CPE 403 ADV EMB SYS DES F 2019

TITLE: Midterm 1

**GOAL**: The TIVA C will be used as the microcontroller of choice, so we’ll be using Code Composer Studio.. The goal of this project was to use the MPU6050 (Accelerometer & Gyroscope) to obtain raw values. Once those values are obtained, we’ll graph the accelerometer and gyroscope as well as print onto a terminal. Afterwards, we’ll use a filter to find the ‘pitch’ and the ‘roll’ values to graph and print. And to improve our precision calculation, we’ll introduce the IQMath library.

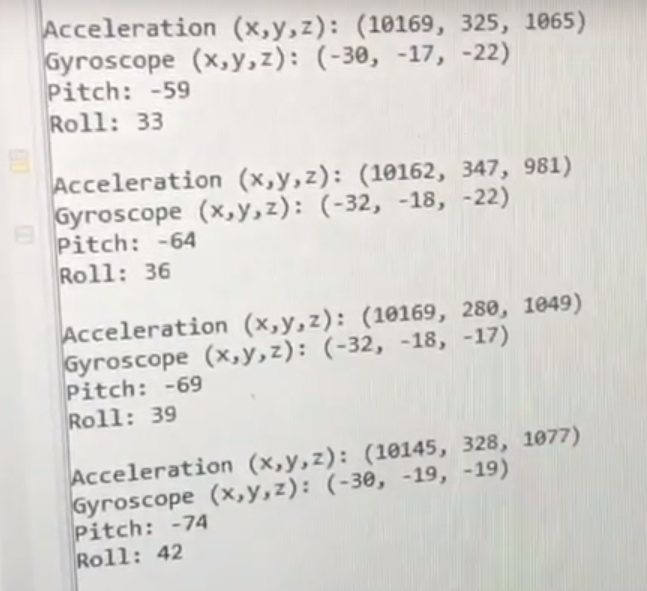
**Make bullet points of the project goal(s).**

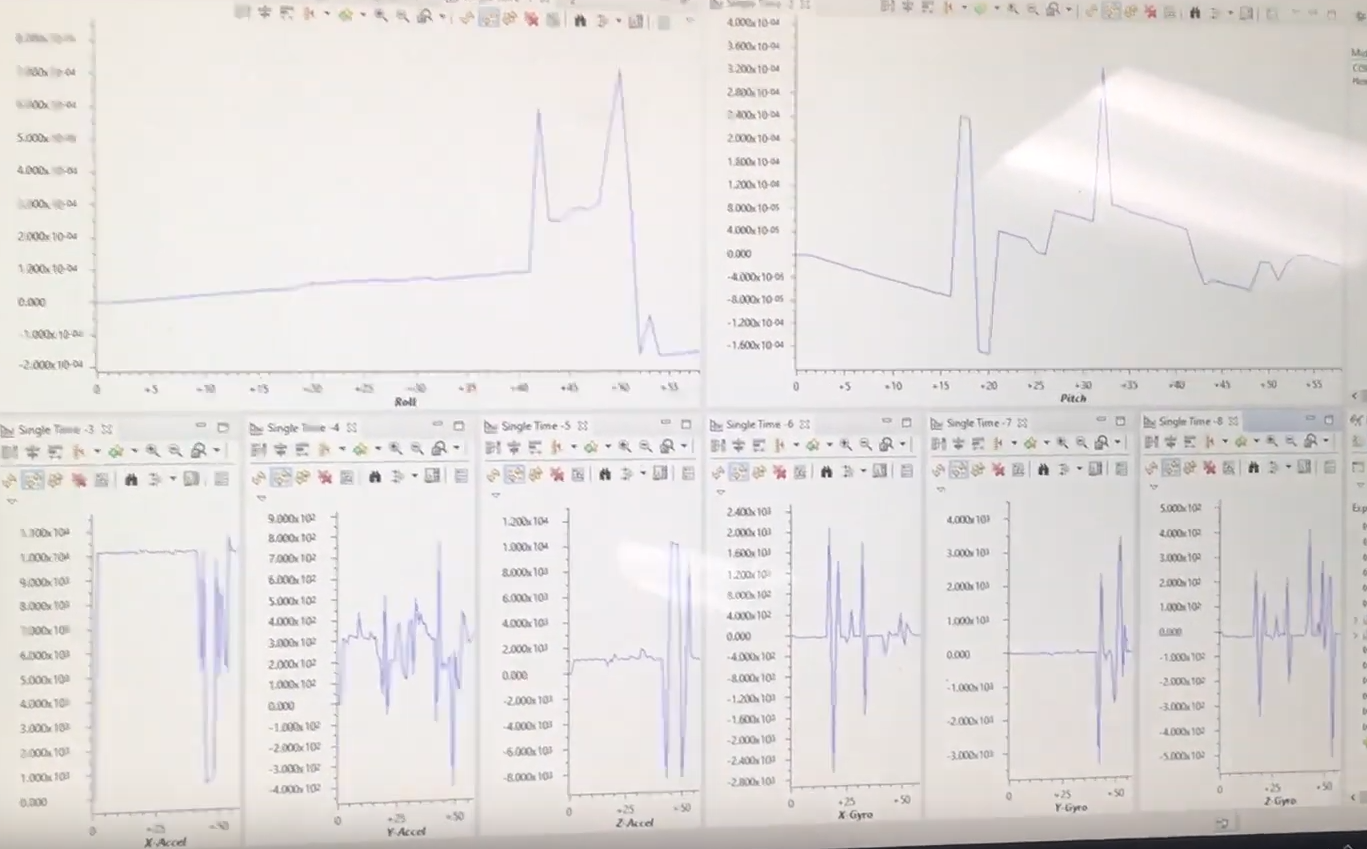
* Get the MPU6050 to work with the TIVA C by using I2C communication
* Graph and record the 3-dimensional raw values obtained by the MPU6050
* Using a filter, find the values of the pitch and roll just with the raw values given from the MPU6050 using its accelerometer and gyroscope
* Utilize the IQMath library to make precision calculations of the pitch and roll values

**DELIVERABLES:**

What is the intended project deliverables?   
Successfully complete a communication between the MPU6050 and the TIVA C using I2C.  
Record the raw 3-dimensional values of the accelerometer and gyroscope.  
Use UART to print those values onto a terminal.  
Use a graphing tool to graph those values.  
Implement the complementary filter to calculate the pitch and roll.  
Use the IQMath library to make a more precise calculation with the pitch and roll.  
Use UART and the graphing tool, again, to the new found values of the pitch and roll.

What was completed?   
Everything in the intended project deliverables above.

  
UART to print to serial terminal with Acceleration, Gyroscope, and Pitch/Roll values

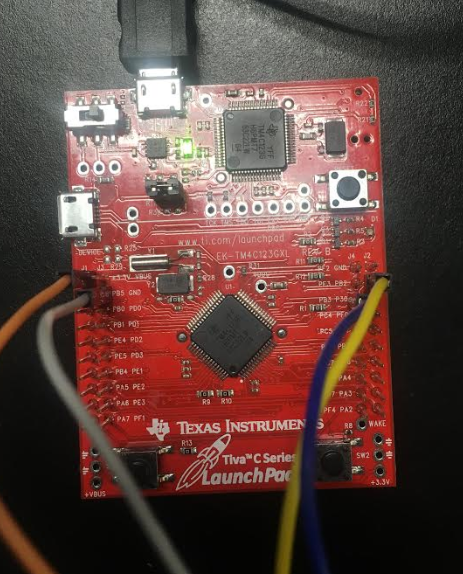
  
Graphs with the accelerometer, gyroscope, and pitch/roll values

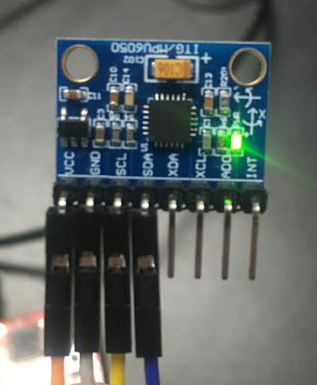
**COMPONENTS:**

Explain the main characteristics, interface, and limitation of the components used in the design, including the registered used and what was initialized? Why?

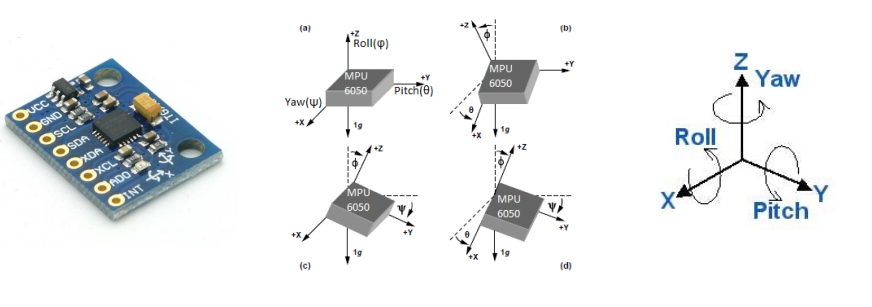
We used the TIVA C (TM4C123GXL) and the MPU6050 to complete this project. The software interface that was used was Code Composer Studio. Some limitations of the devices were the use of many libraries, 3.3V specific (weak regulator), and poor graph tool inside the software interface.  
I initialized the UART to speak to the serial terminal. I initialized the I2C to have both devices speak to each other.  
I used only registers PB2 and PB3 on the TIVA C to I2C communicate between the two devices.

**SCHEMATICS:**

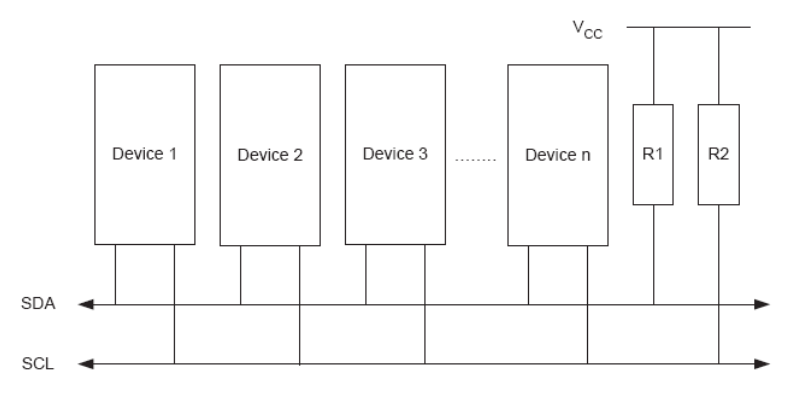
  
TIVA C (TM4C123GXL)

  
MPU6050

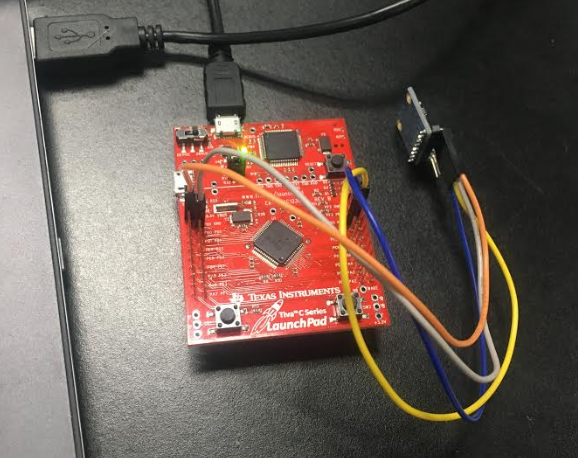
The way the MPU6050 works is through a 3-dimensional graph as shown below (obtained from Dr. Venki’s lecture material):

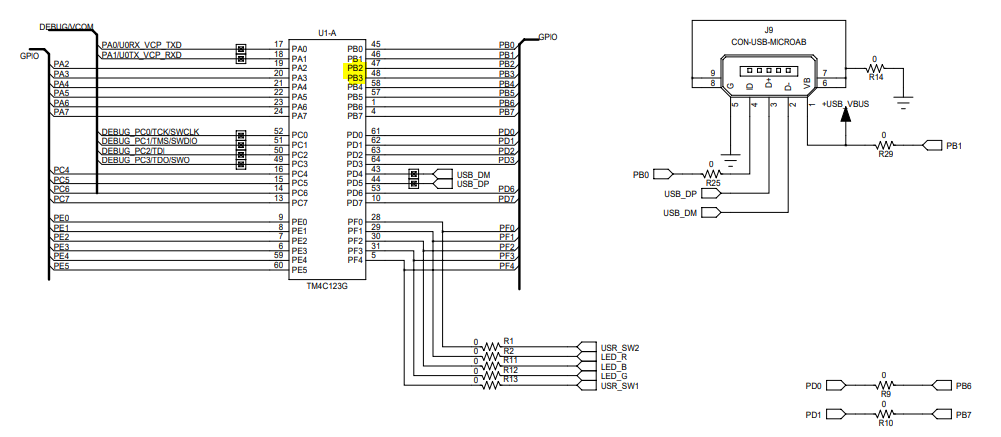
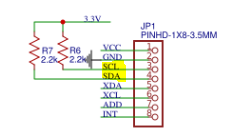


From there, we grab those values using I2C. That will synchronize the data between the two chips and organize data transfer between our devices. Here’s a box diagram as to how the I2C functions between two devices:



Here’s what my two devices look like together:



  
I used the PB2 and PB3 GPIO ports as shown above to communicate with the MPU6050. Which I connected to the SCL and SDA below.  


**IMPLEMENTATION:**

I explain my implementation of my lines of code in the code’s documentation below.

**CODE:**Code used in Task 1 and Task 2:  
//Midterm1

//

//References:

//http://www.ti.com/lit/ug/spmu371d/spmu371d.pdf

//

#include <stdbool.h>

#include <stdint.h>

#include <stdlib.h>

#include <stdio.h>

#include <stdarg.h>

#include <stdbool.h>

#include "sensorlib/i2cm\_drv.h"

#include "sensorlib/hw\_mpu6050.h"

#include "sensorlib/mpu6050.h"

#include "inc/hw\_ints.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_sysctl.h"

#include "inc/hw\_types.h"

#include "inc/hw\_i2c.h"

#include "inc/hw\_types.h"

#include "inc/hw\_gpio.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/rom.h"

#include "driverlib/rom\_map.h"

#include "driverlib/debug.h"

#include "driverlib/interrupt.h"

#include "driverlib/i2c.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "utils/uartstdio.h"

#include "driverlib/uart.h"

void InitUART(void)

{

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); //enables UART module 0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); //enables GPIO port a

GPIOPinConfigure(GPIO\_PA0\_U0RX); //PA0 as RX pin

GPIOPinConfigure(GPIO\_PA1\_U0TX); //PA1 as TX pin

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //sets UART pin type

UARTClockSourceSet(UART0\_BASE, UART\_CLOCK\_PIOSC); //sets the clock source

UARTStdioConfig(0, 115200, 16000000); //disables uartstdio, sets baud rate to 115200, uses clock

}

//

// A boolean that is set when a MPU6050 command has completed.

//

volatile bool g\_bMPU6050Done;

//

// I2C master instance

//

tI2CMInstance g\_sI2CMSimpleInst;

//

// The function that is provided by this example as a callback when MPU6050

// transactions have completed.

//

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;

}

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);

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

//clear I2C FIFOs

HWREG(I2C0\_BASE + I2C\_O\_FIFOCTL) = 8000800;

// Initialize the I2C master driver.

I2CMInit(&g\_sI2CMSimpleInst, I2C0\_BASE, INT\_I2C0, 0xff, 0xff, SysCtlClockGet());

}

// Interrupt for I2CM

void I2CM\_IntHandler(void)

{

I2CMIntHandler(&g\_sI2CMSimpleInst);

}

int main(void)

{

SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ); //use system clock

InitI2C0(); //initializes the I2C

InitUART(); //Initializes the UART

//Local Variables

float fAccel[3], fGyro[3];

float xAccel, yAccel, zAccel;

float xGyro, yGyro, zGyro;

tMPU6050 sMPU6050;

volatile uint32\_t graph\_xAccel, graph\_yAccel, graph\_zAccel;

volatile uint32\_t graph\_xGyro, graph\_yGyro, graph\_zGyro;

//

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

// has already been initialized.

//

g\_bMPU6050Done = false;

MPU6050Init(&sMPU6050, &g\_sI2CMSimpleInst, 0x68, MPU6050Callback, &sMPU6050);

while (!g\_bMPU6050Done);

//

// Configure the MPU6050 for +/- 4 g accelerometer range.

//

//Settings for the Accelerometer

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_ACCEL\_CONFIG, // Accelerometer configuration

0x00, // No need to mask

MPU6050\_ACCEL\_CONFIG\_AFS\_SEL\_4G, // Accelerometer full-scale range 4g

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

//Settings for the Gyroscope

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_GYRO\_CONFIG, // Gyroscope configuration

0x00, // No need to mask

MPU6050\_GYRO\_CONFIG\_FS\_SEL\_250, // Gyro full-scale range +/- 250 degrees/sec

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

//Turns on power for Accelerometer & Gyroscope

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_PWR\_MGMT\_1, // Power management 1 register

0x00, // No need to mask

0x00,//0x02 & MPU6050\_PWR\_MGMT\_1\_DEVICE\_RESET,

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

//Turns on power for Accelerometer & Gyroscope

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_PWR\_MGMT\_2, // Power management 2 register

0x00, // No need to mask

MPU6050\_PWR\_MGMT\_2\_LP\_WAKE\_CTRL\_1\_25, //Wake up at 1.25 Hz

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

// Loop forever reading data from the MPU6050

while (1)

{

//Wait for MPU6050

g\_bMPU6050Done = false;

MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);

while (!g\_bMPU6050Done);

//Obtain raw values of Accelerometer and Gyroscope

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

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

//Place raw values of Accelerometer and Gyroscope into a register

//Added a multiple of a 1000 to make it more visible over the terminal

xAccel = fAccel[0] \* 1000;

yAccel = fAccel[1] \* 1000;

zAccel = fAccel[2] \* 1000;

xGyro = fGyro[0] \* 1000;

yGyro = fGyro[1] \* 1000;

zGyro = fGyro[2] \* 1000;

graph\_xAccel = (int)xAccel;

graph\_yAccel = (int)yAccel;

graph\_zAccel = (int)zAccel;

graph\_xGyro = (int)xGyro;

graph\_yGyro = (int)yGyro;

graph\_zGyro = (int)zGyro;

//UART print to terminal

UARTprintf("Acceleration (x,y,z): (%d, %d, %d)\n", (int)xAccel, (int)yAccel, (int)zAccel);

UARTprintf("Gyroscope (x,y,z): (%d, %d, %d)\n", (int)xGyro, (int)yGyro, (int)zGyro);

UARTprintf("\n");

//System clock delay

SysCtlDelay(15000000);

}

}

Code used in Task 3 and Task 4:  
//Midterm1

//

//References:

//http://www.ti.com/lit/ug/spmu371d/spmu371d.pdf

//

#include <stdbool.h>

#include <stdint.h>

#include <stdlib.h>

#include <stdio.h>

#include <stdarg.h>

#include <stdbool.h>

#include "sensorlib/i2cm\_drv.h"

#include "sensorlib/hw\_mpu6050.h"

#include "sensorlib/mpu6050.h"

#include "inc/hw\_ints.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_sysctl.h"

#include "inc/hw\_types.h"

#include "inc/hw\_i2c.h"

#include "inc/hw\_types.h"

#include "inc/hw\_gpio.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/rom.h"

#include "driverlib/rom\_map.h"

#include "driverlib/debug.h"

#include "driverlib/interrupt.h"

#include "driverlib/i2c.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "utils/uartstdio.h"

#include "driverlib/uart.h"

#include "math.h"

//For Task3/Task4

#include "IQmath/IQmathLib.h"

#define ACCELEROMETER\_SENSITIVITY 8192.0

#define GYROSCOPE\_SENSITIVITY 65.536

#define SAMPLE\_RATE 0.01

#define M\_PI 3.14159265359

#define dt 0.01// 10 ms sample rate!

void ComplementaryFilter(\_iq16 accData[3], \_iq16 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;

}

}

void InitUART(void)

{

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); //enables UART module 0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); //enables GPIO port a

GPIOPinConfigure(GPIO\_PA0\_U0RX); //PA0 as RX pin

GPIOPinConfigure(GPIO\_PA1\_U0TX); //PA1 as TX pin

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); //sets UART pin type

UARTClockSourceSet(UART0\_BASE, UART\_CLOCK\_PIOSC); //sets the clock source

UARTStdioConfig(0, 115200, 16000000); //disables uartstdio, sets baud rate to 115200, uses clock

}

//

// A boolean that is set when a MPU6050 command has completed.

//

volatile bool g\_bMPU6050Done;

//

// I2C master instance

//

tI2CMInstance g\_sI2CMSimpleInst;

//

// The function that is provided by this example as a callback when MPU6050

// transactions have completed.

//

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;

}

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);

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

//clear I2C FIFOs

HWREG(I2C0\_BASE + I2C\_O\_FIFOCTL) = 8000800;

// Initialize the I2C master driver.

I2CMInit(&g\_sI2CMSimpleInst, I2C0\_BASE, INT\_I2C0, 0xff, 0xff, SysCtlClockGet());

}

// Interrupt for I2CM

void I2CM\_IntHandler(void)

{

I2CMIntHandler(&g\_sI2CMSimpleInst);

}

int main(void)

{

SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ); //use system clock

InitI2C0(); //initializes the I2C

InitUART(); //Initializes the UART

//Local Variables

float fAccel[3], fGyro[3];

float xAccel, yAccel, zAccel;

float xGyro, yGyro, zGyro;

float pitch, roll;

\_iq16 QAccel[3];

\_iq16 QGyro[3];

tMPU6050 sMPU6050;

//

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

// has already been initialized.

//

g\_bMPU6050Done = false;

MPU6050Init(&sMPU6050, &g\_sI2CMSimpleInst, 0x68, MPU6050Callback, &sMPU6050);

while (!g\_bMPU6050Done);

//

// Configure the MPU6050 for +/- 4 g accelerometer range.

//

//Settings for the Accelerometer

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_ACCEL\_CONFIG, // Accelerometer configuration

0x00, // No need to mask

MPU6050\_ACCEL\_CONFIG\_AFS\_SEL\_4G, // Accelerometer full-scale range 4g

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

//Settings for the Gyroscope

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_GYRO\_CONFIG, // Gyroscope configuration

0x00, // No need to mask

MPU6050\_GYRO\_CONFIG\_FS\_SEL\_250, // Gyro full-scale range +/- 250 degrees/sec

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

//Turns on power for Accelerometer & Gyroscope

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_PWR\_MGMT\_1, // Power management 1 register

0x00, // No need to mask

0x00,//0x02 & MPU6050\_PWR\_MGMT\_1\_DEVICE\_RESET,

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

//Turns on power for Accelerometer & Gyroscope

g\_bMPU6050Done = false;

MPU6050ReadModifyWrite(&sMPU6050,

MPU6050\_O\_PWR\_MGMT\_2, // Power management 2 register

0x00, // No need to mask

MPU6050\_PWR\_MGMT\_2\_LP\_WAKE\_CTRL\_1\_25, //Wake up at 1.25 Hz

MPU6050Callback,

&sMPU6050);

while (!g\_bMPU6050Done);

// Loop forever reading data from the MPU6050

while (1)

{

//Wait for MPU6050

g\_bMPU6050Done = false;

MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);

while (!g\_bMPU6050Done);

//Obtain raw values of Accelerometer and Gyroscope

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

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

//Place raw values of Accelerometer and Gyroscope into a register

//Added a multiple of a 1000 to make it more visible over the terminal

xAccel = fAccel[0] \* 1000;

yAccel = fAccel[1] \* 1000;

zAccel = fAccel[2] \* 1000;

xGyro = fGyro[0] \* 1000;

yGyro = fGyro[1] \* 1000;

zGyro = fGyro[2] \* 1000;

//UART print to terminal

UARTprintf("Acceleration (x,y,z): (%d, %d, %d)\n", (int)xAccel, (int)yAccel, (int)zAccel);

UARTprintf("Gyroscope (x,y,z): (%d, %d, %d)\n", (int)xGyro, (int)yGyro, (int)zGyro);

//IQMath Conversion

QAccel[0] = \_IQ16(fAccel[0]);

QAccel[1] = \_IQ16(fAccel[1]);

QAccel[2] = \_IQ16(fAccel[2]);

QGyro[0] = \_IQ16(fGyro[0]);

QGyro[1] = \_IQ16(fGyro[1]);

QGyro[2] = \_IQ16(fGyro[2]);

ComplementaryFilter(QAccel, QGyro, &pitch, &roll);

UARTprintf("Pitch: %d\nRoll: %d\n\n", (int)(pitch\*10), (int)(roll\*10)); //Scaled it to make it easier to view on the terminal

//System clock delay

SysCtlDelay(15000000);

}

}

Name: Chris Barr