Eilat Avidan

CPE301 – SPRING 2018

Design Assignment 3

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		

- **Since I did not find a partner to do the midterm with, I borrowed a breadboard with the microcontroller from a class mate and did the midterm by myself.**

1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

- Atmega328P
- Breadboard
- Resistors
- Power supply
- FTDI chip
- NRF24l01

2. INITIAL/DEVELOPED CODE OF TASK 1/A

Transmit code:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL
#define UBRR_9600 51
#include <util/delay.h>
#include <stdbool.h>
#include <string.h>
#include "nrf24l01.h"

void setup_timer(void);
nRF24L01 *setup_rf(void);

volatile unsigned int adc_temp;              //variable to send the ADC value
char outs[20];
volatile bool rf_interrupt = false;
volatile bool send_message = false;

void init_uart(unsigned int ubrr)
{
UBRR0H = (unsigned char)(ubrr>>8);           //set baud rate
UBRR0L = (unsigned char)ubrr;
UCSR0B = (1<<TXEN0) | (1<<RXEN0);            //set transmitter/ receiver
UCSR0C = (1<<UCSZ00) | (1<<UCSZ01);          //USART mode select

}

void USART_tx_string( char *data ) {         //print string
while ((*data != '\0')) {
while (!(UCSR0A & (1 <<UDRE0)));
UDR0 = *data;
data++;
}
}

void adc_init(void)
{
/** Setup and enable ADC **/
ADMUX = (0<<REFS1)|                   // Reference Selection Bits
(1<<REFS0)|                           // AVcc - external cap at AREF
```

```
(1<<ADLAR)|                         // ADC left Adjust Result
(0<<MUX2)|                          // Analog Channel Selection Bits
(0<<MUX1)|                          // ADC0 Pin
(0<<MUX0);

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

}

/* READ ADC PINS*/
void read_adc(void)
{
unsigned char i =4;                 //set i to 4- make 4 readings
adc_temp = 0;                       //initialize ADC_TEMP
while (i--)
{
ADCSRA |= (1<<ADSC);
while((ADCSRA & (1<<ADIF)) == 0);
adc_temp += ADCH;                   //sum up 4 readings
_delay_ms(50);
}
adc_temp = adc_temp / 4;            // Average of four samples

}

int main(void) {
uint8_t to_address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };
bool on = false;
adc_init();                         //initialize ADC
sei();
nRF24L01 *rf = setup_rf();
setup_timer();

while (true) {
    read_adc();                     //keep reading temperate from LM34
    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)
{
    snprintf(outs,sizeof(outs),"%3d\r\n", adc_temp);// print ADC value
    memcpy(msg.data, outs , 3);                     //copy outs to msg.data
    USART_tx_string((char *)msg.data);              //print msg.data
    USART_tx_string("F\r\n");                   //print F and line feed
}
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;
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 code:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <string.h>
#include "nrf24l01.h"
#include "nrf24l01-mnemonics.h"
#define F_CPU 8000000UL
#define UBRR_9600 51
#include <util/delay.h>

nRF24L01 *setup_rf(void);
volatile bool rf_interrupt = false;

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(unsigned int ubrr)
{
	UBRR0H = (unsigned char)(ubrr>>8);        //set baud rate
	UBRR0L = (unsigned char)ubrr;
	UCSR0B = (1<<TXEN0) | (1<<RXEN0);
	UCSR0C = (1<<UCSZ00) | (1<<UCSZ01);

}

void USART_tx_string( char *data ) {
	while ((*data != '\0')) {
		while (!(UCSR0A & (1 <<UDRE0)));
		UDR0 = *data;
		data++;
	}
}


void ADC_init ()
{
	ADMUX = 0;
	ADMUX |= (1<<REFS0);
	ADCSRA |= (1<<ADPS2) | (1<<ADPS1) |(1<<ADEN);
	ADCSRB = 0;
}

int main(void) {
	init_uart(UBRR_9600);
	spi_init();
	_delay_ms(500);
	uint8_t address[5] = { 0x01, 0x01, 0x01, 0x01, 0x01 };
	sei();
	USART_tx_string("connected...\r\n");
```

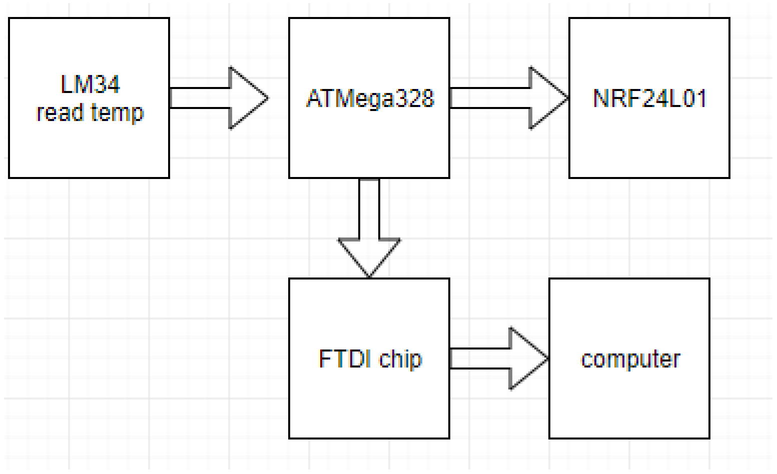
```c
	nRF24L01 *rf = setup_rf();
	nRF24L01_listen(rf, 0, address);
	uint8_t addr[5];
	nRF24L01_read_register(rf, 0x00, addr, 1);
	while (true) {
		if (rf_interrupt) {
			rf_interrupt = false;
			while (nRF24L01_data_received(rf)) {
				nRF24L01Message msg;
				nRF24L01_read_received_data(rf, &msg);
				USART_tx_string((char *)msg.data);
				USART_tx_string("F\r\n");
			}
			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;
	// interrupt on falling edge of INT0 (PD2)
	EICRA |= _BV(ISC01);
	EIMSK |= _BV(INT0);
	nRF24L01_begin(rf);
	return rf;
}



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

3. Flow chart

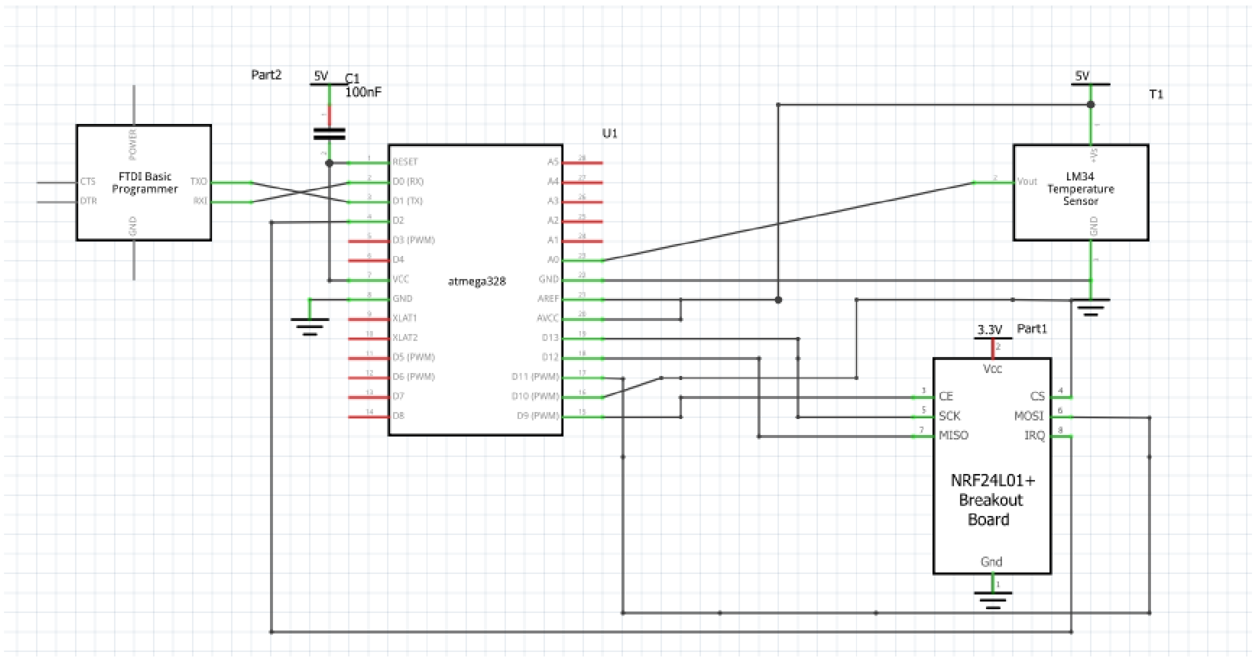
Transmit:



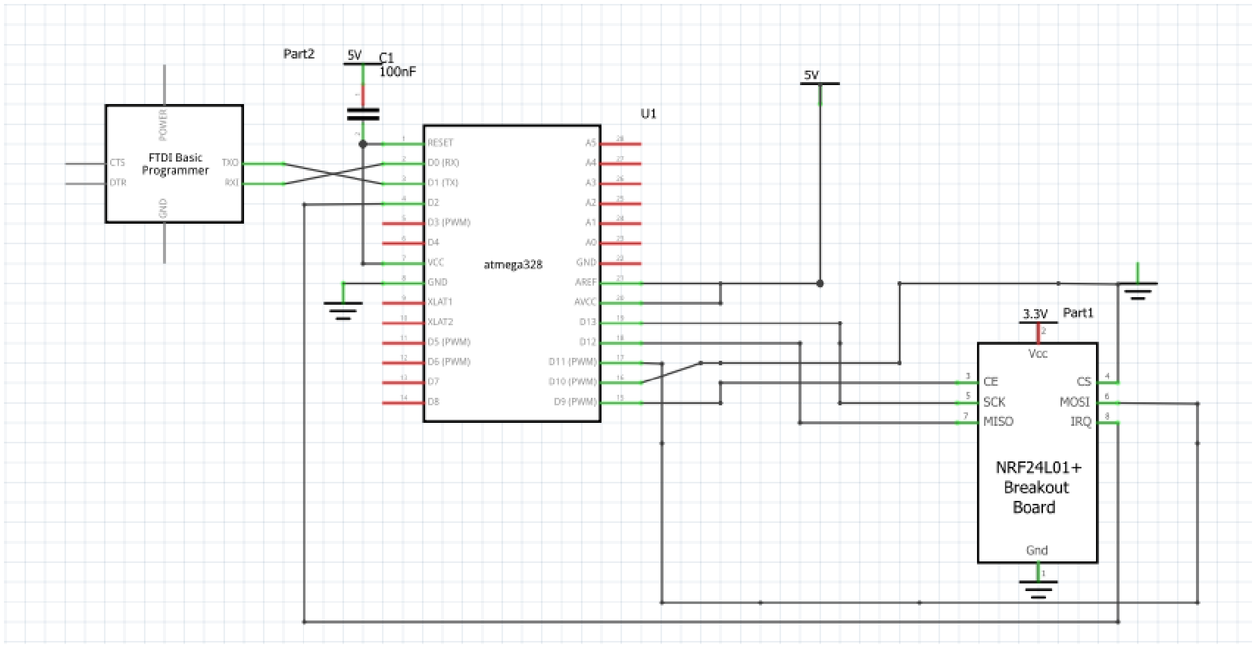
The LM34 is connected to the microcontroller through ADC0 pin and the microcontroller is connected to the FTDI chip to print the value to Putty as well as to the NRF24L01 chip to transfer the data to the receiver to print the temperature to Putty.

The receiver chart flow is identical to the transmit beside the temperature sensor.

4. Schematic- transmit:



Schematic- receiver:



5. SCREENSHOT OF EACH DEMO (BOARD SETUP)

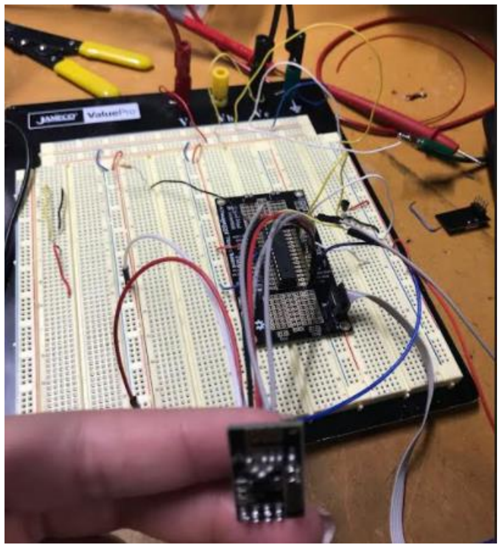


Figure 1 transmit

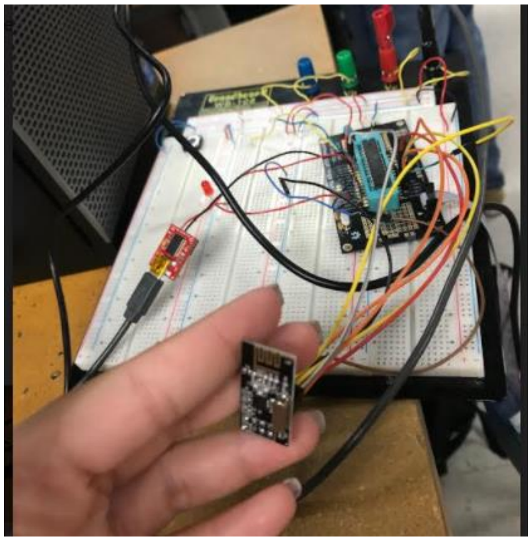


Figure 2 receiver

6. VIDEO LINKS OF EACH DEMO

https://youtu.be/OhaJ_UHHY1Q

7. GITHUB LINK OF THIS DA

git@github.com:EilatAvidan/microcon.git

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

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