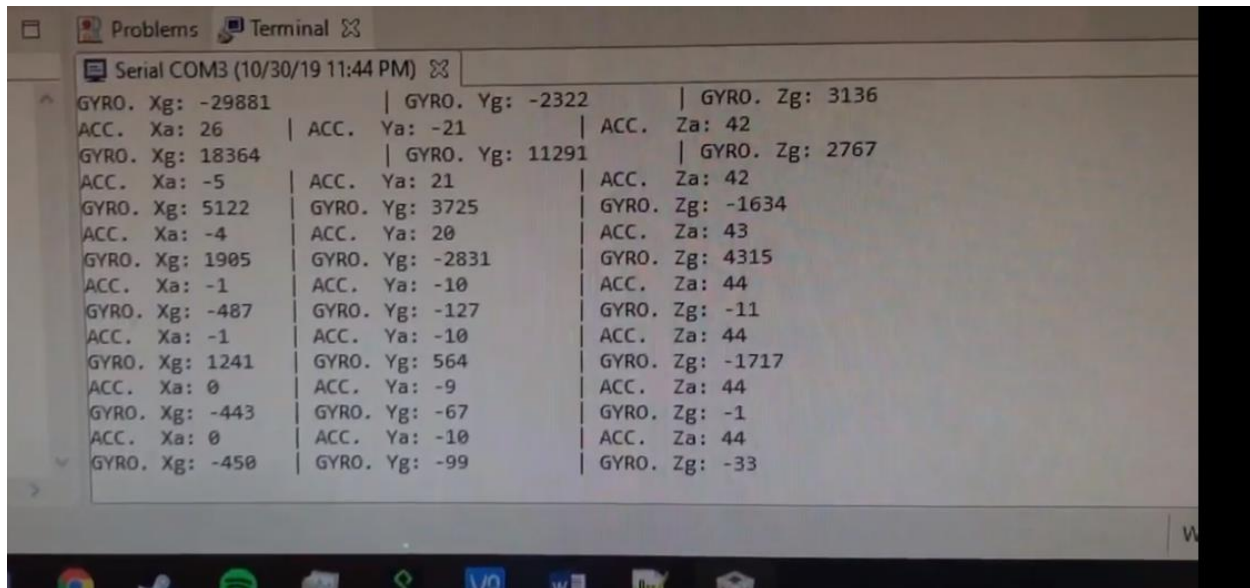


TITLE: MIDTERM I

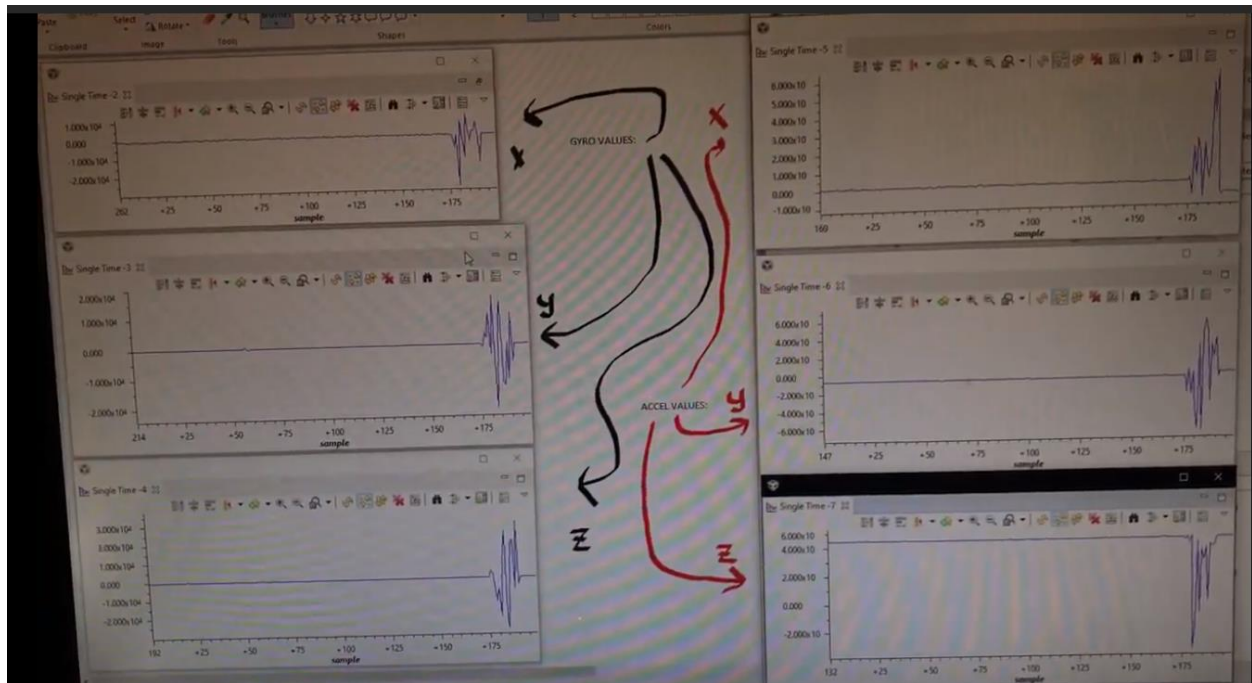
GOAL:

Task01&02: The goal of task 1 and two is to interface the given MPU6050 IMU using I2C protocol to TIVAC. The values will be printed on the serial terminal, and they can also be printed using the graph on code composer studio.

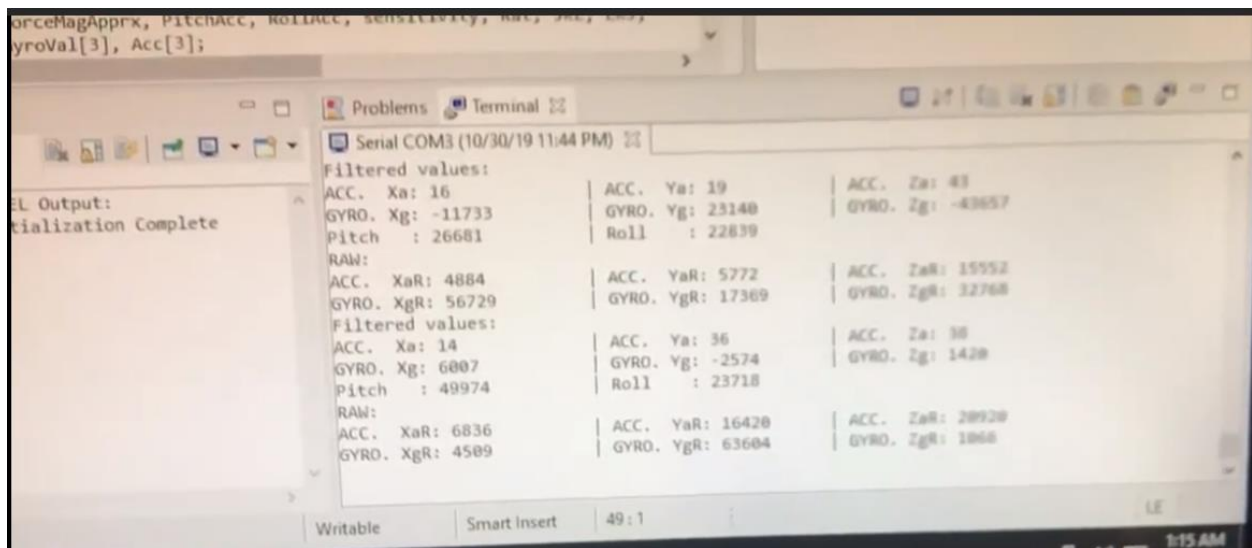


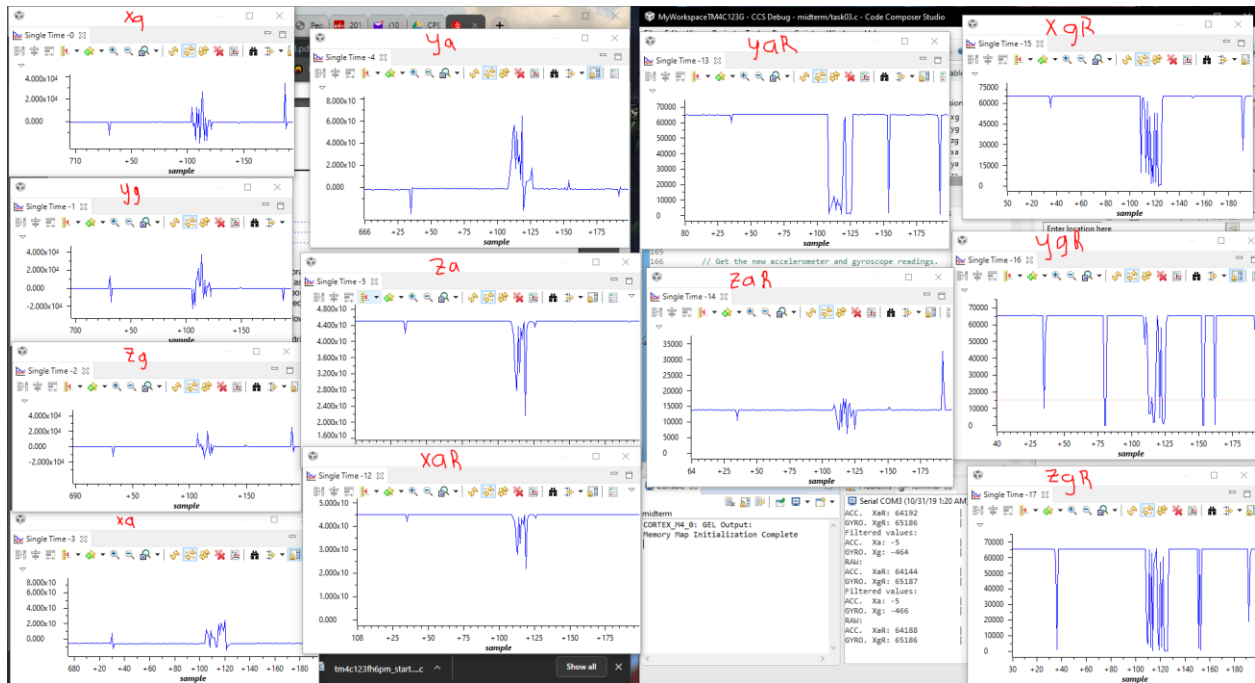
The screenshot shows a serial terminal window titled "Serial COM3 (10/30/19 11:44 PM)". The terminal displays a continuous stream of sensor data from an MPU6050 IMU, organized into three columns. Each row contains three lines of data: Gyroscope (GYRO) X, Y, and Z values, followed by Accelerometer (ACC) X, Y, and Z values. The data is printed in a monospaced font, and the terminal window is part of a larger IDE interface with tabs for "Problems" and "Terminal".

GYRO. Xg	GYRO. Yg	GYRO. Zg	ACC. Xa	ACC. Ya	ACC. Za
-29881	-2322	3136	26	-21	42
18364	11291	2767	-5	21	42
5122	3725	-1634	-4	20	43
1905	-2831	4315	-1	-10	44
-487	-127	-11	-1	-10	44
1241	564	-1717	0	-9	44
-443	-67	-1	0	-10	44
-450	-99	-33			



Task03&04: The goal of this task is to implement a complementary filter to filter the raw values from the accelerometer and gyro.





DELIVERABLES:

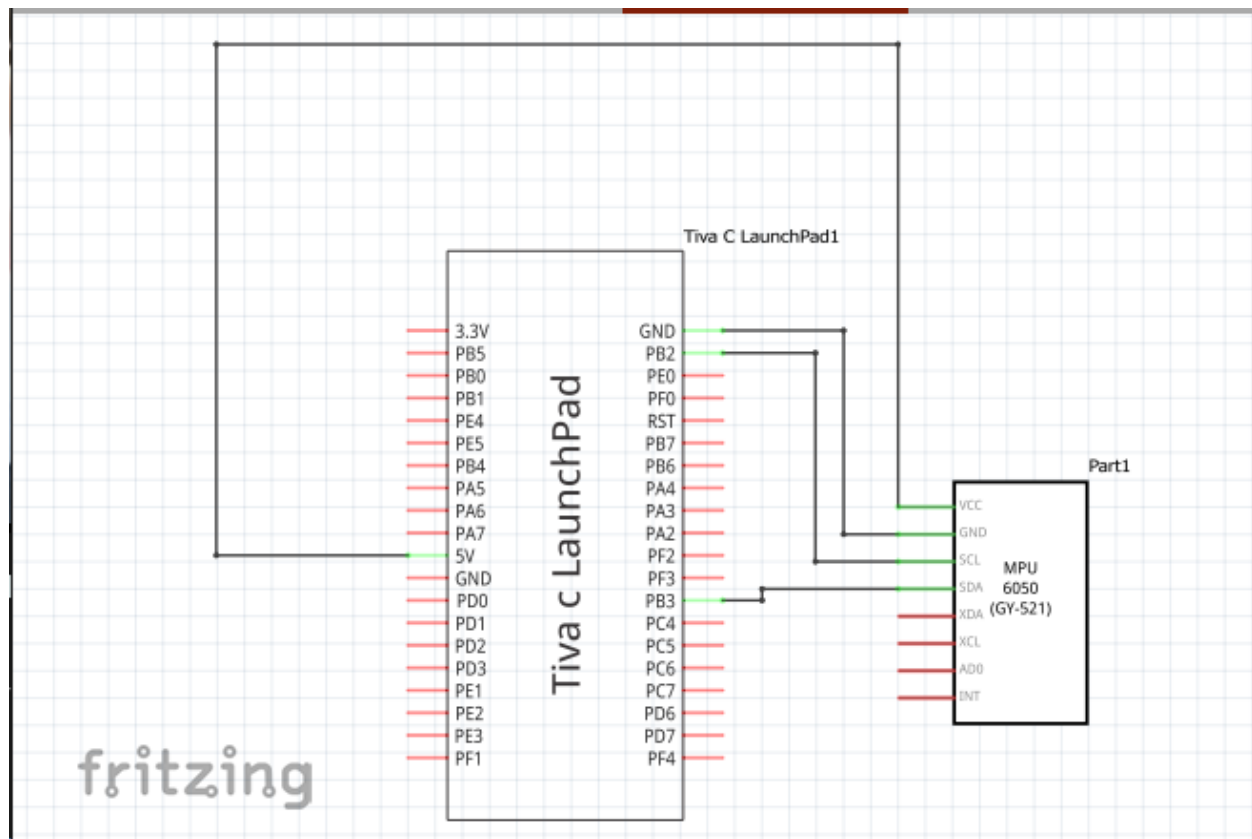
The deliverables include showing the accelerometer and gyro values through either the serial terminal or the the graphing tool on Code Composer Studio.

COMPONENTS:

Tiva C Series Launchpad Ek-TM4C123GXL

MPU6050

SCHEMATICS:



IMPLEMENTATION:

Step implemented in the code - for example initialization of I2C, UART, start reading one set of data, print - explain each subroutine.

I2C involves synchronizing the data transfer between two chips. Every data bit transferred on the SDA (Serial Data) line is synchronized by a high-to-low pulse of clock on the SCL (Serial Clock) line. The data line only changes only when the clock line is low. If a master, which controls the bus, wants to initiate a new transfer and does not want to release the bus before starting the new transfer, it issues a START condition between the pair of START and STOP conditions. Each packet is 9 bits long. The first 8 bits are put on the SDA by the transmitter. The ninth bit is an acknowledge by the receiver. There are two different types of packets: address packet and data packet.

Code:

Task 01_02:

```
/*
Author Name: Serak Gebremedhin
Midterm 1 Code
Date: 10/31/2019
```

```
*/
```

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

Goal: The goal of task 1 and two is to interface the given MPU6050 IMU using I2C protocol to TIVAC. The values will be printed on the serial terminal, and they can also be printed using the graph on code composer studio.

Steps: The raw values will pass the the complementary filter function, and will print on the serial terminal. To implement the filter, the IQMath library will be used.

```
*****
```

```
Chip type           : ARM TM4C123GH6PM
Program type        : Firmware
Core Clock frequency : 80.000000 MHz
```

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

```
#include <stdbool.h>
#include <stdint.h>
#include <math.h>
#include "utils/uartstdio.h"
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/i2c.h"
#include "driverlib/interrupt.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/i2cm_drv.c"
#include "sensorlib/mpu6050.h"
#include "utils/uartstdio.h"
```

```
#define ACCEL_SLAVE_ADDR 0x1D
#define XOUT8 0x06
#define YOUT8 0x07
#define ZOUT8 0x08
```

```

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 131
#define SAMPLE_RATE 0.01
#define RATIO (180/3.14)

volatile bool g_bMPU6050Done; //boolean that set when MPU6050 command is complete
tI2CInstance g_sI2CInst; // I2C master instance
int clockFreq; //Frequency of clock

//function prototypes
static void
MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status);
void I2CIntHandler(void);
void MPU6050Example(void);
void MPU6050Example(void);
void InitI2C0(void);
void ConfigUART(void);
void Complementary_Filter(float fAccel[], float fGyro[]);

int main()
{
    //Set the clocking to run directly from the external crystal/oscillator
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_INT |
SYSCTL_XTAL_16MHZ);
    //Configure UART to print to terminal
    ConfigUART();
    //initialize I2C module 0
    InitI2C0();
    //Get MPU6050 data
    MPU6050Example();

    return(0);
}

static void
MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
//callback when transactions are completed
{
    // 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 I2CIntHandler(void)
// I2C module interrupt handler
{
    I2CMIntHandler(&g_sI2CInst);
}

void MPU6050Example(void)

```

```

{
    float fAccel[3], fGyro[3];

    tMPU6050 sMPU6050;
    float xa = 0, ya = 0, za = 0;
    float xg = 0, yg = 0, zg = 0;

    //Initialized the MPU6050. This code assumes that the I2C master instance
    //has already been initialized.

    g_bMPU6050Done = false;
    // Initialize the MPU6050. This code assumes that the I2C master instance
    // has already been initialized.
    g_bMPU6050Done = false;
    MPU6050Init(&sMPU6050, &g_sI2CInst, 0x68, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Configure the MPU6050 for +/- 4 g accelerometer range.
    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_ACCEL_CONFIG,
~MPU6050_ACCEL_CONFIG_AFS_SEL_M,
                                MPU6050_ACCEL_CONFIG_AFS_SEL_4G, MPU6050Callback,
&sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0b00000010 &
MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00,
MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
    }

    // Loop forever reading data from the MPU6050.
    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.

    xg = fGyro[0]*10000;
    yg = fGyro[1]*10000;
    zg = fGyro[2]*10000;
    xa = (atan2(fAccel[0], sqrt(fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
    ya = (atan2(fAccel[1], sqrt(fAccel[0] * fAccel[0] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
    za = (atan2(fAccel[2], sqrt(fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;

    UARTprintf("ACC. Xa: %d \t | ACC. Ya: %d \t | ACC. Za: %d \n", (int)xa,
(int)ya, (int)za);
    UARTprintf("GYRO. Xg: %d \t | GYRO. Yg: %d \t | GYRO. Zg: %d \n", (int)xg,
(int)yg, (int)zg);
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*1000 );
}
}

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.
    I2CMinInit(&g_sI2CMinst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet());
}

void ConfigUART(void)

```



```

{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIOA peripherals(the UART
pins are on GPIO Port A)

    //Configure pins for the reciever and transmitter
    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);
}

```

Task03_04:

```

/*
Author Name: Serak Gebremedhin
Midterm 1 Code
Date: 10/31/2019

```

```

*/

```

```

/*****

```

Goal: The goal of this task is to implement a complementary filter to filter the raw values from the accelerometer and gyro.

Steps: The raw values will pass the the complementary filter function, and will print on the serial terminal. To implement the filter, the IQMath library will be used.

```

*****
Chip type           : ARM TM4C123GH6PM
Program type        : Firmware
Core Clock frequency : 80.000000 MHz
*****/
#include <stdbool.h>
#include <stdint.h>
#include <math.h>
#include <stdlib.h>

```

```

#include <stdio.h>
#include "utils/uartstdio.h"
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/i2c.h"
#include "driverlib/interrupt.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/i2cm_drv.c"
#include "sensorlib/mpu6050.h"
#include "utils/uartstdio.h"
#include "IQmath/IQmathLib.h"

#define ACCEL_SLAVE_ADDR 0x1D
#define XOUT8 0x06
#define YOUT8 0x07
#define ZOUT8 0x08

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 131
#define SAMPLE_RATE 0.01
#define RATIO (180/3.14)

volatile bool g_bMPU6050Done; //boolean that set when MPU6050 command is complete
tI2CInstance g_sI2CInst; // I2C master instance
int clockFreq; //Frequency of clock

//function prototypes
static void
MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status);
void I2CIntHandler(void);
void MPU6050Example(void);
void MPU6050Example(void);
void InitI2C0(void);
void ConfigUART(void);
void Complementary_Filter(float fAccel[], float fGyro[]);

int main()
{
    //Set the clocking to run directly from the external crystal/oscillator
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_INT |
SYSCTL_XTAL_16MHZ);
    //Configure UART to print to terminal
    ConfigUART();
    //initialize I2C module 0
    InitI2C0();
    //Get MPU6050 data
    MPU6050Example();

```

```

    return(0);
}

void Complementary_Filter(float fAccel[], float fGyro[])
{
    _iq16 ForceMagApprx, PitchAcc, RollAcc, sensitivity, Rat, val1, val2;
    _iq16 Gyro[3], Acc[3];
    _iq16 Pitch = 0;
    _iq16 Roll = 0;

    Rat = _IQ16(RATIO);
    val1 = _IQ16(0.98);
    val2 = _IQ16(0.02);
    Gyro[0] = _IQ16(fGyro[0]);
    Gyro[1] = _IQ16(fGyro[1]);
    Gyro[2] = _IQ16(fGyro[2]);
    Acc[0] = _IQ16(fAccel[0]);
    Acc[1] = _IQ16(fAccel[1]);
    Acc[2] = _IQ16(fAccel[2]);
    sensitivity = _IQ16(GYROSCOPE_SENSITIVITY);

    Pitch += _IQ16mpy(_IQ16div(Gyro[0],sensitivity), _IQ16(SAMPLE_RATE));
    Roll -= _IQ16mpy(_IQ16div(Gyro[1],sensitivity), _IQ16(SAMPLE_RATE));
    ForceMagApprx = _IQabs(Acc[0]) + _IQabs(Acc[1]) + _IQabs(Acc[2]);

    if(ForceMagApprx > 1411510 && ForceMagApprx < 4705028)
    {
        PitchAcc = _IQ16mpy(_IQ16atan2(Acc[1],Acc[2]), Rat);
        Pitch = _IQ16mpy(Pitch,val1) + _IQ16mpy(PitchAcc,val2);
        RollAcc = _IQ16mpy(_IQ16atan2(Acc[0],Acc[2]), Rat);
        Roll = _IQ16mpy(Roll,val1) + _IQ16mpy(RollAcc,val2);
        UARTprintf("Pitch   : %d   \t | Roll    : %d   \t \n", (int)Pitch,
(int)Roll);
    }
}

static void
MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
//callback when transactions are completed
{
    // 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 I2CIntHandler(void)
// I2C module interrupt handler
{
    I2CMIntHandler(&g_sI2CMInst);
}

```

```

void MPU6050Example(void)
{
    float fAccel[3], fGyro[3];
    uint_fast16_t fAccelRaw[3], fGyroRaw[3]; //raw values
    tMPU6050 sMPU6050;
    float xa = 0, ya = 0, za = 0;
    float xg = 0, yg = 0, zg = 0;
    uint_fast16_t xaR = 0, yaR = 0, zaR = 0; //raw values
    uint_fast16_t xgR = 0, ygR = 0, zgR = 0;

    //Initialized the MPU6050. This code assumes that the I2C master instance
    //has already been initialized.

    g_bMPU6050Done = false;
    // Initialize the MPU6050. This code assumes that the I2C master instance
    // has already been initialized.
    g_bMPU6050Done = false;
    MPU6050Init(&sMPU6050, &g_sI2CInst, 0x68, MPU6050Callback, &sMPU6050);
    while (!g_bMPU6050Done)
    {
        // Configure the MPU6050 for +/- 4 g accelerometer range.
        g_bMPU6050Done = false;
        MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_ACCEL_CONFIG,
~MPU6050_ACCEL_CONFIG_AFS_SEL_M,
                                MPU6050_ACCEL_CONFIG_AFS_SEL_4G, MPU6050Callback,
&sMPU6050);
        while (!g_bMPU6050Done)
        {
            g_bMPU6050Done = false;
            MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0b00000010 &
MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &sMPU6050);
            while (!g_bMPU6050Done)
            {
                g_bMPU6050Done = false;
                MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00,
MPU6050Callback, &sMPU6050);
                while (!g_bMPU6050Done)
                {
                    // Loop forever reading data from the MPU6050.
                    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(&MPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
    MPU6050DataGyroGetFloat(&MPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);
    MPU6050DataAccelGetRaw(&MPU6050, &fAccelRaw[0], &fAccelRaw[1],
    &fAccelRaw[2]);
    MPU6050DataGyroGetRaw(&MPU6050, &fGyroRaw[0], &fGyroRaw[1], &fGyroRaw[2]);

    //Do something with the new accelerometer and gyroscope readings.

    xg = fGyro[0]*10000;
    yg = fGyro[1]*10000;
    zg = fGyro[2]*10000;
    xa = (atan2(fAccel[0], sqrt(fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
    ya = (atan2(fAccel[1], sqrt(fAccel[0] * fAccel[0] + fAccel[2] *
fAccel[2]))*180.0)/3.14;
    za = (atan2(fAccel[2], sqrt(fAccel[1] * fAccel[1] + fAccel[2] *
fAccel[2]))*180.0)/3.14;

    //Raw values
    xaR = fAccelRaw[0];
    yaR = fAccelRaw[1];
    zaR = fAccelRaw[2];
    xgR = fGyroRaw[0];
    ygR = fGyroRaw[1];
    zgR = fGyroRaw[2];

    Complementary_Filter(fAccel, fGyro); //values go through filter

    UARTprintf("Filtered values: \n");
    UARTprintf("ACC.  Xa: %d      \t | ACC.  Ya: %d      \t | ACC.  Za: %d \n",
(int)xa, (int)ya, (int)za);
    UARTprintf("GYRO. Xg: %d      \t | GYRO. Yg: %d      \t | GYRO. Zg: %d \n",
(int)xg, (int)yg, (int)zg);

    UARTprintf("RAW: \n");
    UARTprintf("ACC.  XaR: %d      \t | ACC.  YaR: %d      \t | ACC.  ZaR: %d \n",
(uint_fast16_t)xaR, (uint_fast16_t)yaR, (uint_fast16_t)zaR);
    UARTprintf("GYRO. XgR: %d      \t | GYRO. YgR: %d      \t | GYRO. ZgR: %d \n",
(uint_fast16_t)xgR, (uint_fast16_t)ygR, (uint_fast16_t)zgR);
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*1000 );
}
}

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_FIFCTL) = 80008000;

// Initialize the I2C master driver.
I2CInit(&g_sI2CInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet());
}

void ConfigUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIOA peripherals(the UART
pins are on GPIO Port A)

    //Configure pins for the receiver and transmitter
    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);
}

```