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

2. DEVELOPED CODE OF TASK 1

```c
/*
 * 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++;
       }
}
```

3. 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++;
	}
}
```

<u>i2c_master.h</u>

```
/*
 * 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

```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);
}
```

<u>APDS9960_def.h</u>

```
/*
 * 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
```

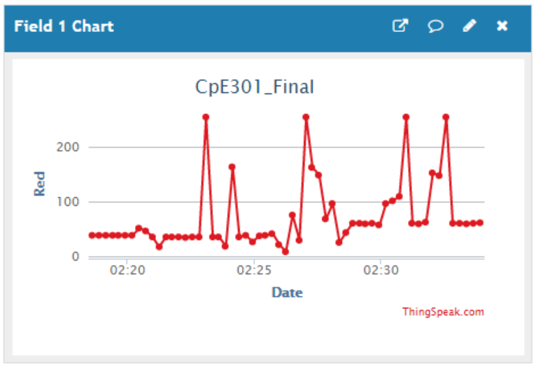
```c
#define APDS9960_GFLVL          0xAE
#define APDS9960_GSTATUS        0xAF
#define APDS9960_IFORCE         0xE4
#define APDS9960_PICLEAR        0xE5
#define APDS9960_CICLEAR        0xE6
#define APDS9960_AICLEAR        0xE7
#define APDS9960_GFIFO_U        0xFC
#define APDS9960_GFIFO_D        0xFD
#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
```

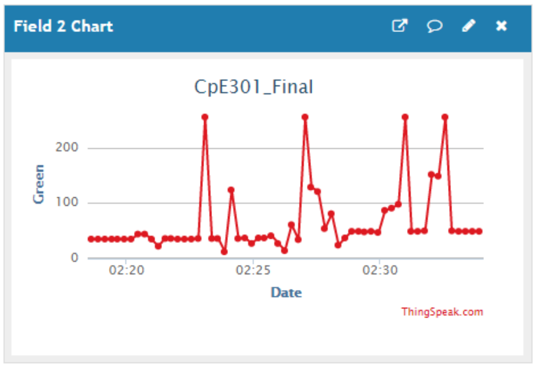
```
/* LED Boost values */
#define LED_BOOST_100           0
#define LED_BOOST_150           1
#define LED_BOOST_200           2
#define LED_BOOST_300           3

/* 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
```

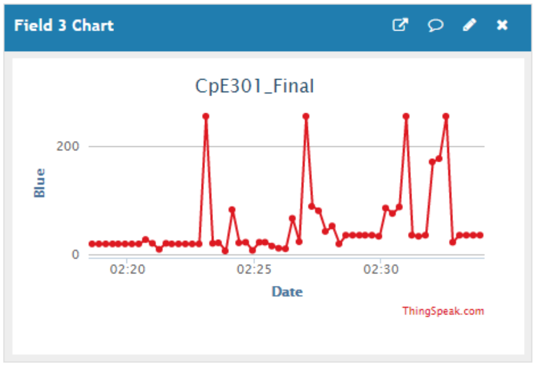
4. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

Channel Stats

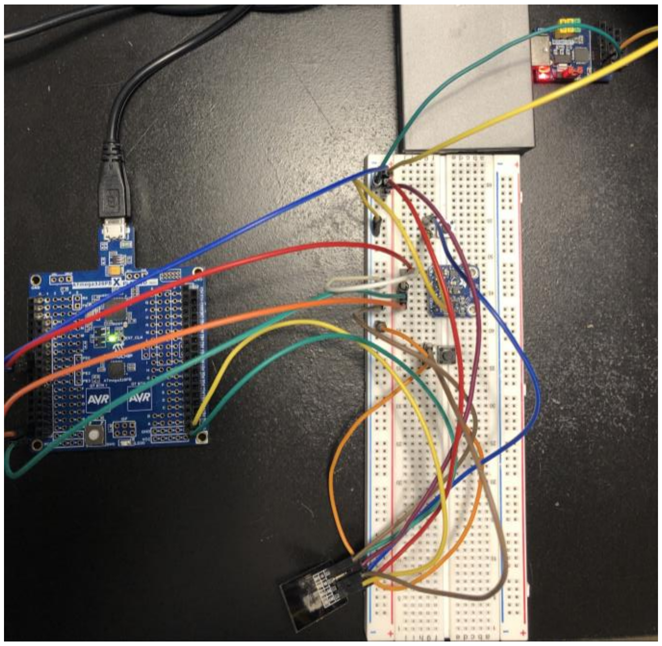
Created: about an hour ago
Entries: 75







5. SCREENSHOT OF EACH DEMO (BOARD SETUP)



6. VIDEO LINKS OF EACH DEMO

https://youtu.be/5QeY6dw3u6M

7. GITHUB LINK OF THIS DA

https://github.com/reedjacobp/submission_da/tree/master/FinalProject