CPE 403 ADV EMB SYS DES F 2019

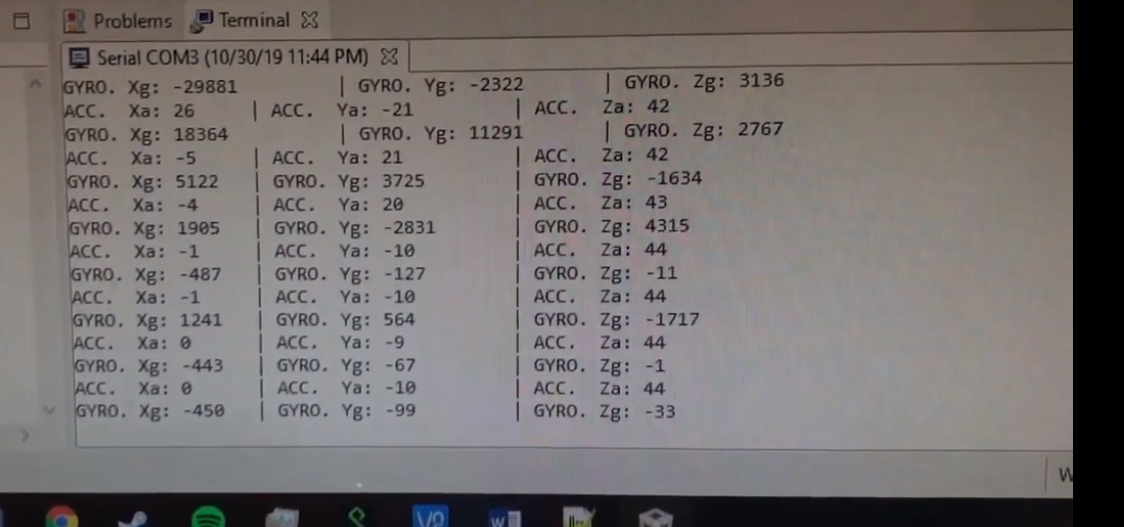
TITLE: MIDTERM I

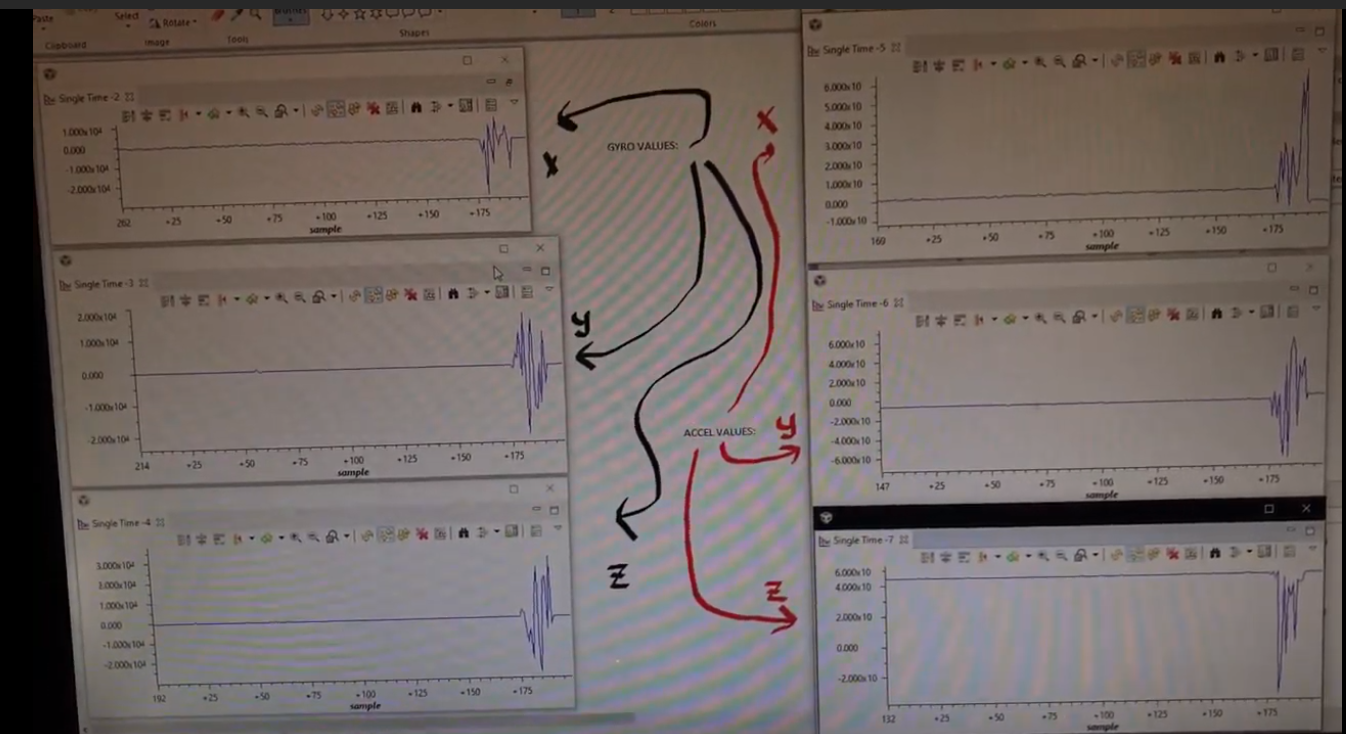
GOAL:

Task01&02: The goal of task 1 ant two is to interface the given MPU6050 IMU using

I2C protocol to TIVAC. The values will be printed on the serial terminal,

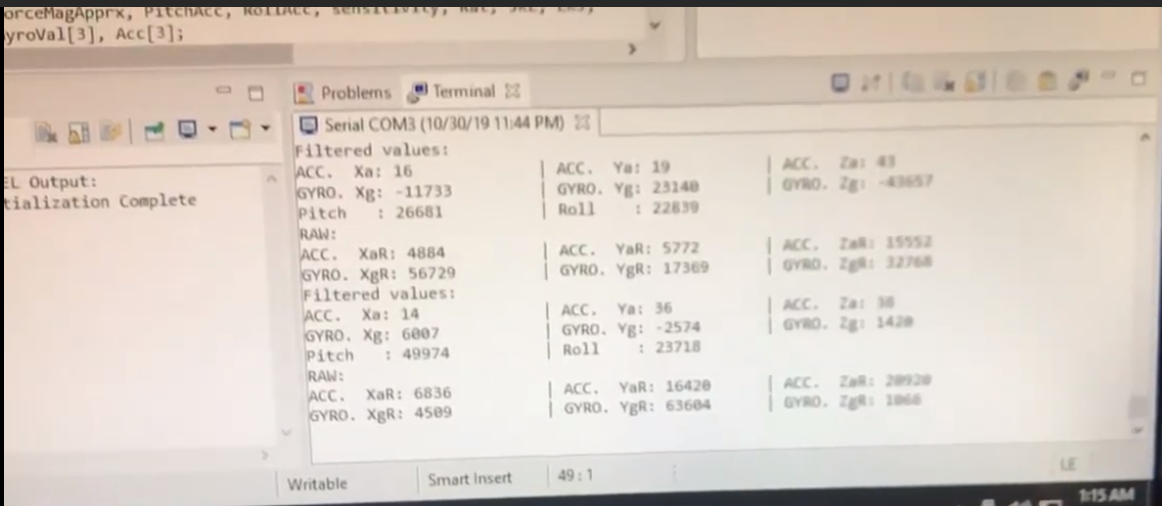
and they can also be printed using the graph on code composer studio.

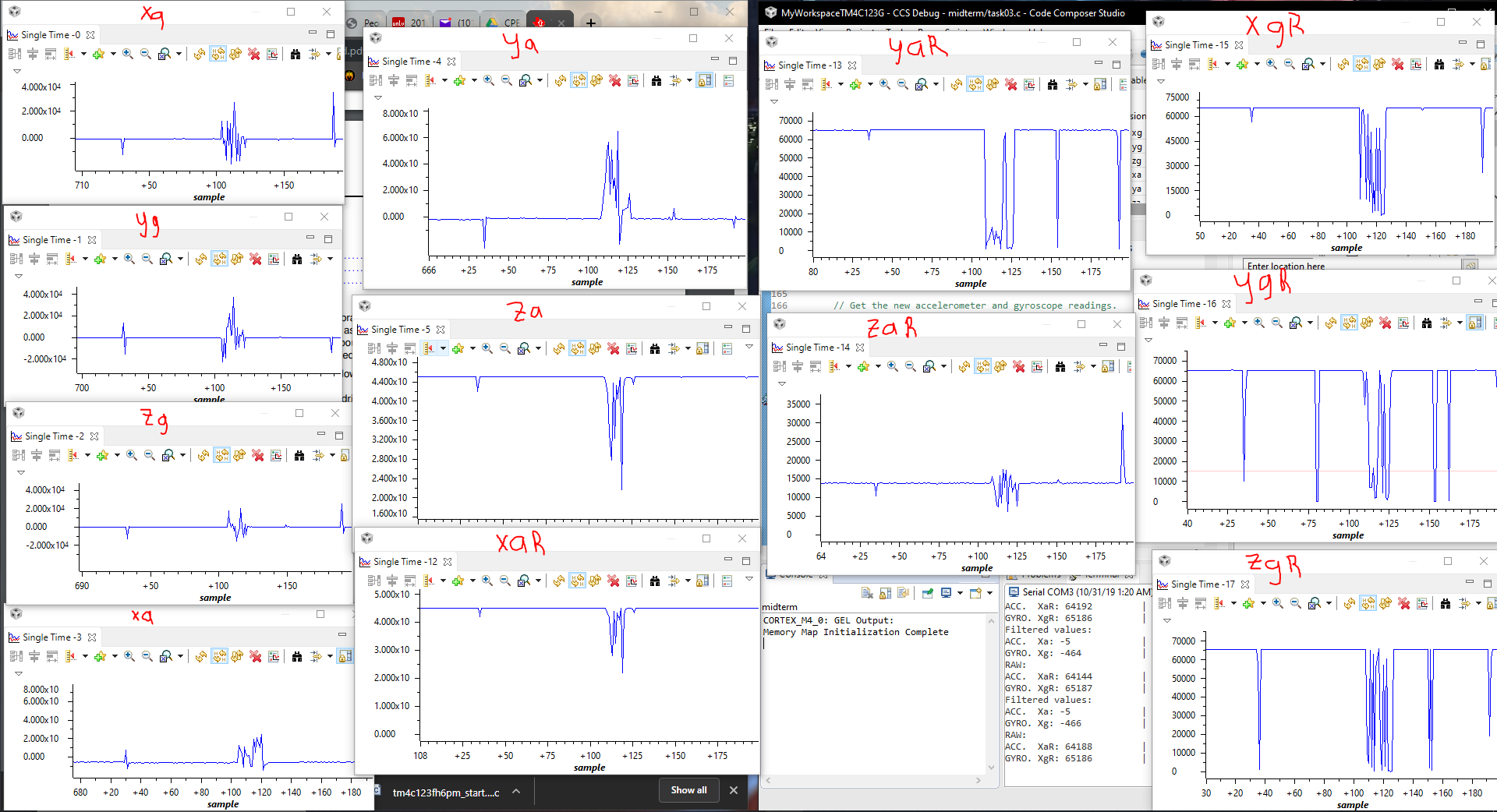




Task03&04: The goal of this task is to implement a complementary filter to filter the

raw values from the accelerometer and gyro.





DELIVERABLES:

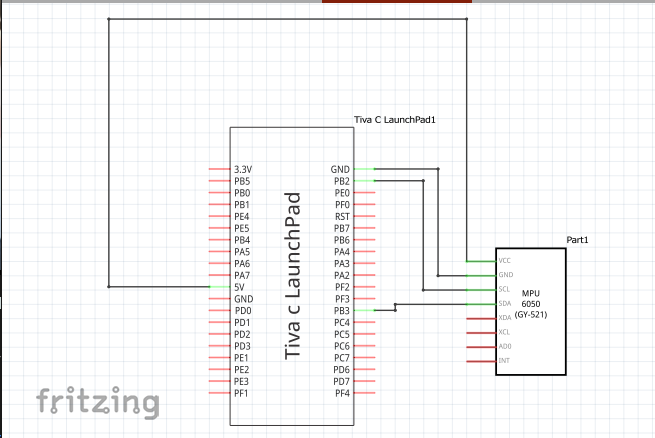
The deliverables include showing the accelerometer and gyro values through either the serial terminal or the the graphing tool on Code Composer Studio.

COMPONENTS:

Tiva C Series Launchpad Ek-TM4C123GXL

MPU6050

SCHEMATICS:



IIMPLEMENTATION:

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 ant 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

tI2CMInstance g\_sI2CMInst; // 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\_sI2CMInst);

}

**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\_sI2CMInst, 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.

**I2CMInit**(&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

tI2CMInstance g\_sI2CMInst; // 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\_sI2CMInst, 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]);

MPU6050DataAccelGetRaw(&sMPU6050, &fAccelRaw[0], &fAccelRaw[1], &fAccelRaw[2]);

MPU6050DataGyroGetRaw(&sMPU6050, &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\_FIFOCTL) = 80008000;

// Initialize the I2C master driver.

**I2CMInit**(&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 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);

}

Name: Serak Gebremedhin Page 1/1