CPE 403 ADV EMB SYS DES F 2019

TITLE: Midterm I (I2C)

GOAL: Interface the MPU6050 IMU module with the Tiva C TM4C123GH6PM microcontroller using I2C given multiple tasks.

* Task 1: Interface the given MPU6050 IMU using I2C protocol to TivaC. Print all accelerometer and gyro values on to the serial terminal
* Task 2: Interface the given MPU6050 IMU using I2C protocol to TivaC. Plot all accelerometer and gyro values on to a graph.
* Task 3: Implement a complementary filter to filter the raw accelerometer and gyro values. Print all raw and filtered accelerometer and gyro values on to the serial terminal. Implement the filter using IQMath Library.
* Task 4: Implement a complementary filter to filter the raw accelerometer and gyro values. Plot all raw and filtered accelerometer and gyro values on to a graph.

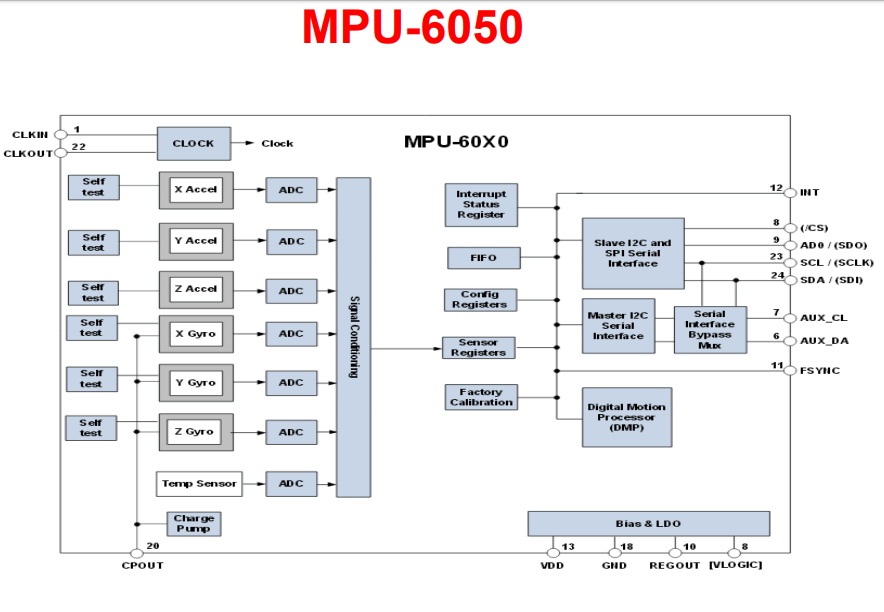
DELIVERABLES: The tasks shown above are the deliverables and they are completed.

COMPONENTS: Tiva C TM4C123GH6PM microcontroller, MPU6050 IMU, jumper wires

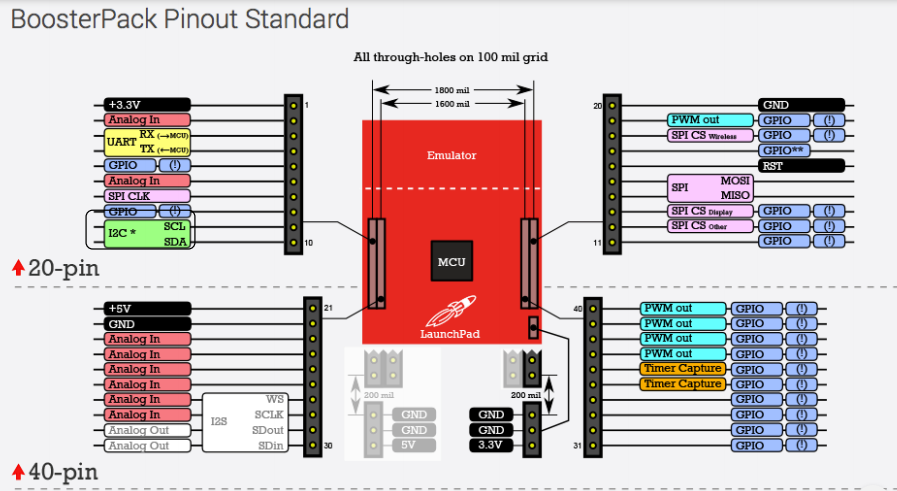
The main components used in this project were the TivaC microcontroller and the MPU6050 module. The i2c interface was used to communicate between the two devices. To initialize communication between the two devices, the I2C module has to be initialized, and I did so by implementing a function that enables the i2c peripheral and configured pins for Rx and Tx. Other modules that were initialized were the UART module (for displaying values) and the MPU6050 module given by the function MPU6050Example(). The next step would be to configure the MPU6050 module. By default that gyroscope is set to +/- 2g and the accelerometer is set to +/-250deg/s. In order to configure the MPU, registers ACCEL\_CONFIG and GYRO\_CONFIG are used and they can be configured up to +/-16g for the sensitivity of the accelerometer or up to +/-2000deg/s for the gyroscope. The PWR\_MGMT\_1 register is also configured in order to configure the power mode of the MPU.

SCHEMATICS:

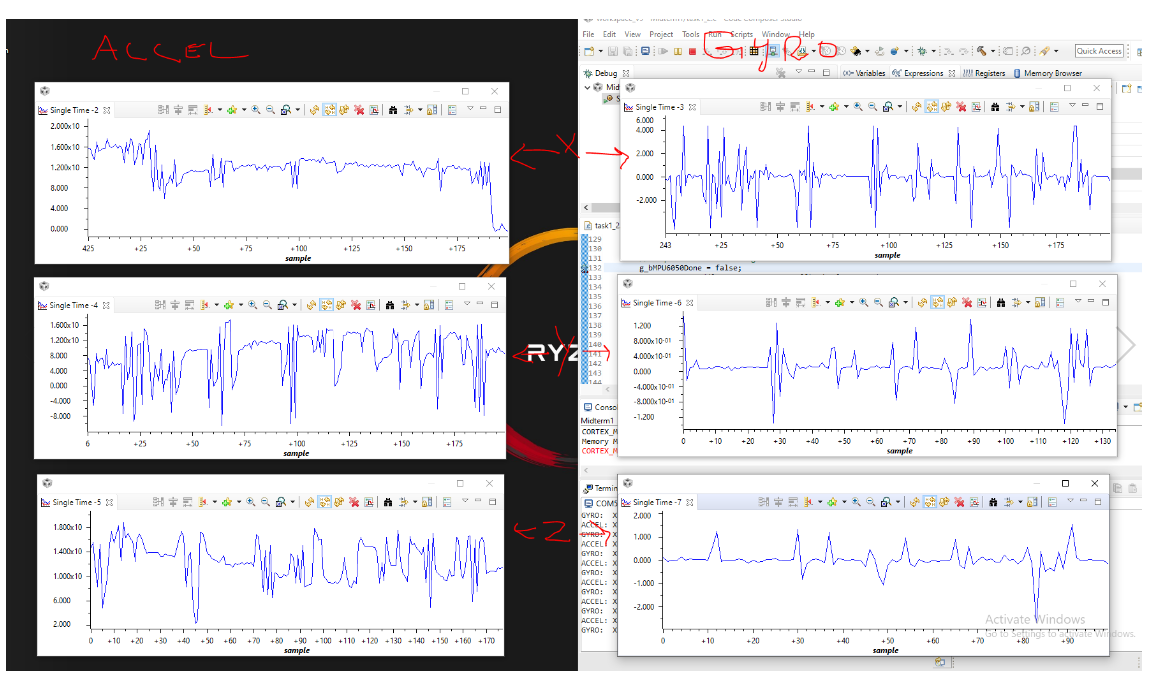
MPU6050

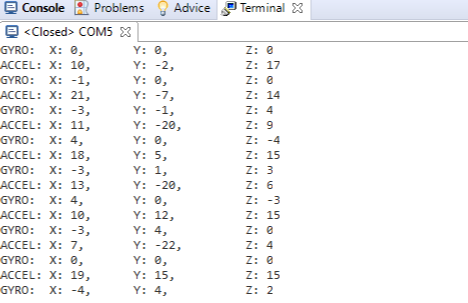


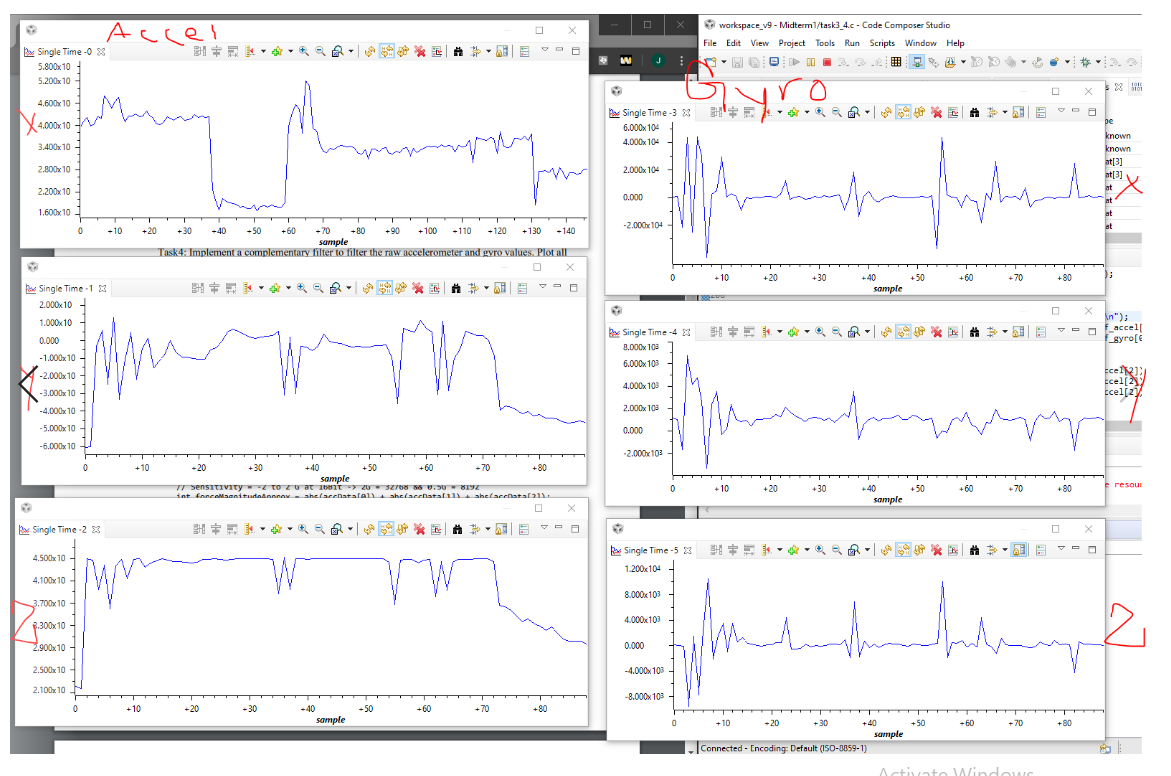
TIVA C TM4C123GH6PM (pin outs)

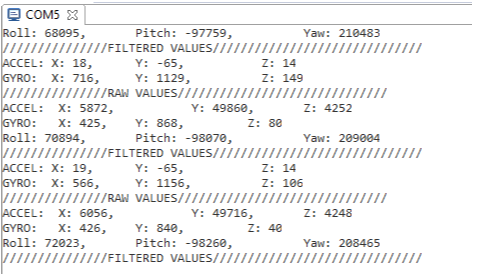


Simulation images:

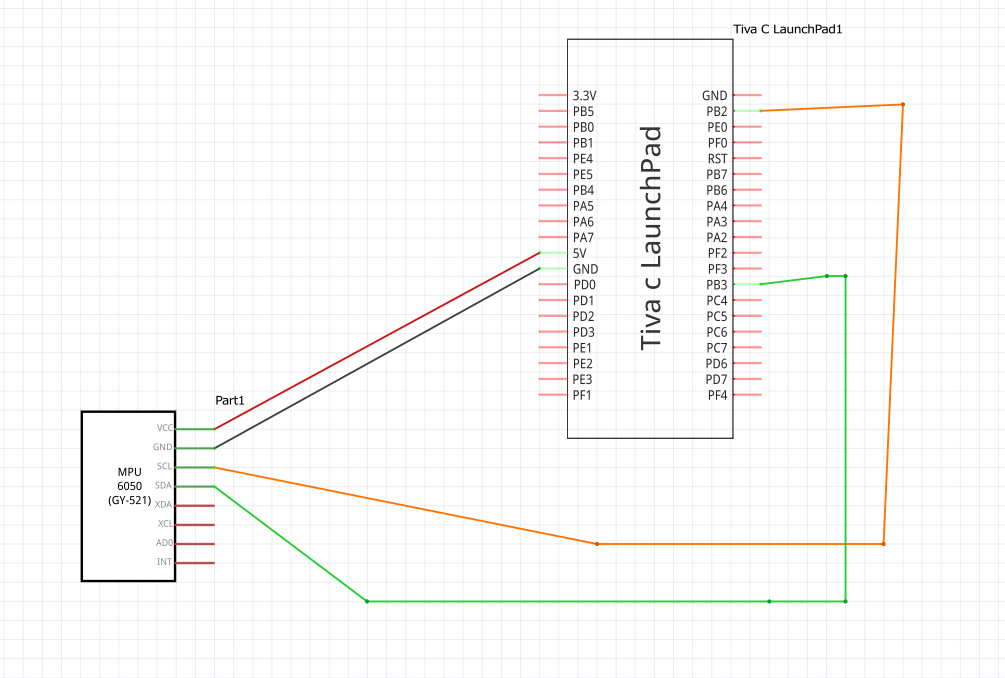
Task 1-2



Task 3-4



Schematic



IIMPLEMENTATION:

CODE:

Task 1-2:

/\*

Author Name: John Patrick Buen

Midterm 1 Code

Date: 10/31/2019

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Goal: Interface with MPU6050 IMU module using I2C given

various tasks.

Steps:

Task 1: Interface the given MPU6050 IMU using I2C protocol to TivaC. Print all accelerometer

and gyro values on to the serial terminal.

Task 2: Interface the given MPU6050 IMU using I2C protocol to TivaC. Plot all accelerometer

and gyro values on to a Graph (you can use any graphing tool).

Task 3: Implement a complementary filter to filter the raw accelerometer and gyro values. Print all

raw and filtered accelerometer and gyro values on to the serial terminal. Implement the filter using IQMath Library.

Task 4: Implement a complementary filter to filter the raw accelerometer and gyro values. Plot all

raw and filtered accelerometer and gyro values on to a Graph (you can use any graphing tool).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Chip type : ARM TM4C123GH6PM (Tiva C)

Program type : Firmware

Core Clock frequency : 80.000000 MHz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <stdarg.h>

**#include** <stdbool.h>

**#include** <stdint.h>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**#include** "sensorlib/i2cm\_drv.c"

**#include** "sensorlib/hw\_mpu6050.h"

**#include** "sensorlib/mpu6050.h"

**#include** "uartstdio.h"

//global variables

**volatile** bool g\_bMPU6050Done;

tI2CMInstance sI2CInst;

//interrupt subroutine for I2C

**void** **I2CIntHandler**(**void**)

{

**I2CMIntHandler**(&sI2CInst);

}

**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. The last parameter sets the I2C data transfer rate.

// If false, the data rate is set to 100kbps and if true the data rate will

// be set to 400kbps.

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

//clear I2C FIFOs

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

// Initialize the I2C master driver.

**I2CMInit**(&sI2CInst, I2C0\_BASE, INT\_I2C0, 0xff, 0xff, **SysCtlClockGet**());

}

**void** **UART\_Init\_Config**(**void**)

{

//using UART module 0

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0); //enable UART0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //enable GPIOA peripheral

//Configure pins Rx and Tx for UART

**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); //using baudrate of 115200

}

**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;

}

//MPU6050 Example

**void** **MPU6050Example**(**void**)

{

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

tMPU6050 sMPU6050;

// Initialize the MPU6050. This code assumes that the I2C master instance

// has already been initialized.

g\_bMPU6050Done = false;

**MPU6050Init**(&sMPU6050, &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)

{

}

// turn on the MPU6050 (power on)

g\_bMPU6050Done = false;

**MPU6050ReadModifyWrite**(&sMPU6050, MPU6050\_O\_PWR\_MGMT\_1, 0x00, 0x00, MPU6050Callback, &sMPU6050);

**while** (!g\_bMPU6050Done)

{

}

// Loop forever keep reading data from 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]);

//display Accel and Gyro values on terminal

UARTprintf("ACCEL: X: %d,\t Y: %d,\t Z: %d\n", (**int**)fAccel[0], (**int**)fAccel[1], (**int**)fAccel[2]);

UARTprintf("GYRO: X: %d,\t Y: %d,\t Z: %d\n", (**int**)fGyro[0], (**int**)fGyro[1], (**int**)fGyro[2]);

**SysCtlDelay**(**SysCtlClockGet**()/10);

}

}

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

{

**SysCtlClockSet**(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_INT | SYSCTL\_XTAL\_16MHZ);

//call InitI2C0

InitI2C0();

//call UART\_Init\_Config

UART\_Init\_Config();

//call MPU6050Example

MPU6050Example();

**return** 0;

}

Task 3-4:

/\*

Author Name: John Patrick Buen

Midterm 1 Code

Date: 10/31/2019

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Goal: Interface with MPU6050 IMU module using I2C given

various tasks.

Steps:

Task 1: Interface the given MPU6050 IMU using I2C protocol to TivaC. Print all accelerometer

and gyro values on to the serial terminal.

Task 2: Interface the given MPU6050 IMU using I2C protocol to TivaC. Plot all accelerometer

and gyro values on to a Graph (you can use any graphing tool).

Task 3: Implement a complementary filter to filter the raw accelerometer and gyro values. Print all

raw and filtered accelerometer and gyro values on to the serial terminal. Implement the filter using IQMath Library.

Task 4: Implement a complementary filter to filter the raw accelerometer and gyro values. Plot all

raw and filtered accelerometer and gyro values on to a Graph (you can use any graphing tool).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Chip type : ARM TM4C123GH6PM (Tiva C)

Program type : Firmware

Core Clock frequency : 80.000000 MHz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <stdarg.h>

**#include** <stdbool.h>

**#include** <stdint.h>

**#include** <math.h>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**#include** "sensorlib/i2cm\_drv.c"

**#include** "sensorlib/hw\_mpu6050.h"

**#include** "sensorlib/mpu6050.h"

**#include** "uartstdio.h"

**#include** "IQmath/IQmathLib.h"

// define and configurations of accelerometer

// and gyroscope modules on MPU6050 as well as

// sample rate, ratio(180deg/3.14), and pi

**#define** ACCELEROMETER\_SENSITIVITY 8192.0

**#define** GYROSCOPE\_SENSITIVITY 131

**#define** SAMPLE\_RATE 0.01

**#define** RATIO (180/3.14)

**#define** M\_PI 3.14

**volatile** bool g\_bMPU6050Done;

tI2CMInstance sI2CInst;

//I2C interrupt subroutine

**void** **I2CIntHandler**(**void**)

{

**I2CMIntHandler**(&sI2CInst);

}

**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. The last parameter sets the I2C data transfer ratioe.

// If false, the data ratioe is set to 100kbps and if true the data ratioe will

// be set to 400kbps.

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

//clear I2C FIFOs

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

// Initialize the I2C master driver.

**I2CMInit**(&sI2CInst, I2C0\_BASE, INT\_I2C0, 0xff, 0xff, **SysCtlClockGet**());

}

**void** **UART\_Init\_Config**(**void**)

{

//using UART module 0

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0); //enable UART0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //enable GPIOA peripheral

//Configure pins Rx and Tx for UART

**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); //using baudratioe of 115200

}

**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;

}

// This function will be used to filter the raw accel and gyroscope

// values and display them on the serial terminal.

**void** **ComplementaryFilter**(**float** accData[3], **float** gyroData[3])

{

\_iq16 ForceMagApprx, PitchAcc, RollAcc, YawAcc, sensitivity, ratio, num1, num2;

\_iq16 Gyro[3], Acc[3];

\_iq16 Pitch = 0; //y value for gyro

\_iq16 Roll = 0; //x value for gyro

\_iq16 Yaw = 0; //z value for gyro

//initialize values using IQ math

ratio = \_IQ16(RATIO);

num1 = \_IQ16(0.98);

num2 = \_IQ16(0.02);

Gyro[0] = \_IQ16(gyroData[0]);

Gyro[1] = \_IQ16(gyroData[1]);

Gyro[2] = \_IQ16(gyroData[2]);

Acc[0] = \_IQ16(accData[0]);

Acc[1] = \_IQ16(accData[1]);

Acc[2] = \_IQ16(accData[2]);

sensitivity = \_IQ16(GYROSCOPE\_SENSITIVITY);

//compute pitch, roll, and yaw values

Pitch += **\_IQ16mpy**(**\_IQ16div**(Gyro[0],sensitivity), \_IQ16(SAMPLE\_RATE));

Roll -= **\_IQ16mpy**(**\_IQ16div**(Gyro[1],sensitivity), \_IQ16(SAMPLE\_RATE));

Yaw = **\_IQ16mpy**(**\_IQ16div**(Gyro[2],sensitivity), \_IQ16(SAMPLE\_RATE));

//compute magnitude of force

ForceMagApprx = \_IQabs(Acc[0]) + \_IQabs(Acc[1]) + \_IQabs(Acc[2]);

**if**(ForceMagApprx > 1411510 && ForceMagApprx < 4705028)

{

//compute gyro and accelerometer values based on magnitude of force

PitchAcc = **\_IQ16mpy**(**\_IQ16atan2**(Acc[1],Acc[2]), ratio);

Pitch = **\_IQ16mpy**(Pitch,num1) + **\_IQ16mpy**(PitchAcc,num2);

RollAcc = **\_IQ16mpy**(**\_IQ16atan2**(Acc[0],Acc[2]), ratio);

Roll = **\_IQ16mpy**(Roll,num1) + **\_IQ16mpy**(RollAcc,num2);

YawAcc = **\_IQ16mpy**(**\_IQ16atan2**(Acc[0],Acc[1]), ratio);

Yaw = **\_IQ16mpy**(Yaw,num1) + **\_IQ16mpy**(YawAcc,num2);

//Display roll, pitch, and yaw

UARTprintf("Roll: %d,\t Pitch: %d,\t Yaw: %d\n", (**int**)Roll, (**int**)Pitch, (**int**)Yaw);

}

}

**void** **MPU6050Example**(**void**)

{

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

uint\_fast16\_t f\_accel[3]; // variable for raw accelerometer values

uint\_fast16\_t f\_gyro[3]; // variable for raw gyroscope values

**float** ax, ay, az, gx, gy, gz; // temp variables for displaying

tMPU6050 sMPU6050;

// Initialize the MPU6050. This code assumes that the I2C master instance

// has already been initialized.

g\_bMPU6050Done = false;

**MPU6050Init**(&sMPU6050, &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)

{

}

// turn on the MPU6050 (power on)

g\_bMPU6050Done = false;

**MPU6050ReadModifyWrite**(&sMPU6050, MPU6050\_O\_PWR\_MGMT\_1, 0x00, 0x00, MPU6050Callback, &sMPU6050);

**while** (!g\_bMPU6050Done)

{

}

// Loop forever keep reading data from MPU6050

**while** (1)

{

// Request another reading from the MPU6050.

g\_bMPU6050Done = false;

**MPU6050DataRead**(&sMPU6050, MPU6050Callback, &sMPU6050);

**while** (!g\_bMPU6050Done)

{

}

// Get accelerometer and gyroscope readings - FLOAT values.

// These values will be passed into the complementary filter function

**MPU6050DataAccelGetFloat**(&sMPU6050, &fAccel[0], &fAccel[1], &fAccel[2]);

**MPU6050DataGyroGetFloat**(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro[2]);

// Get accelerometer and gyroscope readings - RAW values

**MPU6050DataAccelGetRaw**(&sMPU6050, &f\_accel[0], &f\_accel[1], &f\_accel[2]);

**MPU6050DataGyroGetRaw**(&sMPU6050, &f\_gyro[0], &f\_gyro[1], &f\_gyro[2]);

//display Accel and Gyro values on terminal

UARTprintf("///////////////RAW VALUES//////////////////////////////\n");

UARTprintf("ACCEL: X: %d,\t Y: %d,\t Z: %d\n", (uint\_fast16\_t)f\_accel[0], (uint\_fast16\_t)f\_accel[1], (uint\_fast16\_t)f\_accel[2]);

UARTprintf("GYRO: X: %d,\t Y: %d,\t Z: %d\n", (uint\_fast16\_t)f\_gyro[0], (uint\_fast16\_t)f\_gyro[1], (uint\_fast16\_t)f\_gyro[2]);

//compute gyro and accel values prior to filtering

ax = (**atan2**(fAccel[0], **sqrt**((fAccel[1]\*fAccel[1]) + (fAccel[2] \* fAccel[2])))\*180.0)/M\_PI;

ay = (**atan2**(fAccel[1], **sqrt**((fAccel[0]\*fAccel[0]) + (fAccel[2] \* fAccel[2])))\*180.0)/M\_PI;

az = (**atan2**(fAccel[2], **sqrt**((fAccel[1]\*fAccel[1]) + (fAccel[2] \* fAccel[2])))\*180.0)/M\_PI;

gx = fGyro[0]\*10000;

gy = fGyro[1]\*10000;

gz = fGyro[2]\*10000;

//call the complementary filter

ComplementaryFilter(fAccel, fGyro);

//Display filtered values on terminal

UARTprintf("///////////////FILTERED VALUES//////////////////////////////\n");

UARTprintf("ACCEL: X: %d,\t Y: %d,\t Z: %d\n", (**int**)ax, (**int**)ay, (**int**)az);

UARTprintf("GYRO: X: %d,\t Y: %d,\t Z: %d\n", (**int**)gx, (**int**)gy, (**int**)gz);

**SysCtlDelay**(**SysCtlClockGet**()/10);

}

}

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

{

**SysCtlClockSet**(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_INT | SYSCTL\_XTAL\_16MHZ);

//call InitI2C0

InitI2C0();

//call UART\_Init\_Config

UART\_Init\_Config();

//call MPU6050Example

MPU6050Example();

**return** 0;

}

Name: John Patrick Buen Page 1/1