CPE301 – SPRING 2019

MIDTERM 2

Student Name: yeeunLee

Student #:2001614222

Student Email: leey81@nevada.unlv.com

Primary Github address: https://github.com/yeeun219/submission\_da.git

Directory: cpe301\DesignAssignments\MIDTEM2

1. **COMPONENTS LIST OF CODE TASK1/A – DISPLAY COLOR SENSOR**

List of Components used

Block diagram with pins used in the Atmega328P

void ADPS\_Start\_Loc() // Initialize the ADPS

void Read\_RawValue() //READ VALUE OF the ADPS

USART\_init(BAUD\_PRESCALLER); // Initialize the USART and initialize the pin number0

USART\_tx\_string("Connected!\r\n");/\* SEND A STRING TO THE RS-232 \*/

while(1) IN MAIN()//Repeat continuously, current COLOR DATA.

1. **CODE OF TASK 1/A**

/\*

\* color\_sensor.c

\*

\* Created: 5/15/2019 3:36:25 AM

\* Author : llje2

\*/

#define F\_CPU 16000000UL

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

#include <util/delay.h> /\* Include delay header file \*/

#include <inttypes.h> /\* Include integer type header file \*/

#include <stdlib.h> /\* Include standard library file \*/

#include <stdio.h> /\* Include standard I/O library file \*/

#include <stdbool.h> /\* Include standard boolean library \*/

#include "i2c\_master.h"

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

#include "APDS9960\_def.h"

#define BAUDRATE 9600

#define BAUD\_PRESCALLER (((F\_CPU / (BAUDRATE \* 16UL))) - 1)

#include <util/delay.h>

int r,g,b,c, tmp = -11;

void USART\_tx\_string( char \*data )

{

while ((\*data != '\0'))

{

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

UDR0 = \*data;

data++;

}

}

/\* INIT USART (RS-232) \*/

void USART\_init( unsigned int ubrr )

{

UBRR0H = (unsigned char)(ubrr>>8);

UBRR0L = (unsigned char)ubrr;

UCSR0B = (1 << TXEN0); // Enable RX, TX & RX interrupt

UCSR0C = (3 << UCSZ00); //asynchronous 8 N 1

}

void ADPS\_Start\_Loc()

{

i2c\_start(0x72); /\* I2C start with device write address \*/

i2c\_write(APDS9960\_CDATAL);/\* Write start location address from where to read \*/

i2c\_start(0x73);/\* I2C start with device read address \*/

}

void Read\_RawValue()

{

ADPS\_Start\_Loc(); /\* Read Gyro values \*/

c = (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

r = (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

g= (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

b= (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

i2c\_stop();

}

int main(void)

{

char buffer[20];

*uint8\_t* Connect\_Status;

i2c\_init(); /\* Initialize I2C \*/

ADPS\_Init(); /\* Initialize Gyro \*/

USART\_init(BAUD\_PRESCALLER); // Initialize the USART

USART\_tx\_string("Connected!\r\n"); // we're alive!

/\* Replace with your application code \*/

while (1)

{

char \_buffer[150];

Read\_RawValue();

*sprintf*(buffer," clear: %d \t",c);

USART\_tx\_string(buffer);

*sprintf*(buffer, " Red: %d \t",r);

USART\_tx\_string(buffer);

*sprintf*(buffer," Green = %d \t",g);

USART\_tx\_string(buffer);

*sprintf*(buffer," Blue = %d \t",b);

USART\_tx\_string(buffer);

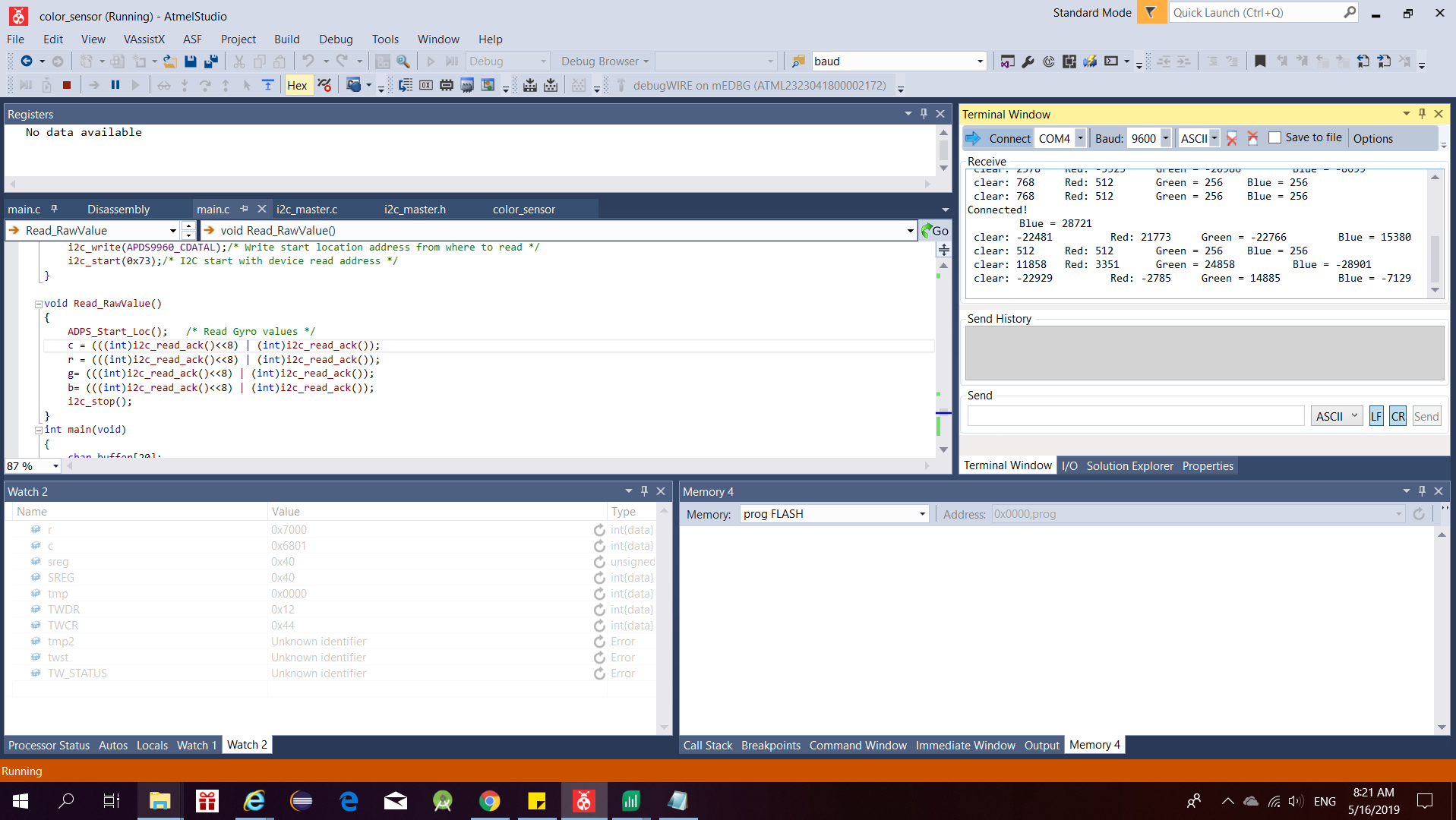
USART\_tx\_string("\n");

*\_delay\_ms*(5000);

}

}

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



1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A) – UPLOAD THE VALUE ON THE THHINGSPEAK**

List of Components used

Block diagram with pins used in the Atmega328P

void ADPS\_Start\_Loc() // Initialize the ADPS

void Read\_RawValue() //READ VALUE OF the ADPS

USART\_init(BAUD\_PRESCALLER); // Initialize the USART and initialize the pin number0

USART\_tx\_string("Connected!\r\n");/\* SEND A STRING TO THE RS-232 \*/

while(1) IN MAIN()//Repeat continuously, current COLOR DATA.

` // about the esp8266

ESP8266\_WIFIMode(BOTH\_STATION\_AND\_ACCESPOINT);/\* 3 = Both (AP and STA) \*/

ESP8266\_ConnectionMode(SINGLE); /\* 0 = Single; 1 = Multi \*/

ESP8266\_ApplicationMode(NORMAL);

ESP8266\_connected()

ESP8266\_JoinAccessPoint(SSID, PASSWORD);

ESP8266\_Start(0, DOMAIN, PORT);

ESP8266\_Send(\_buffer);

Read\_Data(\_buffer);

*memset*(\_buffer, 0, 150);

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

/\*

\* color\_sensor.c

\*

\* Created: 5/15/2019 3:36:25 AM

\* Author : llje2

\*/

#define F\_CPU 16000000UL

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

#include <util/delay.h> /\* Include delay header file \*/

#include <inttypes.h> /\* Include integer type header file \*/

#include <stdlib.h> /\* Include standard library file \*/

#include <stdio.h> /\* Include standard I/O library file \*/

#include <stdbool.h> /\* Include standard boolean library \*/

#include "i2c\_master.h"

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

#include "APDS9960\_def.h"

#define BAUDRATE 9600

#define BAUD\_PRESCALLER (((F\_CPU / (BAUDRATE \* 16UL))) - 1)

#include <util/delay.h>

int r,g,b,c, tmp = -11;

#define SREG \_SFR\_IO8(0x3F)

#define DEFAULT\_BUFFER\_SIZE 160

#define DEFAULT\_TIMEOUT 10000

/\* Connection Mode \*/

#define SINGLE 0

#define MULTIPLE 1

/\* Application Mode \*/

#define NORMAL 0

#define TRANSPERANT 1

/\* Application Mode \*/

#define STATION 1

#define ACCESSPOINT 2

#define BOTH\_STATION\_AND\_ACCESPOINT 3

/\* Select Demo \*/

#define RECEIVE\_DEMO /\* Define RECEIVE demo \*/

//#define SEND\_DEMO /\* Define SEND demo \*/

/\* Define Required fields shown below \*/

#define DOMAIN "api.thingspeak.com"

#define PORT "80"

#define API\_WRITE\_KEY "PWJ8GZ3QXFQ0MXUM"

#define CHANNEL\_ID "751037"

#define SSID "Leeyeeun"

#define PASSWORD "02191919"

enum ESP8266\_RESPONSE\_STATUS{

ESP8266\_RESPONSE\_WAITING,

ESP8266\_RESPONSE\_FINISHED,

ESP8266\_RESPONSE\_TIMEOUT,

ESP8266\_RESPONSE\_BUFFER\_FULL,

ESP8266\_RESPONSE\_STARTING,

ESP8266\_RESPONSE\_ERROR

};

enum ESP8266\_CONNECT\_STATUS {

ESP8266\_CONNECTED\_TO\_AP,

ESP8266\_CREATED\_TRANSMISSION,

ESP8266\_TRANSMISSION\_DISCONNECTED,

ESP8266\_NOT\_CONNECTED\_TO\_AP,

ESP8266\_CONNECT\_UNKNOWN\_ERROR

};

enum ESP8266\_JOINAP\_STATUS {

ESP8266\_WIFI\_CONNECTED,

ESP8266\_CONNECTION\_TIMEOUT,

ESP8266\_WRONG\_PASSWORD,

ESP8266\_NOT\_FOUND\_TARGET\_AP,

ESP8266\_CONNECTION\_FAILED,

ESP8266\_JOIN\_UNKNOWN\_ERROR

};

*int8\_t* Response\_Status;

volatile *int16\_t* Counter = 0, pointer = 0;

*uint32\_t* TimeOut = 0;

char RESPONSE\_BUFFER[DEFAULT\_BUFFER\_SIZE];

void Read\_Response(char\* \_Expected\_Response)

{

*uint8\_t* EXPECTED\_RESPONSE\_LENGTH = *strlen*(\_Expected\_Response);

*uint32\_t* TimeCount = 0, ResponseBufferLength;

char RECEIVED\_CRLF\_BUF[EXPECTED\_RESPONSE\_LENGTH];

while(1)

{

if(TimeCount >= (DEFAULT\_TIMEOUT+TimeOut))

{

TimeOut = 0;

Response\_Status = ESP8266\_RESPONSE\_TIMEOUT;

return;

}

if(Response\_Status == ESP8266\_RESPONSE\_STARTING)

{

Response\_Status = ESP8266\_RESPONSE\_WAITING;

}

ResponseBufferLength = *strlen*(RESPONSE\_BUFFER);

if (ResponseBufferLength)

{

*\_delay\_ms*(1);

TimeCount++;

if (ResponseBufferLength==*strlen*(RESPONSE\_BUFFER))

{

for (*uint16\_t* i=0;i<ResponseBufferLength;i++)

{

*memmove*(RECEIVED\_CRLF\_BUF, RECEIVED\_CRLF\_BUF + 1, EXPECTED\_RESPONSE\_LENGTH-1);

RECEIVED\_CRLF\_BUF[EXPECTED\_RESPONSE\_LENGTH-1] = RESPONSE\_BUFFER[i];

if(!*strncmp*(RECEIVED\_CRLF\_BUF, \_Expected\_Response, EXPECTED\_RESPONSE\_LENGTH))

{

TimeOut = 0;

Response\_Status = ESP8266\_RESPONSE\_FINISHED;

return;

}

}

}

}

*\_delay\_ms*(1);

TimeCount++;

}

}

void ESP8266\_Clear()

{

*memset*(RESPONSE\_BUFFER,0,DEFAULT\_BUFFER\_SIZE);

Counter = 0; pointer = 0;

}

void Start\_Read\_Response(char\* \_ExpectedResponse)

{

Response\_Status = ESP8266\_RESPONSE\_STARTING;

do {

Read\_Response(\_ExpectedResponse);

} while(Response\_Status == ESP8266\_RESPONSE\_WAITING);

}

void GetResponseBody(char\* Response, *uint16\_t* ResponseLength)

{

*uint16\_t* i = 12;

char buffer[5];

while(Response[i] != '\r')

++i;

*strncpy*(buffer, Response + 12, (i - 12));

ResponseLength = *atoi*(buffer);

i += 2;

*uint16\_t* tmp = *strlen*(Response) - i;

*memcpy*(Response, Response + i, tmp);

if(!*strncmp*(Response + tmp - 6, "\r\nOK\r\n", 6))

*memset*(Response + tmp - 6, 0, i + 6);

}

bool WaitForExpectedResponse(char\* ExpectedResponse)

{

Start\_Read\_Response(ExpectedResponse); /\* First read response \*/

if((Response\_Status != ESP8266\_RESPONSE\_TIMEOUT))

return true; /\* Return true for success \*/

return false; /\* Else return false \*/

}

bool SendATandExpectResponse(char\* ATCommand, char\* ExpectedResponse)

{

ESP8266\_Clear();

USART\_SendString(ATCommand); /\* Send AT command to ESP8266 \*/

USART\_SendString("\r\n");

return WaitForExpectedResponse(ExpectedResponse);

}

bool ESP8266\_ApplicationMode(*uint8\_t* Mode)

{

char \_atCommand[20];

*memset*(\_atCommand, 0, 20);

*sprintf*(\_atCommand, "AT+CIPMODE=%d", Mode);

\_atCommand[19] = 0;

return SendATandExpectResponse(\_atCommand, "\r\nOK\r\n");

}

bool ESP8266\_ConnectionMode(*uint8\_t* Mode)

{

char \_atCommand[20];

*memset*(\_atCommand, 0, 20);

*sprintf*(\_atCommand, "AT+CIPMUX=%d", Mode);

\_atCommand[19] = 0;

return SendATandExpectResponse(\_atCommand, "\r\nOK\r\n");

}

bool ESP8266\_Begin()

{

for (*uint8\_t* i=0;i<5;i++)

{

if(SendATandExpectResponse("ATE0","\r\nOK\r\n")||SendATandExpectResponse("AT","\r\nOK\r\n"))

return true;

}

return false;

}

bool ESP8266\_Close()

{

return SendATandExpectResponse("AT+CIPCLOSE=1", "\r\nOK\r\n");

}

bool ESP8266\_WIFIMode(*uint8\_t* \_mode)

{

char \_atCommand[20];

*memset*(\_atCommand, 0, 20);

*sprintf*(\_atCommand, "AT+CWMODE=%d", \_mode);

\_atCommand[19] = 0;

return SendATandExpectResponse(\_atCommand, "\r\nOK\r\n");

}

*uint8\_t* ESP8266\_JoinAccessPoint(char\* \_SSID, char\* \_PASSWORD)

{

char \_atCommand[60];

*memset*(\_atCommand, 0, 60);

*sprintf*(\_atCommand, "AT+CWJAP=\"%s\",\"%s\"", \_SSID, \_PASSWORD);

\_atCommand[59] = 0;

if(SendATandExpectResponse(\_atCommand, "\r\nWIFI CONNECTED\r\n"))

return ESP8266\_WIFI\_CONNECTED;

else{

if(*strstr*(RESPONSE\_BUFFER, "+CWJAP:1"))

return ESP8266\_CONNECTION\_TIMEOUT;

else if(*strstr*(RESPONSE\_BUFFER, "+CWJAP:2"))

return ESP8266\_WRONG\_PASSWORD;

else if(*strstr*(RESPONSE\_BUFFER, "+CWJAP:3"))

return ESP8266\_NOT\_FOUND\_TARGET\_AP;

else if(*strstr*(RESPONSE\_BUFFER, "+CWJAP:4"))

return ESP8266\_CONNECTION\_FAILED;

else

return ESP8266\_JOIN\_UNKNOWN\_ERROR;

}

}

*uint8\_t* ESP8266\_connected()

{

SendATandExpectResponse("AT+CIPSTATUS", "\r\nOK\r\n");

if(*strstr*(RESPONSE\_BUFFER, "STATUS:2"))

return ESP8266\_CONNECTED\_TO\_AP;

else if(*strstr*(RESPONSE\_BUFFER, "STATUS:3"))

return ESP8266\_CREATED\_TRANSMISSION;

else if(*strstr*(RESPONSE\_BUFFER, "STATUS:4"))

return ESP8266\_TRANSMISSION\_DISCONNECTED;

else if(*strstr*(RESPONSE\_BUFFER, "STATUS:5"))

return ESP8266\_NOT\_CONNECTED\_TO\_AP;

else

return ESP8266\_CONNECT\_UNKNOWN\_ERROR;

}

*uint8\_t* ESP8266\_Start(*uint8\_t* \_ConnectionNumber, char\* Domain, char\* Port)

{

bool \_startResponse;

char \_atCommand[60];

*memset*(\_atCommand, 0, 60);

\_atCommand[59] = 0;

if(SendATandExpectResponse("AT+CIPMUX?", "CIPMUX:0"))

*sprintf*(\_atCommand, "AT+CIPSTART=\"TCP\",\"%s\",%s", Domain, Port);

else

*sprintf*(\_atCommand, "AT+CIPSTART=\"%d\",\"TCP\",\"%s\",%s", \_ConnectionNumber, Domain, Port);

\_startResponse = SendATandExpectResponse(\_atCommand, "CONNECT\r\n");

if(!\_startResponse)

{

if(Response\_Status == ESP8266\_RESPONSE\_TIMEOUT)

return ESP8266\_RESPONSE\_TIMEOUT;

return ESP8266\_RESPONSE\_ERROR;

}

return ESP8266\_RESPONSE\_FINISHED;

}

*uint8\_t* ESP8266\_Send(char\* Data)

{

char \_atCommand[20];

*memset*(\_atCommand, 0, 20);

*sprintf*(\_atCommand, "AT+CIPSEND=%d", (*strlen*(Data)+2));

\_atCommand[19] = 0;

SendATandExpectResponse(\_atCommand, "\r\nOK\r\n>");

if(!SendATandExpectResponse(Data, "\r\nSEND OK\r\n"))

{

if(Response\_Status == ESP8266\_RESPONSE\_TIMEOUT)

return ESP8266\_RESPONSE\_TIMEOUT;

return ESP8266\_RESPONSE\_ERROR;

}

return ESP8266\_RESPONSE\_FINISHED;

}

*int16\_t* ESP8266\_DataAvailable()

{

return (Counter - pointer);

}

*uint8\_t* ESP8266\_DataRead()

{

if(pointer < Counter)

return RESPONSE\_BUFFER[pointer++];

else{

ESP8266\_Clear();

return 0;

}

}

*uint16\_t* Read\_Data(char\* \_buffer)

{

*uint16\_t* len = 0;

*\_delay\_ms*(100);

while(ESP8266\_DataAvailable() > 0)

\_buffer[len++] = ESP8266\_DataRead();

return len;

}

void USART\_tx\_string( char \*data )

{

while ((\*data != '\0'))

{

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

UDR0 = \*data;

data++;

}

}

/\* INIT USART (RS-232) \*/

void USART\_init( unsigned int ubrr )

{

UBRR0H = (unsigned char)(ubrr>>8);

UBRR0L = (unsigned char)ubrr;

UCSR0B = (1 << TXEN0); // Enable RX, TX & RX interrupt

UCSR0C = (3 << UCSZ00); //asynchronous 8 N 1

}

void ADPS\_Init() /\* Gyro initialization function \*/

{

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

tmp = i2c\_start(0x72); /\* Start with device write address \*/

tmp = i2c\_write(APDS9960\_ENABLE); /\* Write to sample rate register \*/

tmp = i2c\_write(0x03); /\* 1KHz sample rate \*/

i2c\_stop();

tmp = i2c\_start(0x72);

tmp = i2c\_write(APDS9960\_ATIME); /\* Write to power management register \*/

i2c\_write(0xdb);

i2c\_stop();

tmp = i2c\_start(0x72);

tmp = i2c\_write(APDS9960\_CONTROL); /\* Write to Configuration register \*/

i2c\_write (0x01);

i2c\_stop();

}

void ADPS\_Start\_Loc()

{

i2c\_start(0x72); /\* I2C start with device write address \*/

i2c\_write(APDS9960\_CDATAL);/\* Write start location address from where to read \*/

i2c\_start(0x73);/\* I2C start with device read address \*/

}

void Read\_RawValue()

{

ADPS\_Start\_Loc(); /\* Read Gyro values \*/

c = (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

r = (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

g= (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

b= (((int)i2c\_read\_ack()<<8) | (int)i2c\_read\_ack());

i2c\_stop();

}

int main(void)

{

char buffer[20];

*uint8\_t* Connect\_Status;

i2c\_init(); /\* Initialize I2C \*/

ADPS\_Init(); /\* Initialize Gyro \*/

USART\_init(BAUD\_PRESCALLER); // Initialize the USART

USART\_tx\_string("Connected!\r\n"); // we're alive!

/\* Replace with your application code \*/

ESP8266\_WIFIMode(STATION); //3 = Both (AP and STA) \*/

ESP8266\_ConnectionMode(SINGLE); /\* 0 = Single; 1 = Multi \*/

ESP8266\_ApplicationMode(NORMAL); /\* 0 = Normal Mode; 1 = Transperant Mode \*/

if(ESP8266\_connected() == ESP8266\_NOT\_CONNECTED\_TO\_AP)

ESP8266\_JoinAccessPoint(SSID, PASSWORD);

ESP8266\_Start(0, DOMAIN, PORT);

while (1)

{

char \_buffer[150];

Read\_RawValue();

Connect\_Status = ESP8266\_connected();

if(Connect\_Status == ESP8266\_NOT\_CONNECTED\_TO\_AP)

ESP8266\_JoinAccessPoint(SSID, PASSWORD);

if(Connect\_Status == ESP8266\_TRANSMISSION\_DISCONNECTED)

ESP8266\_Start(0, DOMAIN, PORT);

*sprintf*(buffer," clear: %d \t",c);

USART\_tx\_string(buffer);

*sprintf*(\_buffer, "GET /update?api\_key=%s&field1=%3d", API\_WRITE\_KEY, c);

ESP8266\_Send(\_buffer);

*sprintf*(buffer, " Red: %d \t",r);

USART\_tx\_string(buffer);

*sprintf*(\_buffer, "GET /update?api\_key=%s&field2=%3d", API\_WRITE\_KEY, r);

ESP8266\_Send(\_buffer);

*sprintf*(buffer," Green = %d \t",g);

USART\_tx\_string(buffer);

*sprintf*(\_buffer, "GET /update?api\_key=%s&field3=%3d", API\_WRITE\_KEY, g);

ESP8266\_Send(\_buffer);

*sprintf*(buffer," Blue = %d \t",b);

USART\_tx\_string(buffer);

*sprintf*(\_buffer, "GET /update?api\_key=%s&field4=%3d", API\_WRITE\_KEY, b);

ESP8266\_Send(\_buffer);

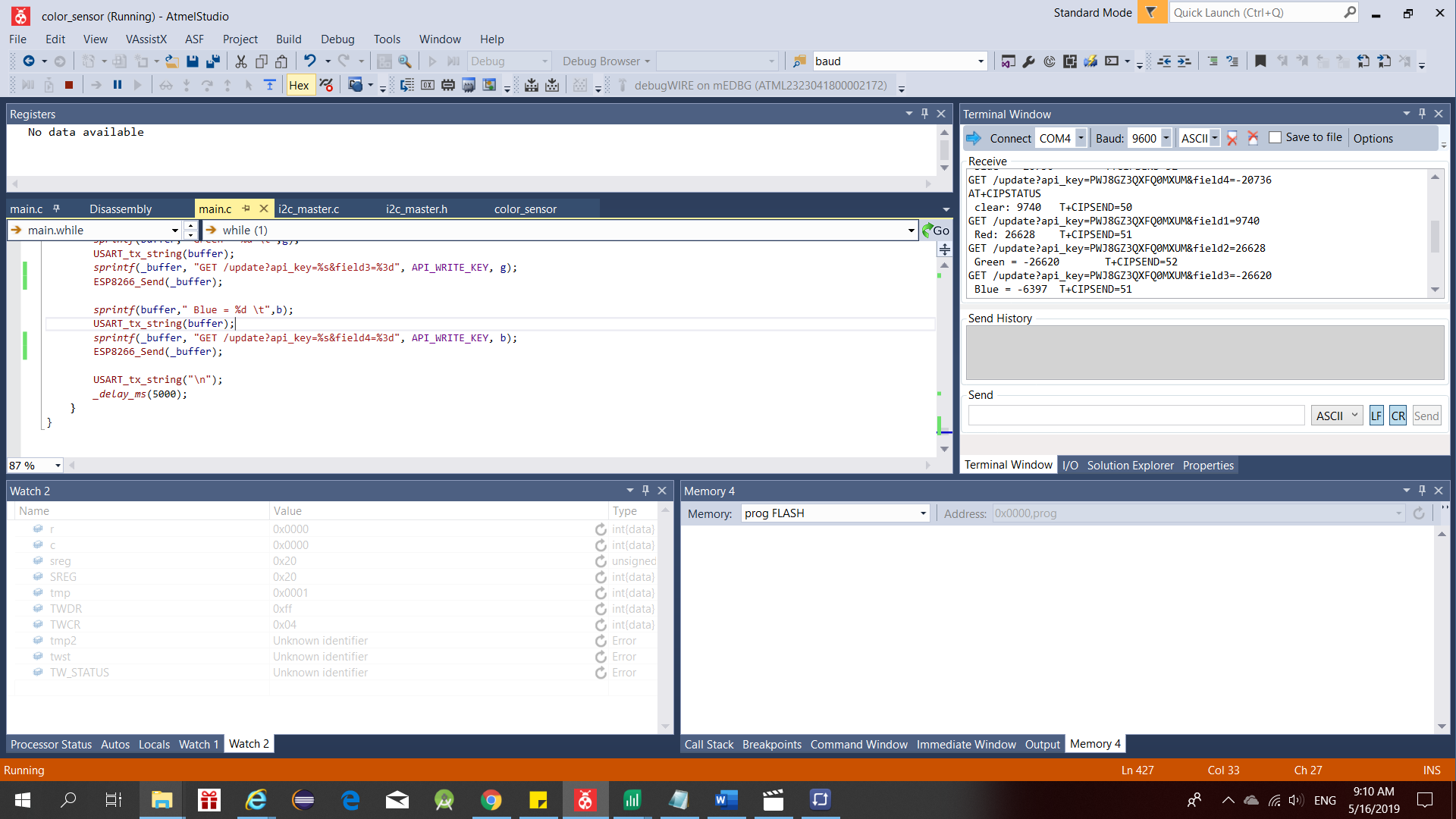
USART\_tx\_string("\n");

*\_delay\_ms*(5000);

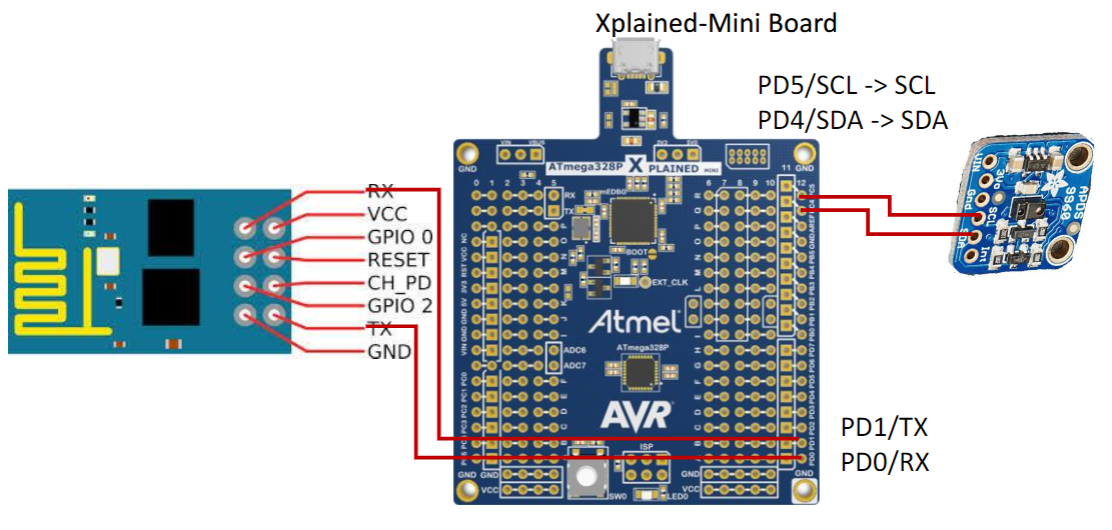
}

}

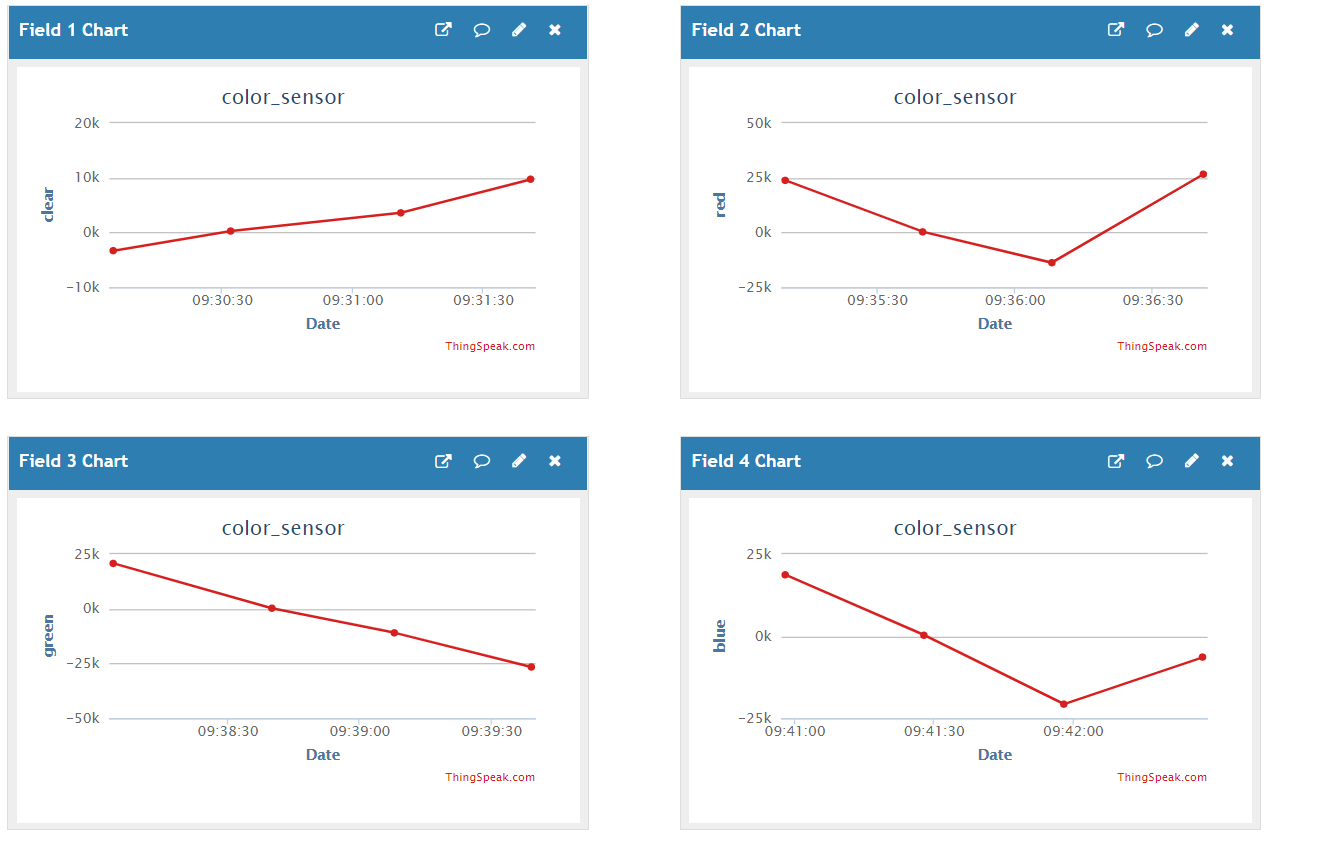
1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

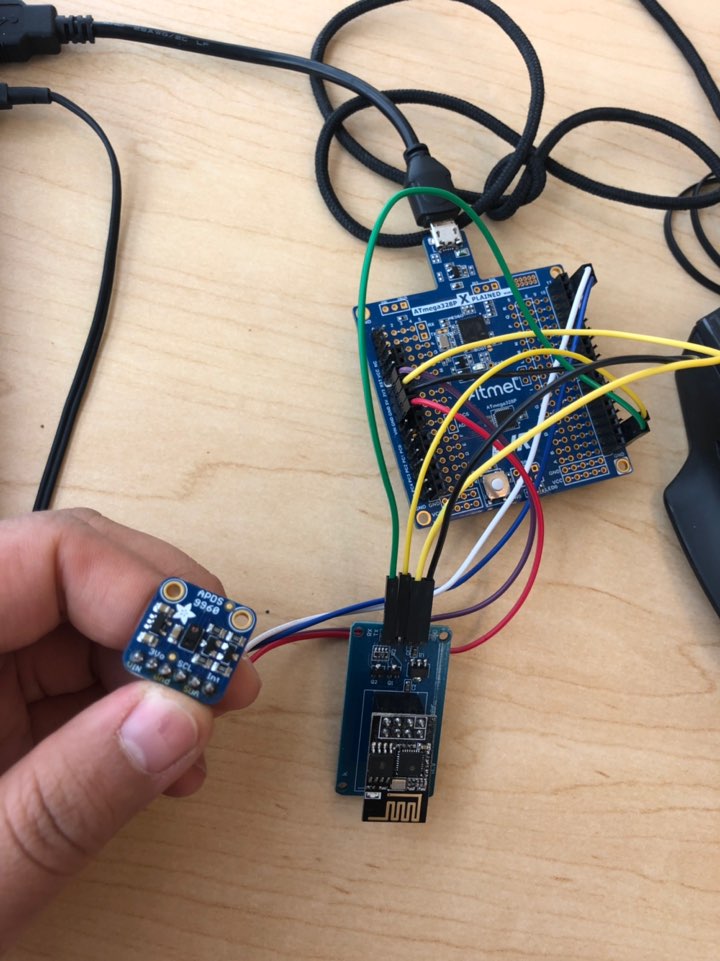
****

1. **SCHEMATICS**



1. **SCREENSHOTS OF EACH TASK OUTPUT (THINGSPEAK)**



1. **SCREENSHOT OF EACH DEMO (BOARD SETUP) **
2. **VIDEO LINKS OF EACH DEMO**

<https://youtu.be/BhCjwT7UXCk>

1. **GITHUB LINK OF THIS DA**

Primary Github address: https://github.com/yeeun219/submission\_da.git

Directory: cpe301\DesignAssignments\MIDTERM2

**Student Academic Misconduct Policy**

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

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

NAME OF THE STUDENT