**Date Submitted: 11/3**

**Main CODE:**

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

**#include** <stdbool.h>

**#include** <stdint.h>

**#include** <stdlib.h>

**#include** <math.h>

**#include** "inc/tm4c123gh6pm.h"

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_i2c.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/pin\_map.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/uart.h"

**#include** "driverlib/adc.h"

**#include** "driverlib/i2c.h"

**#include** "driverlib/fpu.h"

**#include** "utils/uartstdio.h"

// Constants

**#define** MPU\_ADDR 0x68 // Address of MPU6050

**#define** ACCELEROMETER\_SENSITIVITY 8192.0

**#define** GYROSCOPE\_SENSITIVITY 65.536

**#define** dt 0.01 // 10 ms sample rate!

// Function prototypes

**void** **MPUInit**();

**void** **ftoa**(**float** f,**char** \*buf);

uint32\_t **I2CReceive**(uint32\_t slave\_addr, uint8\_t reg);

**void** **ComplementaryFilter**(**short** accData[3], **short** gyrData[3], **float** \*pitch, **float** \*roll);

// Raw data from the accelerometer and gyroscope

**short** accelData[3];

**short** gyroData[3];

// Floating-point data

**volatile** **float** accel[3];

**volatile** **float** gyro[3];

**float** pitch;

**float** roll;

// Buffer to hold strings of floating-point values

**char** bufferPitch[10];

**char** bufferRoll[10];

**char** bufferAX[10];

**char** bufferAY[10];

**char** bufferAZ[10];

**char** bufferGX[10];

**char** bufferGY[10];

**char** bufferGZ[10];

**int** **main**(**void**)

{

// System clock at 50 MHz

**SysCtlClockSet**(SYSCTL\_SYSDIV\_4|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

// Set up GPIOA for UART

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

// Enable UART0 so that we can configure the clock.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

// Use the internal 16MHz oscillator as the UART clock source.

**UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);

// Select the alternate (UART) function for these pins.

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Initialize the UART for console I/O.

UARTStdioConfig(0, 115200, 16000000);

// Set up I2C

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_I2C0); // Enable I2C hardware

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOB); // Enable Pin hardware

**GPIOPinConfigure**(GPIO\_PB3\_I2C0SDA); // Configure GPIO pin for I2C Data line

**GPIOPinConfigure**(GPIO\_PB2\_I2C0SCL); // Configure GPIO Pin for I2C clock line

**GPIOPinTypeI2C**(GPIO\_PORTB\_BASE, GPIO\_PIN\_2 | GPIO\_PIN\_3); // Set Pin Type

// SDA MUST BE STD

**GPIOPadConfigSet**(GPIO\_PORTB\_BASE, GPIO\_PIN\_2, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD);

// SCL MUST BE OPEN DRAIN

**GPIOPadConfigSet**(GPIO\_PORTB\_BASE, GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_OD);

// The False sets the controller to 100kHz communication

**I2CMasterInitExpClk**(I2C0\_BASE, **SysCtlClockGet**(), false);

//clear I2C FIFOs

HWREG(I2C0\_BASE + I2C\_O\_FIFOCTL) = 80008000;

// Set up FPU

**FPULazyStackingEnable**(); // Enables lazy stacking of floating-point registers

**FPUEnable**(); // Enables the FPU since it's disabled by default

MPUInit();

**while**(1)

{

// X-Axis Accel

accelData[0] = (I2CReceive(MPU\_ADDR, 0x3B)) << 8; // Upper-byte of data

accelData[0] |= I2CReceive(MPU\_ADDR, 0x3C); // Lower-byte of data

accel[0] = accelData[0] \* 0.00059875; // Converts raw to floating-point

ftoa(accel[0], bufferAX); // Converts floating-point into string

// Y-Axis Accel

accelData[1] = (I2CReceive(MPU\_ADDR, 0x3D)) << 8;

accelData[1] |= I2CReceive(MPU\_ADDR, 0x3E);

accel[1] = accelData[1] \* 0.00059875;

ftoa(accel[1], bufferAY);

// Z-Axis Accel

accelData[2] = (I2CReceive(MPU\_ADDR, 0x3F)) << 8;

accelData[2] |= I2CReceive(MPU\_ADDR, 0x40);

accel[2] = accelData[2] \* 0.00059875;

ftoa(accel[2], bufferAZ);

// X-Axis Gyro

gyroData[0] = (I2CReceive(MPU\_ADDR, 0x43)) << 8;

gyroData[0] |= I2CReceive(MPU\_ADDR, 0x44);

gyro[0] = gyroData[0] \* 0.00059875;

ftoa(gyro[0], bufferGX);

// Y-Axis Gyro

gyroData[1] = (I2CReceive(MPU\_ADDR, 0x45)) << 8;

gyroData[1] |= I2CReceive(MPU\_ADDR, 0x46);

gyro[1] = gyroData[1] \* 0.00059875;

ftoa(gyro[1], bufferGY);

// Z-Axis Gyro

gyroData[2] = (I2CReceive(MPU\_ADDR, 0x47)) << 8;

gyroData[2] |= I2CReceive(MPU\_ADDR, 0x48);

gyro[2] = gyroData[2] \* 0.00059875;

ftoa(gyro[2], bufferGZ);

// Input the raw data into the complementary filter

ComplementaryFilter(accelData, gyroData, &pitch, &roll);

ftoa(pitch, bufferPitch);

ftoa(roll, bufferRoll);

// Print values to the terminal

UARTprintf("\033[2J");

UARTprintf("Pitch: %s\n", bufferPitch);

UARTprintf("Roll: %s\n", bufferRoll);

UARTprintf("Accelerometer:\nX = %s\nY = %s\nZ = %s\n", bufferAX, bufferAY, bufferAZ);

UARTprintf("Gyroscope:\nX = %s\nY = %s\nZ = %s\n\n", bufferGX, bufferGY, bufferGZ);

**SysCtlDelay**(500000); // Delay to prevent the terminal from being flooded

}

**return** 0;

}

// Function to initiate the MPU6050

**void** **MPUInit**()

{

**I2CMasterSlaveAddrSet**(I2C0\_BASE, MPU\_ADDR, false); // False means transmit

**I2CMasterDataPut**(I2C0\_BASE, 0x6B); // PWR\_MGMT\_1 reg

**I2CMasterControl**(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START); // Start condition

**while**(**I2CMasterBusy**(I2C0\_BASE));

**I2CMasterDataPut**(I2C0\_BASE, 0x00); // Wake up the MPU

**I2CMasterControl**(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_FINISH);

**while**(**I2CMasterBusy**(I2C0\_BASE));

}

// I2C function to receive data from the MPU6050

uint32\_t **I2CReceive**(uint32\_t slave\_addr, uint8\_t reg)

{

//specify that we are writing (a register address) to the

//slave device

**I2CMasterSlaveAddrSet**(I2C0\_BASE, slave\_addr, false);

//specify register to be read

**I2CMasterDataPut**(I2C0\_BASE, reg);

//send control byte and register address byte to slave device

**I2CMasterControl**(I2C0\_BASE, I2C\_MASTER\_CMD\_BURST\_SEND\_START);

//wait for MCU to finish transaction

**while**(**I2CMasterBusy**(I2C0\_BASE));

//specify that we are going to read from slave device

**I2CMasterSlaveAddrSet**(I2C0\_BASE, slave\_addr, true);

//send control byte and read from the register we

//specified

**I2CMasterControl**(I2C0\_BASE, I2C\_MASTER\_CMD\_SINGLE\_RECEIVE);

//wait for MCU to finish transaction

**while**(**I2CMasterBusy**(I2C0\_BASE));

//return data pulled from the specified register

**return** **I2CMasterDataGet**(I2C0\_BASE);

}

// Function to convert floating-point to string. This allows the floating-point

// to be printed in the terminal using UARTprintf (by default, does not support

// floating-point).

**void** **ftoa**(**float** f,**char** \*buf)

{

**int** pos=0,ix,dp,num;

**if** (f<0)

{

buf[pos++]='-';

f = -f;

}

dp=0;

**while** (f>=10.0)

{

f=f/10.0;

dp++;

}

**for** (ix=1;ix<8;ix++)

{

num = (**int**)f;

f=f-num;

**if** (num>9)

buf[pos++]='#';

**else**

buf[pos++]='0'+num;

**if** (dp==0) buf[pos++]='.';

f=f\*10.0;

dp--;

}

}

// Complementary filter function to obtain pitch and roll.

**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;

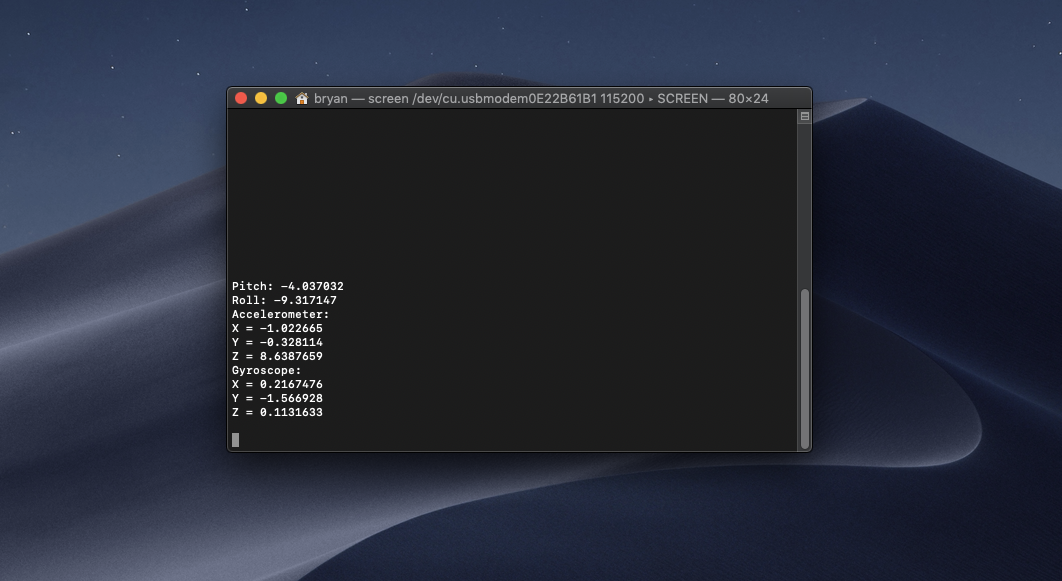
\*roll = \*roll \* 0.98 + rollAcc \* 0.02;

}

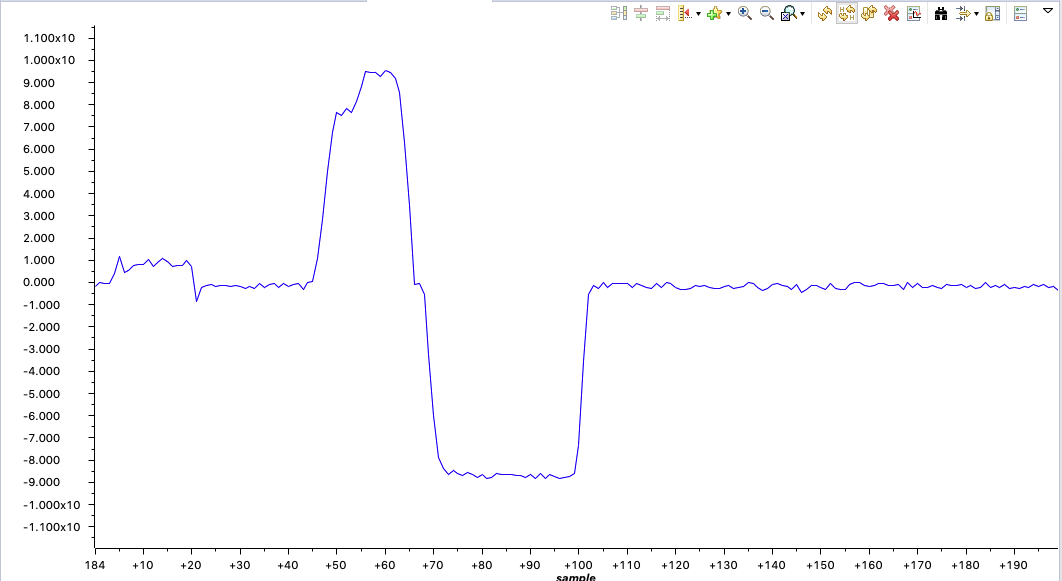
}

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

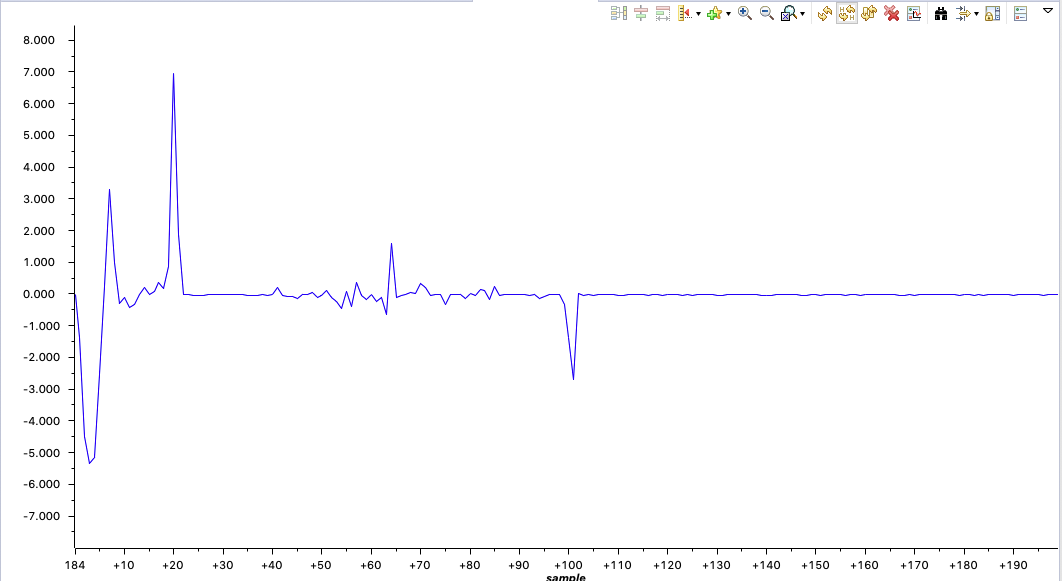
**Task Terminal & Graph:**

****

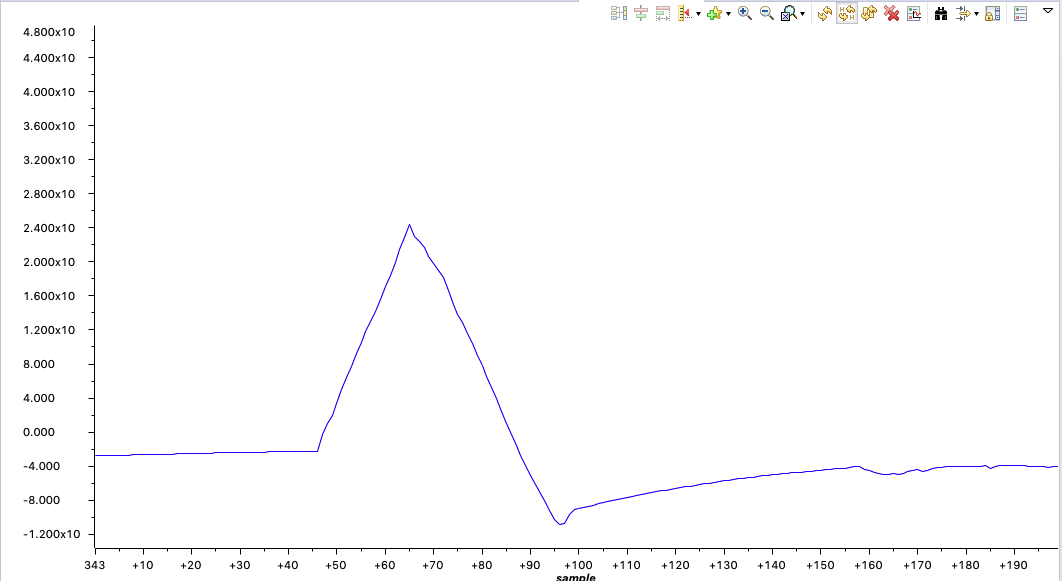
**Terminal**

****

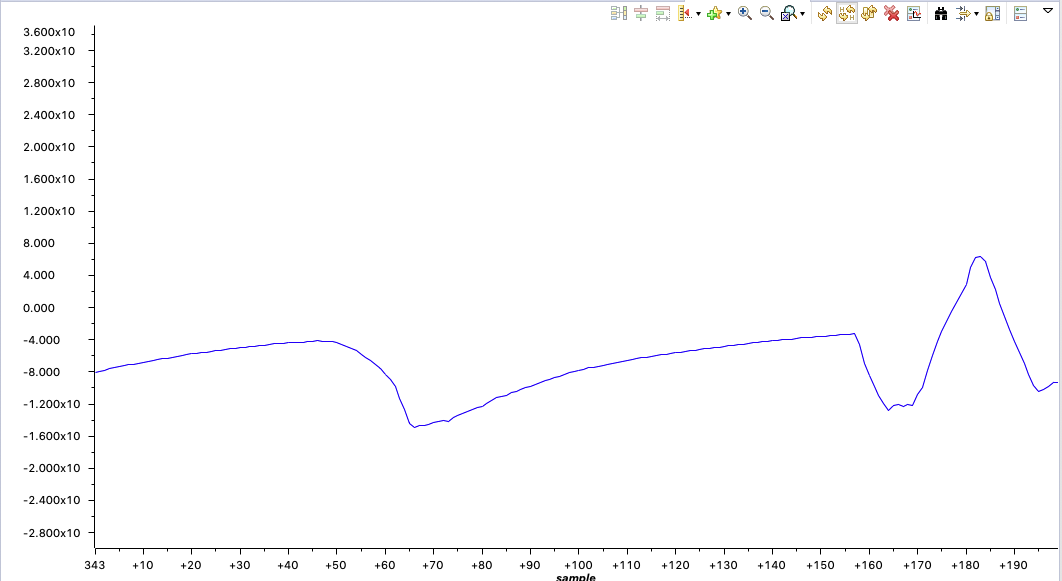
**Example graph of accelerometer on the x-axis.**

****

**Graph of gyroscope on the x-axis.**

****

**Graph of the pitch**

****

**Graph of the roll.**

**My IQMath was not working correctly on the TIVA C.**