LAB 1
Bare-Metal Applications

---

1 Lab Rules

- You have to stick to your Lab slot assigned to you on Mosaic.
- Prepare a demonstration for all the Lab experiments, and get ready to be asked in any part of the experiments.
- The demonstrations of this lab will be held starting from **Feb 6 in the last one and half hour of each lab slot**.
- All the activities and questions written in **blue** should be documented and answered in your lab report.
- Each team needs to submit one report for all the members, and the first page of the report should contain the team number and the names of its members.
- Each team also needs to submit source code of problems (not the whole project) along with the report.
- The report and code should be submitted before **23:59 Feb 10**. Put the report in a PDF format, name it with your group number, and submit it to Avenue through the dropbox that will open for the lab on avenue.
- The first page (After the title page) of the report must include a **Declaration of Contributions**, where each team member writes his own individual tasks conducted towards the completion of the lab.

General Note:

- Make sure to get all your work from the lab computer before leaving the lab room because all the saved work will be deleted after the lab slot.

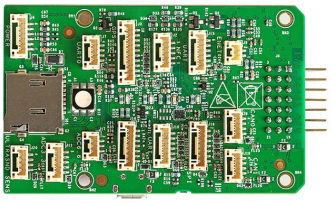
2 Lab Goals

- Control DC and Servo motors using PWM
- Use UART and Telemetry modules to communicate with the host computer wirelessly
- Use FRDMK66F SDKs to configure FMUK66
- Learn basics of SPI communication protocol
- Learn to interface with the on-board Accelerometer and Magnetometer sensors

3 Lab Components

Prepare the following modules before starting the in-lab experiments. Please note that these modules are already assembled into the car. Therefore, you should just try to recognize and find the place of each one.

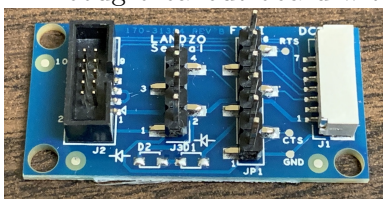
1. 1x RDDRONE-FMUK66 board.



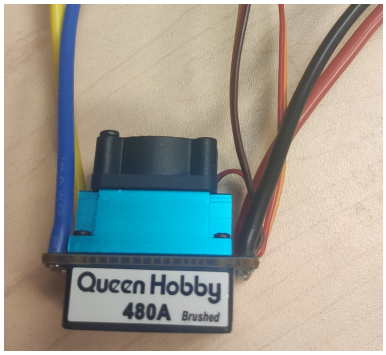
2. 1x Segger J-Link EDU Mini debugger.

3. 3x micro USB cables.

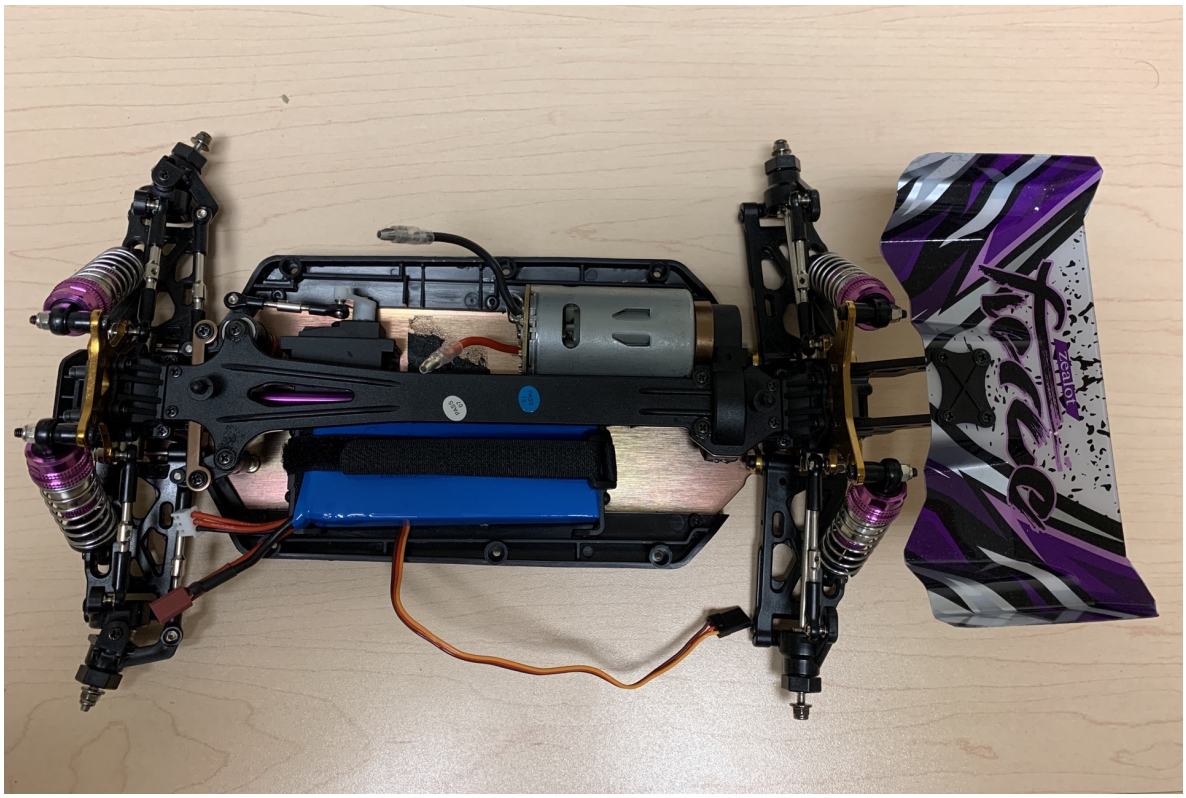
4. 1x Debug breakout board with the 7-wire cable.



5. 1x Brushed Electronic Speed Controller (ESC).



6. 1x RC car chassis with DC motor, Servo motor, and a 7.4v Lithium Polymer battery attached to it.



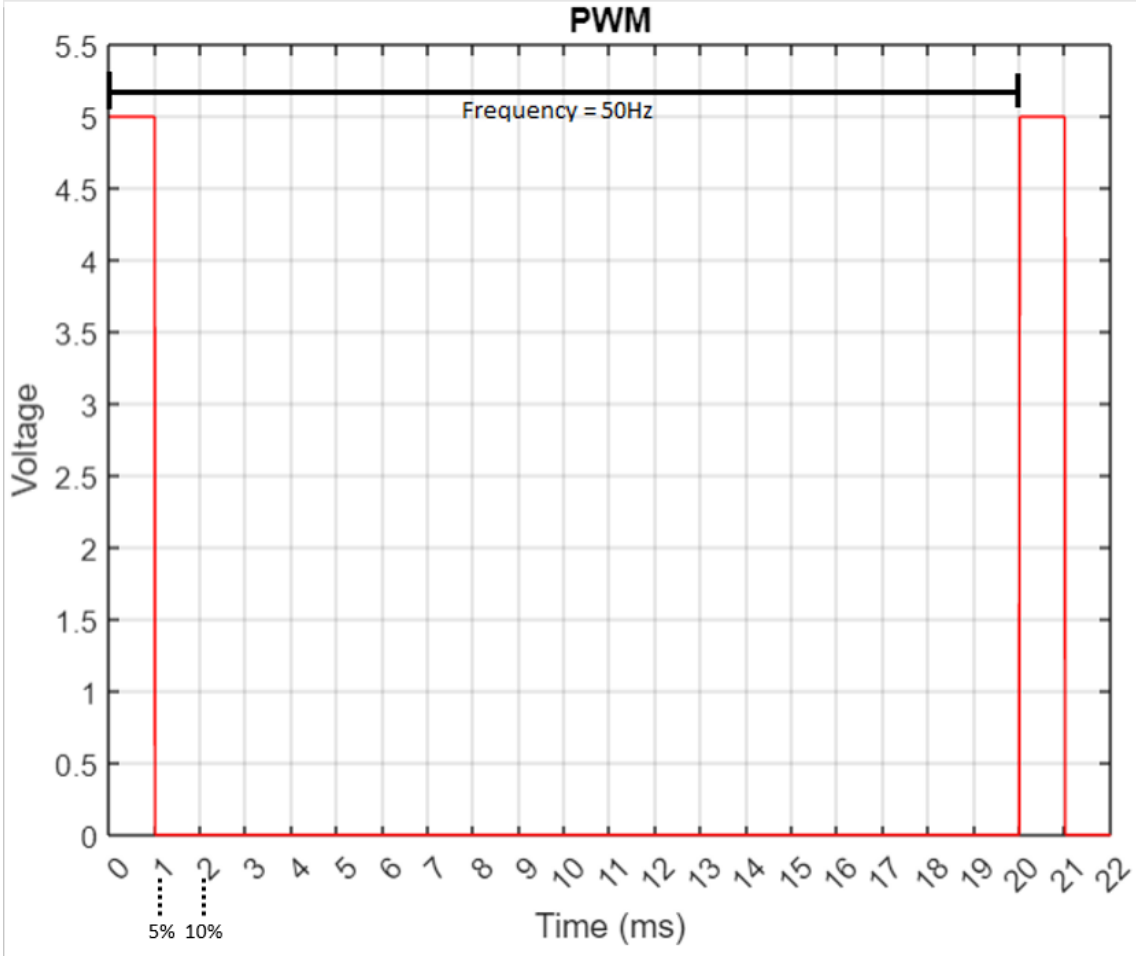
7. 2x Telemetry modules.

8. 1x 6-wire cable.

4 Experiments

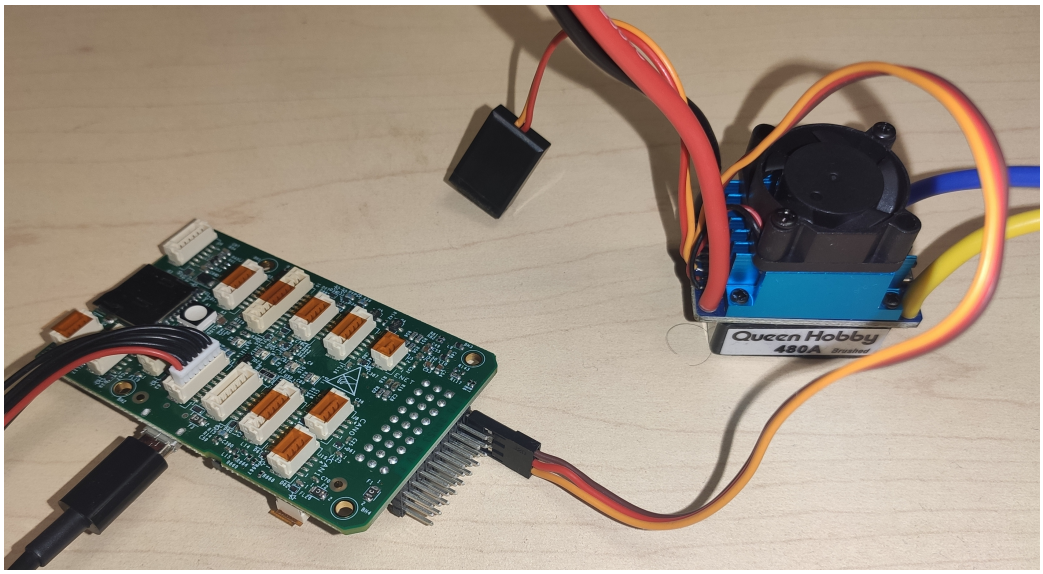
Experiment 1: Control DC motor

In this experiment, we use the PWM to control the speed and the direction of the DC motor. To control the speed, we should use ESC, which reads the PWM signal and change the motor speed according to its duty cycle. PWM frequency is 50 Hz, and duty cycle ranges from 5 % to 10 % as shown in the figure below.

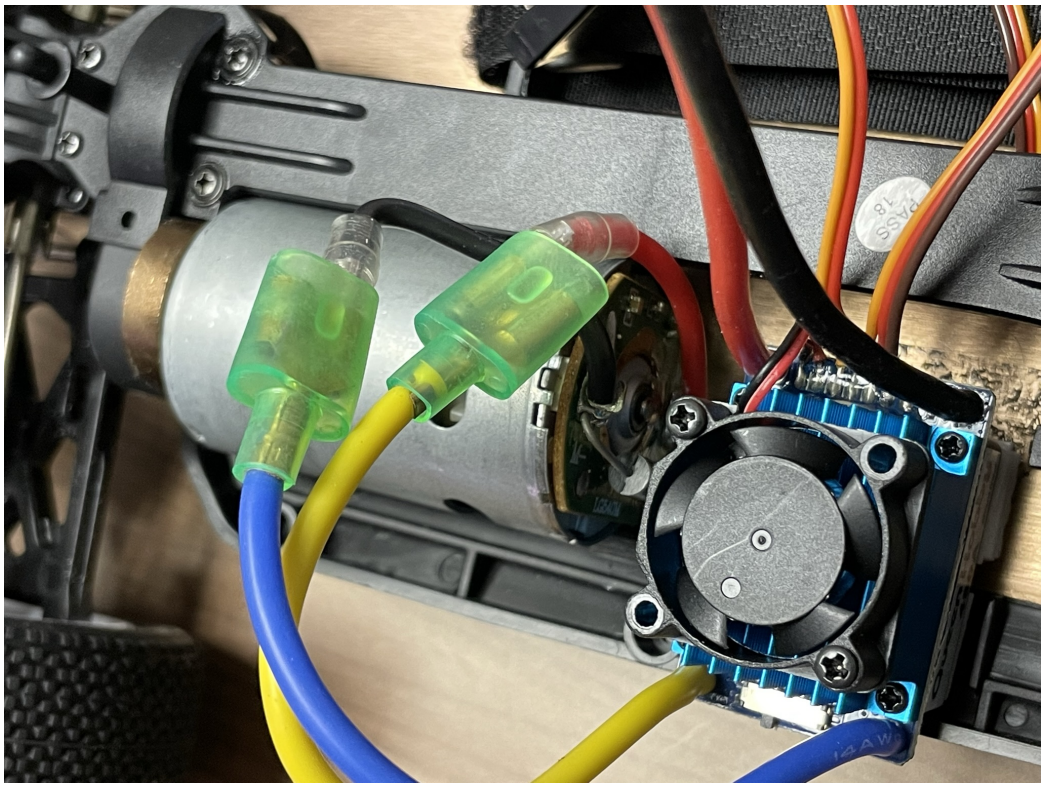


Experiment Setup

1. The J-link debugger is connected to FMUK66.
2. The 3-pin connecter of ESC is connected to the column one of J4 pin header on FMUK66. (It also works on any other column of 3 pins)

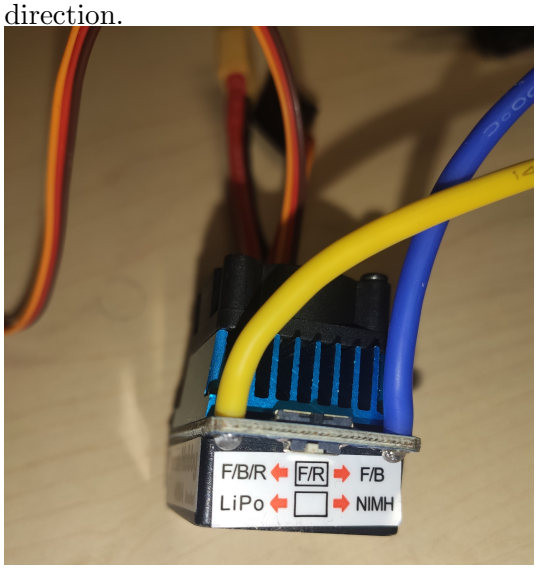


3. The ESC is connected to the DC motor.

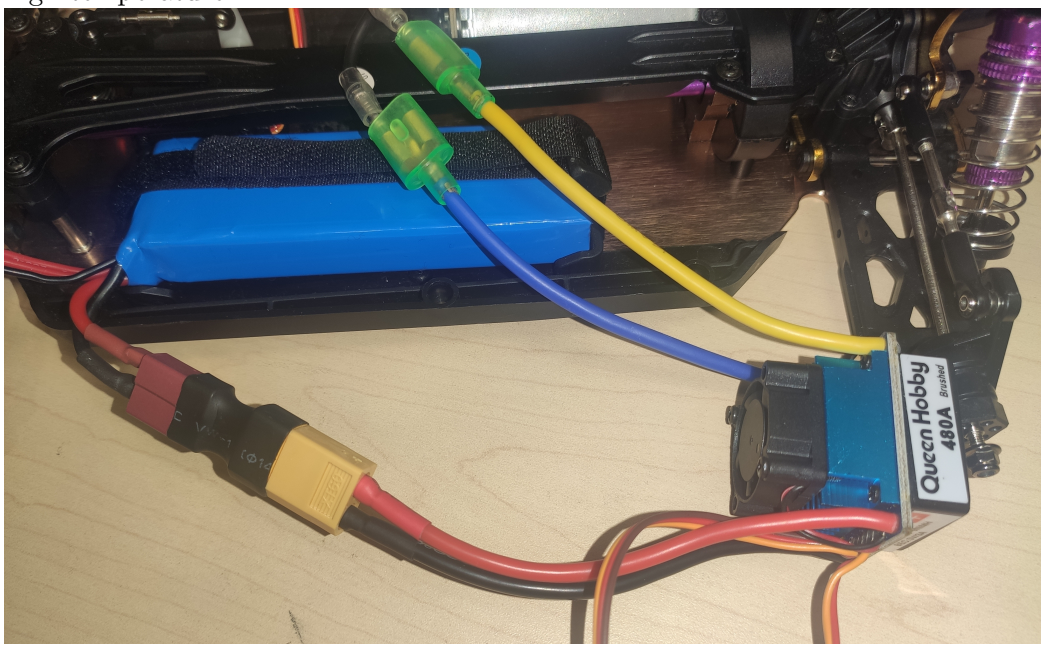


4. On one of ESC sides, you will find two switches. Put the top switch on the middle to choose F/R (Forward and Reverse), and put the bottom one on the left to choose LiPo (lithium polymer battery). As we have selected F/R, 7.5% duty cycle, which is at the mid-point of the duty cycle range, will stop the DC motor. The speed of the motor in the forward direction can be controlled by changing the duty cycle from 7.5% to 10%.

Similarly, keeping the duty cycle between 7.5% to 5% will make it run in the opposite direction.



5. Connect the power wires of ESC to the battery. The batteries are located inside the safety boxes in case of unexpected such as explosion! There are two batteries inside the box, please connect the second battery to the charger while you are using the first one with the car. Also, do not forget to disconnect the batteries from the charger before leaving the lab. Li-Po batteries are susceptible to the explosion due to an overcharge or high temperature.



6. In MCUXpressoIDE, create new "Hello World" project with Semihost debugging, and add FTM driver to it.

7. In the beginning of hello_world.c, include the header file "fsl_ftm.h".

8. Setup the pin used for the PWM signal in "BOARD_InitBootPins" function as you learnt in the previous lab.

9. Change "BOARD_InitBootClocks" to the following code.

```
void BOARD_InitBootClocks(void)
{
    BOARD_BootClockRUN();
}

void BOARD_BootClockRUN(void)
{
    CLOCK_SetSimSafeDivs();

    CLOCK_SetInternalRefClkConfig(kMCG_IrclkEnable, kMCG_IrcFast, 2);
    CLOCK_CONFIG_SetFllExtRefDiv(0);
    CLOCK_SetExternalRefClkConfig(kMCG_OscselIrc);

    CLOCK_SetSimConfig(&simConfig_BOARD_BootClockRUN);
    SystemCoreClock = BOARD_BOOTCLOCKRUN_CORE_CLOCK;
}
```

10. In hello_world.c file, write the code for setupPWM as follows. The codes of this experiment assume that the first pin in J4 is used.

```
#define FTM_MOTOR                FTM0
#define FTM_CHANNEL_DC_MOTOR    kFTM_Chnl_0

void setupPWM()
{
    ftm_config_t ftmInfo;
    ftm_chnl_pwm_signal_param_t ftmParam;
    ftm_pwm_level_select_t pwmLevel = kFTM_HighTrue;

    ftmParam.chnlNumber            = FTM_CHANNEL_DC_MOTOR;
    ftmParam.level                 = pwmLevel;
    ftmParam.dutyCyclePercent      = 7;
    ftmParam.firstEdgeDelayPercent = 0U;
    ftmParam.enableComplementary   = false;
    ftmParam.enableDeadtime        = false;

    FTM_GetDefaultConfig(&ftmInfo);
    ftmInfo.prescale = kFTM_Prescale_Divide_128;

    FTM_Init(FTM_MOTOR, &ftmInfo);
    FTM_SetupPwm(FTM_MOTOR, &ftmParam, 1U, kFTM_EdgeAlignedPwm, 50U, CLOCK_GetFreq(
    kCLOCK_BusClk));
    FTM_StartTimer(FTM_MOTOR, kFTM_SystemClock);
}
```

11. We use the following function to update the duty cycle instead of "FTM_UpdatePwmDutycycle", which was used in the previous lab. The reason of using a new function is to have a floating-point duty cycle, while "FTM_UpdatePwmDutycycle" provides only integer duty cycle. The floating-point value enables us to control the motor speed with fine steps.

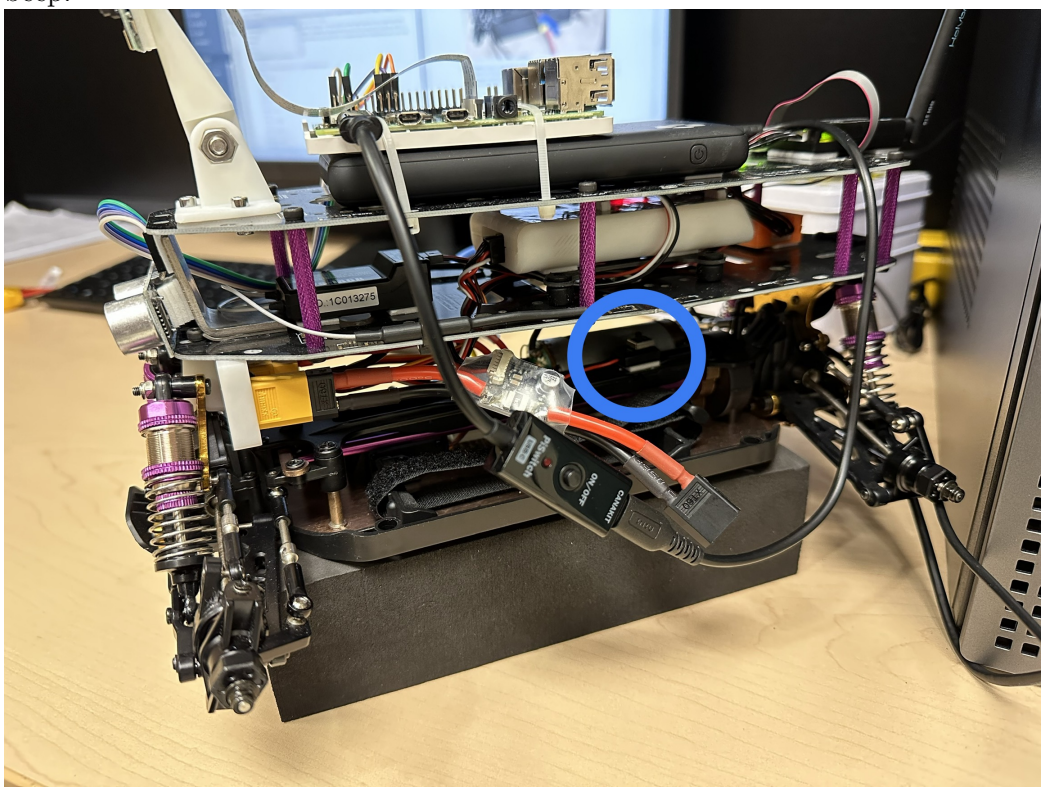
```
void updatePWM_dutyCycle(ftm_chnl_t channel, float dutyCycle)
{
    uint32_t cnv, cnvFirstEdge = 0, mod;

    /* The CHANNEL_COUNT macro returns -1 if it cannot match the FTM instance */
    assert(-1 != FSL_FEATURE_FTM_CHANNEL_COUNTn(FTM_MOTOR));

    mod = FTM_MOTOR->MOD;
    if (dutyCycle == 0U)
    {
        /* Signal stays low */
        cnv = 0;
    }
    else
    {
        cnv = mod * dutyCycle;
        /* For 100% duty cycle */
        if (cnv >= mod)
        {
            cnv = mod + 1U;
        }
    }

    FTM_MOTOR->CONTROLS[channel].CnV = cnv;
}
```

12. Write the following code in the main function. The code expects a value between -100 to 100. Note that the scanf requires input in a certain format.

```
1  int main(void)
2  {
3      uint8_t ch;
4      int input;
5      float dutyCycle;
6
7      BOARD_InitBootPins();
8      BOARD_InitBootClocks();
9
10     setupPWM();
11     /******* Delay *******/
12     for(volatile int i = 0U; i < 1000000; i++)
13         __asm("NOP");
14
15     updatePWM_dutyCycle(FTM_CHANNEL_DC_MOTOR, 0.0615);
16     FTM_SetSoftwareTrigger(FTM_MOTOR, true);
17
18     scanf("s = %d", &input);
19     dutyCycle = input * 0.025f/100.0f + 0.0615;
20     updatePWM_dutyCycle(FTM_CHANNEL_DC_MOTOR, dutyCycle);
21
22     FTM_SetSoftwareTrigger(FTM_MOTOR, true);
23
24     while (1)
25     {}
26 }
```

In the line 21 of the above code, we are calculating the duty cycle. The duty cycle equation should be obtained using the standard PWM range mentioned above. However, the equation we are using for the DC motor does not represent the standard range as the ESC used for this experiments is not standard.

13. Download the code to the board, and switch ESC on.
CAUTION !
DO NOT TURN ESC ON IF THE CAR HAS WHEELS ATTACHED TO IT. THE CAR CAN MOVE IN EXTREMELY HIGH SPEEDS AND BECOME VERY DANGEROUS. REMOVE ALL THE WHEELS BEFORE STARTING.
To turn on the ESC, please switch the key shown on the image below. You will hear a beep.



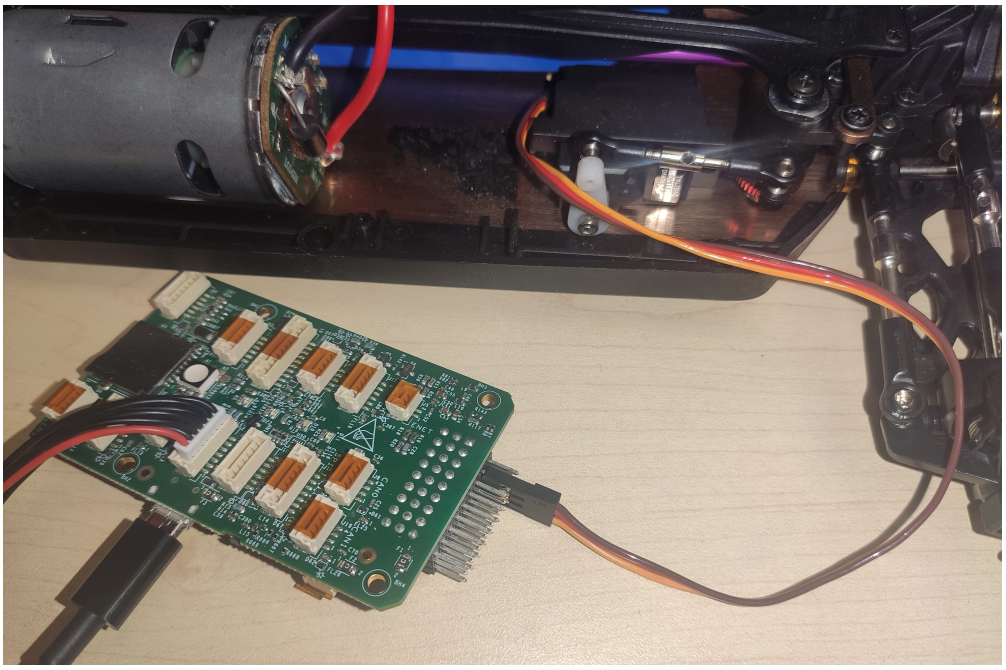
14. Try different duty cycles given in the table below and record your observations regarding speed and direction of the motor. Please be aware that not all of you may end up with a same result. The reason is that the motors are not fully standard and not manufactured with the same characteristics

Duty Cycle	Observation
0	
10	
20	
30	
40	
50	
60	
70	
80	
90	
100	
-10	
-20	
-30	
-40	
-50	
-60	
-70	
-80	
-90	
-100	

Problem 1

Control the car's front wheels position by controlling its servo motor. Controlling servo motor is similar to controlling DC motors, but they can be connected directly to FMUK66 board. The servo motor is connected to the second column of J4 pin header as shown in the figure below. The ESC connected to the DC motor should be turned on for this exercise as it is needed to provide power to the servo motor.

Your program should accept two inputs, first one for the speed of the DC motor and the second for the angle of the servo. You need to come up with an equation to calculate the duty cycle for the servo motor. You should construct your equation according to the standard PWM range (5% to 10%) leveraging what you learned in the previous experiment with the DC motor.

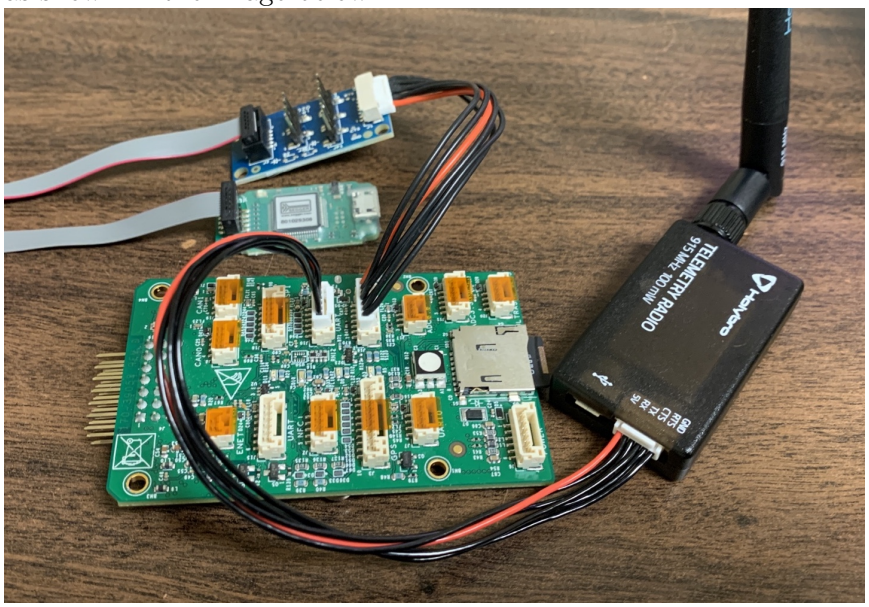


Experiment 2: UART Communication

In this experiment, we use two telemetry modules to send and receive data wirelessly. The telemetry communication is based on UART, thereby we enable UART module in FMUK66.

Experiment Setup

1. The J-link debugger is connected to FMUK66.
2. One of the telemetry modules is already connected to the J10 pin header of the board as shown in the image below.



3. Connect the other telemetry module to your computer using USB cable.
4. The modules connect to each other automatically, and once the connection is established the green LEDs on the modules stop blinking.
5. In MCUXpresso, create the regular "Hello World" project with Semihost debugging.
6. Add UART driver to the project if it is not added.
7. In the beginning of hello_world.c, include "fsl_uart.h".
8. Configure the pins required by UART, which includes UART4_TX, UART4_RX, UART4_CTS_B, and UART4_RTS_B.
9. Change the code of "BOARD_InitBootClocks" function as we did in the previous experiments.
10. Add setupUART function as shown below, which configure the UART module to enable transit (TX), receive (RX), clear to send (CTS), and request to send (RTS). The baud rate of UART is chosen to be 57600.

```
#define TARGET_UART                    UART4

void setupUART()
{
    uart_config_t config;

    UART_GetDefaultConfig(&config);
    config.baudRate_Bps = 57600;
    config.enableTx     = true;
    config.enableRx     = true;
    config.enableRxRTS    = true;
    config.enableTxCTS    = true;

    UART_Init(TARGET_UART, &config, CLOCK_GetFreq(kCLOCK_BusClk));
}
```

11. Write the following code in the main function.

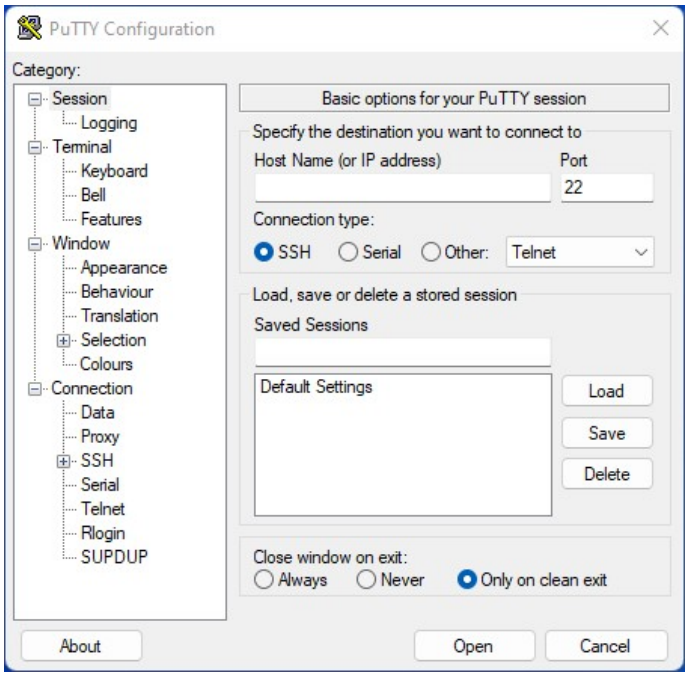
```
int main(void)
{
    char ch;
    char txbuff[] = "Hello World\r\n";

    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    setupUART();
    /******* Delay *******/
    for(volatile int i = 0U; i < 10000000; i++)
        __asm("NOP");

    PRINTF("%s", txbuff);
```

```
    UART_WriteBlocking(TARGET_UART, txbuff, sizeof(txbuff) - 1);

    while (1)
    {
        UART_ReadBlocking(TARGET_UART, &ch, 1);
        PRINTF("%c\r\n", ch);
    }
}
```

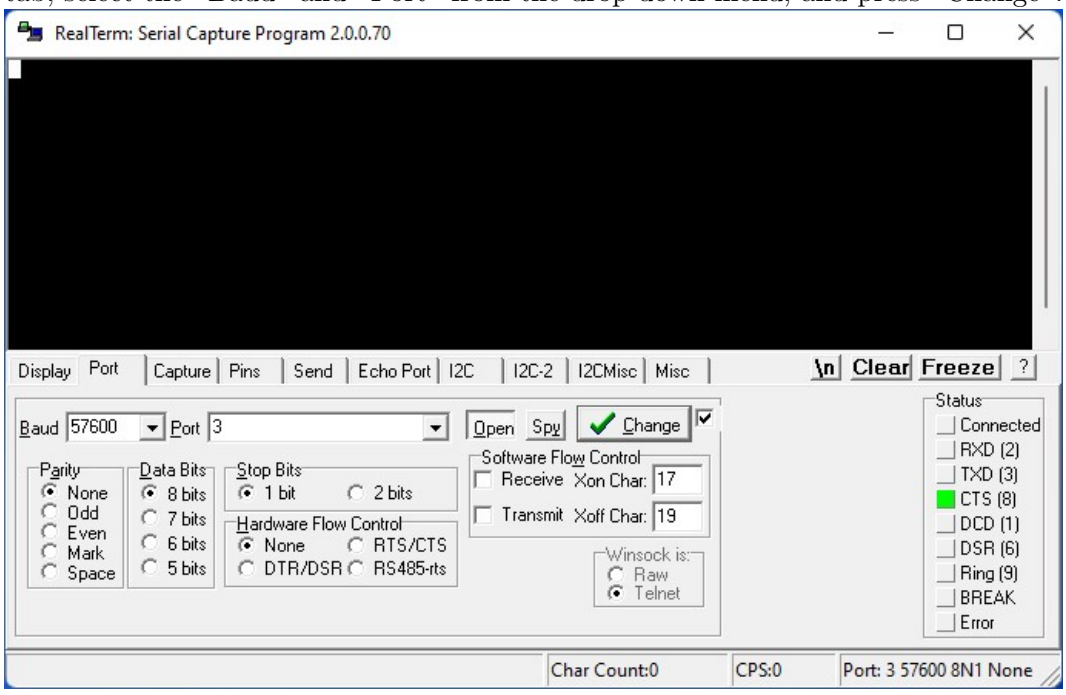
12. As we are going to communicate with the board using the UART telemetry module, connected to the computer, we should find which COM port the module is connected to. Disconnect the telemetry module from the computer and open "Device Manager", look for "Ports (COM & LPT)" category (you may not see this category if no port is connected), then connect the telemetry module and look for changes. You may now see which port was connected recently. e.g. COM3.
    The word "COM" comes from the "Communication Port", and it was initially used to refer to serial UART COM ports on early computers (search "COM (hardware interface)" on Wikipedia). However, when you connect a UART communicator module to the computer using USB, it resembles a COM port from the OS perspective.

13. On the computer side, you should open a serial monitor such as PuTTY or RealTerm. Both of these applications are already installed on the computers and you can use either of them. The reason is that the UART communication we have implemented is not stable enough, therefore, you may want to try different tools to see which one works better for you. You can even use the "screen" command on the Linux Shell.

    PuTTY: Once you open the Putty, you will see a window similar to the image below. From the "Category" box, open "Session", define the "Connection type" as Serial, change the "Serial line" to the COM port you found on the device manager, and finally, enter the baud rate in the "Speed" textbox, and press the "Open".



RealTerm: The RealTerm GUI is as shown in the image below. Navigate to the "Port" tab, select the "Baud" and "Port" from the drop down menu, and press "Change".



Linux Shell: Navigate to the terminal and run command "$ls\ /dev/tty*$", which lists the connected devices. Please note that the COM# you found from the previous step is not going to work here. To get the name of the telemetry module connected to your computer, run the command when the module is unplugged and run it again after

connecting it. The new name appeared after the second run is the name of the module. Run the "*screen /dev/tty*[*modulename*] 57600" command to connect to the telemetry module.

14. Run the code from MCUXpresso into MCU, and you should see "Hello World" printed in Terminal. If you type any character in Terminal, the character will appear in MCUXpresso's console.

    Note: As the telemetry modules we have are not standard and may work in the same band of the frequency, which is 915 MHz, please be prepared for seeing interference with other telemetry modules in the lab. Therefore, you may want to ask your friends to disconnect their module for a short period of time if your connection is not stable, specifically, while doing the demo. Additionally, there is a red LED on both modules, and they will turn on for less than a second once a packet is sent. You can also use this LED for debugging and to see if the packet is being sent

Problem 2

Repeat the same function of Problem 1 but instead of getting inputs from MCUXpresso's console, get the inputs from the UART. Additionally, modify your code in order to enable the user to update the speed and the angle several times in real-time without the need for a reset.

Experiment 3: UART Communication with Interrupts

In this experiment, we update the project of Experiment 2 to use interrupts with UART reciepts instead of the blocking reciepts.

Experiment Setup

1. Use the same project of Experiment 2 and keep all the board connections without change.

2. Modify setupUART and define two volatile global variables as shown in the following code.

```
#define TARGET_UART                    UART4

volatile char ch;
volatile int new_char = 0;

void setupUART()
{
    uart_config_t config;

    UART_GetDefaultConfig(&config);
    config.baudRate_Bps = 57600;
    config.enableTx     = true;
    config.enableRx     = true;
    config.enableRxRTS    = true;
    config.enableTxCTS    = true;
```

```
    UART_Init(TARGET_UART, &config, CLOCK_GetFreq(kCLOCK_BusClk));

    /******** Enable Interrupts *********/
    UART_EnableInterrupts(TARGET_UART, kUART_RxDataRegFullInterruptEnable);
    EnableIRQ(UART4_RX_TX_IRQn);
}
```

3. Add the following interrupt service routine (ISR) and use the **same exact name**. Note that we are not using the status returned by the UART_GetStatusFlags function but it is necessary to call the function as it clears the interrupt. If we do not clear the interrupt, it will be triggered again as soon as we exit the ISR.

```
void UART4_RX_TX_IRQHandler()
{
    UART_GetStatusFlags(TARGET_UART);
    ch = UART_ReadByte(TARGET_UART);
    new_char = 1;
}
```

4. Modify the main function as follows.

```
int main(void)
{
    char txbuff[] = "Hello World\r\n";

    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    setupUART();
    /******* Delay *******/
    for(volatile int i = 0U; i < 10000000; i++)
        __asm("NOP");

    PRINTF("%s", txbuff);
    UART_WriteBlocking(TARGET_UART, txbuff, sizeof(txbuff) - 1);

    while (1)
    {
        if(new_char)
        {
            new_char = 0;
            PRINTF("%c\r\n", ch);
        }
    }
}
```

5. Compile and download the code on the board.

6. Run the code with Terminal and you should get the same behavior of Experiment 2. Try to put breakpoints inside the ISR and inside the conditional if to follow the execution flow of the code.

Problem 3

Update your code of Problem 2 to use interrupts.

Experiment 4: Interface with the Accelerometer Sensor

This experiment is divided into two parts. In part A, we will configure the SPI module in FMUK66, as it is the interfacing module with the accelerometer. Afterwards in part B, we use one of the SDK examples that setups the sensor and port it to the board.

Experiment Setup: Part A

In order to test the operation of the SPI module, we will configure the SPI to interface with the accelerometer sensor and perform a simple operation. The operation is to read one of the internal registers of the accelerometer using SPI.

1. No external connections are required for this experiment, only FMUK66 board and J-Link debugger.
2. Create the usual "Hello World" project with Semihost debugging.
3. Add dspi driver to the project.
4. In hello_world.c file, include "fsl_dspi.h".
5. To configure the pins required by the accelerometer, open the schematics and search for the accelerometer chip which is called "FXOS8700CQ".
6. The pins SPI1_SCK_INTERNAL, SPI1_MOSI_INTERNAL, SPI1_MISO_INTERNAL, and nSPI1_CS0_ACCEL_MAG are the SPI pins. Map them on the MCU and initiate them in your code. Do not forget to choose their alternative functions to be SPI.
7. RST pin is also required by the accelerometer to work properly, and it should be kept at zero (choose its alternative function to be GPIO).
8. Another crucial pin is required to power up the sensor chip. If you notice the VDD of the accelerometer, in the schematics, you will find out that it is connected to 3v3_S. This power signal comes from a voltage regulator chip called "LP5907SNX-3.3", in page 4. To enable the voltage regulator, configure the MCU pin connected to its EN pin to be a GPIO, and keep the GPIO output high.
9. Change the code of "BOARD_InitBootClocks" function as we did in the previous experiments.
10. Back to hello_world.c file, add the following setupSPI function.

```
void setupSPI()
{
    dspi_master_config_t masterConfig;

    /* Master config */
    masterConfig.whichCtar                                  = kDSPI_Ctar0;
```

```
    masterConfig.ctarConfig.baudRate                     = 500000;
    masterConfig.ctarConfig.bitsPerFrame                 = 8U;
    masterConfig.ctarConfig.cpol                         =
    kDSPI_ClockPolarityActiveHigh;
    masterConfig.ctarConfig.cpha                         = kDSPI_ClockPhaseFirstEdge;
    masterConfig.ctarConfig.direction                    = kDSPI_MsbFirst;
    masterConfig.ctarConfig.pcsToSckDelayInNanoSec       = 1000000000U / 500000;
    masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec   = 1000000000U / 500000;
    masterConfig.ctarConfig.betweenTransferDelayInNanoSec = 1000000000U / 500000;

    masterConfig.whichPcs           = kDSPI_Pcs0;
    masterConfig.pcsActiveHighOrLow = kDSPI_PcsActiveLow;

    masterConfig.enableContinuousSCK        = false;
    masterConfig.enableRxFifoOverWrite      = false;
    masterConfig.enableModifiedTimingFormat = false;
    masterConfig.samplePoint                = kDSPI_SckToSin0Clock;

    DSPI_MasterInit(SPI1, &masterConfig, BUS_CLK);
}
```

11. Add two functions to output zero for the RST pin of the accelerometer and output one for the EN pin of the voltage regulator.

```
void voltageRegulatorEnable()
{
    gpio_pin_config_t pin_config = {
            .pinDirection = kGPIO_DigitalOutput,
            .outputLogic = 0U};
    GPIO_PinInit(GPIOB, 8, &pin_config);
    GPIO_PinWrite(GPIOB, 8, 1U);
}

void accelerometerEnable()
{
    gpio_pin_config_t pin_config = {
            .pinDirection = kGPIO_DigitalOutput,
            .outputLogic = 0U};
    GPIO_PinInit(GPIOA, 25, &pin_config);
    GPIO_PinWrite(GPIOA, 25, 0U);
}
```

12. The last function needed is to read any register from the accelerometer sensor. The procedure of reading and writing from and to the sensor's registers is documented in its datasheet, which you can find in the "accelerometer&magnetometer.pdf" file.Check Sections 10.2.1 and 10.2.2 which explains SPI read and write, and then add the following function which implements SPI read. Note: You want to make sure that you free the allocated memory in the following code using "free()" function as it was causing a crash in the experiment.

```
status_t SPI_read(uint8_t regAddress, uint8_t *rxBuff, uint8_t rxBuffSize)
{
    dspi_transfer_t masterXfer;
    uint8_t *masterTxData = (uint8_t*)malloc(rxBuffSize + 2);
```

```
    uint8_t *masterRxData = (uint8_t*)malloc(rxBuffSize + 2);

    masterTxData[0] = regAddress & 0x7F; //Clear the most significant bit
    masterTxData[1] = regAddress & 0x80; //Clear the least significant 7 bits

    masterXfer.txData      = masterTxData;
    masterXfer.rxData      = masterRxData;
    masterXfer.dataSize    = rxBuffSize + 2;
    masterXfer.configFlags = kDSPI_MasterCtar0 | kDSPI_MasterPcs0 |
    kDSPI_MasterPcsContinuous;
    status_t ret = DSPI_MasterTransferBlocking(SPI1, &masterXfer);
    memcpy(rxBuff, &masterRxData[2], rxBuffSize);

    free(masterTxData);
    free(masterRxData);

    return ret;
}
```

13. Write the following code in the main function. The code reads a register called WHO_AM_I, which is located at the address 0x0D. This register contains a constant value, 0xC7, and it is used to identify the sensor (all the sensor's registers are listed in Chapter 14 in its datasheet). Therefore, the expected output of this code is to print 0xC7.

```
int main(void)
{
    uint8_t byte;

    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    setupSPI();
    voltageRegulatorEnable();
    accelerometerEnable();

    /******* Delay *******/
    for(volatile int i = 0U; i < 1000000; i++)
        __asm("NOP");

    SPI_read(0x0D, &byte, 1);
    printf("The expected value is 0xC7 and the read value 0x%X\n", byte);

    while (1)
    {
    }
}
```

14. Download and run the code, and then compare the expected value with the output value.

Problem 4

Add to the code of the previous experiment a function that writes to the sensor's registers, and call it SPI_write. The function should have the following prototype, which writes the `value` to the register whose address is `regAddress`.

```
status_t SPI_write(uint8_t regAddress, uint8_t value);
```

Make sure to keep the code accessible as it will be used in the next experiment.

Experiment Setup: Part B

In this part, we will use an example provided by the SDK for FRDMK66F board. Unlike FMUK66 board, FRDMK66F board connects the accelerometer to I^2C module. Accordingly, our goal in this experiment is to modify the example to work with SPI instead of I^2C.

1. Import from the SDK, demo_apps $->$ bubble_peripheral, and do not forget to choose Semihost for debugging.
2. Add dspi driver to the project, and include "fsl_dspi.h" in bubble.c file.
3. Use the same code of the functions BOARD_InitBootPins and BOARD_InitBootClocks from the project in Part A.
4. Remove the following functions from bubble.c
   - i2c_release_bus_delay
   - BOARD_I2C_ReleaseBus
   - Board_UpdatePwm
5. Add the following functions from the project in Part A
   - setupSPI
   - voltageRegulatorEnable
   - accelerometerEnable
   - SPI_read
   - SPI_write
6. In `fsl_fxos.h` file (which is one of the includes in bubble.c), add the following lines of code. You can open the file by holding "ctrl" and left click on its name in the includes. You can also find the file by navigating the `workspace` tab in the left, you will find it under: `YOUR PROJECT NAME`→`accel`→`fsl_fxos.h` folder. These two lines define new types which they are pointers to functions, and they are called SPI_WriteFunc_t and SPI_ReadFunc_t. Please put these two lines before all other "typedef"s in your code. These two types are chosen to be able to point at our SPI functions, SPI_write and SPI_read.

```
typedef status_t (*SPI_WriteFunc_t)(uint8_t regAddress, uint8_t value);
typedef status_t (*SPI_ReadFunc_t)(uint8_t regAddress, uint8_t *rxBuff, uint8_t
    rxBuffSize);
```

7. In the same file, search for the structures callled fxos_handle_t and fxos_config_t. At the end of both structures add two extra lines to create pointers in the structure which will be used later to point to our SPI read and write functions. The structures should appear as follows.

```
typedef struct _fxos_handle
{
    /* Pointer to the user-defined I2C Send Data function. */
    I2C_SendFunc_t I2C_SendFunc;
    /* Pointer to the user-defined I2C Receive Data function. */
    I2C_ReceiveFunc_t I2C_ReceiveFunc;
    /* The I2C slave address . */
    uint8_t slaveAddress;

    /* Pointer to the user-defined SPI write Data function. */
    SPI_WriteFunc_t SPI_writeFunc;
    /* Pointer to the user-defined SPI read Data function. */
    SPI_ReadFunc_t SPI_readFunc;
} fxos_handle_t;
```

```
typedef struct _fxos_config
{
    /* Pointer to the user-defined I2C Send Data function. */
    I2C_SendFunc_t I2C_SendFunc;
    /* Pointer to the user-defined I2C Receive Data function. */
    I2C_ReceiveFunc_t I2C_ReceiveFunc;
    /* The I2C slave address . */
    uint8_t slaveAddress;

    /* Pointer to the user-defined SPI write Data function. */
    SPI_WriteFunc_t SPI_writeFunc;
    /* Pointer to the user-defined SPI read Data function. */
    SPI_ReadFunc_t SPI_readFunc;
} fxos_config_t;
```

8. In accel− >fsl_fxos.c file (`accel` is the same folder that contains `fsl_fxos.h`) (which is the file that uses the previous structures), replace any I2C_SendFunc with SPI_writeFunc and I2C_ReceiveFunc with SPI_readFunc.

9. In the same file, search for the functions FXOS_ReadReg and FXOS_WriteReg, and replace the I2C_Receive_func with SPI_readFunc as we are using the SPI to communicate with accelerometer sensor. Make sure to update functions arguments and they should appear as follows.

```
status_t FXOS_ReadReg(fxos_handle_t *handle, uint8_t reg, uint8_t *val, uint8_t
    bytesNumber)
{
    assert(handle);
    assert(val);

    if ((handle->SPI_readFunc) == NULL)
    {
        return kStatus_Fail;
    }

    return handle->SPI_readFunc(reg, val, bytesNumber);
}
```

```
status_t FXOS_WriteReg(fxos_handle_t *handle, uint8_t reg, uint8_t val)
{
    assert(handle);

    if ((handle->SPI_writeFunc) == NULL)
    {
        return kStatus_Fail;
    }

    return handle->SPI_writeFunc(reg, val);
}
```

10. Return back to bubble.c, and write the following code in the main function. In the main function, first we are initializing the pins and clocks of the sensor and setting up the SPI module. Then we are configuring the SPI function, initializing and calibrating the sensor device. In the main infinite loop, we are getting the data from the accelerometer sensor and compiling the received data. Finally, we are converting the raw data into X and Y angles and printing it.

```
int main(void)
{
    fxos_handle_t fxosHandle = {0};
    fxos_data_t sensorData   = {0};
    fxos_config_t config     = {0};

    uint8_t sensorRange      = 0;
    uint8_t dataScale        = 0;
    int16_t xData            = 0;
    int16_t yData            = 0;
    uint8_t i                = 0;
    uint8_t array_addr_size  = 0;

    status_t result          = kStatus_Fail;

    volatile int16_t xAngle = 0;
    volatile int16_t yAngle = 0;

    /* Board pin, clock, debug console init */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
```

```
voltageRegulatorEnable();
accelerometerEnable();

setupSPI();

/***** Delay *****/
for (volatile uint32_t i = 0; i < 4000000; i++)
    __asm("NOP");

/* Configure the SPI function */
config.SPI_writeFunc = SPI_write;
config.SPI_readFunc  = SPI_read;

result = FXOS_Init(&fxosHandle, &config);
if (result != kStatus_Success)
{
    PRINTF("\r\nSensor device initialize failed!\r\n");
    return -1;
}

/* Get sensor range */
if (FXOS_ReadReg(&fxosHandle, XYZ_DATA_CFG_REG, &sensorRange, 1) != kStatus_Success)
{
    return -1;
}

if (sensorRange == 0x00)
{
    dataScale = 2U;
}
else if (sensorRange == 0x01)
{
    dataScale = 4U;
}
else if (sensorRange == 0x10)
{
    dataScale = 8U;
}
else
{
}

/* Print a note to terminal */
PRINTF("\r\nWelcome to the BUBBLE example\r\n");
PRINTF("\r\nYou will see the change of angle data\r\n");

/* Main loop. Get sensor data and update duty cycle */
while (1)
{
    /* Get new accelerometer data. */
    if (FXOS_ReadSensorData(&fxosHandle, &sensorData) != kStatus_Success)
    {
        return -1;
    }
```

```
        /* Get the X and Y data from the sensor data structure in 14 bit left format
    data*/
        xData = ((int16_t)((uint16_t)((uint16_t)sensorData.accelXMSB << 8) | (uint16_t)
    sensorData.accelXLSB)) / 4U;
        yData = ((int16_t)((uint16_t)((uint16_t)sensorData.accelYMSB << 8) | (uint16_t)
    sensorData.accelYLSB)) / 4U;

        /* Convert raw data to angle (normalize to 0-90 degrees). No negative angles. */
        xAngle = (int16_t)floor((double)xData * (double)dataScale * 90 / 8192);
        yAngle = (int16_t)floor((double)yData * (double)dataScale * 90 / 8192);

        /* Print out the angle data. */
        PRINTF("x= %2d y = %2d\r\n", xAngle, yAngle);

        /***** Delay *****/
        for (volatile uint32_t i = 0; i < 500000; i++)
            __asm("NOP");
    }
}
```

11. After compiling and downloading the code, the board will print its angles in space around its x-axis and y-axis. Rotate the board while the code is running and monitor the change of the angle according to the rotations.

Problem 5

There is another example in the SDK which creates an e-compass (demo_apps − > ecompass_peripheral). This example utilize the same sensor which works as an accelerometer and a magnetometer at the same time. Accordingly, you need to port this example on FMUK66 board similar to Experiment 4B. Remember that this is the same module with the one we used in the recent experiment, thus, no need to change pin setups.