TITLE:

Midterm by Ron Joshua Recrio

GOAL:

- Understand I2C protocol
- Interact with the MPU-6050
- Filter the values using IQMath

DELIVERABLES:

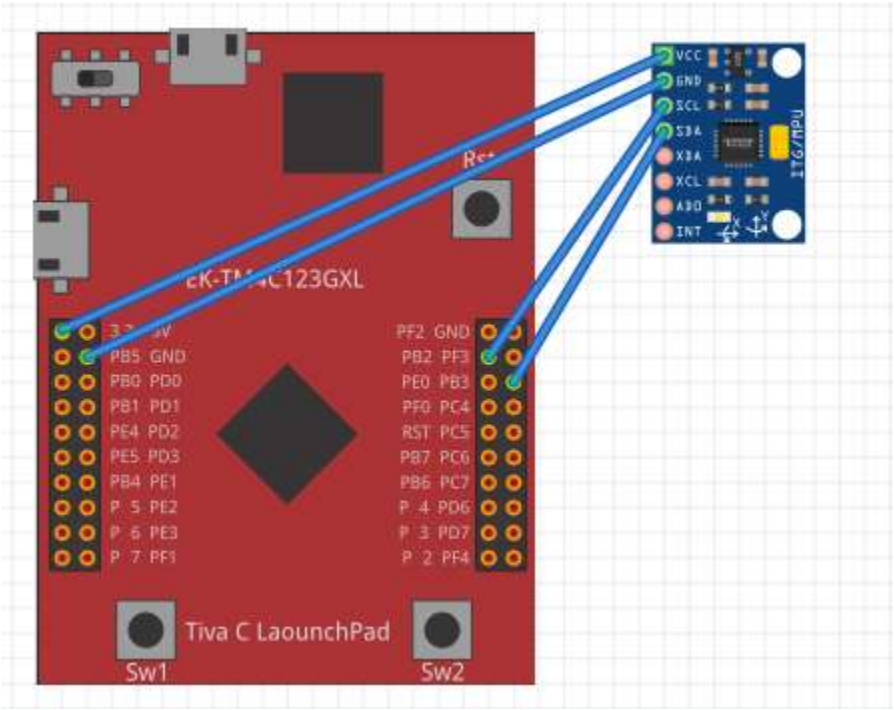
The intended project deliverables were to use the MPU-6050 to firstly show the raw values. Then using those values, implement a filter that uses IQMath in order to have faster efficiency in terms of calculation cycles. Using any tool, graph said values. All parts of this project were completed.

Screenshots of the Graphs:

COMPONENTS:

There were only two components: the Tiva-C, and the MPU-6050. The MPU-6050 is an Inertial Measurement Unit (IMU) that has six degrees of freedom which are up/down, back/forth, right/left, pitch, roll, and yaw. The limitation of this device is that it can only be accurate up to ±16g for it's accelerometer and 2000°/s for the gyroscope. The interface that it uses is I2C which uses Serial Data (SDA) and Serial Clock (SCL) to communicate between a master (Tiva-C) and a slave (MPU-6050). The registers used in the component were 67-72 for the gyroscope's values, and 59-65 for the accelerometer's values. A lot of other registers were used such as 107 for device resetting and other configuration registers, but the sensorlib handles those for us. The I2C is initialized to create a link between the Tiva-C and the IMU, and the UART is configured to send the data to the computer.

SCHEMATICS:



IMPLEMENTATION:

The first thing implemented were the libraries and configuring the search paths for them. The most important ones are driverlib, sensorlib, and IQMath libraries. Going through main we have 3 subroutines: ConfigureUart(), I2C0_Init(), and MPU6050().

Configuring UART is straightforward. It configures two pins to be dedicated UART communication pins and also configures the UART settings using uartstdio funtions.

I2C0_Init() is also straightforward. The subroutine enables I2C0 and configures PB2 as SCL, and PB3 as SDA.

MPU6050() is the main part of the program. This contains setting up the MPU6050 as well as the continous loop to gather data and output the values using UART. The first step is to call on another subroutine MPU6050Init() which initializes the MPU6050 in order for it to properly communicate with the TivaC. Once the initialization is complete, the next step is to configure the settings of the MPU6050. Using MPU6050ReadModifyWrite(), it is possible to rewrite the configuration of the IMU. After modifying the MPU6050, the next part is the infinite loop to gather data. In order to gather data, the MPU6050DataRead() function is used. Once sent, it is possible to use MPU6050DataAccelGetFloat() and MPU6050DataGyroGetFloat() to obtain the raw values. After obtaining the raw values, UARTprintf() is used to output the data to terminal using serial communication. The continued version of this code also uses the Complementary Filter function. This function strictly uses IQMath types and functions to filter the raw values, and provide the pitch and the roll. These values are also outputted to terminal using UARTprintf(). This process loops indefinitely.

CODE:

Task 1-2:

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

// Given, but not used
#define ACCEL_SLAVE_ADDR 0x1D
#define XOUT8 0x06
#define YOUT8 0x07
#define ZOUT8 0x08

volatile bool g_bMPU6050Done; // used for checking if MPU is idle
tI2CMInstance sI2CInst;       // I2C Master instance

// Given
void ConfigureUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    // enable PORTA as peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);    // enable UART0
    GPIOPinConfigure(GPIO_PA0_U0RX);                // Configure PA0 as RX for UART0
    GPIOPinConfigure(GPIO_PA1_U0TX);                // Configure PA1 as TX for UART0
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);  // Set Pin type to
UART
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);   // Set Clock Source as
internal oscillator
    UARTStdioConfig(0, 115200, 16000000);           // Configure UART to send at
115200
}

// Given
void I2C0_Init ()
//Configure/initialize the I2C0
{
    SysCtlPeripheralEnable (SYSCTL_PERIPH_I2C0);    //enables I2C0
```

```
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);   //enable PORTB as peripheral
    GPIOPinTypeI2C (GPIO_PORTB_BASE, GPIO_PIN_3);   //set I2C PB3 as SDA
    GPIOPinConfigure (GPIO_PB3_I2C0SDA);

    GPIOPinTypeI2CSCL (GPIO_PORTB_BASE, GPIO_PIN_2);    //set I2C PB2 as SCLK
    GPIOPinConfigure (GPIO_PB2_I2C0SCL);

    I2CMasterInitExpClk (I2C0_BASE, SysCtlClockGet(), true);   //Set the clock of the
I2C to ensure proper connection

    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
    I2CMInit(&sI2CInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet()); //
Initialize I2C Master
}

//Given, but not used
void I2C0_Write (uint8_t addr, uint8_t N, ...)
//Writes data from master to slave
//Takes the address of the device, the number of arguments, and a variable amount of
register addresses to write to
{
    I2CMasterSlaveAddrSet (I2C0_BASE, addr, false); //Find the device based on the
address given
    while (I2CMasterBusy (I2C0_BASE));

    va_list vargs;  //variable list to hold the register addresses passed

    va_start (vargs, N);    //initialize the variable list with the number of
arguments

    I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));   //put the first argument
in the list in to the I2C bus
    while (I2CMasterBusy (I2C0_BASE));
    if (N == 1) //if only 1 argument is passed, send that register command then stop
    {
        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        while (I2CMasterBusy (I2C0_BASE));
        va_end (vargs);
    }
    else
    //if more than 1, loop through all the commands until they are all sent
    {
        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
        while (I2CMasterBusy (I2C0_BASE));
        uint8_t i;
        for (i = 1; i < N - 1; i++)
        {
            I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));   //send the next
register address to the bus
            while (I2CMasterBusy (I2C0_BASE));

            I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);   //burst
send, keeps receiving until the stop signal is received
            while (I2CMasterBusy (I2C0_BASE));
        }
```

```
        I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));   //puts the last
argument on the SDA bus
        while (I2CMasterBusy (I2C0_BASE));

        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH); //send the
finish signal to stop transmission
        while (I2CMasterBusy (I2C0_BASE));

        va_end (vargs);
    }

}

//Given, but not used
uint32_t I2C0_Read (uint8_t addr, uint8_t reg)
//Read data from slave to master
//Takes in the address of the device and the register to read from
{
    I2CMasterSlaveAddrSet (I2C0_BASE, addr, false); //find the device based on the
address given
    while (I2CMasterBusy (I2C0_BASE));

    I2CMasterDataPut (I2C0_BASE, reg);  //send the register to be read on to the I2C
bus
    while (I2CMasterBusy (I2C0_BASE));

    I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);   //send the send
signal to send the register value
    while (I2CMasterBusy (I2C0_BASE));

    I2CMasterSlaveAddrSet (I2C0_BASE, addr, true);  //set the master to read from the
device
    while (I2CMasterBusy (I2C0_BASE));

    I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);    //send the
receive signal to the device
    while (I2CMasterBusy (I2C0_BASE));

    return I2CMasterDataGet (I2C0_BASE);    //return the data read from the bus
}

// Interrupt handler for I2C
void I2CIntHandler(void)
{
    I2CMIntHandler(&sI2CInst);
}

// Callback function used for checking the success status of the MPU6050
void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    if (ui8Status != I2CM_STATUS_SUCCESS)
    {
        // used for debugging
    }
```

```
    g_bMPU6050Done = true; // set done to true after a successful MPU_6050 command
}

// Given, but modified
void MPU6050 (void)
{
    float fAccel[3], fGyro[3];  // variables used to hold the current values of the
MPU
    tMPU6050 sMPU6050;          // initialize an MPU6050

    g_bMPU6050Done = false;     // always set to false before an MPU6050 command to
check if it changes after.
    MPU6050Init(&sMPU6050, &sI2CInst, 0x68, MPU6050Callback, &sMPU6050); //
initialize MPU6050
    while(!g_bMPU6050Done)      // always wait for the MPU6050 to be done after each
command.
    {
    }
    g_bMPU6050Done = false;
    // Change the accelerometer config as +/- 4G
    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;

    // Reset the device after changing the configuration
    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 gathering values and sending them out
    while(1)
    {
        int i;
        g_bMPU6050Done = false;
        MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050); // send a read
command to the MPU6050
        while(!g_bMPU6050Done)
        {
        }

        // Get both Accelerometer and Gyroscope values as floats.
        MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
        MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);
```

```
        // Since UART cannot send float values, I multiply them by 100 and pass them
as integers.
        for (i = 0; i < 3; i++)
        {
           fAccel[i] *= 100;
           fGyro[i] *= 100;
        }

        UARTprintf("aX: %d aY: %d aZ: %d \n", (int)fAccel[0], (int)fAccel[1],
(int)fAccel[2]);
        UARTprintf("gX: %d gY: %d gZ: %d \n\n", (int)fGyro[0], (int)fGyro[1],
(int)fGyro[2]);

        // Delay of ~1 second.
        SysCtlDelay(SysCtlClockGet()/(3*1000)*1000);
    }
}

void main (void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
// set the clock

    ConfigureUART();    // configure the UART of Tiva C
    I2C0_Init();        // initialize the I2C0 of Tiva C
    MPU6050();          // MPU6050 main function to set up and gather values.

    // should never be reached
    while (1)
    {};
}
```

Task 3-4:

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/hibernate.h"
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
```

```
#include "uartstdio.h"

// new included libraries for IQMath
#include "IQmath/IQmathLib.h"
#include <math.h>

// Given, but not used
#define ACCEL_SLAVE_ADDR 0x1D
#define XOUT8 0x06
#define YOUT8 0x07
#define ZOUT8 0x08

// Given
#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536

// Given
#define M_PI 3.14159265359

// Given
#define dt 0.01

volatile bool g_bMPU6050Done; // used for checking if MPU is idle
tI2CMInstance sI2CInst;       // I2C Master Instance

// Modified Complementary Filter that uses IQMath for efficiency
void ComplementaryFilter(_iq16 accData[3], _iq16 gyrData[3], _iq16 *pitch, _iq16
*roll)
{
    _iq16 pitchAcc, rollAcc; //pitch and roll accumulated

    // calculate pitch and roll changes
    *pitch += _IQ16mpy(_IQ16div(gyrData[0], GYROSCOPE_SENSITIVITY) , dt);
    *roll -= _IQ16mpy(_IQ16div(gyrData[1], GYROSCOPE_SENSITIVITY), dt);

    // Calculate forces on the IMU to detect movement
    int forceMagnitudeApprox = _IQ16abs(accData[0]) + _IQ16abs(accData[1]) +
_IQ16abs(accData[2]);
    UARTprintf("Force: %d\n" , forceMagnitudeApprox); // print Force to check for
movement
    // if movement is detected filter the values
    if (forceMagnitudeApprox > 1390000 && forceMagnitudeApprox < 1390000 * 4)
    {
        pitchAcc = _IQ16div(_IQ16mpy(_IQ16atan2(accData[1], accData[2]) , 180),
M_PI);
        *pitch = *pitch * 0.98 + pitchAcc * 0.02; // filtering formula

        rollAcc = _IQ16div(_IQ16mpy(_IQatan2(accData[0], accData[2]), 180), M_PI);
        *roll = *roll * 0.98 + rollAcc * 0.02;    // filtering formula

        // filtering for the yaw could have also been implemented similarly.
    }
}

// Given
```

```
void ConfigureUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    // enable PORTA as peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);    // enable UART0
    GPIOPinConfigure(GPIO_PA0_U0RX);                // Configure PA0 as RX for UART0
    GPIOPinConfigure(GPIO_PA1_U0TX);                // Configure PA1 as TX for UART0
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);  // Set Pin type to
UART
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);   // Set Clock Source as
internal oscillator
    UARTStdioConfig(0, 115200, 16000000);           // Configure UART to send at
115200
}



// Given
void I2C0_Init ()
//Configure/initialize the I2C0
{
    SysCtlPeripheralEnable (SYSCTL_PERIPH_I2C0);    //enables I2C0
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);   //enable PORTB as peripheral
    GPIOPinTypeI2C (GPIO_PORTB_BASE, GPIO_PIN_3);   //set I2C PB3 as SDA
    GPIOPinConfigure (GPIO_PB3_I2C0SDA);

    GPIOPinTypeI2CSCL (GPIO_PORTB_BASE, GPIO_PIN_2);    //set I2C PB2 as SCLK
    GPIOPinConfigure (GPIO_PB2_I2C0SCL);

    I2CMasterInitExpClk (I2C0_BASE, SysCtlClockGet(), true);   //Set the clock of the
I2C to ensure proper connection

    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
    I2CMInit(&sI2CInst, I2C0_BASE, INT_I2C0, 0xff, 0xff, SysCtlClockGet()); //
Initialize I2C Master
}

//Given, but not used
void I2C0_Write (uint8_t addr, uint8_t N, ...)
//Writes data from master to slave
//Takes the address of the device, the number of arguments, and a variable amount of
register addresses to write to
{
    I2CMasterSlaveAddrSet (I2C0_BASE, addr, false); //Find the device based on the
address given
    while (I2CMasterBusy (I2C0_BASE));

    va_list vargs;  //variable list to hold the register addresses passed

    va_start (vargs, N);    //initialize the variable list with the number of
arguments

    I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));   //put the first argument
in the list in to the I2C bus
    while (I2CMasterBusy (I2C0_BASE));
    if (N == 1) //if only 1 argument is passed, send that register command then stop
```

```
    {
        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        while (I2CMasterBusy (I2C0_BASE));
        va_end (vargs);
    }
    else
    //if more than 1, loop through all the commands until they are all sent
    {
        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
        while (I2CMasterBusy (I2C0_BASE));
        uint8_t i;
        for (i = 1; i < N - 1; i++)
        {
            I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));   //send the next
register address to the bus
            while (I2CMasterBusy (I2C0_BASE));

            I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);   //burst
send, keeps receiving until the stop signal is received
            while (I2CMasterBusy (I2C0_BASE));
        }

        I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));   //puts the last
argument on the SDA bus
        while (I2CMasterBusy (I2C0_BASE));

        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH); //send the
finish signal to stop transmission
        while (I2CMasterBusy (I2C0_BASE));

        va_end (vargs);
    }

}

//Given, but not used
uint32_t I2C0_Read (uint8_t addr, uint8_t reg)
//Read data from slave to master
//Takes in the address of the device and the register to read from
{
    I2CMasterSlaveAddrSet (I2C0_BASE, addr, false); //find the device based on the
address given
    while (I2CMasterBusy (I2C0_BASE));

    I2CMasterDataPut (I2C0_BASE, reg);  //send the register to be read on to the I2C
bus
    while (I2CMasterBusy (I2C0_BASE));

    I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);   //send the send
signal to send the register value
    while (I2CMasterBusy (I2C0_BASE));

    I2CMasterSlaveAddrSet (I2C0_BASE, addr, true);  //set the master to read from the
device
    while (I2CMasterBusy (I2C0_BASE));
```

```
    I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);    //send the
receive signal to the device
    while (I2CMasterBusy (I2C0_BASE));

    return I2CMasterDataGet (I2C0_BASE);    //return the data read from the bus
}

// Interrupt handler for I2C
void I2CIntHandler(void)
{
    I2CMIntHandler(&sI2CInst);
}

// Callback function used for checking the success status of the MPU6050
void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    if (ui8Status != I2CM_STATUS_SUCCESS)
    {
        // used for debugging
    }
    g_bMPU6050Done = true; // set done to true after a successful MPU_6050 command
}

// Given, but modified
void MPU6050 (void)
{
    float fAccel[3], fGyro[3];  // variables used to hold the current values of the
MPU
    _iq16 cFAccel[3], cFGyro[3], pitch, roll; // new variables used for the filter
    tMPU6050 sMPU6050;          // initialize an MPU6050

    g_bMPU6050Done = false;     // always set to false before an MPU6050 command to
check if it changes after.
    MPU6050Init(&sMPU6050, &sI2CInst, 0x68, MPU6050Callback, &sMPU6050); //
initialize MPU6050
    while(!g_bMPU6050Done)      // always wait for the MPU6050 to be done after each
command.
    {
    }
    g_bMPU6050Done = false;
    // Change the accelerometer config as +/- 4G
    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;

    // Reset the device after changing the configuration
    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 gathering values and sending them out
    while(1)
    {
        int i;
        g_bMPU6050Done = false;
        MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050); // send a read
command to the MPU6050
        while(!g_bMPU6050Done)
        {
        }

        // Get both Accelerometer and Gyroscope values as floats.
        MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
        MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);

        // Convert raw values to IQ16 and also for UART
        for (i = 0; i < 3; i++)
        {

           cFAccel[i] = _IQ16(fAccel[i]);
           cFGyro[i] = _IQ16(fGyro[i]);
           fAccel[i] *= 100;
           fGyro[i] *= 100;
        }

        // Print raw values
        UARTprintf("aX: %d aY: %d aZ: %d \n", (int)fAccel[0], (int)fAccel[1],
(int)fAccel[2]);
        UARTprintf("gX: %d gY: %d gZ: %d \n", (int)fGyro[0], (int)fGyro[1],
(int)fGyro[2]);

        // Filter raw values
        ComplementaryFilter(cFAccel, cFGyro, &pitch, &roll);

        // Print filtered values
        UARTprintf("Pitch: %d Roll: %d \n\n", pitch, roll);

        // Delay of ~1 second.
        SysCtlDelay(SysCtlClockGet()/(3*1000)*1000);


    }
}

void main (void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
//set the clock
```

```
    ConfigureUART ();   // configure the UART of Tiva C
    I2C0_Init ();       // initialize the I2C0 of Tiva C
    MPU6050 ();         // MPU6050 main function to set up and gather values.

    // should never be reached
    while (1)
    {};
}
```