CPE301 – SPRING 2019

MIDTERM 2 (Final)

Student Name: Mohamad Jundi

Student #: 8000321867

Student Email: jundi@unlv.nevada.edu

Primary Github address: https://github.com/MohamedJundi1994/Submission\_Midterm2-Final-

Directory:

C:\Users\moham\Documents\School\CPE 301\Repository\CPE\_301\Midterms\Midterm 2 (Final)

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

APDS 9960:

USB port => Xplained Mini => 3V => 3oV

=> GND => GND

=> PC4 => SVA

=> PC5 => SCL

ESP32:

USB port => Xplained Mini => 3V => 3V3

=> PD1 (Tx) => Rx

=> PD0 (Rx) => Tx

=> GND => GND

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

Code:

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include "i2c\_master.h"

#include "9960.h"

#define BAUD 9600

#define BAUDR (*F\_CPU*/16/BAUD) - 1

void UART\_initialization(); // Function declarations

void Colors(); //

void sen9960\_initialization(); //

int sending\_char( char info, *FILE* \* str); //

*uint16\_t* baud\_rate = BAUDR; // Int declaration

*uint8\_t* red\_low, red\_high; //

*uint8\_t* blue\_low, blue\_high; //

*uint8\_t* green\_low, green\_high; //

*uint8\_t* config; //

*FILE* UART\_str = *FDEV\_SETUP\_STREAM*(sending\_char, *NULL* , *\_FDEV\_SETUP\_WRITE*);

char results[256]; // Array results

int main(void)

{

*uint16\_t* red = 0; // Declare red

*uint16\_t* blue = 0; // Declare blue

*uint16\_t* green = 0; // Declare green

i2c\_init(); // i2C function call

UART\_initialization(); // uart initialization function call

*stdout* = &UART\_str; // standard stream

sen9960\_initialization(); // initialize 9960 sensor function call

*\_delay\_ms*(2000); // Give time of 2s

*printf*("AT\r\n"); // AT cmd

*\_delay\_ms*(3000); // Give time of 3s

*printf*("AT+CWMODE=1\r\n"); // AT cmd mode

*\_delay\_ms*(3000); // Give time of 3s

*printf*("AT+CWJAP=\"MOE\_Wifi\",\"moo123456\"\r\n"); // AT cmd to connect to personal hotspot information

while (1)

{

*\_delay\_ms*(1500); // Give time of 1.5s

*printf*("AT+CIPMUX=0\r\n"); // AT cmd for CIPMUX

*\_delay\_ms*(1500); // Give time of 1.5s

*printf*("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80\r\n"); // AT cmd to communicate to thingspeak

*\_delay\_ms*(1500); // Give time of 1.5s

Colors(&red, &blue, &green); // Call colors function

*printf*("AT+CIPSEND=100\r\n"); // AT cmd to send 100 chars

*printf*("GET https://api.thingspeak.com/update?api\_key=VBIZQ9TJL06VKSR8&field1=%05u\r\n", red); // Send red value to thingspeak

*\_delay\_ms*(1500); // Give time of 1.5s

*printf*("AT+CIPMUX=0\r\n"); // AT cmd for CIPMUX

*\_delay\_ms*(1500); // Give time of 1.5s

*printf*("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80\r\n"); // AT cmd to communicate to thingspeak

*\_delay\_ms*(1500); // Give time of 1.5s

Colors(&red, &blue, &green); // Call colors function

*printf*("AT+CIPSEND=100\r\n"); // AT cmd to send 100 chars

*printf*("GET https://api.thingspeak.com/update?api\_key=VBIZQ9TJL06VKSR8&field2=%05u\r\n", blue); // Send blue value to thingspeak

*\_delay\_ms*(1500); // Give time of 1.5s

*printf*("AT+CIPMUX=0\r\n"); // AT cmd for CIPMUX

*\_delay\_ms*(1500); // Give time of 1.5s

*printf*("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80\r\n"); // AT cmd to communicate to thingspeak

*\_delay\_ms*(1500); // Give time of 1.5s

Colors(&red, &blue, &green); // Call colors function

*printf*("AT+CIPSEND=100\r\n"); // AT cmd to send 100 chars

*printf*("GET https://api.thingspeak.com/update?api\_key=VBIZQ9TJL06VKSR8&field3=%05u\r\n", green); // Send green value to thingspeak

*\_delay\_ms*(1500); // Give time of 1.5s

}

}

void UART\_initialization(void)

{

UBRR0H = baud\_rate >> 8; // Setting the baud rate

UBRR0L = baud\_rate & 0xFF; //

UCSR0B = ( 1 << RXEN0) | ( 1 << TXEN0) | (1 << RXCIE0); // Receive and transmit are enabled

UCSR0C = (3 << UCSZ00); // Set frame format: 8data, 1stop bit

}

int sending\_char(char info, *FILE* \*str)

{

while ( !( UCSR0A & ( 1 << UDRE0)) ); // Wait until buffer empty

UDR0 = info; // Data placed into buffer

return 0;

}

void Colors(*uint16\_t* \*red, *uint16\_t* \*blue, *uint16\_t* \*green) // Function used to read the colors for sensor (red, blue and green)

{

i2c\_readReg(APDS\_WRITE, APDS9960\_RDATAL, &red\_low, 1);

i2c\_readReg(APDS\_WRITE, APDS9960\_RDATAH, &red\_high, 1);

i2c\_readReg(APDS\_WRITE, APDS9960\_GDATAL, &green\_low, 1);

i2c\_readReg(APDS\_WRITE, APDS9960\_GDATAH, &green\_high, 1);

i2c\_readReg(APDS\_WRITE, APDS9960\_BDATAL, &blue\_low, 1);

i2c\_readReg(APDS\_WRITE, APDS9960\_BDATAH, &blue\_high, 1);

\*red = red\_high << 8 | red\_low;

\*blue = blue\_high << 8 | blue\_low;

\*green = green\_high << 8 | green\_low;

}

void sen9960\_initialization() // Function used to initialize the Sensor only for the RGB, no gesture or proximity used

{

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

if(config != APDS9960\_ID\_1)

while(1)

{

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

}

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

config = DEFAULT\_ATIME;

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

config = DEFAULT\_WTIME;

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

config = DEFAULT\_PROX\_PPULSE;

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

config = DEFAULT\_POFFSET\_UR;

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

config = DEFAULT\_POFFSET\_DL;

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

config = DEFAULT\_CONFIG1;

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

config = DEFAULT\_PERS;

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

config = DEFAULT\_CONFIG2;

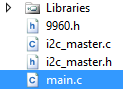
i2c\_writeReg(APDS\_WRITE, APDS9960\_CONFIG2, &config, 1);

config = DEFAULT\_CONFIG3;

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

}

Libraries used:



9960.h:

#ifndef APDS\_H

#define APDS\_H

#include <avr/io.h>

#include "i2c\_master.h"

#include "9960.h"

#define APDS\_WRITE (0x39 << 1) | 0

#define APDS\_READ (0x39 << 1) | 1

/\* Debug \*/

/\* APDS-9960 I2C address \*/

#define APDS9960\_I2C\_ADDR 0x39

/\* 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\_PERS 0x8C

#define APDS9960\_CONFIG1 0x8D

#define APDS9960\_PPULSE 0x8E

#define APDS9960\_CONFIG2 0x90

#define APDS9960\_ID 0x92

#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\_POFFSET\_UR 0x9D

#define APDS9960\_POFFSET\_DL 0x9E

#define APDS9960\_CONFIG3 0x9F

/\* 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 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

/\* LED Boost values \*/

#define LED\_BOOST\_100 0

#define LED\_BOOST\_150 1

#define LED\_BOOST\_200 2

#define LED\_BOOST\_300 3

/\* Default values \*/

#define DEFAULT\_ATIME 219 // 103ms

#define DEFAULT\_WTIME 246 // 27ms

#define DEFAULT\_PROX\_PPULSE 0x87 // 16us, 8 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\_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\_GLDRIVE LED\_DRIVE\_100MA

#define DEFAULT\_GWTIME GWTIME\_2\_8MS

void apds\_init();

/\*void readColor();\*/

#endif

I2c\_master.c:

#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)

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 = *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 i2c\_stop(void)

{

// transmit STOP condition

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

}

I2c\_master.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);

*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);

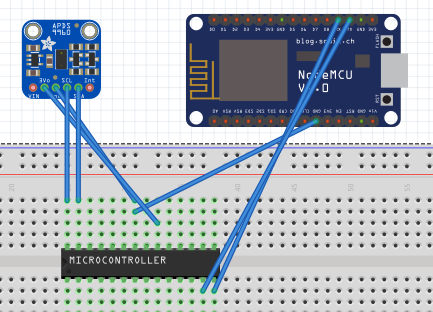
void i2c\_stop(void);

#endif // I2C\_MASTER\_H

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

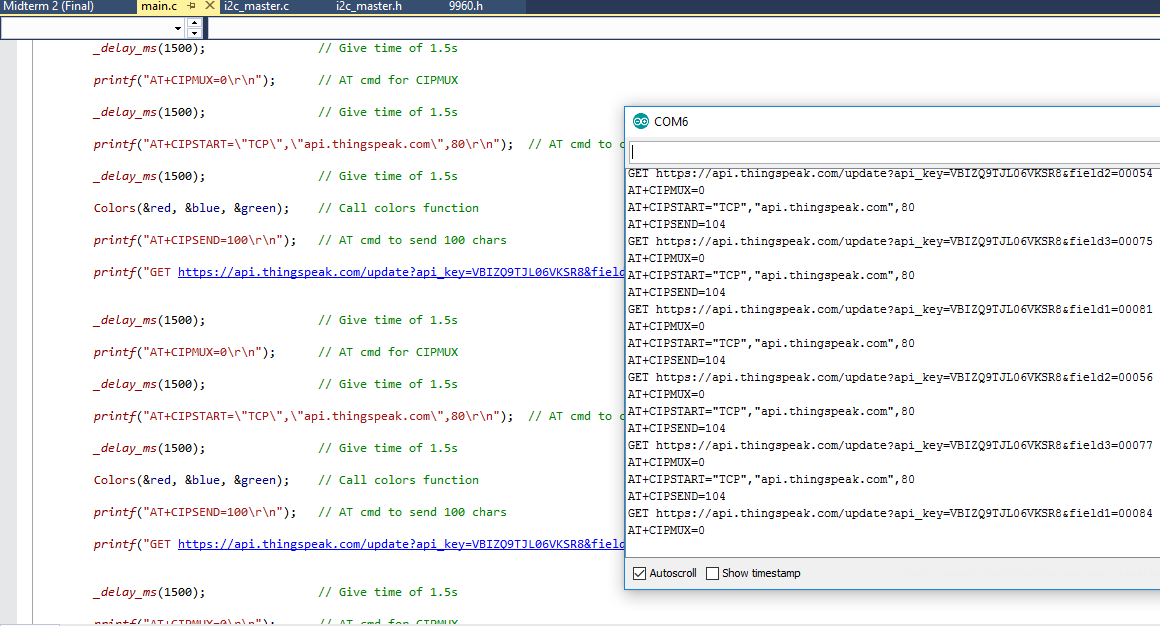
All code and libraries are in number 1.

1. **SCHEMATICS**

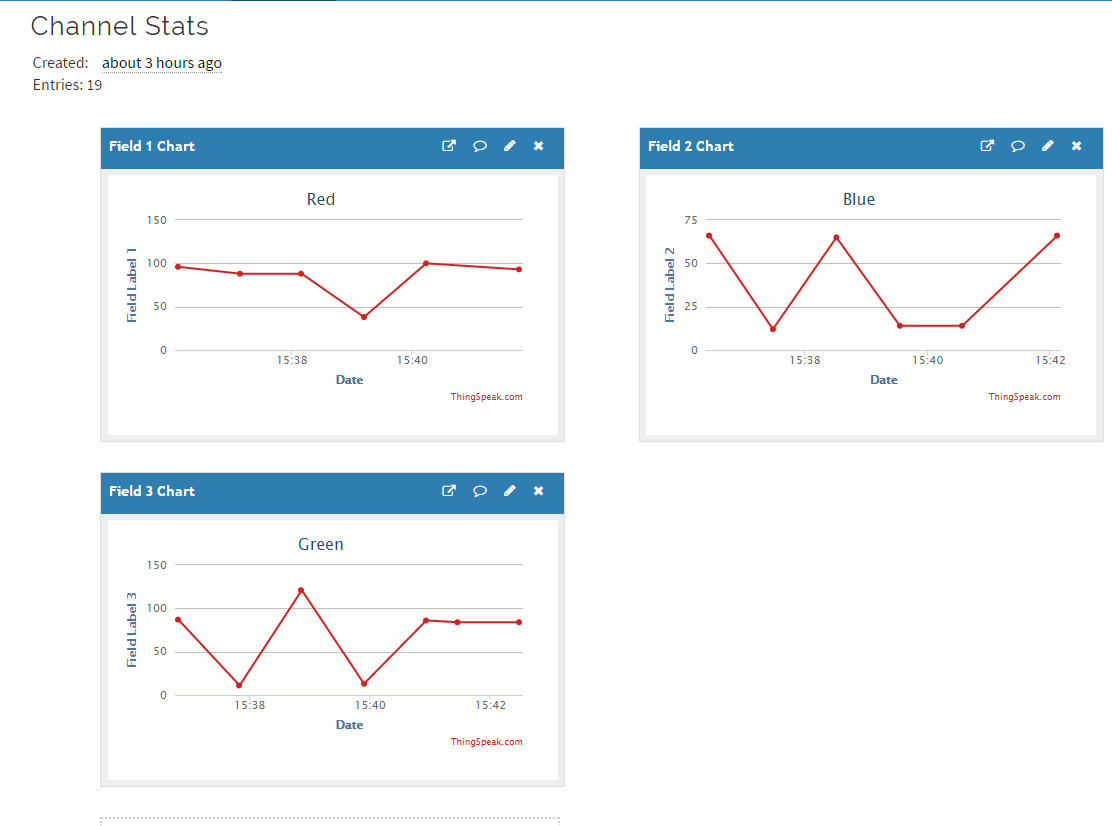


1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

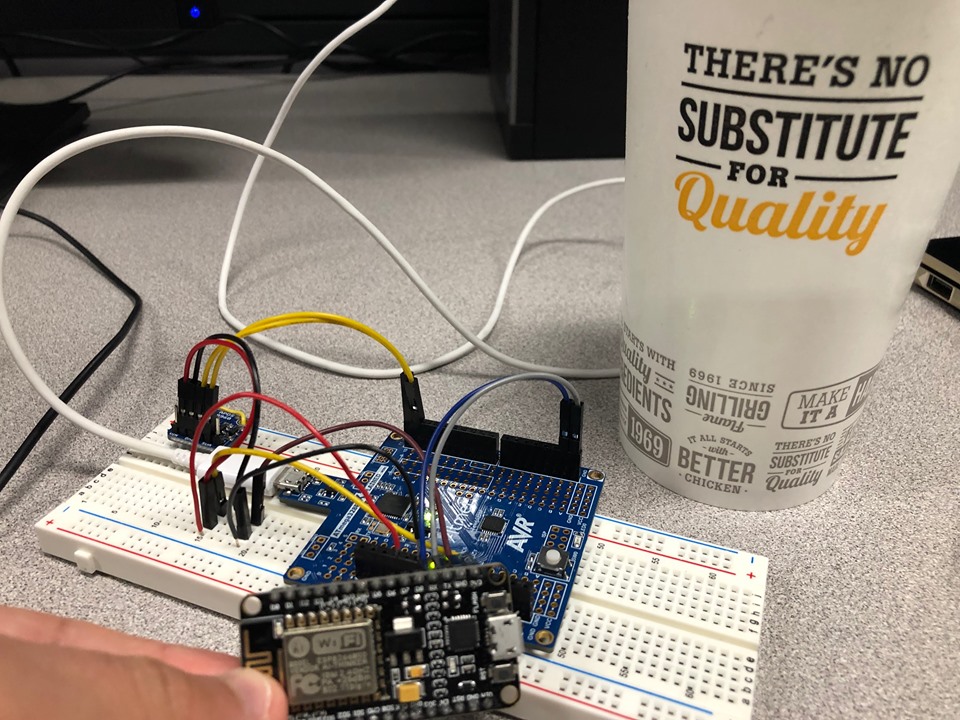
Atmel Studio Output and Arduino Serial Port Output:



Thingspeak charts of Red, Blue and Green:



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



1. **VIDEO LINKS OF EACH DEMO**

LINK: <https://www.youtube.com/watch?v=33WFjyIlcmw&feature=share&fbclid=IwAR0GkQkyGIh0dnR5kzf-DM9BQZzXoNfnIMcAjDZi6olfJkvblBmh6JaHNrU>

1. **GITHUB LINK OF THIS DA**

LINK: https://github.com/MohamedJundi1994/Submission\_Midterm2-Final-

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

MOHAMAD JUNDI