Lab 1: Bare-Metal I/O (Sensors and Actuators) Group 10

COMPENG 4DS4: Embedded Systems

Professor: Mohamed Hassan

Due: Wednesday February 7th, 2024

Name: Cameron Beneteau

Email: beneteac@mcmaster.ca

Student Number: 400274092

Name: Shaqif Ahmed

Email: ahmes80@mcmaster.ca

Student Number: 400263303

Declaration of Contributions

Tasks	Contributor(s)
Experiment 1: Control DC motor	Cameron Beneteau & Shaqif Ahmed
Problem 1	
Experiment 2: UART Communication	
Problem 2	
Experiment 3: UART Communication with Interrupts	
Problem 3	
Experiment 4: Interface with the Accelerometer Sensor	
Experiment Setup: Part A	
Problem 4	
Experiment Setup: Part B	
Problem 5	
Lab Report	

GitHub Repository

https://github.com/COMPENG-4DS4-Winter2024/bare-metal-i-o-sensors-and-actuators-group-10

Overview

This lab focuses on bare-metal I/O with an emphasis on sensors and actuators. The main objectives are to control DC and servo motors using PWM, communicate wirelessly with the host computer using UART and telemetry modules, learn the basics of SPI communication protocol, and interface with onboard sensors like the accelerometer and magnetometer.

Experiment 1: Control DC Motor

Utilizing PWM, the speed and direction of a DC motor were controlled. The ESC (Electronic Speed Controller) translates the PWM signal into motor speed adjustments based on its duty cycle. The setup involved connecting the motor to the FMUK66 board and adjusting the duty cycle to control the motor's speed and direction. The following table shows observations of the motors speed and direction at different duty cycle intervals.

Duty Cycle	Speed Observation	Direction Observation
0	Very fast	Reverse
10	Slower than 0	Reverse
20	Slower than 10	Reverse
30	Much slower than 20	Reverse
40	Similar speed to 30	Forward
50	Faster than 40	Forward
60	Faster than 50	Forward
70	Much faster than 60	Forward
80	Faster than 70	Forward
90	Faster than 80	Forward
100	Faster than 90	Forward

-10	Similar speed to 100	Reverse
-20	Faster than -10	Reverse
-30	Faster than -20	Reverse
-40	Similar speed to -30	Reverse
-50	Similar speed to -40	Reverse
-60	Similar speed to -50	Reverse
-70	Similar speed to -60	Reverse
-80	Similar speed to -70	Reverse
-90	Similar speed to -80	Reverse
-100	Similar speed to -90	Reverse

Note: The observed values for motor speed and direction across the different duty cycle intervals are not indicative of desired behaviour. Adjustments, including an offset correction, will be applied to tune the motor's response appropriately. It's important to note that each motor may operate within a slightly different range than the specified 5-10% duty cycle mentioned in the lab document. This is comparable to the tune dial seen on remote-control transmitters which is used to offset the PWN cycle to achieve expected steering behaviour.

Problem 1

The task was to control the car's front wheels by manipulating the servo motor's angle through PWM, like the DC motor control experiment. The challenge included calculating the appropriate duty cycle for the servo motor to achieve the desired wheel angles. The following equation was developed for normalizing the PWN ranges between 5% to 10% of the servo. Rather than 0.075, a bias of 0.0615 was used for the motor to achieve expected functionality.

$$\text{Duty} = \frac{\text{Input} \cdot 0.025}{100} + 0.075$$

```
printf("Input motor speed\n");
scanf("%d", &inputMotor);
printf("Input servo angle\n");
scanf("%d", &inputServo);

dutyCycleMotor = inputMotor * 0.025f / 100.0f + 0.0615;
dutyCycleServo = inputServo * 0.025f / 100.0f + 0.075;
```

```
frdmk66f_experiment_1 JLink Debug [GDB SEGGER Interface Debugging]
Executed SetRestartOnClose=1
[MCUXpresso Semihosting Telnet console for 'frdmk66f_experiment_1 JLink Debug' started on port 57541 @ 127.0.0.1]

SEGGER J-Link GDB Server V7.94b - Terminal output channel
Input motor speed
40
Input servo angle
80
```

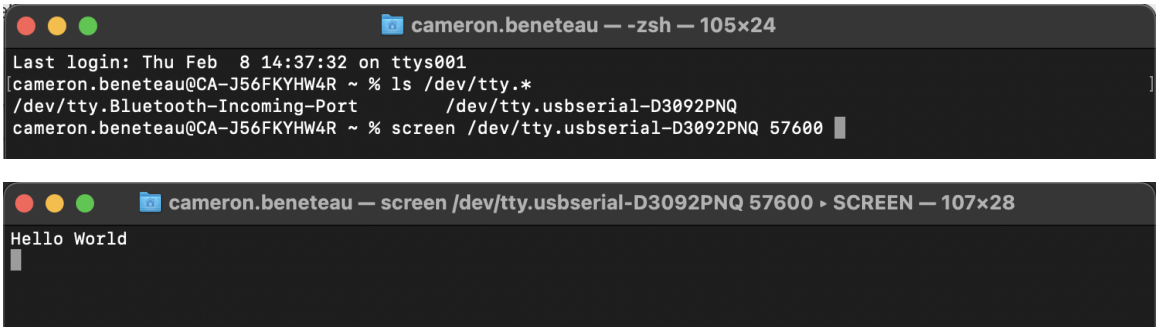
Experiment 2: UART Communication

This experiment introduced wireless data transmission between the FMUK66 board and a computer using UART via telemetry modules. The setup required configuring the UART module on the FMUK66 and establishing a connection to a computer, enabling data to be sent and received wirelessly.

Problem 2

The task extended the UART communication setup from Experiment 2 to control the DC motor and servo motor's speed and angle through wireless UART commands, allowing for real-time adjustments without resetting the system.

Instead of using PuTTY or RealTerm, UART serial connection and reading was made using the following commands through the terminal on MacOS.



Throughout testing, it was found UART was most reliable and predictable when sending one character at a time. Therefore, the implementation seen below takes four characters, accounting for the entire range of –100 to 100. Other input examples could be –070 and 0050. Once the four characters for the motor speed are received, they are converted to an integer, and the same process is used for the servo angle. Using these two values, like problem 1, the PWM duty cycles are calculated and updated accordingly.

```c
while (1)
{
    char motorStr[5] = "0000";
    int motorCharIndex = 0;

    printf("\nEnter motor speed | Range -100 to 100 | Format XXXX\n");

    for (int i = 0; i < 4; i++)
    {
        UART_ReadBlocking(TARGET_UART, &ch, 1);
        printf("%c", ch);
        motorStr[i] = ch;
    }

    int motorInt = atoi(motorStr);

    char servoStr[5] = "0000";
    int servoCharIndex = 0;

    printf("\nEnter servo angle | Range -100 to 100 | Format XXXX\n");

    for (int i = 0; i < 4; i++)
    {
        UART_ReadBlocking(TARGET_UART, &ch, 1);
        printf("%c", ch);
        servoStr[i] = ch;
    }

    int servoInt = atoi(servoStr);

    dutyCycleMotor = motorInt * 0.025f / 100.0f + 0.0615;
    dutyCycleServo = servoInt * 0.025f / 100.0f + 0.075;

    updatePWM_dutyCycle(FTM_CHANNEL_DC_MOTOR, dutyCycleMotor);
    updatePWM_dutyCycle(FTM_CHANNEL_SERVO, dutyCycleServo);

    FTM_SetSoftwareTrigger(FTM_MOTOR, true);

    printf("\nUpdated motor and servo values\n");
}
```

```
frdmk66f_lab_1_experiment_2 JLink Debug [GDB SEGGER Interface Debugging]
Executed SetRestartOnClose=1
[MCUXpresso Semihosting Telnet console for 'frdmk66f_lab_1_experiment_2 JLink Debug' started on port 57593 @ 127.0.0.1]

SEGGER J-Link GDB Server V7.94b - Terminal output channel
Hello World

Enter motor speed | Range -100 to 100 | Format XXXX
0040
Enter servo angle | Range -100 to 100 | Format XXXX
-100
Updated motor and servo values

Enter motor speed | Range -100 to 100 | Format XXXX
```

Experiment 3: UART Communication with Interrupts

Building on Experiment 2, this experiment incorporated interrupts in UART communication to enhance the system's responsiveness. The setup involved modifying the UART configuration to trigger interrupts upon receiving data, enabling immediate processing of incoming commands.

Problem 3

The challenge was to adapt the code from Problem 2 to utilize interrupts for UART communication, improving the system's ability to receive and execute commands in real-time.

The exact same process to receive, calculate, and update the PWN duty cycles as problems 1 and 2 was used. The only difference in this problem is the UART_ReadBlocking function

which waits for a character to be received was replaced by a while loop. The inner code to receive the character only runs when UART receives an interrupt.

```
volatile char ch;
volatile int new_char = 0;
```

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

```
while (1)
{
    char motorStr[5] = "0000";
    int motorCharIndex = 0;

    printf("\nEnter motor speed | Range -100 to 100 | Format XXXX\n");

    for (int i = 0; i < 4; i++)
    {

        while (1)
        {
            if (new_char)
            {
                new_char = 0;
                printf("%c", ch);
                motorStr[i] = ch;
                break;
            }
        }
    }

    int motorInt = atoi(motorStr);

    char servoStr[5] = "0000";
    int servoCharIndex = 0;

    printf("\nEnter servo angle | Range -100 to 100 | Format XXXX\n");

    for (int i = 0; i < 4; i++)
    {

        while (1)
        {
            if (new_char)
            {
                new_char = 0;
                printf("%c", ch);
                servoStr[i] = ch;
                break;
            }
        }
    }

    int servoInt = atoi(servoStr);

    dutyCycleMotor = motorInt * 0.025f / 100.0f + 0.0615;
    dutyCycleServo = servoInt * 0.025f / 100.0f + 0.075;

    updatePWM_dutyCycle(FTM_CHANNEL_DC_MOTOR, dutyCycleMotor);
    updatePWM_dutyCycle(FTM_CHANNEL_SERVO, dutyCycleServo);

    FTM_SetSoftwareTrigger(FTM_MOTOR, true);

    printf("\nUpdated motor and servo values\n");
}
```

```
frdmk66f_lab_1_problem_3 JLink Debug [GDB SEGGER Interface Debugging]
Executed SetRestartOnClose=1
[MCUXpresso Semihosting Telnet console for 'frdmk66f_lab_1_problem_3 JLink Debug' started on port 57640 @ 127.0.0.1]

SEGGER J-Link GDB Server V7.94b - Terminal output channel
Hello World

Enter motor speed | Range -100 to 100 | Format XXXX
0040
Enter servo angle | Range -100 to 100 | Format XXXX
-100
Updated motor and servo values

Enter motor speed | Range -100 to 100 | Format XXXX
```

Experiment 4: Interface with the Accelerometer Sensor

This experiment focused on interfacing with the onboard accelerometer sensor via the SPI communication protocol. The task involved configuring the SPI module and reading data from the accelerometer to understand the sensor's behaviour and data output.

Experiment Setup: Part A

The initial setup for interfacing with the accelerometer involved basic SPI configuration and reading a register from the sensor to verify communication.

Problem 4

The task was to implement a function for writing data to the accelerometer's registers using SPI, expanding the ability to interact with the sensor beyond just reading its data.

Using corresponding datasheets and resources, the function was implemented below. The first bit of masterTxData was set to high (or 1) indicating a write operation. The next 7 least significant bits were cleared and the value to be written was added. After, the same DSPI_MasterTransferBlocking function from SPI_read was used to transfer data between the master and slave to the specified register address. Finally, the allocated memory is freed, and the status of the transfer is returned.

```c
status_t SPI_write(uint8_t regAddress, uint8_t value)
{

    dspi_transfer_t masterXfer;
    uint8_t *masterTxData = (uint8_t *)malloc(sizeof(typeof(value)) + 2);

    masterTxData[0] = regAddress | 0x80; // Set the most significant bit
    masterTxData[1] = regAddress & 0x80; // Clear the least significant 7 bits
    masterTxData[2] = value;             // Add value to stream

    masterXfer.txData = masterTxData;
    masterXfer.rxData = NULL;
    masterXfer.dataSize = sizeof(typeof(value)) + 2;
    masterXfer.configFlags = kDSPI_MasterCtar0 | kDSPI_MasterPcs0 | kDSPI_MasterPcsContinuous;
    status_t ret = DSPI_MasterTransferBlocking(SPI1, &masterXfer);

    free(masterTxData);

    return ret;
}
```

```c
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");

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

    SPI_write(0x14, 0x66);
    SPI_read(0x14, &byte, 1);
    printf("The expected value is 0x66 and the read value 0x%X\n", byte);

    SPI_write(0x14, 0x88);
    SPI_read(0x14, &byte, 1);
    printf("The expected value is 0x88 and the read value 0x%X\n", byte);

    while (1)
    {
    }
}
```

```
frdmk66f_lab_1_problem_4 JLink Debug [GDB SEGGER Interface Debugging]
Executed SetRestartOnClose=1
[MCUXpresso Semihosting Telnet console for 'frdmk66f_lab_1_problem_4 JLink Debug' started on port 57629 @ 127.0.0.1]

SEGGER J-Link GDB Server V7.94b - Terminal output channel
The expected value is 0xC7 and the read value 0xC7
The expected value is 0x66 and the read value 0x66
The expected value is 0x88 and the read value 0x88
```

Experiment Setup: Part B

The task was to implement a function for writing data to the accelerometer's registers using SPI instead of I2C, expanding the ability to interact with the sensor beyond just reading its data.

Problem 5

Similar to Problem 4, this challenge involved porting another SDK example, an e-compass application, to the FMUK66 board. The task required adapting the example to utilize both the accelerometer and magnetometer functions of the sensor through SPI communication instead of I2C. The values obtained from the magnometer and accelerometer are used to determine the angle of the compass in terms of change in the yaw direction, as can be seen in the pictures below.

```c
int main(void)
{
    fxos_config_t config = {0};
    status_t result;
    uint16_t i = 0;
    uint16_t loopCounter = 0;
    double sinAngle = 0;
    double cosAngle = 0;
    double Bx = 0;
    double By = 0;
    uint8_t array_addr_size = 0;

    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();
    BOARD_InitPeripherals();

    HW_Timer_init();

    voltageRegulatorEnable();
    accelerometerEnable();

    setupSPI();

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

```
frdmk66f_lab_1_problem_5 JLink Debug (1) [GDB SEGGER Interface Debugging]
Executed SetRestartOnClose=1
[MCUXpresso Semihosting Telnet console for 'frdmk66f_lab_1_problem_5 JLink Debug (1)' started on port 57668 @ 127.0.0.1]

SEGGER J-Link GDB Server V7.94b - Terminal output channel

To calibrate Magnetometer, roll the board on all orientations to get max and min values

Press any key to start calibrating...
c

Calibrating magnetometer...
Calibrate magnetometer successfully!
Magnetometer offset Mx: -1733 - My: -3223 - Mz: -3053
```

```
Compass Angle: 42.6
Compass Angle: 42.8
Compass Angle: 43.2
Compass Angle: 44.0
Compass Angle: 44.5
Compass Angle: 44.0
Compass Angle: 42.0
Compass Angle: 41.5
Compass Angle: 34.9
Compass Angle: 28.2
Compass Angle: 17.3
Compass Angle: 5.6
Compass Angle: -5.2
Compass Angle: -15.7
Compass Angle: -23.8
Compass Angle: -31.2
Compass Angle: -20.2
Compass Angle: -3.0
Compass Angle: 9.5
Compass Angle: 21.2
Compass Angle: 31.2
Compass Angle: 39.8
Compass Angle: 46.6
Compass Angle: 52.6
Compass Angle: 58.0
Compass Angle: 62.7
Compass Angle: 66.9
Compass Angle: 70.7
Compass Angle: 73.7
Compass Angle: 75.8
Compass Angle: 77.4
```