Yannick Kengne Tatcha

CPE301 – SPRING 2018

Midterm 2

**DO NOT REMOVE THIS PAGE DURING SUBMISSION:**

The student understands that all required components should be submitted in complete for grading of this assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **SUBMISSION ITEM** | **COMPLETED (Y/N)** | **MARKS**  **(/MAX)** |
| 1. | COMPONENTS LIST |  |  |
| 2. | TASK 1 & 2 COMPLETE CODE W/ COMMENTS AND SCREENSHOTS |  |  |
| 3. | SCHEMATICS AND SCREENSHOTS |  |  |
| 4. | GitHub Link |  |  |
|  |  |  |  |
|  |  |  |  |

1. **COMPONENTS LIST**

Needed:

FTDI232R chip

ATMEGA328P Microcontroller

LM34 Sensor

ESP8266-01

1. **TASK 1 & 2 COMPLETE CODE W/ COMMENTS AND SCREENSHOTS**

Transmit Code w/ comments:

#define *F\_CPU* 16000000UL //16 Mhz.

#define BAUD 9600 //Set Baud rate 9600.

#define MYUBRR *F\_CPU*/16/BAUD-1 //Configuration for MYUBRR.

#include <avr/io.h> //include necessary libraries/files

#include <avr/interrupt.h>

#include <stdbool.h>

#include <string.h>

#include <util/delay.h>

#include "nrf24l01.h"

#include "nrf24l01-mnemonics.h"

#include <stdio.h>

void read\_adc(void); //function to read ADC value

void adc\_init(void); //function to initialize the ADC

void USART\_init(unsigned int ubrr); //Function to initialize USART

void USART\_tx\_string(char \*data); //Function to print the ADC value

void setup\_timer(void); //function to set up a timer interrupt every 1 second

nRF24L01 \*setup\_rf(void); //function to set up nRF24L01 chip

volatile bool rf\_interrupt = false;

volatile bool send\_message = false;

volatile unsigned int adc\_temp; //variable to hold ADC value read from LM34

char outs[20]; //variable to hold ADC value

void USART\_init(unsigned int ubrr)

{

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

UBRR0L = (unsigned char)ubrr;

UCSR0B = (1<<TXEN0); //Enable receiver, transmitter & RX interrupt.

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

}

void USART\_Transmit(unsigned char data)

{

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

UDR0=data; //set UDR0 = data

}

int main(void)

{

adc\_init(); //initialize ADC

setup\_timer(); //set up a timer1 overflow. Interrupts every 1 second

USART\_init(MYUBRR); //initialize USART

USART\_tx\_string("Connected!\r\n"); //lets us know if we are connected to the visualizer

*\_delay\_ms*(125); //wait for 125 us

*uint8\_t* to\_address[5] = { 0x21, 0x26, 0x32, 0xFF, 0x3E }; //Can pick any combination of hex values so long as it is the same in the receive address.

bool on = false;

nRF24L01 \*rf = setup\_rf();

sei(); //set enable interrupt

while (true) {

if (rf\_interrupt) {

rf\_interrupt = false;

int success = nRF24L01\_transmit\_success(rf);

if (success != 0)

nRF24L01\_flush\_transmit\_message(rf);

}

if (send\_message) {

send\_message = false;

on = !on;

nRF24L01Message msg;

if (on) *memcpy*(msg.data, "ON", 3);

else *memcpy*(msg.data, "OFF", 4);

msg.length = *strlen*((char \*)msg.data) + 1;

nRF24L01\_transmit(rf, to\_address, &msg);

}

}

return 0;

}

nRF24L01 \*setup\_rf(void)

{

nRF24L01 \*rf = nRF24L01\_init();

rf->ss.port = &PORTB;

rf->ss.pin = PB2 //set ss to PB2

rf->ce.port = &PORTB;

rf->ce.pin = PB1; //set ce to PB1

rf->sck.port = &PORTB;

rf->sck.pin = PB5; //set sck to PB5

rf->mosi.port = &PORTB;

rf->mosi.pin = PB3; //set MOSI to PB3

rf->miso.port = &PORTB;

rf->miso.pin = PB4; //set MISO to PB4

// interrupt on falling edge of INT0 (PD2)

EICRA |= \_BV(ISC01);

EIMSK |= \_BV(INT0);

nRF24L01\_begin(rf);

return rf;

}

// setup timer to trigger interrupt every second when at 1MHz

void setup\_timer(void)

{

TIMSK1 |= (1<<TOIE1); //enable Timer1 overflow interrupt

TCCR1B |= (1<<CS12) | (1<<CS10); //set timer prescale to 1024

TCNT1 = 49911; //set TCNT value. Calculated using 65535 - (16MHz/1024-1)

}

// each one second interrupt

ISR(TIMER1\_COMPA\_vect)

{

send\_message = true; //set send\_message = true

}

// nRF24L01 interrupt

ISR(INT0\_vect)

{

rf\_interrupt = true; //set rf\_interrupt = true

}

void read\_adc(void) //this function is responsible for reading the ADC pins

{

unsigned char i=4; //declare how many times we will measure temperature per average

adc\_temp=0; //initialize adc\_temp to 0

while(i--) //while i != 0, keep looping

{

ADCSRA |= (1<<ADSC); //start the first conversion

while(ADCSRA & (1<<ADSC)); //while ADCSRA and 1 is written to ADSC, keep looping

adc\_temp+=ADC; //sum the ADC values

*\_delay\_ms*(50); //wait for 50 ms

}

adc\_temp = adc\_temp /8; //take the average of the measured temperatures

}

void adc\_init(void) //this function is responsible for setting up and enabling the ADC

{

ADMUX = (0 << REFS1) | (1<<REFS0) | (0<<ADLAR) | (1<<MUX1) | (1<<MUX0);

ADCSRA = (1<<ADEN) | (0<<ADSC) | (0 << ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2)| (0<<ADPS1) | (1<<ADPS0);

}

ISR(TIMER1\_OVF\_vect)//

{

read\_adc();

*snprintf*(outs,sizeof(outs),"%3d\r\n",adc\_temp); //Change to "%3d F\r\n" for degree output.

USART\_tx\_string(outs);

*\_delay\_ms*(125);

TCNT1 = 49911; //Reset TCNT1.

}

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

{

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

{

while(!(UCSR0A & (1<<UDRE0))); //print out the variable data

UDR0 = \*data;

data++;

}

}

Receive Code w/ comments:

#define *F\_CPU* 16000000UL //16 MHz

#define BAUD 9600 //Set Baud rate 9600.

#define MYUBRR *F\_CPU*/16/BAUD-1 //Configuration for MYUBRR.

#include <avr/io.h> //include necessary libraries/files

#include <avr/interrupt.h>

#include <stdbool.h>

#include <string.h>

#include <util/delay.h>

#include <stdio.h>

#include "nrf24l01.h"

#include "nrf24l01-mnemonics.h"

nRF24L01 \*setup\_rf(void);

void process\_message(char \*message);

inline void prepare\_led\_pin(void);

inline void set\_led\_high(void);

inline void set\_led\_low(void);

volatile bool rf\_interrupt = false;

void USART\_init(unsigned int ubrr); //Function to initialize USART

void USART\_Transmit(unsigned char data); //Function to print the value in Visualizer

int main(void) {

*uint8\_t* address[5] = { 0x21, 0x26, 0x32, 0xFF, 0x3E };//Can pick any combination of hex values so long as it is the same in the receive address.

prepare\_led\_pin();

sei(); //set enable interrupt

nRF24L01 \*rf = setup\_rf();

nRF24L01\_listen(rf, 0, address);

*uint8\_t* addr[5];

nRF24L01\_read\_register(rf, CONFIG, addr, 1);

USART\_init(MYUBRR); //initialize USART

while (true) {

if (rf\_interrupt) {

rf\_interrupt = false;

while (nRF24L01\_data\_received(rf)) {

nRF24L01Message msg;

USART\_tx\_string(msg);

nRF24L01\_read\_received\_data(rf, &msg);

process\_message((char \*)msg.data);

}

nRF24L01\_listen(rf, 0, address);

}

}

return 0;

}

nRF24L01 \*setup\_rf(void) {

nRF24L01 \*rf = nRF24L01\_init();

rf->ss.port = &PORTB;

rf->ss.pin = PB2; //set ss to PB2

rf->ce.port = &PORTB;

rf->ce.pin = PB1; //set ce to PB1

rf->sck.port = &PORTB;

rf->sck.pin = PB5; //set sck to PB5

rf->mosi.port = &PORTB;

rf->mosi.pin = PB3; //set MOSI to PB3

rf->miso.port = &PORTB;

rf->miso.pin = PB4; //set MISO to PB4

// interrupt on falling edge of INT0 (PD2)

EICRA |= \_BV(ISC01);

EIMSK |= \_BV(INT0);

nRF24L01\_begin(rf);

return rf;

}

void process\_message(char \*message) {

if (*strcmp*(message, "ON") == 0)

set\_led\_high();

else if (*strcmp*(message, "OFF") == 0)

set\_led\_low();

}

inline void prepare\_led\_pin(void) {

DDRB |= \_BV(PB0);

PORTB &= ~\_BV(PB0);

}

inline void set\_led\_high(void) {

PORTB |= \_BV(PB0);

}

inline void set\_led\_low(void) {

PORTB &= ~\_BV(PB0);

}

// nRF24L01 interrupt

ISR(INT0\_vect) {

rf\_interrupt = true; //set rf\_interrupt = true on INT0 interrupt

}

void USART\_init(unsigned int ubrr)

{

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

UBRR0L = (unsigned char)ubrr;

UCSR0B = (1<<TXEN0); //Enable receiver, transmitter & RX interrupt.

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

}

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

{

unsigned int i=0;

while(str[i]!='\0')

{

USART\_Transmit(str[i]); //print out variable str

i++;

}

}

void USART\_Transmit(unsigned char data)

{

//check if buffer is empty so that data can be written to transmit (check code example in datasheet)

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

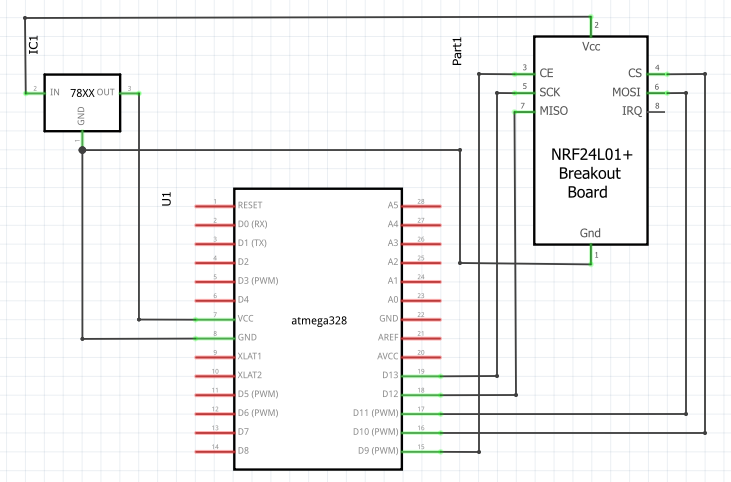
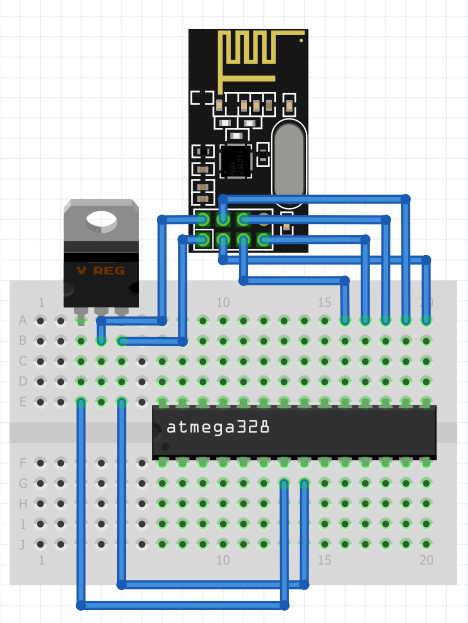
//copy data to be sent to UDR0

UDR0 = data;

}

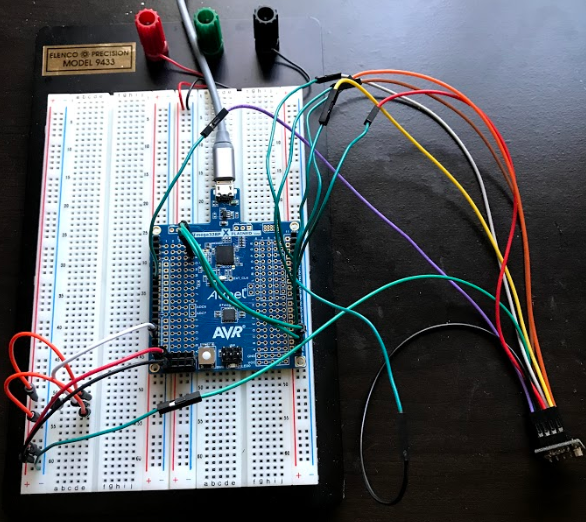
1. **SCHEMATICS AND SCREENSHOTS**

Block and Breadboard Schematic:

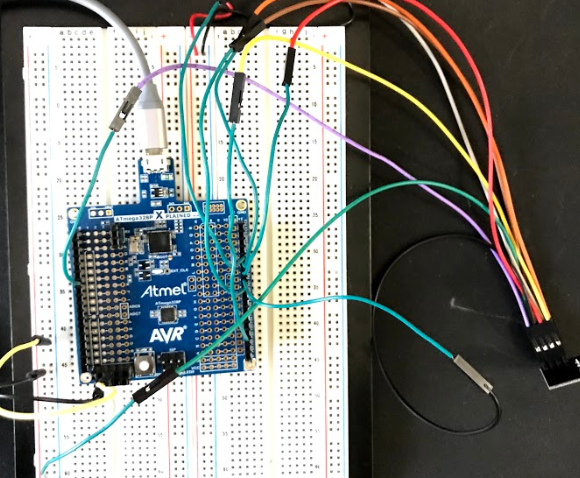


Physical Implementation of Transmitter and Receiver Xmini’s on Breadboard:

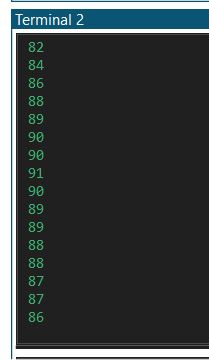
Transmitter:



Receiver:



Operation Screenshots:



1. **GITHUB LINK**

<https://github.com/Vasty1995>

**Student Academic Misconduct Policy**

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

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

Yannick Kengne Tatcha