Jeffrey Razon

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		

Task 1/A. Using the nRF24L01/+ transceiver provided establish a receiver and transmitter node using the ATMEGA328p. Communicate to the nRF24L01 using SPI interface.

Task 2/B. The temperature data (just the whole number) should transmitter from TX node to the RX node and should be displayed on the UART terminal. Both TX and RX should display the temperature data on the UART terminal to validate the transmission.

Full Code (Transmit):

```
#define F_CPU 16000000UL        // CPU Speed for delay
#define FOSC 16000000           // Clock speed
#define BAUD 115200             // Desire baud rate
#define MYUBRR FOSC/8/BAUD-1    // Formula to set the baud rate [Double Transmission Rate]

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "nrf24l01-mnemonics.h"
#include "nrf24l01.h"

void setup_timer(void);
void UART_TX(char *data);
nRF24L01 *setup_rf(void);

volatile bool rf_interrupt = false;
volatile bool send_message = false;
volatile char ADCvalue;

int main(void)
{
    // Variables
    uint8_t i = 0;              // Iterative variable
    char temp[5];               // Temperature string buffer
    char LF = '\n';             // Line feed
    DDRC = (1 << 5);

    // NRF Settings
    uint8_t to_address[5] = {0x02, 0x04, 0x06, 0x08, 0x0A};     // Device address
    sei();                      // Global interrupt enabled
    nRF24L01 *rf = setup_rf();  // Initialize the NRF
    setup_timer();

    // ADC Settings
    ADMUX = 0;                  // Use ADC0
    ADMUX |= (1 << REFS0);      // AVcc is reference with ARef connected to external capacitor
    ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);   // 16 MHz with prescalar of 128
    ADCSRA |= (1 << ADATE);     // Enable auto trigger
    ADCSRB = 0;                 // Free running settings for auto trigger
    ADCSRA |= (1 << ADEN);      // Enable ADC
    ADCSRA |= (1 << ADSC);      // Start conversion

    // UART Settings
    UBRR0H = ((MYUBRR) >> 8);   // Set baud rate for UPPER Register
    UBRR0L = MYUBRR;            // Set baud rate for LOWER Register
    UCSR0A |= (1 << U2X0);      // Double UART transmission speed
    UCSR0B |= (1 << TXEN0);     // Enable transmitter
    UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);    // Frame: 8-bit Data and 1 Stop bit
```

```
    while (1)
    {
        // ADC Conversion
        while((ADCSRA & (1 << ADIF)) == 0);     // Wait for ADC conversion
        ADCvalue = (ADC >> 1);                      // Assign the temperature

        // Check if the message has been successfully transmitted
        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;               // Reset message flag
            nRF24L01Message msg;                // Message structure to be transmitted
            i = 0;                              // Reset iterative variable
            itoa(ADCvalue, temp, 10);           // Convert integer value into ASCII

            // Transmit the temperature to terminal [NULL terminated]
            while(temp[i] != 0)
            {
                UART_TX(&temp[i]);              // Sends temperature to terminal
                i++;                            // Size of the temperature string
            }
            UART_TX(&LF);                       // Line feed

            // NRF Transmission
            msg.length = i + 1;                         // Save the length of the temperature string
            memcpy(msg.data, temp, msg.length);         // Copy the string into the struct
            nRF24L01_transmit(rf, to_address, &msg);    // Transmit the temperature
        }
    }
    return 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;
    EICRA |= _BV(ISC01);
    EIMSK |= _BV(INT0);
    nRF24L01_begin(rf);
    return rf;
}
```

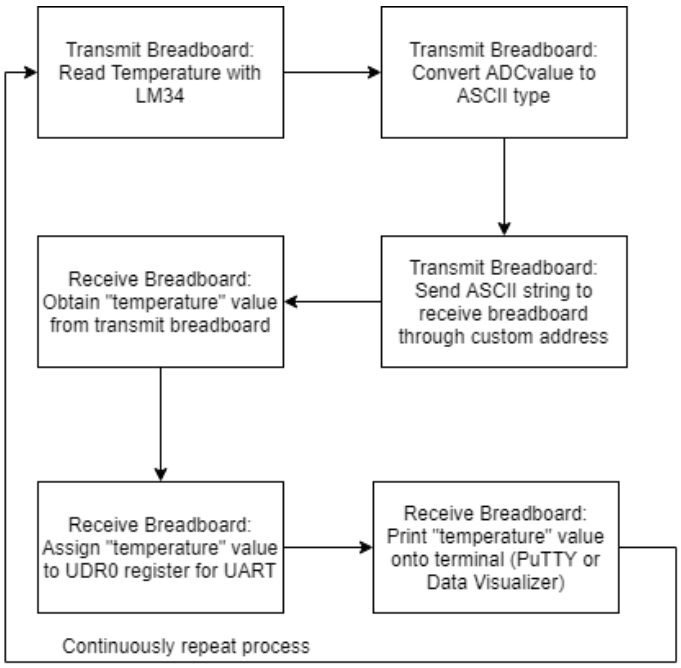
```
void setup_timer(void)
{
    TCCR1B |= _BV(WGM12);                // CTC Mode
    TIMSK1 |= _BV(OCIE1A);               // COMP Interrupt
    OCR1A = 15624;                       // 16 MHz/1024
    TCCR1B |= _BV(CS10) | _BV(CS12);     // Prescalar 1024
}

 // UART transmission function
void UART_TX(char *data)
{
    while(!(UCSR0A & (1 << UDRE0)));     // Wait for UART to be available
    UDR0 = *data;                        // Send the data
}

ISR(TIMER1_COMPA_vect)
{
    send_message = true;
    PORTC ^= (1 << 5);
}

ISR(INT0_vect)
{
    rf_interrupt = true;
}
```

Full Code (Receive):

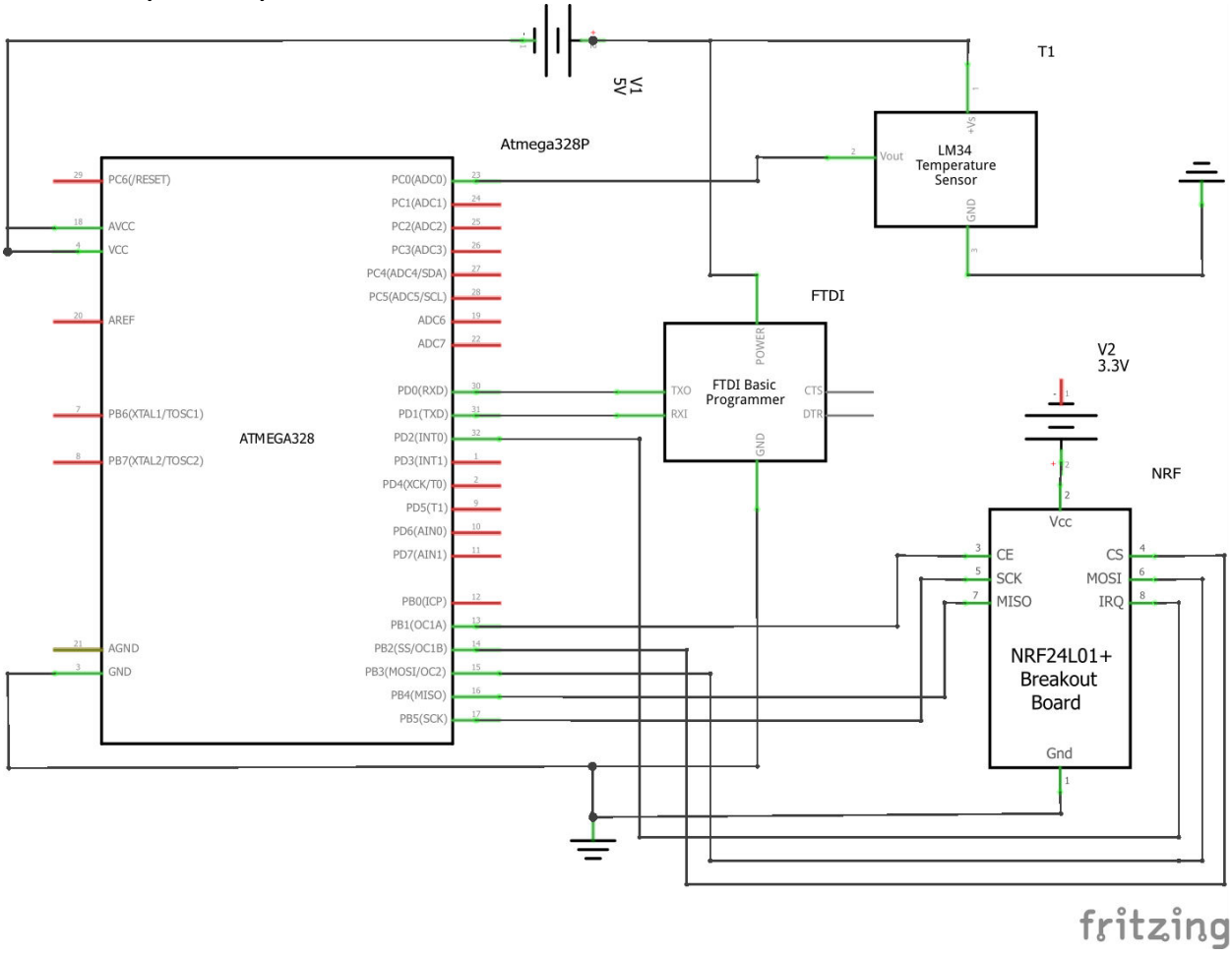
```
#define F_CPU 16000000UL        // CPU Speed for delay
#define FOSC 16000000           // Clock speed
#define BAUD 115200             // Desire baud rate
#define MYUBRR FOSC/8/BAUD-1    // Formula to set the baud rate [Double Transmission]

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "nrf24l01-mnemonics.h"
#include "nrf24l01.h"

nRF24L01 *setup_rf(void);
void process_message(char *message, uint8_t len);
void UART_TX(char *data);

volatile bool rf_interrupt = false;

int main(void)
{
    // NRF Settings
    uint8_t address[5] = {0x02, 0x04, 0x06, 0x08, 0x0A};
    //prepare_led_pin();
    sei();
    nRF24L01 *rf = setup_rf();
    nRF24L01_listen(rf, 0, address);
    uint8_t addr[5];
    nRF24L01_read_register(rf, CONFIG, addr, 1);

    // USART Settings
    UBRR0H = ((MYUBRR) >> 8);   // Set baud rate for UPPER Register
    UBRR0L = MYUBRR;            // Set baud rate for LOWER Register
    UCSR0A |= (1 << U2X0);      // Double UART transmission speed
    UCSR0B |= (1 << TXEN0);     // Enable transmitter
    UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);    // Frame: 8-bit Data and 1 Stop bit
```

```
    while (1)
    {
        if(rf_interrupt)
        {
            rf_interrupt = false;
            while(nRF24L01_data_received(rf))
            {
                nRF24L01Message msg;
                nRF24L01_read_received_data(rf, &msg);
                process_message((char *)msg.data, msg.length);
            }
            nRF24L01_listen(rf, 0, address);
        }
    }
    return 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;
    EICRA |= _BV(ISC01);
    EIMSK |= _BV(INT0);
    nRF24L01_begin(rf);
    return rf;
}

void process_message(char *message, uint8_t len)
{
    uint8_t i = 0;      // Iterative variable
    char LF = '\n';     // Line Feed

    // Process the temperature and display to UART
    for(i = 0; i < len; i++)
    {
        UART_TX(&message[i]);
    }
    UART_TX(&LF);
}

ISR(INT0_vect)
{
    rf_interrupt = true;
}

// UART transmission function
void UART_TX(char *data)
{
    while(!(UCSR0A & (1 << UDRE0)));    // Wait for UART to be available
    UDR0 = *data;                       // Send the data
}
```

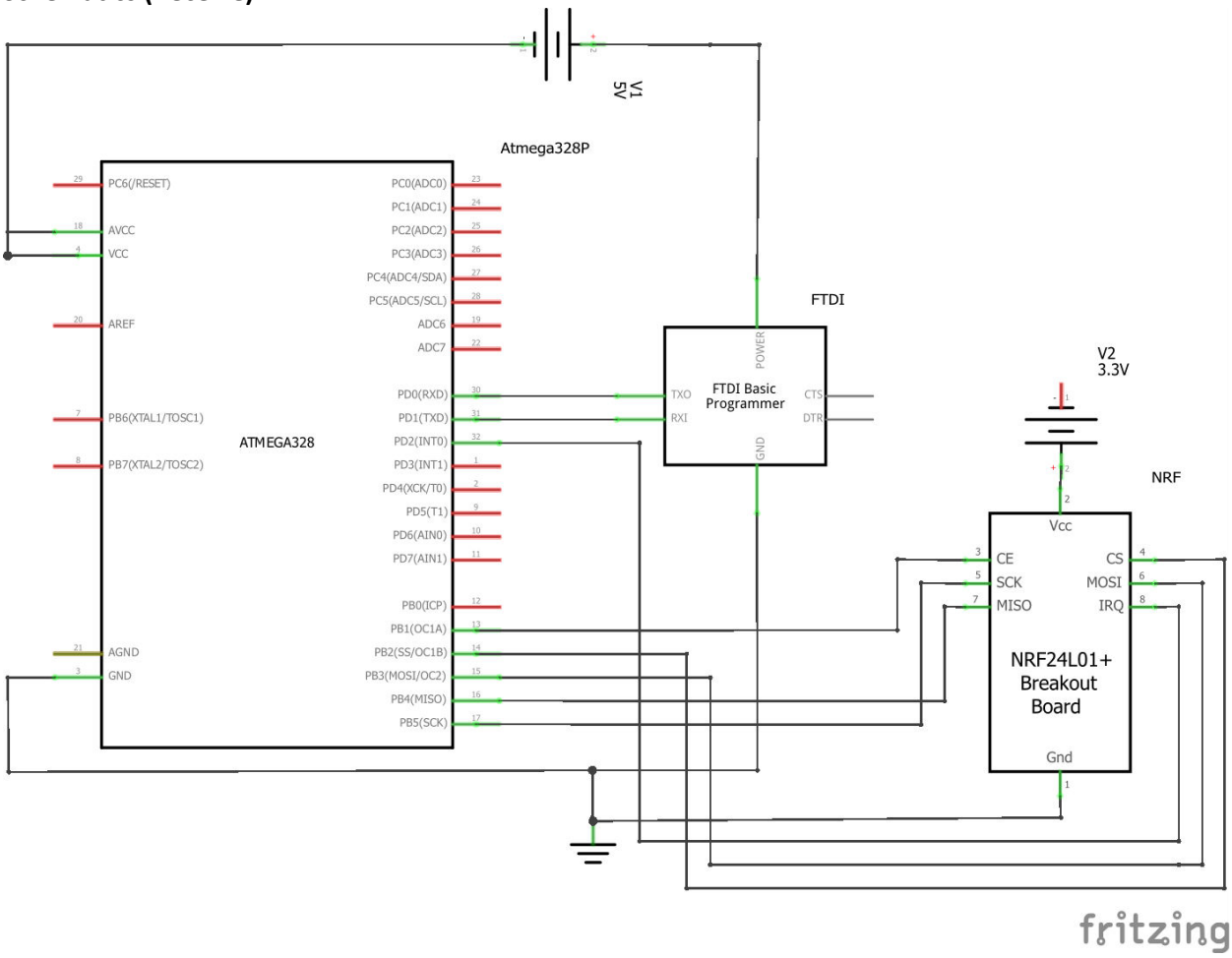
Flowchart:



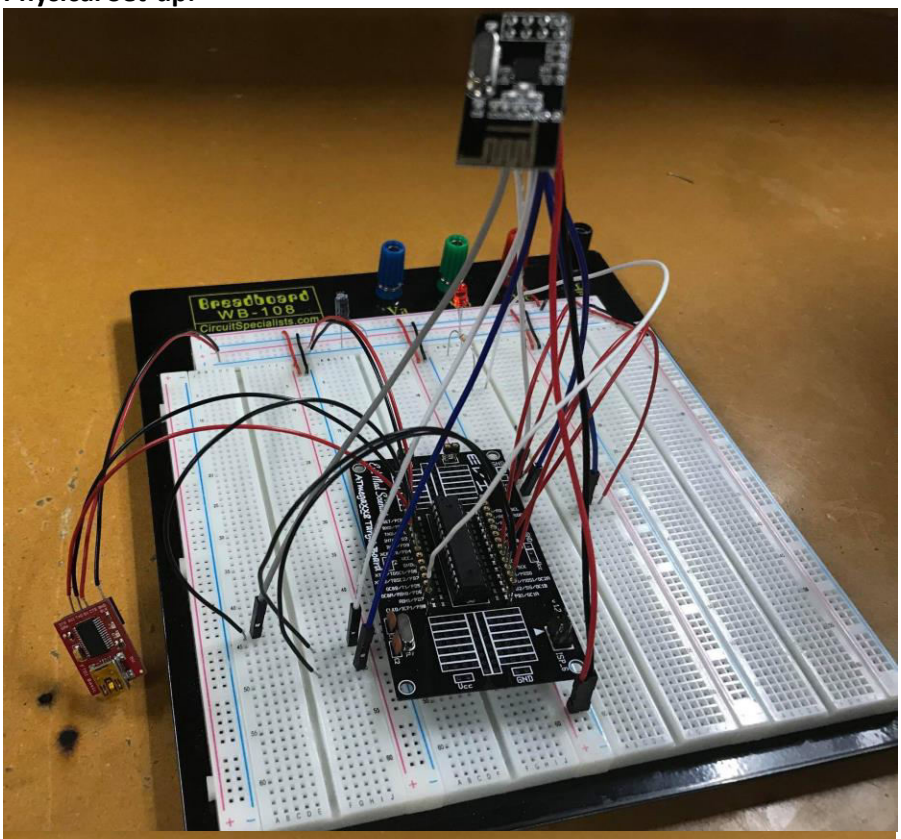
Schematics (Transmit):

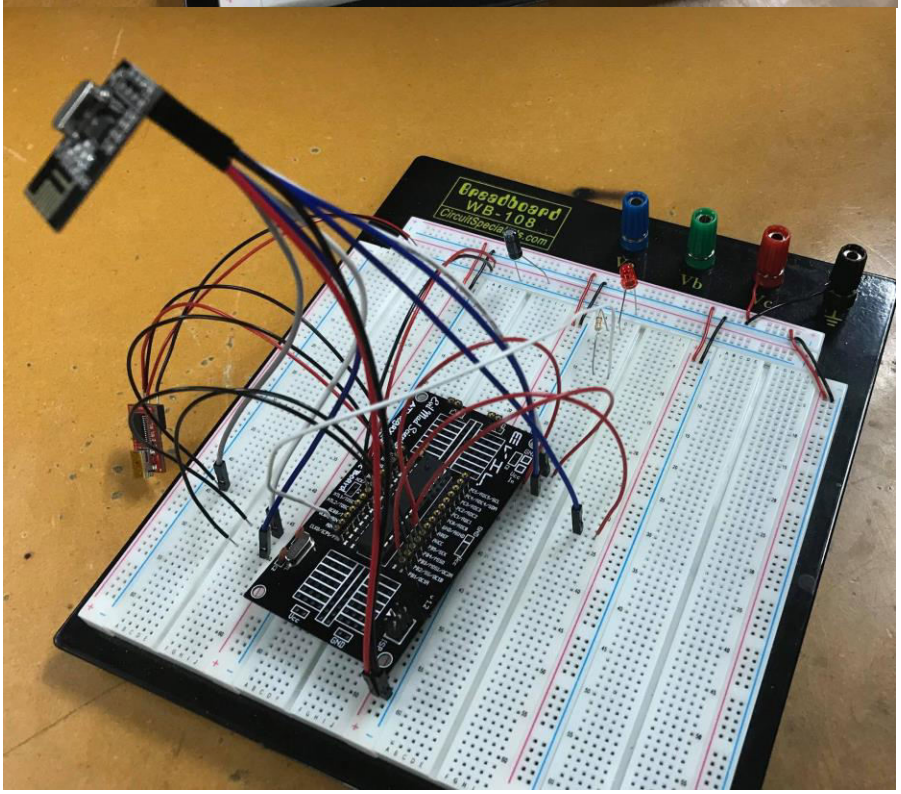


Schematics (Receive):



Physical Set-up:





GITHUB LINK: https://github.com/JeffinVegas/EmbSys.git

YOUTUBE LINK: In the videos_DA_MIDTERM2.txt file
Group:
Jeffrey R.
Argenis J.
Bryan T.
Tenniel T.

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

"*This assignment submission is my own, original work*".
Jeffrey Razon