CpE 403 Adv Embedded Systems Brian West Fall 2019

Title: Using the MPU6050
GitHub:https://github.com/westbrian2/Fall2019

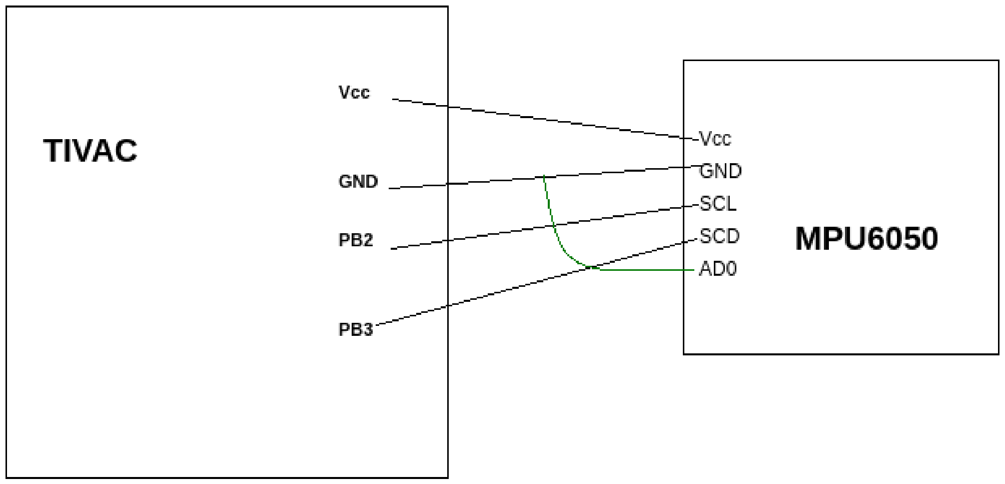
Goals:
- Interface with the MPU6050 using I2C.
- Graph the values collected from the MPU6050.
- Filter the values using a complementary Filtering.
- Graph the raw and filtered values.

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+TM
4C123GXL

#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+TM
4C123GXL
#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...