CPE301 – SPRING 2019

MIDTERM 2

Student Name: Brian West
Student #: 5003032874
Student Email: westb2@unlv.nevada.edu
Primary Github address: https://github.com/westbrian2/Spring2019
Directory: https://github.com/westbrian2/Spring2019/tree/master/Midterms/

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/Midterm, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

EXPLANATION**

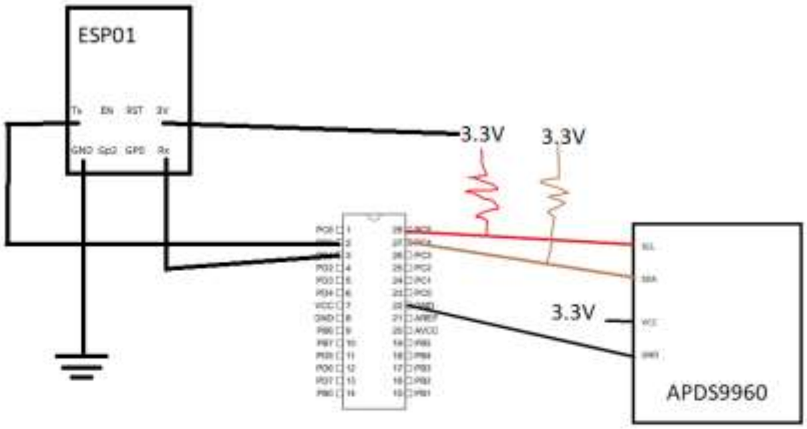
I attempted to get as far as I could with the APDS9660, but I think I burned the board due to an accidental jumper connection. My attempt to work with the APDS9660 got stuck on setting up I2C. For that reason I attempted to finish the rest with the GY-521.

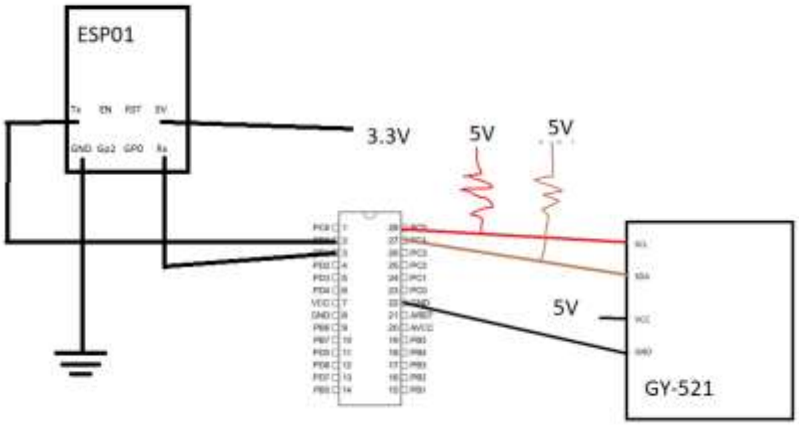
BOTH ATTEMPTS ARE INCLUDED

The ESP01 still isn't working the correctly and will connect to ThingSpeak with ESPlorer, but not when connected to the Xplained Mini.

1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

100k Resistors
Level Shifters
Mini Xplained board
APDS9660
GY-521
ESP01





2. INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A

```
****CODE FOR APDS9960******
#define F_SCL 100000UL // SCL frequency
#define Prescaler 1
#define TWBR_val ((((F_CPU / F_SCL) / Prescaler) - 16 ) / 2)
#define UBRR_9600 103
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <util/twi.h>
/* APDS-9960 I2C address */
#define APDS9960_I2C_ADDR       0x39
#define APDS_WRITE     (0x39 << 1) | 0 //I overheard a student discussing this as a solution to a
problem we both were having, I don't know
#define APDS_READ      (0x39 << 1) | 1 //their name to give proper credit :(
/* Gesture parameters */
#define GESTURE_THRESHOLD_OUT   10
#define GESTURE_SENSITIVITY_1   50
#define GESTURE_SENSITIVITY_2   20

/* Error code for returned values */
#define ERROR                   0xFF

/* Acceptable device IDs */
#define APDS9960_ID_1           0xAB
#define APDS9960_ID_2           0x9C

/* Misc parameters */
#define FIFO_PAUSE_TIME         30      // Wait period (ms) between FIFO reads

/* APDS-9960 register addresses */
#define APDS9960_ENABLE         0x80
#define APDS9960_ATIME          0x81
#define APDS9960_WTIME          0x83
#define APDS9960_AILTL          0x84
#define APDS9960_AILTH          0x85
#define APDS9960_AIHTL          0x86
#define APDS9960_AIHTH          0x87
#define APDS9960_PILT           0x89
#define APDS9960_PIHT           0x8B
#define APDS9960_PERS           0x8C
#define APDS9960_CONFIG1        0x8D
#define APDS9960_PPULSE         0x8E
#define APDS9960_CONTROL        0x8F
#define APDS9960_CONFIG2        0x90
#define APDS9960_ID             0x92
#define APDS9960_STATUS         0x93
#define APDS9960_CDATAL         0x94
```

```
#define APDS9960_CDATAH         0x95
#define APDS9960_RDATAL         0x96
#define APDS9960_RDATAH         0x97
#define APDS9960_GDATAL         0x98
#define APDS9960_GDATAH         0x99
#define APDS9960_BDATAL         0x9A
#define APDS9960_BDATAH         0x9B
#define APDS9960_PDATA          0x9C
#define APDS9960_POFFSET_UR     0x9D
#define APDS9960_POFFSET_DL     0x9E
#define APDS9960_CONFIG3        0x9F
#define APDS9960_GPENTH         0xA0
#define APDS9960_GEXTH          0xA1
#define APDS9960_GCONF1         0xA2
#define APDS9960_GCONF2         0xA3
#define APDS9960_GOFFSET_U      0xA4
#define APDS9960_GOFFSET_D      0xA5
#define APDS9960_GOFFSET_L      0xA7
#define APDS9960_GOFFSET_R      0xA9
#define APDS9960_GPULSE         0xA6
#define APDS9960_GCONF3         0xAA
#define APDS9960_GCONF4         0xAB
#define APDS9960_GFLVL          0xAE
#define APDS9960_GSTATUS        0xAF
#define APDS9960_IFORCE         0xE4
#define APDS9960_PICLEAR        0xE5
#define APDS9960_CICLEAR        0xE6
#define APDS9960_AICLEAR        0xE7
#define APDS9960_GFIFO_U        0xFC
#define APDS9960_GFIFO_D        0xFD
#define APDS9960_GFIFO_L        0xFE
#define APDS9960_GFIFO_R        0xFF

/* Bit fields */
#define APDS9960_PON            0b00000001
#define APDS9960_AEN            0b00000010
#define APDS9960_PEN            0b00000100
#define APDS9960_WEN            0b00001000
#define APSD9960_AIEN           0b00010000
#define APDS9960_PIEN           0b00100000
#define APDS9960_GEN            0b01000000
#define APDS9960_GVALID         0b00000001

/* On/Off definitions */
#define OFF                     0
#define ON                      1

/* Acceptable parameters for setMode */
#define POWER                   0
#define AMBIENT_LIGHT           1
#define PROXIMITY               2
#define WAIT                    3
```

```
#define AMBIENT_LIGHT_INT       4
#define PROXIMITY_INT           5
#define GESTURE                 6
#define ALL                     7

/* LED Drive values */
#define LED_DRIVE_100MA         0
#define LED_DRIVE_50MA          1
#define LED_DRIVE_25MA          2
#define LED_DRIVE_12_5MA        3

/* Proximity Gain (PGAIN) values */
#define PGAIN_1X                0
#define PGAIN_2X                1
#define PGAIN_4X                2
#define PGAIN_8X                3

/* ALS Gain (AGAIN) values */
#define AGAIN_1X                0
#define AGAIN_4X                1
#define AGAIN_16X               2
#define AGAIN_64X               3

/* Gesture Gain (GGAIN) values */
#define GGAIN_1X                0
#define GGAIN_2X                1
#define GGAIN_4X                2
#define GGAIN_8X                3

/* LED Boost values */
#define LED_BOOST_100           0
#define LED_BOOST_150           1
#define LED_BOOST_200           2
#define LED_BOOST_300           3

/* Gesture wait time values */
#define GWTIME_0MS              0
#define GWTIME_2_8MS            1
#define GWTIME_5_6MS            2
#define GWTIME_8_4MS            3
#define GWTIME_14_0MS           4
#define GWTIME_22_4MS           5
#define GWTIME_30_8MS           6
#define GWTIME_39_2MS           7

/* Default values */
#define DEFAULT_ATIME           219     // 103ms
#define DEFAULT_WTIME           246     // 27ms
#define DEFAULT_PROX_PPULSE     0x87    // 16us, 8 pulses
#define DEFAULT_GESTURE_PPULSE  0x89    // 16us, 10 pulses
#define DEFAULT_POFFSET_UR      0       // 0 offset
#define DEFAULT_POFFSET_DL      0       // 0 offset
```

```
#define DEFAULT_CONFIG1         0x60    // No 12x wait (WTIME) factor
#define DEFAULT_LDRIVE          LED_DRIVE_100MA
#define DEFAULT_PGAIN           PGAIN_4X
#define DEFAULT_AGAIN           AGAIN_4X
#define DEFAULT_PILT            0       // Low proximity threshold
#define DEFAULT_PIHT            50      // High proximity threshold
#define DEFAULT_AILT            0xFFFF  // Force interrupt for calibration
#define DEFAULT_AIHT            0
#define DEFAULT_PERS            0x11    // 2 consecutive prox or ALS for int.
#define DEFAULT_CONFIG2         0x01    // No saturation interrupts or LED boost
#define DEFAULT_CONFIG3         0       // Enable all photodiodes, no SAI
#define DEFAULT_GPENTH          40      // Threshold for entering gesture mode
#define DEFAULT_GEXTH           30      // Threshold for exiting gesture mode
#define DEFAULT_GCONF1          0x40    // 4 gesture events for int., 1 for exit
#define DEFAULT_GGAIN           GGAIN_4X
#define DEFAULT_GLDRIVE         LED_DRIVE_100MA
#define DEFAULT_GWTIME          GWTIME_2_8MS
#define DEFAULT_GOFFSET         0       // No offset scaling for gesture mode
#define DEFAULT_GPULSE          0xC9    // 32us, 10 pulses
#define DEFAULT_GCONF3          0       // All photodiodes active during gesture
#define DEFAULT_GIEN            0       // Disable gesture interrupts

#define I2C_READ 0x01
#define I2C_WRITE 0x00
void i2c_init(void)
{
        TWBR = (uint8_t)TWBR_val;
}

void i2c_stop(void)
{
        // transmit STOP condition
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
}

uint8_t i2c_start(uint8_t address)
{
        // reset TWI control register
        TWCR = 0;
        // transmit START condition
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );

        // check if the start condition was successfully transmitted
        if((TWSR & 0xF8) != TW_START){ return 1; }

        // load slave address into data register
        TWDR = address;
        // start transmission of address
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );

        // check if the device has acknowledged the READ / WRITE mode
        uint8_t twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

        return 0;
}

uint8_t i2c_write(uint8_t data)
{
        // load data into data register
        TWDR = data;
        // start transmission of data
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );

        if( (TWSR & 0xF8) != TW_MT_DATA_ACK ){ return 1; }

        return 0;
}

uint8_t i2c_read_ack(void)
{

        // start TWI module and acknowledge data after reception
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );
        // return received data from TWDR
        return TWDR;
}

uint8_t i2c_read_nack(void)
{

        // start receiving without acknowledging reception
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wait for end of transmission
        while( !(TWCR & (1<<TWINT)) );
        // return received data from TWDR
        return TWDR;
}

uint8_t i2c_transmit(uint8_t address, uint8_t* data, uint16_t length)
{
        if (i2c_start(address | I2C_WRITE)) return 1;

        for (uint16_t i = 0; i < length; i++)
        {
                if (i2c_write(data[i])) return 1;
        }

        i2c_stop();

        return 0;
}

uint8_t i2c_receive(uint8_t address, uint8_t* data, uint16_t length)
{
        if (i2c_start(address | I2C_READ)) return 1;

        for (uint16_t i = 0; i < (length-1); i++)
        {
                data[i] = i2c_read_ack();
        }
        data[(length-1)] = i2c_read_nack();

        i2c_stop();

        return 0;
}

uint8_t i2c_writeReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length)
{
        if (i2c_start(devaddr | 0x00)) return 1;

        i2c_write(regaddr);

        for (uint16_t i = 0; i < length; i++)
        {
                if (i2c_write(data[i])) return 1;
        }

        i2c_stop();

        return 0;
}

uint8_t i2c_readReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length)
{
        if (i2c_start(devaddr)) return 1;

        i2c_write(regaddr);

        if (i2c_start(devaddr | 0x01)) return 1;

        for (uint16_t i = 0; i < (length-1); i++)
        {
                data[i] = i2c_read_ack();
        }
        data[(length-1)] = i2c_read_nack();
        i2c_stop();

        return 0;
}


void apds_init(){
        uint8_t setup;
        //read and write commands
        i2c_readReg(APDS_WRITE, APDS9960_ID, &setup,1);
        if(setup != APDS9960_ID_1) while(1);
        setup = 1 << 1 | 1<<0 | 1<<3 | 1<<4;
        i2c_writeReg(APDS_WRITE, APDS9960_ENABLE, &setup, 1);
        setup = DEFAULT_ATIME;
        i2c_writeReg(APDS_WRITE, APDS9960_ATIME, &setup, 1);
        setup = DEFAULT_WTIME;
        i2c_writeReg(APDS_WRITE, APDS9960_WTIME, &setup, 1);
        setup = DEFAULT_PROX_PPULSE;
        i2c_writeReg(APDS_WRITE, APDS9960_PPULSE, &setup, 1);
        setup = DEFAULT_POFFSET_UR;
        i2c_writeReg(APDS_WRITE, APDS9960_POFFSET_UR, &setup, 1);
        setup = DEFAULT_POFFSET_DL;
        i2c_writeReg(APDS_WRITE, APDS9960_POFFSET_DL, &setup, 1);
        setup = DEFAULT_CONFIG1;
        i2c_writeReg(APDS_WRITE, APDS9960_CONFIG1, &setup, 1);
        setup = DEFAULT_PERS;
        i2c_writeReg(APDS_WRITE, APDS9960_PERS, &setup, 1);
        setup = DEFAULT_CONFIG2;
        i2c_writeReg(APDS_WRITE, APDS9960_CONFIG2, &setup, 1);
        setup = DEFAULT_CONFIG3;
        i2c_writeReg(APDS_WRITE, APDS9960_CONFIG3, &setup, 1);
}

void getData(uint16_t red, uint16_t green, uint16_t blue){
        uint8_t rl,rh,gl,gh,bl,bh; //values for color
        //read i2c vlaues
        i2c_readReg(APDS_WRITE, APDS9960_RDATAL, &rl, 1);
        i2c_readReg(APDS_WRITE, APDS9960_RDATAH, &rh, 1);
        i2c_readReg(APDS_WRITE, APDS9960_GDATAL, &gl, 1);
        i2c_readReg(APDS_WRITE, APDS9960_GDATAH, &gh, 1);
        i2c_readReg(APDS_WRITE, APDS9960_BDATAL, &bl, 1);
        i2c_readReg(APDS_WRITE, APDS9960_BDATAH, &bh, 1);
        red = rh << 8 | rl;
        green = gh << 8 | gl;
        blue = bh << 8 | bl;
}
void USART_init( unsigned int ubrr ) {
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;
        UCSR0B = (1 << TXEN0); // Enable receiver, transmitter & RX interrupt
        UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); //asynchronous 8 N 1
}
```

```
void USART_SendString( char *data ) {
        while ((*data != '\0')) {
                while (!(UCSR0A & (1 <<UDRE0)));
                UDR0 = *data;
                data++;
        }
}

uint16_t red,blue,green;

int main(void){
        char buffer[20];
        i2c_init();
        USART_init(UBRR_9600);
        apds_init(); //initializes APDS

        while(1){
        getData(red,green,blue);
        sprintf(buffer,"Red = %d",red);
        USART_SendString(buffer);
        _delay_ms(5000);
        }

}

********************CODE FOR GY-521***************************************
#define F_CPU 16000000UL                                                /* Define CPU clock
Frequency e.g. here its 8MHz */
#define UBRR_9600 103 // for 16Mhz with .2% error
#define XG_OFFS_TC 0x00
#define YG_OFFS_TC 0x01
#define ZG_OFFS_TC 0x02
#define X_FINE_GAIN 0x03
#define Y_FINE_GAIN 0x04
#define Z_FINE_GAIN 0x05
#define XA_OFFS_H 0x06
#define XA_OFFS_L_TC 0x07
#define YA_OFFS_H 0x08
#define YA_OFFS_L_TC 0x09
#define ZA_OFFS_H 0x0A
#define ZA_OFFS_L_TC 0x0B
#define XG_OFFS_USRH 0x13
#define XG_OFFS_USRL 0x14
#define YG_OFFS_USRH 0x15
#define YG_OFFS_USRL 0x16
#define ZG_OFFS_USRH 0x17
#define ZG_OFFS_USRL 0x18
#define SMPLRT_DIV 0x19
#define CONFIG 0x1A
#define GYRO_CONFIG 0x1B
#define ACCEL_CONFIG 0x1C
#define FF_THR 0x1D
#define FF_DUR 0x1E
#define MOT_THR 0x1F
```

```
#define MOT_DUR 0x20
#define ZRMOT_THR 0x21
#define ZRMOT_DUR 0x22
#define FIFO_EN 0x23
#define I2C_MST_CTRL 0x24
#define I2C_SLV0_ADDR 0x25
#define I2C_SLV0_REG 0x26
#define I2C_SLV0_CTRL 0x27
#define I2C_SLV1_ADDR 0x28
#define I2C_SLV1_REG 0x29
#define I2C_SLV1_CTRL 0x2A
#define I2C_SLV2_ADDR 0x2B
#define I2C_SLV2_REG 0x2C
#define I2C_SLV2_CTRL 0x2D
#define I2C_SLV3_ADDR 0x2E
#define I2C_SLV3_REG 0x2F
#define I2C_SLV3_CTRL 0x30
#define I2C_SLV4_ADDR 0x31
#define I2C_SLV4_REG 0x32
#define I2C_SLV4_DO 0x33
#define I2C_SLV4_CTRL 0x34
#define I2C_SLV4_DI 0x35
#define I2C_MST_STATUS 0x36
#define INT_PIN_CFG 0x37
#define INT_ENABLE 0x38
#define DMP_INT_STATUS 0x39
#define INT_STATUS 0x3A
#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 TEMP_OUT_H 0x41
#define TEMP_OUT_L 0x42
#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
#define EXT_SENS_DATA_00 0x49
#define EXT_SENS_DATA_01 0x4A
#define EXT_SENS_DATA_02 0x4B
#define EXT_SENS_DATA_03 0x4C
#define EXT_SENS_DATA_04 0x4D
#define EXT_SENS_DATA_05 0x4E
#define EXT_SENS_DATA_06 0x4F
#define EXT_SENS_DATA_07 0x50
#define EXT_SENS_DATA_08 0x51
#define EXT_SENS_DATA_09 0x52
#define EXT_SENS_DATA_10 0x53
#define EXT_SENS_DATA_11 0x54
#define EXT_SENS_DATA_12 0x55
#define EXT_SENS_DATA_13 0x56
#define EXT_SENS_DATA_14 0x57
#define EXT_SENS_DATA_15 0x58
#define EXT_SENS_DATA_16 0x59
```

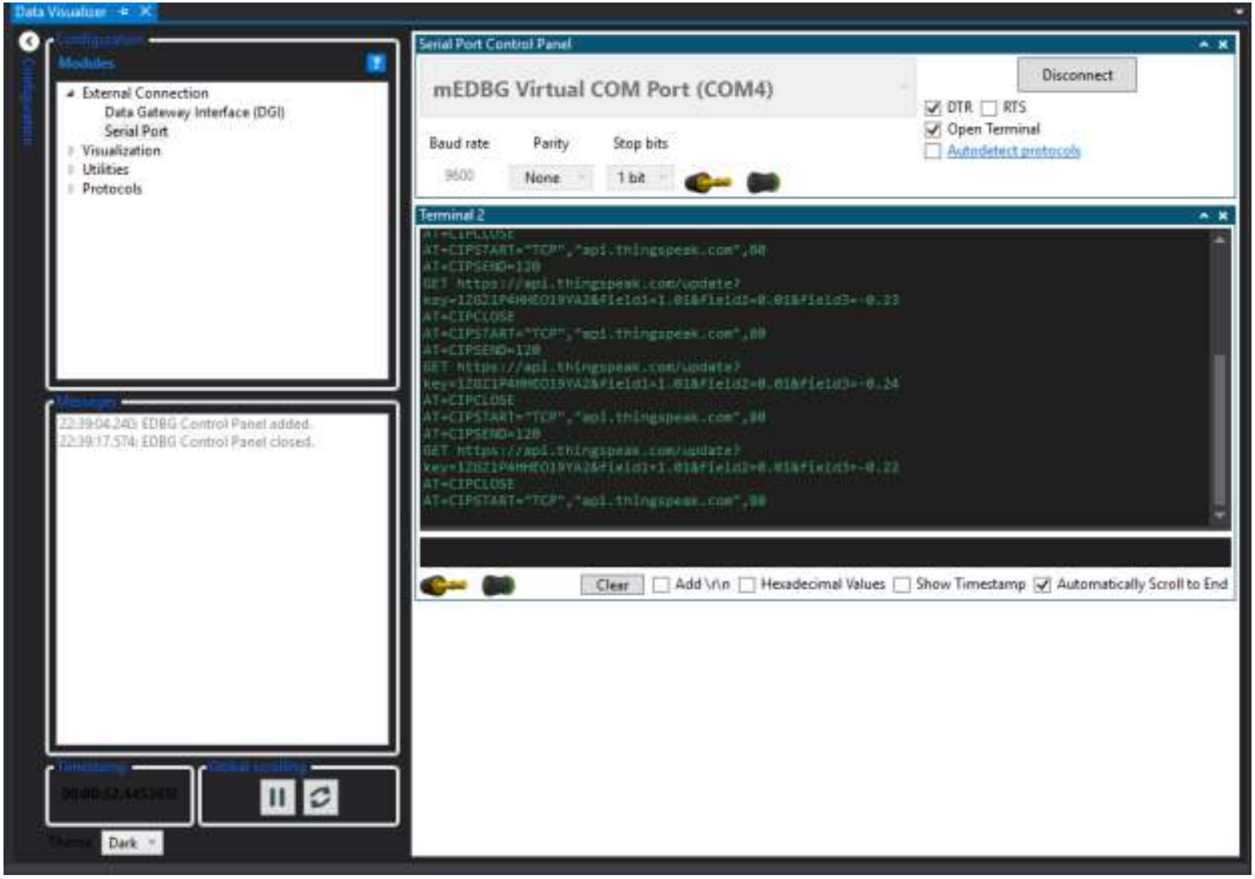
```
#define EXT_SENS_DATA_17 0x5A
#define EXT_SENS_DATA_18 0x5B
#define EXT_SENS_DATA_19 0x5C
#define EXT_SENS_DATA_20 0x5D
#define EXT_SENS_DATA_21 0x5E
#define EXT_SENS_DATA_22 0x5F
#define EXT_SENS_DATA_23 0x60
#define MOT_DETECT_STATUS 0x61
#define I2C_SLV0_DO 0x63
#define I2C_SLV1_DO 0x64
#define I2C_SLV2_DO 0x65
#define I2C_SLV3_DO 0x66
#define I2C_MST_DELAY_CTRL 0x67
#define SIGNAL_PATH_RESET 0x68
#define MOT_DETECT_CTRL 0x69
#define USER_CTRL 0x6A
#define PWR_MGMT_1 0x6B
#define PWR_MGMT_2 0x6C
#define BANK_SEL 0x6D
#define MEM_START_ADDR 0x6E
#define MEM_R_W 0x6F
#define DMP_CFG_1 0x70
#define DMP_CFG_2 0x71
#define FIFO_COUNTH 0x72
#define FIFO_COUNTL 0x73
#define FIFO_R_W 0x74
#define WHO_AM_I 0x75


#include <avr/io.h>                                                     /* Include AVR std.
library file */
#include <util/delay.h>                                                 /* Include delay
header file */
#include <math.h>                                                       /* Include math
function */
#include <stdio.h>
#include <stdlib.h>
#define SCL_CLK 100000L                                                 /* Define SCL clock
frequency */
#define BITRATE(TWSR)      ((F_CPU/SCL_CLK)-16)/(2*pow(4,(TWSR&((1<<TWPS0)|(1<<TWPS1)))))
/* Define bit rate */
#define BAUD_PRESCALE (((F_CPU / (BAUDRATE * 16UL))) - 1)     /* Define prescale value */

float Acc_x,Acc_y,Acc_z,Temperature,Gyro_x,Gyro_y,Gyro_z;
void I2C_Init()
       /* I2C initialize function */
{
       TWBR = BITRATE(TWSR = 0x00);                                                 /* Get
bit rate register value by formula */
}


uint8_t I2C_Start(char slave_write_address)                                          /* I2C
start function */
{
       uint8_t status;
       /* Declare variable */
       TWCR = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT);                              /* Enable
TWI, generate start condition and clear interrupt flag */
       while (!(TWCR & (1<<TWINT)));                                               /*
Wait until TWI finish its current job (start condition) */
       status = TWSR & 0xF8;
       /* Read TWI status register with masking lower three bits */
       if (status != 0x08)
       /* Check weather start condition transmitted successfully or not? */
       return 0;
       /* If not then return 0 to indicate start condition fail */
       TWDR = slave_write_address;                                                 /* If
yes then write SLA+W in TWI data register */
       TWCR = (1<<TWEN)|(1<<TWINT);                                                /*
Enable TWI and clear interrupt flag */
       while (!(TWCR & (1<<TWINT)));                                               /*
Wait until TWI finish its current job (Write operation) */
       status = TWSR & 0xF8;
       /* Read TWI status register with masking lower three bits */
       if (status == 0x18)
       /* Check weather SLA+W transmitted & ack received or not? */
       return 1;
       /* If yes then return 1 to indicate ack received i.e. ready to accept data byte */
       if (status == 0x20)
       /* Check weather SLA+W transmitted & nack received or not? */
       return 2;
       /* If yes then return 2 to indicate nack received i.e. device is busy */
       else
       return 3;
       /* Else return 3 to indicate SLA+W failed */
}

uint8_t I2C_Repeated_Start(char slave_read_address)                  /* I2C repeated
start function */
{
       uint8_t status;
       /* Declare variable */
       TWCR = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT);                              /* Enable
TWI, generate start condition and clear interrupt flag */
       while (!(TWCR & (1<<TWINT)));                                               /*
Wait until TWI finish its current job (start condition) */
       status = TWSR & 0xF8;
       /* Read TWI status register with masking lower three bits */
       if (status != 0x10)
       /* Check weather repeated start condition transmitted successfully or not? */
       return 0;
       /* If no then return 0 to indicate repeated start condition fail */
       TWDR = slave_read_address;                                                  /* If
yes then write SLA+R in TWI data register */
       TWCR = (1<<TWEN)|(1<<TWINT);                                                /*
Enable TWI and clear interrupt flag */
       while (!(TWCR & (1<<TWINT)));                                               /*
Wait until TWI finish its current job (Write operation) */
       status = TWSR & 0xF8;
       /* Read TWI status register with masking lower three bits */
       if (status == 0x40)
       /* Check weather SLA+R transmitted & ack received or not? */
       return 1;
       /* If yes then return 1 to indicate ack received */
       if (status == 0x20)
       /* Check weather SLA+R transmitted & nack received or not? */
       return 2;
       /* If yes then return 2 to indicate nack received i.e. device is busy */
       else
       return 3;
       /* Else return 3 to indicate SLA+W failed */
}

void I2C_Stop()
       /* I2C stop function */
{
       TWCR=(1<<TWSTO)|(1<<TWINT)|(1<<TWEN);                                /* Enable
TWI, generate stop condition and clear interrupt flag */
       while(TWCR & (1<<TWSTO));                                                   /*
Wait until stop condition execution */
}

void I2C_Start_Wait(char slave_write_address)                        /* I2C start wait
function */
{
       uint8_t status;
       /* Declare variable */
       while (1)
       {
              TWCR = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT);                       /* Enable
TWI, generate start condition and clear interrupt flag */
              while (!(TWCR & (1<<TWINT)));                                        /*
Wait until TWI finish its current job (start condition) */
              status = TWSR & 0xF8;
       /* Read TWI status register with masking lower three bits */
              if (status != 0x08)
       /* Check weather start condition transmitted successfully or not? */
              continue;
       /* If no then continue with start loop again */
              TWDR = slave_write_address;                                          /* If
yes then write SLA+W in TWI data register */
              TWCR = (1<<TWEN)|(1<<TWINT);                                         /*
Enable TWI and clear interrupt flag */
              while (!(TWCR & (1<<TWINT)));                                        /*
Wait until TWI finish its current job (Write operation) */
              status = TWSR & 0xF8;
       /* Read TWI status register with masking lower three bits */
              if (status != 0x18 )                                                 /*
Check weather SLA+W transmitted & ack received or not? */
              {
                     I2C_Stop();
       /* If not then generate stop condition */
                     continue;
       /* continue with start loop again */
              }
              break;
       /* If yes then break loop */
       }
}

uint8_t I2C_Write(char data)                                                       /* I2C
write function */
```

```
{
       uint8_t status;
       /* Declare variable */
       TWDR = data;
       /* Copy data in TWI data register */
       TWCR = (1<<TWEN)|(1<<TWINT);                                                 /*
Enable TWI and clear interrupt flag */
       while (!(TWCR & (1<<TWINT)));                                                /*
Wait until TWI finish its current job (Write operation) */
       status = TWSR & 0xF8;
       /* Read TWI status register with masking lower three bits */
       if (status == 0x28)
       /* Check weather data transmitted & ack received or not? */
       return 0;
       /* If yes then return 0 to indicate ack received */
       if (status == 0x30)
       /* Check weather data transmitted & nack received or not? */
       return 1;
       /* If yes then return 1 to indicate nack received */
       else
       return 2;
       /* Else return 2 to indicate data transmission failed */
}

char I2C_Read_Ack()                                                                 /* I2C
read ack function */
{
       TWCR=(1<<TWEN)|(1<<TWINT)|(1<<TWEA);                                  /* Enable
TWI, generation of ack and clear interrupt flag */
       while (!(TWCR & (1<<TWINT)));                                                /*
Wait until TWI finish its current job (read operation) */
       return TWDR;
       /* Return received data */
}

char I2C_Read_Nack()                                                                /* I2C
read nack function */
{
       TWCR=(1<<TWEN)|(1<<TWINT);                                                   /*
Enable TWI and clear interrupt flag */
       while (!(TWCR & (1<<TWINT)));                                                /*
Wait until TWI finish its current job (read operation) */
       return TWDR;
       /* Return received data */
}
void USART_init( unsigned int ubrr ) {
       UBRR0H = (unsigned char)(ubrr>>8);
       UBRR0L = (unsigned char)ubrr;
       UCSR0B = (1 << TXEN0); // Enable receiver, transmitter & RX interrupt
       UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); //asynchronous 8 N 1
}

void USART_SendString( char *data ) {
       while ((*data != '\0')) {
              while (!(UCSR0A & (1 <<UDRE0)));
              UDR0 = *data;
              data++;
       }
```

```
}
void Gyro_Init()            /* Gyro initialization function */
{
        _delay_ms(150);             /* Power up time >100ms */
        I2C_Start_Wait(0xD0);       /* Start with device write address */
        I2C_Write(SMPLRT_DIV);      /* Write to sample rate register */
        I2C_Write(0x07);     /* 1KHz sample rate */
        I2C_Stop();

        I2C_Start_Wait(0xD0);
        I2C_Write(PWR_MGMT_1);      /* Write to power management register */
        I2C_Write(0x01);     /* X axis gyroscope reference frequency */
        I2C_Stop();

        I2C_Start_Wait(0xD0);
        I2C_Write(CONFIG);   /* Write to Configuration register */
        I2C_Write(0x00);     /* Fs = 8KHz */
        I2C_Stop();

        I2C_Start_Wait(0xD0);
        I2C_Write(GYRO_CONFIG);     /* Write to Gyro configuration register */
        I2C_Write(0x18);     /* Full scale range +/- 2000 degree/C */
        I2C_Stop();

        I2C_Start_Wait(0xD0);
        I2C_Write(INT_ENABLE);      /* Write to interrupt enable register */
        I2C_Write(0x01);
        I2C_Stop();
}
void MPU_Start_Loc()
{
        I2C_Start_Wait(0xD0);       /* I2C start with device write address */
        I2C_Write(ACCEL_XOUT_H);/* Write start location address from where to read */
        I2C_Repeated_Start(0xD1);/* I2C start with device read address */
}
void Read_RawValue()
{
        MPU_Start_Loc();                                                            /*
Read Gyro values */
        Acc_x = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Acc_y = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Acc_z = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Temperature = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Gyro_x = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Gyro_y = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Ack());
        Gyro_z = (((int)I2C_Read_Ack()<<8) | (int)I2C_Read_Nack());
        I2C_Stop();
}
int main()
{
        DDRD=0x03;
        PORTD=0x03;
        char output[100];
        char floatx[10],floaty[10],floatz[10];
        float Xa,Ya,Za;
        I2C_Init();          /* Initialize I2C */
        Gyro_Init();         /* Initialize Gyro */
        USART_init(UBRR_9600);      /* Initialize USART with 9600 baud rate */
```

```
       USART_SendString("AT\r\n");
       _delay_ms(1000);
       USART_SendString("AT+CWMODE=3\r\n");
       _delay_ms(1000);
       USART_SendString("AT+CWJAP=\"SSID\",\"password\"\r\n"); //connects to network

       _delay_ms(1000);
       while(1)
       {
              Read_RawValue();

              /* Divide raw value by sensitivity scale factor */
              Xa = Acc_x/16384.0;
              Ya = Acc_y/16384.0;
              Za = Acc_z/16384.0;
              dtostrf( Xa, 3, 2, floatx );
              dtostrf( Ya, 3, 2, floaty );
              dtostrf( Za, 3, 2, floatz );

              USART_SendString("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80\r\n");
//starts session

              _delay_ms(1000);
              USART_SendString("AT+CIPSEND=120\r\n"); //prepares to send data
              _delay_ms(1000);
              snprintf(output,sizeof(output),"GET
https://api.thingspeak.com/update?key=1ZGZ1P4HHEO19YA2&field1=%s&field2=%s&field3=%s\r\n"
,floatx,floaty,floatz);
              USART_SendString(output);//send temp value
              _delay_ms(1000);
              USART_SendString("AT+CIPCLOSE\r\n");
              _delay_ms(1000);
              /*Xg = Gyro_x/16.4;
              Yg = Gyro_y/16.4;
              Zg = Gyro_z/16.4;

              t = (Temperature/340.00)+36.53;
              //Acc data
              dtostrf( Xa, 3, 2, float_ );
              sprintf(buffer," Ax = %s g\t",float_);
              USART_SendString(buffer);

              dtostrf( Ya, 3, 2, float_ );
              sprintf(buffer," Ay = %s g\t",float_);
              USART_SendString(buffer);

              dtostrf( Za, 3, 2, float_ );
              sprintf(buffer," Az = %s g\t",float_);
              USART_SendString(buffer);
*/


       }
}
```

3. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)



mEDBG Virtual COM Port (COM4)

☑ DTR ☐ RTS
☑ Open Terminal
☐ Autodetect protocols

Baud rate 9600   Parity None   Stop bits 1 bit

```
AT+CIPCLOSE
AT+CIPSTART="TCP","api.thingspeak.com",80
AT+CIPSEND=120
GET https://api.thingspeak.com/update?
key=1ZGZ1P4HHEO19YA2&field1=1.01&field2=0.01&field3=-0.23
AT+CIPCLOSE
AT+CIPSTART="TCP","api.thingspeak.com",80
AT+CIPSEND=120
GET https://api.thingspeak.com/update?
key=1ZGZ1P4HHEO19YA2&field1=1.01&field2=0.01&field3=-0.24
AT+CIPCLOSE
AT+CIPSTART="TCP","api.thingspeak.com",80
AT+CIPSEND=120
GET https://api.thingspeak.com/update?
key=1ZGZ1P4HHEO19YA2&field1=1.01&field2=0.01&field3=-0.22
AT+CIPCLOSE
AT+CIPSTART="TCP","api.thingspeak.com",80
```

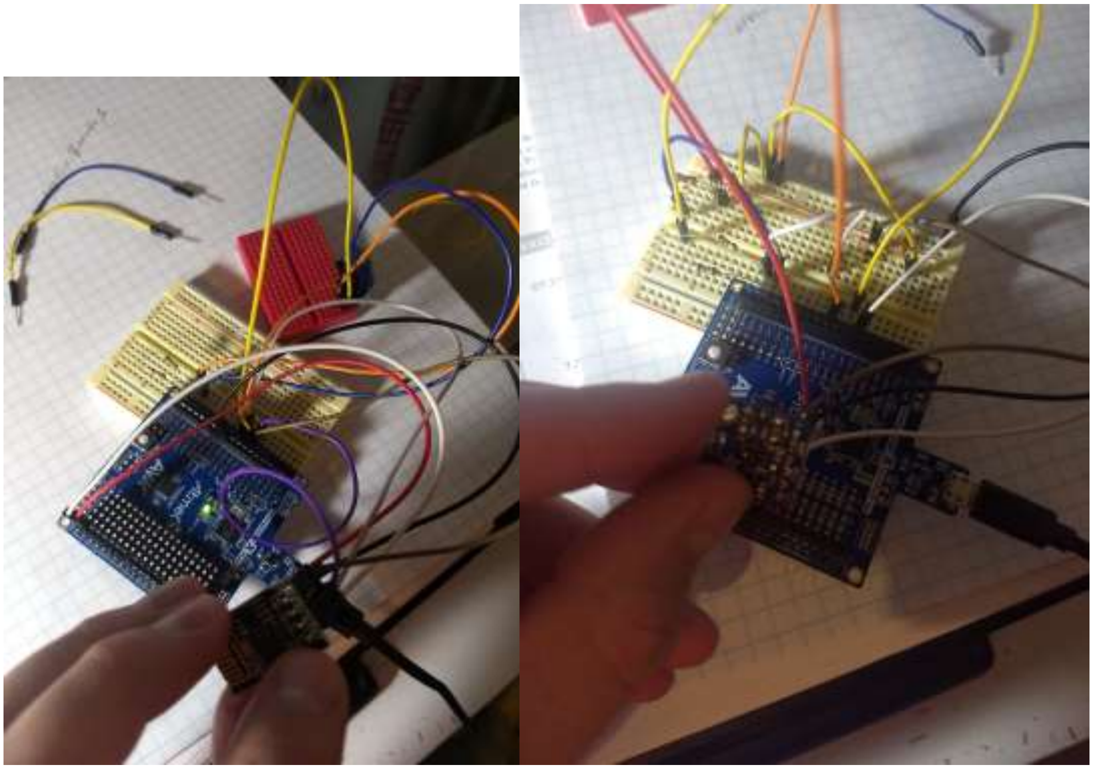
```
22:39:04.240: EDBG Control Panel added.
22:39:17.574: EDBG Control Panel closed.
```

00:00:32.4452410

☐ Add \r\n ☐ Hexadecimal Values ☐ Show Timestamp ☑ Automatically Scroll to End

4. **The numbers in the fields are the X,Y,Z accelerations**

5. SCREENSHOT OF EACH DEMO (BOARD SETUP)



6. GITHUB LINK OF THIS DA

https://github.com/westbrian2/Spring2019/tree/master/Midterms/Midterm2

Student Academic Misconduct Policy

http://studentconduct.unlv.edu/misconduct/policy.html

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

Brian West