TITLE: TIVAC MIDTERM

GOAL:
Interface the given MPU6050 IMU using I2C protocol to TivaC, And then use complementary filter to filter the raw data

DELIVERABLES:
We can use mpu 6050 to measure the 3-axis gyroscope and 3-axis accelerometer to track the motion of the device.

COMPONENTS:

The MPU-6050 features three 16-bit ADCs for digitizing the gyro scope outputs and three 16-bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000 $°/sec$ (dps) and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g.

Communication with all registers of the mpu-6050 is using I2C at 400kHz

SCHEMATICS:

IIMPLEMENTATION:

initialization of I2C:

```
void InitI2C0(void)
{
   SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

   SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
   GPIOPinConfigure(GPIO_PB2_I2C0SCL);
   GPIOPinConfigure(GPIO_PB3_I2C0SDA);

   GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
   GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
```

```
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);

    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;

}
```

Initialization of UART:

```
void ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
```

```
    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);

}
```

Use I2C send to set MPU 6050 as slave

```
I2CSend(ACCEL_SLAVE_ADDR, 2, 0x6B , 0x00);

void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    va_list vargs;
    va_start(vargs, num_of_args);
    I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
    if(num_of_args == 1)
    {
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        while(I2CMasterBusy(I2C0_BASE));
        va_end(vargs);
    }
    else
    {
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
        while(I2CMasterBusy(I2C0_BASE));
        uint8_t i;
        for(i = 1; i < (num_of_args - 1); i++)
```

```
        {
            I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
            I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
            while(I2CMasterBusy(I2C0_BASE));
        }
        I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
        while(I2CMasterBusy(I2C0_BASE));
        va_end(vargs);
    }
}
```

Use I2C to receive 1 set of data:

```
I2CSend(ACCEL_SLAVE_ADDR,2, 0x1C, 0x00);
        AXH = ReadAccel(XOUTH8) << 8;
        AXL = ReadAccel(XOUTL8);
        AYH = ReadAccel(YOUTH8)<<8;
        AYL = ReadAccel(YOUTL8);
        AZH = ReadAccel(ZOUTH8)<<8;
        AZL = ReadAccel(ZOUTL8);
        GXH = ReadAccel(GXOUTH)<<8;
        GXL = ReadAccel(GXOUTL);
        GYH = ReadAccel(GYOUTH)<<8;
        GYL = ReadAccel(GYOUTL);
        GZH = ReadAccel(GZOUTH)<<8;
        GZL = ReadAccel(GZOUTL);
```

```
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
{
    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));

    return I2CMasterDataGet(I2C0_BASE);
}
```

print raw data:

```
        UARTprintf("ACX: \t%d\n", AX);
        UARTprintf("ACY: \t%d\n", AY);
        UARTprintf("ACZ: \t%d\n", AZ);
        UARTprintf("GCX: \t%d\n", GX);
        UARTprintf("GCY: \t%d\n", GY);
        UARTprintf("GCZ: \t%d\n", GZ);
```

Use complementary filter to get roll and pitch:

```
ComplementaryFilter(AX,AY,AZ,GX,GY,GZ,&a,&b);


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

}
```

CODE:

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "math.h"
#include "IQmath/IQmathLib.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "inc/hw_i2c.h"
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/fpu.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "grlib/grlib.h"
#include "drivers/cfal96x64x16.h"
#include "utils/uartstdio.h"
#include "driverlib/gpio.h"



#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
```

```
#include "driverlib/debug.h"
#include "driverlib/i2c.h"
#define ACCELEROMETER_SENSITIVITY 8192.0
#define GYROSCOPE_SENSITIVITY 65.536
#define M_PI 3.14159265359
#define dt 0.01 // 10 ms sample rate!

#define ACCEL_SLAVE_ADDR 0x68
#define XOUTH8 0x3B
#define XOUTL8 0x3C
#define YOUTH8 0x3D
#define YOUTL8 0x3E
#define ZOUTH8 0x3F
#define ZOUTL8 0x40

#define GXOUTH 0x43
#define GXOUTL 0x44
#define GYOUTH 0x45
#define GYOUTL 0x46
#define GZOUTH 0x47
#define GZOUTL 0x48

volatile int16_t AXH;
volatile int16_t AXL;
volatile int16_t AX;

volatile int16_t AYH;
```

```
volatile int16_t AYL;
volatile int16_t AY;

volatile int16_t AZH;
volatile int16_t AZL;
volatile int16_t AZ;

volatile int16_t GXH;
volatile int16_t GXL;
volatile int16_t GX;

volatile int16_t GYH;
volatile int16_t GYL;
volatile int16_t GY;

volatile int16_t GZH;
volatile int16_t GZL;
volatile int16_t GZ;


float a = 0,b=0;
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif
```

```
void InitI2C0(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);




    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);

    HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
}
void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
    va_list vargs;
    va_start(vargs, num_of_args);
```

```
    I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
    if(num_of_args == 1)
    {
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        while(I2CMasterBusy(I2C0_BASE));
        va_end(vargs);
    }
    else
    {
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
        while(I2CMasterBusy(I2C0_BASE));
        uint8_t i;
        for(i = 1; i < (num_of_args - 1); i++)
        {
            I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
            I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
            while(I2CMasterBusy(I2C0_BASE));
        }
        I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
        while(I2CMasterBusy(I2C0_BASE));
        va_end(vargs);
    }
}
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
{
    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));

    return I2CMasterDataGet(I2C0_BASE);
}
uint8_t ReadAccel(uint8_t reg)
{
    uint8_t accelData = I2CReceive(ACCEL_SLAVE_ADDR, reg);

    return accelData;
}
void ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

```
    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

//*****************************************************************************
//
// Print "Hello World!" to the display.
//
//*****************************************************************************
int main(void)
{
```

```
    char PCprint[10];

    // Enable lazy stacking for interrupt handlers.  This allows floating-point
    // instructions to be used within interrupt handlers, but at the expense of
    // extra stack usage.
    //
    ROM_FPULazyStackingEnable();

    //
    // Set the clocking to run directly from the crystal.
    //
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL |
SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

    //
    // Initialize the UART.
    //
    ConfigureUART();

       InitI2C0();

     I2CSend(ACCEL_SLAVE_ADDR, 2, 0x6B , 0x00);
       while(1)
       {
       I2CSend(ACCEL_SLAVE_ADDR,2, 0x1C, 0x00);
       AXH = ReadAccel(XOUTH8) << 8;
       AXL = ReadAccel(XOUTL8);
```

```
AYH = ReadAccel(YOUTH8)<<8;
AYL = ReadAccel(YOUTL8);
AZH = ReadAccel(ZOUTH8)<<8;
AZL = ReadAccel(ZOUTL8);
GXH = ReadAccel(GXOUTH)<<8;
GXL = ReadAccel(GXOUTL);
GYH = ReadAccel(GYOUTH)<<8;
GYL = ReadAccel(GYOUTL);
GZH = ReadAccel(GZOUTH)<<8;
GZL = ReadAccel(GZOUTL);

AX = AXH | AXL;
AY = AYH | AYL;
AZ = AZH | AZL;
GX = GXH | GXL;
GY = GYH | GYL;
GZ = GZH | GZL;


UARTprintf("ACX: \t%d\n", AX);
UARTprintf("ACY: \t%d\n", AY);
UARTprintf("ACZ: \t%d\n", AZ);
UARTprintf("GCX: \t%d\n", GX);
UARTprintf("GCY: \t%d\n", GY);
UARTprintf("GCZ: \t%d\n", GZ);

ComplementaryFilter(AX,AY,AZ,GX,GY,GZ,&a,&b);
```

```
        sprintf(PCprint, "roll: %f\n", b);
        UARTprintf("%s", PCprint);
        sprintf(PCprint, "pitch: %f\n", a);
        UARTprintf("%s", PCprint);
        UARTprintf("\033[0;0H");
        SysCtlDelay(10000);
        };
}

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

}
```