**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):**

**A close up of a piece of paper

Description automatically generated**

**Screenshot of output:**

**A screenshot of a cell phone

Description automatically generated**

**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

//

tI2CMInstance 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\_sI2CMSimpleInst, 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>

**A screenshot of a computer

Description automatically generated**

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:**

A close up of text on a white background

Description automatically generated

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;

tI2CMInstance 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.

**I2CMInit**(&g\_sI2CMSimpleInst, I2C0\_BASE, INT\_I2C0, 0xff, 0xff, **SysCtlClockGet**());

}

**void** **I2CMSimpleIntHandler**(**void**)

{

**I2CMIntHandler**(&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\_sI2CMSimpleInst, 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);

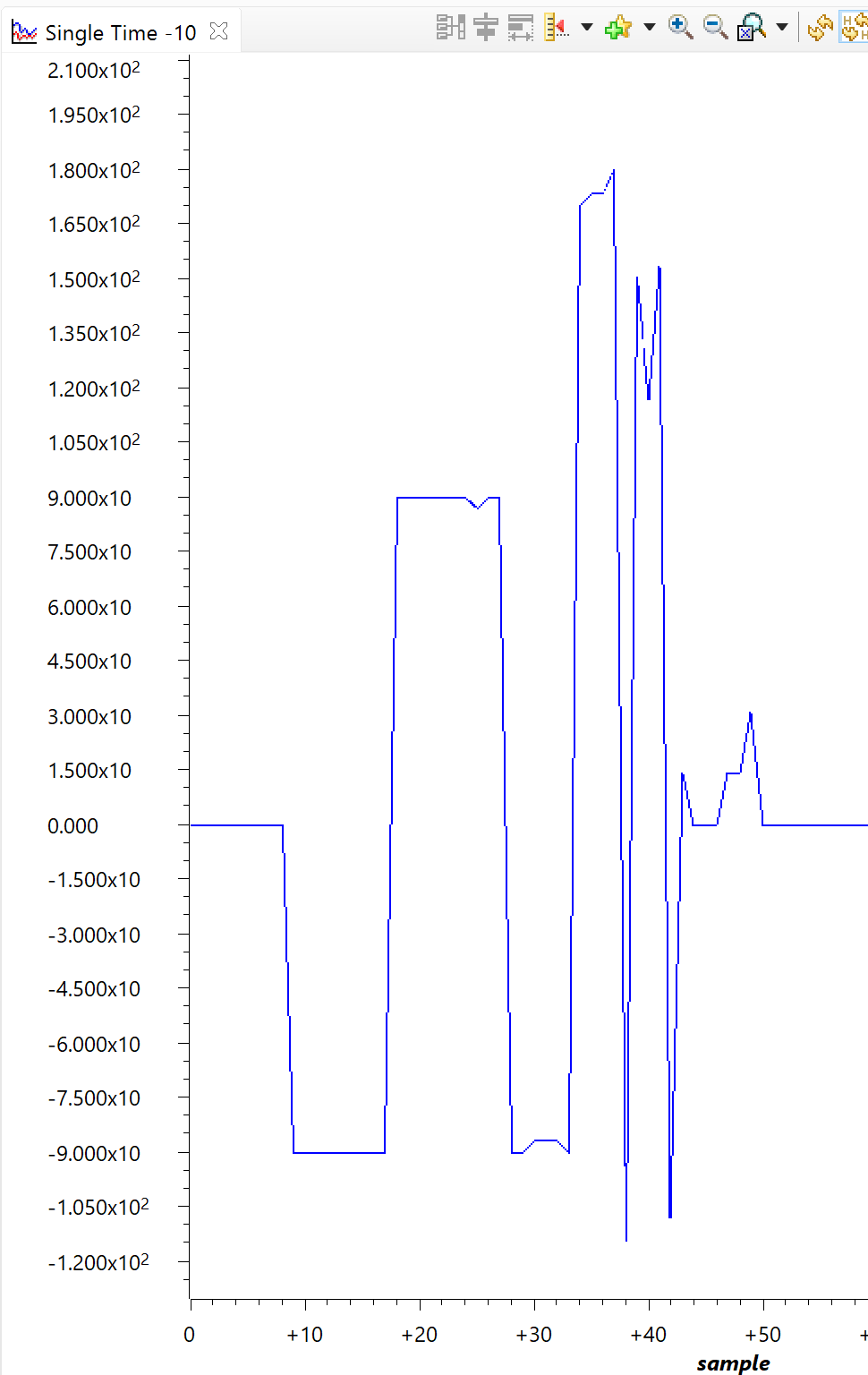
}

}

**------------------------------------------------------------------------------------**

**Task 04:**

A screenshot of a cell phone

Description automatically generated

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.**