CPE301 – SPRING 2019

MIDTERM 2

Student Name: Ali Aside

Student #: 5004200625

Student Email: aliasadivet@gmail.com

Primary Github address: https://github.com/aliasadivet/Submitions\_DA\_AliAsadi.git

Directory:

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

List of Components used:

1. APDS9960
2. Esp-01 Adapter Module 3.3v 5v
3. ESP8266 ESP-01 Serial WIFI Wireless Transceiver Module
4. Atmega 328p Xplained Mini board
5. Breadboard
6. 22G jumper wires

The following picture is the connections of APDS9960 sensor and Atmega328p Xplained mini board.

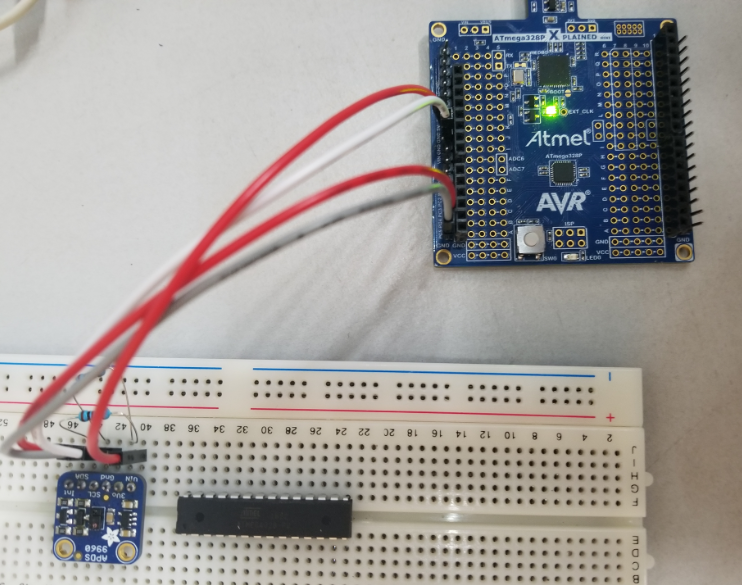
Vcc-Vcc

GND-GND

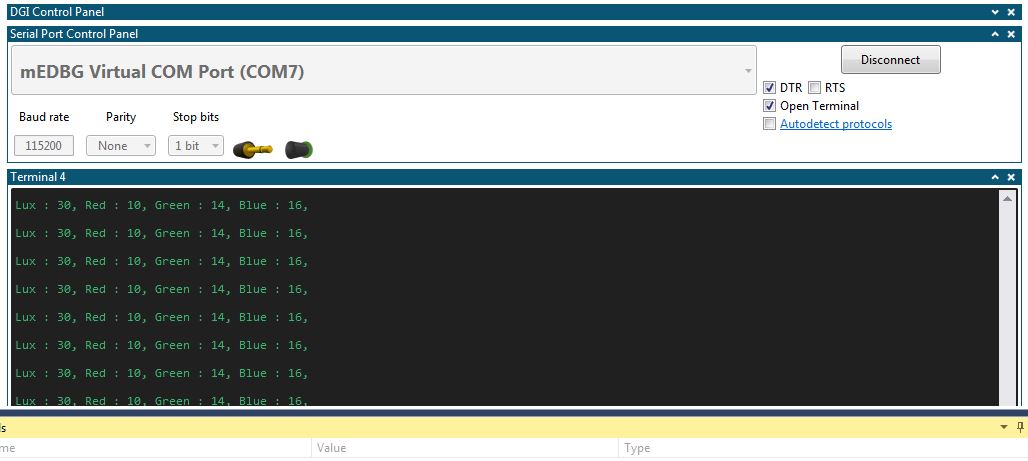
SDA-CS4

SCL-CS5

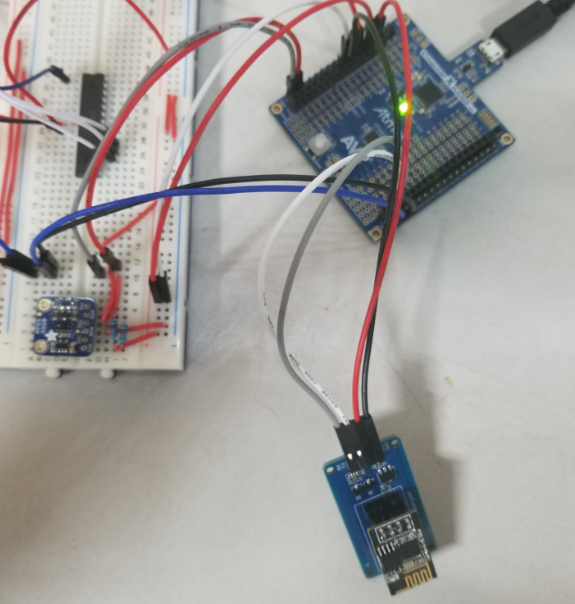
Pull-up resistors(10k) from SDA and ASL to 3V on the sensor



The following screenshot is the reading from the sensor. It demonstrates the intensity of light and three colors of red, green and blue respectively.



The following picture is the connections of APDS9960 sensor, Esp-01 Adapter, ESP8266 module and Atmega328p Xplained mini board.



APDS9960-Atmega328p board:

Vcc-Vcc

GND-GND

SDA-CS4

SCL-CS5

Pull-up resistors(10k) from SDA and ASL to 3V on the sensor

ESP8266- Atmega328p board:

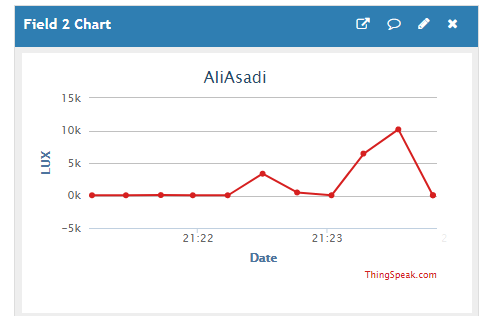
Vcc-Vcc

GND-GND

RX-TX

TX-RX

The following screenshot is the graph of Lux sensor values in Thingspeak.



**Codes:**

**//main.c**

/\*

\* MT2.c

\*

\* Created: 5/13/2019 11:16:10 AM

\* Author : Ali Asadi

\*/

#include <stdlib.h>

#include <stdio.h>

#include <inttypes.h>

#include <avr/io.h>

#include <avr/interrupt.h>

#include <avr/sleep.h>

#include <String.h>

#include <util/delay.h>

#include "i2c.h"

#include "APDS9960\_def.h"

#define APDS9960\_WRITE (APDS9960\_I2C\_ADDR<<1)

#define APDS9960\_READ ((APDS9960\_I2C\_ADDR<<1)|0x01)

void USART\_init();

void USART\_send(char );

void sendString(const char \*);

void init\_APDS9960();

*uint16\_t* getreading(*uint8\_t*);

void ESP\_init();

void ESP\_sendData(int);

char receiveString(char\* ,const char\* );

char buffer[128];

const char ssid[] = "\"D798BE\"", // The SSID (name) of the Wi-Fi network I connect to

pwd[] = "\"72C26B2B20500\"", // The password of the Wi-Fi network

ok[] = "\r\nOK\r\n", // OK respond after connection

sendOK[] = "\r\nSEND OK\r\n",

error[] = "ERROR",

apiKey[] = "1SFM68R6DZI4P8UG", // My channel's Write API Key

wifiMode[] = "AT+CWMODE\_CUR=3",

connectWiFi[] = "AT+CWJAP\_CUR=",

openTCP[] = "AT+CIPSTART=",

dataSend[] = "AT+CIPSEND=",

closeTCP[] = "AT+CIPCLOSE",

link[] = "GET /update?api\_key=%s&field2=%d HTTP/1.1\r\n"

"Host: api.thingspeak.com\r\n"

"Connection: close\r\n\r\n",

send[] = "\r\n";

int main()

{

*uint8\_t* status;

*uint16\_t* value;

USART\_init();

i2c\_init(); // initialize I2C module

ESP\_init(); // initialize ESP module

init\_APDS9960(); // initialize APDS9960 module

while (1)

{

i2c\_start(APDS9960\_WRITE); // I2C start Slave Address + Write mode

i2c\_write(APDS9960\_STATUS); // set pointer

i2c\_start(APDS9960\_READ); // I2C start Slave Address + Read mode

status = i2c\_read\_nack(); // read status byte

i2c\_stop();

if (status & 0x01) // check if Avalid Bit is Set

{

value = getreading(APDS9960\_CDATAL); // read the LUX value

ESP\_sendData(value) ; // sent the LUX value

}

else if ( status & 0x80) // check if saturation bit is set

{

i2c\_start(APDS9960\_READ);

i2c\_write(APDS9960\_CICLEAR); // clear all flags

i2c\_stop();

i2c\_start(APDS9960\_READ);

i2c\_write(APDS9960\_AICLEAR); // clear all flags

i2c\_stop();

}

*\_delay\_ms*(1000);

}

return 0;

}

void USART\_init()

{

UCSR0B = 0x18; // TX enable

UCSR0C = 0x06; // 8 bit data

UBRR0L = 8; // Baud Rate 115200

}

void init\_APDS9960(void) {

*\_delay\_ms*(150); /\* Power up time >100ms \*/

i2c\_start(APDS9960\_WRITE); // I2C start Slave Address + Write mode

i2c\_write(APDS9960\_ENABLE); // set Address

i2c\_write(APDS9960\_PON | APDS9960\_AEN | APDS9960\_WEN); // enable PON, AEN, WEN

i2c\_stop(); // I2C Stop

i2c\_start(APDS9960\_WRITE); // I2C start Slave Address + Write mode

i2c\_write(APDS9960\_ATIME); // set Address

i2c\_write(DEFAULT\_ATIME); // set adc integration time

i2c\_stop(); // I2C Stop

i2c\_start(APDS9960\_WRITE); // I2C start Slave Address + Write mode

i2c\_write(APDS9960\_WTIME); // set Address

i2c\_write(DEFAULT\_WTIME); // set wait time

i2c\_stop(); // I2C Stop

i2c\_start(APDS9960\_WRITE); // I2C start Slave Address + Write mode

i2c\_write(APDS9960\_CONFIG1); // set Address

i2c\_write(DEFAULT\_CONFIG1); // set Config

i2c\_stop(); // I2C Stop

}

*uint16\_t* getreading(*uint8\_t* Register) {

*uint16\_t* reading;

i2c\_start(APDS9960\_WRITE); // I2C start Slave Address + Write mode

i2c\_write(Register); // set pointer

i2c\_start(APDS9960\_READ); // I2C start Slave Address + Read mode

reading = ((int)i2c\_read\_ack() | ((int)i2c\_read\_nack() << 8)); // read both low byte and High byte of 16 bit value

i2c\_stop(); // I2C Stop

return reading; // return the reading

}

void USART\_send(char data)

{

while (!(UCSR0A & \_BV(UDRE0))); // check if the Data register is empty

UDR0 = data; // send the data

}

void sendString(const char \*str)

{

while (\*str) // send all the data till null Character

USART\_send(\*str++);

}

char USART\_Receive()

{

while (!(UCSR0A & \_BV(RXC0))); // wait for character in receiver

return UDR0;

}

void ESP\_init()

{

sendString(wifiMode); // set wifimode = station + softAP

sendString(send);

if(!receiveString(buffer, ok))

{

return;

}

*strcpy*(buffer,connectWiFi); // connect to the router

*strcat*(buffer,ssid); // ssid

*strcat*(buffer,",");

*strcat*(buffer,pwd); // password for the router

*strcat*(buffer,send);

sendString(buffer); // connect

if(!receiveString(buffer, ok))

{

return;

}

}

void ESP\_sendData(int temp)

{

*strcpy*(buffer,openTCP); // open TCP connection

*strcat*(buffer,"\"TCP\""); // type TCP

*strcat*(buffer,",");

*strcat*(buffer,"\"api.thingspeak.com\""); // host

*strcat*(buffer,",");

*strcat*(buffer,"80"); // port

*strcat*(buffer,send);

sendString(buffer); // send it to ESP

*memset*(buffer,0,128);

if(!receiveString(buffer, ok))

{

return;

}

*sprintf*(buffer,link,apiKey,temp);

char tmp[4];

*itoa*(*strlen*(buffer), tmp, 10);

*strcpy*(buffer,dataSend); // data send cmd

*strcat*(buffer,tmp); // number of data to be send

*strcat*(buffer,send);

sendString(buffer); // send to ESP

*memset*(buffer,0,128);

if(!receiveString(buffer, ok))

{

return;

}

*sprintf*(buffer,link,apiKey,temp); // send HTTP text to the ESP

*strcat*(buffer,send);

sendString(buffer);

*memset*(buffer,0,128);

if(!receiveString(buffer, sendOK))

{

return;

}

}

char receiveString(char\* buffer,const char\* limit)

{

char \*ptr = buffer;

while(1)

{

\*ptr++ = USART\_Receive(); // store the received character in buffer

if(*strstr*(buffer, limit)) // check for ok

return 1;

if(*strstr*(buffer, error)) // check for error

return 0;

}

}

**//i2c.h**

#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);

void i2c\_stop(void);

#endif // I2C\_MASTER\_H

**//i2c.c**

#include <avr/io.h>

#include <util/twi.h>

#include "i2c.h"

#define F\_SCL 100000UL // SCL frequency

#define Prescaler 1

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

void i2c\_init(void)

{

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

}

*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 = TWSR & 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)) );

// check if data is acknowledged

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;

}

void i2c\_stop(void)

{

// transmit STOP condition

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

}

**//APDS9960\_def.h**

/\*\*

\* @file SparkFun\_APDS-9960.h

\* @brief Library for the SparkFun APDS-9960 breakout board

\* @author Shawn Hymel (SparkFun Electronics)

\*

\* @copyright This code is public domain but you buy me a beer if you use

\* this and we meet someday (Beerware license).

\*

\* This library interfaces the Avago APDS-9960 to Arduino over I2C. The library

\* relies on the Arduino Wire (I2C) library. to use the library, instantiate an

\* APDS9960 object, call init(), and call the appropriate functions.

\*/

#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

The executions can be found in the linkes below:

<https://youtu.be/g_29rlQqzrw>

<https://youtu.be/TqGbmpHdRyQ>

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

Ali Asadi