

SAMD21 ARM Cortex-M0+ Microcontroller & Harmony v3 SAM2002 v1.00



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



SMART | CONNECTED | SECURE

**CAE Taiwan Team
Microchip Technology Inc.**

2021 May

Class Objectives

- **Before this class:**
 - Please study SAM2001 firstly to familiar with usage of MPLAB X IDE and PLIB implementation of Harmony v3.
- **When you finish this course, you will be able to:**
 - Familiar with Harmony Core System Service usage.
 - Familiar with Harmony Driver based peripheral usage.
- **There have two level course included.**
 - Major course : General study of basic features.
 - Appendix : Advance features and appendix.

Major Course (1/4)

- Software Request
- SAMD21 EVB APP045 EVB Introduction
- MPLAB® Harmony Key Concept
- Harmony System Service introduction
 - TIME System Service
 -  Lab1 – Delay based LED toggle
 -  Lab2 – Callback based LED toggle
 - CONSOLE System Service
 -  Lab3 –UART Console
 - DEBUG System Service
 -  Lab4 –Debug Message Control
 - COMMAND Processor System Service
 -  Lab5 –Custom Command Console

Major Course (2/4)

- USB Device CDC Library

-  Lab6 – USB Device CDC with CONSOLE System Service

- FILE System Service

-  Lab7 –SD Card with file system

-  Lab8 –FILE System Auto Mount Feature

-  Lab9 –USB Host Mass Storage with file system

-  Lab10 –NVMCTRL Internal Flash with file system

Major Course (3/4)

- Harmony Drivers introduction
 - PLIB Limitation
 - Driver Advantage
 - Driver Usage Models
 - How to use Driver in Harmony?
 - Transfer Request Callback Restriction
 - I2C Driver API and Usage Example
 - Driver Execution Model

Major Course (4/4)

- **USART Driver Library**

-  Lab11 – Single-Client UART Communication

-  Lab12 – Multi-Instance UART Communication

- **I2C Driver Library**

-  Lab13 – Single-Client I2C Digital Temperature Sensor

-  Lab14 – Multi-Client I2C Serial EEPROM

-  Lab15 – Multi-Instance I2C G-Sensor

-  Advanced Exercise: MPLAB Data Visualizer

-  Advanced Exercise: I2C Device Auto Probe

- **SPI Driver Library**

-  Lab16 – Single-Client SPI OLED Display

- **SPI Driver - Wi-Fi WINC Driver Library**

-  Lab17 – AP scan application of SPI Wi-Fi Module

- **SPI Driver - SD Card Driver Library**

-  Lab18 – Manage SPI SD card without File System

- **MEMORY Driver Library**

-  Lab19 – Manage NVM Flash without File System

Soft Request In this Course

- **MPLAB® X IDE v5.50** <https://www.microchip.com/mplab/mplab-x-ide>
- **MPLAB® XC32 v3.01** <https://www.microchip.com/mplab/compilers>
- **MPLAB® Harmony 3 Launcher v3.6.4** (From X IDE plugin)
- **MPLAB® Harmony Framework** (From X IDE Harmony 3 Content Manager)
 - core v3.9.2
 - csp v3.9.1
 - mhc v3.8.1
 - dev_packs v3.9.0
 - usb v3.8.0
 - wireless_wifi v3.5.1
- **SAMD21 DFP v3.4.116** (Device Firmware Pack) (From X IDE \Tools\Packs)
- **CMSIS v5.4.0** (Common Microcontroller Software Interface Standard)

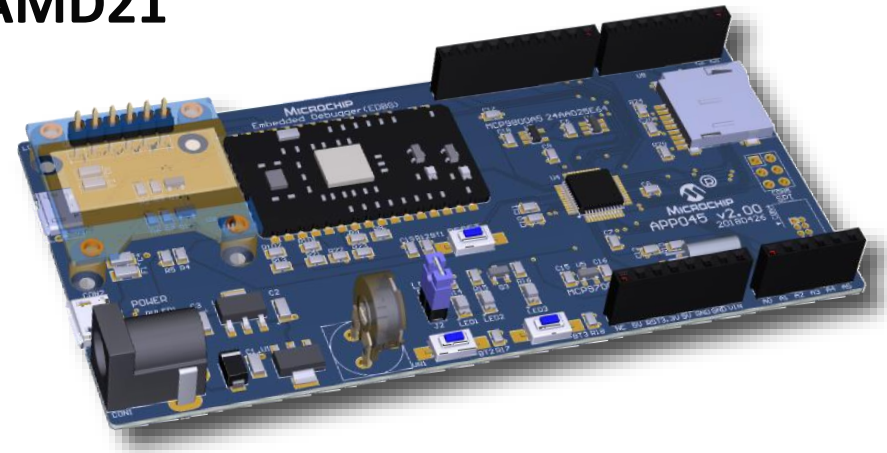
(Lasted Update : 2021/09/17)

SAMD21 EVB APP045 v2.00

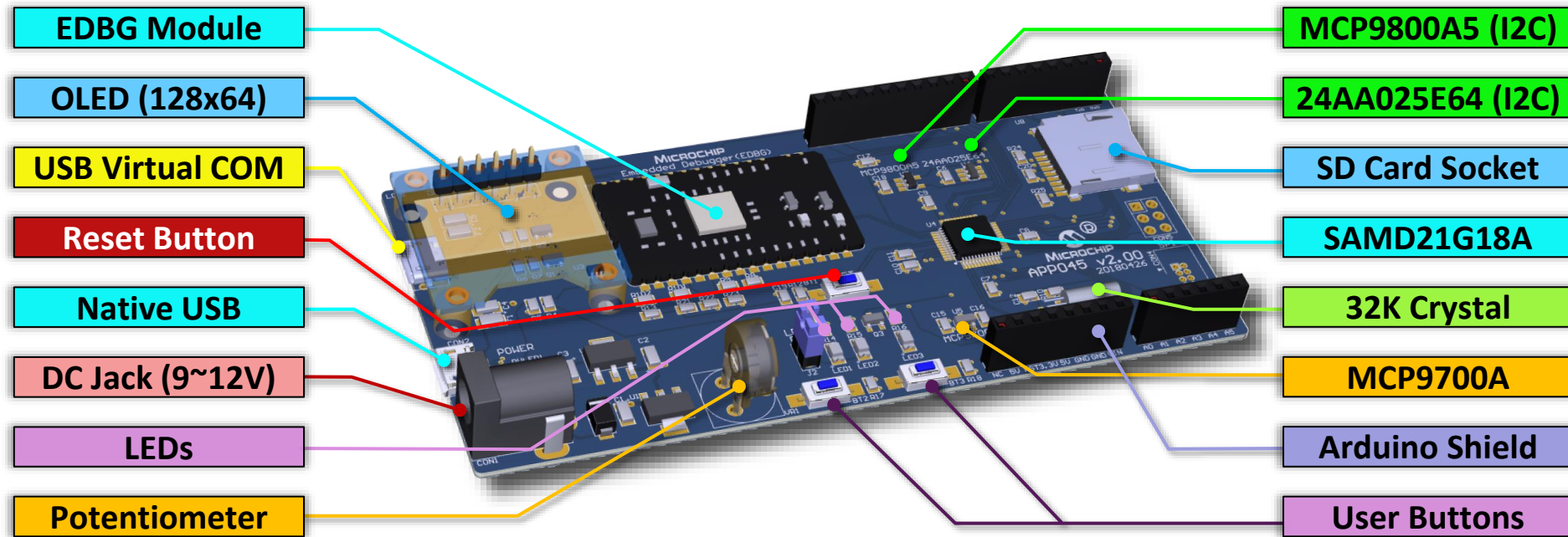
Introduction

APP045 v2.00 EVB Features

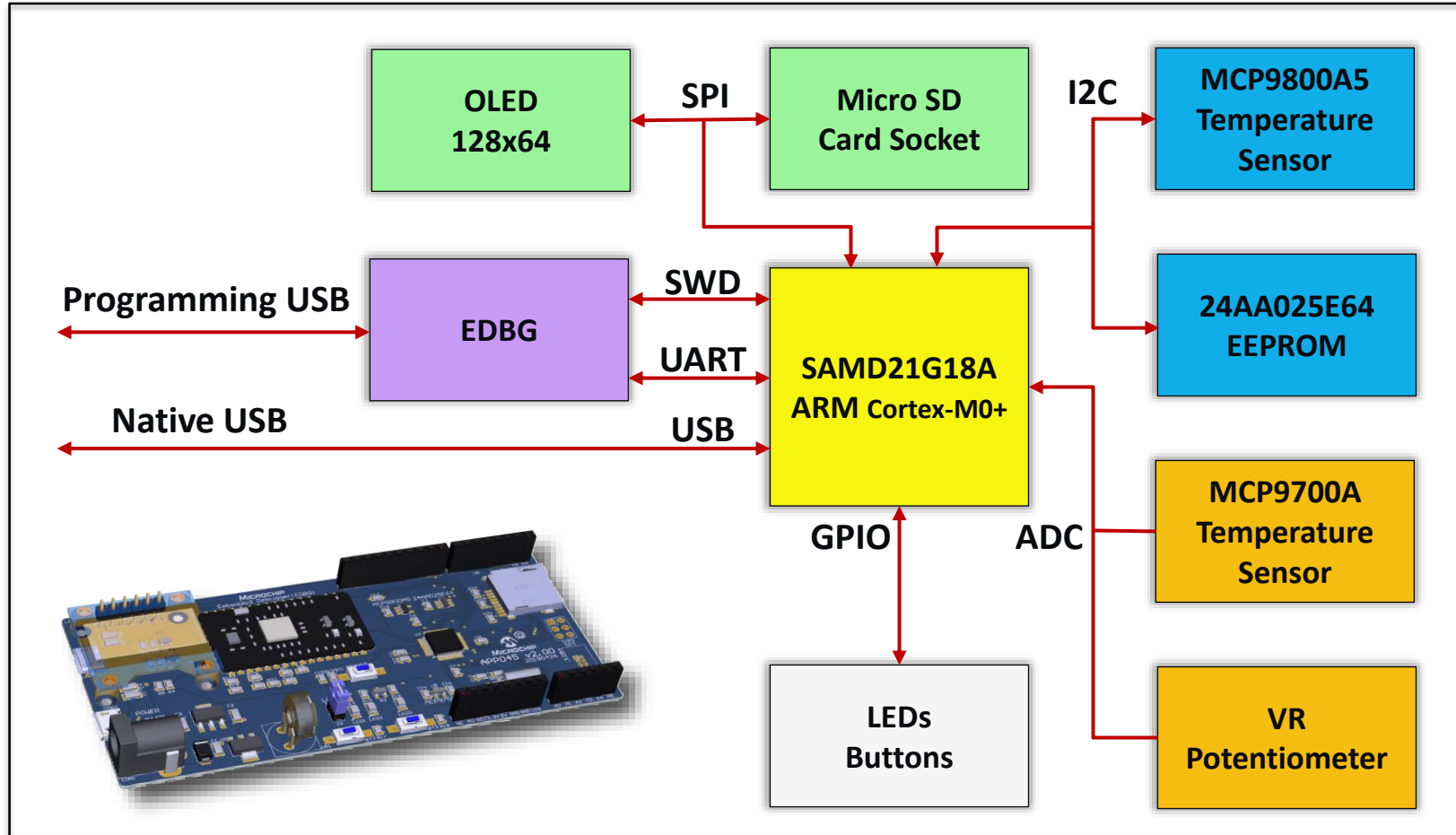
- On board **SAMD21G18A** 32Bit ARM Cortex M0+ Microcontroller
- Built-in **Embedded Debugger (EDBG)** module
Provide Programming & Debugging function for SAMD21
Provide Virtual COM port (CDC bridge)
- **Multiple Power Supply Source**
Programming USB Port
Native USB Port
DC Jack (DC 9 ~ 12V)
- **2 x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



APP045 v2.00 Overview



APP045 v2.00 Block Diagram

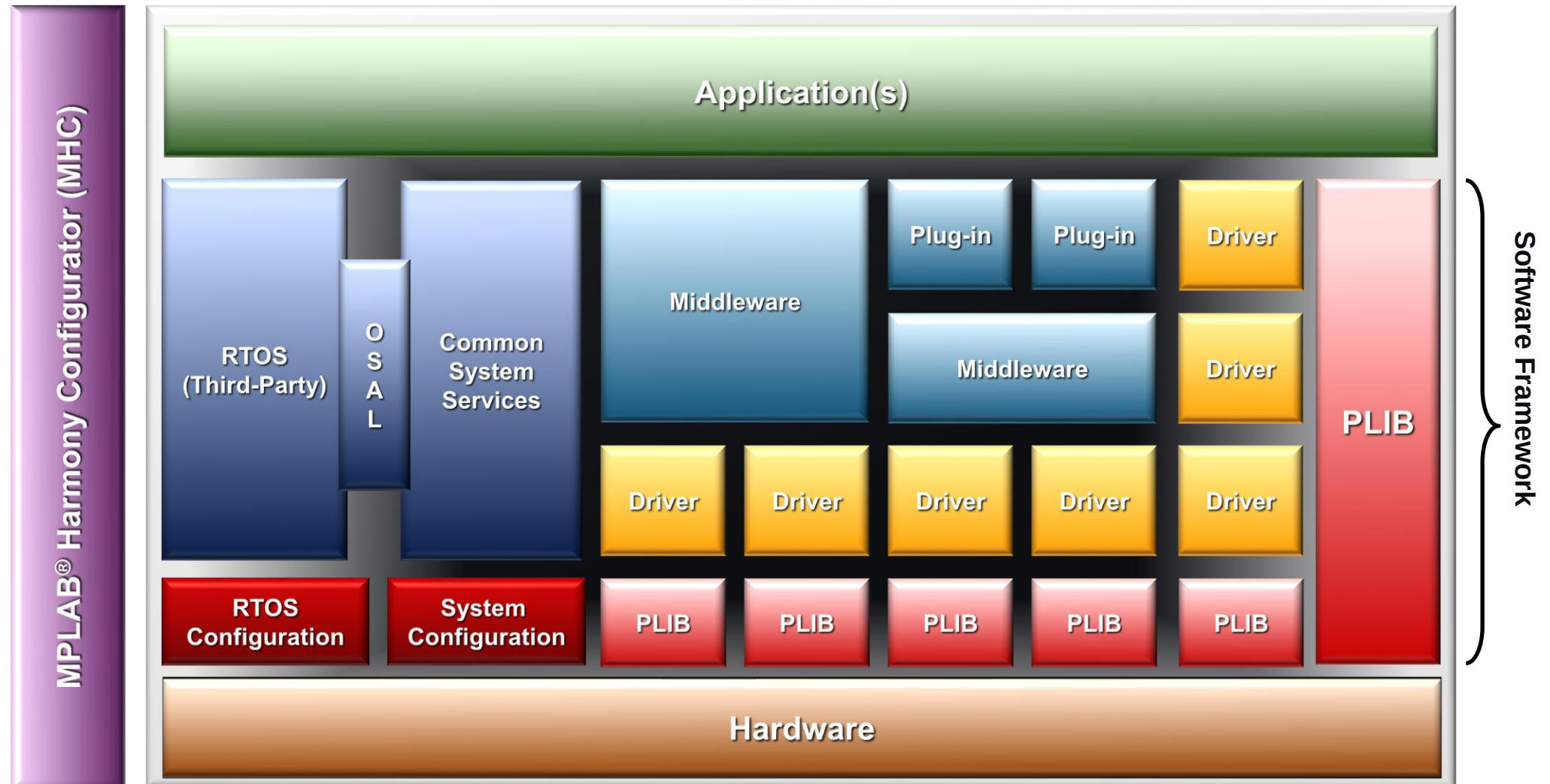


MPLAB® Harmony Key Concept

MASTERS 19 DEV8 referenced

MPLAB® Harmony Key Concept

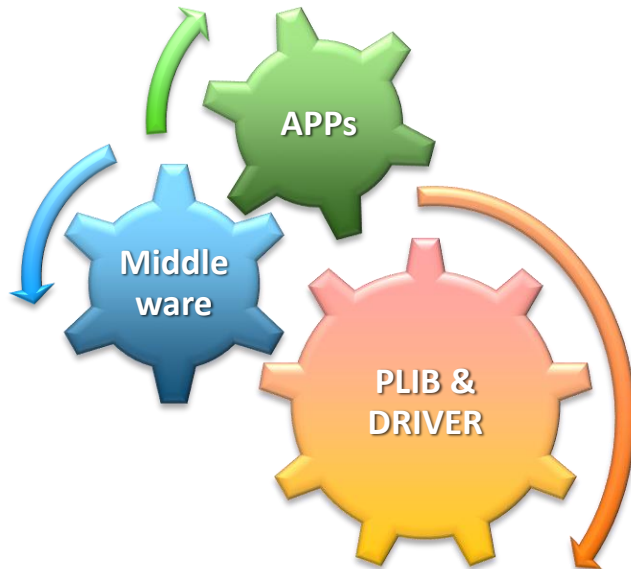
Architecture



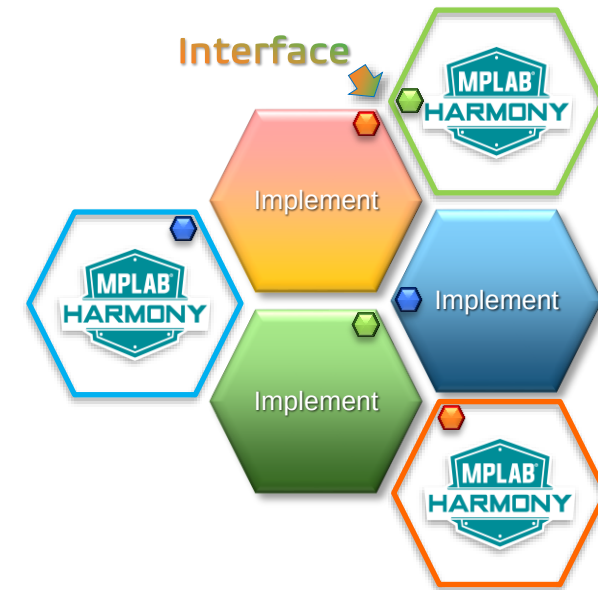
MPLAB® Harmony Key Concept

Modularity and Compatibility

- Implements a single functionality
- A module has an interface and an implementation
- Hides implementation from the user
- Modules interact through interface
- No access to internal data



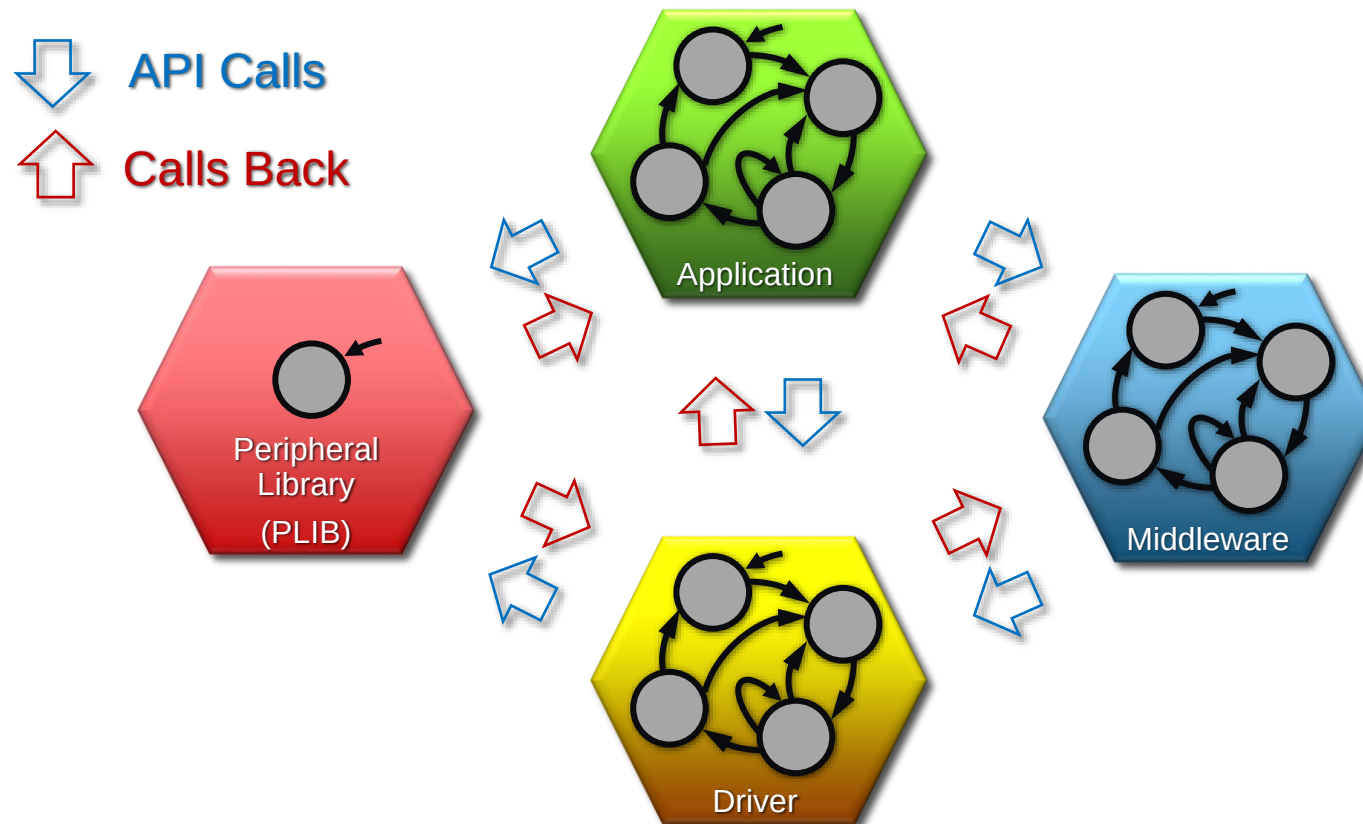
- Modules are compatible
- Adding new modules does not break existing functionality
- State machine ready for RTOS



MPLAB® Harmony Key Concept

Compatible State Machines

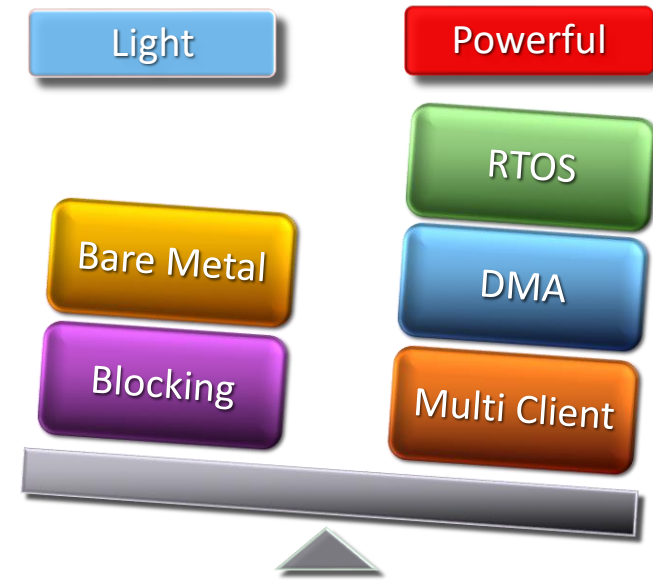
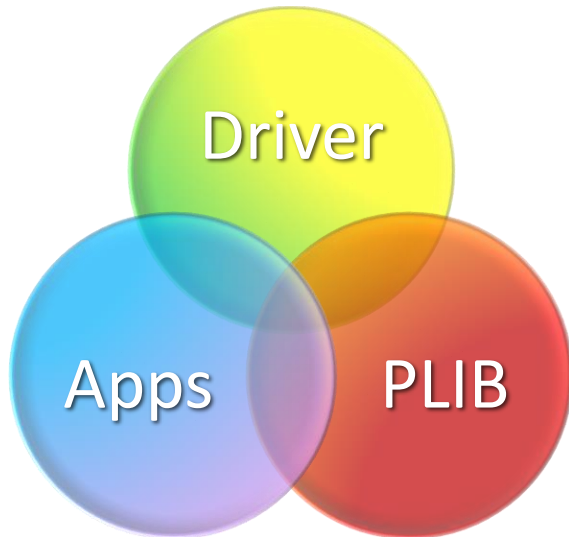
- Modules interact with each other through Interface Functions.
- Effectively Independent.
- No assumptions of sequence or timing between modules.



MPLAB® Harmony Key Concept

Configurability and Flexibility

- Single-client or multi-client
- Bare-metal or RTOS
- Blocking or Non-blocking
- DMA or Non-DMA
- Other library specific options



- Use only PLIBs or combination of drivers and PLIBs
- Porting the project from one folder/machine to another
- Modular download

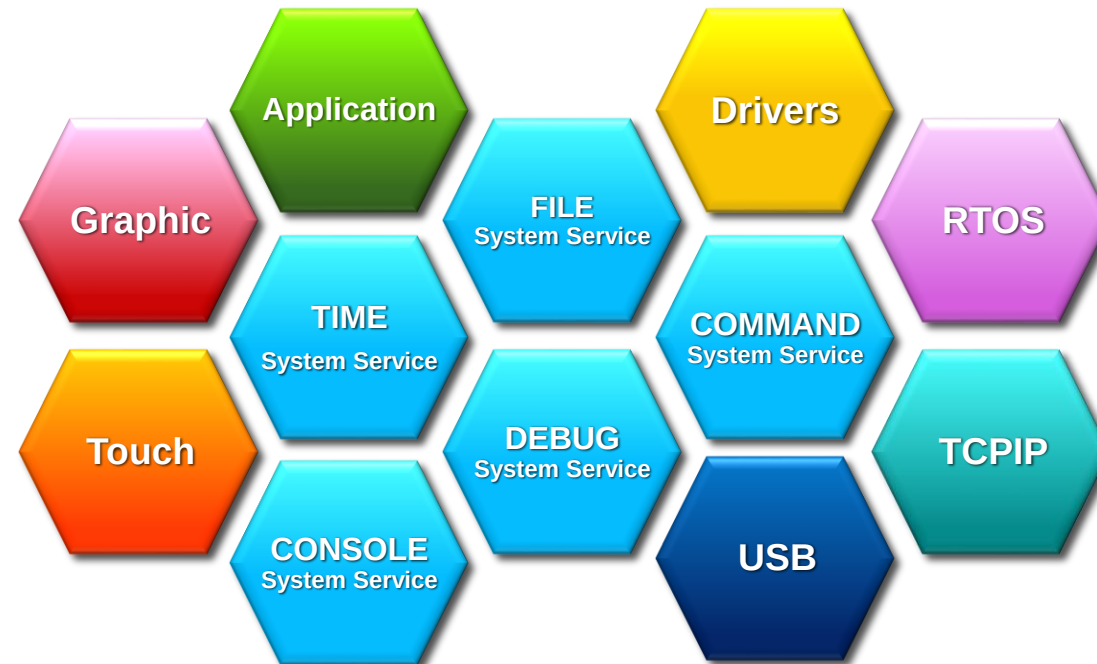
Harmony System Service Introduction

Manage the Common and Shared Resources

Harmony System Service Introduction

Manage the Common and Shared Resources

- Provides common functionality required by different drivers or library modules.
- Manages shared resources.
- Eliminates potential conflicts.
- Keeps the requests separate.



Harmony System Service Introduction

- **TIME system service**
- **CONSOLE system service**
- **DEBUG system service**
- **COMMAND processor system service**
- **FILE SYSTEM service**

TIME System Service

Harmony System Service

TIME System Service

- **Introduction**

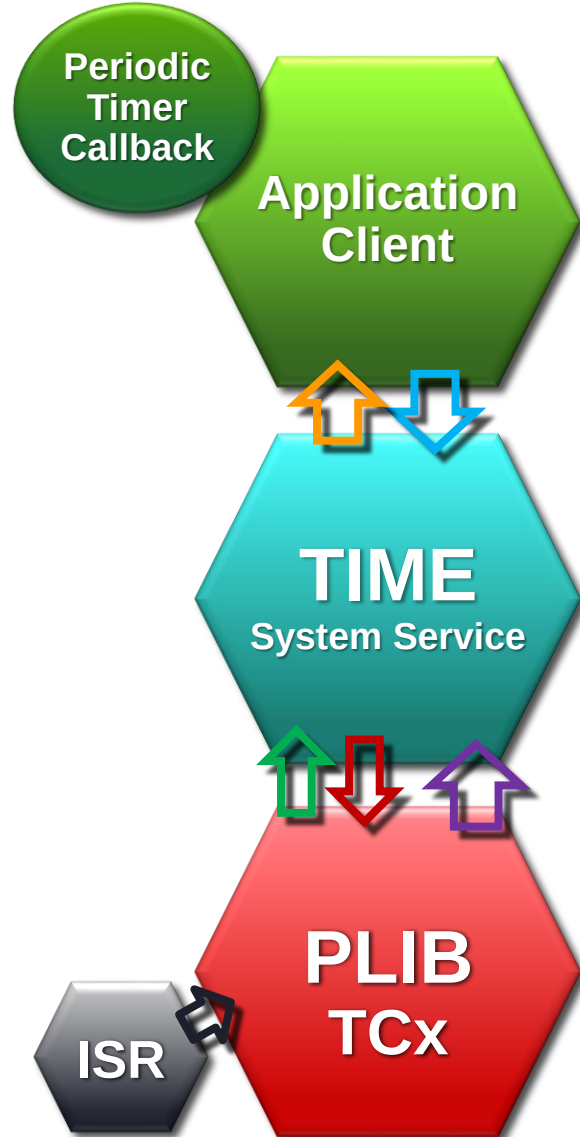
The Time System Service Library can provide periodic or one-shot alarm and delays to the user.

It uses a hardware timer peripheral library for providing its timing services.

- **Key Features:**

- Periodic notifications
- One-shot/single notification
- Delays
- Support for both tick-based and tick-less modes
 - Tick-less mode:
Instead of getting interrupted from the hardware timer periodically, the Time System Service manages the hardware timer so as to generate an interrupt only when required and at appropriate times.
 - Tick-based mode:
A periodic interrupt is generated by the hardware timer. The tick-rate is configurable by the user and defines the resolution for all the timers.

TIME System Service



Tick-less and Tick-based

- Request for periodic or non-periodic timeouts
- Set next compare match value (tick-less)
- Timer Compare Interrupt (tick-less)
- Timer Period Interrupt (tick-based)
- Application Callback (from interrupt context)

TIME System Service

- Harmony Configuration

The image displays the Harmony IDE configuration for the TIME System Service. It includes two side-by-side configuration windows and a diagram of the service and its consumers.

Configuration Options* (Left):

- TIME
 - Use SysTick? ☐
 - PLIB Used TC3
 - Number of Clients 5
 - Operating Mode TICK BASED
 - Tick Rate (ms) 1
 - Use Floating Point Calculations? ☐
 - Achievable Tick Rate Resolution (ms): 1.0ms

Configuration Options* (Right):

- TIME
 - Use SysTick? ☐
 - PLIB Used TC3
 - Number of Clients 5
 - Operating Mode TICKLESS

Diagram:

The diagram shows the TIME System Service (Core Service) connected to the TC3 Peripheral Library (TMR). The TIME System Service is labeled with a green dot and the text "TIME System Service", "Core Service", and "SYS_TIME". The TC3 Peripheral Library is labeled "TC3 Peripheral Library" and "TMR".

Available Consumers:

- sys_time (SYS_TIME) Generic Capability
- Current Consumers
- Available Consumers

Consumer List:

- AmazonHWInterface (AmazonHWInterface)
- SD Card (SPI) (drv_sdspi)
- External Controller (gfx_driver_external)
- ILI9488 (gfx_driver_ili9488)
- SSD1309 (gfx_driver_ssd1309)
- SSD1963 (gfx_driver_ssd1963)
- LE External Controller (le_gfx_driver_external)
- LE ILI9488 (le_gfx_driver_ili9488)
- LE SSD1963 (le_gfx_driver_ssd1963)
- wolfCrypt Library (lib_wolfcrypt)
- TCPIP CORE (tcpipStack)
- Host Layer (usb_host)

TIME System Service

API functions of the Time System Service library.

- **System Functions**

SYS_TIME_Initialize	Initializes the System Time module.
SYS_TIME_Status	Returns System Time status.
SYS_TIME_Deinitialize	Deinitializes the specific module instance of the SYS TIMER module

- **Timer Functions**

SYS_TIME_TimerCreate	Creates and initializes a new 32-bit software timer instance.
SYS_TIME_TimerDestroy	Destroys/deallocates a software timer instance.
SYS_TIME_TimerStart	Starts a software timer running.
SYS_TIME_TimerStop	Stops a running software timer.
SYS_TIME_TimerCounterGet	Gets the elapsed counter value of a software timer.
SYS_TIME_TimerPeriodHasExpired	Reports whether or not the current period of a software timer has expired.
SYS_TIME_TimerReload	Reloads (or reinitializes) the software timer instance.

- **Callback and Delay Functions**

SYS_TIME_CallbackRegisterMS	Registers a function with the time system service to be called back when the requested number of milliseconds has expired (either once or repeatedly).
SYS_TIME_CallbackRegisterUS	Registers a function with the time system service to be called back when the requested number of microseconds have expired (either once or repeatedly).
SYS_TIME_DelayMS	This function is used to generate a delay of a given number of milliseconds.
SYS_TIME_DelayUS	This function is used to generate a delay of a given number of microseconds.
SYS_TIME_DelaysIsComplete	Determines if the given delay timer has completed.

TIME System Service

API functions of the Time System Service library. (cont.)

- **Counter and Conversion Functions**

<code>SYS_TIME_Counter64Get</code>	Get the common 64-bit system counter value.
<code>SYS_TIME_CounterGet</code>	Get the common 32-bit system counter value.
<code>SYS_TIME_CounterSet</code>	Sets the common 32-bit system counter value.
<code>SYS_TIME_CountToMS</code>	Converts a counter value to time interval in milliseconds.
<code>SYS_TIME_CountToUS</code>	Converts a counter value to time interval in microseconds.
<code>SYS_TIME_MSToCount</code>	Convert the given time interval in milliseconds to an equivalent counter value.
<code>SYS_TIME_USToCount</code>	Convert the given time interval in microseconds to an equivalent counter value.
<code>SYS_TIME_FrequencyGet</code>	Gets the frequency at which the hardware timer counts.

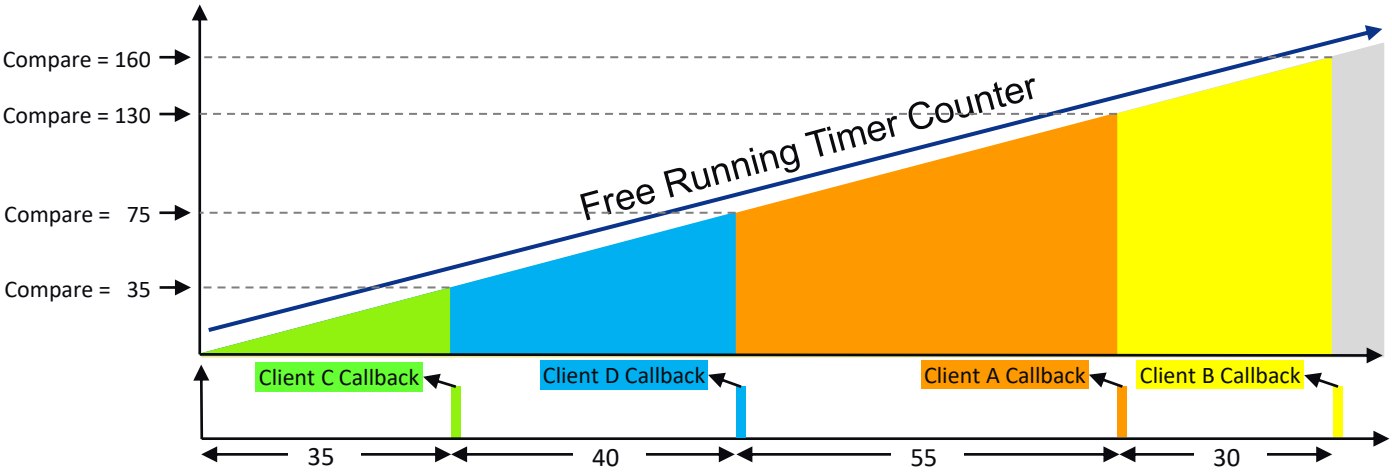
- **Data Types and Constants**

<code>SYS_TIME_INIT</code>	Defines the data required to initialize the TIME system service
<code>SYS_TIME_CALLBACK</code>	Pointer to a time system service callback function.
<code>SYS_TIME_CALLBACK_TYPE</code>	Identifies the type of callback requested (single or periodic).
<code>SYS_TIME_RESULT</code>	Result of a time service client interface operation.
<code>SYS_TIME_HANDLE_INVALID</code>	Invalid handle value to a software timer instance.
<code>SYS_TIME_HANDLE</code>	Handle to a software timer instance.

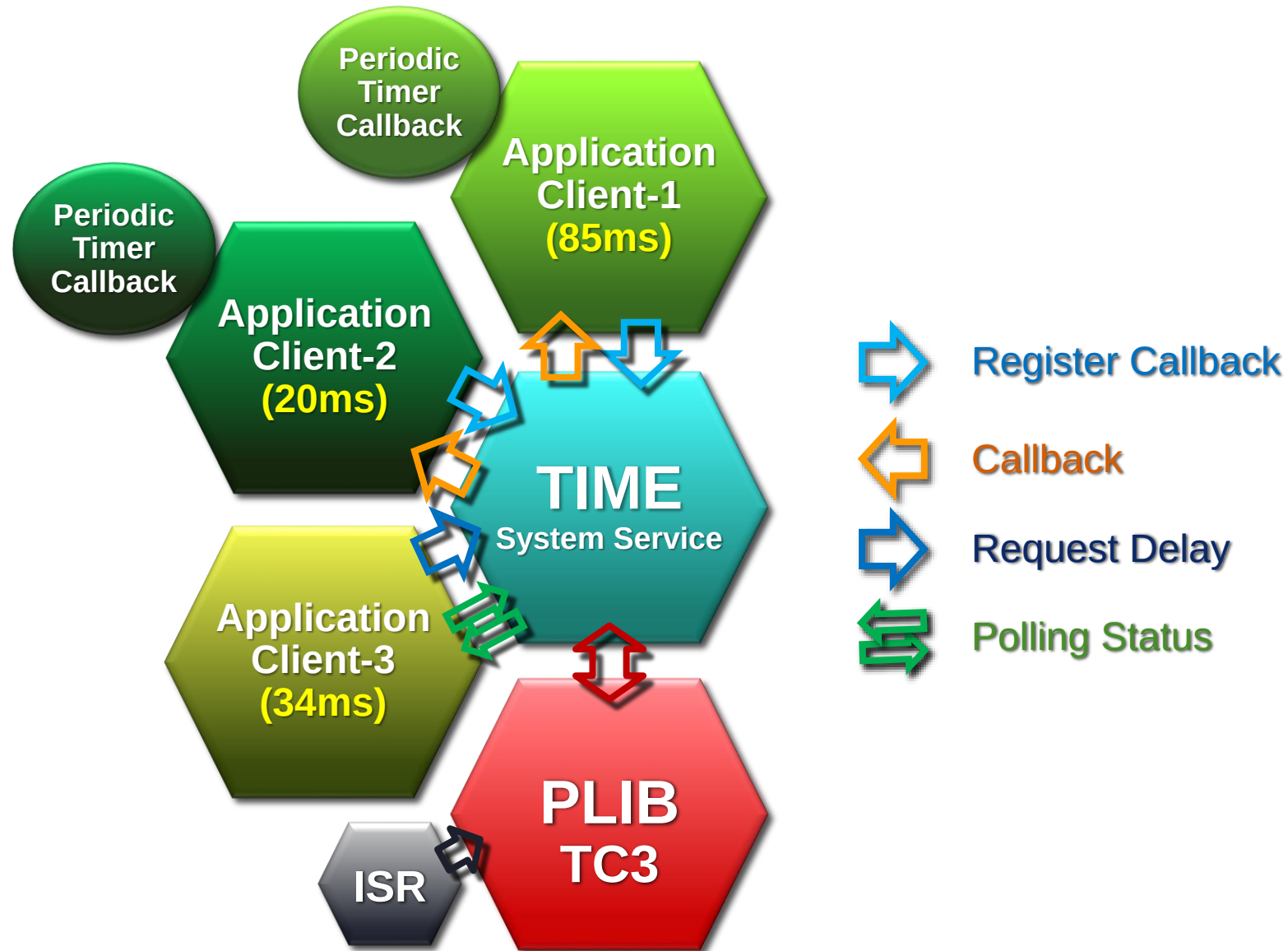
TIME System Service

- No periodic timer tick, reduces overhead of periodically servicing interrupt.
- Provide time resolution in terms of hardware time clock.
- CPU can stay in low power and wakeup only when its time to act.

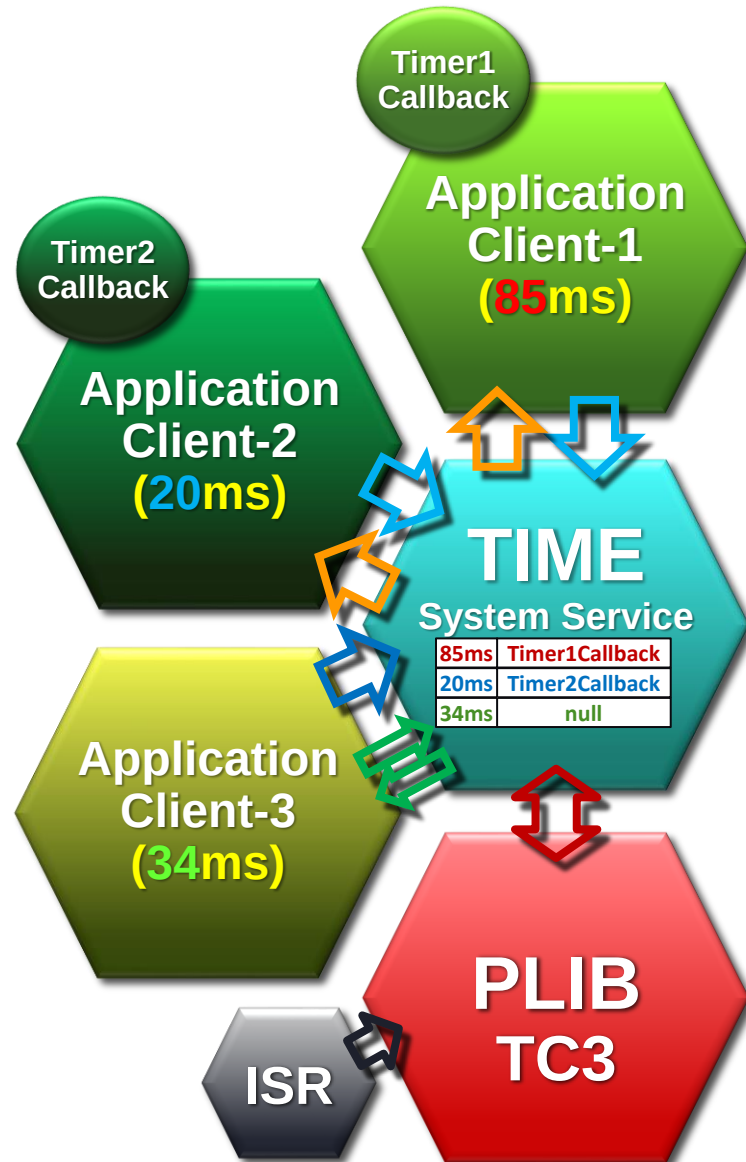
Client Request	Timer Linked List					Timer Compare
Client A : 130ms	130					Compare = 130
Client B : 160ms	130	30				Compare = 130, 160
Client C : 35ms	35	95	30			Compare = 35, 130, 160
Client D : 75ms	35	40	55	30		Compare = 35, 75, 130, 160



TIME System Service



TIME System Service



```
void Timer1Callback(...)
{
    Timer1Expired = true;
}

void Timer2Callback(...)
{
    Timer2Expired = true;
}

void App1_Task()
{
    SYS_TIME_CallbackRegisterMS(Timer1Callback, 0, 85,
                                SYS_TIME_PERIODIC);
}

void App2_Task()
{
    SYS_TIME_CallbackRegisterMS(Timer2Callback, 0, 20,
                                SYS_TIME_PERIODIC);
}

void App3_Task()
{
    SYS_TIME_HANDLE tmrHandle;

    SYS_TIME_DelayMS(34, &tmrHandle);

    while(1)
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true )
        {
            state = APP_DELAY_COMPLETE;
        }
    }
}
```

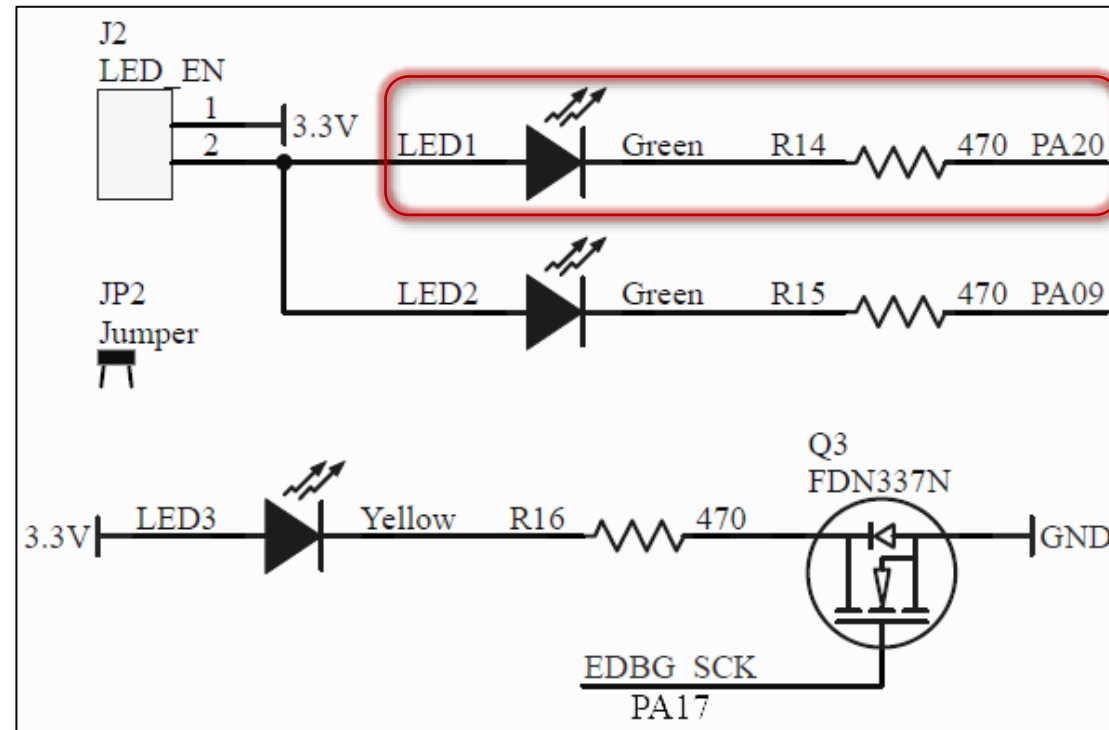
Lab 1

Delay based LED toggle

TIME System Service

⚙️ Lab1 Delay based LED toggle

- Try add TIME system service to create a **delay** for toggle **LED1** in **one second** period.
- The **LED1** is **low active** on pin **PA20**.
- Select **TC3 PLIB** as timer source of your **TIME system service**.



⚙️ Lab1 Delay based LED toggle

- Create a new project of Harmony v3

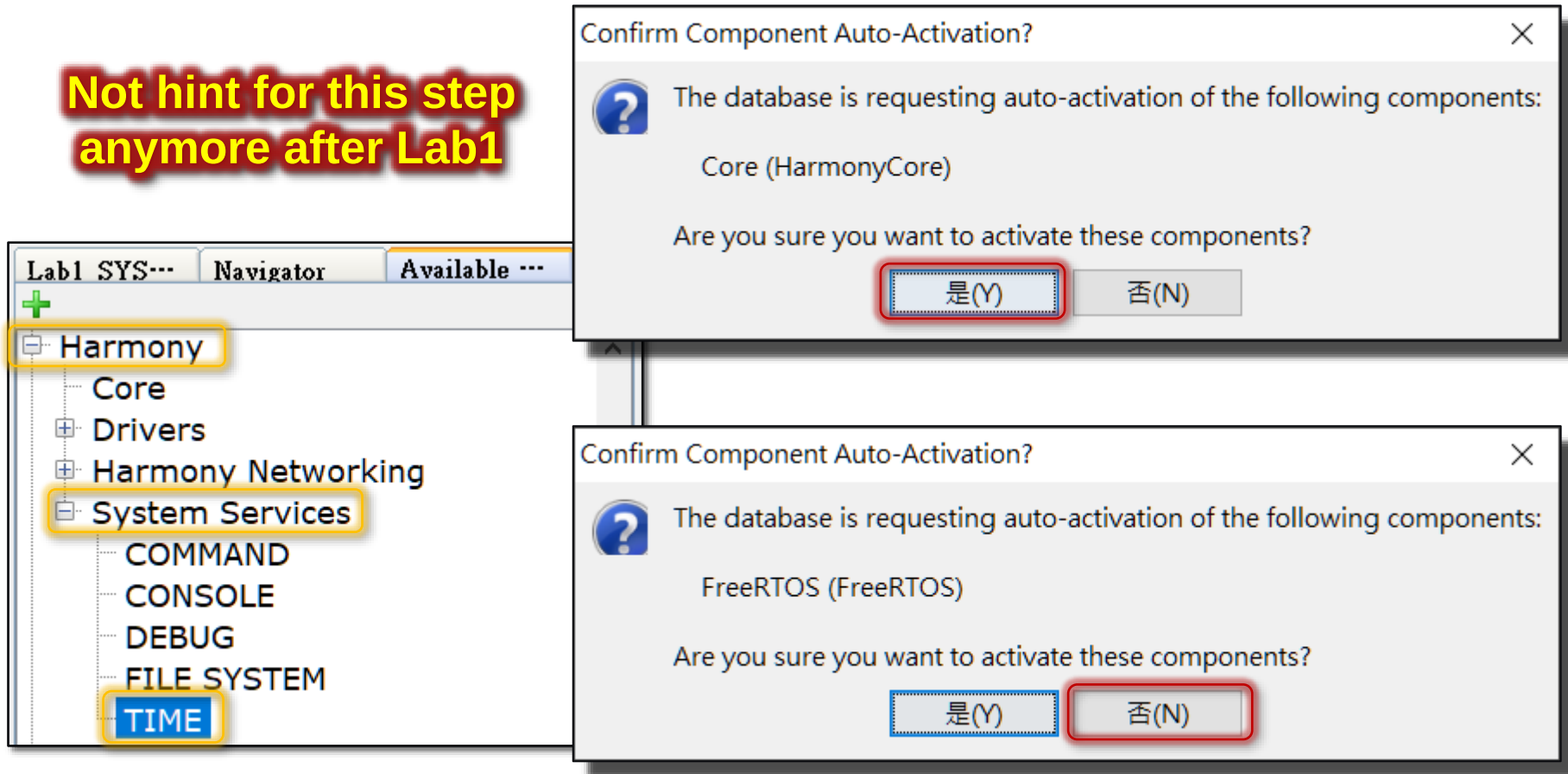
Framework Path : C:\HarmonyFramework
Location : C:\Exercises\Lab1_SYS_TIME_Delay
Folder : Lab1_SYS_TIME_Delay
Target Device : ATSAM21G18A

Not hint for this step anymore after Lab1

⚙️ Lab1 Delay based LED toggle

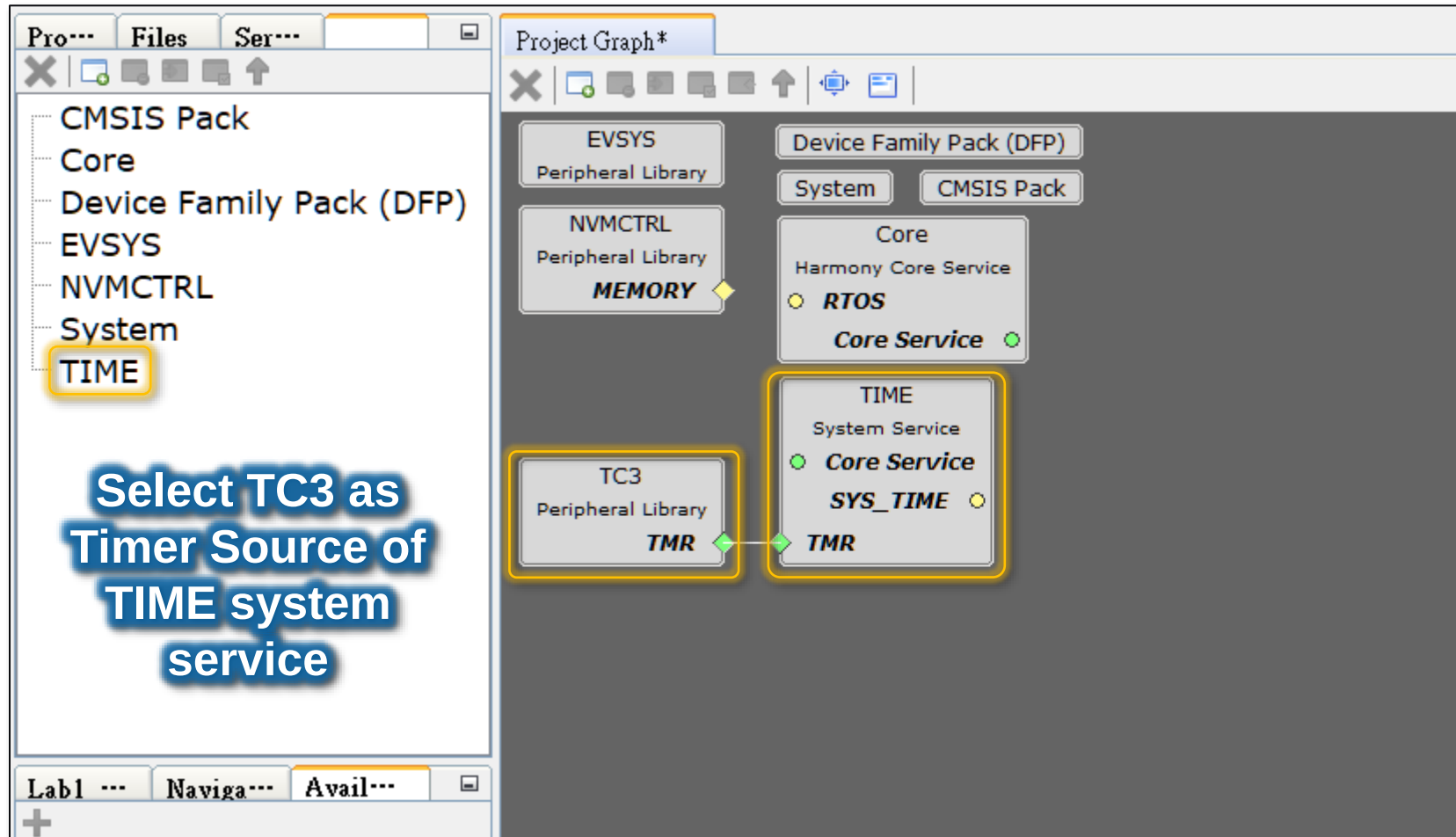
- Harmony Configuration (Add TIME system service)

**Not hint for this step
anymore after Lab1**



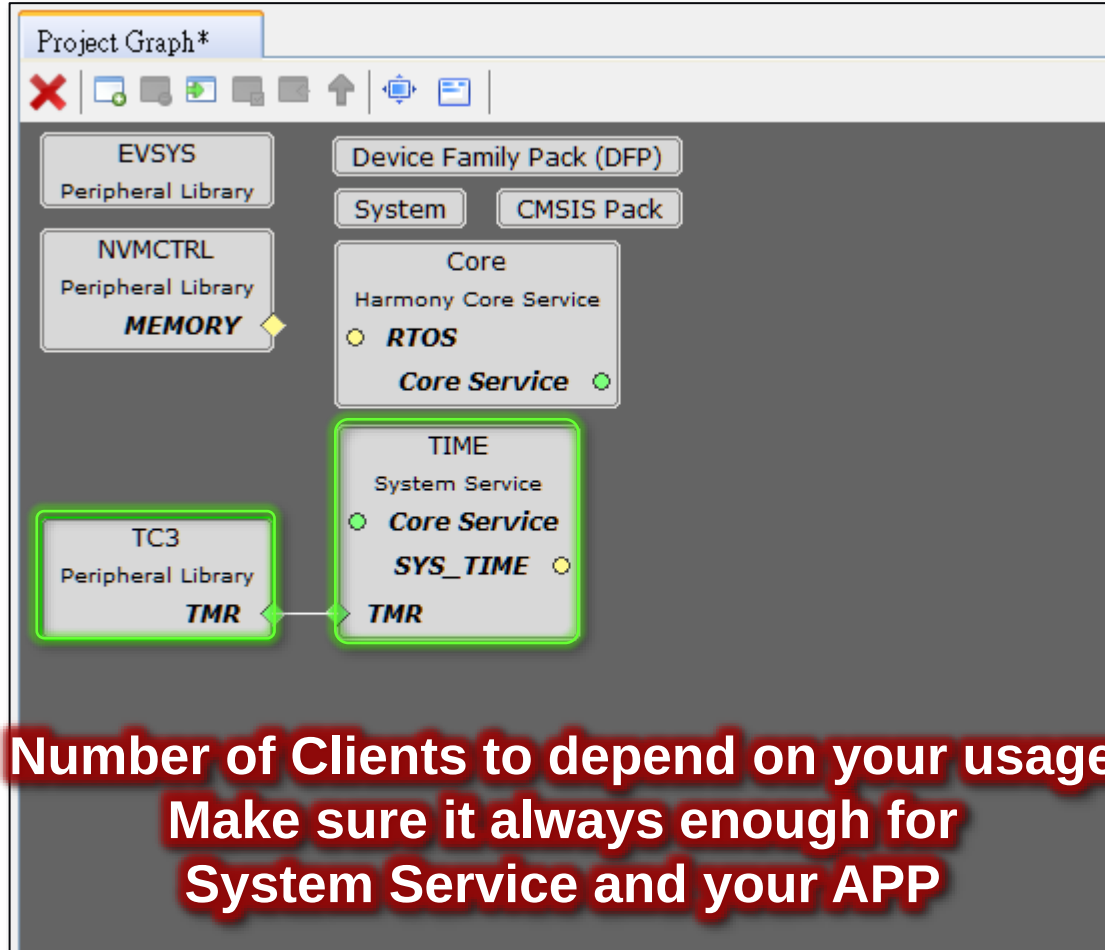
⚙️ Lab1 Delay based LED toggle

- Harmony Configuration (Add TIME system service)

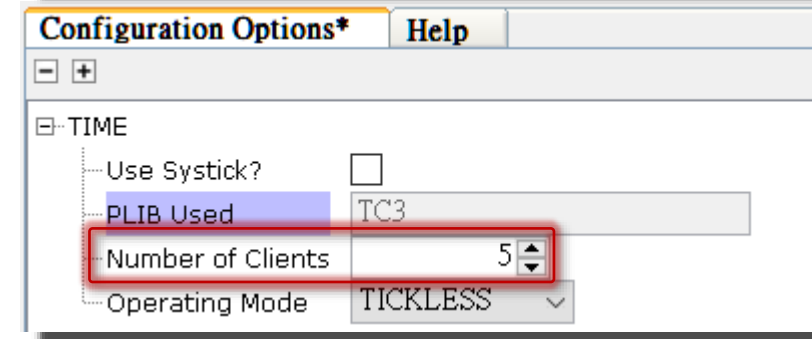
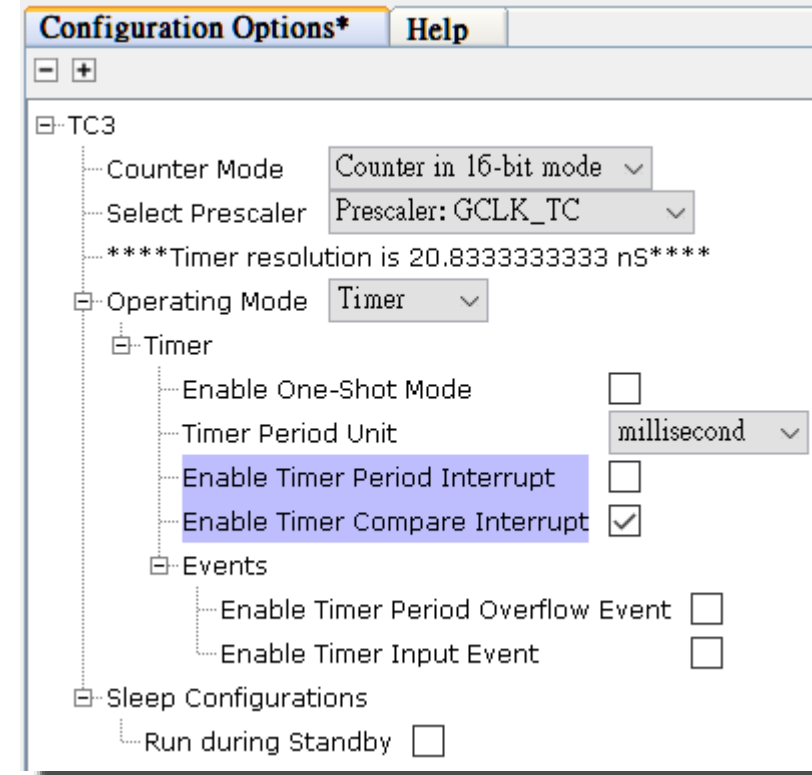


⚙️ Lab1 Delay based LED toggle

- Harmony Configuration

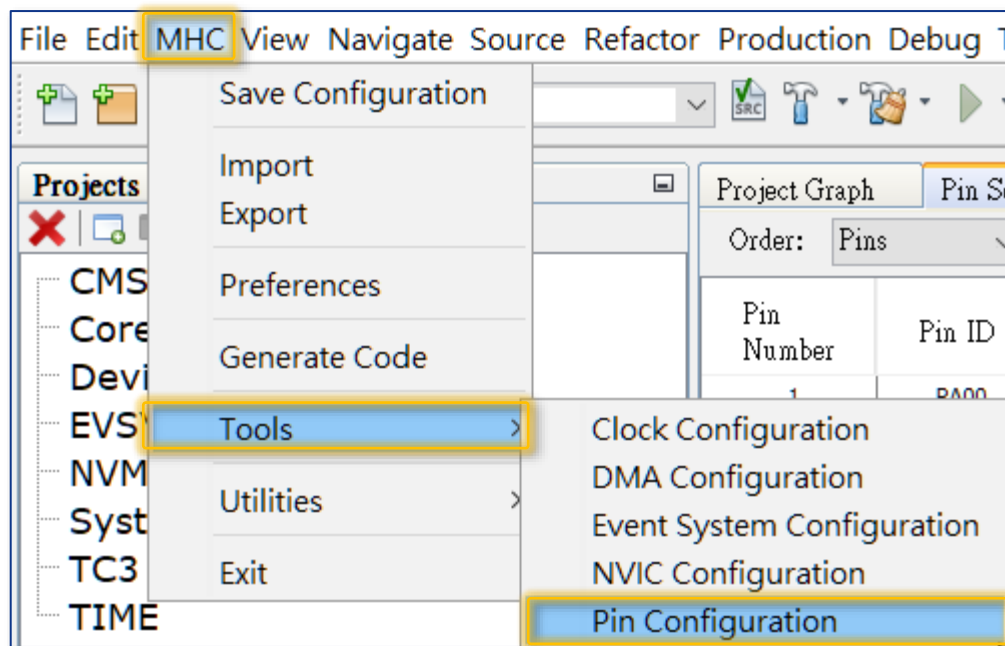


**Number of Clients to depend on your usage
Make sure it always enough for
System Service and your APP**



⚙️ Lab1 Delay based LED toggle

- Harmony Configuration (Setup GPIO)



**Configure PA20 as
GPIO output
Custom Name to LED1,
Latch is High.**

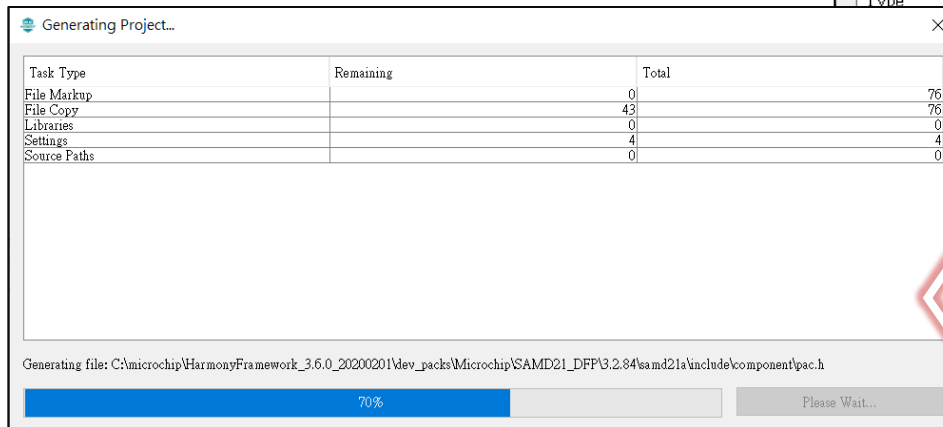
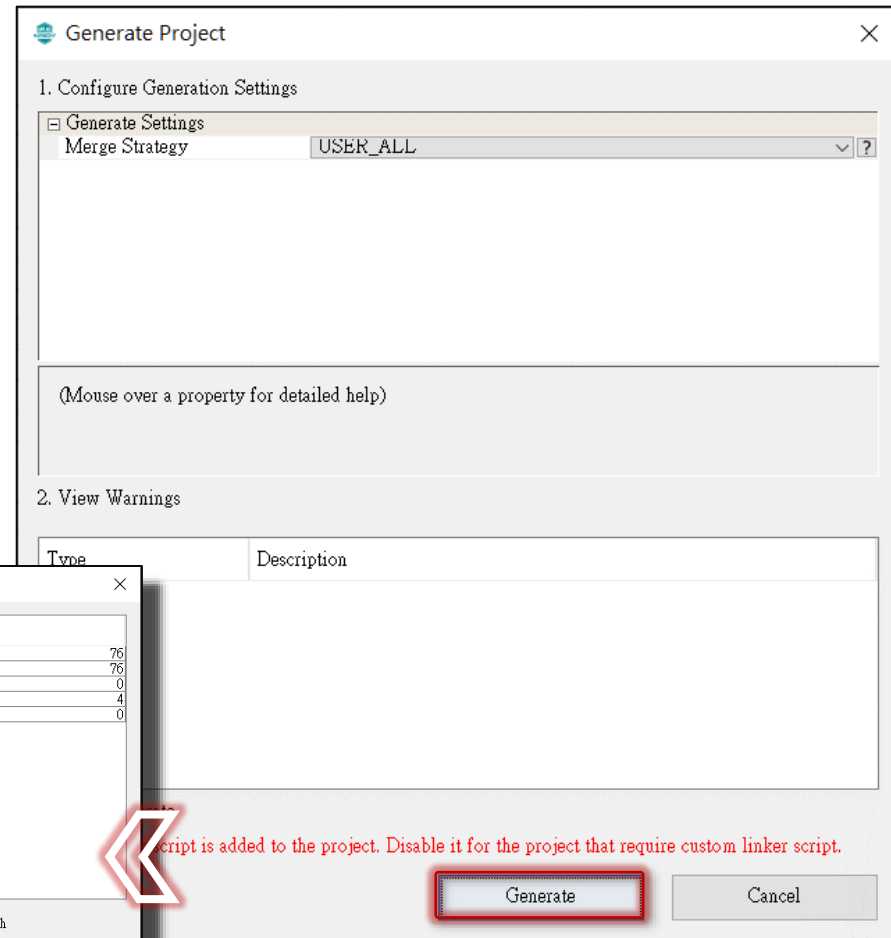
Project Graph* Pin Settings Pin Diagram Pin Table									
Order: Ports Table View ☒ Easy View									
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
28	PA19		Available	Digital	High Impedance	Low	☐	☐	NORMAL
29	PA20	LED1	GPIO	Digital	Out	High	☐	☐	NORMAL
30	PA21		Available	Digital	High Impedance	Low	☐	☐	NORMAL

⚙️ Lab1 Delay based LED toggle

- Harmony Configuration (GenCode)



**Not hint
for this step
anymore after Lab1**



script is added to the project. Disable it for the project that require custom linker script.

⚙️ Lab1 Delay based LED toggle

- Configure your new project

Double confirm the Device : ATSAM21G18A

Select correct Debugger CMSIS 5.4.0 DFP 3.4.116 XC32 (V2.50)

Project Properties - Lab1_SYS_TIME_Delay

Categories:

- General
- File Inclusion/Exclusion
- Conf: [default]**
- Simulator
- Loading
- Libraries
- Building
- XC32 (Global Options)
 - xc32-as
 - xc32-gcc
 - xc32-g++
 - xc32-ld
 - xc32-ar
 - Code Coverage

Configuration

Family: All Families

Device: **ATSAMD21G18A**

Connected Hardware Tool: **Atmel Embedded Debugger-SN: ATM...** ☐ Show All

Supported Debug Header: None

Packs:

- CMSIS 5.4.0
- SAMD21_DFP 3.4.116
- XC32 (V2.50)

Compiler Toolchain:

- XC32
 - XC32 (v3.00) [C:\Program Files\Microchip\xc32\v3.00\bin]**
 - ARM

The device on the kit does not match the device selected

Manage Configurations...

OK Cancel Apply Unlock Help

⚙️ Lab1 Delay based LED toggle

- Modify your main.c as below

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

SYS_TIME_HANDLE tmrHandle;

int main( void )
{
    SYS_Initialize(NULL);

    SYS_TIME_DelayMS(1000, &tmrHandle);

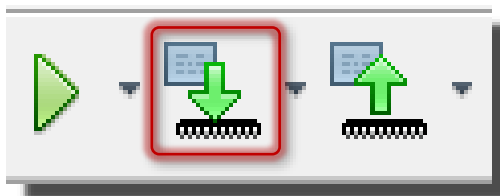
    while( true )
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true )
        {
            LED1_Toggle();
            SYS_TIME_DelayMS(1000, &tmrHandle);
        }

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

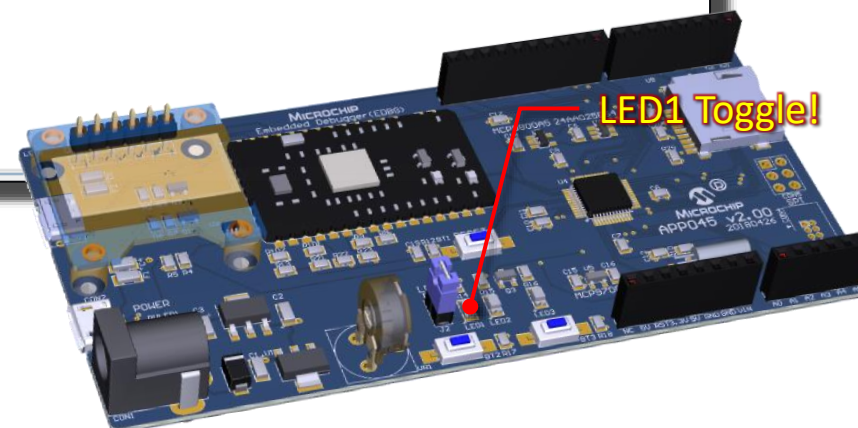
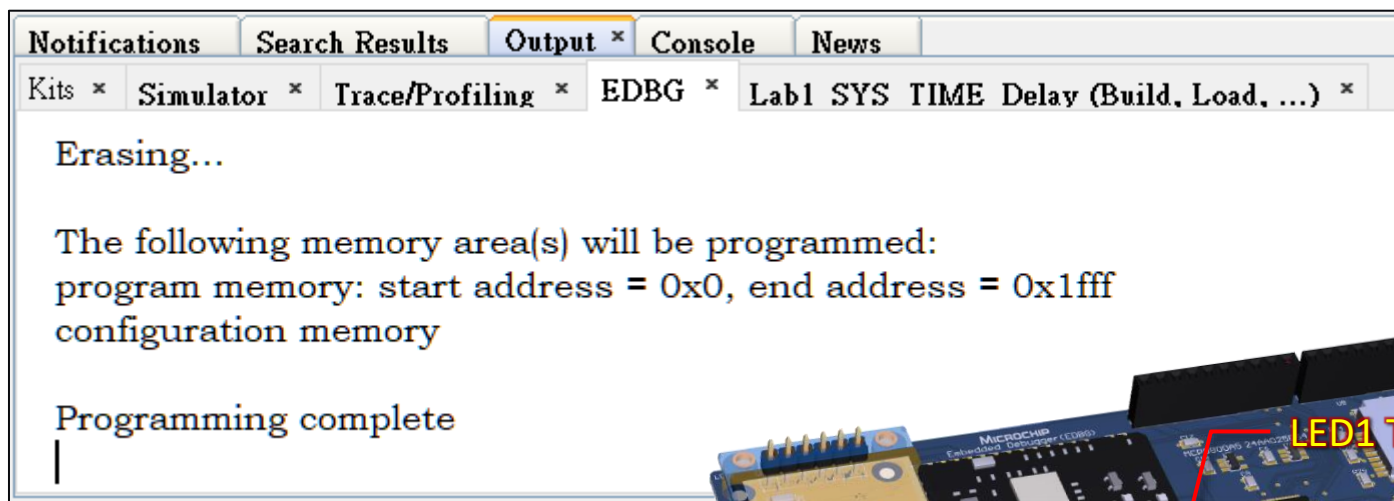
    /* Execution should not come here during normal operation */
    return ( EXIT_FAILURE);
}
```

⚙️ Lab1 Delay based LED toggle

- Compiler and programming your project to EVB



Not hint for this step anymore after Lab1



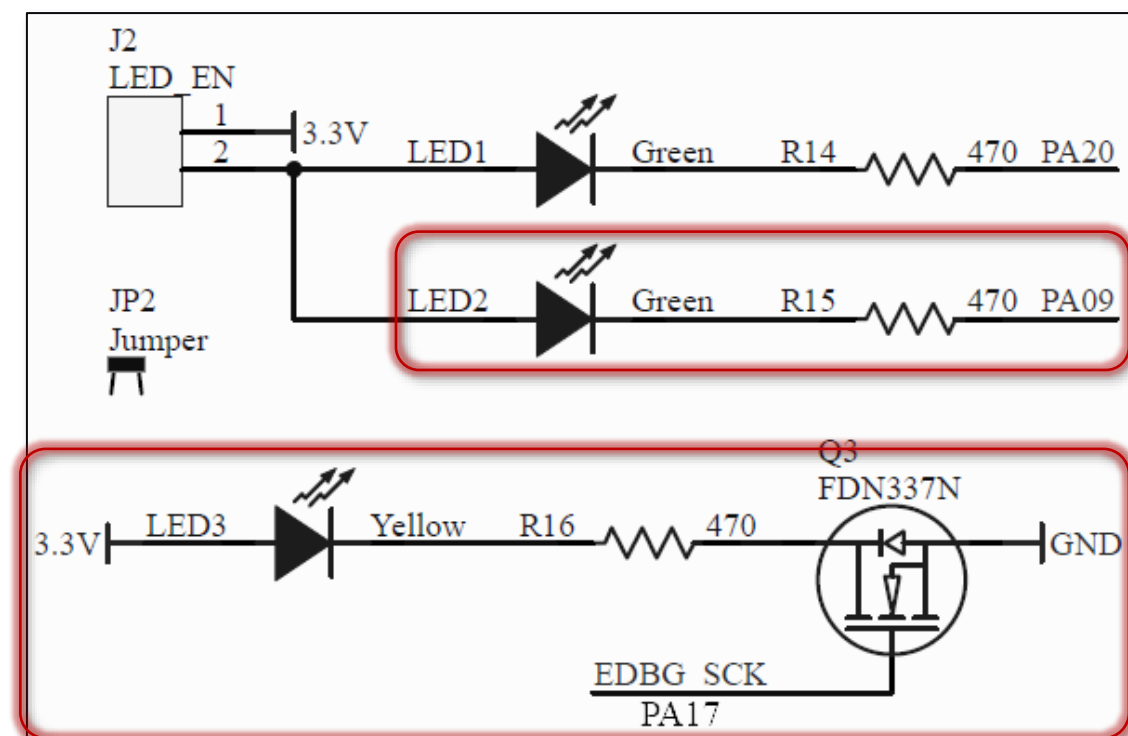
Lab 2

Callback based LED toggle

TIME System Service

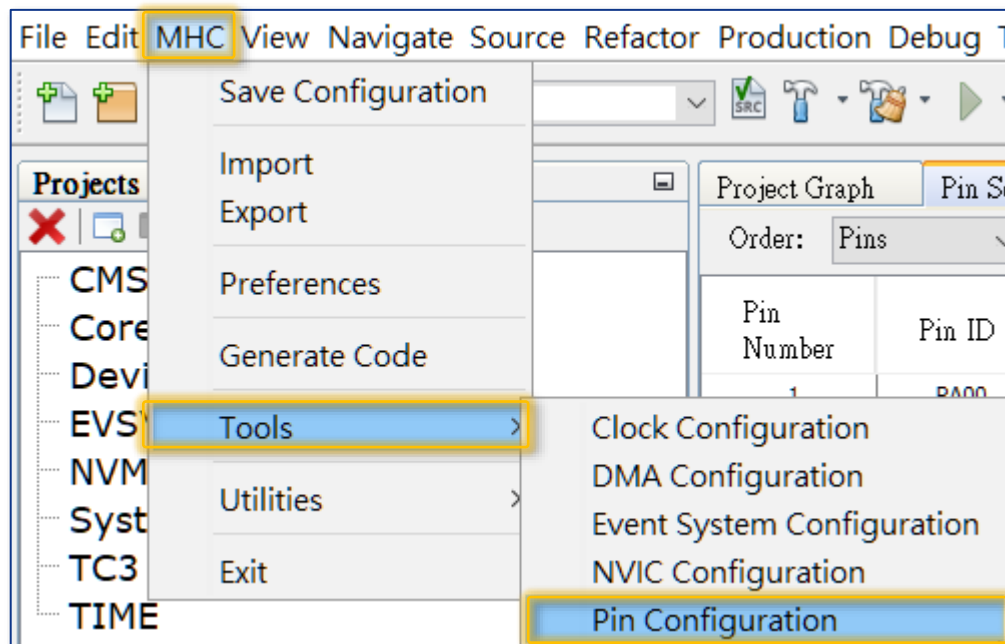
⚙️ Lab2 Callback based LED toggle

- Try use TIME system service to create two **Periodic Callback** for toggle **LED2** in **500ms** period and toggle **LED3** in **250ms**.
- The **LED2** is **low active** on pin **PA09**.
- The **LED3** is **high active** on pin **PA17**.



⚙️ Lab2 Callback based LED toggle

- Harmony Configuration (Setup GPIO)



Configure
PA09 as GPIO output
PA17 as GPIO output
PA09 Custom Name to LED2
PA17 Custom Name to LED3
PA09 Latch is High
PA17 Latch is Low

Project Graph* Pin Settings Pin Diagram Pin Table									
Order: Ports		Table View		☒ Easy View					
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
14	PA09	LED2	GPIO	Digital	Out	High	☐	☐	NORMAL
26	PA17	LED3	GPIO	Digital	Out	Low	☐	☐	NORMAL
29	PA20	LED1	GPIO	Digital	Out	High	☐	☐	NORMAL

⚙️ Lab2 Callback based LED toggle

- Modify your main.c as below

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

SYS_TIME_HANDLE tmrHandle;

void LED2Callback(uintptr_t context)
{
    LED2_Toggle();
}

void LED3Callback(uintptr_t context)
{
    LED3_Toggle();
}

int main( void )
{
    SYS_Initialize(NULL);

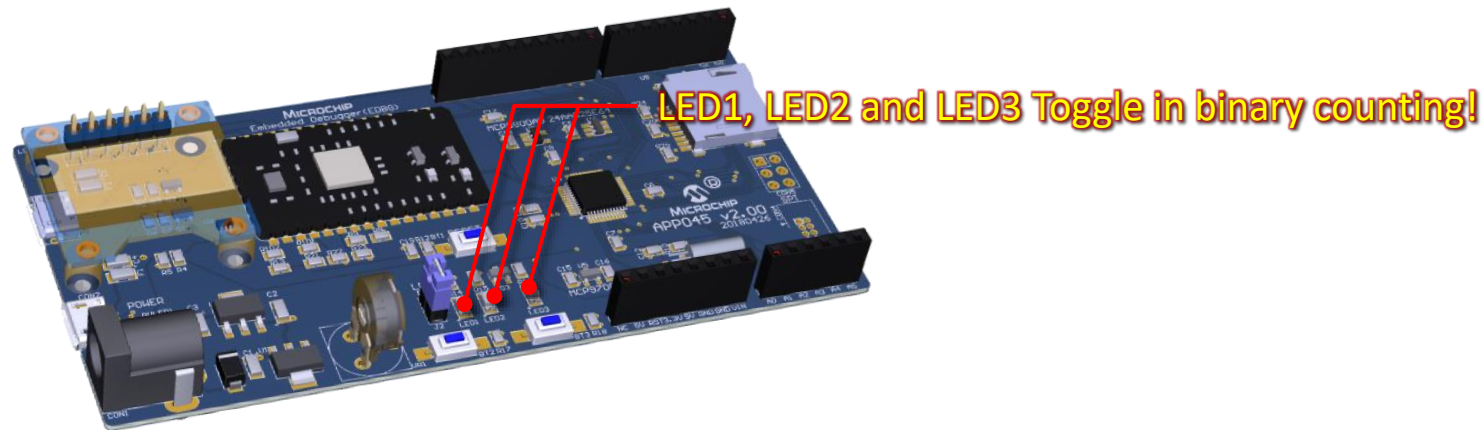
    SYS_TIME_DelayMS(1000, &tmrHandle);

    SYS_TIME_CallbackRegisterMS(LED2Callback, 0, 500, SYS_TIME_PERIODIC);
    SYS_TIME_CallbackRegisterMS(LED3Callback, 0, 250, SYS_TIME_PERIODIC);

    while( true )
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true )
        {
            LED1_Toggle();
        }
        ...
    }
}
```

⚙️ Lab2 Callback based LED toggle

- Result



LED1 (1000ms)	LED2 (500ms)	LED3 (250ms)
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

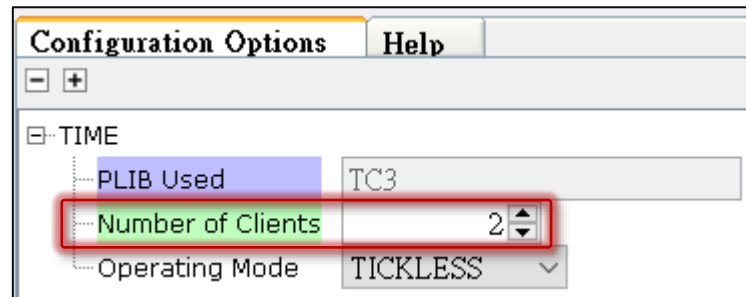
⚙️ Lab2 Callback based LED toggle

- **Discuss**

- What's the effect if change Callback Type to **SYS_TIME_SINGLE** ?

```
SYS_TIME_CallbackRegisterMS(LED2Callback, 0, 500, SYS_TIME_SINGLE);  
SYS_TIME_CallbackRegisterMS(LED3Callback, 0, 250, SYS_TIME_PERIODIC);
```

- What's happen if change **Number of Client** from 5 to 2 ?

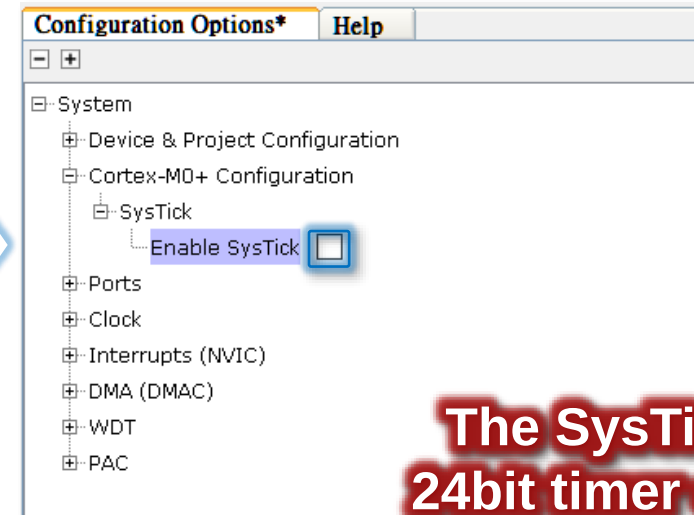
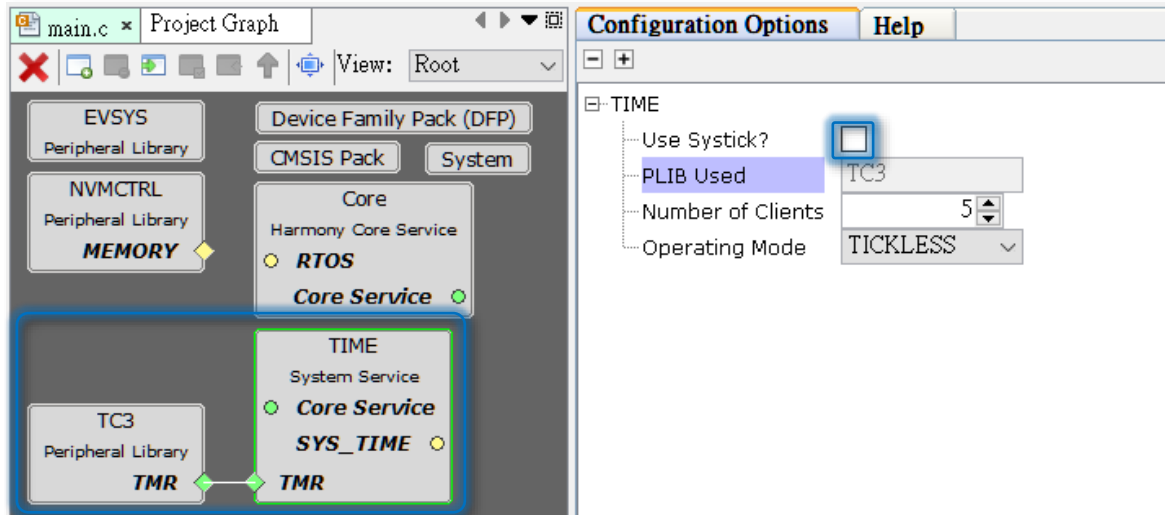


```
SYS_TIME_DelayMS(1000, &tmrHandle);  
SYS_TIME_CallbackRegisterMS(LED2Callback, 0, 500, SYS_TIME_PERIODIC);  
SYS_TIME_CallbackRegisterMS(LED3Callback, 0, 250, SYS_TIME_PERIODIC);
```

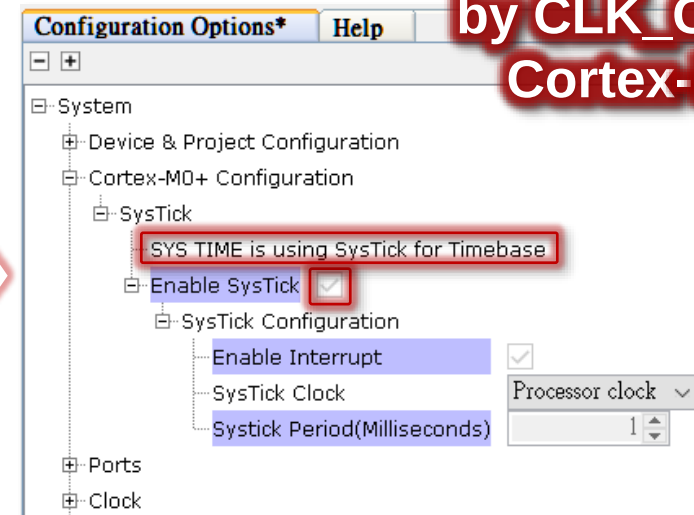
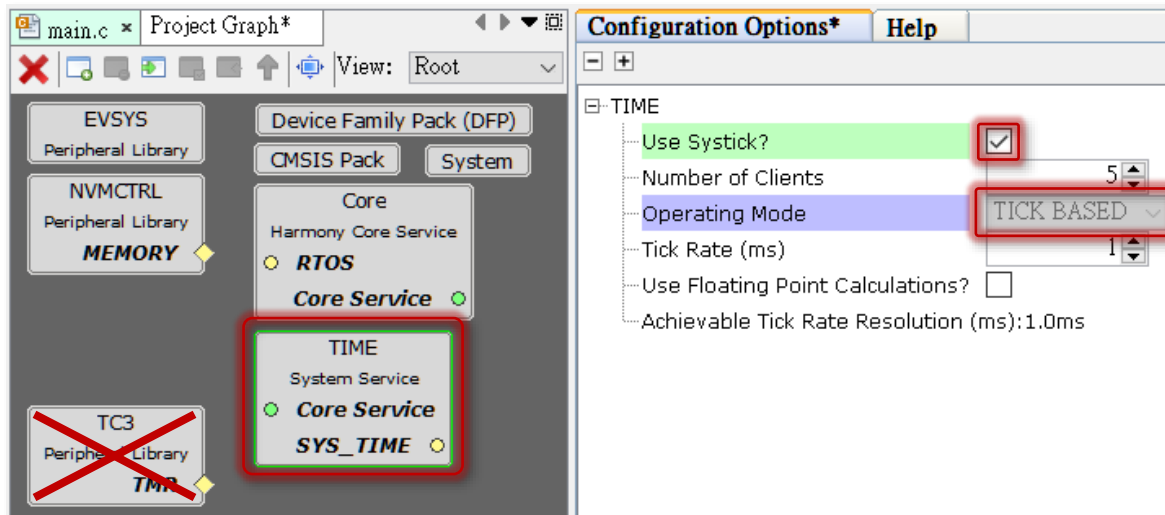
⚙️ Lab2 Callback based LED toggle

- Discuss

- How about to enable **Use Systick?** in TIME System Service ?



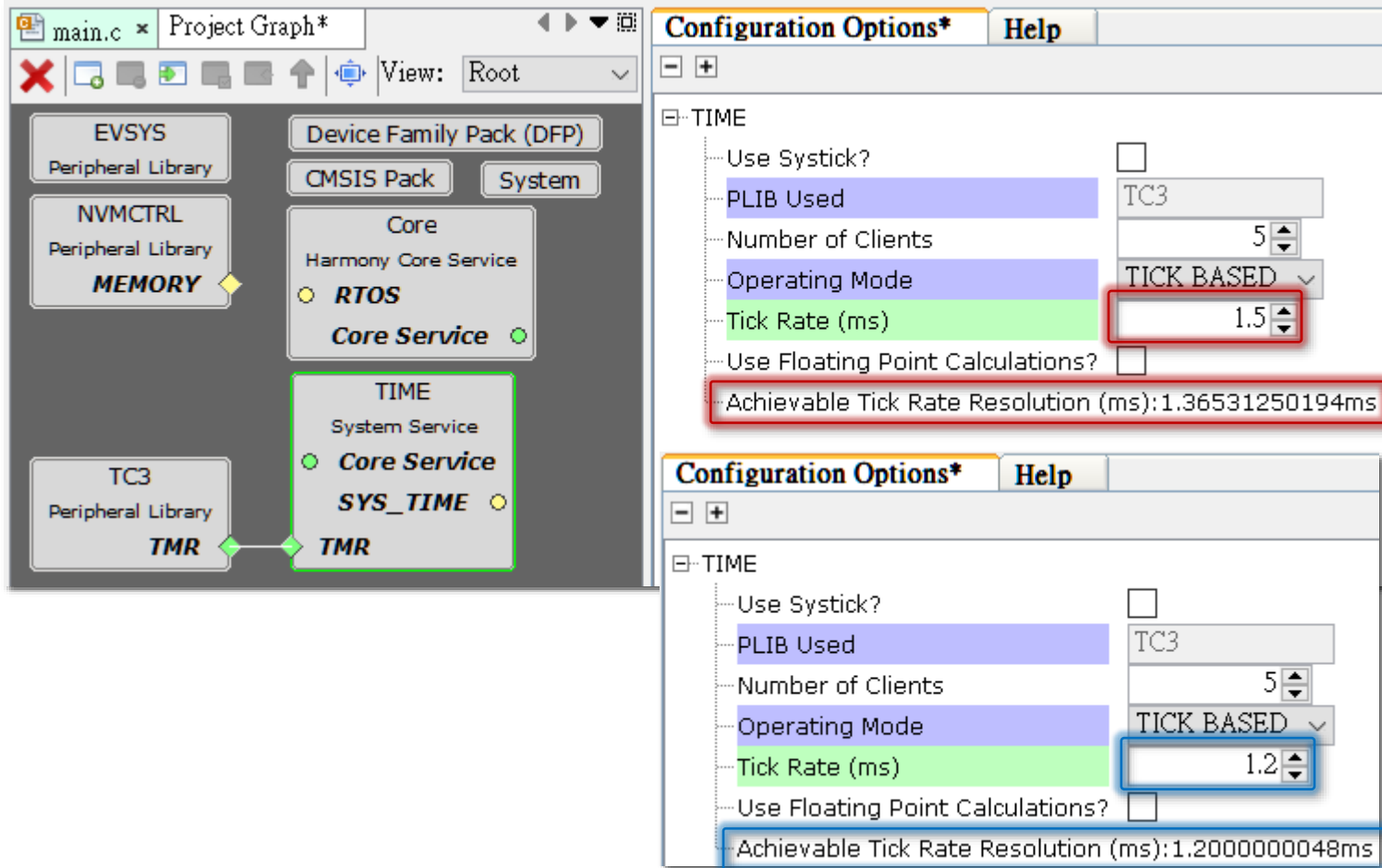
The SysTick is a 24bit timer clocked by CLK_CPU in Cortex-M0+



✖ Lab2 Callback based LED toggle

- Discuss

- What's the **Achievable Tick Rate Resolution (ms)** in TIME System Service means ?



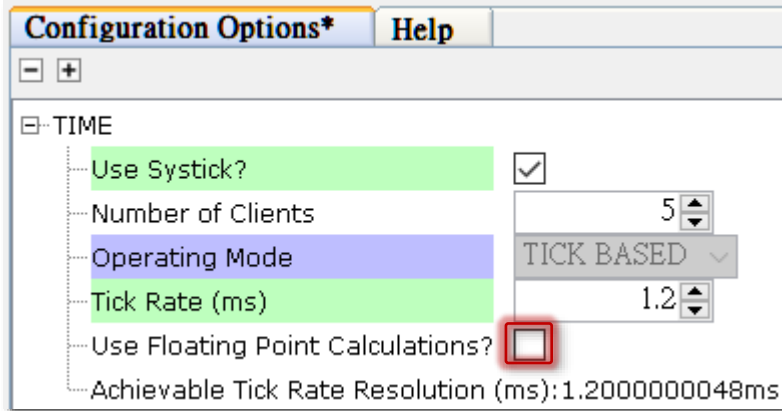
The maximum period of 16 bits TC3 @ 48MHz is $65535 \times (1 / 48 \text{ MHz}) = 1.36531250194 \text{ ms}$
Desired Tick Rate 1.5 ms is not reachable

Desired Tick Rate 1.2 ms is $1 / 0.0012 = 833.33333... \text{ Hz}$
Truncate the decimal, that's $1 / 833.33333 = 1.2000000048 \text{ ms}$

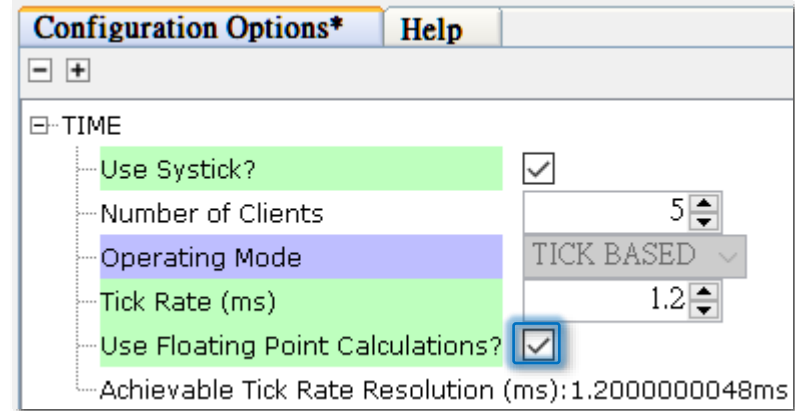
✖ Lab2 Callback based LED toggle

- Discuss

- How about to enable [Use Floating Point Calculations?](#) in TIME System Service ?



```
#define SYS_TIME_TICK_FREQ_IN_HZ (833.33333)
uint32_t hwTimerTickFreq;
counterObj->hwTimerTickFreq = (uint32_t)SYS_TIME_TICK_FREQ_IN_HZ;
uint32_t SYS_TIME_USToCount ( uint32_t us )
{
    uint32_t count = (((uint64_t)us * gSystemCounterObj.hwTimerTickFreq) / 1000000);
    return count;
}
uint32_t SYS_TIME_MSToCount ( uint32_t ms )
{
    uint32_t count = (((uint64_t)ms * gSystemCounterObj.hwTimerTickFreq) / 1000);
    return count;
}
```



```
#define SYS_TIME_TICK_FREQ_IN_HZ (833.33333)
float hwTimerTickFreq;
counterObj->hwTimerTickFreq = SYS_TIME_TICK_FREQ_IN_HZ;
uint32_t SYS_TIME_USToCount ( uint32_t us )
{
    float count = ((us * gSystemCounterObj.hwTimerTickFreq) / 1000000);
    return (uint32_t) ((uint32_t)count + ((count - (uint32_t)count) > 0));
}
uint32_t SYS_TIME_MSToCount ( uint32_t ms )
{
    float count = ((ms * gSystemCounterObj.hwTimerTickFreq) / 1000);
    return (uint32_t) ((uint32_t)count + ((count - (uint32_t)count) > 0));
}
```

CONSOLE System Service

Harmony System Service

CONSOLE System Service

- **Introduction**

The Console System Service routes data or message traffic between a console device and a middleware layer or application. The most common use of the Console Service is to route debug or error messages to a terminal program on a host development system.

The Console System Service consists of a core_layer and a console device_layer. The core_layer handles module initialization and system functions. It provides a common API for use by applications and middleware libraries to send and receive data to and from one or more console devices. The Console System core layer_maintains the status and device descriptor information for each console instance that has been initialized.

The console device_layer_contains the functional implementation of the core_layer_APIs. Each console device may have a different implementation, but the behavior of the API should be uniform across different devices. The console device_layer_interacts with the peripheral libraries or drivers or middle-wares to send and receive data to and from the hardware peripheral. The details of the data transaction are abstracted by the console device layer.

The console implementation is non-blocking in nature. For console write requests, the application data is copied into the ring buffer maintained by the console device layer and the call returns immediately. The return value of the API indicates the number of bytes actually copied. If the return value is less than the requested bytes, the application must retry after some time. The data transfer is controlled by a state machine implemented in the console device_layer.

Similarly, for console read requests, the received data is copied to the application buffer from the internal receive ring buffer maintained by the console device layer. The return value indicates the number of bytes actually copied to the application buffer. Application must try reading again after some time, if the number of read bytes copied (indicated by the return value) is less than the bytes requested.

The size of the transmit and receive ring buffers is configurable in MHC.

CONSOLE System Service

- Harmony Configuration

The image shows the Microchip Harmony IDE configuration interface for the CONSOLE System Service. It includes three configuration windows and a central component diagram.

Configuration Options* (Left):

- CONSOLE
 - Number of Instances Using UART: 2
 - Number of Instances Using USB: 1
 - Console Print Buffer Size (128-8192): 200

Configuration Options* (Right):

- CONSOLE
 - Device Used: SERCOM5
 - TX/RX Ring Buffers must be configured in the attached PLIB

SERCOM5 Configuration (Bottom Right):

- Select SERCOM Operation Mode: USART with internal Clock
- Operating Mode: Ring buffer mode
- Configure Ring Buffer Size:
 - TX Ring Buffer Size: 128
 - RX Ring Buffer Size: 128

Component Diagram (Center):

- CONSOLE System Service** (Core Service)
 - Instances: 0, 1, 2
 - Instance 0: UART, USB_DEVICE_CDC, SYS_CONSOLE
 - Instance 1: UART, USB_DEVICE_CDC, SYS_CONSOLE
 - Instance 2: UART, USB_DEVICE_CDC, SYS_CONSOLE
- USB Full Speed Driver** (Core Service)
 - DRV_USB
- USB Device Layer** (DRV_USB)
 - USB_DEVICE
- CDC Function Driver** (USB_DEVICE)
 - Instances: 0
 - Instance 0: USB_DEVICE_CDC
- SERCOM5 Peripheral Library** (UART)
- SERCOM2 Peripheral Library** (UART)

Consumers (Bottom Right):

- sys_console (SYS_CONSOLE)
- Multi-Connect Capability
- COMMAND (sys_command)
- DEBUG (sys_debug)
- TCPIP CORE (tcpipStack)

USB Device CDC Instance Supported from MHC 3.5.0

CONSOLE System Service

API functions of the Console System Service library.

- **System Functions**

SYS_CONSOLE_Initialize

Initializes the console instance module and opens or initializes the specific module instance to which it is associated.

SYS_CONSOLE_Status

Returns status of the specific instance of the Console module.

SYS_CONSOLE_Tasks

Maintains the console's state machine.

- **Core Functions**

SYS_CONSOLE_DeviceGet

Returns the device type for a given console instance

SYS_CONSOLE_HandleGet

Returns a handle to the requested console instance

SYS_CONSOLE_Read

Reads data from the console device.

SYS_CONSOLE_ReadCountGet

Returns number of unread bytes available in the receive buffer.

SYS_CONSOLE_ReadFreeBufferCountGet

Returns the amount of free space in bytes available in the receive buffer.

SYS_CONSOLE_Write

Writes data to the console device.

SYS_CONSOLE_WriteCountGet

Returns the number of bytes pending for transmission in the buffer.

SYS_CONSOLE_WriteFreeBufferCountGet

Returns the amount of free space in bytes in the transmit buffer.

SYS_CONSOLE_Flush

Flushes the read and write queues for the given console instance.

SYS_CONSOLE_Message

Prints a message to the console

SYS_CONSOLE_Print

Formats and prints a message with a variable number of arguments to the console

CONSOLE System Service

API functions of the Console System Service library. (cont.)

- Data Types and Constants

STDERR_FILENO

This is macro STDERR_FILENO.

STDIN_FILENO

These are in unistd.h

STDOUT_FILENO

This is macro STDOUT_FILENO.

SYS_CONSOLE_DEVICE

Lists the available console devices.

SYS_CONSOLE_HANDLE

Handle to an instance of the console system service.

SYS_CONSOLE_MESSAGE

Print message on the default console instance

SYS_CONSOLE_PRINT

Print formatted message on the default console instance

SYS_CONSOLE_DEFAULT_INSTANCE

This is macro SYS_CONSOLE_DEFAULT_INSTANCE.

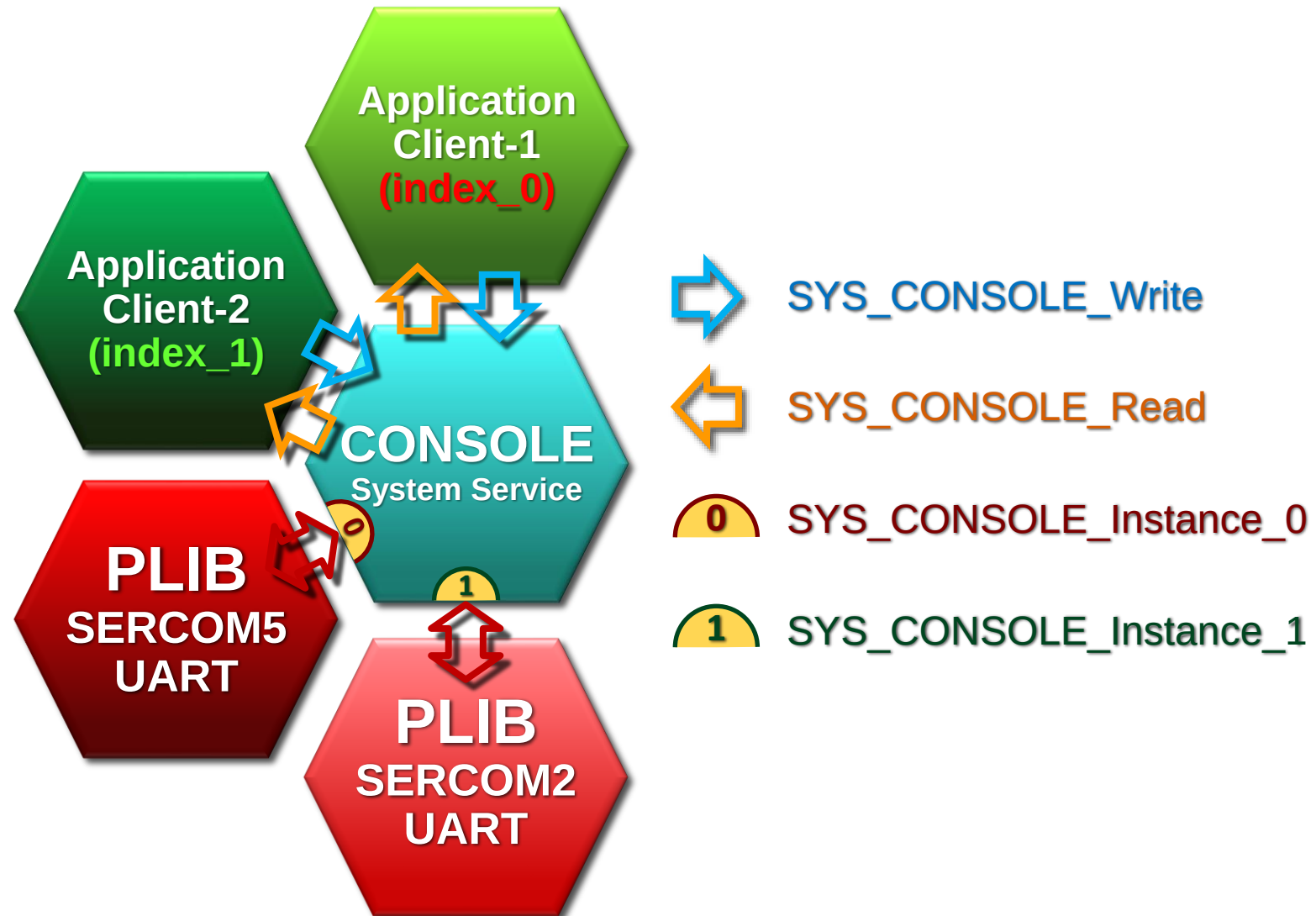
SYS_CONSOLE_HANDLE_INVALID

Invalid console handle

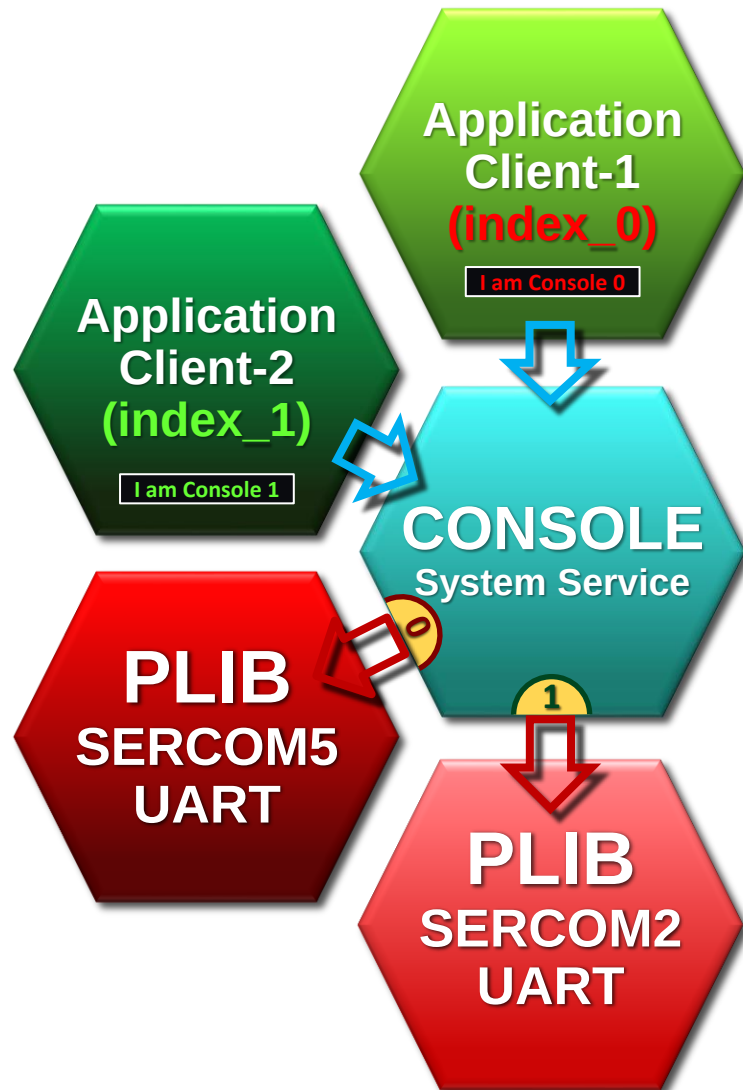
SYS_CONSOLE_STATUS

This is type SYS_CONSOLE_STATUS.

CONSOLE System Service



CONSOLE System Service



```
void App1_Task()  
{  
    SYS_CONSOLE_HANDLE consoleHandle0;  
    consoleHandle0 = SYS_CONSOLE_HandleGet(  
        SYS_CONSOLE_INDEX_0 );  
    SYS_CONSOLE_Write( consoleHandle0,  
        "I am Console 0\r\n", 16 );  
}  
  
void App2_Task()  
{  
    SYS_CONSOLE_HANDLE consoleHandle1;  
    consoleHandle1 = SYS_CONSOLE_HandleGet(  
        SYS_CONSOLE_INDEX_1 );  
    SYS_CONSOLE_Write( consoleHandle1,  
        "I am Console 1\r\n", 16 );  
}
```

VT COM3 - Tera Term VT
File Edit Setup Control Win
I am Console 0

VT COM23 - Tera Te...
File Edit Setup Control Win
I am Console 1

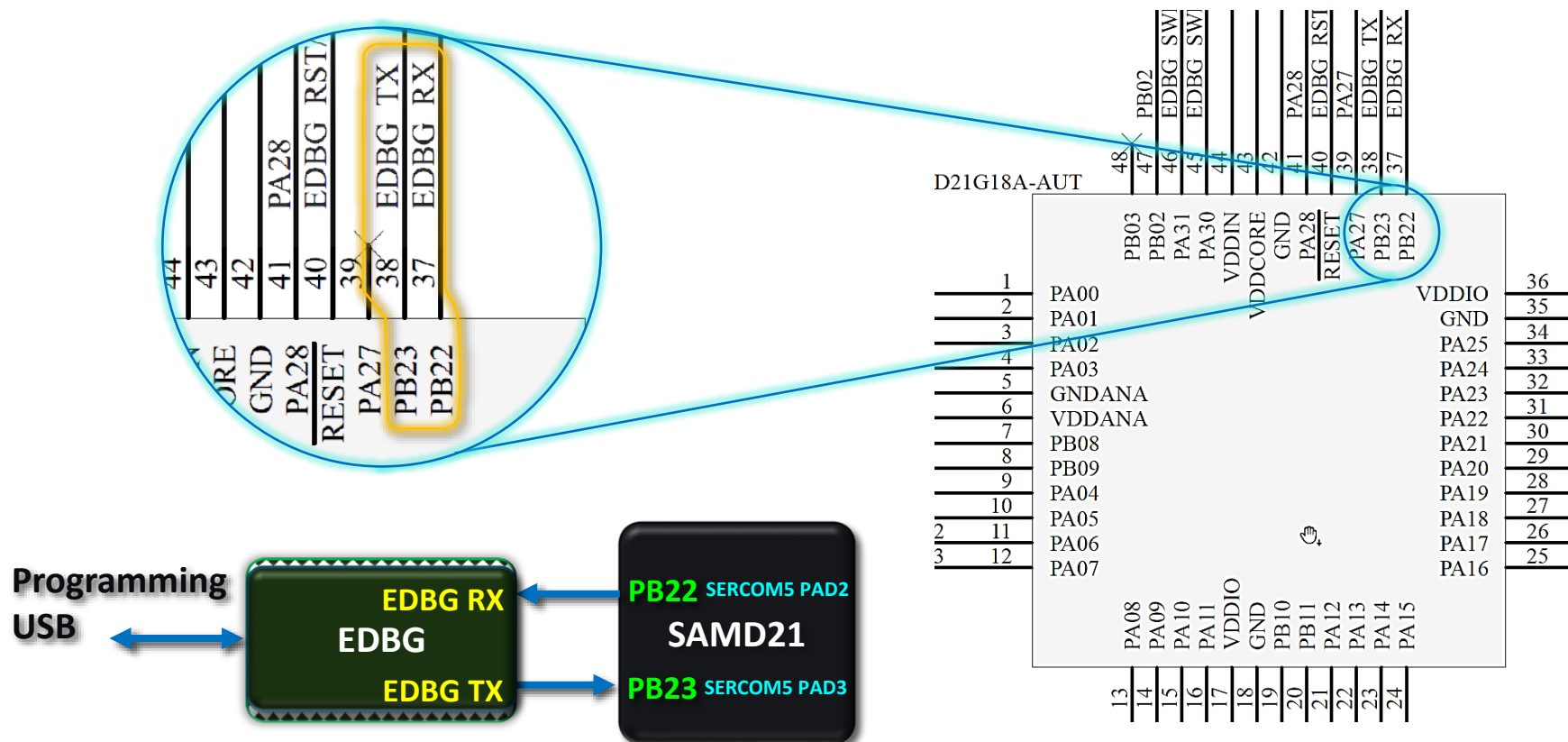
Lab 3

UART Console

CONSOLE System Service

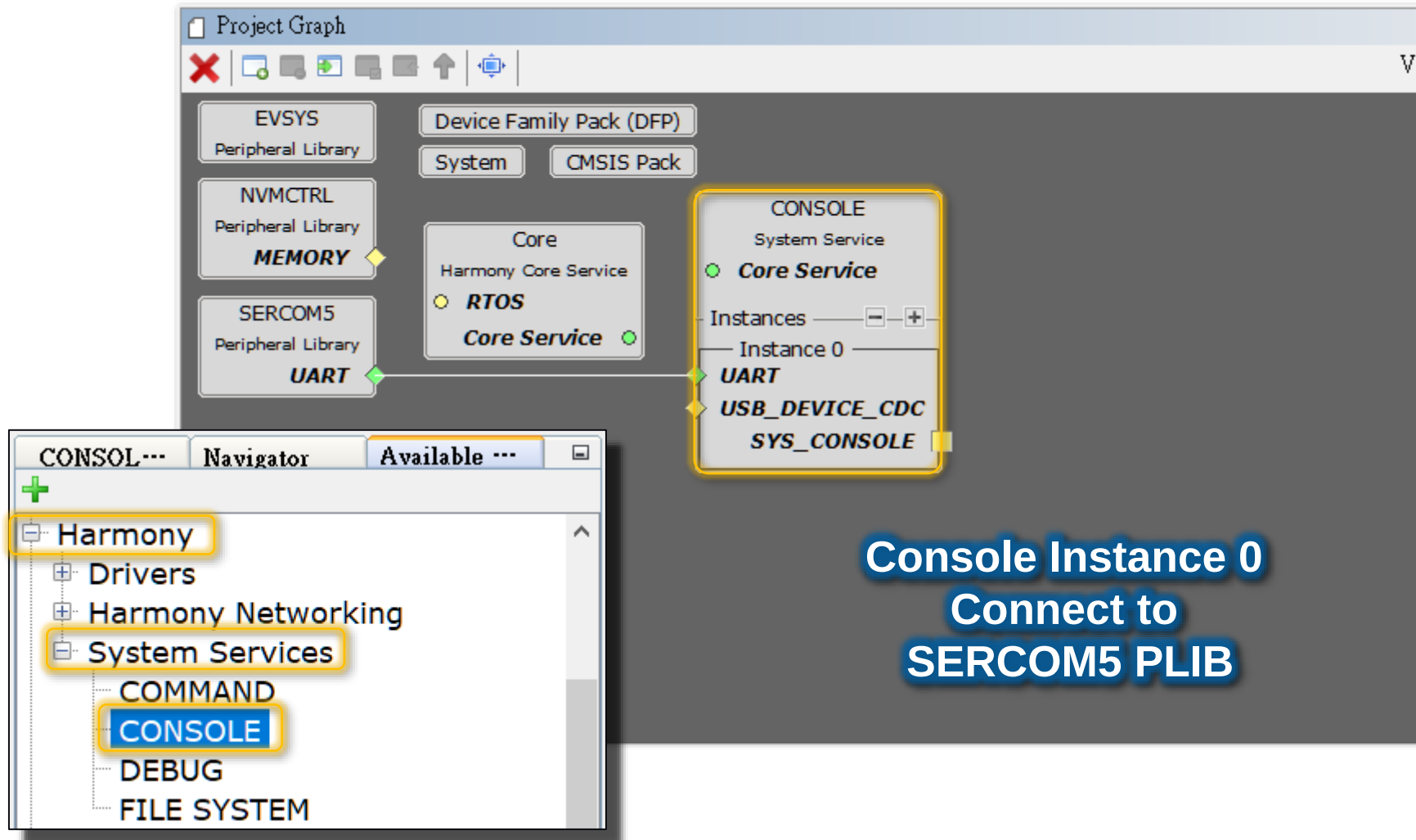
⚙️ Lab3 UART Console

- Try add **CONSOLE** system service to create a **UART console**.
- Select **SERCOM5 PLIB** as **Instance 0** of UART Console.
- Output **“Hello World!”** to UART console.
- Receive console input **X** and print it out as **“Receive : X”**



⚙️ Lab3 UART Console

- Harmony Configuration (Add CONSOLE system service)



⚙️ Lab3 UART Console

- Harmony Configuration (CONSOLE and PLIB SERCOM5 config)

The screenshot displays the Microchip Harmony IDE interface. The central 'Project Graph' shows the system architecture with components like EVSYS, NVMCTRL, SERCOM5, and the Core Service. The SERCOM5 component is highlighted with a green box, and its configuration options are shown in a bottom-left panel. The CONSOLE component is also highlighted with a green box, and its configuration options are shown in a top-right panel. A bottom-right panel shows the 'Device Used' as SERCOM5. A large orange text overlay states: 'Print buffer size should depend to proper size of SERCOM Ring Buffer size'. A red text overlay at the bottom provides pinout information: 'Receive Pinout : SERCOM PAD[3]' and 'Transmit Pinout: SERCOM PAD[2]'. The SERCOM5 configuration panel includes settings for 'Select SERCOM Operation Mode' (USART with internal Clock), 'Operating Mode' (Ring buffer mode), 'Configure Ring Buffer Size' (TX and RX Ring Buffer Size set to 128), 'Receive Enable', 'Transmit Enable', 'Frame Format' (USART frame), 'Baud Rate in Hz' (115,200), 'Parity Mode' (No Parity), 'Character Size' (8 Bits), 'Stop Bit Mode' (One Stop Bit), 'Receive Pinout' (SERCOM PAD[3] is used for data reception), 'Transmit Pinout' (PAD[2] = TxD; PAD[3] = XCK), and 'Enable Run in Standby' (unchecked). The CONSOLE configuration panel shows 'Number of Instances Using UART' set to 1, 'Number of Instances Using USB' set to 0, and 'Console Print Buffer Size (128-8192)' set to 200. The 'Device Used' panel shows 'Device Used' set to SERCOM5 and a note: 'TX/RX Ring Buffers must be configured in the attached PLIB'.

Print buffer size should depend to proper size of SERCOM Ring Buffer size

**Receive Pinout : SERCOM PAD[3]
Transmit Pinout: SERCOM PAD[2]**

✖ Lab3 UART Console

- Harmony Configuration (Setup PLIB SERCOM5 pads)

Project Graph* Pin Diagram * Pin Table * Pin Settings *									
Order:	Ports	Table View	☒ Easy View						
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
37	PB22		SERCOM5_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
38	PB23		SERCOM5_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL

Configure PB22 as SERCOM5_PAD2
Configure PB23 as SERCOM5_PAD3

Project Graph* Pin Diagram * Pin Table * Pin Settings *

Package: TQFP48

		2	23	24	25	26	27	28	29	30	31	32	33	34	39	41	45	46	47	48	7	8	19	20	37	38
		A																		B						
Module	Function	3	14	15	16	17	18	19	20	21	22	23	24	25	27	28	30	31	2	3	8	9	10	11	22	23
SERCOM5	SERCOM5_PAD0																									
	SERCOM5_PAD1																									
	SERCOM5_PAD2																									
	SERCOM5_PAD3																									

⚙️ Lab3 UART Console

- Modify your main.c as below

```
#include "definitions.h"                // SYS function prototypes
#include <stdio.h>
#include <string.h>

SYS_TIME_HANDLE tmrHandle;
SYS_CONSOLE_HANDLE consoleHandlePLIB;
char UartWriteBuf[30];
char UartReadBuf[30];

void LED2Callback(uintptr_t context) {...}
void LED3Callback(uintptr_t context) {...}

int main( void )
{
    SYS_Initialize(NULL);

    consoleHandlePLIB = SYS_CONSOLE_HandleGet( SYS_CONSOLE_INDEX_0 );
    sprintf(UartWriteBuf, "Hello world !(PLIB)\r\n");
    SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));

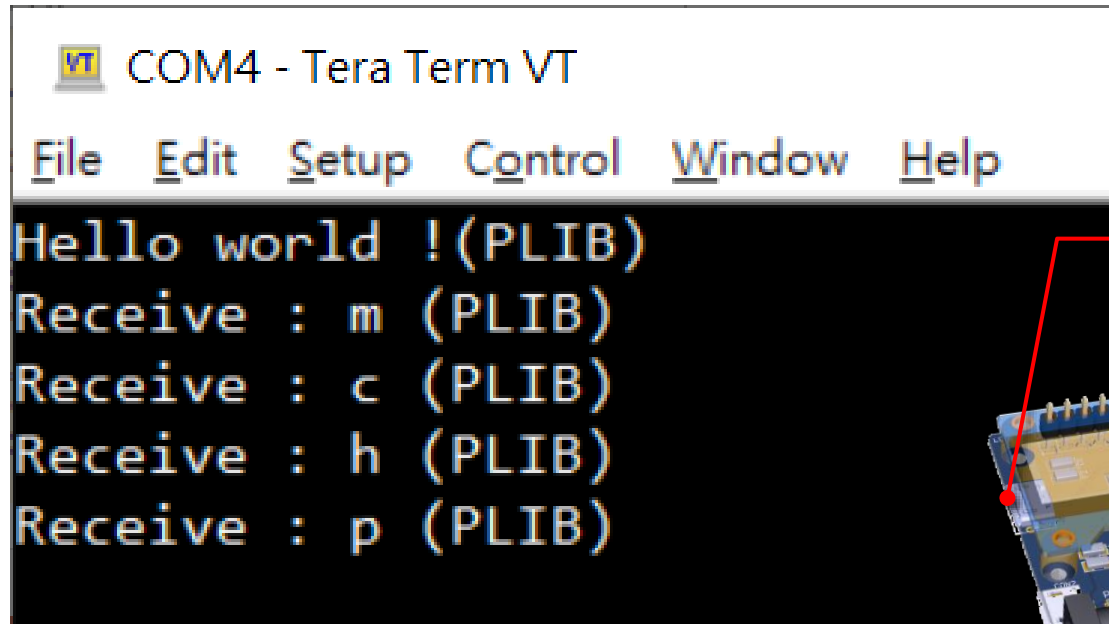
    SYS_TIME_DelayMS(1000, &tmrHandle);

    while ( true )
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true ) {...}

        if( SYS_CONSOLE_Read(consoleHandlePLIB, UartReadBuf, 1) > 0 )
        {
            sprintf(UartWriteBuf, "Receive : %s (PLIB)\r\n", UartReadBuf);
            SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));
        }
    }
}
```

⚙️ Lab3 UART Console

- Compiler and programming your project to EVB.
- Open COM Port terminal console 115200-n-8-1.
- Check the UART output in console.
- Type any character in console and check the response.



VT COM4 - Tera Term VT

File Edit Setup Control Window Help

Hello world !(PLIB)

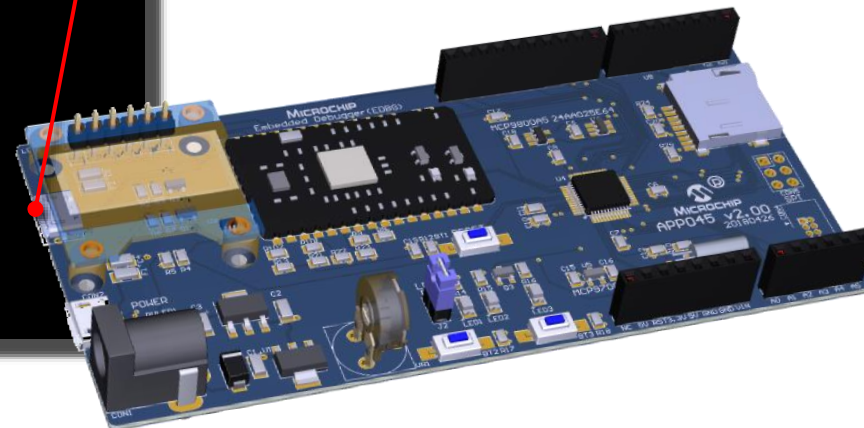
Receive : m (PLIB)

Receive : c (PLIB)

Receive : h (PLIB)

Receive : p (PLIB)

USB CDC UART console



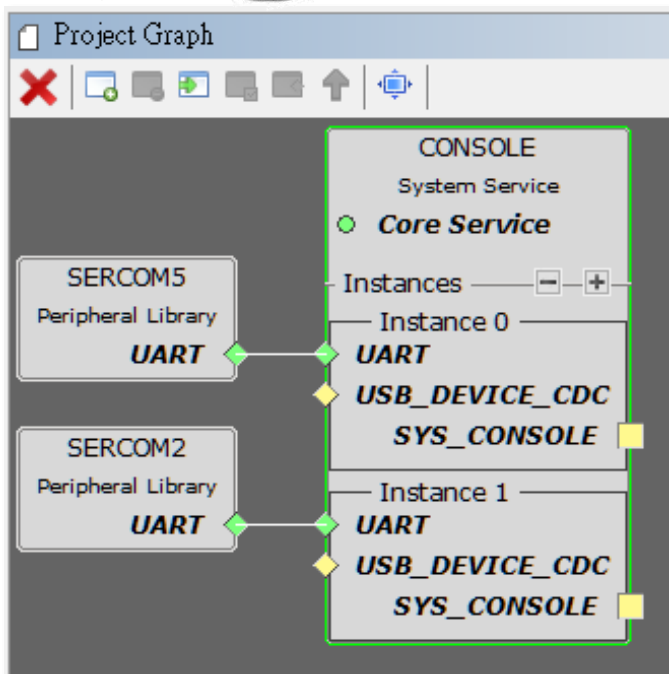
⚙️ Lab3 UART Console

• Discuss

- Try to implement multi-Instance consoles as lecture .
You can a UART TTL to USB CDC cable for second instance of CONSOLE.



UART TTL too
USB CDC cable



Pin Settings			
Order:	Ports	Table View	☒ Easy View
Pin Number	Pin ID	Custom Name	Function
15	PA10	SERCOM2_PAD2	SERCOM2_PAD2
16	PA11	SERCOM2_PAD3	SERCOM2_PAD3

Arduino UNO Socket		
PA11	7	D0/RX
PA10	8	D1/TX
		D2
		D3/PWM
		D4
		D5/PWM
		D6/PWM
		D7

VT

COM4 - Tera Term VT

File Edit Setup Control

Hello world !(PLIB0)

Receive : m (PLIB1)

Receive : c (PLIB1)

Receive : h (PLIB1)

Receive : p (PLIB1)

SERCOM2

Select SERCOM Operation Mode

Enable Interrupts ? ☒

Receive Enable ☒

Transmit Enable ☒

Enable Ring Buffer ? ☒

Enable Run in Standby ☐

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

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

VT

COM5 - Tera Term VT

File Edit Setup Control

Hello world !(PLIB1)

Receive : 1 (PLIB0)

Receive : 2 (PLIB0)

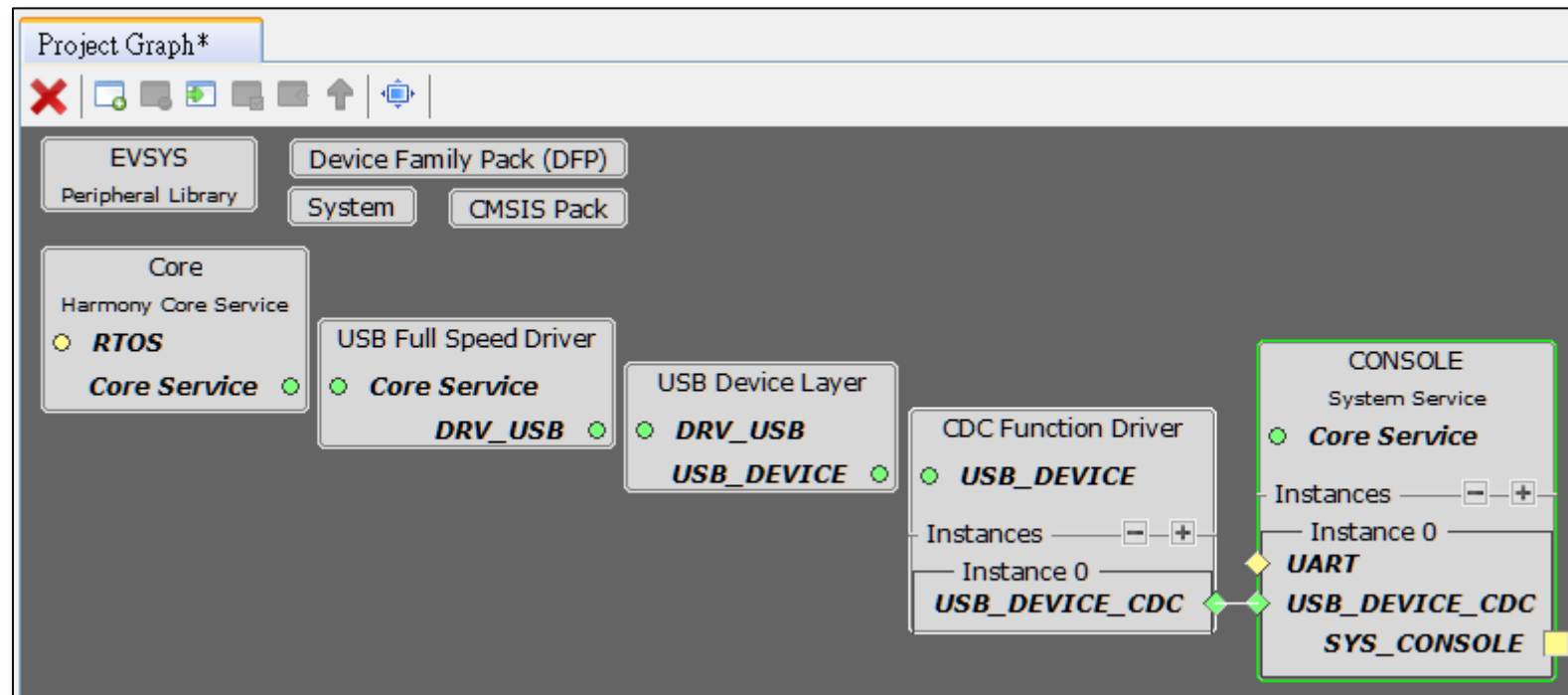
Receive : 3 (PLIB0)

Receive : 4 (PLIB0)

⚙️ Lab3 UART Console

- Discuss

- The **CONSOLE System Service** could support the **USB Device CDC Library** to develop another **USB COM Port** through the **native USB interface** of SAMD21, you could practice the **Lab6** firstly after you finish **Lab3**.
- To go to [Lab6 – USB Device CDC with CONSOLE System Service](#)



DEBUG System Service

Harmony System Service

DEBUG System Service

- **Introduction**

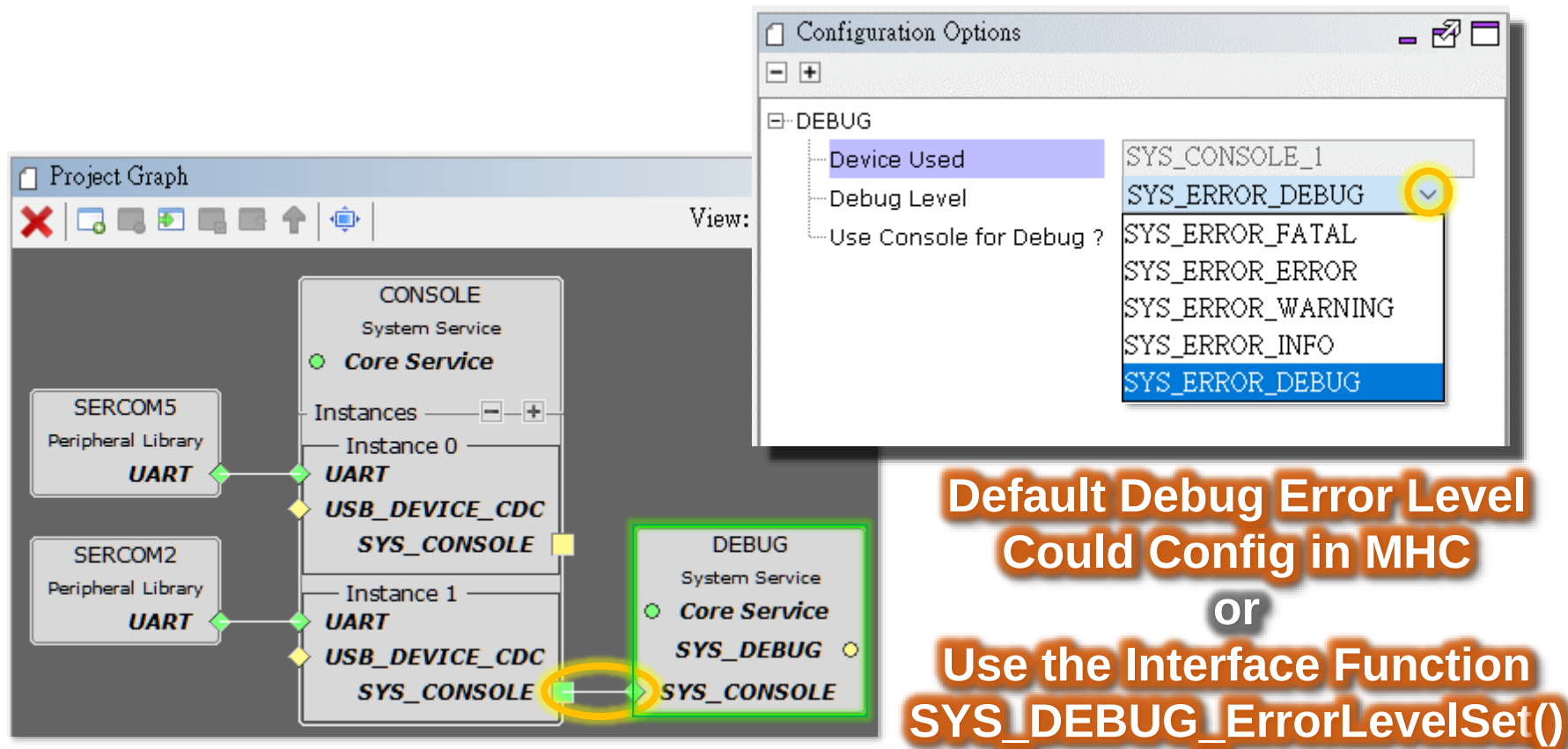
The Debug System Service provides a convenient mechanism for the application developer to send formatted or unformatted messages to a system console.

The Debug System Service provides a set of functions that allow the developer to output debug and/or error messages based on a global error level.

The messages are sent to the System Console Service, where they are routed to a console device.

The Debug System Service maintains a global error level, which may be set during initialization and changed dynamically at run-time. Both formatted and unformatted messages can be output or not, depending on the current global error level.

DEBUG System Service



**DEBUG System Service
must depend to
CONSOLE system Service**

DEBUG System Service

API functions of the Debug System Service library.

- **System Functions**

`SYS_DEBUG_Initialize`

Initializes the global error level and specific module instance.

`SYS_DEBUG_Status`

Returns status of the specific instance of the debug service module.

`SYS_DEBUG_BreakPoint`

Inserts a software breakpoint instruction when building in Debug mode.

- **Changing System Error Level Functions**

`SYS_DEBUG_ErrorLevelGet`

Returns the global system Error reporting level.

`SYS_DEBUG_ErrorLevelSet`

Sets the global system error reporting level.

- **Data Types and Constants**

`SYS_ERROR_LEVEL`

System error message priority levels.

`SYS_DEBUG_INDEX_x`

This is macro `SYS_DEBUG_INDEX_x`.

`SYS_DEBUG_MESSAGE`

Macro of `_SYS_DEBUG_MESSAGE`.

`SYS_DEBUG_PRINT`

Macro of `_SYS_DEBUG_PRINT`.

`_SYS_DEBUG_MESSAGE`

Prints a debug message if the specified level is $\leq$ global system error level.

`_SYS_DEBUG_PRINT`

Format prints a debug message if the specified level is $\leq$ global system error level.

- **Other Functions**

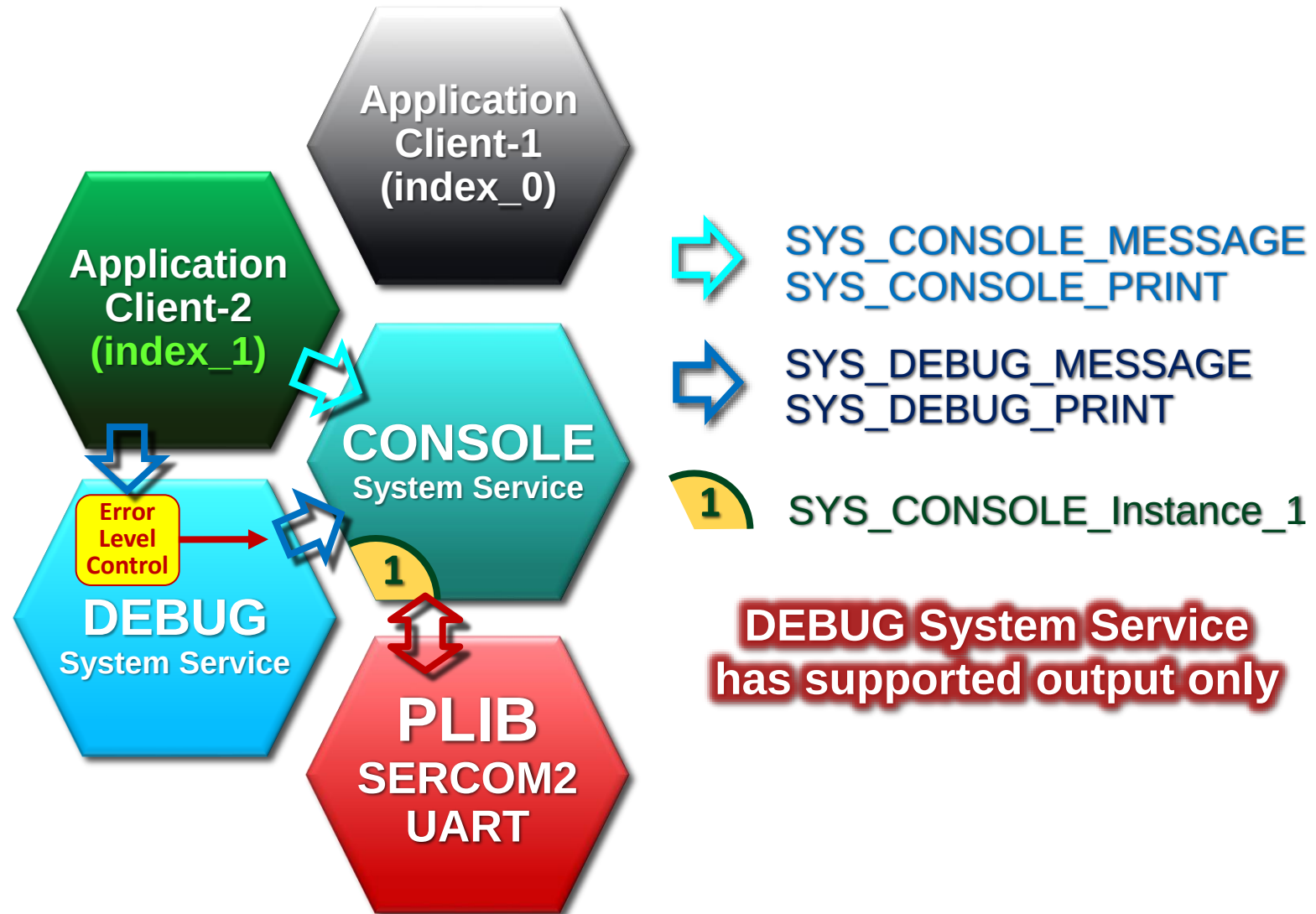
`SYS_DEBUG_Redirect`

Allows re-direction of debug system service to another console instance

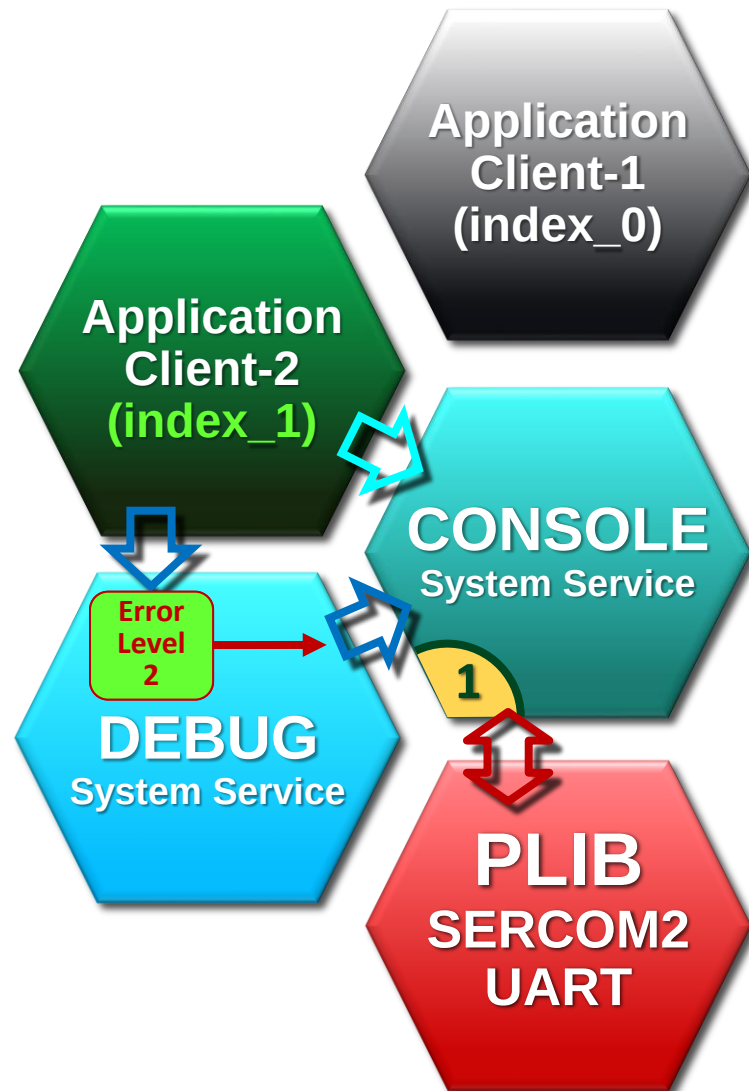
`SYS_DEBUG_ConsoleInstanceGet`

This is function `SYS_DEBUG_ConsoleInstanceGet`.

DEBUG System Service



DEBUG System Service



```
void App2_Task()...
{
    int EL = SYS_DEBUG_ErrorLevelGet();

    SYS_CONSOLE_MESSAGE("Message to Console 1\r\n");
    SYS_CONSOLE_PRINT("Current Error Level: %d\r\n", EL );
    SYS_DEBUG_MESSAGE(SYS_ERROR_DEBUG, "dErrlv 4\r\n");
    SYS_DEBUG_PRINT(SYS_ERROR_WARNING, "pErrlv %d\r\n", EL);
    SYS_CONSOLE_MESSAGE("Set Error Level to WARNING\r\n");
    SYS_DEBUG_ErrorLevelSet(SYS_ERROR_WARNING);

    SYS_DEBUG_MESSAGE(SYS_ERROR_DEBUG, "dErrlv 4\r\n");
    EL = SYS_DEBUG_ErrorLevelGet();
    SYS_DEBUG_PRINT(SYS_ERROR_WARNING, "pErrlv %d\r\n", EL);
}
```

DEBUG

Device Used	SYS_CONSOLE_1
Debug Level	SYS_ERROR_DEBUG
Debug Print Buffer Size (128-8192)	200
Use Console for Debug ?	☒

COM24 - Tera Term VT

```
File Edit Setup Control Window
Message to Console 1
Current Error Level: 4
dErrlv 4
pErrlv 4
Set Error Level to WARNING
pErrlv 2
```

Name	Lv
FATAL	0
ERROR	1
WARNING	2
INFO	3
DEBUG	4

Lab 4

Debug Message Control

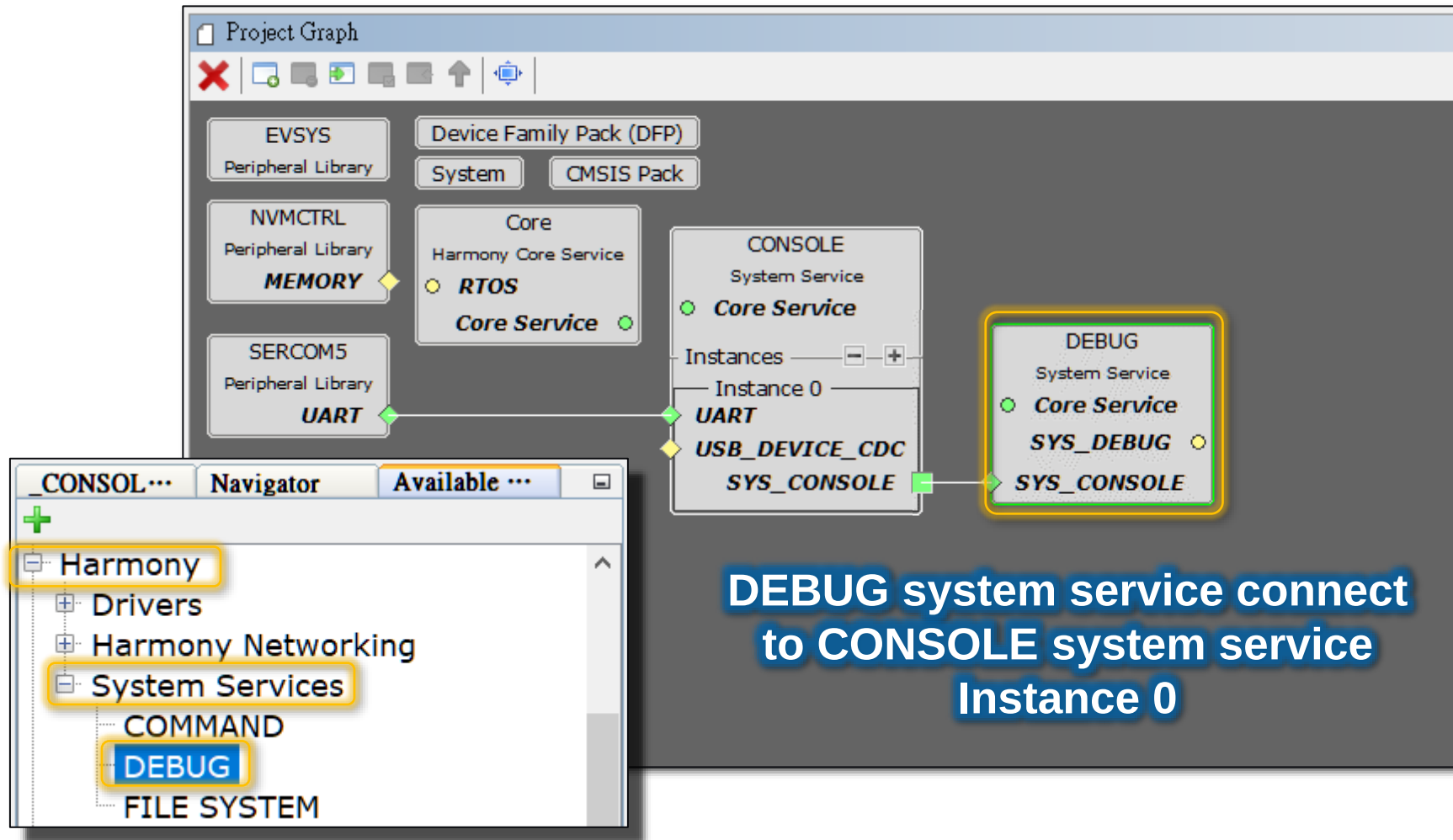
DEBUG System Service

✂ Lab4 Debug Message Control

- Try add **DEBUG system service** and output message through UART to use previous added **CONSOLE system service**.
- Use **CONSOLE system service** Macro to output “**Hello world !**” and “**Receive : X**” message.
- Input number ‘0’, ‘1’, ‘2’, ‘3’, ‘4’ in UART console and use **DEBUG system service** to show error-level controlled message “**Change Error Level to #**” (the # is input number) and then change the correspond **Error Level** could control five different Error Level debug message output as below.
 - “**Error Level 0 - Fatal**”
 - “**Error Level 1 - Error**”
 - “**Error Level 2 - Warning**”
 - “**Error Level 3 - Info**”
 - “**Error Level 4 - Debug**”

⚙️ Lab4 Debug Message Control

- Harmony Configuration (Add DEBUG system service)



⚙️ Lab4 Debug Message Control

- Harmony Configuration (DEBUG system service config)

The screenshot displays the Microchip Harmony IDE interface. The top window, 'Project Graph', shows a hierarchical view of the project components. On the left, a 'Peripheral Library' contains 'EVSYS', 'NVMCTRL' (with 'MEMORY' selected), and 'SERCOM5' (with 'UART' selected). The 'Core' section shows 'Harmony Core Service' with 'RTOS' and 'Core Service' (indicated by a green dot). The 'CONSOLE' section shows 'System Service' with 'Core Service' (green dot) and 'Instance 0' containing 'UART', 'USB_DEVICE_CDC', and 'SYS_CONSOLE'. A 'DEBUG' section is highlighted with a green border, showing 'System Service' with 'Core Service' (green dot), 'SYS_DEBUG' (yellow dot), and 'SYS_CONSOLE'. The bottom window, 'Configuration Options', shows the 'DEBUG' section expanded. It includes 'Device Used' (set to 'SYS_CONSOLE_0'), 'Debug Level' (set to 'SYS_ERROR_DEBUG' via a dropdown menu), and 'Use Console for Debug?' (checked). The dropdown menu for 'Debug Level' lists the following options: SYS_ERROR_FATAL, SYS_ERROR_ERROR, SYS_ERROR_WARNING, SYS_ERROR_INFO, and SYS_ERROR_DEBUG (which is highlighted in blue).

Use Console for Debug is natural

Default Error Level could Config in MHC

⚙️ Lab4 Debug Message Control

- Modify your main.c as below

Comment out

```
int main( void )
{
    ...
    // consoleHandlePLIB = SYS_CONSOLE_HandleGet( SYS_CONSOLE_INDEX_0 );
    // sprintf(UartWriteBuf, "Hello world !(PLIB)\r\n");
    // SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));
    SYS_CONSOLE_MESSAGE("Hello world !(PLIB)\r\n");
    ...
    while ( true )
    {
        ...
        if( SYS_CONSOLE_Read(consoleHandlePLIB, UartReadBuf, 1) > 0 )
        {
            // sprintf(UartWriteBuf, "Receive : %s (PLIB)\r\n", UartReadBuf);
            // SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));
            SYS_CONSOLE_PRINT("\r\nReceive : %s (PLIB)\r\n", UartReadBuf);

            if(UartReadBuf[0]>='0' && UartReadBuf[0]<='4')
            {
                int ErrorLevel = UartReadBuf[0]-'0';

                SYS_DEBUG_ErrorLevelSet(ErrorLevel);
                SYS_DEBUG_PRINT(SYS_ERROR_FATAL, "Change Error Level to %d\r\n",
                                SYS_DEBUG_ErrorLevelGet());
            }
            SYS_DEBUG_MESSAGE(SYS_ERROR_FATAL, "Error Level 0 - Fatal\r\n");
            SYS_DEBUG_MESSAGE(SYS_ERROR_ERROR, "Error Level 1 - Error\r\n");
            SYS_DEBUG_MESSAGE(SYS_ERROR_WARNING, "Error Level 2 - Warning\r\n");
            SYS_DEBUG_MESSAGE(SYS_ERROR_INFO, "Error Level 3 - Info\r\n");
            SYS_DEBUG_MESSAGE(SYS_ERROR_DEBUG, "Error Level 4 - Debug\r\n");
        }
    }
}
```

Comment out

⚙️ Lab4 Debug Message Control

- Compiler and programming your project to EVB.
- Type '0', '1', '2', '3', '4', 'a', 'b' in console to check the difference.

```
COM4 - Tera Ter...  
File Edit Setup Control Window Help  
Hello world !(PLIB)  
Receive : 0 (PLIB)  
Change Error Level to 0  
Error Level 0 - Fatal  
Receive : 1 (PLIB)  
Change Error Level to 1  
Error Level 0 - Fatal  
Error Level 1 - Error  
Receive : 2 (PLIB)  
Change Error Level to 2  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning
```

```
COM4 - Tera Ter...  
File Edit Setup Control Window Help  
Receive : 3 (PLIB)  
Change Error Level to 3  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3   
Receive : 4 (PLIB)  
Change Error Level to 4  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3 
```

```
COM4 - Tera Ter...  
File Edit Setup Control Window Help  
Receive : a (PLIB)  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3 - Info  
Error Level 4 -   
Receive : b (PLIB)  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3 - Info  
Error Level 4 - 
```

✂ Lab4 Debug Message Control

- Why DEBUG print cannot finish output ?

```
COM4 - Tera Ter...  
File Edit Setup Control Window Help  
Receive : 3 (PLIB)  
Change Error Level to 3  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3  
Receive : 4 (PLIB)
```

SERCOM5

Select SERCOM Operation Mode: USART with internal C

Enable Interrupts ? ☒

Receive Enable ☒

Transmit Enable ☒

Enable Ring Buffer ? ☒

TX Ring Buffer Size: 128

RX Ring Buffer Size: 128

Frame Format: USART frame

Enable Run in Standby ☐

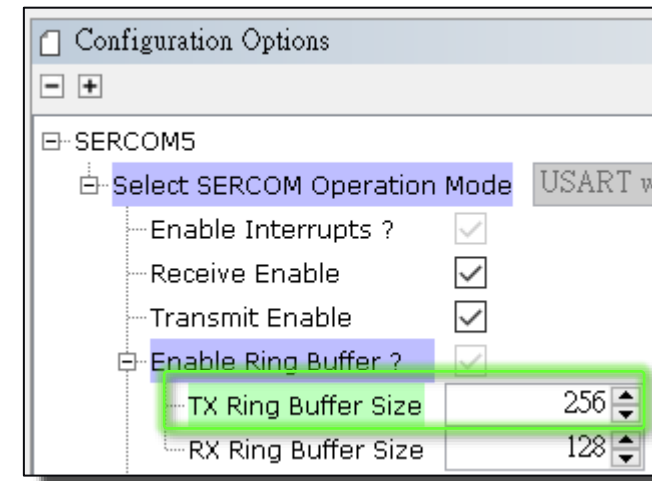
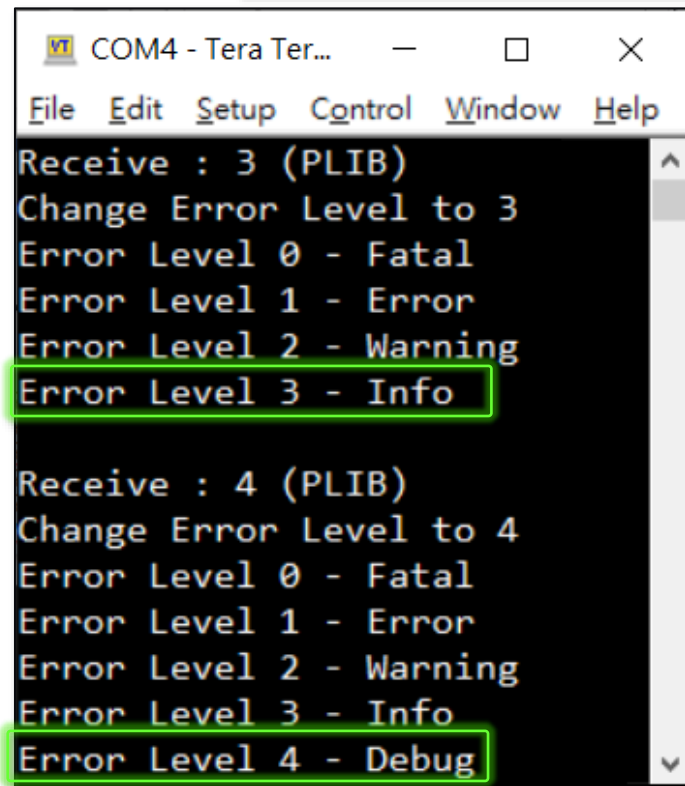
TX Ring Buffer Size is 128 Bytes

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
00	52	65	63	65	69	76	65	20	3A	20	33	20	28	50	4C	49	Receive : 3 (PLI
10	42	29	0D	0A	43	68	61	6E	67	65	20	45	72	72	6F	72	B)..Change Error
20	20	4C	65	76	65	6C	20	74	6F	20	33	0D	0A	45	72	72	Level to 3..Err
30	6F	72	20	4C	65	76	65	6C	20	30	20	2D	20	46	61	74	or Level 0 - Fat
40	61	6C	0D	0A	45	72	72	6F	72	20	4C	65	76	65	6C	20	al..Error Level
50	31	20	2D	20	45	72	72	6F	72	0D	0A	45	72	72	6F	72	1 - Error..Error
60	20	4C	65	76	65	6C	20	32	20	2D	20	57	61	72	6E	69	Level 2 - Warni
70	6E	67	0D	0A	45	72	72	6F	72	20	4C	65	76	65	6C	20	ng..Error Level
80	33																

⚙️ Lab4 Debug Message Control

- Please increase **TX Ring Buffer Size** to **256** or add a wait write finish condition after mass message to output.

```
...  
SYS_DEBUG_MESSAGE(SYS_ERROR_FATAL, "Error Level 0 - Fatal\r\n");  
SYS_DEBUG_MESSAGE(SYS_ERROR_ERROR, "Error Level 1 - Error\r\n");  
SYS_DEBUG_MESSAGE(SYS_ERROR_WARNING, "Error Level 2 - Warning\r\n");  
SYS_DEBUG_MESSAGE(SYS_ERROR_INFO, "Error Level 3 - Info\r\n");  
SYS_DEBUG_MESSAGE(SYS_ERROR_DEBUG, "Error Level 4 - Debug\r\n");  
while( SYS_CONSOLE_WriteCountGet(SYS_CONSOLE_DEFAULT_INSTANCE) );
```



⚙️ Lab4 Debug Message Control

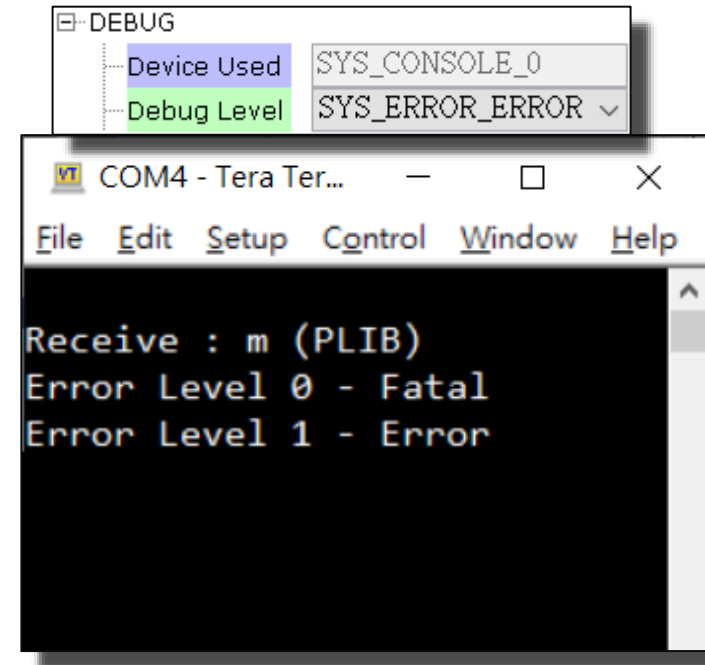
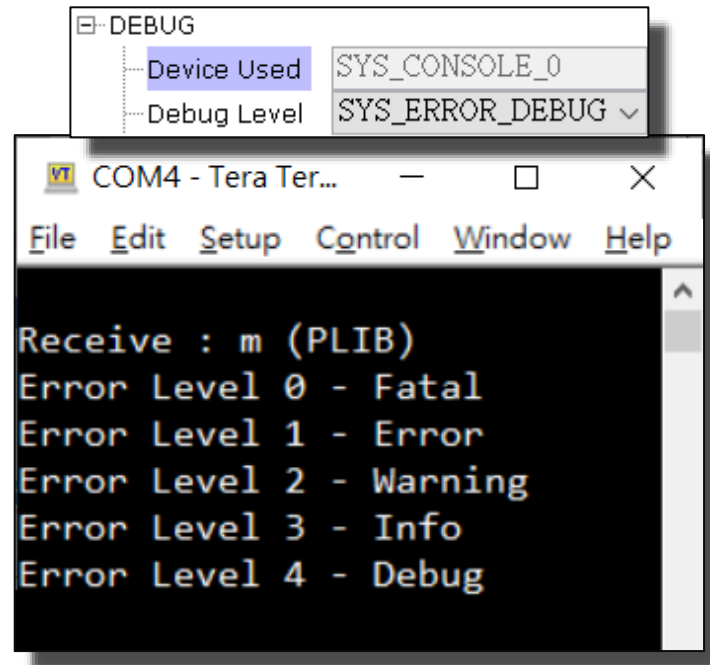
- Discuss

- What's the effect if insert a `SYS_DEBUG_BreakPoint()` while debugging ?

```
if(UartReadBuf[0]>='0' && UartReadBuf[0]<='4')
{
    int ErrorLevel = UartReadBuf[0]-'0';
    SYS_DEBUG_BreakPoint();
    SYS_DEBUG_ErrorLevelSet(ErrorLevel);
}
```

```
56 if(UartReadBuf[0]>='0' && Ua
57 {
58     int ErrorLevel = UartRea
    SYS_DEBUG_BreakPoint();
60     SYS_DEBUG_ErrorLevelSet(
```

- What's the effect if change Debug Level configure in MHC ?



COMMAND Processor System Service

Harmony System Service

COMMAND Processor System Service

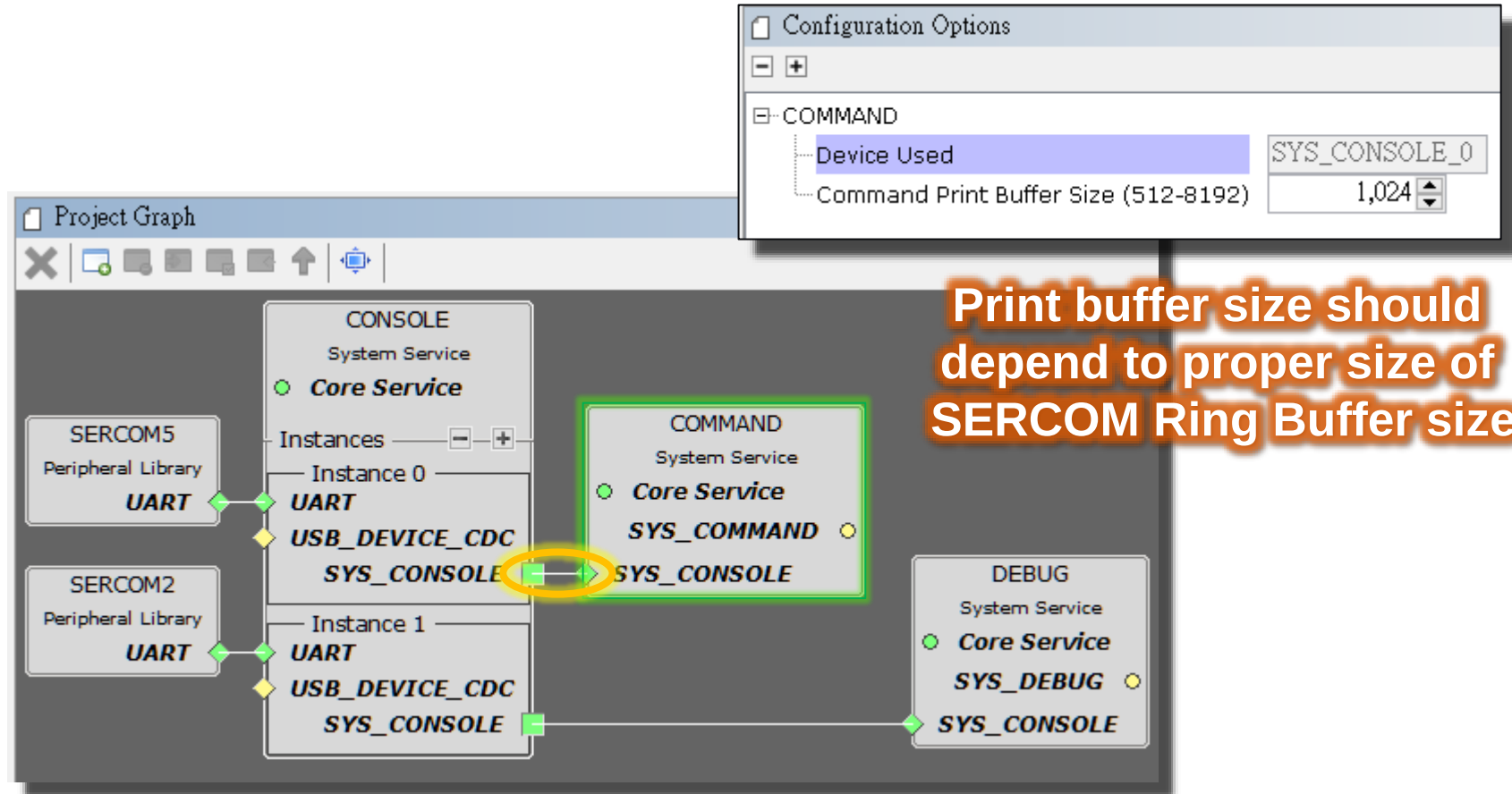
- Introduction

This library provides an abstraction of the Command Processor System Service Library with a convenient C language interface. It provides the framework for a command console that can support commands from multiple client sources.

The Command Processor System Service provides the developer with simple APIs to implement a command console.

The console may support commands from one or more client software modules.

COMMAND Processor System Service



Print buffer size should depend to proper size of SERCOM Ring Buffer size

COMMAND System Service must depend to CONSOLE system Service

COMMAND Processor System Service

API functions of the Command Processor System Service library.

- System Interaction Functions

[SYS_CMD_ADDGRP](#)

Allows clients to add command process to the Command Processor System Service.

[SYS_CMD_DELETE](#)

Removes CMDIO parameters from the Command Processor System Service console.

[SYS_CMD_MESSAGE](#)

Outputs a message to the Command Processor System Service console.

[SYS_CMD_PRINT](#)

Outputs a printout to the Command Processor System Service console.

[SYS_CMD_READY_TO_READ](#)

Allows upper layer application to confirm that the command module is ready to accept command input

[SYS_CMD_READY_TO_WRITE](#)

Allows upper layer application to confirm that the command module is ready to write output to the Console System Service.

[SYS_CMD_Tasks](#)

Maintains the Command Processor System Service's internal state machine.

[SYS_CMD_Initialize](#)

Initializes data for the instance of the Command Processor module.

- Command I/O Functions

[SYS_CMDIO_GET_HANDLE](#)

Gets the CMDIO Handle base via index.

- Data Types and Constants

[COMMAND_HISTORY_DEPTH](#)

Maximum Depth of Command History. (Up to 4 defined in [sys_command.h](#))

[MAX_CMD_ARGS](#)

Maximum Number of Argument definitions. (Up to 8 defined in [sys_command.h](#))

[MAX_CMD_GROUP](#)

Maximum Number of Command Group definitions. (Up to 8 defined in [sys_command.h](#))

[SYS_CMD_MAX_LENGTH](#)

Command Buffer Maximum Length definition. (Up to 80 bytes defined in [sys_command.h](#))

[SYS_CMD_READ_BUFFER_SIZE](#)

Read Buffer Size definition. (Up to 128 bytes defined in [sys_command.h](#))

[SYS_CMD_DEVICE_NODE](#)

Defines the data structure to store each command instance.

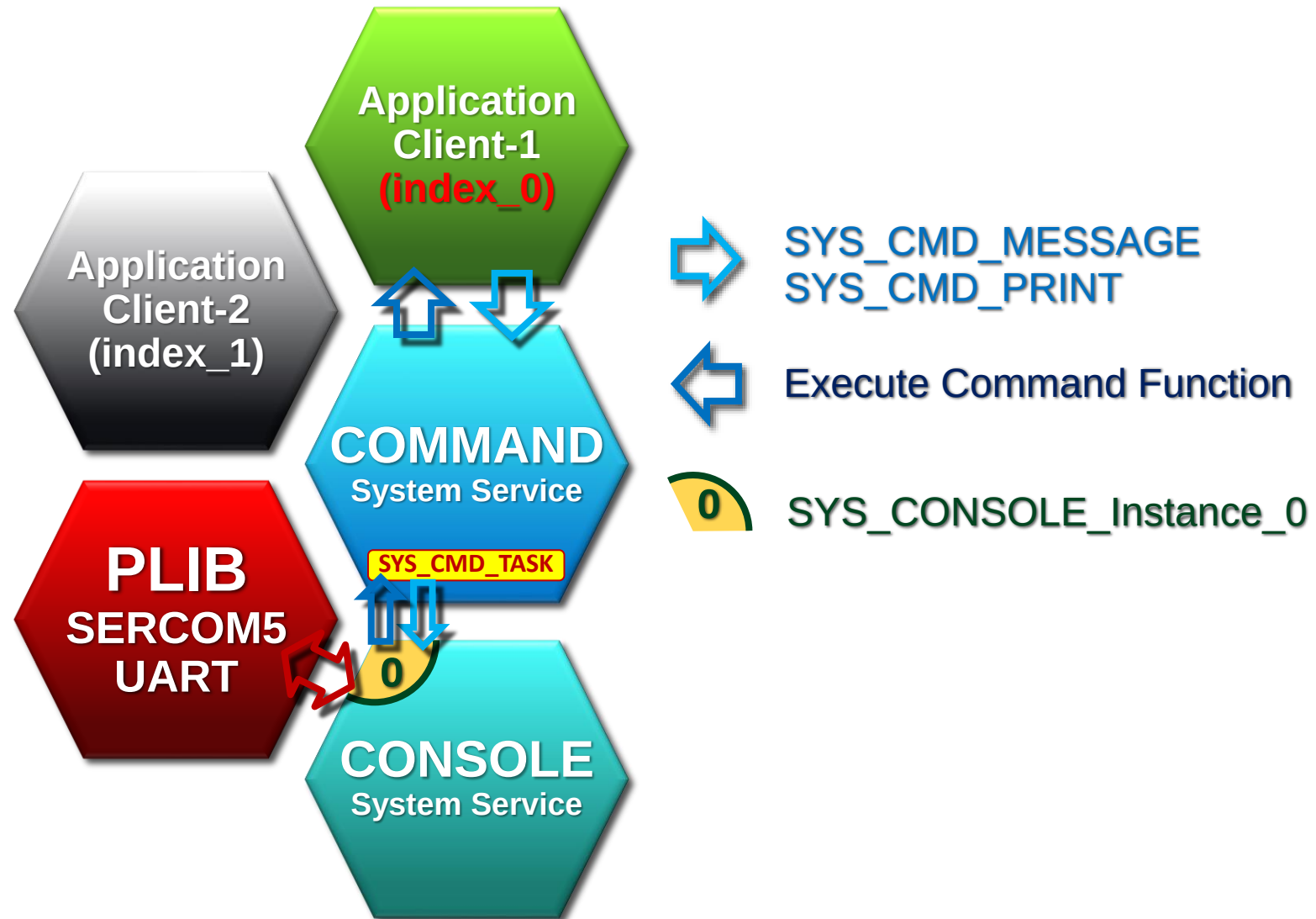
COMMAND Processor System Service

API functions of the Command Processor System Service library. (cont.)

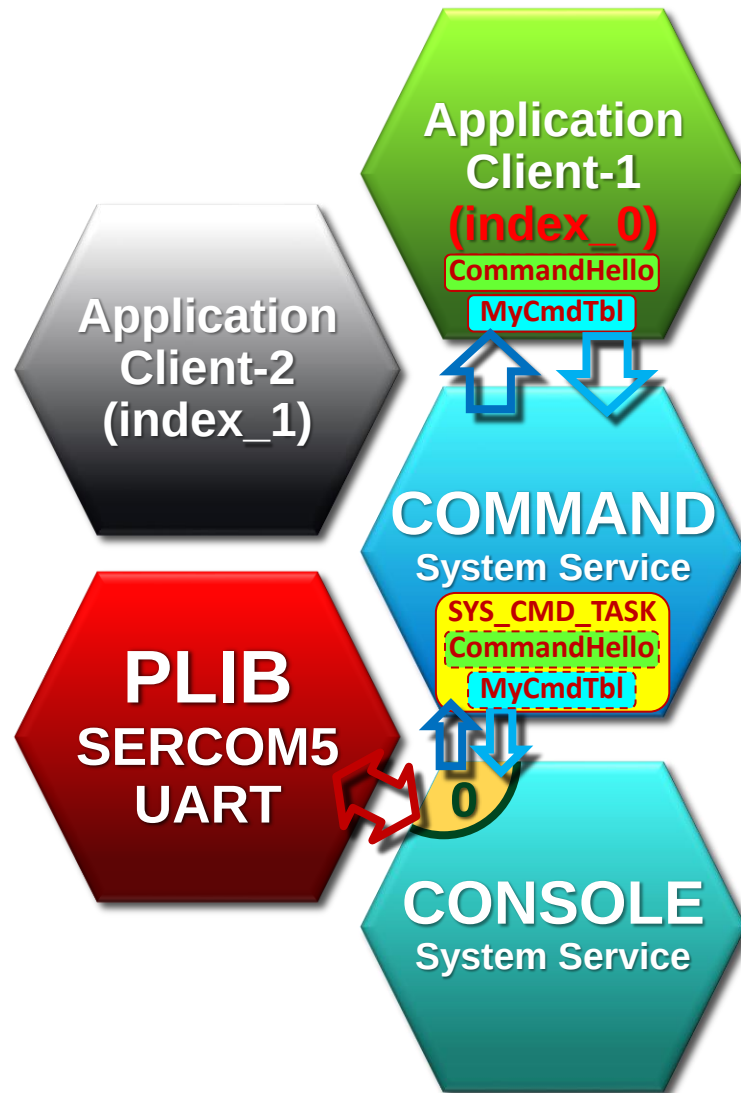
- Data Types and Constants (cont.)

<code>SYS_CMD_DEVICE_NODE</code>	Defines the data structure to store each command instance.
<code>SYS_CMD_API</code>	Identifies the Command API structure.
<code>SYS_CMD_DATA_RDY_FNC</code>	Identifies a data available function API.
<code>SYS_CMD_DESCRIPTOR</code>	A simple command descriptor
<code>SYS_CMD_FNC</code>	Identifies the command process function API.
<code>SYS_CMD_GETC_FNC</code>	Identifies a get single character function API.
<code>SYS_CMD_HANDLE</code>	Identifies a particular Command I/O instance.
<code>SYS_CMD_MSG_FNC</code>	Identifies a message function API.
<code>SYS_CMD_PRINT_FNC</code>	Identifies a print function API.
<code>SYS_CMD_PUTC_FNC</code>	Identifies a single character print function API.
<code>SYS_CMD_READC_FNC</code>	Identifies a read single character function API.
<code>SYS_CMD_CallbackFunction</code>	Command Callback Function Handle.
<code>SYS_CMD_CONSOLE_IO_PARAM</code>	Defines whether the command module is single character or full command read
<code>SYS_CMD_EVENT</code>	Identifies the Command Event Types
<code>SYS_CMD_INIT</code>	Identifies the system command initialize structure.
<code>SYS_CMD_DEVICE_MAX_INSTANCES</code>	This is macro <code>SYS_CMD_DEVICE_MAX_INSTANCES</code> .
<code>SYS_CMD_MESSAGE</code>	This is macro <code>SYS_CMD_MESSAGE</code> .
<code>SYS_CMD_PRINT</code>	This is macro <code>SYS_CMD_PRINT</code> .
<code>SYS_CMD_BUFFER_DMA_READY</code>	Define this for PIC32MX / PIC32MZ
<code>SYS_CMDIO_ADD</code>	Adds CMDIO parameters to the Command Processor System Service console.

COMMAND Processor System Service



COMMAND Processor System Service



```
void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO,  
                  int argc, char** argv)  
{  
    SYS_CMD_MESSAGE("\r\n- Hello !");  
    SYS_CMD_PRINT("\r\n-parameter count : %d\r\n", argc);  
}  
  
static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=  
{  
    { "hello", CommandHello, ": Hello!" },  
};  
  
void App1_Task()  
{  
    SYS_CMD_ADDGRP(MyCmdTbl, 1, "MYCMD", ": My Commands");  
}
```

```
VT COM24 - Tera Term VT  
>hello  
- Hello !  
- parameter count : 1  
  
>help  
----- Supported command groups -----  
*** MYCMD: My Commands ***  
  
>help MYCMD  
*** hello: Hello! ***  
>
```

Lab 5

Custom Command Console

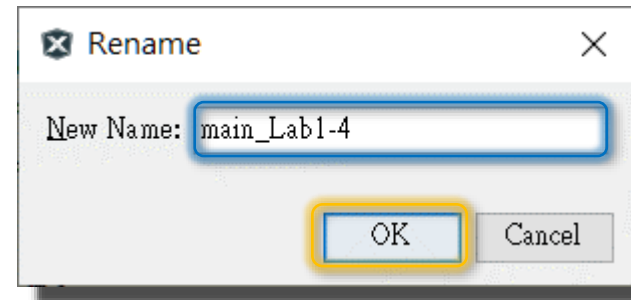
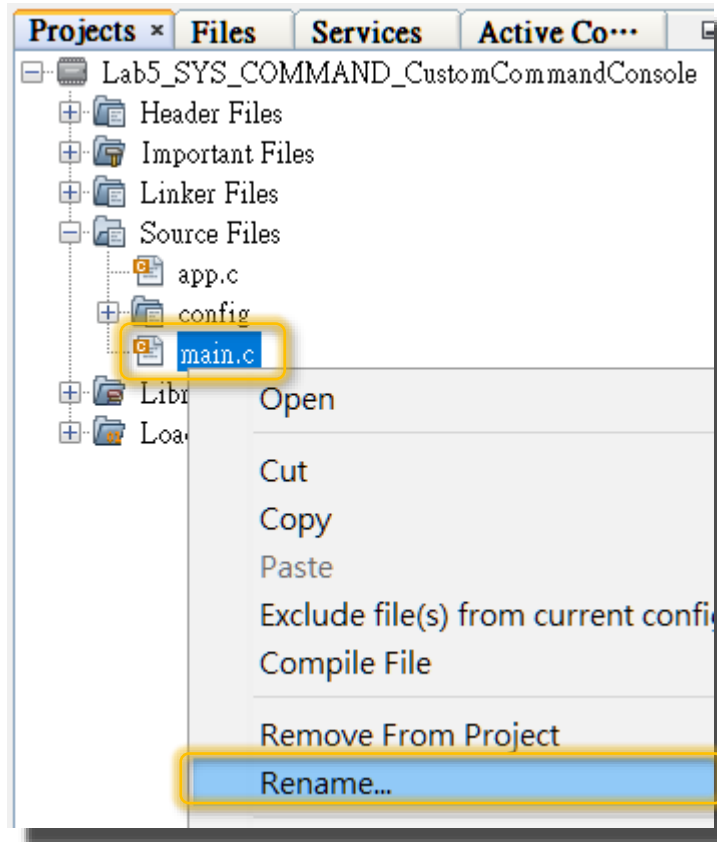
COMMAND Processor System Service

Lab5 Custom Command Console

- Try add **COMMAND processor system service** and communicate through UART to use previous added **CONSOLE system service**.
- In this Lab, you might create a new **main.c** to continue.
- Build your custom command to have below activity :
 - hello** : Print "Hello world !" on console.
 - led # on** : Light LED #.
 - led # off** : Dark LED #.
 - errlv #** : Change Ddebug Error Level to # and output message "Change Error Level to #" then control five different Error Level debug message output as below.
 - "Error Level 0 - Fatal"
 - "Error Level 1 - Error"
 - "Error Level 2 - Warning"
 - "Error Level 3 - Info"
 - "Error Level 4 - Debug"

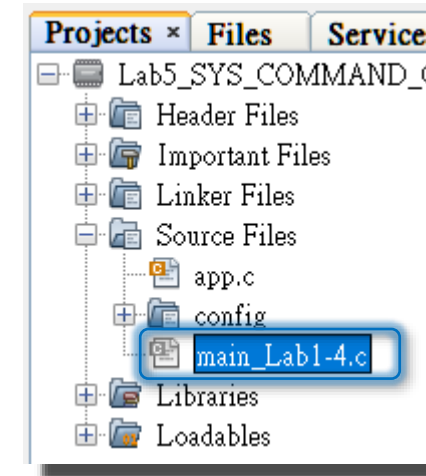
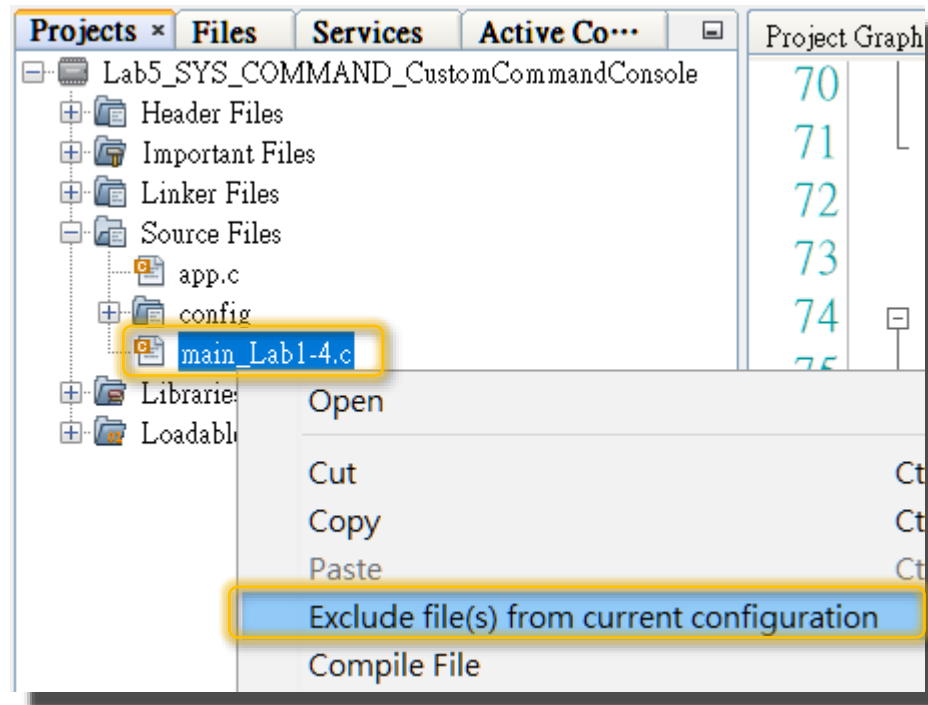
⚙️ Lab5 Custom Command Console

- Right click on main.c and rename it to **main_Lab1-4**



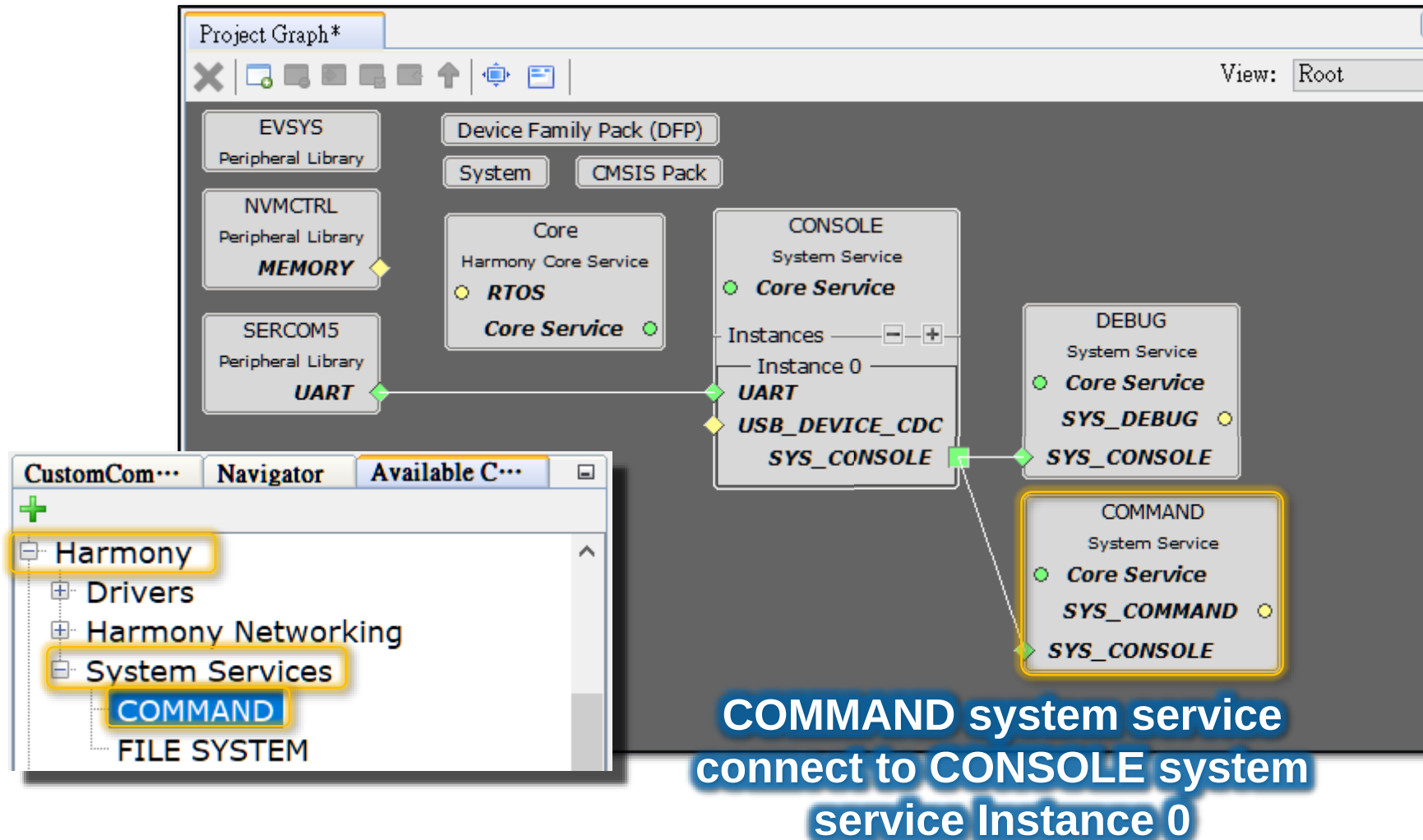
⚙️ Lab5 Custom Command Console

- Right click on `main_Lab1-4.c` and exclude it from current project
- The **new main.c** will **auto generate** while next GenCode to be executed.



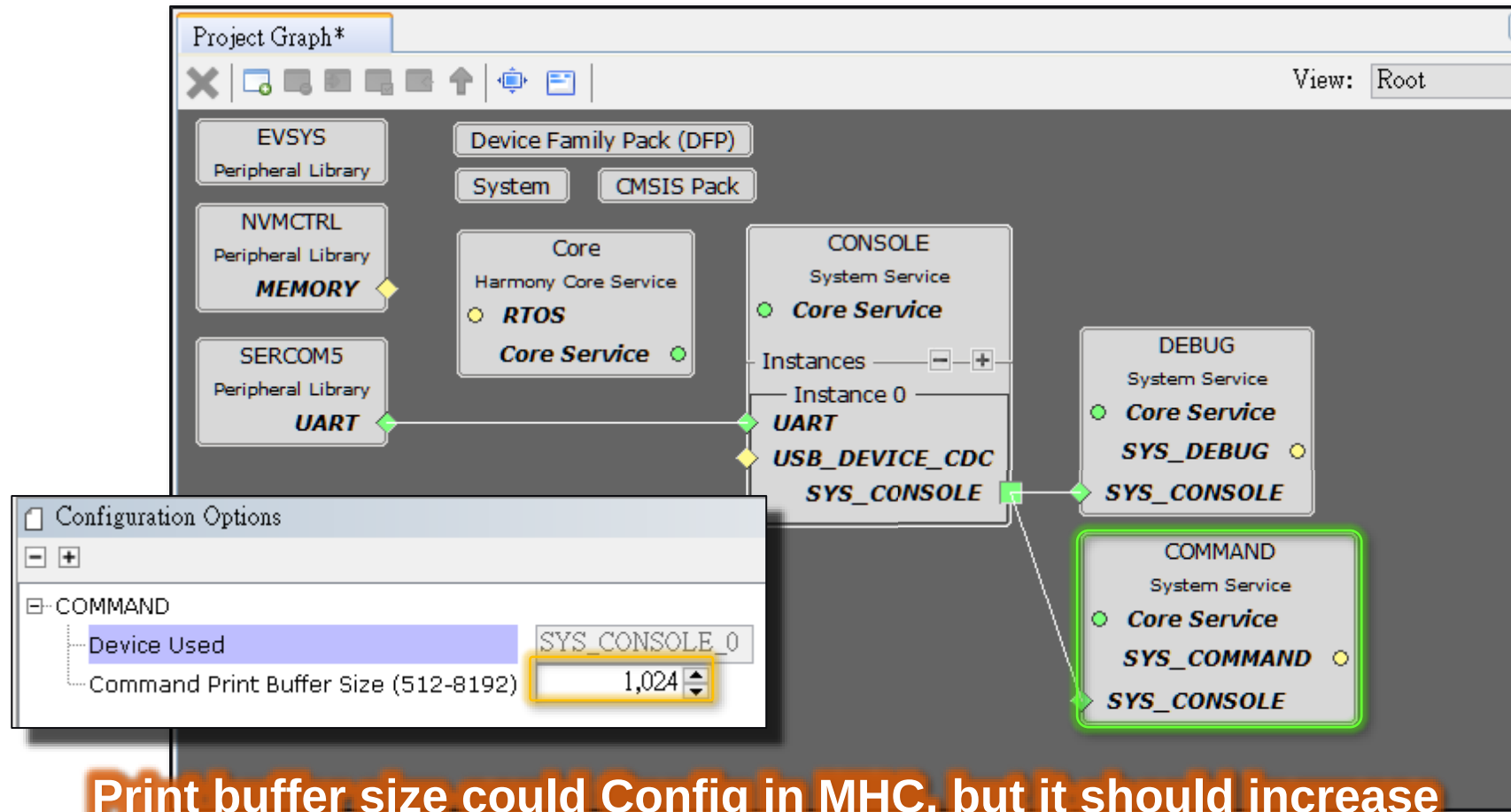
⚙️ Lab5 Custom Command Console

- Harmony Configuration (Add COMMAND system service)



⚙️ Lab5 Custom Command Console

- Harmony Configuration (COMMAND system service config)



Print buffer size could Config in MHC, but it should increase proper size of SERCOM Ring Buffer simultaneously

Lab5 Custom Command Console

- Modify your main.c as below

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

void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_CMD_MESSAGE("Hello world !\r\n");
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=
{
    { "hello", CommandHello, ": Say Hello" },
};

int main( void )
{
    SYS_Initialize(NULL);

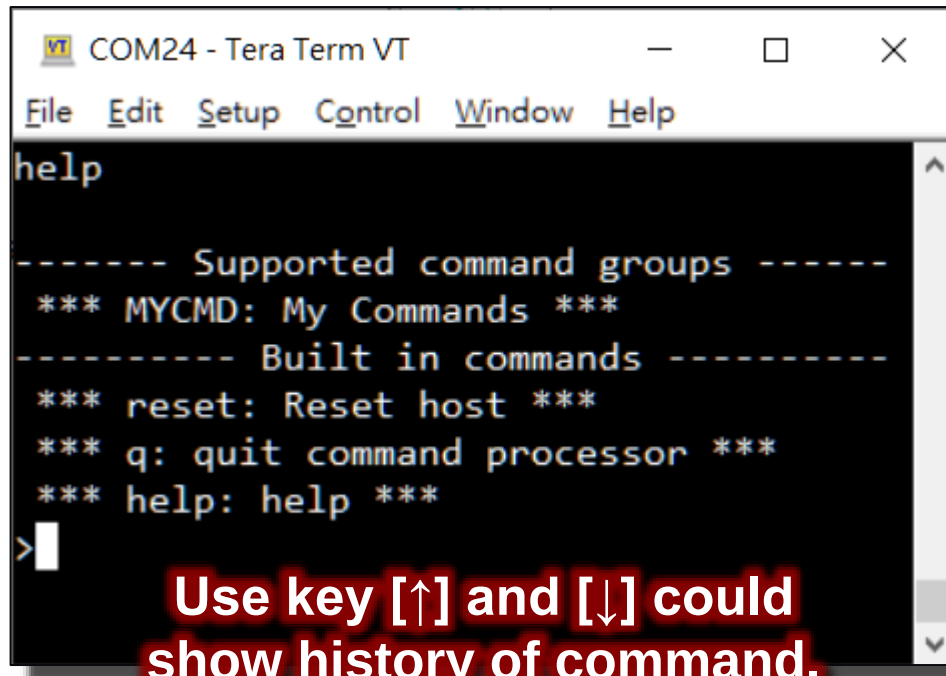
    SYS_CMD_ADDGRP(MyCmdTbl, 1, "MYCMD", ": My Commands");

    while( true )
    {
        SYS_Tasks();
    }

    return ( EXIT_FAILURE);
}
```

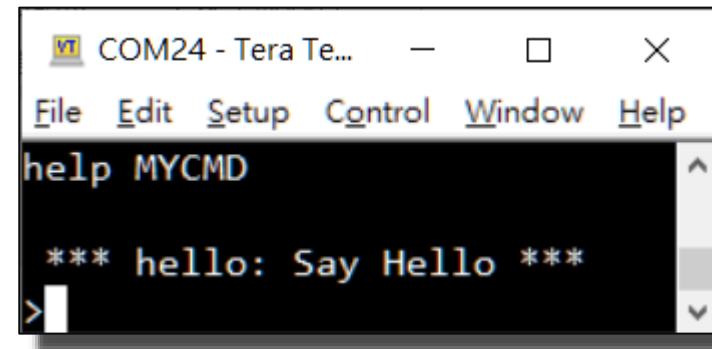
⚙️ Lab5 Custom Command Console

- Compiler and programming your project to EVB.
- Type 'help' + [Enter] will show system command help.
- Type 'help MYCMD' + [Enter] will show custom MYCMD help.
- Type 'hello' + [Enter] will execute hello command function.

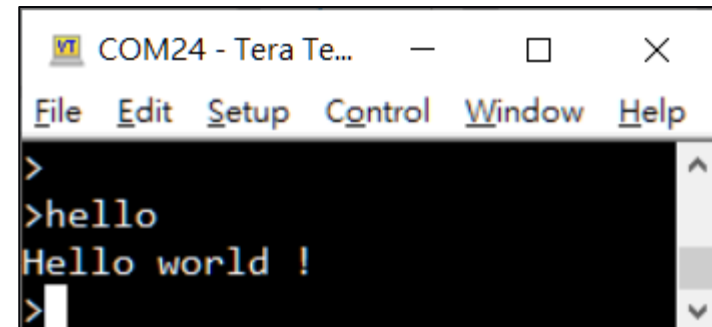


```
COM24 - Tera Term VT
File Edit Setup Control Window Help
help
----- Supported command groups -----
*** MYCMD: My Commands ***
----- Built in commands -----
*** reset: Reset host ***
*** q: quit command processor ***
*** help: help ***
>
```

Use key [↑] and [↓] could show history of command.



```
COM24 - Tera Te...
File Edit Setup Control Window Help
help MYCMD
*** hello: Say Hello ***
>
```



```
COM24 - Tera Te...
File Edit Setup Control Window Help
>
>hello
Hello world !
>
```

Lab5 Custom Command Console

- Modify your main.c as below

```
void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }

void CommandLED(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    if(argc == 3)
    {
        int Led, OnOff = 0;

        if(!strcmp( argv[2], "on" ))
            OnOff = 1;

        Led = atoi(argv[1]);
        switch(Led)
        {
            case 1: (OnOff ? LED1_Clear() : LED1_Set()); break;
            case 2: (OnOff ? LED2_Clear() : LED2_Set()); break;
            case 3: (OnOff ? LED3_Set()   : LED3_Clear()); break;
        }
    }
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=
{
    { "hello", CommandHello, ": Say Hello" },
    { "led",   CommandLED,   ": led # on[off]" },
};

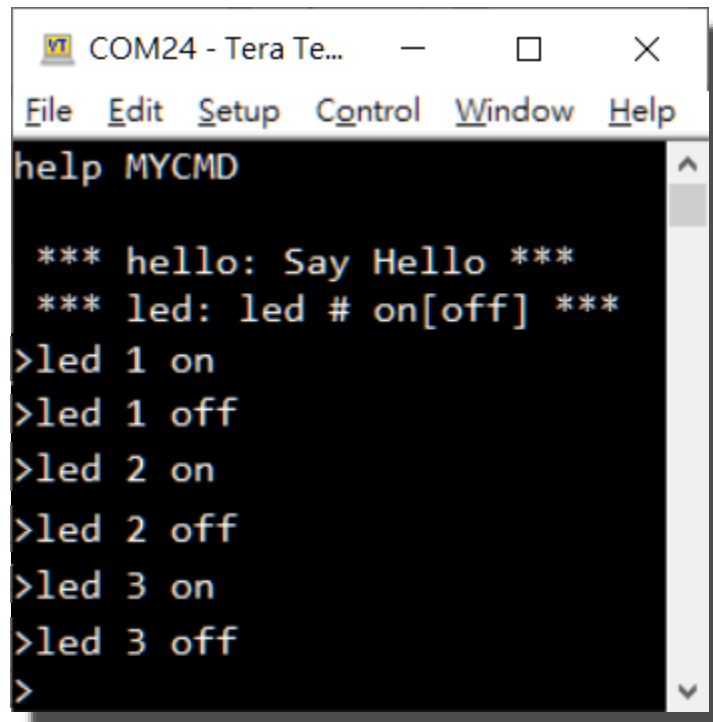
int main( void )
{
    SYS_Initialize(NULL);

    SYS_CMD_ADDGRP(MyCmdTbl, 2, "MYCMD", ": My Commands");
```

Modify this line

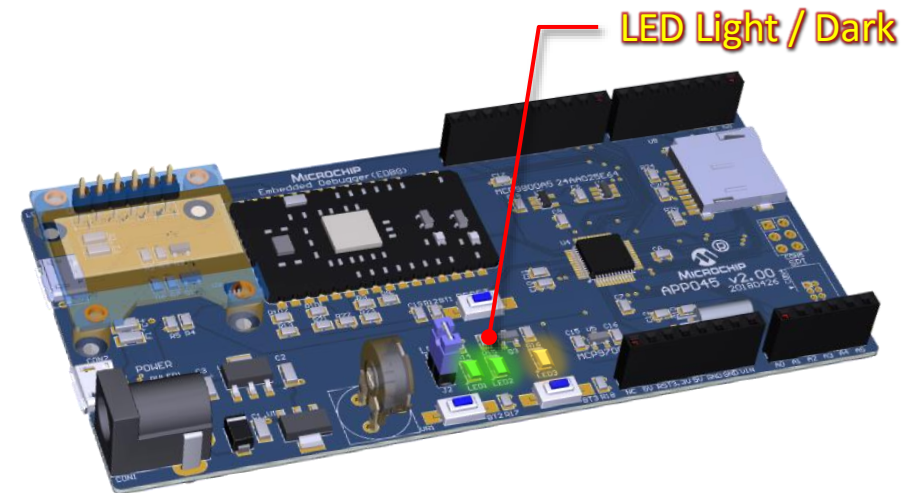
⚙️ Lab5 Custom Command Console

- Compiler and programming your project to EVB.
- Type 'help MYCMD' + [Enter] will show custom MYCMD help.
- Type 'led 1 on' + [Enter] to light **LED1**, 'led 1 off' to dark **LED1**.
- Same control command to **LED2** and **LED3**.



```
COM24 - Tera Te...  
File Edit Setup Control Window Help  
help MYCMD  
  
*** hello: Say Hello ***  
*** led: led # on[off] ***  
>led 1 on  
>led 1 off  
>led 2 on  
>led 2 off  
>led 3 on  
>led 3 off  
>
```

Use command to control LED.



⚙️ Lab5 Custom Command Console

- Modify your main.c as below

```
void CommandHello(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }
void CommandLED(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }
void CommandErrLv(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    if(argc == 2)
    {
        int ErrorLevel = atoi(argv[1]);
        if(ErrorLevel >= 0 && ErrorLevel <= 4 )
        {
            SYS_DEBUG_ErrorLevelSet(ErrorLevel);
            SYS_CMD_PRINT("Change Error Level to %d\r\n", SYS_DEBUG_ErrorLevelGet());
        }
    }
    SYS_DEBUG_MESSAGE(SYS_ERROR_FATAL, "Error Level 0 - Fatal\r\n");
    SYS_DEBUG_MESSAGE(SYS_ERROR_ERROR, "Error Level 1 - Error\r\n");
    SYS_DEBUG_MESSAGE(SYS_ERROR_WARNING, "Error Level 2 - Warning\r\n");
    SYS_DEBUG_MESSAGE(SYS_ERROR_INFO, "Error Level 3 - Info\r\n");
    SYS_DEBUG_MESSAGE(SYS_ERROR_DEBUG, "Error Level 4 - Debug\r\n");
}

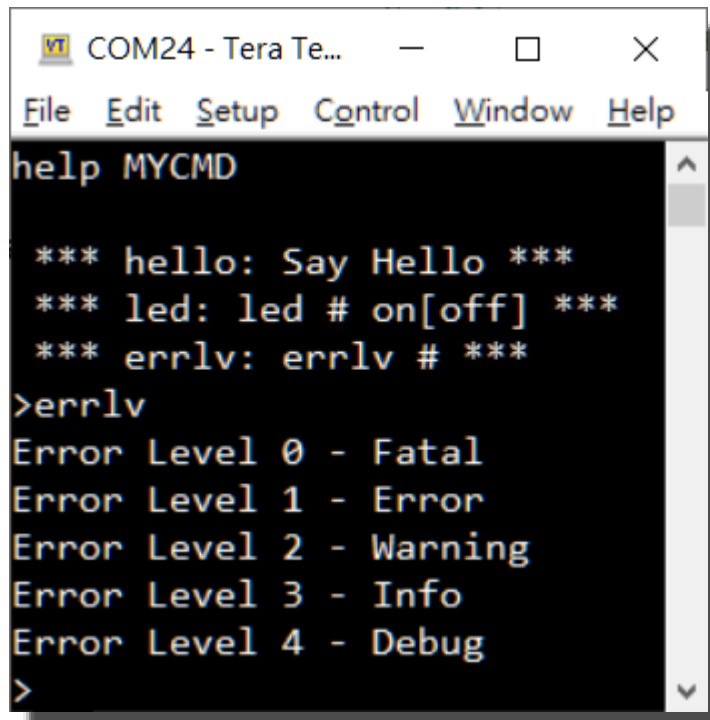
static const SYS_CMD_DESCRIPTOR MyCmdTbl[] =
{
    { "hello", CommandHello, ": Say Hello" },
    { "led", CommandLED, ": led # on[off]" },
    { "errlv", CommandErrLv, ": errlv #" },
};

int main( void )
{
    SYS_Initialize(NULL);
    SYS_CMD_ADDGRP(MyCmdTbl, 3, "MYCMD", ": My Commands");
```

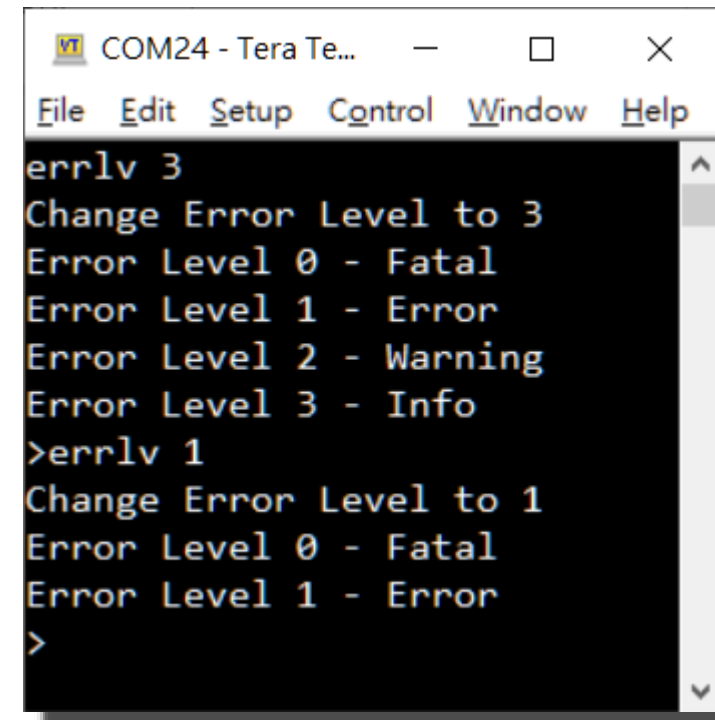
Modify this line

⚙️ Lab5 Custom Command Console

- Compiler and programming your project to EVB.
- Type 'help MYCMD' + [Enter] will show custom MYCMD help.
- Type 'errlv' + [Enter] to show five error level debug messages.
- Type 'errlv 3' + [Enter] change error level to 3 and show only error level ≤ 3 debug messages, 'errlv 1' change error level to 1.



```
COM24 - Tera Te...  
File Edit Setup Control Window Help  
help MYCMD  
  
*** hello: Say Hello ***  
*** led: led # on[off] ***  
*** errlv: errlv # ***  
>errlv  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3 - Info  
Error Level 4 - Debug  
>
```

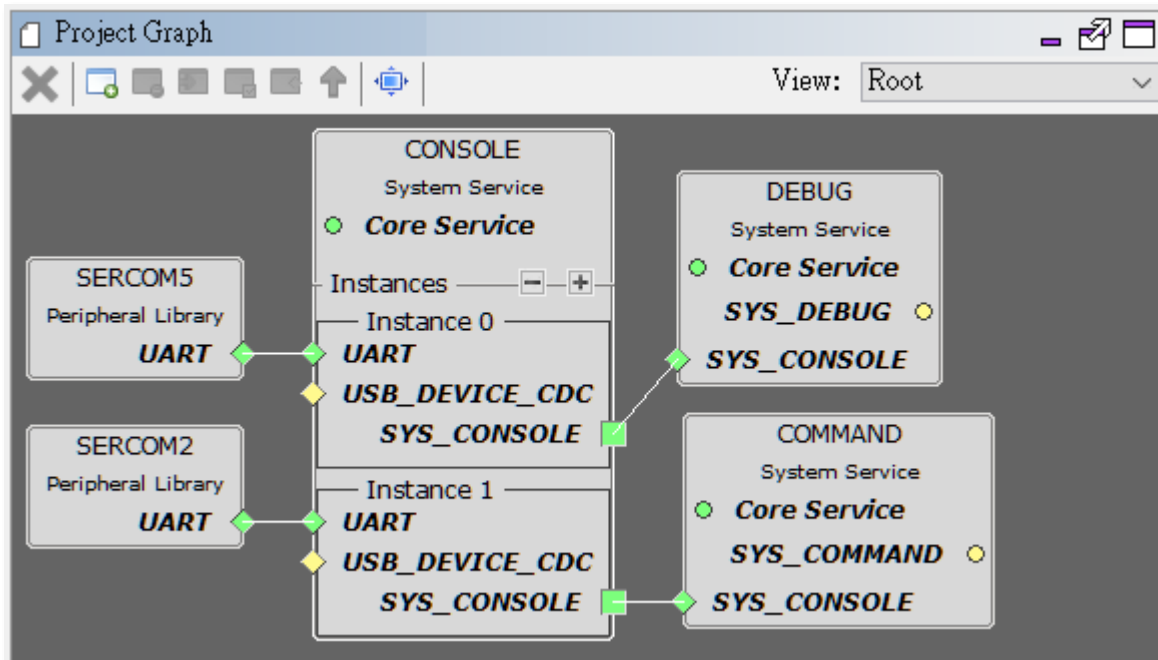


```
COM24 - Tera Te...  
File Edit Setup Control Window Help  
errlv 3  
Change Error Level to 3  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3 - Info  
>errlv 1  
Change Error Level to 1  
Error Level 0 - Fatal  
Error Level 1 - Error  
>
```


⚙️ Lab5 Custom Command Console

- **Discuss**

- Try implement two Instance of CONSOLE system service to separate DEBUG and COMMAND console.



```
COM24 - Tera Te...  
File Edit Setup Control Window Help  
DEBUG console here!  
Error Level 0 - Fatal  
Error Level 1 - Error  
Error Level 2 - Warning  
Error Level 3 - Info
```

```
COM30 - Tera Te...  
File Edit Setup Control Window Help  
COMMAND console here!  
>led 1 on  
>errlv 3  
Change Error Level to 3  
>
```

USB Device CDC Library

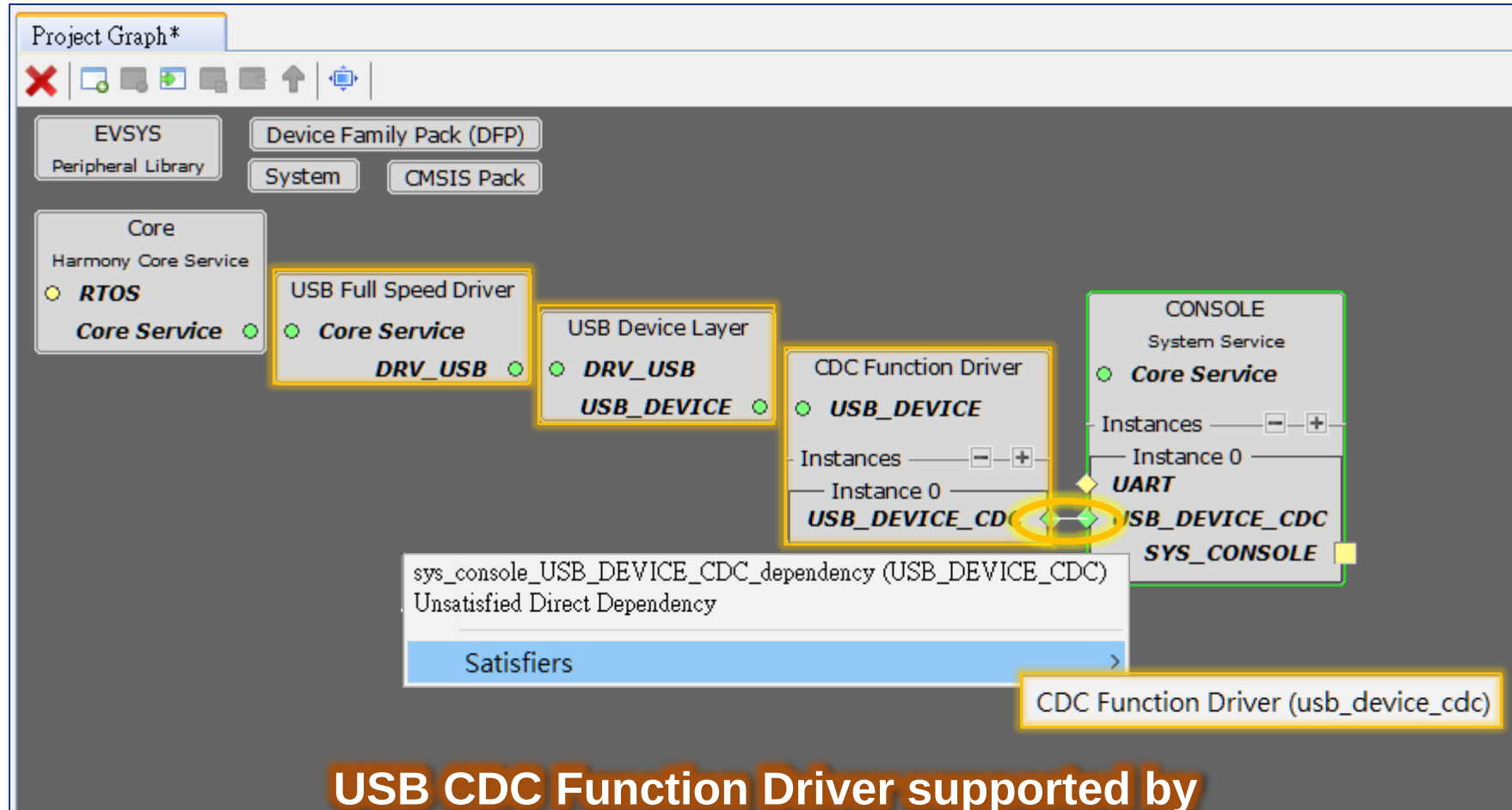
Harmony Device Library

USB Device CDC Library

- **Introduction**

The MPLAB Harmony USB Communications Device Class (CDC) Device Library (also referred to as the CDC function driver or library) provides functions and methods that allow application designers to implement a USB Device CDC. The current version of the library supports the Abstract Control Model (ACM) of the CDC specification revision 1.2 and specifically implements a subset of the AT250 command set. This library must be used in conjunction with the MPLAB Harmony USB Device Layer.

USB Device CDC Library



**USB CDC Function Driver supported by
CONSOLE System Service**

USB Device CDC Library

API functions of the USB Device CDC library.

Functions

USB_DEVICE_CDC_EventHandlerSet

This function registers an event handler for the specified CDC driver instance.

USB_DEVICE_CDC_Read

This function requests a data read from the USB Device CDC Function Driver Layer.

USB_DEVICE_CDC_Write

This function requests a data write to the USB Device CDC Function Driver Layer.

USB_DEVICE_CDC_SerialStateNotificationSend

This function schedules a request to send serial state notification to the host.

Data Types and Constants

USB_DEVICE_CDC_EVENT_DATA_READ_COMPLETE

USB Device CDC Function Driver Read and Write Complete Event Data.

USB_DEVICE_CDC_EVENT_DATA_SERIAL_STATE_NOTIFICATION_COMPLETE USB Device CDC Function Driver Read and Write Complete Event Data.

USB_DEVICE_CDC_EVENT_DATA_WRITE_COMPLETE

USB Device CDC Function Driver Read and Write Complete Event Data.

USB_DEVICE_CDC_INIT

USB Device CDC Function Driver Initialization Data Structure

USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK

USB Device CDC Function Driver Send Break Event Data

USB_DEVICE_CDC_EVENT

USB Device CDC Function Driver Events

USB_DEVICE_CDC_EVENT_HANDLER

USB Device CDC Event Handler Function Pointer Type.

USB_DEVICE_CDC_EVENT_RESPONSE

USB Device CDC Function Driver Event Callback Response Type

USB_DEVICE_CDC_INDEX

USB Device CDC Function Driver Index

USB_DEVICE_CDC_RESULT

USB Device CDC Function Driver USB Device CDC Result enumeration.

USB_DEVICE_CDC_TRANSFER_FLAGS

USB Device CDC Function Driver Transfer Flags

USB_DEVICE_CDC_TRANSFER_HANDLE

USB Device CDC Function Driver Transfer Handle Definition.

USB_DEVICE_CDC_EVENT_RESPONSE_NONE

USB Device CDC Function Driver Event Handler Response Type None.

USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID

USB Device CDC Function Driver Invalid Transfer Handle Definition.

USB_DEVICE_CDC_FUNCTION_DRIVER

USB Device CDC Function Driver Function pointer

USB Device CDC Library

API functions of the USB Device CDC library. (cont.)

Data Types and Constants

USB_CDC_LINESTATE_CARRIER	Bit code information in line state
USB_CDC_LINESTATE_DTR	Bit code information in line state
USB_CDC_DESCRIPTOR_TYPE	Identifies the descriptor types in the CDC.
USB_CDC_INF_PROTOCOL	Identifies the protocol codes.
USB_CDC_INTERFACE_TYPE	Identifies the CDC interface type.
USB_CDC_NOTIFICATION	Identifies the notification codes available for CDC.
USB_CDC_REQUEST	Identifies the CDC specific request codes.
USB_CDC_FUNCTIONAL_DESCRIPTOR	Identifies the CDC function header type.
USB_CDC_SUBCLASS	Identifies the subclass codes for communication interface.
USB_CDC_UNION_FUNCTIONAL_DESCRIPTOR_SUBORDINATE	This is type USB_CDC_UNION_FUNCTIONAL_DESCRIPTOR_SUBORDINATE.
CS_INTERFACE	This is a macro.
USB_CDC_ACM_SUPPORT_BREAK	This is a macro.
USB_CDC_ACM_SUPPORT_COMM_FEATURE	This is a macro.
USB_CDC_ACM_SUPPORT_LINE_CODING_LINE_STATE_AND_NOTIFICATION	This is a macro.
USB_CDC_ACM_SUPPORT_NETWORK_NOTIFICATION	This is a macro.
USB_CDC_ACM_SUPPORT_NONE	Identifies the CDC ACM sub-class capabilities.
USB_CDC_CLASS_CODE	Identifies the CDC Interface Class, Subclass and protocol constants.
USB_CDC_COMMUNICATIONS_INTERFACE_CLASS_CODE	This is a macro.
USB_CDC_DATA_INTERFACE_CLASS_CODE	This is a macro.
USB_CDC_DATA_INTERFACE_PROTOCOL	This is a macro.
USB_CDC_DATA_INTERFACE_SUBCLASS_CODE	This is a macro.

USB Device CDC Library

API functions of the USB Device CDC library. (cont.)

Data Types and Constants

USB_CDC_REQUEST_CLASS_SPECIFIC

USB_CDC_SUBCLASS_CODE

USB_DEVICE_CDC_INDEX_0

USB_DEVICE_CDC_INDEX_1

USB_DEVICE_CDC_INDEX_2

USB_DEVICE_CDC_INDEX_3

USB_DEVICE_CDC_INDEX_4

USB_DEVICE_CDC_INDEX_5

USB_DEVICE_CDC_INDEX_6

USB_DEVICE_CDC_INDEX_7

CDC specific request

This is a macro.

Use this to specify CDC Function Driver Instance 0

Use this to specify CDC Function Driver Instance 1

Use this to specify CDC Function Driver Instance 2

Use this to specify CDC Function Driver Instance 3

Use this to specify CDC Function Driver Instance 4

Use this to specify CDC Function Driver Instance 5

Use this to specify CDC Function Driver Instance 6

Use this to specify CDC Function Driver Instance 7

Lab 6

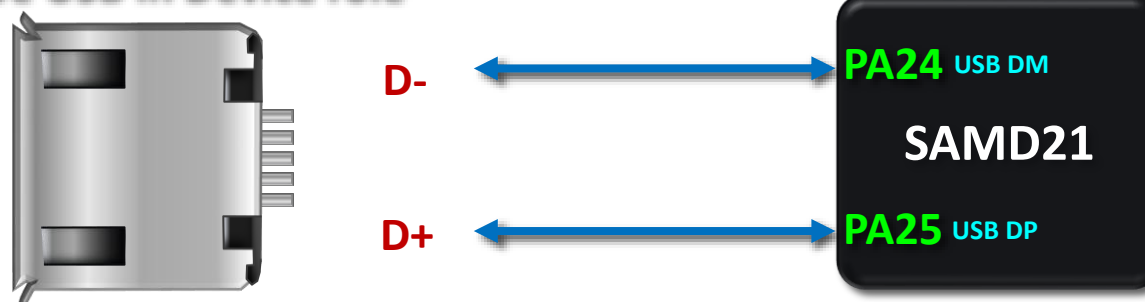
USB Device CDC with CONSOLE System Service

USB Device Library

🔧 Lab6 USB Device CDC with CONSOLE System Service

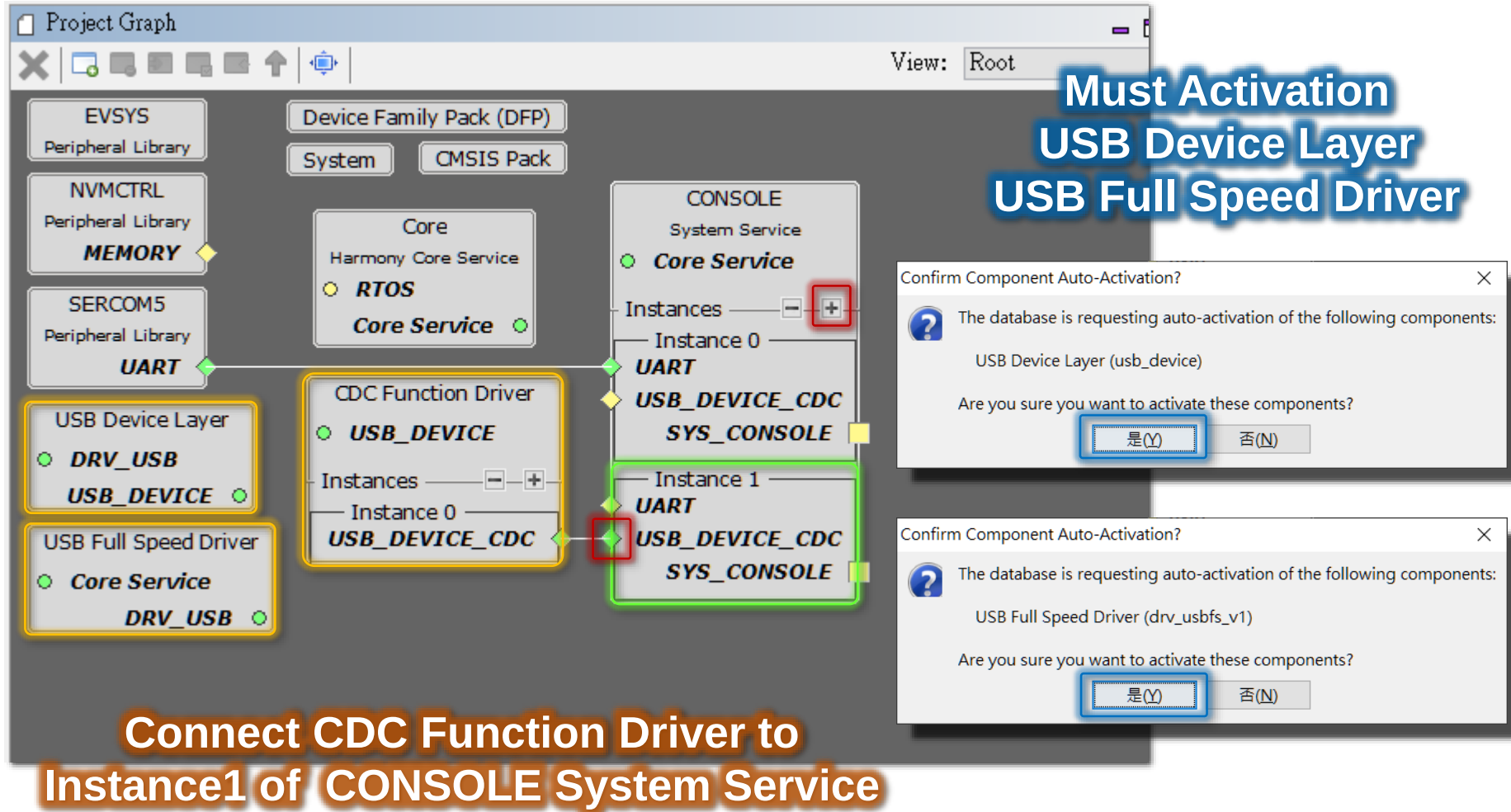
- This Lab related to [Lab3](#) of [CONSOLE System Service](#), please start from the solution project [Lab3_SYS_CONSOLE_UartConsole.zip](#)
- Add another **Instance 1** of [CONSOLE System Service](#).
- Connect [USB Device CDC](#) to **Instance 1** of [CONSOLE System Service](#), the **Device Layer** and **USB Full Speed Driver** will notice to add together.
- Config [PA24/PA25](#) as [USB DM/DP](#) pins.
- Disable [USB_VBUS_SENSE](#) pin feature in [USB Full Speed Driver](#).
- Output “[Hello World! \(PLIB\)](#)” and “[Hello World! \(USB\)](#)” to two consoles.
- Receive console input [X](#) and print “[Receive : X](#)” to opposite console.
- “[Receive : X \(PLIB\)](#)” (Receive in **PLIB** console and output to [USB](#) console)
- “[Receive : X \(USB\)](#)” (Receive in [USB](#) console and output to **PLIB** console)

Native USB in Device role



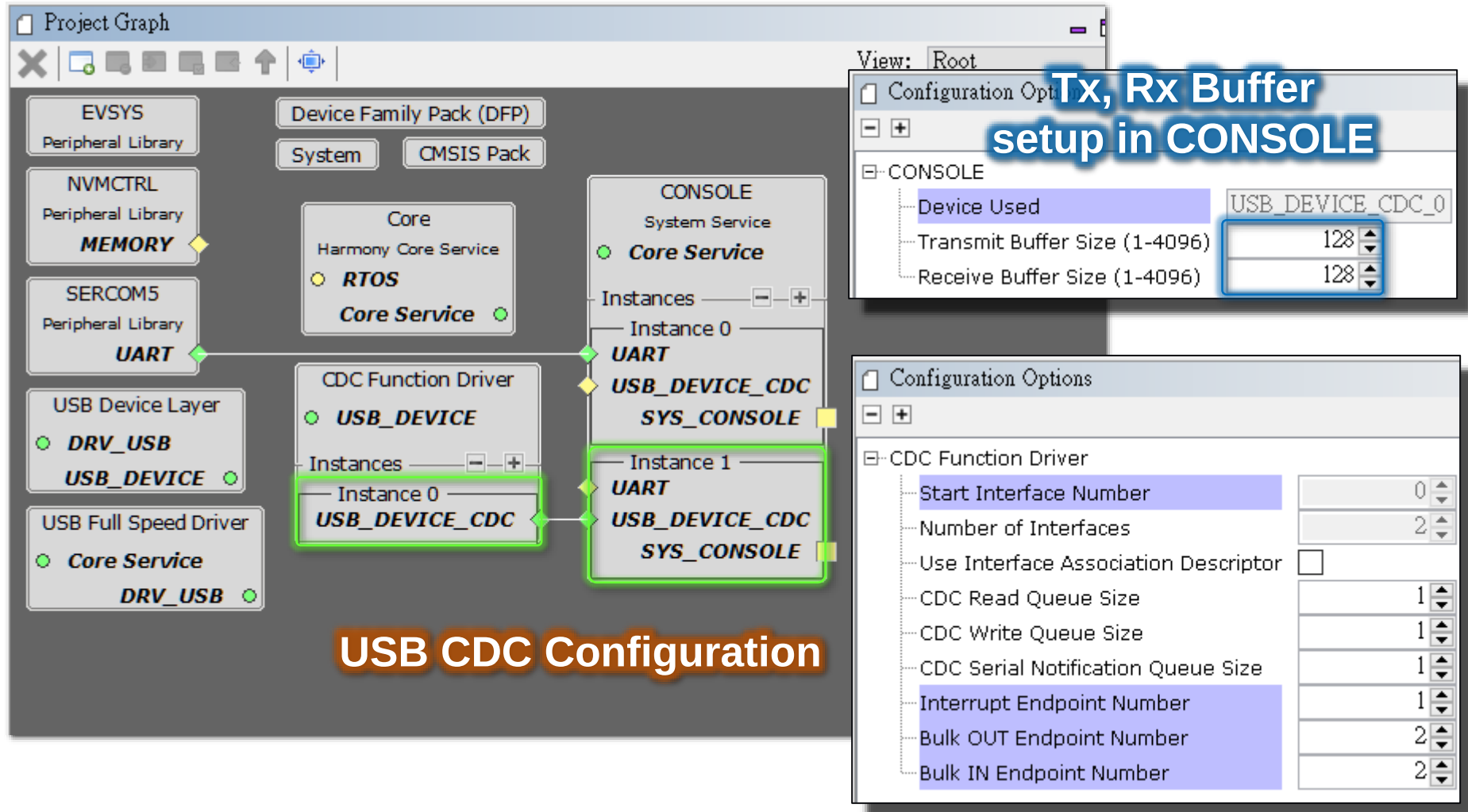
✂ Lab6 USB Device CDC with CONSOLE System Service

- Add USB CDC Function Driver and USB Device modules.



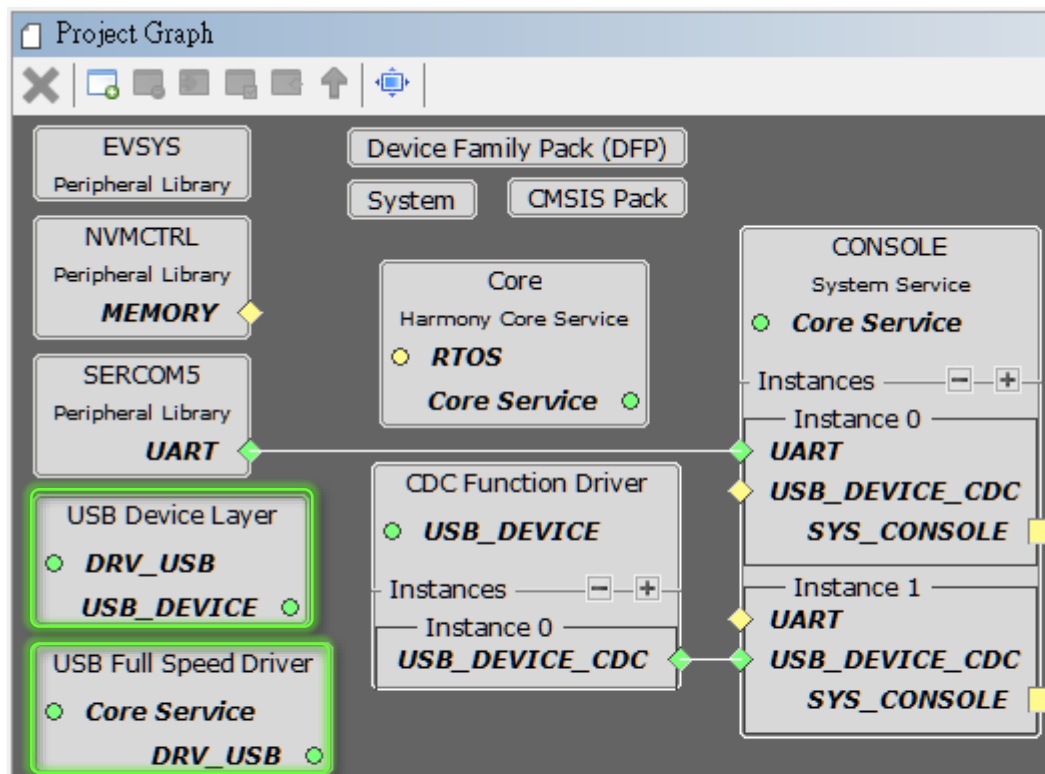
✂ Lab6 USB Device CDC with CONSOLE System Service

- Harmony Configuration of CONSOLE and CDC Function Driver.



Lab6 USB Device CDC with CONSOLE System Service

- Harmony Configuration of USB Device Layer and Full Speed Driver.



**Setup your USB VID / PID
Or use Microchip Demo PID**

USB Device Layer

Number of Functions: 1

Number of Endpoints: 2

Special Events: **cdc_com_port_single_demo**

Special Features:

Endpoint 0 Buffer Size: 64

Vendor ID: 0x04D8

Product ID Selection: **cdc_com_port_single_demo**

Product ID: 0x000A

Manufacturer String: Microchip Technology Inc.

Product String Selection: Simple CDC Device Demo

Add Device Serial Number: ☐

**VID:0x04D8 PID:0x000A
Simple CDC Device Demo**

**Disable the VBUS Sense feature
Because we did not design the pin
could detect USB VBUS Power**

Configuration Options* Help

USB Full Speed Driver

USB Mode Selection: Device

Enable VBUS Sense: ☐

✖ Lab6 USB Device CDC with CONSOLE System Service

- Setup USB pads D-/D+ to PA24/PA25

Pin Settings

Order: Ports Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
33	PA24	USB_DM	USB_DM	Digital	High Impedance	n/a	☐	☐	NORMAL
34	PA25	USB_DP	USB_DP	Digital	High Impedance	n/a	☐	☐	NORMAL

Configure PA24 as USB_DM
Configure PA25 as USB_DP

Pin Table

Package: TQFP48

		3	4	9	10	11	12	13	14	15	16	21	22	23	24	25	26	27	28	29	30	31	32	33	34
															A										
Module	Function	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	USB_DM																								
	USB_DP																								
USB																									

⚙️ Lab6 USB Device CDC with CONSOLE System Service

- Modify your main.c as below

```
SYS_TIME_HANDLE tmrHandle;
SYS_TIME_HANDLE tmrHandle_USB_hello;
SYS_CONSOLE_HANDLE consoleHandlePLIB;
SYS_CONSOLE_HANDLE consoleHandleUSB;
char UartWriteBuf[30];
char UartReadBuf[30];

...
int main( void )
{
    ...
    SYS_TIME_DelayMS(1000, &tmrHandle);
    SYS_TIME_DelayMS(1000, &tmrHandle_USB_hello);
    ...
    consoleHandlePLIB = SYS_CONSOLE_HandleGet( SYS_CONSOLE_INDEX_0 );
    sprintf(UartWriteBuf, "Hello world !(PLIB)\r\n");
    SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));

    consoleHandleUSB = SYS_CONSOLE_HandleGet( SYS_CONSOLE_INDEX_1 );

    while ( true )
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true ) {...}

        if( SYS_TIME_DelayIsComplete(tmrHandle_USB_hello) == true )
        {
            sprintf(UartWriteBuf, "Hello world !(USB)\r\n");
            SYS_CONSOLE_Write(consoleHandleUSB, UartWriteBuf, strlen(UartWriteBuf));
        }

        ...
    }
}
```

Lab6 USB Device CDC with CONSOLE System Service

- Modify your main.c as below

```
int main( void )
{
    ...
    consoleHandlePLIB = SYS_CONSOLE_HandleGet( SYS_CONSOLE_INDEX_0 );
    sprintf(UartWriteBuf, "Hello world !(PLIB)\r\n");
    SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));

    consoleHandleUSB = SYS_CONSOLE_HandleGet( SYS_CONSOLE_INDEX_1 );

    while ( true )
    {
        ...
        if( SYS_TIME_DelayIsComplete(tmrHandle_USB_hello) == true )
        {
            sprintf(UartWriteBuf, "Hello world !(USB)\r\n");
            SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));
        }

        if( SYS_CONSOLE_Read(consoleHandlePLIB, UartReadBuf, 1) > 0 )
        {
            sprintf(UartWriteBuf, "Receive : %s (PLIB)\r\n", UartReadBuf);
            SYS_CONSOLE_Write(consoleHandleUSB, UartWriteBuf, strlen(UartWriteBuf));
        }

        if( SYS_CONSOLE_Read(consoleHandleUSB, UartReadBuf, 1) > 0 )
        {
            sprintf(UartWriteBuf, "Receive : %s (USB)\r\n", UartReadBuf);
            SYS_CONSOLE_Write(consoleHandlePLIB, UartWriteBuf, strlen(UartWriteBuf));
        }

        SYS_Tasks ( );
    }
}
```

Modify this line

✖ Lab6 USB Device CDC with CONSOLE System Service

- Compiler and programming your project to EVB.
- Connect both two USB to your PC.
- The Device manager will detect a new USB CDC device.
- You can check the new COM Port properties.



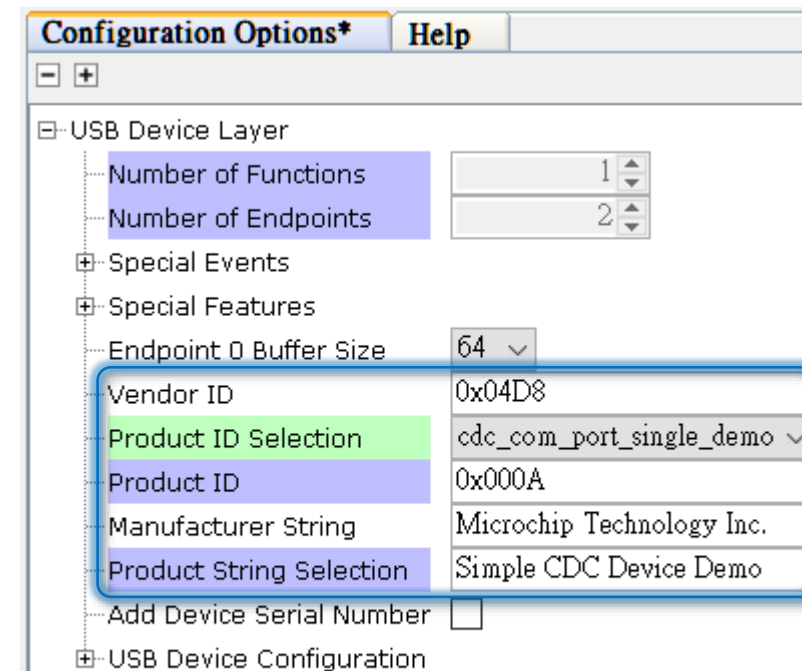
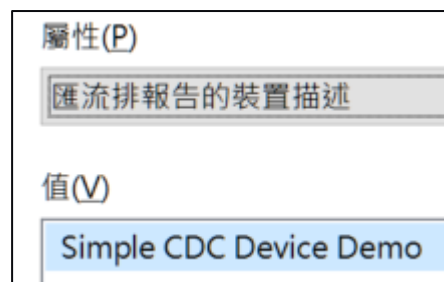
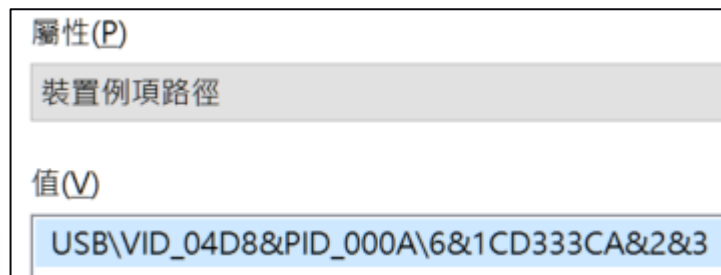
更新驅動程式(P)

停用裝置(D)

解除安裝裝置(U)

掃描硬體變更(A)

內容(R)



⚙ 裝置已就緒

'USB 序列裝置 (COM5)' 已設定完成並隨時可用。

⚙ 正在設定裝置

我們正在設定 'Simple CDC Device Demo'。

⚙️ Lab6 USB Device CDC with CONSOLE System Service

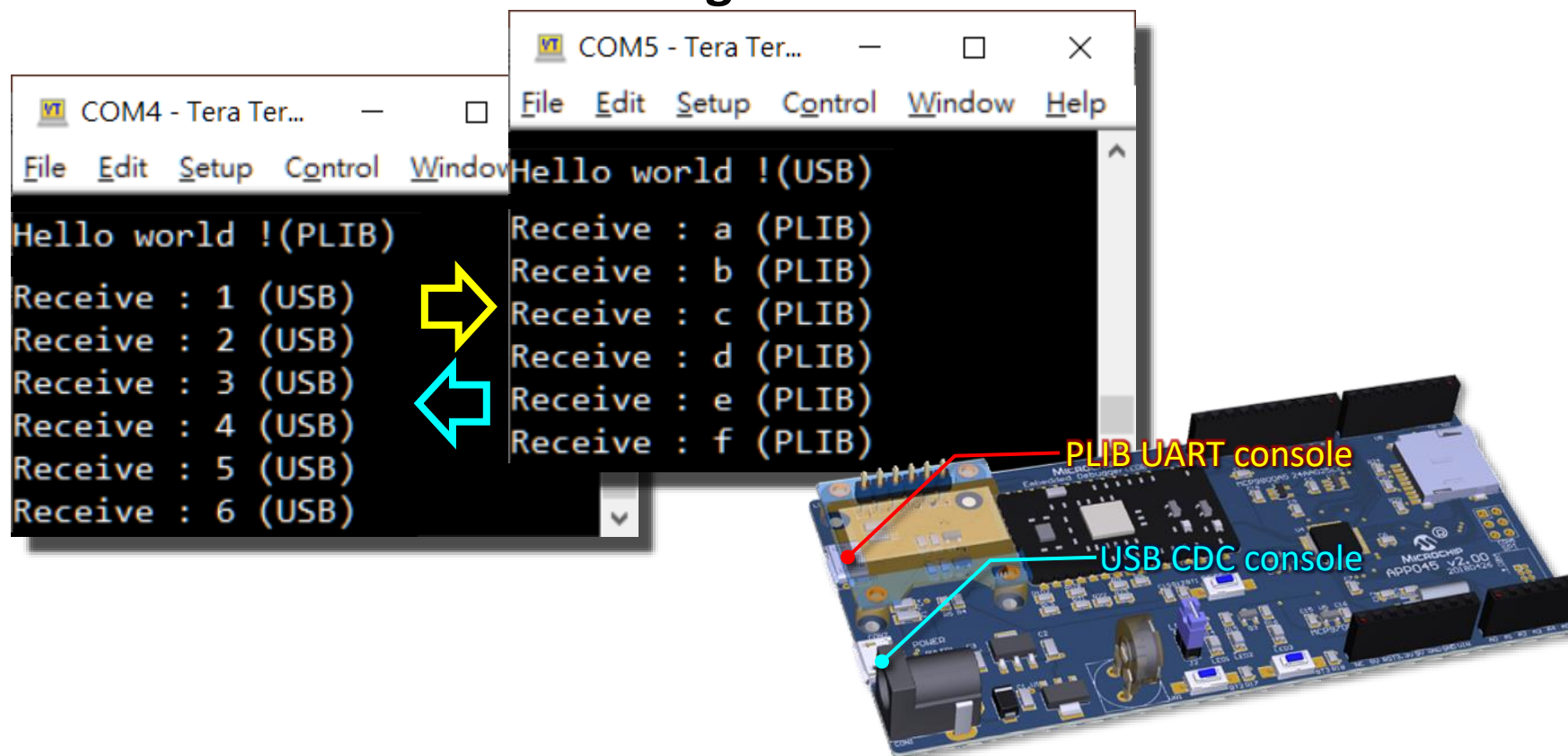
- If you are using Windows 7 or previous version, please install USB CDC driver from below path.

<Harmony Framework Path>\usb\apps\device\cdc_com_port_single\inf\



⚙️ Lab6 USB Device CDC with CONSOLE System Service

- Open two COM Port terminal console 115200-n-8-1.
- Press [Reset] button on EVB and check the 'Hello world!(xxx)' to know which one is from PLIB or from USB.
- Type-in in one console will echo message to another one.



Lab6 USB Device CDC with CONSOLE System Service

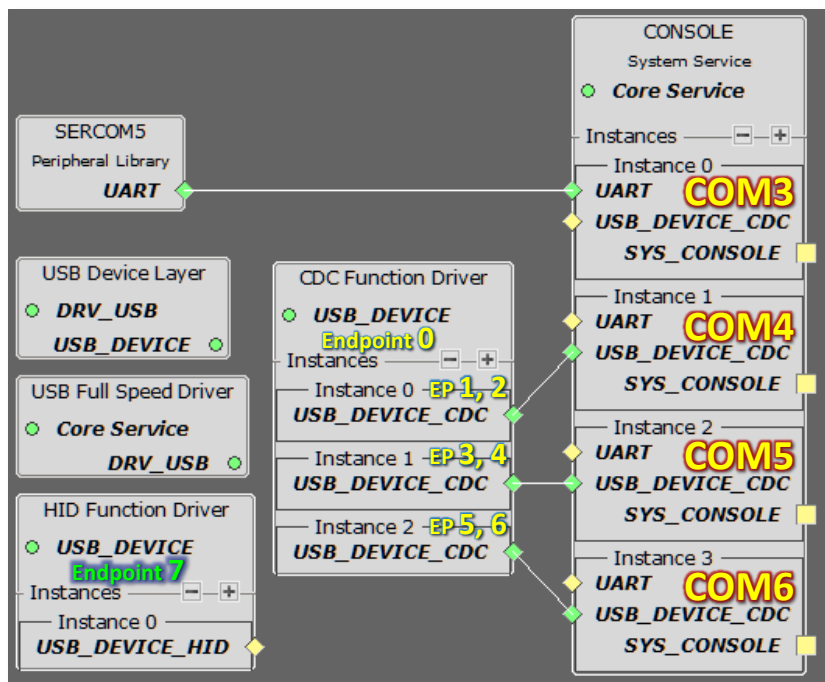
• Discuss

How many USB device CDC instance could support by SAMD21 ?

USB CDC Device is a collection of the following Interfaces and Endpoints

- Communication Interface (Device Management) on [Endpoint 0](#).
- Communication Interface (Notification) on an Interrupt(IN) [Endpoint](#).
- Data Interface on a Bulk(IN/OUT) [Endpoint](#).

SAMD21 USB device mode can support up to 8 endpoints (IN/OUT), the first CDC instance will consume three endpoints, and subsequent instances will consume two.



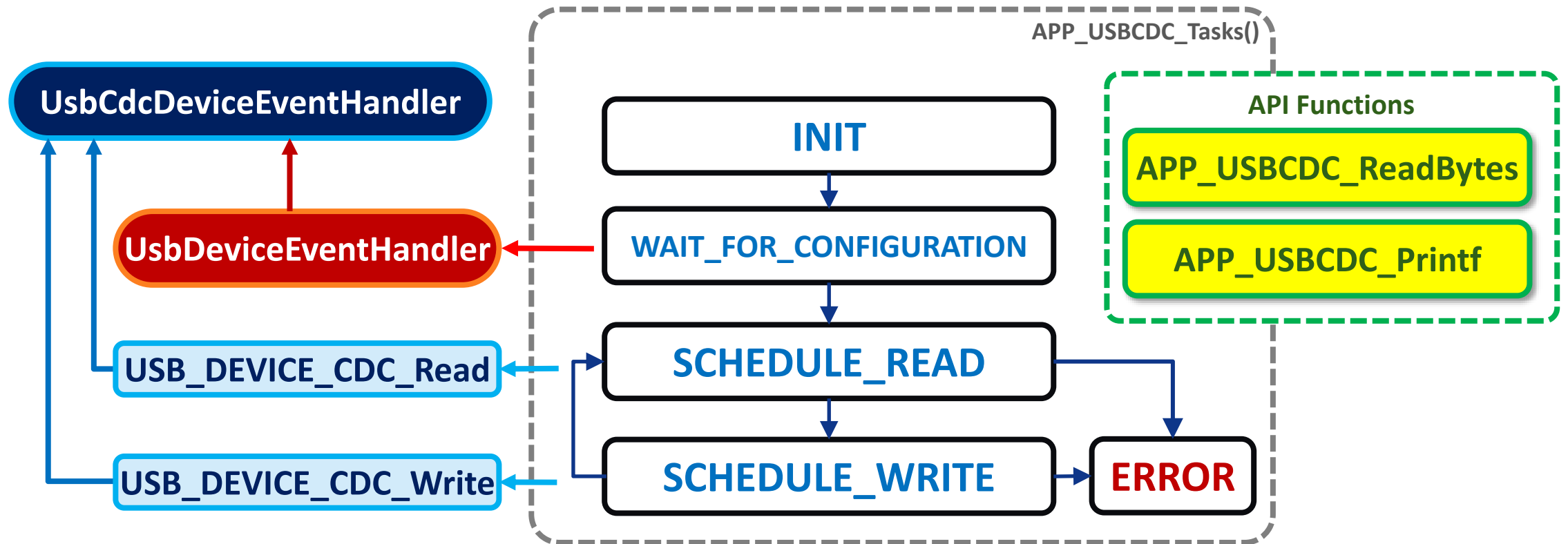
- ✓ 連接埠 (COM 和 LPT)
 - EDBG Virtual COM Port (COM4) **PLIB SERCOM5**
 - USB 序列裝置 (COM5) **CDC Instance 0**
 - USB 序列裝置 (COM6) **CDC Instance 1**
 - USB 序列裝置 (COM7) **CDC Instance 2**
- ✓ 滑鼠及其他指標裝置
 - HID-compliant mouse **HID Instance 0**

⚙️ Lab6 USB Device CDC with CONSOLE System Service

- Discuss

If it possible to implement a USB device CDC console without CONSOLE System service?

Yes, but you will need to implement the USB Device and USB Device CDC Event Handler by yourself through the API functions of USB Device Layer and USB Device CDC library.



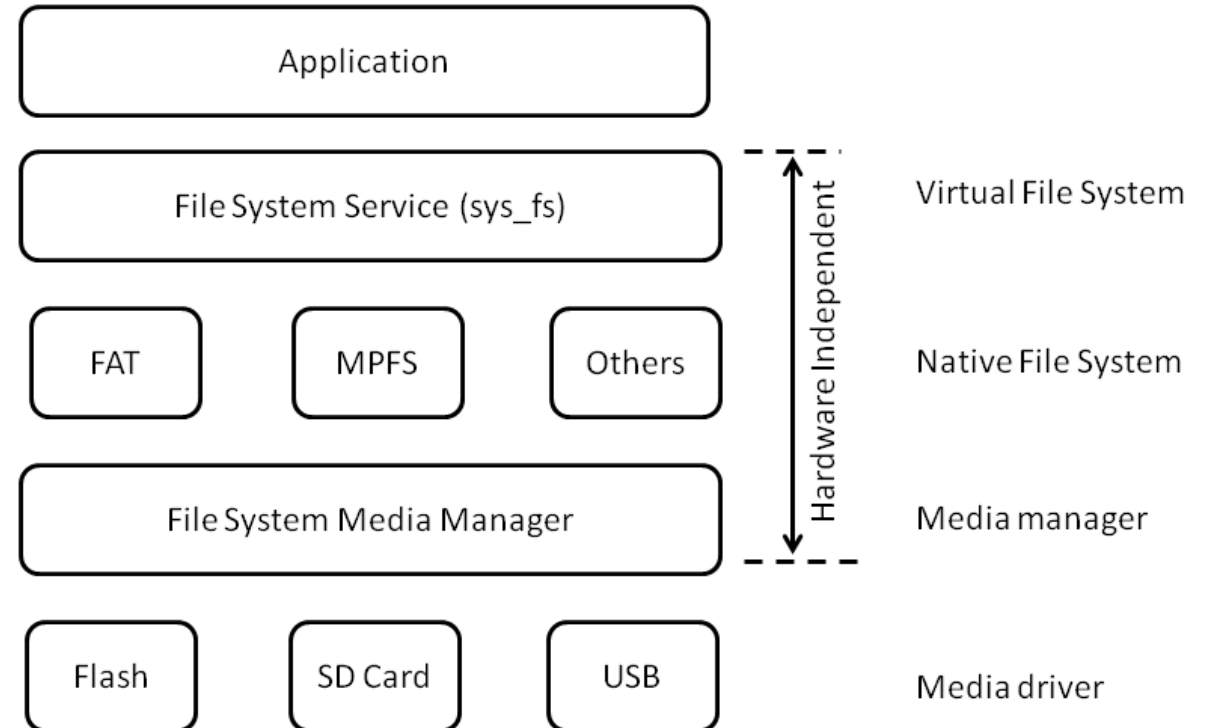
FILE System Service

Harmony System Service

FILE System Service

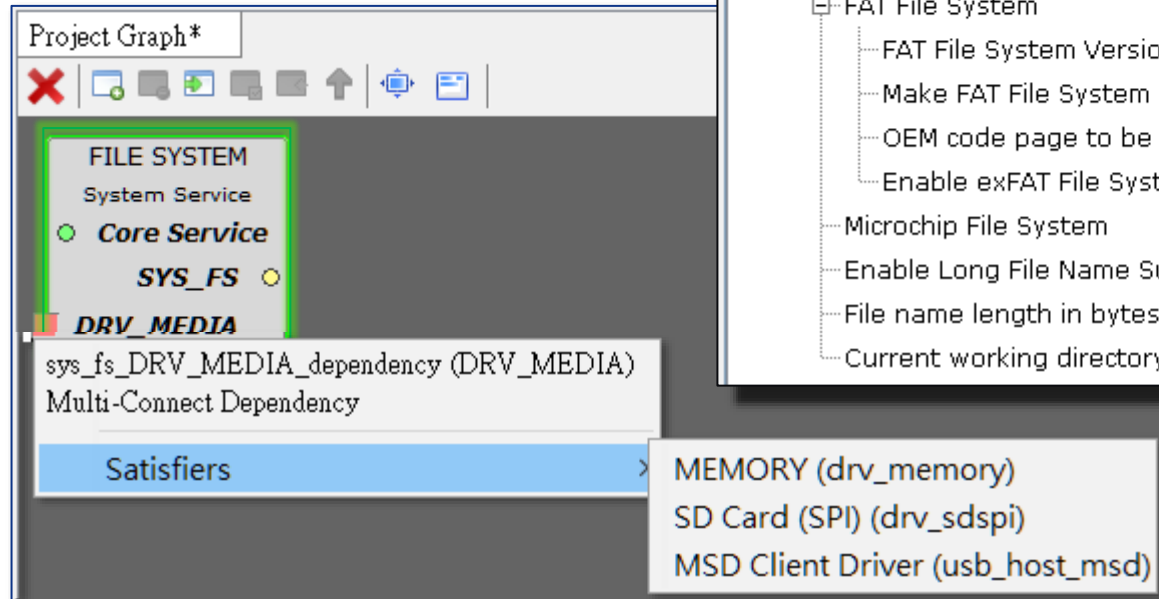
- Introduction

The File System Service provides an application programming interface (API) through which a utility or user program requests services of a file system. The File System service is really a framework designed to support multiple native file system and multiple media in the same application. Examples of native file systems are FAT12, FAT16, FAT32, and the Microchip File System (MPFS) among others. Each of these native file systems have a common set of APIs that can be used to access the files of that particular native file system. The File System Service abstracts the native file system calls and provides a common interface to the user/application_layer.



FILE System Service (DRV_MEDIA)

**FILE SYSTEM service
could support couple type
of Media Content**



Configuration Options* Help

FILE SYSTEM

File System settings

- Maximum Simultaneous File Access: 1
- Size Of Block: 512
- Size Of Media Manager Buffer: 2,048
- Use File System Auto Mount Feature?: ☐
- Total Number Of Media: 1
- Total Number Of Volumes: 1
- Media0: ☒
- File System Types: 1

FAT File System ☒

- FAT File System Version: v0.14a
- Make FAT File System Read-only: ☐
- OEM code page to be used: U.S.
- Enable exFAT File System Support: ☐

Microchip File System: ☐

Enable Long File Name Support: ☒

File name length in bytes: 255

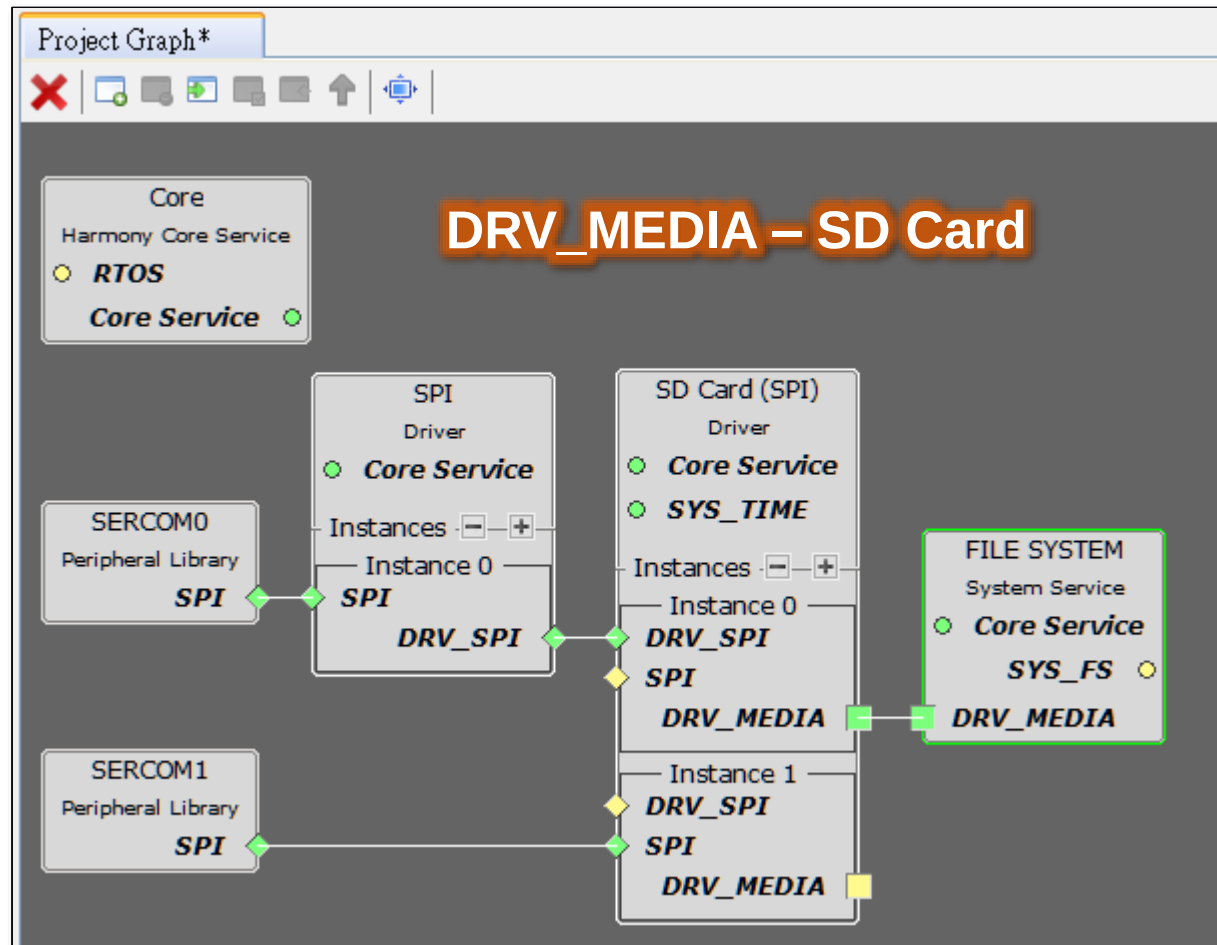
Current working directory scratch buffer length in bytes: 1,024



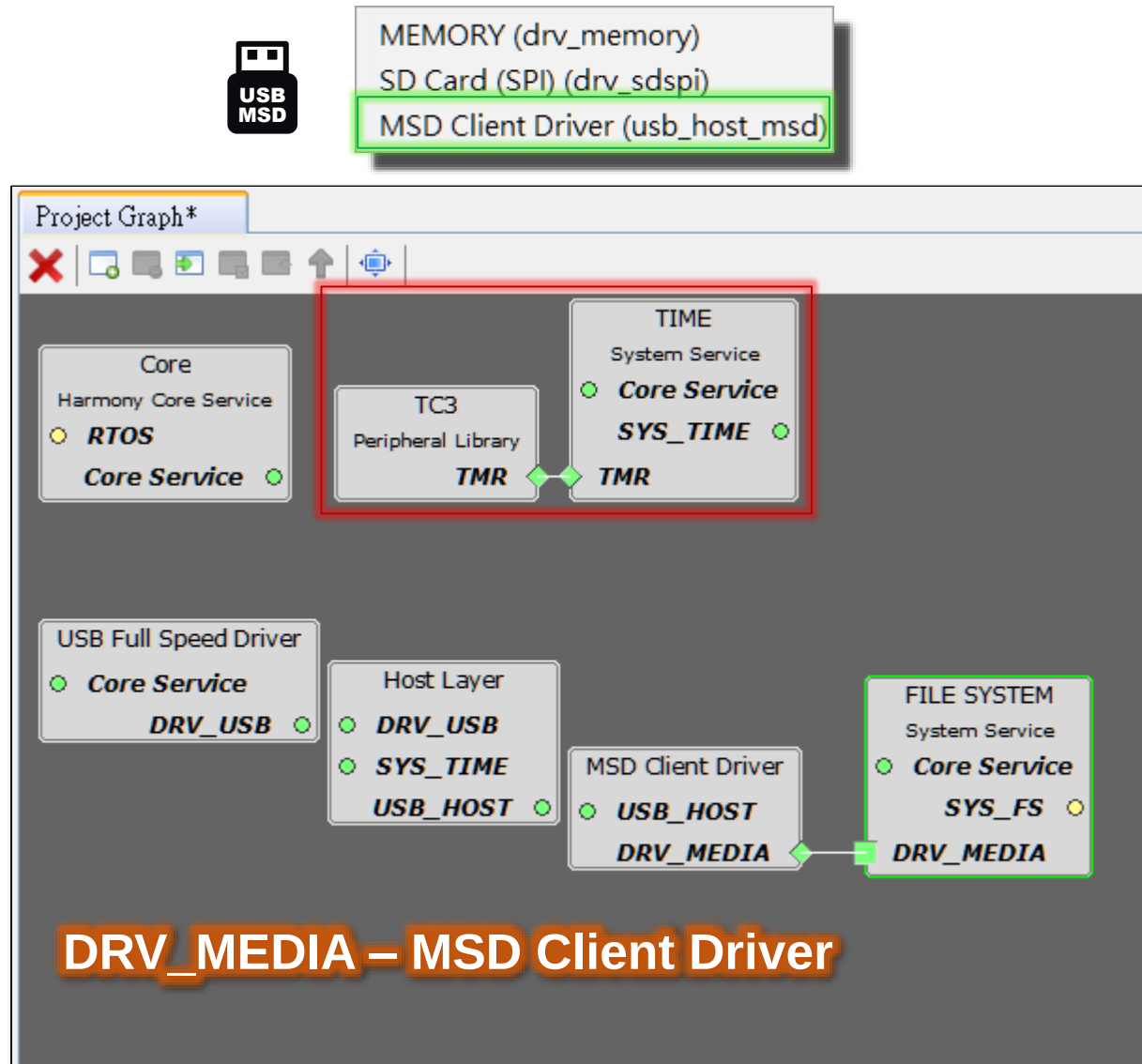
FILE System Service (SD Card)



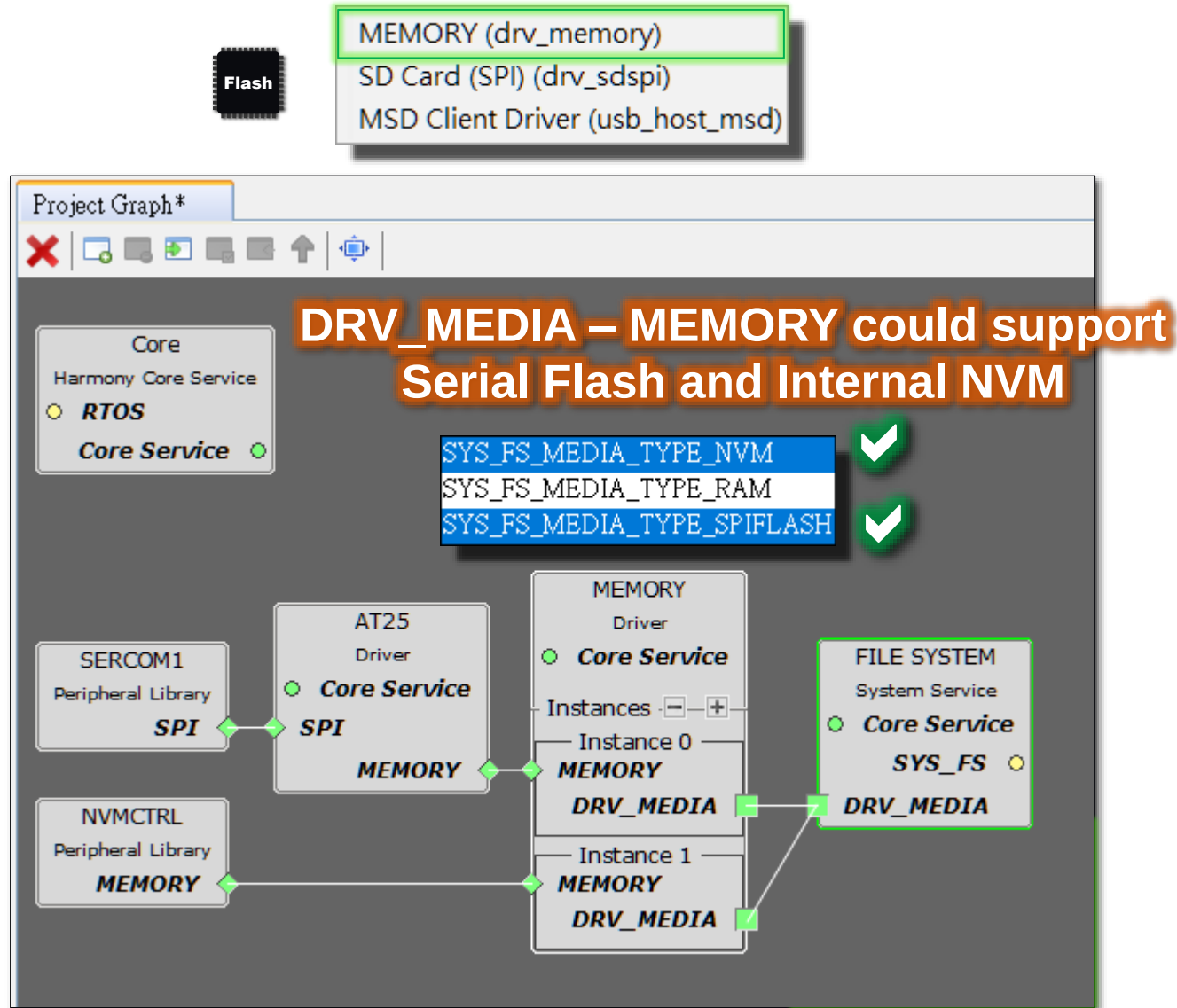
MEMORY (drv_memory)
SD Card (SPI) (drv_sdspi)
MSD Client Driver (usb_host_msdc)



FILE System Service (MSD Client Driver)



FILE System Service (MSD Client Driver)



FILE System Service

API functions of the File System Service library.

- File and Directory Operation Functions

SYS_FS_FileOpen	Opens a file.
SYS_FS_FileClose	Closes a file.
SYS_FS_FileCharacterPut	Writes a character to a file.
SYS_FS_FileEOF	Checks for end of file.
SYS_FS_FileError	Returns the file specific error.
SYS_FS_FileNameGet	Reads the file name.
SYS_FS_FilePrintf	Writes a formatted string into a file.
SYS_FS_FileRead	Read data from the file.
SYS_FS_FileSeek	Moves the file pointer by the requested offset.
SYS_FS_FileSize	Returns the size of the file in bytes.
SYS_FS_FileStat	Gets file status.
SYS_FS_FileStringGet	Reads a string from the file into a buffer.
SYS_FS_FileStringPut	Writes a string to a file.
SYS_FS_FileSync	Flushes the cached information when writing to a file.
SYS_FS_FileTell	Obtains the file pointer position.
SYS_FS_FileTestError	Checks for errors in the file.
SYS_FS_FileTruncate	Truncates a file
SYS_FS_FileWrite	Writes data to the file.
SYS_FS_DirectoryMake	Makes a directory.
SYS_FS_DirOpen	Open a directory
SYS_FS_DirClose	Closes an opened directory.

FILE System Service

API functions of the File System Service library. (cont.)

- File and Directory Operation Functions (cont.)

SYS_FS_DirRead	Reads the files and directories of the specified directory.
SYS_FS_DirRewind	Rewinds to the beginning of the directory.
SYS_FS_DirSearch	Searches for a file or directory.
SYS_FS_DirectoryChange	Changes to the directory specified.
SYS_FS_CurrentWorkingDirectoryGet	Gets the current working directory
SYS_FS_FileDirectoryModeSet	Sets the mode for the file or directory.
SYS_FS_FileDirectoryRemove	Removes a file or directory.
SYS_FS_FileDirectoryRenameMove	Renames or moves a file or directory.
SYS_FS_FileDirectoryTimeSet	Sets or changes the time for a file or directory.
SYS_FS_Error	Returns the last error.

- General Operation Functions

SYS_FS_Initialize	Initializes the file system abstraction layer (sys_fs layer).
SYS_FS_Tasks	Maintains the File System tasks and functionalities.
SYS_FS_Mount	Mounts the file system.
SYS_FS_Unmount	Unmounts the file system.
SYS_FS_CurrentDriveGet	Gets the current drive
SYS_FS_CurrentDriveSet	Sets the drive.
SYS_FS_DriveLabelGet	Gets the drive label.
SYS_FS_DriveLabelSet	Sets the drive label
SYS_FS_DriveFormat	Formats a drive.

FILE System Service

API functions of the File System Service library. (cont.)

- General Operation Functions (cont.)

<code>SYS_FS_DrivePartition</code>	Partitions a physical drive (media).
<code>SYS_FS_DriveSectorGet</code>	Obtains total number of sectors and number of free sectors for the specified drive.
<code>SYS_FS_EventHandlerSet</code>	Allows a client to identify an event handling function for the file system to call back when mount/unmount operation has completed. Functions active when <code>SYS_FS_AUTOMOUNT_ENABLE</code> is true.

- Media Manager Functions

<code>SYS_FS_MEDIA_MANAGER_Tasks</code>	Media manager task function.
<code>SYS_FS_MEDIA_MANAGER_TransferTask</code>	Media manager transfer task function.
<code>SYS_FS_MEDIA_MANAGER_Read</code>	Gets data from a specific media address.
<code>SYS_FS_MEDIA_MANAGER_SectorRead</code>	Reads a specified media sector.
<code>SYS_FS_MEDIA_MANAGER_SectorWrite</code>	Writes a sector to the specified media.
<code>SYS_FS_MEDIA_MANAGER_Register</code>	Function to register media drivers with the media manager.
<code>SYS_FS_MEDIA_MANAGER_RegisterTransferHandler</code>	Register the event handler for data transfer events.
<code>SYS_FS_MEDIA_MANAGER_DeRegister</code>	Function called by a media to deregister itself.
<code>SYS_FS_MEDIA_MANAGER_AddressGet</code>	Gets the starting media address based on a disk number.
<code>SYS_FS_MEDIA_MANAGER_MediaStatusGet</code>	Gets the media status.
<code>SYS_FS_MEDIA_MANAGER_VolumePropertyGet</code>	Gets the volume property.
<code>SYS_FS_MEDIA_MANAGER_CommandStatusGet</code>	Gets the command status.
<code>SYS_FS_MEDIA_MANAGER_GetMediaGeometry</code>	Gets the media geometry information.
<code>SYS_FS_MEDIA_MANAGER_EventHandlerSet</code>	Register the event handler for Mount/Un-Mount events.
<code>_SYS_FS_MountVolume</code>	Mount the volume when AUTOMOUNT feature is enabled.

FILE System Service

API functions of the File System Service library. (cont.)

- File System Data Types and Constants

SYS_FS_ERROR	Lists the various error cases.
SYS_FS_FILE_SEEK_CONTROL	Lists the various modes of file seek.
SYS_FS_FSTAT	File System status
SYS_FS_FUNCTIONS	SYS FS Function signature structure for native file systems.
SYS_FS_REGISTRATION_TABLE	The sys_fs layer has to be initialized by passing this structure with suitably initialized members.
SYS_FS_RESULT	Lists the various results of a file operation.
SYS_FS_FILE_OPEN_ATTRIBUTES	Lists the various attributes (modes) in which a file can be opened.
FAT_FS_MAX_LFN	Maximum length of the Long File Name.
FAT_FS_MAX_SS	Lists the definitions for FAT file system sector size.
FAT_FS_USE_LFN	Lists the definitions for FAT file system LFN selection.
SYS_FS_FILE_SYSTEM_TYPE	Enumerated data type identifying native file systems supported.
SYS_FS_HANDLE	This type defines the file handle.
SYS_FS_HANDLE_INVALID	Invalid file handle
SYS_FS_FILE_DIR_ATTR	Enumerated data type identifying the various attributes for file/directory.
SYS_FS_TIME	The structure to specify the time for a file or directory.
SYS_FS_FORMAT	Specifies the partitioning rule.
SYS_FS_EVENT	Identifies the possible file system events.
SYS_FS_EVENT_HANDLER	Pointer to the File system Handler function.
SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE_INVALID	Defines the invalid media block command handle.
SYS_FS_MEDIA_HANDLE_INVALID	Defines the invalid media handle.

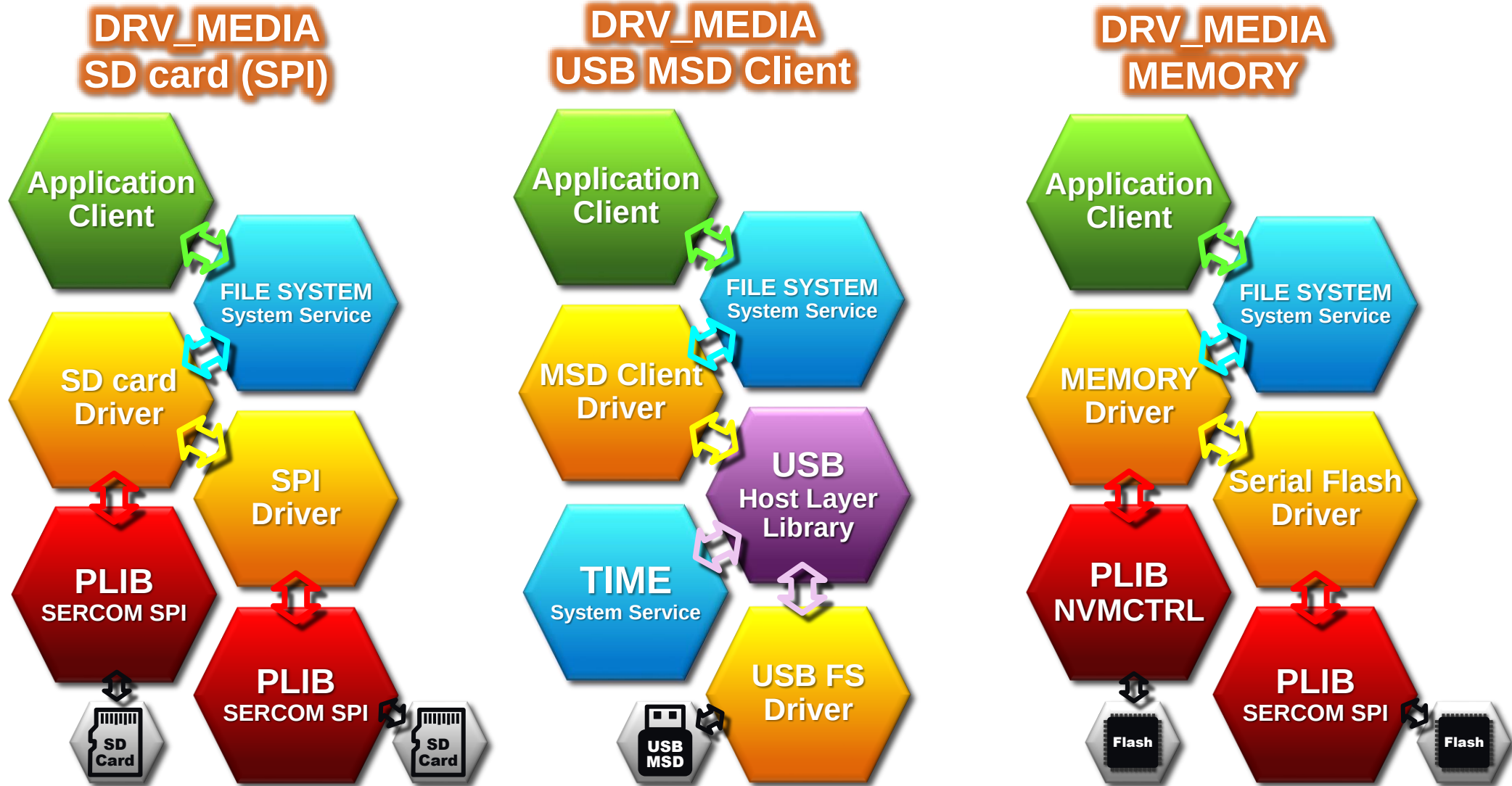
FILE System Service

API functions of the File System Service library. (cont.)

- File System Data Types and Constants (cont.)

<code>_SYS_FS_VOLUME_PROPERTY</code>	Structure to obtain the property of volume
<code>SYS_FS_MEDIA_BLOCK_EVENT</code>	Identifies the possible events that can result from a request.
<code>SYS_FS_MEDIA_COMMAND_STATUS</code>	The enumeration for status of buffer
<code>SYS_FS_MEDIA_FUNCTIONS</code>	Structure of function pointers for media driver
<code>SYS_FS_MEDIA_MOUNT_DATA</code>	Structure to obtain the device and mount name of media
<code>SYS_FS_MEDIA_PROPERTY</code>	Contains information of property of a media.
<code>SYS_FS_MEDIA_STATE</code>	The enumeration for state of media.
<code>SYS_FS_MEDIA_STATUS</code>	The state of media.
<code>SYS_FS_MEDIA_TYPE</code>	The enumeration for type of media.
<code>SYS_FS_MEDIA_BLOCK_COMMAND_HANDLE</code>	Handle identifying block commands of the media.
<code>SYS_FS_MEDIA_EVENT_HANDLER</code>	Pointer to the Media Event Handler function.
<code>SYS_FS_MEDIA_GEOMETRY</code>	Contains all the geometrical information of a media device.
<code>SYS_FS_MEDIA_HANDLE</code>	Handle identifying the media registered with the media manager.
<code>SYS_FS_MEDIA_REGION_GEOMETRY</code>	Contains information of a sys media region.
<code>SYS_FS_VOLUME_PROPERTY</code>	Structure to obtain the property of volume

FILE System Service



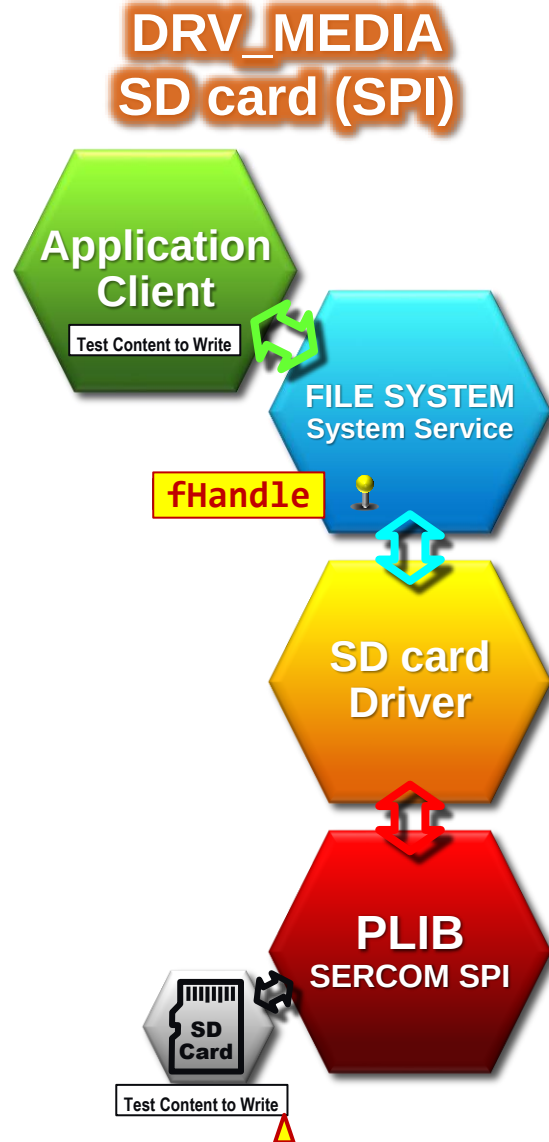
FILE System Service

- The standard names for volumes (devName) used in the MPLAB Harmony file system is as follows:

NVM Internal Flash :	nvm	[media number]	[volume number]
SD card :	mmcblk	[media number]	[volume number]
USB Mass Storage :	sd	[media number]	[volume number]
External SPI Flash :	mtd	[media number]	[volume number]
Internal SRAM :	ram	[media number]	[volume number]

- [media number] is **a, b, c...** depends on the number of connected media.
- [volume number] is **1, 2, 3...** depends on the number partitions of media.
- If a SD card with four partitions is attached to the system, the devNames will be **mmcblk**a**1, mmcblk**a**2, mmcblk**a**3, mmcblk**a**4**
- If a NVM media with one partition attached, the devName will be **nvma**1****.
- If 2nd SD card with one partition attached, the devName will be **mmcblk**b**1**.
- Finally, there will be six devNames which are available for the application to be mounted and used for the file system. (**mmcblk**a**1, mmcblk**a**2, mmcblk**a**3, mmcblk**a**4, nvma**1**, mmcblk**b**1**)
- The media devName must be preceded by the string **"/dev/"**, **the devName should use Harmony naming rule.**
/dev/mmcblk**a**1****
- The drive mount name must be preceded by the string **"/mnt/"**, **the mount name is customizable.**
/mnt/myDrive1****

FILE System Service (SD Card)



```
#define SDCARD_DEV_NAME    "/dev/mmcblk1"
#define SDCARD_MOUNT_NAME "/mnt/myDrive1"
#define FILE_CONTENT      "Test Content to Write"
SYS_FS_HANDLE fHandle = SYS_FS_HANDLE_INVALID;
char readBuffer[30];

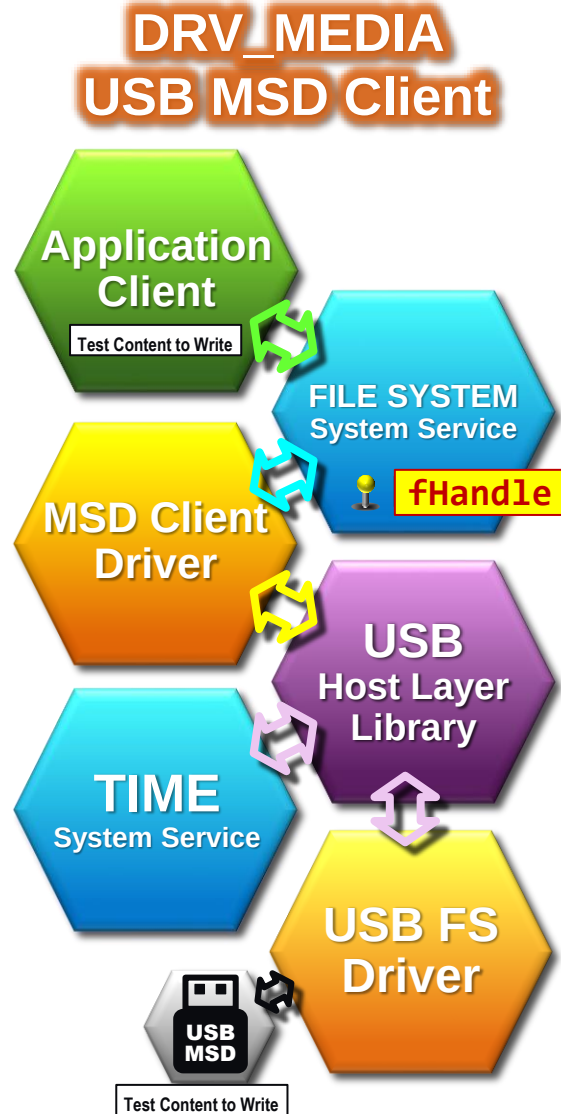
void CommandSDCard(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_FS_Mount(SDCARD_DEV_NAME, SDCARD_MOUNT_NAME, FAT, 0, NULL);
    SYS_FS_CurrentDriveSet(SDCARD_MOUNT_NAME);
    fHandle = SYS_FS_FileOpen(argv[1], SYS_FS_FILE_OPEN_WRITE_PLUS);
    SYS_CMD_PRINT("- Create File : %s\r\n", argv[1] );
    SYS_FS_FileWrite(fHandle, FILE_CONTENT, sizeof(FILE_CONTENT));
    SYS_CMD_PRINT("- Write [%s] to File\r\n", FILE_CONTENT );
    SYS_FS_FileSeek(fHandle, 0, SYS_FS_SEEK_SET);
    SYS_FS_FileRead(fHandle, (void *)readBuffer, 30);
    SYS_CMD_PRINT("- Read content : [%s]\r\n", readBuffer );
    SYS_FS_FileClose(fHandle);
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[] =
{
    { "sdcard", CommandSDCard, ": SD Card Test" },
};
...
```

COM4 - Tera Term VT

```
>sdcard test.txt
- Create File : test.txt
- Write [Test Content to Write] to File
- Read content : [Test Content to Write]
>
```

FILE System Service (USB Mass Storage)



```
#define USBMSD_DEV_NAME    "/dev/sda1"
#define USBMSD_MOUNT_NAME "/mnt/myDrive2"
#define FILE_CONTENT      "Test Content to Write"
SYS_FS_HANDLE fHandle = SYS_FS_HANDLE_INVALID;
char readBuffer[30];

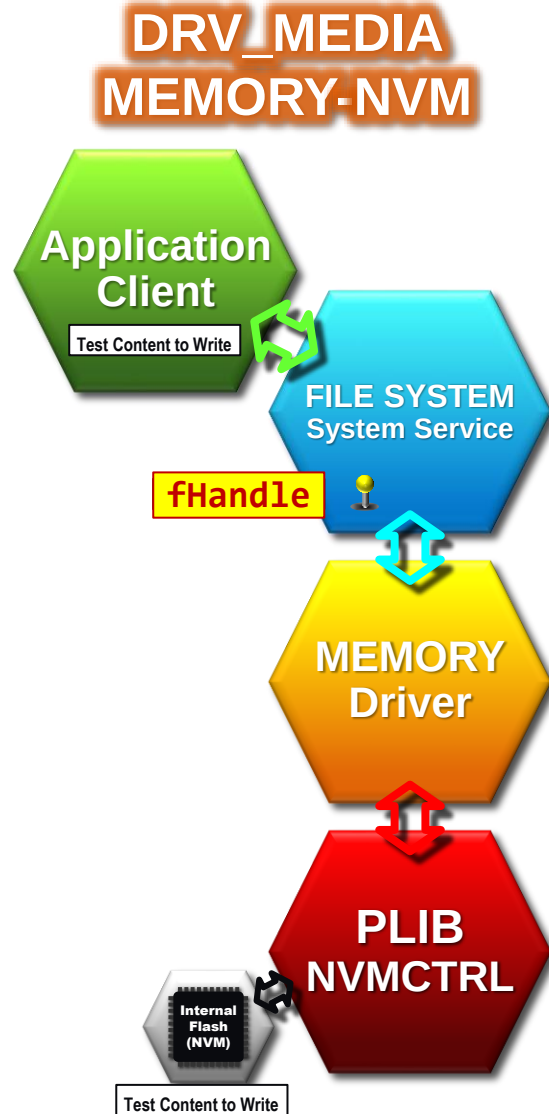
void CommandUsbMSD(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    USB_HOST_BusEnable(0);

    SYS_FS_Mount(USBMSD_DEV_NAME, USBMSD_MOUNT_NAME, FAT, 0, NULL);
    SYS_FS_CurrentDriveSet(USBMSD_MOUNT_NAME);

    fHandle = SYS_FS_FileOpen(argv[1], SYS_FS_FILE_OPEN_WRITE_PLUS);
    SYS_CMD_PRINT("- Create File : %s\r\n", argv[1] );
    SYS_FS_FileWrite(fHandle, FILE_CONTENT, sizeof(FILE_CONTENT));
    SYS_CMD_PRINT("- Write [%s] to File\r\n", FILE_CONTENT );
    SYS_FS_FileSeek(fHandle, 0, SYS_FS_SEEK_SET);
    SYS_FS_FileRead(fHandle, (void *)readBuffer, 30);
    SYS_CMD_PRINT("- Read content : [%s]\r\n", readBuffer );
    SYS_FS_FileClose(fHandle);
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[] =
{
    { "usbmsd", CommandUsbMSD, ": USB Mass Storage Test" },
};
...
```

FILE System Service (MEMORY-NVM)



```
#define MEMNVM_DEV_NAME    "/dev/nvma1"
#define MEMNVM_MOUNT_NAME  "/mnt/myDrive3"
#define FILE_CONTENT       "Test Content to Write"
SYS_FS_HANDLE fHandle = SYS_FS_HANDLE_INVALID;
char readBuffer[30];

void CommandMemNVM(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_FS_FORMAT_PARAM opt = {SYS_FS_FORMAT_FAT, 0, 1, 0, 1 };
    uint8_t CACHE_ALIGN work[512];

    SYS_FS_Mount(MEMNVM_DEV_NAME, MEMNVM_MOUNT_NAME, FAT, 0, NULL);
    SYS_FS_CurrentDriveSet(MEMNVM_MOUNT_NAME);

    SYS_FS_DriveFormat(MEMNVM_MOUNT_NAME, &opt, (void *)work, 512);

    fHandle = SYS_FS_FileOpen(argv[1], SYS_FS_FILE_OPEN_WRITE_PLUS);
    SYS_CMD_PRINT("- Create File : %s\r\n", argv[1] );
    SYS_FS_FileWrite(fHandle, FILE_CONTENT, sizeof(FILE_CONTENT));
    SYS_CMD_PRINT("- Write [%s] to File\r\n", FILE_CONTENT );
    SYS_FS_FileSeek(fHandle, 0, SYS_FS_SEEK_SET);
    SYS_FS_FileRead(fHandle, (void *)readBuffer, 30);
    SYS_CMD_PRINT("- Read content : [%s]\r\n", readBuffer );
    SYS_FS_FileClose(fHandle);
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=
{
    { "memnvm", CommandMemNVM, ": Internal Flash (NVM) Test" },
};
```

Lab 7

SD Card with file system

FILE System Service

🔧 Lab7 SD Card with file system

- Continue this Lab from Lab5.
- Try add **FILE system service** and select **SPI SD Card** as media.
- The **SERCOM4-SPI** as the hardware interface of SD card socket.
- Build custom command to access SD card as below :

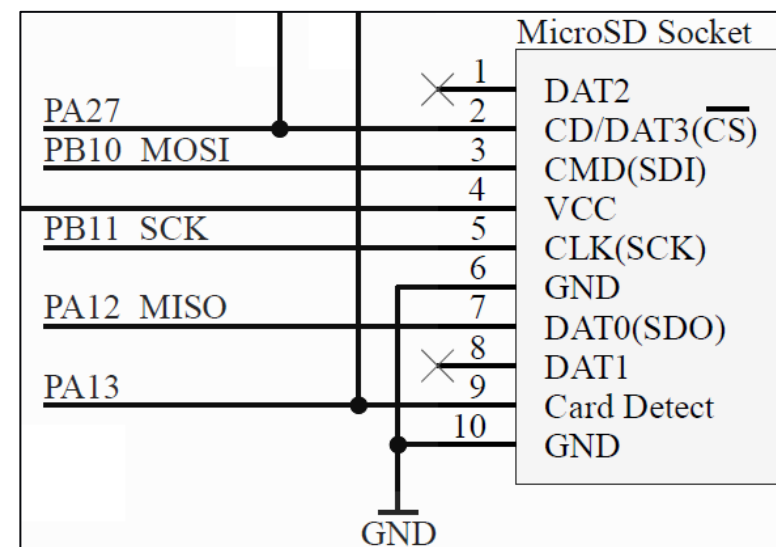
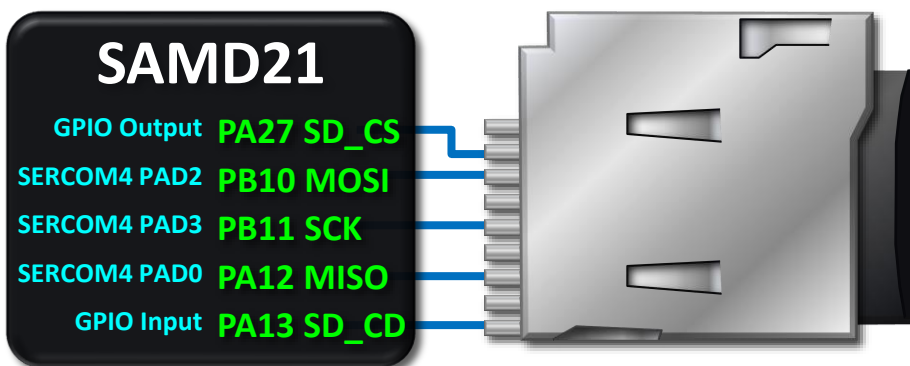
mount : Mount the SD card.

open \$: Open file \$, if not exist then create a new file.

write \$: Write string \$ to opened file.

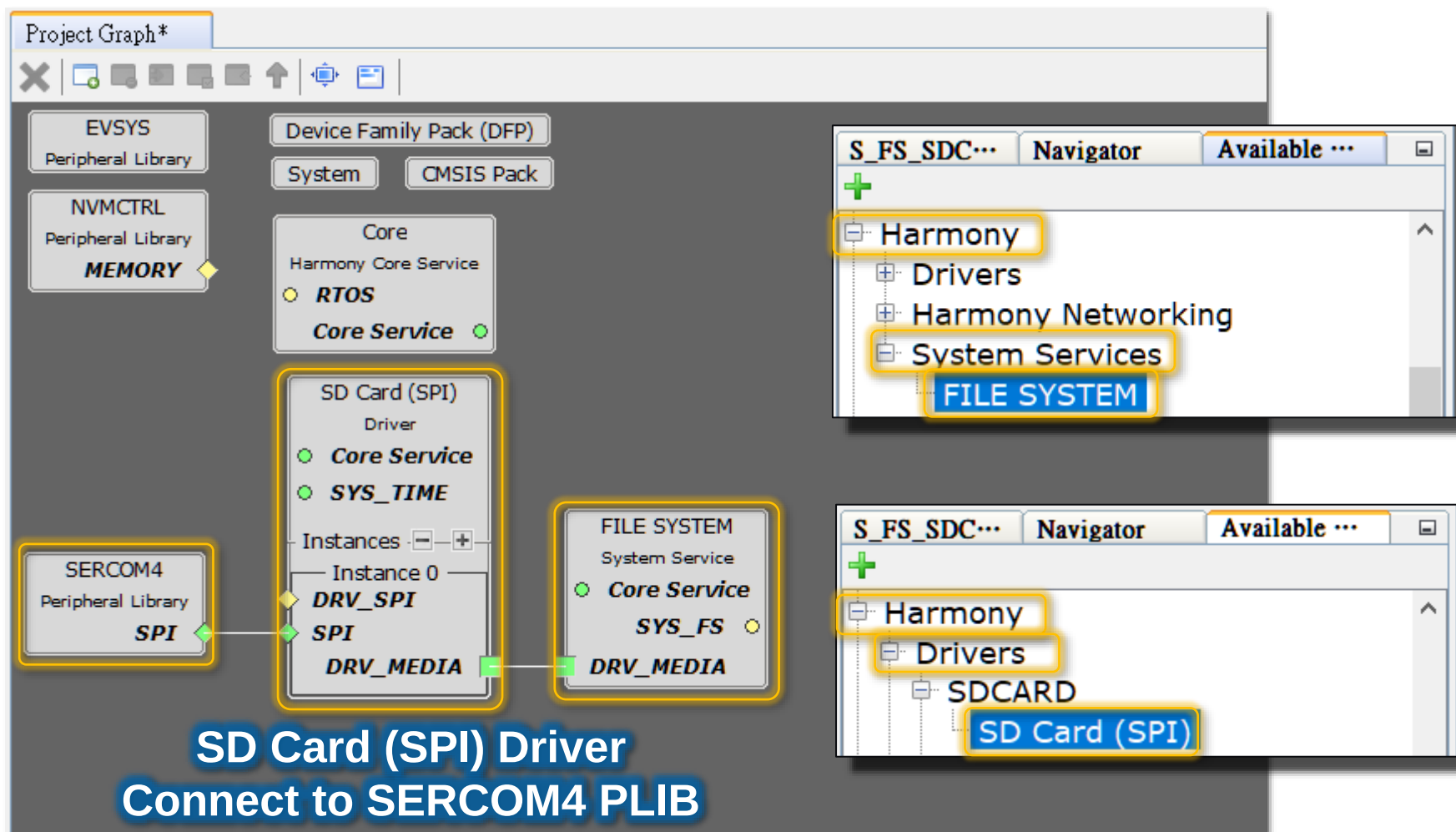
read # : Seek to start and Read # character from file.

close : Close opened file.



⚙️ Lab7 SD Card with file system

- Harmony Configuration (Add **FILE system service** and **SD Card**)
- Select PLIB **SERCOM4** for SD Card (SPI) Driver.



⚙️ Lab7 SD Card with file system

- Harmony Configuration (FILE system service configuration)

The image shows the Harmony IDE interface. On the left is the 'Project Graph*' window, which displays a block diagram of the system components. It includes the 'EVSYS' and 'NVMCTRL' peripheral libraries, the 'Device Family Pack (DFP)' containing 'System' and 'CMSIS Pack', the 'Core' with 'Harmony Core Service', 'RTOS', and 'Core Service', the 'SD Card (SPI)' driver with 'Core Service', 'SYS_TIME', and 'Instances' (Instance 0 with 'DRV_SPI', 'SPI', and 'DRV_MEDIA'), and the 'SERCOM4' peripheral library with 'SPI'. A green box highlights the 'FILE SYSTEM' system service, which is connected to the 'DRV_MEDIA' component of the SD Card driver.

On the right is the 'Configuration Options' dialog box, specifically the 'FILE SYSTEM' configuration page. The 'File System settings' section includes options for 'Maximum Simultaneous File Access' (1), 'Size Of Block' (512), 'Size Of Media Manager Buffer' (2,048), 'Use File System Auto Mount Feature?' (disabled, indicated by a blue box around the unchecked checkbox), 'Total Number Of Media' (1), 'Total Number Of Volumes' (1), 'Media0' (checked), 'File System Types' (1), and 'FAT File System' (checked). The 'FAT File System' section includes 'FAT File System Version' (v0.14), 'Make FAT File System Read-only' (unchecked), 'OEM code page to be used' (U.S.), 'Enable exFAT File System Support' (unchecked), 'Microchip File System' (unchecked), 'Enable Long File Name Support' (checked), 'File name length in bytes' (255), and 'Current working directory scratch buffer length in bytes' (1,024).

**Keep the default settings
The Auto Mount Feature is disabled**

✖ Lab7 SD Card with file system

- Harmony Configuration (PLIB SERCOM0 config)

Configuration Options*

SERCOM4

Select SERCOM Operation Mode: SPI Master

Enable Interrupts ? ☒

Enable operation in Standby mode ☐

SPI Data Out Pad: DO on PAD[2], SCK on PAD[3] and SS on PAD[1]

SPI Data In Pad Selection: SERCOM PAD[0]

SPI Data Order: MSB is transferred first

SPI Speed in Hz: 1,000,000

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

Enable SPI Master Hardware Slave Select ☐

SPI Receiver Enable ☒

SPI Transfer Mode 0 is Selected

DO on PAD[2] (MOSI)
SCK on PAD[3] (SCK)
DI on PAD[0] (MISO)

SAMD21

GPIO Output PA27 SD_CS

SERCOM4 PAD2 PB10 MOSI

SERCOM4 PAD3 PB11 SCK

SERCOM4 PAD0 PA12 MISO

GPIO Input PA13 SD_CD

Driver

Core Service

SYS_TIME

Instances

Instance 0

DRV_SPI

SPI

DRV_MEDIA

FILE SYSTEM

System Service

Core Service

SYS_FS

DRV_MEDIA

SERCOM4 Peripheral Library

SPI

⚙️ Lab7 SD Card with file system

• Harmony Configuration (SD Card configuration)

SAMD21

GPIO Output **PA27 SD_CS**

SERCOM4 PAD2 **PB10 MOSI**

SERCOM4 PAD3 **PB11 SCK**

SERCOM4 PAD0 **PA12 MISO**

GPIO Input **PA13 SD_CD**

The diagram illustrates the hardware and software configuration for an SD card on a SAMD21 microcontroller. It shows the connection between the SERCOM4 peripheral, the SD Card (SPI) driver, and the File System. The SD Card (SPI) driver is configured with Instance 0, DRV_SPI, SPI, and DRV_MEDIA. The File System is configured with SYS_FS and DRV_MEDIA.

Configuration Options*

SD Card (SPI)

- PLIB Used: SERCOM0
- Number of Clients: 1
- SD Card Speed (Hz): 5,000,000
- Transfer Queue Size: 4
- Card Detection Method: Use Polling
- Polling Interval (ms): 1,000
- File system for SDSPI Driver Enabled: ☒
- Chip Select Pin: **PA27** (highlighted with a red box)
- !!! Configure the Chip Select pin as GPIO output under Pin Settings.!!!
- Use Write Protect Pin (Active High)? ☐
- Use DMA for Transmit and Receive ? ☒ (highlighted with a red box)
- DMA Channel For Transmit: 0
- DMA Channel For Receive: 1
- !!! Note: For each instance of SDSPI, connect either SPI PLIB or SPI Driver !!!

Set Chip Select pin to PA27 and Enable DMA

✖ Lab7 SD Card with file system

- Harmony Configuration (Setup SERCOM0 pads and GPIO pins)
- SERCOM4 on **PB10, PB11, PA12** ; GPIO on **PA13(In)**, **PA27(Out)**.

Pin Number	Pin ID	Custom Name	Function	Mode
19	PB10		SERCOM4_PAD2	Digital
20	PB11		SERCOM4_PAD3	Digital
21	PA12		SERCOM4_PAD0	Digital
22	PA13	GPIO_PA13	GPIO	Digital, In
39	PA27	GPIO_PA27	GPIO	Digital, Out

SD Card (SPI) Configuration:

- Number of Clients: 1
- SD Card Speed (Hz): 5,000,000
- Transfer Queue Size: 4
- Card Detection Method: Use Polling
- Polling Interval (ms): 1,000
- File system for SDSPI Driver Enabled: ☒
- Chip Select Pin: PA27

SAMD21 Connections:

- GPIO Output PA27 SD_CS
- SERCOM4 PAD2 PB10 MOSI
- SERCOM4 PAD3 PB11 SCK
- SERCOM4 PAD0 PA12 MISO
- GPIO Input PA13 SD_CD

✖ Lab7 SD Card with file system

- Modify your main.c as below

```
void CommandErrLv(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }

SYS_FS_HANDLE fHandle = SYS_FS_HANDLE_INVALID;
SYS_FS_RESULT fResult;
char fMedia[20] = "/dev/mmcblk1";
char fDrive[20] = "/mnt/myDrive1";
void CommandMount(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    fResult = SYS_FS_Mount(fMedia, fDrive, FAT, 0, NULL);
    if(fResult == SYS_FS_RES_SUCCESS)
    {
        fResult = SYS_FS_CurrentDriveSet(fDrive);
        if(fResult == SYS_FS_RES_SUCCESS)
        {
            SYS_CMD_PRINT("Mount %s success!\r\n", fMedia);
            SYS_CMD_PRINT("Current Drive is %s\r\n", fDrive);
            return;
        }
    }
    SYS_CMD_PRINT("Mount %s failure!\r\n", fMedia);
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=
{
    { "errlv", CommandErrLv, ": errlv #" },
    { "mount", CommandMount, ": Mount Media" },
};

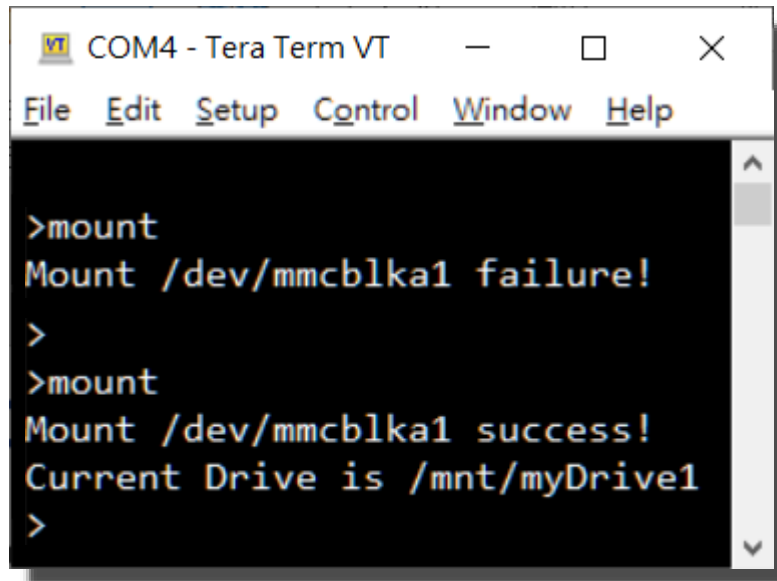
int main( void )
{
    SYS_CMD_ADDGRP(MyCmdTbl, sizeof(MyCmdTbl)/sizeof(SYS_CMD_DESCRIPTOR),
    "MYCMD", ": My Commands");
```

**Auto calculate
Command Table counts.**

Modify this line

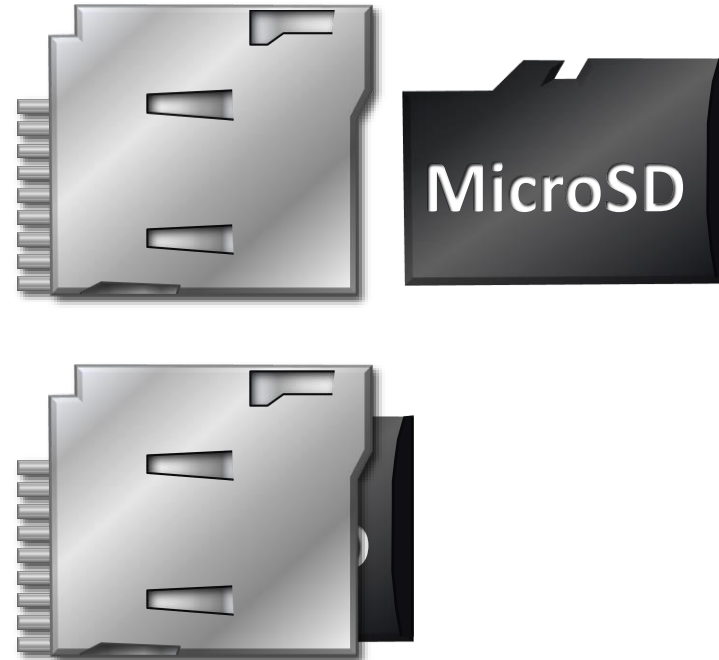
⚙️ Lab7 SD Card with file system

- Compiler and programming your project to EVB.
- **Remove SD card firstly.**
- Type-in '**mount**' to execute SD card mount operation.
- **Put in SD card.**
- Type-in '**mount**' again to see the result.



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

>mount
Mount /dev/mmcblk1 failure!
>
>mount
Mount /dev/mmcblk1 success!
Current Drive is /mnt/myDrive1
>
```



⚙️ Lab7 SD Card with file system

- Modify your main.c as below

```
... void CommandMount(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }

void CommandOpen(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    fHandle = SYS_FS_FileOpen(argv[1], SYS_FS_FILE_OPEN_APPEND_PLUS);
    if(fHandle != SYS_FS_HANDLE_INVALID)
        SYS_CMD_PRINT("File [%s] opened!\r\n", argv[1] );
    else
        SYS_CMD_MESSAGE("Open failure!\r\n");
}

void CommandWrite(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    int fWriteSize = 0;

    fWriteSize = SYS_FS_FileWrite(fHandle, argv[1], strlen(argv[1]));
    if(fWriteSize != -1)
        SYS_CMD_PRINT("Write [%s](%d) to File!\r\n", argv[1], fWriteSize );
    else
        SYS_CMD_MESSAGE("Write failure!\r\n");
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=
{
    { "mount", CommandMount, ": Mount Media" },
    { "open", CommandOpen, ": open $" },
    { "write", CommandWrite, ": write $" },
};

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

✖ Lab7 SD Card with file system

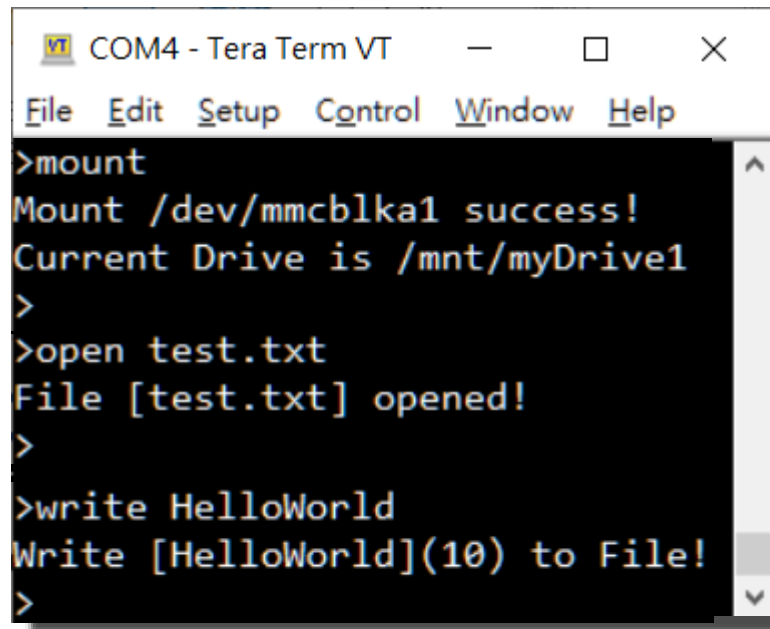
- The attribute **SYS_FS_FILE_OPEN_APPEND_PLUS** of file open operation could open target file or to create a new file if target file not exist and allow append bytes from the end of file.

```
void CommandOpen(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    fHandle = SYS_FS_FileOpen(argv[1], SYS_FS_FILE_OPEN_APPEND_PLUS);
    ...
}
```

SYS_FS_FILE_OPEN_	File exist (Open) or Not exist (Create)	Read from beginning	Read from the end	Write from Beginning	Write from the end
READ	Open / -----	✓			
WRITE	Open / Create			✓	
APPEND	Open / Create				✓
READ_PLUS	Open / -----	✓		✓	
WRITE_PLUS	Open / Create	✓		✓	
APPEND_PLUS	Open / Create		✓		✓

🔧 Lab7 SD Card with file system

- Compiler and programming your project to EVB.
- Put in SD Card and type-in '**mount**' to mount the SD card.
- Type-in '**open test.txt**' to execute file open operation,
if '**test.txt**' not exist then create a new file for write and read.
if '**test.txt**' already exist then to open it for write and read.
- Type-in '**write HelloWorld**' to execute write operation.
- Remove SD Card and check '**test.txt**' in the SD card on your PC.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
>mount
Mount /dev/mmcblk1 success!
Current Drive is /mnt/myDrive1
>
>open test.txt
File [test.txt] opened!
>
>write HelloWorld
Write [HelloWorld](10) to File!
>
```

Why not thing in test.txt ?



✂ Lab7 SD Card with file system

- Modify your main.c as below

```
void CommandMount(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }
void CommandOpen(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }
void CommandWrite(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }
void CommandRead(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    int fReadSize = 0;
    char fReadBuf[50] = "";
    SYS_FS_FileSeek(fHandle, 0, SYS_FS_SEEK_SET);
    fReadSize = SYS_FS_FileRead(fHandle, (void *)fReadBuf, atoi(argv[1]));
    if(fReadSize != -1)
        SYS_CMD_PRINT("Read [%s](%d) from File!\r\n", fReadBuf, fReadSize );
    else
        SYS_CMD_MESSAGE("Read failure!\r\n");
}

void CommandClose(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_FS_FileClose(fHandle);
    SYS_CMD_MESSAGE("File Closed!\r\n");
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=
{
    { "mount", CommandMount, ": Mount SD Card" },
    { "open", CommandOpen, ": open $" },
    { "write", CommandWrite, ": write $" },
    { "read", CommandRead, ": read #" },
    { "close", CommandClose, ": Close file" },
};

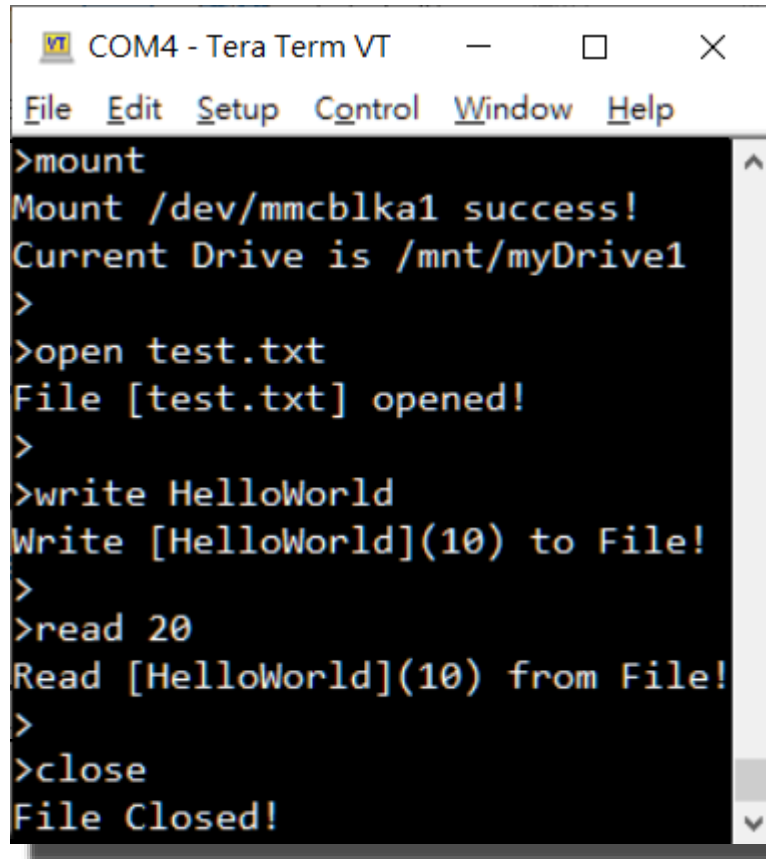
int main( void ) { ... }
```

HelloWorld



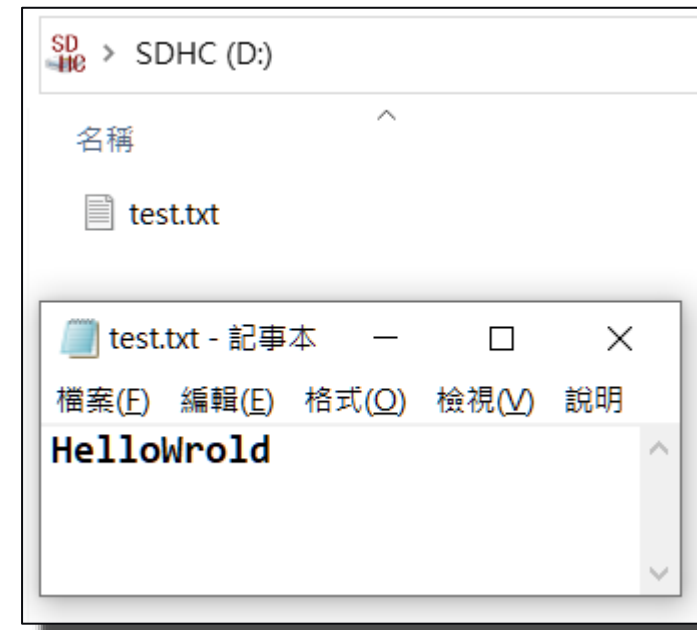
⚙️ Lab7 SD Card with file system

- Compiler and programming your project to EVB.
- Run '**mount**', '**open test.txt**' and '**write HelloWorld**' as previous.
- Type-in '**read 20**' command to **read 20 bytes** from file.
- Type-in '**close**' command to close file then check SD Card again.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
>mount
Mount /dev/mmcblk1 success!
Current Drive is /mnt/myDrive1
>
>open test.txt
File [test.txt] opened!
>
>write HelloWorld
Write [HelloWorld](10) to File!
>
>read 20
Read [HelloWorld](10) from File!
>
>close
File Closed!
```

HelloWorld in test.txt



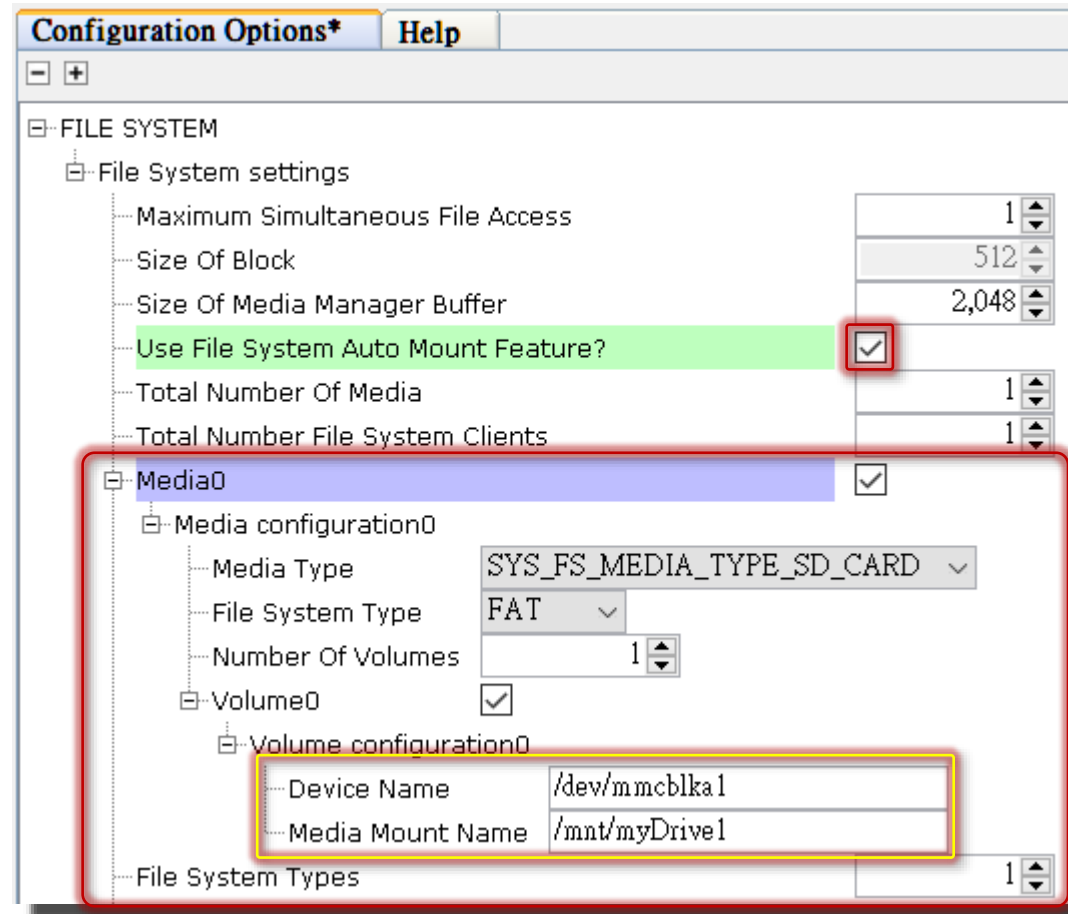
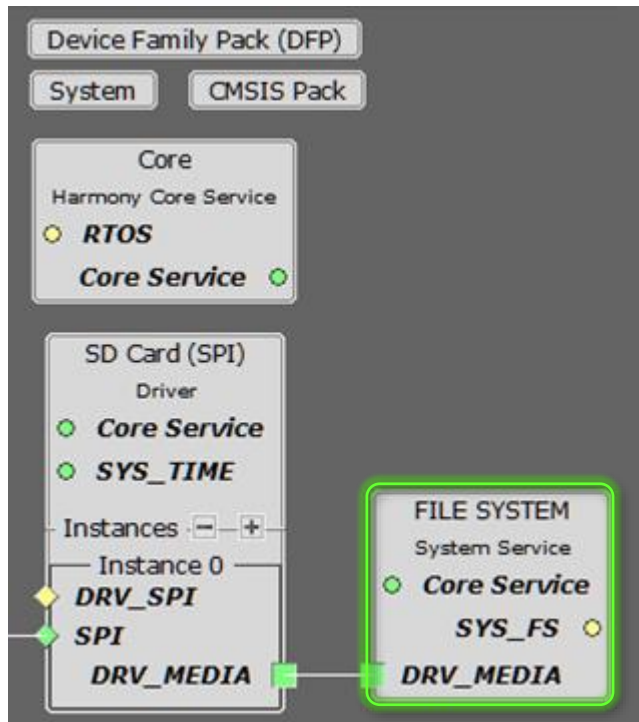
Lab 8

FILE System Auto Mount Feature

FILE System Service

⚙️ Lab8 FILE System Auto Mount Feature

- Harmony FILE system provide the **Auto Mount Feature**. It's helpful to attach or detach media automatically.



✖ Lab8 FILE System Auto Mount Feature

- Code difference after enable the **Auto Mount Feature**.

Use File System Auto Mount Feature? ☐ **configuration.h**

```
#define SYS_FS_AUTOMOUNT_ENABLE false
```

Use File System Auto Mount Feature? ☒

```
#define SYS_FS_AUTOMOUNT_ENABLE true
#define SYS_FS_CLIENT_NUMBER 1
#define SYS_FS_MEDIA_TYPE_IDX0 SYS_FS_MEDIA_TYPE_SD_CARD
#define SYS_FS_TYPE_IDX0 FAT
#define SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0 "/mnt/myDrive1"
#define SYS_FS_MEDIA_IDX0_DEVICE_NAME_VOLUME_IDX0 "/dev/mmcblk1a1"
```

Use File System Auto Mount Feature? ☐ **initialization.c**

```
const SYS_FS_MEDIA_MOUNT_DATA sysfsMountTable[SYS_FS_VOLUME_NUMBER] =
{
    {NULL}
};
```

Use File System Auto Mount Feature? ☒

```
const SYS_FS_MEDIA_MOUNT_DATA sysfsMountTable[SYS_FS_VOLUME_NUMBER] =
{
    {
        .mountName = SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0,
        .devName   = SYS_FS_MEDIA_IDX0_DEVICE_NAME_VOLUME_IDX0,
        .mediaType = SYS_FS_MEDIA_TYPE_IDX0,
        .fsType    = SYS_FS_TYPE_IDX0
    },
};
```

Lab8 FILE System Auto Mount Feature

- API is available when **SYS_FS_AUTOMOUNT_ENABLE** has set to **true**.

Use File System Auto Mount Feature? ☒

sys_fs.c

```
void SYS_FS_EventHandlerSet ( const void * eventHandler, const uintptr_t context )
{
    SYS_FS_MEDIA_MANAGER_EventHandlerSet( eventHandler, context );
}
```

- **Summary:**

Allows a client to identify an event handling function for the file system to call back when mount/unmount operation has completed.

- **Description:**

This function allows a client to identify an event handling function for the File System to call back when mount/unmount operation has completed.

The file system will pass mount name back to the client by calling "eventHandler" when AutoMount feature is enabled for File system.

- **Parameters:**

eventHandler - Pointer to the event handler function implemented by the user.

context - The value of parameter will be passed back to the client unchanged, when the eventHandler function is called. It can be used to identify any client specific data object that identifies the instance of the client module (for example, it may be a pointer to the client module's state structure).

✖ Lab8 FILE System Auto Mount Feature

- Modify your main.c as below

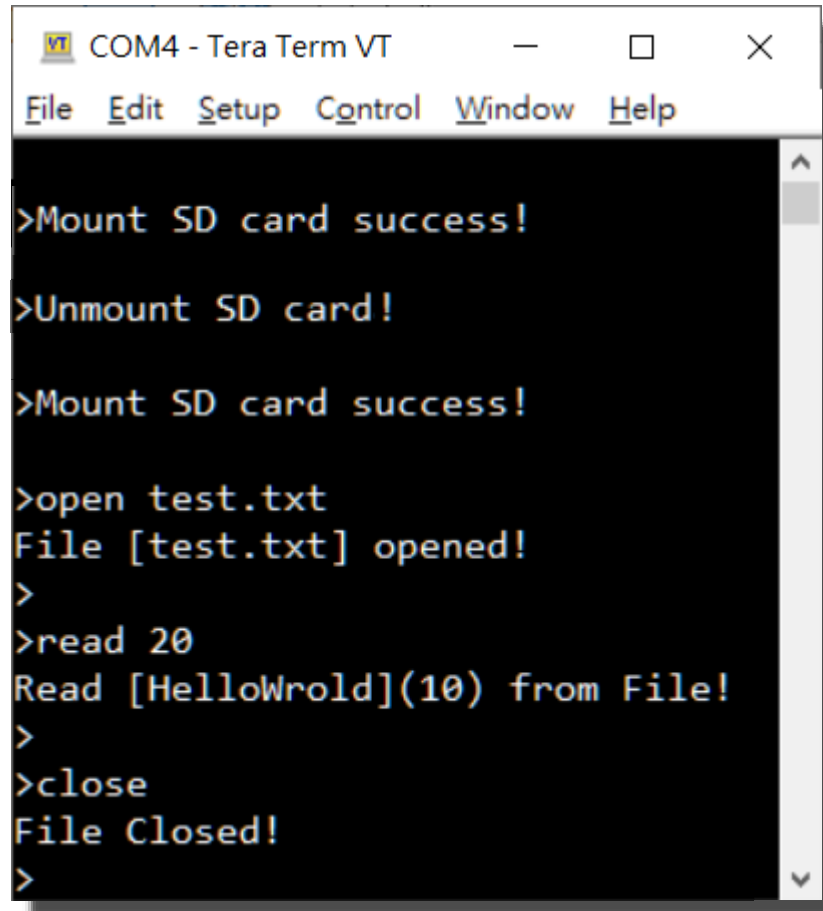
configuration.h

```
#define SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0 "/mnt/myDrive1"  
#define SYS_FS_MEDIA_IDX0_DEVICE_NAME_VOLUME_IDX0 "/dev/mmcblk1"
```

```
void AutoMountEventHandler( SYS_FS_EVENT event,void* eventData, uintptr_t context )  
{  
    switch(event)  
    {  
        case SYS_FS_EVENT_MOUNT:  
        case SYS_FS_EVENT_MOUNT_WITH_NO_FILESYSTEM:  
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0) == 0)  
            {  
                SYS_CMD_MESSAGE("Mount SD card success!\r\n");  
            }  
            break;  
  
        case SYS_FS_EVENT_UNMOUNT:  
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0) == 0)  
            {  
                SYS_CMD_MESSAGE("Unmount SD card!\r\n");  
            }  
            break;  
  
        case SYS_FS_EVENT_ERROR:  
            break;  
    }  
}  
  
int main( void )  
{  
    SYS_Initialize(NULL);  
    SYS_FS_EventHandlerSet(AutoMountEventHandler, (uintptr_t)NULL);  
}
```

⚙️ Lab8 FILE System Auto Mount Feature

- Compiler and programming your project to EVB.
- **Remove SD card firstly.**
- **Put in** SD card and **Remove** it to check the output message.



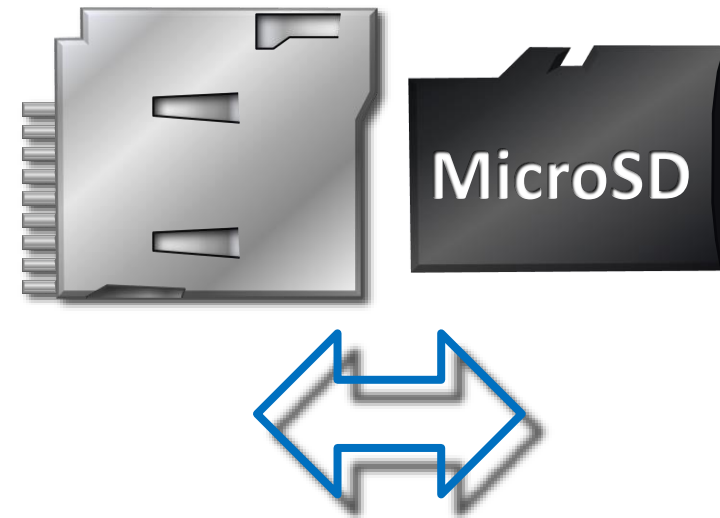
```
COM4 - Tera Term VT
File Edit Setup Control Window Help

>Mount SD card success!

>Unmount SD card!

>Mount SD card success!

>open test.txt
File [test.txt] opened!
>
>read 20
Read [HelloWrold](10) from File!
>
>close
File Closed!
>
```

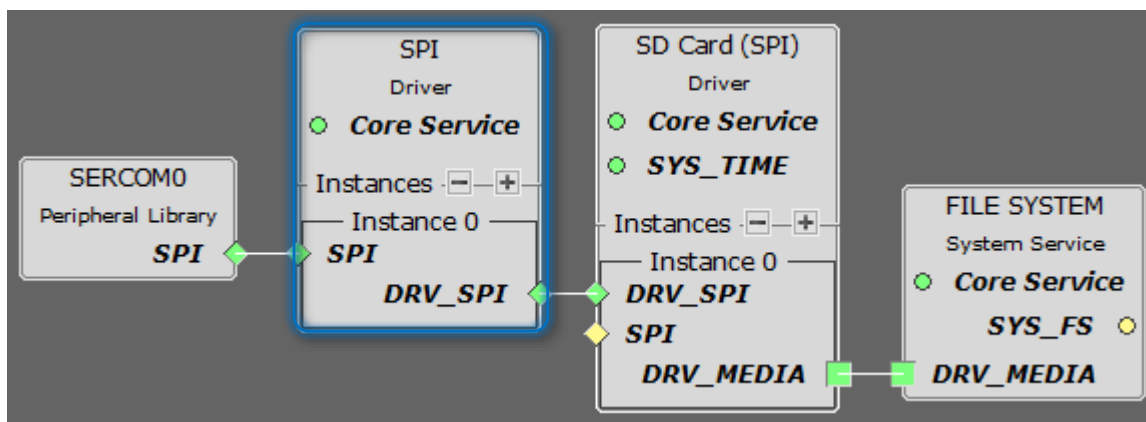


⚙️ Lab8 FILE System Auto Mount Feature

• Discuss

How to use the SPI Master Hardware Slave Select (CS) but not use a GPIO to control ?

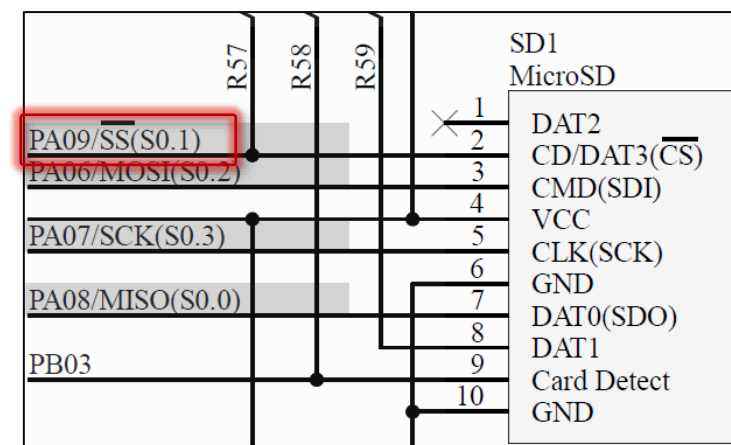
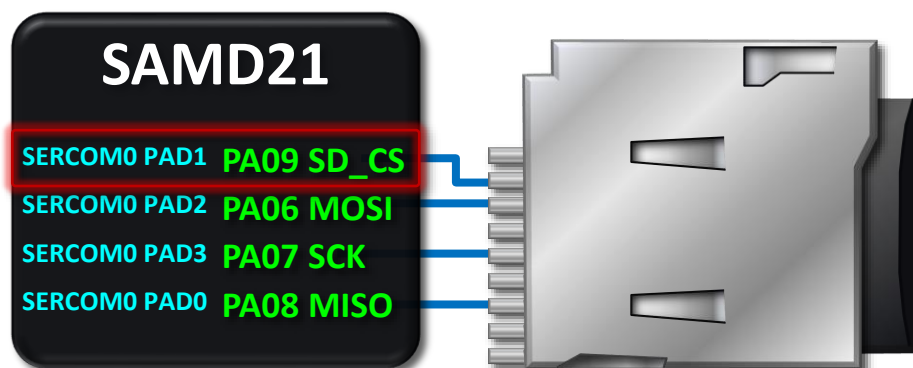
The SPI Master Hardware Slave Select need **SPI Driver** to support, and SPI SD Card had connected the chip select pin to a correct **SERCOM** pad. (Example in use APP045v4)



Pin Table

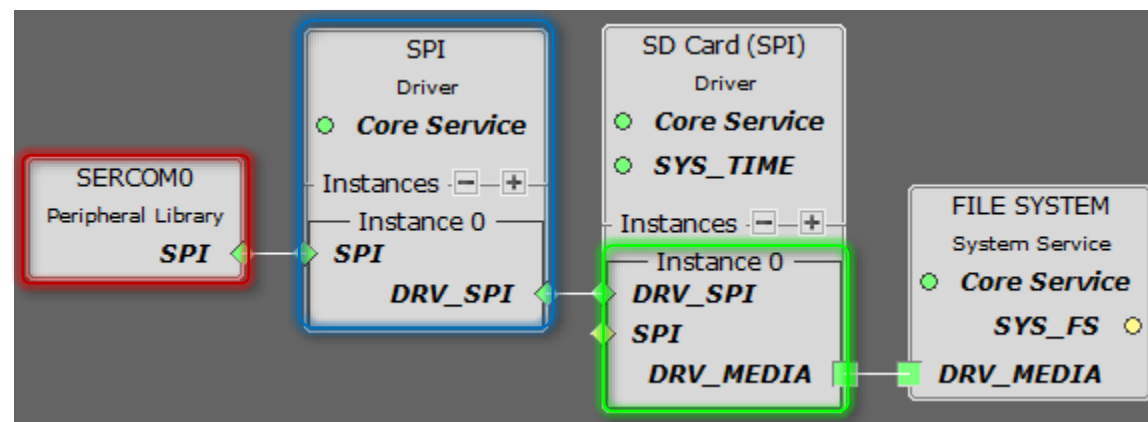
Package: TQFP48

Module	Function	SERCOM0	SERCOM1	SERCOM2	SERCOM3
SERCOM0	SERCOM0_PAD0				
	SERCOM0_PAD1				
	SERCOM0_PAD2				
	SERCOM0_PAD3				



⚙️ Lab8 FILE System Auto Mount Feature

The changes in modules if to use the SPI Master Hardware Slave Select (CS).



Enable SPI Master Hardware Slave Select in SERCOM

Keep default setting of Chip Select Pin In SD Card Driver

SERCOM0

Select SERCOM Operation Mode: **SPI Master**

- Enable Interrupts? ☒
- Enable operation in Standby mode ☐
- SPI Data Out Pad** ☒
- SPI Data In Pad Selection ☐
- SPI Data Order ☐
- SPI Speed in Hz
- SPI Data Character Size
- SPI Clock Phase
- SPI Clock Polarity
- Enable SPI Master Hardware Slave Select** ☒
- SPI Receiver Enable ☒

SPI Transfer Mode 0 is Selected

SPI

PLIB Used: **SERCOM0**

- Number of Clients:
- Transfer Queue Size:
- Use DMA for Transmit and Receive?** ☒
- DMA Channel For Transmit:
- DMA Channel For Receive:

Enable DMA in SPI Driver

SD Card (SPI)

- SPI Driver Instance Used:
- Number of Clients:
- SD Card Speed (Hz):
- Transfer Queue Size:
- Card Detection Method: **Use Polling**
- Polling Interval (ms):
- File system for SDSPI Driver Enabled** ☒
- Chip Select Pin** **PA00**
- !!! Configure the Chip Select pin as GPIO output under
- Use Write Protect Pin (Active High)? ☐
- !!! Note: For each instance of SDSPI, connect either SP

Lab 9

USB Host Mass Storage with file system

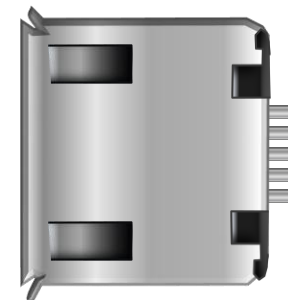
FILE System Service

✖ Lab9 USB Host Mass Storage with file system

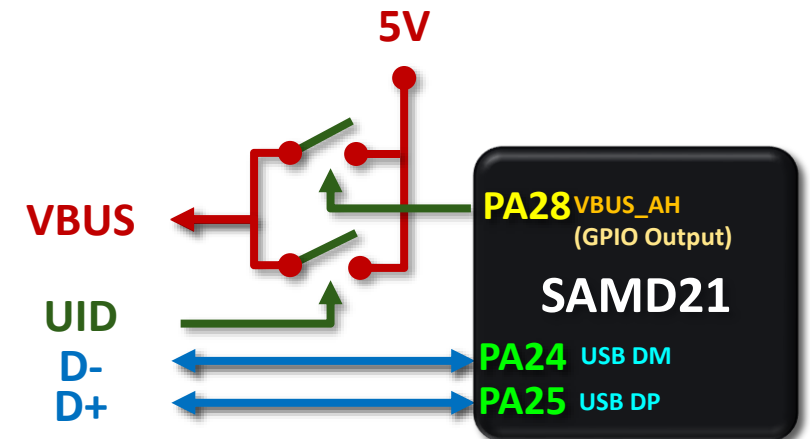
- This Lab to introduce how to use Harmony FILE system Auto Mount Feature to attach a USB pen drive in few steps.
- Try add the **USB MSD** as second media of FILE system service.
- Configure VBUS voltage output control :
 - If use **USB OTG dongle**, the **UID pin** will tie to GND and VBUS will output.
 - If control by Driver, set **PA28** as GPIO output and named to **VBUS_AH**.
- Add USB MSD mount status in **Auto Mount Event Handler**.
- Build a command could change active media as below :
 - **media %** : Set active media to % for operation.
 - % = **SD** for SD card.
 - % = **USB** for USB pen drive.



USB OTG Dongle

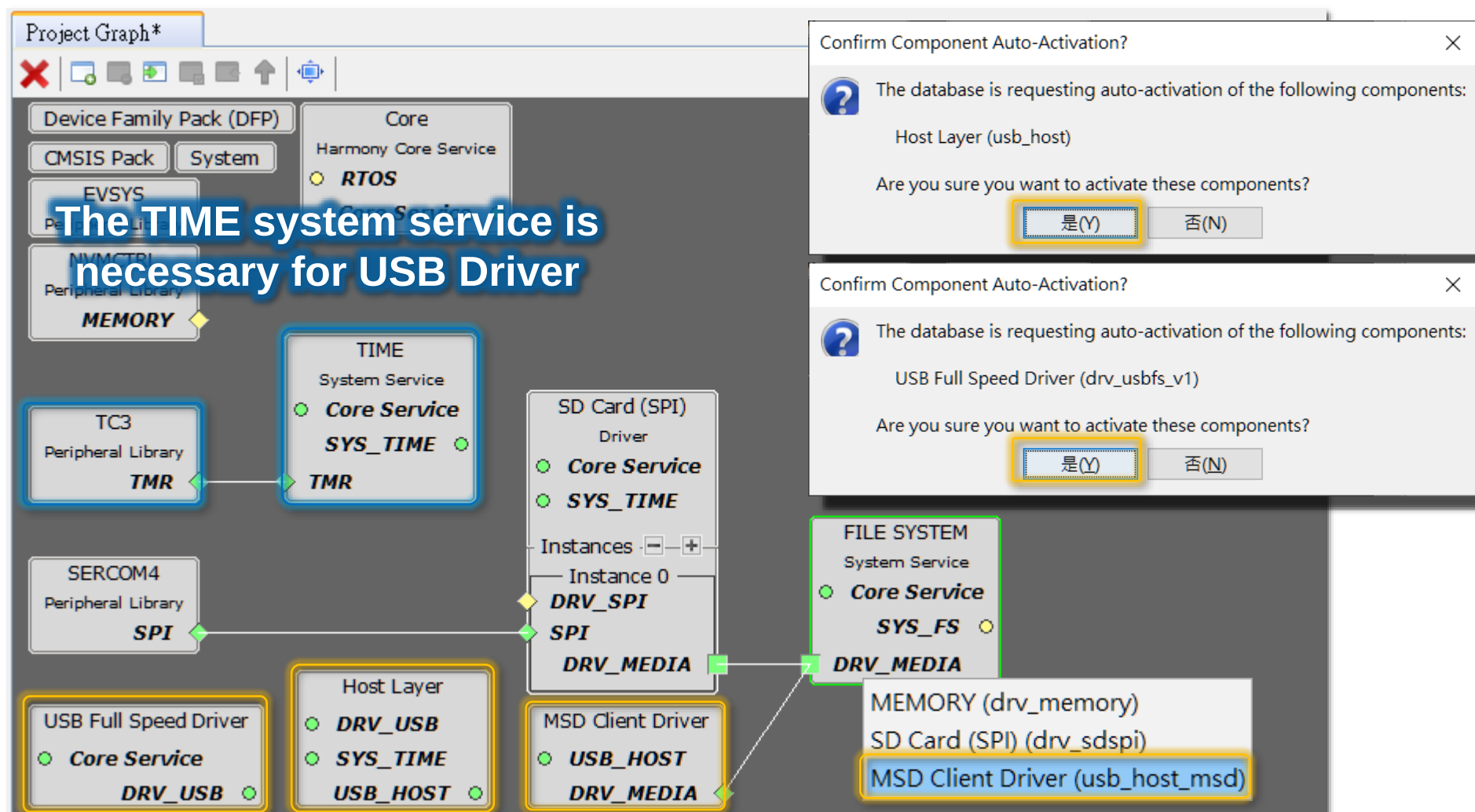


Native USB in Host role



✖ Lab9 USB Host Mass Storage with file system

- Attach media **USB MSD** to **FILE** system service.



✖ Lab9 USB Host Mass Storage with file system

- Configure **FILE SYSTEM** service for 2nd Media

The screenshot shows the 'Configuration Options' window for the 'FILE SYSTEM' service. The 'File System settings' section includes options for 'Maximum Simultaneous File Access' (1), 'Size Of Block' (512), 'Size Of Media Manager Buffer' (2,048), 'Use File System Auto Mount Feature?' (checked), 'Total Number Of Media' (2), and 'Total Number File System Clients' (1). The 'Media0' and 'Media1' sections are expanded. 'Media1' is highlighted with a red box. Under 'Media1', 'Media configuration1' is expanded, showing 'Media Type' set to 'SYS_FS_MEDIA_TYPE_MSD', 'File System Type' set to 'FAT', and 'Number Of Volumes' set to 1. Under 'Volume0', 'Volume configuration0' is expanded, showing 'Device Name' set to '/dev/sda1' and 'Media Mount Name' set to '/mnt/myDrive2'. A green box highlights the 'Use File System Auto Mount Feature?' checkbox. A red box highlights the 'Total Number Of Media' spinner. A yellow box highlights the 'Media Type' dropdown. A blue box highlights the 'Media Mount Name' text field. A green box highlights the 'Device Name' text field. A green box highlights the 'FILE SYSTEM' service block in the Project Graph, showing 'System Service', 'Core Service', 'SYS_FS', and 'DRV_MEDIA'.

Use Auto Mount Feature

Increase Number of Media

Choose 2nd Media Type to MSD
Device Name will be /dev/sda1

Modify Mount Name to /mnt/myDrive2

⚙️ Lab9 USB Host Mass Storage with file system

- Configure **USB Full Speed Driver** **without** VBUS output control.

USB OTG Dongle



Configuration Options* Help

- [-] +
- [-] USB Full Speed Driver
 - [-] USB Mode Selection Host
 - [-] Generate VBUS Enable Function ☐
 - VBUS Enable Pin Name VBUS_AH
 - Attach De-bounce Duration (mSec) 500
 - Reset Duration (mSec) 100

Do not enable VBUS Function if you use the USB OTG dongle

Project Graph*

Device Family Pack (DFP)

- CMSIS Pack
- System
- EVSYS Peripheral Library
- NVMCTRL Peripheral Library **MEMORY**
- TC3 Peripheral Library **TMR**
- SERCOM4 Peripheral Library **SPI**
- USB Full Speed Driver **Core Service** **DRV_USB**

Core

- Harmony Core Service
- RTOS**
- Core Service**
- TIME System Service **Core Service** **SYS_TIME**
- SD Card (SPI) Driver
 - Core Service**
 - SYS_TIME**
 - Instances [-] +
 - Instance 0
 - DRV_SPI**
 - SPI**
 - DRV_MEDIA**
- Host Layer
 - DRV_USB**
 - SYS_TIME**
 - USB_HOST**
- MSD Client Driver
 - USB_HOST**
 - DRV_MEDIA**
- FILE SYSTEM System Service **Core Service** **SYS_FS**
 - DRV_MEDIA**

✖ Lab9 USB Host Mass Storage with file system

- Configure USB D-/D+ on PA24/PA25
- Do not configure the VBUS_AH pin if no VBUS Enable Function.

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

Order: Ports v Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
33	PA24		USB_DM	Digital	High Impedance	n/a	☐	☐	NORMAL
34	PA25		USB_DP	Digital	High Impedance	n/a	☐	☐	NORMAL

USB Full Speed Driver

USB Mode Selection Host v

Generate VBUS Enable Function ☐

VBUS Enable Pin Name VBUS_AH

Attach De-bounce Duration (mSec) 500

Reset Duration (mSec) 100

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

Package: TQFP48 v

Module	Function	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	27	28	30	31	2	3
USB	USB_DM																							
	USB_DP																							
	USB_SOF_1KHZ																							

⚙️ Lab9 USB Host Mass Storage with file system

- Add USB MSD mount status in Auto Mount Event Handler.

```
static const SYS_CMD_DESCRIPTOR MyCmdTbl[] = { ... };

void AutoMountEventHandler( SYS_FS_EVENT event, void* eventData, uintptr_t context )
{
    switch(event)
    {
        case SYS_FS_EVENT_MOUNT:
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Mount SD card success!\r\n");
            }
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX1_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Mount USB Mass Storage success!\r\n");
            }
            break;

        case SYS_FS_EVENT_UNMOUNT:
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Unmount SD card!\r\n");
            }
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX1_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Unmount USB Mass Storage!\r\n");
            }
            break;

        case SYS_FS_EVENT_ERROR:
            break;
    }
}
```

⚙️ Lab9 USB Host Mass Storage with file system

- Add a new command “**media**” to select active Drive of Media.

```
void CommandClose(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }

void CommandMedia(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    if(!strcmp( argv[1], "SD"))
    {
        strcpy(fMedia, SYS_FS_MEDIA_IDX0_DEVICE_NAME_VOLUME_IDX0);
        strcpy(fDrive, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0);
    }
    if(!strcmp( argv[1], "USB"))
    {
        strcpy(fMedia, SYS_FS_MEDIA_IDX1_DEVICE_NAME_VOLUME_IDX0);
        strcpy(fDrive, SYS_FS_MEDIA_IDX1_MOUNT_NAME_VOLUME_IDX0);
    }

    SYS_CMD_PRINT("Set media to %s\r\n", fMedia);

    if(SYS_FS_CurrentDriveSet(fDrive) != SYS_FS_RES_SUCCESS)
    {
        SYS_CMD_PRINT("Drive %s not mounted!\r\n", fDrive);
        return;
    }

    SYS_CMD_PRINT("Set drive to %s\r\n", fDrive);
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[]=
{
    { "close", CommandClose, ": Close file" },
    { "media", CommandMedia, ": media %" },
};

void AutoMountEventHandler( SYS_FS_EVENT event,void* eventData, uintptr_t context ) { ... }
```

⚙️ Lab9 USB Host Mass Storage with file system

- Register the Host Event Handler and enable USB Host Bus.

```
void AutoMountEventHandler( SYS_FS_EVENT event,void* eventData, uintptr_t context ) { ... }

USB_HOST_EVENT_RESPONSE USBHostEventHandler(USB_HOST_EVENT event,
                                             void * eventData, uintptr_t context)
{
    switch (event)
    {
        case USB_HOST_EVENT_DEVICE_REJECTED_INSUFFICIENT_POWER: break;
        case USB_HOST_EVENT_DEVICE_UNSUPPORTED: break;
        case USB_HOST_EVENT_HUB_TIER_LEVEL_EXCEEDED: break;
        case USB_HOST_EVENT_PORT_OVERCURRENT_DETECTED: break;
    }

    return(USB_HOST_EVENT_RESPONSE_NONE);
}

int main( void )
{
    SYS_Initialize(NULL);

    SYS_FS_EventHandlerSet(AutoMountEventHandler, (uintptr_t)NULL);

    SYS_CMD_ADDGRP(MyCmdTbl, sizeof(MyCmdTbl)/sizeof(SYS_CMD_DESCRIPTOR),
                  "MYCMD", ": My Commands");

    USB_HOST_EventHandlerSet(USBHostEventHandler, (uintptr_t)NULL);

    USB_HOST_BusEnable(0);

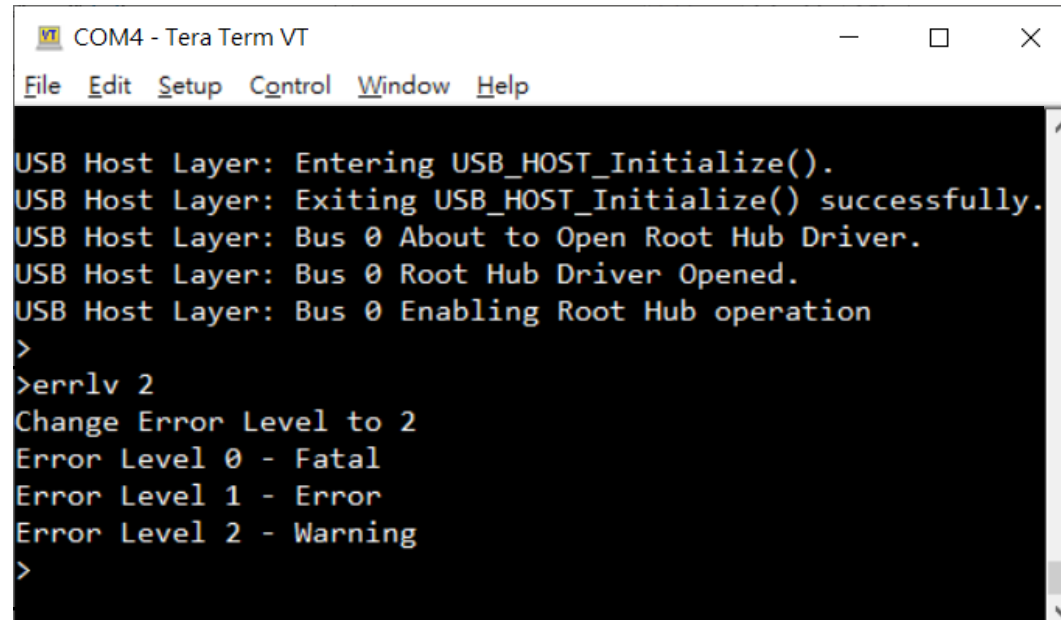
    while( true ) { ... }

    ...
}
```

⚙️ Lab9 USB Host Mass Storage with file system

- Remove SD card and USB pen drive firstly.
- Compiler and programming your project to EVB.
- The **USB Host Layer** debug message will output after reset, that is because the **SYS_DEBUG** been used in USB stack and the error level is **SYS_ERROR_INFO (3)**.
- We could execute the command **"errlv 2"** to stop USB stack debug message.

```
SYS_MODULE_OBJ USB_HOST_Initialize(const SYS_MODULE_INIT * initData )      usb_host.c
{
    ...
    result =SYS_MODULE_OBJ_INVALID;
    SYS_DEBUG_MESSAGE(SYS_ERROR_INFO, "\r\nUSB Host Layer: Entering USB_HOST_Initialize().");
}
```

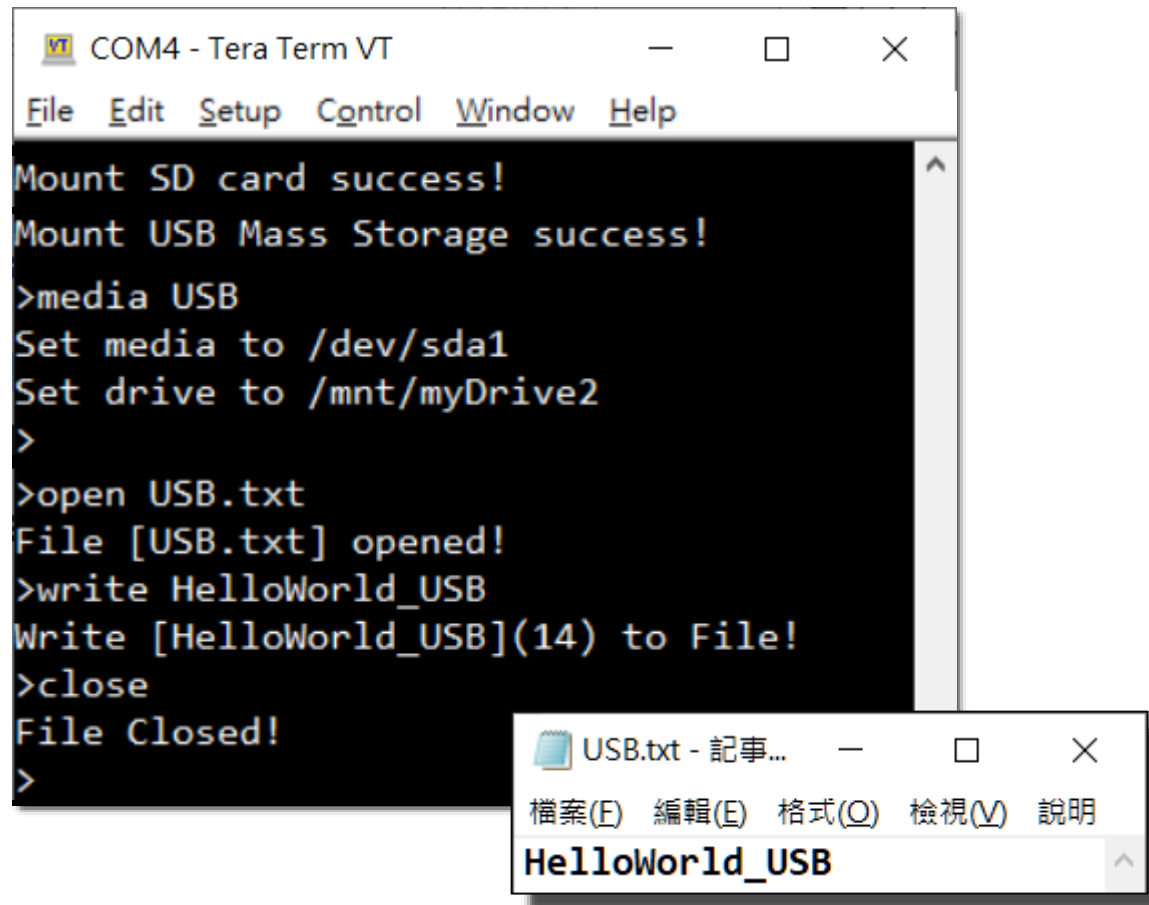


```
COM4 - Tera Term VT
File Edit Setup Control Window Help

USB Host Layer: Entering USB_HOST_Initialize().
USB Host Layer: Exiting USB_HOST_Initialize() successfully.
USB Host Layer: Bus 0 About to Open Root Hub Driver.
USB Host Layer: Bus 0 Root Hub Driver Opened.
USB Host Layer: Bus 0 Enabling Root Hub operation
>
>errlv 2
Change Error Level to 2
Error Level 0 - Fatal
Error Level 1 - Error
Error Level 2 - Warning
>
```

✖ Lab9 USB Host Mass Storage with file system

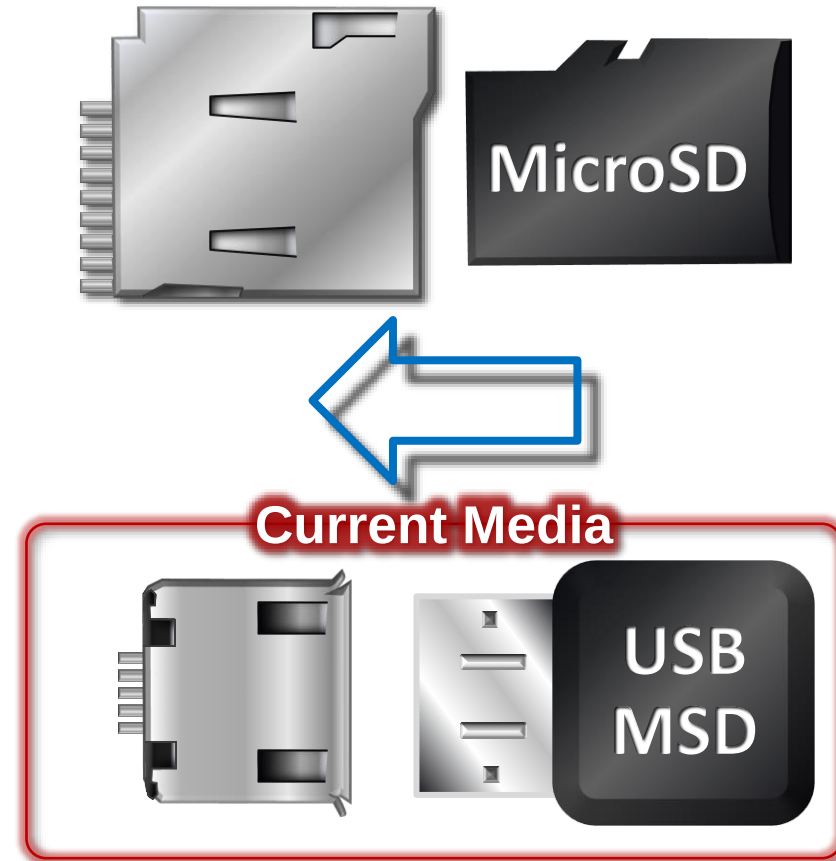
- Put in SD card and USB pen drive individually.
- Command “**media USB**” change active media to USB pen drive.
- Try file operation as below and check content in USB pen drive.



The image shows a Tera Term VT terminal window titled 'COM4 - Tera Term VT' with a menu bar (File, Edit, Setup, Control, Window, Help). The terminal output is as follows:

```
Mount SD card success!  
Mount USB Mass Storage success!  
>media USB  
Set media to /dev/sda1  
Set drive to /mnt/myDrive2  
>  
>open USB.txt  
File [USB.txt] opened!  
>write HelloWorld_USB  
Write [HelloWorld_USB](14) to File!  
>close  
File Closed!  
>
```

Below the terminal window is a text editor window titled 'USB.txt - 記事...' with a menu bar (檔案(E), 編輯(E), 格式(O), 檢視(V), 說明). The text 'HelloWorld_USB' is visible in the editor.

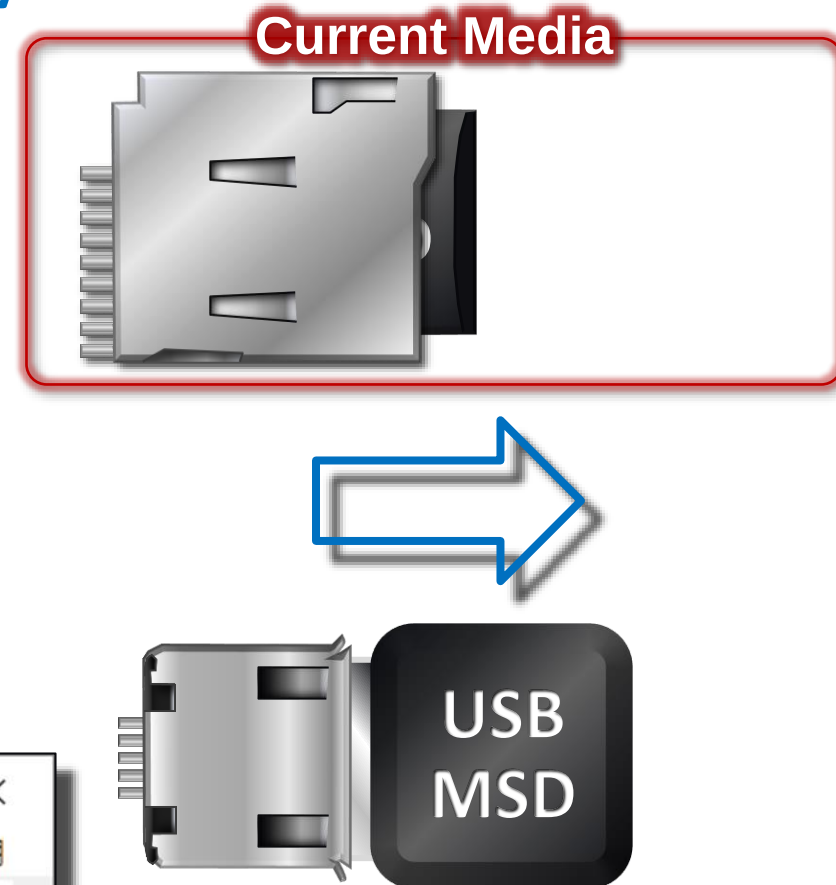


⚙️ Lab9 USB Host Mass Storage with file system

- Command “**media SD**” to change active media to SD Card.
- Try file operation as below and check the content in SD card.
- Remove SD card and USB pen drive individually.

```
COM4 - Tera Term VT
File Edit Setup Control Window Help
>media SD
Set media to /dev/mmcblk1
Set drive to /mnt/myDrive1
>
>open SD.txt
File [SD.txt] opened!
>
>write HelloWorld_SD
Write [HelloWorld_SD](13) to File!
>close
File Closed!
>Unmount SD card!
Unmount USB Mass Storage!
```

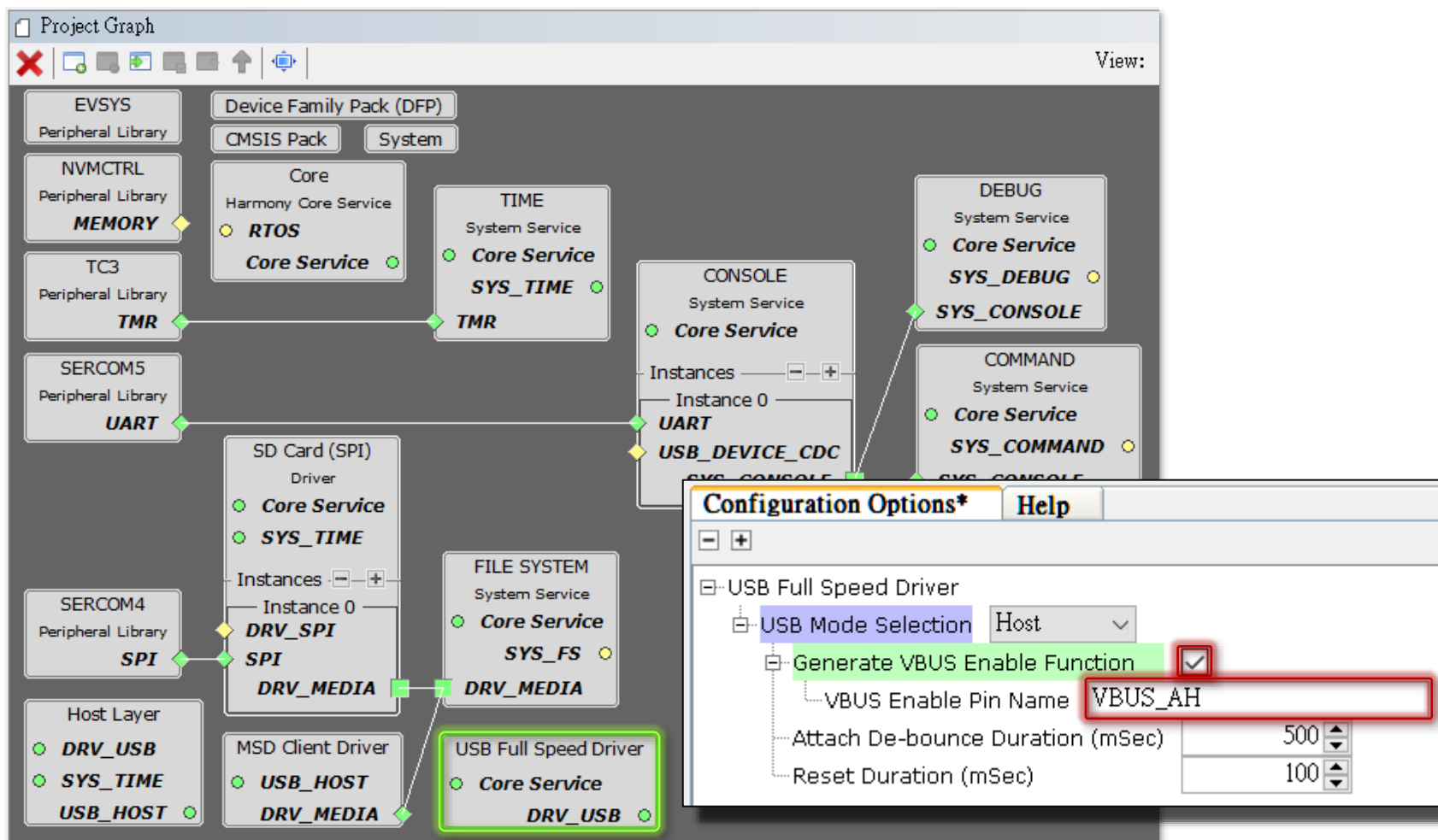
```
SD.txt - 記事本
檔案(E) 編輯(E) 格式(O) 檢視(V) 說明
HelloWorld_SD
```



✖ Lab9 USB Host Mass Storage with file system

• Discuss

- How about if generate VBUS Enable Function in **USB Full Speed Driver** for VBUS output control by USB driver ?



✖ Lab9 USB Host Mass Storage with file system

- The VBUS control pin on GPIO **PA28**, Custom Name to **VBUS_AH** as the name you set in previous page, direction is **Output** and Latch to **High**. (VBUS output is **Low** active)
- Configure this only when you to generate VBUS Enable Function.**

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

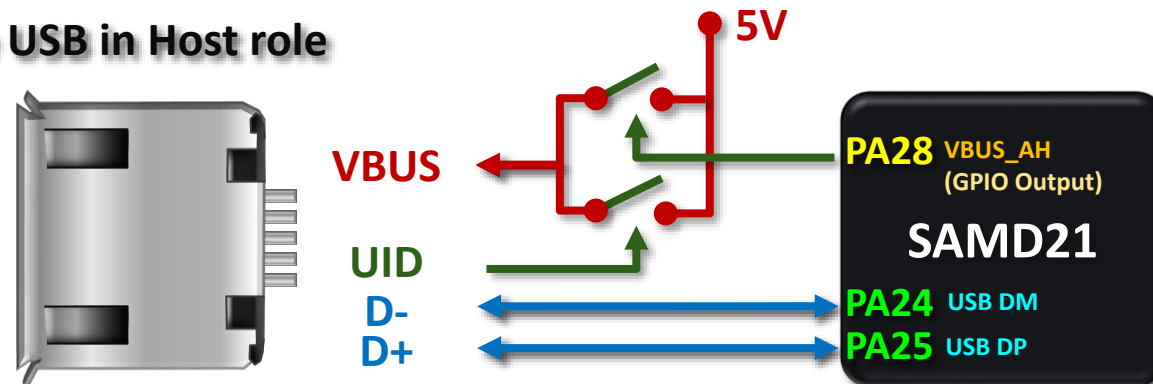
Order: Ports v Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
33	PA24		USB_DM v	Digital	High Impedance v	n/a	☐	☐	NORMAL v
34	PA25		USB_DP v	Digital	High Impedance v	n/a	☐	☐	NORMAL v
41	PA28	VBUS_AH	GPIO v	Digital	Out v	High	☐	☐	NORMAL

USB Full Speed Driver

- USB Mode Selection Host v
- Generate VBUS Enable Function** ☒
 - VBUS Enable Pin Name VBUS_AH
 - Attach De-bounce Duration (mSec) 500
 - Reset Duration (mSec) 100

Native USB in Host role



✖ Lab9 USB Host Mass Storage with file system

- Add two VBUS power control functions in \Source Files\config\default**initialization.c** , That are **VBUS_AH_PowerEnable()** and **VBUS_AH_PowerDisable()**.

```

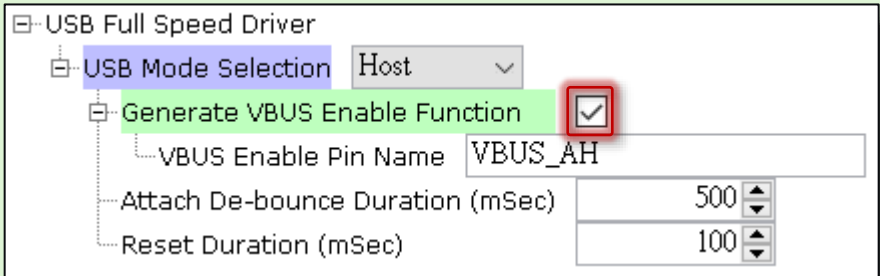
/*****
 * USB Driver Initialization
 *****/
void VBUS_AH_PowerEnable( void )
{
    VBUS_AH_Clear();
}

void VBUS_AH_PowerDisable( void )
{
    VBUS_AH_Set();
}

void _DRV_USB_VBUSPowerEnable(uint8_t port, bool enable)
{
    /* Note: When operating in Host mode, the application can specify a Root
    ...
    User must use the MHC pin configuration utility to configure the pin
    used for enabling VBUS */
    if (enable == true)
    {
        /* Enable the VBUS */
        VBUS_AH_PowerEnable();
    }
    else
    {
        /* Disable the VBUS */
        VBUS_AH_PowerDisable();
    }
}

```

initialization.c



⚙️ Lab9 USB Host Mass Storage with file system

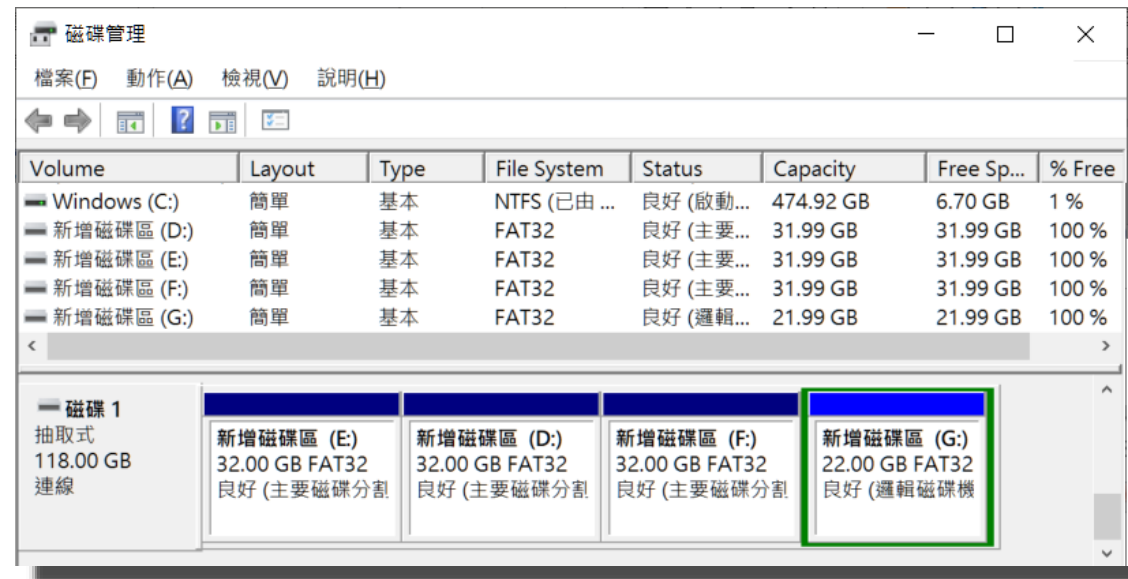
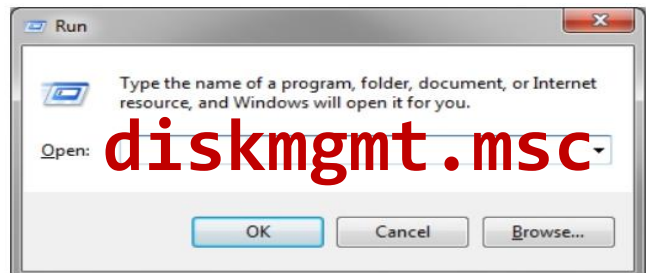
- Discuss

- How to use SD Card / USB pen drive larger than 32GB under Windows ?

Windows operating systems create new **FAT32** volumes only **up to 32 GB** in size. You could use Disk Manager to partition the storage to several 32GB drives.

- microchipdeveloper.com/faq:999

Currently, the **MPLAB[®] Harmony** implementation of the FAT File System has been **tested up to 32 GB** SD cards. The limit for the file system is **2 TB at 512 byte/sector**. At this time, the upper size limit has not been tested.



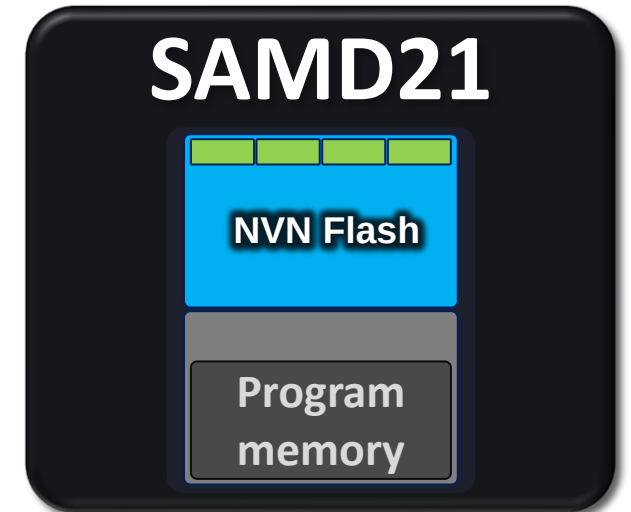
Lab 10

NVMCTRL Internal Flash with file system

FILE System Service

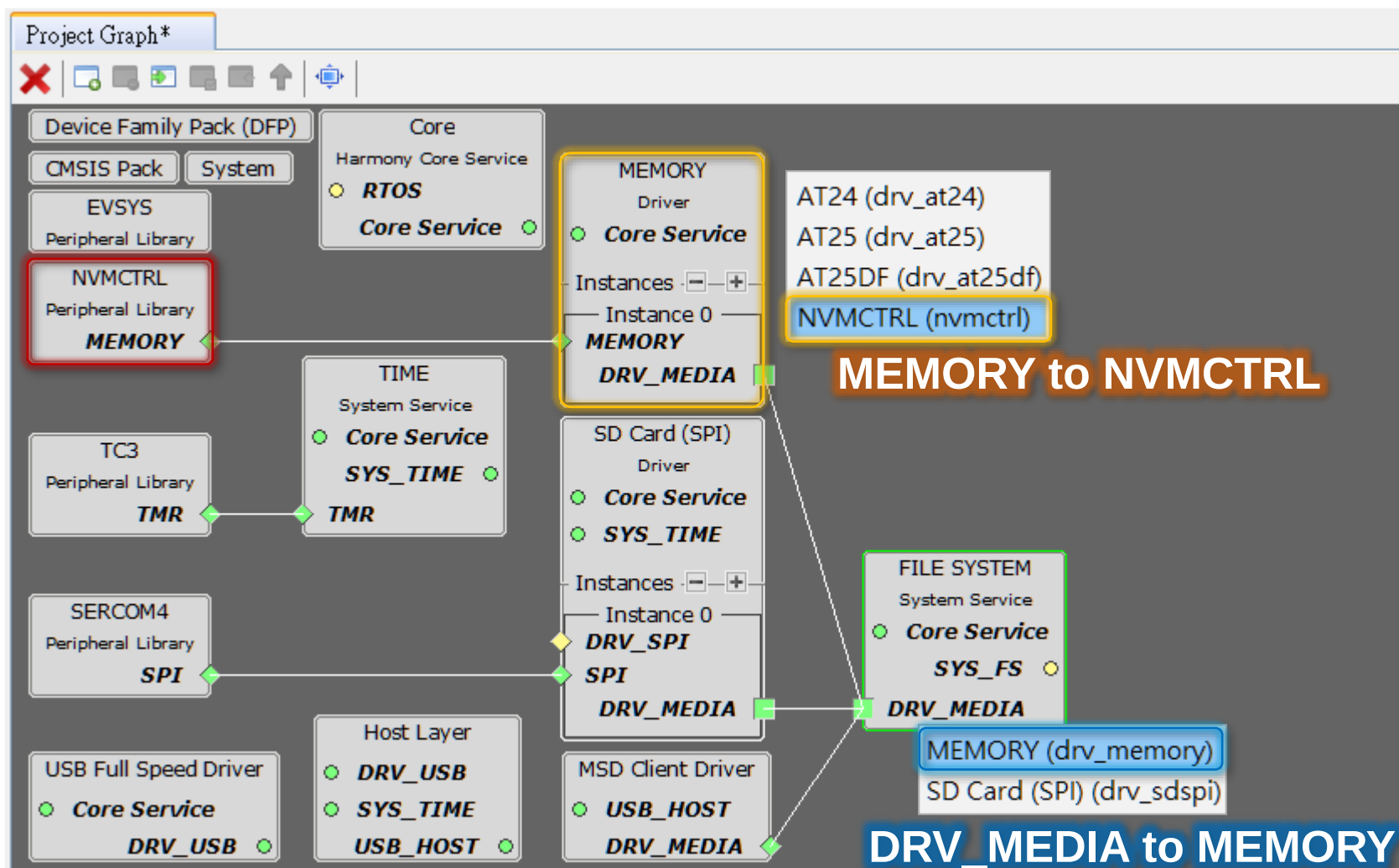
⚙️ Lab10 NVMCTRL Internal Flash with file system

- This Lab to introduce how to use Harmony FILE system to attach the internal NVM Flash as a FILE system drive.
- Try add the **MEMORY** driver as 3rd media of **FILE system service**.
- Choose **NVMCTRL** as satisfiers of **MEMORY** driver.
- Add NVM flash mount status in Auto Mount Event Handler.
- Modify command “**media**” to support media NVM Flash :
media % : Set active media to % for operation.
 - % = SD for SD card.
 - % = USB for USB pen drive.
 - % = **NVM** for NVM flash.
- Build a new command could format active media as below :
format : **Format current media in default FAT configuration.**



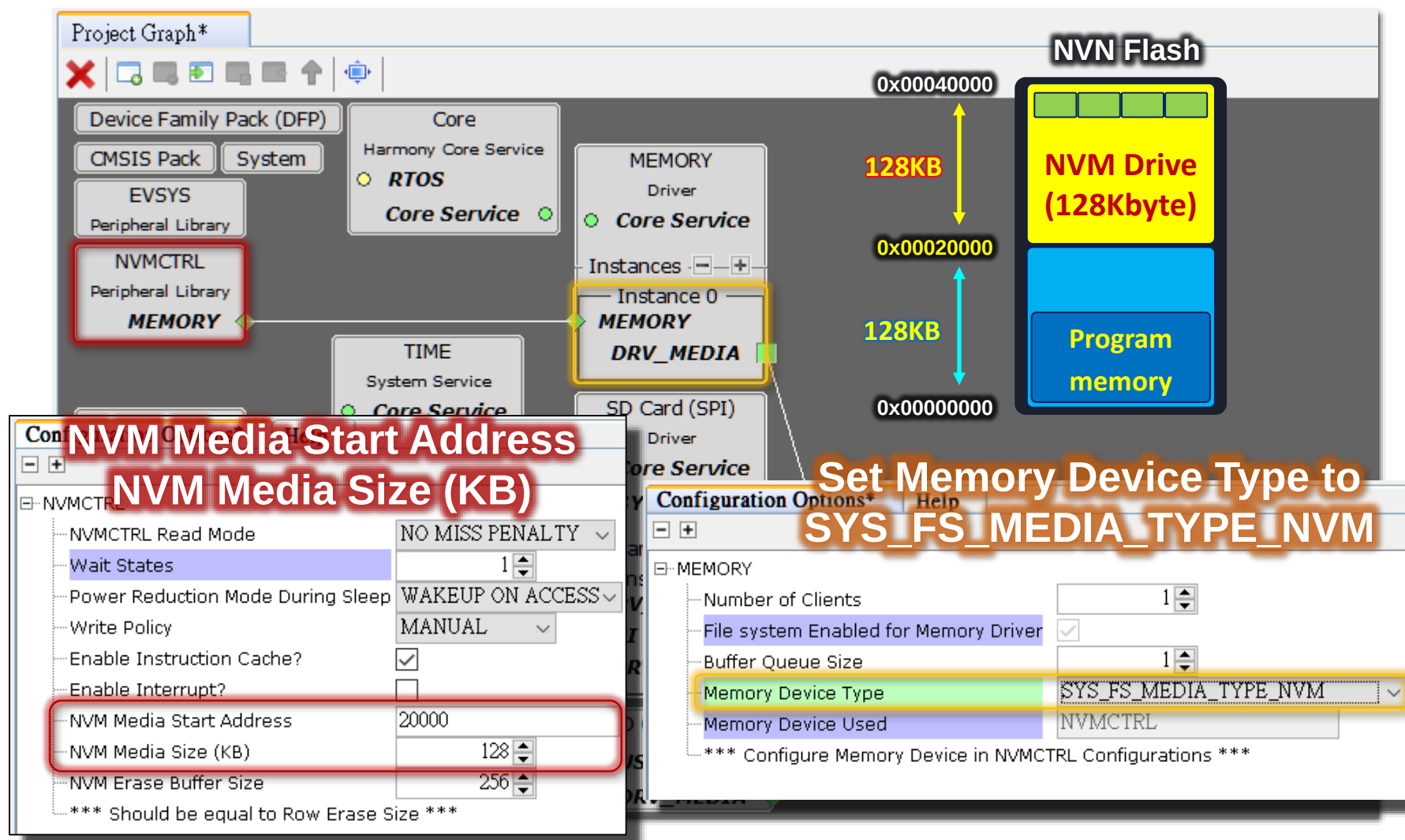
⚙️ Lab10 NVMCTRL Internal Flash with file system

- Attach **MEMORY Driver** + **PLIB NVMCTRL** to **FILE** system service.



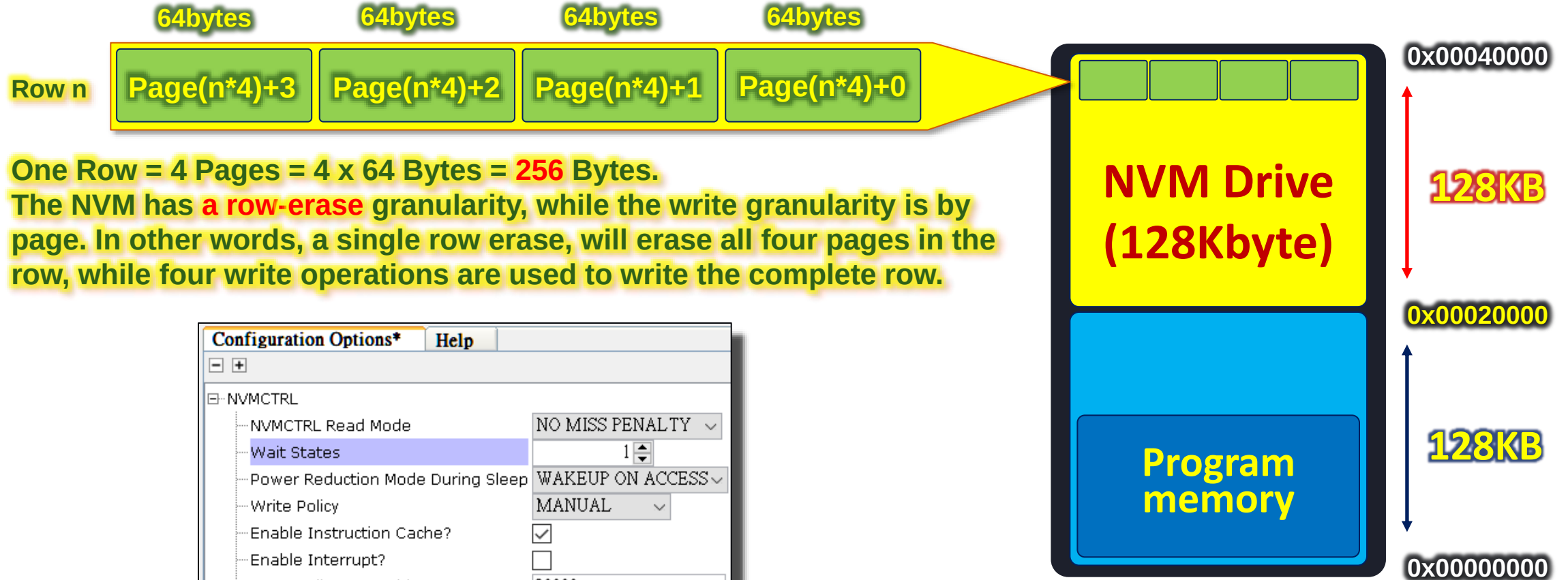
⚙️ Lab10 NVMCTRL Internal Flash with file system

- Configuration of **PLIB NVMCTRL** and **Instance of MEMORY Driver**



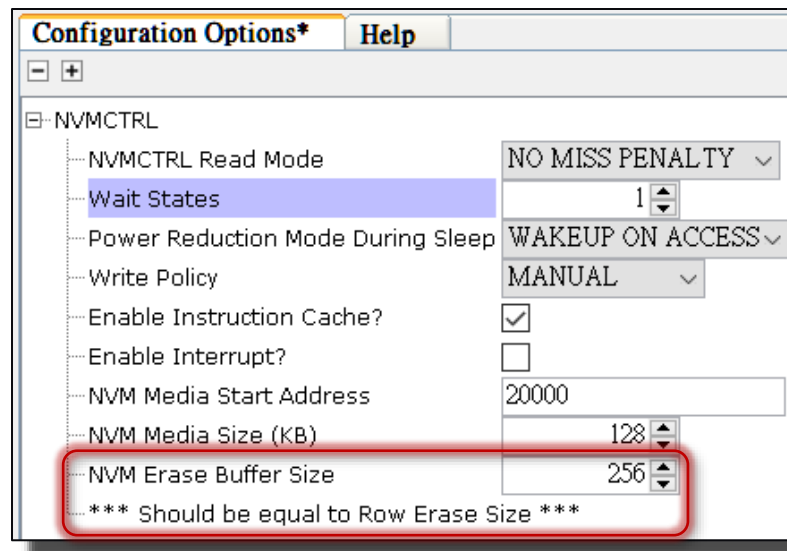
⚙️ Lab10 NVMCTRL Internal Flash with file system

- SAMD21G18A has 256KB flash, the page size is 64 Bytes then total have 4096 pages.
- The NVM Erase Buffer Size should be equal to Row Erase Size of memory organization.



One Row = 4 Pages = 4 x 64 Bytes = **256 Bytes**.

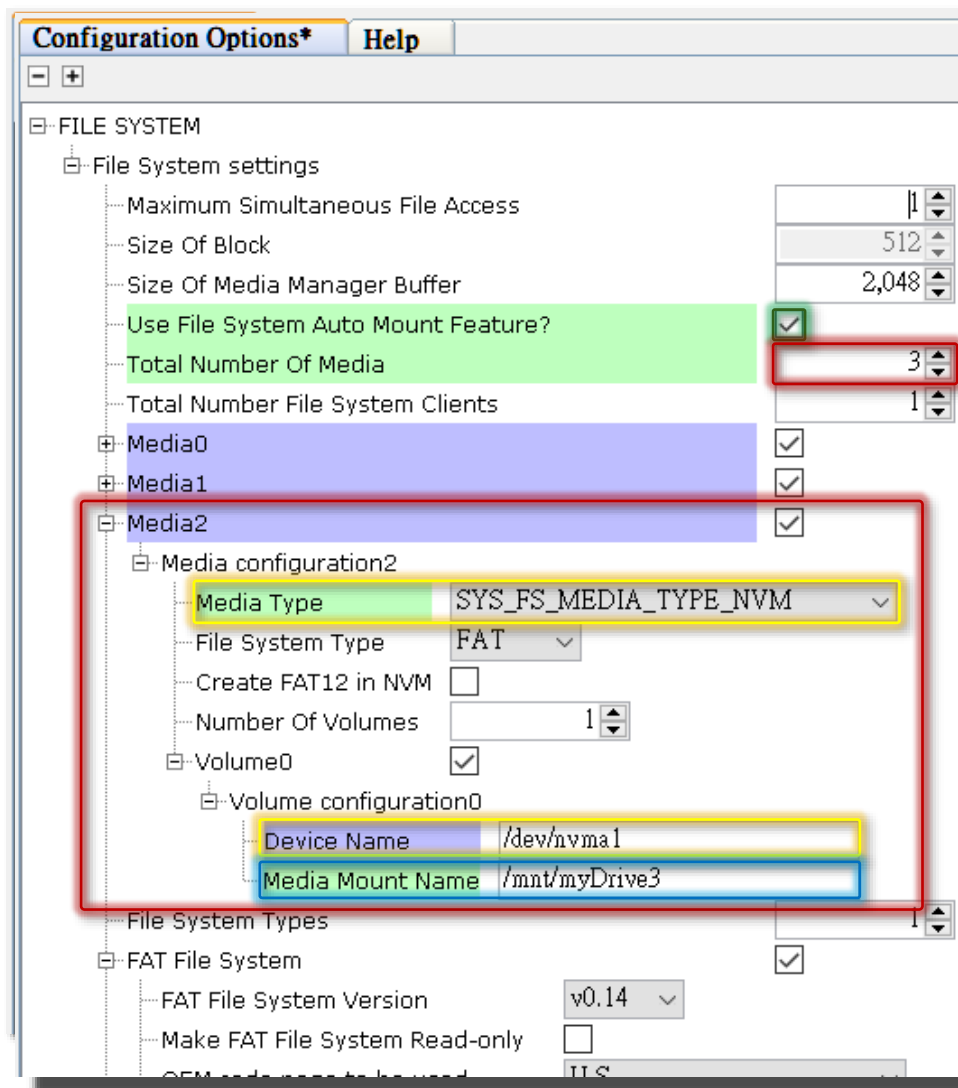
The NVM has a **row-erase** granularity, while the write granularity is by page. In other words, a single row erase, will erase all four pages in the row, while four write operations are used to write the complete row.



Erase buffer Size should be equal to **256 bytes**

⚙️ Lab10 NVMCTRL Internal Flash with file system

- Configure **FILE SYSTEM** service for 3rd Media

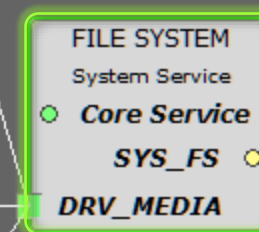


Use Auto Mount Feature

Increase Number of Media

Choose 3rd Media Type to NVM
Device Name will be
/dev/nvma1

Modify Mount Name to
/mnt/myDrive3



⚙️ Lab10 NVMCTRL Internal Flash with file system

- Let's check current Volume configuration of all Media.

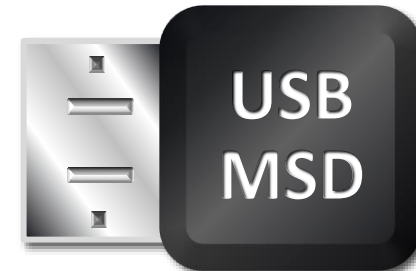
The screenshot shows a configuration window for media devices. It is organized into three sections, one for each media type (Media0, Media1, Media2). Each section contains a 'Media configuration' block with the following fields: 'Media Type' (a dropdown menu), 'File System Type' (a dropdown menu), 'Number Of Volumes' (a numeric input field), and a 'Volume0' block. The 'Volume0' block contains 'Volume configuration0' with 'Device Name' and 'Media Mount Name' text input fields. Media0 is configured as an SD card with FAT file system. Media1 is configured as an MSD with FAT file system. Media2 is configured as NVM with FAT file system and 'Create FAT12 in NVM' unchecked.

Media	Media Type	File System Type	Number Of Volumes	Device Name	Media Mount Name
Media0	SYS_FS_MEDIA_TYPE_SD_CARD	FAT	1	/dev/mmcblk1	/mnt/myDrive1
Media1	SYS_FS_MEDIA_TYPE_MSD	FAT	1	/dev/sda1	/mnt/myDrive2
Media2	SYS_FS_MEDIA_TYPE_NVM	FAT	1	/dev/nvma1	/mnt/myDrive3

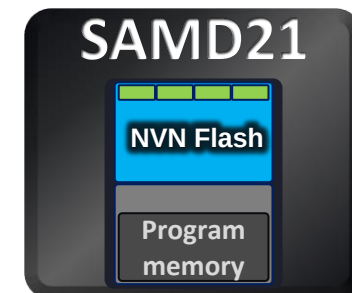
Media0
/dev/mmcblk1
/mnt/myDrive1



Media2
/dev/sda1
/mnt/myDrive2

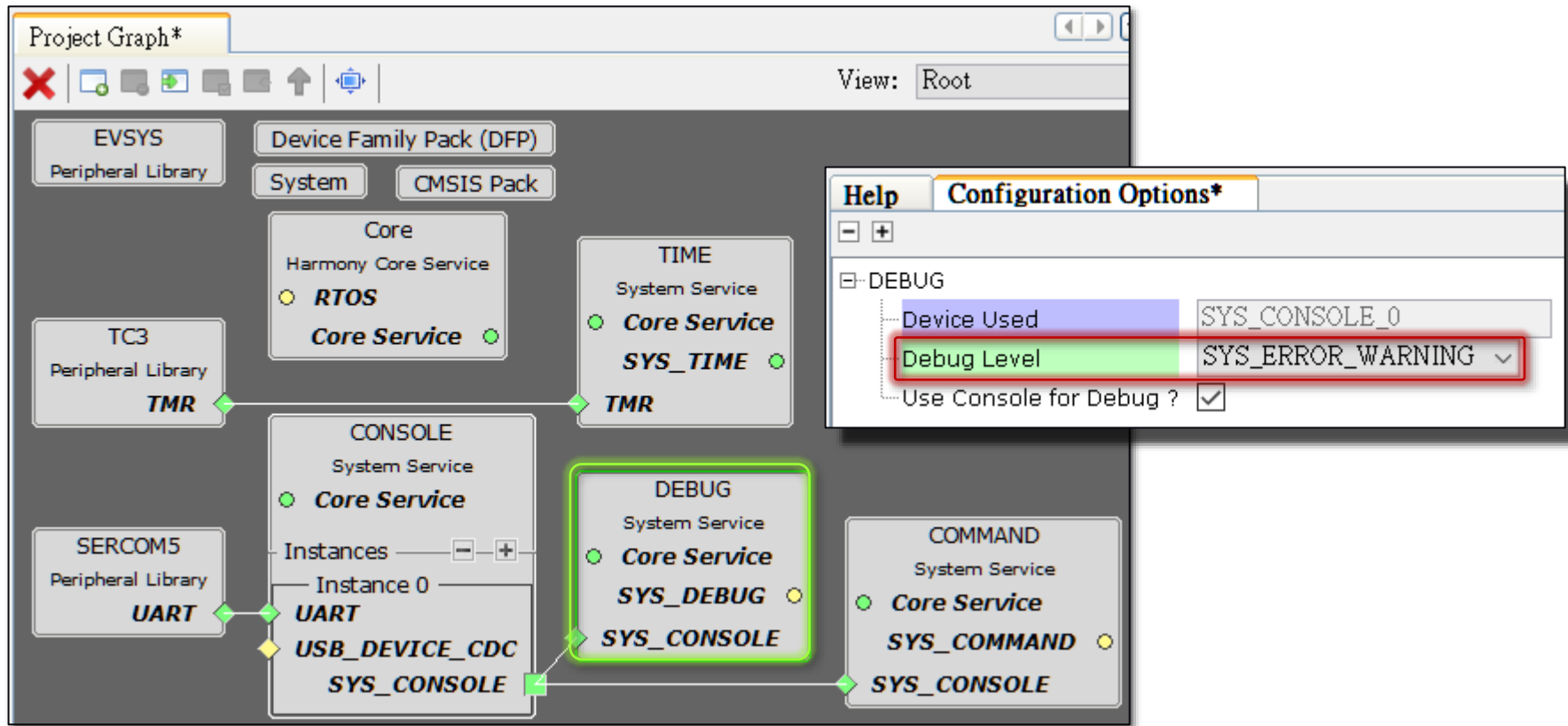


Media2
/dev/nvma1
/mnt/myDrive3



⚙️ Lab10 NVMCTRL Internal Flash with file system

- Let's change the default Debug Error Level to “**SYS_ERROR_WARNING**” in **DEBUG system Service** for disable information message from USB Host Layer.



⚙️ Lab10 NVMCTRL Internal Flash with file system

- Add NVM Flash mount status in Auto Mount Event Handler.

```
static const SYS_CMD_DESCRIPTOR MyCmdTbl[] = { ... };

void AutoMountEventHandler( SYS_FS_EVENT event, void* eventData, uintptr_t context )
{
    switch(event)
    {
        case SYS_FS_EVENT_MOUNT:
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Mount SD card success!\r\n");
            }
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX1_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Mount USB Mass Storage success!\r\n");
            }
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX2_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Mount NVM Flash success!\r\n");
            }
            break;

        case SYS_FS_EVENT_UNMOUNT:
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Unmount SD card!\r\n");
            }
            if(strcmp((const char *)eventData, SYS_FS_MEDIA_IDX1_MOUNT_NAME_VOLUME_IDX0) == 0)
            {
                SYS_CMD_MESSAGE("Unmount USB Mass Storage!\r\n");
            }
            break;
        ...
    }
}
```

NVM Flash doesn't need the unmount message

⚙️ Lab10 NVMCTRL Internal Flash with file system

- Modify command “media” to add NVM Flash support.

```
void CommandClose(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }

void CommandMedia(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    if(!strcmp( argv[1], "SD"))
    {
        strcpy(fMedia, SYS_FS_MEDIA_IDX0_DEVICE_NAME_VOLUME_IDX0);
        strcpy(fDrive, SYS_FS_MEDIA_IDX0_MOUNT_NAME_VOLUME_IDX0);
    }
    if(!strcmp( argv[1], "USB"))
    {
        strcpy(fMedia, SYS_FS_MEDIA_IDX1_DEVICE_NAME_VOLUME_IDX0);
        strcpy(fDrive, SYS_FS_MEDIA_IDX1_MOUNT_NAME_VOLUME_IDX0);
    }
    if(!strcmp( argv[1], "NVM"))
    {
        strcpy(fMedia, SYS_FS_MEDIA_IDX2_DEVICE_NAME_VOLUME_IDX0);
        strcpy(fDrive, SYS_FS_MEDIA_IDX2_MOUNT_NAME_VOLUME_IDX0);
    }

    SYS_CMD_PRINT("Set media to %s\r\n", fMedia);

    if(SYS_FS_CurrentDriveSet(fDrive) != SYS_FS_RES_SUCCESS)
    {
        SYS_CMD_PRINT("Drive %s not mounted!\r\n", fDrive);
        return;
    }

    SYS_CMD_PRINT("Set drive to %s\r\n", fDrive);
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[] = { ... };
```

⚙️ Lab10 NVMCTRL Internal Flash with file system

- Add a new command “**format**” to format active Drive as FAT16.

```
void CommandMedia(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv) { ... }

void CommandFormat(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_FS_FORMAT_PARAM opt = { 0 };
    uint8_t CACHE_ALIGN work[512];

    SYS_FS_CurrentDriveGet( fDrive );
    opt.fmt = SYS_FS_FORMAT_FAT;
    opt.au_size = 0;
    opt.n_fat = 1;
    opt.align = 0;
    opt.n_root = 1;

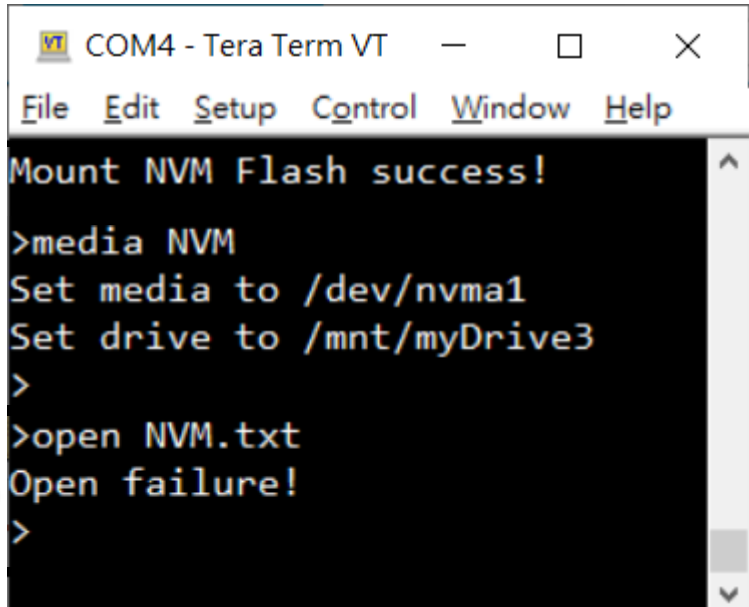
    fResult = SYS_FS_DriveFormat(fDrive, &opt, (void *)work, 512);
    if(fResult == SYS_FS_RES_SUCCESS)
    {
        SYS_CMD_PRINT("Format %s success!\r\n", fDrive);
    }
    else
    {
        SYS_CMD_PRINT("Format %s failure!\r\n", fDrive);
    }
}

static const SYS_CMD_DESCRIPTOR MyCmdTbl[] =
{
    { "close", CommandClose, ": Close file" },
    { "media", CommandMedia, ": media %" },
    { "format", CommandFormat, ": Format Media" },
};

void AutoMountEventHandler( SYS_FS_EVENT event, void* eventData, uintptr_t context ) { ... }
```

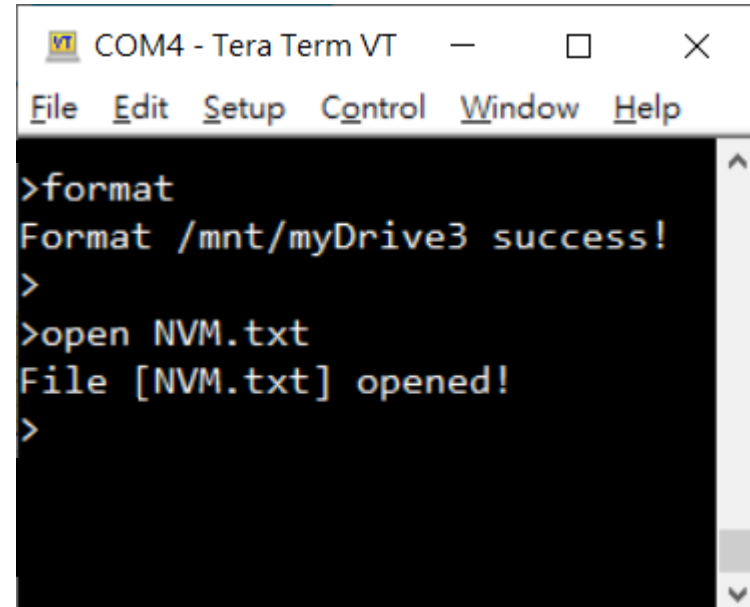
⚙️ Lab10 NVMCTRL Internal Flash with file system

- Remove SD card and USB pen drive firstly.
- Compiler and programming your project to EVB.
- The NVM should **mount automatically** while reboot.
- Command “**media NVM**” to change active media to NVM Flash.
- Command “**open NVM.txt**” try to create a new file in NVM Flash.
- Command “**format**” to format NVM Flash drive to default FAT.
- Try again “**open NVM.txt**” to check the different.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Mount NVM Flash success!
>media NVM
Set media to /dev/nvma1
Set drive to /mnt/myDrive3
>
>open NVM.txt
Open failure!
>
```

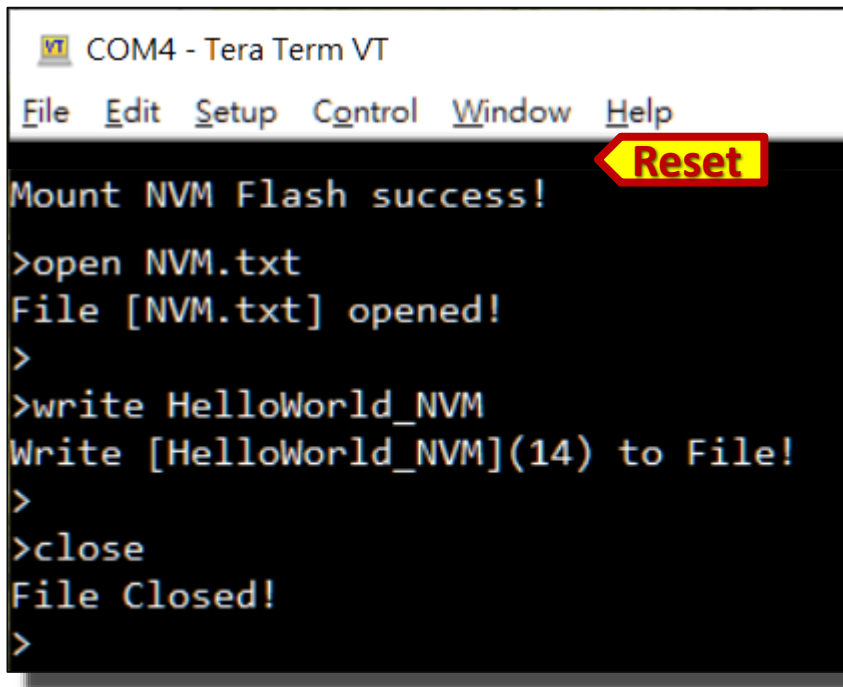
Why? ➔



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
>format
Format /mnt/myDrive3 success!
>
>open NVM.txt
File [NVM.txt] opened!
>
```

⚙️ Lab10 NVMCTRL Internal Flash with file system

- Press the **Reset** button on EVB.
- Try open “**NVM.txt**” and write “**HelloWorld_NVM**” to file in NVM Flash.
- Press the **Reset** button on EVB.
- Try open “**NVM.txt**” and read the content we wrote from file in NVM Flash.

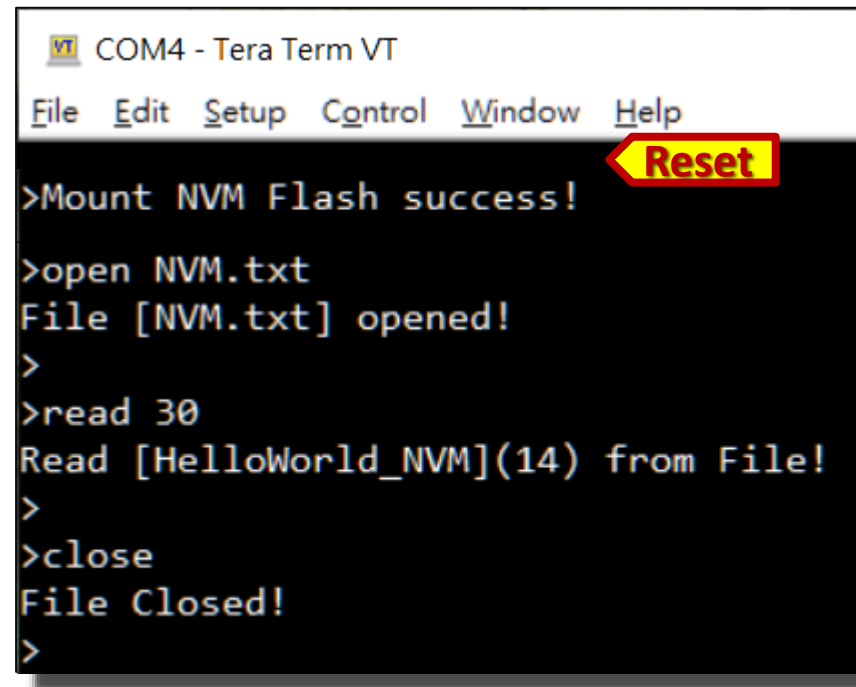


COM4 - Tera Term VT

File Edit Setup Control Window Help

Mount NVM Flash success! **Reset**

```
>open NVM.txt
File [NVM.txt] opened!
>
>write HelloWorld_NVM
Write [HelloWorld_NVM](14) to File!
>
>close
File Closed!
>
```



COM4 - Tera Term VT

File Edit Setup Control Window Help

>Mount NVM Flash success! **Reset**

```
>open NVM.txt
File [NVM.txt] opened!
>
>read 30
Read [HelloWorld_NVM](14) from File!
>
>close
File Closed!
>
```

⚙️ Lab10 NVMMCTRL Internal Flash with file system

- Put in **SD Card** and **USB pen drive** to EVB, the last drive that Auto Mount success to mount is current active media.
- Execute '**media SD**', '**open SD.txt**', '**read 20**', '**close**'
- Execute '**media NVM**', '**open NVM.txt**', '**read 20**', '**close**'
- Execute '**media USB**', '**open USB.txt**', '**read 20**', '**close**'

```
VT COM5 - Tera Term VT
File Edit Setup Control Window Help
Mount NVM Flash success!
Mount SD card success!
Mount USB Mass Storage success!

>media SD
Set media to /dev/mmcblk1
Set drive to /mnt/myDrive1
>open SD.txt
File [SD.txt] opened!
>read 20
Read [HelloWorld_SD](13) from File!
>close
File Closed!
>
```

```
>media NVM
Set media to /dev/nvma1
Set drive to /mnt/myDrive3
>open NVM.txt
File [NVM.txt] opened!
>read 20
Read [HelloWorld_NVM](14) from File!
>close
File Closed!
>
```

```
>media USB
Set media to /dev/sda1
Set drive to /mnt/myDrive2
>open USB.txt
File [USB.txt] opened!
>read 20
Read [HelloWorld_USB](14) from File!
>close
File Closed!
>
```


⚙️ Lab10 NVMCTRL Internal Flash with file system

- Discuss

- How to Format SD Card / USB pen drive ?

The FAT(FAT12) support up to 32MBytes, it's suitable for the NVM Flash drive.

The FAT32 support up to 2TB drive(4GB per file), will proper for a SD Card or USB pen drive.

NVM Flash

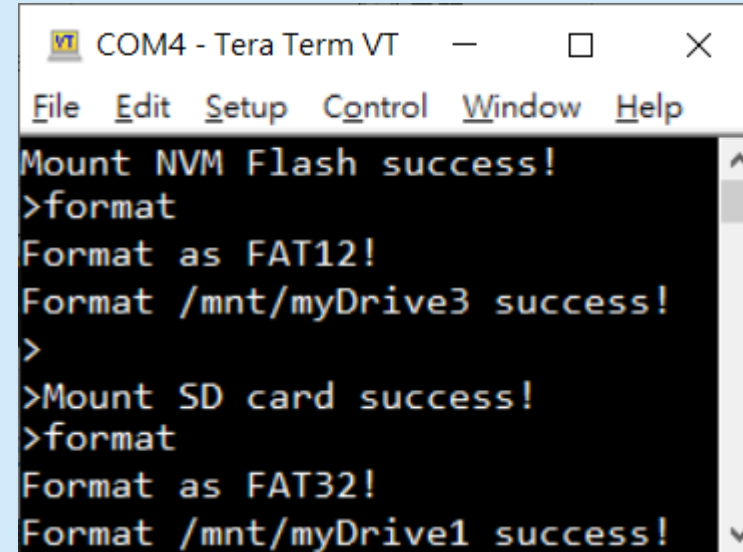
SD/ USB drive

```
void CommandFormat(SYS_CMD_DEVICE_NODE* pCmdIO, int argc, char** argv)
{
    SYS_FS_FORMAT_PARAM opt = { 0 };
    uint8_t CACHE_ALIGN work[512];

    SYS_FS_CurrentDriveGet( fDrive );
    if(!strcmp( fDrive, "/mnt/myDrive3"))
    {
        opt.fmt = SYS_FS_FORMAT_FAT;
        SYS_CMD_PRINT("Format as FAT12!\r\n");
    }
    else
    {
        opt.fmt = SYS_FS_FORMAT_FAT32;
        SYS_CMD_PRINT("Format as FAT32!\r\n");
    }

    opt.au_size = 0;
    opt.n_fat = 1;
    opt.align = 0;
    opt.n_root = 1;

    fResult = SYS_FS_DriveFormat(fDrive, &opt, (void *)work, 512);
    if(fResult == SYS_FS_RES_SUCCESS)
    {
        SYS_CMD_PRINT("Format %s success!\r\n", fDrive);
    }
}
```



COM4 - Tera Term VT

File Edit Setup Control Window Help

Mount NVM Flash success!

>format

Format as FAT12!

Format /mnt/myDrive3 success!

>

>Mount SD card success!

>format

Format as FAT32!

Format /mnt/myDrive1 success!

Harmony Drivers Introduction

Why we to use Driver ?

PLIB Limitation

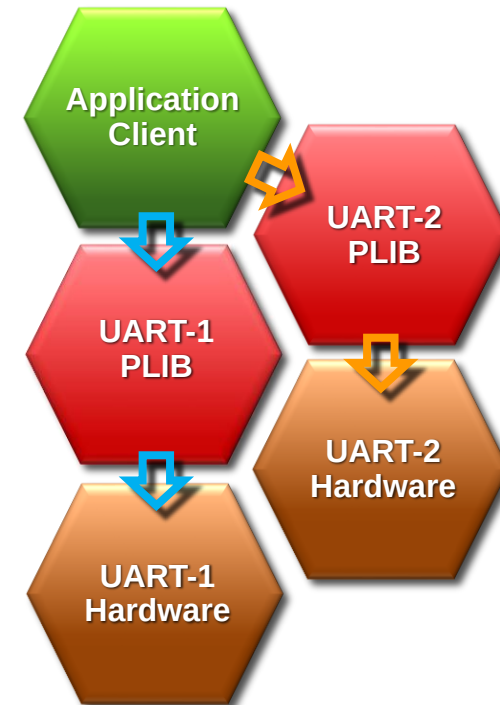
Use the Different Peripheral Instance

```
void my_app(void)
{
    // USART1_Write(pTxBuffer, TX_BYTES);

    // while(USART1_WriteIsBusy() == true);

    USART2_Write(pTxBuffer, TX_BYTES);
    while(USART2_WriteIsBusy() == true);
    ...
}
```

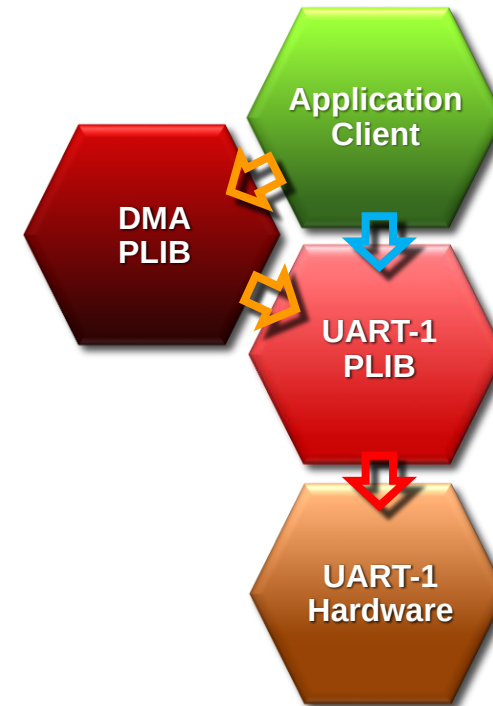
**Application needs to
be modified for different
hardware instance**



PLIB Limitation

Use DMA

```
void USART1_DMA_EventHandler(  
    DMAC_TRANSFER_EVENT event,  
    uintptr_t contextHandle )  
{  
    ...  
}  
  
void my_app(void)  
{  
    // USART1_Write(pTxBuffer, TX_BYTES);  
  
    // while(USART1_WriteIsBusy() == true);  
  
    DMAC_ChannelCallbackRegister(CH_0,  
        USART1_DMA_EventHandler, 0);  
  
    DMAC_ChannelTransfer(CH_0, pTxBuffer,  
        USART1_TRANSMIT_ADDRESS, TX_BYTES);  
    ...  
}
```



**Application needs
to manage DMA !**

PLIB Limitation

Use Multiple Client

```
void my_app(void)
{
    SPI1_TransferSetup(&sensor_spi_setup, 0);

    SENSOR_CS_Clear();

    SPI1_WriteRead (txData, 2, rxData, 5);

    while(SPI1_IsBusy() == true);

    SENSOR_CS_Set();

    SPI1_TransferSetup(&eeprom_spi_setup, 0);

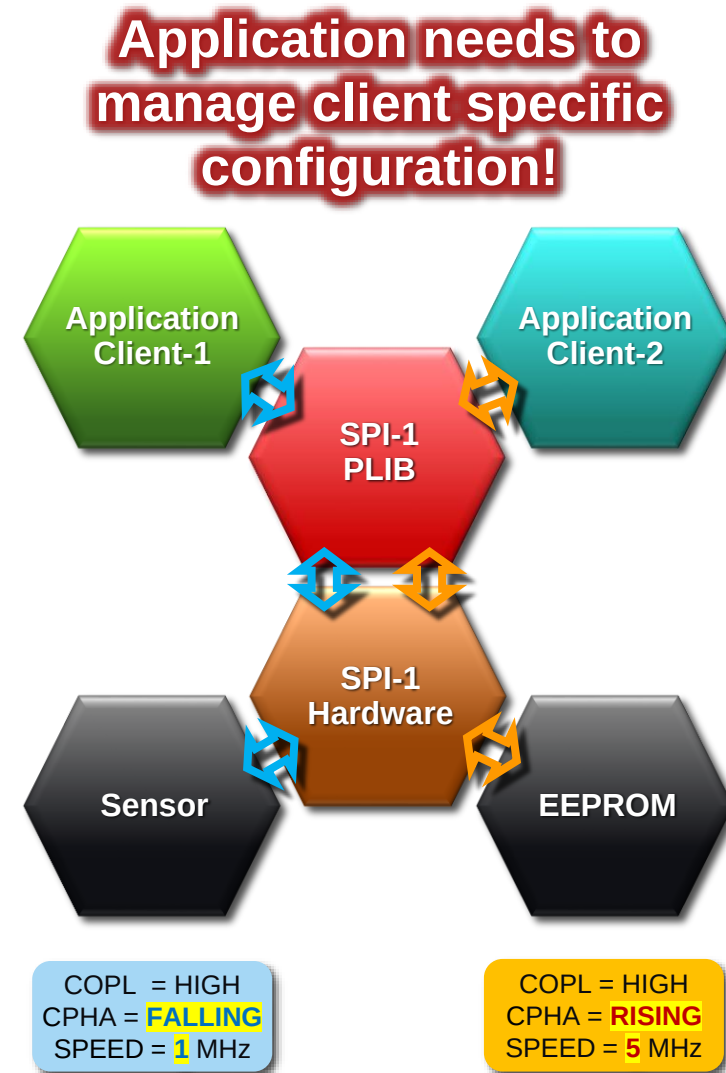
    EEPROM_CS_Clear();

    SPI1_WriteRead (txData, 1, rxData, 3);

    while(SPI1_IsBusy() == true);

    EEPROM_CS_Set();

    ...
}
```



PLIB Limitation

Transfer Request between peripherals

```
void my_app(void)
{
    SPI1_TransferSetup(&sensor_spi_setup, 0);

    SENSOR_CS_Clear();

    SPI1_Write (txdata1, 2);
    while(SPI1_IsBusy() == true);

    EEPROM_CS_Clear();

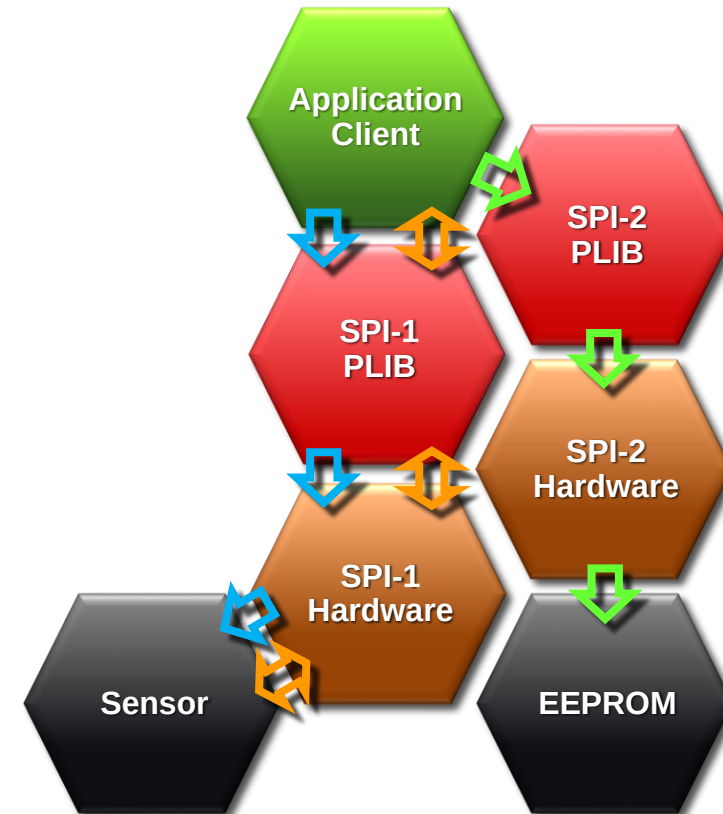
    SPI2_Write (txdata2, 4);
    while(SPI2_IsBusy() == true);

    EEPROM_CS_Set();

    SPI1_WriteRead (txData3, 1, rxData, 8);
    while(SPI1_IsBusy() == true);

    SENSOR_CS_Set();

    ...
}
```



**Application needs to
arrange transfer request!**

PLIB Limitation

Use RTOS

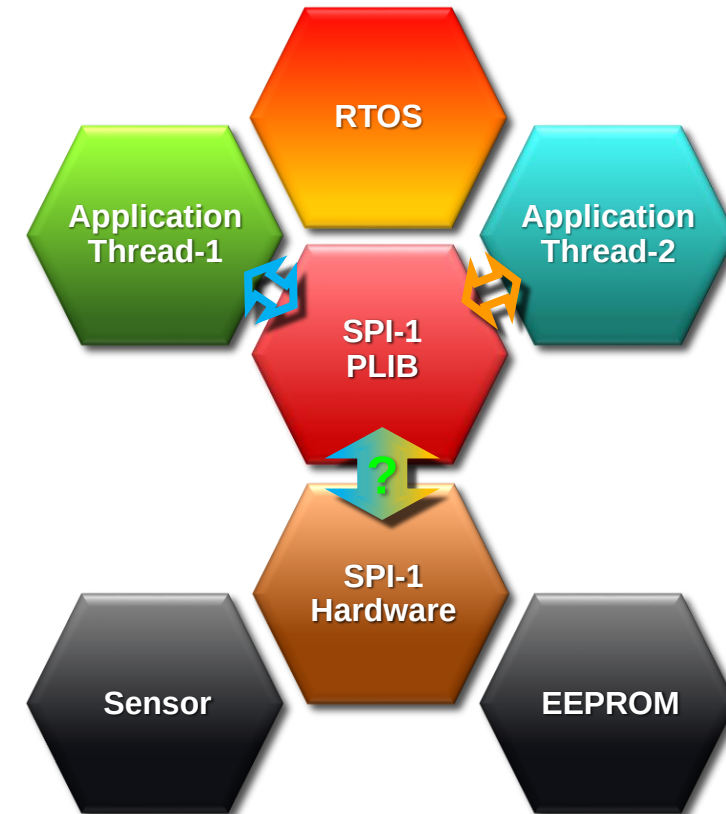
```
void my_app(void)
{
    ThreadCreate( thread_1, "SPI_Task_1" );
    ThreadCreate( thread_2, "SPI_Task_2" );
}

void thread_1(void)
{
    SPI1_TransferSetup(&sensor_spi_setup, 0);

    SENSOR_CS_Clear();
    SPI1_WriteRead (txData, 2, rxData, 5);
    SENSOR_CS_Set();
}

void thread_2(void)
{
    SPI1_TransferSetup(&eeprom_spi_setup, 0);

    EEPROM_CS_Clear();
    SPI1_WriteRead (txData, 1, rxData, 3);
    EEPROM_CS_Set();
}
```



**PLIBs are not Thread Safe
under RTOS!**

Driver Advantage

Harmony drivers provide

- **Same API for all the Instances of a Peripheral**

Application remains same when the peripheral instance is changed

USART Driver API	USART PLIB API
DRV_USART_WriteBufferAdd(...)	USART1_Write(...)
	USART2_Write(...)
DRV_USART_ReadBufferAdd(...)	USART1_Read(...)
	USART2_Read(...)

- **Support for DMA Transfer mode**

Driver interface/API remains same with or without DMA transfer mode

- **Support for Multiple Clients**

Seamlessly handles client specific differences

- **Queue Support**

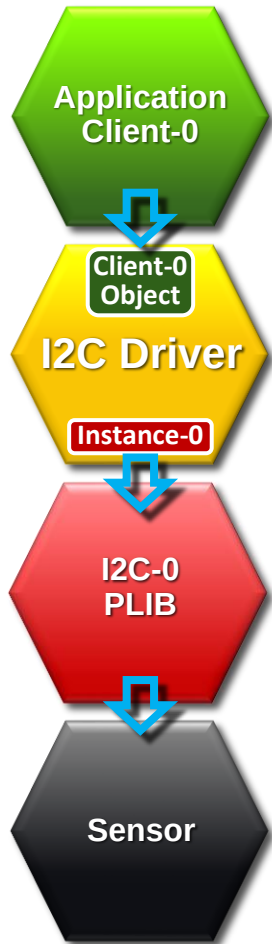
Multiple requests can be queued

- **RTOS Ready**

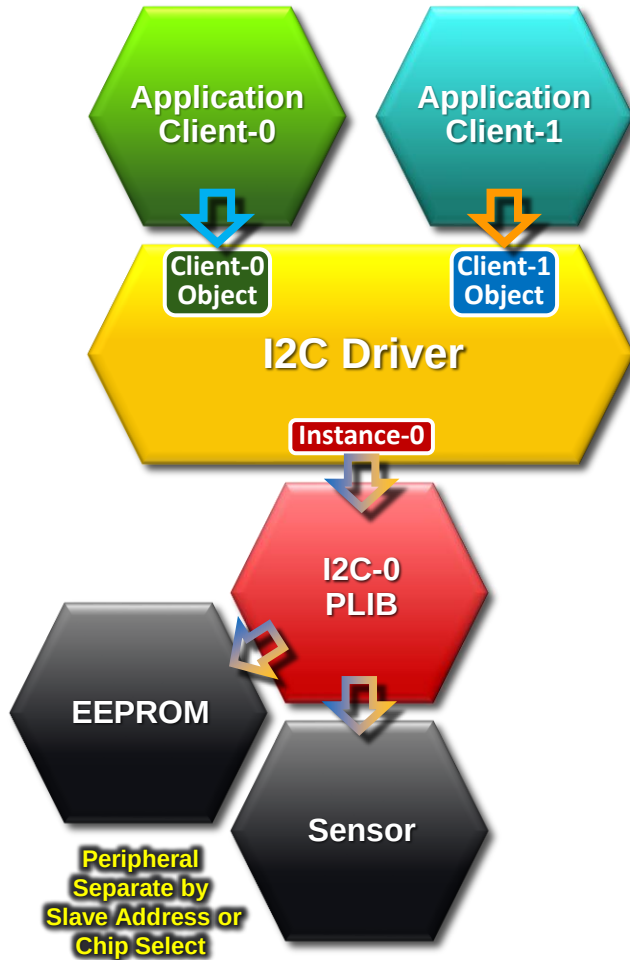
Thread Safe

Driver Usage Models

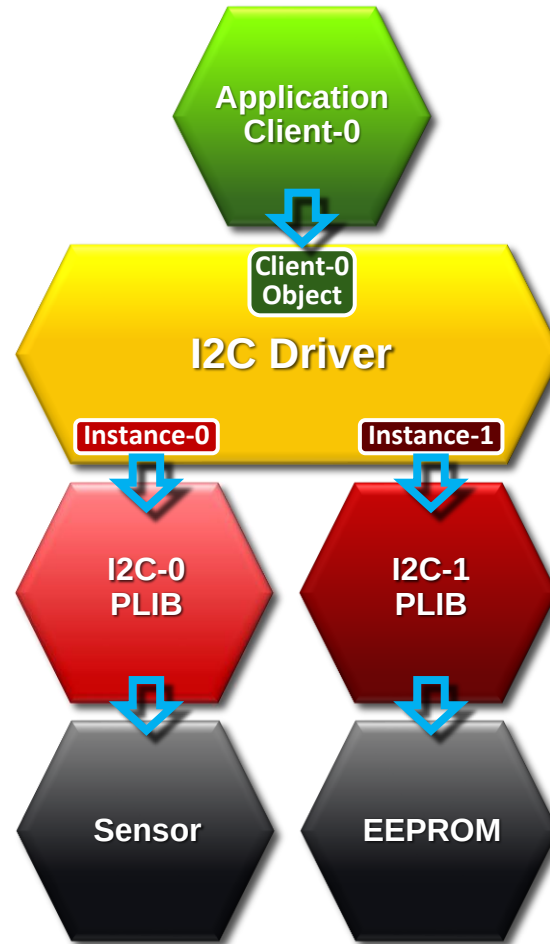
**Single Client
Single Instance**



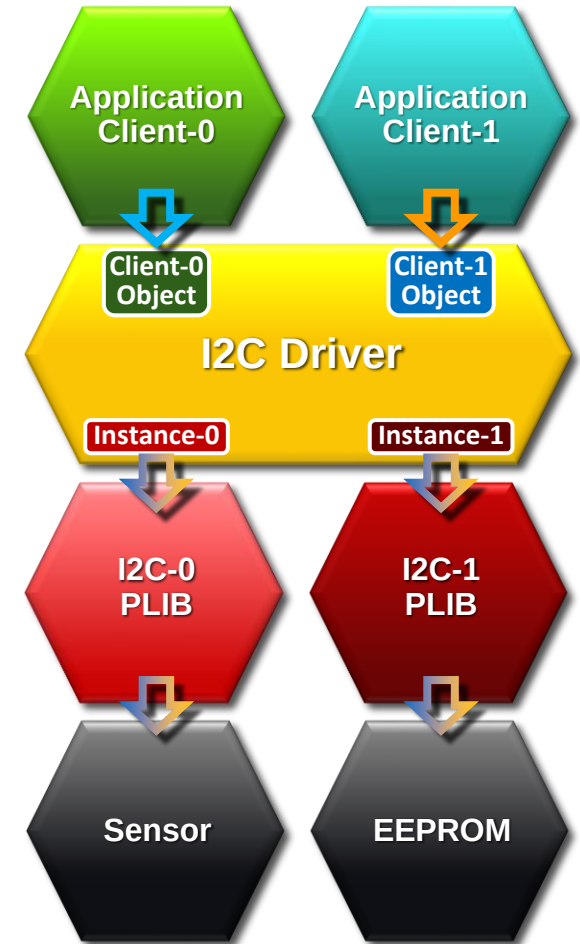
**Multi Client
Single Instance**



**Single Client
Multi Instance**



**Multi Client
Multi Instance**



How to use Driver in Harmony?

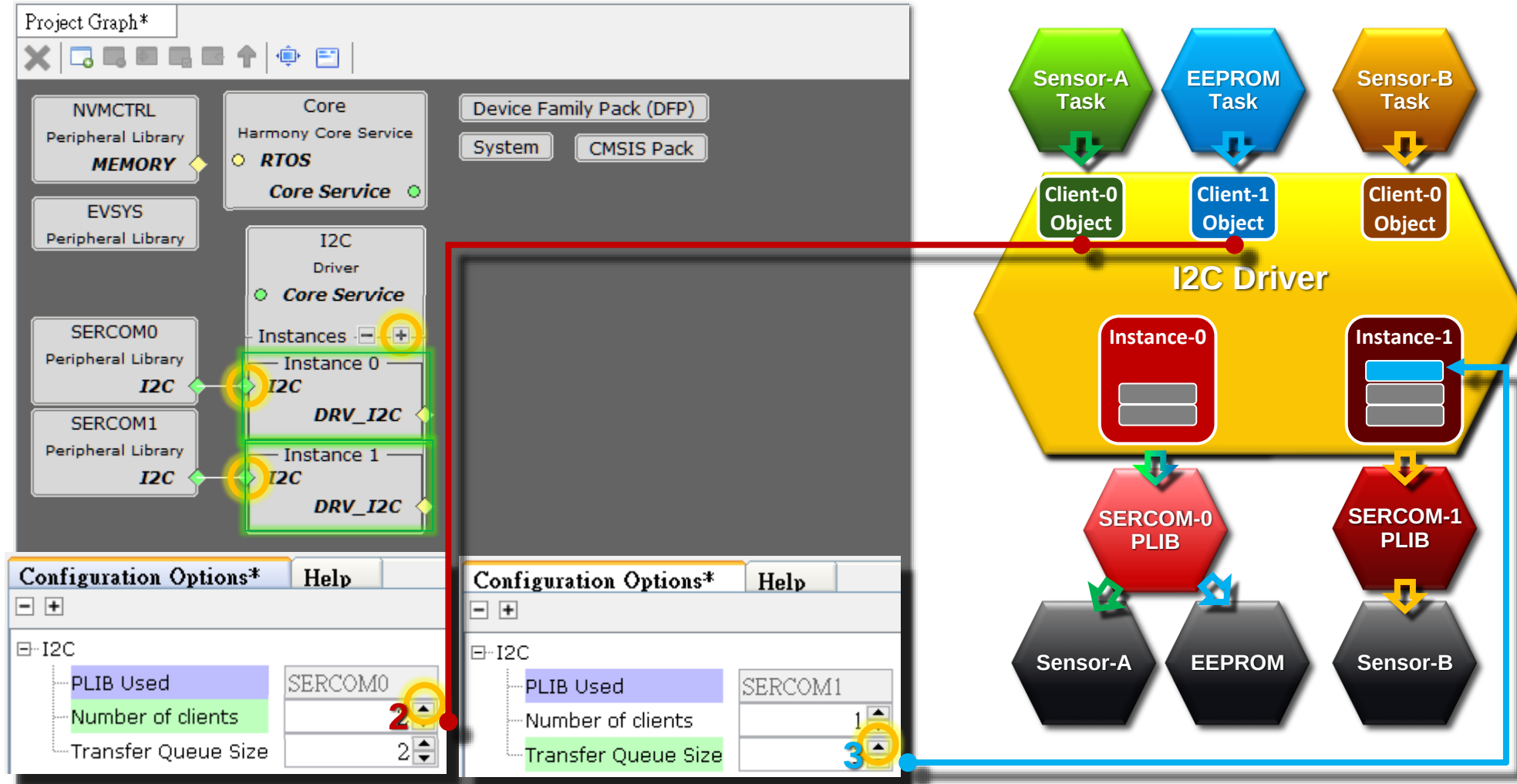
Harmony Drivers Introduction

How to use Driver in Harmony?

- **Add & Configure Using MHC**
- **Initialize Driver**
- **Open Driver and get Client Handle**
- **Register and Define Transfer Callback**
- **Transfer Request with Queue**
- **Transfer Status Polling operation**
- **Transfer Status Callback operation**
- **Two Transfer Request to use one Callback**
- **Two Client to use one Callback**

How to use Driver in Harmony?

Add & Configure Using MHC



How to use Driver in Harmony?

- Add & Configure Using MHC
- **Initialize Driver**
- Open Driver and get Client Handle
- Register and Define Transfer Callback
- Transfer Request with Queue
- Transfer Status Polling operation
- Transfer Status Callback operation
- Two Transfer Request to use one Callback
- Two Client to use one Callback

How to use Driver in Harmony?

Initialize Driver

```
void SYS_Initialize ( void )
{
    ...
    SERCOM0_I2C_Initialize();

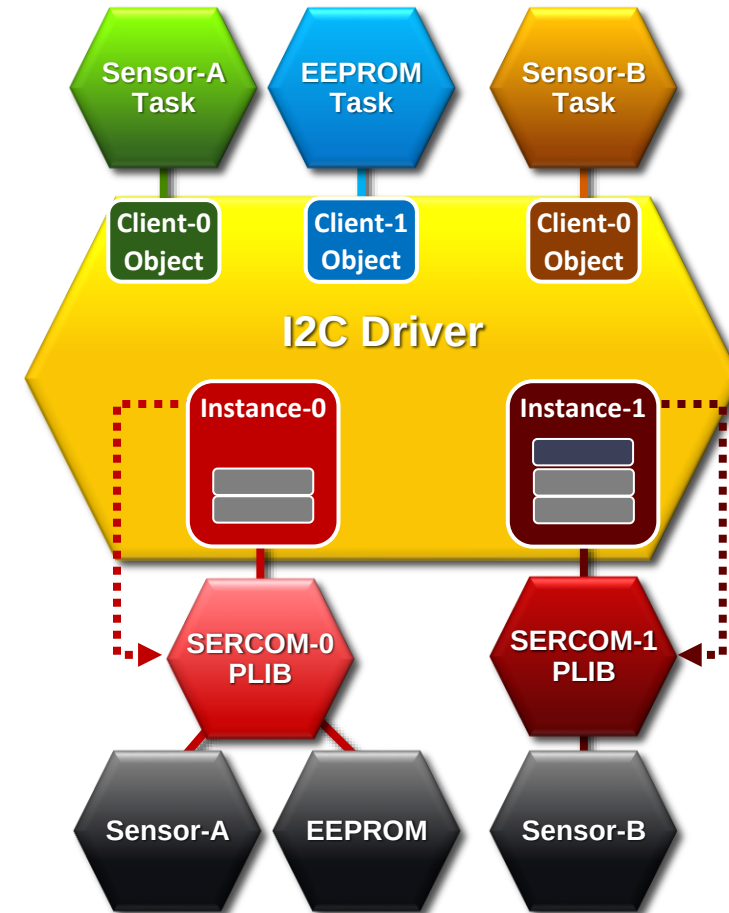
    SERCOM1_I2C_Initialize();

    sysObj.drvI2C0 = DRV_I2C_Initialize(
        DRV_I2C_INDEX_0, Instance-0
        (SYS_MODULE_INIT *)&drvI2C0InitData
    );

    sysObj.drvI2C1 = DRV_I2C_Initialize(
        DRV_I2C_INDEX_1, Instance-1
        (SYS_MODULE_INIT *)&drvI2C1InitData
    );
    ...
}
```

Initializes I2C Driver Instance and associates it with SERCOM PLIB

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



How to use Driver in Harmony?

- Add & Configure Using MHC
- Initialize Driver
- **Open Driver and get Client Handle**
- Register and Define Transfer Callback
- Transfer Request with Queue
- Transfer Status Polling operation
- Transfer Status Callback operation
- Two Transfer Request to use one Callback
- Two Client to use one Callback

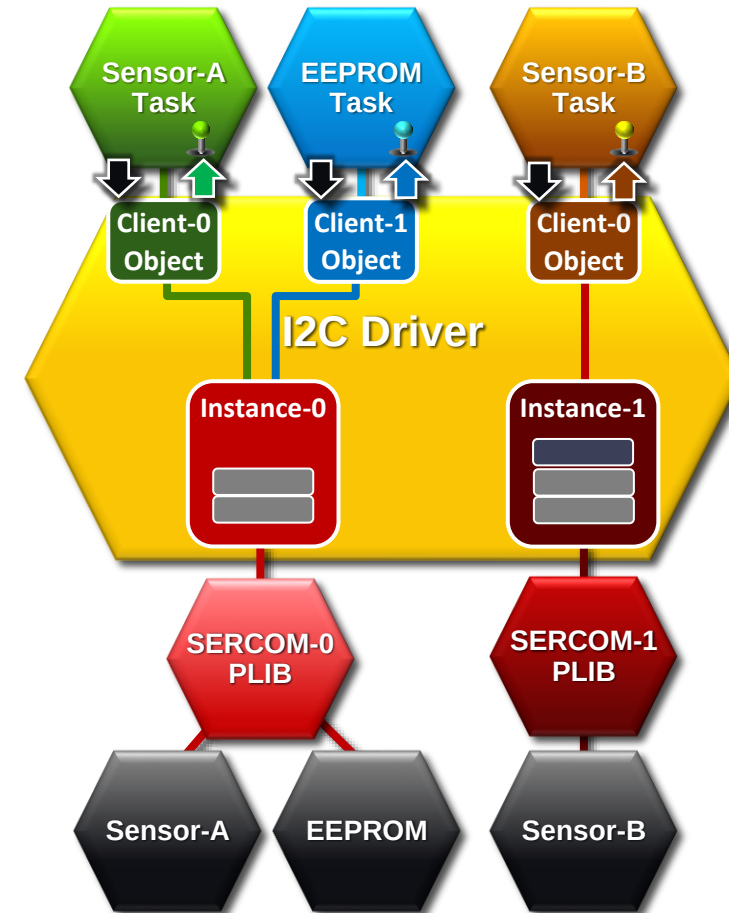
How to use Driver in Harmony?

Open Driver and get Client Handle

```
void sensorA_Task( void )
{
    sensorA_ClientHandle = DRV_I2C_Open (
        DRV_I2C_INDEX_0, Instance-0
        DRV_IO_INTENT_READWRITE
    );
}

void eeprom_Task( void )
{
    eeprom_ClientHandle = DRV_I2C_Open (
        DRV_I2C_INDEX_0, Instance-0
        DRV_IO_INTENT_READWRITE
    );
}

void sensorB_Task( void )
{
    sensorB_ClientHandle = DRV_I2C_Open (
        DRV_I2C_INDEX_1, Instance-1
        DRV_IO_INTENT_READ
    );
}
```



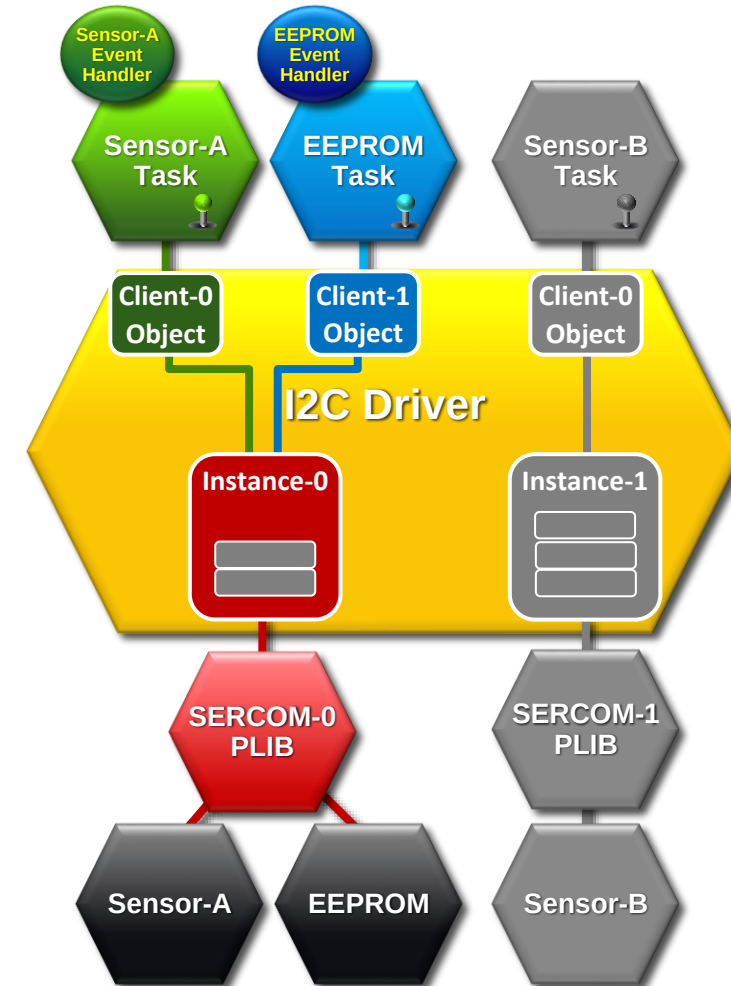
How to use Driver in Harmony?

- Add & Configure Using MHC
- Initialize Driver
- Open Driver and get Client Handle
- **Register and Define Transfer Callback**
- Transfer Request with Queue
- Transfer Status Polling operation
- Transfer Status Callback operation
- Two Transfer Request to use one Callback
- Two Client to use one Callback

How to use Driver in Harmony?

Register and Define Transfer Callback

```
void sensorA_EventHandler(...)  
{  
    sensorA_ReqCompleted = true;  
}  
  
void eeeprom_EventHandler(...)  
{  
    eeeprom_ReqCompleted = true;  
}  
  
void sensorA_Task( void )  
{  
    DRV_I2C_TransferEventHandlerSet (  
        sensorA_ClientHandle,  
        sensorA_EventHandler, 0 );  
}  
  
void eeeprom_Task( void )  
{  
    DRV_I2C_TransferEventHandlerSet (  
        eeeprom_ClientHandle,  
        eeeprom_EventHandler, 0 );  
}
```



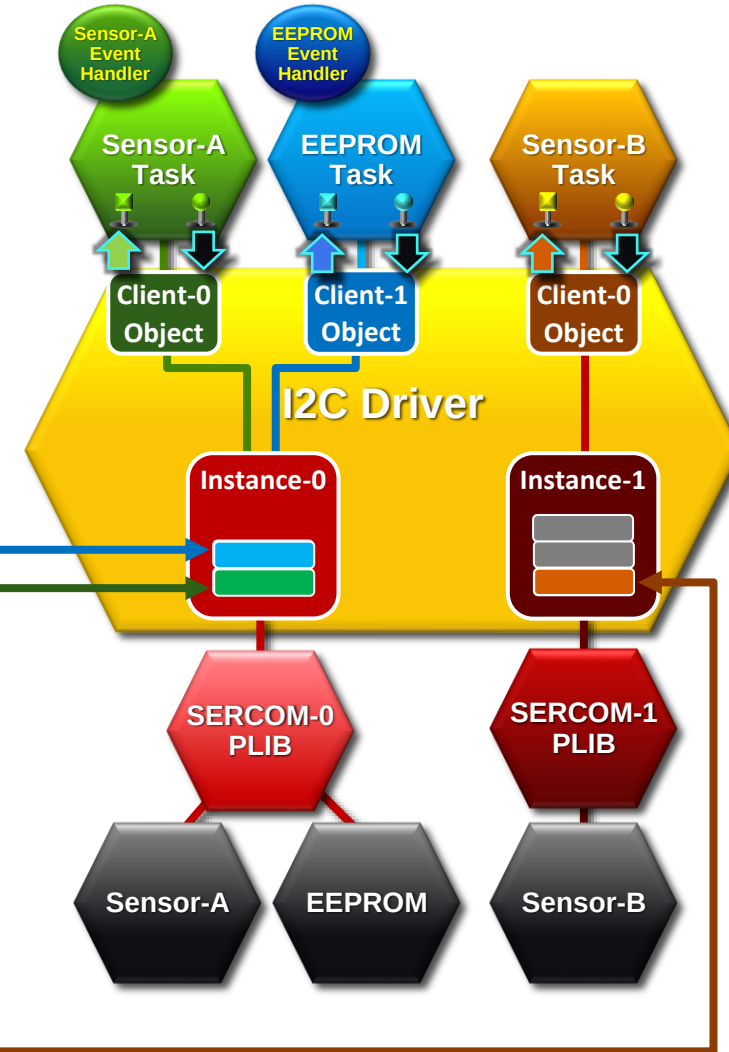
How to use Driver in Harmony?

- Add & Configure Using MHC
- Initialize Driver
- Open Driver and get Client Handle
- Register and Define Transfer Callback
- **Transfer Request with Queue**
- Transfer Status Polling operation
- Transfer Status Callback operation
- Two Transfer Request to use one Callback
- Two Client to use one Callback

How to use Driver in Harmony?

Transfer Request with Queue

```
DRV_I2C_WriteReadTransferAdd (  
    sensorA_ClientHandle,   
    SENSOR_A_SLAVE_ADDR,   
    txBuffer, nBytes,   
    rxBuffer, nBytes,   
    &sensorA_TransferHandle   
);  
  
DRV_I2C_WriteTransferAdd (  
    eeprom_ClientHandle,   
    EEPROM_SLAVE_ADDR,   
    txBuffer,   
    nBytes,   
    &eeprom_TransferHandle   
);  
  
DRV_I2C_ReadTransferAdd (  
    sensorB_ClientHandle,   
    SENSOR_B_SLAVE_ADDR,   
    rxBuffer,   
    nBytes,   
    &sensorB_TransferHandle   
);
```



How to use Driver in Harmony?

- Add & Configure Using MHC
- Initialize Driver
- Open Driver and get Client Handle
- Register and Define Transfer Callback
- Transfer Request with Queue
- **Transfer Status Polling operation**
- Transfer Status Callback operation
- Two Transfer Request to use one Callback
- Two Client to use one Callback

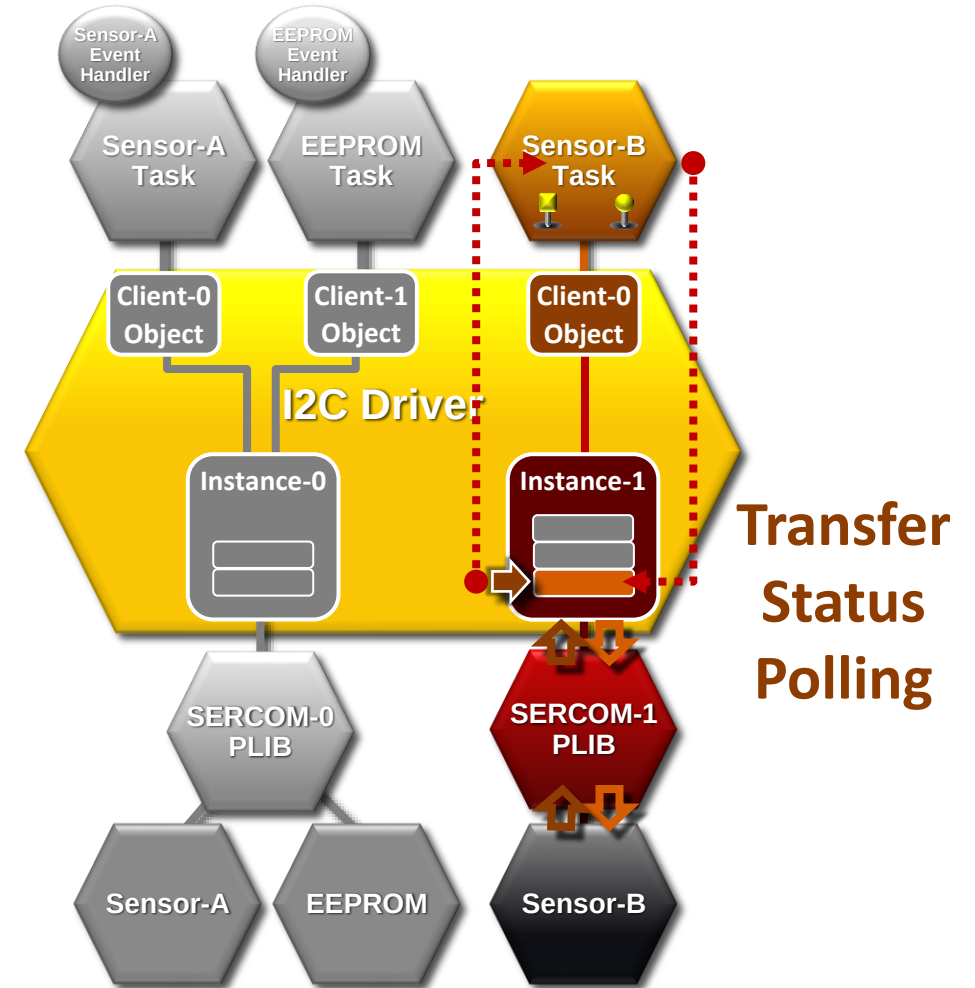
How to use Driver in Harmony?

Transfer Status Polling operation

```
void sensorB_Task( void )
{
    sensorB_ClientHandle = DRV_I2C_Open (
        DRV_I2C_INDEX_1, Instance-1,
        DRV_IO_INTENT_READ
    );

    DRV_I2C_ReadTransferAdd (
        sensorB_ClientHandle,
        SENSOR_B_SLAVE_ADDR,
        rxBuffer,
        nBytes,
        &sensorB_TransferHandle

    while( 1 ) {
        if( DRV_I2C_TransferStatusGet
            ( sensorB_TransferHandle )
            == DRV_I2C_TRANSFER_EVENT_COMPLETE )
        {
            // Do something after read Sensor-B
        }
    }
}
```



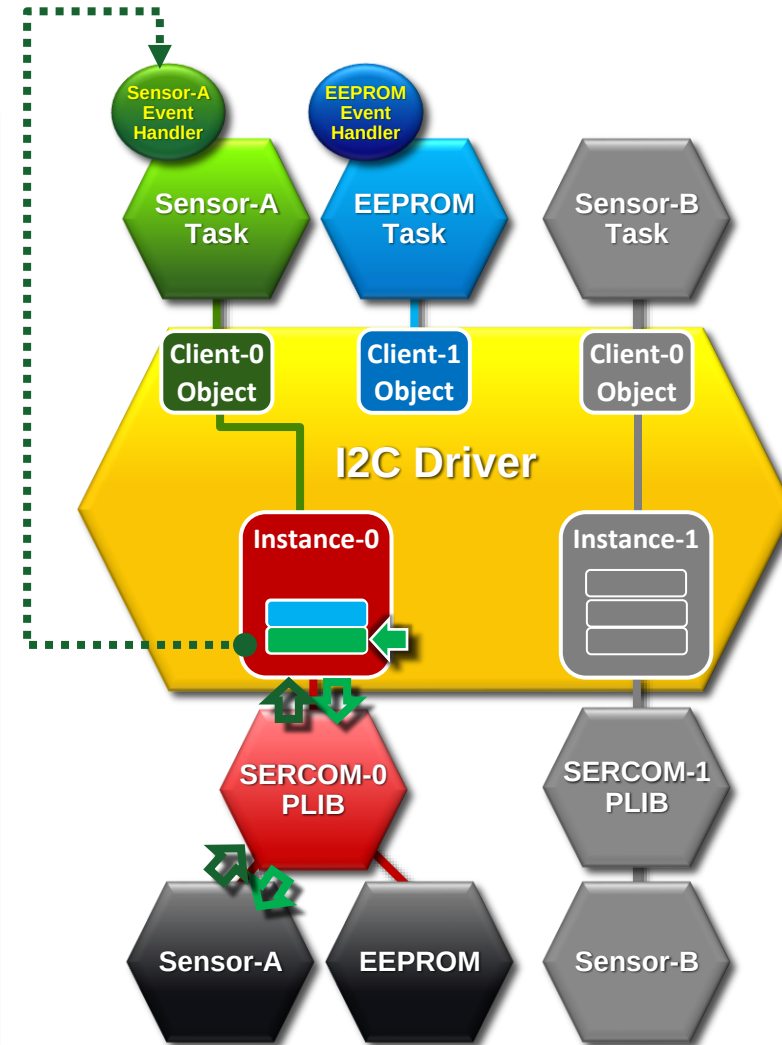
How to use Driver in Harmony?

- Add & Configure Using MHC
- Initialize Driver
- Open Driver and get Client Handle
- Register and Define Transfer Callback
- Transfer Request with Queue
- Transfer Status Polling operation
- **Transfer Status Callback operation**
- Two Transfer Request to use one Callback
- Two Client to use one Callback

How to use Driver in Harmony?

Transfer Status Callback operation

```
void sensorA_EventHandler(...)  
{  
    sensorA_ReqCompleted = true;  
}  
  
void eeprom_EventHandler(...)  
{  
    eeprom_ReqCompleted = true;  
}  
  
void sensorA_Task ( void )  
{  
    if( sensorA_ReqCompleted ) {  
        sensorA_ReqCompleted = false; ←  
    }  
}  
  
void eeprom_Task ( void )  
{  
    if( eeprom_ReqCompleted ) {  
        eeprom_ReqCompleted = false;  
    }  
}
```



How to use Driver in Harmony?

Transfer Status Callback operation

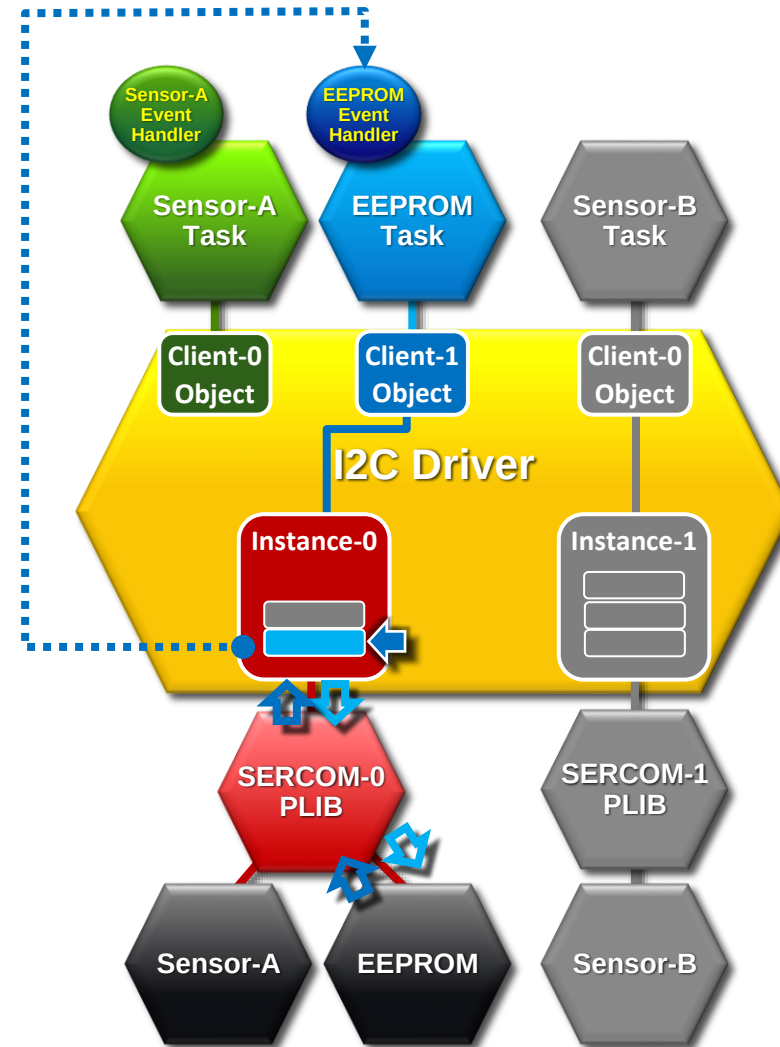
```
void sensorA_EventHandler(...)
{
    sensorA_ReqCompleted = true;
}

void eeprom_EventHandler(...)
{
    eeprom_ReqCompleted = true;
}

void sensorA_Task ( void )
{
    if( sensorA_ReqCompleted ) {
        sensorA_ReqCompleted = false;
    }
}

void eeprom_Task ( void )
{
    if( eeprom_ReqCompleted ) {
        eeprom_ReqCompleted = false;
    }
}
```

EEPROM
Event
Handler



How to use Driver in Harmony?

- Add & Configure Using MHC
- Initialize Driver
- Open Driver and get Client Handle
- Register and Define Transfer Callback
- Transfer Request with Queue
- Transfer Status Polling operation
- Transfer Status Callback operation
- **Two Transfer Request to use one Callback**
- Two Client to use one Callback

How to use Driver in Harmony?

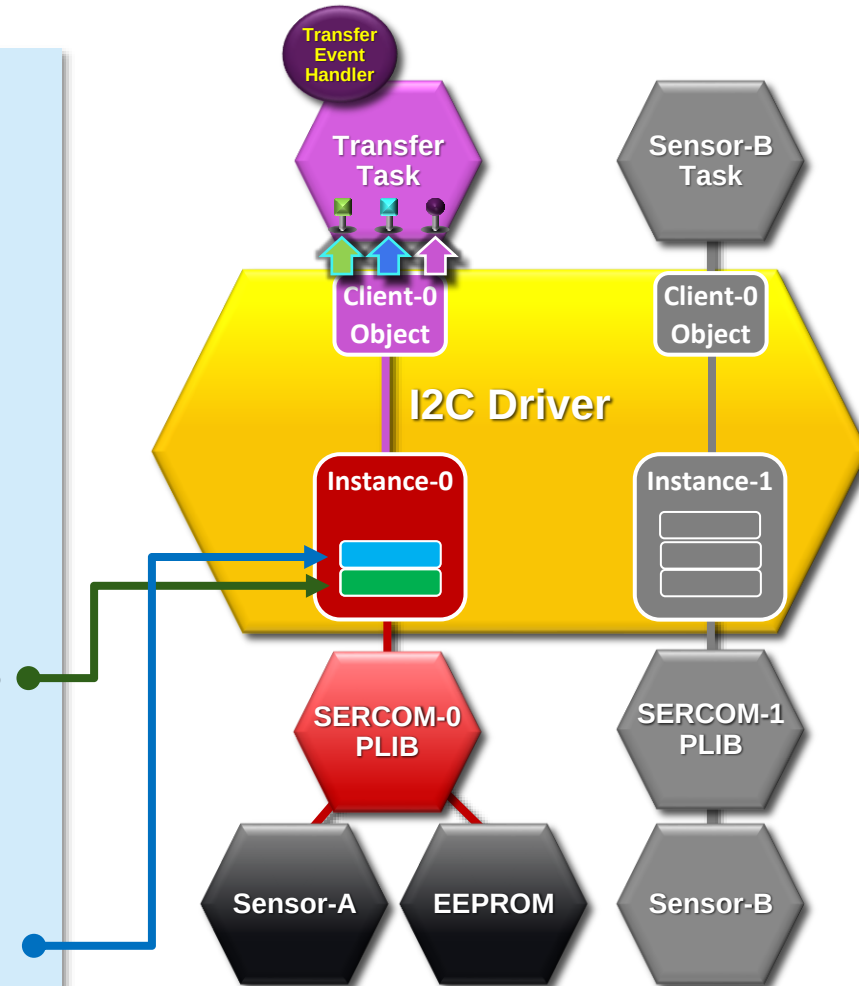
Two Transfer Request to use one Callback

```
void transfer_Task( void )
{
    transfer_ClientHandle = DRV_I2C_Open (
        DRV_I2C_INDEX_0, Instance-0,
        DRV_IO_INTENT_READWRITE
    );

    DRV_I2C_TransferEventHandlerSet (
        transfer_ClientHandle,
        transfer_EventHandler, 0 );

    DRV_I2C_ReadTransferAdd (
        transfer_ClientHandle,
        SENSOR_A_SLAVE_ADDR, rxBuffer, nBytes,
        &sensorA_TransferHandle );

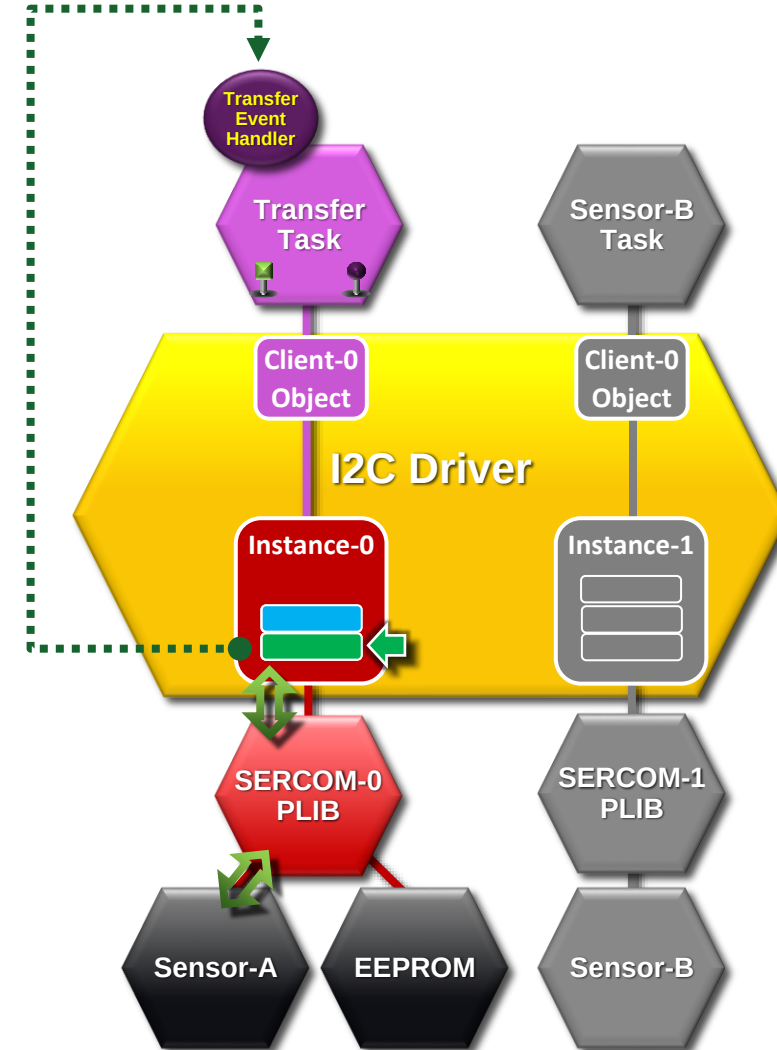
    DRV_I2C_WriteTransferAdd (
        transfer_ClientHandle,
        EEPROM_SLAVE_ADDR, txBuffer, nBytes,
        &eeprom_TransferHandle );
}
```



How to use Driver in Harmony?

Two Transfer Request to use one Callback

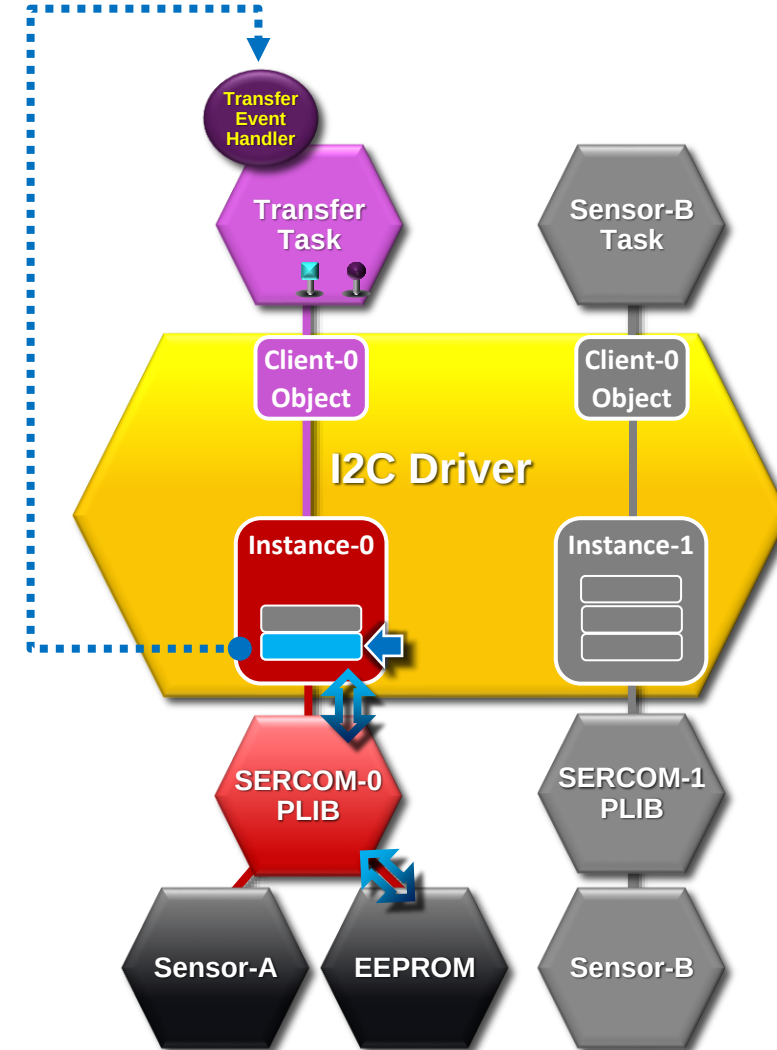
```
void transfer_EventHandler(Transfer  
Event  
Handler  
    DRV_I2C_TRANSFER_EVENT event,  
    DRV_I2C_TRANSFER_HANDLE transferHandle,  
    uintptr_t context )  
{  
    if(event == DRV_I2C_TRANSFER_EVENT_COMPLETE)  
    {  
        if(transferHandle == sensorA_TransferHandle)  
            sensorA_ReqCompleted = true;  
  
        if(transferHandle == eeprom_TransferHandle )  
            eeprom_ReqCompleted = true;  
    }  
}  
  
void transfer_Task ( void )  
{  
    if( sensorA_ReqCompleted ) {  
        sensorA_ReqCompleted = false; ←  
    }  
    if( eeprom_ReqCompleted ) {  
        eeprom_ReqCompleted = false;  
    }  
}
```



How to use Driver in Harmony?

Two Transfer Request to use one Callback

```
void transfer_EventHandler(Transfer  
Event  
Handler  
    DRV_I2C_TRANSFER_EVENT event,  
    DRV_I2C_TRANSFER_HANDLE transferHandle,  
    uintptr_t context )  
{  
    if(event == DRV_I2C_TRANSFER_EVENT_COMPLETE)  
    {  
        if(transferHandle == sensorA_TransferHandle)  
            sensorA_ReqCompleted = true;  
  
        if(transferHandle == eeeprom_TransferHandle )  
            eeeprom_ReqCompleted = true;  
    }  
}  
  
void transfer_Task ( void )  
{  
    if( sensorA_ReqCompleted ) {  
        sensorA_ReqCompleted = false;  
    }  
    if( eeeprom_ReqCompleted ) {  
        eeeprom_ReqCompleted = false; ←  
    }  
}
```



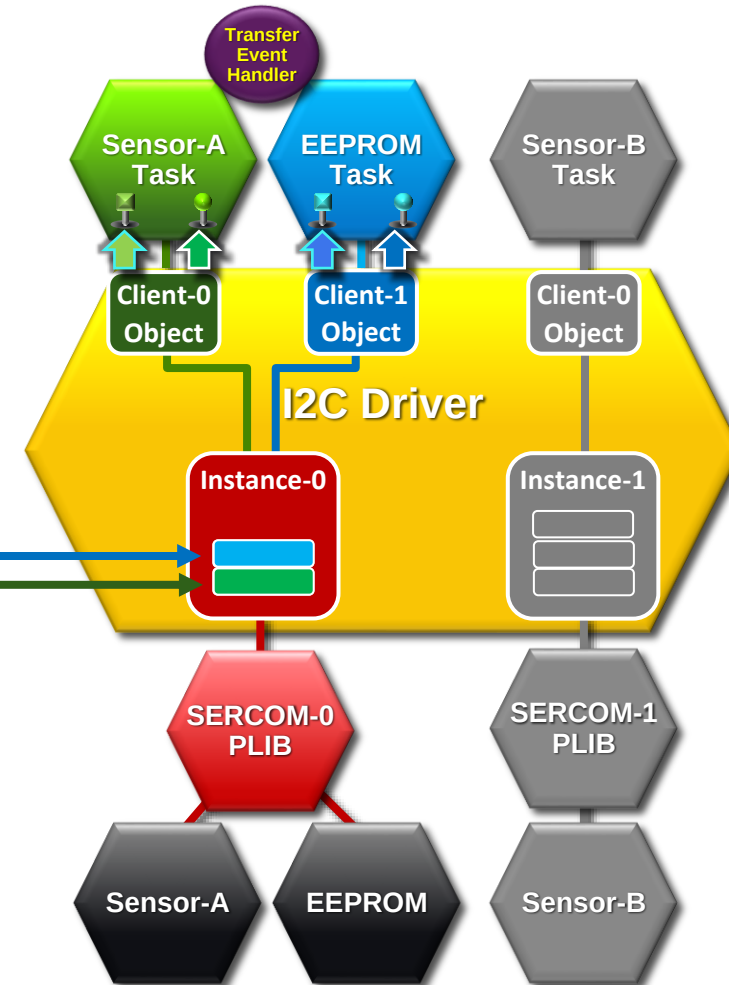
How to use Driver in Harmony?

- Add & Configure Using MHC
- Initialize Driver
- Open Driver and get Client Handle
- Register and Define Transfer Callback
- Transfer Request with Queue
- Transfer Status Polling operation
- Transfer Status Callback operation
- Two Transfer Request to use one Callback
- Two Client to use one Callback

How to use Driver in Harmony?

Two Client to use one Callback

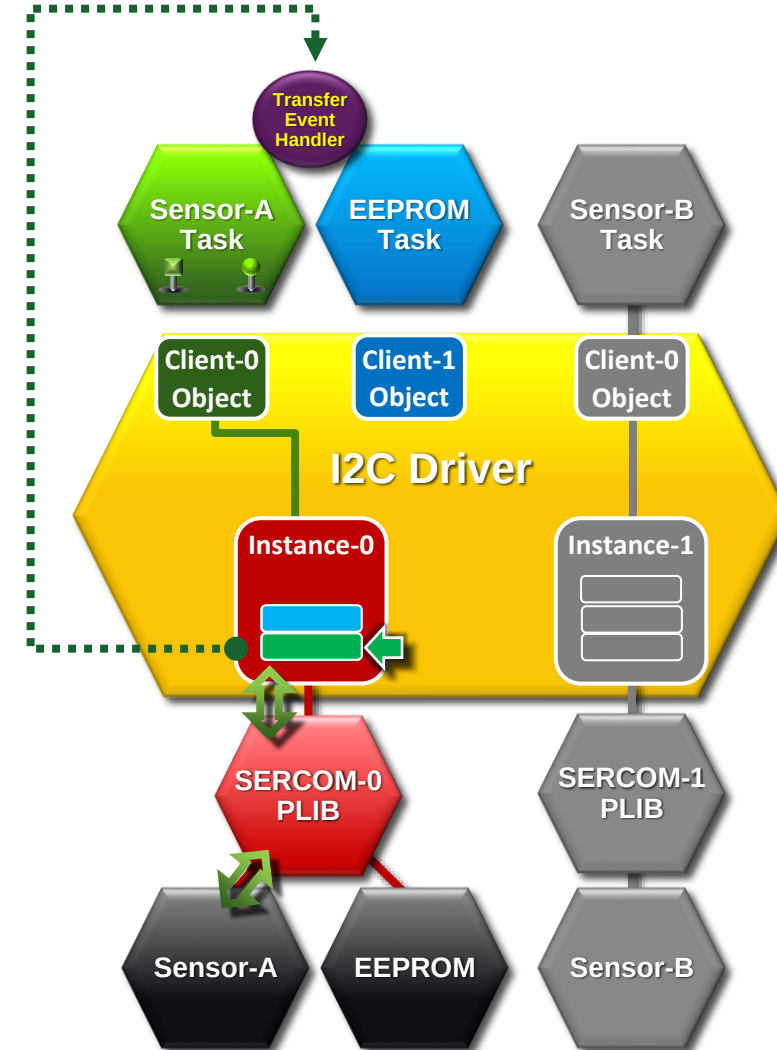
```
void sensorA_Task( void ) {  
    📌 sensorA_ClientHandle = DRV_I2C_Open (   
        DRV_I2C_INDEX_0, Instance-0  
        DRV_IO_INTENT_READWRITE );  
    DRV_I2C_TransferEventHandlerSet (   
        📌 sensorA_ClientHandle,  
        transfer_EventHandler, 0 );  
    DRV_I2C_ReadTransferAdd (   
        📌 sensorA_ClientHandle, ...,  
        &sensorA_TransferHandle 📌 );  
}  
  
void eeprom_Task( void ) {  
    📌 eeprom_ClientHandle = DRV_I2C_Open (   
        DRV_I2C_INDEX_0, Instance-0  
        DRV_IO_INTENT_READWRITE );  
    DRV_I2C_TransferEventHandlerSet (   
        📌 eeprom_ClientHandle,  
        transfer_EventHandler, 0 );  
    DRV_I2C_WriteTransferAdd (   
        📌 eeprom_ClientHandle, ...,  
        &eeprom_TransferHandle 📌 );  
}
```



How to use Driver in Harmony?

Two Client to use one Callback

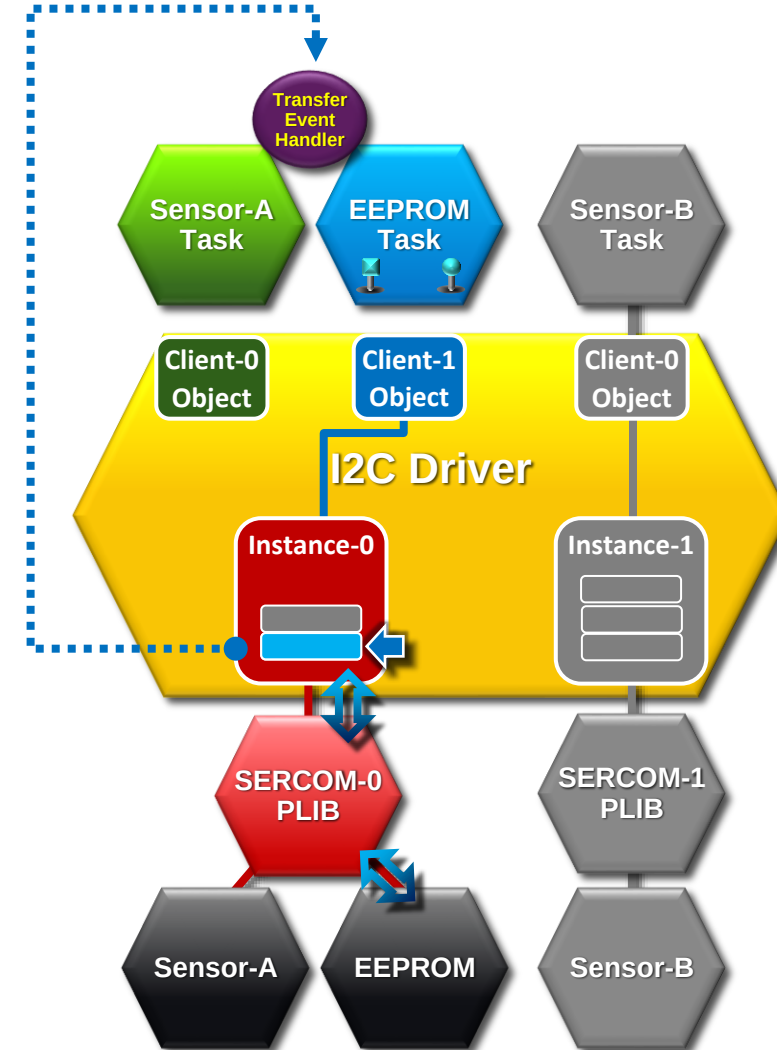
```
void transfer_EventHandler(Transfer  
Event  
Handler  
    DRV_I2C_TRANSFER_EVENT event,  
    DRV_I2C_TRANSFER_HANDLE transferHandle,  
    uintptr_t context )  
{  
    if(event == DRV_I2C_TRANSFER_EVENT_COMPLETE)  
    {  
        if(transferHandle == sensorA_TransferHandle)  
            sensorA_ReqCompleted = true;  
  
        if(transferHandle == eeprom_TransferHandle )  
            eeprom_ReqCompleted = true;  
    }  
}  
  
void transfer_Task ( void )  
{  
    if( sensorA_ReqCompleted ) {  
        sensorA_ReqCompleted = false; ←  
    }  
    if( eeprom_ReqCompleted ) {  
        eeprom_ReqCompleted = false;  
    }  
}
```



How to use Driver in Harmony?

Two Client to use one Callback

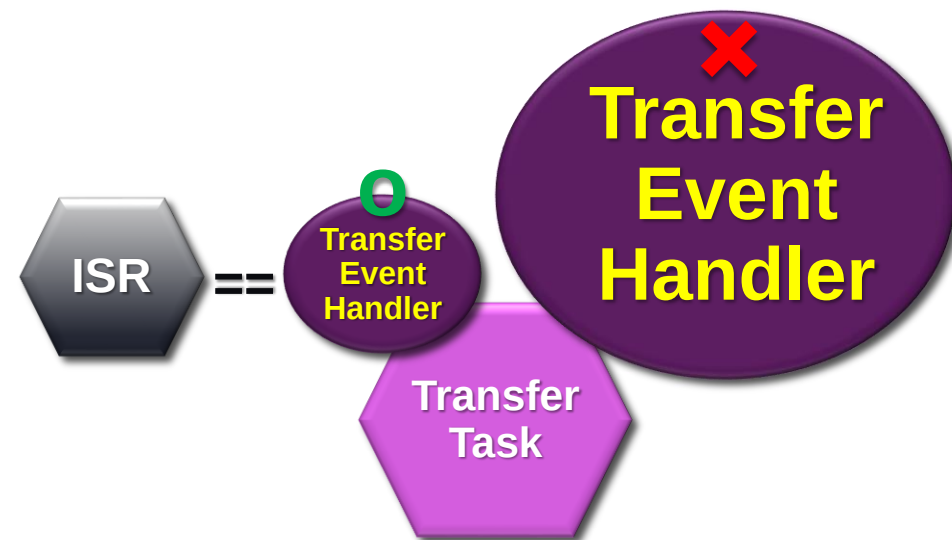
```
void transfer_EventHandler(Transfer  
Event  
Handler  
    DRV_I2C_TRANSFER_EVENT event,  
    DRV_I2C_TRANSFER_HANDLE transferHandle,  
    uintptr_t context )  
{  
    if(event == DRV_I2C_TRANSFER_EVENT_COMPLETE)  
    {  
        if(transferHandle == sensorA_TransferHandle)  
            sensorA_ReqCompleted = true;  
  
        if(transferHandle == eeeprom_TransferHandle )  
            eeeprom_ReqCompleted = true;  
    }  
}  
  
void transfer_Task ( void )  
{  
    if( sensorA_ReqCompleted ) {  
        sensorA_ReqCompleted = false;  
    }  
    if( eeeprom_ReqCompleted ) {  
        eeeprom_ReqCompleted = false; ←  
    }  
}
```



Transfer Request Callback Restriction

- Must be treated like an ISR.
- Must be short.
- Must not call application functions that are not interrupt safe.
- Must not call other driver's interface functions.
- Use volatile keyword.

```
✗ int TransferDone = false;  
○ volatile int TransferFinished = false;  
void transfer_EventHandler( ... )  
{  
    ✗ TransferDone = true;  
    ○ TransferFinished = true;  
  
    ✗ DRV_ReadTransferAdd( ... );  
    ✗ APP_InterruptNotSafe( ... );  
}
```



I2C Driver API and Usage Example

```
void sensorA_Task( void )  
{  
    DRV_I2C_Open(...);  
    DRV_I2C_WriteTransferAdd(...);  
  
    while(1)  
    {  
        if( DRV_I2C_TransferStatusGet(...)   
            == DRV_I2C_TRANSFER_EVENT_COMPLETE )  
        {  
            break;  
        }  
    }  
  
    DRV_I2C_Close(...);  
}
```

Polling method

```
void sensorEventHandler(...)  
{  
    transferDone = true;  
}  
  
void sensorA_Task( void )  
{  
    DRV_I2C_Open(...);  
    DRV_I2C_TransferEventHandlerSet(..., sensorEventHandler, ...);  
    DRV_I2C_WriteTransferAdd(...);  
    while(1)  
    {  
        if( transferDone ) break;  
    }  
    DRV_I2C_Close(...);  
}
```

Callback method

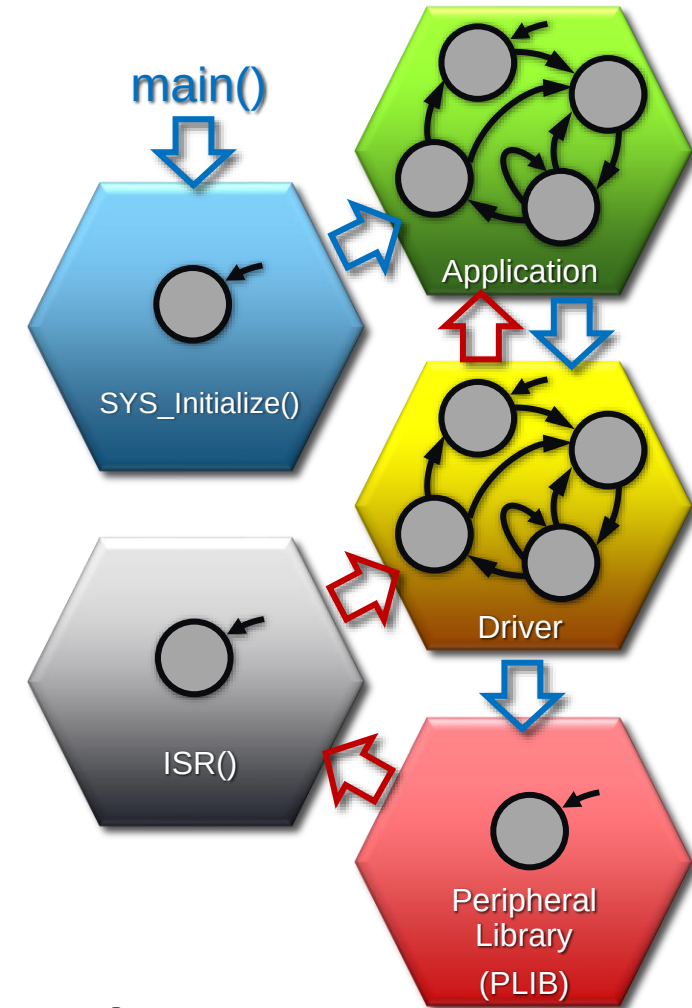
I2C Driver Client API list:

- | | |
|-----------------------------------|--|
| • DRV_I2C_Open | - Open I2C Driver |
| • DRV_I2C_Close | - Close I2C Driver |
| • DRV_I2C_ReadTransferAdd | - Add a I2C read request |
| • DRV_I2C_WriteTransferAdd | - Add a I2C write request |
| • DRV_I2C_WriteReadTransferAdd | - Add a I2C write followed by read request |
| • DRV_I2C_TransferStatusGet | - Get transfer status |
| • DRV_I2C_TransferEventHandlerSet | - Register Callback with the driver |

Driver Execution Model

Let's review our study :

- **main() start**
- **System Initializes Driver and PLIB**
- **Application Tasks start**
- **Application Calls Driver APIs**
(Open, Register Callback, Transfer Request)
- **Driver Starts Operation**
- **PLIB execute Transfer Request**
- **Interrupt Occurs or Polling Complete**
- **Runs Driver State Machine**
- **Driver finishes Operation and Notifies the Application**



USART Driver Library

Harmony Driver

USART Driver Library

- **Introduction**

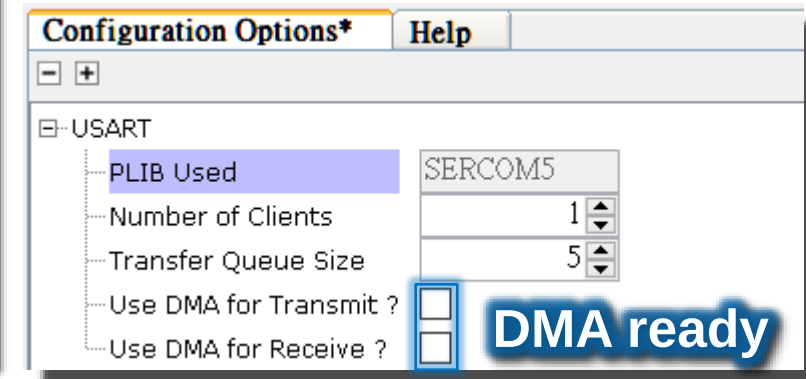
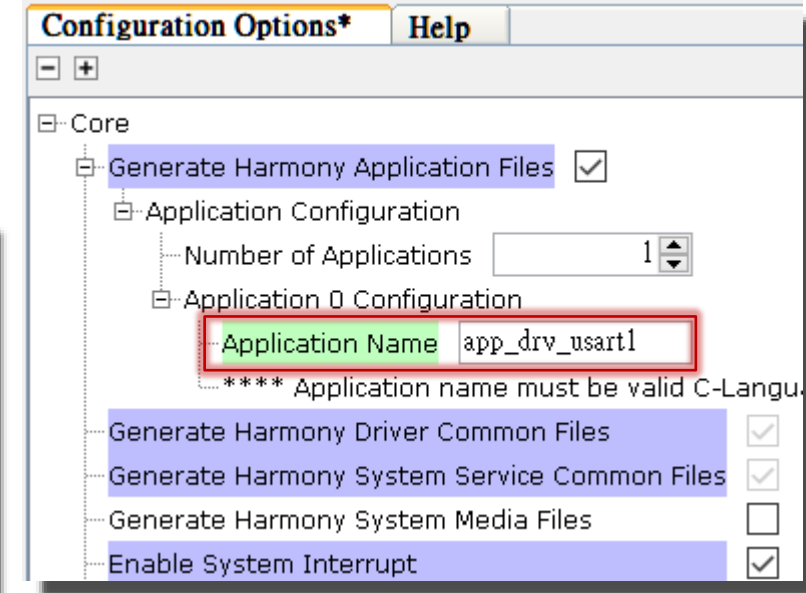
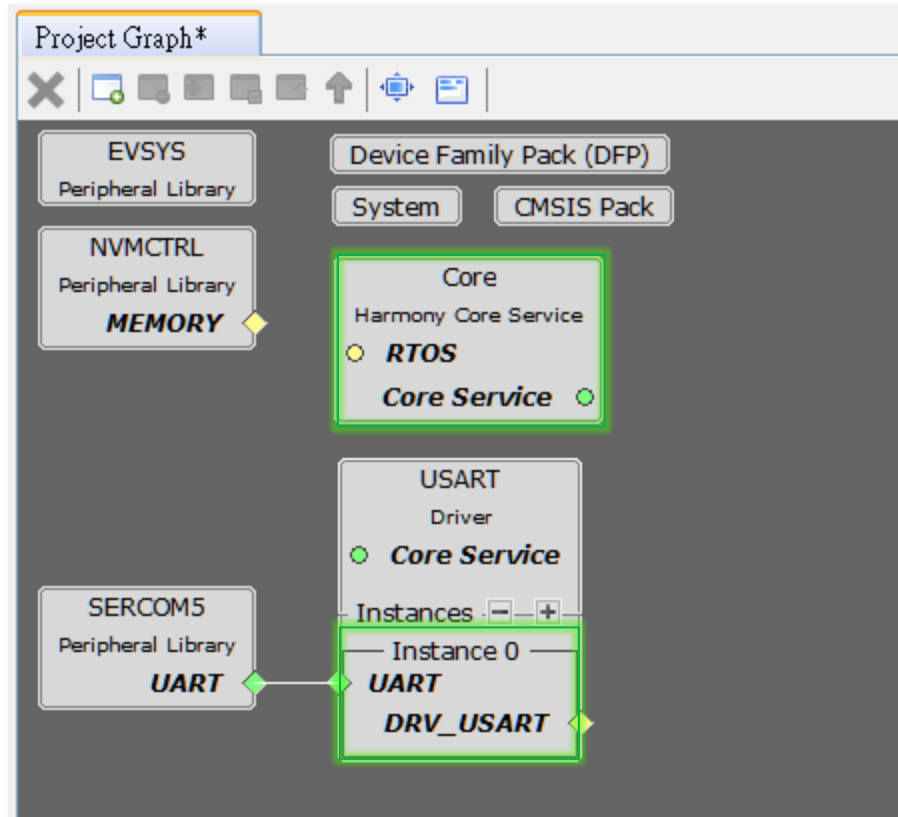
This driver provides application ready routines to read and write data from/to the USART peripheral.

- **The USART driver support the following features:**

- Supports multi-instance and multi-client mode.
- Provides data transfer events.
- Supports blocking and non-blocking operation.
- Features thread-safe functions for use in RTOS applications.
- Supports DMA transfers.

USART Driver Library

To generate an Application File
for Driver State Machine



USART Driver Library

API functions of the USART Driver library.

- System Functions

DRV_USART_Initialize

Initializes the USART instance for the specified driver index.

DRV_USART_Status

Gets the current status of the USART driver module.

- Core Client Functions

DRV_USART_Open

Opens the specified USART driver instance and returns a handle to it.

DRV_USART_Close

Closes an opened-instance of the USART driver.

- Data Transfer Functions

DRV_USART_SerialSetup

Sets the USART serial communication settings dynamically.

DRV_USART_BufferEventHandlerSet

Allows a client to identify a buffer event handling function for the driver to call back when queued buffer transfers have finished.

DRV_USART_ReadBuffer

This is a blocking function that reads data over USART.

DRV_USART_ReadBufferAdd

Queues a read operation.

DRV_USART_WriteBuffer

This is a blocking function that writes data over USART.

DRV_USART_WriteBufferAdd

Queues a write operation.

DRV_USART_BufferCompletedBytesGet

Returns the number of bytes that have been processed for the specified buffer request.

DRV_USART_BufferStatusGet

Returns the transmit/receive request status.

DRV_USART_ReadQueuePurge

Removes all buffer requests from the queue for the given client.

DRV_USART_WriteQueuePurge

Removes all write requests from the queue for the given client.

DRV_USART_ErrorGet

Gets the USART hardware errors associated with the transfer request.

DRV_USART_ReadAbort

Aborts an on-going read request

USART Driver Library

API functions of the USART Driver library. (cont.)

- Data Types and Constants

DRV_USART_INIT	Defines the data required to initialize the USART driver
DRV_USART_BUFFER_HANDLE	Handle identifying a read or write buffer passed to the driver.
DRV_USART_BUFFER_EVENT	Identifies the possible events that can result from a buffer add request.
DRV_USART_BUFFER_EVENT_HANDLER	Pointer to a USART Driver Buffer Event handler function
DRV_USART_ERROR	Defines the different types of errors for USART driver
DRV_USART_BUFFER_HANDLE_INVALID	Definition of an invalid buffer handle.
DRV_USART_SERIAL_SETUP	Defines the data required to dynamically set the serial settings.

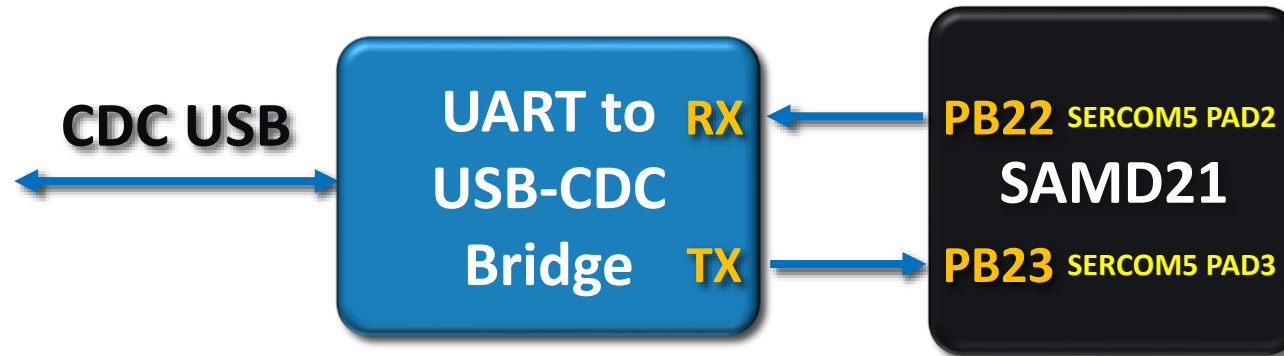
Lab 11

Single-Client UART Communication

USART Driver Library

✂ Lab11 Single-Client UART Communication

- Please create a new project for Harmony Driver exercise.
- Finish the [Lab1 – Delay based LED toggle](#) in main loop for **Driver non-block validate**.
- Try add **USART Driver** and attach **PLIB SERCOM5 UART** to build a custom driver based UART Console. (not to use the **CONSOLE System Service**)
- Implement your UART console state machine in Harmony generated **app_drv_usart1.c**
- Try use the **polling method** of Driver to output the message :
“**DRV_USART1: Hello World**” in Initial state of Driver.
- Try use the **callback method** of Driver to receive the character **X** from console input and echo it as “**Receive : X (DRV_USART1)**”



⚡ Lab11 Single-Client UART Communication

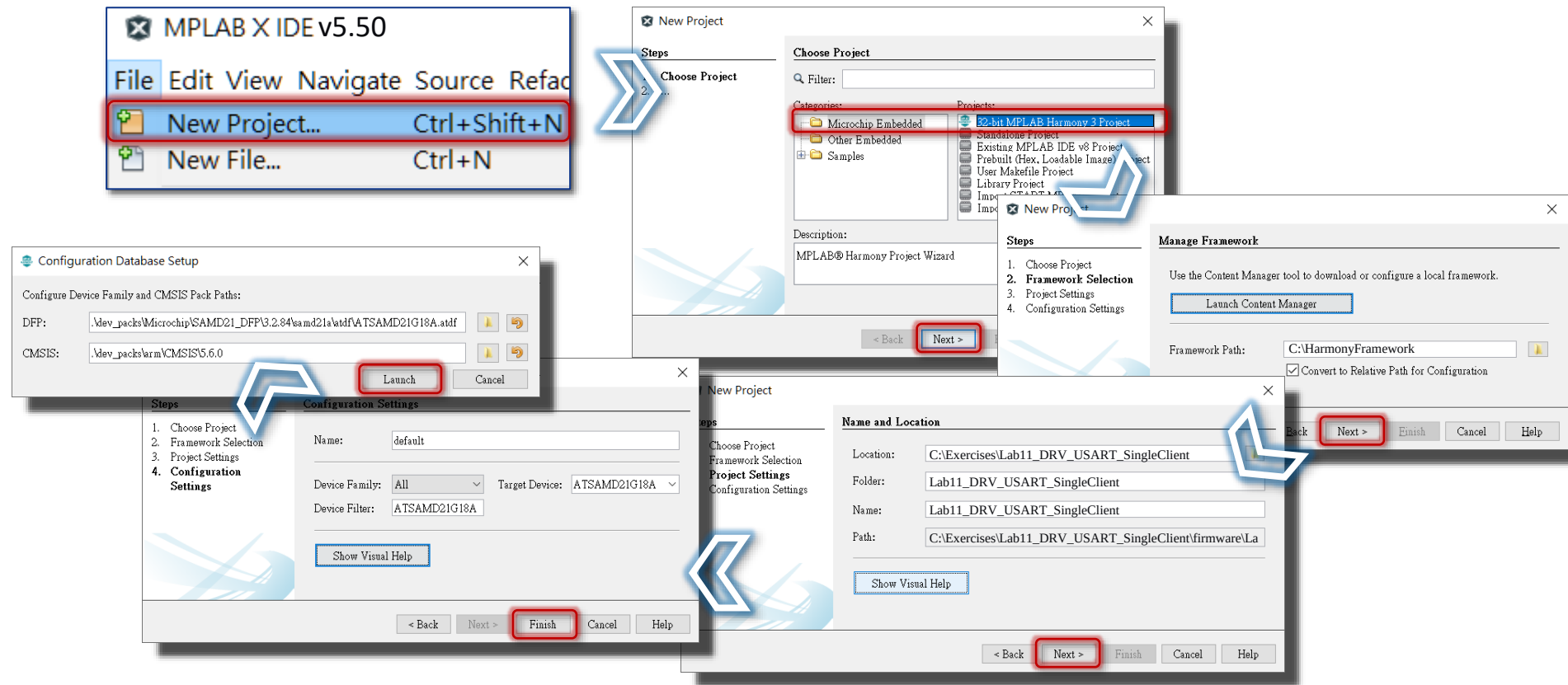
- Create a new project of Harmony v3

Framework Path : C:\HarmonyFramework

Location : C:\Exercises\Lab11_DRV_USART_SingleClient

Folder : Lab11_DRV_USART_SingleClient

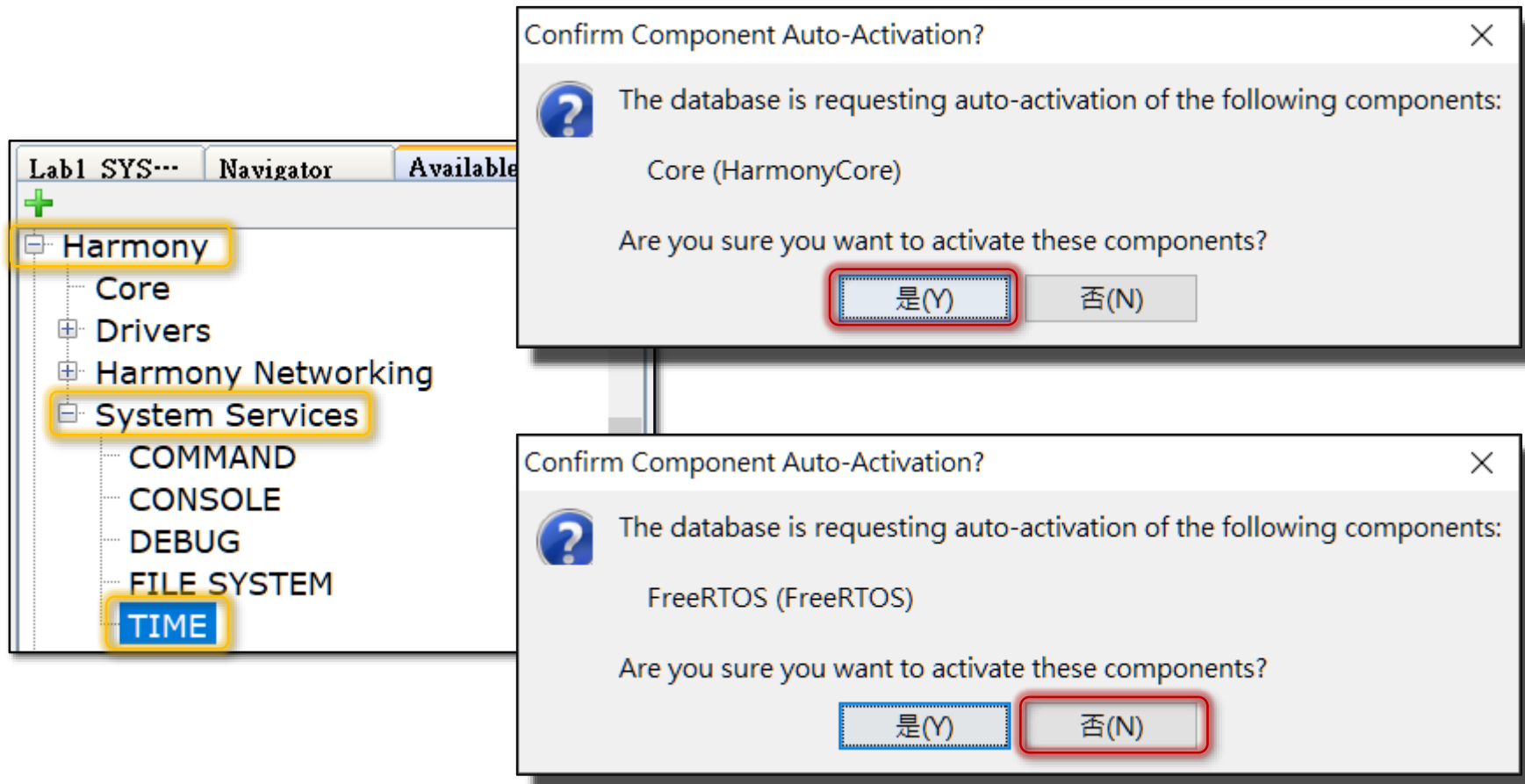
Target Device : ATSAM21G18A



✂ Lab11 Single-Client UART Communication

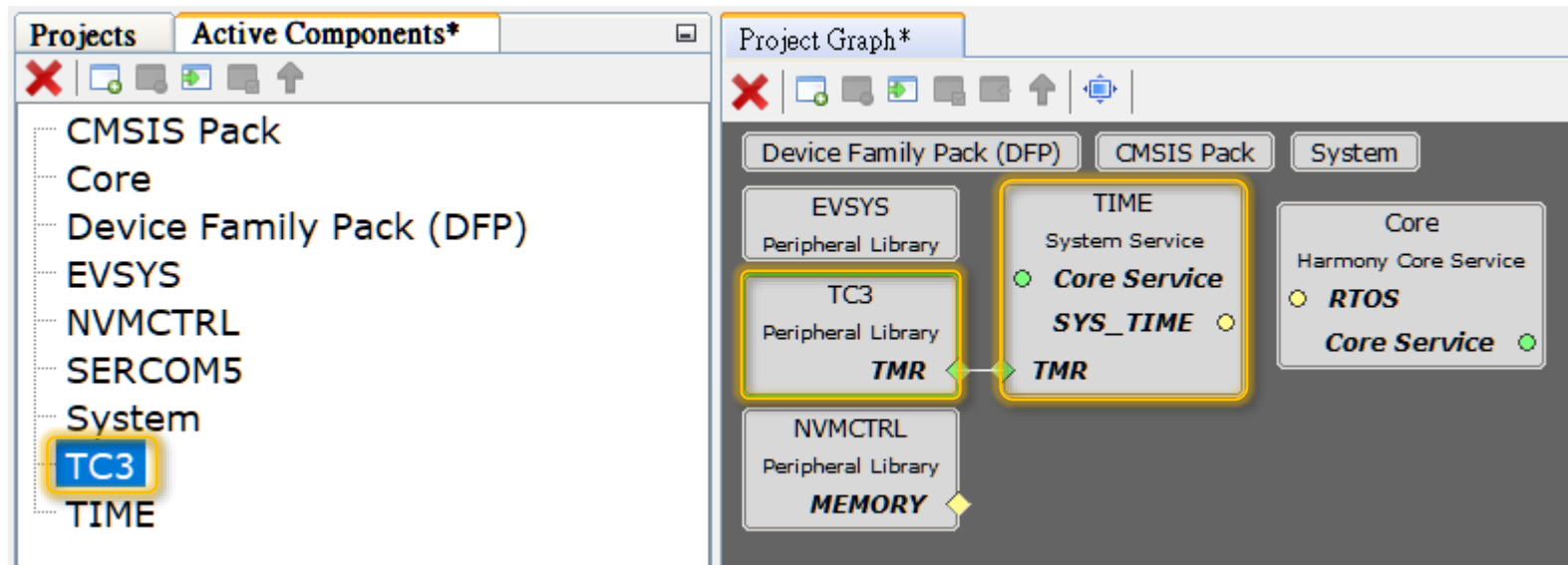
- Harmony Configuration (Add TIME system service)

Add TIME system service as we did in Lab1



✖ Lab11 Single-Client UART Communication

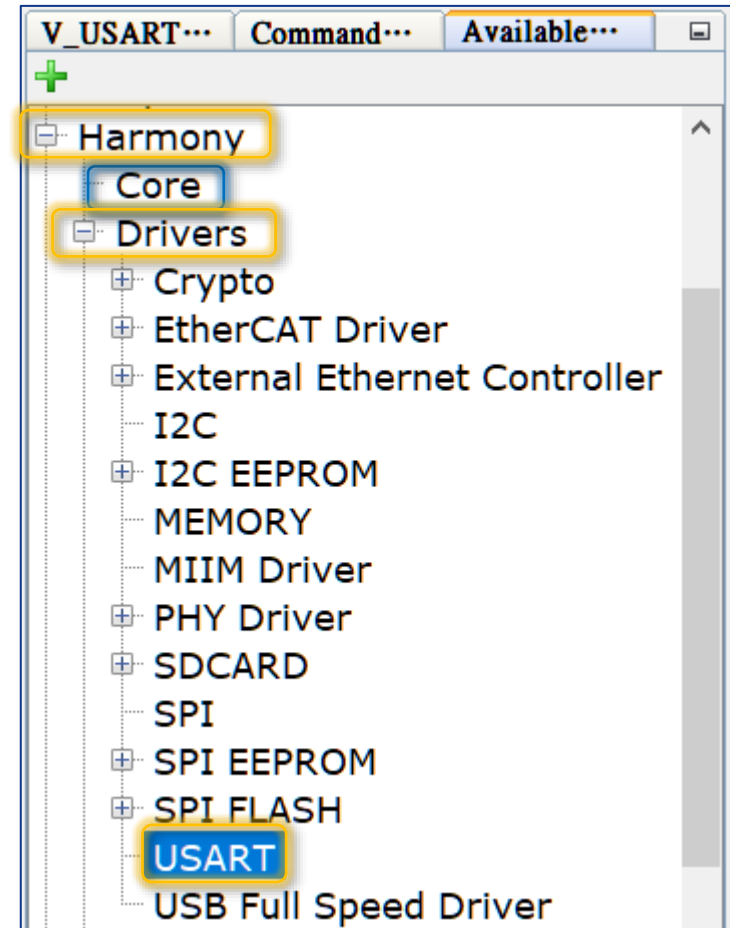
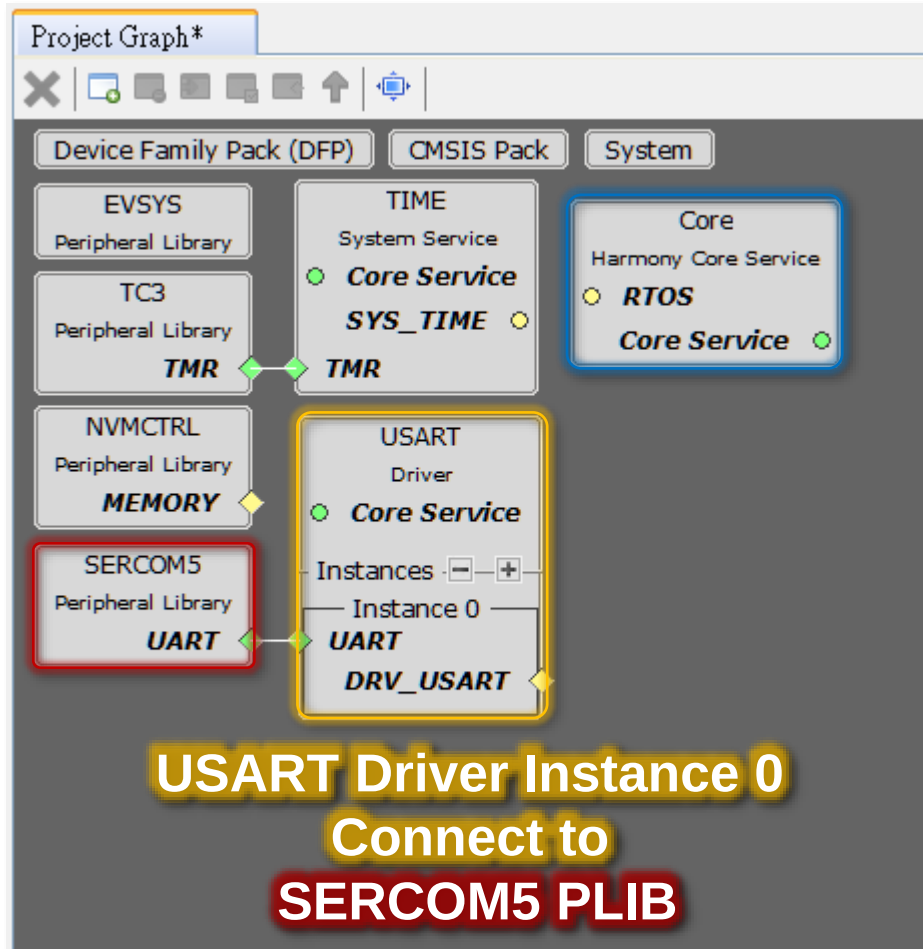
- Choose TC3 as Timer source of TIME system service,
- Use the TIME system service to create a polling-based delay and **toggle LED1** in main loop for **Driver non-block validate**.



Select TC3 as Timer Source of TIME system service

⚙️ Lab11 Single-Client UART Communication

- Add USART driver and attach **PLIB SERCOM5 UART**.



⚙️ Lab11 Single-Client UART Communication

- Configuration of USART driver and PLIB SERCOM5

The image shows two configuration windows from the Microchip Studio IDE. The left window, titled 'Configuration Options*', shows the USART peripheral configuration. The 'PLIB Used' is set to 'SERCOM5'. The 'Number of Clients' is 1, and the 'Transfer Queue Size' is 5. The 'Use DMA for Transmit?' and 'Use DMA for Receive?' options are both unchecked. The right window, titled 'Configuration Options', shows the SERCOM5 peripheral configuration. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is set to 'Non-blocking mode'. The 'Receive Enable' and 'Transmit Enable' options are both checked. The 'Frame Format' is set to 'USART frame'. The 'Baud Rate in Hz' is 115,200. The 'Parity Mode' is 'No Parity'. The 'Character Size' is '8 Bits'. The 'Stop Bit Mode' is 'One Stop Bit'. The 'Receive Pinout' is 'SERCOM PAD[3] is used for data reception'. The 'Transmit Pinout' is 'PAD[2] = TxD; PAD[3] = XCK'. The 'Enable Run in Standby' option is unchecked.

Driver will always select to Non-blocking mode

DMA to be option of Driver

**Receive Pinout : SERCOM PAD[3]
Transmit Pinout: SERCOM PAD[2]**

✖ Lab11 Single-Client UART Communication

- Pin Configuration of **GPIO PA20(LED1)** and **PLIB SERCOM5** pads

Project Graph* Pin Diagram × Pin Table × Pin Settings ×									
Order: Ports ▾		Table View		☒ Easy View					
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
29	PA20	LED1	GPIO ▾	Digital	Out ▾	Low	☐	☐	NORMAL ▾
37	PB22		SERCOM5_PAD2 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾
38	PB23		SERCOM5_PAD3 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾

PA20 as GPIO output and custom name to LED1

PB22 as SERCOM5_PAD2

PB23 as SERCOM5_PAD3

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

Package: TQFP48

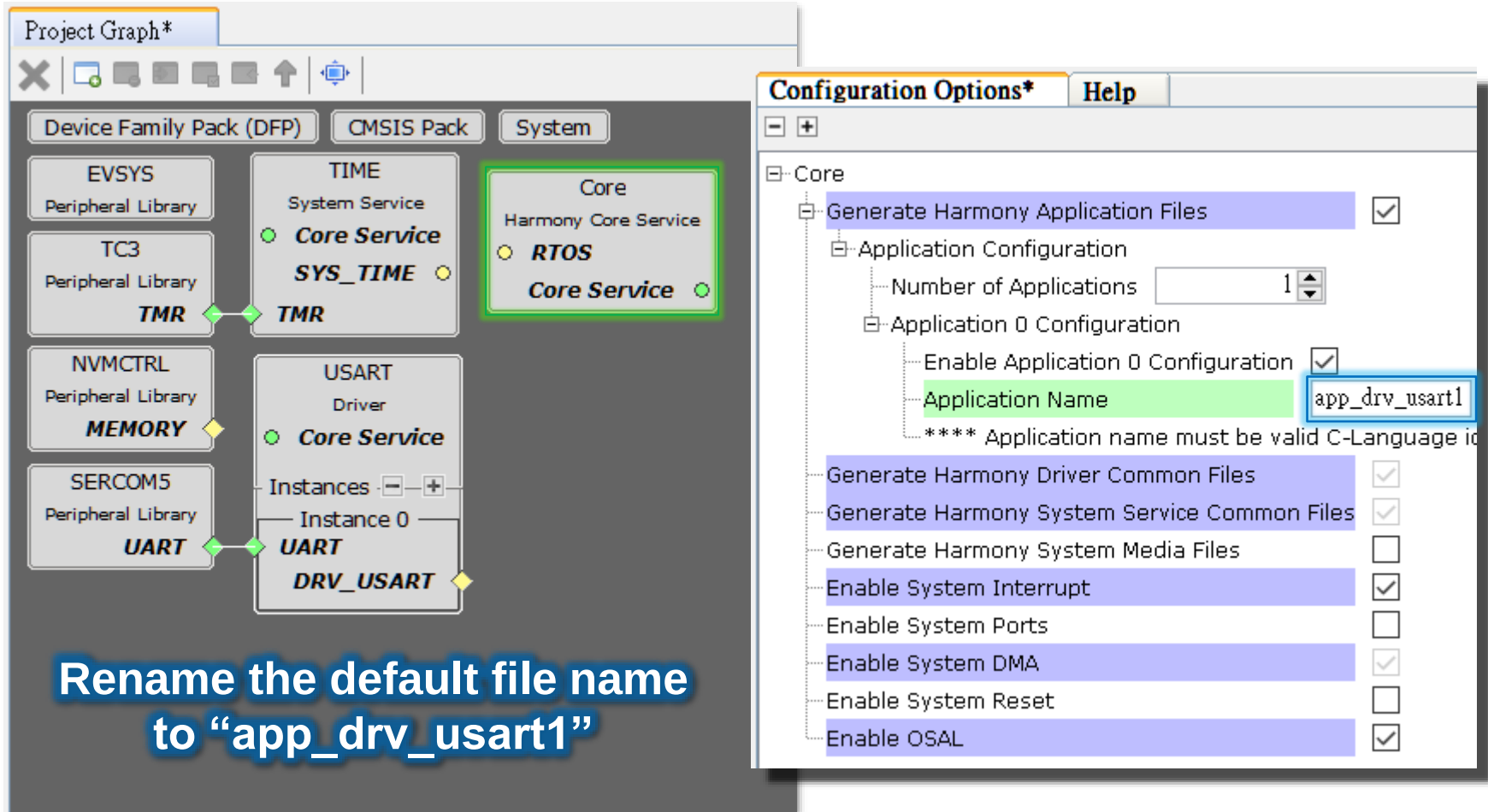
A

B

Module	Function	13	14	15	16	17	18	19	20	21	22	23	24	25	27	28	30	31	2	3	8	9	10	11	22	23
GPIO	GPIO																									
SERCOM5	SERCOM5_PAD0																									
	SERCOM5_PAD1																									
	SERCOM5_PAD2																									
	SERCOM5_PAD3																									

✖ Lab11 Single-Client UART Communication

- In **Core** (Harmony Core Service) to find the default Application Name “app” and rename to “**app_drv_usart1**” for our **USART Driver** state machine implement.



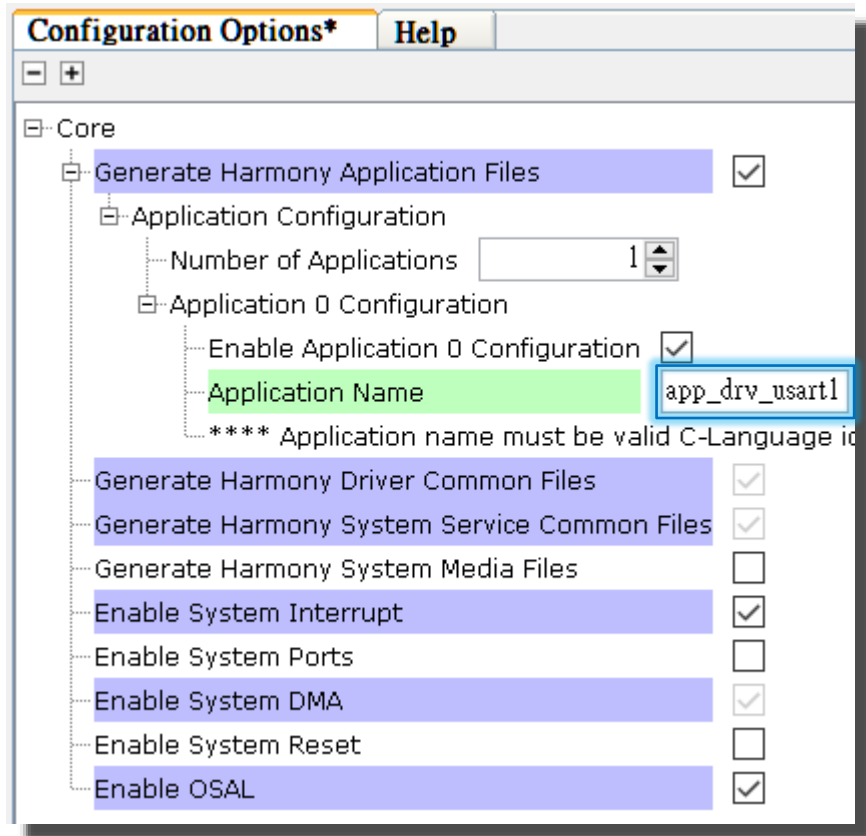
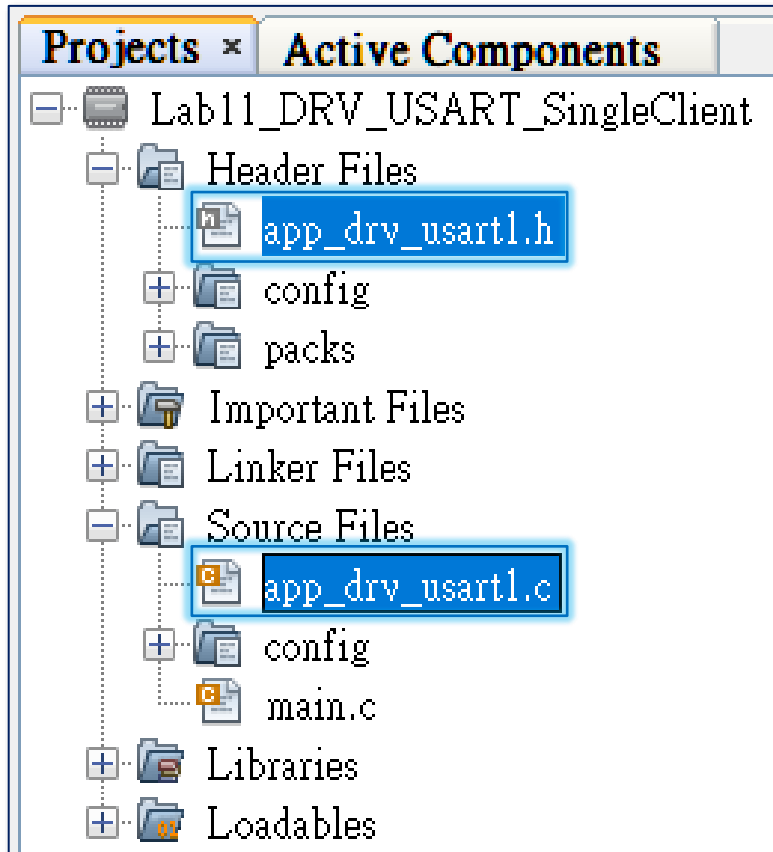
The image shows two windows from the IDE. The left window, titled "Project Graph*", displays a block diagram of the system components. It includes tabs for "Device Family Pack (DFP)", "CMSIS Pack", and "System". Under "System", the "Core" block (Harmony Core Service) is highlighted with a green box. The "Core" block contains "Core Service" (selected), "RTOS", and "Core Service". The "USART Driver" block is also visible, showing "Instance 0" with "DRV_USART".

The right window, titled "Configuration Options*", shows the configuration for the "Core" block. The "Generate Harmony Application Files" option is checked. Under "Application Configuration", the "Number of Applications" is set to 1. Under "Application 0 Configuration", the "Enable Application 0 Configuration" option is checked, and the "Application Name" is set to "app_drv_usart1". A note below states: "**** Application name must be valid C-Language id".

Rename the default file name to “app_drv_usart1”

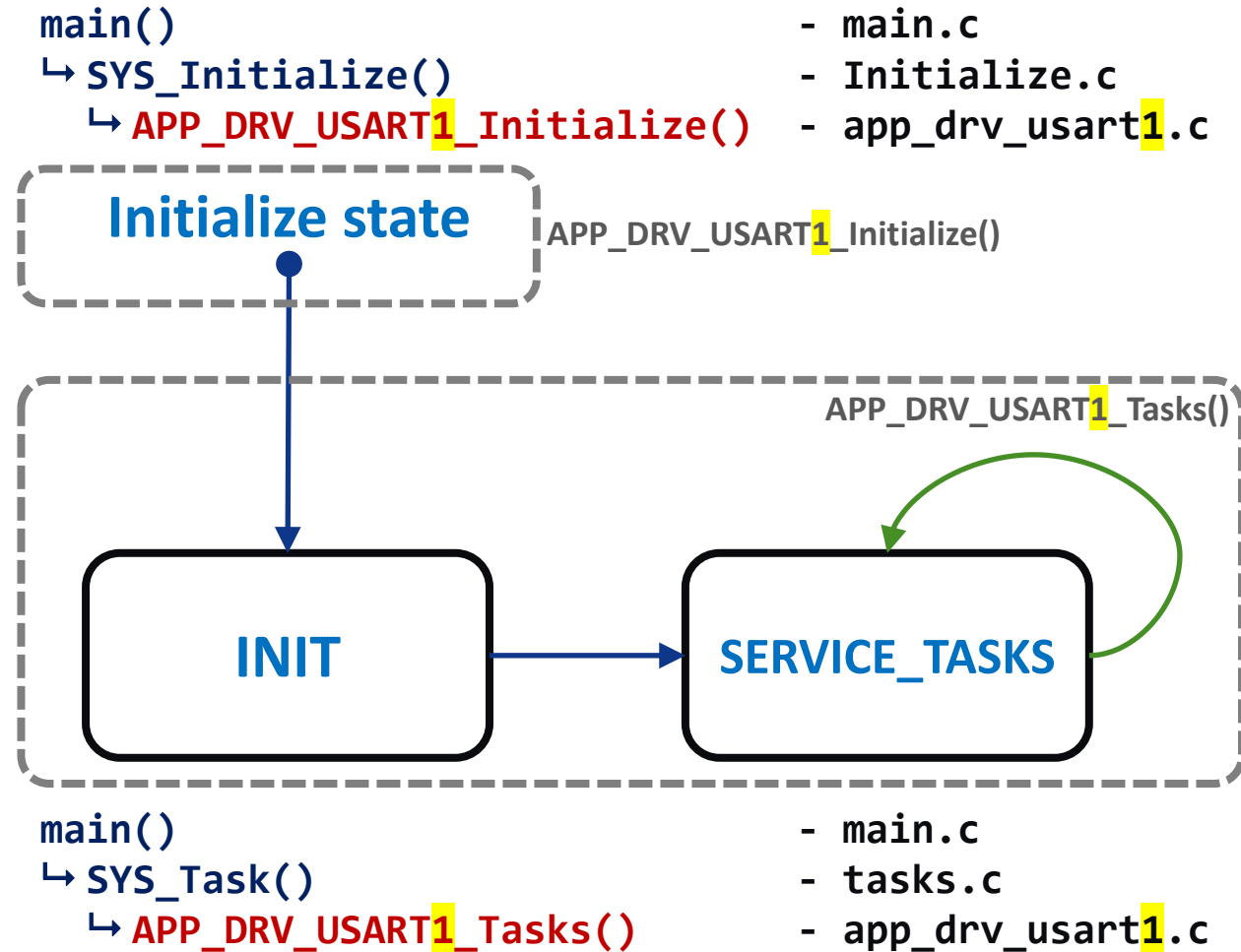
⚙️ Lab11 Single-Client UART Communication

- For **Driver based application**, most custom implement will finish in the APP file **app_drv_usart1.c** but not in **main.c**, let's complete your APP state machine step by step from next page.



⚙️ Lab11 Single-Client UART Communication

- Harmony will create a template APP state machine as below.



⚙️ Lab11 Single-Client UART Communication

- Let's check the template state machine in “app_drv_usart1.c”.

```
#include "app_drv_usart1.h"

APP_DRV_USART1_DATA app_drv_usart1Data;

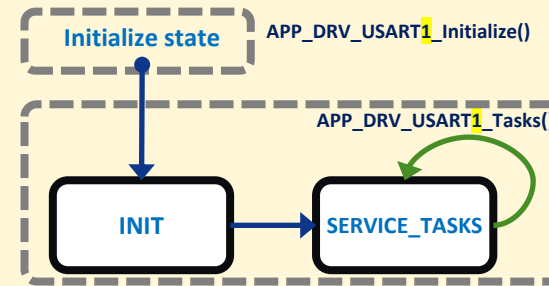
void APP_DRV_USART1_Initialize ( void )
{
    app_drv_usart1Data.state = APP_DRV_USART1_STATE_INIT;
}

void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
        case APP_DRV_USART1_STATE_INIT:
            bool appInitialized = true;

            if (appInitialized)
            {
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_SERVICE_TASKS;
            }
            break;

        case APP_DRV_USART1_STATE_SERVICE_TASKS:
            break;

        default:
            break;
    }
}
```



**For read in clear
all comments has been removed**

⚙️ Lab11 Single-Client UART Communication

- Let's check the header file declaration "`app_drv_usart.h`"

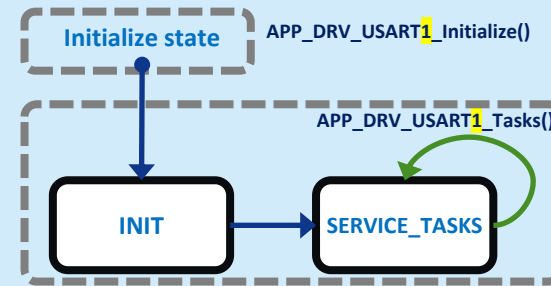
```
#ifndef _APP_DRV_USART1_H
#define _APP_DRV_USART1_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_USART1_STATE_INIT=0,
    APP_DRV_USART1_STATE_SERVICE_TASKS,
} APP_DRV_USART1_STATES;

typedef struct
{
    APP_DRV_USART1_STATES state;
} APP_DRV_USART1_DATA;

void APP_DRV_USART1_Initialize ( void );
void APP_DRV_USART1_Tasks( void );
#endif
```



**For read in clear
all comments and C++
compatible code segment
has been removed**

⚙️ Lab11 Single-Client UART Communication

- To check “**main.c**” and add LED1 toggle as we did in Lab1

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

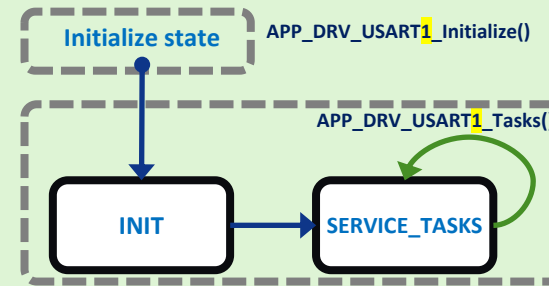
SYS_TIME_HANDLE tmrHandle;

int main( void )
{
    SYS_Initialize(NULL);     APP_DRV_USART1_Initialize()
    SYS_TIME_DelayMS(1000, &tmrHandle);

    while( true )
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true )
        {
            LED1_Toggle();
            SYS_TIME_DelayMS(1000, &tmrHandle);
        }

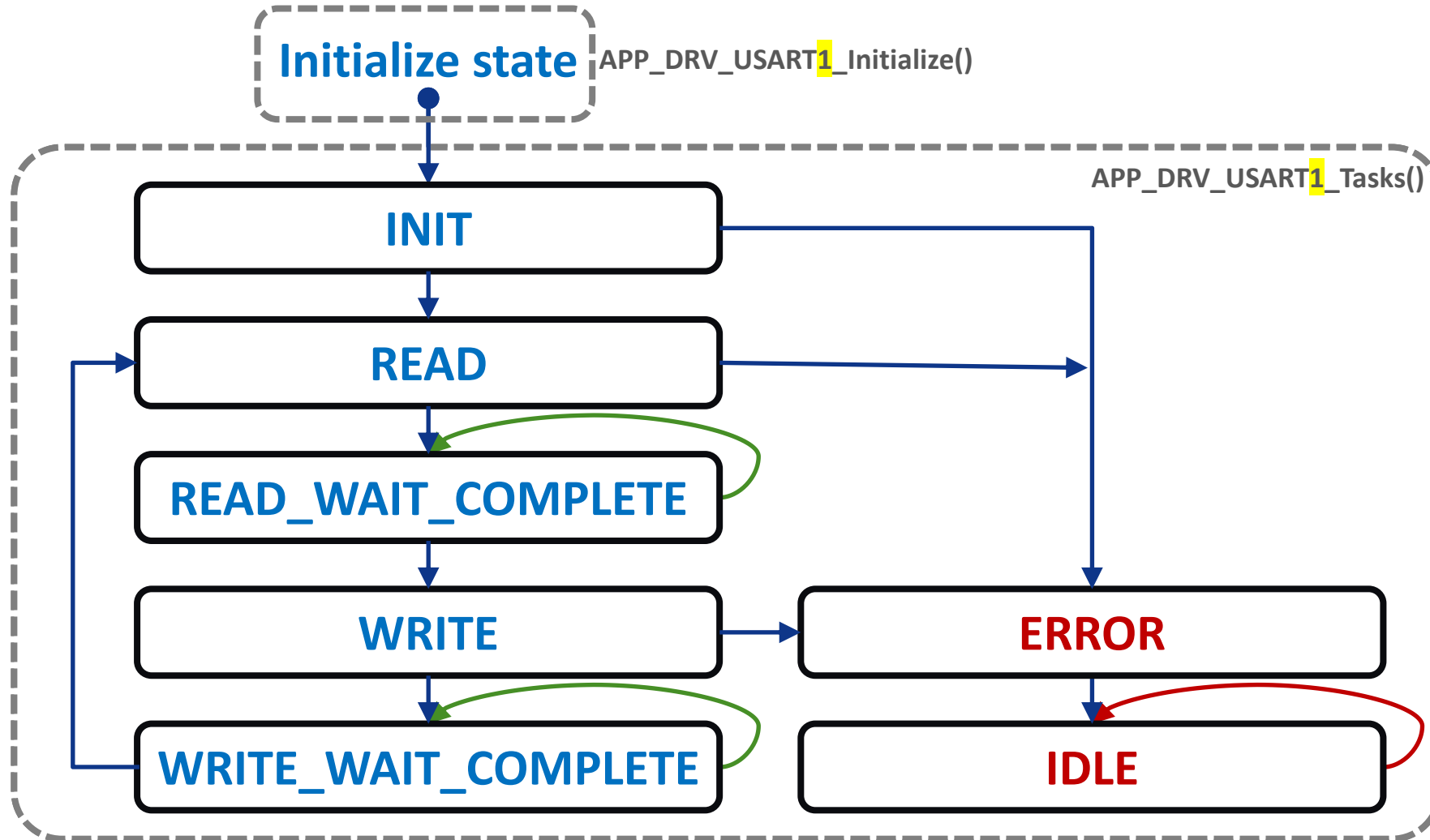
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks();          APP_DRV_USART1_Tasks()
    }

    /* Execution should not come here during normal operation */
    return ( EXIT_FAILURE);
}
```



⚙️ Lab11 Single-Client UART Communication

- The new state machine we want to implement in “`app_drv_uart1.c`” as below.



⚙️ Lab11 Single-Client UART Communication

- Add more states in “app_drv_usart1.h”.

```
#ifndef _APP_DRV_USART1_H
#define _APP_DRV_USART1_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_USART1_STATE_INIT=0,
    // APP_DRV_USART1_STATE_SERVICE_TASKS,
    APP_DRV_USART1_STATE_READ,
    APP_DRV_USART1_STATE_READ_WAIT_COMPLETE,
    APP_DRV_USART1_STATE_WRITE,
    APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE,
    APP_DRV_USART1_STATE_ERROR,
    APP_DRV_USART1_STATE_IDLE,
} APP_DRV_USART1_STATES;

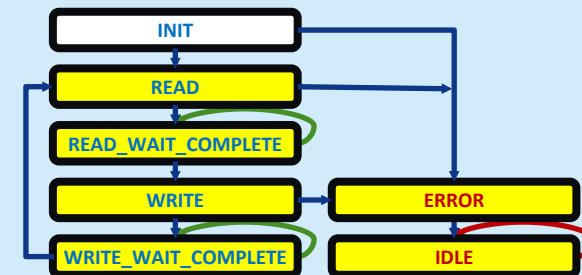
typedef struct
{
    APP_DRV_USART1_STATES state;
} APP_DRV_USART1_DATA;

void APP_DRV_USART1_Initialize ( void );
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



⚙️ Lab11 Single-Client UART Communication

- Construct new states flow of task in “`app_drv_uart1.c`”.

```
void APP_DRV_USART1_Tasks ( void )  
{
```

```
    switch ( app_drv_uart1Data.state )  
    {
```

```
        case APP_DRV_USART1_STATE_INIT:
```

```
            app_drv_uart1Data.state = APP_DRV_USART1_STATE_READ;  
            break;
```

```
        case APP_DRV_USART1_STATE_READ:
```

```
            app_drv_uart1Data.state = APP_DRV_USART1_STATE_READ_WAIT_COMPLETE;  
            break;
```

```
        case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
```

```
            app_drv_uart1Data.state = APP_DRV_USART1_STATE_WRITE;  
            break;
```

```
        case APP_DRV_USART1_STATE_WRITE:
```

```
            app_drv_uart1Data.state = APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE;  
            break;
```

```
        case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
```

```
            app_drv_uart1Data.state = APP_DRV_USART1_STATE_READ;  
            break;
```

```
        case APP_DRV_USART1_STATE_ERROR:
```

```
            app_drv_uart1Data.state = APP_DRV_USART1_STATE_IDLE;  
            break;
```

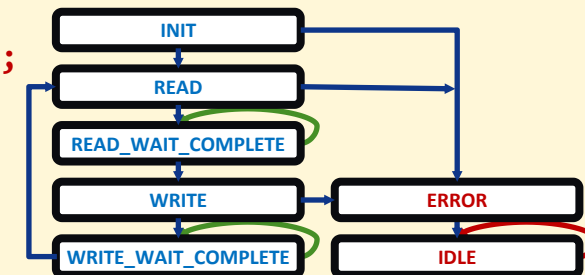
```
        case APP_DRV_USART1_STATE_IDLE:
```

```
            break;
```

```
    }
```

```
}
```

Overwrite original example code



⚙️ Lab11 Single-Client UART Communication

- Header include and Global Declarations in “app_drv_usart1.c”.

Keep this line

```
#include "app_drv_usart1.h"
#include "driver/usart/drv_usart.h"
#include <stdio.h>
#include <string.h>

// *****
// *****
// Section: Global Data Definitions
// *****
// *****

DRV_HANDLE Uart1Handle;
DRV_USART_BUFFER_HANDLE Uart1WriteTransferHandle;
DRV_USART_BUFFER_HANDLE Uart1ReadTransferHandle;
volatile int Uart1WriteComplete = false;
volatile int Uart1ReadComplete = false;
char Uart1WriteBuf[50];
char Uart1ReadBuf[50];

// *****
/* Application Data

Summary:
    Holds application data

Description:
    This structure holds the application's data.

...
*/

APP_DRV_USART1_DATA app_drv_usart1Data;
```

You could put the
global declaration in comment
“Section: Global Data Definitions”

⚙️ Lab11 Single-Client UART Communication

- Add Driver Open, Write and Transfer Status **Polling** Operation in state **INIT**.

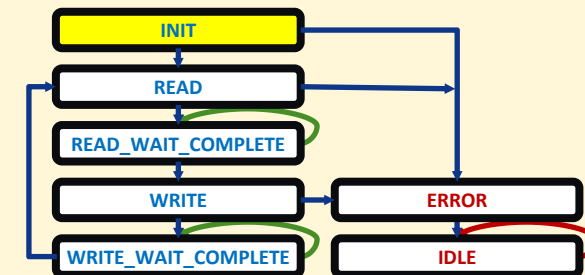
```
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
    case APP_DRV_USART1_STATE_INIT:
        Uart1Handle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_READWRITE);
        if(Uart1Handle == DRV_HANDLE_INVALID)
        {
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_ERROR;
        }
        else
        {
            sprintf(Uart1WriteBuf, "DRV_USART1: Hello world\r\n");
            DRV_USART_WriteBufferAdd(Uart1Handle, Uart1WriteBuf, strlen(Uart1WriteBuf),
                                     &Uart1WriteTransferHandle);
            while( DRV_USART_BufferStatusGet(Uart1WriteTransferHandle)!=
                  DRV_USART_BUFFER_EVENT_COMPLETE ) {}

            app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;
        }
    }
    break;

    case APP_DRV_USART1_STATE_READ:
    ...
    case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
    ...
    case APP_DRV_USART1_STATE_WRITE:
    ...
    case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
    ...
    case APP_DRV_USART1_STATE_ERROR:
    ...
}
```

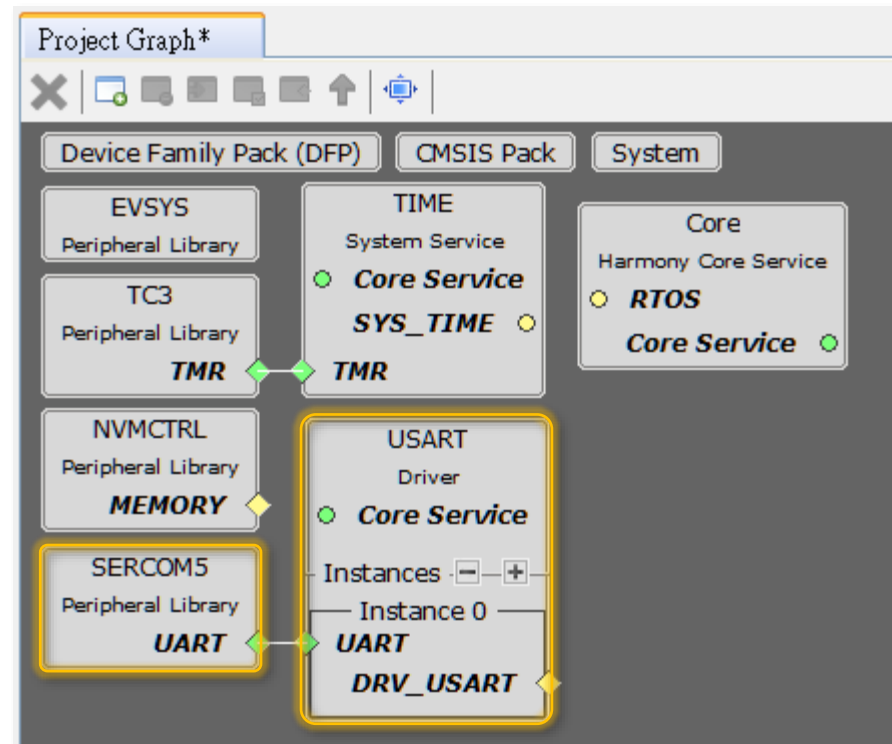
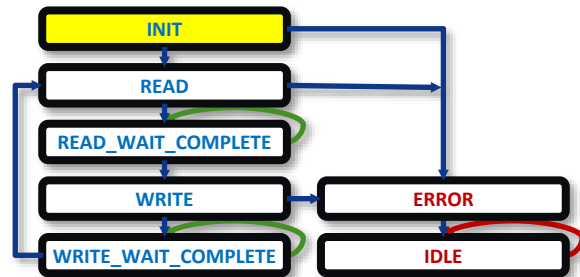
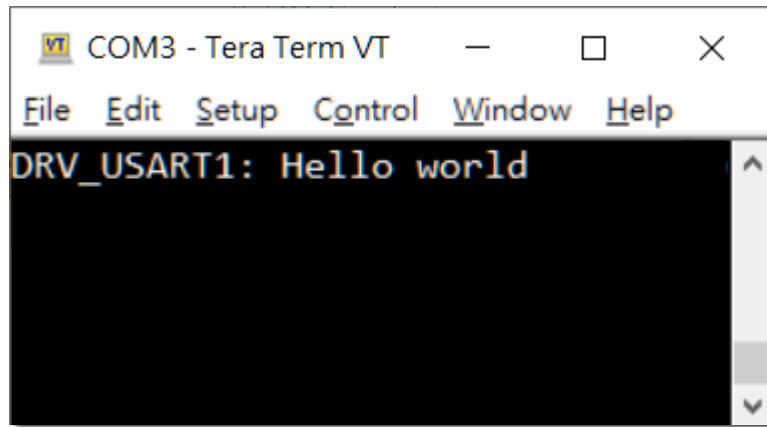
**Transfer Status Polling operation
to check status of USART write**

Keep this line



⚙️ Lab11 Single-Client UART Communication

- Compiler and programming your project to EVB.
- Test our first Driver implementation and see the **USART driver** message output from state **APP_DRV_USART1_STATE_INIT**.



✂ Lab11 Single-Client UART Communication

- Register and Implement Driver Transfer Status **Callback** Operation.

```
// Section: Application Callback Functions
// *****
void Uart1TransferHandler( DRV_USART_BUFFER_EVENT event,
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
{
    switch( event )
    {
        case DRV_USART_BUFFER_EVENT_PENDING: break;
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            break;
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:
        case DRV_USART_BUFFER_EVENT_ERROR:
        case DRV_USART_BUFFER_EVENT_HANDLE_INVALID:
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_ERROR;
            break;
    }
}

...
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
        case APP_DRV_USART1_STATE_INIT:
            ...
        else
        {
            while( DRV_USART_BufferStatusGet(Uart1WriteTransferHandle)!=
                  DRV_USART_BUFFER_EVENT_COMPLETE ) {}
        }
    }
}

Callback Register DRV_USART_BufferEventHandlerSet(Uart1Handle, Uart1TransferHandler, 0);
app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;
```

**Catch Events in
Transfer Status Callback
Function and handle
Error event**

⚙️ Lab11 Single-Client UART Communication

- Implement state **READ** to read one byte from UART console.

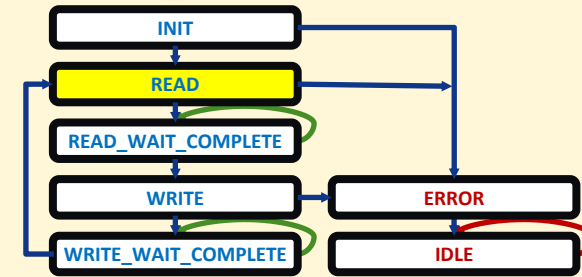
```
void Uart1TransferHandler( DRV_USART_BUFFER_EVENT event,  
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
```

```
{  
    switch( event )  
    {  
        ...  
        case DRV_USART_BUFFER_EVENT_COMPLETE:  
            if(bufferHandle == Uart1ReadTransferHandle)  
                Uart1ReadComplete = true;  
            break;  
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:  
            ...  
    }  
}
```

Keep this line

```
void APP_DRV_USART1_Tasks ( void )  
{  
    switch ( app_drv_usart1Data.state )  
    { ...  
        case APP_DRV_USART1_STATE_READ:  
            DRV_USART_ReadBufferAdd(Uart1Handle, Uart1ReadBuf, 1, &Uart1ReadTransferHandle);  
            if(Uart1ReadTransferHandle == DRV_USART_BUFFER_HANDLE_INVALID)  
            {  
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_ERROR;  
            }  
            else  
            {  
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ_WAIT_COMPLETE;  
            }  
            break;  
        case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
```

Manage Read Complete flag while
Read Transfer Complete Events



⚙️ Lab11 Single-Client UART Communication

- Implement state **READ_WAIT_COMPLETE**.

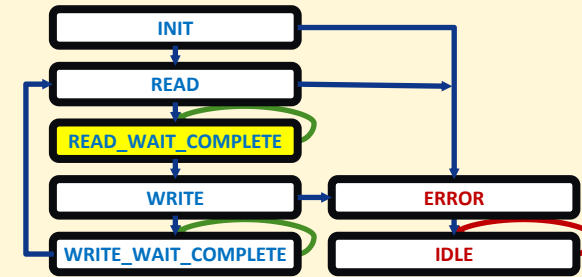
```
void Uart1TransferHandler( DRV_USART_BUFFER_EVENT event,
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart1ReadTransferHandle)
                Uart1ReadComplete = true;
            break;
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:
            ...
    }
}
```

```
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
        ...
        case APP_DRV_USART1_STATE_READ:
            ...

        case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
            if(Uart1ReadComplete == true)
            {
                Uart1ReadComplete = false;
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_WRITE;
            }
            break;

        case APP_DRV_USART1_STATE_WRITE:
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE;
```

Keep this line



State non blocking while waiting Read Complete

⚙️ Lab11 Single-Client UART Communication

- Implement state **WRITE** to output “Received : X” to console.

```
void Uart1TransferHandler( ... )  
{
```

```
    switch( event )
```

```
    {
```

```
        ...  
        case DRV_USART_BUFFER_EVENT_COMPLETE:
```

```
            if(bufferHandle == Uart1ReadTransferHandle)
```

```
                Uart1ReadComplete = true;
```

```
            if(bufferHandle == Uart1WriteTransferHandle)
```

```
                Uart1WriteComplete = true;
```

Keep this line

```
            break;
```

```
        ...
```

```
    }
```

```
void APP_DRV_USART1_Tasks ( void )
```

```
{
```

```
    switch ( app_drv_usart1Data.state )
```

```
    {
```

```
        ...  
        case APP_DRV_USART1_STATE_WRITE:
```

```
            sprintf(Uart1WriteBuf, "Receive : %s (DRV_USART1)\r\n", Uart1ReadBuf);
```

```
            DRV_USART_WriteBufferAdd(Uart1Handle, Uart1WriteBuf, strlen(Uart1WriteBuf),  
                                     &Uart1WriteTransferHandle);
```

```
            if(Uart1WriteTransferHandle == DRV_USART_BUFFER_HANDLE_INVALID)
```

```
            {
```

```
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_ERROR;
```

```
            }
```

```
            else
```

```
            {
```

```
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE;
```

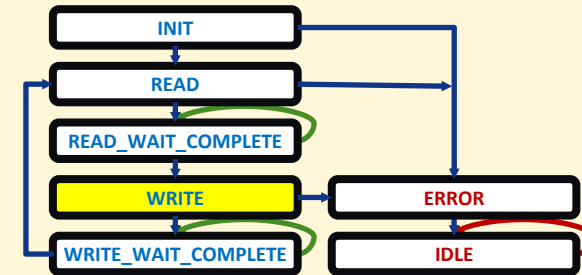
```
            }
```

Keep this line

```
            break;
```

```
        case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
```

Manage Write Complete flag while
Write Transfer Complete Events



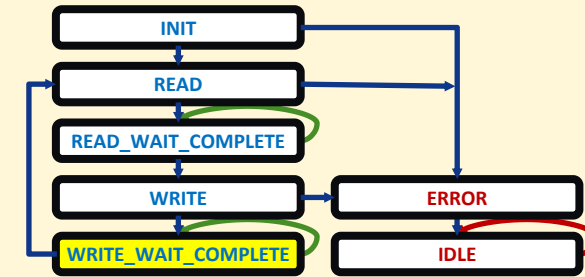
⚙️ Lab11 Single-Client UART Communication

- Implement state **WRITE_WAIT_COMPLETE**

```
void Uart1TransferHandler( ... )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart1ReadTransferHandle)
                Uart1ReadComplete = true;
            if(bufferHandle == Uart1WriteTransferHandle)
                Uart1WriteComplete = true;
            break;
        ...
    }
}

void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
        ...
        case APP_DRV_USART1_STATE_WRITE:
            ...
            case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
                if(Uart1WriteComplete == true)
                {
                    Uart1WriteComplete = false;
                    app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;
                }
                break;
            ...
        case APP_DRV_USART1_STATE_ERROR:
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_IDLE;
            break;
        case APP_DRV_USART1_STATE_IDLE:
            ...
    }
}
```

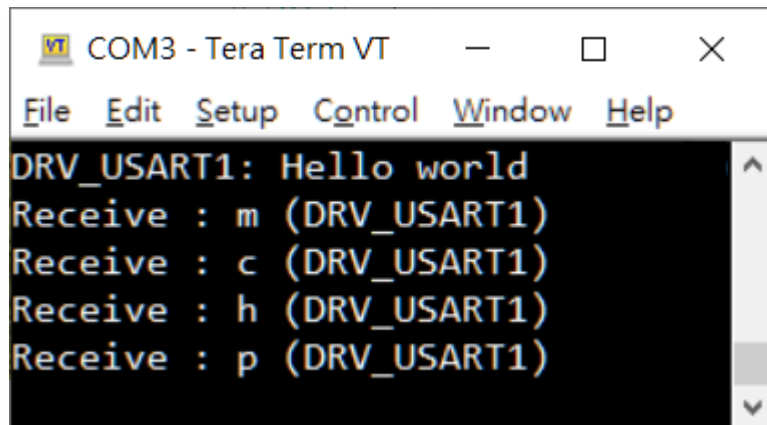
Keep this line



State non blocking while waiting Write Complete

⚙️ Lab11 Single-Client UART Communication

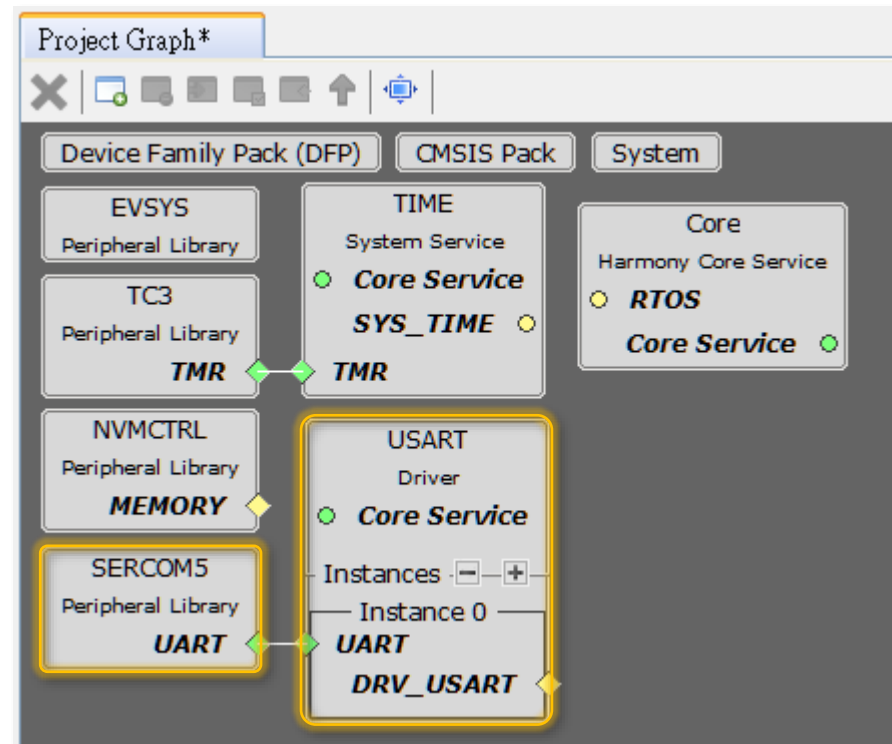
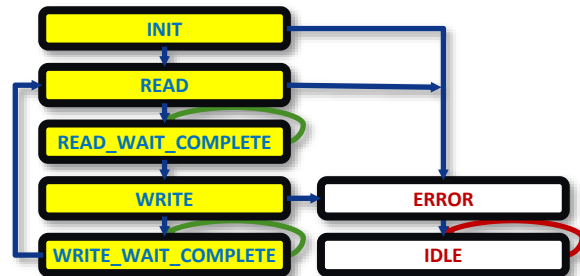
- Compiler and programming your project to EVB.
- Check the UART output in console.
- Type any character in console and check the response.
- The LED1 toggle could prove Driver state machine not to block.



COM3 - Tera Term VT

File Edit Setup Control Window Help

```
DRV_USART1: Hello world
Receive : m (DRV_USART1)
Receive : c (DRV_USART1)
Receive : h (DRV_USART1)
Receive : p (DRV_USART1)
```



⚙️ Lab11 Single-Client UART Communication

- **Additional exercise**

- Implement a C stdio **printf()** like function to output Lab debug message.
- The polling method is good for implement in easily.

```
app_drv_usart1.c

#include <stdio.h>
#include <string.h>
#include <stdarg.h>

APP_DRV_USART1_DATA app_drv_usart1Data;

void USART1printf( const char *format, ... )
{
    size_t len=0;
    va_list args = {0};

    va_start( args, format );
    len = vsnprintf( Uart1WriteBuf, sizeof(Uart1WriteBuf), format, args );
    va_end( args );

    if( len > 0 && len < sizeof(Uart1WriteBuf) )
    {
        Uart1WriteBuf[len] = '\0';
        DRV_USART_WriteBufferAdd(Uart1Handle, Uart1WriteBuf, strlen(Uart1WriteBuf),
                                &Uart1WriteTransferHandle);
        while( DRV_USART_BufferStatusGet(Uart1WriteTransferHandle)!=
                DRV_USART_BUFFER_EVENT_COMPLETE ) {}
    }
}

void Uart1TransferHandler( DRV_USART_BUFFER_EVENT event, ... ) { ... }
```

⚙️ Lab11 Single-Client UART Communication

- Replace “DRV_USART1: Hello World” output by use `USART1printf()`

```
void APP_DRV_USART1_Tasks ( void )                                     app_drv_usart1.c
{
    switch ( app_drv_usart1Data.state )
    {
    case APP_DRV_USART1_STATE_INIT:
        Uart1Handle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_READWRITE);
        if(Uart1Handle == DRV_HANDLE_INVALID)
        {
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_ERROR;
        }
        else
        {
            USART1printf("DRV_USART1: Hello world\r\n");
            // sprintf(Uart1WriteBuf, "DRV_USART1: Hello world\r\n");
            // DRV_USART_WriteBufferAdd(Uart1Handle, Uart1WriteBuf, strlen(Uart1WriteBuf),
            //                               &Uart1WriteTransferHandle);
            // while( DRV_USART_BufferStatusGet(Uart1WriteTransferHandle)!=
            //                               DRV_USART_BUFFER_EVENT_COMPLETE ) {}

            DRV_USART_BufferEventHandlerSet(Uart1Handle, Uart1TransferHandler, 0);

            app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;
        }
        ...
    }
```

- Declare prototype in `app_drv_usart1.h` for external reference use.

```
void APP_DRV_USART1_Initialize ( void );                             app_drv_usart1.h
void APP_DRV_USART1_Tasks( void );

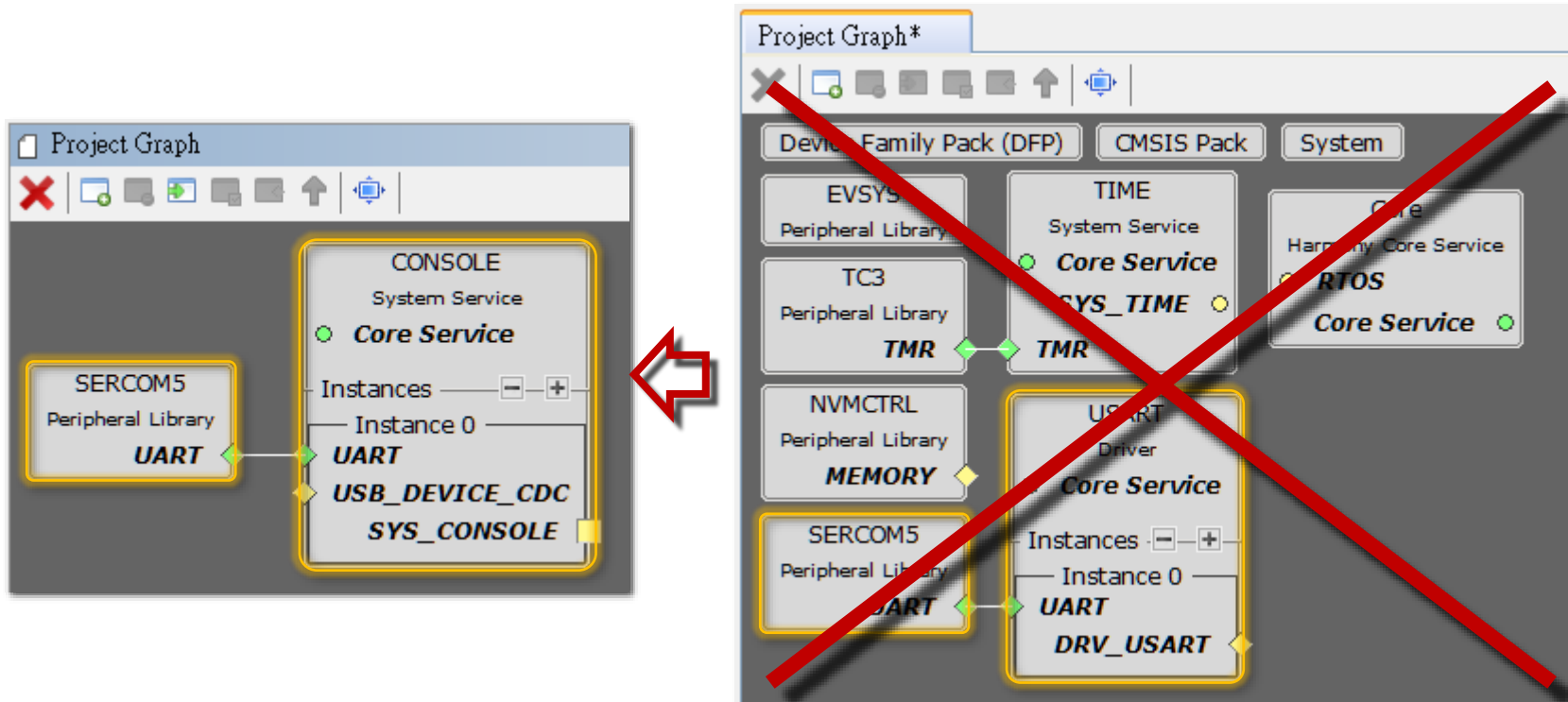
void USART1printf( const char *format, ... );

//DOM-IGNORE-BEGIN
```

⚙️ Lab11 Single-Client UART Communication

- Discuss

What we did ? If you want implement a UART console, why not just use **CONSOLE** system service ?



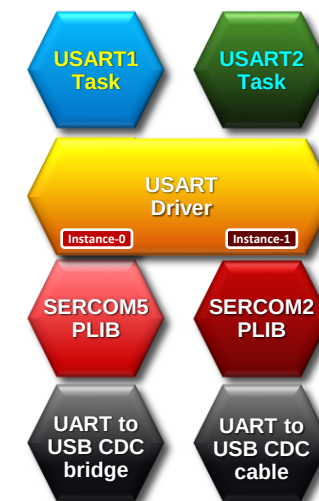
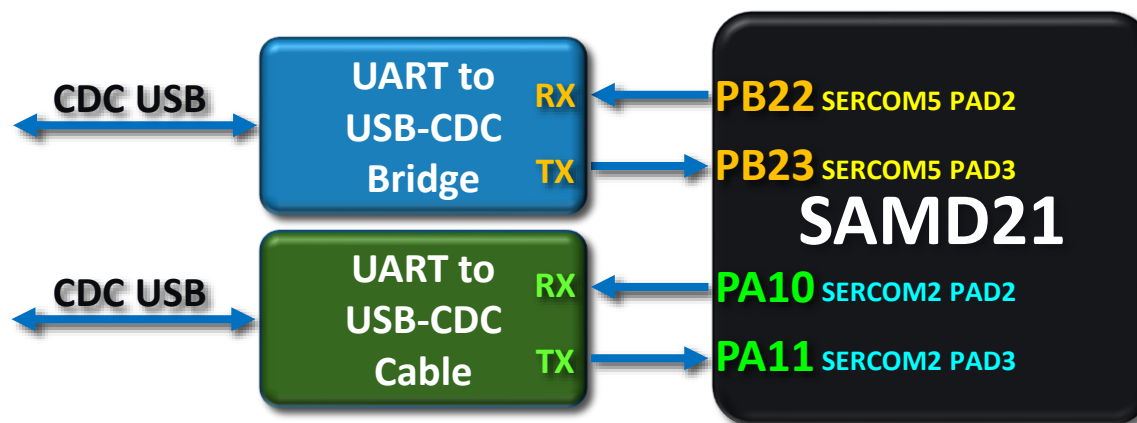
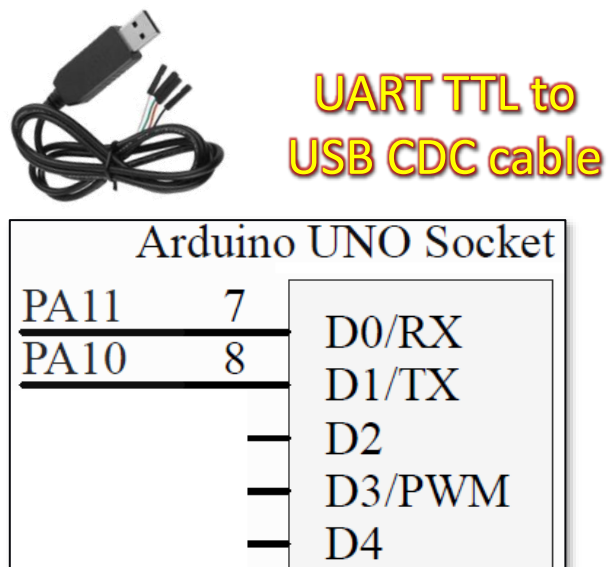
Lab 12

Multi-Instance UART Communication

USART Driver Library

⚙️ Lab12 Multi-Instance UART Communication

- Try add **instance 1** in **USART Driver** to implement another UART console from **Arduino** socket through **PLIB SERCOM2 UART**.
- You can prepare a **UART to USB-CDC cable** for UART-CDC bridge.
- Create a new APP state machine **app_drv_usart2** in Core Service.
- Try output the message “**USART2 Driver: Hello World**” while initialize state of new Driver **instance 1**.
- Try use the callback method of Driver to receive the character **X** from one console and echo it to **opposite** console “**Receive : X (DRV_USART?)**”



⚙️ Lab12 Multi-Instance UART Communication

- Add **Instance 1** in USART Driver and attach **PLIB SERCOM2 UART**, configure pads of SERCOM2.

The image shows two windows from a development IDE. The left window, titled 'Project Graph*', displays a block diagram of the system components. It includes a 'Device Family Pack (DFP)' section with 'EVSYS Peripheral Library', 'TC3 Peripheral Library', 'NVMCTRL Peripheral Library', 'SERCOM5 Peripheral Library', and 'SERCOM2 Peripheral Library'. The 'SERCOM2 Peripheral Library' contains a 'UART' block. The 'CMSIS Pack' section shows 'TIME System Service' with 'Core Service' and 'SYS_TIME'. The 'System' section shows 'Core Harmony Core Service' with 'RTOS' and 'Core Service'. The 'USART Driver' block is connected to 'Instance 0' and 'Instance 1'. 'Instance 1' is connected to 'SERCOM2 UART'. The right window, titled 'Configuration Options', shows the configuration for 'SERCOM2'. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is 'Non-blocking mode'. The 'Receive Enable' and 'Transmit Enable' checkboxes are checked. The 'Frame Format' is 'USART frame'. The 'Baud Rate in Hz' is '115,200'. The 'Parity Mode' is 'No Parity'. The 'Character Size' is '8 Bits'. The 'Stop Bit Mode' is 'One Stop Bit'. The 'Receive Pinout' is 'SERCOM PAD[3] is used for data reception'. The 'Transmit Pinout' is 'PAD[2] = TxD; PAD[3] = XCK'. The 'Enable Run in Standby' checkbox is unchecked.

USART Driver Instance 1
Connect to SERCOM2 PLIB

Configuration of SERCOM2
Receive Pinout : SERCOM PAD[3]
Transmit Pinout: SERCOM PAD[2]

Lab12 Multi-Instance UART Communication

- Pin configuration of **PLIB SERCOM2** pads.

Project Graph* Pin Diagram × Pin Table × **Pin Settings ×**

Order: **Ports** Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
15	PA10		SERCOM2_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
16	PA11		SERCOM2_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL

Configure PA10 as SERCOM2_PAD2
Configure PA11 as SERCOM2_PAD3

[illegible]

✂ Lab12 Multi-Instance UART Communication

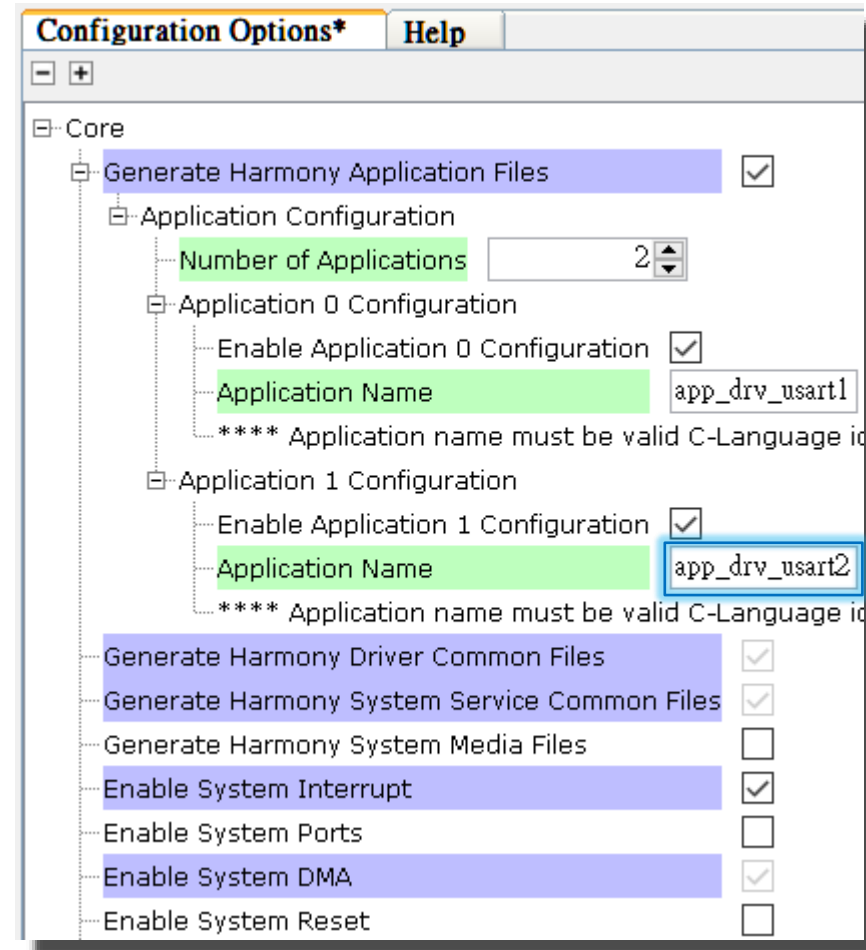
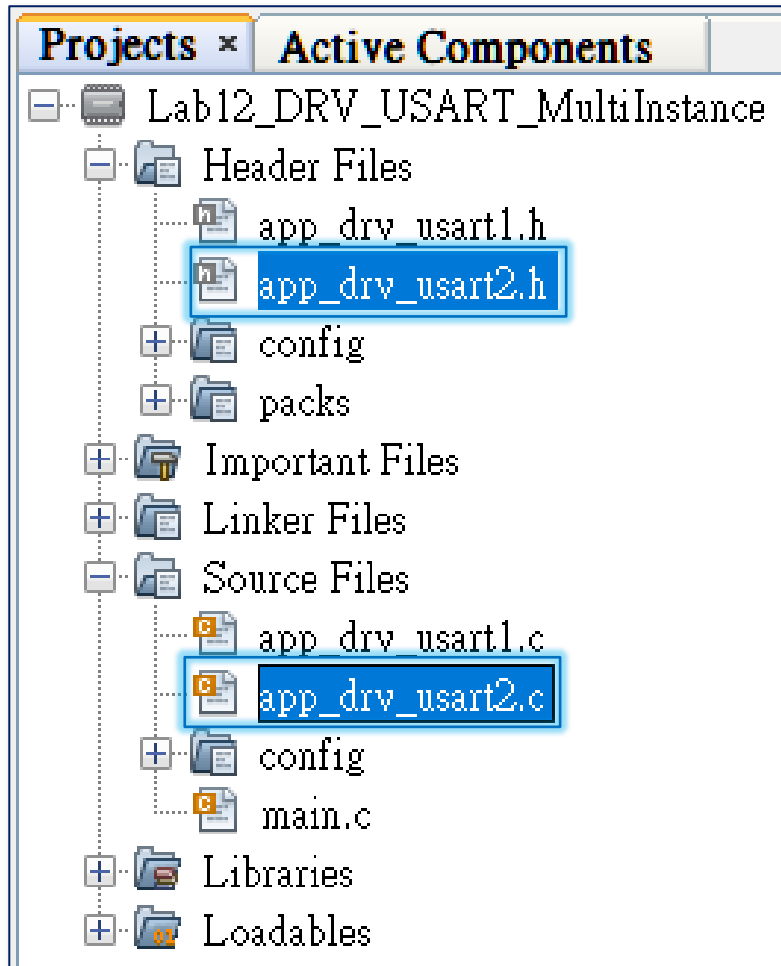
- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**2**” and given the Application Name to “**app_drv_usart2**”.

The image shows two windows from an IDE. The left window, titled 'Project Graph*', displays a block diagram of a system. It includes components like EVSYS, TC3, NVMCTRL, SERCOM5, and SERCOM2 on the left, and TIME, USART, and Instances on the right. The USART component is expanded to show two instances, Instance 0 and Instance 1, both labeled 'DRV_USART'. A green box highlights the 'Core' component, which is labeled 'Harmony Core Service' and 'Core Service'. A red text overlay at the bottom of this window says 'Rename file to “app_drv_usart2”'.

The right window, titled 'Configuration Options*', shows the configuration for the 'Core' component. A red text overlay at the top of this window says 'Increase Number of Application'. Under the 'Application Configuration' section, the 'Number of Applications' is set to '2'. Below this, there are two application configurations. The first is 'Application 0 Configuration' with the name 'app_drv_usart1'. The second is 'Application 1 Configuration' with the name 'app_drv_usart2', which is highlighted with a blue box. Both applications have 'Enable Application Configuration' checked.

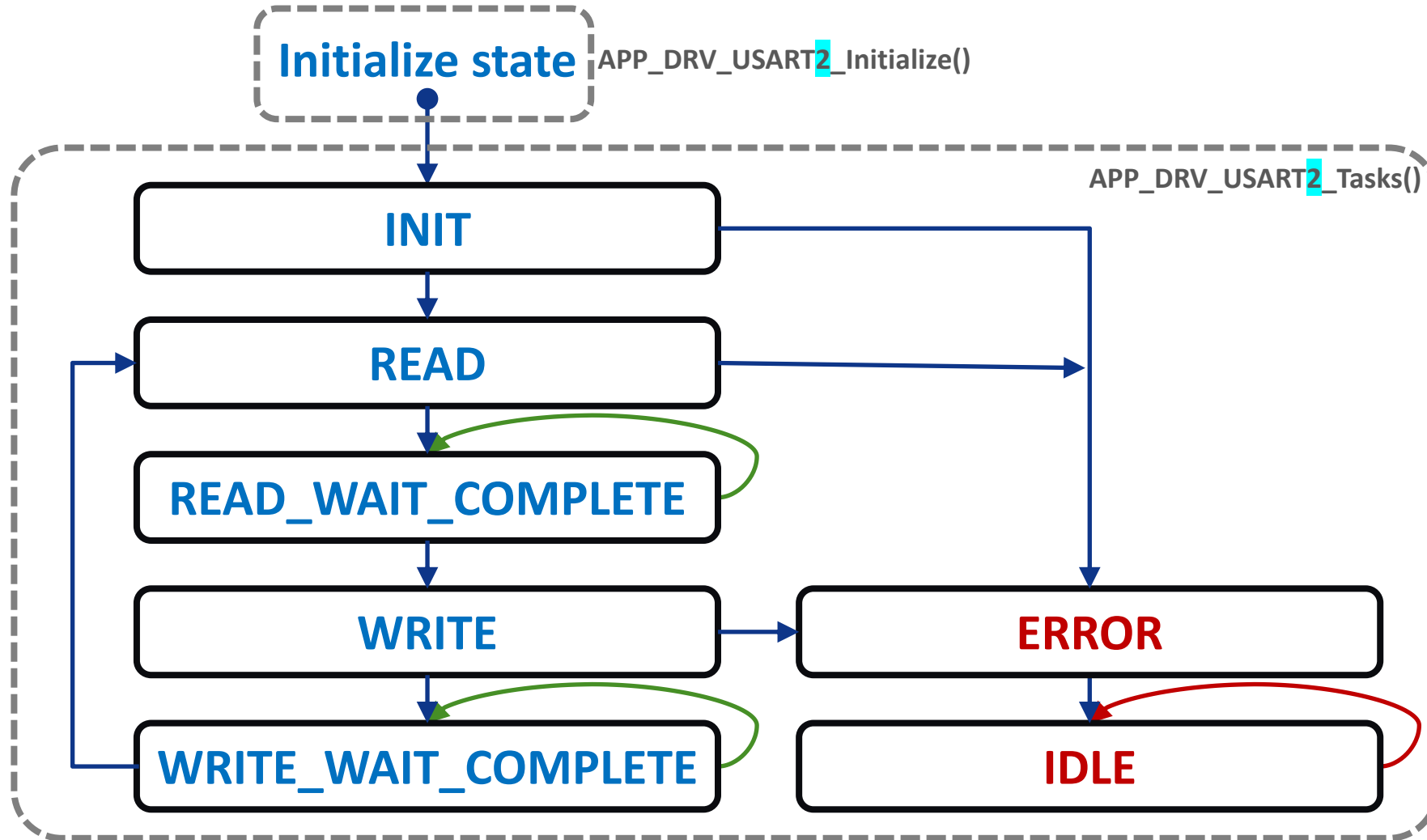
⚙️ Lab12 Multi-Instance UART Communication

- The new APP file `app_drv_usart2.c` will use to implement the state machine of USART Driver **instance 1**.



⚙️ Lab12 Multi-Instance UART Communication

- Refer the state machine of USART1 and change to USART2 in “app_drv_usart2.c”



⚙️ Lab12 Multi-Instance UART Communication

- Add more states in “app_drv_usart2.h”.

```
#ifndef _APP_DRV_USART2_H
#define _APP_DRV_USART2_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_USART2_STATE_INIT=0,
    // APP_DRV_USART2_STATE_SERVICE_TASKS,
    APP_DRV_USART2_STATE_READ,
    APP_DRV_USART2_STATE_READ_WAIT_COMPLETE,
    APP_DRV_USART2_STATE_WRITE,
    APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE,
    APP_DRV_USART2_STATE_ERROR,
    APP_DRV_USART2_STATE_IDLE,
} APP_DRV_USART2_STATES;

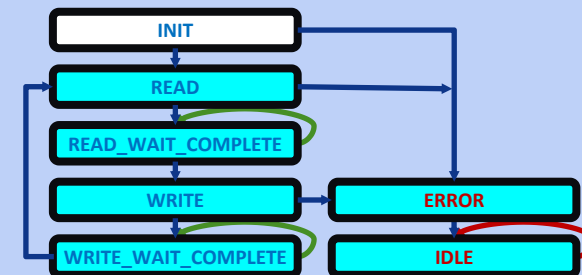
typedef struct
{
    APP_DRV_USART2_STATES state;
} APP_DRV_USART2_DATA;

void APP_DRV_USART2_Initialize ( void );
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



⚙️ Lab12 Multi-Instance UART Communication

- Construct new states flow of task in “app_drv_usart2.c”.

```
void APP_DRV_USART2_Tasks ( void )  
{
```

```
    switch ( app_drv_usart2Data.state )  
    {
```

```
        case APP_DRV_USART2_STATE_INIT:
```

```
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;  
            break;
```

```
        case APP_DRV_USART2_STATE_READ:
```

```
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ_WAIT_COMPLETE;  
            break;
```

```
        case APP_DRV_USART2_STATE_READ_WAIT_COMPLETE:
```

```
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE;  
            break;
```

```
        case APP_DRV_USART2_STATE_WRITE:
```

```
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE;  
            break;
```

```
        case APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE:
```

```
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;  
            break;
```

```
        case APP_DRV_USART2_STATE_ERROR:
```

```
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_IDLE;  
            break;
```

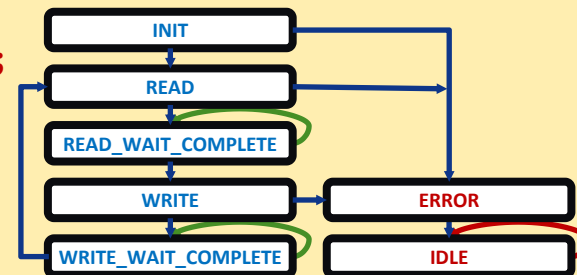
```
        case APP_DRV_USART2_STATE_IDLE:
```

```
            break;
```

```
    }
```

```
}
```

Overwrite original example code



⚙️ Lab12 Multi-Instance UART Communication

- Header include and Global Declarations in “app_drv_usart2.c”.

Keep this line

```
#include "app_drv_usart2.h"
#include "driver/usart/drv_usart.h"
#include <stdio.h>
#include <string.h>

// *****
// *****
// Section: Global Data Definitions
// *****
// *****

DRV_HANDLE  Uart2Handle;
DRV_USART_BUFFER_HANDLE Uart2WriteTransferHandle;
DRV_USART_BUFFER_HANDLE Uart2ReadTransferHandle;
volatile int Uart2WriteComplete = false;
volatile int Uart2ReadComplete = false;
char Uart2WriteBuf[50];
char Uart2ReadBuf[50];

// *****
/* Application Data

Summary:
    Holds application data

Description:
    This structure holds the application's data.

...
*/

APP_DRV_USART2_DATA app_drv_usart2Data;
```


⚙️ Lab12 Multi-Instance UART Communication

- Add Driver Open, Write and Transfer Status **Polling** Operation.

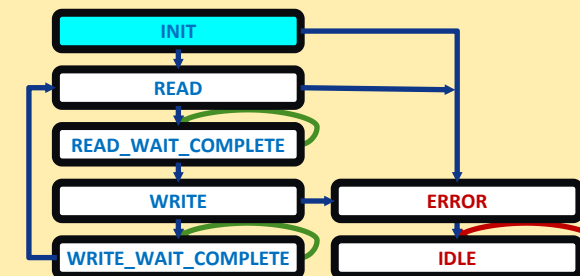
```
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        case APP_DRV_USART2_STATE_INIT:
            Uart2Handle = DRV_USART_Open(DRV_USART_INDEX_1, DRV_IO_INTENT_READWRITE);
            if(Uart2Handle == DRV_HANDLE_INVALID)
            {
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_ERROR;
            }
            else
            {
                sprintf(Uart2WriteBuf, "DRV_USART2: Hello world\r\n");
                DRV_USART_WriteBufferAdd(Uart2Handle, Uart2WriteBuf, strlen(Uart2WriteBuf),
                                         &Uart2WriteTransferHandle);
                while( DRV_USART_BufferStatusGet(Uart2WriteTransferHandle)!=
                     DRV_USART_BUFFER_EVENT_COMPLETE ) {}

                app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;
            }
        }
    }
    break;
}
```

**Transfer Status Polling operation
to check status of USART write**

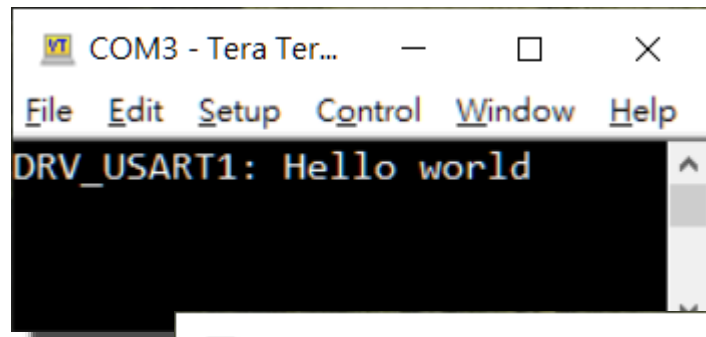
Keep this line

```
case APP_DRV_USART2_STATE_READ:
...
case APP_DRV_USART2_STATE_READ_WAIT_COMPLETE:
...
case APP_DRV_USART2_STATE_WRITE:
...
case APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE:
...
case APP_DRV_USART2_STATE_ERROR:
...
...
```

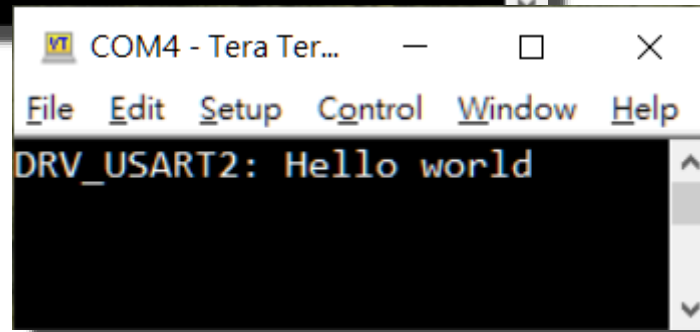


⚙️ Lab12 Multi-Instance UART Communication

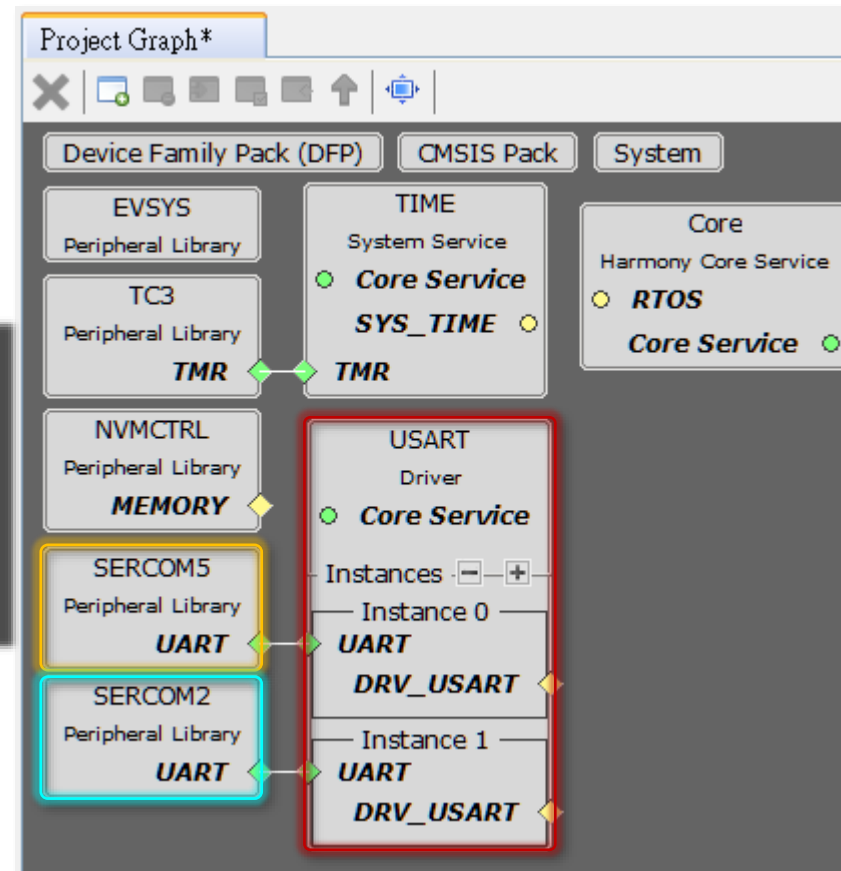
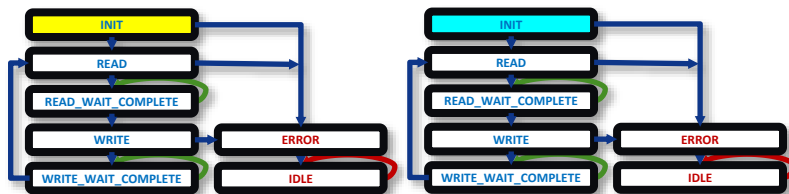
- Compiler and programming your project to EVB.
- Open two terminal to check the message output from Driver
Instance 0 (SERCOM5) and **Instance 1 (SERCOM2)**.



COM3 - Tera Ter...
File Edit Setup Control Window Help
DRV_USART1: Hello world



COM4 - Tera Ter...
File Edit Setup Control Window Help
DRV_USART2: Hello world



⚙️ Lab12 Multi-Instance UART Communication

- Implement Driver Transfer Status **Callback** Operation.

```
// Section: Application Callback Functions
// *****
void Uart2TransferHandler( DRV_USART_BUFFER_EVENT event,
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
{
    switch( event )
    {
        case DRV_USART_BUFFER_EVENT_PENDING: break;
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            break;
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:
        case DRV_USART_BUFFER_EVENT_ERROR:
        case DRV_USART_BUFFER_EVENT_HANDLE_INVALID:
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_ERROR;
            break;
    }
}

...
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        case APP_DRV_USART2_STATE_INIT:
            ...
        else
        {
            while( DRV_USART_BufferStatusGet(Uart2WriteTransferHandle)!=
                  DRV_USART_BUFFER_EVENT_COMPLETE ) {}
        }
    }
}

Callback Register DRV_USART_BufferEventHandlerSet(Uart2Handle, Uart2TransferHandler, 0);
app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;
```

**Catch Events in
Transfer Status Callback
Function and handle
Error event**

⚙️ Lab12 Multi-Instance UART Communication

- Implement state **READ** to read one byte from UART console.

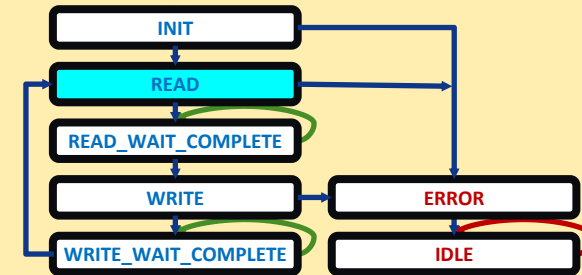
```
void Uart2TransferHandler( DRV_USART_BUFFER_EVENT event,  
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
```

```
{  
    switch( event )  
    {  
        ...  
        case DRV_USART_BUFFER_EVENT_COMPLETE:  
            if(bufferHandle == Uart2ReadTransferHandle)  
                Uart2ReadComplete = true;  
            break;  
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:  
            ...  
    }  
}
```

Keep this line

```
void APP_DRV_USART2_Tasks ( void )  
{  
    switch ( app_drv_usart2Data.state )  
    { ...  
        case APP_DRV_USART2_STATE_READ:  
            DRV_USART_ReadBufferAdd(Uart2Handle, Uart2ReadBuf, 1, &Uart2ReadTransferHandle);  
            if(Uart2ReadTransferHandle == DRV_USART_BUFFER_HANDLE_INVALID)  
            {  
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_ERROR;  
            }  
            else  
            {  
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ_WAIT_COMPLETE;  
            }  
            break;  
        case APP_DRV_USART2_STATE_READ_WAIT_COMPLETE:
```

Manage Read Complete flag while
Read Transfer Complete Events



⚙️ Lab12 Multi-Instance UART Communication

- Implement state **READ_WAIT_COMPLETE**.

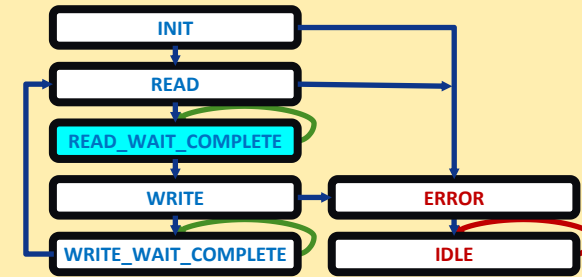
```
void Uart2TransferHandler( DRV_USART_BUFFER_EVENT event,
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart2ReadTransferHandle)
                Uart2ReadComplete = true;
            break;
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:
            ...
    }
}
```

```
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        ...
        case APP_DRV_USART2_STATE_READ:
            ...

        case APP_DRV_USART2_STATE_READ_WAIT_COMPLETE:
            if(Uart2ReadComplete == true)
            {
                Uart2ReadComplete = false;
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE;
            }
            break;

        case APP_DRV_USART2_STATE_WRITE:
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE;
```

Keep this line



State non blocking while waiting Read Complete

⚙️ Lab12 Multi-Instance UART Communication

- Implement state **WRITE** to output “Received : X” to console.

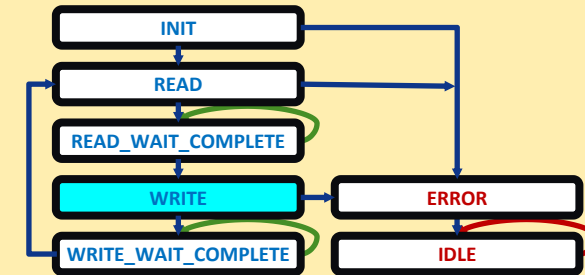
```
void Uart2TransferHandler( ... )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart2ReadTransferHandle)
                Uart2ReadComplete = true;
            if(bufferHandle == Uart2WriteTransferHandle)
                Uart2WriteComplete = true;
            break;
        ...
    }
}
```

Keep this line

```
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        ...
        case APP_DRV_USART2_STATE_WRITE:
            sprintf(Uart2WriteBuf, "Receive : %s (DRV_USART2)\r\n", Uart2ReadBuf);
            DRV_USART_WriteBufferAdd(Uart2Handle, Uart2WriteBuf, strlen(Uart2WriteBuf),
                                     &Uart2WriteTransferHandle);
            if(Uart2WriteTransferHandle == DRV_USART_BUFFER_HANDLE_INVALID)
            {
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_ERROR;
            }
            else
            {
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE;
            }
            break;
        case APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE:
```

Keep this line

Manage Write Complete flag while
Write Transfer Complete Events



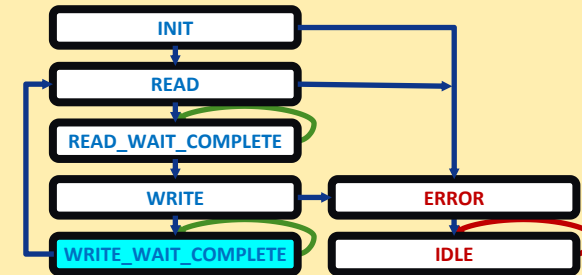
⚙️ Lab12 Multi-Instance UART Communication

- Implement state **WRITE_WAIT_COMPLETE**

```
void Uart2TransferHandler( ... )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart2ReadTransferHandle)
                Uart2ReadComplete = true;
            if(bufferHandle == Uart2WriteTransferHandle)
                Uart2WriteComplete = true;
            break;
        ...
    }
}

void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        ...
        case APP_DRV_USART2_STATE_WRITE:
            ...
            case APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE:
                if(Uart2WriteComplete == true)
                {
                    Uart2WriteComplete = false;
                    app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;
                }
                break;
            case APP_DRV_USART2_STATE_ERROR:
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_IDLE;
                break;
            case APP_DRV_USART2_STATE_IDLE:
                ...
            }
    }
}
```

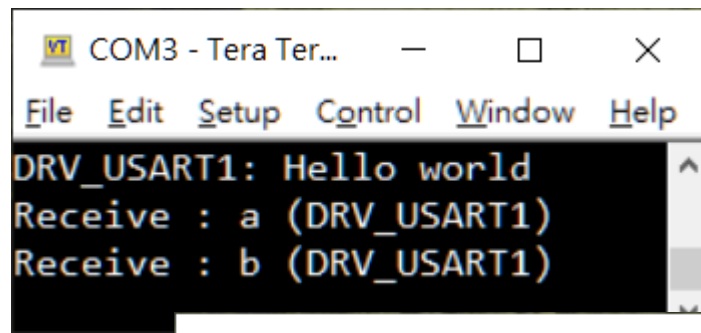
Keep this line



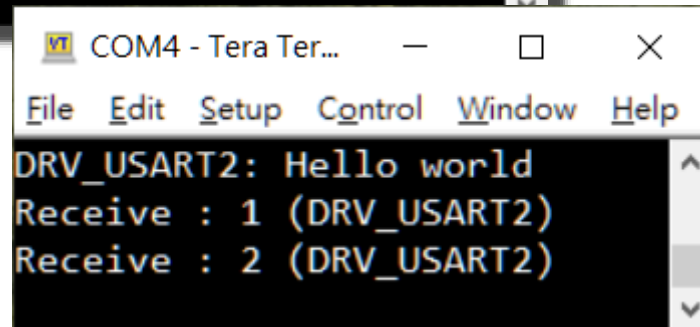
State non blocking while waiting Write Complete

✖ Lab12 Multi-Instance UART Communication

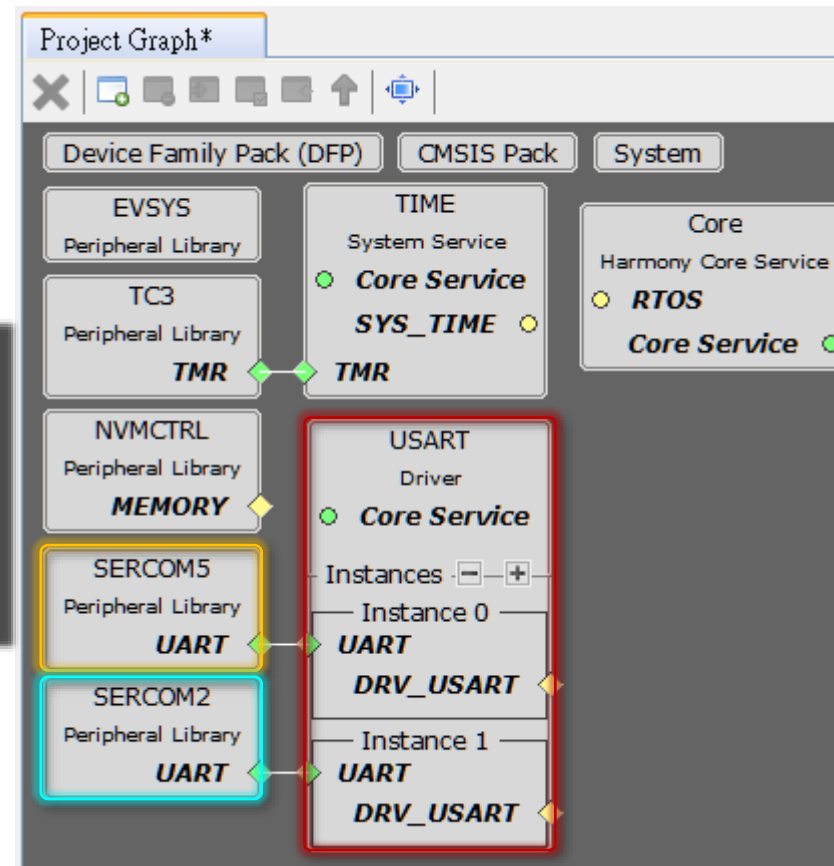
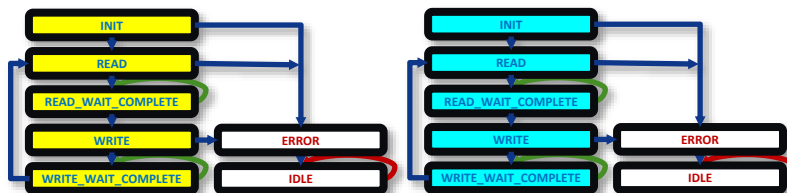
- Compiler and programming your project to EVB.
- Check the UART output from two console in terminal.
- Type any character in console and check the response.



```
COM3 - Tera Ter...  
File Edit Setup Control Window Help  
DRV_USART1: Hello world  
Receive : a (DRV_USART1)  
Receive : b (DRV_USART1)
```

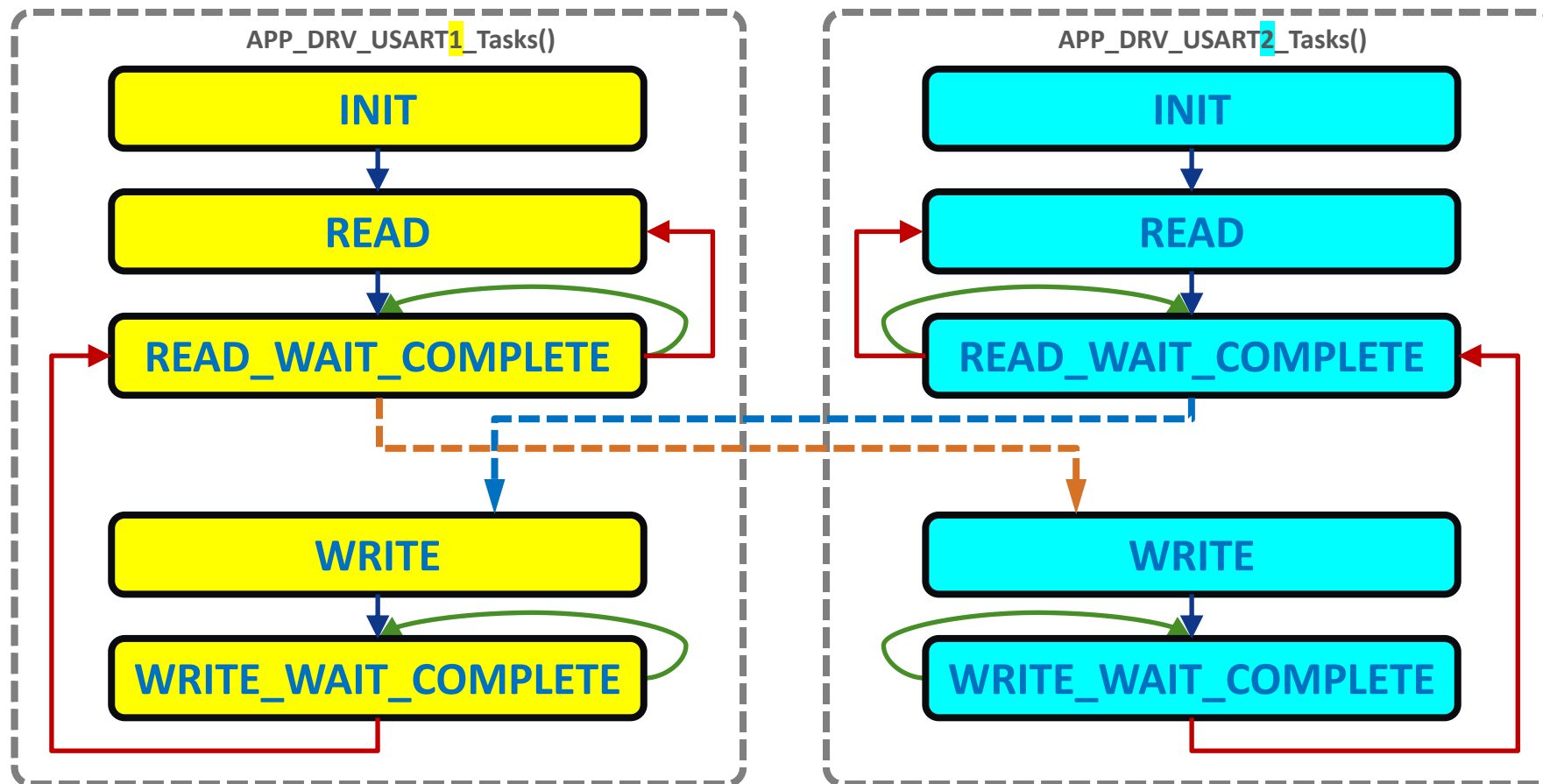


```
COM4 - Tera Ter...  
File Edit Setup Control Window Help  
DRV_USART2: Hello world  
Receive : 1 (DRV_USART2)  
Receive : 2 (DRV_USART2)
```



⚙️ Lab12 Multi-Instance UART Communication

- Let's consider a state machine that could full fill Lab's request. To receive one character from console and echo it to opposite console.



⚙️ Lab12 Multi-Instance UART Communication

- Extern USART1 state and receive buffer in `app_drv_usart1.h`

```
#ifndef _APP_DRV_USART1_H
#define _APP_DRV_USART1_H

...
void APP_DRV_USART1_Initialize ( void );
void APP_DRV_USART1_Tasks( void );
void USART1printf( const char *format, ... );

extern APP_DRV_USART1_DATA app_drv_usart1Data;
extern char Uart1ReadBuf[50];

...
```

Extern USART1
state and variable

`app_drv_usart1.h`

- Extern USART2 state and receive buffer in `app_drv_usart2.h`

```
#ifndef _APP_DRV_USART2_H
#define _APP_DRV_USART2_H

...
void APP_DRV_USART2_Initialize ( void );
void APP_DRV_USART2_Tasks( void );

extern APP_DRV_USART2_DATA app_drv_usart2Data;
extern char Uart2ReadBuf[50];

...
```

Extern USART2
state and variable

`app_drv_usart2.h`

⚙️ Lab12 Multi-Instance UART Communication

- Include `app_drv_usart2.h` in `app_drv_usart1.c`

```
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "driver/usart/drv_usart.h"
#include <stdio.h>
#include <string.h>
```

Include USART2 header file
to get received byte
and the state flow control

```
// *****
// *****
// Section: Global Data Definitions
// *****
// *****
DRV_HANDLE Uart1Handle;
DRV_USART_BUFFER_HANDLE Uart1WriteTransferHandle;
...
```

`app_drv_usart1.c`

- Include `app_drv_usart1.h` in `app_drv_usart2.c`

```
#include "app_drv_usart2.h"
#include "app_drv_usart1.h"
#include "driver/usart/drv_usart.h"
#include <stdio.h>
#include <string.h>
```

Include USART1 header file
to get received byte
and the state flow control

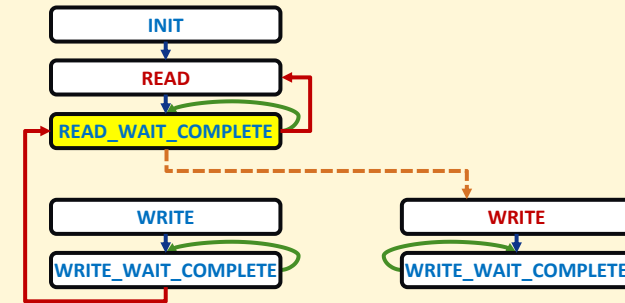
```
// *****
// *****
// Section: Global Data Definitions
// *****
// *****
DRV_HANDLE Uart2Handle;
DRV_USART_BUFFER_HANDLE Uart2WriteTransferHandle;
...
```

`app_drv_usart2.c`

⚡ Lab12 Multi-Instance UART Communication

- Modify state **READ_WAIT_COMPLETE** in “app_drv_usart1.c”.

```
void Uart1TransferHandler( DRV_USART_BUFFER_EVENT event,
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart1ReadTransferHandle)
                Uart1ReadComplete = true;
            break;
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:
            ...
    }
}
```



```
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    { ...
    case APP_DRV_USART1_STATE_READ:
        ...

    case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
        if(Uart1ReadComplete == true)
        {
            Uart1ReadComplete = false;
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE;
        }
        break;

    case APP_DRV_USART1_STATE_WRITE:
```

Modify this Line

State self to **READ**
for next receive

State **USART2** to **WRITE**
for message output

app_drv_usart1.c

⚙️ Lab12 Multi-Instance UART Communication

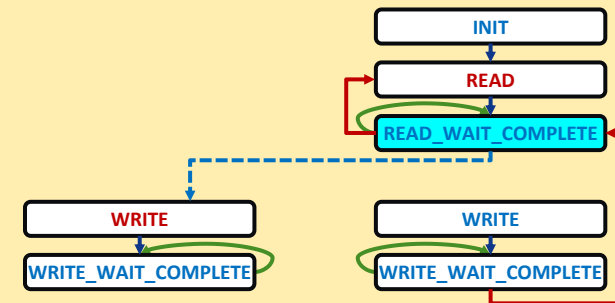
- Modify state **READ_WAIT_COMPLETE** in “app_drv_usart2.c”.

```
void Uart2TransferHandler( DRV_USART_BUFFER_EVENT event,
                          DRV_USART_BUFFER_HANDLE bufferHandle, uintptr_t context )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart2ReadTransferHandle)
                Uart2ReadComplete = true;
            break;
        case DRV_USART_BUFFER_EVENT_HANDLE_EXPIRED:
            ...
    }
}
```

```
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        ...
        case APP_DRV_USART2_STATE_READ:
            ...

        case APP_DRV_USART2_STATE_READ_WAIT_COMPLETE:
            if(Uart2ReadComplete == true)
            {
                Uart2ReadComplete = false;
                Modify this Line app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_WRITE;
            }
            break;

        case APP_DRV_USART2_STATE_WRITE:
```



State self to **READ**
for next receive

State **USART1** to **WRITE**
for message output

app_drv_usart2.c

⚙️ Lab12 Multi-Instance UART Communication

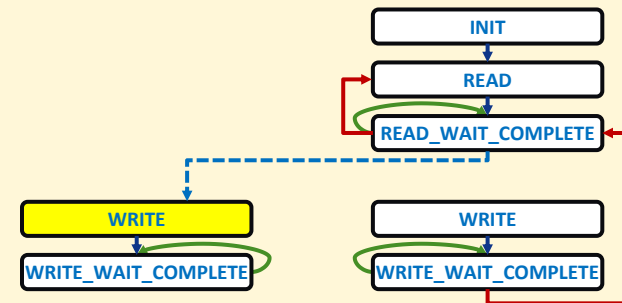
- Modify state **WRITE** in “app_drv_usart1.c”.

```
void Uart1TransferHandler( ... )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart1ReadTransferHandle)
                Uart1ReadComplete = true;
            if(bufferHandle == Uart1WriteTransferHandle)
                Uart1WriteComplete = true;
            break;
        ...
    }
}
```

```
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
        ...
```

```
        case APP_DRV_USART1_STATE_WRITE:
            Modify this Line sprintf(Uart1WriteBuf, "Receive : %s (DRV_USART2)\r\n", Uart2ReadBuf);
            DRV_USART_WriteBufferAdd(Uart1Handle, Uart1WriteBuf, strlen(Uart1WriteBuf),
                                     &Uart1WriteTransferHandle);
            if(Uart1WriteTransferHandle == DRV_USART_BUFFER_HANDLE_INVALID)
            {
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_ERROR;
            }
            else
            {
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE;
            }
            break;
```

```
        case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
```



Output **USART2** received byte
in **USART1**

app_drv_usart1.c

⚙️ Lab12 Multi-Instance UART Communication

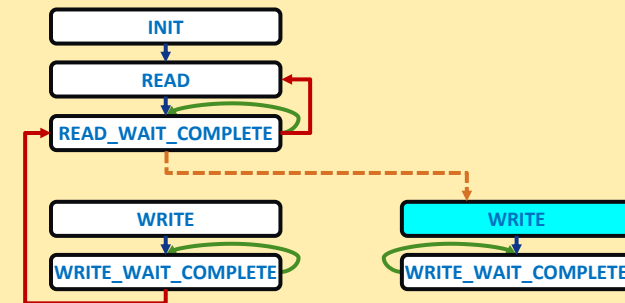
- Modify state **WRITE** in “app_drv_usart2.c”.

```
void Uart2TransferHandler( ... )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart2ReadTransferHandle)
                Uart2ReadComplete = true;
            if(bufferHandle == Uart2WriteTransferHandle)
                Uart2WriteComplete = true;
            break;
        ...
    }
}
```

```
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        ...
```

```
        case APP_DRV_USART2_STATE_WRITE:
            Modify this Line sprintf(Uart2WriteBuf, "Receive : %s (DRV_USART1)\r\n", Uart1ReadBuf);
            DRV_USART_WriteBufferAdd(Uart2Handle, Uart2WriteBuf, strlen(Uart2WriteBuf),
                                     &Uart2WriteTransferHandle);
            if(Uart2WriteTransferHandle == DRV_USART_BUFFER_HANDLE_INVALID)
            {
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_ERROR;
            }
            else
            {
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE;
            }
            break;
```

```
        case APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE:
```



Output **USART1** received byte
in **USART2**

app_drv_usart2.c

⚙️ Lab12 Multi-Instance UART Communication

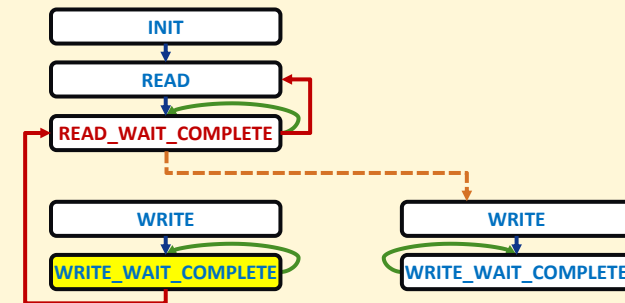
- Modify state **WRITE_WAIT_COMPLETE** in “app_drv_usart1.c”.

```
void Uart1TransferHandler( ... )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart1ReadTransferHandle)
                Uart1ReadComplete = true;
            if(bufferHandle == Uart1WriteTransferHandle)
                Uart1WriteComplete = true;
            break;
        ...
    }
}
```

```
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
        ...
        case APP_DRV_USART1_STATE_WRITE:
            ...
            case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
                if(Uart1WriteComplete == true)
                {
                    Uart1WriteComplete = false;
                    Modify this Line app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ_WAIT_COMPLETE;
                }
                break;

            case APP_DRV_USART1_STATE_ERROR:
                app_drv_usart1Data.state = APP_DRV_USART1_STATE_IDLE;
                break;

            case APP_DRV_USART1_STATE_IDLE:
                ...
            }
    }
}
```



Go **READ_WAIT_COMPLETE**
for next receive complete

app_drv_usart1.c

✖ Lab12 Multi-Instance UART Communication

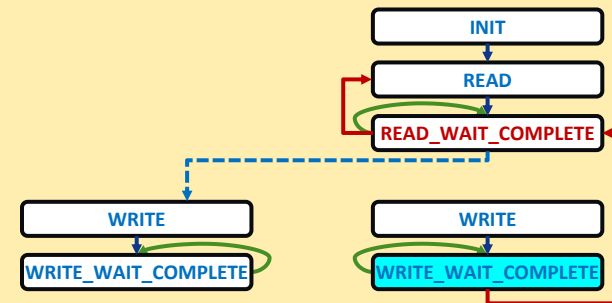
- Modify state **WRITE_WAIT_COMPLETE** in “app_drv_usart2.c”.

```
void Uart2TransferHandler( ... )
{
    switch( event )
    {
        ...
        case DRV_USART_BUFFER_EVENT_COMPLETE:
            if(bufferHandle == Uart2ReadTransferHandle)
                Uart2ReadComplete = true;
            if(bufferHandle == Uart2WriteTransferHandle)
                Uart2WriteComplete = true;
            break;
        ...
    }
}
```

```
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    {
        ...
        case APP_DRV_USART2_STATE_WRITE:
            ...
            case APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE:
                if(Uart2WriteComplete == true)
                {
                    Uart2WriteComplete = false;
                    Modify this Line app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ_WAIT_COMPLETE;
                }
                break;

            case APP_DRV_USART2_STATE_ERROR:
                app_drv_usart2Data.state = APP_DRV_USART2_STATE_IDLE;
                break;

            case APP_DRV_USART2_STATE_IDLE:
```

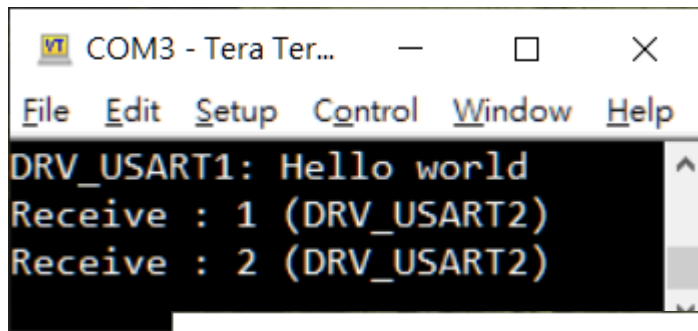


Go **READ_WAIT_COMPLETE**
for next receive complete

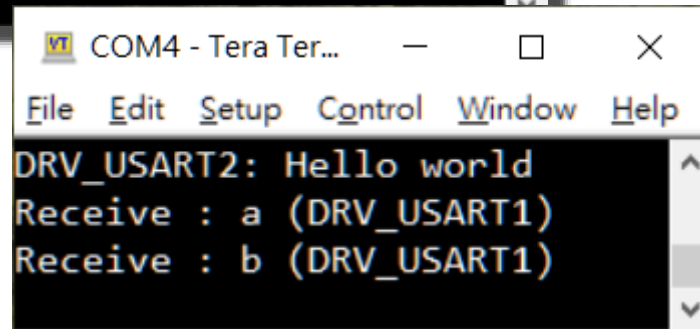
app_drv_usart2.c

⚙️ Lab12 Multi-Instance UART Communication

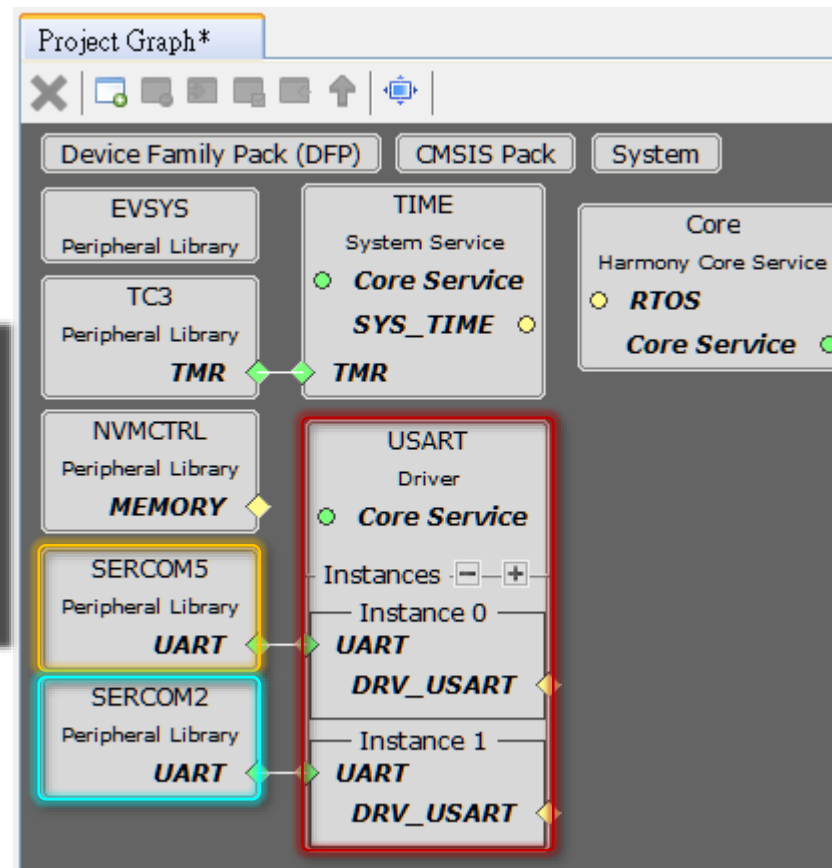
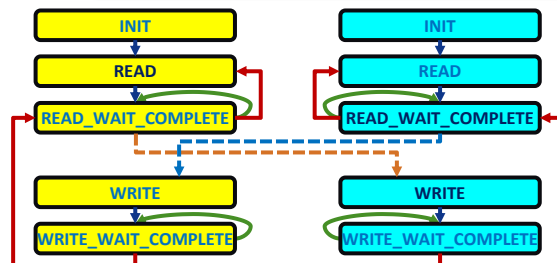
- Compiler and programming your project to EVB.
- Check the UART output from two console in terminal.
- Type any character in console and check the response.



```
COM3 - Tera Ter...  
File Edit Setup Control Window Help  
DRV_USART1: Hello world  
Receive : 1 (DRV_USART2)  
Receive : 2 (DRV_USART2)
```



```
COM4 - Tera Ter...  
File Edit Setup Control Window Help  
DRV_USART2: Hello world  
Receive : a (DRV_USART1)  
Receive : b (DRV_USART1)
```



⚙️ Lab12 Multi-Instance UART Communication

- **Additional exercise**

- To implement a **raw bytes write** function of **USART2** in **app_drv_usart2.c**
- This function is different to the **string operation** of **USART1printf()**

```
...
DRV_HANDLE  Uart2Handle;
DRV_USART_BUFFER_HANDLE Uart2WriteTransferHandle;
DRV_USART_BUFFER_HANDLE Uart2ReadTransferHandle;
volatile int Uart2WriteComplete = false;
volatile int Uart2ReadComplete = false;
char Uart2WriteBuf[50];
char Uart2ReadBuf[50];

APP_DRV_USART2_DATA app_drv_usart2Data;

void USART2write( unsigned char *pdata, int len )
{
    if( len > 0 && len < sizeof(Uart2WriteBuf) )
    {
        memcpy(Uart2WriteBuf, pdata, len);

        DRV_USART_WriteBufferAdd(Uart2Handle, Uart2WriteBuf,
                                   len, &Uart2WriteTransferHandle);

        while( DRV_USART_BufferStatusGet(Uart2WriteTransferHandle)!=
                DRV_USART_BUFFER_EVENT_COMPLETE ) {}
    }
}

void Uart2TransferHandler( DRV_USART_BUFFER_EVENT event, ... ) { ... }
```

⚙️ Lab12 Multi-Instance UART Communication

- Replace “DRV_USART2: Hello World” output by use USART2write()

```
void APP_DRV_USART2_Tasks ( void )  
{  
    switch ( app_drv_usart2Data.state )  
    {  
    case APP_DRV_USART2_STATE_INIT:  
        Uart2Handle = DRV_USART_Open(DRV_USART_INDEX_1, DRV_IO_INTENT_READWRITE);  
        if(Uart2Handle == DRV_HANDLE_INVALID)  
        {  
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_ERROR;  
        }  
        else  
        {  
            USART2write((unsigned char*)"DRV_USART2: Hello world\r\n", 25);  
            // sprintf(Uart2WriteBuf, "DRV_USART2: Hello world\r\n");  
            // DRV_USART_WriteBufferAdd(Uart2Handle, Uart2WriteBuf, strlen(Uart2WriteBuf),  
            //                           &Uart2WriteTransferHandle);  
            // while( DRV_USART_BufferStatusGet(Uart2WriteTransferHandle) !=  
            //         DRV_USART_BUFFER_EVENT_COMPLETE ) {}  
  
            DRV_USART_BufferEventHandlerSet(Uart2Handle, Uart2TransferHandler, 0);  
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;  
        }  
        ...  
    }  
}
```

app_drv_usart2.c

- Declare prototype in app_drv_usart2.h for external reference use.

```
void APP_DRV_USART2_Tasks( void );  
  
void USART2write( unsigned char *pdata, int len );  
  
extern APP_DRV_USART2_DATA app_drv_usart2Data;  
extern char Uart2ReadBuf[50];  
...
```

app_drv_usart2.h

I2C Driver Library

Harmony Driver

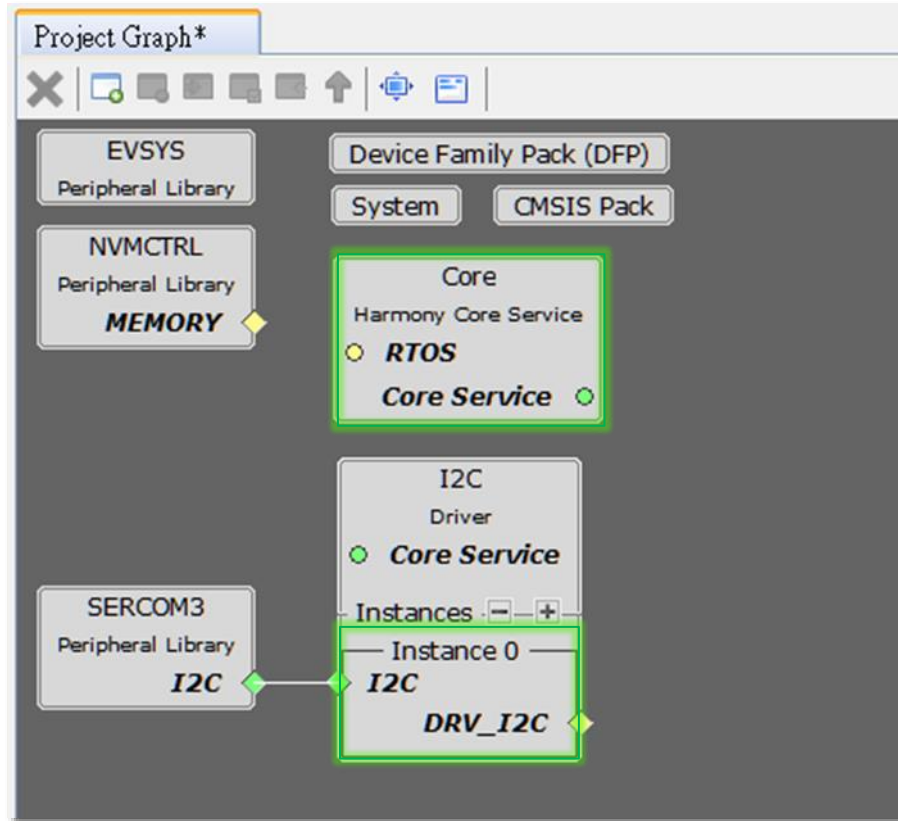
I2C Driver Library

- **Introduction**

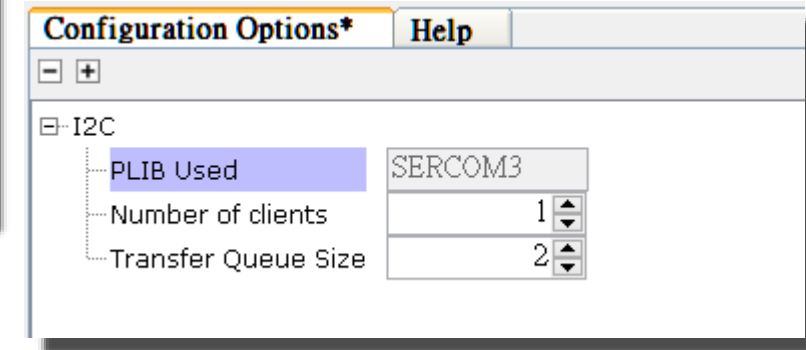
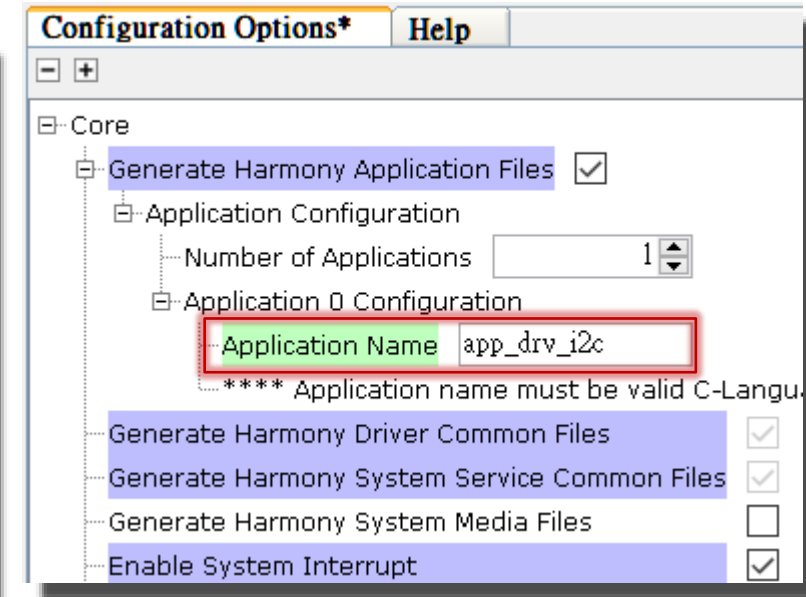
This driver library provides application ready routines to read and write data using the I²C protocol, thus minimizing developer's awareness of the working of the I²C protocol.

- Supports I²C master functionality.
- Provides write, read and write-read transfers.
- Support multi-client and multi-instance operation.
- Provides data transfer events.
- Supports Asynchronous and Synchronous modes of operation.

I2C Driver Library



**Create an Application for
Driver State Machine**



I2C Driver Library

API functions of the I2C Driver library.

- **System Functions**

DRV_I2C_Initialize

Initializes the I2C instance for the specified driver index.

DRV_I2C_Status

Gets the current status of the I2C driver module.

- **Core Client Functions**

DRV_I2C_Open

Opens the specified I2C driver instance and returns a handle to it.

DRV_I2C_Close

Closes an opened-instance of the I2C driver.

DRV_I2C_TransferSetupSets

the dynamic transfer setup of the driver.

DRV_I2C_TransferEventHandlerSet

Allows a client to identify a transfer event handling function for the driver to call back when queued transfers have finished.

- **Data Transfer Functions**

DRV_I2C_ReadTransfer

This is a blocking function that performs a I2C read operation.

DRV_I2C_ReadTransferAdd

Queues a read operation.

DRV_I2C_WriteTransfer

This is a blocking function that performs a I2C write operation.

DRV_I2C_WriteTransferAdd

Queues a write operation.

DRV_I2C_WriteReadTransfer

A blocking function that performs a I2C write followed by a I2C read operation.

DRV_I2C_WriteReadTransferAdd

Queues a write followed by read operation.

DRV_I2C_TransferStatusGet

Returns the status of the write/read/write-read transfer request.

DRV_I2C_ForcedWriteTransfer

This is a blocking function that performs a I2C write operation.

DRV_I2C_ErrorGet

Gets the I2C hardware errors associated with the the transfer request.

DRV_I2C_ForcedWriteTransferAdd

Queues a write operation.

I2C Driver Library

API functions of the I2C Driver library. (cont.)

- Data Types and Constants

DRV_I2C_TRANSFER_HANDLE	Handle identifying a read, write or write followed by read transfer passed to the driver.
DRV_I2C_TRANSFER_EVENT	Identifies the possible events that can result from a buffer add request.
DRV_I2C_TRANSFER_EVENT_HANDLER	Pointer to a I2C Driver Transfer Event handler function
DRV_I2C_TRANSFER_HANDLE_INVALID	Definition of an invalid transfer handle.

I2C Driver Library

PLIB SERCOM I2C pad configurations.

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

Signal Name	Type	Description
PAD[0]	Digital I/O	SDA
PAD[1]	Digital I/O	SCL
PAD[2]	Digital I/O	SDA_OUT (4-wire operation)
PAD[3]	Digital I/O	SCL_OUT (4-wire operation)

Lab 13

Single-Client I2C

Digital Temperature Sensor

I2C Driver Library

✂ Lab13 Single-Client I2C Digital Temperature Sensor

- Try add **I2C Driver** through **PLIB SERCOM3 I2C** to communication the on-board **Digital Temperature Sensor MCP9800A5T**.

- **MCP9800 configuration:**

- **SDA** connect to **PA22** (**SERCOM3 PAD0**)
- **SCL** connect to **PA23** (**SERCOM3 PAD1**)
- Slave address is **0x4D**.
- **TA** register address is **0x00**.

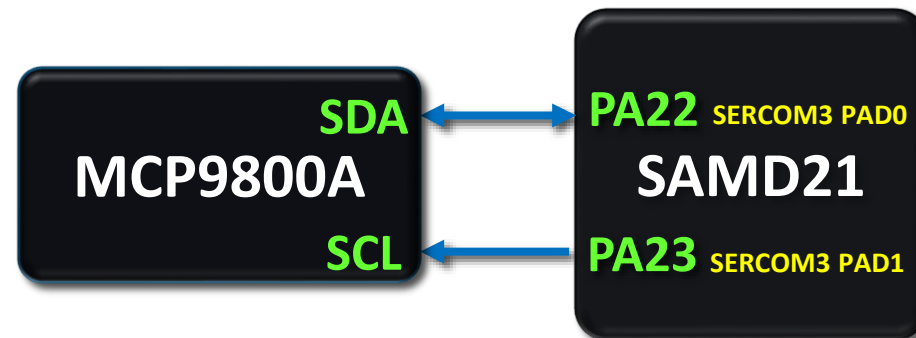
$$T_A = \text{Code} \times 2^{-4}$$

T_A = Ambient Temperature (°C)
Code = MCP9800 output in decimal

$$\text{Result} = (\text{TA}[0] \ll 4 | \text{TA}[1] \gg 4) / 16$$

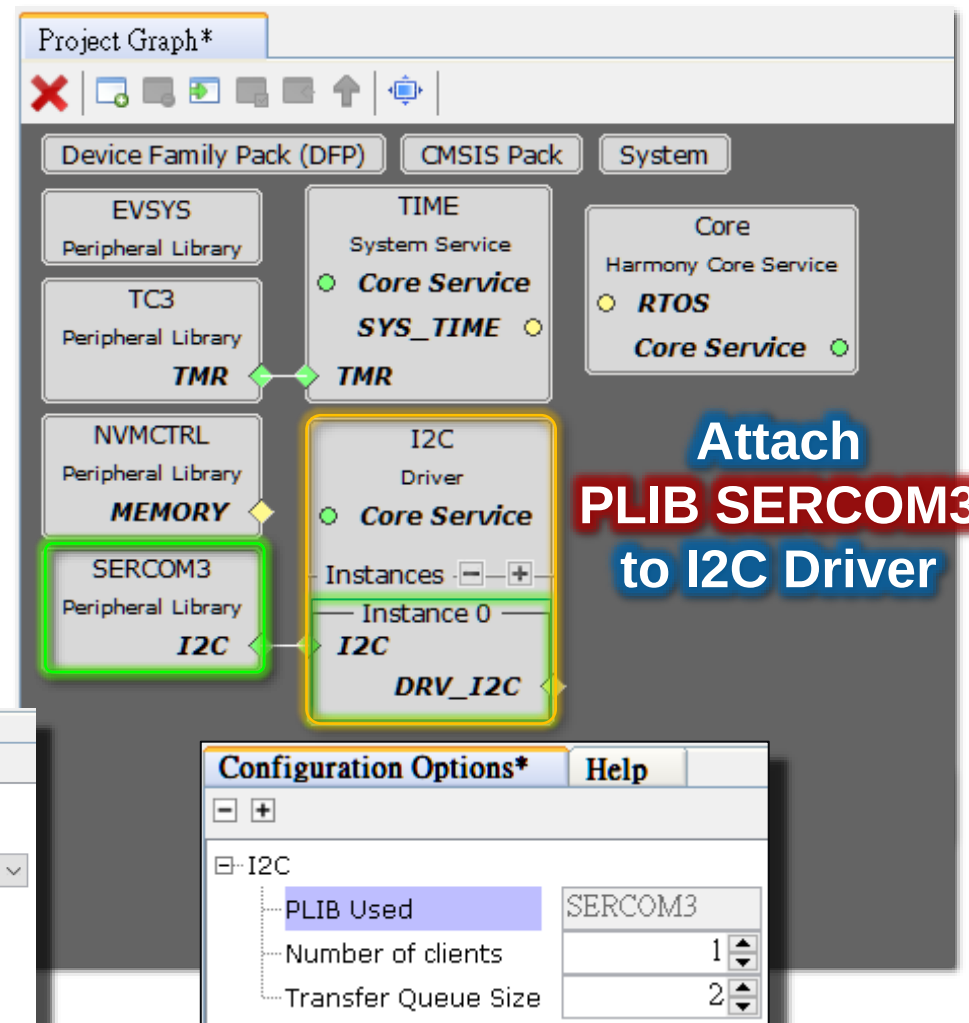
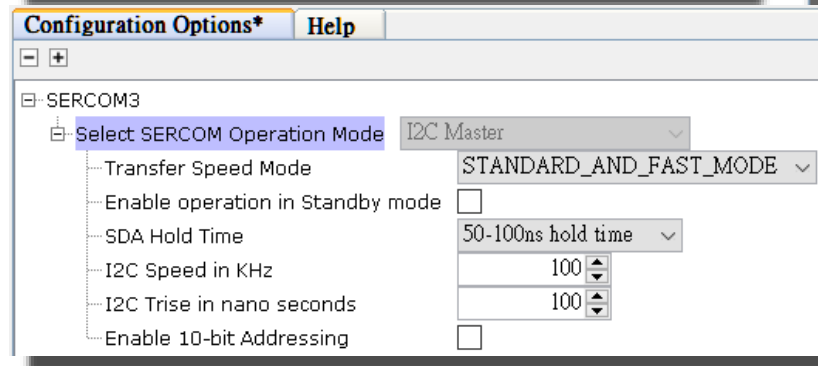
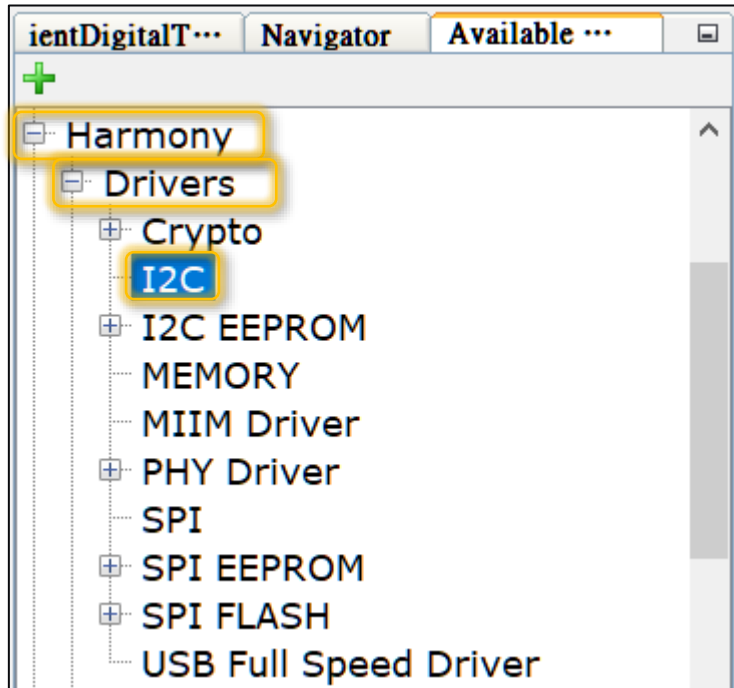
Register Pointer P1 P0	MSB/ LSB	Bit Assignment							
		7	6	5	4	3	2	1	0
Ambient Temperature Register (T _A)									
0 0	MSB	Sign	2 ⁶ °C	2 ⁵ °C	2 ⁴ °C	2 ³ °C	2 ² °C	2 ¹ °C	2 ⁰ °C
	LSB	2 ⁻¹ °C	2 ⁻² °C	2 ⁻³ °C	2 ⁻⁴ °C	0	0	0	0

TA[0]
TA[1]



✖ Lab13 Single-Client I2C Digital Temperature Sensor

- Add **I2C Driver** and attach **PLIB SERCOM3** as its hardware.



✖ Lab13 Single-Client I2C Digital Temperature Sensor

- Enable **PLIB SERCOM3** pads of I2C Driver **Instance 0**.

Project Graph* Pin Diagram × Pin Table × Pin Settings ×									
Order:	Ports ▾	Table View	☒ Easy View						
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
31	PA22		SERCOM3_PAD0 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾
32	PA23		SERCOM3_PAD1 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾

Configure PA22 as SERCOM3_PAD0
Configure PA23 as SERCOM3_PAD1

Project Graph* Pin Diagram × Pin Table × Pin Settings ×

Package: TQFP48 ▾

A

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
SERCOM3	SERCOM3_PAD0																									
	SERCOM3_PAD1																									
	SERCOM3_PAD2																									
	SERCOM3_PAD3																									

✖ Lab13 Single-Client I2C Digital Temperature Sensor

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**3**” and given the Application Name to “**app_drv_i2c1**”.

The image shows two windows from the IDE. The left window, titled 'Project Graph*', displays a system architecture diagram. It includes components like EVSYS, TC3, NVMCTRL, SERCOM3, TIME, and I2C. The I2C component is highlighted with a green box, and its instance 'DRV_I2C' is shown. The right window, titled 'Configuration Options*', shows the 'Core' configuration. Under 'Application Configuration', the 'Number of Applications' is set to 3. Below this, three application configurations are listed: 'Application 0 Configuration' (name: app_drv_usart1), 'Application 1 Configuration' (name: app_drv_usart2), and 'Application 2 Configuration' (name: app_drv_i2c1). The 'Application 2 Configuration' is highlighted with a blue box.

Project Graph*

Device Family Pack (DFP) CMSIS Pack System

EVSYS Peripheral Library

TC3 Peripheral Library

NVMCTRL Peripheral Library

SERCOM3 Peripheral Library

TIME System Service

Core Service

SYS_TIME

TMR

I2C Driver

Core Service

Instances - +

Instance 0

I2C

DRV_I2C

Configuration Options* Help

Increase Number of Application

Core

Generate Harmony Application Files ☒

Application Configuration

Number of Applications 3

Application 0 Configuration

Enable Application 0 Configuration ☒

Application Name app_drv_usart1

**** Application name must be valid C-Language identifier

Application 1 Configuration

Enable Application 1 Configuration ☒

Application Name app_drv_usart2

**** Application name must be valid C-Language identifier

Application 2 Configuration

Enable Application 2 Configuration ☒

Application Name app_drv_i2c1

**** Application name must be valid C-Language identifier

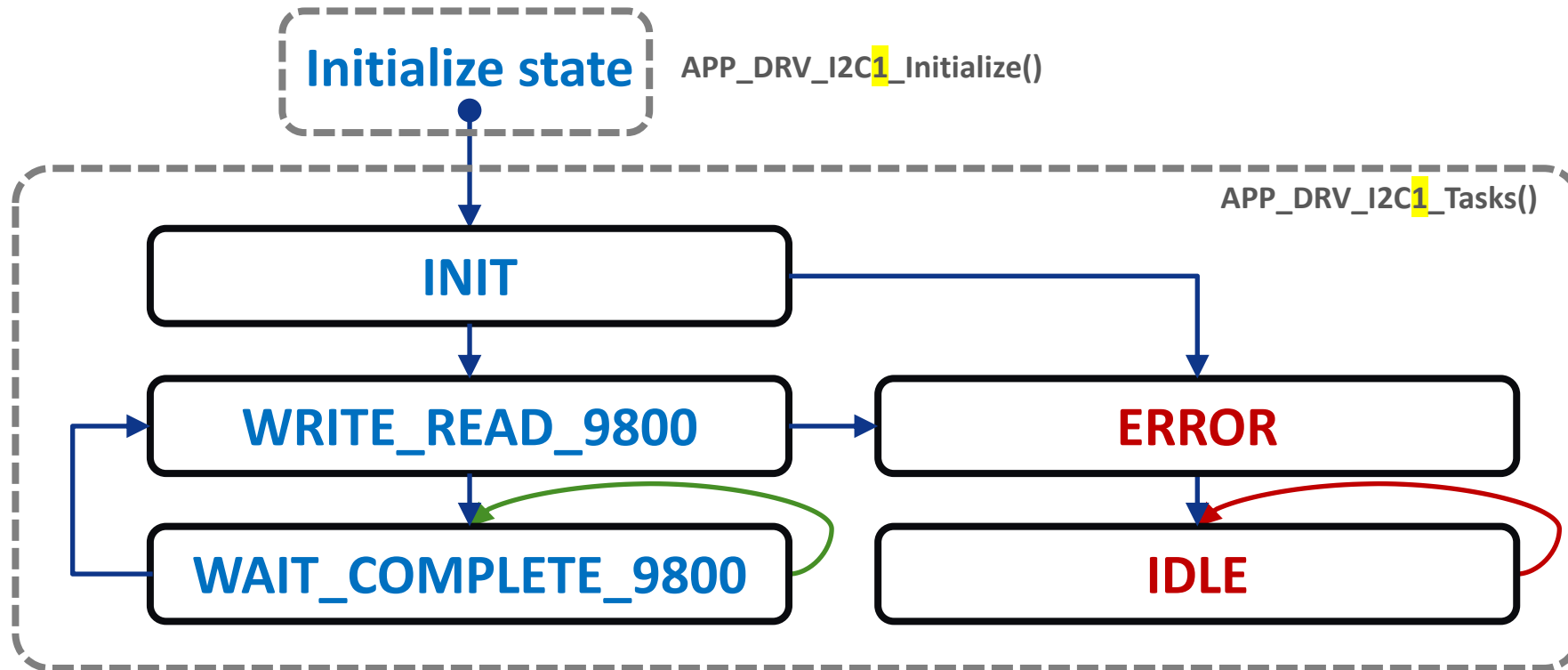
Generate Harmony Driver Common Files ☒

Generate Harmony System Service Common Files ☒

Rename file to “app_drv_i2c1”

⚙️ Lab13 Single-Client I2C Digital Temperature Sensor

- The state machine we want to implement in “`app_drv_i2c1.c`” as below.



⚙️ Lab13 Single-Client I2C Digital Temperature Sensor

- Add more states in “app_drv_i2c1.h”.

```
#ifndef _APP_DRV_I2C1_H
#define _APP_DRV_I2C1_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_I2C1_STATE_INIT=0,
    // APP_DRV_I2C1_STATE_SERVICE_TASKS,
    APP_DRV_I2C1_STATE_WRITE_READ_9800,
    APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800,
    APP_DRV_I2C1_STATE_ERROR,
    APP_DRV_I2C1_STATE_IDLE,
} APP_DRV_I2C1_STATES;

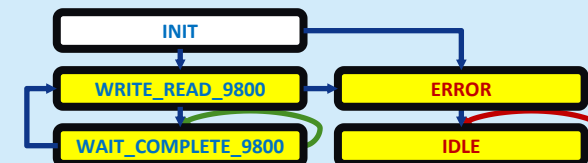
typedef struct
{
    APP_DRV_I2C1_STATES state;
} APP_DRV_I2C1_DATA;

void APP_DRV_I2C1_Initialize ( void );
void APP_DRV_I2C1_Tasks( void );
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



✖ Lab13 Single-Client I2C Digital Temperature Sensor

- Construct new states flow of task in “app_drv_i2c1.c”.

```
/******
```

```
Function:
```

```
void APP_DRV_I2C1_Tasks ( void )
```

```
Remarks:
```

```
See prototype in app_drv_i2c1.h.
```

```
*/
```

```
void APP_DRV_I2C1_Tasks ( void )
```

```
{
```

```
switch ( app_drv_i2c1Data.state )
```

```
{
```

```
case APP_DRV_I2C1_STATE_INIT:
```

```
app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
```

```
break;
```

```
case APP_DRV_I2C1_STATE_WRITE_READ_9800:
```

```
app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800;
```

```
break;
```

```
case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
```

```
app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
```

```
break;
```

```
case APP_DRV_I2C1_STATE_ERROR:
```

```
app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_IDLE;
```

```
break;
```

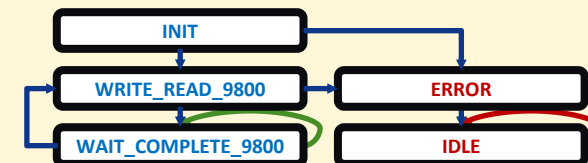
```
case APP_DRV_I2C1_STATE_IDLE:
```

```
break;
```

```
}
```

```
}
```

Overwrite original example code



✖ Lab13 Single-Client I2C Digital Temperature Sensor

- Header include and Global Declarations in “app_drv_i2c1.c”.

Keep this line

```
#include "app_drv_i2c1.h"
#include "app_drv_usart1.h"
#include "driver/i2c/drv_i2c.h"
#include <stdio.h>
#include <string.h>
```

Include app_drv_usart1.h
for debug message output

```
// *****
// *****
// Section: Global Data Definitions
// *****
// *****
#define MCP9800_ADDRESS 0x4D
#define MCP9800_REG_TA 0x0

DRV_HANDLE I2C1Handle_9800;
DRV_I2C_TRANSFER_HANDLE I2C1WriteReadHandle_9800;
volatile int I2C1Complete_9800 = false;
char I2C1WriteBuf_9800[20];
char I2C1ReadBuf_9800[20];

// *****
/* Application Data

Summary:
    Holds application data

Description:
    This structure holds the application's data.

...
*/
APP_DRV_I2C1_DATA app_drv_i2c1Data;
```

⚙️ Lab13 Single-Client I2C Digital Temperature Sensor

- Implement Driver Transfer Handler **Callback Function**.

```
APP_DRV_I2C_DATA app_drv_i2c1Data;

// *****
// *****
// Section: Application Callback Functions
// *****
// *****

void I2C1TransferHandler_9800( DRV_I2C_TRANSFER_EVENT event,
                              DRV_I2C_TRANSFER_HANDLE transferHandle,
                              uintptr_t context )
{
    switch( event )
    {
        case DRV_I2C_TRANSFER_EVENT_PENDING:                break;
        case DRV_I2C_TRANSFER_EVENT_COMPLETE:
            I2C1Complete_9800 = true;
            break;
        case DRV_I2C_TRANSFER_EVENT_HANDLE_EXPIRED:
        case DRV_I2C_TRANSFER_EVENT_ERROR:
        case DRV_I2C_TRANSFER_EVENT_HANDLE_INVALID:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
            break;
    }
}

// *****
// *****
// Section: Application Local Functions
// *****
// *****
```

⚙️ Lab13 Single-Client I2C Digital Temperature Sensor

- Driver Open in state **INIT** and handling error state message. **ERROR**

```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
            I2C1Handle_9800 = DRV_I2C_Open(DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE);
            if(I2C1Handle_9800 == DRV_HANDLE_INVALID)
            {
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
                break;
            }
            DRV_I2C_TransferEventHandlerSet(I2C1Handle_9800, I2C1TransferHandler_9800, 0);
```

Open Driver

Register Callback

Keep this line app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
break;

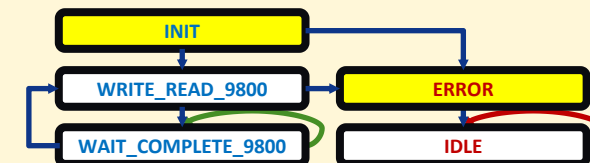
```
        case APP_DRV_I2C1_STATE_WRITE_READ_9800:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800;
            break;
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            break;
```

```
        case APP_DRV_I2C1_STATE_ERROR:
            USART1printf("DRV_I2C1 error!\r\n");
```

Handling error state message

Keep this line app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_IDLE;
break;

```
        case APP_DRV_I2C1_STATE_IDLE:
            break;
    }
}
```



⚙️ Lab13 Single-Client I2C Digital Temperature Sensor

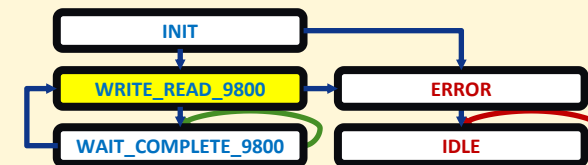
- Driver Write followed by Read in state **WRITE_READ_9800**

```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
            ...
            break;

        case APP_DRV_I2C1_STATE_WRITE_READ_9800:
            I2C1WriteBuf_9800[0] = MCP9800_REG_TA;
            DRV_I2C_WriteReadTransferAdd( I2C1Handle_9800, MCP9800_ADDRESS,
                                           I2C1WriteBuf_9800, 1, I2C1ReadBuf_9800, 2, &I2C1WriteReadHandle_9800 );
            if(I2C1WriteReadHandle_9800 == DRV_I2C_TRANSFER_HANDLE_INVALID)
            {
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
                break;
            }
    }
```

Keep this line app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800;
break;

```
    case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
        app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
        break;
    case APP_DRV_I2C1_STATE_ERROR:
        ...
        break;
    case APP_DRV_I2C1_STATE_IDLE:
        break;
}
```



⚙️ Lab13 Single-Client I2C Digital Temperature Sensor

- Sensor read complete check in state **WAIT_COMPLETE_9800**

```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
            ...
            break;

        case APP_DRV_I2C1_STATE_WRITE_READ_9800:
            ...
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800;
            break;

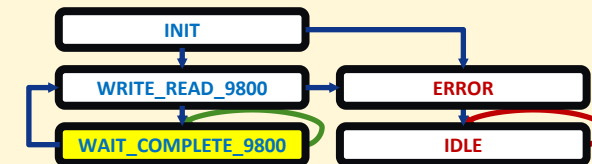
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            if(I2C1Complete_9800 == true)
            {
                I2C1Complete_9800 = false;
                USART1printf("Temp = %2.1f\r",
                    (float)((I2C1ReadBuf_9800[0]<<4)|(I2C1ReadBuf_9800[1]>>4))/16.0f);

                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            }
            break;

        case APP_DRV_I2C1_STATE_ERROR:
            ...
            break;

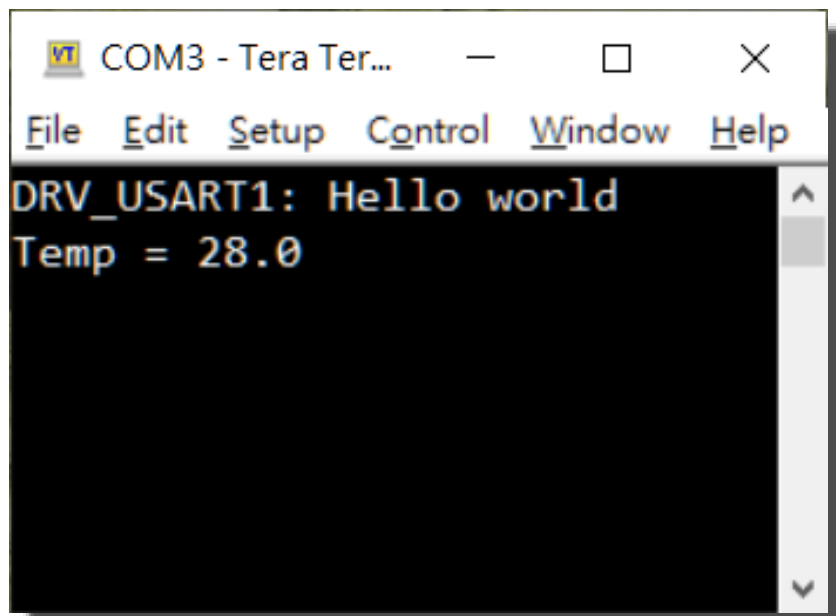
        case APP_DRV_I2C1_STATE_IDLE:
            break;
    }
}
```

Keep this line

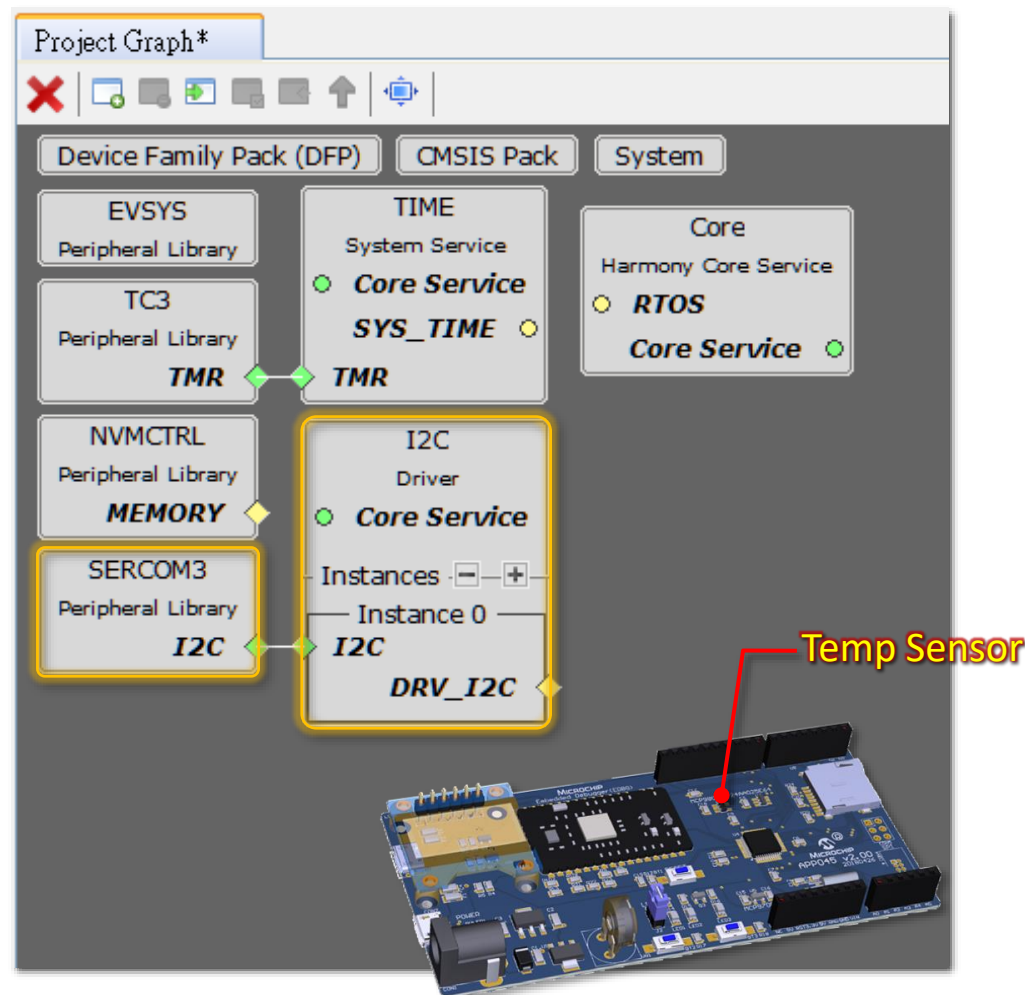
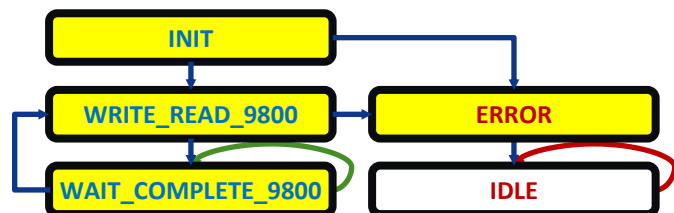


✖ Lab13 Single-Client I2C Digital Temperature Sensor

- Compiler and programming your project to EVB.
- Check the UART output in console. (To do Power Cycle of the EVB might be necessary to reset the I2C device !)



COM3 - Tera Ter...
File Edit Setup Control Window Help
DRV_USART1: Hello world
Temp = 28.0



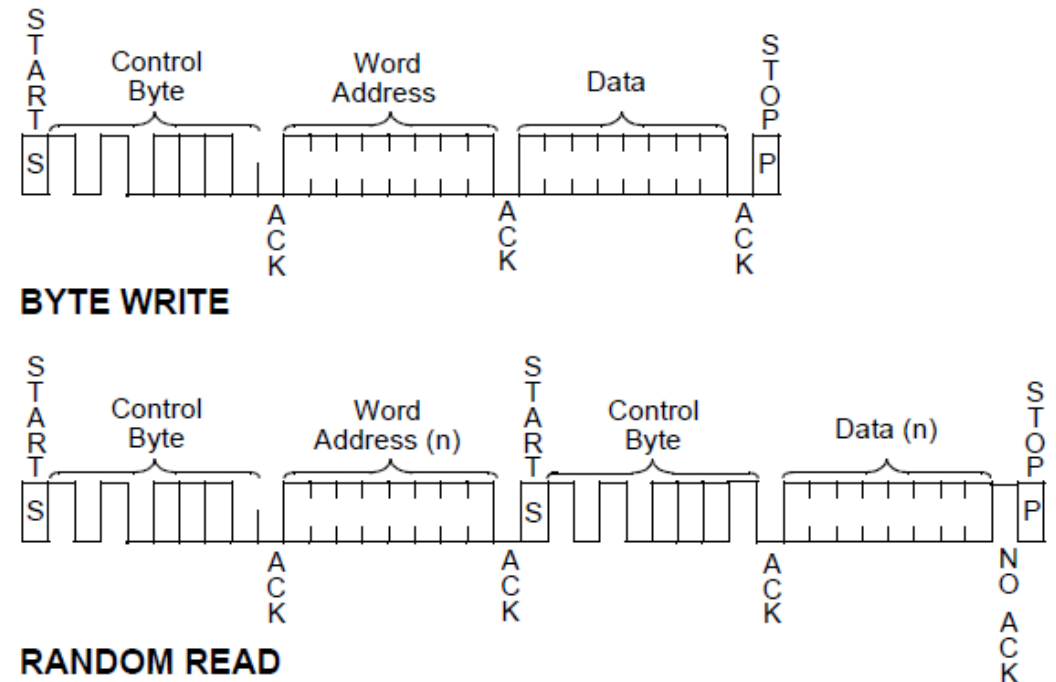
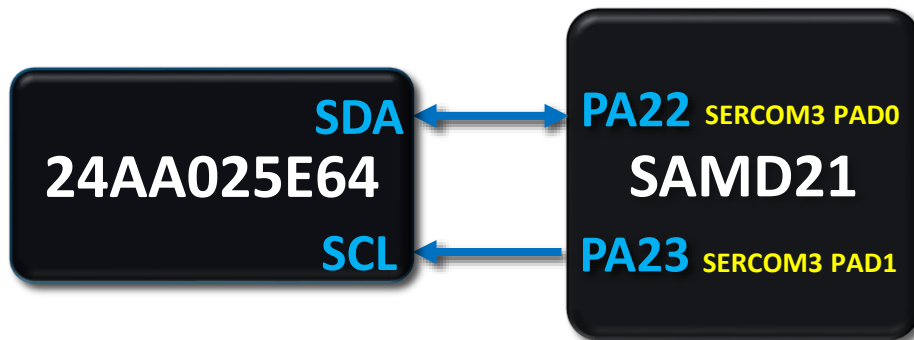
Lab 14

Multi-Client I2C Serial EEPROM

I2C Driver Library

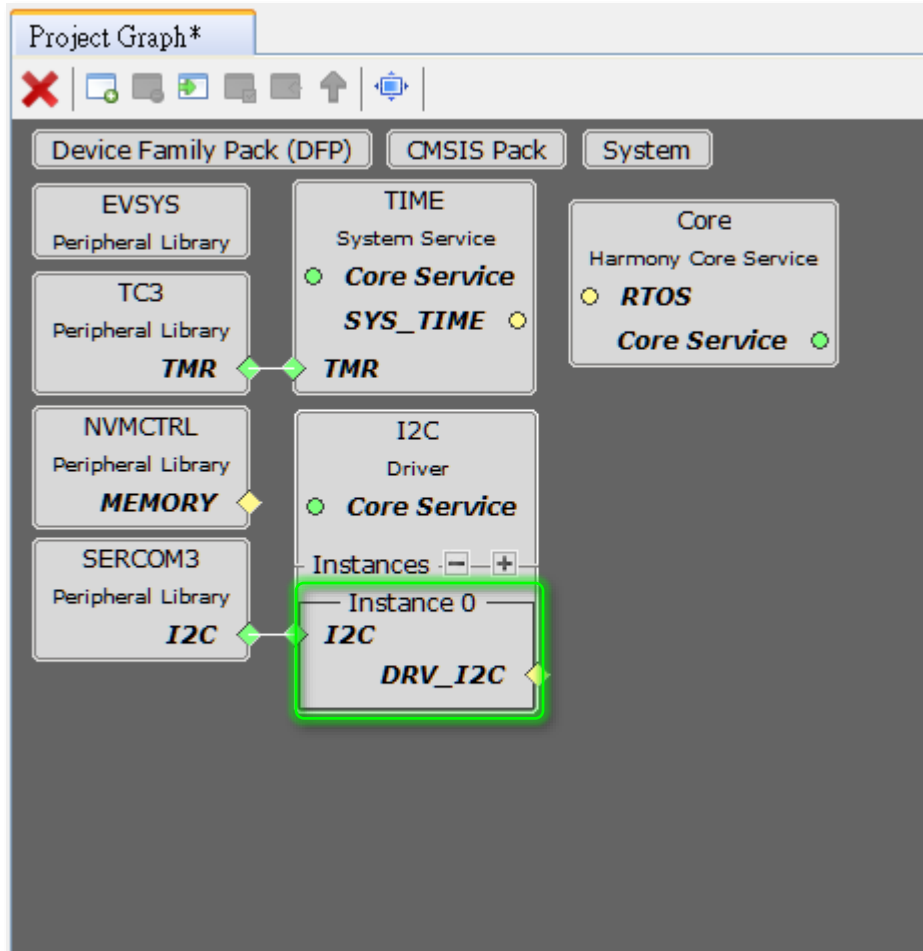
⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Try add another **APP Client** of **I2C Driver** to communication the on-board **Serial EEPROM 24AA025E64**.
- Write one byte "**0x24**" to target word address **0x00** in state **INIT**.
- Random Read it back and output to UART console.
- **24AA025E64** configuration:
 - **SDA** connect to **PA22** (**SERCOM3 PAD0**)
 - **SCL** connect to **PA23** (**SERCOM3 PAD1**)
 - Slave address is **0x50**.

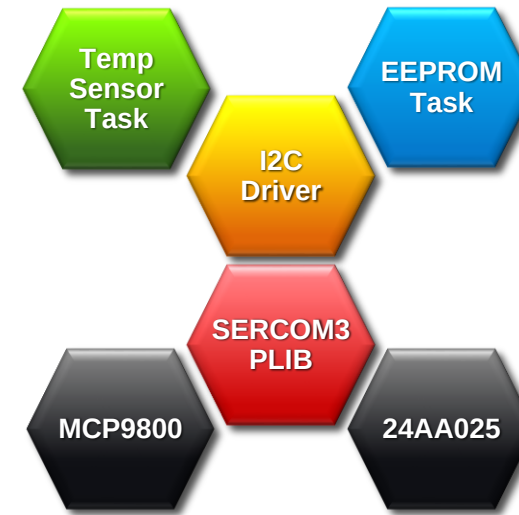
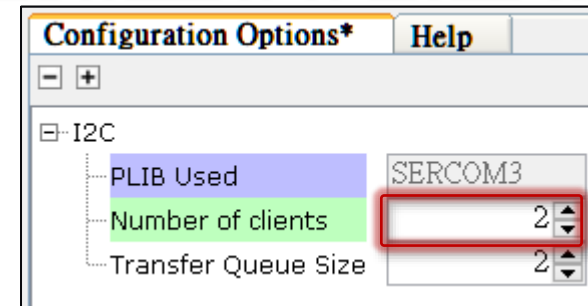


⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Increase “Number of clients” to “2” in I2C Driver Instance 0.

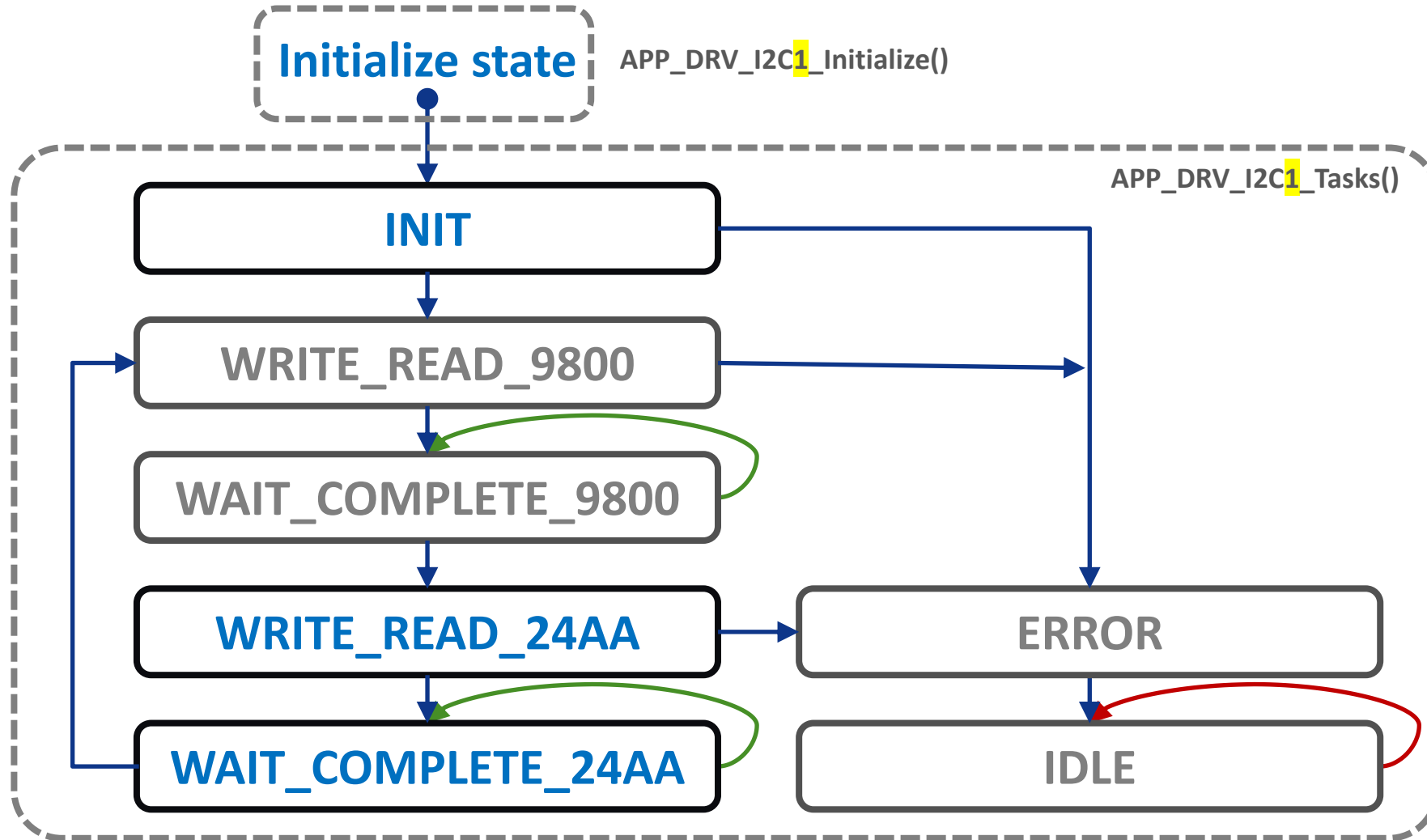


Add Number of clients to 2



⚙️ Lab14 Multi-Client I2C Serial EEPROM

- The state machine we want to modify in “`app_drv_i2c1.c`” as below.



⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Insert two states for EEPROM in “app_drv_i2c1.h”.

```
#ifndef _APP_DRV_I2C1_H
#define _APP_DRV_I2C1_H

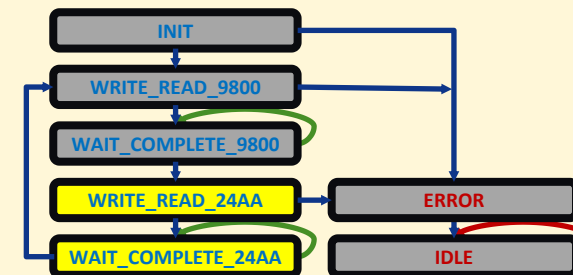
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_I2C1_STATE_INIT=0,
    // APP_DRV_I2C1_STATE_SERVICE_TASKS,
    APP_DRV_I2C1_STATE_WRITE_READ_9800,
    APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800,
    APP_DRV_I2C1_STATE_WRITE_READ_24AA,
    APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA,
    APP_DRV_I2C1_STATE_ERROR,
    APP_DRV_I2C1_STATE_IDLE,
} APP_DRV_I2C1_STATES;

typedef struct
{
    APP_DRV_I2C1_STATES state;
} APP_DRV_I2C1_DATA;

void APP_DRV_I2C1_Initialize ( void );

...
```



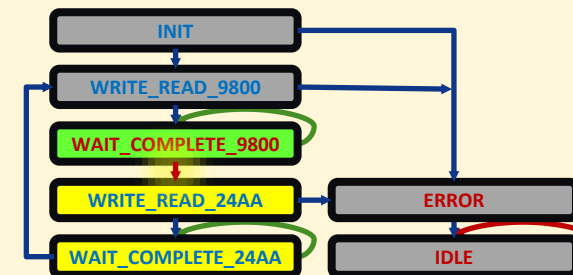
✖ Lab14 Multi-Client I2C Serial EEPROM

- Construct new states flow of task in “app_drv_i2c1.c”.

```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            break;
        case APP_DRV_I2C1_STATE_WRITE_READ_9800:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800;
            break;
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            ... app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_24AA;
            break;
        case APP_DRV_I2C1_STATE_WRITE_READ_24AA:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA;
            break;
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            break;
        case APP_DRV_I2C1_STATE_ERROR:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_IDLE;
            break;
        case APP_DRV_I2C1_STATE_IDLE:
            break;
    }
}
```

Modify this Line

Change next state to WRITE_READ_24AA



⚡ Lab14 Multi-Client I2C Serial EEPROM

- Header include and global declarations in “app_drv_i2c1.c”.

```
#include "app_drv_i2c1.h"
#include "app_drv_usart1.h"
#include "driver/i2c/drv_i2c.h"
#include "system/time/sys_time.h"
#include <stdio.h>
#include <string.h>
```

Include System Time service
for the Delay function

```
// *****
// *****
// Section: Global Data Definitions
// *****
// *****
#define MCP9800_ADDRESS 0x4D
#define MCP9800_REG_TA 0x0
#define MCP24AA_ADDRESS 0x50

DRV_HANDLE I2C1Handle_9800;
DRV_I2C_TRANSFER_HANDLE I2C1WriteReadHandle_9800;
volatile int I2C1Complete_9800 = false;
char I2C1WriteBuf_9800[20];
char I2C1ReadBuf_9800[20];

DRV_HANDLE I2C1Handle_24AA;
DRV_I2C_TRANSFER_HANDLE I2C1WriteReadHandle_24AA;
volatile int I2C1Complete_24AA = false;
char I2C1WriteBuf_24AA[20];
char I2C1ReadBuf_24AA[20];

// *****
/* Application Data
```

⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Implement Driver Transfer Handler **Callback Function**.

```
APP_DRV_I2C1_DATA app_drv_i2c1Data;

// *****
// *****
// Section: Application Callback Functions
// *****
// *****

void I2C1TransferHandler_9800( DRV_I2C_TRANSFER_EVENT event, ...) {...}

void I2C1TransferHandler_24AA( DRV_I2C_TRANSFER_EVENT event,
                              DRV_I2C_TRANSFER_HANDLE transferHandle,
                              uintptr_t context )
{
    switch( event )
    {
        case DRV_I2C_TRANSFER_EVENT_PENDING:                break;
        case DRV_I2C_TRANSFER_EVENT_COMPLETE:
            I2C1Complete_24AA = true;
            break;
        case DRV_I2C_TRANSFER_EVENT_HANDLE_EXPIRED:
        case DRV_I2C_TRANSFER_EVENT_ERROR:
        case DRV_I2C_TRANSFER_EVENT_HANDLE_INVALID:
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
            break;
    }
}

// *****
// *****
// Section: Application Local Functions
// *****
```


⚙️ Lab14 Multi-Client I2C Serial EEPROM

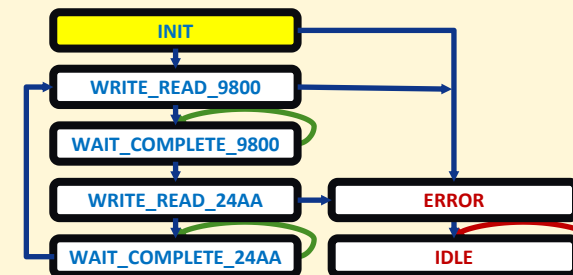
- Add Driver Open for EEPROM client in state **INIT**

```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
            I2C1Handle_9800 = DRV_I2C_Open(DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE);
            if(I2C1Handle_9800 == DRV_HANDLE_INVALID)
            {
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
                break;
            }
            DRV_I2C_TransferEventHandlerSet(I2C1Handle_9800, I2C1TransferHandler_9800, 0);

            I2C1Handle_24AA = DRV_I2C_Open(DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE);
            if(I2C1Handle_24AA == DRV_HANDLE_INVALID)
            {
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
                break;
            }
    }
```

Keep this line app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
break;

```
case APP_DRV_I2C1_STATE_WRITE_READ_9800:
    ...
case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
    ...
case APP_DRV_I2C1_STATE_ERROR:
    ...
case APP_DRV_I2C1_STATE_IDLE:
    ...
```



⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Driver Write byte **0x24** to EEPROM address **0x0** in state **INIT**

```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
            ...
            I2C1Handle_24AA = DRV_I2C_Open(DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE);
            if(I2C1Handle_24AA == DRV_HANDLE_INVALID)
            {
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
                break;
            }

            I2C1WriteBuf_24AA[0] = 0;
            I2C1WriteBuf_24AA[1] = 0x24;
            DRV_I2C_WriteTransferAdd( I2C1Handle_24AA, MCP24AA_ADDRESS,
                                     I2C1WriteBuf_24AA, 2, &I2C1WriteReadHandle_24AA );
            while( DRV_I2C_TransferStatusGet(I2C1WriteReadHandle_24AA) !=
                  DRV_I2C_TRANSFER_EVENT_COMPLETE) {}

            SYS_TIME_HANDLE TimeHandle;
            SYS_TIME_DelayMS(10, &TimeHandle );
            while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

            USART1printf("Wrote 0x24 to EEPROM 24AA\r\n");

            DRV_I2C_TransferEventHandlerSet(I2C1Handle_24AA, I2C1TransferHandler_24AA, 0);

            Keep this line app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            break;

            case APP_DRV_I2C1_STATE_WRITE_READ_9800:
                ...
    }
}
```

Polling method of Write

EEPROM Write delay

Register Transfer Callback

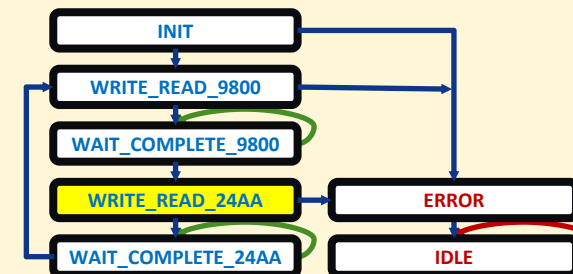
⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Driver Write followed by Read in state **WRITE_READ_24AA**

```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
        ...
        case APP_DRV_I2C1_STATE_WRITE_READ_9800:
        ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
        ...
        app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_24AA;
        break;
        case APP_DRV_I2C1_STATE_WRITE_READ_24AA:
            I2C1WriteBuf_24AA[0] = 0;
            DRV_I2C_WriteReadTransferAdd( I2C1Handle_24AA, MCP24AA_ADDRESS,
                                          I2C1WriteBuf_24AA, 1, I2C1ReadBuf_24AA, 1, &I2C1WriteReadHandle_24AA );
            if(I2C1WriteReadHandle_24AA == DRV_I2C_TRANSFER_HANDLE_INVALID)
            {
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_ERROR;
                break;
            }
    }
```

Keep this line app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA;
break;

```
case APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA:
    ...
case APP_DRV_I2C1_STATE_ERROR:
    ...
case APP_DRV_I2C1_STATE_IDLE:
    ...
}
```



⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Read EEPROM complete check in state **WAIT_COMPLETE_24AA**

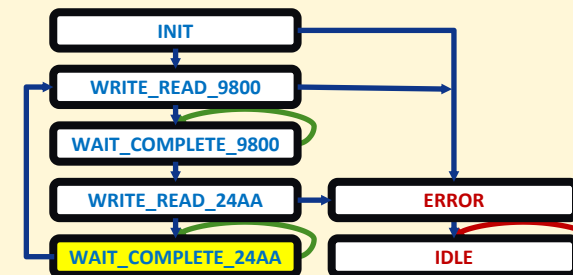
```
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        case APP_DRV_I2C1_STATE_INIT:
            ...
        case APP_DRV_I2C1_STATE_WRITE_READ_9800:
            ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            ...
        case APP_DRV_I2C1_STATE_WRITE_READ_24AA:
            ...
            app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA;
            break;

        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA:
            if(I2C1Complete_24AA == true)
            {
                I2C1Complete_24AA = false;
                USART1printf("\t\tEEPROM = 0x%X\r", I2C1ReadBuf_24AA[0]);

                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            }
            break;

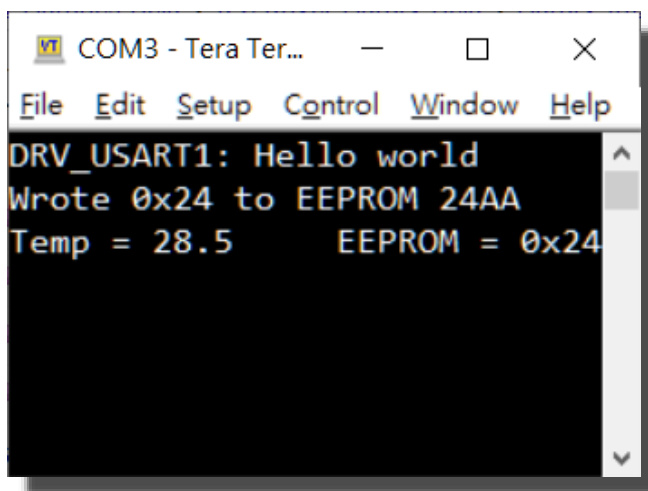
        case APP_DRV_I2C1_STATE_ERROR:
            ...
        case APP_DRV_I2C1_STATE_IDLE:
            ...
    }
}
```

Keep this line

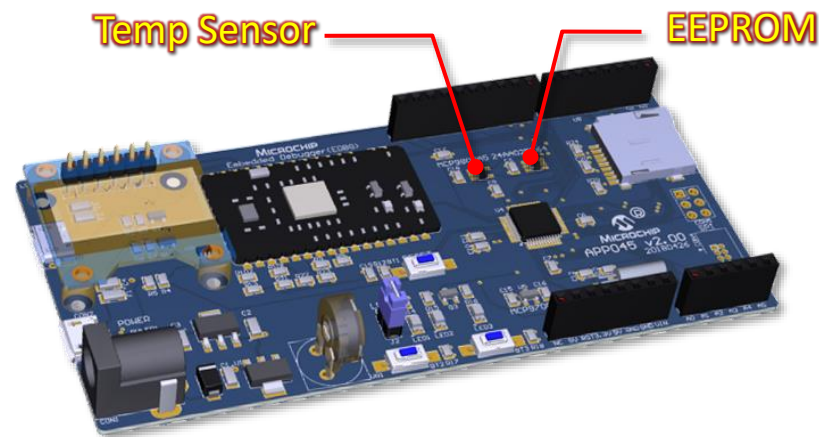
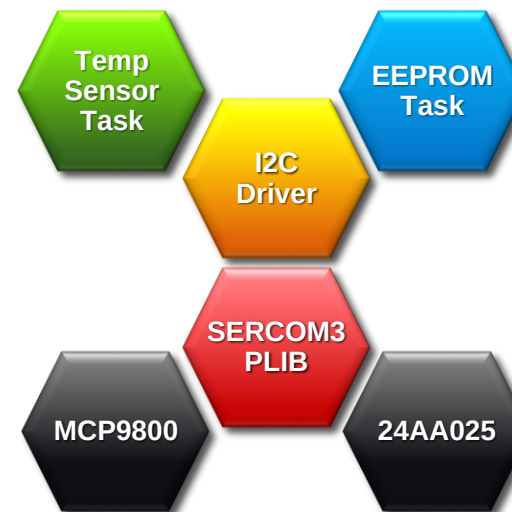
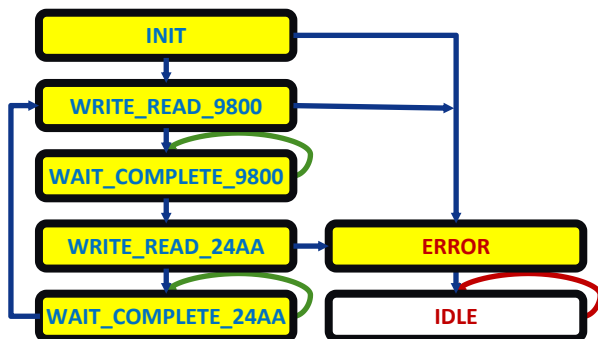


⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Compiler and programming your project to EVB.
- Check the UART output in console. (To do Power Cycle of the EVB might be necessary to reset the I2C device !)



```
COM3 - Tera Ter...  
File Edit Setup Control Window Help  
DRV_USART1: Hello world  
Wrote 0x24 to EEPROM 24AA  
Temp = 28.5 EEPROM = 0x24
```



⚙️ Lab14 Multi-Client I2C Serial EEPROM

- Discuss

- How to implement the Driver to use **one Driver Client**, **one Transfer Callback** and cloud given **Two different I2C speed** properties of two I2C devices **on one PLIB SERCOM** ?

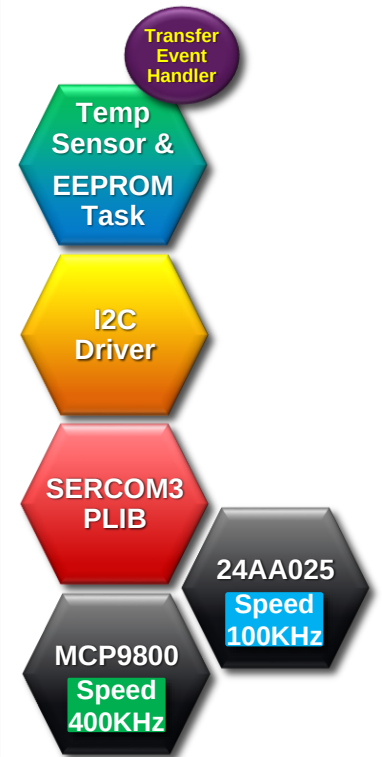
```
DRV_I2C_TRANSFER_SETUP TransferSetup;

void I2C1TransferHandler( DRV_I2C_TRANSFER_EVENT event, DRV_I2C_TRANSFER_HANDLE transferHandle, ... )
{
    switch( event )
    {
        case DRV_I2C_TRANSFER_EVENT_COMPLETE:
            if( transferHandle == I2C1WriteReadHandle_9800 ) I2C1Complete_9800 = true;
            if( transferHandle == I2C1WriteReadHandle_24AA ) I2C1Complete_24AA = true;
            break;
        ...
    }
}

I2C1Handle = DRV_I2C1_Open(DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE);
DRV_I2C_TransferEventHandlerSet(I2C1Handle, I2C1TransferHandler, 0);

TransferSetup.clockSpeed = 400000;
DRV_I2C_TransferSetup(I2C1Handle, &TransferSetup);
DRV_I2C_WriteReadTransferAdd( I2C1Handle, MCP9800_ADDRESS,
                             I2C1WriteBuf_9800, 1, I2C1ReadBuf_9800, 2, &I2C1WriteReadHandle_9800 );

TransferSetup.clockSpeed = 100000;
DRV_I2C_TransferSetup(I2C1Handle, &TransferSetup);
DRV_I2C_WriteReadTransferAdd( I2C1Handle, MCP24AA_ADDRESS,
                             I2C1WriteBuf_24AA, 1, I2C1ReadBuf_24AA, 1, &I2C1WriteReadHandle_24AA );
```



Lab 15

Multi-Instance I2C G-Sensor

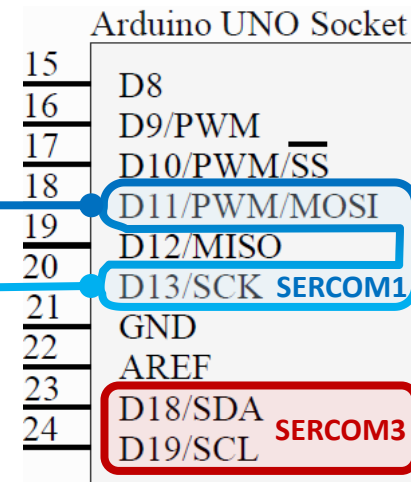
I2C Driver Library

Lab15 Multi-Instance I2C G-Sensor

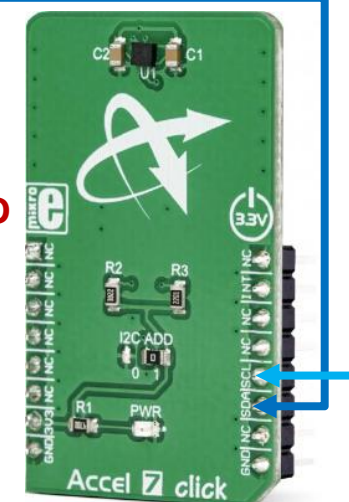
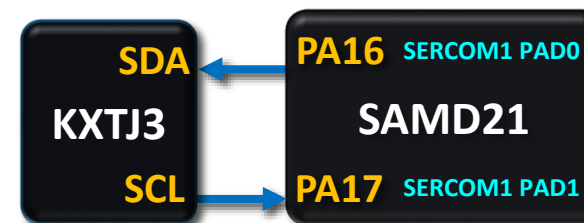
- Add another Driver **Instance 1** of **I2C Driver** and to generate new Application file **app_drv_i2c2** to communication an external **MIKROE Accel 7 CLICK** EVB with **Accelerometer KXTJ3**.
- **KXTJ3 initialize in state INIT:**
 - Write Software Reset "0x80" to Control Register 2 "0x1D" and wait 10ms for stable.
 - Write follow Read the **WHO AM I Register** "0x0F" to check the ID is equal to "0x35" or not.
 - Write Initial Command "0xF0" to Control Register 1 "0x1B" for op-mode and resolution.
- **KXTJ3 Interrupt Source Register read in state WRITE_READ_DRDY_TJ3:**
 - Write follow Read the **Interrupt Source Register** "0x16".
- **KXTJ3 Data Ready Interrupt check in state WAIT_COMPLETE_DRDY_TJ3:**
 - Check "Bit 4 DRDY INT" of **Interrupt Source Register** to make sure data prepared for read.
- **KXTJ3 Axis Data Read in state WRITE_READ_AXIS_TJ3:**
 - Write follow Read the **Axis Output Register** in below sequence to get X, Y, Z data.

XOUL_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

- **KXTJ3 Axis Data Output in state WAIT_COMPLETE_AXIS_TJ3:**
 - Output X, Y, Z Axis data as format "G-Sensor = (+XXXX, +YYYY, +ZZZZ)"
- **KXTJ3 configuration:**
 - **Wire ACCEL 7 CLICK SDA/SCL to Arduino D11(PA16)/D13(PA17)**
 - **SDA** connect to **PA16** (SERCOM1 PAD0)
 - **SCL** connect to **PA17** (SERCOM1 PAD1)
 - Slave address is **0x0E**. (ADDR pin tie to GND)

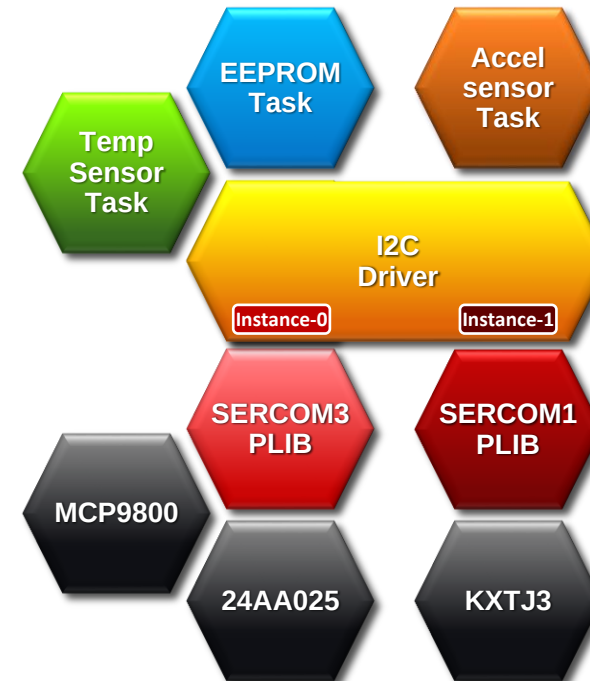
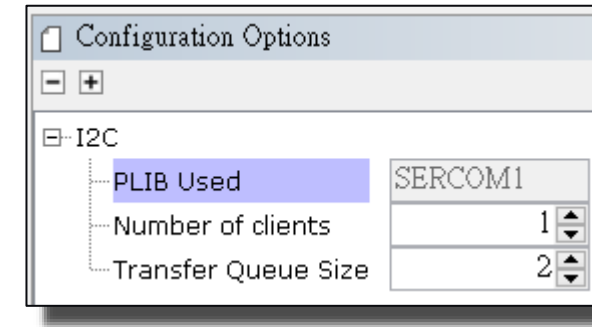
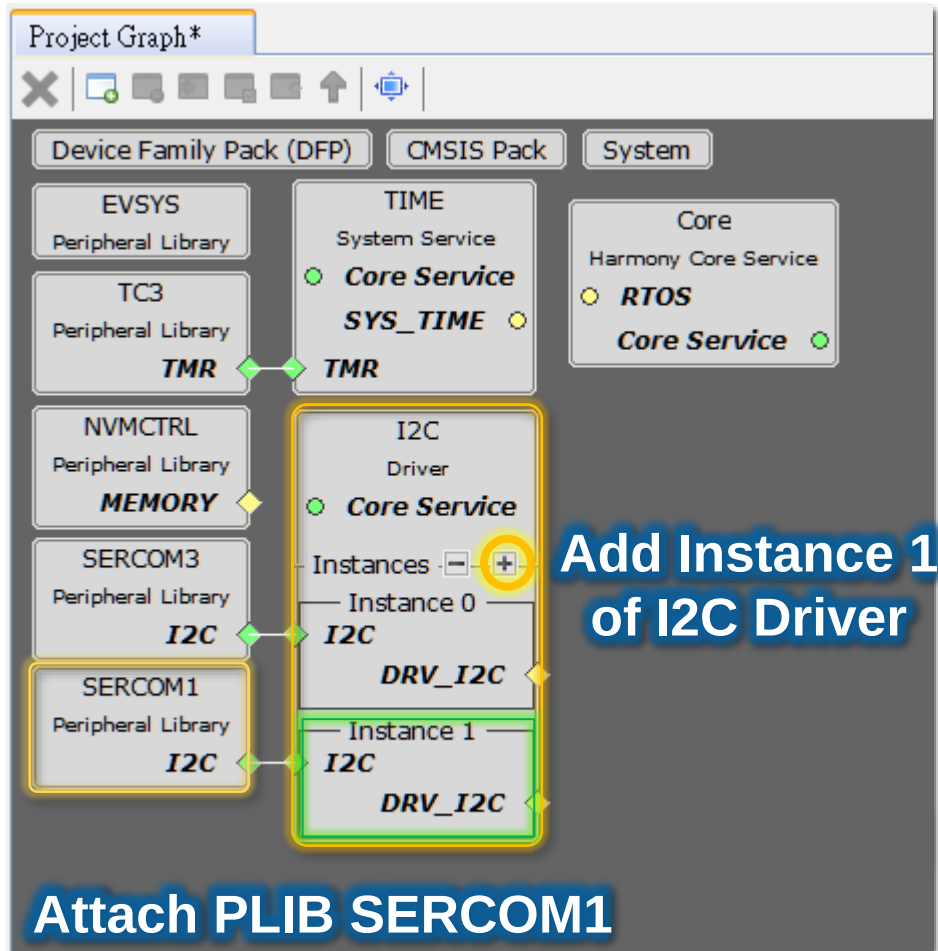


**Remember
Power
3V3 and GND**



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Harmony Configuration (Add **Instance 1** of I2C driver)



✖ Lab15 Multi-Instance I2C G-Sensor

- Enable **PLIB SERCOM1** pads of I2C Driver **Instance 1**.

Project Graph* Pin Diagram * Pin Table * Pin Settings *									
Order:	Ports ▾	Table View	☒ Easy View						
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
25	PA16		SERCOM1_PAD0 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾
26	PA17		SERCOM1_PAD1 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾

Configure PA16 as SERCOM1_PAD0
Configure PA17 as SERCOM1_PAD1

Project Graph* Pin Diagram * Pin Table * Pin Settings *

Package: TQFP48

		1	2	3	4	9	10	11	12	13	14	15	16	21	22	23	24	25	26	27	28	29	30	31	32	33	34							
																	A																	
Module	Function	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25							
SERCOM1	SERCOM1_PAD0																																	
	SERCOM1_PAD1																																	
	SERCOM1_PAD2																																	
	SERCOM1_PAD3																																	

✂ Lab15 Multi-Instance I2C G-Sensor

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**4**” and given the Application Name to “**app_drv_i2c2**”.

The image shows two windows from the Harmony IDE. The left window, titled "Project Graph*", displays a block diagram of the system components. It includes sections for "Device Family Pack (DFP)", "CMSIS Pack", and "System". Under "System", the "Core" block is highlighted with a green box and contains "Harmony Core Service", "RTOS", and "Core Service". The "I2C Driver" block is also highlighted with a green box and shows two instances, "Instance 0" and "Instance 1", both labeled "DRV_I2C". The right window, titled "Configuration Options*", shows the "Core" configuration. A red banner at the top reads "Increase Number of Application". Under "Application Configuration", the "Number of Applications" is set to 4. Below this, "Application 2 Configuration" and "Application 3 Configuration" are shown. "Application 2 Configuration" has "Application Name" set to "app_drv_i2c1". "Application 3 Configuration" has "Application Name" set to "app_drv_i2c2". A blue banner at the bottom of the left window reads "Rename file to 'app_drv_i2c2'".

Project Graph*

Device Family Pack (DFP) CMSIS Pack System

EVSYS Peripheral Library

TC3 Peripheral Library

NVMCTRL Peripheral Library

SERCOM3 Peripheral Library

SERCOM1 Peripheral Library

TIME System Service

Core Service

SYS_TIME

TMR

I2C Driver

Core Service

Instances - +

Instance 0

Instance 1

DRV_I2C

DRV_I2C

Core

Harmony Core Service

RTOS

Core Service

Rename file to “app_drv_i2c2”

Configuration Options* Help

Increase Number of Application

Core

Generate Harmony Application Files ☒

Application Configuration

Number of Applications 4

Application 0 Configuration

Application 1 Configuration

Application 2 Configuration

Enable Application 2 Configuration ☒

Application Name app_drv_i2c1

**** Application name must be valid C-Language id

Application 3 Configuration

Enable Application 3 Configuration ☒

Application Name app_drv_i2c2

**** Application name must be valid C-Language id

Generate Harmony Driver Common Files ☒

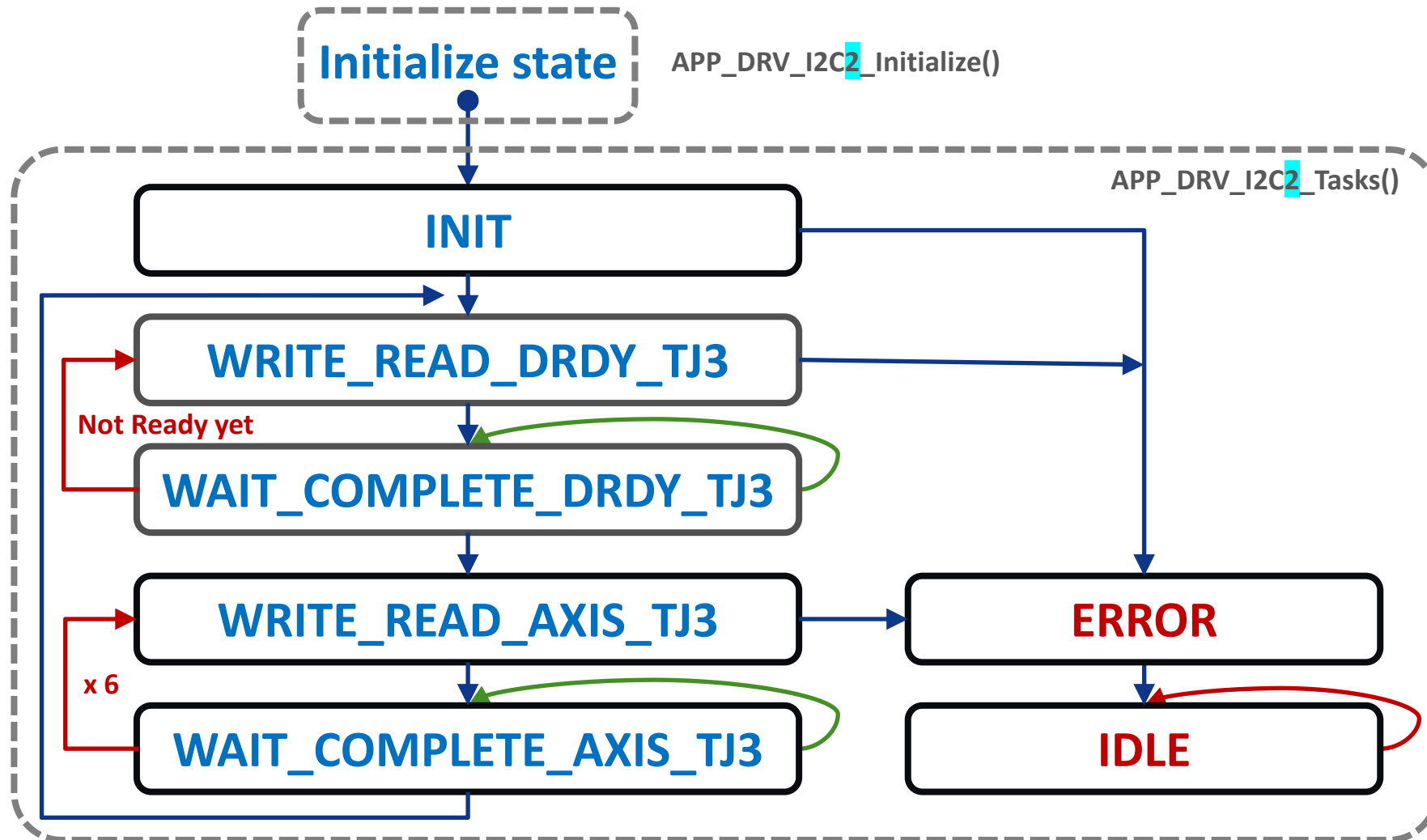
Generate Harmony System Service Common Files ☒

Generate Harmony System Media Files ☐

Enable System Interrupt ☒

⚙️ Lab15 Multi-Instance I2C G-Sensor

- The state machine we want to implement in “`app_drv_i2c2.c`” as below.



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Add more states in “app_drv_i2c2.h”.

```
#ifndef _APP_DRV_I2C2_H
#define _APP_DRV_I2C2_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_I2C2_STATE_INIT=0,
    // APP_DRV_I2C2_STATE_SERVICE_TASKS,
    APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3,
    APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3,
    APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3,
    APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3,
    APP_DRV_I2C2_STATE_ERROR,
    APP_DRV_I2C2_STATE_IDLE,
} APP_DRV_I2C2_STATES;

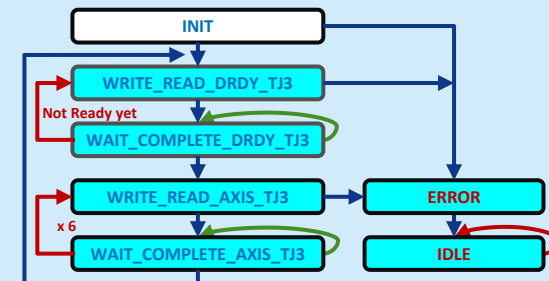
typedef struct
{
    APP_DRV_I2C2_STATES state;
} APP_DRV_I2C2_DATA;

void APP_DRV_I2C2_Initialize ( void );
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Construct new states flow of task in “app_drv_i2c2.c”.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            break;

        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3;
            break;

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
            break;

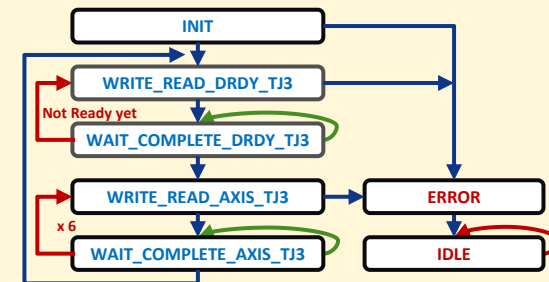
        case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;
            break;

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            break;

        case APP_DRV_I2C2_STATE_ERROR:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;
            break;

        case APP_DRV_I2C2_STATE_IDLE:
            break;
    }
}
```

Overwrite original example code



✂ Lab15 Multi-Instance I2C G-Sensor

- Header include and Global Declarations in “app_drv_i2c2.c”.

Keep this line

```
#include "app_drv_i2c2.h"
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "driver/i2c/drv_i2c.h"
#include "system/time/sys_time.h"
#include <stdio.h>
#include <string.h>
```

app_drv_usart 1 and 2 .h
for debug output
sys_time.h for delay function

```
// *****
// *****
// Section: Global Data Definitions
// *****
// *****
```

```
#define KXTJ3_ADDRESS 0x0E
```

KXTJ3 ADDR pin tie to GND

```
DRV_HANDLE I2C2Handle_TJ3;
DRV_I2C_TRANSFER_HANDLE I2C2WriteReadHandle_TJ3;
volatile int I2C2Complete_TJ3 = false;
char I2C2WriteBuf_TJ3[20];
char I2C2ReadBuf_TJ3[20];
int AxisOutRegIdx = 0;
char AxisOutReg[6] = { 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B };
char AxisOutByte[6];
```

```
// *****
/* Application Data
```

Summary:

```
*/ ...
```

```
APP_DRV_I2C2_DATA app_drv_i2c2Data;
```

XOUL_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

Axis registers sequenced
in AxisOutReg[6]

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Implement Driver Transfer Handler **Callback Function**.

```
APP_DRV_I2C_DATA app_drv_i2c2Data;

// *****
// *****
// Section: Application Callback Functions
// *****
// *****

void I2C2TransferHandler_TJ3( DRV_I2C_TRANSFER_EVENT event,
                             DRV_I2C_TRANSFER_HANDLE transferHandle,
                             uintptr_t context )
{
    switch( event )
    {
        case DRV_I2C_TRANSFER_EVENT_PENDING:                break;
        case DRV_I2C_TRANSFER_EVENT_COMPLETE:
            I2C2Complete_TJ3 = true;
            break;
        case DRV_I2C_TRANSFER_EVENT_HANDLE_EXPIRED:
        case DRV_I2C_TRANSFER_EVENT_ERROR:
        case DRV_I2C_TRANSFER_EVENT_HANDLE_INVALID:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
            break;
    }
}

// *****
// *****
// Section: Application Local Functions
// *****
// *****
```


⚙️ Lab15 Multi-Instance I2C G-Sensor

- Driver open and send **Software Reset** command in state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
    case APP_DRV_I2C2_STATE_INIT:
        I2C2Handle_TJ3 = DRV_I2C_Open(DRV_I2C_INDEX_1, DRV_IO_INTENT_READWRITE);
        if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)
        {
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
            break;
        }

        I2C2WriteBuf_TJ3[0] = 0x1D;
        I2C2WriteBuf_TJ3[1] = 0x80;
        DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );
        while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
                DRV_I2C_TRANSFER_EVENT_COMPLETE) {}

        SYS_TIME_HANDLE TimeHandle;
        SYS_TIME_DelayMS(10, &TimeHandle );
        while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

        USART1printf("G-Sensor Software Reset Completed\r\n");
    }
```

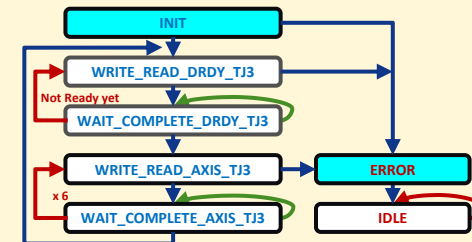
Software Reset

Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
break;

Handling error state message

```
...
case APP_DRV_I2C2_STATE_ERROR:
    USART1printf("DRV_I2C2 error!\r\n");
```

Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;
break;



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Read **WHO AM I** register to check the ID is equal to “**0x35**” or not in state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT:
            SYS_TIME_DelayMS(10, &TimeHandle );
            while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

            USART1printf("G-Sensor Software Reset Completed\r\n");

            I2C2WriteBuf_TJ3[0] = 0x0F;
            DRV_I2C_WriteReadTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                         I2C2WriteBuf_TJ3, 1, I2C2ReadBuf_TJ3, 1, &I2C2WriteReadHandle_TJ3 );
            while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
                  DRV_I2C_TRANSFER_EVENT_COMPLETE) {}

            if( I2C2ReadBuf_TJ3[0] != 0x35 )
            {
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
                break;
            }

            USART1printf("G-Sensor WHO AM I : 0x%X\r\n", I2C2ReadBuf_TJ3[0]);

            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            break;

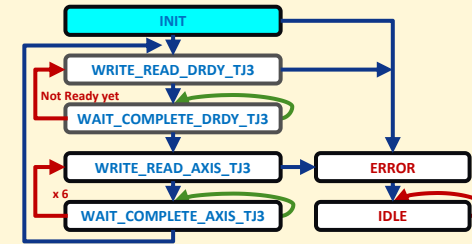
        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3;
            break;

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
            ...
    }
}
```

Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3; break;

case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:

...



Read WHO AM I

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Write **Initial** Command for operation mode and resolution setup in state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT:
            ...
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
            break;
    }

    USART1printf("G-Sensor WHO AM I : 0x%X\r\n", I2C2ReadBuf_TJ3[0]);

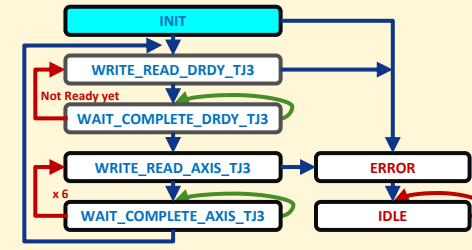
    I2C2WriteBuf_TJ3[0] = 0x1B;
    I2C2WriteBuf_TJ3[1] = 0xF0;
    DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                             I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );
    while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
           DRV_I2C_TRANSFER_EVENT_COMPLETE) {}

    USART1printf("G-Sensor Initial Completed\r\n");
    DRV_I2C_TransferEventHandlerSet(I2C2Handle_TJ3, I2C2TransferHandler_TJ3, 0);

    Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
    break;

    case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3:
        app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3;
        break;

    case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
        ...
    }
}
```



Write Initial

Register Transfer Callback

Lab15 Multi-Instance I2C G-Sensor

- Read **Interrupt Source** register to check **Data Ready** interrupt flag.

[illegible]

```
Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3;  
break;
```

```

case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
    break;

case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;
    break;

case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
    break;

```

• • •

Lab15 Multi-Instance I2C G-Sensor

- Check **Bit 4** of **Interrupt Source** register, if **Data Ready** for read.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT: ...

        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3: ...

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                if(I2C2ReadBuf_TJ3[0] & (1<<4))
                {
                    AxisOutRegIdx = 0;
                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
                    break;
                }
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            }
            break;

        case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;
            break;

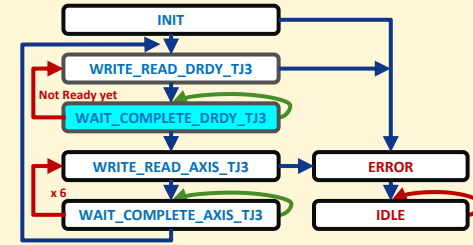
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            break;

        case APP_DRV_I2C2_STATE_ERROR:
            ...
    }
}
```

```
graph TD
    INIT --> WRD_TJ3[WRITE_READ_DRDY_TJ3]
    WRD_TJ3 --> WCDR_TJ3[WAIT_COMPLETE_DRDY_TJ3]
    WCDR_TJ3 -- "Not Ready yet" --> WRD_TJ3
    WCDR_TJ3 --> WRD_AXIS_TJ3[WRITE_READ_AXIS_TJ3]
    WRD_AXIS_TJ3 --> WCAXIS_TJ3[WAIT_COMPLETE_AXIS_TJ3]
    WCAXIS_TJ3 -- "x6" --> WRD_AXIS_TJ3
```

Reset Axis Out Register while Data Read

Keep this line



Reset Axis Out Register Index while Data Ready

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Read **Axis Output** register in order of **AxisOutReg[]** one by one.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT: ...

        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3: ...

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3: ...

        case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:
            I2C2WriteBuf_TJ3[0] = AxisOutReg[AxisOutRegIdx];
            DRV_I2C_WriteReadTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                          I2C2WriteBuf_TJ3, 1, I2C2ReadBuf_TJ3, 1, &I2C2WriteReadHandle_TJ3 );

            if(I2C2WriteReadHandle_TJ3 == DRV_I2C_TRANSFER_HANDLE_INVALID)
            {
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
                break;
            }

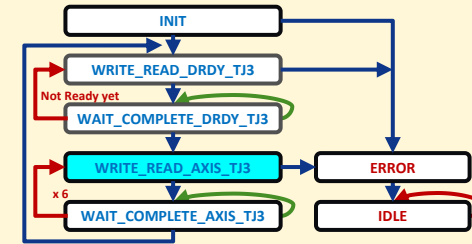
            x 6 {
                char AxisOutReg[6] = { 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B };
            }

            Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;
            break;

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            break;

        case APP_DRV_I2C2_STATE_ERROR:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;
            break;

        ...
    }
}
```



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Output gravity of X, Y, Z after to collected **6 bytes** of Axis Data.

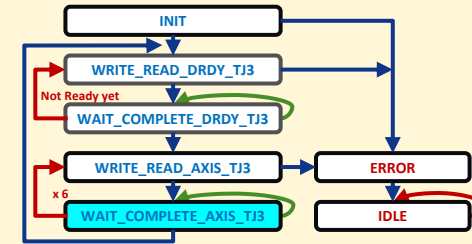
```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT: ...
        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3: ...
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3: ...
        case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3: ...

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];
                if(++ AxisOutRegIdx >= 6)
                {
                    USART1printf("\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",
                        (short)(AxisOutByte[1]<<8|AxisOutByte[0]),
                        (short)(AxisOutByte[3]<<8|AxisOutByte[2]),
                        (short)(AxisOutByte[5]<<8|AxisOutByte[4]));

                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
                    break;
                }
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
            }
            break;

        case APP_DRV_I2C2_STATE_ERROR:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;
            break;
        ...
    }
}
```

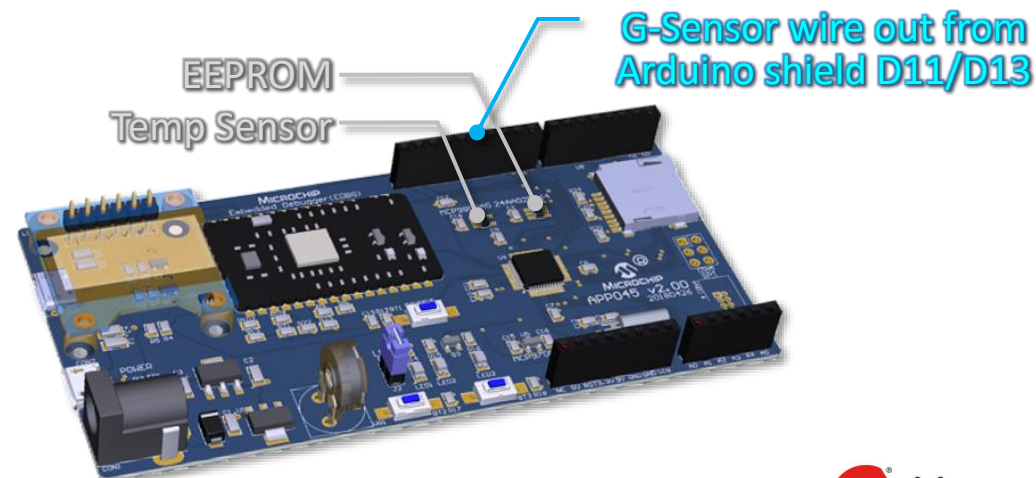
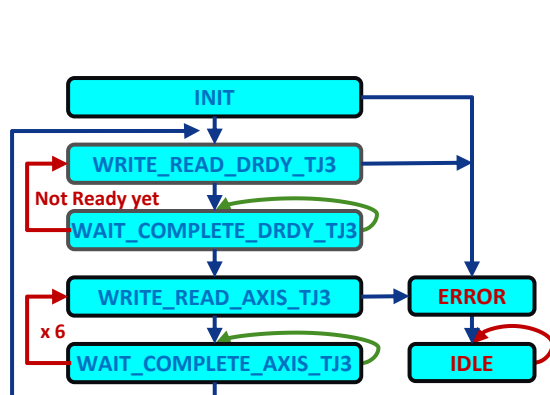


XOUL_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

Keep this line

- **Compiler and programming your project to EVB.**
- **Check the UART output in console.**

MIKORE ACCEL 7 CLICK KXTJ3 G-Sensor



Advanced Exercise

MPLAB Data Visualizer

I2C Driver Library

⚙️ Lab15 Multi-Instance I2C G-Sensor

- **Advanced Exercise**

- Use **MPLAB Data Visualizer** to draw X, Y, Z gravity from **USART2** console.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        ...
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];
                if(++ AxisOutRegIdx >= 6)
                {
                    USART1printf("\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",
                        (short)(AxisOutByte[1]<<8|AxisOutByte[0]),
                        (short)(AxisOutByte[3]<<8|AxisOutByte[2]),
                        (short)(AxisOutByte[5]<<8|AxisOutByte[4]));

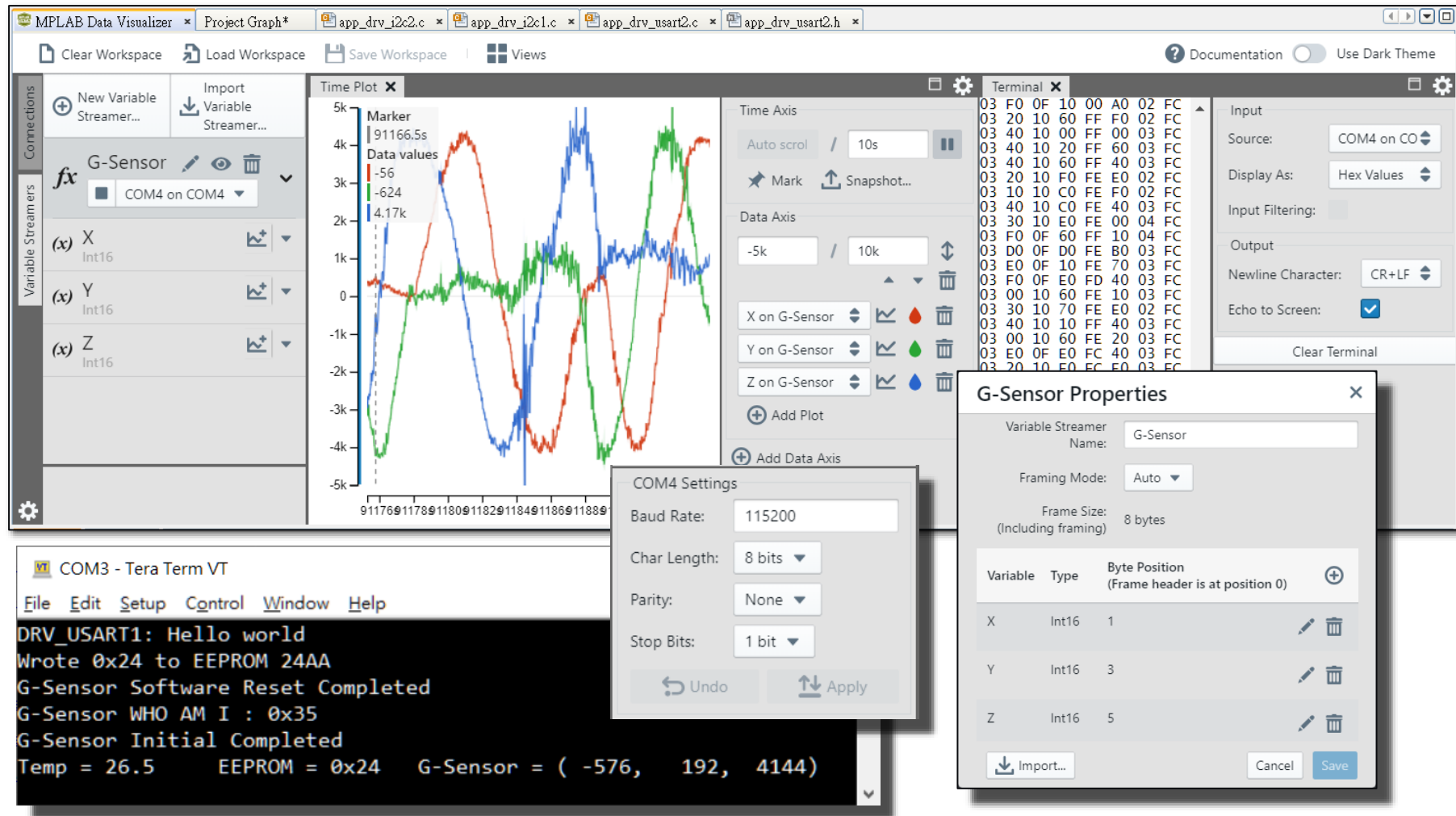
                    unsigned char DVBuf[8] = { 0x03,
                                                AxisOutByte[0], AxisOutByte[1],
                                                AxisOutByte[2], AxisOutByte[3],
                                                AxisOutByte[4], AxisOutByte[5],
                                                0xFC };

                    USART2write( DVBuf, 8 );

                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
                    break;
                }
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
            }
            break;
        ...
    }
```

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Result on **MPLAB Data Visualizer**, Create a streamer **G-Sensor** to have three **int16** variable X, Y, Z gravity parser from **USART2 115200-8-n-1**



Advanced Exercise

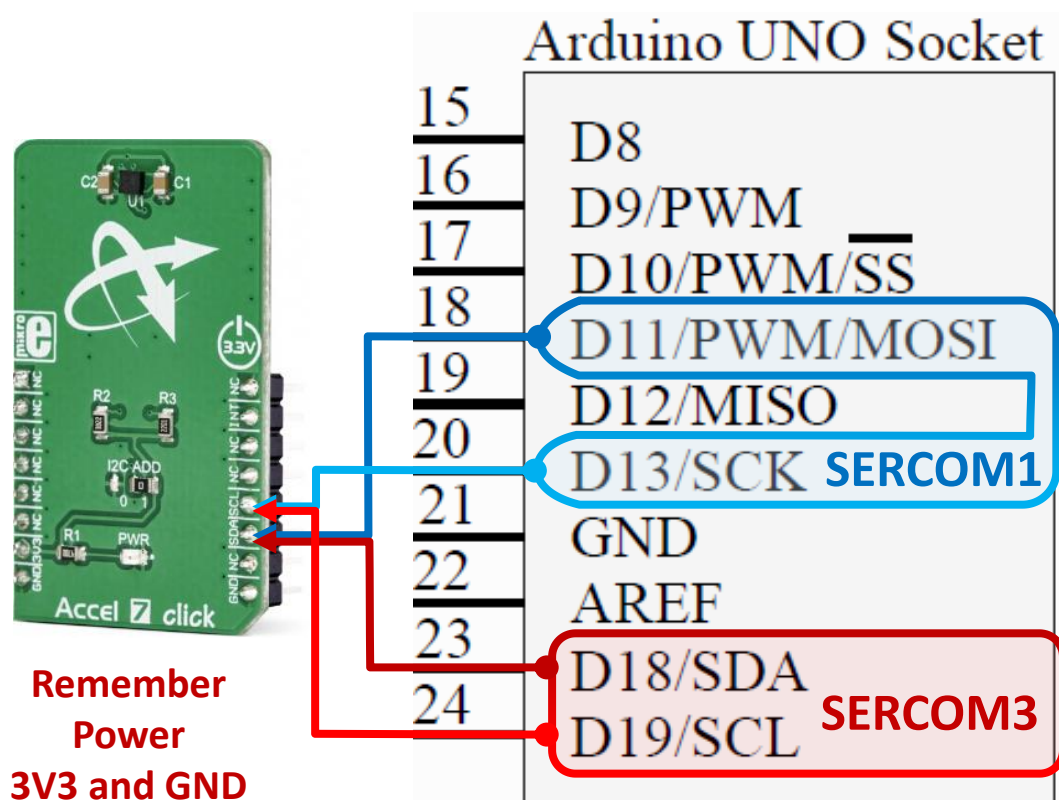
I2C Device Auto Probe

I2C Driver Library

⚙️ Lab15 Multi-Instance I2C G-Sensor

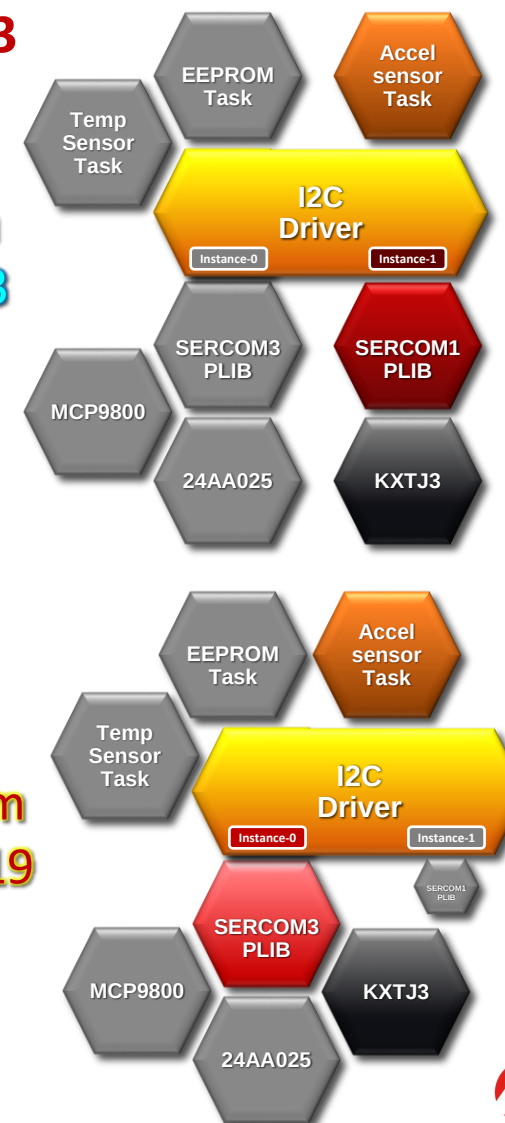
• Advanced Exercise

- Auto probe the G-Sensor connection on **PLIB SERCOM1** or **SERCOM3**
- Please wire **ACCEL 7 CLICK** to different pin out of Arduino Shield.



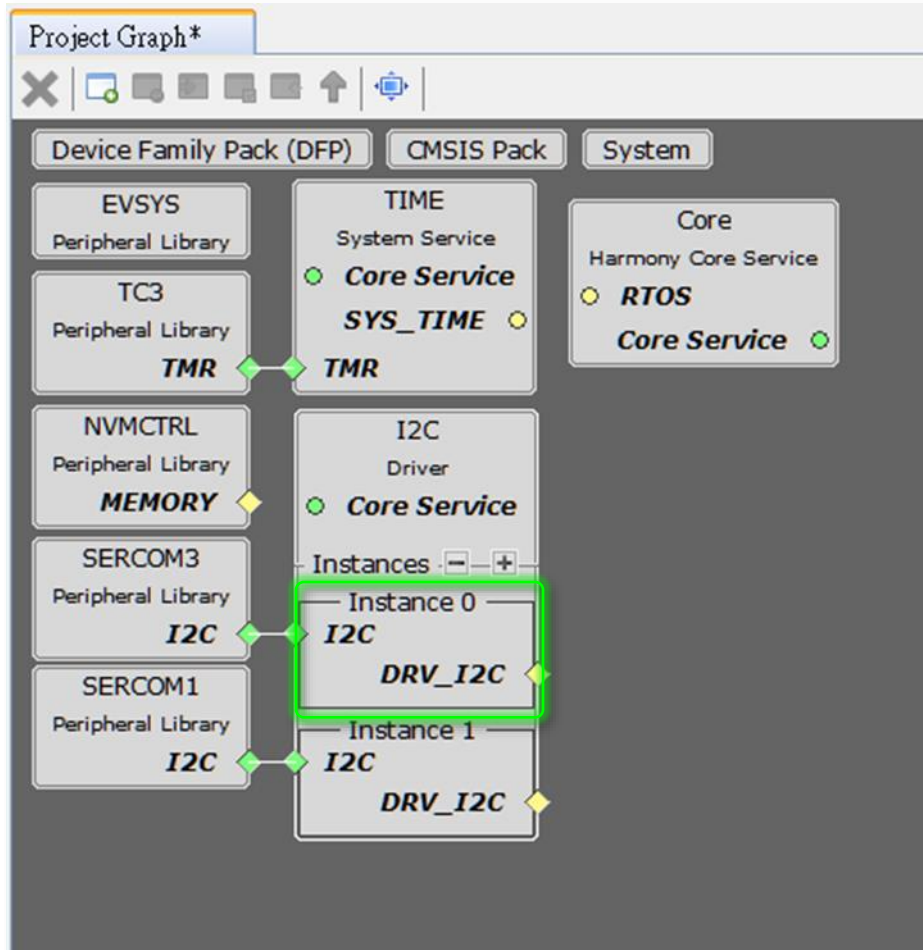
G-Sensor wire out from
Arduino Shield D11/D13
will use **SERCOM1**

G-Sensor wire out from
Arduino Shield D18/D19
will use **SERCOM3**

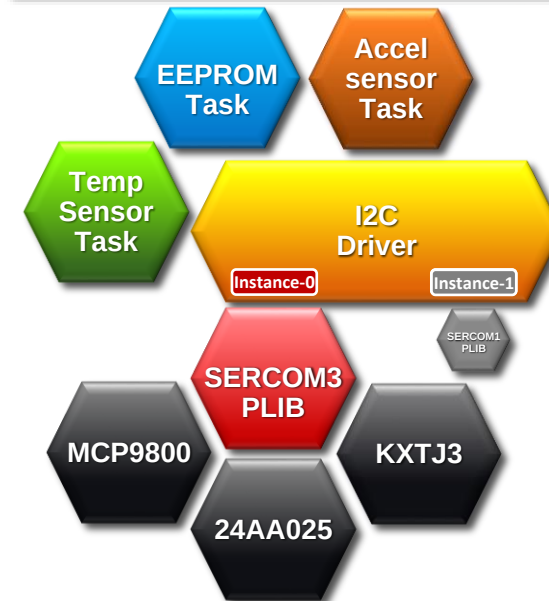
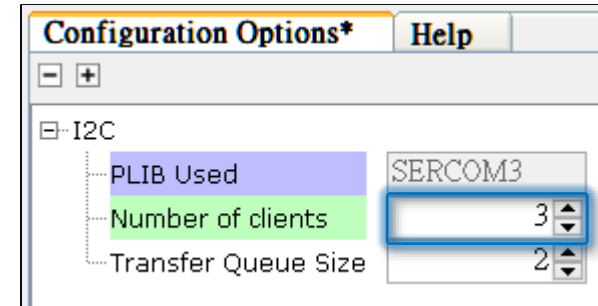


⚙️ Lab15 Multi-Instance I2C G-Sensor

- Increase “**Number of clients**” to **3** in I2C Driver **Instance 0**.



Add Number of clients to 3



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Comment out **Software Reset** in APP_DRV_I2C2_Tasks state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
    case APP_DRV_I2C2_STATE_INIT:
        // I2C2Handle_TJ3 = DRV_I2C_Open(DRV_I2C_INDEX_1, DRV_IO_INTENT_READWRITE);
        // if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)
        // {
        //     app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
        //     break;
        // }

        // I2C2WriteBuf_TJ3[0] = 0x1D;
        // I2C2WriteBuf_TJ3[1] = 0x80;
        // DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
        //                           I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );
        // while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
        //                           DRV_I2C_TRANSFER_EVENT_COMPLETE) {}
        // SYS_TIME_HANDLE TimeHandle;
        // SYS_TIME_DelayMS(10, &TimeHandle );
        // while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

        USART1printf("G-Sensor Software Reset Completed\r\n");

        I2C2WriteBuf_TJ3[0] = 0x0F;
        DRV_I2C_WriteReadTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
        I2C2WriteBuf_TJ3, 1, I2C2ReadBuf_TJ3, 1, &I2C2WriteReadHandle_TJ3 );
        while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
        DRV_I2C_TRANSFER_EVENT_COMPLETE) {}

        if( I2C2ReadBuf_TJ3[0] != 0x35 )
        {
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
            ...
        }
    }
}
```

Comment out

Comment out original implement
before Software Reset Message

Comment out before this line

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Auto Probe loop to find G-Sensor on which **Instance** of **DRV_I2C**.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT:
            ...
            // I2C2WriteBuf_TJ3[0] = 0x1D;
            // I2C2WriteBuf_TJ3[1] = 0x80;
            // DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
            //                               I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );
            // while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
            //                               DRV_I2C_TRANSFER_EVENT_COMPLETE) {}
            // SYS_TIME_HANDLE TimeHandle;
            // SYS_TIME_DelayMS(10, &TimeHandle );
            // while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

            for( int Inst = DRV_I2C_INDEX_0 ; Inst <= DRV_I2C_INDEX_1 ; Inst++ )
            {
                I2C2Handle_TJ3 = DRV_I2C_Open(Inst, DRV_IO_INTENT_READWRITE);
                if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)
                {
                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
                    return;
                }
                I2C2WriteBuf_TJ3[0] = 0x1D;
                I2C2WriteBuf_TJ3[1] = 0x80;
                DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                          I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );
            }

            USART1printf("G-Sensor Software Reset Completed\r\n");
            ...
    }
}
```

Implement a loop for Auto Probe all Instance of DRV_I2C

Exit INIT state if open fail

Insert between this

Insert between this

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Check transfer status to decide G-Sensor exist or not on the bus.

```
case APP_DRV_I2C2_STATE_INIT:
    for( int Inst = DRV_I2C_INDEX_0 ; Inst <= DRV_I2C_INDEX_1 ; Inst++ )
    {
        I2C2Handle_TJ3 = DRV_I2C_Open(Inst, DRV_IO_INTENT_READWRITE);
        if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)
        {
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
            return;
        }
        I2C2WriteBuf_TJ3[0] = 0x1D;
        I2C2WriteBuf_TJ3[1] = 0x80;
        DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                I2C2WriteBuf_TJ3, 2, & I2C2WriteReadHandle_TJ3 );

        DRV_I2C_TRANSFER_EVENT TransferStatus;
        do
        {
            TransferStatus = DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3);
            if( TransferStatus == DRV_I2C_TRANSFER_EVENT_ERROR )
            {
                DRV_I2C_Close(I2C2Handle_TJ3);
                break;
            }
        } while( TransferStatus != DRV_I2C_TRANSFER_EVENT_COMPLETE );

        SYS_TIME_HANDLE TimeHandle;
        SYS_TIME_DelayMS(10, &TimeHandle );
        while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

        if( TransferStatus == DRV_I2C_TRANSFER_EVENT_COMPLETE )
            break;
    }
}
```

Start of For loop

Insert between this

Probe failure try next Instance

Probe success exit probe loop

End of For loop

Insert between this

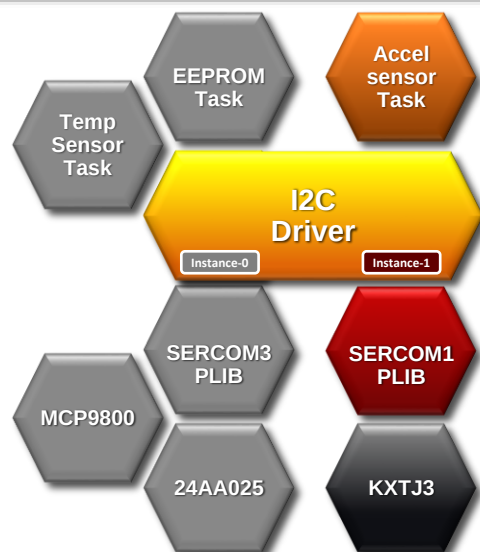
⚙️ Lab15 Multi-Instance I2C G-Sensor

- Try wire G-Sensor to different pin out of Arduino socket and check if the G-Sensor could be **Auto Probed** by our implementation.

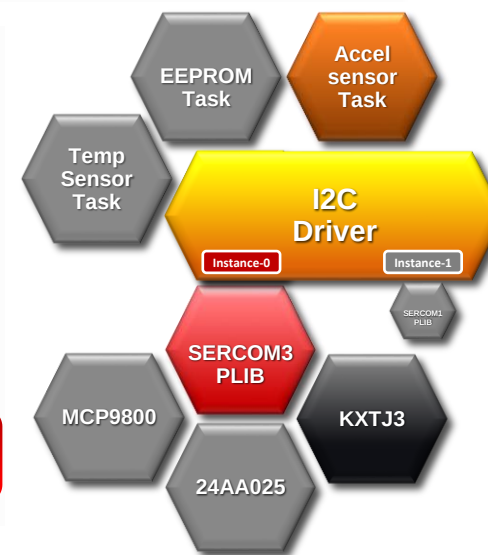
```

COM3 - Tera Term VT
File Edit Setup Control Window Help
DRV_USART1: Hello world
Wrote 0x24 to EEPROM 24AA
G-Sensor Software Reset Completed
G-Sensor WHO AM I : 0x35
G-Sensor Initial Completed
Temp = 26.5      EEPROM = 0x24    G-Sensor = ( -576, 192, 4144)
  
```

Arduino UNO Socket	
15	D8
16	D9/PWM
17	D10/PWM/SS
18	D11/PWM/MOSI
19	D12/MISO
20	D13/SCK SERCOM1
21	GND
22	AREF
23	D18/SDA
24	D19/SCL



Arduino UNO Socket	
15	D8
16	D9/PWM
17	D10/PWM/SS
18	D11/PWM/MOSI
19	D12/MISO
20	D13/SCK
21	GND
22	AREF
23	D18/SDA
24	D19/SCL SERCOM3



SPI Driver Library

Harmony Driver

SPI Driver Library

- **Introduction**

The SPI Driver is a multi-client, multi-instance buffer model-based driver interface which can be used to communicate with various slave devices like EEPROM, ADC etc.

- **Key Features of SPI Driver:**

- **Multi Client:**

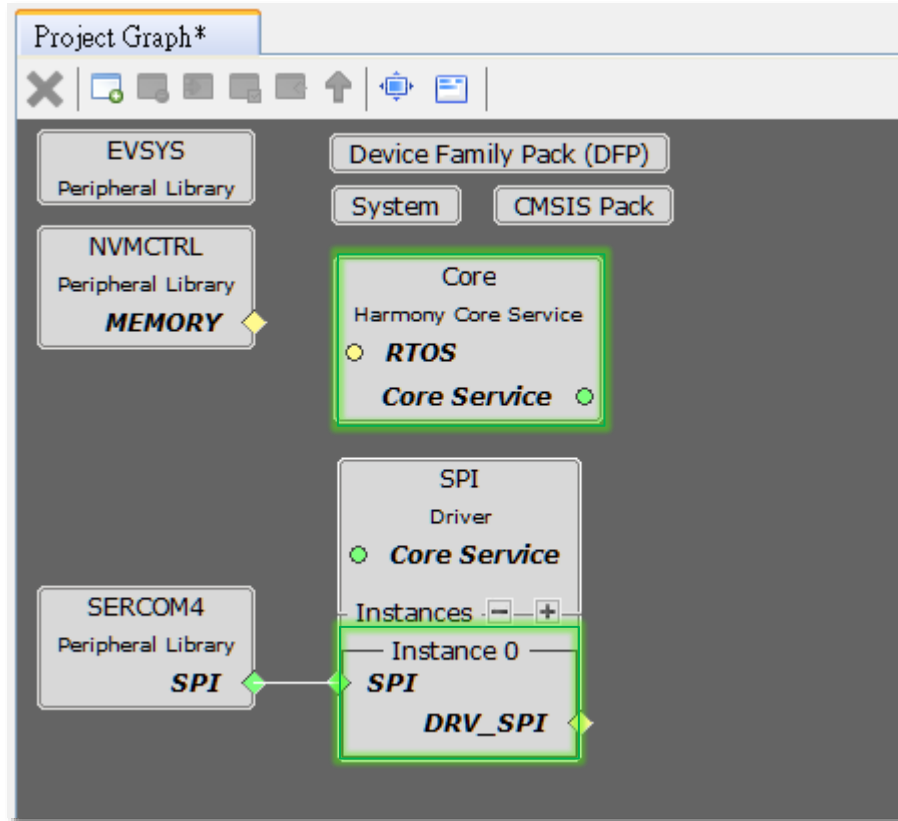
Each instance of SPI driver can have multiple clients. This means the SPI driver supports multiple slaves connected to the same SPI bus.

- **Multi Instance:**

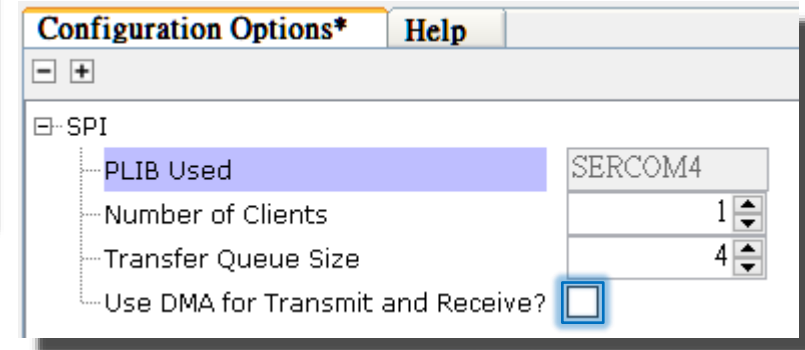
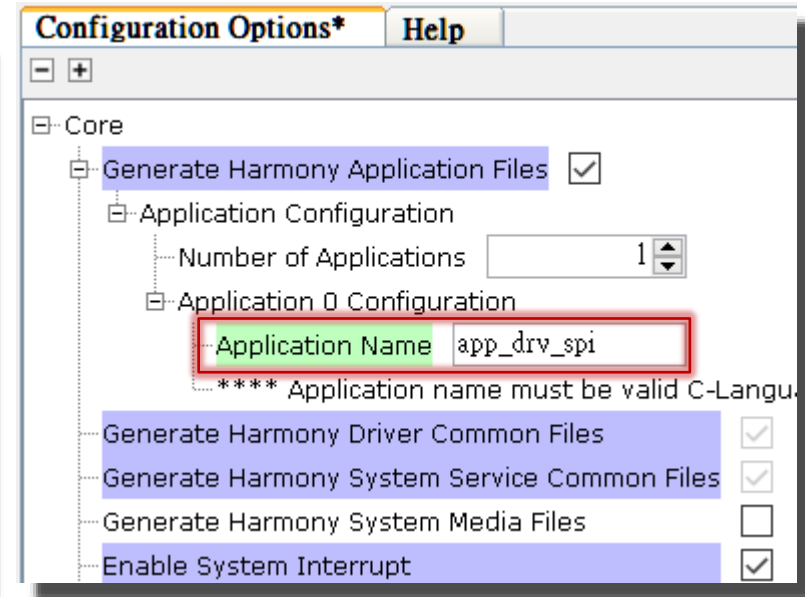
Provides interface to manage multiple instances of SPI peripheral.

- Supports Asynchronous and Synchronous modes of operation.
 - Supports Bare Metal and RTOS environments.
 - Supports DMA.

SPI Driver Library



**Create an Application for
Driver State Machine**



DMA ready

SPI Driver Library

API functions of the SPI Driver library.

- **System Functions**

DRV_SPI_Initialize

Initializes the SPI instance for the specified driver index.

DRV_SPI_Status

Gets the current status of the SPI driver module.

- **Client Setup Functions**

DRV_SPI_Open

Opens the specified SPI driver instance and returns a handle to it.

DRV_SPI_Close

Closes an opened-instance of the SPI driver.

DRV_SPI_TransferSetup

Sets the dynamic configuration of the driver including chip select pin.

DRV_SPI_TransferEventHandlerSet

Allows a client to set a transfer event handling function for the driver to call back when queued transfer has finished.

- **Transfer Functions**

DRV_SPI_ReadTransfer

This is a blocking function that receives data over SPI.

DRV_SPI_WriteTransfer

This is a blocking function that transmits data over SPI.

DRV_SPI_WriteReadTransfer

This is a blocking function that transmits and receives data over SPI.

DRV_SPI_ReadTransferAdd

Queues a read operation.

DRV_SPI_WriteTransferAdd

Queues a write operation.

DRV_SPI_WriteReadTransferAdd

Queues a write-read transfer operation.

- **Transfer Status Functions**

DRV_SPI_TransferStatusGet

Returns transfer add request status.

SPI Driver Library

API functions of the SPI Driver library. (cont.)

- Data Types and Constants

DRV_SPI_CLOCK_PHASE	Identifies SPI Clock Phase Options
DRV_SPI_CLOCK_POLARITY	Identifies SPI Clock Polarity Options
DRV_SPI_CS_POLARITY	Identifies SPI Chip Select Polarity Options
DRV_SPI_DATA_BITS	Identifies SPI bits per transfer
DRV_SPI_TRANSFER_SETUP	Defines the data required to setup the SPI transfer
DRV_SPI_TRANSFER_EVENT	Identifies the possible events that can result from a transfer add request.
DRV_SPI_TRANSFER_HANDLE	Handle identifying the transfer request queued.
DRV_SPI_TRANSFER_EVENT_HANDLER	Pointer to a SPI Driver Transfer Event handler function
DRV_SPI_TRANSFER_HANDLE_INVALID	Definition of an invalid transfer handle.

SPI Driver Library

PLIB SERCOM SPI pad configurations.

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

CTRLA		SERCOM			
DIPO	DOPO	PAD[0]	PAD[1]	PAD[2]	PAD[3]
Hardware Chip Select (SS)					
0x3	0x0	DO	SCK	SS	DI
0x0	0x1	DI	SS	DO	SCK
0x0	0x2	DI	SCK	SS	DO
0x2	0x3	DO	SS	DI	SCK
Software Chip Select (GPIO)					
0x1	0x1	--	DI	DO	SCK
0x1	0x3	DO	DI	--	SCK

CTRLA		SPI			
DIPO	DOPO	DO	SCK	SS	DI
Hardware Chip Select (SS)					
0x3	0x0	PAD[0]	PAD[1]	PAD[2]	PAD[3]
0x0	0x1	PAD[2]	PAD[3]	PAD[1]	PAD[0]
0x0	0x2	PAD[3]	PAD[1]	PAD[2]	PAD[0]
0x2	0x3	PAD[0]	PAD[3]	PAD[1]	PAD[2]
Software Chip Select (GPIO)					
0x1	0x1	PAD[2]	PAD[3]	--	PAD[1]
0x1	0x3	PAD[0]	PAD[3]	--	PAD[1]

Lab 16

Single-Client SPI OLED Display

SPI Driver Library

✖ Lab16 Single-Client SPI OLED Display

- Try add **SPI Driver** through **PLIB SERCOM4 SPI** to control the **OLED** display **SSD1306** to show graphic image and text message.
- A pre-build splash image and text font bitmap will be provided for exercise.
- Please output text result of USART/I2C Driver Labs on **OLED** display as below.

LCM: Hello World

USART1 RX: [The received byte from USART1 console]

USART2 RX: [The received byte from USART2 console]

I2C1 Temp: [The temperature from I2C1 device MCP9800A5]

I2C2 EEPOM: [The byte read from I2C1 device 24AA024E64]

I2C2 Acce1X: [The gravity X data from I2C2 device KXTJ3]

I2C2 Acce1Y: [The gravity Y data from I2C2 device KXTJ3]

I2C2 Acce1Z: [The gravity Z data from I2C2 device KXTJ3]

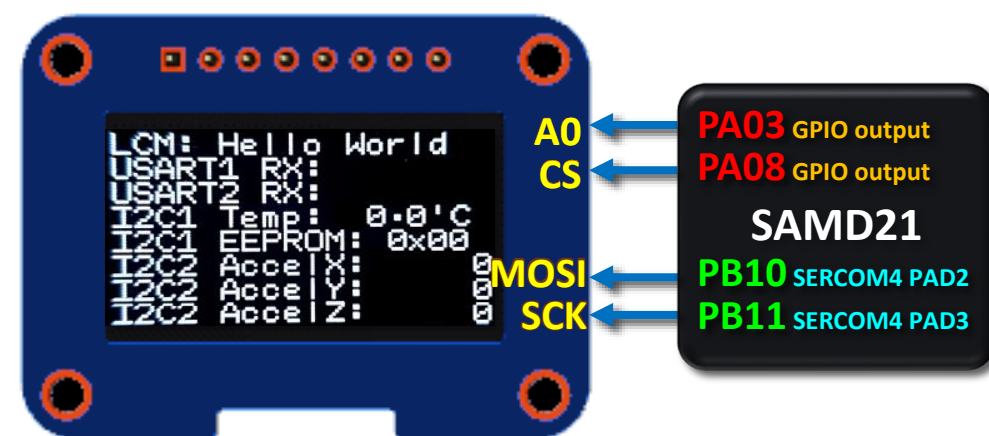
- OLED display (SSD1306) pin configuration: **(SPI Chip Select : GPIO controlled)**

GND/A0 to **PA03** (Custom Name **SSD1306_RS** , GPIO output , latch High)

CS to **PA08** (Custom Name **SSD1306_CS** , GPIO output , latch High)

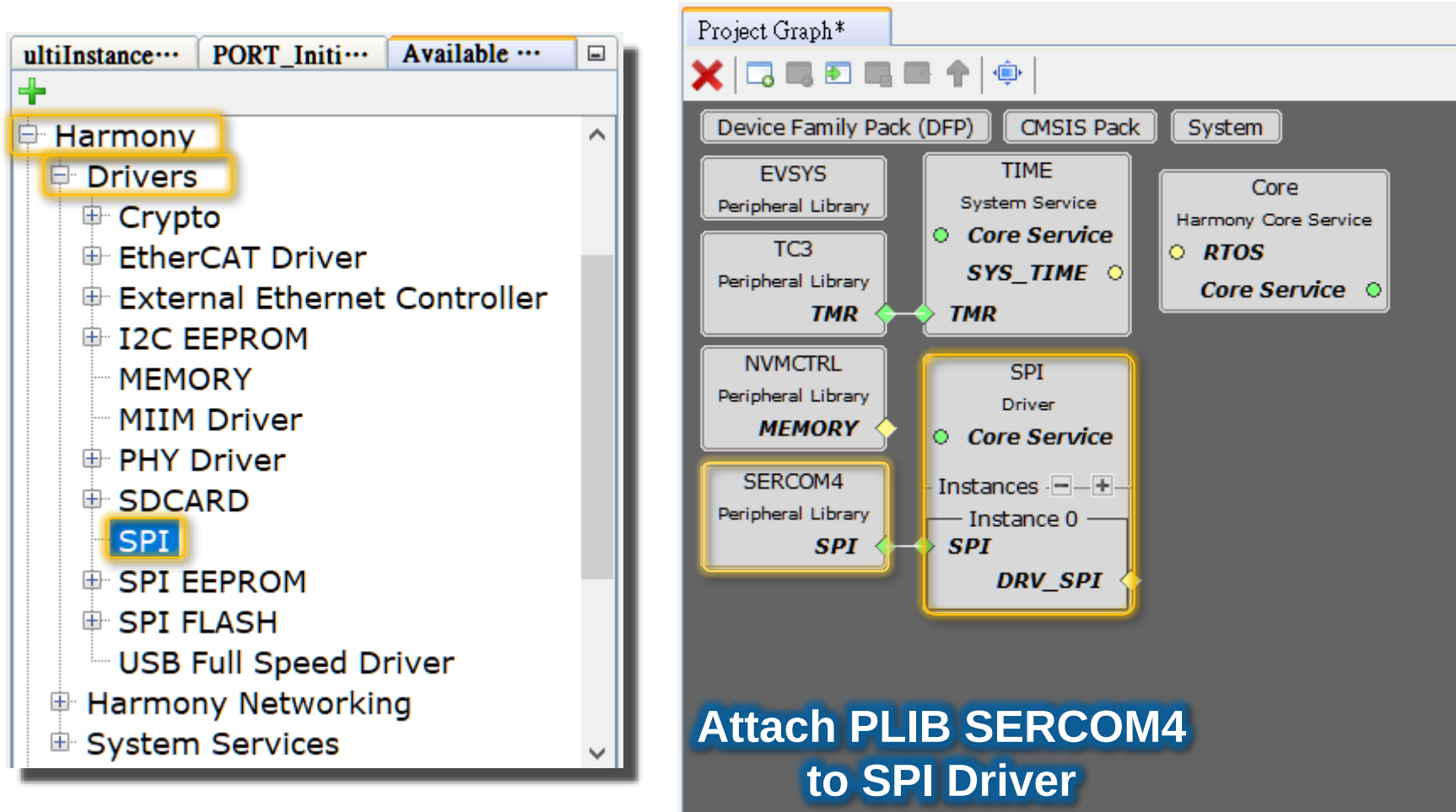
MOSI/SDA to **PB10** (**SERCOM4 PAD2**)

SCK/SCL to **PB11** (**SERCOM4 PAD3**)



⚙️ Lab16 Single-Client SPI OLED Display

- Add **SPI Driver** and attach **PLIB SERCOM4** as its hardware.



⚙️ Lab16 Single-Client SPI OLED Display

- Enable **DMA** of SPI Driver and configure pads of **PLIB SERCOM4**.

The image shows the Microchip Studio configuration interface for the SERCOM4 peripheral and the SPI driver.

SERCOM4 Configuration Options:

- Select SERCOM Operation Mode: SPI Master
- Enable Interrupts?: ☒
- Enable operation in Standby mode: ☐
- SPI Data Out Pad: DO on PAD[2], SCK on PAD[3] and SS on PAD[1] (highlighted with a yellow box and text: **DO on PAD[2] SCK on PAD[3]**)
- SPI Data In Pad Selection: SERCOM PAD[0]
- SPI Data Order: MSB is transferred first
- SPI Speed in Hz: 1,000,000
- 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
- Enable SPI Master Hardware Slave Select: ☐ (highlighted with a red box and text: **Don't use Hardware Slave Select**)
- SPI Receiver Enable: ☒

SPI Driver Configuration Options:

- PLIB Used: SERCOM4
- Number of Clients: 1
- Transfer Queue Size: 4
- Use DMA for Transmit and Receive?: ☒ (highlighted with a blue box and text: **Enable DMA in SPI Driver**)
- DMA Channel For Transmit: 0
- DMA Channel For Receive: 1

Peripheral Library:

- NVMCTRL: Peripheral Library MEMORY
- SERCOM4: Peripheral Library SPI
- SPI: Driver Core Service
- Instances: Instance 0 (DRV_SPI)

✖ Lab16 Single-Client SPI OLED Display

- Configure **PA03** as **GPIO output**, Latch **High**, Custom named as **SSD1306_RS**
- Configure **PA08** as **GPIO output**, Latch **High**, Custom named as **SSD1306_CS**
- Enable **PLIB SERCOM4** pads of SPI Driver Instance 0.

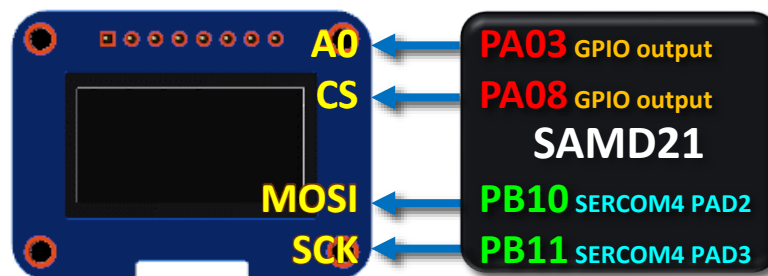
Project Graph* Pin Diagram × Pin Table × Pin Settings ×									
Order:	Pins	Table View	☒ Easy View						
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
4	PA03	SSD1306_RS	GPIO	Digital	Out	High	☐	☐	NORMAL
13	PA08	SSD1306_CS	GPIO	Digital	Out	High	☐	☐	NORMAL
19	PB10		SERCOM4_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
20	PB11		SERCOM4_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL

PA03 as **GPIO output**, custom name **SSD1306_RS**, Latch to **High**

PA08 as **GPIO output**, custom name **SSD1306_CS**, Latch to **High**

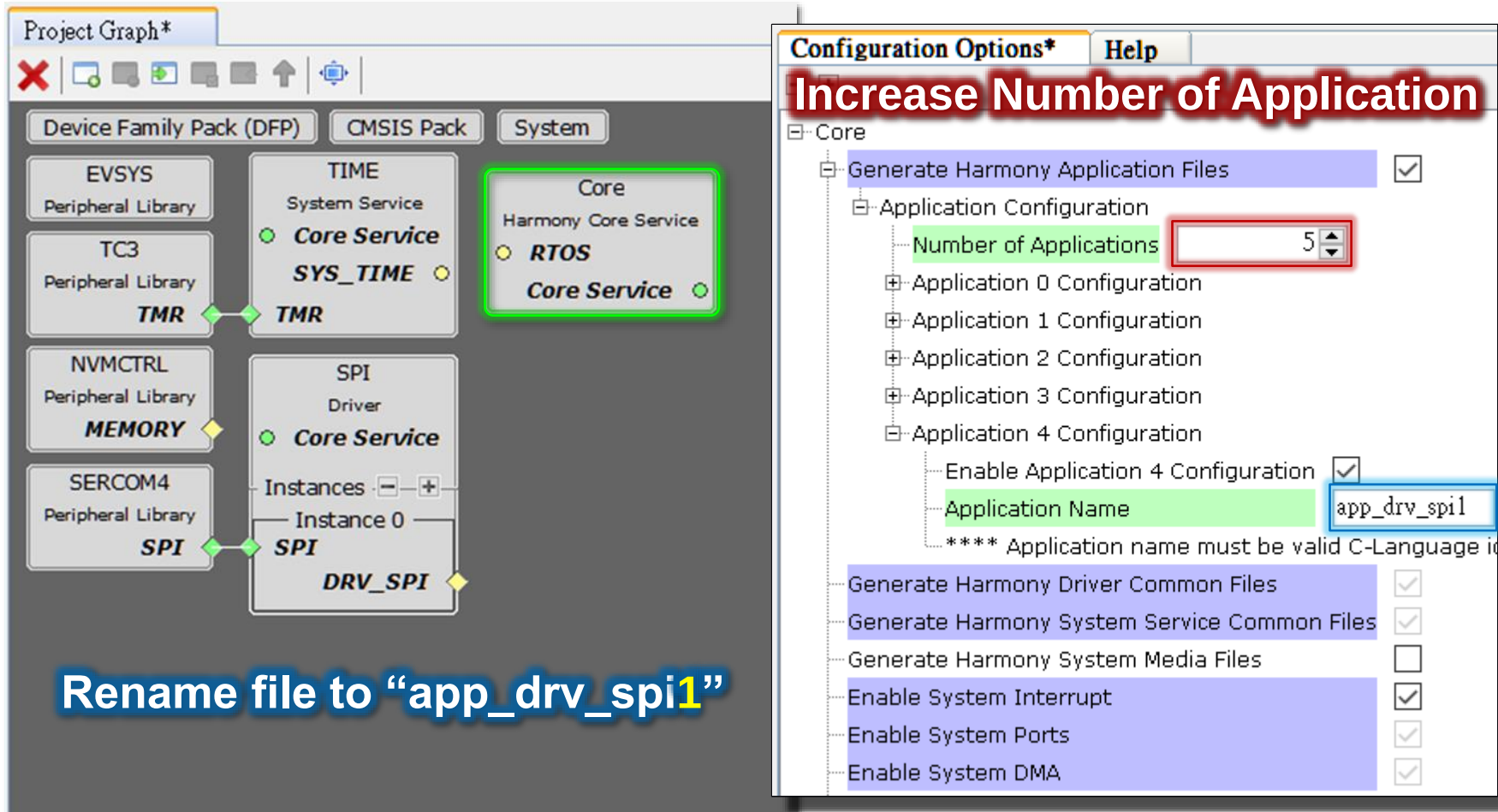
Configure **PB10** as **SERCOM4_PAD2**

Configure **PB11** as **SERCOM4_PAD3**



✖ Lab16 Single-Client SPI OLED Display

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**5**” and given the Application Name to “**app_drv_spi1**”.

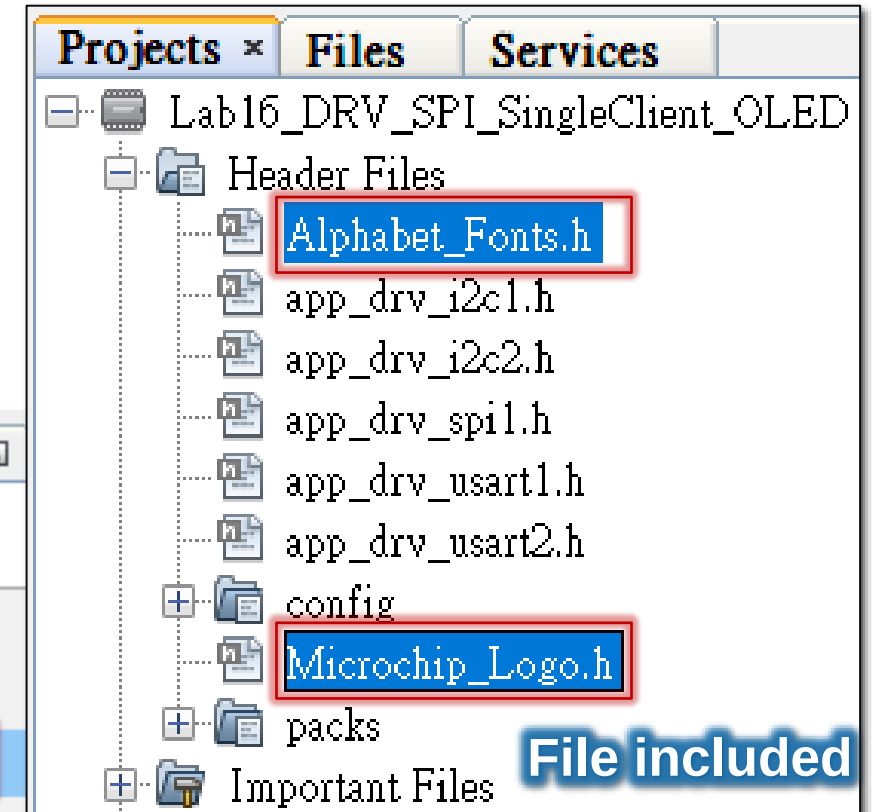
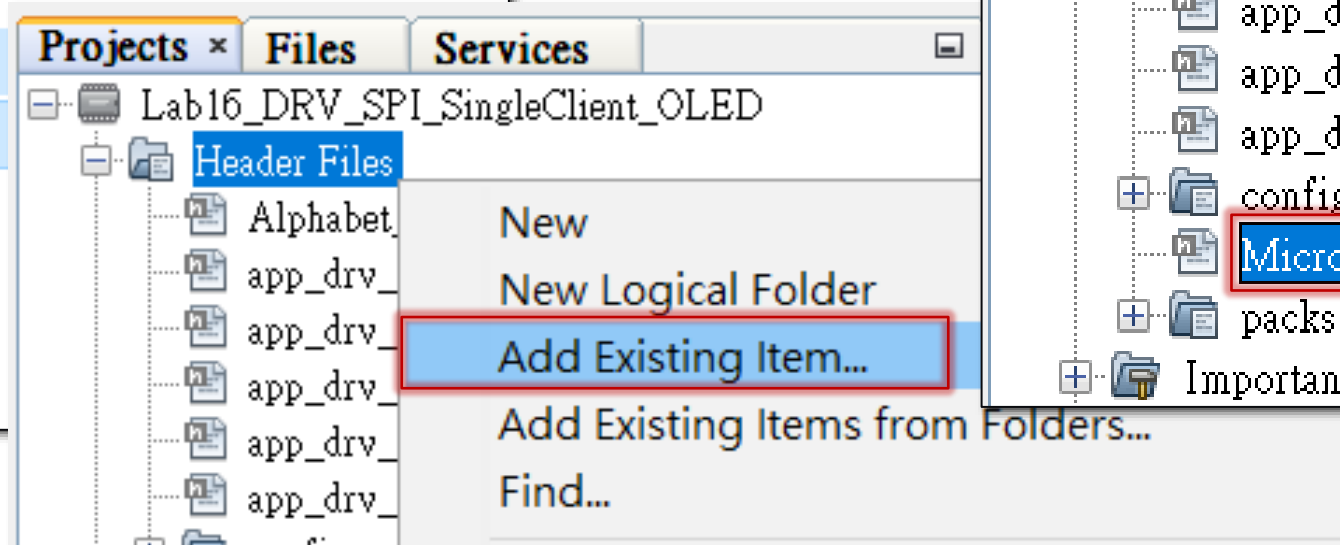
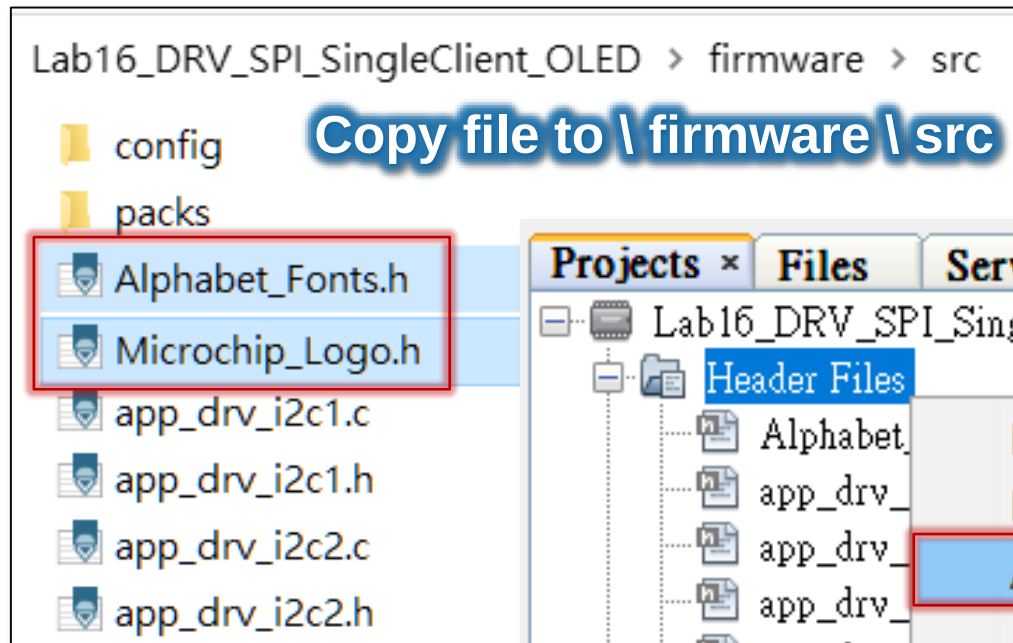


The image shows two windows from the IDE. The left window, titled "Project Graph*", displays a block diagram of the system components. The "Core" block, representing the Harmony Core Service, is highlighted with a green border. It contains a "Core Service" and an "RTOS" component. The "Core Service" is connected to the "SYS_TIME" component, which is in turn connected to the "TMR" component. The "Core Service" is also connected to the "SPI" component, which is connected to the "DRV_SPI" component. The "SPI" component is connected to the "SERCOM4" component, which is connected to the "SPI" component. The "SPI" component is connected to the "NVMCTRL" component, which is connected to the "MEMORY" component. The "SPI" component is connected to the "TC3" component, which is connected to the "Peripheral Library". The "SPI" component is connected to the "EVSYS" component, which is connected to the "Peripheral Library".

The right window, titled "Configuration Options*", shows the configuration for the "Core" component. The "Generate Harmony Application Files" option is checked. Under "Application Configuration", the "Number of Applications" is set to 5. The "Application Name" for Application 4 is set to "app_drv_spi1". A red text overlay "Increase Number of Application" is positioned above the "Number of Applications" field. A blue text overlay "Rename file to 'app_drv_spi1'" is positioned below the "Application Name" field.

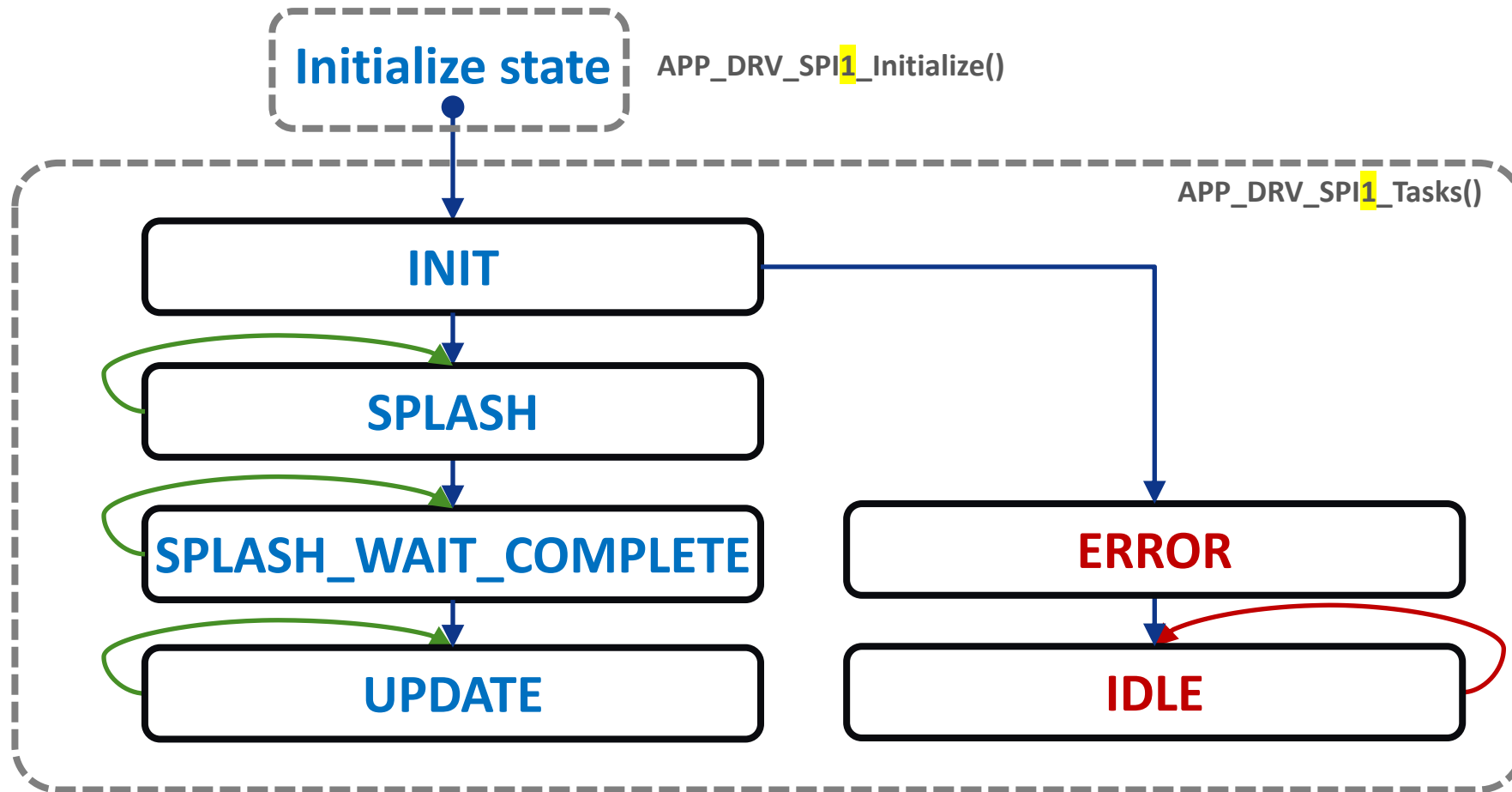
✖ Lab16 Single-Client SPI OLED Display

- Copy pre-build bitmap header files to project folder `\firmware\src`.
 - Splash Logo bitmap image “[Microchip_Logo.h](#)”.
 - Text Font bitmap image “[Alphabet_Fonts.h](#)”.
- Add two header files to project in MPLAB X IDE.



⚙️ Lab16 Single-Client SPI OLED Display

- The state machine to implement in `app_drv_spi1.c` as below.



⚙️ Lab16 Single-Client SPI OLED Display

- Add states in “app_drv_spi1.h”.

```
#ifndef _APP_DRV_SPI1_H
#define _APP_DRV_SPI1_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_SPI1_STATE_INIT=0,
    // APP_DRV_SPI1_STATE_SERVICE_TASKS,
    APP_DRV_SPI1_STATE_SPLASH,
    APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE,
    APP_DRV_SPI1_STATE_UPDATE,
    APP_DRV_SPI1_STATE_ERROR,
    APP_DRV_SPI1_STATE_IDLE,
} APP_DRV_SPI1_STATES;

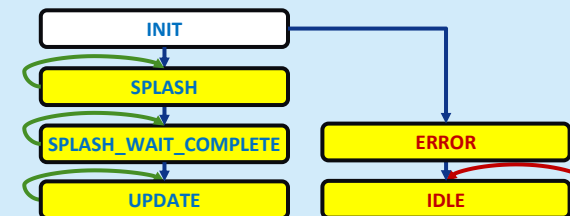
typedef struct
{
    APP_DRV_SPI1_STATES state;
} APP_DRV_SPI1_DATA;

void APP_DRV_SPI1_Initialize ( void );
void APP_DRV_SPI1_Tasks( void );
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



✖ Lab16 Single-Client SPI OLED Display

- Construct new states flow of task in “app_drv_spi1.c”.

```
/******
```

```
Function:
```

```
void APP_DRV_SPI1_Tasks ( void )
```

```
...
```

```
void APP_DRV_SPI1_Tasks ( void )
```

```
{
```

```
switch ( app_drv_spi1Data.state )
```

```
{
```

```
case APP_DRV_SPI1_STATE_INIT:
```

```
app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;
```

```
break;
```

```
case APP_DRV_SPI1_STATE_SPLASH:
```

```
app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE;
```

```
break;
```

```
case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:
```

```
app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;
```

```
break;
```

```
case APP_DRV_SPI1_STATE_UPDATE:
```

```
break;
```

```
case APP_DRV_SPI1_STATE_ERROR:
```

```
app_drv_spi1Data.state = APP_DRV_SPI1_STATE_IDLE;
```

```
break;
```

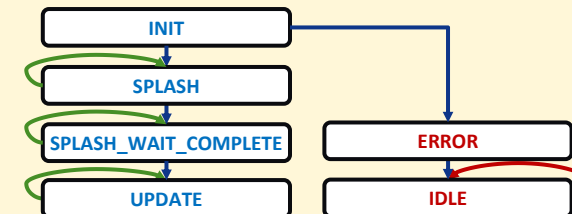
```
case APP_DRV_SPI1_STATE_IDLE:
```

```
break;
```

```
}
```

```
}
```

Overwrite original example code



✂ Lab16 Single-Client SPI OLED Display

- Header include and Global Declarations in “app_drv_spi1.c”.

Keep this line

```
#include "app_drv_spi1.h"
#include "app_drv_usart1.h"
#include "driver/spi/drv_spi.h"
#include "system/time/sys_time.h"
#include "peripheral/port/plib_port.h"
#include "Alphabet_Fonts.h"
#include "Microchip_Logo.h"
#include <stdio.h>
#include <string.h>
```

DRV_USART1 for debug message
SYS_TIME for delay function
PLIB_PORT for GPIO control
Alphabet_Fonts.h for Font bitmap
Microchip_Logo.h for Splash Logo

```
// *****
// *****
// Section: Global Data Definitions
// *****
// *****
```

```
#define LCM_WIDTH 128
#define LCM_HEIGHT 64
int LogoX=127;
const uint8_t LCM_InitCMD[] = {
    0xAE, 0xD5, 0x80, 0xA8, 0x3F, 0xD3, 0x00, 0x40, 0x8D, 0x14, 0x20, 0x02, 0xA1,
    0xC8, 0xDA, 0x12, 0x81, 0xCF, 0xD9, 0xF1, 0xDB, 0x40, 0xA4, 0xA6, 0xAF
};
enum {
    LCM_BUS_CMD = 0,
    LCM_BUS_DAT
};
```

Splash animation initial position X
OLED initial commands

OLED BUS selector

```
DRV_HANDLE SPI1Handle;
SYS_TIME_HANDLE SplashTimerHandle;
```

SPI handle
Delay Timer handle

```
// *****
```

⚙️ Lab16 Single-Client SPI OLED Display

- Implement LCM local function for bus write and draw bitmap.

```
// *****  
// Section: Application Local Functions  
// *****  
// *****  
void LCM_BusWrite( int bus, void *bytes, size_t counts )  
{  
    DRV_SPI_TRANSFER_HANDLE SPI_WriteHandle;  
  
    if ( bus==LCM_BUS_CMD ) SSD1306_RS_Clear();  
    else if( bus==LCM_BUS_DAT ) SSD1306_RS_Set();  
  
    DRV_SPI_WriteTransferAdd( SPI_WriteHandle, bytes, counts, &SPI_WriteHandle );  
    while( DRV_SPI_TransferStatusGet(SPI_WriteHandle)!=DRV_SPI_TRANSFER_EVENT_COMPLETE );  
}  
  
void LCM_SetCursor( uint8_t x, uint8_t y )  
{  
    uint8_t OUT[3] = { 0x0F & x, 0x10 + (x >> 4), 0xB0 + y };  
    LCM_BusWrite( LCM_BUS_CMD, (void*)OUT, 3 );  
}  
  
void LCM_DrawBitmap( uint8_t sx, uint8_t sy, uint8_t width, uint8_t height,  
                    const unsigned char *byte )  
{  
    for( int y=0; y<(height/8); y++ )  
    {  
        LCM_SetCursor( sx, sy+y );  
        LCM_BusWrite( LCM_BUS_DAT, (void*)byte, width-sx );  
        byte+=width;  
    }  
}
```

**OLED Bus write function
Control GPIO to choose
Command or Data transfer**

OLED Cursor Set

Draw Bitmap image

⚙️ Lab16 Single-Client SPI OLED Display

- Driver Open, OLED initial in state **INIT** and handling **ERROR** state.

```

void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
    case APP_DRV_SPI1_STATE_INIT:
        SPI1Handle = DRV_SPI_Open(DRV_SPI_INDEX_0, DRV_IO_INTENT_WRITE);
        if(SPI1Handle == DRV_HANDLE_INVALID)
        {
            app_drv_spi1Data.state = APP_DRV_SPI1_STATE_ERROR;
            break;
        }

        DRV_SPI_TRANSFER_SETUP SPI1Setup;
        SPI1Setup.baudRateInHz = 1000000;
        SPI1Setup.clockPhase = DRV_SPI_CLOCK_PHASE_VALID_LEADING_EDGE;
        SPI1Setup.clockPolarity = DRV_SPI_CLOCK_POLARITY_IDLE_LOW;
        SPI1Setup.dataBits = DRV_SPI_DATA_BITS_8;
        SPI1Setup.chipSelect = (SYS_PORT_PIN)SSD1306_CS_PIN;
        SPI1Setup.csPolarity = DRV_SPI_CS_POLARITY_ACTIVE_LOW;
        DRV_SPI_TransferSetup(SPI1Handle, &SPI1Setup);

        LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );

        Keep this line app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;
        break;
        ...

    case APP_DRV_SPI1_STATE_ERROR:
        USART1printf("DRV_SPI1 error!\r\n");

        Keep this line app_drv_spi1Data.state = APP_DRV_I2C1_STATE_IDLE;
        ...
    }
}

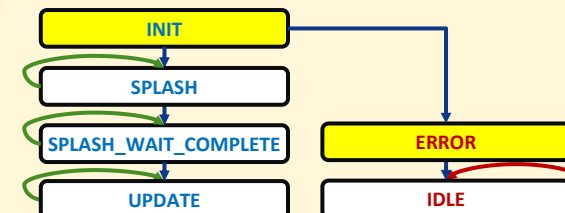
```

SPI Driver Open for write only

**Assign SSD1306_CS (PA08) in
SPI Transfer Setup for Software Chip Select**

Write OLED initial commands

Handling error state message



⚙️ Lab16 Single-Client SPI OLED Display

- Draw animated image in state **SPLASH** and a delay while finish.

```
void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
        case APP_DRV_SPI1_STATE_INIT:
            LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );

            app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;
            break;

        case APP_DRV_SPI1_STATE_SPLASH:
            LCM_DrawBitmap( LogoX, 0, LCM_WIDTH, LCM_HEIGHT, Microchip_Logo );
            LogoX = LogoX-2;
            if( LogoX<0 )
            {
                SYS_TIME_DelayMS(1000, &SplashTimerHandle);
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE;
            }
            break;

        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == true )
            {
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;
            }
            break;

        case APP_DRV_SPI1_STATE_UPDATE:
            break;

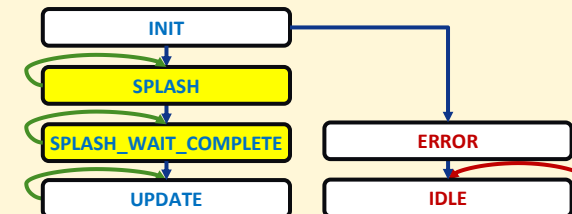
        ...
    }
}
```

Draw animated logo bitmap

Setup one second delay after finish

Keep this line

Wait one second delay



⚙️ Lab16 Single-Client SPI OLED Display

- Please wire external I2C G-Sensor EVB (ACCEL 7 Click) to **SERCOM1**,
If you **have not** implemented the **I2C Auto Probe Advanced Exercise** from **Lab15**.

```
void SYS_Tasks ( void )
{
    /* Maintain Device Drivers */

    /* Maintain Middleware & Other Libraries */

    /* Maintain the application's state machine. */
    /* Call Application task APP_DRV_USART1. */
    APP_DRV_USART1_Tasks();

    /* Call Application task APP_DRV_USART2. */
    APP_DRV_USART2_Tasks();

    /* Call Application task APP_DRV_I2C1. */
    APP_DRV_I2C1_Tasks();

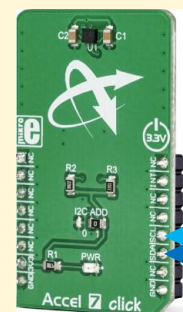
    /* Call Application task APP_DRV_I2C2. */
    APP_DRV_I2C2_Tasks();

    /* Call Application task APP_DRV_SPI1. */
    APP_DRV_SPI1_Tasks();
}
```

No I2C Auto Probe implemented in

...

Task.c



Arduino UNO Socket	
15	D8
16	D9/PWM
17	D10/PWM/SS
18	D11/PWM/MOSI
19	D12/MISO
20	D13/SCK SERCOM1
21	GND
22	AREF
23	D18/SDA
24	D19/SCL SERCOM3

⚙️ Lab16 Single-Client SPI OLED Display

- Please comment out the `APP_DRV_I2C2_Tasks()` in `task.c`,
If you **have not** an external I2C G-Sensor EVB ([ACCEL 7 Click](#)) from [Lab15](#).

```
void SYS_Tasks ( void )
{
    /* Maintain Device Drivers */

    /* Maintain Middleware & Other Libraries */

    /* Maintain the application's state machine. */
    /* Call Application task APP_DRV_USART1. */
    APP_DRV_USART1_Tasks();

    /* Call Application task APP_DRV_USART2. */
    APP_DRV_USART2_Tasks();

    /* Call Application task APP_DRV_I2C1. */
    APP_DRV_I2C1_Tasks();

    /* Call Application task APP_DRV_I2C2. */
    // APP_DRV_I2C2_Tasks();

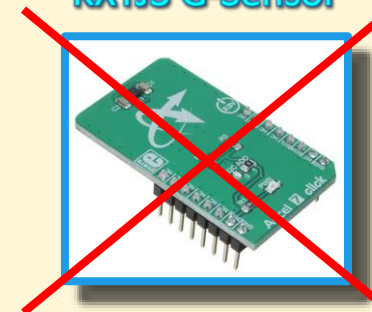
    /* Call Application task APP_DRV_SPI1. */
    APP_DRV_SPI1_Tasks();

    ...
}
```

Comment out if no G-Sensor

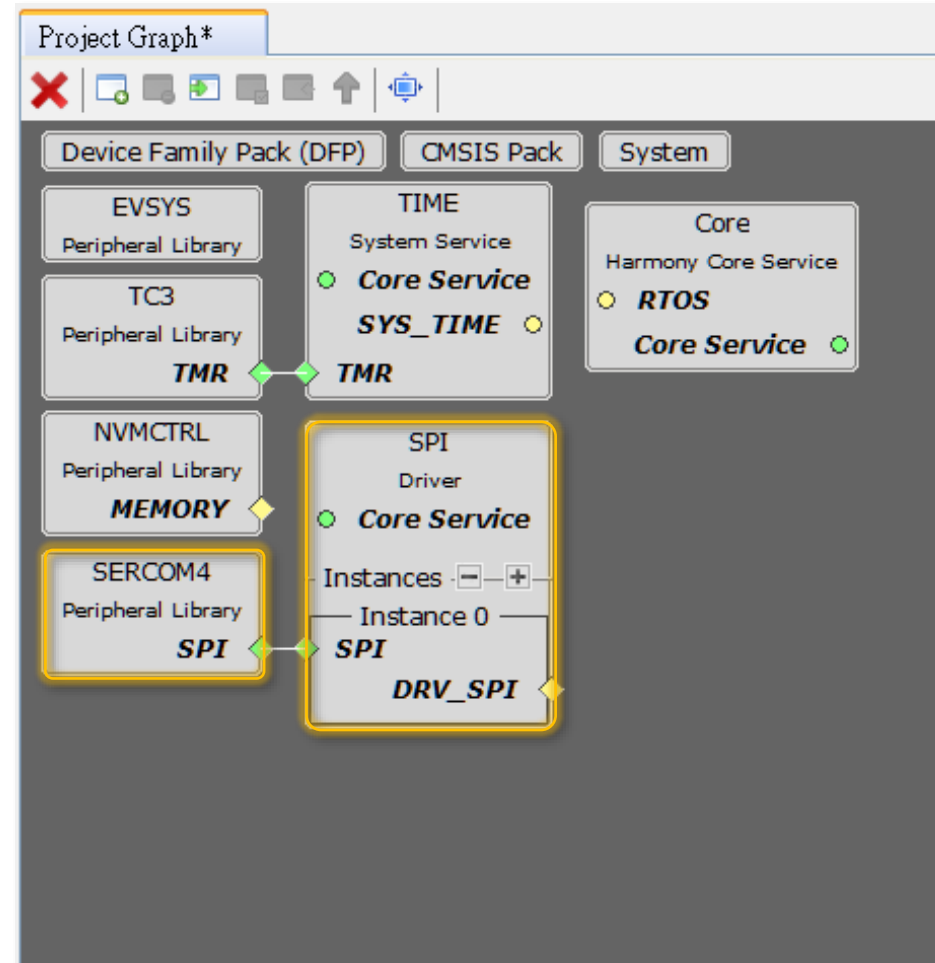
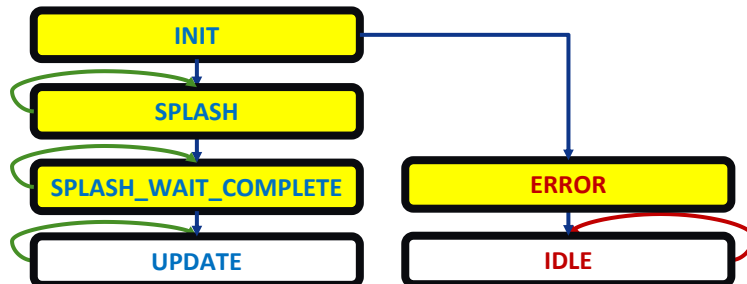
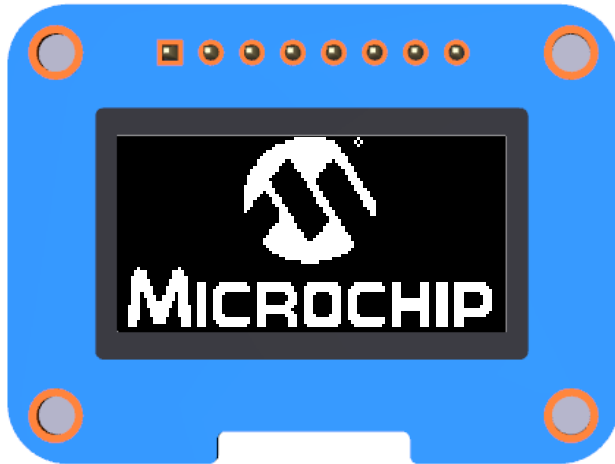
Task.c

MIKORE
ACCEL 7 CLICK
KXTJ3 G-Sensor



⚙️ Lab16 Single-Client SPI OLED Display

- Compiler and programming your project to EVB.
- The OLED will show animated Microchip Logo



✂ Lab16 Single-Client SPI OLED Display

- Let's continue implement to display the text message.

```
#include "app_drv_spi1.h"
#include "app_drv_usart1.h"
#include "driver/spi/drv_spi.h"
#include "system/time/sys_time.h"
#include "peripheral/port/plib_port.h"
#include "Alphabet_Fonts.h"
...

// *****
// *****
// Section: Global Data Definitions
// *****
// *****

#define LCM_WIDTH 128
#define LCM_HEIGHT 64
#define FONT_WIDTH 7
#define FONT_HEIGHT 8
char OutString[20];
char LCM_Usart1Rx;
char LCM_Usart2Rx;
float LCM_I2c1Temp;
char LCM_I2c1EERPOM;
int LCM_I2c2Accel[3];
char Alphabet[95] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz\
~!@#$%^&()[]{}_\\";

int LogoX=127;
const uint8_t LCM_InitCMD[] = {
    0xAE, 0xD5, 0x80, 0xA8, 0x3F, 0xD3, 0x00, 0x40, 0x8D, 0x14, 0x20, 0x02, 0xA1,
    0xC8, 0xDA, 0x12, 0x81, 0xCF, 0xD9, 0xF1, 0xDB, 0x40, 0xA4, 0xA6, 0xAF
};
...
```

Insert between this

Font size (depend to bitmap of Font)

Data collect for OLED text output

Insert between this

To index the font offset in bitmap

✖ Lab16 Single-Client SPI OLED Display

- Implement LCM clear and draw text string functions.

```
// *****  
// Section: Application Local Functions  
// *****  
// *****  
void LCM_BusWrite( int bus, void *bytes, size_t counts ) { ... }  
void LCM_SetCursor( uint8_t x, uint8_t y ) { ... }  
void LCM_DrawBitmap( uint8_t sx, uint8_t sy, uint8_t width, uint8_t height, ...) { ... }  
  
void LCM_Clear( void )  
{  
    unsigned char LCM_Blank[1024];  
    memset(LCM_Blank, 0, 1024);  
    LCM_DrawBitmap(0, 0, LCM_WIDTH, LCM_HEIGHT, LCM_Blank);  
}  
  
void LCM_DrawFont( uint8_t sx, uint8_t row, char ch )  
{  
    int chIdx = 0;  
    while( ch != Alphabet[chIdx] ) { chIdx++; }  
    LCM_SetCursor(sx, row);  
    LCM_BusWrite( LCM_BUS_DAT, (void*)(Alphabet_Fonts+(chIdx*FONT_WIDTH)), FONT_WIDTH );  
}  
  
void LCM_DrawString( uint8_t sx, uint8_t row, char *str )  
{  
    for( int x=0 ; x<strlen(str); x++ )  
    {  
        LCM_DrawFont( sx+(x*FONT_WIDTH), row, str[x] );  
    }  
}  
  
...
```

Insert after this

OLED screen clear

Find offset in font bitmap
and output to target position

Draw string fonts

⚙️ Lab16 Single-Client SPI OLED Display

- Add OLED screen clear before start splash animation and end of animation.

```
void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
        case APP_DRV_SPI1_STATE_INIT:
            SSD1306_RESET_Set();

            Insert between this LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );
                               LCM_Clear();

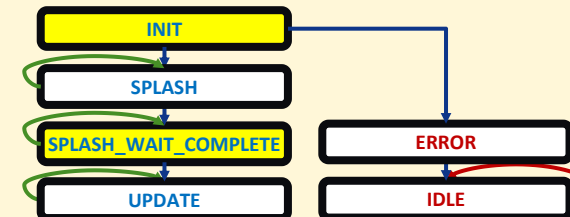
            Insert between this app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;
                               break;

        case APP_DRV_SPI1_STATE_SPLASH:
            ...

        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == true )
            {
                Insert between this LCM_Clear();
                Insert between this app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;
            }
            break;

        case APP_DRV_SPI1_STATE_UPDATE:
            break;

        case APP_DRV_SPI1_STATE_ERROR:
            USART1printf("DRV_SPI1 error!\r\n");
            ...
    }
}
```



⚙️ Lab16 Single-Client SPI OLED Display

- Output text message to OLED in state **UPDATE**.

```
void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
        case APP_DRV_SPI1_STATE_INIT: ...
        case APP_DRV_SPI1_STATE_SPLASH: ...
        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE: ...

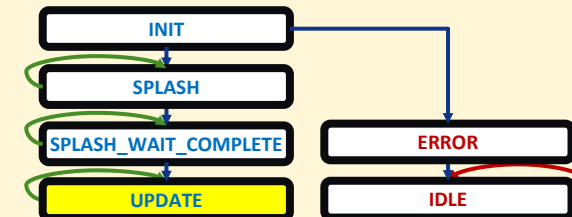
        case APP_DRV_SPI1_STATE_UPDATE:
            sprintf( OutString, "LCM: Hello World" );
            LCM_DrawString(0, 0, OutString);
            sprintf( OutString, "USART1 RX: %c", LCM_Usart1Rx );
            LCM_DrawString(0, 1, OutString);
            sprintf( OutString, "USART2 RX: %c", LCM_Usart2Rx );
            LCM_DrawString(0, 2, OutString);
            sprintf( OutString, "I2C1 Temp: %2.1f'C", LCM_I2c1Temp );
            LCM_DrawString(0, 3, OutString);
            sprintf( OutString, "I2C1 EEPROM: 0x%02X", LCM_I2c1EEPROM );
            LCM_DrawString(0, 4, OutString);
            sprintf( OutString, "I2C2 AccelX: %5d", LCM_I2c2Accel[0] );
            LCM_DrawString(0, 5, OutString);
            sprintf( OutString, "I2C2 AccelY: %5d", LCM_I2c2Accel[1] );
            LCM_DrawString(0, 6, OutString);
            sprintf( OutString, "I2C2 AccelZ: %5d", LCM_I2c2Accel[2] );
            LCM_DrawString(0, 7, OutString);
            Keep this line break;

        case APP_DRV_SPI1_STATE_ERROR:
            USART1printf("DRV_SPI1 error!\r\n");

            app_drv_spi1Data.state = APP_DRV_I2C1_STATE_IDLE;
            break;
        ...
    }
}
```

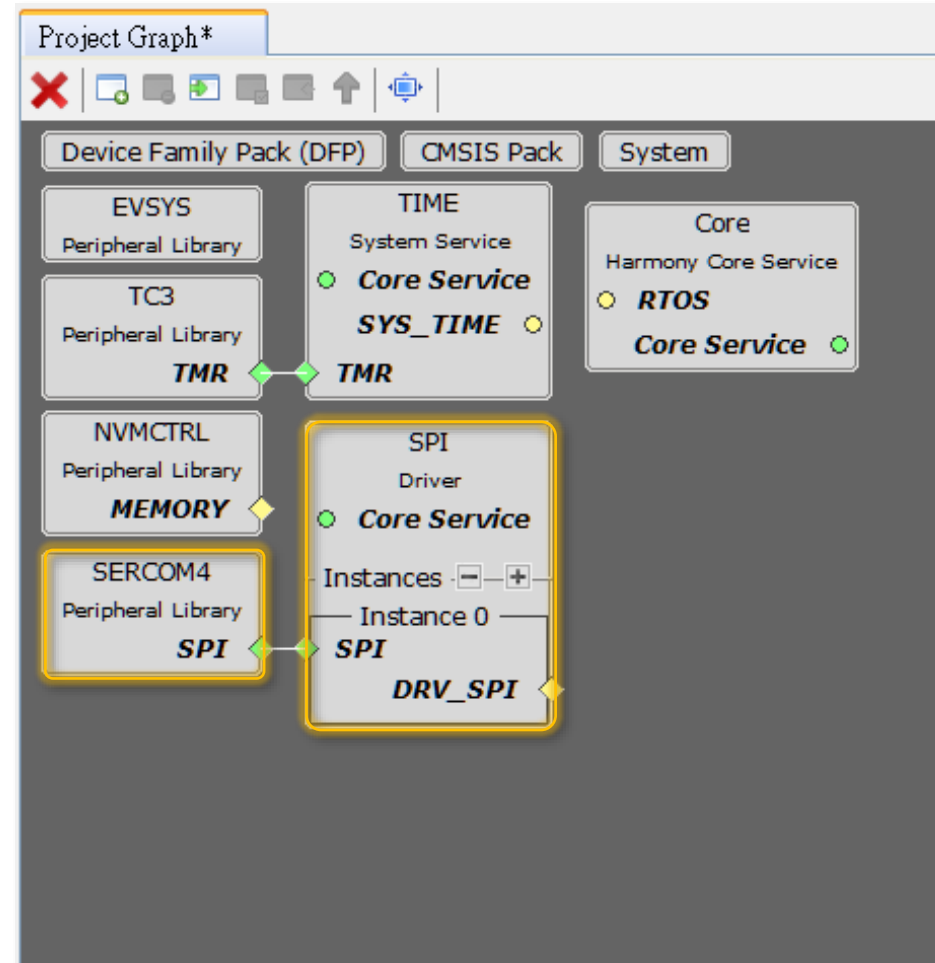
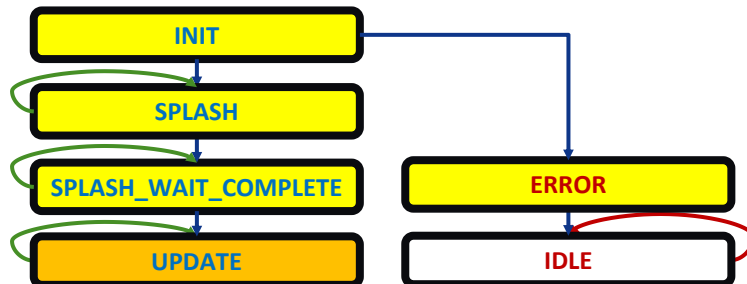


LCM: Hello World
USART1 RX:
USART2 RX:
I2C1 Temp: 0.0'C
I2C1 EEPROM: 0x00
I2C2 AccelX: 0
I2C2 AccelY: 0
I2C2 AccelZ: 0



✖ Lab16 Single-Client SPI OLED Display

- Compiler and programming your project to EVB.
- The OLED will output text message after splash animation.



⚙️ Lab16 Single-Client SPI OLED Display

- Extern variables in `app_drv_spi1.h` for collect data from USART and I2C Drivers.

```
#ifndef _APP_DRV_SPI1_H
#define _APP_DRV_SPI1_H

typedef enum
{
    /* Application's state machine's initial state. */
    APP_DRV_SPI1_STATE_INIT=0,
    // APP_DRV_SPI1_STATE_SERVICE_TASKS,
    APP_DRV_SPI1_STATE_SPLASH,
    APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE,
    APP_DRV_SPI1_STATE_UPDATE,
    APP_DRV_SPI1_STATE_ERROR,
    APP_DRV_SPI1_STATE_IDLE,
} APP_DRV_SPI1_STATES;
...
```

```
void APP_DRV_SPI1_Initialize ( void );
void APP_DRV_SPI1_Tasks( void );
```

```
extern char LCM_Usart1Rx;
extern char LCM_Usart2Rx;
extern float LCM_I2c1Temp;
extern char LCM_I2c1EERPOM;
extern int LCM_I2c2Accel[3];
```

```
//DOM-IGNORE-BEGIN
#ifdef __cplusplus
}
#endif
//DOM-IGNORE-END
```

```
#endif /* _APP_DRV_SPI1_H */
```

Extern variables for collect data

✂ Lab16 Single-Client SPI OLED Display

- Modify state **READ_WAIT_COMPLETE** in “app_drv_usart1.c”.

```
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "app_drv_spi1.h"
#include "driver/usart/drv_usart.h"
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
```

**Include
app_drv_spi1.h**

...

```
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    { ...
    case APP_DRV_USART1_STATE_READ:
        ...
```

```
    case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
        if(Uart1ReadComplete == true)
        {
```

Insert between this

```
            Uart1ReadComplete = false;
```

```
            LCM_Usart1Rx = Uart1ReadBuf[0];
```

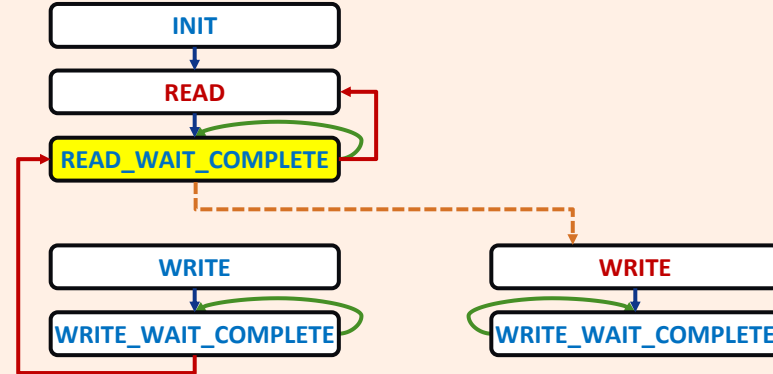
Collect Uart1ReadBuf[0]

Insert between this

```
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE;
```

```
        }
        break;
```

```
    case APP_DRV_USART1_STATE_WRITE:
        ...
```



app_drv_usart1.c

⚙️ Lab16 Single-Client SPI OLED Display

- Modify state **READ_WAIT_COMPLETE** in “app_drv_usart2.c”.

```
#include "app_drv_usart2.h"
#include "app_drv_usart1.h"
#include "app_drv_spi1.h"
#include "driver/usart/drv_usart.h"
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
```

**Include
app_drv_spi1.h**

...

```
void APP_DRV_USART2_Tasks ( void )
{
    switch ( app_drv_usart2Data.state )
    { ...
    case APP_DRV_USART2_STATE_READ:
        ...

    case APP_DRV_USART2_STATE_READ_WAIT_COMPLETE:
```

Insert between this

```
    {
        Uart2ReadComplete = false;
```

```
        LCM_Usart2Rx = Uart2ReadBuf[0];
```

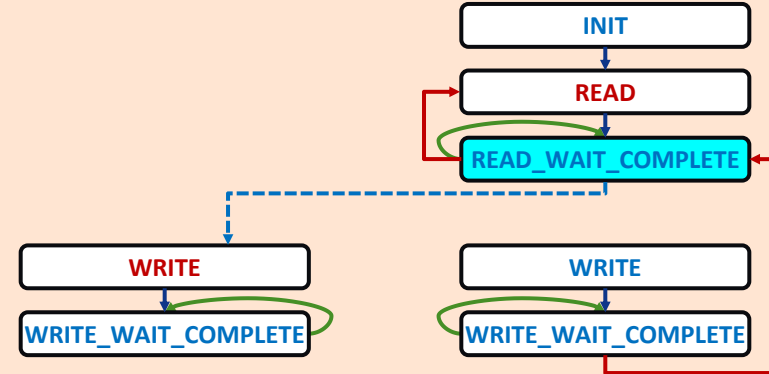
Collect Uart2ReadBuf[0]

Insert between this

```
        app_drv_usart2Data.state = APP_DRV_USART2_STATE_READ;
        app_drv_usart1Data.state = APP_DRV_USART1_STATE_WRITE;
```

```
    }
    break;
```

```
    case APP_DRV_USART2_STATE_WRITE:
        ...
```



app_drv_usart2.c

✂ Lab16 Single-Client SPI OLED Display

- Modify state **WAIT_COMPLETE_9800** and **WAIT_COMPLETE_24AA** in “app_drv_i2c1.c”.

```
#include "app_drv_i2c1.h"
#include "app_drv_usart1.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            if(I2C1Complete_9800 == true)
            {
                I2C1Complete_9800 = false;

                LCM_I2c1Temp = (float)((I2C1ReadBuf_9800[0]<<4)|
                                      (I2C1ReadBuf_9800[1]>>4))/16.0f;
                USART1printf("Temp = %2.1f\r",
                             (float)((I2C1ReadBuf_9800[0]<<4)|(I2C1ReadBuf_9800[1]>>4))/16.0f);

                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_24AA;
            }
            ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA:
            if(I2C1Complete_24AA == true)
            {
                I2C1Complete_24AA = false;

                LCM_I2c1EEPROM = I2C1ReadBuf_24AA[0];
                USART1printf("\t\tEEPROM = 0x%X\r", I2C1ReadBuf_24AA[0]);

                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            }
            ...
    }
}
```

Include app_drv_spi1.h

Collect Temperature result

Collect EEPROM read byte

app_drv_i2c1.c

Flowchart:

```
graph TD
    INIT[INIT] --> WRITE_READ_9800[WRITE_READ_9800]
    WRITE_READ_9800 --> WAIT_COMPLETE_9800[WAIT_COMPLETE_9800]
    WAIT_COMPLETE_9800 --> WRITE_READ_24AA[WRITE_READ_24AA]
    WRITE_READ_24AA --> WAIT_COMPLETE_24AA[WAIT_COMPLETE_24AA]
    WAIT_COMPLETE_24AA --> ERROR[ERROR]
    ERROR --> IDLE[IDLE]
    IDLE --> INIT
```

Insert between this (pointing to the first case block)

Insert between this (pointing to the first case block)

Insert between this (pointing to the second case block)

Insert between this (pointing to the second case block)

✂ Lab16 Single-Client SPI OLED Display

- Modify state **WAIT_COMPLETE_AXIS_TJ3** in “app_drv_i2c2.c”.

```
#include "app_drv_i2c2.h"
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        ...
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];
                if(++ AxisOutRegIdx >= 6)
                {
                    LCM_I2c2Accel[0] = (short)(AxisOutByte[1]<<8|AxisOutByte[0]);
                    LCM_I2c2Accel[1] = (short)(AxisOutByte[3]<<8|AxisOutByte[2]);
                    LCM_I2c2Accel[2] = (short)(AxisOutByte[5]<<8|AxisOutByte[4]);
                    USART1printf("\t\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",
                                (short)(AxisOutByte[1]<<8|AxisOutByte[0]),
                                (short)(AxisOutByte[3]<<8|AxisOutByte[2]),
                                (short)(AxisOutByte[5]<<8|AxisOutByte[4]));

                    unsigned char DVBuf[8] = { 0x03,
                                                AxisOutByte[0], AxisOutByte[1],
                                                AxisOutByte[2], AxisOutByte[3],
                                                AxisOutByte[4], AxisOutByte[5],
                                                ...
                    };
                }
            }
        }
    }
}
```

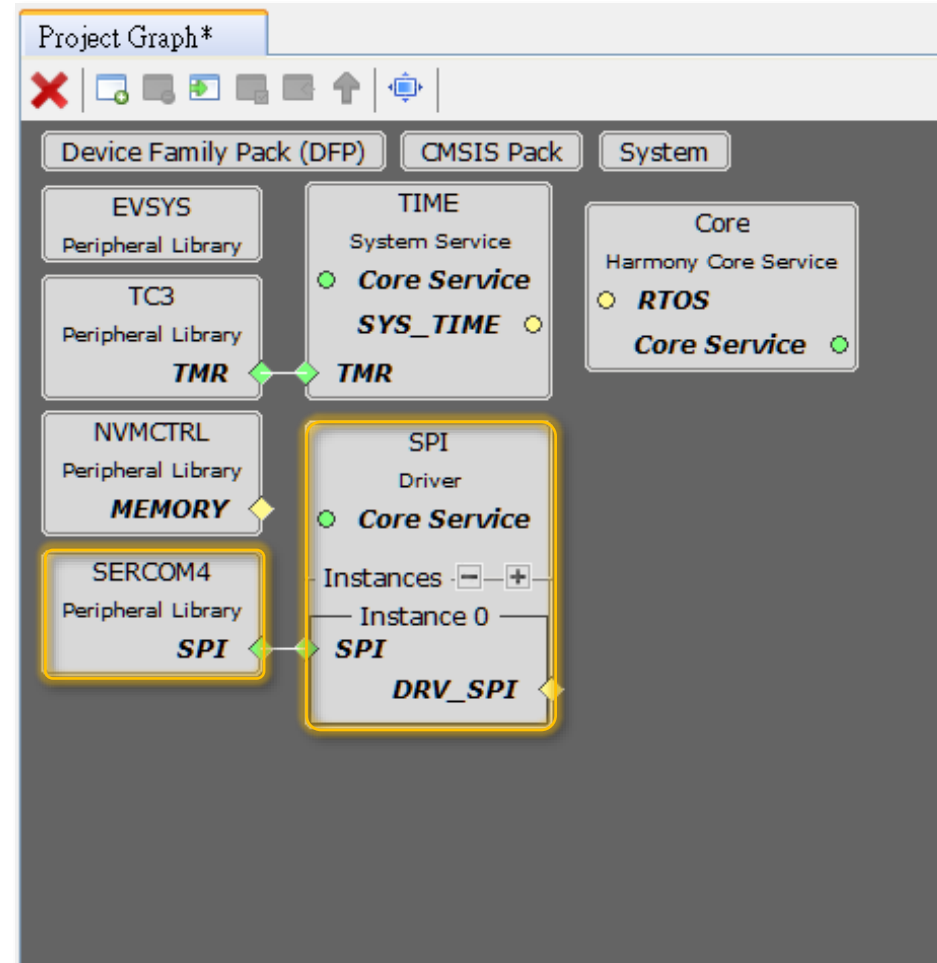
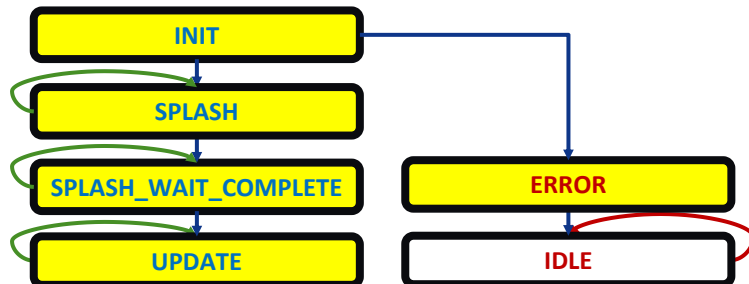
Include app_drv_spi1.h

Collect G-Sensor Gravity data X, Y, Z

app_drv_i2c2.c

✂ Lab16 Single-Client SPI OLED Display

- Compiler and programming your project to EVB.
- The OLED will output the data that we collect from Driver.



✖ Lab16 Single-Client SPI OLED Display

- Try comment out UART debug message in “app_drv_i2c1.c” to speed-up OLED display.

```
#include "app_drv_i2c1.h"
#include "app_drv_usart1.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            if(I2C1Complete_9800 == true)
            {
                I2C1Complete_9800 = false;

                LCM_I2c1Temp = (float)((I2C1ReadBuf_9800[0]<<4)|
                                         (I2C1ReadBuf_9800[1]>>4))/16.0f;
                // USART1printf("Temp = %2.1f\r",
                //                 (float)((I2C1ReadBuf_9800[0]<<4)|(I2C1ReadBuf_9800[1]>>4))/16.0f);
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_24AA;
            }
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA:
            if(I2C1Complete_24AA == true)
            {
                I2C1Complete_24AA = false;

                LCM_I2c1EERPOM = I2C1ReadBuf_24AA[0];
                // USART1printf("\t\tEEPROM = 0x%X\r", I2C1ReadBuf_24AA[0]);
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            }
            ...
    }
}
```

Comment out USART1printf

Comment out

Comment out

app_drv_i2c1.c

⚙️ Lab16 Single-Client SPI OLED Display

- Try comment out UART debug message in “`app_drv_i2c2.c`” to speed-up OLED display.

```
#include "app_drv_i2c2.h"
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        ...
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];
                if(++ AxisOutRegIdx >= 6)
                {
                    LCM_I2c2Accel[0] = (short)(AxisOutByte[1]<<8 | AxisOutByte[0]);
                    LCM_I2c2Accel[1] = (short)(AxisOutByte[3]<<8 | AxisOutByte[2]);
                    LCM_I2c2Accel[2] = (short)(AxisOutByte[5]<<8 | AxisOutByte[4]);
                    // USART1printf("\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",
                    // (short)(AxisOutByte[1]<<8 | AxisOutByte[0]),
                    // (short)(AxisOutByte[3]<<8 | AxisOutByte[2]),
                    // (short)(AxisOutByte[5]<<8 | AxisOutByte[4]));

                    // unsigned char DVBuf[8] = { 0x03,
                    // AxisOutByte[0], AxisOutByte[1],
                    // AxisOutByte[2], AxisOutByte[3],
                    // AxisOutByte[4], AxisOutByte[5],
                    // USART2write( DVBuf, 8 );
                    ...
                }
            }
        }
    }
}
```

Comment out
USART1printf and USART2write

Comment out

Comment out

`app_drv_i2c2.c`

SPI Driver – Wi-Fi WINC Driver Library

Harmony Driver

SPI Driver – Wi-Fi WINC Driver Library

• Introduction

The MPLAB Harmony WINC Driver provides a high-level interface to the ATWINC1500 and ATWINC3400 Wi-Fi network controller modules. The WINC Driver Library, in conjunction with the ATWINC1500 or ATWINC3400 module, allows an application to:

- Join an existing 802.11 b/g/n 2.4GHz Wi-Fi Infrastructure network.
- Create an 802.11 Wi-Fi Ad Hoc or Soft AP network.
- Scan for other Wi-Fi devices in an area
- Supports BLE 4.0 provisioning and profiles in central and peripheral mode (ATWINC3400 only)

• Abstraction Model

This library provides an abstraction of the ATWINC1500 and ATWINC3400 network controllers. The library operates in two forms, a basic driver intended to work solely with an application and a MAC driver intended to work in concert with the Harmony TCP/IP stack and the application. In either form, the driver provides function to perform the following:

- Manage WINC system time
- Manage Wi-Fi associations
- Perform Wi-Fi BSS search operations
- Create and manage Soft AP functionality
- Connect to an existing 802.11 b/g/n infrastructure network
- Control device power save operation
- Manage security profiles
- Configure and control WINC network controllers

SPI Driver – Wi-Fi WINC Driver Library

- **ASIC Socket vs Ethernet Mode**

The WINC modules are capable of below operating modes:

Socket mode :

This mode allows an application running on the host MCU to utilize the **TCP/IP stack within the firmware of the WINC device**, as such the **host MCU does not need to run its own TCP/IP stack**.

Ethernet (bypass) mode :

This mode to disables the TCP/IP stack within the firmware of the WINC device, instead the host MCU can only send and receive Ethernet frames. This mode allows the **host MCU to operate its own TCP/IP stack** treating the WINC device as a simple Ethernet controller.

Note: The WINC device may need separate firmware loaded to support a different mode.

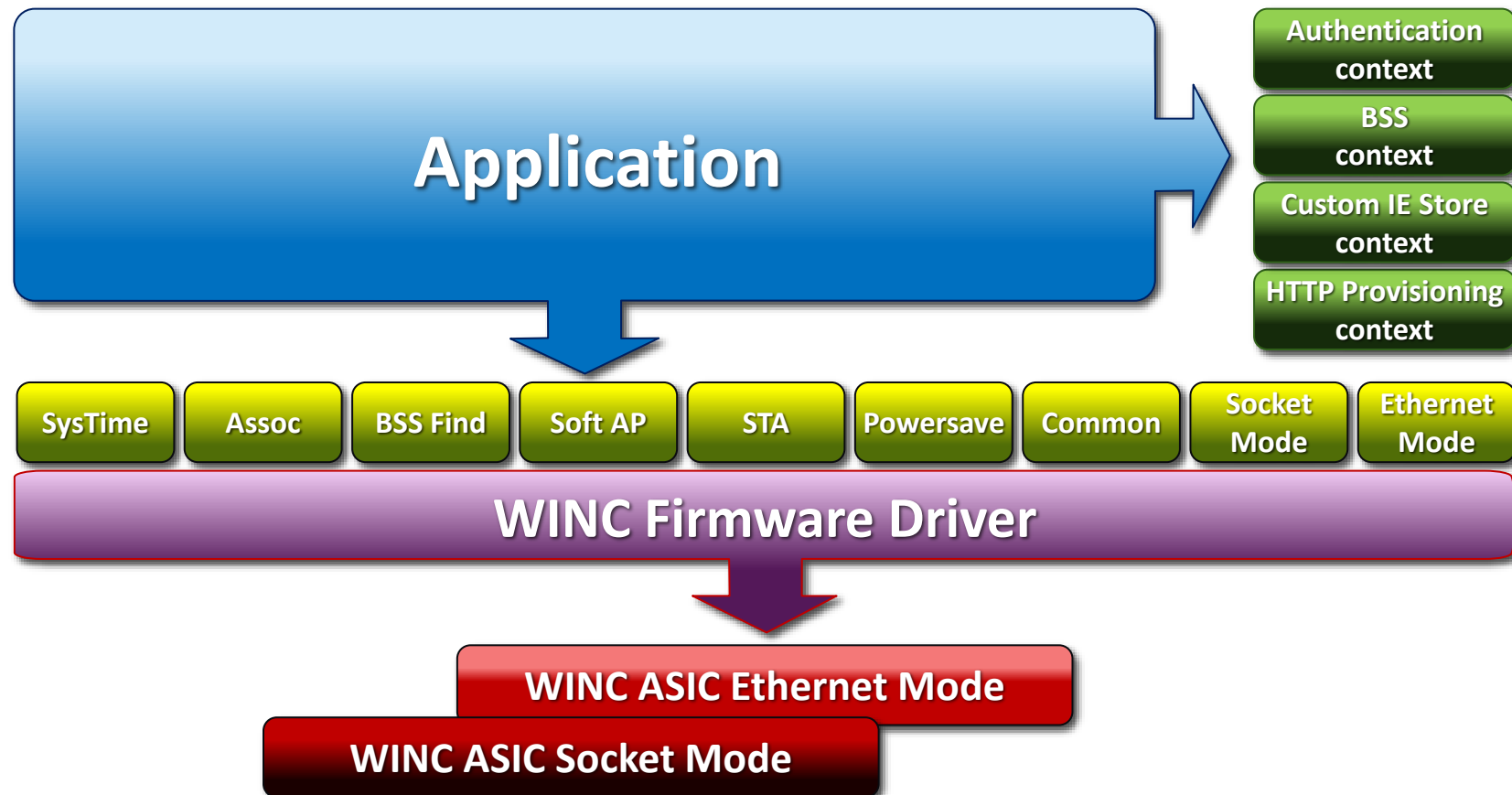
- **Contexts**

The driver uses several contexts to manage complex groupings of information such as authentication, BSS and HTTP provisioning settings. A context can be created external to the WINC Driver before being passed into the driver as part of an operation. The WINC Driver may also pass back to the application a context if it is appropriate for the information being passed. The WINC Driver provides functions to manipulate the contexts and ensure they are valid before being accepted by the driver.

SPI Driver – Wi-Fi WINC Driver Library

- **Basic WINC Driver**

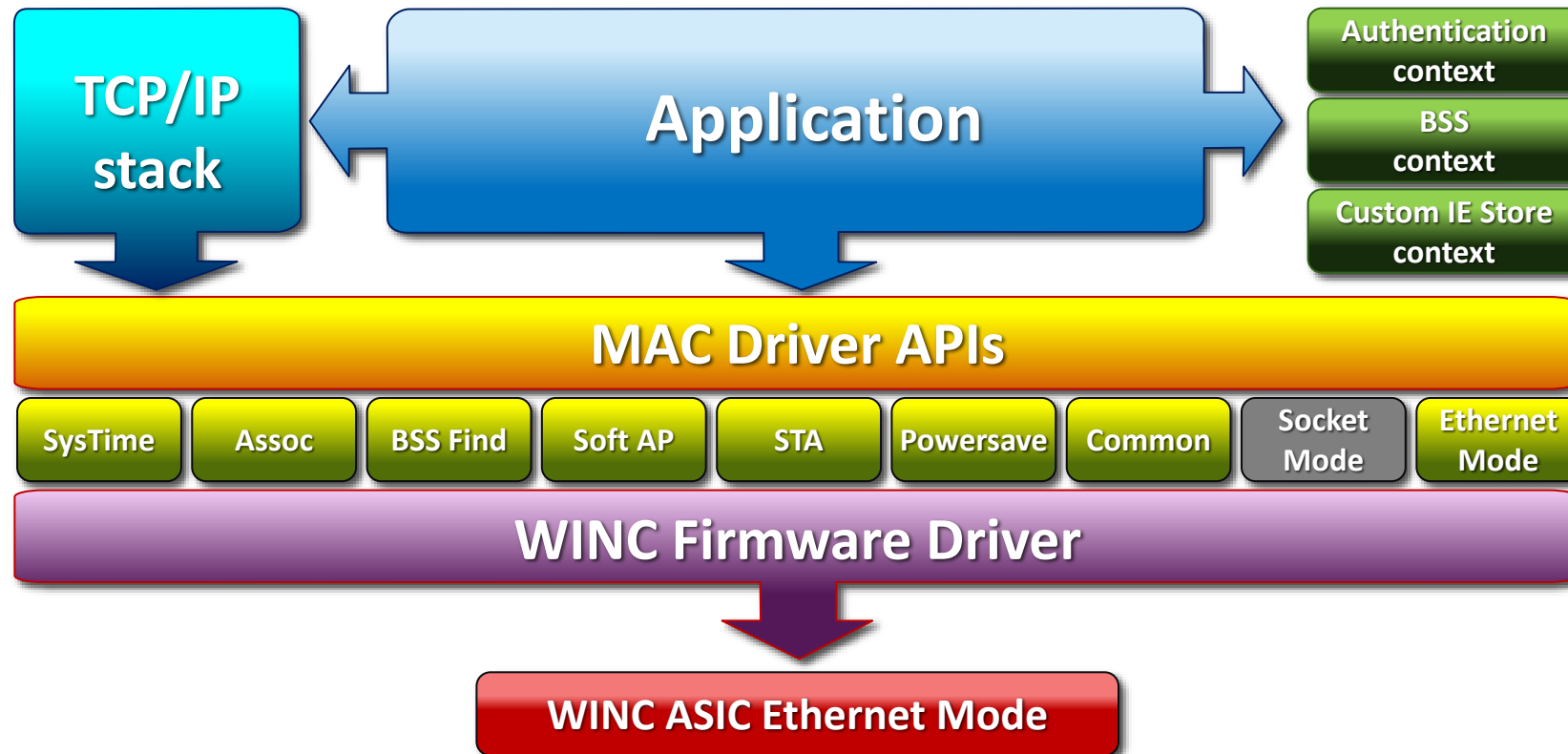
The Basic WINC Driver is utilized when the host MCU application is **the primary client** of the WINC Driver. In this arrangement the host MCU application is either reliant on the TCP/IP stack within the WINC device firmware or is only interested in sending and receiving basic Ethernet frames.



SPI Driver – Wi-Fi WINC Driver Library

- **MAC WINC Driver**

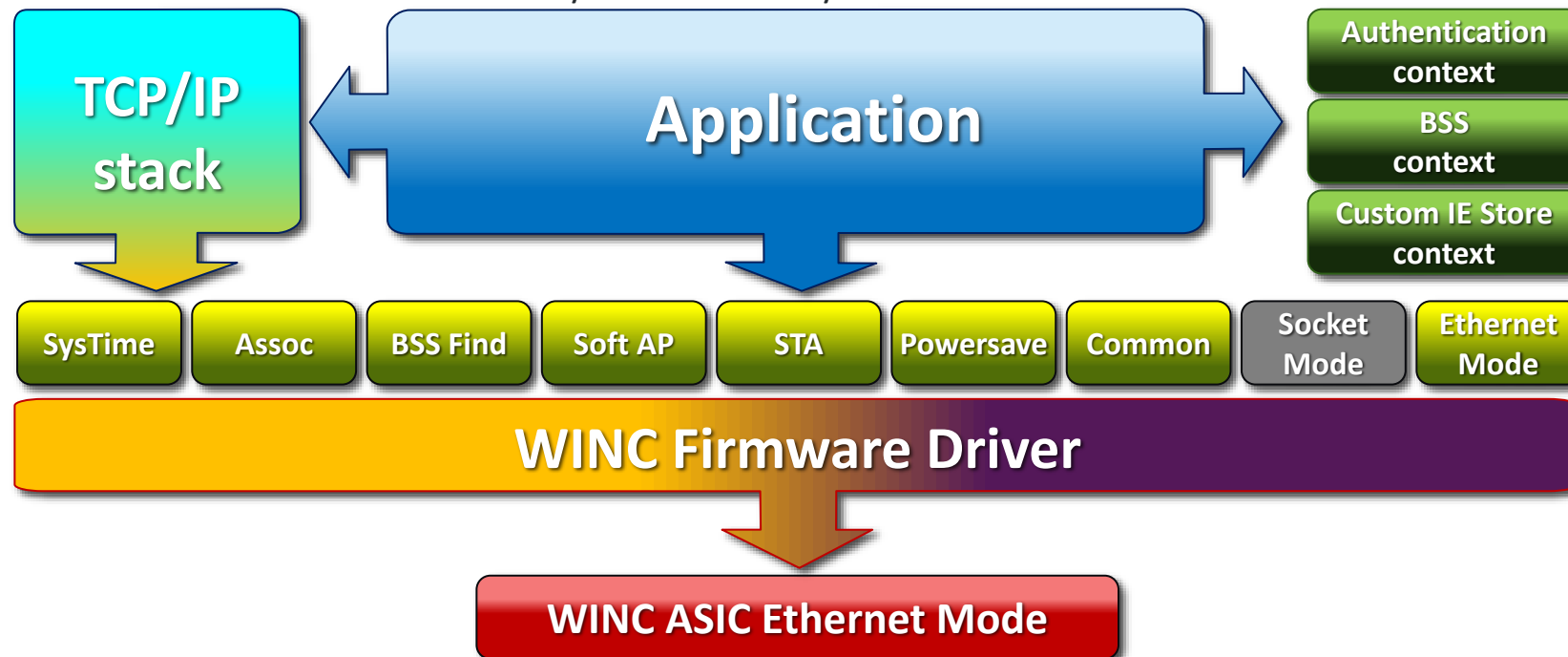
The MAC WINC Driver is utilized when the host MCU application is the **combined with the Harmony TCP/IP stack**. The Harmony TCP/IP stack is responsible for operating the WINC device as an **Ethernet controller** while the application is responsible for managing the Wi-Fi specific functionality such as locating and connecting to 802.11 b/g/n BSSs.



SPI Driver – Wi-Fi WINC Driver Library

- **MAC WINC Driver** (Changes in WINC1500 19.7.3 and WINC3400 1.4.2)

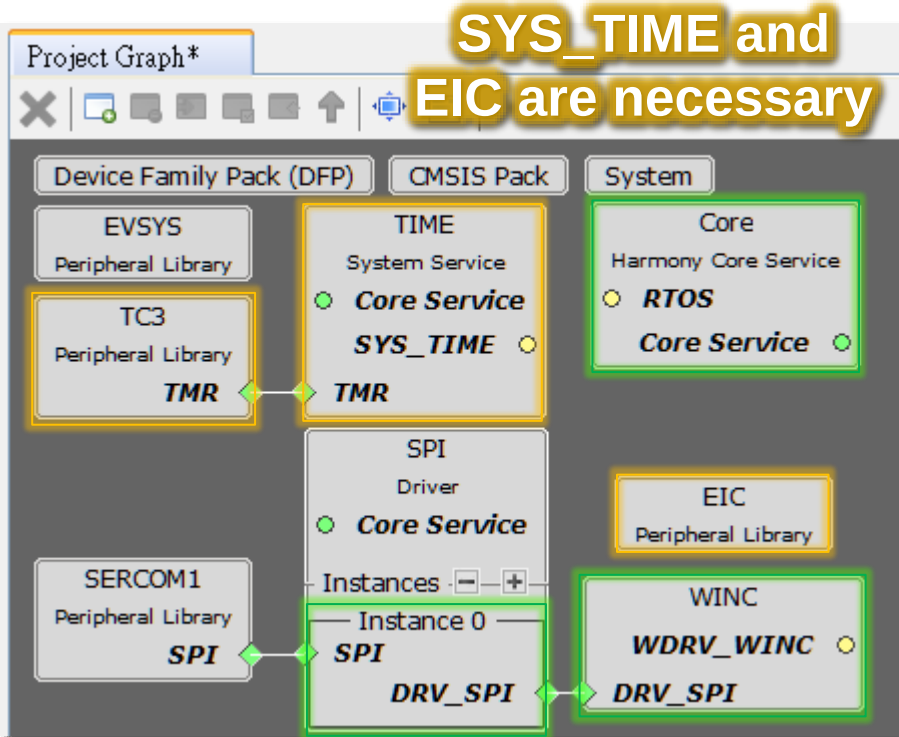
The major change in the later WINC architecture is the **removal of the 'stack driver'**. Previously there was an extra layer over the WINC driver API when the driver was used in bypass mode with the Harmony TCP/IP stack. The application had limited control of the driver through the WDRV_WINC_MAC* APIs. The APIs had to be wrapped in a WDRV_WINC_MAC version and there was no direct control of connection/disconnection. The driver now has two interfaces which can be opened, first the normal interface used in non-bypass mode (called the control interface) and a second interface used by the TCP/IP stack (the MAC interface). In a way, instead of being the MAC interface on top of the control interface as before they are now side by side.



For the applications there is no change to how things work for non-bypass mode. However, now that same application control can be used in bypass mode with the TCP/IP stack, **the only difference being that the WINC socket API is replaced with the Harmony stack API** (and certain WINC APIs become unavailable as they are based on non-bypass firmware).

The example wifi_winc_sta_bypass has been updated to reflect this change.

SPI Driver – Wi-Fi WINC Driver Library



Enable DMA of SPI Driver



Configuration Options* **Help**

Enable DMA of SPI Driver

[-] SPI

- PLIB Used: SERCOM1
- Number of Clients: 1
- Transfer Queue Size: 4
- Use DMA for Transmit and Receive? ☒
- DMA Channel For Transmit: 2
- DMA Channel For Receive: 3

Create an Application for Driver State Machine

Configuration Options* **Help**

[-] [+]

[-] Core

[-] Generate Harmony Application Files ☒

[-] Application Configuration

Number of Applications

[-] Application 0 Configuration

Application Name

**** Application name must be valid C-Language

Configuration Options* **Help**

[-] [+]

[-] WINC

- WiFi Device: WINC1500
- Driver Log Level: None
- SPI Driver: drv_spi_1
- Interrupt Source: EIC
- EIC Channel 4: ☒
- ***EIC channel must be configured in
- Firmware Version: 19.6.1
- Driver Mode: Socket Mode

*** Please must select the correct version of your WIN1500 Module Firmware**

Assign External Interrupt WIN1500 Module Firmware WINC Driver Mode

SPI Driver – Wi-Fi WINC Driver Library

API functions of the Wi-Fi WINC Driver Library.

- WINC wireless driver System Interface implementation (wdrv_winc.c)

WDRV_WINC_Initialize	System interface initialization of the WINC driver.
WDRV_WINC_Deinitialize	WINC driver deinitialization function.
WDRV_WINC_Reinitialize	WINC driver reinitialization function.
WDRV_WINC_Status	Provides the current status of the WINC driver module.
WDRV_WINC_Tasks	Maintains the WINC drivers state machine.
WDRV_WINC_ISR	WINC interrupt handler.
WDRV_WINC_DebugRegisterCallback	Register callback for debug serial stream.
WDRV_WINC_Open	Opens an instance of the WINC driver.
WDRV_WINC_Close	Closes an instance of the WINC driver.
WDRV_WINC_MulticastMACFilterAdd	Adds an Ethernet address to the multicast filter.
WDRV_WINC_MulticastMACFilterRemove	Removes an Ethernet address from the multicast filter.
WDRV_WINC_EthernetAddressGet	Returns the current Ethernet address assigned to the WINC.
WDRV_WINC_OperatingModeGet	Returns the current operating mode of the WINC.
WDRV_WINC_UserHandleSet	Associates a caller supplied handle with the driver instance.
WDRV_WINC_UserHandleGet	Returns the current user supplied handle associated with the driver.
WDRV_WINC_InfoDriverVersionGet	Returns the drivers version information.
WDRV_WINC_InfoDeviceFirmwareVersionGet	Returns the WINC firmware version information.
WDRV_WINC_InfoDeviceIDGet	Returns the device ID.
WDRV_WINC_InfoDeviceMACAddressGet	Retrieves the MAC address of the WINC.
WDRV_WINC_InfoDeviceNameSet	Sets the WiFi Direct Device Name.
WDRV_WINC_GainTableIndexSet	Sets the gain table index.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver Socket Mode implementation (wdrv_winc_socket.c)**

WDRV_WINC_IPAddressSet	Configures an IPv4 address.
WDRV_WINC_IPAddressGet	Returns the current IPv4 address.
WDRV_WINC_IPDNSServerAddressSet	Configures the DNS server address.
WDRV_WINC_IPDefaultGatewaySet	Configures the default gateway used by the WINC.
WDRV_WINC_IPUseDHCPSet	Enables DHCP.
WDRV_WINC_IPUseDHCPGet	Determine if DHCP is to be used.
WDRV_WINC_IPDHCPServerConfigure	Configures the Soft-AP DHCP server.
WDRV_WINC_IPLinkActive	Indicates state of IP link.
WDRV_WINC_ICMPEchoRequest	Sends an ICMP echo request.
WDRV_WINC_OTAUUpdateFromURL	Perform OTA from URL provided.
WDRV_WINC_OTAUUpdateAbort	Abort an OTA update.
WDRV_WINC_SwitchActiveFirmwareImage	Switch active firmware image to inactive partition.
WDRV_WINC_SocketRegisterEventCallback	Register an event callback for socket events.
WDRV_WINC_SocketRegisterResolverCallback	Register an event callback for DNS resolver events.
WDRV_WINC_SwitchActiveFirmwareImage	Switch active firmware image to inactive partition.
WDRV_WINC_SocketRegisterEventCallback	Register an event callback for socket events.
WDRV_WINC_SocketRegisterResolverCallback	Register an event callback for DNS resolver events.

- **WINC wireless driver STA implementation (wdrv_winc_sta.c)**

WDRV_WINC_BSSConnect	Connects to a BSS in infrastructure station mode.
WDRV_WINC_BSSReconnect	Reconnects to a BSS using stored credentials.
WDRV_WINC_BSSDisconnect	Disconnects from a BSS.
WDRV_WINC_BSSRoamingConfigure	Configures BSS roaming support.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver BSS Scanning implementation (wdrv_winc_bssfind.c)**

WDRV_WINC_BSSFindFirst	Request a BSS scan is performed by the WINC.
WDRV_WINC_BSSFindNext	Request the next scan results be provided.
WDRV_WINC_BSSFindReset	Requests the information structure of the current BSS scan result.
WDRV_WINC_BSSFindGetInfo	Adds an Ethernet address to the multicast filter.
WDRV_WINC_BSSFindSetScanParameters	Configures the scan operation.
WDRV_WINC_BSSFindSetRSSIThreshold	Configures the scan RSSI threshold.
WDRV_WINC_BSSFindSetEnabledChannels	Configures the channels which can be scanned.
WDRV_WINC_BSSFindGetNumBSSResults	Returns the number of BSS scan results found.
WDRV_WINC_BSSFindInProgress	Indicates if a BSS scan is in progress.

- **WINC wireless driver BSS Context implementation (wdrv_winc_bssctx.c)**

WDRV_WINC_BSSCtxIsValid	Tests if a BSS context is valid.
WDRV_WINC_BSSCtxSetDefaults	Configures a BSS context into a default legal state.
WDRV_WINC_BSSCtxSetSSID	Configures the SSID of the BSS context.
WDRV_WINC_BSSCtxSetBSSID	Configures the BSSID of the BSS context.
WDRV_WINC_BSSCtxSetChannel	Configures the channel of the BSS context.
WDRV_WINC_BSSCtxSetSSIDVisibility	Configures the visibility of the BSS context.

- **WINC wireless driver Association implementation (wdrv_winc_assoc.c)**

WDRV_WINC_AssociationSSIDGet	Retrieve the current association SSID.
WDRV_WINC_AssociationPeerAddressGet	Retrieve the current association peer device network address.
WDRV_WINC_AssociationAuthTypeGet	Retrieve the current association authentication type.
WDRV_WINC_AssociationRSSIGet	Retrieve the current association RSSI.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver Authentication Context implementation (wdrv_winc_authctx.c)**

WDRV_WINC_AuthCtxIsValid	Tests if an authentication context is valid.
WDRV_WINC_AuthCtxSetDefaults	Configures an authentication context into a default state.
WDRV_WINC_AuthCtxSetOpen	Configure an authentication context for Open authentication.
WDRV_WINC_AuthCtxSetWEP	Configure an authentication context for WEP authentication.
WDRV_WINC_AuthCtxSetWPA	Configure an authentication context for WPA-PSK authentication.
WDRV_WINC_AuthCtxSetWPAEnterpriseMSCHAPv2	Configure an authentication context for WPA(2)-Enterprise authentication using MS-CHAPv2.
WDRV_WINC_AuthCtxSetWPAEnterpriseTLS	Configure an authentication context for WPA(2)-Enterprise authentication using TLS.

- **WINC wireless driver Host File Download implementation (wdrv_winc_host_file.c)**

WDRV_WINC_HostFileDownload	Request the WINC downloads a file to its internal memory.
WDRV_WINC_HostFileRead	Requests to read the data from a downloaded file.
WDRV_WINC_HostFileErase	Requests to erase a downloaded file.
WDRV_WINC_HostFileFindByID	Maps a file ID to a driver file handle.
WDRV_WINC_HostFileGetSize	Retrieves the file size from a file handle.

- **WINC wireless driver HTTP Provisioning Context implementation (wdrv_winc_httpprovctx.c)**

WDRV_WINC_HTTPProvCtxSetDefaults	Initialize provisioning store.
WDRV_WINC_HTTPProvCtxSetDomainName	Define the DNS domain name for the provisioning server.
WDRV_WINC_HTTPProvCtxSetURLWildcard	Define the use of a wildcard URL check.
WDRV_WINC_HTTPProvCtxSetConnectCallback	Defines the connection information callback.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver NVM Access implementation (wdrv_winc_nvm.c)**

WDRV_WINC_NVM_EraseSector	Erase a sector within an NVM region.
WDRV_WINC_NVM_Write	Writes data to an NVM region.
WDRV_WINC_NVM_Read	Reads data from an NVM region.
- **WINC wireless driver Power Save implementation (wdrv_winc_powersave.c)**

WDRV_WINC_PowerSave_SetMode	Sets the power save mode.
WDRV_WINC_PowerSave_GetMode	Returns the current power save mode.
WDRV_WINC_PowerSave_SetBeaconInterval	Configures the beacon listening interval.
WDRV_WINC_PowerSave_ManualSleep	Manual sleep power save operation.
- **WINC wireless driver Soft-AP implementation (wdrv_winc_softap.c)**

WDRV_WINC_AP_Start	Starts an instance of Soft-AP.
WDRV_WINC_AP_Stop	Stops an instance of Soft-AP.
WDRV_WINC_AP_SetCustomIE	Configures the custom IE.
- **WINC wireless driver TLS/SSL implementation (wdrv_winc_ssl.c)**

WDRV_WINC_SSLCTX_Default	Initialize an SSL cipher suite context.
WDRV_WINC_SSLCTX_CipherSuitesSet	Set the cipher suites enabled in a context.
WDRV_WINC_SSLCTX_CipherSuitesGet	Returns the enabled cipher suites from a context.
WDRV_WINC_SSLActiveCipherSuitesSet	Set the active cipher suites.
WDRV_WINC_SSLECCHandShakeRsp	Handshake Response for ECC cipher suites.
WDRV_WINC_SSLRetrieveCert	Retrieve the certificate to be verified from the WINC.
WDRV_WINC_SSLRetrieveHash	Retrieve the certificate hash.
WDRV_WINC_SSLStopRetrieveCert	Stop processing the certificate.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver Custom IE implementation (wdrv_winc_custie.c)**
 - WDRV_WINC_CustIESToreCtxSetStorage Initialize the custom IE store.
 - WDRV_WINC_CustIESToreCtxAddIE Add data to the custom IE store.
 - WDRV_WINC_CustIESToreCtxRemoveIE Removes data from the custom IE store.
- **WINC wireless driver SysTime implementation (wdrv_winc_systime.c)**
 - WDRV_WINC_LocalTimeToUTC Convert local system time to UTC time.
 - WDRV_WINC_UTCToLocalTime Convert UTC time to local system time.
 - WDRV_WINC_SystemTimeSNTPClientEnable Enable the SNTP client to set system time.
 - WDRV_WINC_SystemTimeSNTPClientDisable Disable the SNTP client.
 - WDRV_WINC_SystemTimeSetCurrent Sets the current system time on the WINC.
 - WDRV_WINC_SystemTimeGetCurrent Requests the current system time from the WINC.
- **WINC wireless driver WPS implementation (wdrv_winc_wps.c)**
 - WDRV_WINC_WPSEnrolleeDiscoveryStartPIN Start WPS discovery (PIN).
 - WDRV_WINC_WPSEnrolleeDiscoveryStartPBC Start WPS discovery (Push Button).
 - WDRV_WINC_WPSEnrolleeDiscoveryStop Stop WPS discovery.
- **WINC wireless driver OS Abstraction Layer implementation (wdrv_winc_oscal.c)**
 - WDRV_MSDelay Performs a delay specified in milli-seconds.
- **WINC Driver BLE Implementation (wdrv_winc_ble.c)**
 - WDRV_WINC_BLEIsStarted Determines if BLE has been started.
 - WDRV_WINC_BLEStart Enables the BLE interface.
 - WDRV_WINC_BLEStop Disables the BLE interface.

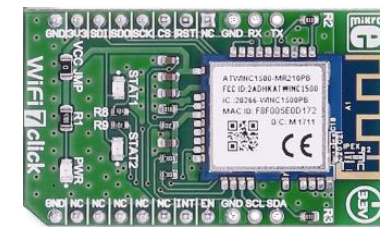
Lab 17

AP scan application of SPI Wi-Fi Module

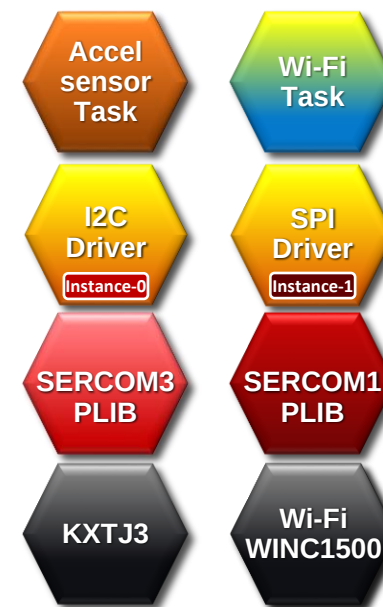
SPI Driver – Wi-Fi WINC Driver Library

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Try implement a **Wi-Fi Access Point (AP) scan** application.
- Toggle **S3** to **SERCOM3** to connect **G-Sensor** on **I2C Driver Instance 0**.
- **Disconnect** **SERCOM1** from **I2C Driver Instance 1** and then **delete(-)** unused **Instance 1**.
- Add(+) **SPI Driver Instance1** to be consumed by **Wi-Fi WINC Driver**.
- **Reconnect** **SERCOM1** to **SPI Driver Instance 1** to communicate with an external “**Wi-Fi 7 Click EVB**” that plug on **mikroBUS** socket.
- The **EXINT4** need setup for interrupt handling and two extra GPIOs, custom name to “**WDRV_WINC_RESETN**” and “**WDRV_WINC_CHIP_EN**” for Reset and Enable control.



Wi-Fi 7 click (WINC1500)	Arduino socket	SAMD21 Pin Configure (Custom Name)	SAMD21 Pin MUX
SDI	D11/MOSI	PA16	SERCOM1-PAD0
SCK	D13/SCK	PA17	SERCOM1-PAD1
CS	D10/SS	PA18	SERCOM1-PAD2
SDO	D12/MISO	PA19	SERCOM1-PAD3
INT	A3/D17	PA04	EXINT4, Falling Edge Detect
RST	D8	PA06 (WDRV_WINC_RESETN)	GPIO output, Latch High
EN	D6/PWM	PA20 (WDRV_WINC_CHIP_EN)	GPIO output, Latch High

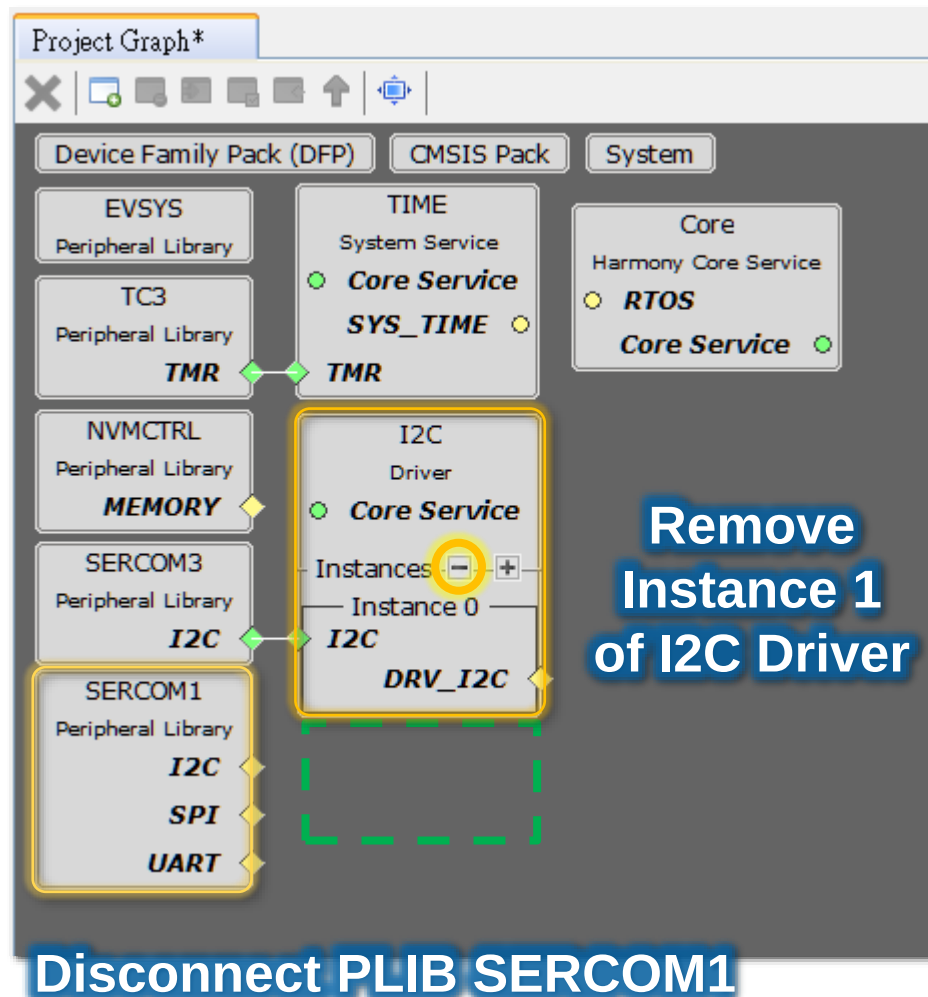


Arduino UNO Socket	
15	D8
16	D9/PWM
17	D10/PWM/SS
18	D11/PWM/MOSI
19	D12/MISO SERCOM1
20	D13/SCK Wi-Fi 7 Click
21	GND
22	AREF
23	D18/SDA SERCOM3
24	D19/SCL G-Sensor

I2C Auto Probe must implement

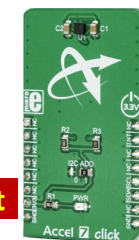
⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Disconnect **SERCOM1** from **I2C driver Instance1** and delete **Instance 1**.



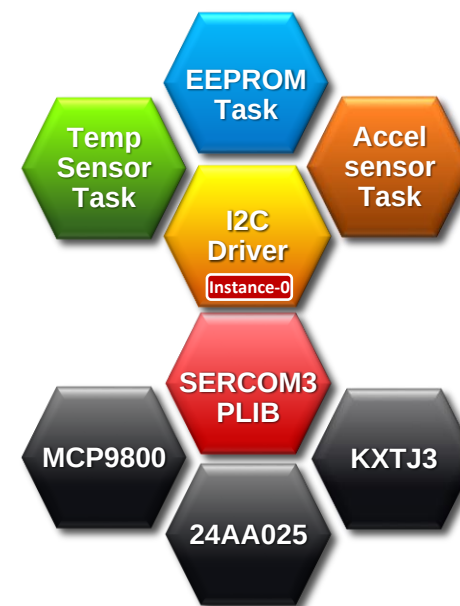
Wire
Accel 7 Click
to SERCOM3

I2C Auto Probe must implement



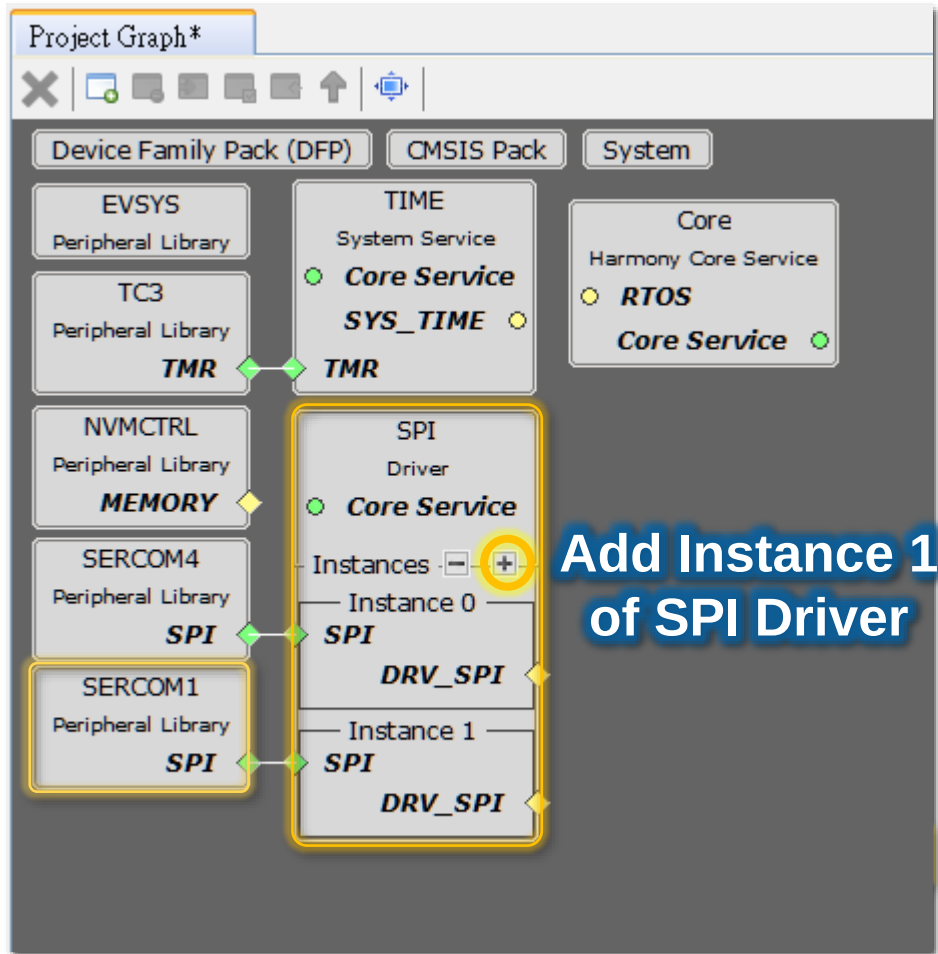
Arduino UNO Socket

15	D8
16	D9/PWM
17	D10/PWM/SS
18	D11/PWM/MOSI
19	D12/MISO
20	D13/SCK
21	GND
22	AREF
23	D18/SDA
24	D19/SCL SERCOM3

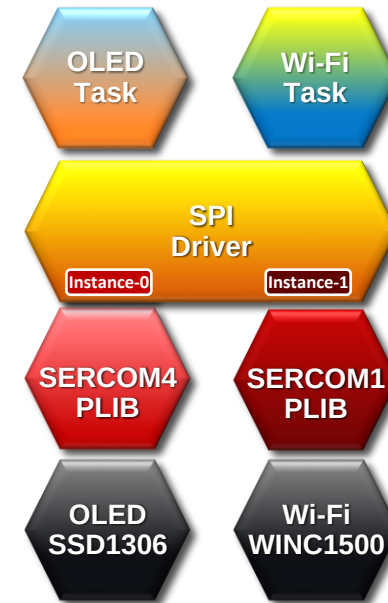


⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Add **SPI Driver Instance 1** and attach **SERCOM1-SPI** as hardware for Wi-Fi module connection.

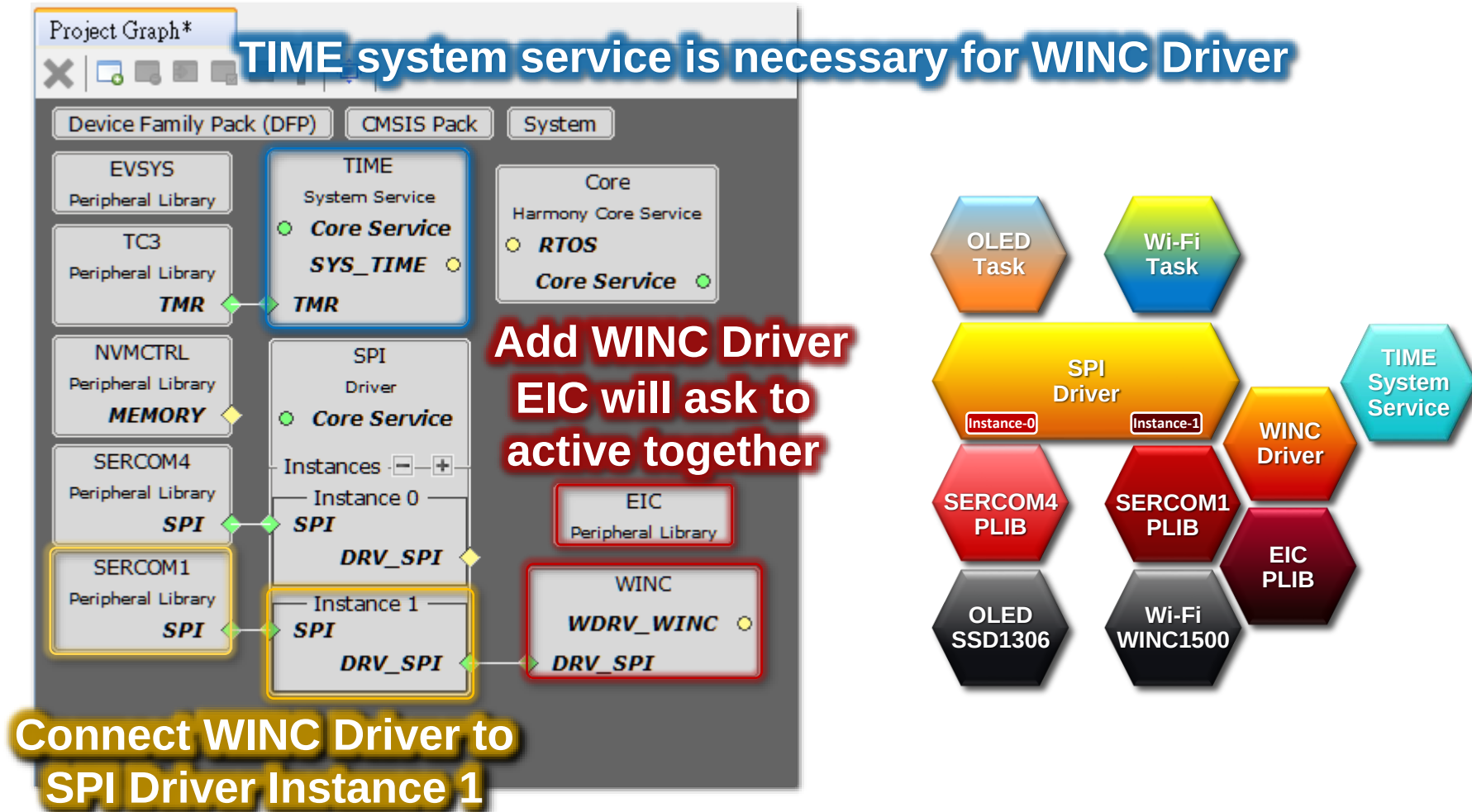


Attach PLIB SERCOM1



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Add **Wi-Fi WINC Driver** and **EIC (External Interrupt)** will ask to active together.
- The **TIME system service** is necessary for **WINC Driver**.



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Enable DMA of **SPI Driver** and configure pads of **PLIB SERCOM1**.

The image displays two configuration windows from the Microchip Studio IDE, illustrating the setup for the SPI driver and SERCOM1 peripheral.

Top Window: SERCOM1 Configuration Options*

- Select SERCOM Operation Mode:** SPI Master
- Enable Interrupts ?** ☒
- Enable operation in Standby mode** ☐
- SPI Data Out Pad:** DO on PAD[0], SCK on PAD[1] and SS on PAD[2]
- SPI Data In Pad Selection:** SERCOM PAD[3] (highlighted with a green box and labeled **DI on PAD3**)
- SPI Data Order:** MSB is transferred first
- SPI Speed in Hz:** 1,000,000
- 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
- Enable SPI Master Hardware Slave Select** ☒ (highlighted with a green box and labeled **Enable Hardware Slave Select**)
- SPI Receiver Enable** ☒

Bottom Window: SPI Configuration Options*

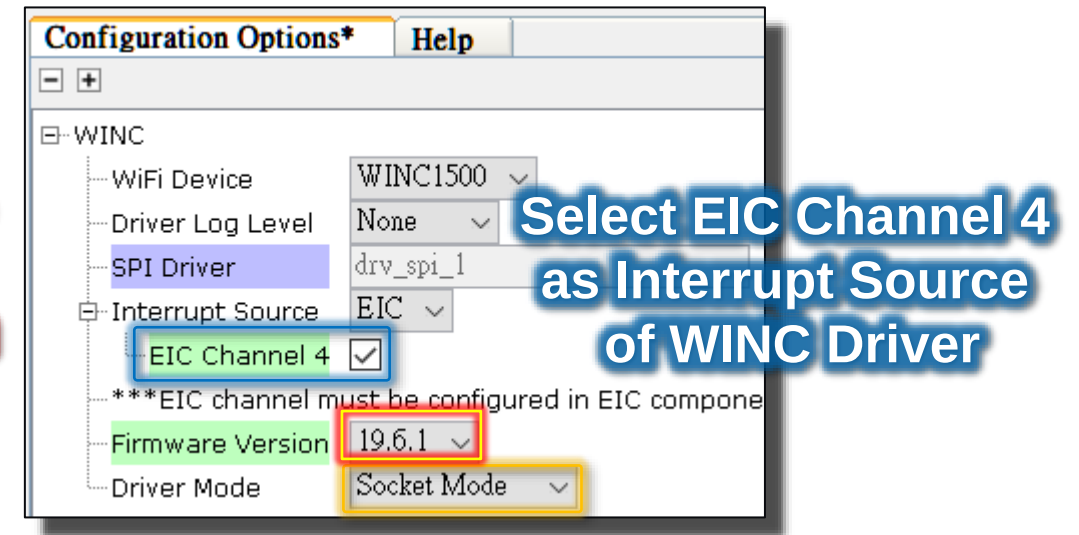
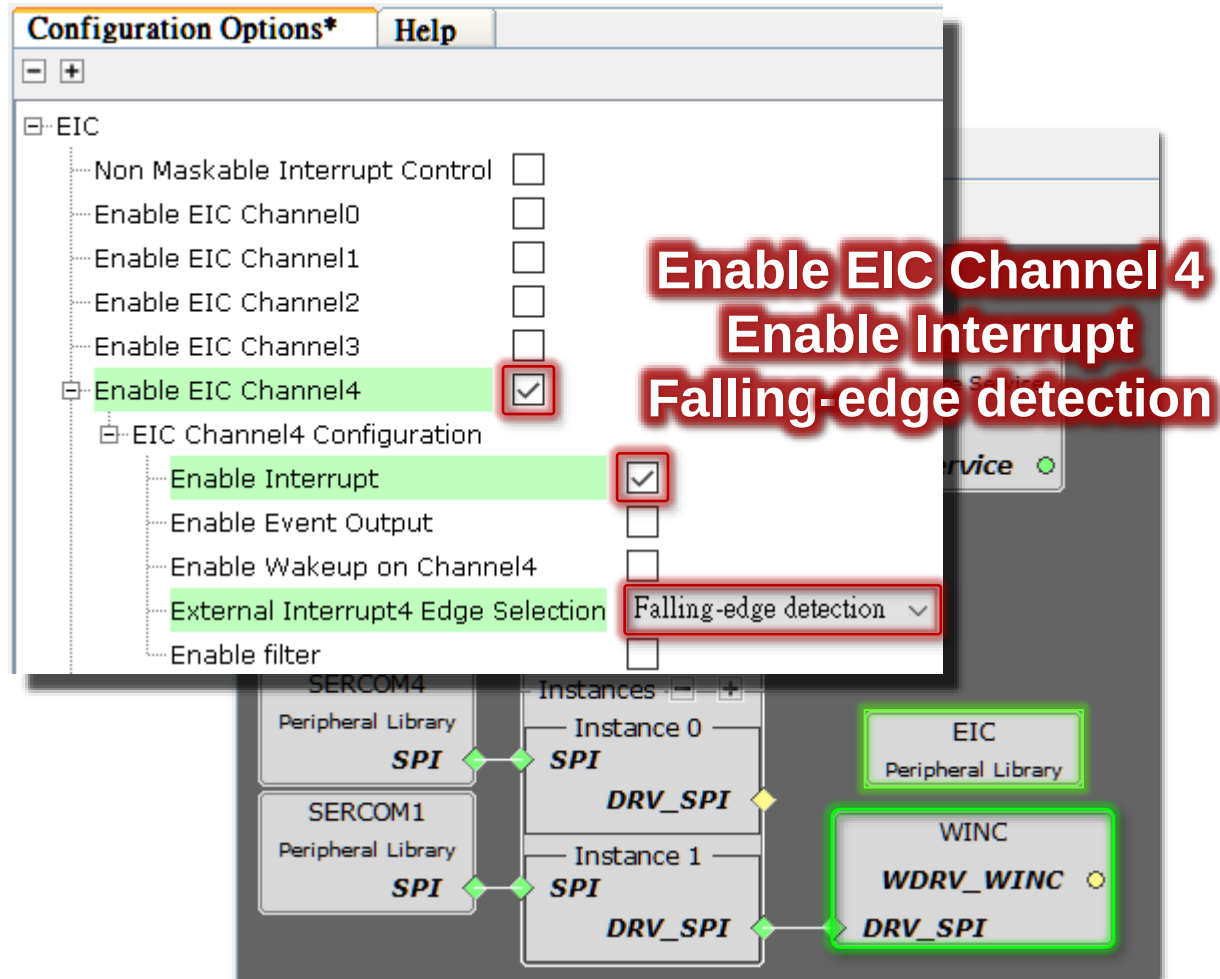
- PLIB Used:** SERCOM1
- Number of Clients:** 1
- Transfer Queue Size:** 4
- Use DMA for Transmit and Receive?** ☒ (highlighted with a green box and labeled **Enable DMA in SPI Driver**)
- DMA Channel For Transmit:** 2
- DMA Channel For Receive:** 3

Peripheral Library Diagram:

The diagram shows the connection between the SERCOM1 peripheral library and the SPI driver. SERCOM1 is connected to Instance 0 and Instance 1 of the SPI driver. Instance 0 is connected to Instance 1 of the SPI driver. Instance 1 is connected to the WDRV_WINC peripheral library. The SPI driver is also connected to the WDRV_WINC peripheral library.

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Enable interrupt of **EIC Channel4** and **Falling Edge detection**.
- Configure interrupt source of **WINC Driver** to use **EIC Channel4**.



!!! Must select to Correct Firmware of the WIN1500 Module you to use

The default WINC Driver mode has set to Socket Mode

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Configure **SERCOM1-SPI** pads, **EXINT4** and **custom name GPIOs**.

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
9	PA04		EIC_EXTINT4	Digital	High Impedance	n/a	☐	☐	NORMAL
11	PA06	WDRV_WINC_RESETN	GPIO	Digital	Out	High	☐	☐	NORMAL
14	PA09	LED2	GPIO	Digital	Out	High	☐	☐	NORMAL
25	PA16		SERCOM1_PAD0	Digital	High Impedance	n/a	☐	☐	NORMAL
26	PA17		SERCOM1_PAD1	Digital	High Impedance	n/a	☐	☐	NORMAL
27	PA18		SERCOM1_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
28	PA19		SERCOM1_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL
29	PA20	WDRV_WINC_CHIP_EN	GPIO	Digital	Out	High	☐	☐	NORMAL

Custom PA06 as **WDRV_WINC_RESETN**
 Custom PA20 as **WDRV_WINC_CHIP_EN**

Wi-Fi 7 click (WINC1500)	Arduino socket	SAMD21 Pin Configure (Custom Name)	SAMD21 Pin MUX
SDI	D11/MOSI	PA16	SERCOM1-PAD0
SCK	D13/SCK	PA17	SERCOM1-PAD1
CS	D10/SS	PA18	SERCOM1-PAD2
SDO	D12/MISO	PA19	SERCOM1-PAD3
INT	A3/D17	PA04	EXINT4, Falling Edge Detect
RST	D8	PA06 (WDRV_WINC_RESETN)	GPIO output, Latch High
EN	D6/PWM	PA20 (WDRV_WINC_CHIP_EN)	GPIO output, Latch High
		PA09 (LED2)	GPIO output, Latch High

Already done in previous I2C Lab

Already done in previous I2C Lab

Additional CS pin for SERCOM-SPI

Additional MISO pin for SERCOM-SPI

External Interrupt on PA04

Reset pin GPIO control

Modify from LED1

For instead of LED1

✂ Lab17 AP scan application of SPI Wi-Fi Module

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**6**” and given the Application Name to to “**app_drv_winc**”.

The image shows two windows from the Harmony IDE. The left window, titled "Project Graph*", displays a block diagram of the system components. It includes a "Core" block (Harmony Core Service) which is highlighted with a green box. Other components include "EVSYS", "TC3", "NVMCTRL", "SERCOM4", "SERCOM1", "TIME", "SPT", "EIC", and "WINC". The right window, titled "Configuration Options*", shows the "Core" configuration options. A red box highlights the "Number of Applications" field, which is set to 6. A blue box highlights the "Application Name" field, which is set to "app_drv_winc". A red banner at the top of the right window reads "Increase Number of Application". A blue banner at the bottom of the left window reads "Rename file to 'app_drv_winc'".

Project Graph*

Device Family Pack (DFP) CMSIS Pack System

EVSYS Peripheral Library

TC3 Peripheral Library

NVMCTRL Peripheral Library

SERCOM4 Peripheral Library

SERCOM1 Peripheral Library

TIME System Service

Core Service

SYS_TIME

TMR

SPI Driver

Core Service

Instances

Instance 0

Instance 1

EIC Peripheral Library

WINC

Configuration Options* Help

Increase Number of Application

Core

Generate Harmony Application Files ☒

Application Configuration

Number of Applications 6

Application 0 Configuration

Application 1 Configuration

Application 2 Configuration

Application 3 Configuration

Application 4 Configuration

Application 5 Configuration

Enable Application 5 Configuration ☒

Application Name app_drv_winc

**** Application name must be valid C-Language id

Generate Harmony Driver Common Files ☒

Generate Harmony System Service Common Files ☒

Generate Harmony System Media Files ☐

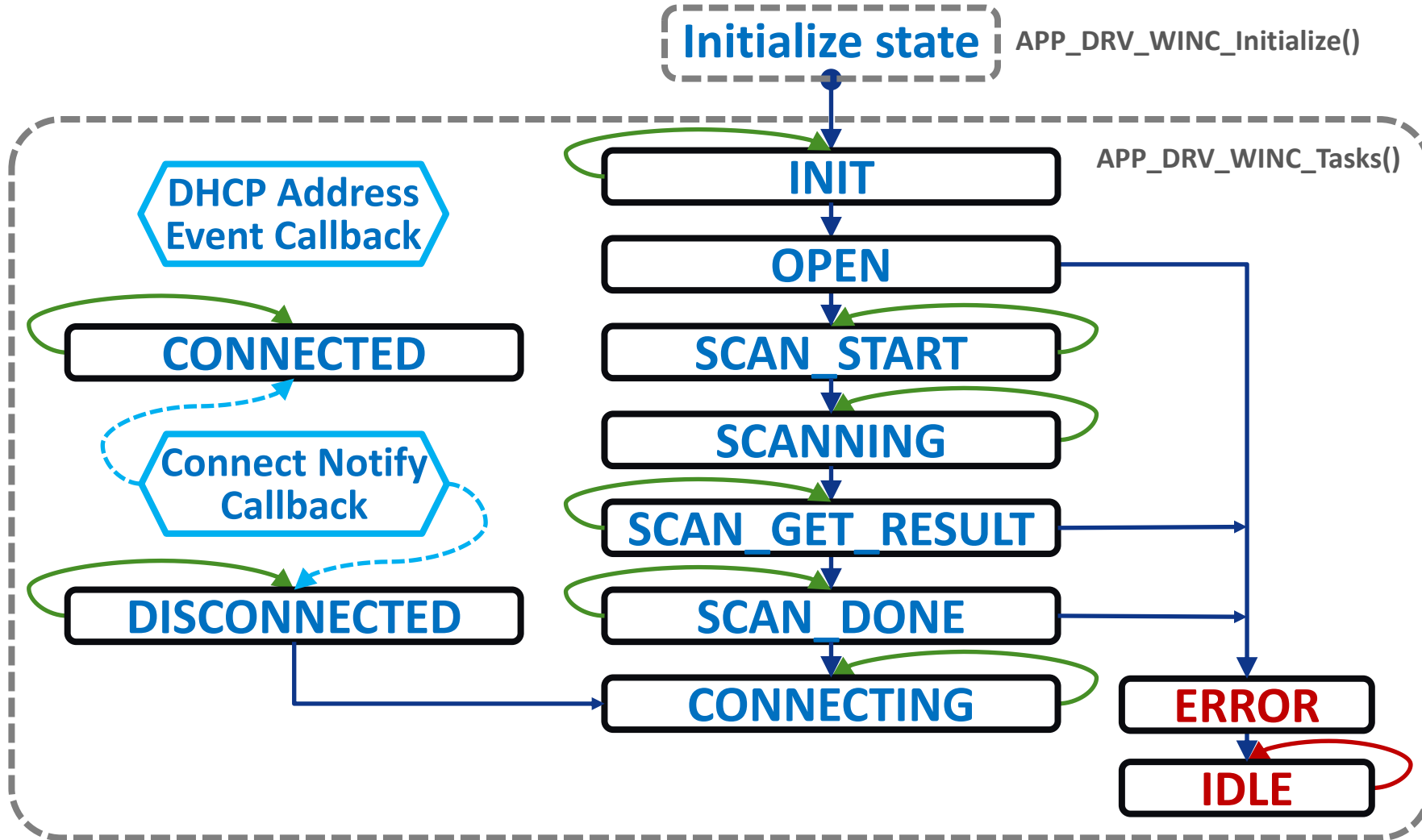
Enable System Interrupt ☒

Enable System Ports ☒

Rename file to "app_drv_winc"

Lab17 AP scan application of SPI Wi-Fi Module

- The state machine to implement in `app_drv_winc.c` as below.



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Add states in “[app_drv_winc.h](#)”.

```
#ifndef _APP_DRV_WINC_H
#define _APP_DRV_WINC_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

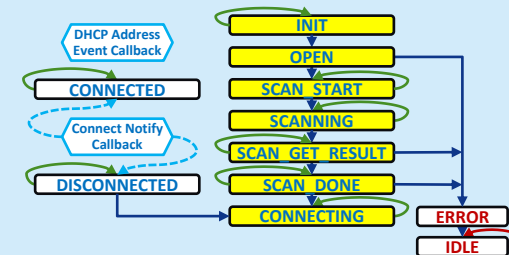
typedef enum
{
    APP_DRV_WINC_STATE_INIT=0,
    // APP_DRV_WINC_STATE_SERVICE_TASKS,
    APP_DRV_WINC_STATE_OPEN,
    APP_DRV_WINC_STATE_SCAN_START,
    APP_DRV_WINC_STATE_SCANNING,
    APP_DRV_WINC_STATE_SCAN_GET_RESULT,
    APP_DRV_WINC_STATE_SCAN_DONE,
    APP_DRV_WINC_STATE_CONNECTING,
    APP_DRV_WINC_STATE_CONNECTED,
    APP_DRV_WINC_STATE_DISCONNECTED,
    APP_DRV_WINC_STATE_ERROR,
    APP_DRV_WINC_STATE_IDLE,
} APP_DRV_WINC_STATES;

typedef struct
{
    APP_DRV_WINC_STATES state;
} APP_DRV_WINC_DATA;
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



✖ Lab17 AP scan application of SPI Wi-Fi Module

- Construct new states flow of task in “app_drv_winc.c”.

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT:
            app_drv_wincData.state = APP_DRV_WINC_STATE_OPEN;
            break;

        case APP_DRV_WINC_STATE_OPEN:
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_START;
            break;

        case APP_DRV_WINC_STATE_SCAN_START:
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;
            break;

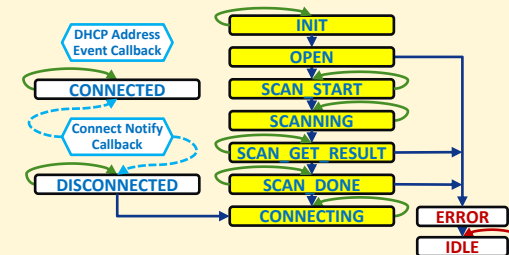
        case APP_DRV_WINC_STATE_SCANNING:
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_GET_RESULT;
            break;

        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_DONE;
            break;

        case APP_DRV_WINC_STATE_SCAN_DONE :
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;
            break;

        case APP_DRV_WINC_STATE_CONNECTING:
            break;
    }
}
```

Overwrite original example code



Next page to continue

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Construct last four states of task in “app_drv_winc.c”.

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT: ...
        case APP_DRV_WINC_STATE_OPEN: ...
        case APP_DRV_WINC_STATE_SCAN_START: ...
        case APP_DRV_WINC_STATE_SCANNING: ...
        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:
            ...

        case APP_DRV_WINC_STATE_SCAN_DONE :
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;
            break;

        case APP_DRV_WINC_STATE_CONNECTING:
            break;

        case APP_DRV_WINC_STATE_CONNECTED:
            break;

        case APP_DRV_WINC_STATE_DISCONNECTED:
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;
            break;

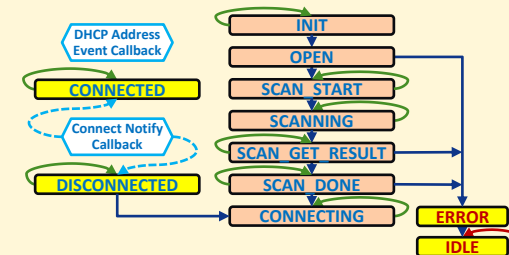
        case APP_DRV_WINC_STATE_ERROR:
            app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE;
            break;

        case APP_DRV_WINC_STATE_IDLE:
            break;
    }
}
```

Overwrite original example code

Insert between this

Insert between this



✂ Lab17 AP scan application of SPI Wi-Fi Module

- Header include and Global Declarations in “app_drv_winc.c”.

Keep this line

```
#include "app_drv_winc.h"
#include "app_drv_usart1.h"
#include "driver/spi/drv_spi.h"
#include "driver/winc/include/wdrv_winc_client_api.h"
#include <stdio.h>
#include <string.h>
```

```
// *****
// *****
// Section: Global Data Definitions
// *****
// *****
```

```
#define WLAN_SSID "RTC_ROOM"
#define WLAN_AUTH_WPA_PSK
#define WLAN_PSK "0800717718"
WDRV_WINC_AUTH_CONTEXT authCtx;
WDRV_WINC_BSS_CONTEXT bssCtx;
WDRV_WINC_BSS_INFO BSSInfo;
bool foundBSS;
```

```
DRV_HANDLE WINCHandle;
```

```
// *****
/* Application Data
```

```
Summary:
    Holds application data
```

```
Description:
    This structure holds the application's data.
```

```
Remarks:
```

```
...
```

SSID of Target AP to connect
The Auth type and Password
Flag foundBSS to note
Target AP has found

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- The callback functions for DHCP address and connection status.

```
// Section: Application Callback Functions
// *****
// *****
void DHCPAddressEventCallback(DRV_HANDLE handle, uint32_t ipAddress)
{
    char s[20];
    USART1printf("IP address is %s\r\n", inet_ntop(AF_INET, &ipAddress, s, sizeof(s)));
}

void ConnectNotifyCallback(DRV_HANDLE handle, WDRV_WINC_ASSOC_HANDLE assocHandle,
                           WDRV_WINC_CONN_STATE currentState, WDRV_WINC_CONN_ERROR errorCode)
{
    switch( currentState )
    {
        case WDRV_WINC_CONN_STATE_CONNECTED:
            USART1printf("Connected\r\n");

            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTED;
            break;

        case WDRV_WINC_CONN_STATE_DISCONNECTED:
            USART1printf("Disconnected\r\n");

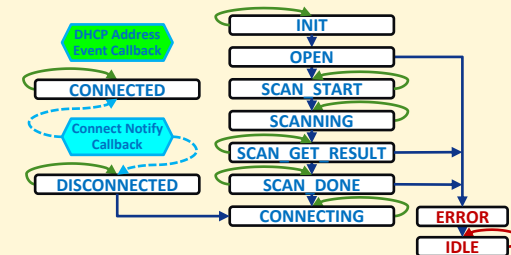
            app_drv_wincData.state = APP_DRV_WINC_STATE_DISCONNECTED;
            break;

#ifdef WDRV_WINC_DEVICE_BSS_ROAMING
        case WDRV_WINC_CONN_STATE_ROAMED:
            break;
#endif
    }
}
```

Callback for DHCP assigned IP Address

Callback for Connection status notify

Dynamic state depend to configuration



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Wait WINC Driver ready in state **INIT**, Open Driver and handling state **ERROR**.

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT:
            if(WDRV_WINC_Status(sysObj.drvWifiWinc) == SYS_STATUS_READY)
            {
                app_drv_wincData.state = APP_DRV_WINC_STATE_OPEN;
            }
            break;
```

Wait WINC Driver ready

```
        case APP_DRV_WINC_STATE_OPEN:
            WINCHandle = WDRV_WINC_Open(0, 0);
            if(WINCHandle == DRV_HANDLE_INVALID)
            {
                app_drv_wincData.state = APP_DRV_WINC_STATE_ERROR;
                break;
            }
        ...
```

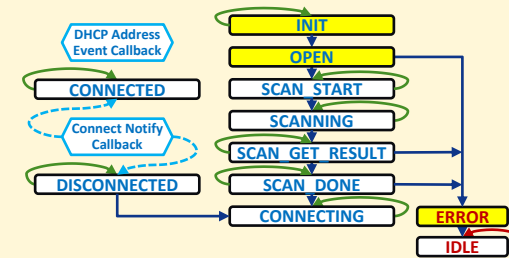
Open WINC Driver to get Driver Handle

```
        case APP_DRV_WINC_STATE_SCAN_START:
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;
            break;
```

```
        case APP_DRV_WINC_STATE_ERROR:
            USART1printf("DRV_WINC error!\r\n");
        ...
```

Handling error state message

```
        case APP_DRV_WINC_STATE_IDLE:
            app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE;
            break;
        ...
    }
}
```



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Register DHCP callback to get IP address and start find AP station

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT: ...
        case APP_DRV_WINC_STATE_OPEN: ...

        case APP_DRV_WINC_STATE_SCAN_START:
            WDRV_WINC_IPUseDHCPSet( WINCHandle, &DHCPAddressEventCallback );

            if( WDRV_WINC_BSSFindFirst(WINCHandle, WDRV_WINC_ALL_CHANNELS, true, NULL, NULL)
                == WDRV_WINC_STATUS_OK )
            {
                app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;
                foundBSS = false;
            }
            break;

        case APP_DRV_WINC_STATE_SCANNING:
            if( WDRV_WINC_BSSFindInProgress(WINCHandle) == false )
            {
                USART1printf("Scan complete, %d AP(s) found\r\n",
                    WDRV_WINC_BSSFindGetNumBSSResults(WINCHandle));
                app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_GET_RESULT;
            }
            break;

        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_DONE;
            break;

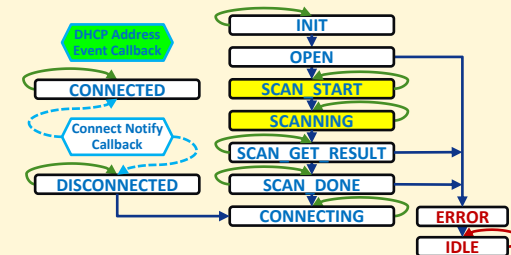
        case APP_DRV_WINC_STATE_SCAN_DONE :
            ...
    }
}
```

Register the callback function to retrieval DHCP IP address and try find first AP station

Check if AP scanning to finish or not

Keep this line break;

Keep this line break;



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- List found AP stations and check if Target AP in list or not.

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT: ...
        case APP_DRV_WINC_STATE_OPEN: ...
        case APP_DRV_WINC_STATE_SCAN_START: ...
        case APP_DRV_WINC_SCANNING: ...

        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:
            if( WDRV_WINC_BSSFindGetInfo(WINCHandle, &BSSInfo) == WDRV_WINC_STATUS_OK )
            {
                USART1printf("AP found: RSSI: %d %s\r\n", BSSInfo.rssi, BSSInfo.ctx.ssid.name);

                if( ((sizeof(WLAN_SSID)-1) == BSSInfo.ctx.ssid.length) &&
                    (0 == memcmp(BSSInfo.ctx.ssid.name, WLAN_SSID, BSSInfo.ctx.ssid.length)) )
                {
                    foundBSS = true;
                }
            }

            break;

        case APP_DRV_WINC_STATE_SCAN_DONE :
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;
            break;

        case APP_DRV_WINC_STATE_CONNECTING:
            break;

        case APP_DRV_WINC_STATE_CONNECTED:
            break;

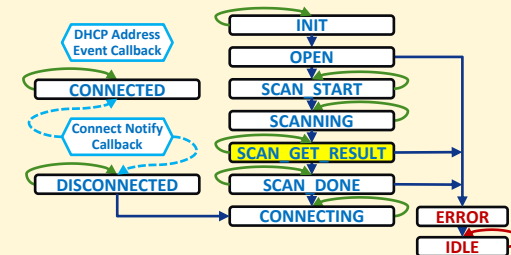
        ...
    }
}
```

List found AP stations and check if Target AP in list or not.

Set foundBSS flag if find Target AP

Next page to continue

Keep this line



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Check if end of AP finding and try connect Target AP if it's existed.

```
case APP_DRV_WINC_STATE_SCAN_GET_RESULT:
    if( WDRV_WINC_BSSFindGetInfo(WINCHandle, &BSSInfo) == WDRV_WINC_STATUS_OK )
    {
        ...
        foundBSS = true;
    }
    switch( WDRV_WINC_BSSFindNext(WINCHandle, NULL) )
    {
    case WDRV_WINC_STATUS_BSS_FIND_END:
        if( foundBSS == true )
        {
            USART1printf("Target AP found, trying to connect\r\n");
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_DONE;
        }
        else
        {
            USART1printf("Target BSS not found\r\n");
            app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE;
        }
        break;
    case WDRV_WINC_STATUS_NOT_OPEN:
    case WDRV_WINC_STATUS_INVALID_ARG:
        app_drv_wincData.state = APP_DRV_WINC_STATE_ERROR;
        break;
    default:
        break;
    }
    break;
case APP_DRV_WINC_STATE_SCAN_DONE : ...
```

Check if end of AP finding

Try to connect Target AP

Insert between this

Insert between this

Keep this line

```
graph TD
    INIT[INIT] --> OPEN[OPEN]
    OPEN --> SCAN_START[SCAN_START]
    SCAN_START --> SCANNING[SCANNING]
    SCANNING --> SCAN_GET_RESULT[SCAN_GET_RESULT]
    SCAN_GET_RESULT --> SCAN_DONE[SCAN_DONE]
    SCAN_GET_RESULT --> ERROR[ERROR]
    SCAN_DONE --> CONNECTING[CONNECTING]
    CONNECTING --> CONNECTED[CONNECTED]
    CONNECTED --> DISCONNECTED[DISCONNECTED]
    DISCONNECTED --> SCAN_START
    DISCONNECTED --> SCAN_START
    CONNECTED --> SCAN_START
    DISCONNECTED --> SCAN_START
```

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Start connect to Target AP, the result will return through **ConnectNotifyCallback**.

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        ...
        case APP_DRV_WINC_STATE_SCAN_DONE :
            app_drv_wincData.state = APP_DRV_WINC_STATE_ERROR; Set SSID of Target AP

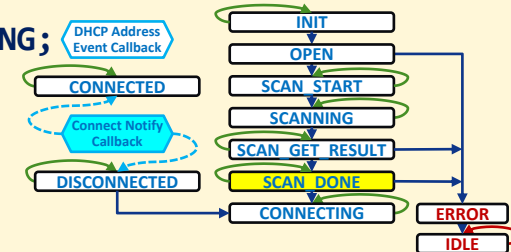
            if( WDRV_WINC_BSSCtxSetDefaults(&bssCtx) != WDRV_WINC_STATUS_OK )
            { break; }

            if( WDRV_WINC_BSSCtxSetSSID(&bssCtx, (uint8_t*)WLAN_SSID, strlen(WLAN_SSID)) !=
                WDRV_WINC_STATUS_OK )
            { break; }

            Authentication must use respective function
            #if defined(WLAN_AUTH_OPEN)
                if( WDRV_WINC_AuthCtxSetOpen(&authCtx) ) != WDRV_WINC_STATUS_OK )
            #elif defined(WLAN_AUTH_WPA_PSK)
                if( WDRV_WINC_AuthCtxSetWPA(&authCtx, (uint8_t*)WLAN_PSK, strlen(WLAN_PSK)) !=
                    WDRV_WINC_STATUS_OK )
            #endif
            { break; }

            Connect AP station and register callback
            if( WDRV_WINC_BSSConnect(WINCHandle, &bssCtx, &authCtx, &ConnectNotifyCallback) ==
                WDRV_WINC_STATUS_OK )
            {
                app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;
            }
            Keep this line
            break;

            case APP_DRV_WINC_STATE_CONNECTING:
                break;
        ...
    }
}
```



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Active IP link in state **CONNECTED** and handling disconnect in state **DISCONNECTED**.

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT: ...
        case APP_DRV_WINC_STATE_OPEN: ...
        case APP_DRV_WINC_STATE_SCAN_START: ...
        case APP_DRV_WINC_SCANNING: ...
        case APP_DRV_WINC_STATE_SCAN_GET_RESULT: ...
        case APP_DRV_WINC_STATE_SCAN_DONE : ...
```

```
        case APP_DRV_WINC_STATE_CONNECTING:
            break;
```

```
        case APP_DRV_WINC_STATE_CONNECTED:
            WDRV_WINC_IPLinkActive( WINCHandle );
            break;
```

Keep this line

```
        case APP_DRV_WINC_STATE_DISCONNECTED:
            if( WDRV_WINC_BSSReconnect(WINCHandle, &ConnectNotifyCallback) ==
                                                    WDRV_WINC_STATUS_OK )
```

```
            {
                app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;
            }
```

Keep this line

```
            break;

        case APP_DRV_WINC_STATE_ERROR:
            USART1printf("DRV_WINC error!\r\n");

            app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE;
            break;
```

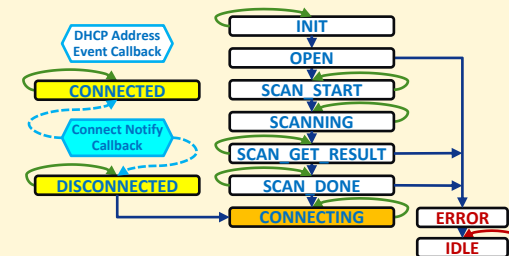
```
        case APP_DRV_WINC_STATE_IDLE:
```

Do not thing in state **CONNECTING**

Connect Notify Callback

will handling state change of

CONNECTED and **DISCONNECTED**



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Use **LED2** to instead of removed LED1 in “main.c” to indicate Driver non-blocking.

```
SYS_TIME_HANDLE tmrHandle;

// *****
// *****
// Section: Main Entry Point
// *****
// *****

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

    SYS_TIME_DelayMS(1000, &tmrHandle);

    while ( true )
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true )
        {
            LED2_Toggle();
            SYS_TIME_DelayMS(1000, &tmrHandle);
        }

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

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

    return ( EXIT_FAILURE );
}
```

Change to **LED2_Toggle()**

✖ Lab17 AP scan application of SPI Wi-Fi Module

- Modify I2C auto probe count to disable **Instance 1** of **DRV_I2C** in “app_drv_i2c2.c”

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT:
            ...
            // SYS_TIME_HANDLE TimeHandle;
            // SYS_TIME_DelayMS(10, &TimeHandle );
            // while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

            for( int Inst = DRV_I2C_INDEX_0 ; Inst <= DRV_I2C_INDEX_0 ; Inst++ )
            {
                I2C2Handle_TJ3 = DRV_I2C_Open(Inst, DRV_IO_INTENT_READWRITE);
                if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)
                {
                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
                    return;
                }
                I2C2WriteBuf_TJ3[0] = 0x1D;
                I2C2WriteBuf_TJ3[1] = 0x80;
                DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                         I2C2WriteBuf_TJ3, 2, & I2C2WriteReadHandle_TJ3 );
            }

            USART1printf("G-Sensor Software Reset Completed\r\n");

            I2C2WriteBuf_TJ3[0] = 0x0F;
            DRV_I2C_WriteReadTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                         I2C2WriteBuf_TJ3, 1, I2C2ReadBuf_TJ3, 1, &I2C2WriteReadHandle_TJ3 );
            while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
            ...
    }
```

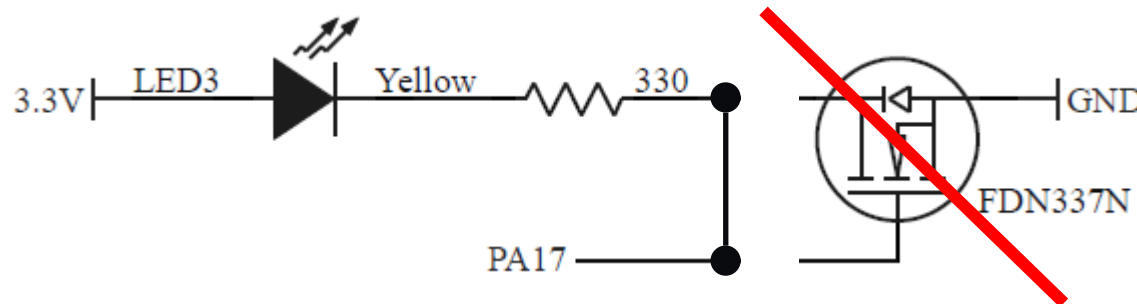
Change to **DRV_I2C_INDEX_0**

✖ Lab17 AP scan application of SPI Wi-Fi Module

- The **PA17** has connect to a **MOSFET** to drive the **LED3**, it will **affect** the **SPI SCK** to communication with WINC1500, please remove it and direct short to LED3 as below.

Wi-Fi 7 click (WINC1500)	Arduino socket	SAMD21 Pin Configure (Custom Name)	SAMD21 Pin MUX
SDI	D11/MOSI	PA16	SERCOM1-PAD0
SCK	D13/SCK	PA17	SERCOM1-PAD1
CS	D10/SS	PA18	SERCOM1-PAD2
SDO	D12/MISO	PA19	SERCOM1-PAD3
INT	A3/D17	PA04	EXINT4, Falling Edge Detect
RST	D8	PA06 (WDRV_WINC_RESETN)	GPIO output, Latch High
EN	D6/PWM	PA20 (WDRV_WINC_CHIP_EN)	GPIO output, Latch High

Arduino UNO Socket	
15	D8
16	D9/PWM
17	D10/PWM/SS
18	D11/PWM/MOSI
19	D12/MISO SERCOM1
20	D13/SCK Wi-Fi 7 Click
21	GND
22	AREF
23	D18/SDA
24	D19/SCL



⚡ Lab17 AP scan application of SPI Wi-Fi Module

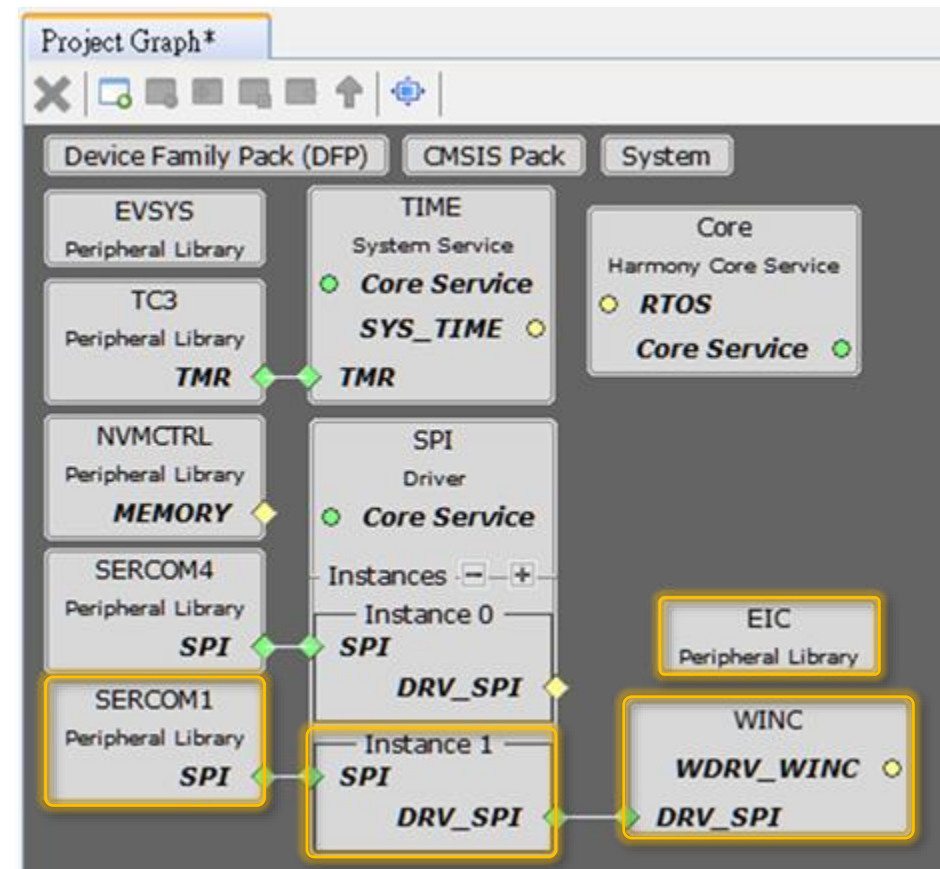
- Compiler and programming your project to EVB.
- Check USART1 debug output to see the result

```

VT COM3 - Tera Term VT
File Edit Setup Control Window Help
DRV_USART1: Hello world
Wrote 0x24 to EEPROM 24AA
G-Sensor Software Reset Completed
G-Sensor WHO AM I : 0x35
G-Sensor Initial Completed
Scan complete, 12 AP(s) found
AP found: RSSI: -66 BravoGroup
AP found: RSSI: -67 mchp-secure
AP found: RSSI: -68 mguest
AP found: RSSI: -55 Taipei RTC ADSL
AP found: RSSI: -73 Leedarson03
AP found: RSSI: -48 MLI_WiFi
AP found: RSSI: -47 MLI_CHTWiFi
AP found: RSSI: -48 MLI_WiFi_OTP
AP found: RSSI: -41 roving1
AP found: RSSI: -76 TP-Link_E13A
AP found: RSSI: -48
Target AP found, trying to connect
Connected
IP address is 192.168.4.134
  
```

Arduino UNO Socket

15	D8
16	D9/PWM
17	D10/PWM/SS
18	D11/PWM/MOSI
19	D12/MISO
20	D13/SCK SERCOM1
21	GND
22	AREF
23	D18/SDA
24	D19/SCL



✂ Lab17 AP scan application of SPI Wi-Fi Module

- Discuss

How to setup the **SPI Software Slave Select pin** in WINC SPI Driver ?

You can apply a **Custom GPIO** via Transfer Setup in **WDRV_WINC_SPIInitialize()** of “wdrv_winc_spi.c”

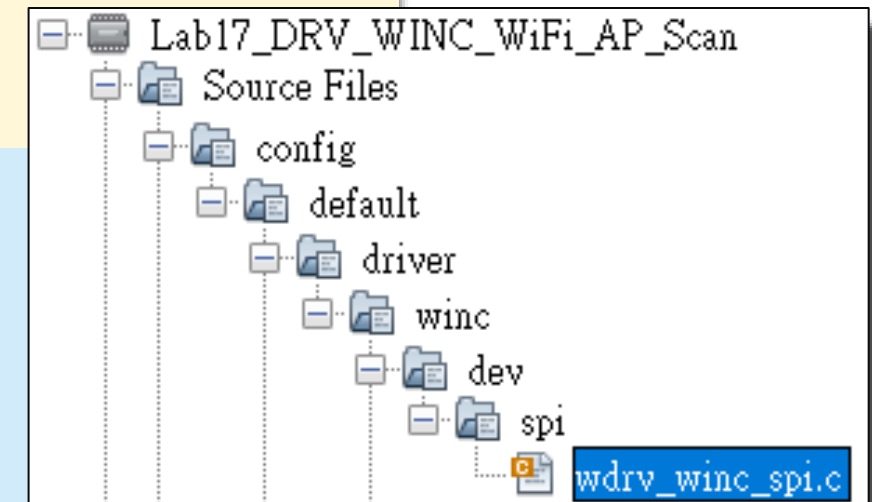
```
void WDRV_WINC_SPIInitialize(void)
{
    ...
    if (DRV_HANDLE_INVALID == spiHandle)
    {
        spiHandle = DRV_SPI_Open(WDRV_WINC_SPI_INDEX, DRV_IO_INTENT_READWRITE | ...

        if (DRV_HANDLE_INVALID == spiHandle)
        {
            WDRV_DBG_ERROR_PRINT("SPI init failed\r\n");
        }

        DRV_SPI_TRANSFER_SETUP setup;
        setup.baudRateInHz = 1000000;
        setup.clockPhase = DRV_SPI_CLOCK_PHASE_VALID_LEADING_EDGE;
        setup.clockPolarity = DRV_SPI_CLOCK_POLARITY_IDLE_LOW;
        setup.dataBits = DRV_SPI_DATA_BITS_8;
        setup.chipSelect = (SYS_PORT_PIN)WDRV_WINC_CS_PIN;
        setup.csPolarity = DRV_SPI_CS_POLARITY_ACTIVE_LOW;
        if(DRV_SPI_TransferSetup(spiHandle, &setup) == false)
        {
            WDRV_DBG_ERROR_PRINT("SPI setup failed\r\n");
        }
    }

    DRV_SPI_TransferEventHandlerSet( spiHandle, _WDRV_WINC_SPITransferEventHandler, 0);
}
```

Setup software Slave Select pin
WDRV_WINC_CS
in WDRV_WINC_SPIInitialize()



✖ Lab17 AP scan application of SPI Wi-Fi Module

- Discuss

What's difference of WINC and WINC (Lite) ?

The screenshot displays the Microchip Studio IDE interface for configuring the WINC and WINC (Lite) modules. The component palette on the left shows the 'Wireless' section with 'WINC' and 'WINC (Lite)' selected. The central workspace shows two module configurations: 'WINC' and 'WINC (Lite)'. The 'WINC' configuration includes the 'SPI Driver' and 'WDRV_WINC' components. The 'WINC (Lite)' configuration includes the 'SPI Driver', 'WDRV_WINC', and 'Serial Bridge Application' components. The configuration windows on the right show the settings for each module.

WINC Configuration:

- WiFi Device: WINC1500
- Driver Log Level: None
- SPI Driver: drv_spi_1
- Interrupt Source: EIC
- EIC Channel 4: ☒
- Firmware Version: 19.6.1
- Driver Mode: Socket Mode

WINC (Lite) Configuration:

- WiFi Device: WINC1500
- Driver Log Level: None
- SPI Driver: drv_spi_1
- Interrupt Source: EIC
- EIC Channel 4: ☒
- Firmware Version: 19.7.3
- Driver Mode: Socket Mode
- Reset assert time (in ms): 0
- Enable serial bridge?: ☒
- Support baud rate changes?: ☐
- Send sync ID before request?: ☐

Memory Usage:

Component	Used (bytes)	Free (bytes)	Percentage
Data	32,768 (0x8000)	-	14%
Program	262,144 (0x40000)	-	27%

Support both v19.6.1 and v19.7.3

Support only v19.7.3

Support Serial Bridge

MICROCHIP

SPI Driver – SD Card Driver Library

Harmony Driver

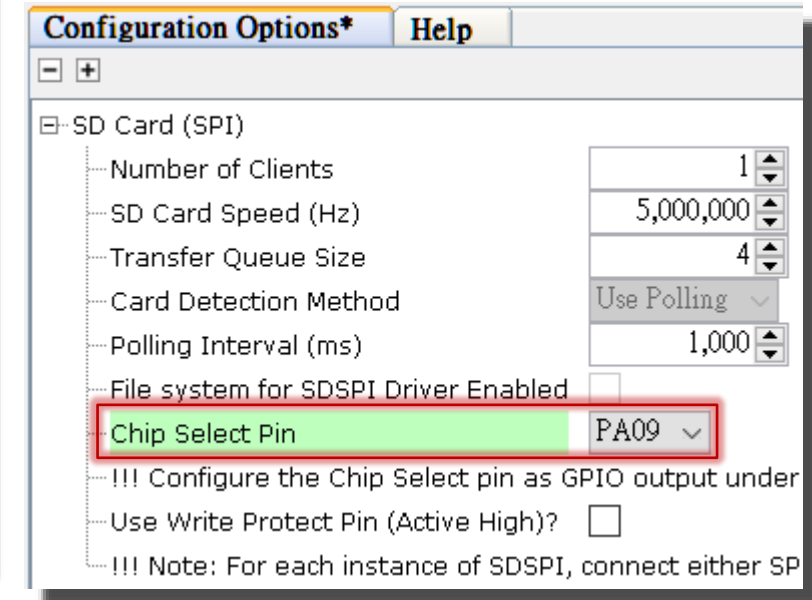
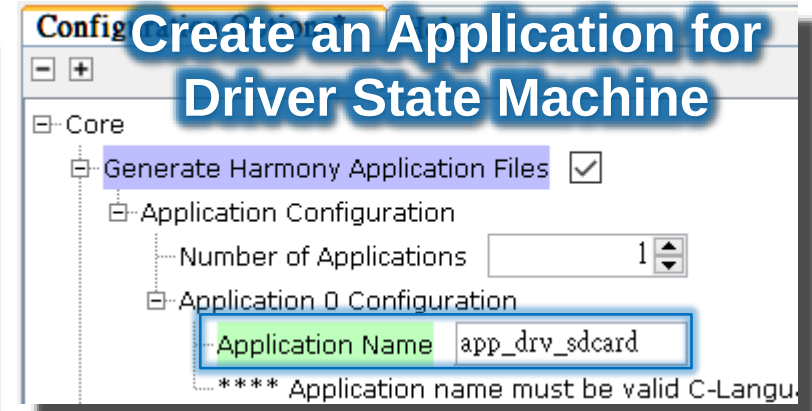
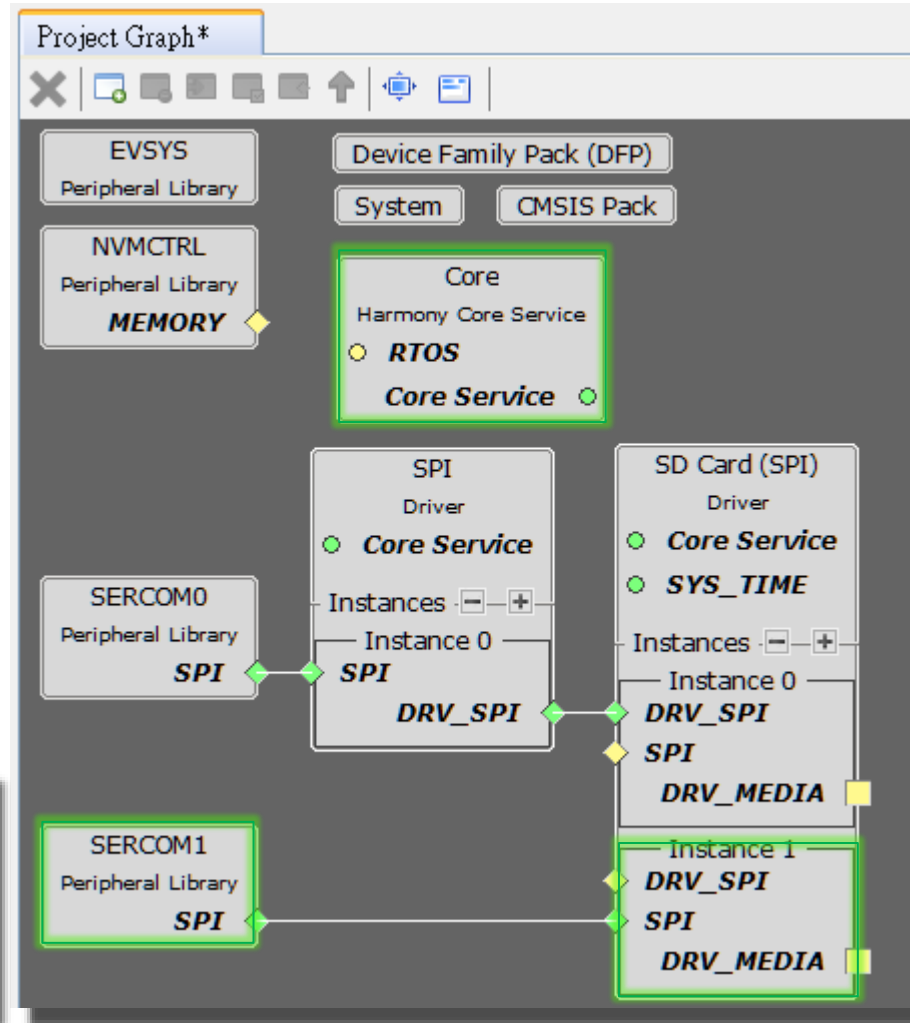
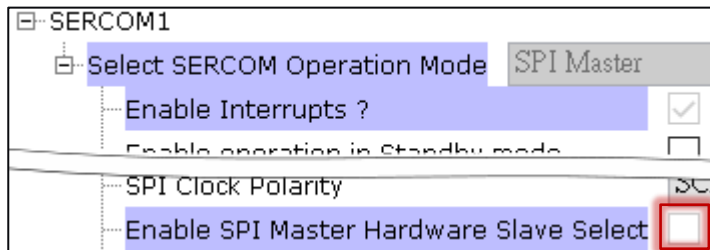
SD Card (SPI) Driver Library

- **Introduction**

- This driver library provides application ready routines to read and write to the SD Card, thus minimizing developer's awareness of working of the SD Card protocol.
- Supports asynchronous (non-blocking) and synchronous (blocking) mode of operation.
- Supports both DMA and non-DMA modes.
- The SDSPI driver is capable of interacting to the SD Card either via SPI PLIB interface or via SPI Driver interface, selectable in MHC configuration of the SDSPI driver.
- The driver API interface can be used directly by the application to read/write directly from the SD Card Or the SDSPI driver can be used with a file system.

SD Card (SPI) Driver Library

Connect PLIB SERCOM
through SPI Driver
or
Direct to PLIB SERCOM



Always to use software chip select of SPI SD Card Driver

SD Card (SPI) Driver Library

API functions of the SD Card (SPI) Driver Library.

- System Functions

DRV_SDSPI_Initialize	Initializes the SD Card driver.
DRV_SDSPI_Status	Provides the current status of the SDSPI driver module.
DRV_SDSPI_Tasks	Maintains the driver's state machine.

- Core Client Functions

DRV_SDSPI_Open	Opens the specified SD Card driver instance and returns a handle to it.
DRV_SDSPI_Close	Closes an opened-instance of the SD Card driver.
DRV_SDSPI_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. For the synchronous SDSPI driver, the event handler is only used by the file system.

- Data Transfer Functions

DRV_SDSPI_AsyncRead	Reads blocks of data from the specified block address of the SD Card.
DRV_SDSPI_SyncRead	Reads blocks of data from the specified block address of the SD Card.
DRV_SDSPI_Read	This routine provides interface to the file system to perform a media read operation in synchronous mode of the SDSPI driver.
DRV_SDSPI_SyncWrite	Writes blocks of data starting at the specified address of the SD Card.
DRV_SDSPI_AsyncWrite	Writes blocks of data starting at the specified address of the SD Card.
DRV_SDSPI_Write	This routine provides interface to the file system to perform a media operation in synchronous mode of the SDSPI driver.

SD Card (SPI) Driver Library

API functions of the SD Card (SPI) Driver Library. (cont.)

- Data Transfer Functions

DRV_SDSPI_CommandStatusGet

Gets the current status of the command. For synchronous mode, the command status is only used by the file system.

- Status Functions

DRV_SDSPI_IsAttached

Returns the physical attach status of the SD Card.

DRV_SDSPI_IsWriteProtected

Returns the write protect status of the SD Card.

DRV_SDSPI_GeometryGet

Returns the geometry of the device.

- Data Types and Constants

DRV_SDSPI_COMMAND_HANDLE

This is type DRV_SDSPI_COMMAND_HANDLE.

DRV_SDSPI_COMMAND_STATUS

This is type DRV_SDSPI_COMMAND_STATUS.

DRV_SDSPI_EVENT

Identifies the possible events that can result from a request.

DRV_SDSPI_EVENT_HANDLER

Pointer to a SDSPI Driver Event handler function

DRV_SDSPI_COMMAND_HANDLE_INVALID

SDSPI Driver's Invalid Command Handle.

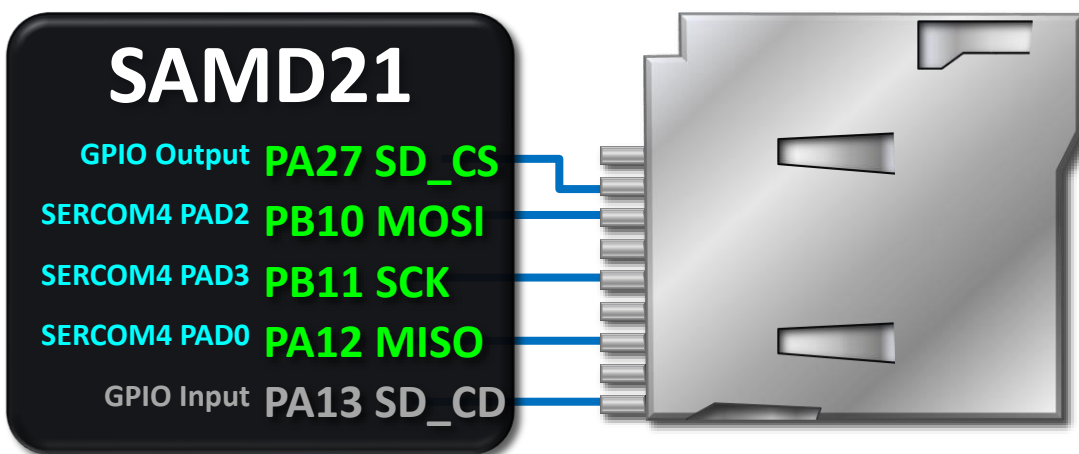
Lab 18

Manage SPI SD card without File System

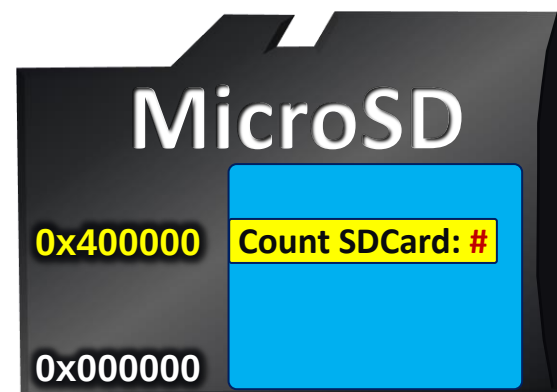
SPI Driver - SD Card Driver Library

🔧 Lab18 Manage SPI SD card without File System

- Increase **Number of Clients** of **SPI Driver Instance 0** to **2** and attach the **SD Card (SPI) Driver** and **PLIB SERCOM4 SPI** to access a **SPI SD Card** without File System.
- The pins define and configuration are same to **Lab7** we did.
- Try store "**Count SDCard: #**" to **block 0x2000** of SD Card , the '**#**' is count number start from **1**.
- Reset EVB to read "**Count SDCard: #**" out and increase the count '**#**' then store it back to SD Card.

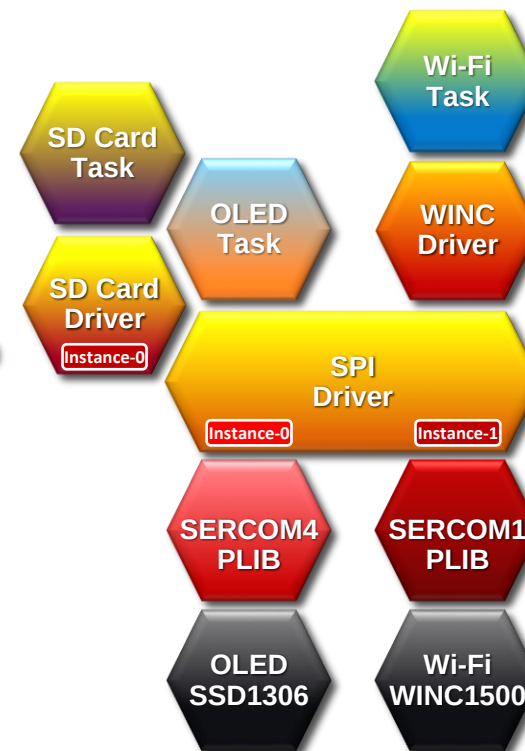
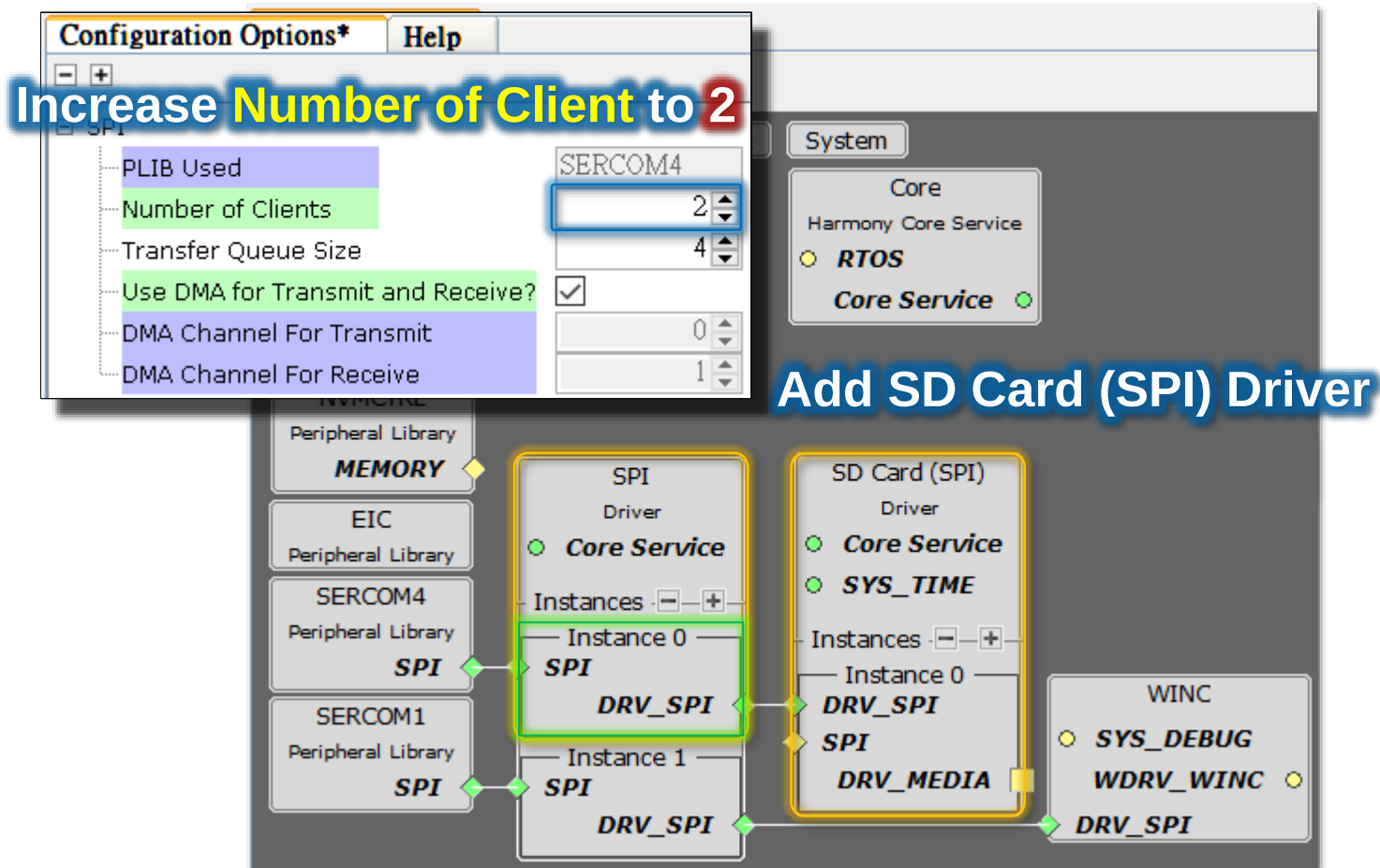


Block Size is **512** bytes
Store Block is **0x2000**
Start Address is **0x40000**



✖ Lab18 Manage SPI SD card without File System

- Attach the **SD Card (SPI) Driver** as consumer of **Instance 0** of **SPI Driver**, It's shared the same **PLIB SERCOM4 SPI** interface with **OLED** display.
- Increase **Number of Clients** to **2** in **Instance 0** to increase a task for SD Card access.



⚙️ Lab18 Manage SPI SD card without File System

- Set **Chip Select** pin to **GPIO PA27** in **SD Card Driver** configuration.

The image shows the Microchip Studio configuration interface for a SAMD21 microcontroller. The main window displays a project configuration tree on the left and a central configuration area. Two configuration windows are overlaid on the main interface.

Left Configuration Window: SPI

- Configuration Options***
 - PLIB Used: SERCOM4
 - Number of Clients: 2
 - Transfer Queue Size: 4
 - Use DMA for Transmit and Receive? ☒
 - DMA Channel For Transmit: 0
 - DMA Channel For Receive: 1

Right Configuration Window: SD Card (SPI)

- Configuration Options***
 - SPI Driver Instance Used: 0
 - Number of Clients: 1
 - SD Card Speed (Hz): 5,000,000
 - Transfer Queue Size: 4
 - Card Detection Method: Use Polling
 - Polling Interval (ms): 1,000
 - File system for SDSPI Driver Enabled: ☐
 - Chip Select Pin: PA27
 - !!! Configure the Chip Select pin as GPIO output under
 - Use Write Protect Pin (Active High)? ☐
 - !!! Note: For each instance of SDSPI, connect either SP

Central Configuration Area:

- Peripheral Library:** TC3, NVMCTRL, EIC, SERCOM4, SERCOM1.
- Core Service:** SYS_TIME, TMR, SPI, SD Card (SPI).
- Instances:** Instance 0 (SPI DRV_SPI), Instance 1 (SPI DRV_SPI).
- WINC:** SYS_DEBUG, WDRV_WINC.

SAMD21 Pin Configuration:

- GPIO Output PA27 SD_CS
- SERCOM4 PAD2 PB10 MOSI
- SERCOM4 PAD3 PB11 SCK
- SERCOM4 PAD0 PA12 MISO
- GPIO Input PA13 SD_CD

✖ Lab18 Manage SPI SD card without File System

- Configure **SERCOM4 SPI** pads and **disable** hardware **Slave Select**. (It should be done in Lab16)

SAMD21

GPIO Output PA27 SD_CS

SERCOM4 PAD2 PB10 MOSI

SERCOM4 PAD3 PB11 SCK

SERCOM4 PAD0 PA12 MISO

GPIO Input PA13 SD_CD

DO on PAD[2] (MOSI)

SCK on PAD[3] (SCK)

DI on PAD[0] (MISO)

Don't use Hardware Slave Select

✖ Lab18 Manage SPI SD card without File System

- Configure **SERCOM4** pads, **PB10(PAD2)**, **PB11(PAD3)**, **PA12(PAD0)**.
- Configure GPIOs : **PA27(Output, Latch High, Custom Name is optional)**, **PA13(Input)**.

Pin Settings × Pin Diagram × Pin Table × Project Graph

Order: Pins ▼ Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode
19	PB10	SERCOM4_PAD2	SERCOM4_PAD2 ▼	Digital
20	PB11	SERCOM4_PAD3	SERCOM4_PAD3 ▼	Digital
21	PA12	SERCOM4_PAD0	SERCOM4_PAD0 ▼	Digital
22	PA13	SD_CD	GPIO ▼	Digital
39	PA27	SD_CS	GPIO ▼	Digital

SD Card (SPI)

SPI Driver Instance Used: 0

Number of Clients: 1

SD Card Speed (Hz): 5,000,000

Transfer Queue Size: 4

Card Detection Method: Use Polling ▼

Polling Interval (ms): 1,000

File system for SDSPI Driver Enabled: ☐

Chip Select Pin: PA27 ▼

High Impedance ▼ n/a

In ▼ Low

Out ▼ High

Pin Settings × Pin Diagram × Pin Table × Project Graph

Package: TQFP48 ▼

GPIO	SERCOM4	PA08	LED2	PA10	PA11	VDDIO	GNDDIO	SERCOM4	SERCOM4	SERCOM4	GPIO_P	PA14	PA15	PA16	LED3	VDDIO	SERCOM4	SERCOM4	GPIO_P	RESET
GPIO	SERCOM4_PAD0																			
	SERCOM4_PAD1																			
	SERCOM4_PAD2																			
	SERCOM4_PAD3																			

SAMD21

GPIO Output PA27 SD_CS

SERCOM4 PAD2 PB10 MOSI

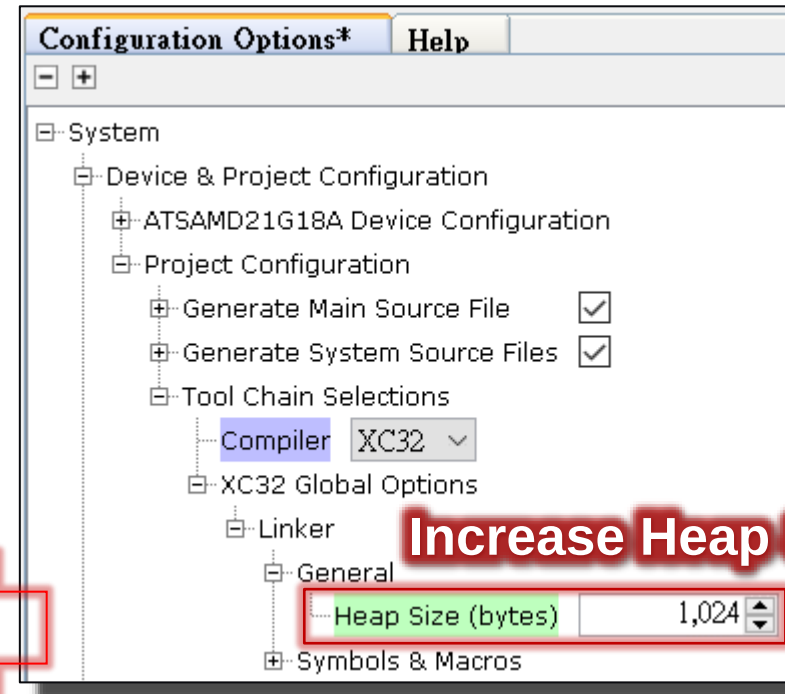
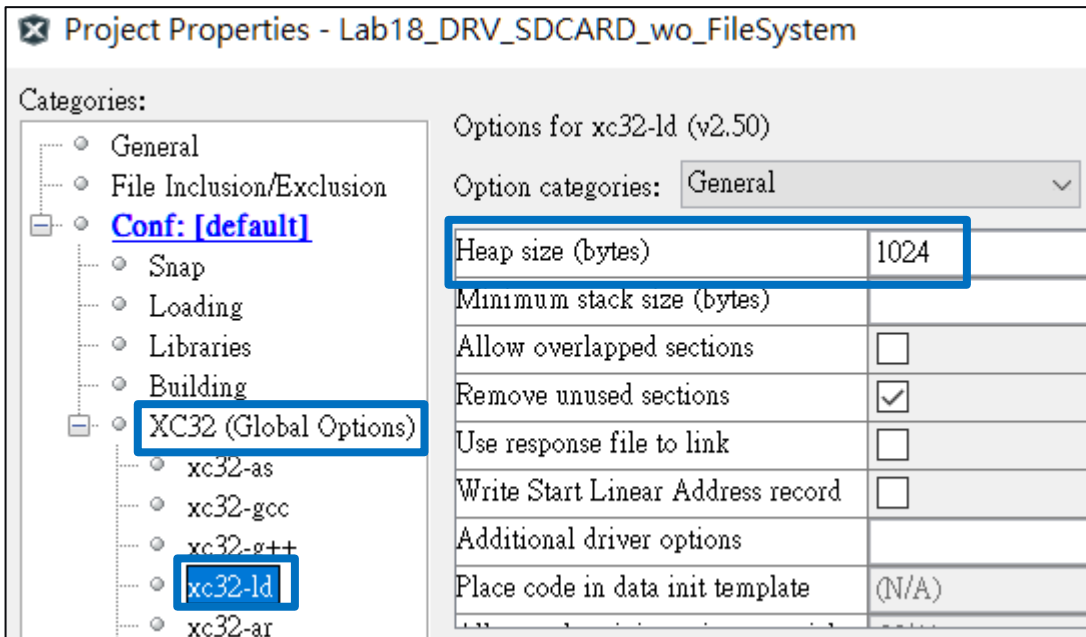
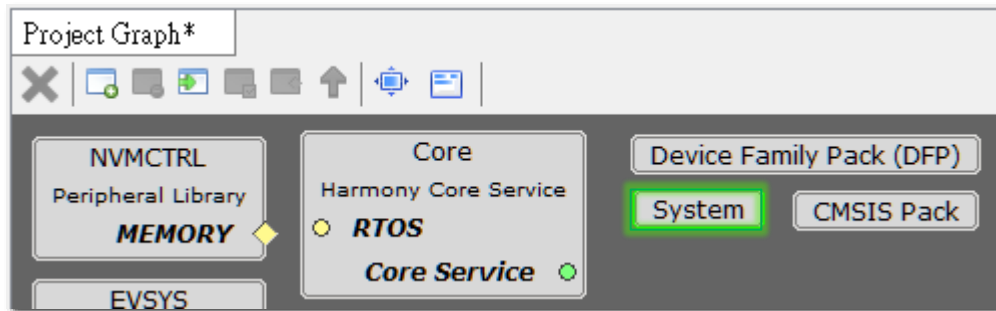
SERCOM4 PAD3 PB11 SCK

SERCOM4 PAD0 PA12 MISO

GPIO Input PA13 SD_CD

✖ Lab18 Manage SPI SD card without File System

- Increase **Heap Size** to **1024** bytes in **System** to reserve more memory for string process.

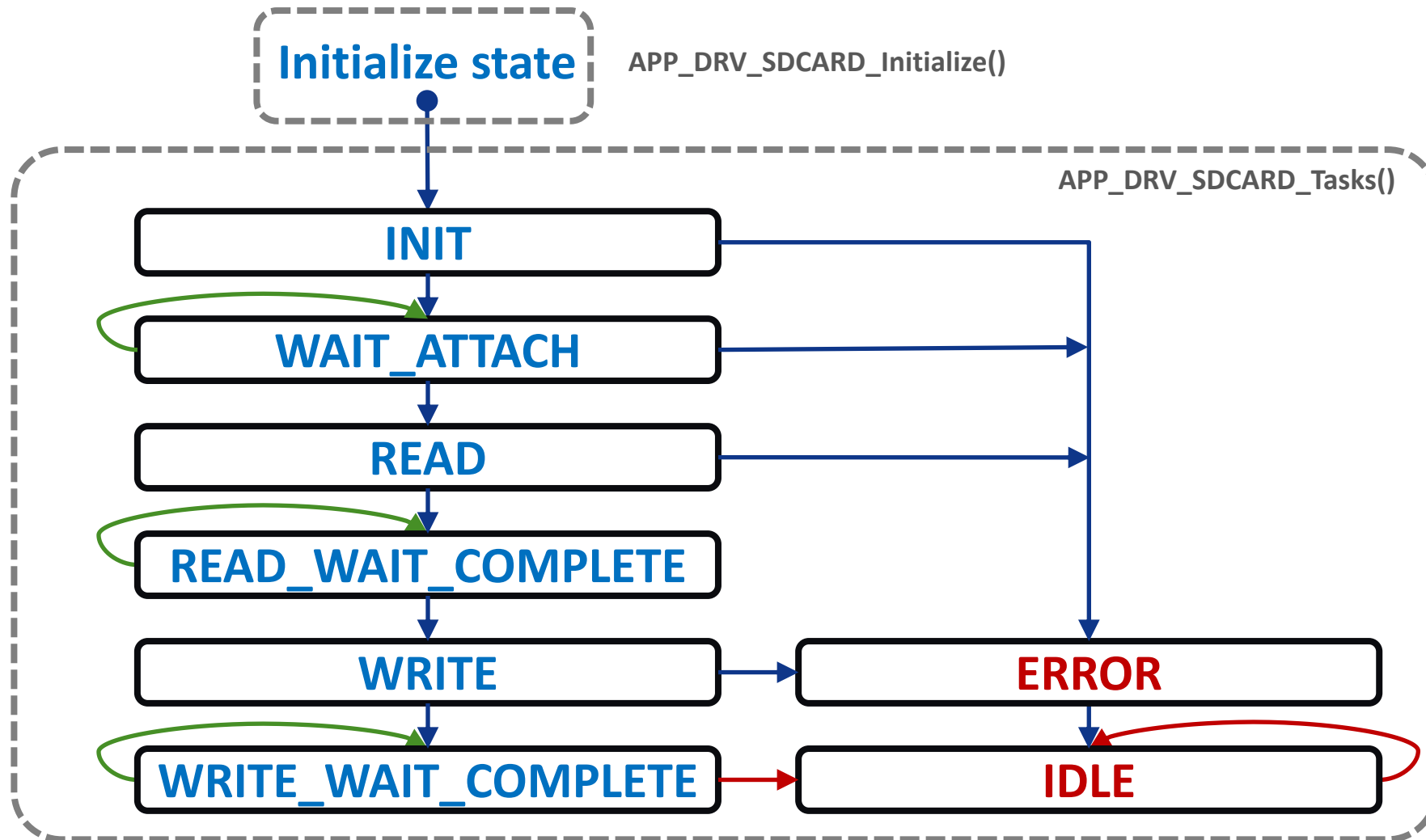


Increase Heap Size to 1024

Harmony Heap setting will overwrite the Heap Size of xc32-ld configuration in Project Properties

⚙️ Lab18 Manage SPI SD card without File System

- The new state machine we wish to complete as below.



⚙️ Lab18 Manage SPI SD card without File System

- Add states in “`app_drv_sdcard.h`”.

```
#ifndef _APP_DRV_SDCARD_H
#define _APP_DRV_SDCARD_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_SDCARD_STATE_INIT=0,
    // APP_DRV_SDCARD_STATE_SERVICE_TASKS,
    APP_DRV_SDCARD_STATE_WAIT_ATTACH,
    APP_DRV_SDCARD_STATE_READ,
    APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE,
    APP_DRV_SDCARD_STATE_WRITE,
    APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE,
    APP_DRV_SDCARD_STATE_ERROR,
    APP_DRV_SDCARD_STATE_IDLE,
} APP_DRV_SDCARD_STATES;

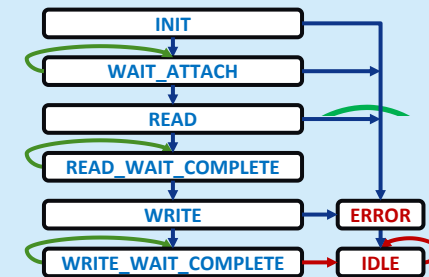
typedef struct
{
    APP_DRV_SDCARD_STATES state;
} APP_DRV_SDCARD_DATA;

void APP_DRV_SDCARD_Initialize ( void );
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



✖ Lab18 Manage SPI SD card without File System

- Construct new states flow of task in “`app_drv_sdcard.c`”.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WAIT_ATTACH;
            break;

        case APP_DRV_SDCARD_STATE_WAIT_ATTACH:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
            break;

        case APP_DRV_SDCARD_STATE_READ:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE;
            break;

        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE;
            break;

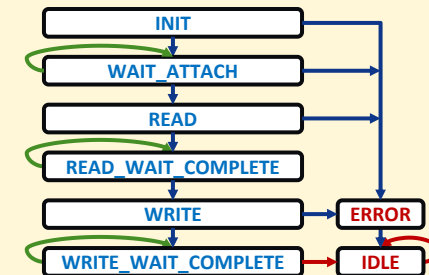
        case APP_DRV_SDCARD_STATE_WRITE:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE;
            break;

        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;
            break;

        case APP_DRV_SDCARD_STATE_ERROR:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;
            break;

        case APP_DRV_SDCARD_STATE_IDLE:
            break;
    }
}
```

Overwrite original example code



✂ Lab18 Manage SPI SD card without File System

- Header include and Global Declarations in “app_drv_sdcard.c”.

```
Keep this line #include "app_drv_sdcard.h"
#include "app_drv_usart1.h"
#include "driver/sdspi/drv_sdspi.h"
#include <stdio.h>
#include <string.h>

// *****
// *****
// Section: Global Data Definitions
// *****
// *****

#define SDCARD_BLOCK_START          0x2000
#define SDCARD_BUFFER_SIZE          (4096U)
#define SDCARD_GEOMETRY_TABLE_READ_ENTRY (0)
#define SDCARD_GEOMETRY_TABLE_WRITE_ENTRY (1)
#define SDCARD_GEOMETRY_TABLE_ERASE_ENTRY (2)
SYS_MEDIA_GEOMETRY *SDCardGeometry = NULL;
int SDCardReadBlocks;
int SDCardWriteBlocks;
int SDCardEraseBlocks;

DRV_HANDLE SDCardHandle;
DRV_SDSPI_COMMAND_HANDLE SDCardReadHandle;
DRV_SDSPI_COMMAND_HANDLE SDCardWriteHandle;
volatile int SDCardWriteComplete = false;
volatile int SDCardReadComplete = false;
char SDCardWriteBuf[SDCARD_BUFFER_SIZE];
char SDCardReadBuf[SDCARD_BUFFER_SIZE];
int SDCardResetCount = 1;

// *****
```

**For SD Card Driver
The Geometry information
is necessary for operation**

✂ Lab18 Manage SPI SD card without File System

- Implement **Transfer Callback** for **SD Card Driver** Read and Write.

```
*/
APP_DRV_SDCARD_DATA app_drv_sdcardData;

// *****
// *****
// Section: Application Callback Functions
// *****
// *****
void SDCardTransferHandler( SYS_MEDIA_BLOCK_EVENT event,
                           SYS_MEDIA_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context )
{
    switch( event )
    {
        case SYS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE:
            if (commandHandle == SDCardReadHandle)
            { SDCardReadComplete = true; }
            else if(commandHandle == SDCardWriteHandle)
            { SDCardWriteComplete = true; }
            break;

        case SYS_MEDIA_EVENT_BLOCK_COMMAND_ERROR:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
            break;
    }
}

// *****
// *****
// Section: Application Local Functions
// *****
// *****
```

SD Card Driver Transfer Callback

⚙️ Lab18 Manage SPI SD card without File System

- Open Driver, register callback in state **INIT** and handling **ERROR**.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT:
            SDCardHandle = DRV_SDSPI_Open(DRV_SDSPI_INDEX_0, DRV_IO_INTENT_READWRITE);
            if(SDCardHandle == DRV_HANDLE_INVALID)
            {
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
                break;
            }

```

Open SD Card Driver to get Handle

```
        DRV_SDSPI_EventHandlerSet(SDCardHandle, SDCardTransferHandler, 0);

```

Register Transfer Callback

Keep this line

```
        app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WAIT_ATTACH;
        break;

```

```
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
            break;

```

...

Handling Error state

```
        case APP_DRV_SDCARD_STATE_ERROR:
            USART1printf("DRV_SDCARD error!\r\n");

```

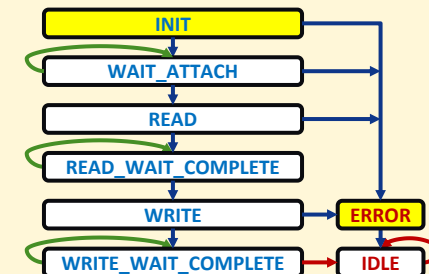
Keep this line

```
        app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;
        break;

```

```
        case APP_DRV_SDCARD_STATE_IDLE:
            break;
        ...

```



⚙️ Lab18 Manage SPI SD card without File System

- Use the polling method to detect SD Card in state **WAIT_ATTACH**.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT: ...
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH:
            if( DRV_SDSPI_IsAttached(SDCardHandle) == false )
                break;

            USART1printf("\r\nSDCard attached!\r\n");

            SDCardGeometry = DRV_SDSPI_GeometryGet(SDCardHandle);
            if (SDCardGeometry == NULL)
            {
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
                break;
            }
            SDCardReadBlocks  = (SDCARD_BUFFER_SIZE / \
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_READ_ENTRY].blockSize);
            SDCardWriteBlocks = (SDCARD_BUFFER_SIZE / \
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_WRITE_ENTRY].blockSize);
            SDCardEraseBlocks = (SDCARD_BUFFER_SIZE / \
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);

            Keep this line app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
            break;

        case APP_DRV_SDCARD_STATE_READ:
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_SDCARD_STATE_WRITE: ...
        ...
    }
}
```

Using Polling method to Check SD Card attach status

Get SD Card Geometry

```
graph TD
    INIT[INIT] --> WAIT_ATTACH[WAIT_ATTACH]
    WAIT_ATTACH --> READ[READ]
    READ --> READ_WAIT_COMPLETE[READ_WAIT_COMPLETE]
    READ_WAIT_COMPLETE --> WRITE[WRITE]
    WRITE --> WRITE_WAIT_COMPLETE[WRITE_WAIT_COMPLETE]
    WRITE_WAIT_COMPLETE --> IDLE[IDLE]
    WAIT_ATTACH --> ERROR[ERROR]
    READ --> ERROR
    WRITE --> ERROR
    WRITE_WAIT_COMPLETE --> ERROR
    ERROR --> IDLE
```

✖ Lab18 Manage SPI SD card without File System

- Read **SDCardReadBlocks** from address **SDCARD_BLOCK_START**.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT: ...
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH:
            ...
            SDCardEraseBlocks = (SDCARD_BUFFER_SIZE / \
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);

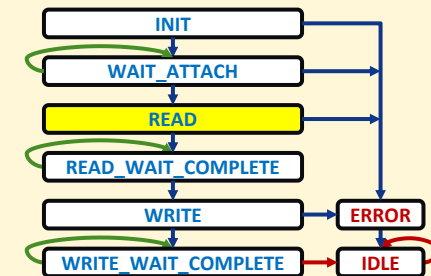
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
            break;
```

Read one Block from SD Card

```
        case APP_DRV_SDCARD_STATE_READ:
            DRV_SDSPI_AsyncRead(SDCardHandle, &SDCardReadHandle,
                                SDCardReadBuf, SDCARD_BLOCK_START, SDCardReadBlocks);
            if(SDCardReadHandle == DRV_SDSPI_COMMAND_HANDLE_INVALID)
            {
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
                break;
            }
```

Keep this line app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE;
break;

```
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_SDCARD_STATE_WRITE: ...
        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE: ...
        case APP_DRV_SDCARD_STATE_ERROR: ...
        case APP_DRV_SDCARD_STATE_IDLE: ...
        ...
```



✂ Lab18 Manage SPI SD card without File System

- Increase "**Count**" of read content or **output SD Card Geometry** if it's first time to write.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT: ...
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH: ...
        case APP_DRV_SDCARD_STATE_READ: ...
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE:
            if(SDCardReadComplete == true)
            {
```

```
                SDCardReadComplete = false;
```

Output current count and increase it

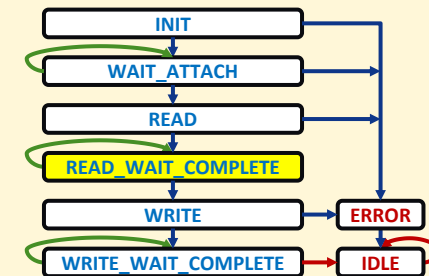
```
                if( !strncmp((char*)SDCardReadBuf, "Count SDCard: ", 14) )
                {
                    USART1printf("%s\r\n", SDCardReadBuf);
                    sscanf(SDCardReadBuf, "Count SDCard: %d", &SDCardResetCount);
                    SDCardResetCount = (SDCardResetCount+1)%10;
                }
            }
            else
            {
```

Output Geometry if nothing to be wrote

```
                USART1printf("Geometry SDCard: Buff(%d), R(%d), W(%d), E(%d)\r\n",
                             SDCARD_BUFFER_SIZE,
                             SDCardReadBlocks, SDCardWriteBlocks, SDCardEraseBlocks);
            }
        }
    }
    app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE;
}
break;
```

Keep this line

```
case APP_DRV_SDCARD_STATE_WRITE: ...
...
```



⚙️ Lab18 Manage SPI SD card without File System

- Write back content to target address **SDCARD_BLOCK_START**.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT: ...
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH: ...
        case APP_DRV_SDCARD_STATE_READ: ...
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE:
            ...
            break;

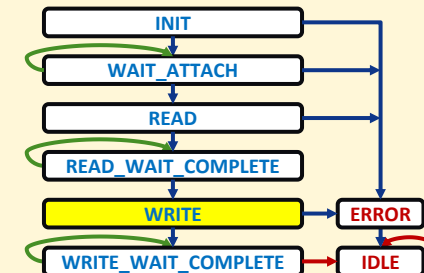
        case APP_DRV_SDCARD_STATE_WRITE:
            sprintf( SDCardWriteBuf, "Count SDCard: %d", SDCardResetCount);
            DRV_SDSPI_AsyncWrite(SDCardHandle, &SDCardWriteHandle,
                                SDCardWriteBuf, SDCARD_BLOCK_START, SDCardWriteBlocks);

            if(SDCardWriteHandle == DRV_SDSPI_COMMAND_HANDLE_INVALID)
            {
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
            }
            else
            {
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE;
            }
            break;

        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE: ...
        case APP_DRV_SDCARD_STATE_ERROR: ...
        case APP_DRV_SDCARD_STATE_IDLE:
            break
        ...
    }
}
```

Write count to target Block of SD Card

Keep this line



⚙️ Lab18 Manage SPI SD card without File System

- Close **handle** while write completed and go state **IDLE** forever.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT: ...
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH: ...
        case APP_DRV_SDCARD_STATE_READ: ...
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_SDCARD_STATE_WRITE:
            ...
        }
        else
        {
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE;
        }
        break;

        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE:
            if(SDCardWriteComplete == true)
            {
                SDCardWriteComplete = false;

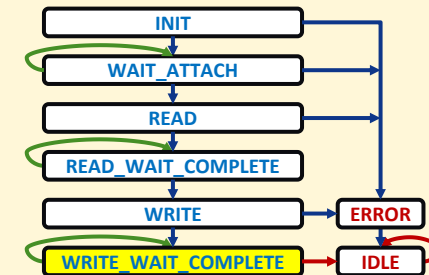
                DRV_SDSPI_Close( SDCardHandle );

                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;
            }
            break;

        case APP_DRV_SDCARD_STATE_ERROR: ...
        case APP_DRV_SDCARD_STATE_IDLE: ...
        ...
    }
```

Go state IDLE after finished

Keep this line



⚙️ Lab18 Manage SPI SD card without File System

- **Put in SD Card first !!!**
- Compiler and programming your project to EVB.
- Geometry will output while first time to reboot because there has nothing wrote at **Block Start Address 0x400000**.
- The **“Count”** will increase while each time you to press **Reset** button on the EVB.

```
COM3 - Tera Term VT
File Edit Setup Control Window Help

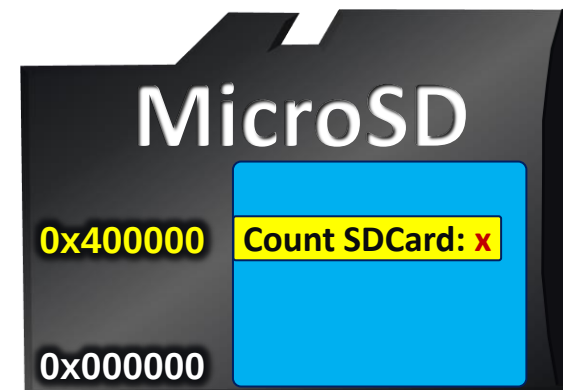
1st Boot → ...
SDCard attached!
Geometry SDCard: Buffer(4096), R(8), W(8), E(8)

Reset → ...
SDCard attached!
Count SDCard: 1

Reset → ...
SDCard attached!
Count SDCard: 2

Reset → ...
SDCard attached!
Count SDCard: 3
```

Block Size is **512** bytes
Store Block is **0x2000**
Start Address is **0x40000**



🔧 Lab18 Manage SPI SD card without File System

• Discuss

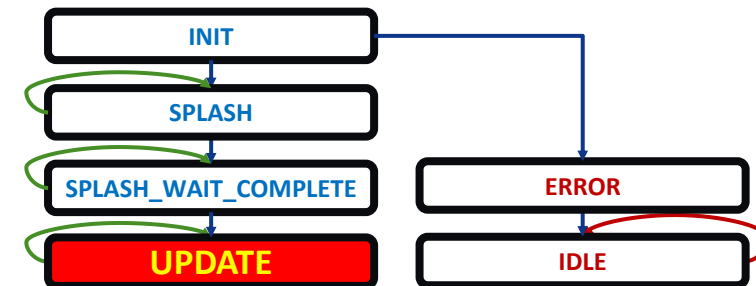
- **Remove SD Card firstly** and press Reset button on the EVB then put SD Card in.
Why cannot to detect SD Card successfully even we to use **Polling Method** for SD Card Detect ?
- The **SERCOM4-SPI** to share with OLED and we to use **blocking transfer** for GUI update in SPI1 task.
It will affect the SD Card Driver to **miss polling** of SD Card detection.
- Let's **add 200ms** delay between each update in state **UPDATE** of SPI1 task.

```
void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
        ...
        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == true )
            {
                LCM_Clear();

                SYS_TIME_DelayMS(200, &SplashTimerHandle);

                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;
            }
            break;
        case APP_DRV_SPI1_STATE_UPDATE:
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == false )
                break;
            SYS_TIME_DelayMS(200, &SplashTimerHandle);
            sprintf( OutString, "LCM: Hello World" );
            ...
    }
}
```

app_drv_spi1.c



MEMORY Driver Library

Harmony Driver

MEMORY Driver Library

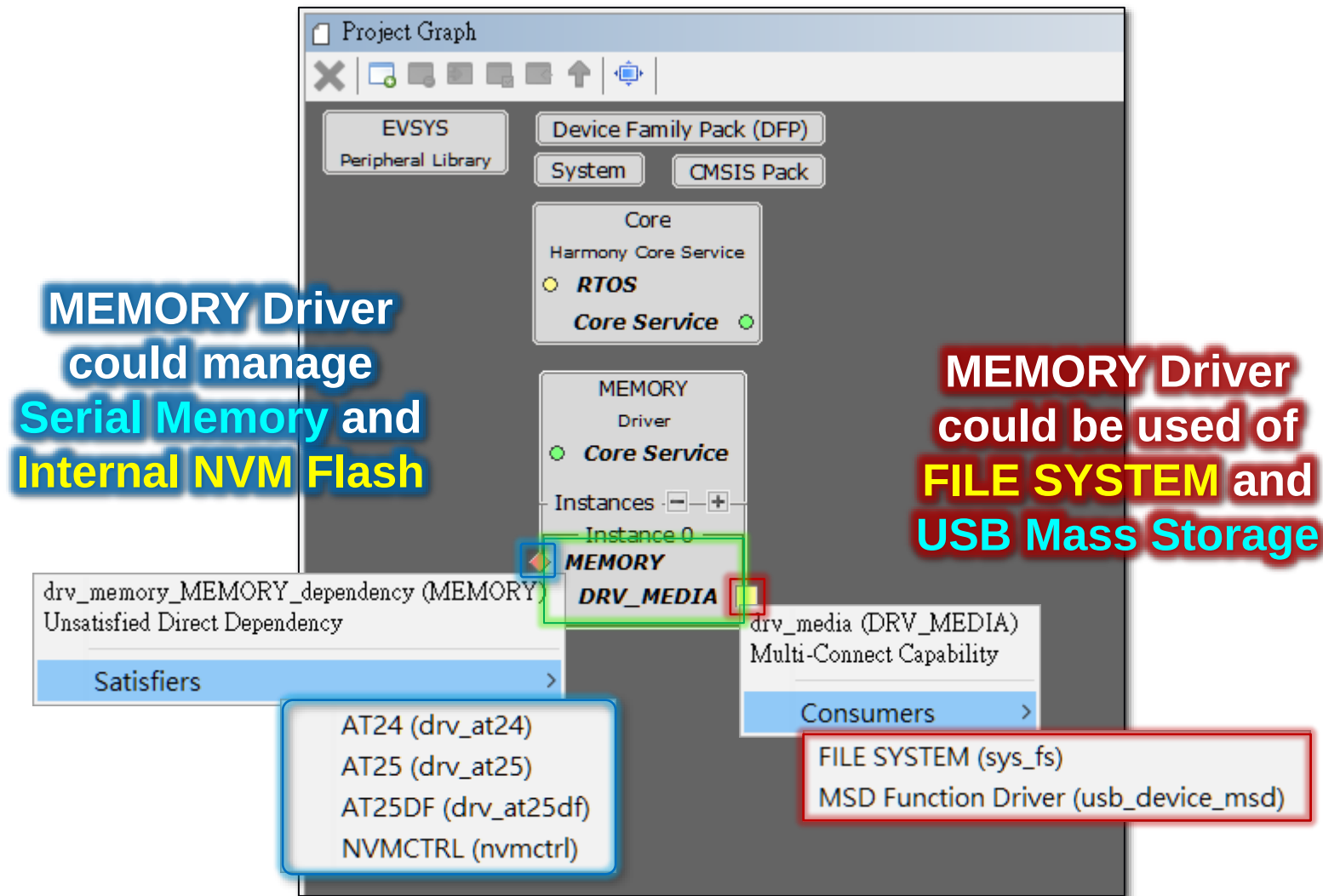
- **Introduction**

The Memory driver is a multi-client multi-instance buffer model-based block driver interface which can be used to communicate with various media devices.

- **Key Features of Memory Driver:**

- It works in both Asynchronous and Synchronous modes of operation.
- It works in both Bare Metal and RTOS environment.
- It can be used to communicate with Different Media's via common Media Interface.
- Media which can be accessed directly by Peripheral Libraries (NVM).
- Media which can be accessed by protocol drivers which in-turn connect to respective peripheral Libraries (SST26 Driver --> QSPI peripheral library)

MEMORY Driver Library



MEMORY Driver Library

API functions of the MEMORY Driver library.

- **System Functions**

DRV_MEMORY_Initialize

Initializes the Memory instance for the specified driver index

DRV_MEMORY_Status

Gets the current status of the Memory driver module.

DRV_MEMORY_Tasks

Maintains the Memory driver's internal state machine.

- **Core Client Functions**

DRV_MEMORY_Open

Opens the specified Memory driver instance and returns a handle to it

DRV_MEMORY_Close

Closes an opened-instance of the Memory driver

DRV_MEMORY_TransferHandlerSet

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

- **Block Operation Functions**

DRV_MEMORY_AsyncErase

Erase the specified number of memory blocks from the specified block start. Each block is equal to sector size of the memory device attached.

DRV_MEMORY_AsyncEraseWrite

Erase and Write blocks of data in the sectors where the block start belongs.

DRV_MEMORY_AsyncRead

Reads nblocks of data from the specified block start.

DRV_MEMORY_AsyncWrite

Writes nblocks of data starting at the specified block start.

DRV_MEMORY_SyncErase

Erase the specified number of memory blocks from the specified block start. Each block is equal to sector size of the memory device attached.

DRV_MEMORY_Erase

This routine provides interface to the file system to perform a media erase operation in synchronous mode of the Memory driver.

DRV_MEMORY_SyncEraseWrite

Erase and Write blocks of data in the sectors where the block start belongs.

MEMORY Driver Library

API functions of the MEMORY Driver library. (cont.)

- **Block Operation Functions (cont.)**

DRV_MEMORY_EraseWrite	This routine provides interface to the file system to perform a media erase-write operation in synchronous mode of the Memory driver.
DRV_MEMORY_SyncRead	Reads nblock of data from the specified block start.
DRV_MEMORY_Read	This routine provides interface to the file system to perform a media read operation in synchronous mode of the Memory driver.
DRV_MEMORY_SyncWrite	Writes nblock of data starting at the specified block start.
DRV_MEMORY_Write	This routine provides interface to the file system to perform a media write operation in synchronous mode of the Memory driver.
DRV_MEMORY_CommandStatusGet	Gets the current status of the command.
DRV_MEMORY_TransferStatusGet	Gets the current status of the transfer request on attached device.

- **Media Interface Functions**

DRV_MEMORY_AddressGet	Returns the Memory media start address
DRV_MEMORY_GeometryGet	Returns the geometry of the memory device.
DRV_MEMORY_IsAttached	Returns the physical attach status of the Media.
DRV_MEMORY_IsWriteProtected	Returns the write protect status of the Memory.

MEMORY Driver Library

API functions of the MEMORY Driver library. (cont.)

- Data Types and Constants

DRV_MEMORY_COMMAND_HANDLE	Handle to identify commands queued in the driver.
DRV_MEMORY_COMMAND_HANDLE_INVALID	This value defines the Memory Driver's Invalid Command Handle.
DRV_MEMORY_COMMAND_STATUS	Memory Driver command Status
DRV_MEMORY_DEVICE_INTERFACE	Memory Device API.
DRV_MEMORY_DEVICE_OPEN	Function pointer typedef to open the attached media
DRV_MEMORY_DEVICE_CLOSE	Function pointer typedef to close the attached media
DRV_MEMORY_DEVICE_SECTOR_ERASE	Function pointer typedef to erase a sector from attached media
DRV_MEMORY_DEVICE_STATUS	Function pointer typedef to get the status of the attached media
DRV_MEMORY_DEVICE_READ	Function pointer typedef to read from the attached media
DRV_MEMORY_DEVICE_PAGE_WRITE	Function pointer typedef to write a page to the attached media
DRV_MEMORY_DEVICE_EVENT_HANDLER_SET	Function pointer typedef to set event handler with attached media
DRV_MEMORY_DEVICE_GEOMETRY_GET	Function pointer typedef to get the Geometry details
DRV_MEMORY_DEVICE_TRANSFER_STATUS_GET	Function pointer typedef to get the transfer Status
DRV_MEMORY_EVENT	Identifies the possible events that can result from a request.
DRV_MEMORY_EVENT_HANDLER	Function pointer typedef for event handler to be sent to attached media
DRV_MEMORY_INIT	Memory Driver Initialization Data
DRV_MEMORY_TRANSFER_HANDLER	Pointer to a Memory Driver Event handler function
MEMORY_DEVICE_GEOMETRY	Memory Device Geometry Table.
MEMORY_DEVICE_TRANSFER_STATUS	Memory Device Transfer Status.

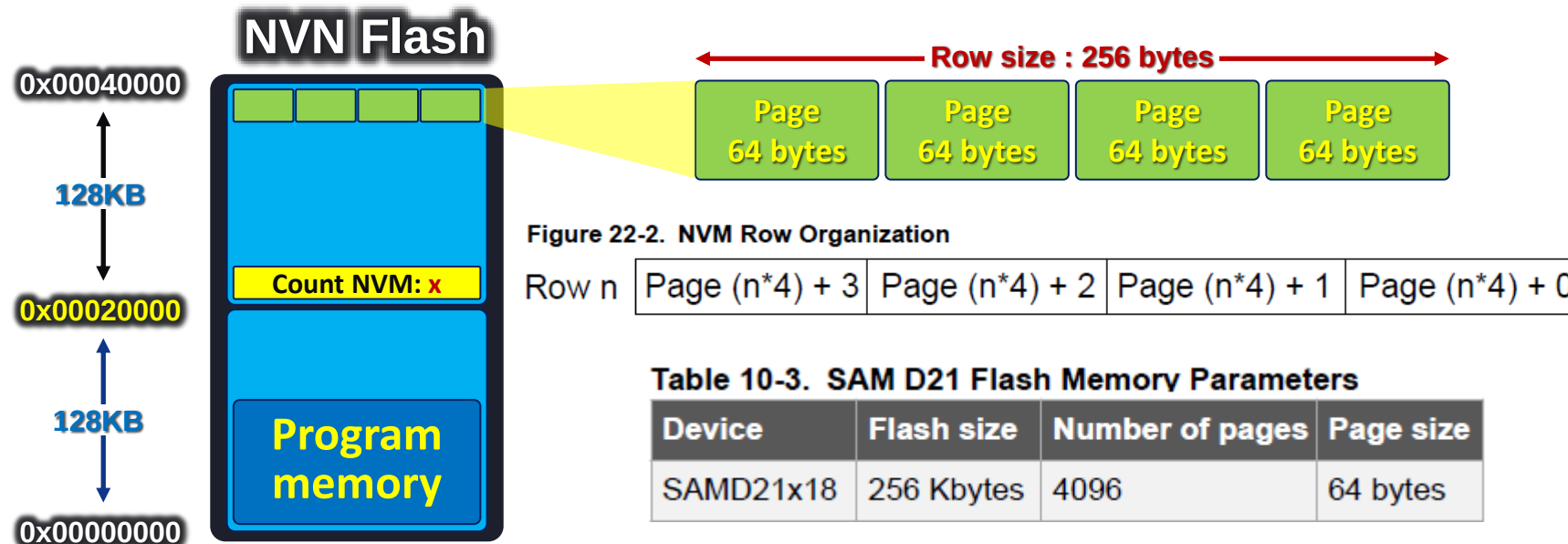
Lab 19

Manage NVM Flash without File System

MEMORY Driver Library

⚙️ Lab19 Manage NVM Flash without File System

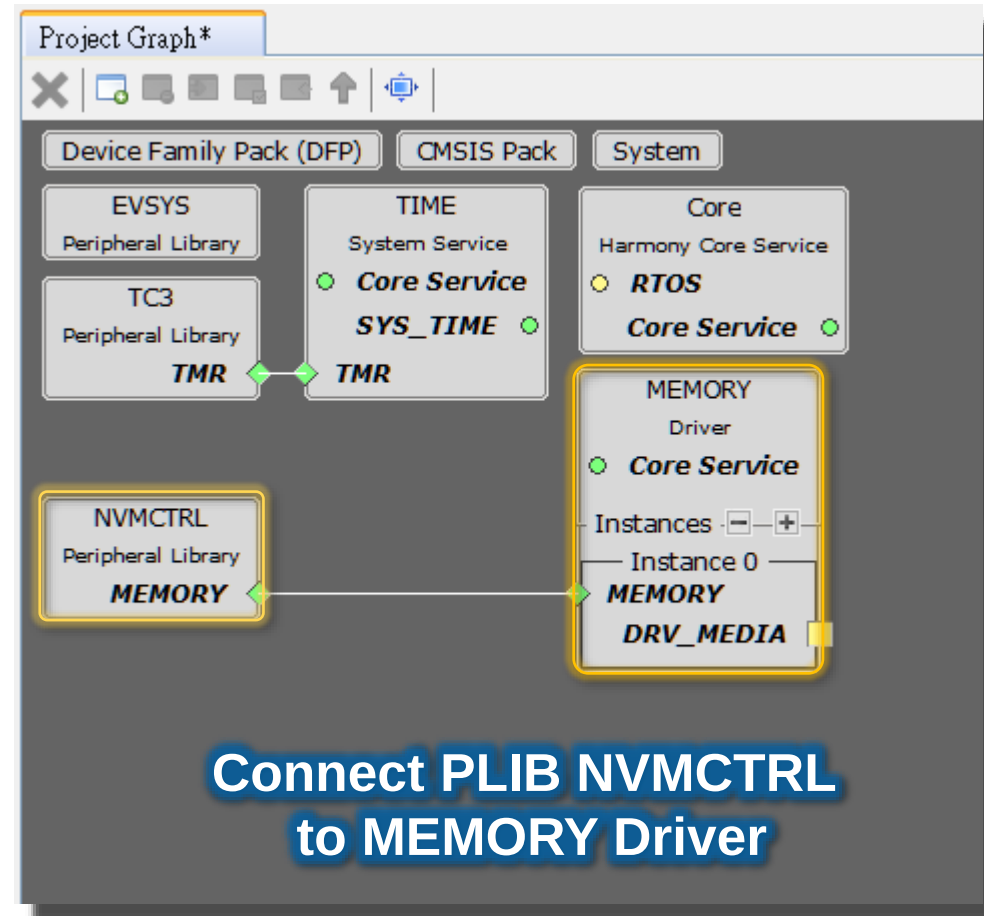
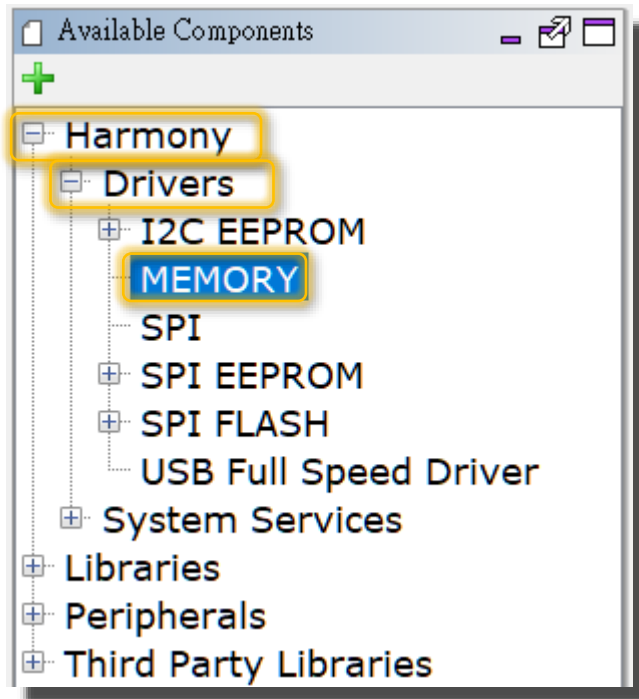
- Add **MEMORY Driver** and choose **PLIB NVMCTRL** as its media.
- The **PLIB NVMCTRL** already exist in project, please just connect it.
- Try store “**Count NVM: #**” to **Start Address 0x20000** of **NVM Flash**, the ‘#’ is count number start from **1**.
- Reset EVB to read “**Count NVM: #**” out and increase the count ‘#’ then store it back to **NVM Flash**.



⚙️ Lab19 Manage NVM Flash without File System

- Add **MEMORY Driver** and connect **PLIB NVMCTRL** to Driver.

Add MEMORY Driver



⚙️ Lab19 Manage NVM Flash without File System

- Default configuration of **MEMORY Driver** and **PLIB NVMCTRL**.

The image displays the Microchip Studio Project Graph and Configuration Options for the MEMORY Driver and NVMCTRL peripheral.

Project Graph: The graph shows the system architecture. The MEMORY Driver (Core Service) is connected to the NVMCTRL (Peripheral Library) via the MEMORY (Peripheral Library). The MEMORY Driver is also connected to the TC3 (Peripheral Library) via the TMR (Peripheral Library). The TC3 is connected to the TIME (System Service) via the TMR (Peripheral Library). The TIME is connected to the Core Service (SYS_TIME) via the TMR (Peripheral Library). The Core Service (SYS_TIME) is connected to the Core Service (RTOS) via the TMR (Peripheral Library). The Core Service (RTOS) is connected to the Core Service (Core Service) via the TMR (Peripheral Library).

Configuration Options - MEMORY:

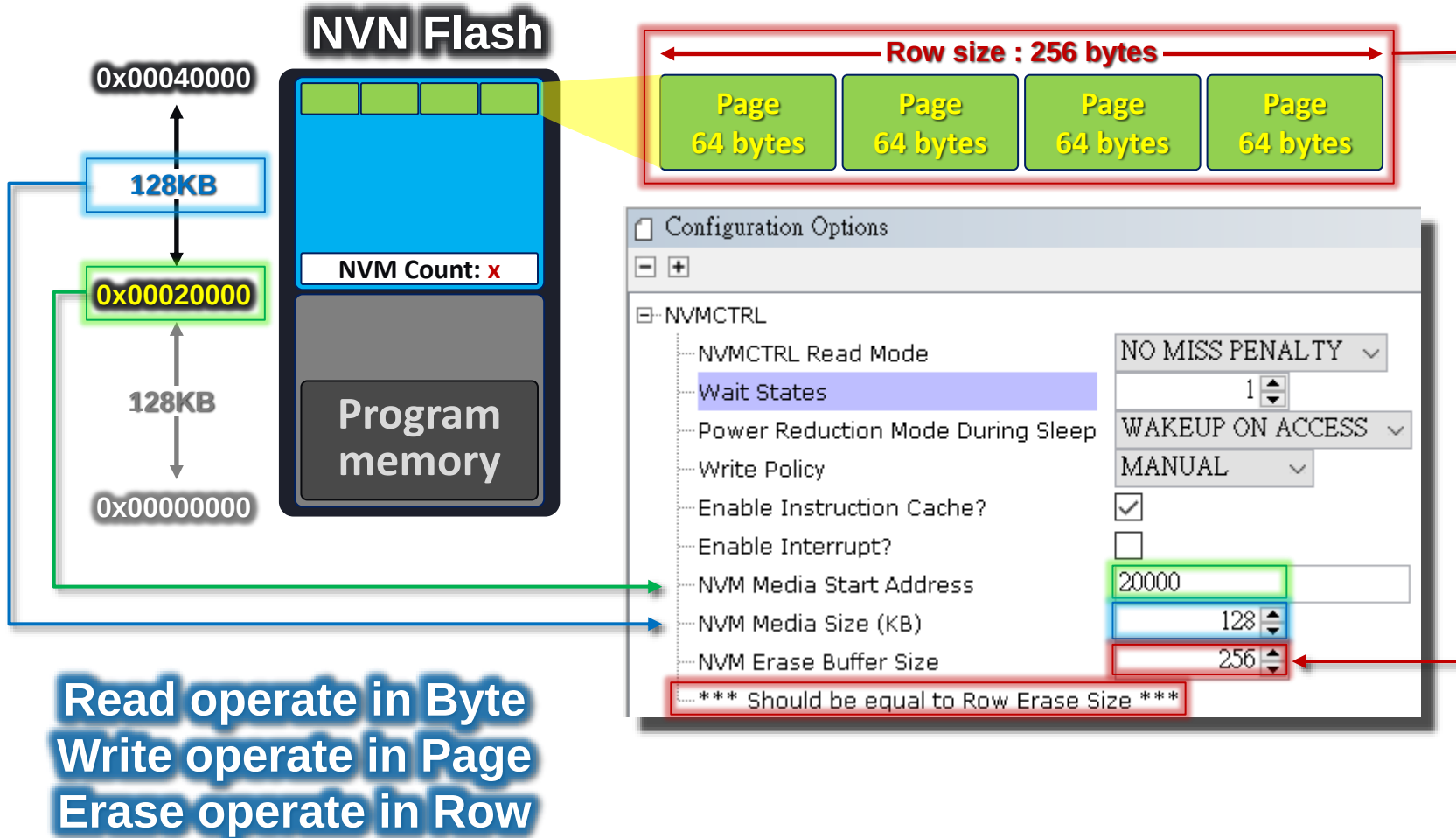
- Number of Clients: 1
- File system Enabled for Memory Driver: ☐
- Buffer Queue Size: 1
- Memory Device Used: NVMCTRL
- *** Configure Memory Device in NVMCTRL Configurations ***

Configuration Options - NVMCTRL:

- NVMCTRL Read Mode: NO MISS PENALTY
- Wait States: 1
- Power Reduction Mode During Sleep: WAKEUP ON ACCESS
- Write Policy: MANUAL
- Enable Instruction Cache? ☒
- Enable Interrupt? ☐
- NVM Media Start Address: 20000
- NVM Media Size (KB): 128
- NVM Erase Buffer Size: 256
- *** Should be equal to Row Erase Size ***

⚙️ Lab19 Manage NVM Flash without File System

- NVM Flash media address and size description.



✖ Lab19 Manage NVM Flash without File System

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**8**” and given the Application Name to “**app_drv_memory**”.

The image shows two windows from a development tool. The left window, titled 'Project Graph*', displays a block diagram of a system. It includes components like EVSYS, TC3, NVMCTRL, TIME, and Core. The Core component is highlighted with a green box. The right window, titled 'Configuration Options', shows the configuration for the Core. A red box highlights the 'Number of Applications' field, which is set to 8. A blue box highlights the 'Application Name' field, which is set to 'app_drv_memory'. A red banner at the top of the right window reads 'Increase Number of Application'. A blue banner at the bottom of the left window reads 'Rename file to “app_drv_memory”'.

Project Graph*

Device Family Pack (DFP) CMSIS Pack System

EVSYS Peripheral Library

TC3 Peripheral Library TMR

NVMCTRL Peripheral Library MEMORY

TIME System Service

Core Service SYS_TIME

Core Harmony Core Service

RTOS

Core Service

MEMORY Driver

Core Service

Instances - +

Instance 0

MEMORY DRV_MEDIA

Rename file to “app_drv_memory”

Configuration Options Help

Increase Number of Application

Core

Generate Harmony Application Files ☒

Application Configuration

Number of Applications 8

Application 0 Configuration

Application 1 Configuration

Application 2 Configuration

Application 3 Configuration

Application 4 Configuration

Application 5 Configuration

Application 6 Configuration

Application 7 Configuration

Enable Application 7 Configuration ☒

Application Name app_drv_memory

**** Application name must be valid C-Lang

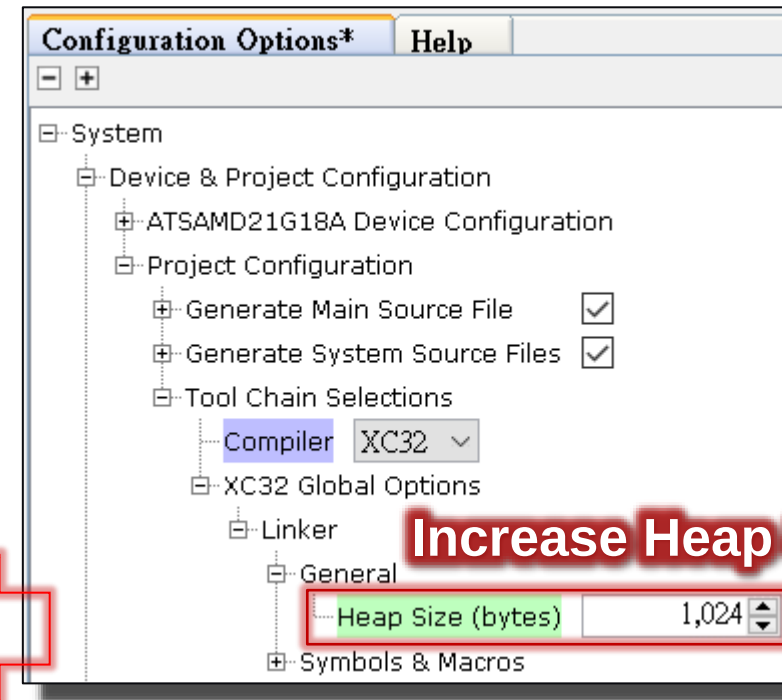
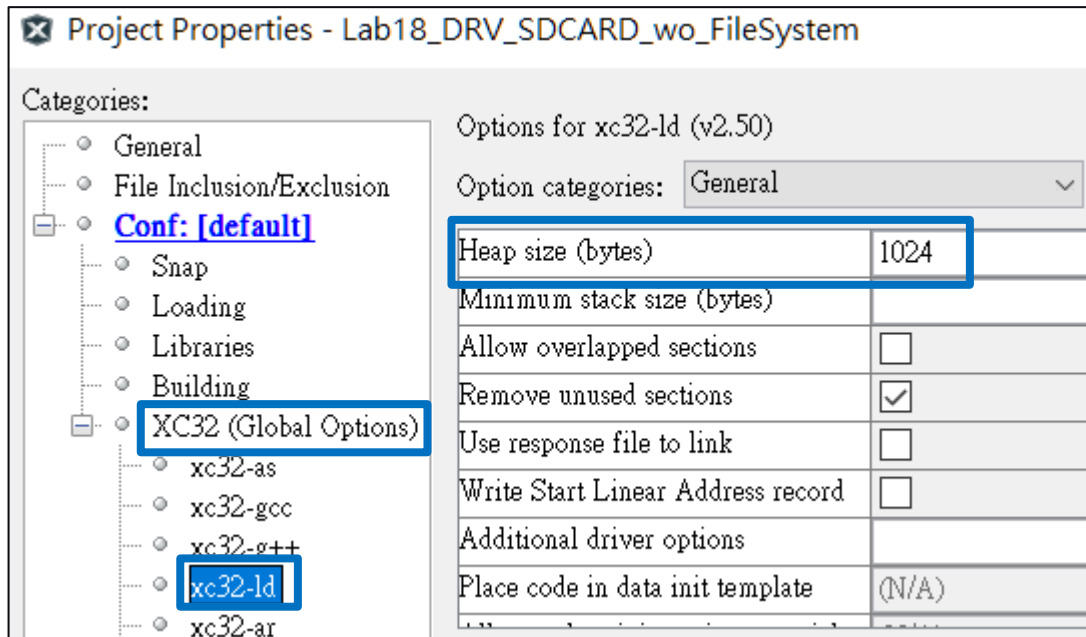
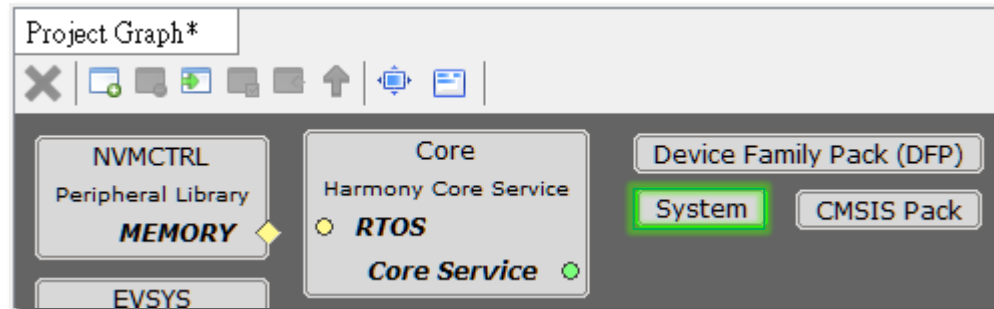
Generate Harmony Driver Common Files ☒

Generate Harmony System Service Common Files ☒

Generate Harmony System Media Files ☒

✖ Lab19 Manage NVM Flash without File System

- Increase **Heap Size** to **1024** bytes in **System** to reserve more memory for string process, if you have not configured this before.

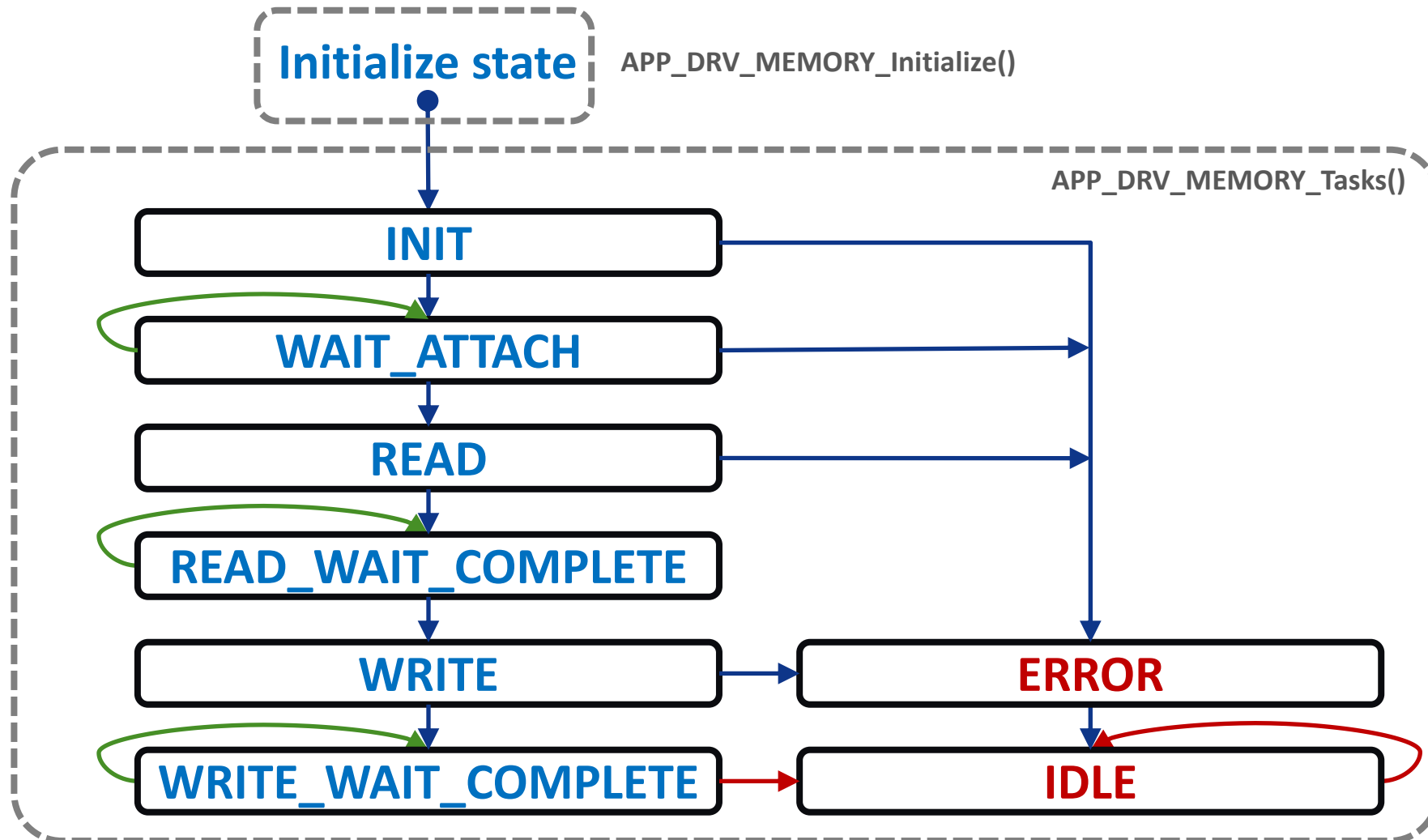


Increase Heap Size to 1024

Harmony Heap setting will overwrite the Heap Size of xc32-ld configuration in Project Properties

⚙️ Lab19 Manage NVM Flash without File System

- The new state machine we wish to complete as below.



⚙️ Lab19 Manage NVM Flash without File System

- Add states in “`app_drv_memory.h`”.

```
#ifndef _APP_DRV_MEMORY_H
#define _APP_DRV_MEMORY_H

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"

typedef enum
{
    APP_DRV_MEMORY_STATE_INIT=0,
    // APP_DRV_MEMORY_STATE_SERVICE_TASKS,
    APP_DRV_MEMORY_STATE_WAIT_ATTACH,
    APP_DRV_MEMORY_STATE_READ,
    APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE,
    APP_DRV_MEMORY_STATE_WRITE,
    APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE,
    APP_DRV_MEMORY_STATE_ERROR,
    APP_DRV_MEMORY_STATE_IDLE,
} APP_DRV_MEMORY_STATES;

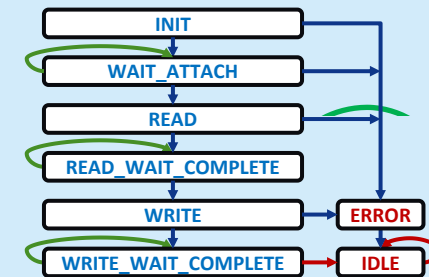
typedef struct
{
    APP_DRV_MEMORY_STATES state;
} APP_DRV_MEMORY_DATA;

void APP_DRV_MEMORY_Initialize ( void );
...
```

Keep this line

Comment out this line

You could remove the
SERVICE_TASKS state
because we will
not use it anymore.



⚙️ Lab19 Manage NVM Flash without File System

- Construct new states flow of task in “`app_drv_memory.c`”.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WAIT_ATTACH;
            break;

        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;
            break;

        case APP_DRV_MEMORY_STATE_READ:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE;
            break;

        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE;
            break;

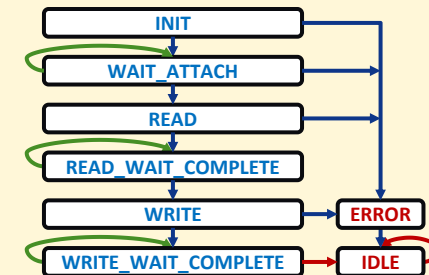
        case APP_DRV_MEMORY_STATE_WRITE:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE;
            break;

        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;
            break;

        case APP_DRV_MEMORY_STATE_ERROR:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;
            break;

        case APP_DRV_MEMORY_STATE_IDLE:
            break;
    }
}
```

Overwrite original example code



✂ Lab19 Manage NVM Flash without File System

- Header include and Global Declarations in “app_drv_memory.c”.

```
Keep this line #include "app_drv_memory.h"
#include "app_drv_usart1.h"
#include "driver/memory/drv_memory.h"
#include <stdio.h>
#include <string.h>

// *****
// *****
// Section: Global Data Definitions
// *****
// *****

#define NVM_BLOCK_START 0
#define NVM_BUFFER_SIZE 256
#define NVM_GEOMETRY_TABLE_READ_ENTRY (0)
#define NVM_GEOMETRY_TABLE_WRITE_ENTRY (1)
#define NVM_GEOMETRY_TABLE_ERASE_ENTRY (2)
SYS_MEDIA_GEOMETRY *MemoryGeometry = NULL;
int MemoryReadBlocks;
int MemoryWriteBlocks;
int MemoryEraseBlocks;

DRV_HANDLE MemoryHandle;
DRV_MEMORY_COMMAND_HANDLE MemoryReadHandle;
DRV_MEMORY_COMMAND_HANDLE MemoryWriteHandle;
volatile int MemoryWriteComplete = false;
volatile int MemoryReadComplete = false;
char MemoryWriteBuf[NVM_BUFFER_SIZE];
char MemoryReadBuf[NVM_BUFFER_SIZE];
int MemoryResetCount = 1;

// *****
```

**For Memory Driver
The Geometry information
is necessary for operation**

⚙️ Lab19 Manage NVM Flash without File System

- Implement **Transfer Callback** for **Memory Driver** Read and Write.

```
*/
APP_DRV_MEMORY_DATA app_drv_memoryData;

// *****
// *****
// Section: Application Callback Functions
// *****
// *****
void MemoryTransferHandler( SYS_MEDIA_BLOCK_EVENT event,
                           SYS_MEDIA_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context )
{
    switch( event )
    {
        case SYS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE:
            if (commandHandle == MemoryReadHandle)
            { MemoryReadComplete = true; }
            else if(commandHandle == MemoryWriteHandle)
            { MemoryWriteComplete = true; }
            break;

        case SYS_MEDIA_EVENT_BLOCK_COMMAND_ERROR:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;
            break;
    }
}

// *****
// *****
// Section: Application Local Functions
// *****
// *****
```

Memory Driver Transfer Callback

⚙️ Lab19 Manage NVM Flash without File System

- Open Driver, register callback in state **INIT** and handling **ERROR**.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT:
            MemoryHandle = DRV_MEMORY_Open(DRV_MEMORY_INDEX_0, DRV_IO_INTENT_READWRITE);
            if(MemoryHandle == DRV_HANDLE_INVALID)
            {
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;
                break;
            }

```

Open Memory Driver to get Handle

```
        DRV_MEMORY_TransferHandlerSet(MemoryHandle, MemoryTransferHandler, 0);

```

Register Transfer Callback

Keep this line

```
        app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WAIT_ATTACH;
        break;

```

```
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;
            break;

```

...

Handling Error state

```
        case APP_DRV_MEMORY_STATE_ERROR:
            USART1printf("DRV_MEMORY error!\r\n");

```

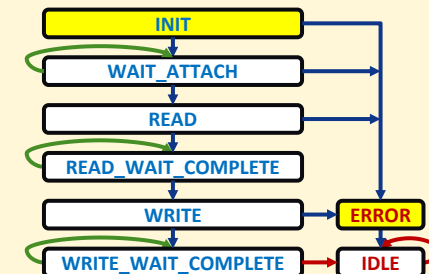
Keep this line

```
        app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;
        break;

```

```
        case APP_DRV_MEMORY_STATE_IDLE:
            break;
    }

```



⚙️ Lab19 Manage NVM Flash without File System

- Get Geometry of **NVM Flash** in state **WAIT_ATTACH**.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT: ...
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:
            if( DRV_MEMORY_IsAttached(MemoryHandle) == false )
                break;

            USART1printf("\r\nNVM attached!\r\n");

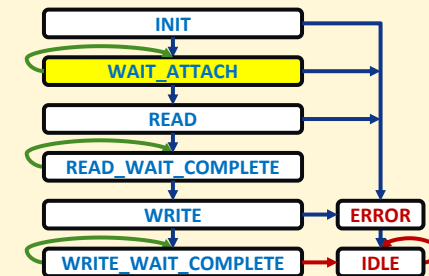
            MemoryGeometry = DRV_MEMORY_GeometryGet(MemoryHandle);
            if (MemoryGeometry == NULL)
            {
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;
                break;
            }
            MemoryReadBlocks = (NVM_BUFFER_SIZE / \
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_READ_ENTRY].blockSize);
            MemoryWriteBlocks = (NVM_BUFFER_SIZE / \
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_WRITE_ENTRY].blockSize);
            MemoryEraseBlocks = (NVM_BUFFER_SIZE / \
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);

            Keep this line app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;
            break;

        case APP_DRV_MEMORY_STATE_READ:
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_MEMORY_STATE_WRITE: ...
        ...
    }
}
```

Using Polling method to
Check Memory attach status

Get NVM Flash Geometry



⚙️ Lab19 Manage NVM Flash without File System

- Read MemoryReadBlocks from address **NVM_BLOCK_START**.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT: ...
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:
            ...
            MemoryEraseBlocks = (NVM_BUFFER_SIZE / \
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);

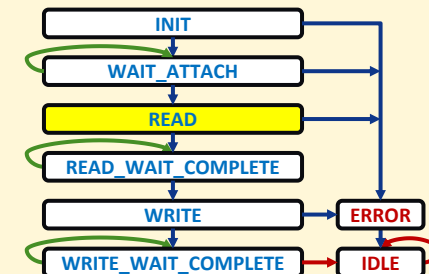
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;
            break;
```

Read one Block from NVM Flash

```
        case APP_DRV_MEMORY_STATE_READ:
            DRV_MEMORY_AsyncRead(MemoryHandle, &MemoryReadHandle,
                                MemoryReadBuf, NVM_BLOCK_START, MemoryReadBlocks);
            if(MemoryReadHandle == DRV_MEMORY_COMMAND_HANDLE_INVALID)
            {
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;
                break;
            }
```

Keep this line app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE;
break;

```
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_MEMORY_STATE_WRITE: ...
        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE: ...
        case APP_DRV_MEMORY_STATE_ERROR: ...
        case APP_DRV_MEMORY_STATE_IDLE: ...
        ...
```



⚙️ Lab19 Manage NVM Flash without File System

- Increase "**Count**" of read content or **output NVM Flash Geometry** if first time to write.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT: ...
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH: ...
        case APP_DRV_MEMORY_STATE_READ: ...
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE:
            if(MemoryReadComplete == true)
```

```
    {
        MemoryReadComplete = false;
```

Output current count and increase it

```
        if( !strncmp((char*)MemoryReadBuf, "Count NVM: ", 11) )
        {
```

```
            USART1printf("%s\r\n", MemoryReadBuf);
            sscanf(MemoryReadBuf, "Count NVM: %d", &MemoryResetCount);
            MemoryResetCount = (MemoryResetCount+1)%10;
        }
```

```
    }
    else
    {
```

Output Geometry if nonthing to be wrote

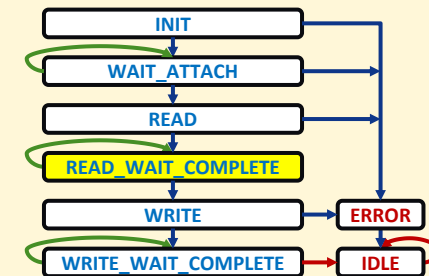
```
        USART1printf("Geometry NVM: Buff(%d), R(%d), W(%d), E(%d)\r\n",
                     NVM_BUFFER_SIZE,
                     MemoryReadBlocks, MemoryWriteBlocks, MemoryEraseBlocks);
    }
```

```
    app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE;
```

Keep this line

```
    break;
```

```
    case APP_DRV_MEMORY_STATE_WRITE: ...
    ...
```



⚙️ Lab19 Manage NVM Flash without File System

- Write back content to target address **NVM_BLOCK_START**.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT: ...
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH: ...
        case APP_DRV_MEMORY_STATE_READ: ...
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE:
            ...
            break;
```

Write count to target Block of SD Card

```
        case APP_DRV_MEMORY_STATE_WRITE:
            sprintf( MemoryWriteBuf, "Count NVM: %d", MemoryResetCount);
            DRV_MEMORY_AsyncEraseWrite(MemoryHandle, &MemoryWriteHandle,
                                       MemoryWriteBuf, NVM_BLOCK_START, MemoryWriteBlocks);
```

```
            if(MemoryWriteHandle == DRV_MEMORY_COMMAND_HANDLE_INVALID)
            {
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;
```

```
            }
            else
            {
```

```
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE;
```

Keep this line

```
            break;
```

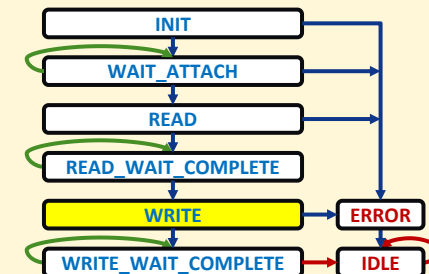
```
        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE: ...
```

```
        case APP_DRV_MEMORY_STATE_ERROR: ...
```

```
        case APP_DRV_MEMORY_STATE_IDLE:
```

```
            break;
```

```
    ...
```



⚙️ Lab19 Manage NVM Flash without File System

- Close **handle** while write completed and go state **IDLE** forever.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT: ...
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH: ...
        case APP_DRV_MEMORY_STATE_READ: ...
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_MEMORY_STATE_WRITE:
            ...
        }
        else
        {
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE;
        }
        break;

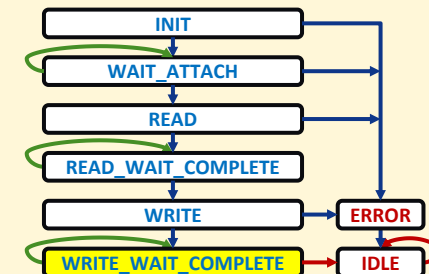
        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE:
            if(MemoryWriteComplete == true)
            {
                MemoryWriteComplete = false;

                DRV_MEMORY_Close( MemoryHandle );

                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;
            }
            break;

        case APP_DRV_MEMORY_STATE_ERROR: ...
        case APP_DRV_MEMORY_STATE_IDLE: ...
        ...
    }
```

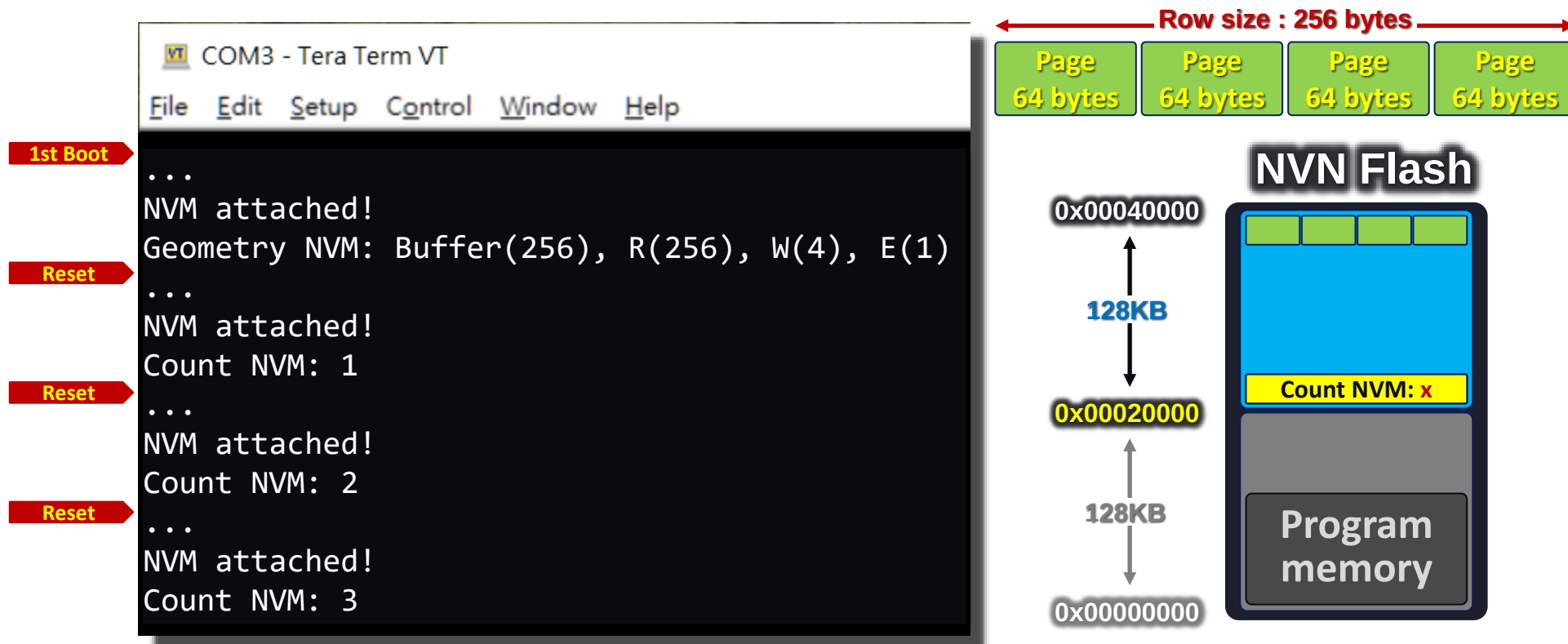
Go state IDLE after finished



Keep this line

⚙️ Lab19 Manage NVM Flash without File System

- Compiler and programming your project to EVB.
- Geometry will output while first time to reboot because there has nothing wrote at **Media Start Address 0x20000**.
- The “**Count**” will increase while each time you to press **Reset** button on the EVB.



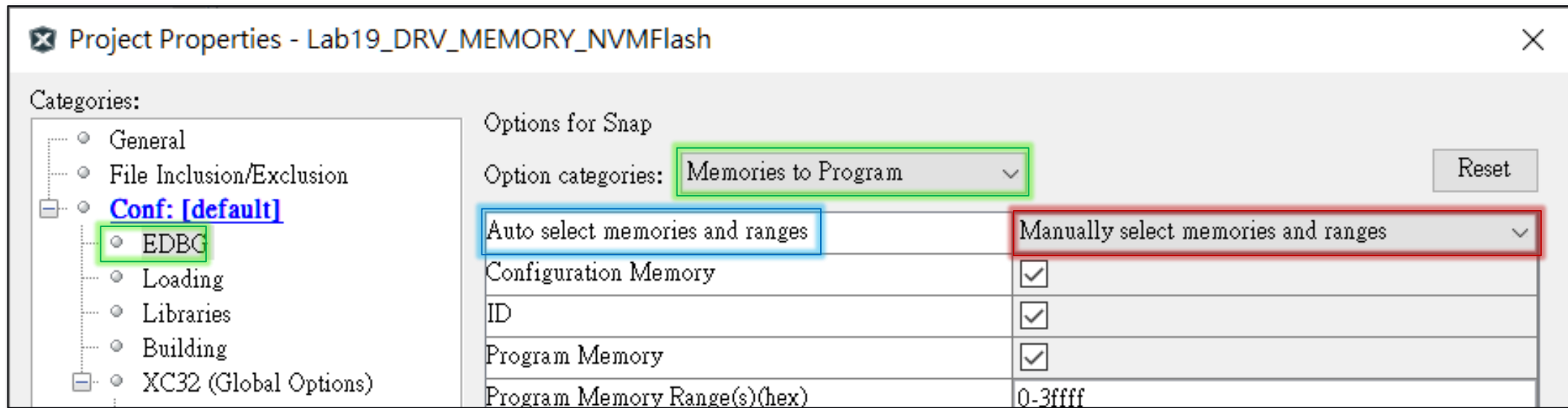
⚡ Lab19 Manage NVM Flash without File System

- Discuss

- How to use MPLAB X IDE to check content of NVM Flash we wrote ?

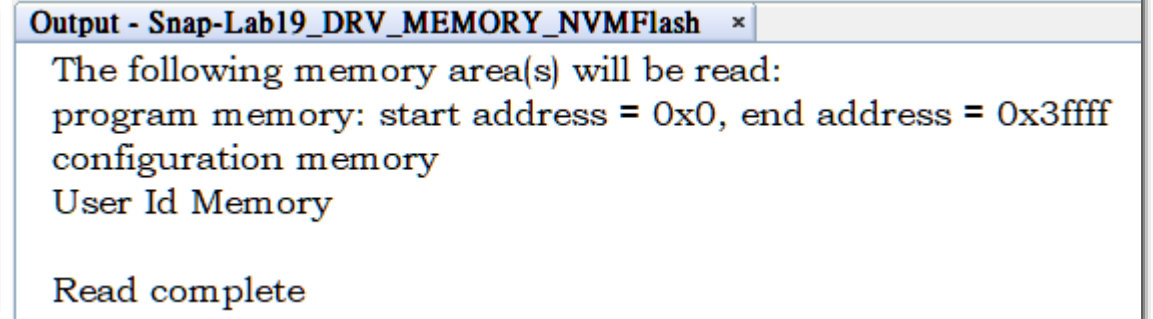
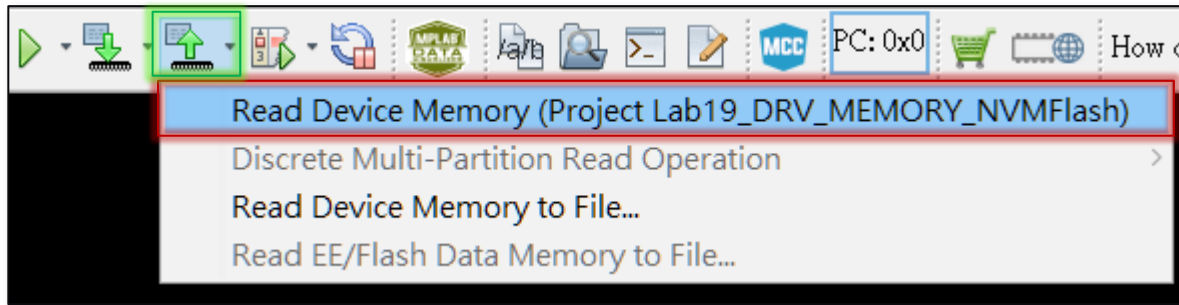
The configuration of **debugger EDBG** in

“**Memories to Program**” -> “**Auto select memories and ranges**” has default choose to “**Allow SNAP to select Memories**”, this option will cause only the areas of MPLAB's memory that **aren't blank** to be programmed. Let's choose to “**Manually select memories and ranges**” for full NVM area.

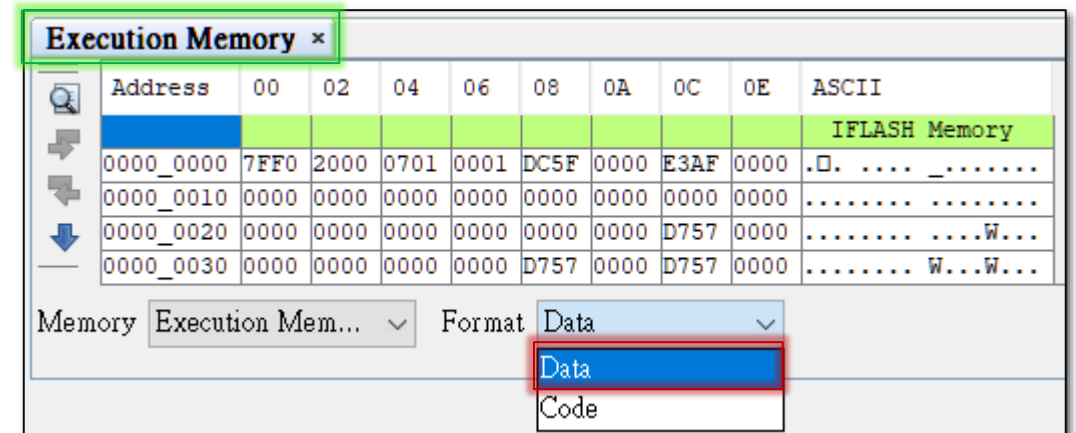
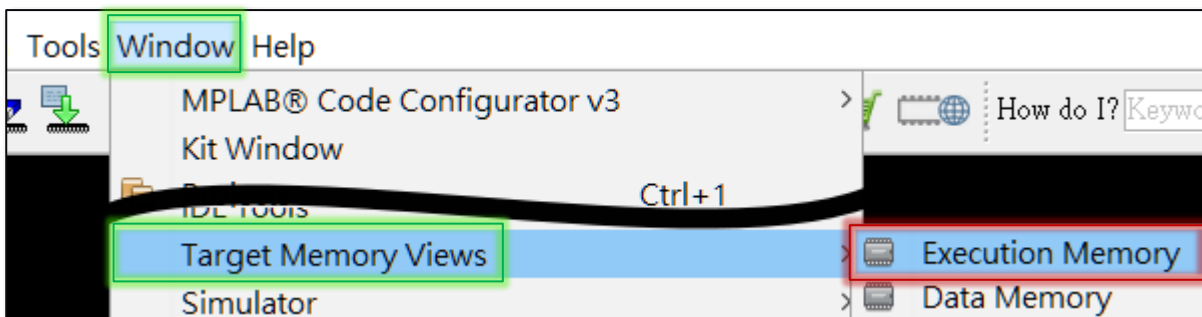


⚙️ Lab19 Manage NVM Flash without File System

- Read whole SAMD21 **NVM Flash** memory by click .

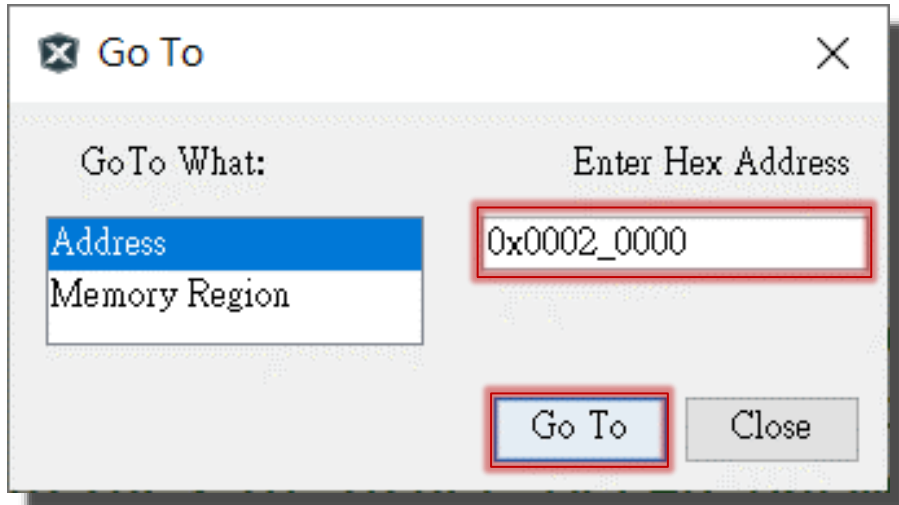


- Open “**Execution Memory**” interface in “**Target Memory Views**”.
- Choose Format to “**Data**” for show raw content of NVM Flash.



⚙️ Lab19 Manage NVM Flash without File System

- Click  and enter Hex Address **0x0002_0000** to find the content we wrote to NVM Flash.



Execution Memory x									
Address	00	02	04	06	08	0A	0C	0E	ASCII
0001_FFE0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
0001_FFF0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
0002_0000	6F43	6E75	2074	564E	3A4D	3520	0000	0000	Count NV M: 5...
0002_0010	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0020	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0030	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0040	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0050	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0060	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0070	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0080	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0090	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00A0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00B0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00C0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00D0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00E0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00F0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0100	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	

The 256 Bytes
We to wrote

Trademarks

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

Thank You !!
