Ameera Essaqi
Github root directory: https://github.com/AmeeraE/microcontrollers/tree/master/TIVAC

CPE 403 ADV EMB SYS DES F 2019

Date Submitted: **10/30/19**

TITLE: Midterm Project

GOAL: The goal of this project was to interface the MPU6050 using I2C protocol to the TivaC. We then wanted to print the values of the accelerometer and gyro on the serial terminal and then to a graph. After that, we wanted to implement a complementary filter using the IQMath Library and filter out the accelerometer and gyro values. We then wanted to display all values, filtered and unfiltered, on the serial terminal and then on the graph.

DELIVERABLES:

In Task1, we interfaced the MPU6050 IMU using I2C protocol to the TivaC. We then printed all accelerometer and gyro values on the serial terminal.

In Task2, we interfaced the MPU6050 IMU using I2C protocol to the TivaC. We then plotted all the accelerometer and gyro values on the graph using the graphing tool provided in CCS studio.

In Task3, we implemented a complementary filter using the IQMath Library. We used this to filter the raw accelerometer and gyro values received from Task2. We then printed both the raw and filtered accelerometer and gyro values on the serial terminal.

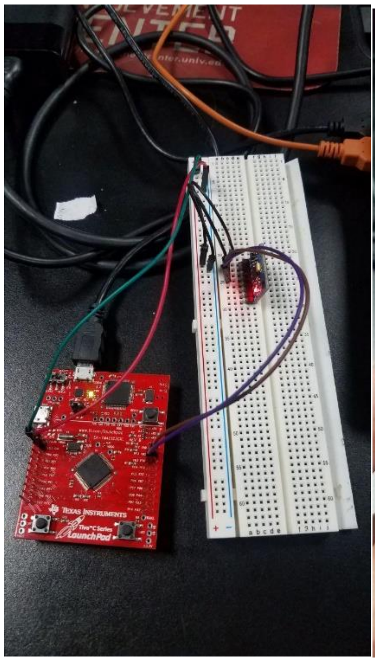
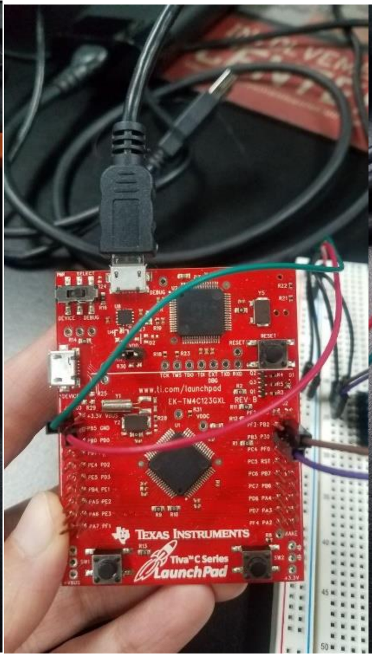
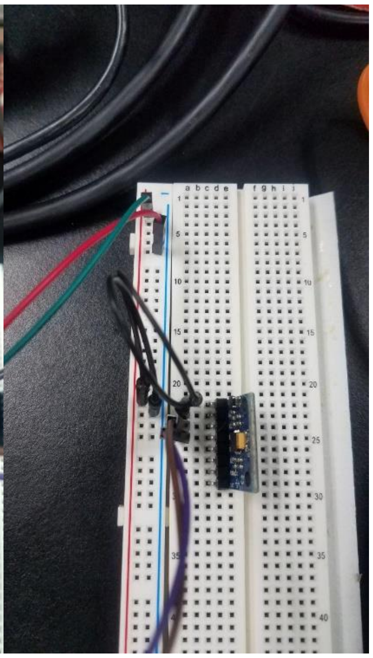
In Task4, we implemented a complementary filter to filter the raw accelerometer and gyro values. We then plotted both raw and filtered values on to the graphing tool provided in CCS studio.

COMPONENTS:

In this lab, we used the TIVA-C Series TM4C123G LaunchPad and the MPU 6050 device. I connected the MPU 6050's VCC to 3.3V on the Tiva-C and the Tiva-C's GND to MPU 6050's GND. I then connected the PB2 on the Tiva-C to SCL on MPU 6050 and PB3 on the Tiva-C to SDA on the MPU6050.

Ameera Essaqi
Github root directory: https://github.com/AmeeraE/microcontrollers/tree/master/TIVAC

SCHEMATICS:

IMPLEMENTATION:

First, I used a beginning code given by TivaWare Sensor Library and added functions as needed.

I used: `void InitUART(void)`

This was needed to print the values onto the terminal which we did in Task 1 and Task 3.

Then I used: void InitI2c0(void) and void I2CMSimpleIntHandler(void)

This is the I2C communication. This is communication between the Tiva C board and the MPU 6050.

I also used the void ComplementaryFilter()

This used the IQMath library and was needed in order to complete Task 3 and Task 4. This function was given in our midterm assignment and was used to filter the values that we later printed in the terminal and displayed on the graph.

CODE:

Task 01 and Task02:

```
Youtube Link(Task01): https://youtu.be/mzp92DPXdmQ
Youtube Link(Task02): https://youtu.be/Y_CqU2-5dqc https://youtu.be/fEs7QvWwYHI
```

`Modified Code:`

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
// Ameera Essaqi CPE403
//Midterm Project 10/30/2019
#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"

void InitUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIO port a

    GPIOPinConfigure(GPIO_PA0_U0RX); //set PA0 as RX pin
    GPIOPinConfigure(GPIO_PA1_U0TX); //set PA1 as TX pin
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); //sets UART pin type

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC); //sets clock source
    UARTStdioConfig(0, 115200, 16000000); //disables uartstdio, sets baud rate to 115200 for terminal,
uses clock
}

//
// 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);

    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

    //clear I2C FIFOs
    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 8000800;

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

// Interrupt for I2CM
void I2CMSimpleIntHandler(void)
{
    I2CMIntHandler(&g_sI2CMSimpleInst);
}

void DelayinMS(int ms) {
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*ms ) ;  // created delay in micro seconds
}


int main(void)
{

    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ); //system clock
    InitI2C0(); //initializes I2C
    InitUART(); //Initializes UART

    //Local Variables
    float fAccel[3], fGyro[3]; //two float array for readings
    float xA, yA, zA;
    float xG, yG, zG;
    float accData[3];   //two float array for temp readings
    float gyrData[3];
    volatile uint32_t xA_graph, yA_graph, zA_graph , xG_graph, yG_graph, zG_graph; //variables for graph
readings, volatile so it continues to update

    tMPU6050 sMPU6050;      // device name
    g_bMPU6050Done = false;

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

    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, // No need to mask
                           MPU6050_ACCEL_CONFIG_AFS_SEL_4G, // Accelerometer full-scale range 4g, was 8g
but changed it back
                           MPU6050Callback,
                           &sMPU6050);

    while (!g_bMPU6050Done);

    //Settings for the Gyroscope
    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);

    //Turns on power for Accelerometer & Gyroscope

    g_bMPU6050Done = false;

    MPU6050ReadModifyWrite(&sMPU6050,
                           MPU6050_O_PWR_MGMT_1, // Power management 1 register
                           0x00, // No need to mask
                           0x00,//0x02 & MPU6050_PWR_MGMT_1_DEVICE_RESET,
                           MPU6050Callback,
                           &sMPU6050);

    while (!g_bMPU6050Done);

    //Turns on power for Accelerometer & Gyroscope
    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050,
                           MPU6050_O_PWR_MGMT_2, // Power management 2 register
                           0x00, // No need to mask
                           0x00,
                           MPU6050Callback,
                           &sMPU6050);

    while (!g_bMPU6050Done);


    // Loop forever reading data from the MPU6050
    while (1)
    {
        //Wait for MPU6050
        g_bMPU6050Done = false;
        MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
        while (!g_bMPU6050Done);

        //Obtain raw values of Accelerometer and Gyroscope
        MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
        MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);
        //Place raw values of Accelerometer and Gyroscope into a register
        //multiplied by 100, helps the values be more readable on display. Value doesn't matter.
        xA = fAccel[0] * 100;
        yA = fAccel[1] * 100;
        zA = fAccel[2] * 100;

        xG = fGyro[0] * 100;
        yG = fGyro[1] * 100;
        zG = fGyro[2] * 100;

        xA_graph = (int)xA;
       yA_graph = (int)yA;
      zA_graph = (int)zA;

      xG_graph = (int)xG;
       yG_graph = (int)yG;
      zG_graph = (int)zG;

        //UART print to terminal
        UARTprintf("Acceleration:\n (X:%d Y:%d Z:%d)\n", (int)xA, (int)yA, (int)zA);
        UARTprintf("Gyroscope: \n (X:%d, Y:%d, Z:%d)\n", (int)xG, (int)yG, (int)zG);
        UARTprintf("\n");

        //System clock delay
        DelayinMS(500);  //delay for 500ms
    }
}
```

Task 03 and Task04:

Youtube Link(Task03): https://youtu.be/6JNAqlQdLGU
 Youtube Link(Task04): https://youtu.be/yei--RhxgUY

Modified Code:

```
// Ameera Essaqi CPE403
//Midterm Project 10/30/2019
#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 "IQmath/IQmathLib.h"

//defined values here
#define ACCELEROMETER_SENSITIVITY 8192.0    //defines accelerometer sensitivtiy
#define GYROSCOPE_SENSITIVITY 65.536    //defines gyro sensitivity
#define M_PI 3.14159265359      //defines pi
#define d_t 0.01 // 10 ms sample rate

_iq16 pitch, roll, pitchAcc; //making this global for the Complementary Filter


void InitUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0); //enable UART module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //enable GPIO port a

    GPIOPinConfigure(GPIO_PA0_U0RX); //set PA0 as RX pin
    GPIOPinConfigure(GPIO_PA1_U0TX); //set PA1 as TX pin
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1); //sets UART pin type

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC); //sets clock source
    UARTStdioConfig(0, 115200, 16000000); //disables uartstdio, sets baud rate to 115200 for terminal,
uses clock
}

//
// 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);

    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

    //clear I2C FIFOs
    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 8000800;

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

// Interrupt for I2CM
void I2CMSimpleIntHandler(void)
{
    I2CMIntHandler(&g_sI2CMSimpleInst);
}

void DelayinMS(int ms) {
    SysCtlDelay( (SysCtlClockGet()/(3*1000))*ms ) ;  // created delay in micro seconds
}

void ComplementaryFilter(float accData[3], float gyrData[3])
{
_iq16 rollAcc, forceMagnitudeApprox;
_iq16 zerotwo=0.02;
_iq16 nineeight=0.98;
_iq16 pi=_IQ16(M_PI);
_iq16 dt=_IQ16(d_t);
_iq16 VAR=_IQ16div(180, pi);
_iq16 temp_accData[3];
_iq16 temp_gyrData[3];

temp_accData[0]= _IQ16(accData[0]);
temp_accData[1]= _IQ16(accData[1]);
temp_accData[2]= _IQ16(accData[2]);

temp_gyrData[0]= _IQ16(gyrData[0]);
temp_gyrData[1]= _IQ16(gyrData[1]);
temp_gyrData[2]= _IQ16(gyrData[2]);

_iq16 g_sen= _IQ16(GYROSCOPE_SENSITIVITY);


// Integrate the gyroscope data -> int(angularSpeed) = angle
// Angle around the X-axis
pitch += _IQ16mpy(_IQ16div(temp_gyrData[0],g_sen), dt);
// Angle around the Y-axis
roll -= _IQ16mpy(_IQ16div(temp_gyrData[1], g_sen),dt);
// Compensate for drift with accelerometer data
// Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192
forceMagnitudeApprox = _IQabs(temp_accData[0]) + _IQabs(temp_accData[1]) + _IQabs(temp_accData[2]);
if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)
{
// Turning around the X axis results in a vector on the Y-axis
pitchAcc = _IQ16mpy(_IQ16atan2(temp_accData[1],temp_accData[2]), VAR);
pitch =_IQ16mpy(pitch, nineeight) + _IQ16mpy(pitchAcc, zerotwo);
// Turning around the Y axis results in a vector on the X-axis
rollAcc = _IQ16mpy(_IQ16atan2(temp_accData[0],temp_accData[2]), VAR);
roll = _IQ16mpy(roll, nineeight) + _IQ16mpy(rollAcc, zerotwo);
}
}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
int main(void)
{


    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ); //system clock
    InitI2C0(); //initializes I2C
    InitUART(); //Initializes UART

    //Local Variables
    float fAccel[3], fGyro[3]; //two float array for readings
    float xA, yA, zA;
    float xG, yG, zG;
    float accData[3];   //two float array for temp readings
    float gyrData[3];
    volatile uint32_t xA_graph, yA_graph, zA_graph , xG_graph, yG_graph, zG_graph; //variables for graph
readings, volatile so it continues to update

    tMPU6050 sMPU6050;      // device name
    g_bMPU6050Done = false;

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

    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, // No need to mask
                           MPU6050_ACCEL_CONFIG_AFS_SEL_4G, // Accelerometer full-scale range 4g, was 8g
but changed it back
                           MPU6050Callback,
                           &sMPU6050);

    while (!g_bMPU6050Done);

    //Settings for the Gyroscope
    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);

    //Turns on power for Accelerometer & Gyroscope

    g_bMPU6050Done = false;

    MPU6050ReadModifyWrite(&sMPU6050,
                           MPU6050_O_PWR_MGMT_1, // Power management 1 register
                           0x00, // No need to mask
                           0x00,//0x02 & MPU6050_PWR_MGMT_1_DEVICE_RESET,
                           MPU6050Callback,
                           &sMPU6050);

    while (!g_bMPU6050Done);

    //Turns on power for Accelerometer & Gyroscope
    g_bMPU6050Done = false;
    MPU6050ReadModifyWrite(&sMPU6050,
                           MPU6050_O_PWR_MGMT_2, // Power management 2 register
                           0x00, // No need to mask
                           0x00,
                           MPU6050Callback,
                           &sMPU6050);

    while (!g_bMPU6050Done);


    // Loop forever reading data from the MPU6050
    while (1)
    {
        //Wait for MPU6050
        g_bMPU6050Done = false;
        MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
        while (!g_bMPU6050Done);

        //Obtain raw values of Accelerometer and Gyroscope
        MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);
        MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);


        //Place raw values of Accelerometer and Gyroscope into a register
        //multiplied by 100, helps the values be more readable on display. Value doesn't matter.
        xA = fAccel[0] * 100;
        yA = fAccel[1] * 100;
        zA = fAccel[2] * 100;

        xG = fGyro[0] * 100;
        yG = fGyro[1] * 100;
        zG = fGyro[2] * 100;

        xA_graph = (int)xA;
       yA_graph = (int)yA;
      zA_graph = (int)zA;

      xG_graph = (int)xG;
       yG_graph = (int)yG;
      zG_graph = (int)zG;


      //copy values into temp varables for complementary filter

       accData[0]=fAccel[0];
       accData[1]=fAccel[1];
       accData[2]=fAccel[2];
       gyrData[0]=fGyro[0];
       gyrData[1]=fGyro[1];
       gyrData[2]=fGyro[2];


    ComplementaryFilter(accData,gyrData); //call complementary filter here

        //UART print to terminal
        UARTprintf("Roll(filtered):\n (X:%d)\n", (int)roll);
        UARTprintf("Pitch(filtered): \n (X:%d)\n", (int)pitch);

        UARTprintf("Acceleration(unfiltered):\n (X:%d Y:%d Z:%d)\n", (int)xA, (int)yA, (int)zA);
        UARTprintf("Gyroscope(unfiltered): \n (X:%d, Y:%d, Z:%d)\n", (int)xG, (int)yG, (int)zG);
        UARTprintf("\n");

        //System clock delay
        DelayinMS(500);  //delay for 500ms
    }
}
--------------------------------------------------------------------------------
```

<u>Grading scheme:</u> 30% Coding, 30% Documentation, 40% Execution/Video.