CAE專家教室 SAM2002系列-02

SAMD21 ARM Cortex-M0+ Microcontroller & MCC Harmony

SAM2002 v1.1

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

CAE Taiwan Team
Microchip Technology Inc.

October 30, 2024

SAM2002-02 Index

SAM2002-02 Index

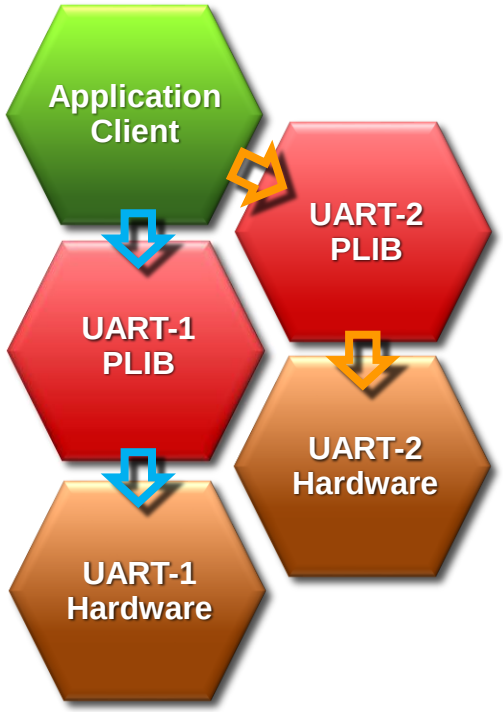
Harmony Drivers Introduction

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);

    ...
}
```

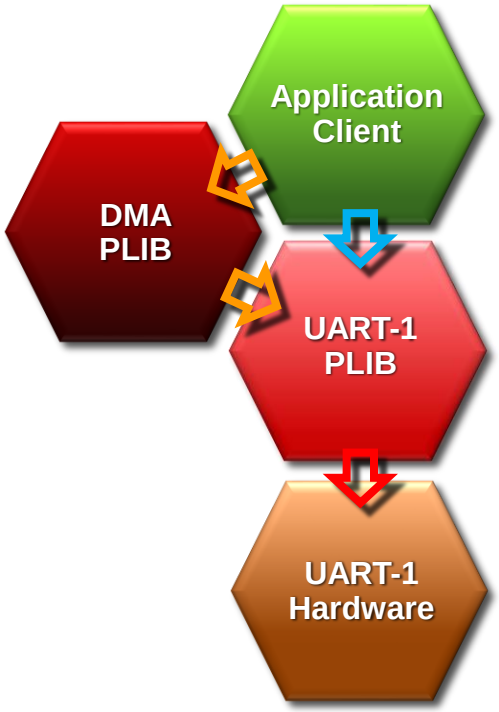


Application needs to be modified for different hardware instance

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);
    ...
}
```

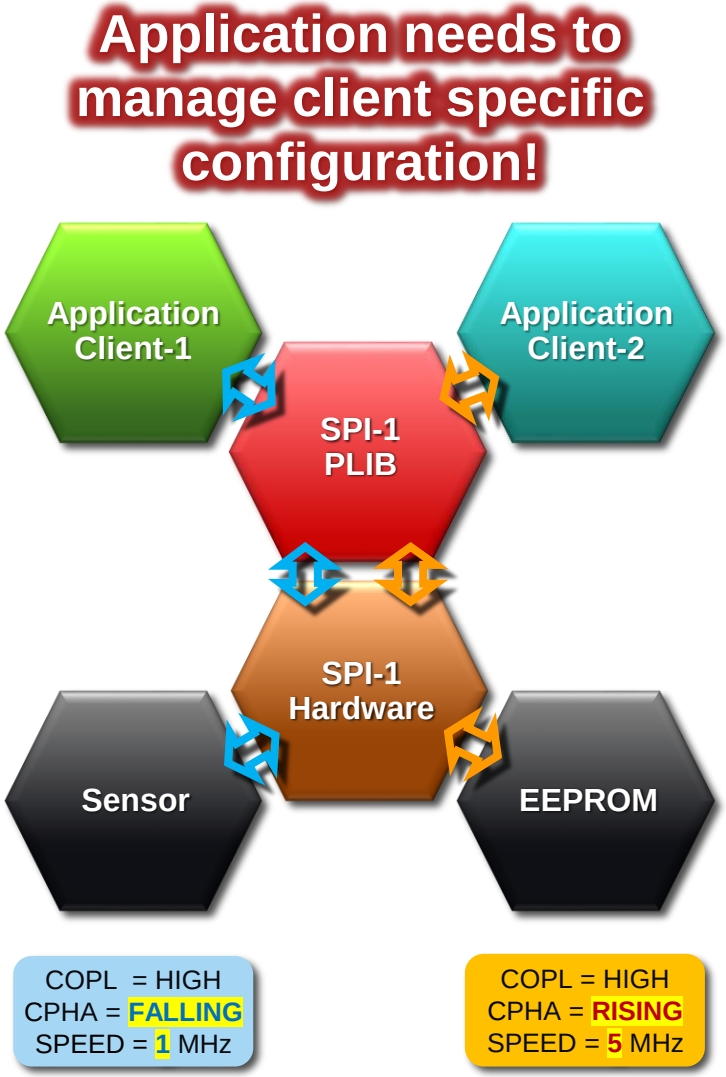


Application needs to manage DMA !

PLIB Limitation

Use Multiple Client

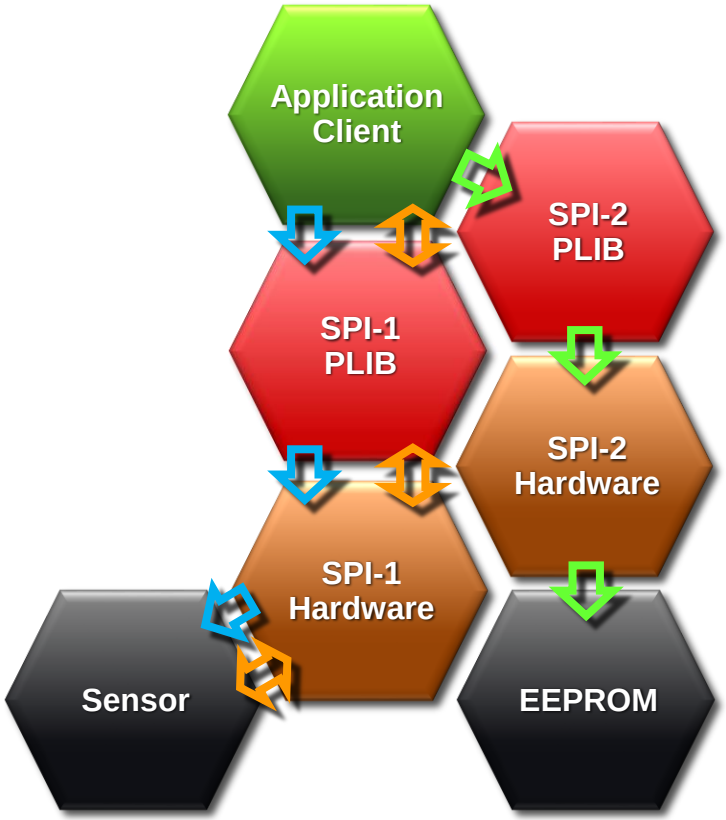
```
void my_app(void)
{
    SPI1_TransferSetup(&sensor_spi_setup, 0);

    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();

    ...
}
```



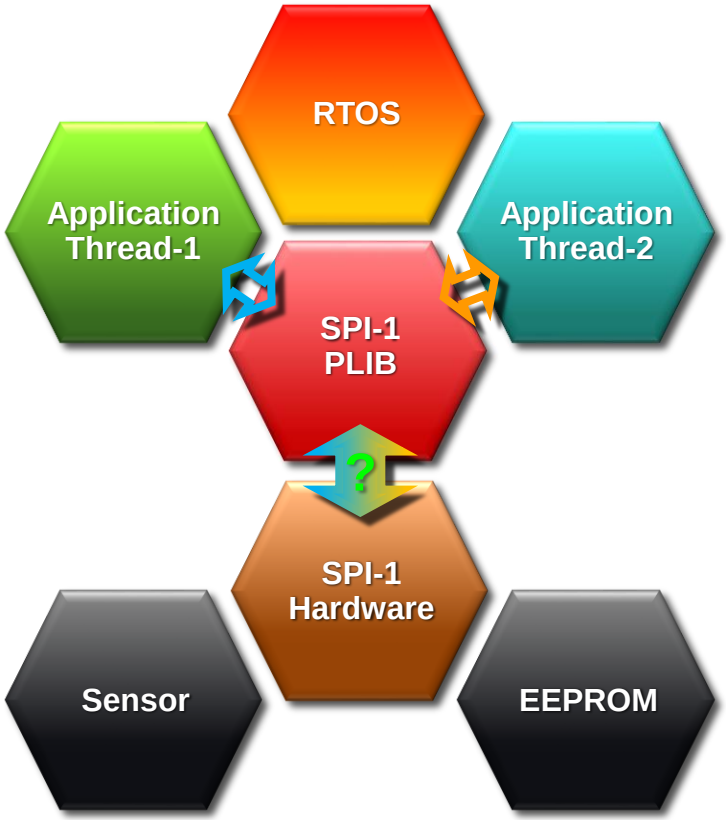
Application needs to arrange transfer request!

MICROCHIP

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!

MICROCHIP

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**

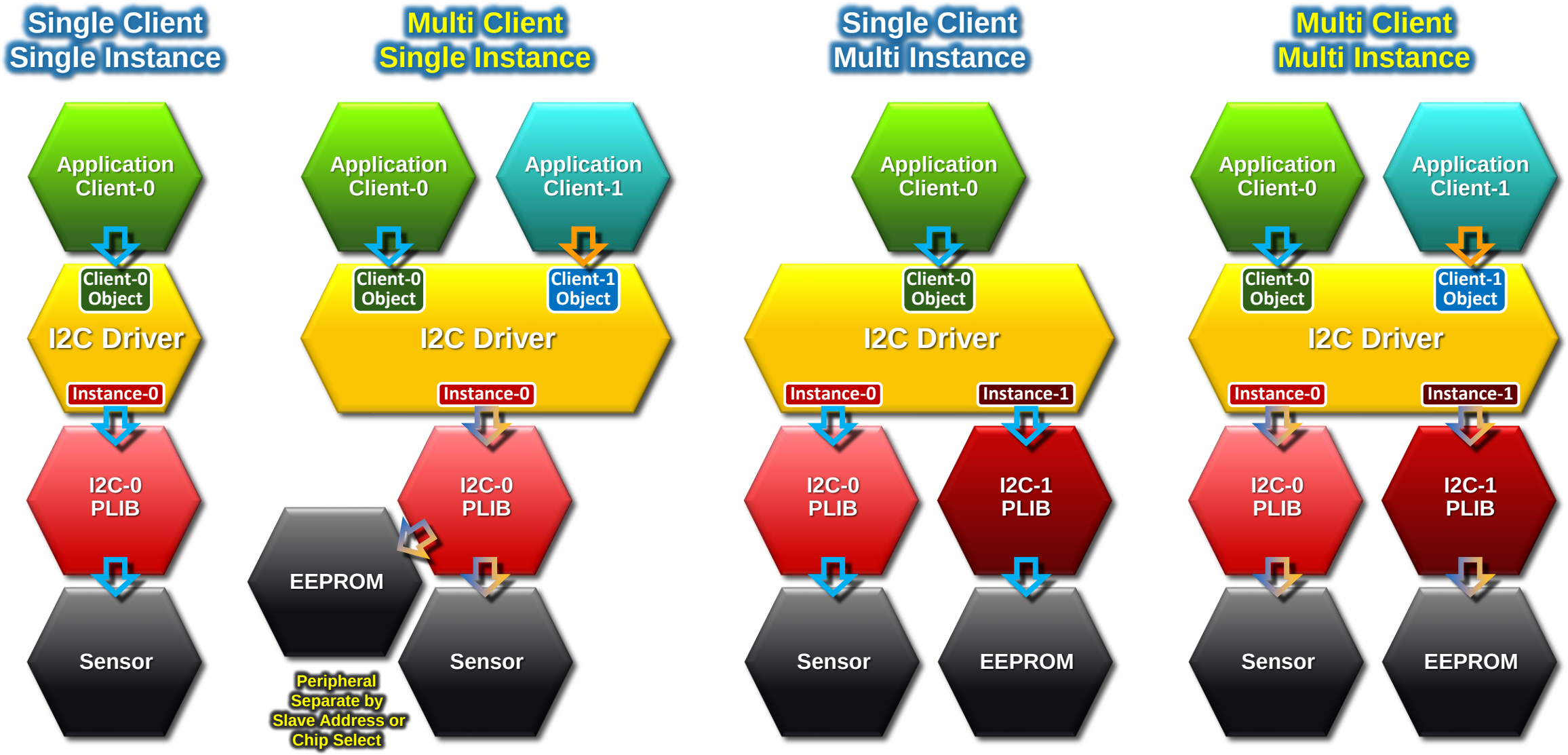
  Driver interface/API remains same with or without DMA transfer mode
- **Support for Multiple Clients**

  Seamlessly handles client specific differences
- **Queue Support**

  Multiple requests can be queued
- **RTOS Ready**

  Thread Safe

Driver Usage Models



MICROCHIP

How to use Driver in Harmony?

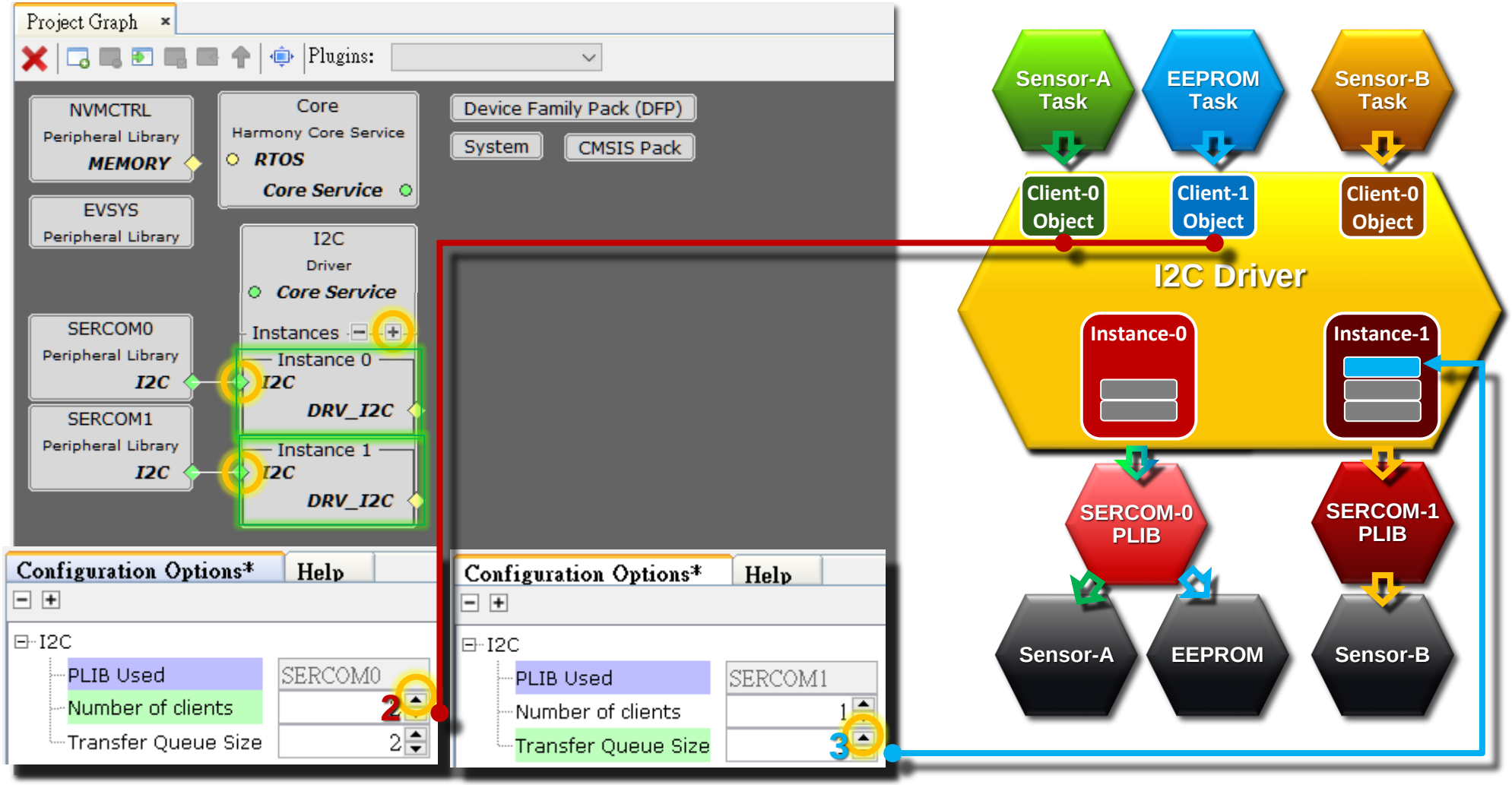
Harmony Drivers Introduction

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

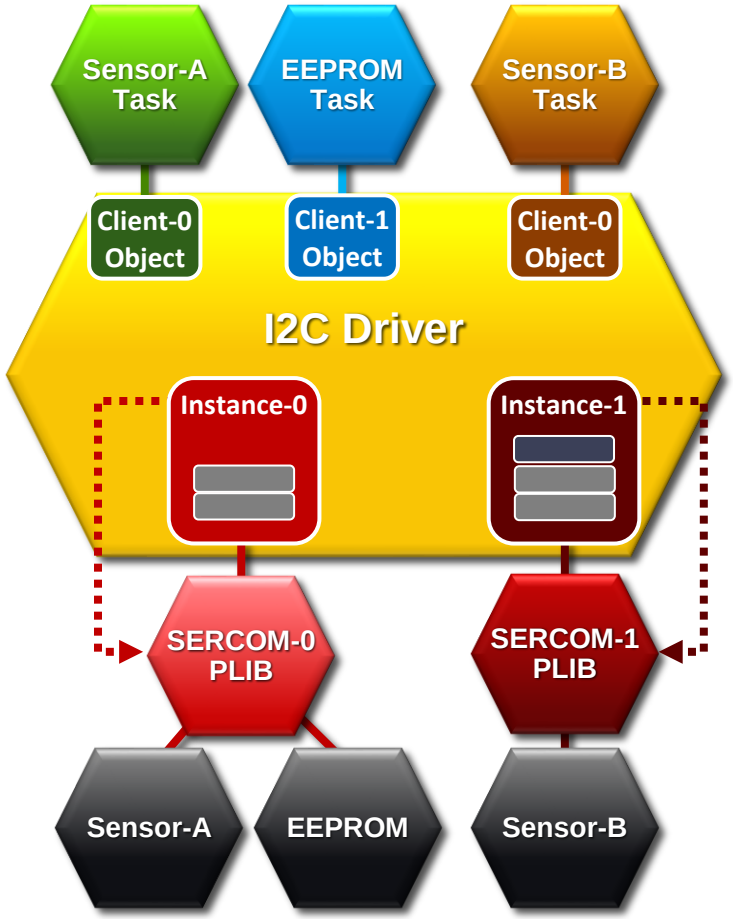
How to use Driver in Harmony?

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();
    ...
}
```



MICROCHIP

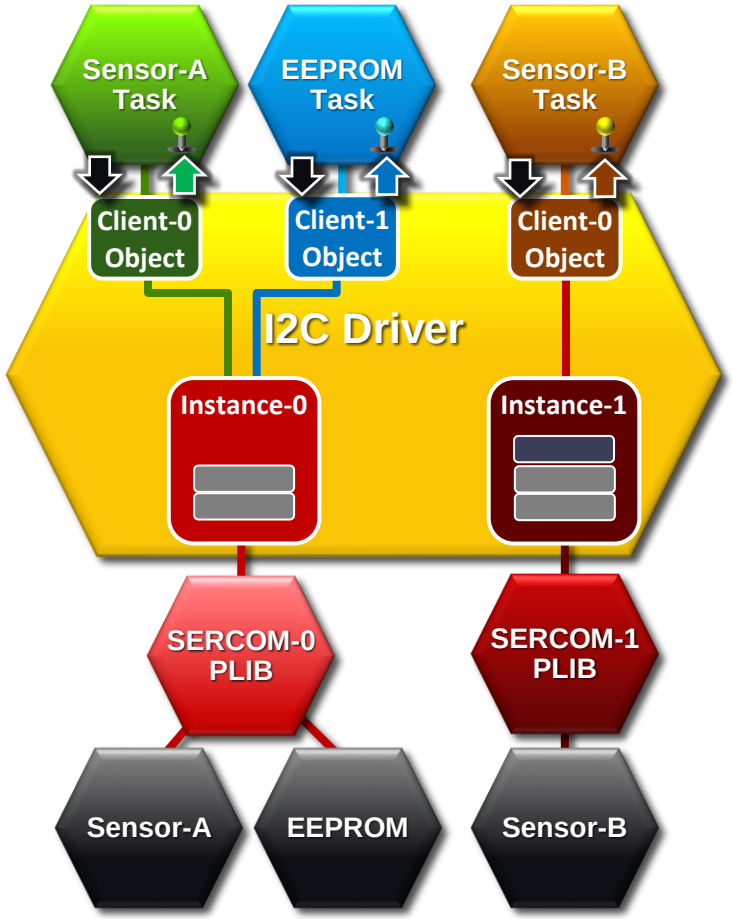
How to use Driver in Harmony?

- 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
    );
}
```



MICROCHIP

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

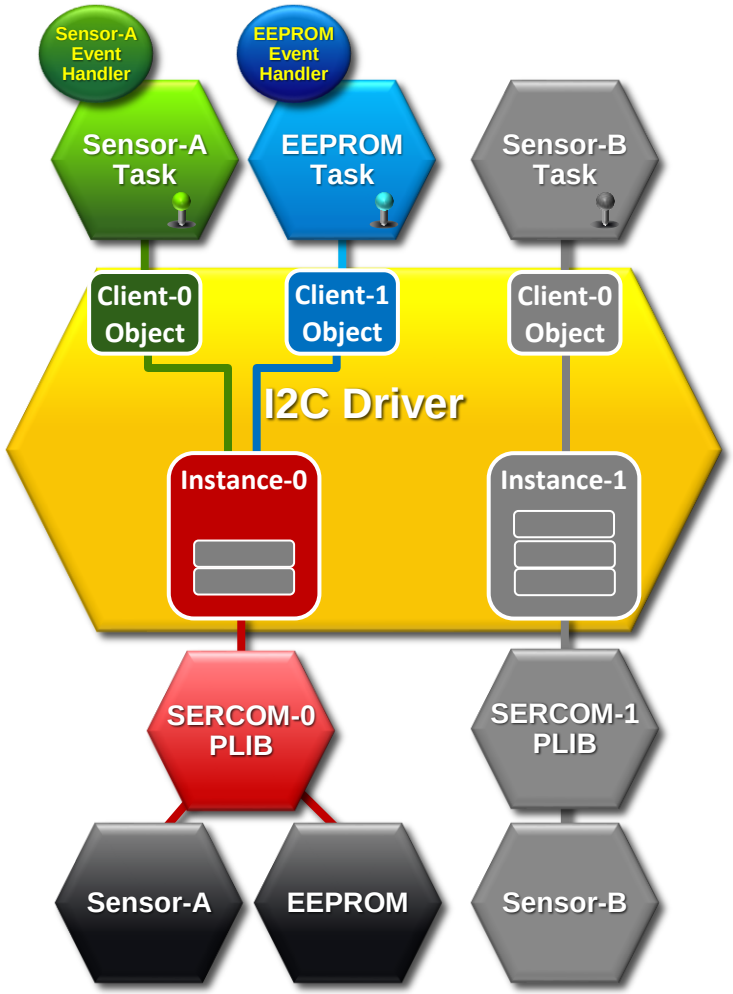
How to use Driver in Harmony?

Register and Define Transfer Callback

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

void eeprom_EventHandler(…)
{
    eeprom_ReqCompleted = true;
}

void sensorA_Task( void )
{
    DRV_I2C_TransferEventHandlerSet (
        sensorA_ClientHandle,
        sensorA_EventHandler, 0 );
}

void eeprom_Task( void )
{
    DRV_I2C_TransferEventHandlerSet (
        eeprom_ClientHandle,
        eeprom_EventHandler, 0 );
}
```

Sensor-A Event Handler

EEPROM Event Handler



MICROCHIP

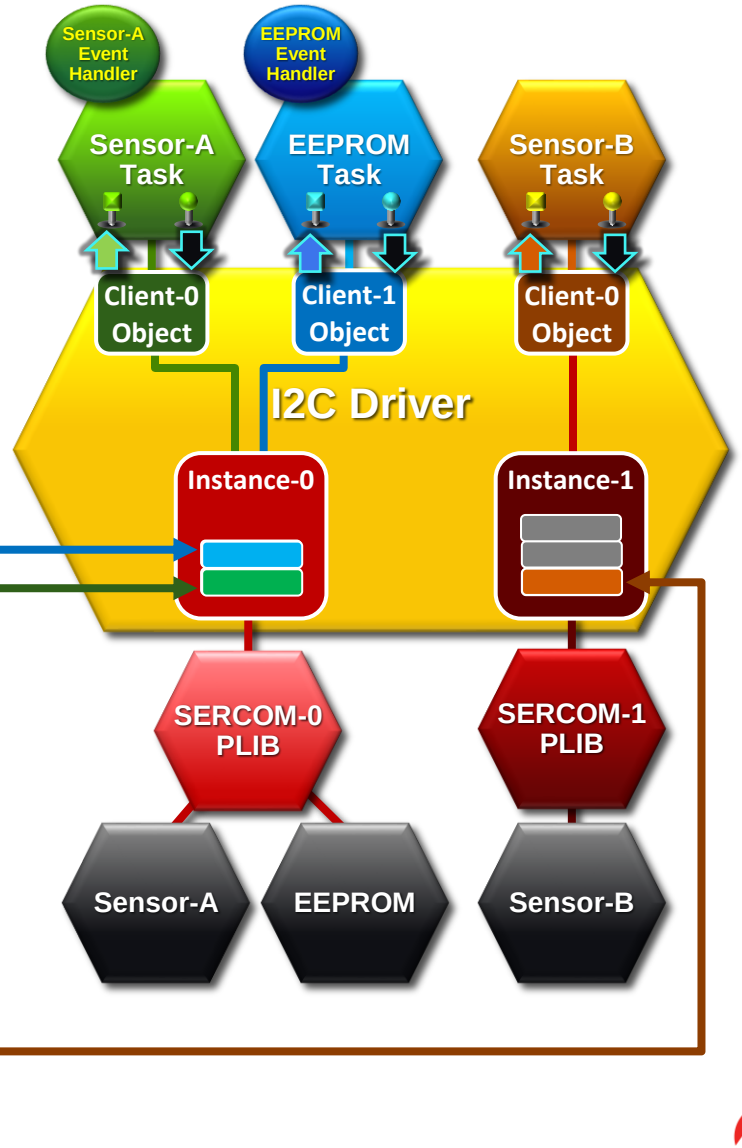
How to use Driver in Harmony?

- 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
);
```



MICROCHIP

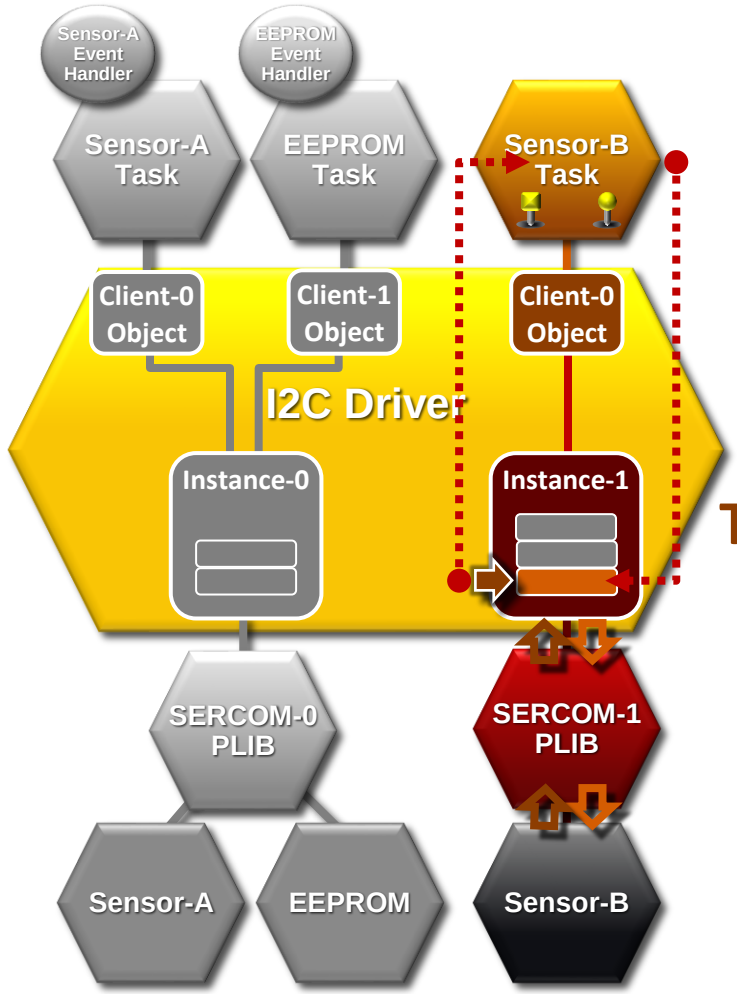
How to use Driver in Harmony?

- 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
        }
    }
}
```



Transfer Status Polling

MICROCHIP

How to use Driver in Harmony?

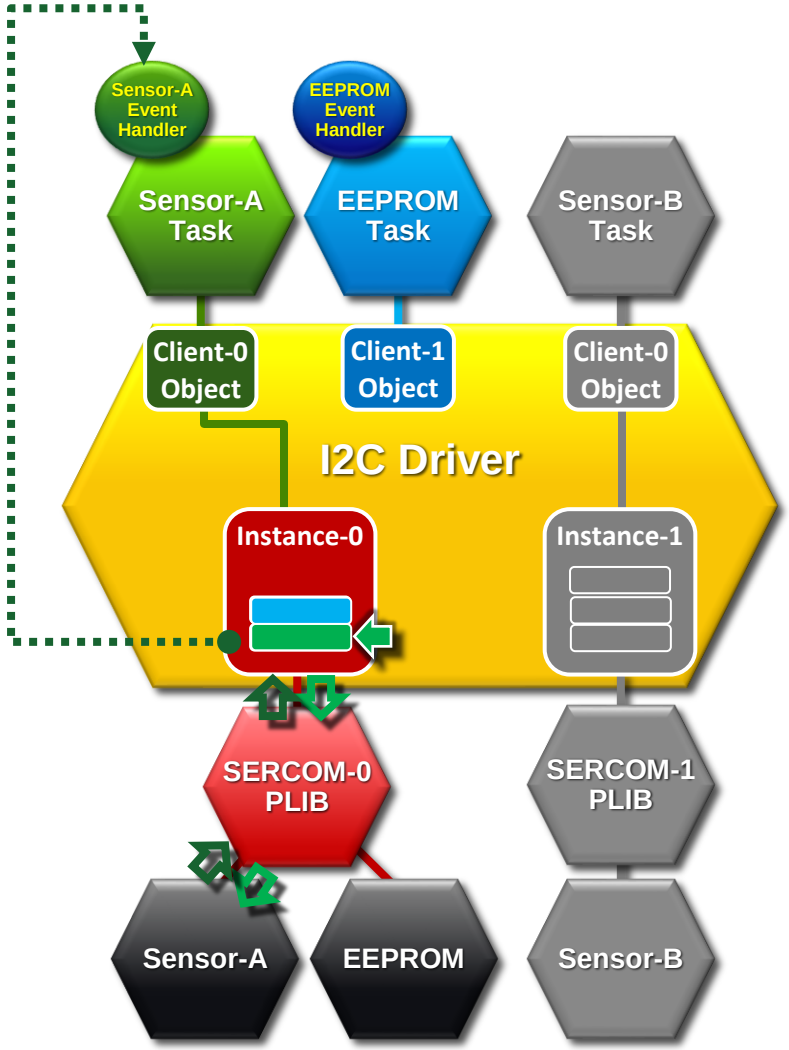
- Add & Configure Using MHC
- Initialize Driver
- Open Driver and get Client Handle
- Register and Define Transfer Callback
- Transfer Request with Queue
- Transfer Status Polling operation
- Transfer Status Callback operation
- Two Transfer Request to use one Callback
- Two Client to use one Callback

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;
    }
}
```

Sensor-A Event Handler

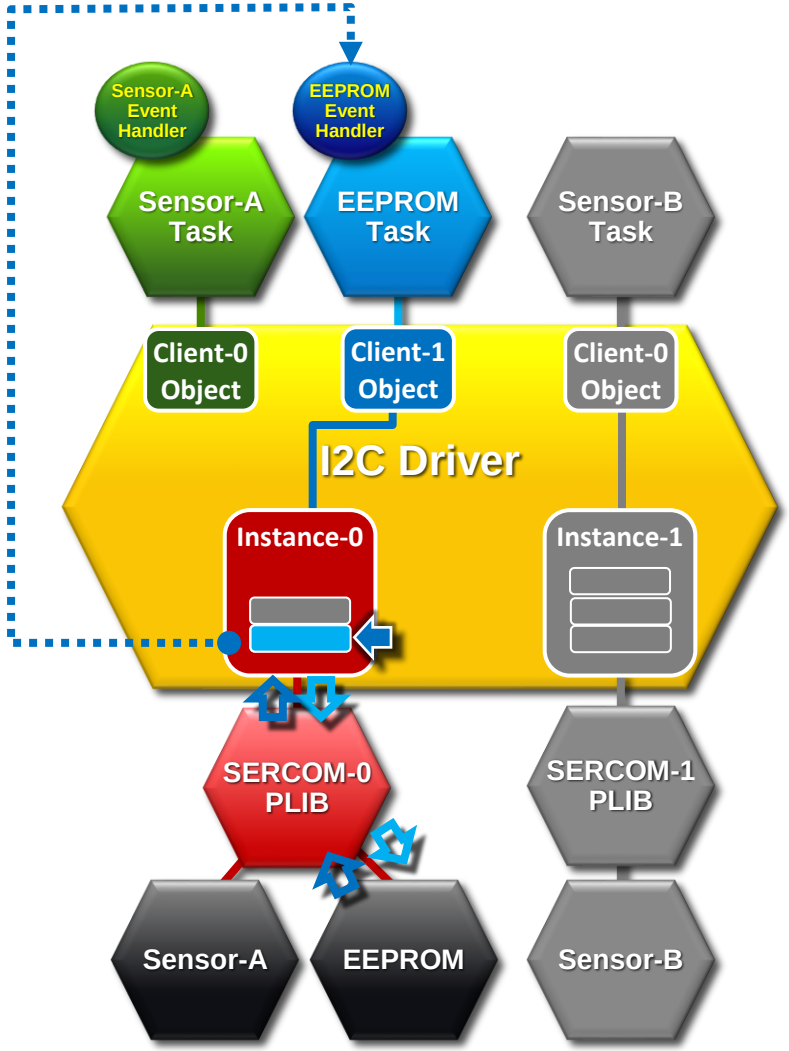


How to use Driver in Harmony?

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



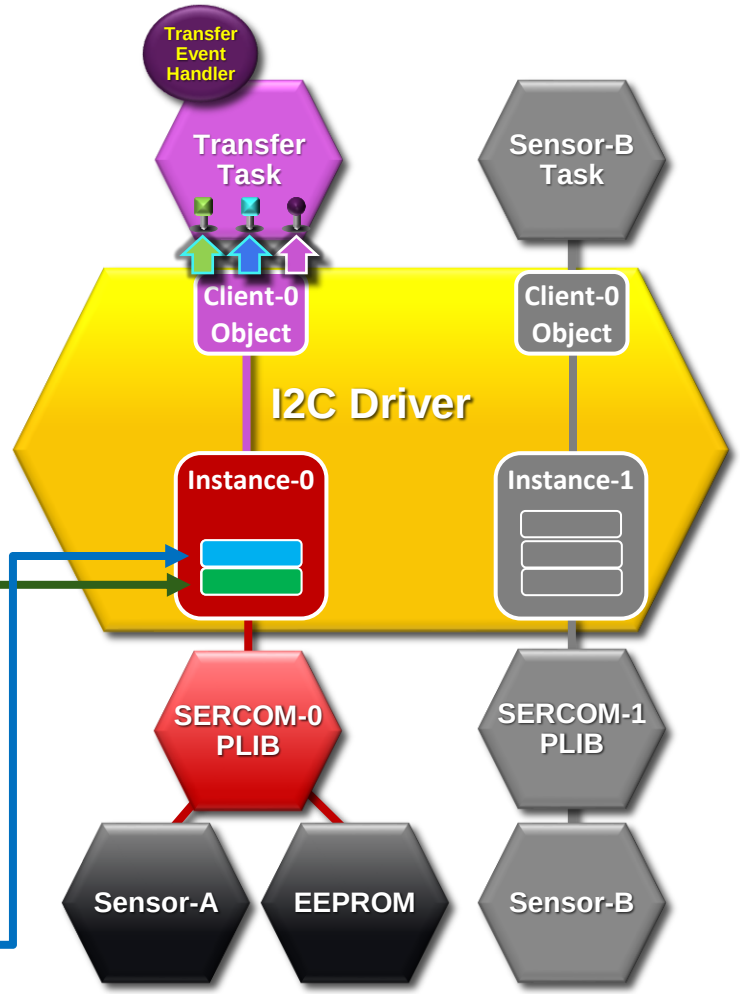
How to use Driver in Harmony?

- 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
    );
}
```



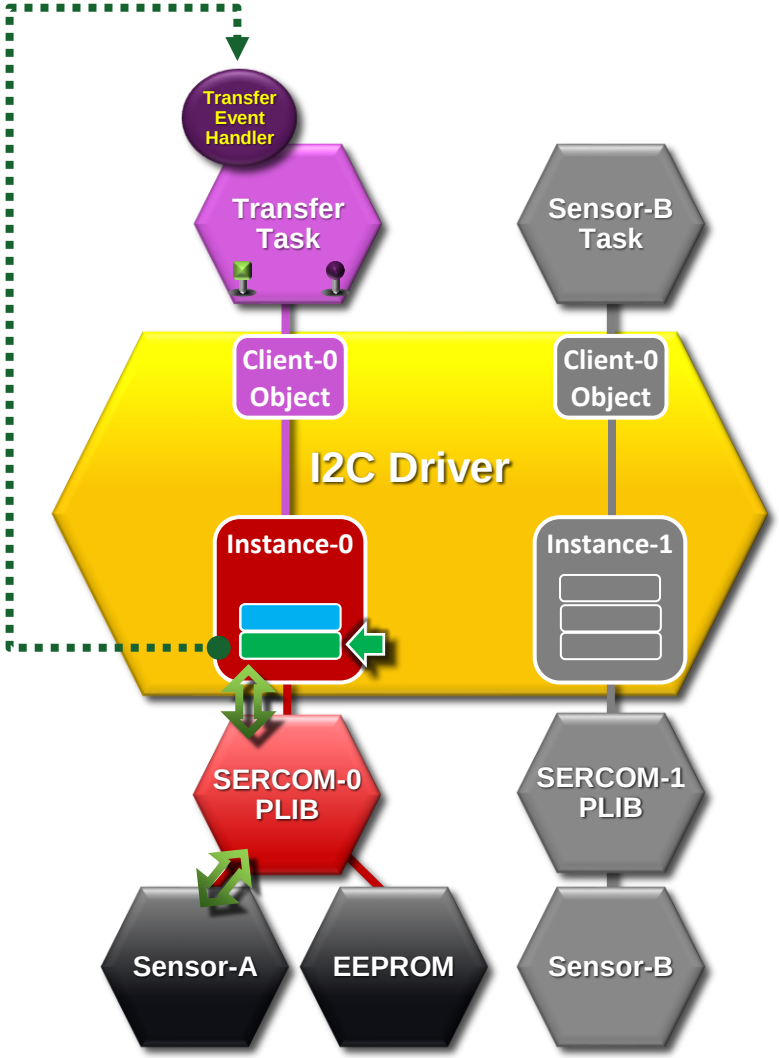
MICROCHIP

How to use Driver in Harmony?

Two Transfer Request to use one Callback

```
void transfer_EventHandler(
    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;
    }
}
```

Transfer Event Handler



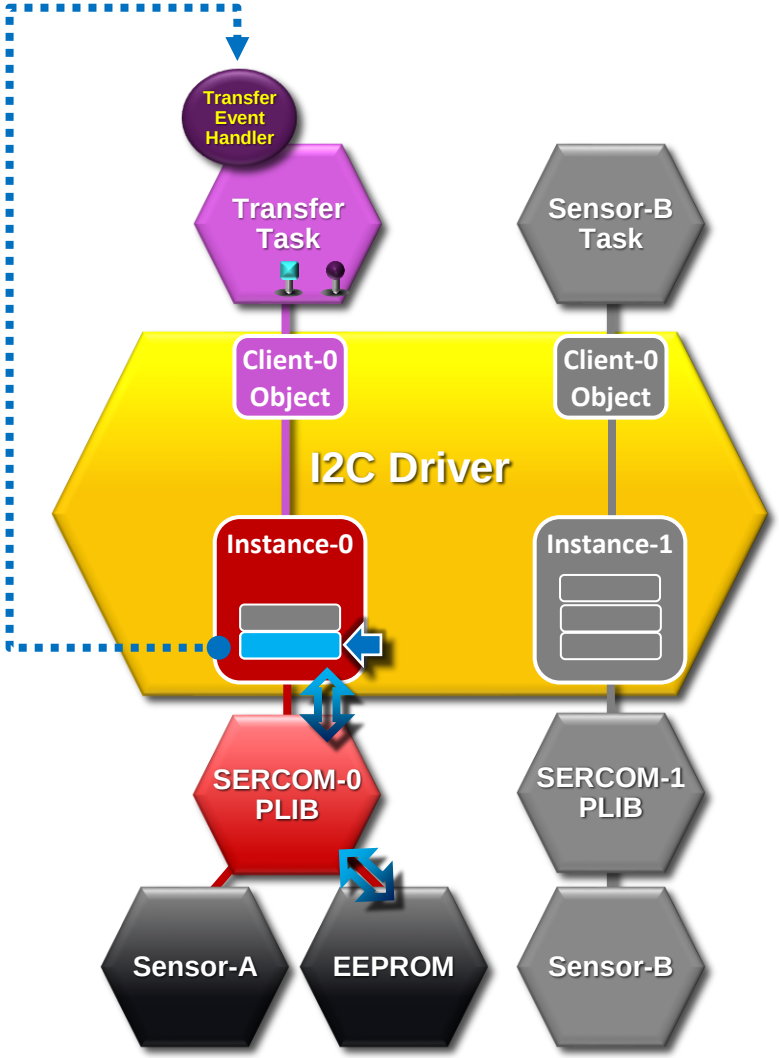
MICROCHIP

How to use Driver in Harmony?

Two Transfer Request to use one Callback

```
void transfer_EventHandler(
    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;
    }
}
```

Transfer Event Handler



MICROCHIP

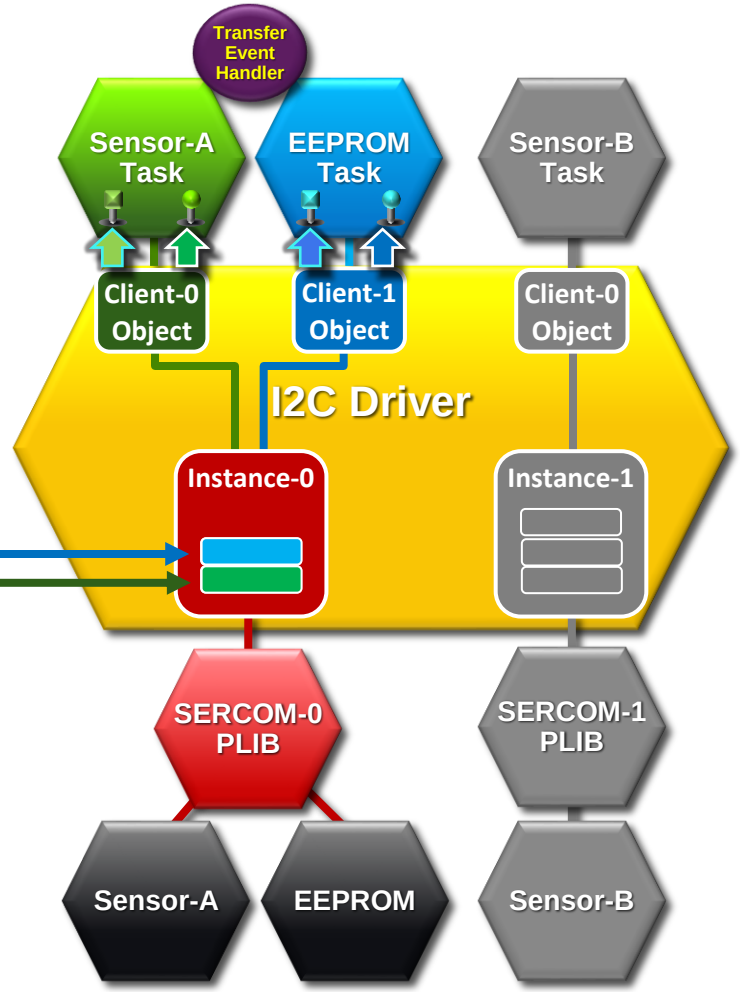
How to use Driver in Harmony?

- 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

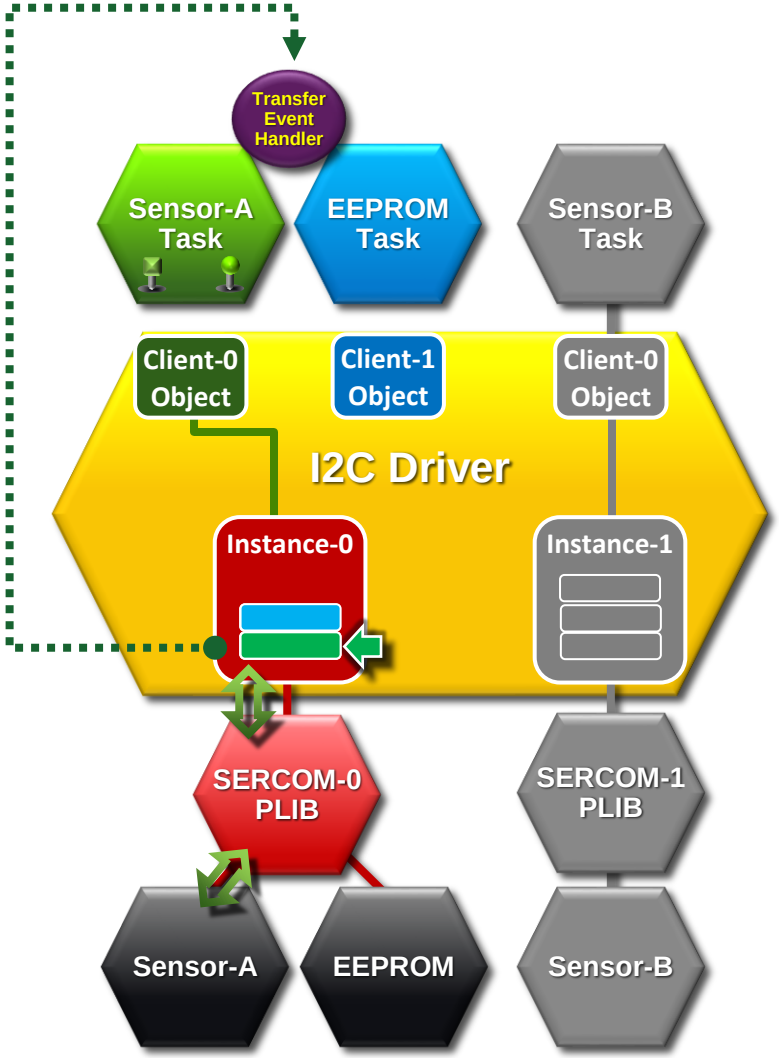
```
void sensorA_Task( void ) {
   sensorA_ClientHandle = DRV_I2C_Open (
         DRV_I2C_INDEX_0, Instance-0
         DRV_IO_INTENT_READWRITE  );
     DRV_I2C_TransferEventHandlerSet (
       sensorA_ClientHandle,
       transfer_EventHandler, 0 );
     DRV_I2C_ReadTransferAdd (
       sensorA_ClientHandle, ...,
       &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(
    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;
    }
}
```

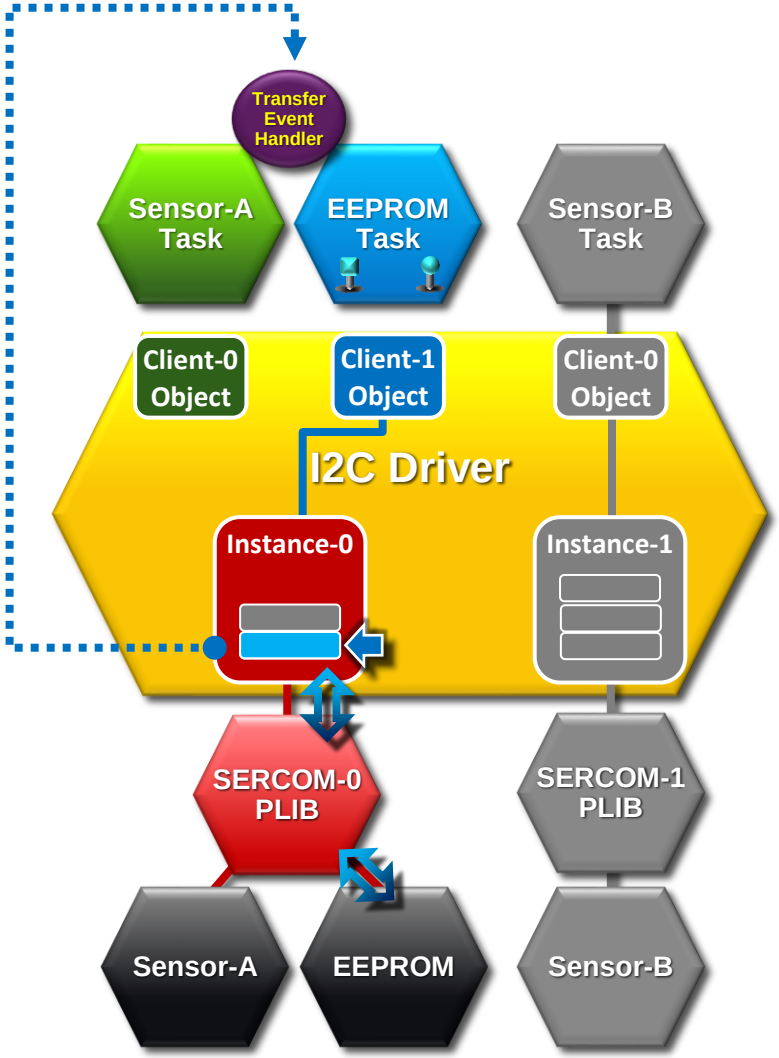


How to use Driver in Harmony?

Two Client to use one Callback

```
void transfer_EventHandler(
    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;
    }
}
```

Transfer Event Handler

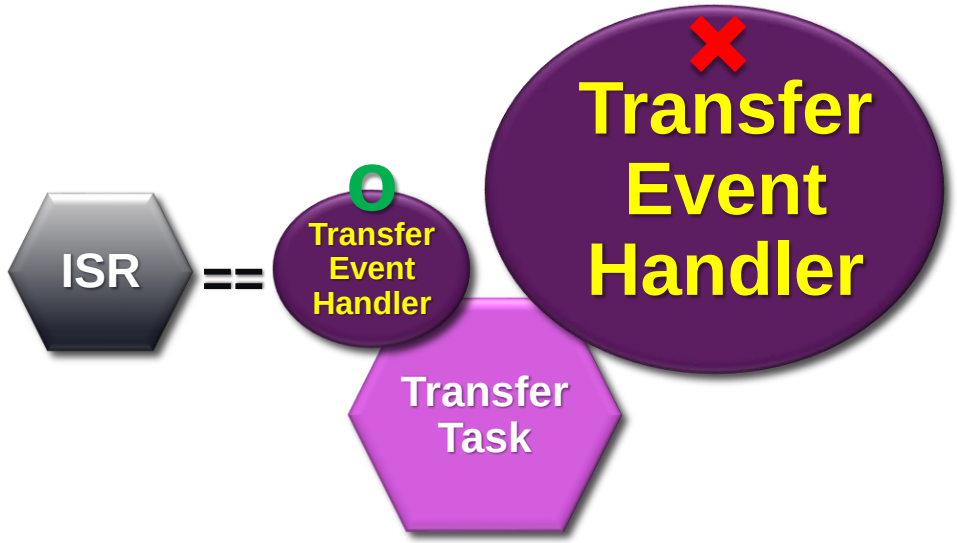


MICROCHIP

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;
O volatile int TransferFinished = false;
  void transfer_EventHandler( ... )
  {
    ✖ TransferDone = true;
    O TransferFinished = true;

    ✖ DRV_ReadTransferAdd( ... );
    ✖ APP_InterruptNotSafe( ... );
  }
```



I2C Driver API and Usage Example

Polling method

```
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(…);
}
```

Callback 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(…);
}
```

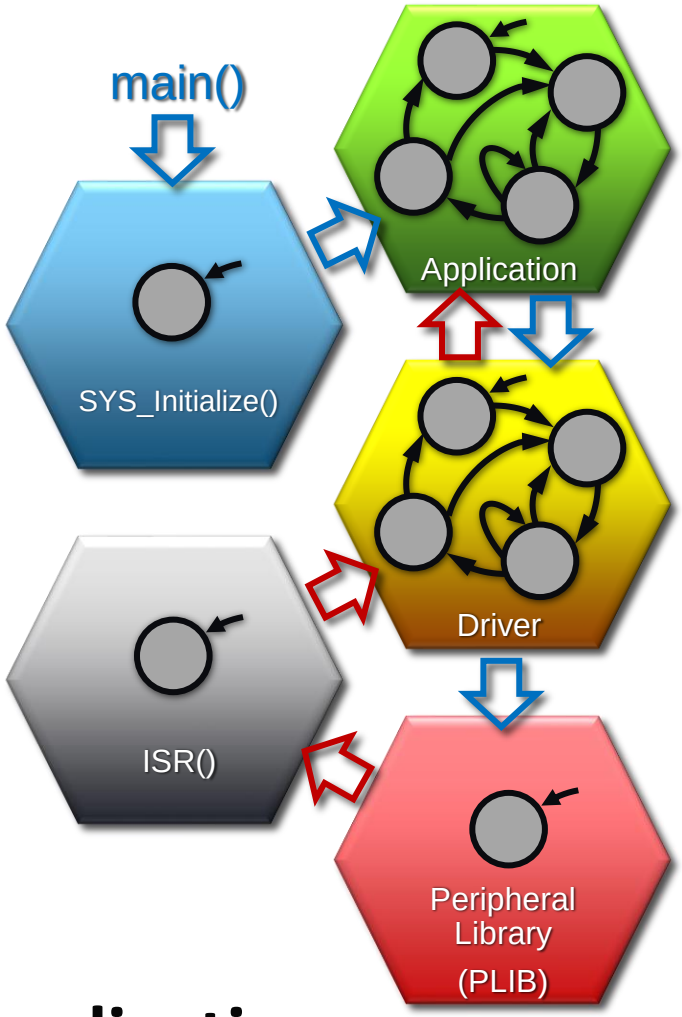
I2C Driver Client API list:

- 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

MICROCHIP

USART Driver Library

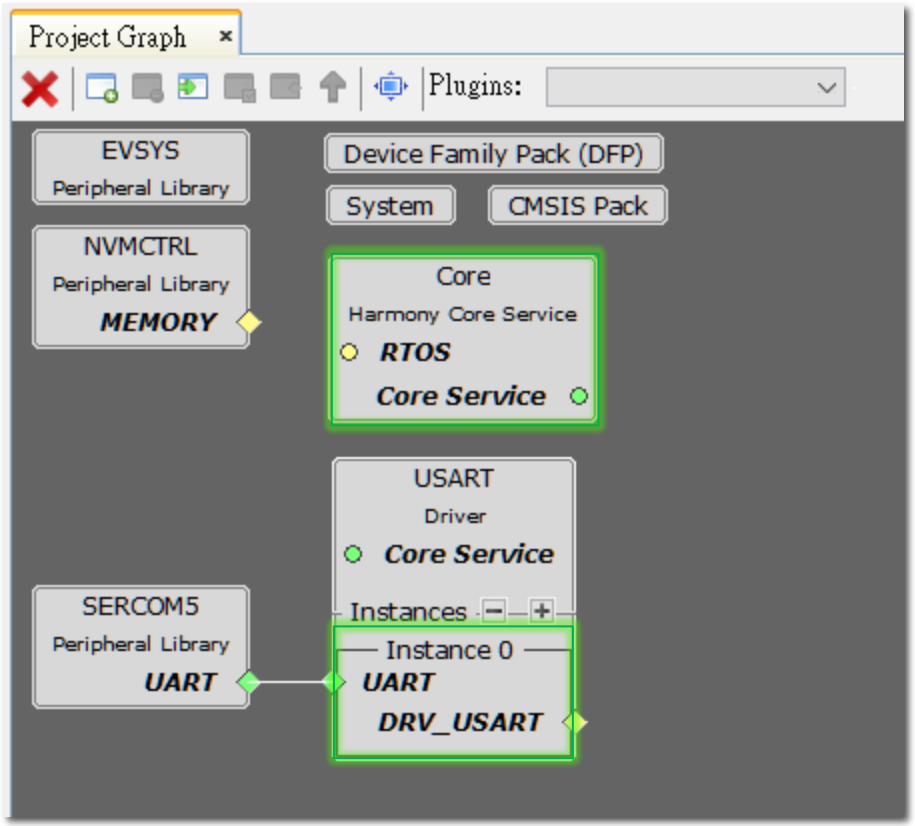
- **Introduction**

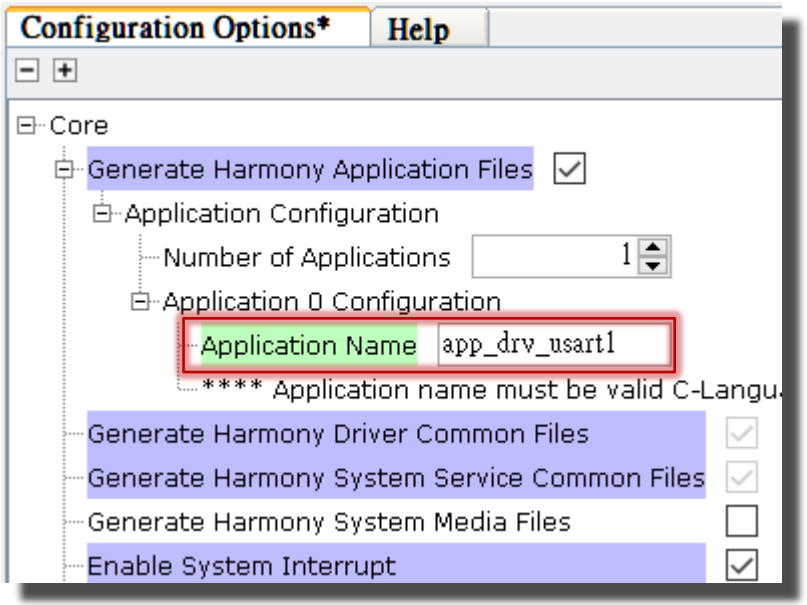
  This driver provides application ready routines to read and write data from/to the USART peripheral.

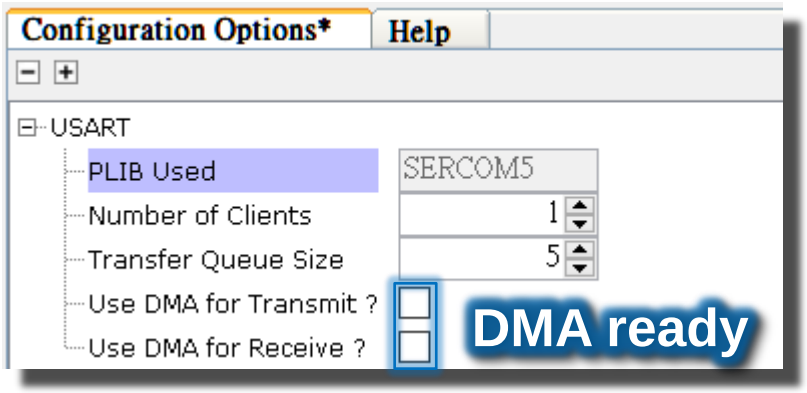
- **The USART driver support the following features:**
  - Supports multi-instance and multi-client mode.
  - Provides data transfer events.
  - Supports blocking and non-blocking operation.
  - Features thread-safe functions for use in RTOS applications.
  - Supports DMA transfers.

USART Driver Library

To generate an Application File for Driver State Machine







MICROCHIP

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.

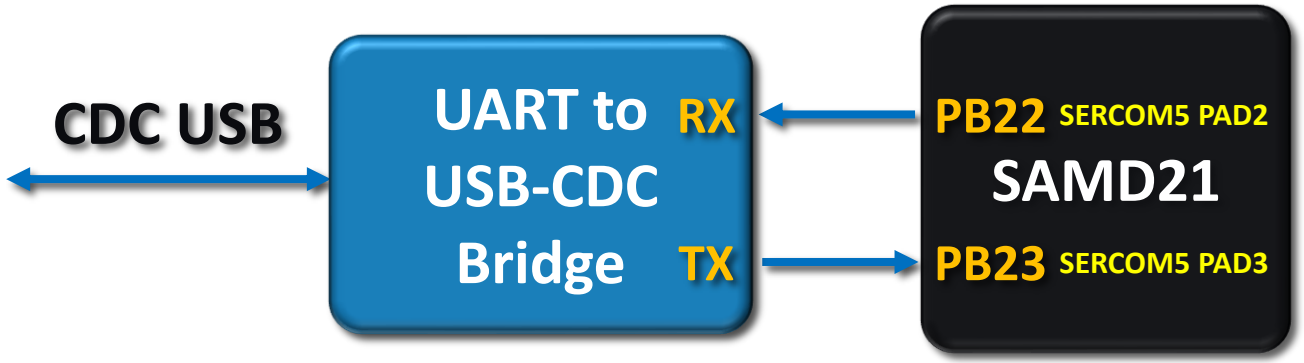
🛠 Lab 11
Single-Client UART Communication

USART Driver Library

MICROCHIP

🛠 Lab11 Single-Client UART Communication

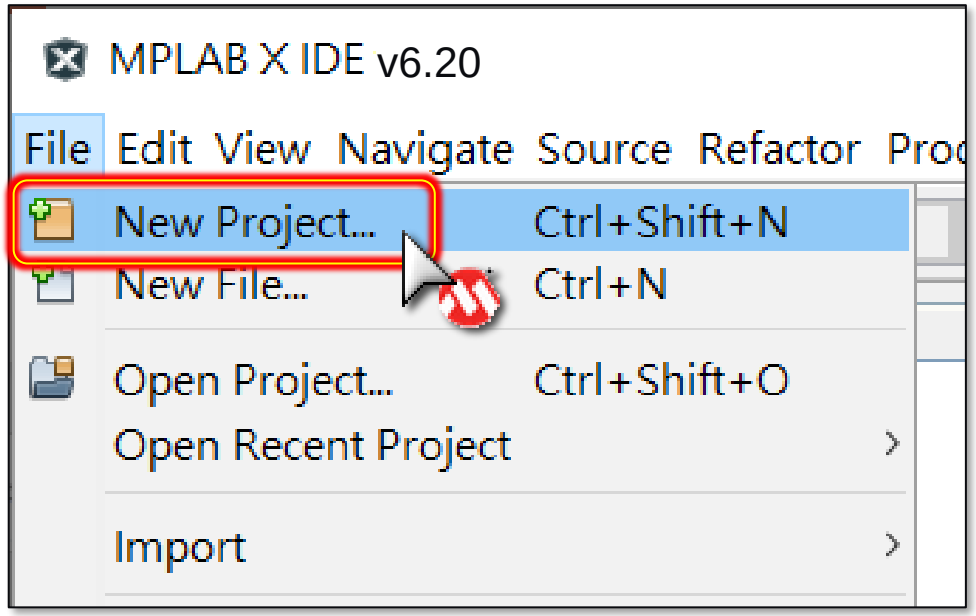
- Please create a new project for Harmony Driver exercise.
- Finish the Lab1 – Delay based LED toggle in main loop for Driver non-block validate.
- Try add USART Driver and attach PLIB SERCOM5 UART to build a custom driver based UART Console. (not to use the CONSOLE System Service)
- Implement your UART 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)"



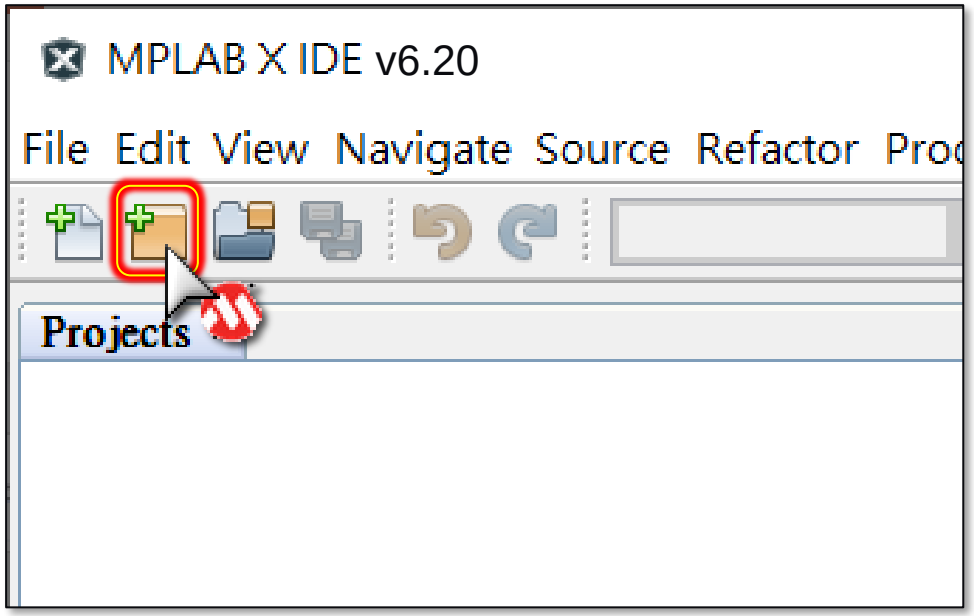
MICROCHIP

🛠 Lab11 Single-Client UART Communication

- **Create a new 32 bits MCC Harmony Project**



OR

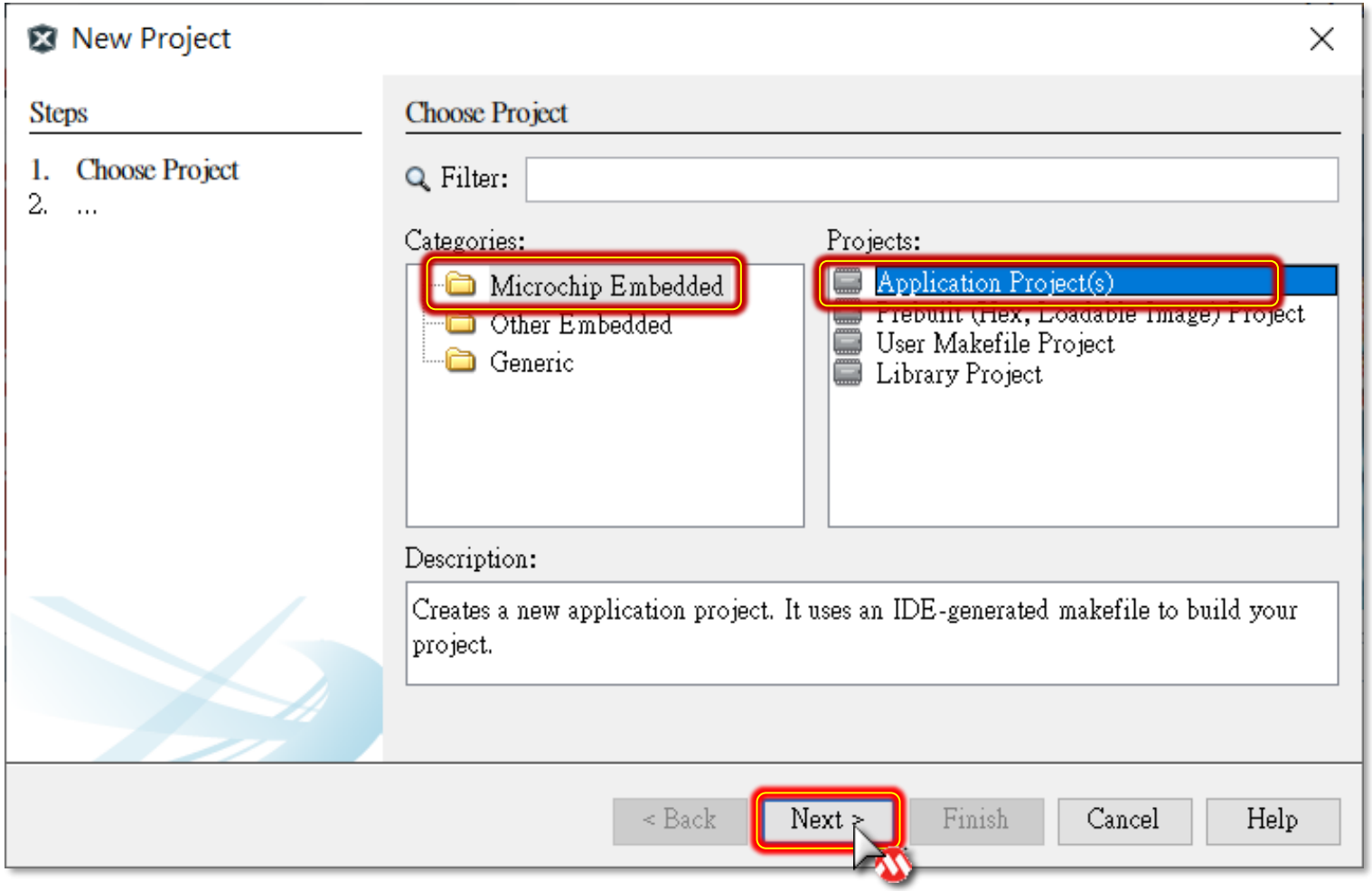


Not hint for this step anymore after Lab1

MICROCHIP

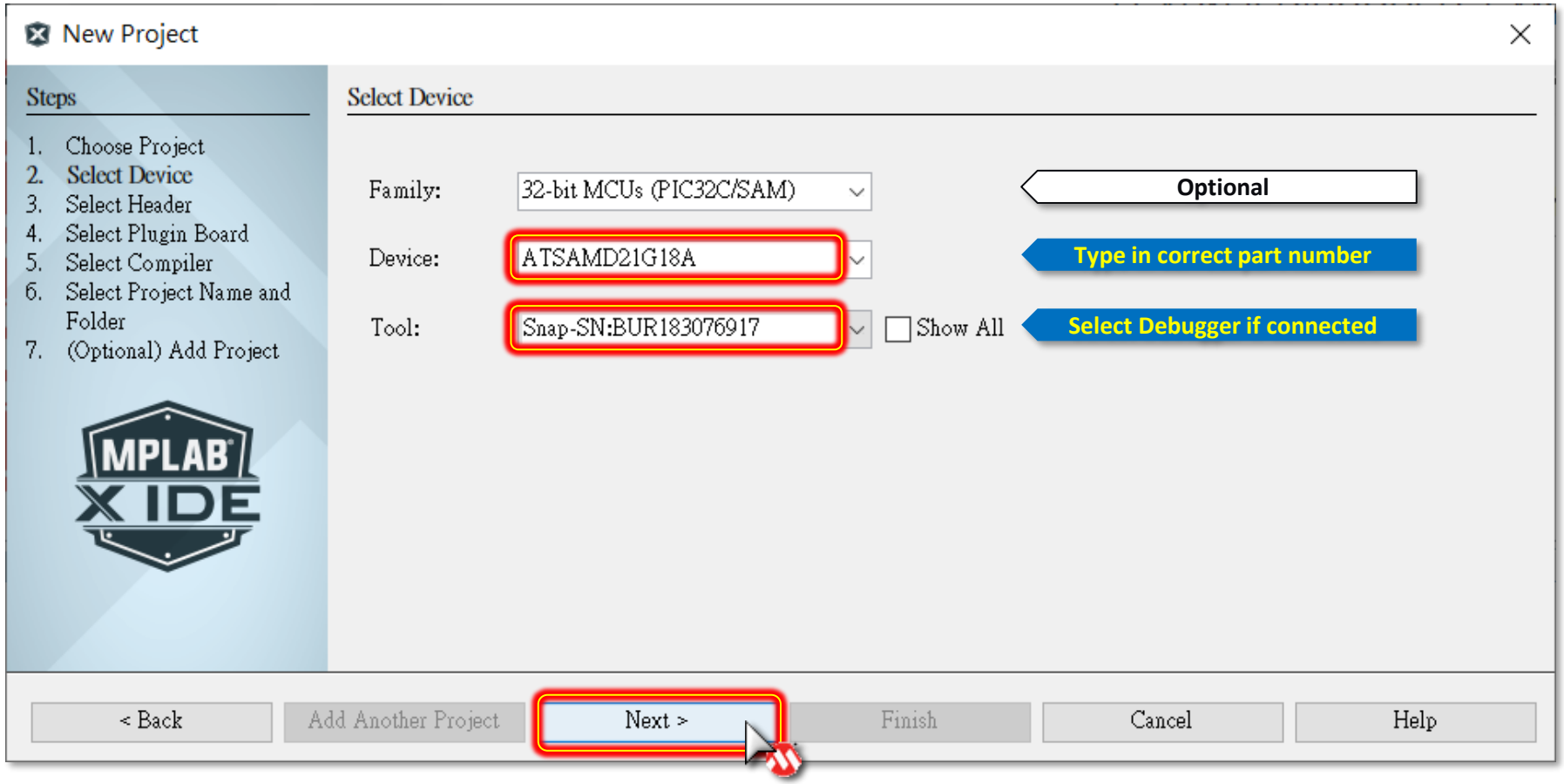
🛠 Lab11 Single-Client UART Communication

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



🛠 Lab11 Single-Client UART Communication

- **Family : 32-bit MCUs (PIC32C/SAM)**
- **Device : ATSAMD21G18A** Please do not miss any character of your part number
- **Tool : Snap-SN:BURxxxxxxxxx**

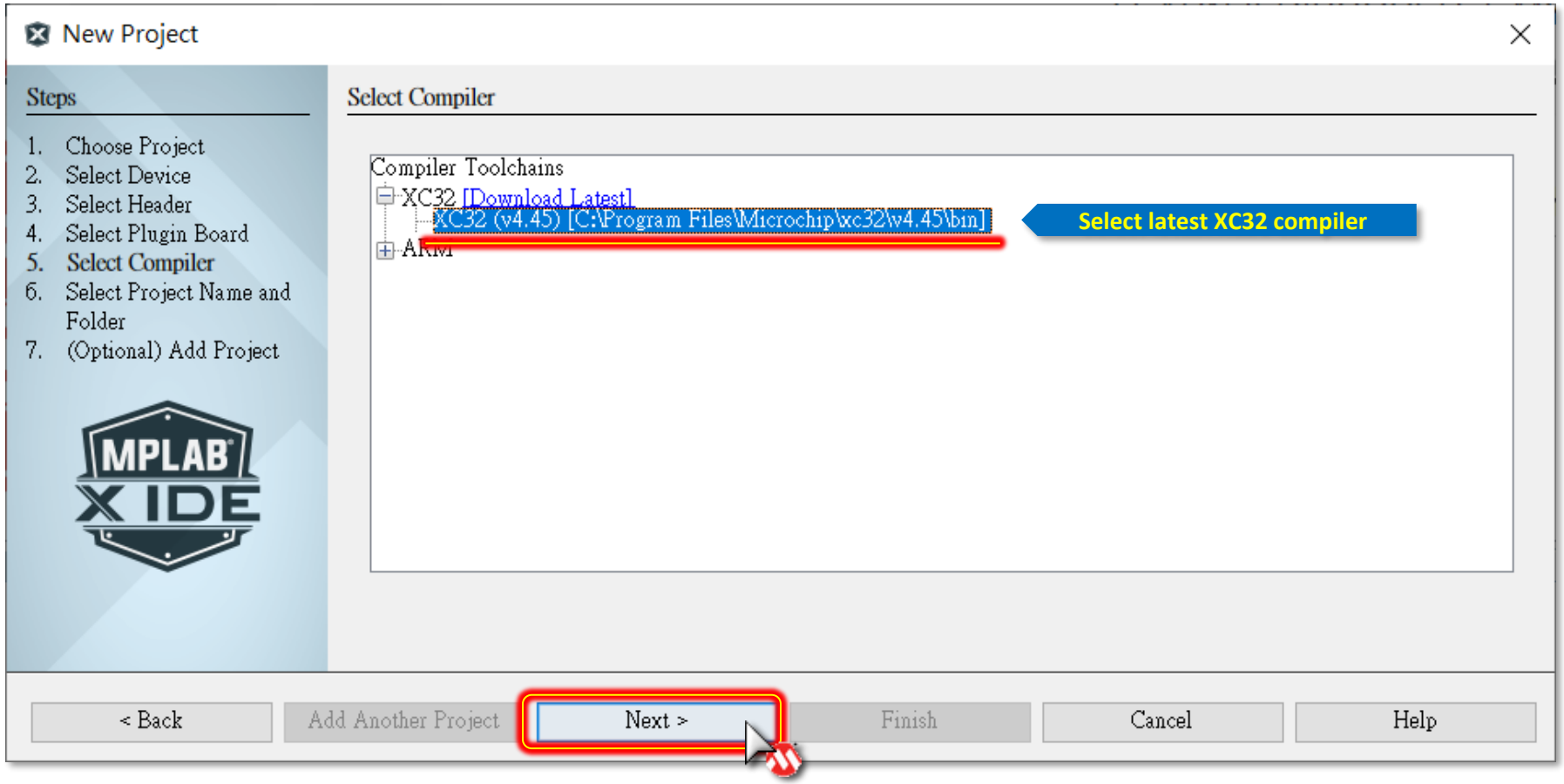


MICROCHIP

🛠 Lab11 Single-Client UART Communication

Select latest XC32 Compiler.

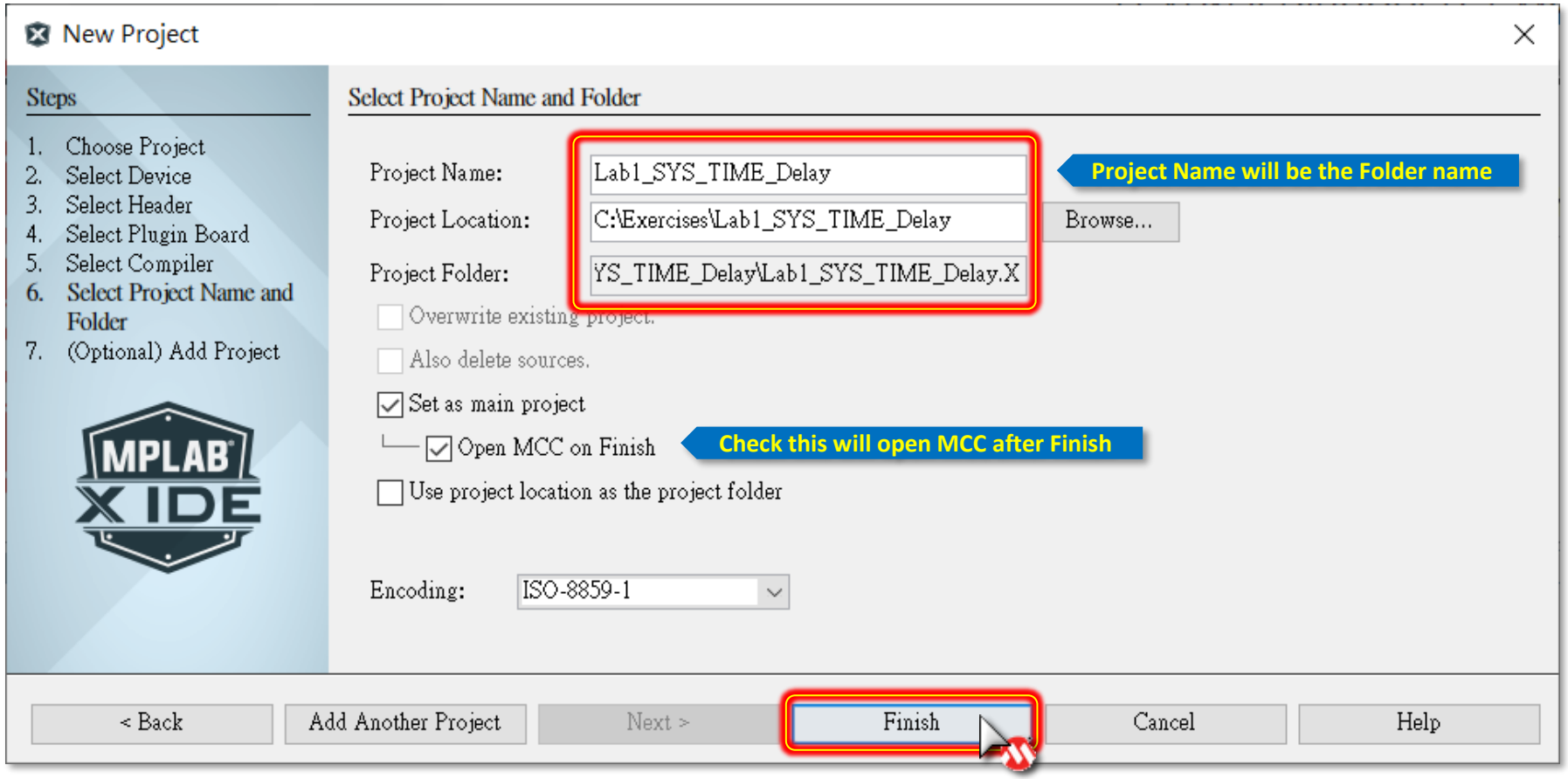
- **XC32**
  - XC32 (v4.45) [C:\Program File\Microchip\xc32\..)



MICROCHIP

🛠 Lab11 Single-Client UART Communication

- **Project Name** : **Lab11_DRV_USART_SingleClient**
- **Project Location** : **C:\Exercises\Lab11_DRV_USART_SingleClient**
- **Project Folder** : **C:\Exercises\Lab11_DRV_USART_SingleClient\Lab11_DRV_USART_SingleClient.X**

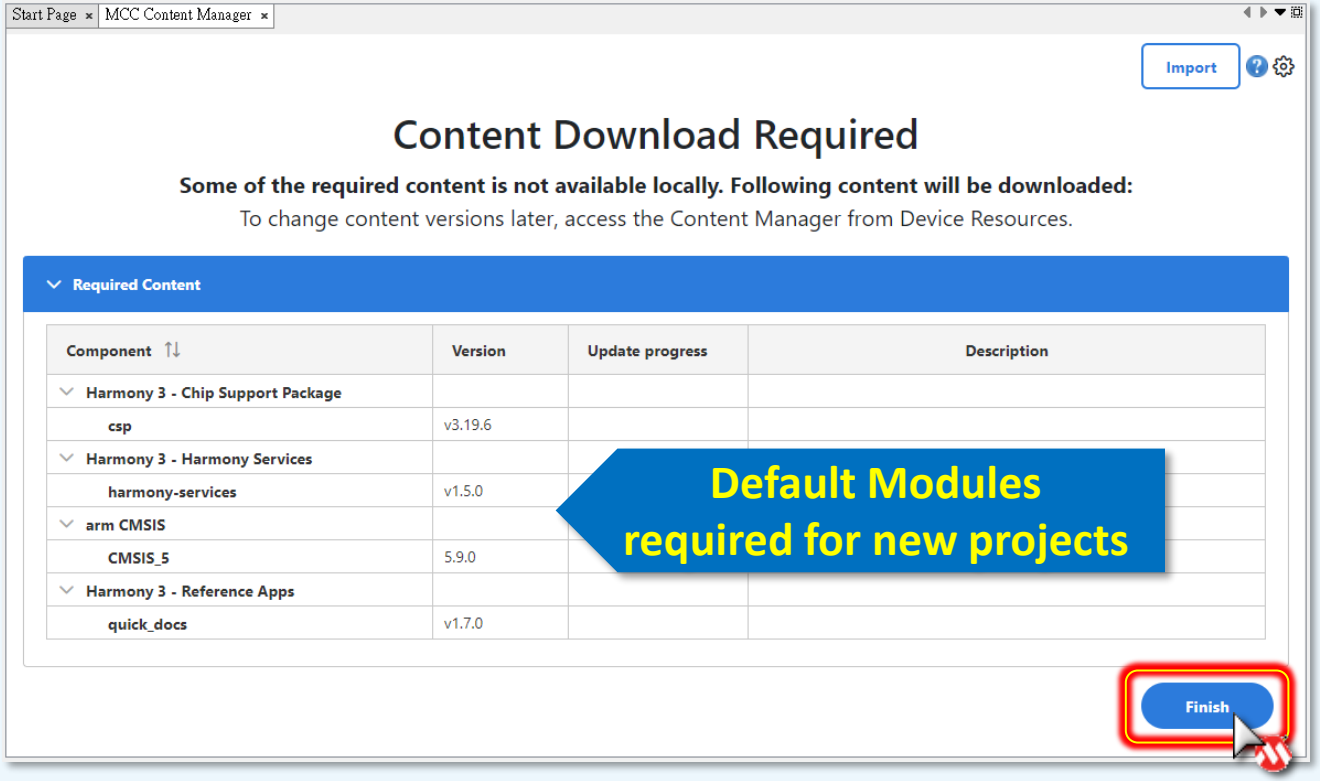


MICROCHIP

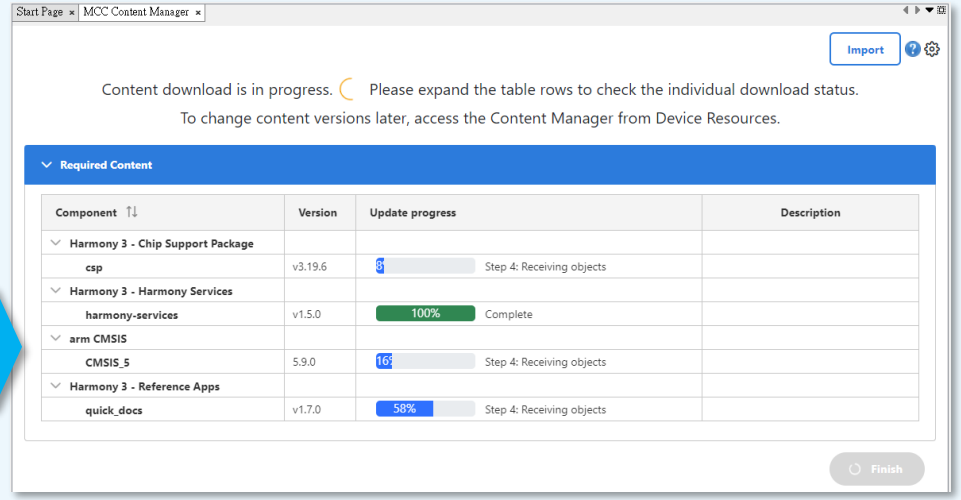
🛠 Lab11 Single-Client UART Communication

New Download of MCC Harmony Framework

- **Lunching the Finish in MCC Content Manager to download Required Content.**
  - Default required Modules : **csp**, **harmony-services**, **CMSIS_5** and **quick_docs**.



It will take about 10 mins to download



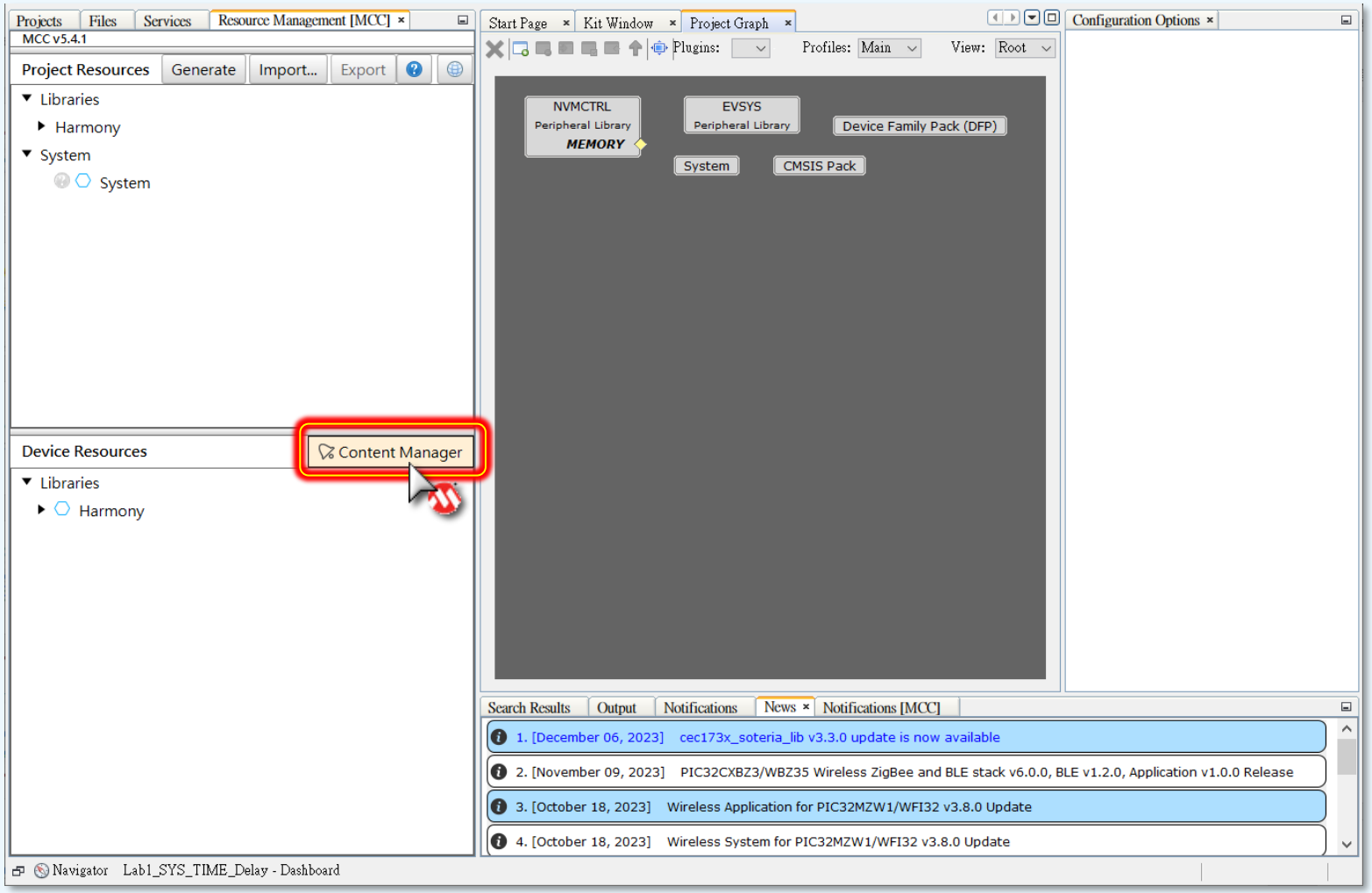
MICROCHIP

🛠 Lab11 Single-Client UART Communication

New Download of MCC Harmony Framework

- **Lunching the** 

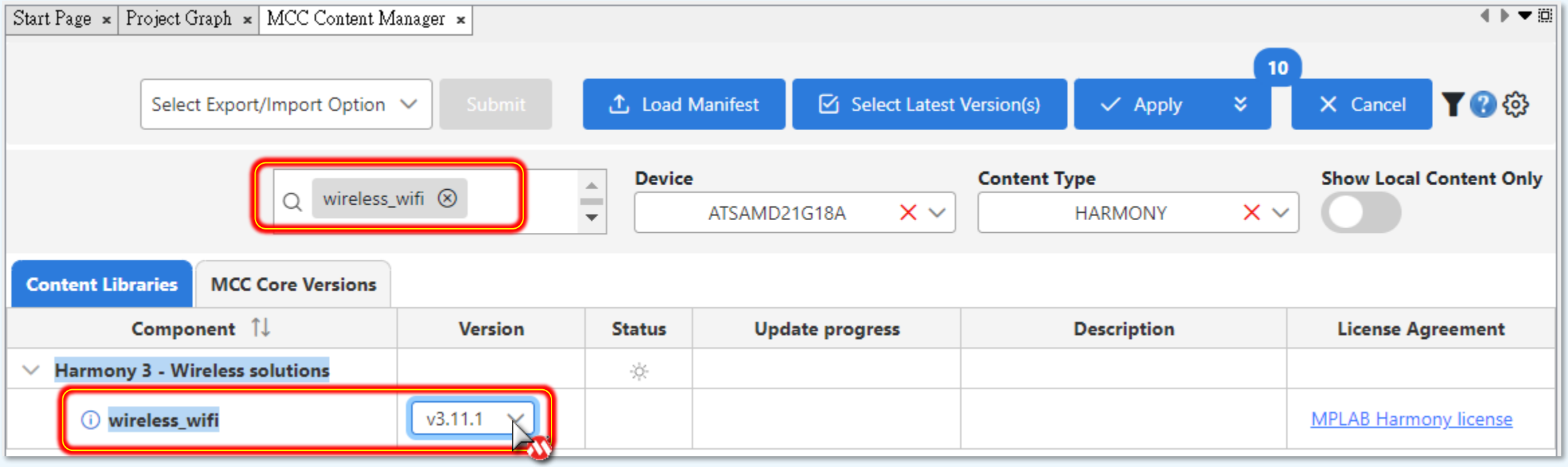
in “Device Resources” window of MCC Harmony.



🛠 Lab11 Single-Client UART Communication

New Download of MCC Harmony Framework

- Input “wireless_wifi” in search window.
- Under “Harmony 3 – Wireless solution” > “wireless_wifi” select to “v3.11.1”.

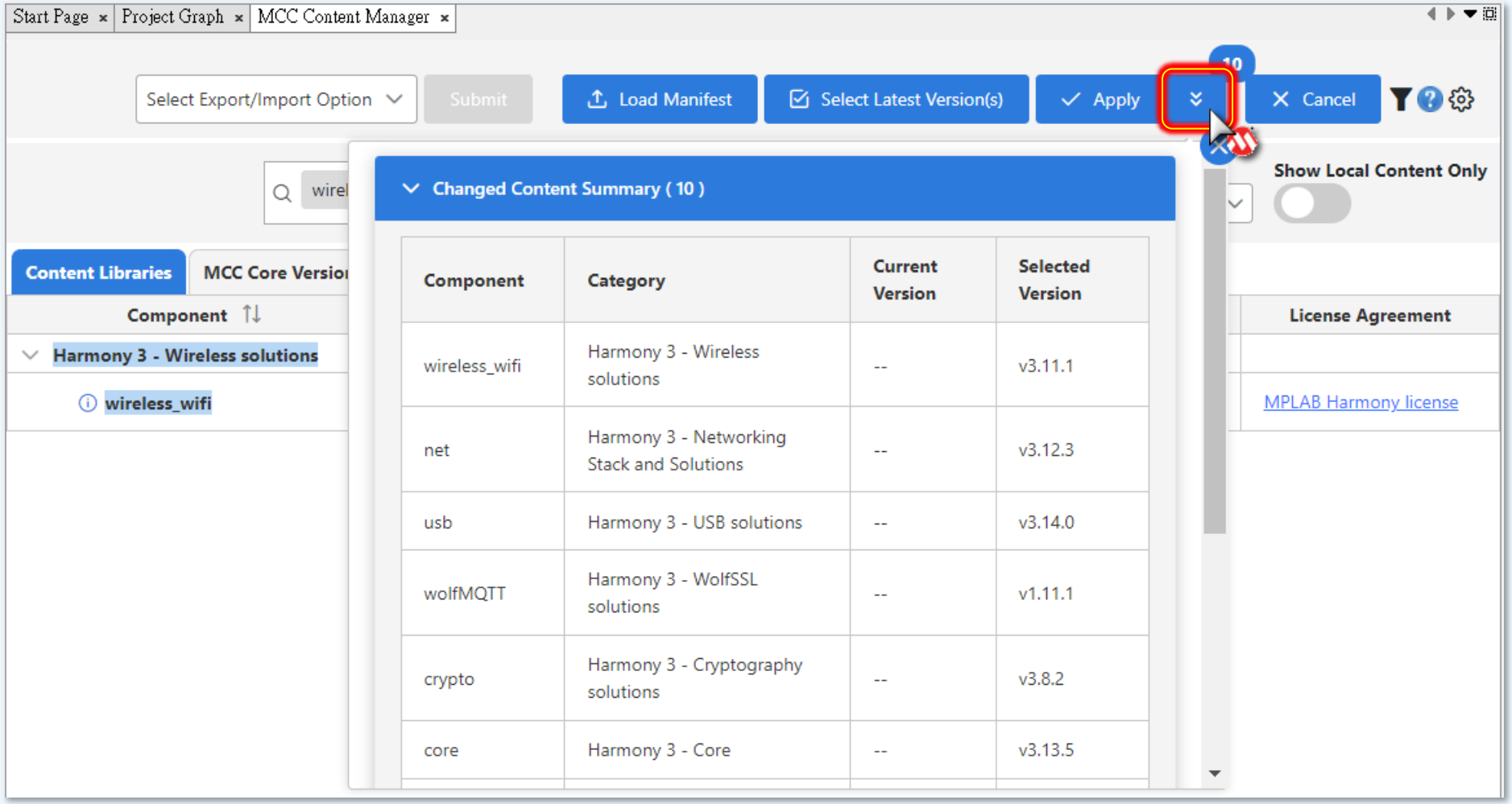


MICROCHIP

🛠 Lab11 Single-Client UART Communication

New Download of MCC Harmony Framework

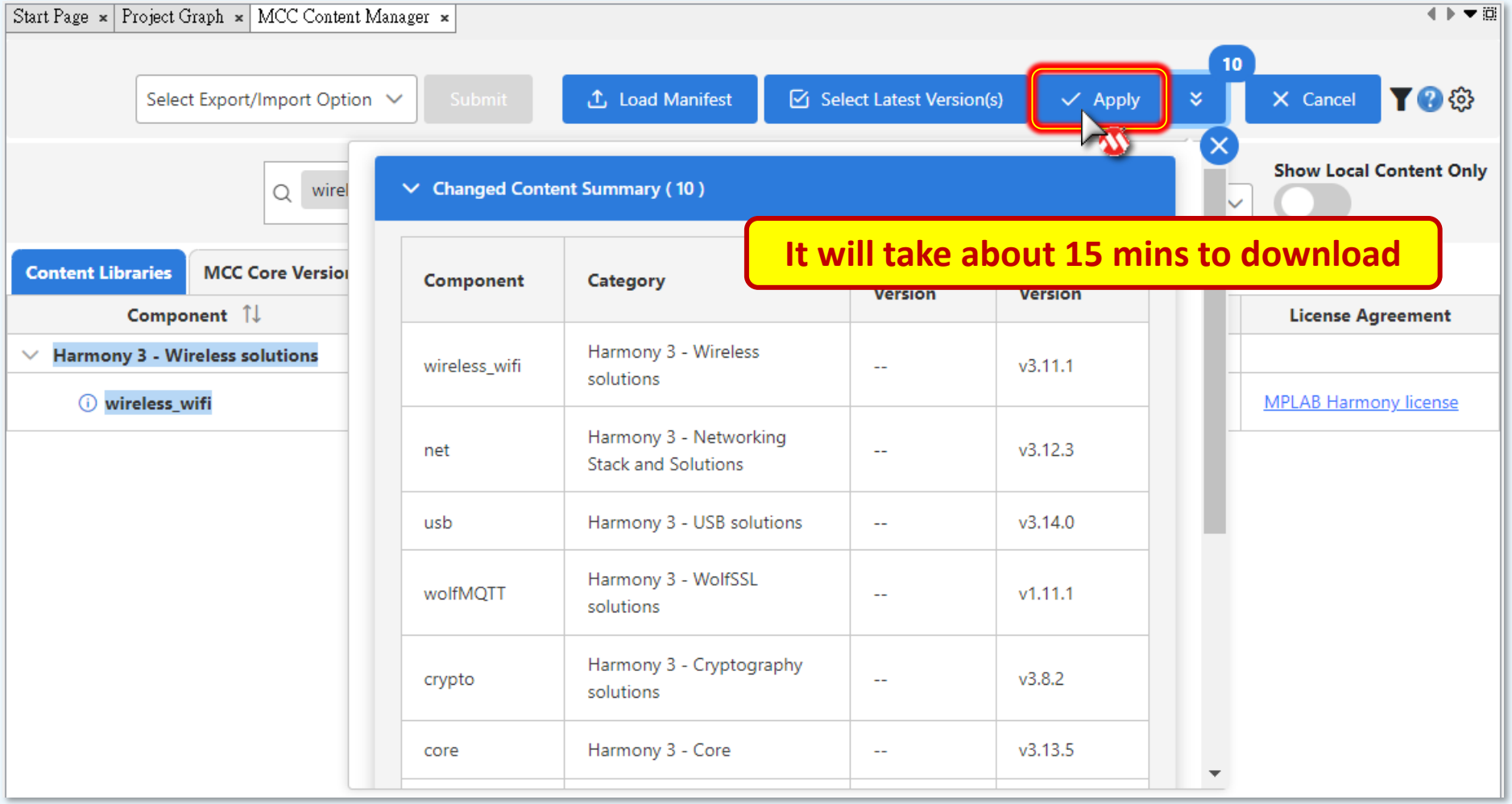
- **In Content Manager, it will automatically select necessary dependences modules.**



🛠 Lab11 Single-Client UART Communication

New Download of MCC Harmony Framework

- **Lunching the** ✓ Apply **in MCC Content Manager to download Optional Content.**

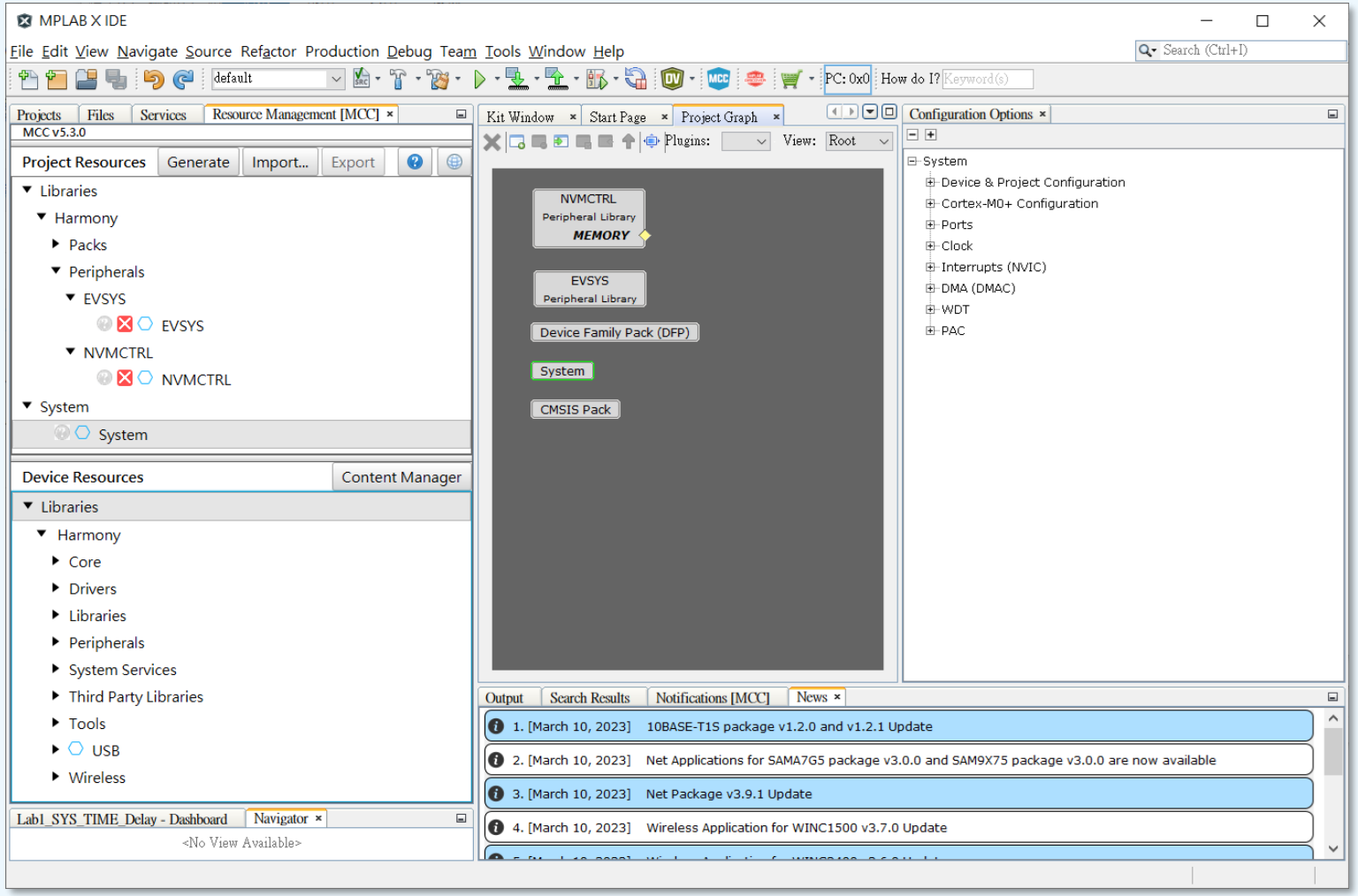


MICROCHIP

🛠 Lab11 Single-Client UART Communication

New Download of MCC Harmony Framework

- **MCC Harmony Configurator Interface.**

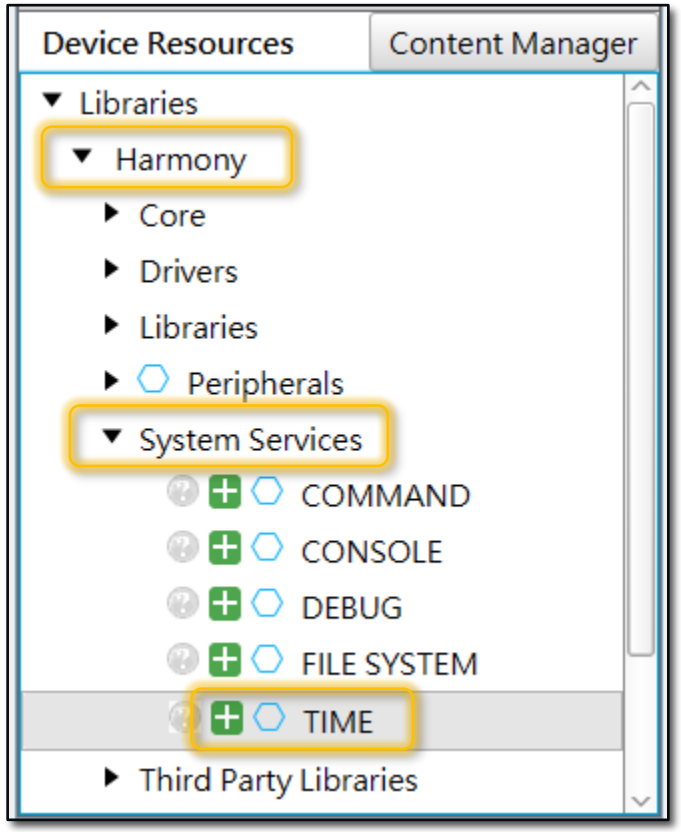


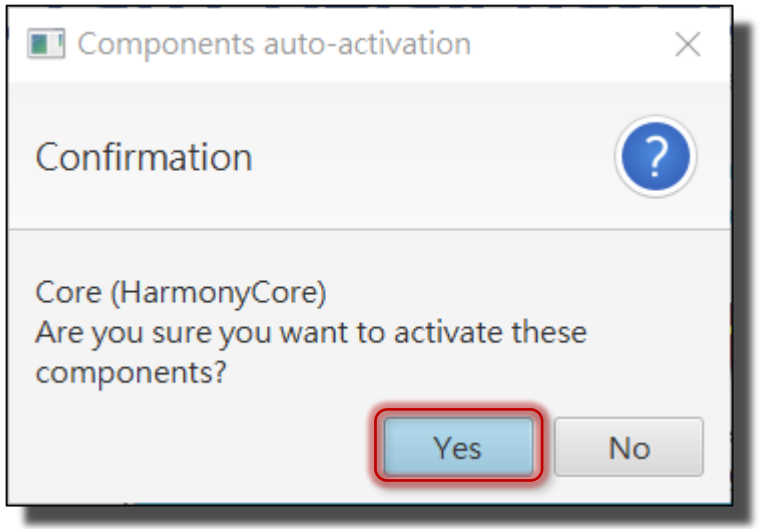
MICROCHIP

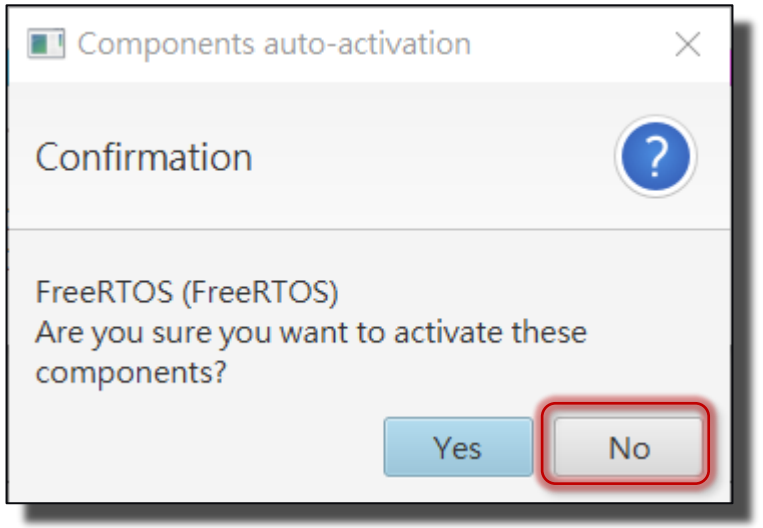
🛠 Lab11 Single-Client UART Communication

- **Harmony Configuration (Add TIME System Service)**

Add TIME System Service as we did in Lab1



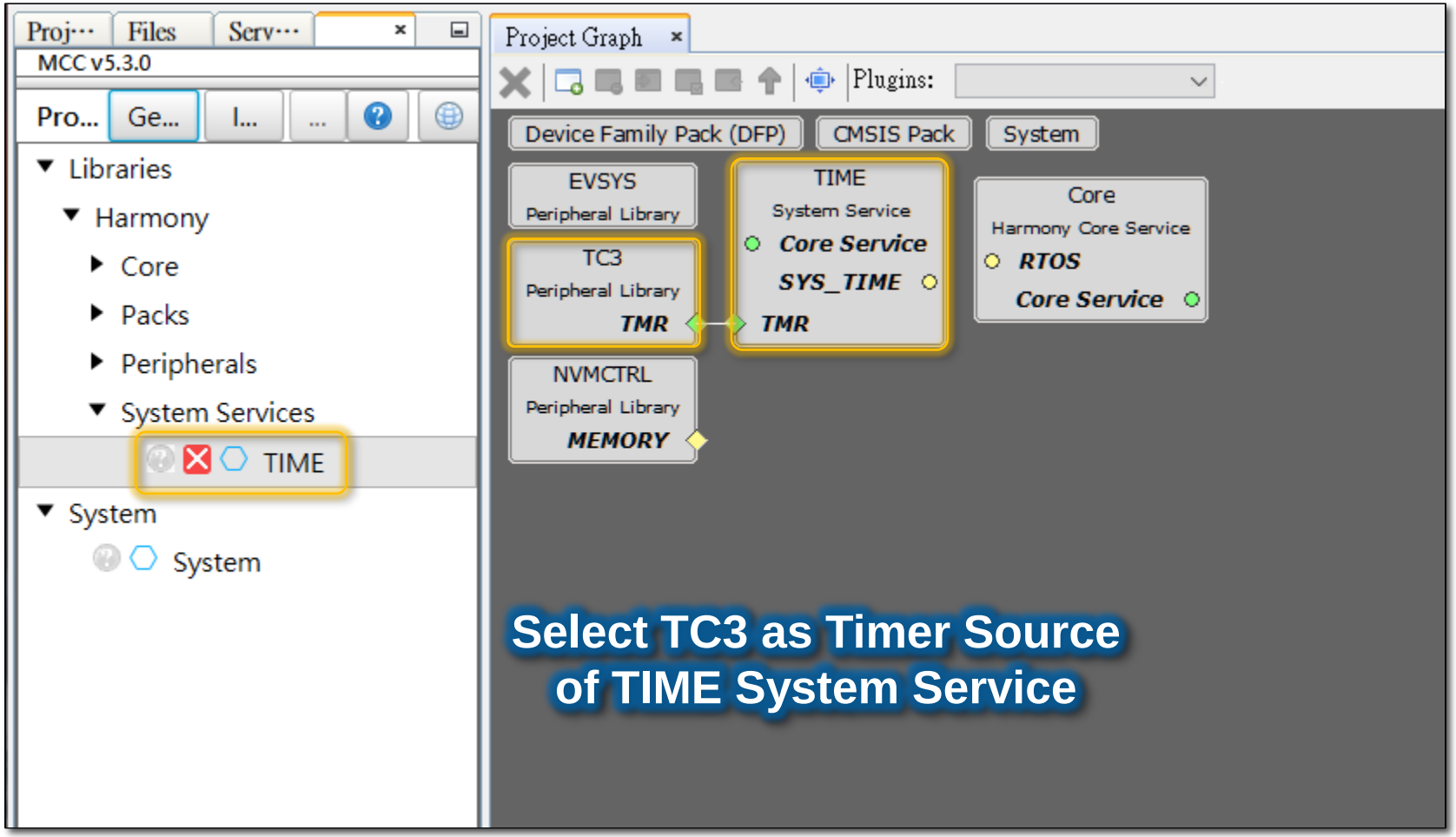




MICROCHIP

🛠 Lab11 Single-Client UART Communication

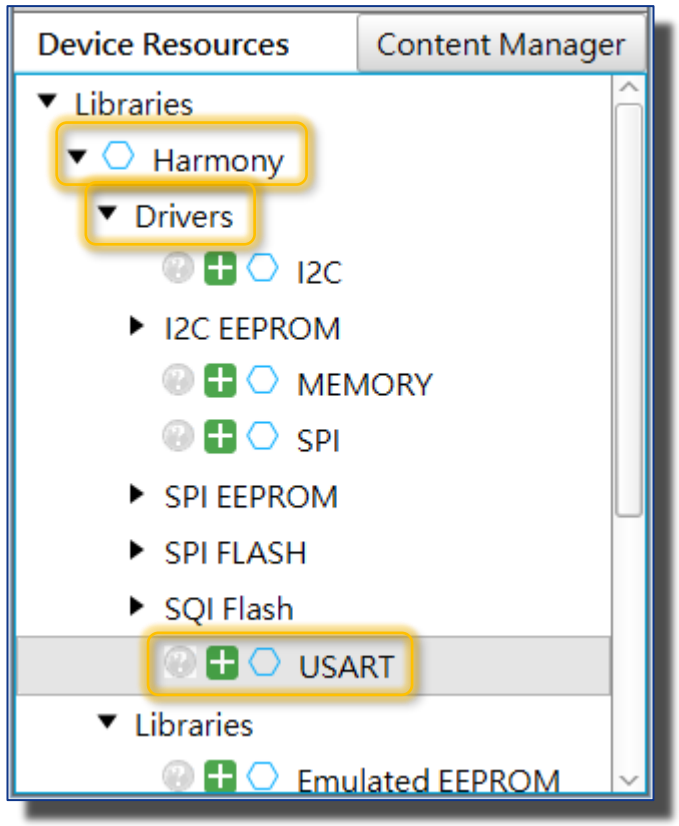
- Choose TC3 as Timer source of TIME System Service,
- Use the TIME System Service to create a polling-based delay and toggle LED1 in main loop for Driver non-block validate.

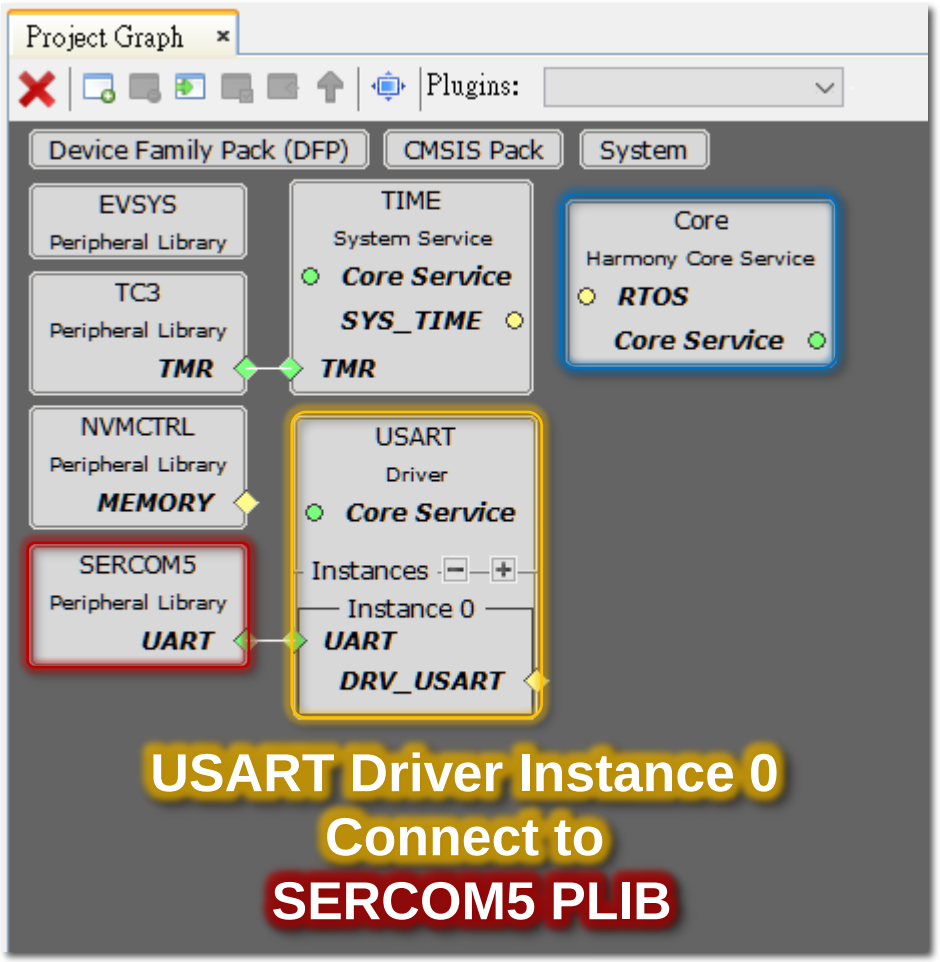


MICROCHIP

🛠 Lab11 Single-Client UART Communication

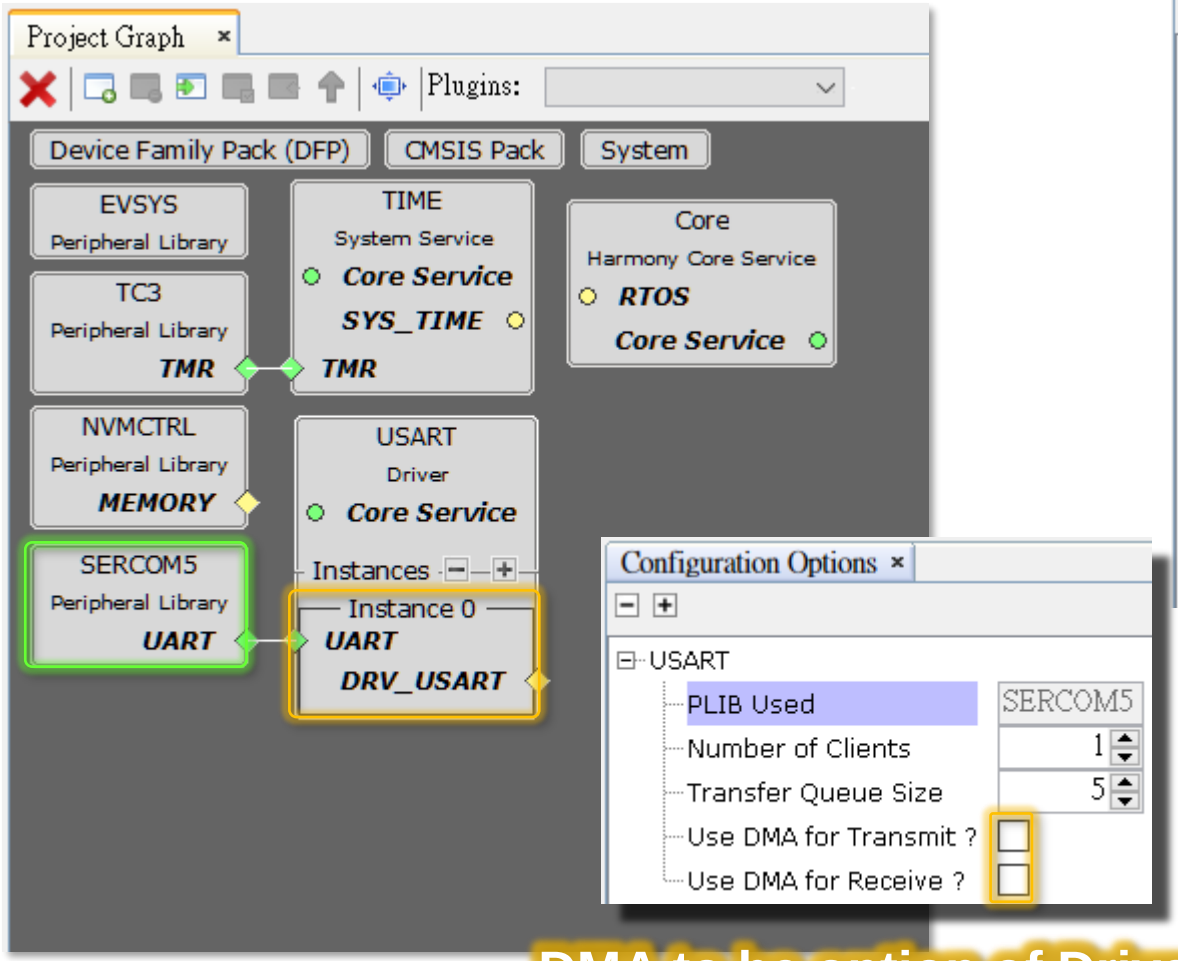
- **Add USART driver and attach PLIB SERCOM5 UART.**



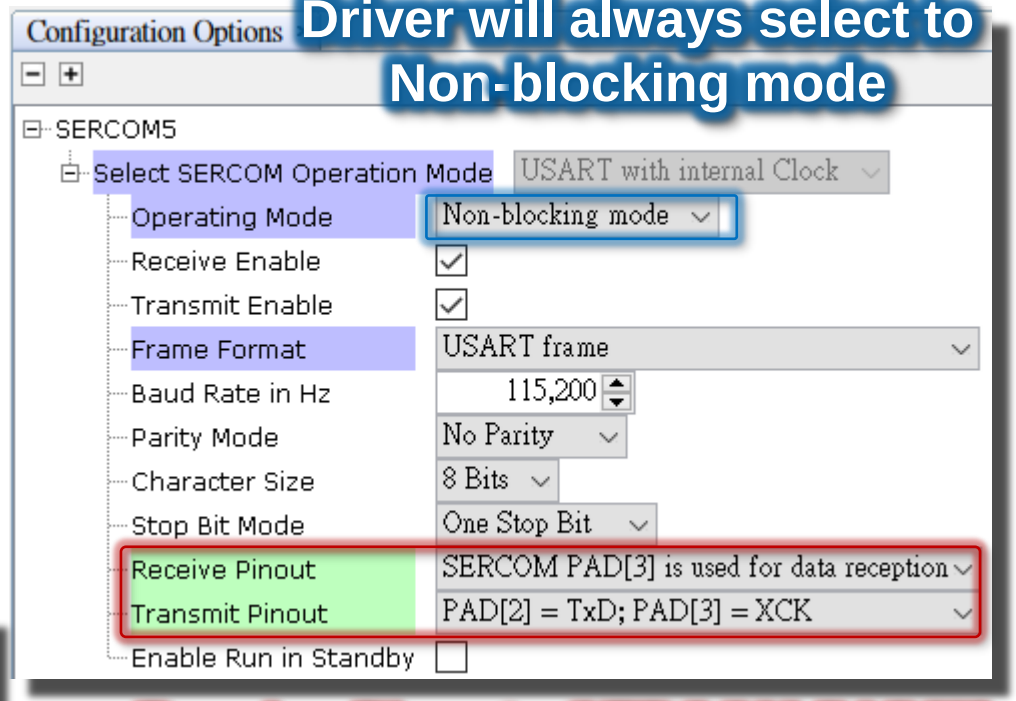


🛠 Lab11 Single-Client UART Communication

- **Configuration of USART driver and PLIB SERCOM5**



DMA to be option of Driver



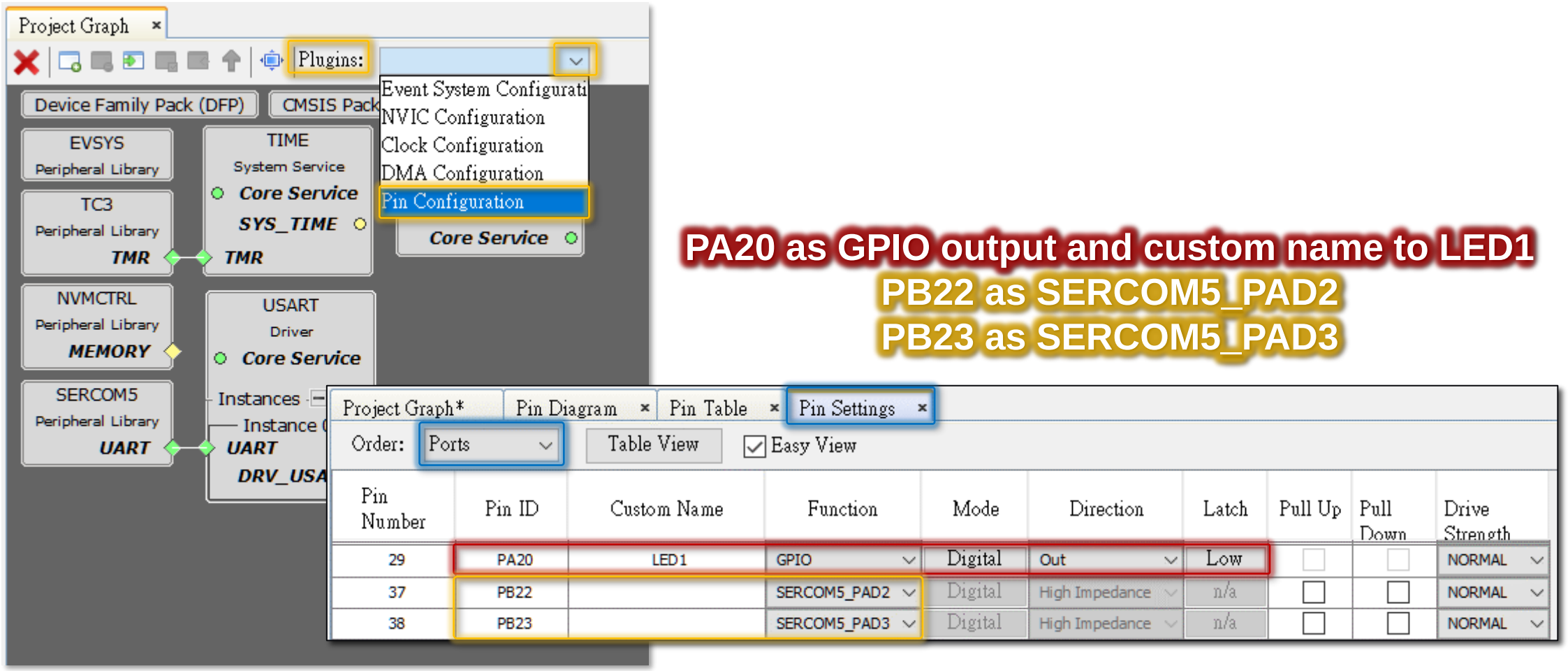
Driver will always select to Non-blocking mode

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

MICROCHIP

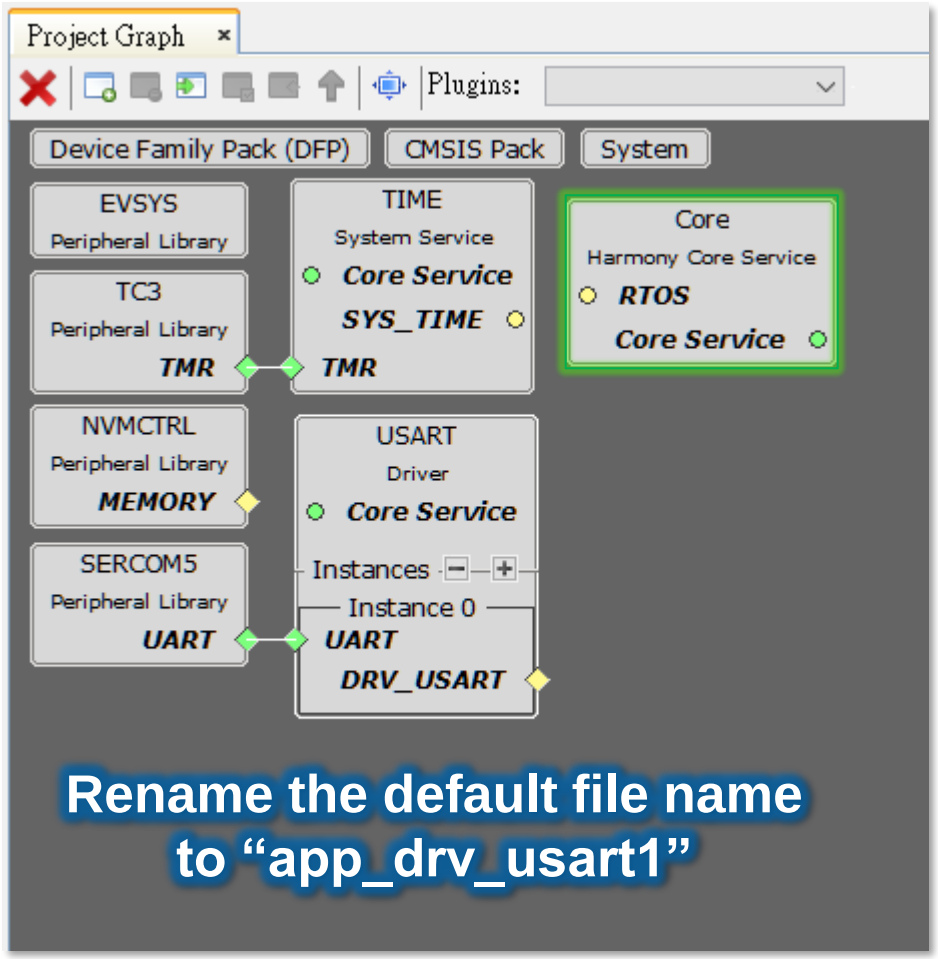
🛠 Lab11 Single-Client UART Communication

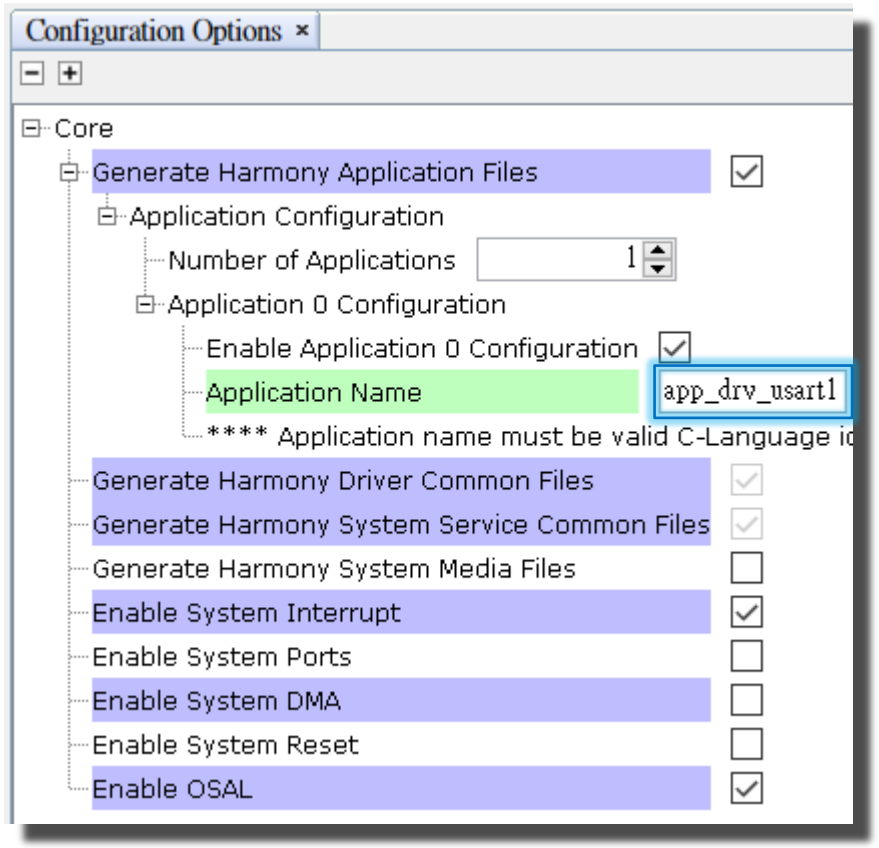
- **Pin Configuration of GPIO PA20(LED1) and PLIB SERCOM5 pads**



🛠 Lab11 Single-Client UART Communication

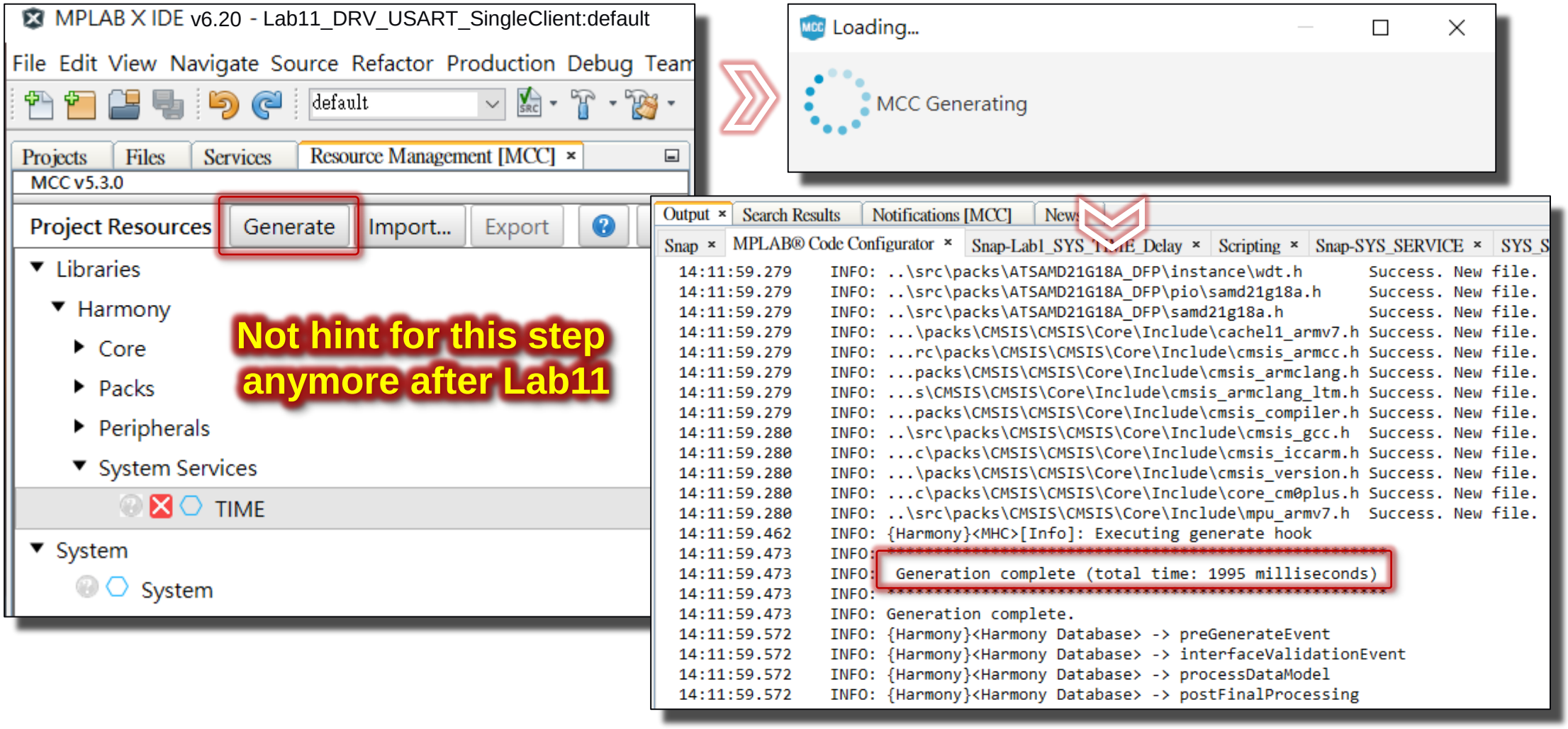
- In Core (Harmony Core Service) to find the default Application Name “app” and rename to “app_drv_usart1” for our USART Driver state machine implement.





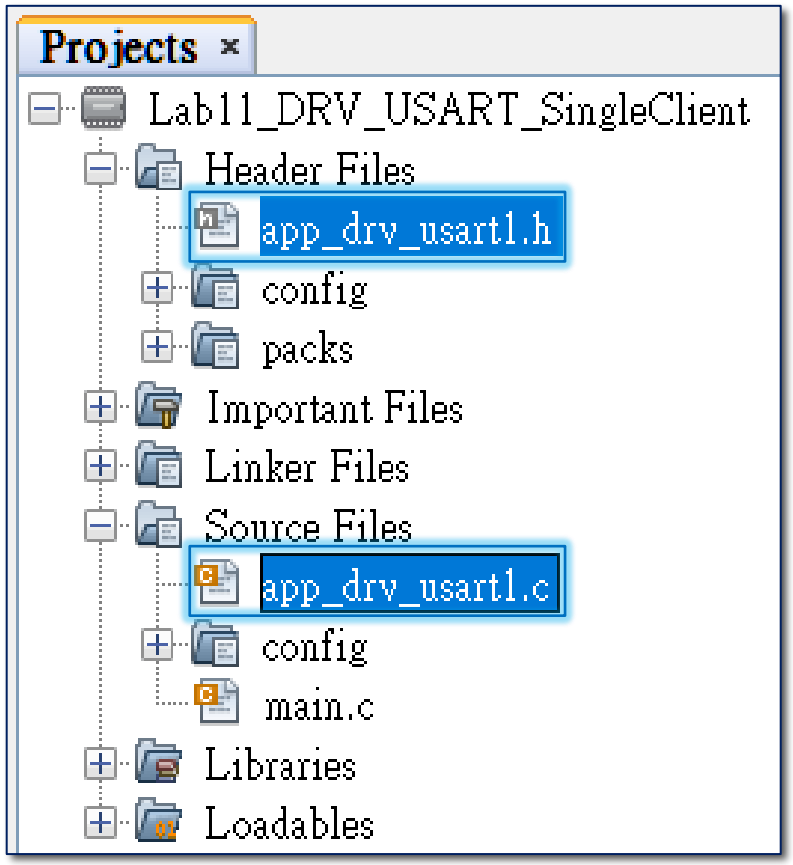
🛠 Lab11 Single-Client UART Communication

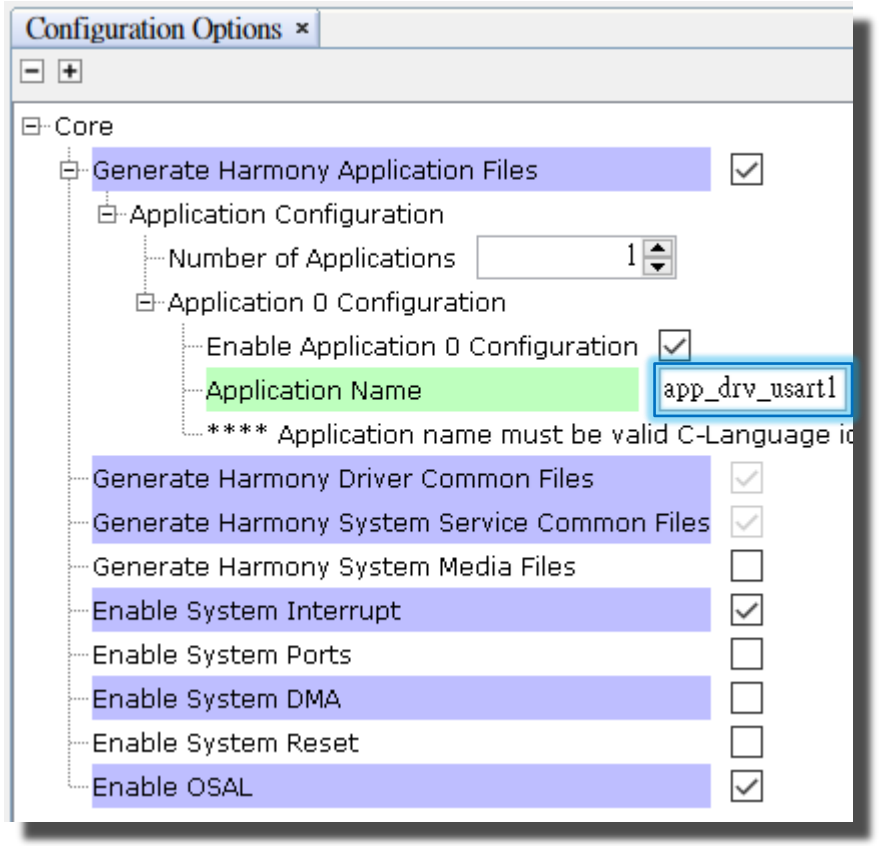
- **Harmony Configuration (Generate Code)**



🛠 Lab11 Single-Client UART Communication

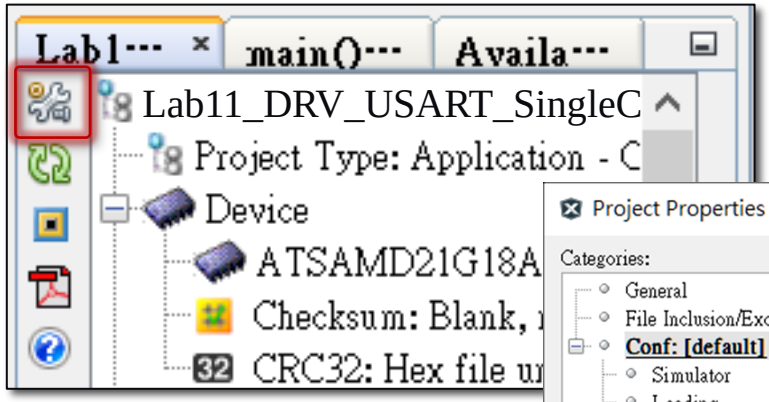
- For Driver based application, most custom implement will finish in the APP file app_drv_usart1.c but not in main.c, let's complete your APP state machine step by step from next page.



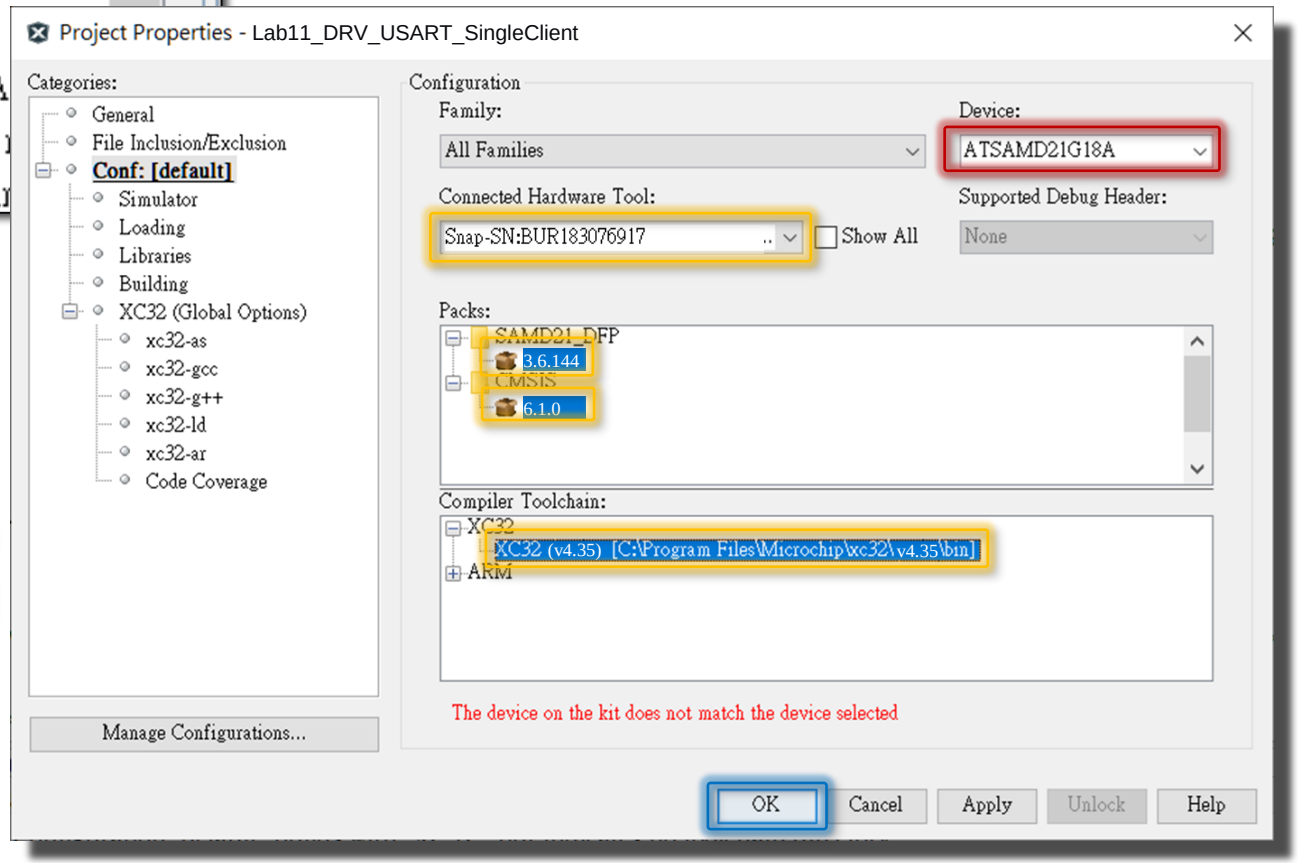


🛠 Lab11 Single-Client UART Communication

- **Configure your new project**



Double confirm the Device : ATSAMD21G18A



Select correct

Debugger

DFP 3.6.144
CMSIS 6.1.0

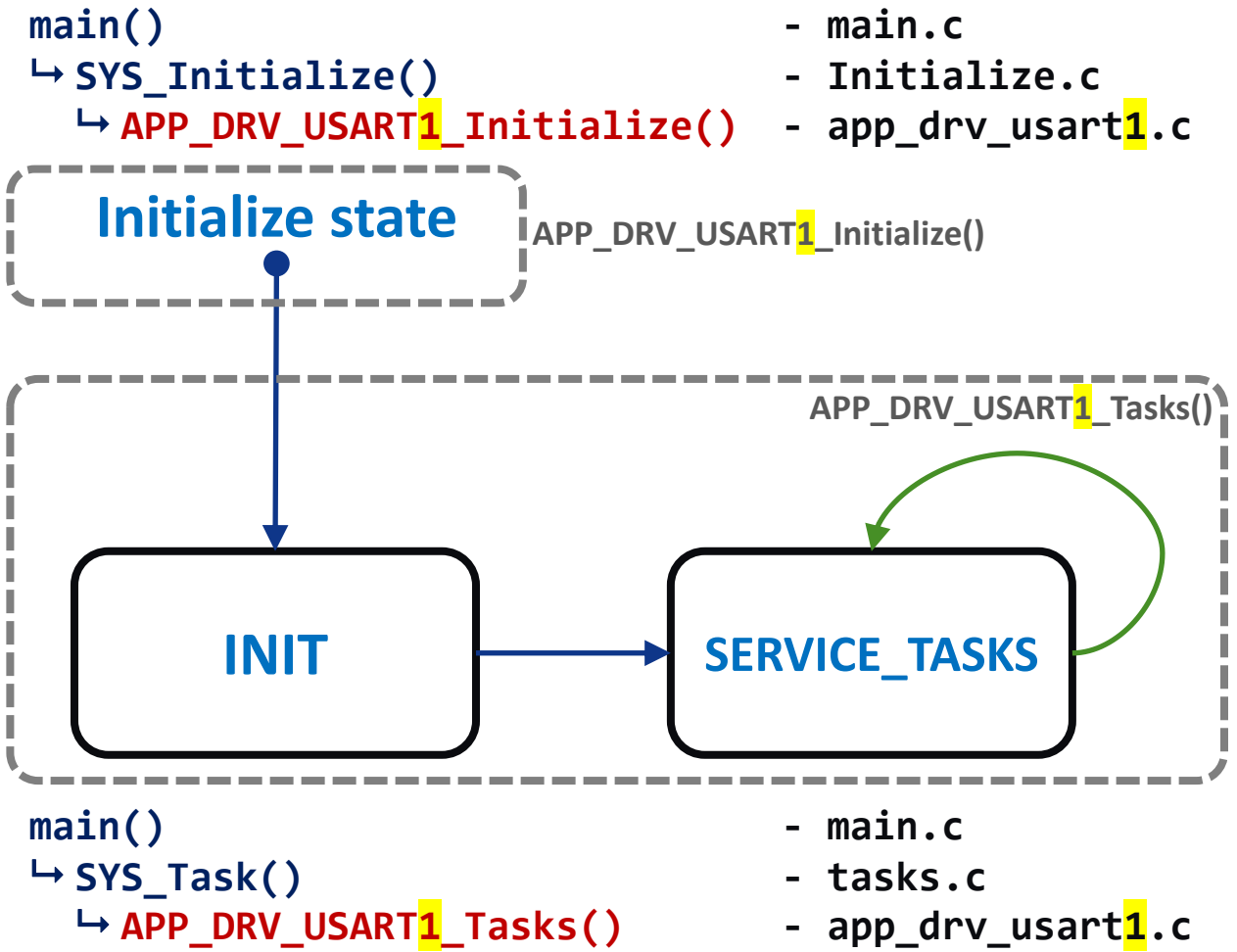
XC32 (v4.45)

Not hint for this step anymore after Lab11

MICROCHIP

🛠 Lab11 Single-Client UART Communication

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

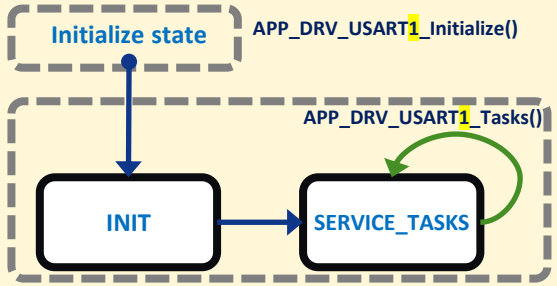


🛠 Lab11 Single-Client UART Communication

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

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

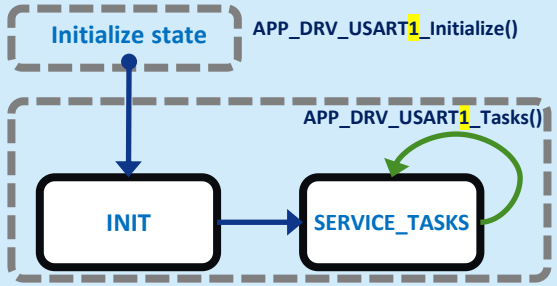
MICROCHIP

🛠 Lab11 Single-Client UART Communication

- Let's check the header file declaration "app_drv_usart1.h"

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_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

MICROCHIP

🛠 Lab11 Single-Client UART Communication

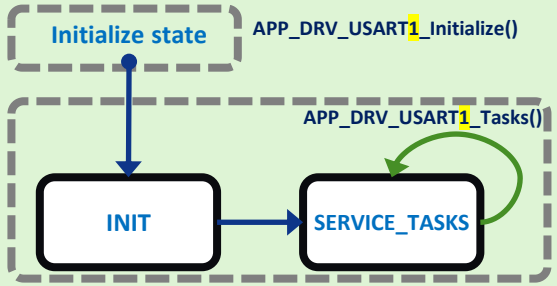
- To check “main.c“ and add LED1 toggle as we did in Lab1

main.c

```
#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);
}
```

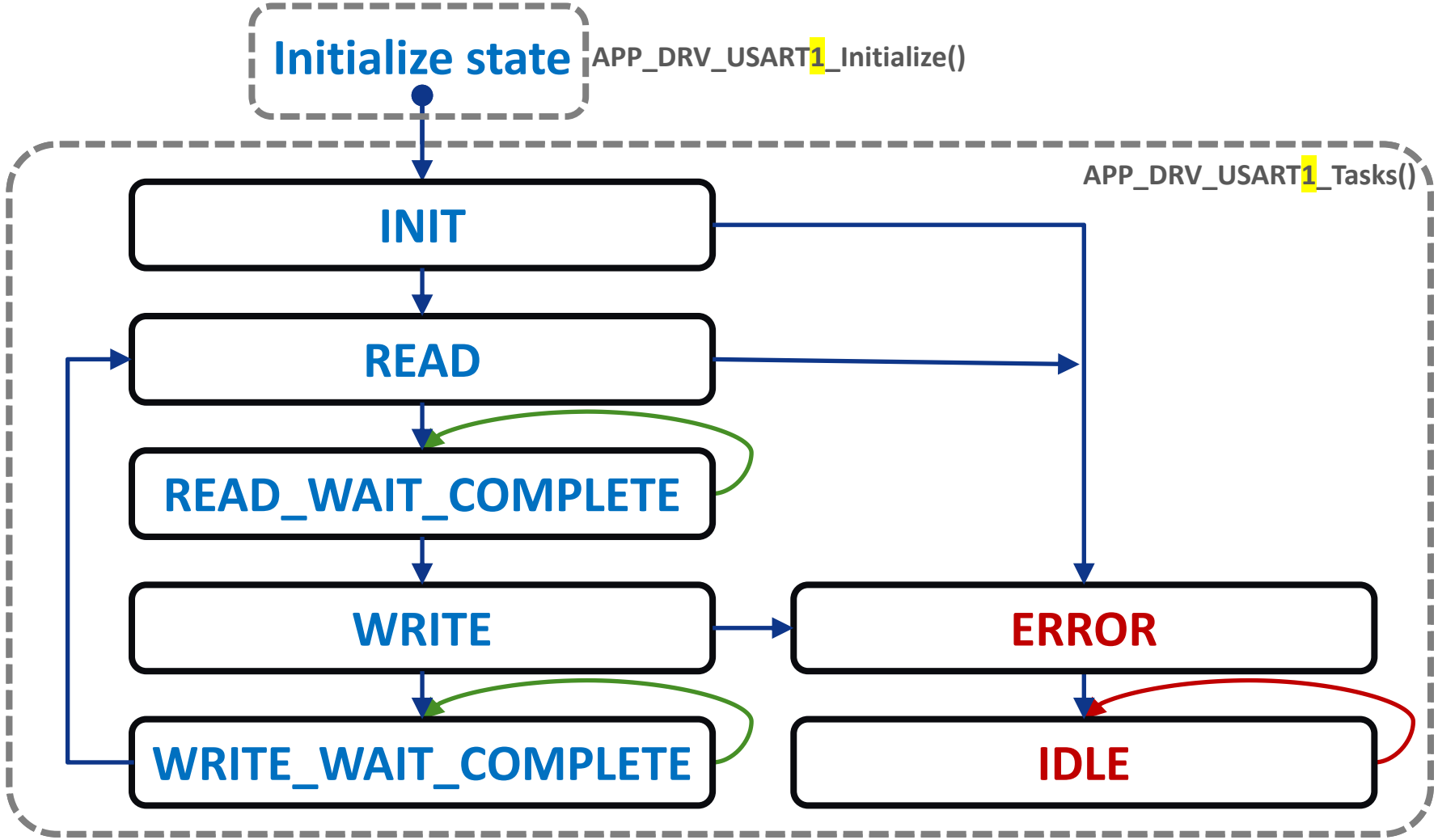
APP_DRV_USART1_Initialize()

APP_DRV_USART1_Tasks()



🛠 Lab11 Single-Client UART Communication

- The new state machine we want to implement in “app_drv_usart1.c” as below.



🛠 Lab11 Single-Client UART Communication

- **Add more states in "app_drv_usart1.h".**

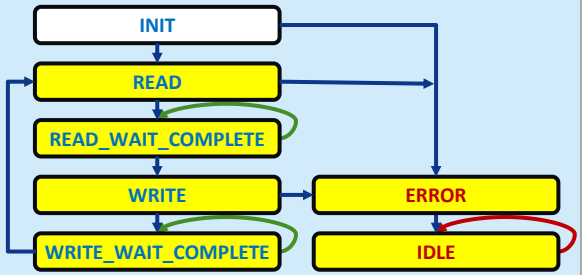
app_drv_usart1.h

```
#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.



MICROCHIP

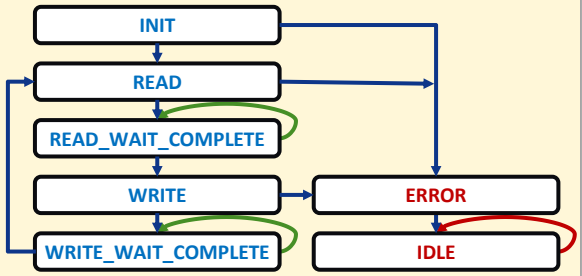
🛠 Lab11 Single-Client UART Communication

- **Construct new states flow of task in “app_drv_usart1.c”.**

app_drv_usart1.c

Overwrite original example code

```
void APP_DRV_USART1_Tasks ( void )
{
    switch ( app_drv_usart1Data.state )
    {
    case APP_DRV_USART1_STATE_INIT:
        app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;
        break;

    case APP_DRV_USART1_STATE_READ:
        app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ_WAIT_COMPLETE;
        break;

    case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
        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;
        break;

    case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
        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:
        break;
    }
}
```



MICROCHIP

🛠 Lab11 Single-Client UART Communication

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

Keep this line

app_drv_usart1.c

```
#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[10];


// *****************************************************************************
/* 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“

MICROCHIP

🛠 Lab11 Single-Client UART Communication

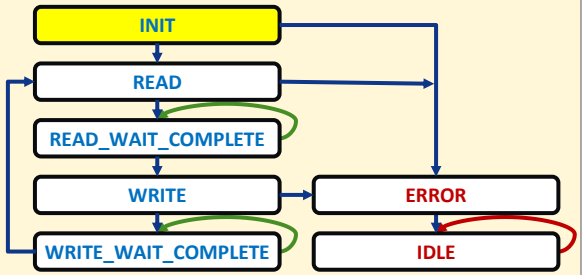
- **Add Driver Open, Write and Transfer Status Polling Operation in state INIT.**

app_drv_usart1.c

```
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:
        ...
```

Keep this line

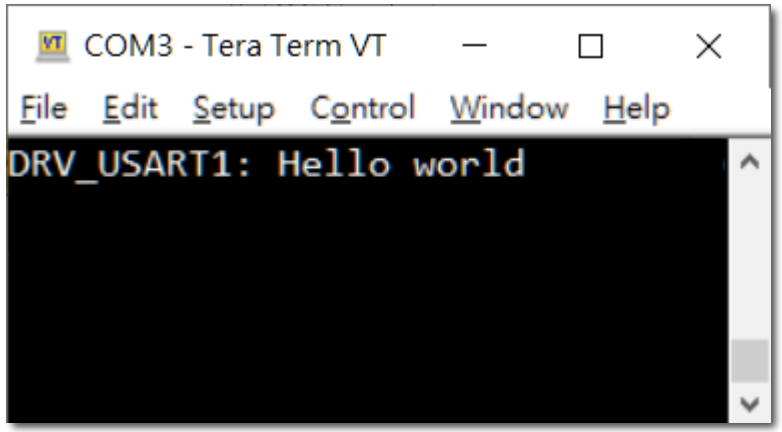
Transfer Status Polling operation to check status of USART write

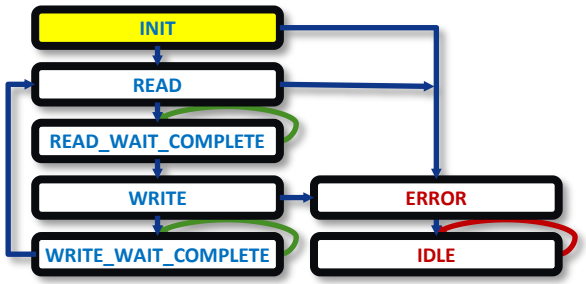


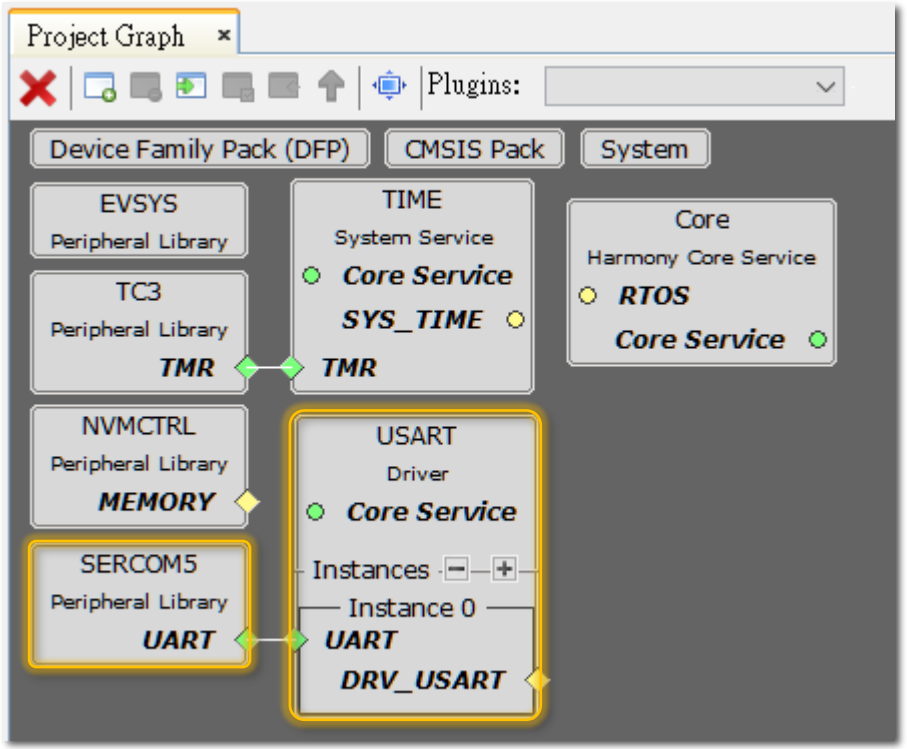
MICROCHIP

🛠 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`.







MICROCHIP

🛠 Lab11 Single-Client UART Communication

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

app_drv_usart1.c

```
// 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 )
            {}

            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

Register Callback Function in INIT state

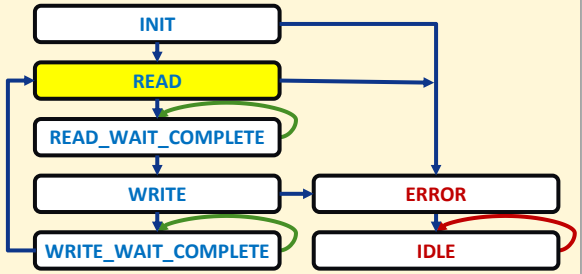
MICROCHIP

🛠 Lab11 Single-Client UART Communication

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

app_drv_usart1.c

Manage Read Complete flag while Read Transfer Complete Events




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

     case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:
```

MICROCHIP

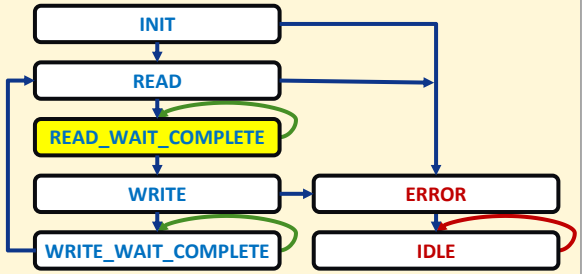
🛠 Lab11 Single-Client UART Communication

- Implement state **READ_WAIT_COMPLETE**.

app_drv_usart1.c

```
void Uart1TransferHandler( DRV_USART_BUFFER_EVENT event,
                           DRV_USART_BUFFER_HANDLE bufferHandle, ...
{
    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

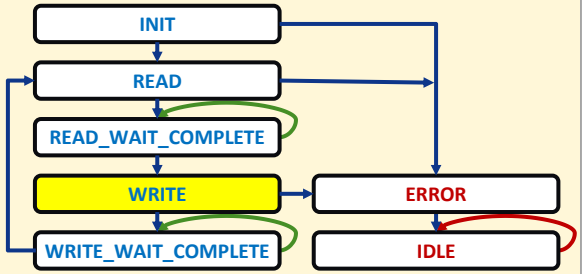
MICROCHIP

🛠 Lab11 Single-Client UART Communication

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

app_drv_usart1.c

Manage Write Complete flag while Write Transfer Complete Events




```
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:
        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;
        }
        break;

     case APP_DRV_USART1_STATE_WRITE_WAIT_COMPLETE:
```

Keep this line

Keep this line

MICROCHIP

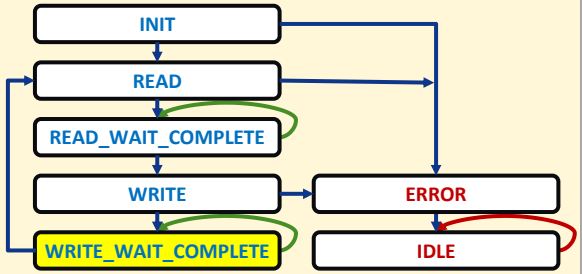
🛠 Lab11 Single-Client UART Communication

- Implement state **WRITE_WAIT_COMPLETE**

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;
            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:
```

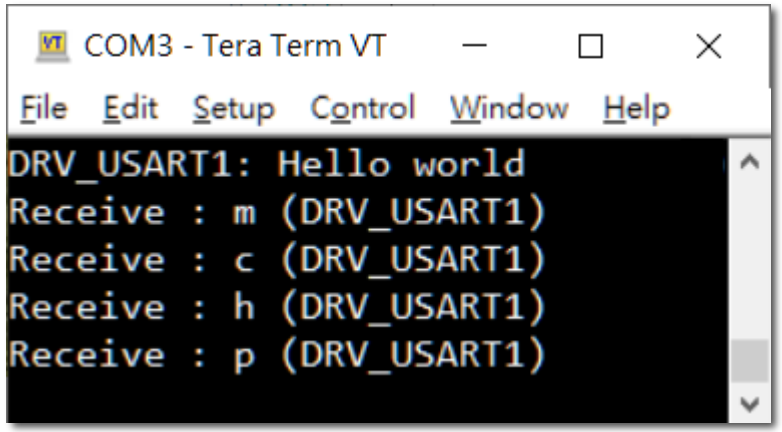
Keep this line

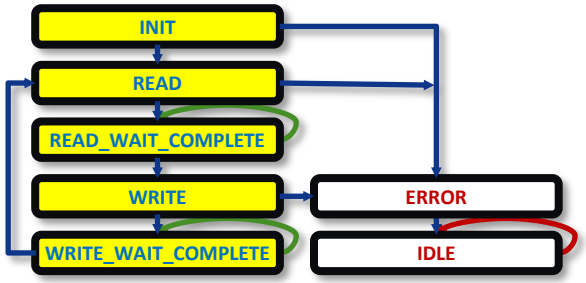


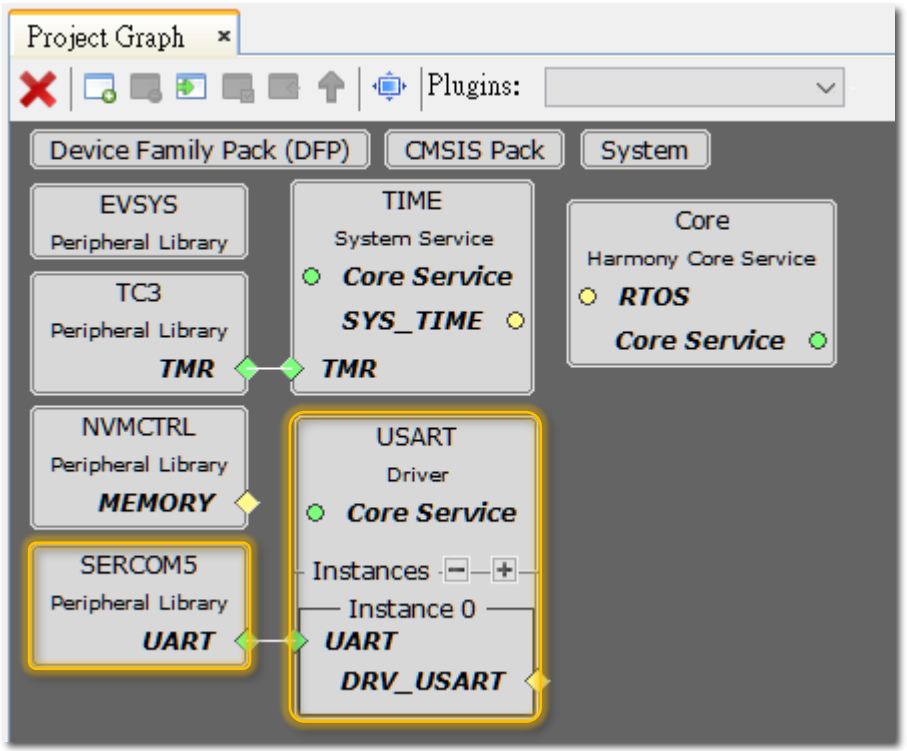
State non blocking while waiting Write Complete

🛠 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.







MICROCHIP

🛠 Lab11 Single-Client UART Communication

- **Implement a C STDIO printf() like function to output Lab debug message.**

app_drv_usart1.c

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

DRV_HANDLE Uart1Handle;
DRV_USART_BUFFER_HANDLE Uart1WriteTransferHandle;
char Uart1WriteBuf[50];
...

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, ... ) { ... }
```

The polling method is good for implement in easily

MICROCHIP

🛠 Lab11 Single-Client UART Communication

- **Replace "DRV_USART1: Hello World" output by use USART1printf()**

app_drv_usart1.c

```
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
        {
            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.**

app_drv_usart1.h

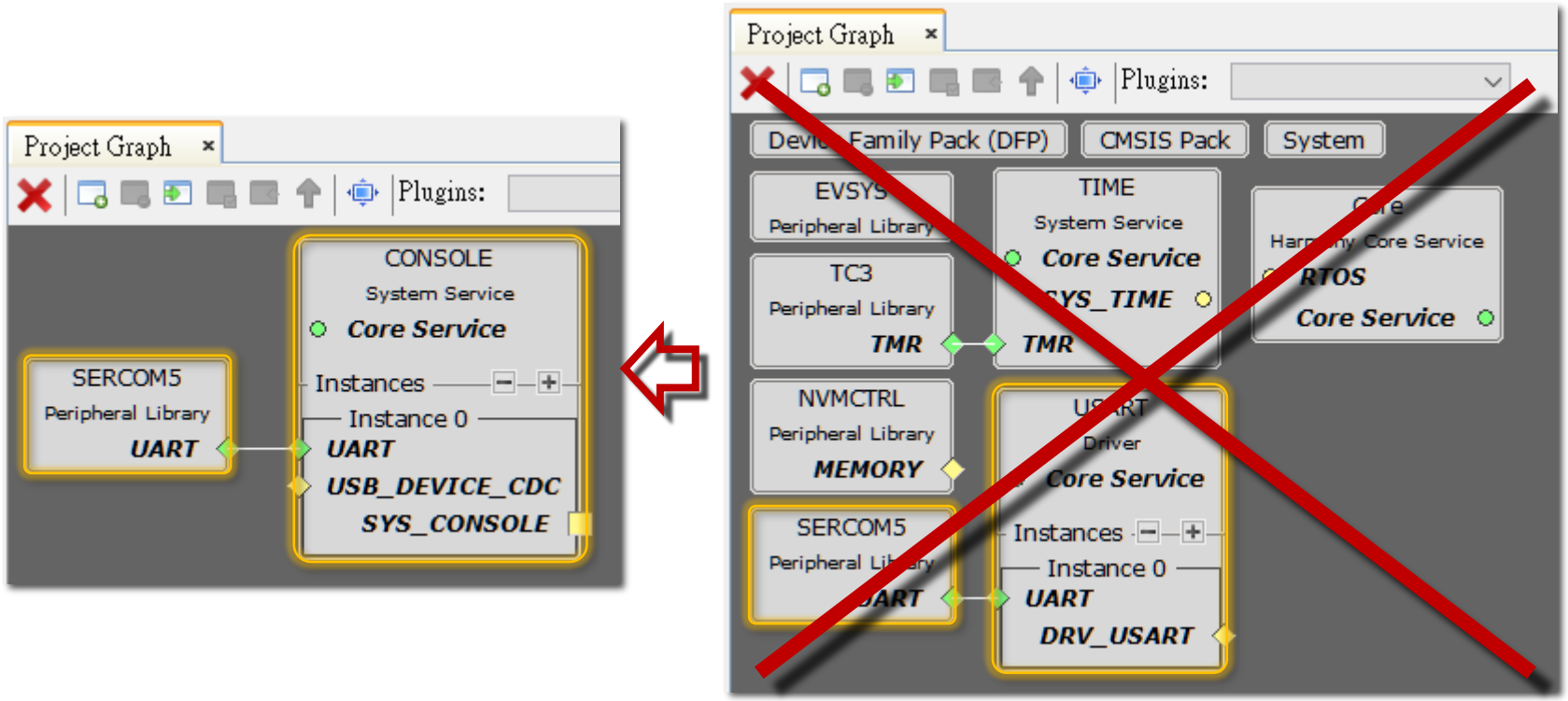
```
void APP_DRV_USART1_Initialize ( void );
void APP_DRV_USART1_Tasks( void );

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

//DOM-IGNORE-BEGIN
```

MICROCHIP

🛠 Lab11 Single-Client UART Communication

Discuss 1

- **What we did ?**
  If you want implement a UART console, why not use the CONSOLE System Service ?



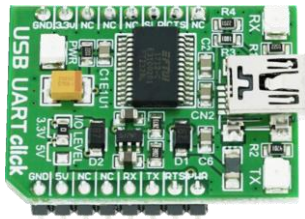
🛠 Lab 12
Multi-Instance UART Communication

USART Driver Library

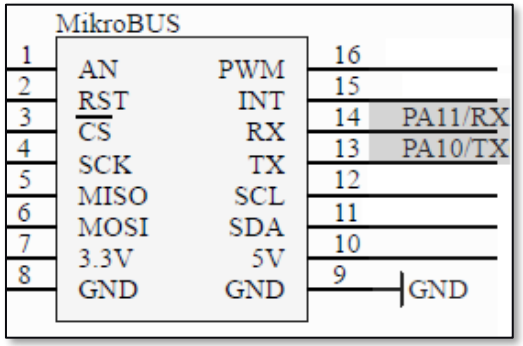
MICROCHIP

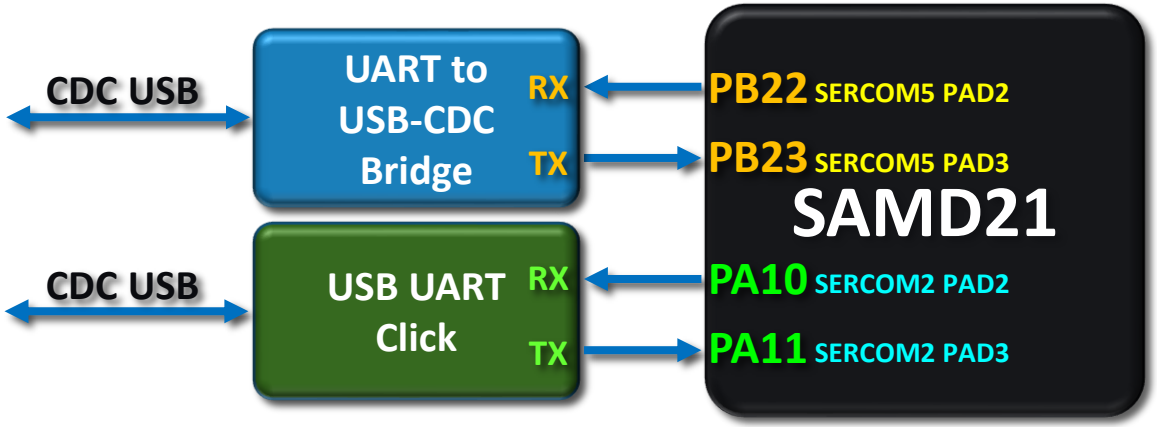
🛠 Lab12 Multi-Instance UART Communication

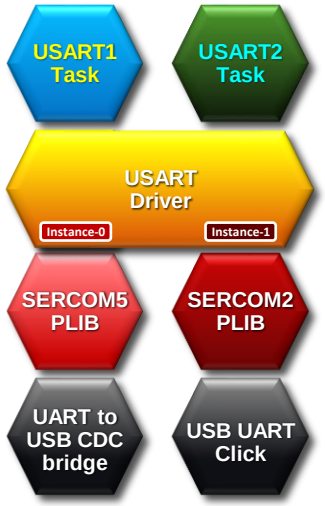
- Try add instance 1 in USART Driver to implement another UART console from mikroBUS socket through PLIB SERCOM2 UART.
- You can prepare a USB UART Click EVB for UART-CDC bridge.
- Create a new APP state machine app_drv_usart2 in Core Service.
- Try output the message "USART2 Driver: Hello World" while 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?)"



mikroBUS
USB UART click



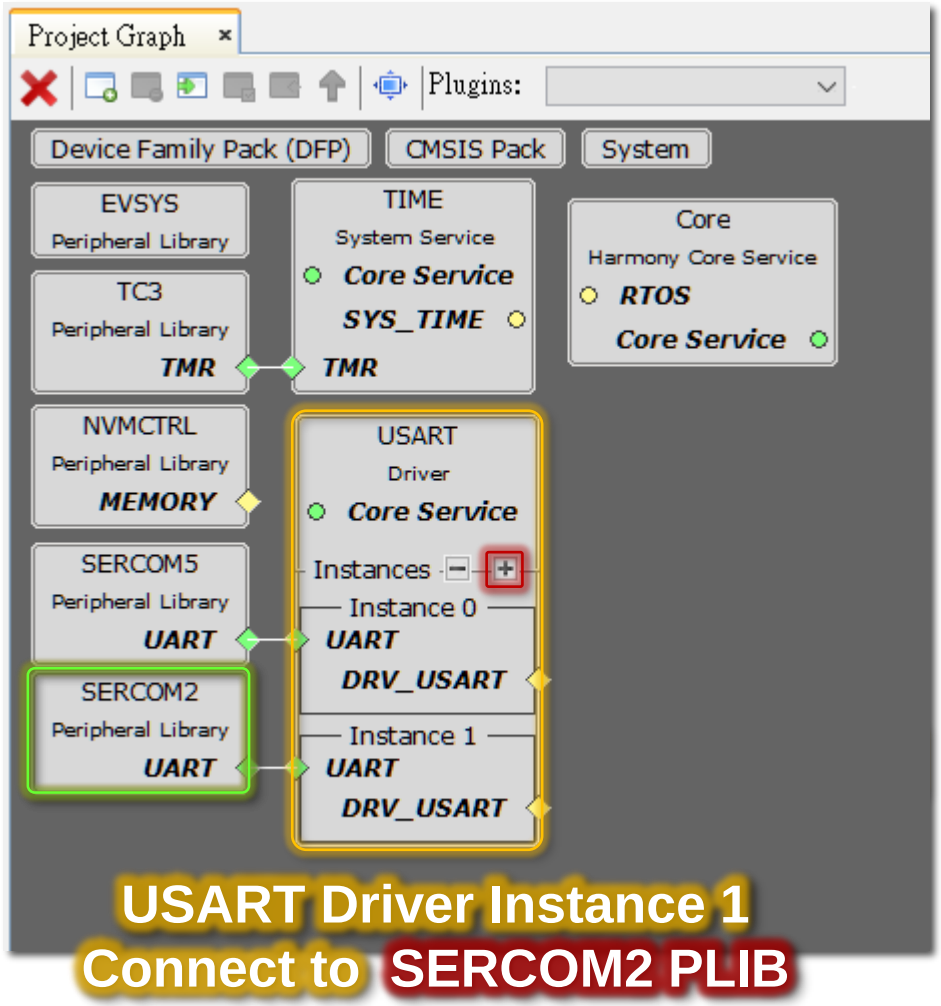


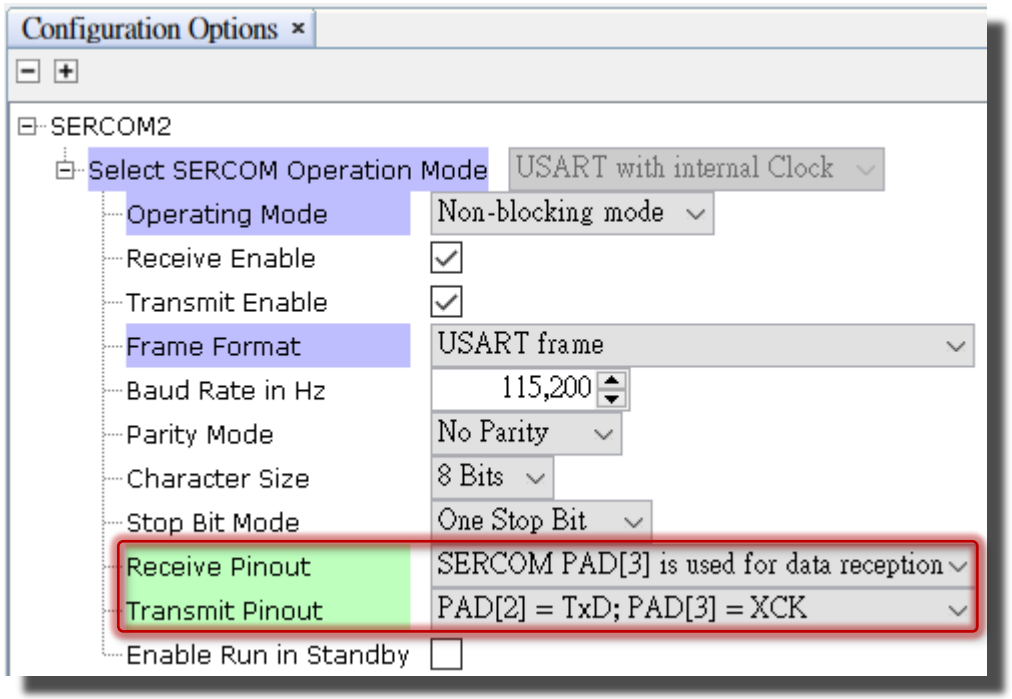


MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

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



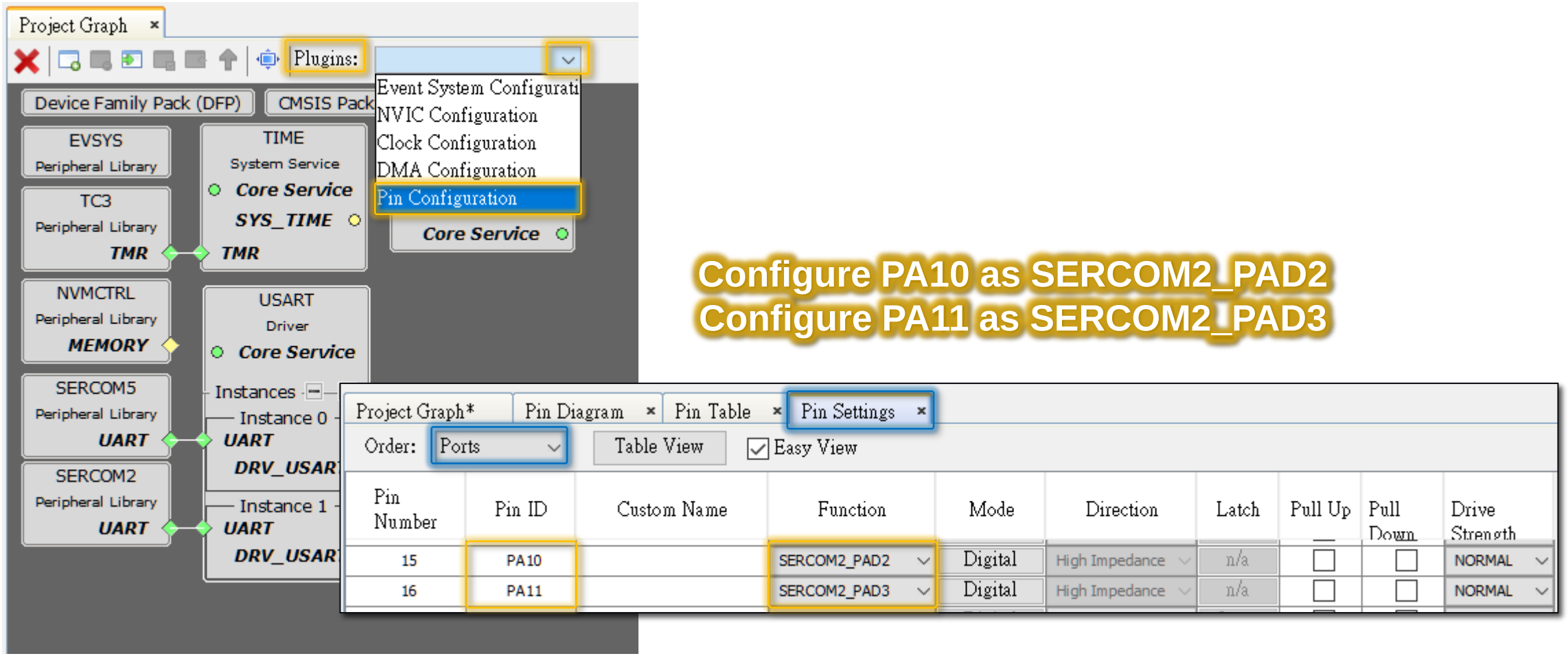


Configuration of SERCOM2
Receive Pinout : SERCOM PAD[3]
Transmit Pinout: SERCOM PAD[2]

MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

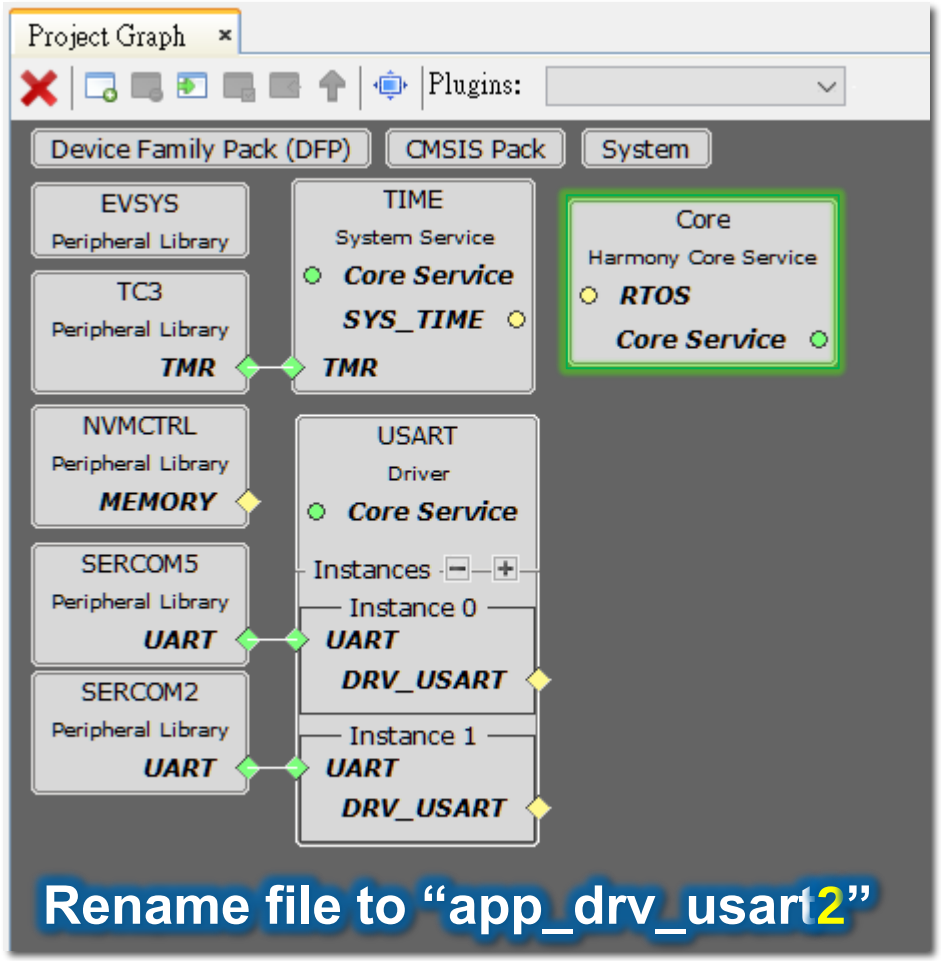
- **Pin configuration of PLIB SERCOM2 pads.**

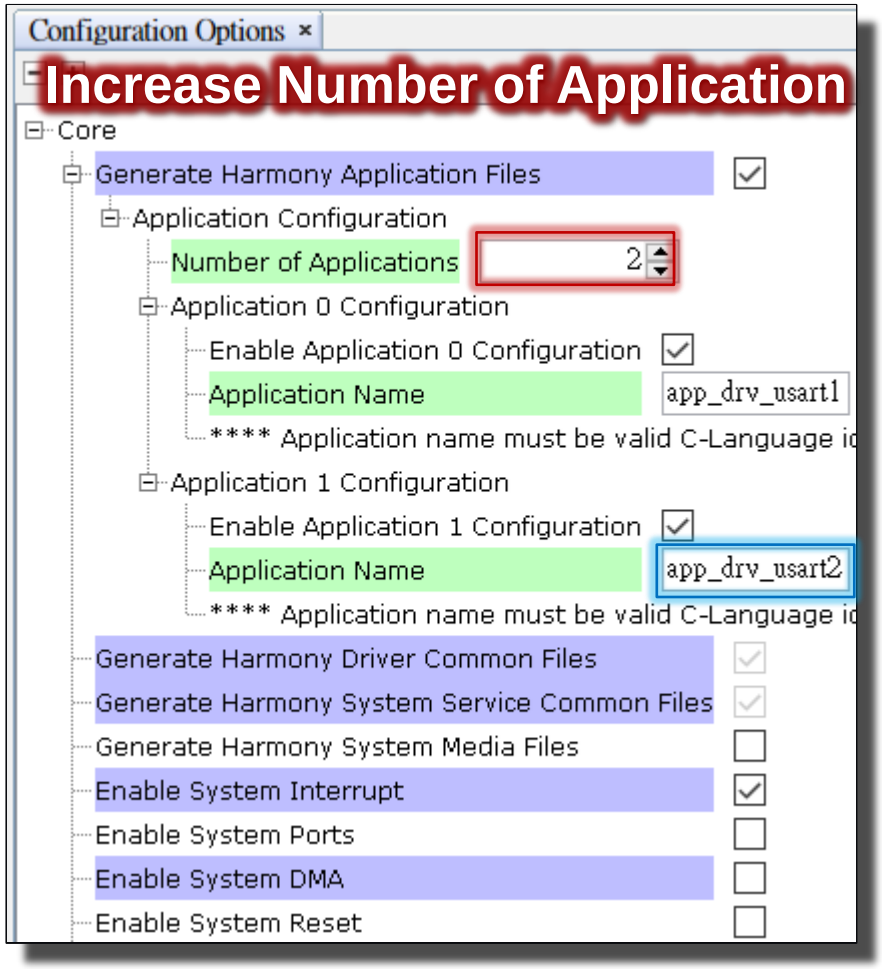


MICROCHIP

🛠 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".

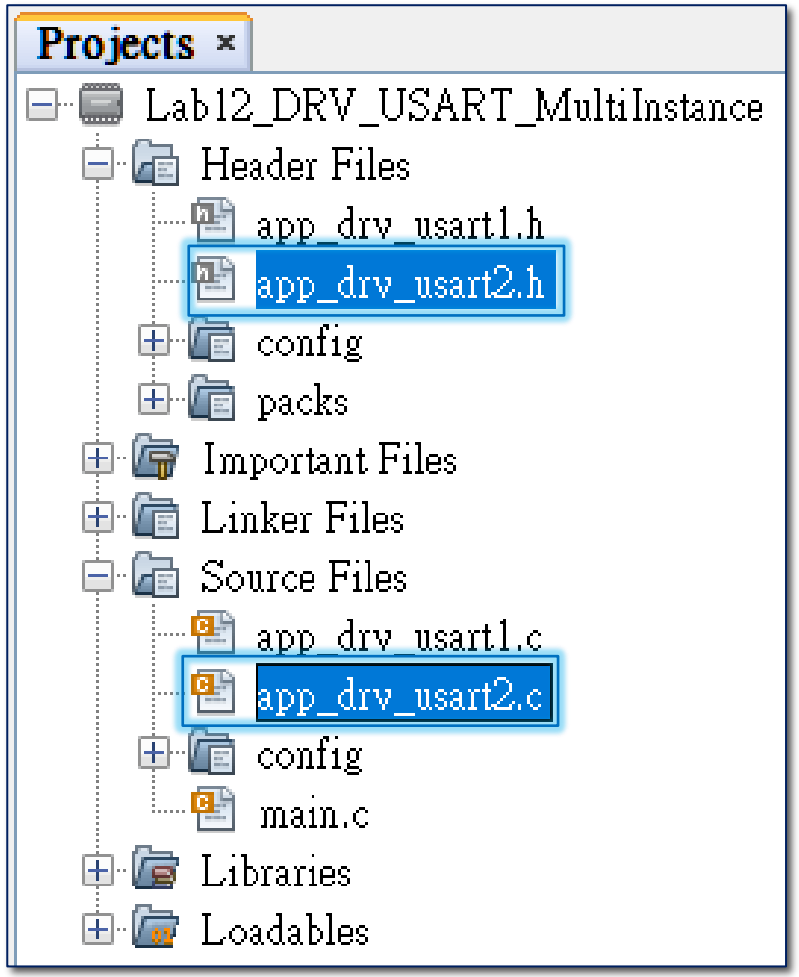


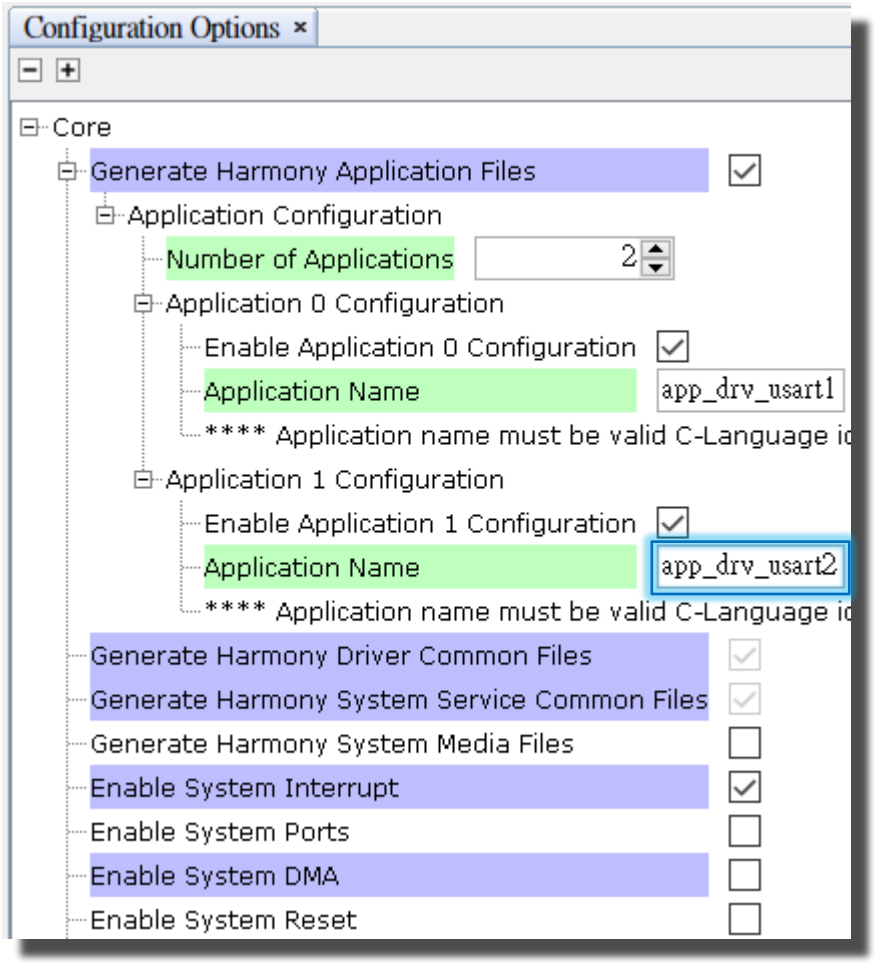


MICROCHIP

🛠 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.**

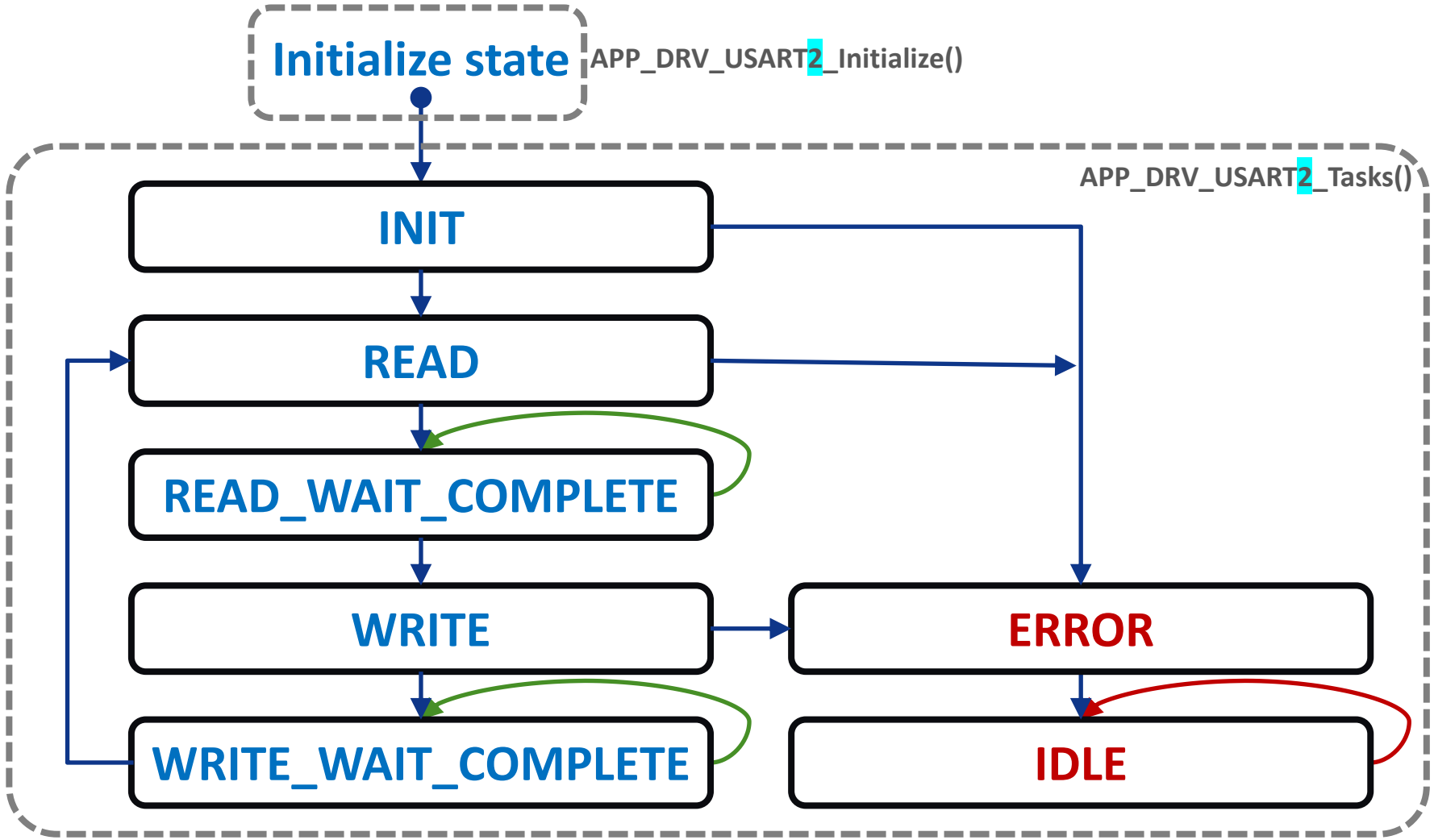




MICROCHIP

🛠 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”.**

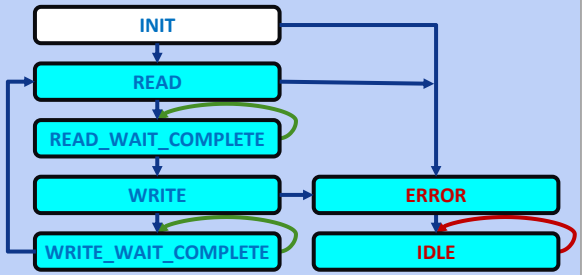
app_drv_usart2.h

```
#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.



MICROCHIP

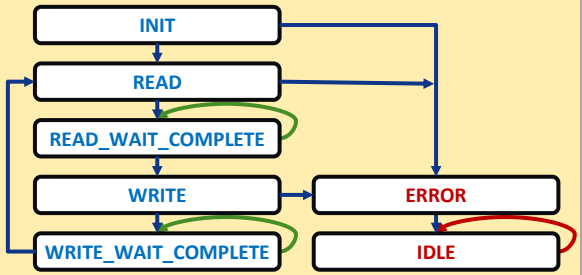
🛠 Lab12 Multi-Instance UART Communication

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

app_drv_usart2.c

Overwrite original example code

```
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;
    }
}
```



MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

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

Keep this line

app_drv_usart2.c

```
#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[10];


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

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  ...
*/

APP_DRV_USART2_DATA app_drv_usart2Data;
```

MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

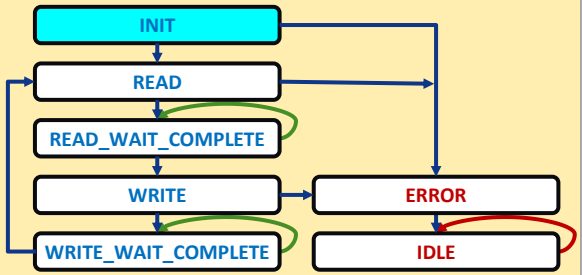
- **Add Driver Open, Write and Transfer Status Polling Operation.**

app_drv_usart2.c

```
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;

    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:
        ...
```

Keep this line

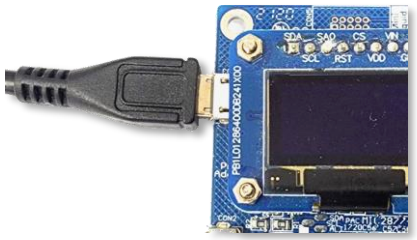
Transfer Status Polling operation to check status of USART write

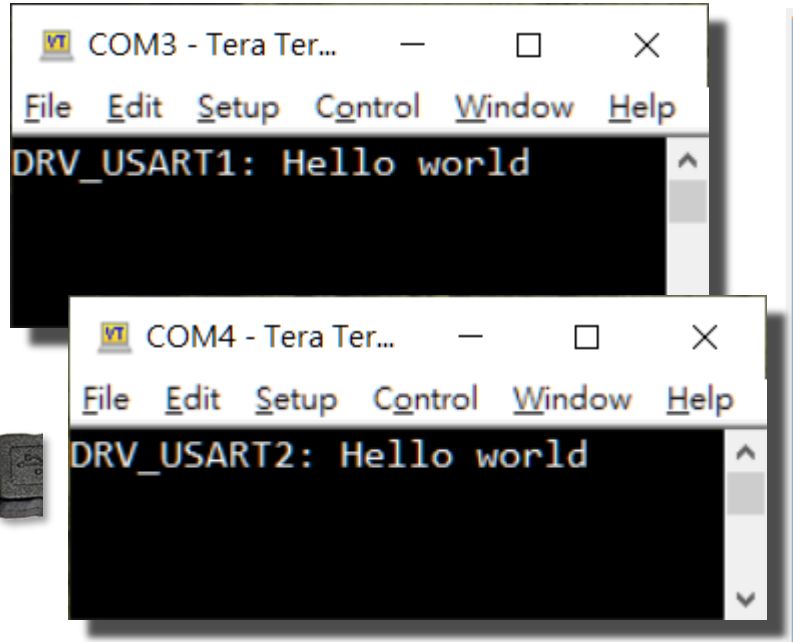


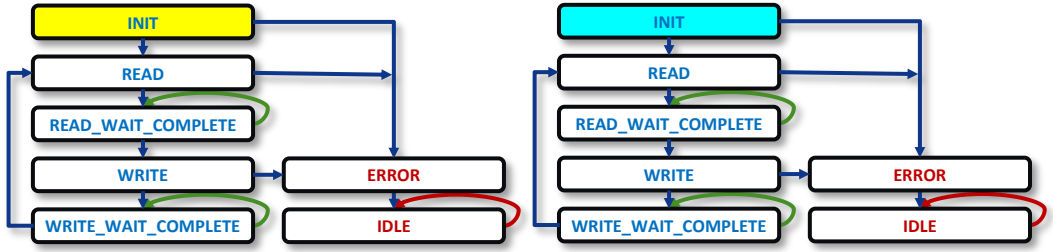
MICROCHIP

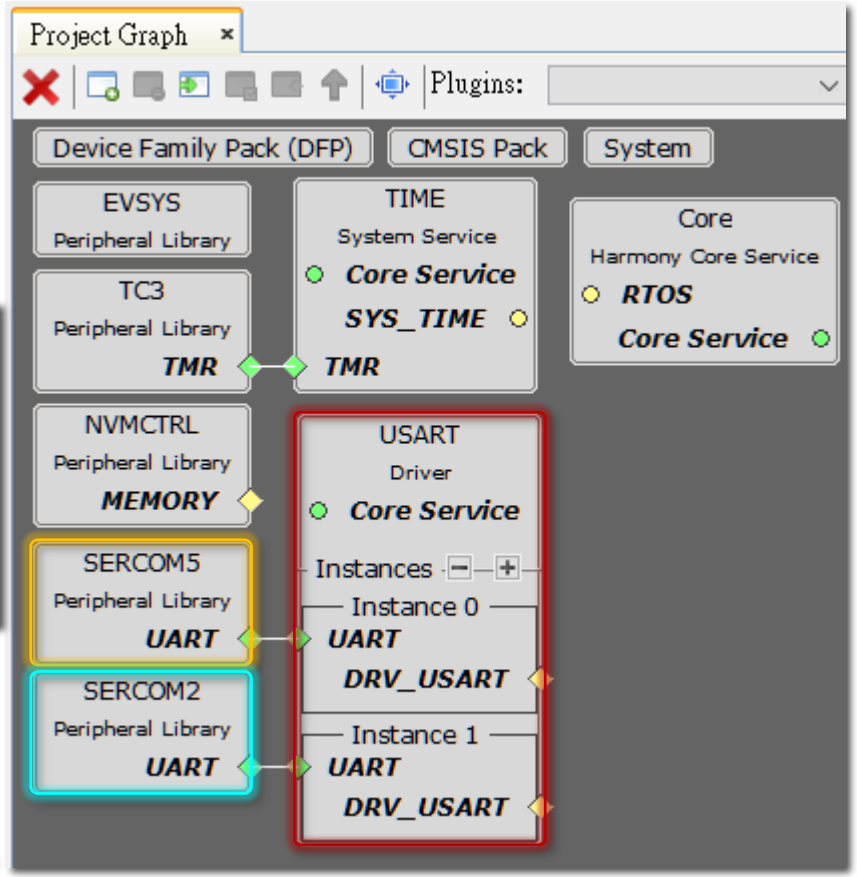
🛠 Lab12 Multi-Instance UART Communication

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









MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

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

app_drv_usart2.c

```
// 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 )
            {}

            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

Register Callback Function in INIT state

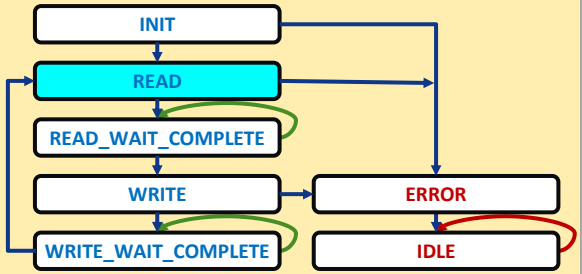
MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

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

app_drv_usart2.c

Manage Read Complete flag while Read Transfer Complete Events




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

     case APP_DRV_USART2_STATE_READ_WAIT_COMPLETE:
```

MICROCHIP

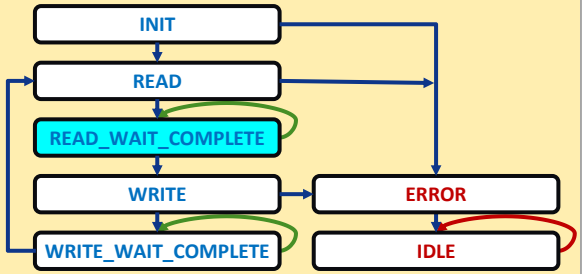
🛠 Lab12 Multi-Instance UART Communication

- Implement state **READ_WAIT_COMPLETE**.

app_drv_usart2.c

```
void Uart2TransferHandler( DRV_USART_BUFFER_EVENT event,
                           DRV_USART_BUFFER_HANDLE bufferHandle, ...
{
    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

MICROCHIP

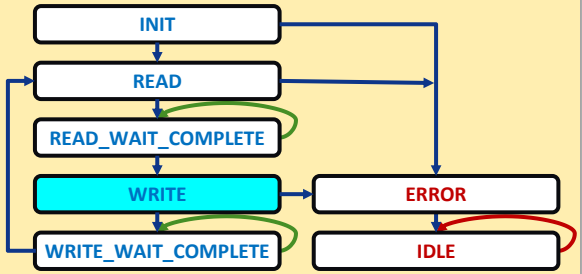
🛠 Lab12 Multi-Instance UART Communication

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

app_drv_usart2.c

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

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;
        }
Keep this line  break;

     case APP_DRV_USART2_STATE_WRITE_WAIT_COMPLETE:
```

Manage Write Complete flag while Write Transfer Complete Events



MICROCHIP

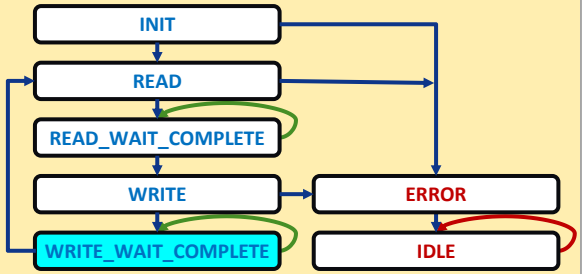
🛠 Lab12 Multi-Instance UART Communication

- Implement state **WRITE_WAIT_COMPLETE**

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;
            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:
```

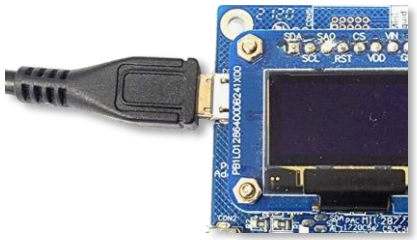
Keep this line

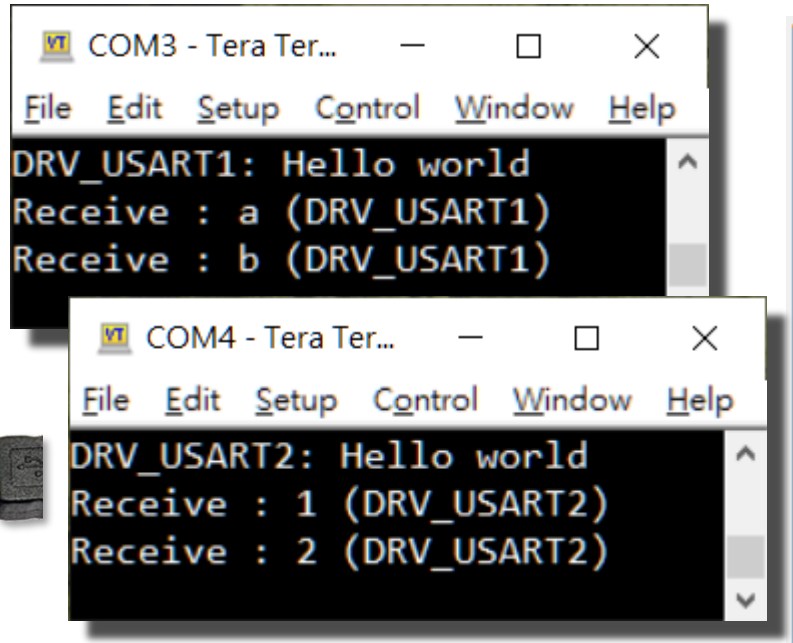


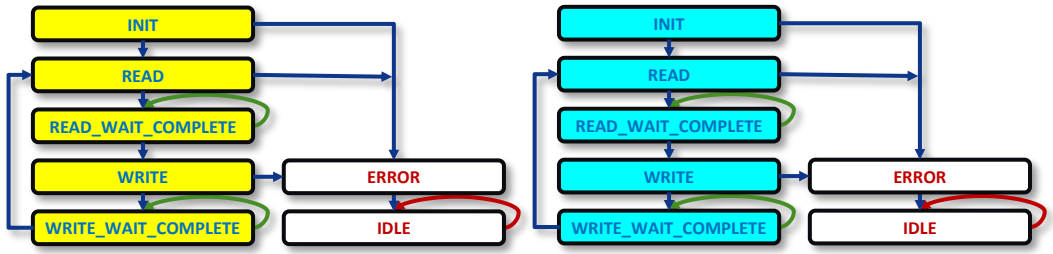
State non blocking while waiting Write Complete

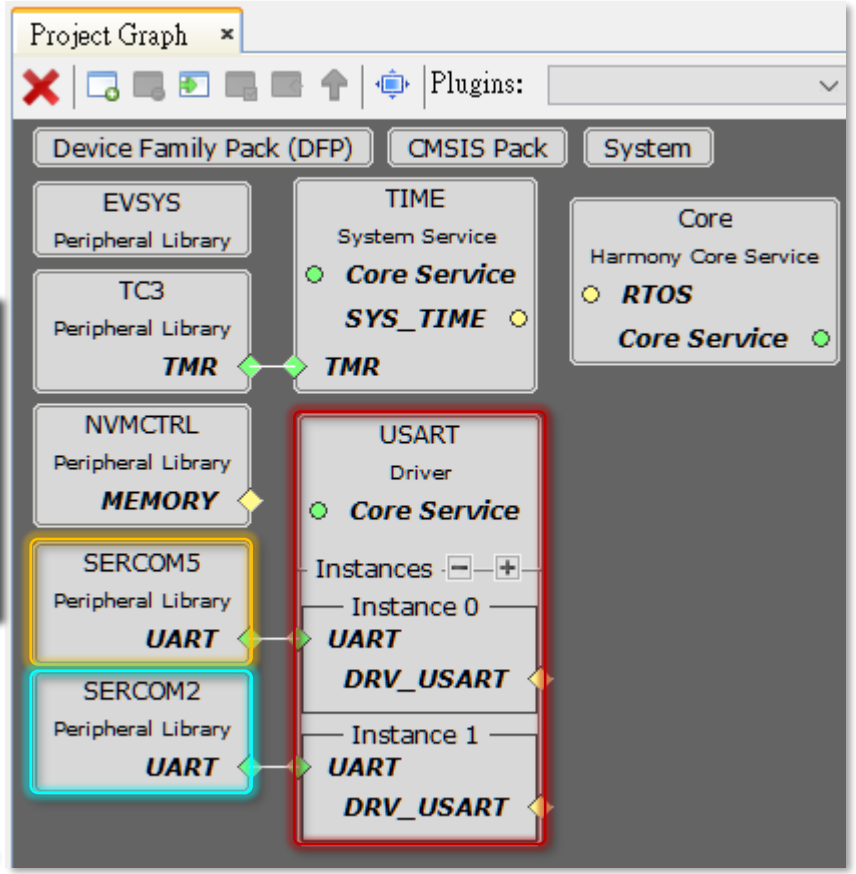
🛠 Lab12 Multi-Instance UART Communication

- 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.





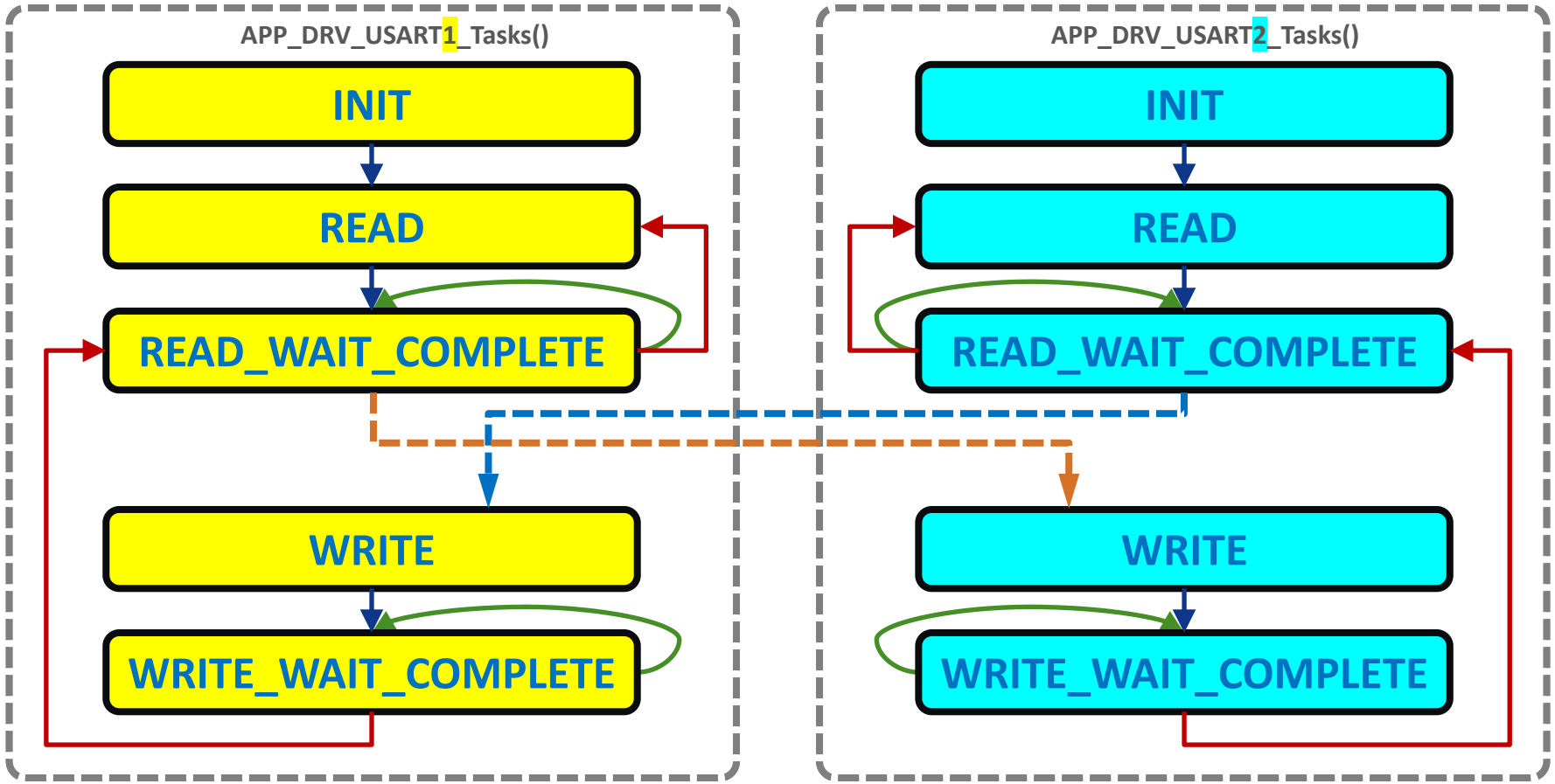




MICROCHIP

🛠 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**

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[10];

...
```

Extern USART1 state and variable

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

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[10];


...
```

Extern USART2 state and variable

MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

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

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>

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

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

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

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>

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

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

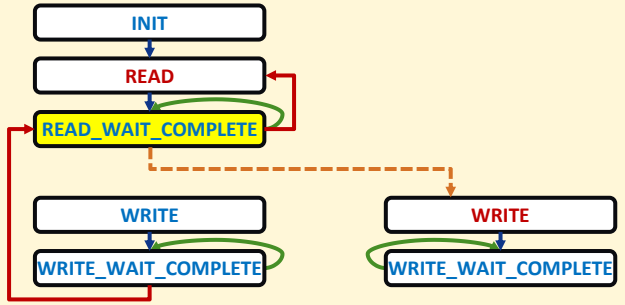
MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

- **Modify state READ_WAIT_COMPLETE in "app_drv_usart1.c".**

app_drv_usart1.c

```
void Uart1TransferHandler( DRV_USART_BUFFER_EVENT event,
                           DRV_USART_BUFFER_HANDLE bufferHandle, ...
{
    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;
Modify this Line  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:
```



State self to READ for next receive

State USART2 to WRITE for message output

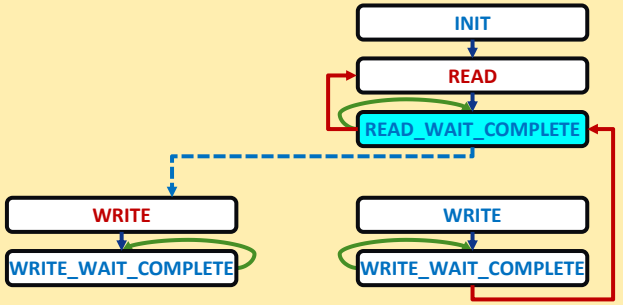
MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

- **Modify state READ_WAIT_COMPLETE in "app_drv_usart2.c".**

app_drv_usart2.c

```
void Uart2TransferHandler( DRV_USART_BUFFER_EVENT event,
                           DRV_USART_BUFFER_HANDLE bufferHandle, ...
{
    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

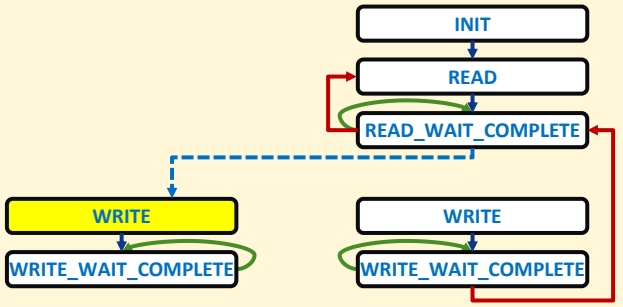
MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

- **Modify state WRITE in "app_drv_usart1.c".**

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

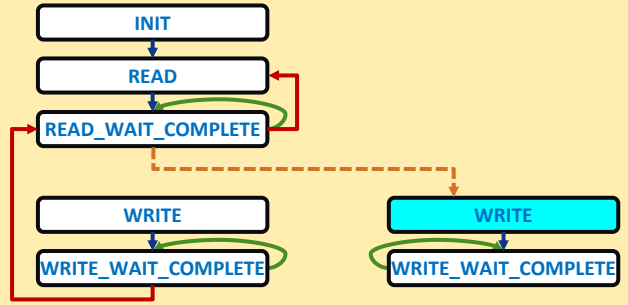
MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

- **Modify state WRITE in "app_drv_usart2.c".**

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

MICROCHIP

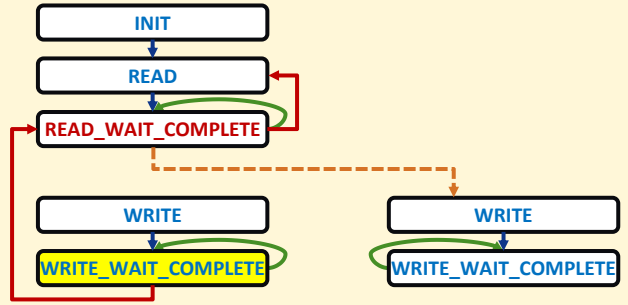
🛠 Lab12 Multi-Instance UART Communication

- **Modify state WRITE_WAIT_COMPLETE in "app_drv_usart1.c".**

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;
            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:
```

Modify this Line



Go READ_WAIT_COMPLETE for next receive complete

MICROCHIP

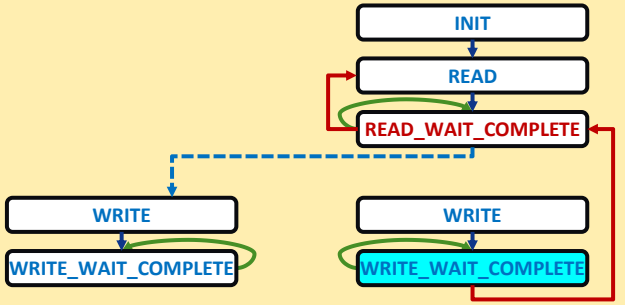
🛠 Lab12 Multi-Instance UART Communication

- **Modify state WRITE_WAIT_COMPLETE in "app_drv_usart2.c".**

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;
            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:
```

Modify this Line

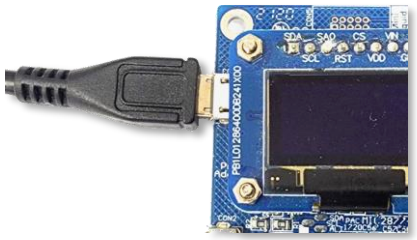


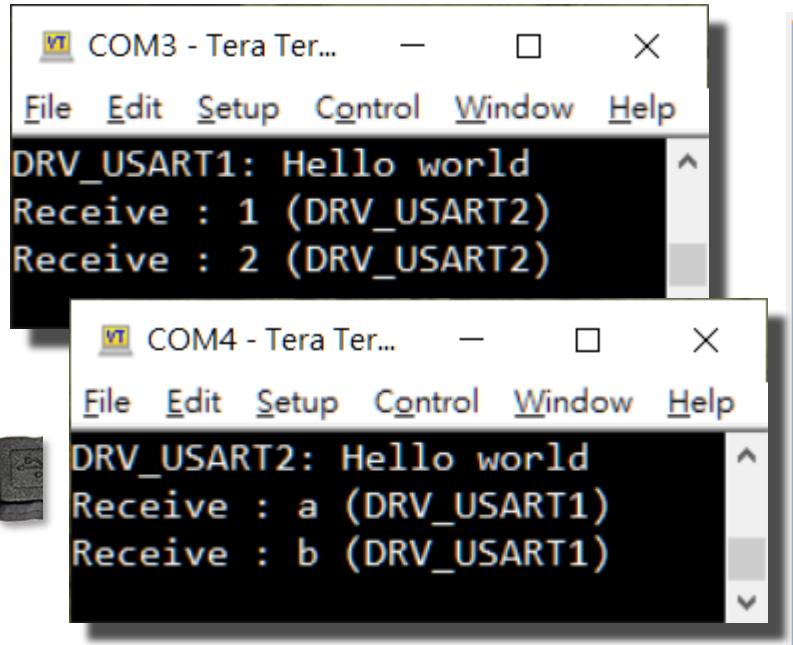
Go READ_WAIT_COMPLETE for next receive complete

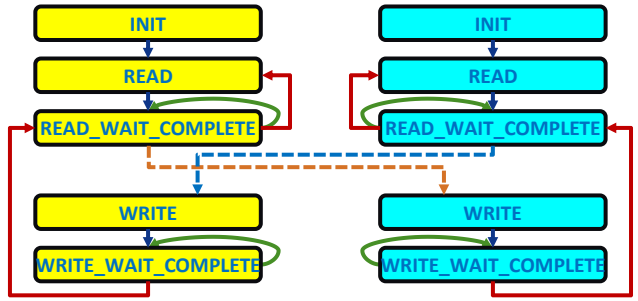
MICROCHIP

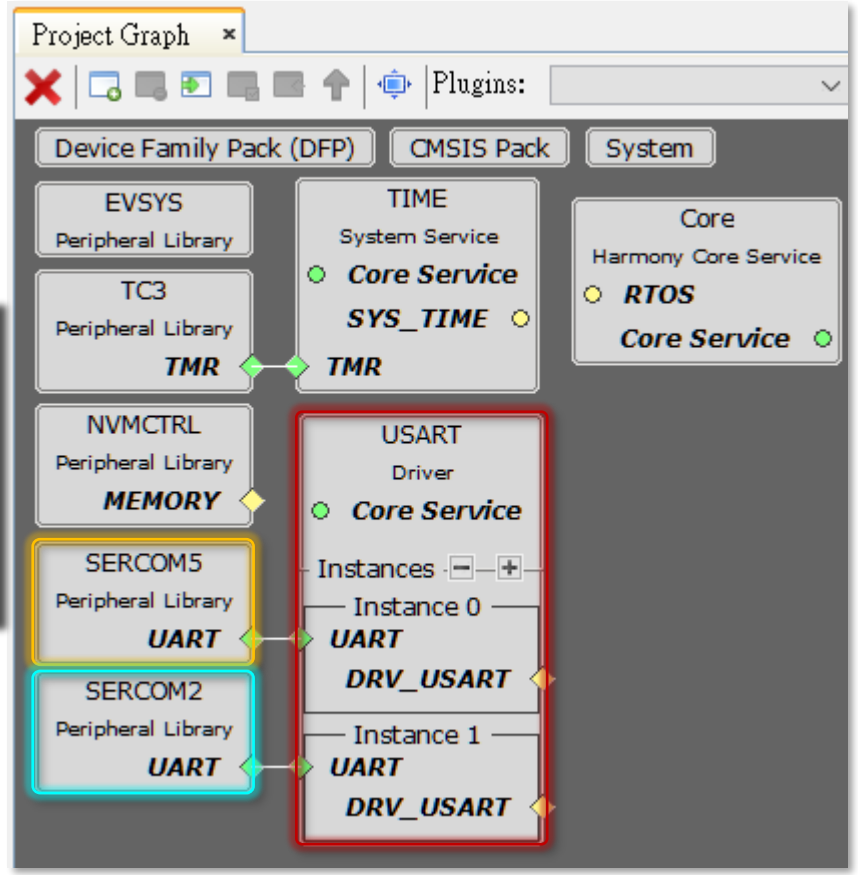
🛠 Lab12 Multi-Instance UART Communication

- 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.









MICROCHIP

🛠 Lab12 Multi-Instance UART Communication

- To implement a raw bytes write function of USART2 in app_drv_usart2.c

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>

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[10];

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, ... ) { ... }
```

This function is different to the string operation of USART1printf()

🛠 Lab12 Multi-Instance UART Communication

- **Replace "DRV_USART2: Hello World" output by use USART2write()**

app_drv_usart2.c

```
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;
        }
        ...
```

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

app_drv_usart2.h

```
void APP_DRV_USART2_Tasks( void );

void USART2write( unsigned char *pdata, int len );

extern APP_DRV_USART2_DATA app_drv_usart2Data;
...
```

MICROCHIP

I2C Driver Library

Harmony Driver

MICROCHIP

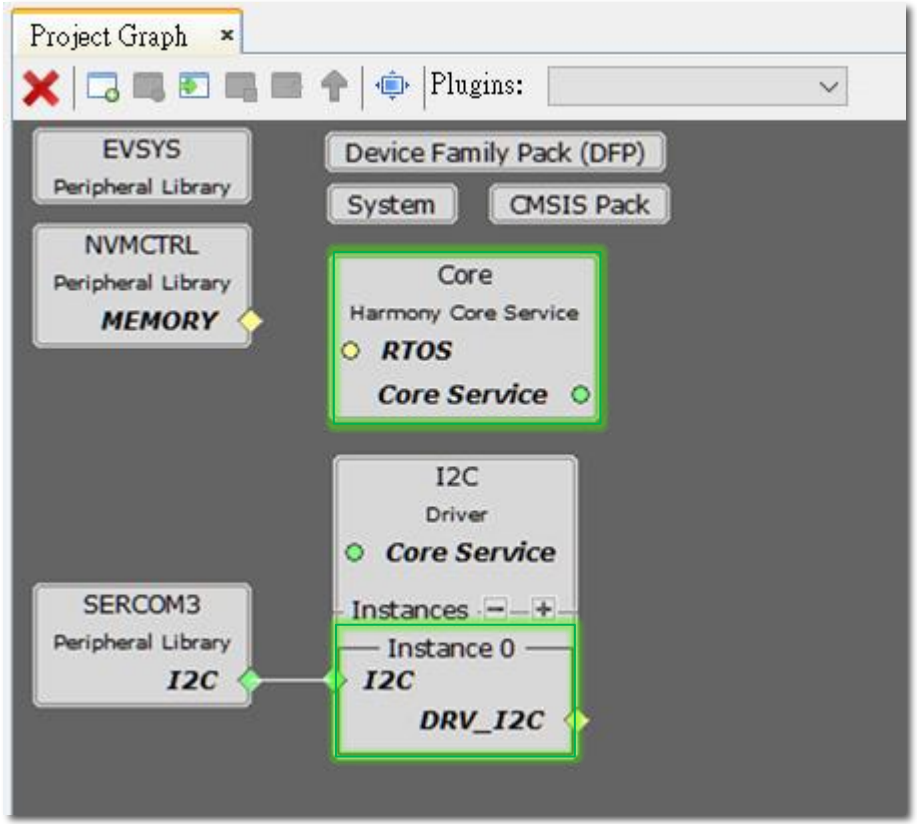
I2C Driver Library

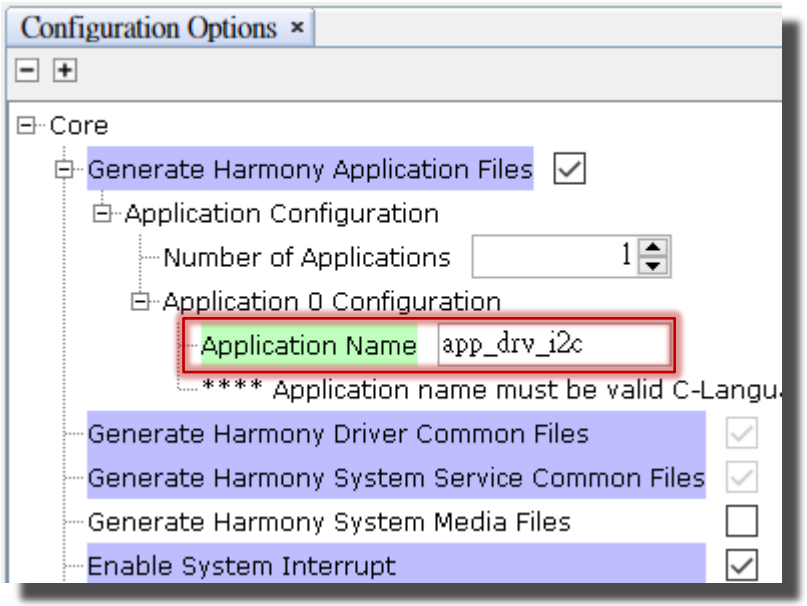
- **Introduction**

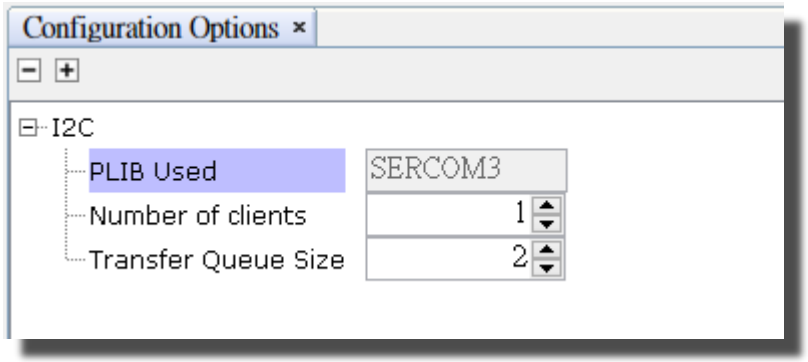
This driver library provides application ready routines to read and write data using the I^2C protocol, thus minimizing developer's awareness of the working of the I^2C protocol.

- Supports I^2C 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

MICROCHIP

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

MICROCHIP

🛠 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)

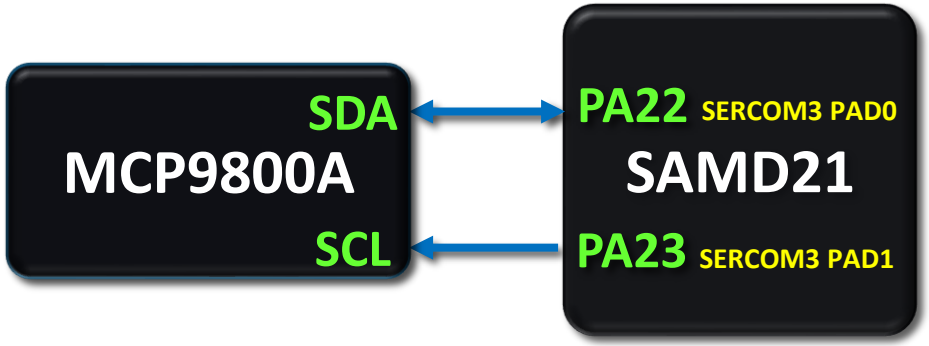
Code = MCP9800 output in decimal

```
Result = (TA[0]<<4|TA[1]>>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]

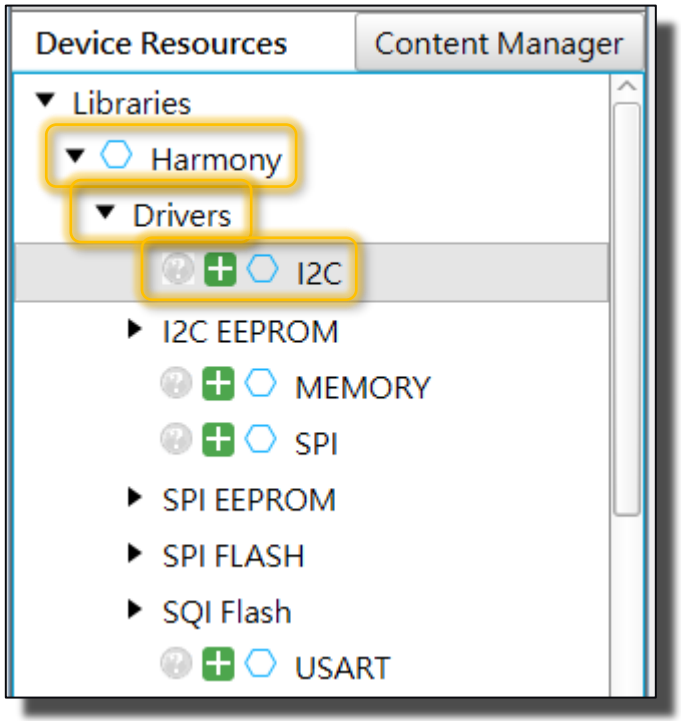
TA[1]

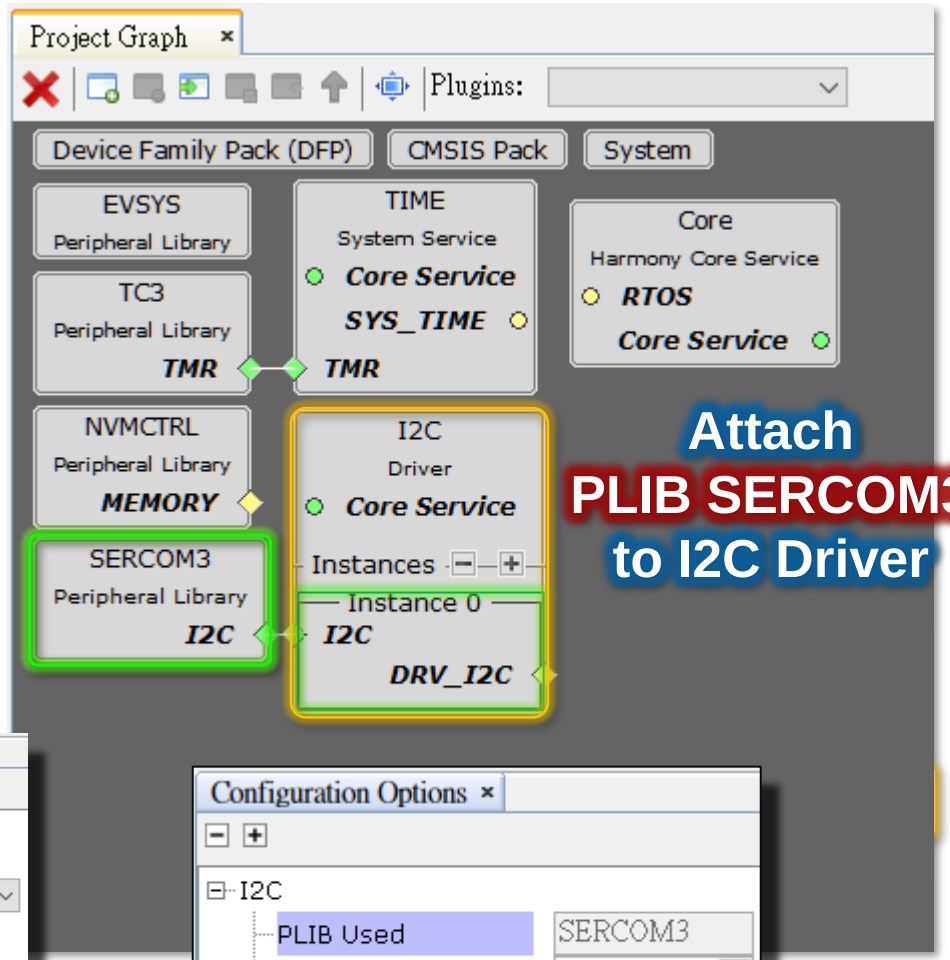


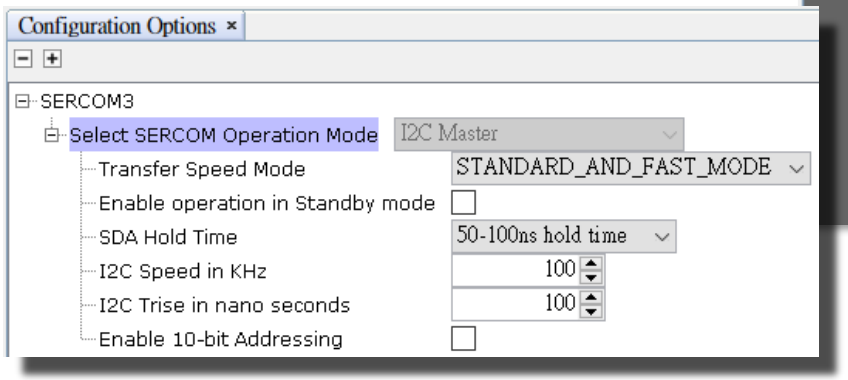
MICROCHIP

🛠 Lab13 Single-Client I2C Digital Temperature Sensor

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



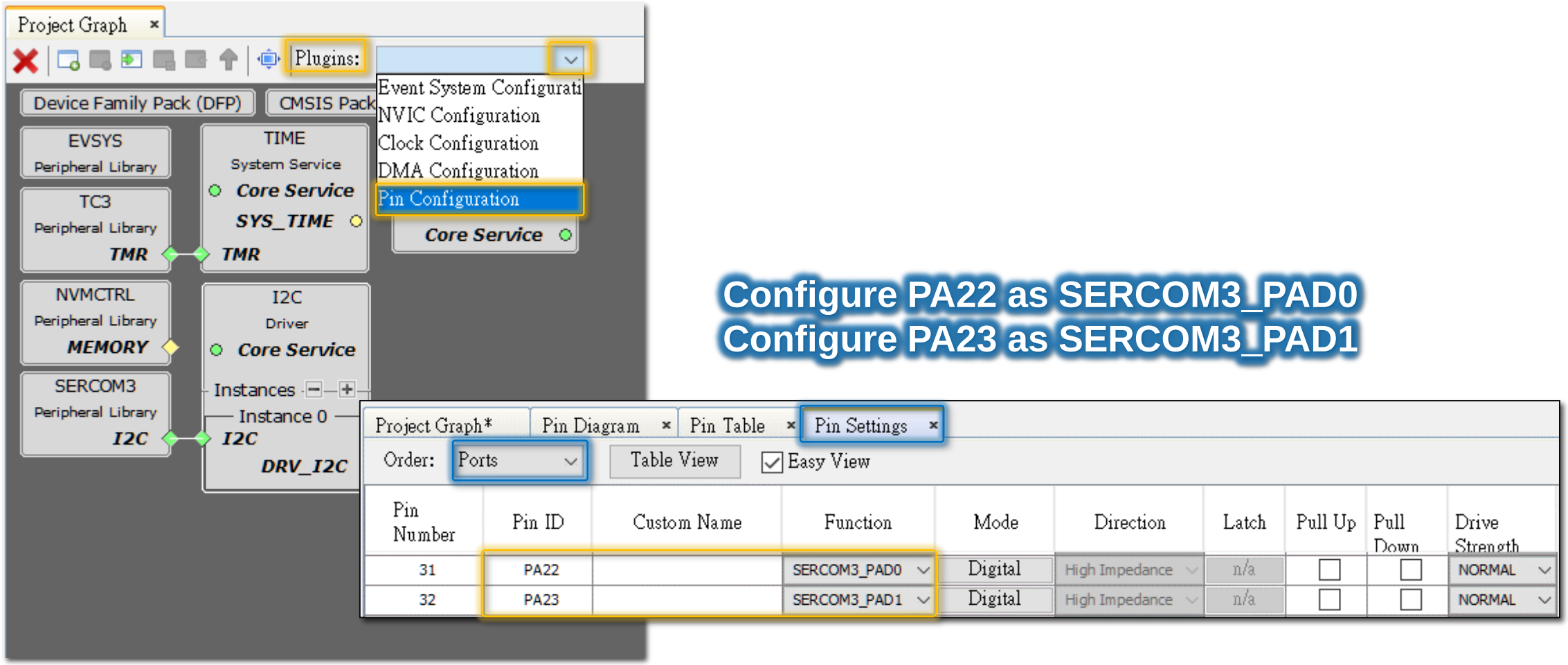




MICROCHIP

🛠 Lab13 Single-Client I2C Digital Temperature Sensor

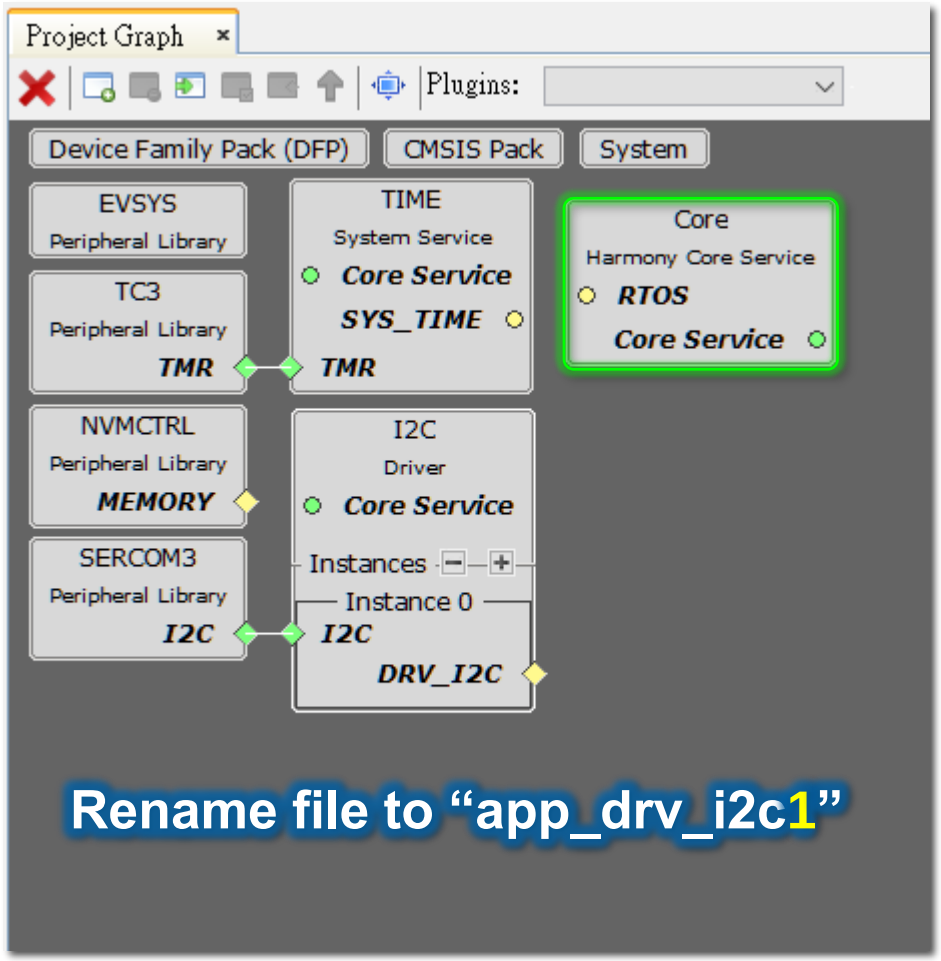
- Enable **PLIB SERCOM3** pads of I2C Driver **Instance 0**.

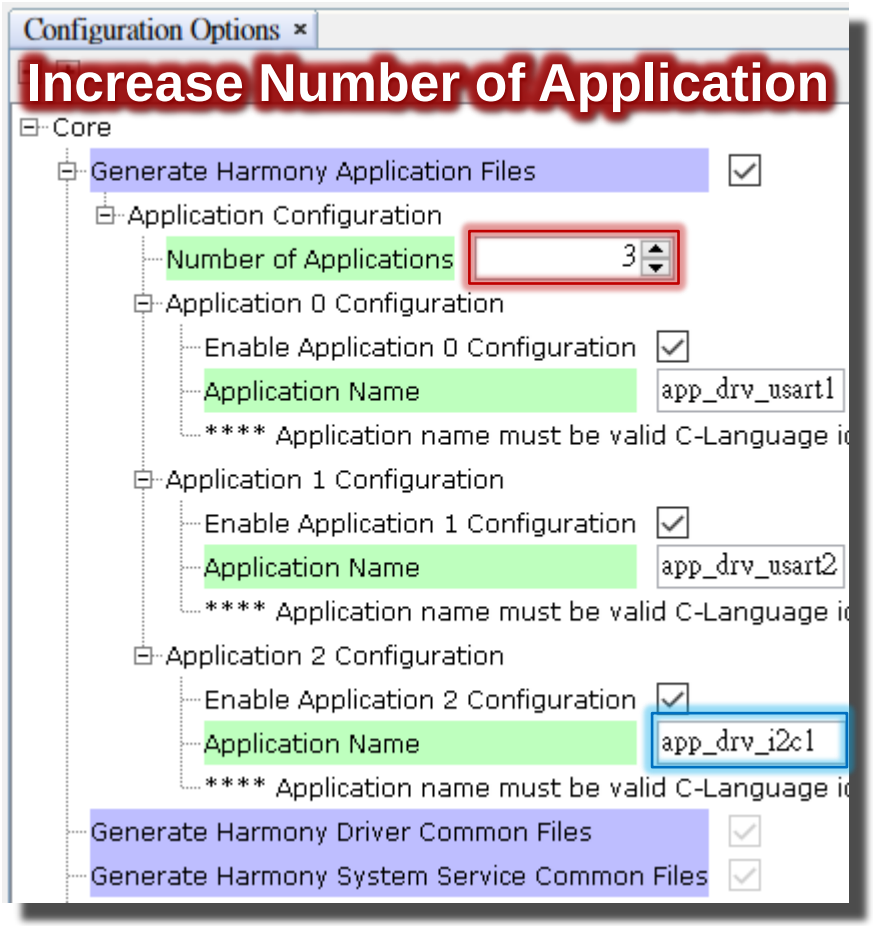


MICROCHIP

🛠 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".

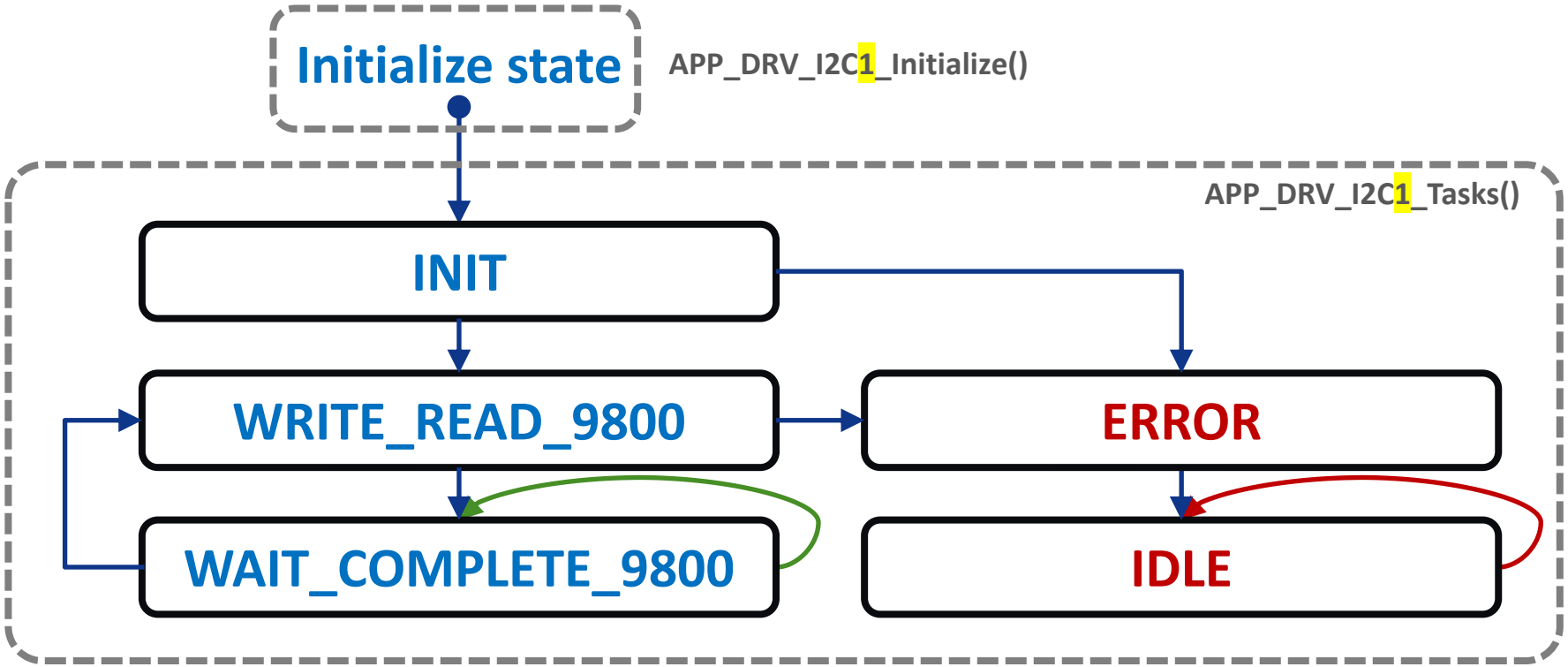




MICROCHIP

🛠 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".**

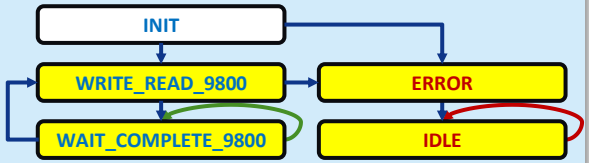
app_drv_i2c1.h

```
#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.



MICROCHIP

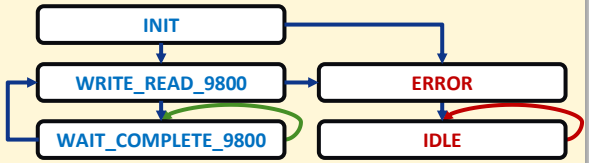
🛠 Lab13 Single-Client I2C Digital Temperature Sensor

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

app_drv_i2c1.c

Overwrite original example code

```
/*******************************************************************************
  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;
    }
}
```



MICROCHIP

🛠 Lab13 Single-Client I2C Digital Temperature Sensor

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

Keep this line

app_drv_i2c1.c

Include app_drv_usart1.h for debug message output

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

// *****************************************************************************
// *****************************************************************************
// 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_i2c1.c

```
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**

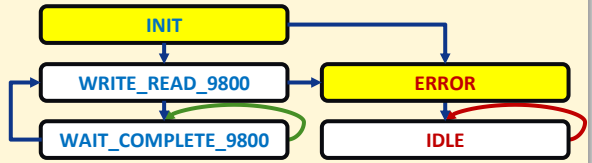
app_drv_i2c1.c

Open Driver

Register Callback

Handling error state message

```
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);

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");

Keep this line  app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_IDLE;
        break;

    case APP_DRV_I2C1_STATE_IDLE:
        break;
    }
}
```



MICROCHIP

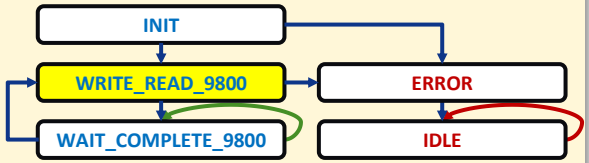
🛠 Lab13 Single-Client I2C Digital Temperature Sensor

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

app_drv_i2c1.c

```
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;
        }

        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;
    }
}
```

Keep this line



MICROCHIP

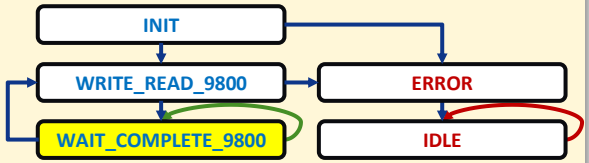
🛠 Lab13 Single-Client I2C Digital Temperature Sensor

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

app_drv_i2c1.c

```
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;
    }
}
```

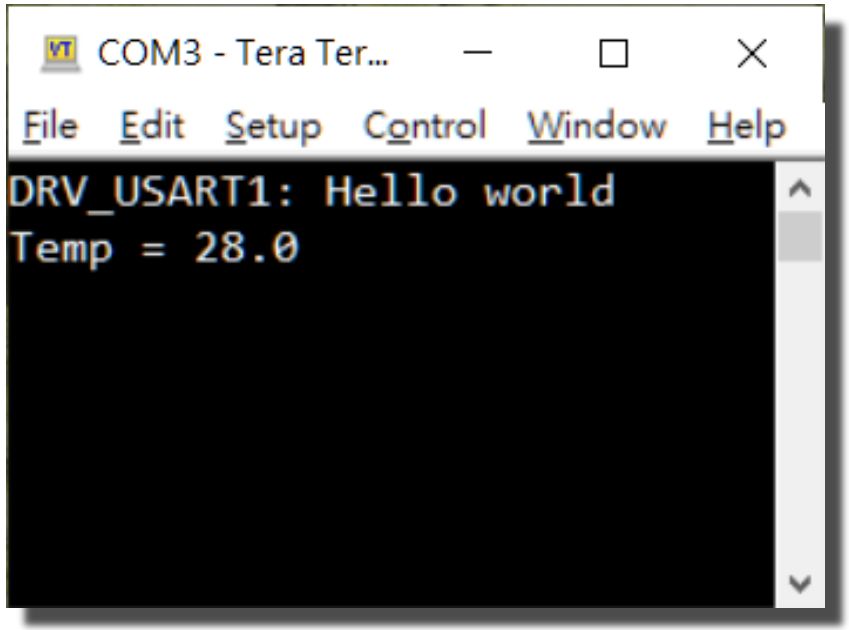
Keep this line

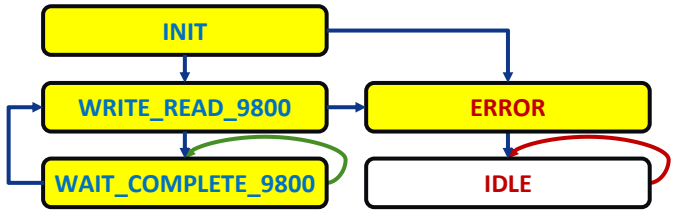


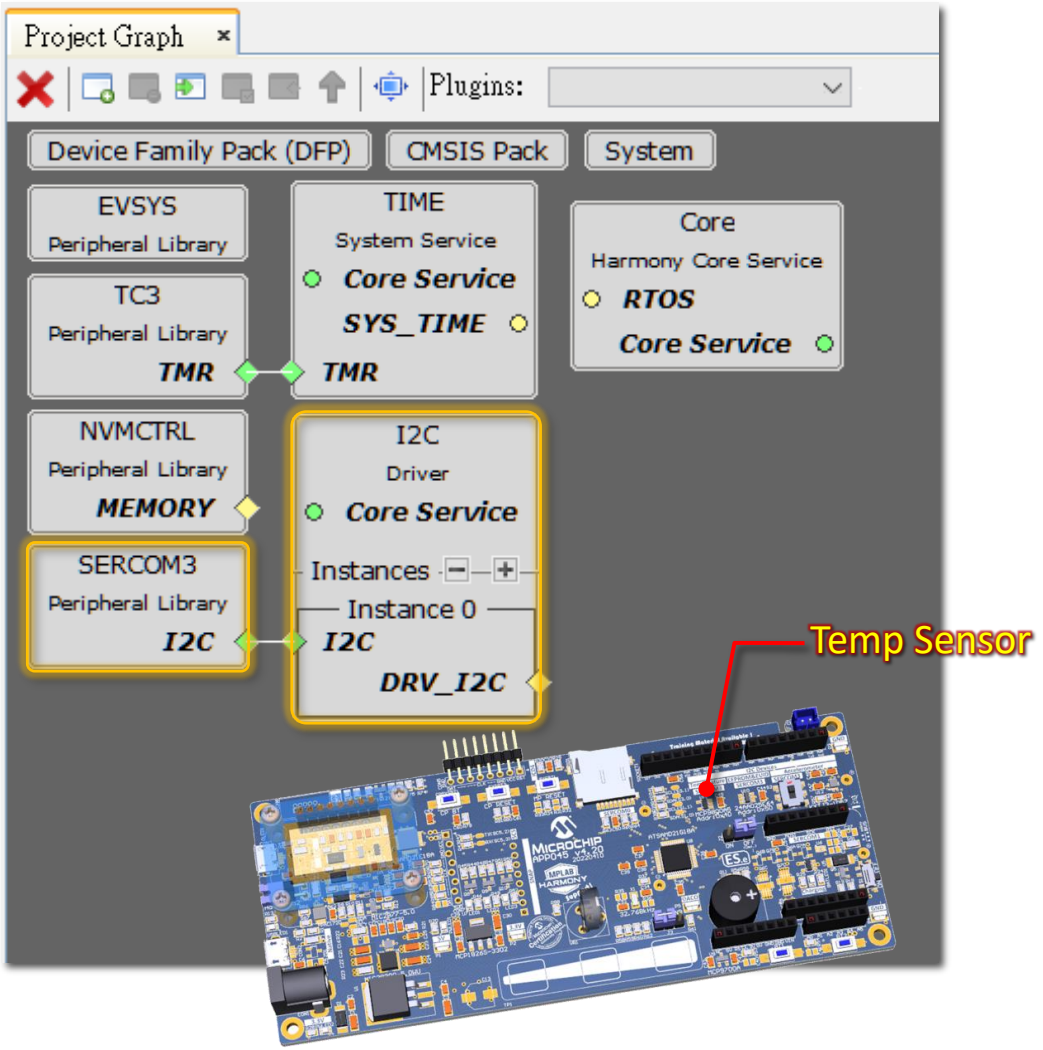
MICROCHIP

🛠 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 !)**

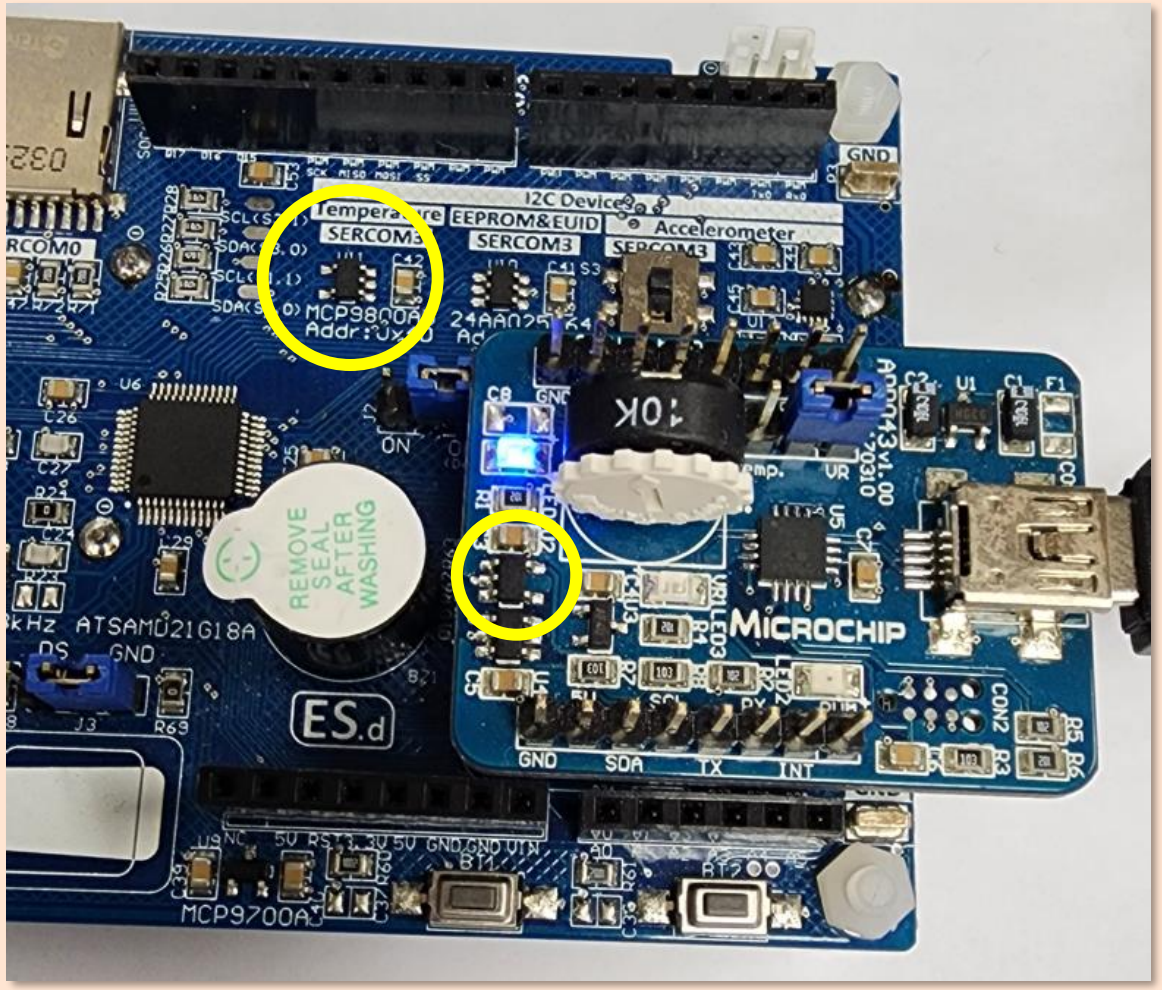






🛠 Lab13 Single-Client I2C Digital Temperature Sensor

- If you installed the **Microchip UART Click board** in the previous Lab, there is another **MCP9800** that will share the same I2C bus.



MICROCHIP

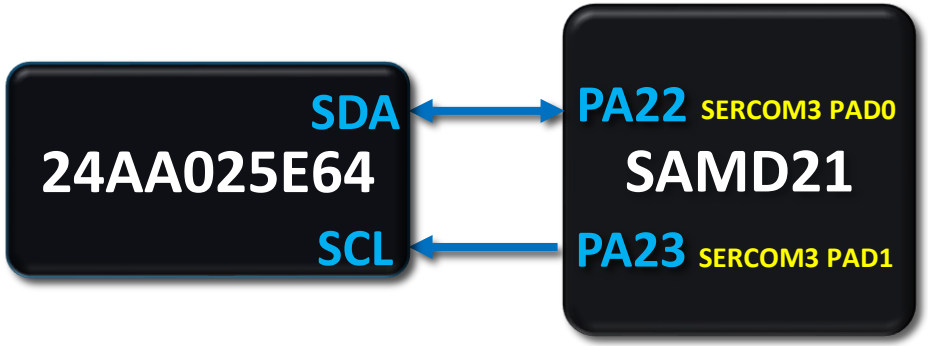
🛠 Lab 14
Multi-Client I2C Serial EEPROM

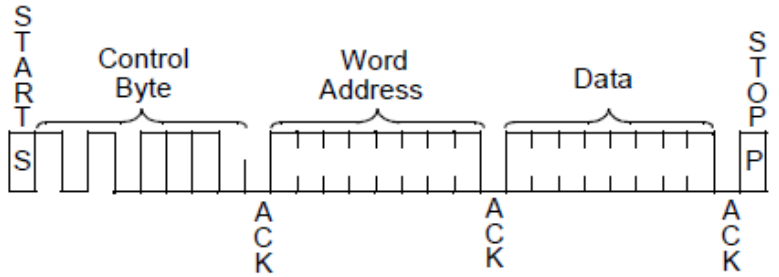
I2C Driver Library

MICROCHIP

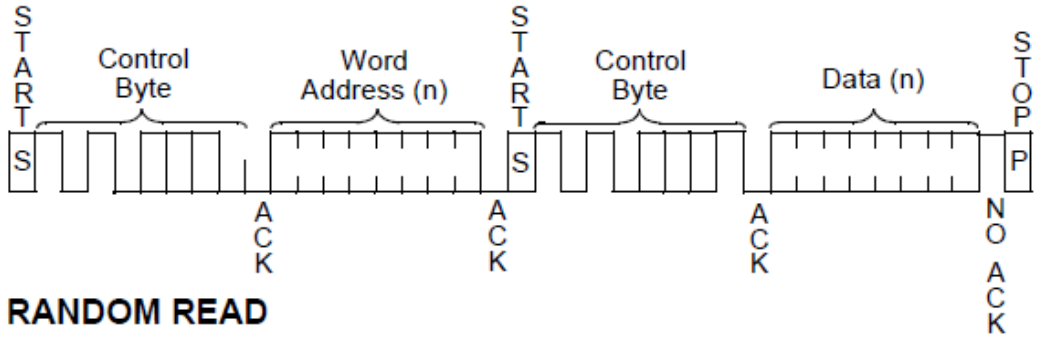
🛠 Lab14 Multi-Client I2C Serial EEPROM

- **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.





BYTE WRITE

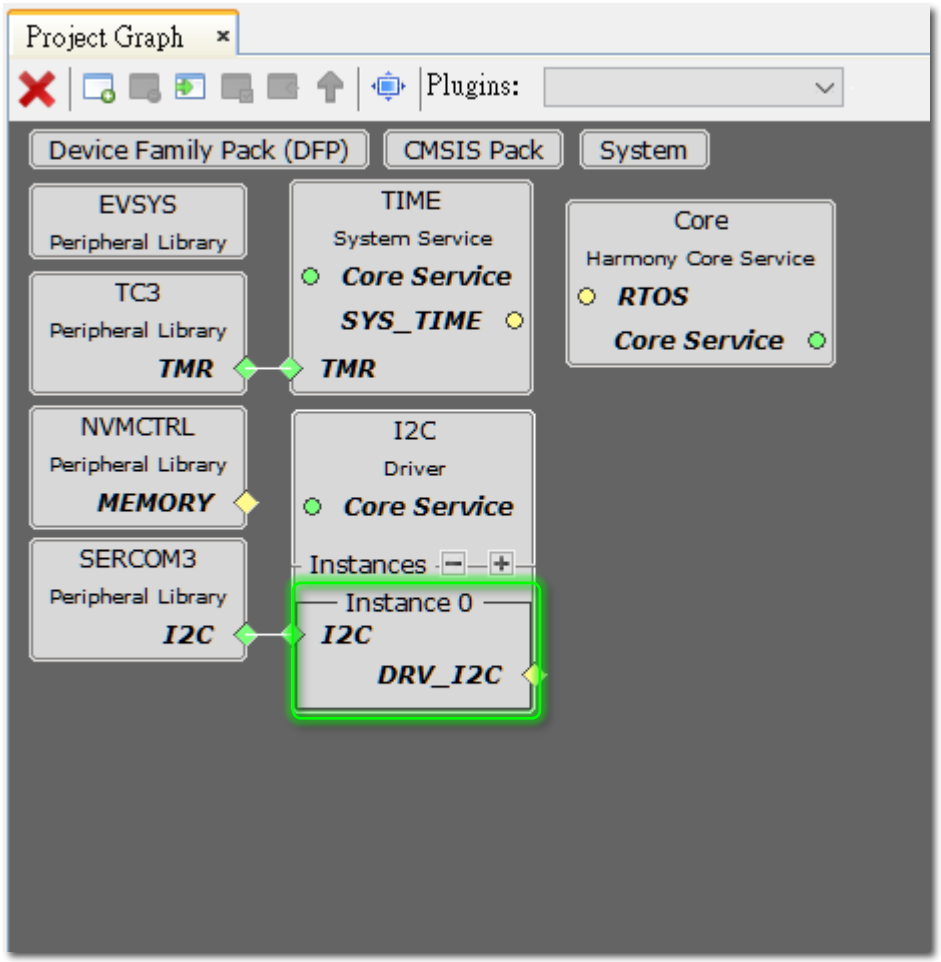


RANDOM READ

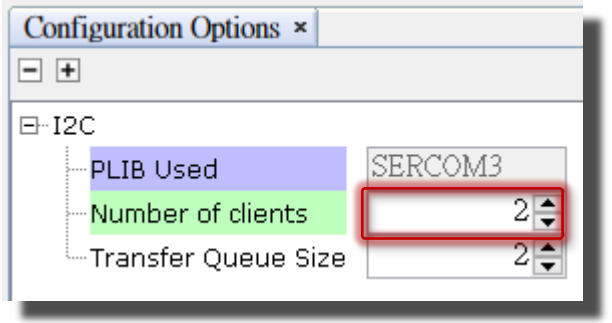
MICROCHIP

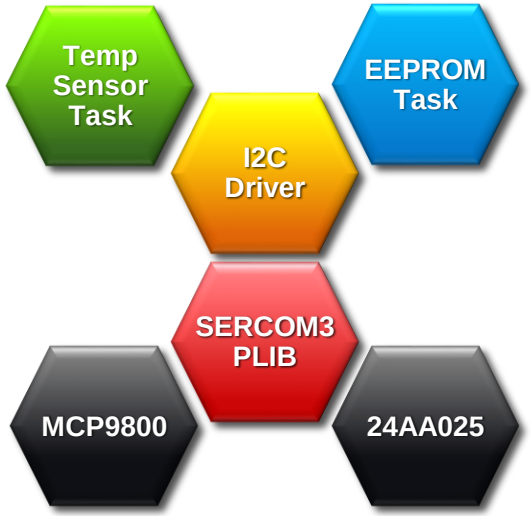
🛠 Lab14 Multi-Client I2C Serial EEPROM

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



Add Number of clients to 2

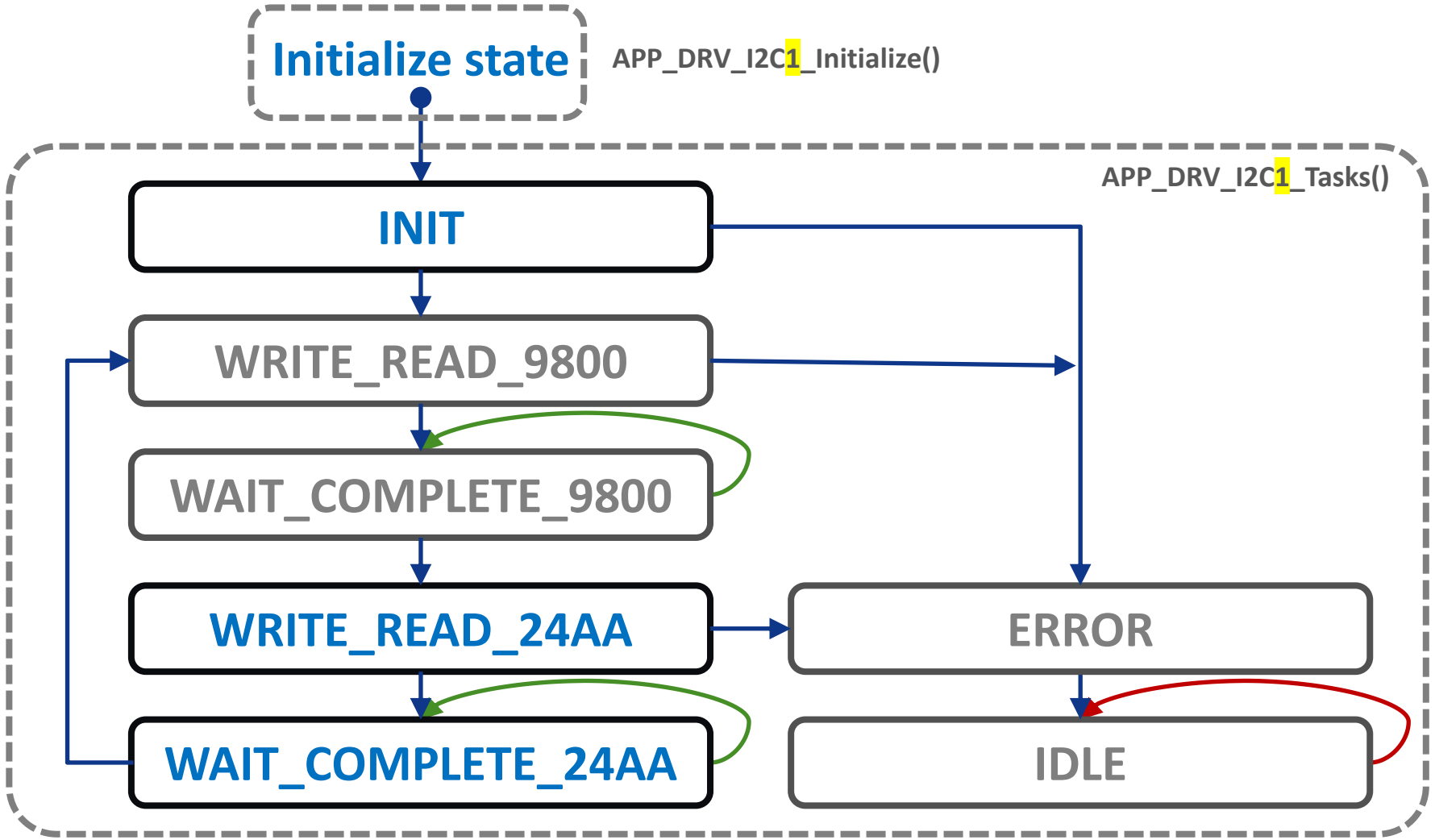




MICROCHIP

🛠 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

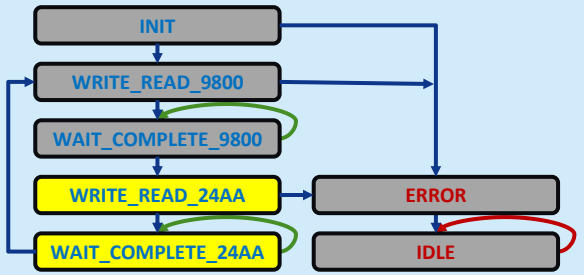
- **Insert two states for EEPROM in “app_drv_i2c1.h”.**

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 );
...
```

Insert between this

Insert between this



MICROCHIP

🛠 Lab14 Multi-Client I2C Serial EEPROM

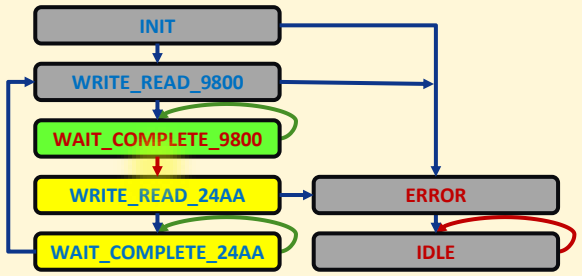
- **Construct new states flow of task in "app_drv_i2c1.c".**

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;
    }
}
```

Change next state to WRITE_READ_24AA

Modify this Line



MICROCHIP

🛠 Lab14 Multi-Client I2C Serial EEPROM

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

app_drv_i2c1.c

Include System Time service for the Delay function

```
#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>

// *****************************************************************************
// *****************************************************************************
// 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.c

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

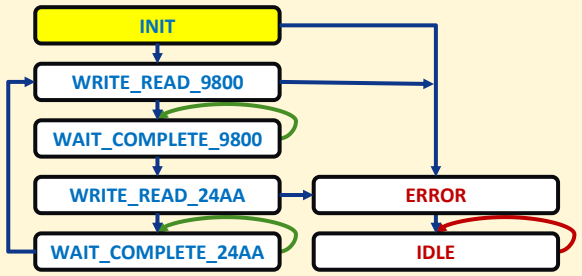
🛠 Lab14 Multi-Client I2C Serial EEPROM

- **Add Driver Open for EEPROM client in state INIT**

app_drv_i2c1.c

```
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;
        }

        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:
        ...
```

Keep this line



MICROCHIP

🛠 Lab14 Multi-Client I2C Serial EEPROM

- Driver Write byte 0x24 to EEPROM address 0x0 in state INIT

app_drv_i2c1.c

```
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);

        app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
        break;

    case APP_DRV_I2C1_STATE_WRITE_READ_9800:
        ...
```

Keep this line

Polling method of Write

EEPROM Write delay

Register Transfer Callback

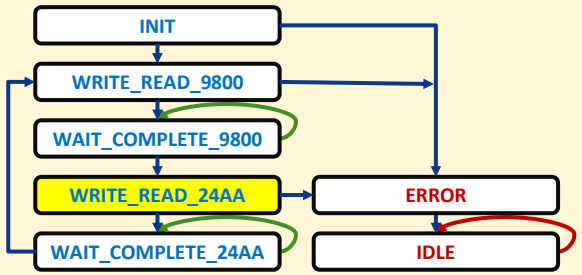
MICROCHIP

🛠 Lab14 Multi-Client I2C Serial EEPROM

- **Driver Write followed by Read in state WRITE_READ_24AA**

app_drv_i2c1.c

```
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:
        ...
    }
```



MICROCHIP

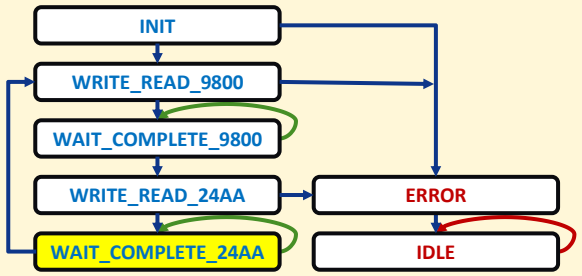
🛠 Lab14 Multi-Client I2C Serial EEPROM

- Read EEPROM complete check in state **WAIT_COMPLETE_24AA**

app_drv_i2c1.c

```
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_I2C_STATE_ERROR:
        ...
    case APP_DRV_I2C_STATE_IDLE:
        ...
    }
}
```

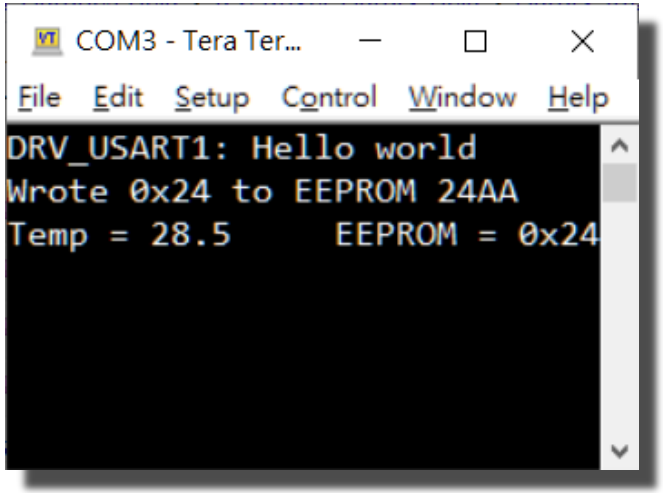
Keep this line

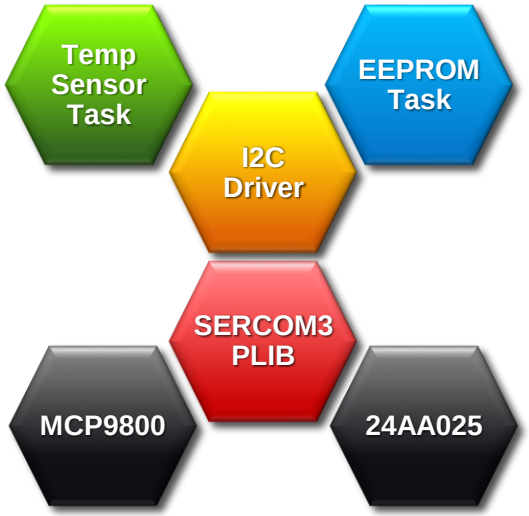


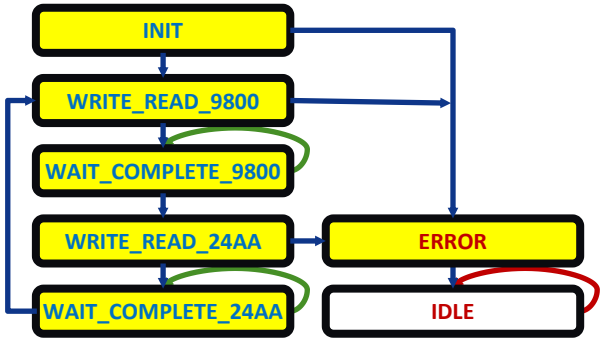
MICROCHIP

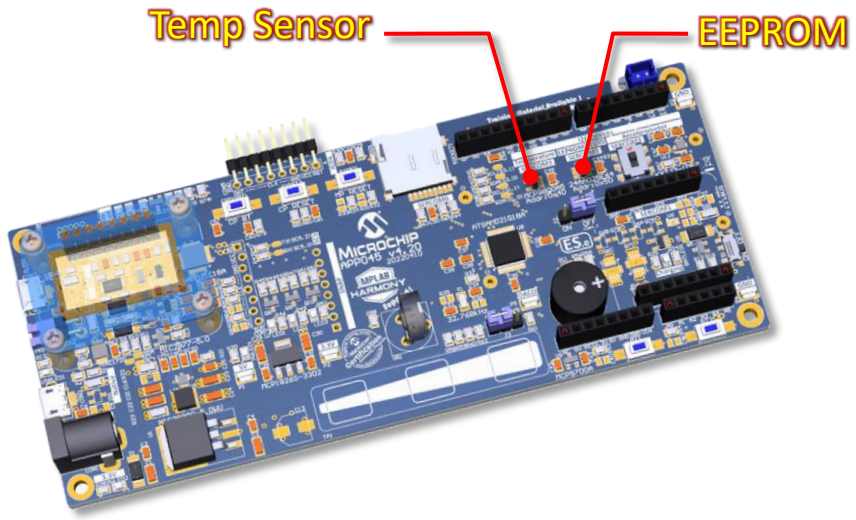
🛠 Lab14 Multi-Client I2C Serial EEPROM

- **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 !)**









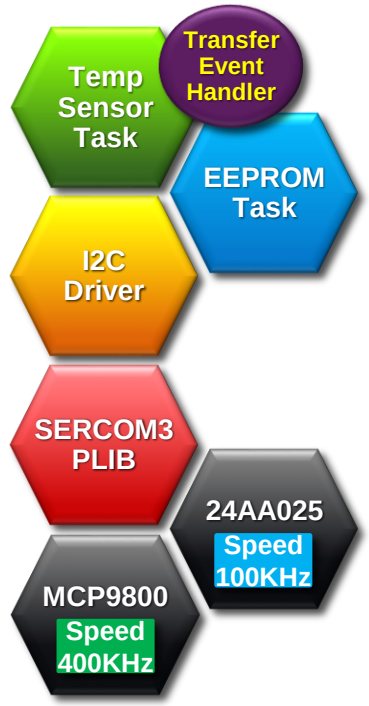
MICROCHIP

🛠 Lab14 Multi-Client I2C Serial EEPROM

Discuss 1

- **How to implement the I2C Driver to given Two different I2C speed properties of two I2C devices that to shared of the same PLIB SERCOM ?**

```
DRV_I2C_TRANSFER_SETUP TransferSetup;

void I2C1TransferHandler( ... )
{
    switch( event )
    {
    case DRV_I2C_TRANSFER_EVENT_PENDING:        break;
    case DRV_I2C_TRANSFER_EVENT_COMPLETE:
        if( transferHandle == I2C1WriteReadHandle_9800 ) I2C1Complete_9800 = true;
        if( transferHandle == I2C1WriteReadHandle_24AA ) I2C1Complete_24AA = true;
        break;
    ...
}

I2C1Handle_9800 = DRV_I2C_Open(DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE);
I2C1Handle_24AA = DRV_I2C_Open(DRV_I2C_INDEX_0, DRV_IO_INTENT_READWRITE);

DRV_I2C_TransferEventHandlerSet(I2C1Handle_9800 , I2C1TransferHandler, 0);
DRV_I2C_TransferEventHandlerSet(I2C1Handle_24AA , I2C1TransferHandler, 0);

TransferSetup.clockSpeed = 400000;
DRV_I2C_TransferSetup(I2C1Handle_9800, &TransferSetup);
DRV_I2C_WriteReadTransferAdd( I2C1Handle_9800, MCP9800_ADDRESS,
                              I2C1WriteBuf_9800, 1, I2C1ReadBuf_9800,  2, &I2C1WriteReadHandle_9800 );
TransferSetup.clockSpeed = 100000;
DRV_I2C_TransferSetup(I2C1Handle_24AA, &TransferSetup);
DRV_I2C_WriteReadTransferAdd( I2C1Handle_24AA, MCP24AA_ADDRESS,
                              I2C1WriteBuf_24AA, 1, I2C1ReadBuf_24AA,  1, &I2C1WriteReadHandle_24AA );
```



MICROCHIP

Microchip Trademarks

https://www.microchip.com/en-us/about/legal-information/microchip-trademarks

Trademarks	Description/Appropriate Nouns	Notes
Adaptec "a" logo		Registered in other countries (not US)
Adaptec®	products, solutions, cables, software, drivers, adapters, controllers, series, family	
Adjacent Key Suppression™	technology	
AgileSwitch®	digital programmable gate driver, gate driver, drivers, family, device	US Only
AKS™	technology	
Analog-for-the-Digital Age™	AIPD family tagline	
Any Capacitor™	stable	
AnyIn™	feature, structure, capability	
AnyOut™	feature, capability	
Augmented Switching™	technology	
AVR®	devices, microcontrollers, MCUs, CPU, architecture, family	
AVR Logo		
AVR Freaks®	forum	
BesTime®	technology	
BitCloud®	SDK	
BlueSky™	firewall, subscription service, technology	
BodyCom™	technology, system, development kit	
Clockstudio™	application, software, program, utility	
ClockWorks®	Configurator, products, series, family, devices	US Only
CodeGuard™	security	
CryptoAuthentication™	development library, library, devices, ICs, family	
CryptoAutomotive™	technology, security ICs, development kits	
CryptoCompanion™	chip	
CryptoController™	solution, Trusted Platform Module (TPM)	
CryptoMemory®	cryptographic security ICs, ICs	

NOTE: *Always check with the Legal Department for the most current trademarks.

Trademarks	Description/Appropriate Nouns	Notes
CryptoRF®	transponder, devices	
dsPIC®	Digital Signal Controllers (DSCs)	
dsPIC Logo		
dsPICDEM™	demonstration board, development board	
dsPICDEM.net™	board	
Dynamic Averaged Matching™ (DAM™)	architecture	
ECAN™	technology, solution	
Espresso T1S™	device	
EtherGREEN™	platform, technology, family	
EtherSynch®	platform, technology, family	US Only
EyeOpen™	cable compensation	
Flashtec®	controller, products, family	US Only
flexPWR®	technology	
Frequency on Demand®		Registered in other countries (not US)
GestIC®	technology, sensing, controllers	
GridTime™	device, time server, GNSS time server	
HELDO®	family, regulators	
Hyper Speed Control®	family, architecture	US Only
HyperLight Load®	mode	US Only
ICSP™ or In-Circuit Serial Programming™	programming capability	
IdealBridge™	rectifier, dual MOSFET-bridge rectifier	
IGaT™	board, demonstration board	
IGLOO®	FPGAs, family, devices, series	
INICnet™	technology, sniffer	
Intelligent Paralleling™	technology	
IntelliMOS™	family, power stage family	US Only
Inter-Chip Connectivity™	technology	
JitterBlocker™	family, attenuators	
JukeBlox®	technology, platform	

NOTE: *Always check with the Legal Department for the most current trademarks.

Trademarks	Description/Appropriate Nouns	Notes
JukeBlox with design	Registered word mark JukeBlox with a design 	Image is not registered; this is a graphic treatment with the word mark
KEELOQ®	security ICs, protocol, technology	
Kleer®	technology	
Kleer with design	Registered word mark Kleer with a design 	Image is not registered; this is a graphic treatment with the word mark
Knob-on-Display™/KoD™	technology, solution, HMI, user interface, touch controller, family, devices, Widget Configurator, Knob Designer, Position Wizard	
LANCheck®	online design review, online review, review	
Libero®	SoC Design Suite, tools, software, design files, IDE, designer	US registration
LinkMD®	cable diagnostics, technology, family, diagnostics engine	
MarginLink™	signal integrity testing	
maxCrypto™	controller-based encryption, encryption	
maXStylus®	solution	
maXTouch®	touchscreen controllers, technology, studio, family	
maxView™	storage manager, tools, drivers	
MediaLB®	bus, controller, device	
megaAVR®	devices, MCUs	
memBrain™	neuromorphic memory product, product, solution, technology	
Microchip Logo		
Microchip Name and Logo		
Microsemi Logo		
Microsemi®		

NOTE: *Always check with the Legal Department for the most current trademarks.

Trademarks	Description/Appropriate Nouns	Notes
Mindi™	analog simulator, analog simulator tool	
MiWi™	wireless networking protocol, protocol, network, stack, software, applications, mesh, coordinator	
MOST®	technology, NetServices, evaluation kit, board, system, device, product, network, specification, platform, cooperation, design	
MOST Logo		
motorBench®	Development Suite	US Only
MPASM™	assembler	
MPF™	products, devices	
MPLAB®	Integrated Development Environment, In-circuit Debugger, In-Circuit Emulator, IDE, ICD, REAL ICE™ in-circuit emulator, starter kit, compiler, XC compiler, C compiler, Harmony, Integrated Programming Environment, IPE, Xpress, Code Configurator, development ecosystem, PICkit™, server, library, Snap, Connect Configurator, Network Creator, Analog Designer, Discover, Mindi™, Analysis Tool Suite, Discover, Universal Device Programmer, Machine Learning Development Suite, PTG, Programmer-to-Go, Programmer-to-Go Mobil App, SiC Power Simulator	
MPLAB Certified Logo		
MPLIB™	object librarian or librarian	
MPLINK™	object linker or linker	
mSiC™	MOSFETs, diodes, products, gate drivers, modules, bare die, technology, solutions	
mTouch®	solution, sensing solution, evaluation kit, technology, development kit, library	US Only
MultiTRAK™	technology	
NetDetach™	technology	
Omniscient Code Generation™ (OCG)	compilation technology	
OptoLyzer®	tool, network analysis tool, Studio	
PIC®	microcontroller, MCU, device(s), assembler	

NOTE: *Always check with the Legal Department for the most current trademarks.

Trademarks	Description/Appropriate Nouns	Notes
PIC32 Logo		
PICDEM™	demonstration board	
PICDEM.net™	Internet/Ethernet demonstration board	
PICkit™	starter kit, programmer, debugger, in-circuit debugger, development tool, tool, on-board	
picoPower®	technology	
PICSTART®	development programmer	
PICtail™	daughter board, board	
PolarFire®	FPGA, family, kits, devices, SoC, System-on-Chip	
Power MOS IV™		
Power MOS 7™		
Powermite 3®	package	US Only
PowerSmart™	digital control library designer, development suite	
Precision Edge®	family, product family	US Only
ProASIC®	FPGA, architecture, board, kit, family	US Only
ProASIC Plus®	FPGA, devices, family, architecture, switch	US Only
ProASIC Plus Logo		US Only
Prochip Designer®	software suite	
PureSilicon™	oscillator, technology	
QMatrix™	sensor, devices	
QTouch®	sensor, devices, library, tools, development platform, project builder, Composer, Project Builder, board, solution, technology, Configurator	
Quiet-Wire®	enhanced EMI technology, family, technology	
Ripple Blocker™	technology, attenuators, product line, family of linear regulators	
REAL ICE™	In-circuit emulator	
RTAX™	FPGAs	
RTG4™	FPGAs	
SAM-BA®	software, application, monitor	

NOTE: *Always check with the Legal Department for the most current trademarks.

Trademarks	Description/Appropriate Nouns	Notes
SAM-ICE™	emulator	
SenGenuity®	sensors, products	
Serial Quad I/O™ or SQI™	family, flash device, product	
Silicon Storage Technology®		Japan only
simpleMAP™	protocol	
SimpliPHY™	family, products, devices	
SmartBuffer™	devices, ICs	
SmartFusion®	FPGA, evaluation kit, SoC, System-on-Chip,	US Only
SmartHLS™	IDE, compiler software, compiler, software	
SMART-I.S.™	technology	
SpyNIC®	device	
SST®		
SST Logo		
storClad™	encryption technology, technology	
SuperFlash®	technology, kit, macro, device, memory	
SuperSwitcher™	family, regulator, buck regulator	
SuperSwitcher II™	family	
Switchtec™	technology, family, switch(es), product line	
Symmcom®		China only
Symmetricom®		
SynchroPHY™	family, products, devices	
SyncServer®	server, instrument, clock, oscillator, device	
SyncWorld®	timing, Ecosystem Program	US only
Tachyon®	products, family, controllers, platform, technology	
The Embedded Control Solutions Company®	**MarCom Use Only**	
TimeCesium®	products, primary frequency reference	US only
TimeHub®	platform, system	US only
TimePictra®	software suite, platform, Synchronization Management System	US only

NOTE: *Always check with the Legal Department for the most current trademarks.

Trademarks	Description/Appropriate Nouns	Notes
TimeProvider®	series, server, clock, products, grandmaster, ePRTC, enhanced PRTC	US only
TimeSource®	Enhanced PRTC	
tinyAVR®	microcontroller, MCU	
Total Endurance™	software model	
Trusted Time™	technology	
TSHARC™	touch screen controller, controller	
Turing™	hardware security module, programmer	
UNI/O®	memory chips, bus, hardware port, communication protocol, firmware	
USBCheck™	design review	
VariSense™	technology	
VectorBlox™	accelerator, software development kit, SDK, intellectual property, IP, solution, platform, tools, tool chain, tool flow	
Vectron®		
VeriPHY™	cable diagnostics algorithm, cable diagnostics software suite	
ViewSpan™	technology	
WiperLock™	technology	
XMEGA®	microcontroller, MCU, devices, Custom Logic (XCL), family, series	
XpressConnect™	retimer	
ZENA™	software, analyzer or wireless adapter	
ZL®		

NOTE: *Always check with the Legal Department for the most current trademarks.

Microchip Technology Inc. Servicemarks*

Trademarks	Description/Appropriate Nouns	Notes
SQTP℠	Serial Quick Turn Programming capability	

NOTE: *Always check with the Legal Department for the most current servicemarks.

MICROCHIP RESERVES THE RIGHT TO CHANGE THESE GUIDELINES SOLELY AT ITS DISCRETION.

MICROCHIP