Brian Lopez
Nathan Hanuscin

CPE301 – SPRING 2018

Design Assignment 4

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

Disclaimer: My partner for this assignment is Nathan Hanuscin

- Atmega328P x 2
- FTDI x 2
- NRF24l01 x 2
- One LM34
- Jumper cables

2. CODE FOR RECEIVER

```
#define F_CPU 8000000UL
#include <avr/io.h>
#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;


// Set up all of the SPI ports with the nRF struct and turn on int0 interrupt
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);		// turn on int0 interrupt
	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 ADC_init() {
       ADMUX = 0;                  // read from port ADC0
       ADMUX |= (1<<REFS0);        // use Vcc for reference
       ADCSRA |= (1<<ADPS2) | (1<<ADPS1); // prescalar of 64
       ADCSRA |= (1<<ADEN);        // enable ADC
       ADCSRB = 0;                 // free running mode
}

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
       }
}

unsigned int readADC()
{
       ADMUX &= ~(1<<ADLAR);               // clear the adc value
       unsigned int val = 0;
       ADCSRA |= (1 << ADSC);              // start adc
       while(ADCSRA & (1<<ADSC));          // wait until adc is done

       val = ADC;
       val = val * 0.427;                // doing (5 * 100 * adc) / 1024, just simplified

       return val;
}

// nRF24L01 interrupt
ISR(INT0_vect) {
       rf_interrupt = true;    // turn on variable for while loop
       EIFR |= (INTF0);   // reset interrupt flag
}

int main(void)
{
    init_uart();         // set UART variables
       ADC_init();          // set ADC variables
       _delay_ms(150);
       USART_Transmit("Started!\r\n");
       uint8_t address[5] = {0x01, 0x01, 0x01, 0x01, 0x01 };   // address for nRF
       sei();               // turn on global interrupts
       nRF24L01 *rf = setup_rf();      // initialze and setup nRF struct
       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);   // gets data that was
sent
                            USART_Transmit((char *)msg.data);     // transmit the
temperature value sent
                            USART_Transmit("\r\n");               // transmit a line feed
                     }

                     nRF24L01_listen(rf, 0, address);
              }
    }

       return 0;
}
```

3. CODE FOR TRANSMITTER

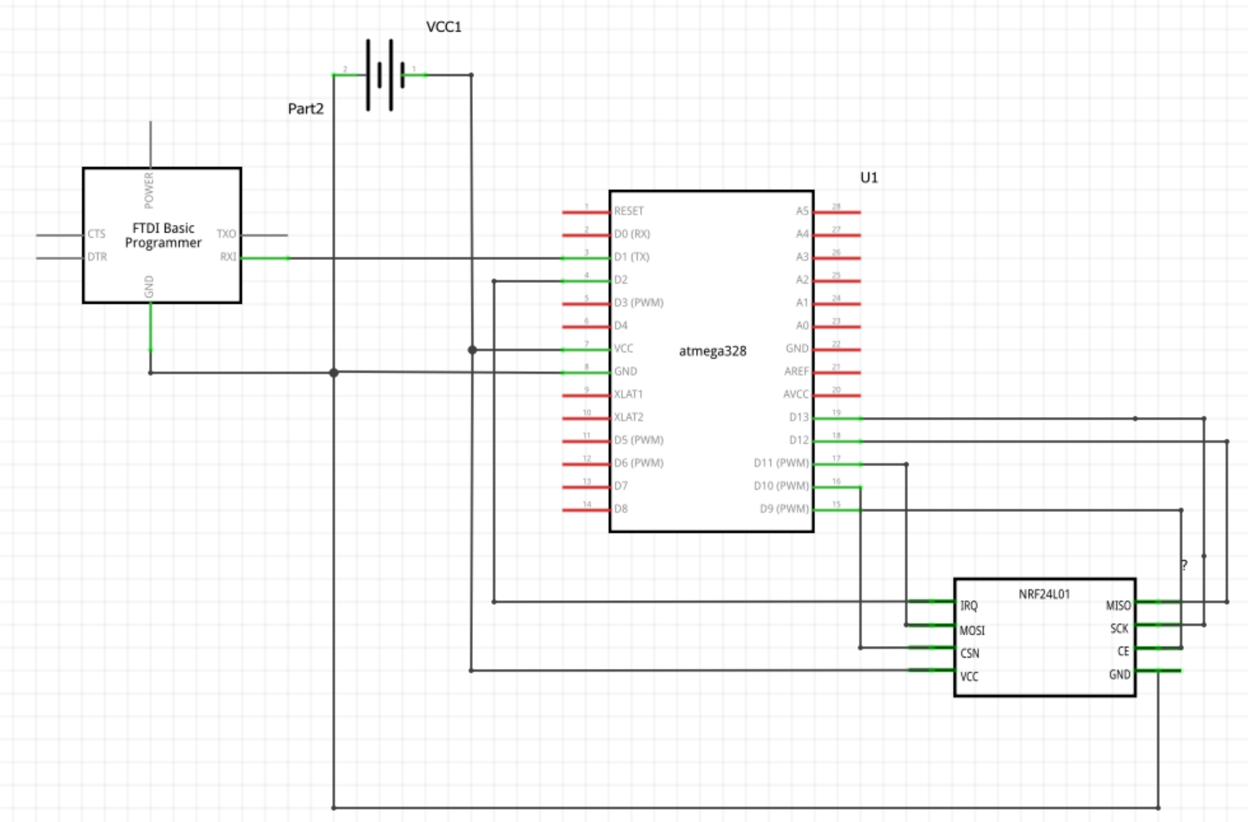
```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include "nrf24l01.h"
#define UBRR_9600 51 // for 8Mhz with .2% error
#define F_CPU 8000000UL
#include <util/delay.h>

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];

int main(void)
{
    uint8_t to_address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };
    spi_init();
    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();                        // start ADC
    setup_timer();                     // setup timer for CTC every 1 second

    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();           // read ADC value
                  send_message = false;
                  nRF24L01Message msg;
                  snprintf(outs,sizeof(outs),"%3d\r\n", adc_temp);
                  USART_tx_string(outs);      // display temperature read
                  memcpy(msg.data, outs, 3);
                  msg.length = strlen((char *)msg.data) + 1;
                  nRF24L01_transmit(rf, to_address, &msg);  // transmit the
temperature value
           }

    }
    return 0;
}

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
    (1<<ADPS1)|
    (0<<ADPS0);

    ADCSRB = 0;
}

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
}

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++;
    }
```

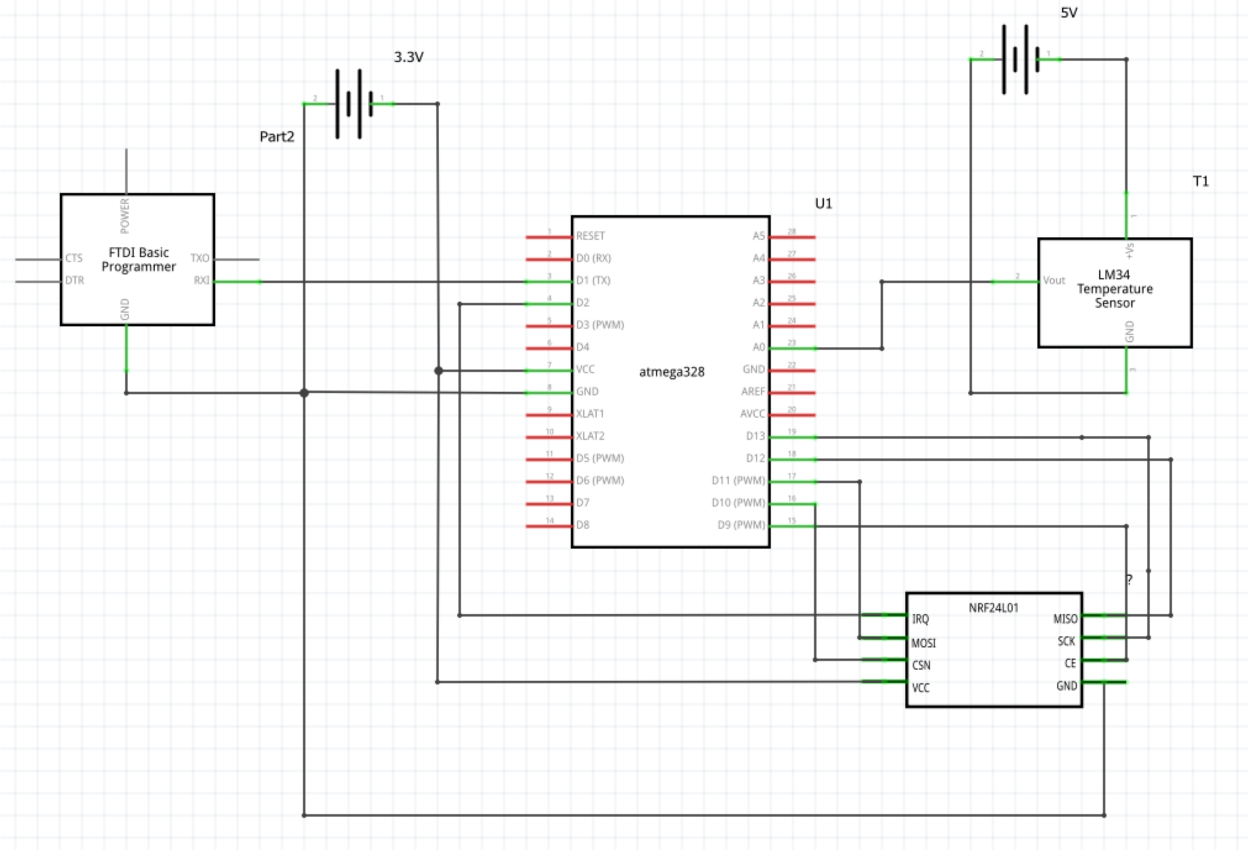
```
}

// 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);
}
```

4. SCHEMATICS

For Receiver:

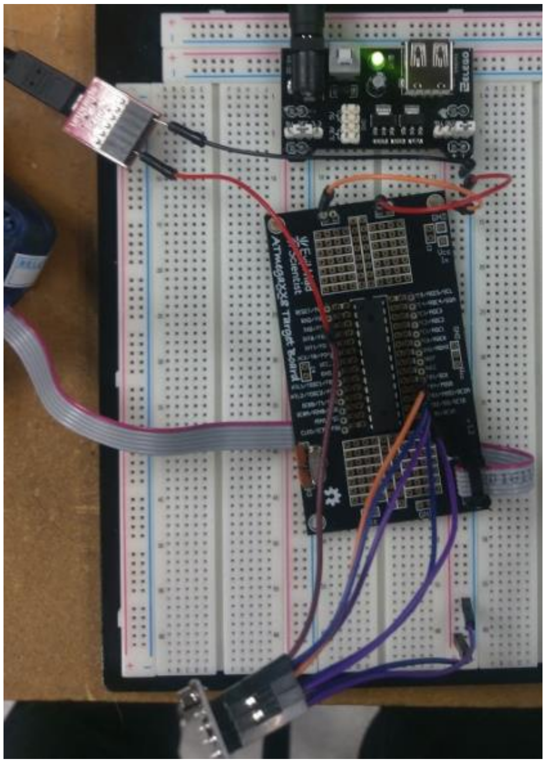


For Transmitter:

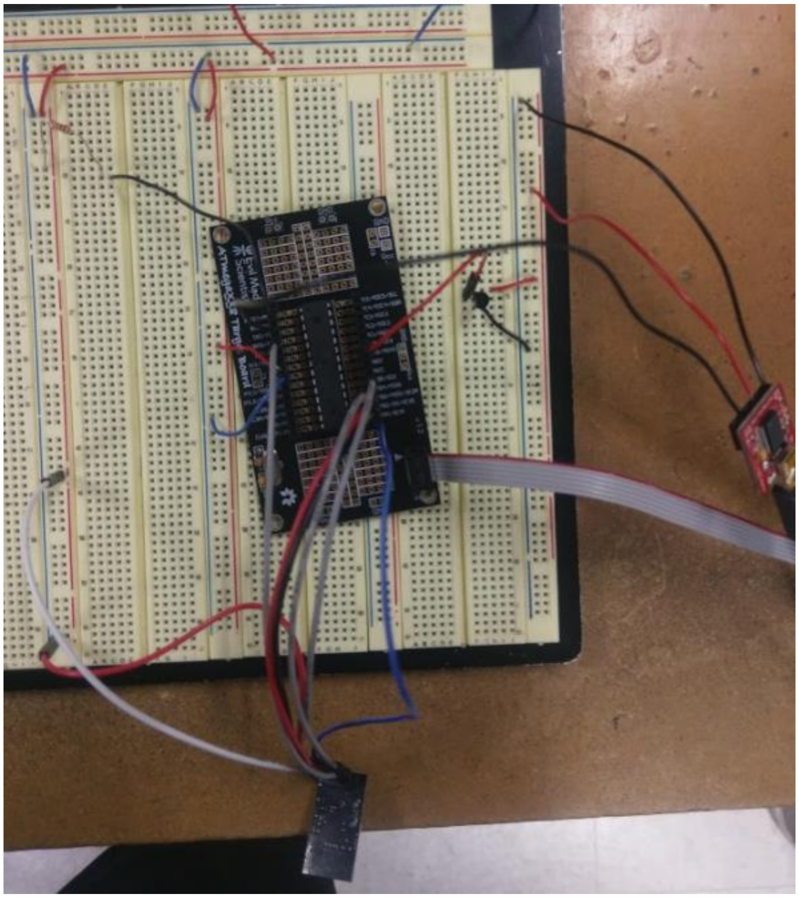


5. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**
   N/A
6. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

Receiver:



Transmitter:



7. VIDEO LINKS OF EACH DEMO

https://www.youtube.com/watch?v=BUFPaYAD4H8

8. GITHUB LINK OF THIS DA

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

"This assignment submission is my own, original work".
Brian Lopez