CPE301 – SPRING 2019

Design Assignment 5

Student Name: Benjamin Johnson

Student #: 5003284489

Student Email: johnsb18@unlv.nevada.edu

Primary Github address: https://github.com/johnsb18/ClassRepository

Directory:

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Atmega328P

LM35 temperature sensor

NRF24L01

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

#ifndef *F\_CPU*

#define *F\_CPU* 16000000UL

#endif

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// nRF24L01+ include files

#include "nrf24l01.h"

#include "nrf24l01-mnemonics.h"

#include "spi.h"

// Settings

*uint8\_t* rx\_address[5] = { 0x54, 0x54, 0x54, 0x54, 0x54 }; // Read pipe address

*uint8\_t* tx\_address[5] = { 0xf7, 0xf7, 0xf7, 0xf7, 0xf7 }; // Write pipe address

#define READ\_PIPE 0 // Number of read pipe

//

// -AUTO\_ACK can be disabled when running on 2MBPS @ <= 32 byte messages.

// -250KBPS and 1MBPS with AUTO\_ACK disabled lost many packets

// if the packet size was bigger than 4 bytes.

// -If AUTO\_ACK is enabled, tx\_address = rx\_address.

//

#define AUTO\_ACK false // Auto acknowledgment

#define DATARATE RF\_DR\_2MBPS // 250kbps, 1mbps, 2mbps

#define POWER POWER\_MAX // Set power (MAX 0dBm..HIGH -6dBm..LOW -12dBm.. MIN -18dBm)

#define CHANNEL 0x74 // 2.4GHz-2.5GHz channel selection (0x01 - 0x7C)

#define DYN\_PAYLOAD true // Dynamic payload enabled

#define CONTINUOUS false // Continuous carrier transmit mode (not tested)

//

// ISR(INT0\_vect) is triggered depending on config (only one can be true)

//

#define RX\_INTERRUPT true // Interrupt when message is received (RX)

#define TX\_INTERRUPT false // Interrupt when message is sent (TX)

#define RT\_INTERRUPT false // Interrupt when maximum re-transmits are reached (MAX\_RT)

//

// -PIN map.

// -If CE or CSN is changed to different PIN e.g. PC0

// then change DDRB -> DDRC, PORTB -> PORTC and so on

//

// CE

#define CE\_DDR DDRB

#define CE\_PORT PORTB

#define CE\_PIN DDB1 // CE connected to PB1

// CSN

#define CSN\_DDR DDRB

#define CSN\_PORT PORTB

#define CSN\_PIN DDB2 // CSN connected to PB2

// IRQ

#define IRQ\_DDR DDRD

#define IRQ\_PORT PORTD

#define IRQ\_PIN DDD2 // IRQ connected to PD2

// MOSI

#define MOSI\_DDR DDRB

#define MOSI\_PORT PORTB

#define MOSI\_PIN DDB3

// MISO

#define MISO\_DDR DDRB

#define MISO\_PORT PORTB

#define MISO\_PIN DDB4

// SCK

#define SCK\_DDR DDRB

#define SCK\_PORT PORTB

#define SCK\_PIN DDB5

// PIN toggling

#define setbit(port, bit) (port) |= (1 << (bit))

#define clearbit(port, bit) (port) &= ~(1 << (bit))

#define ce\_low clearbit(CE\_PORT,CE\_PIN)

#define ce\_high setbit(CE\_PORT,CE\_PIN)

#define csn\_low clearbit(CSN\_PORT,CSN\_PIN)

#define csn\_high setbit(CSN\_PORT,CSN\_PIN)

// Used to store SPI commands

*uint8\_t* data;

*uint8\_t* nrf24\_send\_spi(*uint8\_t* register\_address, void \*data, unsigned int bytes)

{

*uint8\_t* status;

csn\_low;

status = spi\_exchange(register\_address);

for (unsigned int i = 0; i < bytes; i++)

((*uint8\_t*\*)data)[i] = spi\_exchange(((*uint8\_t*\*)data)[i]);

csn\_high;

return status;

}

*uint8\_t* nrf24\_write(*uint8\_t* register\_address, *uint8\_t* \*data, unsigned int bytes)

{

return nrf24\_send\_spi(W\_REGISTER | register\_address, data, bytes);

}

*uint8\_t* nrf24\_read(*uint8\_t* register\_address, *uint8\_t* \*data, unsigned int bytes)

{

return nrf24\_send\_spi(R\_REGISTER | register\_address, data, bytes);

}

void nrf24\_init(void)

{

// Interrupt on falling edge of INT0 (PD2) from IRQ pin

cli(); // Disable interrupts

EICRA |= (1 << ISC01);

EIMSK |= (1 << INT0);

sei(); // Enable interrupts

// CSN and CE as outputs and initial states

setbit(CE\_DDR,CE\_PIN);

setbit(CSN\_DDR,CSN\_PIN);

csn\_high;

ce\_low;

// Initialize SPI

spi\_master\_init();

*\_delay\_ms*(100); // Power on reset 100ms

// Start nRF24L01+ config

data =

(!(RX\_INTERRUPT) << MASK\_RX\_DR) | // IRQ interrupt on RX (0 = enabled)

(!(TX\_INTERRUPT) << MASK\_TX\_DS) | // IRQ interrupt on TX (0 = enabled)

(!(RT\_INTERRUPT) << MASK\_MAX\_RT) | // IRQ interrupt on auto retransmit counter overflow (0 = enabled)

(1 << EN\_CRC) | // CRC enable

(1 << CRC0) | // CRC scheme

(1 << PWR\_UP) | // Power up

(1 << PRIM\_RX); // TX/RX select

nrf24\_write(CONFIG,&data,1);

// Auto-acknowledge on all pipes

data =

(AUTO\_ACK << ENAA\_P5) |

(AUTO\_ACK << ENAA\_P4) |

(AUTO\_ACK << ENAA\_P3) |

(AUTO\_ACK << ENAA\_P2) |

(AUTO\_ACK << ENAA\_P1) |

(AUTO\_ACK << ENAA\_P0);

nrf24\_write(EN\_AA,&data,1);

// Set retries

data = 0xF0; // Delay 4000us with 1 re-try (will be added in settings)

nrf24\_write(SETUP\_RETR,&data,1);

// Disable RX addresses

data = 0;

nrf24\_write(EN\_RXADDR, &data, 1);

// Set channel

data = CHANNEL;

nrf24\_write(RF\_CH,&data,1);

// Setup

data =

(CONTINUOUS << CONT\_WAVE) | // Continuous carrier transmit

((DATARATE >> RF\_DR\_HIGH) << RF\_DR\_HIGH) | // Data rate

((POWER >> RF\_PWR) << RF\_PWR); // PA level

nrf24\_write(RF\_SETUP,&data,1);

// Status - clear TX/RX FIFO's and MAX\_RT by writing 1 into them

data =

(1 << RX\_DR) | // RX FIFO

(1 << TX\_DS) | // TX FIFO

(1 << MAX\_RT); // MAX RT

nrf24\_write(STATUS,&data,1);

// Dynamic payload on all pipes

data =

(DYN\_PAYLOAD << DPL\_P0) |

(DYN\_PAYLOAD << DPL\_P1) |

(DYN\_PAYLOAD << DPL\_P2) |

(DYN\_PAYLOAD << DPL\_P3) |

(DYN\_PAYLOAD << DPL\_P4) |

(DYN\_PAYLOAD << DPL\_P5);

nrf24\_write(DYNPD, &data,1);

// Enable dynamic payload

data =

(DYN\_PAYLOAD << EN\_DPL) |

(AUTO\_ACK << EN\_ACK\_PAY) |

(AUTO\_ACK << EN\_DYN\_ACK);

nrf24\_write(FEATURE,&data,1);

// Flush TX/RX

// Clear RX FIFO which will reset interrupt

*uint8\_t* data = (1 << RX\_DR) | (1 << TX\_DS) | (1 << MAX\_RT);

nrf24\_write(FLUSH\_RX,0,0);

nrf24\_write(FLUSH\_TX,0,0);

// Open pipes

nrf24\_write(RX\_ADDR\_P0 + READ\_PIPE,rx\_address,5);

nrf24\_write(TX\_ADDR,tx\_address,5);

nrf24\_write(EN\_RXADDR,&data,1);

data |= (1 << READ\_PIPE);

nrf24\_write(EN\_RXADDR,&data,1);

}

void nrf24\_write\_ack(void)

{

const void \*ack = "A";

unsigned int length = 1;

csn\_low;

spi\_send(W\_ACK\_PAYLOAD);

while (length--) spi\_send(\*(*uint8\_t* \*)ack++);

csn\_high;

}

void nrf24\_state(*uint8\_t* state)

{

*uint8\_t* config\_register;

nrf24\_read(CONFIG,&config\_register,1);

switch (state)

{

case POWERUP:

// Check if already powered up

if (!(config\_register & (1 << PWR\_UP)))

{

data = config\_register | (1 << PWR\_UP);

nrf24\_write(CONFIG,&data,1);

// 1.5ms from POWERDOWN to start up

*\_delay\_ms*(2);

}

break;

case POWERDOWN:

data = config\_register & ~(1 << PWR\_UP);

nrf24\_write(CONFIG,&data,1);

break;

case RECEIVE:

data = config\_register | (1 << PRIM\_RX);

nrf24\_write(CONFIG,&data,1);

// Clear STATUS register

data = (1 << RX\_DR) | (1 << TX\_DS) | (1 << MAX\_RT);

nrf24\_write(STATUS,&data,1);

break;

case TRANSMIT:

data = config\_register & ~(1 << PRIM\_RX);

nrf24\_write(CONFIG,&data,1);

break;

case STANDBY1:

ce\_low;

break;

case STANDBY2:

data = config\_register & ~(1 << PRIM\_RX);

nrf24\_write(CONFIG,&data,1);

ce\_high;

*\_delay\_us*(150);

break;

}

}

void nrf24\_start\_listening(void)

{

nrf24\_state(RECEIVE); // Receive mode

//if (AUTO\_ACK) nrf24\_write\_ack(); // Write acknowledgment

ce\_high;

*\_delay\_us*(150); // Settling time

}

*uint8\_t* nrf24\_send\_message(const void \*tx\_message)

{

// For printf();

char temp[32];

*memset*(temp,0,32);

*strcpy*(temp,tx\_message);

// Message length

*uint8\_t* length = *strlen*(tx\_message);

// Transmit mode

nrf24\_state(TRANSMIT);

// Flush TX/RX and clear TX interrupt

nrf24\_write(FLUSH\_RX,0,0);

nrf24\_write(FLUSH\_TX,0,0);

data = (1 << TX\_DS);

nrf24\_write(STATUS,&data,1);

// Disable interrupt on RX

nrf24\_read(CONFIG,&data,1);

data |= (1 << MASK\_RX\_DR);

nrf24\_write(CONFIG,&data,1);

// Start SPI, load message into TX\_PAYLOAD

csn\_low;

if (AUTO\_ACK) spi\_send(W\_TX\_PAYLOAD);

else spi\_send(W\_TX\_PAYLOAD\_NOACK);

while (length--) spi\_send(\*(*uint8\_t* \*)tx\_message++);

spi\_send(0);

csn\_high;

// Send message by pulling CE high for more than 10us

ce\_high;

*\_delay\_us*(15);

ce\_low;

// Wait for message to be sent (TX\_DS flag raised)

nrf24\_read(STATUS,&data,1);

while(!(data & (1 << TX\_DS))) nrf24\_read(STATUS,&data,1);

*printf*("Message sent: %s\n",temp);

// Enable interrupt on RX

nrf24\_read(CONFIG,&data,1);

data &= ~(1 << MASK\_RX\_DR);

nrf24\_write(CONFIG,&data,1);

// Continue listening

nrf24\_start\_listening();

return 1;

}

unsigned int nrf24\_available(void)

{

*uint8\_t* config\_register;

nrf24\_read(FIFO\_STATUS,&config\_register,1);

if (!(config\_register & (1 << RX\_EMPTY))) return 1;

return 0;

}

const char \* nrf24\_read\_message(void)

{

// Message placeholder

static char rx\_message[32];

*memset*(rx\_message,0,32);

// Write ACK message

if (AUTO\_ACK) nrf24\_write\_ack();

// Get length of incoming message

nrf24\_read(R\_RX\_PL\_WID,&data,1);

// Read message

if (data > 0) nrf24\_send\_spi(R\_RX\_PAYLOAD,&rx\_message,data+1);

// Check if there is message in array

if (*strlen*(rx\_message) > 0)

{

// Clear RX interrupt

data = (1 << RX\_DR);

nrf24\_write(STATUS,&data,1);

return rx\_message;

}

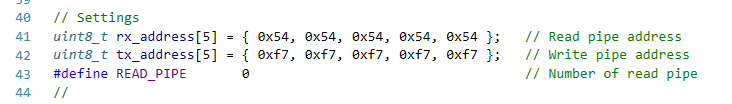
// Clear RX interrupt

data = (1 << RX\_DR);

nrf24\_write(STATUS,&data,1);

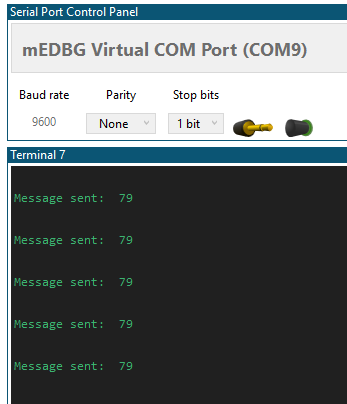
return "failed";

}



This is a screenshot of my read and write addresses. My partner used the flipped addresses in his code.

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**
2. **SCHEMATICS**
3. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**



1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**
2. **VIDEO LINKS OF EACH DEMO**
3. **GITHUB LINK OF THIS DA**

<https://github.com/johnsb18/ClassRepository/tree/master/DesignAssignments/DA5>

**Student Academic Misconduct Policy**

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

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

Benjamin Johnson