Title: Using the MPU6050

GitHub:<https://github.com/westbrian2/Fall2019>

Goals:

* Interface with the MPU6050 using I2C.
* Graph the values collected from the MPU6050.
* Filter the values using a complementary Filtering.
* Graph the raw and filtered values.

Deliverables:

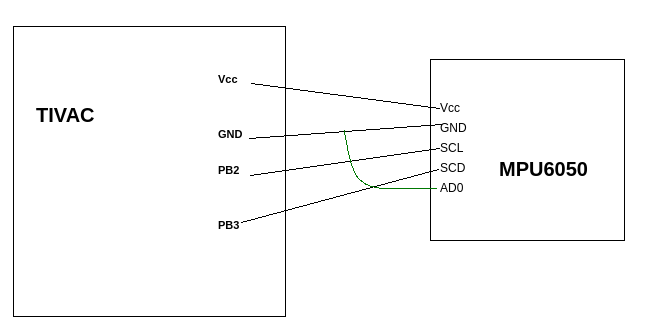
* The purpose of the project was to ultimately take the values from the MPU to determine the roll and pitch the device is experiencing. As such the programs were the steps taken to get the roll and pitch. The final product is a program that outputs roll, pitch, gyro values, and the accelerometer values.

Components:

* TIVA C board
* MPU 6050

The main limitations of this project was the lack of documentation with the IQmath. Despite the one support document that outlines the API, there is no documentation surrounding what operators can be used without type conversions. This may have lead to more processing time needed than was strictly necessary.

Schematic:

Code Task1: Getting data from the MPU using I2C

//Noted functions were used from:

// https://www.digikey.com/eewiki/display/microcontroller/I2C+Communication+with+the+TI+Tiva+TM4C123GXL

#include <stdarg.h>

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "inc/hw\_i2c.h"

#include "driverlib/i2c.h"

#include "inc/hw\_gpio.h"

#include "utils/uartstdio.h"

#include "utils/uartstdio.c"

static uint32\_t mpu6050add = 0x68;

static uint8\_t acc\_xh\_reg = 59;

static uint8\_t acc\_xl\_reg = 60;

static uint8\_t acc\_yh\_reg = 61;

static uint8\_t acc\_yl\_reg = 62;

static uint8\_t acc\_zh\_reg = 63;

static uint8\_t acc\_zl\_reg = 64;

static uint8\_t gyro\_xh\_reg = 67;

static uint8\_t gyro\_xl\_reg = 68;

static uint8\_t gyro\_yh\_reg = 69;

static uint8\_t gyro\_yl\_reg = 70;

static uint8\_t gyro\_zh\_reg = 71;

static uint8\_t gyro\_zl\_reg = 72;

//setting up I2C (function was taken from source

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(), false);

//clear I2C FIFOs

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

}

//read specified register on slave device

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

uint16\_t x = I2CMasterDataGet(I2C0\_BASE);

return I2CMasterDataGet(I2C0\_BASE);

}

void I2CSend(uint8\_t slave\_addr, uint8\_t num\_of\_args, ...){

// Tell the master module what address it will place on the bus when

// communicating with the slave.

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

//stores list of variable number of arguments

va\_list vargs;

//specifies the va\_list to "open" and the last fixed argument

//so vargs knows where to start looking

va\_start(vargs, num\_of\_args);

//put data to be sent into FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//if there is only one argument, we only need to use the

//single send I2C function

if(num\_of\_args == 1)

{

//Initiate send of data from the MCU

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//"close" variable argument list

va\_end(vargs);

}

//otherwise, we start transmission of multiple bytes on the

//I2C bus

else

{

//Initiate send of data from the MCU

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//send num\_of\_args-2 pieces of data, using the

//BURST\_SEND\_CONT command of the I2C modules

uint8\_t i;

for(i = 1; i < (num\_of\_args - 1); i++)

{

//put next piece of data into I2C FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//send next data that was just placed into FIFO

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

}

//put last piece of data into I2C FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//send next data that was just placed into FIFO

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//"close" variable args list

va\_end(vargs);

}

}

void InitConsole(void){

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

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

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

UARTStdioConfig(0, 115200, 16000000);

}

int main(void) {

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

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

InitConsole();

InitI2C0();

uint16\_t accel\_x;

uint16\_t accel\_y;

uint16\_t accel\_z;

uint16\_t gyro\_x;

uint16\_t gyro\_y;

uint16\_t gyro\_z;

I2CSend(mpu6050add,0,0x6B);

while (1)

{

accel\_x = I2CReceive(mpu6050add, acc\_xh\_reg); //getting accel values

accel\_x += I2CReceive(mpu6050add, acc\_xl\_reg); //adding lower half to upper half

accel\_y = I2CReceive(mpu6050add, acc\_yh\_reg);

accel\_y +=I2CReceive(mpu6050add, acc\_yl\_reg);

accel\_z =I2CReceive(mpu6050add, acc\_zh\_reg);

accel\_z +=I2CReceive(mpu6050add, acc\_zl\_reg);

gyro\_x = I2CReceive(mpu6050add, gyro\_xh\_reg);

gyro\_x +=I2CReceive(mpu6050add, gyro\_xl\_reg);

gyro\_y =I2CReceive(mpu6050add, gyro\_yh\_reg);

gyro\_y +=I2CReceive(mpu6050add, gyro\_yl\_reg);

gyro\_z =I2CReceive(mpu6050add, gyro\_zh\_reg);

gyro\_z +=I2CReceive(mpu6050add, gyro\_zl\_reg);

UARTprintf("Accel X: %d ", accel\_x); //Printing the values to UART

UARTprintf("Accel Y: %d ", accel\_y);

UARTprintf("Accel Z: %d\n", accel\_z);

UARTprintf("Gyro X : %d ", gyro\_x);

UARTprintf("Gyro Y : %d ", gyro\_x);

UARTprintf("Gyro Z : %d\n", gyro\_x);

SysCtlDelay(80000000 / 12); //slowing down output for readability

}

}

Code Task2: Basically the same as task2, but with CSV in the UART output

//Noted functions were used from:

// https://www.digikey.com/eewiki/display/microcontroller/I2C+Communication+with+the+TI+Tiva+TM4C123GXL

#include <stdarg.h>

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "inc/hw\_i2c.h"

#include "driverlib/i2c.h"

#include "inc/hw\_gpio.h"

#include "utils/uartstdio.h"

#include "utils/uartstdio.c"

static uint32\_t mpu6050add = 0x68;//values to make I2C stuff easier

static uint8\_t acc\_xh\_reg = 59;

static uint8\_t acc\_xl\_reg = 60;

static uint8\_t acc\_yh\_reg = 61;

static uint8\_t acc\_yl\_reg = 62;

static uint8\_t acc\_zh\_reg = 63;

static uint8\_t acc\_zl\_reg = 64;

static uint8\_t gyro\_xh\_reg = 67;

static uint8\_t gyro\_xl\_reg = 68;

static uint8\_t gyro\_yh\_reg = 69;

static uint8\_t gyro\_yl\_reg = 70;

static uint8\_t gyro\_zh\_reg = 71;

static uint8\_t gyro\_zl\_reg = 72;

//setting up I2C (function was taken from source

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(), false);

//clear I2C FIFOs

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

}

//read specified register on slave device

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

uint16\_t x = I2CMasterDataGet(I2C0\_BASE);

return I2CMasterDataGet(I2C0\_BASE);

}

void I2CSend(uint8\_t slave\_addr, uint8\_t num\_of\_args, ...){

// Tell the master module what address it will place on the bus when

// communicating with the slave.

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

//stores list of variable number of arguments

va\_list vargs;

//specifies the va\_list to "open" and the last fixed argument

//so vargs knows where to start looking

va\_start(vargs, num\_of\_args);

//put data to be sent into FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//if there is only one argument, we only need to use the

//single send I2C function

if(num\_of\_args == 1)

{

//Initiate send of data from the MCU

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//"close" variable argument list

va\_end(vargs);

}

//otherwise, we start transmission of multiple bytes on the

//I2C bus

else

{

//Initiate send of data from the MCU

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//send num\_of\_args-2 pieces of data, using the

//BURST\_SEND\_CONT command of the I2C modules

uint8\_t i;

for(i = 1; i < (num\_of\_args - 1); i++)

{

//put next piece of data into I2C FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//send next data that was just placed into FIFO

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

}

//put last piece of data into I2C FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//send next data that was just placed into FIFO

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//"close" variable args list

va\_end(vargs);

}

}

void InitConsole(void){

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

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

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

UARTStdioConfig(0, 115200, 16000000);

}

int main(void) {

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

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

InitConsole();

InitI2C0();

uint16\_t accel\_x;

uint16\_t accel\_y;

uint16\_t accel\_z;

uint16\_t gyro\_x;

uint16\_t gyro\_y;

uint16\_t gyro\_z;

I2CSend(mpu6050add,0,0x6B);

while (1)

{

accel\_x = I2CReceive(mpu6050add, acc\_xh\_reg);//still just getting values

accel\_x += I2CReceive(mpu6050add, acc\_xl\_reg);

accel\_y = I2CReceive(mpu6050add, acc\_yh\_reg);

accel\_y +=I2CReceive(mpu6050add, acc\_yl\_reg);

accel\_z =I2CReceive(mpu6050add, acc\_zh\_reg);

accel\_z +=I2CReceive(mpu6050add, acc\_zl\_reg);

gyro\_x = I2CReceive(mpu6050add, gyro\_xh\_reg);

gyro\_x +=I2CReceive(mpu6050add, gyro\_xl\_reg);

gyro\_y =I2CReceive(mpu6050add, gyro\_yh\_reg);

gyro\_y +=I2CReceive(mpu6050add, gyro\_yl\_reg);

gyro\_z =I2CReceive(mpu6050add, gyro\_zh\_reg);

gyro\_z +=I2CReceive(mpu6050add, gyro\_zl\_reg);

UARTprintf("AccelX:%d,", accel\_x); //Formatting was used to work with graphing utility

UARTprintf("AccelY:%d,", accel\_y);// similar to CSV

UARTprintf("AccelZ:%d,", accel\_z);

UARTprintf("GyroX:%d,", gyro\_x);

UARTprintf("GyroY:%d,", gyro\_x);

UARTprintf("GyroZ:%d\n", gyro\_x);

}

}

Code Task 3 & Task 4: Literally the same code just with an added filter to get the roll and pitch values.

//Noted functions were used from:

// https://www.digikey.com/eewiki/display/microcontroller/I2C+Communication+with+the+TI+Tiva+TM4C123GXL

#define GLOBAL\_Q 8

#define ACCELEROMETER\_SENSITIVITY 8192.0

#define GYROSCOPE\_SENSITIVITY 65.536

#define M\_PI 3.14159265359

#define dt 0.01 // 10 ms sample rate!

#include <stdarg.h>

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "inc/hw\_i2c.h"

#include "driverlib/i2c.h"

#include "inc/hw\_gpio.h"

#include "utils/uartstdio.h"

#include "utils/uartstdio.c"

#include "IQmath/IQmathLib.h"

#include <string.h>

#include <stdio.h>

static uint32\_t mpu6050add = 0x68;

static uint8\_t acc\_xh\_reg = 59;

static uint8\_t acc\_xl\_reg = 60;

static uint8\_t acc\_yh\_reg = 61;

static uint8\_t acc\_yl\_reg = 62;

static uint8\_t acc\_zh\_reg = 63;

static uint8\_t acc\_zl\_reg = 64;

static uint8\_t gyro\_xh\_reg = 67;

static uint8\_t gyro\_xl\_reg = 68;

static uint8\_t gyro\_yh\_reg = 69;

static uint8\_t gyro\_yl\_reg = 70;

static uint8\_t gyro\_zh\_reg = 71;

static uint8\_t gyro\_zl\_reg = 72;

//setting up I2C (function was taken from source

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(), false);

//clear I2C FIFOs

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

}

//read specified register on slave device

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

uint16\_t x = I2CMasterDataGet(I2C0\_BASE);

return I2CMasterDataGet(I2C0\_BASE);

}

void I2CSend(uint8\_t slave\_addr, uint8\_t num\_of\_args, ...){

// Tell the master module what address it will place on the bus when

// communicating with the slave.

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

//stores list of variable number of arguments

va\_list vargs;

//specifies the va\_list to "open" and the last fixed argument

//so vargs knows where to start looking

va\_start(vargs, num\_of\_args);

//put data to be sent into FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//if there is only one argument, we only need to use the

//single send I2C function

if(num\_of\_args == 1)

{

//Initiate send of data from the MCU

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//"close" variable argument list

va\_end(vargs);

}

//otherwise, we start transmission of multiple bytes on the

//I2C bus

else

{

//Initiate send of data from the MCU

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//send num\_of\_args-2 pieces of data, using the

//BURST\_SEND\_CONT command of the I2C modules

uint8\_t i;

for(i = 1; i < (num\_of\_args - 1); i++)

{

//put next piece of data into I2C FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//send next data that was just placed into FIFO

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

}

//put last piece of data into I2C FIFO

I2CMasterDataPut(I2C0\_BASE, va\_arg(vargs, uint32\_t));

//send next data that was just placed into FIFO

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

// Wait until MCU is done transferring.

while(I2CMasterBusy(I2C0\_BASE));

//"close" variable args list

va\_end(vargs);

}

}

void InitConsole(void){

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

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

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

UARTStdioConfig(0, 115200, 16000000);

}

int main(void) {

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

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

InitConsole();

InitI2C0();

char pstring[50];

\_iq pitch,roll = \_IQ(0.0);

\_iq pitchAcc, rollAcc = \_IQ(0.0);

float accel[3];

\_iq iq\_accel[3];

float test;

float gyro[3];

\_iq iq\_gyro[3];

uint8\_t i;

I2CSend(mpu6050add,0,0x6B);

while (1)

{

accel[0] =(float) I2CReceive(mpu6050add, acc\_xh\_reg);

accel[0] +=(float) I2CReceive(mpu6050add, acc\_xl\_reg);

accel[1] =(float) I2CReceive(mpu6050add, acc\_yh\_reg);

accel[1] +=(float)I2CReceive(mpu6050add, acc\_yl\_reg);

accel[2] =(float)I2CReceive(mpu6050add, acc\_zh\_reg);

accel[2] +=(float)I2CReceive(mpu6050add, acc\_zl\_reg);

gyro[0] =(float)I2CReceive(mpu6050add, gyro\_xh\_reg);

gyro[0] +=(float)I2CReceive(mpu6050add, gyro\_xl\_reg);

gyro[1] =(float)I2CReceive(mpu6050add, gyro\_yh\_reg);

gyro[1] +=(float)I2CReceive(mpu6050add, gyro\_yl\_reg);

gyro[2] =(float)I2CReceive(mpu6050add, gyro\_zh\_reg);

gyro[2] +=(float)I2CReceive(mpu6050add, gyro\_zl\_reg);

for(i = 0;i<3;i++){

iq\_accel[i] = \_IQ(accel[i]);

iq\_gyro[i] = \_IQ(gyro[i]);

}

test = \_IqtoF(iq\_gyro[0]); //Everything past here is filtering the data to get roll and pitch

// Integrate the gyroscope data -> int(angularSpeed) = angle

// Angle around the X-axis

pitch += \_IQmpy(\_IQdiv(iq\_gyro[0],\_IQ(GYROSCOPE\_SENSITIVITY)), \_IQ(dt));

test = \_IQtoF(pitch);

// Angle around the Y-axis

roll -= \_IQmpy(\_IQdiv(iq\_gyro[1],\_IQ(GYROSCOPE\_SENSITIVITY)),\_IQ(dt));

test = \_IQtoF(roll);

// Compensate for drift with accelerometer data

// Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192

\_iq forceMagnitudeApprox = \_IQabs(iq\_accel[0]) + \_IQabs(iq\_accel[1]) + \_IQabs(iq\_accel[2]);

if (\_IQint(forceMagnitudeApprox) > 8192 && \_IQint(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;

pitchAcc = \_IQdiv(\_IQmpy(\_IQatan2(iq\_accel[1],iq\_accel[2]),\_IQ(180.0)),\_IQ(M\_PI));

pitch = \_IQmpy(pitch,\_IQ(0.98)) + \_IQmpy(pitchAcc, \_IQ(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;

rollAcc = \_IQdiv(\_IQmpy(\_IQatan2(iq\_accel[0],iq\_accel[2]),\_IQ(180.0)),\_IQ(M\_PI));

roll = \_IQmpy(roll,\_IQ(0.98)) + \_IQmpy(rollAcc,\_IQ(0.02));

}

//special formatting for the graph stuff

sprintf(pstring,"Pitch:%f,",\_IQtoF(pitch));

UARTprintf("%s",pstring);

sprintf(pstring,"Roll:%f,",\_IQtoF(roll));

UARTprintf("%s",pstring);

sprintf(pstring,"AccelX:%f,",accel[0]);

UARTprintf("%s,",pstring);

sprintf(pstring,"AccelY:%f,",accel[1]);

UARTprintf("%s",pstring);

sprintf(pstring,"AccelZ:%f,",accel[2]);

UARTprintf("%s",pstring);

sprintf(pstring,"GyroX:%f,",gyro[0]);

UARTprintf("%s",pstring);

sprintf(pstring,"GyroY:%f,",gyro[1]);

UARTprintf("%s",pstring);

sprintf(pstring,"GyroZ:%f\n",gyro[2]);

UARTprintf("%s",pstring);

SysCtlDelay(80000000 / 12);

}

}

Code Task4:

Same as Task3, which I said already...