Nathan Hanuscin
Partner – Brian Lopez

CPE301 – SPRING 2018

Design Assignment Midterm 2

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

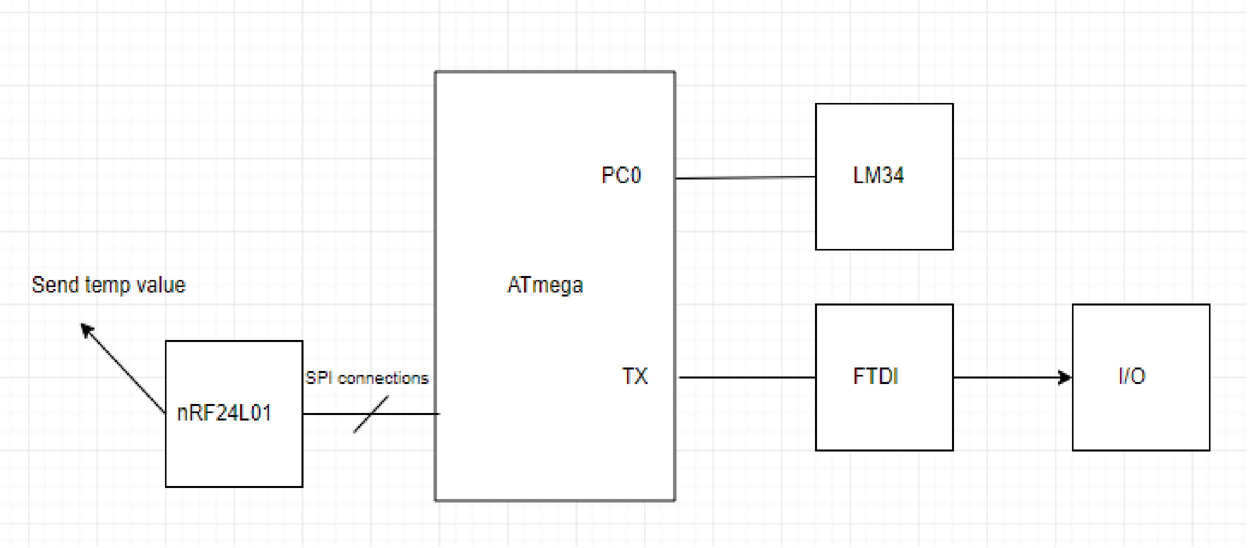
The student understands that all required components should be submitted in complete for grading of this assignment.

NO	SUBMISSION ITEM	COMPLETED (Y/N)	MARKS (/MAX)
1	COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS		
2.	INITIAL CODE OF TASK 1/A		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 2/B		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 3/C		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 4/D		
3.	INCREMENTAL / DIFFERENTIAL CODE OF TASK 5/E		
4.	SCHEMATICS		
5.	SCREENSHOTS OF EACH TASK OUTPUT		
5.	SCREENSHOT OF EACH DEMO		
6.	VIDEO LINKS OF EACH DEMO		
7.	GOOGLECODE LINK OF THE DA		

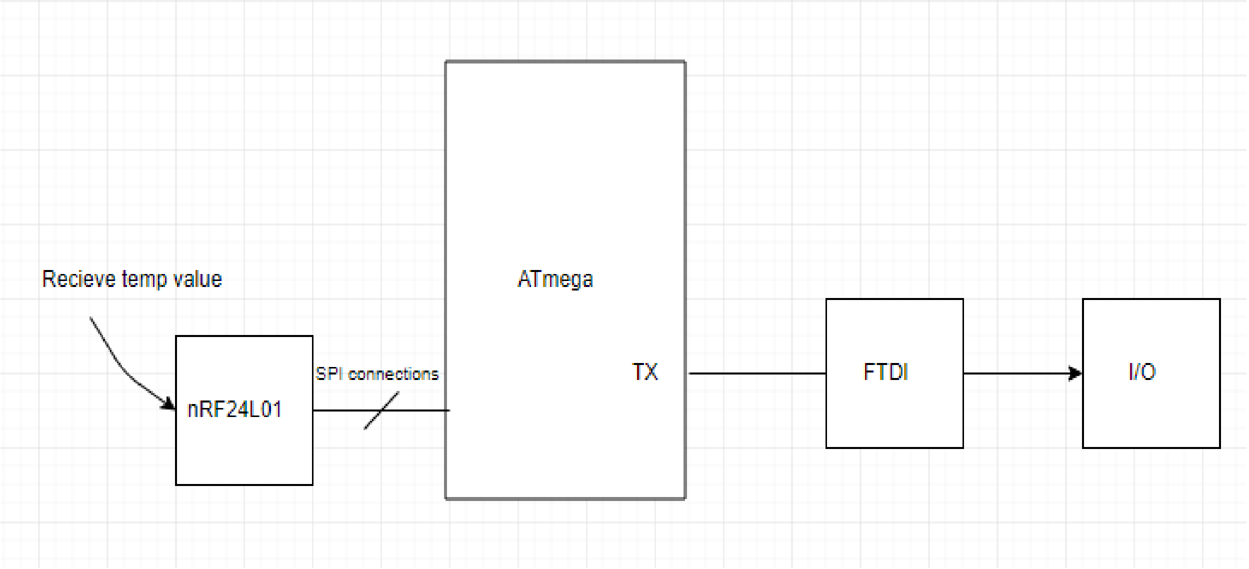
1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

2 nFR24L01/+
LM34 temperature sensor
2 FTDI breakout boards

Transmitter block diagram:

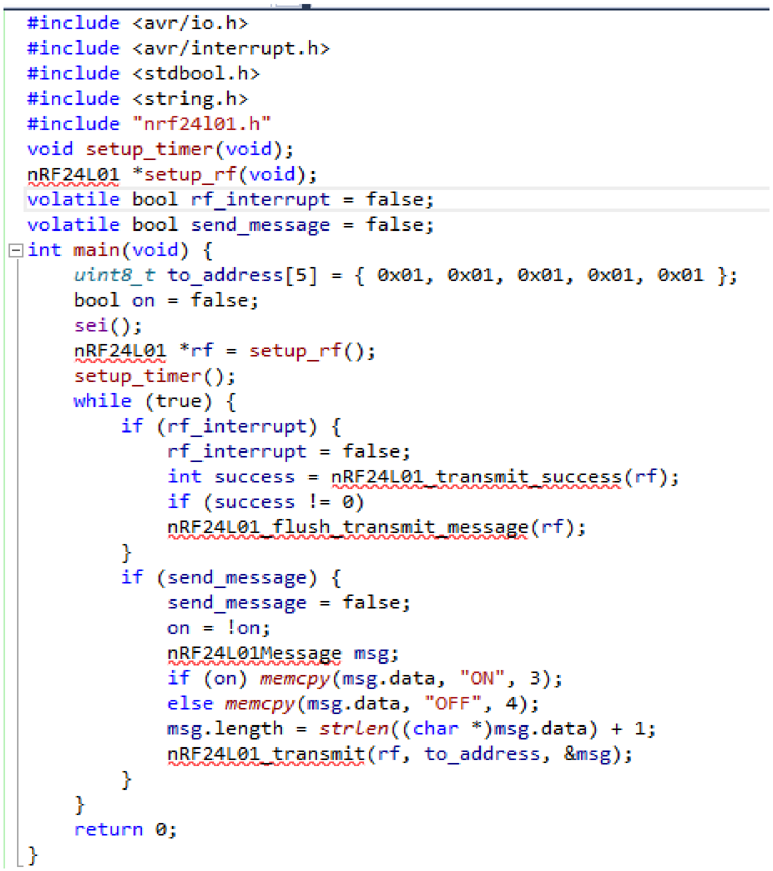


Receiver block diagram:

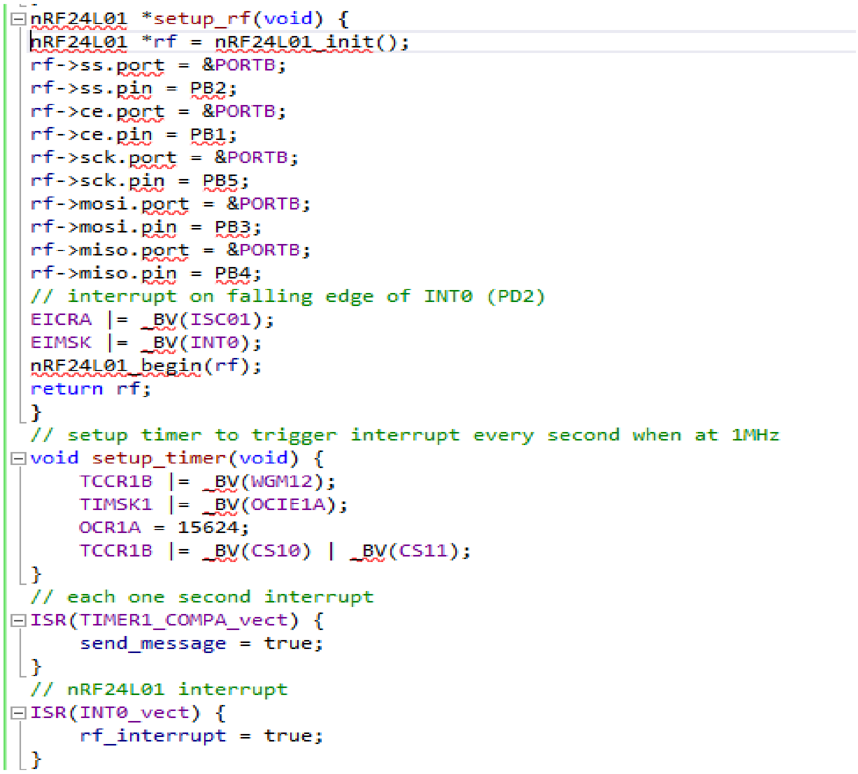


2. INITIAL/DEVELOPED CODE OF TASK 1/A

Transmitter Given code:

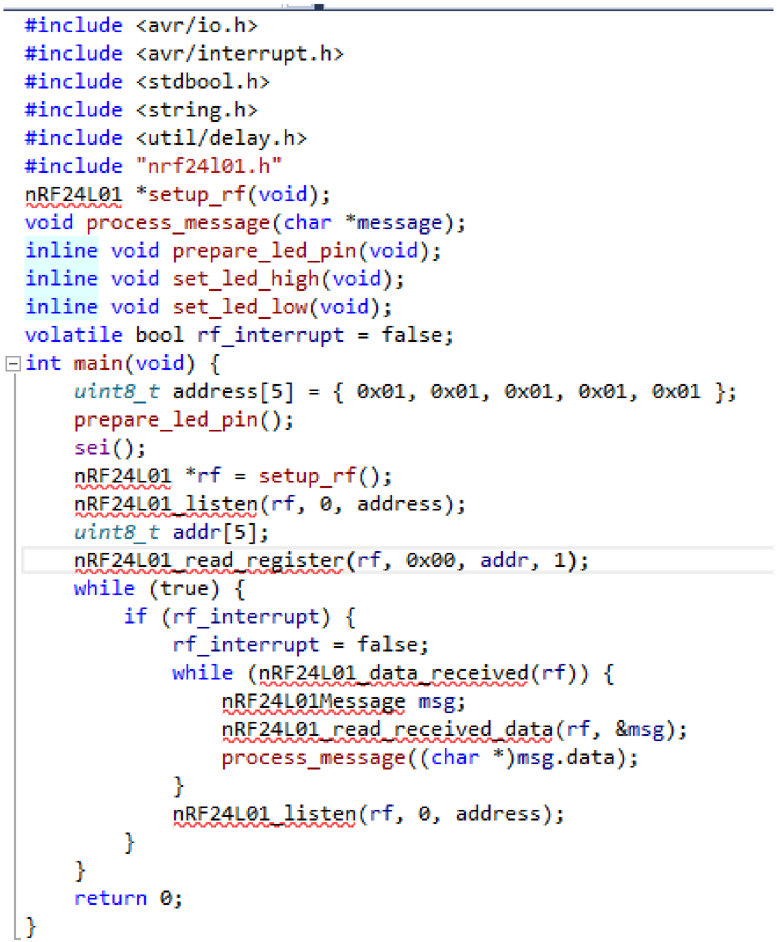


```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <string.h>
#include "nrf24l01.h"
void setup_timer(void);
nRF24L01 *setup_rf(void);
volatile bool rf_interrupt = false;
volatile bool send_message = false;
int main(void) {
    uint8_t to_address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };
    bool on = false;
    sei();
    nRF24L01 *rf = setup_rf();
    setup_timer();
    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;
}
```

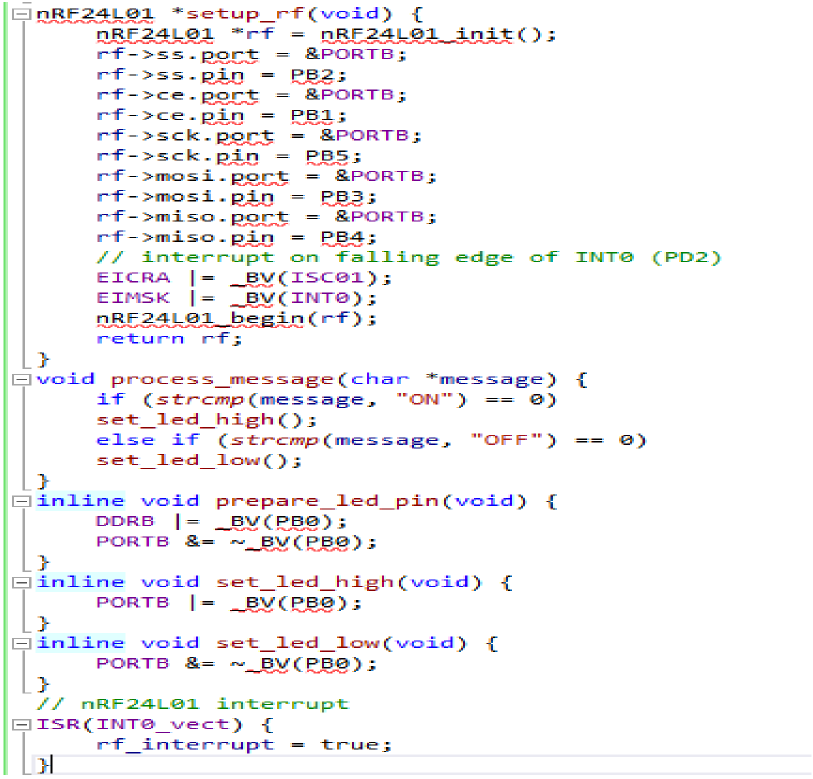


```c
nRF24L01 *setup_rf(void) {
nRF24L01 *rf = nRF24L01_init();
rf->ss.port = &PORTB;
rf->ss.pin = PB2;
rf->ce.port = &PORTB;
rf->ce.pin = PB1;
rf->sck.port = &PORTB;
rf->sck.pin = PB5;
rf->mosi.port = &PORTB;
rf->mosi.pin = PB3;
rf->miso.port = &PORTB;
rf->miso.pin = PB4;
// interrupt on falling edge 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) {
    TCCR1B |= _BV(WGM12);
    TIMSK1 |= _BV(OCIE1A);
    OCR1A = 15624;
    TCCR1B |= _BV(CS10) | _BV(CS11);
}
// each one second interrupt
ISR(TIMER1_COMPA_vect) {
    send_message = true;
}
// nRF24L01 interrupt
ISR(INT0_vect) {
    rf_interrupt = true;
}
```

Receiver given code:



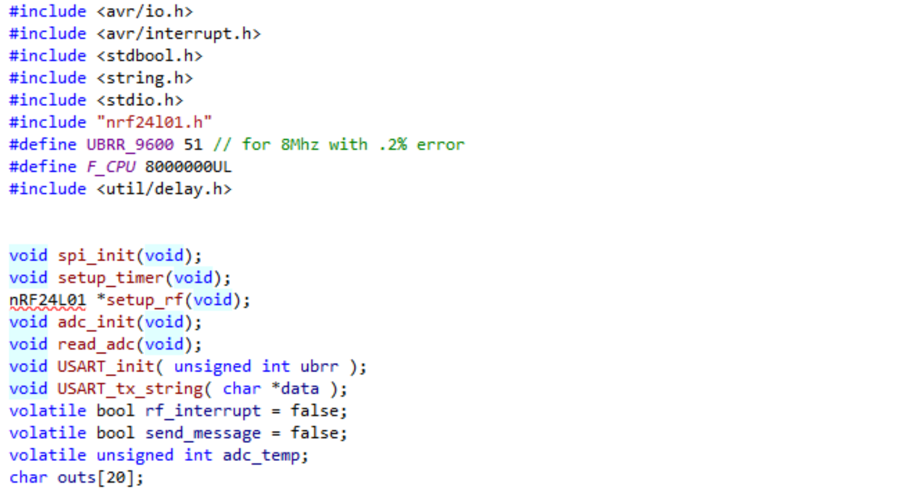
```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <string.h>
#include <util/delay.h>
#include "nrf24l01.h"
nRF24L01 *setup_rf(void);
void process_message(char *message);
inline void prepare_led_pin(void);
inline void set_led_high(void);
inline void set_led_low(void);
volatile bool rf_interrupt = false;
int main(void) {
    uint8_t address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };
    prepare_led_pin();
    sei();
    nRF24L01 *rf = setup_rf();
    nRF24L01_listen(rf, 0, address);
    uint8_t addr[5];
    nRF24L01_read_register(rf, 0x00, addr, 1);
    while (true) {
        if (rf_interrupt) {
            rf_interrupt = false;
            while (nRF24L01_data_received(rf)) {
                nRF24L01Message msg;
                nRF24L01_read_received_data(rf, &msg);
                process_message((char *)msg.data);
            }
            nRF24L01_listen(rf, 0, address);
        }
    }
    return 0;
}
```



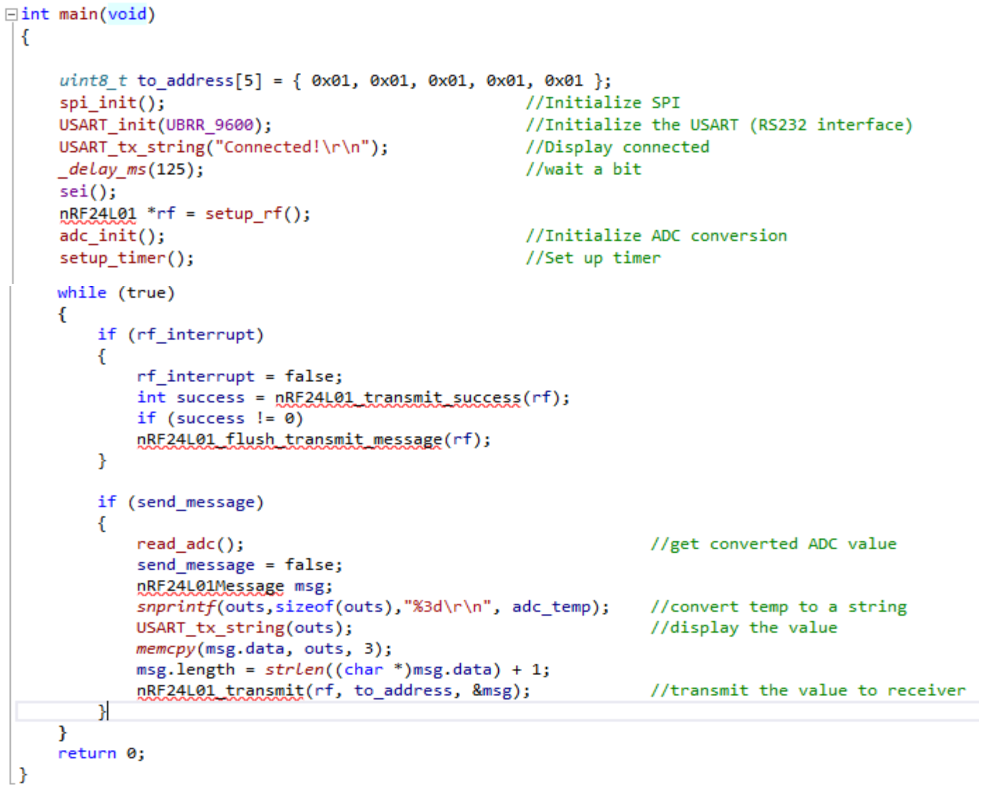
```c
nRF24L01 *setup_rf(void) {
    nRF24L01 *rf = nRF24L01_init();
    rf->ss.port = &PORTB;
    rf->ss.pin = PB2;
    rf->ce.port = &PORTB;
    rf->ce.pin = PB1;
    rf->sck.port = &PORTB;
    rf->sck.pin = PB5;
    rf->mosi.port = &PORTB;
    rf->mosi.pin = PB3;
    rf->miso.port = &PORTB;
    rf->miso.pin = 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;
}
```

3. MODIFIED CODE OF TASK 2/A from TASK 1/A

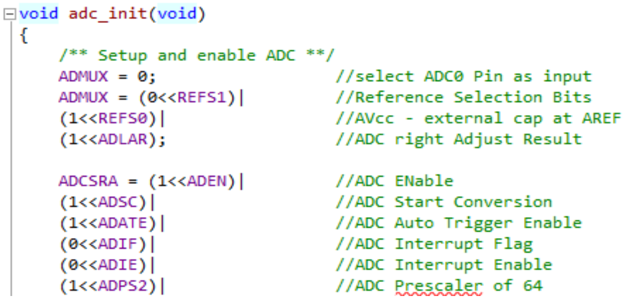
Transmitter modified code:



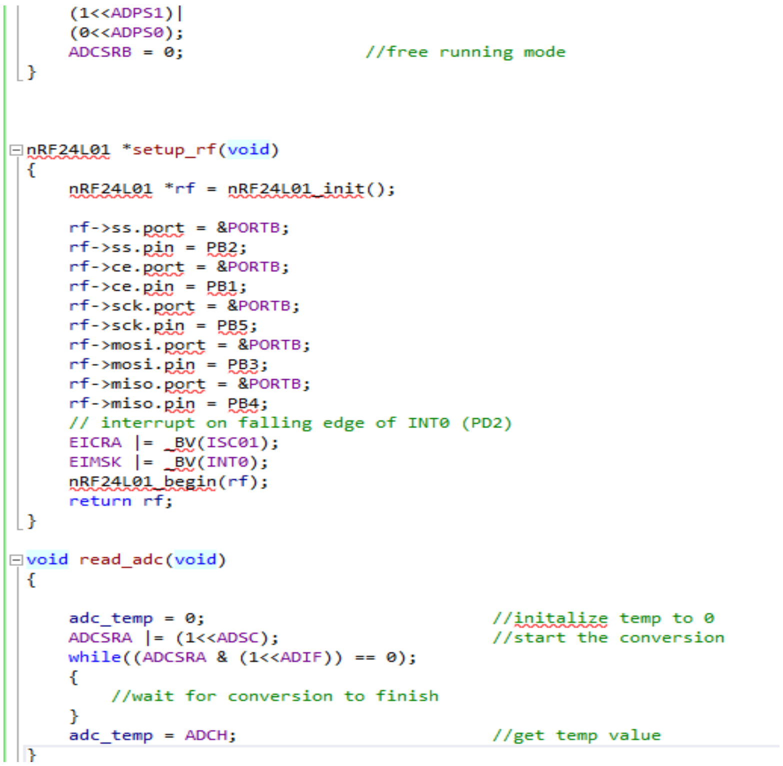
```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include "nrf24l01.h"
#define UBRR_9600 51 // for 8Mhz with .2% error
#define F_CPU 8000000UL
#include <util/delay.h>


void spi_init(void);
void setup_timer(void);
nRF24L01 *setup_rf(void);
void adc_init(void);
void read_adc(void);
void USART_init( unsigned int ubrr );
void USART_tx_string( char *data );
volatile bool rf_interrupt = false;
volatile bool send_message = false;
volatile unsigned int adc_temp;
char outs[20];
```



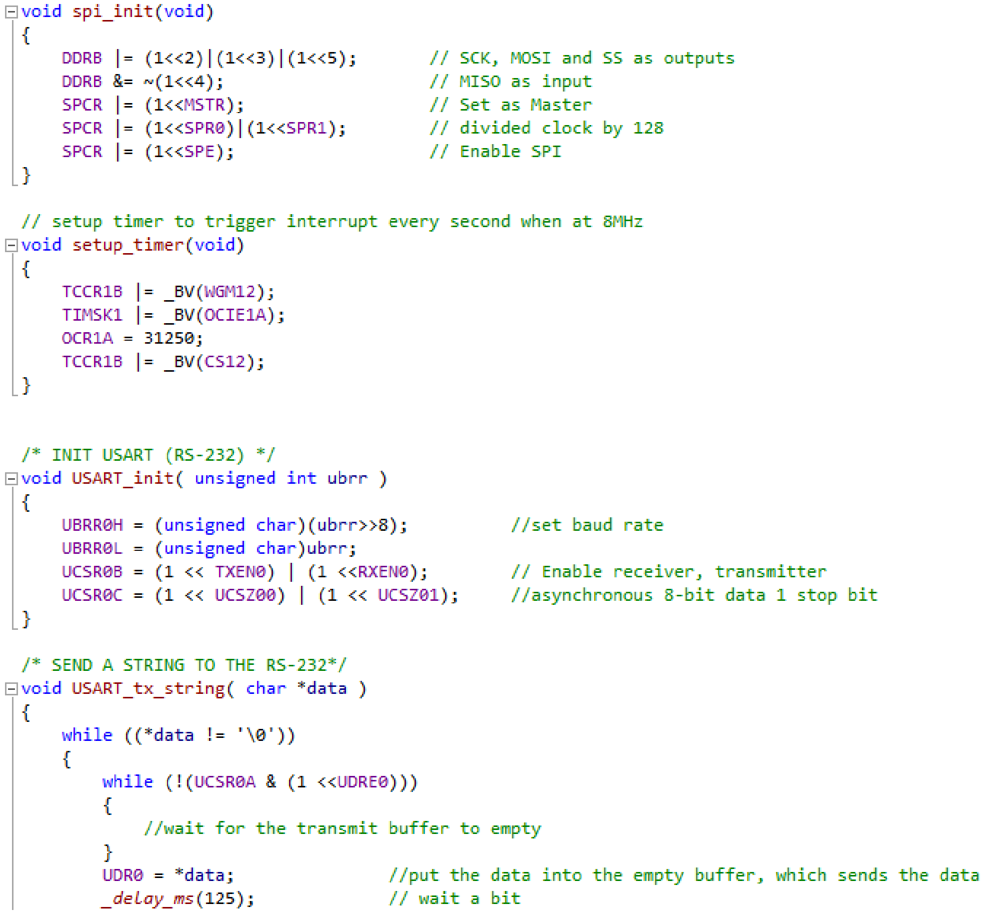
```c
int main(void)
{

    uint8_t to_address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };
    spi_init();                                     //Initialize SPI
    USART_init(UBRR_9600);                          //Initialize the USART (RS232 interface)
    USART_tx_string("Connected!\r\n");              //Display connected
    _delay_ms(125);                                 //wait a bit
    sei();
    nRF24L01 *rf = setup_rf();
    adc_init();                                     //Initialize ADC conversion
    setup_timer();                                  //Set up timer

    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)
        {
            read_adc();                                          //get converted ADC value
            send_message = false;
            nRF24L01Message msg;
            snprintf(outs,sizeof(outs),"%3d\r\n", adc_temp);     //convert temp to a string
            USART_tx_string(outs);                               //display the value
            memcpy(msg.data, outs, 3);
            msg.length = strlen((char *)msg.data) + 1;
            nRF24L01_transmit(rf, to_address, &msg);             //transmit the value to receiver
        }
    }
    return 0;
}
```



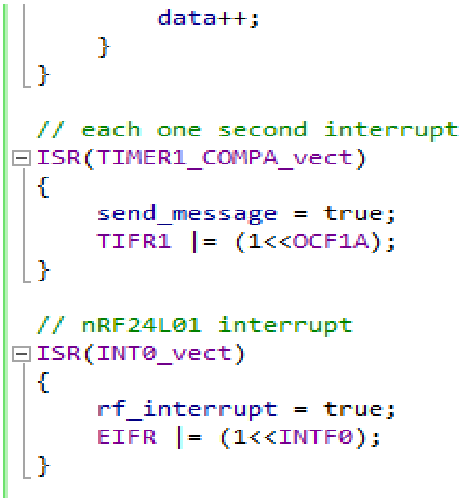
```c
void adc_init(void)
{
    /** Setup and enable ADC **/
    ADMUX = 0;                  //select ADC0 Pin as input
    ADMUX = (0<<REFS1)|         //Reference Selection Bits
    (1<<REFS0)|                 //AVcc - external cap at AREF
    (1<<ADLAR);                 //ADC right Adjust Result

    ADCSRA = (1<<ADEN)|         //ADC ENable
    (1<<ADSC)|                  //ADC Start Conversion
    (1<<ADATE)|                 //ADC Auto Trigger Enable
    (0<<ADIF)|                  //ADC Interrupt Flag
    (0<<ADIE)|                  //ADC Interrupt Enable
    (1<<ADPS2)|                 //ADC Prescaler of 64
```



```c
    (1<<ADPS1)|
    (0<<ADPS0);
    ADCSRB = 0;                    //free running mode
}


nRF24L01 *setup_rf(void)
{
    nRF24L01 *rf = nRF24L01_init();

    rf->ss.port = &PORTB;
    rf->ss.pin = PB2;
    rf->ce.port = &PORTB;
    rf->ce.pin = PB1;
    rf->sck.port = &PORTB;
    rf->sck.pin = PB5;
    rf->mosi.port = &PORTB;
    rf->mosi.pin = PB3;
    rf->miso.port = &PORTB;
    rf->miso.pin = PB4;
    // interrupt on falling edge of INT0 (PD2)
    EICRA |= _BV(ISC01);
    EIMSK |= _BV(INT0);
    nRF24L01_begin(rf);
    return rf;
}

void read_adc(void)
{

    adc_temp = 0;                           //initalize temp to 0
    ADCSRA |= (1<<ADSC);                    //start the conversion
    while((ADCSRA & (1<<ADIF)) == 0);
    {
        //wait for conversion to finish
    }
    adc_temp = ADCH;                        //get temp value
}
```

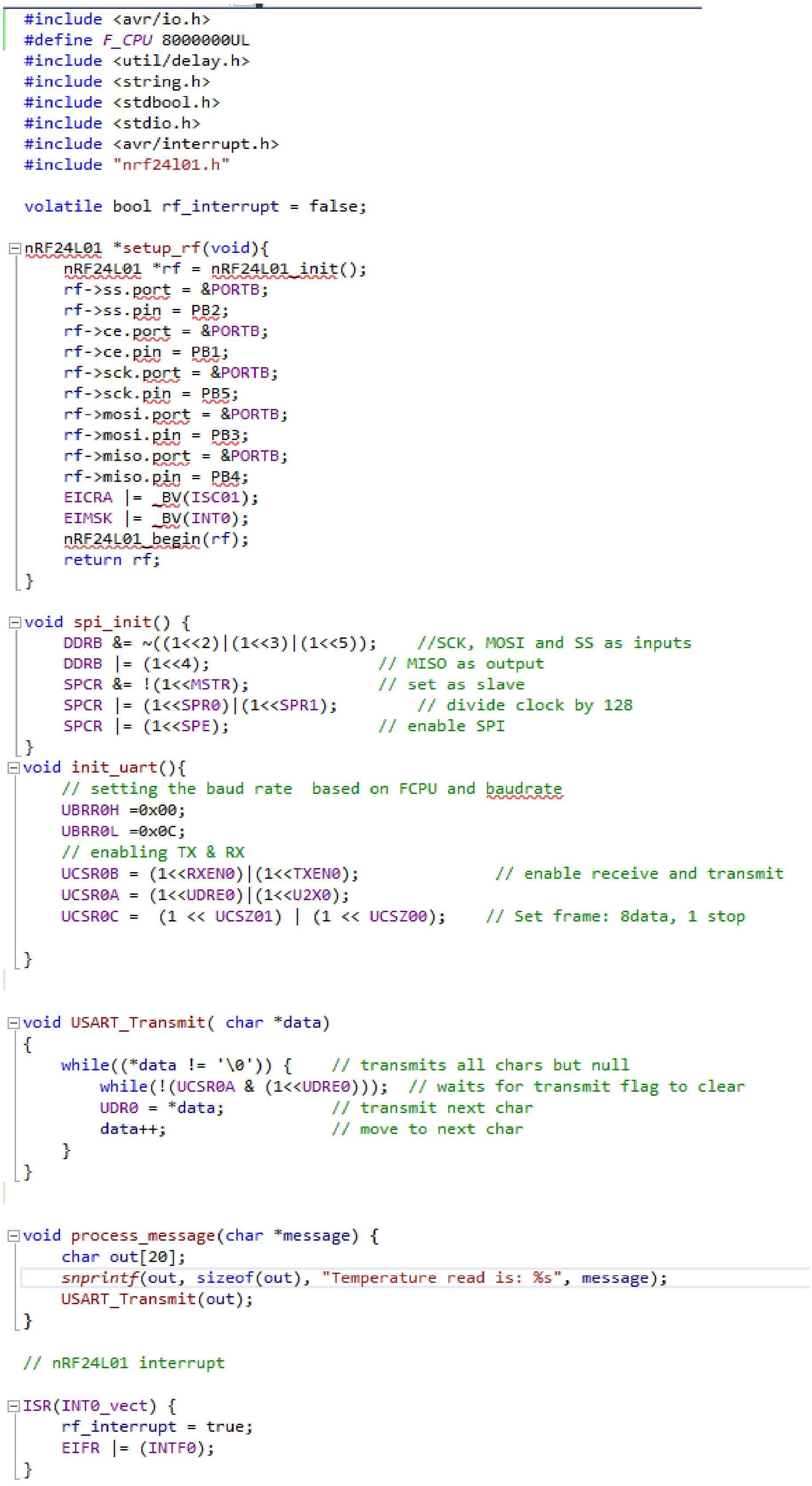


```c
void spi_init(void)
{
    DDRB |= (1<<2)|(1<<3)|(1<<5);       // SCK, MOSI and SS as outputs
    DDRB &= ~(1<<4);                    // MISO as input
    SPCR |= (1<<MSTR);                  // Set as Master
    SPCR |= (1<<SPR0)|(1<<SPR1);        // divided clock by 128
    SPCR |= (1<<SPE);                   // Enable SPI
}

// setup timer to trigger interrupt every second when at 8MHz
void setup_timer(void)
{
    TCCR1B |= _BV(WGM12);
    TIMSK1 |= _BV(OCIE1A);
    OCR1A = 31250;
    TCCR1B |= _BV(CS12);
}


/* INIT USART (RS-232) */
void USART_init( unsigned int ubrr )
{
    UBRR0H = (unsigned char)(ubrr>>8);          //set baud rate
    UBRR0L = (unsigned char)ubrr;
    UCSR0B = (1 << TXEN0) | (1 <<RXEN0);        // Enable receiver, transmitter
    UCSR0C = (1 << UCSZ00) | (1 << UCSZ01);     //asynchronous 8-bit data 1 stop bit
}

/* SEND A STRING TO THE RS-232*/
void USART_tx_string( char *data )
{
    while ((*data != '\0'))
    {
        while (!(UCSR0A & (1 <<UDRE0)))
        {
            //wait for the transmit buffer to empty
        }
        UDR0 = *data;               //put the data into the empty buffer, which sends the data
        _delay_ms(125);             // wait a bit
```

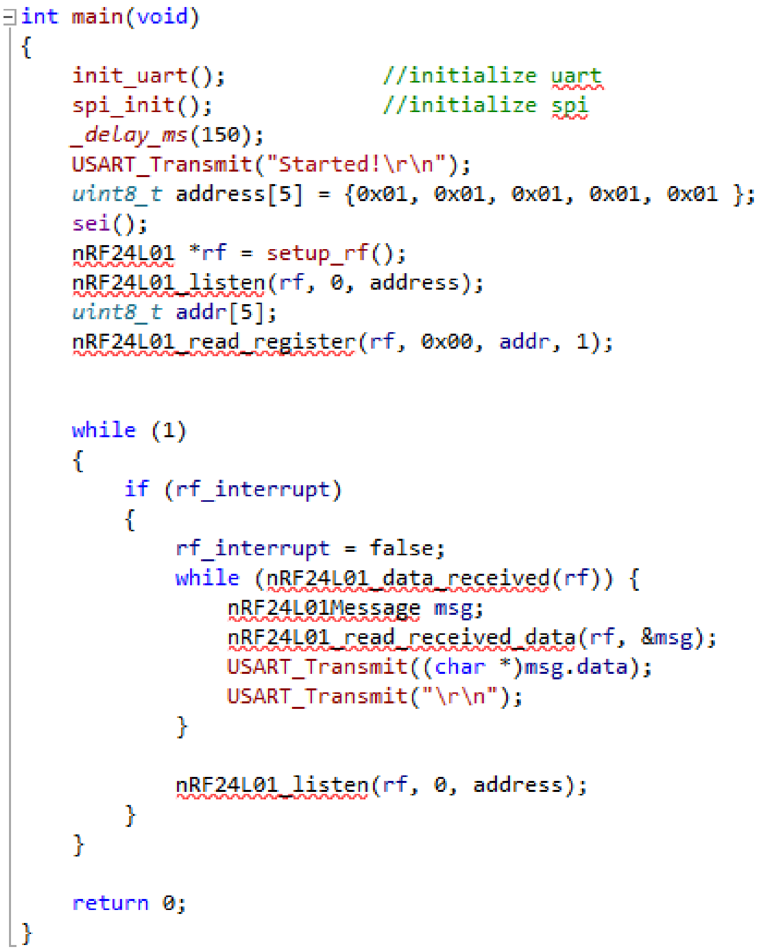


```
        data++;
    }
}

// each one second interrupt
ISR(TIMER1_COMPA_vect)
{
    send_message = true;
    TIFR1 |= (1<<OCF1A);
}

// nRF24L01 interrupt
ISR(INT0_vect)
{
    rf_interrupt = true;
    EIFR |= (1<<INTF0);
}
```

Receiver modified code:



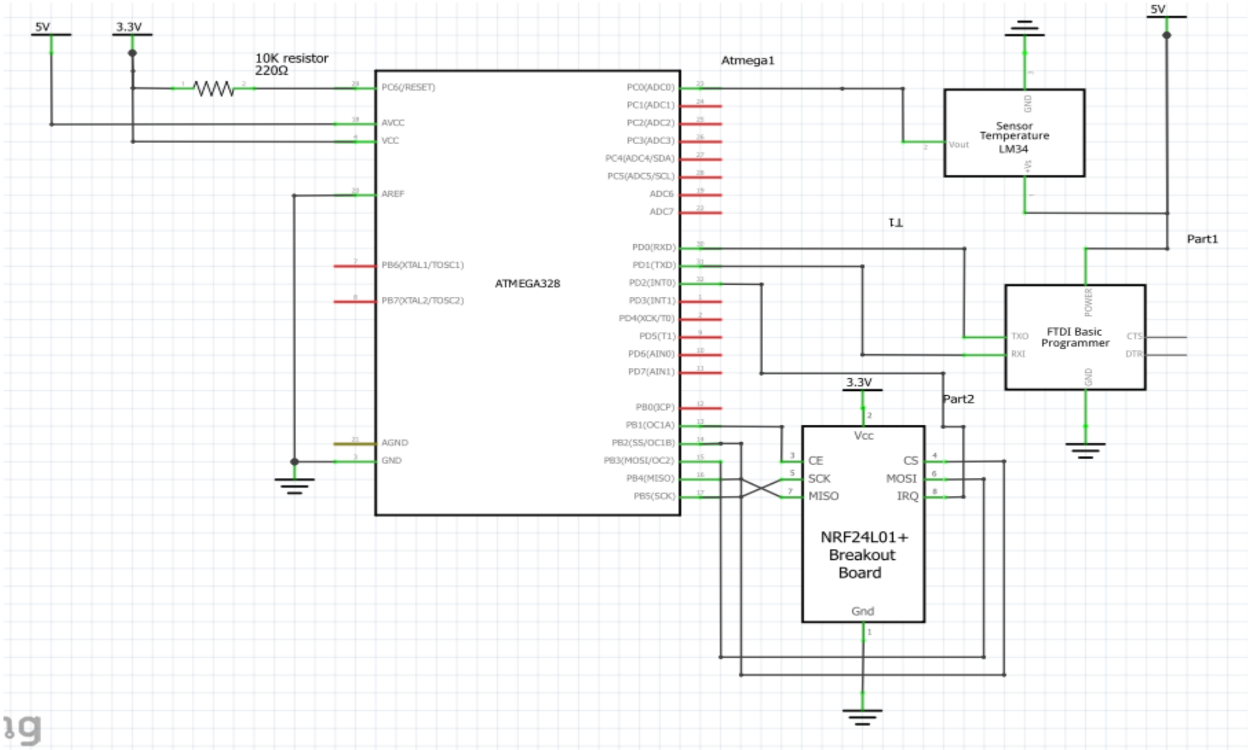
```
#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#include <string.h>
#include <stdbool.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include "nrf24l01.h"

volatile bool rf_interrupt = false;

nRF24L01 *setup_rf(void){
    nRF24L01 *rf = nRF24L01_init();
    rf->ss.port = &PORTB;
    rf->ss.pin = PB2;
    rf->ce.port = &PORTB;
    rf->ce.pin = PB1;
    rf->sck.port = &PORTB;
    rf->sck.pin = PB5;
    rf->mosi.port = &PORTB;
    rf->mosi.pin = PB3;
    rf->miso.port = &PORTB;
    rf->miso.pin = PB4;
    EICRA |= _BV(ISC01);
    EIMSK |= _BV(INT0);
    nRF24L01_begin(rf);
    return rf;
}

void spi_init() {
    DDRB &= ~((1<<2)|(1<<3)|(1<<5));     //SCK, MOSI and SS as inputs
    DDRB |= (1<<4);                  // MISO as output
    SPCR &= !(1<<MSTR);              // set as slave
    SPCR |= (1<<SPR0)|(1<<SPR1);         // divide clock by 128
    SPCR |= (1<<SPE);                // enable SPI
}
void init_uart(){
    // setting the baud rate  based on FCPU and baudrate
    UBRR0H =0x00;
    UBRR0L =0x0C;
    // enabling TX & RX
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);               // enable receive and transmit
    UCSR0A = (1<<UDRE0)|(1<<U2X0);
    UCSR0C =  (1 << UCSZ01) | (1 << UCSZ00);    // Set frame: 8data, 1 stop

}

void USART_Transmit( char *data)
{
    while((*data != '\0')) {    // transmits all chars but null
        while(!(UCSR0A & (1<<UDRE0)));  // waits for transmit flag to clear
        UDR0 = *data;           // transmit next char
        data++;                 // move to next char
    }
}

void process_message(char *message) {
    char out[20];
    snprintf(out, sizeof(out), "Temperature read is: %s", message);
    USART_Transmit(out);
}

// nRF24L01 interrupt

ISR(INT0_vect) {
    rf_interrupt = true;
    EIFR |= (INTF0);
}
```



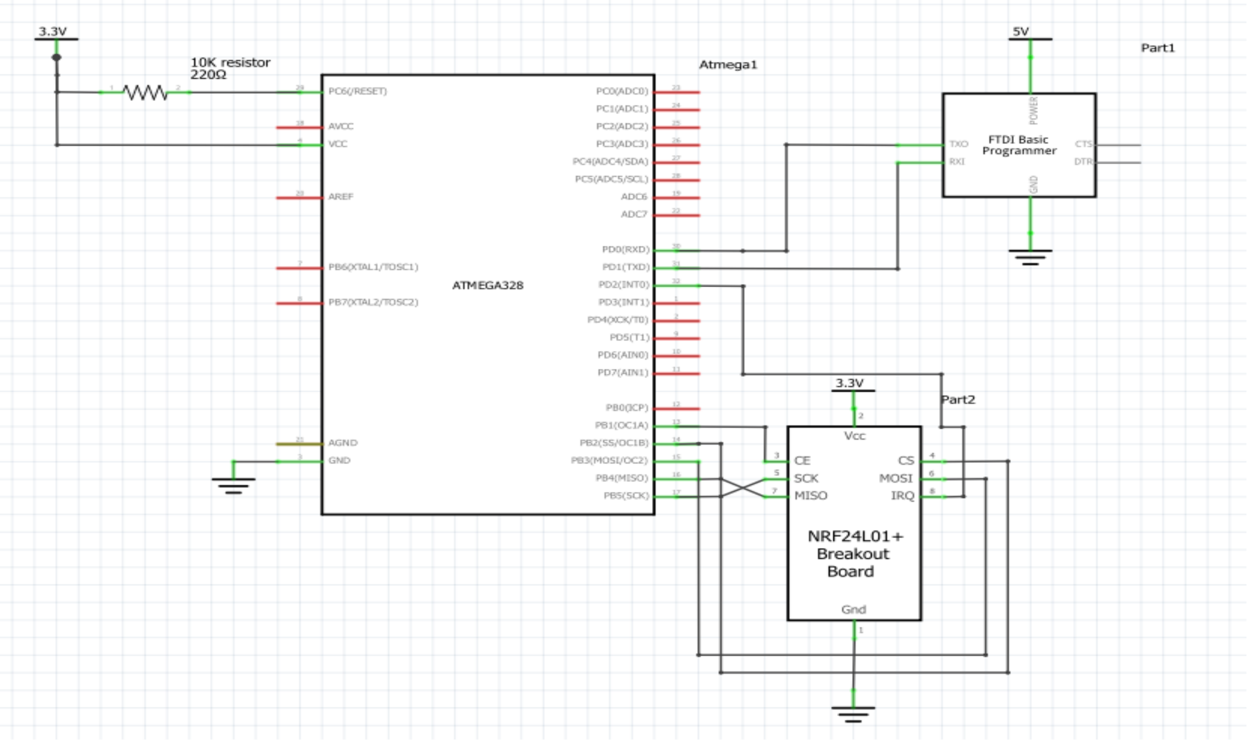
```c
int main(void)
{
    init_uart();            //initialize uart
    spi_init();             //initialize spi
    _delay_ms(150);
    USART_Transmit("Started!\r\n");
    uint8_t address[5] = {0x01, 0x01, 0x01, 0x01, 0x01 };
    sei();
    nRF24L01 *rf = setup_rf();
    nRF24L01_listen(rf, 0, address);
    uint8_t addr[5];
    nRF24L01_read_register(rf, 0x00, addr, 1);


    while (1)
    {
        if (rf_interrupt)
        {
            rf_interrupt = false;
            while (nRF24L01_data_received(rf)) {
                nRF24L01Message msg;
                nRF24L01_read_received_data(rf, &msg);
                USART_Transmit((char *)msg.data);
                USART_Transmit("\r\n");
            }

            nRF24L01_listen(rf, 0, address);
        }
    }

    return 0;
}
```

4. SCHEMATICS

Transmitter Schematic:



Receiver Schematic:

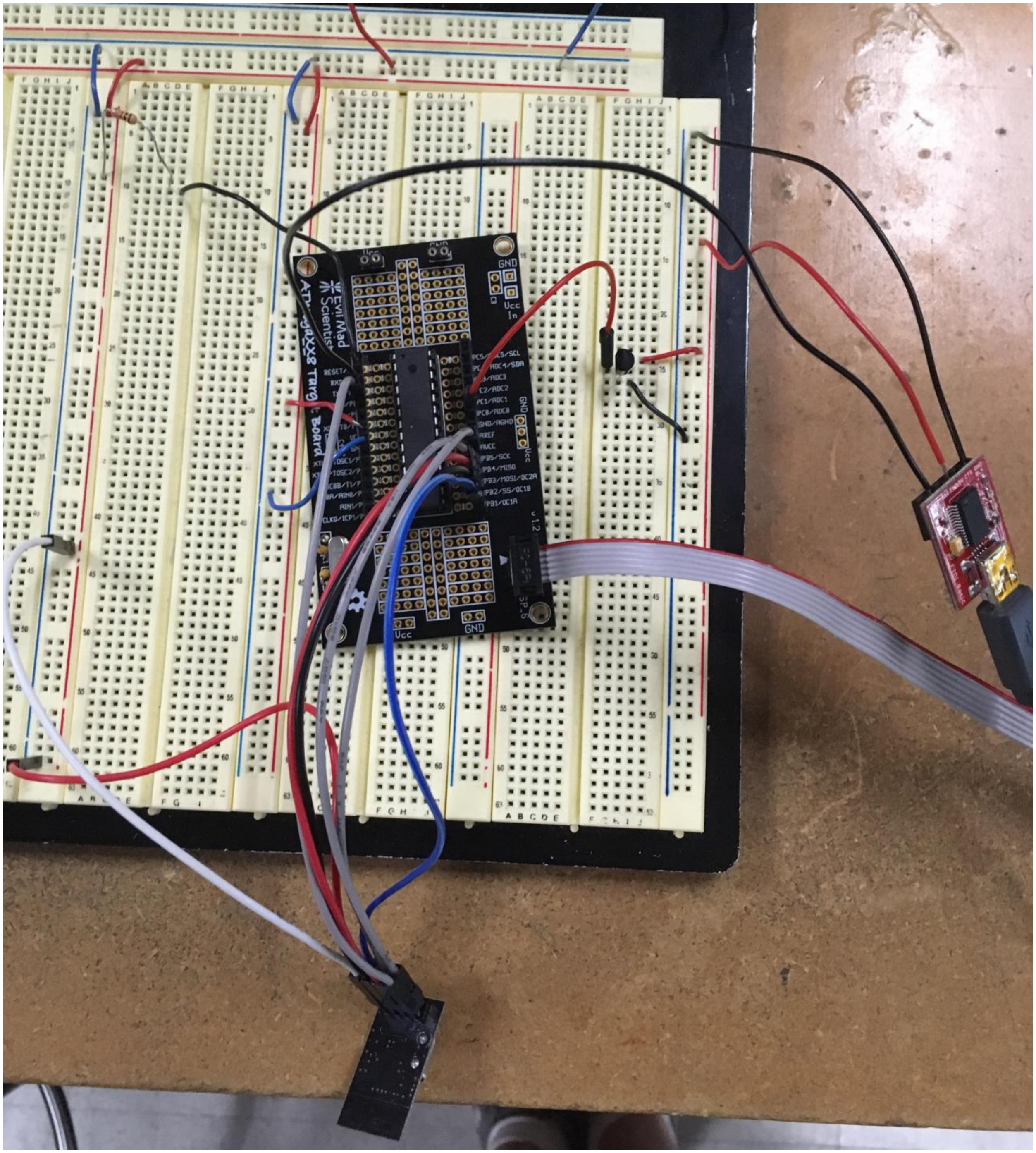


5. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

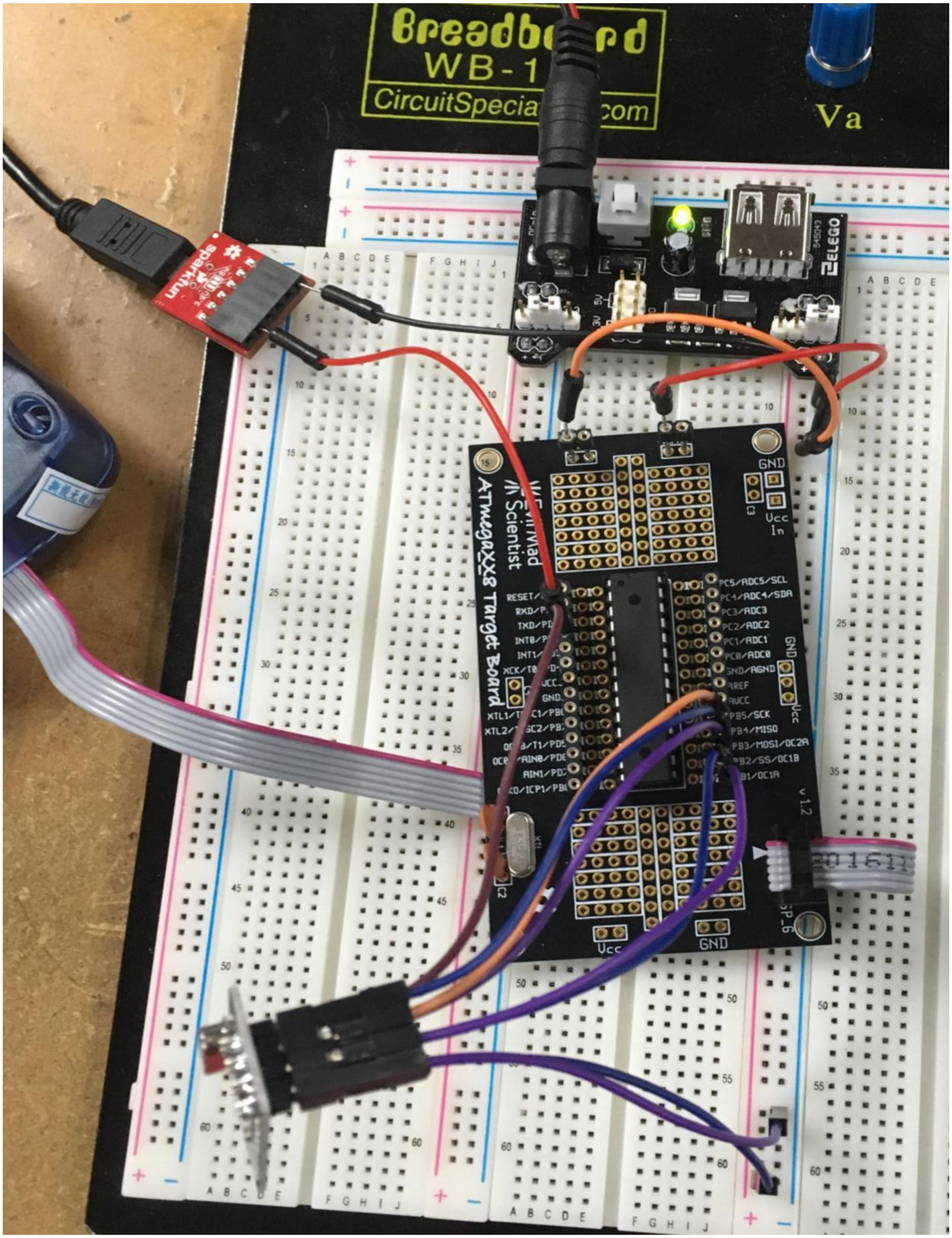
See video for output

6. SCREENSHOT OF EACH DEMO (BOARD SETUP)

Transmitter:



Receiver:


Breadboard
WB-1
CircuitSpecialists.com
Va
sparkfun
ELEGOO
Evil Mad Scientist
ATmegaXX8 Target Board
RESET
RXD
TXD
INT0
INT1
XCK/T0
VCC
GND
XTL1
XTL2
AIN0
AIN1
PC5/ADC5/SCL
PC4/ADC4/SDA
PC3/ADC3
PC2/ADC2
PC1/ADC1
PC0/ADC0
GND/AGND
AREF
AVCC
PB5/SCK
PB4/MISO
PB3/MOSI/OC2A
PB2/SS/OC1B
PB1/OC1A
GND
Vcc In
v1.2
ISP_6
Vcc
GND

7. VIDEO LINKS OF EACH DEMO

Video - https://www.youtube.com/watch?v=LW3HZ5QPJ_w
Playlist - https://www.youtube.com/channel/UCX_dEuWexNMLRw5YqdTRQTg/playlists

8. GITHUB LINK OF THIS DA

https://github.com/nhanuscin/submit/tree/master/DA_Midterm2

Student Academic Misconduct Policy
http://studentconduct.unlv.edu/misconduct/policy.html

"This assignment submission is my own, original work".
Nathan Hanuscin
Partner – Brain Lopez