

CAE專家教室 SAM2002系列-03
SAMD21 ARM Cortex-M0+
Microcontroller & MCC Harmony
SAM2002 v1.1



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



SMART | CONNECTED | SECURE

CAE Taiwan Team
Microchip Technology Inc.

October 30, 2024

SAM2002-03 Index

- I2C Driver Library

-  Lab13 – Single-Client I2C Digital Temperature Sensor

-  Lab14 – Multi-Client I2C Serial EEPROM

-  Lab15 – Multi-Instance I2C G-Sensor

-  Advanced Exercise : MPLAB Data Visualizer

-  Advanced Exercise : I2C Device Auto Probe

- SPI Driver Library

-  Lab16 – Single-Client SPI OLED Display

- SPI Driver - Wi-Fi WINC Driver Library

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

- SPI Driver - SD Card Driver Library

-  Lab18 – Manage SPI SD card without File System

- MEMORY Driver Library

-  Lab19 – Manage NVM Flash without File System

Lab 15

Multi-Instance I2C G-Sensor

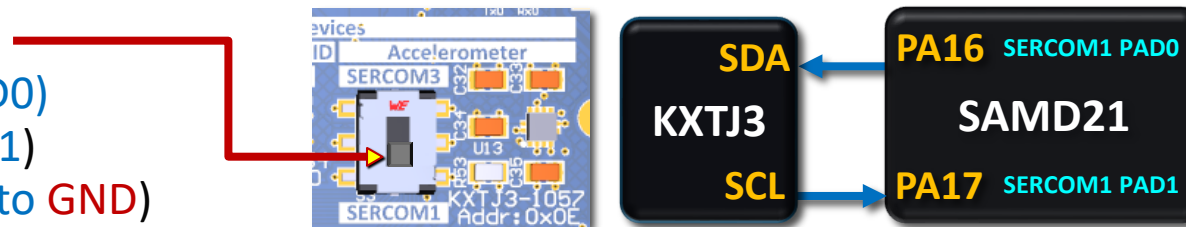
I2C Driver Library

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Add another Driver **Instance 1** of **I2C Driver** and to generate new Application file **app_drv_i2c2** to communication the on-board **Accelerometer KXTJ3**.
- **KXTJ3 initialize in state INIT:**
 - Write Software Reset “0x80” to Control Register 2 “0x1D” and wait 10ms for stable.
 - Write follow Read the **WHO AM I Register** “0x0F” to check the ID is equal to “0x35” or not.
 - Write Initial Command “0xF0” to Control Register 1 “0x1B” for op-mode and resolution.
- **KXTJ3 Interrupt Source Register read in state WRITE_READ_DRDY_TJ3:**
 - Write follow Read the **Interrupt Source Register** “0x16”.
- **KXTJ3 Data Ready Interrupt check in state WAIT_COMPLETE_DRDY_TJ3:**
 - Check “**Bit 4 DRDY INT**” of **Interrupt Source Register** to make sure data prepared for read.
- **KXTJ3 Axis Data Read in state WRITE_READ_AXIS_TJ3:**
 - Write follow Read the **Axis Output Register** in below sequence to get X, Y, Z data.

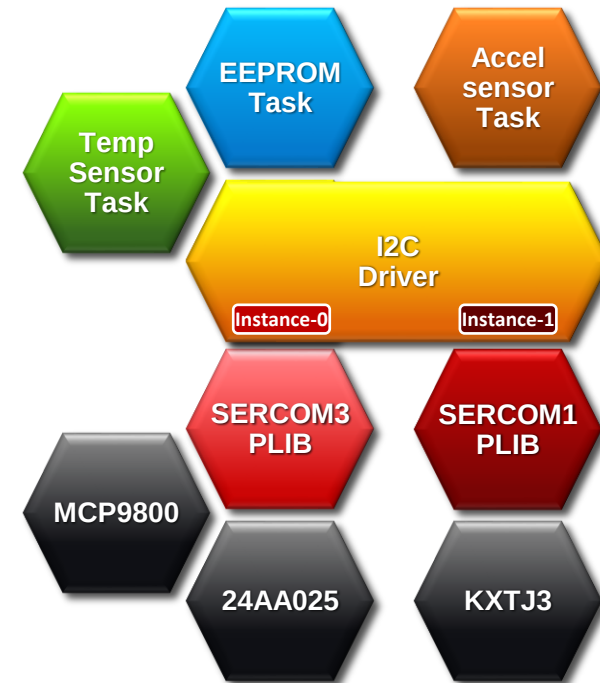
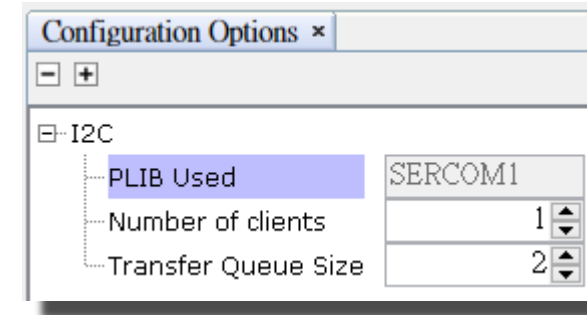
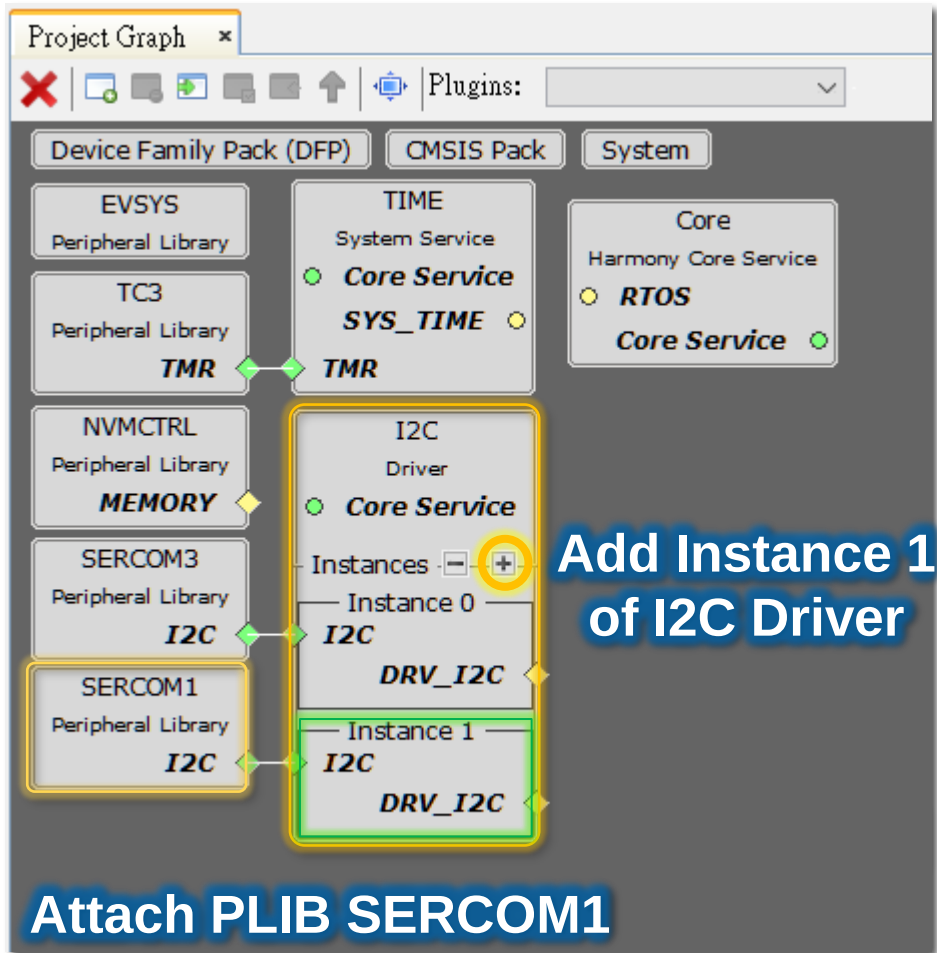
XOUT_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

- **KXTJ3 Axis Data Output in state WAIT_COMPLETE_AXIS_TJ3:**
 - Output X, Y, Z Axis data as format “**G-Sensor** = (+XXXX, +YYYY, +ZZZZ)”
- **KXTJ3 configuration:**
 - **Switch the S3 to SERCOM1 on EVB**
 - **SDA** connect to **PA16** (SERCOM1 PAD0)
 - **SCL** connect to **PA17** (SERCOM1 PAD1)
 - Slave address is **0x0E**. (ADDR pin tie to GND)



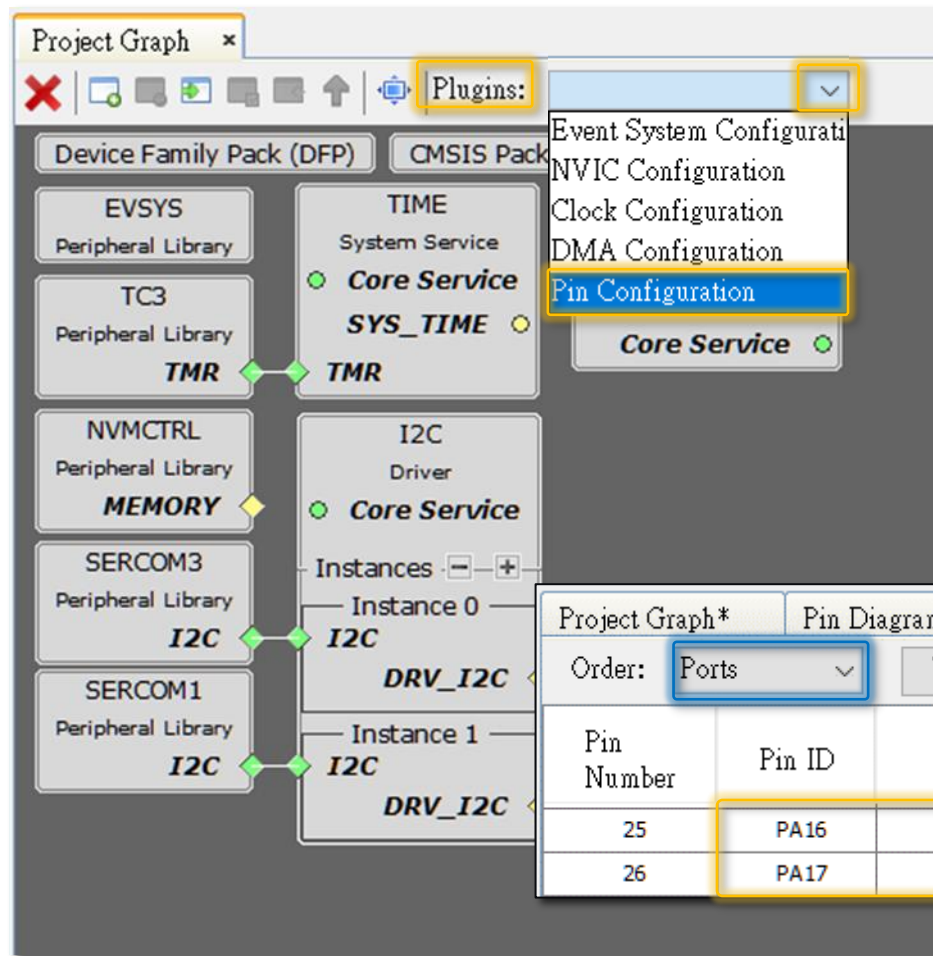
⚙️ Lab15 Multi-Instance I2C G-Sensor

- Harmony Configuration (Add **Instance 1** of I2C driver)



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Enable **PLIB SERCOM1** pads of I2C Driver **Instance 1**.

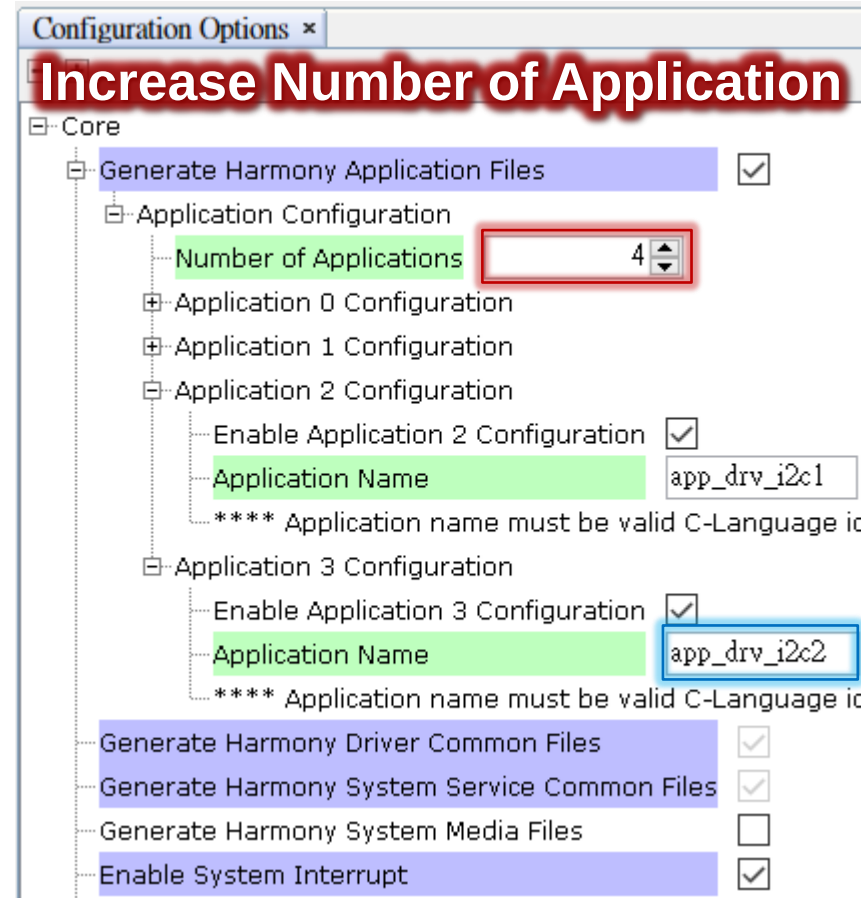
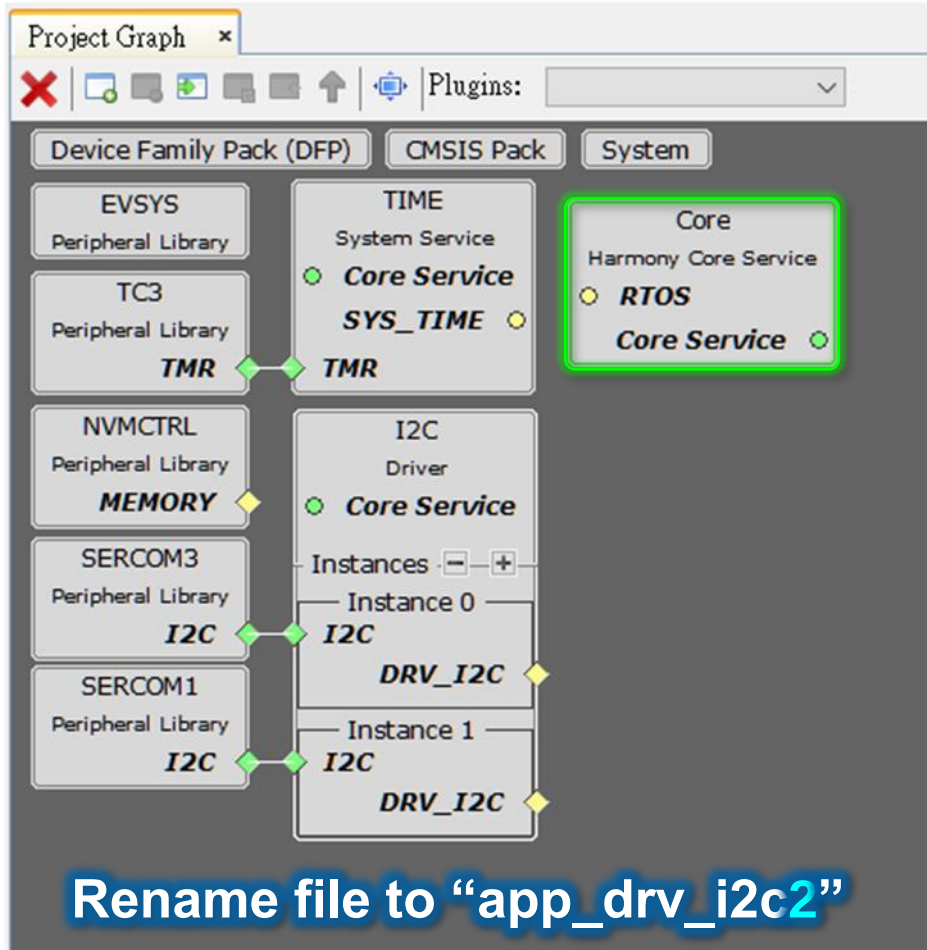


Configure PA16 as SERCOM1_PAD0
Configure PA17 as SERCOM1_PAD1

Project Graph* Pin Diagram x Pin Table x Pin Settings x									
Order: Ports v		Table View		☒ Easy View					
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
25	PA16		SERCOM1_PAD0 v	Digital	High Impedance v	n/a	☐	☐	NORMAL v
26	PA17		SERCOM1_PAD1 v	Digital	High Impedance v	n/a	☐	☐	NORMAL v

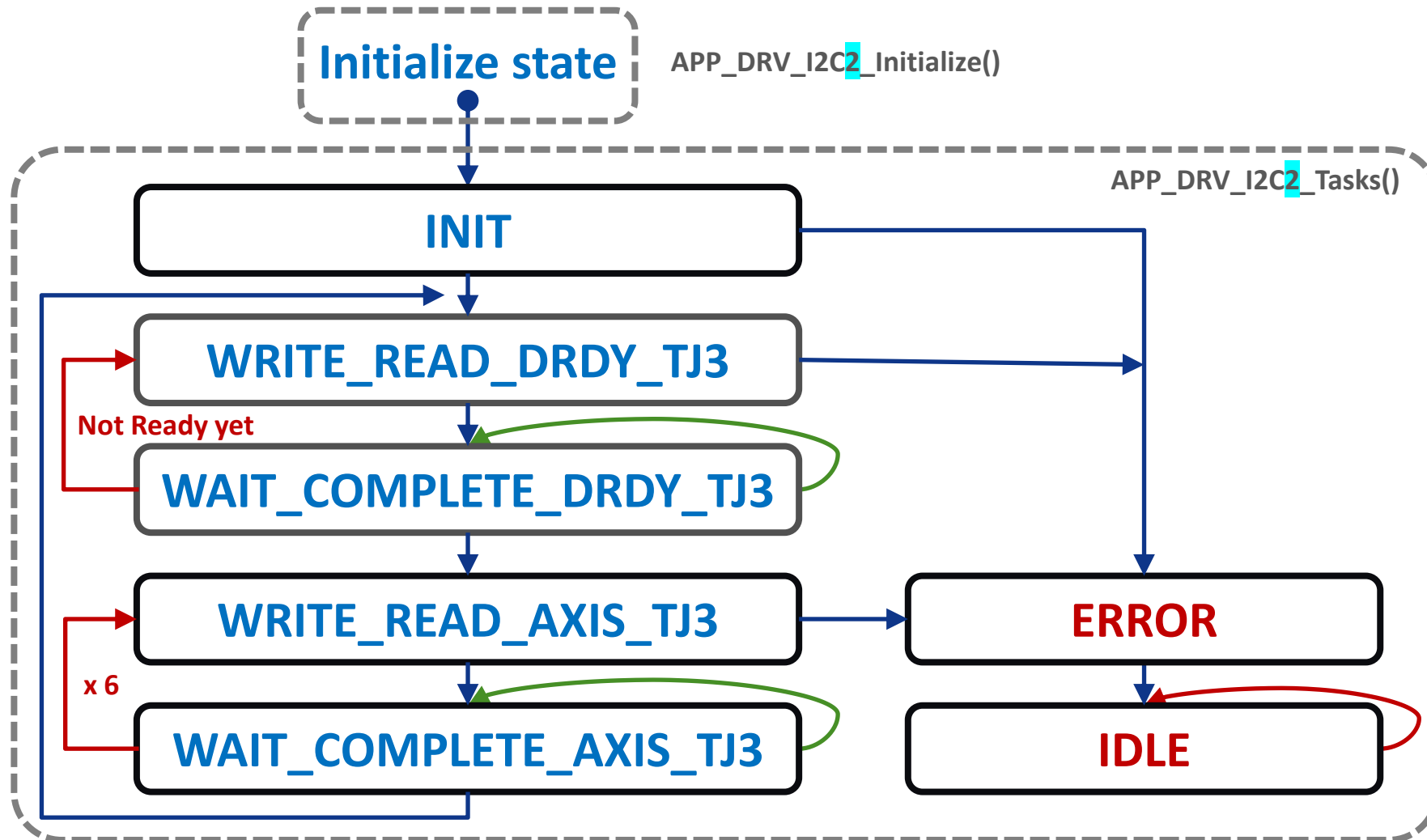
Lab15 Multi-Instance I2C G-Sensor

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**4**” and given the Application Name to “**app_drv_i2c2**”.



⚙️ Lab15 Multi-Instance I2C G-Sensor

- The state machine we want to implement in “`app_drv_i2c2.c`” as below.



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Add more states in “app_drv_i2c2.h”.

```
#ifndef _APP_DRV_I2C2_H
#define _APP_DRV_I2C2_H

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

```
typedef enum
{
```

Keep this line

```
APP_DRV_I2C2_STATE_INIT=0,
```

Comment out this line

```
// APP_DRV_I2C2_STATE_SERVICE_TASKS,
APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3,
APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3,
APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3,
APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3,
APP_DRV_I2C2_STATE_ERROR,
APP_DRV_I2C2_STATE_IDLE,
```

```
} APP_DRV_I2C2_STATES;
```

```
typedef struct
{
```

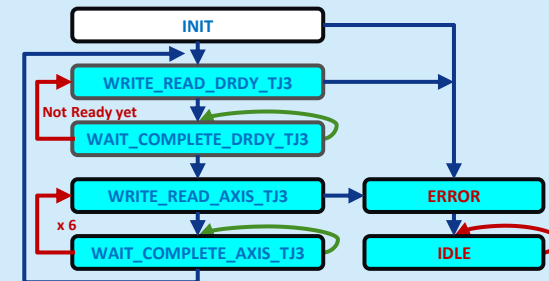
```
APP_DRV_I2C2_STATES state;
```

```
} APP_DRV_I2C2_DATA;
```

```
void APP_DRV_I2C2_Initialize ( void );
...
```

app_drv_i2c2.h

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



✖ Lab15 Multi-Instance I2C G-Sensor

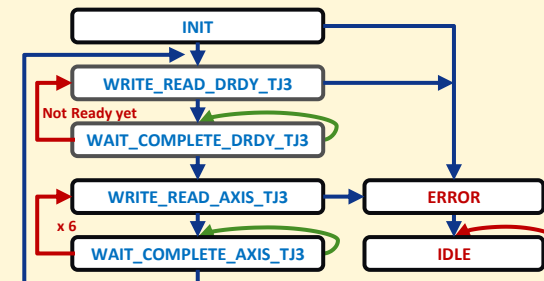
- Construct new states flow of task in “app_drv_i2c2.c”.

```
void APP_DRV_I2C2_Tasks ( void )  
{
```

app_drv_i2c2.c

Overwrite original example code

```
    switch ( app_drv_i2c2Data.state )  
    {  
        case APP_DRV_I2C2_STATE_INIT:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;  
            break;  
  
        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3;  
            break;  
  
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;  
            break;  
  
        case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;  
            break;  
  
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;  
            break;  
  
        case APP_DRV_I2C2_STATE_ERROR:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;  
            break;  
  
        case APP_DRV_I2C2_STATE_IDLE:  
            break;  
    }  
}
```



✂ Lab15 Multi-Instance I2C G-Sensor

- Header include and Global Declarations in “app_drv_i2c2.c”.

Keep this line

```
#include "app_drv_i2c2.h"
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "driver/i2c/drv_i2c.h"
#include "system/time/sys_time.h"
#include <stdio.h>
#include <string.h>
```

app_drv_usart 1 and 2.h
for debug output
sys_time.h for delay function

app_drv_i2c2.c

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

```
#define KXTJ3_ADDRESS 0x0E
```

KXTJ3 ADDR pin tie to GND

```
DRV_HANDLE I2C2Handle_TJ3;
DRV_I2C_TRANSFER_HANDLE I2C2WriteReadHandle_TJ3;
volatile int I2C2Complete_TJ3 = false;
char I2C2WriteBuf_TJ3[20];
char I2C2ReadBuf_TJ3[20];
int AxisOutRegIdx = 0;
char AxisOutReg[6] = { 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B };
char AxisOutByte[6];
```

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

Summary:

```
*/ ...
```

```
APP_DRV_I2C2_DATA app_drv_i2c2Data;
```

XOUT_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

Axis registers sequenced
in AxisOutReg[6]

⚙️ Lab15 Multi-Instance I2C G-Sensor

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

```
APP_DRV_I2C_DATA app_drv_i2c2Data; app_drv_i2c2.c

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

void I2C2TransferHandler_TJ3( DRV_I2C_TRANSFER_EVENT event,
                             DRV_I2C_TRANSFER_HANDLE transferHandle,
                             uintptr_t context )
{
    switch( event )
    {
        case DRV_I2C_TRANSFER_EVENT_PENDING:                break;
        case DRV_I2C_TRANSFER_EVENT_COMPLETE:
            I2C2Complete_TJ3 = true;
            break;
        case DRV_I2C_TRANSFER_EVENT_HANDLE_EXPIRED:
        case DRV_I2C_TRANSFER_EVENT_ERROR:
        case DRV_I2C_TRANSFER_EVENT_HANDLE_INVALID:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
            break;
    }
}

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


⚙️ Lab15 Multi-Instance I2C G-Sensor

- Driver open and send **Software Reset** command in state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )  
{  
    switch ( app_drv_i2c2Data.state )  
    {  
        case APP_DRV_I2C2_STATE_INIT:  
            I2C2Handle_TJ3 = DRV_I2C_Open(DRV_I2C_INDEX_1, DRV_IO_INTENT_READWRITE);  
            if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)  
            {  
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;  
                break;  
            }  
  
            I2C2WriteBuf_TJ3[0] = 0x1D;  
            I2C2WriteBuf_TJ3[1] = 0x80;  
            DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,  
                                     I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );  
            while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=  
                  DRV_I2C_TRANSFER_EVENT_COMPLETE) {}  
  
            SYS_TIME_HANDLE TimeHandle;  
            SYS_TIME_DelayMS(10, &TimeHandle );  
            while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}  
  
            USART1printf("G-Sensor Software Reset Completed\r\n");  
        }  
    }
```

app_drv_i2c2.c

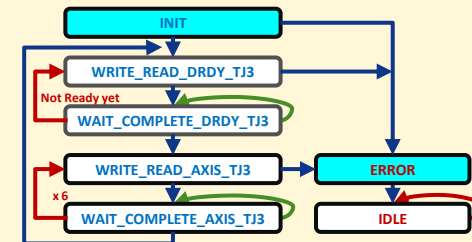
Software Reset

Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
break;

Handling error state message

```
...  
case APP_DRV_I2C2_STATE_ERROR:  
    USART1printf("DRV_I2C2 error!\r\n");
```

Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;
break;



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Read **WHO AM I** register to check the ID is equal to “**0x35**” or not in state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT:
            SYS_TIME_DelayMS(10, &TimeHandle );
            while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

            USART1printf("G-Sensor Software Reset Completed\r\n");

            I2C2WriteBuf_TJ3[0] = 0x0F;
            DRV_I2C_WriteReadTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
                                          I2C2WriteBuf_TJ3, 1, I2C2ReadBuf_TJ3, 1, &I2C2WriteReadHandle_TJ3 );
            while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
                  DRV_I2C_TRANSFER_EVENT_COMPLETE) {}

            if( I2C2ReadBuf_TJ3[0] != 0x35 )
            {
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
                break;
            }

            USART1printf("G-Sensor WHO AM I : 0x%X\r\n", I2C2ReadBuf_TJ3[0]);

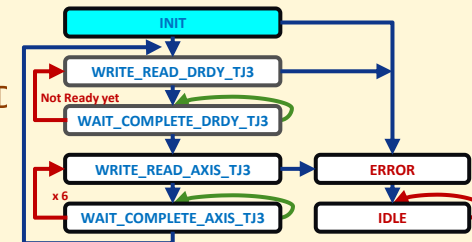
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            break;

        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3:
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_I
            break;

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
            ...
    }
}
```

Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3; break;

Read WHO AM I



⚙️ Lab15 Multi-Instance I2C G-Sensor

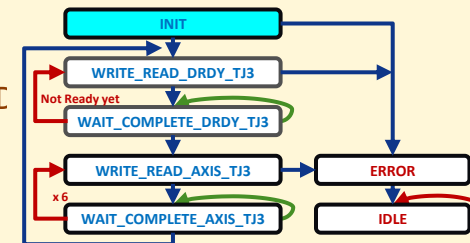
- Write **Initial** Command for operation mode and resolution setup in state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )  
{  
    switch ( app_drv_i2c2Data.state )  
    {  
        case APP_DRV_I2C2_STATE_INIT:  
            ...  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;  
            break;  
        }  
  
        USART1printf("G-Sensor WHO AM I : 0x%X\r\n", I2C2ReadBuf_TJ3[0]);  
  
        I2C2WriteBuf_TJ3[0] = 0x1B;  
        I2C2WriteBuf_TJ3[1] = 0xF0;  
        DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,  
                                I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );  
        while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=  
                DRV_I2C_TRANSFER_EVENT_COMPLETE ) {}  
  
        USART1printf("G-Sensor Initial Completed\r\n");  
        DRV_I2C_TransferEventHandlerSet(I2C2Handle_TJ3, I2C2TransferHandler_TJ3, 0);  
  
        Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;  
        break;  
  
        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_I  
            break;  
  
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:  
            ...  
    }  
}
```

app_drv_i2c2.c

Write Initial

Register Transfer Callback



Lab15 Multi-Instance I2C G-Sensor

- Read **Interrupt Source** register to check **Data Ready** interrupt flag.

app_drv_i2c2.c

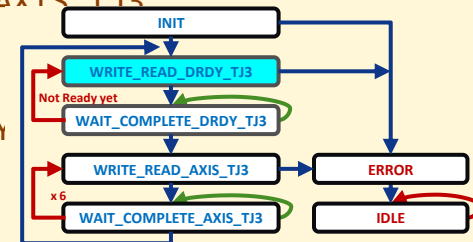
```
Keep this line app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3;  
break;
```

```
case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
    break;
```

```
case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;
    break;
```

```
case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY;
    break;
```

• • •



Lab15 Multi-Instance I2C G-Sensor

- Check **Bit 4** of **Interrupt Source** register, if **Data Ready** for read.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT: ...

        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3: ...

        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                if(I2C2ReadBuf_TJ3[0] & (1<<4))
                {
                    AxisOutRegIdx = 0;
                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
                    break;
                }
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
            }
            break;
    }
}
```

Reset Axis Out Register while Data Read

Keep this line

Reset Axis Out Register Index while Data Ready

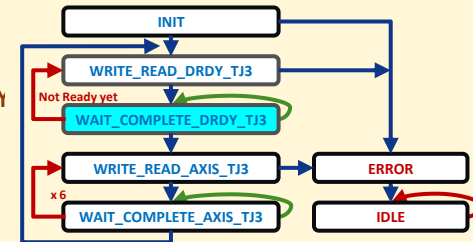
```

case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;
    break;

case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY;
    break;

case APP_DRV_I2C2_STATE_ERROR:
    ...

```



⚙️ Lab15 Multi-Instance I2C G-Sensor

- Read **Axis Output** register in order of **AxisOutReg[]** one by one.

```
void APP_DRV_I2C2_Tasks ( void )  
{  
    switch ( app_drv_i2c2Data.state )  
    {  
        case APP_DRV_I2C2_STATE_INIT: ...  
  
        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3: ...  
  
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3: ...  
  
        case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3:  
            I2C2WriteBuf_TJ3[0] = AxisOutReg[AxisOutRegIdx];  
            DRV_I2C_WriteReadTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,  
                                          I2C2WriteBuf_TJ3, 1, I2C2ReadBuf_TJ3, 1, &I2C2WriteReadHandle_TJ3 );  
  
            if(I2C2WriteReadHandle_TJ3 == DRV_I2C_TRANSFER_HANDLE_INVALID)  
            {  
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;  
                break;  
            }  
  
            char AxisOutReg[6] = { 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B };  
  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3;  
            break;  
  
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;  
            break;  
  
        case APP_DRV_I2C2_STATE_ERROR:  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;  
            break;  
  
        ...  
    }  
}
```

app_drv_i2c2.c

x 6

Keep this line

Flowchart:

```
graph TD  
    INIT[INIT] --> WRITE_READ_DRDY_TJ3[WRITE_READ_DRDY_TJ3]  
    WRITE_READ_DRDY_TJ3 --> WAIT_COMPLETE_DRDY_TJ3[WAIT_COMPLETE_DRDY_TJ3]  
    WAIT_COMPLETE_DRDY_TJ3 --> WRITE_READ_AXIS_TJ3[WRITE_READ_AXIS_TJ3]  
    WRITE_READ_AXIS_TJ3 --> WAIT_COMPLETE_AXIS_TJ3[WAIT_COMPLETE_AXIS_TJ3]  
    WAIT_COMPLETE_AXIS_TJ3 -- x 6 --> WRITE_READ_AXIS_TJ3  
    WRITE_READ_AXIS_TJ3 -- Not Ready yet --> WAIT_COMPLETE_AXIS_TJ3  
    WRITE_READ_AXIS_TJ3 --> ERROR[ERROR]  
    WAIT_COMPLETE_AXIS_TJ3 --> ERROR  
    ERROR --> IDLE[IDLE]
```

⚙️ Lab15 Multi-Instance I2C G-Sensor

- Output gravity of X, Y, Z after to collected **6 bytes** of Axis Data.

```
void APP_DRV_I2C2_Tasks ( void )  
{
```

app_drv_i2c2.c

```
    switch ( app_drv_i2c2Data.state )  
    {
```

```
        case APP_DRV_I2C2_STATE_INIT: ...
```

```
        case APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3: ...
```

```
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_DRDY_TJ3: ...
```

```
        case APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3: ...
```

```
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
```

```
            if(I2C2Complete_TJ3 == true)
```

```
            {  
                I2C2Complete_TJ3 = false;
```

```
                AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];
```

```
                if(++ AxisOutRegIdx >= 6)
```

```
                {
```

```
                    USART1printf("\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",
```

```
                        (short)(AxisOutByte[1]<<8|AxisOutByte[0]),
```

```
                        (short)(AxisOutByte[3]<<8|AxisOutByte[2]),
```

```
                        (short)(AxisOutByte[5]<<8|AxisOutByte[4]));
```

```
                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
```

```
                    break;
```

```
                }
```

```
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;
```

```
            }
```

Keep this line

```
            break;
```

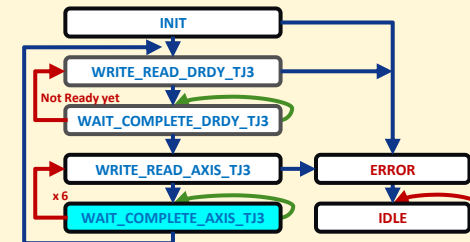
```
        case APP_DRV_I2C2_STATE_ERROR:
```

```
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_IDLE;
```

```
            break;
```

```
        ...
```

XOUT_L	XOUT_H	YOUT_L	YOUT_H	ZOUT_L	ZOUT_H
0x06	0x07	0x08	0x09	0x0A	0x0B

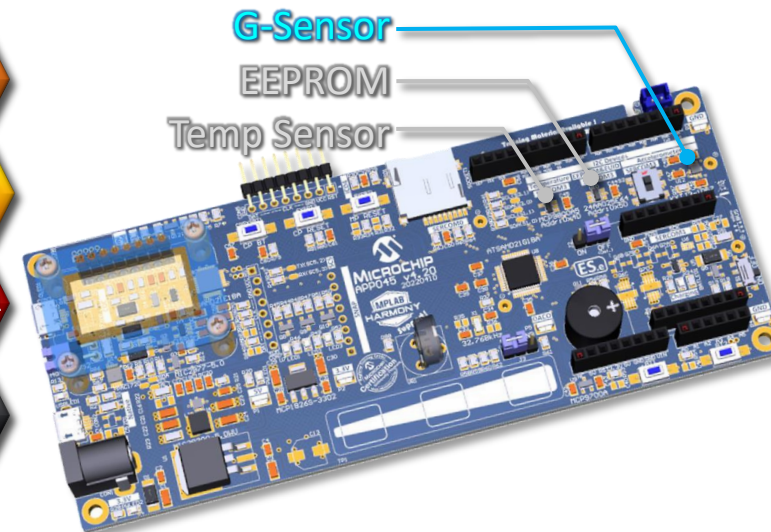
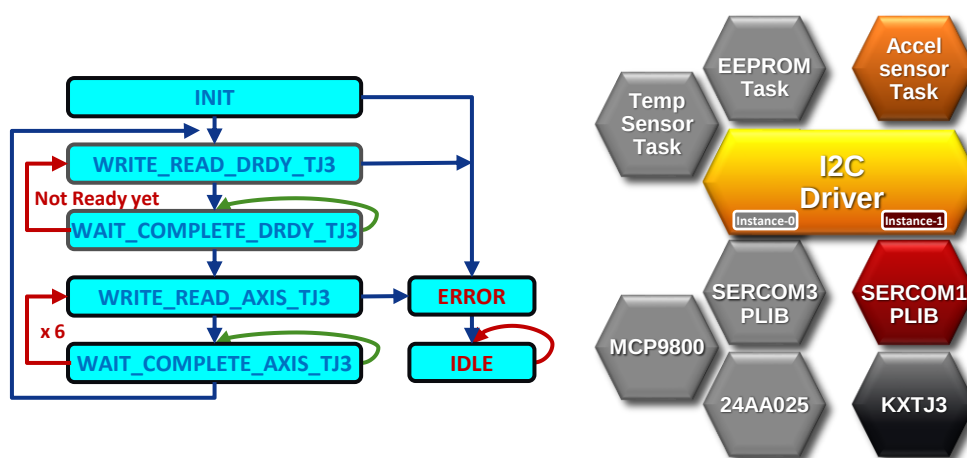
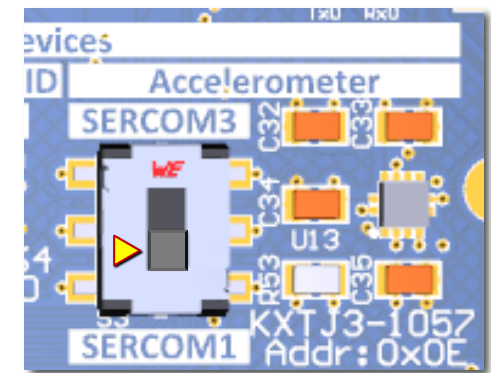


⚙️ Lab15 Multi-Instance I2C G-Sensor

- Compiler and programming your project to EVB.
- Check the UART output in console.

```
COM3 - Tera Term VT
File Edit Setup Control Window Help
DRV_USART1: Hello world
Wrote 0x24 to EEPROM 24AA
G-Sensor Software Reset Completed
G-Sensor WHO AM I : 0x35
G-Sensor Initial Completed
Temp = 26.5      EEPROM = 0x24      G-Sensor = ( -576, 192, 4144)
```

Switch the **S3** to **SERCOM1** on EVB



Advanced Exercise **MPLAB Data Visualizer**

I2C Driver Library

✖ Advanced Exercise : MPLAB Data Visualizer

- Use **MPLAB Data Visualizer** to draw X, Y, Z gravity from **USART2** console.

```
void APP_DRV_I2C2_Tasks ( void )  
{  
    switch ( app_drv_i2c2Data.state )  
    { ...  
    case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:  
        if(I2C2Complete_TJ3 == true)  
        {  
            I2C2Complete_TJ3 = false;  
  
            AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];  
            if(++ AxisOutRegIdx >= 6)  
            {  
                USART1printf("\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",  
                    (short){AxisOutByte[1]<<8|AxisOutByte[0]},  
                    (short){AxisOutByte[3]<<8|AxisOutByte[2]},  
                    (short){AxisOutByte[5]<<8|AxisOutByte[4]});  
  
                unsigned char DVBuf[8] = { 0x03,  
                    AxisOutByte[0], AxisOutByte[1],  
                    AxisOutByte[2], AxisOutByte[3],  
                    AxisOutByte[4], AxisOutByte[5],  
                    0xFC };  
  
                USART2write( DVBuf, 8 );  
  
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;  
                break;  
            }  
            app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_AXIS_TJ3;  
        }  
        break;  
    case APP_DRV_I2C2_STATE_ERROR:  
        ...  
    }
```

app_drv_i2c2.c

Insert between this

Insert between this

Advanced Exercise : MPLAB Data Visualizer

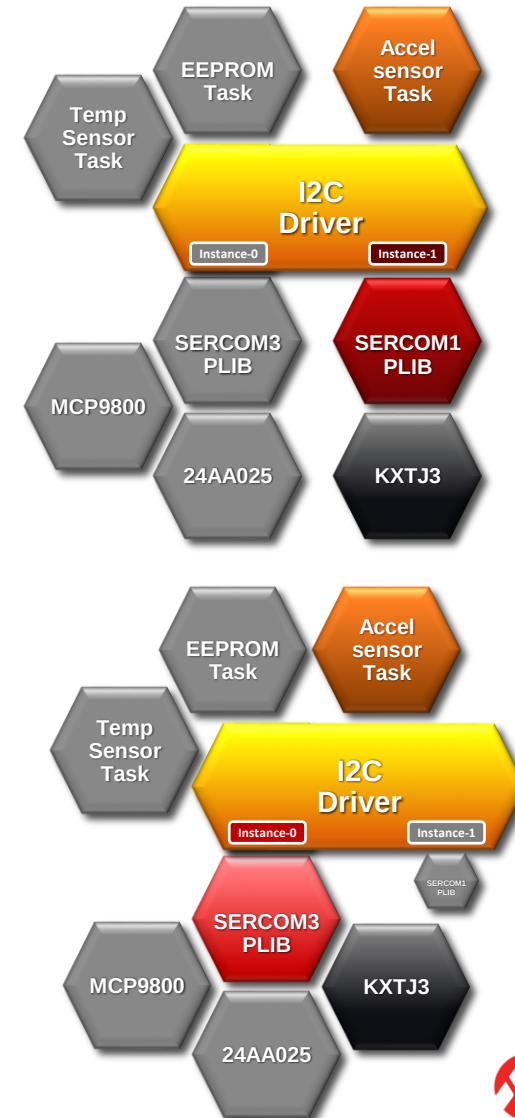
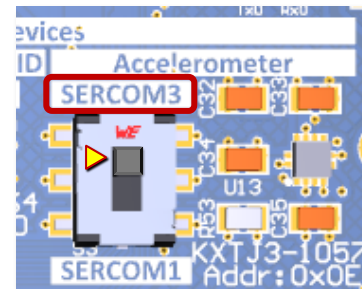
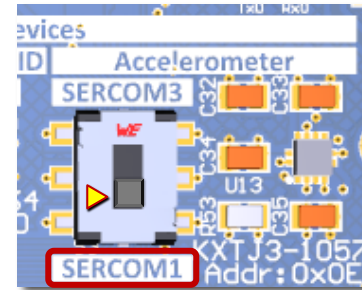
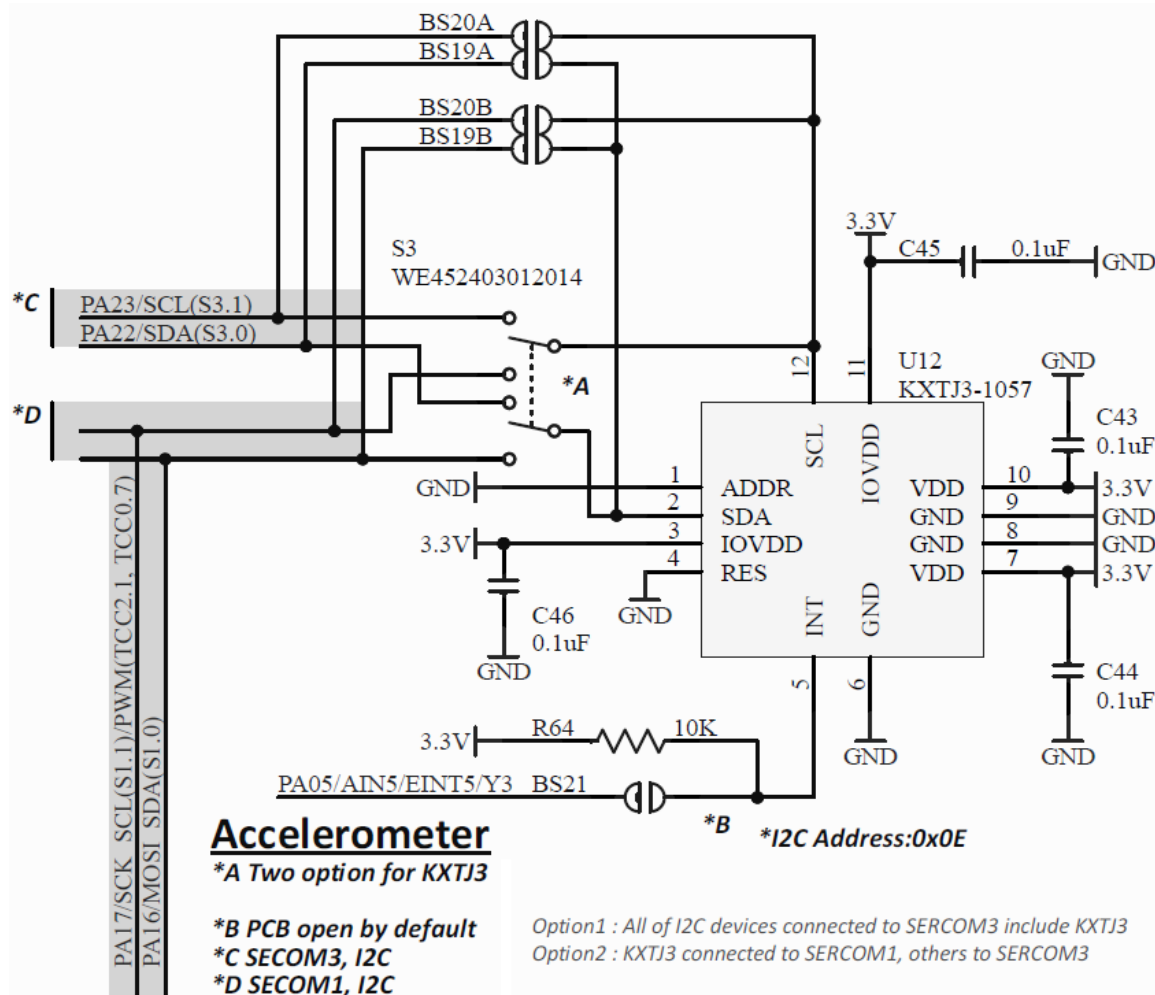
- Result on [MPLAB Data Visualizer](#), Create a streamer **G-Sensor** to have three **int16** variable X, Y, Z gravity parser from **USART2 115200-8-n-1**

Advanced Exercise **I2C Device Auto Probe**

I2C Driver Library

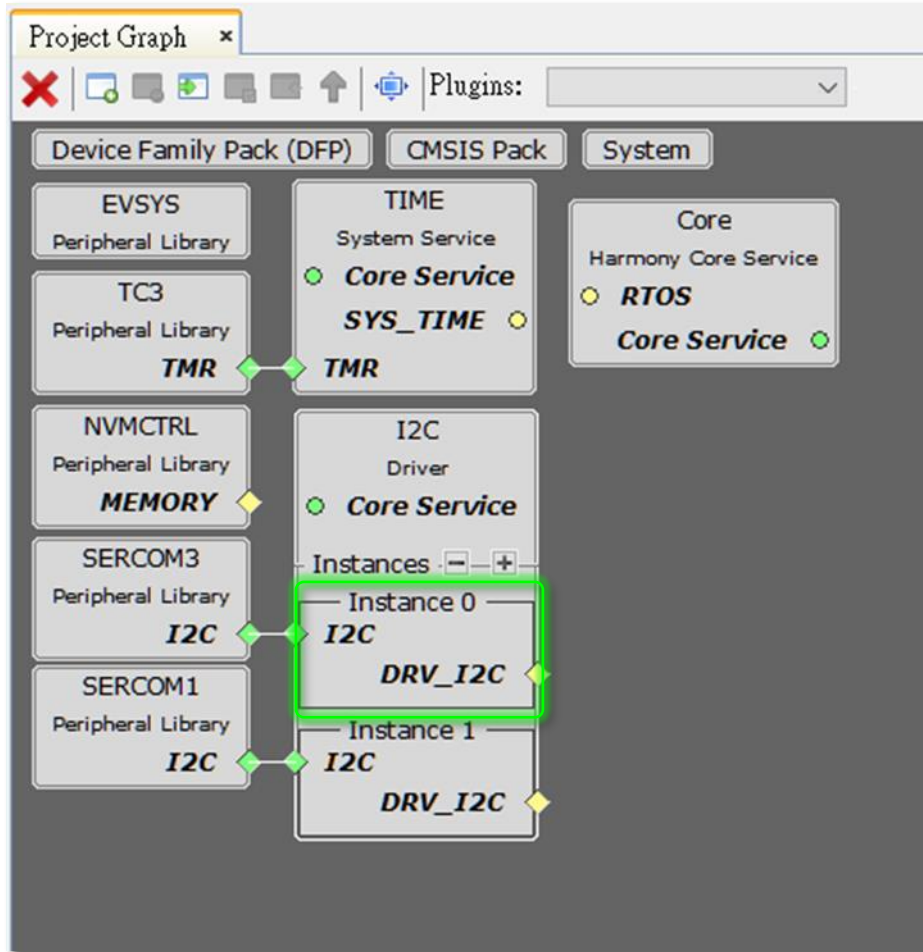
✂ Advanced Exercise : I2C Device Auto Probe

- Auto probe the G-Sensor connection on **PLIB SERCOM1** or **SERCOM3**
- The **Switch S3** could choose either one for connection.

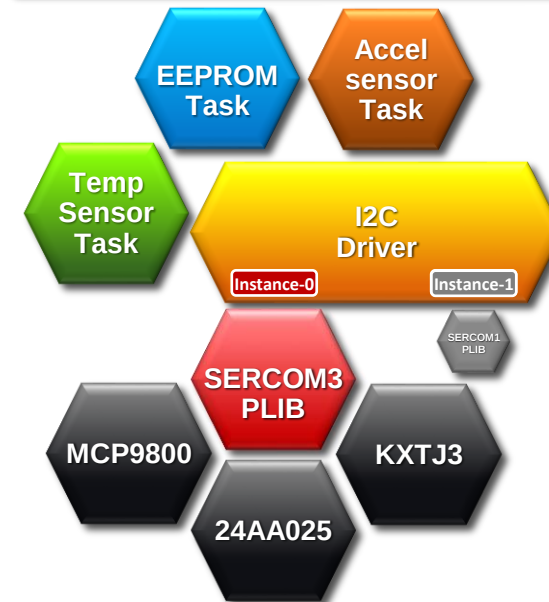
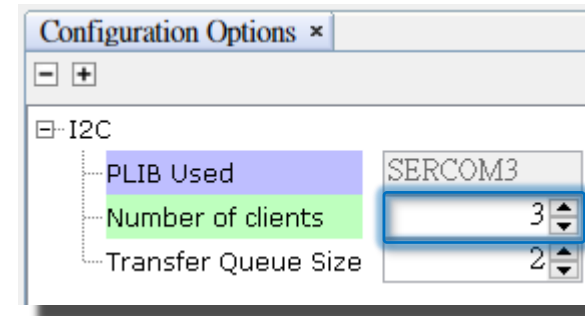


✖ Advanced Exercise : I2C Device Auto Probe

- Increase “**Number of clients**” to **3** in I2C Driver **Instance 0**.



Add Number of clients to 3



✂ Advanced Exercise : I2C Device Auto Probe

- Comment out **Software Reset** in APP_DRV_I2C2_Tasks state **INIT**.

```
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        case APP_DRV_I2C2_STATE_INIT:
            // I2C2Handle_TJ3 = DRV_I2C_Open(DRV_I2C_INDEX_1, DRV_IO_INTENT_READWRITE);
            // if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)
            // {
            //     app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
            //     break;
            // }

            // I2C2WriteBuf_TJ3[0] = 0x1D;
            // I2C2WriteBuf_TJ3[1] = 0x80;
            // DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
            //                           I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );
            // while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
            //                           DRV_I2C_TRANSFER_EVENT_COMPLETE) {}
            // SYS_TIME_HANDLE TimeHandle;
            // SYS_TIME_DelayMS(10, &TimeHandle );
            // while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}

            USART1printf("G-Sensor Software Reset Completed\r\n");

            I2C2WriteBuf_TJ3[0] = 0x0F;
            DRV_I2C_WriteReadTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,
            I2C2WriteBuf_TJ3, 1, I2C2ReadBuf_TJ3, 1, &I2C2WriteReadHandle_TJ3 );
            while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=
            DRV_I2C_TRANSFER_EVENT_COMPLETE) {}
            if( I2C2ReadBuf_TJ3[0] != 0x35 )
            {
                app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;
                ...
            }
        }
    }
}
```

app_drv_i2c2.c

Comment out

Comment out original implement
before Software Reset Message

Comment out before this line

✂ Advanced Exercise : I2C Device Auto Probe

- Auto Probe loop to find G-Sensor on which **Instance** of **DRV_I2C**.

```
void APP_DRV_I2C2_Tasks ( void )  
{  
    switch ( app_drv_i2c2Data.state )  
    {  
        case APP_DRV_I2C2_STATE_INIT:  
            ...  
            // I2C2WriteBuf_TJ3[0] = 0x1D;  
            // I2C2WriteBuf_TJ3[1] = 0x80;  
            // DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,  
            //                          I2C2WriteBuf_TJ3, 2, &I2C2WriteReadHandle_TJ3 );  
            // while( DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3) !=  
            //                          DRV_I2C_TRANSFER_EVENT_COMPLETE) {}  
            // SYS_TIME_HANDLE TimeHandle;  
            // SYS_TIME_DelayMS(10, &TimeHandle );  
            // while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}  
            for( int Inst = DRV_I2C_INDEX_0 ; Inst <= DRV_I2C_INDEX_1 ; Inst++ )  
            {  
                I2C2Handle_TJ3 = DRV_I2C_Open(Inst, DRV_IO_INTENT_READWRITE);  
                if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)  
                {  
                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;  
                    return;  
                }  
                I2C2WriteBuf_TJ3[0] = 0x1D;  
                I2C2WriteBuf_TJ3[1] = 0x80;  
                DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,  
                I2C2WriteBuf_TJ3, 2, & I2C2WriteReadHandle_TJ3 );  
            }  
            ...  
            USART1printf("G-Sensor Software Reset Completed\r\n");  
            ...  
    }  
}
```

app_drv_i2c2.c

Implement a loop for Auto Probe all Instance of DRV_I2C

Insert between this

Exit INIT state if open fail

Insert between this

✖ Advanced Exercise : I2C Device Auto Probe

- Check transfer status to decide G-Sensor exist or not on the bus.

```
case APP_DRV_I2C2_STATE_INIT:
```

```
app_drv_i2c2.c
```

Start of For loop

```
for( int Inst = DRV_I2C_INDEX_0 ; Inst <= DRV_I2C_INDEX_1 ; Inst++ )  
{
```

```
    I2C2Handle_TJ3 = DRV_I2C_Open(Inst, DRV_IO_INTENT_READWRITE);  
    if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)
```

```
    {  
        app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;  
        return;  
    }
```

```
    I2C2WriteBuf_TJ3[0] = 0x1D;
```

```
    I2C2WriteBuf_TJ3[1] = 0x80;
```

```
    DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,  
                             I2C2WriteBuf_TJ3, 2, & I2C2WriteReadHandle_TJ3 );
```

Insert between this

```
    DRV_I2C_TRANSFER_EVENT TransferStatus;
```

```
    do
```

```
    {
```

```
        TransferStatus = DRV_I2C_TransferStatusGet(I2C2WriteReadHandle_TJ3);
```

```
        if( TransferStatus == DRV_I2C_TRANSFER_EVENT_ERROR )
```

```
        {
```

```
            DRV_I2C_Close(I2C2Handle_TJ3);
```

```
            break;
```

```
        }
```

```
    } while( TransferStatus != DRV_I2C_TRANSFER_EVENT_COMPLETE );
```

```
    SYS_TIME_HANDLE TimeHandle;
```

```
    SYS_TIME_DelayMS(10, &TimeHandle );
```

```
    while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}
```

```
    if( TransferStatus == DRV_I2C_TRANSFER_EVENT_COMPLETE )
```

```
        break;
```

Probe failure try next Instance

Probe success exit probe loop

End of For loop

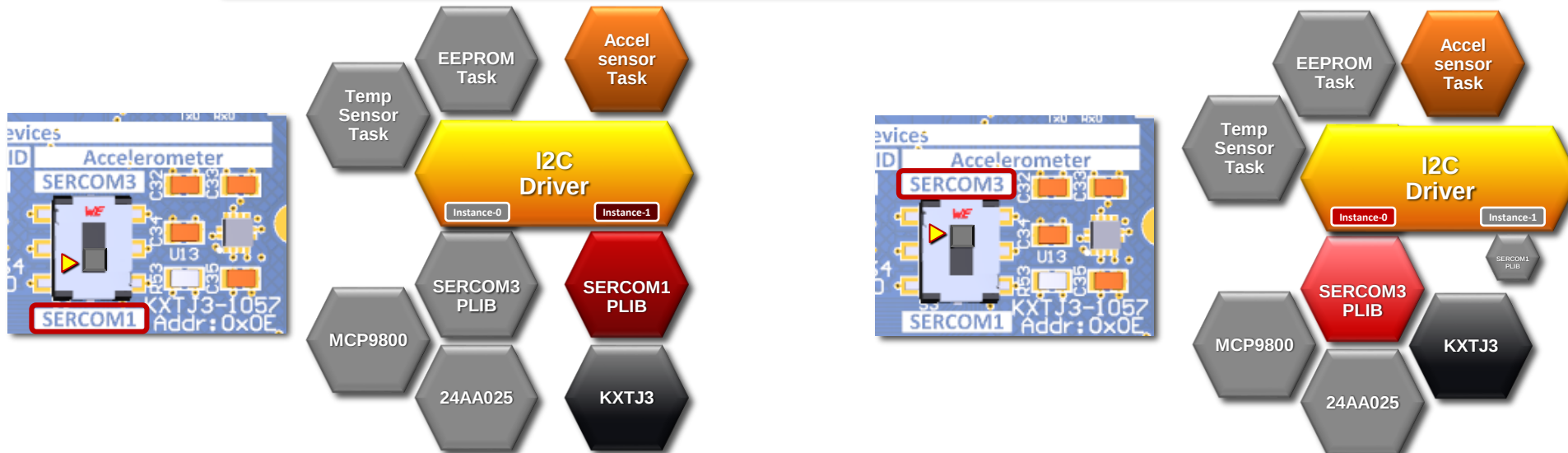
Insert between this

```
}
```

✖ Advanced Exercise : I2C Device Auto Probe

- Try switch **S3** to different connection of G-Sensor to check if the G-Sensor could be **Auto Probed** by our implementation.

```
COM3 - Tera Term VT
File Edit Setup Control Window Help
DRV_USART1: Hello world
Wrote 0x24 to EEPROM 24AA
G-Sensor Software Reset Completed
G-Sensor WHO AM I : 0x35
G-Sensor Initial Completed
Temp = 26.5      EEPROM = 0x24   G-Sensor = ( -576, 192, 4144)
```



SPI Driver Library

Harmony Driver

SPI Driver Library

- **Introduction**

The SPI Driver is a multi-client, multi-instance buffer model-based driver interface which can be used to communicate with various slave devices like EEPROM, ADC etc.

- **Key Features of SPI Driver:**

- **Multi Client:**

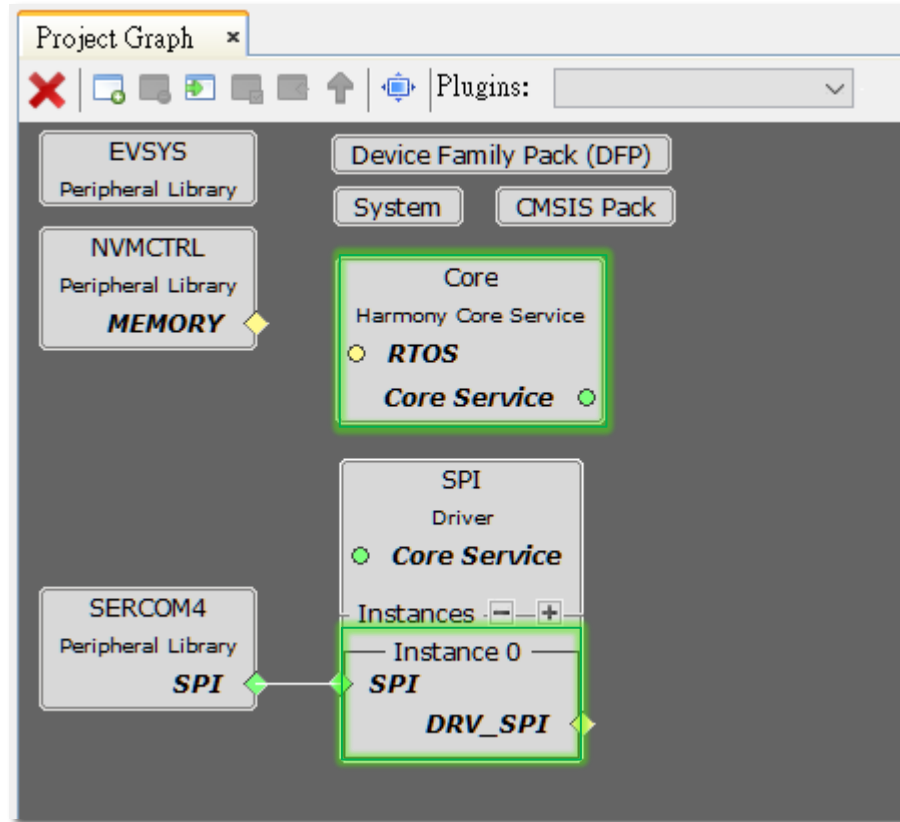
Each instance of SPI driver can have multiple clients. This means the SPI driver supports multiple slaves connected to the same SPI bus.

- **Multi Instance:**

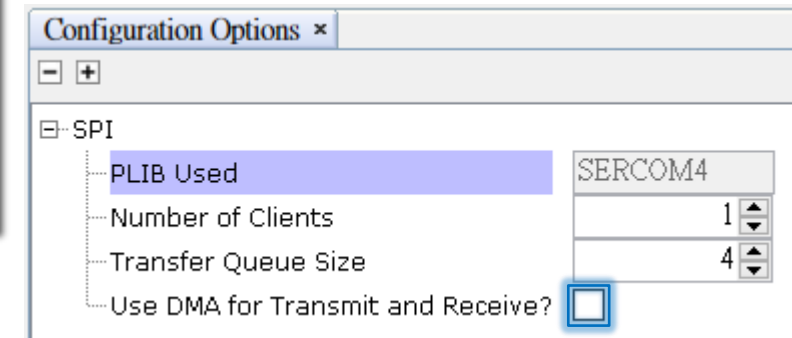
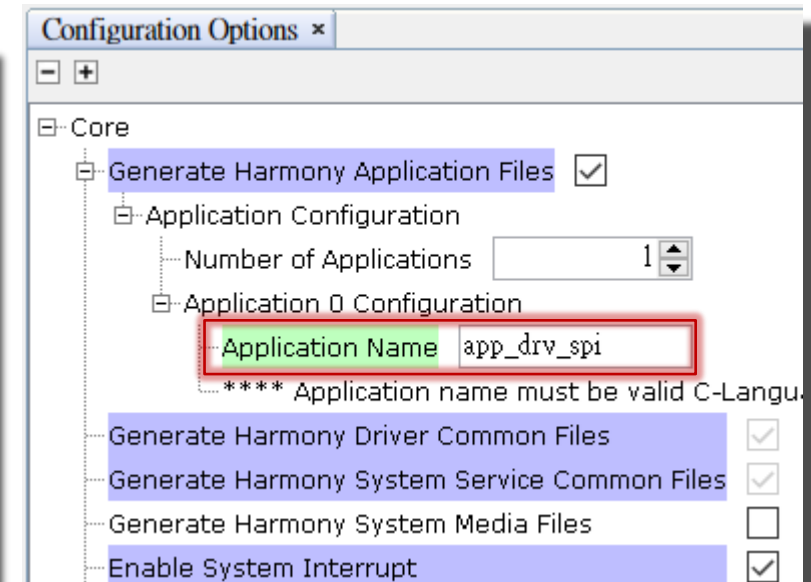
Provides interface to manage multiple instances of SPI peripheral.

- Supports Asynchronous and Synchronous modes of operation.
 - Supports Bare Metal and RTOS environments.
 - Supports DMA.

SPI Driver Library



**Create an Application for
Driver State Machine**



DMA ready

SPI Driver Library

API functions of the SPI Driver library.

- **System Functions**

DRV_SPI_Initialize

Initializes the SPI instance for the specified driver index.

DRV_SPI_Status

Gets the current status of the SPI driver module.

- **Client Setup Functions**

DRV_SPI_Open

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

DRV_SPI_Close

Closes an opened-instance of the SPI driver.

DRV_SPI_TransferSetup

Sets the dynamic configuration of the driver including chip select pin.

DRV_SPI_TransferEventHandlerSet

Allows a client to set a transfer event handling function for the driver to call back when queued transfer has finished.

- **Transfer Functions**

DRV_SPI_ReadTransfer

This is a blocking function that receives data over SPI.

DRV_SPI_WriteTransfer

This is a blocking function that transmits data over SPI.

DRV_SPI_WriteReadTransfer

This is a blocking function that transmits and receives data over SPI.

DRV_SPI_ReadTransferAdd

Queues a read operation.

DRV_SPI_WriteTransferAdd

Queues a write operation.

DRV_SPI_WriteReadTransferAdd

Queues a write-read transfer operation.

- **Transfer Status Functions**

DRV_SPI_TransferStatusGet

Returns transfer add request status.

SPI Driver Library

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

- **Data Types and Constants**

DRV_SPI_CLOCK_PHASE	Identifies SPI Clock Phase Options
DRV_SPI_CLOCK_POLARITY	Identifies SPI Clock Polarity Options
DRV_SPI_CS_POLARITY	Identifies SPI Chip Select Polarity Options
DRV_SPI_DATA_BITS	Identifies SPI bits per transfer
DRV_SPI_TRANSFER_SETUP	Defines the data required to setup the SPI transfer
DRV_SPI_TRANSFER_EVENT	Identifies the possible events that can result from a transfer add request.
DRV_SPI_TRANSFER_HANDLE	Handle identifying the transfer request queued.
DRV_SPI_TRANSFER_EVENT_HANDLER	Pointer to a SPI Driver Transfer Event handler function
DRV_SPI_TRANSFER_HANDLE_INVALID	Definition of an invalid transfer handle.

SPI Driver Library

PLIB SERCOM SPI pad configurations.

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

CTRLA		SERCOM			
DIPO	DOPO	PAD[0]	PAD[1]	PAD[2]	PAD[3]
Hardware Chip Select (SS)					
0x3	0x0	DO	SCK	SS	DI
0x0	0x1	DI	SS	DO	SCK
0x0	0x2	DI	SCK	SS	DO
0x2	0x3	DO	SS	DI	SCK
Software Chip Select (GPIO)					
0x1	0x1	--	DI	DO	SCK
0x1	0x3	DO	DI	--	SCK

CTRLA		SPI			
DIPO	DOPO	DO	SCK	SS	DI
Hardware Chip Select (SS)					
0x3	0x0	PAD[0]	PAD[1]	PAD[2]	PAD[3]
0x0	0x1	PAD[2]	PAD[3]	PAD[1]	PAD[0]
0x0	0x2	PAD[3]	PAD[1]	PAD[2]	PAD[0]
0x2	0x3	PAD[0]	PAD[3]	PAD[1]	PAD[2]
Software Chip Select (GPIO)					
0x1	0x1	PAD[2]	PAD[3]	--	PAD[1]
0x1	0x3	PAD[0]	PAD[3]	--	PAD[1]

Lab 16

Single-Client SPI OLED Display

SPI Driver Library

✖ Lab16 Single-Client SPI OLED Display

- Try add **SPI Driver** through **PLIB SERCOM4 SPI** to control the **OLED** display **SSD1306** to show graphic image and text message.
- A pre-build splash image and text font bitmap will be provided for exercise.
- Please output text result of USART/I2C Driver Labs on **OLED** display as below.

LCM: Hello World

USART1 RX: [The received byte from USART1 console]

USART2 RX: [The received byte from USART2 console]

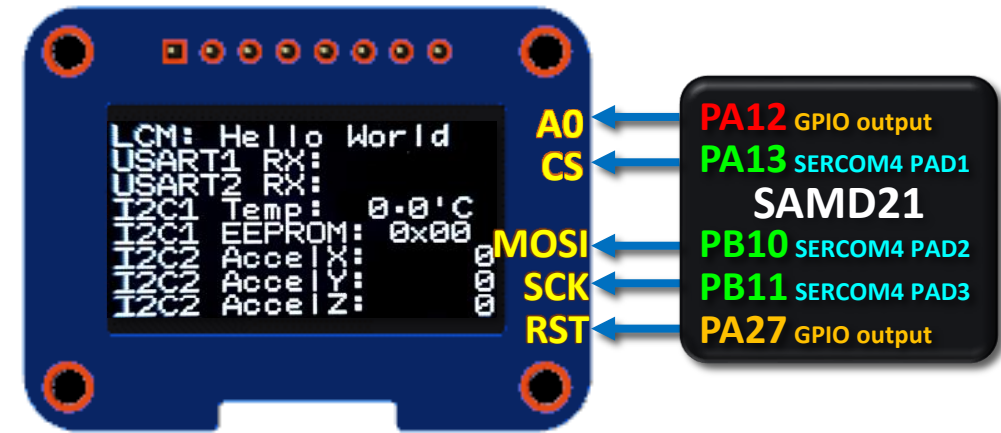
I2C1 Temp: [The temperature from I2C1 device MCP9800A5]

I2C2 EERPOM: [The byte read from I2C1 device 24AA024E64]

I2C2 Acce1X: [The gravity X data from I2C2 device KXTJ3]

I2C2 Acce1Y: [The gravity Y data from I2C2 device KXTJ3]

I2C2 Acce1Z: [The gravity Z data from I2C2 device KXTJ3]



- OLED display (SSD1306) pin configuration: **(SPI Chip Select : SERCOM-SPI Hardware controlled)**

GND/A0 to **PA12** (Custom Name **SSD1306_RS** , GPIO output , latch High)

CS to **PA13** (SERCOM4 PAD1)

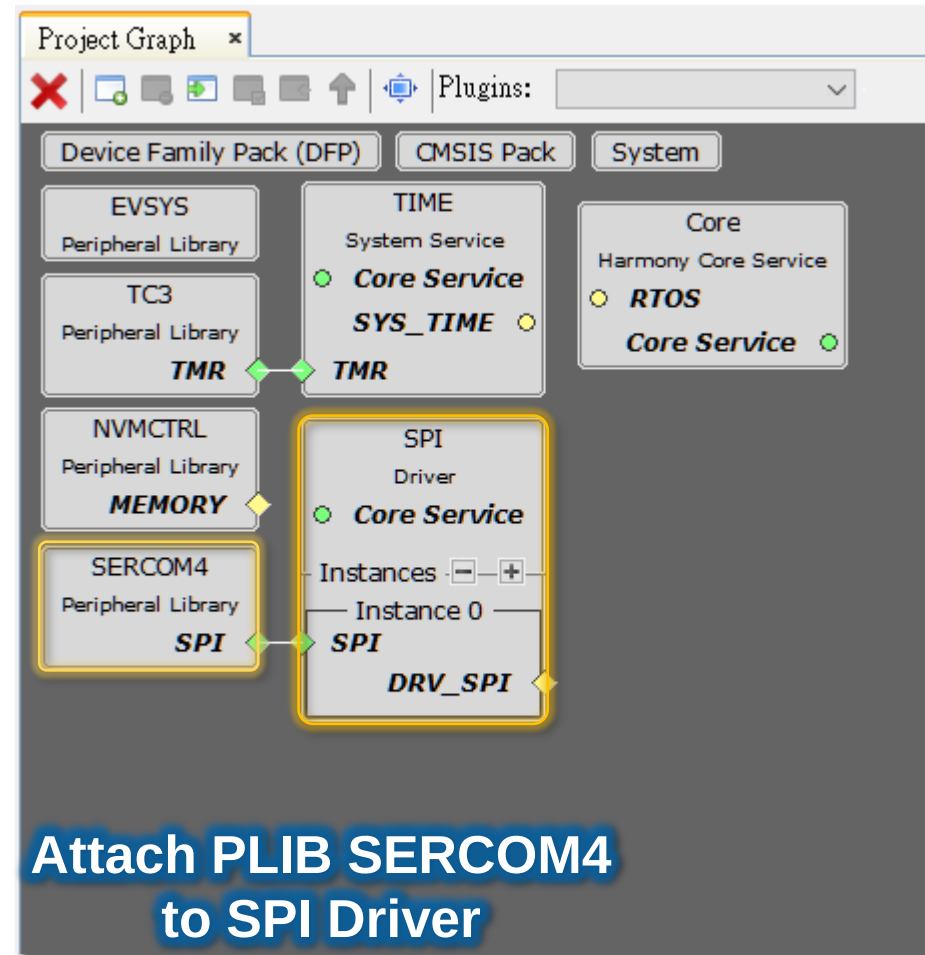
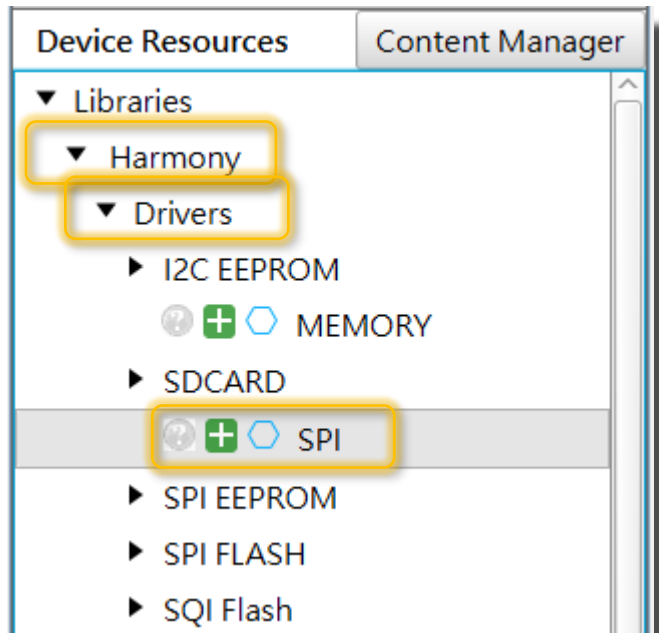
MOSI/SDA to **PB10** (SERCOM4 PAD2)

SCK/SCL to **PB11** (SERCOM4 PAD3)

RST to **PA27** (Custom Name **SSD1306_RESET** , GPIO output , latch High)

⚙️ Lab16 Single-Client SPI OLED Display

- Add **SPI Driver** and attach **PLIB SERCOM4** as its hardware.



⚙️ Lab16 Single-Client SPI OLED Display

- Enable **DMA** of SPI Driver and configure pads of **PLIB SERCOM4**.

The screenshot displays the Microchip Studio configuration environment. On the left, the **Project Graph** shows the hardware components: **SERCOM4** (Peripheral Library **SPI**) is connected to **Instance 0** (SPI **DRV_SPI**). The **Configuration Options** window for **SERCOM4** is open, showing the **SPI Master** mode. The **SPI Data Out Pad** is configured to **DO on PAD[2], SCK on PAD[3] and SS on PAD[1]**. The **Enable SPI Master Hardware Slave Select** checkbox is checked. The **Configuration Options** window for **SPI** is also open, showing the **PLIB Used** as **SERCOM4**. The **Use DMA for Transmit and Receive?** checkbox is checked. The **DMA Channel For Transmit** is set to **0** and the **DMA Channel For Receive** is set to **1**.

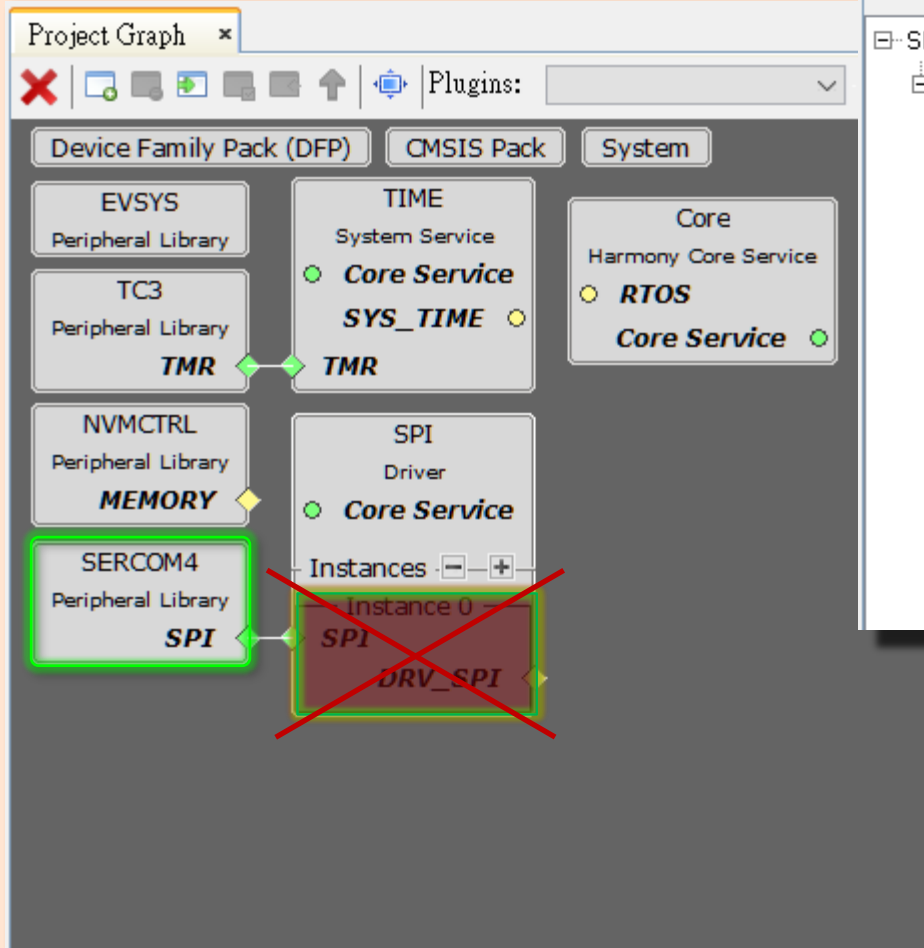
**DO on PAD[2]
SCK on PAD[3]**

**Enable SPI Master
Hardware Slave Select**

**Enable DMA
in SPI Driver**

⚙️ Lab16 Single-Client SPI OLED Display

- **Important!!** Latest SPI Driver will not use the SERCOM-SPI Transfer Parameters as initial value of SPI Driver.



Configuration Options x

SERCOM4

Select SERCOM Operation Mode: SPI Master

Enable Interrupts ? ☒

Enable operation in Standby mode ☐

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

SPI Data In Pad Selection: SERCOM PAD[0]

SPI Data Order: MSB is transferred first

SPI Speed in Hz: 1,000,000

SPI Data Character Size: 8 bits

SPI Clock Phase: The data is sampled on a leading SCK edge and changed on a trailing SCK edge

SPI Clock Polarity: SCK is low when idle

Enable SPI Master Hardware Slave Select ☒

SPI Receiver Enable ☒

SPI Transfer Mode 0 is Selected

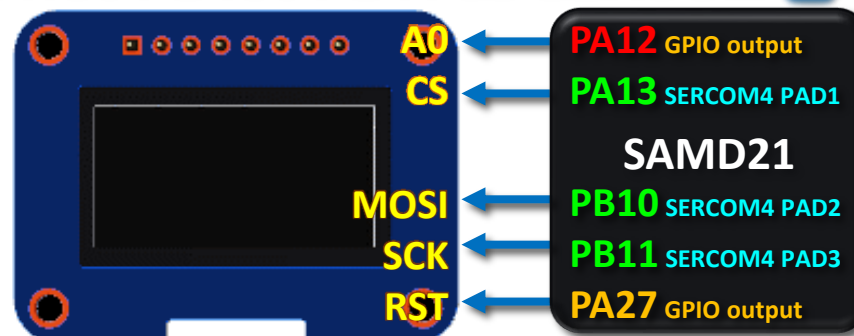
SERCOM4 - SPI settings will not use for the initial value of SPI Driver.
It's active on SERCOM4 - SPI PLIB Only

✖ Lab16 Single-Client SPI OLED Display

- Configure **PA12** as **GPIO output**, Latch **High**, Custom named as **SSD1306_RS**
- Configure **PA27** as **GPIO output**, Latch **High**, Custom named as **SSD1306_RESET**
- Enable **PLIB SERCOM4** pads **PA13**, **PB10** and **PB11** of SPI Driver Instance 0.

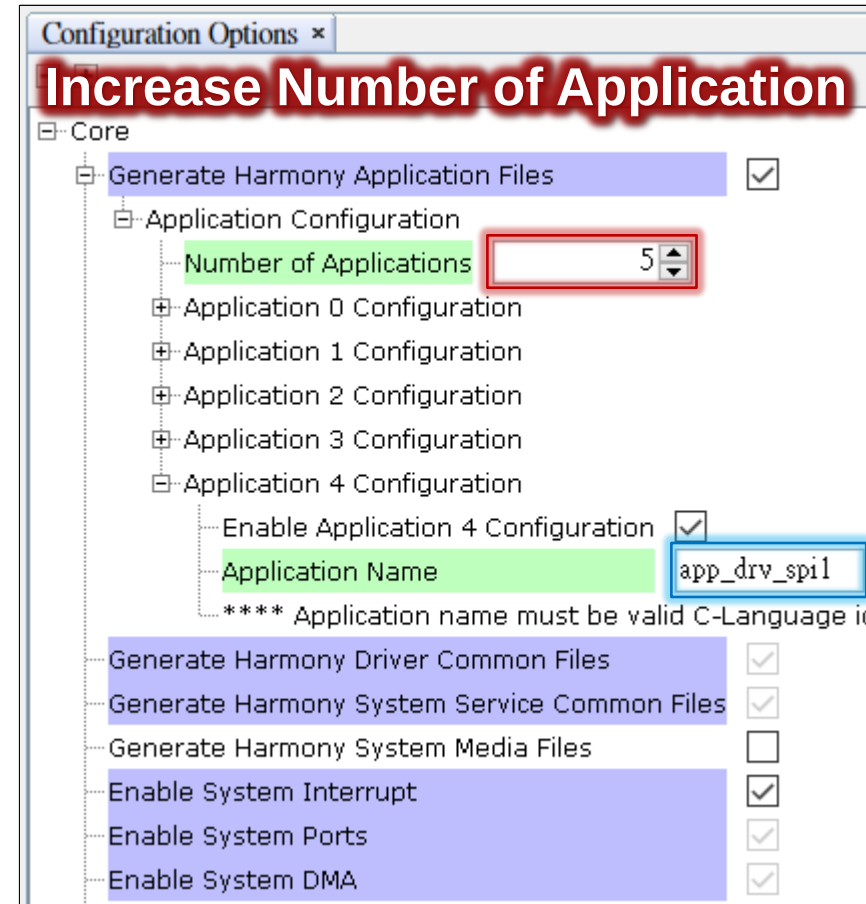
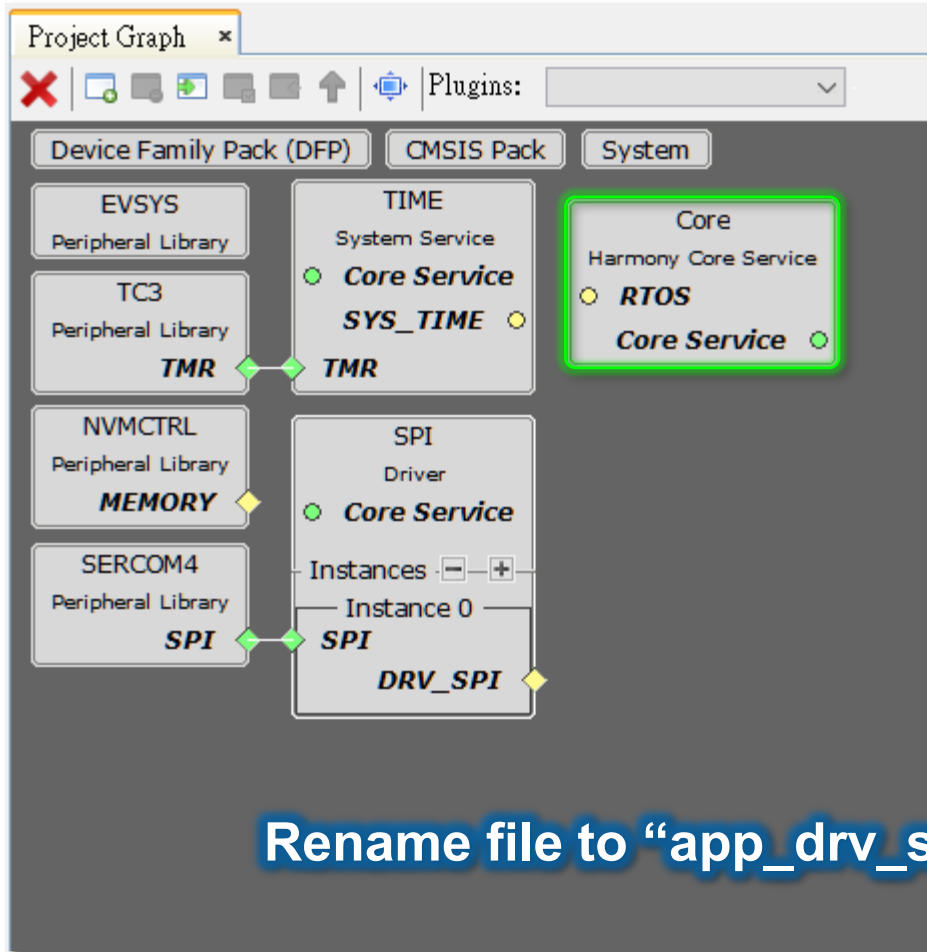
Project Graph* Pin Diagram * Pin Table * Pin Settings *									
Order: Pins Table View ☒ Easy View									
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
19	PB10		SERCOM4_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
20	PB11		SERCOM4_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL
21	PA12	SSD1306_RS	GPIO	Digital	Out	High	☐	☐	NORMAL
22	PA13		SERCOM4_PAD1	Digital	High Impedance	n/a	☐	☐	NORMAL
39	PA27	SSD1306_RESET	GPIO	Digital	Out	High	☐	☐	NORMAL

PA12 as GPIO output, custom name SSD1306_RS , Latch to High
PA27 as GPIO output, custom name SSD1306_RESET , Latch to High
Configure PA13 / PB10 / PB11 as SERCOM4_PAD1 / PAD2 / PAD3



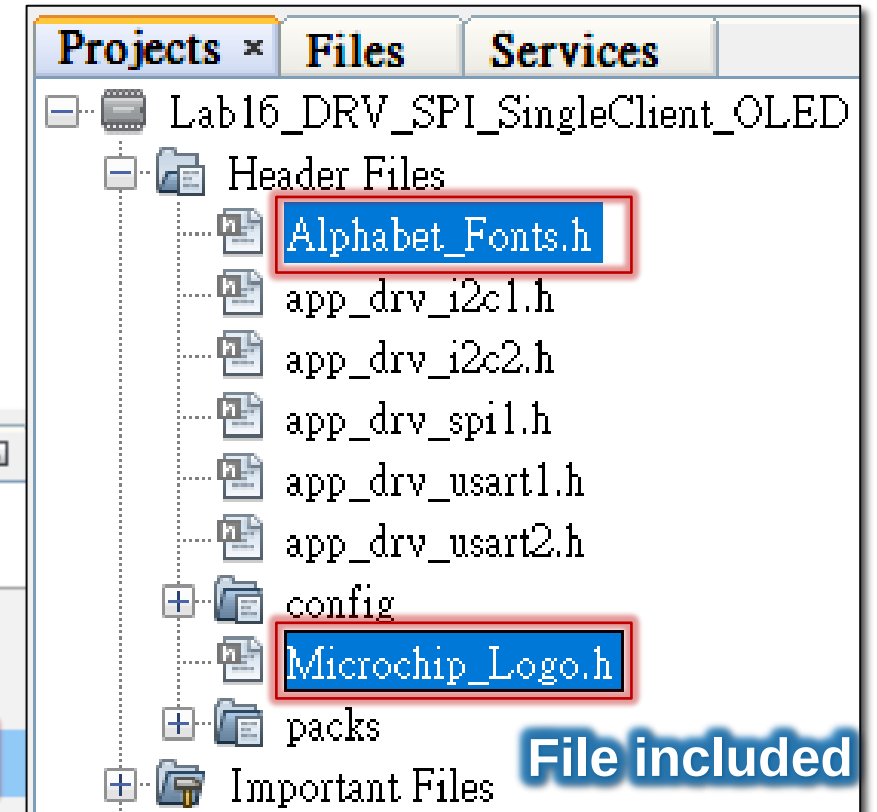
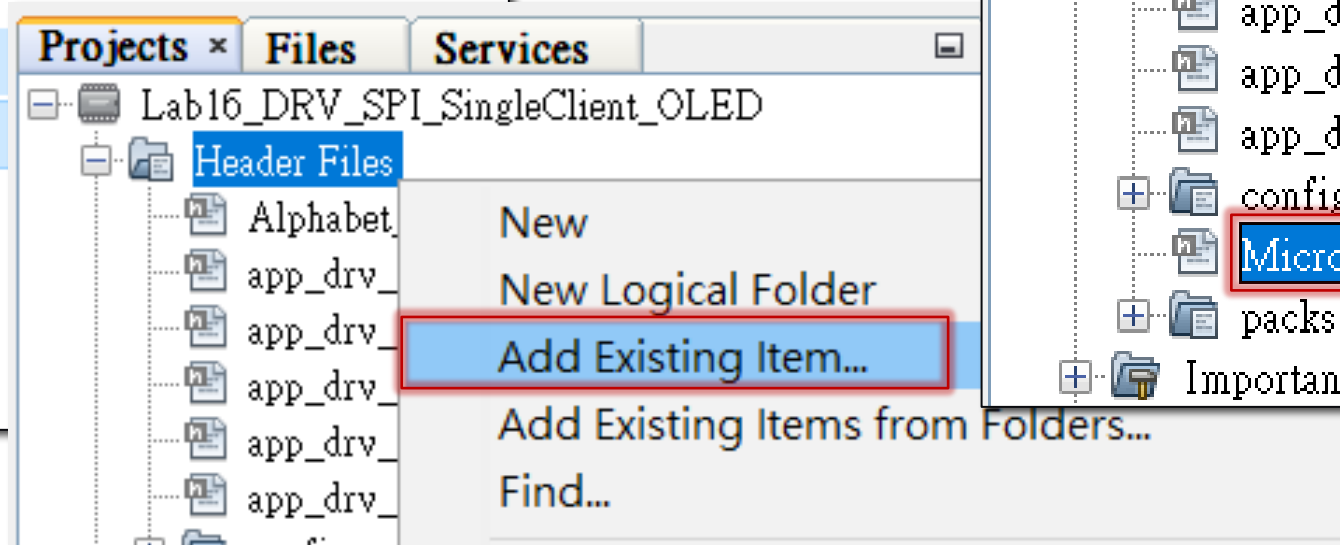
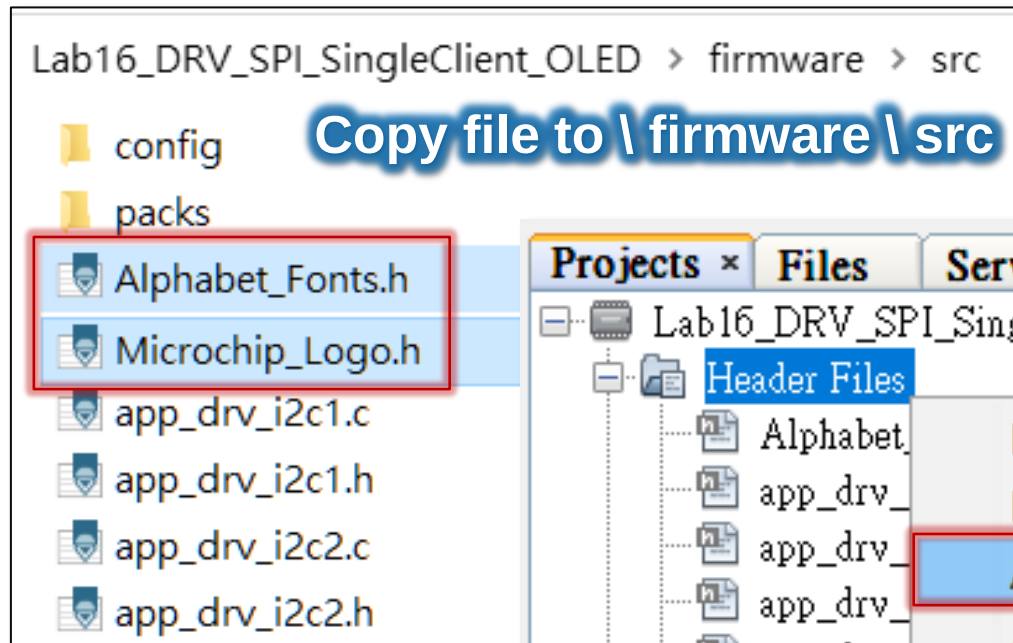
✖ Lab16 Single-Client SPI OLED Display

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**5**” and given the Application Name to “**app_drv_spi1**”.



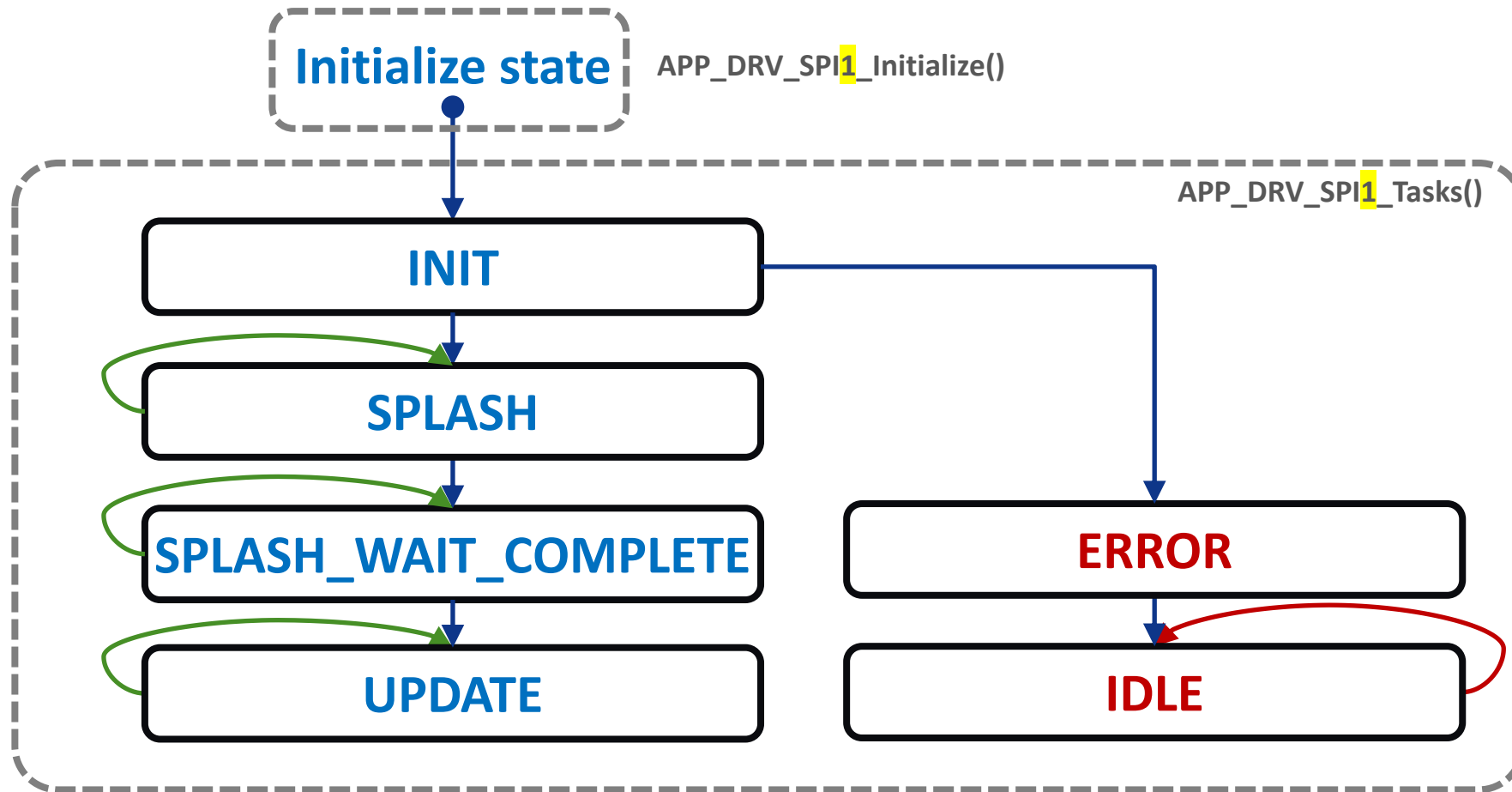
✖ Lab16 Single-Client SPI OLED Display

- Copy pre-build bitmap header files to project folder `\firmware\src`.
 - Splash Logo bitmap image “`Microchip_Logo.h`”.
 - Text Font bitmap image “`Alphabet_Fonts.h`”.
- Add two header files to project in MPLAB X IDE.



⚙️ Lab16 Single-Client SPI OLED Display

- The state machine to implement in `app_drv_spi1.c` as below.



⚙️ Lab16 Single-Client SPI OLED Display

- Add states in “app_drv_spi1.h”.

```
#ifndef _APP_DRV_SPI1_H
#define _APP_DRV_SPI1_H

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

```
typedef enum
{
```

Keep this line

```
APP_DRV_SPI1_STATE_INIT=0,
// APP_DRV_SPI1_STATE_SERVICE_TASKS,
APP_DRV_SPI1_STATE_SPLASH,
APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE,
APP_DRV_SPI1_STATE_UPDATE,
APP_DRV_SPI1_STATE_ERROR,
APP_DRV_SPI1_STATE_IDLE,
```

Comment out this line

```
} APP_DRV_SPI1_STATES;
```

```
typedef struct
{
```

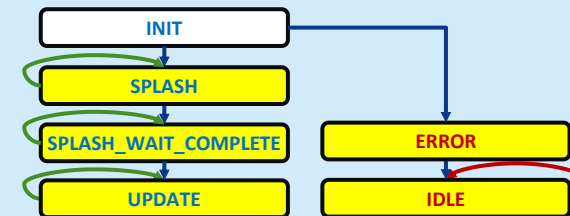
```
    APP_DRV_SPI1_STATES state;
```

```
} APP_DRV_SPI1_DATA;
```

```
void APP_DRV_SPI1_Initialize ( void );
void APP_DRV_SPI1_Tasks( void );
...
```

app_drv_spi1.h

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



✖ Lab16 Single-Client SPI OLED Display

- Construct new states flow of task in “app_drv_spi1.c”.

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

app_drv_spi1.c

Function:

```
void APP_DRV_SPI1_Tasks ( void )
```

```
...
```

Overwrite original example code

```
void APP_DRV_SPI1_Tasks ( void )
```

```
{
```

```
switch ( app_drv_spi1Data.state )
```

```
{
```

```
case APP_DRV_SPI1_STATE_INIT:
```

```
    app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;
```

```
    break;
```

```
case APP_DRV_SPI1_STATE_SPLASH:
```

```
    app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE;
```

```
    break;
```

```
case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:
```

```
    app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;
```

```
    break;
```

```
case APP_DRV_SPI1_STATE_UPDATE:
```

```
    break;
```

```
case APP_DRV_SPI1_STATE_ERROR:
```

```
    app_drv_spi1Data.state = APP_DRV_SPI1_STATE_IDLE;
```

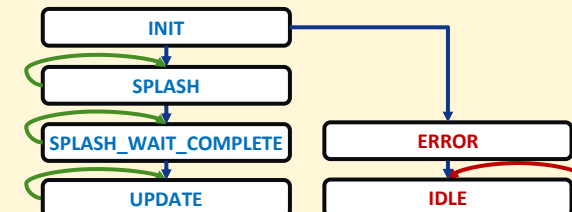
```
    break;
```

```
case APP_DRV_SPI1_STATE_IDLE:
```

```
    break;
```

```
}
```

```
}
```



✂ Lab16 Single-Client SPI OLED Display

- Header include and Global Declarations in “app_drv_spi1.c”.

```
Keep this line #include "app_drv_spi1.h"
#include "app_drv_usart1.h"
#include "driver/spi/drv_spi.h"
#include "system/time/sys_time.h"
#include "peripheral/port/plib_port.h"
#include "Alphabet_Fonts.h"
#include "Microchip_Logo.h"
#include <stdio.h>
#include <string.h>

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

#define LCM_WIDTH 128
#define LCM_HEIGHT 64
int LogoX=126;
const uint8_t LCM_InitCMD[] = {
    0xAE, 0xD5, 0x80, 0xA8, 0x3F, 0xD3, 0x00, 0x40, 0x8D, 0x14, 0x20, 0x02, 0xA1,
    0xC8, 0xDA, 0x12, 0x81, 0xCF, 0xD9, 0xF1, 0xDB, 0x40, 0xA4, 0xA6, 0xAF
};
enum {
    LCM_BUS_CMD = 0,
    LCM_BUS_DAT
};

DRV_HANDLE SPI1Handle;
SYS_TIME_HANDLE SplashTimerHandle;

// *****
```

app_drv_spi11.c

DRV_USART1 for debug message
SYS_TIME for delay function
PLIB_PORT for GPIO control
Alphabet_Fonts.h for Font bitmap
Microchip_Logo.h for Splash Logo

Splash animation initial position X
OLED initial commands

OLED BUS selector

SPI handle
Delay Timer handle

✖ Lab16 Single-Client SPI OLED Display

- Implement LCM local function for bus write and draw bitmap.

```
// *****  
// Section: Application Local Functions  
// *****  
// *****  
  
void LCM_BusWrite( int bus, void *bytes, size_t counts )  
{  
    DRV_SPI_TRANSFER_HANDLE SPI_WriteHandle;  
  
    if ( bus==LCM_BUS_CMD ) SSD1306_RS_Clear();  
    else if( bus==LCM_BUS_DAT ) SSD1306_RS_Set();  
  
    DRV_SPI_WriteTransferAdd( SPI_WriteHandle, bytes, counts, &SPI_WriteHandle );  
    while( DRV_SPI_TransferStatusGet(SPI_WriteHandle)!=DRV_SPI_TRANSFER_EVENT_COMPLETE );  
}  
  
void LCM_SetCursor( uint8_t x, uint8_t y )  
{  
    uint8_t OUT[3] = { 0x0F & x, 0x10 + (x >> 4), 0xB0 + y };  
  
    LCM_BusWrite( LCM_BUS_CMD, (void*)OUT, 3 );  
}  
  
void LCM_DrawBitmap( uint8_t sx, uint8_t sy, uint8_t width, uint8_t height,  
                    const unsigned char *byte )  
{  
    for( int y=0; y<(height/8); y++ )  
    {  
        LCM_SetCursor( sx, sy+y );  
        LCM_BusWrite( LCM_BUS_DAT, (void*)byte, width-sx );  
        byte+=width;  
    }  
}
```

app_drv_spi11.c

OLED Bus write function
Control GPIO to choose
Command or Data transfer

OLED Cursor Set

Draw Bitmap image

⚙️ Lab16 Single-Client SPI OLED Display

- Driver Open, OLED Hardware reset, initial in state **INIT** and handling **ERROR** state.

```
void APP_DRV_SPI1_Tasks ( void )  
{  
    switch ( app_drv_spi1Data.state )  
    {  
        case APP_DRV_SPI1_STATE_INIT:  
            SSD1306_RESET_Clear();  
            SPI1Handle = DRV_SPI_Open(DRV_SPI_INDEX_0, DRV_IO_INTENT_WRITE);  
            if(SPI1Handle == DRV_HANDLE_INVALID)  
            {  
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_ERROR;  
                break;  
            }  
            SSD1306_RESET_Set();  
  
            DRV_SPI_TRANSFER_SETUP SPI1Setup;  
            SPI1Setup.baudRateInHz = 1000000;  
            SPI1Setup.clockPhase = DRV_SPI_CLOCK_PHASE_VALID_LEADING_EDGE;  
            SPI1Setup.clockPolarity = DRV_SPI_CLOCK_POLARITY_IDLE_LOW;  
            SPI1Setup.dataBits = DRV_SPI_DATA_BITS_8;  
            SPI1Setup.chipSelect = (SYS_PORT_PIN)SYS_PORT_PIN_NONE;  
            SPI1Setup.csPolarity = DRV_SPI_CS_POLARITY_ACTIVE_LOW;  
            DRV_SPI_TransferSetup(SPI1Handle, &SPI1Setup);  
  
            LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );  
  
            app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;  
            break;  
            ...  
  
        case APP_DRV_SPI1_STATE_ERROR:  
            USART1printf("DRV_SPI1 error!\r\n");  
  
            app_drv_spi1Data.state = APP_DRV_I2C1_STATE_IDLE;
```

app_drv_spi11.c

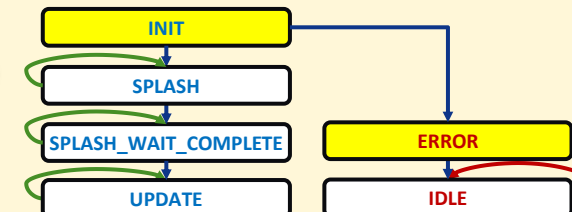
OLED hardware reset by GPIO

SPI Driver Open for write only

Individual SPI Setup for each SPI Instance

Write OLED initial commands

Handling error state message



✂ Lab16 Single-Client SPI OLED Display

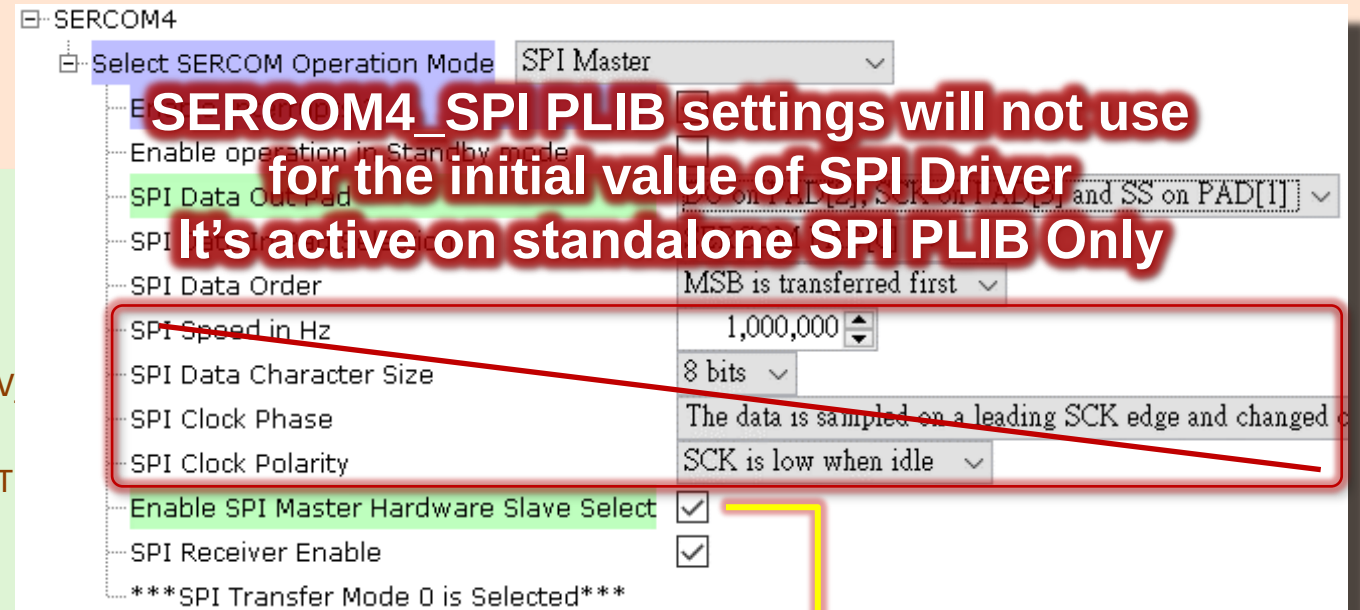
- Please always execute the correct SPI Transfer Setup for each SPI Driver Client by use the **DRV_SPI_TransferSetup()**.

```
void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
        case APP_DRV_SPI1_STATE_INIT:
            SSD1306_RESET_Clear();
            SPI1Handle = DRV_SPI_Open(DRV_SPI_INDEX_0, DRV_SPI_HANDLE_INVALID);
            if(SPI1Handle == DRV_HANDLE_INVALID)
            {
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_ERROR;
                break;
            }
            SSD1306_RESET_Set();
        }
    }
```

```
DRV_SPI_TRANSFER_SETUP SPI1Setup;
SPI1Setup.baudRateInHz = 1000000;
SPI1Setup.clockPhase = DRV_SPI_CLOCK_PHASE_VALID_LEADING_EDGE;
SPI1Setup.clockPolarity = DRV_SPI_CLOCK_POLARITY_IDLE_LOW;
SPI1Setup.dataBits = DRV_SPI_DATA_BITS_8;
SPI1Setup.chipSelect = (SYS_PORT_PIN)SYS_PORT_PIN_NONE;
SPI1Setup.csPolarity = DRV_SPI_CS_POLARITY_ACTIVE_LOW;
DRV_SPI_TransferSetup(SPI1Handle, &SPI1Setup);
```

```
LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );
```

```
app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;
break;
...
```



⚙️ Lab16 Single-Client SPI OLED Display

- Draw animated image in state **SPLASH** and a delay while finish.

```
void APP_DRV_SPI1_Tasks ( void )  
{  
    switch ( app_drv_spi1Data.state )  
    {  
        case APP_DRV_SPI1_STATE_INIT:  
            ...  
            LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );  
  
            app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;  
            break;  
  
        case APP_DRV_SPI1_STATE_SPLASH:  
            LCM_DrawBitmap( LogoX, 0, LCM_WIDTH, LCM_HEIGHT, Microchip_Logo );  
            LogoX = LogoX-2;  
            if( LogoX<0 )  
            {  
                SYS_TIME_DelayMS(1000, &SplashTimerHandle);  
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE;  
            }  
            break;  
  
        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:  
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == true )  
            {  
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;  
            }  
            break;  
  
        case APP_DRV_SPI1_STATE_UPDATE:  
            break;  
  
        ...  
    }  
}
```

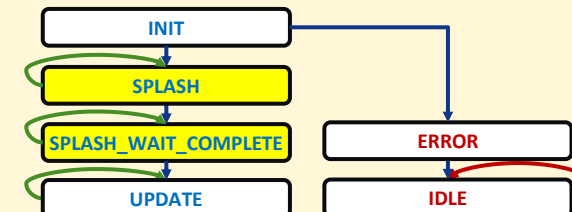
app_drv_spi11.c

Draw animated logo bitmap

Setup one second delay after finish

Keep this line

Wait one second delay



⚙️ Lab16 Single-Client SPI OLED Display

- Please switch the **S3** to **SERCOM1**,
If you **have not** implemented the **Advanced Exercise : I2C Auto Probe** after **Lab15**.

```
void SYS_Tasks ( void )
{
    /* Maintain Device Drivers */

    /* Maintain Middleware & Other Libraries */

    /* Maintain the application's state machine. */
    /* Call Application task APP_DRV_USART1. */
    APP_DRV_USART1_Tasks();

    /* Call Application task APP_DRV_USART2. */
    APP_DRV_USART2_Tasks();

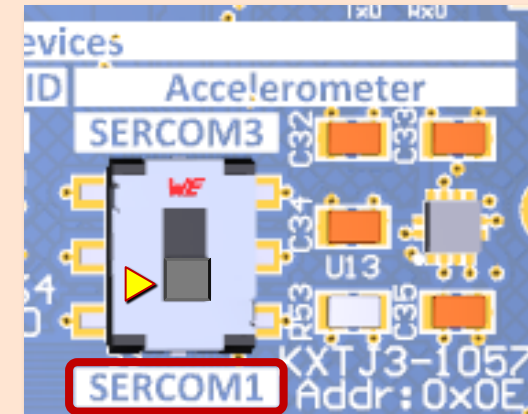
    /* Call Application task APP_DRV_I2C1. */
    APP_DRV_I2C1_Tasks();

    /* Call Application task APP_DRV_I2C2. */
    APP_DRV_I2C2_Tasks();

    /* Call Application task APP_DRV_SPI1. */
    APP_DRV_SPI1_Tasks();

    ...
}
```

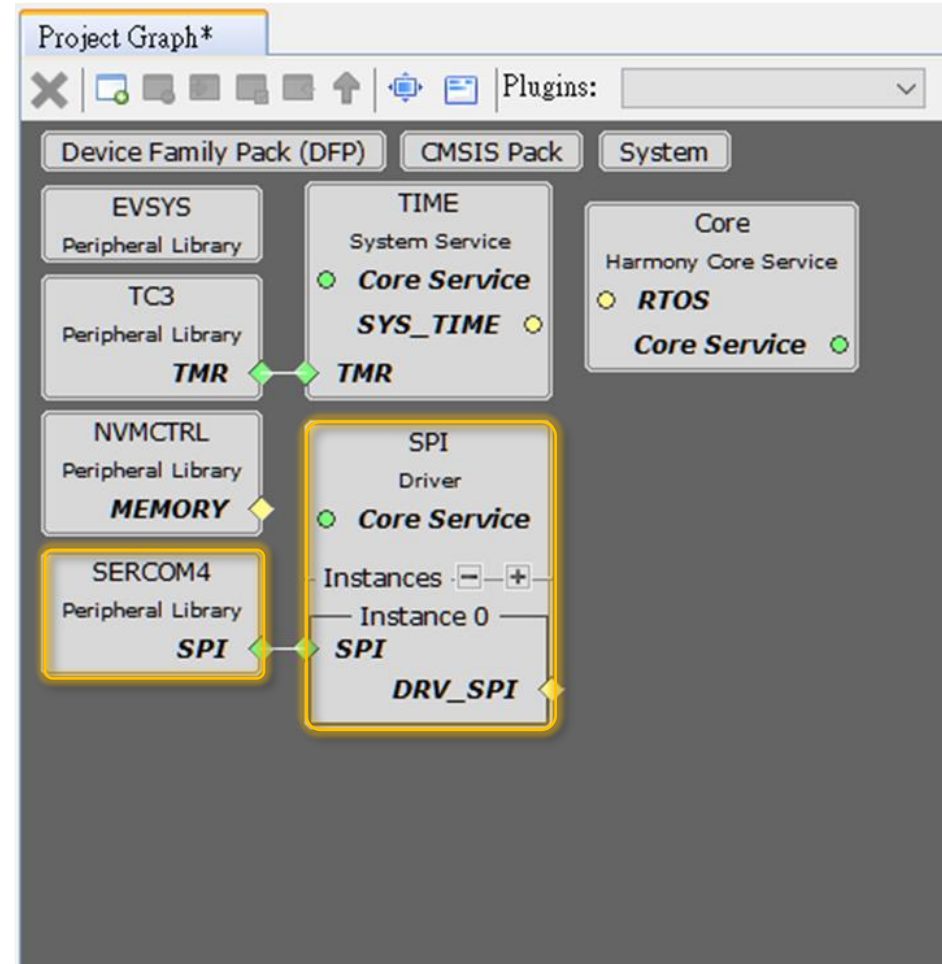
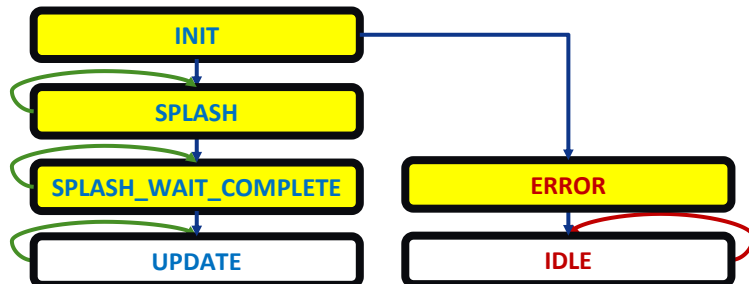
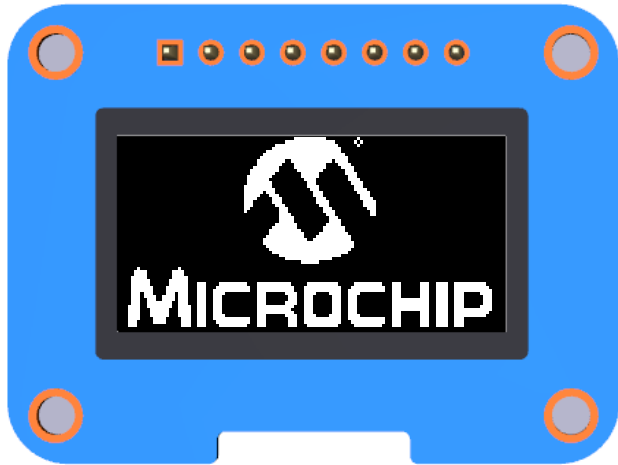
task.c



No I2C Auto Probe implemented in

⚙️ Lab16 Single-Client SPI OLED Display

- Compiler and programming your project to EVB.
- The OLED will show animated Microchip Logo



Lab16 Single-Client SPI OLED Display

- **Let's continue implement to display the text message.**

```

#include "app_drv_spi1.h"
#include "app_drv_usart1.h"
#include "driver/spi/drv_spi.h"
#include "system/time/sys_time.h"
#include "peripheral/port/plib_port.h"
#include "Alphabet_Fonts.h"
...

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

#define LCM_WIDTH 128
#define LCM_HEIGHT 64
#define FONT_WIDTH 7
#define FONT_HEIGHT 8
char OutString[20];
char LCM_Usart1Rx;
char LCM_Usart2Rx;
float LCM_I2c1Temp;
char LCM_I2c1EEPOM;
int LCM_I2c2Accel[3];
char Alphabet[95] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz~!@#$%^&()[]{}_-\\|;:.,`\"'<>+~*/=? ";
int LogoX=126;
const uint8_t LCM_Blank[1024];
const uint8_t LCM_InitCMD[] = {
    0xAE, 0xD5, 0x80, 0xA8, 0x3F, 0xD3, 0x00, 0x40, 0x8D, 0x14, 0x20, 0x02, 0xA1,
    0xC8, 0xDA, 0x12, 0x81, 0xCF, 0xD9, 0xF1, 0xDB, 0x40, 0xA4, 0xA6, 0xAF
};
...

```

✂ Lab16 Single-Client SPI OLED Display

- Implement LCM clear and draw text string functions.

```
// *****  
// Section: Application Local Functions  
// *****  
// *****  
void LCM_BusWrite( int bus, void *bytes, size_t counts ) { ... }  
void LCM_SetCursor( uint8_t x, uint8_t y ) { ... }  
void LCM_DrawBitmap( uint8_t sx, uint8_t sy, uint8_t width, uint8_t height, ...) { ... }  
  
void LCM_Clear( void )  
{  
    unsigned char LCM_Blank[1024];  
    memset(LCM_Blank, 0, 1024);  
    LCM_DrawBitmap(0, 0, LCM_WIDTH, LCM_HEIGHT, LCM_Blank);  
}  
  
void LCM_DrawFont( uint8_t sx, uint8_t row, char ch )  
{  
    int chIdx = 0;  
    while( ch != Alphabet[chIdx] ) { chIdx++; }  
    LCM_SetCursor(sx, row);  
    LCM_BusWrite( LCM_BUS_DAT, (void*)(Alphabet_Fonts+(chIdx*FONT_WIDTH)), FONT_WIDTH );  
}  
  
void LCM_DrawString( uint8_t sx, uint8_t row, char *str )  
{  
    for( int x=0 ; x<strlen(str); x++ )  
    {  
        LCM_DrawFont( sx+(x*FONT_WIDTH), row, str[x] );  
    }  
}  
  
...
```

app_drv_spi11.c

Insert after this

OLED screen clear

Find offset in font bitmap and output to target position

Draw string fonts

✂ Lab16 Single-Client SPI OLED Display

- Add OLED screen clear before start splash animation and end of animation.

```
void APP_DRV_SPI1_Tasks ( void )  
{  
    switch ( app_drv_spi1Data.state )  
    {  
        case APP_DRV_SPI1_STATE_INIT:  
            SSD1306_RESET_Set();  
  
            LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );  
            LCM_Clear();  
  
            app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;  
            break;  
  
        case APP_DRV_SPI1_STATE_SPLASH:  
            ...  
  
        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:  
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == true )  
            {  
                LCM_Clear();  
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;  
            }  
            break;  
  
        case APP_DRV_SPI1_STATE_UPDATE:  
            break;  
  
        case APP_DRV_SPI1_STATE_ERROR:  
            USART1printf("DRV_SPI1 error!\r\n");  
            ...  
    }  
}
```

app_drv_spi11.c

```
graph TD  
    INIT[INIT] --> SPLASH[SPLASH]  
    SPLASH --> SPLASH_WAIT_COMPLETE[SPLASH_WAIT_COMPLETE]  
    SPLASH_WAIT_COMPLETE --> UPDATE[UPDATE]  
    UPDATE --> IDLE[IDLE]  
    SPLASH --> SPLASH  
    SPLASH_WAIT_COMPLETE --> SPLASH_WAIT_COMPLETE  
    ERROR[ERROR] --> IDLE
```

✖ Lab16 Single-Client SPI OLED Display

- Output text message to OLED in state **UPDATE**.

```
void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
        case APP_DRV_SPI1_STATE_INIT: ...
        case APP_DRV_SPI1_STATE_SPLASH: ...
        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE: ...

        case APP_DRV_SPI1_STATE_UPDATE:
            sprintf( OutString, "LCM: Hello World" );
            LCM_DrawString(0, 0, OutString);
            sprintf( OutString, "USART1 RX: %c", LCM_Usart1Rx );
            LCM_DrawString(0, 1, OutString);
            sprintf( OutString, "USART2 RX: %c", LCM_Usart2Rx );
            LCM_DrawString(0, 2, OutString);
            sprintf( OutString, "I2C1 Temp: %2.1f'C", LCM_I2c1Temp );
            LCM_DrawString(0, 3, OutString);
            sprintf( OutString, "I2C1 EEPROM: 0x%02X", LCM_I2c1EEPROM );
            LCM_DrawString(0, 4, OutString);
            sprintf( OutString, "I2C2 AccelX: %5d", LCM_I2c2Accel[0] );
            LCM_DrawString(0, 5, OutString);
            sprintf( OutString, "I2C2 AccelY: %5d", LCM_I2c2Accel[1] );
            LCM_DrawString(0, 6, OutString);
            sprintf( OutString, "I2C2 AccelZ: %5d", LCM_I2c2Accel[2] );
            LCM_DrawString(0, 7, OutString);
            Keep this line break;

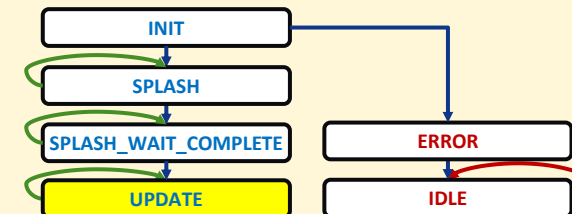
        case APP_DRV_SPI1_STATE_ERROR:
            USART1printf("DRV_SPI1 error!\r\n");

            app_drv_spi1Data.state = APP_DRV_I2C1_STATE_IDLE;
            break;
        ...
    }
}
```

app_drv_spi11.c

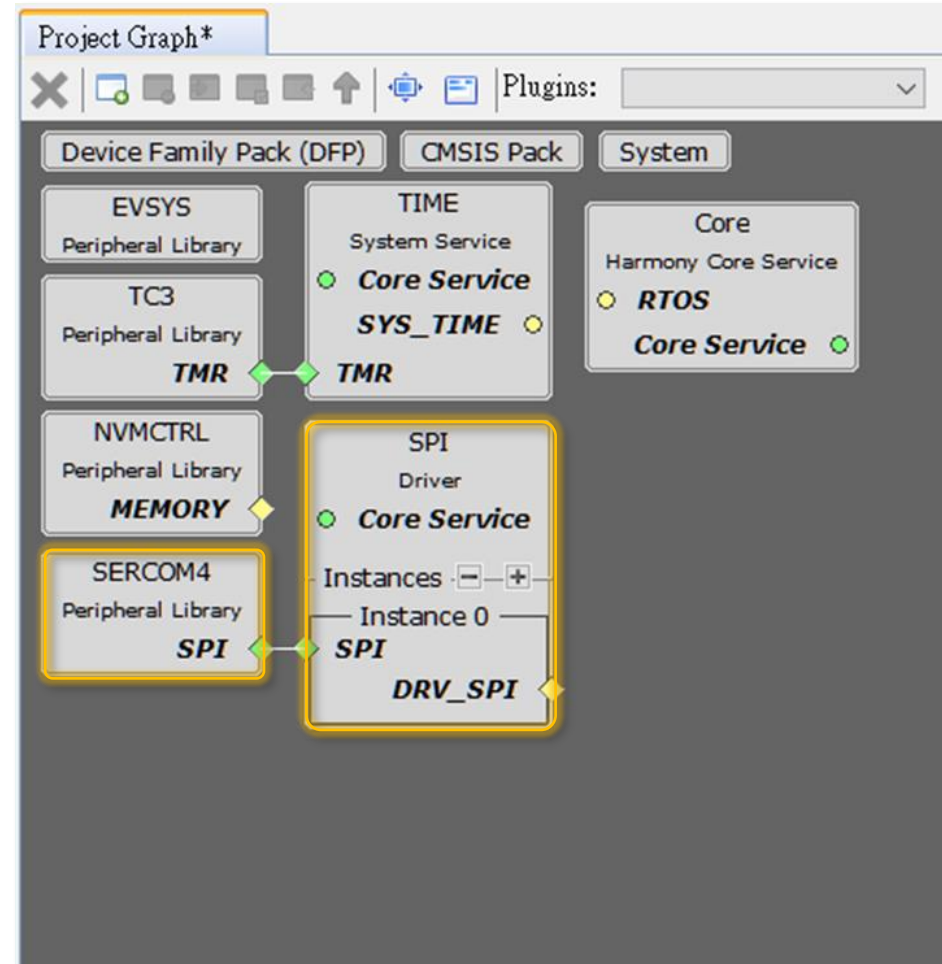
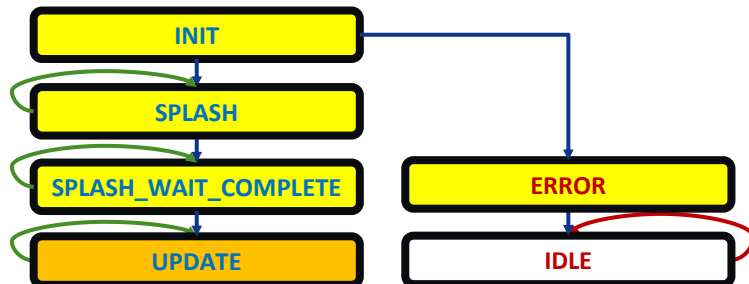


LCM: Hello World
USART1 RX:
USART2 RX:
I2C1 Temp: 0.0'C
I2C1 EEPROM: 0x00
I2C2 AccelX: 0
I2C2 AccelY: 0
I2C2 AccelZ: 0



✖ Lab16 Single-Client SPI OLED Display

- Compiler and programming your project to EVB.
- The OLED will output text message after splash animation.



✂ Lab16 Single-Client SPI OLED Display

- Extern variables in **app_drv_spi1.h** for collect data from USART and I2C Drivers.

```
#ifndef _APP_DRV_SPI1_H
#define _APP_DRV_SPI1_H

typedef enum
{
    /* Application's state machine's initial state. */
    APP_DRV_SPI1_STATE_INIT=0,
    // APP_DRV_SPI1_STATE_SERVICE_TASKS,
    APP_DRV_SPI1_STATE_SPLASH,
    APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE,
    APP_DRV_SPI1_STATE_UPDATE,
    APP_DRV_SPI1_STATE_ERROR,
    APP_DRV_SPI1_STATE_IDLE,
} APP_DRV_SPI1_STATES;
...
```

```
void APP_DRV_SPI1_Initialize ( void );
void APP_DRV_SPI1_Tasks( void );
```

```
extern char LCM_Usart1Rx;
extern char LCM_Usart2Rx;
extern float LCM_I2c1Temp;
extern char LCM_I2c1EERPOM;
extern int LCM_I2c2Accel[3];
```

```
//DOM-IGNORE-BEGIN
#ifdef __cplusplus
}
#endif
//DOM-IGNORE-END
```

```
#endif /* _APP_DRV_SPI1_H */
```

app_drv_spi1.h

Extern variables for collect data

✂ Lab16 Single-Client SPI OLED Display

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

```
#include "app_drv_usart1.h"  
#include "app_drv_usart2.h"  
#include "app_drv_spi1.h"  
#include "driver/usart/drv_usart.h"  
#include <stdio.h>  
#include <string.h>  
#include <stdarg.h>
```

Include app_drv_spi1.h

app_drv_usart1.c

...

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

```
    case APP_DRV_USART1_STATE_READ_WAIT_COMPLETE:  
        if(Uart1ReadComplete == true)  
        {
```

Insert between this

```
            Uart1ReadComplete = false;
```

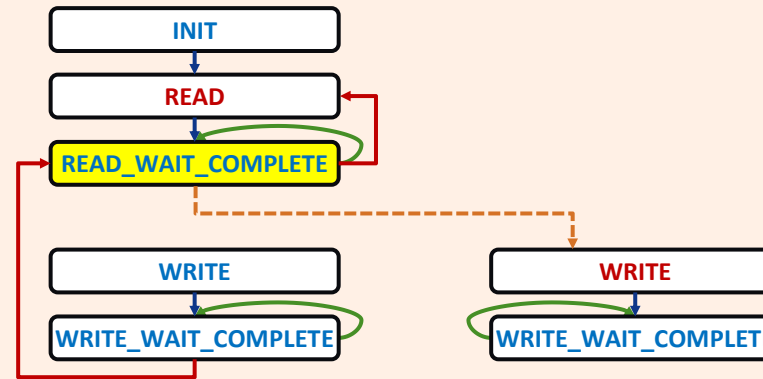
```
            LCM_Usart1Rx = Uart1ReadBuf[0];
```

Insert between this

```
            app_drv_usart1Data.state = APP_DRV_USART1_STATE_READ;  
            app_drv_usart2Data.state = APP_DRV_USART2_STATE_WRITE;
```

```
        }  
        break;
```

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



Collect Uart1ReadBuf[0]

⚙️ Lab16 Single-Client SPI OLED Display

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

```
#include "app_drv_usart2.h"
#include "app_drv_usart1.h"
#include "app_drv_spi1.h"
#include "driver/usart/drv_usart.h"
#include <stdio.h>
#include <string.h>
#include <stdarg.h>

...

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;
                LCM_Usart2Rx = Uart2ReadBuf[0];
                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:
            ...
    }
}
```

Include app_drv_spi1.h

app_drv_usart2.c

Collect Uart2ReadBuf[0]

Insert between this

Insert between this

✖ Lab16 Single-Client SPI OLED Display

- Modify state **WAIT_COMPLETE_9800** and **WAIT_COMPLETE_24AA** in “app_drv_i2c1.c”.

```
#include "app_drv_i2c1.h"
#include "app_drv_usart1.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            if(I2C1Complete_9800 == true)
            {
                I2C1Complete_9800 = false;

                LCM_I2c1Temp = (float)((I2C1ReadBuf_9800[0]<<4)|
                (I2C1ReadBuf_9800[1]>>4))/16.0f;
                USART1printf("Temp = %2.1f\r",
                (float)((I2C1ReadBuf_9800[0]<<4)|(I2C1ReadBuf_9800[1]>>4))/16.0f);

                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_24AA;
            }
            ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA:
            if(I2C1Complete_24AA == true)
            {
                I2C1Complete_24AA = false;

                LCM_I2c1EEPROM = I2C1ReadBuf_24AA[0];
                USART1printf("\t\tEEPROM = 0x%X\r", I2C1ReadBuf_24AA[0]);

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

app_drv_i2c1.c

Include app_drv_spi1.h

```
graph TD
    INIT[INIT] --> WR9800[WRITE_READ_9800]
    WR9800 --> WC9800[WAIT_COMPLETE_9800]
    WC9800 --> WR24AA[WRITE_READ_24AA]
    WR24AA --> WC24AA[WAIT_COMPLETE_24AA]
    WC24AA --> ERROR[ERROR]
    ERROR --> IDLE[IDLE]
    IDLE --> INIT
    WC9800 --> WC9800
    WC24AA --> WC24AA
```

Insert between this →

↑ **Insert between this**

Insert between this →

↑ **Insert between this**

Collect Temperature result

Collect EEPROM read byte

✖ Lab16 Single-Client SPI OLED Display

- Modify state **WAIT_COMPLETE_AXIS_TJ3** in “app_drv_i2c2.c”.

```
#include "app_drv_i2c2.h"
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        ...
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];
                if(++ AxisOutRegIdx >= 6)
                {
                    LCM_I2c2Accel[0] = (short)(AxisOutByte[1]<<8|AxisOutByte[0]);
                    LCM_I2c2Accel[1] = (short)(AxisOutByte[3]<<8|AxisOutByte[2]);
                    LCM_I2c2Accel[2] = (short)(AxisOutByte[5]<<8|AxisOutByte[4]);

                    USART1printf("\t\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",
                                (short)(AxisOutByte[1]<<8|AxisOutByte[0]),
                                (short)(AxisOutByte[3]<<8|AxisOutByte[2]),
                                (short)(AxisOutByte[5]<<8|AxisOutByte[4]));

                    ...

                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_WRITE_READ_DRDY_TJ3;
                    break;
                }
            }
            ...
    }
}
```

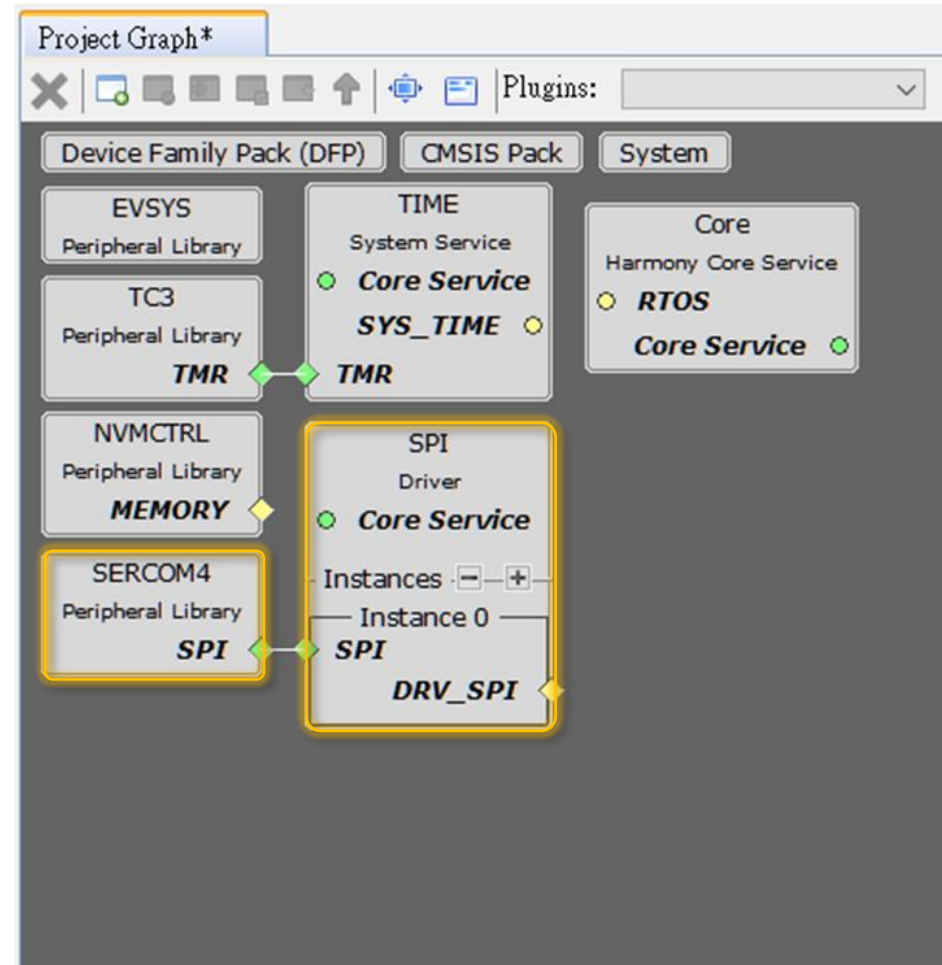
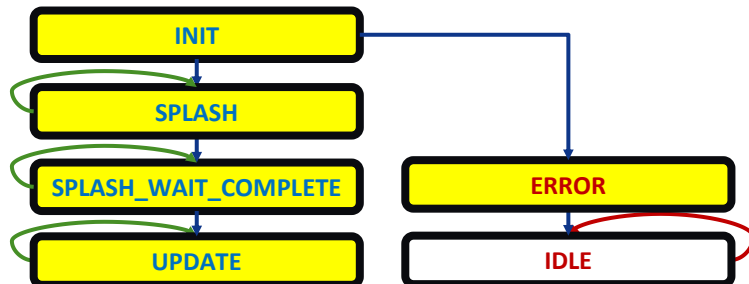
app_drv_i2c2.c

Include app_drv_spi1.h

Collect G-Sensor Gravity data X, Y, Z

⚙️ Lab16 Single-Client SPI OLED Display

- Compiler and programming your project to EVB.
- The OLED will output the data that we collect from Driver.



✖ Lab16 Single-Client SPI OLED Display

- Try comment out UART debug message in “app_drv_i2c1.c” to speed-up OLED display.

```
#include "app_drv_i2c1.h"
#include "app_drv_usart1.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C1_Tasks ( void )
{
    switch ( app_drv_i2c1Data.state )
    {
        ...
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_9800:
            if(I2C1Complete_9800 == true)
            {
                I2C1Complete_9800 = false;

                LCM_I2c1Temp = (float)((I2C1ReadBuf_9800[0]<<4)|
                                         (I2C1ReadBuf_9800[1]>>4))/16.0f;
                // USART1printf("Temp = %2.1f\r",
                //                 (float)((I2C1ReadBuf_9800[0]<<4)|(I2C1ReadBuf_9800[1]>>4))/16.0f);
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_24AA;
            }
        case APP_DRV_I2C1_STATE_WAIT_COMPLETE_24AA:
            if(I2C1Complete_24AA == true)
            {
                I2C1Complete_24AA = false;

                LCM_I2c1EEPROM = I2C1ReadBuf_24AA[0];
                // USART1printf("\t\tEEPROM = 0x%X\r", I2C1ReadBuf_24AA[0]);
                app_drv_i2c1Data.state = APP_DRV_I2C1_STATE_WRITE_READ_9800;
            }
        ...
    }
}
```

app_drv_i2c1.c

Comment out USART1printf

Comment out

Comment out

⚙️ Lab16 Single-Client SPI OLED Display

- Try comment out UART debug message in “app_drv_i2c2.c” to speed-up OLED display.

```
#include "app_drv_i2c2.h"
#include "app_drv_usart1.h"
#include "app_drv_usart2.h"
#include "app_drv_spi1.h"
#include "driver/i2c/drv_i2c.h"

...
void APP_DRV_I2C2_Tasks ( void )
{
    switch ( app_drv_i2c2Data.state )
    {
        ...
        case APP_DRV_I2C2_STATE_WAIT_COMPLETE_AXIS_TJ3:
            if(I2C2Complete_TJ3 == true)
            {
                I2C2Complete_TJ3 = false;

                AxisOutByte[AxisOutRegIdx] = I2C2ReadBuf_TJ3[0];
                if(++ AxisOutRegIdx >= 6)
                {
                    LCM_I2c2Accel[0] = (short)(AxisOutByte[1]<<8 | AxisOutByte[0]);
                    LCM_I2c2Accel[1] = (short)(AxisOutByte[3]<<8 | AxisOutByte[2]);
                    LCM_I2c2Accel[2] = (short)(AxisOutByte[5]<<8 | AxisOutByte[4]);
                    // USART1printf("\t\t\t\tG-Sensor = (%5d, %5d, %5d)\r",
                    //                (short)(AxisOutByte[1]<<8 | AxisOutByte[0]),
                    //                (short)(AxisOutByte[3]<<8 | AxisOutByte[2]),
                    //                (short)(AxisOutByte[5]<<8 | AxisOutByte[4]));

                    // unsigned char DVBuf[8] = { 0x03,
                    //                               AxisOutByte[0], AxisOutByte[1],
                    //                               AxisOutByte[2], AxisOutByte[3],
                    //                               AxisOutByte[4], AxisOutByte[5],
                    //                               0x00, 0x00 };
                    // USART2write( DVBuf, 8 );
                }
            }
            ...
    }
}
```

**Comment out
USART1printf and USART2write**

Comment out

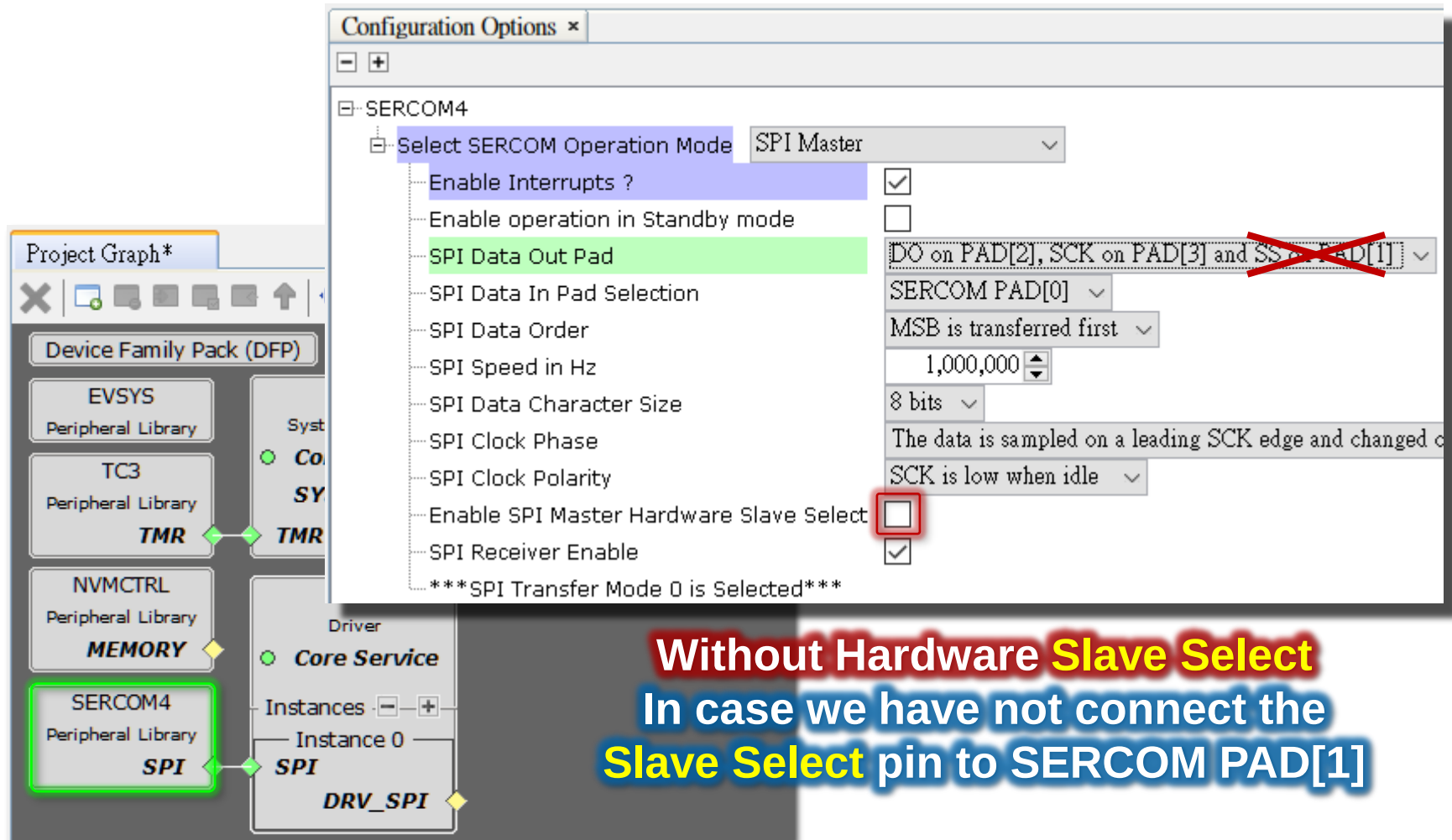
Comment out

**If you have implemented
Advanced Exercise : MPLAB Data Visualizer**

⚙️ Lab16 Single-Client SPI OLED Display

Discuss 1

- Try use **GPIO** for **SPI Driver** controlled **Software Chip Select** pin of OLED.



Configuration Options

SERCOM4

Select SERCOM Operation Mode: SPI Master

Enable Interrupts ? ☒

Enable operation in Standby mode ☐

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

SPI Data In Pad Selection: SERCOM PAD[0]

SPI Data Order: MSB is transferred first

SPI Speed in Hz: 1,000,000

SPI Data Character Size: 8 bits

SPI Clock Phase: The data is sampled on a leading SCK edge and changed on a trailing SCK edge

SPI Clock Polarity: SCK is low when idle

Enable SPI Master Hardware Slave Select ☐

SPI Receiver Enable ☒

SPI Transfer Mode 0 is Selected

Project Graph*

Device Family Pack (DFP)

EVSY

Peripheral Library

TC3

Peripheral Library

TMR

NVMCTRL

Peripheral Library

MEMORY

SERCOM4

Peripheral Library

SPI

Instances

Instance 0

DRV_SPI

Without Hardware Slave Select
In case we have not connect the
Slave Select pin to SERCOM PAD[1]

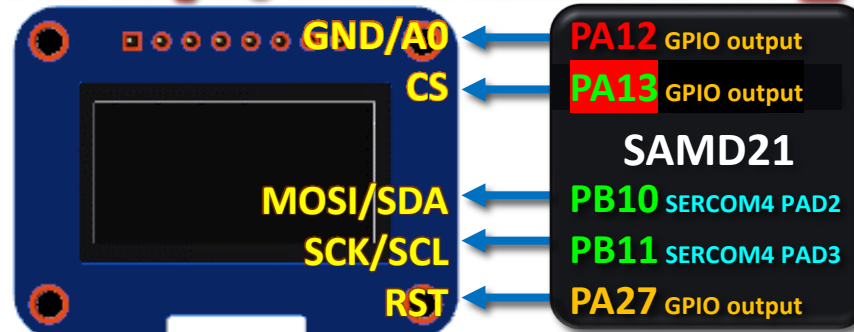
✖ Lab16 Single-Client SPI OLED Display

Discuss 1

- Change **PA13** to **GPIO output**, Latch **High** and Custom name as **SSD1306_CS**.

Project Graph* Pin Diagram * Pin Table * Pin Settings *									
Order: Pins Table View ☒ Easy View									
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
19	PB10		SERCOM4_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
20	PB11		SERCOM4_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL
21	PA12	SSD1306_RS	GPIO	Digital	Out	High	☐	☐	NORMAL
22	PA13	SSD1306_CS	GPIO	Digital	Out	High	☐	☐	NORMAL
39	PA27	SSD1306_RESET	GPIO	Digital	Out	High	☐	☐	NORMAL

PA12 as GPIO output, custom name SSD1306_RS , Latch to High
PA27 as GPIO output, custom name SSD1306_RESET , Latch to High
Configure PB10 / PB11 as SERCOM4_PAD2 / PAD3
PA13 as GPIO output, custom name SSD1306_CS , Latch to High



🔧 Lab16 Single-Client SPI OLED Display

Discuss 1

- Do **SPI Transfer setup** could assign Software **Chip Select GPIO Pin** in state **INIT**

```
void APP_DRV_SPI1_Tasks ( void )  
{  
    switch ( app_drv_spi1Data.state )  
    {  
        case APP_DRV_SPI1_STATE_INIT:  
            SSD1306_RESET_Clear();  
  
            SPI1Handle = DRV_SPI_Open(DRV_SPI_INDEX_0, DRV_IO_INTENT_WRITE);  
            if(SPI1Handle == DRV_HANDLE_INVALID)  
            {  
                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_ERROR;  
                break;  
            }  
  
            DRV_SPI_TRANSFER_SETUP SPI1Setup;  
            SPI1Setup.baudRateInHz = 1000000;  
            SPI1Setup.clockPhase = DRV_SPI_CLOCK_PHASE_VALID_LEADING_EDGE;  
            SPI1Setup.clockPolarity = DRV_SPI_CLOCK_POLARITY_IDLE_LOW;  
            SPI1Setup.dataBits = DRV_SPI_DATA_BITS_8;  
            SPI1Setup.chipSelect = (SYS_PORT_PIN)SSD1306_CS_PIN;  
            SPI1Setup.csPolarity = DRV_SPI_CS_POLARITY_ACTIVE_LOW;  
            DRV_SPI_TransferSetup(SPI1Handle, &SPI1Setup);  
  
            SSD1306_RESET_Set();  
  
            LCM_BusWrite( LCM_BUS_CMD, (void*)LCM_InitCMD, sizeof(LCM_InitCMD) );  
  
            app_drv_spi1Data.state = APP_DRV_SPI1_STATE_SPLASH;  
            break;  
  
        case APP_DRV_SPI1_STATE_SPLASH: ...
```

app_drv_spi11.c

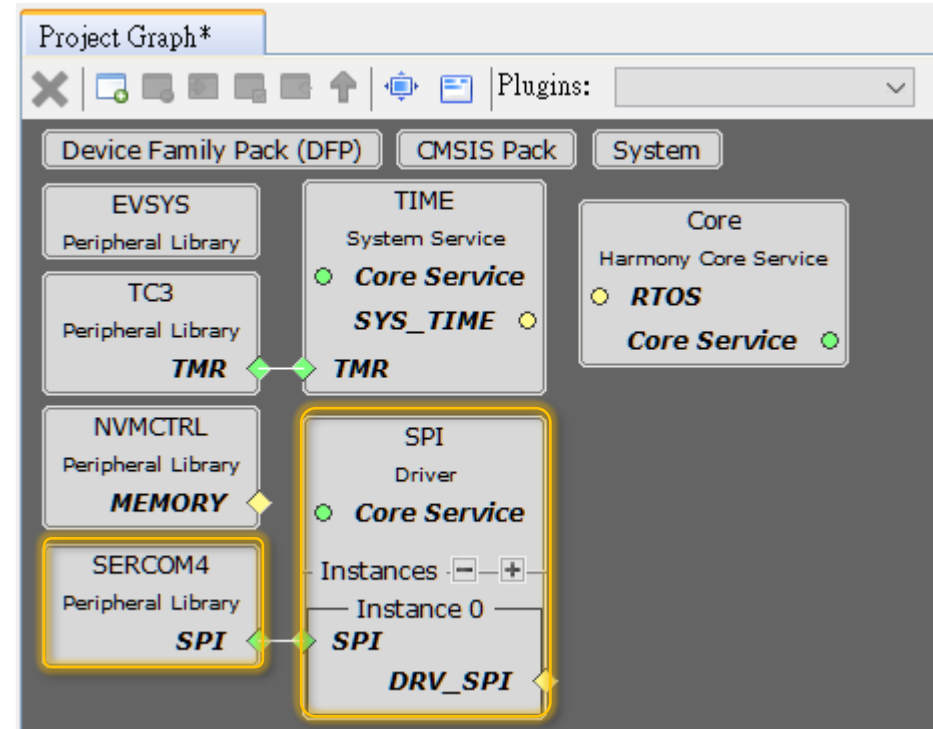
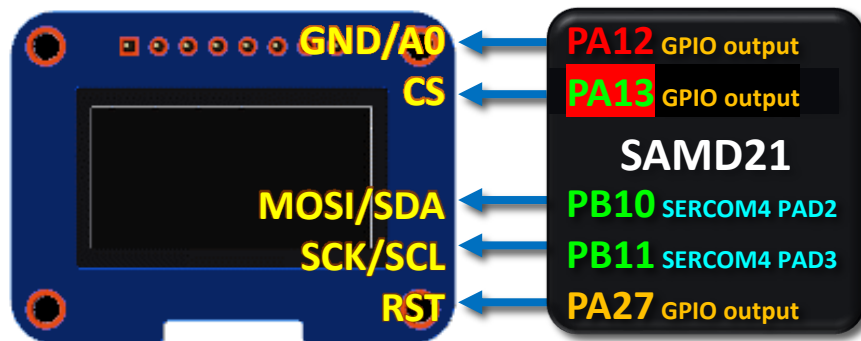
**Assign SSD1306_CS (PA13) in
SPI Transfer Setup for Software Chip Select**

Modify this Line

⚙️ Lab16 Single-Client SPI OLED Display

Discuss 1

- Compiler and programming your project to EVB.
- It's work fine even we use a **GPIO** for **SPI Driver** controlled **Software Chip Select**.



SPI Driver – Wi-Fi WINC Driver Library

Harmony Driver

SPI Driver – Wi-Fi WINC Driver Library

• Introduction

The MPLAB Harmony WINC Driver provides a high-level interface to the ATWINC1500 and ATWINC3400 Wi-Fi network controller modules. The WINC Driver Library, in conjunction with the ATWINC1500 or ATWINC3400 module, allows an application to:

- Join an existing 802.11 b/g/n 2.4GHz Wi-Fi Infrastructure network.
- Create an 802.11 Wi-Fi Ad Hoc or Soft AP network.
- Scan for other W-Fi devices in an area
- Supports BLE 4.0 provisioning and profiles in central and peripheral mode (ATWINC3400 only)

• Abstraction Model

This library provides an abstraction of the ATWINC1500 and ATWINC3400 network controllers. The library operates in two forms, a basic driver intended to work solely with an application and a MAC driver intended to work in concert with the Harmony TCP/IP stack and the application. In either form, the driver provides function to perform the following:

- Manage WINC system time
- Manage Wi-Fi associations
- Perform Wi-Fi BSS search operations
- Create and manage Soft AP functionality
- Connect to an existing 802.11 b/g/n infrastructure network
- Control device power save operation
- Manage security profiles
- Configure and control WINC network controllers

SPI Driver – Wi-Fi WINC Driver Library

- **ASIC Socket vs Ethernet Mode**

The WINC modules are capable of below operating modes:

Socket mode : (Basic Driver)

This mode allows an application running on the host MCU to utilize the **TCP/IP stack within the firmware of the WINC device**, as such the **host MCU does not need to run its own TCP/IP stack**.

Ethernet (bypass) mode : (MAC Driver)

This mode to disables the TCP/IP stack within the firmware of the WINC device, instead the host MCU can only send and receive Ethernet frames. This mode allows the **host MCU to operate its own TCP/IP stack** treating the WINC device as a simple Ethernet controller.

Note: The WINC device may need separate firmware loaded to support a different mode.

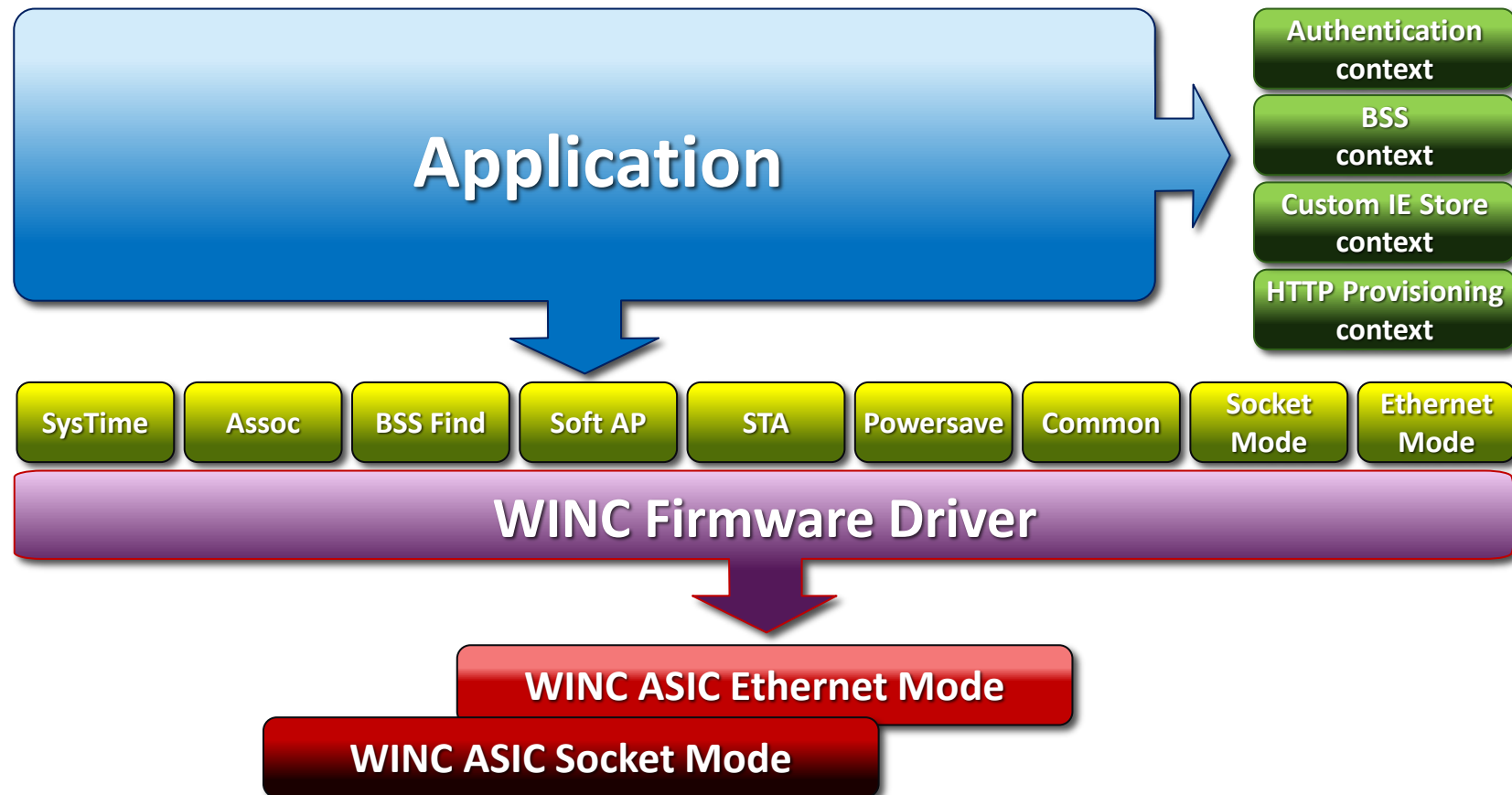
- **Contexts**

The driver uses several contexts to manage complex groupings of information such as authentication, BSS and HTTP provisioning settings. A context can be created external to the WINC Driver before being passed into the driver as part of an operation. The WINC Driver may also pass back to the application a context if it is appropriate for the information being passed. The WINC Driver provides functions to manipulate the contexts and ensure they are valid before being accepted by the driver.

SPI Driver – Wi-Fi WINC Driver Library

- **Basic WINC Driver (Socket mode)**

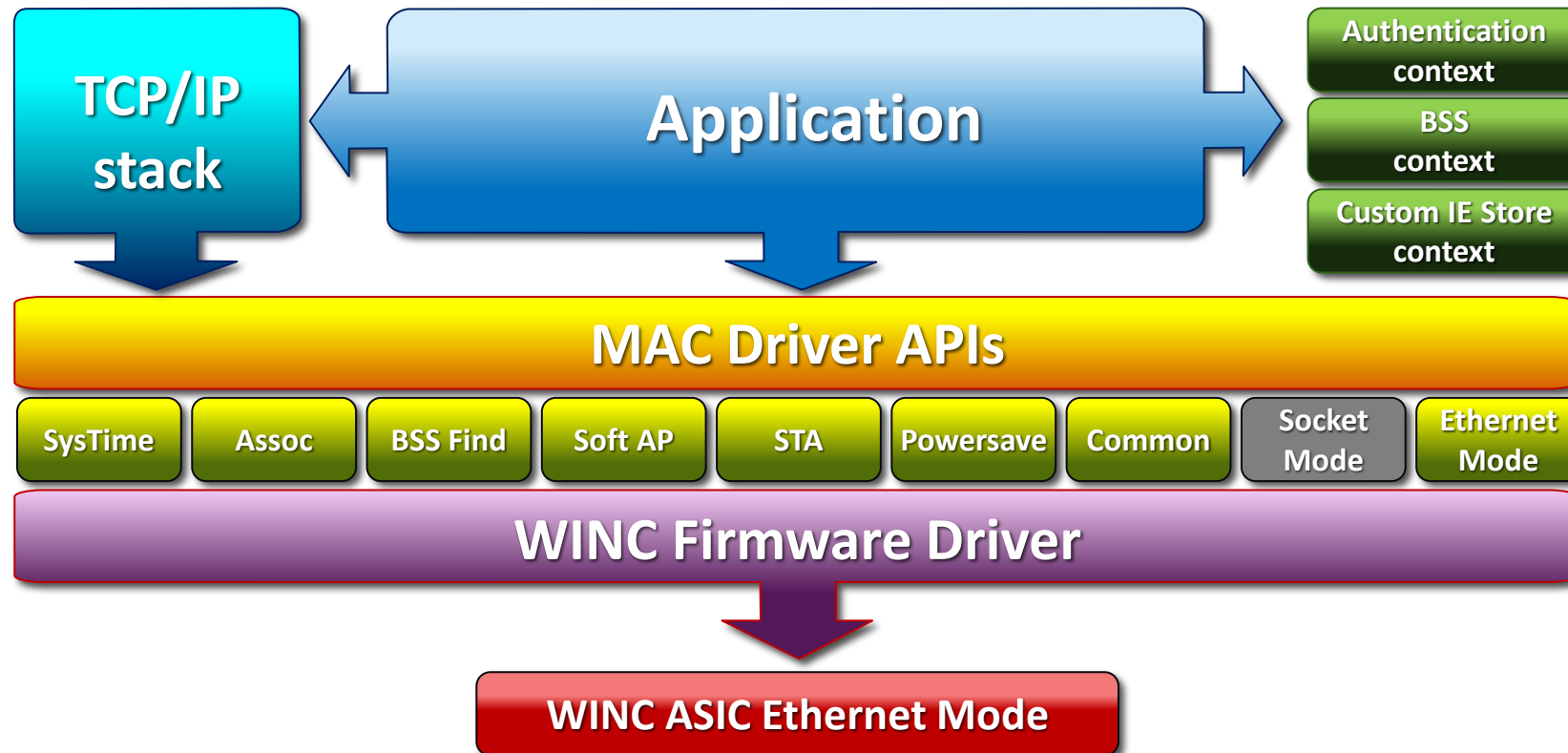
The Basic WINC Driver is utilized when the host MCU application is **the primary client** of the WINC Driver. In this arrangement the host MCU application is either reliant on the TCP/IP stack within the WINC device firmware or is only interested in sending and receiving basic Ethernet frames.



SPI Driver – Wi-Fi WINC Driver Library

- **MAC WINC Driver (Ethernet/bypass mode)**

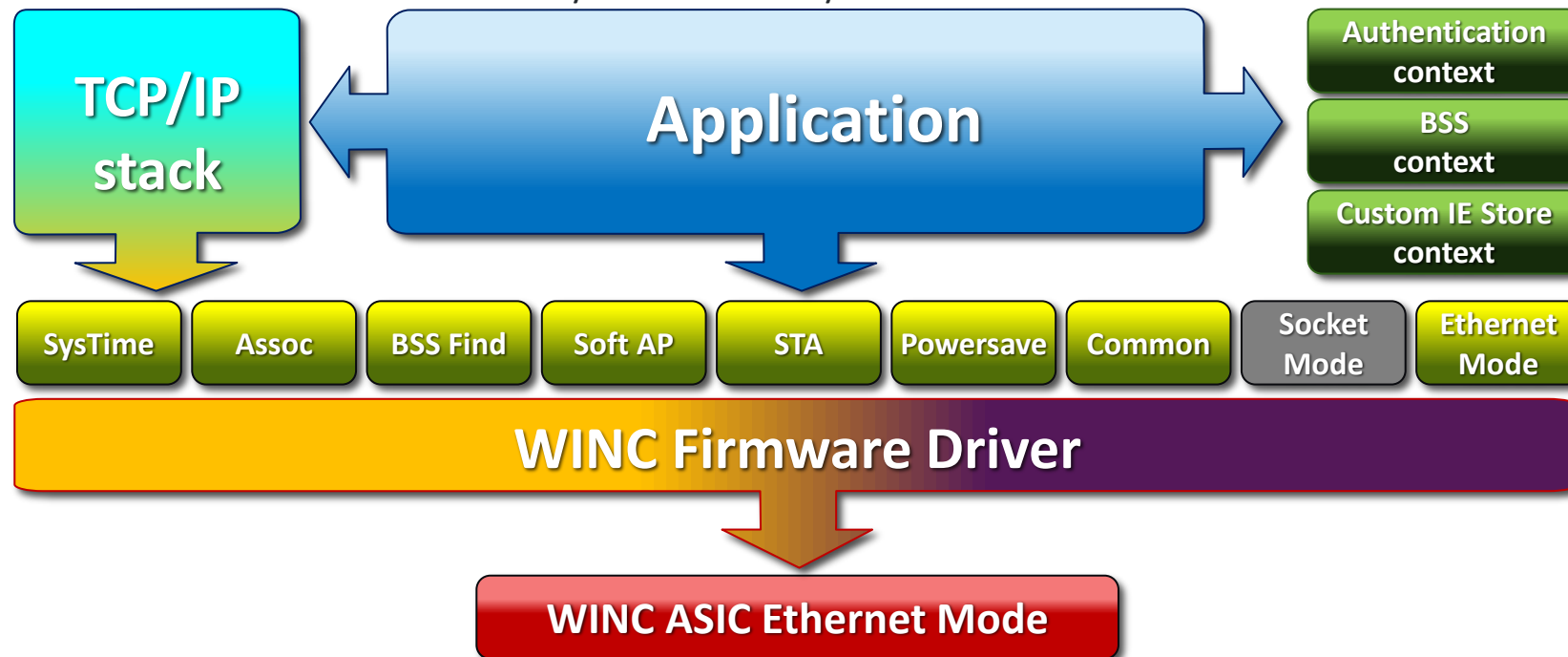
The MAC WINC Driver is utilized when the host MCU application is the **combined with the Harmony TCP/IP stack**. The Harmony TCP/IP stack is responsible for operating the WINC device as an **Ethernet controller** while the application is responsible for managing the Wi-Fi specific functionality such as locating and connecting to 802.11 b/g/n BSSs.



SPI Driver – Wi-Fi WINC Driver Library

- **MAC WINC Driver (Ethernet/bypass mode)** (Changes in WINC1500 19.7.x and WINC3400 1.4.2)

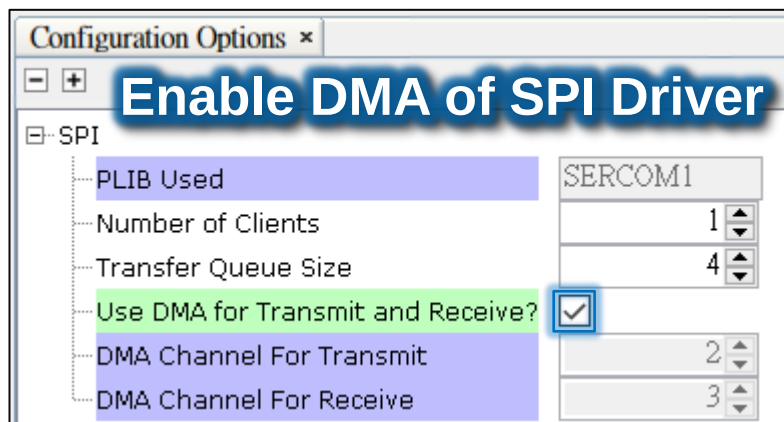
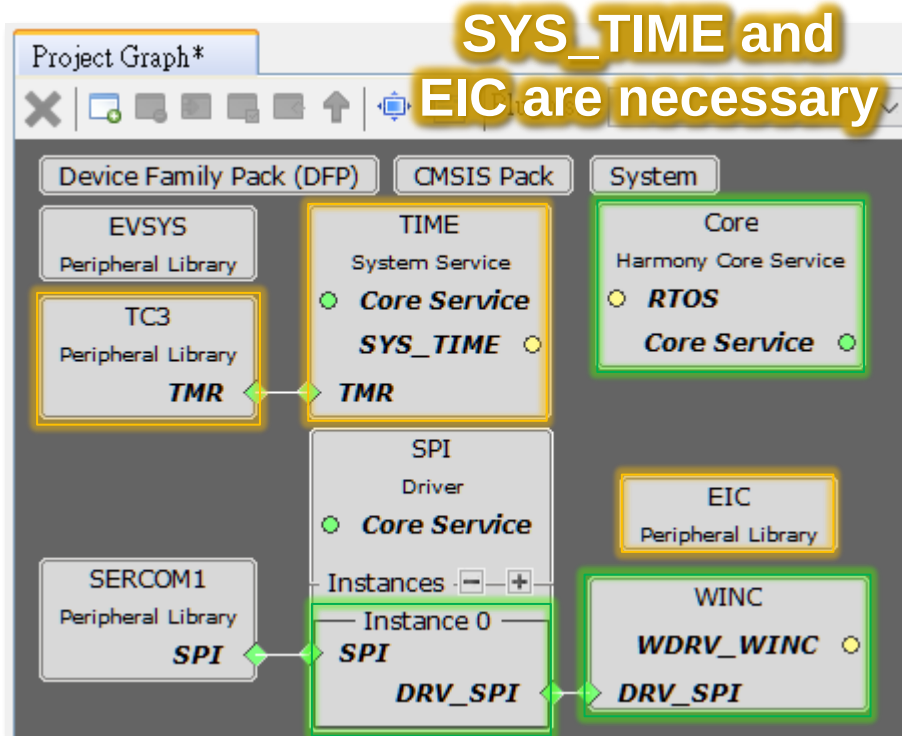
The major change in the later WINC architecture is the **removal of the 'stack driver'**. Previously there was an extra layer over the WINC driver API when the driver was used in bypass mode with the Harmony TCP/IP stack. The application had limited control of the driver through the WDRV_WINC_MAC* APIs. The APIs had to be wrapped in a WDRV_WINC_MAC version and there was no direct control of connection/disconnection. The driver now has two interfaces which can be opened, first the normal interface used in non-bypass mode (called the control interface) and a second interface used by the TCP/IP stack (the MAC interface). In a way, instead of being the MAC interface on top of the control interface as before they are now side by side.



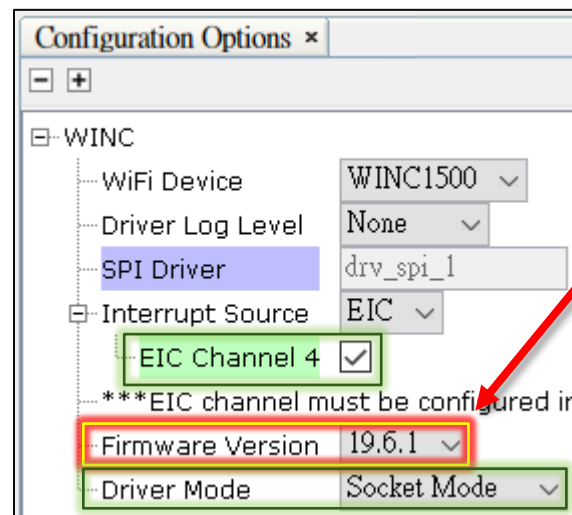
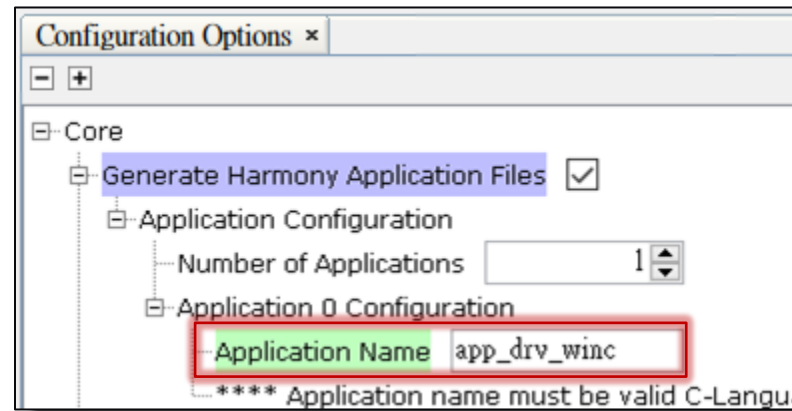
For the applications there is no change to how things work for non-bypass mode. However, now that same application control can be used in bypass mode with the TCP/IP stack, **the only difference being that the WINC socket API is replaced with the Harmony stack API** (and certain WINC APIs become unavailable as they are based on non-bypass firmware).

The example `wifi_winc_sta_bypass` has been updated to reflect this change.

SPI Driver – Wi-Fi WINC Driver Library



Create an Application for Driver State Machine



*** Please must select the correct version of your WIN1500 Module Firmware**

**Assign External Interrupt
WIN1500 Module Firmware
WINC Driver Mode**

SPI Driver – Wi-Fi WINC Driver Library

API functions of the Wi-Fi WINC Driver Library.

- **WINC wireless driver System Interface implementation (wdrv_winc.c)**

WDRV_WINC_Initialize	System interface initialization of the WINC driver.
WDRV_WINC_Deinitialize	WINC driver deinitialization function.
WDRV_WINC_Reinitialize	WINC driver reinitialization function.
WDRV_WINC_Status	Provides the current status of the WINC driver module.
WDRV_WINC_Tasks	Maintains the WINC drivers state machine.
WDRV_WINC_ISR	WINC interrupt handler.
WDRV_WINC_DebugRegisterCallback	Register callback for debug serial stream.
WDRV_WINC_Open	Opens an instance of the WINC driver.
WDRV_WINC_Close	Closes an instance of the WINC driver.
WDRV_WINC_MulticastMACFilterAdd	Adds an Ethernet address to the multicast filter.
WDRV_WINC_MulticastMACFilterRemove	Removes an Ethernet address from the multicast filter.
WDRV_WINC_EthernetAddressGet	Returns the current Ethernet address assigned to the WINC.
WDRV_WINC_OperatingModeGet	Returns the current operating mode of the WINC.
WDRV_WINC_UserHandleSet	Associates a caller supplied handle with the driver instance.
WDRV_WINC_UserHandleGet	Returns the current user supplied handle associated with the driver.
WDRV_WINC_InfoDriverVersionGet	Returns the drivers version information.
WDRV_WINC_InfoDeviceFirmwareVersionGet	Returns the WINC firmware version information.
WDRV_WINC_InfoDeviceIDGet	Returns the device ID.
WDRV_WINC_InfoDeviceMACAddressGet	Retrieves the MAC address of the WINC.
WDRV_WINC_InfoDeviceNameSet	Sets the WiFi Direct Device Name.
WDRV_WINC_GainTableIndexSet	Sets the gain table index.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver Socket Mode implementation (wdrv_winc_socket.c)**

WDRV_WINC_IPAddressSet	Configures an IPv4 address.
WDRV_WINC_IPAddressGet	Returns the current IPv4 address.
WDRV_WINC_IPDNSServerAddressSet	Configures the DNS server address.
WDRV_WINC_IPDefaultGatewaySet	Configures the default gateway used by the WINC.
WDRV_WINC_IPUseDHCPSet	Enables DHCP.
WDRV_WINC_IPUseDHCPGet	Determine if DHCP is to be used.
WDRV_WINC_IPDHCPServerConfigure	Configures the Soft-AP DHCP server.
WDRV_WINC_IPLinkActive	Indicates state of IP link.
WDRV_WINC_ICMPEchoRequest	Sends an ICMP echo request.
WDRV_WINC_OTAUUpdateFromURL	Perform OTA from URL provided.
WDRV_WINC_OTAUUpdateAbort	Abort an OTA update.
WDRV_WINC_SwitchActiveFirmwareImage	Switch active firmware image to inactive partition.
WDRV_WINC_SocketRegisterEventCallback	Register an event callback for socket events.
WDRV_WINC_SocketRegisterResolverCallback	Register an event callback for DNS resolver events.
WDRV_WINC_SwitchActiveFirmwareImage	Switch active firmware image to inactive partition.
WDRV_WINC_SocketRegisterEventCallback	Register an event callback for socket events.
WDRV_WINC_SocketRegisterResolverCallback	Register an event callback for DNS resolver events.

- **WINC wireless driver STA implementation (wdrv_winc_sta.c)**

WDRV_WINC_BSSConnect	Connects to a BSS in infrastructure station mode.
WDRV_WINC_BSSReconnect	Reconnects to a BSS using stored credentials.
WDRV_WINC_BSSDisconnect	Disconnects from a BSS.
WDRV_WINC_BSSRoamingConfigure	Configures BSS roaming support.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver BSS Scanning implementation (wdrv_winc_bssfind.c)**

WDRV_WINC_BSSFindFirst	Request a BSS scan is performed by the WINC.
WDRV_WINC_BSSFindNext	Request the next scan results be provided.
WDRV_WINC_BSSFindReset	Requests the information structure of the current BSS scan result.
WDRV_WINC_BSSFindGetInfo	Adds an Ethernet address to the multicast filter.
WDRV_WINC_BSSFindSetScanParameters	Configures the scan operation.
WDRV_WINC_BSSFindSetRSSIThreshold	Configures the scan RSSI threshold.
WDRV_WINC_BSSFindSetEnabledChannels	Configures the channels which can be scanned.
WDRV_WINC_BSSFindGetNumBSSResults	Returns the number of BSS scan results found.
WDRV_WINC_BSSFindInProgress	Indicates if a BSS scan is in progress.

- **WINC wireless driver BSS Context implementation (wdrv_winc_bssctx.c)**

WDRV_WINC_BSSCtxIsValid	Tests if a BSS context is valid.
WDRV_WINC_BSSCtxSetDefaults	Configures a BSS context into a default legal state.
WDRV_WINC_BSSCtxSetSSID	Configures the SSID of the BSS context.
WDRV_WINC_BSSCtxSetBSSID	Configures the BSSID of the BSS context.
WDRV_WINC_BSSCtxSetChannel	Configures the channel of the BSS context.
WDRV_WINC_BSSCtxSetSSIDVisibility	Configures the visibility of the BSS context.

- **WINC wireless driver Association implementation (wdrv_winc_assoc.c)**

WDRV_WINC_AssociationSSIDGet	Retrieve the current association SSID.
WDRV_WINC_AssociationPeerAddressGet	Retrieve the current association peer device network address.
WDRV_WINC_AssociationAuthTypeGet	Retrieve the current association authentication type.
WDRV_WINC_AssociationRSSIGet	Retrieve the current association RSSI.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver Authentication Context implementation (wdrv_winc_authctx.c)**

WDRV_WINC_AuthCtxIsValid	Tests if an authentication context is valid.
WDRV_WINC_AuthCtxSetDefaults	Configures an authentication context into a default state.
WDRV_WINC_AuthCtxSetOpen	Configure an authentication context for Open authentication.
WDRV_WINC_AuthCtxSetWEP	Configure an authentication context for WEP authentication.
WDRV_WINC_AuthCtxSetWPA	Configure an authentication context for WPA-PSK authentication.
WDRV_WINC_AuthCtxSetWPAEnterpriseMSCHAPv2	Configure an authentication context for WPA(2)-Enterprise authentication using MS-CHAPv2.
WDRV_WINC_AuthCtxSetWPAEnterpriseTLS	Configure an authentication context for WPA(2)-Enterprise authentication using TLS.

- **WINC wireless driver Host File Download implementation (wdrv_winc_host_file.c)**

WDRV_WINC_HostFileDownload	Request the WINC downloads a file to its internal memory.
WDRV_WINC_HostFileRead	Requests to read the data from a downloaded file.
WDRV_WINC_HostFileErase	Requests to erase a downloaded file.
WDRV_WINC_HostFileFindByID	Maps a file ID to a driver file handle.
WDRV_WINC_HostFileGetSize	Retrieves the file size from a file handle.

- **WINC wireless driver HTTP Provisioning Context implementation (wdrv_winc_httpprovctx.c)**

WDRV_WINC_HTTPProvCtxSetDefaults	Initialize provisioning store.
WDRV_WINC_HTTPProvCtxSetDomainName	Define the DNS domain name for the provisioning server.
WDRV_WINC_HTTPProvCtxSetURLWildcard	Define the use of a wildcard URL check.
WDRV_WINC_HTTPProvCtxSetConnectCallback	Defines the connection information callback.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver NVM Access implementation (wdrv_winc_nvm.c)**

WDRV_WINC_NVM_EraseSector	Erase a sector within an NVM region.
WDRV_WINC_NVM_Write	Writes data to an NVM region.
WDRV_WINC_NVM_Read	Reads data from an NVM region.
- **WINC wireless driver Power Save implementation (wdrv_winc_powersave.c)**

WDRV_WINC_PowerSave_SetMode	Sets the power save mode.
WDRV_WINC_PowerSave_GetMode	Returns the current power save mode.
WDRV_WINC_PowerSave_SetBeaconInterval	Configures the beacon listening interval.
WDRV_WINC_PowerSave_ManualSleep	Manual sleep power save operation.
- **WINC wireless driver Soft-AP implementation (wdrv_winc_softap.c)**

WDRV_WINC_AP_Start	Starts an instance of Soft-AP.
WDRV_WINC_AP_Stop	Stops an instance of Soft-AP.
WDRV_WINC_AP_SetCustomIE	Configures the custom IE.
- **WINC wireless driver TLS/SSL implementation (wdrv_winc_ssl.c)**

WDRV_WINC_SSLCTX_Default	Initialize an SSL cipher suite context.
WDRV_WINC_SSLCTX_CipherSuitesSet	Set the cipher suites enabled in a context.
WDRV_WINC_SSLCTX_CipherSuitesGet	Returns the enabled cipher suites from a context.
WDRV_WINC_SSLActiveCipherSuitesSet	Set the active cipher suites.
WDRV_WINC_SSLECCHandShakeRsp	Handshake Response for ECC cipher suites.
WDRV_WINC_SSLRetrieveCert	Retrieve the certificate to be verified from the WINC.
WDRV_WINC_SSLRetrieveHash	Retrieve the certificate hash.
WDRV_WINC_SSLStopRetrieveCert	Stop processing the certificate.

SPI Driver – Wi-Fi WINC Driver Library

- **WINC wireless driver Custom IE implementation (wdrv_winc_custie.c)**
 - WDRV_WINC_CustIESToreCtxSetStorage Initialize the custom IE store.
 - WDRV_WINC_CustIESToreCtxAddIE Add data to the custom IE store.
 - WDRV_WINC_CustIESToreCtxRemoveIE Removes data from the custom IE store.
- **WINC wireless driver SysTime implementation (wdrv_winc_systime.c)**
 - WDRV_WINC_LocalTimeToUTC Convert local system time to UTC time.
 - WDRV_WINC_UTCToLocalTime Convert UTC time to local system time.
 - WDRV_WINC_SystemTimeSNTPClientEnable Enable the SNTP client to set system time.
 - WDRV_WINC_SystemTimeSNTPClientDisable Disable the SNTP client.
 - WDRV_WINC_SystemTimeSetCurrent Sets the current system time on the WINC.
 - WDRV_WINC_SystemTimeGetCurrent Requests the current system time from the WINC.
- **WINC wireless driver WPS implementation (wdrv_winc_wps.c)**
 - WDRV_WINC_WPSEnrolleeDiscoveryStartPIN Start WPS discovery (PIN).
 - WDRV_WINC_WPSEnrolleeDiscoveryStartPBC Start WPS discovery (Push Button).
 - WDRV_WINC_WPSEnrolleeDiscoveryStop Stop WPS discovery.
- **WINC wireless driver OS Abstraction Layer implementation (wdrv_winc_oscal.c)**
 - WDRV_MSDelay Performs a delay specified in milli-seconds.
- **WINC Driver BLE Implementation (wdrv_winc_ble.c)**
 - WDRV_WINC_BLEIsStarted Determines if BLE has been started.
 - WDRV_WINC_BLEStart Enables the BLE interface.
 - WDRV_WINC_BLEStop Disables the BLE interface.

Lab 17

AP scan application of SPI Wi-Fi Module

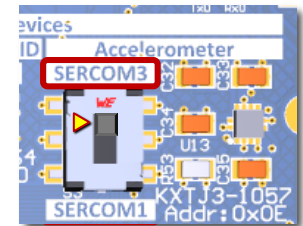
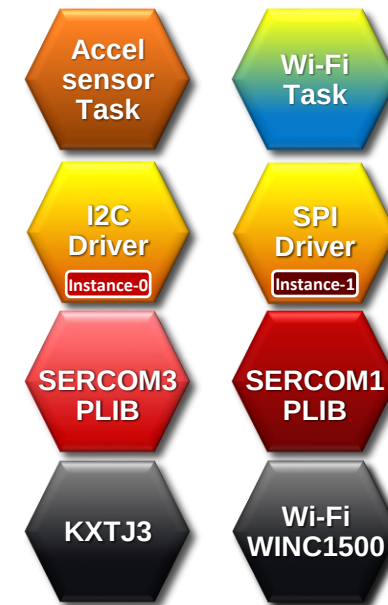
SPI Driver – Wi-Fi WINC Driver Library

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Try implement a **Wi-Fi Access Point (AP) scan** application.
- Toggle **S3** to **SERCOM3** to connect **G-Sensor** on **I2C Driver Instance 0**.
- **Disconnect** **SERCOM1** from **I2C Driver Instance 1** and then **delete(-)** unused **Instance 1**.
- Add(+) **SPI Driver Instance1** to be consumed by **Wi-Fi WINC Driver**.
- **Reconnect** **SERCOM1** to **SPI Driver Instance 1** to communicate with an external “**Wi-Fi 7 Click EVB**” that plug on **mikroBUS** socket.
- The **EIC_EXTINT4** need setup for interrupt handling and two extra GPIOs, custom name to “**WDRV_WINC_RESETN**” and “**WDRV_WINC_CHIP_EN**” for Reset and Enable control.



Wi-Fi 7 click (WINC1500)	mikroBUS socket	SAMD21 Pin Configure (Custom Name)	SAMD21 Pin MUX
SDI	MOSI	PA16	SERCOM1-PAD0
SCK	SCK	PA17	SERCOM1-PAD1
CS	CS	PA18	SERCOM1-PAD2
SDO	MISO	PA19	SERCOM1-PAD3
INT	INT	PA04	EIC_EXTINT4, Falling Edge Detect
RST	RST	PA03 (WDRV_WINC_RESETN)	GPIO output, Latch High
EN	PWM	PA20 (WDRV_WINC_CHIP_EN)	GPIO output, Latch High

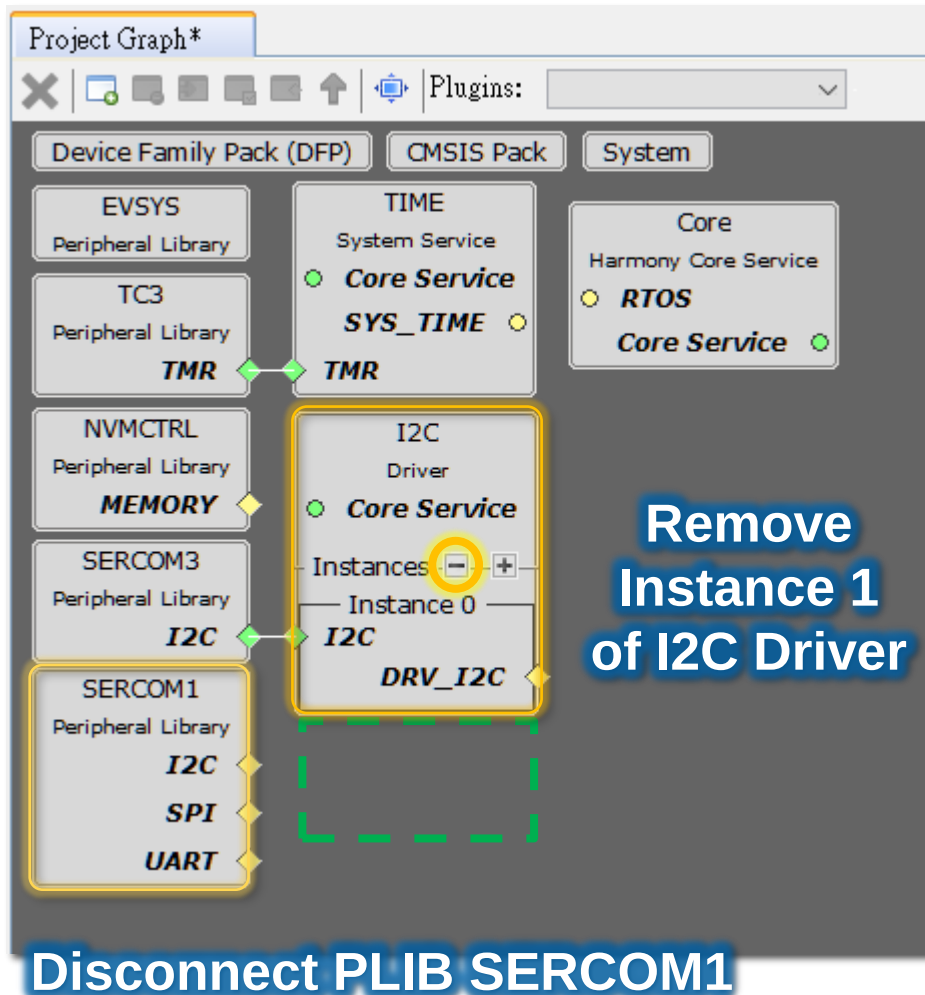


S3 to SERCOM3
for G-Sensor
SERCOM1
for Wi-Fi 7 click

I2C Auto Probe must implement

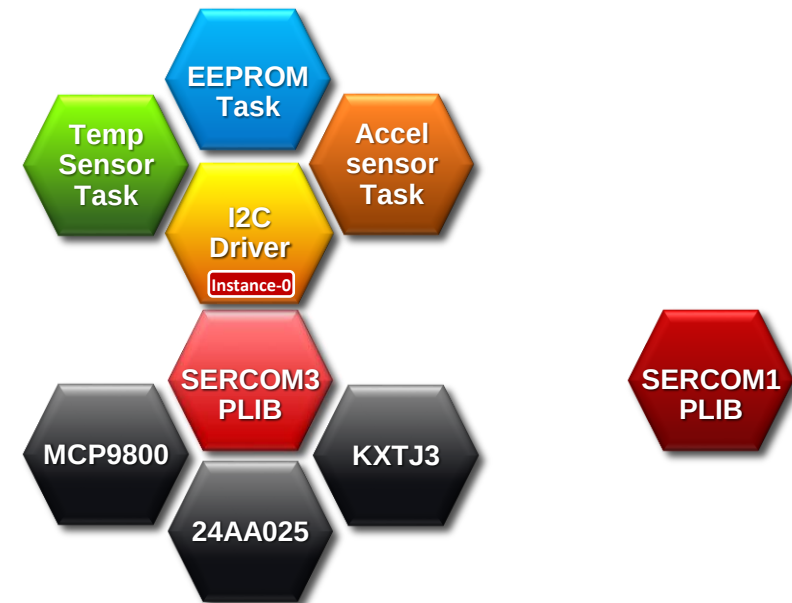
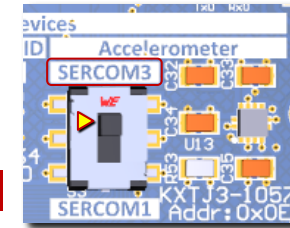
⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Disconnect **SERCOM1** from **I2C driver Instance1** and delete **Instance 1**.



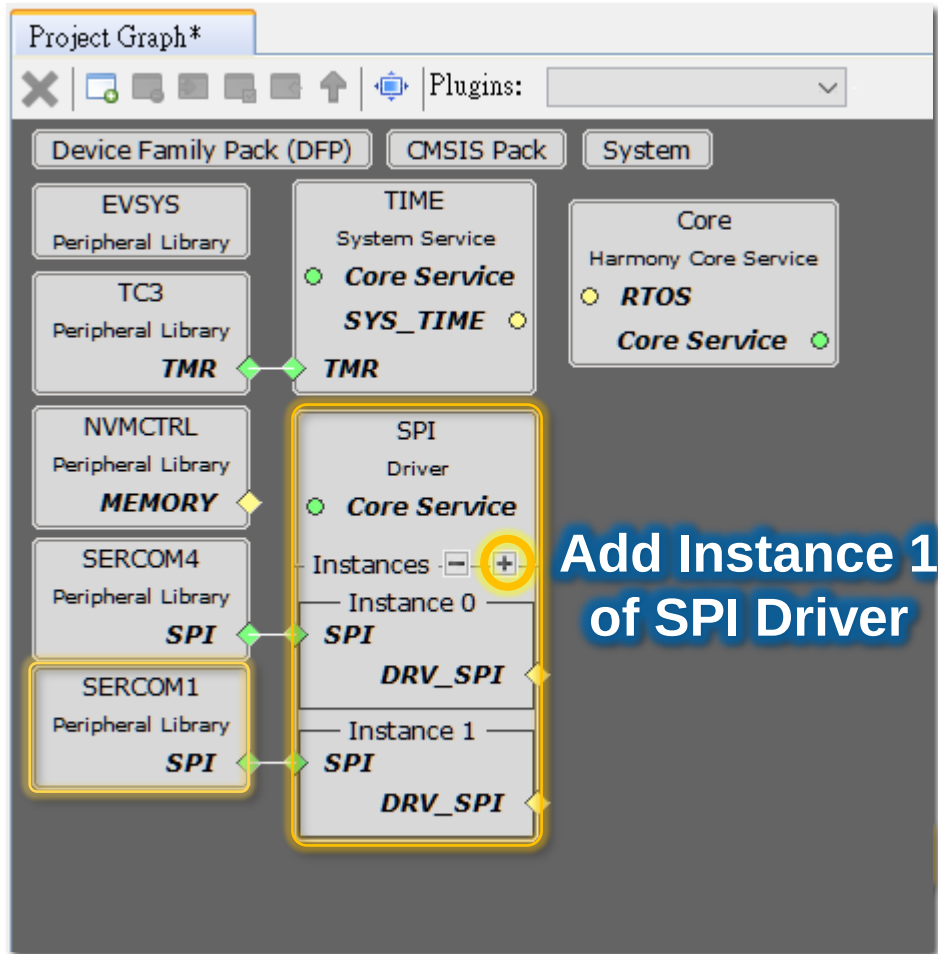
**S3 to SERCOM3
for G-Sensor**

I2C Auto Probe must implement

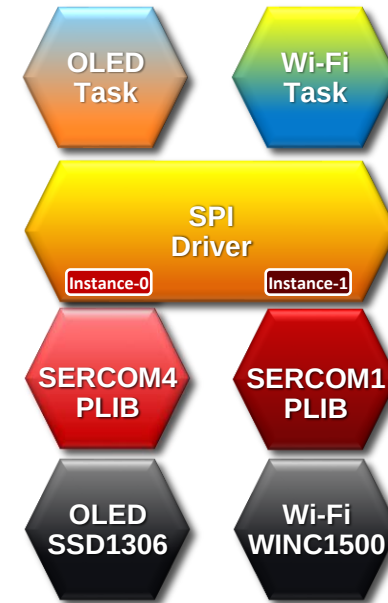


⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Add **SPI Driver Instance 1** and attach **SERCOM1-SPI** as hardware for Wi-Fi module connection.

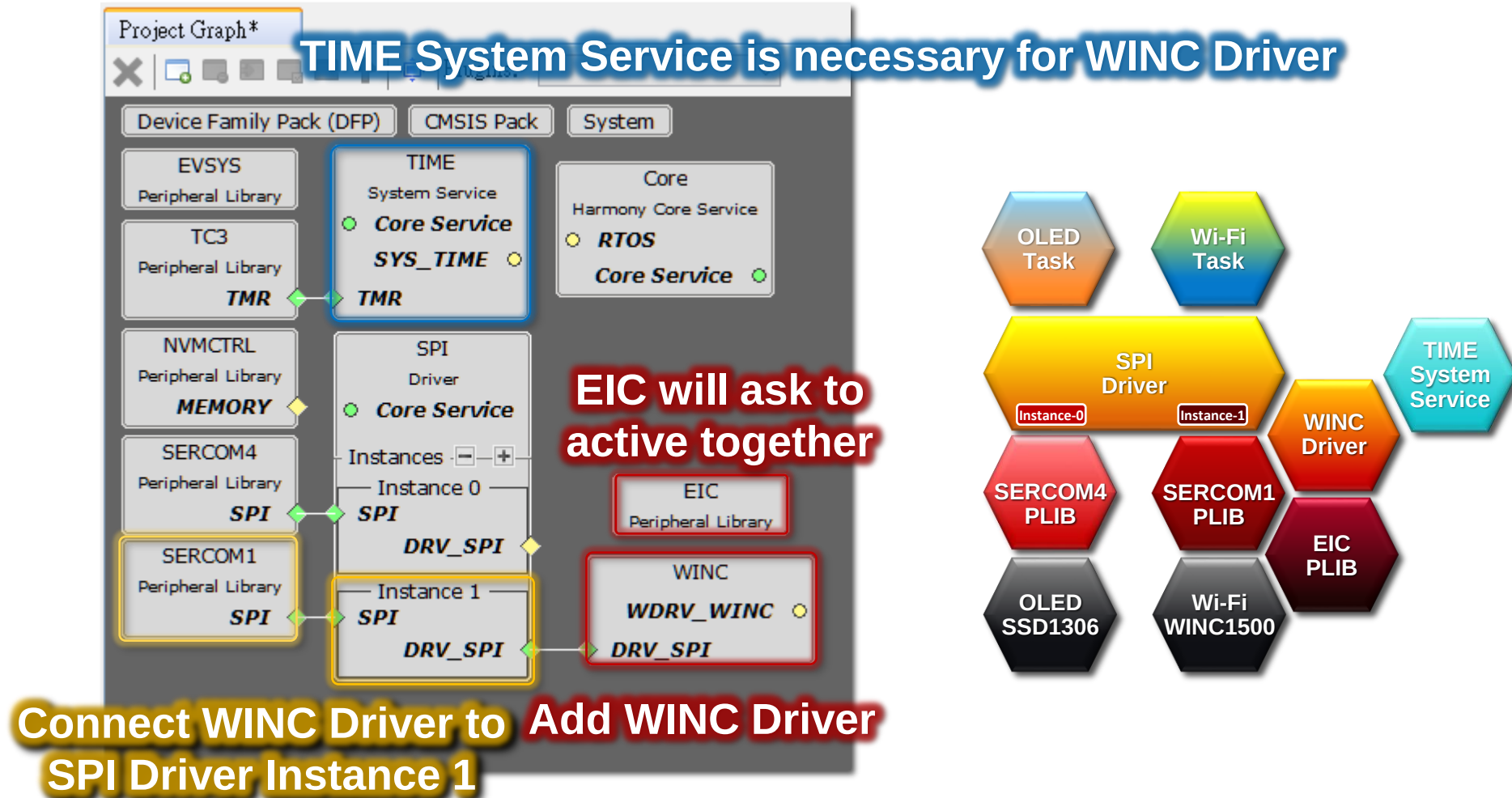


Attach PLIB SERCOM1



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Add **Wi-Fi WINC Driver** and **EIC (External Interrupt)** will ask to active together.
- The **TIME System Service** is necessary for **WINC Driver**.



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Enable DMA of **SPI Driver** and configure pads of **PLIB SERCOM1**.

The image displays two configuration windows from the Microchip Studio IDE, illustrating the setup for the SPI driver and SERCOM1 peripheral.

Top Window: Configuration Options for SERCOM1

- Select SERCOM Operation Mode:** SPI Master
- Enable Interrupts ?** ☒
- Enable operation in Standby mode** ☐
- SPI Data Out Pad:** DO on PAD[0], SCK on PAD[1] and SS on PAD[2]
- SPI Data In Pad Selection:** SERCOM PAD[3] (highlighted with a green box and labeled **DI on PAD3**)
- SPI Data Order:** MSB is transferred first
- SPI Speed in Hz:** 1,000,000
- SPI Data Character Size:** 8 bits
- SPI Clock Phase:** The data is sampled on a leading SCK edge and changed on a trailing SCK edge
- SPI Clock Polarity:** SCK is low when idle
- Enable SPI Master Hardware Slave Select** ☒ (highlighted with a green box and labeled **Enable Hardware Slave Select**)
- SPI Receiver Enable** ☒

Bottom Window: Configuration Options for SPI Driver

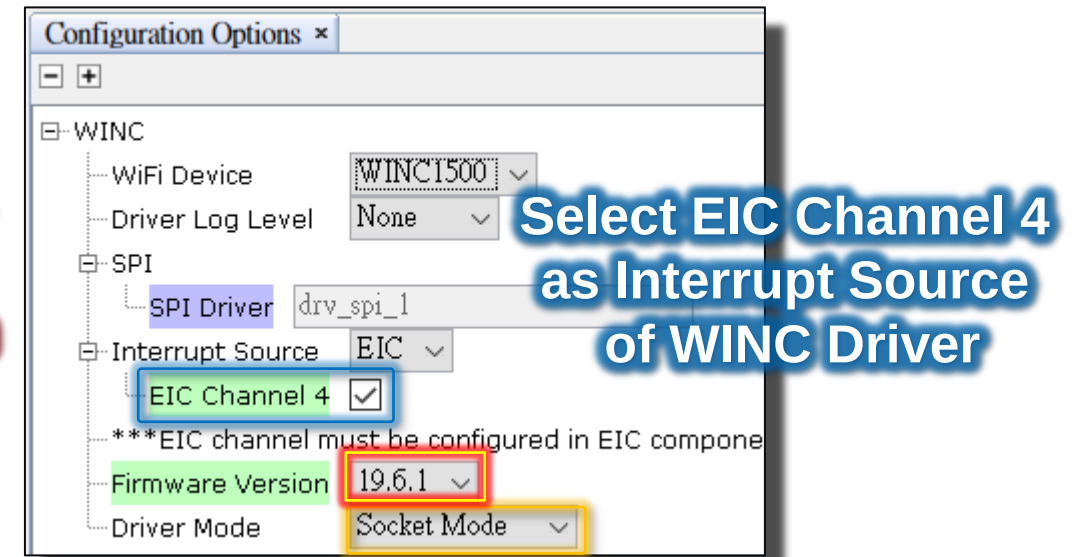
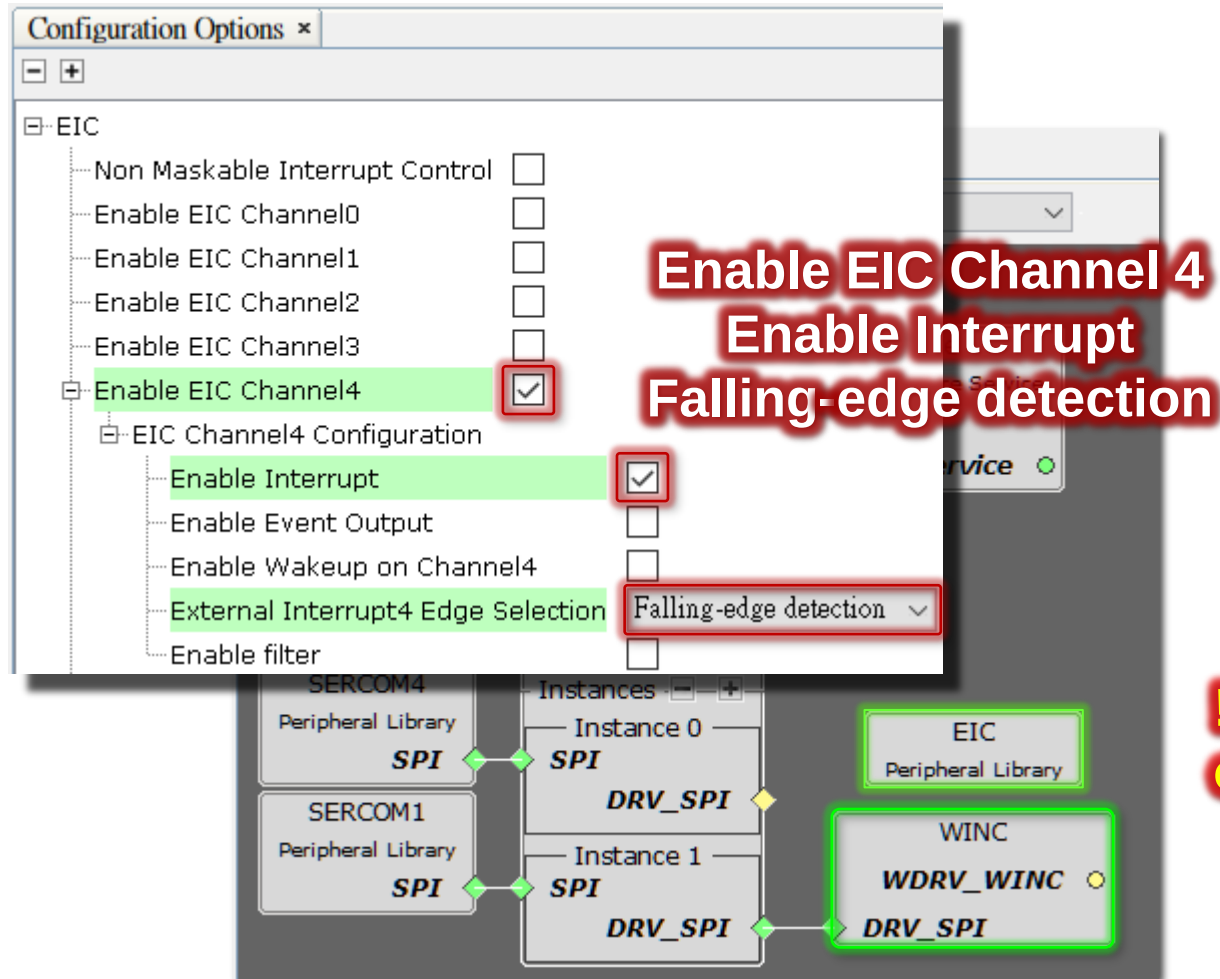
- PLIB Used:** SERCOM1
- Number of Clients:** 1
- Transfer Queue Size:** 4
- Use DMA for Transmit and Receive?** ☒ (highlighted with a green box and labeled **Enable DMA in SPI Driver**)
- DMA Channel For Transmit:** 2
- DMA Channel For Receive:** 3

Peripheral Library Diagram:

The diagram shows the connection between the Peripheral Library and the Driver Library. SERCOM1 (Peripheral Library) is connected to DRV_SPI (Driver Library). The SPI driver is configured to use SERCOM1 as the PLIB. The diagram also shows other peripherals like SERCOM4, EIC, and WINC, and their respective drivers (DRV_SPI, WDRV_WINC).

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Enable interrupt of **EIC Channel4** and **Falling Edge detection**.
- Configure interrupt source of **WINC Driver** to use **EIC Channel4**.



!!! Must select to Correct Firmware of the WIN1500 Module you to use

The default WINC Driver mode has set to Socket Mode

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Configure **SERCOM1-SPI** pads, **EIC_EXTINT4** and custom name **WINC GPIOs**.

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
4	PA03	WDRV_WINC_RESETN	GPIO	Digital	Out	High	☐	☐	NORMAL
9	PA04		EIC_EXTINT4	Digital	High Impedance	n/a	☐	☐	NORMAL
25	PA16		SERCOM1_PAD0	Digital	High Impedance	n/a	☐	☐	NORMAL
26	PA17		SERCOM1_PAD1	Digital	High Impedance	n/a	☐	☐	NORMAL
27	PA18		SERCOM1_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
28	PA19		SERCOM1_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL
29	PA20	WDRV_WINC_CHIP_EN	GPIO	Digital	Out	High	☐	☐	NORMAL
30	PA21	LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL

Custom PA03 as **WDRV_WINC_RESETN**
 Custom PA20 as **WDRV_WINC_CHIP_EN**

Wi-Fi 7 click (WINC1500)	mikroBUS socket	SAMD21 Pin Configure (Custom Name)	SAMD21 Pin MUX
SDI	MOSI	PA16	SERCOM1-PAD0
SCK	SCK	PA17	SERCOM1-PAD1
CS	CS	PA18	SERCOM1-PAD2
SDO	MISO	PA19	SERCOM1-PAD3
INT	INT	PA04	EIC_EXTINT4, Falling Edge Detect
RST	RST	PA03 (WDRV_WINC_RESETN)	GPIO output, Latch High
EN	PWM	PA20 (WDRV_WINC_CHIP_EN)	GPIO output, Latch High
		PA21 (LED2)	GPIO output, Latch Low

Already done in previous I2C Lab

Already done in previous I2C Lab

Additional CS pin for SERCOM-SPI

Additional MISO pin for SERCOM-SPI

External Interrupt on PA04

Reset pin GPIO control

Modify from LED1

For instead of LED1

✖ Lab17 AP scan application of SPI Wi-Fi Module

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**6**” and given the Application Name to to “**app_drv_winc**”.

The image shows two windows from the Harmony IDE. The left window, titled 'Project Graph*', displays a block diagram of the system components. It includes peripheral libraries like EVSYS, TC3, NVMCTRL, SERCOM4, and SERCOM1, along with system services like TIME and SPI. The 'Core' block is highlighted with a green border and contains 'Harmony Core Service', 'RTOS', and 'Core Service'. The right window, titled 'Configuration Options', shows the 'Core' configuration. A red banner at the top of this window reads 'Increase Number of Application'. Under 'Application Configuration', the 'Number of Applications' is set to 6. Under 'Application 5 Configuration', the 'Application Name' is set to 'app_drv_winc'. A note below states: '**** Application name must be valid C-Language id'. Other options like 'Generate Harmony Application Files', 'Generate Harmony Driver Common Files', 'Generate Harmony System Service Common Files', 'Generate Harmony System Media Files', 'Enable System Interrupt', and 'Enable System Ports' are also visible.

Project Graph*

Device Family Pack (DFP) CMSIS Pack System

EVSYS Peripheral Library

TC3 Peripheral Library

NVMCTRL Peripheral Library

SERCOM4 Peripheral Library

SERCOM1 Peripheral Library

TIME System Service

Core Service

SYS_TIME

TMR

SPI Driver

Core Service

Instances

Instance 0

Instance 1

DRV_SPI

DRV_SPI

DRV_SPI

EIC Peripheral Library

WINC

WDRV_WINC

DRV_SPI

Configuration Options *

Increase Number of Application

Core

Generate Harmony Application Files ☒

Application Configuration

Number of Applications 6

Application 0 Configuration

Application 1 Configuration

Application 2 Configuration

Application 3 Configuration

Application 4 Configuration

Application 5 Configuration

Enable Application 5 Configuration ☒

Application Name app_drv_winc

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

Generate Harmony Driver Common Files ☒

Generate Harmony System Service Common Files ☒

Generate Harmony System Media Files ☐

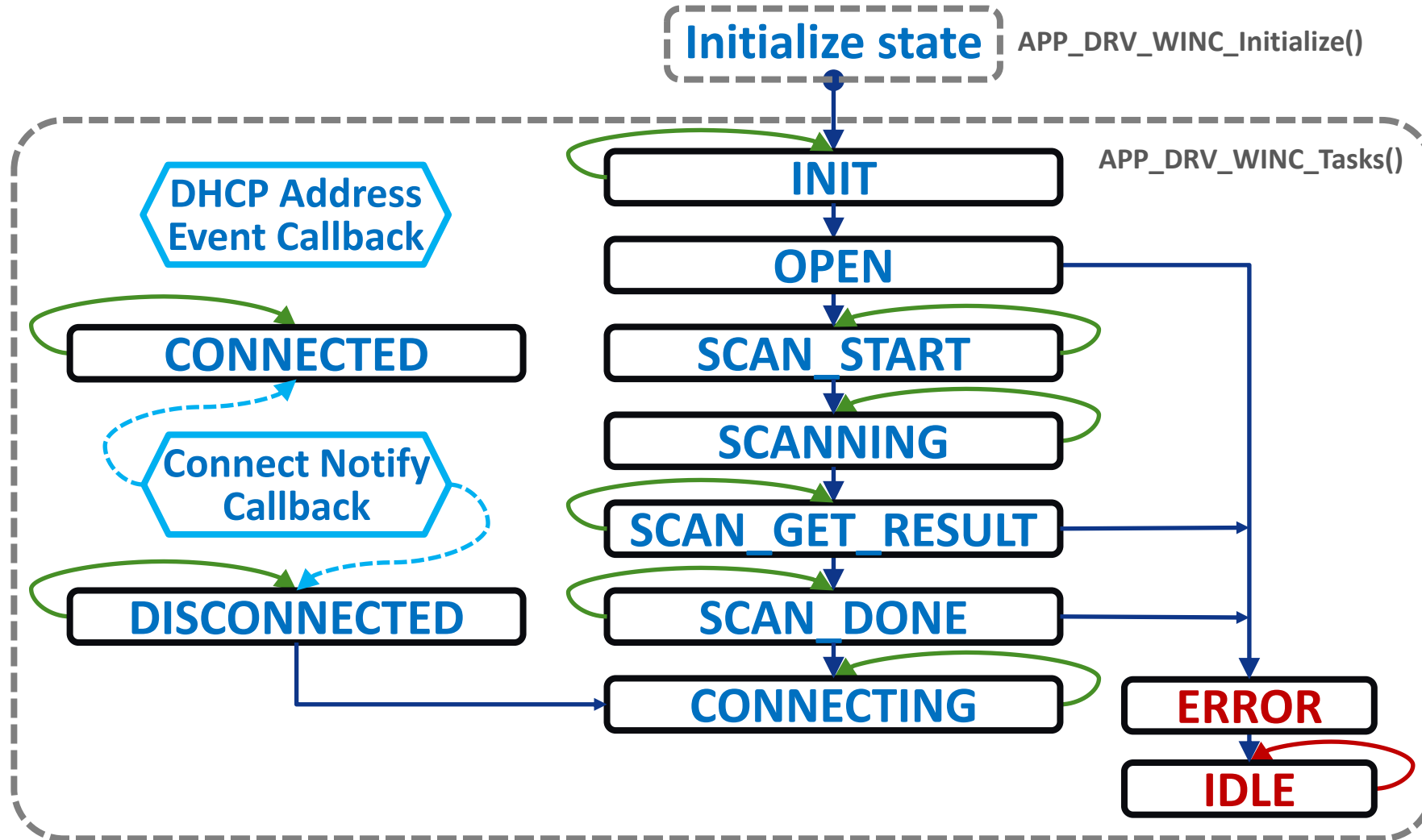
Enable System Interrupt ☒

Enable System Ports ☒

Rename file to “app_drv_winc”

✂ Lab17 AP scan application of SPI Wi-Fi Module

- The state machine to implement in `app_drv_winc.c` as below.



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Add states in “app_drv_winc.h”.

```
#ifndef _APP_DRV_WINC_H
#define _APP_DRV_WINC_H

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

```
typedef enum
{
```

Keep this line

```
APP_DRV_WINC_STATE_INIT=0,
// APP_DRV_WINC_STATE_SERVICE_TASKS,
APP_DRV_WINC_STATE_OPEN,
APP_DRV_WINC_STATE_SCAN_START,
APP_DRV_WINC_STATE_SCANNING,
APP_DRV_WINC_STATE_SCAN_GET_RESULT,
APP_DRV_WINC_STATE_SCAN_DONE,
APP_DRV_WINC_STATE_CONNECTING,
APP_DRV_WINC_STATE_CONNECTED,
APP_DRV_WINC_STATE_DISCONNECTED,
APP_DRV_WINC_STATE_ERROR,
APP_DRV_WINC_STATE_IDLE,
```

Comment out this line

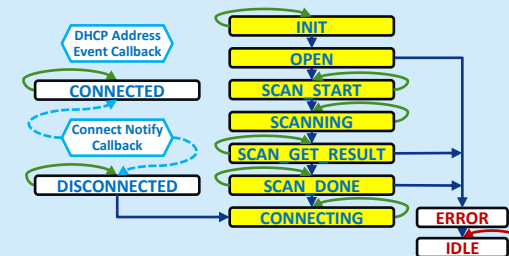
```
} APP_DRV_WINC_STATES;
```

```
typedef struct
```

```
{
    APP_DRV_WINC_STATES state;
} APP_DRV_WINC_DATA;
```

app_drv_winc.h

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



✖ Lab17 AP scan application of SPI Wi-Fi Module

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

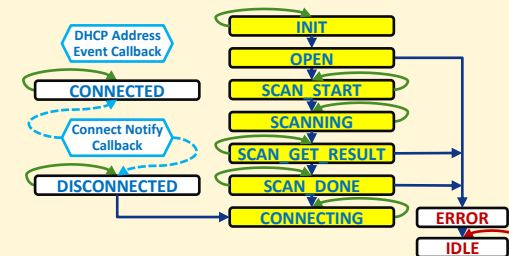
```
void APP_DRV_WINC_Tasks ( void )  
{  
    switch ( app_drv_wincData.state )  
    {  
        case APP_DRV_WINC_STATE_INIT:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_OPEN;  
            break;  
  
        case APP_DRV_WINC_STATE_OPEN:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_START;  
            break;  
  
        case APP_DRV_WINC_STATE_SCAN_START:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;  
            break;  
  
        case APP_DRV_WINC_STATE_SCANNING:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_GET_RESULT;  
            break;  
  
        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_DONE;  
            break;  
  
        case APP_DRV_WINC_STATE_SCAN_DONE :  
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;  
            break;  
  
        case APP_DRV_WINC_STATE_CONNECTING:  
            break;  
    }  
}
```

app_drv_winc.c

Overwrite original example code

Next page to continue

Keep this line



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Construct last four states of task in “app_drv_winc.c”.

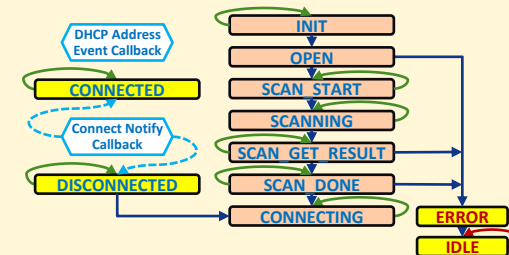
```
void APP_DRV_WINC_Tasks ( void )  
{  
    switch ( app_drv_wincData.state )  
    {  
        case APP_DRV_WINC_STATE_INIT: ...  
        case APP_DRV_WINC_STATE_OPEN: ...  
        case APP_DRV_WINC_STATE_SCAN_START: ...  
        case APP_DRV_WINC_STATE_SCANNING: ...  
        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:  
            ...  
  
        case APP_DRV_WINC_STATE_SCAN_DONE :  
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;  
            break;  
  
        case APP_DRV_WINC_STATE_CONNECTING:  
            break;  
  
        case APP_DRV_WINC_STATE_CONNECTED:  
            break;  
  
        case APP_DRV_WINC_STATE_DISCONNECTED:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;  
            break;  
  
        case APP_DRV_WINC_STATE_ERROR:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE;  
            break;  
  
        case APP_DRV_WINC_STATE_IDLE:  
            break;  
    }  
}
```

app_drv_winc.c

Overwrite original example code

Continue previous

Keep this line



✂ Lab17 AP scan application of SPI Wi-Fi Module

- Header include and Global declarations in “app_drv_winc.c”.

Keep this line

```
#include "app_drv_winc.h"
#include "app_drv_usart1.h"
#include "driver/spi/drv_spi.h"
#include "driver/winc/include/wdrv_winc_client_api.h"
#include <stdio.h>
#include <string.h>
```

app_drv_winc.c

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

```
#define WLAN_SSID "Taipei RTC Mesh"
#define WLAN_AUTH_WPA_PSK
#define WLAN_PSK "0800717718"
WDRV_WINC_AUTH_CONTEXT authCtx;
WDRV_WINC_BSS_CONTEXT bssCtx;
WDRV_WINC_BSS_INFO BSSInfo;
bool foundBSS;
```

```
DRV_HANDLE WINCHandle;
```

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

```
Summary:
    Holds application data
```

```
Description:
    This structure holds the application's data.
```

```
Remarks:
```

```
...
```

SSID of Target AP to connect
The Auth type and Password
Flag foundBSS to note
Target AP has found

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- The callback functions for DHCP address and connection status.

```
// Section: Application Callback Functions
// *****
// *****
void DHCPAddressEventCallback(DRV_HANDLE handle, uint32_t ipAddress)
{
    char s[20];
    USART1printf("IP address is %s\r\n", inet_ntop(AF_INET, &ipAddress, s, sizeof(s)));
}

void ConnectNotifyCallback(DRV_HANDLE handle, WDRV_WINC_ASSOC_HANDLE assocHandle,
                           WDRV_WINC_CONN_STATE currentState, WDRV_WINC_CONN_ERROR errorCode)
{
    switch( currentState )
    {
        case WDRV_WINC_CONN_STATE_CONNECTED:
            USART1printf("Connected\r\n");

            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTED;
            break;

        case WDRV_WINC_CONN_STATE_DISCONNECTED:
            USART1printf("Disconnected\r\n");

            app_drv_wincData.state = APP_DRV_WINC_STATE_DISCONNECTED;
            break;

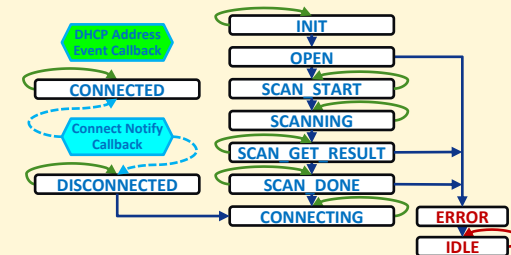
#ifdef WDRV_WINC_DEVICE_BSS_ROAMING
        case WDRV_WINC_CONN_STATE_ROAMED:
            break;
#endif
    }
}
```

app_drv_winc.c

Callback for DHCP assigned IP Address

Callback for Connection status notify

Dynamic state depend to configuration



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Wait WINC Driver ready in state **INIT**, Open Driver and handling state **ERROR**.

```
void APP_DRV_WINC_Tasks ( void )  
{  
    switch ( app_drv_wincData.state )  
    {  
        case APP_DRV_WINC_STATE_INIT:  
            if(WDRV_WINC_Status(sysObj.drivWifiWinc) == SYS_STATUS_READY)  
            {  
                app_drv_wincData.state = APP_DRV_WINC_STATE_OPEN;  
            }  
            break;  
        case APP_DRV_WINC_STATE_OPEN:  
            WINCHandle = WDRV_WINC_Open(0, 0);  
            if(WINCHandle == DRV_HANDLE_INVALID)  
            {  
                app_drv_wincData.state = APP_DRV_WINC_STATE_ERROR;  
                break;  
            }  
            break;  
        case APP_DRV_WINC_STATE_SCAN_START:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;  
            break;  
        ...  
        case APP_DRV_WINC_STATE_ERROR:  
            USART1printf("DRV_WINC error!\r\n");  
            break;  
        ...  
    }  
}
```

app_drv_winc.c

Wait WINC Driver ready

Open WINC Driver to get Driver Handle

Handling error state message

Keep this line app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_START; break;

Keep this line app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE; break;

```
graph TD
    INIT[INIT] --> OPEN[OPEN]
    OPEN --> SCAN_START[SCAN_START]
    SCAN_START --> SCANNING[SCANNING]
    SCANNING --> SCAN_GET_RESULT[SCAN_GET_RESULT]
    SCAN_GET_RESULT --> SCAN_DONE[SCAN_DONE]
    SCAN_DONE --> CONNECTING[CONNECTING]
    CONNECTING --> DISCONNECTED[DISCONNECTED]
    DISCONNECTED --> CONNECTED[CONNECTED]
    CONNECTED --> CONNECTED
    CONNECTED --> INIT
    CONNECTED --> ERROR[ERROR]
    ERROR --> IDLE[IDLE]
```

✂ Lab17 AP scan application of SPI Wi-Fi Module

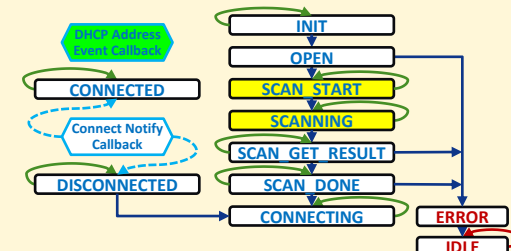
- Register DHCP callback to get IP address and start find AP station

```
void APP_DRV_WINC_Tasks ( void )  
{  
    switch ( app_drv_wincData.state )  
    {  
        case APP_DRV_WINC_STATE_INIT: ...  
        case APP_DRV_WINC_STATE_OPEN: ...  
  
        case APP_DRV_WINC_STATE_SCAN_START:  
            WDRV_WINC_IPUseDHCPSet( WINCHandle, &DHCPAddressEventCallback );  
  
            if( WDRV_WINC_BSSFindFirst(WINCHandle, WDRV_WINC_ALL_CHANNELS, true, NULL, NULL)  
                == WDRV_WINC_STATUS_OK )  
            {  
                app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;  
                foundBSS = false;  
            }  
            break;  
  
        case APP_DRV_WINC_STATE_SCANNING:  
            if( WDRV_WINC_BSSFindInProgress(WINCHandle) == false )  
            {  
                USART1printf("Scan complete, %d AP(s) found\r\n",  
                             WDRV_WINC_BSSFindGetNumBSSResults(WINCHandle));  
                app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_GET_RESULT;  
            }  
            break;  
  
        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:  
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_DONE;  
            break;  
  
        case APP_DRV_WINC_STATE_SCAN_DONE :  
            ...  
    }  
}
```

app_drv_winc.c

Register the callback function to retrieval DHCP IP address and try find first AP station

Check if AP scanning to finish or not



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- List found AP stations and check if Target AP in list or not.

```
void APP_DRV_WINC_Tasks ( void )  
{  
    switch ( app_drv_wincData.state )  
    {  
        case APP_DRV_WINC_STATE_INIT: ...  
        case APP_DRV_WINC_STATE_OPEN: ...  
        case APP_DRV_WINC_STATE_SCAN_START: ...  
        case APP_DRV_WINC_STATE_SCANNING: ...  
        case APP_DRV_WINC_STATE_SCAN_GET_RESULT:  
            if( WDRV_WINC_BSSFindGetInfo(WINCHandle, &BSSInfo) == WDRV_WINC_STATUS_OK )  
            {  
                USART1printf("AP found: RSSI: %d %s\r\n", BSSInfo.rssi, BSSInfo.ctx.ssid.name);  
  
                if( ((sizeof(WLAN_SSID)-1) == BSSInfo.ctx.ssid.length) &&  
                    (0 == memcmp(BSSInfo.ctx.ssid.name, WLAN_SSID, BSSInfo.ctx.ssid.length)) )  
                {  
                    foundBSS = true;  
                }  
            }  
            break;  
        case APP_DRV_WINC_STATE_SCAN_DONE :  
            app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;  
            break;  
        case APP_DRV_WINC_STATE_CONNECTING:  
            break;  
        case APP_DRV_WINC_STATE_CONNECTED:  
            break;  
        ...  
    }  
}
```

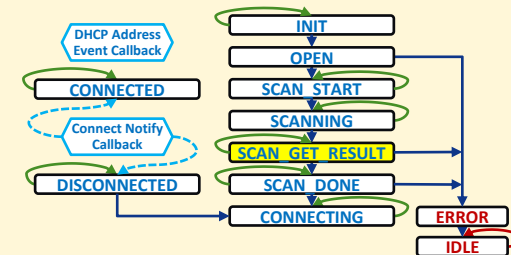
app_drv_winc.c

List found AP stations and check if Target AP in list or not.

Set foundBSS flag if find Target AP

Next page to continue

Keep this line



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Check if end of AP finding and try connect Target AP if it's existed.

app_drv_winc.c

```
...
case APP_DRV_WINC_STATE_SCAN_GET_RESULT:
    if( WDRV_WINC_BSSFindGetInfo(WINCHandle, &BSSInfo) == WDRV_WINC_STATUS_OK )
    {
        ...
        foundBSS = true;
    }
    }
    switch( WDRV_WINC_BSSFindNext(WINCHandle, NULL) )
    {
    case WDRV_WINC_STATUS_BSS_FIND_END:
        if( foundBSS == true )
        {
            USART1printf("Target AP found, trying to connect\r\n");
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_DONE;
        }
        else
        {
            USART1printf("Target BSS not found\r\n");
            app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE;
        }
        break;
    case WDRV_WINC_STATUS_NOT_OPEN:
    case WDRV_WINC_STATUS_INVALID_ARG:
        app_drv_wincData.state = APP_DRV_WINC_STATE_ERROR;
        break;
    default:
        break;
    }
    break;
}
case APP_DRV_WINC_STATE_SCAN_DONE : ...
```

Check if end of AP finding

Try to connect Target AP

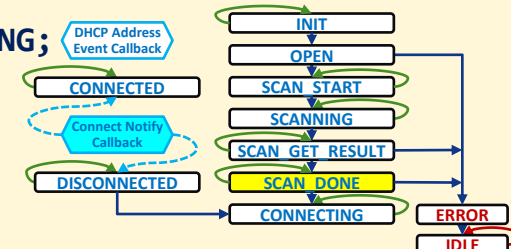
Keep this line

```
graph TD
    INIT[INIT] --> OPEN[OPEN]
    OPEN --> CONNECTED[CONNECTED]
    CONNECTED -- "DHCP Address Event Callback" --> CONNECTED
    CONNECTED --> SCAN_START[SCAN_START]
    SCAN_START --> SCANNING[SCANNING]
    SCANNING --> SCAN_GET_RESULT[SCAN_GET_RESULT]
    SCAN_GET_RESULT --> SCAN_DONE[SCAN_DONE]
    SCAN_DONE --> CONNECTING[CONNECTING]
    CONNECTING -- "Connect Notify Callback" --> CONNECTED
    CONNECTING --> IDLE[IDLE]
    IDLE -- "ERROR" --> IDLE
```

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Start connect to Target AP, the result will return through **ConnectNotifyCallback**.

```
void APP_DRV_WINC_Tasks ( void )  
{  
    switch ( app_drv_wincData.state )  
    {  
        ...  
        case APP_DRV_WINC_STATE_SCAN_DONE :  
            app_drv_wincData.state = APP_DRV_WINC_STATE_ERROR; Set SSID of Target AP  
  
            if( WDRV_WINC_BSSCtxSetDefaults(&bssCtx) != WDRV_WINC_STATUS_OK )  
            { break; }  
  
            if( WDRV_WINC_BSSCtxSetSSID(&bssCtx, (uint8_t*)WLAN_SSID, strlen(WLAN_SSID)) !=  
                WDRV_WINC_STATUS_OK )  
            { break; }  
  
            Authentication must use respective function  
  
            #if defined(WLAN_AUTH_OPEN)  
                if( WDRV_WINC_AuthCtxSetOpen(&authCtx) ) != WDRV_WINC_STATUS_OK )  
            #elif defined(WLAN_AUTH_WPA_PSK)  
                if( WDRV_WINC_AuthCtxSetWPA(&authCtx, (uint8_t*)WLAN_PSK, strlen(WLAN_PSK)) !=  
                    WDRV_WINC_STATUS_OK )  
            #endif  
            { break; }  
  
            Connect AP station and register callback  
  
            if( WDRV_WINC_BSSConnect(WINCHandle, &bssCtx, &authCtx, &ConnectNotifyCallback) ==  
                WDRV_WINC_STATUS_OK )  
            {  
                app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;  
            }  
            Keep this line break;  
  
            case APP_DRV_WINC_STATE_CONNECTING:  
                break;  
        ...  
    }  
}
```



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Active IP link in state **CONNECTED** and handling disconnect in state **DISCONNECTED**.

```
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT: ...
        case APP_DRV_WINC_STATE_OPEN: ...
        case APP_DRV_WINC_STATE_SCAN_START: ...
        case APP_DRV_WINC_SCANNING: ...
        case APP_DRV_WINC_STATE_SCAN_GET_RESULT: ...
        case APP_DRV_WINC_STATE_SCAN_DONE : ...
```

```
        case APP_DRV_WINC_STATE_CONNECTING:
            break;
```

```
        case APP_DRV_WINC_STATE_CONNECTED:
            WDRV_WINC_IPLinkActive( WINCHandle );
            break;
```

Keep this line

```
        case APP_DRV_WINC_STATE_DISCONNECTED:
            if( WDRV_WINC_BSSReconnect(WINCHandle, &ConnectNotifyCallback) ==
                                                    WDRV_WINC_STATUS_OK )
            {
                app_drv_wincData.state = APP_DRV_WINC_STATE_CONNECTING;
            }
            break;
```

Keep this line

```
        case APP_DRV_WINC_STATE_ERROR:
            USART1printf("DRV_WINC error!\r\n");

            app_drv_wincData.state = APP_DRV_WINC_STATE_IDLE;
            break;

        case APP_DRV_WINC_STATE_IDLE:
```

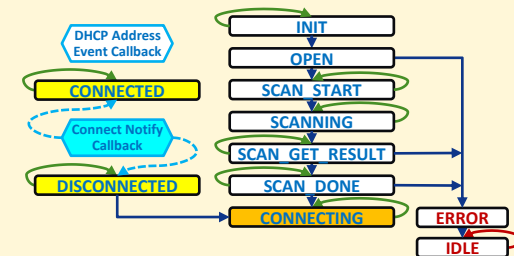
app_drv_winc.c

Do not thing in state **CONNECTING**

Connect Notify Callback

will handling state change of

CONNECTED and **DISCONNECTED**



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Use **LED2** to instead of removed LED1 in “**main.c**” to indicate Driver non-blocking.

```
main.c
SYS_TIME_HANDLE tmrHandle;

// *****
// *****
// Section: Main Entry Point
// *****
// *****

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

    SYS_TIME_DelayMS(1000, &tmrHandle);

    while ( true )
    {
        if( SYS_TIME_DelayIsComplete(tmrHandle) == true )
        {
            LED2_Toggle();
            SYS_TIME_DelayMS(1000, &tmrHandle);
        }

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

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

    return ( EXIT_FAILURE );
}
```

Change to **LED2_Toggle()**

⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Modify I2C auto probe count to disable **Instance 1** of **DRV_I2C** in “app_drv_i2c2.c”
- **You must** implement the **Advanced Exercise : I2C Auto Probe after Lab15.**

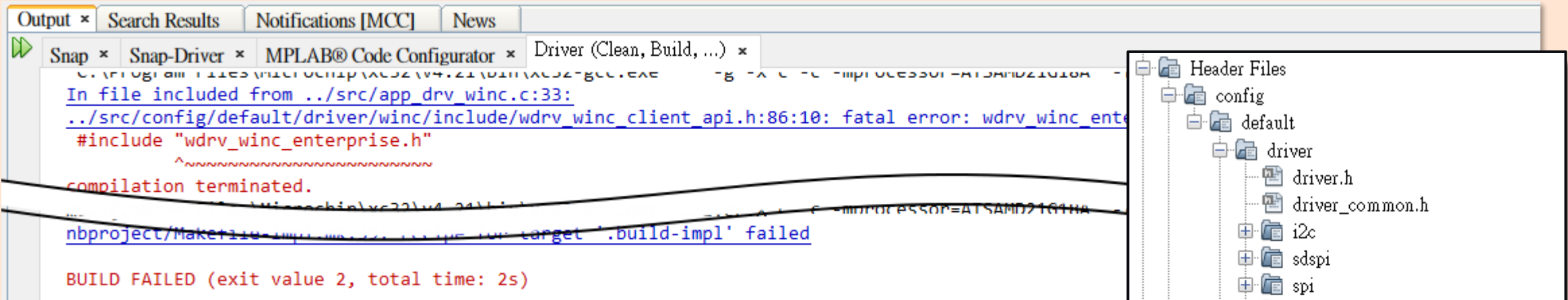
```
void APP_DRV_I2C2_Tasks ( void )  
{  
    switch ( app_drv_i2c2Data.state )  
    {  
        case APP_DRV_I2C2_STATE_INIT:  
            ...  
            // SYS_TIME_HANDLE TimeHandle;  
            // SYS_TIME_DelayMS(10, &TimeHandle );  
            // while(!SYS_TIME_DelayIsComplete(TimeHandle)) {}  
  
            for( int Inst = DRV_I2C_INDEX_0 ; Inst <= DRV_I2C_INDEX_0 ; Inst++ )  
            {  
                I2C2Handle_TJ3 = DRV_I2C_Open(Inst, DRV_IO_INTENT_READWRITE);  
                if(I2C2Handle_TJ3 == DRV_HANDLE_INVALID)  
                {  
                    app_drv_i2c2Data.state = APP_DRV_I2C2_STATE_ERROR;  
                    return;  
                }  
                I2C2WriteBuf_TJ3[0] = 0x1D;  
                I2C2WriteBuf_TJ3[1] = 0x80;  
                DRV_I2C_WriteTransferAdd( I2C2Handle_TJ3, KXTJ3_ADDRESS,  
                                         I2C2WriteBuf_TJ3, 2, & I2C2WriteReadHandle_TJ3 );  
            }  
  
            USART1printf("G-Sensor Software Reset Completed\r\n");  
  
            ...  
        }  
    }
```

app_drv_i2c2.c

Change to **DRV_I2C_INDEX_0**

⚡ Lab17 AP scan application of SPI Wi-Fi Module

- Compiler error because miss “wdrv_winc_enterprise.h” in wdrv_winc_client_api.h



The screenshot shows the MPLAB Code Configurator interface. The 'Output' window displays a fatal error: 'fatal error: wdrv_winc_enterprise.h: No such file or directory'. The error message includes the file path and the line number where the error occurred. The 'Header Files' window on the right shows the project structure, including the 'winc' directory and its 'include' subdirectory. The 'wdrv_winc_client_api.h' file is highlighted in the 'include' subdirectory.

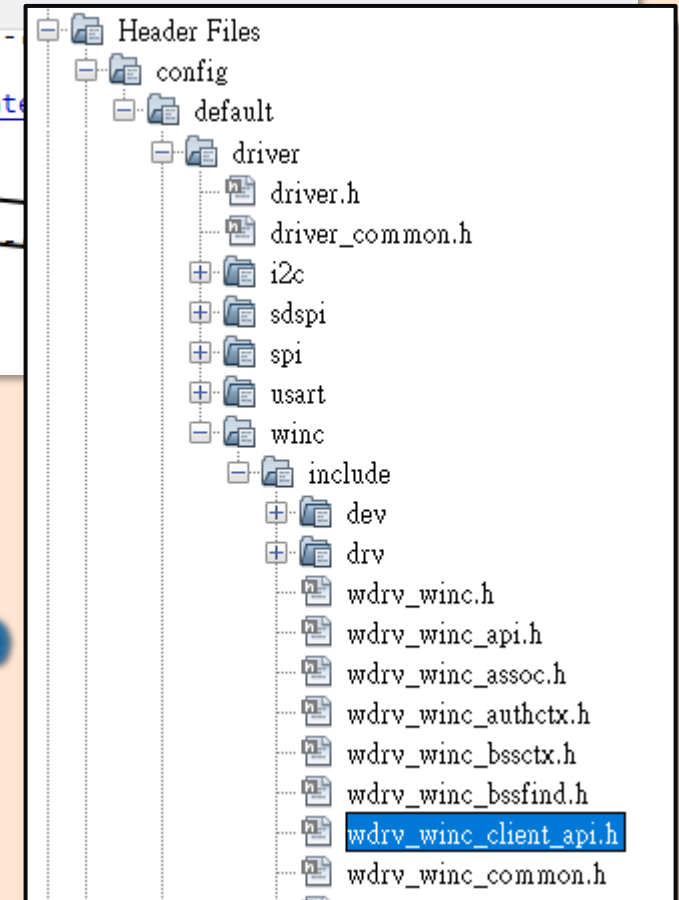
```
Output × Search Results Notifications [MCC] News
Snap × Snap-Driver × MPLAB® Code Configurator × Driver (Clean, Build, ...) ×
C:\Program Files\Microchip\XC32\v4.21\bin\XC32-gcc.exe -g -x c -c -mprocessor=ATSAM21G1HA -
In file included from ../src/app_drv_winc.c:33:
../src/config/default/driver/winc/include/wdrv_winc_client_api.h:86:10: fatal error: wdrv_winc_enterprise.h
#include "wdrv_winc_enterprise.h"
^~~~~~
compilation terminated.
nbproject/Makefile110-impl.mk:44: recipe for target '.build-impl' failed
BUILD FAILED (exit value 2, total time: 2s)
```

```
...
#include "wdrv_winc_ssl.h"
#endif
#ifdef WDRV_WINC_DEVICE_HOST_FILE_DOWNLOAD
#include "wdrv_winc_host_file.h"
#endif
#ifdef WDRV_WINC_ENABLE_BLE
#include "wdrv_winc_ble.h"
#endif
#include "wdrv_winc_nvm.h"
#ifdef WDRV_WINC_DEVICE_ENTERPRISE_CONNECT
// #include "wdrv_winc_enterprise.h"
#endif

#endif /* _WDRV_WINC_CLIENT_API_H */
```

Comment out this line

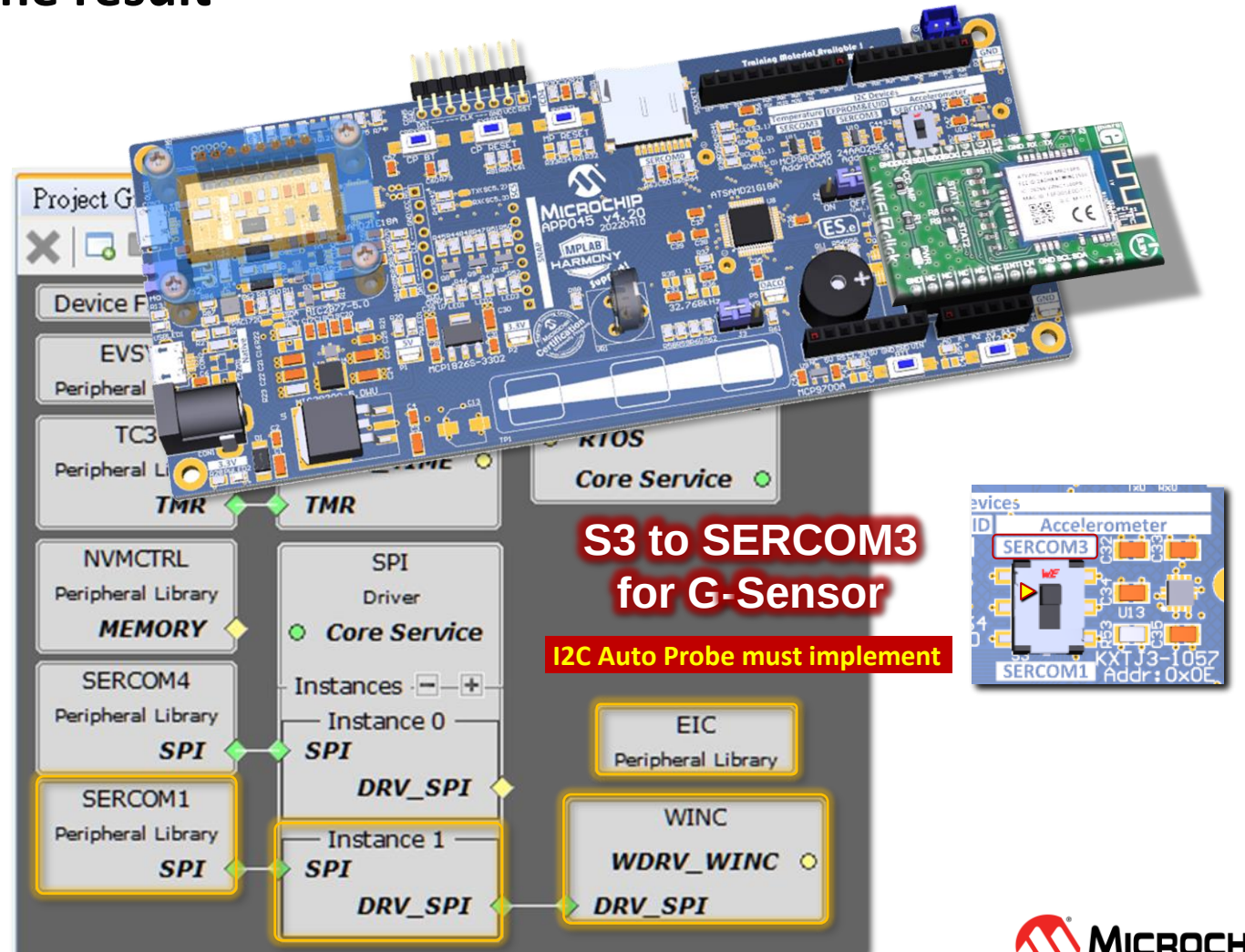
Please comment out below code
In wdrv_winc_client_api.h



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

- Compiler and programming your project to EVB.
- Check USART1 debug output to see the result

```
VT COM3 - Tera Term VT
File Edit Setup Control Window Help
DRV_USART1: Hello world
Wrote 0x24 to EEPROM 24AA
G-Sensor Software Reset Completed
G-Sensor WHO AM I : 0x35
G-Sensor Initial Completed
Scan complete, 12 AP(s) found
AP found: RSSI: -66 BravoGroup
AP found: RSSI: -67 mchp-secure
AP found: RSSI: -68 mguest
AP found: RSSI: -55 Taipei RTC ADSL
AP found: RSSI: -73 Leedarson03
AP found: RSSI: -48 MLI_WiFi
AP found: RSSI: -47 MLI_CHTWiFi
AP found: RSSI: -48 MLI_WiFi_OTP
AP found: RSSI: -41 roving1
AP found: RSSI: -76 TP-Link_E13A
AP found: RSSI: -48
Target AP found, trying to connect
Connected
IP address is 192.168.4.134
```



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

Discuss 1

- How to know the firmware version of WINC modules ?

```
...
DRV_HANDLE WINCHandle;
WDRV_WINC_FIRMWARE_VERSION_INFO FirmwareVersion;

...
void APP_DRV_WINC_Tasks ( void )
{
    switch ( app_drv_wincData.state )
    {
        case APP_DRV_WINC_STATE_INIT:
            ...

        case APP_DRV_WINC_STATE_OPEN:
            WINCHandle = WDRV_WINC_Open(0, 0);
            if(WINCHandle == DRV_HANDLE_INVALID)
            {
                app_drv_wincData.state = APP_DRV_WINC_STATE_ERROR;
                break;
            }

            WDRV_WINC_InfoDeviceFirmwareVersionGet( WINCHandle, true, &FirmwareVersion );
            USART1printf("Firmware Version : %d.%d.%d\r\n", FirmwareVersion.version.major,
                FirmwareVersion.version.minor, FirmwareVersion.version.patch );

            app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_START;
            break;

        case APP_DRV_WINC_STATE_SCAN_START:
            app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;
            break;

        ...
    }
}
```

Insert between this

Insert between this

```
WDRV_WINC_InfoDeviceFirmwareVersionGet( WINCHandle, true, &FirmwareVersion );
USART1printf("Firmware Version : %d.%d.%d\r\n", FirmwareVersion.version.major,
    FirmwareVersion.version.minor, FirmwareVersion.version.patch );
```

```
app_drv_wincData.state = APP_DRV_WINC_STATE_SCAN_START;
break;
```

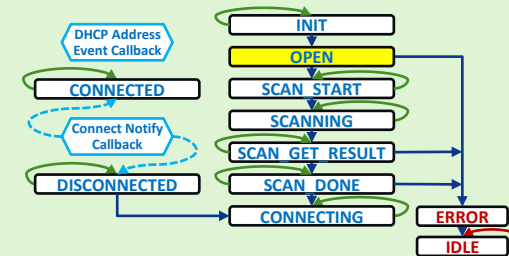
```
case APP_DRV_WINC_STATE_SCAN_START:
    app_drv_wincData.state = APP_DRV_WINC_STATE_SCANNING;
    break;
...
```

app_drv_winc.c

COM3 - Tera Term VT

File Edit Setup Control Window Help

DRV_USART1: Hello world
Wrote 0x24 to EEPROM 24AA
G-Sensor Software Reset Completed
G-Sensor WHO AM I : 0x35
G-Sensor Initial Completed
Firmware Version : 19.6.1
Scan complete, 2 AP(s) found



✖ Lab17 AP scan application of SPI Wi-Fi Module

Discuss 2

- How to setup the **SPI Software Slave Select pin** in WINC SPI Driver ?
 - You can assign a **GPIO** via Transfer Setup in **WDRV_WINC_SPIOpen()** of “wdrv_winc_spi.c”

```
bool WDRV_WINC_SPIOpen(void)
{
    if (DRV_HANDLE_INVALID == spiDcpt.spiHandle)
    {
        spiDcpt.spiHandle = DRV_SPI_Open(spiDcpt.cfg.drvIndex, ...
        if (DRV_HANDLE_INVALID == spiDcpt.spiHandle)
        {
            WDRV_DBG_ERROR_PRINT("SPI open failed\r\n");
            return false;
        }
    }

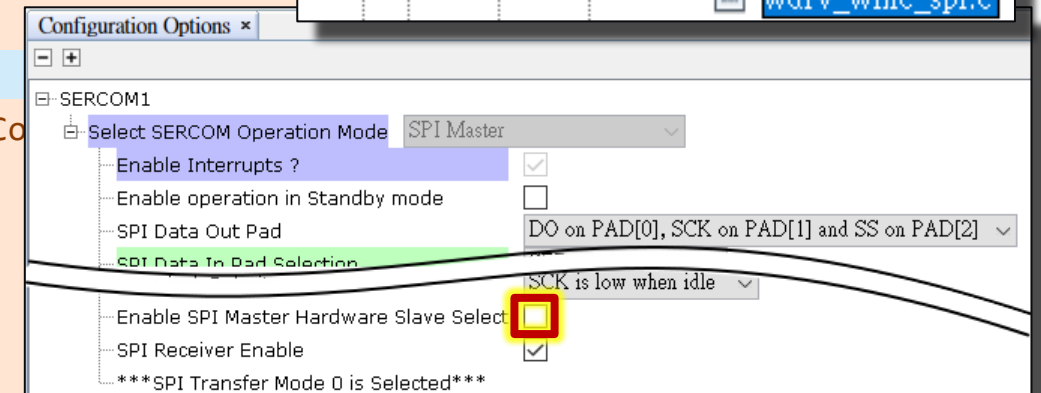
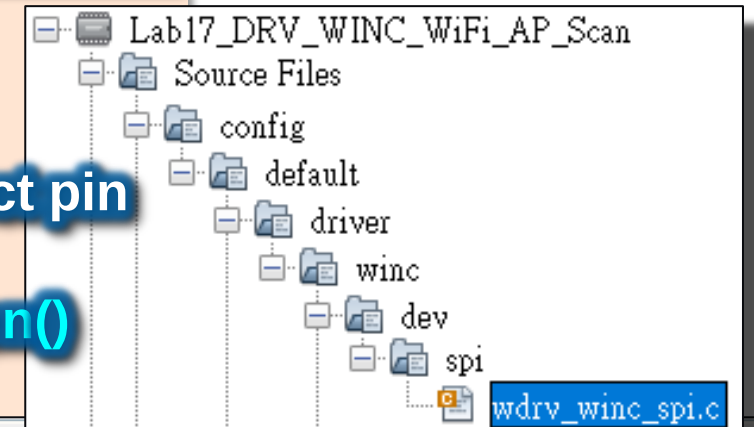
    spiTransConf.baudRateInHz = spiDcpt.cfg.baudRateInHz;
    spiTransConf.chipSelect = (SYS_PORT_PIN)WDRV_WINC_CS_PIN;

    if (false == DRV_SPI_TransferSetup(spiDcpt.spiHandle, &spiTransConf)
    {
        WDRV_DBG_ERROR_PRINT("SPI transfer setup failed\r\n");
        return false;
    }
    return true;
}
```

Setup software Slave Select pin
WDRV_WINC_CS
in WDRV_WINC_SPIOpen()

PA16		SERCOM1_PAD0	Digital	High Impedance	n/a
PA17		SERCOM1_PAD1	Digital	High Impedance	n/a
PA18	WDRV_WINC_CS	GPIO	Digital	Out	High
PA19		SERCOM1_PAD3	Digital	High Impedance	n/a

wdrv_winc_spi.c



⚙️ Lab17 AP scan application of SPI Wi-Fi Module

Discuss 3

- What's difference of WINC and WINC (Lite) ?

The image displays the configuration of the WINC and WINC (Lite) modules in the Microchip Studio IDE. The top section shows the WINC configuration, and the bottom section shows the WINC (Lite) configuration. A callout box on the left highlights the 'Wireless' drivers, including WINC and WINC (Lite).

WINC Configuration:

- WiFi Device: WINC1500
- Driver Log Level: None
- SPI Driver: drv_spi_1
- Interrupt Source: EIC
- EIC Channel 4: ☒
- Firmware Version: 19.6.1
- Driver Mode: Socket Mode

WINC (Lite) Configuration:

- WiFi Device: WINC1500
- Driver Log Level: None
- SPI Driver: drv_spi_1
- Interrupt Source: EIC
- EIC Channel 4: ☒
- Firmware Version: 19.7.7
- Driver Mode: Socket Mode
- Reset assert time (in ms): 0
- Enable serial bridge?: ☒
- Support baud rate changes?: ☐
- Send sync ID before request?: ☐

Memory Usage Summary:

Module	Data Used (bytes)	Data Free (bytes)	Program Used (bytes)	Program Free (bytes)
WINC	4,575 (0x11DF)	28,193 (0x6E21)	70,588 (0x113BC)	191,556 (0x2EC44)
WINC (Lite)	4,642 (0x1222)	28,126 (0x6DDE)	67,890 (0x10932)	194,254 (0x2F6CE)

Support both v19.6.1 and v19.7.x

Support only v19.7.x

Support Serial Bridge

Microchip

SPI Driver – SD Card Driver Library

Harmony Driver

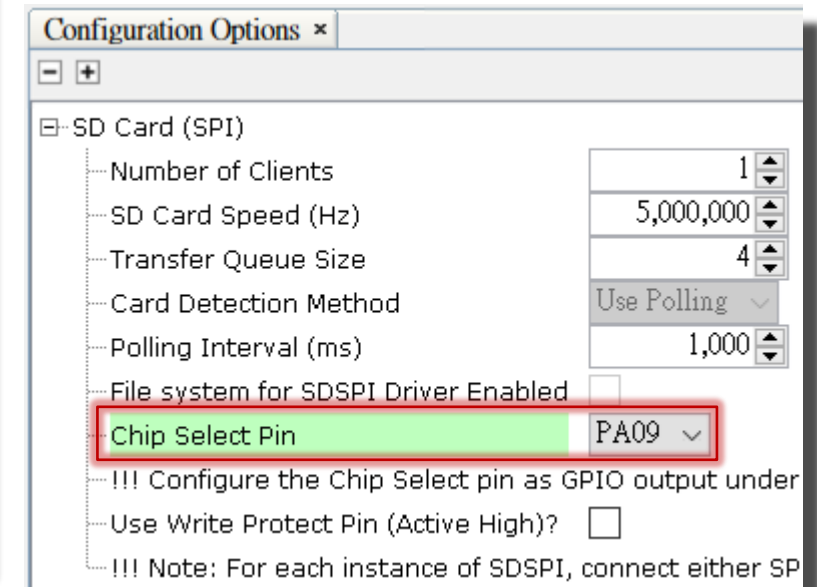
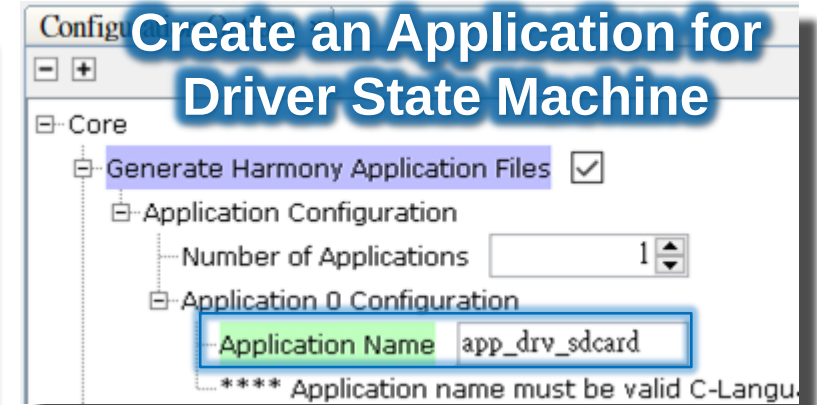
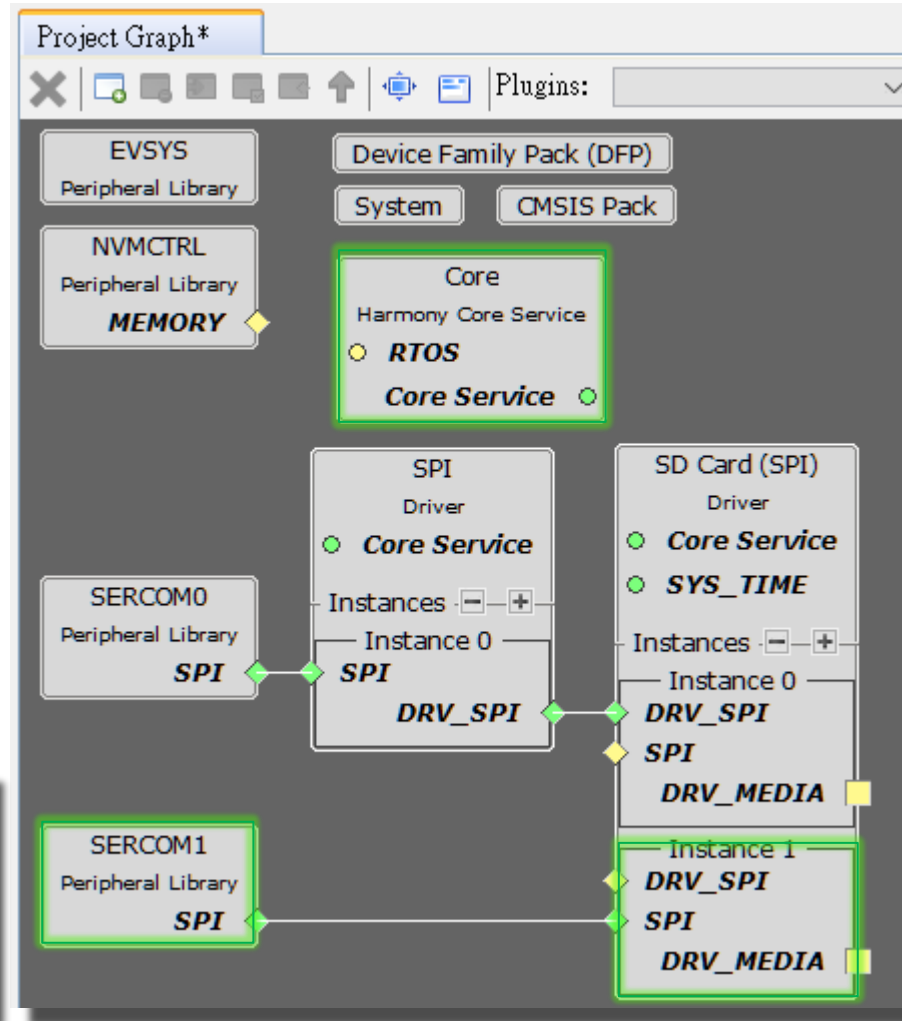
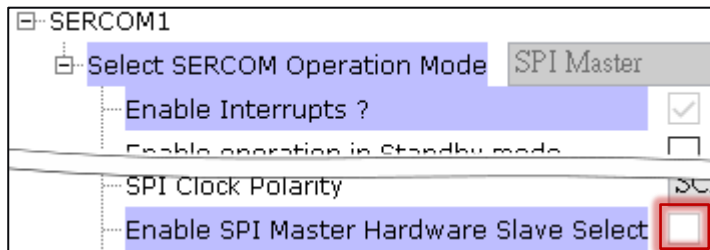
SD Card (SPI) Driver Library

- **Introduction**

- This driver library provides application ready routines to read and write to the SD Card, thus minimizing developer's awareness of working of the SD Card protocol.
- Supports asynchronous (non-blocking) and synchronous (blocking) mode of operation.
- Supports both DMA and non-DMA modes.
- The SDSPI driver is capable of interacting to the SD Card either via SPI PLIB interface or via SPI Driver interface, selectable in MHC configuration of the SDSPI driver.
- The driver API interface can be used directly by the application to read/write directly from the SD Card Or the SDSPI driver can be used with a file system.

SD Card (SPI) Driver Library

Connect PLIB SERCOM
through SPI Driver
or
Direct to PLIB SERCOM



Always to use software chip select of SPI SD Card Driver

SD Card (SPI) Driver Library

API functions of the SD Card (SPI) Driver Library.

- **System Functions**

DRV_SDSPI_Initialize	Initializes the SD Card driver.
DRV_SDSPI_Status	Provides the current status of the SDSPI driver module.
DRV_SDSPI_Tasks	Maintains the driver's state machine.

- **Core Client Functions**

DRV_SDSPI_Open	Opens the specified SD Card driver instance and returns a handle to it.
DRV_SDSPI_Close	Closes an opened-instance of the SD Card driver.
DRV_SDSPI_EventHandlerSet	Allows a client to identify an event handling function for the driver to call back when queued operation has completed. For the synchronous SDSPI driver, the event handler is only used by the file system.

- **Data Transfer Functions**

DRV_SDSPI_AsyncRead	Reads blocks of data from the specified block address of the SD Card.
DRV_SDSPI_SyncRead	Reads blocks of data from the specified block address of the SD Card.
DRV_SDSPI_Read	This routine provides interface to the file system to perform a media read operation in synchronous mode of the SDSPI driver.
DRV_SDSPI_SyncWrite	Writes blocks of data starting at the specified address of the SD Card.
DRV_SDSPI_AsyncWrite	Writes blocks of data starting at the specified address of the SD Card.
DRV_SDSPI_Write	This routine provides interface to the file system to perform a media operation in synchronous mode of the SDSPI driver.

SD Card (SPI) Driver Library

API functions of the SD Card (SPI) Driver Library. (cont.)

- Data Transfer Functions

DRV_SDSPI_CommandStatusGet

Gets the current status of the command. For synchronous mode, the command status is only used by the file system.

- Status Functions

DRV_SDSPI_IsAttached

Returns the physical attach status of the SD Card.

DRV_SDSPI_IsWriteProtected

Returns the write protect status of the SD Card.

DRV_SDSPI_GeometryGet

Returns the geometry of the device.

- Data Types and Constants

DRV_SDSPI_COMMAND_HANDLE

This is type DRV_SDSPI_COMMAND_HANDLE.

DRV_SDSPI_COMMAND_STATUS

This is type DRV_SDSPI_COMMAND_STATUS.

DRV_SDSPI_EVENT

Identifies the possible events that can result from a request.

DRV_SDSPI_EVENT_HANDLER

Pointer to a SDSPI Driver Event handler function

DRV_SDSPI_COMMAND_HANDLE_INVALID

SDSPI Driver's Invalid Command Handle.

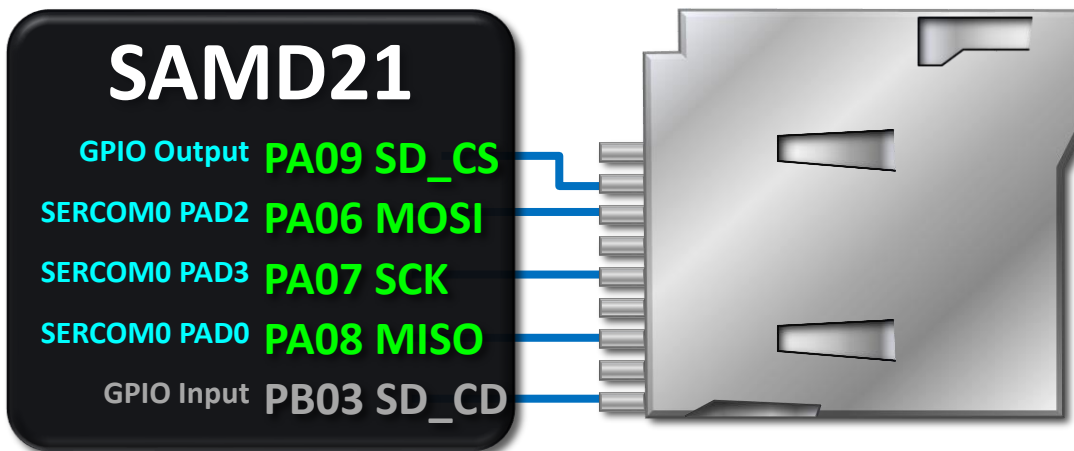
Lab 18

Manage SPI SD card without File System

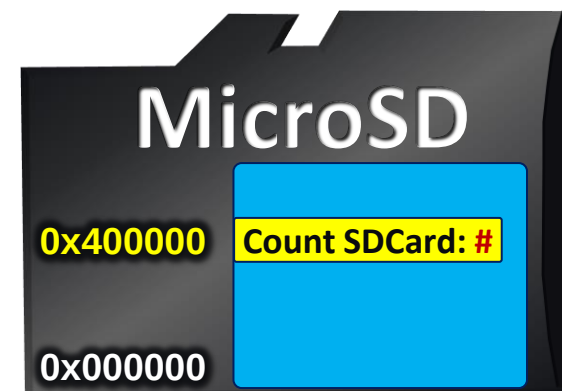
SPI Driver - SD Card Driver Library

⚙️ Lab18 Manage SPI SD card without File System

- Add an **SPI Driver Instance 2** and attach the **SD Card (SPI) Driver** and **PLIB SERCOM0 SPI** to access a **SPI SD Card** without File System.
- The pins define and configuration are same to **Lab7** we did.
- Try store “**Count SDCard: #**” to **block 0x2000** of SD Card , the ‘**#**’ is count number start from **1**.
- Reset EVB to read “**Count SDCard: #**” out and increase the count ‘**#**’ then store it back to SD Card.

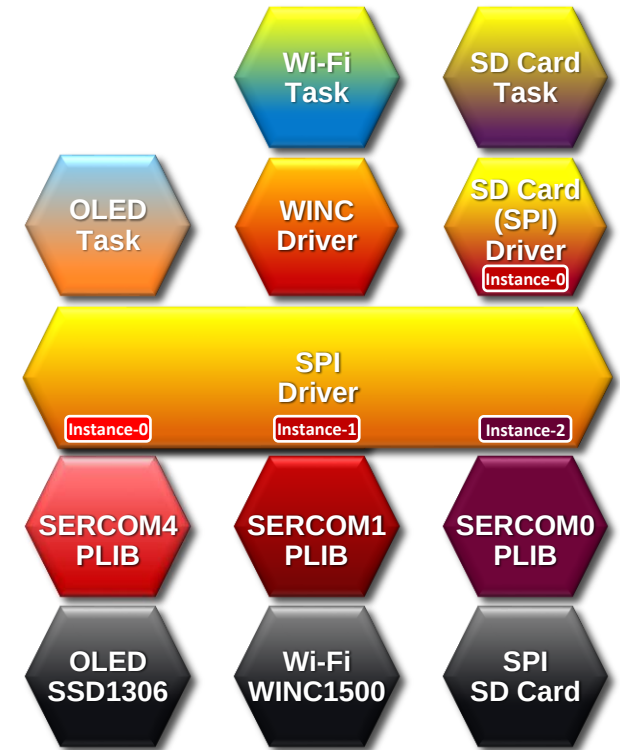
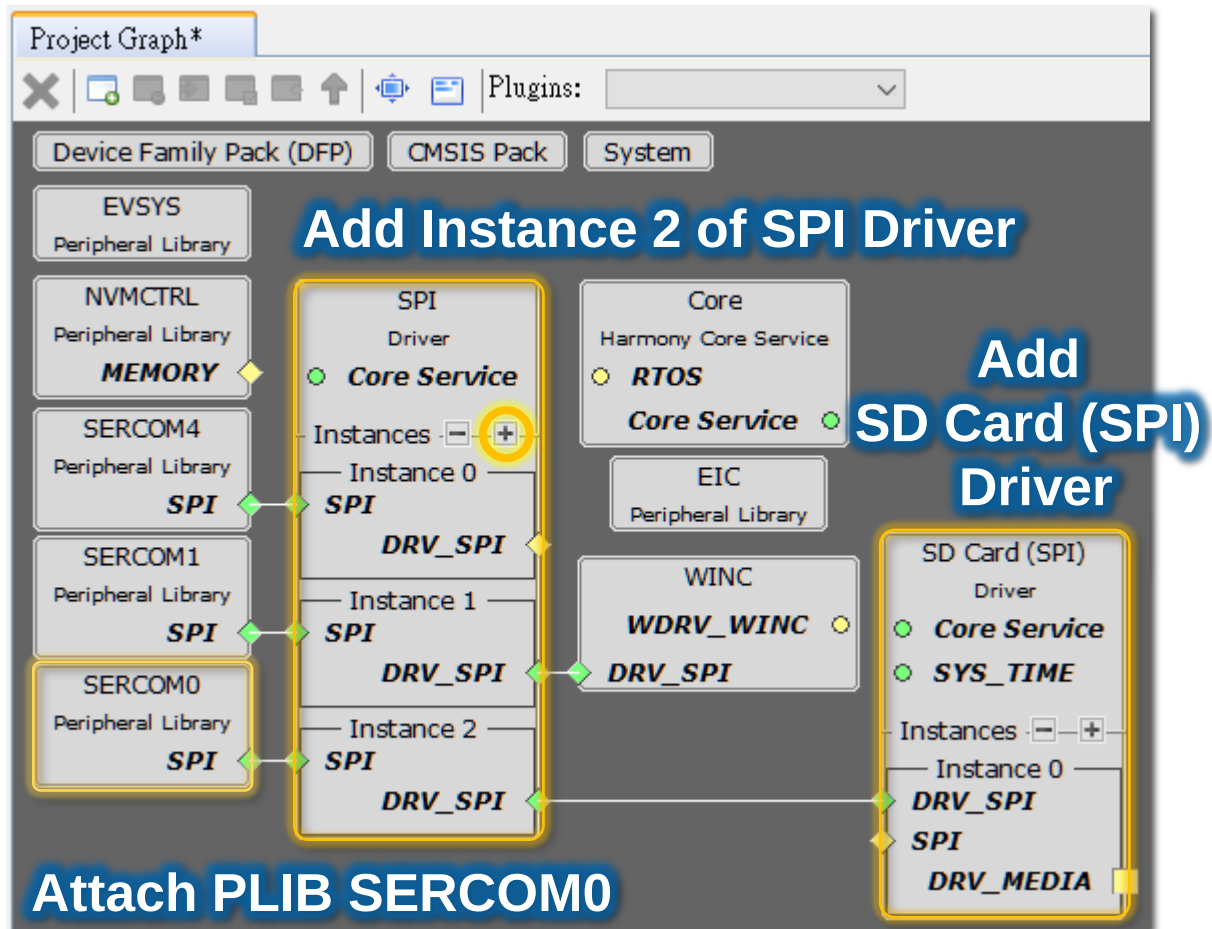


Block Size is **512 bytes**
Store Block is **0x2000**
Start Address is **0x400000**



⚙️ Lab18 Manage SPI SD card without File System

- Add **SPI Driver Instance 2**
- Attach the **SD Card (SPI) Driver** and **PLIB SERCOM0 SPI** for SPI SD Card communication.



⚙️ Lab18 Manage SPI SD card without File System

- Set **Chip Select** pin to **GPIO PA09** in **SD Card Driver** configuration.

The image shows the SAMD21 configuration tool interface. Two configuration windows are open:

- Configuration Options * SPI:** This window has a blue header "Enable DMA in SPI Driver". The "Use DMA for Transmit and Receive?" checkbox is checked. Other settings include PLIB Used (SERCOM0), Number of Clients (1), Transfer Queue Size (4), DMA Channel For Transmit (4), and DMA Channel For Receive (5).
- Configuration Options * SD Card (SPI):** This window has a red header "Set Chip Select pin to PA09". The "Chip Select Pin" dropdown is set to "PA09". Other settings include SPI Driver Instance Used (2), Number of Clients (1), SD Card Speed (Hz) (5,000,000), Transfer Queue Size (4), Card Detection Method (Use Polling), Polling Interval (ms) (1,000), and File system for SDSPI Driver Enabled (unchecked).

The main configuration area shows a block diagram of the SAMD21 components. The SPI Driver is configured with three instances (0, 1, 2). Instance 2 is highlighted with a green box. The SD Card (SPI) Driver is also configured with Instance 0, which is also highlighted with a green box. The SD Card (SPI) Driver is connected to the SPI Driver Instance 2.

SAMD21

- GPIO Output **PA09 SD_CS**
- SERCOM0 PAD2 **PA06 MOSI**
- SERCOM0 PAD3 **PA07 SCK**
- SERCOM0 PAD0 **PA08 MISO**
- GPIO Input **PB03 SD_CD**

⚙️ Lab18 Manage SPI SD card without File System

- Configure **SERCOM0 SPI** pads and keep default **disable** hardware **Slave Select**.

The image shows the SAMD21 configuration tool interface. On the left, the 'Peripheral Library' lists various components, with 'SERCOM0 SPI' highlighted in a green box. Above this, a summary box for 'SAMD21' lists GPIO outputs (PA09 SD_CS, PA06 MOSI, PA07 SCK, PA08 MISO) and inputs (PB03 SD_CD). The main configuration window, titled 'Configuration Options', shows the 'SERCOM0' settings. The 'Select SERCOM Operation Mode' is set to 'SPI Master'. The 'Enable SPI Master Hardware Slave Select' checkbox is unchecked, highlighted with a red box and the text 'Don't use Hardware Slave Select'. The 'SPI Data Out Pad' is set to 'DO on PAD[2], SCK on PAD[3] and SS on PAD[1]'. The 'SPI Data In Pad Selection' is set to 'SERCOM PAD[0]'. Other settings include 'Enable Interrupts?' (checked), 'Enable operation in Standby mode' (unchecked), 'SPI Data Order' (MSB is transferred first), 'SPI Speed in Hz' (1,000,000), 'SPI Data Character Size' (8 bits), 'SPI Clock Phase' (The data is sampled on a leading SCK edge and changed on a trailing SCK edge), 'SPI Clock Polarity' (SCK is low when idle), and 'SPI Receiver Enable' (checked). The bottom of the window shows the 'Instances' section with 'Instance 0' selected, showing 'DRV_SPI' and 'SPI' components.

SAMD21
GPIO Output PA09 SD_CS
SERCOM0 PAD2 PA06 MOSI
SERCOM0 PAD3 PA07 SCK
SERCOM0 PAD0 PA08 MISO
GPIO Input PB03 SD_CD

Configuration Options

SERCOM0
Select SERCOM Operation Mode SPI Master

Enable Interrupts ? ☒
Enable operation in Standby mode ☐
SPI Data Out Pad DO on PAD[2], SCK on PAD[3] and SS on PAD[1]
SPI Data In Pad Selection SERCOM PAD[0]
SPI Data Order MSB is transferred first
SPI Speed in Hz 1,000,000
SPI Data Character Size 8 bits
SPI Clock Phase The data is sampled on a leading SCK edge and changed on a trailing SCK edge
SPI Clock Polarity SCK is low when idle
Enable SPI Master Hardware Slave Select ☐ **Don't use Hardware Slave Select**
SPI Receiver Enable ☒
SPI Transfer Mode 0 is Selected

Instances
Instance 0
DRV_SPI
SPI
DRV_MEDIA

✖ Lab18 Manage SPI SD card without File System

- Configure **SERCOM0-SPI** pads, **PA06(PAD2)**, **PA07(PAD3)**, **PA08(PAD0)**.
- Configure GPIOs : **PA09(Output, Latch High, Custom Name is optional)**.

Pin Number	Pin ID	Custom Name	Function	Mode
11	PA06		SERCOM0_PAD2	Digital
12	PA07	Custom Name is optional	SERCOM0_PAD3	Digital
13	PA08		SERCOM0_PAD0	Digital
14	PA09	SD_CS	GPIO	Digital

SD Card (SPI) configuration:

- SPI Driver Instance Used: 2
- Number of Clients: 1
- SD Card Speed (Hz): 5,000,000
- Transfer Queue Size: 4
- Card Detection Method: Use Polling
- Polling Interval (ms): 1,000
- File system for SDSPI Driver Enabled: ☐
- Chip Select Pin: PA09

Module	Function	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	7			
GPIO	GPIO																																																
SERCOM0	SERCOM0_PAD0																																																
	SERCOM0_PAD1																																																
	SERCOM0_PAD2																																																
	SERCOM0_PAD3																																																

SAMD21

- GPIO Output PA09 SD_CS
- SERCOM0 PAD2 PA06 MOSI
- SERCOM0 PAD3 PA07 SCK
- SERCOM0 PAD0 PA08 MISO

✖ Lab18 Manage SPI SD card without File System

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**7**” and given the Application Name to “**app_drv_sdcard**”.

The screenshot displays the Microchip Harmony IDE interface. On the left, the 'Project Graph*' window shows a block diagram of the system components. The 'Core' block, labeled 'Harmony Core Service', is highlighted with a green box. It contains sub-components like 'RTOS' and 'Core Service'. The 'SPI' driver is also visible, with multiple instances (Instance 0, Instance 1, Instance 2) and their corresponding 'DRV_SPI' blocks. The 'SERCOM' blocks (SERCOM4, SERCOM1, SERCOM0) are connected to the SPI driver. The 'WINC' block is also present, with a 'WDRV_WINC' component. The 'Configuration Options' dialog is open on the right, showing the 'Core' configuration. The 'Generate Harmony Application Files' option is checked. Under 'Application Configuration', the 'Number of Applications' is set to 7, highlighted with a red box. The 'Application Name' for Application 6 is set to 'app_drv_sdcard', highlighted with a blue box. A red text overlay reads 'Increase Number of Application' and a blue text overlay reads 'Rename file to "app_drv_sdcard"'. The dialog also shows options for 'Application 0 Configuration' through 'Application 6 Configuration', and 'Enable Application 6 Configuration' is checked. At the bottom, there are options for 'Generate Harmony Driver Common Files', 'Generate Harmony System Service Common Files', 'Generate Harmony System Media Files', and 'Enable System Interrupt', all of which are checked.

Project Graph*

Device Family Pack (DFP) CMSIS Pack System

EVSYSPeripheral Library

NVMCTRLPeripheral Library
MEMORY

SERCOM4Peripheral Library
SPI

SERCOM1Peripheral Library
SPI

SERCOM0Peripheral Library
SPI

SPI Driver

Core Service

Instances - +

Instance 0
SPI
DRV_SPI

Instance 1
SPI
DRV_SPI

Instance 2
SPI
DRV_SPI

Core
Harmony Core Service
● **RTOS**
● **Core Service**

EICPeripheral Library

WINC
WDRV_WINC

DRV_SPI

DRV_MEDIA

Configuration Options

Increase Number of Application

Core

Generate Harmony Application Files ☒

Application Configuration

Number of Applications 7

Application 0 Configuration

Application 1 Configuration

Application 2 Configuration

Application 3 Configuration

Application 4 Configuration

Application 5 Configuration

Application 6 Configuration

Enable Application 6 Configuration ☒

Application Name app_drv_sdcard

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

Generate Harmony Driver Common Files ☒

Generate Harmony System Service Common Files ☒

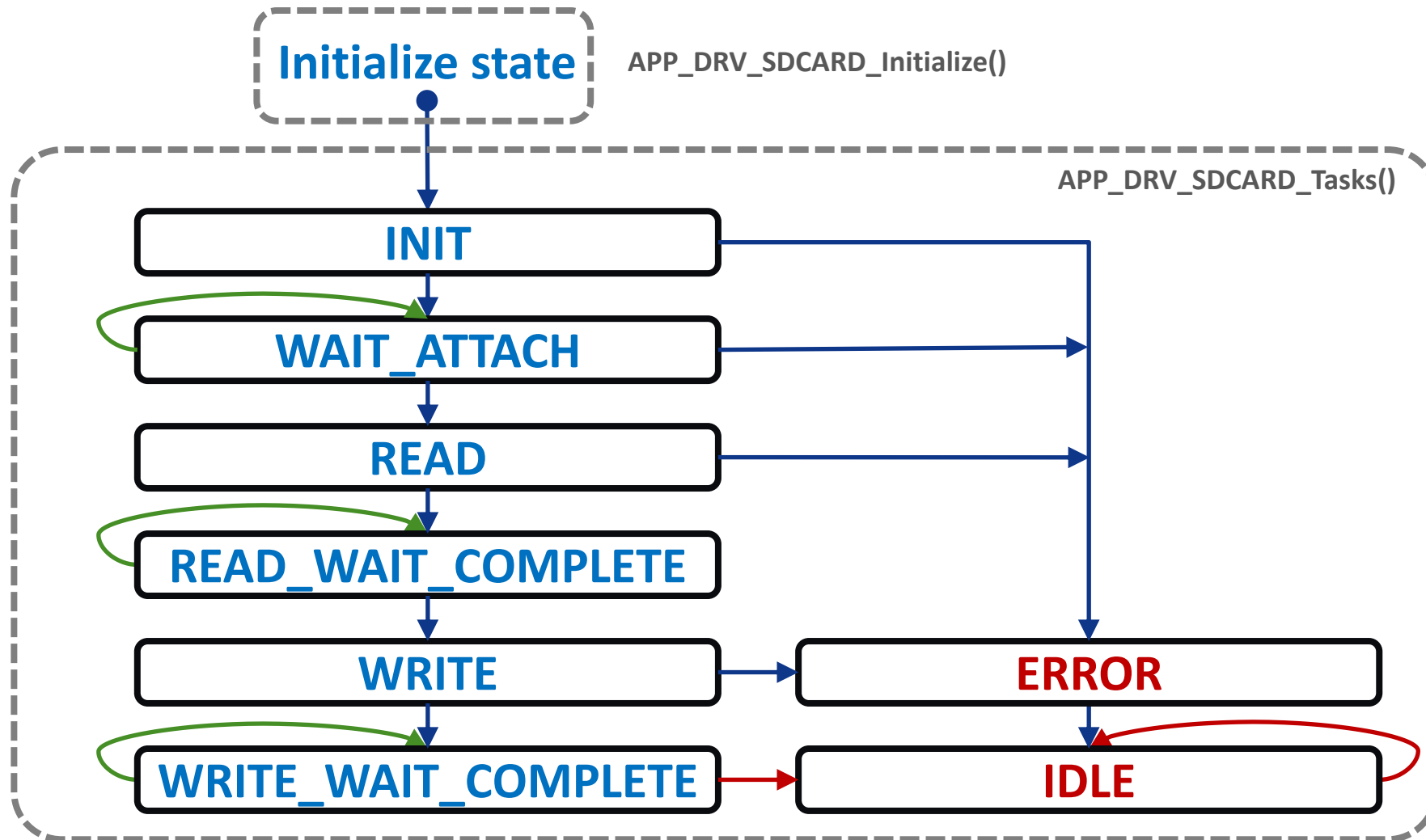
Generate Harmony System Media Files ☒

Enable System Interrupt ☒

Rename file to "app_drv_sdcard"

⚙️ Lab18 Manage SPI SD card without File System

- The new state machine we wish to complete as below.



✖ Lab18 Manage SPI SD card without File System

- Please comment out the `WDRV_WINC_Tasks` and `APP_DRV_WINC_Tasks` in `task.c`,
If you **have not** an external WINC1500 EVB ([WiFi 7 Click](#)) from [Lab17](#).

```
void SYS_Tasks ( void )  
{  
  
    /* Maintain Device Drivers */  
  
    DRV_SDSPI_Tasks(sysObj.drivSDSPI0);
```

Comment out if no WINC1500 // WDRV_WINC_Tasks(sysObj.drivWifiWinc);

```
    /* Maintain Middleware & Other Libraries */
```

```
    /* Maintain the application's state machine. */
```

```
    ...  
    /* Call Application task APP_DRV_I2C1. */  
    APP_DRV_I2C1_Tasks();
```

```
    /* Call Application task APP_DRV_I2C2. */  
    APP_DRV_I2C2_Tasks();
```

```
    /* Call Application task APP_DRV_SPI1. */  
    APP_DRV_SPI1_Tasks();
```

```
    /* Call Application task APP_DRV_WINC. */
```

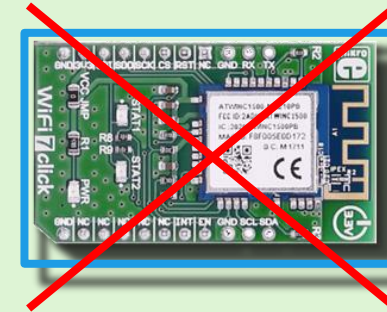
Comment out if no WINC1500 // APP_DRV_WINC_Tasks();

```
    /* Call Application task APP_DRV_SDCARD. */  
    APP_DRV_SDCARD_Tasks();
```

```
    ...  
}
```

task.c

MIKORE
WIFI 7 CLICK



⚙️ Lab18 Manage SPI SD card without File System

- Add states in “app_drv_sdcard.h”.

```
#ifndef _APP_DRV_SDCARD_H
#define _APP_DRV_SDCARD_H
```

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

```
typedef enum
{
```

Keep this line APP_DRV_SDCARD_STATE_INIT=0,

Comment out this line // APP_DRV_SDCARD_STATE_SERVICE_TASKS,
APP_DRV_SDCARD_STATE_WAIT_ATTACH,
APP_DRV_SDCARD_STATE_READ,
APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE,
APP_DRV_SDCARD_STATE_WRITE,
APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE,
APP_DRV_SDCARD_STATE_ERROR,
APP_DRV_SDCARD_STATE_IDLE,

```
} APP_DRV_SDCARD_STATES;
```

```
typedef struct
{
```

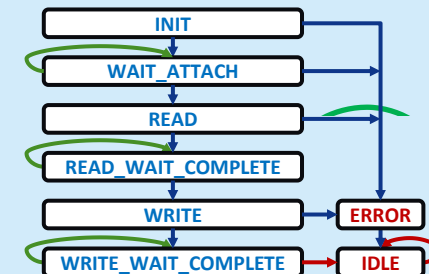
```
    APP_DRV_SDCARD_STATES state;
```

```
} APP_DRV_SDCARD_DATA;
```

```
void APP_DRV_SDCARD_Initialize ( void );
```

app_drv_sdcard.h

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



✖ Lab18 Manage SPI SD card without File System

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

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WAIT_ATTACH;
            break;

        case APP_DRV_SDCARD_STATE_WAIT_ATTACH:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
            break;

        case APP_DRV_SDCARD_STATE_READ:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE;
            break;

        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE;
            break;

        case APP_DRV_SDCARD_STATE_WRITE:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE;
            break;

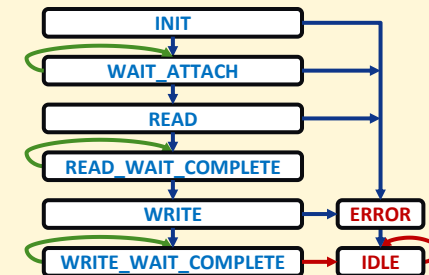
        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;
            break;

        case APP_DRV_SDCARD_STATE_ERROR:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;
            break;

        case APP_DRV_SDCARD_STATE_IDLE:
            break;
    }
}
```

app_drv_sdcard.c

Overwrite original example code



✂ Lab18 Manage SPI SD card without File System

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

Keep this line

```
#include "app_drv_sdcard.h"
#include "app_drv_usart1.h"
#include "driver/sdspi/drv_sdspi.h"
#include <stdio.h>
#include <string.h>
```

app_drv_sdcard.c

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

```
#define SDCARD_BLOCK_START          0x2000
#define SDCARD_BUFFER_SIZE          (4096U)
#define SDCARD_GEOMETRY_TABLE_READ_ENTRY (0)
#define SDCARD_GEOMETRY_TABLE_WRITE_ENTRY (1)
#define SDCARD_GEOMETRY_TABLE_ERASE_ENTRY (2)
SYS_MEDIA_GEOMETRY *SDCardGeometry = NULL;
int SDCardReadBlocks;
int SDCardWriteBlocks;
int SDCardEraseBlocks;
```

```
DRV_HANDLE SDCardHandle;
DRV_SDSPI_COMMAND_HANDLE SDCardReadHandle;
DRV_SDSPI_COMMAND_HANDLE SDCardWriteHandle;
volatile int SDCardWriteComplete = false;
volatile int SDCardReadComplete = false;
char SDCardWriteBuf[SDCARD_BUFFER_SIZE];
char SDCardReadBuf[SDCARD_BUFFER_SIZE];
int SDCardResetCount = 1;
```

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

**For SD Card Driver
The Geometry information
is necessary for operation**

✂ Lab18 Manage SPI SD card without File System

- Implement **Transfer Callback** for **SD Card Driver** Read and Write.

```
*/
APP_DRV_SDCARD_DATA app_drv_sdcardData;

// *****
// *****
// Section: Application Callback Functions
// *****
// *****
void SDCardTransferHandler( SYS_MEDIA_BLOCK_EVENT event,
                           SYS_MEDIA_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context )
{
    switch( event )
    {
        case SYS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE:
            if (commandHandle == SDCardReadHandle)
            { SDCardReadComplete = true; }
            else if(commandHandle == SDCardWriteHandle)
            { SDCardWriteComplete = true; }
            break;

        case SYS_MEDIA_EVENT_BLOCK_COMMAND_ERROR:
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
            break;
    }
}

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

app_drv_sdcard.c

SD Card Driver Transfer Callback

⚙️ Lab18 Manage SPI SD card without File System

- Open Driver, register callback in state **INIT** and handling **ERROR**.

```
void APP_DRV_SDCARD_Tasks ( void )  
{
```

app_drv_sdcard.c

```
    switch ( app_drv_sdcardData.state )  
    {
```

Open SD Card Driver to get Handle

```
    case APP_DRV_SDCARD_STATE_INIT:
```

```
        SDCardHandle = DRV_SDSPI_Open(DRV_SDSPI_INDEX_0, DRV_IO_INTENT_READWRITE);
```

```
        if(SDCardHandle == DRV_HANDLE_INVALID)
```

```
        {
```

```
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
```

```
            break;
```

```
        }
```

Register Transfer Callback

```
        DRV_SDSPI_EventHandlerSet(SDCardHandle, SDCardTransferHandler, 0);
```

Keep this line

```
        app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WAIT_ATTACH;  
        break;
```

```
    case APP_DRV_SDCARD_STATE_WAIT_ATTACH:
```

```
        app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
```

```
        break;
```

```
    ...
```

Handling Error state

```
    case APP_DRV_SDCARD_STATE_ERROR:
```

```
        USART1printf("DRV_SDCARD error!\r\n");
```

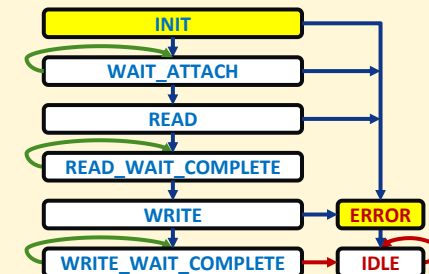
Keep this line

```
        app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;  
        break;
```

```
    case APP_DRV_SDCARD_STATE_IDLE:
```

```
        break;
```

```
    ...
```



⚙️ Lab18 Manage SPI SD card without File System

- Use the polling method to detect SD Card in state **WAIT_ATTACH**.

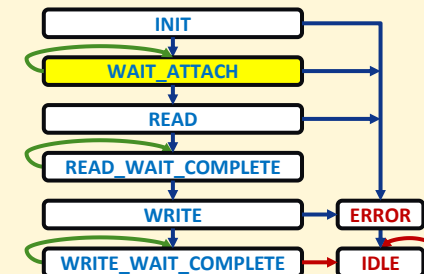
```
void APP_DRV_SDCARD_Tasks ( void )  
{  
    switch ( app_drv_sdcardData.state )  
    {  
        case APP_DRV_SDCARD_STATE_INIT: ...  
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH:  
            if( DRV_SDSPI_IsAttached(SDCardHandle) == false )  
                break;  
  
            USART1printf("\r\nSDCard attached!\r\n");  
  
            SDCardGeometry = DRV_SDSPI_GeometryGet(SDCardHandle);  
            if (SDCardGeometry == NULL)  
            {  
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;  
                break;  
            }  
            SDCardReadBlocks  = (SDCARD_BUFFER_SIZE / \  
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_READ_ENTRY].blockSize);  
            SDCardWriteBlocks = (SDCARD_BUFFER_SIZE / \  
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_WRITE_ENTRY].blockSize);  
            SDCardEraseBlocks = (SDCARD_BUFFER_SIZE / \  
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);  
  
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;  
            break;  
  
        case APP_DRV_SDCARD_STATE_READ:  
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE: ...  
        case APP_DRV_SDCARD_STATE_WRITE: ...  
        ...  
    }  
}
```

app_drv_sdcard.c

Using Polling method to Check SD Card attach status

Get SD Card Geometry

Keep this line app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
break;



✂ Lab18 Manage SPI SD card without File System

- Read **SDCardReadBlocks** from address **SDCARD_BLOCK_START**.

```
void APP_DRV_SDCARD_Tasks ( void )
{
    switch ( app_drv_sdcardData.state )
    {
        case APP_DRV_SDCARD_STATE_INIT: ...
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH:
            ...
            SDCardEraseBlocks = (SDCARD_BUFFER_SIZE / \
                SDCardGeometry->geometryTable[SDCARD_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);

            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ;
            break;
```

app_drv_sdcard.c

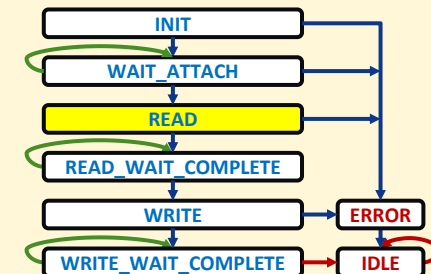
Read one Block from SD Card

```
        case APP_DRV_SDCARD_STATE_READ:
            DRV_SDSPI_AsyncRead(SDCardHandle, &SDCardReadHandle,
                                SDCardReadBuf, SDCARD_BLOCK_START, SDCardReadBlocks);
            if(SDCardReadHandle == DRV_SDSPI_COMMAND_HANDLE_INVALID)
            {
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;
                break;
            }
```

Keep this line app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE;

```
        break;

        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_SDCARD_STATE_WRITE: ...
        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE: ...
        case APP_DRV_SDCARD_STATE_ERROR: ...
        case APP_DRV_SDCARD_STATE_IDLE: ...
        ...
```



✂ Lab18 Manage SPI SD card without File System

- Increase "**Count**" of read content or **output SD Card Geometry** if it's first time to write.

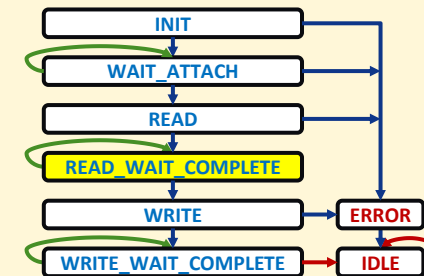
```
void APP_DRV_SDCARD_Tasks ( void )  
{  
    switch ( app_drv_sdcardData.state )  
    {  
        case APP_DRV_SDCARD_STATE_INIT: ...  
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH: ...  
        case APP_DRV_SDCARD_STATE_READ: ...  
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE:  
            if(SDCardReadComplete == true)  
            {  
                SDCardReadComplete = false;  
  
                if( !strncmp((char*)SDCardReadBuf, "Count SDCard: ", 14) )  
                {  
                    USART1printf("%s\r\n", SDCardReadBuf);  
                    sscanf(SDCardReadBuf, "Count SDCard: %d", &SDCardResetCount);  
                    SDCardResetCount = (SDCardResetCount+1)%10;  
                }  
                else  
                {  
                    USART1printf("Geometry SDCard: Buff(%d), R(%d), W(%d), E(%d)\r\n",  
                                SDCARD_BUFFER_SIZE,  
                                SDCardReadBlocks, SDCardWriteBlocks, SDCardEraseBlocks);  
                }  
  
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE;  
            }  
            break;  
  
        case APP_DRV_SDCARD_STATE_WRITE: ...  
        ...  
    }  
}
```

app_drv_sdcard.c

Output current count and increase it

Output Geometry if nothing to be wrote

Keep this line



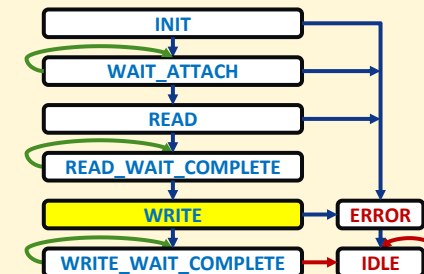
✖ Lab18 Manage SPI SD card without File System

- Write back content to target address **SDCARD_BLOCK_START**.

```
void APP_DRV_SDCARD_Tasks ( void )  
{  
    switch ( app_drv_sdcardData.state )  
    {  
        case APP_DRV_SDCARD_STATE_INIT: ...  
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH: ...  
        case APP_DRV_SDCARD_STATE_READ: ...  
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE:  
            ...  
            break;  
  
        case APP_DRV_SDCARD_STATE_WRITE:  
            sprintf( SDCardWriteBuf, "Count SDCard: %d", SDCardResetCount);  
            DRV_SDSPI_AsyncWrite(SDCardHandle, &SDCardWriteHandle,  
                                SDCardWriteBuf, SDCARD_BLOCK_START, SDCardWriteBlocks);  
  
            if(SDCardWriteHandle == DRV_SDSPI_COMMAND_HANDLE_INVALID)  
            {  
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_ERROR;  
            }  
            else  
            {  
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE;  
            }  
            break;  
  
        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE: ...  
        case APP_DRV_SDCARD_STATE_ERROR: ...  
        case APP_DRV_SDCARD_STATE_IDLE:  
            break  
        ...  
    }  
}
```

Write count to target Block of SD Card

Keep this line



⚙️ Lab18 Manage SPI SD card without File System

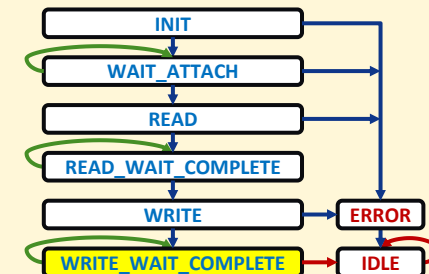
- Close **handle** while write completed and go state **IDLE** forever.

```
void APP_DRV_SDCARD_Tasks ( void )  
{  
    switch ( app_drv_sdcardData.state )  
    {  
        case APP_DRV_SDCARD_STATE_INIT: ...  
        case APP_DRV_SDCARD_STATE_WAIT_ATTACH: ...  
        case APP_DRV_SDCARD_STATE_READ: ...  
        case APP_DRV_SDCARD_STATE_READ_WAIT_COMPLETE: ...  
        case APP_DRV_SDCARD_STATE_WRITE:  
            ...  
        }  
        else  
        {  
            app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE;  
        }  
        break;  
  
        case APP_DRV_SDCARD_STATE_WRITE_WAIT_COMPLETE:  
            if(SDCardWriteComplete == true)  
            {  
                SDCardWriteComplete = false;  
  
                DRV_SDSPI_Close( SDCardHandle );  
  
                app_drv_sdcardData.state = APP_DRV_SDCARD_STATE_IDLE;  
            }  
            break;  
  
        case APP_DRV_SDCARD_STATE_ERROR: ...  
        case APP_DRV_SDCARD_STATE_IDLE: ...  
        ...  
    }
```

app_drv_sdcard.c

Go state IDLE after finished

Keep this line



⚙️ Lab18 Manage SPI SD card without File System

- **Put in SD Card first !!!**
- Compiler and programming your project to EVB.
- Geometry will output while first time to reboot because there has nothing wrote at **Block Start Address 0x400000**.
- The **“Count”** will increase while each time you to press **Reset** button on the EVB.

```
COM3 - Tera Term VT
File Edit Setup Control Window Help

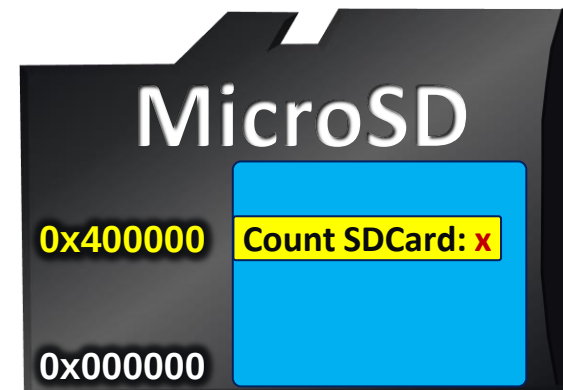
1st Boot → ...
SDCard attached!
Geometry SDCard: Buffer(4096), R(8), W(8), E(8)

Reset → ...
SDCard attached!
Count SDCard: 1

Reset → ...
SDCard attached!
Count SDCard: 2

Reset → ...
SDCard attached!
Count SDCard: 3
```

Block Size is **512 bytes**
Store Block is **0x2000**
Start Address is **0x40000**



⚙️ Lab18 Manage SPI SD card without File System

- **Remove SD Card firstly** and press Reset button on the EVB then put SD Card in.
- Why detect SD Card is slowly ? even we to use **Polling Method** for SD Card Detect.
 - The **SERCOM4-SPI** share with OLED to use **blocking transfer** for GUI update in SPI1 task. It will affect the SD Card Driver to **miss polling** of SD Card detection.
 - Let's **add 200ms** delay between each update in state **UPDATE** of SPI1 task.

```
void APP_DRV_SPI1_Tasks ( void )
{
    switch ( app_drv_spi1Data.state )
    {
        ...
        case APP_DRV_SPI1_STATE_SPLASH_WAIT_COMPLETE:
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == true )
            {
                LCM_Clear();

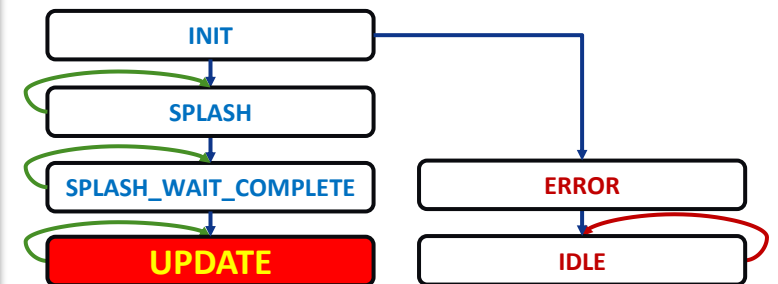
                SYS_TIME_DelayMS(200, &SplashTimerHandle);

                app_drv_spi1Data.state = APP_DRV_SPI1_STATE_UPDATE;
            }
            break;

        case APP_DRV_SPI1_STATE_UPDATE:
            if( SYS_TIME_DelayIsComplete(SplashTimerHandle) == false )
                break;
            SYS_TIME_DelayMS(200, &SplashTimerHandle);

            sprintf( OutString, "LCM: Hello World" );
            ...
    }
}
```

app_drv_spi1.c



MEMORY Driver Library

Harmony Driver

MEMORY Driver Library

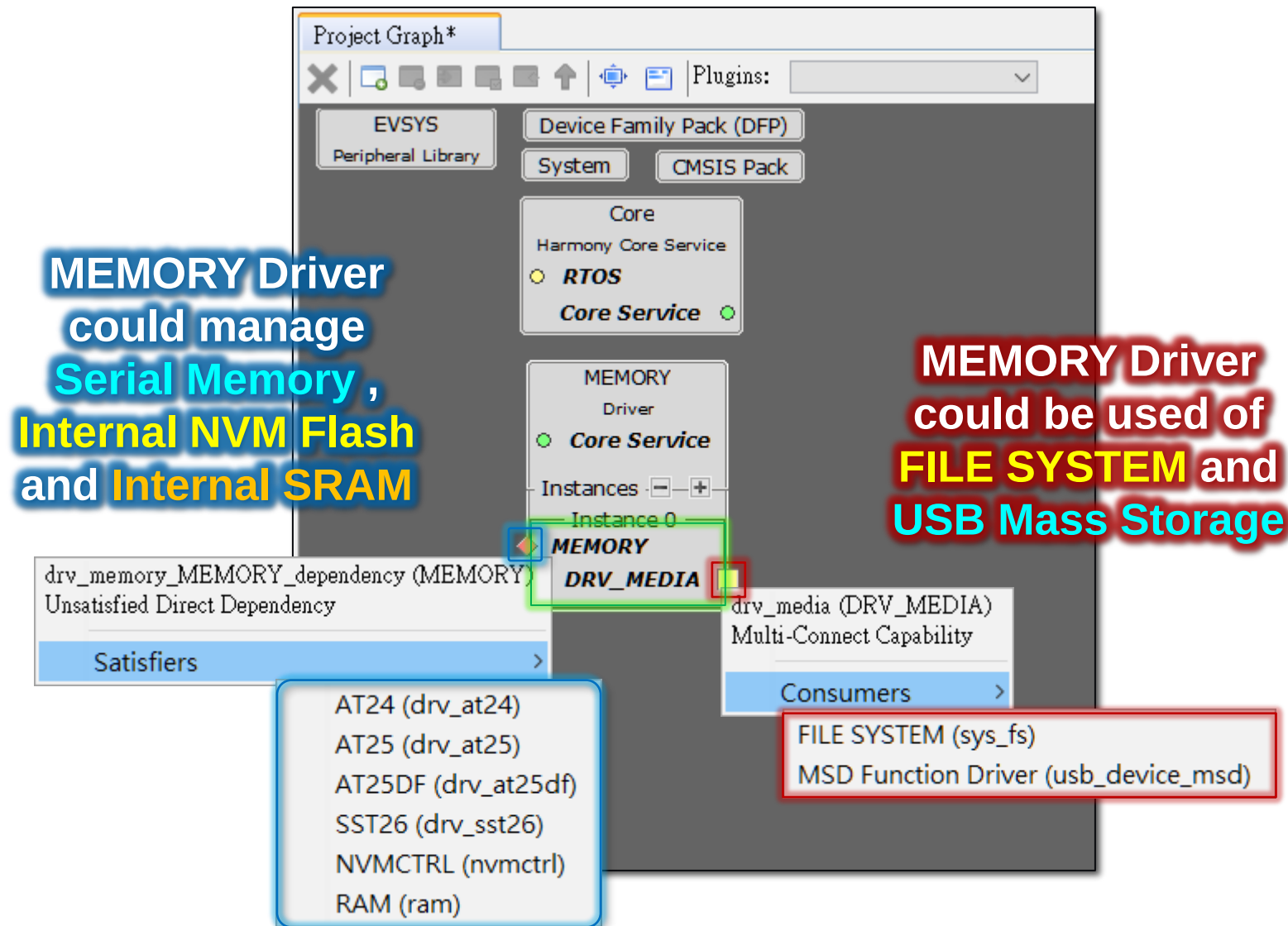
- **Introduction**

The Memory driver is a multi-client multi-instance buffer model-based block driver interface which can be used to communicate with various media devices.

- **Key Features of Memory Driver:**

- It works in both Asynchronous and Synchronous modes of operation.
- It works in both Bare Metal and RTOS environment.
- It can be used to communicate with Different Media's via common Media Interface.
- Media which can be accessed directly by Peripheral Libraries (NVM).
- Media which can be accessed by protocol drivers which in-turn connect to respective peripheral Libraries (SST26 Driver --> QSPI peripheral library)

MEMORY Driver Library



MEMORY Driver Library

API functions of the MEMORY Driver library.

- **System Functions**

DRV_MEMORY_Initialize

Initializes the Memory instance for the specified driver index

DRV_MEMORY_Status

Gets the current status of the Memory driver module.

DRV_MEMORY_Tasks

Maintains the Memory driver's internal state machine.

- **Core Client Functions**

DRV_MEMORY_Open

Opens the specified Memory driver instance and returns a handle to it

DRV_MEMORY_Close

Closes an opened-instance of the Memory driver

DRV_MEMORY_TransferHandlerSet

Allows a client to identify an event handling function for the driver to call back when queued operation has completed.

- **Block Operation Functions**

DRV_MEMORY_AsyncErase

Erase the specified number of memory blocks from the specified block start. Each block is equal to sector size of the memory device attached.

DRV_MEMORY_AsyncEraseWrite

Erase and Write blocks of data in the sectors where the block start belongs.

DRV_MEMORY_AsyncRead

Reads nblocks of data from the specified block start.

DRV_MEMORY_AsyncWrite

Writes nblocks of data starting at the specified block start.

DRV_MEMORY_SyncErase

Erase the specified number of memory blocks from the specified block start. Each block is equal to sector size of the memory device attached.

DRV_MEMORY_Erase

This routine provides interface to the file system to perform a media erase operation in synchronous mode of the Memory driver.

DRV_MEMORY_SyncEraseWrite

Erase and Write blocks of data in the sectors where the block start belongs.

MEMORY Driver Library

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

- **Block Operation Functions (cont.)**

DRV_MEMORY_EraseWrite	This routine provides interface to the file system to perform a media erase-write operation in synchronous mode of the Memory driver.
DRV_MEMORY_SyncRead	Reads nblock of data from the specified block start.
DRV_MEMORY_Read	This routine provides interface to the file system to perform a media read operation in synchronous mode of the Memory driver.
DRV_MEMORY_SyncWrite	Writes nblock of data starting at the specified block start.
DRV_MEMORY_Write	This routine provides interface to the file system to perform a media write operation in synchronous mode of the Memory driver.
DRV_MEMORY_CommandStatusGet	Gets the current status of the command.
DRV_MEMORY_TransferStatusGet	Gets the current status of the transfer request on attached device.

- **Media Interface Functions**

DRV_MEMORY_AddressGet	Returns the Memory media start address
DRV_MEMORY_GeometryGet	Returns the geometry of the memory device.
DRV_MEMORY_IsAttached	Returns the physical attach status of the Media.
DRV_MEMORY_IsWriteProtected	Returns the write protect status of the Memory.

MEMORY Driver Library

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

- Data Types and Constants

DRV_MEMORY_COMMAND_HANDLE	Handle to identify commands queued in the driver.
DRV_MEMORY_COMMAND_HANDLE_INVALID	This value defines the Memory Driver's Invalid Command Handle.
DRV_MEMORY_COMMAND_STATUS	Memory Driver command Status
DRV_MEMORY_DEVICE_INTERFACE	Memory Device API.
DRV_MEMORY_DEVICE_OPEN	Function pointer typedef to open the attached media
DRV_MEMORY_DEVICE_CLOSE	Function pointer typedef to close the attached media
DRV_MEMORY_DEVICE_SECTOR_ERASE	Function pointer typedef to erase a sector from attached media
DRV_MEMORY_DEVICE_STATUS	Function pointer typedef to get the status of the attached media
DRV_MEMORY_DEVICE_READ	Function pointer typedef to read from the attached media
DRV_MEMORY_DEVICE_PAGE_WRITE	Function pointer typedef to write a page to the attached media
DRV_MEMORY_DEVICE_EVENT_HANDLER_SET	Function pointer typedef to set event handler with attached media
DRV_MEMORY_DEVICE_GEOMETRY_GET	Function pointer typedef to get the Geometry details
DRV_MEMORY_DEVICE_TRANSFER_STATUS_GET	Function pointer typedef to get the transfer Status
DRV_MEMORY_EVENT	Identifies the possible events that can result from a request.
DRV_MEMORY_EVENT_HANDLER	Function pointer typedef for event handler to be sent to attached media
DRV_MEMORY_INIT	Memory Driver Initialization Data
DRV_MEMORY_TRANSFER_HANDLER	Pointer to a Memory Driver Event handler function
MEMORY_DEVICE_GEOMETRY	Memory Device Geometry Table.
MEMORY_DEVICE_TRANSFER_STATUS	Memory Device Transfer Status.

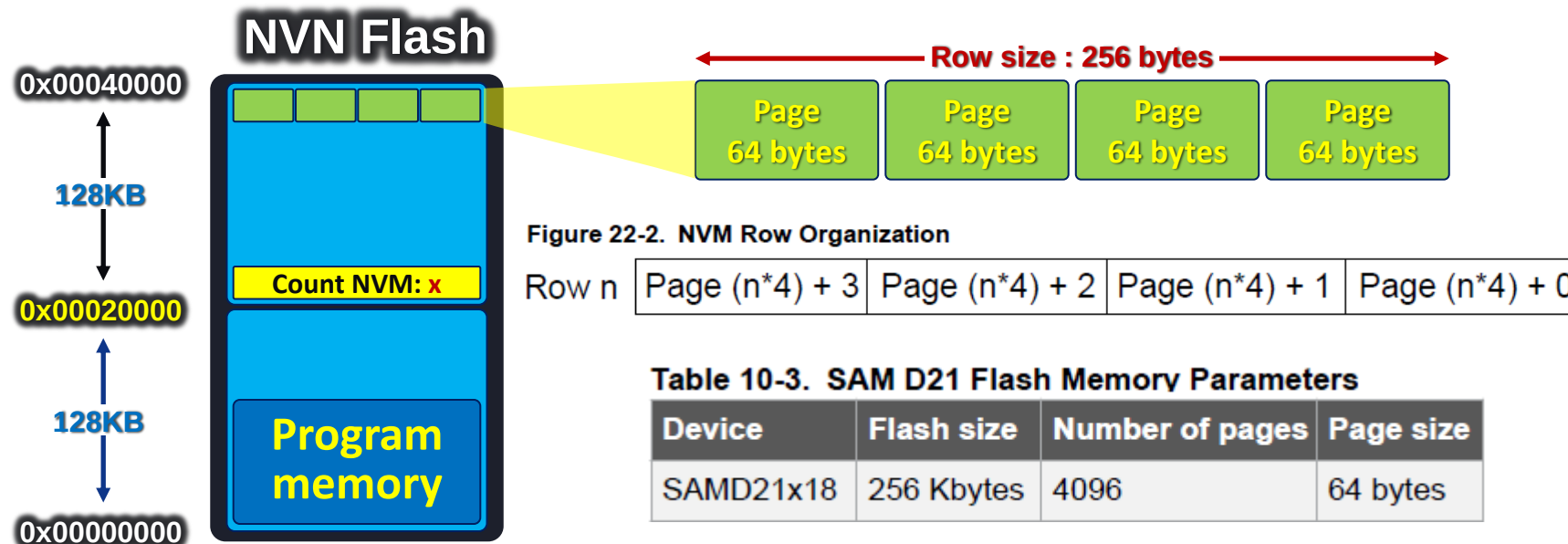
Lab 19

Manage NVM Flash without File System

MEMORY Driver Library

⚙️ Lab19 Manage NVM Flash without File System

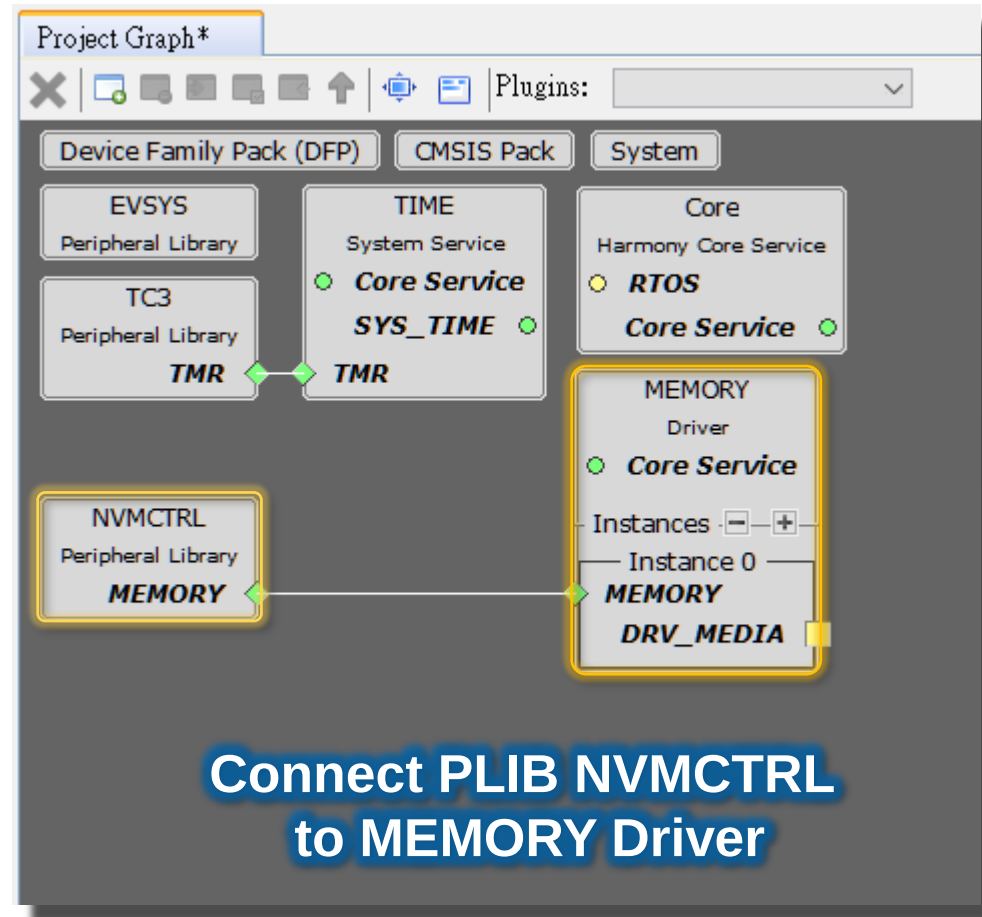
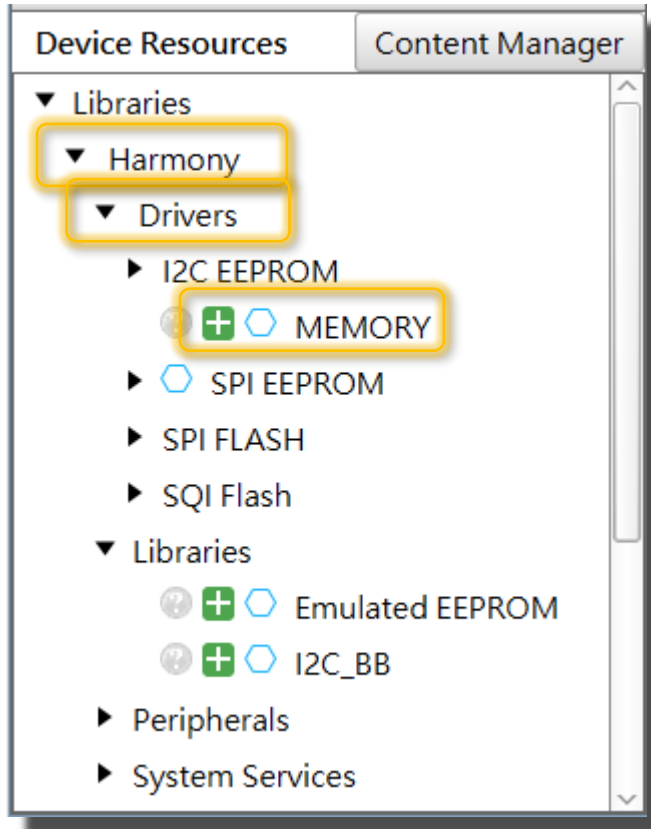
- Add **MEMORY Driver** and choose **PLIB NVMCTRL** as its media.
- The **PLIB NVMCTRL** already exist in project, please just connect it.
- Try store “**Count NVM: #**” to **Start Address 0x20000** of **NVM Flash**, the ‘#’ is count number start from **1**.
- Reset EVB to read “**Count NVM: #**” out and increase the count ‘#’ then store it back to **NVM Flash**.



⚙️ Lab19 Manage NVM Flash without File System

- Add **MEMORY Driver** and connect **PLIB NVMCTRL** to Driver.

Add MEMORY Driver



✖ Lab19 Manage NVM Flash without File System

- Default configuration of **MEMORY Driver** and **PLIB NVMCTRL**.

The image displays the Microchip Studio Project Graph and Configuration Options windows, illustrating the default configuration for the MEMORY Driver and NVMCTRL peripheral.

Project Graph: The graph shows the system components and their interconnections. The **MEMORY Driver** (Core Service) is connected to the **NVMCTRL** (Peripheral Library) via the **INSTANCE 0** instance. The **MEMORY Driver** is also connected to the **TC3** (Peripheral Library) via the **TMR** (Timer Module).

Configuration Options - MEMORY:

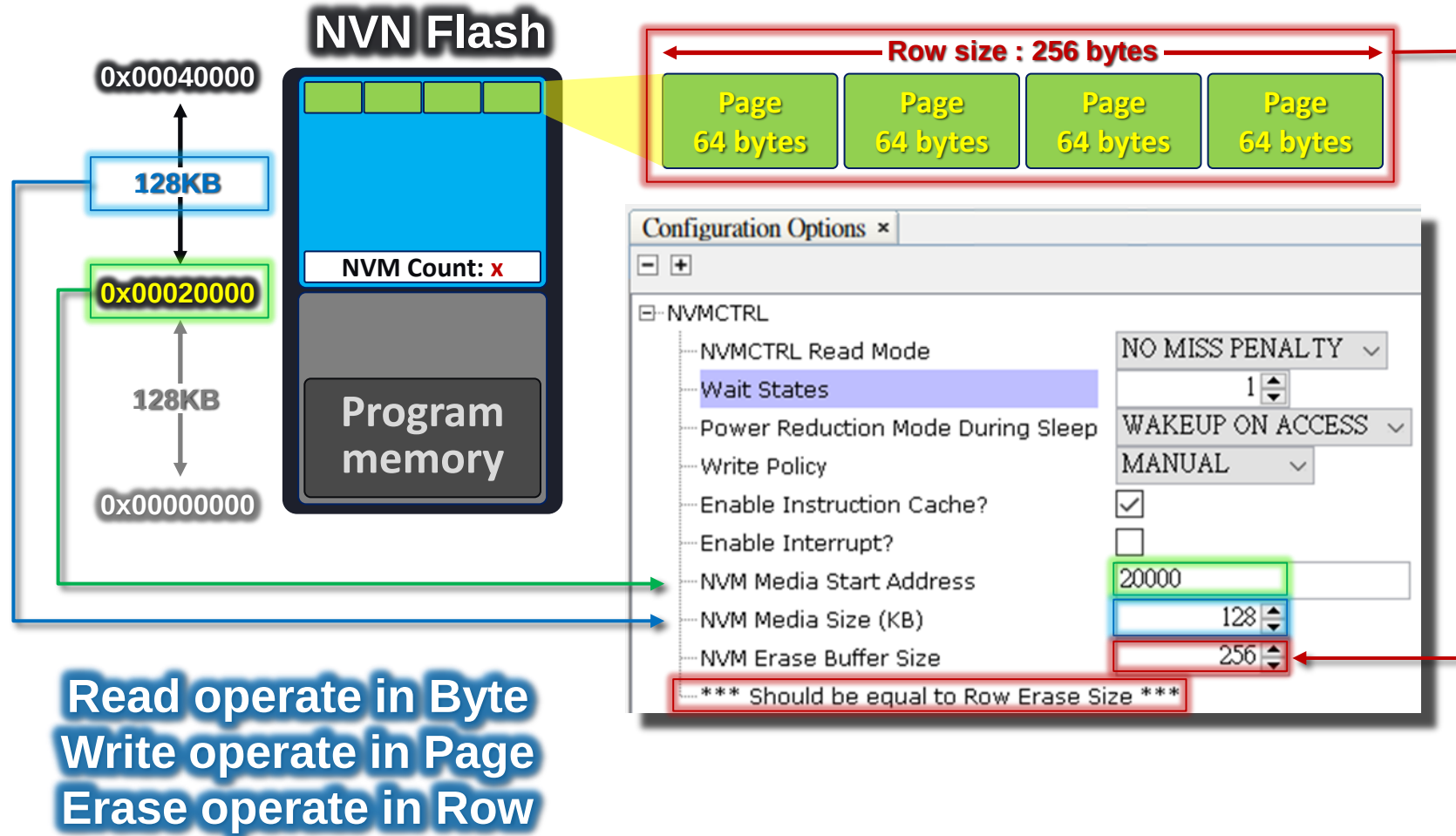
- Number of Clients: 1
- File system Enabled for Memory Driver: ☐
- Buffer Queue Size: 1
- Memory Device Used: NVMCTRL
- *** Configure Memory Device in NVMCTRL Configurations ***

Configuration Options - NVMCTRL:

- NVMCTRL Read Mode: NO MISS PENALTY
- Wait States: 1
- Power Reduction Mode During Sleep: WAKEUP ON ACCESS
- Write Policy: MANUAL
- Enable Instruction Cache? ☒
- Enable Interrupt? ☐
- NVM Media Start Address: 20000
- NVM Media Size (KB): 128
- NVM Erase Buffer Size: 256
- *** Should be equal to Row Erase Size ***

✖ Lab19 Manage NVM Flash without File System

- NVM Flash media address and size description.



✖ Lab19 Manage NVM Flash without File System

- In **Core** (Harmony Core Service), to Increase the “**Number of Applications**” to “**8**” and given the Application Name to “**app_drv_memory**”.

The image shows two windows from a development tool. The left window, titled 'Project Graph*', displays a block diagram of a system. It includes components like EVSYS, TC3, NVMCTRL, TIME, and Core. The Core component is highlighted with a green box. The right window, titled 'Configuration Options', shows the configuration for the Core. A red box highlights the 'Number of Applications' field, which is set to 8. A blue box highlights the 'Application Name' field, which is set to 'app_drv_memory'. A red banner at the top of the right window says 'Increase Number of Application'. A blue banner at the bottom of the left window says 'Rename file to “app_drv_memory”’.

Project Graph*

Device Family Pack (DFP) CMSIS Pack System

EVSYS Peripheral Library

TC3 Peripheral Library TMR

NVMCTRL Peripheral Library MEMORY

TIME System Service

Core Service SYS_TIME

Core Harmony Core Service

RTOS

Core Service

MEMORY Driver

Core Service

Instances - +

Instance 0

MEMORY DRV_MEDIA

Rename file to “app_drv_memory”

Configuration Options

Increase Number of Application

Core

Generate Harmony Application Files ☒

Application Configuration

Number of Applications 8

Application 0 Configuration

Application 1 Configuration

Application 2 Configuration

Application 3 Configuration

Application 4 Configuration

Application 5 Configuration

Application 6 Configuration

Application 7 Configuration

Enable Application 7 Configuration ☒

Application Name app_drv_memory

**** Application name must be valid C-Lang

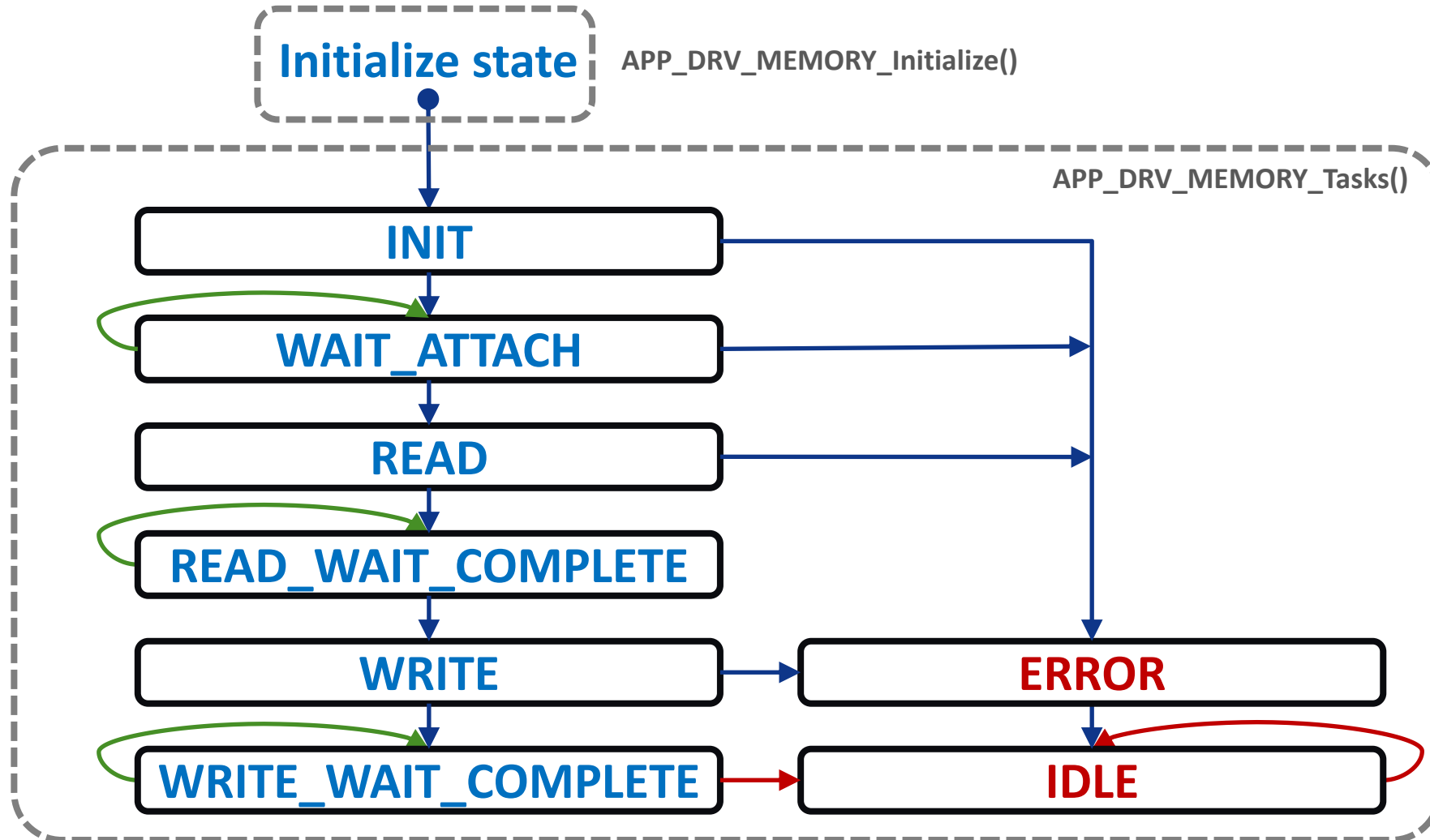
Generate Harmony Driver Common Files ☒

Generate Harmony System Service Common Files ☒

Generate Harmony System Media Files ☒

⚙️ Lab19 Manage NVM Flash without File System

- The new state machine we wish to complete as below.



⚙️ Lab19 Manage NVM Flash without File System

- Add states in “`app_drv_memory.h`”.

```
#ifndef _APP_DRV_MEMORY_H
#define _APP_DRV_MEMORY_H
```

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

```
typedef enum
{
```

Keep this line

```
APP_DRV_MEMORY_STATE_INIT=0,
```

Comment out this line

```
// APP_DRV_MEMORY_STATE_SERVICE_TASKS,
APP_DRV_MEMORY_STATE_WAIT_ATTACH,
APP_DRV_MEMORY_STATE_READ,
APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE,
APP_DRV_MEMORY_STATE_WRITE,
APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE,
APP_DRV_MEMORY_STATE_ERROR,
APP_DRV_MEMORY_STATE_IDLE,
```

```
} APP_DRV_MEMORY_STATES;
```

```
typedef struct
{
```

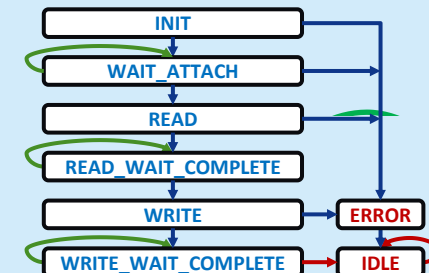
```
    APP_DRV_MEMORY_STATES state;
```

```
} APP_DRV_MEMORY_DATA;
```

```
void APP_DRV_MEMORY_Initialize ( void );
```

app_drv_memory.h

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



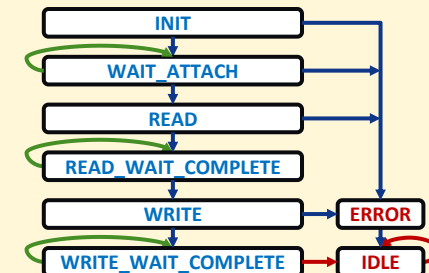
✖ Lab19 Manage NVM Flash without File System

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

```
void APP_DRV_MEMORY_Tasks ( void )  
{  
    switch ( app_drv_memoryData.state )  
    {  
        case APP_DRV_MEMORY_STATE_INIT:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WAIT_ATTACH;  
            break;  
  
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;  
            break;  
  
        case APP_DRV_MEMORY_STATE_READ:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE;  
            break;  
  
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE;  
            break;  
  
        case APP_DRV_MEMORY_STATE_WRITE:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE;  
            break;  
  
        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;  
            break;  
  
        case APP_DRV_MEMORY_STATE_ERROR:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;  
            break;  
  
        case APP_DRV_MEMORY_STATE_IDLE:  
            break;  
    }  
}
```

app_drv_memory.c

Overwrite original example code



✂ Lab19 Manage NVM Flash without File System

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

Keep this line

```
#include "app_drv_memory.h"
#include "app_drv_usart1.h"
#include "driver/memory/drv_memory.h"
#include <stdio.h>
#include <string.h>
```

app_drv_memory.c

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

```
#define NVM_BLOCK_START 0
#define NVM_BUFFER_SIZE 256
#define NVM_GEOMETRY_TABLE_READ_ENTRY (0)
#define NVM_GEOMETRY_TABLE_WRITE_ENTRY (1)
#define NVM_GEOMETRY_TABLE_ERASE_ENTRY (2)
SYS_MEDIA_GEOMETRY *MemoryGeometry = NULL;
int MemoryReadBlocks;
int MemoryWriteBlocks;
int MemoryEraseBlocks;
```

```
DRV_HANDLE MemoryHandle;
DRV_MEMORY_COMMAND_HANDLE MemoryReadHandle;
DRV_MEMORY_COMMAND_HANDLE MemoryWriteHandle;
volatile int MemoryWriteComplete = false;
volatile int MemoryReadComplete = false;
char MemoryWriteBuf[NVM_BUFFER_SIZE];
char MemoryReadBuf[NVM_BUFFER_SIZE];
int MemoryResetCount = 1;
```

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

**For Memory Driver
The Geometry information
is necessary for operation**

⚙️ Lab19 Manage NVM Flash without File System

- Implement **Transfer Callback** for **Memory Driver** Read and Write.

```
*/
APP_DRV_MEMORY_DATA app_drv_memoryData;

// *****
// *****
// Section: Application Callback Functions
// *****
// *****
void MemoryTransferHandler( SYS_MEDIA_BLOCK_EVENT event,
                           SYS_MEDIA_BLOCK_COMMAND_HANDLE commandHandle, uintptr_t context )
{
    switch( event )
    {
        case SYS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE:
            if (commandHandle == MemoryReadHandle)
            { MemoryReadComplete = true; }
            else if(commandHandle == MemoryWriteHandle)
            { MemoryWriteComplete = true; }
            break;

        case SYS_MEDIA_EVENT_BLOCK_COMMAND_ERROR:
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;
            break;
    }
}

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

app_drv_memory.c

Memory Driver Transfer Callback

⚙️ Lab19 Manage NVM Flash without File System

- Open Driver, register callback in state **INIT** and handling **ERROR**.

```
void APP_DRV_MEMORY_Tasks ( void )  
{  
    switch ( app_drv_memoryData.state )  
    {  
        case APP_DRV_MEMORY_STATE_INIT:  
            MemoryHandle = DRV_MEMORY_Open(DRV_MEMORY_INDEX_0, DRV_IO_INTENT_READWRITE);  
            if(MemoryHandle == DRV_HANDLE_INVALID)  
            {  
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;  
                break;  
            }  
        
```

Open Memory Driver to get Handle

Register Transfer Callback

```
        DRV_MEMORY_TransferHandlerSet(MemoryHandle, MemoryTransferHandler, 0);
```

Keep this line

```
        app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WAIT_ATTACH;  
        break;
```

```
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;  
            break;
```

...

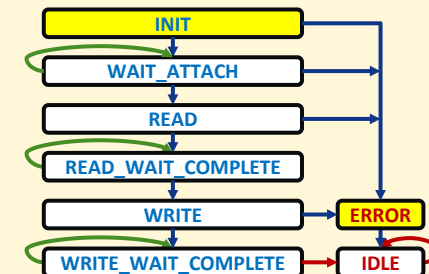
Handling Error state

```
        case APP_DRV_MEMORY_STATE_ERROR:  
            USART1printf("DRV_MEMORY error!\r\n");
```

Keep this line

```
        app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;  
        break;
```

```
        case APP_DRV_MEMORY_STATE_IDLE:  
            break;  
        ...
```



⚙️ Lab19 Manage NVM Flash without File System

- Get Geometry of **NVM Flash** in state **WAIT_ATTACH**.

```
void APP_DRV_MEMORY_Tasks ( void )
{
    switch ( app_drv_memoryData.state )
    {
        case APP_DRV_MEMORY_STATE_INIT: ...
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:
            if( DRV_MEMORY_IsAttached(MemoryHandle) == false )
                break;

            USART1printf("\r\nNVM attached!\r\n");

            MemoryGeometry = DRV_MEMORY_GeometryGet(MemoryHandle);
            if (MemoryGeometry == NULL)
            {
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;
                break;
            }
            MemoryReadBlocks  = (NVM_BUFFER_SIZE / \
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_READ_ENTRY].blockSize);
            MemoryWriteBlocks = (NVM_BUFFER_SIZE / \
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_WRITE_ENTRY].blockSize);
            MemoryEraseBlocks = (NVM_BUFFER_SIZE / \
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);

            Keep this line app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;
            break;

        case APP_DRV_MEMORY_STATE_READ:
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE: ...
        case APP_DRV_MEMORY_STATE_WRITE: ...
        ...
    }
}
```

Using Polling method to Check Memory attach status

Get NVM Flash Geometry

```
graph TD
    INIT[INIT] --> WAIT_ATTACH[WAIT_ATTACH]
    WAIT_ATTACH --> READ[READ]
    READ --> READ_WAIT_COMPLETE[READ_WAIT_COMPLETE]
    READ_WAIT_COMPLETE --> WRITE[WRITE]
    WRITE --> WRITE_WAIT_COMPLETE[WRITE_WAIT_COMPLETE]
    WRITE_WAIT_COMPLETE --> IDLE[IDLE]
    WAIT_ATTACH --> ERROR[ERROR]
    READ --> ERROR
    WRITE --> ERROR
    WRITE_WAIT_COMPLETE --> ERROR
    ERROR --> IDLE
```

⚙️ Lab19 Manage NVM Flash without File System

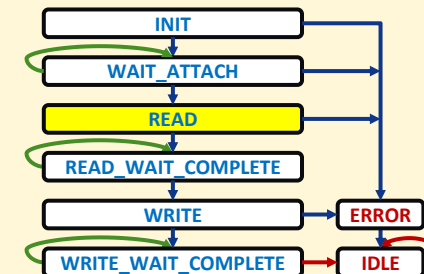
- Read MemoryReadBlocks from address **NVM_BLOCK_START**.

```
void APP_DRV_MEMORY_Tasks ( void )  
{  
    switch ( app_drv_memoryData.state )  
    {  
        case APP_DRV_MEMORY_STATE_INIT: ...  
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH:  
            ...  
            MemoryEraseBlocks = (NVM_BUFFER_SIZE / \  
                MemoryGeometry->geometryTable[NVM_GEOMETRY_TABLE_ERASE_ENTRY].blockSize);  
  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ;  
            break;  
  
        case APP_DRV_MEMORY_STATE_READ:  
            DRV_MEMORY_AsyncRead(MemoryHandle, &MemoryReadHandle,  
                MemoryReadBuf, NVM_BLOCK_START, MemoryReadBlocks);  
            if(MemoryReadHandle == DRV_MEMORY_COMMAND_HANDLE_INVALID)  
            {  
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;  
                break;  
            }  
  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE;  
            break;  
  
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE: ...  
        case APP_DRV_MEMORY_STATE_WRITE: ...  
        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE: ...  
        case APP_DRV_MEMORY_STATE_ERROR: ...  
        case APP_DRV_MEMORY_STATE_IDLE: ...  
        ...  
    }  
}
```

app_drv_memory.c

Read one Block from NVM Flash

Keep this line app_drv_memoryData.state = APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE;
break;



⚙️ Lab19 Manage NVM Flash without File System

- Increase "**Count**" of read content or **output NVM Flash Geometry** if first time to write.

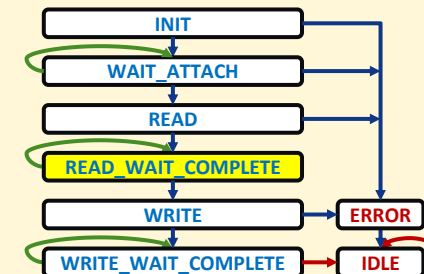
```
void APP_DRV_MEMORY_Tasks ( void )  
{  
    switch ( app_drv_memoryData.state )  
    {  
        case APP_DRV_MEMORY_STATE_INIT: ...  
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH: ...  
        case APP_DRV_MEMORY_STATE_READ: ...  
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE:  
            if(MemoryReadComplete == true)  
            {  
                MemoryReadComplete = false;  
  
                if( !strncmp((char*)MemoryReadBuf, "Count NVM: ", 11) )  
                {  
                    USART1printf("%s\r\n", MemoryReadBuf);  
                    sscanf(MemoryReadBuf, "Count NVM: %d", &MemoryResetCount);  
                    MemoryResetCount = (MemoryResetCount+1)%10;  
                }  
            }  
            else  
            {  
                USART1printf("Geometry NVM: Buff(%d), R(%d), W(%d), E(%d)\r\n",  
                             NVM_BUFFER_SIZE,  
                             MemoryReadBlocks, MemoryWriteBlocks, MemoryEraseBlocks);  
            }  
  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE;  
        }  
        break;  
    }  
    case APP_DRV_MEMORY_STATE_WRITE: ...  
    ...  
}
```

app_drv_memory.c

Output current count and increase it

Output Geometry if nonthing to be wrote

Keep this line



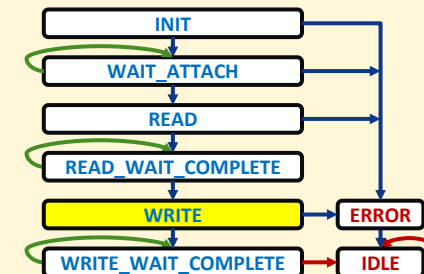
⚙️ Lab19 Manage NVM Flash without File System

- Write back content to target address **NVM_BLOCK_START**.

```
void APP_DRV_MEMORY_Tasks ( void )  
{  
    switch ( app_drv_memoryData.state )  
    {  
        case APP_DRV_MEMORY_STATE_INIT: ...  
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH: ...  
        case APP_DRV_MEMORY_STATE_READ: ...  
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE:  
            ...  
            break;  
  
        case APP_DRV_MEMORY_STATE_WRITE:  
            sprintf( MemoryWriteBuf, "Count NVM: %d", MemoryResetCount);  
            DRV_MEMORY_AsyncEraseWrite(MemoryHandle, &MemoryWriteHandle,  
                                       MemoryWriteBuf, NVM_BLOCK_START, MemoryWriteBlocks);  
  
            if(MemoryWriteHandle == DRV_MEMORY_COMMAND_HANDLE_INVALID)  
            {  
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_ERROR;  
            }  
            else  
            {  
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE;  
            }  
            break;  
  
        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE: ...  
        case APP_DRV_MEMORY_STATE_ERROR: ...  
        case APP_DRV_MEMORY_STATE_IDLE:  
            break;  
        ...  
    }  
}
```

Write count to target Block of SD Card

Keep this line



⚙️ Lab19 Manage NVM Flash without File System

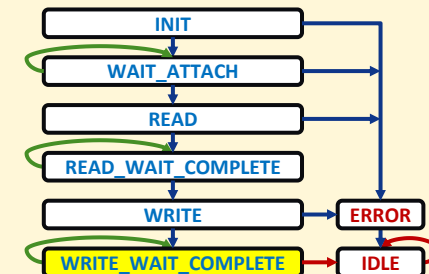
- Close **handle** while write completed and go state **IDLE** forever.

```
void APP_DRV_MEMORY_Tasks ( void )  
{  
    switch ( app_drv_memoryData.state )  
    {  
        case APP_DRV_MEMORY_STATE_INIT: ...  
        case APP_DRV_MEMORY_STATE_WAIT_ATTACH: ...  
        case APP_DRV_MEMORY_STATE_READ: ...  
        case APP_DRV_MEMORY_STATE_READ_WAIT_COMPLETE: ...  
        case APP_DRV_MEMORY_STATE_WRITE:  
            ...  
        }  
        else  
        {  
            app_drv_memoryData.state = APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE;  
        }  
        break;  
  
        case APP_DRV_MEMORY_STATE_WRITE_WAIT_COMPLETE:  
            if(MemoryWriteComplete == true)  
            {  
                MemoryWriteComplete = false;  
  
                DRV_MEMORY_Close( MemoryHandle );  
  
                app_drv_memoryData.state = APP_DRV_MEMORY_STATE_IDLE;  
            }  
            break;  
  
        case APP_DRV_MEMORY_STATE_ERROR: ...  
        case APP_DRV_MEMORY_STATE_IDLE: ...  
        ...  
    }
```

app_drv_memory.c

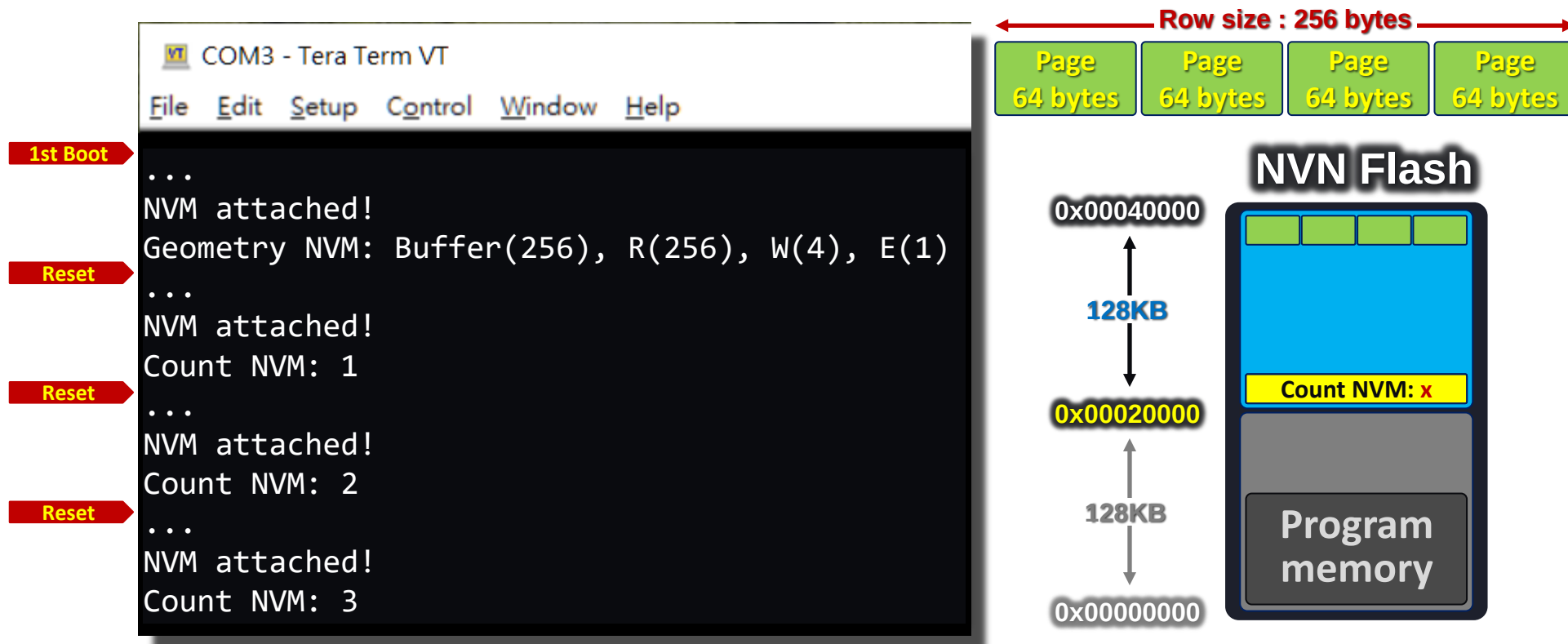
Go state IDLE after finished

Keep this line



⚙️ Lab19 Manage NVM Flash without File System

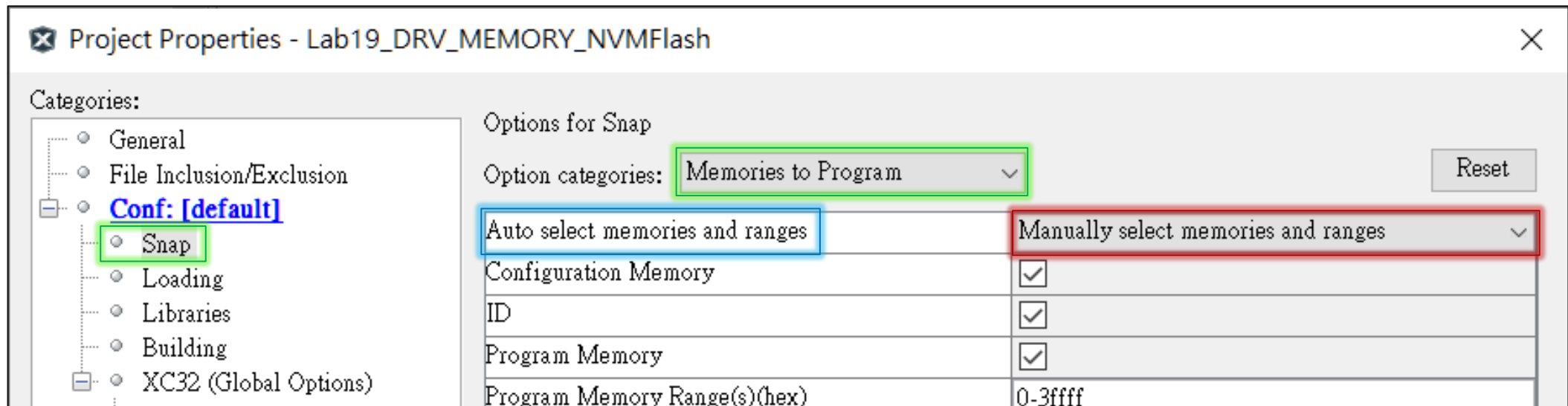
- Compiler and programming your project to EVB.
- Geometry will output while first time to reboot because there has nothing wrote at **Media Start Address 0x20000**.
- The “**Count**” will increase while each time you to press **Reset** button on the EVB.



⚙️ Lab19 Manage NVM Flash without File System

Discuss 1

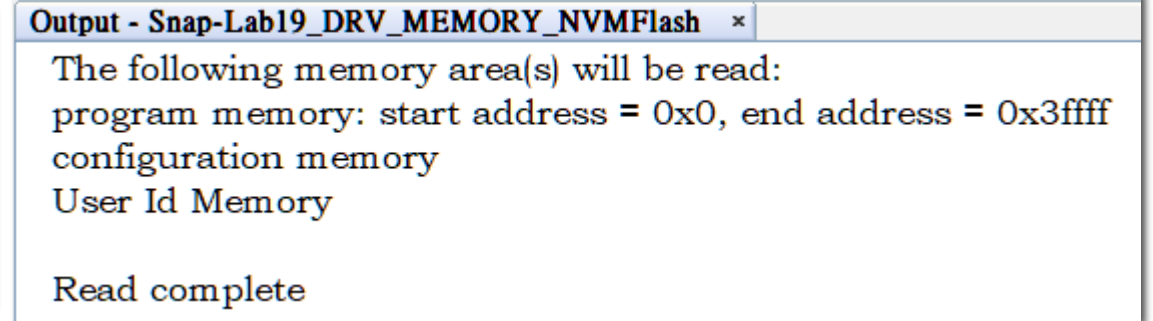
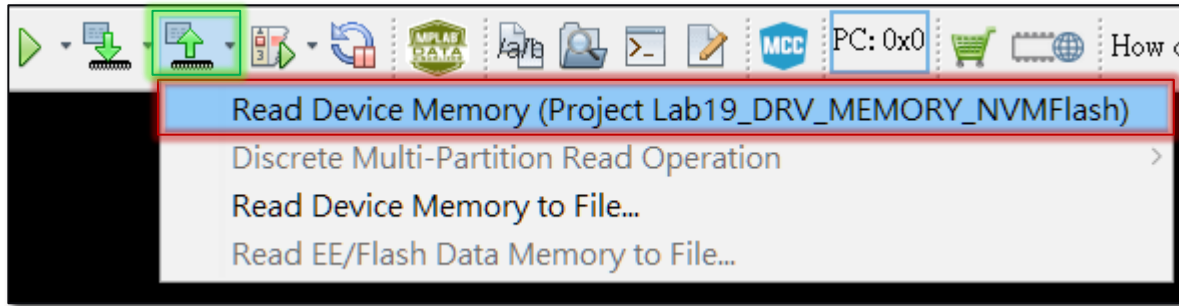
- How to use MPLAB X IDE to check content of NVM Flash we wrote ?
 - The configuration of **debugger MPLAB SNAP** in “Memories to Program” -> “Auto select memories and ranges” has default choose to “Allow SNAP to select Memories”, this option will cause only the areas of MPLAB's memory that **aren't blank** to be programmed.
 - Let's choose to “**Manually select memories and ranges**” for full NVM area.



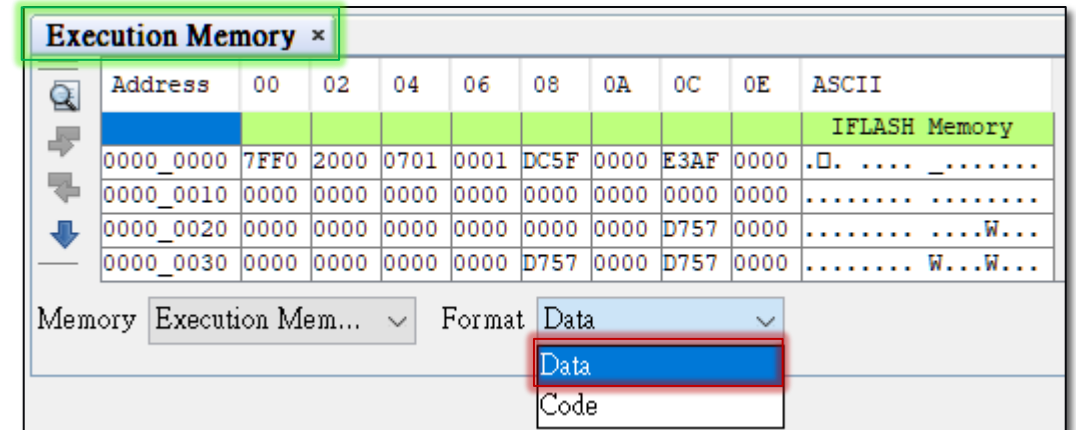
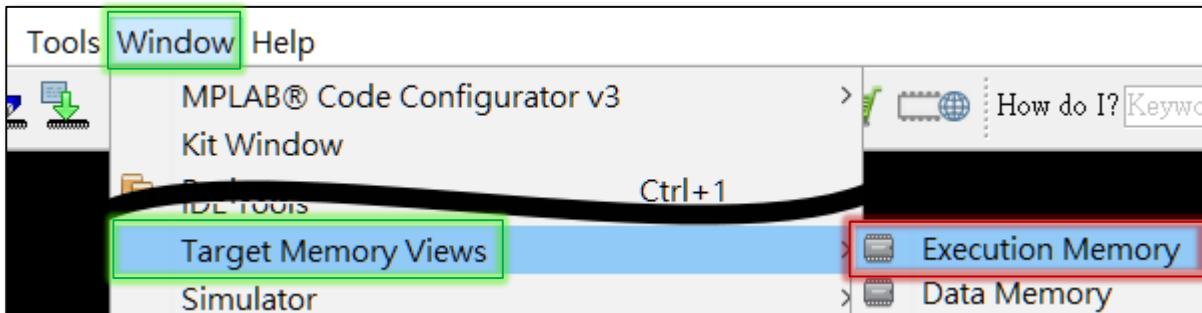
⚙️ Lab19 Manage NVM Flash without File System

Discuss 1

- Read whole SAMD21 **NVM Flash** memory by click  .



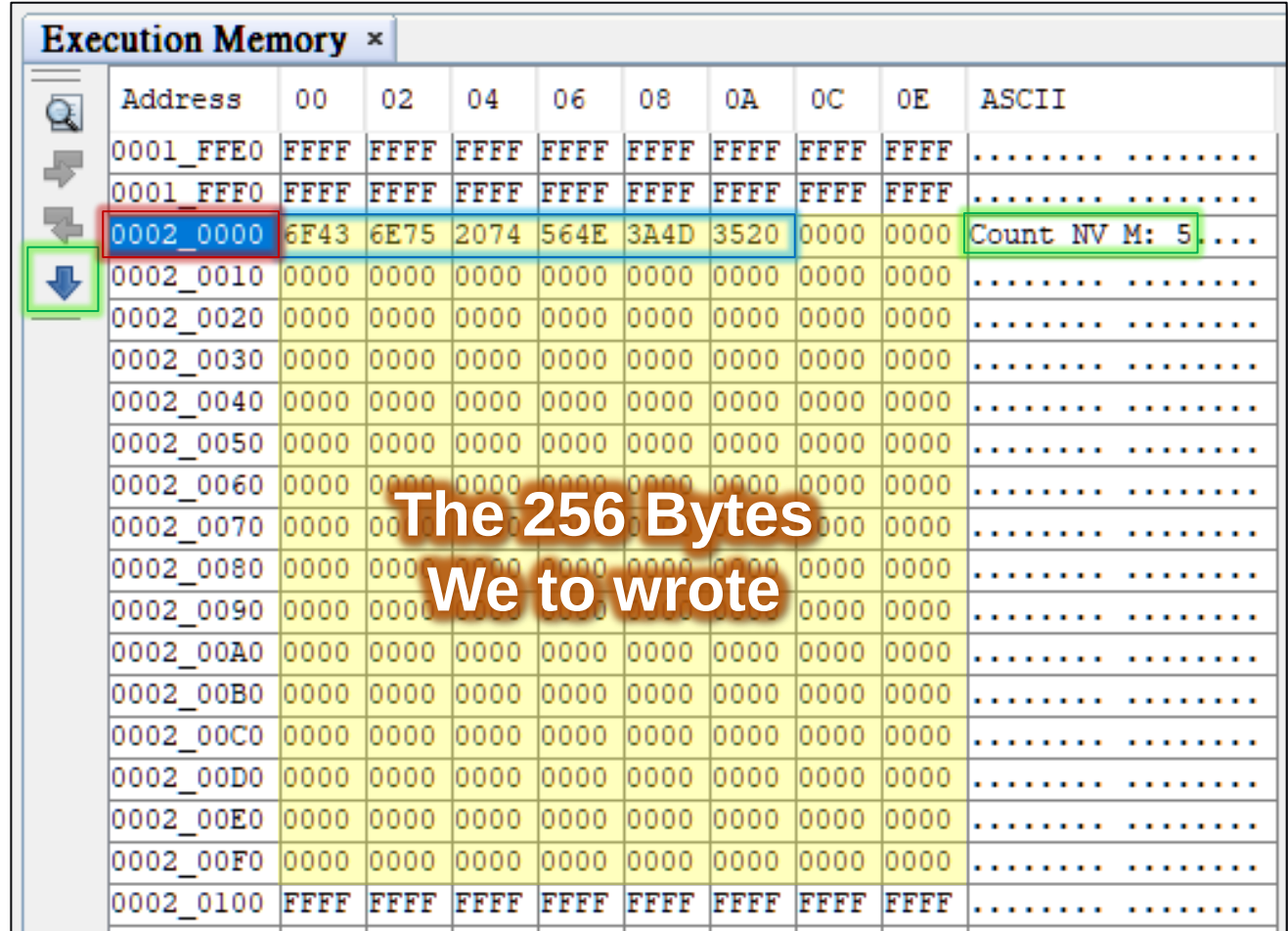
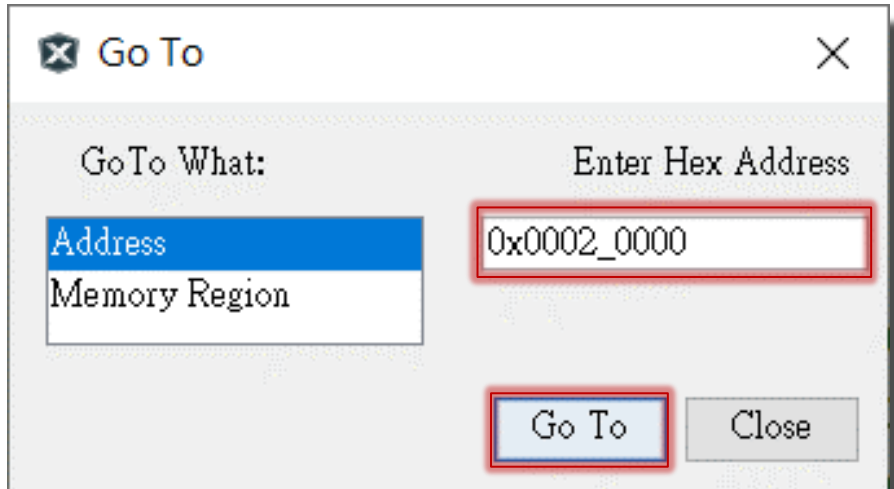
- Open “**Execution Memory**” interface in “**Target Memory Views**”.
- Choose Format to “**Data**” for show raw content of NVM Flash.



⚡ Lab19 Manage NVM Flash without File System

Discuss 1

- Click  and enter Hex Address **0x0002_0000** to find the content we wrote to NVM Flash.



Execution Memory x

Address	00	02	04	06	08	0A	0C	0E	ASCII
0001_FFE0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
0001_FFF0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
0002_0000	6F43	6E75	2074	564E	3A4D	3520	0000	0000	Count NV M: 5....
0002_0010	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0020	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0030	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0040	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0050	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0060	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0070	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0080	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0090	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00A0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00B0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00C0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00D0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00E0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_00F0	0000	0000	0000	0000	0000	0000	0000	0000	
0002_0100	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	

The 256 Bytes We to wrote

Microchip Trademarks

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

Trademarks	Description/Appropriate Nouns	Notes
Adapteck "a" logo		Registered in other countries (not US)
Adapteck®	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
PIC ³² 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		
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
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	
JukeBox®	technology, platform	

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

Trademarks	Description/Appropriate Nouns	Notes
SAM-ICE™	emulator	
SerGenuity®	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	China only
Symcom®		
Symmetricon®		
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
TimePicta®	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
JukeBox with design	Registered word mark JukeBox with a design 	Image is not registered; this is a graphic treatment with the word mark
KEELO®	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
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	
UNIQ®	memory chips, bus, hardware port, communication protocol, firmware	
USBCheck™	design review	
VarSense™	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	
XMGA®	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.

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.

