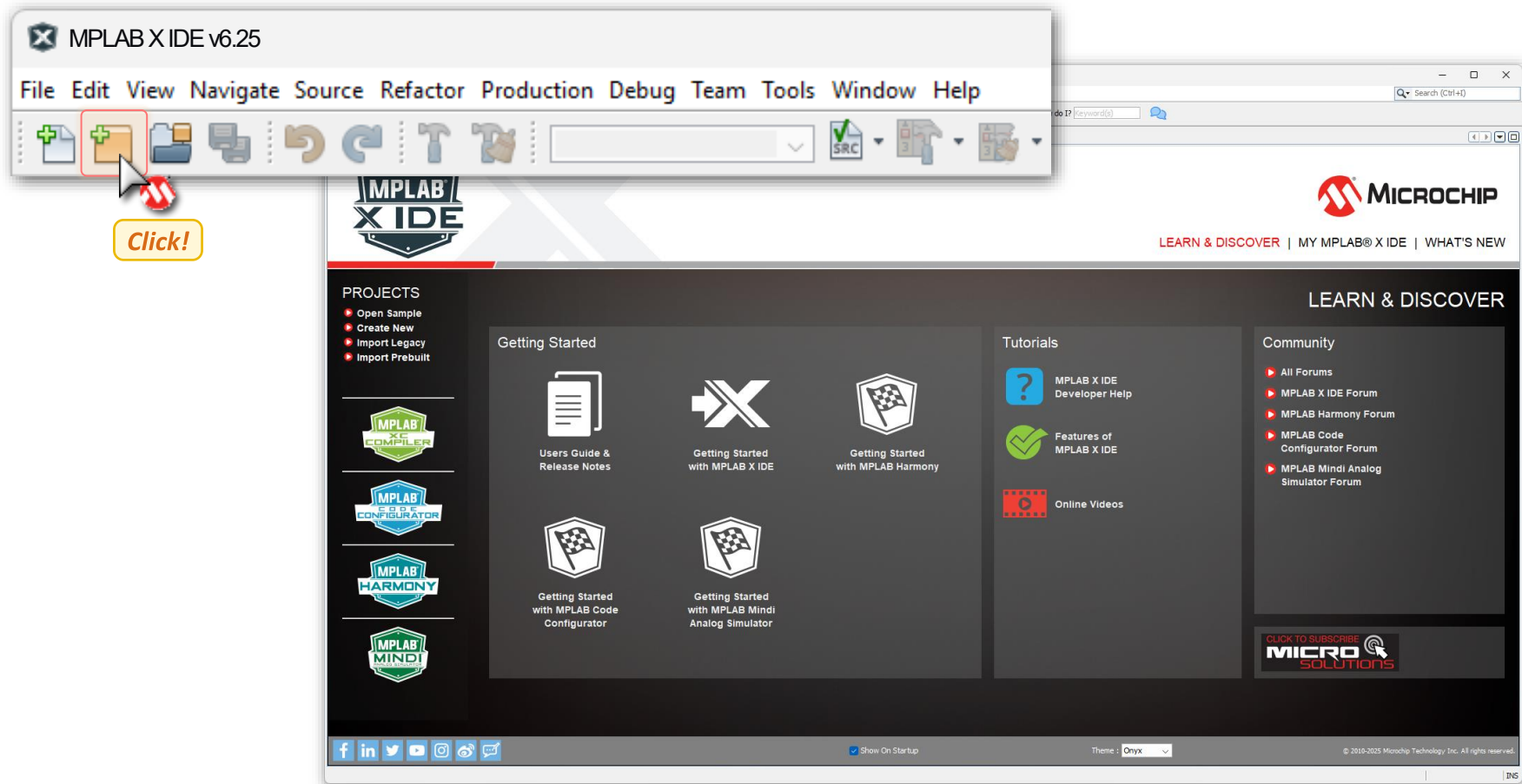
- Open **MPLAB X IDE v6.25** firstly.



⚙️ Lab0 First Project

Step 2

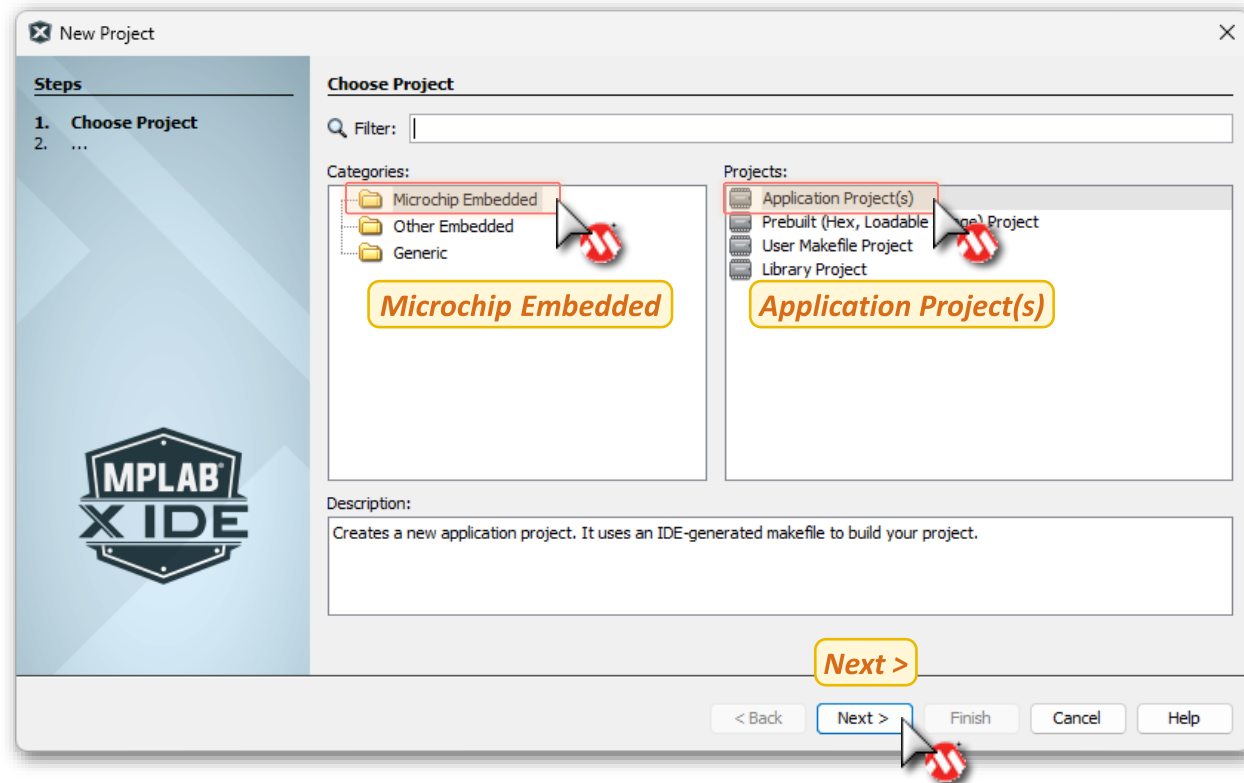
- Select to **File > New Project** or **Click icon** 



Lab0 First Project

Step 3

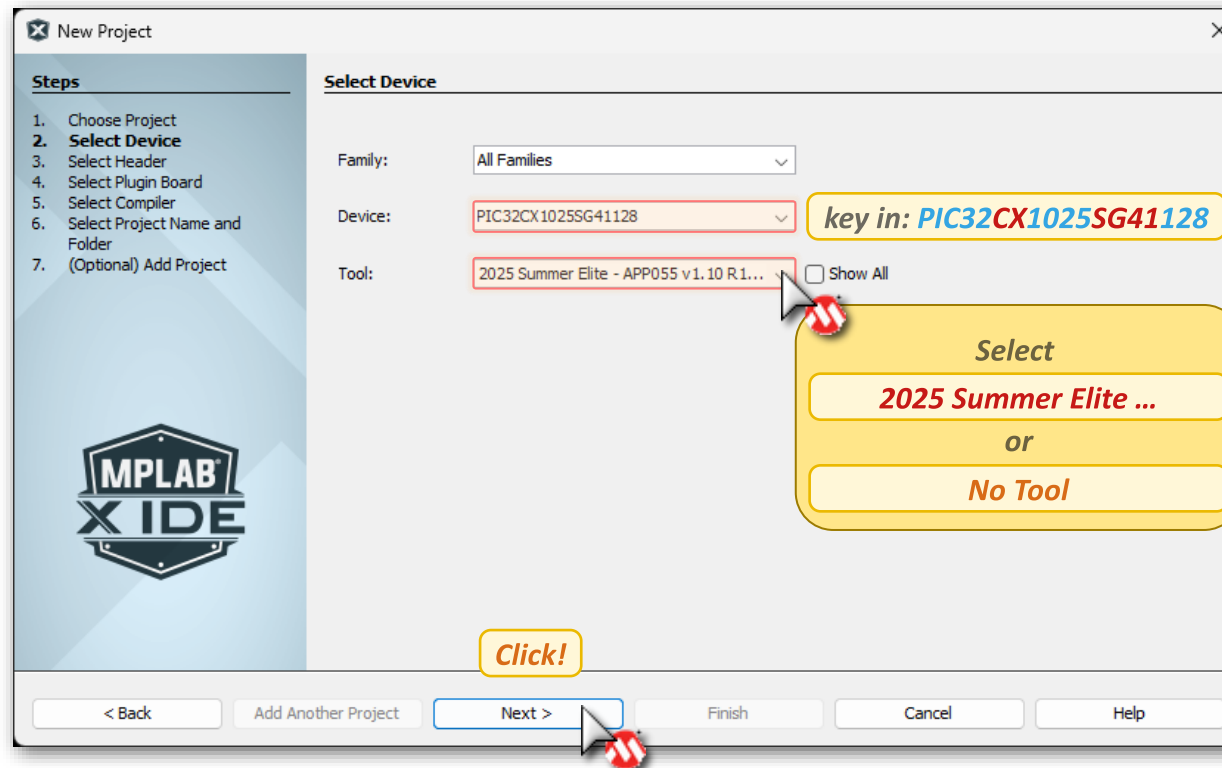
- Categories : **Microchip Embedded**
- Projects : **Application Project(s)**



Lab0 First Project

Step 4

- Family : All Families
- Device : **PIC32CX1025SG41128** *Don't miss any character*
- Tool : **2025 Summer Elite - APP055 v1.10 R1...** or No Tool

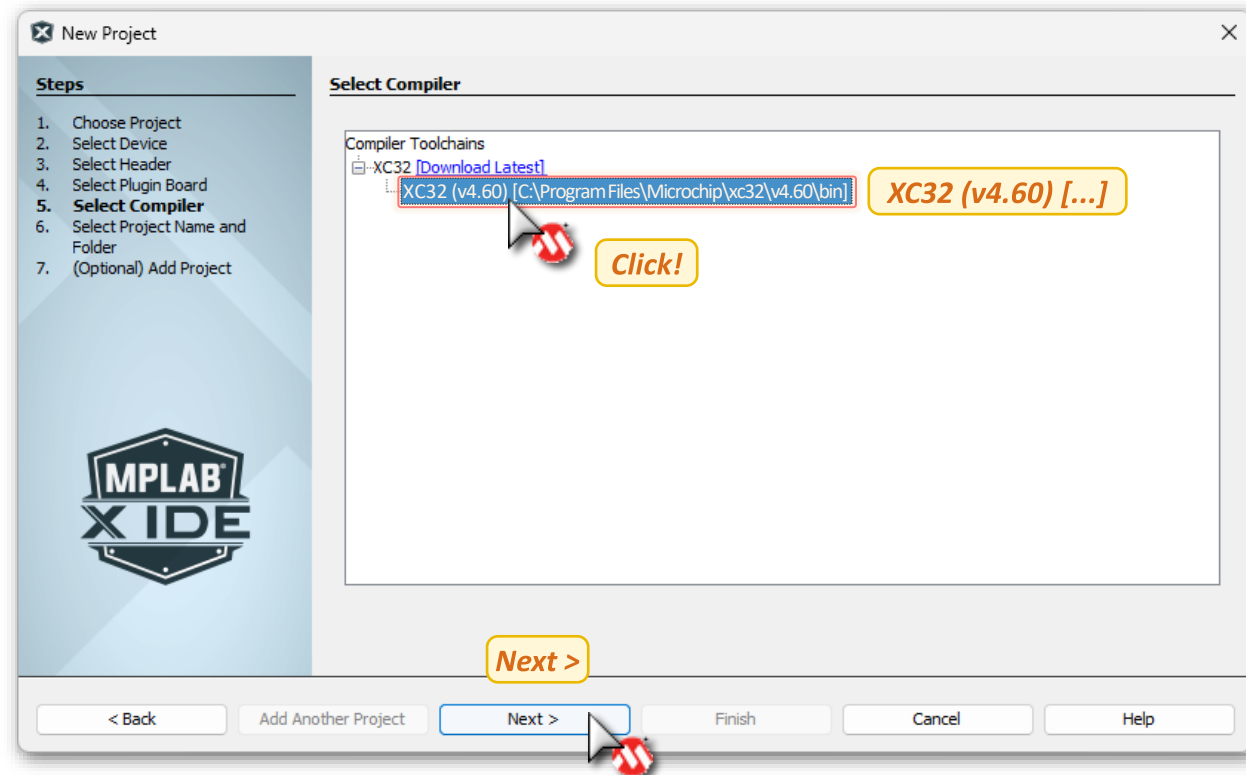


Lab0 First Project

Step 5

Assign Compiler Toolchains

- XC32 : **XC32 (v4.60)[...]**

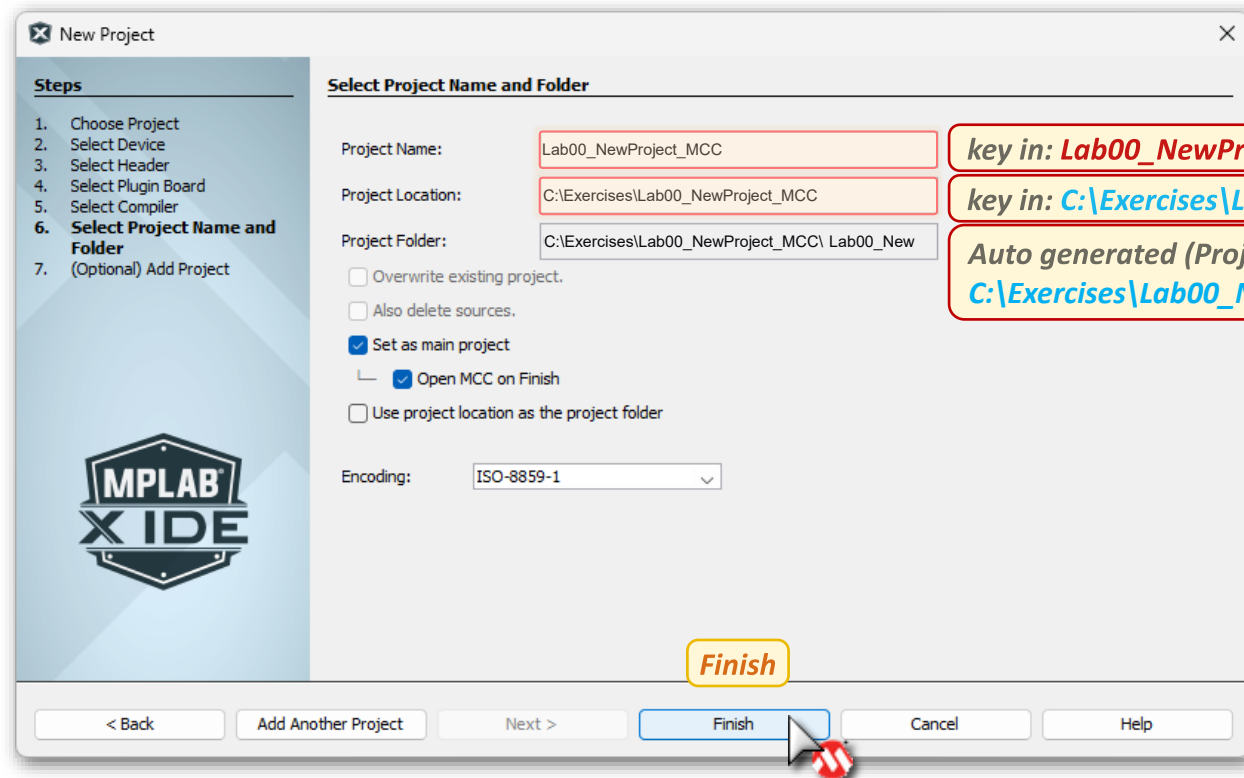


Lab0 First Project

Step 6

Configure the Project Location and Project Folder Name.

- Project Name : **Lab00_NewProject_MCC** *No white space of project name or path*
- Project Location : **C:\Exercises\Lab00_NewProject_MCC**



key in: **Lab00_NewProject_MCC**

key in: **C:\Exercises\Lab00_NewProject_MCC**

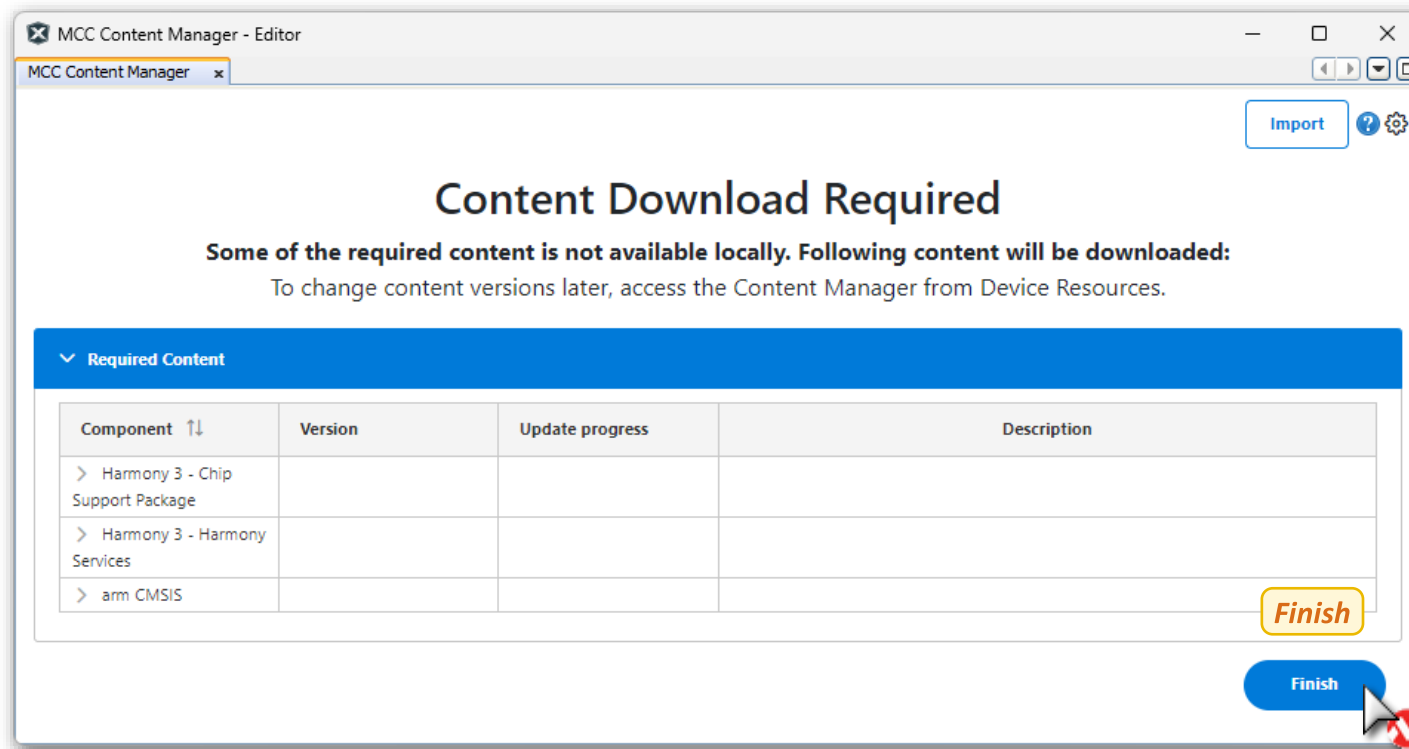
Auto generated (Projects Folder) :
C:\Exercises\Lab00_NewProject_MCC\Lab00_NewProject_MCC.X

** Check the path of Projects Folder.*

⚙️ Lab0 First Project

Step 7

- Click **Finish** and **Waiting for automatic download MCC Framework**.



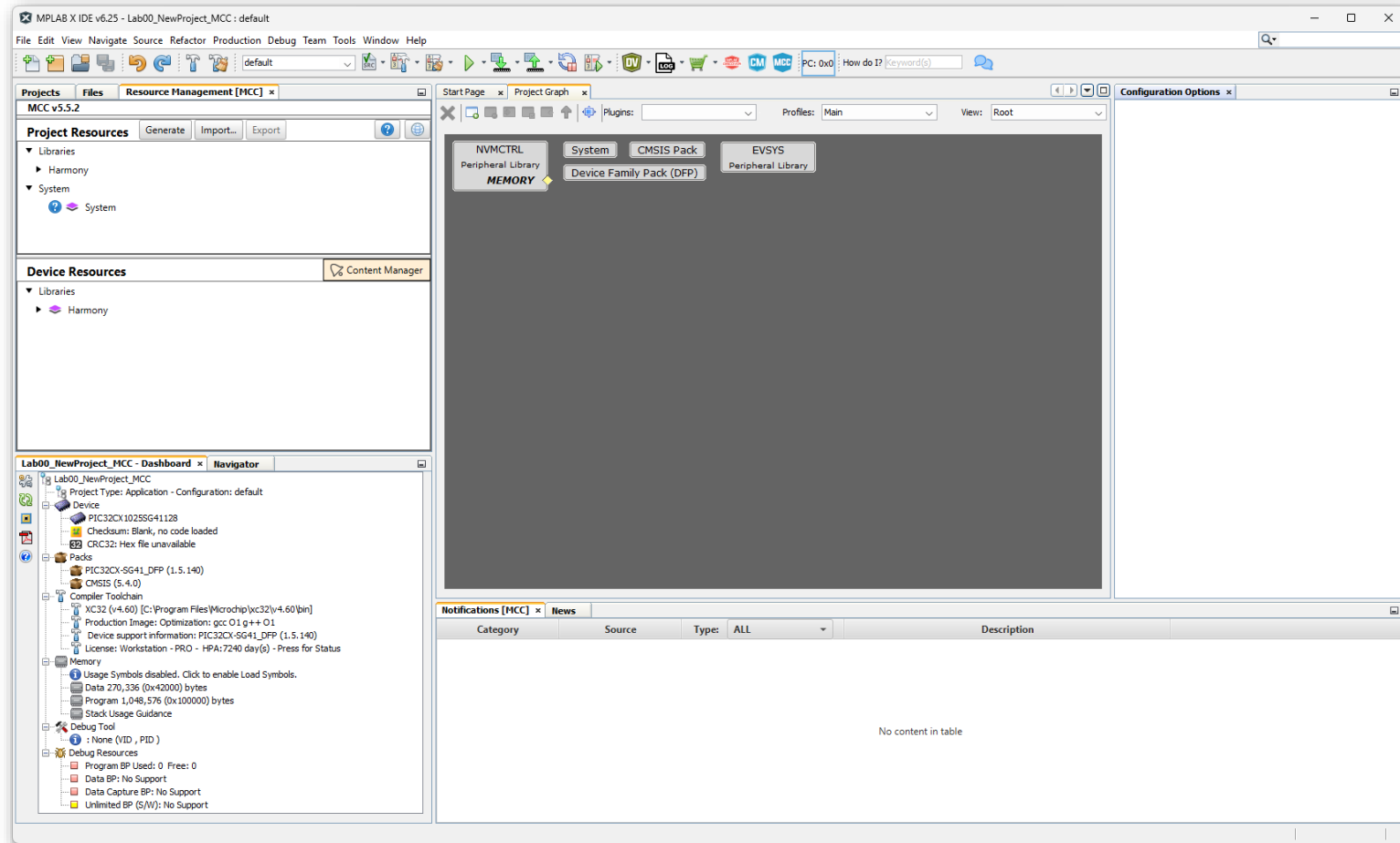
** MCC will download the latest Framework, which only needs to be performed once.*

** This step will not occur in the course because the MCC Framework is already installed on the computer.*

Lab0 First Project

Step 8

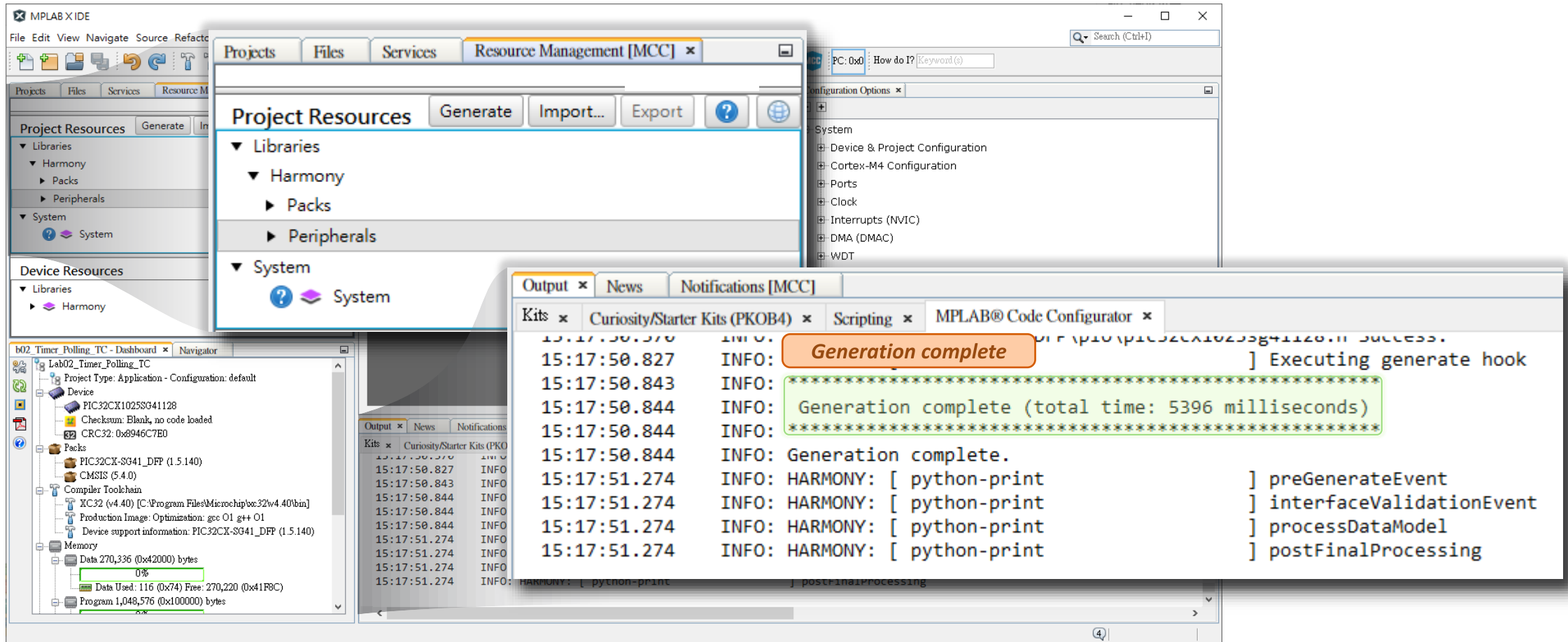
- MCC Harmony Configurator interface show up.



Lab0 First Project

Step 9

- Click on **Generate** button to **generate** your code after configuration.

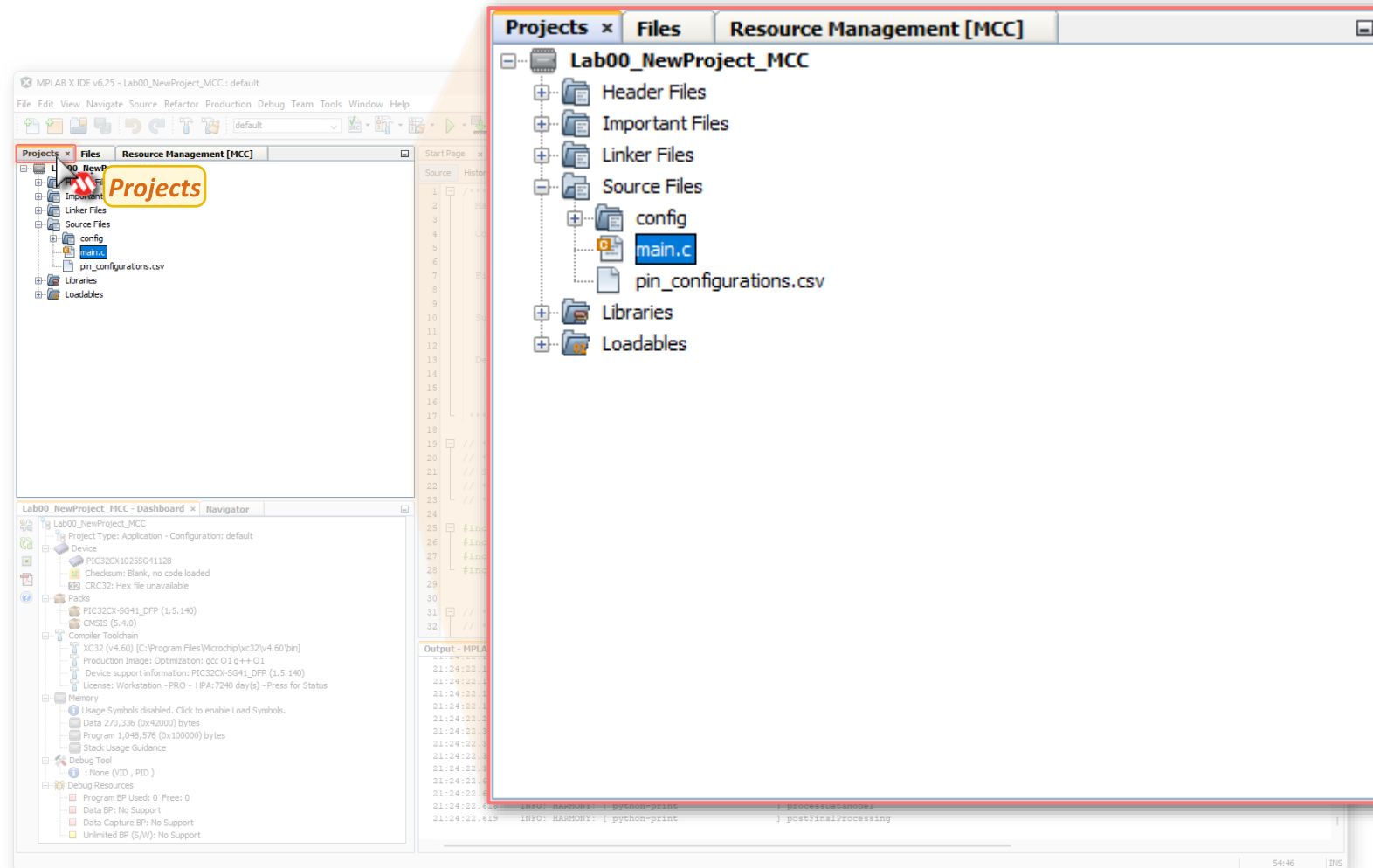


The screenshot displays the MPLAB X IDE interface. The **Resource Management [MCC]** window is open, showing the **Project Resources** tab with the **Generate** button highlighted. The **Output** window shows the following log:

```
15:17:50.827 INFO: Generation complete
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

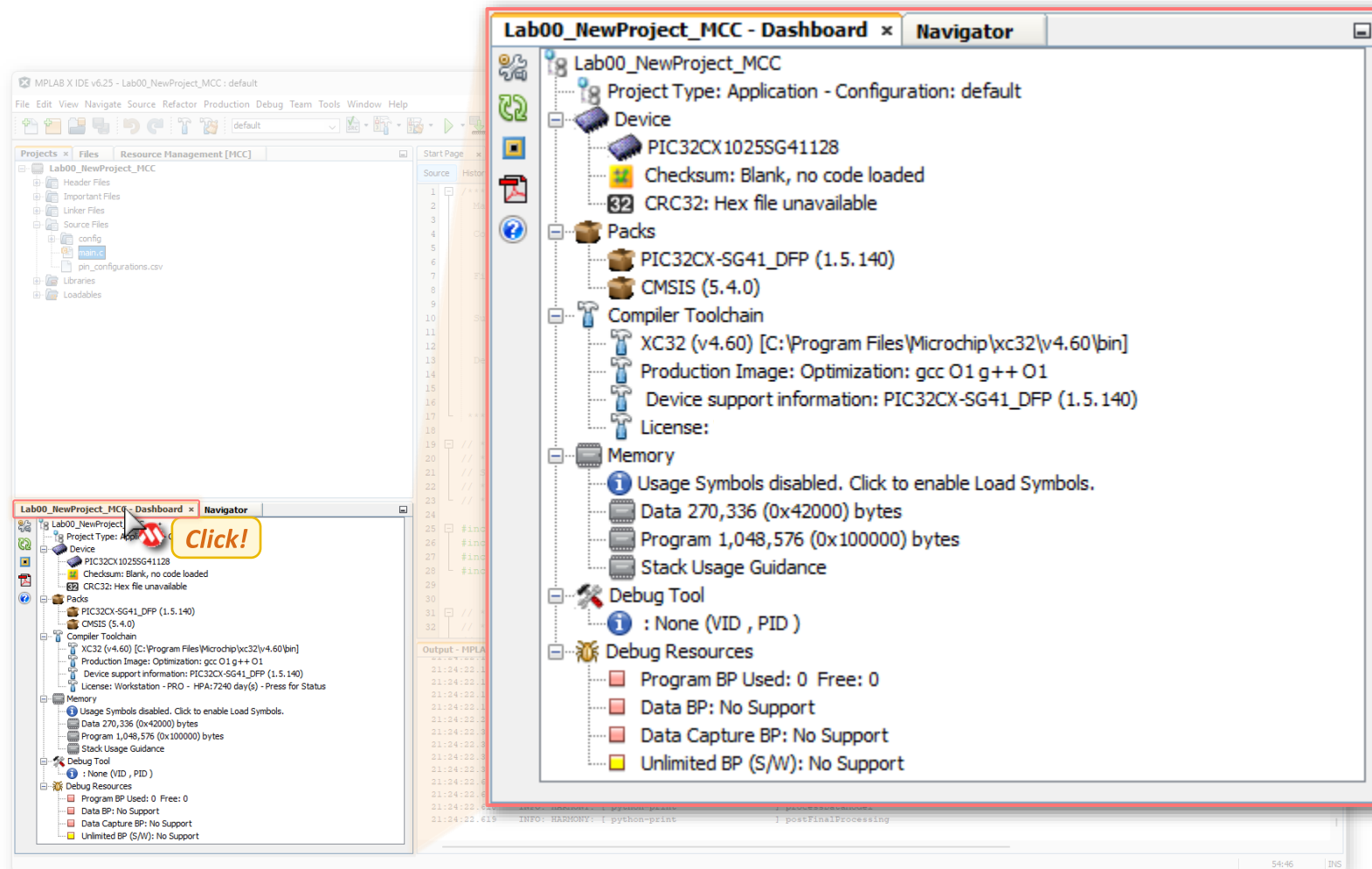
MPLAB X IDE Interface

- Project window.



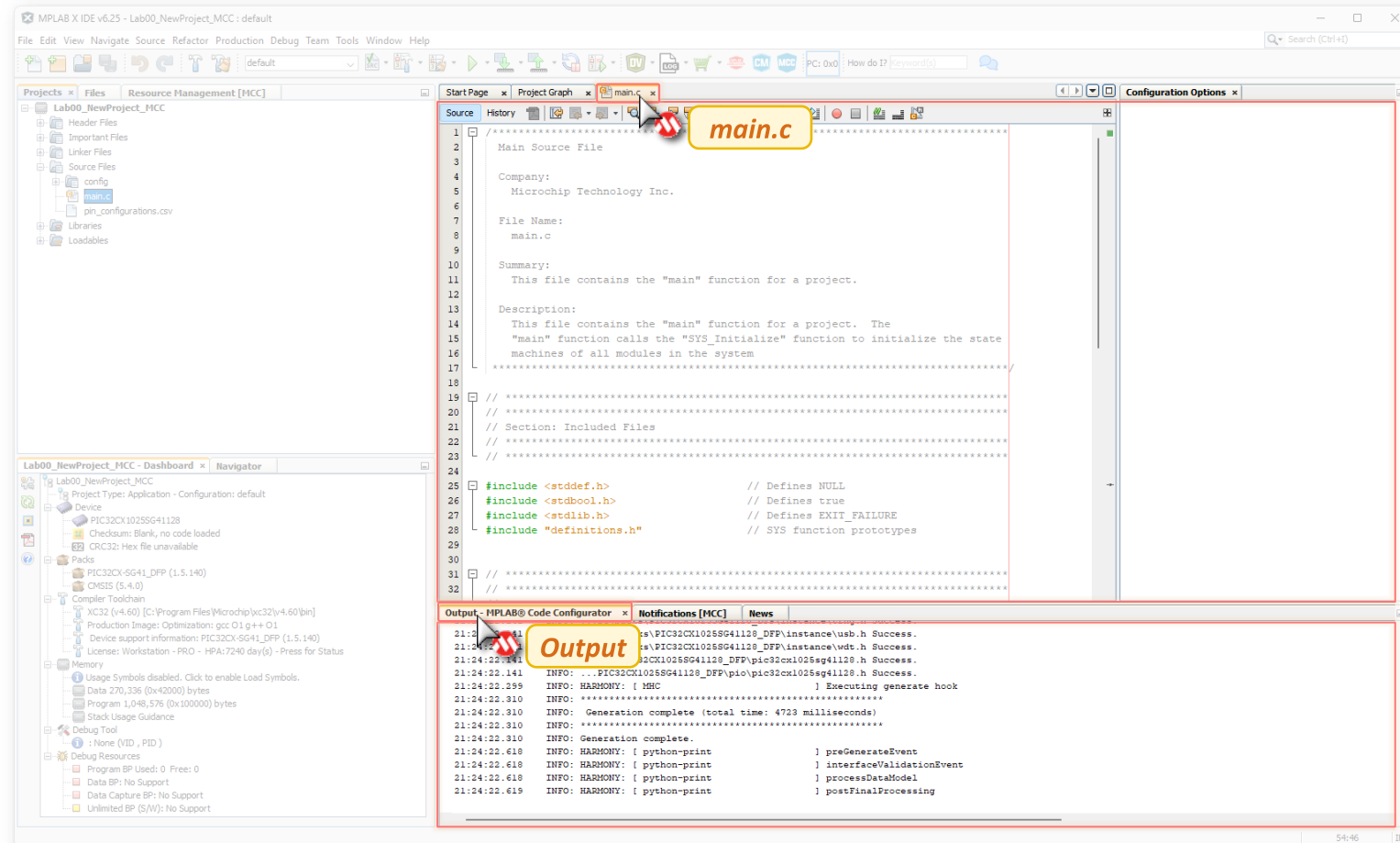
MPLAB X IDE Interface

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



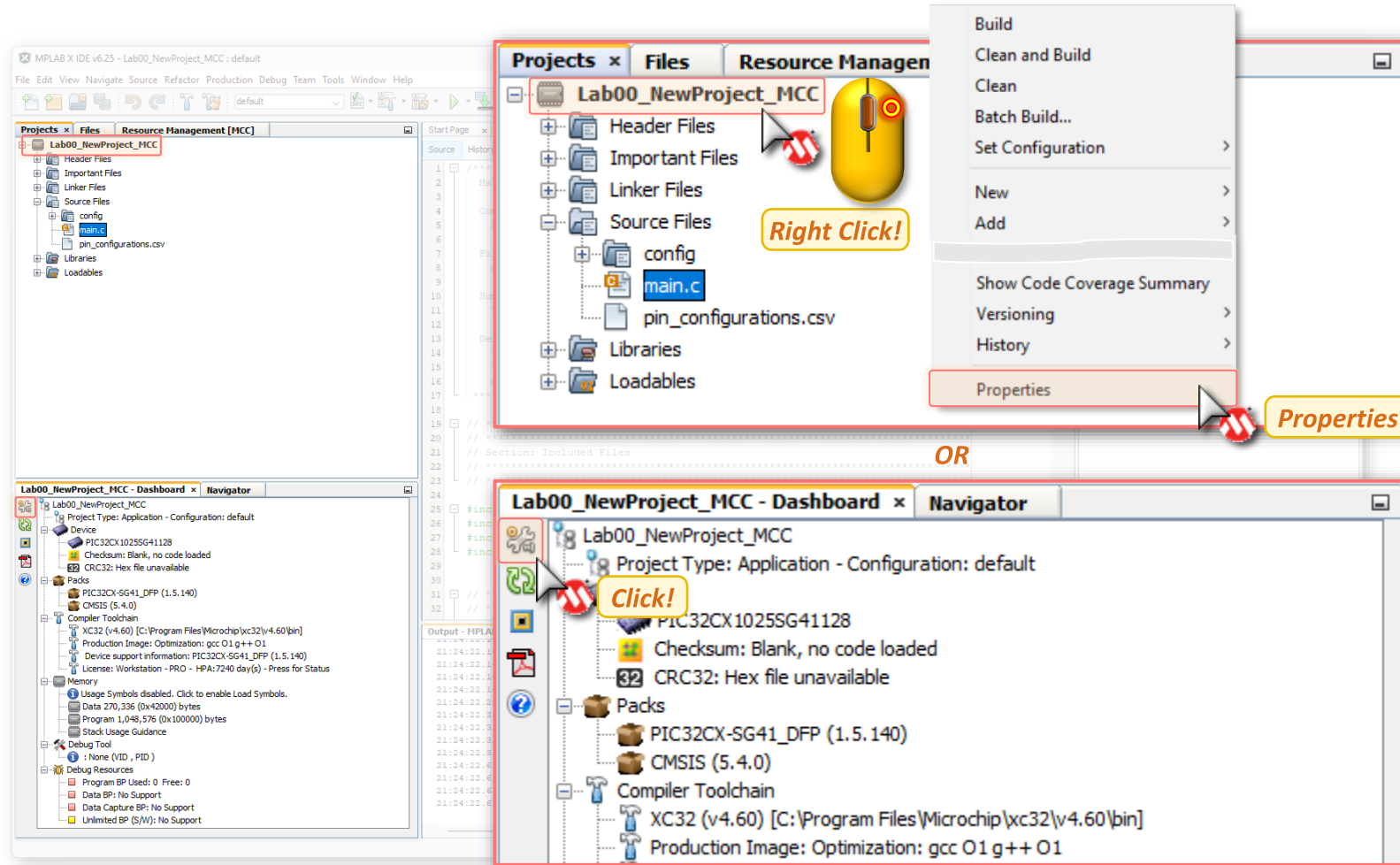
MPLAB X IDE Interface

- Source code and Output window.



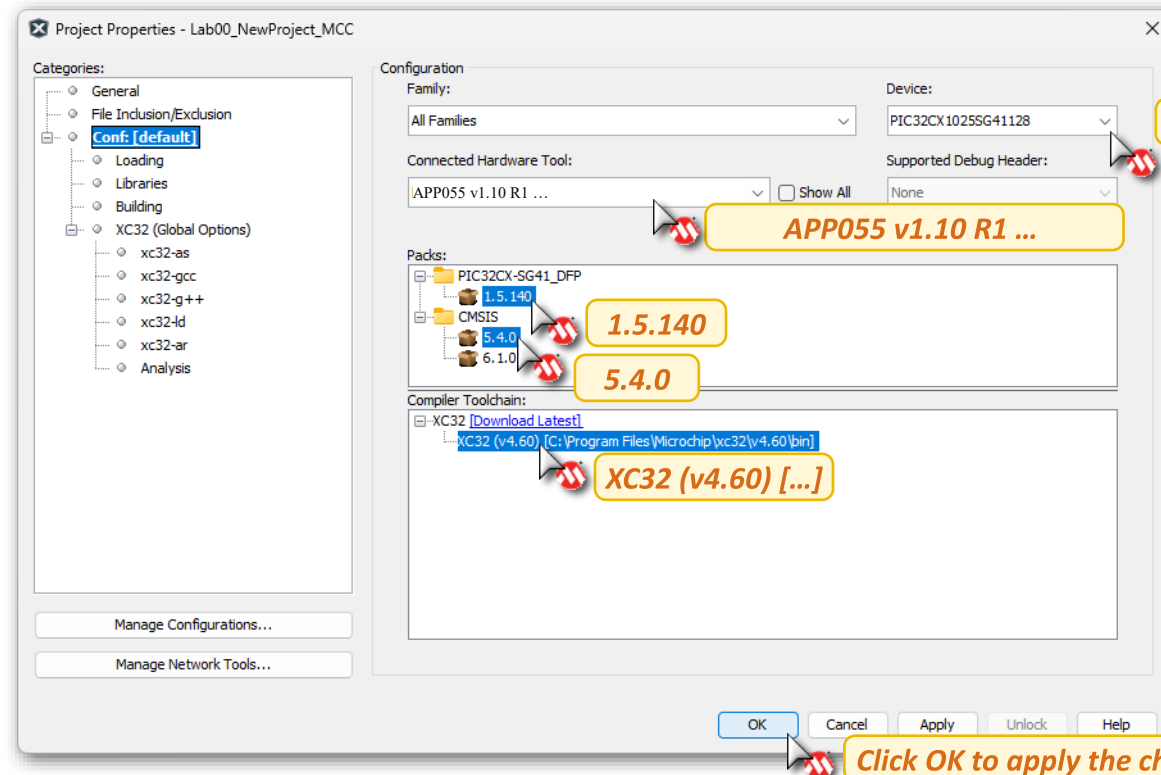
Project Properties

- Click  on Dashboard or Right click on project and select the **Properties**.



Project Properties

- Connected Hardware Tool : APP055 v1.10 R1 ...
- Packs - PIC32CX-SG41_DFP : 1.5.140
- Packs - CMSIS : 5.4.0 or Newest
- Compiler Toolchain : XC32 (v4.60) [...]



PIC32CX1025SG41128

* Please confirm the selected Device again !

APP055 v1.10 R1 ...

1.5.140

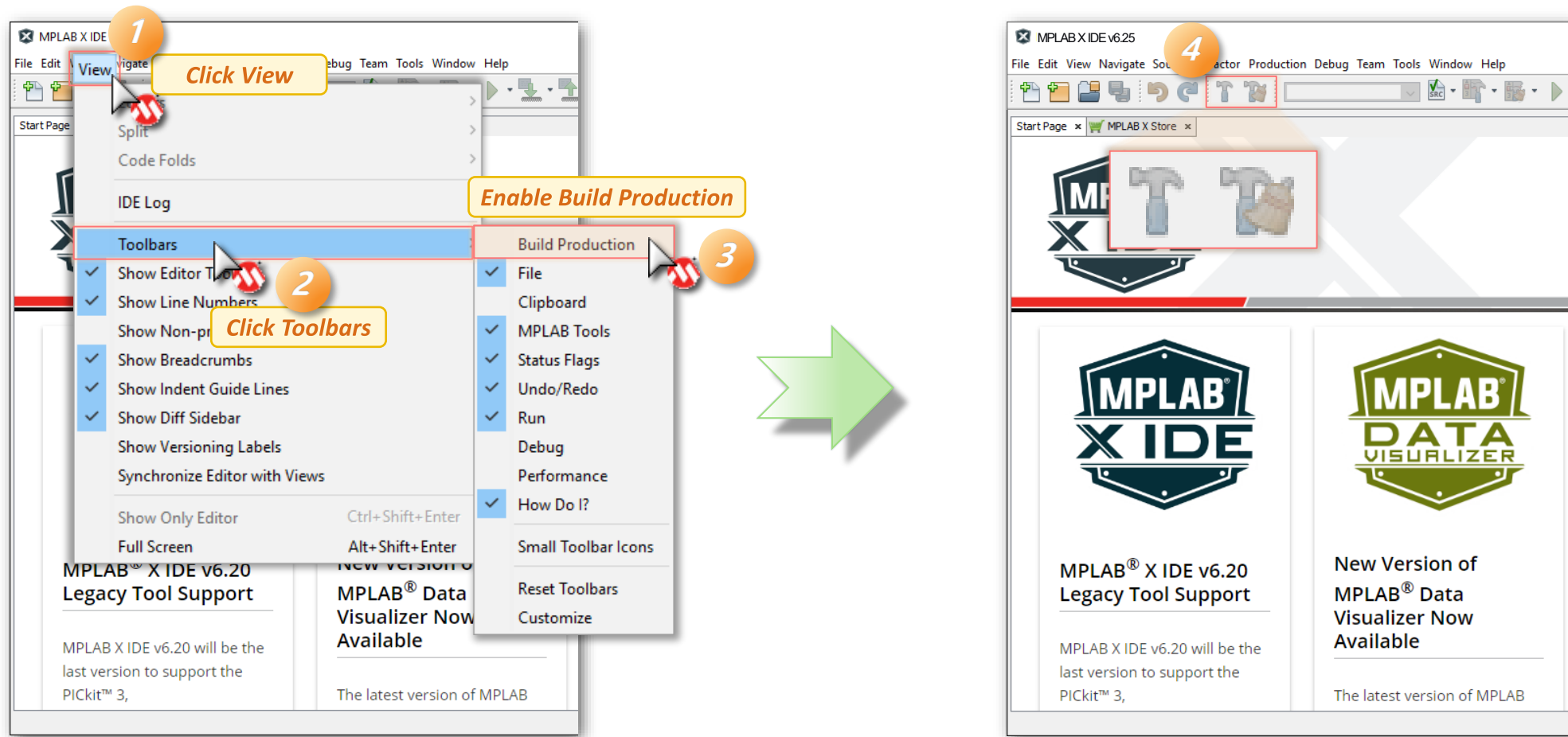
5.4.0

XC32 (v4.60) [...]

Click OK to apply the changed for Project Properties

Build Production Tools

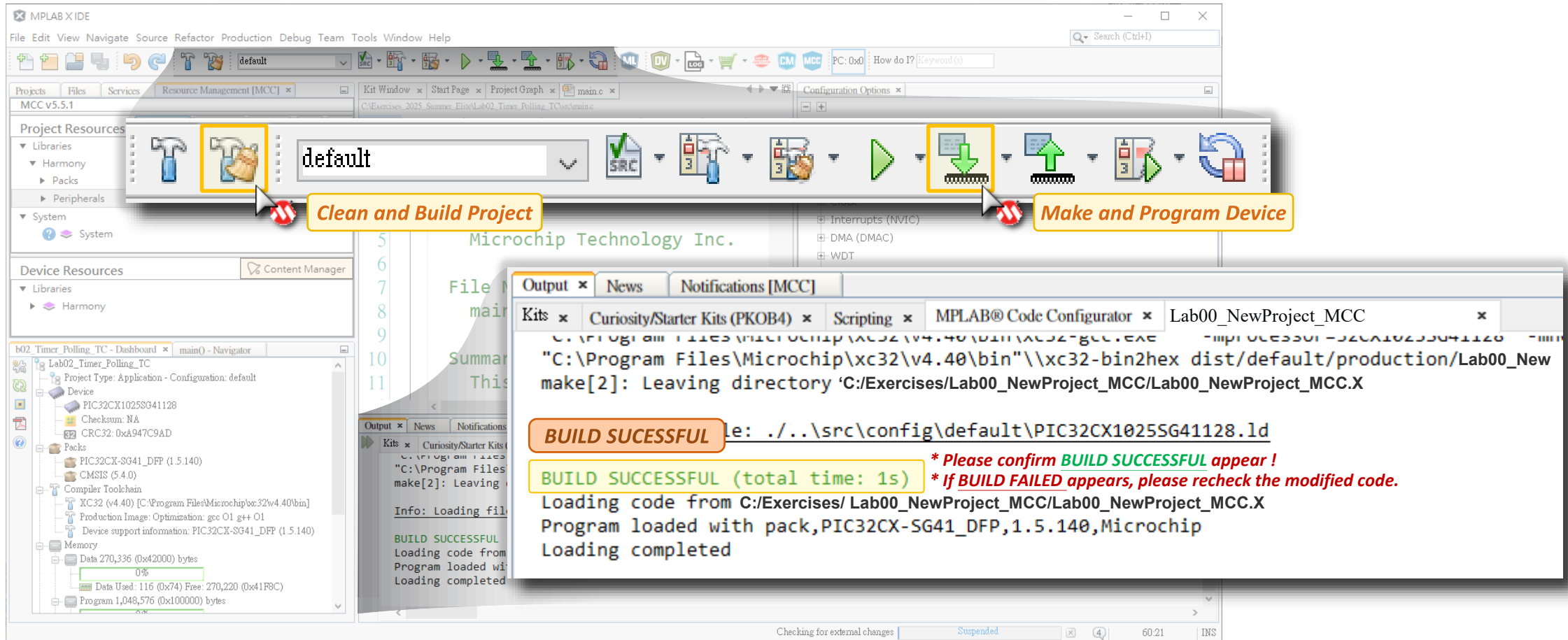
- Add the Build Production to Toolbars to make it easy to Clear and Build projects.



⚙️ Lab0 First Project

Step 10

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains several icons, with two highlighted by yellow boxes and red arrows. The first icon, a hammer and wrench, is labeled "Clean and Build Project". The second icon, a green arrow pointing down to a chip, is labeled "Make and Program Device".

The "Output" window shows the following text:

```
Le: ../../src/config/default/PIC32CX1025SG41128.ld
BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/ Lab00_NewProject_MCC/Lab00_NewProject_MCC.X
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

Below the output text, there are two red text boxes with white text:

- BUILD SUCCESSFUL**
- BUILD SUCCESSFUL (total time: 1s)**

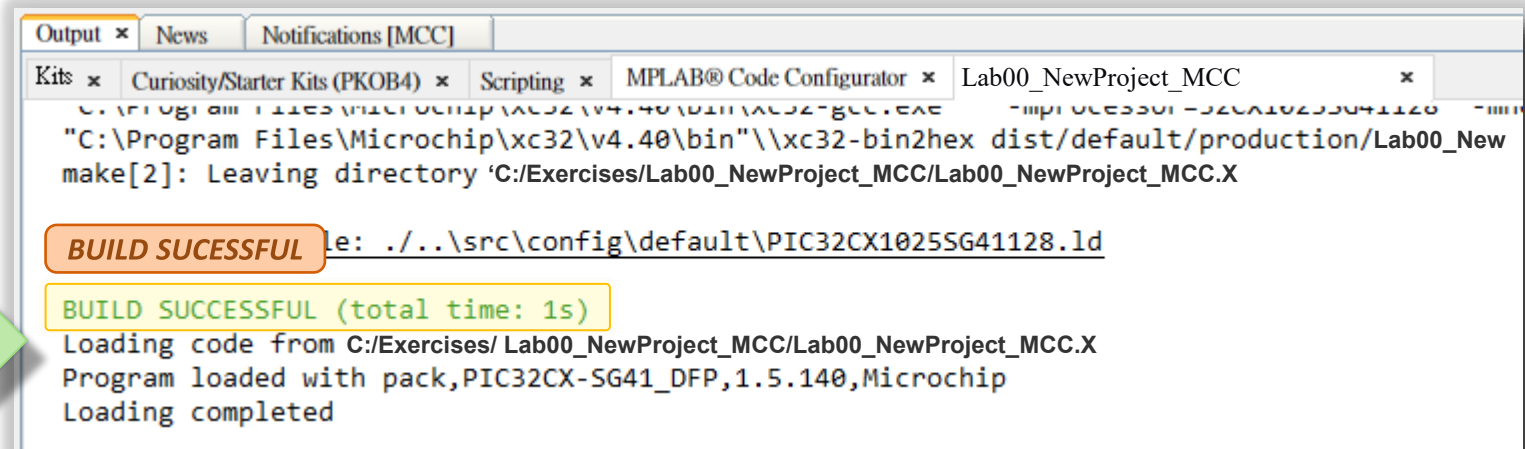
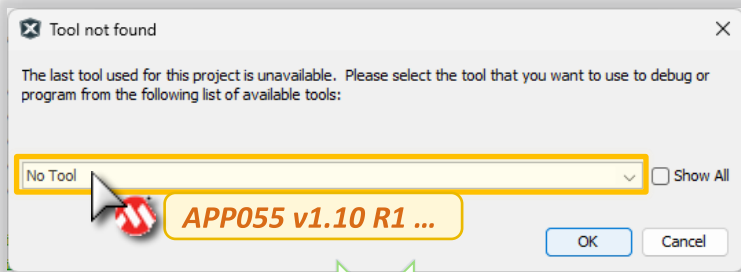
Below these, there are two red text boxes with white text:

- * Please confirm BUILD SUCCESSFUL appear !**
- * If BUILD FAILED appears, please recheck the modified code.**

Lab0 First Project

Hint - Tool not found

- If your project has not selected the tool before, "**Tool not found**" window will appear, please select "**APP055 v1.10 R1 ...**" and then click OK, programming will continue.

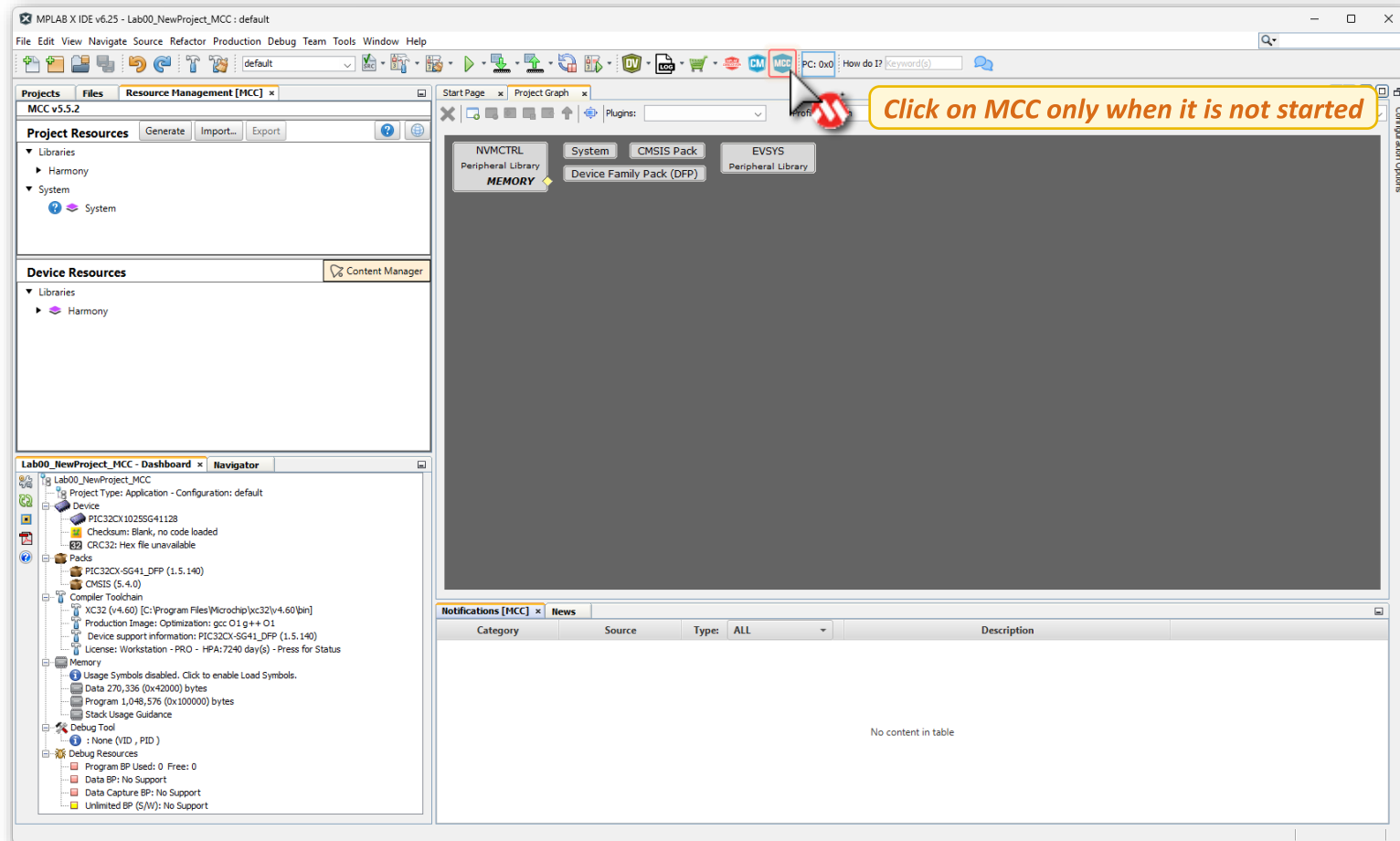


MPLAB® Code Configurator (MCC)

Harmony Interface Introduction

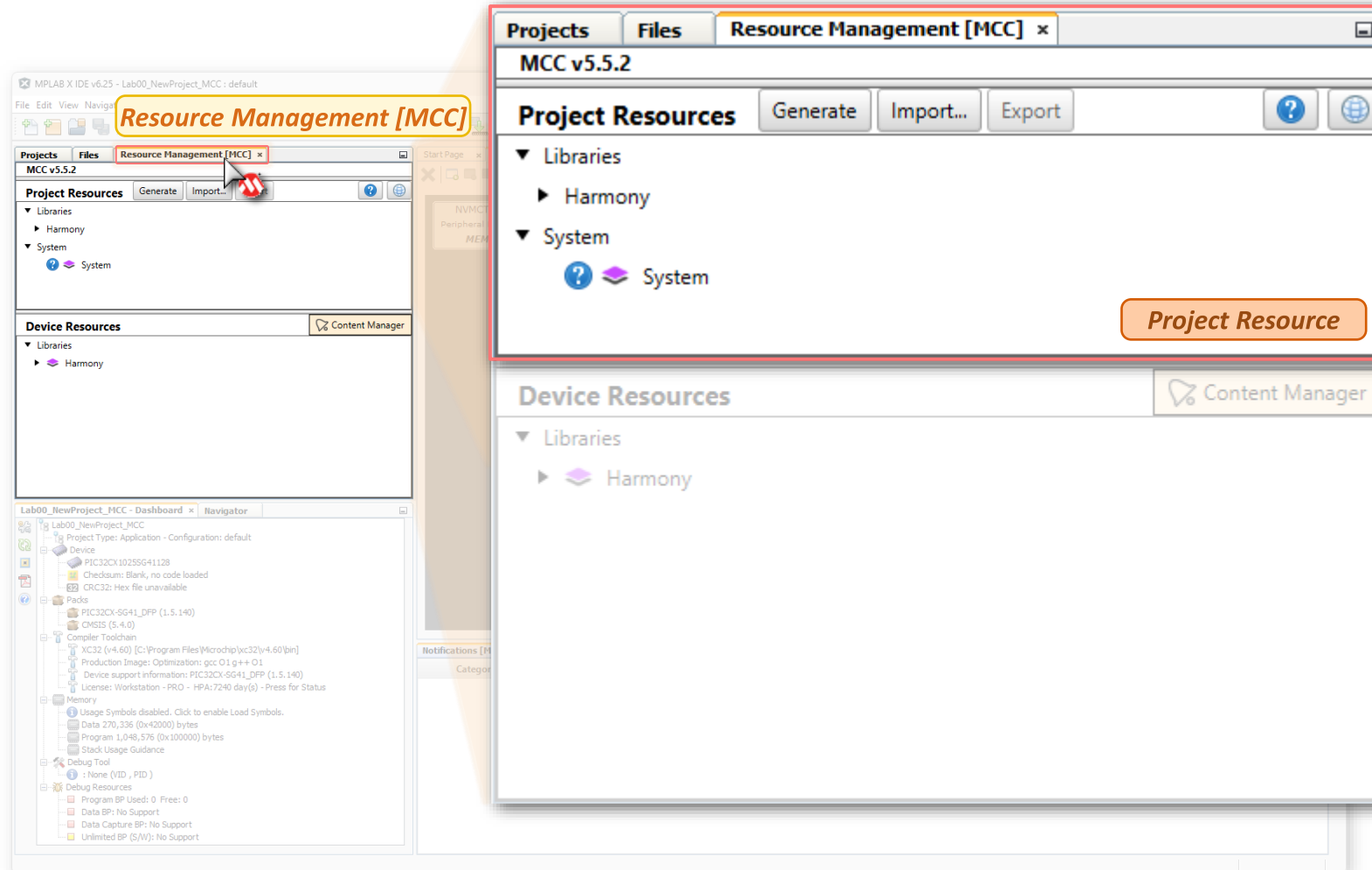
MCC Harmony Interface

- Let's go back to MCC Harmony and view the interface.
- **Notice !** If you close MCC already , please click the  icon and reopen MCC.



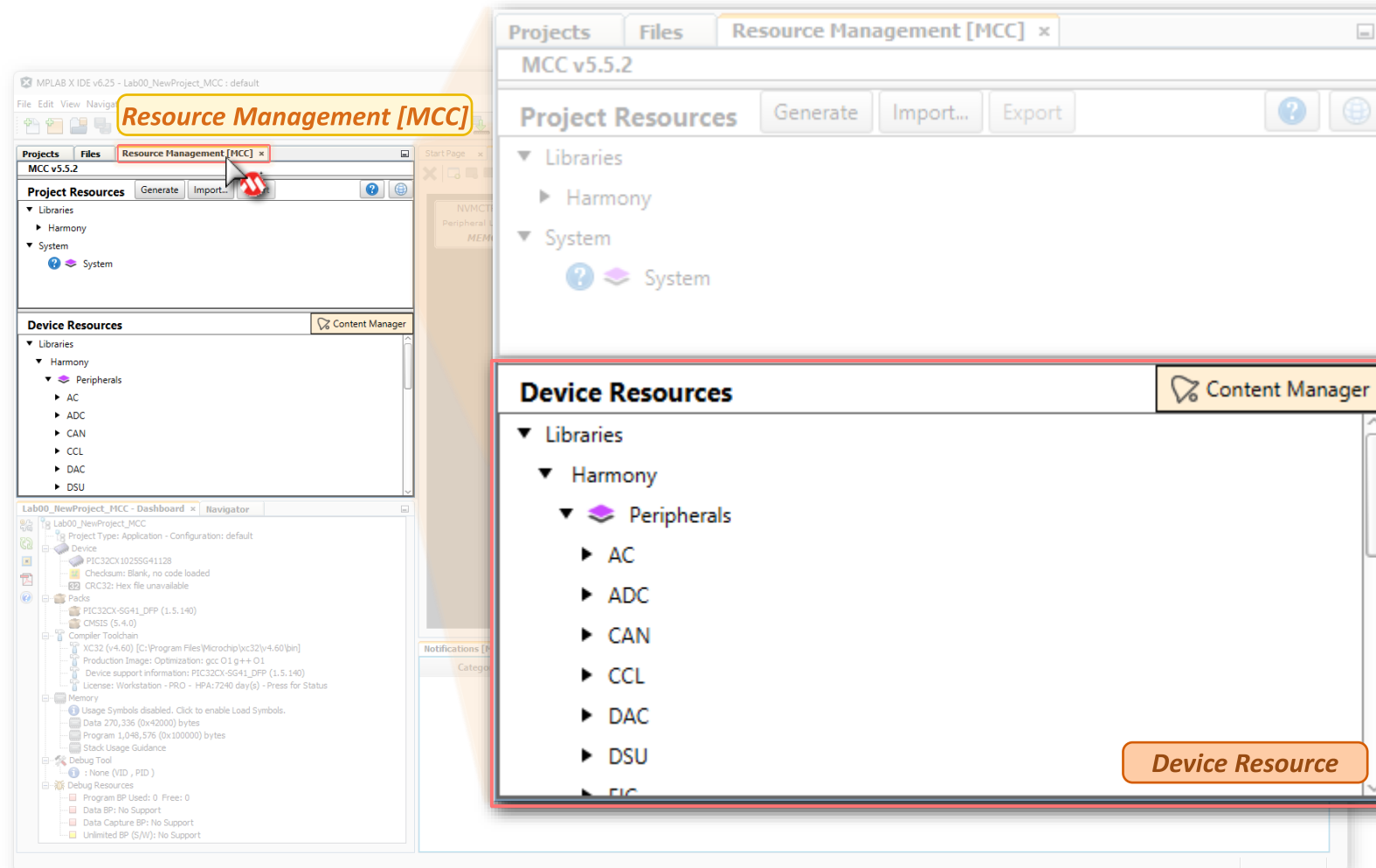
MCC Harmony Interface

- Project Resource, the modules **already added** in current project.



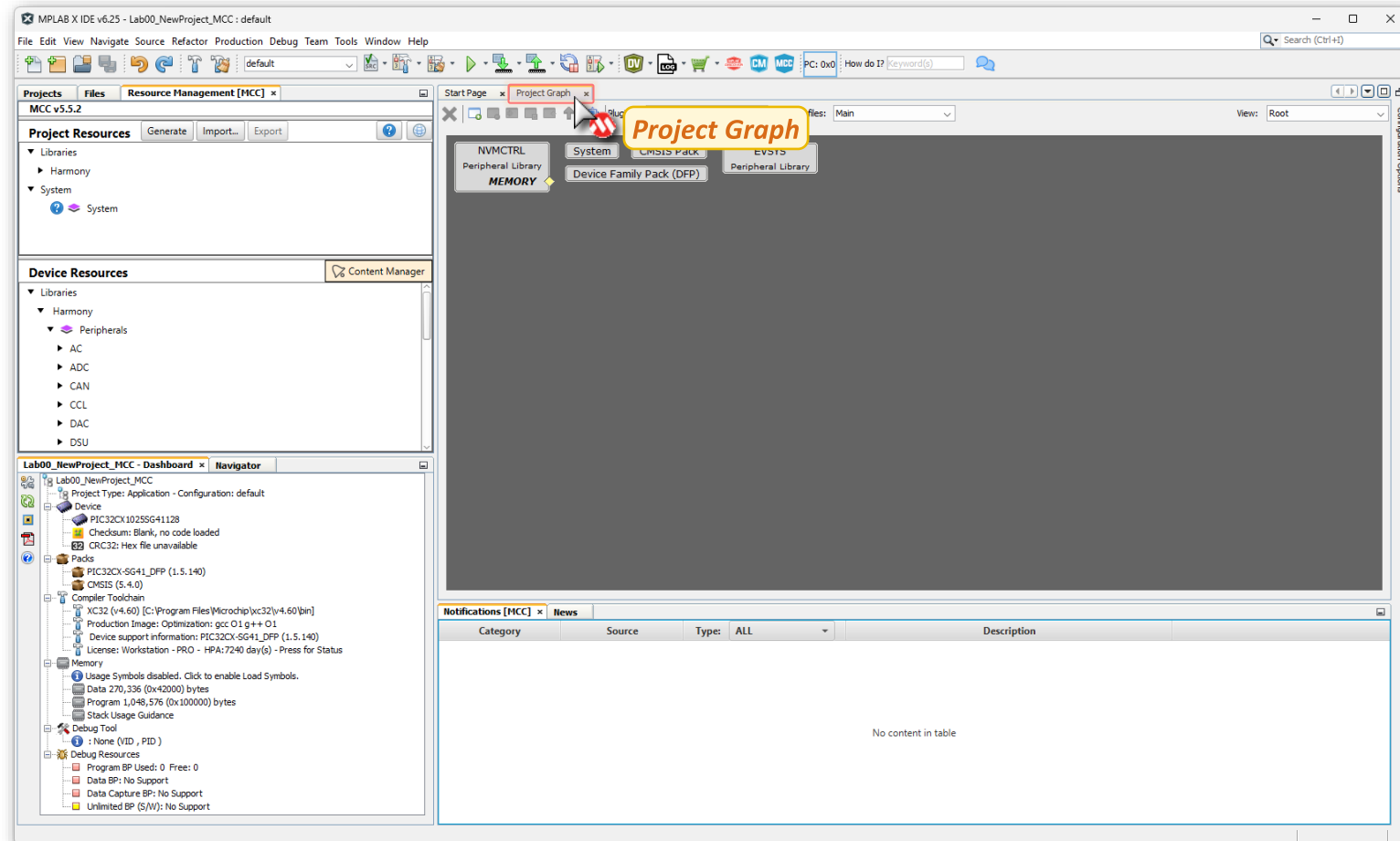
MCC Harmony Interface

- **Device Resource**, the modules not in current project but **available to be added**.



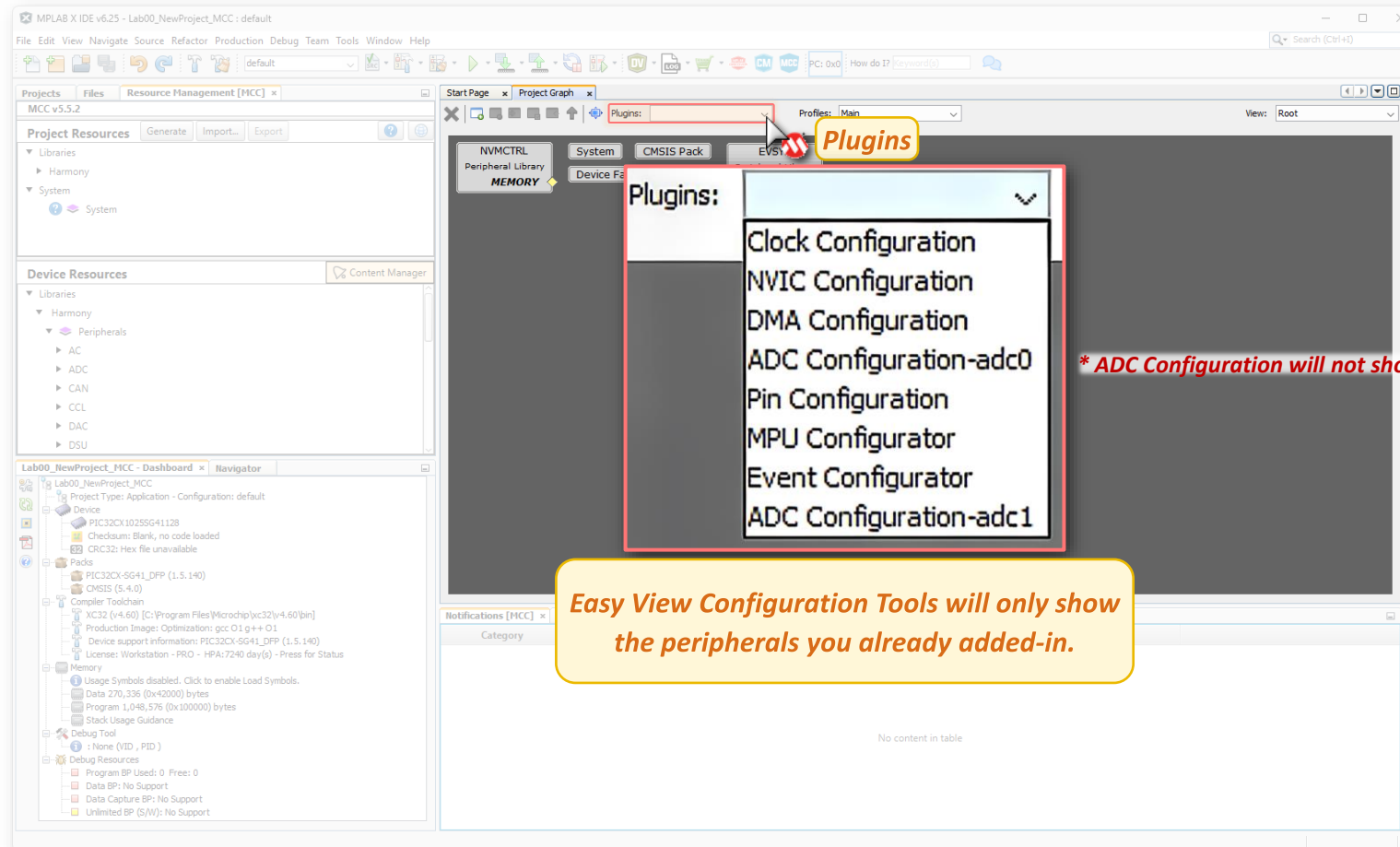
MCC Harmony Interface

- Project Graph, the connection diagram between modules.



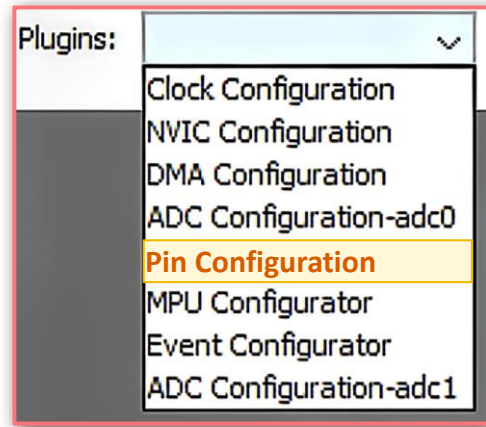
MCC Harmony Interface

- Project Graph → **Plugins**, the Easy View Configuration of modules.



MCC Harmony Interface

- Project Graph → **Plugins : Pin Configuration**



MPLAB X IDE v6.25 - Lab00_NewProject_MCC : default

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

default

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

Projects Files Resource Management [MCC] x

MCC v5.5.2

Project Resources Generate Import... Export

Libraries

- Harmony
- System

Device Resources Content Manager

Libraries

- Harmony
- Peripherals
- AC
- ADC
- CAN
- CCL
- DAC
- DSU

Lab00_NewProject_MCC - Dashboard x Navigator

Lab00_NewProject_MCC

Project Type: Application - Configuration: default

Device

- PIC32CX1025SG41128
- Checksum: Blank, no code loaded
- CRC32: Hex file unavailable

Packs

- PIC32CX-SG41_DFP (1.5.140)
- CHSIS (5.4.0)

Compiler Toolchain

- XC32 (v4.60) [C:\Program Files\Microchip\xc32\v4.60\bin]
- Production Image: Optimization: gcc O1 g++ O1
- Device support information: PIC32CX-SG41_DFP (1.5.140)
- License: Workstation - PRO - HPA:7240 day(s) - Press for Status

Memory

- Usage Symbols disabled. Click to enable Load Symbols.
- Data 270,336 (0x42000) bytes
- Program 1,048,576 (0x100000) bytes
- Stack Usage Guidance

Debug Tool

- None (VID, PID)

Debug Resources

- Program BP Used: 0 Free: 0
- Data BP: No Support
- Data Capture BP: No Support
- Unlimited BP (S/W): No Support

Notifications [MCC] News

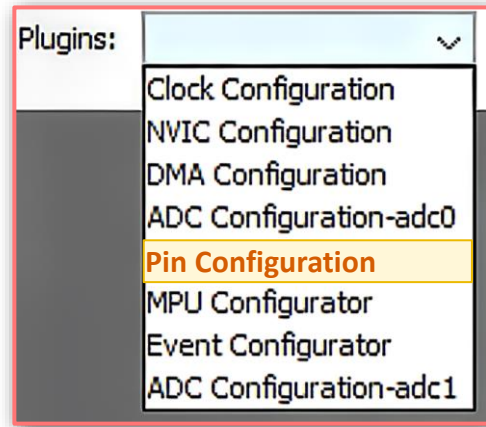
Start Page x Project Graph x Pin Diagram x Pin Table x Pin Settings x

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
1	PA00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
2	PA01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
3	PC00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
4	PC01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
5	NC			Digital	High Impedance	Low	☐	☐	NORMAL
6	AVDD			Digital	High Impedance	Low	☐	☐	NORMAL
7	PC02		Available	Digital	High Impedance	Low	☐	☐	NORMAL
8	PC03		Available	Digital	High Impedance	Low	☐	☐	NORMAL
9	PA02		Available	Digital	High Impedance	Low	☐	☐	NORMAL
10	PA03		Available	Digital	High Impedance	Low	☐	☐	NORMAL
11	PB04		Available	Digital	High Impedance	Low	☐	☐	NORMAL
12	PB05		Available	Digital	High Impedance	Low	☐	☐	NORMAL
13	PD00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
14	AVSS			Digital	High Impedance	Low	☐	☐	NORMAL
15	AVDD			Digital	High Impedance	Low	☐	☐	NORMAL
16	PD01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
17	PB06		Available	Digital	High Impedance	Low	☐	☐	NORMAL
18	PB07		Available	Digital	High Impedance	Low	☐	☐	NORMAL
19	PB08		Available	Digital	High Impedance	Low	☐	☐	NORMAL
20	PB09		Available	Digital	High Impedance	Low	☐	☐	NORMAL
21	PA04		Available	Digital	High Impedance	Low	☐	☐	NORMAL
22	PA05		Available	Digital	High Impedance	Low	☐	☐	NORMAL
23	PA06		Available	Digital	High Impedance	Low	☐	☐	NORMAL
24	PA07		Available	Digital	High Impedance	Low	☐	☐	NORMAL
25	AVSS			Digital	High Impedance	Low	☐	☐	NORMAL
26	AVDD			Digital	High Impedance	Low	☐	☐	NORMAL
27	PC04		Available	Digital	High Impedance	Low	☐	☐	NORMAL
28	PC05		Available	Digital	High Impedance	Low	☐	☐	NORMAL
29	PC06		Available	Digital	High Impedance	Low	☐	☐	NORMAL
30	PC07		Available	Digital	High Impedance	Low	☐	☐	NORMAL
31	VSS			Digital	High Impedance	Low	☐	☐	NORMAL
32	VDD			Digital	High Impedance	Low	☐	☐	NORMAL
33	PA08		Available	Digital	High Impedance	Low	☐	☐	NORMAL
34	PA09		Available	Digital	High Impedance	Low	☐	☐	NORMAL
35	PA10		Available	Digital	High Impedance	Low	☐	☐	NORMAL
36	PA11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
37	VDD			Digital	High Impedance	Low	☐	☐	NORMAL

MCC Harmony Interface

- Project Graph → **Plugins : Pin Configuration** (Dock three windows to different locations)



Pin Settings

Pin Number	Pin ID	Custom Name	Function	Mode	Direction
1	PA00		Available	Digital	High Impedance
2	PA01		Available	Digital	High Impedance
3	PC00		Available	Digital	High Impedance
4	PC01		Available	Digital	High Impedance
5	NC				
6	AVDD			Digital	High Impedance
7	PC02		Available	Digital	High Impedance
8	PC03		Available	Digital	High Impedance
9	PA02		Available	Digital	High Impedance
10	PA03		Available	Digital	High Impedance
11	PB04		Available	Digital	High Impedance
12	PB05		Available	Digital	High Impedance
13	PD00		Available	Digital	High Impedance
14	AVSS			Digital	High Impedance
15	AVDD			Digital	High Impedance
16	PD01		Available	Digital	High Impedance
17	PB06		Available	Digital	High Impedance
18	PB07		Available	Digital	High Impedance
19	PB08		Available	Digital	High Impedance
20	PB09		Available	Digital	High Impedance
21	PA04		Available	Digital	High Impedance
22	PA05		Available	Digital	High Impedance
23	PA06		Available	Digital	High Impedance

Pin Diagram

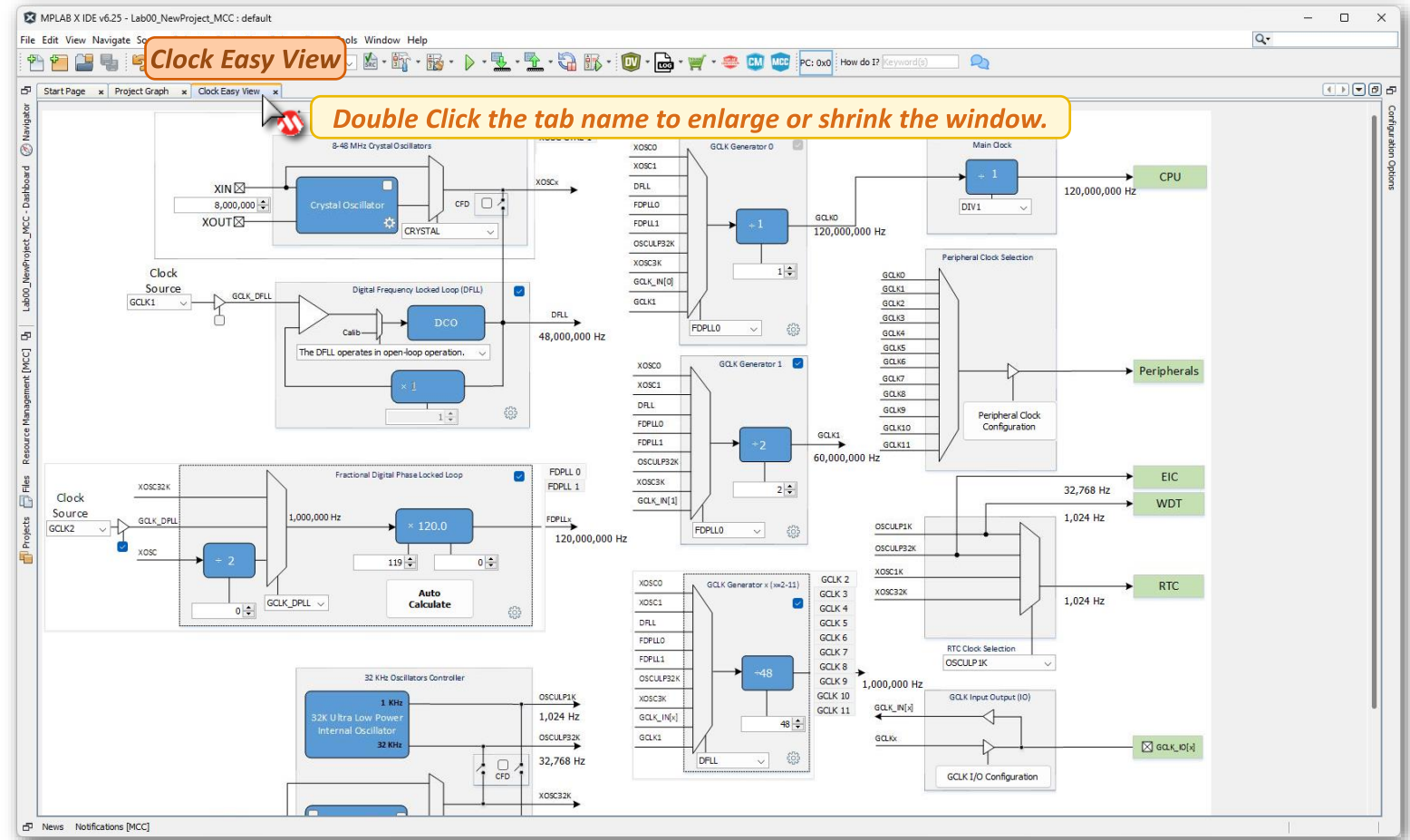
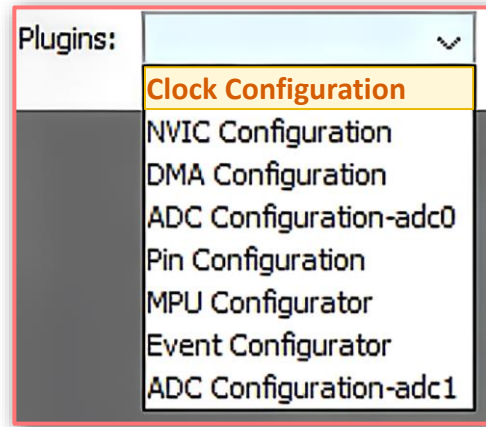
PIC32CX1025SG41128

Pin Table

Module	Function	PA00	PA01	PC00	PC01	NC	AVDD	PC02	PC03	PA02	PA03	PB04	PB05	PD00	AVSS	PD01	PB06	PB07	PB08	PB09	PA04	PA05	PA06	PA07	AVSS	PC04	PC05	PC06	PC07	VSS	VDD
AC	AC_AIN0																														
	AC_AIN1																														
	AC_AIN2																														
	AC_AIN3																														
	AC_CMP0																														
	AC_CMP1																														

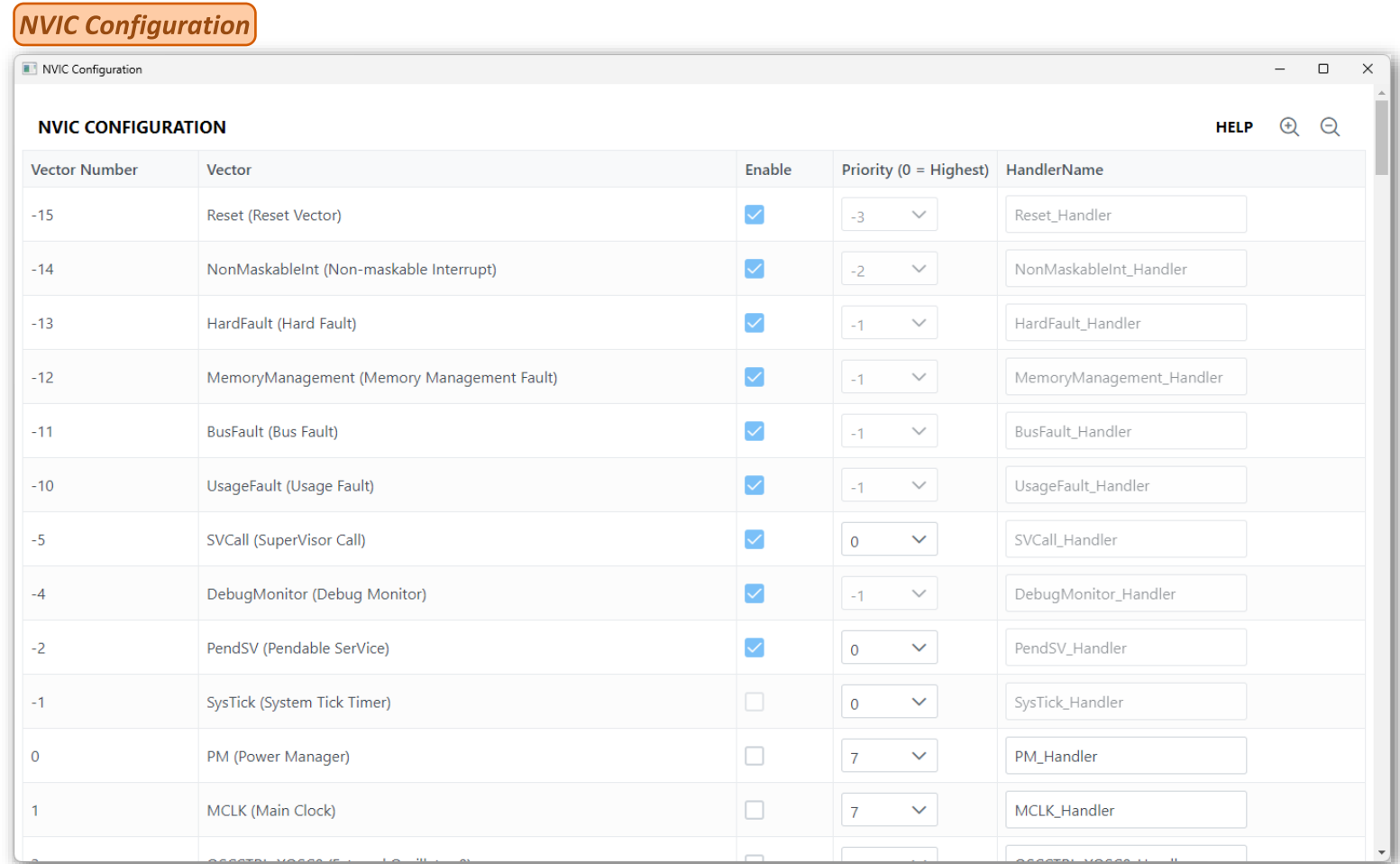
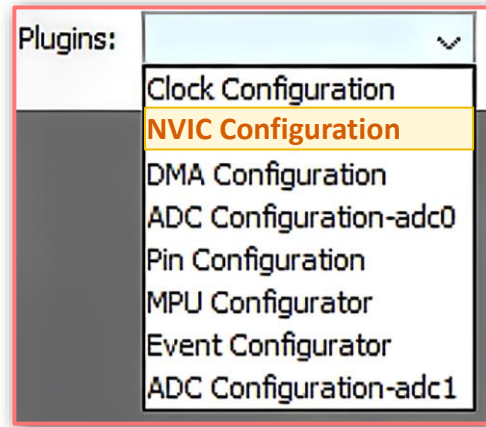
MCC Harmony Interface

- Project Graph → **Plugins : Clock Configuration**



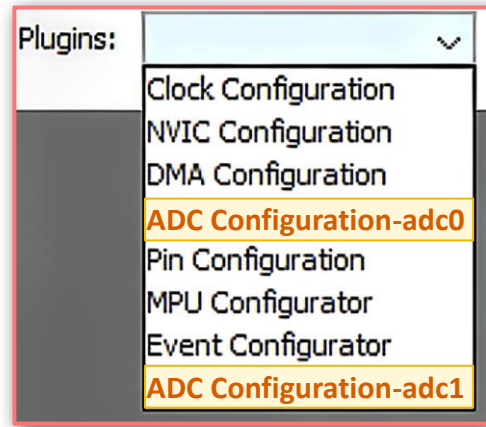
MCC Harmony Interface

- Project Graph → **Plugins : NVIC Configuration** (Open a new window after clicked)

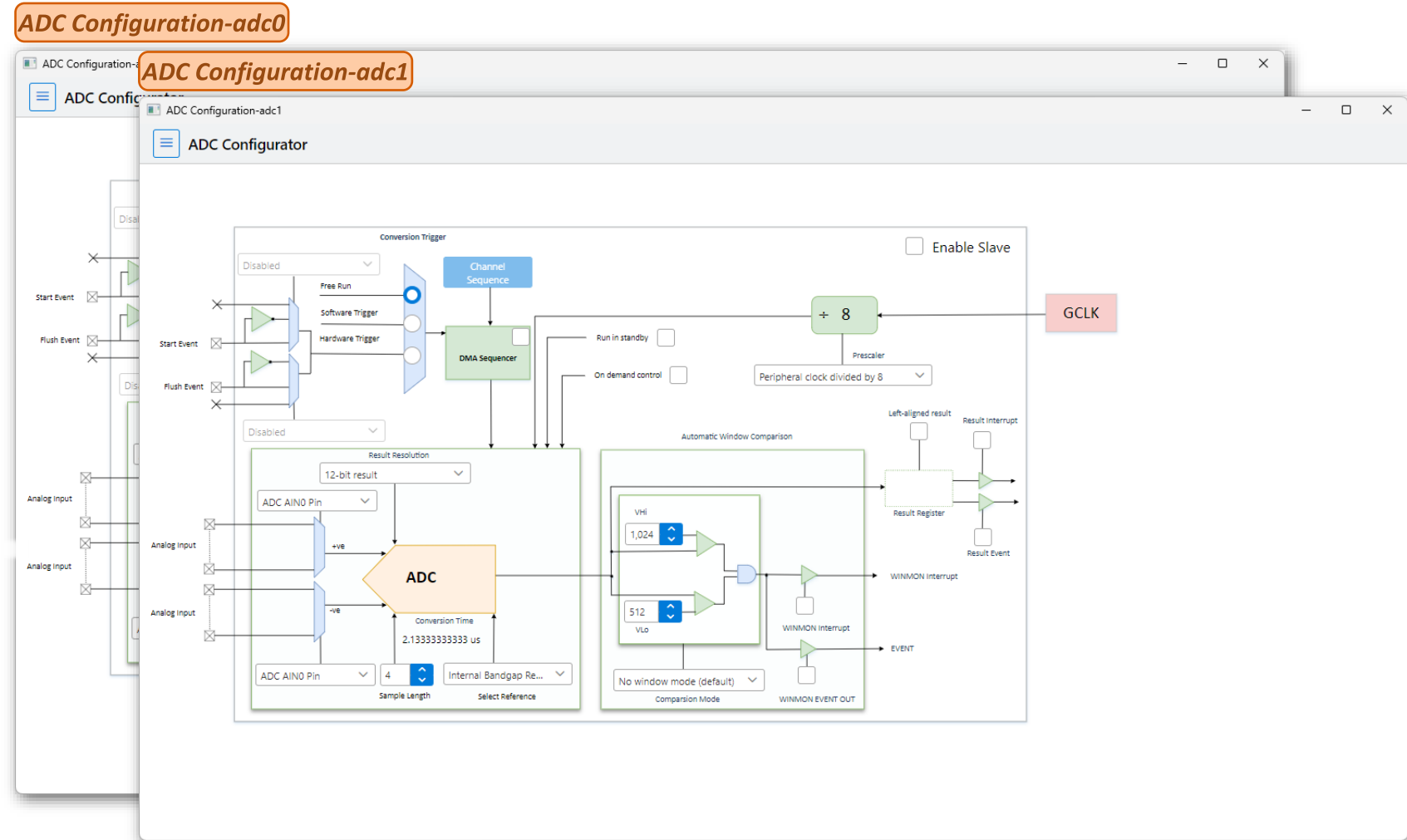


MCC Harmony Interface

- Project Graph → **Plugins : ADC Configuration** (Open a new window after clicked)

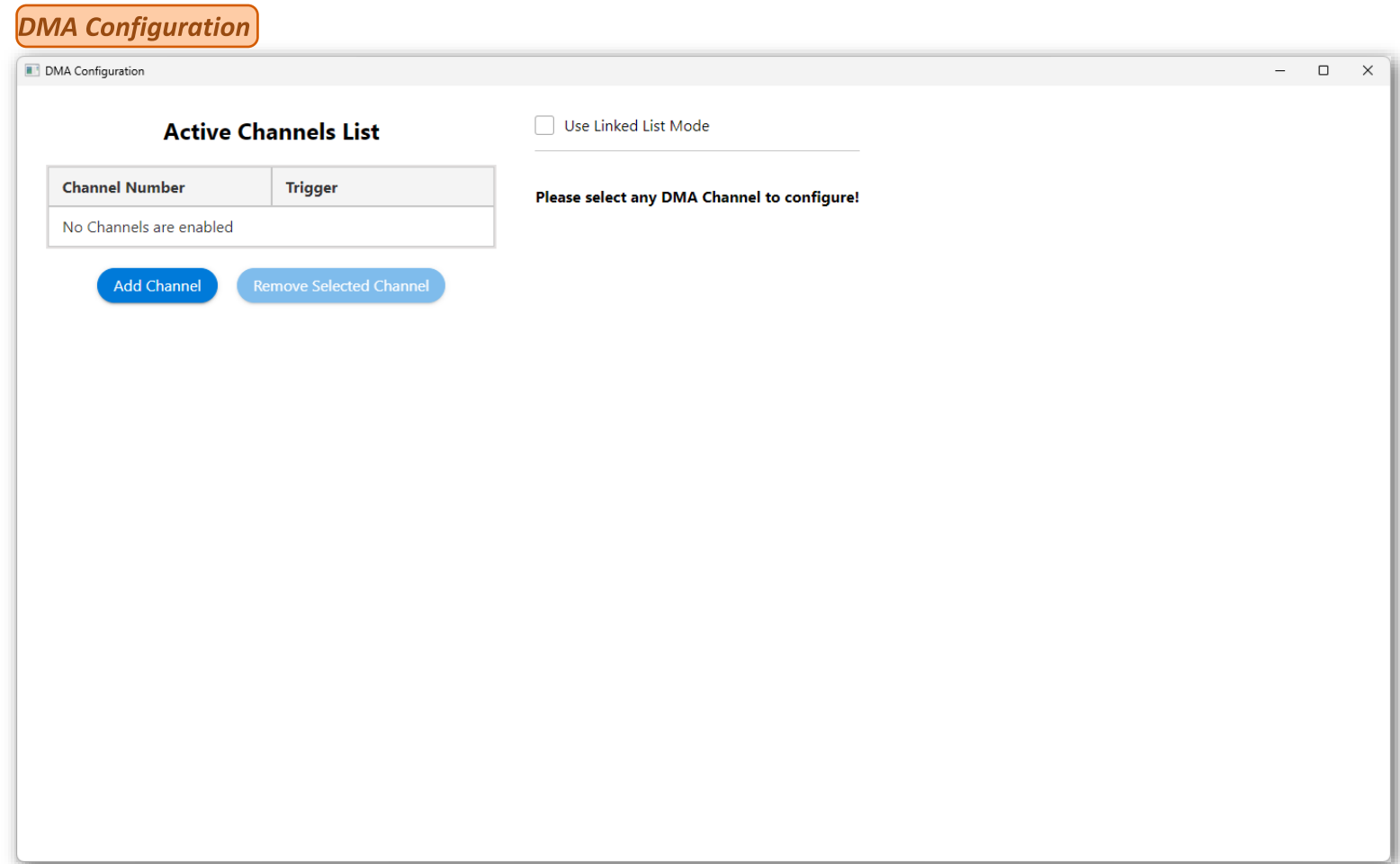
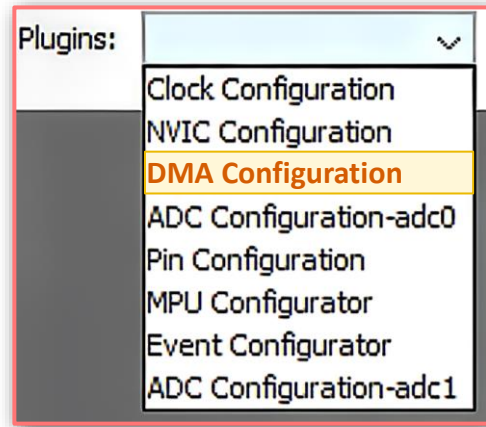


* ADC Configuration will not show in a new project



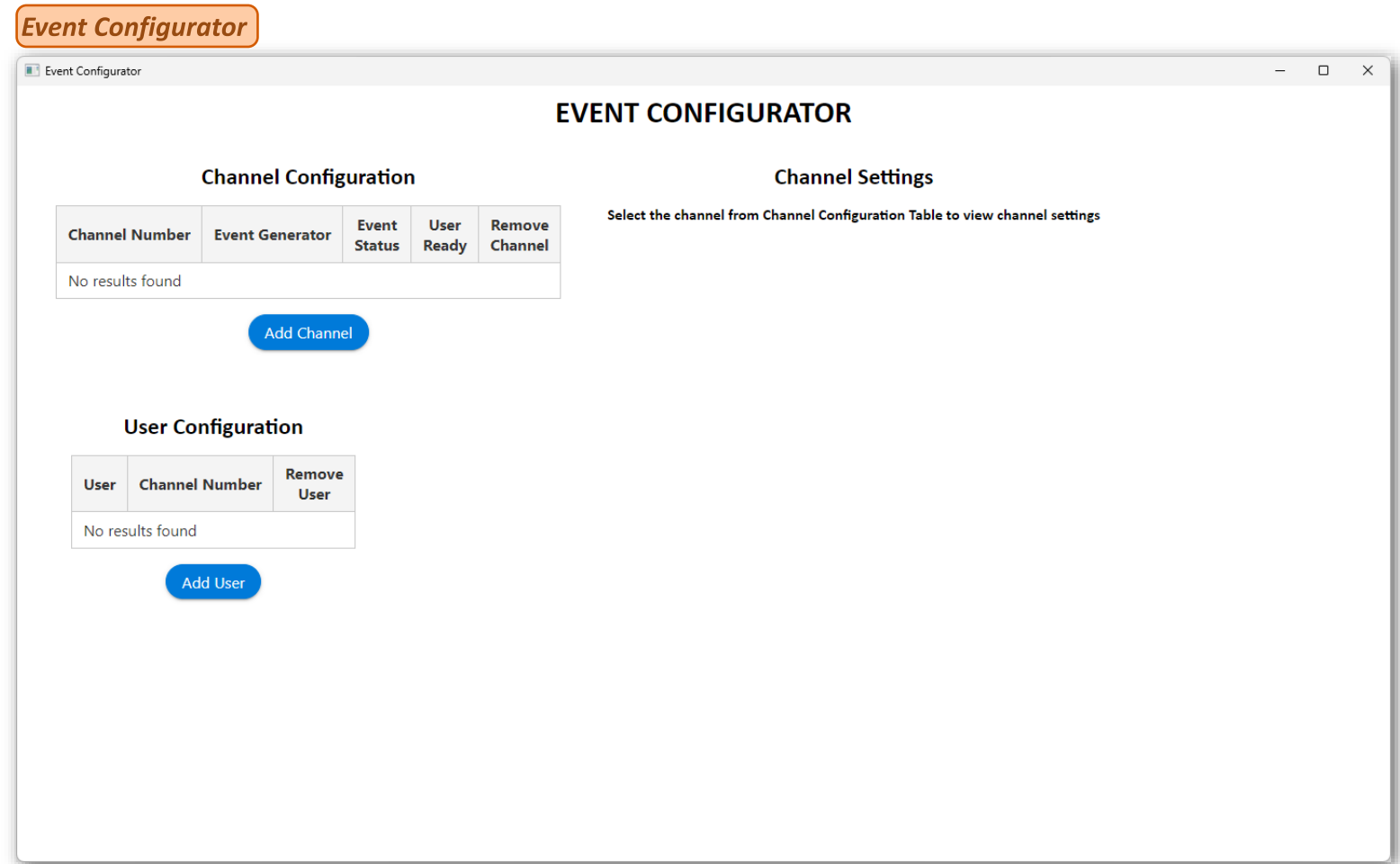
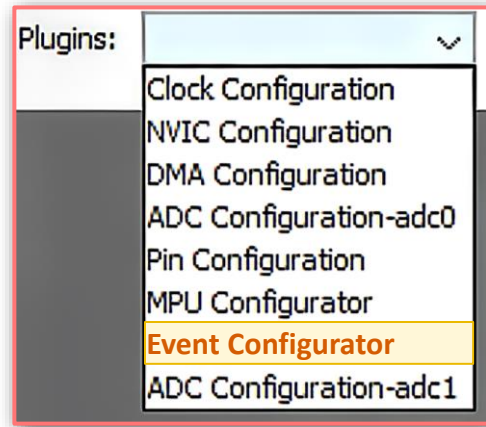
MCC Harmony Interface

- Project Graph → **Plugins : DMA Configuration** (Open a new window after clicked)



MCC Harmony Interface

- Project Graph → **Plugins : Event Configurator** (Open a new window after clicked)

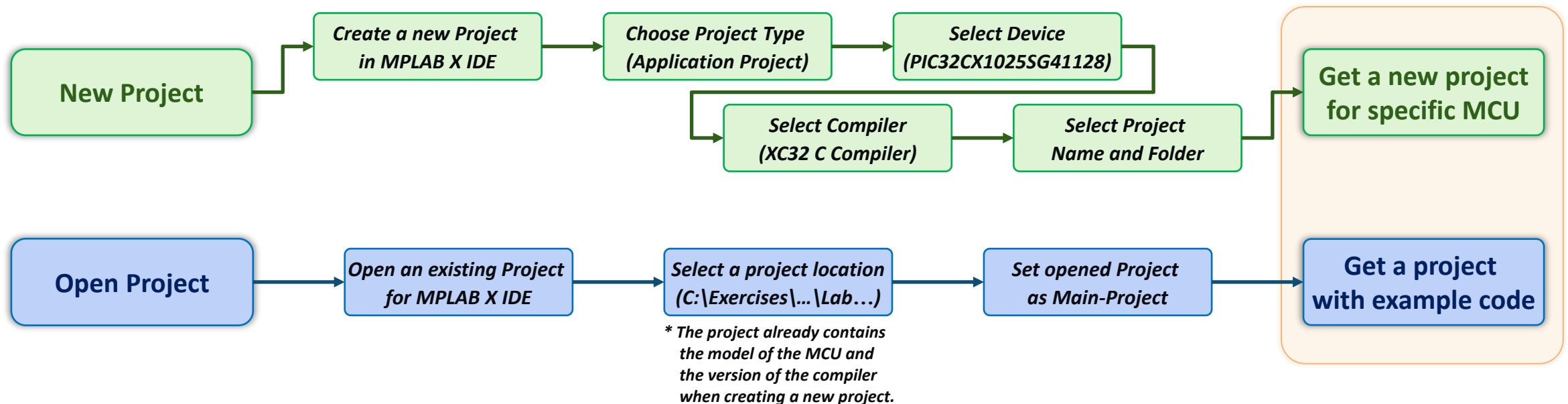


Appendix : Open MPLAB X IDE Projects

Open an existing project

MPLAB X IDE Develop Process

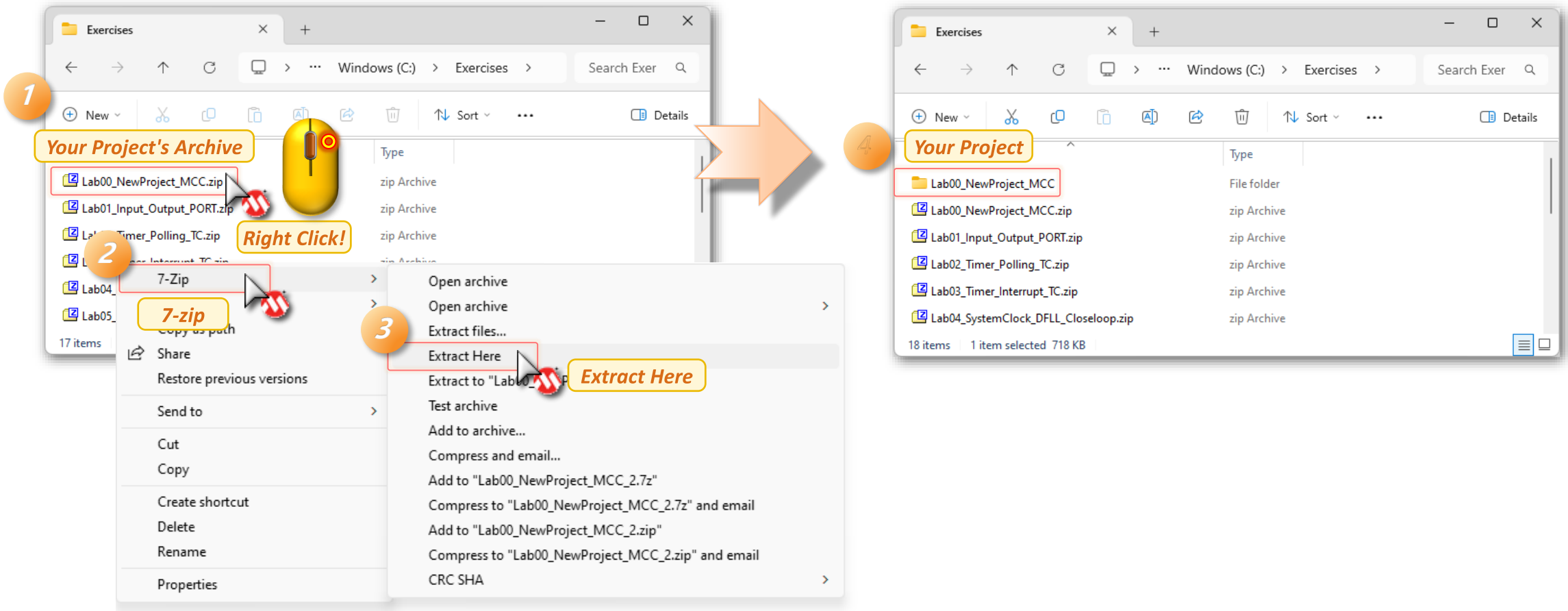
- When you want to develop a specific MCU, you need to create a new project or start with an existing example project.
- Both methods can obtain Projects that can be used for development. Creating a new project allows you to design your code and features from scratch; Open an existing example project will have some preset content that you can modify or add the features you needs.



Open an existing Project

Step 1

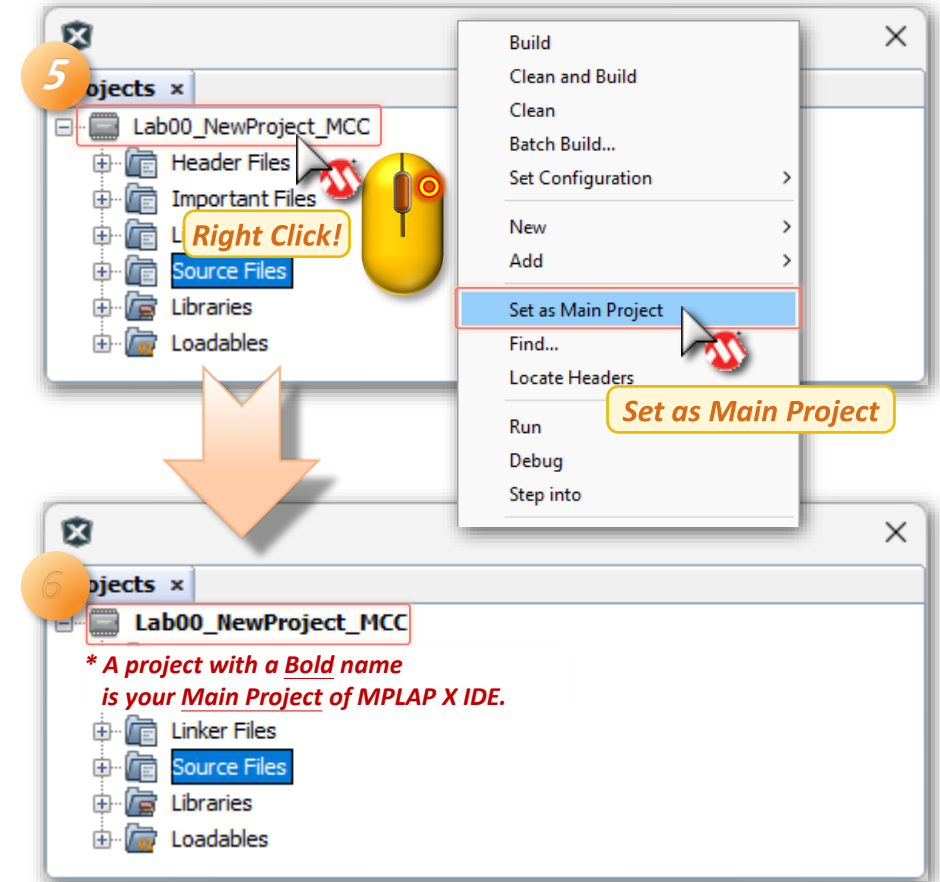
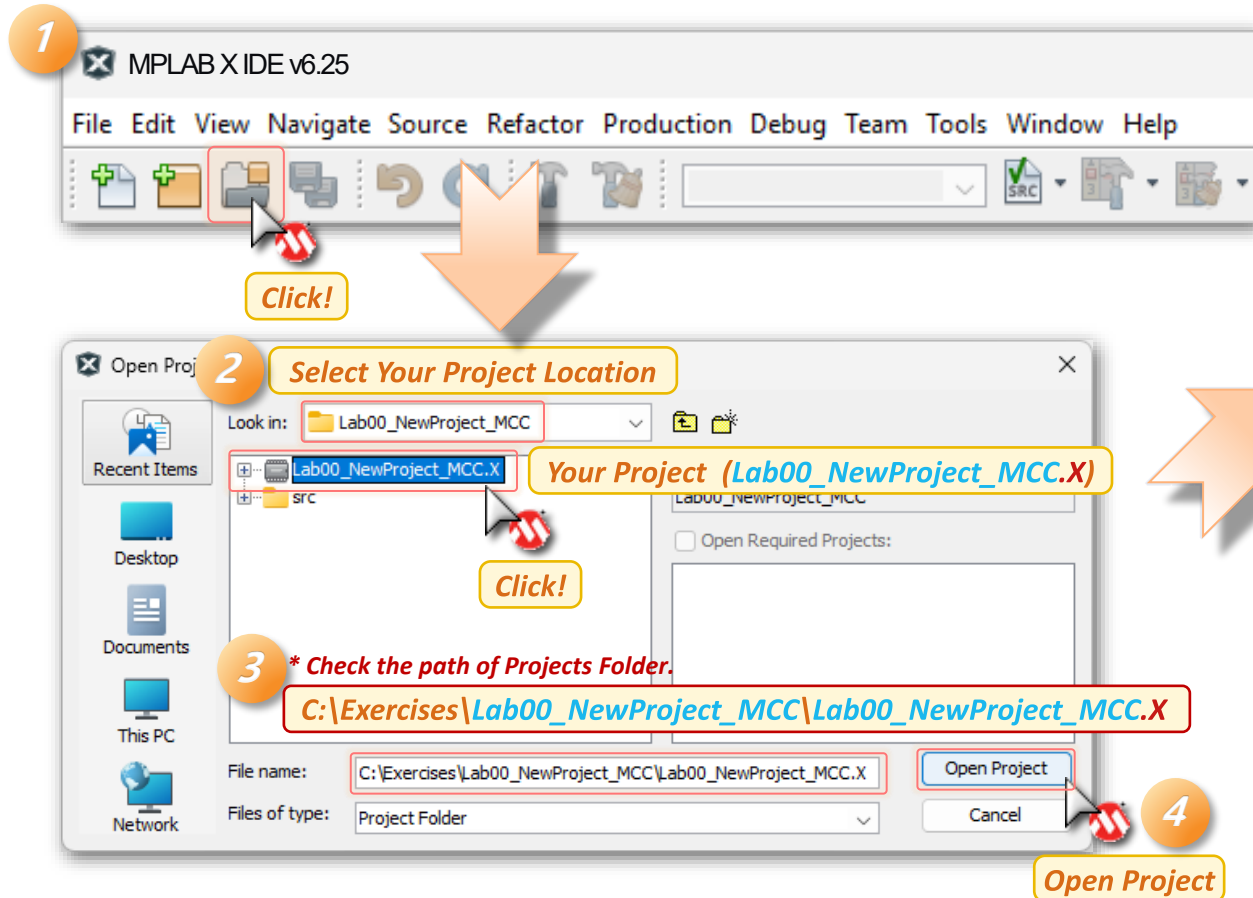
- **Extract the archive of the project with 7zip Tool**



🔧 Open an existing Project

Step 2

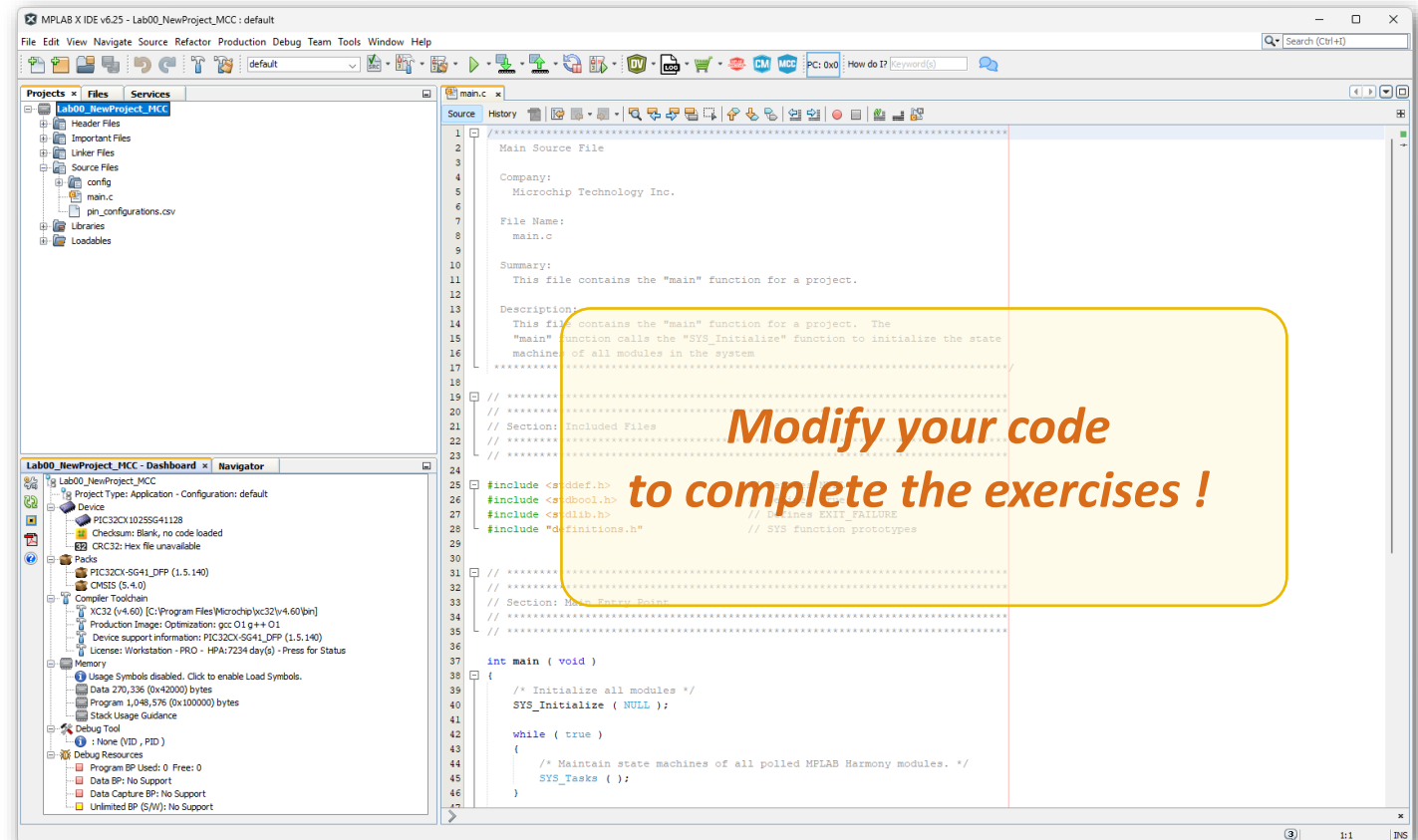
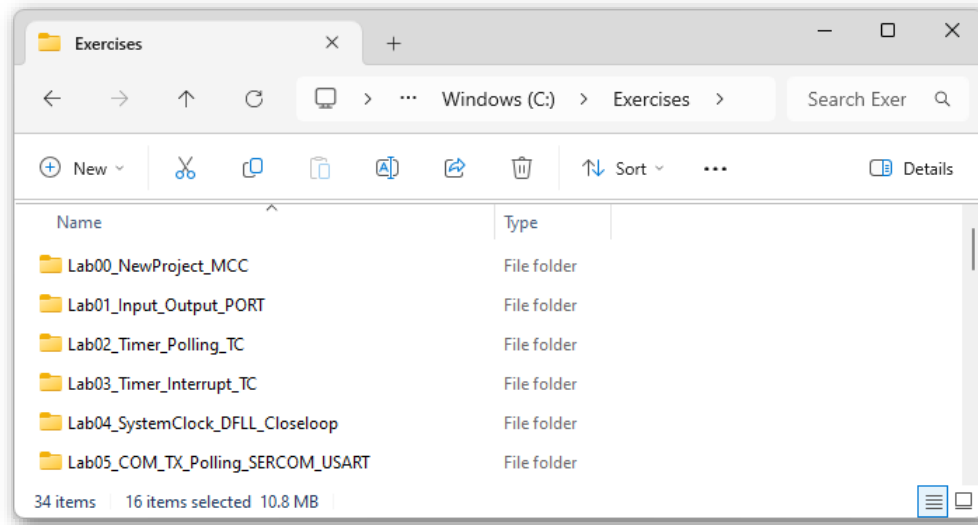
- Select to **File > Open Project** or **Click icon**



Open an existing Project

Result

- Now you can start adding or modifying the code required by next Labs. You can choose to open the previous project to complete the exercises in this section, or observe the Labs completed results.



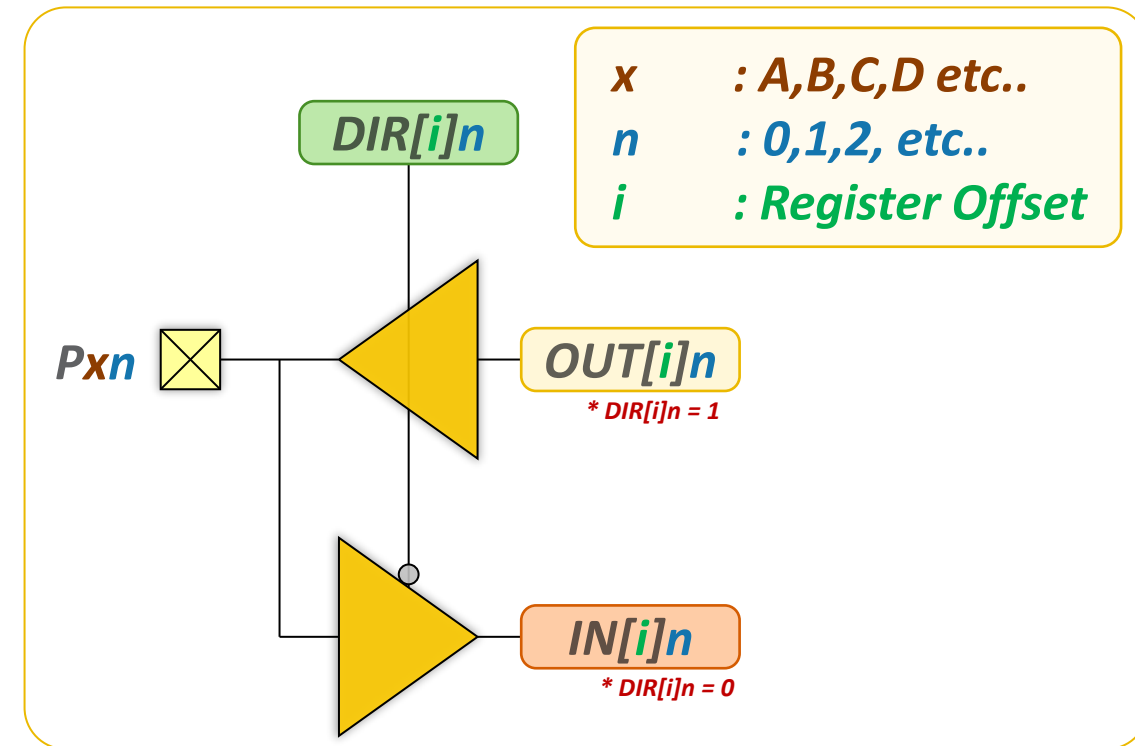
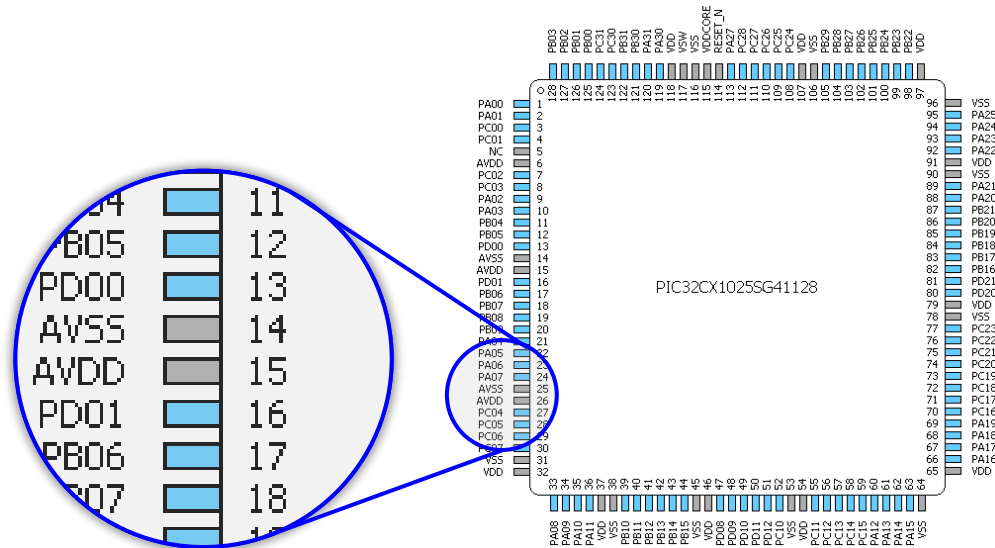
I/O - PORT

PIC32CX-SG I/O Pin Controller (PORT)

- The I/O Pin Controller (PORT) controls the I/O pins of the device. The I/O pins are organized in a series of groups, collectively referred to as a PORT group.
- PIC32CX-SG Family has up to **four PORT group (A~D)**, each PORT group can have **up to 32 pins** that can be configured and controlled individually or as a group. The number of PORT groups on a device may depend on the package or number of pins.
- All I/O pins have true **read-modify-write functionality** when used for general purpose I/O. The direction or output value of pins may be changed explicitly without unintentionally changing the state of any other pins in the same port group by a single.

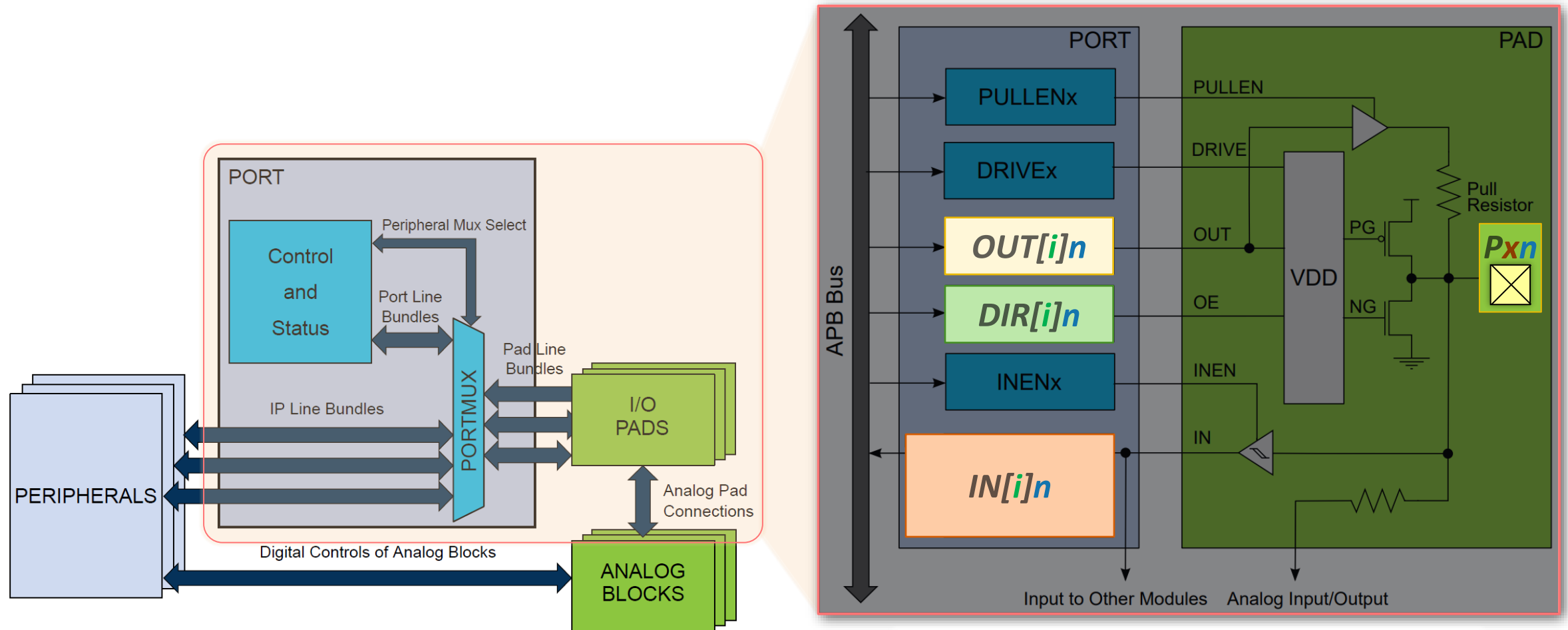
PORT Block Diagram

- The block diagram just a concept. For actual and detailed block diagram, please refer to datasheet.
- Each I/O pin P_{xn} can be controlled by the registers in PORT.
 - **DIR:** set the data direction for I/O pins. 0 for Input, 1 for Output.
 - **OUT:** set output drive value for Output pins.
 - **IN:** reaction logical level from Input pins.



PORT Block Diagram

- PIC32CX-SG PORT Block Diagram



* **PORT** contains powerful functional, the course only focus on basic controls.

* For more information, please refer to the datasheet.

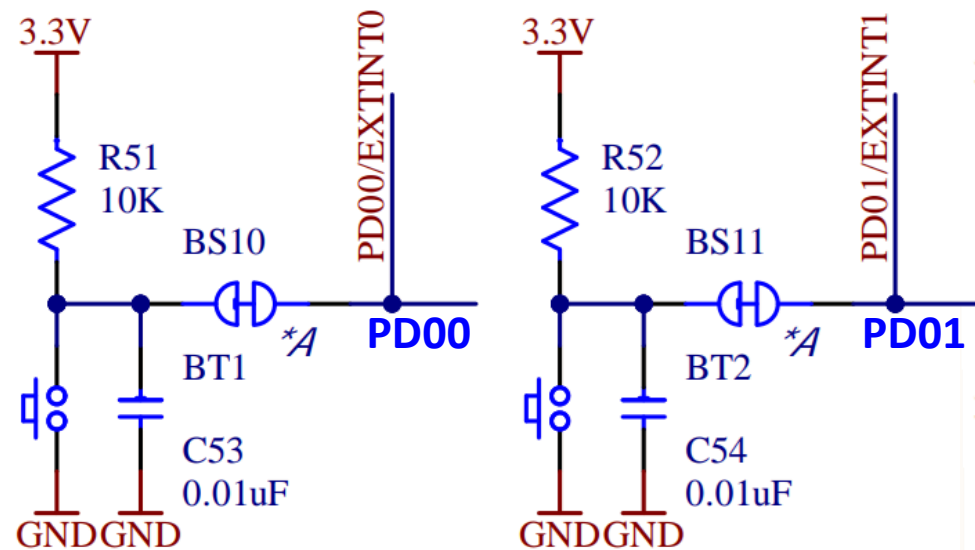
Lab1 PORT Input and Output

Lab1 PORT Input and Output

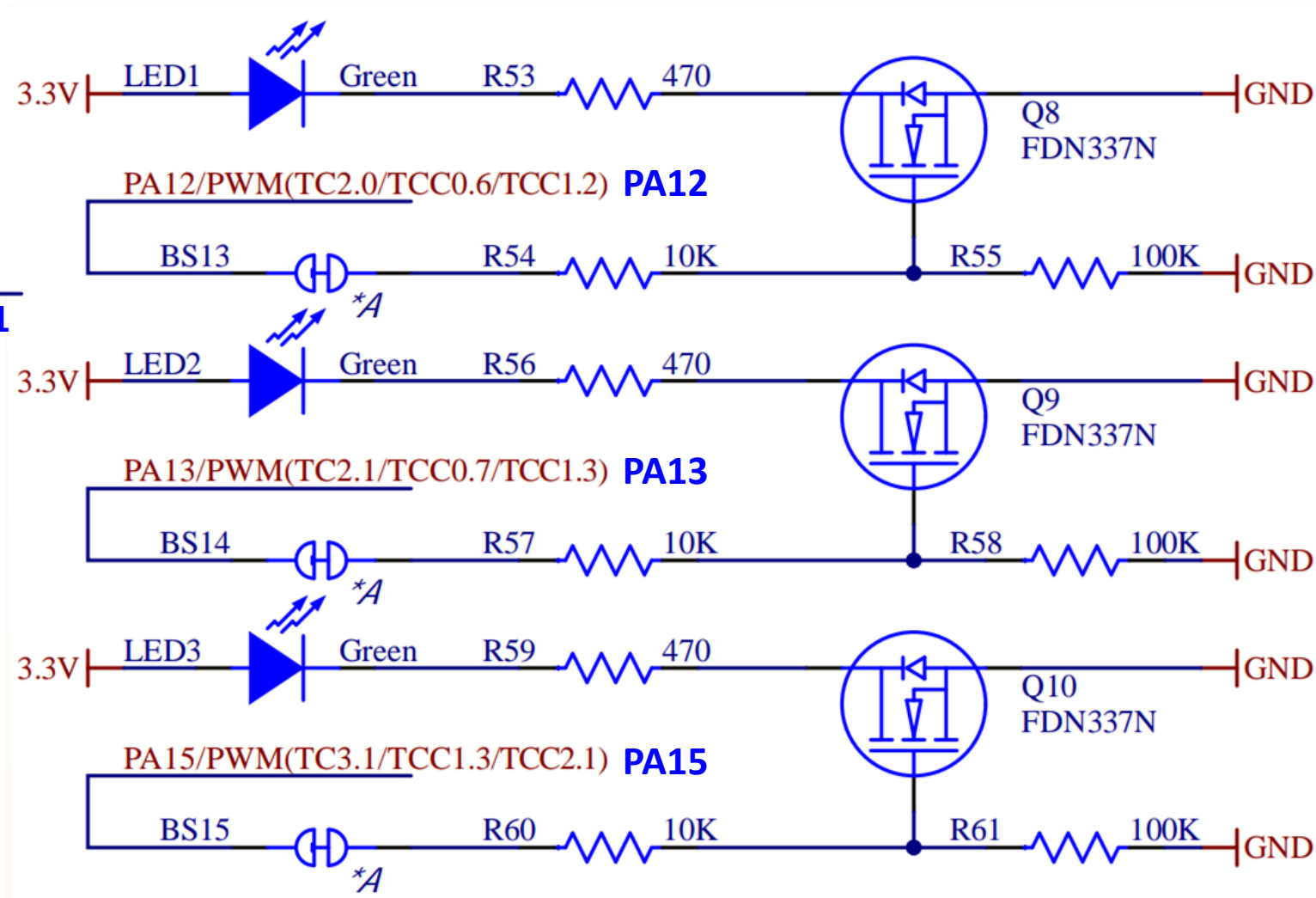
- Try to control **LED1(PA12)** and **LED2(PA13)** continues toggle.
(period around 500ms ~ 1S).
- Try to add **BT1(PD00)** and **BT2(PD01)**,
and get BT1 status then use to control **LED3(PA15)**.
Pressed -> LED3 ON
Released -> LED3 OFF

Let's go !

Check Schematic

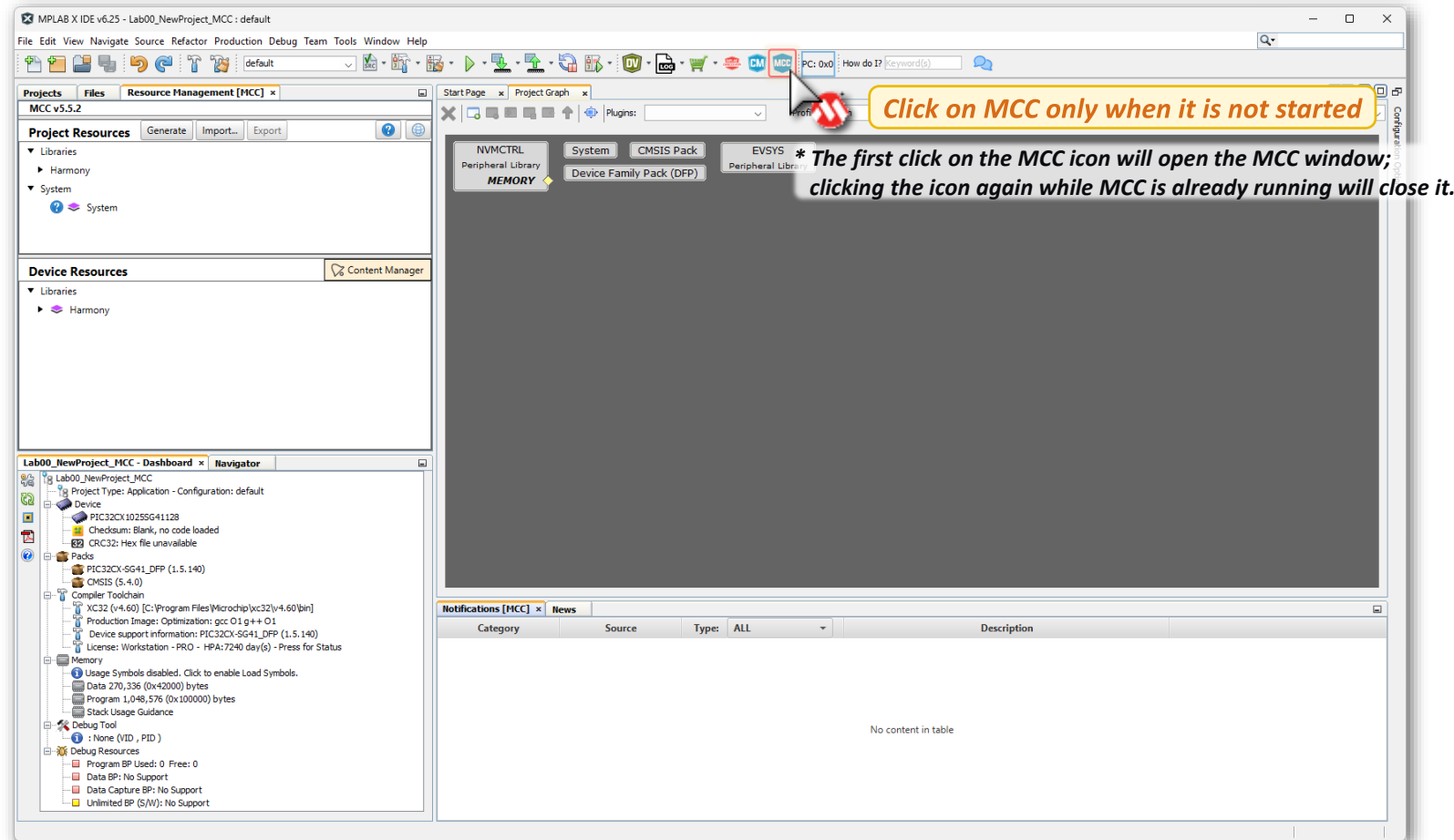


Schematic	Pin	Direction
BT1	PD00	Input
BT2	PD01	Input
LED1	PA12	Output
LED2	PA13	Output
LED3	PA15	Output



MCC Harmony Interface

- Let's go back to MCC Harmony and view the interface.
- **Notice !** If you close MCC already , please click the  icon and reopen MCC.

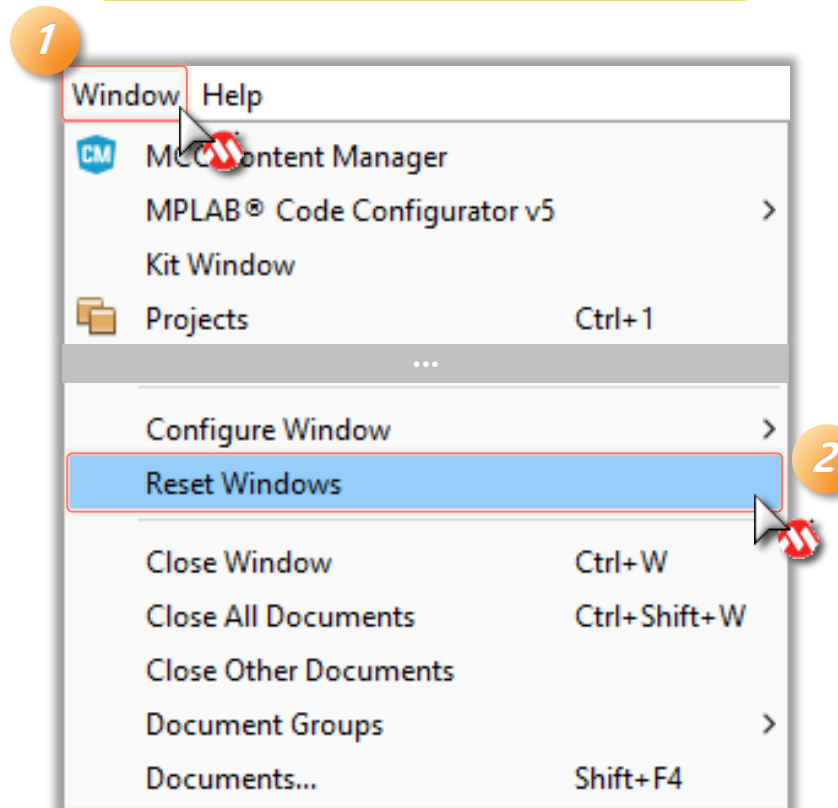


Reset the MPLAB X IDE and MCC Windows

- If you close the feature windows or can't find it while testing or using MPLAB X IDE and MCC, you can follow the steps below to reset your windows.

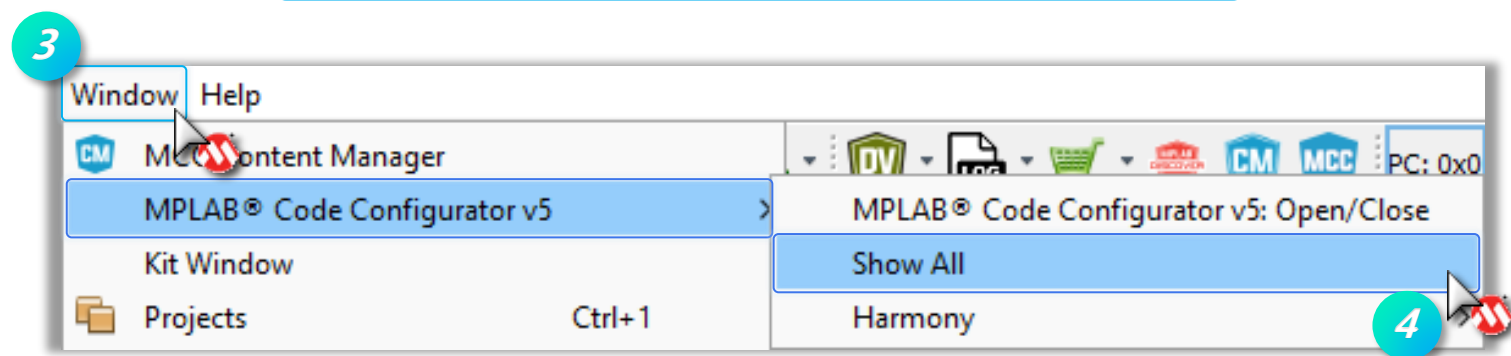
Reset MPLAB X IDE Windows

Windows > Reset Windows



Reset MCC Windows

Windows > MPLAB® Code Configurator v5 > Show All



⚡ Lab1 PORT Input and Output

Step 1

- Open Project Graph → **Plugins : Pin Configuration**

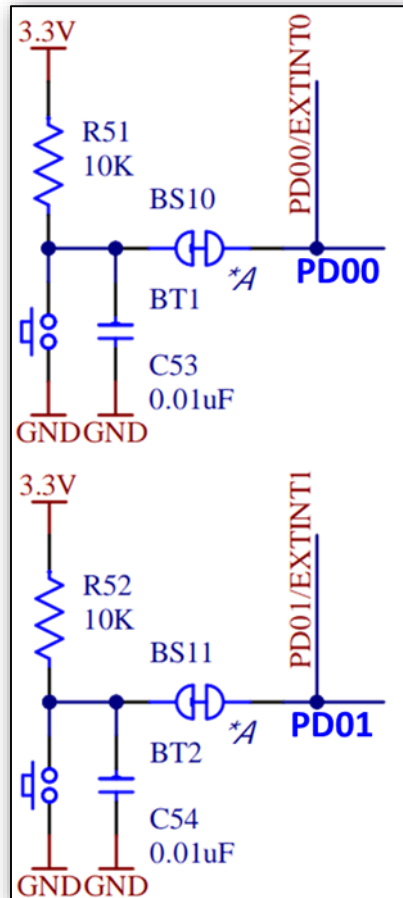
The screenshot displays the MPLAB X IDE interface. The 'Project Graph' window is open, showing a tree view of the project structure. A red box highlights the 'Plugins' button, with a red arrow pointing to it and the text 'Click!'. The 'Plugins' menu is open, showing a list of configuration options. A red box highlights the 'Pin Configuration' option, with a red arrow pointing to it and the text 'Click!'. The 'Pin Table' window is also open, showing a table of pin configurations. The table has columns for Pin Number, Pin ID, Custom Name, Function, Mode, Direction, Latch, Pull Up, Pull Down, and Drive Strength. The table lists pins 1 through 37, with their respective IDs and configurations. The 'Pin Diagram' and 'Pin Settings' windows are also visible, with red arrows pointing to them and the text 'Click!'.

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
1	PA00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
2	PA01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
3	PC00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
4	PC01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
5	NC			Digital	High Impedance	Low	☐	☐	NORMAL
6	AVDD			Digital	High Impedance	Low	☐	☐	NORMAL
7	PC02		Available	Digital	High Impedance	Low	☐	☐	NORMAL
8	PC03		Available	Digital	High Impedance	Low	☐	☐	NORMAL
9	PA02		Available	Digital	High Impedance	Low	☐	☐	NORMAL
10	PA03		Available	Digital	High Impedance	Low	☐	☐	NORMAL
11	PB04		Available	Digital	High Impedance	Low	☐	☐	NORMAL
12	PB05		Available	Digital	High Impedance	Low	☐	☐	NORMAL
13	PD00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
14	AVSS			Digital	High Impedance	Low	☐	☐	NORMAL
15	AVDD			Digital	High Impedance	Low	☐	☐	NORMAL
16	PD01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
17	PB06		Available	Digital	High Impedance	Low	☐	☐	NORMAL
18	PB07		Available	Digital	High Impedance	Low	☐	☐	NORMAL
19	PB08		Available	Digital	High Impedance	Low	☐	☐	NORMAL
20	PB09		Available	Digital	High Impedance	Low	☐	☐	NORMAL
21	PA04		Available	Digital	High Impedance	Low	☐	☐	NORMAL
22	PA05		Available	Digital	High Impedance	Low	☐	☐	NORMAL
23	PA06		Available	Digital	High Impedance	Low	☐	☐	NORMAL
24	PA07		Available	Digital	High Impedance	Low	☐	☐	NORMAL
25	AVSS			Digital	High Impedance	Low	☐	☐	NORMAL
26	AVDD			Digital	High Impedance	Low	☐	☐	NORMAL
27	PC04		Available	Digital	High Impedance	Low	☐	☐	NORMAL
28	PC05		Available	Digital	High Impedance	Low	☐	☐	NORMAL
29	PC06		Available	Digital	High Impedance	Low	☐	☐	NORMAL
30	PC07		Available	Digital	High Impedance	Low	☐	☐	NORMAL
31	VSS			Digital	High Impedance	Low	☐	☐	NORMAL
32	VDD			Digital	High Impedance	Low	☐	☐	NORMAL
33	PA08		Available	Digital	High Impedance	Low	☐	☐	NORMAL
34	PA09		Available	Digital	High Impedance	Low	☐	☐	NORMAL
35	PA10		Available	Digital	High Impedance	Low	☐	☐	NORMAL
36	PA11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
37	VDD			Digital	High Impedance	Low	☐	☐	NORMAL

⚙️ Lab1 PORT Input and Output

Step 2

- Enable GPIO as below to connect pins and BUTTONs by Pin Table.



Pin Table

Package: TQFP128

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
	GMAC_GTXER																						
GPIO	GPIO																						
I2S	I2S_FS0																						
	I2S_FS1																						
	I2S_MCK0																						
	I2S_MCK1																						
	I2S_SCK0																						
	I2S_SCK1																						

Schematic	Pin	Direction
BT1	PD00	Input
BT2	PD01	Input
LED1	PA12	Output
LED2	PA13	Output
LED3	PA15	Output

⚙️ Lab1 PORT Input and Output

Step 3

- Enable GPIO as below to connect pins and LEDs by Pin Table.

The image displays a circuit schematic and a Pin Table for a Microchip device, specifically a TQFP128 package.

Circuit Schematic: The schematic shows three LEDs (LED1, LED2, LED3) connected to the 3.3V supply. Each LED is connected to a green output pin (PA12, PA13, PA15) through a 470Ω resistor (R53, R56, R59). The output pins are also connected to a 100KΩ pull-up resistor (R55, R58, R61) to GND. The input pins are connected to a 10KΩ pull-down resistor (R54, R57, R60) to GND. The output pins are labeled PA12, PA13, and PA15, and the input pins are labeled BS13, BS14, and BS15. The output pins are also connected to a 100KΩ pull-up resistor (R55, R58, R61) to GND.

Pin Table: The Pin Table shows the configuration for the TQFP128 package. The columns represent the pin number (PD08 to PA18) and the function (Function, GMAC_GTXER, GPIO, I2S_FS1, I2S_MCK0, I2S_MCK1, I2S_SCK0, I2S_SCK1). The rows represent the pin number (PD08 to PA18). The GPIO row is highlighted in green, indicating that the GPIO pins are enabled. The output pins (PA12, PA13, PA15) are also highlighted in green.

Click! Click to Enable (Green)

Schematic	Pin	Direction
BT1	PD00	Input
BT2	PD01	Input
LED1	PA12	Output
LED2	PA13	Output
LED3	PA15	Output

✖ Lab1 PORT Input and Output

Step 4

- Modify the pin name and direction by Pin Setting.

Pin Settings

Start Page x Project Graph x Pin Diagram x Pin Table x **Pin Settings** x

Order: Pins Table View ☒ Easy View **Click!**

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength	
10	PA03		Available	Digital	High Impedance	Low	☐	☐	NORMAL	
11	PB04		Available	Digital	High Impedance	Low	☐	☐	NORMAL	
12	PB05		Available	Digital	High Impedance	Low	☐	☐	NORMAL	
PD00	13	PD00	BT1 BT1	GPIO	Digital	In	In	☐	☐	NORMAL
	14	AVSS		Digital	High Impedance	Low	☐	☐	NORMAL	
	15	AVDD		Digital	High Impedance	Low	☐	☐	NORMAL	
PD01	16	PD01	BT2 BT2	GPIO	Digital	In	In	☐	☐	NORMAL
	17	PB06		Available	Digital	High Impedance	Low	☐	☐	NORMAL
...										
	59	PC15		Available	Digital	High Impedance	Low	☐	☐	NORMAL
PA12	60	PA12	LED1 LED1	GPIO	Digital	Out	Out	☐	☐	NORMAL
PA13	61	PA13	LED2 LED2	GPIO	Digital	Out	Out	☐	☐	NORMAL
	62	PA14		Available	Digital	High Impedance	Low	☐	☐	NORMAL
PA15	63	PA15	LED3 LED3	GPIO	Digital	Out	Out	☐	☐	NORMAL
	64			Digital	High Impedance	Low	☐	☐	NORMAL	

Schematic	Pin	Direction
BT1	(13) PD00	Input
BT2	(16) PD01	Input
LED1	(60) PA12	Output
LED2	(61) PA13	Output
LED3	(63) PA15	Output

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

⚙️ Lab1 PORT Input and Output

Step 6

- Click on **Generate** button to generate your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Project Resources' tab. The 'Generate' button is highlighted with a yellow box and a red arrow. Below it, the 'System' resource is also highlighted. The 'Output' window is open, showing the 'MPLAB® Code Configurator' tab. The output text includes the following lines:

```
15:17:50.827 INFO: ...PIC32CX1025SG41128_DFP (p10 (pic32cx1025sg41128.n)) Success.  
15:17:50.827 INFO: HARMONY: [ MHC ] Executing generate hook  
15:17:50.843 INFO:   
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)  
15:17:50.844 INFO:   
15:17:50.844 INFO:   
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent  
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent  
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel  
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

A green box highlights the 'Generation complete (total time: 5396 milliseconds)' message, and an orange box highlights the 'Generation complete' text in the output.

I/O Pin Controller (PORT) Macros

MCC Harmony GPIO Macros

- Using MCC Harmony to configure PORT and generate the code will get some Macros of custom name, and automatically initialize.

```
...
/**** Macros for LED1 pin ****/
#define LED1_Set()          (PORT_REGS->GROUP[0].PORT_OUTSET = ((uint32_t)1U << 12U))
#define LED1_Clear()        (PORT_REGS->GROUP[0].PORT_OUTCLR = ((uint32_t)1U << 12U))
#define LED1_Toggle()        (PORT_REGS->GROUP[0].PORT_OUTTGL = ((uint32_t)1U << 12U))
#define LED1_OutputEnable()  (PORT_REGS->GROUP[0].PORT_DIRSET = ((uint32_t)1U << 12U))
#define LED1_InputEnable()   (PORT_REGS->GROUP[0].PORT_DIRCLR = ((uint32_t)1U << 12U))
#define LED1_Get()           (((PORT_REGS->GROUP[0].PORT_IN >> 12U) & 0x01U))
#define LED1_PIN              PORT_PIN_PA12
...
* Macros will be named according to your custom GPIO name.
```

plib_port.h

```
#include "plib_port.h"
...
void PORT_Initialize(void)
{
    /***** GROUP 0 Initialization *****/
    PORT_REGS->GROUP[0].PORT_DIR = 0x4000U;
    PORT_REGS->GROUP[0].PORT_PINCFG[12] = 0x00U;

    PORT_REGS->GROUP[0].PORT_PMUX[6] = 0x00U;
    ...
}
```

plib_port.c

** MCC Harmony will automatically generate configuration code and execute it at initialization.*

I/O Pin Controller (PORT) Macros

MCC Harmony GPIO Macros

Output :

- *CustomName*_Set : Set output pin to high level.
- *CustomName*_Clear : Set output pin to low level.
- *CustomName*_Toggle : Switch the output pin state, high level to low level, vice versa.

Input :

- *CustomName*_Get : Reaction logical level from Input pin.

Common :

- *CustomName*_OutputEnable : Set pin to output mode.
- *CustomName*_InputEnable : Set pin to input mode.

⚙️ Lab1 PORT Input and Output

Control PORT Example

- Control output by executing **Set**, **Clear** and **Toggle** macros.
- Read the input status by executing **Get** macros.

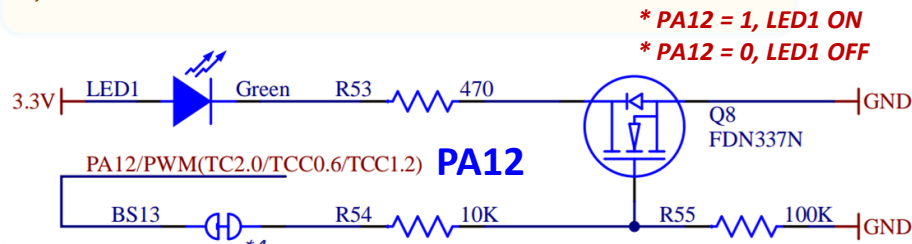
```
uint32_t i = 0;

int main (void)
{
    SYS_Initialize ( NULL );

    LED1_Set(); LED1 ON
    for(i=0;i<2000000;i++){ // Delay

    LED1_Clear(); LED1 OFF
    for(i=0;i<2000000;i++){ // Delay

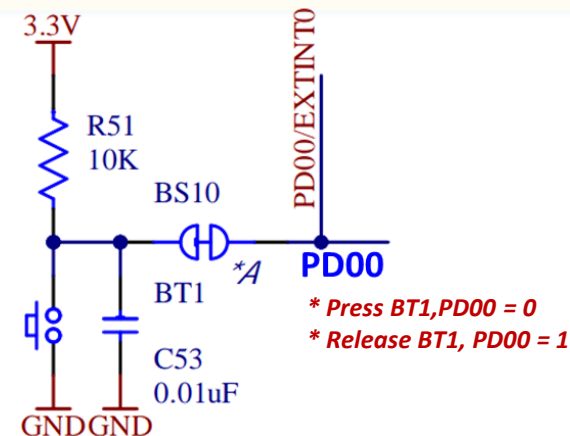
    while( true )
    {
        for(i=0;i<2000000;i++){ // Delay
            LED1_Toggle(); LED1 Toggle
        }
    }
}
```



```
uint32_t i = 0;

int main (void)
{
    SYS_Initialize ( NULL );

    while( true )
    {
        if ( BT1_Get() ) LEDn_Clear(); Release BT1, LED1 OFF
        else LEDn_Set(); Press BT1, LED1 ON
    }
}
```



🔧 Lab1 PORT Input and Output

Step 7

- Open **main.c** and finish **TODO 1.01 to 1.03** for PORT Input and Output.

```
#include <stddef.h>           // Defines NULL
#include <stdbool.h>          // Defines true
#include <stdlib.h>           // Defines EXIT_FAILURE
#include "definitions.h"      // SYS function prototypes

// TODO 1.01
uint32_t i = 0;

int main (void)
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

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

// TODO 1.02
        for ( i = 0 ; i < 2000000 ; i++ ){
            LED1_Toggle();
            LED2_Toggle();

// TODO 1.03
            if ( BT1_Get() ) LED3_Clear();
            else LED3_Set();

        }
        return ( EXIT_FAILURE );
    }
}
```

main.c

PORT initialization already done here.

Delay some time.

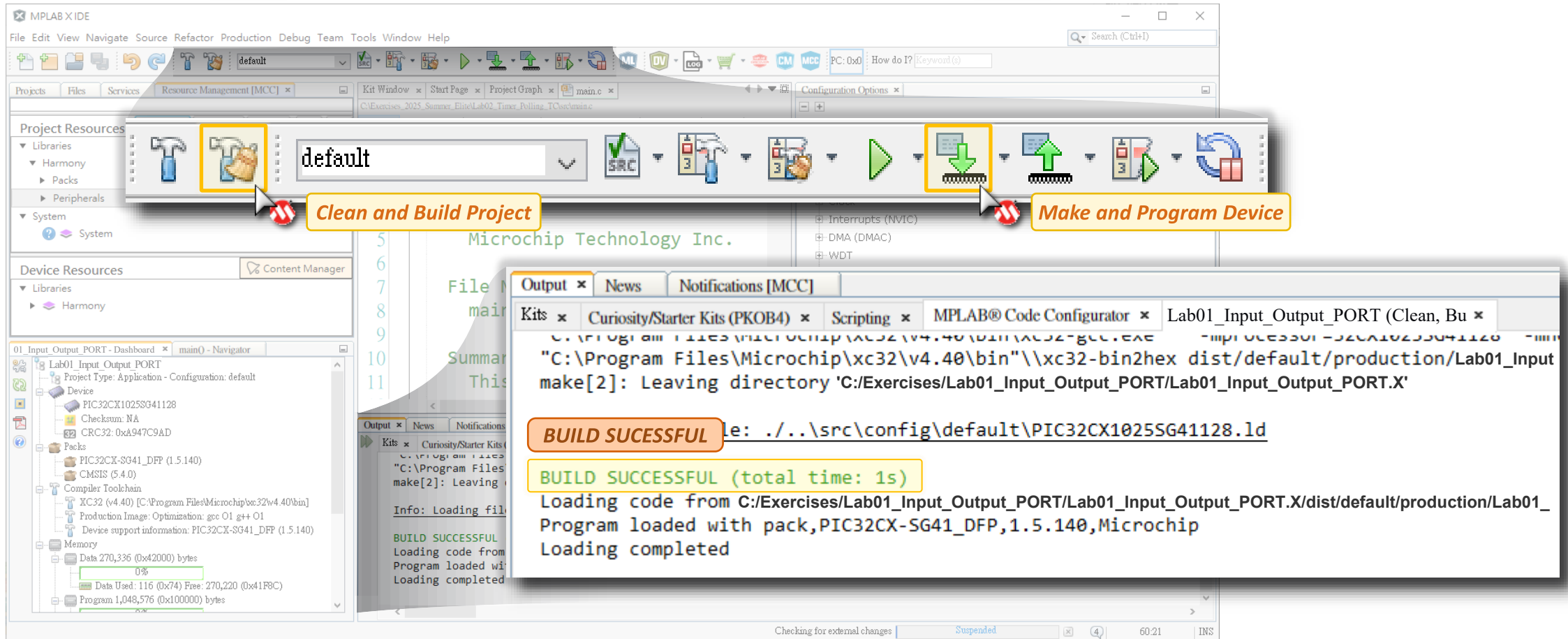
Flash LEDs.

Detect BT1 to control LED3.

⚡ Lab1 PORT Input and Output

Step 8

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains icons for various actions. Two specific icons are highlighted with yellow boxes and red arrows pointing to them:

- Clean and Build Project:** Indicated by a yellow box around the icon showing a hammer and a wrench, with a red arrow pointing to it from a yellow callout box labeled "Clean and Build Project".
- Make and Program Device:** Indicated by a yellow box around the icon showing a green arrow pointing down to a chip, with a red arrow pointing to it from a yellow callout box labeled "Make and Program Device".

The bottom panel shows the Output window with the following text:

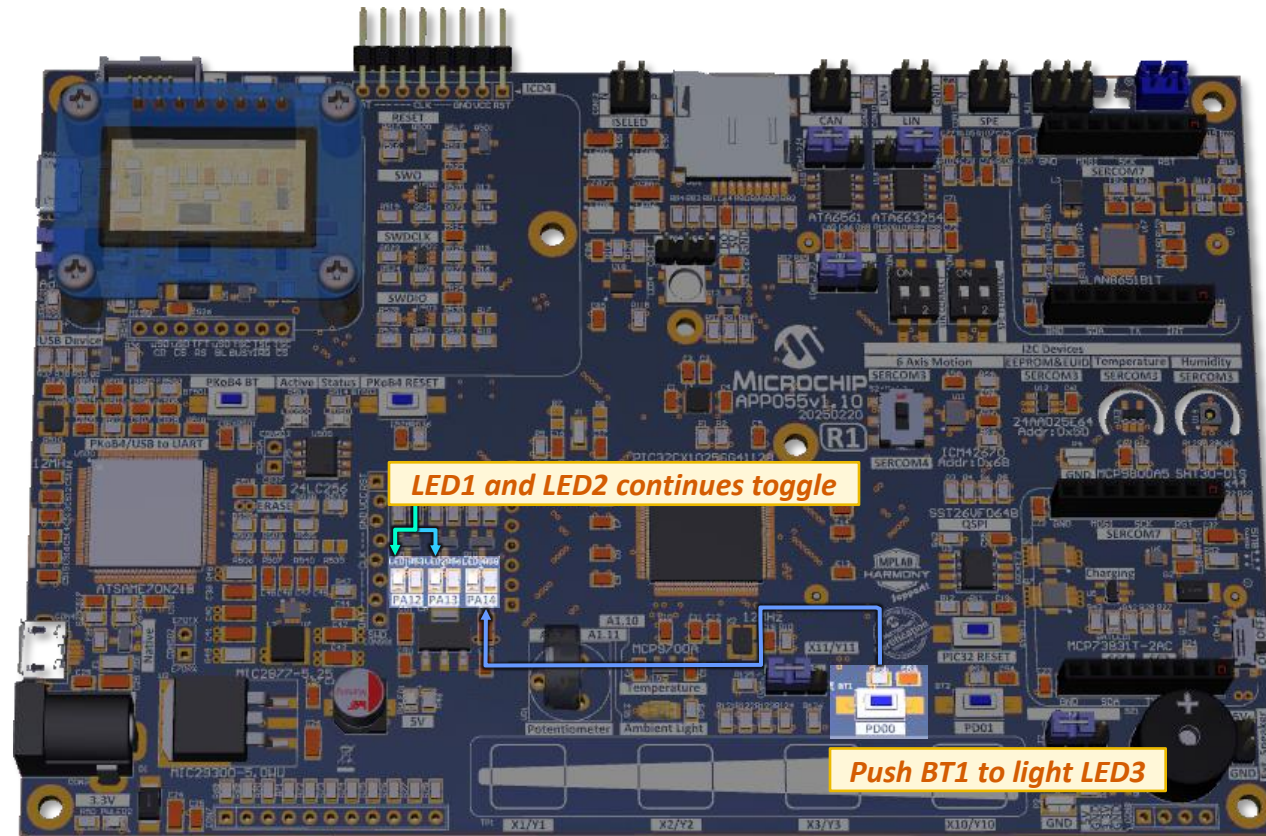
```
Output x News Notifications [MCC]
Kits x Curiosity/Starter Kits (PKOB4) x Scripting x MPLAB® Code Configurator x Lab01_Input_Output_PORT (Clean, Bu x
C:\Program Files\Microchip\xc32\v4.40\bin\XC32-GCC.EXE -imp 0CES301 -J2CA10255G41128 -imp
"C:\Program Files\Microchip\xc32\v4.40\bin"\xc32-bin2hex dist/default/production/Lab01_Input
make[2]: Leaving directory 'C:/Exercises/Lab01_Input_Output_PORT/Lab01_Input_Output_PORT.X'

BUILD SUCCESSFUL le: ../../src/config/default/PIC32CX10255G41128.ld

BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab01_Input_Output_PORT/Lab01_Input_Output_PORT.X/dist/default/production/Lab01_
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

⚙️ Lab1 PORT Input and Output

Result (to be continued)



⚙️ Lab1 PORT Input and Output

Button Response Discuss

- Does the response speed of the button depend on the LED flash period ? **Why ?**
- Try to change detection mechanism to state machine.

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

int main (void)
{
    SYS_Initialize ( NULL );

    while( true )
    {

// TODO 1.02
        for ( i = 0 ; i < 2000000 ; i++ ){}
        LED1_Toggle();
        LED2_Toggle();      * Wait until delay finish,
                           * then continue.

// TODO 1.03
        if ( BT1_Get() ) LED3_Clear();
        else LED3_Set();

    }
}
```

main.c



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

int main (void)
{
    SYS_Initialize ( NULL );

    while( true )
    {

// TODO 1.02
        if ( ++i >= 2000000 )
        {
            i = 0;      * Continuously check the status,
                       * only do something when condition match.
            LED1_Toggle();
            LED2_Toggle();
        }

// TODO 1.03
        if ( BT1_Get() ) LED3_Clear();
        else LED3_Set();

    }
}
```

main.c

🔧 Lab1 PORT Input and Output

Step 8

- Modify the **TODO 1.02** to state machine to improve detection mechanism and efficiency.

main.c

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

int main (void)
{
    SYS_Initialize ( NULL );

    while( true )
    {

// TODO 1.02
        // for ( i = 0 ; i < 2000000 ; i++ ){}
        // LED1_Toggle();
        // LED2_Toggle();

        if ( ++i >= 2000000 )
        {
            i = 0;
            LED1_Toggle();
            LED2_Toggle();
        }

// TODO 1.03
        if ( BT1_Get() ) LED3_Clear();
        else LED3_Set();

    }
}
```

*The response speed of the Button
depend on LED flash period.*

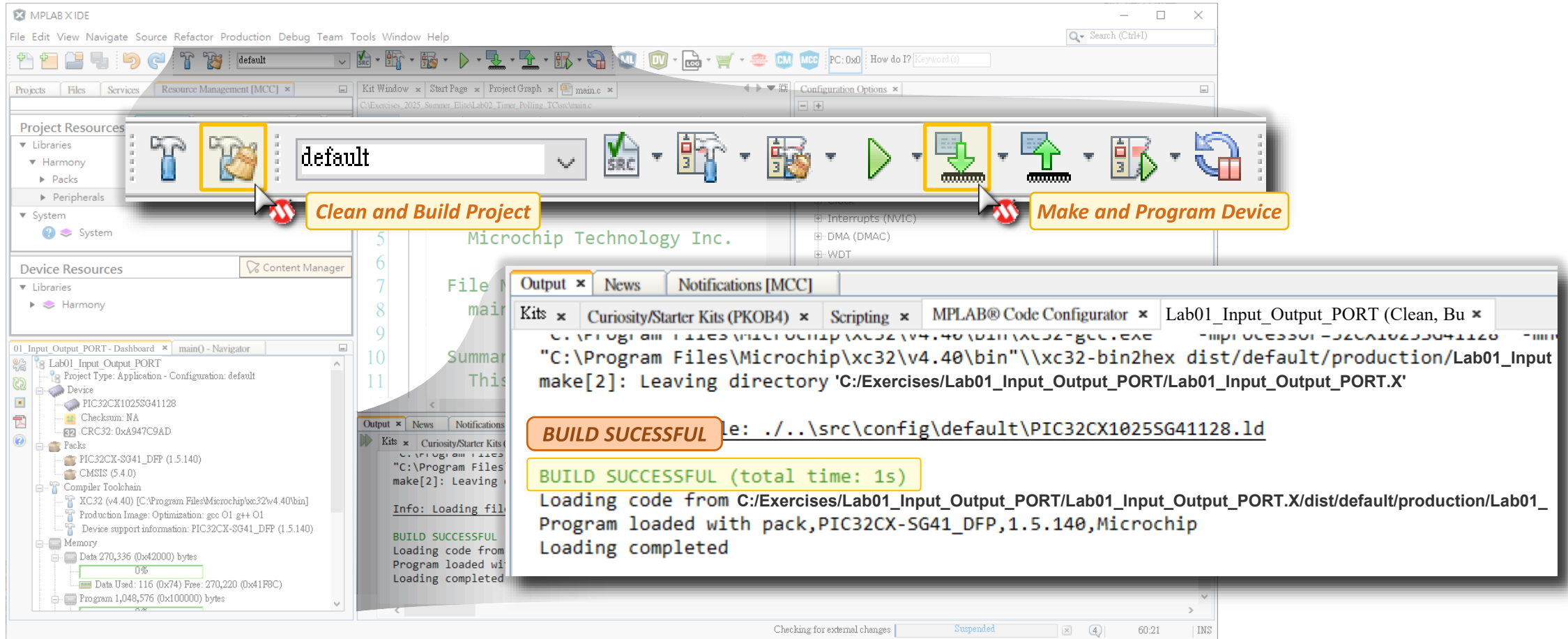
Why ?

- *Because in the previous mechanism the CPU needs to wait delay complete and can't do other things at the same time.*
- *Changing the detection mechanism to state machine can improve control method and efficiency.*

⚙️ Lab1 PORT Input and Output

Step 9

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains several icons, with two highlighted by yellow boxes and red arrows. The first icon, labeled "Clean and Build Project", is a hammer and wrench. The second icon, labeled "Make and Program Device", is a green arrow pointing down to a chip. Below the toolbar, the "Output" window is open, showing the results of the build process. The output text includes:

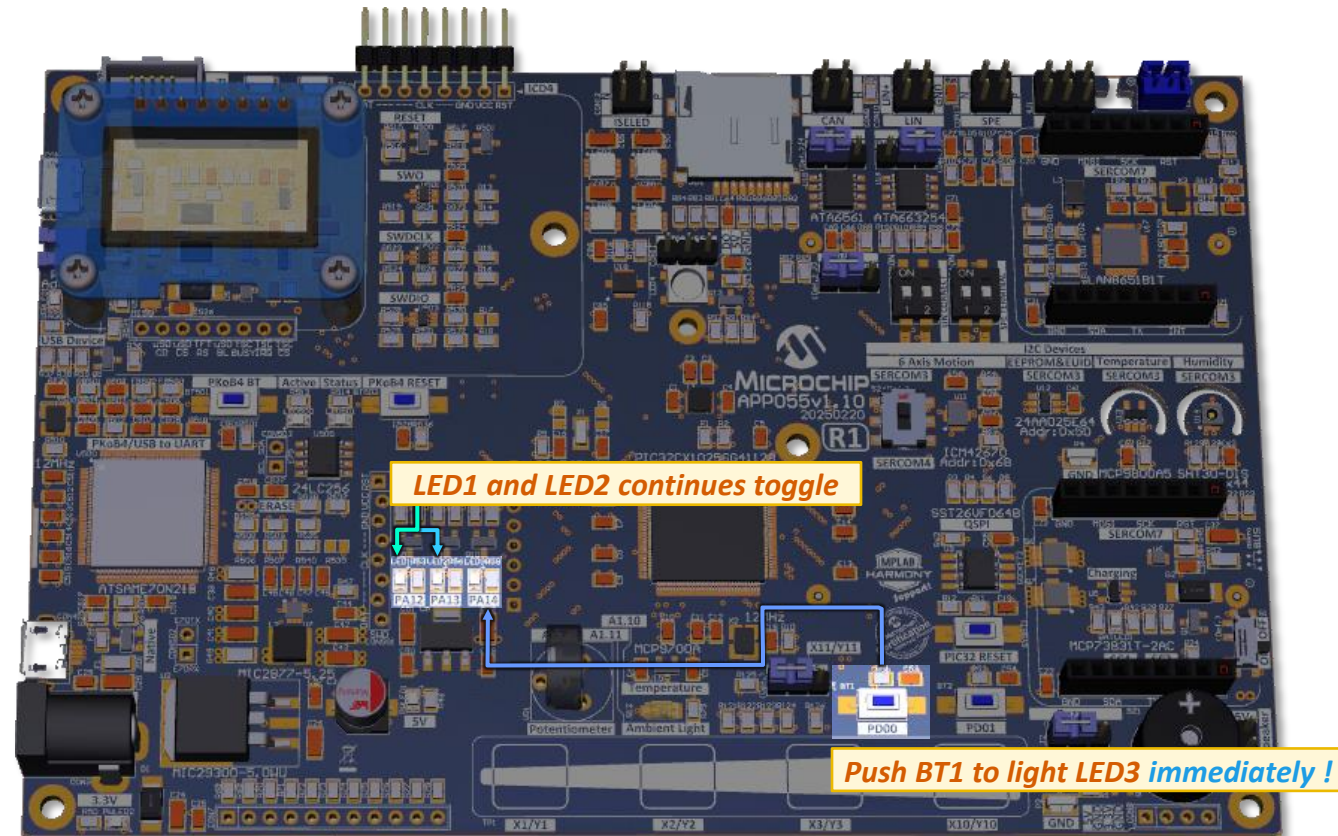
```
Le: ../../src/config/default/PIC32CX1025SG41128.ld  
BUILD SUCCESSFUL (total time: 1s)  
Loading code from C:/Exercises/Lab01_Input_Output_PORT/Lab01_Input_Output_PORT.X/dist/default/production/Lab01_  
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip  
Loading completed
```

The "Output" window also shows the following text:

```
make[2]: Leaving directory 'C:/Exercises/Lab01_Input_Output_PORT/Lab01_Input_Output_PORT.X'
```


⚡ Lab1 PORT Input and Output

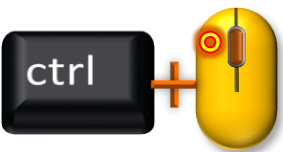
Result



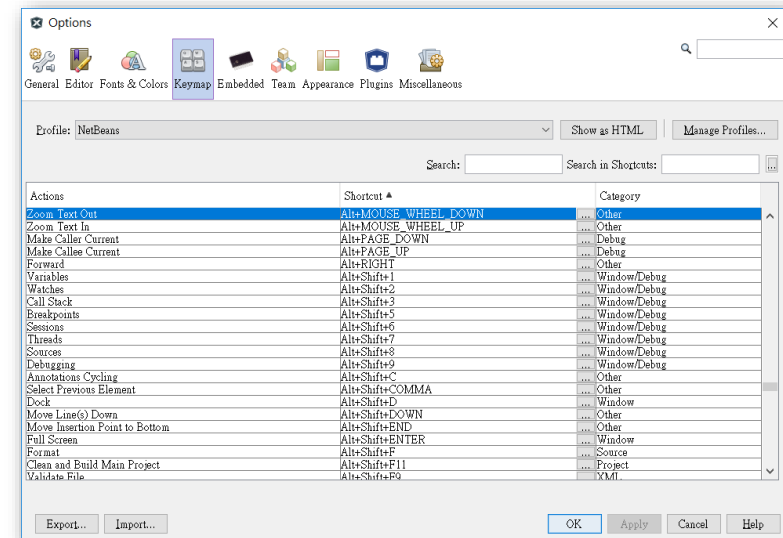
Appendix : Code Trace and PORT Register

MPLAB X IDE Shortcut key

- **MPLAB X IDE Provide a few shortcut key to enhance your performance.**

-  Show All Code Completion Popup (Code Automatically Complete)
(Alternative shortcut : Ctrl + \ , Ctrl + Alt + Space)
-  Open Macro or Function (Code Trace)
-  Backward (Code Trace)
-  Code Format
-  Toggle Comment

- **More ? Tools > Options > Keymap**



MPLAB X IDE Code Trace

Prototype of Macros or Functions

- Using shortcut keys, you can easily find the code prototype of Macros or Functions, and return to the previous location at any time.

```
int main (void)
{
    ...
    while( true )
    {
        if ( ++i >= 2000000 )
        {
            i = 0;
            LED1_Toggle();
            LED2_Toggle();
        }
        ...
    }
}
```

main.c

```
...
/** Macros for LED1 pin */
#define LED1_Set() (PORT_REGS->GROUP[0].PORT_OUTSET = ((uint32_t)1U << 12U))
#define LED1_Clear() (PORT_REGS->GROUP[0].PORT_OUTCLR = ((uint32_t)1U << 12U))
#define LED1_Toggle() (PORT_REGS->GROUP[0].PORT_OUTTGL = ((uint32_t)1U << 12U))
#define LED1_OutputEnable() (PORT_REGS->GROUP[0].PORT_DIRSET = ((uint32_t)1U << 12U))
#define LED1_InputEnable() (PORT_REGS->GROUP[0].PORT_DIRCLR = ((uint32_t)1U << 12U))
#define LED1_Get() (((PORT_REGS->GROUP[0].PORT_IN >> 12U)) & 0x01U)
#define LED1_PIN PORT_PIN_PA12
...
* Macros will be named according to your custom GPIO name.
```

plib_port.h

PORT Register Access

AHB-APB Bridge B

* Advanced High-performance Bus.

* Advanced Peripheral Bus.

0x41000000	USB
0x41002000	DSU
0x41004000	NVMCTRL
0x41006000	CMCC
0x41008000	PORT
0x4100A000	DMAC
0x4100C000	Reserved
0x4100E000	...
...	...

PORTA	DIR	0x41008000
	DIRCLR	0x41008004
	DIRSET	0x41008008
	DIRTGL	0x4100800C
	OUT	0x41008010
	OUTCLR	0x41008014
	OUTSET	0x41008018
	OUTTGL	0x4100801C
	...	
	DIR	0x41008080
PORTB	DIRCLR	0x41008084
	...	0x41008088
	...	0x4100808C
...	...	0x41008090

Register Summary from datasheet

Offset	Name	Bit Pos.
0x00	DIR	7:0
		15:8
		23:16
		31:24
0x04	DIRCLR	7:0
		15:8
		23:16
		31:24
0x08	DIRSET	7:0
		15:8
		23:16
		31:24
0x0C	DIRTGL	7:0
		15:8
		23:16
		31:24
0x10	OUT	7:0
		15:8
		23:16
		31:24
0x14	OUTCLR	7:0
		15:8
		23:16
		31:24
0x18	OUTSET	7:0
		15:8
		23:16
		31:24
0x1C	OUTTGL	7:0
		15:8
		23:16
		31:24
...		

PORT Register Access

```
...  
/*** Macros for LED1 pin ***/  
...  
#define LED1_Toggle() (PORT_REGS->GROUP[0].PORT_OUTTGL = ((uint32_t)1U << 12U))  
...
```

plib_port.h

- There are two way to access Register:

- Access by Register Pointer : `*(__IO uint32_t *) (0x4100801CU) = ((uint32_t)1U << 12U));`
- Access by Pre-defined Struct : `(PORT_REGS->GROUP[0].PORT_OUTTGL = ((uint32_t)1U << 12U))`
**PORT_REGS->GROUP[0].PORT_OUTTGL = 0x4100801C*

PORT_REGS->GROUP[0]

PORTA	DIR	0x41008000
	DIRCLR	0x41008004
	DIRSET	0x41008008
	DIRTGL	0x4100800C
	OUT	0x41008010
	OUTCLR	0x41008014
	OUTSET	0x41008018
	OUTTGL	0x4100801C
	...	

```
...  
#define PORT_REGS ((port_registers_t*) 0x41008000)  
...  
...  
/* GROUP register API structure */  
typedef struct  
{  
    __IO uint32_t    PORT_DIR;    /* Offset: 0x00 */  
    ...  
    __IO uint32_t    PORT_OUT;    /* Offset: 0x10 */  
    __IO uint32_t    PORT_OUTCLR; /* Offset: 0x14 */  
    __IO uint32_t    PORT_OUTSET; /* Offset: 0x18 */  
    __IO uint32_t    PORT_OUTTGL; /* Offset: 0x1C */  
    ...  
} port_group_registers_t;  
  
#define PORT_GROUP_NUMBER 4  
  
/* PORT register API structure */  
typedef struct  
{ /* Port Module */  
    port_group_registers_t GROUP[PORT_GROUP_NUMBER];  
} port_registers_t;  
...
```

pic32cx1025sg41128.h

port.h

`((uint32_t)1U << 12U))`
0x1000 (Hex)
0001 0000 0000 0000 b (Binary)
1111 11
5432 1098 7654 3210

PORT Register Access

Access PORT Register to control LED example

- You can control LED by accessing the Register Pointer or Pre-defined Struct of PORT, but it requires some knowledge.

```
uint32_t i = 0;
int main (void)
{
    SYS_Initialize ( NULL );
    while( true )
    {
        if ( ++i >= 2000000 )
        {
            i = 0;
            *(__IO uint32_t *) (0x4100801CU) = ((uint32_t)1U << 12U);
        }
    }
}
```

main.c

Access by Register Pointer to flash LED1

OR

```
uint32_t i = 0;
int main (void)
{
    SYS_Initialize ( NULL );
    while( true )
    {
        if ( ++i >= 2000000 )
        {
            i = 0;
            (PORT_REGS->GROUP[0].PORT_OUTTGL = ((uint32_t)1U << 12U))
        }
    }
}
```

main.c

Access by Pre-defined Struct to flash LED1

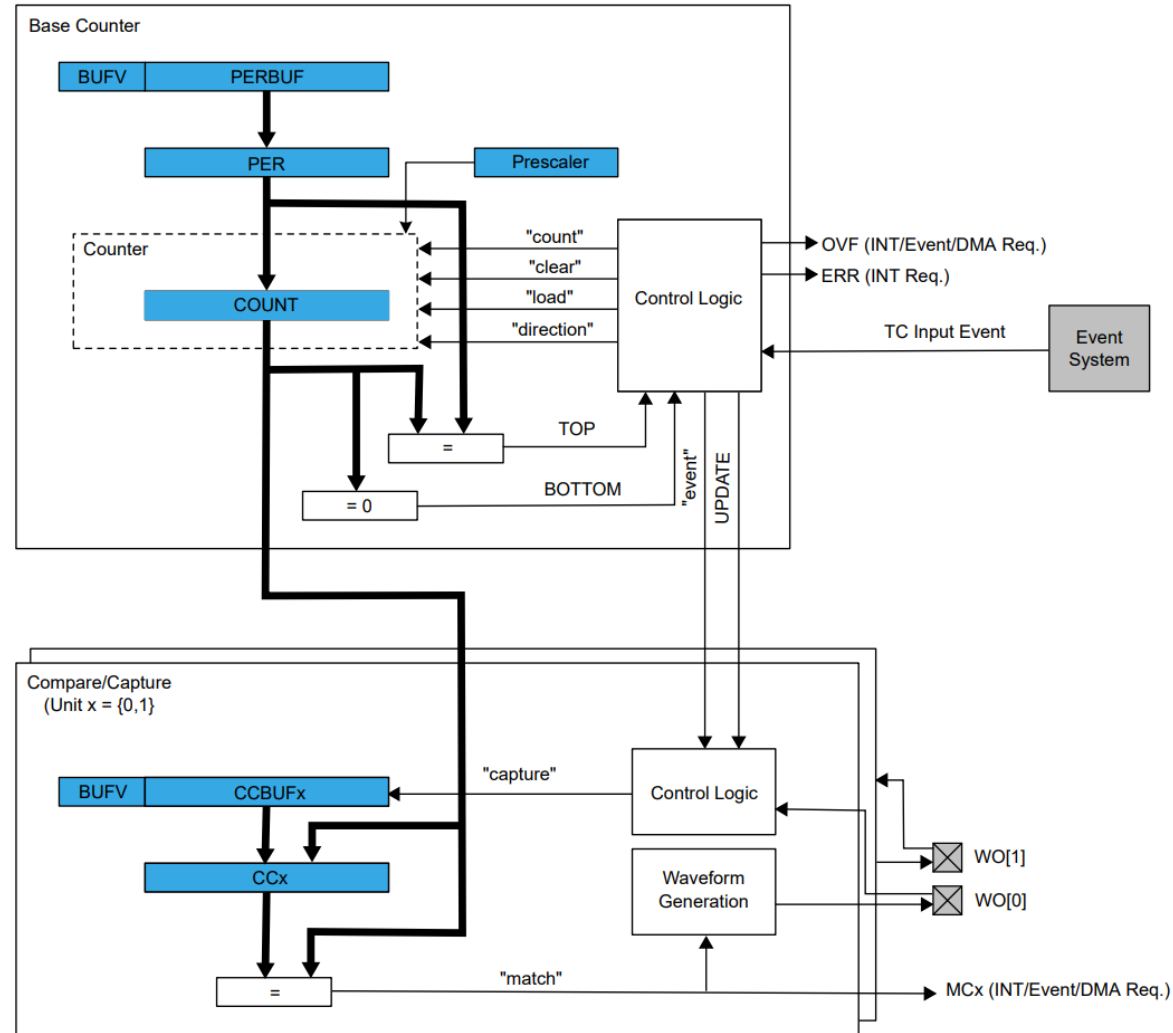
Timer by Polling - TC

PIC32CX-SG Timer/Counter

- The PIC32CX-SG Timer/Counter (TC) modules consists of Counter, Compare, Capture and Control Logic.
- The counter can be set to count events or clock pulses. The counter, together with the compare/capture channels, can be configured to timestamp input events or I/O pin edges, allowing for capturing of frequency and/or pulse width.
- It can also perform waveform generation, such as frequency generation and pulse-width modulation (PWM).

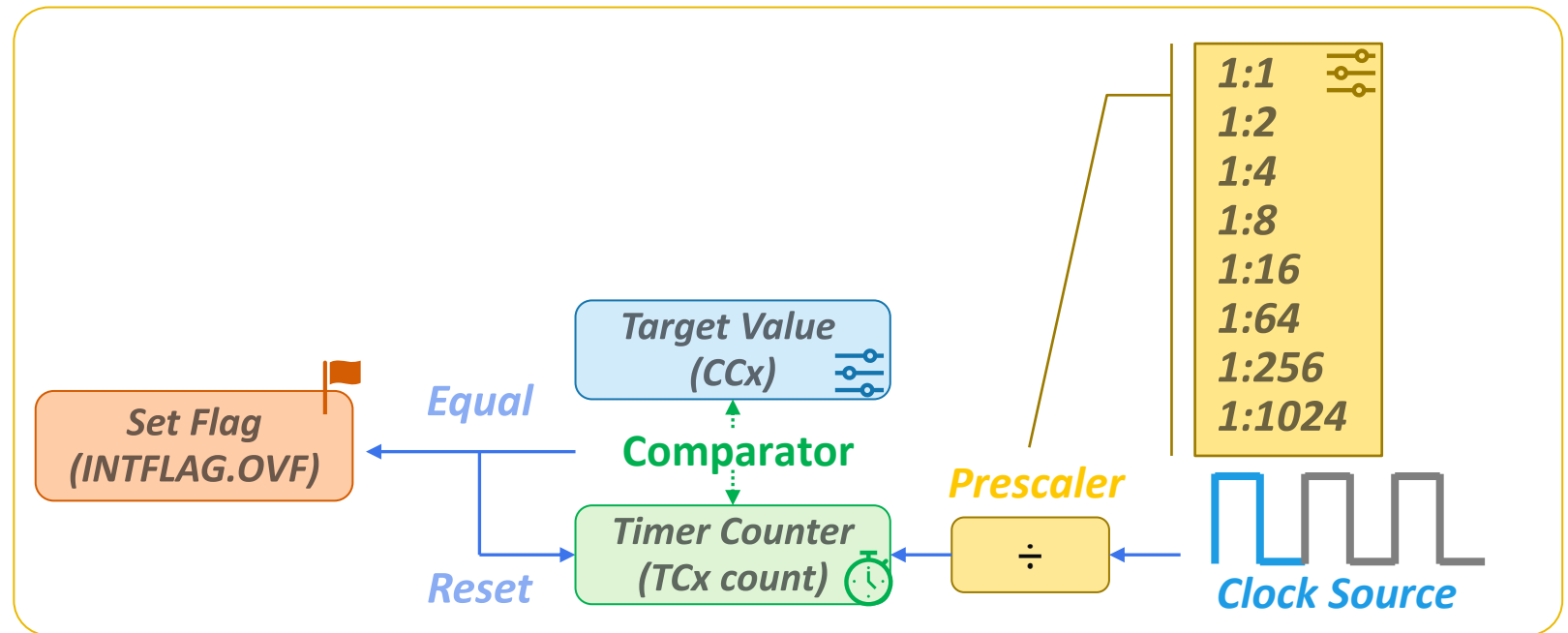
Timer/Counter (TC) Block Diagram

- PIC32CX-SG TC Block Diagram



PIC32CX-SG Timer/Counter

- The block diagram just a concept. For actual and detailed block diagram, please refer to datasheet.
- The clock will via the prescaler and into **TCx count**, continuously comparing with the **set target value (CCx)**. When matched, it will set the flag and reset the counter to zero.
- You can fill a value to **CCx** to determine period you want.



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

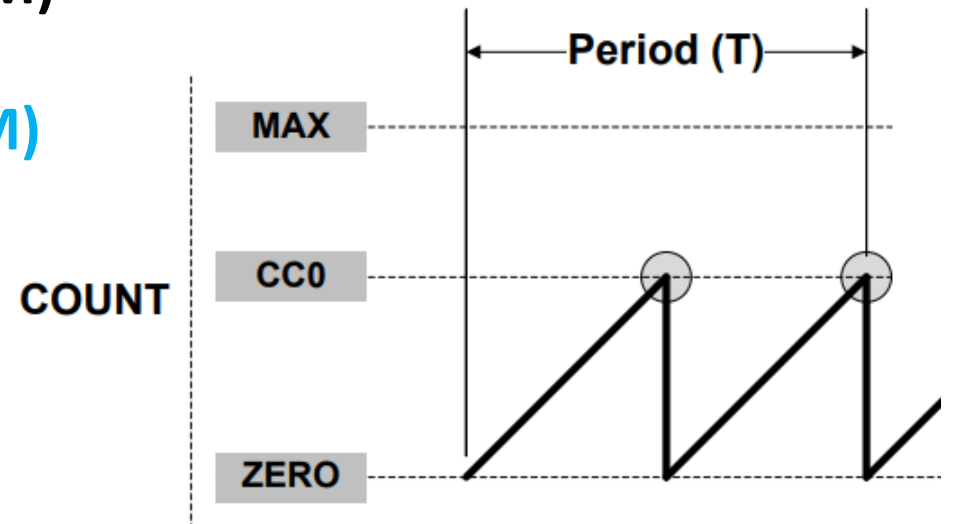
PIC32CX-SG Timer/Counter

- **TC modules provide more difference operation mode for various control methods.**

This training only force on Timer (default is **MPWM** mode). It's easy to use.

If you want to know more, please refer the datasheet.

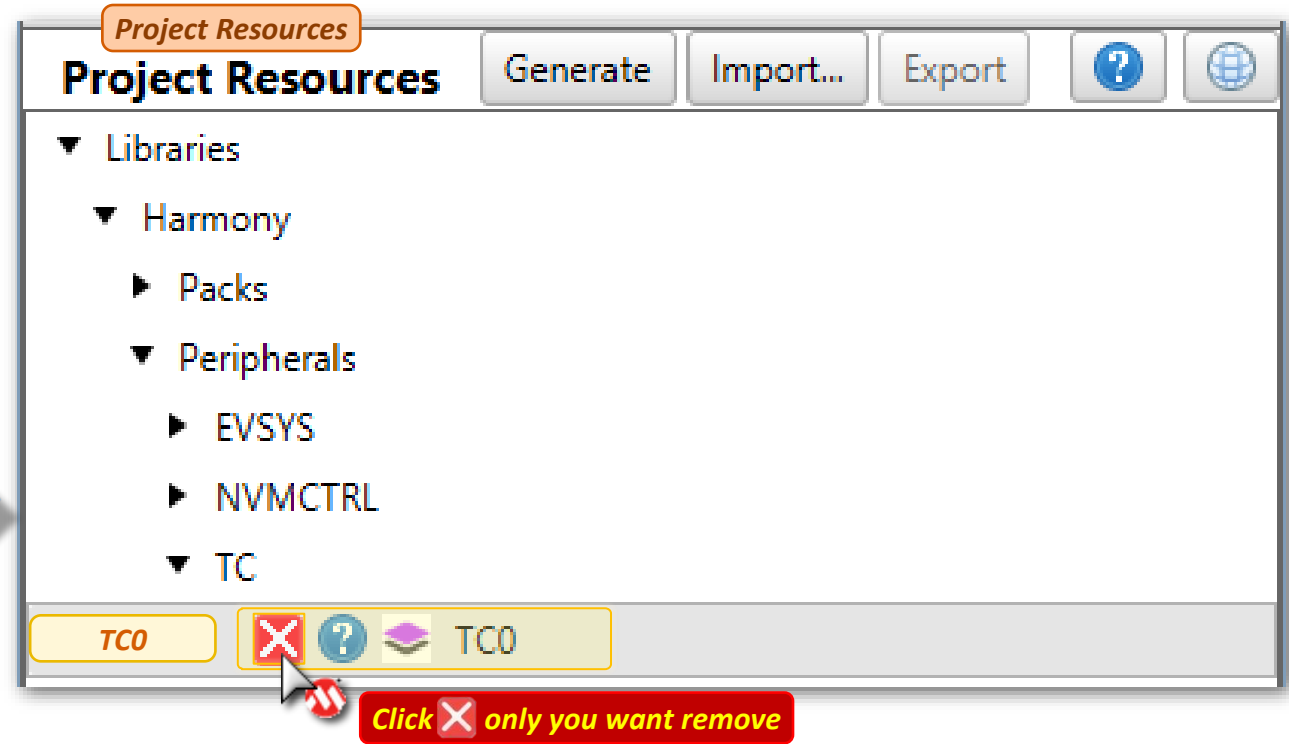
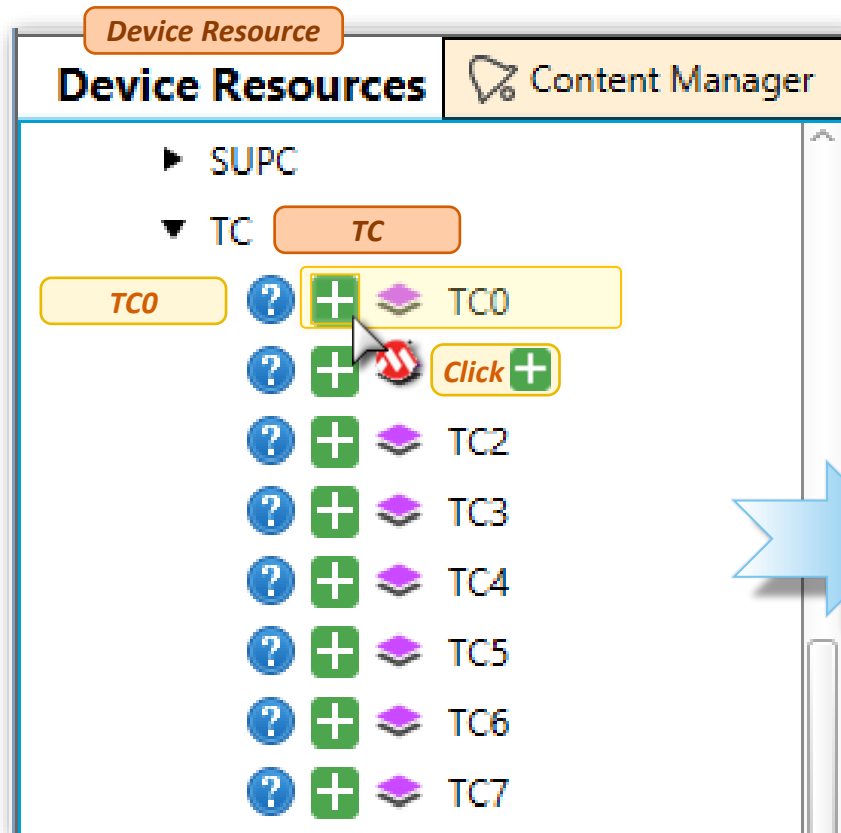
- **Normal Frequency Generation (NFRQ)**
toggled on each compare match
- **Match Frequency Generation (MFRQ)**
toggles on each update condition
- **Normal Pulse-Width Modulation Operation (NPWM)**
toggles on each match & update condition
- **Match Pulse-Width Modulation Operation (MPWM)**
toggles on each match & update condition



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

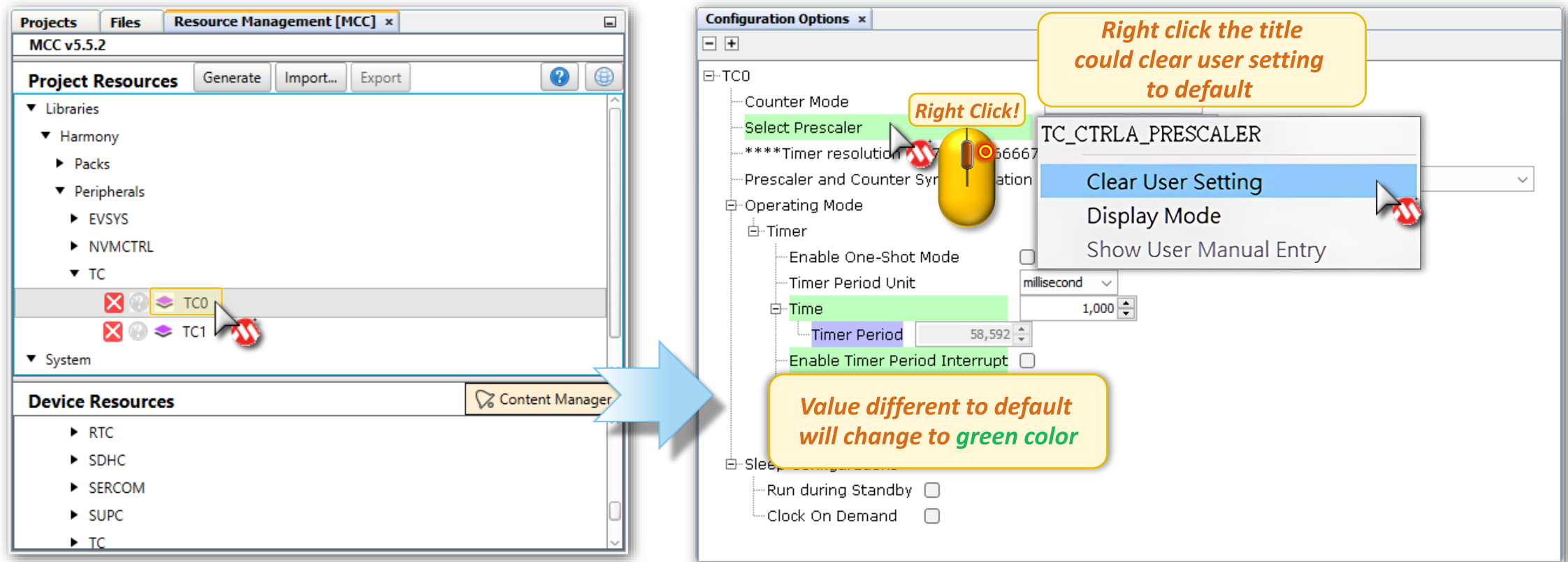
Add or Remove MCC Harmony Module

- Find required component in **Device Resources**.
- Click on  to add the module to your **Project Resources**.
- Click on  could remove the module from Project Resource.



MCC Harmony Module Configuration Options Example

- Click the modules under **Project Resources** to open its configuration options interface.
- You can modify the value of the option, and the modified items will be highlighted in **Green**, then you can use the **Clear User Setting** feature to restore the modified items to the default when you need.



Lab2 TC Timer Method Polling

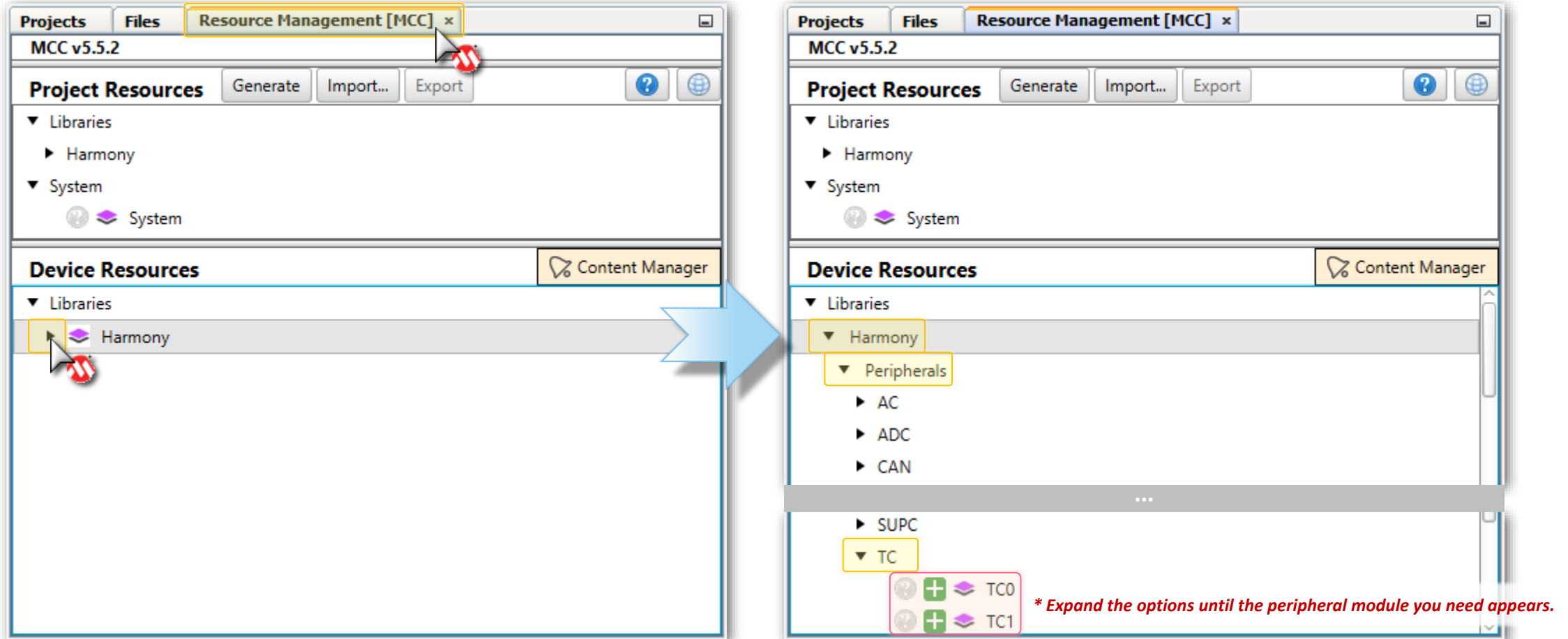
Lab2 TC Timer Method Polling

- Try adding **TC0** and **TC1** instead of software delay to control **LED1(PA12)** and **LED2(PA13)** to continues toggle.
- Setup the **TC0** to **Timer** and **Polling method** (MPWM 16-bit mode), and the Timer period to **500ms**, LED1 will flash at 500ms ON/OFF.
- Setup the **TC1** to **Timer** and **Polling method** (MPWM 16-bit mode), and the Timer period to **40ms**, LED2 will flash at 40ms ON/OFF.

Let's go !

Find the Peripheral Modules in the MCC Device Resources

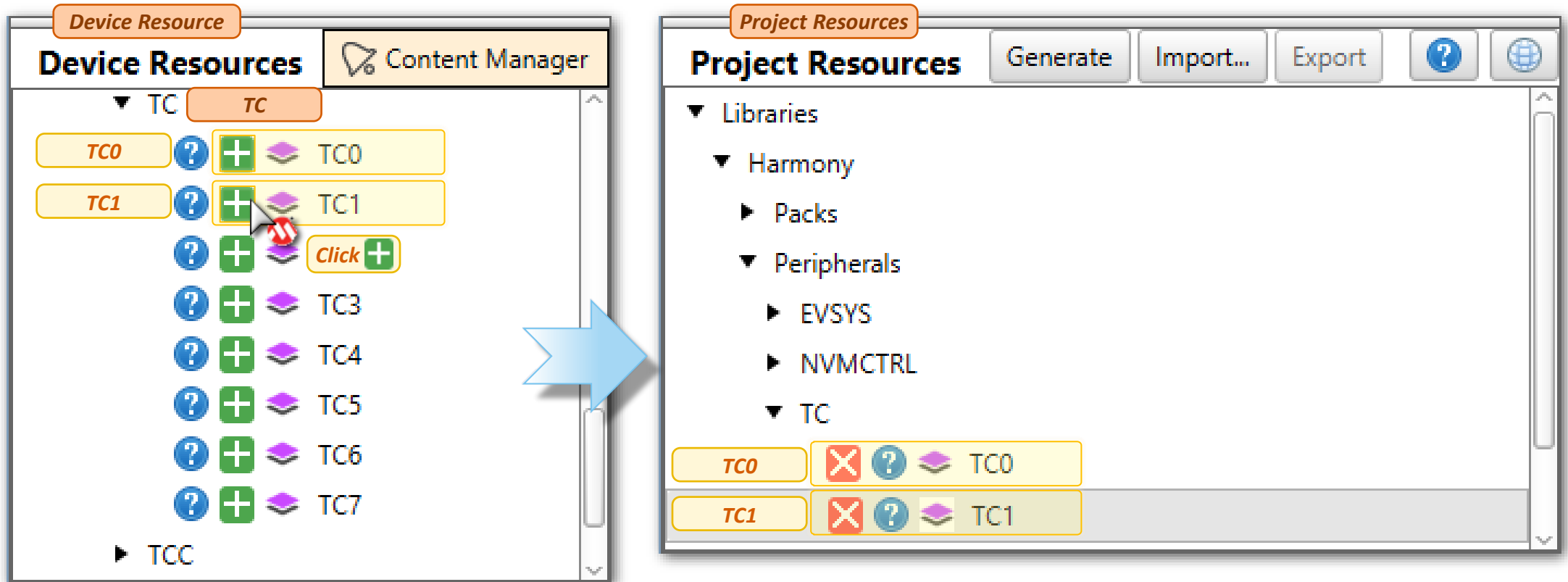
- By default, the MCC Device Resources are collapsed. You need to expand them before you want find the peripheral modules and the click  button.



⚙️ Lab2 TC Timer Method Polling

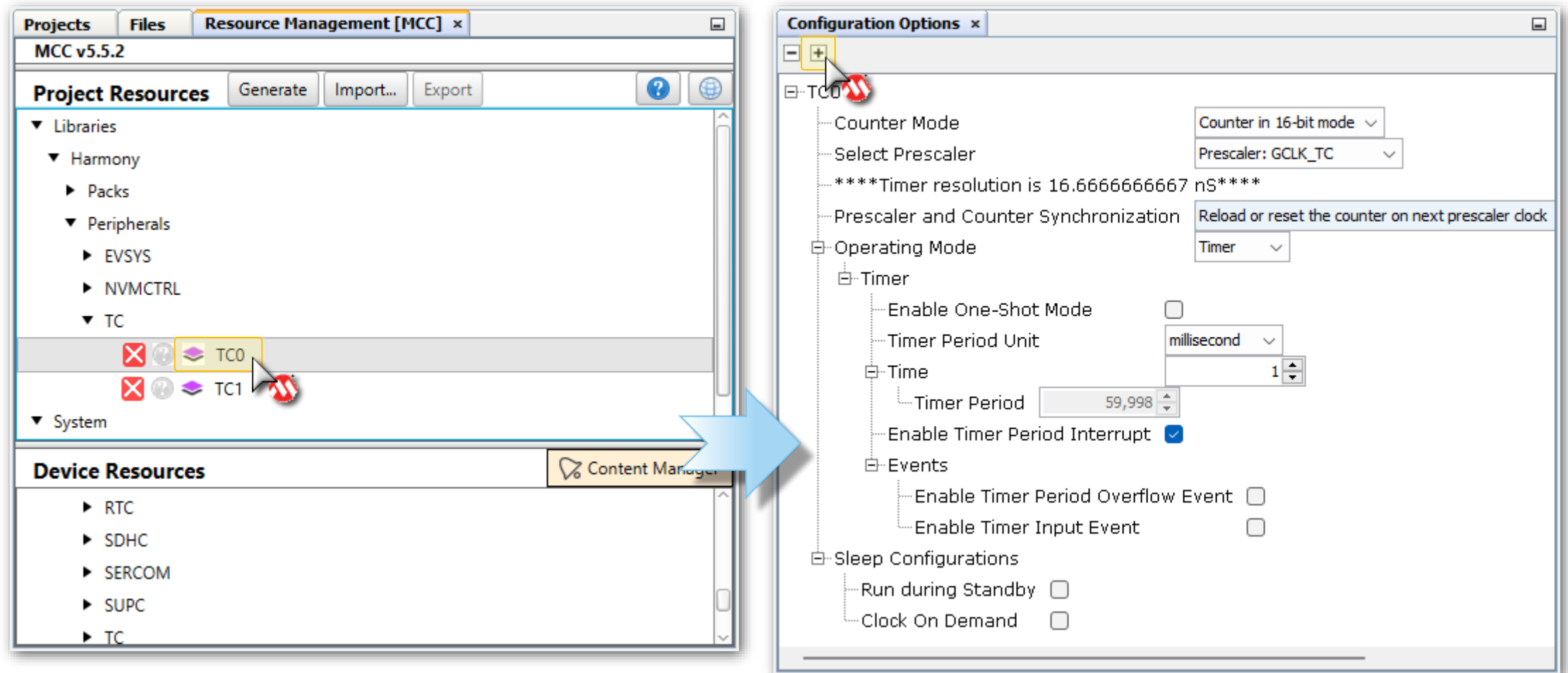
Step 1

- Find **TC0** and **TC1** component in **Device Resources**, we use it for LED toggle delay.
- Click on **+** to add the **TC0** and **TC1** module to your **Project Resources**.



MCC Harmony Module Configuration Options Example

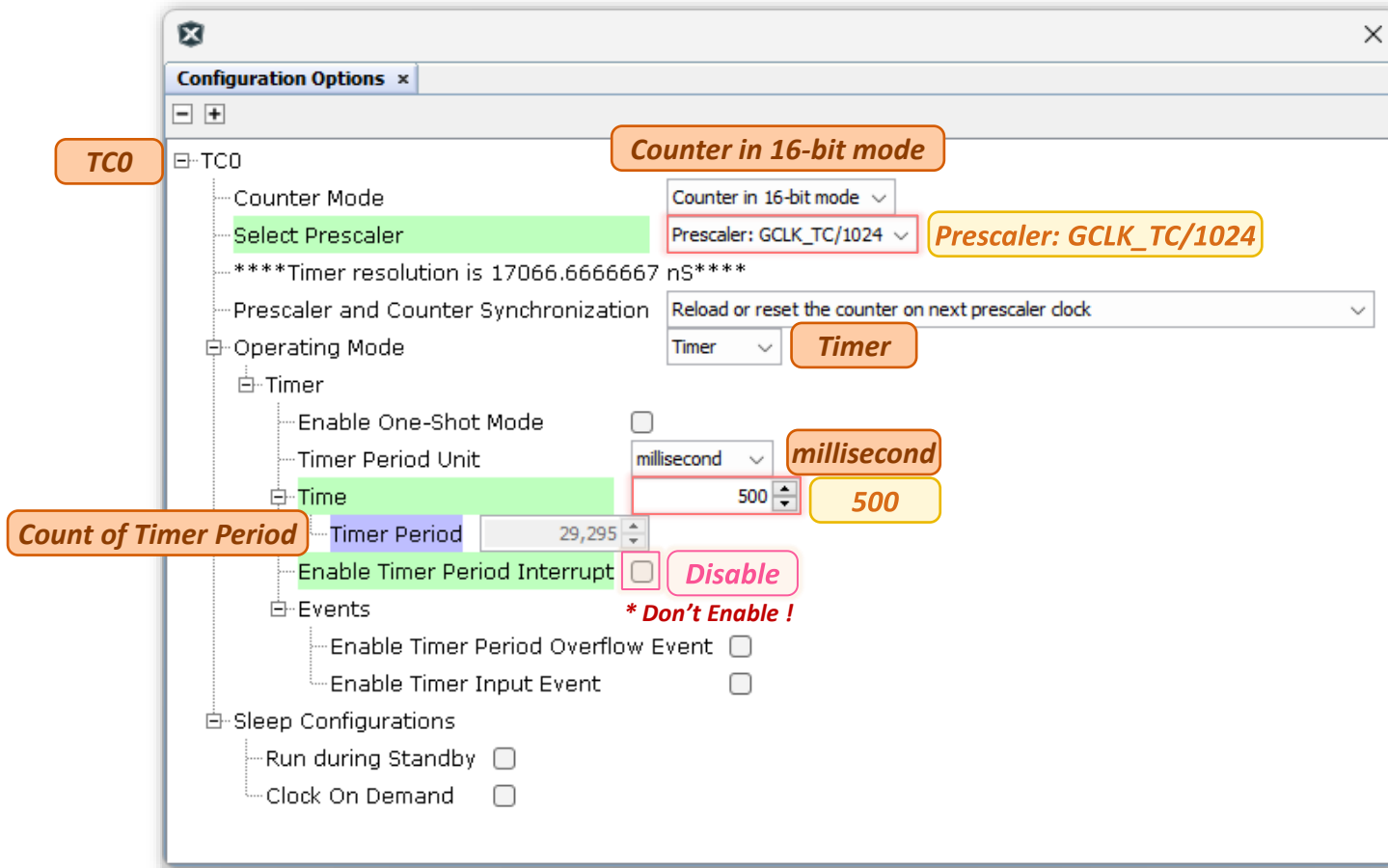
- Click the peripheral modules under Project Resources to open its configuration options interface. You can click on the top  to expand all options.



✖ Lab2 TC Timer Method Polling

Step 2

- Modify the **Prescaler** and fill the **Timer Period** of **TC0**.



TC Clock = **60MHz**

One second counts of system clock :

60,000,000 counts

One second counts after Prescaler :

$60,000,000 / 1024 = 58,593.75$
 $\approx 58,593 \text{ counts}$

500ms counts for target :

$(500\text{ms} / 1\text{s}) * 58,593 = 29,296.5$
 $\approx 29,296 \text{ counts}$

Counter clear to zero at the next tick :

$29,295 + 1 = 29,296 \text{ counts}$

How to calculate count 29,295 ?

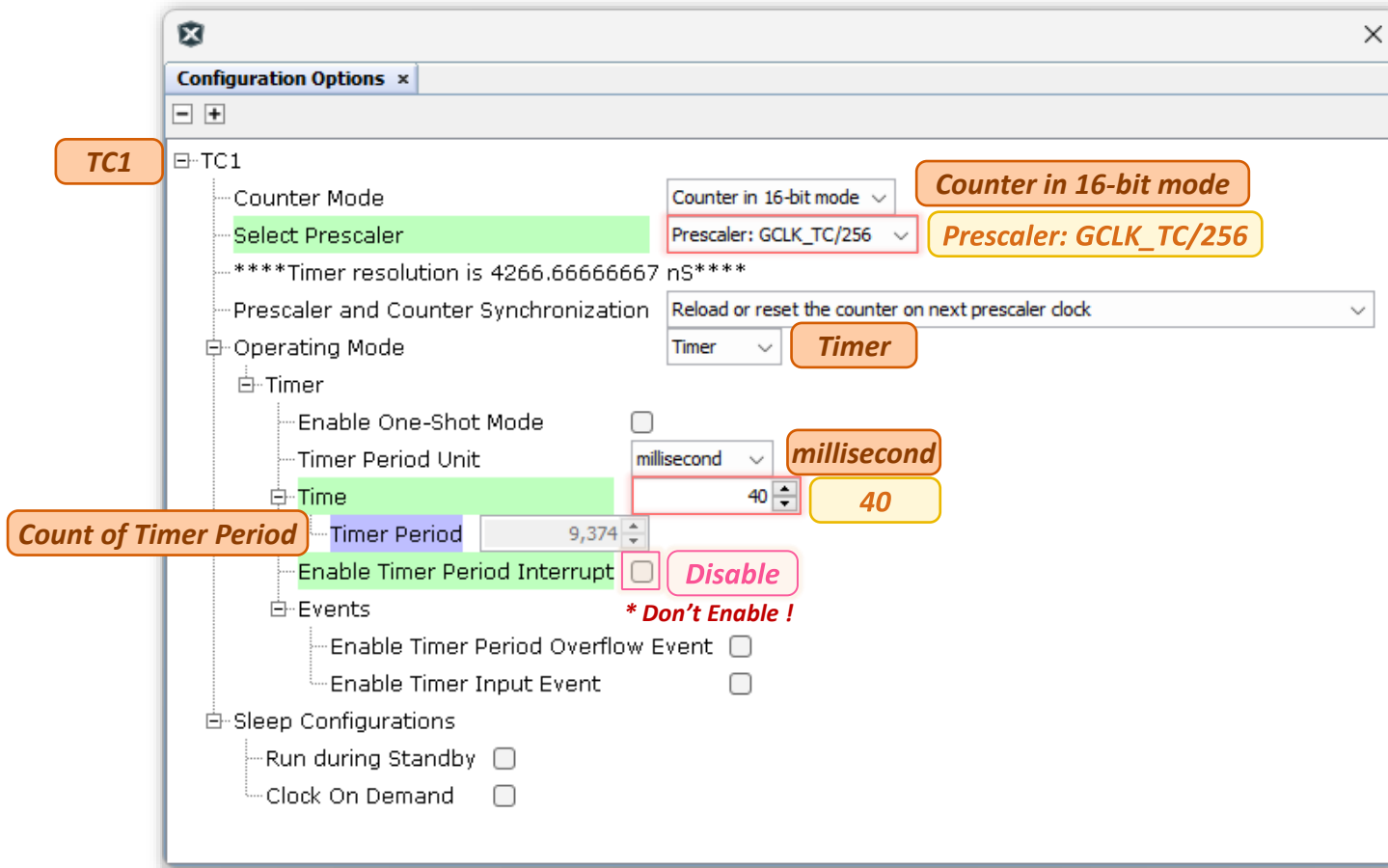
** The timer period may have error when the set time is not a multiple of the clock cycle, because the minimum unit of the timer is 1 clock.*

** The actual timer period = clock cycle * timer count.
 $(1/60\text{MHz} * 1,024) * 29,296 = 0.499985\text{s} \approx 500\text{ms}$*

✖ Lab2 TC Timer Method Polling

Step 3

- Modify the **Prescaler** and fill the **Timer Period** of TC1.



TC Clock = 60MHz

One second counts of system clock :

60,000,000 counts

One second counts after Prescaler :

$60,000,000 / 256$ 234,375 counts

40ms counts for target :

$(40ms / 1s) * 234,375$ 9,375 counts

Counter clear to zero at the next tick :

$9,374 + 1$ 9,375 counts

How to calculate count 9,374 ?

* The timer period = clock cycle * timer count.
 $(1/60MHz * 256) * 9,375 = 40ms$

Lab2 TC Timer Method Polling

Step 4

- Click on **Generate** button to generate your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Project Resources' tab. The 'Generate' button is highlighted with a yellow box and a red arrow. Below it, the 'System' resource is also highlighted. The 'MPLAB® Code Configurator' window is open, showing the 'Output' tab. The output log displays the following information:

```
15:17:50.827 INFO: ... PIC32CX1023SG41128_DFP (p10 (pic32cx1023sg41128.1)) Success.  
15:17:50.827 INFO: HARMONY: [ MHC ] Executing generate hook  
15:17:50.843 INFO:   
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)  
15:17:50.844 INFO:   
15:17:50.844 INFO: Generation complete  
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent  
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent  
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel  
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

TC Polling

TC n Interface Functions

- **TC n _TimerInitialize** : Initialize the TC n module in Timer mode
- **TC n _TimerStart** : Enable the TC n counter/timer
- **TC n _TimerStop** : Disable the TC n counter/timer
- **TC n _Timer16bitPeriodSet** : Configure timer period
- **TC n _Timer16bitPeriodGet** : Read the timer period value
- **TC n _Timer16bitCounterSet** : Configure timer counter value
- **TC n _Timer16bitCounterGet** : Get the current timer counter value
- **TC n _TimerPeriodHasExpired** : Polling method to check if timer period interrupt flag is set

TC Polling

TC_n Interface Functions

void TC_n_TimerStart (**void**)

- **Description:**
 - Enable the TC_n counter/timer.

bool TC_n_TimerPeriodHasExpired (**void**)

- **Description:**
 - Return timer period interrupt flag status.

- **Example :**

```
TCn_TimerStart();  
...  
if( TCn_TimerPeriodHasExpired() )  
{  
    // do something ...  
}
```

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.

Bit	7	6	5	4	3	2	1	0
			MC1	MC0			ERR	OVF
Access			R/W	R/W			R/W	R/W
Reset			0	0			0	0

- **How Polling method to check timer overflow interrupt flag status ?**

TC Polling Code Example

TC Timer Polling method (MPWM Mode) Overflow Example

- Use functions to start the timer and continuously check whether the timer complete.

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

    while( true )
    {
        if ( TCx_TimerPeriodHasExpired() )
        {
            LED1_Toggle();
        }

    }

    /* Execution should not come here during normal operation */

    return ( EXIT_FAILURE );
}
```

main.c

```
...
bool TCx_TimerPeriodHasExpired( void )
{
    uint8_t timer_status = 0U;
    timer_status = (uint8_t) ( (TCx_REGS->COUNT16.TC_INTFLAG) & TC_INTFLAG_OVF_Msk );
    TCx_REGS->COUNT16.TC_INTFLAG = timer_status;
    return (timer_status != 0U);
}
```

plib_tcX.c

* When the timer is completed, timer overflow interrupt flag (OVF) = 1.
* Use TC_INTFLAG_OVF_Msk to mask TC interrupt flag register to check OVF event.

0
OVF

Bit	7	6	5	4	3	2	1	0
			MC1	MC0			ERR	OVF
Access			R/W	R/W			R/W	R/W
Reset			0	0			0	0

Lab2 TC Timer Method Polling

Step 5

- Open **main.c** and finish **TODO 2.01** to comment out or remove the code of previous lab.

```
...
#include <stdlib.h>                // Defines EXIT_FAILURE
#include "definitions.h"          // SYS function prototypes

uint32_t i = 0;

int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    while ( true )
    {
        SYS_Tasks ( );

// TODO 2.01
        // if ( ++i >= 2000000 )
        // {
        //     i = 0;
        //     LED1_Toggle();
        //     LED2_Toggle();
        // }

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

    return ( EXIT_FAILURE );
}
```

main.c

Lab2 TC Timer Method Polling

Step 6

- Finish **TODO 2.02** to **2.03** for enable TC0 and TC1, then flash LEDs regularly.

```
int main (void)
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    // TODO 2.02
    TC0_TimerStart();
    TC1_TimerStart();

    while( true )
    {
        ...

        // TODO 2.03
        if ( TC0_TimerPeriodHasExpired() )
        {
            LED1_Toggle();
        }

        if ( TC1_TimerPeriodHasExpired() )
        {
            LED2_Toggle();
        }

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

    return ( EXIT_FAILURE );
}
```

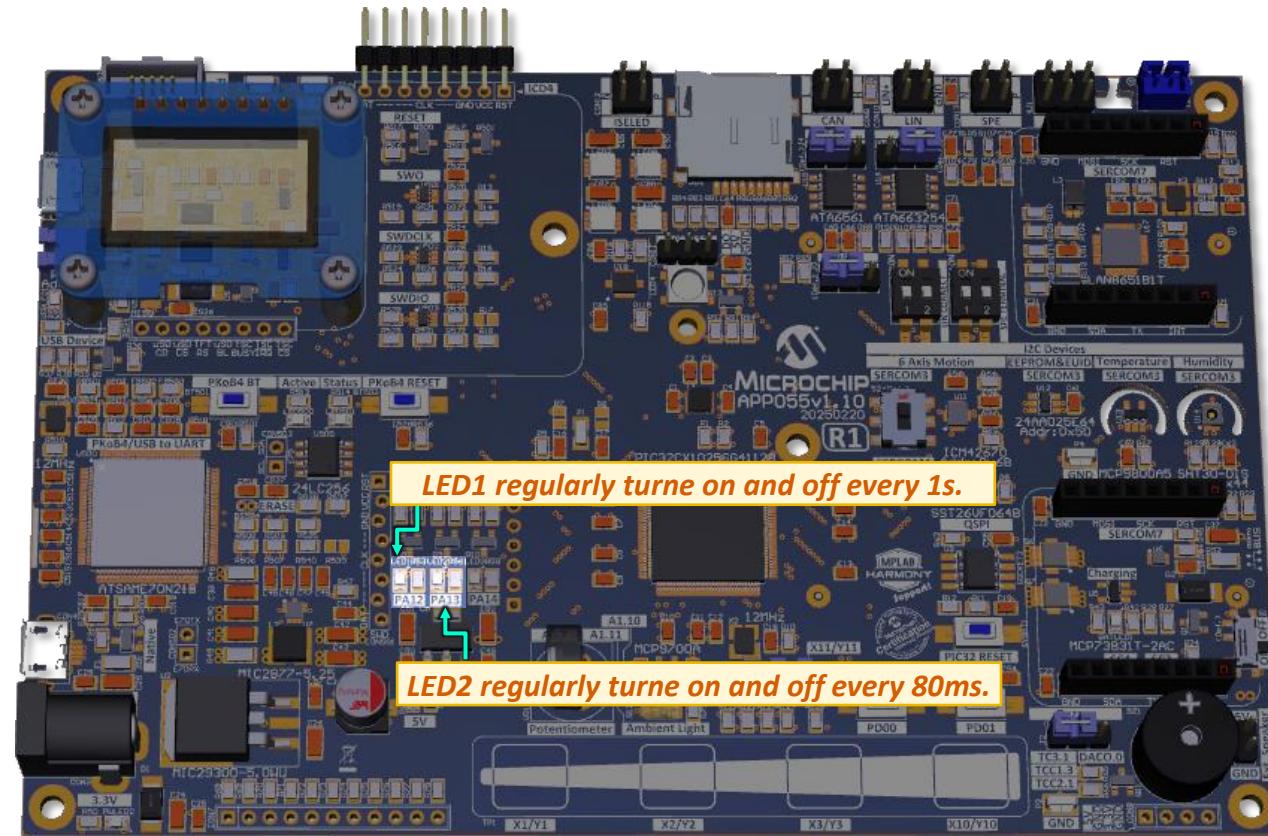
main.c

Step 7

-
- The screenshot displays the MPLAB X IDE interface. The top toolbar contains various icons for project management and execution. Two specific icons are highlighted with yellow boxes and red arrows: the 'Clean and Build Project' icon (a hammer and a wrench) and the 'Make and Program Device' icon (a green arrow pointing down to a chip). Below these icons, the 'Output' window is open, showing the results of the build and programming process. The output text indicates that the build was successful and the code was loaded onto the device.
- Clean and Build Project**
- Make and Program Device**
- BUILD SUCCESSFUL** Info: ../../src/config/default/PIC32CX1025SG41128.ld
- BUILD SUCCESSFUL (total time: 1s)**
- Loading code from C:/Exercises/Lab02_Timer_Polling_TC/Lab02_Timer_Polling_TC.X/dist/default/production/Lab02_Timer
 Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
 Loading completed

✂ Lab2 TC Timer Method Polling

Result



Interrupt - NVIC

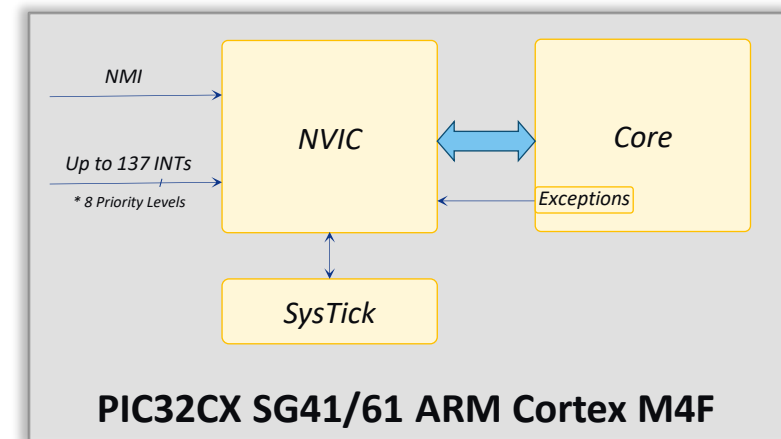
What's Interrupt

- An interrupt is a mechanism that temporarily halts the main program to respond to important or urgent events.
- When multiple interrupts occur at the same time, the system uses predefined priorities to decide which one to handle first. Higher-priority interrupts are serviced before lower-priority ones.
- For Example,
 - **Single Interrupt Example**
You are doing homework 📝. The phone rings 📞
you pause and answer it.
After the call you return to your homework.
 - **Multiple Interrupts Example**
You are reading a book 📖.
The gas alarm (🚒 high priority)
and doorbell (🔔 low priority) go off at the same time
you handle the alarm first.
Then answer the door
and return to reading.



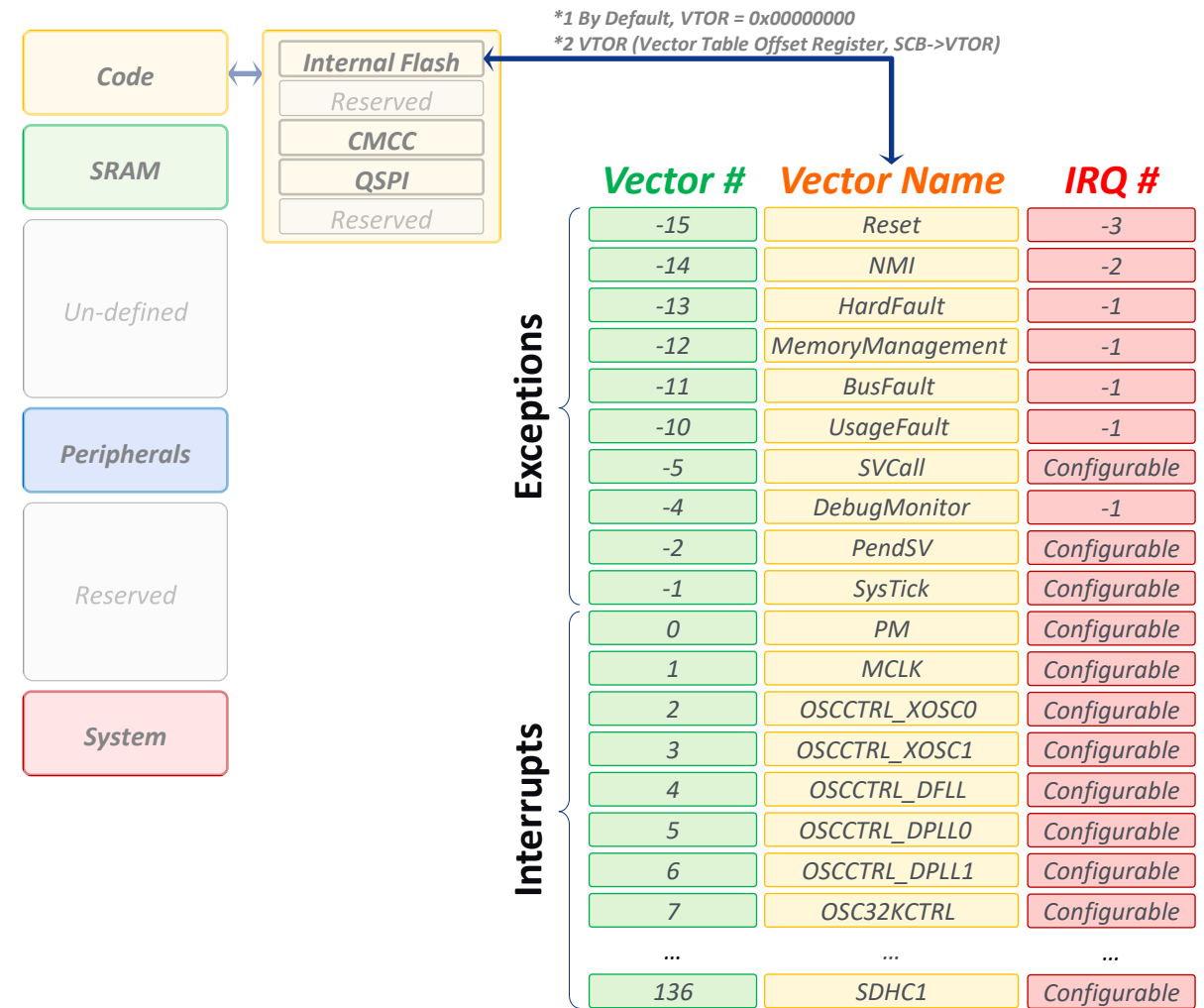
Nested Vector Interrupt Controller

- The Nested Vectored Interrupt Controller, NVIC of Cortex M4F provides an interface between interrupt sources (peripherals and external pins) and the core.
- **1 Non-maskable interrupt (NMI)**, **SysTick** and **137 physical interrupts** for PIC32-SG41/61 series.
- **8 priority Levels** for each interrupt source is software-configured, **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**.



Interrupt & Exception

- The Arm Cortex-M4F supports two types of event-driven mechanisms:
 - Exceptions (System-Level Events)
 - Exception Numbers: **-15** to **-1**
 - Managed by: **SCB** (System Control Block)
 - Interrupts (Peripheral-Level Events)
 - Exception Numbers: **0** to **136**
 - Managed by: **NVIC** (Nested Vector Interrupt Controller)
 - Fully configurable for Enable/Disable and Priority
 - All exceptions and interrupts use a vector table for their handlers.



Natural Priority

- Natural Priority is a built-in, fixed priority scheme based on interrupt numbers. It is used when two or more interrupts have the same software-configured priority.
- Lower interrupt numbers (**Vector #**) have higher natural priority.
- Applies to both,
Exceptions (System-Level Events)
Interrupts (Peripheral-Level Events)

Natural Priority Order



	Vector #	Vector Name	IRQ #
Exceptions	-15	Reset	-3
	-14	NMI	-2
	-13	HardFault	-1
	-12	MemoryManagement	-1
	-11	BusFault	-1
	-10	UsageFault	-1
	-5	SVCall	Configurable
	-4	DebugMonitor	-1
	-2	PendSV	Configurable
	-1	SysTick	Configurable
Interrupts	0	PM	Configurable
	1	MCLK	Configurable
	2	OSCCTRL_XOSC0	Configurable
	3	OSCCTRL_XOSC1	Configurable
	4	OSCCTRL_DFLL	Configurable
	5	OSCCTRL_DPLL0	Configurable
	6	OSCCTRL_DPLL1	Configurable
	7	OSC32KCTRL	Configurable
	...	...	...
	136	SDHC1	Configurable

Interrupt Arbitration

- When multiple interrupts are pending at the same time, uses a two-level arbitration mechanism to decide which interrupt to handle,

1. Configurable Priority

Lower software-configurable priority values mean higher priority (priority 0 is highest, 0 - 7)

2. Natural Priority

If interrupts have the same programmed priority, uses natural priority as a tiebreaker.

Lower interrupt numbers have higher natural priority.

- Arbitration Example,**

TC0 and TC1 interrupts are pending at the same time, TC0(**107**) have higher priority than TC1(**108**) if both have the same software-configurable priority(**0**).

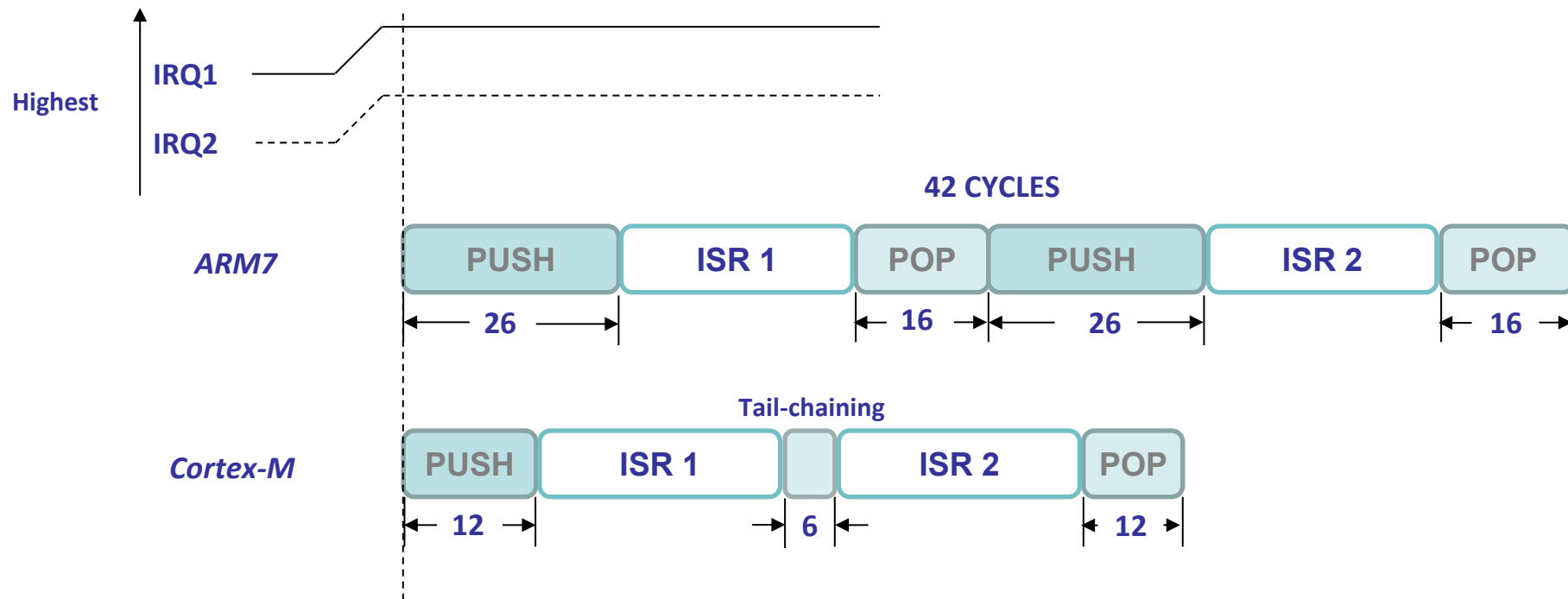
Higher Priority



	Vector #	Vector Name	IRQ #
Exceptions	-15	Reset	-3
	-14	NMI	-2
	-13	HardFault	-1
	-12	MemoryManagement	-1
	-11	BusFault	-1
	-10	UsageFault	-1
	-5	SVCall	Configurable
	-4	DebugMonitor	-1
	-2	PendSV	Configurable
	-1	SysTick	Configurable
	0	PM	Configurable
	1	MCLK	Configurable
Interrupts	...	...	...
	107	TC0	0
	108	TC1	0
	109	TC2	Configurable
	110	TC3	Configurable
	111	TC4	Configurable
	...	...	...
	136	SDHC1	Configurable

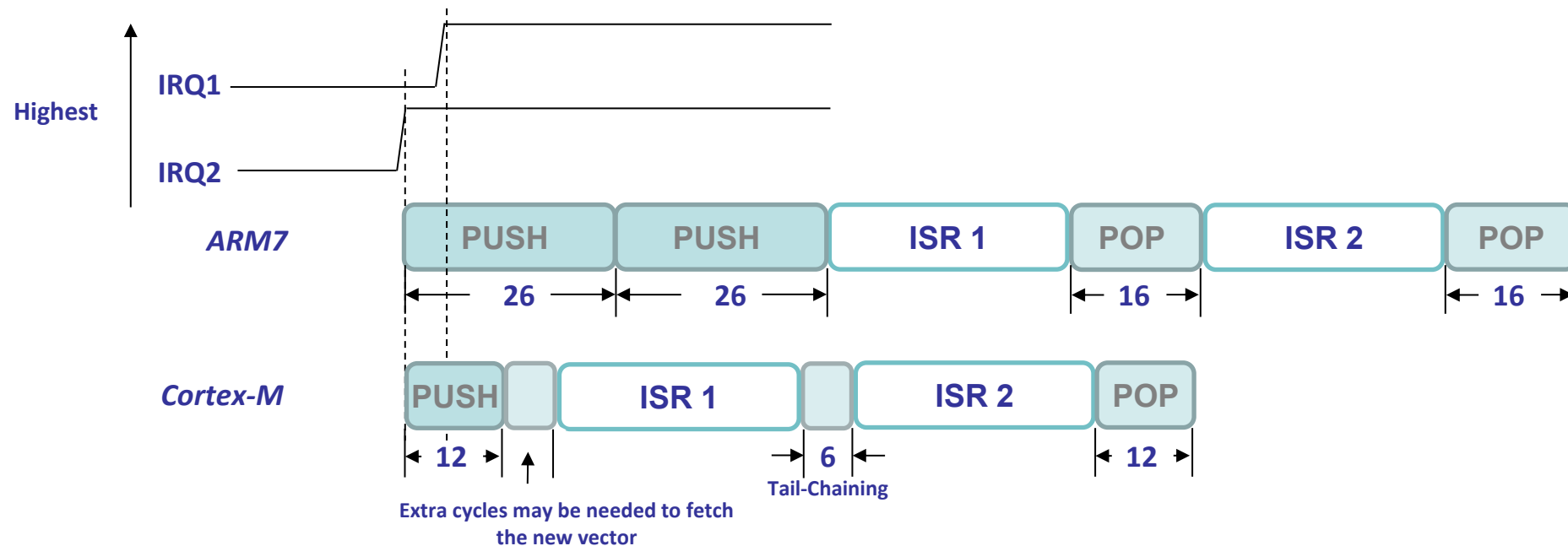
Tail Chaining in the NVIC

- 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.



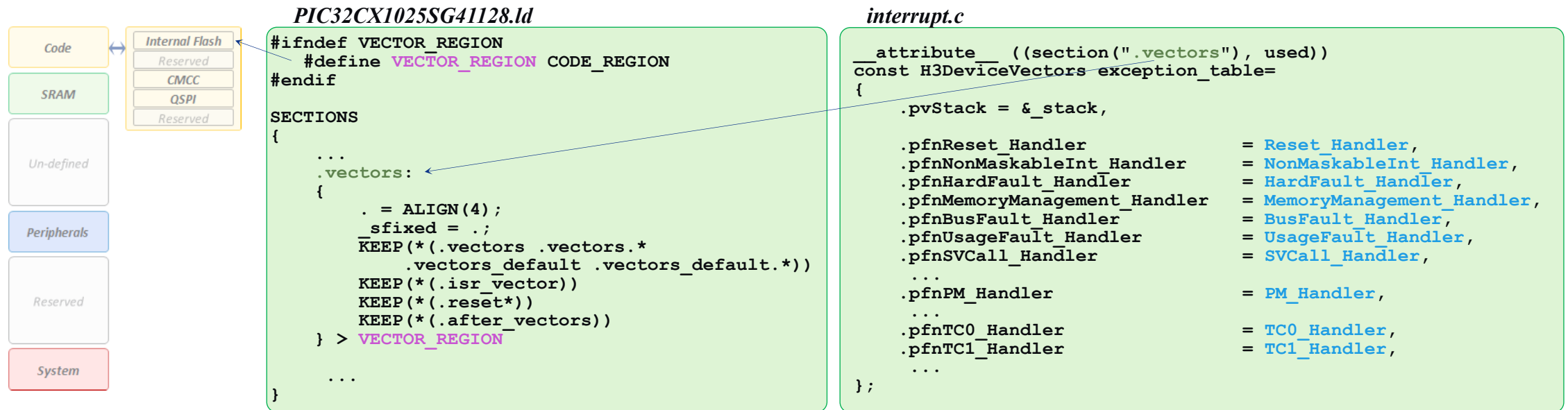
Late Arriving in the NVIC

- 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.



Interrupt Vector Table, IVT

- The IVT is defined in the *interrupt.c* source file.
- It uses the GCC section attribute `__attribute__((section(".vectors"), used))` to place the vector table in a dedicated section named `.vectors`.
- The linker script defines the placement of the `.vectors` section at a specific memory address, typically at `0x00000000`.



Interrupt Service Routine, ISR

- An ISR is a special function automatically executed in response to a hardware interrupt. When an interrupt occurs, the processor uses the IVT to locate and jump to the ISR. The ISR performs **low-level handling, such as clearing interrupt flags**, and checks whether a callback function is registered. If a callback is set, the ISR invokes it to execute user-defined application.

interrupt.c

```
__attribute__((section(".vectors"), used))
const H3DeviceVectors exception_table=
{
    .pvStack = &_stack,
    ...
    .pfnTC0_Handler = TC0_TimerInterruptHandler,
    .pfnTC1_Handler = TC1_Handler,
    ...
};
```

plib_tc0.c

```
/* Timer Interrupt handler */
void __attribute__((used)) TC0_TimerInterruptHandler( void )
{
    if (TC0_REGS->COUNT16.TC_INTENSET != 0U)
    {
        TC_TIMER_STATUS status;
        status = (TC_TIMER_STATUS) TC0_REGS->COUNT16.TC_INTFLAG;
        /* Clear interrupt flags */
        TC0_REGS->COUNT16.TC_INTFLAG = (uint8_t)TC_INTFLAG_Msk;
        if((TC0_CallbackObject.callback != NULL) &&
            (status != TC_TIMER_STATUS_NONE))
        {
            uintptr_t context = TC0_CallbackObject.context;
            TC0_CallbackObject.callback(status, context);
        }
    }
}
```

```
/* Timer Interrupt handler */
void User_CallBack( void )
{
    ...
}
```

Callback

- Callbacks help decouple application logic from hardware-level tasks, allowing the user focus on functionality while the ISR handles low-level operations such as clearing interrupt flags, thereby reducing system complexity and enabling easier maintenance.
- MCC Harmony supports a callback mechanism, Callback can be registered via the *xxxx_CallbackRegister()* function and executed when the event occurs.

main.c

```
/* Timer Interrupt handler */
void User_Callback( void )
{
    /* user-defined application */
}

int main ( void )
{
    SYS_Initialize ( NULL );
    TC0_TimerCallbackRegister
        (User_Callback , ( uintptr_t ) 0 );
    while ( true )
        SYS_Tasks ( );
    return ( EXIT_FAILURE );
}
```

```
/* Register callback function */
void TC0_TimerCallbackRegister( , )
{
    TC0_CallbackObject.callback = callback;
    TC0_CallbackObject.context = context;
}
```

plib_tc0.c

```
/* Timer Interrupt handler */
void __attribute__((used)) TC0_TimerInterruptHandler( void )
{
    if (TC0_REGS->COUNT16.TC_INTENSET != 0U)
    {
        TC_TIMER_STATUS status;
        status = (TC_TIMER_STATUS) TC0_REGS->COUNT16.TC_INTFLAG;
        /* Clear interrupt flags */
        TC0_REGS->COUNT16.TC_INTFLAG = (uint8_t)TC_INTFLAG_Msk;
        if((TC0_CallbackObject.callback != NULL) &&
            (status != TC_TIMER_STATUS_NONE))
        {
            uintptr_t context = TC0_CallbackObject.context;
            TC0_CallbackObject.callback(status, context);
        }
    }
}
```


Flowchart of MCC Harmony Callback

- Define and register the callback during initialization; it will be automatically executed when the event occurs.

Initialize

Define Callback

```
/* Timer Interrupt handler */
void User_Callback( void )
{
    /* user-defined application */
}
```

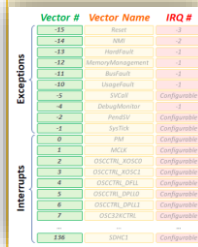
Register Callback

```
int main ( void )
{
    SYS_Initialize ( NULL );
    TC0_TimerCallbackRegister
        (User_Callback , ( uintptr_t ) 0 );

    while ( true )
        SYS_Tasks ( );
}
return ( EXIT_FAILURE );
```

Event Occurred

Check IVT & Jump to ISR

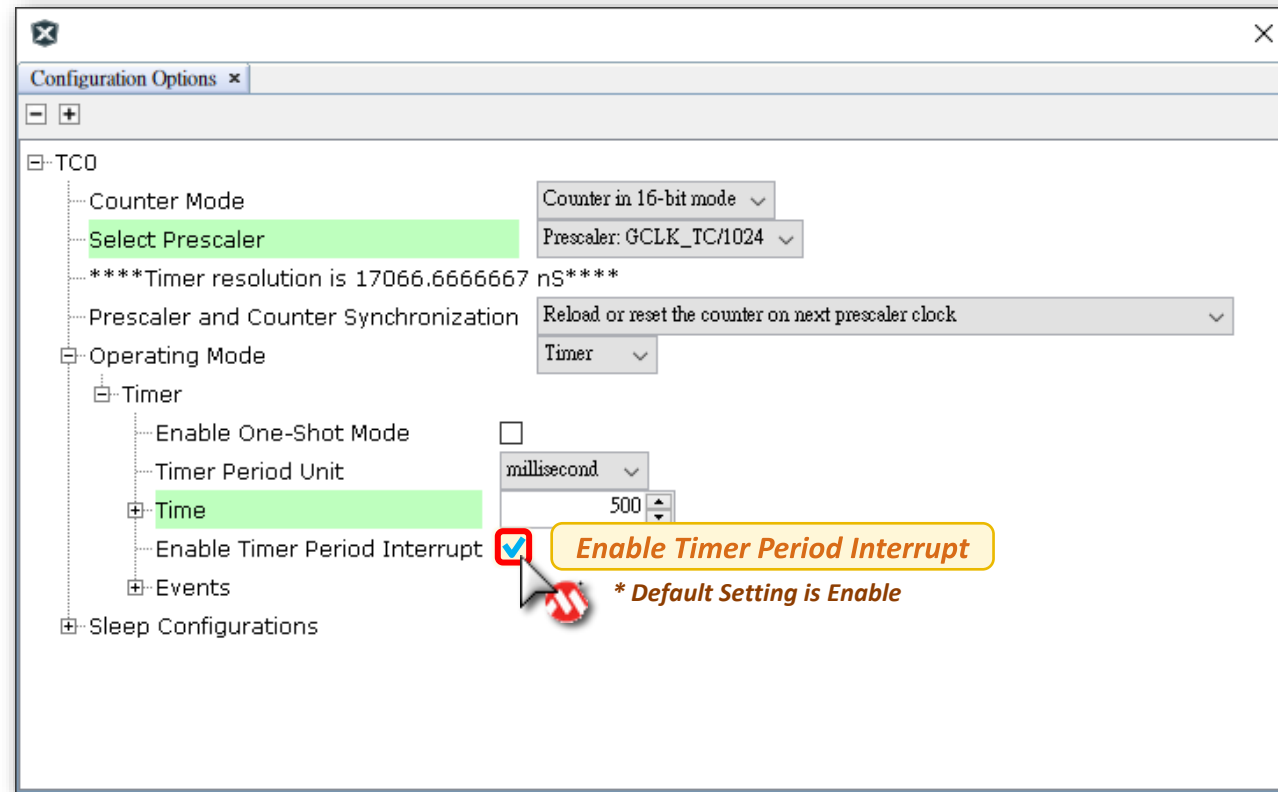


Vector #	Vector Name	IRQ #
0	Reset	0
1	NMI	1
2	HardFault	2
3	MemoryManagement	3
4	BusFault	4
5	UsageFault	5
6	SVCall	6
7	DebugMonitor	7
8	PendSV	8
9	SysTick	9
10	WFI	10
11	WFE	11
12	TC0_TM0_R0SC	12
13	TC0_TM0_R1SC	13
14	TC0_TM0_R2SC	14
15	TC0_TM0_R3SC	15
16	TC0_TM0_R4SC	16
17	TC0_TM0_R5SC	17
18	TC0_TM0_R6SC	18
19	TC0_TM0_R7SC	19
20	TC0_TM0_R8SC	20
21	TC0_TM0_R9SC	21
22	TC0_TM0_R10SC	22
23	TC0_TM0_R11SC	23
24	TC0_TM0_R12SC	24
25	TC0_TM0_R13SC	25
26	TC0_TM0_R14SC	26
27	TC0_TM0_R15SC	27
28	TC0_TM0_R16SC	28
29	TC0_TM0_R17SC	29
30	TC0_TM0_R18SC	30
31	TC0_TM0_R19SC	31
32	TC0_TM0_R20SC	32
33	TC0_TM0_R21SC	33
34	TC0_TM0_R22SC	34
35	TC0_TM0_R23SC	35
36	TC0_TM0_R24SC	36
37	TC0_TM0_R25SC	37
38	TC0_TM0_R26SC	38
39	TC0_TM0_R27SC	39
40	TC0_TM0_R28SC	40
41	TC0_TM0_R29SC	41
42	TC0_TM0_R30SC	42
43	TC0_TM0_R31SC	43
44	TC0_TM0_R32SC	44
45	TC0_TM0_R33SC	45
46	TC0_TM0_R34SC	46
47	TC0_TM0_R35SC	47
48	TC0_TM0_R36SC	48
49	TC0_TM0_R37SC	49
50	TC0_TM0_R38SC	50
51	TC0_TM0_R39SC	51
52	TC0_TM0_R40SC	52
53	TC0_TM0_R41SC	53
54	TC0_TM0_R42SC	54
55	TC0_TM0_R43SC	55
56	TC0_TM0_R44SC	56
57	TC0_TM0_R45SC	57
58	TC0_TM0_R46SC	58
59	TC0_TM0_R47SC	59
60	TC0_TM0_R48SC	60
61	TC0_TM0_R49SC	61
62	TC0_TM0_R50SC	62
63	TC0_TM0_R51SC	63
64	TC0_TM0_R52SC	64
65	TC0_TM0_R53SC	65
66	TC0_TM0_R54SC	66
67	TC0_TM0_R55SC	67
68	TC0_TM0_R56SC	68
69	TC0_TM0_R57SC	69
70	TC0_TM0_R58SC	70
71	TC0_TM0_R59SC	71
72	TC0_TM0_R60SC	72
73	TC0_TM0_R61SC	73
74	TC0_TM0_R62SC	74
75	TC0_TM0_R63SC	75
76	TC0_TM0_R64SC	76
77	TC0_TM0_R65SC	77
78	TC0_TM0_R66SC	78
79	TC0_TM0_R67SC	79
80	TC0_TM0_R68SC	80
81	TC0_TM0_R69SC	81
82	TC0_TM0_R70SC	82
83	TC0_TM0_R71SC	83
84	TC0_TM0_R72SC	84
85	TC0_TM0_R73SC	85
86	TC0_TM0_R74SC	86
87	TC0_TM0_R75SC	87
88	TC0_TM0_R76SC	88
89	TC0_TM0_R77SC	89
90	TC0_TM0_R78SC	90
91	TC0_TM0_R79SC	91
92	TC0_TM0_R80SC	92
93	TC0_TM0_R81SC	93
94	TC0_TM0_R82SC	94
95	TC0_TM0_R83SC	95
96	TC0_TM0_R84SC	96
97	TC0_TM0_R85SC	97
98	TC0_TM0_R86SC	98
99	TC0_TM0_R87SC	99
100	TC0_TM0_R88SC	100
101	TC0_TM0_R89SC	101
102	TC0_TM0_R90SC	102
103	TC0_TM0_R91SC	103
104	TC0_TM0_R92SC	104
105	TC0_TM0_R93SC	105
106	TC0_TM0_R94SC	106
107	TC0_TM0_R95SC	107
108	TC0_TM0_R96SC	108
109	TC0_TM0_R97SC	109
110	TC0_TM0_R98SC	110
111	TC0_TM0_R99SC	111
112	TC0_TM0_R100SC	112
113	TC0_TM0_R101SC	113
114	TC0_TM0_R102SC	114
115	TC0_TM0_R103SC	115
116	TC0_TM0_R104SC	116
117	TC0_TM0_R105SC	117
118	TC0_TM0_R106SC	118
119	TC0_TM0_R107SC	119
120	TC0_TM0_R108SC	120
121	TC0_TM0_R109SC	121
122	TC0_TM0_R110SC	122
123	TC0_TM0_R111SC	123
124	TC0_TM0_R112SC	124
125	TC0_TM0_R113SC	125
126	TC0_TM0_R114SC	126
127	TC0_TM0_R115SC	127
128	TC0_TM0_R116SC	128
129	TC0_TM0_R117SC	129
130	TC0_TM0_R118SC	130
131	TC0_TM0_R119SC	131
132	TC0_TM0_R120SC	132
133	TC0_TM0_R121SC	133
134	TC0_TM0_R122SC	134
135	TC0_TM0_R123SC	135
136	TC0_TM0_R124SC	136
137	TC0_TM0_R125SC	137
138	TC0_TM0_R126SC	138
139	TC0_TM0_R127SC	139
140	TC0_TM0_R128SC	140
141	TC0_TM0_R129SC	141
142	TC0_TM0_R130SC	142
143	TC0_TM0_R131SC	143
144	TC0_TM0_R132SC	144
145	TC0_TM0_R133SC	145
146	TC0_TM0_R134SC	146
147	TC0_TM0_R135SC	147
148	TC0_TM0_R136SC	148
149	TC0_TM0_R137SC	149
150	TC0_TM0_R138SC	150
151	TC0_TM0_R139SC	151
152	TC0_TM0_R140SC	152
153	TC0_TM0_R141SC	153
154	TC0_TM0_R142SC	154
155	TC0_TM0_R143SC	155
156	TC0_TM0_R144SC	156
157	TC0_TM0_R145SC	157
158	TC0_TM0_R146SC	158
159	TC0_TM0_R147SC	159
160	TC0_TM0_R148SC	160
161	TC0_TM0_R149SC	161
162	TC0_TM0_R150SC	162
163	TC0_TM0_R151SC	163
164	TC0_TM0_R152SC	164
165	TC0_TM0_R153SC	165
166	TC0_TM0_R154SC	166
167	TC0_TM0_R155SC	167
168	TC0_TM0_R156SC	168
169	TC0_TM0_R157SC	169
170	TC0_TM0_R158SC	170
171	TC0_TM0_R159SC	171
172	TC0_TM0_R160SC	172
173	TC0_TM0_R161SC	173
174	TC0_TM0_R162SC	174
175	TC0_TM0_R163SC	175
176	TC0_TM0_R164SC	176
177	TC0_TM0_R165SC	177
178	TC0_TM0_R166SC	178
179	TC0_TM0_R167SC	179
180	TC0_TM0_R168SC	180
181	TC0_TM0_R169SC	181
182	TC0_TM0_R170SC	182
183	TC0_TM0_R171SC	183
184	TC0_TM0_R172SC	184
185	TC0_TM0_R173SC	185
186	TC0_TM0_R174SC	186
187	TC0_TM0_R175SC	187
188	TC0_TM0_R176SC	188
189	TC0_TM0_R177SC	189
190	TC0_TM0_R178SC	190
191	TC0_TM0_R179SC	191
192	TC0_TM0_R180SC	192
193	TC0_TM0_R181SC	193
194	TC0_TM0_R182SC	194
195	TC0_TM0_R183SC	195
196	TC0_TM0_R184SC	196
197	TC0_TM0_R185SC	197
198	TC0_TM0_R186SC	198
199	TC0_TM0_R187SC	199
200	TC0_TM0_R188SC	200
201	TC0_TM0_R189SC	201
202	TC0_TM0_R190SC	202
203	TC0_TM0_R191SC	203
204	TC0_TM0_R192SC	204
205	TC0_TM0_R193SC	205
206	TC0_TM0_R194SC	206
207	TC0_TM0_R195SC	207
208	TC0_TM0_R196SC	208
209	TC0_TM0_R197SC	209
210	TC0_TM0_R198SC	210
211	TC0_TM0_R199SC	211
212	TC0_TM0_R200SC	212
213	TC0_TM0_R201SC	213
214	TC0_TM0_R202SC	214
215	TC0_TM0_R203SC	215
216	TC0_TM0_R204SC	216
217	TC0_TM0_R205SC	217
218	TC0_TM0_R206SC	218
219	TC0_TM0_R207SC	219
220	TC0_TM0_R208SC	220
221	TC0_TM0_R209SC	221
222	TC0_TM0_R210SC	222
223	TC0_TM0_R211SC	223
224	TC0_TM0_R212SC	224
225	TC0_TM0_R213SC	225
226	TC0_TM0_R214SC	226
227	TC0_TM0_R215SC	227
228	TC0_TM0_R216SC	228
229	TC0_TM0_R217SC	229
230	TC0_TM0_R218SC	230
231	TC0_TM0_R219SC	231
232	TC0_TM0_R220SC	232
233	TC0_TM0_R221SC	233
234	TC0_TM0_R222SC	234
235	TC0_TM0_R223SC	235
236	TC0_TM0_R224SC	236
237	TC0_TM0_R225SC	237
238	TC0_TM0_R226SC	238
239	TC0_TM0_R227SC	239
240	TC0_TM0_R228SC	240
241	TC0_TM0_R229SC	241
242	TC0_TM0_R230SC	242
243	TC0_TM0_R231SC	243
244	TC0_TM0_R232SC	244
245	TC0_TM0_R233SC	245
246	TC0_TM0_R234SC	246
247	TC0_TM0_R235SC	247
248	TC0_TM0_R236SC	248
249	TC0_TM0_R237SC	249
250	TC0_TM0_R238SC	250
251	TC0_TM0_R239SC	251
252	TC0_TM0_R240SC	252
253	TC0_TM0_R241SC	253
254	TC0_TM0_R242SC	254
255	TC0_TM0_R243SC	255
256	TC0_TM0_R244SC	256
257	TC0_TM0_R245SC	257
258	TC0_TM0_R246SC	258
259	TC0_TM0_R247SC	259
260	TC0_TM0_R248SC	260
261	TC0_TM0_R249SC	261
262	TC0_TM0_R250SC	262
263	TC0_TM0_R251SC	263
264	TC0_TM0_R252SC	264
265	TC0_TM0_R253SC	265
266	TC0_TM0_R254SC	266
267	TC0_TM0_R255SC	267
268	TC0_TM0_R256SC	268
269	TC0_TM0_R257SC	269
270	TC0_TM0_R258SC	270
271	TC0_TM0_R259SC	271
272	TC0_TM0_R260SC	272
273	TC0_TM0_R261SC	273
274	TC0_TM0_R262SC	274
275	TC0_TM0_R263SC	275
276	TC0_TM0_R264SC	276
277	TC0_TM0_R265SC	277
278	TC0_TM0_R266SC	278
279	TC0_TM0_R267SC	279
280	TC0_TM0_R268SC	280
281	TC0_TM0_R269SC	281
282	TC0_TM0_R270SC	282
283	TC0_TM0_R271SC	283
284	TC0_TM0_R272SC	284
285	TC0_TM0_R273SC	285
286	TC0_TM0_R274SC	286
287	TC0_TM0_R275SC	287
288	TC0_TM0_R276SC	288
289	TC0_TM0_R277SC	289
290	TC0_TM0_R278SC	290
291	TC0_TM0_R279SC	291
292	TC0_TM0_R280SC	292
293	TC0_TM0_R281SC	293
294	TC0_TM0_R282SC	294
295	TC0_TM0_R283SC	295
296	TC0_TM0_R284SC	296
297	TC0_TM0_R285SC	297
298	TC0_TM0_R286SC	298
299	TC0_TM0_R287SC	299
300	TC0_TM0_R288SC	300
301	TC0_TM0_R289SC	301
302	TC0_TM0_R290SC	302
303	TC0_TM0_R291SC	303
304	TC0_TM0_R292SC	304
305	TC0_TM0_R293SC	305
306	TC0_TM0_R294SC	306
307	TC0_TM0_R295SC	307
308	TC0_TM0_R296SC	308
309	TC0_TM0_R297SC	309
310	TC0_TM0_R298SC	310
311	TC0_TM0_R299SC	311
312	TC0_TM0_R300SC	312
313	TC0_TM0_R301SC	313
314	TC0_TM0_R302SC	314
315	TC0_TM0_R303SC	315
316	TC0_TM0_R304SC	316
317	TC0_TM0_R305SC	317
318	TC0_TM0_R306SC	318
319	TC0_TM0_R307SC	319
320	TC0_TM0_R308SC	320
321	TC0_TM0_R309SC	321
322	TC0_TM0_R310SC	322
323	TC0_TM0_R311SC	323
324	TC0_TM0_R312SC	324
325	TC0_TM0_R313SC	325
326	TC0_TM0_R314SC	326
327	TC0_TM0_R315SC	327
328	TC0_TM0_R316SC	328
329	TC0_TM0_R317SC	329
330	TC0_TM0_R318SC	330
331	TC0_TM0_R319SC	331
332	TC0_TM0_R320SC	332
333	TC0_TM0_R321SC	333
334	TC0_TM0_R322SC	334
335	TC0_TM0_R323SC	335
336	TC0_TM0_R324SC	336
337	TC0_TM0_R325SC	337
338	TC0_TM0_R326SC	338
339	TC0_TM0_R327SC	339
340	TC0_TM0_R328SC	340
341	TC0_TM0_R329SC	341
342	TC0_TM0_R330SC	342
343	TC0_TM0_R331SC	343
344	TC0_TM0_R332SC	344
345	TC0_TM0_R333SC	345
346	TC0_TM0_R334SC	346
347	TC0_TM0_R335SC	347
348	TC0_TM0_R336SC	348
349	TC0_TM0_R337SC	349
350	TC0_TM0_R338SC	350
351	TC0_TM0_R339SC	351
352	TC0_TM0_R340SC	352
353	TC0_TM0_R341SC	353
354	TC0_TM0_R342SC	354
355	TC0_TM0_R343SC	355
356	TC0_TM0_R344SC	356
357	TC0_TM0_R345SC	357
358	TC0_TM0_R346SC	358
359	TC0_TM0_R347SC	3

Timer by Interrupt Callback - TC

Interrupt Callback in MCC Harmony

- MCC Harmony provides a simple way to enable interrupt callbacks for peripherals, just one click in the configuration window.



Priority Settings in MCC Harmony

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

The screenshot shows the MPLAB X IDE interface. The 'Project Graph' window is active, and the 'Plugins' menu is open, with 'NVIC Configuration' selected. The main window displays the NVIC Configuration table, which lists various peripherals and their interrupt settings. A red box highlights the entries for TC0 (Timer Counter 0) and TC1 (Timer Counter 1).

Peripheral	Enabled	Priority	Handler
TCC4_MC0 (TCC Match/Compare 0)	☐	7	TCC4_MC0_Handler
TCC4_MC1 (TCC Match/Compare 1)	☐	7	TCC4_MC1_Handler
TC0 (Timer Counter 0)	☒	7	TC0_TimerInterruptHandler
TC1 (Timer Counter 1)	☐	7	TC1_Handler
TC2 (Timer Counter 2)	☐	7	TC2_Handler
TC3 (Timer Counter 3)	☐	7	TC3_Handler
TC4 (Timer Counter 4)	☐	7	TC4_Handler

107	TC0 (Timer Counter 0)	☒	7	TC0_TimerInterruptHandler
108	TC1 (Timer Counter 1)	☐	7	TC1_Handler

PDEC_OTHER (Position Decoder)	☐	7	PDEC_OTHER_Handler
PDEC_MC0 (PDEC Match/Compare 0)	☐	7	PDEC_MC0_Handler
PDEC_MC1 (PDEC Match/Compare 1)	☐	7	PDEC_MC1_Handler

TC Interrupt

TC n Interrupt Interface Functions

- **TC n _TimerInitialize** : Initializes the TC n timer module in 16-bit mod.
 - **TC n _TimerStart** : Start the TC n timer.
 - **TC n _TimerStop** : Stop the TC n timer.
 - **TC n _TimerFrequencyGet** : Returns the operating frequency of the TC n timer.
 - **TC n _Timer16bitPeriodSet** : Set the TC n period value for 16-bit mode.
 - **TC n _Timer16bitCounterSet** : Set the TC n counter value in 16-bit mode.
 - **TC n _Timer16bitPeriodGet** : Get the TC n period value for 16-bit mode.
 - **TC n _Timer16bitCounterGet** : Get the TC n counter value in 16-bit mode.
 - ~~• **TC n _TimerPeriodHasExpired** : Get TC n period interrupt flag is set.~~
 - **TC n _TimerCallbackRegister** : Registers a callback function to be called on TC n interrupt events.
- One new function than polling mode.*

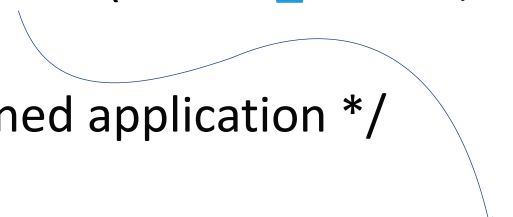
TC Interrupt

TC Interface Functions

void TCn_TimerCallbackRegister(TC_TIMER_CALLBACK *callback*, uintptr_t *context*)

- **Description:**
 - Registers a callback function to be called on TCn interrupt events
- **Parameter:**
 - *callback* : Pointer to the callback function to be called on timer event.
 - *context* : A user-defined value (typically a pointer) passed to the callback when invoked.
- **Example :**

```
void User_Callback( uint32_t status, uintptr_t context )  
{  
    /* User-defined application */  
}  
TC0_TimerCallbackRegister( User_Callback, (uintptr_t) 0 );
```



Lab3 Timer Interrupt Callback TC

Lab3 Timer Interrupt Callback TC

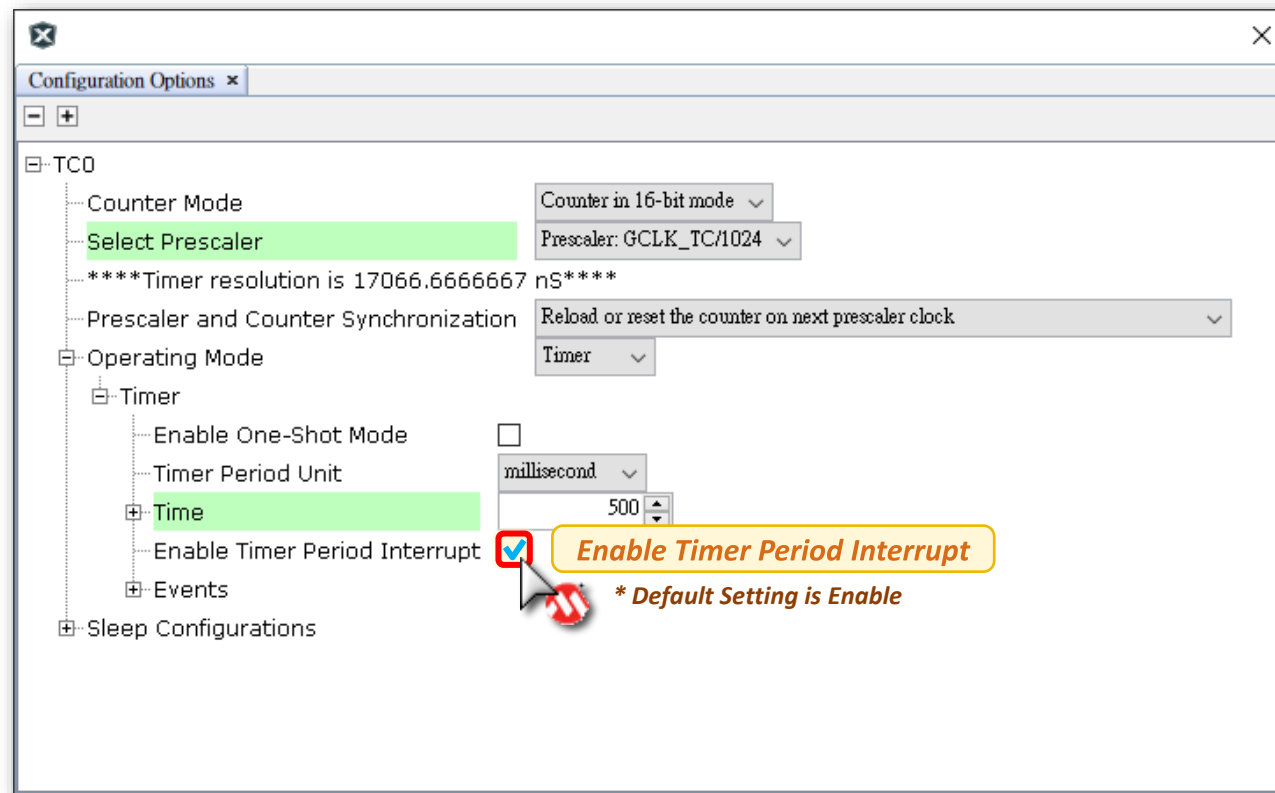
- Try configure **TC0** and **TC1** to use **interrupt callback** method.
- **LED1/2** will toggle in **TC0/TC1 interrupt Callback** while **Timer Period Expired** interrupt.
 - The **TC0** setup to **Timer** and **interrupt method** and keep the Timer period (500ms).
 - The **TC1** setup to **Timer** and **interrupt method** and keep the Timer period (40ms).

Let's go !

⚙️ Lab3 Timer Interrupt Callback TC

Step 1

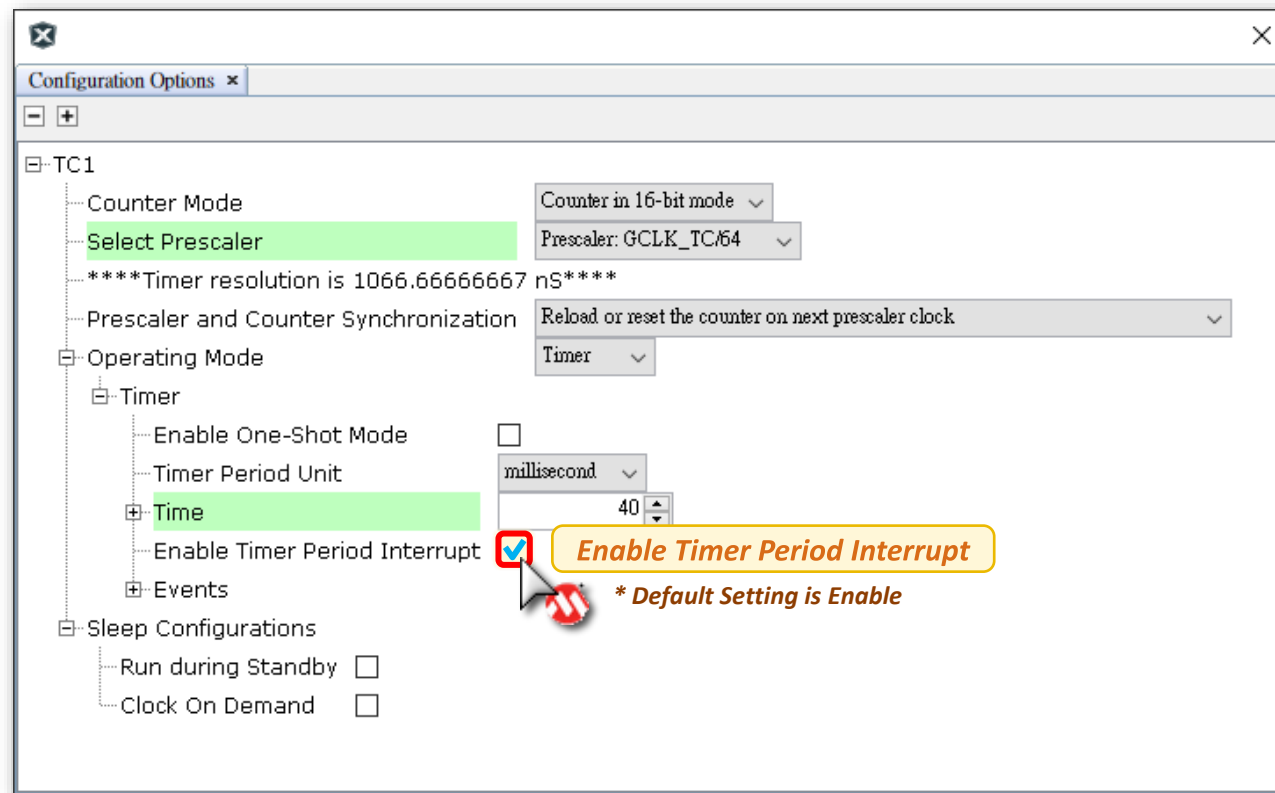
- Enable TC0 **Timer Period Interrupt** in configuration.



⚙️ Lab3 Timer Interrupt Callback TC

Step 2

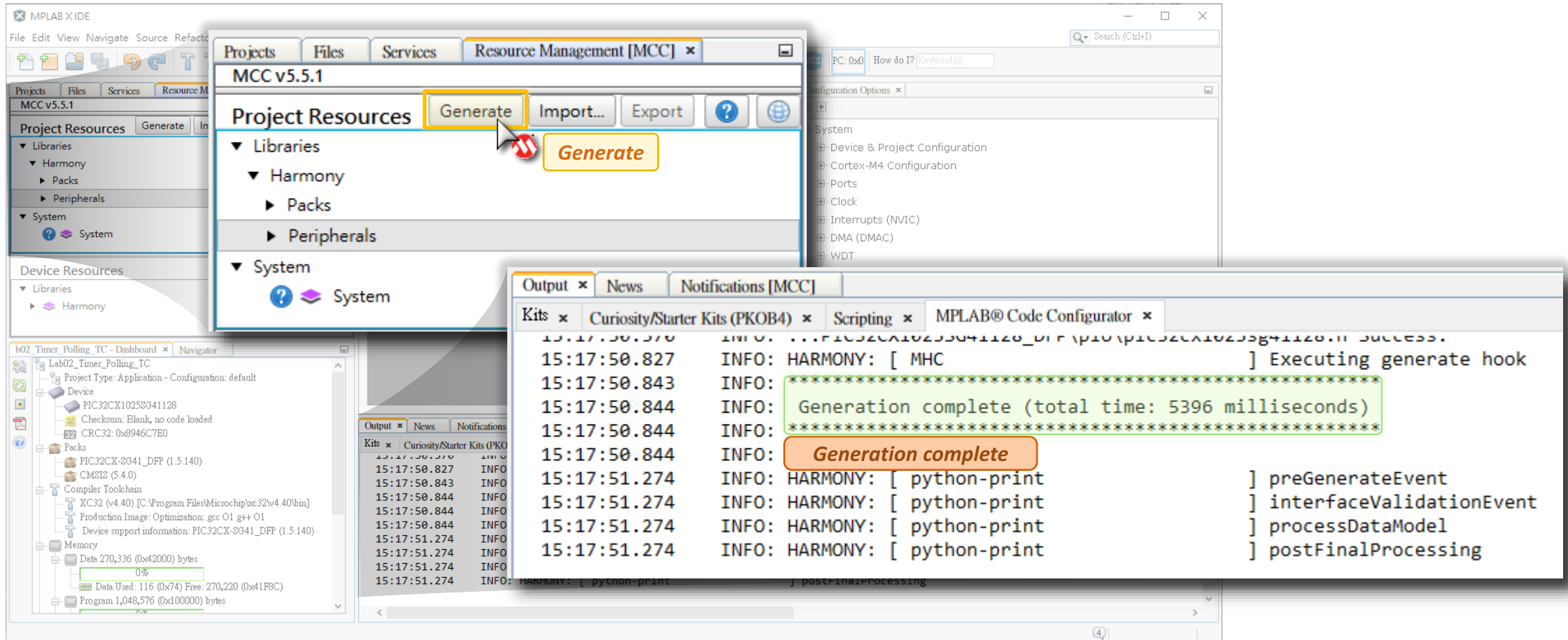
- Enable TC1 **Timer Period Interrupt** in configuration.



⚙️ Lab3 Timer Interrupt Callback TC

Step 3

- Click on **Generate** button to generate your code after configuration.



The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC] v5.5.1' window is open, showing the 'Project Resources' tab. The 'Generate' button is highlighted with a yellow box and a red arrow. Below it, the 'System' resource is selected. The 'Output' window is also open, showing the 'MPLAB® Code Configurator' tab. The output log shows the following messages:

```
15:17:50.827 INFO: ... PIC32CX1025SG41128_DFP (p10 (pic32cx1025sg41128.1)) Success.  
15:17:50.827 INFO: HARMONY: [ MHC ] Executing generate hook  
15:17:50.843 INFO:   
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)  
15:17:50.844 INFO:   
15:17:50.844 INFO: Generation complete  
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent  
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent  
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel  
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

Lab3 Timer Interrupt Callback TC

Step 4

- Open **main.c** and finish **TODO3.01** to comment out or remove the code of previous lab.

```
int main (void)
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    TC0_TimerStart();
    TC1_TimerStart();

    while( true )
    {
        SYS_Tasks ( );

        // TODO 3.01
        // if ( TC0_TimerPeriodHasExpired() )
        // {
        //     LED1_Toggle();
        // }

        // if ( TC1_TimerPeriodHasExpired() )
        // {
        //     LED2_Toggle();
        // }

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

    return ( EXIT_FAILURE );
}
```

main.c

Lab3 Timer Interrupt Callback TC

Step 5

- Finish **TODO3.02** to **3.04** for define Custom Software Flags and Callback functions.

```
...
#include "definitions.h"                // SYS function prototypes

uint32_t i = 0;

// TODO 3.02
volatile uint8_t TC0_HasExpired = 0;
volatile uint8_t TC1_HasExpired = 0;

// TODO 3.03
void TC0_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if (status & TC_INTFLAG_OVF_Msk)
    {
        TC0_HasExpired = 1;
    }
}

// TODO 3.04
void TC1_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if (status & TC_INTFLAG_OVF_Msk)
    {
        TC1_HasExpired = 1;
    }
}

int main ( void )
{
```

main.c

Lab3 Timer Interrupt Callback TC

Step 6

- Finish **TODO3.05** to **3.06** for register callback functions and start TC0 and TC1.

```
// TODO 3.03
void TC0_TimerExpired(TC_TIMER_STATUS status, uintptr_t context){}

// TODO 3.04
void TC1_TimerExpired(TC_TIMER_STATUS status, uintptr_t context){}

int main(void)
{
    /* Initialize all modules */
    SYS_Initialize(NULL);

    // TODO 3.05
    TC0_TimerCallbackRegister(TC0_TimerExpired, (uintptr_t) NULL);
    TC0_TimerStart();

    // TODO 3.06
    TC1_TimerCallbackRegister(TC1_TimerExpired, (uintptr_t) NULL);
    TC1_TimerStart();

    while (true)
    {
        SYS_Tasks();

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

    return ( EXIT_FAILURE );
}
```

main.c

Lab3 Timer Interrupt Callback TC

Step 7

- Finish **TODO3.07** to **3.08** for check and clear Custom Software Flags and toggle LEDs

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

        // TODO 3.07
        if (TC0_HasExpired)
        {
            TC0_HasExpired = 0;
            LED1_Toggle();
        }

        // TODO 3.08
        if (TC1_HasExpired)
        {
            TC1_HasExpired = 0;
            LED2_Toggle();
        }

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

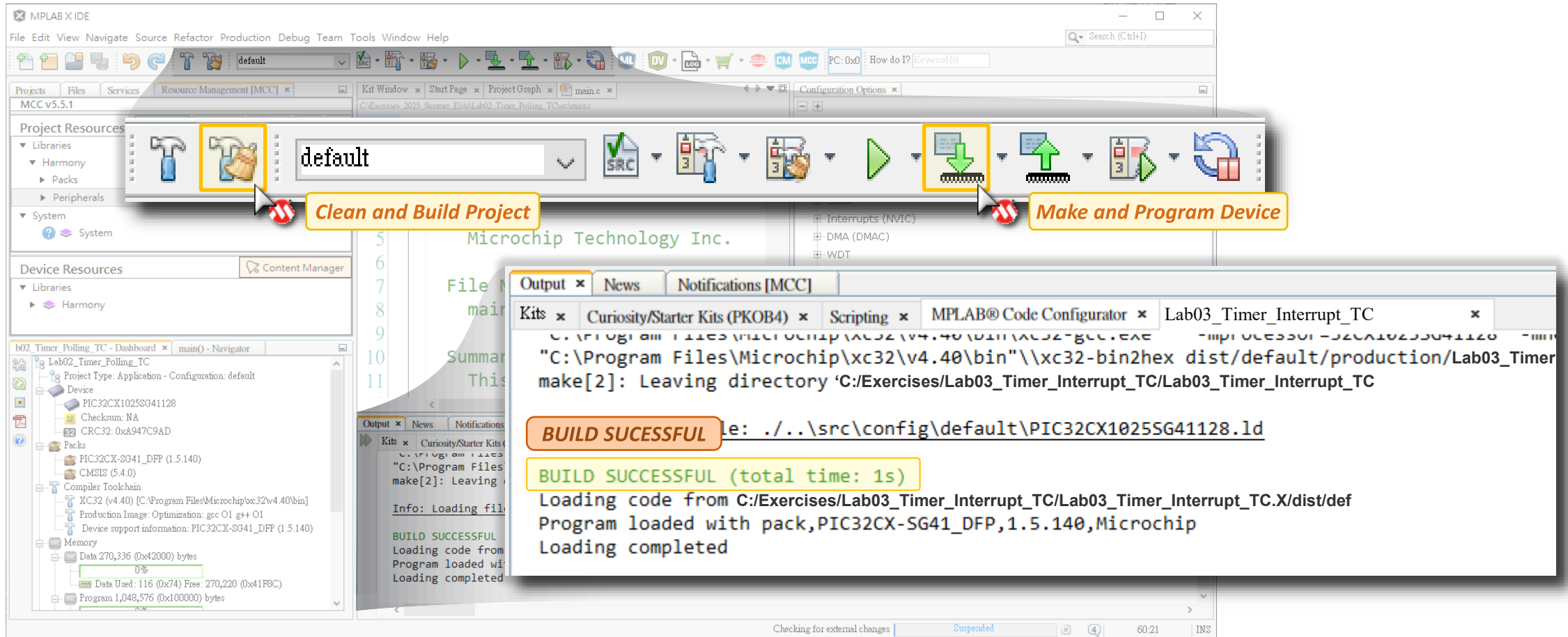
    return ( EXIT_FAILURE );
}
```

main.c

⚙️ Lab3 Timer Interrupt Callback TC

Step 8

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains several icons, with two highlighted by yellow boxes and red arrows. The first icon, a hammer and wrench, is labeled "Clean and Build Project". The second icon, a green arrow pointing down to a circuit board, is labeled "Make and Program Device".

The bottom panel shows the Output window with the following text:

```
Output x News Notifications [MCC]
Kits x Curiosity/Starter Kits (PKOB4) x Scripting x MPLAB® Code Configurator x Lab03_Timer_Interrupt_TC
C:\Program Files\Microchip\XC32\v4.40\bin\XC32-GCC.EXE -impf UCES301 -J2CA10255G41128 -impf
"C:\Program Files\Microchip\xc32\v4.40\bin"\xc32-bin2hex dist/default/production/Lab03_Timer
make[2]: Leaving directory 'C:/Exercises/Lab03_Timer_Interrupt_TC/Lab03_Timer_Interrupt_TC'

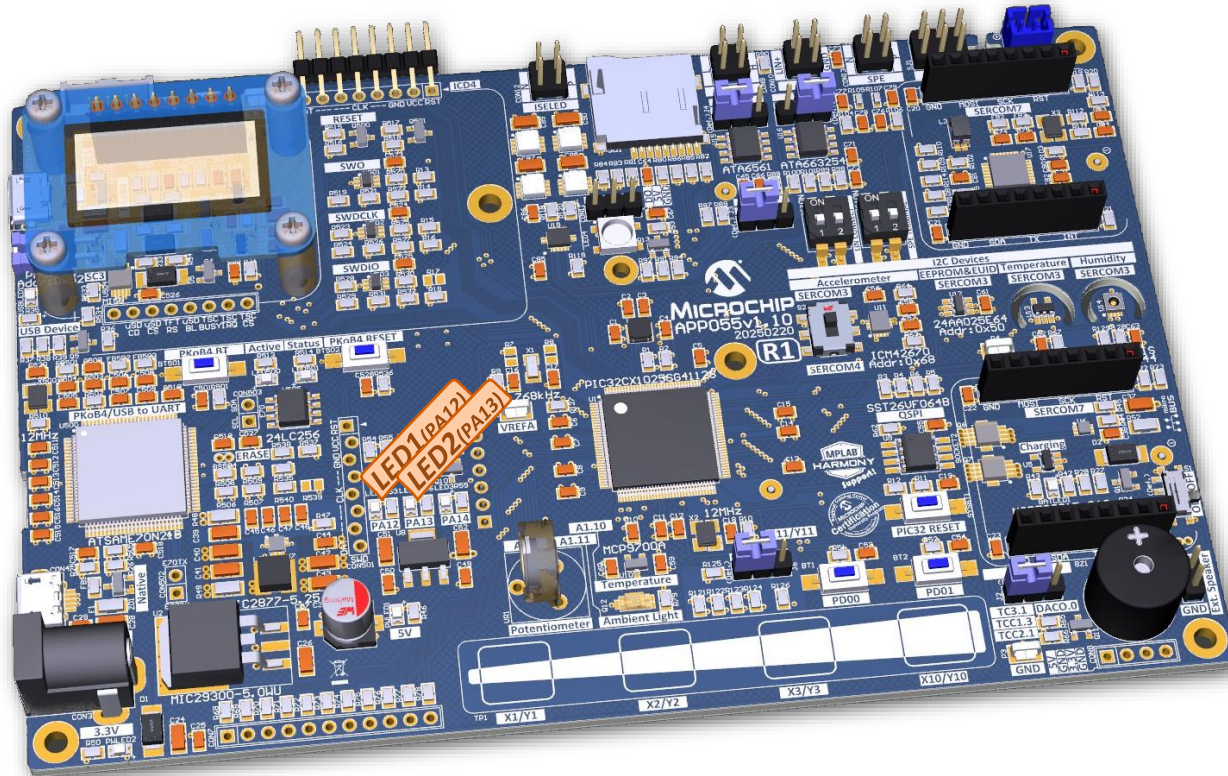
BUILD SUCCESSFUL le: ../../src/config/default/PIC32CX10255G41128.ld

BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab03_Timer_Interrupt_TC/Lab03_Timer_Interrupt_TC.X/dist/def
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```


Lab3 Timer Interrupt Callback TC

Result

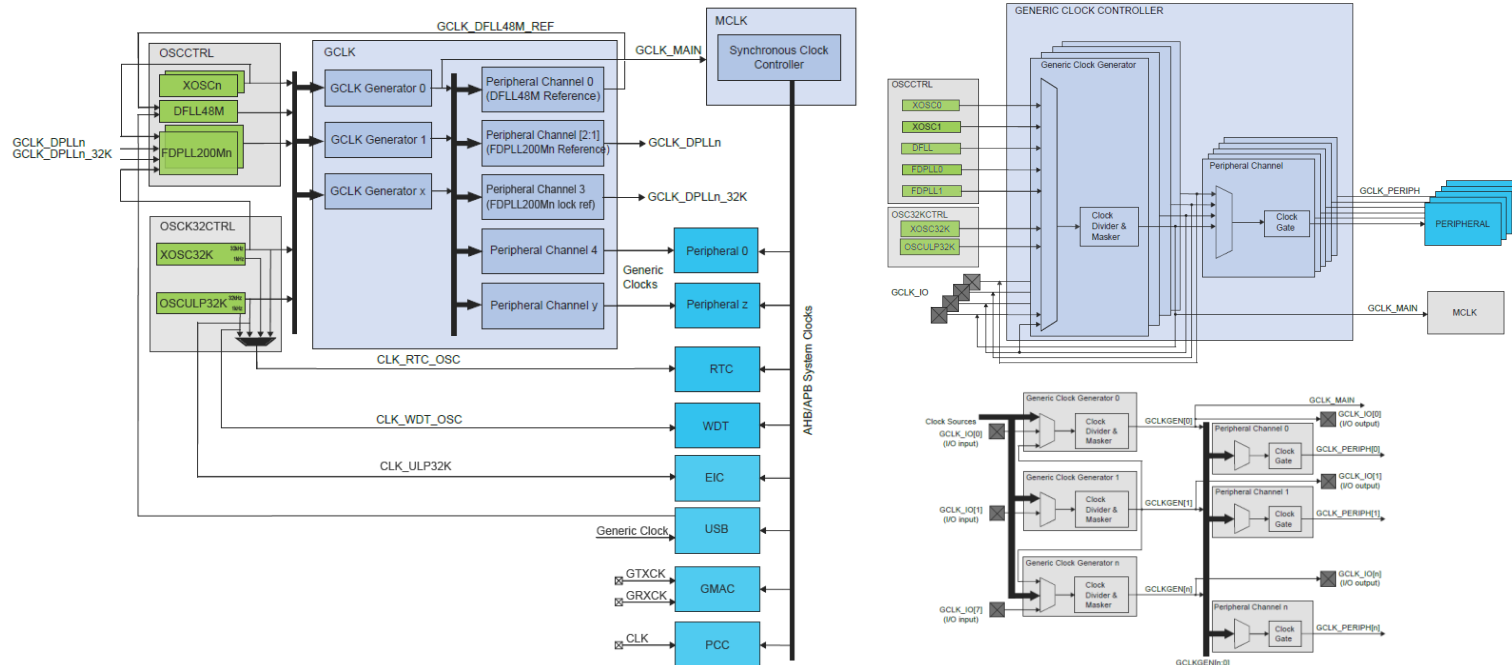
- **The LEDs Toggle same as Lab2 but change to interrupt callback method.**
 - LED1(PA12), Toggle by TC0 interrupt callback.
 - LED2(PA13), Toggle by TC1 interrupt callback.



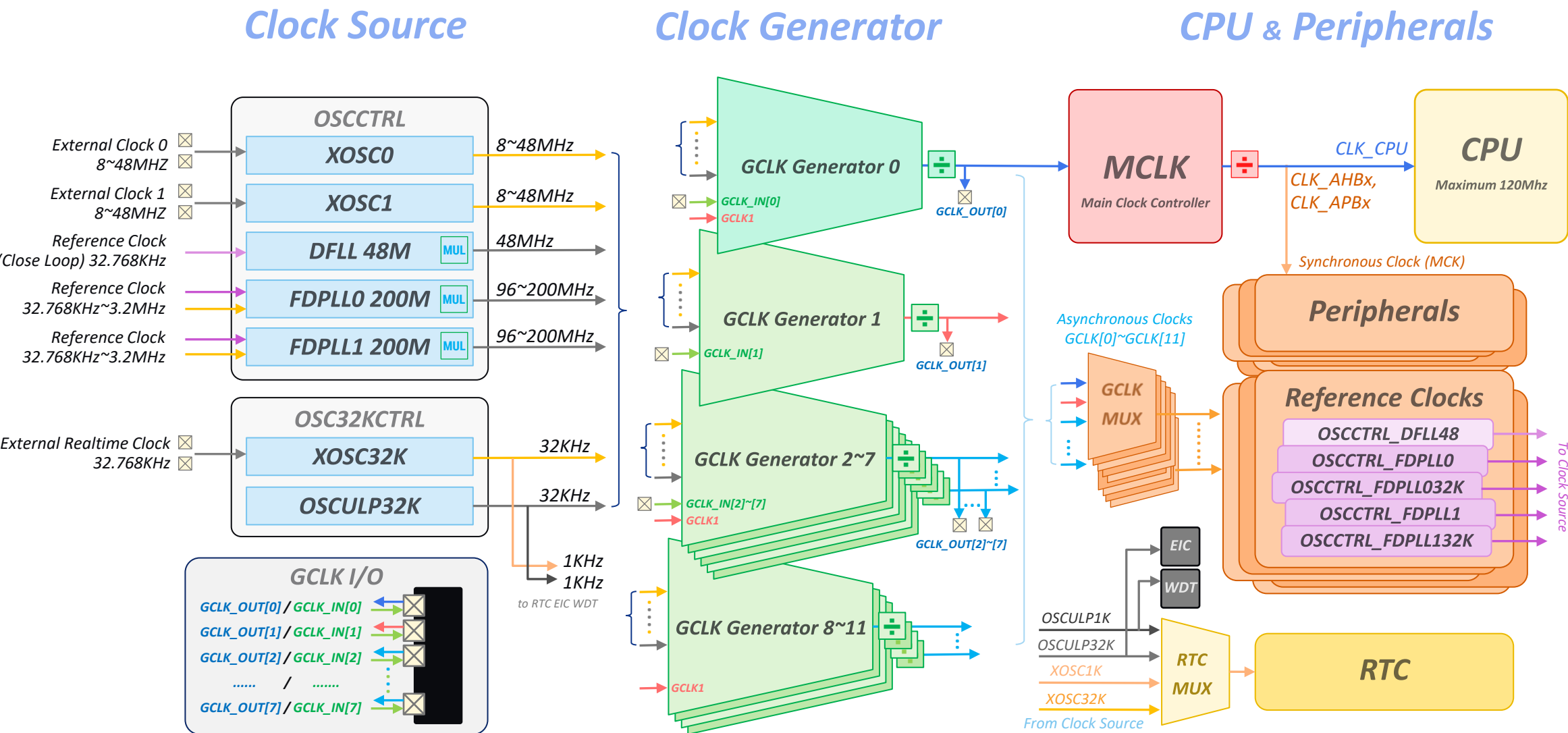
Oscillator and Clock Generator

Clock System

- Multiple clock sources managed by OSCCTRL/OSC32KCTRL include External and DFLL/FDPLL.
- 12 Generic Clock Generators (GCLKs) create configurable clock signals from these sources.
- GCLKs feed peripheral channels and DFLL input.
- GCLK0 dedicate provides the MCLK (maximum 120MHz) for CPU.
- MCLK to generate synchronous clocks for CPU, AHB/APB buses, and peripherals.
- MCLK also includes prescalers and clock masks for efficient power and clock control.

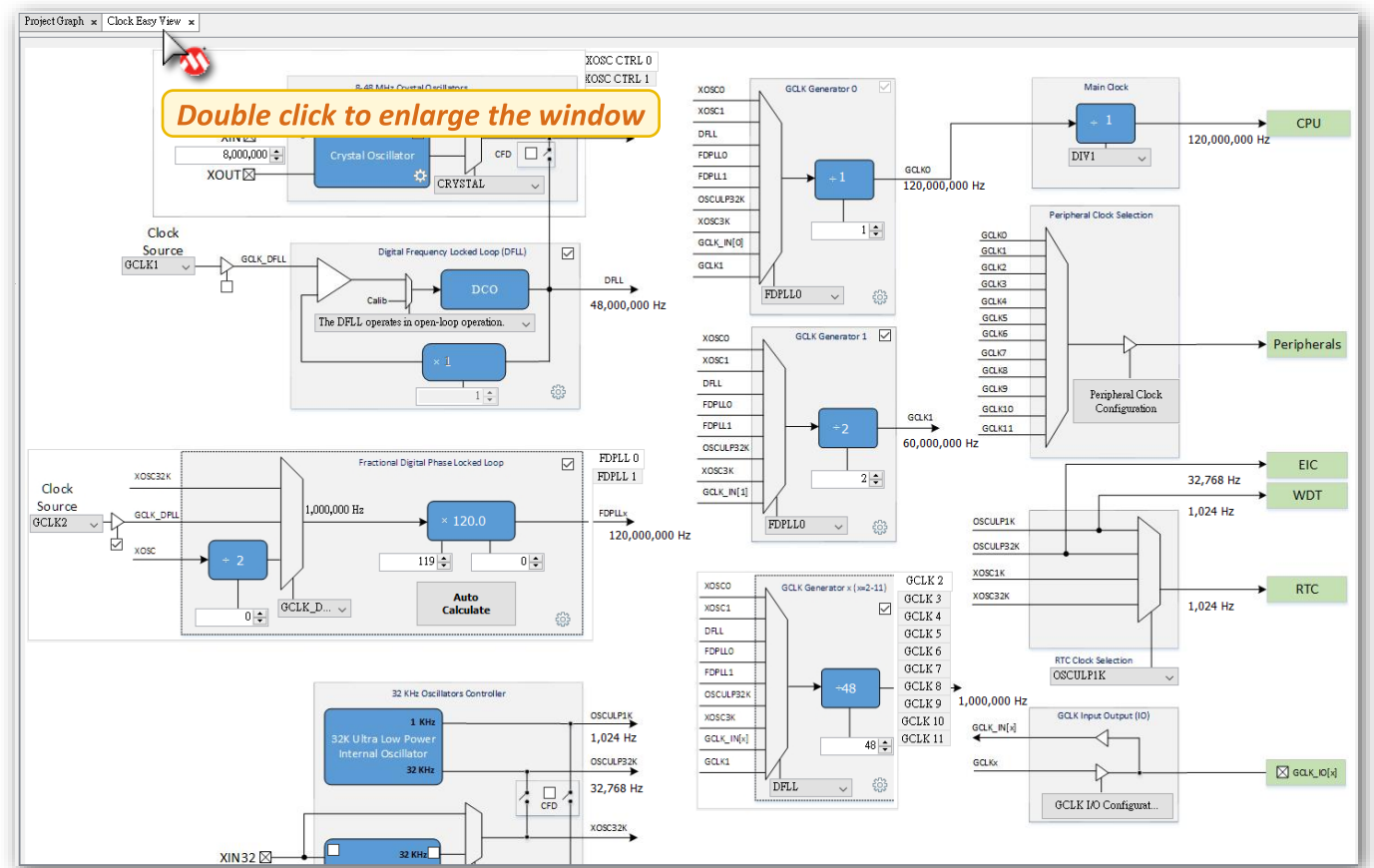
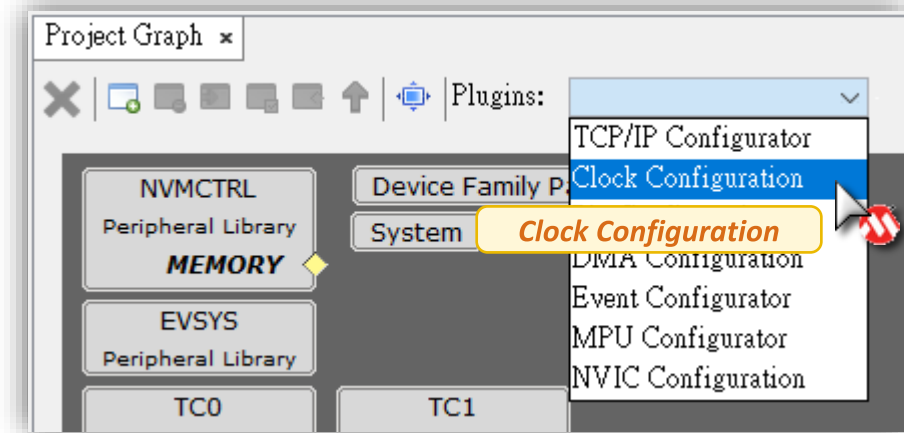


Clock System



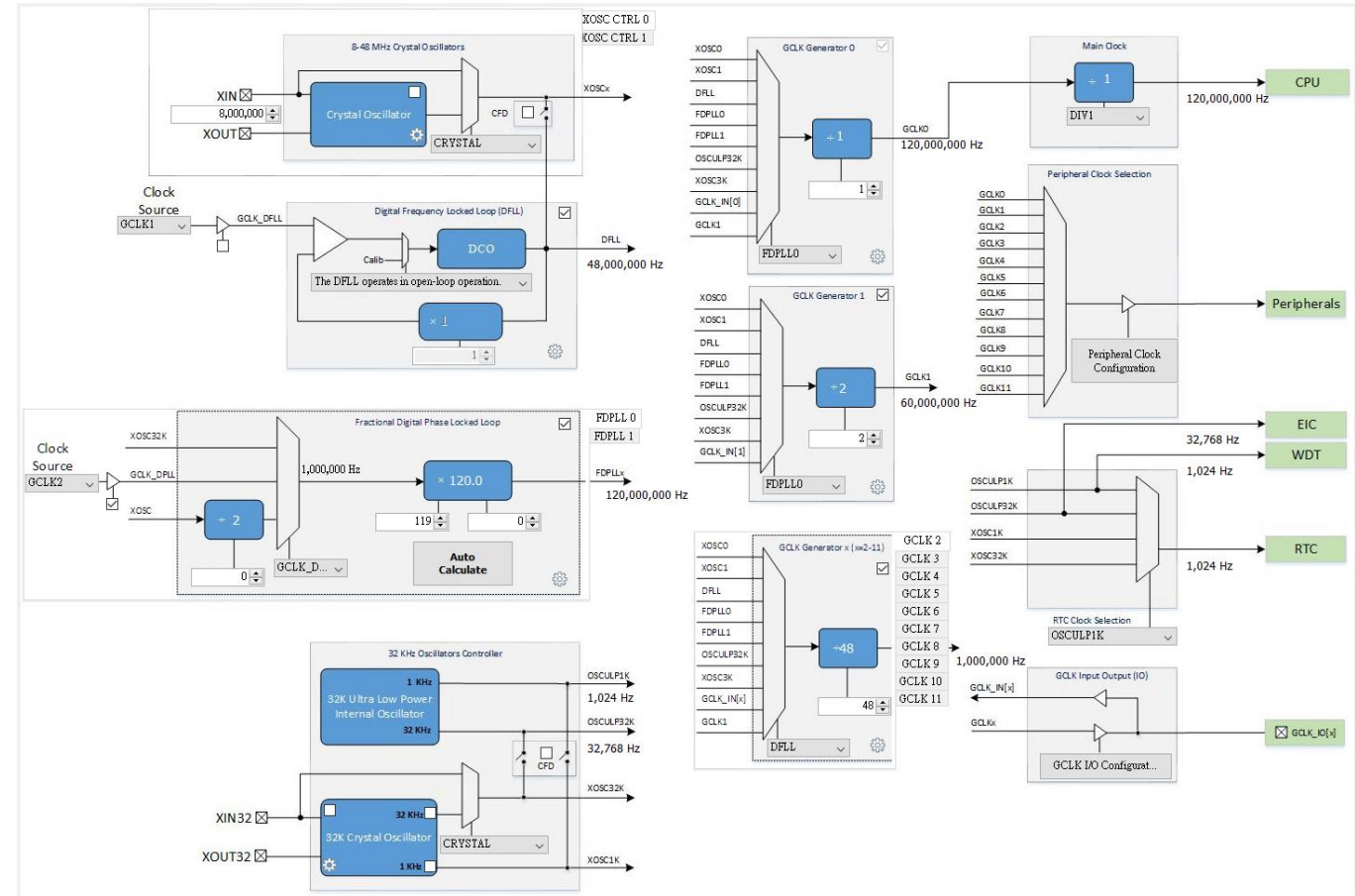
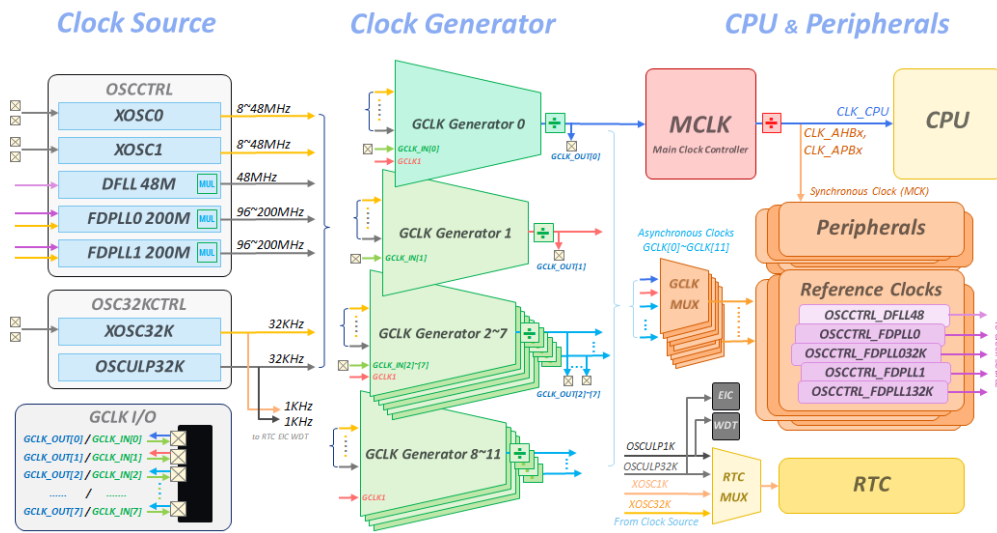
Clock Setting Use MCC Harmony

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



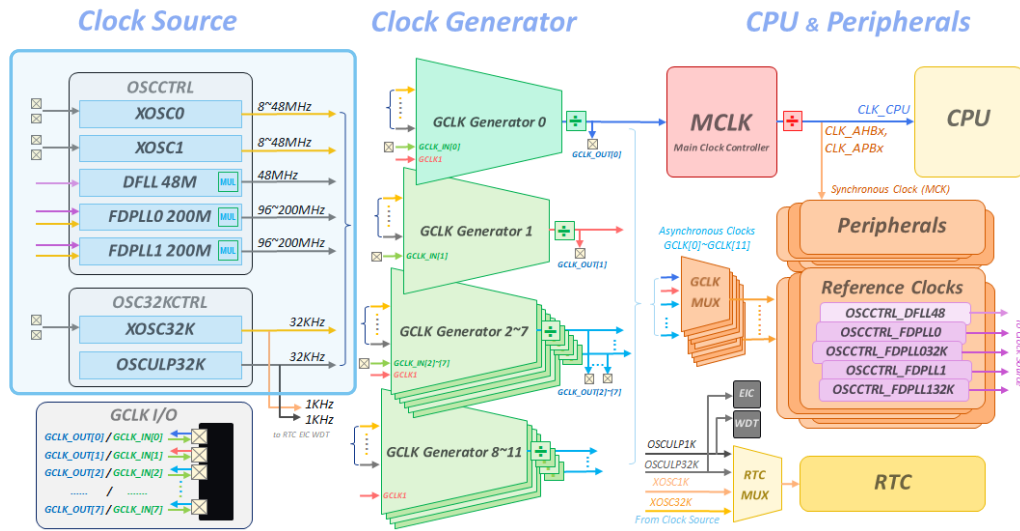
Clock Setting Use MCC Harmony

- The Clock Easy View GUI layout.

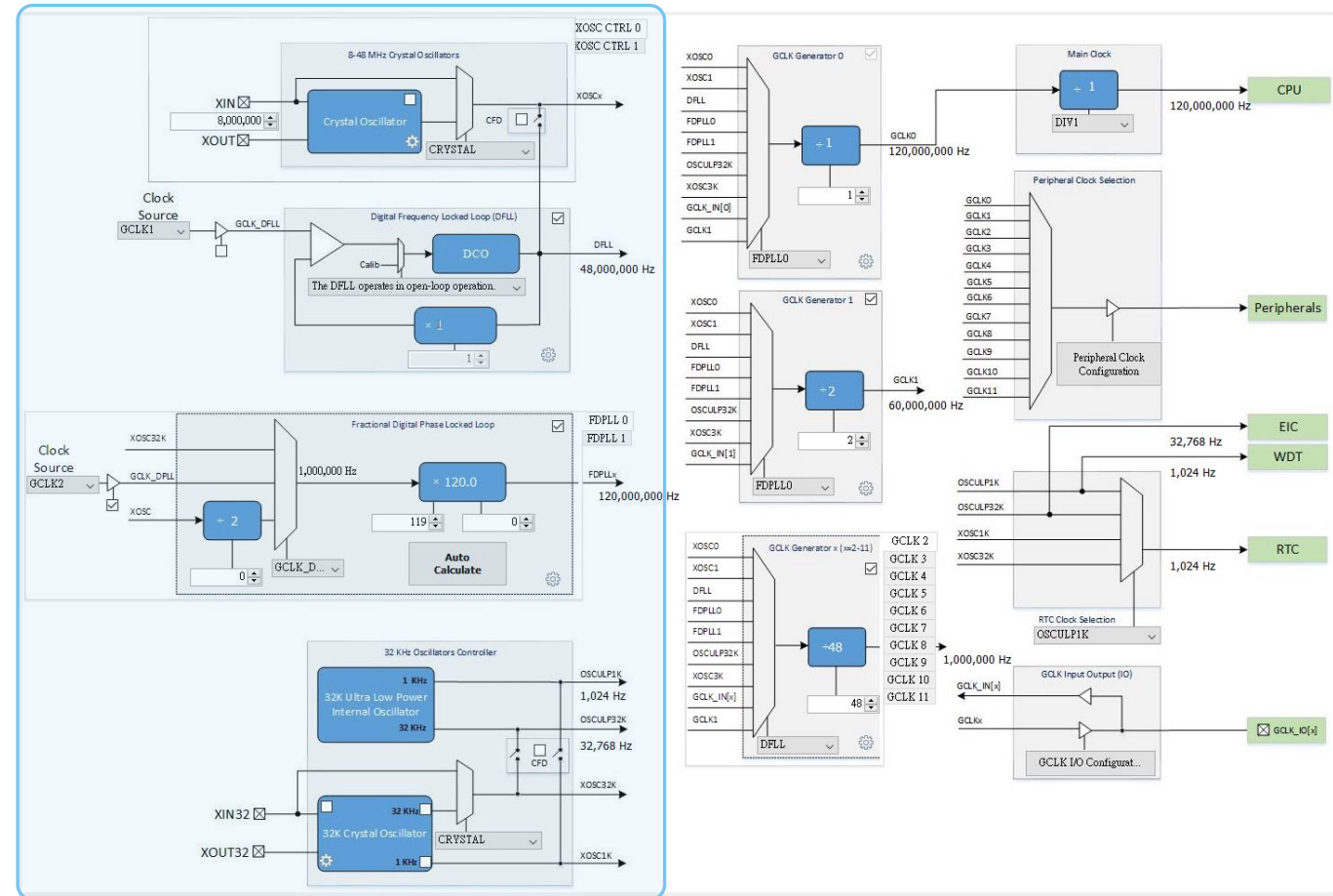


Clock Setting Use MCC Harmony

- Clock Source

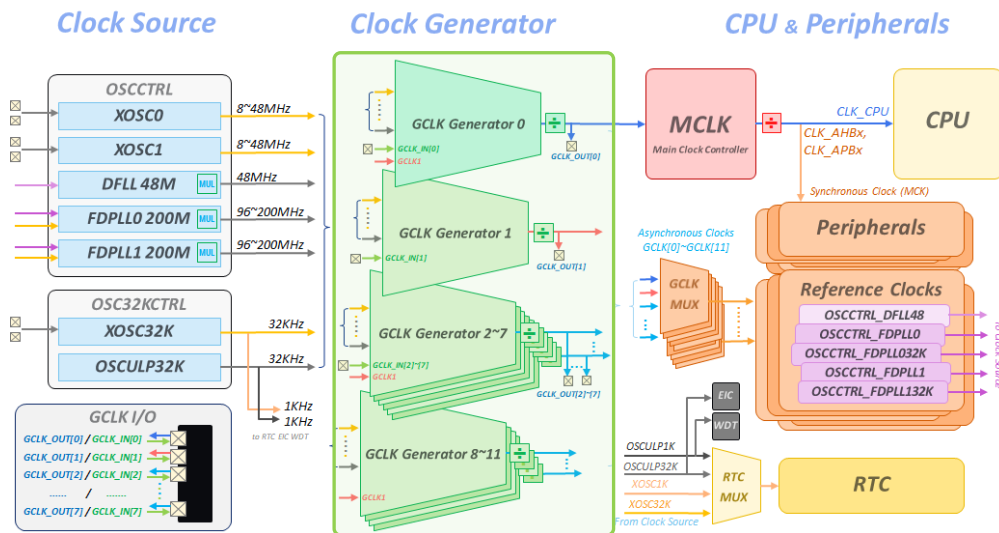


Clock Source

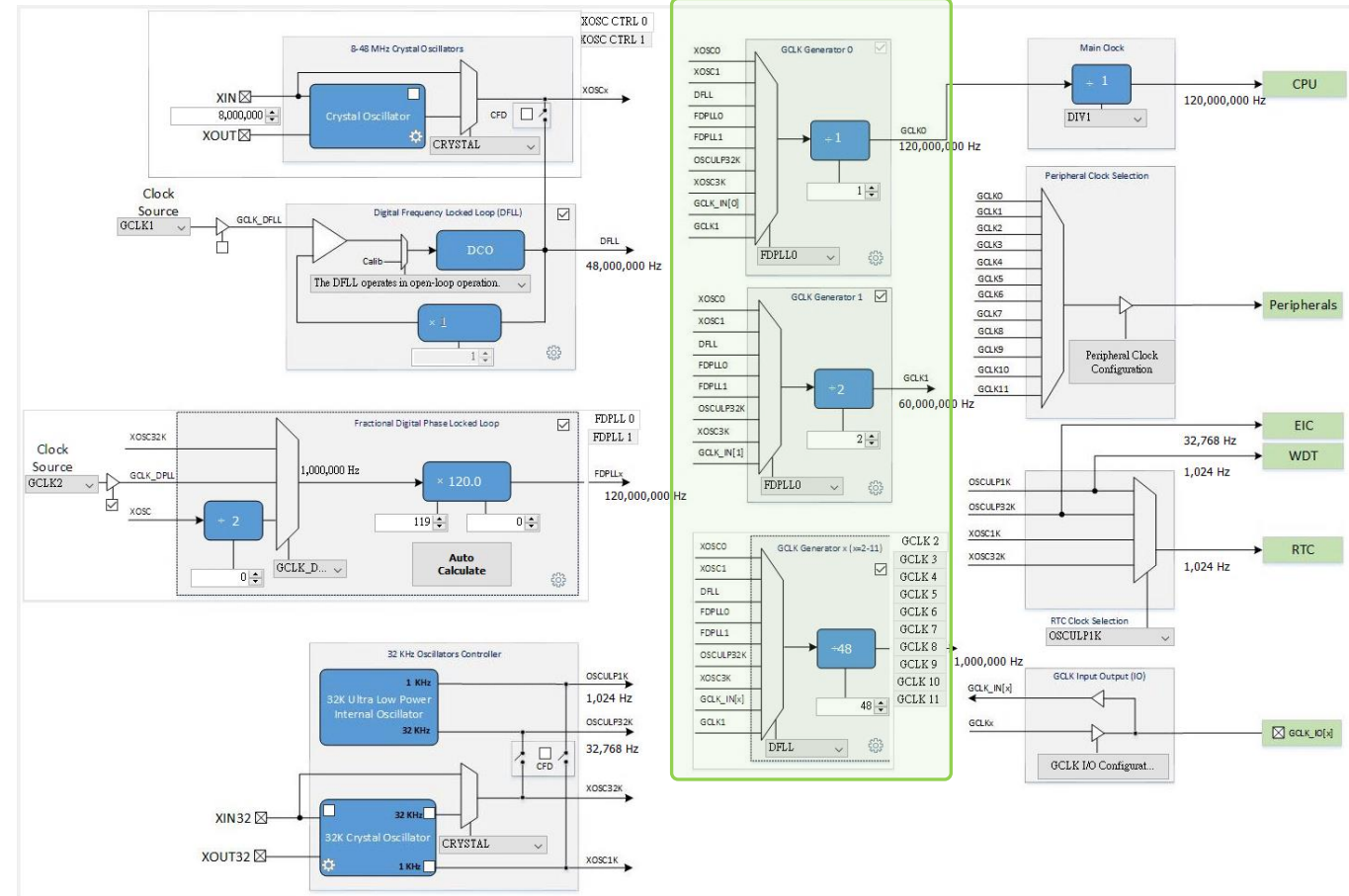


Clock Setting Use MCC Harmony

- Clock Generator

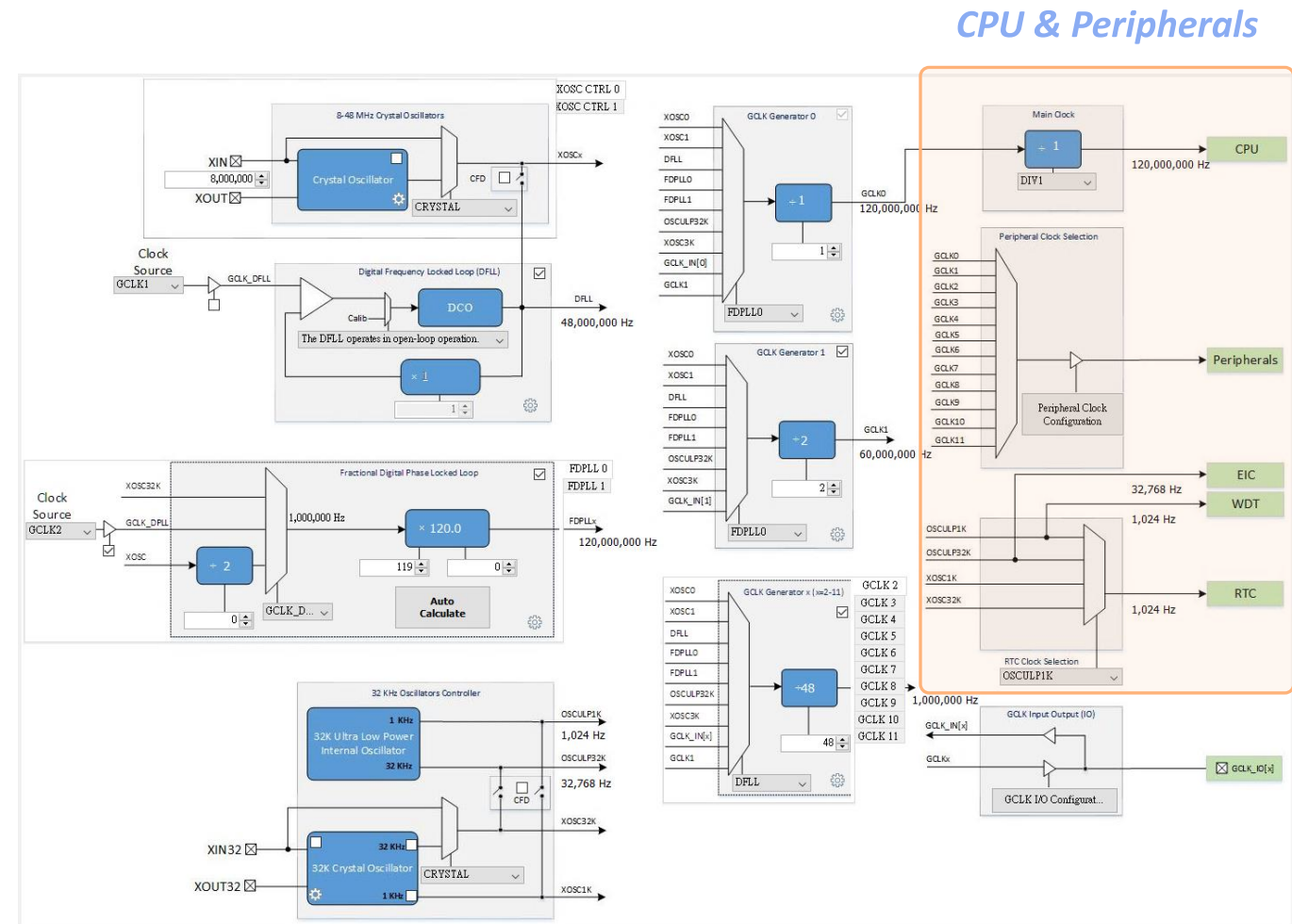
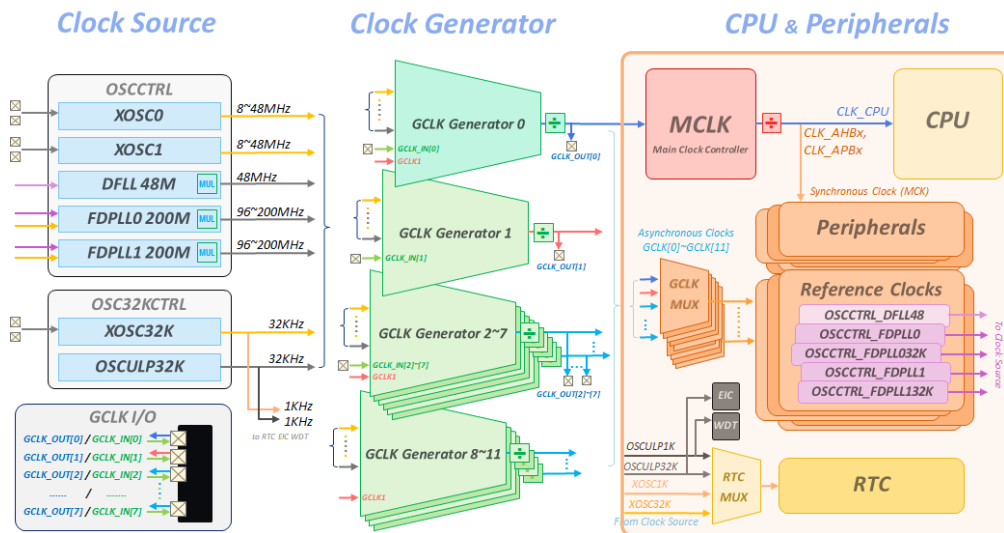


Clock Generator



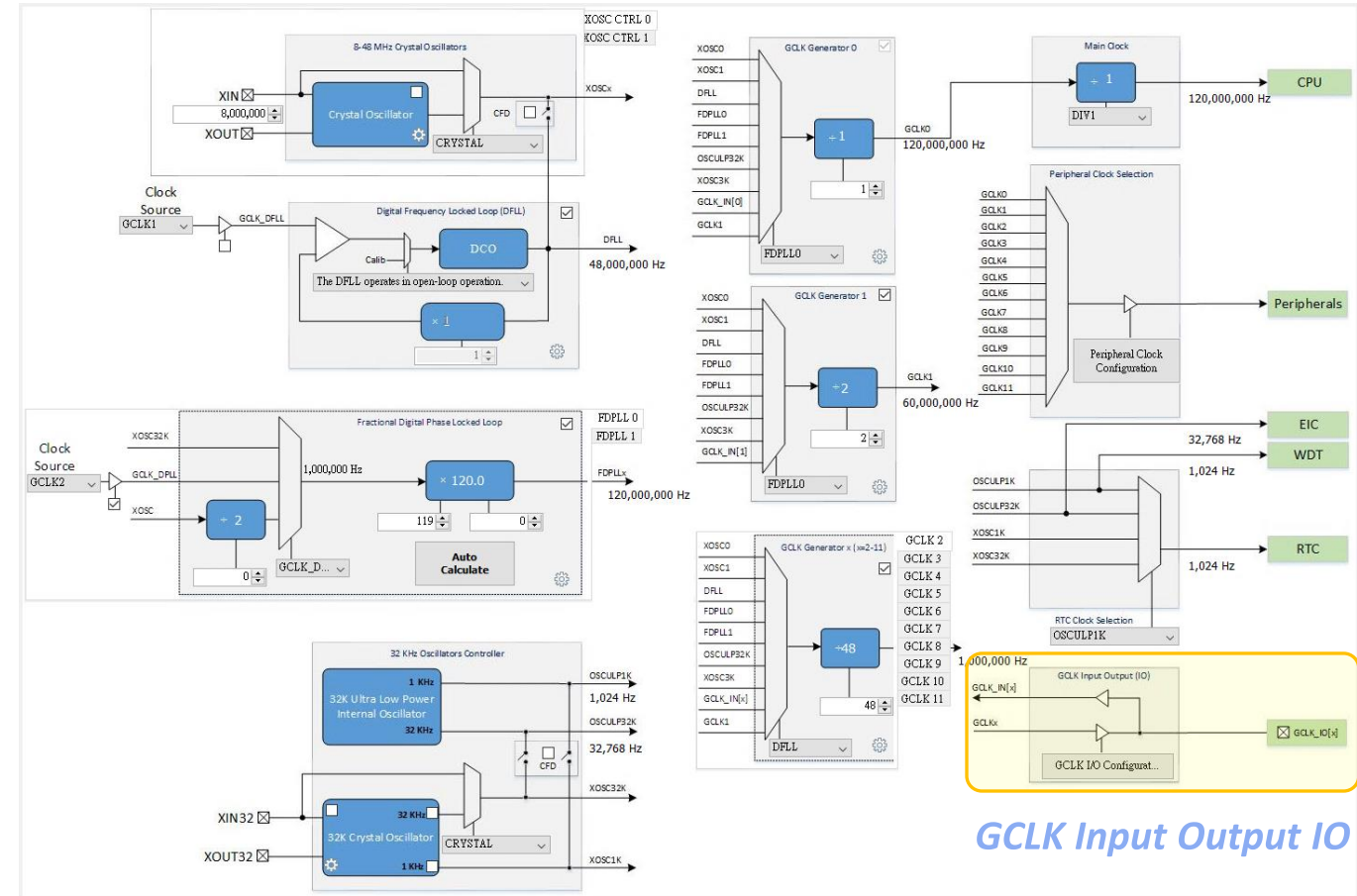
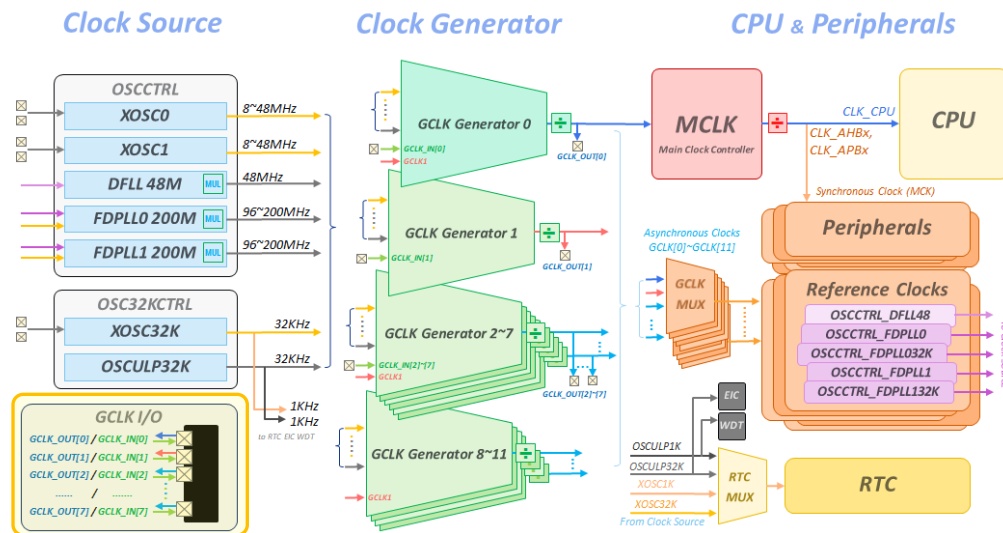
Clock Setting Use MCC Harmony

- CPU & Peripherals



Clock Setting Use MCC Harmony

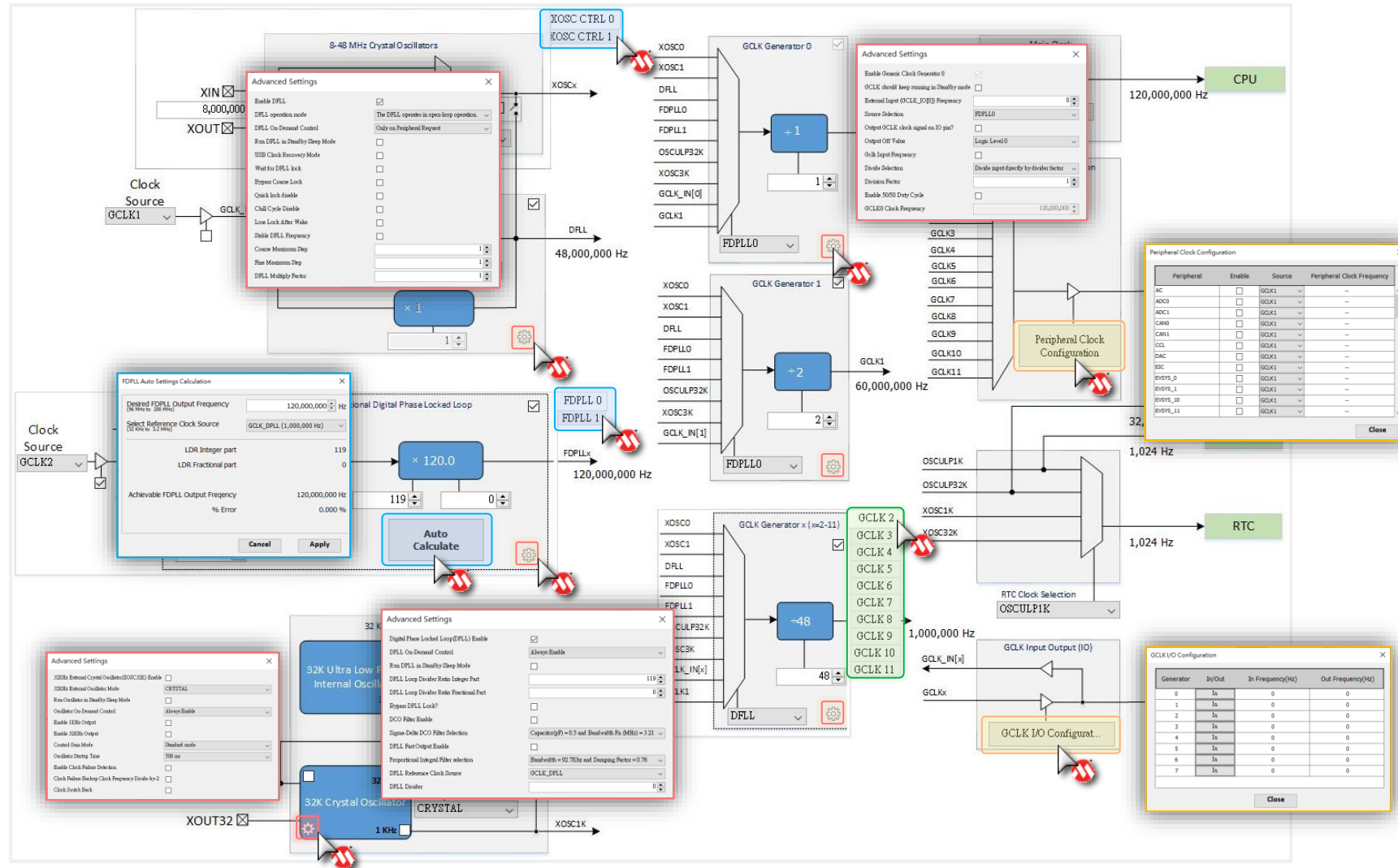
- GCLK Input Output (IO)



GCLK Input Output IO

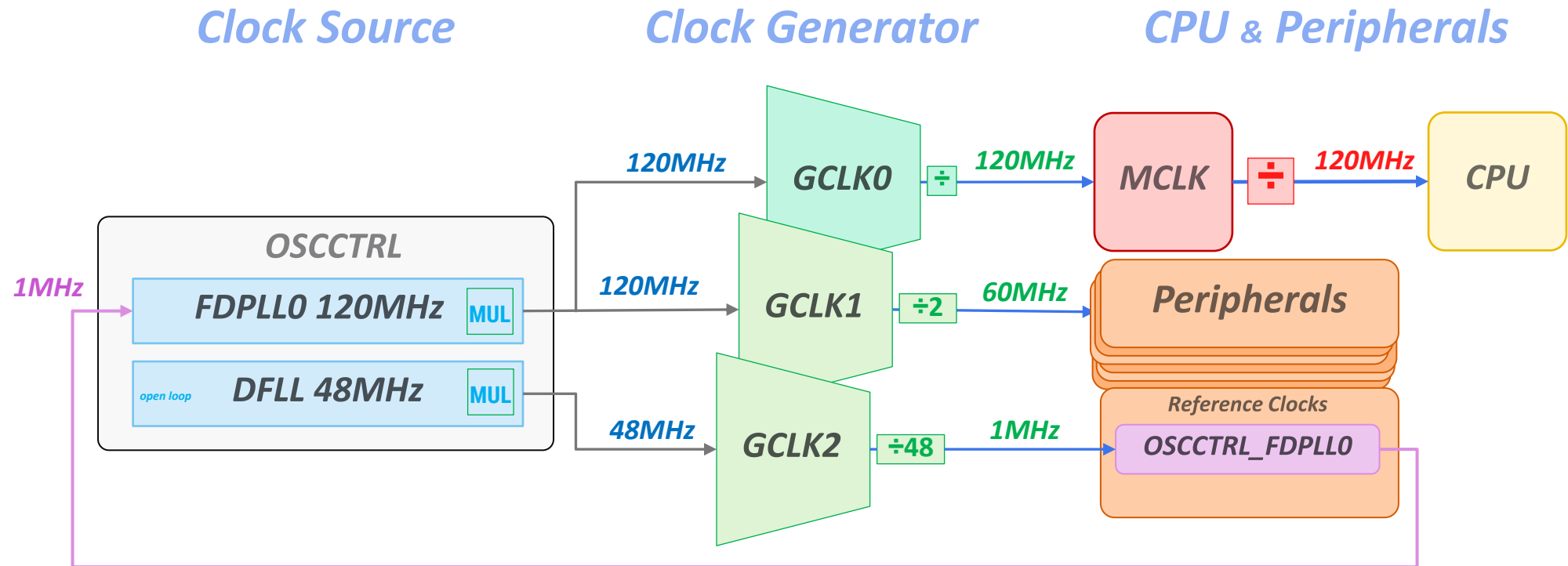
Clock Setting Use MCC Harmony

- The Multiple Modules Selection and Extra Configure Dialog.



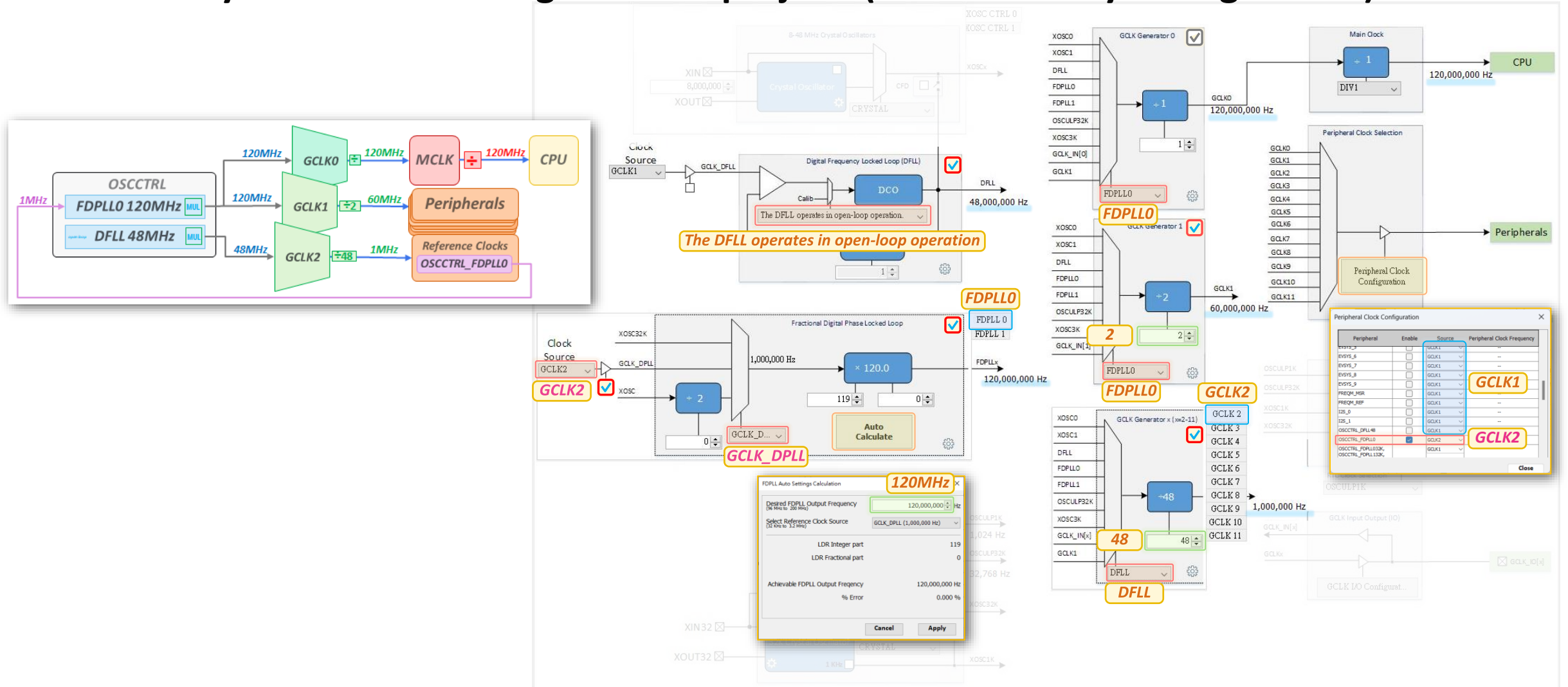
Clock Setting Use MCC Harmony

- Default System Clock routing of a new project.
 - DFLL48MHz (open loop) > GCLK2 (DIV48) > FDPLL0 120MHz > GCLK0 > MCLK 120MHz
 - DFLL48MHz (open loop) > GCLK2 (DIV48) > FDPLL0 120MHz > GCLK1(DIV2) > Peripherals 60MHz



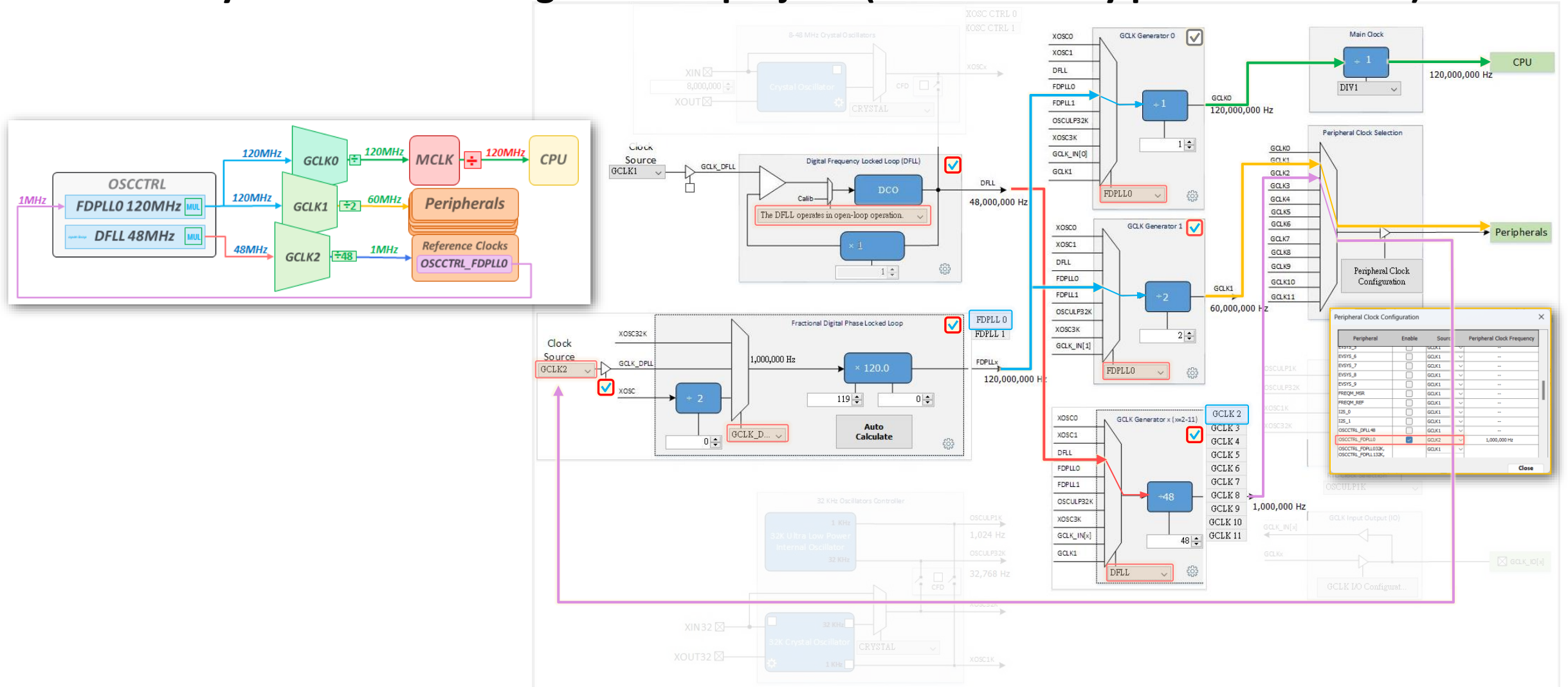
Clock Setting Use MCC Harmony

- Default System Clock routing of a new project. (MCC Harmony Configuration)



Clock Setting Use MCC Harmony

- Default System Clock routing of a new project. (MCC Harmony pseudo connect)



Lab4 System Clock

DFLL Close Loop and Clock Output

Lab4 System Clock - DFLL Close Loop and Clock Output

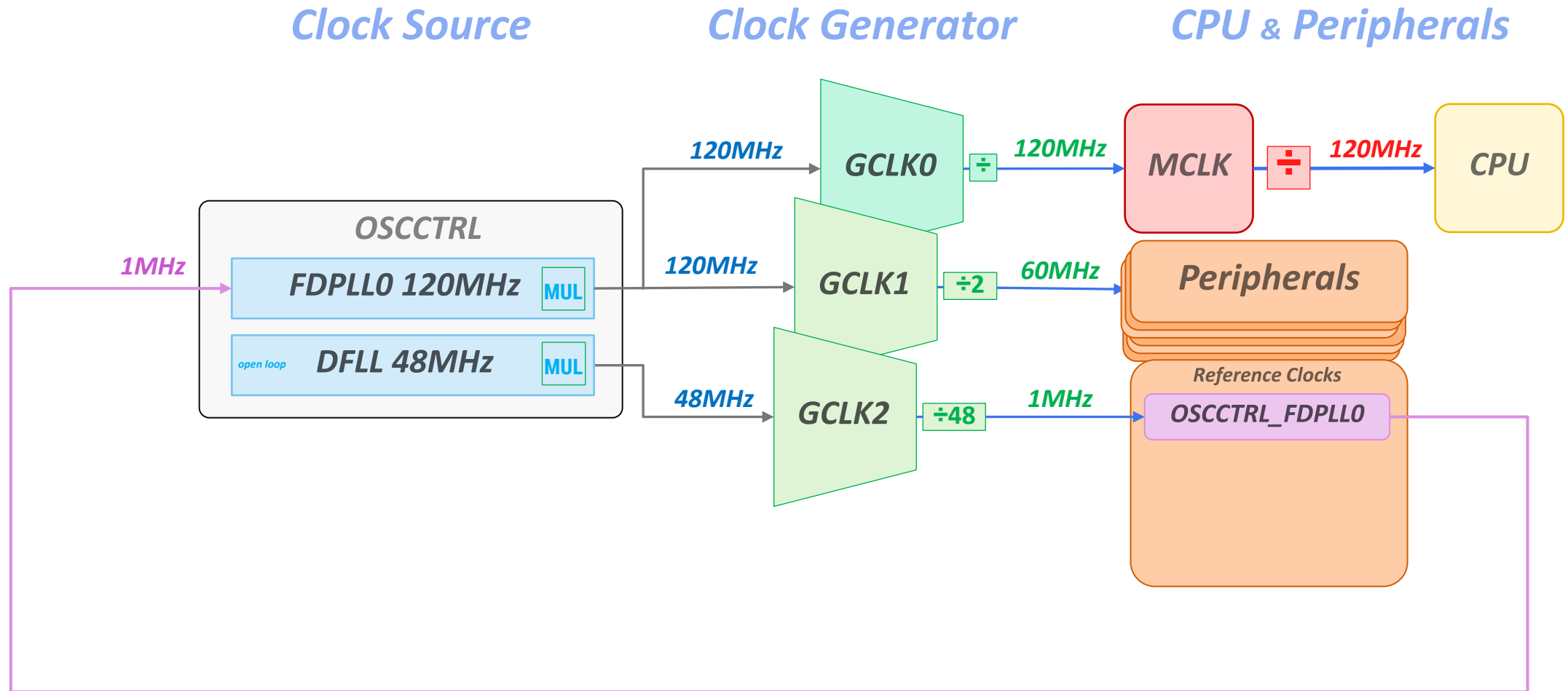
- The current system clock source is generated by DFLL48M using Open Loop, which is an internal clock source with an RC oscillator and a Digital Frequency-Locked Loop. Therefore, the frequency itself is less accurate and is prone to larger errors as the temperature changes.
- In this experiment, our goal is to achieve the Close Loop of DFLL48M and use an external 32K oscillator crystal as the Reference Frequency of the DFLL48M to achieve higher frequency accuracy.
- The following are the steps of clock routing:
 1. Enable the external XOSC32K clock source. (APP055 has a on-boarded external 32K oscillator)
 2. Use XOSC32K as the clock source of GCLK3 and divide it by 32 to get a 1KHz frequency.
 3. GCLK3 now output 1KHz to reference clock peripheral OSCCTRL_DFLL48 as the clock source of DFLL48M.
 4. Enable the Close Loop of DFLL48M and input the Multiply Factor as 46,875 to calculate a more accurate 48MHz frequency.

The following parts maintain the original clock configuration.

Let's go !

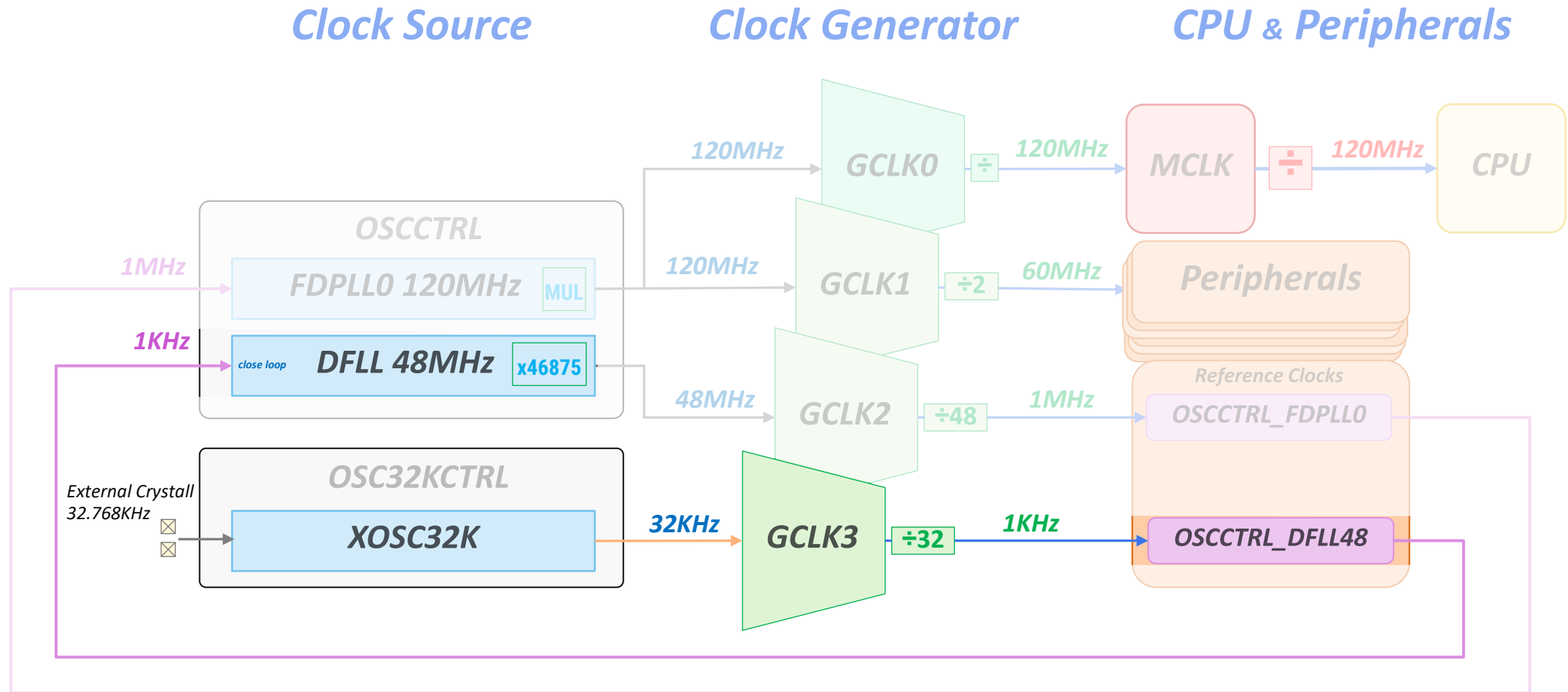
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

- The default configuration of the system clock routing.



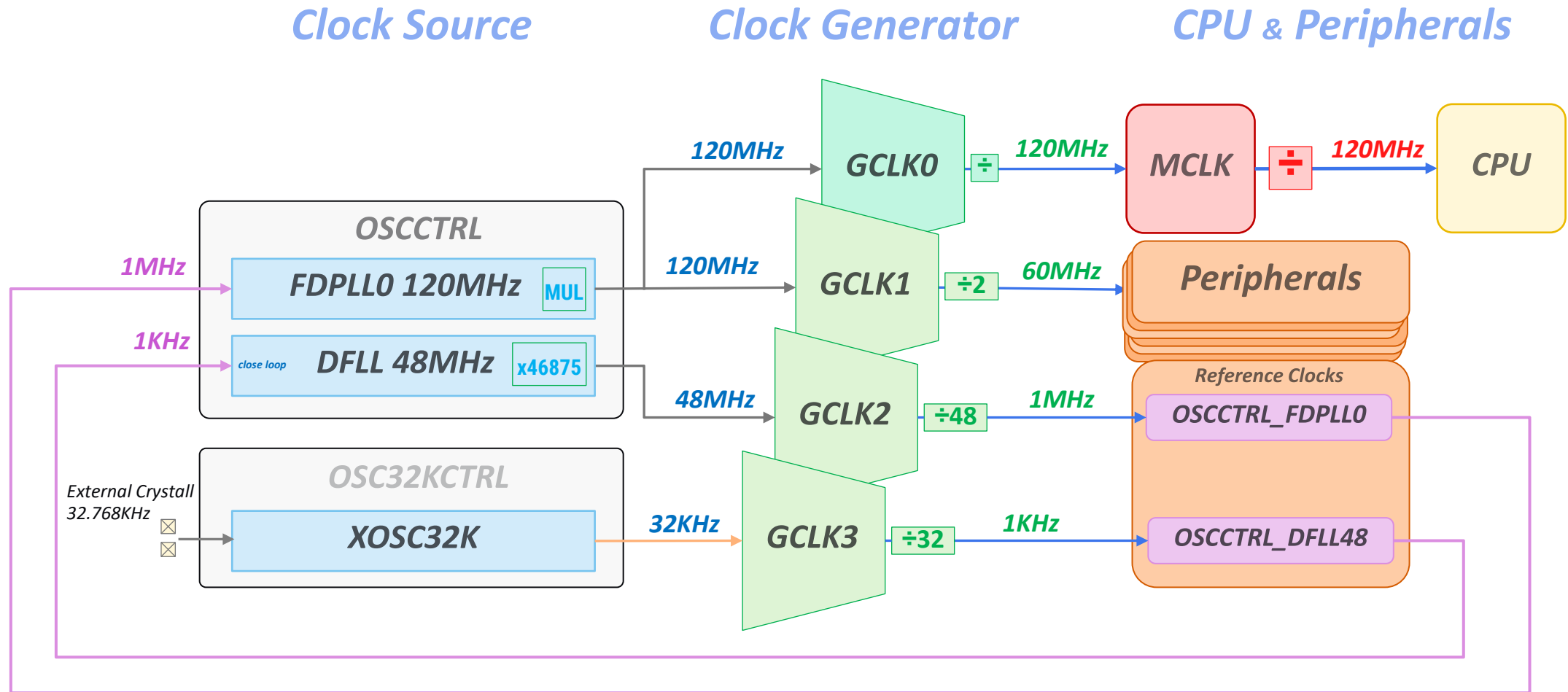
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

- The new configuration we are going to add and modify.



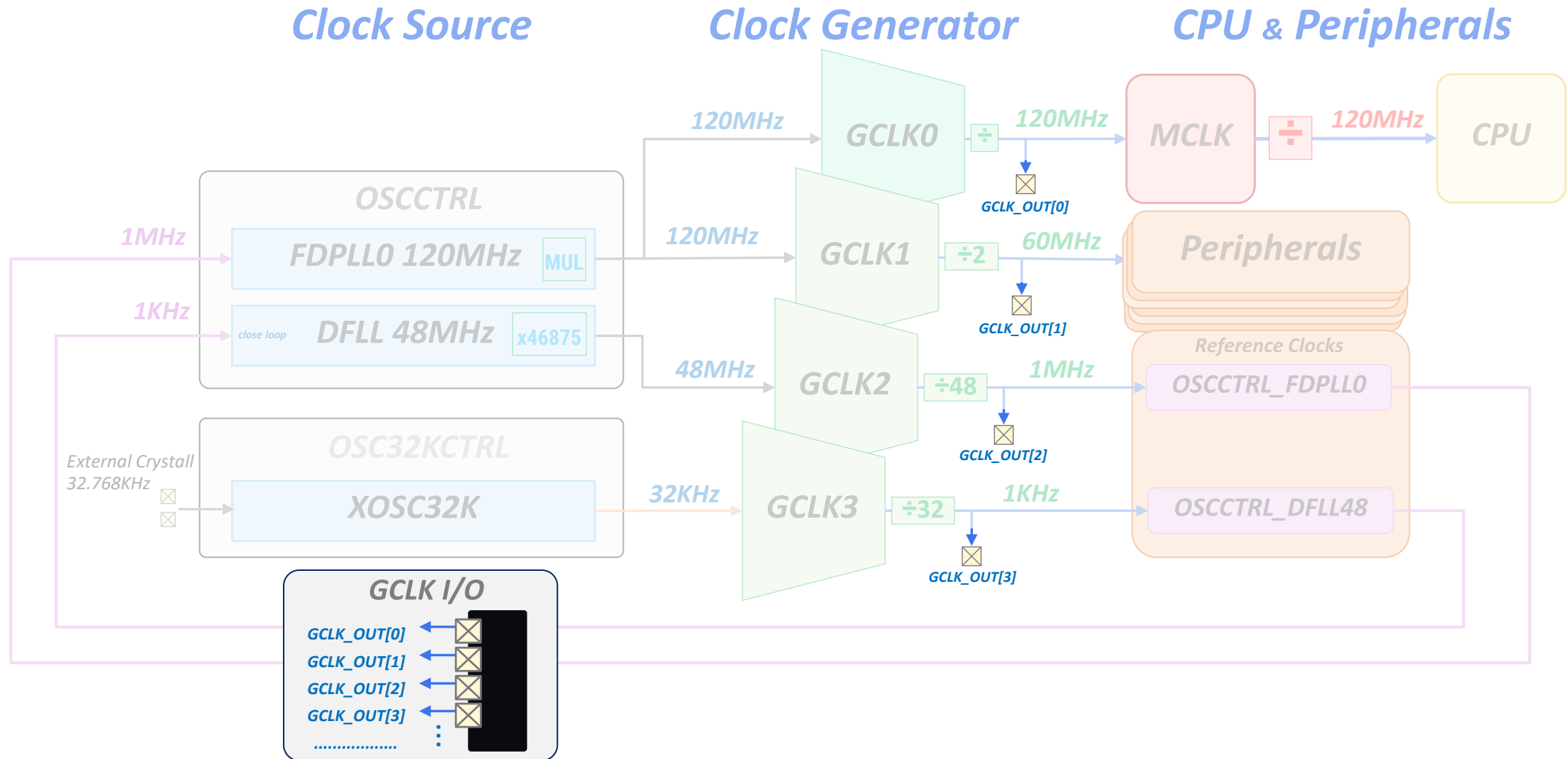
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

- The configuration of the system clock routing we want to implement.



⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

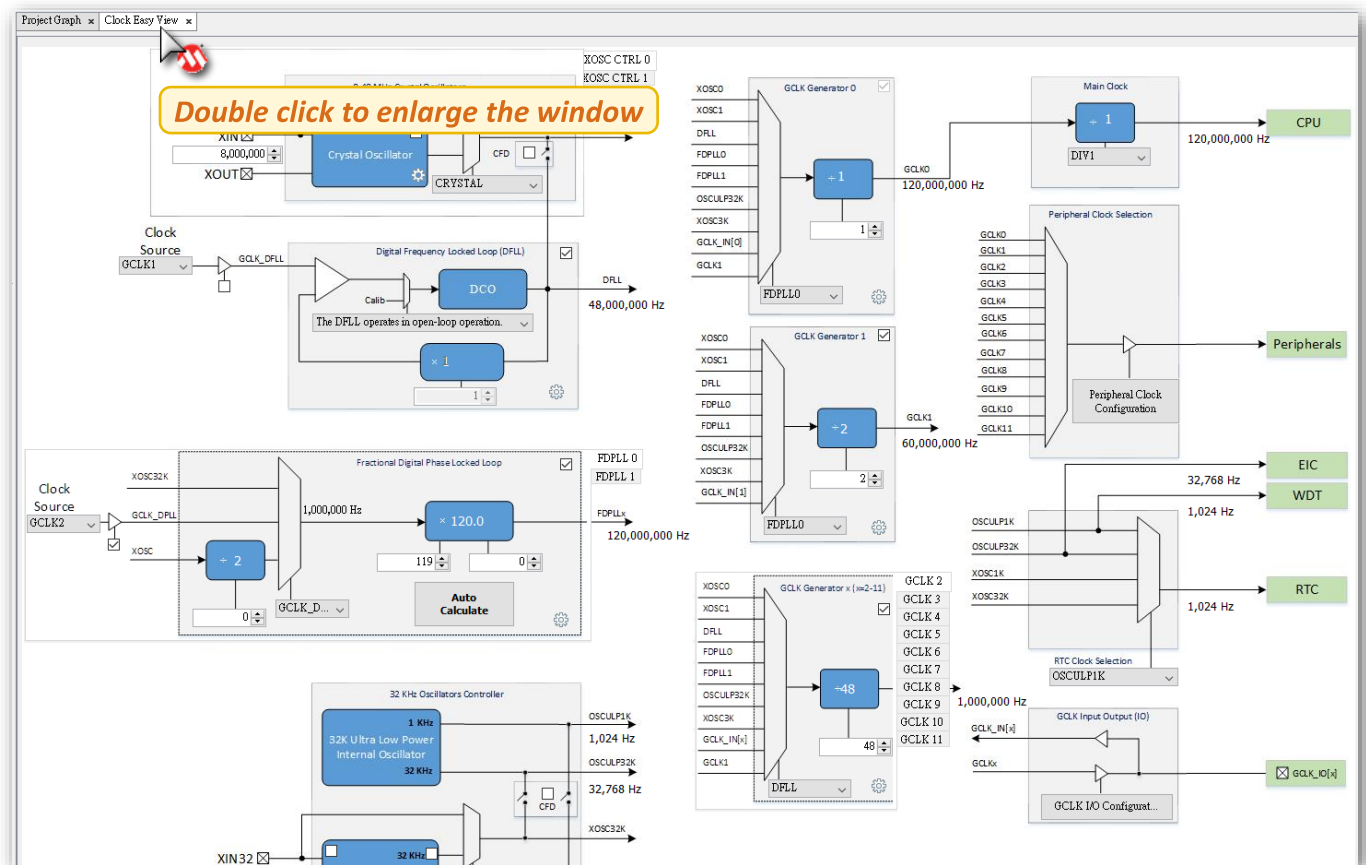
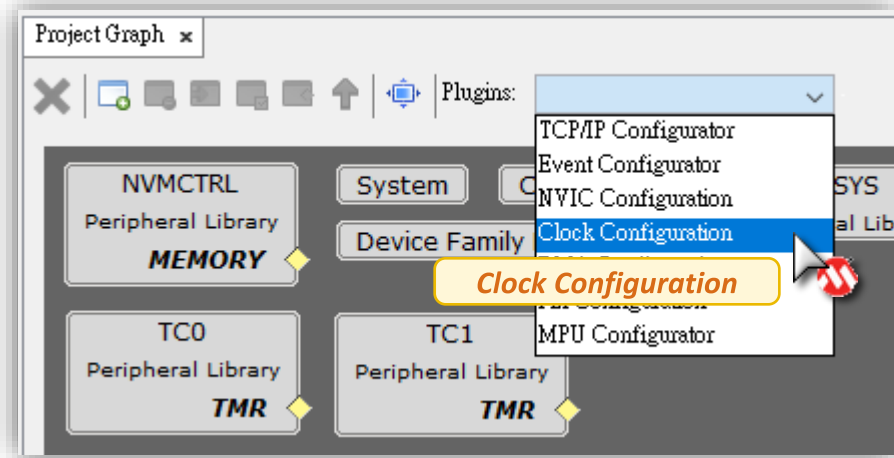
- Clock output configuration of the final system clock.



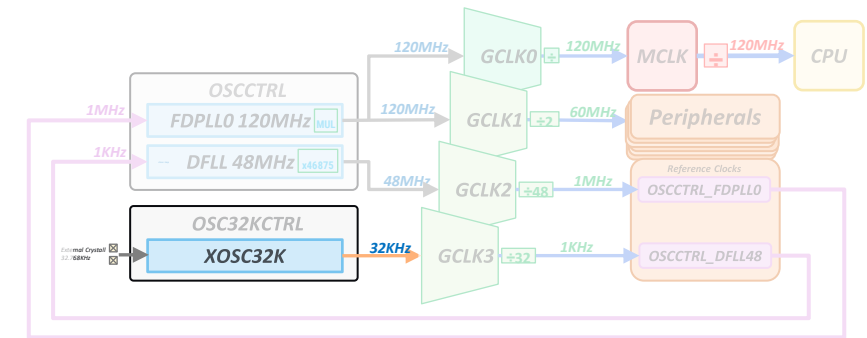
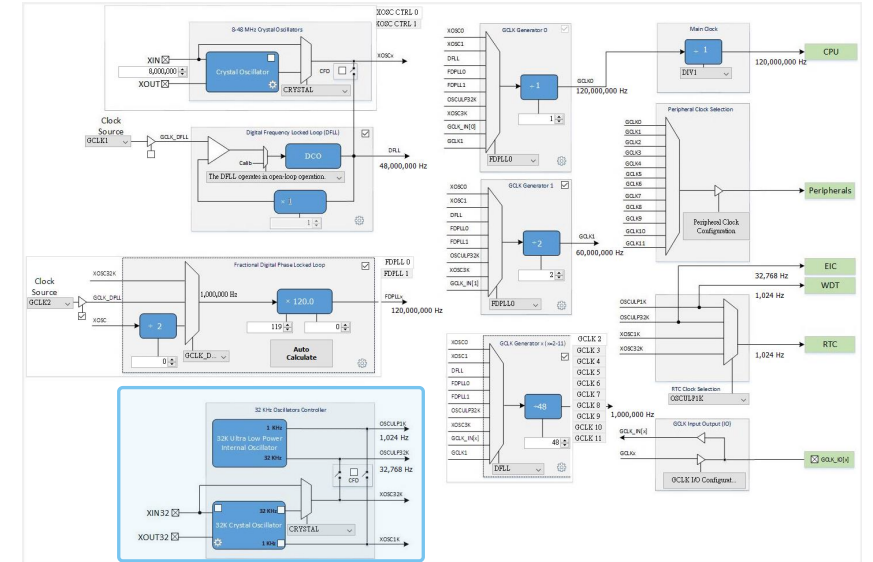
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 1

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

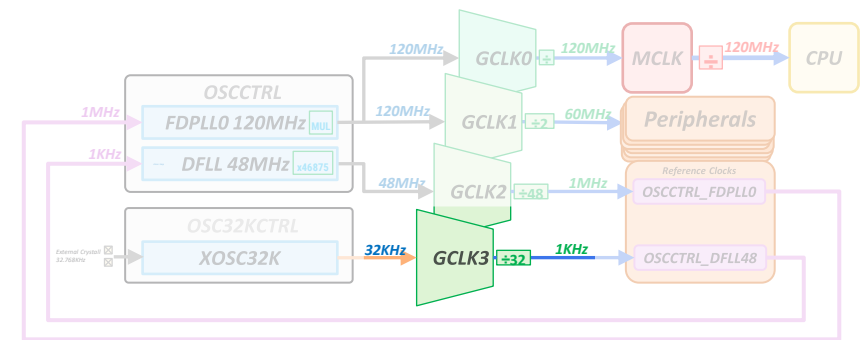
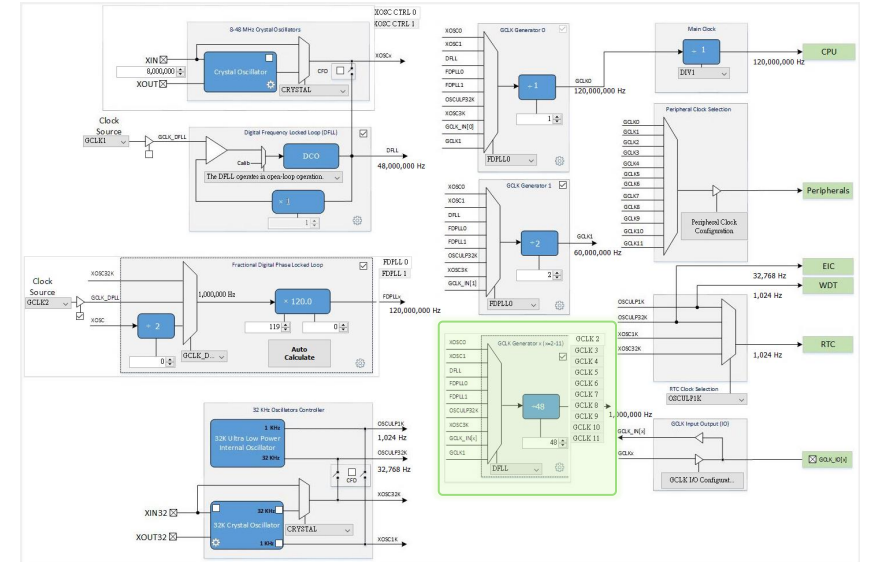
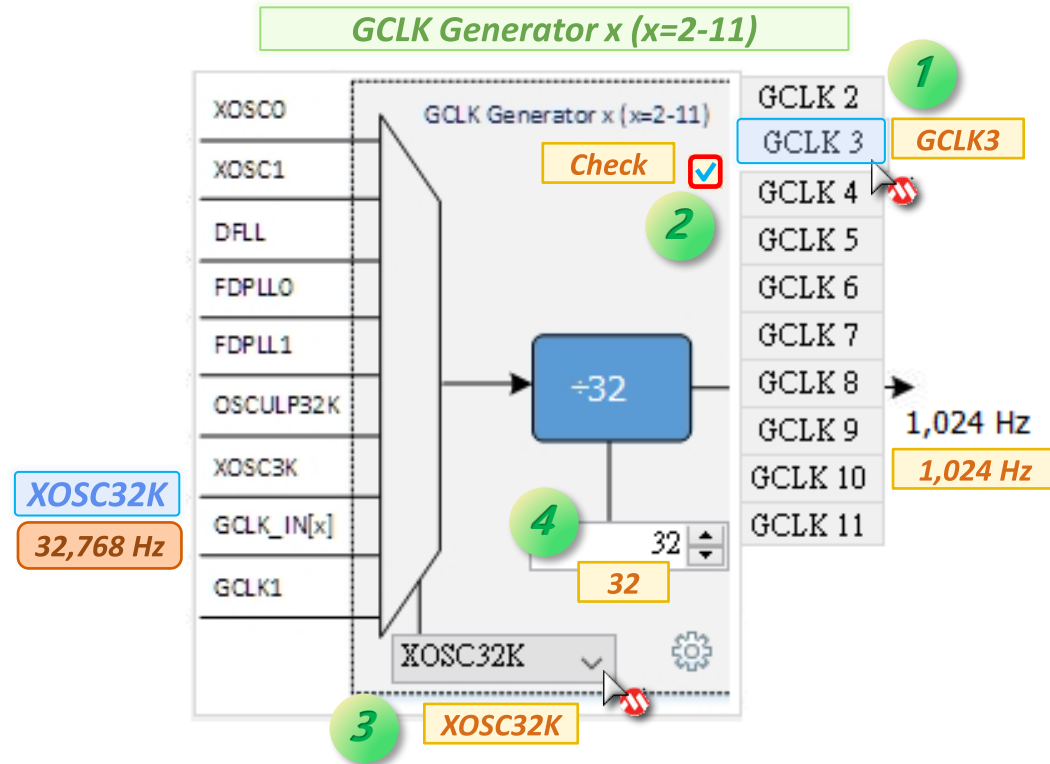


Step 2

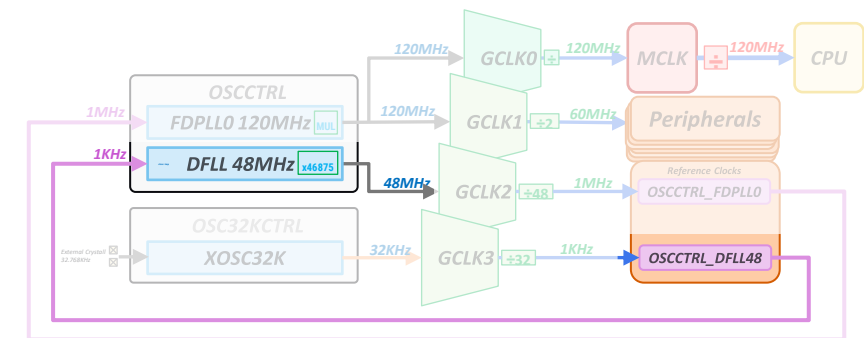
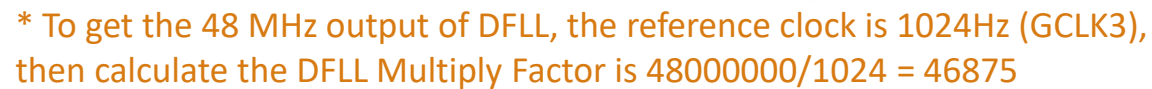


Lab4 System Clock - DFLL Close Loop and Clock Output

Step 3

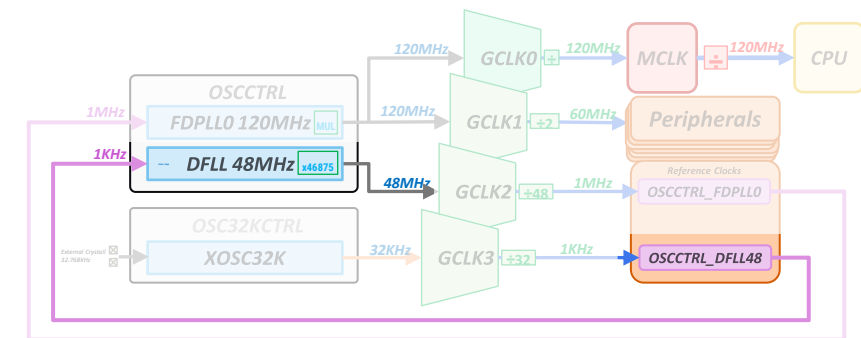
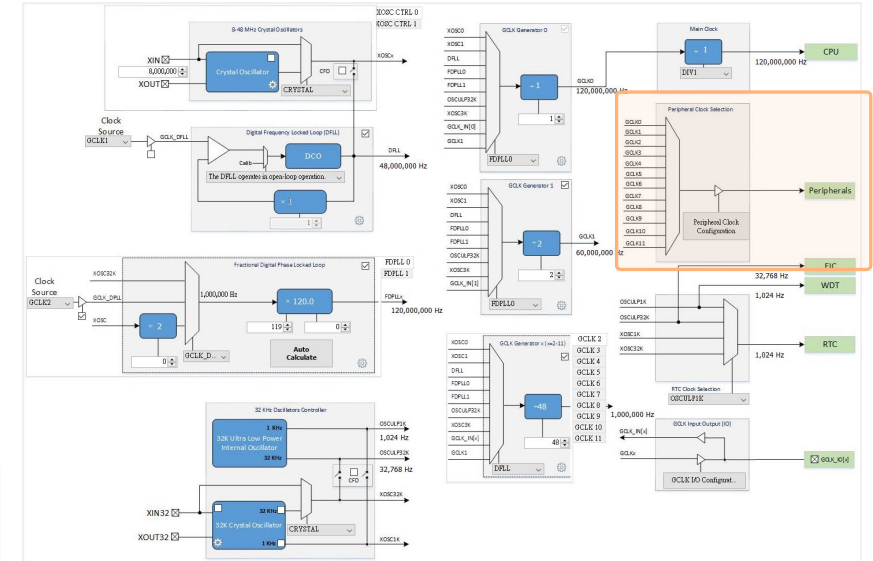
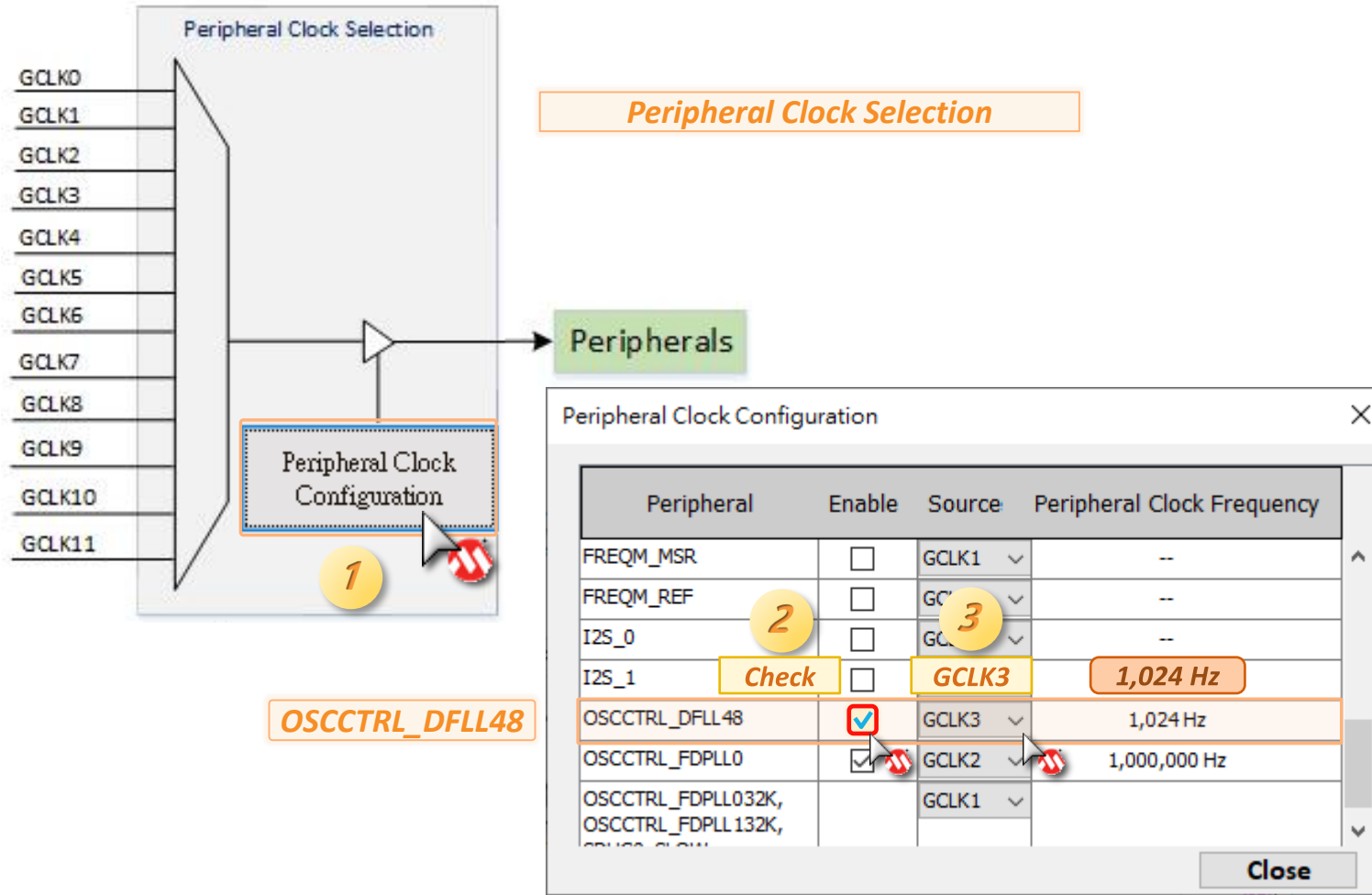


Step 4



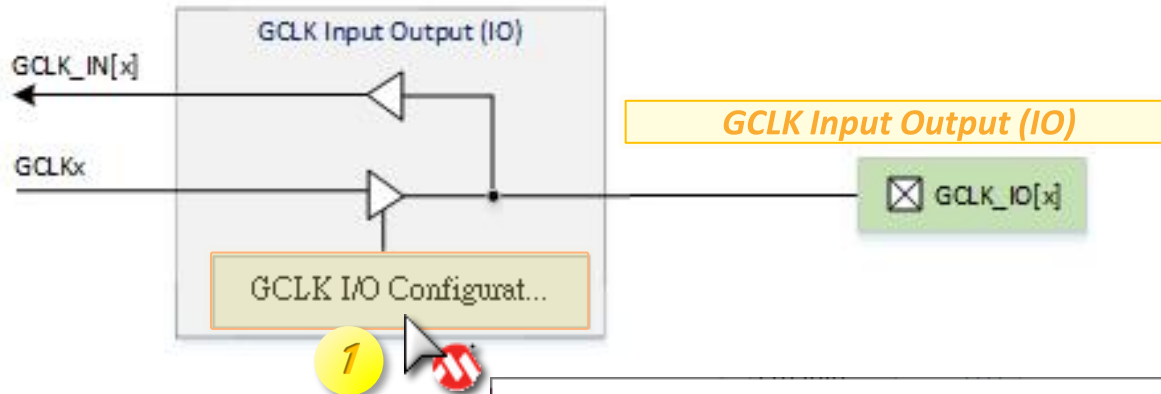
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 4 (DFLL48M alternative method to set the clock source.)



Lab4 System Clock - DFLL Close Loop and Clock Output

Step 5



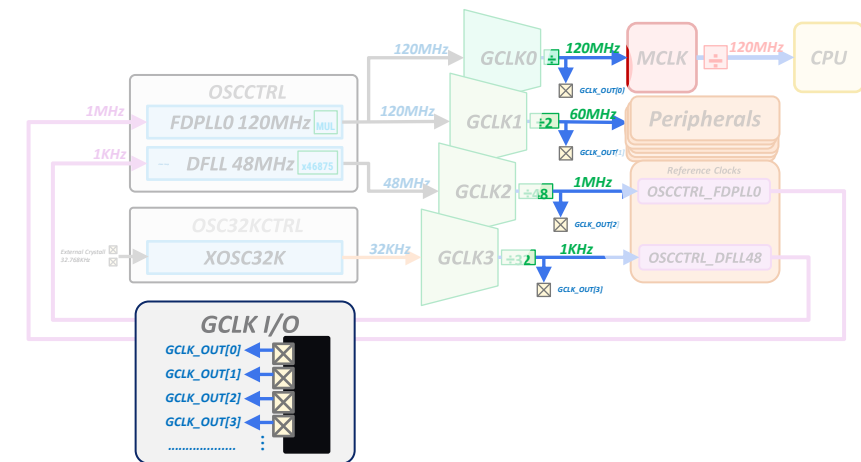
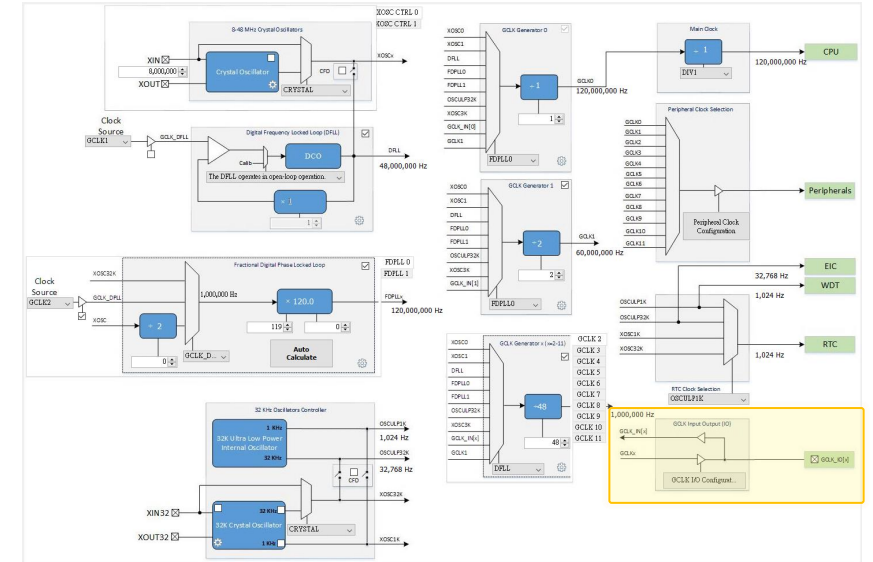
1

GCLK I/O Configuration

Generator	In/Out	In Frequency(Hz)	Out Frequency(Hz)
GCLK0	Out	0	120000000 120 MHz
GCLK1	Out	0	60000000 60 MHz
GCLK2	Out	0	1000000 1 MHz
GCLK3	Out	0	1024 1 kHz
4	In	0	0
5	In	0	0
6	In	0	0
7	In	0	0

2

Close

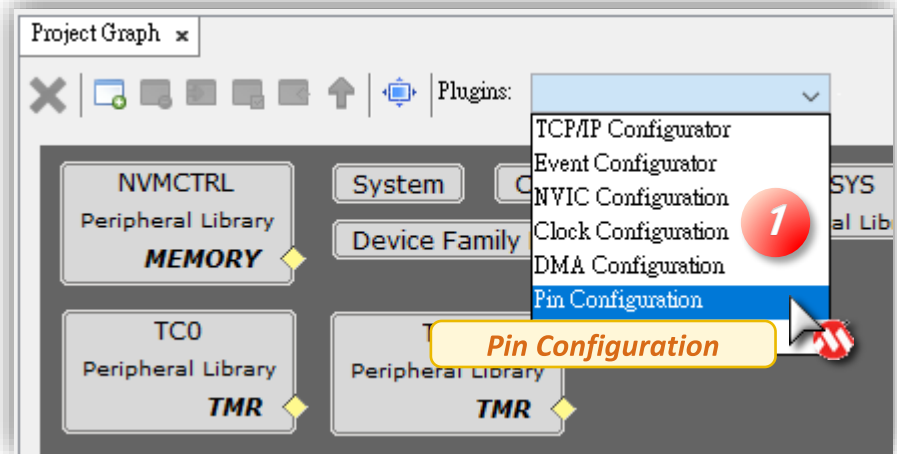


⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 6

- Open **Pin Table** by **Project Graph > Plugins > Pin Configuration**.

- Pin Setting > **Order** : Ports
- Pin Table > double click to enlarge



The screenshot shows the 'Pin Settings' window. At the top, there is a tab labeled 'Pin Settings' with a red circle and the number '2'. Below the tab, there is a dropdown menu labeled 'Order' with 'Ports' selected, indicated by a red circle and the number '3'. To the right of the dropdown is a 'Table View' button and a checked 'Easy View' checkbox. Below this is a table with the following columns: Pin Number, Pin ID, Custom Name, Function, Mode, Direction, Latch, Pull Up, Pull Down, and Drive Strength. The table contains four rows of data for pins 1, 2, 9, and 10.

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
1	PA00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
2	PA01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
9	PA02		Available	Digital	High Impedance	Low	☐	☐	NORMAL
10	PA03		Available	Digital	High Impedance	Low	☐	☐	NORMAL

The screenshot shows the 'Pin Table' window. At the top, there is a tab labeled 'Pin Table' with a red circle and the number '4'. Below the tab, there is a dropdown menu labeled 'Package' with 'TQFP128' selected. Below this is a table with the following columns: Module, Function, and pin numbers 0 through 20. The table contains three rows of data for the 'AC' module.

Module	Function	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AC	AC_AIN0																					
	AC_AIN1																					
	AC_AIN2																					

⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 7

- Right-click on **GCLK** and **View > Ports** to sort by port and **Isolate** for all GCLK I/O pins.

1 Right-Click on GCLK

2 Isolate

4 Isolated GCLK Input Output (IO)

Module	Function	10	11	14	15	16	17	27	30	10	11	12	13	14	15	16	17	18	19	20	21	22	23
GCLK	GCLK_IO0																						
	GCLK_IO1																						
	GCLK_IO2																						
	GCLK_IO3																						
	GCLK_IO4																						
	GCLK_IO5																						
	GCLK_IO6																						
	GCLK_IO7																						

⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 8

- Enable **GCLK I/O** as below to output the clock and **quit** the Isolate view by **Show > All**.

The image shows the Microchip Studio IDE interface for configuring GCLK I/O. On the left, a block diagram shows four GCLK modules (GCLK0, GCLK1, GCLK2, GCLK3) with their respective output frequencies and pin assignments: GCLK0 (120MHz, PB22), GCLK1 (60MHz, PB23), GCLK2 (1MHz, PA16), and GCLK3 (1KHz, PA17). The main window displays the Pin Diagram for a TQFP128 package, showing the pin assignments for GCLK_I/O0 through GCLK_I/O7. A context menu is open over the GCLK_I/O0 pin, showing the 'Show' option selected, with a sub-menu showing 'All' selected. A table at the bottom right summarizes the GCLK configurations.

GCLK	Pin	Clock Output
GCLK_I/O0	PB22	120 MHz
GCLK_I/O1	PB23	60 MHz
GCLK_I/O2	PA16	1 MHz
GCLK_I/O3	PA17	1 kHz

GCLK I/O Configuration

Generator	In/Out	In Frequency(Hz)	Out Frequency(Hz)
0	Out	0	120000000
1	Out	0	60000000
2	Out	0	1000000
3	Out	0	1024

⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 9

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Generate' button highlighted with a red circle and a yellow 'Generate' label. The 'Output' window shows the results of the generation process, including a green box indicating 'Generation complete (total time: 5396 milliseconds)' and a list of HARMONY events.

Output Window Content:

```
15:17:50.827 INFO: Generation complete.
15:17:50.843 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:50.844 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:50.844 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

Lab4 System Clock - DFLL Close Loop and Clock Output

Step 10

- This Lab does not require any program modification.

```
#include <stddef.h>
#include <stdbool.h>
#include <stdlib.h>
#include "definitions.h"

uint32_t i = 0;

void TC0_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if (status & TC_INTFLAG_OVF_Msk)
    {
        LED1_Toggle();
    }
}

void TC1_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if (status & TC_INTFLAG_OVF_Msk)
    {
        LED2_Toggle();
    }
}

int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    TC0_TimerCallbackRegister(TC0_TimerExpired, (uintptr_t) NULL);
    TC0_TimerStart();
    TC1_TimerCallbackRegister(TC1_TimerExpired, (uintptr_t) NULL);
    TC1_TimerStart();
```

```
while ( true )
{
    if ( TC1_TimerPeriodHasExpired() )
    {
        LED2_Toggle();
    }

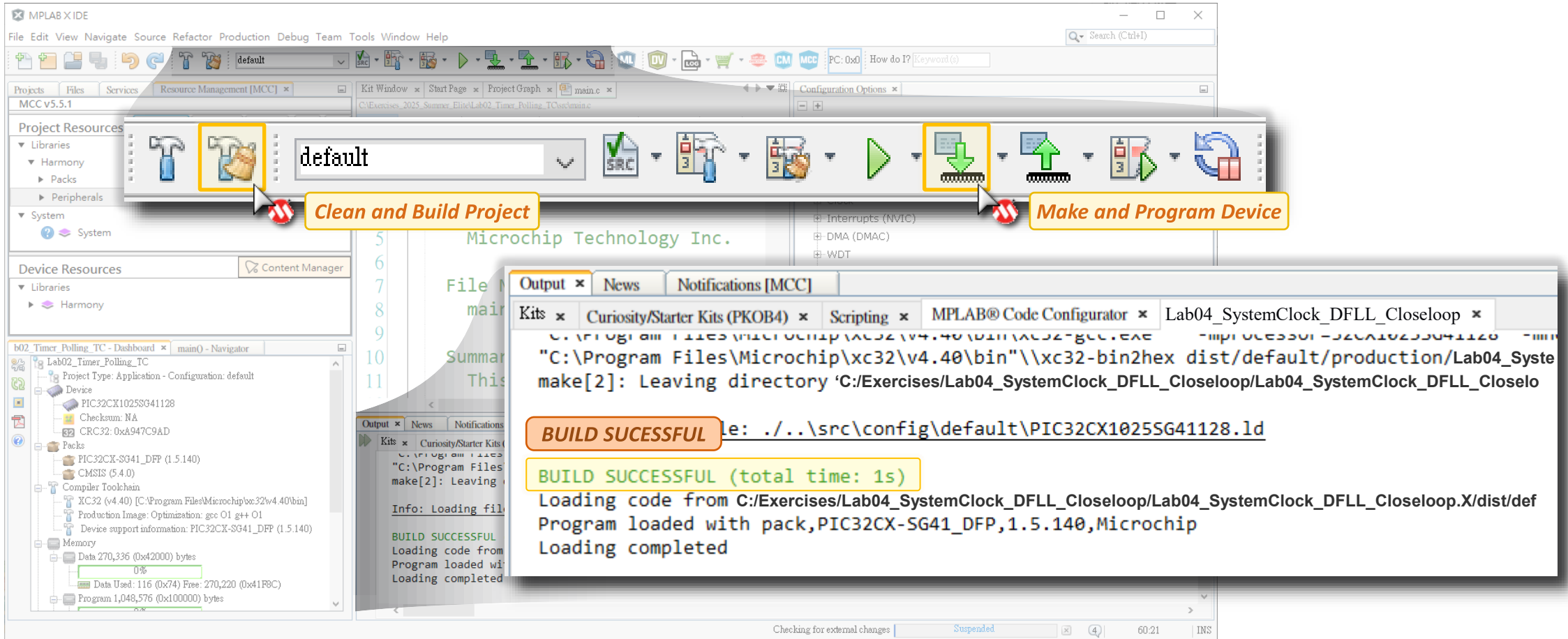
    if ( BT1_Get() ) LED3_Clear();
    else LED3_Set();
}

return ( EXIT_FAILURE );
}
```


⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 11

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains several icons, with two highlighted by yellow boxes and red arrows. The first icon, a hammer and wrench, is labeled "Clean and Build Project". The second icon, a green arrow pointing down to a circuit board, is labeled "Make and Program Device".

The "Output" window is open, showing the build process. The output text includes:

```
make[2]: Leaving directory 'C:/Exercises/Lab04_SystemClock_DFLL_Closetloop/Lab04_SystemClock_DFLL_Closetloop'

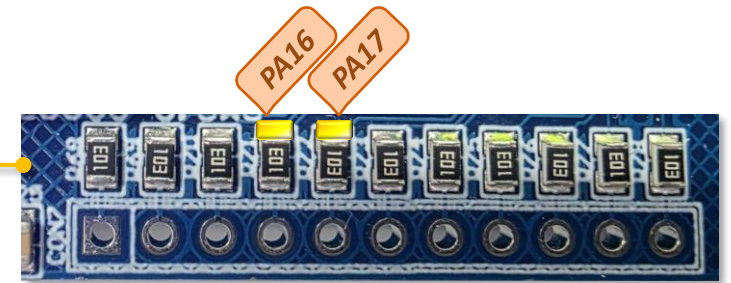
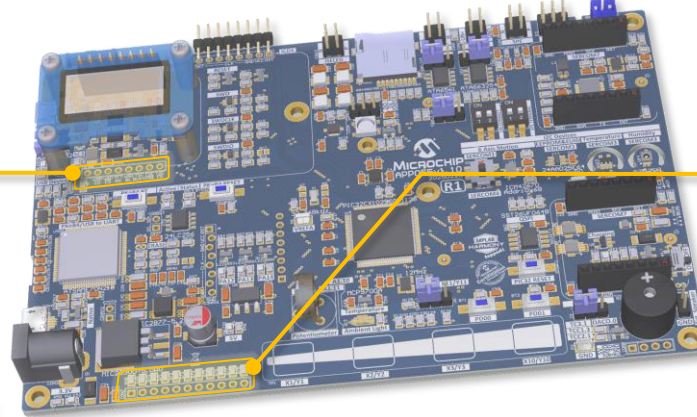
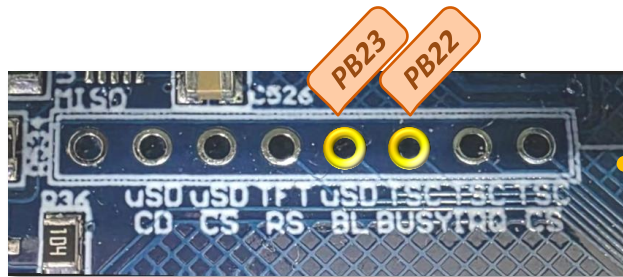
BUILD SUCCESSFUL le: ../../src/config/default/PIC32CX1025SG41128.ld

BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab04_SystemClock_DFLL_Closetloop/Lab04_SystemClock_DFLL_Closetloop.X/dist/default
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```


⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 12

- Locate the GCLK clock output pins on EVB APP055.



SOCKET3
OELD-APP050-Combine Socket

Pin	Signal	Pin	Signal
1	MOSI/SDA	16	PB01/MISO(S5.3)
2	SCK/SCL	15	PC31/uSD CD(INT14)
3	GND/A0	14	PC30/uSD nCS
4	OLED_RST	13	PA27/TFT RS
5	OLED_CS	12	PB23/TFT BL
6	3.3V	11	PB22/TSC BUSY(INT6)
7	5V	10	PC26/TSC nIRQ(INT10)
8	GND	9	PC25/TSC nCS

Monochrome OLED & Display Socket

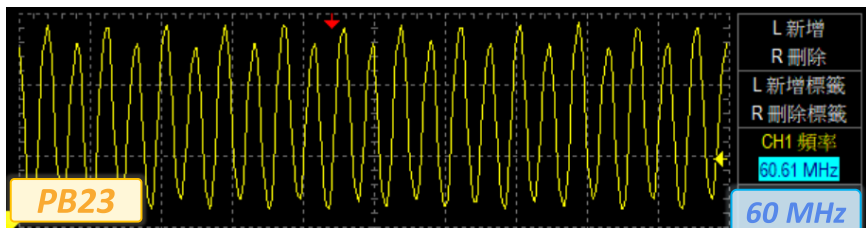
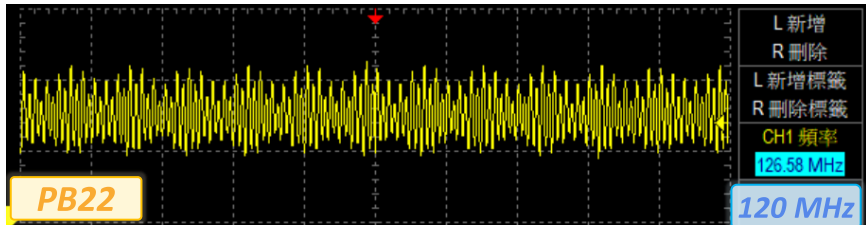
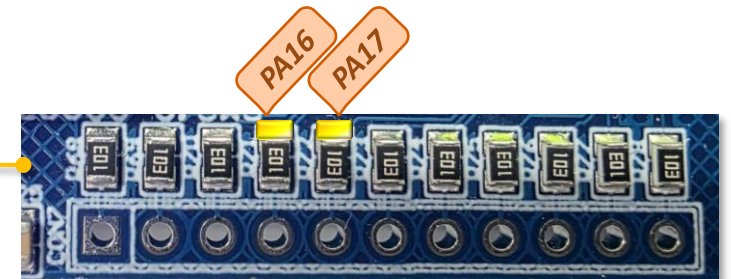
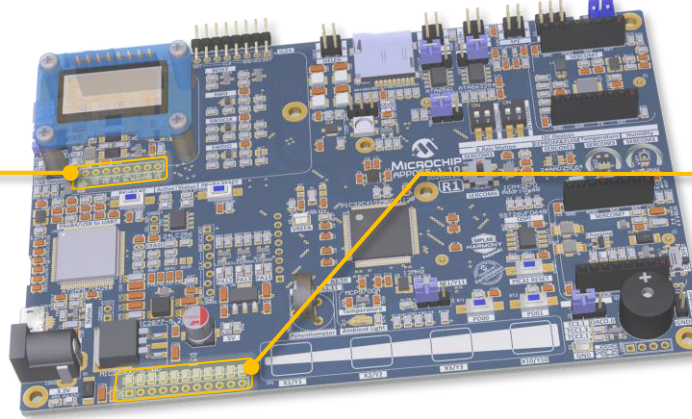
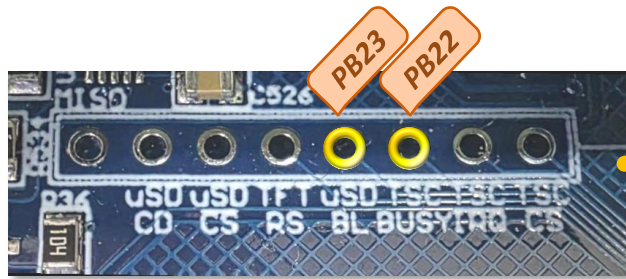
GCLK	Pin	Clock Output
GCLK_IO0	PB22	120 MHz
GCLK_IO1	PB23	60 MHz
GCLK_IO2	PA16	1 MHz
GCLK_IO3	PA17	1 kHz

PA17/DS(TC2.1)/X11/Y11				CON7 SC-11
PB08/X1/Y1	R68	10K	1	
PB09/X2/Y2	R69	10K	2	
PA04/X3/Y3	R70	10K	3	
PA16/X10/Y10	R71	10K	4	
PA17/X11/Y11	R72	10K	5	
PA18/X12/Y12	R73	10K	6	
PA19/X13/Y13	R74	10K	7	
PB04/X22/Y22	R75	10K	8	
PB05/X23/Y23	R76	10K	9	
PB06/X24/Y24	R77	10K	10	
PB07/X25/Y25	R78	10K	11	

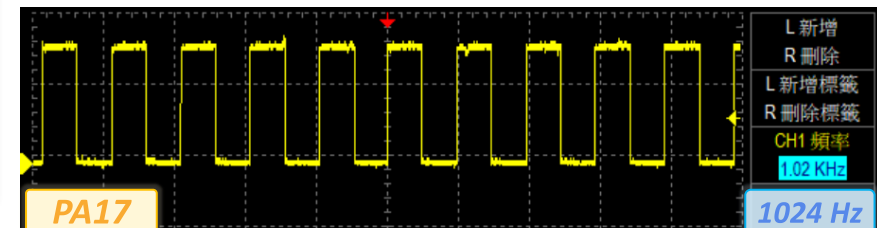
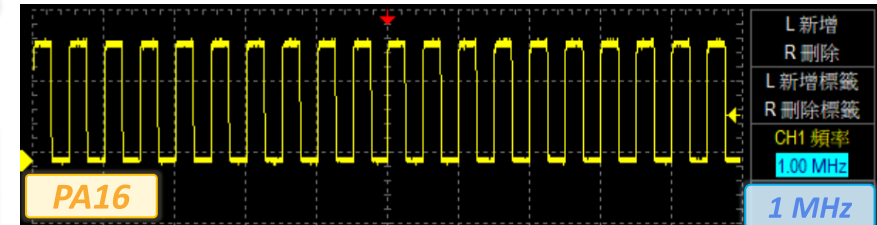
✖ Lab4 System Clock - DFLL Close Loop and Clock Output

Result

- Use an oscilloscope to observe the waveform of GCLK output and measure frequency.



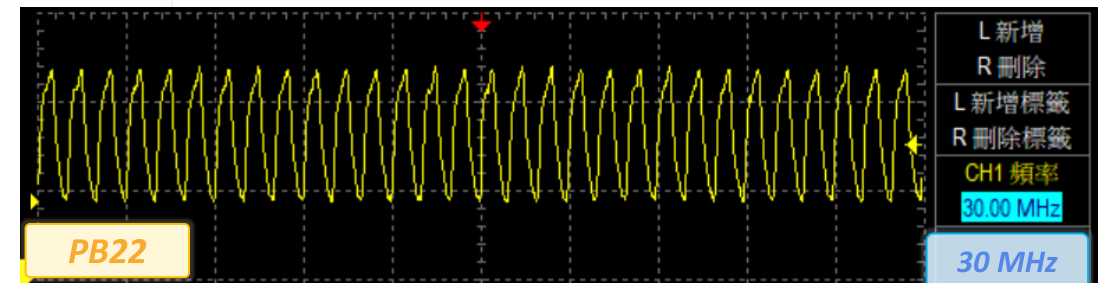
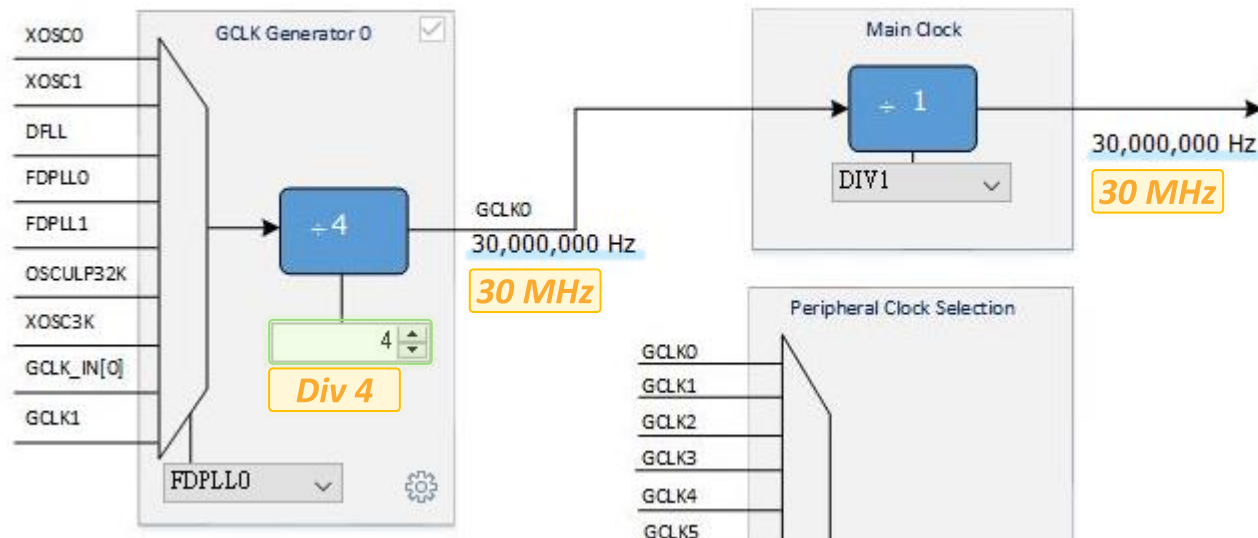
GCLK	Pin	Clock Output
GCLK_IO0	PB22	120 MHz
GCLK_IO1	PB23	60 MHz
GCLK_IO2	PA16	1 MHz
GCLK_IO3	PA17	1 kHz



⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Appendix

- Due to the circuit filtering effect, the oscilloscope cannot measure higher frequency signals. We try to divide the frequency of GCLK0 by 4 to get a lower 30MHz and observe it again.

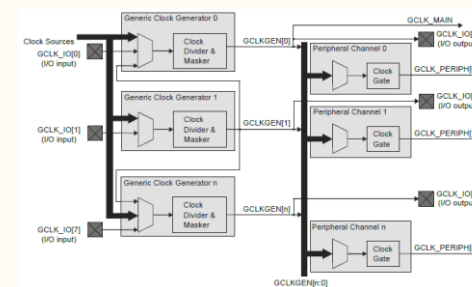
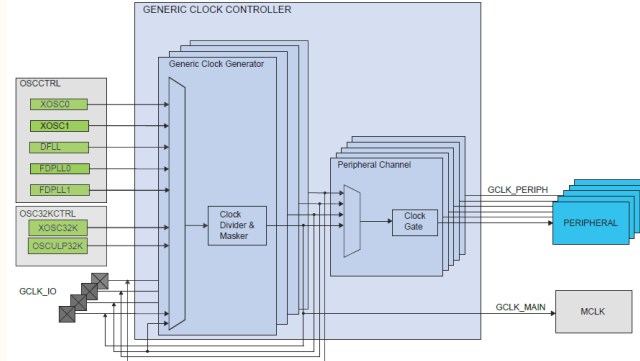
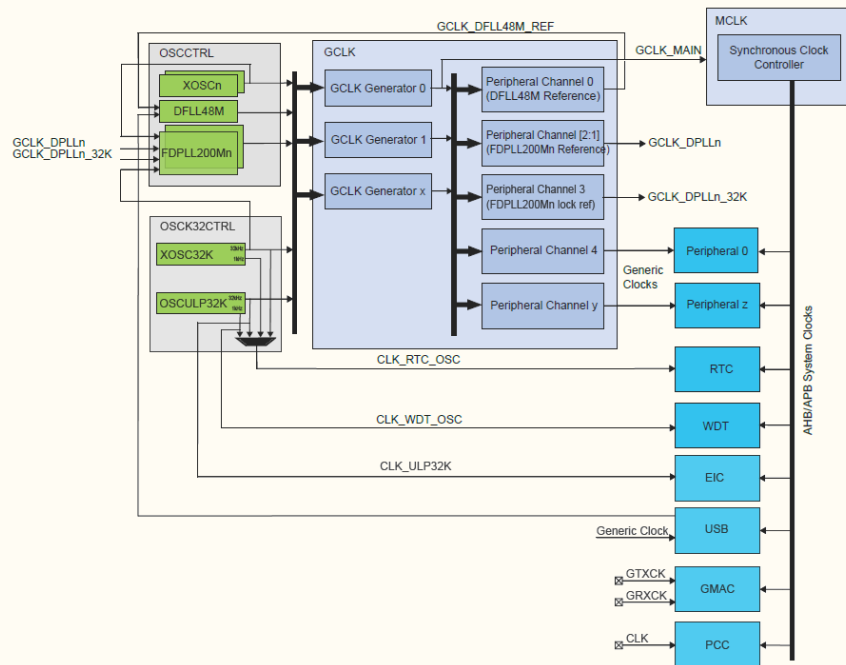


Appendix : Oscillator and Clock Generator

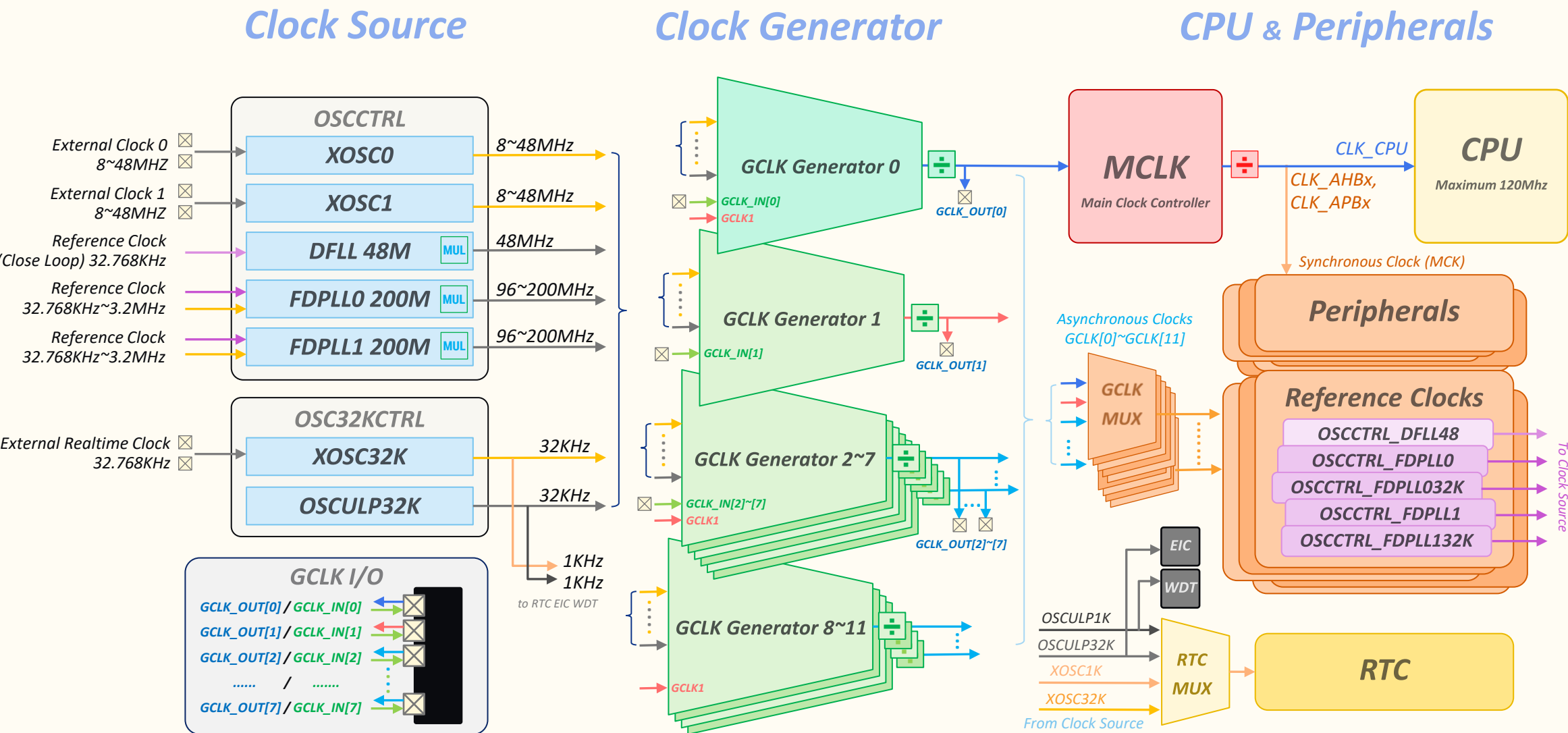
New Clock Configurator GUI

Clock System

- Multiple clock sources managed by OSCCTRL/OSC32KCTRL include External and DFLL/FDPLL.
- 12 Generic Clock Generators (GCLKs) create configurable clock signals from these sources.
- GCLKs feed peripheral channels and DFLL input.
- GCLK0 dedicate provides the MCLK (maximum 120MHz) for CPU.
- MCLK to generate synchronous clocks for CPU, AHB/APB buses, and peripherals.
- MCLK also includes prescalers and clock masks for efficient power and clock control.

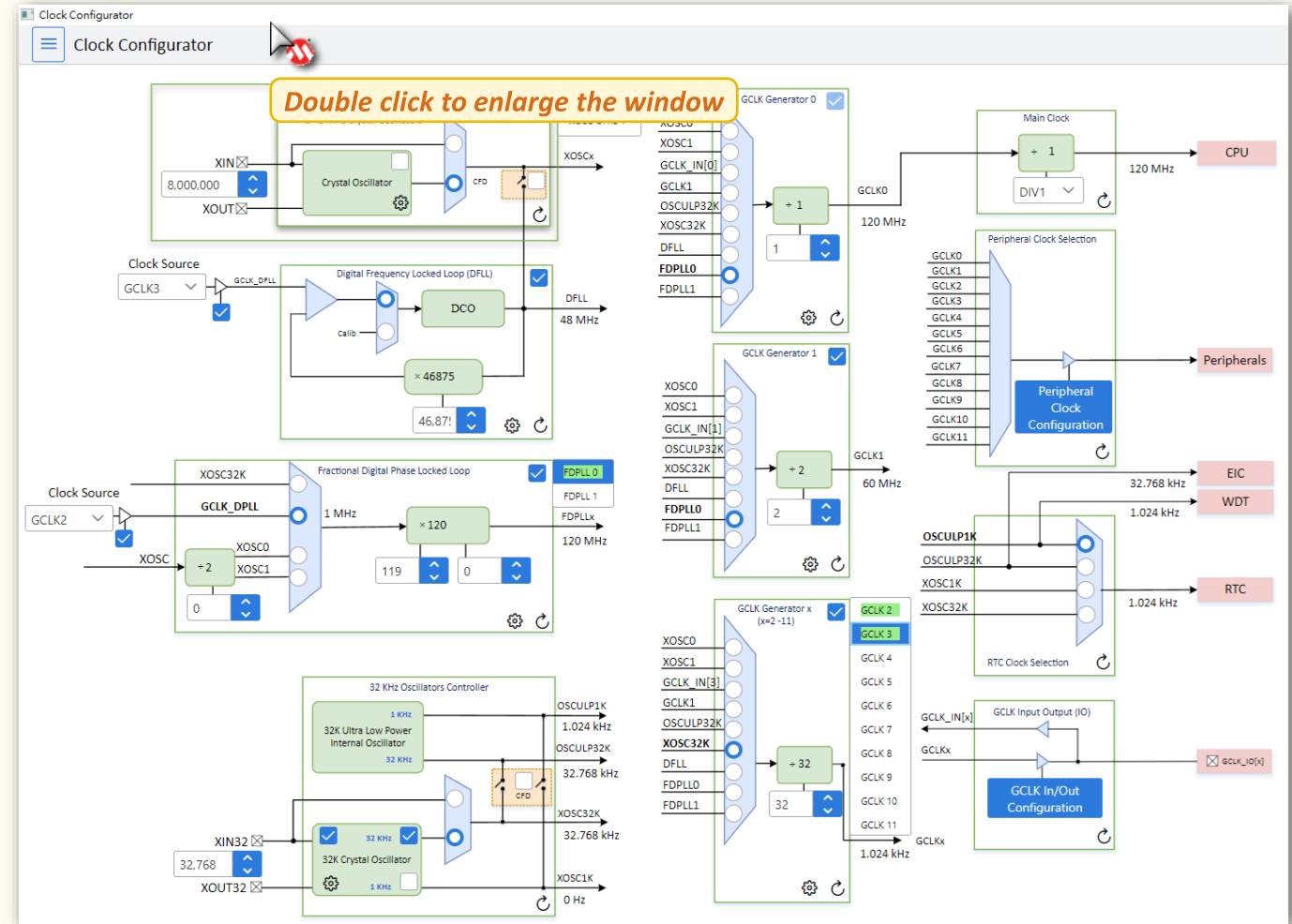
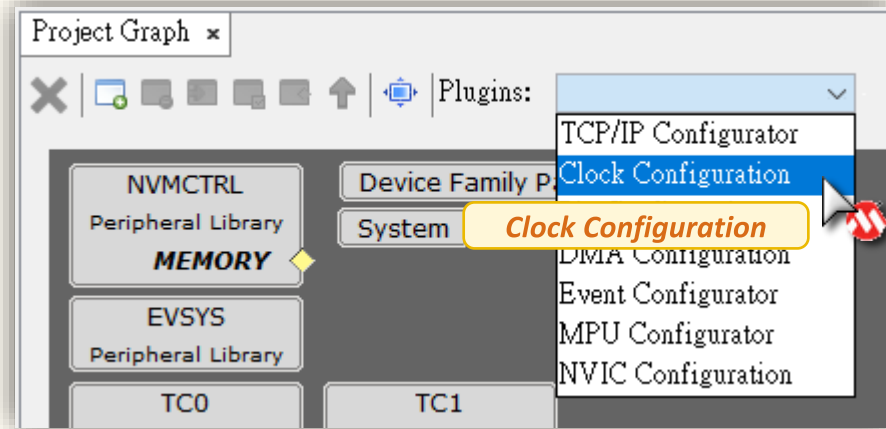


Clock System



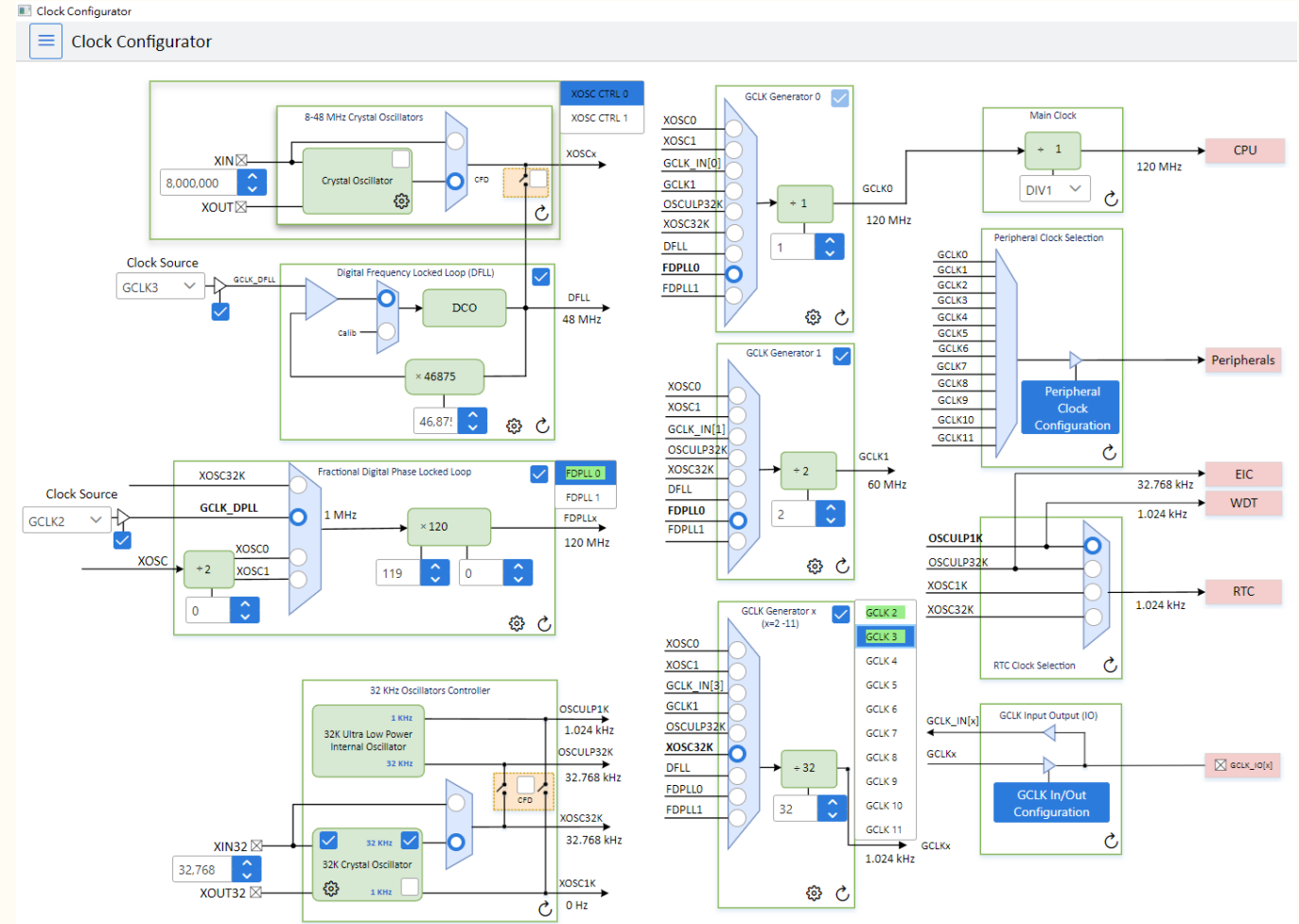
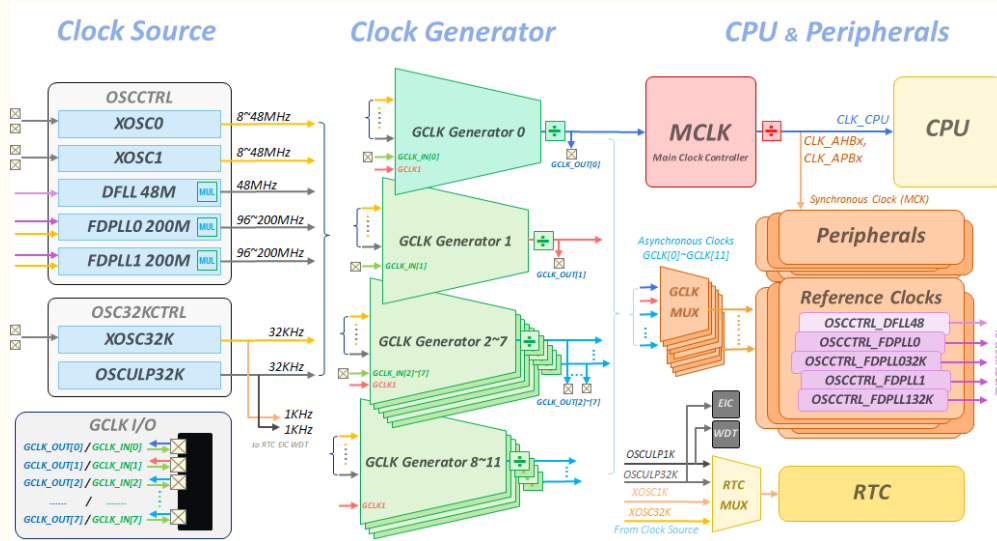
Clock Setting Use MCC Harmony

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



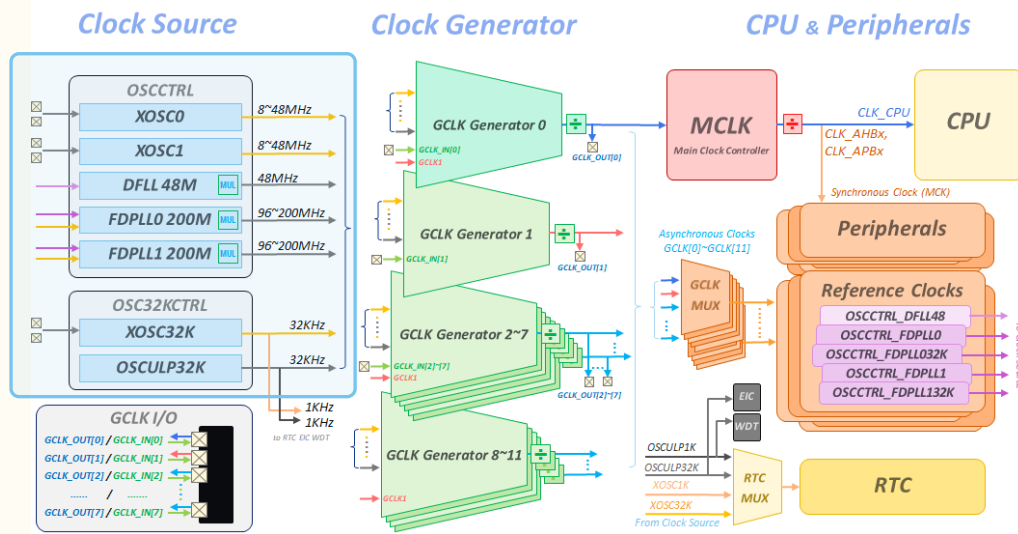
Clock Setting Use MCC Harmony

- The Clock Easy View GUI layout.

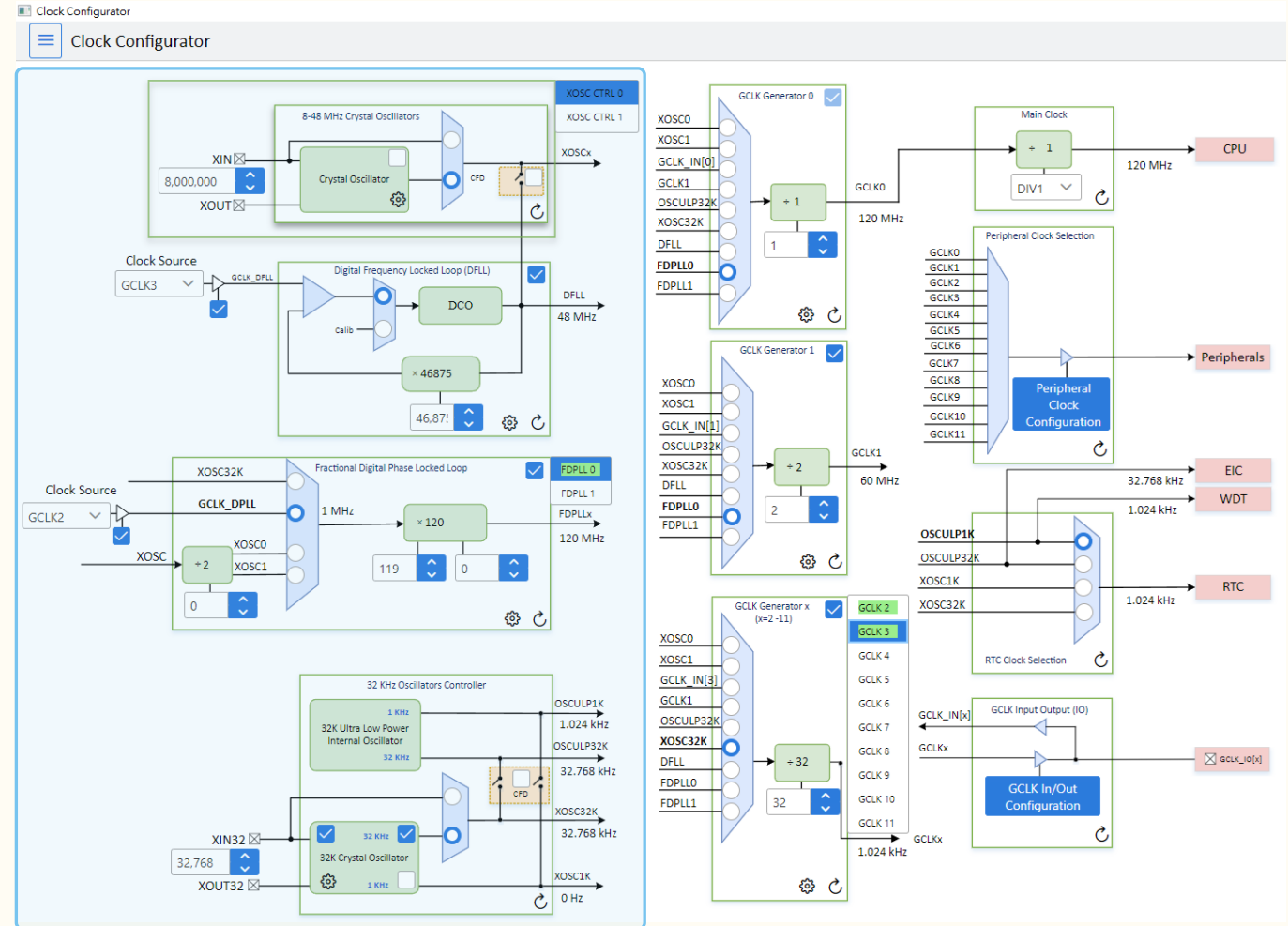


Clock Setting Use MCC Harmony

• Clock Source

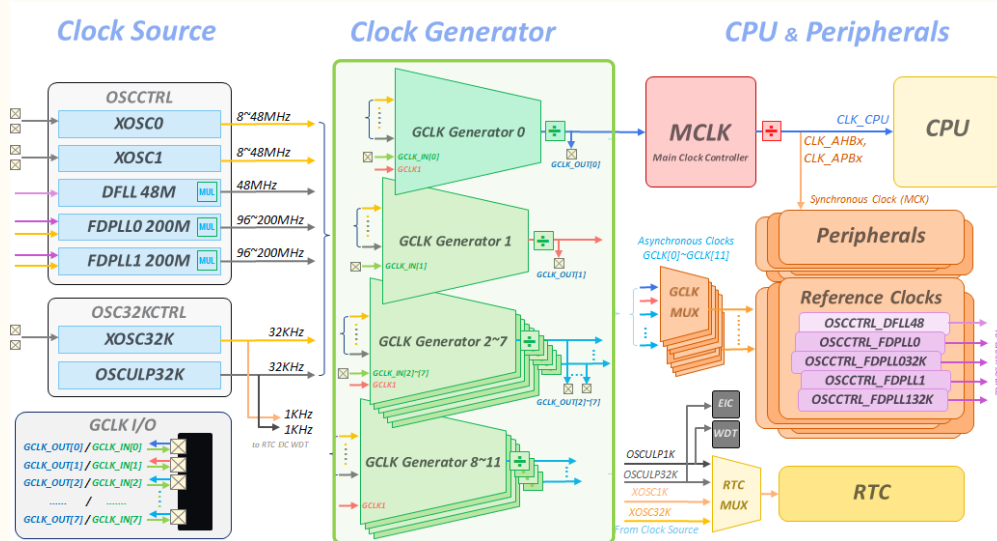


Clock Source

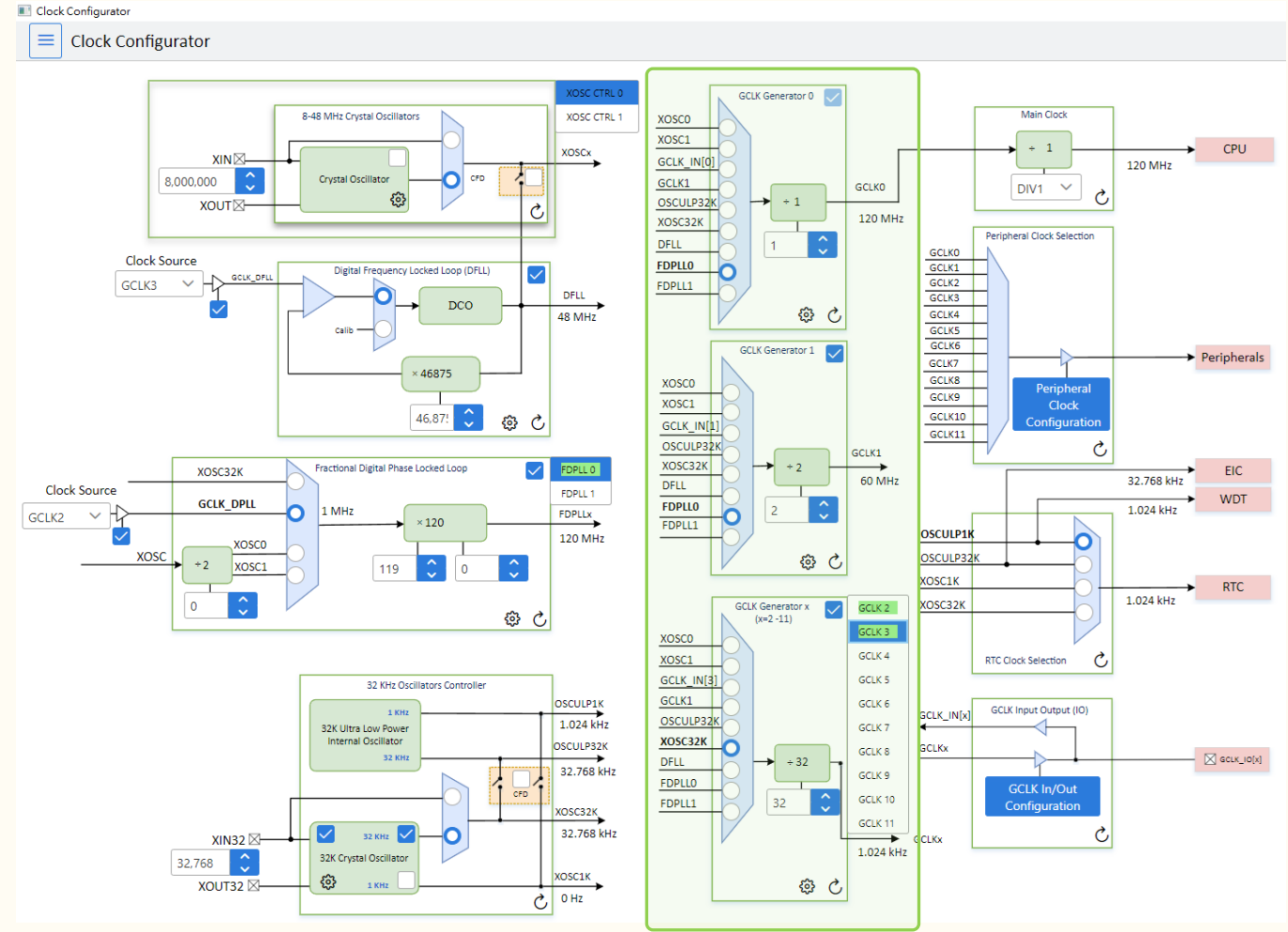


Clock Setting Use MCC Harmony

• Clock Generator



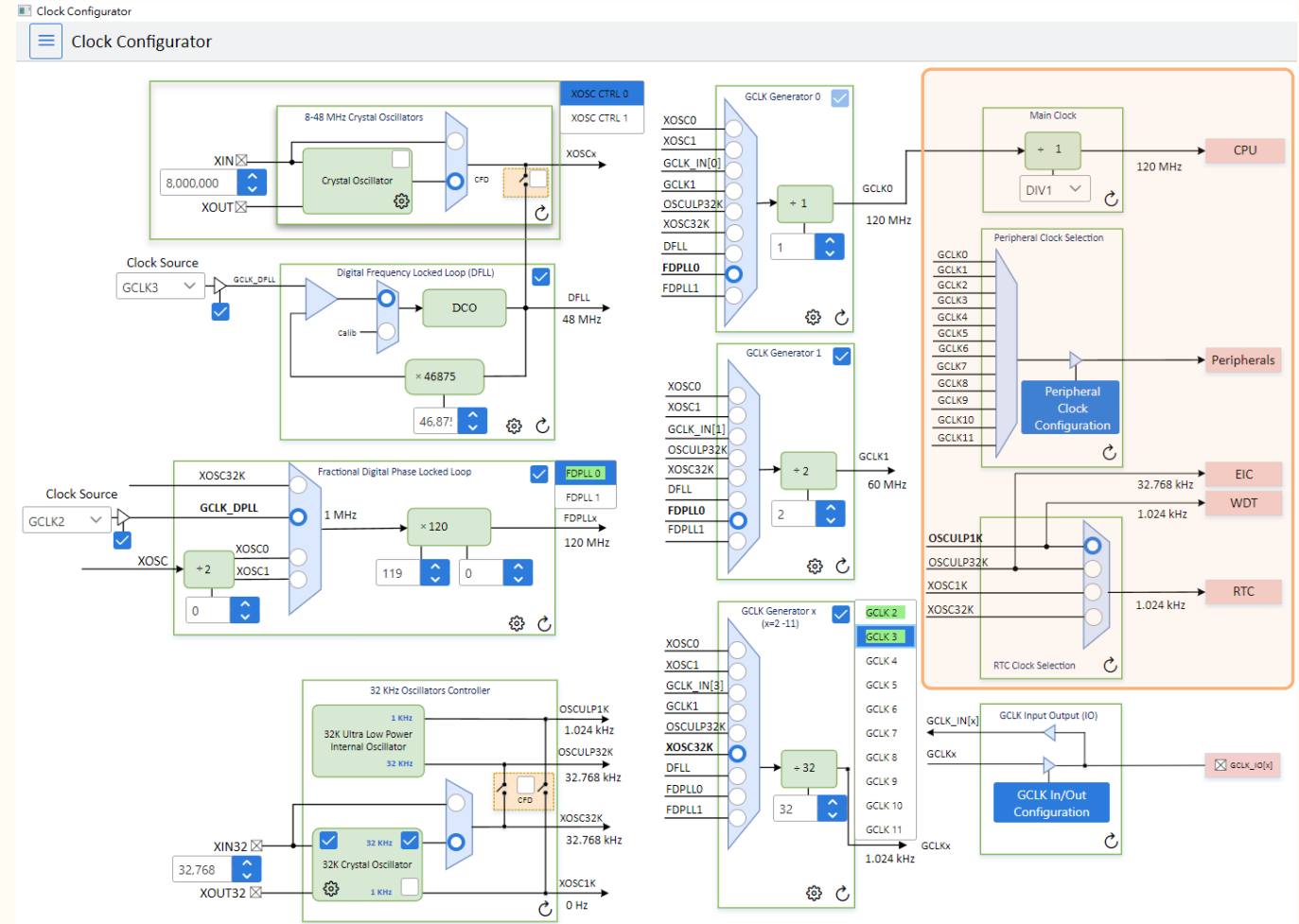
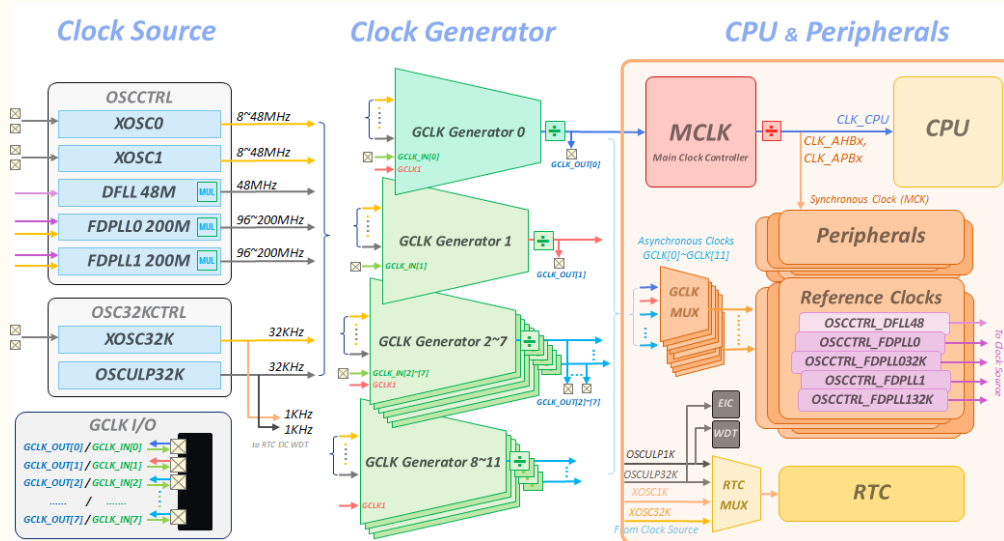
Clock Generator



Clock Setting Use MCC Harmony

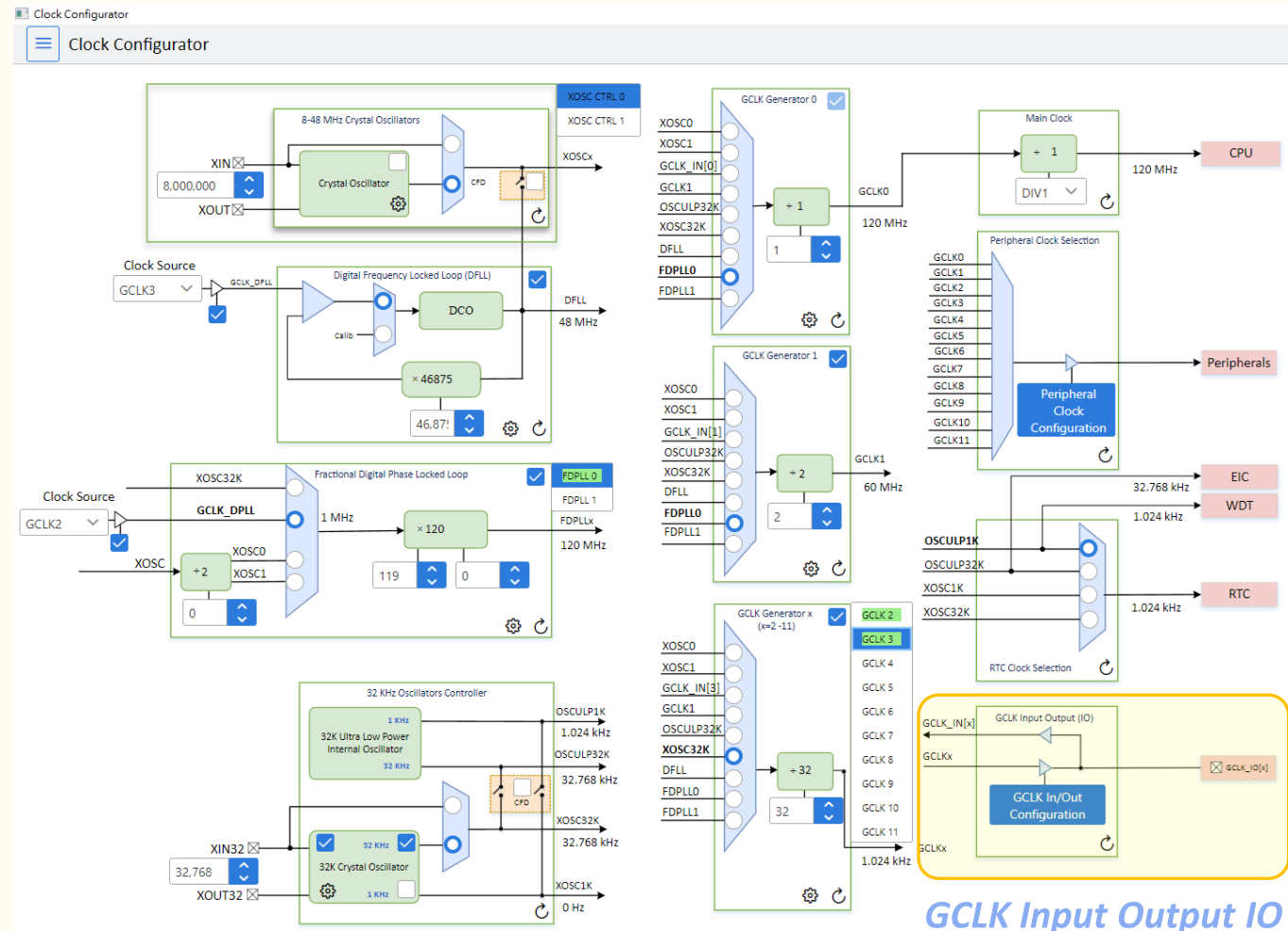
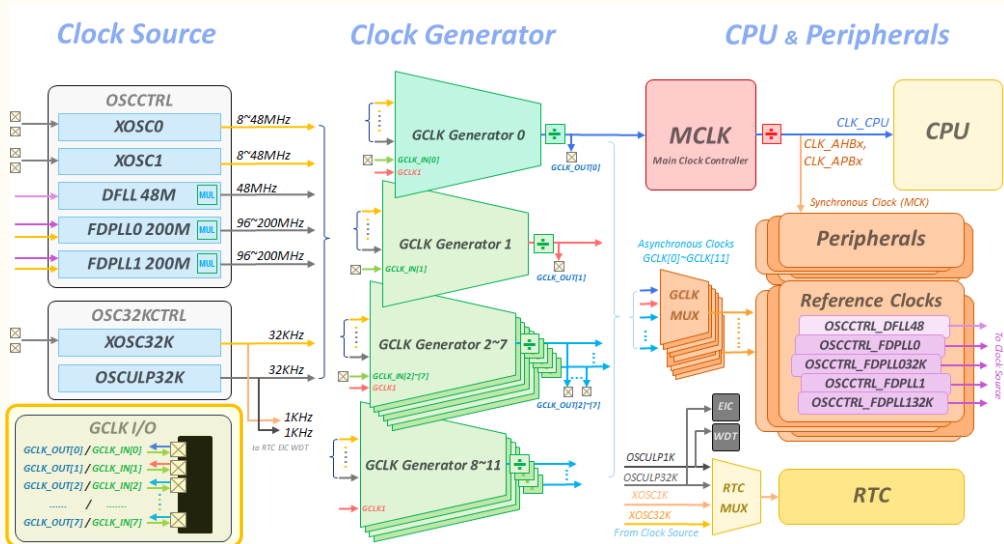
• CPU & Peripherals

CPU & Peripherals



Clock Setting Use MCC Harmony

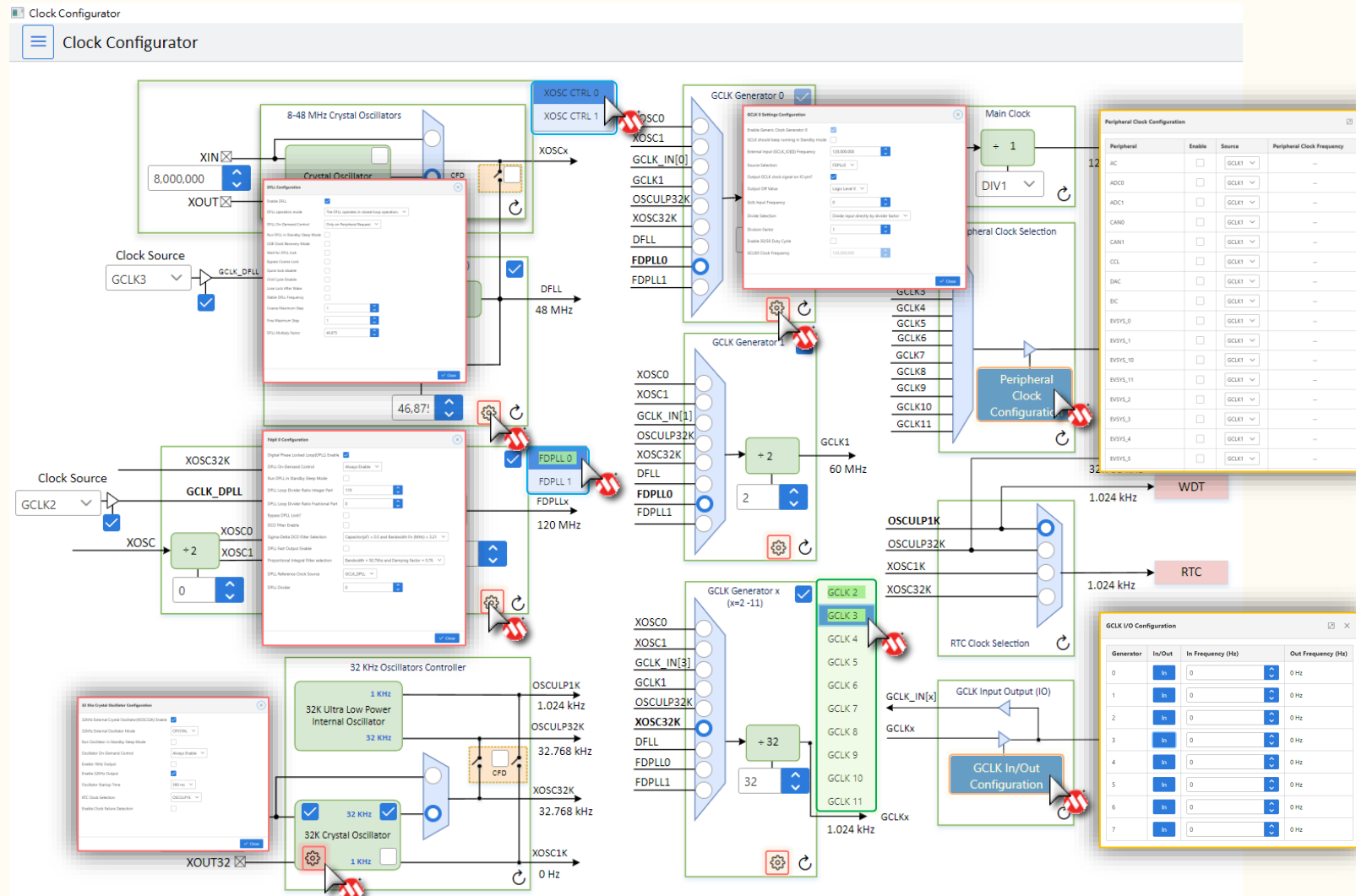
- GCLK Input Output (IO)



GCLK Input Output IO

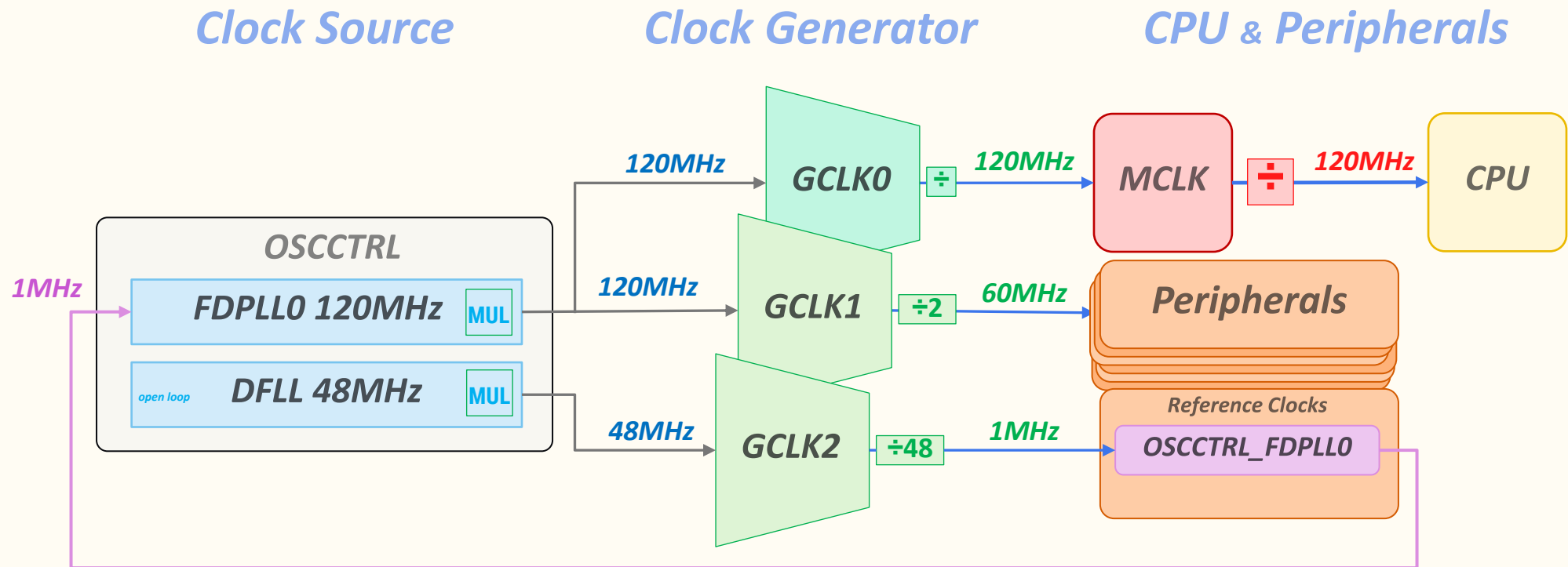
Clock Setting Use MCC Harmony

- The Multiple Modules Selection and Extra Configure Dialog.



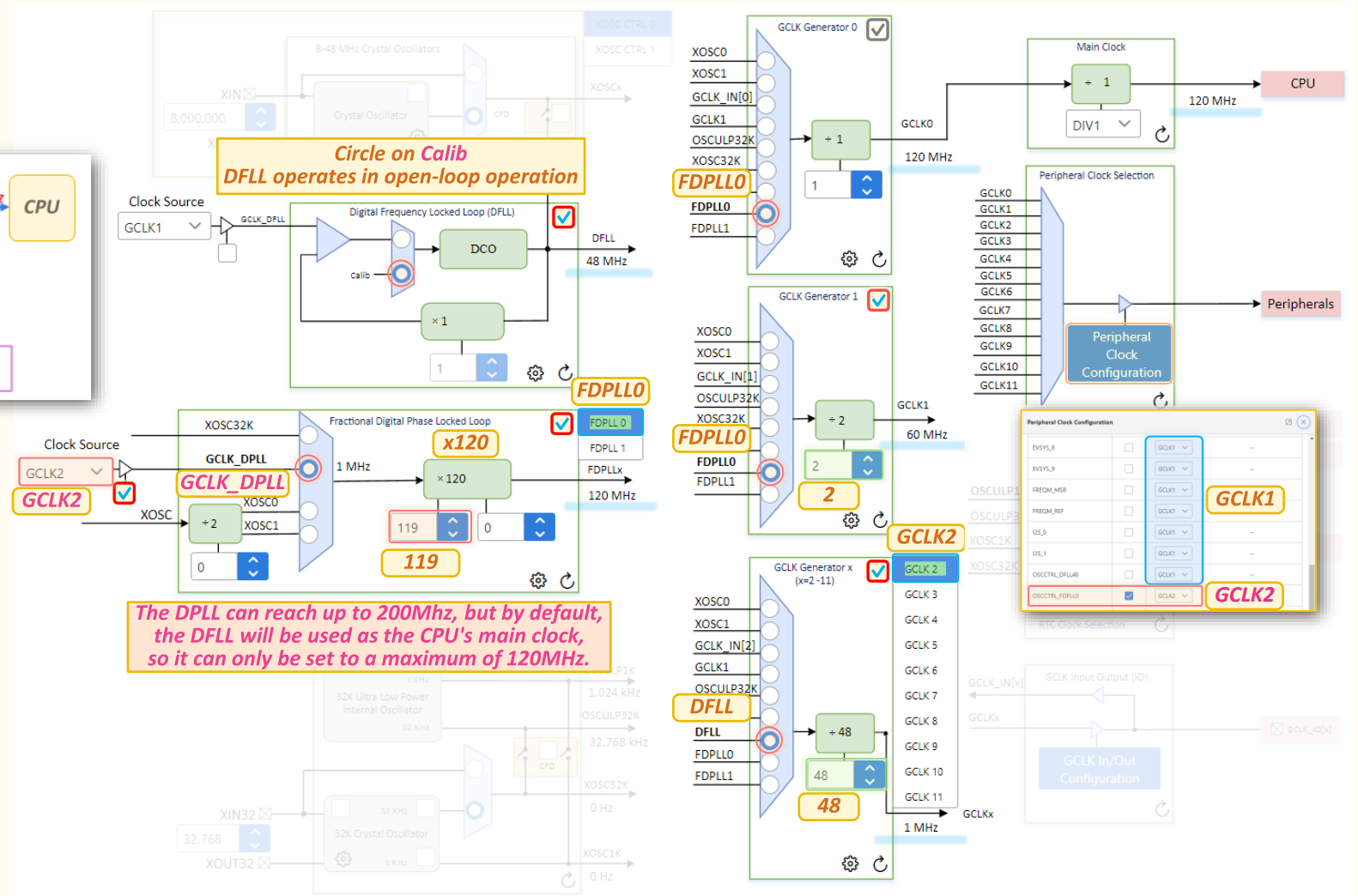
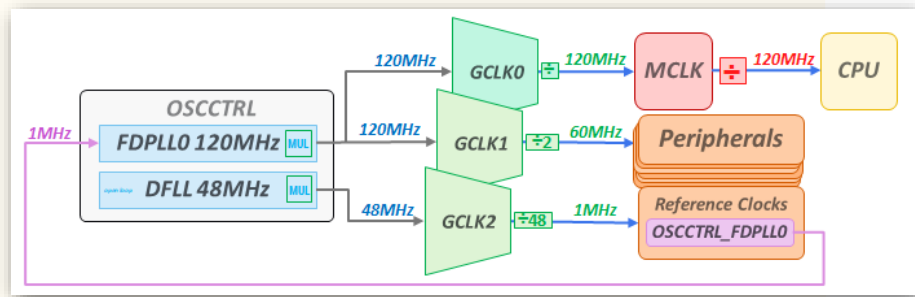
Clock Setting Use MCC Harmony

- Default System Clock routing of a new project.
 - DFLL48MHz (open loop) > GCLK2 (DIV48) > FDPLL0 120MHz > GCLK0 > MCLK 120MHz
 - DFLL48MHz (open loop) > GCLK2 (DIV48) > FDPLL0 120MHz > GCLK1(DIV2) > Peripherals 60MHz



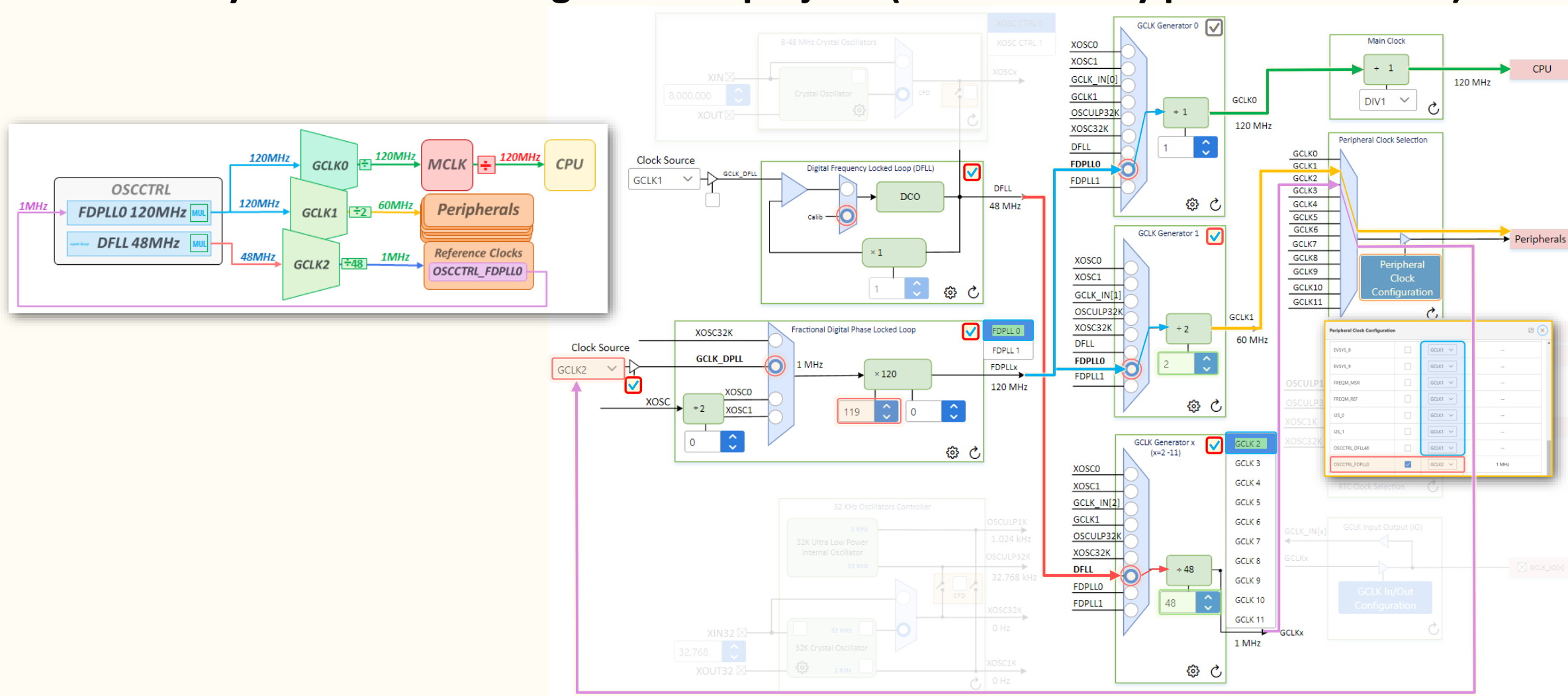
Clock Setting Use MCC Harmony

- Default System Clock routing of a new project. (MCC Harmony Configuration)



Clock Setting Use MCC Harmony

- Default System Clock routing of a new project. (MCC Harmony pseudo connect)



Lab4 System Clock

DFLL Close Loop and Clock Output

Lab4 System Clock - DFLL Close Loop and Clock Output

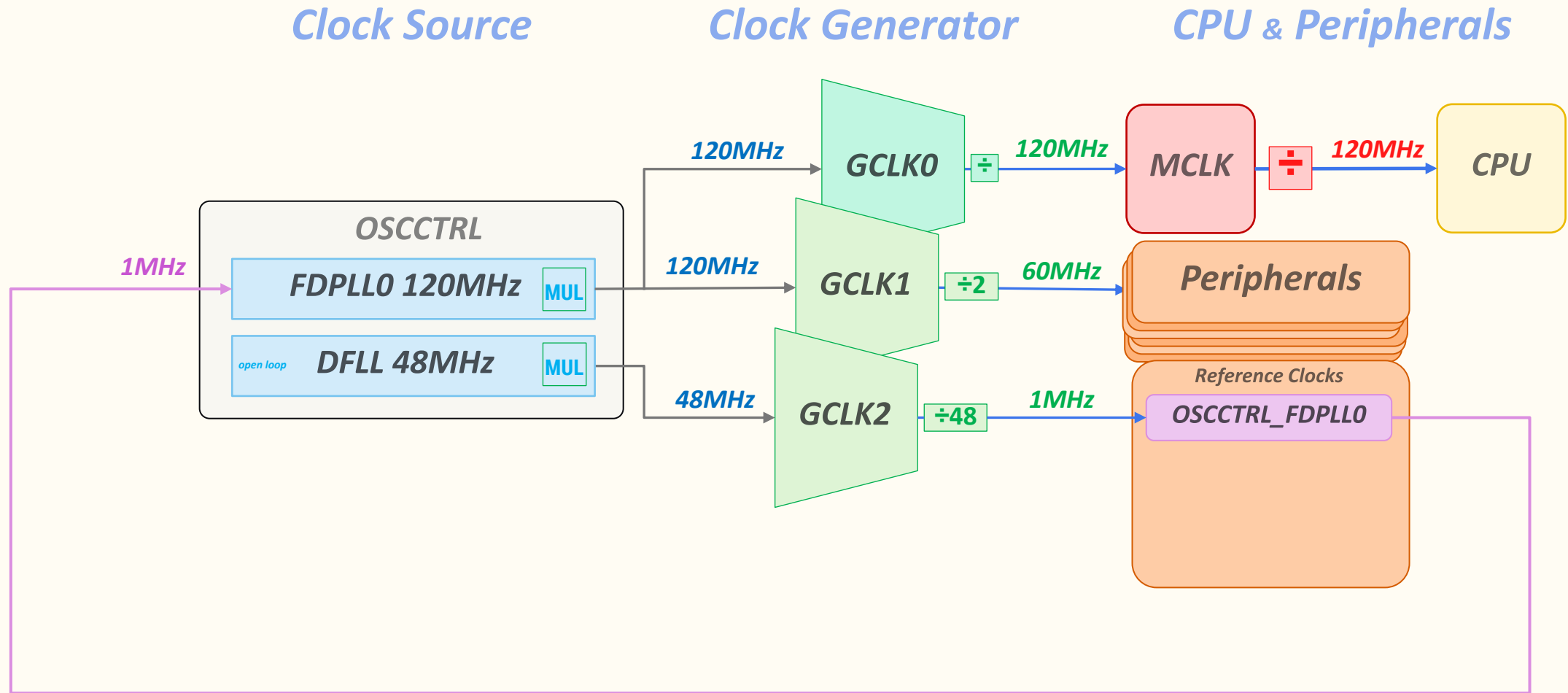
- The current system clock source is generated by DFLL48M using Open Loop, which is an internal clock source with an RC oscillator and a Digital Frequency-Locked Loop. Therefore, the frequency itself is less accurate and is prone to larger errors as the temperature changes.
- In this experiment, our goal is to achieve the Close Loop of DFLL48M and use an external 32K oscillator crystal as the Reference Frequency of the DFLL48M to achieve higher frequency accuracy.
- The following are the steps of clock routing:
 1. Enable the external XOSC32K clock source. (APP055 has a on-boarded external 32K oscillator)
 2. Use XOSC32K as the clock source of GCLK3 and divide it by 32 to get a 1KHz frequency.
 3. GCLK3 now output 1KHz to reference clock peripheral OSCCTRL_DFLL48 as the clock source of DFLL48M.
 4. Enable the Close Loop of DFLL48M and input the Multiply Factor as 46,875 to calculate a more accurate 48MHz frequency.

The following parts maintain the original clock configuration.

Let's go !

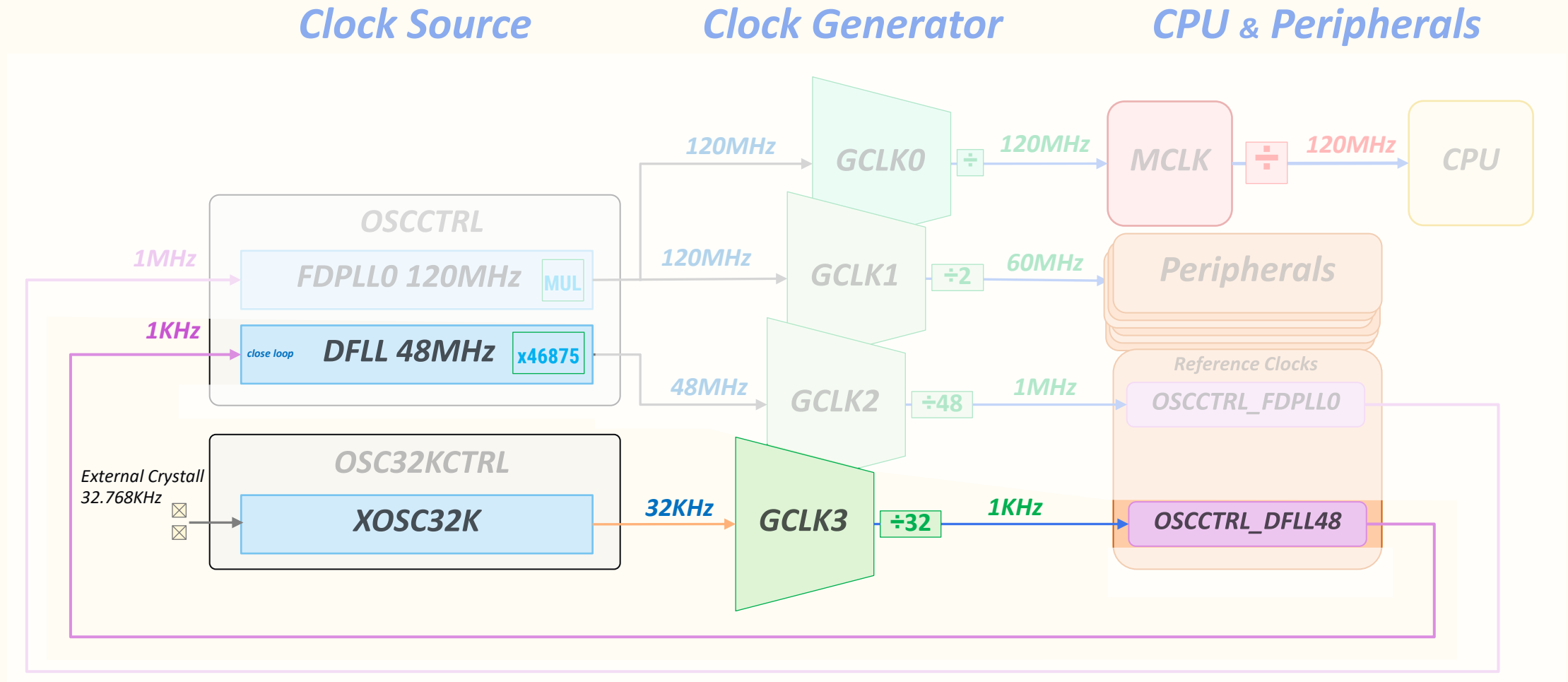
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

- The default configuration of the system clock routing.



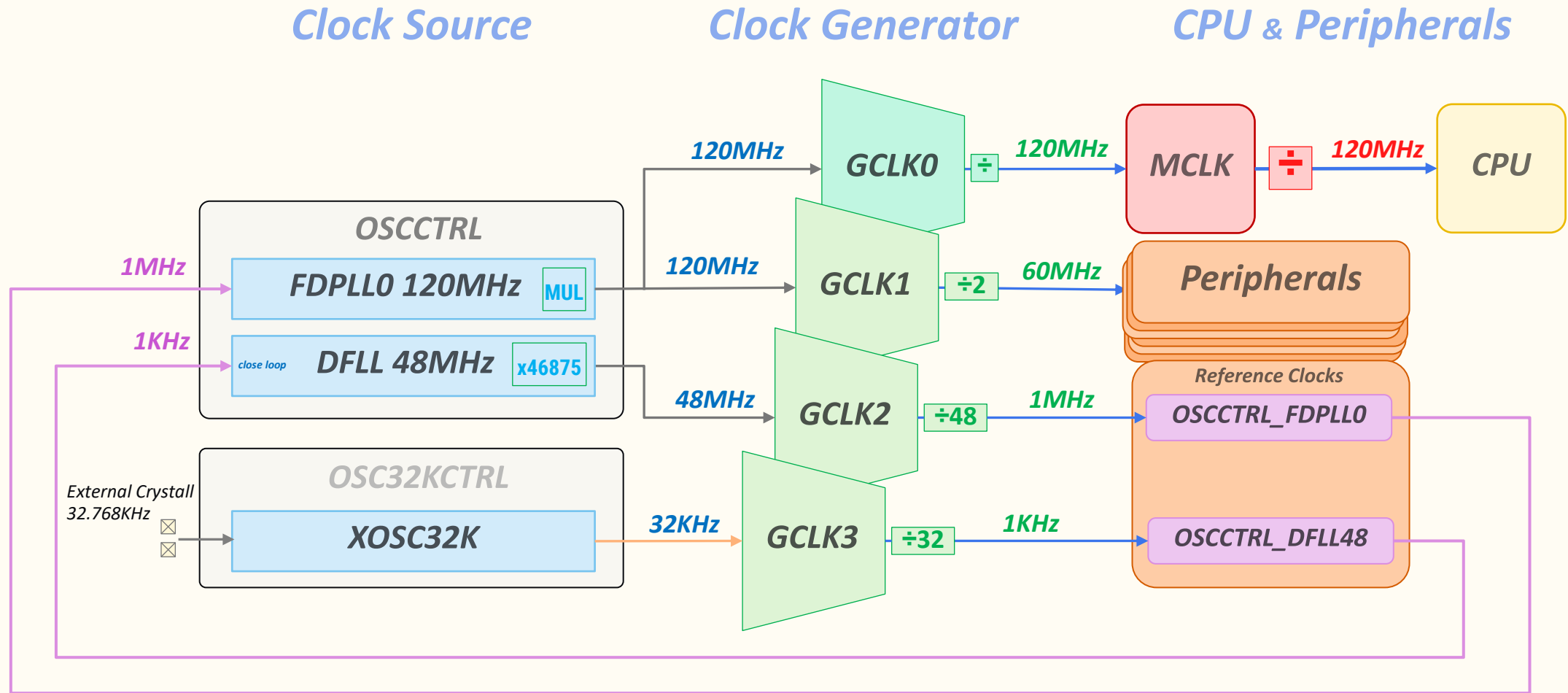
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

- The new configuration we are going to add and modify.



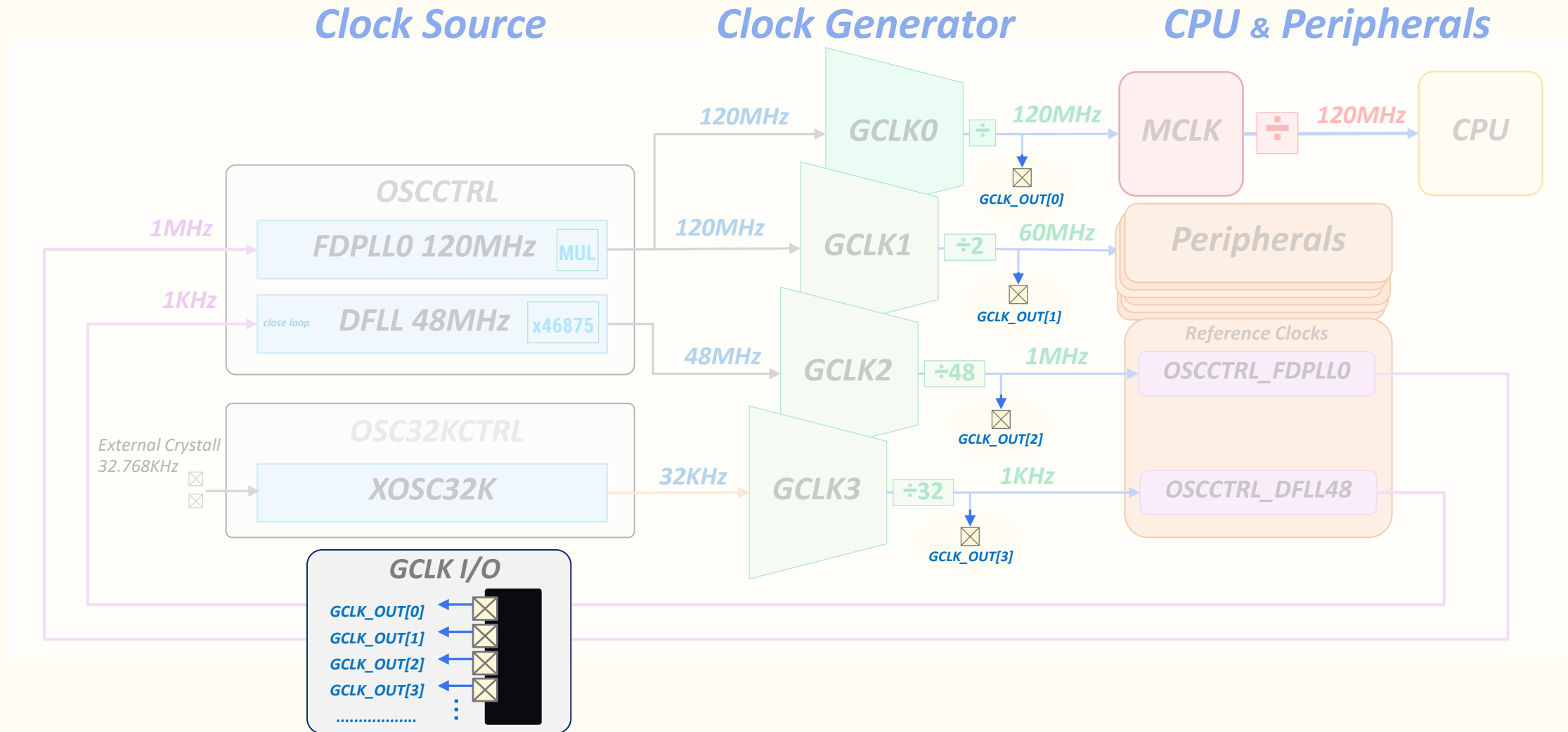
⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

- The configuration of the system clock routing we want to implement.



⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

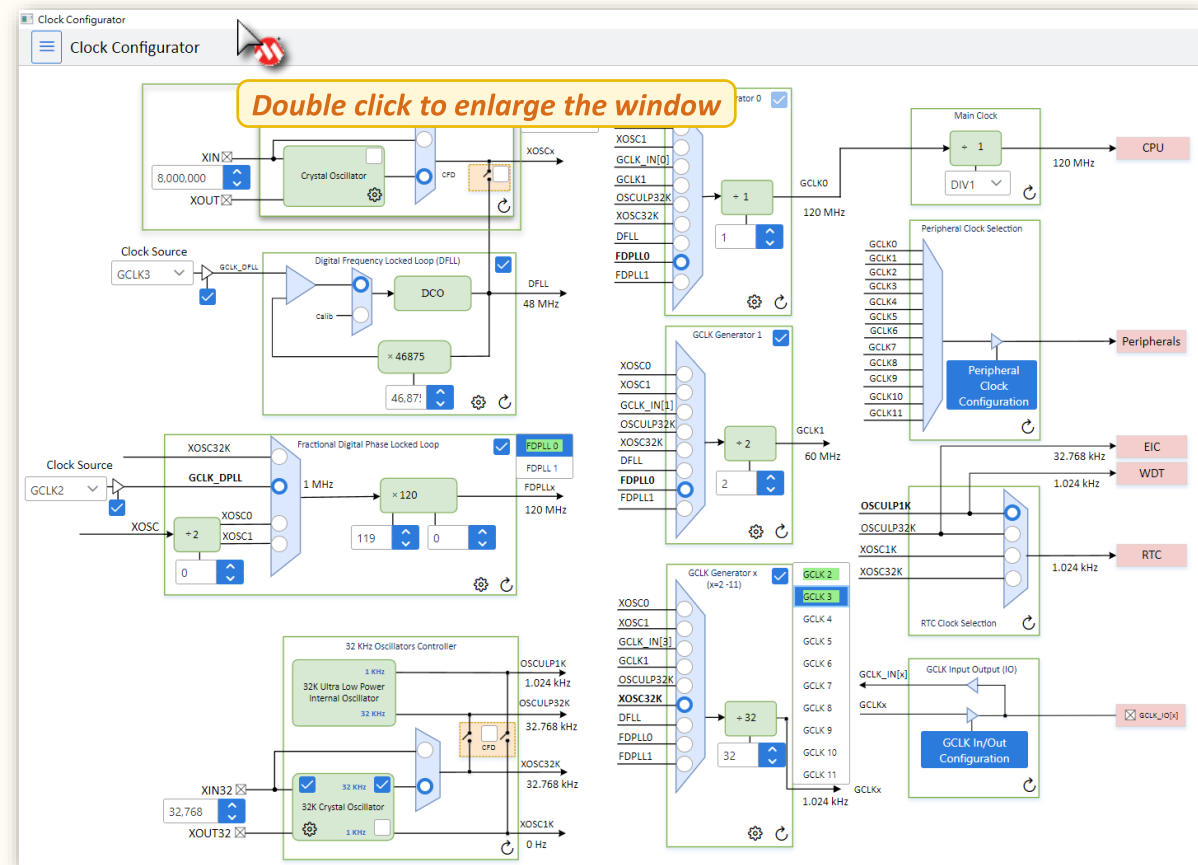
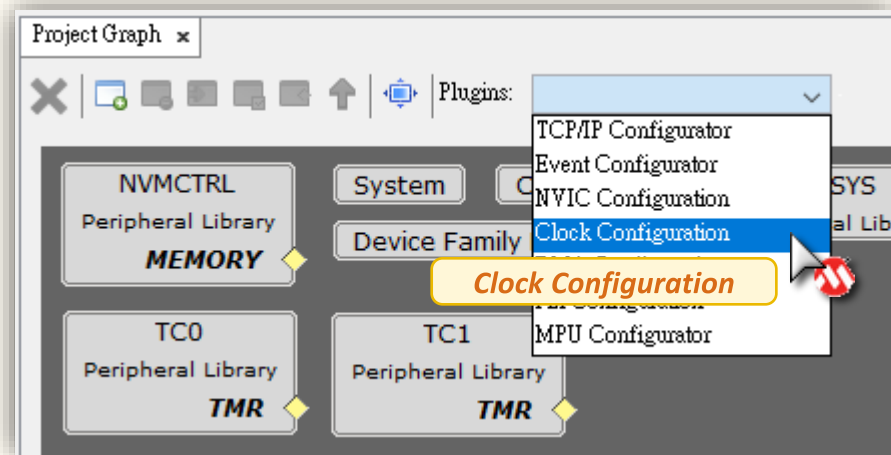
- Clock output configuration of the final system clock.



Lab4 System Clock - DFLL Close Loop and Clock Output

Step 1

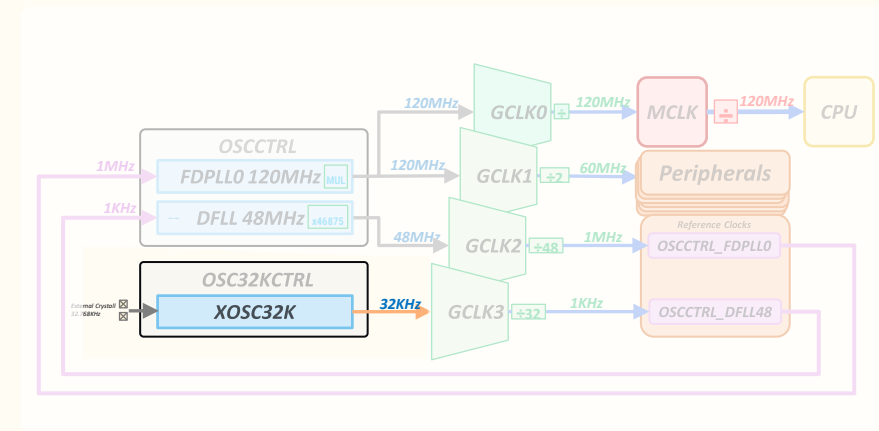
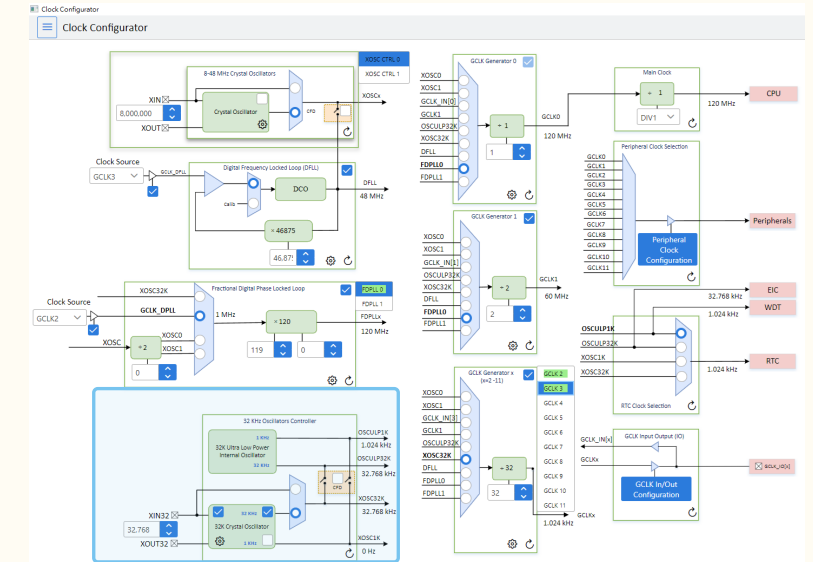
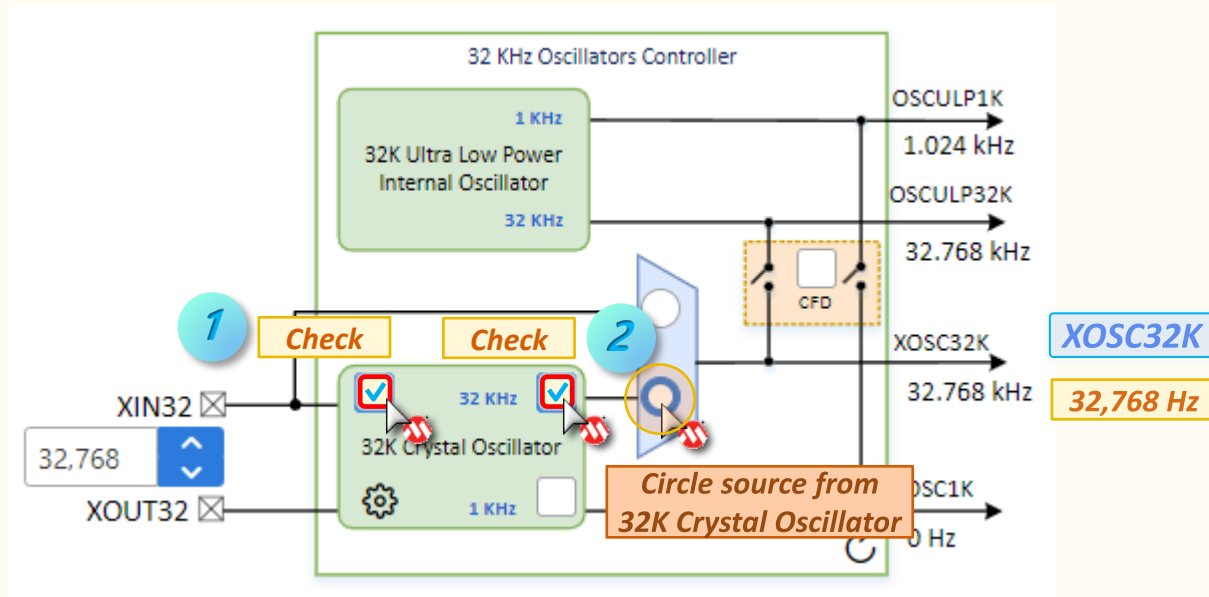
- Open Clock Easy View windows in Project Graph > Plugins > Clock Configuration.

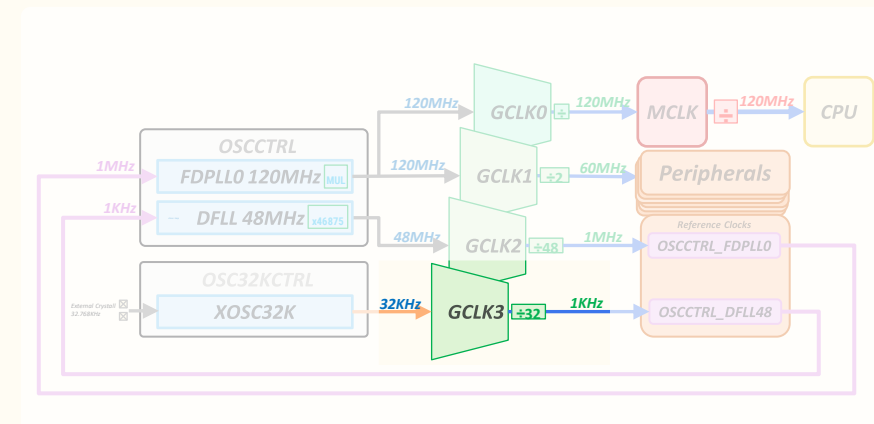
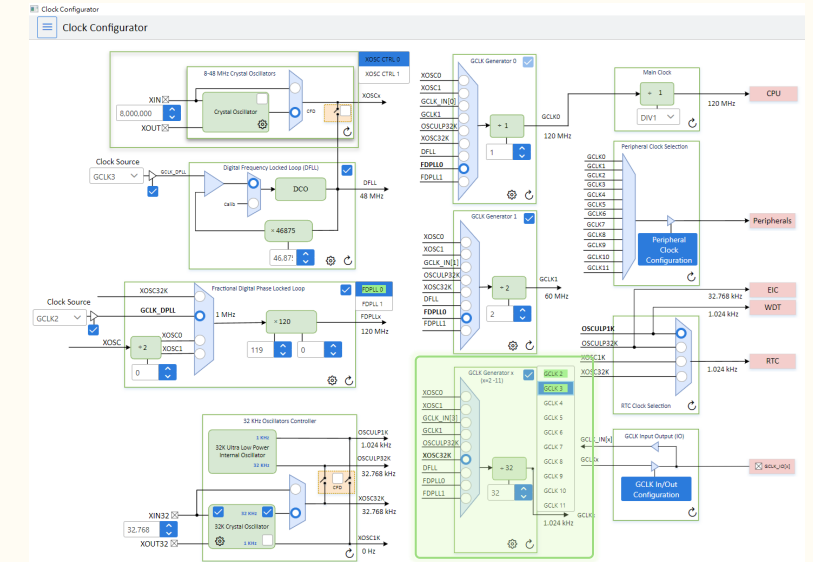


Lab4 System Clock - DFLL Close Loop and Clock Output

Step 2

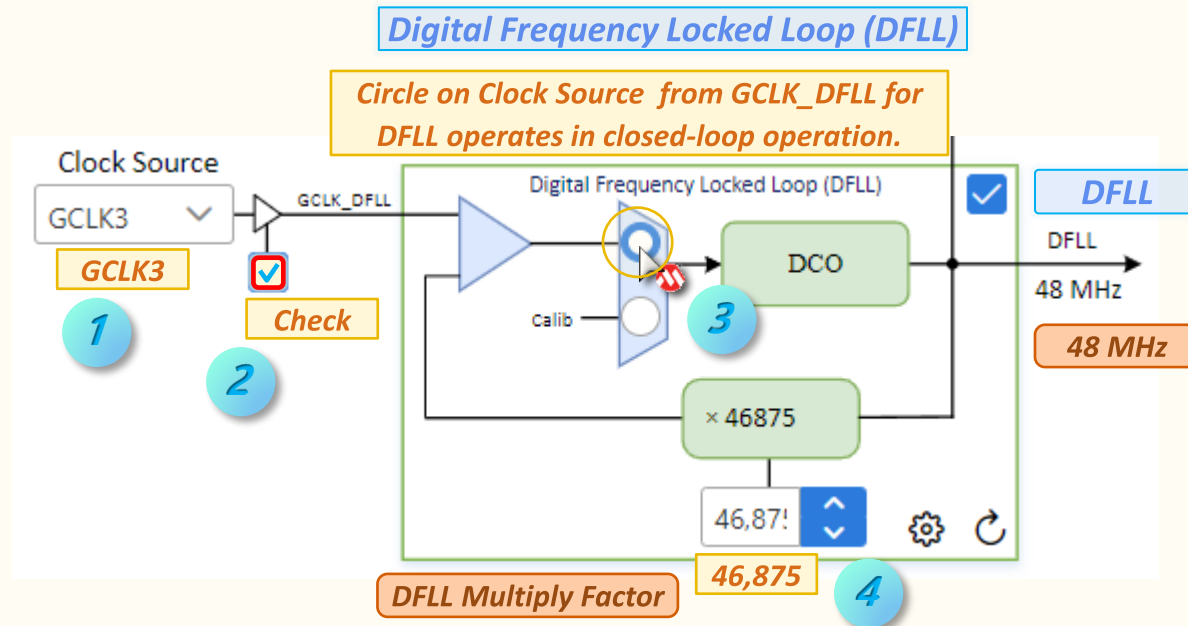
32 kHz Oscillators Controller



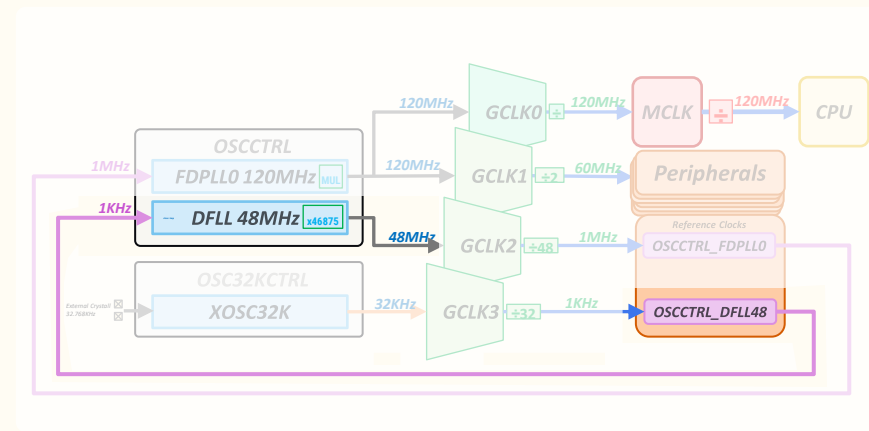
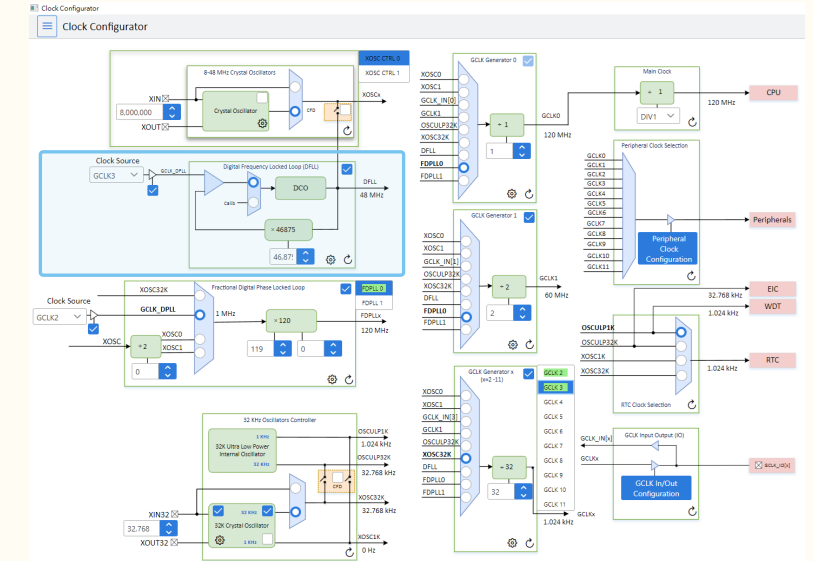


Lab4 System Clock - DFLL Close Loop and Clock Output

Step 4 (New MCC core GUI)



* To get the 48 MHz output of DFLL, the reference clock is 1024Hz (GCLK3), then calculate the DFLL Multiply Factor is $48000000/1024 = 46875$



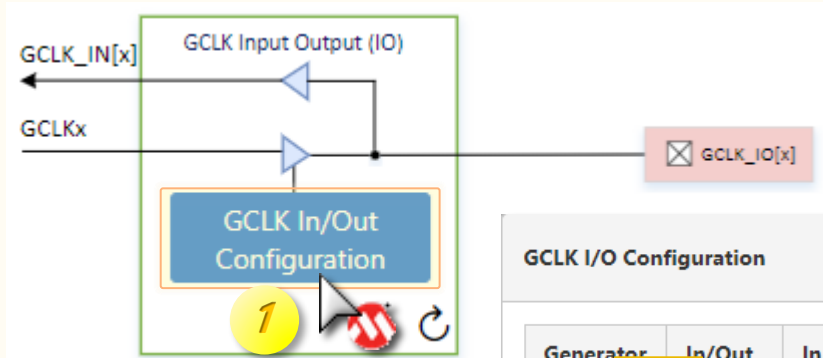
Peripheral Clock Selection



Lab4 System Clock - DFLL Close Loop and Clock Output

Step 5

GCLK_Input Output (IO)



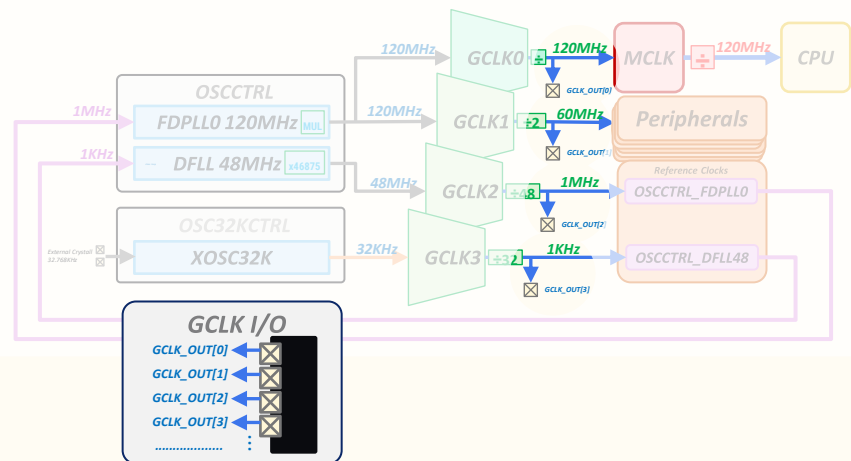
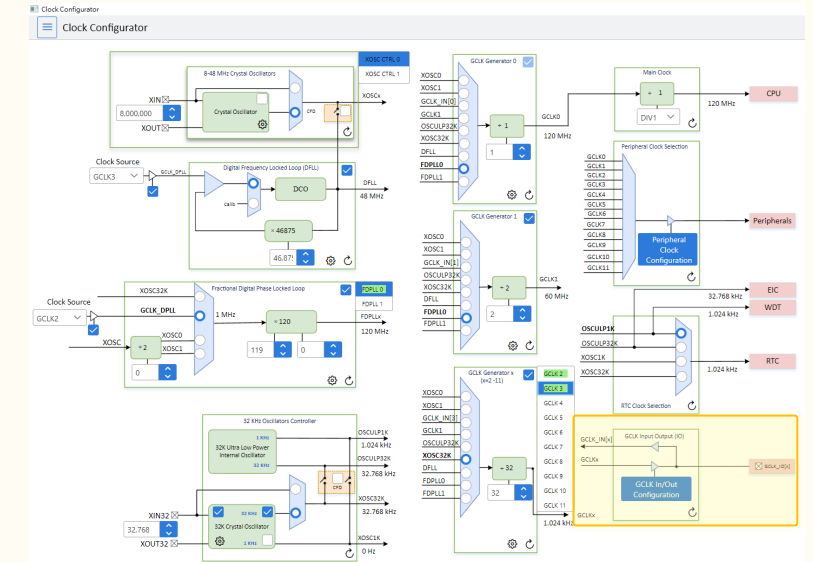
GCLK0

GCLK1

GCLK2

GCLK3

Generator	In/Out	In Frequency (Hz)	Out Frequency (Hz)
0	Out	0	120 MHz
1	Out	0	60 MHz
2	Out	0	1 MHz
3	Out	0	1.024 kHz
4	In	0	0 Hz
5	In	0	0 Hz
6	In	0	0 Hz
7	In	0	0 Hz

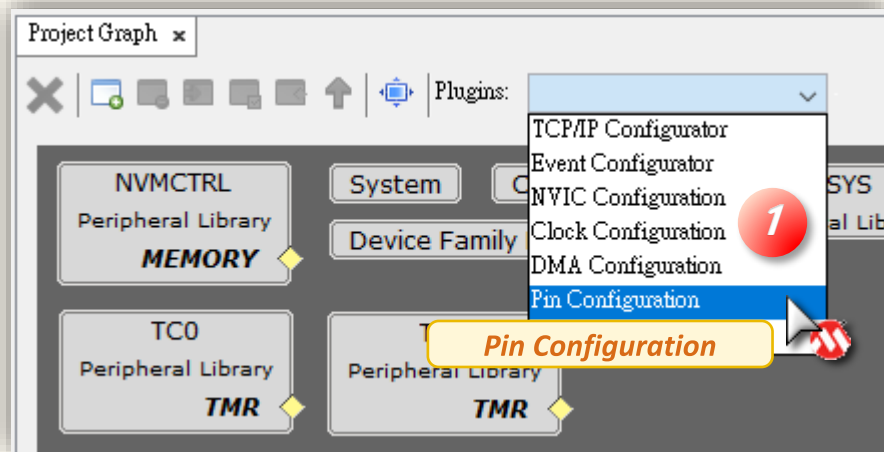


⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 6

- Open **Pin Table** by **Project Graph** > **Plugins** > **Pin Configuration**.

- Pin Setting > **Order** : Ports
- Pin Table > double click to enlarge



The screenshot shows the 'Pin Settings' window. At the top, there are tabs: 'Pin Diagram', 'Pin Table', and 'Pin Settings'. The 'Pin Settings' tab is active. Below the tabs, there is a section for 'Order by Ports' and 'Table View' with a checked 'Easy View' checkbox. A table lists pin configurations. A red circle with the number '2' points to the 'Pin Settings' tab, and a red circle with the number '3' points to the 'Order by Ports' dropdown.

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
1	PA00		Available	Digital	High Impedance	Low	☐	☐	NORMAL
2	PA01		Available	Digital	High Impedance	Low	☐	☐	NORMAL
9	PA02		Available	Digital	High Impedance	Low	☐	☐	NORMAL
10	PA03		Available	Digital	High Impedance	Low	☐	☐	NORMAL

The screenshot shows the 'Pin Table' window. At the top, there are tabs: 'Project Graph', 'Pin Diagram', 'Pin Table', and 'Pin Settings'. The 'Pin Table' tab is active. Below the tabs, there is a 'Package' dropdown set to 'TQFP128'. A grid shows pin assignments for various modules. A red circle with the number '4' points to the 'Pin Table' tab. A yellow callout box labeled 'Pin Table' points to the grid. A yellow callout box labeled 'Double click the [Pin Setting] to enlarge the window' points to a cell in the grid.

Module	Function	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
AC	AC_AIN0																						
	AC_AIN1																						
	AC_AIN2																						

Lab4 System Clock - DFLL Close Loop and Clock Output

Step 7

- **Right-click** on **GCLK** and **View > Ports** to sort by port and **Isolate** for all GCLK I/O pins.

[illegible]

Lab4 System Clock - DFLL Close Loop and Clock Output

Step 8

- Enable **GCLK I/O** as below to output the clock and **quit** the Isolate view by **Show > All**.

The diagram illustrates the configuration of GCLK I/O for clock output. On the left, a block diagram shows four GCLK modules (GCLK0, GCLK1, GCLK2, GCLK3) connected to their respective outputs (GCLK_OUT[0] to GCLK_OUT[3]) with specific frequencies and pin assignments. The main part of the image shows the Pin Diagram window with the GCLK module selected. A right-click context menu is open, showing the 'Show' option. A table at the bottom right summarizes the GCLK configurations.

GCLK I/O Configuration

Generator	In/Out	In Frequency (Hz)	Out Frequency (Hz)
0	Out	0	120 MHz
1	Out	0	60 MHz
2	Out	0	1 MHz
3	Out	0	1.024 kHz

GCLK I/O Configuration Table:

GCLK	Pin	Clock Output
GCLK_IO0	PB22	120 MHz
GCLK_IO1	PB23	60 MHz
GCLK_IO2	PA16	1 MHz
GCLK_IO3	PA17	1 kHz

⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 9

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Generate' button highlighted with a red circle and a yellow 'Generate' label. The 'Output' window shows the following log:

```
15:17:50.827 INFO: Generation complete
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```


Lab4 System Clock - DFLL Close Loop and Clock Output

Step 10

- This Lab does not require any program modification.

```
#include <stddef.h>
#include <stdbool.h>
#include <stdlib.h>
#include "definitions.h"

uint32_t i = 0;

void TC0_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if (status & TC_INTFLAG_OVF_Msk)
    {
        LED1_Toggle();
    }
}

void TC1_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
    if (status & TC_INTFLAG_OVF_Msk)
    {
        LED2_Toggle();
    }
}

int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    TC0_TimerCallbackRegister(TC0_TimerExpired, (uintptr_t) NULL);
    TC0_TimerStart();
    TC1_TimerCallbackRegister(TC1_TimerExpired, (uintptr_t) NULL);
    TC1_TimerStart();
```

```
while ( true )
{
    if ( TC1_TimerPeriodHasExpired() )
    {
        LED2_Toggle();
    }

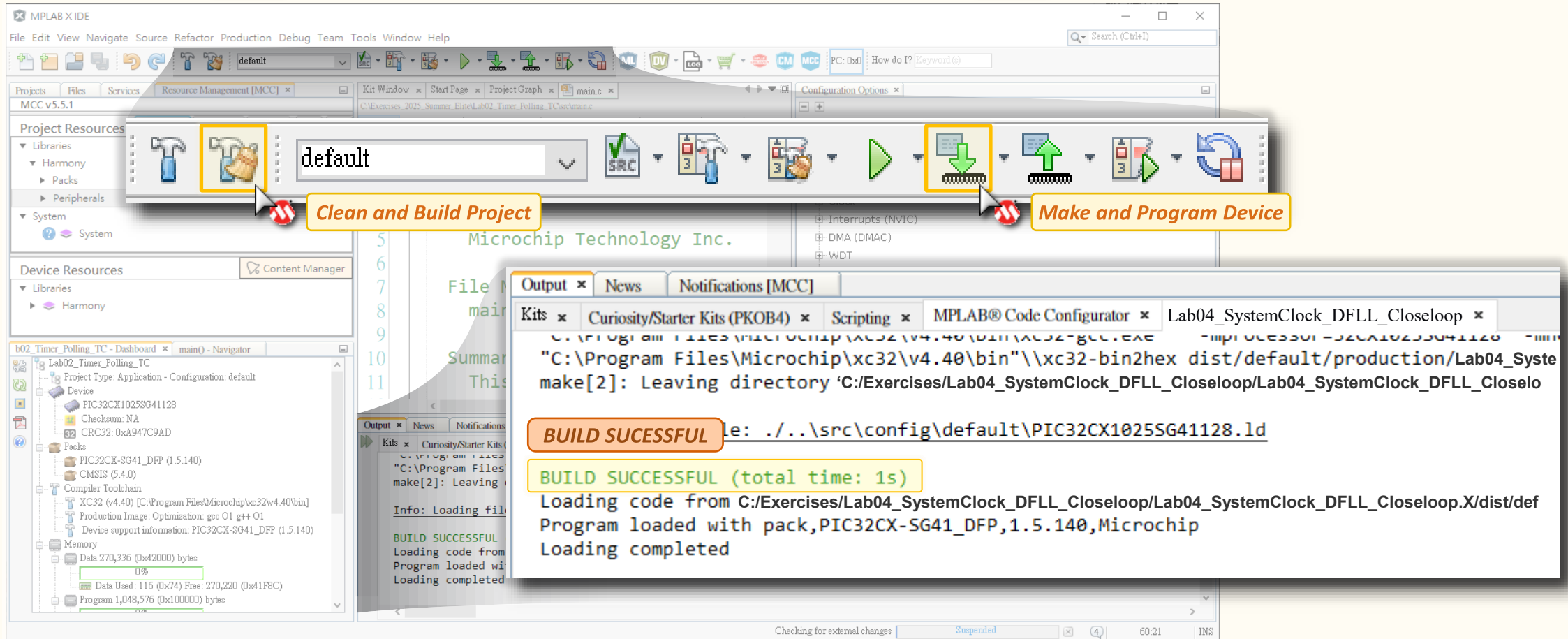
    if ( BT1_Get() ) LED3_Clear();
    else LED3_Set();
}

return ( EXIT_FAILURE );
}
```

⚙️ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 11

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains icons for various actions. Two specific icons are highlighted with yellow boxes and red arrows pointing to them:

- Clean and Build Project:** Indicated by a yellow box around the icon showing a hammer and a wrench, with a red arrow pointing to it from a yellow callout box labeled "Clean and Build Project".
- Make and Program Device:** Indicated by a yellow box around the icon showing a green arrow pointing down to a chip, with a red arrow pointing to it from a yellow callout box labeled "Make and Program Device".

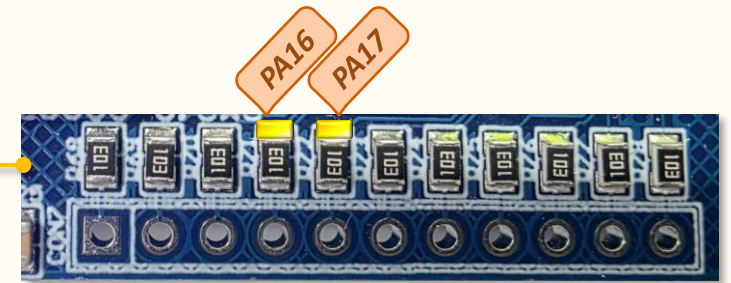
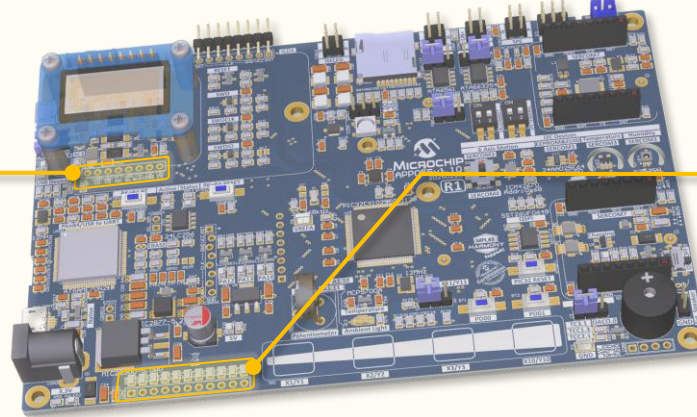
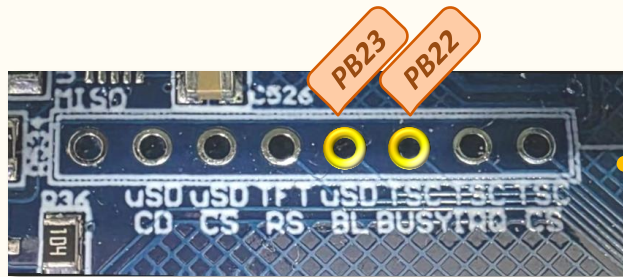
The bottom right of the IDE shows the Output window with the following text:

```
Le: ../../src/config/default/PIC32CX1025SG41128.ld
BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab04_SystemClock_DFLL_Closetoop/Lab04_SystemClock_DFLL_Closetoop.X/dist/def
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

✖ Lab4 System Clock - DFLL Close Loop and Clock Output

Step 12

- Locate the GCLK clock output pins on EVB APP055.



SOCKET3
OELD-APP050-Combine Socket

1	MOSI/SDA	MISO	16	PB01/MISO(S5.3)
2	SCK/SCL	uSD_CD	15	PC31/uSD_CD(INT14)
3	GND/A0	uSD_CS	14	PC30/uSD_nCS
4	OLED_RST	TFT_RS	13	PA27/TFT_RS
5	OLED_CS	TFT_BL	12	PB23/TFT_BL
6	3.3V	TSC_BUSY	11	PB22/TSC_BUSY(INT6)
7	5V	TSC_IRQ	10	PC26/TSC_nIRQ(INT10)
8	GND	TSC_CS	9	PC25/TSC_nCS

Monochrome OLED & Display Socket

GCLK	Pin	Clock Output
GCLK_IO0	PB22	120 MHz
GCLK_IO1	PB23	60 MHz
GCLK_IO2	PA16	1 MHz
GCLK_IO3	PA17	1 kHz

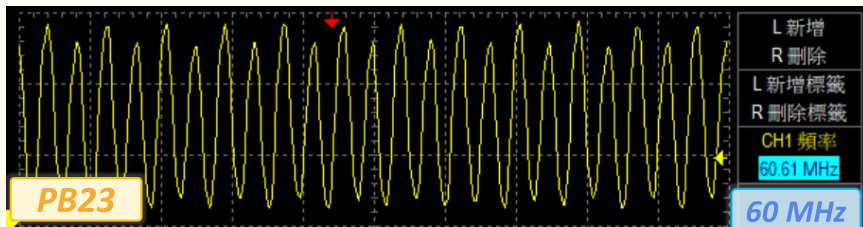
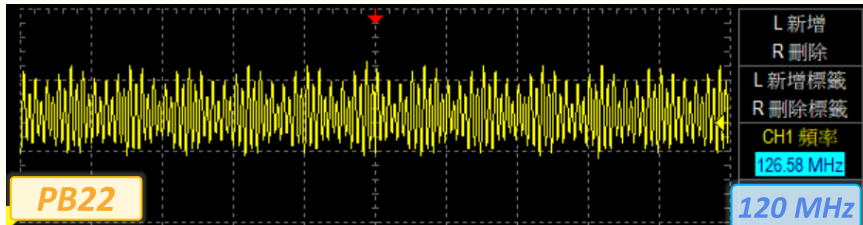
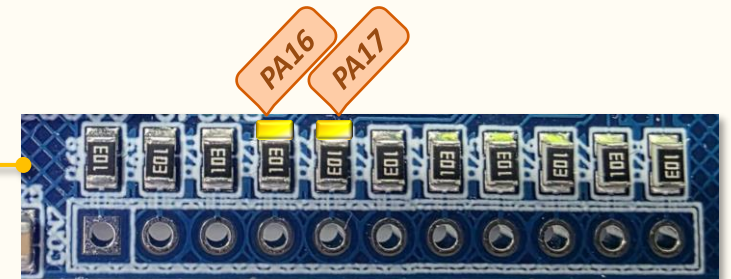
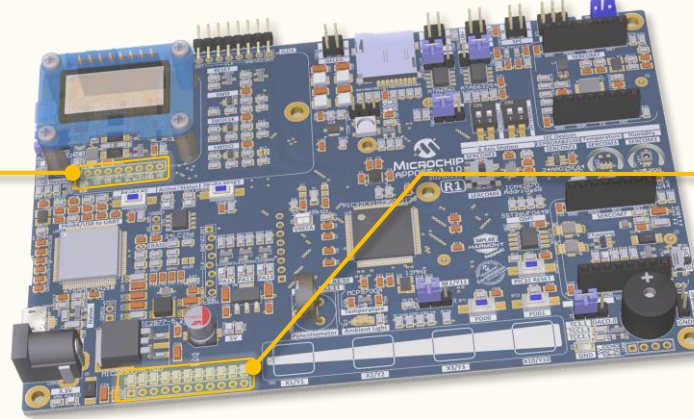
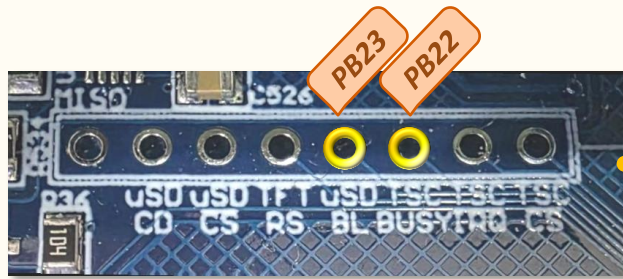
PA17/DS(TC2.1)/X11/Y11

PB08/X1/Y1	R68	10K	1
PB09/X2/Y2	R69	10K	2
PA04/X3/Y3	R70	10K	3
PA16/X10/Y10	R71	10K	4
PA17/X11/Y11	R72	10K	5
PA18/X12/Y12	R73	10K	6
PA19/X13/Y13	R74	10K	7
PB04/X22/Y22	R75	10K	8
PB05/X23/Y23	R76	10K	9
PB06/X24/Y24	R77	10K	10
PB07/X25/Y25	R78	10K	11

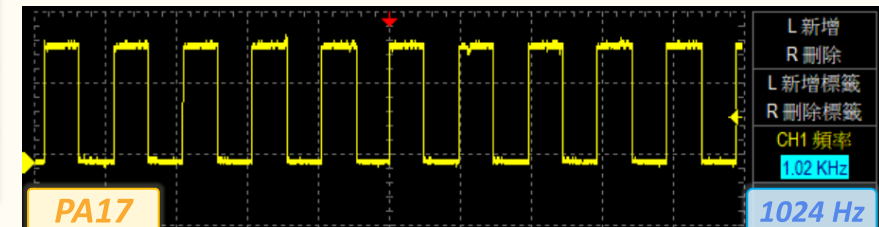
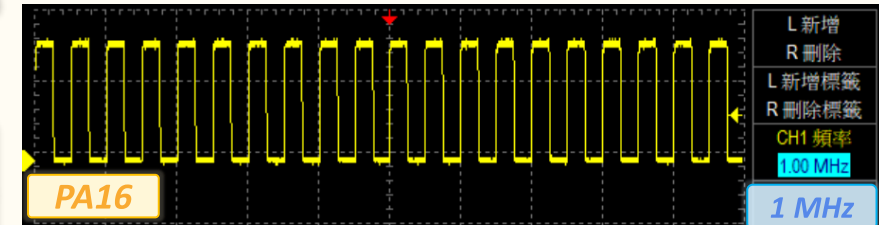
✖ Lab4 System Clock - DFLL Close Loop and Clock Output

Result

- Use an oscilloscope to observe the waveform of GCLK output and measure frequency.



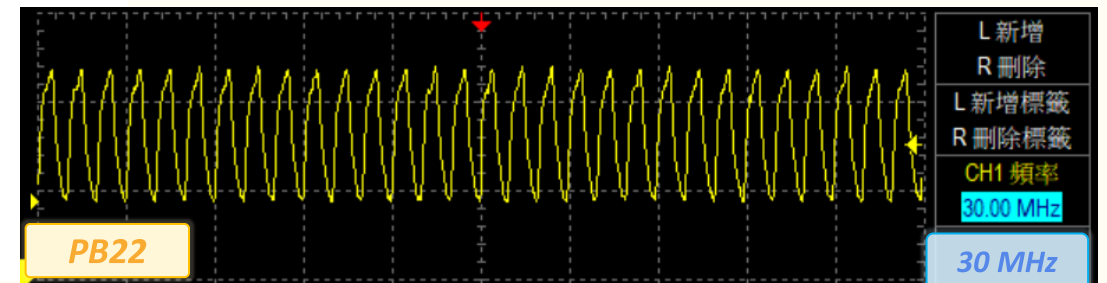
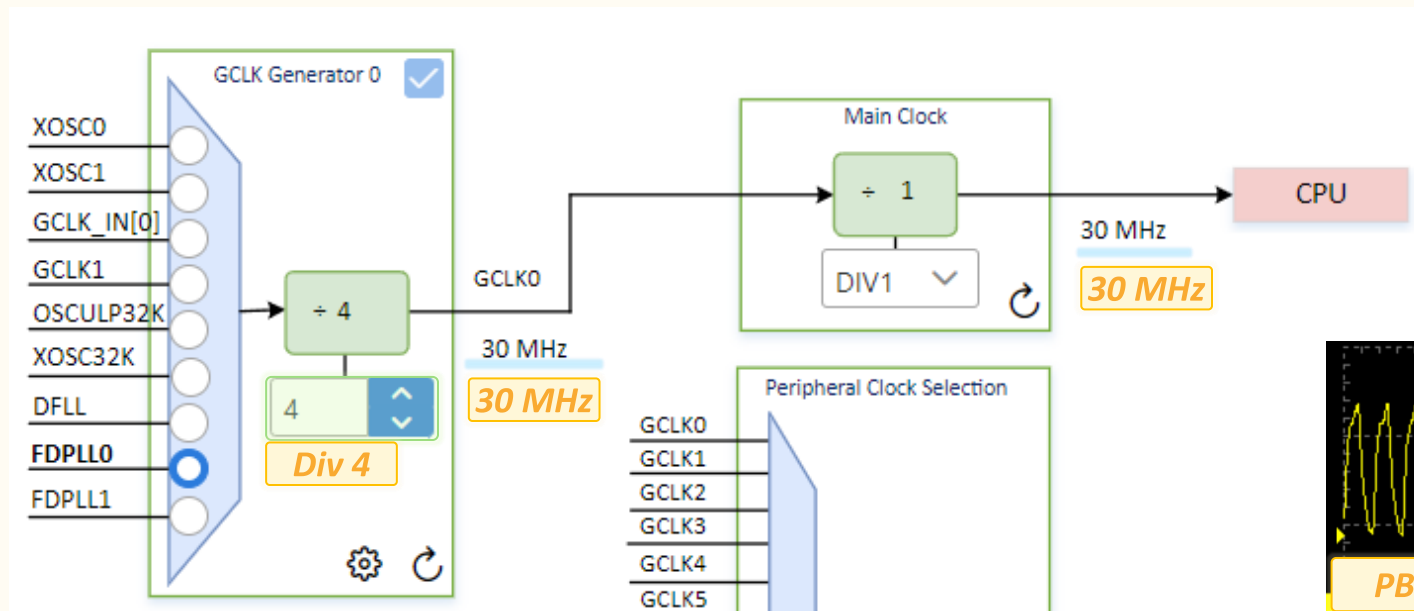
GCLK	Pin	Clock Output
GCLK_IO0	PB22	120 MHz
GCLK_IO1	PB23	60 MHz
GCLK_IO2	PA16	1 MHz
GCLK_IO3	PA17	1 kHz



Lab4 System Clock - DFLL Close Loop and Clock Output

Appendix

- Due to the circuit filtering effect, the oscilloscope cannot measure higher frequency signals. We try to divide the frequency of GCLK0 by 4 to get a lower 30MHz and observe it again.

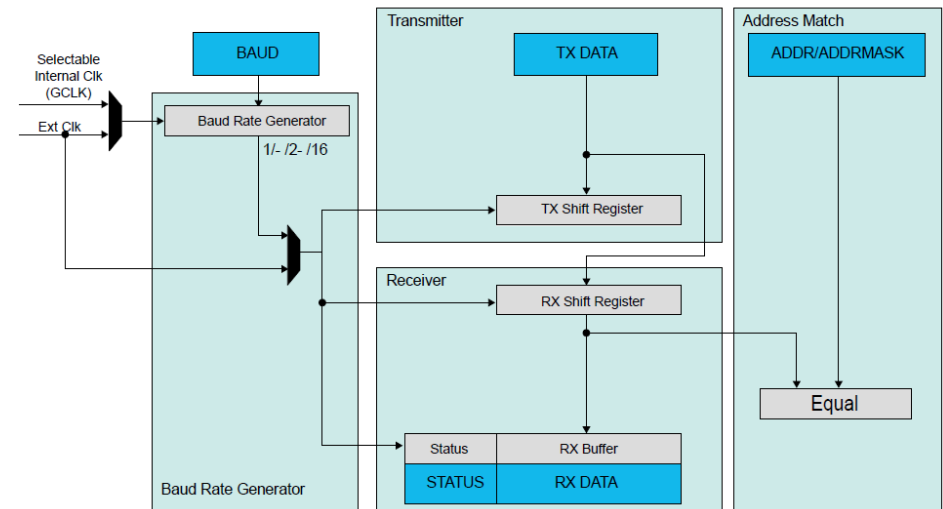
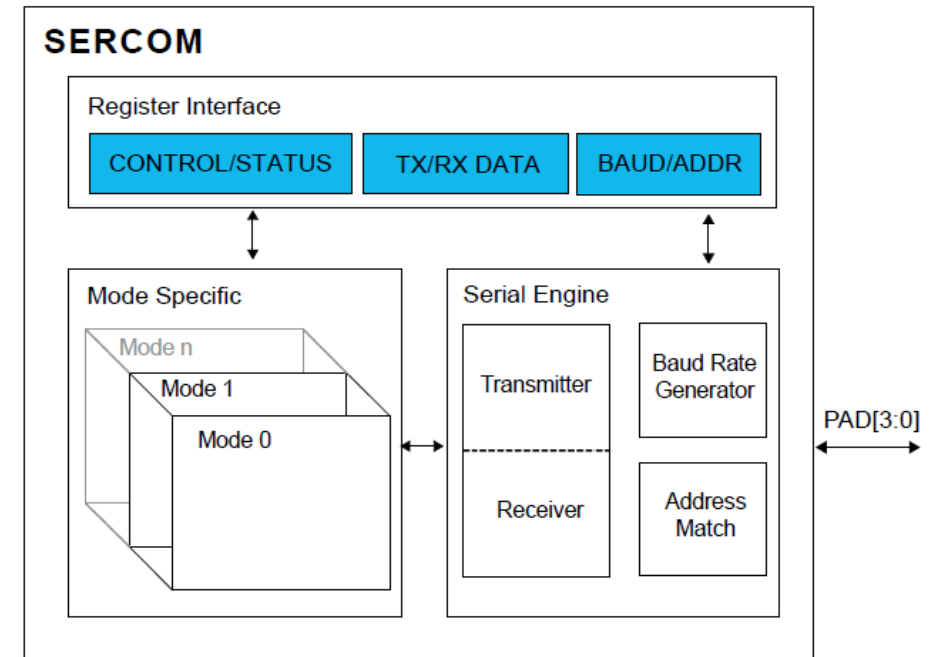


SERCOM Introduction

Serial Communication Interface

SERCOM Introduction

- The **Serial Communication Interface (SERCOM)** can be configured to support a number of modes, **I2C**, **SPI**, **USART (RS232, RS485, LIN, IrDA, ISO7816)**.
- There are up to **eight** instances of the SERCOM in PIC32CX MCU.
- 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.
- Using an external clock allows the SERCOM to be operated in all Sleep modes.
- Using the SERCOM I/O lines requires the I/O pins to be configured using port configuration (PORT).
- The SERCOM has four internal pads, **PAD[3:0]**, and the signals from I2C, SPI and USART are routed through these SERCOM pads through a multiplexer.



SERCOM Introduction

- **For USART**

- Up to 33Mb/s
- Full-duplex operation
- Sync and Async operation
- Protocol support Async :RS232, RS485, LIN, IrDA and Sync/Async : ISO7816

- **For I2C**

- Mode supported (100k/400k/1M/3.4MHz)
- Master or Slave operation
- SMBUS compatible
- Wake on address match

- **For SPI**

- Up to 20Mb/s
- Supports all four SPI modes
- Full-duplex operation
- Single data direction mode frees unused MISO or MOSI pin



SERCOM Introduction

- Pin Multiplex in MCC Harmony (Clear)

Package: TQFP128		21	22	23	24	33	34	35	36	100	101	70	71	72	73	108	109	
		A								B		C						
Module	Function	4	5	6	7	8	9	10	11	24	25	16	17	18	19	24	25	
SERCOM0	SERCOM0_PAD0																	
	SERCOM0_PAD1																	
	SERCOM0_PAD2																	
	SERCOM0_PAD3																	

Package: TQFP128		60	61	62	63	19	20	39	40	41	42	43	44	102	103	104	105
		A				B											
Module	Function	12	13	14	15	8	9	10	11	12	13	14	15	26	27	28	29
SERCOM4	SERCOM4_PAD0																
	SERCOM4_PAD1																
	SERCOM4_PAD2																
	SERCOM4_PAD3																

Package: TQFP128		1	2	66	67	68	69	119	120	98	99	76	77	111	112	80	81
		A								B				C		D	
Module	Function	0	1	16	17	18	19	30	31	22	23	22	23	27	28	20	21
SERCOM1	SERCOM1_PAD0																
	SERCOM1_PAD1																
	SERCOM1_PAD2																
	SERCOM1_PAD3																

Package: TQFP128		88	89	92	93	94	95	125	126	127	128	82	83	84	85	98	99	121	122
		A								B									
Module	Function	20	21	22	23	24	25	0	1	2	3	16	17	18	19	22	23	30	31
SERCOM5	SERCOM5_PAD0																		
	SERCOM5_PAD1																		
	SERCOM5_PAD2																		
	SERCOM5_PAD3																		

Package: TQFP128		33	34	35	36	60	61	62	63	100	101	102	103	104	105	108	109
		A								B				C			
Module	Function	8	9	10	11	12	13	14	15	24	25	26	27	28	29	24	25
SERCOM2	SERCOM2_PAD0																
	SERCOM2_PAD1																
	SERCOM2_PAD2																
	SERCOM2_PAD3																

Package: TQFP128		27	28	29	30	52	55	56	57	58	59	70	71	72	73	47	48	49	50
		C												D					
Module	Function	4	5	6	7	10	11	12	13	14	15	16	17	18	19	8	9	10	11
SERCOM6	SERCOM6_PAD0																		
	SERCOM6_PAD1																		
	SERCOM6_PAD2																		
	SERCOM6_PAD3																		

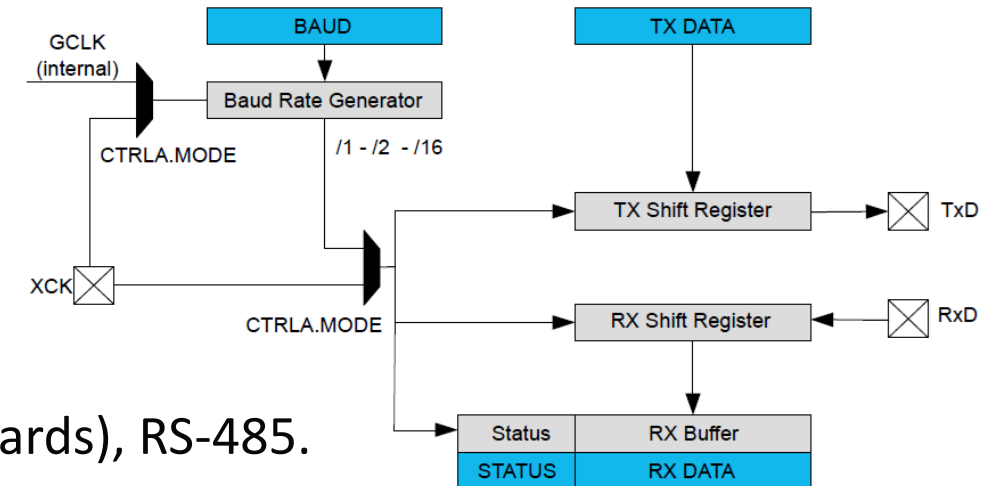
Package: TQFP128		66	67	68	69	88	89	92	93	94	95	86	87	76	77	80	81
		A								B				C		D	
Module	Function	16	17	18	19	20	21	22	23	24	25	20	21	22	23	20	21
SERCOM3	SERCOM3_PAD0																
	SERCOM3_PAD1																
	SERCOM3_PAD2																
	SERCOM3_PAD3																

Package: TQFP128		119	120	84	85	86	87	121	122	52	55	56	57	58	59	47	48	49	50
		A				B				C				D					
Module	Function	30	31	18	19	20	21	30	31	10	11	12	13	14	15	8	9	10	11
SERCOM7	SERCOM7_PAD0																		
	SERCOM7_PAD1																		
	SERCOM7_PAD2																		
	SERCOM7_PAD3																		

SERCOM-USART Architecture

SERCOM-USART

- Full-duplex communication, supports both asynchronous and synchronous modes.
- Flexible clocking: internal or external clock sources.
- Configurable data frame: 5~9 data bits, 1~2 stop bits, with optional parity.
- LSB/MSB-first transfer selection.
- Error detection: buffer overflow, frame error, parity check.
- Noise filtering with false start bit detection.
- Collision detection and sleep mode compatibility.
- Operates up to $\frac{1}{2}$ system clock or full system clock.
- Hardware flow control (RTS/CTS).
- Advanced protocol support:
 - IrDA (up to 115.2 kbps), LIN (host/client), ISO 7816 (smart cards), RS-485.
- Start-of-frame detection, two-level RX buffer, and DMA support.
- 32-bit interface for efficient system bus usage.



What's UART ?

- **Universal Asynchronous Receiver/Transmitter (UART)**

A widely used module for asynchronous serial communication between two devices. It does not require a shared clock. Instead, both devices agree on a baud rate for data transfer.

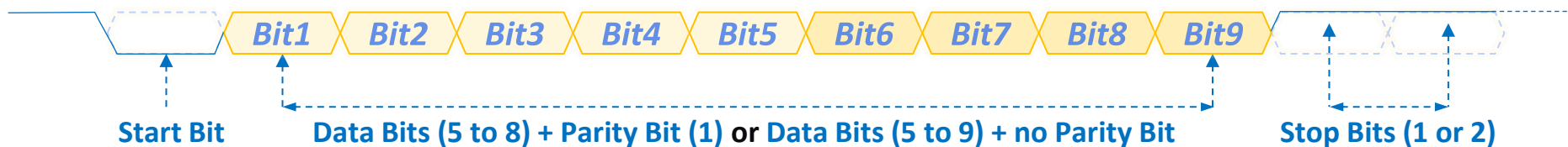
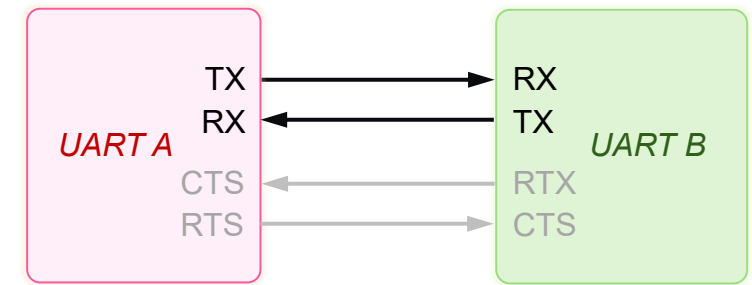
Data is transmitted through the TX (transmit) line and received through the RX (receive) line.

- **Each UART transmission typically consists of:**

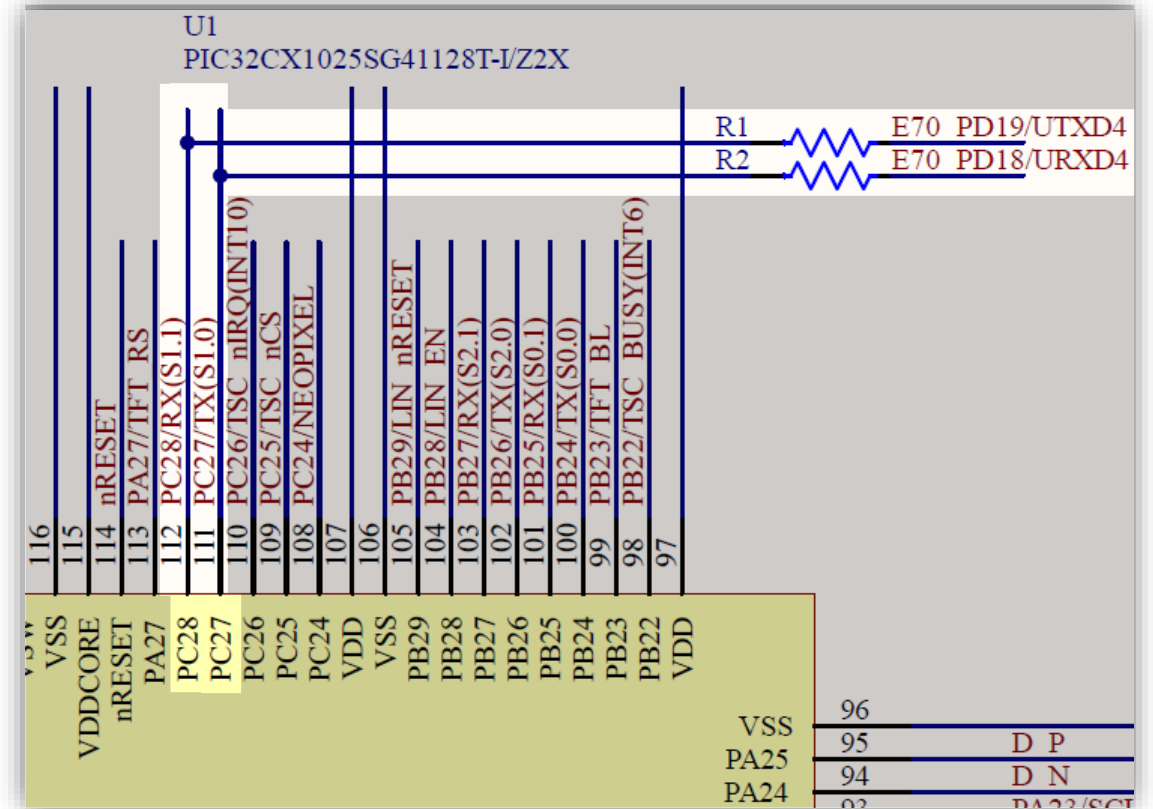
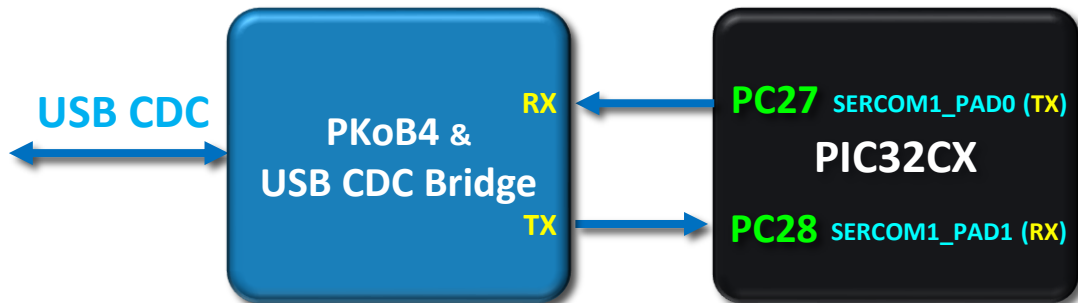
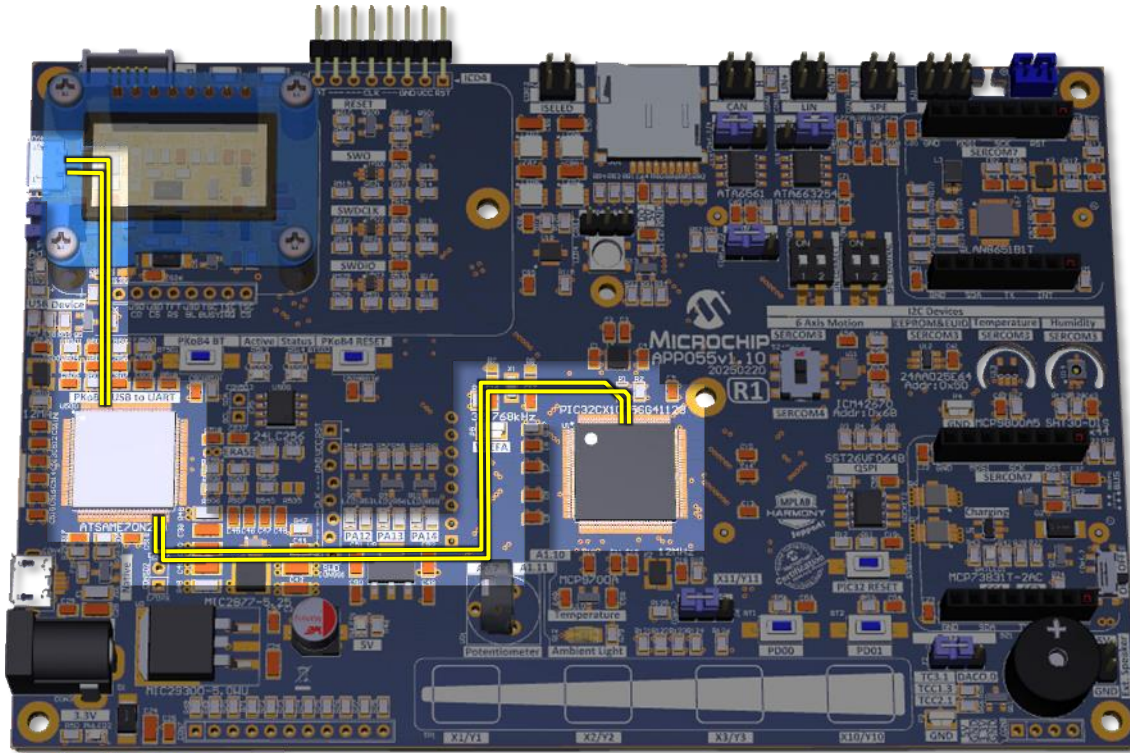
- Start Bit : 1 bit, signals the beginning of data (logic low).
- Data Bits : 5 to 9 bits, usually 8 bits (configurable).
- Parity Bit (Optional) : Used for basic error checking (optional).
- Stop Bit(s) : 1 or 2 bits (logic high), marks the end of data.

- **Key Features**

- Simple and cost-effective point-to-point communication.
- Only two wires required: TX and RX, optional flow control lines : RTS (Request To Send) and CTS (Clear To Send).
- Commonly used in microcontrollers, sensors, GPS modules, and wireless communication.
- Ideal for applications where synchronous communication is not required.



Schematic of APP055 for SERCOM-USART to USB CDC



Pin ID	Function	UART Pin
PC27	SERCOM1_PAD0	TX (Transmit)
PC28	SERCOM1_PAD1	RX (Receive)

UART Transmit by Polling

SERCOM-USART Blocking Mode

Lab5 COM Port TX by polling - SERCOM-USART

Lab5 COM Port TX by polling - SERCOM-USART

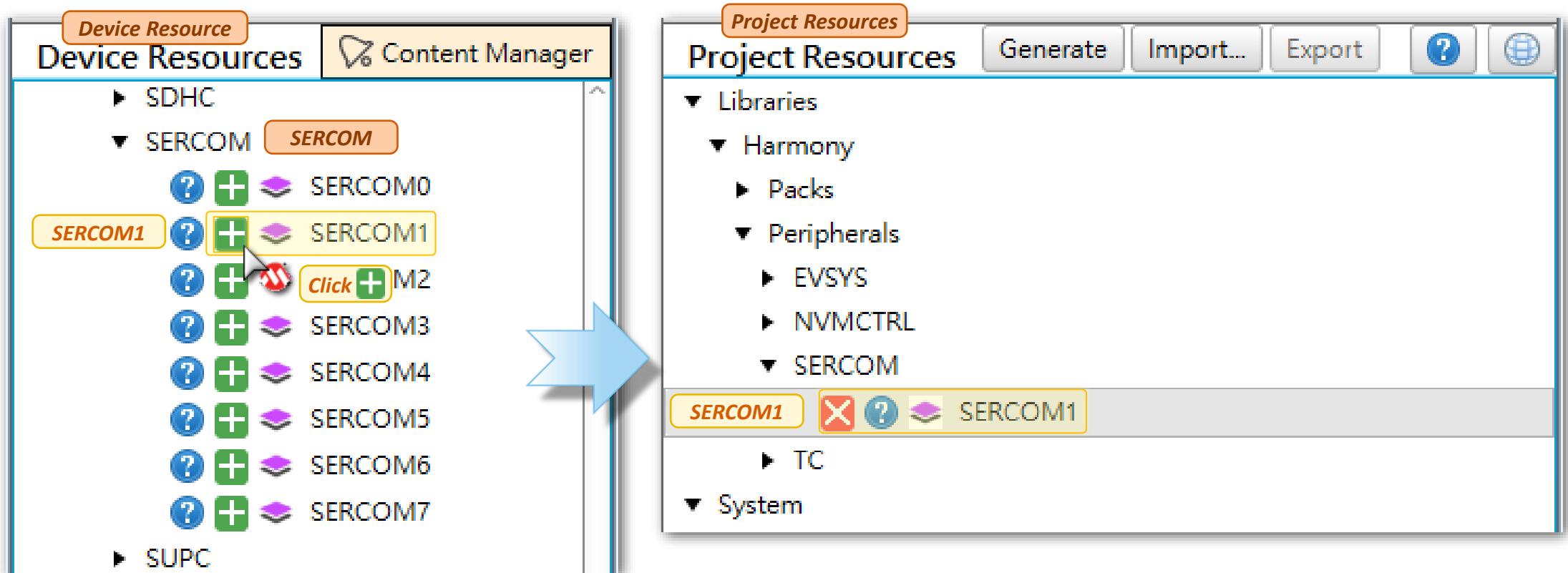
- In this Lab, we will use **TC0 Timer** to send the string "**Hello World!**" to the Tera Term terminal at 500ms intervals.
- We will use the **SERCOM-USART** interface to perform **UART COM Port** string transmission in **Polling mode**.
- The built-in UART to USB CDC bridge of the PKoB4 programmer on APP055 to send and receive data with the computer, so just connect the USB port below the OLED to the PC to get a simulated UART COM Port interface.
- We have designed the circuit to connect MCU's **PIN PC27** and **PC28** to the USB CDC bridge. These two PINs will be set as **SERCOM1_PAD[0]** and **SERCOM1_PAD[1]** in MCC Harmony as the **TX** and **RX** of UART communication on the MCU side
- The default UART transmission parameters will be used for communication:
 - Baudrate 115200 bps, Non parity check, 8 Bits data size, 1 stop bit. (**115200-n-8-1**).
- The pre-installed terminal software **Tera Term** on the computer will display the string we send from the MCU. Similarly, **Tera Term** also needs to be set to **11520-n-8-1** to connect to the UART communication settings on the MCU side.

Let's go !

⚙️ Lab5 COM Port TX by polling - SERCOM-USART

Step 1

- Find **SERCOM1** component in **Device Resources**.
- Click on **+** to add the **SERCOM1** module to your **Project Resources**.



⚙️ Lab5 COM Port TX by polling - SERCOM-USART

Step 2

- Configure the **SERCOM1-USART** to **Blocking** mode (Polling Mode).

The screenshot shows the 'Configuration Options' dialog for SERCOM1. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is set to 'Blocking mode'. The 'Receive Enable' and 'Transmit Enable' checkboxes are checked. The 'Frame Format' is set to 'USART frame'. The 'Baud Rate in Hz' is set to 115,200. The 'Use fractional baud?' checkbox is unchecked. The 'Parity Mode' is set to 'No Parity'. The 'Character Size' is set to '8 Bits'. The 'Stop Bit Mode' is set to 'One Stop Bit'. The 'Receive Pinout' is set to 'SERCOM PAD[1] is used for data reception'. The 'Transmit Pinout' is set to 'PAD[0] = TxD; PAD[1] = XCK'. The 'Enable Run in Standby' checkbox is unchecked.

Configuration Options

SERCOM1

Select SERCOM Operation Mode: USART with internal Clock

Operating Mode: Blocking mode

Receive Enable: ☒

Transmit Enable: ☒

Frame Format: USART frame

Baud Rate in Hz: 115,200

Use fractional baud?: ☐

Parity Mode: No Parity

Character Size: 8 Bits

Stop Bit Mode: One Stop Bit

Receive Pinout: SERCOM PAD[1] is used for data reception

Transmit Pinout: PAD[0] = TxD; PAD[1] = XCK

Enable Run in Standby: ☐

** We switch operating mode to "Blocking" for study of Polling Mode firstly.*

** Default Settings are Correct for use*

SERCOM PAD[1] is used for data reception

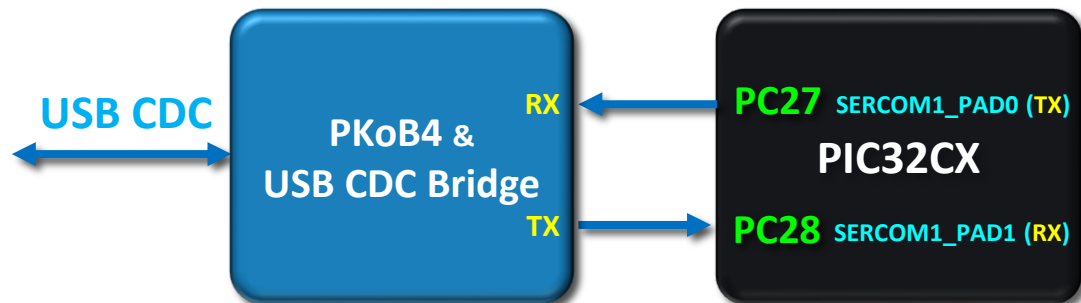
PAD[0] = TxD ; PAD[1] = XCK

⚙️ Lab5 COM Port TX by polling - SERCOM-USART

Step 3

- Pin Settings of the **SERCOM1-USART**.

Pin Settings									
Project Graph × Pin Settings × Pin Diagram × Pin Table ×									
Order: Ports Ports Table View ☒ Easy View									
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
110	PC26		Available	Digital	High Impedance	Low	☐	☐	NORMAL
111	PC27		SERCOM1_PAD0	SERCOM1_PAD0	High Impedance	n/a	☐	☐	NORMAL
112	PC28		SERCOM1_PAD1	SERCOM1_PAD1	High Impedance	n/a	☐	☐	NORMAL
123	PC30		Available	Digital	High Impedance	Low	☐	☐	NORMAL



Pin ID	Function	UART Pin
PC27	SERCOM1_PAD0	TX (Transmit)
PC28	SERCOM1_PAD1	RX (Receive)

⚙️ Lab5 COM Port TX by polling - SERCOM-USART

Step 4

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Generate' button highlighted with a red circle and a yellow 'Generate' label. The 'Project Resources' section is expanded, showing 'Libraries', 'Harmony', 'Packs', 'Peripherals', and 'System'. The 'Output' window is also open, showing the 'MPLAB® Code Configurator' tab. The output text includes:

```
15:17:50.827 INFO: Generation complete
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

SERCOM-USART Blocking Mode (Polling Mode)

Interface Functions

- | | |
|------------------------------------|--|
| • SERCOM1_USART_Initialize | : Initialize the USART module |
| • SERCOM1_USART_SerialSetup | : Configure baud rate, parity, stop bits |
| • SERCOM1_USART_Enable | : Enable USART operation |
| • SERCOM1_USART_Disable | : Disable USART operation |
| • SERCOM1_USART_TransmitterEnable | : Enable data transmitter |
| • SERCOM1_USART_TransmitterDisable | : Disable data transmitter |
| • SERCOM1_USART_Write | : Send data buffer |
| • SERCOM1_USART_TransmitComplete | : Check if transmission is complete |
| • SERCOM1_USART_TransmitterIsReady | : Check if ready to transmit |
| • SERCOM1_USART_WriteByte | : Send one byte |
| • SERCOM1_USART_ReceiverEnable | : Enable data receiver |
| • SERCOM1_USART_ReceiverDisable | : Disable data receiver |
| • SERCOM1_USART_Read | : Read received data buffer |
| • SERCOM1_USART_ReceiverIsReady | : Check if data received |
| • SERCOM1_USART_ReadByte | : Read one received byte |
| • SERCOM1_USART_ErrorGet | : Get error status |
| • SERCOM1_USART_FrequencyGet | : Get USART clock frequency |

SERCOM-USART Blocking Mode (Polling Mode)

The interface functions we will use in this Lab

void SERCOM1_USART_Write(const void *buffer, size_t size)

- Description:
 - Transmit a data buffer via SERCOM1-USART.
- Parameter:
 - buffer : Pointer to the data to transmit.
 - size : Number of bytes to write.
- Example:

```
const char msg[] = "Hello World!\r\n";  
SERCOM1_USART_Write( msg, strlen(msg) );
```

Lab5 COM Port TX by polling - SERCOM-USART

Step 5

- Open **main.c** and finish **TODO 5.01** for output the “Hello World!” to Tera Term.

```
int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    TC0_TimerCallbackRegister(TC0_TimerExpired, (uintptr_t) NULL);
    TC0_TimerStart();

    TC1_TimerCallbackRegister(TC1_TimerExpired, (uintptr_t) NULL);
    TC1_TimerStart();

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

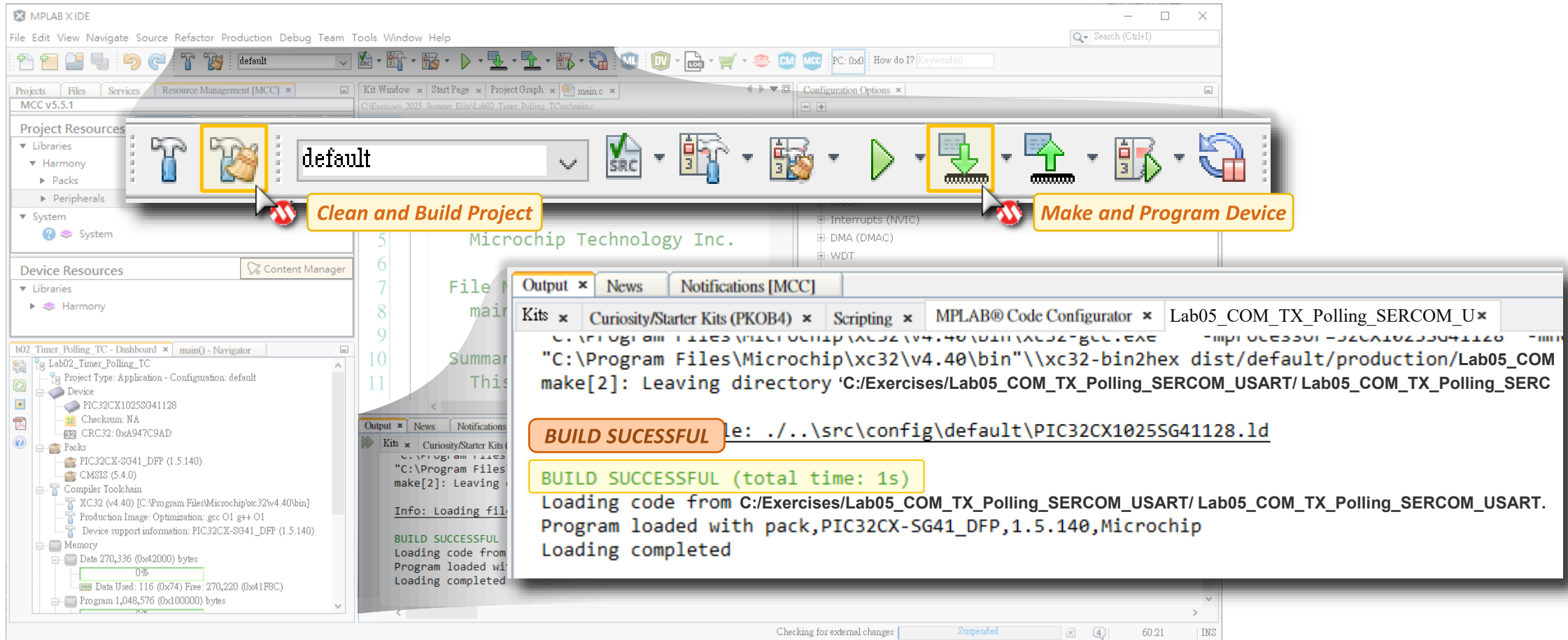
        if (TC0_HasExpired)
        {
            TC0_HasExpired = 0;
            LED1_Toggle();
// TODO 5.01
            SERCOM1_USART_Write("Hello world!\r\n", 14);
        }

        if (TC1_HasExpired)
        {
            TC1_HasExpired = 0;
            LED2_Toggle();
        }
    }
}
```

⚡ Lab5 COM Port TX by polling - SERCOM-USART

Step 6

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.

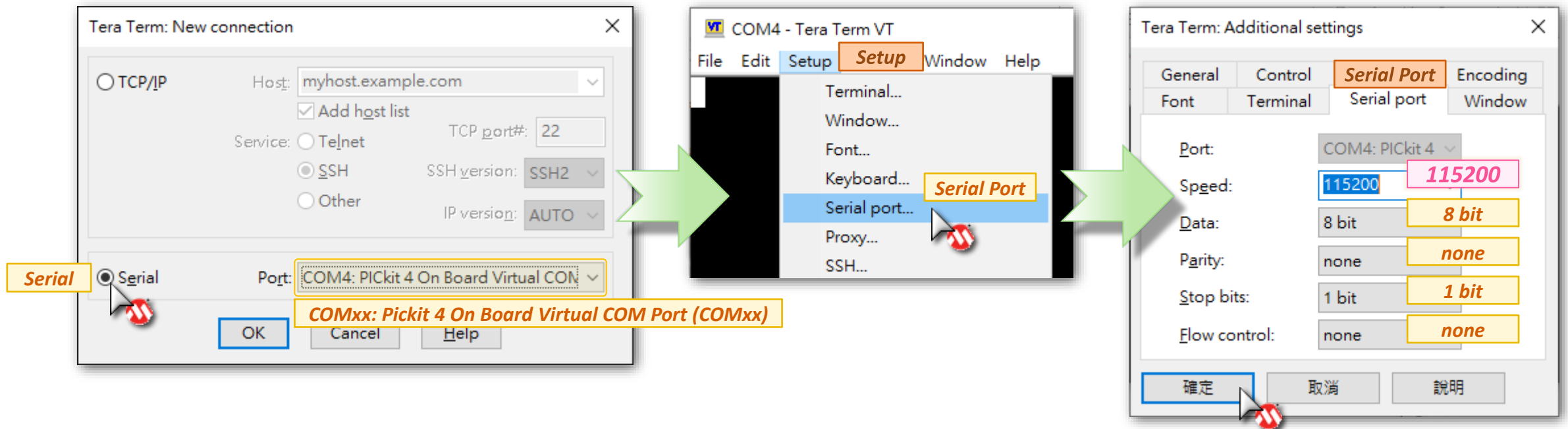


The screenshot displays the MPLAB X IDE interface. The top toolbar contains several icons, with two highlighted by yellow boxes and red arrows. The first icon, labeled "Clean and Build Project", is a hammer and wrench. The second icon, labeled "Make and Program Device", is a green arrow pointing down to a chip. Below the toolbar, the "Output" window is open, showing the build process. The output text includes:

```
Le: ../../src/config/default/PIC32CX1025SG41128.ld  
BUILD SUCCESSFUL (total time: 1s)  
Loading code from C:/Exercises/Lab05_COM_TX_Polling_SERCOM_USART/ Lab05_COM_TX_Polling_SERCOM_USART.  
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip  
Loading completed
```

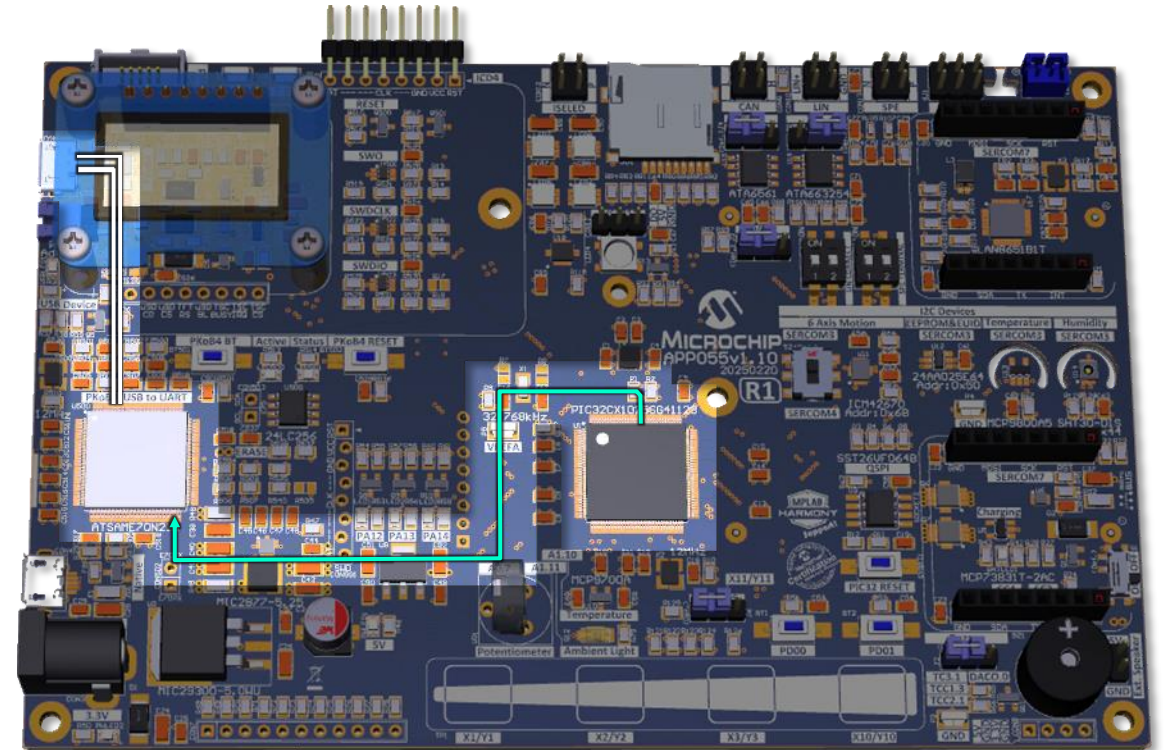
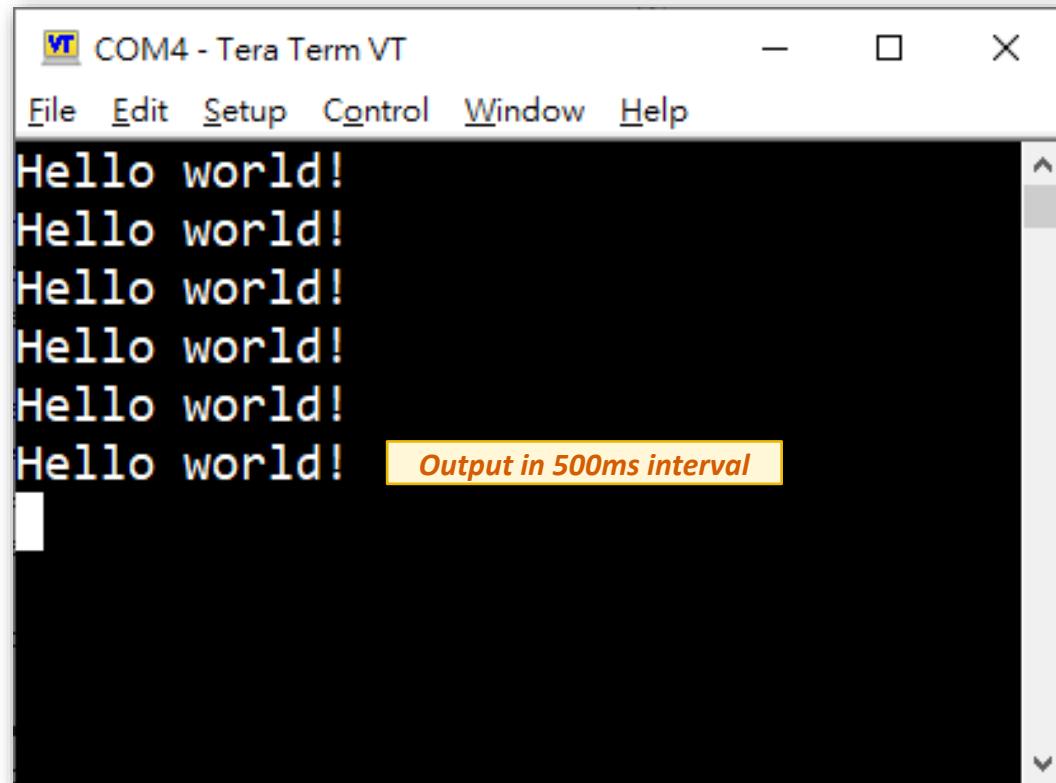

Tera Term : The Terminal Console Software

- Tera Term is a popular and free to use terminal console software, you could download it from <https://teratermproject.github.io/index-en.html>
- Execute the Tera Term by double Click on the Icon  on your PC's desktop.
- Choice a New Connection “Serial” and select correct COM Port of your EVB,
 - It might show as **COMxx: Pickit 4 On Board Vritual COM Port (COMxx)**
- Go to the top menu and enter “Setup” tab.
- Config the “Serial Port” to **115200-n-8-1**.



🔧 Lab5 COM Port TX by polling - SERCOM-USART

Result



UART Receive by Interrupt

SERCOM-USART Non-Blocking Mode

Lab6 COM Port RX by interrupt - SERCOM-USART

⚙️ Lab6 COM Port RX by interrupt - SERCOM-USART

- In the previous LAB, we used **Polling Mode(Blocking)** to implement SERCOM-USART transmission. Since we can decide when to start the transmission, once the transmission starts, we can wait in place (Blocking) for the transmission to complete before continuing with the following actions.
- However, when receiving data, since we don't know when the data will be received, waiting in place (Blocking) or constantly querying (Polling) will cause program flow blockage and waste of resources. Therefore, using **Interrupts(Non-Blocking)** to handle UART Receive will improve these shortcomings and is also the normal way to use UAR Receive.
- In this LAB, we will change the operation mode of **SERCOM-USART** to **Non-Blocking Mode (Interrupt)** to receive a character **c** from Tera Term and reconstruct the received characters into the string **"Received Data : c \r\n"** and then respond to Tera Term.
- Similar to the previous interrupt usage, we will create two custom functions, register them as the **callback** functions when the **UART transmit and receive** interrupts occur, and set two software flags in these two callback functions to serve as the basis for the main loop of the program to determine whether an interrupt has occurred.

Let's go !

⚙️ Lab6 COM Port RX by interrupt - SERCOM-USART

Step 1

- Configure the **SERCOM1-USART** to **Non-blocking** mode (Interrupt Mode).

The screenshot shows the 'Configuration Options' dialog for the SERCOM1 peripheral. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is set to 'Non-blocking mode', which is highlighted with a pink box and labeled '* The default operating mode.' The 'Receive Enable' and 'Transmit Enable' checkboxes are both checked. The 'Frame Format' is set to 'USART frame'. The 'Baud Rate in Hz' is set to 115,200. The 'Use fractional baud?' checkbox is unchecked. The 'Parity Mode' is set to 'No Parity'. The 'Character Size' is set to '8 Bits'. The 'Stop Bit Mode' is set to 'One Stop Bit'. The 'Start-of-Frame Detection Enable' checkbox is unchecked. The 'Receive Pinout' is set to 'SERCOM PAD[1] is used for data reception'. The 'Transmit Pinout' is set to 'PAD[0] = TxD; PAD[1] = XCK'. The 'Enable Run in Standby' checkbox is unchecked.

Option	Value
Select SERCOM Operation Mode	USART with internal Clock
Operating Mode	Non-blocking mode
Receive Enable	☒
Transmit Enable	☒
Frame Format	USART frame
Baud Rate in Hz	115,200
Use fractional baud?	☐
Parity Mode	No Parity
Character Size	8 Bits
Stop Bit Mode	One Stop Bit
Start-of-Frame Detection Enable	☐
Receive Pinout	SERCOM PAD[1] is used for data reception
Transmit Pinout	PAD[0] = TxD; PAD[1] = XCK
Enable Run in Standby	☐

⚙️ Lab6 COM Port RX by interrupt - SERCOM-USART

Step 2

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC] v5.5.1' window is open, showing the 'Project Resources' tab. The 'Generate' button is highlighted with a yellow box and a red arrow. Below it, the 'System' resource is also highlighted. The 'MPLAB® Code Configurator' window is open, showing the 'Output' tab. The output log displays the following information:

```
15:17:50.827 INFO: Generation complete.
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

SERCOM-USART Non-Blocking Mode (Interrupt Mode)

Interface Functions

• SERCOM1_USART_Initialize	: Initialize the USART module
• SERCOM1_USART_SerialSetup	: Configure baud rate, parity, stop bits
• SERCOM1_USART_Enable	: Enable USART operation
• SERCOM1_USART_Disable	: Disable USART operation
• SERCOM1_USART_TransmitterEnable	: Enable data transmitter
• SERCOM1_USART_TransmitterDisable	: Disable data transmitter
• SERCOM1_USART_Write	: Send data buffer
• SERCOM1_USART_TransmitComplete	: Check if transmission is complete
• SERCOM1_USART_TransmitterIsReady	: Check if ready to transmit
• SERCOM1_USART_WriteByte	: Send one byte
• SERCOM1_USART_WriteIsBusy	: Check if write is ongoing
• SERCOM1_USART_WriteCountGet	: Get number of bytes written
• SERCOM1_USART_WriteCallbackRegister	: Set callback for write complete
• SERCOM1_USART_ReceiverEnable	: Enable data receiver
• SERCOM1_USART_ReceiverDisable	: Disable data receiver
• SERCOM1_USART_Read	: Read received data buffer
• SERCOM1_USART_ReceiverIsReady	: Check if data received
• SERCOM1_USART_ReadByte	: Read one received byte
• SERCOM1_USART_ReadIsBusy	: Check if read is ongoing
• SERCOM1_USART_ReadCountGet	: Get number of bytes read
• SERCOM1_USART_ReadAbort	: Abort current read operation
• SERCOM1_USART_ReadCallbackRegister	: Set callback for read complete
• SERCOM1_USART_ErrorGet	: Get error status
• SERCOM1_USART_FrequencyGet	: Get USART clock frequency

SERCOM-USART Non-Blocking Mode (Interrupt Mode)

The interface functions we will use in this Lab

```
typedef void (*SERCOM_USART_CALLBACK)( uintptr_t context );
```

```
void SERCOM1_USART_ReadCallbackRegister( SERCOM_USART_CALLBACK callback, uintptr_t context )
```

- Description:
 - Register a callback function for receive complete event.
- Parameter:
 - callback : Function to call when read is complete.
 - context : User-defined data passed to callback.

- Example:

```
void MyReadHandler(uintptr_t context)
```

```
{
```

```
    // Handle received data here
```

```
}
```

```
SERCOM1_USART_ReadCallbackRegister( MyReadHandler, (uintptr_t)NULL );
```

SERCOM-USART Non-Blocking Mode (Interrupt Mode)

The interface functions we will use in this Lab

```
typedef void (*SERCOM_USART_CALLBACK)( uintptr_t context );
```

```
void SERCOM1_USART_WriteCallbackRegister( SERCOM_USART_CALLBACK callback, uintptr_t context )
```

- Description:
 - Register a callback function for transmit complete event.
- Parameter:
 - callback : Function to call when write is complete.
 - context : User-defined data passed to callback.

- Example:

```
void MyWriteHandler(uintptr_t context)
```

```
{
```

```
    // Handle transmit data here
```

```
}
```

```
SERCOM1_USART_WriteCallbackRegister( MyWriteHandler, (uintptr_t)NULL );
```

SERCOM-USART Non-Blocking Mode (Interrupt Mode)

The interface functions we will use in this Lab

void SERCOM1_USART_Read(void *buffer, const size_t size)

- Description:
 - Receive a data buffer via SERCOM1-USART.
- Parameter:
 - buffer : Pointer to the data to receive.
 - size : Number of bytes to read.
- Example:

```
const char msg[14];
SERCOM1_USART_Read( msg, 14 );
```

void SERCOM1_USART_Write(void *buffer, const size_t size)

- Description:
 - Transmit a data buffer via SERCOM1-USART.
- Parameter:
 - buffer : Pointer to the data to transmit.
 - size : Number of bytes to write.
- Example:

```
const char msg[] = "Hello World!\r\n";
SERCOM1_USART_Write( msg, strlen(msg) );
```

⚙️ Lab6 COM Port RX by interrupt - SERCOM-USART

Step 3

- Open **main.c** to include the C string header, declare the buffer, software flags, and two callback functions for TX/RX.

```
#include <stddef.h>           // Defines NULL
#include <stdbool.h>          // Defines true
#include <stdlib.h>           // Defines EXIT_FAILURE
// TODO 6.01
#include <string.h>
#include <stdio.h>
#include "definitions.h"      // SYS function prototypes
```

```
uint32_t i = 0;
volatile uint8_t TC0_HasExpired = 0;
volatile uint8_t TC1_HasExpired = 0;
// TODO 6.02
uint8_t USART1_TransmitData[100];
uint8_t USART1_ReceiveData[1];
volatile uint8_t USART1_IsReceived = 0;
volatile uint8_t USART1_IsTransmitted = 1;
```

```
void USART1_Transmit(uintptr_t context)
{
    USART1_IsTransmitted = 1;
}
```

The Tx callback only raises the flag, but we don't do anything here

```
void USART1_Receive(uintptr_t context)
{
    USART1_IsReceived = 1;
}
```

** We will do the processing in the main loop to exit the interrupt as soon as possible*

Raise the Rx software flag in the callback

```
void TC0_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
{
```

⚙️ Lab6 COM Port RX by interrupt - SERCOM-USART

Step 4

- Register two callback functions, start UART receive on byte, once the Rx flagged in callback then echo received char.

```
int main ( void )
{
    ...
    TC1_TimerStart();

    // TODO 6.03
    SERCOM1_USART_ReadCallbackRegister(USART1_Receive, (uintptr_t) NULL);
    SERCOM1_USART_WriteCallbackRegister(USART1_Transmit, (uintptr_t) NULL);
    SERCOM1_USART_Read(USART1_ReceiveData, 1);

    while ( true )
    {
        if (TC1_HasExpired)
        {
            TC1_HasExpired = 0;
            LED2_Toggle();
        }

        // TODO 6.04
        if (USART1_IsReceived)
        {
            USART1_IsReceived = 0;

            sprintf( (char*)USART1_TransmitData, "Received Data : %1c\r\n", USART1_ReceiveData[0] );
            SERCOM1_USART_Write( USART1_TransmitData, strlen((char*)USART1_TransmitData) );
            SERCOM1_USART_Read( USART1_ReceiveData, 1 );
        }

        if ( BT1_Get() ) LED3_Clear();
    }
}
```

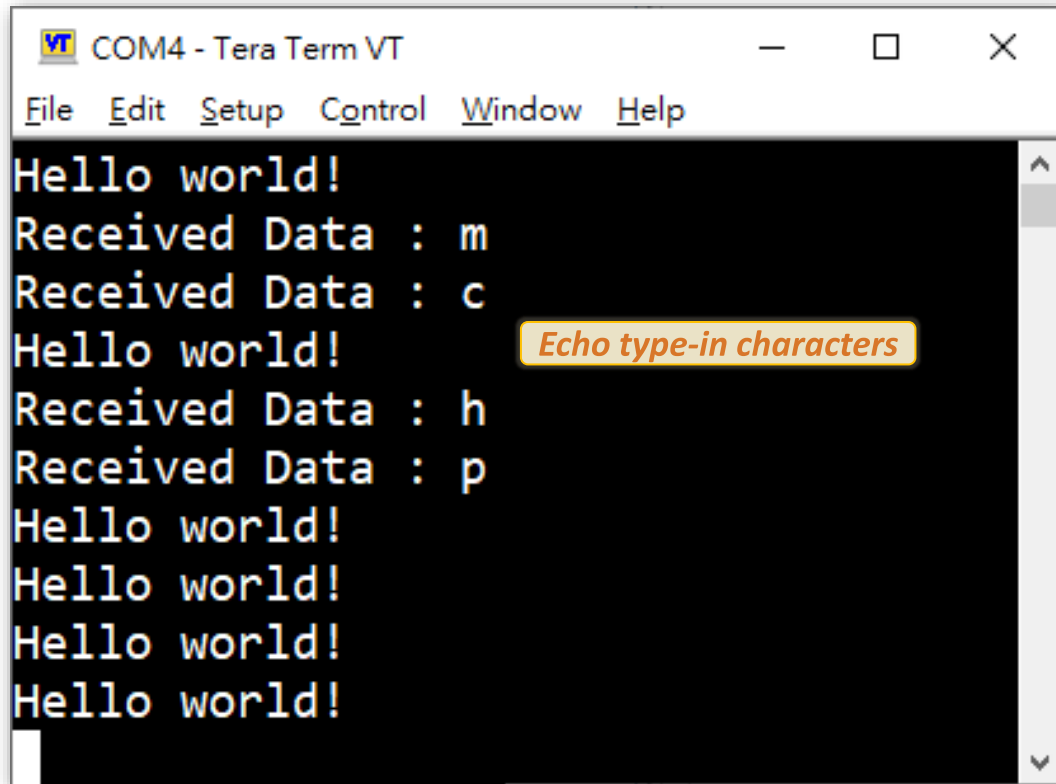
Note! You must first execute the UART read function, Otherwise, no characters will be read and no interrupt will occur.

Remember to clear the Rx software flag

If you need to read characters again after the interrupt is completed, you must execute the UART read function again.

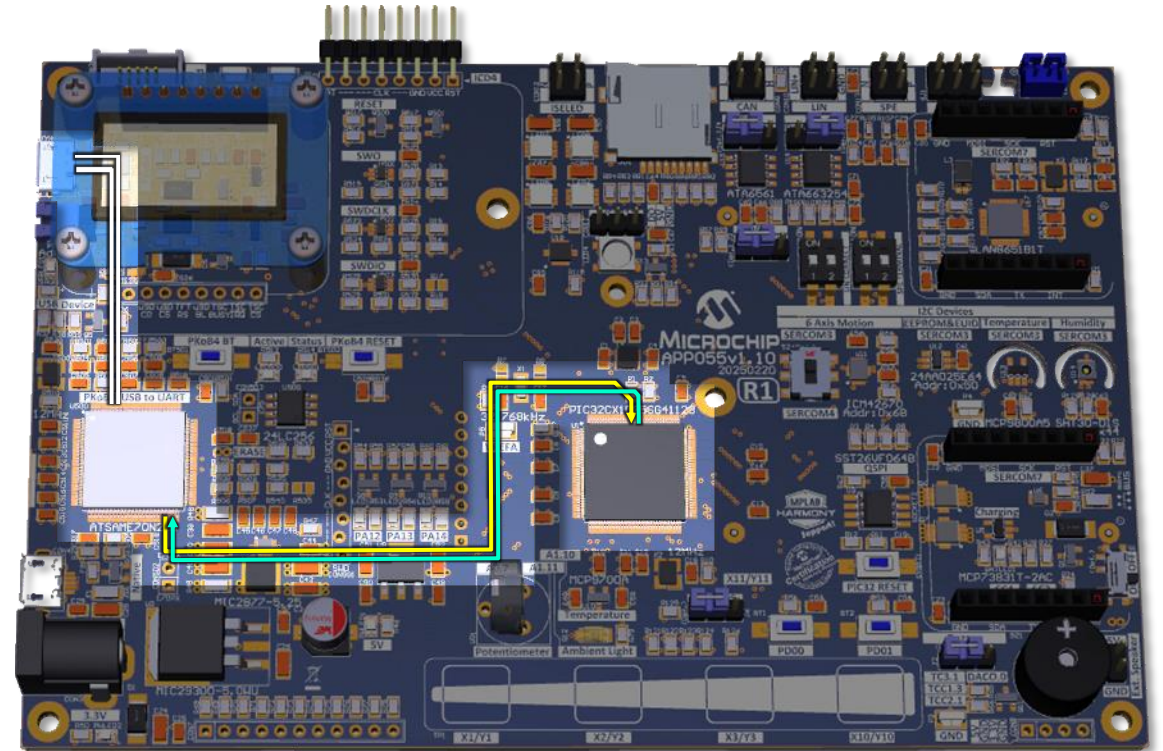
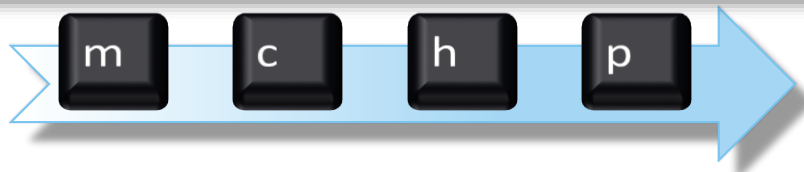
⚙️ Lab6 COM Port RX by interrupt - SERCOM-USART

Result



```
COM4 - Tera Term VT
File Edit Setup Control Window Help

Hello world!
Received Data : m
Received Data : c
Hello world!
Received Data : h
Received Data : p
Hello world!
Hello world!
Hello world!
Hello world!
```



myprintf

Implementing a custom printf function using SERCOM-USART

Lab7 myprintf - SERCOM-USART

⚙️ Lab7 myprintf - SERCOM-USART

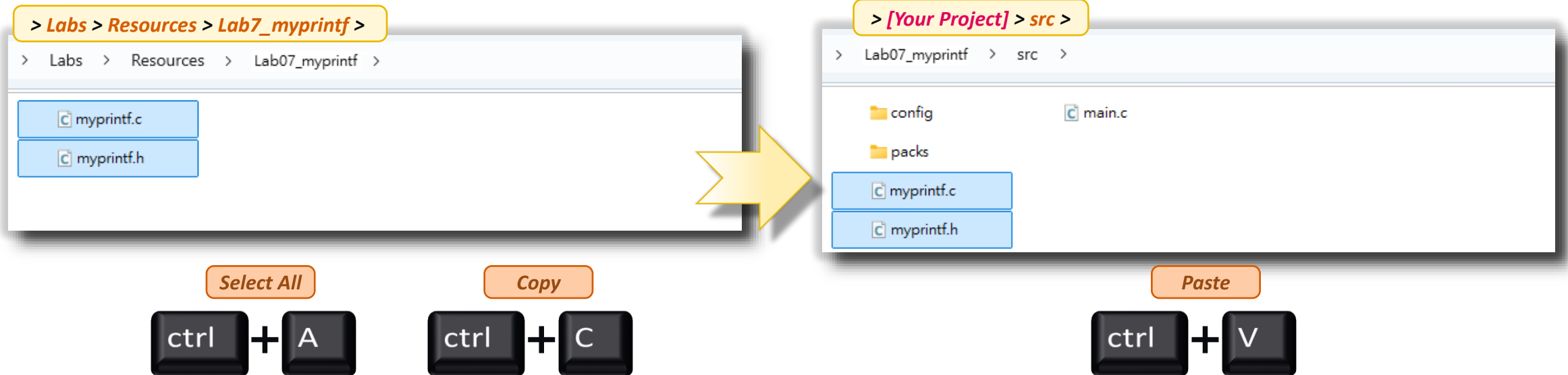
- In the previous LAB, we noticed that sending strings to Tera Term wasn't as convenient as using the standard C `printf()` function. Specifically, it lacked the ability to format variable outputs and display text at fixed screen positions.
- In this LAB, we will implement a custom `myprintf()` function that offers similar formatting capabilities to C's `printf()`, allowing you to define variable types in the string and print them accordingly.
- Additionally, we will introduce the classic **VT100 console commands**, enabling features such as:
 - Clearing the screen
 - Printing text at specified coordinates
 - Changing text color and other effects
- Since **VT100 commands** can be complex, we've integrated a simple command parser within `myprintf()` to interpret custom, simplified directives that execute VT100 control sequences easily and intuitively.
- All of this has been pre-designed for you — simply add the `myprintf.c` and `myprintf.h` files to your project to start using it right away.

Let's go !

⚙️ Lab7 myprintf - SERCOM-USART

Step 1

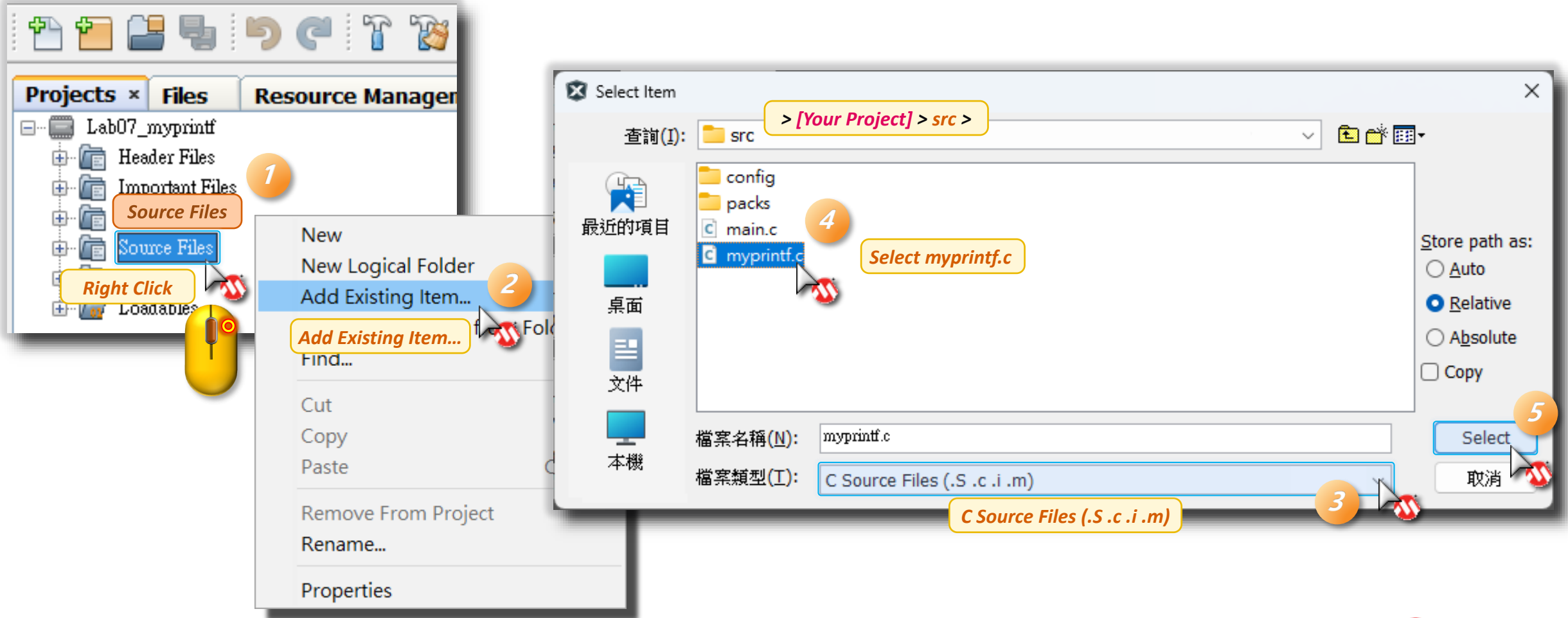
- Copy all pre-developed files
in `\Labs\Resource\Lab07_myprintf\` to the project folder `\[Your Project]\src\`
 - Custom printf Function Header File : myprintf.h
 - Custom printf Function Source File : myprintf.c



⚙️ Lab7 myprintf - SERCOM-USART

Step 2

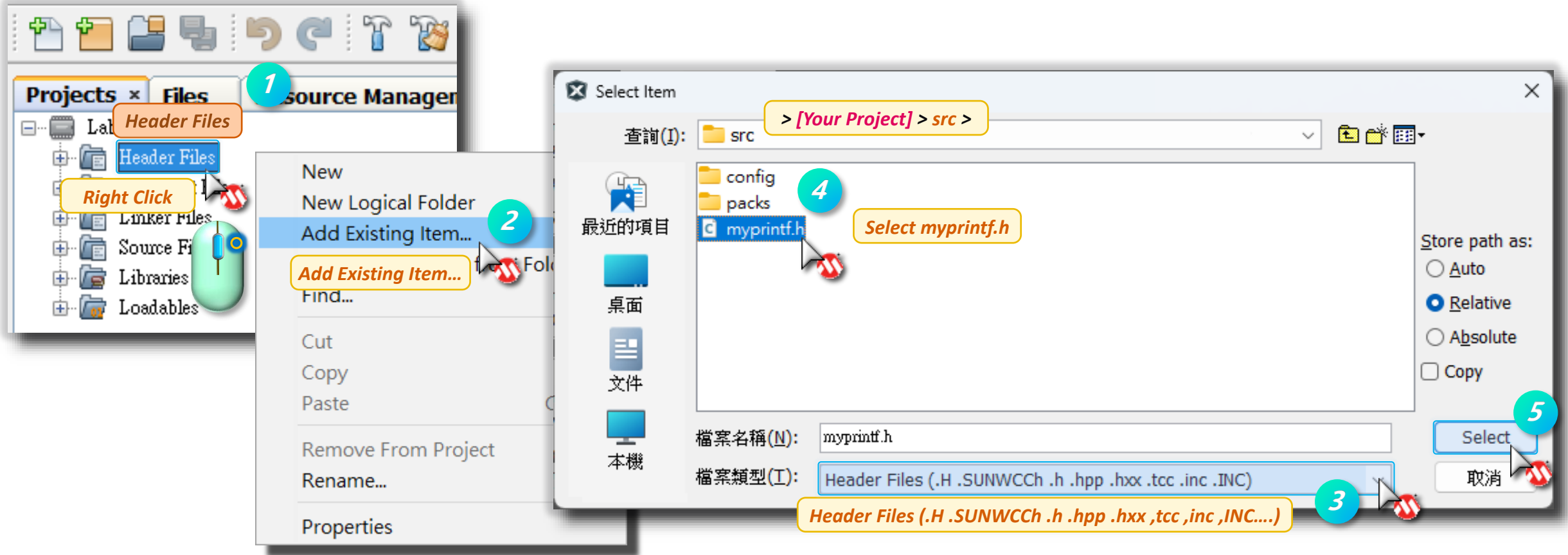
- In MPLAB X IDE, add new **Source Files (*.c)** under **\[Your Project]\src** to your project.



⚙️ Lab7 myprintf - SERCOM-USART

Step 3

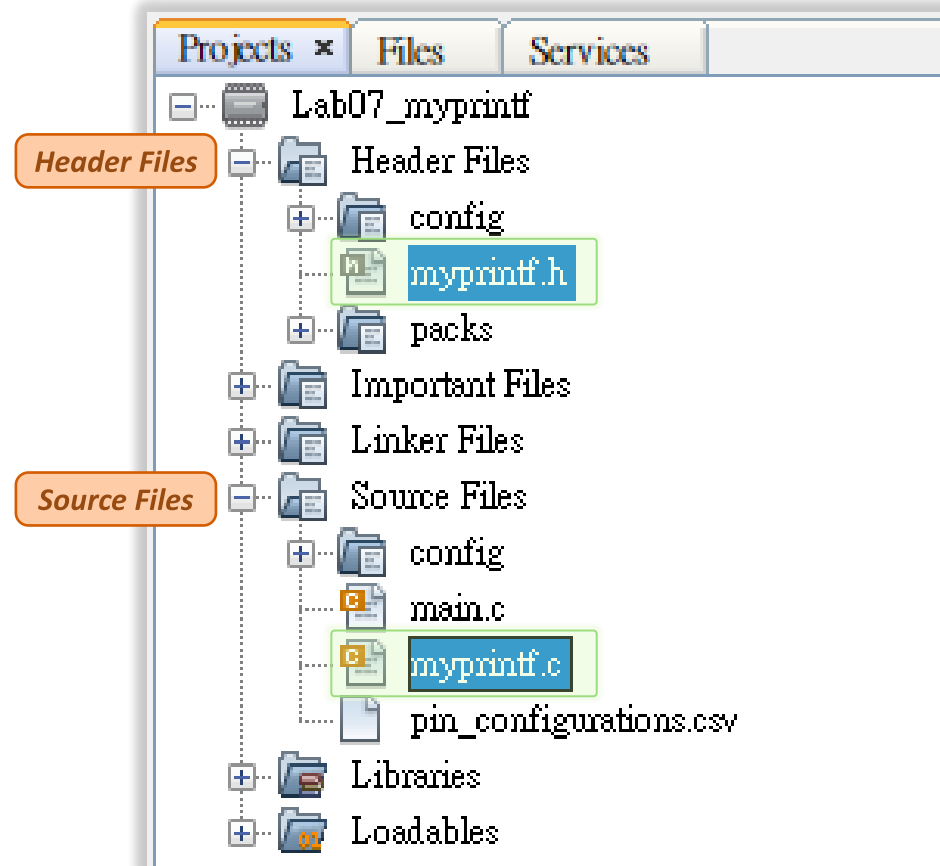
- In MPLAB X IDE, add new **Header Files (*.h)** under **\[Your Project]\src** to your project.



⚙️ Lab7 myprintf - SERCOM-USART

Step 4

- In MPLAB X IDE, check below files been added to your project.



VT100 Terminal

The Legacy Terminal

- **Introduction**

- Manufacturer: Digital Equipment Corporation (DEC).
- Release Year: 1978.
- Type: Video Display Terminal (VDT).
- Standard: First to implement ANSI X3.64 (ANSI escape codes).

- **Key Features**

- Screen Size: 80 columns × 24 rows (25th row for status).
- Display: Text-based, monochrome CRT.
- Scroll & Cursor Control: Via ANSI escape sequences.
- Keyboard: Full-sized with function keys.
- Connection: RS-232 serial port.

- **Why 80×24?**

- 80 columns from IBM punch card standard.
- 24 rows due to screen readability and hardware limitations.
- Became a de facto layout for CLI interfaces.



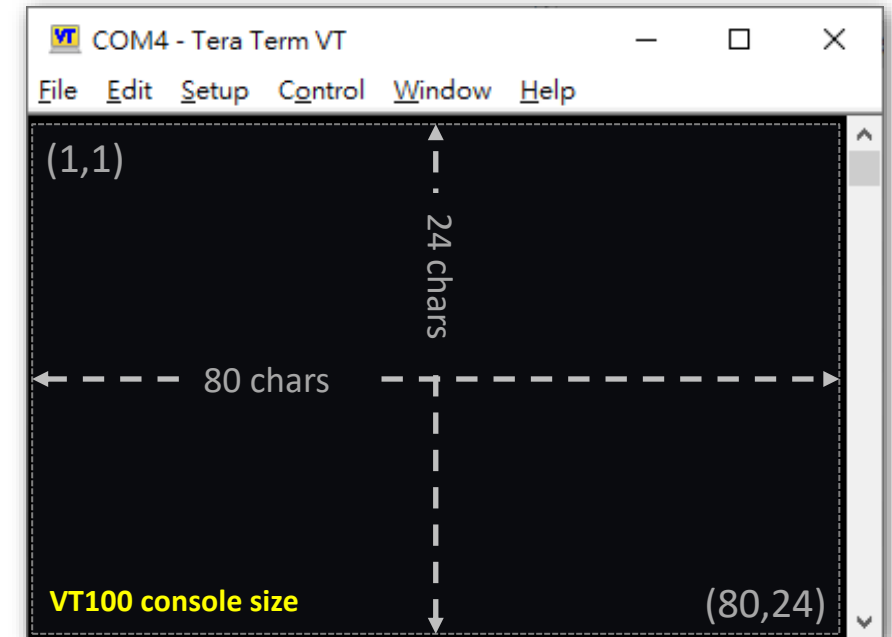
Custom VT100 Command

Pre-designed custom VT100 command usage in this Lab

- **#RST;** **VT100 Command :** `"\033[0m"`
 - Description :
 - Reset VT100 console formatting to default.
 - Example :
`myprintf( "#RST;" );`
- **#CLS;** **VT100 Command :** `"\033[2J\033[H"`
 - Description :
 - Clear the entire screen and move the cursor to the home position (top-left).
 - Example :
`myprintf( "#CLS;" );`
- **#HID;** **VT100 Command :** `"\033[?251"`
 - Description :
 - Hide the cursor from view.
 - Example :
`myprintf( "#HID;" );`
- **@x,y;** **VT100 Command :** `"\033[y;xH"`
 - Description :
 - Move the cursor to the specified column x and row y (1-based position).
 - Example :
`myprintf("@10,5;Hello World!!");`

`myprintf.c`

** Appendix for more command*



** The VT100 console in Tera Term v5 could up to 118x28*

⚙️ Lab7 myprintf - SERCOM-USART

Step 5

- Open **main.c** , include **myprintf.h** and send custom VT100 command to **reset console, clean screen and hide cursor**.

```
#include <stddef.h>           // Defines NULL
#include <stdbool.h>          // Defines true
#include <stdlib.h>           // Defines EXIT_FAILURE
#include <string.h>
#include <stdio.h>
#include "definitions.h"      // SYS function prototypes
// TODO 7.01
#include "myprintf.h"

uint32_t i = 0;
...

int main ( void )
{
    SYS_Initialize ( NULL );

    TC0_TimerCallbackRegister(TC0_TimerExpired, (uintptr_t) NULL);
    TC0_TimerStart();

    TC1_TimerCallbackRegister(TC1_TimerExpired, (uintptr_t) NULL);
    TC1_TimerStart();

    SERCOM1_USART_ReadCallbackRegister(USART1_Receive, (uintptr_t) NULL);
    SERCOM1_USART_WriteCallbackRegister(USART1_Transmit, (uintptr_t) NULL);
    SERCOM1_USART_Read(USART1_ReceiveData, 1);
// TODO 7.02
    myprintf( "#RST;#CLS;#HID;" );    // Reset Console, Clear screen and Hide cursor

    while ( true )
```


⚙️ Lab7 myprintf - SERCOM-USART

Step 6

- Comment out the original `SERCOM1_USART_Write()` and `sprintf()` functions and use the `myrprintf()` function instead.

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

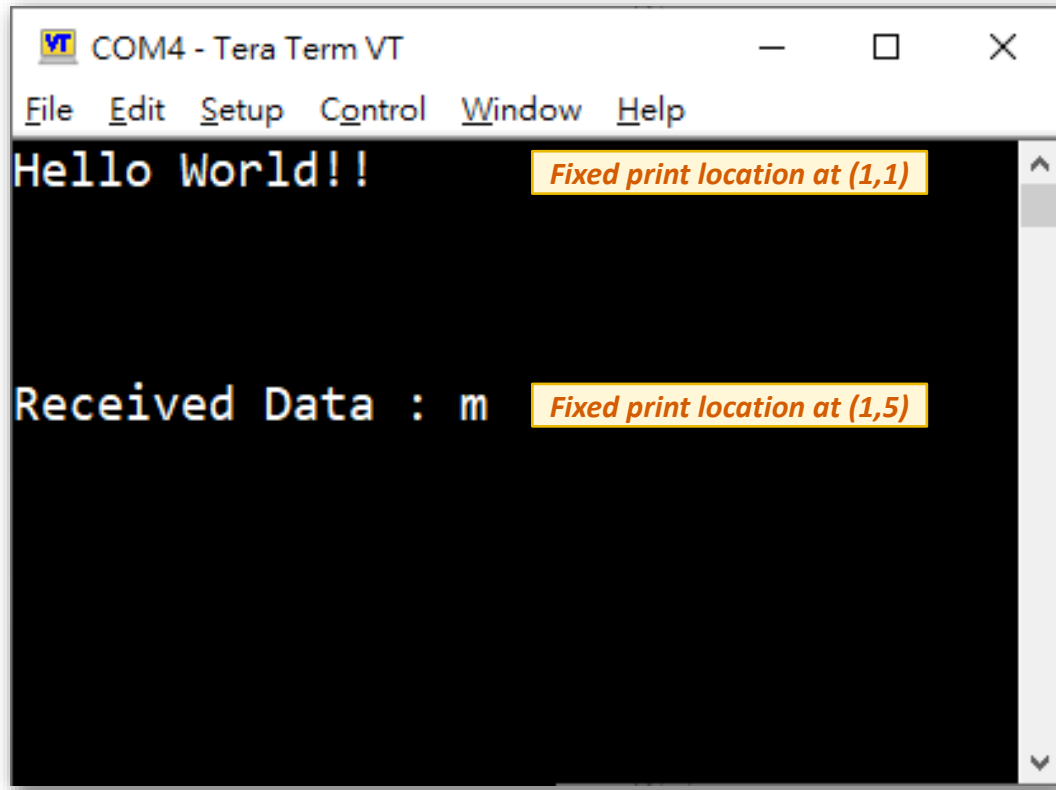
    if (TC0_HasExpired)
    {
        TC0_HasExpired = 0;
        LED1_Toggle();
// TODO 7.03
        // SERCOM1_USART_Write("Hello world!\r\n", 14);
        myrprintf( "@1,1;Hello World!!" );
    }

    if (TC1_HasExpired)
    {
        TC1_HasExpired = 0;
        LED2_Toggle();
    }

    if (USART1_IsReceived)
    {
        USART1_IsReceived = 0;
// TODO 7.04
        // sprintf( (char*)USART1_TransmitData, "Received Data : %1c\r\n", USART1_ReceiveData[0] );
        // SERCOM1_USART_Write( USART1_TransmitData, strlen((char*)USART1_TransmitData) );
        myrprintf( "@1,5;Received Data : %1c", USART1_ReceiveData[0] );
        SERCOM1_USART_Read( USART1_ReceiveData, 1 );
    }
}
```

🔧 Lab7 myprintf - SERCOM-USART

Result

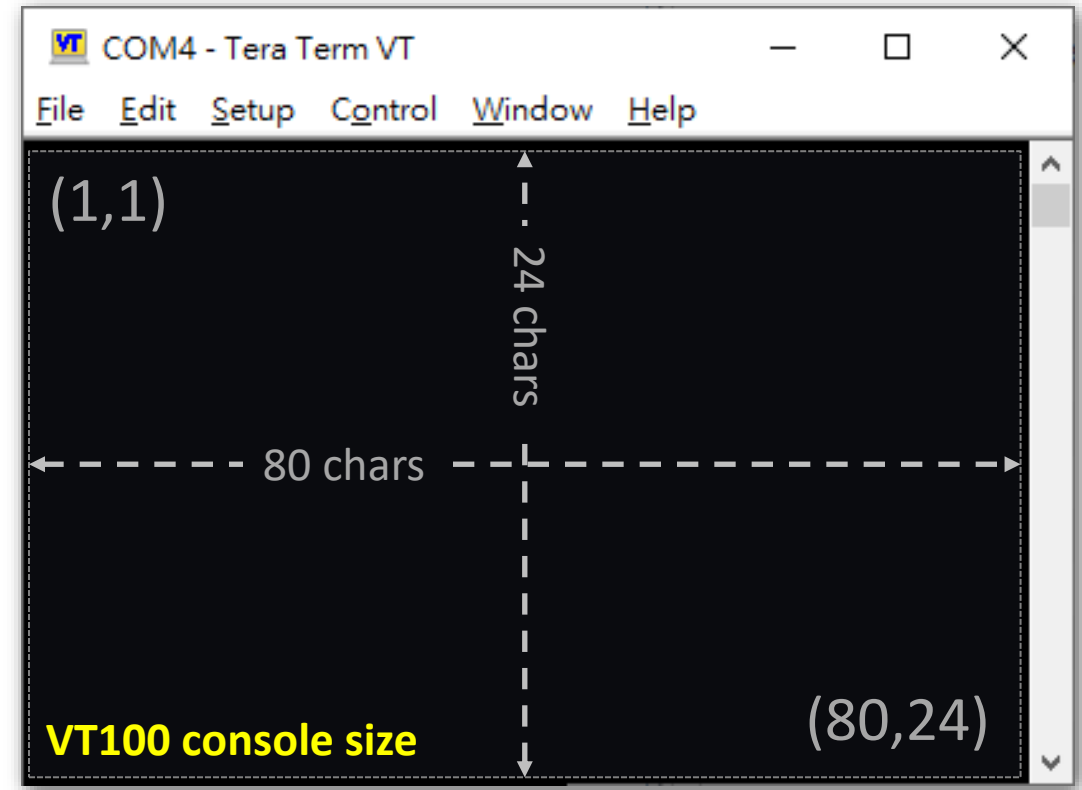


COM4 - Tera Term VT

File Edit Setup Control Window Help

Hello World!! *Fixed print location at (1,1)*

Received Data : m *Fixed print location at (1,5)*



** The VT100 console in Tera Term v5 could up to 118x28*

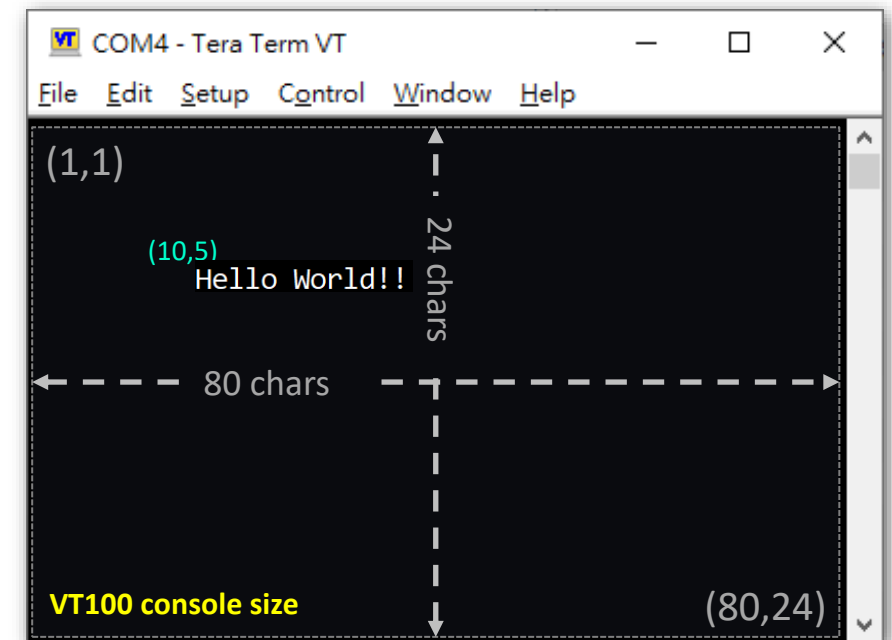
Appendix : Custom VT100 command

myprintf

Custom VT100 Command

Pre-designed custom VT100 command

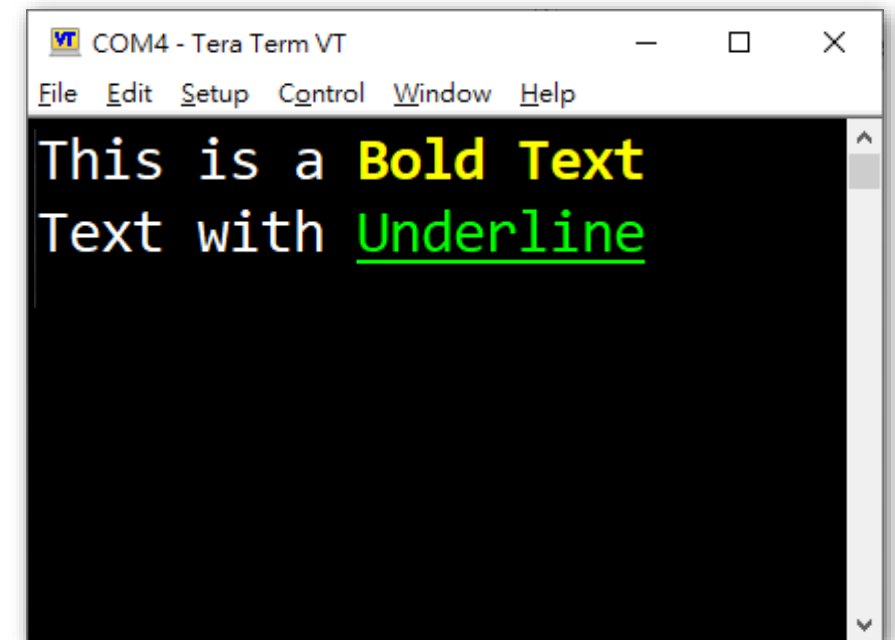
- **#RST;** **VT100 Command :** `"\033[0m"`
 - Description :
 - Reset VT100 console formatting to default.
 - Example :
`myprintf( "#RST;" );`
- **#CLS;** **VT100 Command :** `"\033[2J\033[H"`
 - Description :
 - Clear the entire screen and move the cursor to the home position (top-left).
 - Example :
`myprintf( "#CLS;" );`
- **#HID;** **VT100 Command :** `"\033[?251"`
 - Description :
 - Hide the cursor from view.
 - Example :
`myprintf( "#HID;" );`
- **@x,y;** **VT100 Command :** `"\033[y;xH"`
 - Description :
 - Move the cursor to the specified column x and row y (1-based position).
 - Example :
`myprintf("@10,5;Hello World!!");`



Custom VT100 Command

Pre-designed custom VT100 command

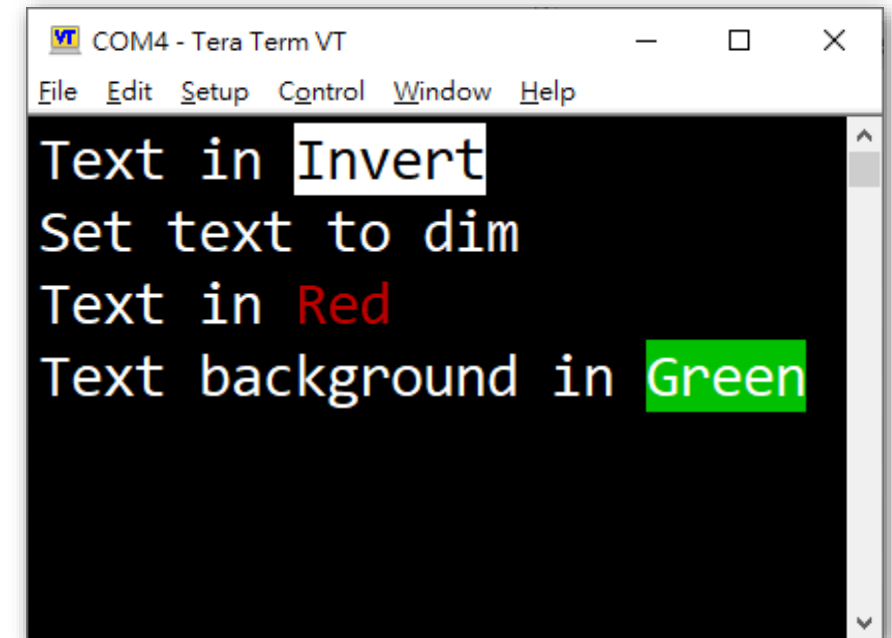
- **#C2E;** **VT100 Command :** `"\033[K"`
 - Description :
 - Clear from the current cursor position to the end of the line.
 - Example :
`myprintf( "#C2E;" );`
- **#CLR;** **VT100 Command :** `"\033[2K"`
 - Description :
 - Clear the entire current line(Row).
 - Example :
`myprintf( "#CLR;" );`
- **#BOL;** **VT100 Command :** `"\033[1m"`
 - Description :
 - Set text to bold style.
 - Example :
`myprintf( "This is a #BOL;Bold Text" );`
- **#UND;** **VT100 Command :** `"\033[4m"`
 - Description :
 - Set text to underline style.
 - Example :
`myprintf( "Text with #UND;Underline" );`



Custom VT100 Command

Pre-designed custom VT100 command

- **#INV;** **VT100 Command :** `"\033[7m"`
 - Description :
 - Enable inverse video (swap foreground and background colors).
 - Example :
`myprintf( "Text in #INV;Invert" );`
- **#DIM;** **VT100 Command :** `"\033[2m"`
 - Description :
 - Set text to dim (faint brightness).
 - Example :
`myprintf( "Set #DIM;text to dim" );`
- **#FGC:RRGGBB;** **VT100 Command :** `"\033[38;2;RR;GG;BBm"`
 - Description :
 - Set the foreground color of text using 24-bit RGB format.
 - Example :
`myprintf( "Text in #FGC:FF0000;Red" );`
- **#BGC:RRGGBB;** **VT100 Command :** `"\033[48;2;RR;GG;BBm"`
 - Description :
 - Set the background color of text using 24-bit RGB format.
 - Example :
`myprintf( "Text background in #BGC:00FF00;Green" );`

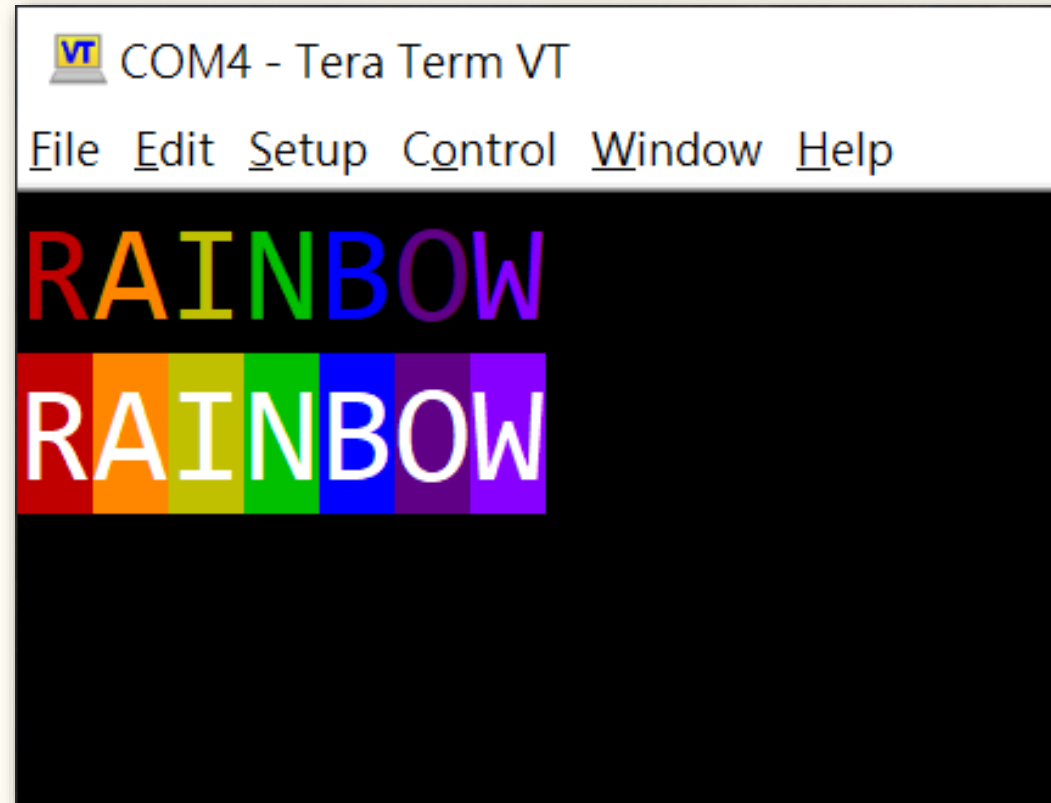


Custom VT100 Command

Rainbow Color Text example (copy code from \Labs\CopyPaste\Lab07_myprintf.txt)

```
// Reset Console, Clear screen and Hide cursor
myprintf( "#RST;#CLS;#HID;" );
myprintf(
    "#FGC:FF0000;R"
    "#FGC:FF7F00;A"
    "#FGC:FFFF00;I"
    "#FGC:00FF00;N"
    "#FGC:0000FF;B"
    "#FGC:4B0082;O"
    "#FGC:8B00FF;W"
    "#RST;\r\n"
);

myprintf(
    "#BGC:FF0000;R"
    "#BGC:FF7F00;A"
    "#BGC:FFFF00;I"
    "#BGC:00FF00;N"
    "#BGC:0000FF;B"
    "#BGC:4B0082;O"
    "#BGC:8B00FF;W"
    "#RST;\r\n"
);
```



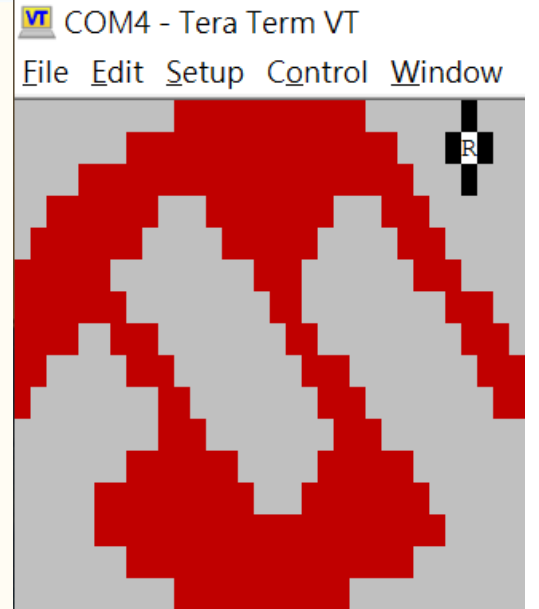
Custom VT100 Command

Microchip Logo Text example (copy code from \Labs\CopyPaste\Lab07_myprintf.txt)

```
// Picture to Text (Please use monochrome PNG images, the maximum output size is 118x28 pixels as ASCII image)
// http://www.digole.com/tools/PicturetoC_Hex_converter.php
#define ASCII_IMAGE_WIDTH 32 // 32 pixels is 32/8 = 4 bytes per Row
#define ASCII_IMAGE_HEIGHT 16 // 16 pixels
#define ASCII_IMAGE_PIXEL ' ' // The ASCII pixel character
#define ASCII_PIXEL_COLOR "#BGC:FF0000;" // The color "1" of image
#define ASCII_BKGND_COLOR "#BGC:FFFFFF;" // The color "0" of image
uint8_t ImageArray[(ASCII_IMAGE_WIDTH/8)*ASCII_IMAGE_HEIGHT] =
{
    0x00,0x3f,0xfc,0x00,0x01,0xff,0xff,0x00,0x0f,0xff,0xff,0x80,0x3f,0x8f,0xf1,0xc0,
    0x7f,0x07,0xe0,0xe0,0xfc,0x01,0xc0,0x70,0xfe,0x00,0xc0,0x1c,0xf3,0x80,0x60,0x0e,
    0xc1,0xc0,0x38,0x07,0x80,0x60,0x1c,0x03,0x00,0x70,0x0e,0x00,0x01,0xfc,0x1f,0x80,
    0x07,0xfe,0x3f,0xc0,0x07,0xff,0xff,0xe0,0x01,0xff,0xff,0x80,0x00,0x3f,0xf8,0x00
};

// Reset Console, Clear screen and Hide cursor
myprintf( "#RST;#CLS;#HID;" );
// ASCII Image Decode
for (int y = 0; y < ASCII_IMAGE_HEIGHT; y++)
{
    for (int byte_idx = 0; byte_idx < (ASCII_IMAGE_WIDTH/8); byte_idx++)
    {
        uint8_t byte = ImageArray[y * (ASCII_IMAGE_WIDTH/8) + byte_idx];
        for (int bit = 0; bit < 8; bit++)
        {
            int x = byte_idx * 8 + bit + 1;
            myprintf("@%d,%d;%s%c", x, y+1, (byte&(0x80>>bit)) ? ASCII_PIXEL_COLOR : ASCII_BKGND_COLOR, ASCII_IMAGE_PIXEL);
        }
    }
}
// Reset Console
myprintf( "#RST;" );

// Microchip Logo Copyright circle
myprintf( "@29,1;#BGC:000000; @28,2;#BGC:000000; @30,2;#BGC:000000; @29,3;#BGC:000000; " );
// Microchip Logo Copyright R
myprintf( "@29,2;#INV;R#RST;" );
```

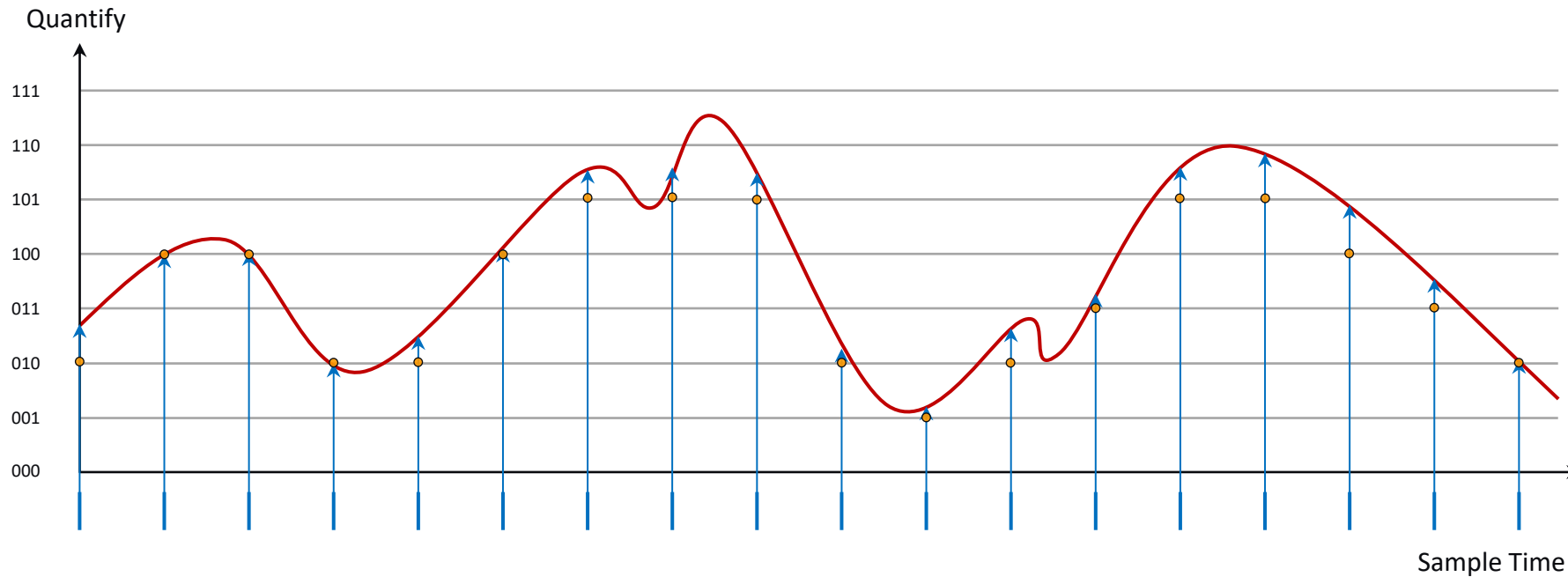


** The better Font for this demo is using "Consolas" or "Courier".*

High Speed ADC Architecture

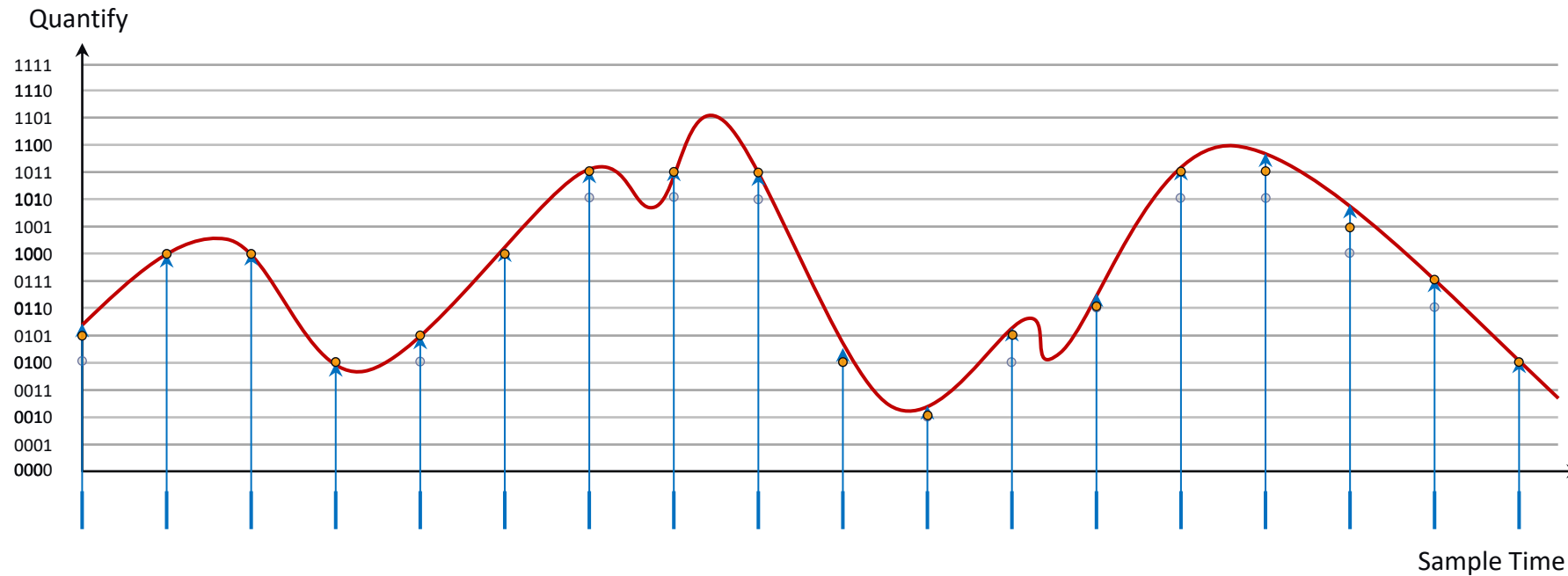
What's ADC ?

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



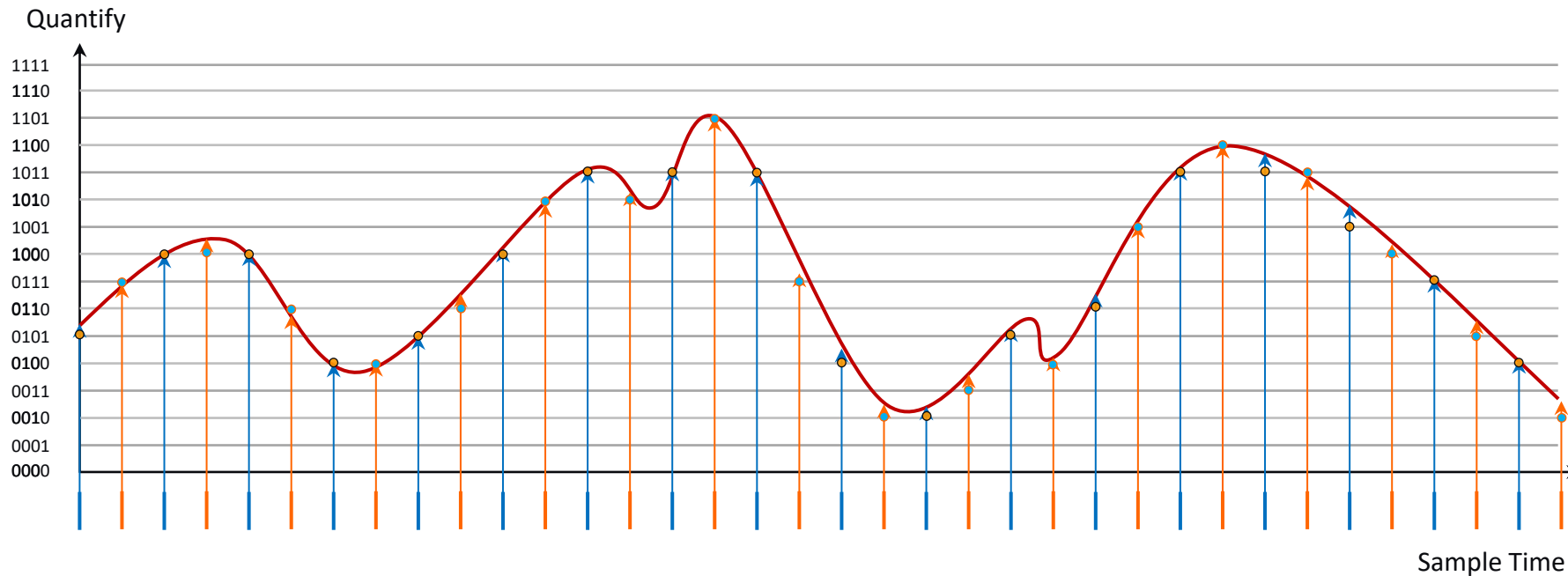
Resolution

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



Sample Rate

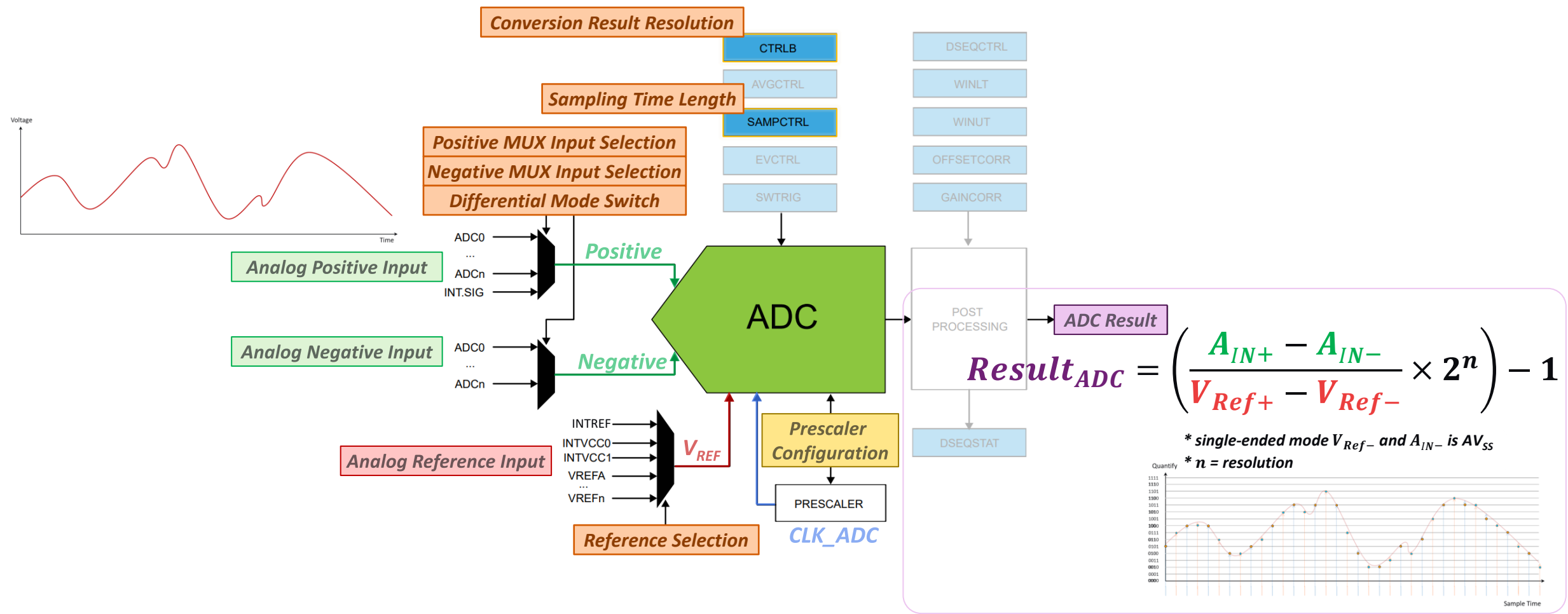
- The analog signal is required to define the rate at which new digital values are sampled from the analog signal.
- This faithful reproduction is only possible if the sampling rate is higher than twice the highest frequency of the signal. (wiki)



PIC32CX-SG Analog-to-Digital Converter

- The PIC32CX-SG Analog-to-Digital Converter (ADC) converts analog signals to digital values, has **up to 12-bit resolution**, and is capable of a sampling rate of **up to 1Mps**.
- The ADC has **two core (ADC0 and ADC1)** with separate dedicated circuit, and **up to 23 analog inputs per ADC core**, including single-ended and differential modes, and can measure external or internal signals. Both signed and unsigned results are supported.
- **ADC Advanced Features :**
 - Windowing monitor with selectable channel
 - Built-in internal reference and external reference options
 - Event-triggered conversion for accurate timing
 - Optional DMA transfer of conversion settings or result
 - Hardware gain and offset compensation
 - Averaging and oversampling with decimation to support up to 16-bit result
 - Selectable sampling time
 - Flexible Power or Throughput rate management

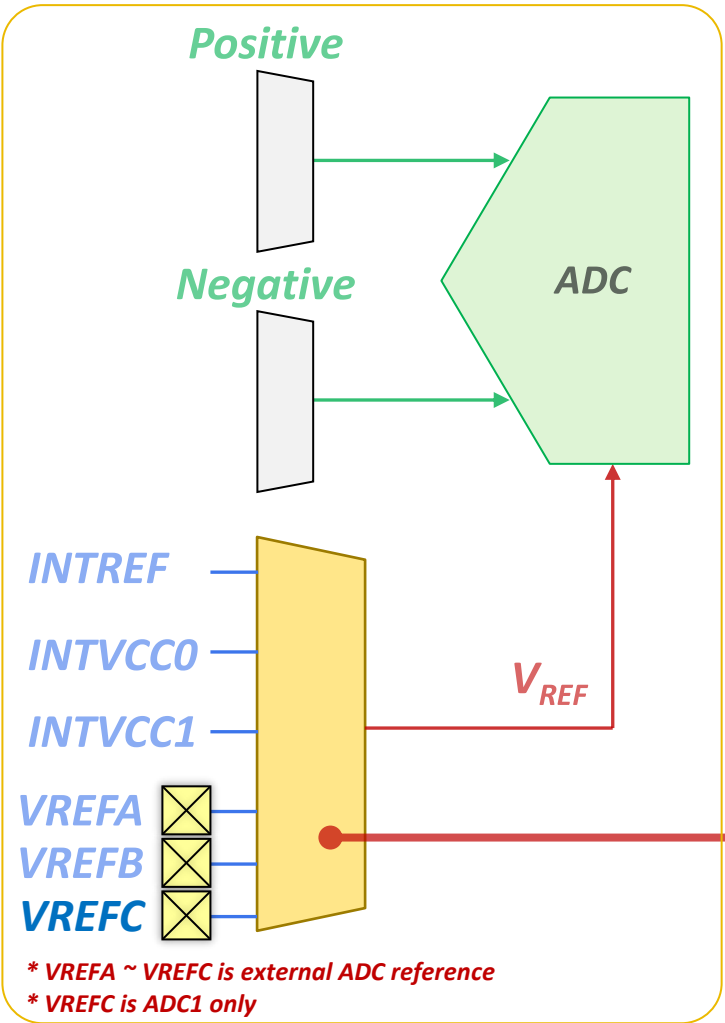
ADC Block Diagram



* ADC contains powerful functional, the course only focus on basic controls.
* For more information, please refer to the datasheet.

ADC Reference Voltage

- Select the ADC reference suitable for your analog input range.



Conversion range:

– Single-ended mode

$0V \sim VREF$

– Differential mode

$-VREF \sim VREF$

VREFA ~ VREFC range

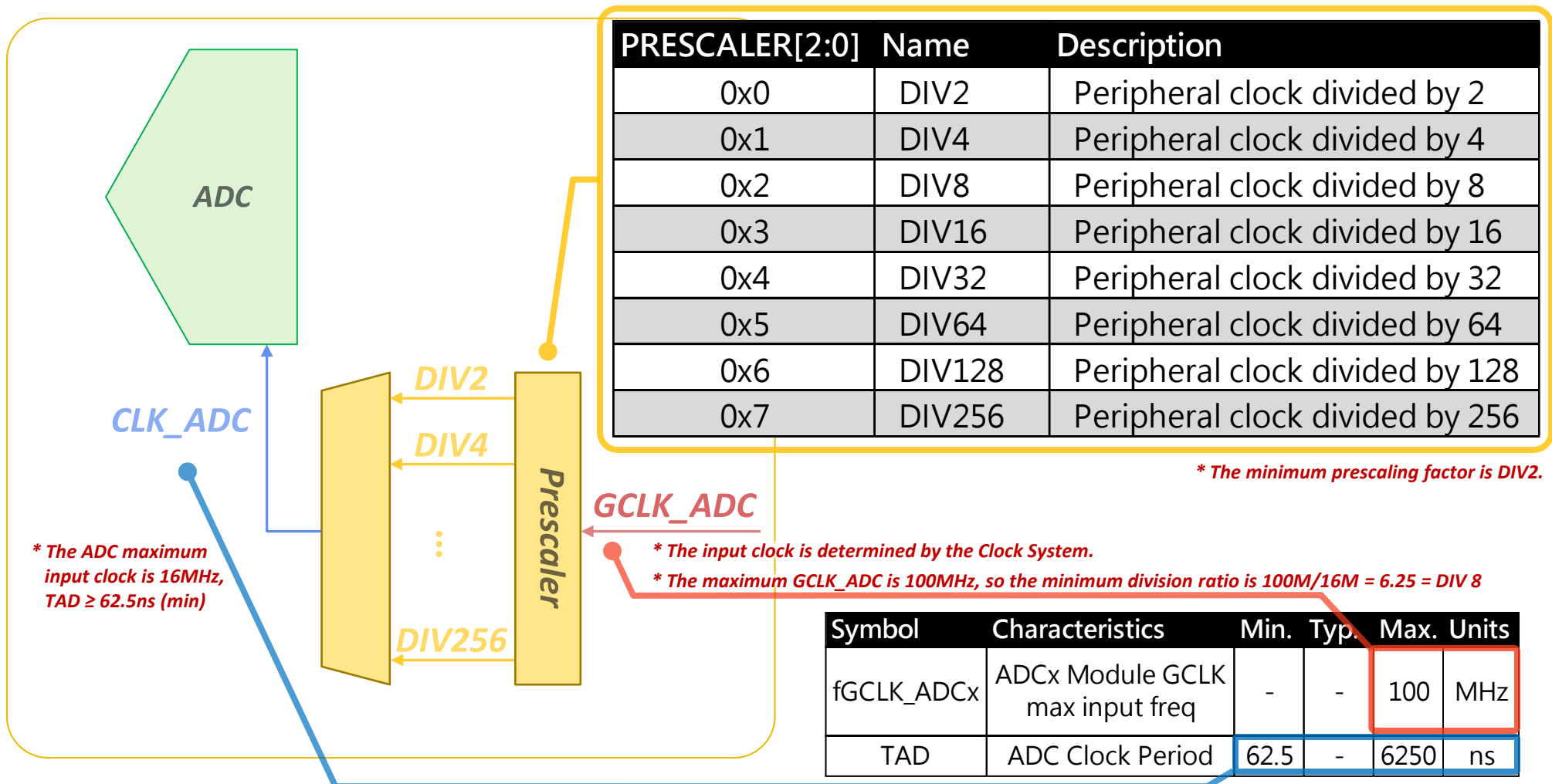
$2.4V \sim (AVDD - 0.4V)$

** VREFA ~ VREFC is external ADC reference*

REFSEL[3:0]	Name	Description
0x0	INTREF	internal bandgap reference, refer to the VREF.
0x1		Reserved
0x2	INTVCC0	1/2 AVDD (only for AVDD > 2.0V)
0x3	INTVCC1	AVDD <i>*DC Characteristics of AVDD is 1.71v ~ 3.63v</i>
0x4	VREFA	External reference
0x5	VREFB	External reference
0x6	VREFC	External reference (ADC1 only)
other	-	Reserved

ADC Input Clock Pre-scaler

- Select the ADC prescaler to ensure the ADC input clock less than or equal to 16MHz.



ADC Trigger Source

- There are three way to be started data conversion.

- **Software Trigger**

One shot conversion control via software.

- **Free Run**

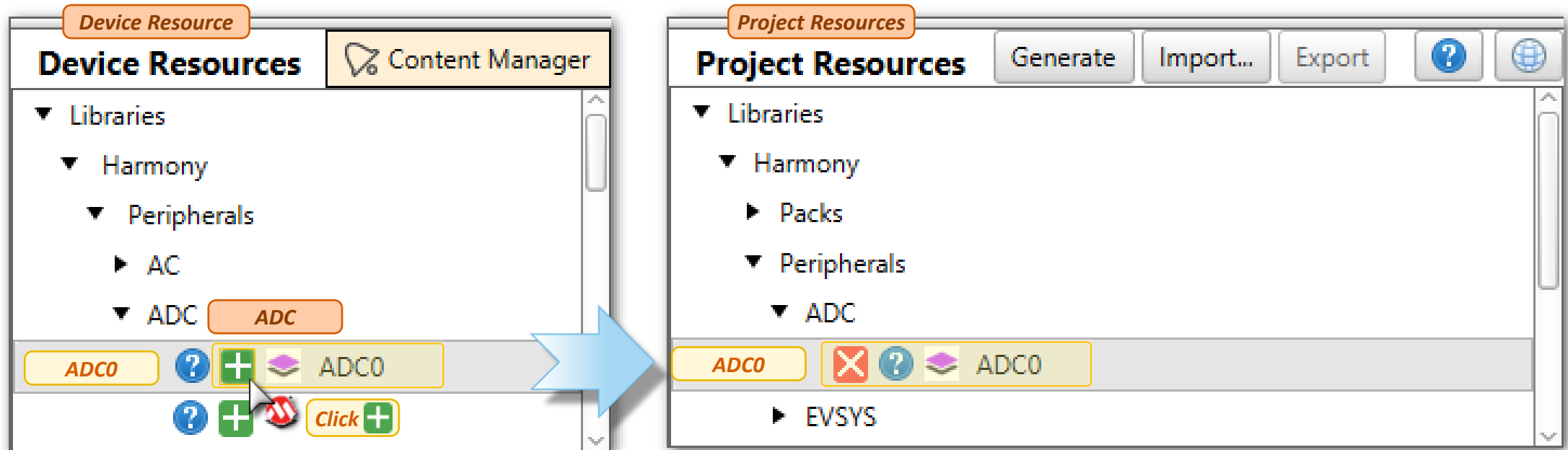
A free-running mode can be used to continuously convert an input channel. When using free-running mode the first conversion must be started, while subsequent conversions will start automatically at the end of previous conversions.

- **HW Event Trigger**

Through Event System peripheral to automatically conversion.

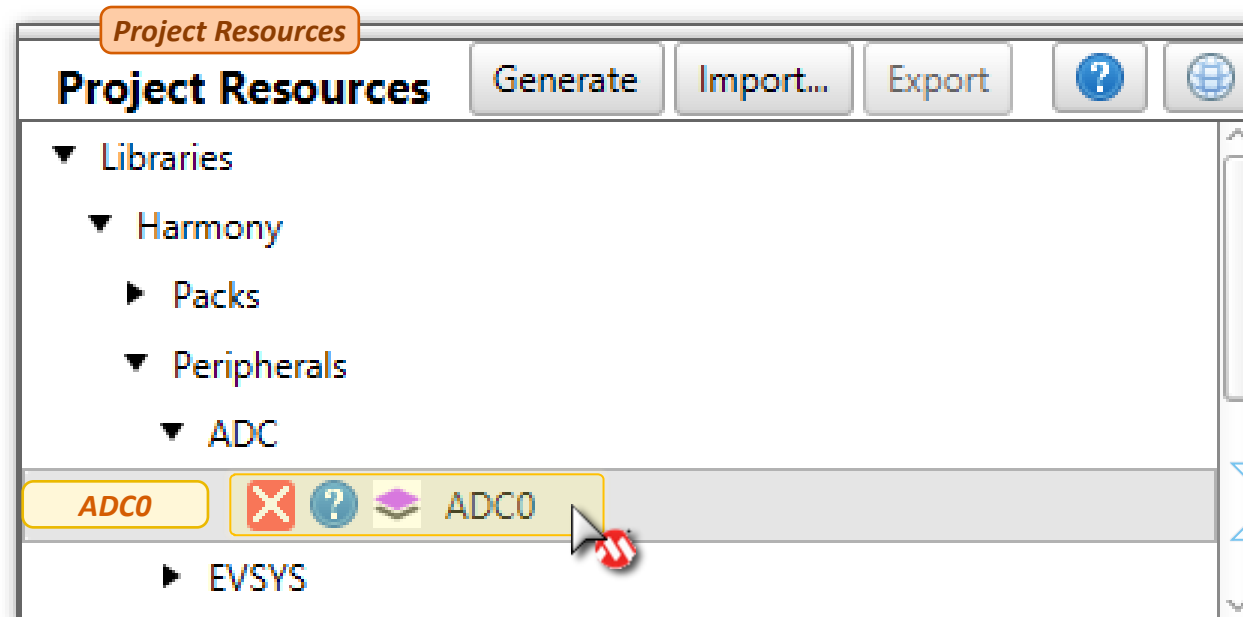
Add or Remove MCC Harmony Module

- Find required component in **Device Resources**.
- Click on **+** to add the module to your **Project Resources**.

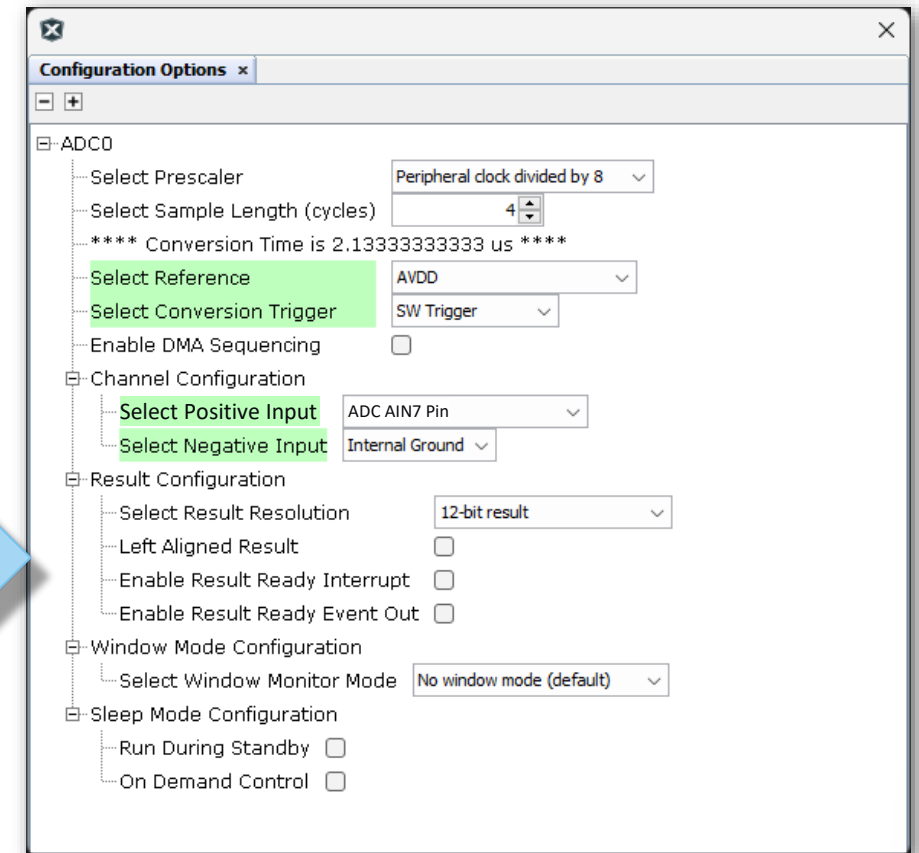


MCC Harmony Module Configuration Options Example

ADC Configuration Example (Configuration Options)



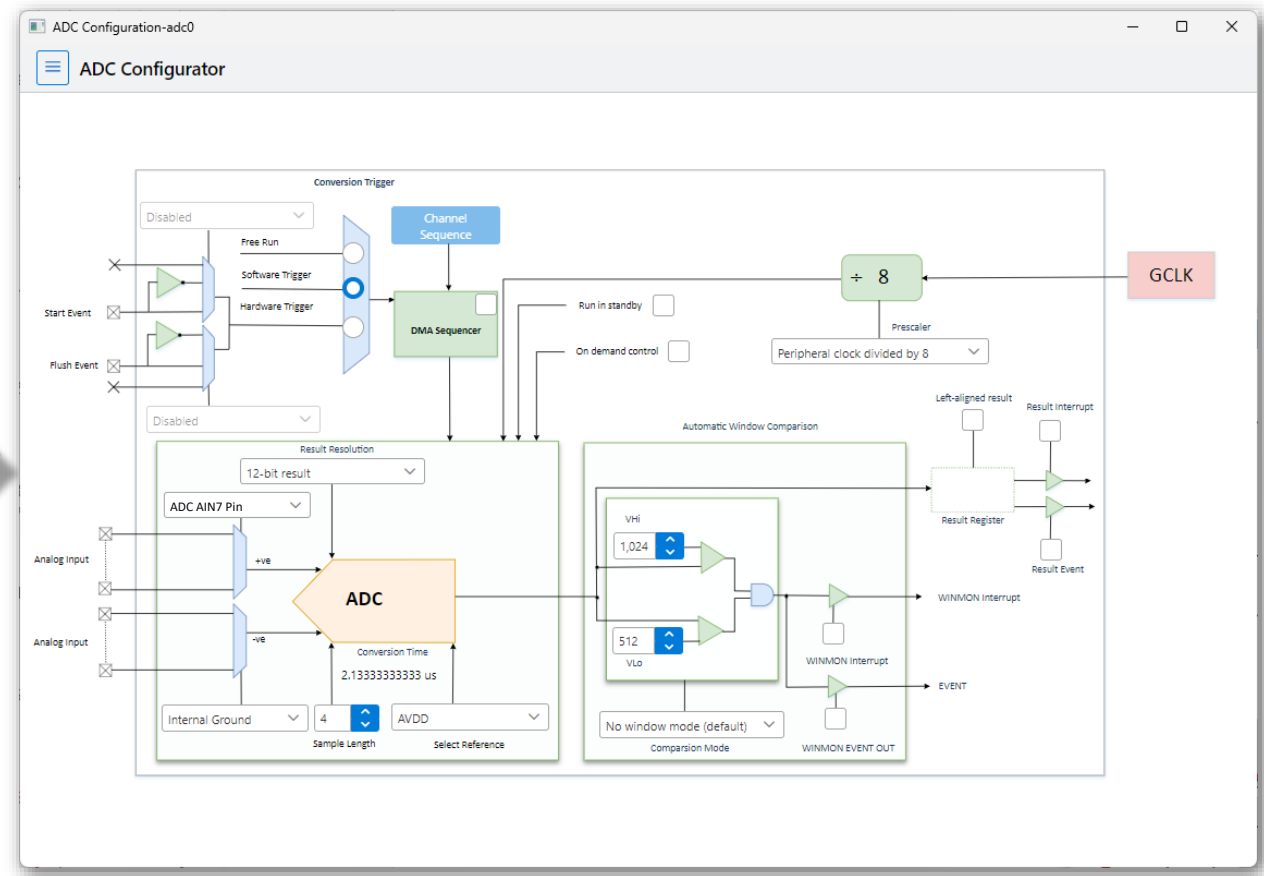
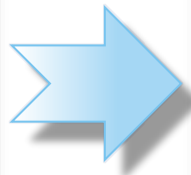
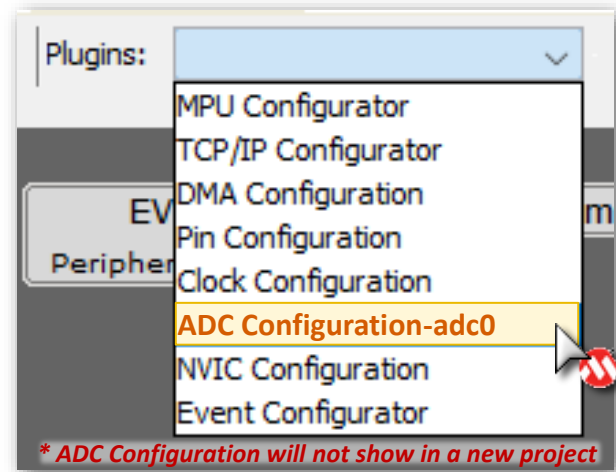
Configuration Options



MCC Harmony Module Configuration Options Example

ADC Configuration Example (Easy View Configuration)

ADC Easy View Configuration



* The setting values of Easy View Configuration and Configuration Options will be changed synchronously.

ADC PINMUX Configuration Example

- Remember assign **Positive Input** and **Negative Input** pin in ADC module, MCC will automatically switch to single-ended mode or differential mode according to the settings.
- For example : **Positive Input Pin as ADC0 AIN7 Pin (PA07)**
 Negative Input Pin as Internal Ground

ADC0

Select Prescaler: Peripheral clock divided by 8

Select Sample Length (cycles): 4

**** Conversion Time is 2.13333333333 us ****

Select Reference: AVDD

Select Conversion Trigger: SW Trigger

Enable DMA Sequencing: ☐

Channel Configuration

Select Positive Input: ADC AIN7 Pin

Select Negative Input: Internal Ground

Result Configuration

Select Result Resolution: 12-bit result

Left Aligned Result: ☐

Enable Result Ready Interrupt: ☐

Enable Result Ready Event Out: ☐

Window Mode Configuration

Sleep Mode Configuration

ADC PINMUX Configuration Example

- **Analog pins** will all belong to **PINMUX function group B**.
- For example : **PA07** as **ADC0 AIN[7]** (ADC Core 0 analog input Channel 7)
PC00 as **ADC1 AIN[10]** (ADC Core 1 analog input Channel 10)
PC01 as **ADC1 AIN[11]** (ADC Core 1 analog input Channel 11)

	128 Pin	Port Group	A	B					
	TQFP	Pad Name	EIC	ANAREF	ADC0	ADC1	AC	DAC	PTC
PA07	24	PA07	EXTIN[7]	-	AIN[7]	-	AIN[3]	-	X5 / Y6
PC00	3	PC00	EXTINT[0]	-	-	AIN[10]	-	-	-
PC01	4	PC01	EXTINT[1]	-	-	AIN[11]	-	-	-

Pin Table		PC00	PC01	PA02	PA07
Package: TQFP128		3	4	9	24
Module	Function				
ADC0	ADC0_AIN7/X5/Y5				
ADC1	ADC1_AIN10				
	ADC1_AIN11				

Lab8 ADC Polling Software Trigger

🔧 Lab8 ADC Polling Software Trigger

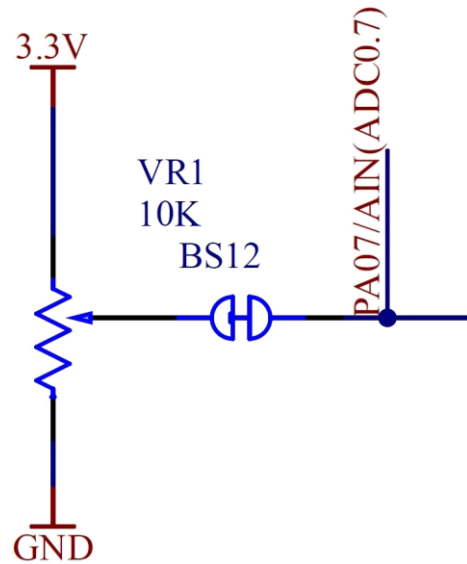
- Try to add **ADC-0** module to project.
- Initialize the ADC and get the conversion value from **VR1 (ADC0 AIN[7])** periodically, and then send it to PC by using UART .
- ADC set to **Software Trigger, 12 bits resolution, Single-End** mode.

Schematic	Pin	Analog input
Potentiometer (VR1)	PA07	ADC0 CH.7
Temperature Sensor (MCP9700A)	PC00	ADC1 CH.10
Light Sensor (TEMT6000)	PC01	ADC1 CH.11

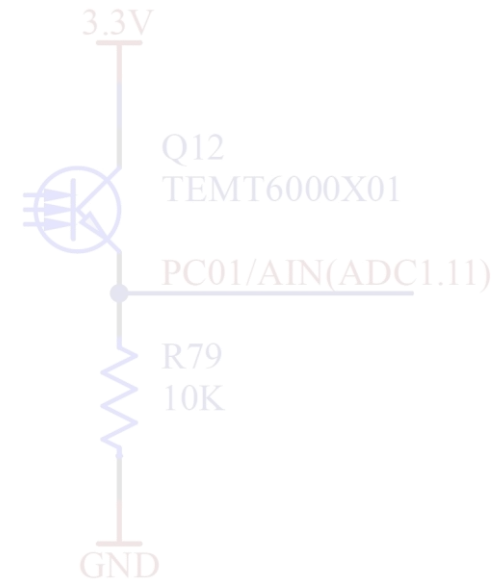
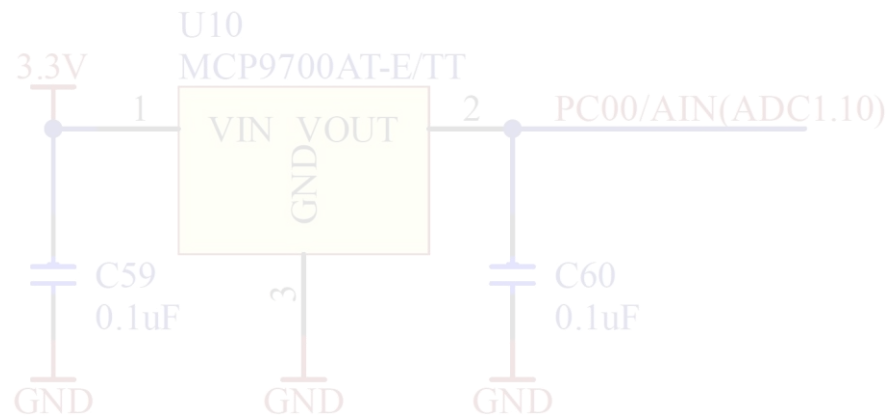
Let's go !

⚙️ Lab8 ADC Polling Software Trigger

Check Schematic



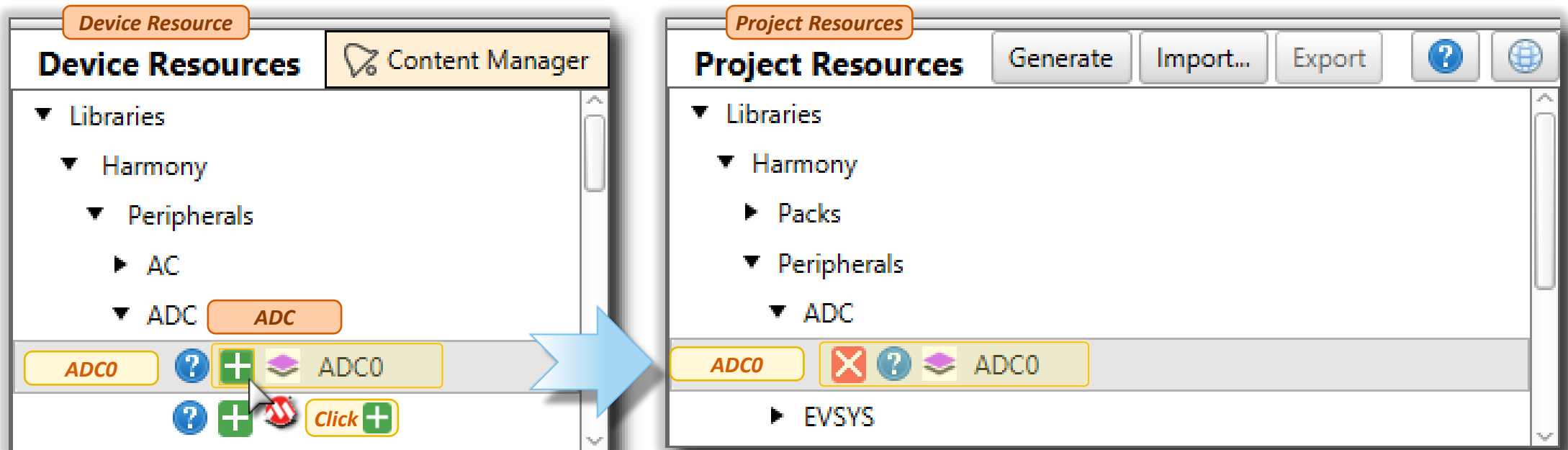
Schematic	Pin	Analog input
Potentiometer (VR1)	PA07	ADC0 CH.7
Temperature Sensor (MCP9700A)	PC00	ADC1 CH.10
Light Sensor (TEMT6000)	PC01	ADC1 CH.11



⚙️ Lab8 ADC Polling Software Trigger

Step 1

- Find **ADC0** component in **Device Resources**, we use it for sample analog input pin.
- Click on **+** to add the **ADC0** module to your **Project Resources**.



⚙️ Lab8 ADC Polling Software Trigger

Step 2 (Use Configuration Options)

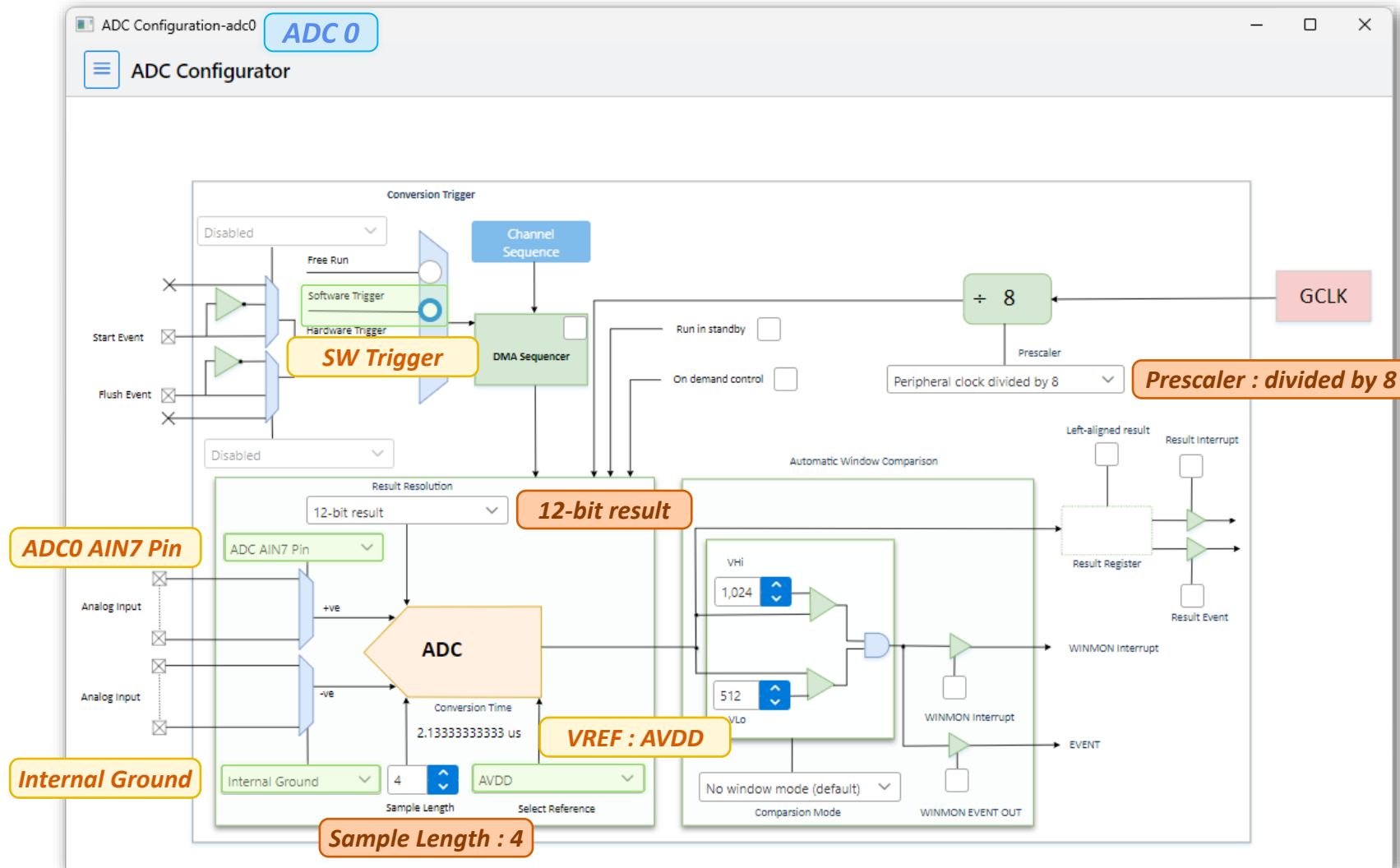
The screenshot shows the 'Configuration Options' dialog for ADC0. The settings are as follows:

- Select Prescaler:** Peripheral clock divided by 8 (Callout: *Prescaler : divided by 8*)
- Select Sample Length (cycles):** 4 (Callout: *Sample Length : 4*)
- **** Conversion Time is 2.1333333333 us ******
- Select Reference:** AVDD (Callout: *VREF : AVDD*)
- Select Conversion Trigger:** SW Trigger (Callout: *SW Trigger*)
- Enable DMA Sequencing:** ☐
- Channel Configuration:**
 - Select Positive Input:** ADC AIN7 Pin (Callout: *ADC0 AIN7 Pin*)
 - Select Negative Input:** Internal Ground (Callout: *Internal Ground*)
- Result Configuration:**
 - Select Result Resolution:** 12-bit result (Callout: *12-bit result*)
 - Left Aligned Result:** ☐
 - Enable Result Ready Interrupt:** ☐
 - Enable Result Ready Event Out:** ☐
- Window Mode Configuration:** ☐
- Sleep Mode Configuration:** ☐

* The setting values of Easy View Configuration and Configuration Options will be changed synchronously.

⚙️ Lab8 ADC Polling Software Trigger

Step 2 (Use Easy View)



** The setting values of Easy View Configuration and Configuration Options will be changed synchronously.*

Lab8 ADC Polling Software Trigger

Step 3

- Pin Settings of the **VR1 (ADC0 AIN[7])** .

Pin Settings

Project Graph x Pin Settings x Pin Diagram x Pin Table x

Order: Ports Ports Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
23	PA06		ADC0_AIN7	Digital	High Impedance	Low	☐	☐	NORMAL
24	PA07		ADC0_AIN7/X5/Y5	Analog	High Impedance	n/a	☐	☐	NORMAL
33	PA08		Available	Digital	High Impedance	Low	☐	☐	NORMAL
34	PA09		Available	Digital	High Impedance	Low	☐	☐	NORMAL

Schematic	Pin	Analog input
Potentiometer (VR1)	PA07	ADC0 CH.7

⚙️ Lab8 ADC Polling Software Trigger

Step 4

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Generate' button highlighted with a red circle and a yellow 'Generate' label. The 'Output' window shows the following log:

```
15:17:50.827 INFO: Generation complete
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

ADC Polling

ADC n Interface Functions

- **ADC n _Initialize** : Initialize ADC n module
- **ADC n _Enable** : Enable ADC n module
- **ADC n _Disable** : Disable ADC n module
- **ADC n _ChannelSelect** : Configure channel input
- **ADC n _ConversionStart** : Start the ADC n conversion by Software
- **ADC n _ConversionStatusGet** : Check whether result is ready
- **ADC n _ConversionResultGet** : Read the conversion result

ADC Polling

ADC n Interface Functions

void ADC n _Enable (**void**)

- **Description:**
 - Enable ADC n module.

void ADC n _ConversionStart (**void**)

- **Description:**
 - Start the ADC n conversion by Software.

bool ADC n _ConversionStatusGet (**void**)

- **Description:**
 - Return ADC n conversion status.

uint16_t ADC n _ConversionResultGet (**void**)

- **Description:**
 - Return ADC n conversion result.

ADC Polling

ADC n Interface Functions

void ADC n _ChannelSelect (**ADC_POSINPUT** positiveInput, **ADC_NEGINPUT** negativeInput)

- **Description:**

- Configure channel input.

- **Parameter:**

- positiveInput : Positive analog input pin.
- negativeInput : Negative analog input pin.

- **Example :**

```
uint16_t ADC_value;  
ADCn_Enable();  
...  
ADCn_ChannelSelect( ADC_POSINPUT_AIN $x$ , ADC_INPUTCTRL_MUXNEG_AVSS );  
ADCn_ConversionStart();  
while( ADC0_ConversionStatusGet() != ADC_STATUS_RESRDY ){}  
ADC_value = ADCn_ConversionResultGet();
```

ADC Polling Code Example

ADC Polling method Software Trigger Example

- Use functions to enable the ADC module and periodically sampling ADC input.

```
int main ( void )
{
    ...
    ADC0_Enable();

    while( true )
    {
        ...
        ADC0_ChannelSelect( ADC_POSINPUT_AINx, ADC_INPUTCTRL_MUXNEG_AVSS );
        ADC0_ConversionStart();
        while( ADC0_ConversionStatusGet() != ADC_STATUS_RESRDY ){}
        ADC0_Result = ADC0_ConversionResultGet();
        ...

        /* Execution should not come here during normal operation */

        return ( EXIT_FAILURE );
    }
}
```

Enable ADC0 module

Switch the input channel of ADC0 to Channel x

ADC0 start sampling and conversion.

Wait for the ADC0 conversion to complete.

Get the ADC0 conversion result

ADC Factory Calibration

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

```
void ADCx_Initialize( void )
{
    ...
    /* Writing calibration values in BIASREFBUF, BIASCOMP and BIASR2R */
    ADCx_REGS->ADC_CALIB =(uint16_t) ((ADC_CALIB_BIASCOMP(((uint32_t*)SW0_ADDR) & ADCx_BIASCOMP_Msk) >>
        ADCx_BIASCOMP_POS))) \
        | ADC_CALIB_BIASR2R(((uint32_t*)SW0_ADDR) & ADCx_BIASR2R_Msk) >> ADCx_BIASR2R_POS))
        | ADC_CALIB_BIASREFBUF(((uint32_t*)SW0_ADDR) & ADCx_BIASREFBUF_Msk)>> ADCx_BIASREFBUF_POS );
    ...
}
```

plib_adcX.c

Table 9-5. NVM Software Calibration Area Mapping

Bit Position	Name	Description	
1:0	AC BIAS	AC Comparator 0/1 Bias Scaling. To be written to the AC CALIB register.	
4:2	ADC0 BIASCOMP	Bias comparator scaling. To be written to the ADC0 CALIB register.	ADC0_BIASCOMP
7:5	ADC0 BIASREFBUF	Bias reference buffer scaling. To be written to the ADC0 CALIB register.	ADC0_BIASREFBUF
10:8	ADC0 BIASR2R	Bias rail-to-rail amplifier scaling. To be written to the ADC0 CALIB register.	ADC0_BIASR2R
15:11	Reserved	Reserved	
18:16	ADC1 BIASCOMP	Bias comparator scaling. To be written to the ADC1 CALIB register.	ADC1_BIASCOMP
21:19	ADC1 BIASREFBUF	Bias reference buffer scaling. To be written to the ADC1 CALIB register.	ADC1_BIASREFBUF
24:22	ADC1 BIASR2R	Bias rail-to-rail amplifier scaling. To be written to the ADC1 CALIB register.	ADC1_BIASR2R
35:25	Reserved	Reserved	

Lab8 ADC Polling Software Trigger

Step 5

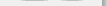
- Open **main.c** and finish **TODO 8.01** to **8.03** for output the ADC value to Tera Term.

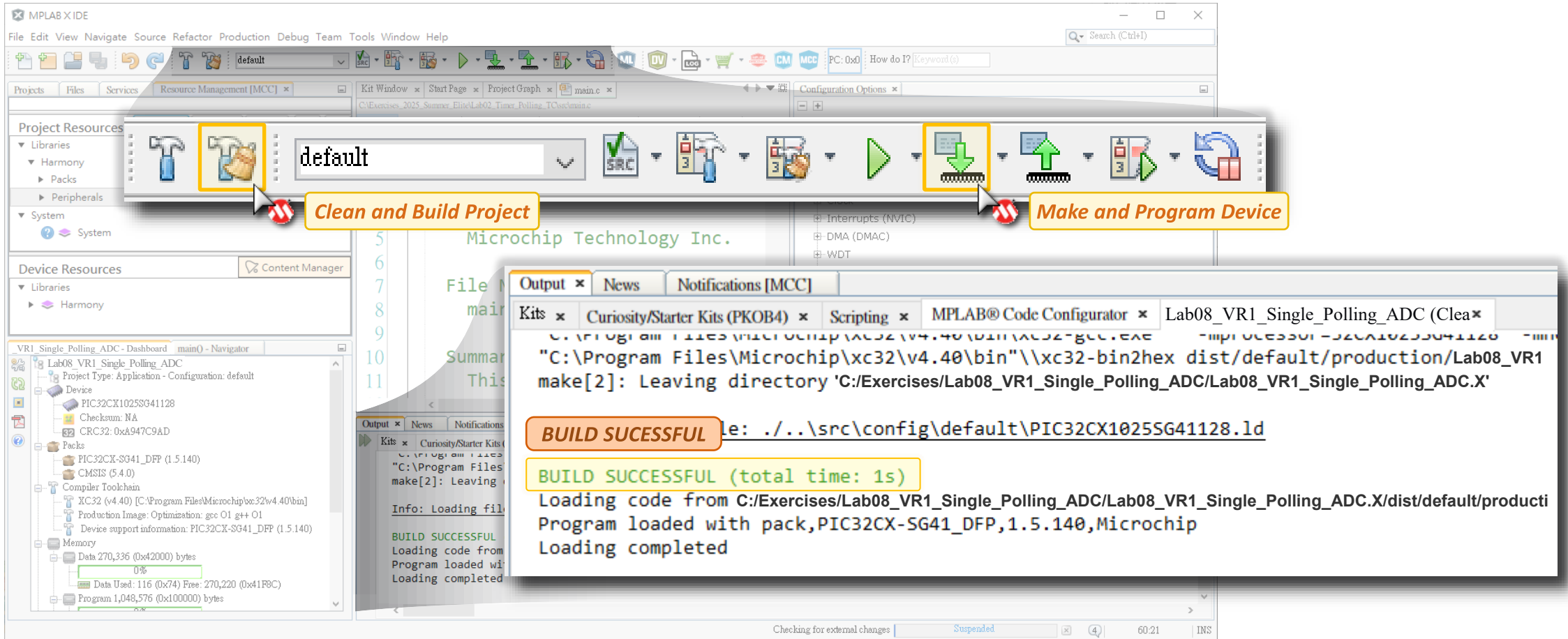
```
uint32_t i = 0;
...
// TODO 8.01
uint16_t ADC0_Result;
...
int main ( void )
{
    // TODO 8.02 Enable ADC0 module
    ADC0_Enable();
    ...
    while ( true )
    {
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks ( );

        if (TC0_HasExpired)
        {
            TC0_HasExpired = 0;
            ...
        }
        // TODO 8.03 Switch the input channel of ADC0 to Channel 7
        ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS ); ADC0 start sampling and conversion
        ADC0_ConversionStart();
        while( ADC0_ConversionStatusGet() != ADC_STATUS_RESRDY ){} Wait for the ADC0 conversion to complete
        ADC0_Result = ADC0_ConversionResultGet();
        myprintf( "@1,2;VR1 Value : %4d", ADC0_Result ); Get ADC0 conversion result and output to the terminal
    }
    ...
}
```

Lab8 ADC Polling Software Trigger

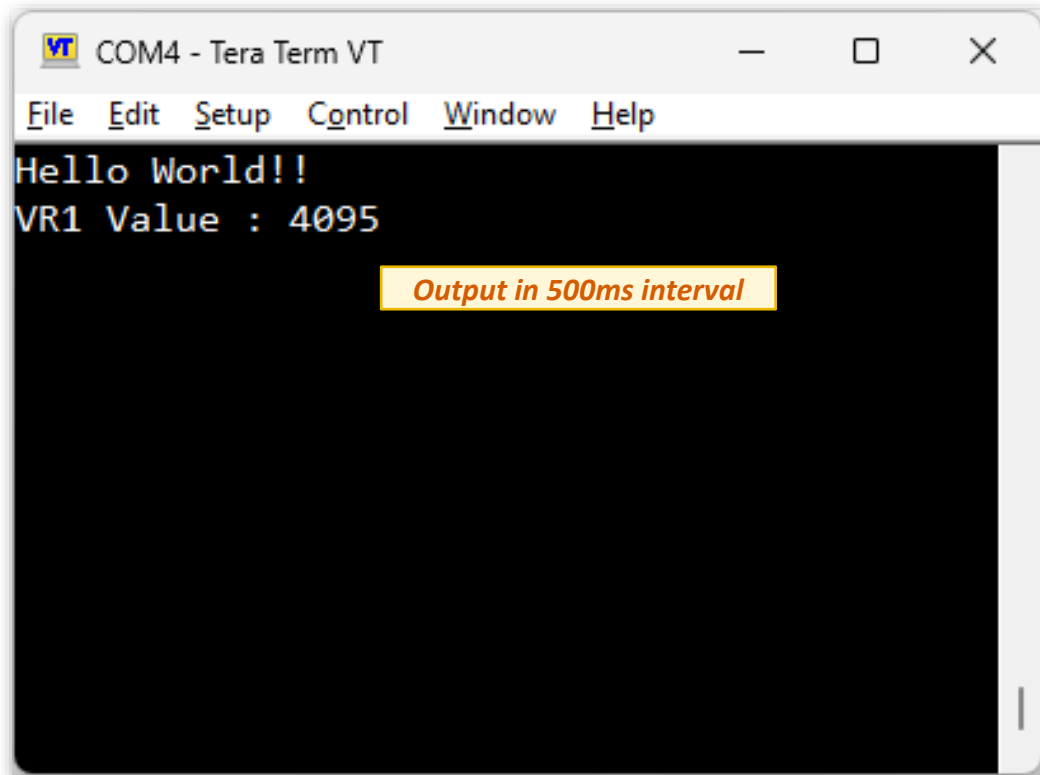
Step 6

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



Lab8 ADC Polling Software Trigger

Result

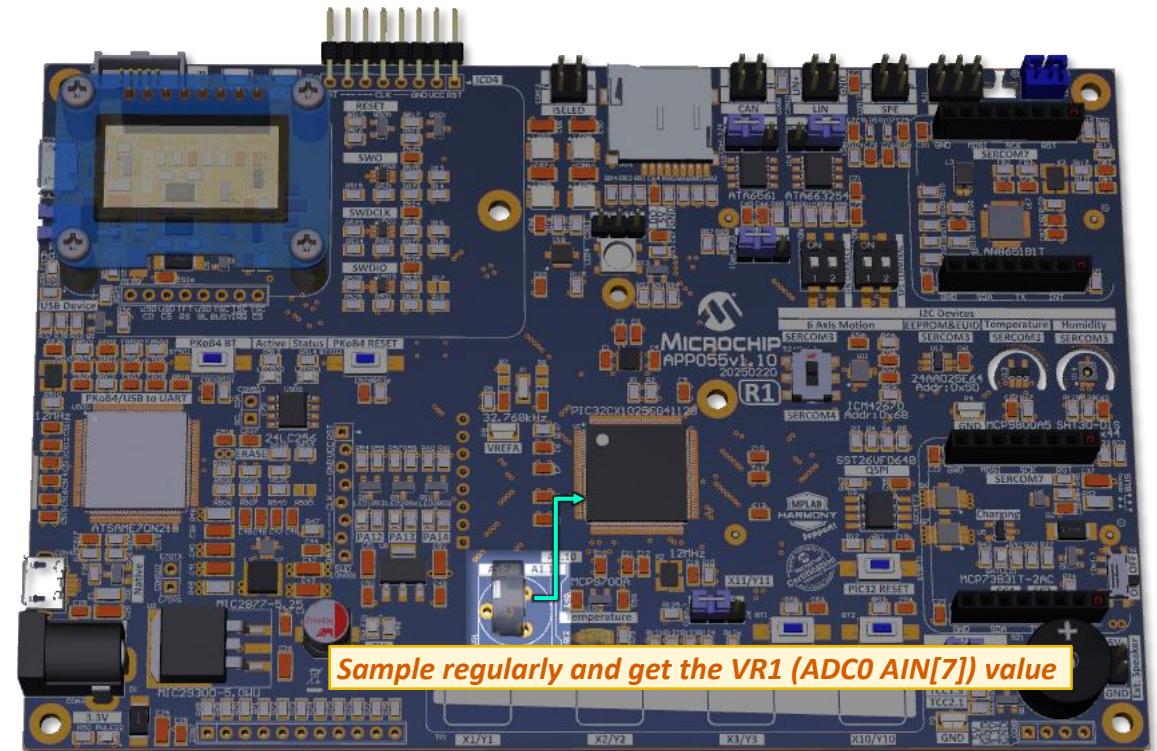


COM4 - Tera Term VT

File Edit Setup Control Window Help

```
Hello World!!  
VR1 Value : 4095
```

Output in 500ms interval

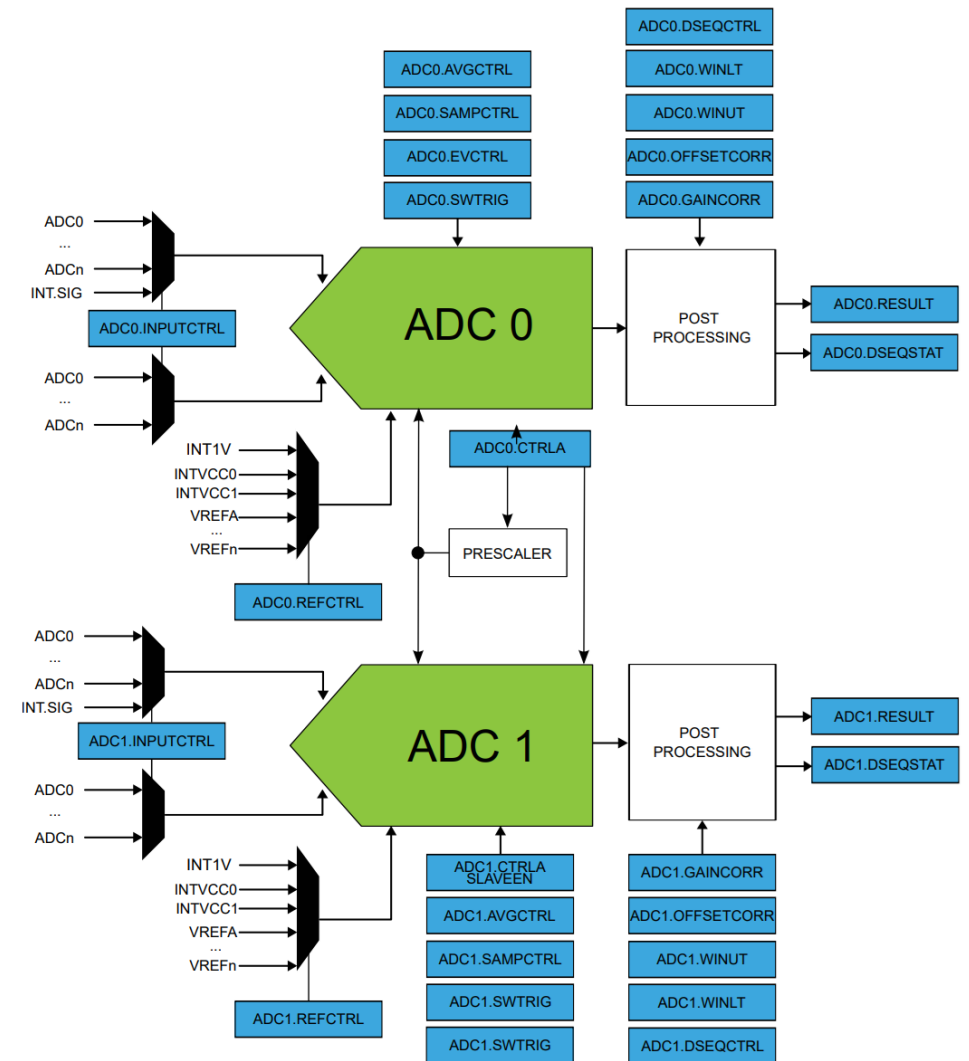


Dual ADC Instances

ADC0 and ADC1

Dual Core ADC

- The PIC32CX-SG has two ADC core, **ADC0** and **ADC1**.
- The two inputs can be **sampled simultaneously**, as each ADC includes a dedicated sample and hold circuit.
- Each ADC core has **independent register space**, allowing set ADC configuration separately.
- User can optionally set the partial configuration of two ADC core at the same time, and obtain the conversion values simultaneously or interleaved of two analog inputs after trigger ADC sampling once.



ADC0/ADC1 Configuration Example

Configuration Options Example

- Each ADC core has **independent register space**, allowing set ADC configuration separately.

The screenshot shows the 'Configuration Options' dialog box for ADC0. The 'ADC0' tab is selected. The configuration options are as follows:

- Select Prescaler: Peripheral clock divided by 8
- Select Sample Length (cycles): 4
- **** Conversion Time is 2.133333333333 us ****
- Select Reference: AVDD
- Select Conversion Trigger: SW Trigger
- Enable DMA Sequencing: ☐
- Channel Configuration
 - Select Positive Input: ADC AIN0 Pin
 - Select Negative Input: Internal Ground
- Result Configuration
 - Select Result Resolution: 12-bit result
 - Left Aligned Result: ☐
 - Enable Result Ready Interrupt: ☐
 - Enable Result Ready Event Out: ☐
- Window Mode Configuration
 - Select Window Monitor Mode: No window mode (default)
- Sleep Mode Configuration
 - Run During Standby: ☐
 - On Demand Control: ☐

The screenshot shows the 'Configuration Options' dialog box for ADC1. The 'ADC1' tab is selected. The configuration options are as follows:

- Enable Slave: ☐
- Select Prescaler: Peripheral clock divided by 8
- Select Sample Length (cycles): 4
- **** Conversion Time is 2.133333333333 uS ****
- Select Reference: AVDD
- Select Conversion Trigger: SW Trigger
- Enable DMA Sequencing: ☐
- Channel Configuration
 - Select Positive Input: ADC AIN0 Pin
 - Select Negative Input: Internal Ground
- Result Configuration
 - Select Result Resolution: 12-bit result
 - Left Aligned Result: ☐
 - Enable Result Ready Interrupt: ☐
- Window Mode Configuration
 - Select Window Monitor Mode: No window mode (default)
- Sleep Mode Configuration
 - Run During Standby: ☐
 - On Demand Control: ☐

Lab9 Dual ADC Core Polling Software Trigger

⚙️ Lab9 Dual ADC Core Polling Software Trigger

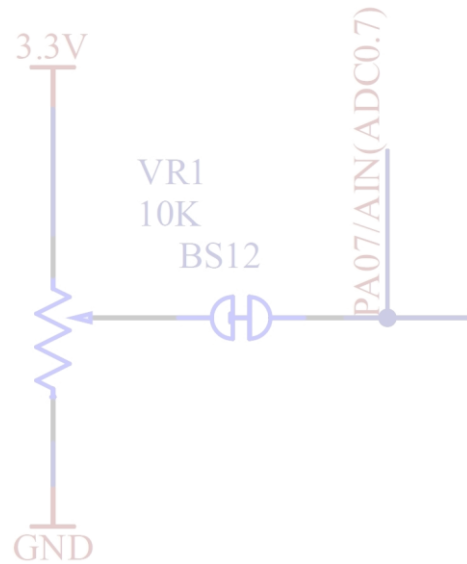
- Try to add **ADC-1** module to project.
- Initialize the ADC and get the conversion value from **Temperature Sensor (ADC1 AIN[10])** and **Light Sensor (ADC1 AIN[11])** periodically, and then send it to PC by using UART .
- ADC set to **Software Trigger, 12 bits resolution, Single-End** mode.

Schematic	Pin	Analog input
Potentiometer (VR1)	PA07	ADC0 CH.7
Temperature Sensor (MCP9700A)	PC00	ADC1 CH.10
Light Sensor (TEMT6000X01)	PC01	ADC1 CH.11

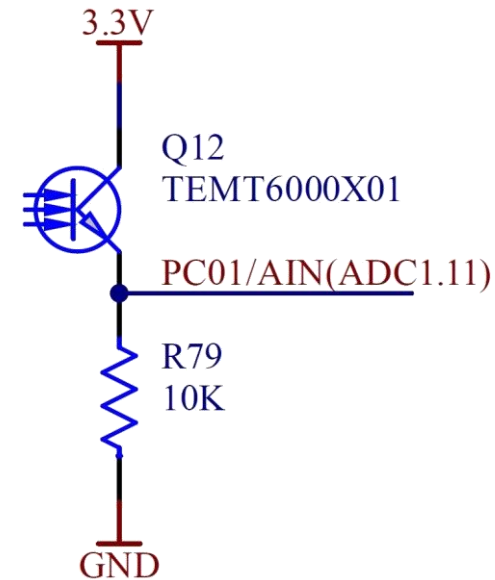
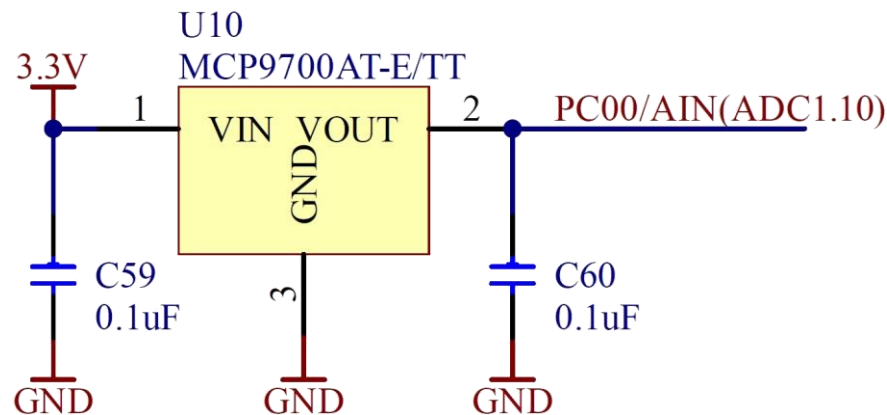
Let's go !

⚙️ Lab9 Dual ADC Core Polling Software Trigger

Check Schematic



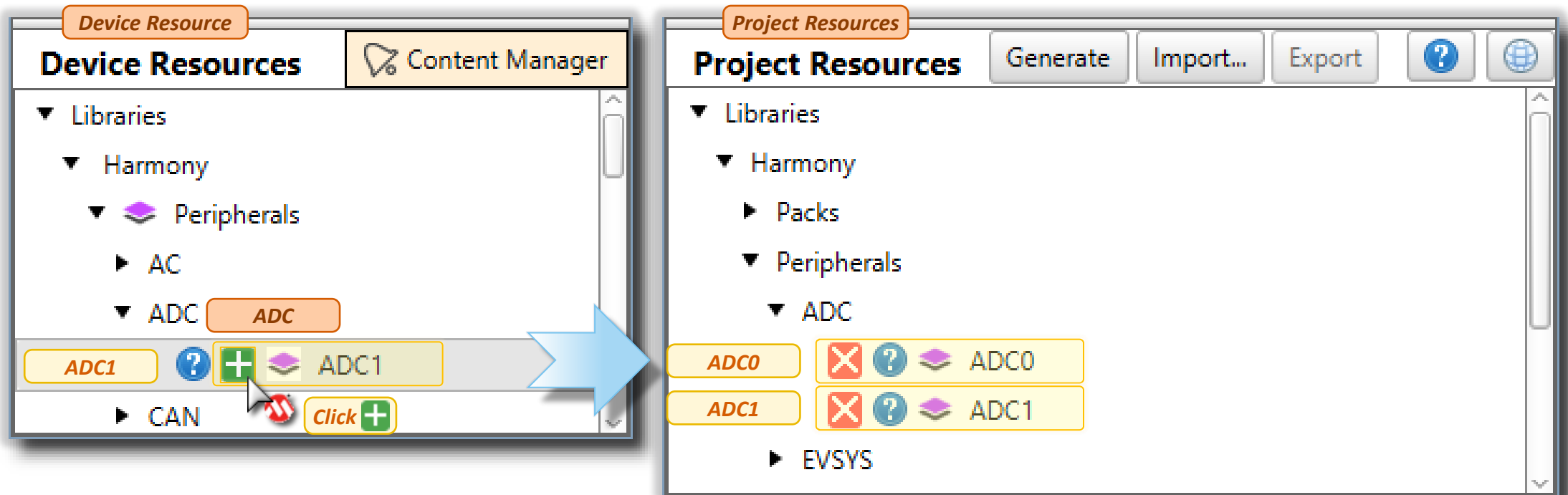
Schematic	Pin	Analog input
Potentiometer (VR1)	PA07	ADC0 CH.7
Temperature Sensor (MCP9700A)	PC00	ADC1 CH.10
Light Sensor (TEMT6000)	PC01	ADC1 CH.11



⚙️ Lab9 Dual ADC Core Polling Software Trigger

Step 1

- Find **ADC1** component in **Device Resources**, we use it for sample analog input pin.
- Click on **+** to add the **ADC1** module to your **Project Resources**.



** The setting values of Easy View Configuration and Configuration Options will be changed synchronously.*

⚙️ Lab9 Dual ADC Core Polling Software Trigger

Step 2 (Use Configuration Options)

The screenshot shows the 'Configuration Options' dialog for ADC1. The dialog is titled 'Configuration Options' and has a tab labeled 'ADC 1'. Below the tab, there is a tree view with 'ADC1' expanded. The settings are as follows:

- ADC1** (Note: * ADC1 has additional Enable Slave feature)
 - Enable Slave**: ☐
 - Select Prescaler**: Peripheral clock divided by 8 (Note: Prescaler : divided by 8)
 - Select Sample Length (cycles)**: 4 (Note: Sample Length : 4)
 - **** Conversion Time is 2.1333333333 us ****
 - Select Reference**: AVDD (Note: VREF : AVDD)
 - Select Conversion Trigger**: SW Trigger (Note: SW Trigger)
 - Enable DMA Sequencing**: ☐
 - Channel Configuration**
 - Select Positive Input**: ADC AIN10 Pin (Note: ADC AIN10 Pin)
 - Select Negative Input**: Internal Ground (Note: Internal Ground)
 - Result Configuration**
 - Select Result Resolution**: 12-bit result (Note: 12-bit result)
 - Left Aligned Result**: ☐
 - Enable Result Ready Interrupt**: ☐
 - Enable Result Ready Event Out**: ☐
 - Window Mode Configuration**: ☐
 - Sleep Mode Configuration**: ☐

* The setting values of Easy View Configuration and Configuration Options will be changed synchronously.

⚙️ Lab9 Dual ADC Core Polling Software Trigger

Step 3

- Pin Settings of the **Temperature Sensor (ADC1 AIN[10])** and **Light Sensor (ADC1 AIN[11])**.

Pin Settings

Project Graph x Pin Settings x Pin Diagram x Pin Table x

Order: Ports Ports Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
122	PB31		Available	Digital	High Impedance	Low	☐	☐	NORMAL
3	PC00		ADC1_AIN10	ADC1_AIN10	High Impedance	n/a	☐	☐	NORMAL
4	PC01		ADC1_AIN11	ADC1_AIN11	High Impedance	n/a	☐	☐	NORMAL
7	PC02		Available	Digital	High Impedance	Low	☐	☐	NORMAL

Schematic	Pin	Analog input
Potentiometer (VR1)	PA07	ADC0 CH.7
Temperature Sensor (MCP9700A)	PC00	ADC1 CH.10
Light Sensor (TEMT6000)	PC01	ADC1 CH.11

⚙️ Lab9 Dual ADC Core Polling Software Trigger

Step 4

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Generate' button highlighted with a red circle and a yellow 'Generate' label. The 'Output' window shows the results of the generation process, including a green box with the text 'Generation complete (total time: 5396 milliseconds)' and a list of events.

Output Window Content:

```
15:17:50.827 INFO: Generation complete.
15:17:50.843 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:50.844 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:50.844 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```


⚙️ Lab9 Dual ADC Core Polling Software Trigger

Step 5

- Open **main.c** and finish **TODO 9.01** to **9.02** for enable ADC-1 module.

```
uint32_t i = 0;
...
volatile uint8_t USART1_IsReceived = 0;
volatile uint8_t USART1_IsTransmitted = 1;
uint16_t ADC0_Result;
// TODO 9.01
uint16_t ADC1_Result[2];

void USART1_Transmit(uintptr_t context)
{
    USART1_IsTransmitted = 1;
}
...
int main ( void )
{
    ...
    myprintf( "#RST;#CLS;#HID;" );    // Reset Console, Clear screen and Hide cursor

    ADC0_Enable();
// TODO 9.02
    ADC1_Enable();

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

Two 16bits result buffers for ADC1 Channel 10 and 11

Enable ADC1 module

⚙️ Lab9 Dual ADC Core Polling Software Trigger

Step 6

- Open **main.c** and finish **TODO 9.03** to **9.04** for output the ADC value to Tera Term.

```
while ( true )
{
    ...
    if (TC0_HasExpired)
    {
        TC0_HasExpired = 0;
        LED1_Toggle();

        myprintf( "@1,1;Hello World!!" );

        ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS );
        ADC0_ConversionStart();
        while( ADC0_ConversionStatusGet() != ADC_STATUS_RESRDY ) {}
        ADC0_Result = ADC0_ConversionResultGet();
        myprintf( "@1,2;VR1 Value : %4d", ADC0_Result );

// TODO 9.03
        ADC1_ChannelSelect( ADC_POSINPUT_AIN10, ADC_INPUTCTRL_MUXNEG_AVSS );
        ADC1_ConversionStart();
        while( ADC1_ConversionStatusGet() != ADC_STATUS_RESRDY ) {}
        ADC1_Result[0] = ADC1_ConversionResultGet();
        myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );

// TODO 9.04
        ADC1_ChannelSelect( ADC_POSINPUT_AIN11, ADC_INPUTCTRL_MUXNEG_AVSS );
        ADC1_ConversionStart();
        while( ADC1_ConversionStatusGet() != ADC_STATUS_RESRDY ) {}
        ADC1_Result[1] = ADC1_ConversionResultGet();
        myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );
    }
}
```

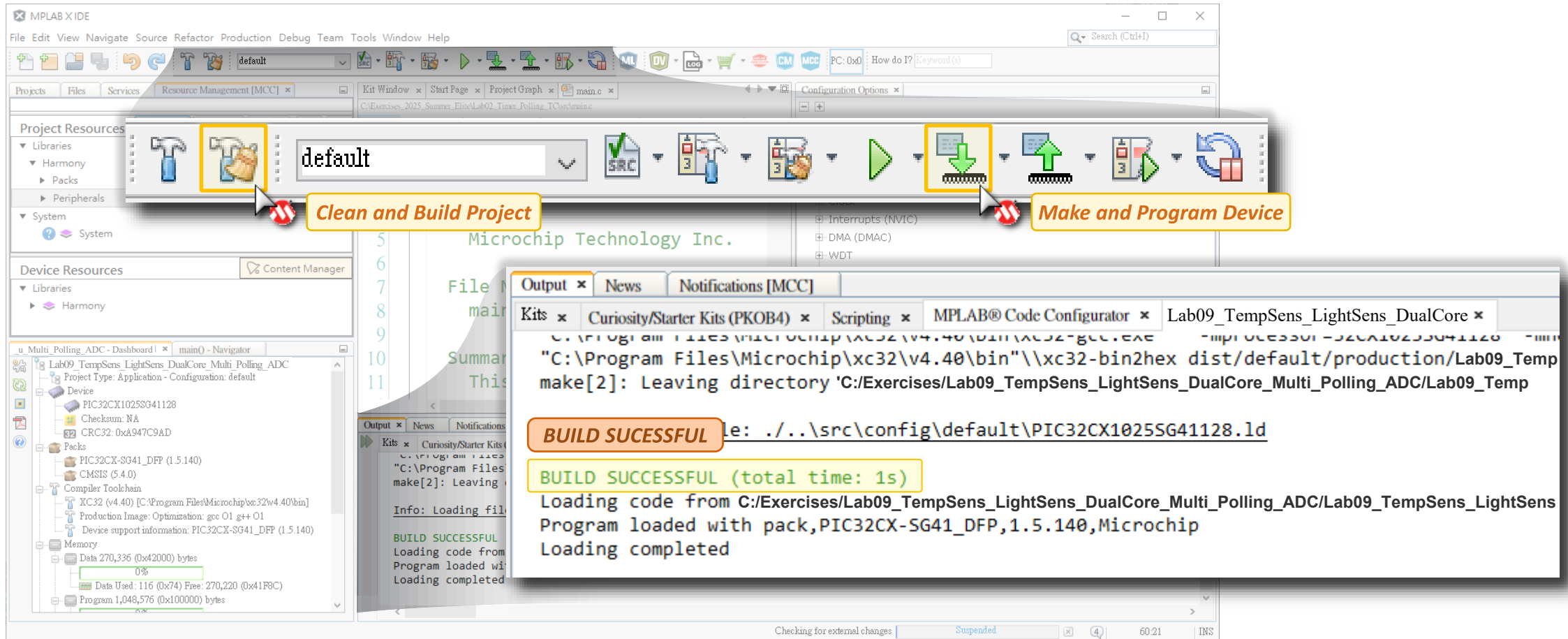
Switch the input channel of ADC1 to 10
(Temperature Sensor)

Switch the input channel of ADC1 to 11
(Light Sensor)

⚙️ Lab9 Dual ADC Core Polling Software Trigger

Step 7

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The main toolbar contains several icons. Two icons are highlighted with yellow boxes and red arrows pointing to them:

- The **Clean and Build Project** icon (a hammer and a wrench) is highlighted with a yellow box and a red arrow. A yellow callout box with the text "Clean and Build Project" points to it.
- The **Make and Program Device** icon (a green arrow pointing down to a chip) is highlighted with a yellow box and a red arrow. A yellow callout box with the text "Make and Program Device" points to it.

The IDE shows the project "Lab09_TempSens_LightSens_DualCore_Multi_Polling_ADC" with the device "PIC32CX1025SG41128" selected. The output window shows the following messages:

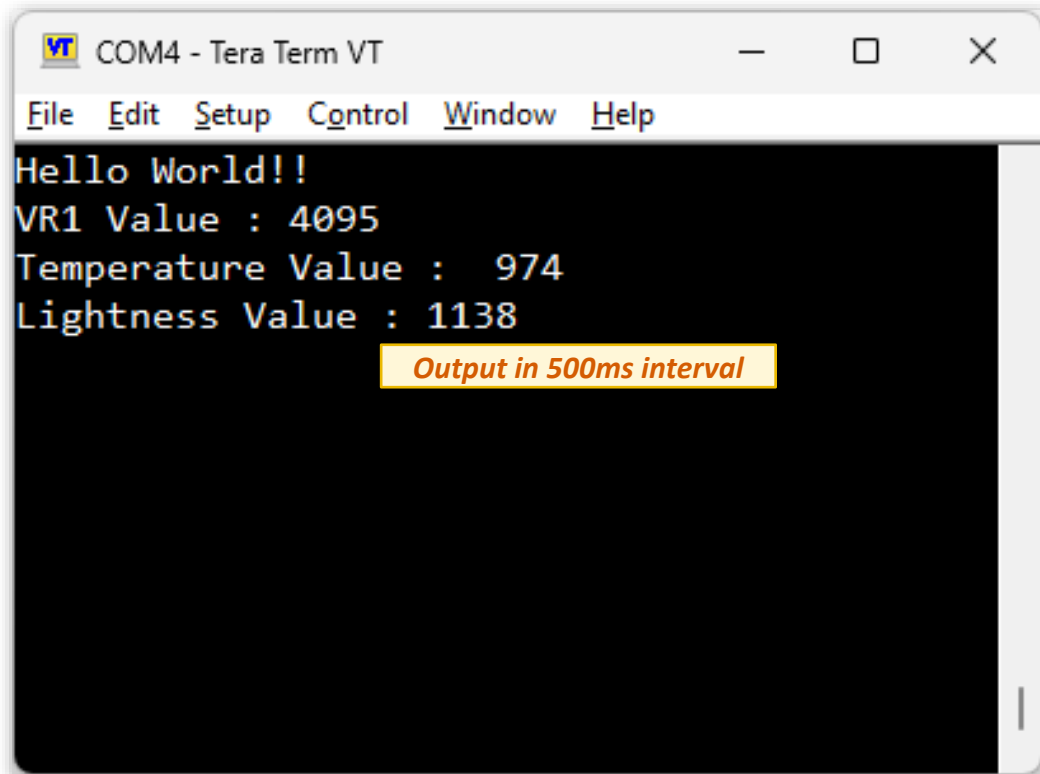
```
make[2]: Leaving directory 'C:/Exercises/Lab09_TempSens_LightSens_DualCore_Multi_Polling_ADC/Lab09_TempSens_LightSens_DualCore_Multi_Polling_ADC'

BUILD SUCCESSFUL le: ../../src/config/default/PIC32CX1025SG41128.ld

BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab09_TempSens_LightSens_DualCore_Multi_Polling_ADC/Lab09_TempSens_LightSens
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

⚙️ Lab9 Dual ADC Core Polling Software Trigger

Result

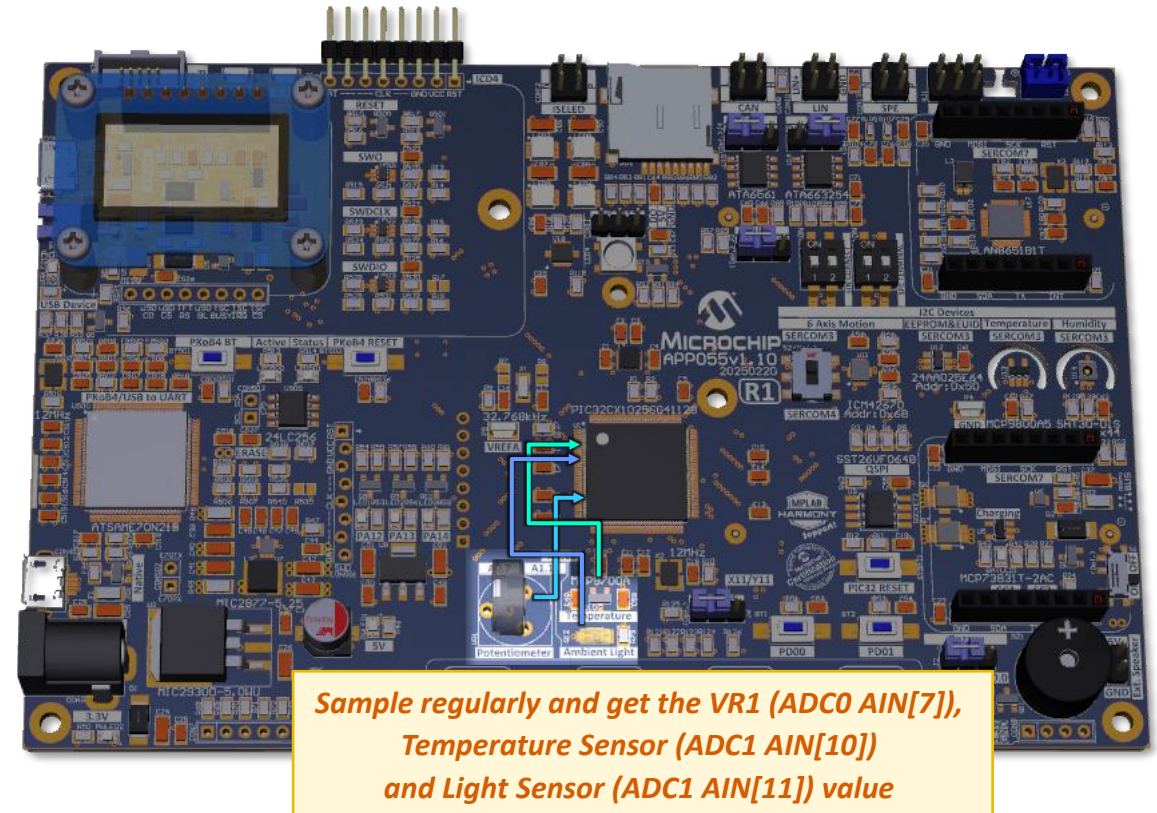


COM4 - Tera Term VT

File Edit Setup Control Window Help

```
Hello World!!  
VR1 Value : 4095  
Temperature Value : 974  
Lightness Value : 1138
```

Output in 500ms interval



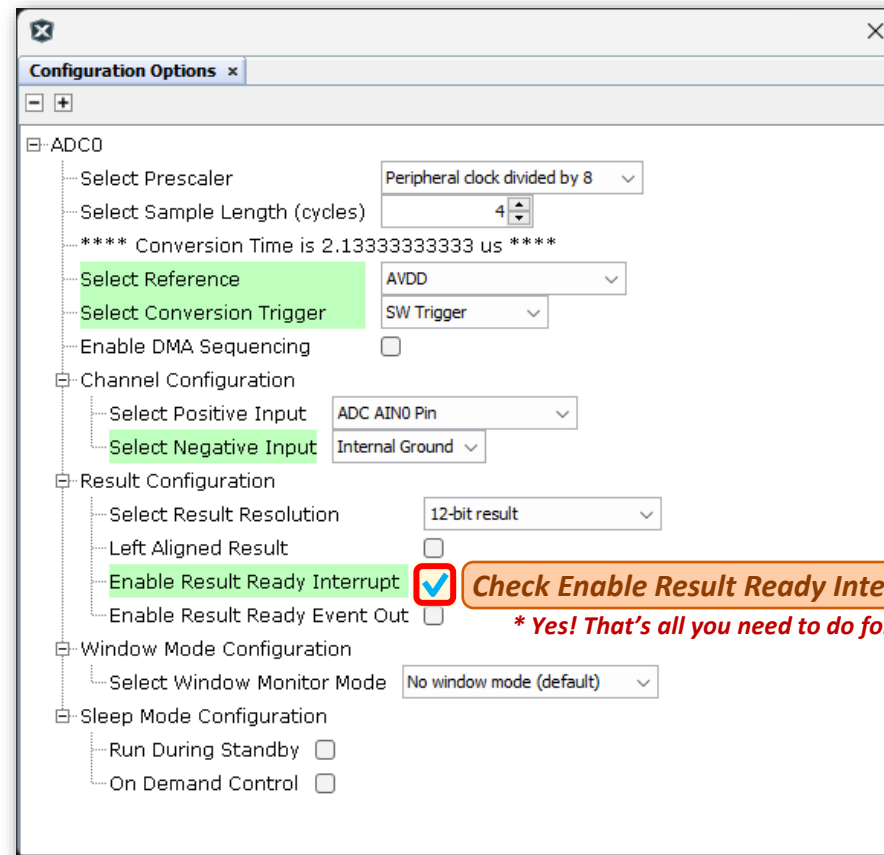
ADC Interrupt Method

ADC0 and ADC1

ADC Interrupt

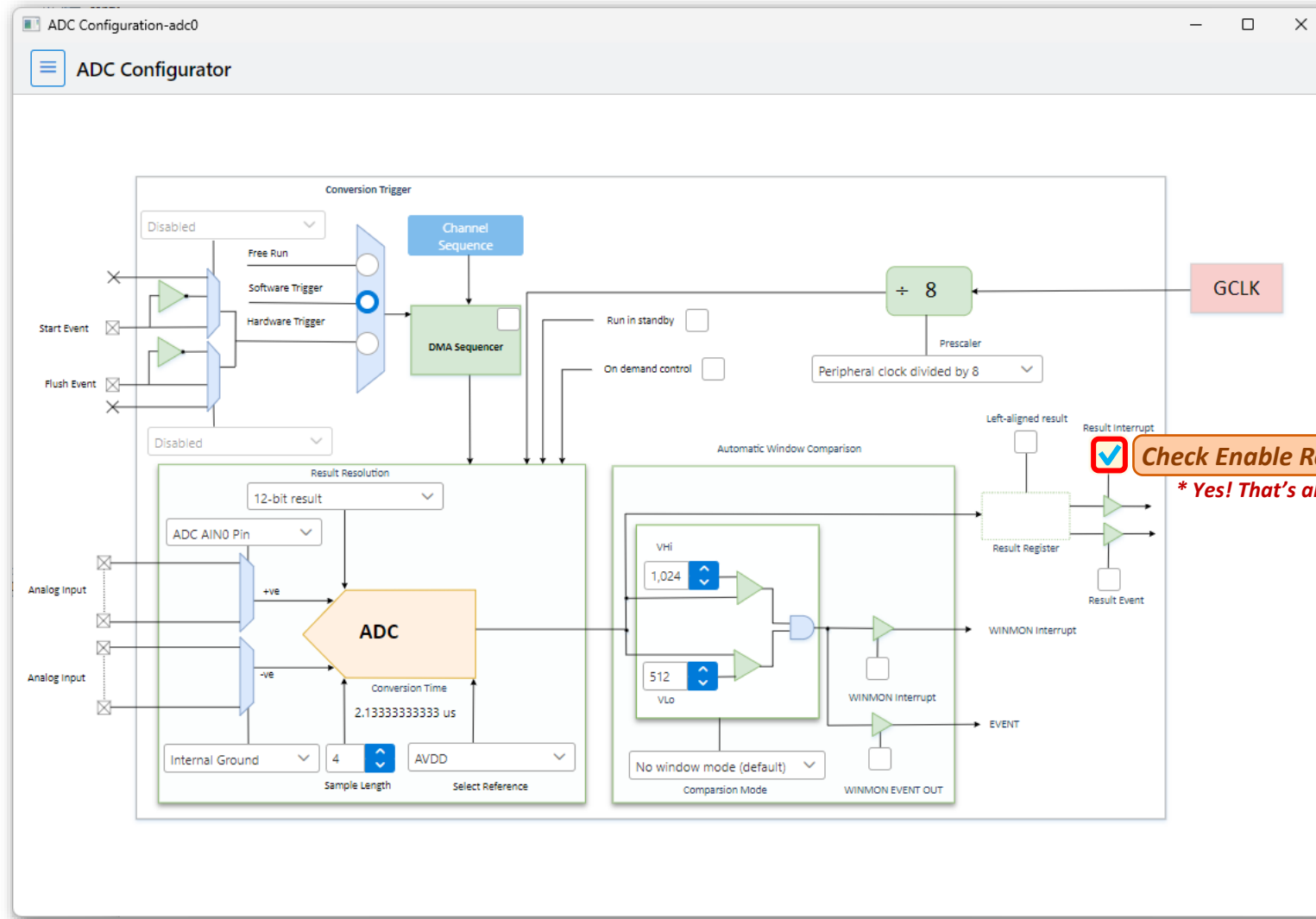
Configuration Options Example

- The ADC has the following interrupt sources :
 - RESRDY (Result Conversion Ready)
 - WINMON (Window Monitor)
 - OVERRUN



ADC Interrupt

Easy View Configuration Example



Check Enable Result Ready Interrupt

** Yes! That's all you need to do for enable Interrupt in MCC.*

Lab10 Dual ADC Core Interrupt Software Trigger

Lab10 Dual ADC Core Interrupt Software Trigger

- Change ADC polling to **Interrupt** mode.
- Get the ADC conversion value from VR1, Temperature Sensor and Light Sensor with one second period and then use UART to output ADC values to PC.

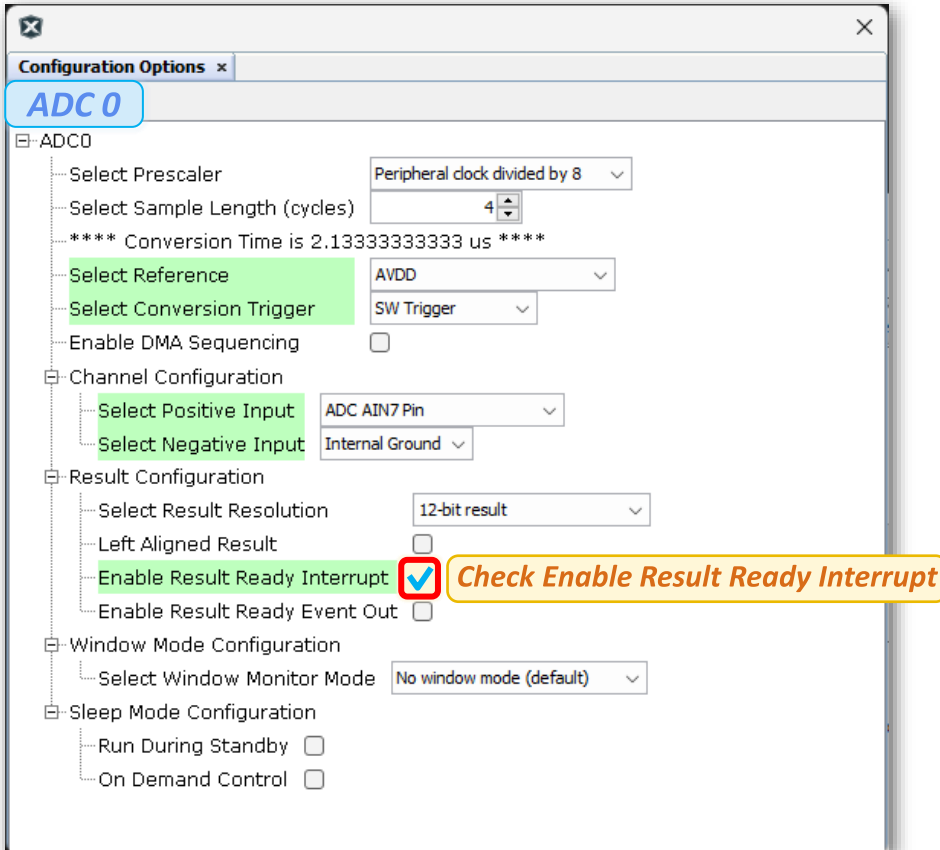
Schematic	Pin	Analog input
Potentiometer (VR1)	PA07	ADC0 CH.7
Temperature Sensor (MCP9700A)	PC00	ADC1 CH.10
Light Sensor (TEMT6000)	PC01	ADC1 CH.11

Let's go !

Lab10 Dual ADC Core Interrupt Software Trigger

Step 1

- Configure the ADC0 and ADC1 to Enable Result Ready Interrupt.



Configuration Options x

ADC 0

ADC0

Select Prescaler: Peripheral clock divided by 8

Select Sample Length (cycles): 4

**** Conversion Time is 2.13333333333 us ****

Select Reference: AVDD

Select Conversion Trigger: SW Trigger

Enable DMA Sequencing: ☐

Channel Configuration

Select Positive Input: ADC AIN7 Pin

Select Negative Input: Internal Ground

Result Configuration

Select Result Resolution: 12-bit result

Left Aligned Result: ☐

Enable Result Ready Interrupt: ☒ **Check Enable Result Ready Interrupt**

Enable Result Ready Event Out: ☐

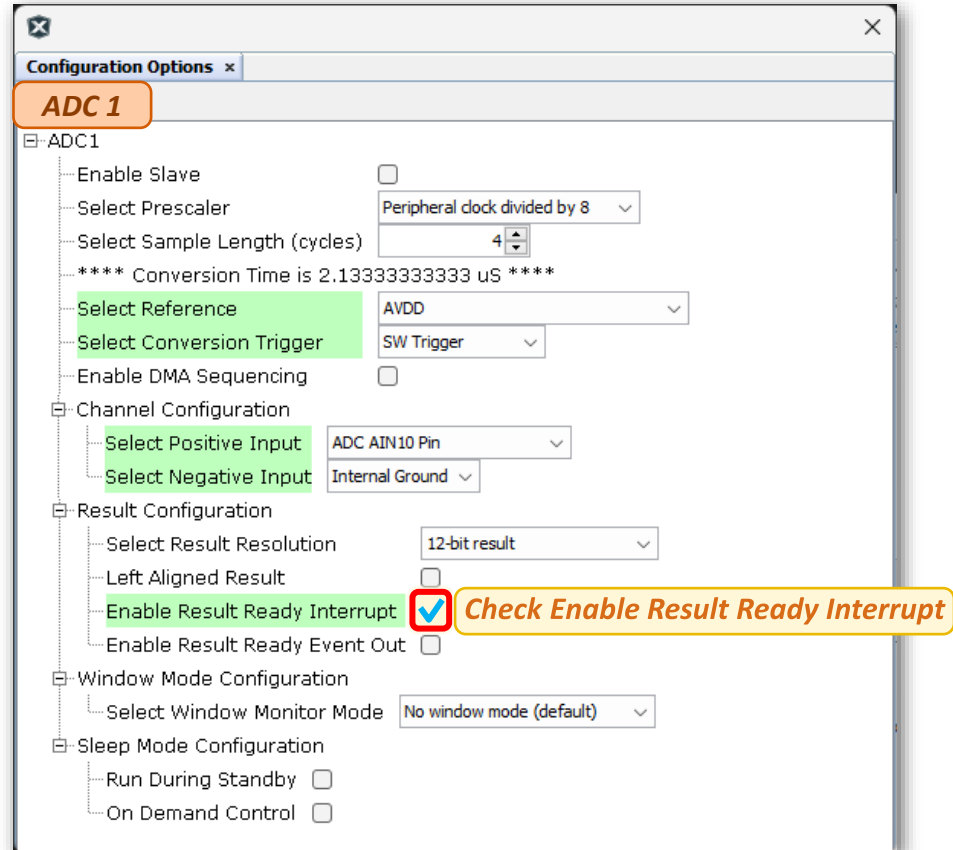
Window Mode Configuration

Select Window Monitor Mode: No window mode (default)

Sleep Mode Configuration

Run During Standby: ☐

On Demand Control: ☐



Configuration Options x

ADC 1

ADC1

Enable Slave: ☐

Select Prescaler: Peripheral clock divided by 8

Select Sample Length (cycles): 4

**** Conversion Time is 2.13333333333 us ****

Select Reference: AVDD

Select Conversion Trigger: SW Trigger

Enable DMA Sequencing: ☐

Channel Configuration

Select Positive Input: ADC AIN10 Pin

Select Negative Input: Internal Ground

Result Configuration

Select Result Resolution: 12-bit result

Left Aligned Result: ☐

Enable Result Ready Interrupt: ☒ **Check Enable Result Ready Interrupt**

Enable Result Ready Event Out: ☐

Window Mode Configuration

Select Window Monitor Mode: No window mode (default)

Sleep Mode Configuration

Run During Standby: ☐

On Demand Control: ☐

⚙️ Lab10 Dual ADC Core Interrupt Software Trigger

Step 2

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Generate' button highlighted with a red circle and a yellow 'Generate' label. The 'Project Resources' tree shows the 'System' resource selected. The 'Output' window shows the following log:

```
15:17:50.827 INFO: Generation complete.
15:17:50.843 INFO: ] Executing generate hook
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

ADC Interrupt

ADC_n Interface Functions

- **ADC_n_Initialize** : Initialize ADC_n module
- **ADC_n_Enable** : Enable ADC_n module
- **ADC_n_Disable** : Disable ADC_n module
- **ADC_n_ChannelSelect** : Configure channel input
- **ADC_n_ConversionStart** : Start the ADC_n conversion by Software
- ~~• **ADC_n_ConversionStatusGet** : Check whether result is ready~~ * Available only in ADC polling mode.
- **ADC_n_ConversionResultGet** : Read the conversion result
- **ADC_n_CallbackRegister** : Register callback function

** Available in ADC Interrupt mode.*

ADC Interrupt

ADC_n Interface Functions

void ADC_n_CallbackRegister (**ADC_CALLBACK** callback, **uintptr_t** context)

- **Description:**
 - Register callback function.
- **Parameter:**
 - callback : Pointer to the callback function to be called on adc event.
 - context : A user-defined value (typically a pointer) passed to the callback when invoked.

- **Example :**

```
uint16_t ADC_value;
void ADCn_Complete ( ADC_STATUS status, uintptr_t context ){
    if( status & ADC_STATUS_RESRDY ) ADC_Result = ADCn_ConversionResultGet();
}
...
ADCn_CallbackRegister( ADCn_Complete, (uintptr_t )NULL );
ADCn_Enable();
...
ADCn_ChannelSelect( ADC_POSINPUT_AINx, ADC_INPUTCTRL_MUXNEG_AVSS );
ADCn_ConversionStart();
```

⚙️ Lab10 Dual ADC Core Interrupt Software Trigger

Step 3

- Open **main.c** and finish **TODO 10.01** to comment out or remove the previous lab code.

```
while ( true )
{
    ...
    if (TC0_HasExpired)
    {
        TC0_HasExpired = 0;
        LED1_Toggle();

        myprintf( "@1,1;Hello World!!" );

// TODO 10.01  * These codes are suitable for ADC polling mode; We will modify to interrupt mode in the following steps.
        // ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS );
        // ADC0_ConversionStart();
        // while( ADC0_ConversionStatusGet() != ADC_STATUS_RESRDY ) {}
        // ADC0_Result = ADC0_ConversionResultGet();
        // myprintf( "@1,2;VR1 Value : %4d", ADC0_Result );

        // ADC1_ChannelSelect( ADC_POSINPUT_AIN10, ADC_INPUTCTRL_MUXNEG_AVSS );
        // ADC1_ConversionStart();
        // while( ADC1_ConversionStatusGet() != ADC_STATUS_RESRDY ) {}
        // ADC1_Result[0] = ADC1_ConversionResultGet();
        // myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );

        // ADC1_ChannelSelect( ADC_POSINPUT_AIN11, ADC_INPUTCTRL_MUXNEG_AVSS );
        // ADC1_ConversionStart();
        // while( ADC1_ConversionStatusGet() != ADC_STATUS_RESRDY ) {}
        // ADC1_Result[1] = ADC1_ConversionResultGet();
        // myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );

    }
    ...
}
```

⚙️ Lab10 Dual ADC Core Interrupt Software Trigger

Step 4

- Finish **TODO 10.02** for create variables and calculate the ADC1 Channel Counts.

```
...
uint32_t i = 0;
volatile uint8_t TC0_HasExpired = 0;
volatile uint8_t TC1_HasExpired = 0;
uint8_t USART1_TransmitData[100];
uint8_t USART1_ReceiveData[1];
volatile uint8_t USART1_IsReceived = 0;
volatile uint8_t USART1_IsTransmitted = 1;
uint16_t ADC0_Result;
uint16_t ADC1_Result[2];
// TODO 10.02
volatile uint8_t ADC0_IsCompleted = 0;
volatile uint8_t ADC1_IsCompleted = 0;
volatile uint8_t ADC1_ChannelIdx = 0;
volatile uint16_t ADC1_Channel[] = { ADC_POSINPUT_AIN10, ADC_POSINPUT_AIN11 };
volatile uint8_t ADC1_ChannelCount = sizeof(ADC1_Channel) / sizeof(ADC1_Channel[0]); * This exercise ADC1_ChannelCount is 2.

void USART1_Transmit(uintptr_t context)
{
    USART1_IsTransmitted = 1;
}

void USART1_Receive(uintptr_t context)
{
    USART1_IsReceived = 1;
}

void TC0_TimerExpired(TC_TIMER_STATUS status, uintptr_t context)
```

Assign ADC1 Channels

Calculate ADC1 Channel Counts

⚙️ Lab10 Dual ADC Core Interrupt Software Trigger

Step 5

- Finish **TODO 10.03** to **10.04** for ADC0 and ADC1 callback functions.

```
// TODO 10.02
volatile uint8_t ADC0_IsCompleted = 0;
volatile uint8_t ADC1_IsCompleted = 0;
volatile uint8_t ADC1_ChannelIdx = 0;
volatile uint16_t ADC1_Channel[] = { ADC_POSINPUT_AIN10, ADC_POSINPUT_AIN11 };
volatile uint8_t ADC1_ChannelCount = sizeof(ADC1_Channel) / sizeof(ADC1_Channel[0]);
```

```
// TODO 10.03
void ADC0_Complete( ADC_STATUS status, uintptr_t context )
{
```

```
    if( status & ADC_STATUS_RESRDY ) { Check for interrupt flags.
```

```
    {
        ADC0_Result = ADC0_ConversionResultGet();
        ADC0_IsCompleted = 1; Store the ADC0 conversion result.
```

```
    }
} Set the SW flag to know that the ADC0 conversion is completed
```

```
// TODO 10.04
void ADC1_Complete( ADC_STATUS status, uintptr_t context )
{
```

```
    if( status & ADC_STATUS_RESRDY ) { Check for interrupt flags.
```

```
    {
        ADC1_Result[ADC1_ChannelIdx] = ADC1_ConversionResultGet();
        ADC1_ChannelIdx = ( ADC1_ChannelIdx + 1 ) % ADC1_ChannelCount; * This exercise ADC1_ChannelCount is 2.
        ADC1_ChannelSelect( ADC1_Channel[ADC1_ChannelIdx], ADC_INPUTCTRL_MUXNEG_AVSS );
        if( ADC1_ChannelIdx != 0 ) ADC1_ConversionStart();
        else
            ADC1_IsCompleted = 1;
```

```
    } Continue sampling and converting the next until all channels are completed.
```

```
} Set the SW flag to know that the ADC1 all-channels conversion is completed.
```

ADC1_ChannelIdx will cycle sequentially from 0 to (ChannelCount - 1), result of AIN10 is stored in ADC1_Result[0] result of AIN11 is stored in ADC1_Result[1] When the all-channels results are obtained ADC1_IsCompleted will be set

⚙️ Lab10 Dual ADC Core Interrupt Software Trigger

Step 6

- Finish **TODO 10.05** to **10.06** for callback register and start ADC conversion.

```
int main (void)
{
    ...
    SERCOM1_USART_Read(USART1_ReceiveData, 1);

    myprintf( "#RST;#CLS;#HID;" );    // Reset Console, Clear screen and Hide cursor

    // TODO 10.05
    ADC0_CallbackRegister( ADC0_Complete, (uintptr_t) NULL );
    ADC1_CallbackRegister( ADC1_Complete, (uintptr_t) NULL );
    ADC0_Enable();
    ADC1_Enable();

    while( true )
    {
        ...
        if (TC0_HasExpired)
        {
            TC0_HasExpired = 0;
            LED1_Toggle();

            myprintf( "@1,1;Hello World!!" );
        }

        // TODO 10.06
        ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS );
        ADC0_ConversionStart();

        ADC1_ChannelIdx = 0;
        ADC1_ChannelSelect( ADC1_Channel[ADC1_ChannelIdx], ADC_INPUTCTRL_MUXNEG_AVSS );
        ADC1_ConversionStart();
    }
    ...
}
```

Register the Callback function for ADC0

Register the Callback function for ADC1.

ADC0 start sampling and conversion.

ADC1 start sampling and conversion.

⚙️ Lab10 Dual ADC Core Interrupt Software Trigger

Step 7

- Finish **TODO 10.07** to **10.08** for output the ADC values to Tera Term.

```
while( true )
{
    ...
    if (USART1_IsReceived)
    {
        USART1_IsReceived = 0;

        myprintf( "@1,5;Received Data : %1c", USART1_ReceiveData[0] );
        SERCOM1_USART_Read( USART1_ReceiveData, 1 );
    }

    // TODO 10.07
    if( ADC0_IsCompleted )
    {
        ADC0_IsCompleted = 0;
        myprintf( "@1,2;VR1 Value : %4d", ADC0_Result );
    }

    // TODO 10.08
    if( ADC1_IsCompleted )
    {
        ADC1_IsCompleted = 0;
        myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );
        myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );
    }

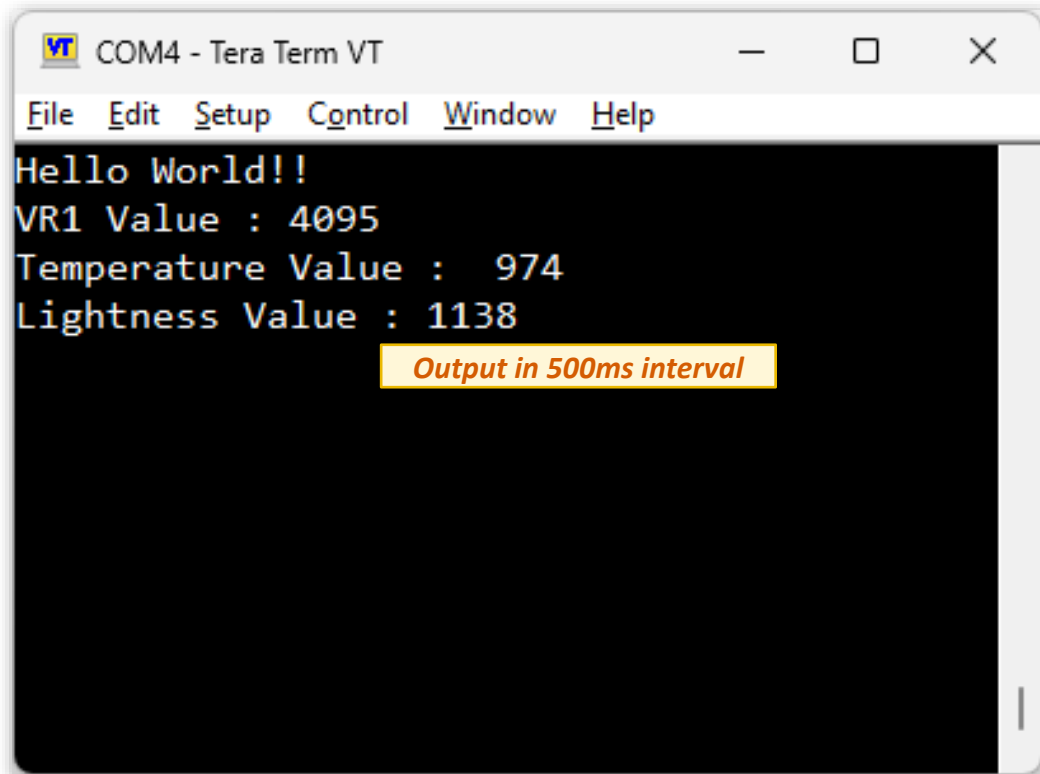
    if ( BT1_Get() ) LED3_Clear();
    else             LED3_Set();
}
```

Step 11

-
- The screenshot shows the MPLAB X IDE interface. The 'Project Resources' pane on the left shows the project structure. The 'Device Resources' pane shows the selected device, PIC32CX1025SG41128. The 'Output' window displays the build and programming process, including the command to build the project and the successful loading of the code onto the device.
- Clean and Build Project**
- Make and Program Device**
- BUILD SUCCESSFUL** le: ../../src/config/default/PIC32CX1025SG41128.ld
- BUILD SUCCESSFUL (total time: 1s)**
- Loading code from C:/Exercises/Lab10_DualCore_Interrupt_ADC/Lab10_DualCore_Interrupt_ADC.X/dist/default/productio
 Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
 Loading completed

⚙️ Lab10 Dual ADC Core Interrupt Software Trigger

Result

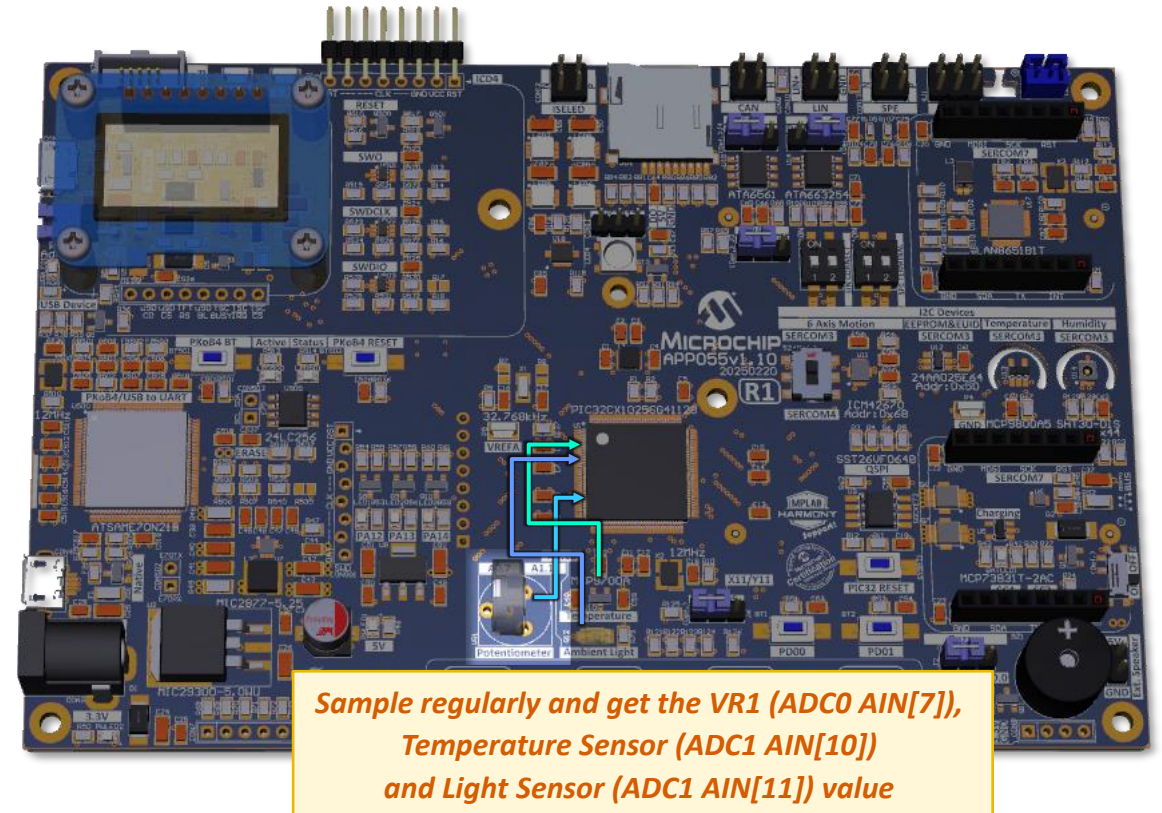


COM4 - Tera Term VT

File Edit Setup Control Window Help

```
Hello World!!  
VR1 Value : 4095  
Temperature Value : 974  
Lightness Value : 1138
```

Output in 500ms interval

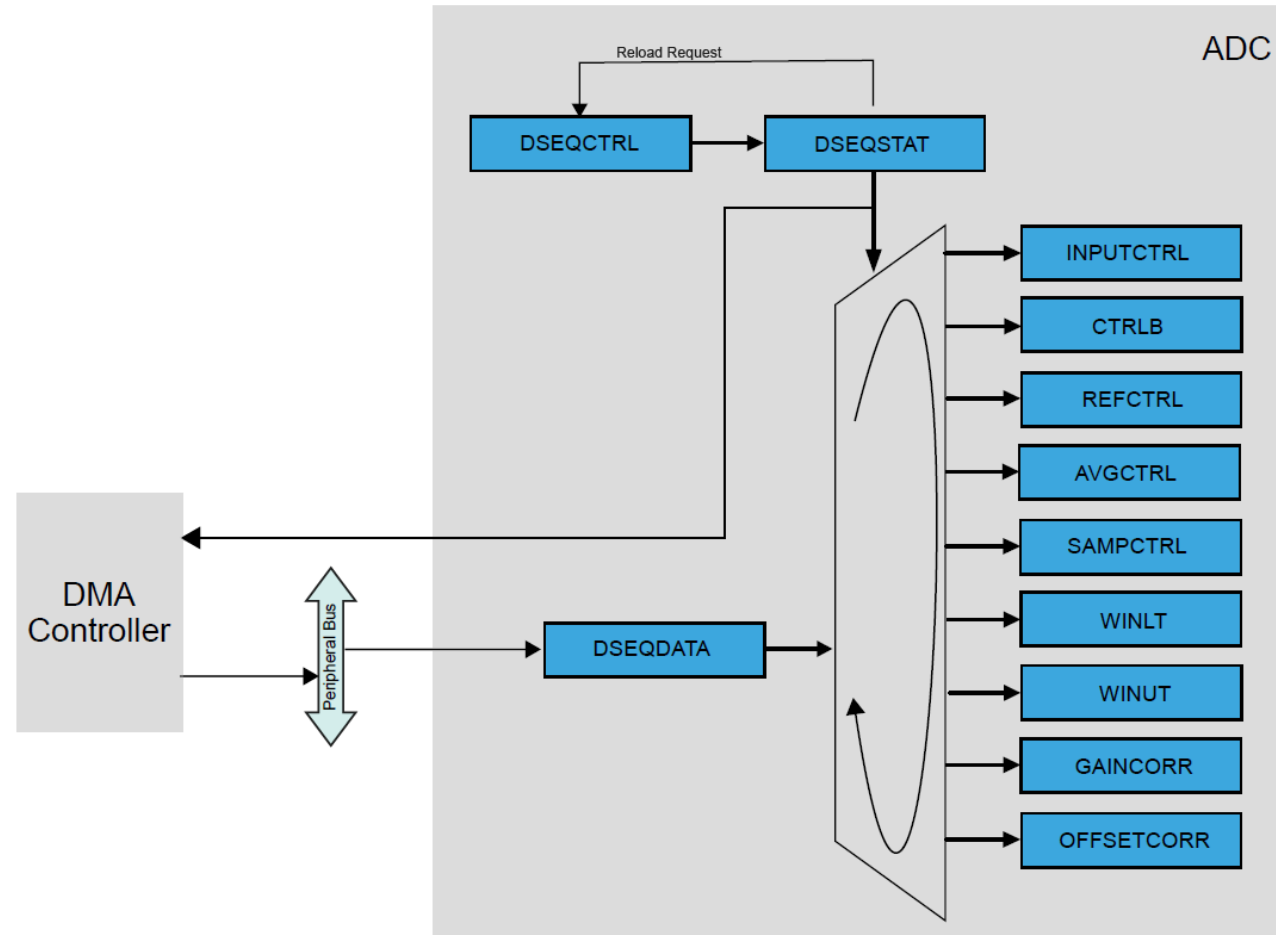


Appendix : ADC DMA Sequencing

Automatic Channel ADC Conversion by Sequence

ADC DMA Sequencing

- The ADC can sequence a series of conversion.
- When DMA sequencing is enabled, a set of ADC configuration registers can be automatically refreshed using the DMA controller.



ADC DMA Sequencing

- Enable the **DMA Sequencing** of ADC1 to automatic conversion both Channel 10 and Channel 11 by DMAC.
- Many of ADC registers could be auto apply by DMA automatically but we will use only the **Input Control Register** for the demonstration of Channel Scanning.

ADC 1

ADC1

Enable Slave ☐

Select Prescaler Peripheral clock divided by 8

Select Sample Length (cycles) 4

**** Conversion Time is 2.133333333333 uS ****

Select Reference AVDD

Select Conversion Trigger SW Trigger

Enable DMA Sequencing ☒ **Check Enable DMA Sequencing**

Enable Input Control ☒ **Check Enable Input Control**

Enable Control B ☐

Enable Reference Control ☐

Enable Average Control ☐

Enable Sampling Time Control ☐

Enable Window Monitor Lower Threshold ☐

Enable Window Monitor Upper Threshold ☐

Enable Gain Correction ☐

Enable Offset Correction ☐

Channel Configuration

Select Positive Input ADC AIN10 Pin

Select Negative Input Internal Ground

Result Configuration

Select Result Resolution 12-bit result

Left Aligned Result ☐

Enable Result Ready Interrupt ☒

Enable Result Ready Event Out ☐

Window Mode Configuration

Select Window Monitor Mode No window mode (default)

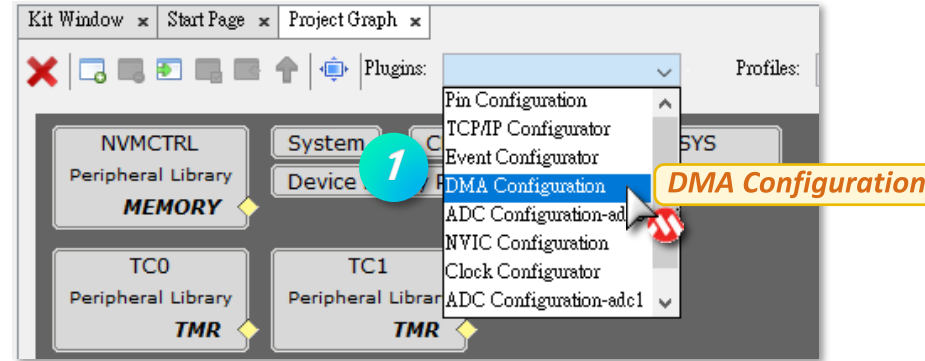
Sleep Mode Configuration

Run During Standby ☐

On Demand Control ☐

ADC DMA Sequencing

- Configure the DMA channel for the **DMA Sequencing** of ADC1.



DMA Configuration

☐ Use Linked List Mode

Channel Number	Trigger
DMAC Channel 0	ADC1_SEQ ADC1_SEQ

2 Add Channel Remove Selected Channel **Add Channel**

4 Source Address Mode **Increment Address After Every Transfer**

5 Destination Address Mode **Fixed Address**

6 Beat Size **32-bit bus transfer**

Burst Length SINGLE

FIFO Threshold 1BEAT

DMA Channel 0 Settings

Enable Interrupt ☒

Trigger Action One Beat Transfer per DMA Request

Increment Address After Every Transfer

Fixed Address Mode Fixed Address

32-bit bus transfer 32-bit bus transfer

SINGLE

1BEAT

ADC DMA Sequencing

- Define the Input Control Register of ADC1 for DMA transfer by sequence.

```
...
uint16_t ADC0_Result;
uint16_t ADC1_Result[2];
volatile uint8_t  ADC0_IsCompleted = 0;
volatile uint8_t  ADC1_IsCompleted = 0;
volatile uint8_t  ADC1_ChannelIdx = 0;
volatile uint16_t ADC1_Channel[] = { ADC_POSINPUT_AIN10, ADC_POSINPUT_AIN11 };
volatile uint8_t  ADC1_ChannelCount = sizeof(ADC1_Channel) / sizeof(ADC1_Channel[0]);

// TODO 10.1.01
volatile uint32_t ADC1_SEQREG[2] = {(ADC_POSINPUT_AIN10|ADC_NEGINPUT_AVSS), (ADC_POSINPUT_AIN11|ADC_NEGINPUT_AVSS)};

void ADC0_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_STATUS_RESRDY )
    {
        ADC0_Result = ADC0_ConversionResultGet();
        ADC0_IsCompleted = 1;
    }
}

void ADC1_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_STATUS_RESRDY )
    {
        ADC1_Result[ADC1_ChannelIdx] = ADC1_ConversionResultGet();
        ADC1_ChannelIdx = ( ADC1_ChannelIdx + 1 ) % ADC1_ChannelCount;
        ADC1_ChannelSelect( ADC1_Channel[ADC1_ChannelIdx], ADC_INPUTCTRL_MUXNEG_AVSS );
        if( ADC1_ChannelIdx != 0 ) ADC1_ConversionStart();
        else
            ADC1_IsCompleted = 1;
    }
}
```

ADC DMA Sequencing

- Remove the software Channel Select and Conversion Start functions in ADC1 callback.

```
...
uint16_t ADC0_Result;
uint16_t ADC1_Result[2];
volatile uint8_t ADC0_IsCompleted = 0;
volatile uint8_t ADC1_IsCompleted = 0;
volatile uint8_t ADC1_ChannelIdx = 0;
volatile uint16_t ADC1_Channel[] = { ADC_POSINPUT_AIN10, ADC_POSINPUT_AIN11 };
volatile uint8_t ADC1_ChannelCount = sizeof(ADC1_Channel) / sizeof(ADC1_Channel[0]);

// TODO 10.1.01
volatile uint32_t ADC1_SEQREG[2] = { (ADC_POSINPUT_AIN10|ADC_NEGINPUT_AVSS), (ADC_POSINPUT_AIN11|ADC_NEGINPUT_AVSS) };

void ADC0_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_STATUS_RESRDY )
    {
        ADC0_Result = ADC0_ConversionResultGet();
        ADC0_IsCompleted = 1;
    }
}

void ADC1_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_STATUS_RESRDY )
    {
        ADC1_Result[ADC1_ChannelIdx] = ADC1_ConversionResultGet();
        ADC1_ChannelIdx = ( ADC1_ChannelIdx + 1 ) % ADC1_ChannelCount;

// TODO 10.1.02
        if( ADC1_ChannelIdx == 0 ) ADC1_IsCompleted = 1;
    }
}
```

ADC DMA Sequencing

- Add DMA Channel Transfer before ADC1 Conversion Start to instead of Channel select.

```
...
int main ( void )
{
    SYS_Initialize ( NULL );
    ...
    ADC0_CallbackRegister( ADC0_Complete, (uintptr_t) NULL );
    ADC1_CallbackRegister( ADC1_Complete, (uintptr_t) NULL );
    ADC0_Enable();
    ADC1_Enable();

    while ( true )
    {
        SYS_Tasks ( );

        if (TC0_HasExpired)
        {
            TC0_HasExpired = 0;
            LED1_Toggle();

            myprintf( "@1,1;Hello World!!" );

            ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS );
            ADC0_ConversionStart();

            // TODO 10.1.03
            DMAC_ChannelTransfer( DMAC_CHANNEL_0,
                                  (const void *)ADC1_SEQREG, (const void *)&ADC1_REGS->ADC_DSEQDATA, sizeof(ADC1_SEQREG) );
            ADC1_ConversionStart();
        }

        if (TC1_HasExpired)
        {
```

ADC DMA Sequencing with DMA Conversion Result

- In the previous steps, we still used software index switching in the callback to store the conversion results. Now we will try to use an additional DMA channel to automatically transfer the ADC1 conversion results to the result buffer.

The screenshot shows the 'DMA Configuration' window. On the left, the 'Active Channels List' contains two entries: 'DMAC Channel 0' with trigger 'ADC1_SEQ' and 'DMAC Channel 1' with trigger 'ADC1_RESRDY'. A yellow callout 'ADC1_RESRDY' points to the trigger for Channel 1. Below the list are buttons for 'Add Channel' (labeled 1), 'Remove Selected Channel', and another 'Add Channel' (labeled 2). On the right, 'DMA Channel 1 Settings' are shown. A 'Use Linked List Mode' checkbox is at the top. Settings include: 'Enable Interrupt' (checked), 'Trigger Action' (One Beat Transfer per DMA Request), 'Source Address Mode' (Fixed Address Mode, with a yellow callout 'Fixed Address'), 'Destination Address Mode' (Increment Address After Every Transfer, with a yellow callout 'Increment Address After Every Transfer'), 'Beat Size' (16-bit bus transfer, with a yellow callout '16-bit bus transfer'), 'Burst Length' (SINGLE), and 'FIFO Threshold' (1BEAT). Numbered callouts 3, 4, and 5 point to 'Source Address Mode', 'Destination Address Mode', and 'Beat Size' respectively.

Channel Number	Trigger
DMAC Channel 0	ADC1_SEQ
DMAC Channel 1	ADC1_RESRDY

DMA Channel 1 Settings

- Use Linked List Mode: ☐
- Enable Interrupt: ☒
- Trigger Action: One Beat Transfer per DMA Request
- Source Address Mode: Fixed Address Mode (Fixed Address)
- Destination Address Mode: Increment Address After Every Transfer (Increment Address After Every Transfer)
- Beat Size: 16-bit bus transfer (16-bit bus transfer)
- Burst Length: SINGLE
- FIFO Threshold: 1BEAT

ADC DMA Sequencing with DMA Conversion Result

- Define the ADC1 DMA RESRDY finish flag and callback.

```
...
uint16_t ADC0_Result;
uint16_t ADC1_Result[2];
volatile uint8_t  ADC0_IsCompleted = 0;
volatile uint8_t  ADC1_IsCompleted = 0;
volatile uint8_t  ADC1_ChannelIdx = 0;
volatile uint16_t ADC1_Channel[] = { ADC_POSINPUT_AIN10, ADC_POSINPUT_AIN11 };
volatile uint8_t  ADC1_ChannelCount = sizeof(ADC1_Channel) / sizeof(ADC1_Channel[0]);
volatile uint32_t ADC1_SEQREG[2] = {(ADC_POSINPUT_AIN10|ADC_NEGINPUT_AVSS), (ADC_POSINPUT_AIN11|ADC_NEGINPUT_AVSS)};
// TODO 10.2.01
volatile bool     ADC1_DMA_RESRDY_Done = false;

void ADC1_DMA_RESRDY_Callback(DMAC_TRANSFER_EVENT event, uintptr_t contextHandle)
{
    if (event == DMAC_TRANSFER_EVENT_COMPLETE)
    {
        ADC1_DMA_RESRDY_Done = true;
    }
}

void ADC0_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_STATUS_RESRDY )
    {
        ADC0_Result = ADC0_ConversionResultGet();
        ADC0_IsCompleted = 1;
    }
}

void ADC1_Complete( ADC_STATUS status, uintptr_t context )
{
    if( status & ADC_STATUS_RESRDY )
```

ADC DMA Sequencing with DMA Conversion Result

- Remove ADC1 callback to instead of DMA Channel callback and add DMA Channel 1 Transfer then output.

```
...
int main ( void )
{
    ...
    ADC0_CallbackRegister( ADC0_Complete, (uintptr_t )NULL );
// TODO 10.2.02
    // ADC1_CallbackRegister( ADC1_Complete, (uintptr_t )NULL );
    DMAC_ChannelCallbackRegister( DMAC_CHANNEL_1, ADC1_DMA_RESRDY_Callback, 0 );
    ADC0_Enable();
    ADC1_Enable();

    while ( true )
    {
        ...

        if (TC0_HasExpired)
        {
            ...
            ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS );
            ADC0_ConversionStart();
// TODO 10.2.03
            DMAC_ChannelTransfer( DMAC_CHANNEL_1,
                                  (const void *)&ADC1_REGS->ADC_RESULT, (const void *)&ADC1_Result, sizeof(ADC1_Result) );
            DMAC_ChannelTransfer( DMAC_CHANNEL_0,
                                  (const void *)&ADC1_SEQREG, (const void *)&ADC1_REGS->ADC_DSEQDATA, sizeof(ADC1_SEQREG) );
            ADC1_ConversionStart();
        }

// TODO 10.2.04
        if( ADC1_DMA_RESRDY_Done )
        {
            ADC1_DMA_RESRDY_Done = false;
            myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );
            myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );
        }
    }
}
```

Appendix : ADC Host-Client Operation

Simultaneously or Interleaved Conversion on ADCs

ADC Host-Client Operation with DMA Sequencing

- ADC support the Host and Client operation of two ADC Cores, here we will provide the simple steps and important note about how to enable Host-Client mode of ADC.

ADC 1

ADC1

Enable Slave ☒ **Enable Slave**

Select Sample Length (cycles) 4

**** Conversion Time is 2.133333333333 uS ****

Select Reference AVDD

Enable DMA Sequencing ☒

Enable Input Control ☒

Enable Control B ☐

Enable Reference Control ☐

Enable Average Control ☐

Enable Sampling Time Control ☐

Enable Window Monitor Lower Threshold ☐

Enable Window Monitor Upper Threshold ☐

Enable Gain Correction ☐

Enable Offset Correction ☐

Channel Configuration

Select Positive Input ADC AIN10 Pin

Select Negative Input Internal Ground

Result Configuration

Select Result Resolution 12-bit result

Left Aligned Result ☐

Enable Result Ready Interrupt ☒

Enable Result Ready Event Out ☐

Window Mode Configuration

Select Window Monitor Mode No window mode (default)

Sleep Mode Configuration

Run During Standby ☐

On Demand Control ☐

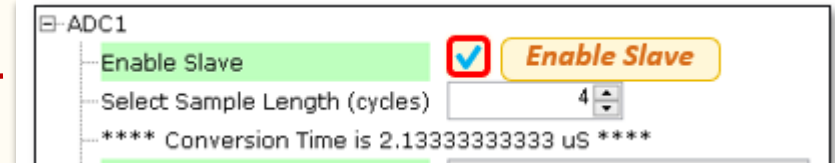
ADC Host-Client Operation with DMA Sequencing

- In Host-Client mode, ADC1 will trigger by ADC0 then you will not need ADC1 Enable and Conversion Start.

```
...
int main ( void )
{
    SYS_Initialize ( NULL );
    ...
    ADC0_CallbackRegister( ADC0_Complete, (uintptr_t) NULL );
    DMAC_ChannelCallbackRegister( DMAC_CHANNEL_1, ADC1_DMA_RESRDY_Callback, 0 );

```

ADC_DSEQCTRL bit been set to 1 in System Initialize.



```
// TODO 10.3.01
ADC1_REGS->ADC_DSEQCTRL &= ~ADC_DSEQCTRL_INPUTCTRL_Msk;

```

```
ADC0_Enable();
```

Clean ADC_DSEQCTRL bit before ADC0 Enable

```
// TODO 10.3.02
ADC1_REGS->ADC_DSEQCTRL |= ADC_DSEQCTRL_INPUTCTRL_Msk;

```

```
// TODO 10.3.03
// ADC1_Enable();

```

Set ADC_DSEQCTRL bit after ADC0 Enable

```
while ( true )
{
    ...
    if (TC0_HasExpired)
    {
        ...
        ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS );
        DMAC_ChannelTransfer( DMAC_CHANNEL_1,
                               (const void *)&ADC1_REGS->ADC_RESULT, (const void *)ADC1_Result, sizeof(ADC1_Result) );
        DMAC_ChannelTransfer( DMAC_CHANNEL_0,
                               (const void *)&ADC1_SEQREG, (const void *)&ADC1_REGS->ADC_DSEQDATA, sizeof(ADC1_SEQREG) );
    }
}

```

```
// TODO 10.3.04
ADC0_ConversionStart();
// ADC1_ConversionStart();

```

ADC0 will trigger both ADC0 and ADC1 to Start Conversion, then you need put the ADC0_ConversionStart after the DMA configuration of ADC1

In this mode of operation, the Client ADC1 is enabled by accessing the CTRLA register of the Host ADC0. In the same way, the Host ADC event inputs will be automatically routed to the Client ADC, meaning that the input events configuration must be done in the Host ADC (ADC0.EVCTRL).

Note: To enable DMA sequencing for the ADC1 (Client), the ADC0 (Host) must be enabled first (ADC0.CTRLA.ENABLE = 1) before DMA sequencing is enabled (ADC1.DSEQCTRL != 0).

ADC measurements can either start simultaneously on both ADCs, or be interleaved. The trigger mode selection is available in the Host ADC Control A register (ADC0.CTRLA.DUALSEL).

Note: The interleaved sampling is only usable in single conversion mode (ADC.CTRLB.FREERUN = 0).

ADC Host-Client Operation with DMA Sequencing Interleave

- In Host-Client mode, ADC measurements can either start **simultaneously** on both ADCs or be **interleaved**.

```
...
int main ( void )
{
    SYS_Initialize ( NULL );
    ...
    ADC0_CallbackRegister( ADC0_Complete, (uintptr_t) NULL );
    DMAC_ChannelCallbackRegister( DMAC_CHANNEL_1, ADC1_DMA_RESRDY_Callback, 0 );

    ADC1_REGS->ADC_DSEQCTRL &= ~ADC_DSEQCTRL_INPUTCTRL_Msk;

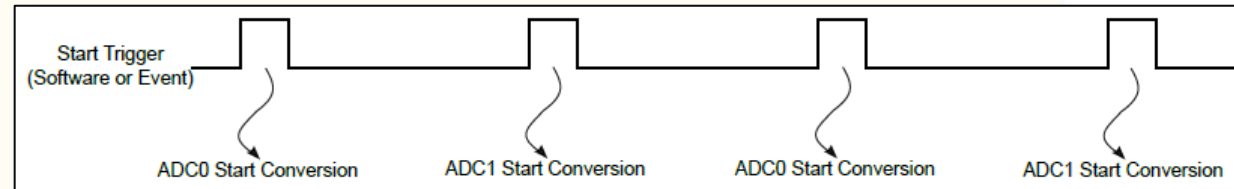
    // TODO 10.4.01
    ADC0_REGS->ADC_CTRLA |= ADC_CTRLA_DUALSEL_INTERLEAVE;

    ADC0_Enable();

    ADC1_REGS->ADC_DSEQCTRL |= ADC_DSEQCTRL_INPUTCTRL_Msk;

    while ( true )
    {
        ...
        if (TC0_HasExpired)
        {
            ...
            ADC0_ChannelSelect( ADC_POSINPUT_AIN7, ADC_INPUTCTRL_MUXNEG_AVSS );
            DMAC_ChannelTransfer( DMAC_CHANNEL_1,
                                (const void *)&ADC1_REGS->ADC_RESULT, (const void *)ADC1_Result, sizeof(ADC1_Result) );
            DMAC_ChannelTransfer( DMAC_CHANNEL_0,
                                (const void *)&ADC1_SEQREG, (const void *)&ADC1_REGS->ADC_DSEQDATA, sizeof(ADC1_SEQREG) );
            ADC0_ConversionStart();
        }

        if( ADC1_DMA_RESRDY_Done )
        {
            ADC1_DMA_RESRDY_Done = false;
        }
    }
}
```



Bits 4:3 - DUALSEL[1:0] Dual Mode Trigger Selection

These bits define the trigger mode, as shown in Table below. These bits are available in the Host ADC and have no effect if the Host/Client operation is disabled (ADC1.CTRLA.SLAVEEN = 0).

Value	Name	Description
0x0	BOTH	Start event or software trigger will start a conversion on both ADCs
0x1	INTERLEAVE	START event or software trigger will alternately start a conversion on ADC0 and ADC1. Note: The interleaved sampling is only usable in single conversion mode (ADC.CTRLB.FREERUN = 0).

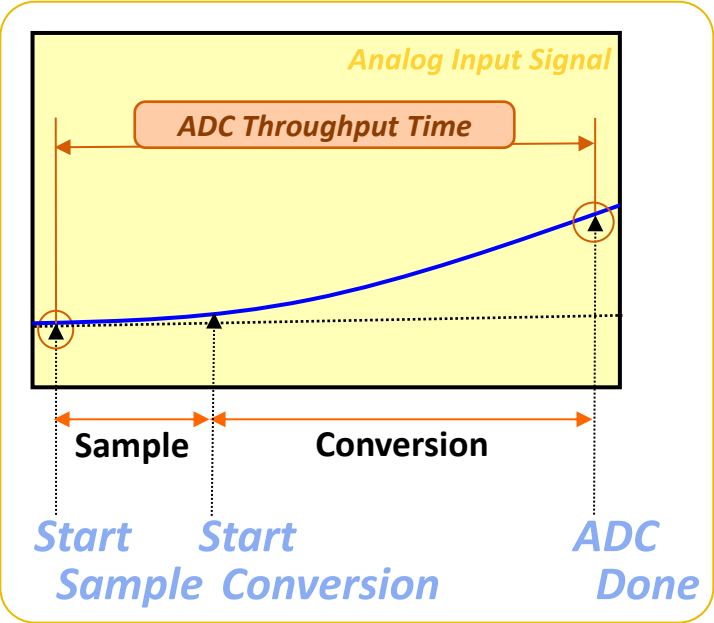
Appendix : ADC Throughput Rate

Sample Time and Conversion Time

ADC Throughput Rate

- The maximum ADC throughput rate depends on the **sample time** and **conversion time**, lower input impedance and resolution can improve the throughput rate.
- When connecting to a typical input impedance source, the maximum ADC throughput rate is **1M samples/second** by 12-bit resolution.

** 1Mps = 16MHz / (4+12)
* 4 T_{AD} and 12 T_{AD} (12-bit resolution)*

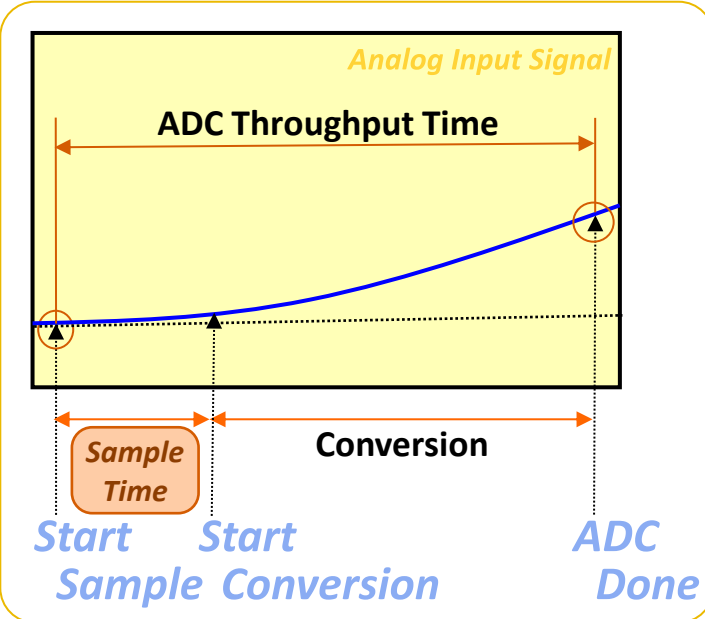


ADC Throughput Rate Requirements

Symbol	Characteristics	Min.	Typ.	Max.	Units	Conditions
F _{TPR} (Single-Ended Mode)	Throughput Rate (Single-Ended)	10	-	1000	ksps	12-bit resolution, R _{source} ≤ 6,560 Ω SAMPCTRL.OFFCOMP=1, * T _{AD} = 4 SAMPCTRL.SAMPLEN=n/a
F _{TPR} (Differential Mode)	Throughput Rate (Differential Mode)	10	-	1000	ksps	12-bit resolution, R _{source} ≤ 6,560 Ω SAMPCTRL.OFFCOMP=1, * T _{AD} = 4 SAMPCTRL.SAMPLEN=n/a

ADC Sample Time Requirement

- The minimum ADC Sample Time depends on the resolution and impedance of the input source, lower input impedance and resolution can improve the sample rate.
- When connecting to a typical input impedance source, the minimum ADC Sample Time is $4 T_{AD}$ by 12-bit resolution.

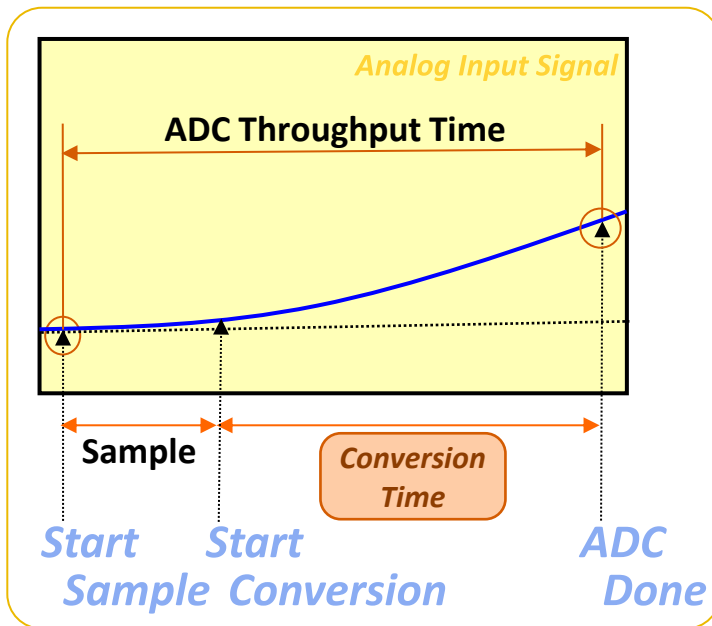


ADC Sample Electrical Requirements

Symbol	Characteristics	Min.	Typ.	Max.	Units	Conditions
T_{SAMP}	ADC Sample Time	4	-	-	T_{AD}	TAD(min) of 12-bit resolution, Ext Analog Input Rsource $\leq 6,560 \Omega$

ADC Conversion Time Requirement

- The Conversion Time depends on the resolution, refer to the ADC Sample Timing Requirements table in datasheet to get the different conversion time for 8-bit, 10-bit, and 12-bit resolution.
- When the ADC in 12-bit resolution, required conversion time is **12 T_{AD}** .



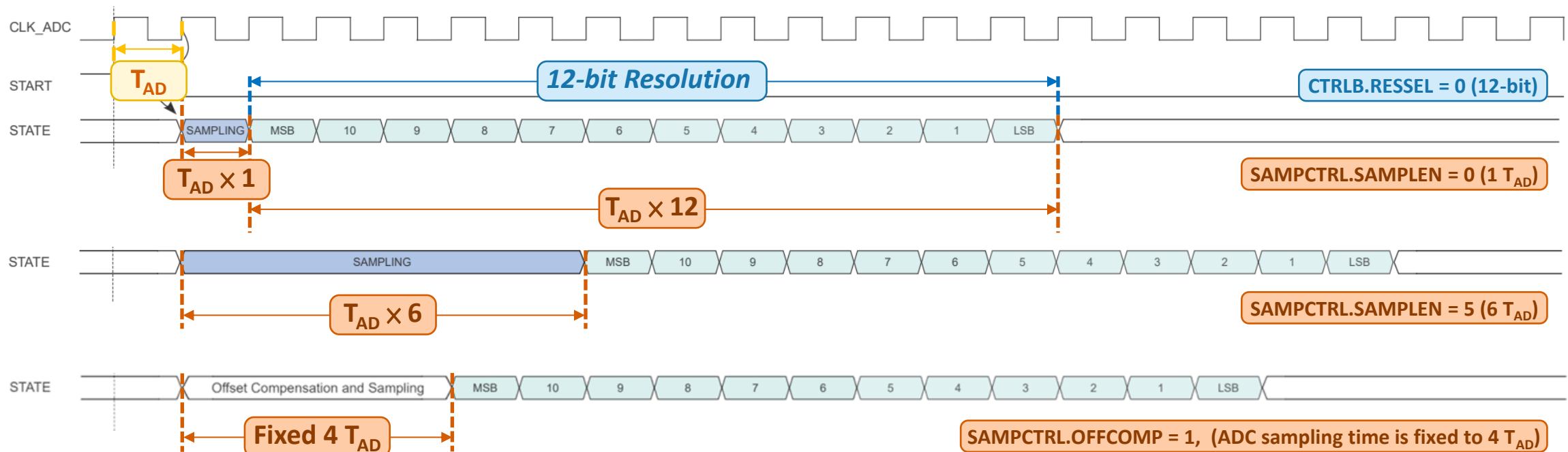
ADC Sample Electrical Requirements

Symbol	Characteristics	Min.	Typ.	Max.	Units	Conditions
T_{CNV}	Conversion Time (Single-Ended)	-	12	-	T_{AD}	12-bit resolution
		-	11	-		10-bit resolution
		-	9	-		8-bit resolution
	Conversion Time (Differential Mode)	-	12	-	T_{AD}	12-bit resolution
		-	11	-		10-bit resolution
		-	8	-		8-bit resolution

Register of ADC Sample Time and Conversion Time

Discuss how to set the register of sample time and conversion time

- The ADC Sample Time Length can be set on **SAMPLEN** or **OFFCOMP** register, depending on the setting method, it can be **$(\text{SAMPLEN}+1) \times T_{AD}$** or **fixed to $4 T_{AD}$** .
- The ADC Conversion Time depends on the resolution, can be set on **RESSEL** register, when the ADC in 12-bit resolution fixed to **$12 T_{AD}$** .



ADC Sample Time Calculation

Discuss how to calculate ADC Sample Time

- In PIC32CX-SG datasheet, we can get the specifications and calculation formulas of ADC and calculate the ADC Sample Time.

Formula

$$T_{\text{SAMP}}(\text{minimum, 12-bit}) = ((R_{\text{SAMPLE}} + R_{\text{SOURCE}}) * C_{\text{SAMPLE}} * (\# \text{BitsResolution} + 2) * \ln(2)) / T_{\text{AD}} + 1$$

Specification

#BitsResolution = 12 (maximum, 12-bit)
 $R_{\text{SOURCE}} = 6560 \, \Omega$ (typical, $R_{\text{SOURCE}} \leq 6560 \, \Omega$)
 $T_{\text{AD}} = 62.5 \, \text{ns}$ (minimum, $62.5 \sim 6250 \, \text{ns}$)
 $C_{\text{SAMPLE}} = 3 \, \text{pF}$ (PIC32CX-SG Specification)
 $R_{\text{SAMPLE}} = 2000 \, \Omega$ (PIC32CX-SG Specification)

Target

Calculate the minimum ADC Sample Time by the maximum resolution.

Calculate

$$\begin{aligned} T_{\text{SAMP}}(\text{minimum, 12-bit}) &= ((R_{\text{SAMPLE}} + R_{\text{SOURCE}}) * C_{\text{SAMPLE}} * (\# \text{BitsResolution} + 2) * \ln(2)) / T_{\text{AD}} + 1 \\ &= ((2000 + 6560) * 3 * 10^{-12} * (12 + 2) * \ln(2)) / 62.5 * 10^{-9} + 1 \\ &\approx 4.9872 \approx 4 \text{ (rounded down to nearest whole integer)} \end{aligned}$$

Sampling Time Length

$$\text{SAMPCTRL.SAMPLEN} = (T_{\text{SAMP}} - 1) = (4 - 1) = 3$$

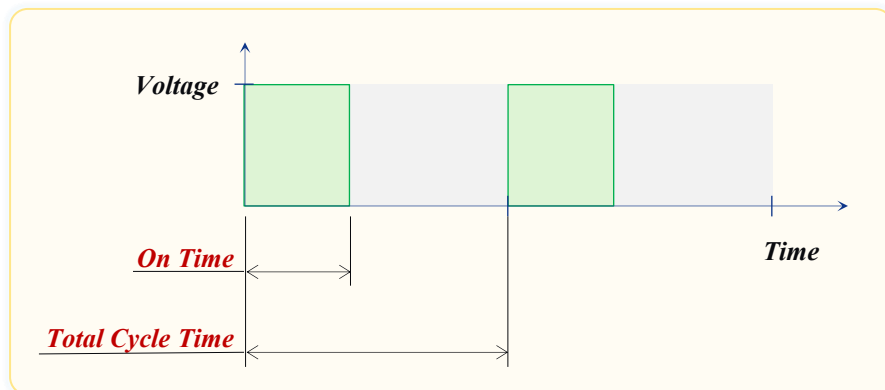
Pulse-Width Modulation - TCC

Pulse-Width Modulation, PWM

- PWM is a technique used to control the average power delivered to a load by rapidly switching a digital signal **ON/OFF** rapidly.
- This method allows digital signals to simulate analog voltage levels by adjusting the "**Duty Cycle**" ratio.

The "**Duty Cycle**" is defined as the ratio of the "**On Time**" to the "**Total Cycle Time**".

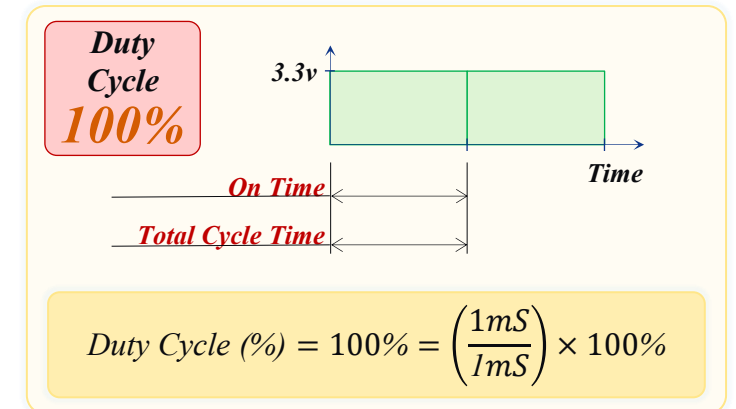
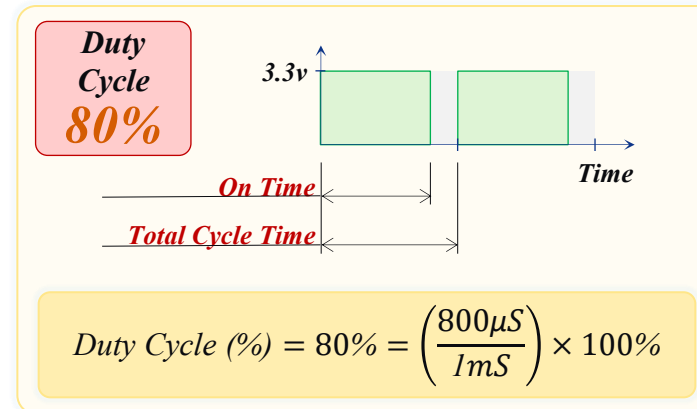
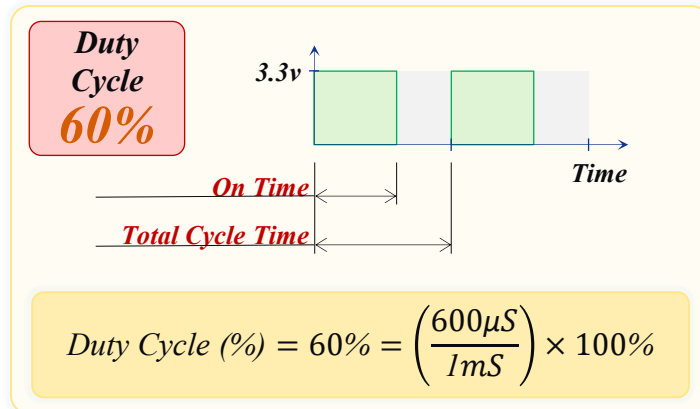
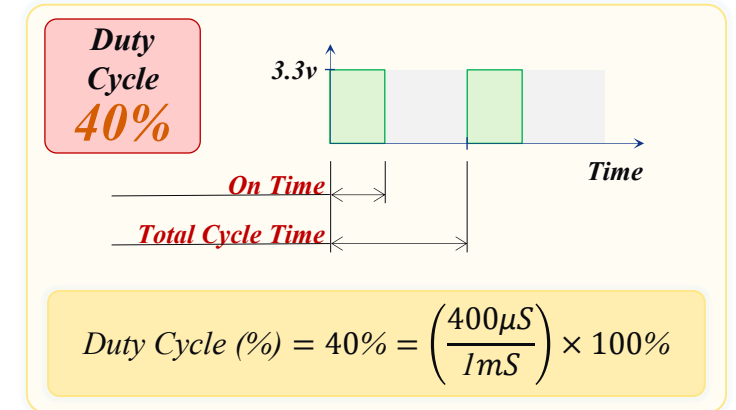
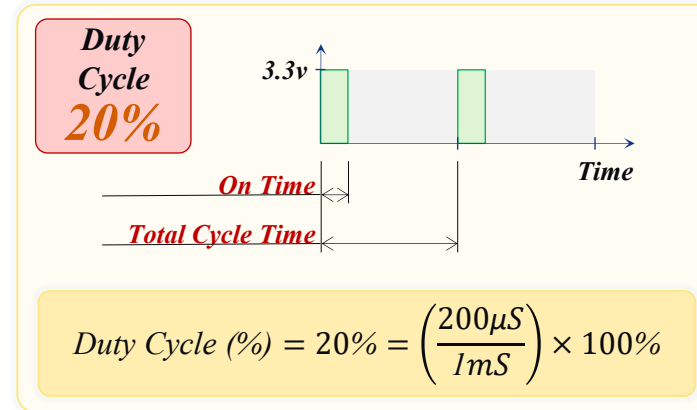
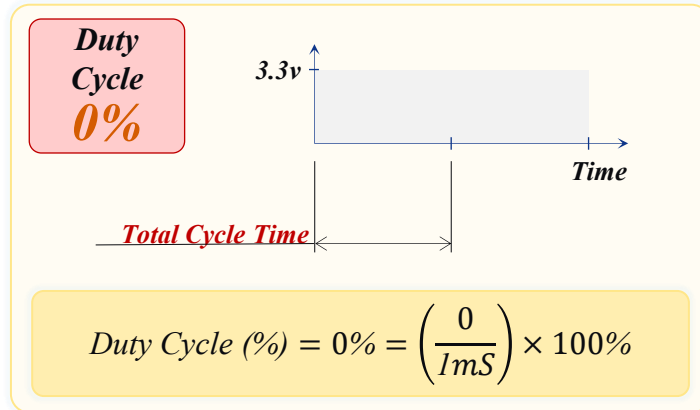
- By adjusting the "**Total Cycle Time**", the PWM frequency can be changed.
- By adjusting the "**On Time**", the "**Duty Cycle**" ratio is modified.



$$\text{Duty Cycle (\%)} = \left(\frac{\text{On Time}}{\text{Total Cycle Time}} \right) \times 100\%$$

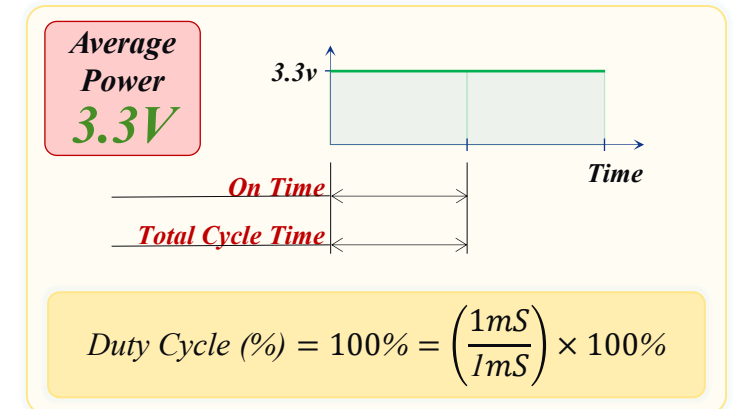
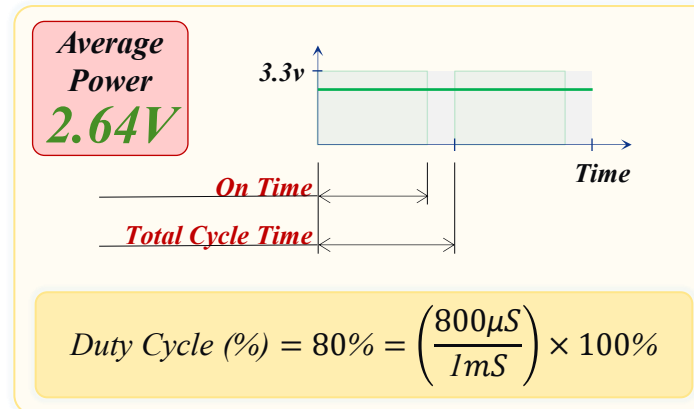
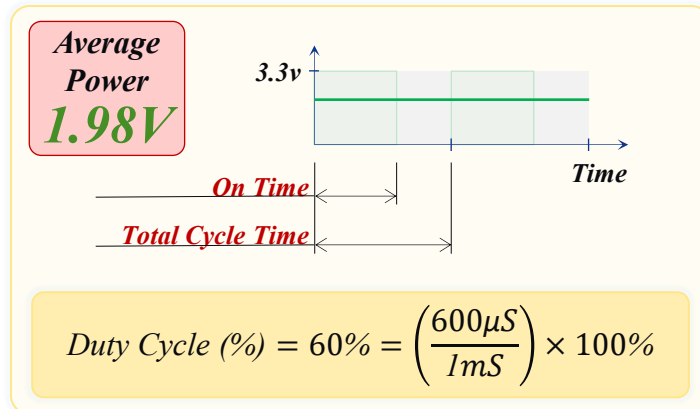
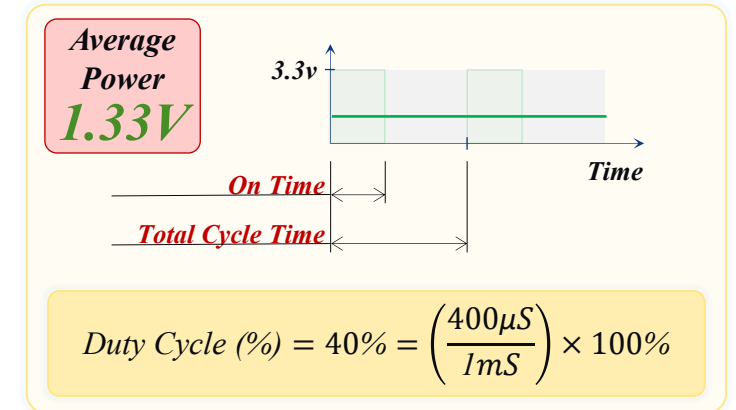
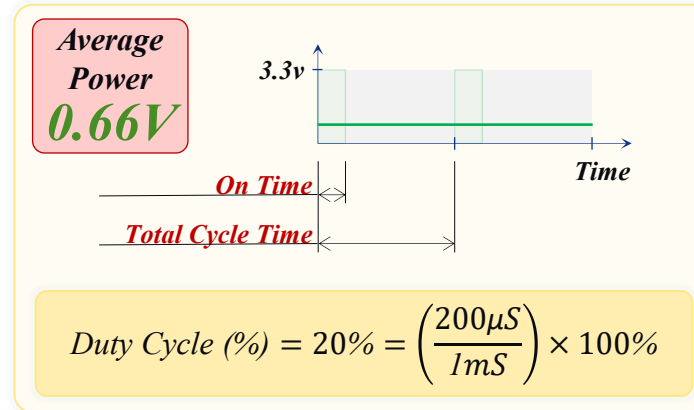
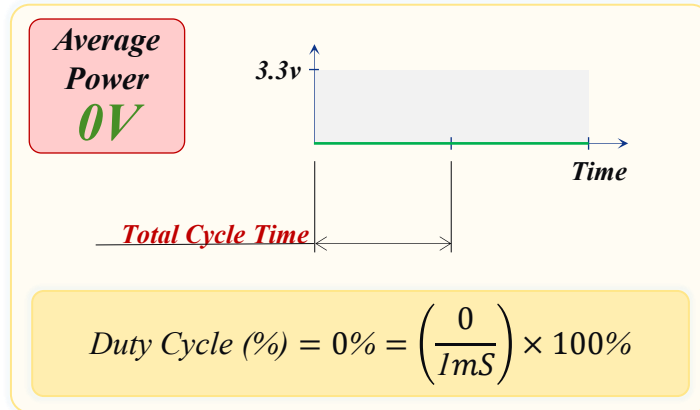
Pulse-Width Modulation, PWM

- This is an example for difference duty cycle ratio and simulate analog average power.



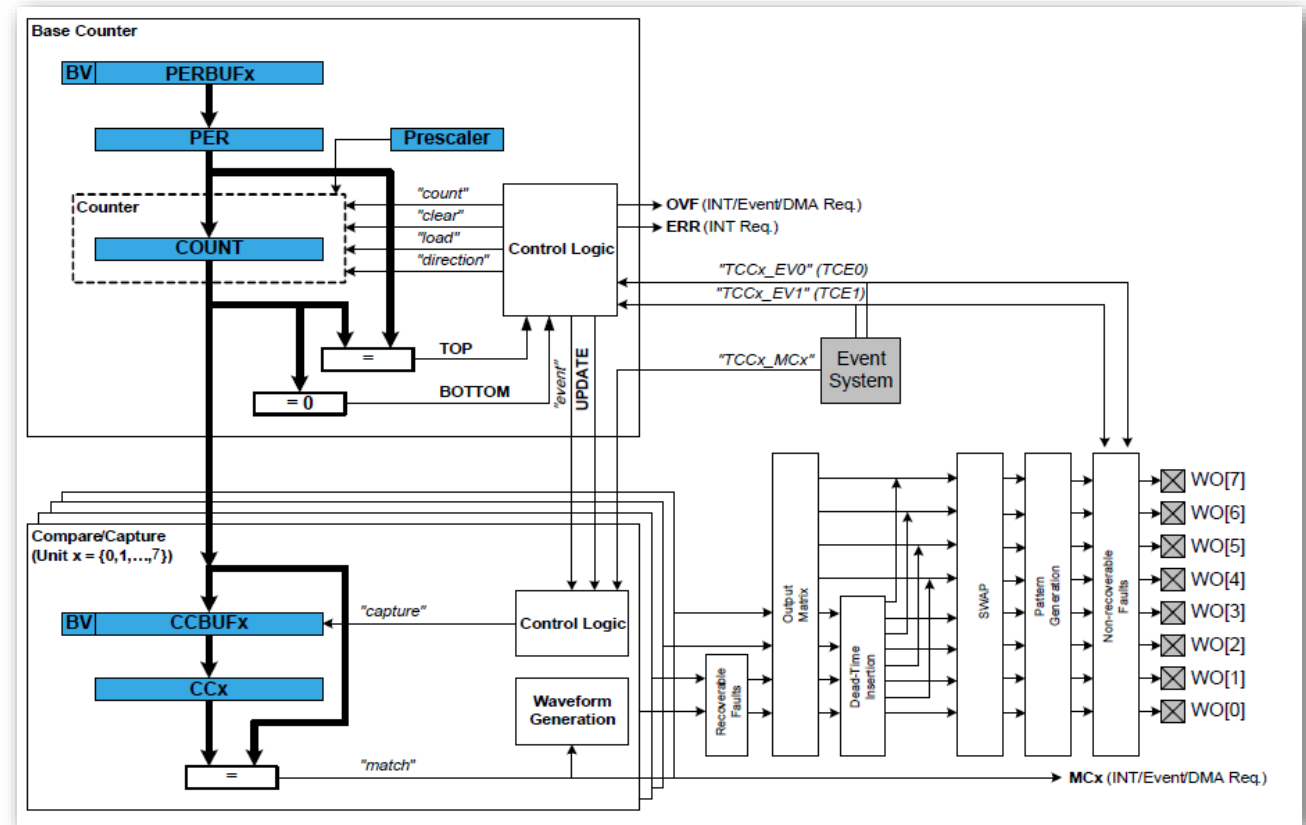
Pulse-Width Modulation, PWM

- This is an example for difference duty cycle ratio and simulate analog average power.



Timer/Counter for Control Applications, TCC

- The device provides five instances of the TCC, [TCC0 - TCC4] and provide feature,
 - Up to six Compare/Capture Channels (CC)
 - Waveform
 - PWM, Frequency Generation
 - Input Capture
 - Event, Frequency, PWM Capture
 - Waveform Extensions
 - Configurable Outputs, Dead-time, Waveform Swap, Pattern Generation, Dithering Support
 - Fault Protection
 - 3 x Input Events, 4 Output Event
 - Support Interrupt, DMA, Event



Timer/Counter for Control Applications, TCC

- Each TCC module supports a different subset of features, as shown in the figure below.
- **TCC0, TCC1 are high-end modules with:**
 - 24-bit counters and more compare/capture channels.
 - Full feature set : fault protection, dithering, output matrix, dead-time, SWAP, and pattern generation
- **TCC2, TCC3, TCC4 are simplified modules with:**
 - 16-bit counters and fewer compare/capture channels.
 - Basic fault support only no advanced waveform features

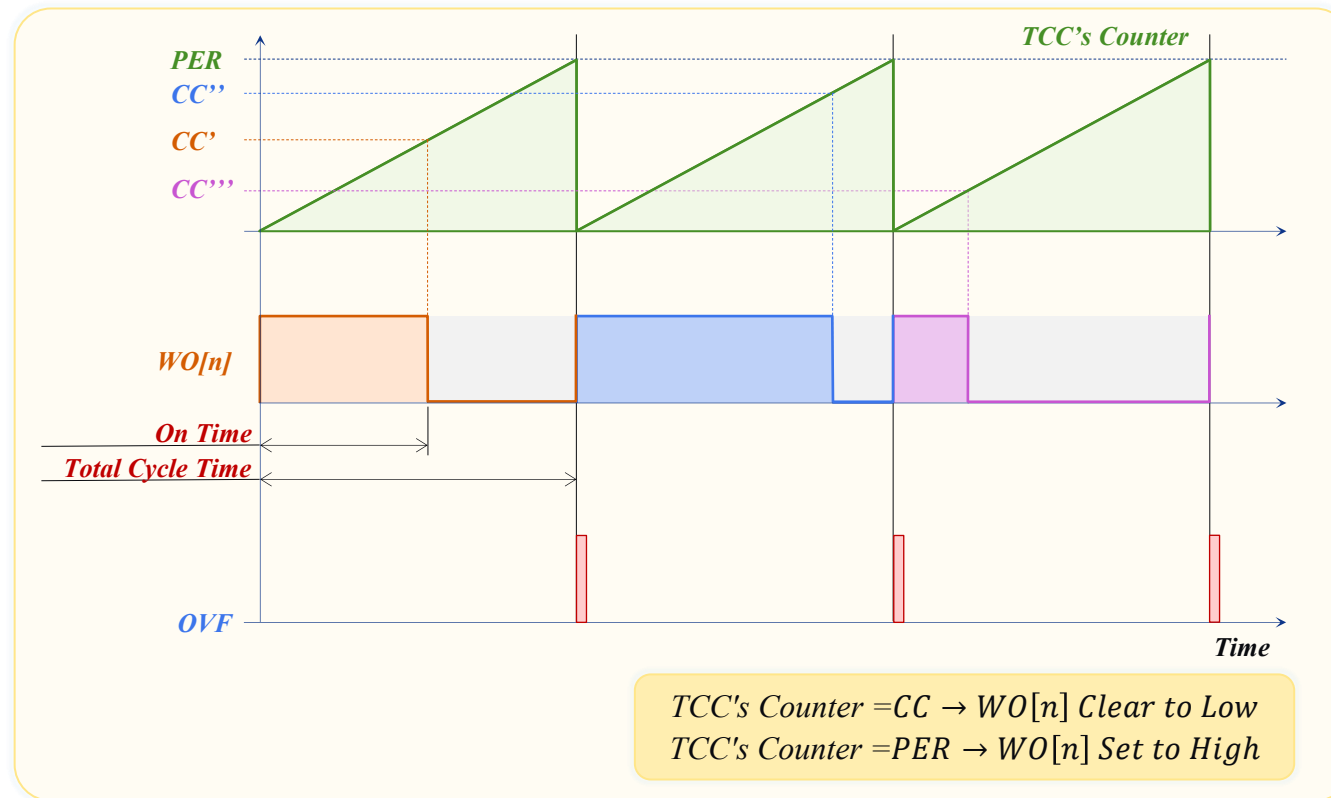
TCC#	Channels (CC_NUM)	Waveform Output (WO_NUM)	Counter size	Fault	Dithering	Output matrix	Dead Time Insertion (DTI)	SWAP	Pattern generation
0	6	8	24-bit	Yes	Yes	Yes	Yes	Yes	Yes
1	4	8	24-bit	Yes	Yes	Yes	Yes	Yes	Yes
2	3	3	16-bit	Yes	-	Yes	-	-	-
3	2	2	16-bit	Yes	-	-	-	-	-
4	2	2	16-bit	Yes	-	-	-	-	-

Timer/Counter for Control Applications, TCC

- **TCC provides 7 difference operation mode for Timer, Counter and Compare.**
 - **Normal Frequency Generation (NFRQ)**
toggled on each compare match
 - **Match Frequency Generation (MFRQ)**
toggles on each update condition
 - **Normal Pulse-Width Modulation (NPWM)**
toggles on each match & update condition
 - **Dual-slope, interrupt/event at TOP (DSTOP)**
toggles on each match & clear on update condition
 - **Dual-slope, interrupt/event at ZERO (DSBOTTOM)**
toggles on each match & clear on update condition
 - **Dual-slope, interrupt/event at Top and ZERO (DSBOTH)**
toggles on each match & clear on update condition
 - **Dual-slope, critical interrupt/event at ZERO (DSCRITICAL)**
toggles on each match & clear on update condition

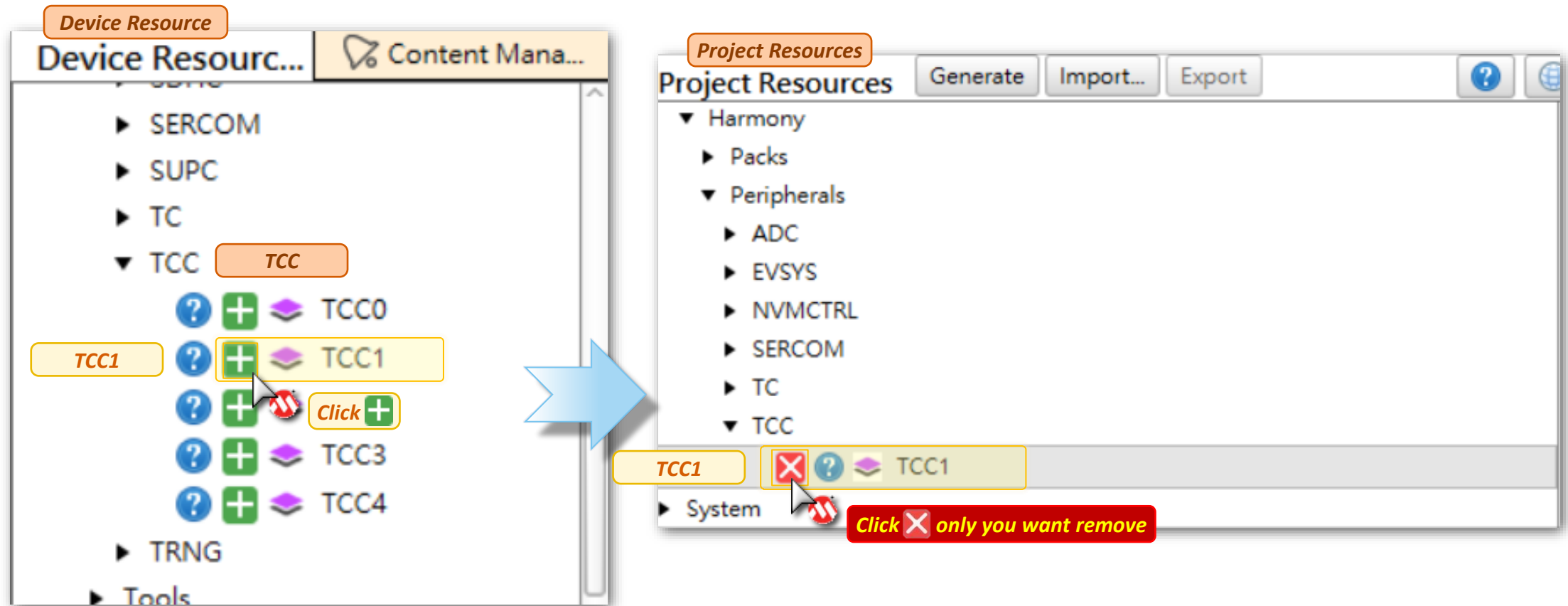
Normal Pulse-Width Modulation, NPWM

- For NPWM,
the Total Cycle Time is controlled by the PER register; the ON Time is controlled by the CC register, WO[x] is set at start and cleared on compare match.



Add or Remove MCC Harmony Module

- Find TCC_n component in **Device Resources**.
- Click on  to add the module to your **Project Resources**.
- Click on  could remove the module from Project Resource.



MCC Harmony Configuration Example - TCC1

Configuration Options x

- TCC1

- Enable Slave ☐
- Run during Standby ☐
- Select Prescaler No division v
- Prescaler and Counter Synchronization Reload or reset counter on next prescaler clock v
- Operating Mode PWM v
 - PWM
 - Select PWM Type NPWM v
 - PWM Direction - Count Down ☐
 - Period Value 59,999 v
 - **** PWM Frequency is 1000 Hz ****
 - Enable Period Interrupt ☐
 - Enable Period Event Out ☐
 - Select Output Matrix Default Channel Outputs v
 - Channel Configurations
 - Channel 0
 - Channel 1
 - Channel 2
 - Channel 3
 - Duty Value 12,000 v
 - Output Polarity Output is ~DIR and set to DIR when counter matches CCx value v
 - Enable Dead Time ☐
 - Swap Outputs ☐
 - Enable Compare Match Interrupt ☐
 - Enable Compare Match Event OUT ☐

TCC1 Polling

TCC1 Polling Interface Functions

- **TCC1_PWMInitialize** : Initializes the TCC1 module for PWM operation.
- **TCC1_PWMStart** : Start the TCC1 counter and begins PWM output.
- **TCC1_PWMStop** : Stop the TCC1 counter and disables PWM output.
- **TCC1_PWMDeadTimeSet** : Sets the dead time values for complementary PWM outputs.
- **TCC1_PWMForceUpdate** : Update of buffered compare/period values.
- **TCC1_PWMPatternSet** : Configures the output pattern enable mask and output value.
- **TCC1_PWMPeriodInterruptEnable** : Enables the period (PER) interrupt.
- **TCC1_PWMPeriodInterruptDisable** : Disables the period (PER) interrupt.
- **TCC1_PWMInterruptStatusGet** : Returns the interrupt status for the TCC1 module.
- **TCC1_PWM24bitPeriodSet** : Set the TCC1 period value for 24-bit mode.
- **TCC1_PWM24bitCounterSet** : Set the TCC1 counter value for 24-bit mode.
- **TCC1_PWM24bitPeriodGet** : Get the TCC1 period value for 24-bit mode.
- **TCC1_PWM24bitCounterGet** : Get the TCC1 counter value for 24-bit mode.
- **TCC1_PWM24bitDutySet** : Sets the duty cycle for a specific channel.

TCC1 Polling

TCC1 Polling Interface Functions

bool TCC1_PWM24bitDutySet(TCC1_CHANNEL_NUM *channel*, uint32_t *duty*)

- **Description:**

- Sets the duty cycle for a specific channel

- **Parameter:**

- *channel* : WM output channel to update (TCC1_CHANNEL0 ~ TCC1_CHANNEL3).
- *duty* : Duty cycle value to be set (0 to 0xFFFFFFFF for 24-bit resolution).

- **Return:**

- *true* : Duty value successfully written to buffer.
- *false* : Buffer is busy; update not performed.

- **Example :**

```
/* Set 10% duty on channel 3 */
```

```
TCC1_PWM24bitDutySet(TCC1_CHANNEL3 , ( TCC1_PWM24bitPeriodGet( ) + 1 ) / 10 );
```

```
/* Set 50% duty on channel 3 */
```

```
TCC1_PWM24bitDutySet(TCC1_CHANNEL3 , ( TCC1_PWM24bitPeriodGet( ) + 1 ) * 0.5 );
```

TCC Polling Example

Configuration Options x

TCC1

- Enable Slave ☐
- Run during Standby ☐
- Select Prescaler No division v
- Prescaler and Counter Synchronization Reload or reset counter on next prescaler clock v
- Operating Mode PWM v
 - PWM
 - Select PWM Type NPWM v
 - PWM Direction - Count Down ☐
 - Period Value 59,999
 - **** PWM Frequency is 1000 Hz ****
 - Enable Period Interrupt ☐
 - Enable Period Event Out ☐
 - Select Output Matrix Default Channel Outputs v
 - Channel Configurations
 - Channel 0
 - Channel 1
 - Channel 2
 - Channel 3
 - Duty Value 12,000
 - Output Polarity Output is ~DIR and set to DIR when counter matches CCx value v
 - Enable Dead Time ☐
 - Swap Outputs ☐
 - Enable Compare Match Interrupt ☐
 - Enable Compare Match Event OUT ☐

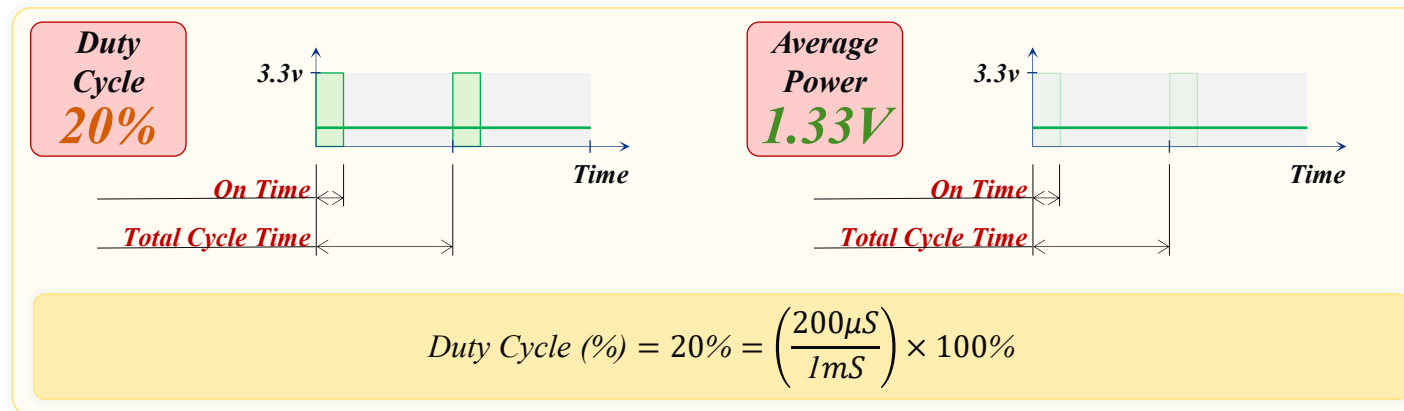
```
int main ( void )
{
    SYS_Initialize ( NULL );
    ...
    TCC1_PWMStart( );
    TCC1_PWM24bitDutySet(TCC1_CHANNEL3, ( ( TCC1_PWM24bitPeriodGet() + 1 ) / 5 );

    while ( true )
        SYS_Tasks ( );
    return ( EXIT_FAILURE );
}
```

Lab11 LED Dimming TCC PWM

🔧 Lab11 LED Dimming TCC PWM

- Add **TCC1 NPWM Mode (Single Slope PWM)** function to your project.
- **Disable LED3(PA15) GPIO function firstly.**
- Configure **TCC1** to output PWM waveform to **LED3(PA15)**.
- PWM frequency is **1kHz** with **20% Duty**.

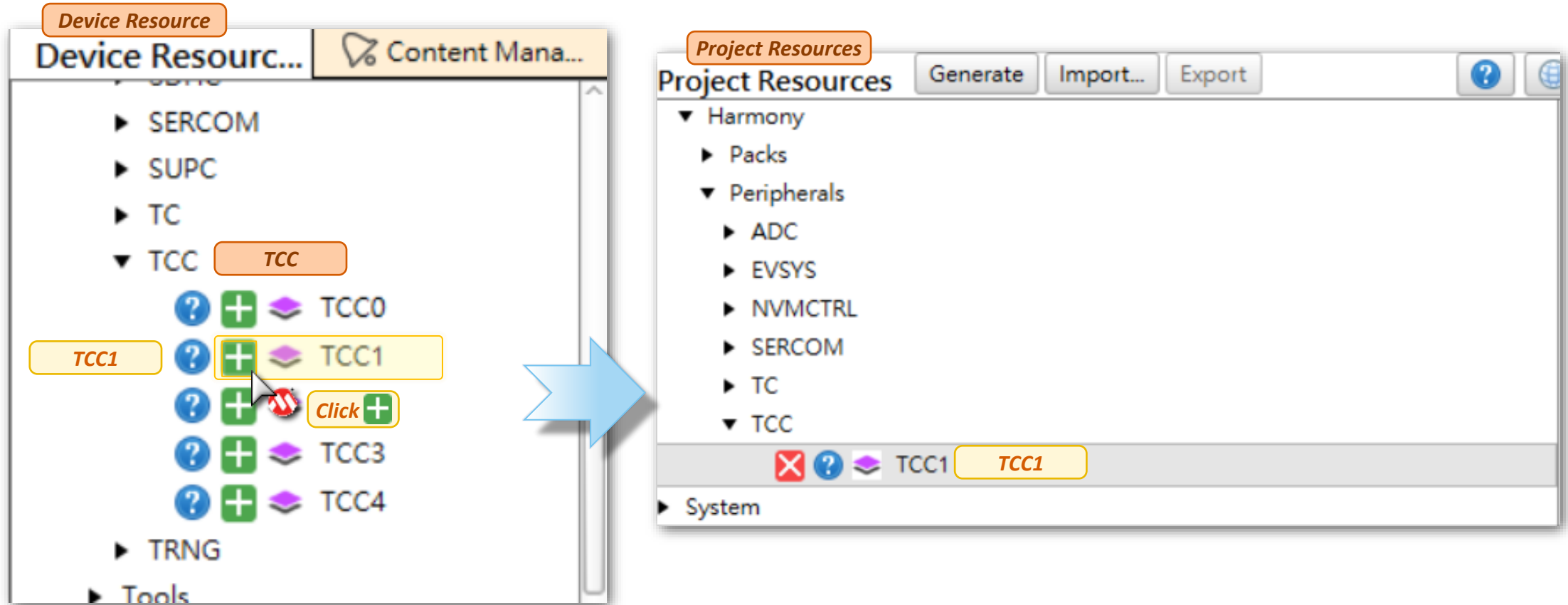


Let's go !

⚙️ Lab11 LED Dimming TCC PWM

Step 1

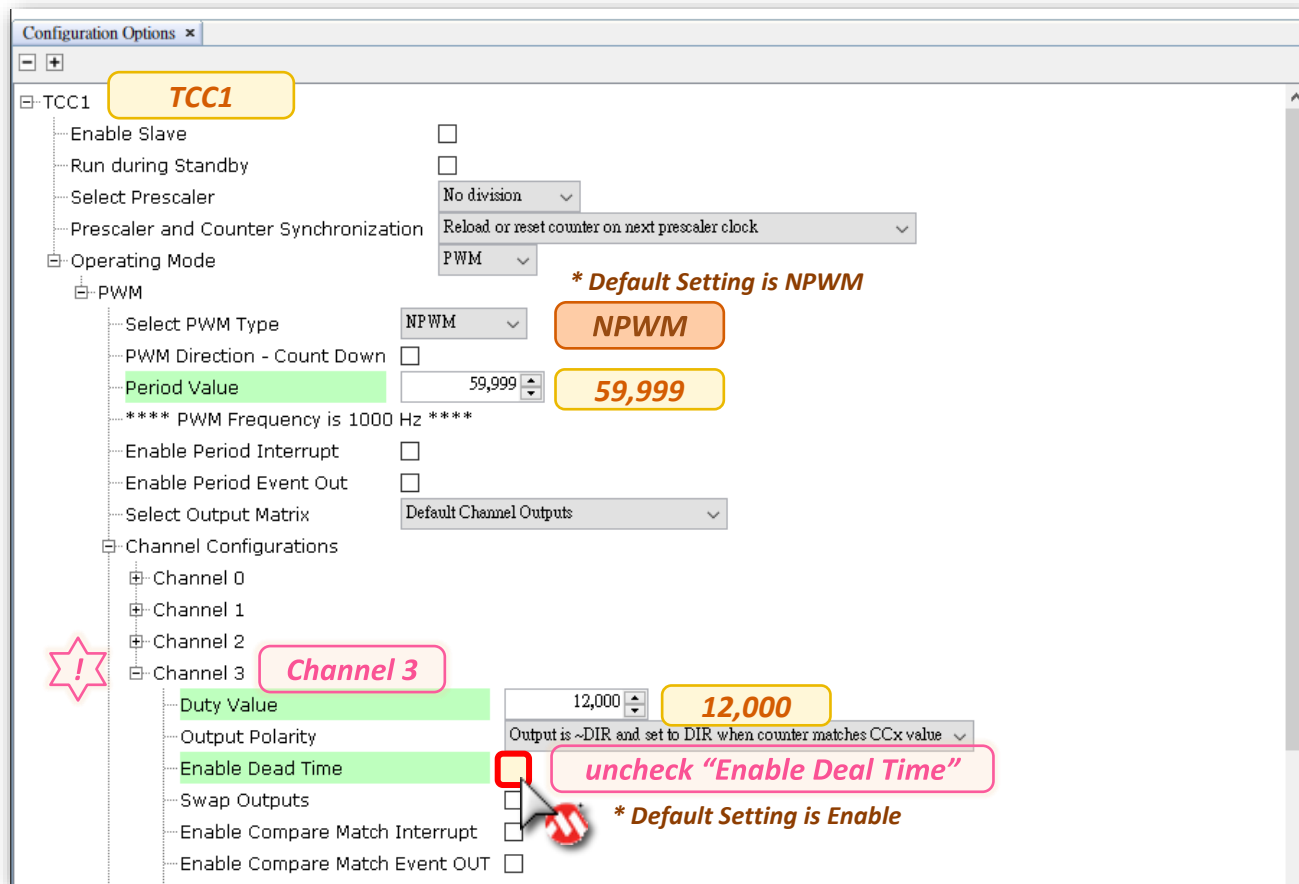
- Find **TCC1** component in **Device Resources**
- Double click **TCC1** to add the module to your **Project Resources**.



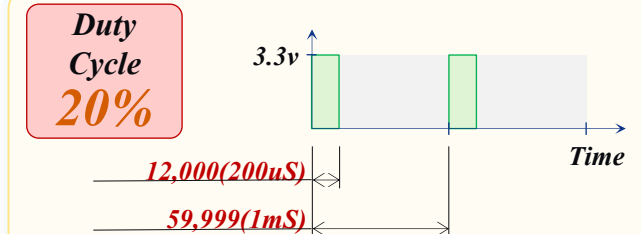
⚙️ Lab11 LED Dimming TCC PWM

Step 2

- Set TCC1 to **PWM** Mode, Period Value **59,999**, Duty Value **6,000** in **Channel 3**.



TCC1 Clock = 60MHz (GCLK1)
One second counts of system clock :
60,000,000 counts
One second counts after Prescaler :
60000000 / 1 60,000,000 counts
1ms counts for PWM period target
60000000 / 1000 60,000 counts
Counter clear to zero at the next tick
59,999 + 1 60,000 counts
20% counts for PWM duty target :
60,000 * 20% 12,000 counts

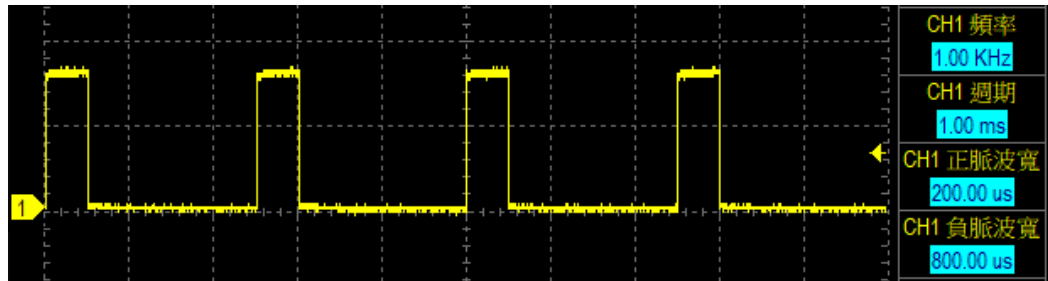


$$\text{Duty Cycle (\%)} = 20\% = \left(\frac{200\mu\text{s}}{1\text{ms}} \right) \times 100\%$$

⚙️ Lab11 LED Dimming TCC PWM

Dead Time Control

- When the TCC generates complementary PWM signals for a high-side (HS) and low-side (LS) switch in the same half-bridge, it adds a small delay so that:
 - When HS turns off → wait for dead time → LS turns on, When LS turns off → wait for dead time → HS turns on
- This ensures there's no moment when HS and LS are both on at the same time, which would short the power supply through the switches; Set the dead time in the TCC's DTI (Dead-Time Insertion) unit, and it will automatically apply this delay to both switching directions.



Channel 3

Duty Value 12,000

Output Polarity Output is ~DIR and set

Enable Dead Time ☐

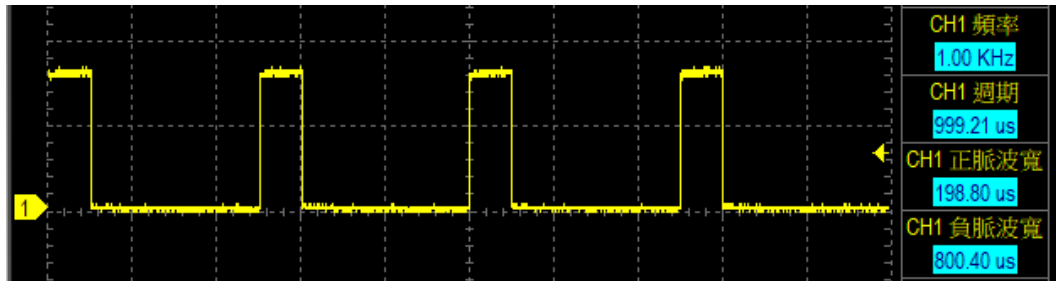
Swap Outputs ☐

Enable Compare Match Interrupt ☐

Enable Compare Match Event OUT ☐

Enable Compare Match Event IN ☐

uncheck "Enable Deal Time"



Channel 3

Duty Value 12,000

Output Polarity Output is ~DIR and set

Enable Dead Time ☒

Swap Outputs ☐

Enable Compare Match Interrupt ☐

Enable Compare Match Event OUT ☐

Enable Compare Match Event IN ☐

Dead Time

Dead Time for Low Side Output 64

**** Low side dead time is 1.06666666667 us ****

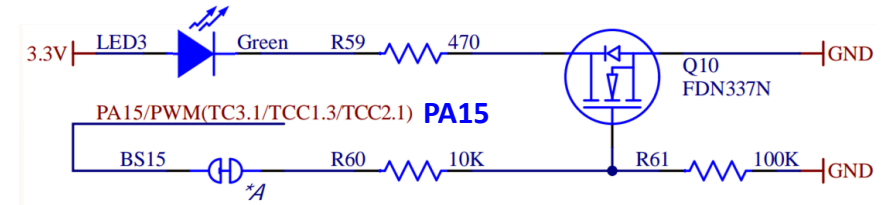
Dead Time for High Side Output 64

**** High side dead time is 1.06666666667 us ****

✖ Lab11 LED Dimming TCC PWM

Step 3

- Pin Settings of the **TCC_PWM**
 - Disable LED3(PA15) GPIO function firstly.
 - Enable TCC1 Waveform Output 3 (TCC1_WO3) to PA15



Schematic	Pin	Direction
TCC1_WO3	PA15	TCC1.3 Output

Pin Settings - Editor

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
55	PC11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
56	PC12		Available	Digital	High Impedance	Low	☐	☐	NORMAL
57	PC13		Available	Digital	High Impedance	Low	☐	☐	NORMAL
58	PC14		Available	Digital	High Impedance	Low	☐	☐	NORMAL
59	PC15		Available	Digital	High Impedance	Low	☐	☐	NORMAL
60	PA12	LED1	GPIO	Digital	Out	Low	☐	☐	NORMAL
61	PA13	LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL
62	PA14		Available	Digital	High Impedance	Low	☐	☐	NORMAL
63	PA15	LED3	GPIO	Digital	Out	Low	☐	☐	NORMAL
64	VSS				High Impedance	Low	☐	☐	NORMAL
65	VDD				High Impedance	Low	☐	☐	NORMAL

Table View Easy View

Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
	Available	Digital	High Impedance	Low	☐	☐	NORMAL
	Available	Digital	High Impedance	Low	☐	☐	NORMAL
	Available	Digital	High Impedance	Low	☐	☐	NORMAL
	Available	Digital	High Impedance	Low	☐	☐	NORMAL
LED1	GPIO	Digital	Out	Low	☐	☐	NORMAL
LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL
	Available	Digital	High Impedance	Low	☐	☐	NORMAL
PA15	PA14						
PA15	PA15	TCC1_WO3	Digital	Out	n/a	☐	NORMAL
64	VSS		High Impedance	Low	☐	☐	NORMAL
65	VDD		High Impedance	Low	☐	☐	NORMAL

⚙️ Lab11 LED Dimming TCC PWM

Step 4

- Click on **Generate** button to generate your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Project Resources' tab. The 'Generate' button is highlighted with a yellow box and a red arrow. Below it, the 'System' resource is also highlighted. The 'Output' window shows the following log:

```
15:17:50.827 INFO: ... PIC32CX1025SG41128_DFP (p10 (pic32cx1025sg41128.11) Success.
15:17:50.827 INFO: HARMONY: [ MHC ] Executing generate hook
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

Lab11 LED Dimming TCC PWM

Step 5

- Open **main.c**, **TODO11.01** to Start TCC1 PWM and **11.02** to comment out BT1/LED3 control function.

```
int main ( void )
{
    SYS_Initialize ( NULL );
    ...
    ADC0_CallbackRegister( ADC0_Complete, (uintptr_t )NULL );
    ADC1_CallbackRegister( ADC1_Complete, (uintptr_t )NULL );
    ADC0_Enable();
    ADC1_Enable();

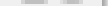
    // TODO 11.01
    TCC1_PWMStart();

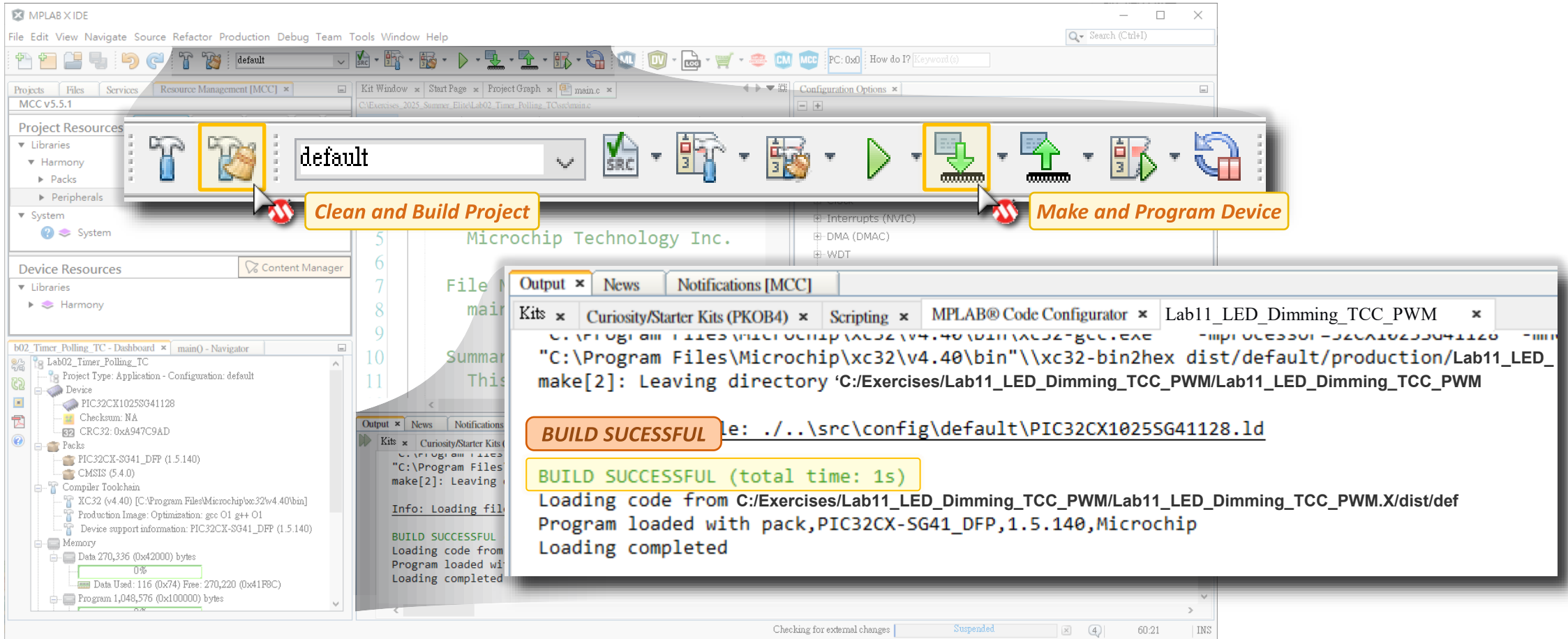
    while (true)
    {
        SYS_Tasks();
        ...
        if( ADC1_IsCompleted )
        {
            ADC1_IsCompleted = 0;
            myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );
            myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );
        }

        // TODO 11.02
        // if ( BT1_Get() ) LED3_Clear();
        // else             LED3_Set();
    }
    return ( EXIT_FAILURE);
}
```

Lab11 LED Dimming TCC PWM

Step 6

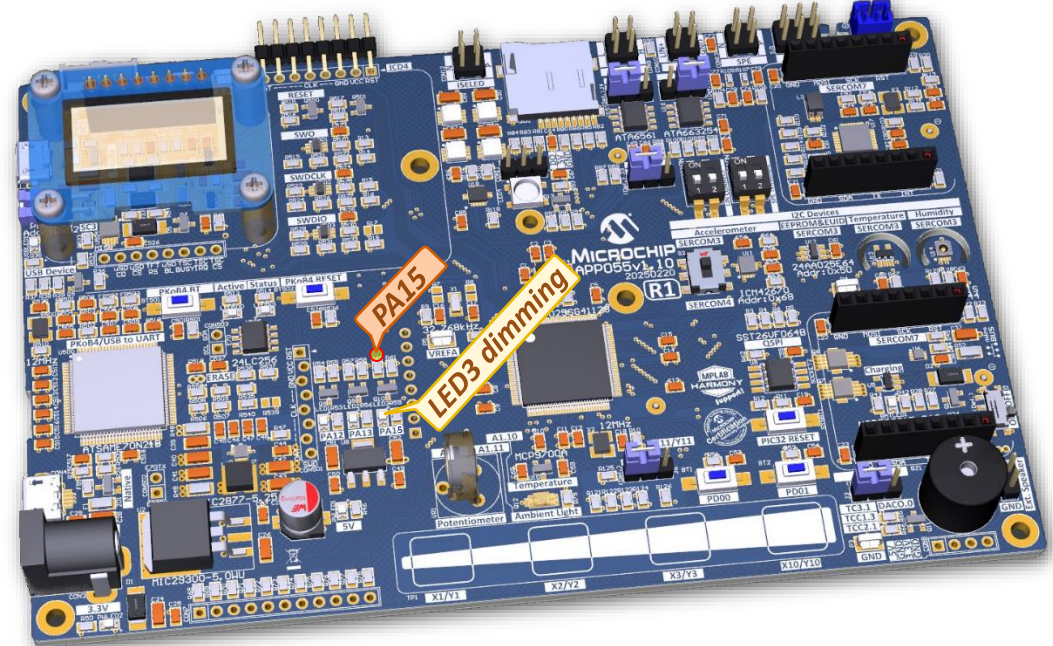
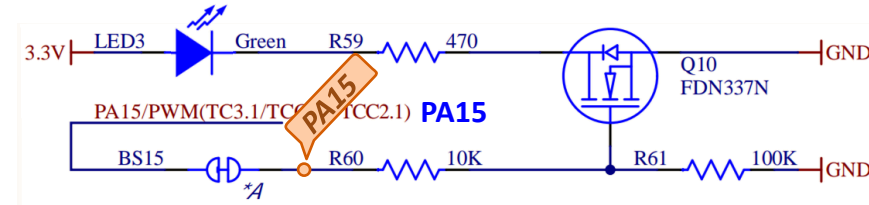
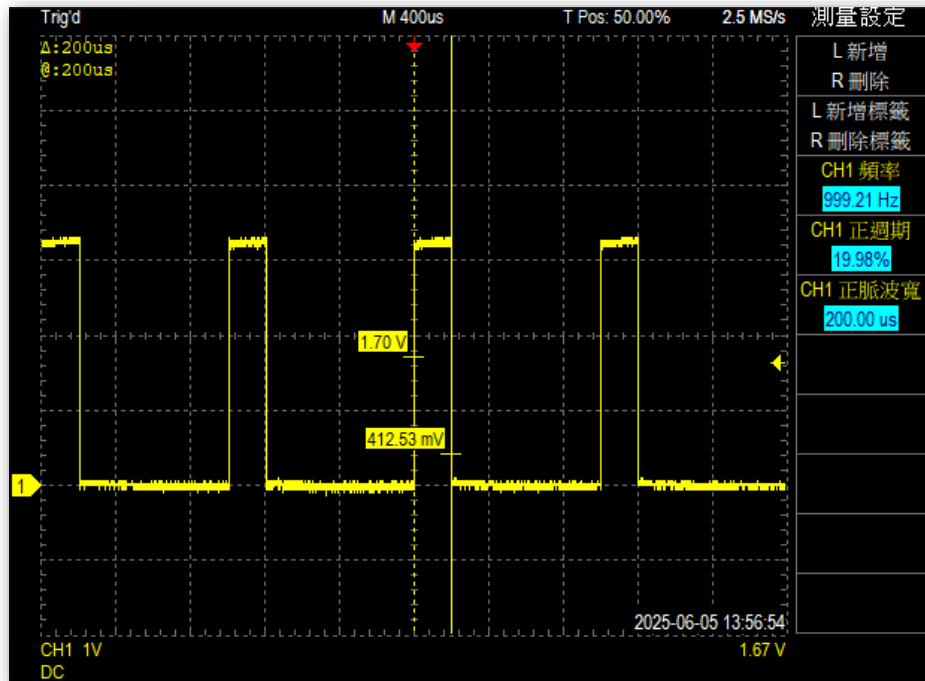
- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



Lab11 LED Dimming TCC PWM

Result

- LED3 brightness is controlled by the TCC1 PWM



Lab12 LED Dimming Adjustable TCC PWM

⚙️ Lab12 LED Dimming Adjustable TCC PWM

- Try to use **VR1** to adjust **TCC1** PWM duty for **LED3** dimming control.

VR1 Value(ADC0_Result) = 0, TCC1 PWM Duty = 0%

VR1 Value(ADC0_Result) = 4,095, TCC1 PWM Duty = 100%



$$\frac{ADC\ Value}{4095} \times PWM\ Period$$

Let's go !

Lab12 LED Dimming Adjustable TCC PWM

Step 1

- Open **main.c** and finish **TODO12.01** to Turning TCC1 PWM by ADC Value from VR1.

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

    ADC1_Enable();

    TCC1_PWMStart();

    while (true)
    {
        SYS_Tasks();
        ...

        if( ADC0_IsCompleted )
        {
            ADC0_IsCompleted = 0;
            myprintf( "@1,2;VR1 Value : %4d", ADC0_Result )

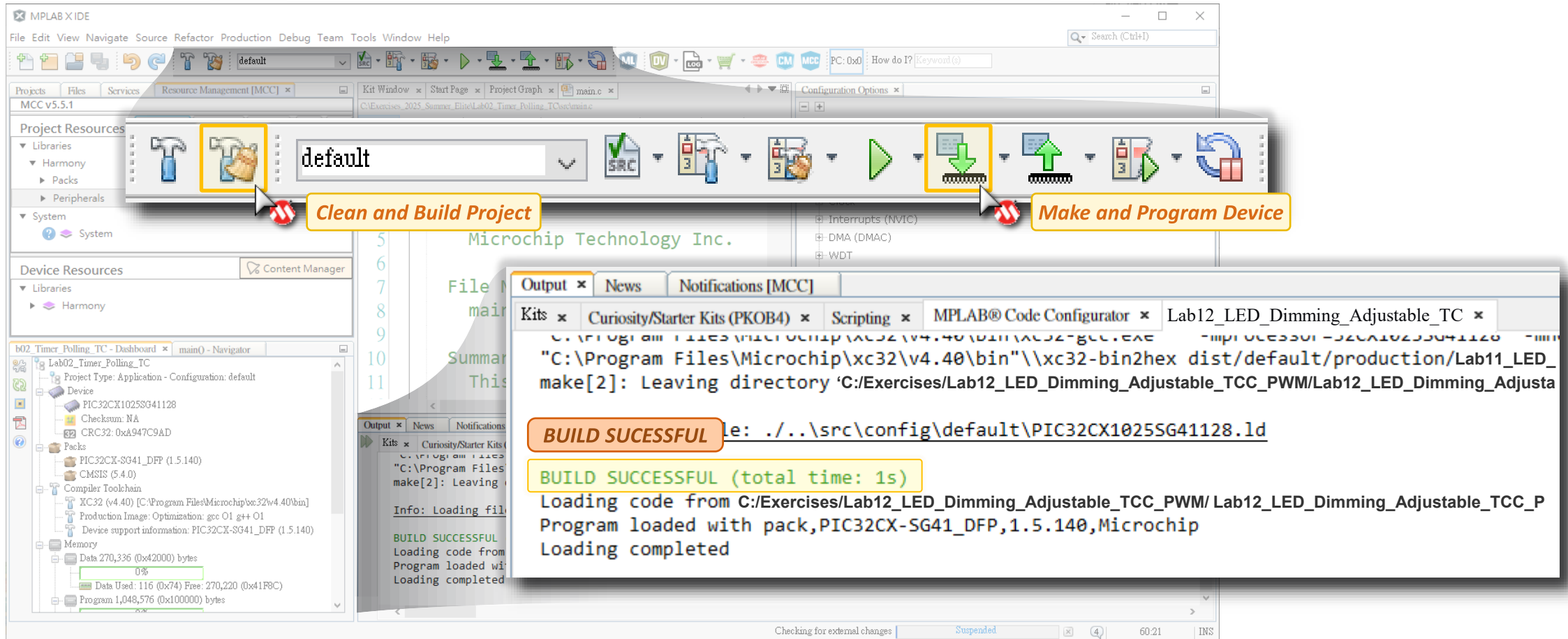
// TODO 12.01
            TCC1_PWM24bitDutySet(TCC1_CHANNEL3, ( (uint32_t)ADC0_Result * (TCC1_PWM24bitPeriodGet() + 1) ) / 4095 );
        }

        if( ADC1_IsCompleted )
        {
            ADC1_IsCompleted = 0;
            myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );
            myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );
        }
    }
}
```

⚙️ Lab12 LED Dimming Adjustable TCC PWM

Step 2

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The main toolbar contains several icons. Two icons are highlighted with yellow boxes and red arrows pointing to them: the 'Clean and Build Project' icon (a hammer and a document) and the 'Program code to EVB' icon (a green arrow pointing down to a circuit board). Below these icons are yellow callout boxes with the text 'Clean and Build Project' and 'Make and Program Device' respectively.

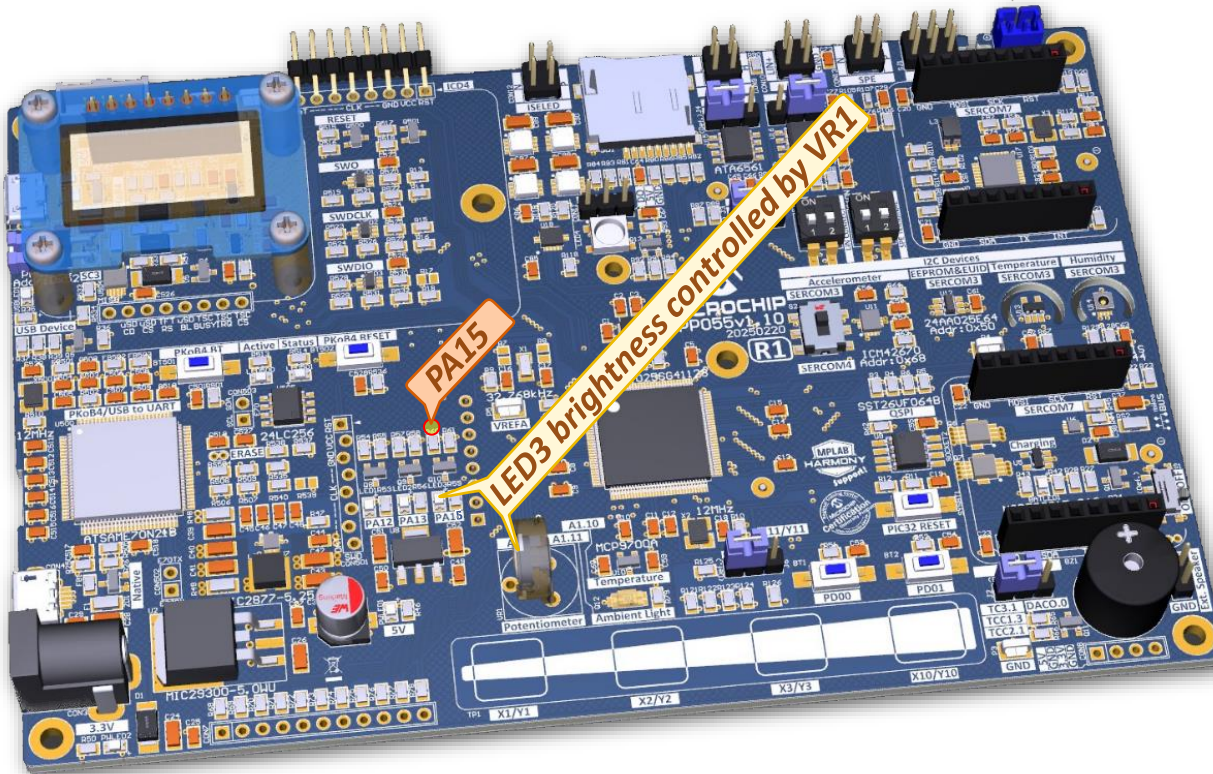
The 'Output' window is open, showing the results of the build process. The output text includes:

```
le: ../../src/config/default/PIC32CX1025SG41128.ld
BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab12_LED_Dimming_Adjustable_TCC_PWM/ Lab12_LED_Dimming_Adjustable_TCC_P
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

⚙️ Lab12 LED Dimming Adjustable TCC PWM

Result

- LED3 brightness is controlled by the ADC value from VR1.
- The result of the mathematical comparison is displayed on Tera Term.

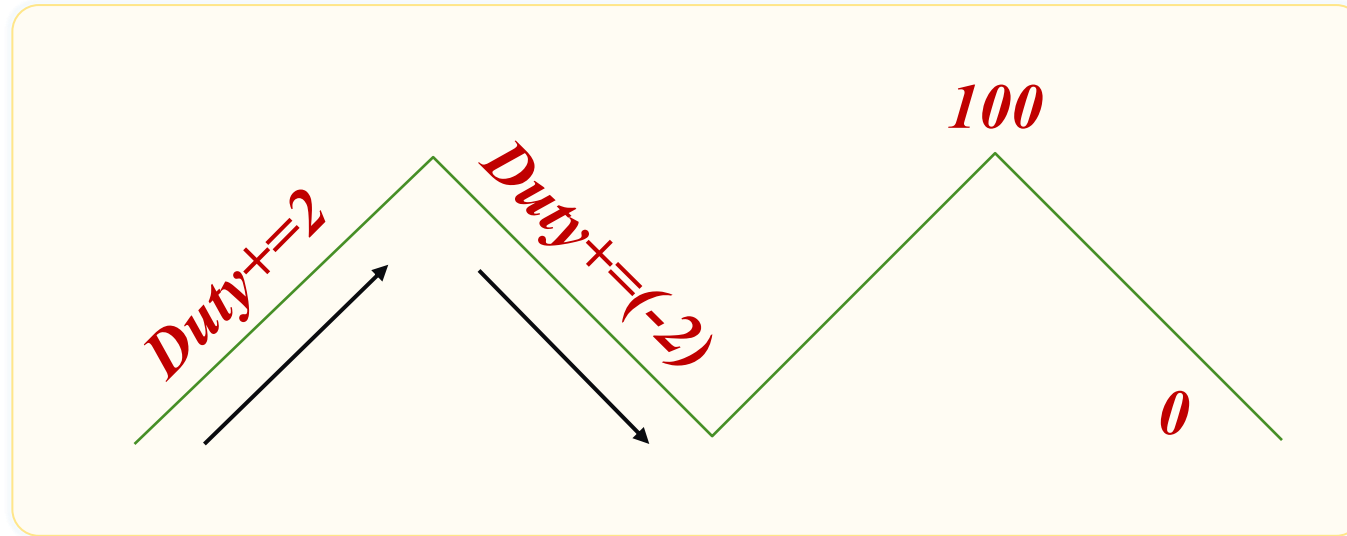


```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Hello World!!
VR1 Value : 4095
Temperature Value : 985
Lightness Value : 3256
```

Lab13 LED Breath TCC PWM

⚙️ Lab13 LED Breath TCC PWM

- Use **TC1** to control PWM duty for **breathing light effect** on **LED3** .
Duty cycle changes in steps of 2% every 40ms intervals.
(Duty 0% -> Duty 100% -> Duty 0% -> ... repeat)



⚙️ Lab13 LED Breath TCC PWM

Step 1

- Finish **TODO13.01** to **13.02** for define variables and place breathing light algorithm.

```
...
volatile uint8_t  ADC1_ChannelCount = sizeof(ADC1_Channel) / sizeof(ADC1_Channel[0]);

// TODO 13.01
uint8_t Duty = 50;
int8_t DutyDistance = 2;

void ADC0_Complete( ADC_STATUS status, uintptr_t context )
{
}

...
int main ( void )
{
    ...
    while (true)
    {
        ...
        if (TC1_HasExpired)
        {
            TC1_HasExpired = 0;
            LED2_Toggle();

// TODO 13.02
            Duty += DutyDistance;
            if (Duty >= 100 || Duty <= 0)
                DutyDistance = -DutyDistance;

            TCC1_PWM24bitDutySet(TCC1_CHANNEL3, ( (uint32_t)Duty * (TCC1_PWM24bitPeriodGet() + 1) ) / 100 );
        }

        if (USART1_IsReceived)
        {
            USART1_IsReceived = 0;
        }
    }
}
```

Lab13 LED Breath TCC PWM

Step 2

- Finish **TODO13.03** to comment out the ADC dimming algorithm from previous Lab.

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

    ADC1_Enable();

    TCC1_PWMStart();

    while (true)
    {
        SYS_Tasks();
        ...

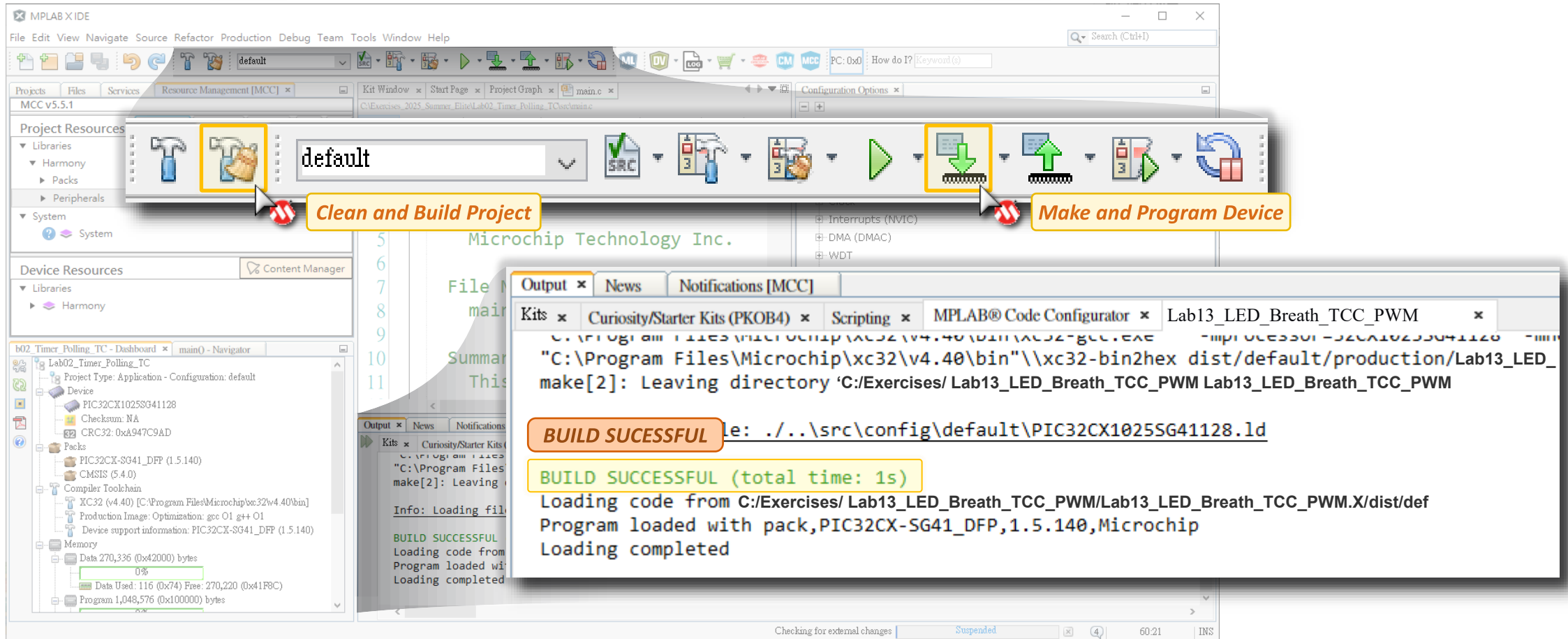
        if( ADC0_IsCompleted )
        {
            ADC0_IsCompleted = 0;
            myprintf( "@1,2;VR1 Value : %4d", ADC0_Result )
// TODO 13.03
            // TCC1_PWM24bitDutySet(TCC1_CHANNEL3, (((uint32_t) ADC0_Result * TCC1_PWM24bitPeriodGet() + 1) / 4095));
        }

        if( ADC1_IsCompleted )
        {
            ADC1_IsCompleted = 0;
            myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );
            myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );
        }
    }
}
```


⚙️ Lab13 LED Breath TCC PWM

Step 3

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains icons for various actions. Two icons are highlighted with yellow boxes and red arrows pointing to them:

- Clean and Build Project**: Indicated by a yellow box around the icon showing a hammer and a wrench.
- Make and Program Device**: Indicated by a yellow box around the icon showing a green arrow pointing down to a circuit board.

The bottom right of the IDE shows the **Output** window with the following text:

```
Kits x Curiosity/Starter Kits (PKOB4) x Scripting x MPLAB® Code Configurator x Lab13_LED_Breath_TCC_PWM x
C:\Program Files\Microchip\XC32\v4.40\bin\XC32-GCC.EXE -imp -O3 -S -x -c32 -x10255G41128 -Tlink
"C:\Program Files\Microchip\xc32\v4.40\bin"\xc32-bin2hex dist/default/production/Lab13_LED_
make[2]: Leaving directory 'C:/Exercises/ Lab13_LED_Breath_TCC_PWM Lab13_LED_Breath_TCC_PWM'

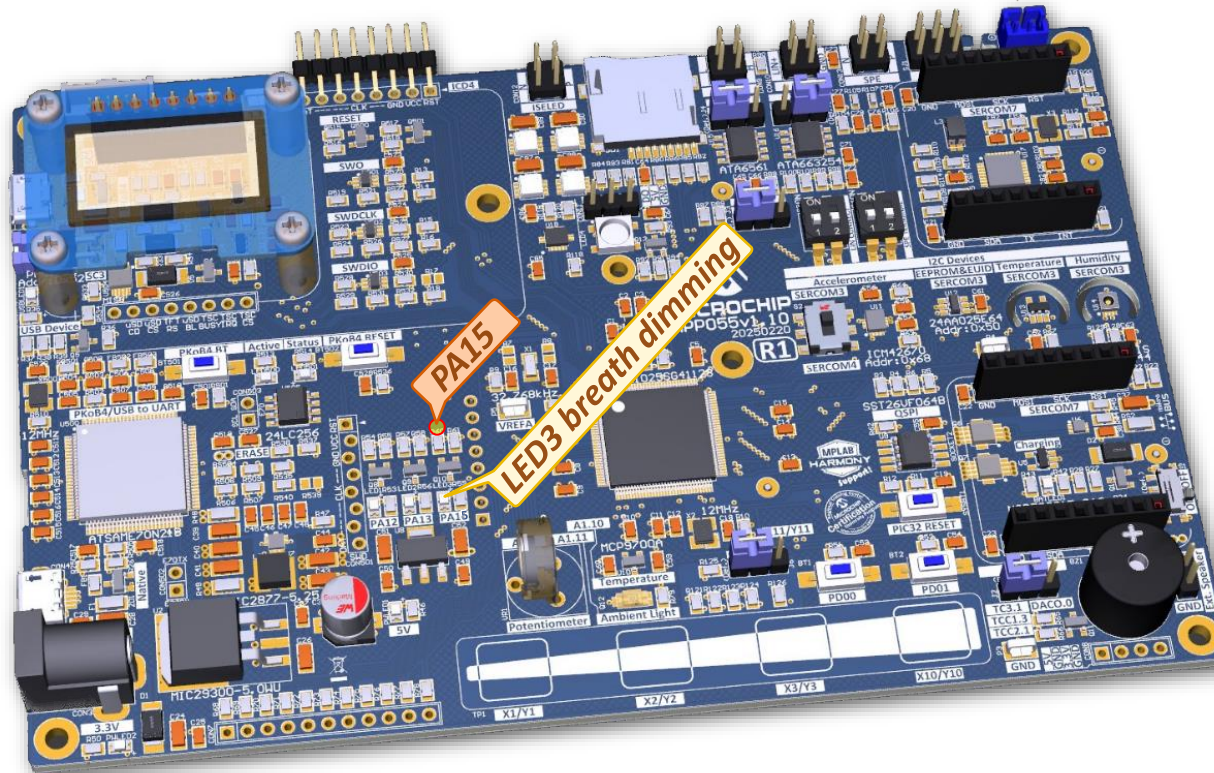
BUILD SUCCESSFUL le: ../../src/config/default/PIC32CX10255G41128.ld

BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/ Lab13_LED_Breath_TCC_PWM/Lab13_LED_Breath_TCC_PWM.X/dist/def
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

Lab13 LED Breath TCC PWM

Result

- **The brightness of LED3 is controlled automatically.**
The PWM duty cycle changes in steps of 2% at 40 ms intervals.



Appendix : TCC PWM Waveform Output Matrix

Sample Time and Conversion Time

TCC PWM Waveform Output Matrix

- The TCC Configuration and Output Matrix

Table 45-1. TCC Configuration Summary

TCC#	Channels (CC_NUM)	Waveform Output (WO_NUM)	Counter size	Fault	Dithering	Output matrix	Dead Time Insertion (DTI)	SWAP	Pattern generation
0	6	8	24-bit	Yes	Yes	Yes	Yes	Yes	Yes
1	4	8	24-bit	Yes	Yes	Yes	Yes	Yes	Yes
2	3	3	16-bit	Yes	-	Yes	-	-	-
3	2	2	16-bit	Yes	-	-	-	-	-
4	2	2	16-bit	Yes	-	-	-	-	-

TCC3 and TCC4 do not support Output Matrix

Table 45-5. Output Matrix Channel Pin Routing Configuration

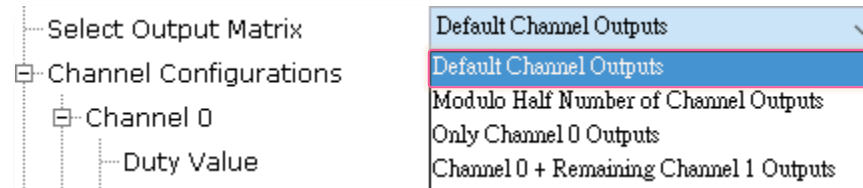
Value	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
0x0	CC1	CC0	CC5	CC4	CC3	CC2	CC1	CC0
0x1	CC1	CC0	CC2	CC1	CC0	CC2	CC1	CC0
0x2	CC0	CC0	CC0	CC0	CC0	CC0	CC0	CC0
0x3	CC1	CC1	CC1	CC1	CC1	CC1	CC1	CC0

TCC PWM Waveform Output Matrix

- The TCC Output Matrix OTMX=0. (Default Channel Outputs)

Table 45-5. Output Matrix Channel Pin Routing Configuration

Value	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
0x0	CC1	CC0	CC5	CC4	CC3	CC2	CC1	CC0
0x1	CC1	CC0	CC2	CC1	CC0	CC2	CC1	CC0
0x2	CC0	CC0	CC0	CC0	CC0	CC0	CC0	CC0
0x3	CC1	CC1	CC1	CC1	CC1	CC1	CC1	CC0



TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
TCC0	Channel	CC_NUM=6	1	0	5	4	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	3	2	1	0	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						2	1	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]
TCC3	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]
TCC4	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]

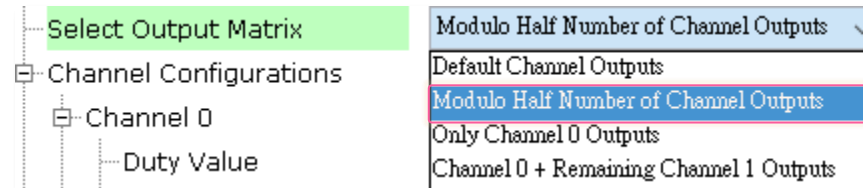
TCC3 and TCC4 do not support Output Matrix

TCC PWM Waveform Output Matrix

- The TCC Output Matrix OTMX=1. (Module Half Number of Channel Outputs)

Table 45-5. Output Matrix Channel Pin Routing Configuration

Value	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
0x0	CC1	CC0	CC5	CC4	CC3	CC2	CC1	CC0
0x1	CC1	CC0	CC2	CC1	CC0	CC2	CC1	CC0
0x2	CC0	CC0	CC0	CC0	CC0	CC0	CC0	CC0
0x3	CC1	CC1	CC1	CC1	CC1	CC1	CC1	CC0



TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
TCC0	Channel	CC_NUM=6	1	0	2	1	0	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	1	0	1	0	1	0	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						0	1	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]
TCC3	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]
TCC4	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]

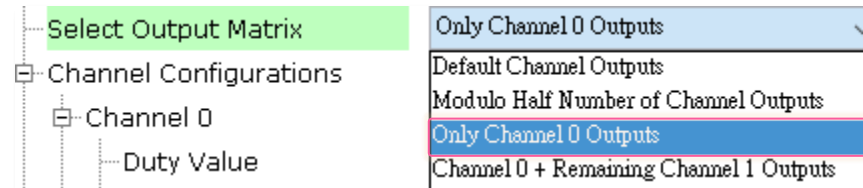
TCC3 and TCC4 do not support Output Matrix

TCC PWM Waveform Output Matrix

- The TCC Output Matrix OTMX=2. (Only Channel 0 Output)

Table 45-5. Output Matrix Channel Pin Routing Configuration

Value	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
0x0	CC1	CC0	CC5	CC4	CC3	CC2	CC1	CC0
0x1	CC1	CC0	CC2	CC1	CC0	CC2	CC1	CC0
0x2	CC0	CC0	CC0	CC0	CC0	CC0	CC0	CC0
0x3	CC1	CC1	CC1	CC1	CC1	CC1	CC1	CC0



TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
TCC0	Channel	CC_NUM=6	0	0	0	0	0	0	0	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	0	0	0	0	0	0	0	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						0	0	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]
TCC3	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]
TCC4	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]

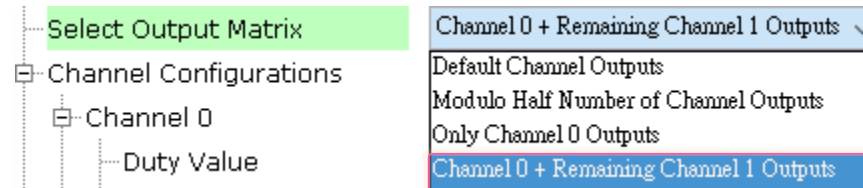
TCC3 and TCC4 do not support Output Matrix

TCC PWM Waveform Output Matrix

- The TCC Output Matrix OTMX=3. (Channel 0 + Remaining Channel 1 Output)

Table 45-5. Output Matrix Channel Pin Routing Configuration

Value	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
0x0	CC1	CC0	CC5	CC4	CC3	CC2	CC1	CC0
0x1	CC1	CC0	CC2	CC1	CC0	CC2	CC1	CC0
0x2	CC0	CC0	CC0	CC0	CC0	CC0	CC0	CC0
0x3	CC1	CC1	CC1	CC1	CC1	CC1	CC1	CC0



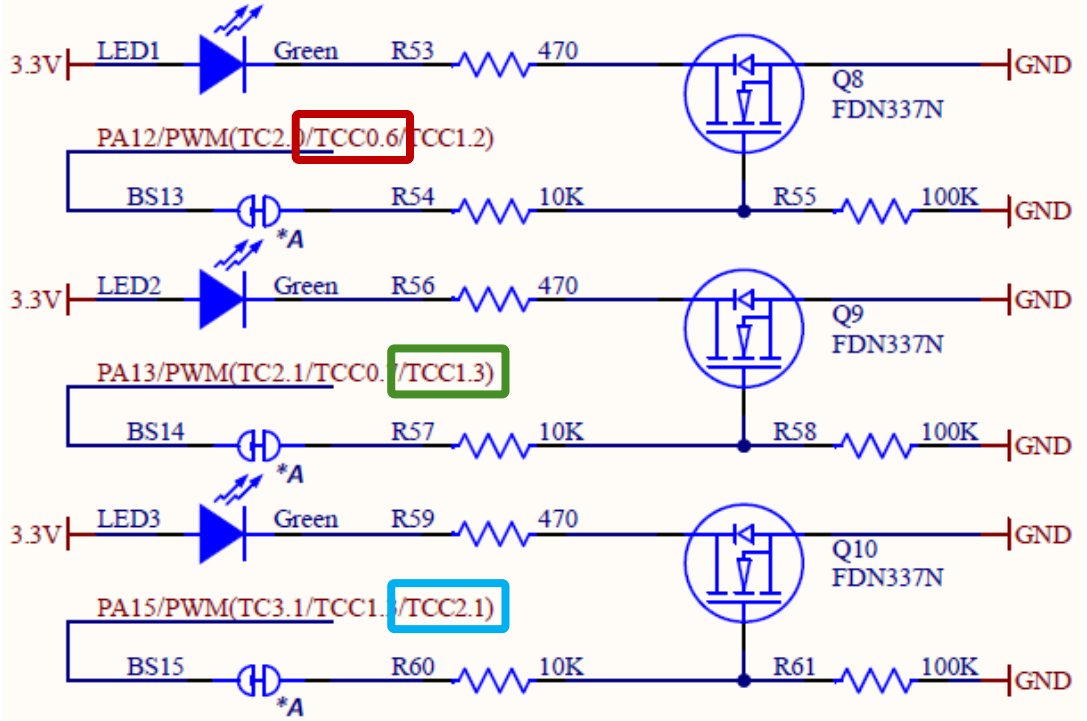
TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
TCC0	Channel	CC_NUM=6	1	1	1	1	1	1	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	1	1	1	1	1	1	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						1	1	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]
TCC3	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]
TCC4	Channel	CC_NUM=2							1	0
	WO	WO_NUM=2							WO[1]	WO[0]

TCC3 and TCC4 do not support Output Matrix

TCC PWM Waveform Output Matrix

- Try output PWM to LEDs by Default TCC Output Matrix as below table.

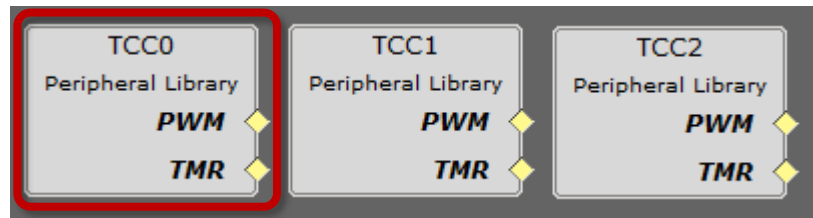
TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
				LED1			LED2		LED3	
TCC0	Channel	CC_NUM=6	1	0	5	4	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	3	2	1	0	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						2	1	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]



TCC PWM Waveform Output Matrix

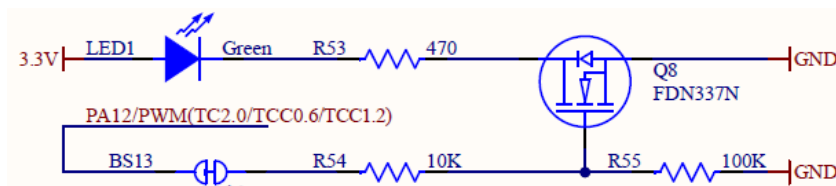
- Try output PWM to LEDs by Default TCC Output Matrix as below table.

TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
				LED1						
TCC0	Channel	CC_NUM=6	1	0	5	4	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	3	2	1	0	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						2	1	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]



Pin Number	Pin ID	Custom Name	Function
60	PA12		TCC0_WO6

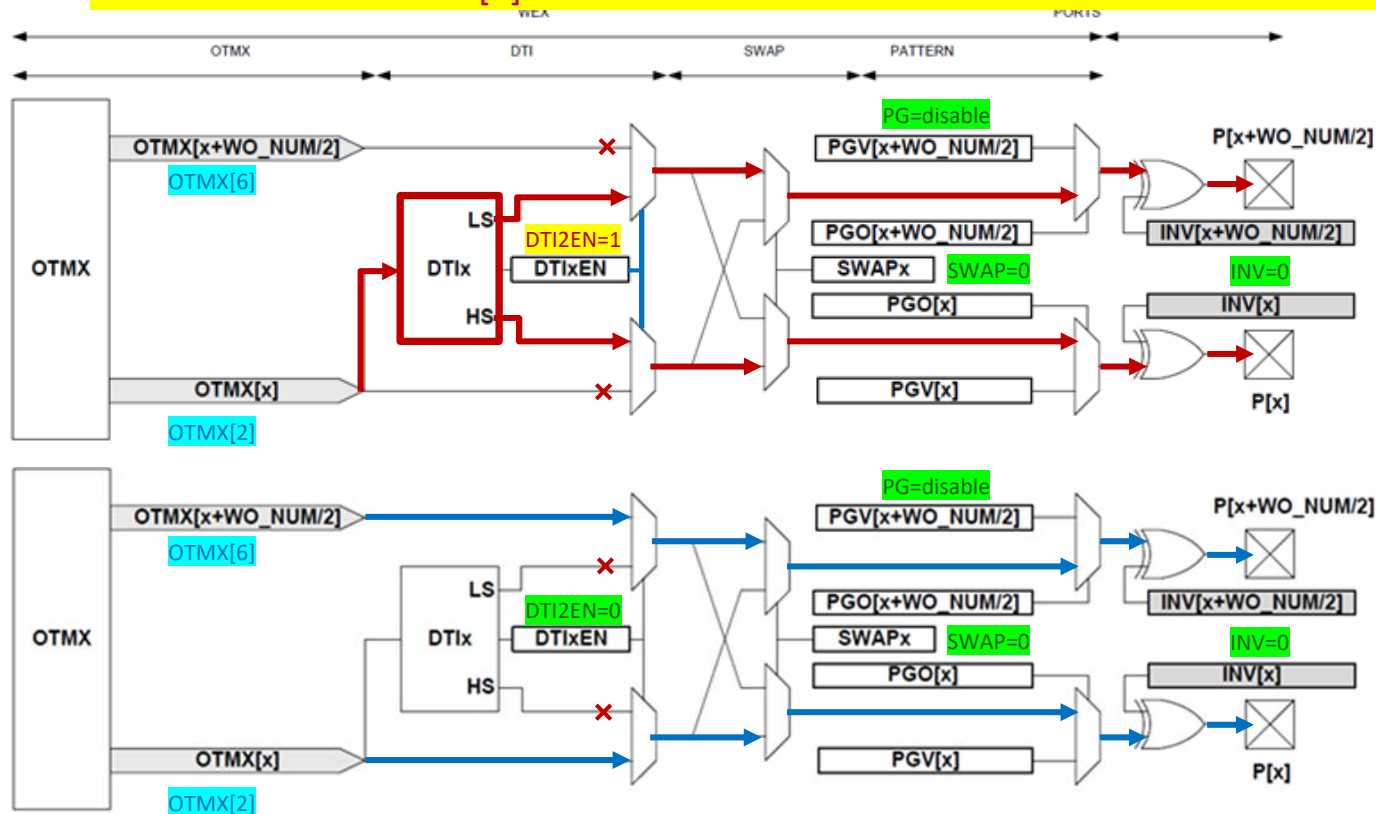
Diagram showing the configuration of TCC0. The 'Channel 2' configuration is highlighted with a red box, showing 'Enable Dead Time' unchecked. A yellow callout box says "uncheck "Enable Deal Time" in Channel 2". Another pink callout box says "uncheck "Enable Deal Time"".



TCC PWM Waveform Output Matrix

- **Dead Time Enable (DTIxEN) and Waveform Output Matrix (OTMX[x])**

- The Dead-Time Insertion (DTI) unit splits the four lower OTMX outputs into two non-overlapping signals.
- The DTI and SWAP units can be seen as a four port pairs slice DTI[x] / SWAP[x] acting on port pins (WO[x], WO[WO_NUM/2 + x]).
 - DTI[0]/SWAP[0] acting on port pins WO[0], WO[4]
 - DTI[1]/SWAP[1] acting on port pins WO[1], WO[5]
 - DTI[2]/SWAP[2] acting on port pins WO[2], WO[6]
 - DTI[3]/SWAP[3] acting on port pins WO[3], WO[7]
- Since LED1 on TCC0 WO[6] then we must disable Dead Time Enable of Channel 2 to avoid the DTI to act the WO[6] output.



Channel 2

Duty Value: 0

Output Polarity: Output is ~DIR and set

Enable Dead Time: ☒

Swap Outputs: ☐

Channel 2

Duty Value: 0

Output Polarity: Output is ~DIR and set

Enable Dead Time: ☐

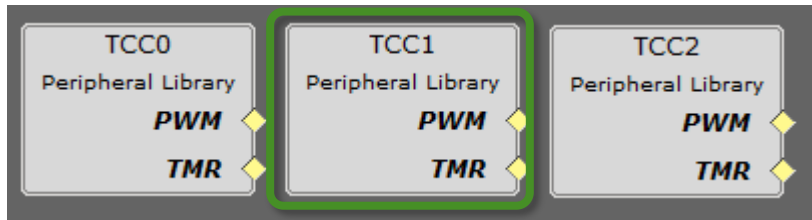
Swap Outputs: ☐

uncheck "Enable Deal Time" in Channel 2

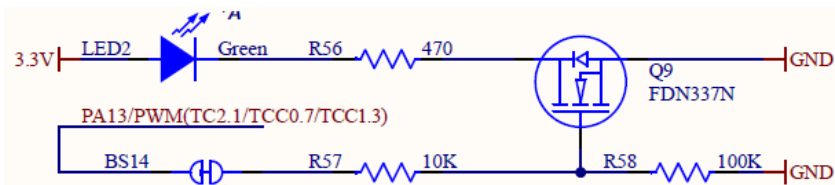
TCC PWM Waveform Output Matrix

- Try output PWM to LEDs by Default TCC Output Matrix as below table.

TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
							LED2			
TCC0	Channel	CC_NUM=6	1	0	5	4	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	3	2	1	0	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						2	1	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]



Pin Number	Pin ID	Custom Name	Function
61	PA13		TCC1_WO3 ▾



TCC1

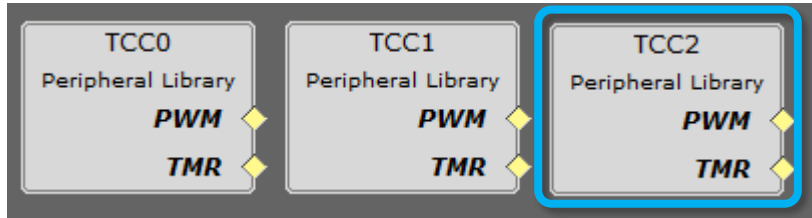
- Enable Slave ☐
- Run during Standby ☐
- Select Prescaler **No division**
- Prescaler and Counter Synchronization **Reload or reset counter on next prescaler clock**
- Operating Mode**
 - PWM**
 - Select PWM Type **NPWM**
 - PWM Direction - Count Down ☐
 - Period Value **59,999**
 - **** PWM Frequency is 1000 Hz ****
 - Enable Period Interrupt ☐
 - Enable Period Event Out ☐
 - Select Output Matrix **Default Channel Outputs**
 - Channel Configurations
 - Channel 0
 - Channel 1
 - Channel 2
 - Channel 3**
 - Duty Value **12,000**
 - Output Polarity **Output is ~DIR and set to DIR when counter matches CCx value**
 - Enable Dead Time** ☐

uncheck "Enable Deal Time"

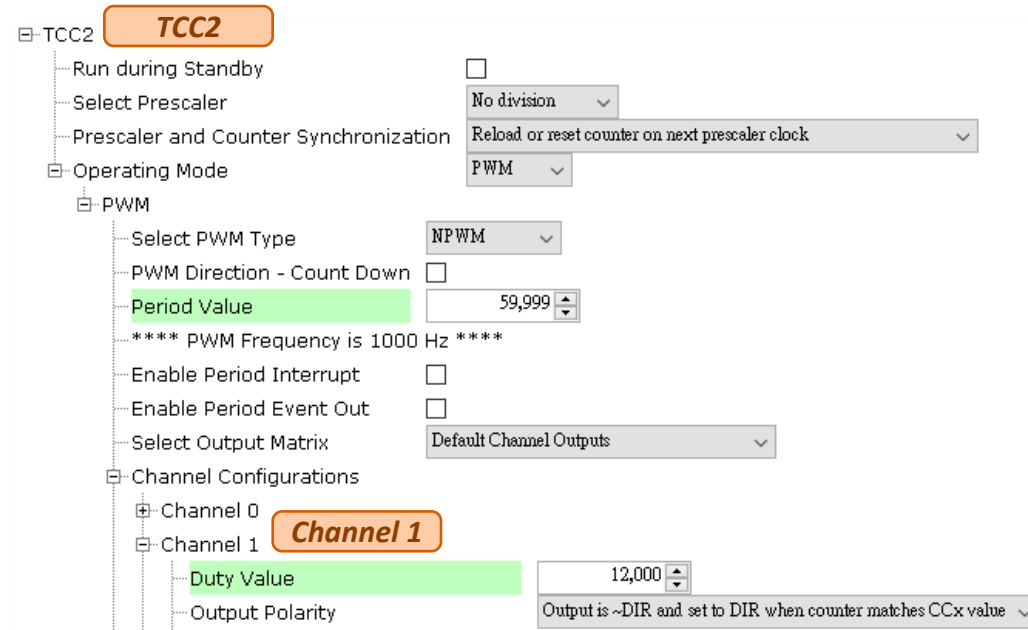
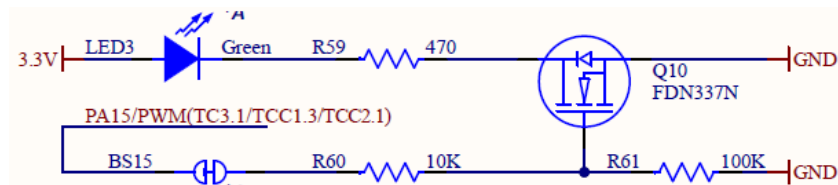
TCC PWM Waveform Output Matrix

- Try output PWM to LEDs by Default TCC Output Matrix as below table.

TCC#	Channe/WO	Numbers	OTMX[7]	OTMX[6]	OTMX[5]	OTMX[4]	OTMX[3]	OTMX[2]	OTMX[1]	OTMX[0]
									LED3	
TCC0	Channel	CC_NUM=6	1	0	5	4	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC1	Channel	CC_NUM=4	3	2	1	0	3	2	1	0
	WO	WO_NUM=8	WO[7]	WO[6]	WO[5]	WO[4]	WO[3]	WO[2]	WO[1]	WO[0]
TCC2	Channel	CC_NUM=3						2	1	0
	WO	WO_NUM=3						WO[2]	WO[1]	WO[0]



Pin Number	Pin ID	Custom Name	Function
63	PA15		TCC2_WO1



TCC PWM Waveform Output Matrix

- Configure WO[x] to LEDs and add TCC Start, Duty Set then remove the LED1 and LED2 GPIO control.

```
int main ( void )
{
    ...
    ADC1_Enable();

    // TODO 13.1.01
    TCC0_PWMStart();
    TCC1_PWMStart();
    TCC2_PWMStart();

    while (true)
    {
        ...
        if (TC0_HasExpired)
        {
            TC0_HasExpired = 0;
            // TODO 13.1.02
            // LED1_Toggle();
            ...
        }

        if (TC1_HasExpired)
        {
            TC1_HasExpired = 0;
            // TODO 13.1.03
            // LED2_Toggle();

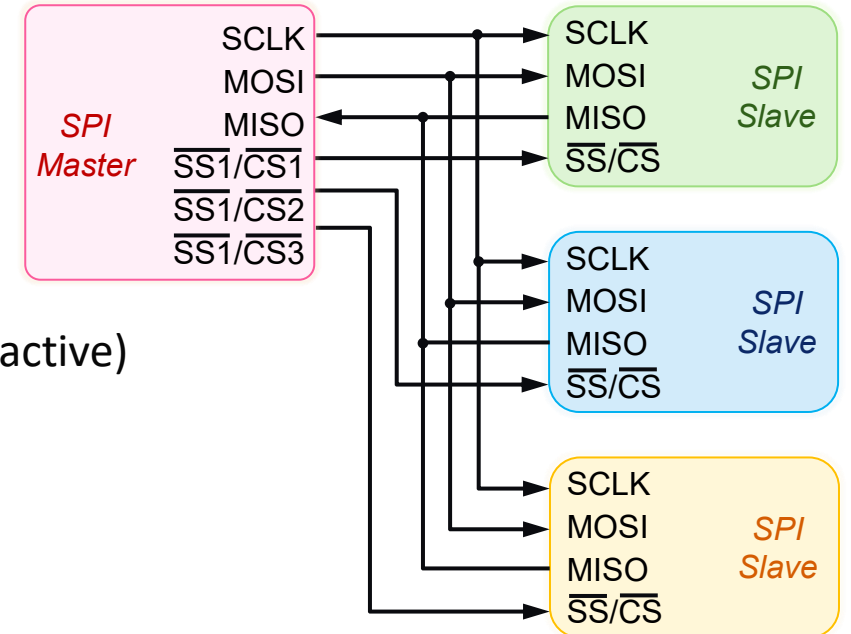
            Duty += DutyDistance;
            if (Duty >= 100 || Duty <= 0)
                DutyDistance = -DutyDistance;
            // TODO 13.1.04
            TCC0_PWM24bitDutySet(TCC0_CHANNEL0, ( (uint32_t)Duty * (TCC0_PWM24bitPeriodGet() + 1) ) / 100 );
            TCC1_PWM24bitDutySet(TCC1_CHANNEL3, ( (uint32_t)Duty * (TCC1_PWM24bitPeriodGet() + 1) ) / 100 );
            TCC2_PWM16bitDutySet(TCC2_CHANNEL1, ( (uint32_t)Duty * (TCC2_PWM16bitPeriodGet() + 1) ) / 100 );
        }
    }
}
```

Pin Number	Pin ID	Custom Name	Function
60	PA12		TCC0_WO6 ✓
61	PA13		TCC1_WO3 ✓
62	PA14		Available ✓
63	PA15		TCC2_WO1 ✓

SERCOM-SPI Architecture

What's SPI

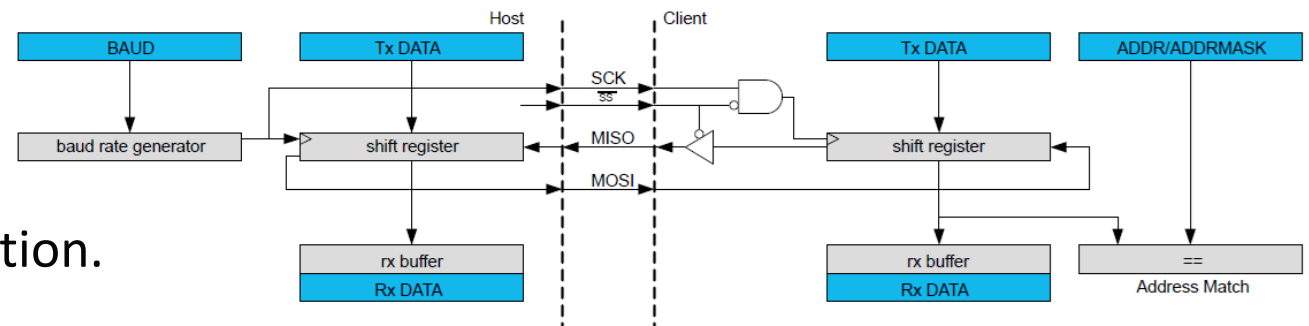
- **SPI (Serial Peripheral Interface)** is a synchronous serial communication protocol used to transfer data between a microcontroller and peripheral devices such as displays, sensors, memory chips, and more.
- **Key Features:**
 - Full duplex: Data can be sent and received at the same time.
 - Master-slave architecture: One master device controls one or more slave devices.
 - High speed: Faster than I²C in most cases.
 - Simple wiring: Typically uses 4 main lines:
 - MOSI (Master Out Slave In): Data sent from master to slave
 - MISO (Master In Slave Out): Data sent from slave to master
 - SCLK (Serial Clock): Clock signal from master
 - $\overline{SS}/\overline{CS}$ (Slave Select/Chip Select): Activates the target slave device, (Low active)



SERCOM-SPI

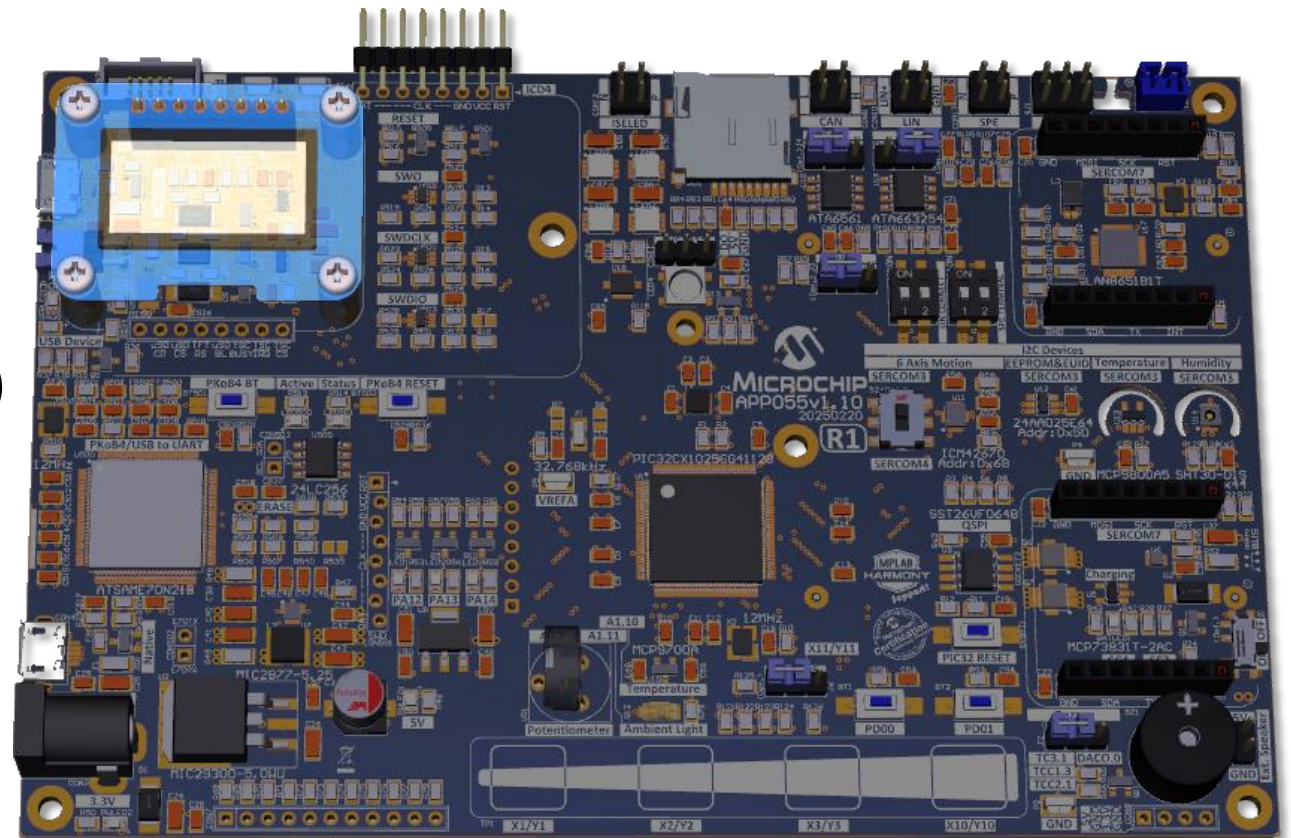
High-Speed Full-Duplex Serial Communication

- Full-duplex 4-wire interface: MISO, MOSI, SCK, SS.
- One-level transmit buffer, two-level receive buffer.
- Supports all four SPI modes (clock polarity and phase).
- MSB/LSB-first selectable data transfer.
- DMA-compatible for efficient data movement.
- 32-bit data path for optimized system bus access.
- Supports single-direction mode for freeing MISO or MOSI pin.
- **Host Mode Features**
 - Programmable serial clock (fSCK) via 8-bit clock generator.
 - Optional inter-character spacing between transfers.
- **Client Mode Features**
 - Operates in all sleep modes.
 - Optional 8-bit address match.
 - Wake-on-SS transition for low-power operation.



SSD1306 128x64 Monochrome OLED Display (SPI Interface)

- The SSD1306 128x64 OLED display is a compact, high-contrast screen commonly used in embedded systems, IoT devices, and portable electronics. It uses organic light-emitting diode (OLED) technology to provide a bright, sharp monochrome display with low power consumption.
- **Key Features:**
 - Resolution: 128x64 pixels
 - Display Type: Monochrome
 - Driver IC: SSD1306
 - Interface: SPI (Serial Peripheral Interface)
 - Screen Size: 0.96 inches
 - Color Depth: 1-bit per pixel (on/off)
 - View Angle: Wide (up to 160°)
 - Backlight: No backlight required



Lab14 OLED Graphics

SERCOM-SPI

Lab14 OLED Graphics - SERCOM-SPI

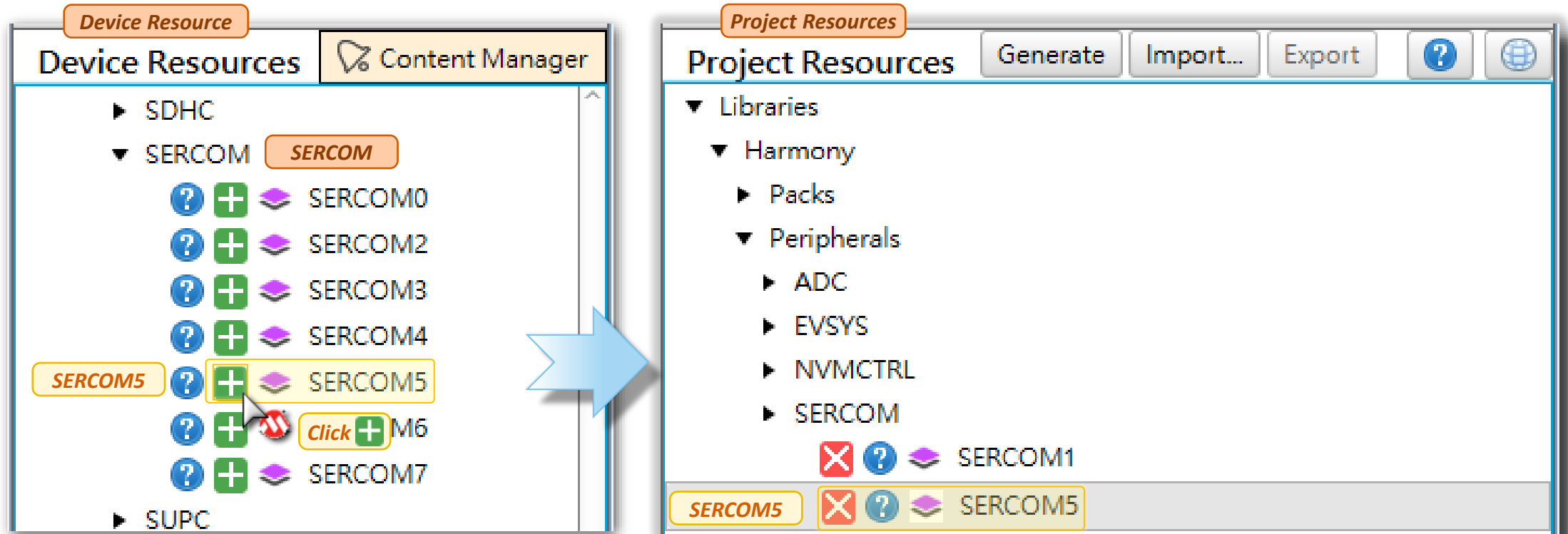
- In this chapter, we will use the **SERCOM-SPI** interface to control the **SSD1306 OLED** display on the APP055 evaluation board, enabling it to display a splash logo animation and real-time peripheral information (such as Timer, ADC, UART, etc.).
- We will provide the **SSD1306 driver** and a **simple graphics library**, which you can directly copy into your project. Just add the provided .c and .h files to your project, then call the available OLED graphics and text rendering functions in your main program.
- Finally, we will introduce an **OLED drawing tool** that allows you to design your own splash image on a PC and convert it into a C header file. This file can then be programmed into the APP055 EVB to display your custom design.

Let's go !

⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 1

- Find **SERCOM5** component in **Device Resources**.
- Click on **+** to add the **SERCOM5** module to your **Project Resources**.



⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 2

- Configure the SERCOM5 **SPI Master**. (SPI Transfer Mode 0)

The screenshot shows the 'Configuration Options' dialog for the SERCOM5 peripheral. The 'SERCOM5' section is expanded, and the 'Select SERCOM Operation Mode' is set to 'SPI Master'. The 'Enable Interrupts ?' checkbox is unchecked, with a 'Disable Interrupt' button next to it. The 'Enable operation in Standby mode' checkbox is also unchecked. The 'SPI Data Out Pad' is set to 'DO on PAD[0], SCK on PAD[1] and SS on PAD[2]'. The 'SPI Data In Pad Selection' is set to 'SERCOM PAD[3] is used as data input'. The 'SPI Data Order' is set to 'MSB is transferred first'. The 'SPI Speed in Hz' is set to '1,000,000'. The 'SPI Data Character Size' is set to '8 bits'. The 'SPI Clock Phase' is set to 'The data is sampled on a leading SCK edge and changed on a trailing SCK edge'. The 'SPI Clock Polarity' is set to 'SCK is low when idle'. The 'SPI Receiver Enable' checkbox is checked. At the bottom, a status message states '***SPI Transfer Mode 0 is Selected***'. Several orange callout boxes provide additional information: 'SPI Master', 'Disable Interrupt', '* Default Setting is Correct for use', 'DO on PAD[0], SCK on PAD[1] and SS on PAD[2]', 'SERCOM PAD[3] is used as data input', '* SPI Input is not used for OLED, but it needs to be separated from Output', '1,000,000 (1MHz)', and 'SPI Transfer Mode 0 is Selected'.

Configuration Options x

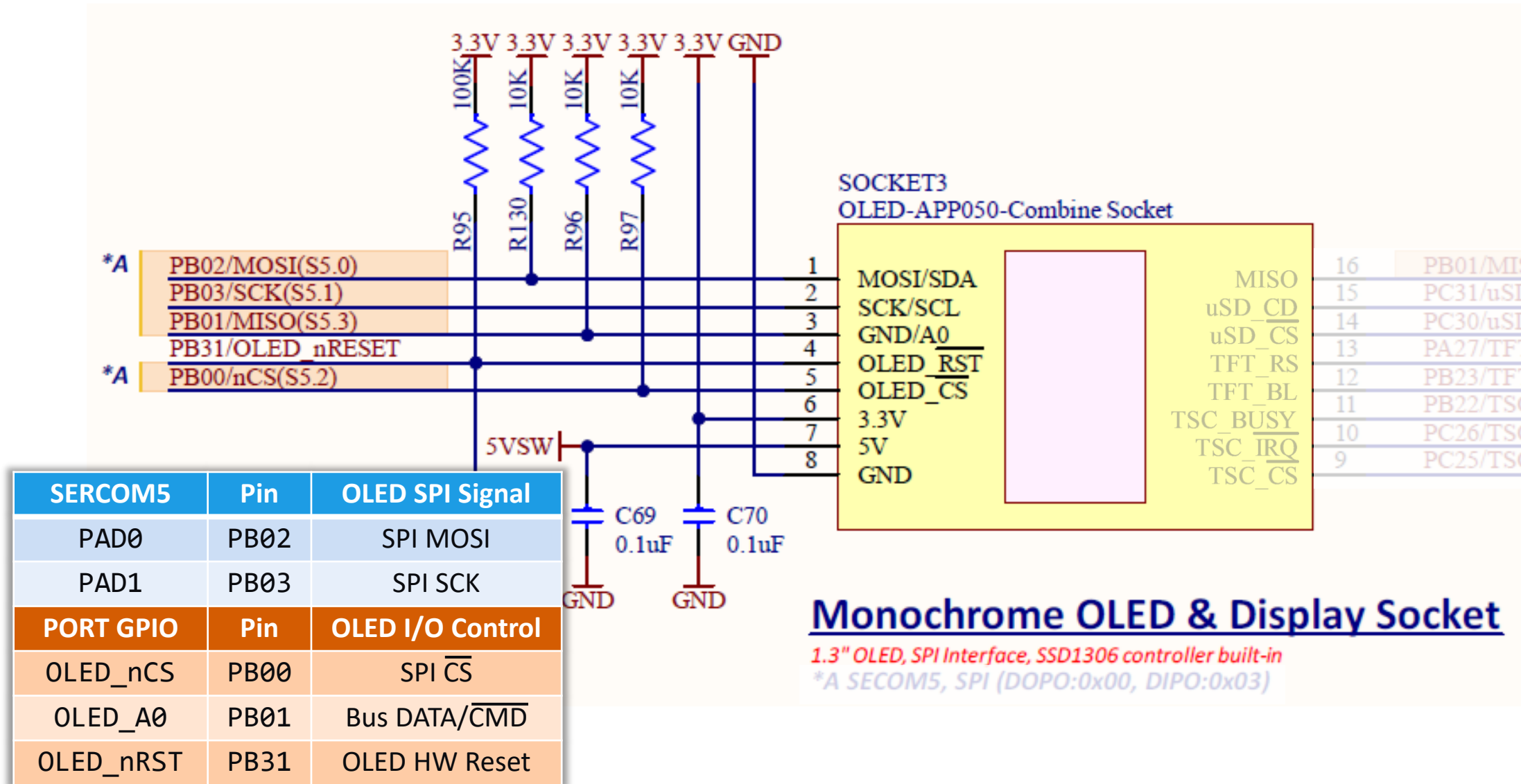
SERCOM5

- Select SERCOM Operation Mode: SPI Master **SPI Master**
- Enable Interrupts ? ☐ **Disable Interrupt**
- Enable operation in Standby mode ☐
- SPI Data Out Pad: DO on PAD[0], SCK on PAD[1] and SS on PAD[2] **DO on PAD[0], SCK on PAD[1] and SS on PAD[2]**
- SPI Data In Pad Selection: SERCOM PAD[3] is used as data input **SERCOM PAD[3] is used as data input**
- SPI Data Order: MSB is transferred first
- SPI Speed in Hz: 1,000,000 **1,000,000 (1MHz)**
- SPI Data Character Size: 8 bits
- SPI Clock Phase: The data is sampled on a leading SCK edge and changed on a trailing SCK edge
- SPI Clock Polarity: SCK is low when idle
- SPI Receiver Enable ☒
- ***SPI Transfer Mode 0 is Selected*** **SPI Transfer Mode 0 is Selected**

** Default Setting is Correct for use*

** SPI Input is not used for OLED, but it needs to be separated from Output*

APP055 OLED SSD1306 Schematic



⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 3

- Pin Settings of the SERCOM5 SPI Master and GPIO pins.

Project Graph x Pin Settings *Pin Settings* Diagram x Pin Table x

Order: **Ports** *Ports* Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
120	PA31		Available	Digital	High Impedance	Low	☐	☐	NORMAL
125 <i>PB00</i>	PB00	OLED_nCS <i>OLED_nCS</i>	GPIO <i>GPIO</i>	Digital	Out <i>Out</i>	High <i>High</i>	☐	☐	NORMAL
126 <i>PB01</i>	PB01	OLED_A0 <i>OLED_A0</i>	GPIO <i>GPIO</i>	Digital	Out <i>Out</i>	Low <i>Low</i>	☐	☐	NORMAL
127 <i>PB02</i>	PB02		SERCOM5_PAD0 <i>SERCOM5_PAD0</i>		High Impedance	n/a	☐	☐	NORMAL
128 <i>PB03</i>	PB03		SERCOM5_PAD1 <i>SERCOM5_PAD1</i>		High Impedance	n/a	☐	☐	NORMAL
122 <i>PB31</i>	PB31	OLED_nRST <i>OLED_nRST</i>	GPIO <i>GPIO</i>	Digital	Out <i>Out</i>	High <i>High</i>	☐	☐	NORMAL

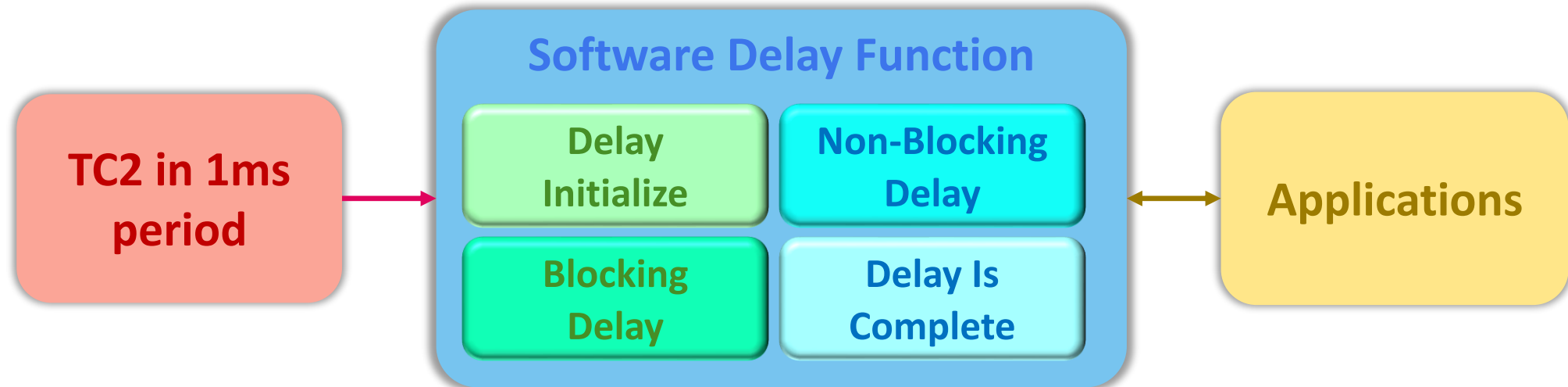
Pin ID	Custom Name	Function	Dir	Latch
PB00	OLED_nCS	GPIO	Out	High
PB01	OLED_A0	GPIO	Out	Low
PB31	OLED_nRST	GPIO	Out	High

Pin ID	Custom Name	Function
PB02		SERCOM5_PAD0
PB03		SERCOM5_PAD1

⚙️ Lab14 OLED Graphics - SERCOM-SPI

TC timer tick based Software Delay

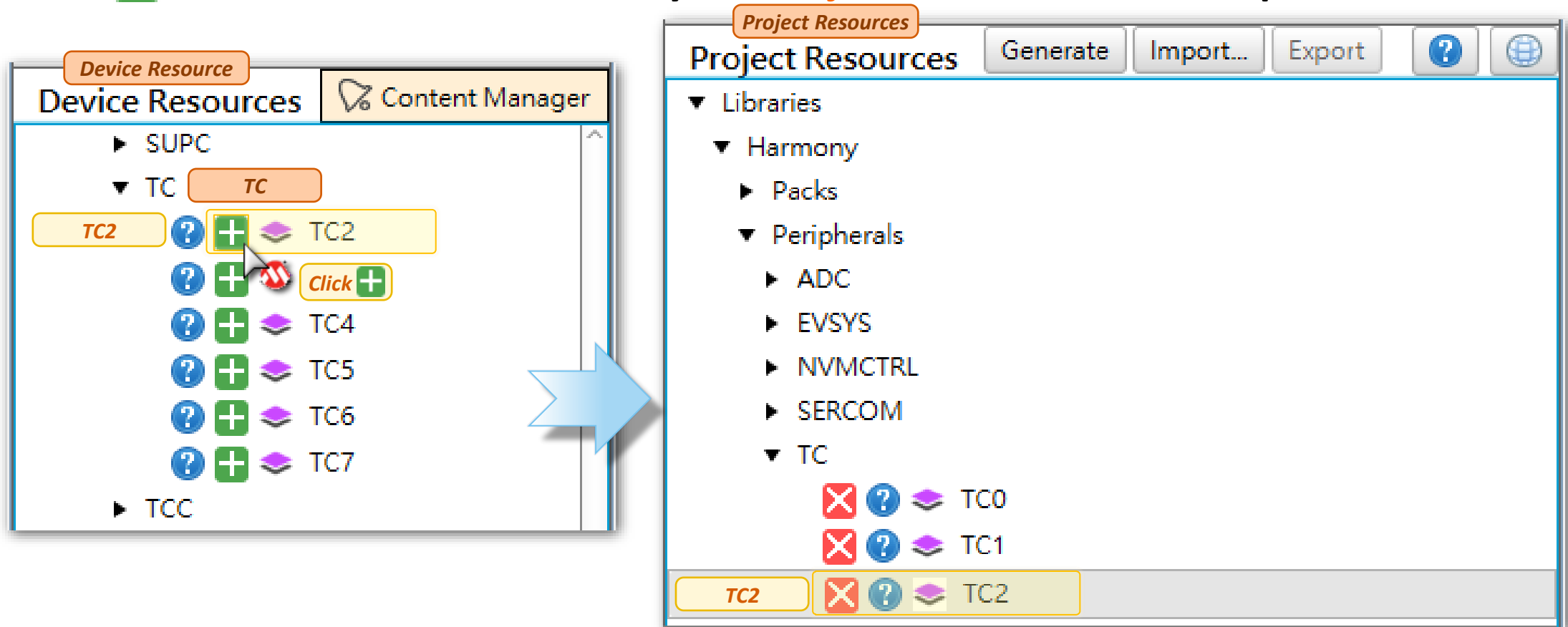
- In the LAB later, we will need to use the **Delay** function. For ease of use, we have pre-designed a **Software Delay function** that supports both **Blocking** and **Non-Blocking** modes. This function will use the **TC2** module with **1ms** as the **tick base** for timing.



✖ Lab14 OLED Graphics - SERCOM-SPI

Step 4

- Find **TC2** component in **Device Resources**, we use it for tick base of a **software delay**.
- Click on **+** to add the **TC2** module to your **Project Resources** and keep it as default.



⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 5

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Project Resources' tab. The 'Generate' button is highlighted with a yellow box and a red arrow. Below it, the 'System' resource is also highlighted. The 'Output' window is open, showing the 'MPLAB® Code Configurator' tab. The output text indicates that the generation process is complete, with a total time of 5396 milliseconds. The output also shows the execution of the 'generate hook' and the 'preGenerateEvent', 'interfaceValidationEvent', 'processDataModel', and 'postFinalProcessing' events.

Generation complete

```
15:17:50.827 INFO: [p10 (p1032CX1025SG41128)] Executing generate hook
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

SERCOM-SPI Master

Interface Functions

Block Mode (Interrupt Disabled)

- **SERCOM5_SPI_Initialize** : Initializes instance SERCOM5 operating in SPI mode.
- **SERCOM5_SPI_TransferSetup** : Configure SERCOM SPI operational parameters at run time.
- **SERCOM5_SPI_Write** : Writes data to SERCOM5 SPI peripheral.
- **SERCOM5_SPI_Read** : Reads data on the SERCOM5 SPI peripheral.
- **SERCOM5_SPI_WriteRead** : Write and Read data on SERCOM5 SPI peripheral.
- **SERCOM5_SPI_IsTransmitterBusy** : Returns hardware transfer status of the SPI transmit shift register.

Non-Blocking Mode (Interrupt Enabled)

- **SERCOM5_SPI_CallbackRegister** : Allows application to register callback with PLIB.
- **SERCOM5_SPI_IsBusy** : Returns transfer status of SERCOM5 SPI.

SERCOM-SPI Master Blocking Mode (Interrupt Disabled)

The interface functions we will use in this Lab

bool SERCOM5_SPI_Write(void* pTransmitData, size_t txSize)

- Description:
 - This function writes "txSize" number of bytes on SERCOM5 SPI module. Data pointed by pTransmitData is transmitted.
- Parameter:
 - pTransmitData : Pointer to the buffer containing the data which has to be transmitted.
For 9-bit mode, data should be right aligned in the 16 bit memory location.
 - txSize : Number of bytes to be transmitted. For 9-bit mode, 2 bytes make up a count of 1.

bool SERCOM5_SPI_IsTransmitterBusy(void)

- Description:
 - Returns hardware transfer status of the SPI transmit shift register.

- Example:

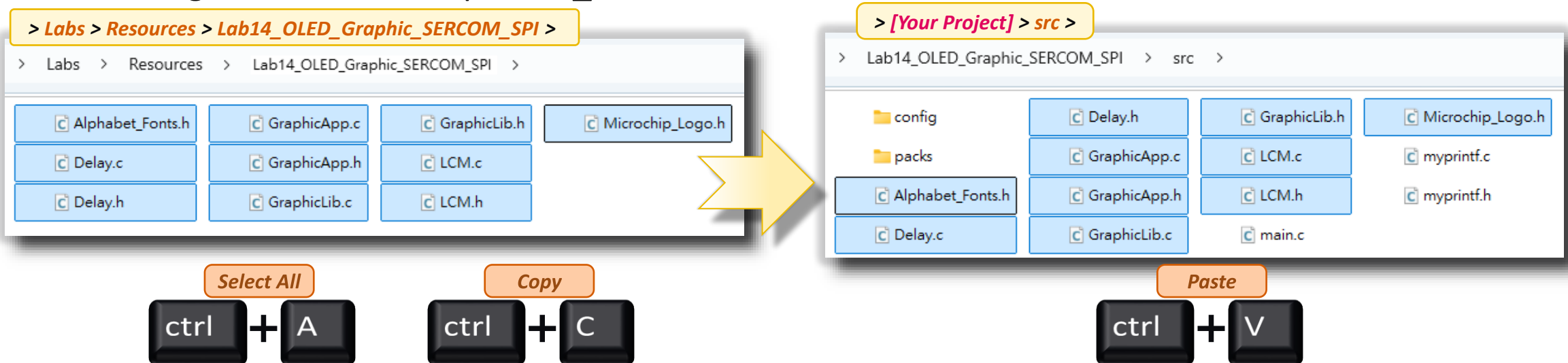
```
uint8_t txBuffer[4];  
SERCOM5_SPI_Write( &txBuffer, 4 );  
While( SERCOM5_SPI_IsTransmitterBusy() ) {}
```

⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 6

- Copy all pre-developed files in `\Labs\Resource\Lab14_OLED_Graphic_SERCOM_SPI\` to the project folder `\[Your Project]\src\`

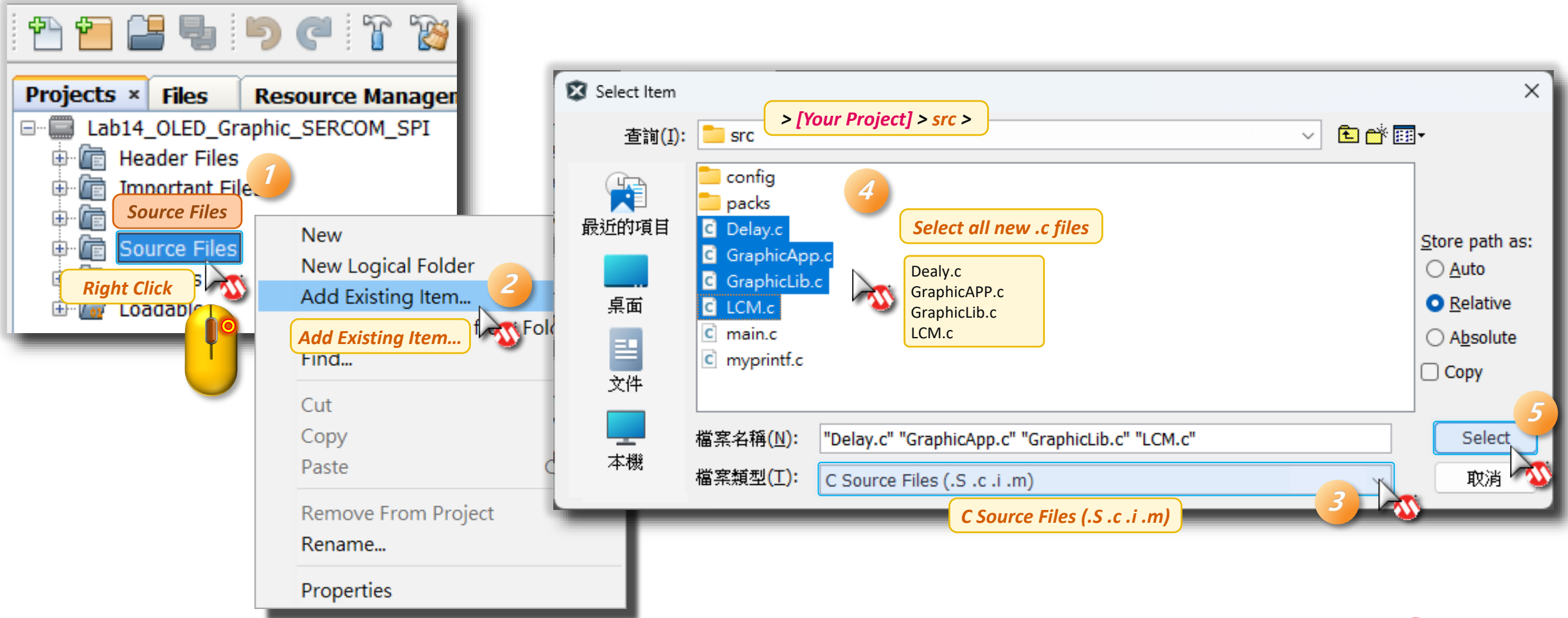
- Delay Function : Delay.c Delay.h
- OLED display driver : LCM.c LCM.h
- Graphic Library : GraphicLib.c GraphicLib.h
- Graphic Application : GraphicApp.c GraphicApp.h
- Splash Logo image : Microchip_Logo.h
- Fonts image : Alphabet_Fonts.h



⚡ Lab14 OLED Graphics - SERCOM-SPI

Step 7

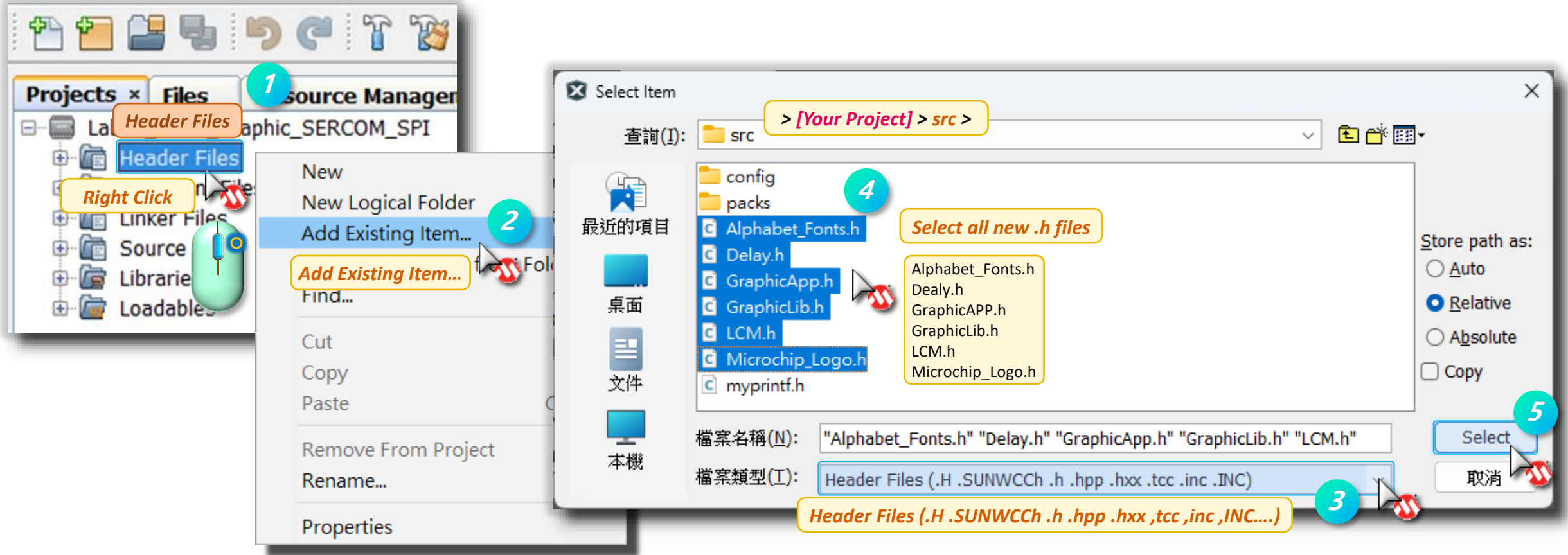
- In MPLAB X IDE, add new **Source Files (*.c)** under **\[Your Project]\src** to your project.



⚡ Lab14 OLED Graphics - SERCOM-SPI

Step 8

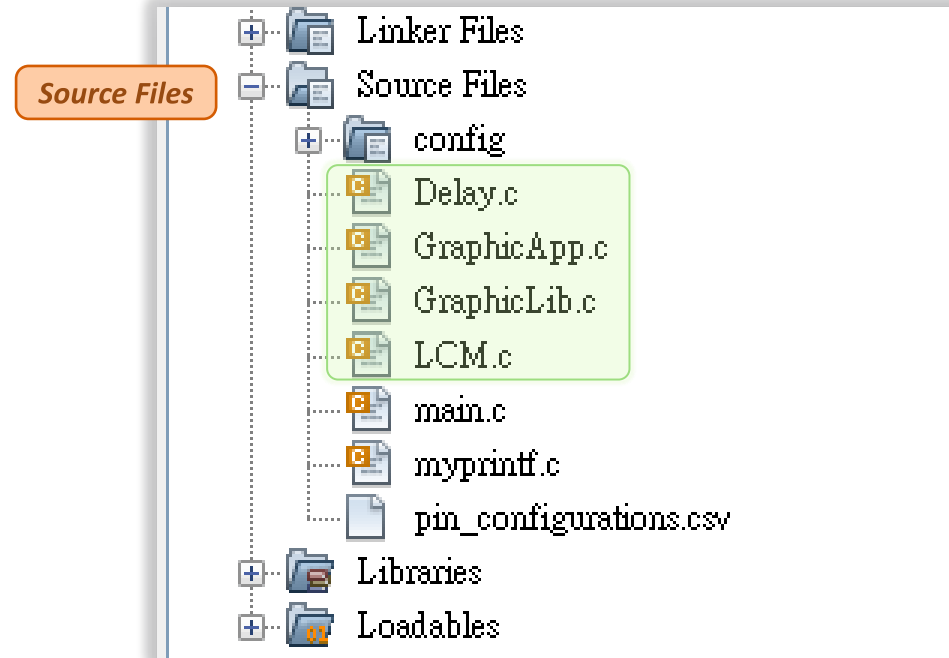
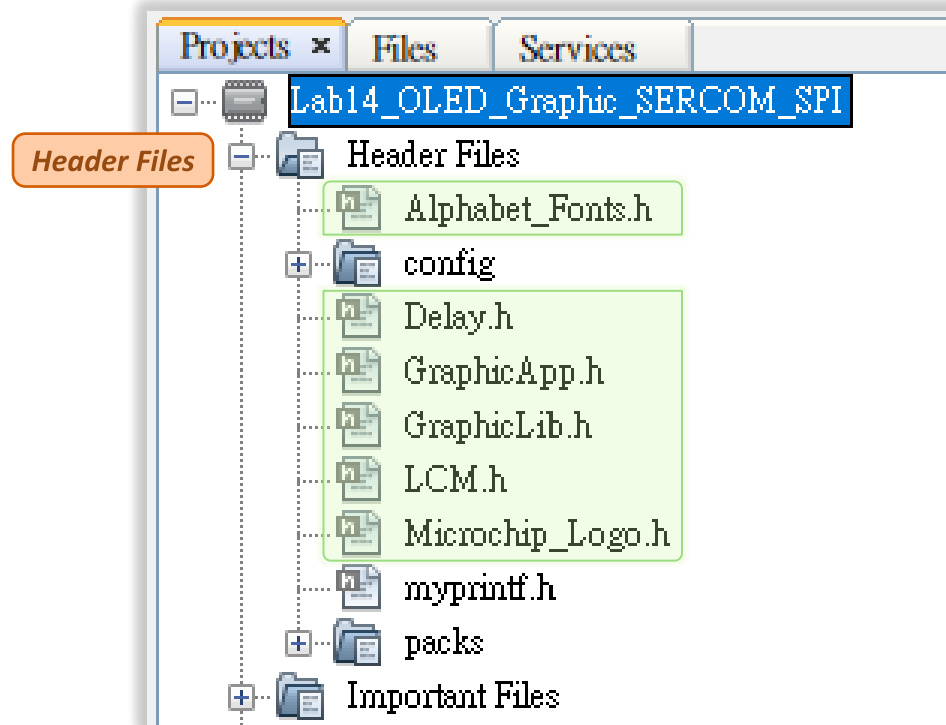
- In MPLAB X IDE, add new Header Files (*.h) under \[Your Project]\src\ to your project.



⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 9

- In MPLAB X IDE, check below files been added to your project.



OLED Graphic Application and Library

Pre-designed drawing applications and library

void DrawSplashAnimation(int x0, int y0, int x1, int y1, uint8_t step, const unsigned char *bitmap, uint8_t layer)

- Description:
 - Animates a bitmap moving from (x0, y0) to (x1, y1) on the specified graphic layer using defined step size.
- Parameter:
 - x0, y0 : Starting coordinates of the animation.
 - x1, y1 : Ending coordinates of the animation.
 - step : Step size per frame (pixels).
 - bitmap : Pointer to bitmap image data array buffer.
 - layer : Graphic layer to display the animation.
- Example:
DrawSplashAnimation(0, 0, 64, 32, 4, splash_logo, LAYER_GRAPHIC);

GarphicApp.c

** Appendix for more functions*

void GPL_Printf(uint16_t x, uint16_t y, GPL_BACKGROUND background, GPL_TEXT highlight, const char *format, ...)

- Description:
 - Print formatted text on screen at specified position.
- Parameter:
 - x, y : Position to print the text.
 - background : Background pixel (BG_TRANSPARENT or BG_SOLID).
 - highlight : Text Highlight (TEXT_NORMAL or TEXT_HIGHLIGHT).
 - format : Text format string (like printf).
- Example:
GPL_Printf(67, 38, BG_SOLID, TEXT_NORMAL, "Microchip");
GPL_Printf(67, 50, BG_SOLID, TEXT_HIGHLIGHT, "GPL Lib v%d.%d", 1, 2);

GarphicLib.c

** Appendix for more functions*

OLED Graphic Application and Library

Pre-designed software delay function by using TC timer as the tick

```
/* ***** */
/* ***** */
/* Section: File Scope or Global Data */
/* ***** */
/* ***** */
// Timer PLIB function Name (SAM/PIC32C only) in 1ms period
#define TC_PLIB_CALLBACK(...) TC2_TimerCallbackRegister(__VA_ARGS__)
#define TC_PLIB_START(...) TC2_TimerStart(__VA_ARGS__)
```

Delay.c

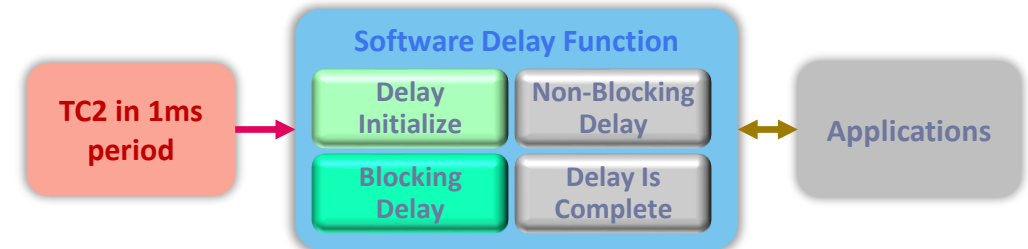
void DelayInitial(void)

- Description:
 - Initializes the delay timer system, Configure the TC timer and prepare delay tracking

void DelayMS(uint8_t idx, uint32_t ms)

- Description:
 - Performs a blocking delay for the given duration.
- Parameter:
 - Idx : Timer index.
 - ms : Delay duration in milliseconds.
- Example:

```
#define MY_DELAY_IDX 0
DelayMS( MY_DELAY_IDX , 1000 ); // Blocking delay of 1 second of Timer Index MY_DELAY_IDX
```



OLED Graphic Application and Library

Pre-designed software delay function by using TC timer as the tick

void DelayMS_NonBlock(uint8_t idx, uint32_t ms)

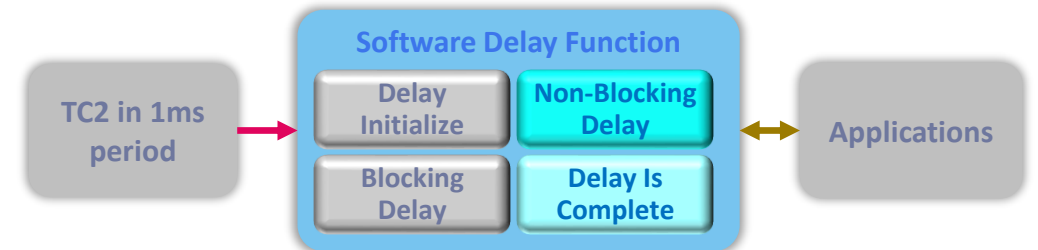
- Description:
 - Starts a non-blocking delay timer.
- Parameter:
 - idx : Timer index.
 - ms : Delay duration in milliseconds.
- Example:

```
#define MY_TIMER_IDX 1  
DelayMS_NonBlock( MY_TIMER_IDX, 500 ); // 500ms non-blocking delay of Timer Index MY_TIMER_IDX
```

bool DelayIsComplete(uint8_t idx)

- Description:
 - Checks if non-blocking delay has finished.
- Parameter:
 - idx : Timer index to check.
- Return:
 - bool : true or false – if return true, the delay for the given channel is done
- Example:

```
if( DelayIsComplete( MY_TIMER_IDX ) )  
{ // Do something after delay }
```



🔧 Lab14 OLED Graphics - SERCOM-SPI

Step 10

- Open **LCM.c**, and check **line 55** to see how **SERCOM5** and **GPIOs** are used in the LCM driver.

Line 55

LCM.c

```
/* ***** */
/* ***** */
/* Section: File Scope or Global Data */
/* ***** */
/* ***** */
// APP055 SERCOM5_SPI Pin Configuration
// SERCOM5 SPI : MOSI (PB02)
// SERCOM5 SPI : SCK (PB03)
// OLED_nCS : SS (PB00), GPIO Output Latch high
// OLED_nRST : Reset(PB31), GPIO Output Latch high
// OLED_A0 : A0 (PB01), GPIO Output Latch low (Command:0, Data:1)
#define LCM_HW_RESET_ENABLE 1 // Enable OLED Reset PIN on PB31
#define LCM_SPI_CS_GPIO 1 // Enable SPI GPIO Slave Select on PB00
#define LCM_SPI_Write(...) SERCOM5_SPI_Write(__VA_ARGS__)
#define LCM_SPI_IsWriteBusy(...) SERCOM5_SPI_IsTransmitterBusy(__VA_ARGS__)
#define LCM_RESET_Enable(...) OLED_nRST_Clear(__VA_ARGS__)
#define LCM_RESET_Disable(...) OLED_nRST_Set(__VA_ARGS__)
#define LCM_BUS_CMD(...) OLED_A0_Clear(__VA_ARGS__)
#define LCM_BUS_DAT(...) OLED_A0_Set(__VA_ARGS__)

#if LCM_SPI_CS_GPIO
#define LCM_SPI_CS_Enable(...) OLED_nCS_Clear(__VA_ARGS__)
#define LCM_SPI_CS_Disable(...) OLED_nCS_Set(__VA_ARGS__)
#endif

uint8_t LCM_FrameBuf[LCM_FRAME_SIZE]; // (128x64)/8, 1bit/pixel
uint8_t *LCM_pFrameBuf = NULL; // The Frame Buffer Pointer
const uint8_t LCM_InitCMD[]={ ... };
```

Pin ID	Custom Name	Function
PB02		SERCOM5_PAD0
PB03		SERCOM5_PAD1

Pin ID	Custom Name	Function	Dir	Latch
PB00	OLED_nCS	GPIO	Out	High
PB01	OLED_A0	GPIO	Out	Low
PB31	OLED_nRST	GPIO	Out	High

⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 11

- Open **main.c** and finish **TODO 14.01** to **14.06** for Splash Logo and data shows on OLED.

```
#include <stddef.h>           // Defines NULL
#include <stdbool.h>          // Defines true
#include <stdlib.h>           // Defines EXIT_FAILURE
#include <string.h>
#include <stdio.h>
#include "definitions.h"      // SYS function prototypes
#include "myprintf.h"
// TODO 14.01
#include "Delay.h"
#include "GraphicLib.h"
#include "GraphicApp.h"
#include "Microchip_Logo.h"

uint32_t i = 0;
uint8_t USART1_TransmitData[100];
uint8_t USART1_ReceiveData[1];
volatile uint8_t USART1_IsReceived = 0;
volatile uint8_t USART1_IsTransmitted = 1;
volatile uint8_t TC0_HasExpired = 0;
volatile uint8_t TC1_HasExpired = 0;
uint16_t ADC0_Result;
uint16_t ADC1_Result[2];
volatile uint8_t ADC0_IsCompleted = 0;
volatile uint8_t ADC1_IsCompleted = 0;
volatile int8_t ADC1_ChannelIdx = 0;
uint8_t Duty = 50;
int8_t DutyDistance = 2;

void ADC0_Complete( ADC_STATUS status, uintptr_t context )
```

main.c

⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 12

- 1st part of **TODO 14.02** is for LCM and Graphic initial and the Splash Logo animation.

```
int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );
    ...

    TCC1_PWMStart();

    // TODO 14.02
    // OLED HW and Graphic Layers Initial and Clean all Frame Buffer
    GPL_ScreenInit();

    // Show Only Graphic Layers
    GPL_LayerShow( LAYER_GRAPHIC, LAYER_SHOW );
    GPL_LayerShow( LAYER_TEXT,    LAYER_HIDE );

    // Splash Logo In
    DrawSplashAnimation( 128, 0, 0, 0, 4, Microchip_Logo, LAYER_GRAPHIC );

    // Delay after animation ends
    DelayMS(DELAY_TIMER_OLED_DELAY, 1000);

    // Splash Logo Out
    DrawSplashAnimation( 0, 0, -128, 0, 8, Microchip_Logo, LAYER_GRAPHIC );

    // Show Both Text and Graphic Layers
    GPL_LayerShow( LAYER_GRAPHIC, LAYER_SHOW );
    GPL_LayerShow( LAYER_TEXT,    LAYER_SHOW );
}
```

⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 13

- 2nd part of **TODO 14.02** is to prepare the System Clock, TC0, TC1 and UART read text.

```
// TODO 14.02

...

// Draw on Text Layer
GPL_LayerSet( LAYER_TEXT );

// Clean Text Layer Buffer
GPL_LayerClean( LAYER_TEXT );
GPL_Printf( 0, 0, BG_SOLID, TEXT_NORMAL, "SysClock  : %dMHz", CPU_CLOCK_FREQUENCY/1000000 );
GPL_Printf( 0, 8, BG_SOLID, TEXT_NORMAL, "TC0 period:  %ldms",
                                                    (TC0_Timer16bitPeriodGet()+1)*1000 / TC0_TimerFrequencyGet());
GPL_Printf( 0, 16, BG_SOLID, TEXT_NORMAL, "TC1 period:  %ldms",
                                                    (TC1_Timer16bitPeriodGet()+1)*1000 / TC1_TimerFrequencyGet());
GPL_Printf( 0, 56, BG_SOLID, TEXT_NORMAL, "Received Data :");
GPL_ScreenUpdate();

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

    if (TC0_HasExpired)
    {
        TC0_HasExpired = 0;
        LED1_Toggle();

        myprintf( "@1,1;Hello World!!" );
        ADC0_ConversionStart();
    }
}
```


⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 14

- The **TODO 14.03** for TCC1 PWM Duty % and **TODO 14.04** to update UART received char.

```
while ( true )
{
    ...

    if (TC1_HasExpired)
    {
        TC1_HasExpired = 0;
        LED2_Toggle();

        Duty += DutyDistance;
        if (Duty >= 100 || Duty <= 0)
            DutyDistance = -DutyDistance;

        TCC1_PWM24bitDutySet(TCC1_CHANNEL3, (((uint32_t) Duty * TCC1_PWM24bitPeriodGet() + 1) / 100));
// TODO 14.03
        GPL_Printf( 0, 24, BG_SOLID, TEXT_NORMAL, "PWM Duty : %3d%%", Duty);
        GPL_ScreenUpdate();
    }

    if (USART1_IsReceived)
    {
        USART1_IsReceived = 0;

        myprintf( "@1,10;Received Data : %1c", USART1_ReceiveData[0] );
        SERCOM1_USART_Read(USART1_ReceiveData, 1);
// TODO 14.04
        GPL_Printf( 0, 56, BG_SOLID, TEXT_NORMAL, "Received Data : %c", USART1_ReceiveData[0]);
        GPL_ScreenUpdate();
    }
```

Normally, we use the TC1 timer to update the screen at a period of 40ms

Update the screen immediately when UART receives data

⚙️ Lab14 OLED Graphics - SERCOM-SPI

Step 15

- The **TODO 14.05** for ADC0 VR1 value and **TODO 14.06** for ADC1 Temp/LightSens values.

```
while ( true )
{
    ...

    if (TC1_HasExpired)
    { ... }

    if (USART1_IsReceived)
    { ... }

    if( ADC0_IsCompleted )
    {
        ADC0_IsCompleted = 0;
        myprintf( "@1,2;VR1 Value : %4d", ADC0_Result );

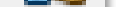
// TODO 14.05
        GPL_Printf( 0, 32, BG_SOLID, TEXT_NORMAL, "VR1    : %4d", ADC0_Result);
    }

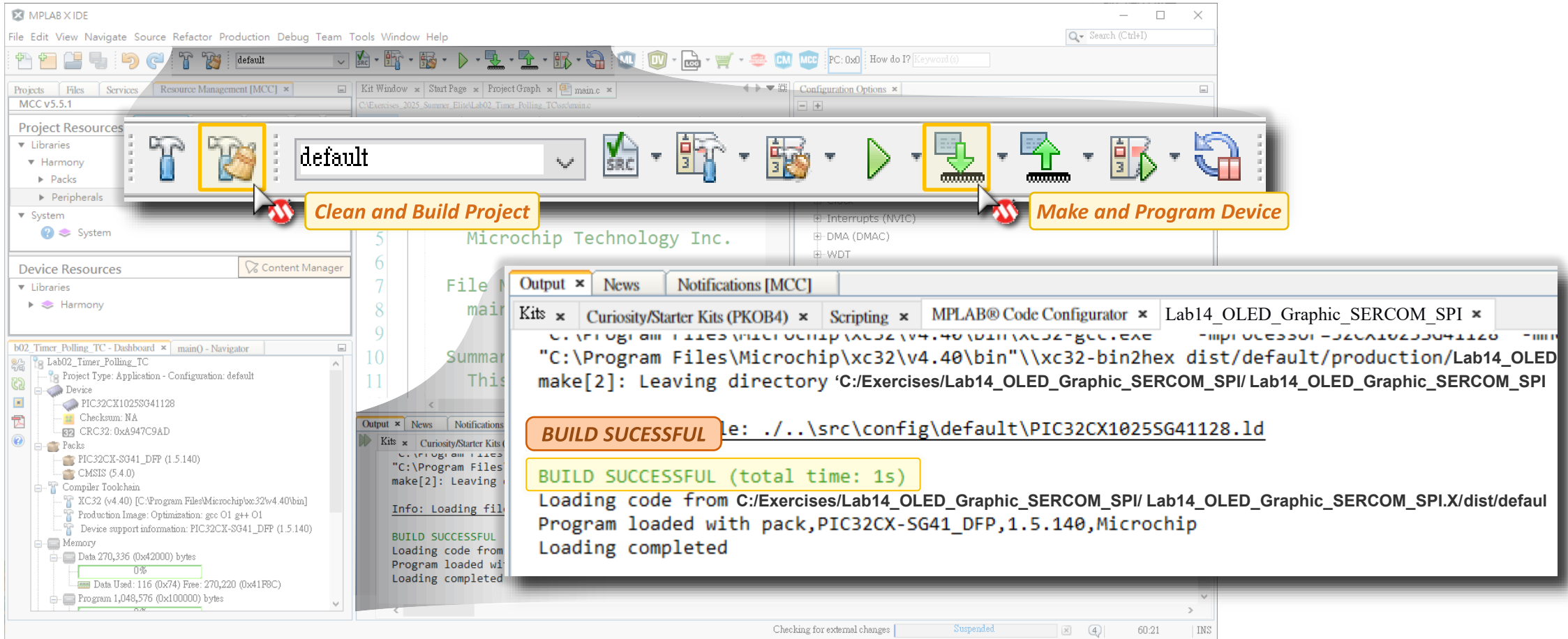
    if( ADC1_IsCompleted )
    {
        ADC1_IsCompleted = 0;
        myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );
        myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );

// TODO 14.06
        GPL_Printf( 0, 40, BG_SOLID, TEXT_NORMAL, "Temp  : %4d", ADC1_Result[0]);
        GPL_Printf( 0, 48, BG_SOLID, TEXT_NORMAL, "Light : %4d", ADC1_Result[1]);
    }
}
```

Lab14 OLED Graphics - SERCOM-SPI

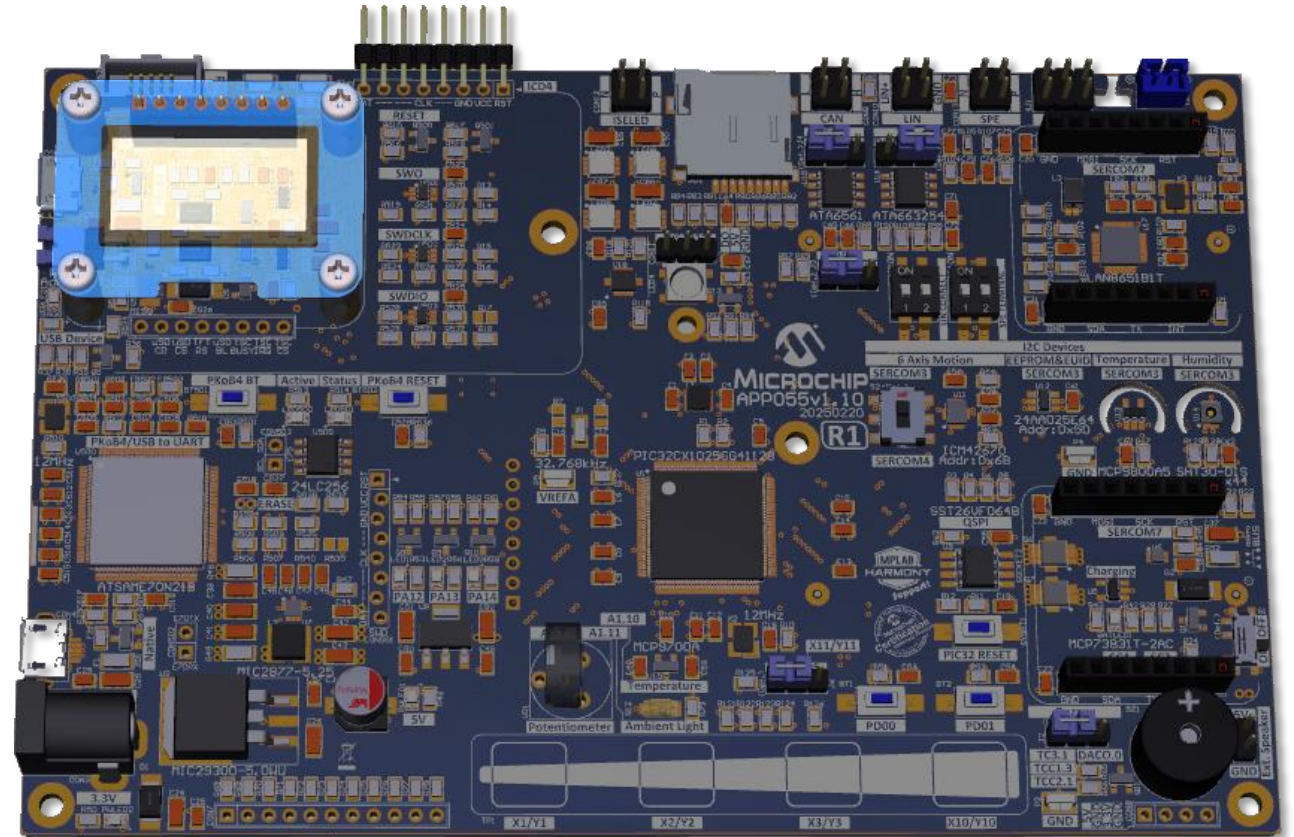
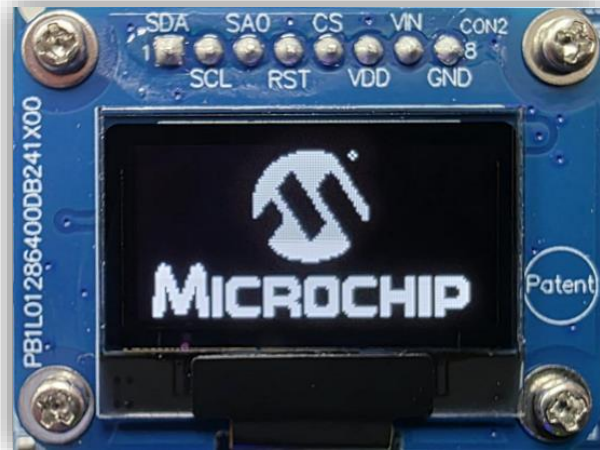
Step 16

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



⚡ Lab14 OLED Graphics - SERCOM-SPI

Result



Lab15 Design Your Own Logo

OLED Graphic

✂ Lab15 Design Your Own Logo

- In this chapter, we will use a software named **OLED Assistant**, a computer graphics editing application developed by **Microchip**, to draw graphics on the OLED.
- You can use existing graphics files to import and edit them or use your creativity to design patterns freely.
- You can also preview the effect of the county on the OLED in the graphics software and finally export it to a **C language Header File** and then download it to the current Lab to replace the original Microchip Logo with your own design.

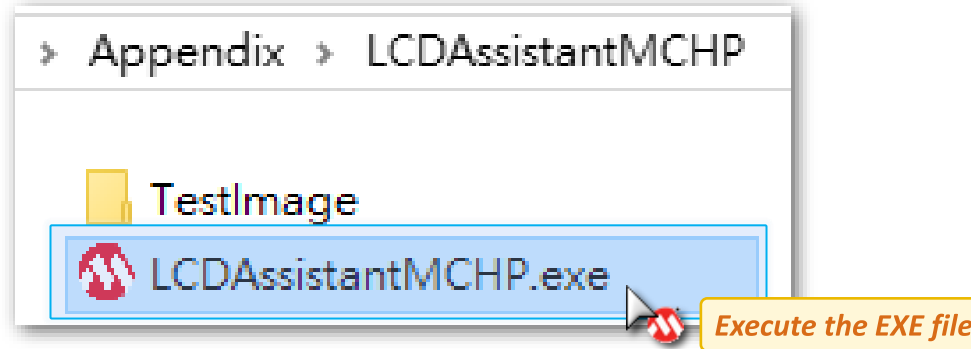


Let's go !

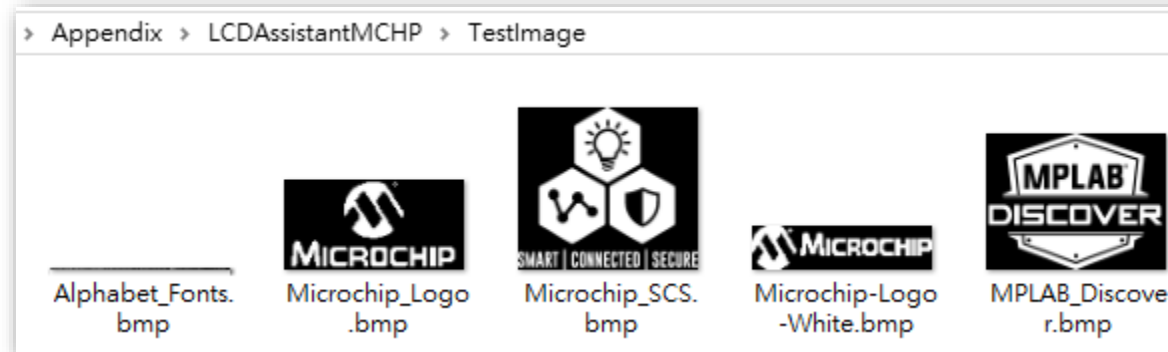
Lab15 Design Your Own Logo

Step 1

- Go to the folder \Appendix\LCDAssistantMCHP\
 - Execute the EXE file LCDAssistantMCHP.exe



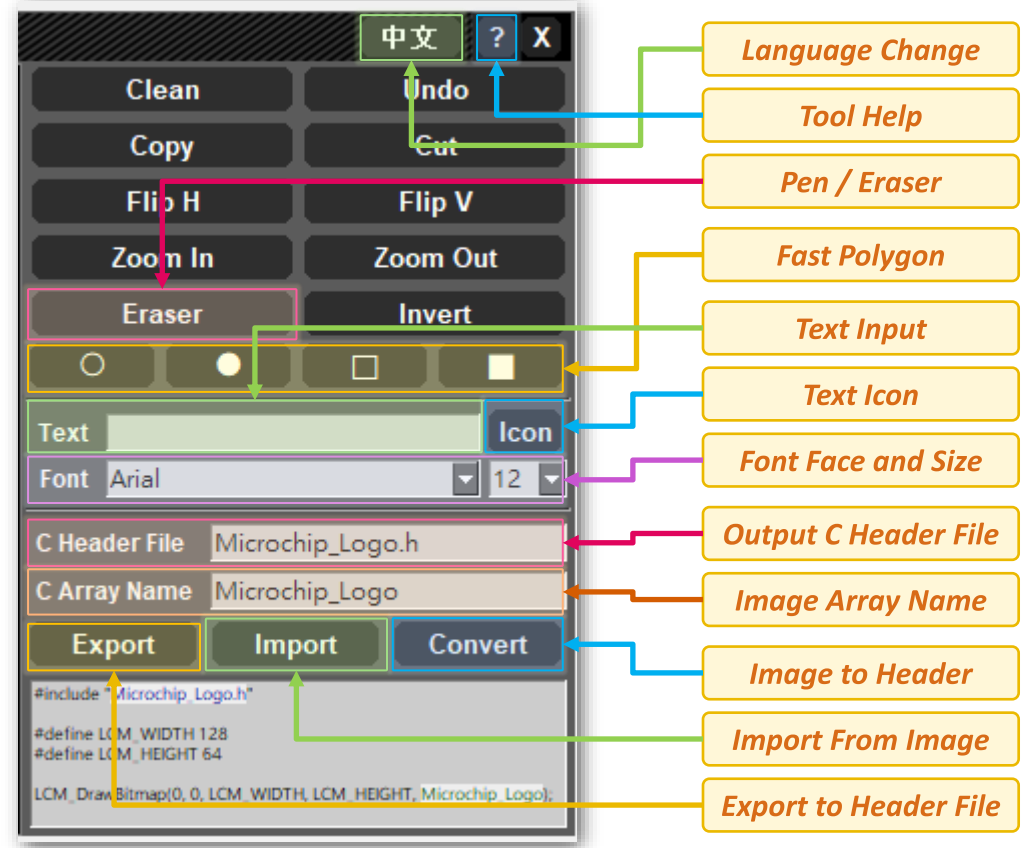
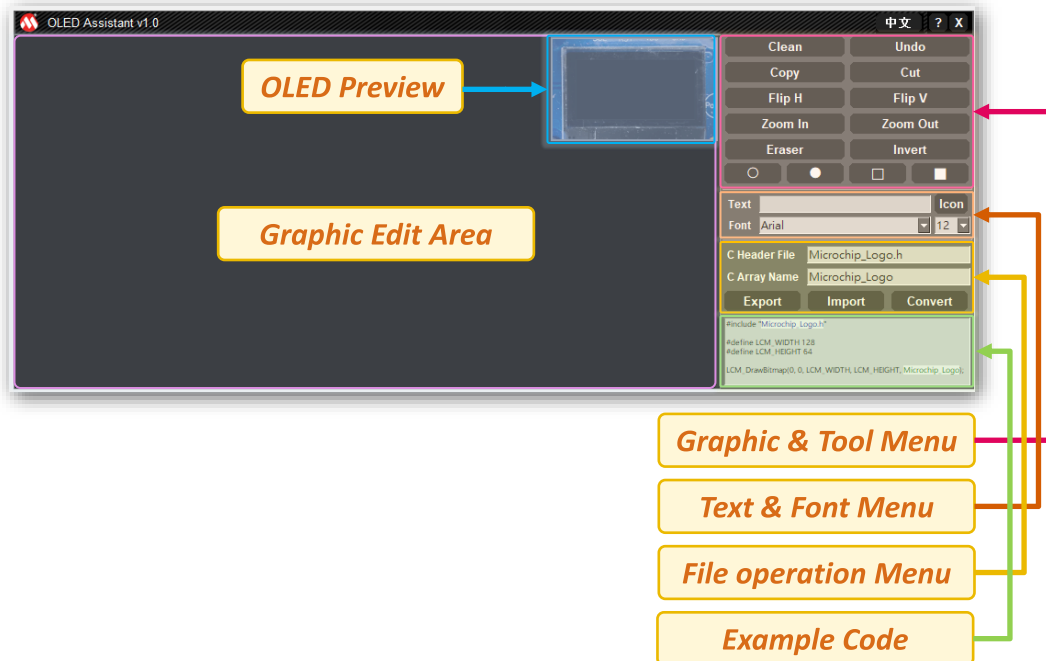
- There are couple of test images in folder \Appendix\LCDAssistantMCHP\TestImage\



✂ Lab15 Design Your Own Logo

Step 2

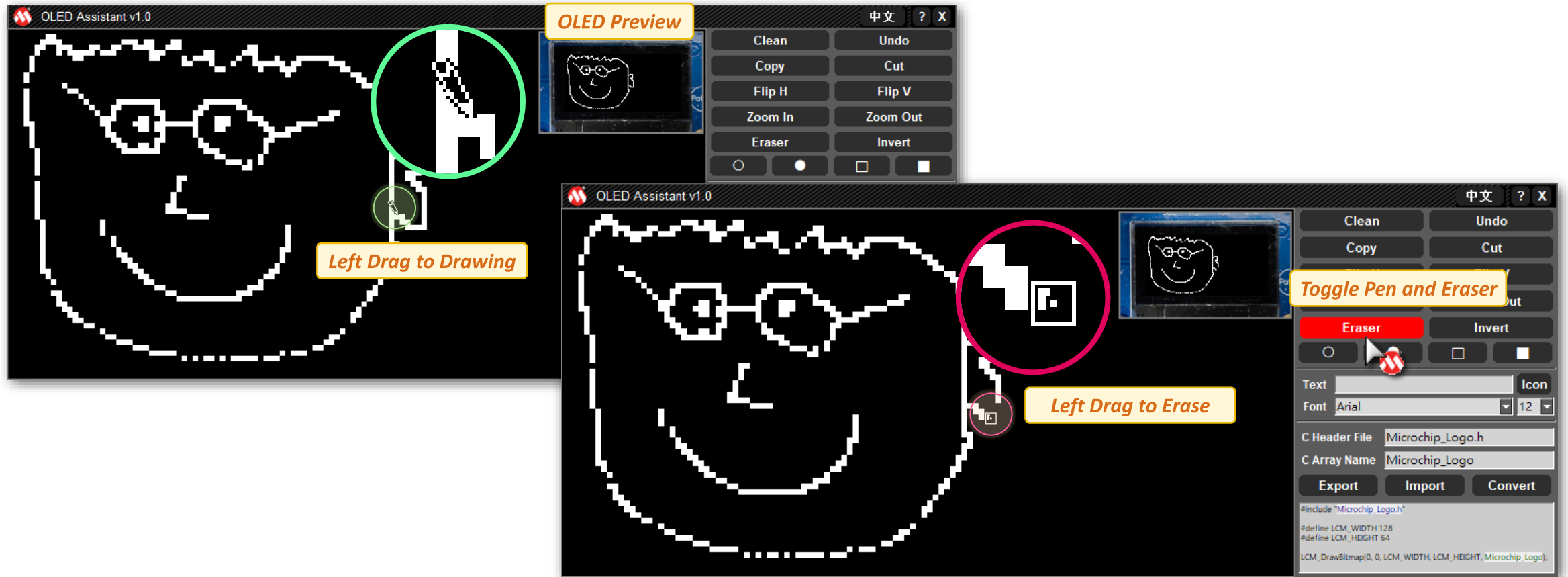
- The OLED Assistant interface introduction.



✂ Lab15 Design Your Own Logo

Step 3

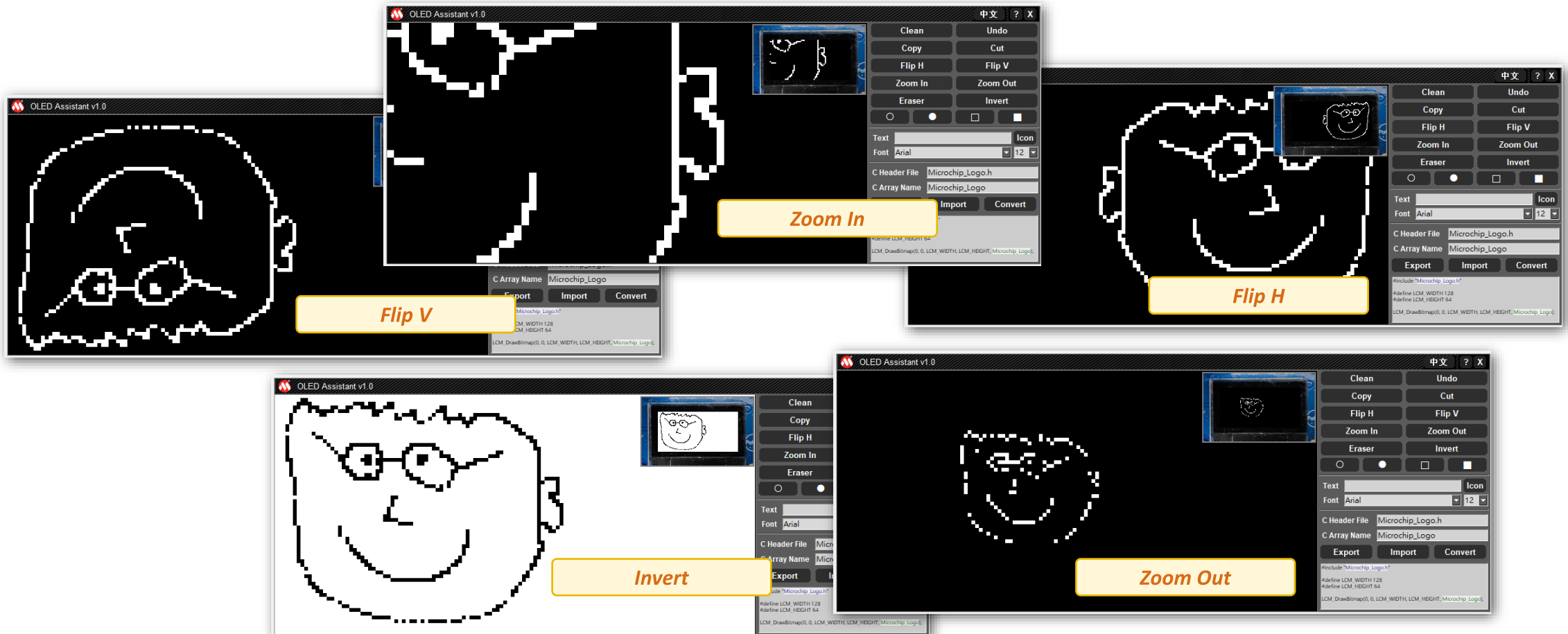
- Basic operation to have your creativity to design the pattern.



✂ Lab15 Design Your Own Logo

Step 4

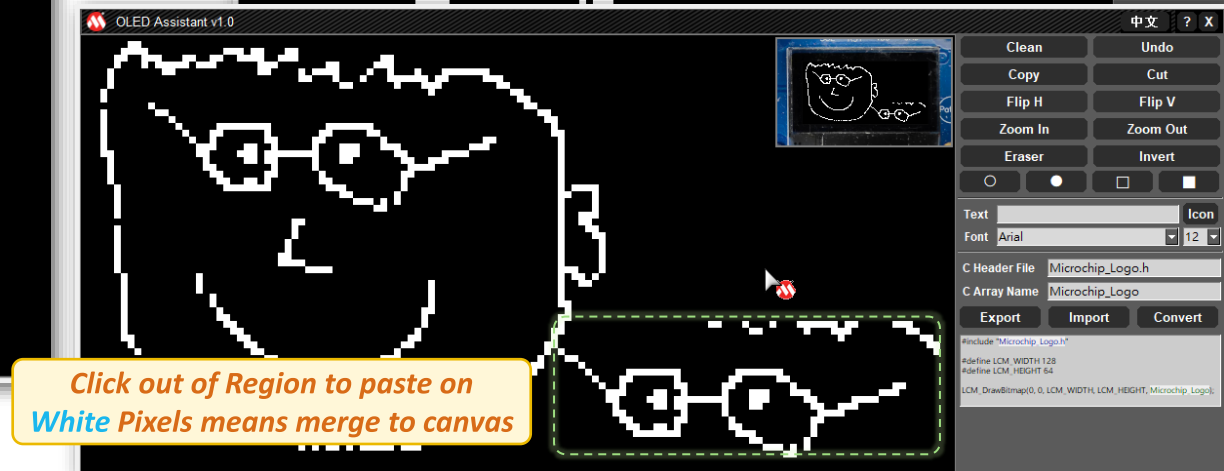
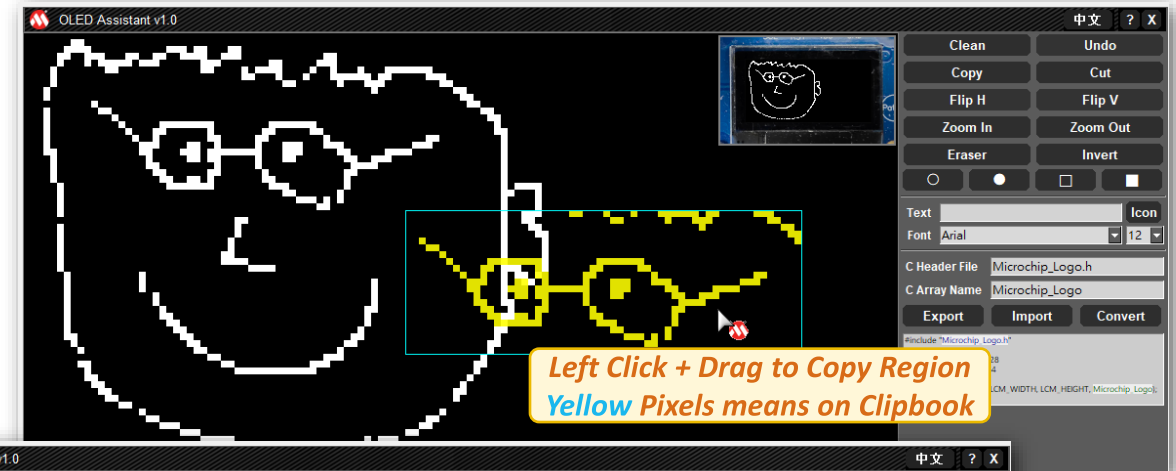
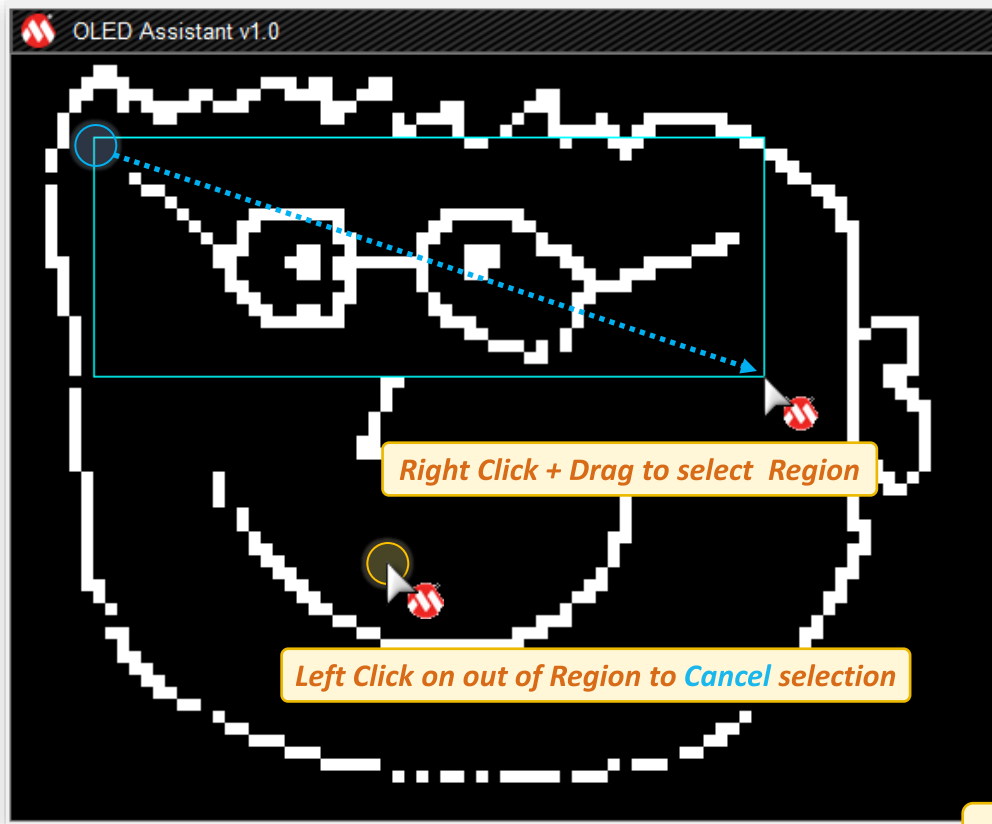
- Basic operation on whole Image (Zoon In/Out, Flip H/V, Invert).



✂ Lab15 Design Your Own Logo

Step 5

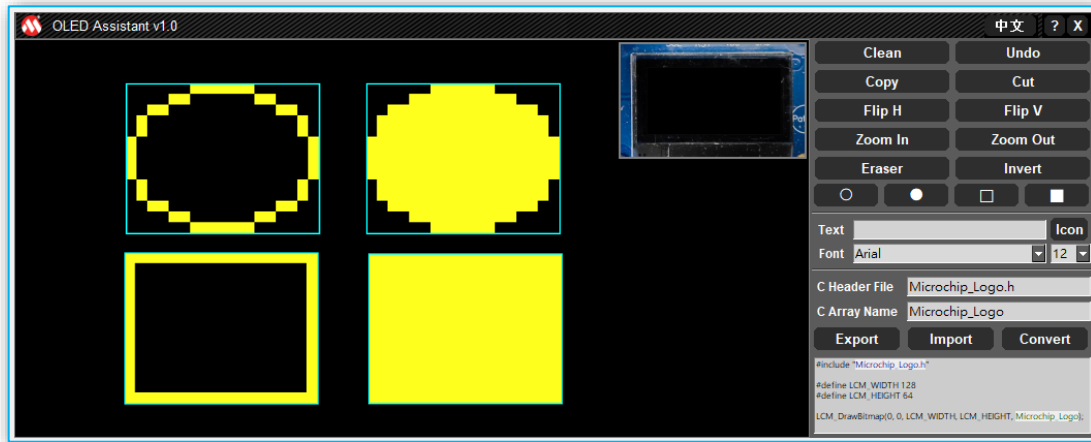
- Region Select by “Right Click + Drag” and “Left Click + Drag” to copy the Region.



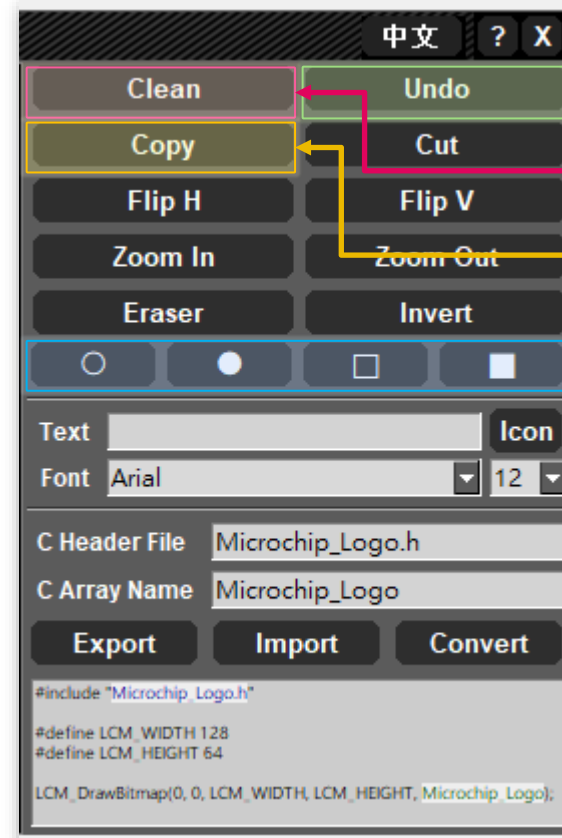
✂ Lab15 Design Your Own Logo

Step 6

- Whole image or selected region hyper operation include the keyboard shortcuts.



*Right Click + Drag to select Region
Click on the Icon to Outline or Fill-in*



Undo the changes (Ctrl + Z)

Clean whole image or selected region

Copy whole image or selected region

Outline or Filled-in whole Image or region

✖ Lab15 Design Your Own Logo

Step 7

- Select a region firstly, then type Text and choice proper Font face and Size to output.

The screenshot shows the OLED Assistant v1.0 interface. The main canvas displays a pixelated 'Welcome' text and a character. The software includes a toolbar with options like Clean, Copy, Cut, Flip H, Flip V, Zoom In, Zoom Out, Eraser, and Invert. A text input field shows 'Welcome' and a font dropdown shows 'Arial'. A callout window shows a list of fonts, with 'Bahnschrift' selected. Annotations 1-4 guide the user through the design process.

1 Select a Region for paste the Text

2 Type Text

3 Font Face

4 Font Size

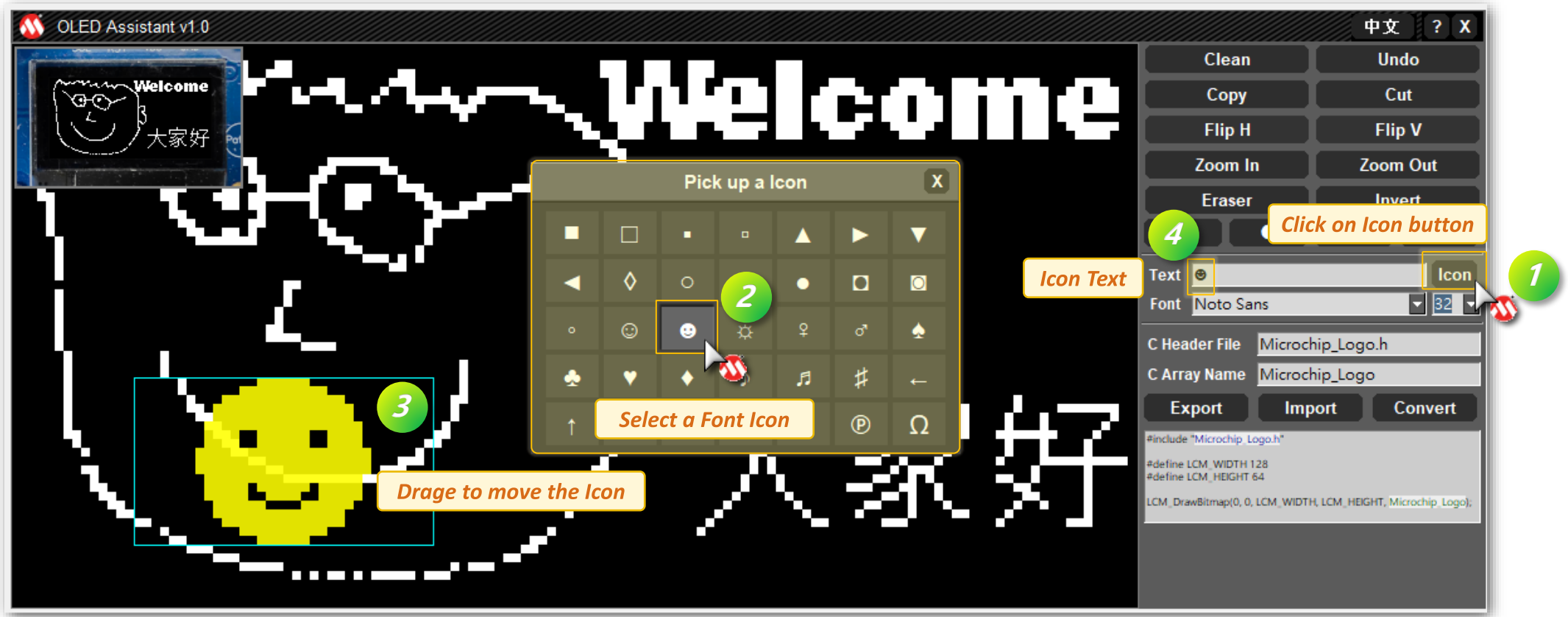
Chinese is support if select to correct Fone face

Font Face is collected from \Windows\Fonts

✂ Lab15 Design Your Own Logo

Step 8

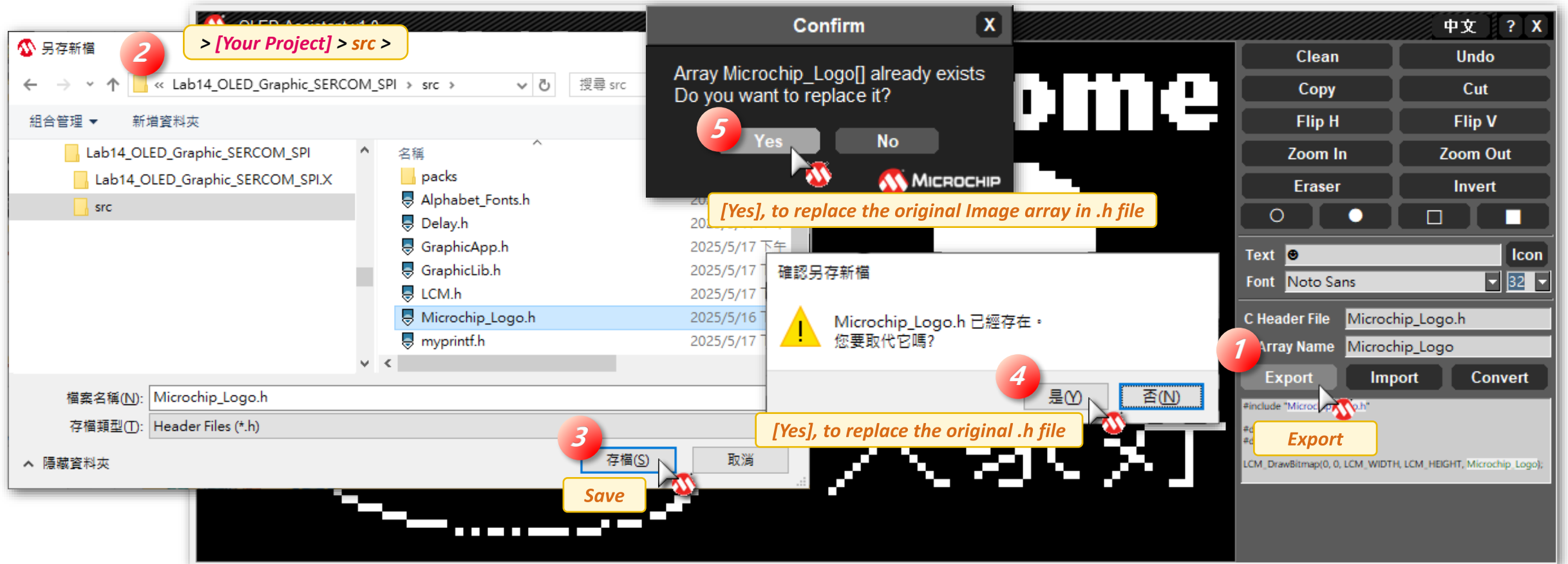
- Click on **[Icon]** button to paste a Icon by using Text and also support to select font size.



✖ Lab15 Design Your Own Logo

Step 9

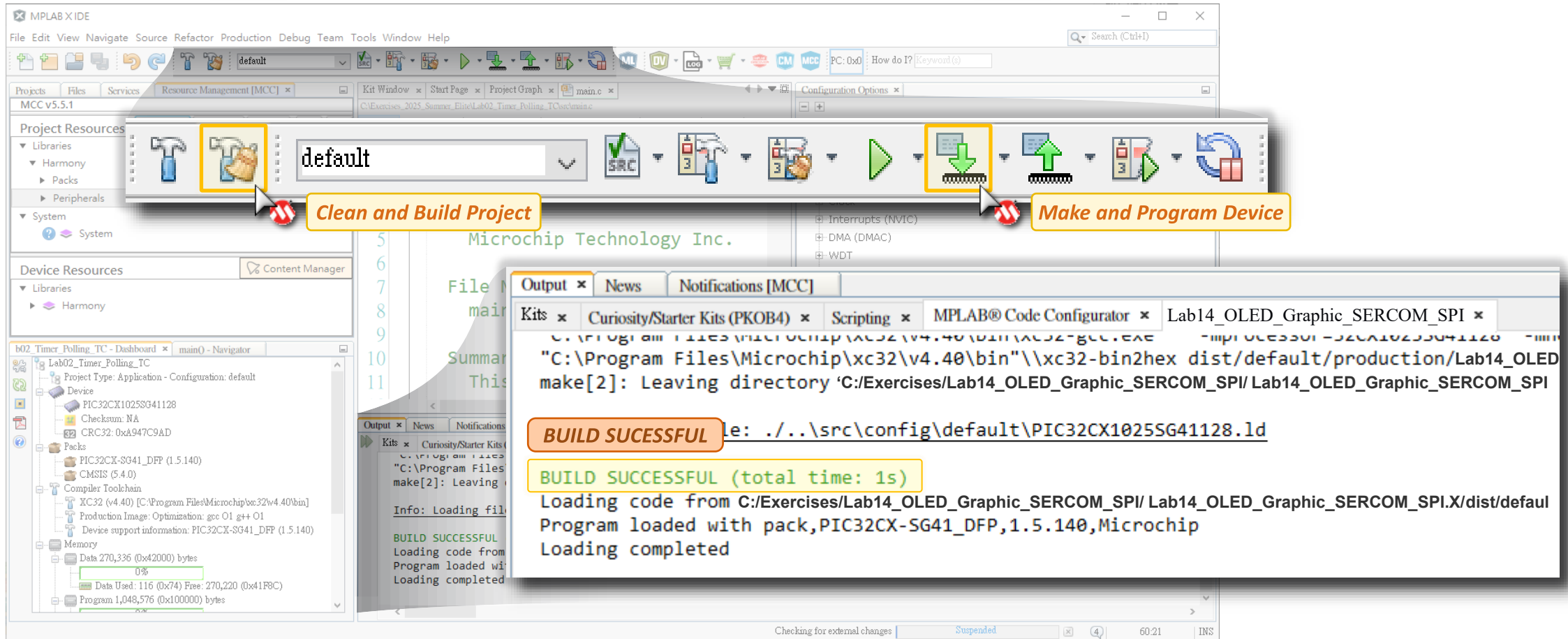
- Click on **[Export]** to output your design to a C Header file and store the Bitmap files.
- Store the **Microchip_Logo.h** to overwrite the same file in your project.



✖ Lab15 Design Your Own Logo

Step 10

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



The screenshot displays the MPLAB X IDE interface. The top toolbar contains several icons, with two highlighted by yellow boxes and red arrows. The first icon, labeled "Clean and Build Project", is a hammer and wrench. The second icon, labeled "Make and Program Device", is a green arrow pointing down to a circuit board. Below these icons, the "Output" window is open, showing the results of the build process. The output text includes:

```
le: ../../src/config/default/PIC32CX1025SG41128.ld
BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab14_OLED_Graphic_SERCOM_SPI/ Lab14_OLED_Graphic_SERCOM_SPI.X/dist/default
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

The "Output" window also shows the following text:

```
make[2]: Leaving directory 'C:/Exercises/Lab14_OLED_Graphic_SERCOM_SPI/ Lab14_OLED_Graphic_SERCOM_SPI'
```


✂ Lab15 Design Your Own Logo

Result

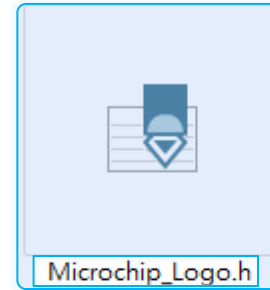


> [Your Project] > src >

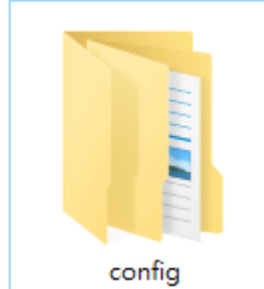
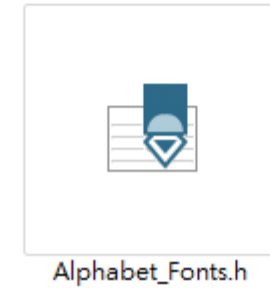
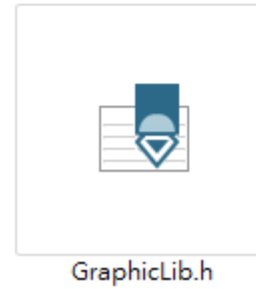
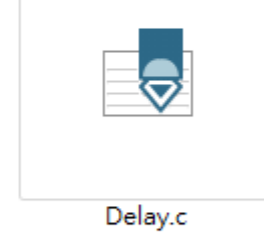
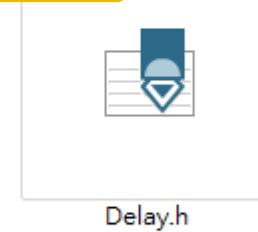
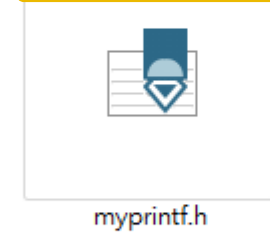
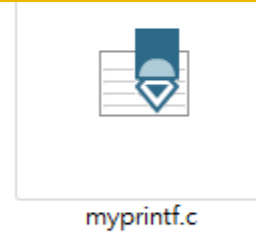
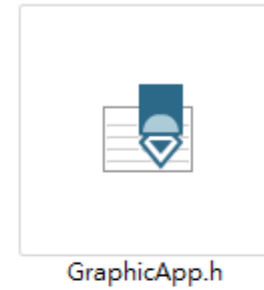
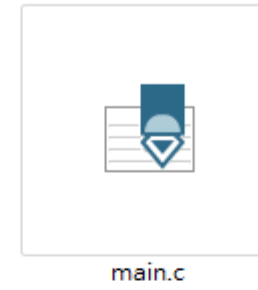
> Lab14_OLED_Graphic_SERCOM_SPI > src >



The design output Bitmap file



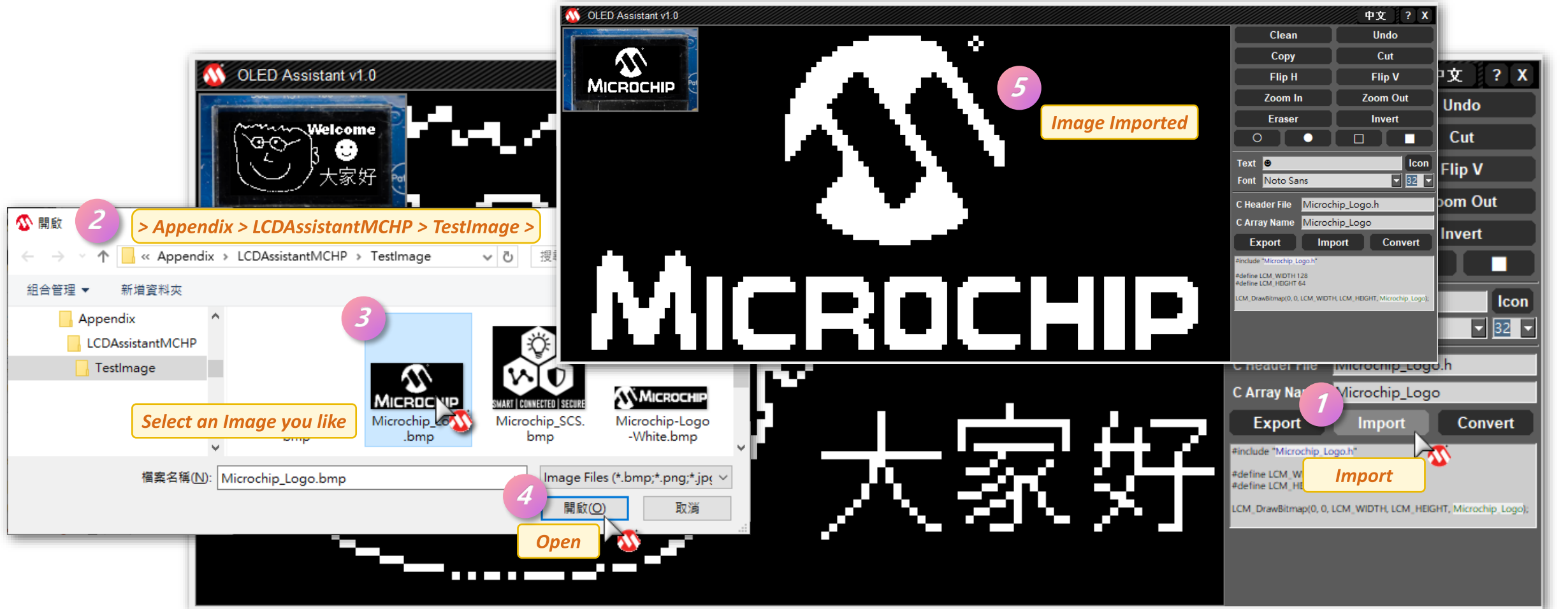
The updated C Header file



✖ Lab15 Design Your Own Logo

Advanced – Import Images

- Import the Image and re-modify it for your new image.



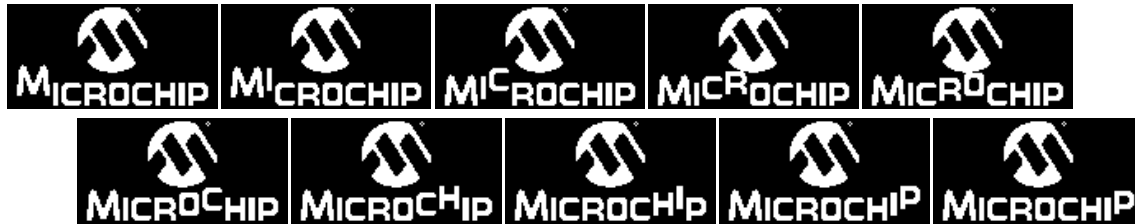
Lab15 Design Your Own Logo

Advanced – Store into multiple image arrays

- Store into multiple image arrays for later use or combine them for animation.

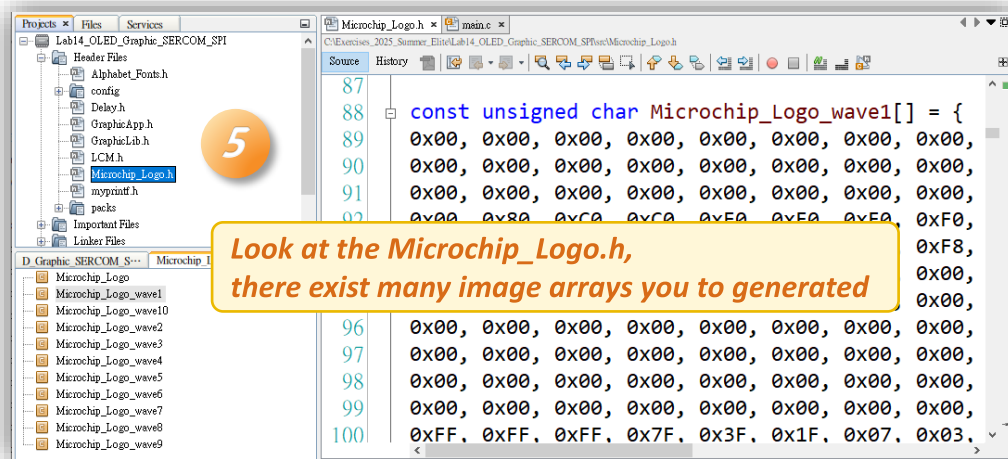
4

Generate sequence of images for animation



5

Look at the Microchip_Logo.h, there exist many image arrays you to generated



1

Modify your image slightly

2

Change the C Array Name to different

3

Export to the same C Header file



```
void MicrochipLogoWave( uint32_t ms )
{
    const unsigned char* WaveImageSequences[11] = {
        Microchip_Logo_wave1, Microchip_Logo_wave2, Microchip_Logo_wave3,
        Microchip_Logo_wave4, Microchip_Logo_wave5, Microchip_Logo_wave6,
        Microchip_Logo_wave7, Microchip_Logo_wave8, Microchip_Logo_wave9,
        Microchip_Logo_wave10, Microchip_Logo_wave11 };

    GPL_LayerSet(LAYER_GRAPHIC);
    for(int i=0 ; i<11 ; i++ )
    {
        GPL_LayerClean(LAYER_GRAPHIC);
        GPL_DrawBitmap(0, 0, LCM_WIDTH, LCM_HEIGHT, BG_SOLID, WaveImageSequences[i]);
        GPL_ScreenUpdate();
        DelayMS(DELAY_TIMER_OLED_DELAY, ms);
    }
}
```

6

Develop a Logo Wave animation function

Appendix : OLED Graphic Library

GraphicLib

OLED Graphic Library

Interface Functions

- **GPL_ScreenInit** : Initialize the screen and graphics system
- **GPL_LayerSet** : Set the active drawing layer
- **GPL_LayerShow** : Show or hide a specific layer
- **GPL_LayerClean** : Clear all pixels in a specific layer
- **GPL_ScreenClean** : Clear the entire screen
- **GPL_ScreenUpdate** : Refresh the OLED with current layer data
- **GPL_SetPenSize** : Set the pen (brush) pixel size
- **GPL_GetPenSize** : Get the current pen size
- **GPL_DrawPoint** : Draw a pixel at the given coordinate
- **GPL_DrawLine** : Draw a line between two points
- **GPL_DrawRect** : Draw an unfilled rectangle
- **GPL_FillRect** : Draw a filled rectangle with a specified color
- **GPL_DrawCross** : Draw a cross centered at a point
- **GPL_DrawEllipse** : Draw an ellipse centered at a point
- **GPL_FillEllipse** : Draw a filled ellipse centered at a point
- **GPL_DrawCircle** : Draw a circle centered at a point
- **GPL_FillCircle** : Draw a filled circle centered at a point
- **GPL_DrawArc** : Draw an arc (portion of an ellipse) with angle range
- **GPL_DrawBitmap** : Draw a bitmap image to the screen with optional background
- **GPL_Scale** : Scale a region with a given ratio
- **GPL_Printf** : Print formatted text at a specific position with optional background and highlight

OLED Graphic Library

Parameter Definitions and Enumerates

- The Graphic Library pre-defined parameters.

```
#define SCREEN_WIDTH      LCM_WIDTH
#define SCREEN_HEIGHT    LCM_HEIGHT
#define SCREEN_CENTER_X  (LCM_WIDTH>>1)
#define SCREEN_CENTER_Y  (LCM_HEIGHT>>1)

#define FONT_WIDTH        7
#define FONT_HEIGHT       8

typedef enum
{
    PIXEL_CLEAN = 0,
    PIXEL_SET
} GPL_PIXEL;

typedef enum
{
    BG_TRANSPARENT = 0,
    BG_SOLID
} GPL_BACKGROUND;
```

```
typedef enum
{
    TEXT_NORMAL = 0,
    TEXT_HIGHLIGHT
} GPL_TEXT;

typedef enum
{
    LAYER_HIDE,
    LAYER_SHOW,
} GPL_LAYERSHOW;

enum
{
    LAYER_GRAPHIC,
    LAYER_TEXT,
    LAYER_MAX
};
```

GraphicLib.h

OLED Graphic Library

The usage flow of the Graphic Library

This graphics library implementation supports multiple layers and one-time screen updates.

```
#include "LCM.h"
enum {
    LAYER_GRAPHIC,
    LAYER_TEXT,
    LAYER_MAX
};

#define GPL_LAYERS LAYER_MAX
struct
{
    // (128x64)/8, 1bit/pixel, Layer Buffer
    uint8_t Buf[LCM_FRAME_SIZE];
    uint8_t Show;
} GPL_Layer[GPL_LAYERS];

uint8_t GPL_Invalidate = false;
uint16_t GPL_PenSize = 1; // Good for Odd number 1,3,5,7,9
```

GraphicLib.c & .h

```
#include "GraphicLib.h"

void DrawGraphicExample( void )
{
    // OLED HW and Graphic Layers Initial
    // and Clean all Frame Buffer
    GPL_ScreenInit();

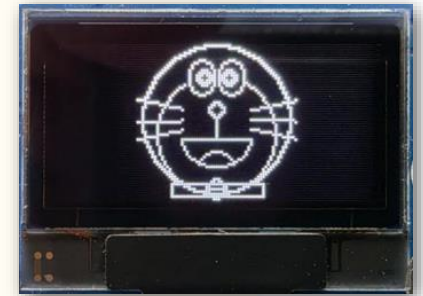
    // Show Only Graphic Layers
    GPL_LayerShow( LAYER_GRAPHIC, LAYER_SHOW );
    GPL_LayerShow( LAYER_TEXT, LAYER_HIDE );

    // Set active graphic Layer
    GPL_LayerSet( LAYER_GRAPHIC );
}
```

```
// Clear the graphic layer and set Default Pen Size
GPL_LayerClean( LAYER_GRAPHIC );
GPL_SetPenSize( 1 );
```

```
// Graphic Functions
GPL_DrawCircle(64, 30, 30, PIXEL_SET);
GPL_DrawArc(64, 36, 24, 24, 120, 60, PIXEL_SET);
GPL_DrawEllipse(58, 13, 5, 7, PIXEL_SET);
GPL_DrawEllipse(70, 13, 5, 7, PIXEL_SET);
GPL_DrawCircle(59, 13, 3, PIXEL_SET);
GPL_DrawCircle(69, 13, 3, PIXEL_SET);
GPL_DrawCircle(59, 13, 2, PIXEL_SET);
GPL_DrawCircle(69, 13, 2, PIXEL_SET);
GPL_DrawCircle(64, 26, 3, PIXEL_SET);
GPL_DrawLine(64, 29, 64, 38, PIXEL_SET);
GPL_DrawLine(50, 38, 78, 38, PIXEL_SET);
GPL_DrawArc(64, 38, 14, 11, 180, 0, PIXEL_SET);
GPL_DrawArc(64, 49, 7, 6, 10, 170, PIXEL_SET);
GPL_DrawLine(32, 22, 50, 26, PIXEL_SET);
GPL_DrawLine(32, 31, 50, 30, PIXEL_SET);
GPL_DrawLine(32, 38, 50, 34, PIXEL_SET);
GPL_DrawLine(78, 26, 96, 22, PIXEL_SET);
GPL_DrawLine(78, 30, 96, 31, PIXEL_SET);
GPL_DrawLine(78, 34, 96, 38, PIXEL_SET);
GPL_DrawRect(46, 56, 38, 6, PIXEL_SET);
GPL_DrawCircle(64, 58, 4, PIXEL_SET);
GPL_DrawLine(60, 58, 68, 58, PIXEL_SET);
GPL_DrawPoint(64, 60, 1, PIXEL_SET);
```

```
// Update to LCM
GPL_ScreenUpdate();
}
```



OLED Graphic Library

Graphic Library Drawing Functions

void GPL_DrawPoint(uint16_t x, uint16_t y, uint8_t pixel)

- Description:
 - Draw a single pixel.
- Parameter:
 - x, y: The coordinat of point.
 - pixel : The point type. (PIXEL_SET or PIXEL_CLEAN)
- Example :

```
GPL_DrawPoint( 72, 3, PIXEL_SET );  
GPL_DrawPoint( 16, 16, PIXEL_CLEAN );
```

void GPL_DrawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t pixel)

- Description:
 - Draw a straight line.
- Parameter:
 - x1, y1 : The start point of a line.
 - x2, y2 : The end point of a line.
 - pixel : The point type. (PIXEL_SET or PIXEL_CLEAN)
- Example :

```
GPL_DrawLine( 64, 0, 80, 30, PIXEL_CLEAN );  
GPL_DrawLine( 64, 30, 80, 0, PIXEL_SET );
```


OLED Graphic Library

Graphic Library Drawing Functions

void GPL_DrawRect(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint8_t pixel)

- Description:
 - Draw rectangle outline.
- Parameter:
 - x, y: The top-left coordinate of the rectangle.
 - w, h : Width and height of the rectangle.
 - pixel : The point type. (PIXEL_SET or PIXEL_CLEAN)
- Example:

```
GPL_DrawRect( 0, 0, 16, 16, PIXEL_SET );  
GPL_DrawRect( 5, 5, 32, 10, PIXEL_CLEAN );
```

void GPL_FillRect(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint8_t pixel)

- Description:
 - Draw filled rectangle.
- Parameter:
 - x, y: The top-left coordinate of the rectangle.
 - w, h : Width and height of the rectangle.
 - pixel : Fill or Clean the area (PIXEL_SET or PIXEL_CLEAN)
- Example:

```
GPL_FillRect( 10, 10, 12, 12, PIXEL_SET );  
GPL_FillRect( 47, 4, 4, 4, PIXEL_CLEAN );
```

OLED Graphic Library

Graphic Library Drawing Functions

void GPL_DrawEllipse(uint16_t x0, uint16_t y0, uint16_t rx, uint16_t ry, uint8_t pixel)

- Description:
 - Draw ellipse outline.
- Parameter:
 - x0, y0 : Center of the ellipse.
 - rx, ry : Horizontal and vertical radius.
 - pixel : The point type. (PIXEL_SET or PIXEL_CLEAN)
- Example:

```
GPL_DrawEllipse( 48, 40, 6, 18, PIXEL_SET );  
GPL_DrawEllipse( 48, 40, 10, 6, PIXEL_CLEAN );
```

void GPL_FillEllipse(uint16_t x0, uint16_t y0, uint16_t rx, uint16_t ry, uint8_t pixel)

- Description:
 - Draw filled ellipse.
- Parameter:
 - x0, y0 : Center of the ellipse.
 - rx, ry : Horizontal and vertical radius.
 - pixel : Fill or Clean the area (PIXEL_SET or PIXEL_CLEAN)
- Example:

```
GPL_FillEllipse( 48, 40, 6, 18, PIXEL_CLEAN );  
GPL_FillEllipse( 48, 40, 10, 6, PIXEL_SET );
```

OLED Graphic Library

Graphic Library Drawing Functions

void GPL_DrawCircle(uint16_t x0, uint16_t y0, uint16_t r, uint8_t pixel)

- Description:
 - Draw circle outline.
- Parameter:
 - x0, y0 : Center of the circle.
 - r : Radius.
 - pixel : The point type. (PIXEL_SET or PIXEL_CLEAN)
- Example:

```
GPL_DrawCircle( 16, 42, 15, PIXEL_SET );  
GPL_DrawCircle( 72, 25, 3, PIXEL_CLEAN );
```

void GPL_FillCircle(uint16_t x0, uint16_t y0, uint16_t r, uint8_t pixel)

- Description:
 - Draw filled circle.
- Parameter:
 - x0, y0 : Center of the circle.
 - r : Radius.
 - pixel : Fill or Clean the area (PIXEL_SET or PIXEL_CLEAN)
- Example:

```
GPL_FillCircle( 16, 42, 15, PIXEL_CLEAN );  
GPL_FillCircle( 72, 25, 3, PIXEL_SET );
```

OLED Graphic Library

Graphic Library Drawing Functions

void GPL_DrawArc(uint16_t x0, uint16_t y0, uint16_t rx, uint16_t ry, int start_angle, int end_angle, uint8_t pixel)

- Description:
 - Draw elliptical arc segment.
- Parameter:
 - x0, y0 : Center of the arc.
 - rx, ry : Horizontal and vertical radius.
 - start_angle : Start angle in degrees.
 - end_angle : End angle in degrees.
 - pixel : The point type. (PIXEL_SET or PIXEL_CLEAN)
- Example:
GPL_DrawArc(108, 16, 16, 16, 340, 230, PIXEL_SET);

void GPL_DrawCross(uint16_t x0, uint16_t y0, uint16_t r, uint8_t pixel)

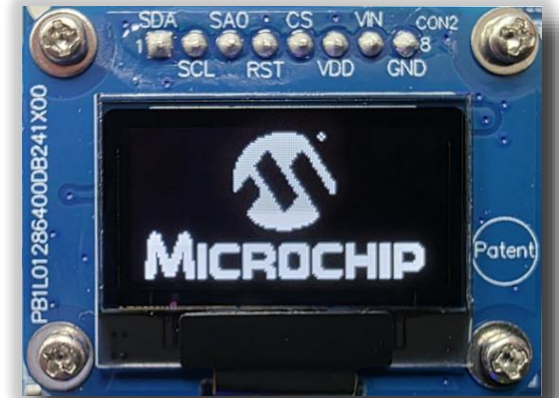
- Description:
 - Draw a cross line.
- Parameter:
 - x0, y0 : Center of the cross.
 - r : Half-length of the cross arms.
 - pixel : The point type. (PIXEL_SET or PIXEL_CLEAN)
- Example:
GPL_DrawCross(16, 42, 10, PIXEL_SET);

OLED Graphic Library

Graphic Library Drawing Functions

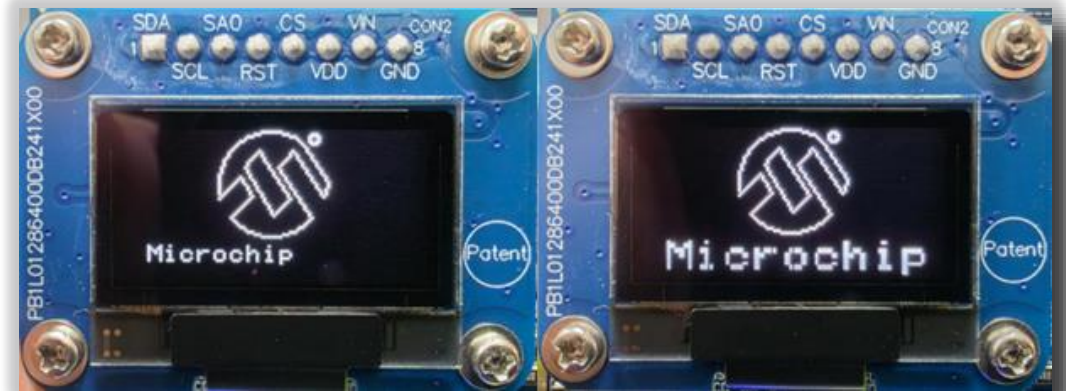
void GPL_DrawBitmap(uint16_t x, uint16_t y, uint16_t w, uint16_t h, GPL_BACKGROUND background, const unsigned char *bitmap)

- Description:
 - Display a bitmap image.
- Parameter:
 - x, y : Top-left position to draw bitmap.
 - w, h : Bitmap width and height.
 - Background : Background pixel (BG_TRANSPARENT or BG_SOLID).
 - Bitmap : Bitmap data.
- Example:
GPL_DrawBitmap(0, 0, 128, 64, BG_SOLID, Microchip_Logo);



void GPL_Scale(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, float Ratio)

- Description:
 - Scale graphics within region.
- Parameter:
 - x0, y0 : Top-left of source area.
 - w, h : Width and height of source.
 - Ratio : Scale factor (e.g., 2.0 means 200%).
- Example:
GPL_Printf(10, 48, BG_SOLID, TEXT_NORMAL, "Microchip");
GPL_Scale(10, 48, 68, 8, 1.8f);



OLED Graphic Library

Graphic Library Drawing Functions

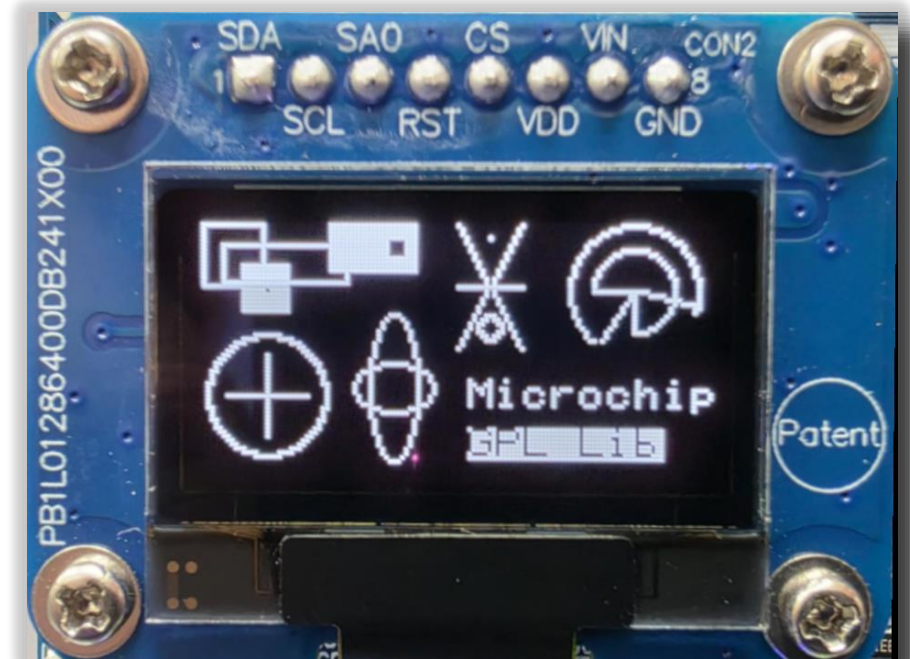
void GPL_Printf(**uint16_t** x, **uint16_t** y, **GPL_BACKGROUND** background, **GPL_TEXT** highlight, **const char** *format, ...)

- Description:
 - Print formatted text on screen at specified position.
- Parameter:
 - x, y : Position to print the text.
 - background : Background pixel (BG_TRANSPARENT or BG_SOLID).
 - highlight : Text Highlight (TEXT_NORMAL or TEXT_HIGHLIGHT).
 - format : Text format string (like printf).

- Example:

```
GPL_Printf( 67, 38, BG_SOLID, TEXT_NORMAL, "Microchip" );
```

```
GPL_Printf( 67, 50, BG_SOLID, TEXT_HIGHLIGHT, "GPL Lib" );
```



Appendix : OLED Graphic Application

GraphicApp

OLED Graphic Application

Pre-designed drawing applications

- **DrawClockAnimationInit** : Initialize clock animation parameters by compiler time
- **DrawClockAnimation** : Animate clock hands and elements
- **DrawDoraemonFace** : Draw Doraemon cartoon face
- **DrawMicrochipLogo** : Display Microchip company logo by using the Graphic Library
- **DrawGraphicElements** : Render various UI graphics elements

void DrawSplashAnimation(int x0, int y0, int x1, int y1, uint8_t step, const unsigned char *bitmap, uint8_t layer)

- Description:
 - Animates a bitmap moving from (x0, y0) to (x1, y1) on the specified graphic layer using defined step size.
- Parameter:
 - x0, y0 : Starting coordinates of the animation.
 - x1, y1 : Ending coordinates of the animation.
 - step : Step size per frame (pixels).
 - bitmap : Pointer to bitmap image data.
 - layer : Graphic layer to display the animation.
- Example:
 - DrawSplashAnimation(0, 0, 64, 32, 4, splash_logo, LAYER_GRAPHIC);

6-Axis Motion Sensor Gyroscope and Accelerometer

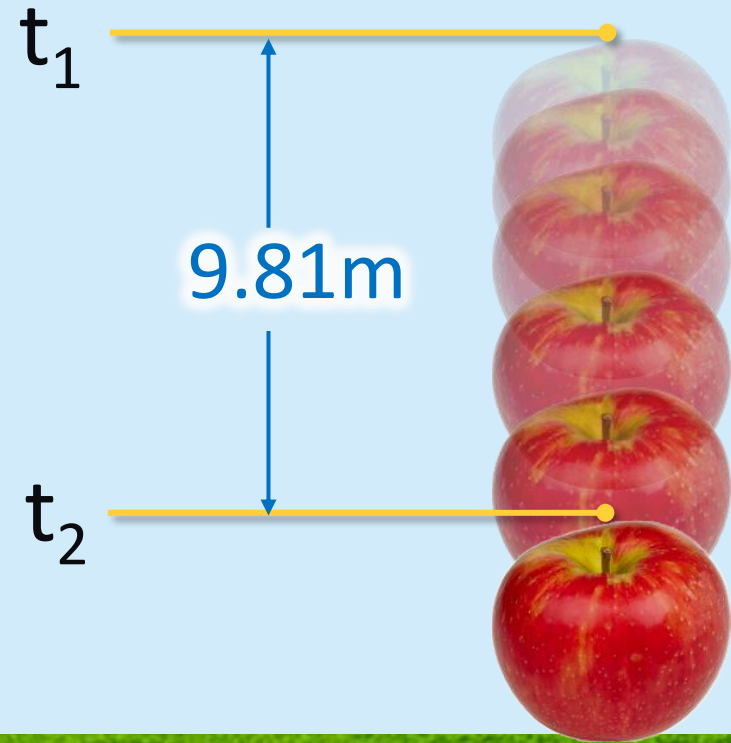
What's the Accelerometer and Gravity

- An accelerometer is a device that measures the proper acceleration of an object. Proper acceleration is the acceleration (the rate of change of velocity) of the object relative to an observer who is in free fall (that is, relative to an inertial frame of reference).
- Proper acceleration is different from coordinate acceleration, which is acceleration with respect to a given coordinate system, which may or may not be accelerating.
- For example, an accelerometer at rest on the surface of the Earth will measure an acceleration due to Earth's gravity straight upwards of about $g \approx 9.81 \text{ m/s}^2$. By contrast, an accelerometer that is in free fall will measure zero acceleration.



Gravity and Falling Acceleration

One Gravity : $g \approx 9.81 \text{ m/s}^2$



Falling Time :

$$T = t_2 - t_1 = 1 \text{ (s)}$$

t_1 Speed :

$$S_0 = 0 \text{ (m/s)}$$

t_2 Speed :

$$S = 9.81\text{m}/1\text{s} = 9.81 \text{ (m/s)}$$

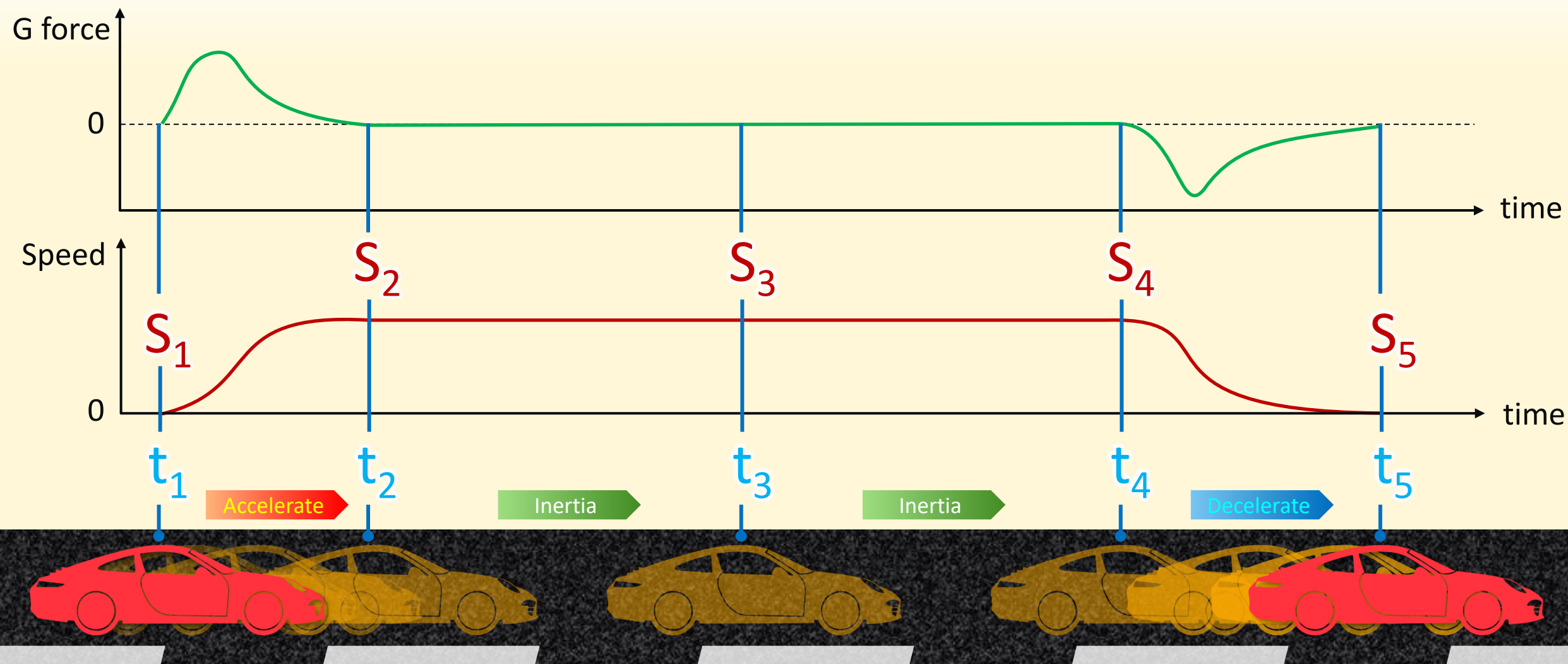
Falling Acceleration :

$$A = S / 1\text{s} = 9.81 \text{ (m/s}^2\text{)}$$

The apple starts falling from rest, regardless of air resistance.



Accelerate, Inertia and Decelerate



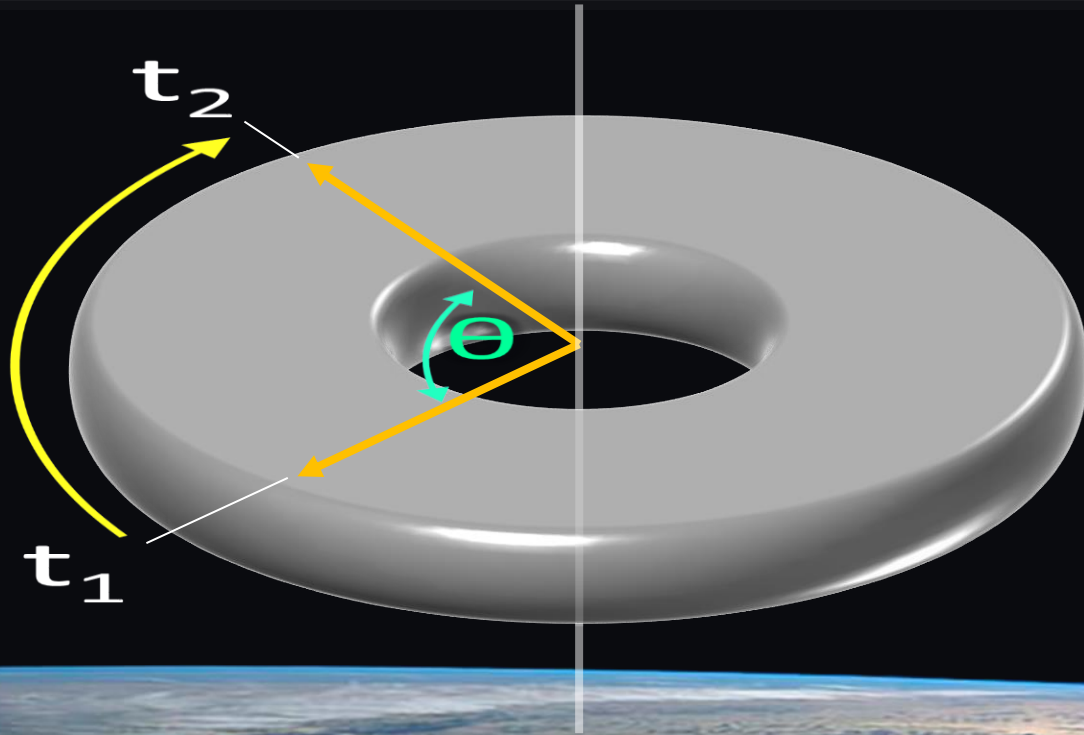
TIME TO UPGRADE

What's the Gyroscope

- A gyroscope is a device used for measuring or maintaining orientation and angular velocity. It is a spinning wheel or disc in which the axis of rotation (spin axis) is free to assume any orientation by itself. When rotating, the orientation of this axis is unaffected by tilting or rotation of the mounting, according to the conservation of angular momentum.
- Gyroscopes based on other operating principles also exist, such as the microchip-packaged MEMS gyroscopes found in electronic devices (sometimes called gyrometers), solid-state ring lasers, fibre optic gyroscopes, and the extremely sensitive quantum gyroscope.



Angular Velocity of Earth's Rotation



Rotation Time :

$$T = t_2 - t_1 = 5.98371 \text{ (h)}$$

Rotation Angle :

$$\theta = 90 \text{ (degree)}$$

$$\theta = 90 \times \pi / 180 \text{ (radian)}$$

Angular Velocity :

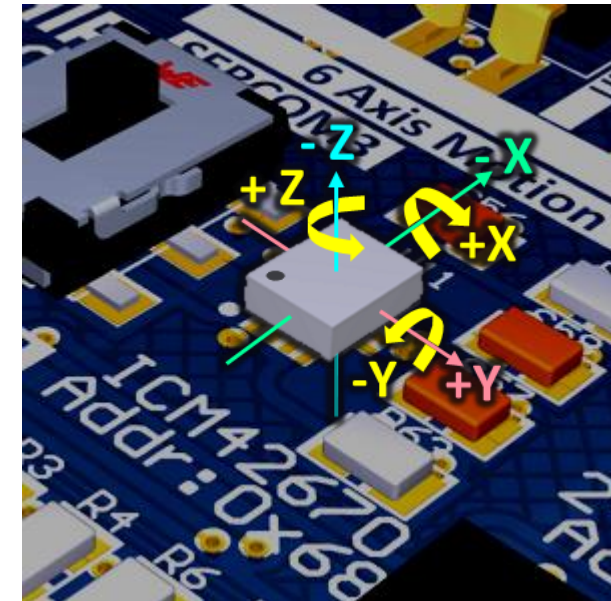
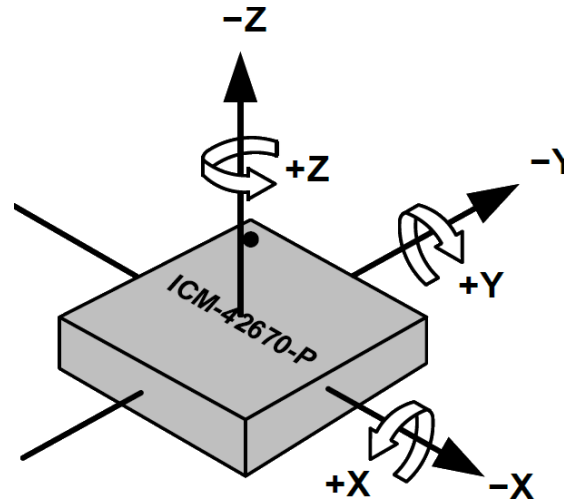
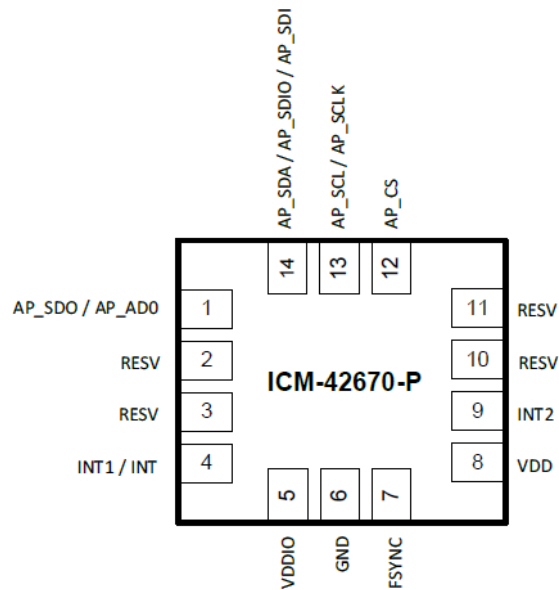
$$\omega = 7.292 \times 10^{-5} \text{ (radian/s)}$$



TIME TO UPGRADE

6DOF Motion Sensor Board (TDK ICM-42670P)

- The **TDK ICM-42670-P** is a high performance 6-axis MEMS MotionTracking™ device that combines a **3-axis Gyroscope** and a **3-axis Accelerometer** is design for **6DOF** (Six Degree Of Freedom) Motion Application.
- It has a configurable host interface that supports I3CSM, I2C, and SPI serial communication, features up to 2.25 Kbytes FIFO and 2 programmable interrupts with ultra-low-power wake-on-motion support to minimize system power consumption.



SERCOM-I2C Architecture

What's I2C ?

- **Inter-Integrated Circuit (I²C):**

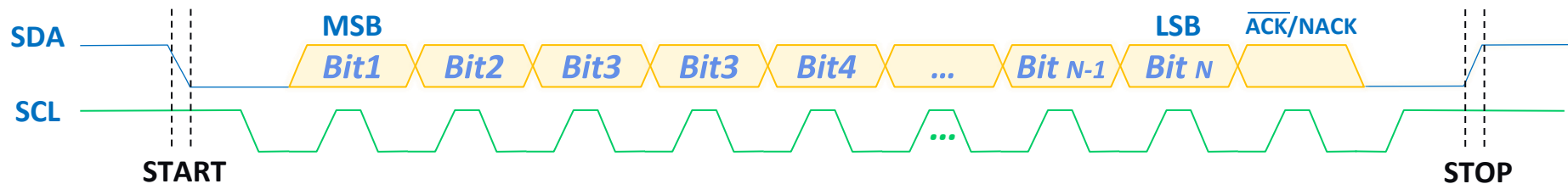
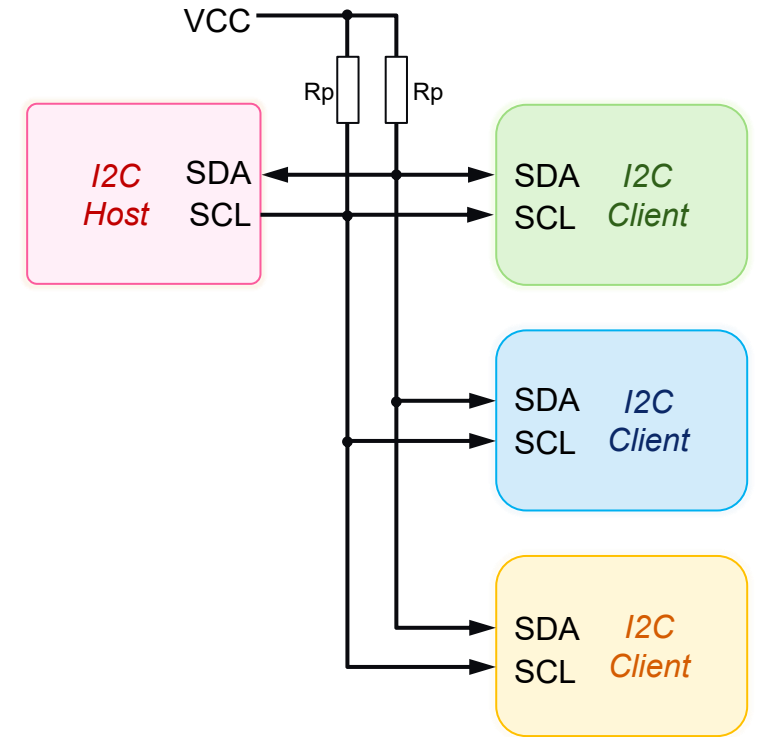
- A synchronous serial communication protocol used between multiple devices. It uses two lines: SDA (data) and SCL (clock), shared by all devices on the bus.

- **Each I²C transmission includes:**

- Start Condition: SDA goes low while SCL is high.
- Address + R/W Bit: 7 or 10-bit address plus a read/write bit.
- ACK/NACK: Receiver sends ACK (low) or NACK (high) after each byte.
- Data Frames: 8-bit data sent by master or slave.
- Stop Condition: SDA goes high while SCL is high.

- **Key Features:**

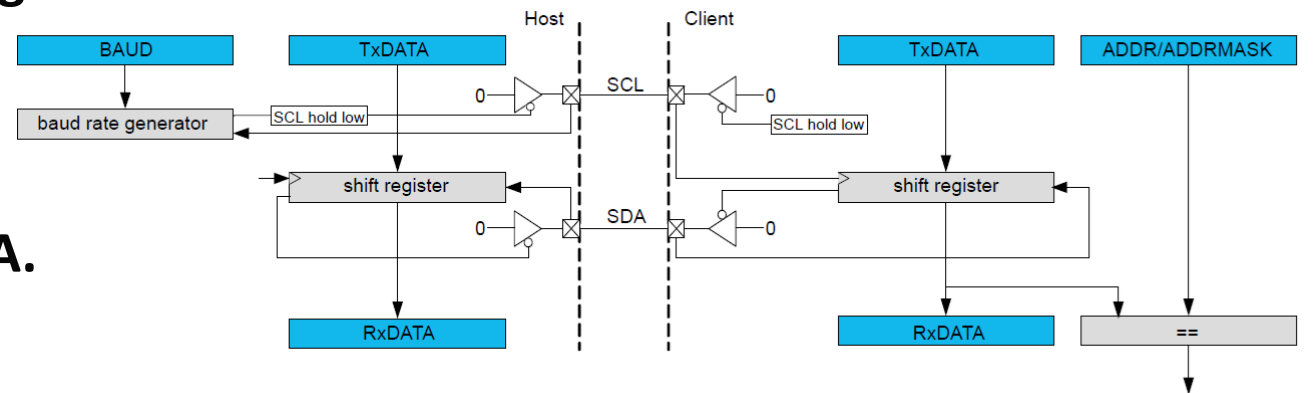
- Supports multiple masters and slaves.
- Uses only two wires: SDA and SCL.
- Speeds: 100 kbps (Standard), 400 kbps (Fast), and higher.
- Common in sensors, displays, EEPROMs, and RTCs.
- Ideal for short-distance, low-speed, on-board communication.



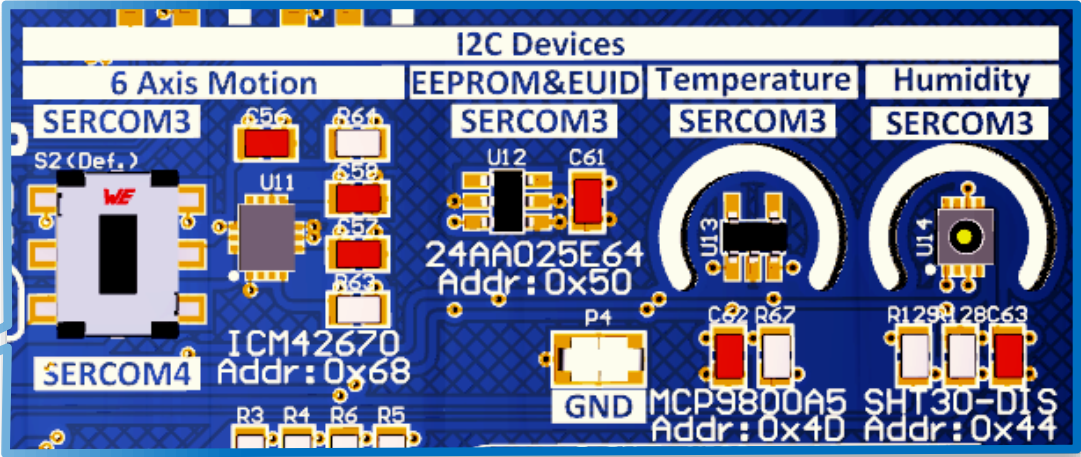
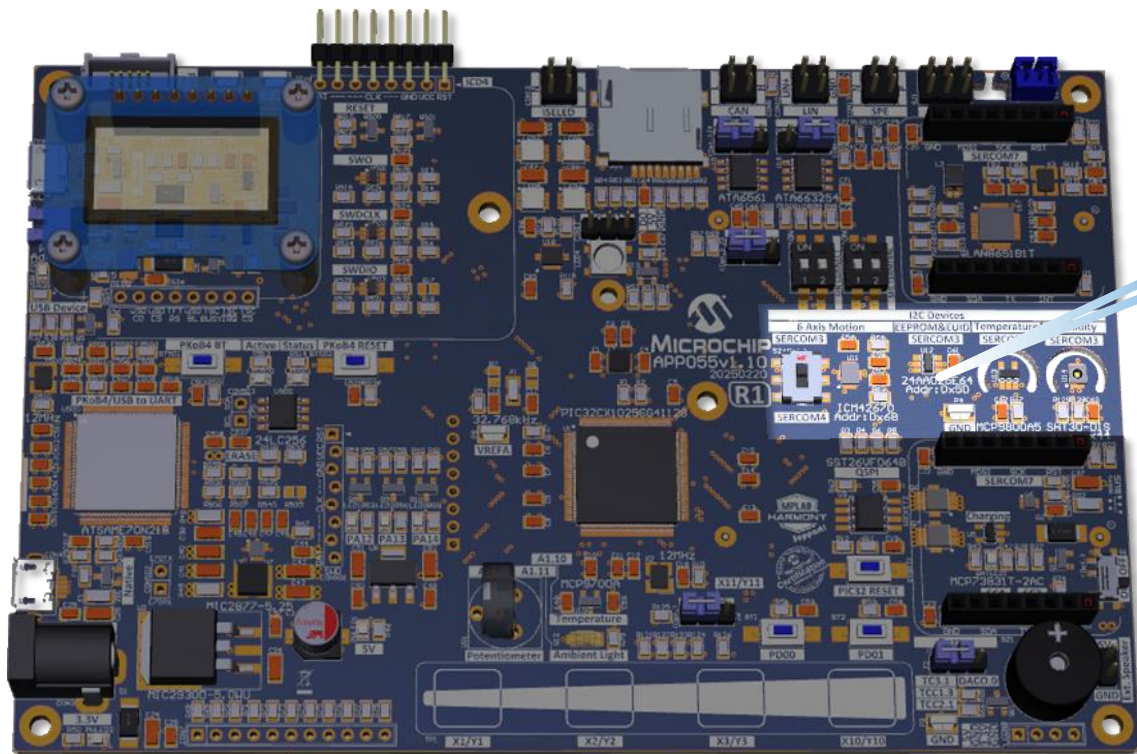
SERCOM-I2C

Flexible Synchronous Communication

- Supports both Host and Client modes.
- DMA-compatible for efficient data transfer.
- I²C speeds: 100 kHz, 400 kHz, 1 MHz, and 3.4 MHz.
- SMBus and PMBus™ compliant.
- Utilizes shift register, TX/RX buffers, and address match logic.
- 32-bit data interface for better system bus usage.
- Operates in all Sleep modes, with wake-up on address match in Client mode.
- Hardware support for 7-bit and 10-bit addressing.
- Recognizes unique, general call, and address range.
- Slew-rate limited outputs and input filtering.
- Robust Physical and Power Features
- Supports 4-wire operation.
- Client Addressing Capabilities
- Two unique addresses supported with DMA.



I2C Devices on APP055



Function	Brand	Part No.	I2C Address
6-axis Motion Sensor	TDK (InvenSense)	<u>ICM-42670-P</u>	0x68
EEPROM with EUID	Microchip	<u>24AA025E64</u>	0x50
Temperature Sensor	Microchip	<u>MCP9800A5</u>	0x4D
Humidity Sensor	Sensirion	<u>SHT30-DIS</u>	0x44

Lab16 Motion Sensor Application

SERCOM-I2C

🔧 Lab16 Motion Sensor Application

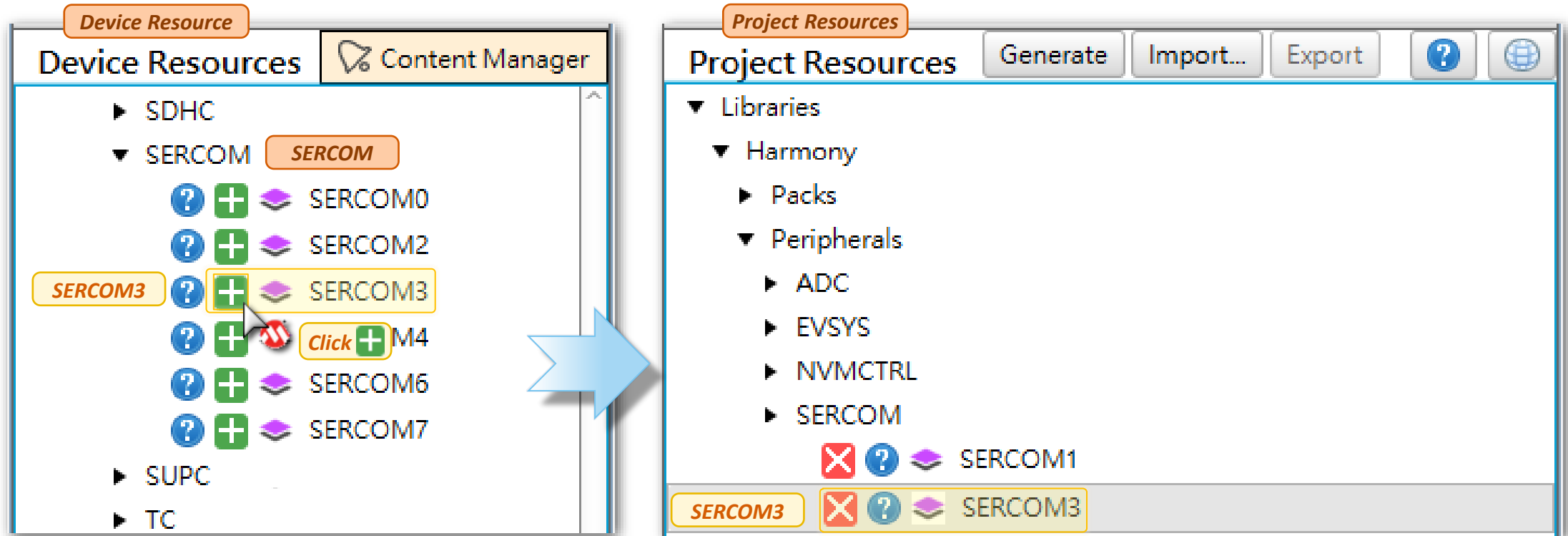
- In this Lab, we will use the **SERCOM-I2C** to communicate with the **ICM-42670-P 6-axis MEMS Motion Sensor** on the APP055 EVB. We will read the **gyroscope data (gx, gy, gz)** and **accelerometer data (ax, ay, az)**, and use **myprintf** via **SERCOM-USART** to output the sensor values and display them on Tera Term.
- To make the motion sensor data more engaging, we've also added two fun applications: **a Level Bubble** and **a Balance Ball**. These applications use the accelerometer data (ax, ay, az) to detect the **level orientation** of the APP055 EVB and display the results as animations on the OLED via **SERCOM-SPI**.
- Naturally, these applications make extensive use of the drawing functions provided in our **Graphics Library**, as well as some **3D math algorithms**. What's even more interesting is that these applications were **developed with the help of AI**. If you're interested, feel free to explore the code yourself, and combine it with the hardware capabilities to create even more fun and creative applications.

Let's go !

⚙️ Lab16 Motion Sensor Application

Step 1

- Find **SERCOM3** component in **Device Resources**.
- Click on **+** to add the **SERCOM3** module to your **Project Resources**.



⚙️ Lab16 Motion Sensor Application

Step 2

- Configure the **SERCOM3-I2C Master**.

Configuration Options ✕

[-] [+]

[-] SERCOM3

[-] Select SERCOM Operation Mode I2C Master I2C Master

... Transfer Speed Mode STANDARD_AND_FAST_MODE ▾ STANDARD_AND_FAST_MODE

... Enable operation in Standby mode ☐

... SDA Hold Time 50-100ns hold time ▾

... I2C Speed in KHz 100 ▴ ▾ 100 (100 kHz)

... I2C Trise in nano seconds 100 ▴ ▾

... **** Achievable I2C Clock Frequency = 114068Hz

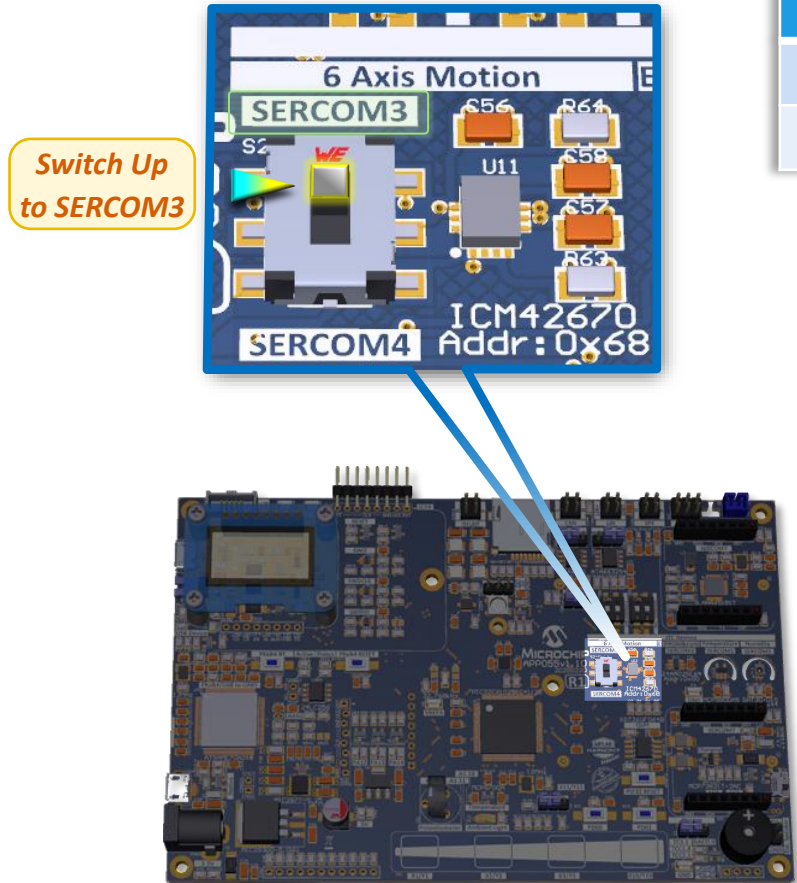
... Enable 10-bit Addressing ☐

SERCOM-I2C	I2C Signal
PAD0	SDA
PAD1	SCL

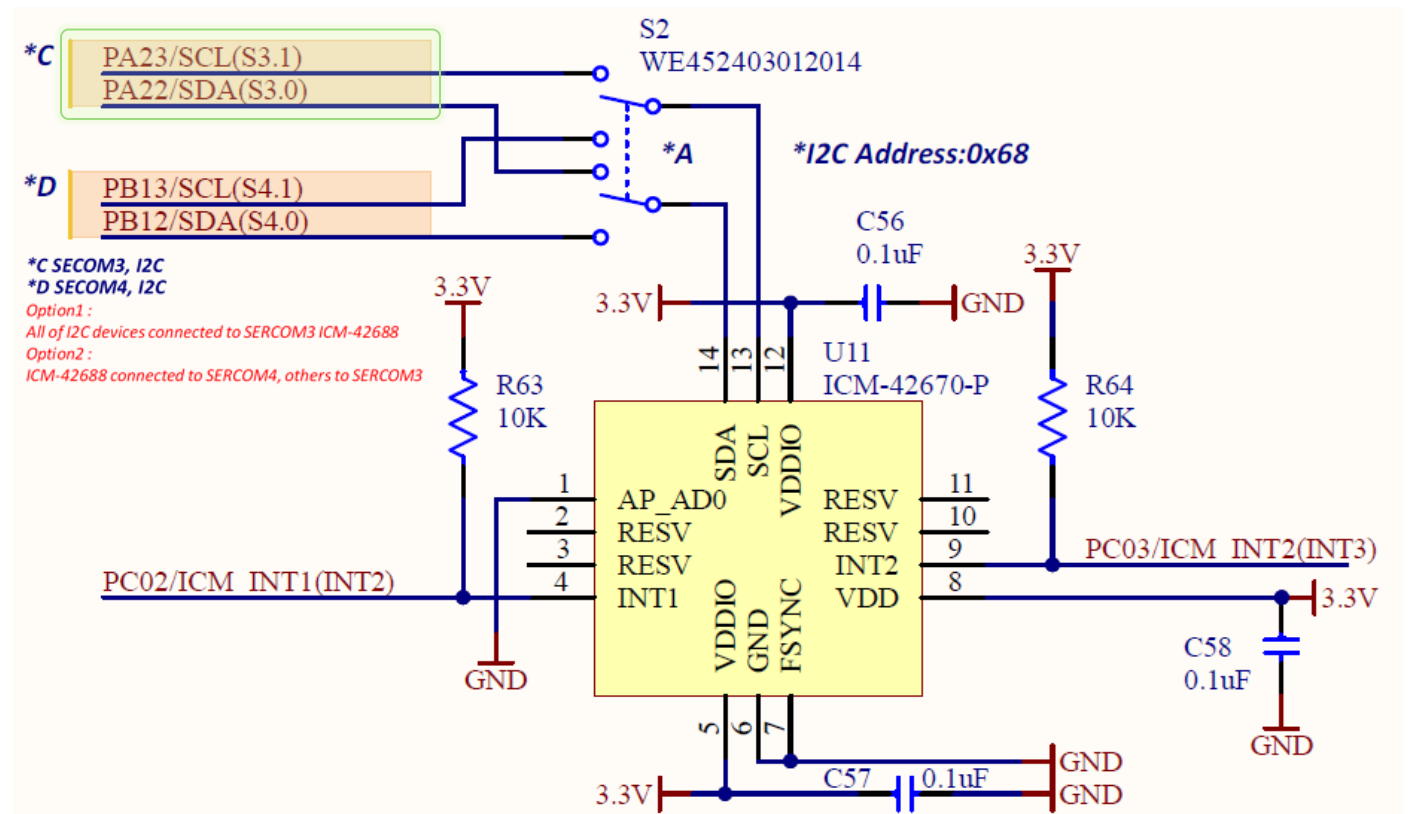
** SERCOM-I2C uses fixed PAD mapping,
no need to configure PAD in the module*

⚙️ Lab16 Motion Sensor Application

Pin connection of Motion Sensor on APP055



SERCOM3	Pin	I2C Signal
PAD0	PA22	SDA
PAD1	PA23	SCL



⚙️ Lab16 Motion Sensor Application

Step 3

- Pin Settings of the **SERCOM3-I2C** pins.

Pin Settings

Project Graph × Pin Settings × Pin Diagram × Pin Table ×

Order: Ports **Ports** Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
89	PA21		Available ▾	Digital	High Impedance ▾	Low	☐	☐	NORMAL ▾
92	PA22	PA22	SERCOM3_PAD0	SERCOM3_PAD0	High Impedance ▾	n/a	☐	☐	NORMAL ▾
93	PA23	PA23	SERCOM3_PAD1	SERCOM3_PAD1	High Impedance ▾	n/a	☐	☐	NORMAL ▾
94	PA24		Available ▾	Digital	High Impedance ▾	Low	☐	☐	NORMAL ▾

Pin ID	Function	I2C Signal
PA22	SERCOM3_PAD0	SDA
PA23	SERCOM3_PAD1	SCL

⚙️ Lab16 Motion Sensor Application

Step 4

- Click on **Generate** button to **generate** your code after configuration.

The screenshot displays the MPLAB X IDE interface. The 'Resource Management [MCC]' window is open, showing the 'Project Resources' tab. The 'Generate' button is highlighted with a yellow box and a red arrow. Below it, the 'System' resource is also highlighted. The 'Output' window is open, showing the 'MPLAB® Code Configurator' tab. The output text indicates that the generation process is complete, with a total time of 5396 milliseconds. The output also shows the execution of the generate hook and the preGenerateEvent, interfaceValidationEvent, processDataModel, and postFinalProcessing events.

```
15:17:50.827 INFO: Generation complete
15:17:50.843 INFO:
15:17:50.844 INFO: Generation complete (total time: 5396 milliseconds)
15:17:50.844 INFO:
15:17:50.844 INFO: Generation complete.
15:17:51.274 INFO: HARMONY: [ python-print ] preGenerateEvent
15:17:51.274 INFO: HARMONY: [ python-print ] interfaceValidationEvent
15:17:51.274 INFO: HARMONY: [ python-print ] processDataModel
15:17:51.274 INFO: HARMONY: [ python-print ] postFinalProcessing
```

SERCOM-I2C Master

Interface Functions

- **SERCOM3_I2C_Initialize** : Initializes instance SERCOM3 operating in I2C mode.
- **SERCOM3_I2C_TransferSetup** : Configures I2C clock speed and timing parameters for SERCOM3.
- **SERCOM3_I2C_CallbackRegister** : Registers a callback function for SERCOM3 I2C transfer completion or error.
- **SERCOM3_I2C_Read** : Performs I2C read operation from the specified slave address using SERCOM3.
- **SERCOM3_I2C_Write** : Performs I2C write operation to the specified slave address using SERCOM3.
- **SERCOM3_I2C_WriteRead** : Performs a combined write followed by read operation using SERCOM3.
- **SERCOM3_I2C_BusScan** : Scans I2C bus between specified addresses and lists active slave devices.
- **SERCOM3_I2C_InterruptHandler** : Handles I2C interrupt events including transfer progress and error handling.
- **SERCOM3_I2C_ErrorGet** : Returns the last error status encountered during I2C operations on SERCOM3.
- **SERCOM3_I2C_IsBusy** : Checks if the SERCOM3 I2C module is currently performing a transfer.
- **SERCOM3_I2C_TransferAbort** : Aborts the ongoing I2C transfer and resets the I2C module to an idle state.

SERCOM-I2C Master

The interface functions we will use in this Lab

bool SERCOM3_I2C_Write(uint16_t address, uint8_t* wrData, uint32_t wrLength)

- Description:
 - This function initiates an I2C write operation using the SERCOM3 I2C module. The specified number of bytes from the write buffer are transmitted to the given slave address.

bool SERCOM3_I2C_WriteRead(uint16_t address, uint8_t* wrData, uint32_t wrLength, uint8_t* rdData, uint32_t rdLength)

- Description:
 - This function initiates a combined I2C write followed by a read operation on the SERCOM3 I2C module. It first writes the specified number of bytes to the slave, then reads data back from the same slave without releasing the bus.
- Parameters:
 - address : 7-bit I2C slave address.
 - wrData : Pointer to the buffer containing the data to be written.
 - wrLength : Number of bytes to be written.
 - rdData : Pointer to the buffer to store received data.
 - rdLength : Number of bytes to read.
- Example:

```
uint8_t txdata[3] = {0x01, 0x02, 0x03};
uint8_t rxData[2];
SERCOM3_I2C_Write(0x50, txData, 3);
while (SERCOM3_I2C_IsBusy()) {}
SERCOM3_I2C_WriteRead(0x50, txData, 1, rxData, 2);
while (SERCOM3_I2C_IsBusy()) {}
```

SERCOM-I2C Master

The interface functions we will use in this Lab

void SERCOM3_I2C_CallbackRegister(**SERCOM_I2C_CALLBACK** callback, **uintptr_t** contextHandle)

- Description:
 - This function registers a callback handler to be called upon transfer completion or error in SERCOM3 I2C operations. The context handle will be passed back during callback execution.
- Parameters:
 - callback : Pointer to the function to be called when transfer completes or an error occurs.
 - contextHandle : Application-defined context data passed into the callback.
- Example:

```
void i2cCallback( uintptr_t context )
{ // Handle completion
}
SERCOM3_I2C_CallbackRegister(i2cCallback, 0);
```

bool SERCOM3_I2C_IsBusy(**void**)

- Description:
 - This function returns the current busy status of the SERCOM3 I2C module. It can be used to check whether a transfer is in progress.
- Return Value:
 - true : Transfer is ongoing or the bus is not idle.
 - false: I2C module is idle and ready for a new operation.
- Example:

```
if ( !SERCOM3_I2C_IsBusy() )
{ // Start new transfer
}
```

⚙️ Lab16 Motion Sensor Application

Step 5

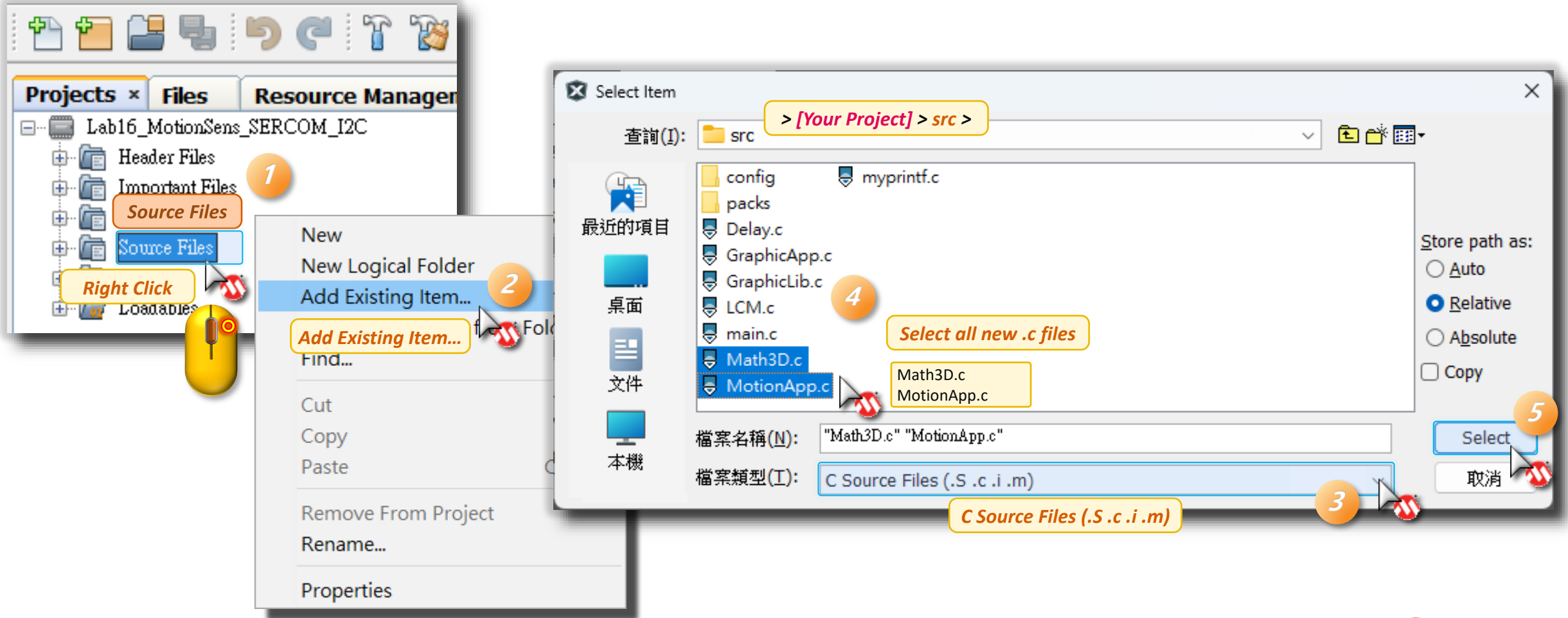
- Copy all pre-developed files in `\Labs\Resource\Lab16_MotionSens_SERCOM_I2C\` to the project folder `\[Your Project]\src\`
 - Main Function : `main.c` *To backup your original main.c if you want to keep it before replace*
 - Motion Application : `MotionApp.c` `MotionApp.h`
 - 3D Math Library : `Math3D.c` `Math3D.h`

The diagram illustrates the file transfer process. On the left, a file explorer window shows the source directory `> Labs > Resources > Lab16_MotionSens_SERCOM_I2C >` containing files `main.c`, `MotionApp.c`, `MotionApp.h`, `Math3D.c`, and `Math3D.h`. A yellow arrow points to the right, where another file explorer window shows the destination directory `> [Your Project] > src >`. This window contains various files, with `main.c`, `MotionApp.c`, `MotionApp.h`, `Math3D.c`, and `Math3D.h` highlighted in blue, indicating they have been pasted. Below the source window, a button labeled `Select All` is shown above the keyboard shortcut `ctrl + A`. Below the destination window, a button labeled `Paste` is shown above the keyboard shortcut `ctrl + V`. A button labeled `Copy` is shown above the keyboard shortcut `ctrl + C`.

⚙️ Lab16 Motion Sensor Application

Step 6

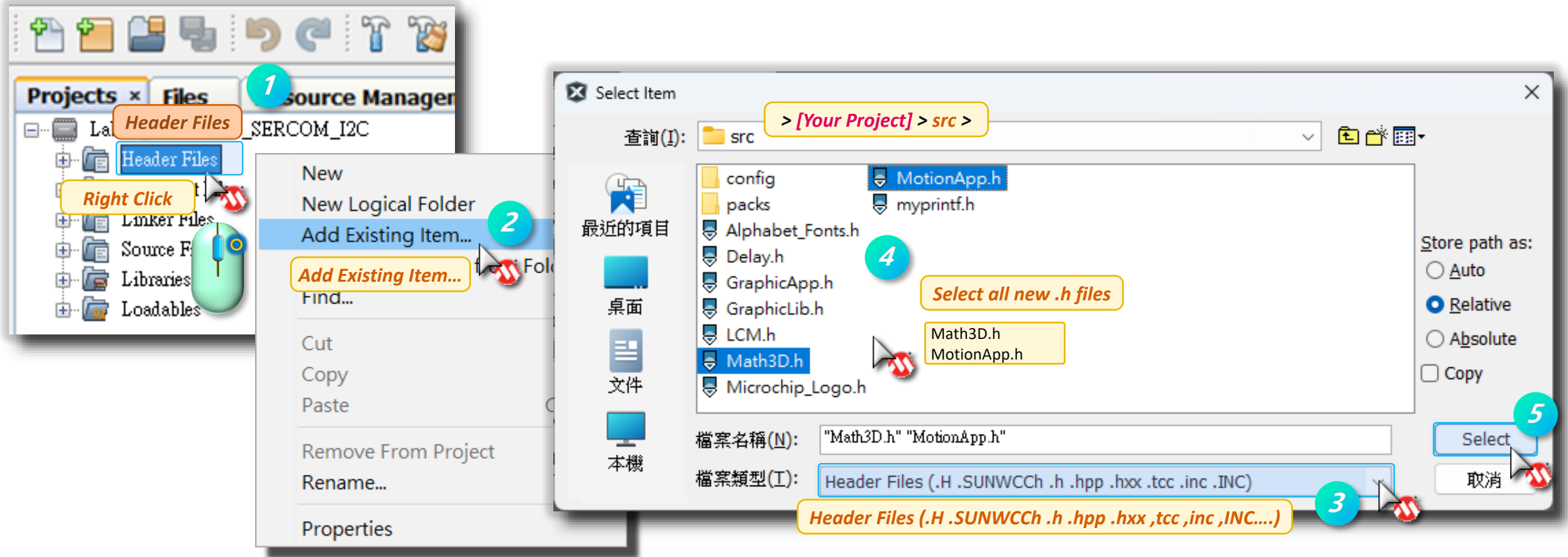
- In MPLAB X IDE, add new **Source Files (*.c)** under **\[Your Project]\src** to your project.



⚙️ Lab16 Motion Sensor Application

Step 7

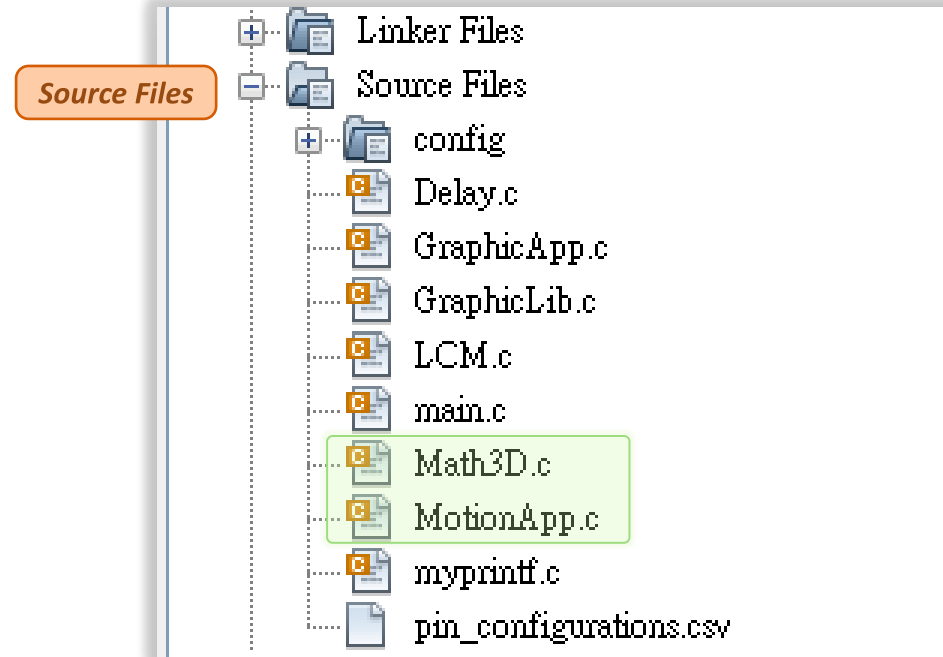
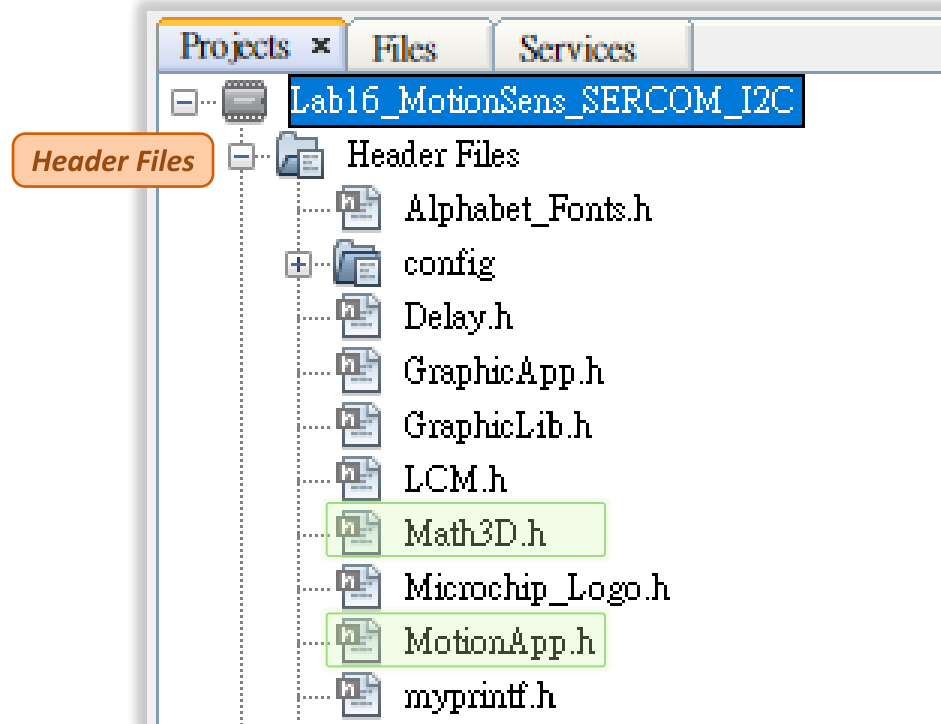
- In MPLAB X IDE, add new **Header Files (*.h)** under **\[Your Project]\src** to your project.



⚙️ Lab16 Motion Sensor Application

Step 8

- In MPLAB X IDE, check below files been added to your project.



⚙️ Lab16 Motion Sensor Application

ICM-42670P Initialize and Read Sensor Data

- **ICM-42670P initialize:**

- I2C Slave address is **0x68**.
- Write Software Reset "**0x10**" to register **SIGNAL_PATH_RESET** "**0x02**".
- Write follow Read the register **WHO_AM_I** "**0x75**" to check the ID is equal to "**0x67**" or not.
- Write "**0x0C**" to register **GYRO_CONFIG0** "**0x20**" and write "**0x0C**" to register **ACCEL_CONFIG0** "**0x21**" to config the full-scale range and output data rate of Gyro and Accelerometer.
- Write "**0x0F**" to register **PWR_MGMT0** "**0x1F**" to enable both Gyro and Accelerometer in low noise mode.

- **ICM-42670P could read Data Ready status by read the Interrupt Status Register:**

- Write follow Read the register **INT_STATUS_DRDY** "**0x39**" to get the Data Ready interrupt flag status.

- **ICM-42670P check Data Ready status in SERCOM3-I2C Callback function:**

- Check "**Bit 0** : DATA_RDY_INT" of **Interrupt Status Register** to make sure sensor data prepared for read.

- **ICM-42670P Accelerometer and Gyro Data Read:**

- Write follow Read the registers **ACCEL_DATA** and **GYRO_DATA** as below to get AX, AY, AZ and GX, GY, GZ, data.

register	AX_H	AX_L	AY_H	AY_L	AZ_H	AZ_L
address	0x0B	0x0C	0x0D	0x0E	0x0F	0x10

register	GX_H	GX_L	GY_H	GY_L	GZ_H	GZ_L
address	0x11	0x12	0x13	0x14	0x15	0x16

ICM-42670P Data Output:

- Output **ax**, **ay**, **az** and **gx**, **gy**, **gz** data as format : "**Accel** = (**±ax**, **±ay**, **±az**)" and "**Gyro** = (**±gx**, **±gy**, **±gz**)"

⚡ Lab16 Motion Sensor Application

Step 9

- Open **main.c**, declare global variables and define **SERCOM-I2C callback function**.

```
#include "GraphicLib.h"
#include "GraphicApp.h"
#include "Microchip_Logo.h"
// TODO 16.01
#include "MotionApp.h"
...

volatile uint8_t ADC1_IsCompleted = 0;
volatile int8_t ADC1_ChannelIdx = 0;
uint8_t Duty = 50;
int8_t DutyDistance = 2;
// TODO 16.02
#define ICM42670P_ADDRESS 0x68
uint8_t ICM42670P_WriteData[2];
uint8_t ICM42670P_ReadData[1];
volatile uint8_t I2C3_IsDataReady = 0;
uint8_t AxisOutReg_Accel[6] = { 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10 };
uint8_t AxisOutReg_Gyro[6] = { 0x11, 0x12, 0x13, 0x14, 0x15, 0x16 };
uint8_t AxisOutByte_Accel[6];
uint8_t AxisOutByte_Gyro[6];
short ax, ay, az, gx, gy, gz;

void I2C3_TransferCallback( uintptr_t contextHandle )
{
    if( ICM42670P_ReadData[0] & 0x1 )
    {
        I2C3_IsDataReady = 1;
    }
}
```

register	AX H	AX L	AY H	AY L	AZ H	AZ L
address	0x0B	0x0C	0x0D	0x0E	0x0F	0x10
register	GX H	GX L	GY H	GY L	GZ H	GZ L
address	0x11	0x12	0x13	0x14	0x15	0x16

⚙️ Lab16 Motion Sensor Application

Step 10

- The initialization step for the motion sensor we described on the previous page.

```
// TODO 16.02
...
}

// TODO 16.03
void MotionSensorInit( void )
{
    ICM42670P_WriteData[0] = 0x02;
    ICM42670P_WriteData[1] = 0x10;
    SERCOM3_I2C_Write( ICM42670P_ADDRESS, ICM42670P_WriteData, 2 );
    myprintf( "@1,8;ICM42670P Software Reset Completed" );
    ICM42670P_WriteData[0] = 0x75;
    SERCOM3_I2C_WriteRead( ICM42670P_ADDRESS, ICM42670P_WriteData, 1, ICM42670P_ReadData, 1 );
    while( SERCOM3_I2C_IsBusy() );
    myprintf( "@1,9;ICM42670P WHO AM I : 0x%X", ICM42670P_ReadData[0]);
    ICM42670P_WriteData[0] = 0x20;
    ICM42670P_WriteData[1] = 0x0C;
    SERCOM3_I2C_Write( ICM42670P_ADDRESS, ICM42670P_WriteData, 2 );
    ICM42670P_WriteData[0] = 0x21;
    ICM42670P_WriteData[1] = 0x0C;
    SERCOM3_I2C_Write( ICM42670P_ADDRESS, ICM42670P_WriteData, 2 );
    ICM42670P_WriteData[0] = 0x1F;
    ICM42670P_WriteData[1] = 0x0F;
    SERCOM3_I2C_Write( ICM42670P_ADDRESS, ICM42670P_WriteData, 2 );
    myprintf( "@1,10;ICM42670P Initial Completed" );
    SERCOM3_I2C_CallbackRegister( I2C3_TransferCallback, 0 );
}

// TODO 16.04
```

⚙️ Lab16 Motion Sensor Application

Step 11

- Send **Data Ready** status request and read sensor data to **ax, ay, az, gx, gy, gz** variables.

```
// TODO 16.03
}

// TODO 16.04
void MotionSensorCheckDataReady( void )
{
    ICM42670P_WriteData[0] = 0x39;
    SERCOM3_I2C_WriteRead( ICM42670P_ADDRESS, ICM42670P_WriteData, 1, ICM42670P_ReadData, 1 );
}

// TODO 16.05
void MotionSensorRead( void )
{
    for( int i=0 ; i<6 ; i++ )
    {
        SERCOM3_I2C_WriteRead( ICM42670P_ADDRESS, &AxisOutReg_Accel[i], 1, &AxisOutByte_Accel[i], 1 );
        while( SERCOM3_I2C_IsBusy() );
        SERCOM3_I2C_WriteRead( ICM42670P_ADDRESS, &AxisOutReg_Gyro[i], 1, &AxisOutByte_Gyro[i], 1 );
        while( SERCOM3_I2C_IsBusy() );
    }
    ax = (short) (AxisOutByte_Accel[0]<<8|AxisOutByte_Accel[1]);
    ay = (short) (AxisOutByte_Accel[2]<<8|AxisOutByte_Accel[3]);
    az = (short) (AxisOutByte_Accel[4]<<8|AxisOutByte_Accel[5]);
    gx = (short) (AxisOutByte_Gyro[0]<<8|AxisOutByte_Gyro[1]);
    gy = (short) (AxisOutByte_Gyro[2]<<8|AxisOutByte_Gyro[3]);
    gz = (short) (AxisOutByte_Gyro[4]<<8|AxisOutByte_Gyro[5]);
}

// TODO 16.06
```

register	AX H	AX L	AY H	AY L	AZ H	AZ L
address	0x0B	0x0C	0x0D	0x0E	0x0F	0x10
register	GX H	GX L	GY H	GY L	GZ H	GZ L
address	0x11	0x12	0x13	0x14	0x15	0x16

⚙️ Lab16 Motion Sensor Application

Step 12

- **BT1** to switch between four Applications and **BT2** to Zero Calibration .

```
// TODO 16.05
// TODO 16.06
void MotionSensorGUI( void )
{
    static uint8_t ToggleGUI = 0;
    bool Reset =false;

    if( !BT1_Get() )
    {
        while( !BT1_Get() ){} // Button debounce
        ToggleGUI=(ToggleGUI+1)%4;
    }

    if( !BT2_Get() )
    {
        while( !BT2_Get() ) {} // Button debounce
        if( ToggleGUI==2 ) ax=ay=TC2_Timer16bitCounterGet()%2048;
        else Reset=true;
    }

    switch( ToggleGUI )
    {
    case 0: DrawLevelBubble ( ax, ay, Reset, LAYER_GRAPHIC ); break;
    case 1: DrawBalanceBall ( ax, ay, Reset, LAYER_GRAPHIC ); break;
    case 2: DrawRotationCube( ax, ay, Reset, LAYER_GRAPHIC ); break;
    case 3: ViewRotationCube( ax, ay, az, gx, gy, gz, Reset, LAYER_GRAPHIC ); break;;
    }
    GPL_ScreenUpdate();
}
```

To uses the TC2 count as the random number for ax & ay

* The TC2 to be used as the tick base of Delay function

⚙️ Lab16 Motion Sensor Application

Step 13

- Sensor initialize before Splash Screen and request a “Check Data Ready” in **TC1 (40ms)**.

```
int main ( void )
{
    ...
    TCC1_PWMStart();

    // TODO 16.07
    MotionSensorInit();

    ...

    while ( true )
    {
        ...

        if (TC1_HasExpired)
        {
            TC1_HasExpired = 0;
            LED2_Toggle();

            Duty += DutyDistance;
            if (Duty >= 100 || Duty <= 0)
                DutyDistance = -DutyDistance;

            if (Duty >= 100)
                TCC1_PWM24bitDutySet(3, TCC1_PWM24bitPeriodGet() + 1);
            else
                TCC1_PWM24bitDutySet(3, ((uint32_t) Duty * TCC1_PWM24bitPeriodGet()) / 100);

            // TODO 16.08
            MotionSensorCheckDataReady();
        }
    }
}
```

⚙️ Lab16 Motion Sensor Application

Step 14

- Output Sensor data and execute Application if software flag been set by Data Ready.

```
while ( true )
{
    ...

    if( ADC0_IsCompleted )
    {
        ADC0_IsCompleted = 0;
        myprintf( "@1,2;VR1 Value : %4d", ADC0_Result );
    }

    if( ADC1_IsCompleted )
    {
        ADC1_IsCompleted = 0;
        myprintf( "@1,3;Temperature Value : %4d", ADC1_Result[0] );
        myprintf( "@1,4;Lightness Value : %4d", ADC1_Result[1] );
    }

    // TODO 16.09
    if( I2C3_IsDataReady )
    {
        MotionSensorRead();

        myprintf( "@1,6;Accel = (%+5d, %+5d, %+5d)", ax, ay, az );
        myprintf( "@1,7;Gyro = (%+5d, %+5d, %+5d)", gx, gy, gz );

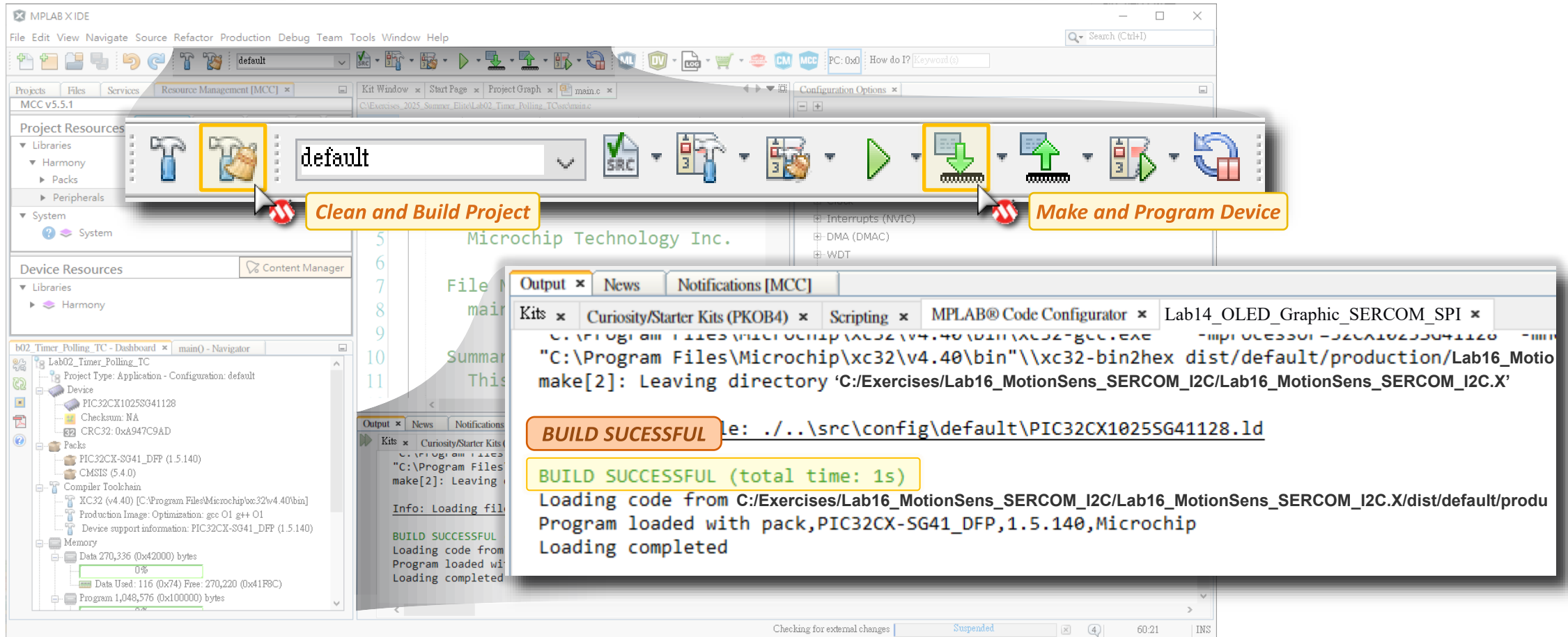
        MotionSensorGUI();

        I2C3_IsDataReady = 0;
    }
}
```


⚙️ Lab16 Motion Sensor Application

Step 15

- Click on icon  to **Clean and Build Project** and then  to **Program code to EVB**.



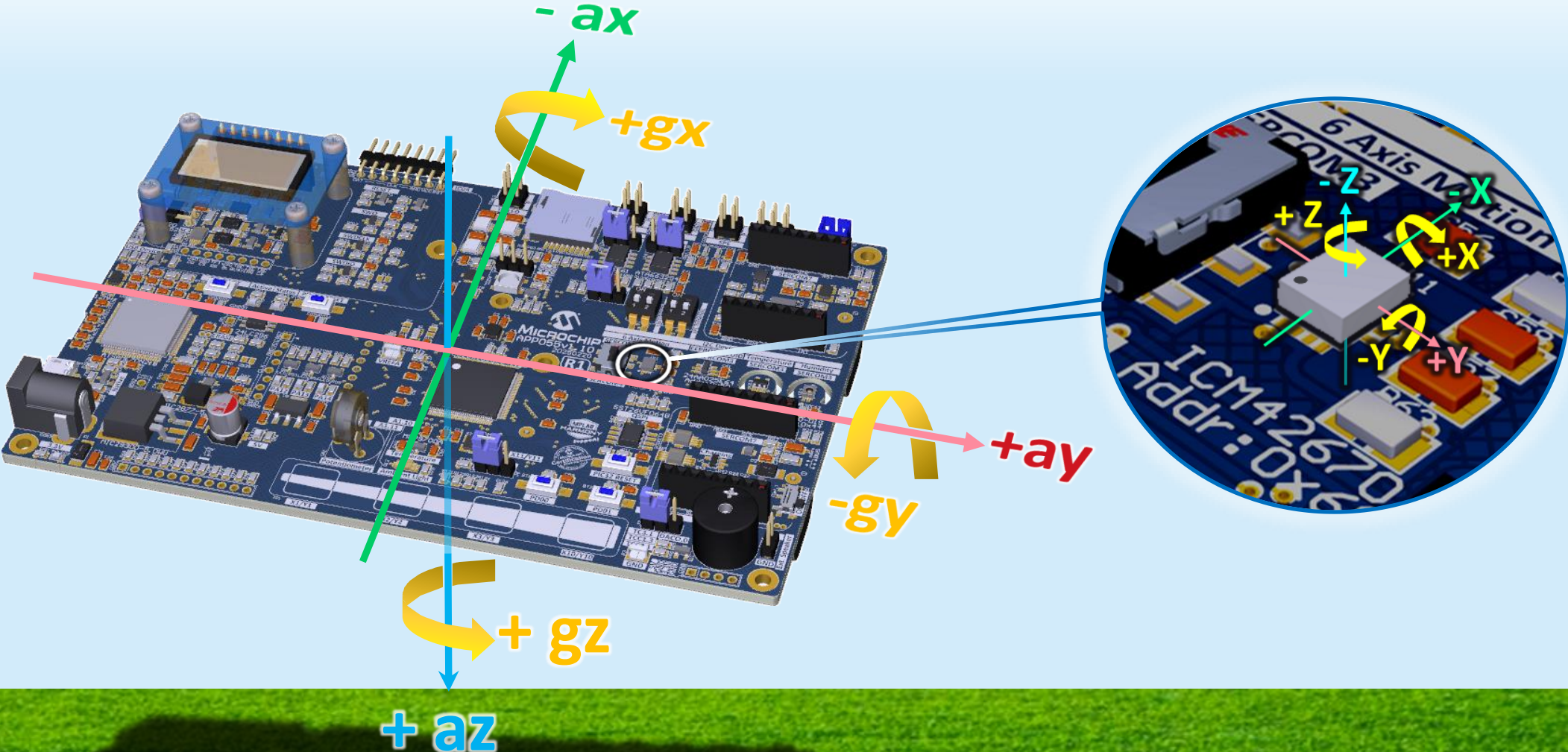
The screenshot displays the MPLAB X IDE interface. The top toolbar contains icons for various actions. Two specific icons are highlighted with yellow boxes and red arrows pointing to them:

- Clean and Build Project:** Indicated by a yellow box around the icon showing a hammer and a wrench, with a red arrow pointing to it from a yellow callout box labeled "Clean and Build Project".
- Make and Program Device:** Indicated by a yellow box around the icon showing a green arrow pointing down to a circuit board, with a red arrow pointing to it from a yellow callout box labeled "Make and Program Device".

The bottom right of the image shows the Output window with the following text:

```
le: ../../src/config/default/PIC32CX1025SG41128.ld
BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Exercises/Lab16_MotionSens_SERCOM_I2C/Lab16_MotionSens_SERCOM_I2C.X/dist/default/produ
Program loaded with pack,PIC32CX-SG41_DFP,1.5.140,Microchip
Loading completed
```

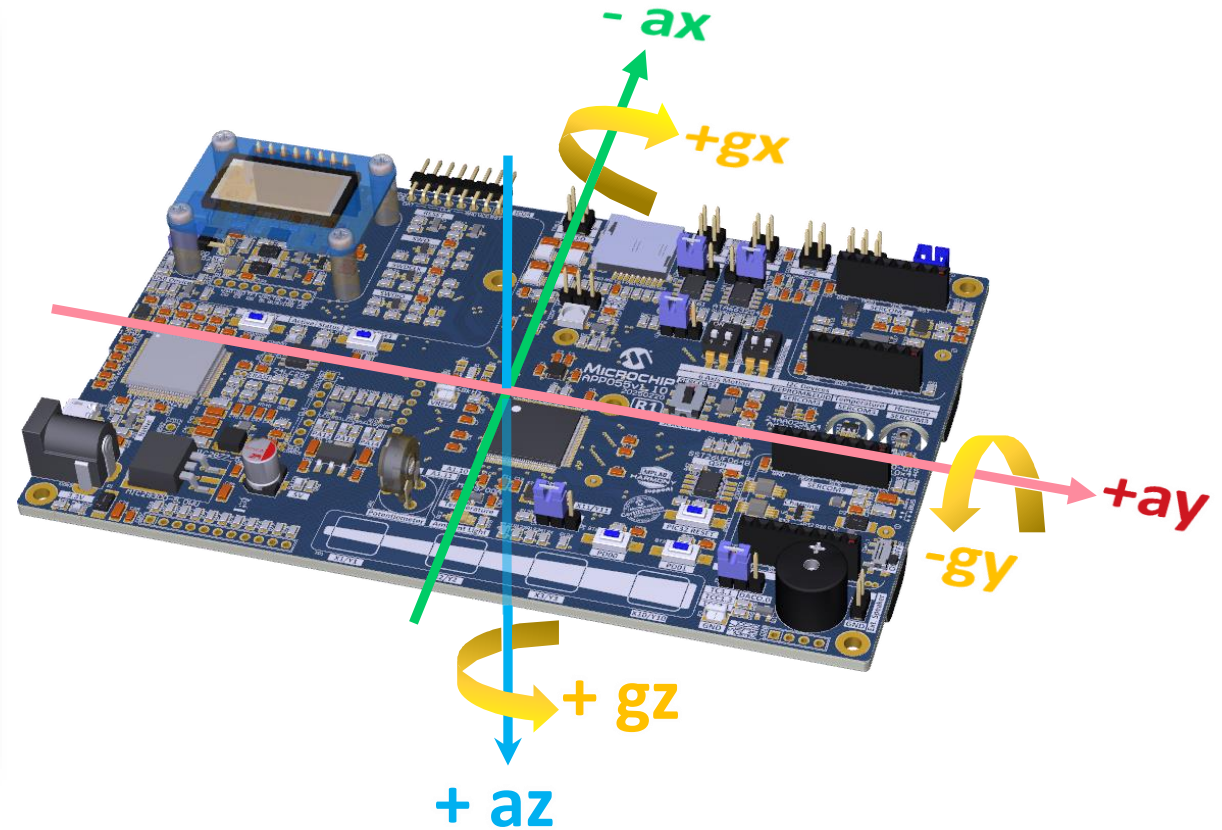
The location of TDK ICM-42670P on APP055



⚙️ Lab16 Motion Sensor Application

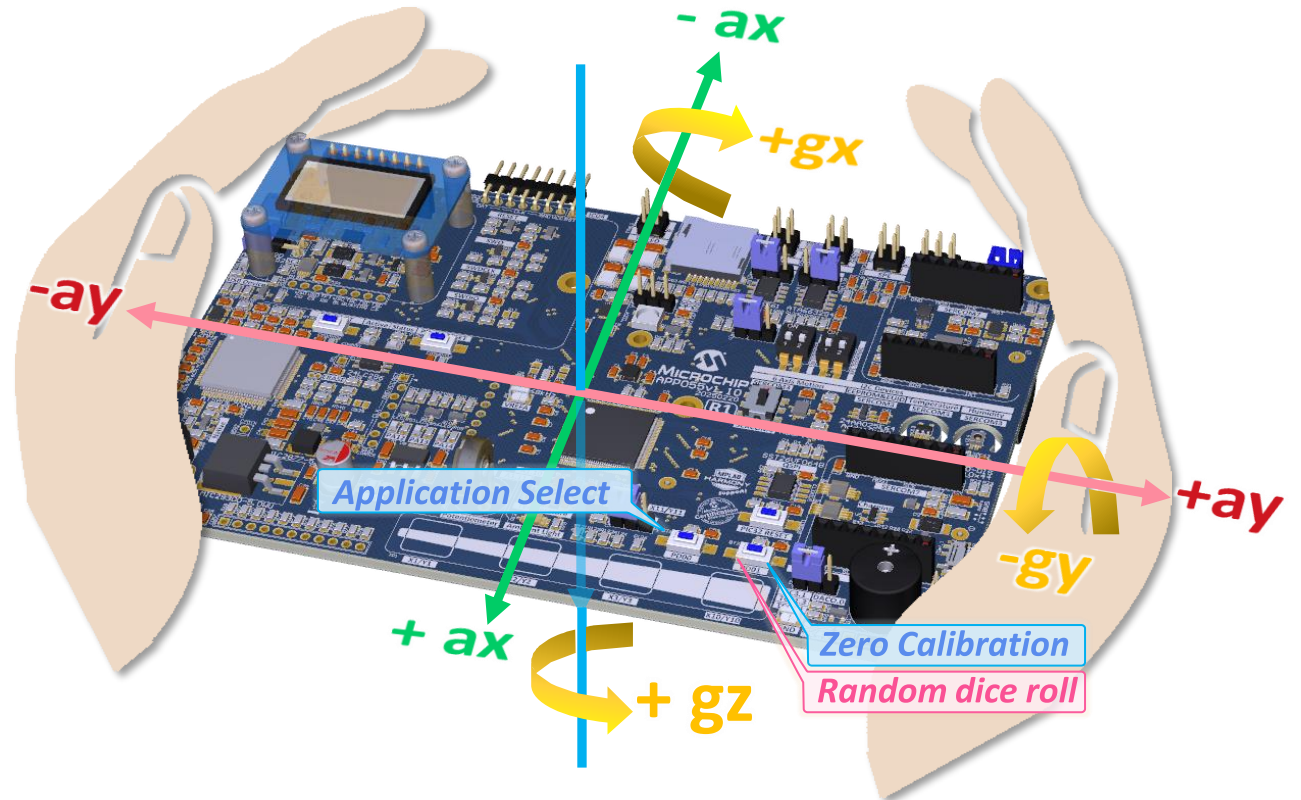
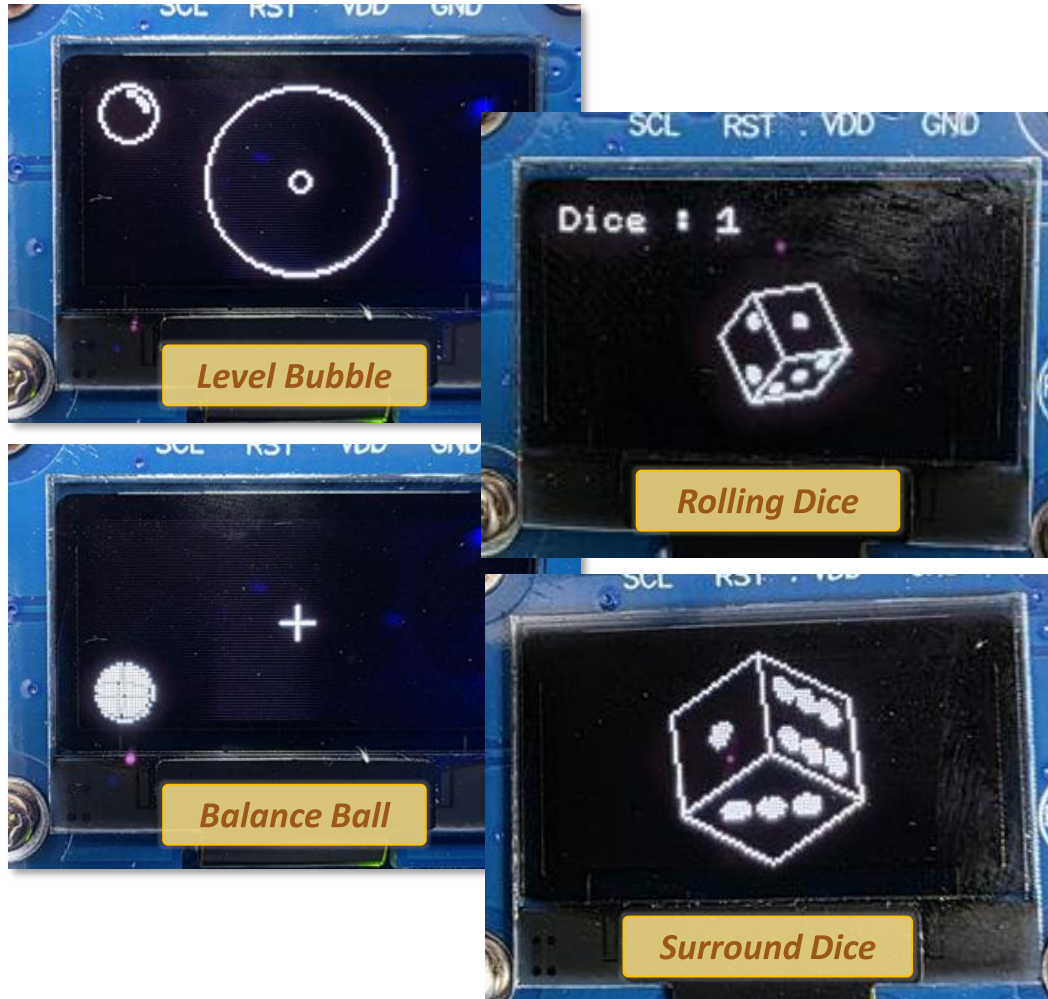
Result (Tera Term)

```
COM7 - Tera Term VT
File Edit Setup Control Window Help
Hello World!!
VR1 Value : 2480
Temperature Value : 956
Lightness Value : 1466
Received Data : m
Accel = ( -22, -13, +2061)
Gyro = ( -9, +14, -5)
ICM42670P Software Reset Completed
ICM42670P WHO AM I : 0x67
ICM42670P Initial Completed
```



⚙️ Lab16 Motion Sensor Application

Result (Application on OLED)



Appendix : Motion Application

MotionApp

Motion Application

Static Functions

- **_AlignFaceToUp** : Returns a rotation matrix to rotates the object face up.
- **_EstimateYawFromMatrix** : Computes the yaw angle based on the rotation matrix.
- **_GetTopFaceIndex** : Return the index (0~5) of the dice face currently facing up (+Z)
- **_ClearZBuffer** : Clears the Z-buffer by setting all values to infinity.
- **_IsFaceVisible** : Checks if polygonal face of a 3D object is facing towards.
- **_CubeFaceUpRotation** : Smoothly aligns the cube's rotation for current top face points upward.
- **_UpdateObjectRotation** : Updates object rotation using XY velocity as axis.
- **_UpdateCameraRotation** : Updates camera's rotation matrix using Accel & Gyro input.
- **_GravityObject** : Simulates gravity-driven object using accelerometer input.
- **_DrawLatitudeLine** : Draws a latitude circle on the ball.
- **_DrawLongitudeLine** : Draws a longitude circle on the ball.
- **_DrawBallLatLonLines** : Rotates ball and draws equator + 2 meridians.
- **_FillTriangleZBuffer** : Fill a triangle into the Z-buffer only (no color drawing)
- **_DrawLineWithZBuffer** : Draw a line with per-pixel Z-buffer depth testing.
- **_DrawDiceDotsOnFace3D** : Draw dice dots on a cube face using Z-buffered.
- **_DrawCube** : Draw 3D cube with hidden line removal using Z-buffer.

Motion Application

Interface Functions

void DrawLevelBubble(short ax, short ay, bool reset, uint8_t layer)

- **Description:**

- Draws a horizontal bubble level simulation with centering force based on g-sensor ax, ay input.

void DrawBalanceBall(short ax, short ay, bool reset, uint8_t layer)

- **Description:**

- Draws a vertically reactive ball with bounce physics based on g-sensor ax, ay input.

- **Parameter:**

- ax, ay : Accelerometer readings for tilt direction.
- reset : If true, resets ball position and velocity.
- layer : Graphic layer to draw the ball and reference lines.

- **Example:**

- DrawLevelBubble(64, 32, false, LAYER_GRAPHIC);
- DrawBalanceBall(100, 50, true, LAYER_GRAPHIC);

Motion Application

Interface Functions

void DrawRotationCube(short ax, short ay, bool reset, uint8_t layer)

- **Description:**

- Draws a rotation cube with bounce physics based on g-sensor ax, ay input.

void ViewRotationCube(short ax, short ay, short az, short gx, short gy, short gz, bool reset, uint8_t layer)

- **Description:**

- View a 3D cube with six faces and rotation based on accelerometer ax, ay, az and gyroscope gx, gy, gz input.

- **Parameter:**

- ax, ay, az : Accelerometer readings for tilt direction.
- gx, gy, gz : Gyroscope readings for roll/pitch/yaw direction.
- reset : If true, resets ball position and velocity.
- layer : Graphic layer to draw the ball and reference lines.

- **Example:**

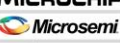
- DrawRotationCube(64, 32, false, LAYER_GRAPHIC);
- ViewRotationCube(0, 0, 0, true, LAYER_GRAPHIC);

Microchip Trademarks

<https://www.microchip.com/en-us/about/legal-information/microchip-trademarks>

Trademark	Description/Appropriate Nouns	Notes
ClockWorks™	configurator, device, family, product, series	US only
CryptoSecure™	security	
CryptoAuthentication™	development library, device, family, IC, library	
CryptoAutomotive™	development kit, security IC, technology	
CryptoCompanion™	chip	
CryptoController™	solution, TPM, trusted platform module	
CryptoMemory®	cryptographic security IC, IC	
Cryptosafe®	device, transponder	
dsPIC®	DSIC, digital signal controller	
dsPIC®	dsPIC® logo	
dsPICDEM™	demonstration board, development board	
dsPICDEM.net™	board	
Dynamic Averaged Matching™ (DAM™)	architecture	
ECAN™	solution, technology	
Espresso T15™	device	
EtherGREEN™	family, platform, technology	
EtherSynch™	family, platform, technology	US only
EyeOpen™	cable compensation	
Flashtec™	controller, family, product	US only
FlexPWS™	technology	
Frequency on Demand®		Registered in other countries (not US)
Genie™	controller, sensing, technology	
GridTime™	device, OASIS time server, time server	
HELDO™	family, regulator	
Hyper Speed Control®	architecture, family	US only
HyperLight Load™	mode	US only
In-Circuit Serial Programming™ (ICSP™)	programming capability	
IsaCircuit™	dual MOSFET bridge inverter, rectifier	
ISCI™	board, demonstration board	
ISOC™	device, family, FPGA, series	
INCE™	sniffer, technology	
Note: *Always check with the Legal Department for the most current trademarks.		

Trademark	Description/Appropriate Nouns	Notes
IntDrive™	driver board, integrated solution, power module	
Intelligent Paralleling™	technology	
IntelliMOS®	family, power stage family	US only
Inter-Chip Connectivity™	technology	
JitterBlocker™	attenuator, family	
JukeBlox®	platform, technology	
jukeblox®	jukeblox® registered word mark with a design Image is not registered; this is a graphic treatment with the registered word mark	
KeeLoq®	protocol, security IC, technology	
Kleer™	technology	
KLEER™	Kleer® registered word mark with a design Image is not registered; this is a graphic treatment with the word mark	
Knob-on-Chip™ (Knob™)	device, family, HMI, knob designer, position wizard, solution, technology, touch controller, user interface, widget, configurator	
LANCheck®	online design review, online review, review	
Libero®	design file, designer, ICSE, SOC design suite, software, tool	US only
LINKM®	cable diagnostics, diagnostics engine, family, technology	
Marginal™	signal integrity testing	
maxCrypto™	controller-based encryption, encryption	
maxStyle®	solution	
maxTouch®	family, studio, technology, touchscreen controller	
maxtwin™	driver, storage manager, tool	
ModuS®	bus, controller, device	
megaBuck™	device, MCU, microcontroller	
memBlox™	neuromorphic memory product, product, solution, technology	
Microchip	Microchip logo	Please do not use this logo; always use the Microchip name and logo.
Note: *Always check with the Legal Department for the most current trademarks.		

Trademark	Description/Appropriate Nouns	Notes
	Microchip name and logo	This is the vertical representation; the horizontal representation is also available.
	Microsemi logo	Do not use
Microsemi®		Do not use
Mind™	analog simulator, analog simulator tool	
Mini™	application, coordinator, mesh, network, protocol, software, stack, wireless networking protocol	
MOST®	board, cooperation, design, device, evaluation kit, NetServices, network, platform, product, specification, system, technology	
MOST®	MOST® logo	
MotorBeacon®	development suite	US only
MPASQ™	assembler	
MPK™	device, product	
MPLAB®	Notes: C compiler, compiler, development ecosystem, ICD, ICE, IDE, PE, in-circuit debugger, in-circuit emulator, integrated development environment, integrated programming environment, library, programmer, server, starter kit, universal device programmer, XC compiler Tool Names: All Edges, Analog Designer, Analysis Tool Suite, Code Configurator, Connect Configurator, Data Visualizer, Discover, Extensions, Harmony, Mind™, Network Creator, PICkit™, Programmer-to-Go, Programmer-to-Go Mobile App, PPS, SoC Power Simulator, Snap, Xpress	
	MPLAB® Certified logo	
MPLIB™	libraries, object libraries	
Note: *Always check with the Legal Department for the most current trademarks.		

Trademark	Description/Appropriate Nouns	Notes
MLINK™	linker, object linker	
mSC™	bare die, diode, gate driver, module, MOSTET product, solution, technology	
miTouch™	development kit, evaluation kit, library, sensing solution, solution, technology	US only
MultiTRAM™	technology	
NetDetach™	technology	
Noninvasive Code Generation™ (NCG)	compilation technology	
OptoLynx™	network analysis tool, studio, tool	
pic®	assembler, device, MCU, microcontroller	
	PIC32 logo	
	PIC64 logo	
PICDEM™	demonstration board	
PICDEM.net™	Internet/Ethernet demonstration board	
PICkit™	debugger, development tool, in-circuit debugger, on-board, programmer, starter kit, tool	
picPower®	technology	
PICSTART®	development programmer	
PICtail™	board, daughter board	
PolarFire®	device, daughter board	
PowerSmart™	designer, development suite, digital control library	
Precision Edge®	family, product family	US only
ProASIC®	architecture, board, family, FPGA, kit	US only
ProASIC Plus®	architecture, device, family, FPGA, switch	US only
Prochip Designer®	software suite	
PureSilicon™	oscillator, technology	
QMatrix™	device, sensor	
QTouch®	board, compiler, configurator, device, development platform, library, project builder, sensor, solution, technology, tool	
Quiet-Wire™	enhanced EMI technology, family, technology	
Note: *Always check with the Legal Department for the most current trademarks.		

Trademark	Description/Appropriate Nouns	Notes
Single Blocker™	attenuator, family of linear regulation, product line, technology	
SEAL ICE™		
SESA™	FPGA	
SESA™	FPGA	
SAM 9A™	applet, application, host, in-system programmer, ISP, monitor, software	
SAM-ICE™	emulator	
SenGenuity™	product, sensor	
SerDes Quad I/O™ (SQI™)	family, flash device, product	
Silicon Storage Technology®		Japan only
SimpleLink™	protocol	
Simplexity™	device, family, product	
SmartBuck™	device, IC	
SmartFusion®	evaluation kit, FPGA, SoC, System-on-Chip	US only
SmartLink™	compiler, compiler software, IDE, software	
SMART-LS™	technology	
SpyPic®	device	
SST®	Acronym for Silicon Storage Technology	
	SST logo	
A Microchip Technology Company		
STK™	starter kit	
storCue™	encryption technology, technology	
SuperFlash™	device, kit, matrix, memory, technology	
SuperSwitcher™	bus regulator, family, regulator	
SuperSwitcher II™	family	
Switchbox™	family, product line, switch, technology	
Symmetron™		China only
Symmetron™		
SynchroPac™	device, family, product	
SynServer™	clock, device, instrument, oscillator, server	
Note: *Always check with the Legal Department for the most current trademarks.		

Trademark	Description/Appropriate Nouns	Notes
SynWorld™	ecosystem program, timing	US only
Tachyon™	controller, family, platform, product, technology	
The Embedded Control Solutions Company®		US only; should be in italics
TimeExam™	primary frequency reference, product	US only
TimeLink™	platform, system	US only
TimePicks™	platform, software suite, synchronization management system	US only
TimeProvider®	clock, enhanced PRTC, APRTC, extension, system, grandmaster, kit, module, product, series, series, shelf	US only
TimeSource™	PAC, PPS, primary reference clock, primary reference source, shelf, system	
tinyAUX™	MCU, microcontroller	
Total Endurance™	software model	
TSPAB™	controller, touchscreen controller	
Tuning™	hardware security module, program	
UNICOT™	bus, communication protocol, firmware, hardware port, memory chip	
USBCheck™	design review	
VarSense™	technology	
VectorBox™	accelerator, intellectual property, IP, platform, SDK, software development kit, solution, tool, tool chain, tool flow	
Vibron™	oscillator, products	
Vibron™	CT, configuration tool, product, solution, software platform, SP	
Vibron™	cable diagnostics algorithm, cable diagnostics software suite	
ViewSpan™	technology	
WaterLock™	technology	
XMEGA®	Custom Logic (XCL) device, family, MCU, microcontroller, series	
XpressConnect™	retimer	
ZENA™	analyzer, software, wireless adapter	
ZL®		US only
Note: *Always check with the Legal Department for the most current trademarks.		

Trademark	Description/Appropriate Nouns	Notes
	Adaptic "A" logo	Registered in other countries (not US)
Adaptic®	adaptor, cable, controller, driver, family, product, series, software, solution	
Adjacent Key Suppression™	technology	
AgileSwitch™	device, digital programmable gate driver, driver, family, gate driver	US only
AKS™	technology	
Asking for the Digital Age™	APD family tagline	
Any Capacitor™	stable	
AnyPin™	capability, feature, structure	
AnyDuo™	capability, feature	
Augmented Switching™	technology	
AVR®	architecture, CPU, device, family, MCU, microcontroller	
	AVR® logo	
AVR Freaks®	forum	
BackTime™	technology	
BeaTime™	SDK, software development kit	
BlackSky®	firewall, subscription service, technology	
BodyCom™	development kit, system, technology	
chipKIT®	IO shield, board, hardware, network, platform, software	
	chipKIT logo	
ClockStudio™	application, program, software, utility	
Note: *Always check with the Legal Department for the most current trademarks.		