CPE301 – SPRING 2019

Design Assignment 5A

Student Name: Saul Alejandro Mendoza Guzman

Student #: 2000540481

Student Email: mendos1@unlv.nevada.edu

Primary Github address: <https://github.com/mendos1/subnission_da>

Directory: DA5A

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 x2
   * Nrf24l01 + x2
   * Wires
   * Lm3x x2
   * Atmel Studio 7
2. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ MAIN.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// Set clock frequency

#ifndef *F\_CPU*

#define *F\_CPU* 16000000UL

#endif

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

unsigned int ADC\_TEMP;

// Set up UART for printf();

#ifndef BAUD

#define BAUD 9600

#endif

#include "STDIO\_UART.h"

// Include nRF24L01+ library

#include "nrf24l01.h"

#include "nrf24l01-mnemonics.h"

#include "spi.h"

void print\_config(void);

void ADC\_INIT(void);

void READ\_ADC(void);

// Used in IRQ ISR

volatile bool message\_received = false;

volatile bool status = false;

int main(void){

// Set cliche message to send (message cannot exceed 32 characters)

char tx\_message[32]; // Define string array

char \*tx\_ptr = tx\_message;

*strcpy*(tx\_message,"Hi :) !"); // Copy string into array

// Initialize UART

uart\_init();

// Initialize nRF24L01+ and print configuration info

nrf24\_init();

print\_config();

ADC\_INIT();

// Start listening to incoming messages

*printf*("start listening\n");

nrf24\_start\_listening();

*printf*("Done listening\n");

ADC\_TEMP = 0;

while (1){

//printf("outside if\n");

READ\_ADC();

tx\_ptr = tx\_message;

//sprintf(tx\_ptr, "%d", ADC\_TEMP);

nrf24\_send\_message(tx\_ptr);

delay\_ms(100);

//continue;

//;

if (message\_received){

*printf*("inside if condition\n");

// Message received, print it

message\_received = false;

*printf*("Received message: %s\n",nrf24\_read\_message());

// Send message as response

*\_delay\_ms*(500);

status = nrf24\_send\_message(tx\_message);

if (status == true) *printf*("Message sent successfully\n");

}

}

}

// Interrupt on IRQ pin

ISR(INT0\_vect) {

message\_received = true;

}

void ADC\_INIT(void){

ADMUX = (0<<REFS1)| // Reference Selection Bits

(1<<REFS0)| // AVcc - external cap at AREF

(0<<ADLAR)| // ADC Left Adjust Result

(1<<MUX2)| // ANalog Channel Selection Bits

(0<<MUX1)| //

(0<<MUX0);

ADCSRA = (1<<ADEN)| // ADC ENable

(0<<ADSC)| // ADC Start Conversion

(0<<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);

// Timer/Counter1 Interrupt Mask Register

TIMSK1 |= (1<<TOIE1); // enable overflow interrupt

TCCR1B |= (1<<CS12)|(1<<CS10); // clock

TCNT1 = 49911; //((16MHz/1024)\*1)-1 = 15624

}

void READ\_ADC(void) {

unsigned char i =4;

ADC\_TEMP = 0; //initialize

while (i--) {

ADCSRA |= (1<<ADSC);

while(ADCSRA & (1<<ADSC));

ADC\_TEMP+= ADC;

*\_delay\_ms*(50);

}

ADC\_TEMP = ADC\_TEMP/8 ; // Average

}

void print\_config(void){

*uint8\_t* data;

*printf*("Startup successful\n\n nRF24L01+ configured as:\n");

*printf*("-------------------------------------------\n");

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

*printf*("CONFIG 0x%02X\n",data);

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

*printf*("EN\_AA 0x%02X\n",data);

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

*printf*("EN\_RXADDR 0x%02X\n",data);

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

*printf*("SETUP\_RETR 0x%02X\n",data);

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

*printf*("RF\_CH 0x%02X\n",data);

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

*printf*("RF\_SETUP 0x%02X\n",data);

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

*printf*("STATUS 0x%02X\n",data);

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

*printf*("FEATURE 0x%02X\n",data);

*printf*("-------------------------------------------\n\n");

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**| THESE ARE THE LIBRARY FUNCTIONS |**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NRF24L01.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// MIT License

//

// Copyright (c) 2018 Helvijs Adams

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in all

// copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

// SOFTWARE.

// Set clock frequency

#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] = { 0x73, 0x73, 0x73, 0x73, 0x73 }; // Read pipe address

*uint8\_t* tx\_address[5] = { 0x42, 0x42, 0x42, 0x42, 0x42 }; // 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";

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NRF24L01.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// MIT License

//

// Copyright (c) 2018 Helvijs Adams

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in all

// copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

// SOFTWARE.

#ifndef \_NRF24L01\_H

#define \_NRF24L01\_H

// States

#define POWERUP 1

#define POWERDOWN 2

#define RECEIVE 3

#define TRANSMIT 4

#define STANDBY1 5

#define STANDBY2 6

// Forward declarations

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

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

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

void nrf24\_init(void);

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

void nrf24\_start\_listening(void);

unsigned int nrf24\_available(void);

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

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

#endif /\*\_NRF24L01\_H\*/

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NRF24l01-MNEMONICS.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

/\* Copyright (c) 2015 B. Sidhipong <bsidhipong@gmail.com>

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>. \*/

#ifndef \_\_NORDIC\_NRF24L01\_RADIO\_H\_\_

#define \_\_NORDIC\_NRF24L01\_RADIO\_H\_\_

#include <avr/common.h>

/\* SPI commands \*/

#define REGISTER\_MASK 0b00011111

#define R\_REGISTER 0b00000000 /\* 000A AAAA | AAAAA = 5-bit register map address \*/

#define W\_REGISTER 0b00100000 /\* 001A AAAA \*/

#define R\_RX\_PAYLOAD 0b01100001

#define W\_TX\_PAYLOAD 0b10100000

#define FLUSH\_TX 0b11100001

#define FLUSH\_RX 0b11100010

#define REUSE\_TX\_PL 0b11100011

#define ACTIVATE 0b01010000

#define R\_RX\_PL\_WID 0b01100000

#define ACK\_PAYLOAD\_MASK 0b00000111

#define W\_ACK\_PAYLOAD 0b10101000 /\* 1010 1PPP | PPP = pipe number \*/

#define W\_TX\_PAYLOAD\_NOACK 0b10110000

#define NOP 0b11111111

/\* CONFIG: configuration register \*/

#define CONFIG 0x00

#define MASK\_RX\_DR 6

#define MASK\_TX\_DS 5

#define MASK\_MAX\_RT 4

#define EN\_CRC 3

#define CRC0 2

#define PWR\_UP 1

#define PRIM\_RX 0

/\* EN\_AA: Enhanced ShockBurst™

\* Enable 'Auto Acknowledgment' function. Disable this functionality

\* to be compatible with nRF2401. \*/

#define EN\_AA 0x01

#define ENAA\_P5 5

#define ENAA\_P4 4

#define ENAA\_P3 3

#define ENAA\_P2 2

#define ENAA\_P1 1

#define ENAA\_P0 0

/\* EN\_RXADDR: enable RX addresses \*/

#define EN\_RXADDR 0x02

#define ERX\_P5 5

#define ERX\_P4 4

#define ERX\_P3 3

#define ERX\_P2 2

#define ERX\_P1 1

#define ERX\_P0 0

/\* SETUP\_AW: seup of address widths \*/

#define SETUP\_AW 0x03

#define AW 0 /\* 1:0 \*/

/\* SETUP\_RETR: setup of automatic retransmission \*/

#define SETUP\_RETR 0x04

#define ARD 4 /\* 7:4 \*/

#define ARC 0 /\* 3:0 \*/

#define RF\_CH 0x05

/\* RF\_SETUP: RF setup register \*/

#define RF\_SETUP 0x06

#define CONT\_WAVE 7

#define RF\_DR\_LOW 5

#define PLL\_LOCK 4

#define RF\_DR\_HIGH 3 /\* RF\_DR:RF\_DR\_LOW = RF data rate \*/

#define RF\_DR\_250KBPS 0b00100000 /\* available on nRF24L01+ \*/

#define RF\_DR\_1MBPS 0b00000000

#define RF\_DR\_2MBPS 0b00001000

#define RF\_PWR 1 /\* 2:1 \*/

#define POWER\_MIN 0b00000000

#define POWER\_LOW 0b00000010

#define POWER\_HIGH 0b00000100

#define POWER\_MAX 0b00000110

/\* STATUS: status register \*/

#define STATUS 0x07

#define RX\_DR 6

#define TX\_DS 5

#define MAX\_RT 4

#define RX\_P\_NO 1 /\* 3:1 \*/

#define TX\_FULL 0

/\* OBSERVE\_TX: transmit observe register \*/

#define OBSERVE\_TX 0x08

#define PLOS\_CNT 4 /\* 7:4 \*/

#define ARC\_CNT 0 /\* 3:0 \*/

#define RPD 0x09

#define RX\_ADDR\_P0 0x0A

#define RX\_ADDR\_P1 0x0B

#define RX\_ADDR\_P2 0x0C

#define RX\_ADDR\_P3 0x0D

#define RX\_ADDR\_P4 0x0E

#define RX\_ADDR\_P5 0x0F

#define TX\_ADDR 0x10

#define RX\_PW\_P0 0x11

#define RX\_PW\_P1 0x12

#define RX\_PW\_P2 0x13

#define RX\_PW\_P3 0x14

#define RX\_PW\_P4 0x15

#define RX\_PW\_P5 0x16

/\* FIFO\_STATUS: FIFO status register \*/

#define FIFO\_STATUS 0x17

#define TX\_REUSE 6

#define FIFO\_FULL 5

#define TX\_EMPTY 4

#define RX\_FULL 1

#define RX\_EMPTY 0

/\* DYNPD: enable dynamic payload length \*/

#define DYNPD 0x1C

#define DPL\_P5 5

#define DPL\_P4 4

#define DPL\_P3 3

#define DPL\_P2 2

#define DPL\_P1 1

#define DPL\_P0 0

/\* FEATURE: \*/

#define FEATURE 0x1D

#define EN\_DPL 2

#define EN\_ACK\_PAY 1

#define EN\_DYN\_ACK 0

#endif /\* \_\_NORDIC\_NRF24L01\_RADIO\_H\_\_ \*/

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ SPI.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

/\* Copyright (c) 2015 B. Sidhipong <bsidhipong@gmail.com>

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>. \*/

#include <avr/common.h>

#include <avr/io.h>

#include <avr/interrupt.h>

#include "spi.h"

#define DDR\_SPI DDRB

#define DD\_MOSI DDB3

#define DD\_MISO DDB4

#define DD\_SCK DDB5

void spi\_master\_init( void )

{

DDR\_SPI &= ~\_BV(DD\_MISO);

DDR\_SPI |= (\_BV(DD\_MOSI) | \_BV(DD\_SCK));

/\* 19.5.1 SPCR – SPI Control Register

\*

\* • Bit 7 – SPIE: SPI Interrupt Enable

\* This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the Global

\* Interrupt Enable bit in SREG is set.

\*

\* • Bit 6 – SPE: SPI Enable

\* When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

\*

\* • Bit 5 – DORD: Data Order

\* When the DORD bit is written to one, the LSB of the data word is transmitted first.

\* When the DORD bit is written to zero, the MSB of the data word is transmitted first.

\*

\* • Bit 4 – MSTR: Master/Slave Select

\* This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If SS is

\* configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become

\* set. The user will then have to set MSTR to re-enable SPI Master mode.

\*

\* • Bit 3 – CPOL: Clock Polarity

\* When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle.

\*

\* • Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0

\* These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on

\* the Slave. The relationship between SCK and the Oscillator Clock frequency fosc is shown in the following table:

\*

\* Table 19-5 Relationship Between SCK and Oscillator Frequency

\* SPI2X | SPR1 | SPR0 | SCK Frequency

\* ===================================

\* 0 | 0 | 0 | fosc/4

\* 0 | 0 | 1 | fosc/16

\* 0 | 1 | 0 | fosc/64

\* 0 | 1 | 1 | fosc/128

\* 1 | 0 | 0 | fosc/2

\* 1 | 0 | 1 | fosc/8

\* 1 | 1 | 0 | fosc/32

\* 1 | 1 | 1 | fosc/64

\*/

SPCR = (0 << SPIE) |

(1 << SPE) |

(0 << DORD) |

(1 << MSTR) |

(0 << CPOL) |

(0 << SPR1) | (0 << SPR0);

/\* 19.5.2 SPSR – SPI Status Register

\*

\* • Bit 7 – SPIF: SPI Interrupt Flag

\* When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and

\* global interrupts are enabled. If SS is an input and is driven low when the SPI is in Master mode, this will also

\* set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector.

\* Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the

\* SPI Data Register (SPDR).

\*

\* • Bit 0 – SPI2X: Double SPI Speed Bit

\* When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master

\* mode (see Table 19-5). This means that the minimum SCK period will be two CPU clock periods. When the SPI

\* is configured as Slave, the SPI is only guaranteed to work at fosc/4 or lower. \*/

SPSR |= \_BV(SPI2X);

}

void spi\_bulk\_send( *uint8\_t* \*send\_buffer, *uint8\_t* count )

{

while ( count-- ) {

SPDR = \*send\_buffer++;

loop\_until\_bit\_is\_set(SPSR, SPIF);

}

}

void spi\_send( *uint8\_t* send\_data )

{

SPDR = send\_data;

loop\_until\_bit\_is\_set(SPSR, SPIF);

}

void spi\_bulk\_exchange( *uint8\_t* \*send\_buffer, *uint8\_t* \*receive\_buffer, *uint8\_t* count )

{

while ( count-- ) {

SPDR = \*send\_buffer++;

loop\_until\_bit\_is\_set(SPSR, SPIF);

\*receive\_buffer++ = SPDR;

}

}

*uint8\_t* spi\_exchange( *uint8\_t* send\_data )

{

SPDR = send\_data;

loop\_until\_bit\_is\_set(SPSR, SPIF);

return SPDR;

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ SPI.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

/\* Copyright (c) 2015 B. Sidhipong <bsidhipong@gmail.com>

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>. \*/

#ifndef \_\_SMALL\_SPI\_H\_\_

#define \_\_SMALL\_SPI\_H\_\_

#include <avr/common.h>

void spi\_master\_init( void );

void spi\_bulk\_send( *uint8\_t* \*send\_buffer, *uint8\_t* count );

void spi\_send( *uint8\_t* send\_data );

void spi\_bulk\_exchange( *uint8\_t* \*send\_buffer, *uint8\_t* \*receive\_buffer, *uint8\_t* count );

*uint8\_t* spi\_exchange( *uint8\_t* send\_data );

#endif /\* \_\_SPI\_H\_\_ \*/

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ STDIO\_UART.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// https://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

#ifndef *F\_CPU*

#define *F\_CPU* 16000000UL

#endif

#include <stdio.h>

#include <avr/io.h>

#include "STDIO\_UART.h"

#ifndef BAUD

#define BAUD 9600

#endif

#define MYUBRR (((*F\_CPU* / (BAUD \* 16UL))) - 1)

static *FILE* mystdout = *FDEV\_SETUP\_STREAM*(uart\_putchar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

static *FILE* mystdin = *FDEV\_SETUP\_STREAM*(*NULL*, uart\_getchar, *\_FDEV\_SETUP\_READ*);

void uart\_init(void)

{

UBRR0H = MYUBRR >> 8;

UBRR0L = MYUBRR;

UCSR0B = (1<<RXEN0)|(1<<TXEN0);

*stdout* = &mystdout;

*stdin* = &mystdin;

}

// Redirect stdout to UART

int uart\_putchar(char c, *FILE* \*stream) {

if (c == '\n') {

uart\_putchar('\r', stream);

}

loop\_until\_bit\_is\_set(UCSR0A, UDRE0);

UDR0 = c;

return 0;

}

// Redirect stdin to UART

int uart\_getchar(*FILE* \*stream) {

loop\_until\_bit\_is\_set(UCSR0A, RXC0);

return UDR0;

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ STDIO\_UART.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// https://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

#ifndef *F\_CPU*

#define *F\_CPU* 16000000UL

#endif

#include <stdio.h>

#include <avr/io.h>

#include "STDIO\_UART.h"

#ifndef BAUD

#define BAUD 9600

#endif

#define MYUBRR (((*F\_CPU* / (BAUD \* 16UL))) - 1)

static *FILE* mystdout = *FDEV\_SETUP\_STREAM*(uart\_putchar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

static *FILE* mystdin = *FDEV\_SETUP\_STREAM*(*NULL*, uart\_getchar, *\_FDEV\_SETUP\_READ*);

void uart\_init(void)

{

UBRR0H = MYUBRR >> 8;

UBRR0L = MYUBRR;

UCSR0B = (1<<RXEN0)|(1<<TXEN0);

*stdout* = &mystdout;

*stdin* = &mystdin;

}

// Redirect stdout to UART

int uart\_putchar(char c, *FILE* \*stream) {

if (c == '\n') {

uart\_putchar('\r', stream);

}

loop\_until\_bit\_is\_set(UCSR0A, UDRE0);

UDR0 = c;

return 0;

}

// Redirect stdin to UART

int uart\_getchar(*FILE* \*stream) {

loop\_until\_bit\_is\_set(UCSR0A, RXC0);

return UDR0;

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ STDIO\_UART.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// https://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

#ifndef STDIO\_UART\_H\_

#define STDIO\_UART\_H\_

void uart\_init(void);

int uart\_putchar(char c, *FILE* \*stream);

int uart\_getchar(*FILE* \*stream);

#endif /\* STDIO\_UART\_H\_ \*/

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**| THIS IS MY PARTNERS CODE |**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ MAIN.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// Set clock frequency

#ifndef *F\_CPU*

#define *F\_CPU* 16000000UL

#endif

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

unsigned int ADC\_TEMP;

// Set up UART for printf();

#ifndef BAUD

#define BAUD 9600

#endif

#include "STDIO\_UART.h"

// Include nRF24L01+ library

#include "nrf24l01.h"

#include "nrf24l01-mnemonics.h"

#include "spi.h"

void print\_config(void);

void ADC\_INIT(void);

void READ\_ADC(void);

// Used in IRQ ISR

volatile bool message\_received = false;

volatile bool status = false;

int main(void){

// Set cliche message to send (message cannot exceed 32 characters)

char tx\_message[32]; // Define string array

char \*tx\_ptr = tx\_message;

*strcpy*(tx\_message,"Hi :) !"); // Copy string into array

// Initialize UART

uart\_init();

// Initialize nRF24L01+ and print configuration info

nrf24\_init();

print\_config();

ADC\_INIT();

// Start listening to incoming messages

*printf*("start listening\n");

nrf24\_start\_listening();

*printf*("Done listening\n");

ADC\_TEMP = 0;

while (1){

//printf("outside if\n");

READ\_ADC();

tx\_ptr = tx\_message;

//sprintf(tx\_ptr, "%d", ADC\_TEMP);

nrf24\_send\_message(tx\_ptr);

delay\_ms(100);

//continue;

//;

if (message\_received){

*printf*("inside if condition\n");

// Message received, print it

message\_received = false;

*printf*("Received message: %s\n",nrf24\_read\_message());

// Send message as response

*\_delay\_ms*(500);

status = nrf24\_send\_message(tx\_message);

if (status == true) *printf*("Message sent successfully\n");

}

}

}

// Interrupt on IRQ pin

ISR(INT0\_vect) {

message\_received = true;

}

void ADC\_INIT(void){

ADMUX = (0<<REFS1)| // Reference Selection Bits

(1<<REFS0)| // AVcc - external cap at AREF

(0<<ADLAR)| // ADC Left Adjust Result

(1<<MUX2)| // ANalog Channel Selection Bits

(0<<MUX1)| //

(0<<MUX0);

ADCSRA = (1<<ADEN)| // ADC ENable

(0<<ADSC)| // ADC Start Conversion

(0<<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);

// Timer/Counter1 Interrupt Mask Register

TIMSK1 |= (1<<TOIE1); // enable overflow interrupt

TCCR1B |= (1<<CS12)|(1<<CS10); // clock

TCNT1 = 49911; //((16MHz/1024)\*1)-1 = 15624

}

void READ\_ADC(void) {

unsigned char i =4;

ADC\_TEMP = 0; //initialize

while (i--) {

ADCSRA |= (1<<ADSC);

while(ADCSRA & (1<<ADSC));

ADC\_TEMP+= ADC;

*\_delay\_ms*(50);

}

ADC\_TEMP = ADC\_TEMP/8 ; // Average

}

void print\_config(void){

*uint8\_t* data;

*printf*("Startup successful\n\n nRF24L01+ configured as:\n");

*printf*("-------------------------------------------\n");

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

*printf*("CONFIG 0x%02X\n",data);

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

*printf*("EN\_AA 0x%02X\n",data);

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

*printf*("EN\_RXADDR 0x%02X\n",data);

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

*printf*("SETUP\_RETR 0x%02X\n",data);

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

*printf*("RF\_CH 0x%02X\n",data);

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

*printf*("RF\_SETUP 0x%02X\n",data);

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

*printf*("STATUS 0x%02X\n",data);

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

*printf*("FEATURE 0x%02X\n",data);

*printf*("-------------------------------------------\n\n");

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**| THESE ARE THE LIBRARY FUNCTIONS |**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NRF24L01.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// MIT License

//

// Copyright (c) 2018 Helvijs Adams

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in all

// copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

// SOFTWARE.

// Set clock frequency

#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"

0x73, 0x73, 0x73, 0x73,

// Settings

*uint8\_t* rx\_address[5] = { 0x42, 0x42, 0x42, 0x42, 0x42 }; // Read pipe address

*uint8\_t* tx\_address[5] = { 0x73, 0x73, 0x73, 0x73, 0x73 }; // 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";

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NRF24L01.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// MIT License

//

// Copyright (c) 2018 Helvijs Adams

//

// Permission is hereby granted, free of charge, to any person obtaining a copy

// of this software and associated documentation files (the "Software"), to deal

// in the Software without restriction, including without limitation the rights

// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

// copies of the Software, and to permit persons to whom the Software is

// furnished to do so, subject to the following conditions:

//

// The above copyright notice and this permission notice shall be included in all

// copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

// SOFTWARE.

#ifndef \_NRF24L01\_H

#define \_NRF24L01\_H

// States

#define POWERUP 1

#define POWERDOWN 2

#define RECEIVE 3

#define TRANSMIT 4

#define STANDBY1 5

#define STANDBY2 6

// Forward declarations

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

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

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

void nrf24\_init(void);

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

void nrf24\_start\_listening(void);

unsigned int nrf24\_available(void);

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

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

#endif /\*\_NRF24L01\_H\*/

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NRF24l01-MNEMONICS.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

/\* Copyright (c) 2015 B. Sidhipong <bsidhipong@gmail.com>

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>. \*/

#ifndef \_\_NORDIC\_NRF24L01\_RADIO\_H\_\_

#define \_\_NORDIC\_NRF24L01\_RADIO\_H\_\_

#include <avr/common.h>

/\* SPI commands \*/

#define REGISTER\_MASK 0b00011111

#define R\_REGISTER 0b00000000 /\* 000A AAAA | AAAAA = 5-bit register map address \*/

#define W\_REGISTER 0b00100000 /\* 001A AAAA \*/

#define R\_RX\_PAYLOAD 0b01100001

#define W\_TX\_PAYLOAD 0b10100000

#define FLUSH\_TX 0b11100001

#define FLUSH\_RX 0b11100010

#define REUSE\_TX\_PL 0b11100011

#define ACTIVATE 0b01010000

#define R\_RX\_PL\_WID 0b01100000

#define ACK\_PAYLOAD\_MASK 0b00000111

#define W\_ACK\_PAYLOAD 0b10101000 /\* 1010 1PPP | PPP = pipe number \*/

#define W\_TX\_PAYLOAD\_NOACK 0b10110000

#define NOP 0b11111111

/\* CONFIG: configuration register \*/

#define CONFIG 0x00

#define MASK\_RX\_DR 6

#define MASK\_TX\_DS 5

#define MASK\_MAX\_RT 4

#define EN\_CRC 3

#define CRC0 2

#define PWR\_UP 1

#define PRIM\_RX 0

/\* EN\_AA: Enhanced ShockBurst™

\* Enable 'Auto Acknowledgment' function. Disable this functionality

\* to be compatible with nRF2401. \*/

#define EN\_AA 0x01

#define ENAA\_P5 5

#define ENAA\_P4 4

#define ENAA\_P3 3

#define ENAA\_P2 2

#define ENAA\_P1 1

#define ENAA\_P0 0

/\* EN\_RXADDR: enable RX addresses \*/

#define EN\_RXADDR 0x02

#define ERX\_P5 5

#define ERX\_P4 4

#define ERX\_P3 3

#define ERX\_P2 2

#define ERX\_P1 1

#define ERX\_P0 0

/\* SETUP\_AW: seup of address widths \*/

#define SETUP\_AW 0x03

#define AW 0 /\* 1:0 \*/

/\* SETUP\_RETR: setup of automatic retransmission \*/

#define SETUP\_RETR 0x04

#define ARD 4 /\* 7:4 \*/

#define ARC 0 /\* 3:0 \*/

#define RF\_CH 0x05

/\* RF\_SETUP: RF setup register \*/

#define RF\_SETUP 0x06

#define CONT\_WAVE 7

#define RF\_DR\_LOW 5

#define PLL\_LOCK 4

#define RF\_DR\_HIGH 3 /\* RF\_DR:RF\_DR\_LOW = RF data rate \*/

#define RF\_DR\_250KBPS 0b00100000 /\* available on nRF24L01+ \*/

#define RF\_DR\_1MBPS 0b00000000

#define RF\_DR\_2MBPS 0b00001000

#define RF\_PWR 1 /\* 2:1 \*/

#define POWER\_MIN 0b00000000

#define POWER\_LOW 0b00000010

#define POWER\_HIGH 0b00000100

#define POWER\_MAX 0b00000110

/\* STATUS: status register \*/

#define STATUS 0x07

#define RX\_DR 6

#define TX\_DS 5

#define MAX\_RT 4

#define RX\_P\_NO 1 /\* 3:1 \*/

#define TX\_FULL 0

/\* OBSERVE\_TX: transmit observe register \*/

#define OBSERVE\_TX 0x08

#define PLOS\_CNT 4 /\* 7:4 \*/

#define ARC\_CNT 0 /\* 3:0 \*/

#define RPD 0x09

#define RX\_ADDR\_P0 0x0A

#define RX\_ADDR\_P1 0x0B

#define RX\_ADDR\_P2 0x0C

#define RX\_ADDR\_P3 0x0D

#define RX\_ADDR\_P4 0x0E

#define RX\_ADDR\_P5 0x0F

#define TX\_ADDR 0x10

#define RX\_PW\_P0 0x11

#define RX\_PW\_P1 0x12

#define RX\_PW\_P2 0x13

#define RX\_PW\_P3 0x14

#define RX\_PW\_P4 0x15

#define RX\_PW\_P5 0x16

/\* FIFO\_STATUS: FIFO status register \*/

#define FIFO\_STATUS 0x17

#define TX\_REUSE 6

#define FIFO\_FULL 5

#define TX\_EMPTY 4

#define RX\_FULL 1

#define RX\_EMPTY 0

/\* DYNPD: enable dynamic payload length \*/

#define DYNPD 0x1C

#define DPL\_P5 5

#define DPL\_P4 4

#define DPL\_P3 3

#define DPL\_P2 2

#define DPL\_P1 1

#define DPL\_P0 0

/\* FEATURE: \*/

#define FEATURE 0x1D

#define EN\_DPL 2

#define EN\_ACK\_PAY 1

#define EN\_DYN\_ACK 0

#endif /\* \_\_NORDIC\_NRF24L01\_RADIO\_H\_\_ \*/

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ SPI.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

/\* Copyright (c) 2015 B. Sidhipong <bsidhipong@gmail.com>

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>. \*/

#include <avr/common.h>

#include <avr/io.h>

#include <avr/interrupt.h>

#include "spi.h"

#define DDR\_SPI DDRB

#define DD\_MOSI DDB3

#define DD\_MISO DDB4

#define DD\_SCK DDB5

void spi\_master\_init( void )

{

DDR\_SPI &= ~\_BV(DD\_MISO);

DDR\_SPI |= (\_BV(DD\_MOSI) | \_BV(DD\_SCK));

/\* 19.5.1 SPCR – SPI Control Register

\*

\* • Bit 7 – SPIE: SPI Interrupt Enable

\* This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the Global

\* Interrupt Enable bit in SREG is set.

\*

\* • Bit 6 – SPE: SPI Enable

\* When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

\*

\* • Bit 5 – DORD: Data Order

\* When the DORD bit is written to one, the LSB of the data word is transmitted first.

\* When the DORD bit is written to zero, the MSB of the data word is transmitted first.

\*

\* • Bit 4 – MSTR: Master/Slave Select

\* This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If SS is

\* configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become

\* set. The user will then have to set MSTR to re-enable SPI Master mode.

\*

\* • Bit 3 – CPOL: Clock Polarity

\* When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle.

\*

\* • Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0

\* These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on

\* the Slave. The relationship between SCK and the Oscillator Clock frequency fosc is shown in the following table:

\*

\* Table 19-5 Relationship Between SCK and Oscillator Frequency

\* SPI2X | SPR1 | SPR0 | SCK Frequency

\* ===================================

\* 0 | 0 | 0 | fosc/4

\* 0 | 0 | 1 | fosc/16

\* 0 | 1 | 0 | fosc/64

\* 0 | 1 | 1 | fosc/128

\* 1 | 0 | 0 | fosc/2

\* 1 | 0 | 1 | fosc/8

\* 1 | 1 | 0 | fosc/32

\* 1 | 1 | 1 | fosc/64

\*/

SPCR = (0 << SPIE) |

(1 << SPE) |

(0 << DORD) |

(1 << MSTR) |

(0 << CPOL) |

(0 << SPR1) | (0 << SPR0);

/\* 19.5.2 SPSR – SPI Status Register

\*

\* • Bit 7 – SPIF: SPI Interrupt Flag

\* When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and

\* global interrupts are enabled. If SS is an input and is driven low when the SPI is in Master mode, this will also

\* set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector.

\* Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the

\* SPI Data Register (SPDR).

\*

\* • Bit 0 – SPI2X: Double SPI Speed Bit

\* When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master

\* mode (see Table 19-5). This means that the minimum SCK period will be two CPU clock periods. When the SPI

\* is configured as Slave, the SPI is only guaranteed to work at fosc/4 or lower. \*/

SPSR |= \_BV(SPI2X);

}

void spi\_bulk\_send( *uint8\_t* \*send\_buffer, *uint8\_t* count )

{

while ( count-- ) {

SPDR = \*send\_buffer++;

loop\_until\_bit\_is\_set(SPSR, SPIF);

}

}

void spi\_send( *uint8\_t* send\_data )

{

SPDR = send\_data;

loop\_until\_bit\_is\_set(SPSR, SPIF);

}

void spi\_bulk\_exchange( *uint8\_t* \*send\_buffer, *uint8\_t* \*receive\_buffer, *uint8\_t* count )

{

while ( count-- ) {

SPDR = \*send\_buffer++;

loop\_until\_bit\_is\_set(SPSR, SPIF);

\*receive\_buffer++ = SPDR;

}

}

*uint8\_t* spi\_exchange( *uint8\_t* send\_data )

{

SPDR = send\_data;

loop\_until\_bit\_is\_set(SPSR, SPIF);

return SPDR;

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ SPI.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

/\* Copyright (c) 2015 B. Sidhipong <bsidhipong@gmail.com>

\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>. \*/

#ifndef \_\_SMALL\_SPI\_H\_\_

#define \_\_SMALL\_SPI\_H\_\_

#include <avr/common.h>

void spi\_master\_init( void );

void spi\_bulk\_send( *uint8\_t* \*send\_buffer, *uint8\_t* count );

void spi\_send( *uint8\_t* send\_data );

void spi\_bulk\_exchange( *uint8\_t* \*send\_buffer, *uint8\_t* \*receive\_buffer, *uint8\_t* count );

*uint8\_t* spi\_exchange( *uint8\_t* send\_data );

#endif /\* \_\_SPI\_H\_\_ \*/

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ STDIO\_UART.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// https://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

#ifndef *F\_CPU*

#define *F\_CPU* 16000000UL

#endif

#include <stdio.h>

#include <avr/io.h>

#include "STDIO\_UART.h"

#ifndef BAUD

#define BAUD 9600

#endif

#define MYUBRR (((*F\_CPU* / (BAUD \* 16UL))) - 1)

static *FILE* mystdout = *FDEV\_SETUP\_STREAM*(uart\_putchar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

static *FILE* mystdin = *FDEV\_SETUP\_STREAM*(*NULL*, uart\_getchar, *\_FDEV\_SETUP\_READ*);

void uart\_init(void)

{

UBRR0H = MYUBRR >> 8;

UBRR0L = MYUBRR;

UCSR0B = (1<<RXEN0)|(1<<TXEN0);

*stdout* = &mystdout;

*stdin* = &mystdin;

}

// Redirect stdout to UART

int uart\_putchar(char c, *FILE* \*stream) {

if (c == '\n') {

uart\_putchar('\r', stream);

}

loop\_until\_bit\_is\_set(UCSR0A, UDRE0);

UDR0 = c;

return 0;

}

// Redirect stdin to UART

int uart\_getchar(*FILE* \*stream) {

loop\_until\_bit\_is\_set(UCSR0A, RXC0);

return UDR0;

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ STDIO\_UART.C ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// https://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

#ifndef *F\_CPU*

#define *F\_CPU* 16000000UL

#endif

#include <stdio.h>

#include <avr/io.h>

#include "STDIO\_UART.h"

#ifndef BAUD

#define BAUD 9600

#endif

#define MYUBRR (((*F\_CPU* / (BAUD \* 16UL))) - 1)

static *FILE* mystdout = *FDEV\_SETUP\_STREAM*(uart\_putchar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

static *FILE* mystdin = *FDEV\_SETUP\_STREAM*(*NULL*, uart\_getchar, *\_FDEV\_SETUP\_READ*);

void uart\_init(void)

{

UBRR0H = MYUBRR >> 8;

UBRR0L = MYUBRR;

UCSR0B = (1<<RXEN0)|(1<<TXEN0);

*stdout* = &mystdout;

*stdin* = &mystdin;

}

// Redirect stdout to UART

int uart\_putchar(char c, *FILE* \*stream) {

if (c == '\n') {

uart\_putchar('\r', stream);

}

loop\_until\_bit\_is\_set(UCSR0A, UDRE0);

UDR0 = c;

return 0;

}

// Redirect stdin to UART

int uart\_getchar(*FILE* \*stream) {

loop\_until\_bit\_is\_set(UCSR0A, RXC0);

return UDR0;

}

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ STDIO\_UART.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

// https://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

#ifndef STDIO\_UART\_H\_

#define STDIO\_UART\_H\_

void uart\_init(void);

int uart\_putchar(char c, *FILE* \*stream);

int uart\_getchar(*FILE* \*stream);

#endif /\* STDIO\_UART\_H\_ \*/

1. **SCHEMATICS**

NRF24L01+

328P

Mini

POWERSUPPLY

NRF24L01+

328P

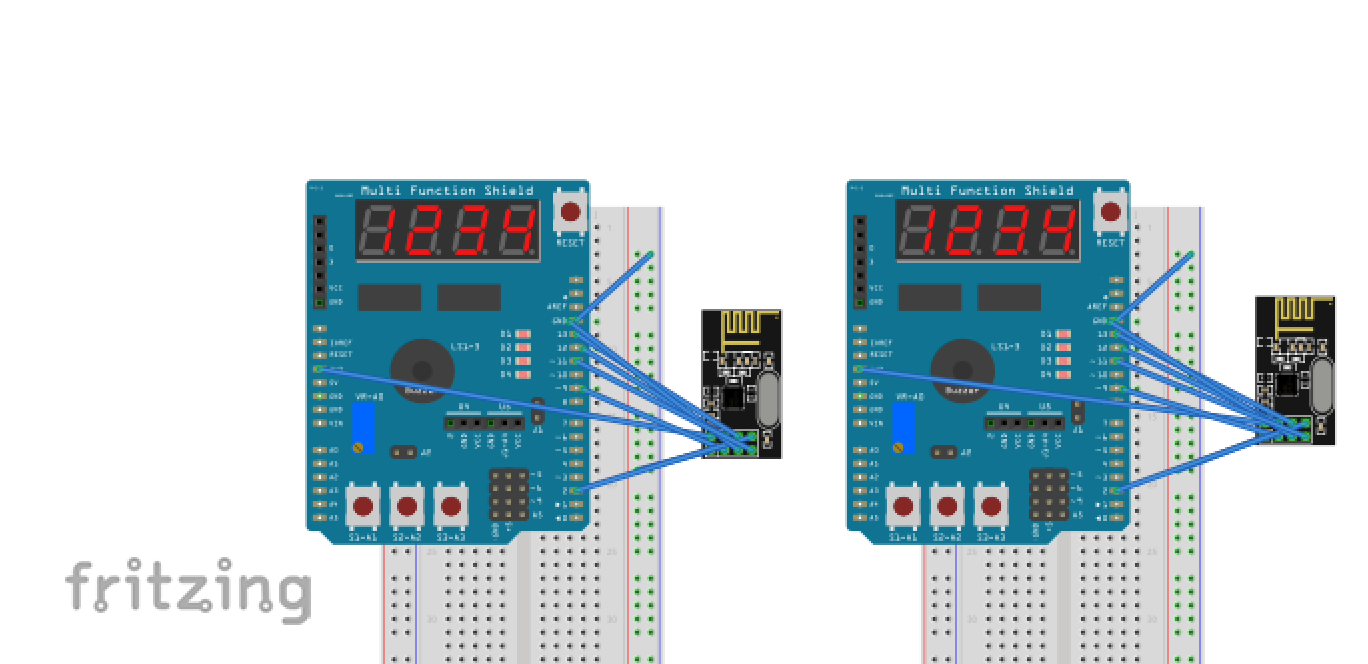
Mini

POWERSUPPLY

1. **PARTNERS NAME**

My partner for this assignment was Ricky Perez.

1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**



1. **VIDEO LINKS OF EACH DEMO**

<https://www.youtube.com/watch?v=BPUSEU625Hc>

1. **GITHUB LINK OF THIS DA**

<https://github.com/mendos1/subnission_da>

**Student Academic Misconduct Policy**

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

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

NAME OF THE STUDENT