CPE301 – Fall 2019

Final Project

Student Name: Jacob Patrick Reed

Student #: 1008448895

Student Email: reedj35@unlv.nevada.edu

Primary Github address: <https://github.com/reedjacobp>

Directory: <https://github.com/reedjacobp/submission_da>

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/DA, 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).

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

ATmega328PB Xplained Mini

ESP Module

APDS9960

1. **DEVELOPED CODE OF TASK 1**

/\*

\* FinalProject.c

\*

\* Created: 12/8/2019 12:32:56 AM

\* Author : jreed

\*/

#ifndef F\_CPU

#define F\_CPU 16000000UL

#endif

// Global constants for uart

#define BAUD 115200

#define FOSC 16000000

#define UBRR FOSC/8/BAUD-1

#define APDS9960\_WRITE 0x72

#define APDS9960\_READ 0x73

//include standard libraries

#include <avr/io.h>

#include <stdlib.h>

#include <stdio.h>

#include <util/delay.h>

#include <math.h>

//include custom libraries

#include "APDS9960\_def.h"

#include "i2c\_master.h"

//Function declarations

void getValues(void);

// void TIMER1\_init();

void init\_APDS9960(void);

void usart\_init();

void USART\_putstring(volatile unsigned char \*StringPtr);

//AT commands

volatile unsigned char AT[] = "AT\r\n"; // Test

volatile unsigned char CWMODE[] = "AT+CWMODE=3\r\n"; // Set Wi-Fi mode

volatile unsigned char CWJAP[] = "AT+CWJAP=\"blahblah\",\"blahblah\"\r\n"; // Get Wi-Fi info

volatile unsigned char CIPSTART[] = "AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\r\n"; // Establish connection with ThingSpeak

volatile unsigned char CIPSEND[] = "AT+CIPSEND=104\r\n"; // Set send function to 104

volatile unsigned char CIPMUX[] = "AT+CIPMUX=0\r\n"; // Enable connection

volatile unsigned char SEND\_DATA[] = "GET /update?key=IW250NQFTF4HL1KZ&field1="; // Get Write Key

volatile unsigned char RESET[] = "AT+RST\r\n"; // Get AT Firmware info

volatile unsigned char LINEBREAK[] = "\r\n"; // end of temperature transmission

volatile unsigned char CLOSE[] = "AT+CIPCLOSE\r\n";

//string for colors

volatile unsigned char RedStr[10];

volatile unsigned char GreenStr[10];

volatile unsigned char BlueStr[10];

*uint16\_t* redVal, greenVal, blueVal;

int main(void){

i2c\_init();

usart\_init(115200);

init\_APDS9960();

//Start up Esp

//Start AT communication

*\_delay\_ms*(10);

USART\_putstring(AT); //send AT to the USART

//connect to network

*\_delay\_ms*(10);

USART\_putstring(RESET); //reset ESP

*\_delay\_ms*(10);

USART\_putstring(AT); //confirm communication

*\_delay\_ms*(10);

USART\_putstring(CWMODE); //WiFi mode = 3

*\_delay\_ms*(10);

USART\_putstring(CWJAP); //Send wifi login

while(1){

//getValues();

*\_delay\_ms*(10);

USART\_putstring(CIPMUX); //Single connection point

*\_delay\_ms*(10);

USART\_putstring(CIPSTART); // Connect to ThingSpeak

*\_delay\_ms*(10);

USART\_putstring(CIPSEND); // Declare send length 50

*\_delay\_ms*(10);

getValues();

USART\_putstring(SEND\_DATA); // Connect to proper key

USART\_putstring(RedStr); // Send adc data

USART\_putstring("&field2=");

USART\_putstring(GreenStr); // Send adc data

USART\_putstring("&field3=");

USART\_putstring(BlueStr); // Send adc data

*\_delay\_ms*(1000);

}

return 0;

}

void init\_APDS9960(void){

*uint8\_t* setup;

i2c\_readReg(APDS9960\_WRITE, APDS9960\_ID, &setup,1);

if(setup != APDS9960\_ID\_1) while(1);

setup = 1 << 1 | 1<<0 | 1<<3 | 1<<4;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_ENABLE, &setup, 1);

setup = DEFAULT\_ATIME;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_ATIME, &setup, 1);

setup = DEFAULT\_WTIME;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_WTIME, &setup, 1);

setup = DEFAULT\_PROX\_PPULSE;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_PPULSE, &setup, 1);

setup = DEFAULT\_POFFSET\_UR;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_POFFSET\_UR, &setup, 1);

setup = DEFAULT\_POFFSET\_DL;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_POFFSET\_DL, &setup, 1);

setup = DEFAULT\_CONFIG1;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_CONFIG1, &setup, 1);

setup = DEFAULT\_PERS;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_PERS, &setup, 1);

setup = DEFAULT\_CONFIG2;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_CONFIG2, &setup, 1);

setup = DEFAULT\_CONFIG3;

i2c\_writeReg(APDS9960\_WRITE, APDS9960\_CONFIG3, &setup, 1);

}

void getValues(void){

*uint8\_t* redVH, redVL;

*uint8\_t* greenVH, greenVL;

*uint8\_t* blueVH, blueVL;

unsigned char i;

char dummy[10];

// Read red value

i2c\_readReg(APDS9960\_WRITE, APDS9960\_RDATAH, &redVH, 1);

i2c\_readReg(APDS9960\_WRITE, APDS9960\_RDATAL, &redVL, 1);

// Read green value

i2c\_readReg(APDS9960\_WRITE, APDS9960\_GDATAH, &greenVH, 1);

i2c\_readReg(APDS9960\_WRITE, APDS9960\_GDATAL, &greenVL, 1);

// Read blue value

i2c\_readReg(APDS9960\_WRITE, APDS9960\_BDATAH, &blueVH, 1);

i2c\_readReg(APDS9960\_WRITE, APDS9960\_BDATAL, &blueVL, 1);

redVal = (redVH << 8) | redVL;

greenVal = (greenVH << 8) | greenVL;

blueVal = (blueVH << 8) | blueVL;

// Set max threshold values

if (redVal > 255){

redVal = 255;

}

if (greenVal > 255){

greenVal = 255;

}

if (blueVal > 255){

blueVal = 255;

}

*itoa*(redVal, dummy, 10); //convert char to ascii

for(i = 0 ; i < 10 ; i++){

RedStr[i] = dummy[i]; //move converted ascii

}

*itoa*(greenVal, dummy, 10); //convert char to ascii

for(i = 0 ; i < 10 ; i++){

GreenStr[i] = dummy[i]; //move converted ascii

}

*itoa*(blueVal, dummy, 10); //convert char to ascii

for(i = 0 ; i < 10 ; i++){

BlueStr[i] = dummy[i]; //move converted ascii

}

}

void usart\_init() {

UBRR0H = ((UBRR) >> 8);

UBRR0L = UBRR;

UCSR0A |= (1<< U2X0); // divisor baud = 8

UCSR0B |= (1 << TXEN0); // Enable transmission

UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // 8 bits

}

void USART\_putstring(volatile unsigned char \*StringPtr)

{

while ((\*StringPtr != '\0')){

while (!(UCSR0A & (1 << UDRE0)));

UDR0 = \*StringPtr;

StringPtr++;

}

}

1. **INCLUDE FILES**

uart.h

/\*

\* uart.h

\*

\* Created: 12/8/2019 12:55:25 AM

\* Author: jreed

\*/

#ifndef USART\_RS232\_H\_FILE\_H\_ /\* Define library H file if not defined \*/

#define USART\_RS232\_H\_FILE\_H\_

#define F\_CPU 16000000UL /\* Define CPU clock Frequency e.g. here its 8MHz \*/

#include <avr/io.h> /\* Include AVR std. library file \*/

#define BAUD\_PRESCALE (((F\_CPU / (BAUDRATE \* 16UL))) - 1) /\* Define prescale value \*/

void USART\_Init(unsigned long); /\* USART initialize function \*/

char USART\_RxChar(); /\* Data receiving function \*/

void USART\_TxChar(char); /\* Data transmitting function \*/

void USART\_SendString(char\*); /\* Send string of USART data function \*/

#endif

uart.c

/\*

\* uart.c

\*

\* Created: 12/8/2019 12:54:27 AM

\* Author: jreed

\*/

#include "uart.h" /\* Include USART header file \*/

void USART\_Init(unsigned long BAUDRATE) /\* USART initialize function \*/

{

UCSR0B |= (1 << RXEN0) | (1 << TXEN0); /\* Enable USART transmitter and receiver \*/

UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01); /\* Write USCRC for 8 bit data and 1 stop bit \*/

UBRR0L = BAUD\_PRESCALE; /\* Load UBRRL with lower 8 bit of prescale value \*/

UBRR0H = (BAUD\_PRESCALE >> 8); /\* Load UBRRH with upper 8 bit of prescale value \*/

}

char USART\_RxChar() /\* Data receiving function \*/

{

while (!(UCSR0A & (1 << RXC0))); /\* Wait until new data receive \*/

return(UDR0); /\* Get and return received data \*/

}

void USART\_TxChar(char data) /\* Data transmitting function \*/

{

UDR0 = data; /\* Write data to be transmitting in UDR \*/

while (!(UCSR0A & (1<<UDRE0))); /\* Wait until data transmit and buffer get empty \*/

}

void USART\_SendString(char \*str) /\* Send string of USART data function \*/

{

int i=0;

while (str[i]!=0)

{

USART\_TxChar(str[i]); /\* Send each char of string till the NULL \*/

i++;

}

}

i2c\_master.h

/\*

\* i2c\_master.h

\*

\* Created: 12/8/2019 12:59:39 AM

\* Author: jreed

\*/

#ifndef I2C\_MASTER\_H

#define I2C\_MASTER\_H

#define I2C\_READ 0x01

#define I2C\_WRITE 0x00

void i2c\_init(void);

*uint8\_t* i2c\_start(*uint8\_t* address);

*uint8\_t* i2c\_write(*uint8\_t* data);

*uint8\_t* i2c\_read\_ack(void);

*uint8\_t* i2c\_read\_nack(void);

*uint8\_t* i2c\_transmit(*uint8\_t* address, *uint8\_t*\* data, *uint16\_t* length);

*uint8\_t* i2c\_receive(*uint8\_t* address, *uint8\_t*\* data, *uint16\_t* length);

*uint8\_t* i2c\_writeReg(*uint8\_t* devaddr, *uint8\_t* regaddr, *uint8\_t*\* data, *uint16\_t* length);

*uint8\_t* i2c\_readReg(*uint8\_t* devaddr, *uint8\_t* regaddr, *uint8\_t*\* data, *uint16\_t* length);

void i2c\_stop(void);

#endif // I2C\_MASTER\_H

i2c\_master.c

/\*

\* i2c\_master.c

\*

\* Created: 12/8/2019 12:58:22 AM

\* Author: jreed

\*/

#ifndef F\_CPU

#define F\_CPU 16000000UL

#endif

#include <avr/io.h>

#include <util/twi.h>

#include "i2c\_master.h"

#define F\_SCL 100000UL // SCL frequency

#define Prescaler 1

#define TWBR\_val ((((F\_CPU / F\_SCL) / Prescaler) - 16 ) / 2)

#define *TW\_STATUS* (TWSR0 & *TW\_STATUS\_MASK*)

void i2c\_init(void)

{

TWBR0 = (*uint8\_t*)TWBR\_val;

}

*uint8\_t* i2c\_start(*uint8\_t* address)

{

// reset TWI control register

TWCR0 = 0;

// transmit START condition

TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

// check if the start condition was successfully transmitted

if((TWSR0 & 0xF8) != *TW\_START*){ return 1; }

// load slave address into data register

TWDR0 = address;

// start transmission of address

TWCR0 = (1<<TWINT) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (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

TWDR0 = data;

// start transmission of data

TWCR0 = (1<<TWINT) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

if( (TWSR0 & 0xF8) != *TW\_MT\_DATA\_ACK* ){ return 1; }

return 0;

}

*uint8\_t* i2c\_read\_ack(void)

{

// start TWI module and acknowledge data after reception

TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

// return received data from TWDR

return TWDR0;

}

*uint8\_t* i2c\_read\_nack(void)

{

// start receiving without acknowledging reception

TWCR0 = (1<<TWINT) | (1<<TWEN);

// wait for end of transmission

while( !(TWCR0 & (1<<TWINT)) );

// return received data from TWDR

return TWDR0;

}

*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 i2c\_stop(void)

{

// transmit STOP condition

TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

}

APDS9960\_def.h

/\*

\* APDS9960\_def.h

\*

\* Created: 12/8/2019 2:04:11 AM

\* Author: jreed

\*/

#ifndef SparkFun\_APDS9960\_H

#define SparkFun\_APDS9960\_H

/\* APDS-9960 I2C address \*/

#define APDS9960\_I2C\_ADDR 0x39

/\* 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

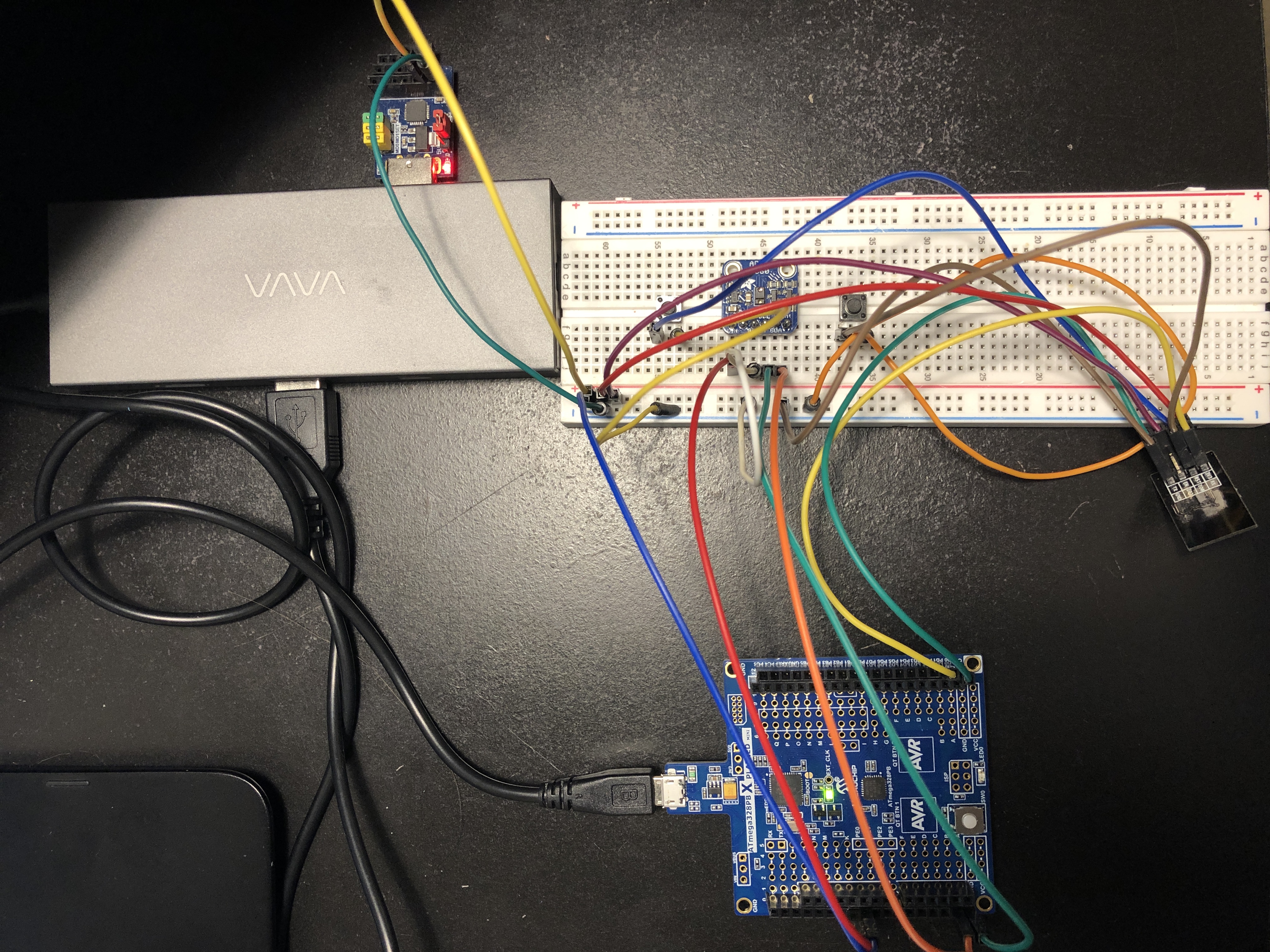
#define DEFAULT\_GIEN 0 // Disable gesture interrupts

#endif

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**



1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**



1. **VIDEO LINKS OF EACH DEMO**

https://youtu.be/5QeY6dw3u6M

1. **GITHUB LINK OF THIS DA**

**Student Academic Misconduct Policy**

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

“This assignment submission is my own, original work”.

NAME OF THE STUDENT