

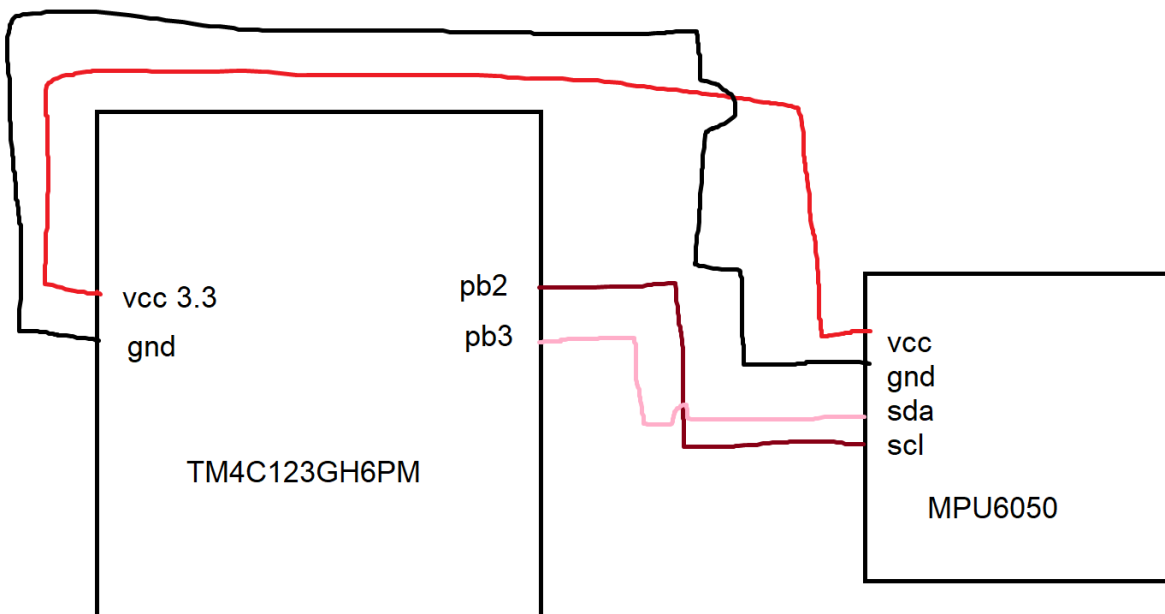
Date Submitted: 10/26/19

Goal: The goal of this midterm is to interface the MPU6050 with i2c to communicate with the TIVA C. In task 1, students will communicate the gyroscopic and acceleration values to the TIVA C then display those values using UART and in graph form in task 2. In task 3 & 4, we will add a complimentary filter using IQmath to the raw gyroscopic and acceleration values which we will display to the UART terminal and graph.



Task 01:

Youtube Link: <https://youtu.be/x3mtSjX5cBo>

Modified Schematic (if applicable):



Screenshot of output:

 <Closed> COM5 

Acc. X: -13345 | Acc. Y: -1997 | Acc. Z: 13584
 Gyro. X: -26 | Gyro. Y: -2 | Gyro. Z: 17

Acc. X: -13407 | Acc. Y: -1983 | Acc. Z: 13699
 Gyro. X: -14 | Gyro. Y: 0 | Gyro. Z: 20

Acc. X: -13249 | Acc. Y: -1973 | Acc. Z: 13450
 Gyro. X: -12 | Gyro. Y: Acc. X: 13785 | Acc. Y: -1714 | Acc. Z: 13824
 Gyro. X: 50 | Gyro. Y: 18 | Gyro. Z: 43

Acc. X: -13685 | Acc. Y: -1978 | Acc. Z: 13009
 Gyro. X: -16 | Gyro. Y: 0 | Gyro. Z: 15

Acc. X: -33721 | Acc. Y: -7381 | Acc. Z: 23811
 Gyro. X: 4365 | Gyro. Y: 2832 | Gyro. Z: 683

Acc. X: -10518 | Acc. Y: -4732 | Acc. Z: 9795
 Gyro. X: 587 | Gyro. Y: 960 | Gyro. Z: 1037

Acc. X: -14164 | Acc. Y: -239 | Acc. Z: 13014
 Gyro. X: -21 | Gyro. Y: -2 | Gyro. Z: 19

Acc. X: -14039 | Acc. Y: -296 | Acc. Z: 12870
 Gyro. X: -17 | Gyro. Y: 0 | Gyro. Z: 20

Acc. X: -14097 | Acc. Y: -354 | Acc. Z: 12827
 Gyro. X: -9 | Gyro. Y: 0 | Gyro. Z: 20

Acc. X: -13962 | Acc. Y: -301 | Acc. Z: 12923
 Gyro. X: -13 | Gyro. Y: -3 | Gyro. Z: 20

Modified Code:**int main(void)**

```
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
```

```

#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.h"
#include "driverlib/uart.h"

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536
#define SAMPLE_RATE 0.01

void ConfigureUART(void) { //config uart to output accel & gyro values
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
}

//
// A boolean that is set when a MPU6050 command has completed.
//
volatile bool g_bMPU6050Done;

//
// I2C master instance
//
tI2CInstance g_sI2CMSimpleInst;

//
// The function that is provided by this example as a callback when MPU6050
// transactions have completed.
//
void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    //
    // See if an error occurred.
    //
    if (ui8Status != I2CM_STATUS_SUCCESS)
    {
        //
        // An error occurred, so handle it here if required.
        //
    }
    //
    // Indicate that the MPU6050 transaction has completed.
    //
    g_bMPU6050Done = true;
}

```

```

}

void InitI2C0(void)
{
    //enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    //enable GPIO peripheral that contains I2C 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    // Enable and initialize the I2C0 master module. Use the system clock for
    // the I2C0 module.
    // I2C data transfer rate set to 400kbps.
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

    //clear I2C FIFOs
    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;

    // Initialize the I2C master driver.
    I2CMInit(&g_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet());
}

//
// The interrupt handler for the I2C module.
//
void I2CMSimpleIntHandler(void)
{
    //
    // Call the I2C master driver interrupt handler.
    //
    I2CMIntHandler(&g_sI2CMSimpleInst);
}

void delayMS(int ms) {
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*ms ); // less accurate
}

void MPU6050Code(void) {

    float fAccel[3], fGyro[3];
    float aAccel = 0, bAccel = 0, cAccel = 0;
    float aGyro = 0, bGyro = 0, cGyro = 0;
    tMPU6050 sMPU6050;

```

```

//
// Initialize the MPU6050. This code assumes that the I2C master instance
// has already been initialized.
//
g_bMPU6050Done = false;
MPU6050Init(&sMPU6050, &g_sI2CSimpleInst, 0x68, MPU6050Callback, &sMPU6050);
while (!g_bMPU6050Done);
//
// Configure the MPU6050 for +/- 4 g accelerometer range.
//

//Settings for the Accelerometer
g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_ACCEL_CONFIG, // Accelerometer configuration
                        0xFF, // ~MPU6050_ACCEL_CONFIG_AFS_SEL_M, // No need to mask
                        MPU6050_ACCEL_CONFIG_AFS_SEL_4G, // Accelerometer full-
scale range 4g
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_GYRO_CONFIG, // Gyroscope configuration
                        0xFF, // No need to mask
                        MPU6050_GYRO_CONFIG_FS_SEL_250, // Gyro full-scale range
+/- 250 degrees/sec
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_PWR_MGMT_1, // Power management 1 register
                        0x00,
                        0x00, // 0x02 & MPU6050_PWR_MGMT_1_DEVICE_RESET,
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_PWR_MGMT_2, // Power management 2 register
                        0x00,
                        0x00,
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

//
// Loop forever reading data from the MPU6050. Typically, this process
// would be done in the background, but for the purposes of this example,
// it is shown in an infinite loop.
//

```

```

while (1)
{
    //
    // Request another reading from the MPU6050.
    //

    g_bMPU6050Done = false;
    MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {

    }
    //
    // Get the new accelerometer and gyroscope readings.
    //
    MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
    MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);
    //
    // Do something with the new accelerometer and gyroscope readings.
    //
    aGyro = fGyro[0] * 1000; //multiply by 1000 to see the value better
    bGyro = fGyro[1] * 1000;
    cGyro = fGyro[2] * 1000;

    aAccel = fAccel[0] * 1000;
    bAccel = fAccel[1] * 1000;
    cAccel = fAccel[2] * 1000;

    UARTprintf("Acc. X: %d | Acc. Y: %d | Acc. Z: %d\n", (int)aAccel,
(int)bAccel, (int)cAccel); //print data
    UARTprintf("Gyro. X: %d | Gyro. Y: %d | Gyro. Z: %d\n", (int)aGyro,
(int)bGyro, (int)cGyro);
    UARTprintf("\n");
    delayMS(1000);
}
}

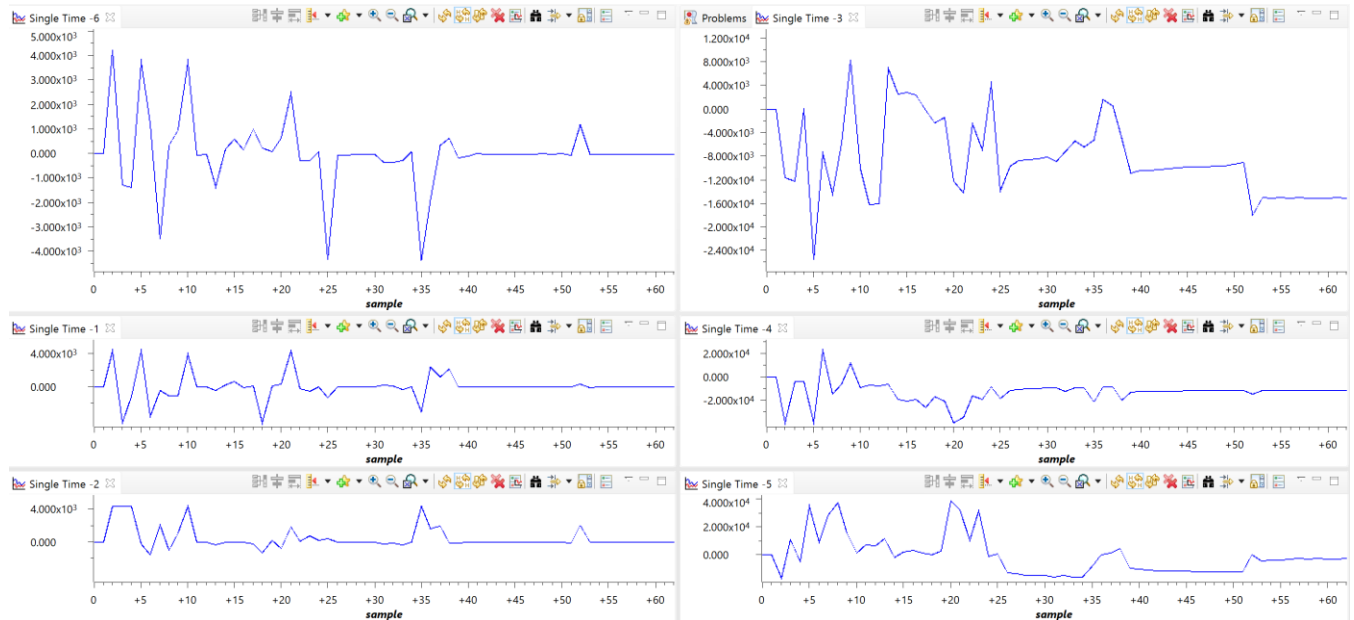
int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    InitI2C0();
    ConfigureUART();
    MPU6050Code();

    while(1)
    {
    }
}

```

Task 02:

Youtube Link: <https://youtu.be/pMkw7X2Cd8k>



The first column from top to bottom is:

aGyro, bGyro, cGyro

The second column from top to bottom is:

aAccel, bAccel, cAccel

Modified Code:

The same code as task 1

Task 03:

```

Acc. X: -354 | Acc. Y: 19969 | Acc. Z: -62
Gyro. X: -26 | Gyro. Y: -15 | Gyro. Z: -52
Pitch: 0 Roll: 90
Acc. X: 153 | Acc. Y: 20089 | Acc. Z: -215
Gyro. X: -13 | Gyro. Y: 27 | Gyro. Z: -102
Pitch: 0 Roll: 90
Acc. X: 3602 | Acc. Y: 18029 | Acc. Z: -2088
Gyro. X: -286 | Gyro. Y: -238 | Gyro. Z: 437
Pitch: 123 Roll: 96
Acc. X: 17660 | Acc. Y: 8277 | Acc. Z: 3151
Gyro. X: -790 | Gyro. Y: -108 | Gyro. Z: 926
Pitch: 79 Roll: 69
Acc. X: 21253 | Acc. Y: 2600 | Acc. Z: 2826
Gyro. X: 199 | Gyro. Y: -176 | Gyro. Z: 317
Pitch: 84 Roll: 45
Acc. X: 20161 | Acc. Y: 967 | Acc. Z: 2203
Gyro. X: -29 | Gyro. Y: -8 | Gyro. Z: -20
Pitch: 84 Roll: 0
Acc. X: 20367 | Acc. Y: 900 | Acc. Z: 2136
Gyro. X: -32 | Gyro. Y: -21 | Gyro. Z: -17

```

Youtube Link: <https://youtu.be/sIMX9TLPqHE>

Modified Code:

```

#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "uartstdio.h"
#include "driverlib/uart.h"
#include "math.h"

```



```

#include "IQmathLib.h"

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536
#define SAMPLE_RATE 0.01
#define dt 0.01 // 10 ms sample rate!

void ComplementaryFilter(short accData[3], short gyrData[3], float *pitch, float
*roll)
{
    float pitchAcc, rollAcc;
    // Integrate the gyroscope data -> int(angularSpeed) = angle
    // Angle around the X-axis
    *pitch += ((float)gyrData[0] / GYROSCOPE_SENSITIVITY) * dt;
    // Angle around the Y-axis
    *roll += ((float)gyrData[1] / GYROSCOPE_SENSITIVITY) * dt;
    // Compensate for drift with accelerometer data
    // Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192
    int forceMagnitudeApprox = abs(accData[0]) + abs(accData[1]) + abs(accData[2]);
    if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)
    {
        // Turning around the X axis results in a vector on the Y-axis
        pitchAcc = atan2f((float)accData[1], (float)accData[2]) * 180 / M_PI;
        *pitch = *pitch * 0.98 + pitchAcc * 0.02;
        // Turning around the Y axis results in a vector on the X-axis
        rollAcc = atan2f((float)accData[0], (float)accData[2]) * 180 / M_PI;
        *roll = *roll * 0.98 + rollAcc * 0.02;
    }
}

void ConfigureUART(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
}

volatile bool g_bMPU6050Done;

tI2CInstance g_sI2CMSimpleInst;

void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    if (ui8Status != I2CM_STATUS_SUCCESS) {}
    g_bMPU6050Done = true;
}

void InitI2C0(void)
{
    //enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
}

```

```

//reset module
SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

//enable GPIO peripheral that contains I2C 0
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

// Configure the pin muxing for I2C0 functions on port B2 and B3.
GPIOPinConfigure(GPIO_PB2_I2C0SCL);
GPIOPinConfigure(GPIO_PB3_I2C0SDA);

// Select the I2C function for these pins.
GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

// Enable and initialize the I2C0 master module. Use the system clock for
// the I2C0 module.
// I2C data transfer rate set to 400kbps.
I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

//clear I2C FIFOs
HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;

// Initialize the I2C master driver.
I2CInit(&g_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet());
}

void I2CMSimpleIntHandler(void)
{
    I2CIntHandler(&g_sI2CMSimpleInst);
}

void delayMS(int ms) {
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*ms ); // less accurate
}

int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    InitI2C0();
    ConfigureUART();

    float pitch, roll, tempPitch, tempRoll;
    float fAccel[3], fGyro[3];
    float xAccel = 0, yAccel = 0, zAccel = 0;
    float xGyro = 0, yGyro = 0, zGyro = 0;
    iq16 qAccel[3], qGyro[3];

    tMPU6050 sMPU6050;
    //
    // Initialize the MPU6050. This code assumes that the I2C master instance
    // has already been initialized.
    //
    g_bMPU6050Done = false;

```

```

MPU6050Init(&sMPU6050, &g_sI2C.SimpleInst, 0x68, MPU6050Callback, &sMPU6050);
while (!g_bMPU6050Done);
//
// Configure the MPU6050 for +/- 4 g accelerometer range.
//

//Settings for the Accelerometer
g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_ACCEL_CONFIG, // Accelerometer configuration
                        0xFF, // ~MPU6050_ACCEL_CONFIG_AFS_SEL_M, // No need to mask
                        MPU6050_ACCEL_CONFIG_AFS_SEL_4G, // Accelerometer full-
scale range 4g
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_GYRO_CONFIG, // Gyroscope configuration
                        0xFF, // No need to mask
                        MPU6050_GYRO_CONFIG_FS_SEL_250, // Gyro full-scale range
+/- 250 degrees/sec
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_PWR_MGMT_1, // Power management 1 register
                        0x00,
                        0x00, // 0x02 & MPU6050_PWR_MGMT_1_DEVICE_RESET,
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

g_bMPU6050Done = false;
MPU6050ReadModifyWrite(&sMPU6050,
                        MPU6050_O_PWR_MGMT_2, // Power management 2 register
                        0x00,
                        0x00,
                        MPU6050Callback,
                        &sMPU6050);
while (!g_bMPU6050Done);

//
// Loop forever reading data from the MPU6050. Typically, this process
// would be done in the background, but for the purposes of this example,
// it is shown in an infinite loop.
//
while (1)
{
    //
    // Request another reading from the MPU6050.
    //

```

```

g_bMPU6050Done = false;
MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
while (!g_bMPU6050Done)
{
    //
    // Get the new accelerometer and gyroscope readings.
    //
    MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
    MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);
    //
    // Do something with the new accelerometer and gyroscope readings.
    //
    xAccel = fAccel[0] * 1000;
    yAccel = fAccel[1] * 1000;
    zAccel = fAccel[2] * 1000;
    xGyro = fGyro[0] * 1000;
    yGyro = fGyro[1] * 1000;
    zGyro = fGyro[2] * 1000;

    short fAccelShort[3];
    fAccelShort[0] = (short)fAccel[0];
    fAccelShort[1] = (short)fAccel[1];
    fAccelShort[2] = (short)fAccel[2];

    short fGryoShort[3];
    fGryoShort[0] = (short)fGyro[0];
    fGryoShort[1] = (short)fGyro[1];
    fGryoShort[2] = (short)fGyro[2];

    qAccel[0] = _IQ16(fAccel[0]);
    qAccel[1] = _IQ16(fAccel[1]);
    qAccel[2] = _IQ16(fAccel[2]);

    qGyro[0] = _IQ16(fGyro[0]);
    qGyro[1] = _IQ16(fGyro[1]);
    qGyro[2] = _IQ16(fGyro[2]);

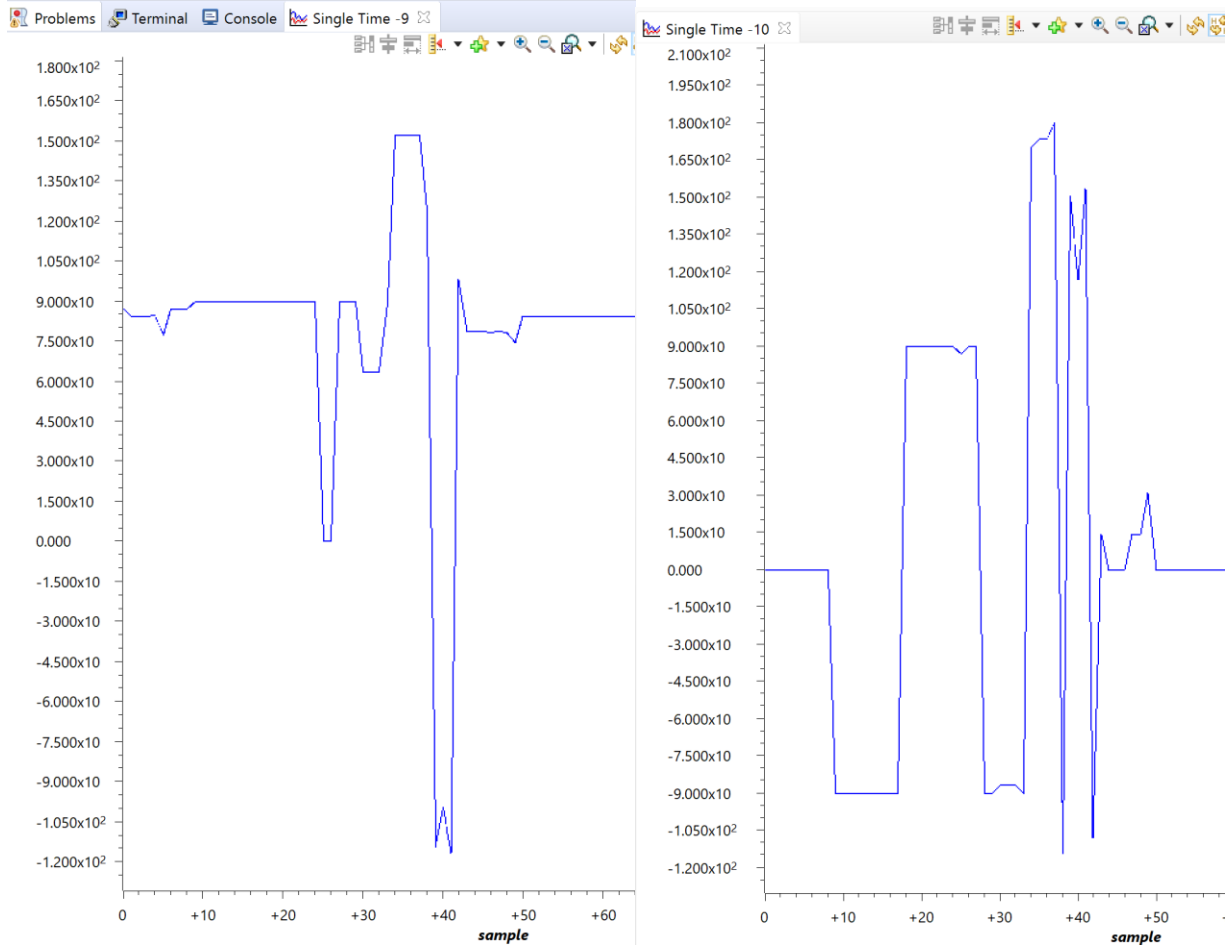
    ComplementaryFilter(&qAccel[0], &qGyro[0], &pitch, &roll); //filter values

    UARTprintf("Acc. X: %d | Acc. Y: %d | Acc. Z: %d\n", (int)xAccel,
(int)yAccel, (int)zAccel);
    UARTprintf("Gyro. X: %d | Gyro. Y: %d | Gyro. Z: %d\n", (int)xGyro,
(int)yGyro, (int)zGyro);

    UARTprintf("Roll: %d", (int)roll);
    UARTprintf("\n");
    //UARTprintf("test\n");
    delayMS(1000);
}
}

```

Task 04:



Left pic: pitch right pic: roll

Youtube Link: https://youtu.be/cjacm6v9m_o

Modified Schematic (if applicable):

Modified Code:

Same code as above

Conclusion:

I was able to complete all tasks. The first two tasks involved getting the raw gyro and accelerometer files from the mpu 6050 using i2c. The last two tasks involved filtering the raw gyro and accel values then print those using uart and in the graph.