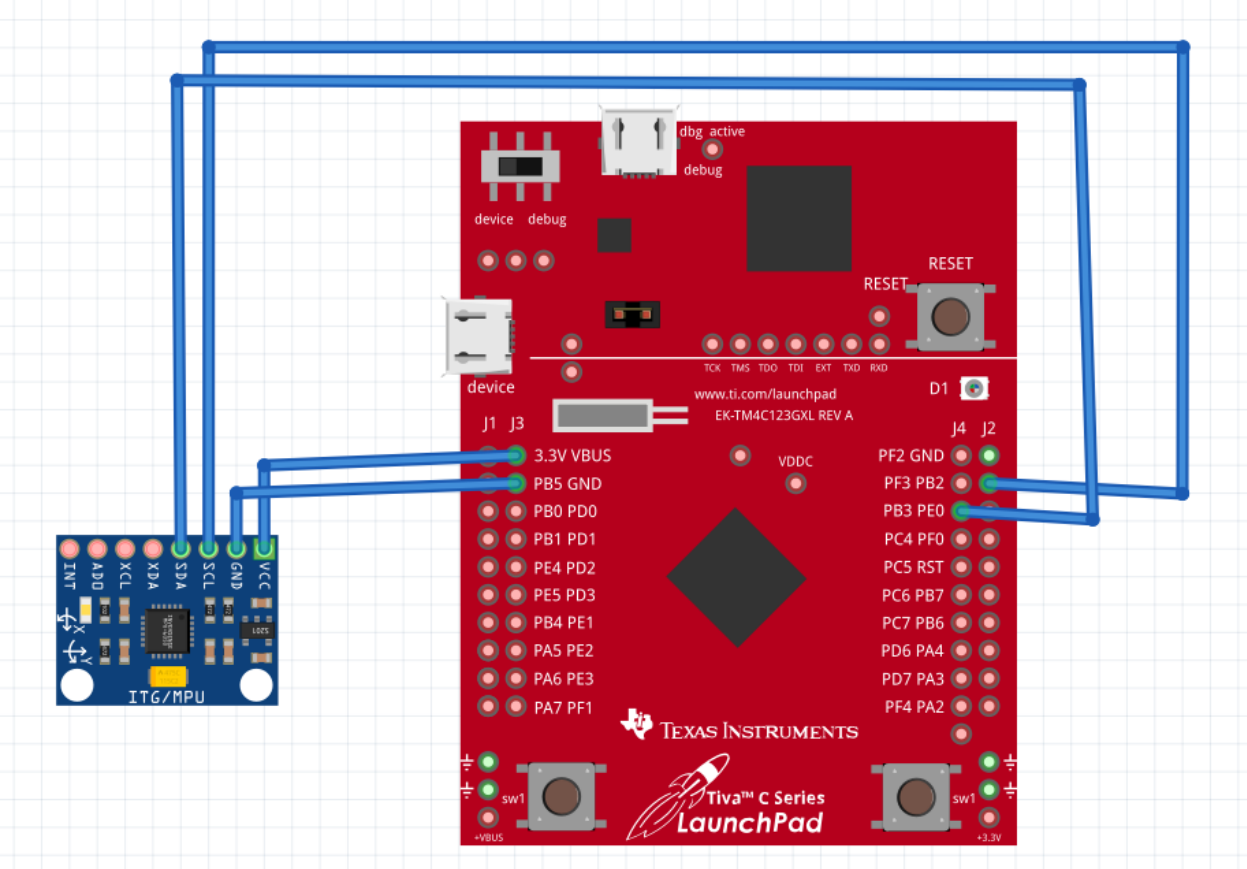
Nathan Hanuscin
Github root directory: https://github.com/nhanuscin/HappyFunStuff

Date Submitted: **11/14/2019**

Task 01:

Youtube Link: https://www.youtube.com/watch?v=TiD0jiqBB-o

Modified Schematic (if applicable):



Modified Code:

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
#include "math.h"


/* ------------------------------------------------- */
//-------- MPU6050 Register Addresses -----------------//
#define MPU_ADDRESS     0x68

#define WHO_AM_I        0x75
#define PWR_MGMT_1      0x6B
#define SMPRT_DIV       0x19
#define CONFIG          0x1A
#define GYRO_CONFIG     0x1B
#define ACC_CONFIG      0x1C
#define INT_PIN_CFG     0x37
#define INT_ENABLE      0x38

#define ACCEL_XOUT_H    0x3B
#define ACCEL_XOUT_L    0x3C
#define ACCEL_YOUT_H    0x3D
#define ACCEL_YOUT_L    0x3E
#define ACCEL_ZOUT_H    0x3F
#define ACCEL_ZOUT_L    0x40

#define GYRO_XOUT_H     0x43
#define GYRO_XOUT_L     0x44
#define GYRO_YOUT_H     0x45
#define GYRO_YOUT_L     0x46
#define GYRO_ZOUT_H     0x47
#define GYRO_ZOUT_L     0x48

//---------------------------------------//

/* ---------------------------------------*/
// ------- Variable Declarations --------//

int WhoAmI, RegReset;

int accXout_L, accXout_H, accXout;
int accYout_L, accYout_H, accYout;
int accZout_L, accZout_H, accZout;
float accXos[3] = {0,0,0};
float accYos[3]= {0,0,0};

int gyroXout_L, gyroXout_H, gyroXout;
int gyroYout_L, gyroYout_H, gyroYout;
int gyroZout_L, gyroZout_H, gyroZout;
float gyroXos[3] = {0,0,0};
float gyroYos[3] = {0,0,0};
float gyroZos;
float fiXos[3]={0,0,0};
float fiYos[3]= {0,0,0};

float kotXos;
 //-----------------------------------//
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
#define tip 0.001
#define tau 1.6
#define ett 0.999375195
int data_ready;
float Itemp;

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536
#define M_PI 3.14159265359
#define dt 0.01 // 10 ms sample rate!


void initMPU6050()
{
    writeI2C(MPU_ADDRESS, PWR_MGMT_1, (1 << 3) || 0x03 );       // power managment
setup, temp sensor OFF, sleep mode OFF ...
    writeI2C(MPU_ADDRESS, SMPRT_DIV, 0x01);                     // sample rate 1kHz
    writeI2C(MPU_ADDRESS, CONFIG, 0x03);                        // disable FSYNC, 41
Hz gyro filtering, 1 kHz sampling      ??????????
    writeI2C(MPU_ADDRESS, GYRO_CONFIG, (3 << 3));               // gyro full scale
range --> 2000 deg/s (3 << 3)
    writeI2C(MPU_ADDRESS, ACC_CONFIG, (2 << 3));                // acc full scale
range  --> 8g (2 << 3)

    //writeI2C(MPU_ADDRESS, INT_PIN_CFG, 0x30);   // Configure INT pin or 0011 0000
??? 0x30
    //writeI2C(MPU_ADDRESS, INT_ENABLE, 0x01);    // Enable interrupt DATA READY bit


    //SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    //SysCtlDelay(3);
    //GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_2); // Set as input
//  GPIOIntTypeSet(GPIO_PORTE_BASE, GPIO_PIN_2, GPIO_RISING_EDGE);
//  GPIOIntRegister(GPIO_PORTE_BASE, readMPU);
//  IntPrioritySet(INT_GPIOE, 0);
//  GPIOIntEnable(GPIO_PORTE_BASE, GPIO_INT_PIN_2);

}
//Configures the UART to run at 115200 baud rate
void ConfigureUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);    //enables UART module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    //enables GPIO port a

    GPIOPinConfigure(GPIO_PA1_U0TX);    //configures PA1 as TX pin
    GPIOPinConfigure(GPIO_PA0_U0RX);    //configures PA0 as RX pin
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);  //sets the UART pin
type

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);   //sets the clock source
    UARTStdioConfig(0, 115200, 16000000);    //enables UARTstdio baud rate, clock,
and which UART to use
}
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
//Configure/initialize the I2C0
void I2C0_Init(void)
{

    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);    //enables I2C0
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);      //reset module
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);   //enable PORTB as peripheral
    //Configure the pin muxing for I2C0 functions on port B2 and B3
    GPIOPinConfigure (GPIO_PB3_I2C0SDA);
    GPIOPinConfigure (GPIO_PB2_I2C0SCL);
    //Select the I2C function for these pins
    GPIOPinTypeI2CSCL (GPIO_PORTB_BASE, GPIO_PIN_2);    //set I2C PB2 as SCLK
    GPIOPinTypeI2C (GPIO_PORTB_BASE, GPIO_PIN_3);   //set I2C PB3 as SDA

    I2CMasterInitExpClk (I2C0_BASE, SysCtlClockGet(), true);   //Set the clock of the
I2C to ensure proper connection
    //while (I2CMasterBusy (I2C0_BASE));  //wait while the master SDA is busy

}
void readI2C(uint8_t slave_addr, uint8_t reg, int *data)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
    while(I2CMasterBusy(I2C0_BASE));
    *data = I2CMasterDataGet(I2C0_BASE);
}

// Sends 1 byte over i2c
void writeI2C(uint8_t slave_addr, uint8_t reg, uint8_t data)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterDataPut(I2C0_BASE, data);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
    while(I2CMasterBusy(I2C0_BASE));
}

void readMPU()
{
    readI2C(MPU_ADDRESS, ACCEL_XOUT_H, &accXout_H);
    readI2C(MPU_ADDRESS, ACCEL_XOUT_L, &accXout_L);
    readI2C(MPU_ADDRESS, ACCEL_YOUT_H, &accYout_H);
    readI2C(MPU_ADDRESS, ACCEL_YOUT_L, &accYout_L);
    readI2C(MPU_ADDRESS, ACCEL_ZOUT_H, &accZout_H);
    readI2C(MPU_ADDRESS, ACCEL_ZOUT_L, &accZout_L);

    readI2C(MPU_ADDRESS, GYRO_XOUT_H, &gyroXout_H);
    readI2C(MPU_ADDRESS, GYRO_XOUT_L, &gyroXout_L);
    readI2C(MPU_ADDRESS, GYRO_YOUT_H, &gyroYout_H);
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    readI2C(MPU_ADDRESS, GYRO_YOUT_L, &gyroYout_L);
    readI2C(MPU_ADDRESS, GYRO_ZOUT_H, &gyroZout_H);
    readI2C(MPU_ADDRESS, GYRO_ZOUT_L, &gyroZout_L);

    accXout = ((accXout_H << 8) | accXout_L);
    accYout = ((accYout_H << 8) | accYout_L);
    accZout = ((accZout_H << 8) | accZout_L);

    gyroXout = ((gyroXout_H << 8) | gyroXout_L);
    gyroYout = ((gyroYout_H << 8) | gyroYout_L);
    gyroZout = ((gyroZout_H << 8) | gyroZout_L);

    if(accXout&0x8000) accXout|=0xFFFF0000;
    if(accYout&0x8000) accYout|=0xFFFF0000;
    if(accZout&0x8000) accZout|=0xFFFF0000;

    if(gyroXout&0x8000) gyroXout|=0xFFFF0000;
    if(gyroYout&0x8000) gyroYout|=0xFFFF0000;
    if(gyroZout&0x8000) gyroZout|=0xFFFF0000;

    accXos[1]=accXos[0];
    accYos[1]=accYos[0];
    gyroXos[1]=gyroXos[0];
    gyroYos[1]=gyroYos[0];
    fiXos[2]=fiXos[1];
    fiXos[1]=fiXos[0];
    fiYos[2]=fiYos[1];
    fiYos[1]=fiYos[0];

    float a = 2*ett,
          b = -ett*ett,
          c = tip*ett/tau-ett+1,
          d = ett*ett-tip*ett/tau-ett,
          e = tip*ett;

    accXos[0] = -atan2(accXout, accZout);
    accYos[0] = -atan2(accYout, accZout);

    gyroXos[0] = (float)gyroYout * 0.00106422515365507901031932363932f;     //
pi/(180*16.4)
    gyroYos[0] = -(float)gyroXout * 0.00106422515365507901031932363932f;
    gyroZos = (float)gyroZout * 0.06097478f; //degree / second

    fiXos[0] = a*fiXos[1] + b*fiXos[2] + c*accXos[0] + d*accXos[1] + e*(gyroXos[0]-
gyroXos[1]);
    fiYos[0] = a*fiYos[1] + b*fiYos[2] + c*accYos[0] + d*accYos[1] + e*(gyroYos[0]-
gyroYos[1]);

}

int main (void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
//set the main clock to run at 40MHz
    ConfigureUART();
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    UARTprintf("Uart Configured\r\n");
    SysCtlDelay(13500000);  // ~1sec delay
    I2C0_Init();
    UARTprintf("IC2 initialized\r\n");
    SysCtlDelay(13500000);  // ~1sec delay
    initMPU6050();
    UARTprintf("MPU6050 initialized\r\n");
    UARTprintf("Starting data collection\r\n");
    SysCtlDelay(13500000);  // ~1sec delay

    while(1)
    {
        readMPU();
        UARTprintf("AX: %d\r\n", accXout);
        UARTprintf("AY: %d\r\n", accYout);
        UARTprintf("AZ: %d\r\n", accZout);
        UARTprintf("GX: %d\r\n", gyroXout);
        UARTprintf("GY: %d\r\n", gyroYout);
        UARTprintf("GZ: %d\r\n", gyroZout);
        SysCtlDelay(13500000);  // ~1sec delay

    }
}

--------------------------------------------------------------------------------
```

Nathan Hanuscin
Github root directory: https://github.com/nhanuscin/HappyFunStuff

Task 02:

Youtube Link: https://www.youtube.com/watch?v=6Bf9JkX-s5Q

Modified Schematic (if applicable): Same as Task 1

Modified Code:
Same code as Task 01

Task 03:

Youtube Link: https://www.youtube.com/watch?v=BBN4dFjWfVM

Modified Schematic (if applicable): Same as Task 1

Modified Code:

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include "math.h"


/* ------------------------------------------------- */
//-------- MPU6050 Register Addresses -----------------//
#define MPU_ADDRESS     0x68

#define WHO_AM_I        0x75
#define PWR_MGMT_1      0x6B
#define SMPRT_DIV       0x19
#define CONFIG          0x1A
#define GYRO_CONFIG     0x1B
#define ACC_CONFIG      0x1C
#define INT_PIN_CFG     0x37
#define INT_ENABLE      0x38

#define ACCEL_XOUT_H    0x3B
#define ACCEL_XOUT_L    0x3C
#define ACCEL_YOUT_H    0x3D
#define ACCEL_YOUT_L    0x3E
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
#define ACCEL_ZOUT_H    0x3F
#define ACCEL_ZOUT_L    0x40

#define GYRO_XOUT_H     0x43
#define GYRO_XOUT_L     0x44
#define GYRO_YOUT_H     0x45
#define GYRO_YOUT_L     0x46
#define GYRO_ZOUT_H     0x47
#define GYRO_ZOUT_L     0x48

//--------------------------------------//

/* -------------------------------------*/
// ------- Variable Declarations --------//

int WhoAmI, RegReset;
int accXout_L, accXout_H, accXout;
int accYout_L, accYout_H, accYout;
int accZout_L, accZout_H, accZout;
int pitchp, rollp;      //used for printing floats
int pitchsign, rollsign;    //used for printing negatives
//int fractional;
float accXos[3] = {0,0,0};
float accYos[3]= {0,0,0};

int gyroXout_L, gyroXout_H, gyroXout;
int gyroYout_L, gyroYout_H, gyroYout;
int gyroZout_L, gyroZout_H, gyroZout;
float gyroXos[3] = {0,0,0};
float gyroYos[3] = {0,0,0};
float gyroZos;
float fiXos[3]={0,0,0};
float fiYos[3]= {0,0,0};

float pitch = 0;
float roll = 0;

float kotXos;
 //-------------------------------------//


#define tip 0.001
#define tau 1.6
#define ett 0.999375195
int data_ready;
float Itemp;

#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536
#define M_PI 3.14159265359
#define dt 0.01 // 10 ms sample rate!


void initMPU6050()
{
```

```
    writeI2C(MPU_ADDRESS, PWR_MGMT_1, (1 << 3) || 0x03 );       // power managment
setup, temp sensor OFF, sleep mode OFF ...
    writeI2C(MPU_ADDRESS, SMPRT_DIV, 0x01);                     // sample rate 1kHz
    writeI2C(MPU_ADDRESS, CONFIG, 0x03);                        // disable FSYNC, 41
Hz gyro filtering, 1 kHz sampling      ??????????
    writeI2C(MPU_ADDRESS, GYRO_CONFIG, (3 << 3));               // gyro full scale
range --> 2000 deg/s (3 << 3)
    writeI2C(MPU_ADDRESS, ACC_CONFIG, (2 << 3));                // acc full scale
range  --> 8g (2 << 3)

    //writeI2C(MPU_ADDRESS, INT_PIN_CFG, 0x30);   // Configure INT pin or 0011 0000
??? 0x30
    //writeI2C(MPU_ADDRESS, INT_ENABLE, 0x01);    // Enable interrupt DATA READY bit


    //SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    //SysCtlDelay(3);
    //GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_2); // Set as input
//  GPIOIntTypeSet(GPIO_PORTE_BASE, GPIO_PIN_2, GPIO_RISING_EDGE);
//  GPIOIntRegister(GPIO_PORTE_BASE, readMPU);
//  IntPrioritySet(INT_GPIOE, 0);
//  GPIOIntEnable(GPIO_PORTE_BASE, GPIO_INT_PIN_2);

}
//Configures the UART to run at 115200 baud rate
void ConfigureUART(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);    //enables UART module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    //enables GPIO port a

    GPIOPinConfigure(GPIO_PA1_U0TX);    //configures PA1 as TX pin
    GPIOPinConfigure(GPIO_PA0_U0RX);    //configures PA0 as RX pin
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);  //sets the UART pin
type

    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);   //sets the clock source
    UARTStdioConfig(0, 115200, 16000000);    //enables UARTstdio baud rate, clock,
and which UART to use
}

//Configure/initialize the I2C0
void I2C0_Init(void)
{

    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);    //enables I2C0
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);      //reset module
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);   //enable PORTB as peripheral
    //Configure the pin muxing for I2C0 functions on port B2 and B3
    GPIOPinConfigure (GPIO_PB3_I2C0SDA);
    GPIOPinConfigure (GPIO_PB2_I2C0SCL);
    //Select the I2C function for these pins
    GPIOPinTypeI2CSCL (GPIO_PORTB_BASE, GPIO_PIN_2);    //set I2C PB2 as SCLK
    GPIOPinTypeI2C (GPIO_PORTB_BASE, GPIO_PIN_3);   //set I2C PB3 as SDA
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    I2CMasterInitExpClk (I2C0_BASE, SysCtlClockGet(), true);   //Set the clock of the
I2C to ensure proper connection
    //while (I2CMasterBusy (I2C0_BASE));  //wait while the master SDA is busy

}
void readI2C(uint8_t slave_addr, uint8_t reg, int *data)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
    while(I2CMasterBusy(I2C0_BASE));
    *data = I2CMasterDataGet(I2C0_BASE);
}

// Sends 1 byte over i2c
void writeI2C(uint8_t slave_addr, uint8_t reg, uint8_t data)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterDataPut(I2C0_BASE, data);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
    while(I2CMasterBusy(I2C0_BASE));
}

void readMPU()
{
    readI2C(MPU_ADDRESS, ACCEL_XOUT_H, &accXout_H);
    readI2C(MPU_ADDRESS, ACCEL_XOUT_L, &accXout_L);
    readI2C(MPU_ADDRESS, ACCEL_YOUT_H, &accYout_H);
    readI2C(MPU_ADDRESS, ACCEL_YOUT_L, &accYout_L);
    readI2C(MPU_ADDRESS, ACCEL_ZOUT_H, &accZout_H);
    readI2C(MPU_ADDRESS, ACCEL_ZOUT_L, &accZout_L);

    readI2C(MPU_ADDRESS, GYRO_XOUT_H, &gyroXout_H);
    readI2C(MPU_ADDRESS, GYRO_XOUT_L, &gyroXout_L);
    readI2C(MPU_ADDRESS, GYRO_YOUT_H, &gyroYout_H);
    readI2C(MPU_ADDRESS, GYRO_YOUT_L, &gyroYout_L);
    readI2C(MPU_ADDRESS, GYRO_ZOUT_H, &gyroZout_H);
    readI2C(MPU_ADDRESS, GYRO_ZOUT_L, &gyroZout_L);

    accXout = ((accXout_H << 8) | accXout_L);
    accYout = ((accYout_H << 8) | accYout_L);
    accZout = ((accZout_H << 8) | accZout_L);

    gyroXout = ((gyroXout_H << 8) | gyroXout_L);
    gyroYout = ((gyroYout_H << 8) | gyroYout_L);
    gyroZout = ((gyroZout_H << 8) | gyroZout_L);

    if(accXout&0x8000) accXout|=0xFFFF0000;
    if(accYout&0x8000) accYout|=0xFFFF0000;
```

```
    if(accZout&0x8000) accZout|=0xFFFF0000;

    if(gyroXout&0x8000) gyroXout|=0xFFFF0000;
    if(gyroYout&0x8000) gyroYout|=0xFFFF0000;
    if(gyroZout&0x8000) gyroZout|=0xFFFF0000;

    accXos[1]=accXos[0];
    accYos[1]=accYos[0];
    gyroXos[1]=gyroXos[0];
    gyroYos[1]=gyroYos[0];
    fiXos[2]=fiXos[1];
    fiXos[1]=fiXos[0];
    fiYos[2]=fiYos[1];
    fiYos[1]=fiYos[0];

    float a = 2*ett,
          b = -ett*ett,
          c = tip*ett/tau-ett+1,
          d = ett*ett-tip*ett/tau-ett,
          e = tip*ett;

    accXos[0] = -atan2(accXout, accZout);
    accYos[0] = -atan2(accYout, accZout);

    gyroXos[0] = (float)gyroYout * 0.00106422515365507901031932363932f;     //
pi/(180*16.4)
    gyroYos[0] = -(float)gyroXout * 0.00106422515365507901031932363932f;
    gyroZos = (float)gyroZout * 0.06097478f; //degree / second

    fiXos[0] = a*fiXos[1] + b*fiXos[2] + c*accXos[0] + d*accXos[1] + e*(gyroXos[0]-
gyroXos[1]);
    fiYos[0] = a*fiYos[1] + b*fiYos[2] + c*accYos[0] + d*accYos[1] + e*(gyroYos[0]-
gyroYos[1]);

}

void ComplementaryFilter(void)
{
    float pitchAcc, rollAcc;
    // Integrate the gyroscope data -> int(angularSpeed) = angle
    // Angle around the X-axis
    pitch += ((float)gyroXout / GYROSCOPE_SENSITIVITY) * dt;
    // Angle around the Y-axis
    roll -= ((float)gyroYout / GYROSCOPE_SENSITIVITY) * dt;
    // Compensate for drift with accelerometer data
    // Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192
    int forceMagnitudeApprox = abs(accXout) + abs(accYout) + abs(accZout);
    if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)
    {
        // Turning around the X axis results in a vector on the Y-axis
        pitchAcc = atan2f((float)accYout, (float)accZout) * 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)accXout, (float)accZout) * 180 / M_PI;
        roll = roll * 0.98 + rollAcc * 0.02;
```

```
    }
    rollp = roll * 10000;
    pitchp = pitch * 10000;

    if(rollp < 0)
    {
        rollp = rollp *-1;
        rollsign = 1;
    }
    else
        rollsign = 0;
    if(pitchp < 0)
    {
        pitchp = pitchp *-1;
        pitchsign = 1;
    }
    else
        pitchsign = 0;


}
int main (void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
//set the main clock to run at 40MHz
    ConfigureUART();
    UARTprintf("Uart Configured\r\n");
    SysCtlDelay(13500000);  // ~1sec delay
    I2C0_Init();
    UARTprintf("IC2 initialized\r\n");
    SysCtlDelay(13500000);  // ~1sec delay
    initMPU6050();
    UARTprintf("MPU6050 initialized\r\n");
    UARTprintf("Starting data collection\r\n");
    SysCtlDelay(13500000);  // ~1sec delay

    while(1)
    {
        readMPU();
        ComplementaryFilter();
        UARTprintf("AX: %d\r\n", accXout);
        UARTprintf("AY: %d\r\n", accYout);
        UARTprintf("AZ: %d\r\n", accZout);
        UARTprintf("GX: %d\r\n", gyroXout);
        UARTprintf("GY: %d\r\n", gyroYout);
        UARTprintf("GZ: %d\r\n", gyroZout);
        if(rollsign == 1)
        {
            if(rollp > 1000 && rollp < 9999)
                UARTprintf("Roll: -0.%d\r\n", rollp);
            else if(rollp < 1000 && rollp > 100)
                UARTprintf("Roll: -0.0%d\r\n", rollp);
            else if(rollp < 100)
                UARTprintf("Roll: -0.00%d\r\n", rollp);
            else
```

```
                UARTprintf("Roll: -%d.%d\r\n", rollp/10000, (((roll*-1.0) -
rollp/10000)*1000));
        }
        else
        {
            if(rollp > 1000 && rollp < 9999)
                UARTprintf("Roll: 0.%d\r\n", rollp);
            else if(rollp < 1000 && rollp > 100)
                UARTprintf("Roll: 0.0%d\r\n", rollp);
            else if(rollp < 100)
                UARTprintf("Roll: 0.00%d\r\n", rollp);
            else
                UARTprintf("Roll: %d.%d\r\n", rollp/10000, (roll -
(rollp/10000))*1000);
        }

        if(pitchsign == 1)
        {
            if(pitchp > 1000 && pitchp < 9999)
                UARTprintf("Pitch: -0.%d\r\n", pitchp);
            else if(pitchp < 1000 && pitchp > 100)
                UARTprintf("Pitch: -0.0%d\r\n", pitchp);
            else if(pitchp < 100)
                UARTprintf("Pitch: -0.00%d\r\n", pitchp);
            else
                UARTprintf("Pitch: -%d.%d\r\n", pitchp/10000, ((pitch*-1.0) -
(pitchp/10000))*1000);
        }
        else
        {
            if(pitchp > 1000 && pitchp < 9999)
                UARTprintf("Pitch: 0.%d\r\n", pitchp);
            else if(pitchp < 1000 && pitchp > 100)
                UARTprintf("Pitch: 0.0%d\r\n", pitchp);
            else if(pitchp < 100)
                UARTprintf("Pitch: 0.00%d\r\n", pitchp);
            else
                UARTprintf("Pitch: %d.%d\r\n", pitchp/10000, (pitch -
(pitchp/10000))*1000);
        }

        SysCtlDelay(13500000);  // ~1sec delay

    }
}
```

Nathan Hanuscin
Github root directory: https://github.com/nhanuscin/HappyFunStuff

Task 04:

Youtube Link: https://www.youtube.com/watch?v=WdjwwapB-7Y

Modified Schematic (if applicable): Same as Task 1

Modified Code:

Same code as Task 03

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.