CPE301 – SPRING 2019

MIDTERM 1

Student Name: Cody Hudson

Student #: 8000603642

Student Email: hudsoc1@unlv.nevada.edu

Primary Github address: https://github.com/eed911/class\_proj.git

Directory: https://github.com/eed911/class\_proj/tree/master/Midterm2\_Final

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

List of Components used:

* Atmega 328p
* ESP01
* APDS 9960 RGB light sensor

Block diagram with pins used in the Atmega328P:



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

Q: Write, simulate, and demonstrate using Atmel Studio 7 a C code for the AVR

ATMEGA328p microcontroller that performs the following functions:

1. Program the I2C of ATmega328/p to read RGB/Ambient Light data from APDS

9960 sensor.

2. Display the value to UART.

3. Make sure the AT Firmware is downloaded into the ESP-01/ESP32 module.

4. Register for a free Thingspeak account with MATHWORK. Setup and get the

channel Key.

5. Transmit Lux sensor value to ESP-01/ESP32 through UART port using AT

Commands.

6. Display the Lux sensor value as a graph in Thingspeak

/\*

midterm2.c

Created: 5/13/2019 5:57:39 PM

Author : Cody Hudson

\*/

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include "i2c\_master.h"

#include "uart.h"

#include "apds.h"

#define BAUD 9600

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

#ifndef APDS\_H

#define APDS\_H

#include <avr/io.h>

#include "i2c\_master.h"

#include "apds.h"

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

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

/\* 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 proxy or ALS for int.

#define DEFAULT\_CONFIG2 0x01 // No saturation interrupts or LED boost

#define DEFAULT\_CONFIG3 0 // Enable all photo diodes, no SAI

#define DEFAULT\_GLDRIVE LED\_DRIVE\_100MA

#define DEFAULT\_GWTIME GWTIME\_2\_8MS

void apds\_init();

void readColor();

#endif

*FILE* str\_uart = *FDEV\_SETUP\_STREAM*(uart\_putchar, *NULL* , *\_FDEV\_SETUP\_WRITE*);

char results[256];

void init\_UART();

int uart\_putchar( char c, *FILE* \*stream);

int main(void){

*uint16\_t* red = 0, green = 0, blue = 0; //initialize variables red, green, blue

i2c\_init(); // call i2c init

init\_UART(); //initialize USART

*stdout* = &str\_uart;

apds\_init();

*\_delay\_ms*(2000); //delay 2 seconds

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

*\_delay\_ms*(5000); //delay 5 seconds

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

*\_delay\_ms*(5000); //delay 5 seconds

*printf*("AT+CWJAP=\"Michael iPhone\",\"hello dad\"\r\n"); //connect to michaels iphone wifi

while (1) { // continuously send the data to thingspeak

*\_delay\_ms*(5000);

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

*\_delay\_ms*(5000);

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

*\_delay\_ms*(5000);

readColor(&red, &green, &blue);

*printf*("AT+CIPSEND=104\r\n");

*printf*("GET https://api.thingspeak.com/update?api\_key=4GOY7N6MQ2MKJOS9&field1=%05u&field2=%05u&field3=%05u\r\n", red, green, blue);

*\_delay\_ms*(5000);

}

}

void apds\_init(){

*uint8\_t* setup;

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

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

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

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

setup = DEFAULT\_ATIME;

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

setup = DEFAULT\_WTIME;

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

setup = DEFAULT\_PROX\_PPULSE;

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

setup = DEFAULT\_POFFSET\_UR;

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

setup = DEFAULT\_POFFSET\_DL;

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

setup = DEFAULT\_CONFIG1;

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

setup = DEFAULT\_PERS;

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

setup = DEFAULT\_CONFIG2;

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

setup = DEFAULT\_CONFIG3;

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

}

void readColor(*uint16\_t* \*red, *uint16\_t* \*green, *uint16\_t* \*blue){

*uint8\_t* redl, redh;

*uint8\_t* greenl, greenh;

*uint8\_t* bluel, blueh;

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

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

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

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

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

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

\*red = redh << 8 | redl;

\*green = greenh << 8 | greenl;

\*blue = blueh << 8 | bluel;

}

void init\_UART(void){

//Set baud rate

*uint16\_t* baud\_rate = BRGVAL;

UBRR0H = baud\_rate >> 8;

UBRR0L = baud\_rate & 0xFF;

//Enable receiver and transmitter

UCSR0B = ( 1 <<RXEN0)|( 1 <<TXEN0);

// Set frame format: 8data, 1stop bit

UCSR0C = (3 <<UCSZ00);

}

int uart\_putchar(char c, *FILE* \*stream){

//wait until buffer empty

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

//Put data into buffer

UDR0 = c;

return 0;

}

**I2C master included file:**

#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 included file:**

#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);

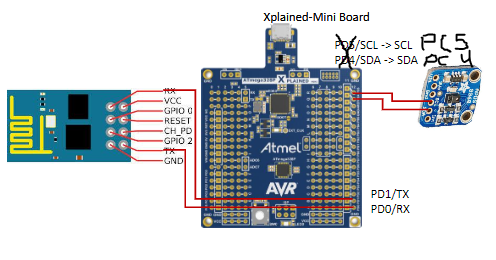
*uint8\_t* i2c\_writeReg(*uint8\_t* devaddr, *uint8\_t* regaddr, *uint8\_t*\* data, *uint16\_t* length);

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

void i2c\_stop(void);

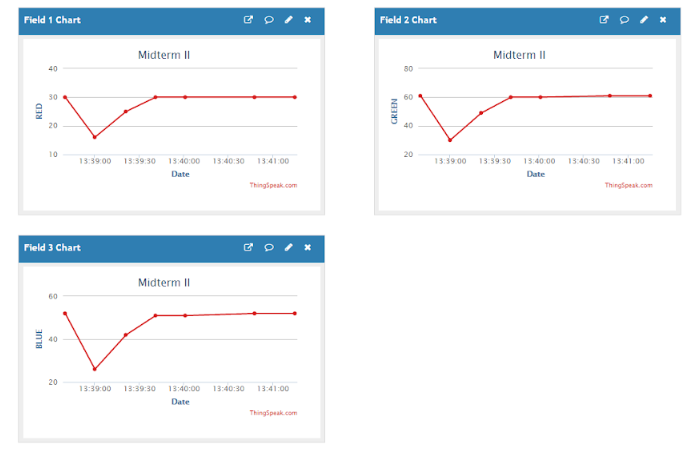
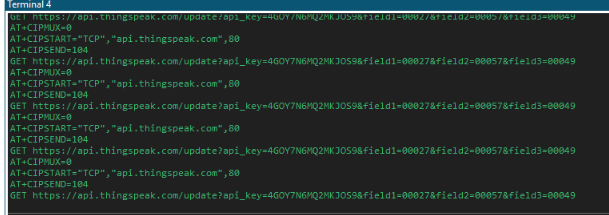
#endif // I2C\_MASTER\_H

**SCHEMATICS**

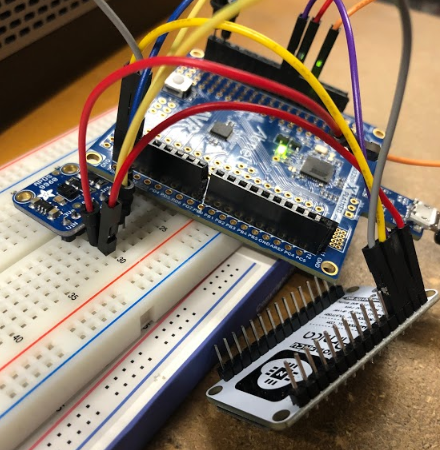


Couldn’t find components using fritzing

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



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



1. **VIDEO LINKS OF EACH DEMO**

Demo1:

<https://www.youtube.com/watch?v=XGq0Urg_gPs>

1. **GITHUB LINK OF THIS DA**

https://github.com/eed911/class\_proj.git

**Student Academic Misconduct Policy**

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

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

Cody Hudson