CPE403 – Advanced Embedded Systems

Design Assignment #2

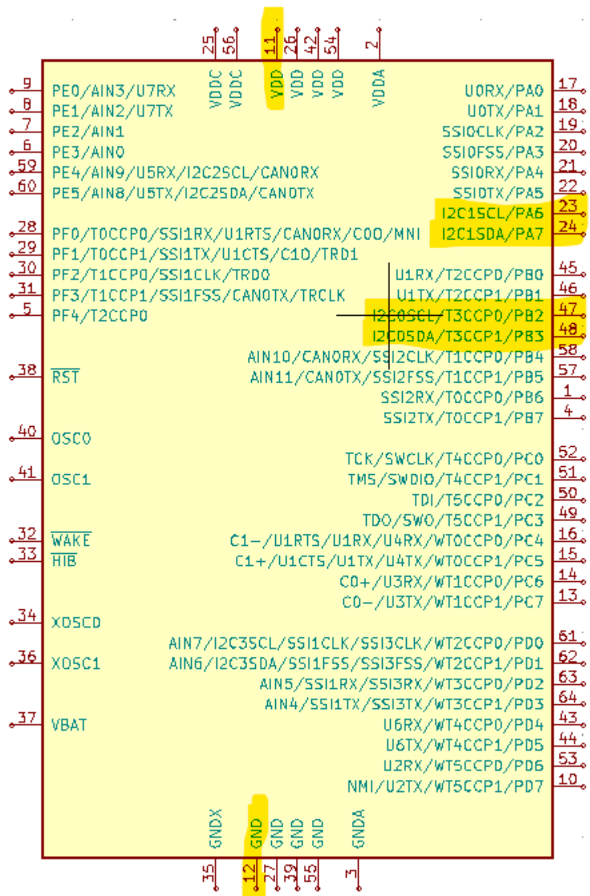
Name: Brian Wolak

Email: wolak@unlv.nevada.edu

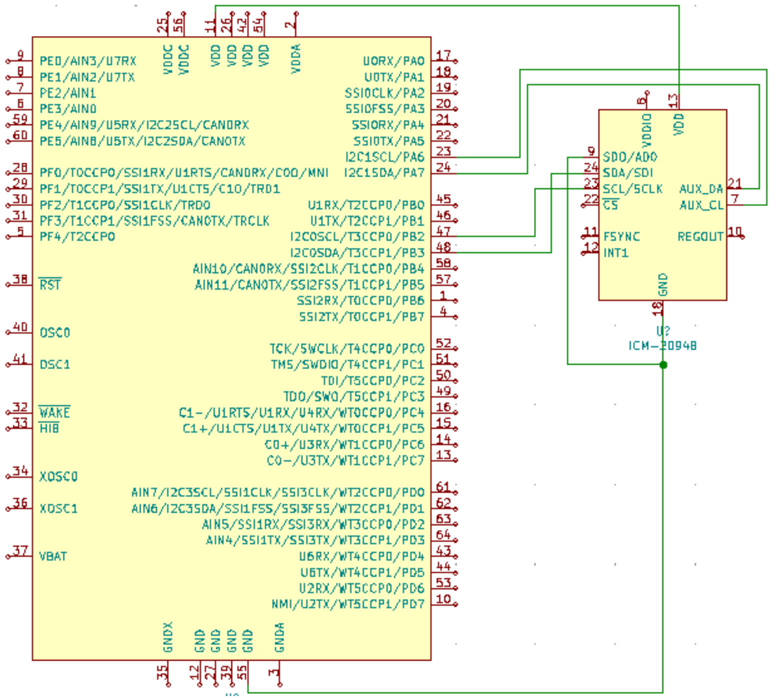
GitHub Repository link (root): https://github.com/brianwolak/advanced_submissions

YouTube Playlist (root): https://youtube.com/playlist?list=PLl6a3M--0IcYnZIMGyucOLe8c-16wAClN

Interface the ICM-20948(IMU) and SSD1306* (OLED) modules to TIVAC. Determine the Euler angles (Roll, Pitch, Yaw) using a filter of your choice(Complementary, Madgwick or Mahony Orientation Filter, etc.). Perform all computations using IQMath Structures and Functions. Display the results in the terminal and SSD1306 OLED* (extra credit not completed).



Tiva-C TM4C123 Pins Used



Tiva-C TM4C123 & IMU-20948 Circuit

C Code:

```c
#include <stdio.h>
#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 "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/debug.h"
#include "utils/uartstdio.h"
#include <string.h>
#include "IQmath/IQmathLib.h"
#include "math.h"
#include "inc/tm4c123gh6pm.h"
#include "ICM20948.h"
// global variables
char inputRead[10];
char displayACCEL[13];

void masterI2Cinit (){
    // configure I2C0
```

```
    SysCtlPeripheralEnable (SYSCTL_PERIPH_I2C0);                        // enable I2C0
    SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);                       // enable PORTB

    GPIOPinTypeI2C (GPIO_PORTB_BASE, GPIO_PIN_3);                       // set pin PB3 as
SDA
    GPIOPinConfigure (GPIO_PB3_I2C0SDA);

    GPIOPinTypeI2CSCL (GPIO_PORTB_BASE, GPIO_PIN_2);                    //set pin PB2 as
SCLK
    GPIOPinConfigure (GPIO_PB2_I2C0SCL);

    I2CMasterInitExpClk (I2C0_BASE, SysCtlClockGet(), false);           // set I2C0 clock
    while (I2CMasterBusy (I2C0_BASE));                                  // wait until
master is free
    // configure I2C1
    SysCtlPeripheralEnable (SYSCTL_PERIPH_I2C1);                        // enable I2C1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);                        // enable PORTA
for I2C1

    GPIOPinTypeI2C (GPIO_PORTA_BASE, GPIO_PIN_7);                       // set pin PA7 as
SDA
    GPIOPinConfigure (GPIO_PA7_I2C1SDA);

    GPIOPinTypeI2CSCL (GPIO_PORTA_BASE, GPIO_PIN_6);                    // set pin PA6 as
SCLK
    GPIOPinConfigure (GPIO_PA6_I2C1SCL);

    I2CMasterInitExpClk (I2C1_BASE, SysCtlClockGet(), false);           // set I2C1 clock
    while (I2CMasterBusy (I2C1_BASE));                                  // wait until
master is free
}
void UART0config(void){
    // ENABLE PERIPHERAL UART 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // ENABLE GPIO PORT A,FOR UART
    GPIOPinConfigure(GPIO_PA0_U0RX);                                    // PA0 IS
CONFIGURED TO UART RX
    GPIOPinConfigure(GPIO_PA1_U0TX);                                    // PA1 IS
CONFIGURED TO UART TX
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    UARTStdioConfig(0, 115200, 16000000);
}

// I2C0 write function
void I2C0_Write (uint8_t addr, uint8_t N, ...){
    I2CMasterSlaveAddrSet (I2C0_BASE, addr, false);                             // find
I2C0 device
    while (I2CMasterBusy (I2C0_BASE));
    va_list vargs;
    va_start (vargs, N);                                                        //
initialize arguments using unit8_t N
```

```
    I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));                   // send
first argument
    while (I2CMasterBusy (I2C0 BASE));
    if (N == 1){
        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        while (I2CMasterBusy (I2C0 BASE));
        va_end (vargs);
    }
    else{
        // loop for multiple send
        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);      // burst
send start
        while (I2CMasterBusy (I2C0 BASE));
        int k;
        for (k=1; k<N-1; k++){
            I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));           // send
next register
            while (I2CMasterBusy (I2C0 BASE));                              // wait
til master is free
            I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);   // burst
send from master
            while (I2CMasterBusy (I2C0 BASE));                              // wait
til master is free
        }
        I2CMasterDataPut (I2C0_BASE, va_arg(vargs, uint8_t));               // large
argument send
        while (I2CMasterBusy (I2C0 BASE));                                  // wait
til master is free
        I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);     // send
finish signal
        while (I2CMasterBusy (I2C0 BASE));                                  // wait
til master is free
        va_end (vargs);
    }

}
// I2C1 write function
void I2C1_Write (uint8_t addr, uint8_t N, ...){
    I2CMasterSlaveAddrSet (I2C1_BASE, addr, false);                             //
find I2C1 device
    while (I2CMasterBusy (I2C1 BASE));                                          //
wait til master is free
    va_list vargs;
    va_start (vargs, N);                                                        //
initialize arguments using unit8_t N

    I2CMasterDataPut (I2C1_BASE, va_arg(vargs, uint8_t));                       //
send first argument
    while (I2CMasterBusy (I2C1_BASE));                                          //
wait til master is free
    if (N == 1){                                                                //
send single argument
        I2CMasterControl (I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        while (I2CMasterBusy (I2C1 BASE));                                      //
wait til master is free
```

```
        va_end (vargs);
    }
    else{
        // loop for multiple send
        I2CMasterControl (I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);          //
burst send start
        while (I2CMasterBusy (I2C1_BASE));                                      //
wait til master is free
        int k;
        for (k=1; k<N-1; k++){
            I2CMasterDataPut (I2C1_BASE, va_arg(vargs, uint8_t));              //
send next register
            while (I2CMasterBusy (I2C1_BASE));                                  //
wait til master is free
            I2CMasterControl (I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);       //
burst send
            while (I2CMasterBusy (I2C1_BASE));                                  //
wait til master is free
        }
        I2CMasterDataPut (I2C1_BASE, va_arg(vargs, uint8_t));                  //
large argument send
        while (I2CMasterBusy (I2C1_BASE));                                      //
wait til master is free
        I2CMasterControl (I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);         //
send finish signal
        while (I2CMasterBusy (I2C1_BASE));                                      //
wait til masater is free
        va_end (vargs);
    }

}
// I2C0 read function
uint32_t I2C0_Read (uint8_t addr, uint8_t location){
    I2CMasterSlaveAddrSet (I2C0_BASE, addr, false);                             //
find I2C0 device
    while (I2CMasterBusy (I2C0_BASE));                                          //
wait

    I2CMasterDataPut (I2C0_BASE, location);                                     //
send read location
    while (I2CMasterBusy (I2C0_BASE));                                          //
wait

    I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);                   //
send the send signal to send the register value
    while (I2CMasterBusy (I2C0_BASE));                                          //
wait

    I2CMasterSlaveAddrSet (I2C0_BASE, addr, true);                              //
set the master to read from the device
    while (I2CMasterBusy (I2C0_BASE));                                          //
wait

    I2CMasterControl (I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);                //
send the receive signal to the device
```

```
    while (I2CMasterBusy (I2C0_BASE));                                          //
wait

    return I2CMasterDataGet (I2C0_BASE);                                        //
return read data
}
// I2C1 read function
uint32_t I2C1_Read (uint8_t addr, uint8_t location){
    I2CMasterSlaveAddrSet (I2C1_BASE, addr, false);                             //
find I2C0 device
    while (I2CMasterBusy (I2C1_BASE));                                          //
wait

    I2CMasterDataPut (I2C1_BASE, location);                                     //
send read location
    while (I2CMasterBusy (I2C1_BASE));                                          //
wait

    I2CMasterControl (I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);                   //
send the send signal to send the register value
    while (I2CMasterBusy (I2C1_BASE));                                          //
wait

    I2CMasterSlaveAddrSet (I2C1_BASE, addr, true);                              //
set the master to read from the device
    while (I2CMasterBusy (I2C1_BASE));                                          //
wait

    I2CMasterControl (I2C1_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);                //
send the receive signal to the device
    while (I2CMasterBusy (I2C1_BASE));                                          //
wait

    return I2CMasterDataGet (I2C1_BASE);                                        //
return read data
}
// ICM20948 configure function
void IMU20948_init (){
    uint8_t dev1, dev2;                                                         //
IMU20948 variables
    UARTprintf("Beginning IMU20948 Configuration.. \n");
    // verify device
    dev1 = I2C0_Read (0x68, 0x00);                                              //
device 1 read 0x00 address
    if (dev1 == 0xEA){                                                          //
check device ID
        UARTprintf("IMU20948 found! Device ID = %d \n", dev1);
    }
    else{
        UARTprintf("IMU20948 device not found, Device ID = %d\n", dev1);
    }
    SysCtlDelay (10000000);
    I2C0_Write(104, 208, 0x06, 0x01);                                           //
send 0x01 to register 0x06
```

```
    I2C0_Write(68, 0,INT_PIN_CFG, 0x02);                                        //
pass through mode
    I2C1_Write(AK09916_ADDRESS, 0, AK09916_CNTL2, 0x08);                        //
enable magnetometer
    dev2 = I2C1_Read (AK09916_ADDRESS, WHO_AM_I_AK09916);                       //
read who am i
    I2C1_Read(AK09916_ADDRESS,AK09916_ST2);
    // verify magnetometer
    if(dev2 == 9){                                                             // check
dev2 ID
        UARTprintf("Magnetometer found! Device ID = %d \n", dev2);
    }
    else{
        UARTprintf("Magnetometer not found, Devioce ID = %d\n", dev2);
    }
}

// tilt function used to calculate roll, pitch and yaw
void applyTiltFilter(_iq13 XmagAVG, _iq13 YmagAVG, _iq13 ZmagAVG, _iq13 XaccelAVG,
_iq13 YaccelAVG, _iq13 ZaccelAVG){
    // iq13 tilt variables
    _iq13 roll, pitch, yaw, accelVector, magVector;
    _iq13 sum1, sum2, Pi, Xmag, Ymag, Zmag, Xaccel, Yaccel, Zaccel,  rX, rY;

    Pi = _IQ13(3.14159);                                                    // iq13 pi
value used to convert
    // normal vector calculations
    sum1 = _IQ13mpy(XaccelAVG, XaccelAVG) + _IQ13mpy(YaccelAVG, YaccelAVG) +
_IQ13mpy(ZaccelAVG, ZaccelAVG);
    sum2 = _IQ13mpy(XmagAVG, XmagAVG) + _IQ13mpy(YmagAVG, YmagAVG) +
_IQ13mpy(ZmagAVG, ZmagAVG);
    accelVector = _IQ13sqrt(sum1);
    magVector = _IQ15sqrt(sum2);

    Xmag = _IQ13div(XmagAVG, magVector);                                    // x AVG
magnetometer unit vector
    Ymag = _IQ13div(YmagAVG, magVector);                                    // y AVG
magnetometer unit vector
    Zmag = _IQ13div(ZmagAVG, magVector);                                    // z AVG
magnetometer unit vector
    Xaccel = _IQ13div(XaccelAVG, accelVector);                              // x AVG
acceleration unit vector
    Yaccel = _IQ13div(YaccelAVG, accelVector);                              // y AVG
acceleration unit vector
    Zaccel = _IQ13div(ZaccelAVG, accelVector);                              // z AVG
acceleration unit vector

    // calculate roll and pitch
    roll = _IQ13atan2(Yaccel, Zaccel);
    pitch = _IQ13atan2(_IQ13mpy(Xaccel, _IQ13(-1)), _IQ13mag(Yaccel, Zaccel));

    // calculate yaw
    rX = _IQ13mpy(Xmag, _IQ13cos(roll)) + _IQ13mpy(_IQ13mpy(Ymag, _IQ13sin(roll)),
_IQ13sin(pitch)) +  _IQ13mpy(_IQ13mpy(Zmag, _IQ13sin(roll)), _IQ13cos(pitch));
    rY = _IQ13mpy(Ymag, _IQ13cos(pitch)) - _IQ13mpy(Zmag, _IQ13sin(pitch));
```

```
    yaw = _IQ13atan2(_IQ13mpy(rY, _IQ13(-1)), rX);

    // convert from radian to degrees
    pitch = _IQ13div(_IQ13mpy(pitch, _IQ13(180)), Pi);
    roll = _IQ13div(_IQ13mpy(roll, _IQ13(180)), Pi);
    yaw = _IQ13div(_IQ13mpy(yaw, _IQ13(180)), Pi);

    // UART display
    _IQ13toa(displayACCEL, "%4.4f", pitch);                 // convert pitch to float
    UARTprintf("\nPitch is %s \n", displayACCEL);
    _IQ13toa(displayACCEL, "%4.4f", roll);                  // convert roll to float
    UARTprintf("Roll is %s \n", displayACCEL);
    _IQ13toa(displayACCEL, "%4.4f", yaw);                   // convert yaw to float
    UARTprintf("Yaw is %s \n\n", displayACCEL);
}
// magnetometer calculation function
_iq13 magnetometer(uint16_t magH, uint16_t magL){

    _iq13 magUP, magLO, magSensitivity;
    // convert values to iq
    magUP = _IQ13(magH);
    magLO = _IQ13(magL);

    //Normalize the value and then return it
    magUP = (magUP + magLO) - _IQ13(500);                // subtract 500
    magSensitivity = _IQ13(0.15);                        // set mag sensitivity
    magUP = _IQ13mpy(magUP, magSensitivity);             // multiply by .15
    return magUP;
}
// accelerometer calculation function
_iq13 accelerometer(uint16_t XaccelH, uint16_t XaccelL){

    _iq13 Aaccel, Baccel;
    // convert values to iq
    Aaccel = _IQ13(XaccelH);
    Baccel = _IQ13(XaccelL);

//  UARTprintf("hello from accel \n");
    Aaccel = (_IQmpy64(Aaccel));                        // shift 8 bits
    Aaccel = (_IQmpy4(Aaccel));
    Aaccel = Aaccel + Baccel;                           // add high and low bits
    Aaccel = (_IQdiv64(Aaccel));                        // divide by 16,384 value
    Aaccel = (_IQdiv64(Aaccel));
    Aaccel = (_IQdiv4(Aaccel));
    return Aaccel;
}

void main (void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
// set 40Mhz clock

    int16_t XaccelH, XaccelL, YaccelH, YaccelL, ZaccelH, ZaccelL, XmagH, XmagL,
YmagH, YmagL, ZmagH, ZmagL;
    _iq13 XaccelAVG, YaccelAVG, ZaccelAVG, XaccelTOTAL, YaccelTOTAL, ZaccelTOTAL;
```

```
    _iq13 XmagTOTAL, YmagTOTAL, ZmagTOTAL, XmagAVG, YmagAVG, ZmagAVG;
    int j;

    UART0config();                                                  // configure UART0
    masterI2Cinit();                                                // configure I2C0 &
I2C1
    IMU20948_init();                                                // configure IMU20948

    while(1){
          XmagTOTAL = 0;                              // set totals to zero at beginning of
every loop
          YmagTOTAL = 0;
          ZmagTOTAL = 0;
          XaccelTOTAL = 0;
          YaccelTOTAL = 0;
          ZaccelTOTAL = 0;

      // average calculation of accelerometer and magnetometer
        for (j = 0; j < 16; j++) {
          // I2C0 read accelerometer values
          XaccelH = I2C0_Read(104, 0x2D);
          XaccelL = I2C0_Read(104, 0x2E);
          YaccelH = I2C0_Read(104, 0x2F);
          YaccelL = I2C0_Read(104, 0x30);
          ZaccelH = I2C0_Read(104, 0x31);
          ZaccelL = I2C0_Read(104, 0x32);
          // calculate accel totals
          XaccelTOTAL = XaccelTOTAL + accelerometer(XaccelH, XaccelL);
          YaccelTOTAL = YaccelTOTAL + accelerometer(YaccelH, YaccelL);
          ZaccelTOTAL = ZaccelTOTAL + accelerometer(ZaccelH, ZaccelL);
          // I2C1 read magnetometer values
          while(!(I2C1_Read(AK09916_ADDRESS,AK09916_ST1)&1)){}
          XmagH = I2C1_Read(AK09916_ADDRESS, AK09916_XOUT_H);
          XmagL = I2C1_Read(AK09916_ADDRESS, AK09916_XOUT_L);
          YmagH = I2C1_Read(AK09916_ADDRESS, AK09916_YOUT_H);
          YmagL = I2C1_Read(AK09916_ADDRESS, AK09916_YOUT_L);
          ZmagH = I2C1_Read(AK09916_ADDRESS, AK09916_ZOUT_H);
          ZmagL = I2C1_Read(AK09916_ADDRESS, AK09916_ZOUT_L);
          I2C1_Read(AK09916_ADDRESS,AK09916_ST2);
// finish reading I2C1
//          UARTprintf("hello from while loop \n");
          XmagTOTAL = XmagTOTAL + magnetometer(XmagH, XmagL);
// calculate X magnetometer total
          YmagTOTAL = YmagTOTAL + magnetometer(YmagH, YmagL);
// calculate Y magnetometer total
          ZmagTOTAL = ZmagTOTAL + magnetometer(ZmagH, ZmagL);
    }
    XaccelAVG = _IQdiv16(XaccelTOTAL) - _IQ13(2);
// subtract 2
    YaccelAVG = _IQdiv16(YaccelTOTAL) - _IQ13(2);
    ZaccelAVG = _IQdiv16(ZaccelTOTAL) - _IQ13(2);
    // calculate magnetometer averages
    XmagAVG = _IQdiv16(XmagTOTAL);
    YmagAVG = _IQdiv16(YmagTOTAL);
    ZmagAVG = _IQdiv16(ZmagTOTAL);
```

```
    applyTiltFilter(XaccelAVG, YaccelAVG, ZaccelAVG, XmagAVG, YmagAVG, ZmagAVG);
// call tilt filter

    SysCtlDelay(100 * (SysCtlClockGet() / 3 / 1000));
// delay 10ms
   }
}
```

---------- INDIVIDUAL TASK SNIPS ----------

Interfacing IMU20948:

```
// ICM20948 configure function
void IMU20948_init (){
    uint8_t dev1, dev2;                                                      //
IMU20948 variables
    UARTprintf("Beginning IMU20948 Configuration.. \n");
    // verify device
    dev1 = I2C0_Read (0x68, 0x00);                                           //
device 1 read 0x00 address
    if (dev1 == 0xEA){                                                       //
check device ID
        UARTprintf("IMU20948 found! Device ID = %d \n", dev1);
    }
    else{
        UARTprintf("IMU20948 device not found, Device ID = %d\n", dev1);
    }
    SysCtlDelay (10000000);
    I2C0_Write(104, 208, 0x06, 0x01);                                        //
send 0x01 to register 0x06
    I2C0_Write(68, 0,INT_PIN_CFG, 0x02);                                     //
pass through mode
    I2C1_Write(AK09916_ADDRESS, 0, AK09916_CNTL2, 0x08);                     //
enable magnetometer
    dev2 = I2C1_Read (AK09916_ADDRESS, WHO_AM_I_AK09916);                    //
read who am i
    I2C1_Read(AK09916_ADDRESS,AK09916_ST2);
    // verify magnetometer
    if(dev2 == 9){                                                        // check
dev2 ID
        UARTprintf("Magnetometer found! Device ID = %d \n", dev2);
    }
    else{
        UARTprintf("Magnetometer not found, Devioce ID = %d\n", dev2);
    }
}
```

Calculate Roll, Pitch, and Yaw using Tilt Filter:

```
// tilt function used to calculate roll, pitch and yaw
void applyTiltFilter(_iq13 XmagAVG, _iq13 YmagAVG, _iq13 ZmagAVG, _iq13 XaccelAVG,
_iq13 YaccelAVG, _iq13 ZaccelAVG){
    // iq13 tilt variables
    _iq13 roll, pitch, yaw, accelVector, magVector;
    _iq13 sum1, sum2, Pi, Xmag, Ymag, Zmag, Xaccel, Yaccel, Zaccel,  rX, rY;

    Pi = _IQ13(3.14159);                                        // iq13 pi
value used to convert
    // normal vector calculations
    sum1 = _IQ13mpy(XaccelAVG, XaccelAVG) + _IQ13mpy(YaccelAVG, YaccelAVG) +
_IQ13mpy(ZaccelAVG, ZaccelAVG);
    sum2 = _IQ13mpy(XmagAVG, XmagAVG) + _IQ13mpy(YmagAVG, YmagAVG) +
_IQ13mpy(ZmagAVG, ZmagAVG);
    accelVector = _IQ13sqrt(sum1);
    magVector = _IQ15sqrt(sum2);

    Xmag = _IQ13div(XmagAVG, magVector);                        // x AVG
magnetometer unit vector
    Ymag = _IQ13div(YmagAVG, magVector);                        // y AVG
magnetometer unit vector
    Zmag = _IQ13div(ZmagAVG, magVector);                        // z AVG
magnetometer unit vector
    Xaccel = _IQ13div(XaccelAVG, accelVector);                  // x AVG
acceleration unit vector
    Yaccel = _IQ13div(YaccelAVG, accelVector);                  // y AVG
acceleration unit vector
    Zaccel = _IQ13div(ZaccelAVG, accelVector);                  // z AVG
acceleration unit vector

    // calculate roll and pitch
    roll = _IQ13atan2(Yaccel, Zaccel);
    pitch = _IQ13atan2(_IQ13mpy(Xaccel, _IQ13(-1)), _IQ13mag(Yaccel, Zaccel));

    // calculate yaw
    rX = _IQ13mpy(Xmag, _IQ13cos(roll)) + _IQ13mpy(_IQ13mpy(Ymag, _IQ13sin(roll)),
_IQ13sin(pitch)) +  _IQ13mpy(_IQ13mpy(Zmag, _IQ13sin(roll)), _IQ13cos(pitch));
    rY = _IQ13mpy(Ymag, _IQ13cos(pitch)) - _IQ13mpy(Zmag, _IQ13sin(pitch));
    yaw = _IQ13atan2(_IQ13mpy(rY, _IQ13(-1)), rX);

    // convert from radian to degrees
    pitch = _IQ13div(_IQ13mpy(pitch, _IQ13(180)), Pi);
    roll = _IQ13div(_IQ13mpy(roll, _IQ13(180)), Pi);
    yaw = _IQ13div(_IQ13mpy(yaw, _IQ13(180)), Pi);

    // UART display
    _IQ13toa(displayACCEL, "%4.4f", pitch);             // convert pitch to float
    UARTprintf("\nPitch is %s \n", displayACCEL);
    _IQ13toa(displayACCEL, "%4.4f", roll);              // convert roll to float
    UARTprintf("Roll is %s \n", displayACCEL);
    _IQ13toa(displayACCEL, "%4.4f", yaw);               // convert yaw to float
    UARTprintf("Yaw is %s \n\n", displayACCEL);
```

```
}
```

Display Results in Terminal:

```
// UART display
 _IQ13toa(displayACCEL, "%4.4f", pitch);                 // convert pitch to float
 UARTprintf("\nPitch is %s \n", displayACCEL);
 _IQ13toa(displayACCEL, "%4.4f", roll);                  // convert roll to float
 UARTprintf("Roll is %s \n", displayACCEL);
 _IQ13toa(displayACCEL, "%4.4f", yaw);                   // convert yaw to float
 UARTprintf("Yaw is %s \n\n", displayACCEL);
```

Video and GitHub Links:

GitHub: https://github.com/brianwolak/advanced_submissions/tree/main/DA_2

YouTube: **https://youtu.be/WVS3PalK-cw**

"This assignment submission is my own, original work".
Brian Wolak