CPE 403 ADV EMB SYS DES F 2019

GITHUB LINK: https://github.com/sotoi2/Class3.0.4 Ivan Soto

TITLE: Midterm for CPE 403

GOAL: Interface the given MPU6050 IMU using I2C protocol to TivaC. Print all accelerometer and gyro values on to the serial terminal.

Make bullet points of the project goal(s).

- Solder and learn how to wire the Tiva C and MPU 6050
- Decide to use the sensorlib library
- Discover how to print required values to serial monitor and graphs.

DELIVERABLES:

What is the intended project deliverables? What was completed?

For task 1 and 2, we were to display the accelerometer and gyroscope reading in the three dimensions to a serial monitor and graph tool respectively. The task was finished completely, with reading being shown in the youtube videos linked on the github.

COMPONENTS:

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

Tiva C Series TM4C123G LaunchPad Evaluation Kit- Microcontroller for reading the data from the MPU

Breadboard and wires- For making the appropriate connections

MPU6050- Gives us accelerometer and gyroscope readings in three dimensions that are needed for both UART and graph display.

CCS 7.4 – The software needed to program into our microcontroller and receive the MPU data.

Limitations: Faulty MPU units, Using correct CCS libraries, faulty Tiva C microcontrollers.

IMPLEMENTATION:

Step implemented in the code - for example initialization of I2C, UART, start reading one set of data, print - explain each subroutine.

-Void I2CSimpleIntHandler

A handler for calling the I2CMIntHandler for use, necessary when used as arguments.

-void MPU6050CallBack

Is able to check for status errors.

-void InitI2C

This part of the code is what initializes the I2C modules and sets the appropriate settings for communicating with the MPU6050.

- Void MPU6050Example

  This part of the code, using the sensorlib libraryis what is able initialize the MPU, read the data from it, and also print it.
- Int main

  Calls all other modules for use. Sets the clock frequency as well.

CODE:

```c
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/uart.h"
#include "driverlib/i2c.h"
#include "sensorlib/i2cm_drv.h"
#include "driverlib/pin_map.h"
#include "inc/hw_memmap.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "inc/hw_ints.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include <string.h>
#include "utils/uartstdio.h"



volatile bool g_bMPU6050Done;


    tI2CMInstance g_sI2CMSimpleInst;

    void I2CMSimpleIntHandler(void)
    {

        I2CMIntHandler(&g_sI2CMSimpleInst);
    }


    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_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff,
SysCtlClockGet());

}

void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{

    if (ui8Status != I2CM_STATUS_SUCCESS)
    {

    }

    g_bMPU6050Done = true;
}




//
// The MPU6050 example.
//
void MPU6050Example(void)
{
    float fAccel[3], fGyro[3];
    float gx, gy, gz;
    float ax, ay, az;

    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.
```

```
    //
    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. Typically, this process
    // would be done in the background, but for the purposes of this example,
    // it is shown in an infinite loop.
    //
    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.
```

```
//


SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    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);
    //These variables were used because the original values were too small
    //to read easily.



    ax = fAccel[0] * 15000;
    ay = fAccel[1] * 15000;
    az = fAccel[2] * 15000;

    gx = fGyro[0] * 15000;
    gy = fGyro[1] * 15000;
    gz = fGyro[2] * 15000;

    // This is where the UART will be configured and enabled.
    // the clock is enabled to the gpioa and the uart
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        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);



        // printing the necessary values

                    UARTprintf("Accelerometer X: %d", (int)ax);

                    UARTprintf(" Acceloerometer Y: %d", (int)ay);

                    UARTprintf(" Accelerometer Z: %d\n", (int)az);




                     UARTprintf("Gyro X: %d", (int)gx);

                     UARTprintf(" Gyro Y: %d", (int)gy);

                     UARTprintf(" Gyro Z: %d\n", (int)gz);
```

```
// a delay to make the data easier to aread
            SysCtlDelay(2500000);

    }
}

int main()
{
    //clock setting
    SysCtlClockSet(SYSCTL_SYSDIV_1|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    //call the
    InitI2C0();
    MPU6050Example();
    return(0);
}
```


TITLE: Midterm for CPE 403

GOAL: 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. Also use a graph tool

Make bullet points of the project goal(s).

- Solder and learn how to wire the Tiva C and MPU 6050

- Decide to use the sensorlib library
- Learn to use IQMath and its required libraries
- Discover how to print required values to serial monitor and graphs.

DELIVERABLES:

What is the intended project deliverables? What was completed?

The project delivered gyroscope and accelerometer values, along with pitch and roll from a filter onto the serial monitor and a graph. The roll and pitch were able to be displayed and showed off.

The project intends to

For task 3 and 4, we were to display the accelerometer, gyroscope, as well as implement a complimentary filter on the code that we had previously done. The filter was able to be called, then supply the pitch and roll. The implementation was to involve IQMath.

COMPONENTS:

Explain the main characteristics, interface, and limitation of the components used in the design,

including the registered used and what was initialized? Why?

Tiva C Series TM4C123G LaunchPad Evaluation Kit- Microcontroller for reading the data from the MPU

Breadboard and wires- For making the appropriate connections

MPU6050- Gives us accelerometer and gyroscope readings in three dimensions that are needed for both UART and graph display.

IMPLEMENTATION:

Step implemented in the code - for example initialization of I2C, UART, start reading one set of data, print - explain each subroutine.

-Void I2CSimpleIntHandler

A handler for calling the I2CMIntHandler for use, necessary when used as arguments.

-void InitI2C

This part of the code is what initializes the I2C modules and sets the appropriate settings for communicating with the MPU6050.

- Void MPU6050Example

  This part of the code, using the sensorlib libraryis what is able initialize the MPU, read the data from it, and also print it.
- Int main

  Calls all other modules for use. Sets the clock frequency as well.

-void ComplementaryFilter

The filter to provide the pitch and roll values, one could use IQMath in this section.

-void MPU6050CallBack

Is able to check for status errors.

CODE:

```c
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/uart.h"
#include "driverlib/i2c.h"
#include "sensorlib/i2cm_drv.h"
#include "driverlib/pin_map.h"
#include "inc/hw_memmap.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "inc/hw_ints.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "driverlib/rom_map.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include <string.h>
#include "utils/uartstdio.h"
#include "IQmath/IQmathLib.h"
#include <math.h>



#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536

#define M_PI 3.14159265359

#define dt 0.01 // 10ms samplej rate!


volatile bool g_bMPU6050Done;


    tI2CMInstance g_sI2CMSimpleInst;

    void I2CMSimpleIntHandler(void)
    {

        I2CMIntHandler(&g_sI2CMSimpleInst);
    }




    void ComplementaryFilter(short accData[3], short gyrData[3], float *pitch, float
*roll)
```

```
    {
     float pitchAcc, rollAcc;
     float n1;
     float n2;




     // 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;

     n1 = _IQ21toF(pitchAcc);
        n2 = _IQ21toF(rollAcc);
     }
    }

    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_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff,
SysCtlClockGet());

}

void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{

    if (ui8Status != I2CM_STATUS_SUCCESS)
    {

    }

    g_bMPU6050Done = true;
}




//
// The MPU6050 example.
//
void MPU6050Example(void)
{
    float fAccel[3], fGyro[3];
    float gx, gy, gz;
    float ax, ay, az;

    float pitch;
    float roll;

    float temp1;
    float temp2;

    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.
    //
    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. Typically, this process
    // would be done in the background, but for the purposes of this example,
    // it is shown in an infinite loop.
    //
    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.
        //


        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

```
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

            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);
            //These variables were used because the original values were too small
            //to read easily.



            ax = fAccel[0] * 15000;
            ay = fAccel[1] * 15000;
            az = fAccel[2] * 15000;

            gx = fGyro[0] * 15000;
            gy = fGyro[1] * 15000;
            gz = fGyro[2] * 15000;

            // This is where the UART will be configured and enabled.
            // the clock is enabled to the gpioa and the uart
                SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
                SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
                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);



                // printing the necessary values

                           UARTprintf("Accelerometer X: %d", (int)ax);

                           UARTprintf(" Acceloerometer Y: %d", (int)ay);

                           UARTprintf(" Accelerometer Z: %d\n", (int)az);




                            UARTprintf("Gyro X: %d", (int)gx);

                            UARTprintf(" Gyro Y: %d", (int)gy);

                            UARTprintf(" Gyro Z: %d\n", (int)gz);


// a delay to make the data easier to aread
```

```
            SysCtlDelay(2500000);

            short fAccelShort[3];
            fAccelShort[0] = (short)(fAccel[0]);
            fAccelShort[1] = (short)(fAccel[1]);
            fAccelShort[2] = (short)(fAccel[2]);

            short fGyroShort[3];
            fGyroShort[0] = (short)(fGyro[0]);
            fGyroShort[1] = (short)(fGyro[1]);
            fGyroShort[2] = (short)(fGyro[2]);



            ComplementaryFilter(&fAccelShort[0],&fGyroShort[0], &pitch, &roll );
            temp1 += (float)fGyroShort[0]/6553.6;
            temp2 = atan2f((float)fAccelShort[1], (float)fAccelShort[2])*180/M_PI;

            UARTprintf(" Pitch: %d", (int)temp1);

           UARTprintf(" Roll: %d\n", (int)temp2);


    }
}

int main()
{
    //clock setting
    SysCtlClockSet(SYSCTL_SYSDIV_1|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
    //call the
    InitI2C0();
    MPU6050Example();

    return(0);
}
```