Power supply (GND & VCC)
The module has to be powered with 3,3 V and **cannot** be powered by a 5 V power supply! Since it takes very little current I use a linear regulator to drop the voltage down to 3,3 V.
To make things a little easier for us, the chip can handle 5 V on the i/O ports, which is nice since it would be a pain to regulate down all the i/O cables from the AVR chip.

Chip Enable (CE)
Is used when to either send the data (transmitter) or start receive data (receiver).
The CE-pin is connected to any unused i/O port on the AVR and is set as output (set bit to one in the DDx register where x is the port letter.)
Atmega88: PB1, ATtiny26: PA0, ATtiny85: PB3

SPI Chip Select (CSN)
Also known as "Ship select not". The CSN-pin is also connected to any unused i/O port on the AVR and set to output. The CSN pin is held high at all the time except for when to send a SPI-command from the AVR to the nRF.
Atmega88: PB2, ATtiny26: PA1, ATtiny85: PB4

SPI Clock (SCK)
This is the serial clock. The SCK connects to the SCK-pin on the AVR.
Atmega88: PB5, ATtiny26: PB2, ATtiny85: PB2

SPI Master output Slave input (MOSI or MO)
This is the data line in the SPI system.
If your AVR chip supports SPI-transfere like the Atmega88, this connects to MOSI on the AVR as well and is set as output.

On AVR's that lacks SPI, like the ATtiny26 and ATtiny85 they come with USI instead, and the datasheet it says:

"The USI Three-wire mode is compliant to the Serial Peripheral Interface (SPI) mode 0 and 1, but
does not have the slave select (SS) pin functionality. However, this feature can be implemented
in software if necessary"

The "SS" refered to is the same as "CSN"
And after some research i found this blog that helped me allot.

To get the USI to SPI up and running I found out that I had to connect the **MOSI** pin from the nRF to the **MISO** pin on the AVR and set it as output.
Atmega88: PB3, ATtiny26: PB1, ATtiny85: PB1

SPI Master input Slave output (MISO or MI)
This is the data line in the SPI system.

If your AVR chip supports SPI-transfere like the Atmega88, this connects to MISO on the AVR and this one stays as an input.

To get it working on the ATtiny26 and ATtiny85, i had to use USI as mentioned above. This only worked when I connected the **MISO** pin on the nRF to the **MOSI** pin on the AVR and set it as input and enable internal pullup.
Atmega88: PB4, ATtiny26: PB0, ATtiny85: PB0

Interrupt Request (IRQ)
The IRQ pin is not necessary, but a great way of knowing when something has happened to the nRF. you can for example tell the nRF to set set the IRQ high when a package is received, or when a successful transmission is completed. Very useful!

If your AVR has more than 8 pins and an available interrupt-pin i would highly suggest you to connect the IRQ to that one and setup an interrupt request.
Atmega88: PD2, ATtiny26: PB6, ATtiny85: -

Part two- Programming

Here i will explain the c-program that runs on the AVR-chip. You can find a working copy of my code here (easier to copy and paste from).

Includes
I have included these lines to get my code to work:

```
#include <avr/io.h>
#include <stdio.h>
#define F_CPU 1000000UL  // 1 MHz change to match your AVR
#include <util/delay.h>
#include <avr/interrupt.h>

#include "nRF24L01.h"
```

You can see that i am importing a file called nRF24L01.h. This is a small library that defines the registers of the nRF so that i for example can call register "STATUS" instead of the register "0x07"... Just copy the text in the link and paste it into a file that you name "nRF24L01.h" and put it in the root of your folder.

Defines
To make the code cleaner i also put these definitions in the "nRF24L01.h"-file:

```
#define BIT(x) (1 << (x))
#define SETBITS(x,y) ((x) |= (y))
#define CLEARBITS(x,y) ((x) &= (~(y)))
#define SETBIT(x,y) SETBITS((x), (BIT((y))))
#define CLEARBIT(x,y) CLEARBITS((x), (BIT((y))))
```

And also add the defines:
#define W 1
#define R 0

SPI

Initialization
The nRF chip communicates with the AVR chip using SPI which has to be initialized in the AVR according to its datasheet. Here is the initializing code

```
void InitSPI(void)
{
    //Set SCK (PB5), MOSI (PB3) , CSN (SS & PB2) & CE  as outport
    //OBS!!! Has to be set before SPI-Enable below
    DDRB |= (1<<DDB5) | (1<<DDB3) | (1<<DDB2) |(1<<DDB1);

    // SPI Enable, Master, set clock rate fck/16. clock rate not to important..
    SPCR |= (1<<SPE)|(1<<MSTR);

    SETBIT(PORTB, 2);   //CSN IR_High to start with, nothing to be sent to the nRF yet!
    CLEARBIT(PORTB, 1); //CE low to start with, nothing to send/receive yet!
}
```

ATtiny26:

```
void InitSPI(void)
{
    //Set SCK (PB2), MISO (PB1 connects to nRF MOSI), CSN (PA1) & CE (PA0) as outport
    //OBS!!! Has to be set before SPI-Enable below
    DDRB |= (1<<PB2) | (1<<PB1);
    DDRA |= (1<<PA1) | (1<<PA0);

    ////Set MOSI (PB0) as input OBS: connects to nRF MISO
    DDRB &= ~(1<<PB0);
    PORTB |= (1<<PB0);

    USICR |= (1<<USIWM0)|(1<<USICS1)|(1<<USICLK);

    SETBIT(PORTA, 1);   //CSN high to start with, nothing to be sent to the nRF yet!
    CLEARBIT(PORTA, 0); //CE low to start with, nothing to send/receive yet!
}
```

ATtiny85:

```
void InitSPI(void)
{

    //Set SCK (PB2), MISO (PB1 connects to nRF MOSI) , CSN (PB4) and CE (PB3) as outport
    //OBS!!! Has to be set before SPI-Enable below
    DDRB |= (1<<PB2) | (1<<PB1) | (1<<PB4)| (1<<PB3);

    ////Set MOSI (PB0) as input OBS: connects to nRF MISO
    DDRB &= ~(1<<PB0);
    PORTB |= (1<<PB0);

    USICR |= (1<<USIWM0)|(1<<USICS1)|(1<<USICLK);

    SETBIT(PORTB, 4);   //CSN high to start with, nothing to be sent to the nRF yet!
    CLEARBIT(PORTB, 3); //CE low to start with, nothing to send/receive yet!
}
```

Communication
Now to send and receive a byte from the nRF with the SPI all you have to do is to use this function:
Atmega88 (SPI):

```
char WriteByteSPI(unsigned char cData)
{
    //Load byte to Data register
    SPDR = cData;

    /* Wait for transmission complete */
    while(!(SPSR & (1<<SPIF)));

    //Return what's receaved from the nrf
    return SPDR;
}
```

ATtiny(26 & 85) (USI as SPI):

```
uint8_t WriteByteSPI(uint8_t cData)
{
    //Load byte to Data register
    USIDR = cData;

    USISR |= (1<<USIOIF); // clear flag to be able to receive new data

    /* Wait for transmission complete */
    while ( (USISR & (1<<USIOIF)) == 0 )
    {
        USICR |=(1<<USITC);     //Toggle SCK and count a 4-bit counter from 0-15, when it reaches 15 USIOIF is set!
    }

    return USIDR;
}
```

I don't think it matters if you send a char or an integer, this is just how i got it to work... Note that these functions always returns something, this returned message is only cared fore when reading data from the nRF (a more appropriate name of the function might be "Write_Read_Byte_SPI").

nRF24L01(+) communication

Now to the fun part...

How it works
1) The nRF starts listening for commands when the CSN-pin goes low.
2) after a delay of 10us it accepts a single byte through SPI, which tells the nRF which bytes you want to read/write to, and if you want to read or write to it.
3) a 10us delay later it then accepts further bytes which is either written to the above specified register, or a number of dummy bytes (that tells the nRF how many bytes you want to read out)
4) when finished close the connection by setting CSN to high again.

Reading bytes from nRF
To start off, make sure your SPI communication is working by reading out something from the nRF. Reading a register on the nRF is accomplished by this function: (all of my example codes is for Atmega88, just change to the right port and pin number for the CSN and CE to get it to work on ATtiny as well)

```
uint8_t GetReg(uint8_t reg)
{
    _delay_us(10); //make sure last command was a while ago...
    CLEARBIT(PORTB, 2); //CSN low - nRF starts to listen for command
    _delay_us(10);
    WriteByteSPI(R_REGISTER + reg); //R_Register = set the nRF to reading mode, "reg" = this registry will be read back
    _delay_us(10);
    reg = WriteByteSPI(NOP);    //Send NOP (dummy byte) once to receive back the first byte in the "reg" register
    _delay_us(10);
    SETBIT(PORTB, 2);   //CSN Hi - nRF goes back to doing nothing
    return reg; // Return the read registry
}
```

I recommend you to start out by reading the STATUS register like this:

USART

If you have an AVR that supports USART like the Atmega88, i highly recommend you to use that as a way of sending the data back to the computer with this little friend... (I have written a small tutorial in the subject)

This is done simply by calling the function like this:

```
USART_Transmit(GetReg(STATUS));
```

If you like me have a function called "USART_Transmit(uint8_t data)"
The usart should send 0b00001110 (or 0x0E) to the computer since it is the preset configuration of the STATUS registry (see the end of this blogpost).

LED

If you are using an ATtiny it lacks the USART and thereby the ability to write things back to the computer, then you can use a more hardcore way using an LED to turn on if the STATUS register is set correctly. Since the bites in the STATUS (0x07) register are preset to 0b00001110 (or 0x0E) you can test if this is true by this function:

```
if (GetReg(STATUS)==0x0E)
{
    SETBIT(PORTB, 5); //LED on when true
}
```

Make sure you remember to first set the LED-pin to output: DDRB |=(1<<5);

If you are using an 8-pin ATtiny like the ATtiny85, there is not a single free pin on the chip to put the LED on, so i think a good idea would be to use the MISO-pin as a temporary LED output (since it is an output already and the SPI is not in use at the moment). Attach the LED to the PB1 and via a resistor to GND, then in the if-function above change the port number to 1. I haven't tested this my selves but i doubt it would cause any problem to the SPI-connection.

Writing bytes to the nRF

Now it's time to send a command to the nRF, this is done almost the exact same way as the reading command with this function:

```
void WriteToNrf(uint8_t reg, uint8_t Package)
{
    _delay_us(10); //make sure last command was a while ago...
    CLEARBIT(PORTB, 2); //CSN low - nRF starts to listen for command
    _delay_us(10);
    WriteByteSPI(W_REGISTER + reg); //W_Register = set the nRF to Write mode, "reg" = this registry will be written to
    _delay_us(10);
    WriteByteSPI(Package);  //Send the package to be written to the registry "reg"
    _delay_us(10);
    SETBIT(PORTB, 2);   //CSN Hi - nRF goes back to doing nothing
}
```

If the register holds more then one byte, the TX_ADDR-byte for example holds five bytes, then you have to send them one at a time after each other with 10us delay in between. This makes the function a bit more complicated since C-code is unwilling to pas arrays of integers into functions as is.

I also wanted to clean up my code a bit, so I decided to make one function that I can use to both read and write to the nRF. The function should also accept an array of integers and be able to return an array of integers. This is the result:

```
uint8_t *WriteToNrf(uint8_t ReadWrite, uint8_t reg, uint8_t *val, uint8_t antVal)
{
    //"ReadWrite" ("W" or "R"), "reg" (the register), "*val" (an array with the package) & "antVal" (number of integers in the package)

    if (ReadWrite == W) //if "W" then you want to write to the nRF (read mode "R" == 0x00, so skipping that one)
    {
        reg = W_REGISTER + reg; //Add the "write" bit to the "reg"
    }

    //Create an array to be returned at the end
    //Static uint8_t is needed to be able to return an array (notice the "*" to the left of the function: *WriteToNrf())
    static uint8_t ret[32];

    _delay_us(10);      //make sure last command was a while ago...
    CLEARBIT(PORTB, 2); //CSN low - nRF starts to listen for command
    _delay_us(10);
    WriteByteSPI(reg);  //set the nRF to Write or read mode of "reg"
    _delay_us(10);

    int i;
    for(i=0; i<antVal; i++)
    {
        if (ReadWrite == R && reg != W_TX_PAYLOAD)  //Did you want to read a registry?
        {                                           //When writing to W_TX_Payload you cannot add the "W" since it is on the same level in the registry...
            ret[i]=WriteByteSPI(NOP);   //Send dummy bytes to read out the data
            _delay_us(10);
        }
        else
        {
            WriteByteSPI(val[i]);   //Send the commands to the nRF once at a time
            _delay_us(10);
        }
    }
    SETBIT(PORTB, 2);   //CSN Hi - nRF goes back to doing nothing

    return ret; //return the array
}
```

The most confusing thing whit this function is the W_TX_PAYLOAD in the if-statement... The thing is that when you want to write bytes to the W_TX_PAYLOAD you cannot add the W_REGISTER as you normally does when you want to write to a register. Have a look at the registry setup at the very bottom of this blog post, and you will see that the W_REGISTER and the W_TX_PAYLOAD is in the same "top level" of the registers. The same goes for the TX_FLUSH registers...

Now here is some examples that shows how to use the function:

```
uint8_t val[5]; //An array av integers to send to the *WriteToNrf function

//Write one byte to the nRF:

//EN_RXADDR registry
val[0]=0x01;    //chose which of the data pipes to enable (0-5), now just nr 0.
WriteToNrf(W, EN_RXADDR, val, 1); //enable data pipe 0

//Write 5 bytes to the nRF:

//RX_ADDR_P0 registry. 1-5 bytes - the receiver address in channel P0
int i;
for(i=0; i<5; i++)
{
    val[i]=0x12;    //RF channel 0 address 0x12 x 5 - (make sure to put the same address on the transmitter!!)
}
WriteToNrf(W, RX_ADDR_P0, val, 5);


//Read an array of bytes from the nRF:
uint8_t *data;

data=WriteToNrf(R, RX_ADDR_P0, data, 5);    //store the Receiver address in the "data"-array
```

To come back to the W_TX_PAYLOAD, when you want to ad the payload to the nRF, you simply use the "R" instead of the "W" to trick the WeiteToNrf-function to not add the W_REGISTER.

Setting up nRF24L01(+)

Now it is time to setup the nRF for your specifications. In the example codes, I will send a 5 byte payload with the nRF. This is easily changed to a 1-32 byte payload by changing a bit in the initialization step of the nRF (see below)... This is how i usually set it up for simple communication between two nRF's:

```
void nrf24L01_init(void)
{
    _delay_ms(100); //allow radio to reach power down if shut down

    uint8_t val[5]; //An array av integers to send to the *WriteToNrf function

    //EN_AA - (enable auto-acknowledgments) - Transmitter gets automatic response from receiver when successful transmission! (lovely function!)
    //Only works if Transmitter has identical RF_Adress on its channel ex: RX_ADDR_P0 = TX_ADDR
    val[0]=0x01;    //set value
    WriteToNrf(W, EN_AA, val, 1);   //W=write mode, EN_AA=register to write to, val=data to write, 1=number of data bytes.

    //Choose number of enabled data pipes (1-5)
    val[0]=0x01;
    WriteToNrf(W, EN_RXADDR, val, 1); //enable data pipe 0

    //RF_Adress width setup (how many bytes is the receiver address, the more the merrier 1-5)
    val[0]=0x03;    //0b0000 00011 = 5 bytes RF_Adress
    WriteToNrf(W, SETUP_AW, val, 1);

    //RF channel setup - choose frequency 2,400-2,527GHz 1MHz/step
    val[0]=0x01;    //0b0000 0001 = 2,401GHz (same on TX and RX)
    WriteToNrf(W, RF_CH, val, 1);

    //RF setup  - choose power mode and data speed. Here is the diference with the (+) version!!!
    val[0]=0x07;    //00000111 bit 3="0" 1Mbps=longer range, bit 2-1 power mode ("11" = -0dB ; "00"=-18dB)
    WriteToNrf(W, RF_SETUP, val, 1);

    //RX RF_Adress setup 5 byte - Set Receiver address (set RX_ADDR_P0 = TX_ADDR if EN_AA  is enabled!!!)
    int i;
    for(i=0; i<5; i++)
    {
        val[i]=0x12;    //0x12 x 5 to get a long and secure address.
    }
    WriteToNrf(W, RX_ADDR_P0, val, 5); //since we chose pipe 0 on EN_RXADDR we give this address to that channel.
    //Here you can give different addresses to different channels (if they are enabled in EN_RXADDR) to listen on several different transmitters)

    //TX RF_Adress setup 5 byte -  Set Transmitter address (not used in a receiver but can be set anyway)
    for(i=0; i<5; i++)
    {
        val[i]=0x12;    //0x12 x 5 - same on the Receiver chip and the RX-RF_Address above if EN_AA is enabled!!!
    }
    WriteToNrf(W, TX_ADDR, val, 5);

    //Payload Width Setup - 1-32byte (how many bytes to send per transmission)
    val[0]=5;   //Send 5 bytes per package this time (same on receiver and transmitter)
    WriteToNrf(W, RX_PW_P0, val, 1);

    //CONFIG reg setup - Now it's time to boot up the nrf and choose if it's suppose to be a transmitter or receiver
    val[0]=0x1E;  //0b0001 1110 - bit 0="0":transmitter bit 0="1":Receiver, bit 1="1"=power up,
                  //bit 4="1"= mask_Max_RT i.e. IRQ-interrupt is not triggered if transmission failed.
    WriteToNrf(W, CONFIG, val, 1);

    //device need 1.5ms to reach standby mode (CE=low)
    _delay_ms(100);


}
```

In the code above, i missed a very important setting (if using EN_AA) that sets the number of retries and the retry delay like this:

```
val[0]=0x2F;    //0b0010 00011 "2" sets it up to 750uS delay between every retry (at least 500us at 250kbps and if payload >5bytes in 1Mbps,
                //and if payload >15byte in 2Mbps) "F" is number of retries (1-15, now 15)
WriteToNrf(W, SETUP_RETR, val, 1);
```

Add these lines in the function above to set the number of retries to 15 and the delay to 750us... the delay has to be greater than 500us if you are in the 250kbps mode, or if the payload is greater than 5bytes when in 1Mbps-mode or a payload greater then 15bytes when in 2Mbps mode. Note that the default value of this is only 250us, and will therefore cause trouble when in 250kbps mode and with bigger payloads!

As you can see i decided to make it a transmitter this time. This is easily changed in the code by first delay 50ms, to make sure the nRF is in sleep mode, then send 0x1F to the CONFIG register, than make it wait 50ms again before the first receive-command.

Transmit data

When in transmitting mode this is the function that sends your payload:

```
void transmit_payload(uint8_t * W_buff)
{
    WriteToNrf(R, FLUSH_TX, W_buff, 0); //Sends 0xE1 to flush the registry from old data! W_buff[] is only there because an array has to be called with an array..
    WriteToNrf(R, W_TX_PAYLOAD, W_buff, 5); //Sends the data in W_buff to the nrf
    //Why Flush_TX and W_TX_Payload is sent with an "R" instead of "W" is because they are on the highest byte-level in the nRF (see datasheet below)!

    //sei(); //Enable global interrupt (if interrupt is used)

    _delay_ms(10);      //needs a 10ms delay to work after loading the nrf with the payload for some reason
    SETBIT(PORTB, 1);   //CE high=transmit the data!
    _delay_us(20);      //delay at least 10us!
    CLEARBIT(PORTB, 1); //CE low = stop transmitting
    _delay_ms(10);      //long delay again before proceeding.
}
```

And you call the transmit function like this: (now i just send 0x93 five times in a row, you can fill the array with any bytes you want to send)

```
uint8_t W_buffer[5];
int i;
for (i=0;i<5;i++)
{
    W_buffer[i]=0x93;
}

transmit_payload(W_buffer);
```

Receive data

When in receiver mode this is the function that listens and receives your data:

```
void receive_payload(void)
{
    //sei();        //Enable global interrupt (if interrupt is used)

    SETBIT(PORTB, 1);   //CE IR_High = "listens" for data
    _delay_ms(1000);    //listens for 1s at a time
    CLEARBIT(PORTB, 1); //ce low again -stop listening

    //cli();    //Disable global interrupt
}
```

```
void reset(void)
{
    _delay_us(10);
    CLEARBIT(PORTB, 2); //CSN low
    _delay_us(10);
    WriteByteSPI(W_REGISTER + STATUS);  //write to STATUS registry
    _delay_us(10);
    WriteByteSPI(0x70); //Reset all irq in STATUS registry
    _delay_us(10);
    SETBIT(PORTB, 2);   //CSN IR_High
}
```

Verify transmitted/received data using

Interrupt

If you have more than an 8-pin AVR, I say use an interrupt to get triggered when data is successfully received or transmitted. INT0 interrupt is setup like this:

First you have to initialize the interrupt like this on Atmega88:

```
void INT0_interrupt_init(void)
{
    DDRD &= ~(1<<DDD2); //Extern interrupt on INT0, make sure it is input

    EICRA |=  (1<<ISC01);   // INT0 falling edge    PD2
    EICRA  &=  ~(1<<ISC00); // INT0 falling edge    PD2

    EIMSK |=  (1<<INT0);    //enable int0 interrupt
    //sei();              // Enable global interrupts do later
}
```

And initialization on ATtiny26:

```
void INT0_interrupt_init(void)
{
    DDRB &= ~(1<<DDB6); //External interrupt on INT0, make sure it is input

    MCUCR |=  (1<<ISC01);   // INT0 falling edge    PB6
    MCUCR  &=  ~(1<<ISC00); // INT0 falling edge    PB6

    GIMSK |=  (1<<INT0);    //enable int0 interrupt
    //sei();              // Enable global interrupts do later
}
```

Interrupt caused when receiving data

Setup a function triggered by the corresponding vector (INT0) at the very bottom of your code like this: (the global array "*data" is at the very top of my code...) And here i use it to send the received payload to the computer by usart:

```
uint8_t *data;

ISR(INT0_vect)  //vektorn that is run when transmit_payload succeed or when receive_payload received data
{
    //OBS: Mask_Max_rt in config registry has to be set to stop it from trigger on failed transmission!

    cli();  //Disable global interrupt
    CLEARBIT(PORTB, 1);     //ce low -stop sending/listening

    SETBIT(PORTB, 0); //led on to show success
    _delay_ms(500);
    CLEARBIT(PORTB, 0); //led off

    data=WriteToNrf(R, R_RX_PAYLOAD, data, 5);  //read out received message
    reset();    //reset the chip for further comunication

    for (int i=0;i<5;i++)
    {
        USART_Transmit(data[i]);    //send the received data to the computor with usart
    }

    sei();

}
```

Interrupt caused on transmission success

Use the same interrupt function when transmitting data but change its content to just flash the LED to tell you that transmission completed, or you can tell the nRF to switch for a receiver, if what you just sent was a question to a receiver that in turn changed to a transmitter to return your call.

When you use interrupt, it is crucial that you enables the external interrupts by the command sei(); This should be done before the receive_payload, and transmit_payload is used.

Verify transmitted received data without interrupt

If your chip lacks interrupt or if you ran out of free interrupt pins, you can manually check if the IRQ-flags are set in the status register after every time you either transmit data, or run the receive_payload function (before the reset function!!!)

Transmitting

The easiest when you want to see if transmission succeeded or failed, is to check if the MAX_RT is set which mean it failed. This is done like this:

```
if((GetReg(STATUS) & (1 << 4)) != 0 )   //if bit:4 MAX_RT is "1" then transmission failed!
```

Then you know you have to resend the package.... i usually put this statement in a while loop that loops untill the package is received!

Receiving

Do the same check to see if no data is received by checking the RX_DR-bit (data-ready) like this:

```
if (((GetReg(STATUS) & (1 << 6)) != 0 )) //Set high when new data arrives
```

Or it might be a better idea to check if data is received by changing "!=" to "==", if not, you start the receive_payload function again!

Code overview

```
#include <avr/io.h>
#include <stdio.h>
#define F_CPU 1000000UL  // 1 MHz
#include <util/delay.h>
#include <avr/interrupt.h>

#include "nRF24L01.h"

uint8_t *data;

/*****************SPI*****************************/
void InitSPI(void)...
char WriteByteSPI(unsigned char cData)...

/****************in/out LED*************************/
void ioinit(void)          ...

/****************interrupt*************************/
void INT0_interrupt_init(void)  ...

/****************Functions*************************/
uint8_t *WriteToNrf(uint8_t ReadWrite, uint8_t reg, uint8_t *val, uint8_t antVal)...
void reset(void)...
uint8_t GetReg(uint8_t reg)...

/****************nrf-setup*************************/
void nrf24L01_init(void)...

/*******************Reciver functions*****************************/
void receive_payload(void)...

/*******************Transmitter functions*****************************/
void transmit_payload(uint8_t * W_buff)...

/*******************Main*****************************/
int main(void)
{
    InitSPI();
    ioinit();
    INT0_interrupt_init();
    nrf24L01_init();

    SETBIT(PORTB,0);        //To se the LED is working an the chip is on
    _delay_ms(1000);
    CLEARBIT(PORTB,0);

    while(1)
    {
    }
    return 0;
}

ISR(INT0_vect)  //vektorn that is run when transmit_payload succeed or when receive_payload received data ...
```

If you are using the devise as a transmitter, I usually have an USART-interrupt function called "ISR(USART_RX_vect)" at the very bottom that triggers when the computer sends something to the microchip. In the usart interrupt vector it then calls transmit_payload with the data received from the usart.

If you don't use usart, in the main while loop, send the data as described above "Transmit data".

If you don't use the INT0-interrupt to check if the transmission succeeded put an if-statement in the main while loop to check whether the correct IRQ flag (nr 4) in the STATUS register is cleared as described earlier.

If your chip is in receiver mode, in the main while loop, i usually call:

```
while(1)
{
    reset();
    receive_payload();
}
```

And if i don't have an interrupt, followed by an if-statement to see if anything was received by checking if the correct IRQ flag (nr 6) is set!

Long range mode

If you have the + version, than you can set the RF_SETUP byte to 0x27 instead of 0x07, which will enable the 250 kbps mode (long range) on full power, and i also recommend you to read this tutorial on how to build an amplifying biquad antenna. (remember to set the EN_AA-delay to at least 500us as described above)

Registers

This is straight from the datashet of the older version of the nRF24L01 (i find it easier to read thean the + version)

This is the layout of the "top level" registers as i call them:

Instruction Name	Instruction Format [binary]	# Data Bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read registers. AAAAA = 5 bit Memory Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write registers. AAAAA = 5 bit Memory Map Address *Executable in power down or standby modes only.*
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation will always start at byte 0. Payload will be deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Used in TX mode. Write TX-payload: 1 – 32 bytes. A write operation will always start at byte 0.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, i.e. acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last sent payload. Packets will be repeatedly resent as long as CE is high. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

And here are all the other registers and there configurations:

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the IRQ pin 0: Reflect RX_DR as active low interrupt on the IRQ pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the IRQ pin 0: Reflect TX_DS as active low interrupt on the IRQ pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the IRQ pin 0: Reflect MAX_RT as active low interrupt on the IRQ pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' – 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0:POWER DOWN
	PRIM_RX	0	0	R/W	1: PRX, 0: PTX
01	EN_AA Enhanced ShockBurst™				Enable 'Auto Acknowledgment' Function Disable this functionality to be compatible with nRF2401, see page 26
	Reserved	7:6	00	R/W	Only '00' allowed
	ENAA_P5	5	1	R/W	Enable auto ack. data pipe 5
	ENAA_P4	4	1	R/W	Enable auto ack. data pipe 4
	ENAA_P3	3	1	R/W	Enable auto ack. data pipe 3
	ENAA_P2	2	1	R/W	Enable auto ack. data pipe 2
	ENAA_P1	1	1	R/W	Enable auto ack. data pipe 1
	ENAA_P0	0	1	R/W	Enable auto ack. data pipe 0
02	EN_RXADDR				Enabled RX Addresses
	Reserved	7:6	00	R/W	Only '00' allowed
	ERX_P5	5	0	R/W	Enable data pipe 5.
	ERX_P4	4	0	R/W	Enable data pipe 4.
	ERX_P3	3	0	R/W	Enable data pipe 3.
	ERX_P2	2	0	R/W	Enable data pipe 2.
	ERX_P1	1	1	R/W	Enable data pipe 1.
	ERX_P0	0	1	R/W	Enable data pipe 0.
03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' – 5 bytes LSByte will be used if address width below 5 bytes
04	SETUP_RETR				Setup of Automatic Retransmission
	ARD	7:4	0000	R/W	Auto Re-transmit Delay '0000' – Wait 250+86uS

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
					'0001' – Wait 500+86uS '0010' – Wait 750+86uS …….. '1111' – Wait 4000+86uS (Delay defined from end of transmission to start of next transmission)[14]
	ARC	3:0	0011	R/W	Auto Retransmit Count '0000' –Re-Transmit disabled '0001' – Up to 1 Re-Transmit on fail of AA …… '1111' – Up to 15 Re-Transmit on fail of AA
05	RF_CH				RF Channel
	Reserved	7	0	R/W	Only '0' allowed
	RF_CH	6:0	0000010	R/W	Sets the frequency channel nRF24L01 operates on
06	RF_SETUP				RF Setup Register
	Reserved	7:5	000	R/W	Only '000' allowed
	PLL_LOCK	4	0	R/W	Force PLL lock signal. Only used in test
	RF_DR	3	1	R/W	Data Rate '0' – 1 Mbps '1' – 2 Mbps
	RF_PWR	2:1	11	R/W	Set RF output power in TX mode '00' – -18 dBm '01' – -12 dBm '10' – -6 dBm '11' – 0 dBm
	LNA_HCURR	0	1	R/W	Setup LNA gain
07	STATUS				Status Register (In parallel to the SPI instruction word applied on the MOSI pin, the STATUS register is shifted serially out on the MISO pin)
	Reserved	7	0	R/W	Only '0' allowed
	RX_DR	6	0	R/W	Data Ready RX FIFO interrupt. Set high when new data arrives RX FIFO[15]. Write 1 to clear bit.
	TX_DS	5	0	R/W	Data Sent TX FIFO interrupt. Set high when packet sent on TX. If AUTO_ACK is activated, this bit will be set high only when ACK is received. Write 1 to clear bit.
	MAX_RT	4	0	R/W	Maximum number of TX retries interrupt Write 1 to clear bit. If MAX_RT is set it must be cleared to enable further communication.
	RX_P_NO	3:1	111	R	Data pipe number for the payload

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
					available for reading from RX_FIFO 000-101: Data Pipe Number 110: Not Used 111: RX FIFO Empty
	TX_FULL	0	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
08	OBSERVE_TX				Transmit observe register
	PLOS_CNT	7:4	0	R	Count lost packets. The counter is overflow protected to 15, and discontinue at max until reset. The counter is reset by writing to RF_CH. See page 14 and 17.
	ARC_CNT	3:0	0	R	Count resent packets. The counter is reset when transmission of a new packet starts. See page 14.
09	CD				
	Reserved	7:1	000000	R	
	CD	0	0	R	Carrier Detect. See page 17.
0A	RX_ADDR_P0	39:0	0xE7E7E7E7E7	R/W	Receive address data pipe 0. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0B	RX_ADDR_P1	39:0	0xC2C2C2C2C2	R/W	Receive address data pipe 1. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0C	RX_ADDR_P2	7:0	0xC3	R/W	Receive address data pipe 2. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
0D	RX_ADDR_P3	7:0	0xC4	R/W	Receive address data pipe 3. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
0E	RX_ADDR_P4	7:0	0xC5	R/W	Receive address data pipe 4. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
0F	RX_ADDR_P5	7:0	0xC6	R/W	Receive address data pipe 5. Only LSB. MSBytes will be equal to RX_ADDR_P1[39:8]
10	TX_ADDR	39:0	0xE7E7E7E7E7	R/W	Transmit address. Used for a PTX device only. (LSByte is written first) Set RX_ADDR_P0 equal to this address to handle automatic acknowledge if this is a PTX device with Enhanced ShockBurst™ enabled. See page 14.
11	RX_PW_P0				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P0	5:0	0	R/W	Number of bytes in RX payload in data pipe 0 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
12	RX_PW_P1				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P1	5:0	0	R/W	Number of bytes in RX payload in data pipe 1 (1 to 32 bytes). 0 Pipe not used

I hope you enjoyed my tutorial...
as always, if there is questions there might be an answer in the comment field.
/Kalle

Upplagd av Kalle Löfgren, kl. 19:24 +1 +4 Rekommendera detta på Google

Etiketter: Atmega88, ATtiny26, ATtiny85, AVR, nRF24L01, RF
Plats: Umeå, Sverige

182 kommentarer:

Andreas Kristensson 10 april 2013 10:00

Kalle, great tutorials! Will use a Raspberry to control my sun shades (and then Nexa switches, etc) - the information you provide on your blog is spot on. Thanks

Svara

Svar

Kalle Löfgren, 10 april 2013 10:12

Thanks, glad to hear someone has use of the blog =)
If you have any questions just post them here, and i might be able to help!

Kalle Löfgren, 28 maj 2013 09:46

Have a look at my new blog post on how to control nRF24L01 with RPi with a python program (http://gizmosnack.blogspot.se/2013/05/raspberry-pi-nrf24l01-and-tcp.html)!

very easy programming, and the hardware setup couldn't be easier (only needs a cable in between)!

Svara

dan spataru 18 april 2013 21:23

great tutorial. unfortunately i still can't make my pair of nrf24l01 to communicate. i want to connect two atmega32 on two computers (thanks for the UART tutorial, it works) and send a caracter from a computer to another. hope i have enough brain to make this thing work

Svara

Svar

Kalle Löfgren, 20 april 2013 00:49

Hi Dan!
Where does it go wrong?
Do you get the SPI-communication to work between the atmega and the nrf?

dan spataru 20 april 2013 13:02

the character.
I changed the config register
val[0]=0x1F;
WriteToNrf(2, CONFIG, val, 1);//the second nrf is a receiver
And that's all. I do not know how to put on screen the character received. It must be something like USART_Transmit(the caracter received).

Svara

Kalle Löfgren, 20 april 2013 15:18

Ah, i realize now that i have forgotten to add the defines:
#define W 1
#define R 0

in the tutorial above..
i see you have put a "2" as read or write. in that case it will never return anything from the nrf (the if statement takes the 2 as a "Write" command)
Put the defines in your code, and try calling the function:
val[0]=0x1F;
WriteToNrf(W, CONFIG, val, 1);
on the receiver
Yes send the data to the computer with usart_transmit(data[i]) (in a for-loop that increments the i from 0 to the number of bytes you have received -1)

Svara

Svar

dan spataru 20 april 2013 16:34

thanks for the quick answer. still no luck.
the code for receiveris
while(1)
{
reset();
receive_payload();
_delay_us(10);
data=WriteToNrf(R,R_RX_PAYLOAD,data,1);
USART_Transmit(data);
}
the code for transmitter (in the while loop) is
reset();
transmit_payload(0x90);
and the data that I receive is 0x60. i do not understand why

Kalle Löfgren, 21 april 2013 11:50

Here is the if-statement i talked about yesterday that checks if anything is received (if the IRQ bit "RX_DR" (bit 6) is set, data is received):

reset();
receive_payload();

if ((GetReg(STATUS)&(1<<RX_DR))!=0)
{
data=WriteToNrf(R, R_RX_PAYLOAD, data, 1);
USART_Transmit(data[0]);
}

Svara

Kalle Löfgren, 21 april 2013 01:44

Ok, first of all, there is no payload data to read from the nrf if the receive_payload didn't receive anything during its 1 second of listening...
there is no point of reading out the payload if there is nothing in it!
either you use the if-statement described above, or my advise is to use the interrupt (also described above) that triggers when the receive_payload actually gets a payload!
In the interrupt, you then calls the "data=WriteToNrf(R,R_RX_PAYLOAD,data,1);"
and because the "data" is a vector, i think that you even if you just sends one byte, have to send if to the computer by:
USART_Transmit(data[0]); //(sends the first byte in the vector to computer)

Svara

Liu Peng 26 maj 2013 05:01

hi, Kalle, this is great tutorial! But maybe I typped something is wrong, I use Atmega16a, nRF24L01+, But Now It seems that even the GetReg() function do not return correct answer!

My code snippet in github gist: https://gist.github.com/van9ogh/5651555

actually, when i read the GetReg() from serial, the value is 0x1E. sometimes it change to 0x1F!

Svara

Svar

Kalle Löfgren, 26 maj 2013 11:53

(sorry if you got my last reply, this is how it is suppose to bee:)
Hi! I have checked your code, and I have a couple of questions:
1. The reset(void) should be "0x70" (line 107) and not "0x07" (i see i wrote 0b70 (wrong) up in the tutorial...)
2. Line 176 "}Kalle" ? yes that is my name ;)
3. I highly recommend you to set the SETUP_RETR to "0x2F" as described above (When using EN_AA)
4. Why do you initiate the interrupt-ISR-function when you are not using it (or even have the USART initiated)
5. Actually the STATUS registry is suppose to bee 0x0E (thanks, I will change my tutorial, I mixed up the binary to hex when reading the data sheet...)

When the "1" appears before the "E", it means that maximum number of retries has accrued (you must have tried to send something without a successful transmission (and EN_AA on!).
When status turns to 0x1F, the F tells you that you have filled your TX FIFO (the cash that stores your payload!) when this happens, i guess the FLUSH_TX is not working..
But your code does not call the Transmit-function, so this is strange!

Liu Peng 26 maj 2013 14:17

thank you! Kalle, I have set SETUP_RETR to "0X2F", Now the GetReg(STATUS) return 0x0E!, You know, I have debugging this device for two days! this small progress make me so exciting!

related to this question, so I hide some functions.

last, sorry for my careless in line 176 "jKalle" ;-)

Kalle Löfgren, 26 maj 2013 14:36

Sweet!
Glad i could help =)
I posted the code on github as well: https://gist.github.com/klalle/5652658 (easier to copy/paste)

Svara

Nambela_Smith 31 maj 2013 01:16

Nice tutorial Sir,

Btw would you like to send me your email??

you can send here too

jony.nambela@gmail.com

thanks

Svara

swharden 9 juni 2013 04:31

I notice you use screenshots for your code. It's hard because you can't search it and can't copy/paste it. I assume the reason is for syntax highlighting. Consider Google's prettify: https://code.google.com/p/google-code-prettify/

I use it all the time, see the code on: http://www.swharden.com/

Svara

Svar

Kalle Löfgren, 10 juni 2013 14:26

Thanks man, will look into that for my next posts! (I have posted links to my github page where you can copy and past from!)

Svara

Trung Phan Văn 9 juni 2013 20:31

code Tranceiver:
http://www.mediafire.com/view/ih5a3k26xm6kci9/E3.c
code Receiver:
https://www.mediafire.com/view/eft42po1cbxe55u/receive.c
I use pic 16f887.
Can you help me find error in my project?
Thanks Svara

Svara

Svar

Kalle Löfgren, 10 juni 2013 14:48

First thing i notice in the transmitter-code:
line 341: W_buffer[1] = 0x90;
You only sets the second bit ("1") in the 5-byte array... you want to set all the 5 bytes to something since you have declared on line 262-263 that you will send 5 bytes each time!
have a look at this code:
http://3.bp.blogspot.com/-Ea8Q1HoFwbM/UWLhxjsr1ZI/AAAAAAAAAsk/vQgZxZC0LF0/s1600/Capture21.PNG

On line 345, why do you send 0x1e to the status register? it will wipe the MAX_RT flag (the "1"), and do nothing else ("e")..
Since you clear the MAX_RT flag in the status register, line 350 will always bee false!

hope this helped you somehow! (if you want to send only one byte at a time, change line 262 to one, and use W_buffer[0] = 0x90;)

Svara

Mircea X 26 juni 2013 20:24

Very good post.
One question, I cannot understand why I should send "n" dummy bytes when I want to read something.
10x

Svara

Svar

Kalle Löfgren, 27 juni 2013 22:40

"n" is the number of bytes... the nrf returns a single byte evert time you write a byte to it, which means you have to send 10 dummy bytes to receive 10 bytes from the payload for example....

Mircea X 28 juni 2013 15:22

I am a beginner in the uC world. Now I have read the SPI protocol and I understood that for each bit sent to the Slave, the Slave returns 1 bit.

Svara

snoop 26 juni 2013 21:06

Hi, sorry i am french so i will speak english with my "bad" english :D

So, i have make your tutorial on ATTINY 2313A.
I arrive to receive the STATUS register 0x0E
I arrive to SET something like the TX adress etc (i have try to set, and read)

void transmit_payload(uint8_t * W_buff)
void receive_payload(void)

I don't know if it realy works but with
((GetReg(STATUS) & (1<<6)) !=0) and ((GetReg(STATUS)& (1<<4))==1) verification it's look likes OK (but i doesn't verify with scope)

Where i have a problem is when i start the function : void nrf24L01_init(void) :
Then i can Receive always OK.
If i try to send something : the program is like "stoped" or "paused" (it enter in the fonction "void transmit_payload(uint8_t * W_buff)" but doesn't go out !

So i see that i have to change the CONFIG for be in TRANSMITER MODE, but if i see the datasheet i have just to put the value to 0x1F for be in TRANSMIT mode and 0x1E for be in received mode (for exemple)
But when i call the void transmit_payload(uint8_t * W_buff) it do the same it enter in the fonction and the program is like "halted" i can't reay debug, i use serial communication for see where i go in, and i see that i go in the fonction and something crash in it ..
Can you help me ? Did you know something that i doesn't know ? is there a special action to do for transmit something ?
You can contact me at cybermath1@gmail.com but i can be connected on irc.freenode.org if you want

Svara

Svar

snoop 26 juni 2013 21:25

I locate the problem, in the nrf24l01_ini() when i remove the part for shoose if the module is RX or TX :

// val[0]=0x1F; //0b0001 1110
// WriteToNrf(W, CONFIG, val, 1);

It's realy curious, i will try now to see what is in CONFIG register by default

snoop 27 juni 2013 20:39

Ok now my problem is solved, if someone want, i can upload my code he is fully functionnal for ATTINY2313 Recepter and transmiter, thanks for your tutorial Gizmosnack and sorry for "flood"

Kalle Löfgren, 27 juni 2013 22:45

Glad to hear you figured it out! please post your code on github (or anywhare else), and a link here, so that others can get helpe from your code!

M Sree Abhinav 16 januari 2014 10:22

Can you upload the code for Attiny2313 snoop

Róbert Žalman 2 maj 2014 16:15

Is this code still avalible? or can be upload somewhere?
At least thx for reply.

Fernando Ortega 19 oktober 2014 21:36

Has anyone this code ?
Best regards,

Svara

NGUYEN MINH Vu 13 september 2013 16:28

nice to meet you!
Sorry about my skill english. because it't terrible.
why do we send 0x93 in code:
"And you call the transmit function like this: (now i just send 0x93 five times in a row, you can fill the array with any bytes you want to send)"

Svara

Svar

Kalle Löfgren, 13 september 2013 21:34

Hi!
it was just an example!
you change the 0x93 to the byte that you want to send!

NGUYEN MINH Vu 14 september 2013 04:36

^.^ ! thanks so much.
im very interestted in style your code. very clear and easy to understand

Kalle Löfgren, 14 september 2013 09:39

Thanks! Im not an expert programmer so i try to make it as simple as possible!

Svara

NGUYEN MINH Vu 18 september 2013 05:30

here my code : http://www.mediafire.com/?apmst7dmwmwa0sb
Im using atmega16 with crystal 11.0592

Svara

Svar

Kalle Löfgren, 18 september 2013 09:37

It's a bit time consuming to error check you code since you have your own way of doing things, but in general it seams like you have got the hang of it! I can't find anything wrong with it by just looking through it!
Here is how i would do the error checking:
1) make sure you have a working USART setup, so that you can send things from the atmega to the computor (to termite.exe or any other usart program)

USART)

3) Make sure step 1, and 2 is working on two setups, so that you can have one as a transmitter and one as a receiver (if you only have one USART=>USB, make sure the SPI works on both atmega by trying one at a time with USART)

4) Configure on of the nRF to a receiver (with all the settings i wrote about in the blog), and confirm by reading out the "config" register to the computer. Set it to listen for data, and make sure to reset its status register regularly.

5) configure the other nRF to a transmitter (with all the settings i wrote about in the blog) and confirm by reading out the config register.

6) transmit something to the receiver, and after every transmission, read out the status register to see if it was successful or not!

Good luck, sorry i couldn't find your problem...

Svara

NGUYEN MINH Vu 18 september 2013 17:37

thanks for your intructions!
i'll try.
thanks you Kalle!

Svara

Karol Tatar 21 oktober 2013 00:57

Greatings from poland,

Perfect tutorial,

it has helped me a lot. I rewrote your code on smt32f4 as a transmiter and atmega 32 as a reciver (there were no changes in that one except from pins).

Again excelent work. In next couple of days i will post my code if anyone have problems with stm32f4 (i hope it will help someone someday)

Svara

Svar

Kalle Löfgren, 22 oktober 2013 22:05

Glad to hear I could help! =)

Konstantin Glushkov 23 oktober 2013 01:19

Karol, hope to see your avr code =)
Kalle, great tutorial, im newble in avr, so trying understand it all))) also see your instruction about nrf && rpi, hope soon to connect mega32 with rpi =)

Svara

Unknown 16 november 2013 19:22

Hi Kalle!
First of all: great tutorial :) However I've a little problem: my transmission fails all the time. (I don't use auto-acknowledgement, I just have the same adress on my both nRF.) When I check registers - they all look just like they should, so it can't be problem with communication with nRF24l01. My sending function, and configuration are the same as yours. Is the lack of auto acknowledgement a problem? Or shall it be with some other stuff?

Svara

Svar

Unknown 16 november 2013 21:20

In fact it works, but only once - it looks like it's full after just one transmission. I tried flushing receiver manually after that, but still it won't work with me like I would love it to :) Sorry for multiple comments.
Well, at least I know it's not issue with auto-acknowledgement, still the same problem after enabling it.

Kalle Löfgren, 16 november 2013 21:58

Hi!
Glad to see this many people using my code =)
Try reading out the status register on both the transmitter and receiver before transmission, after first successful transmission, and after second failed transmission and post the results here if you are having trouble understanding them... hopefully that will tell you if its a problem on the transmitter or receiver side and what it might bee.
Good luck!
/Kalle

Unknown 17 november 2013 19:44

It's very neat code :)
I've my answer now - transmitter firstly send packet than on receiver in interrupt CE is being cleared and than... it newer went on again - so it was never ever listening again. It's like that in your code too, does it lack there, or it's meaningful?
After this little change it works like it should :)

Kalle Löfgren, 17 november 2013 20:04

Congrats!
Well, it is correct that i turn off CE in the receiving interrupt.
The purpose of this is to turn off the listening for new data, so that i can read out the received data and not having to worry that new data will be received during the read out, which would corrupt the last package...
The CE pin is turned back on, when i call the "receive_payload();" function, which on a receiver is done in every loop in the "main loop"...

Syed Muhammad Ali 18 november 2013 12:03

hi sir,
I have nrf24l01+ long range module, it is working fine in straight line of sight (300 m) but not working perfectly when i used in industrial environment due to obstracts, pillars and walls. while my required range is maximum 200 meters. Please guide me if i made any mistake and which one is better for mine nrf24l01+ or nrf905. Because gsm band also use 800 to 950mhz freq. Waiting for ur prompt reply
thanks

Interesting!
Well, i am not an expert in radios but too me it sounds like the 905 should be the better one in long range since it is working in the MHz freq compared to the 24l01 which uses 2,4GHz. Have you tried the 488MHz setting on the 905?
If that doesn't work, try making a biquad antenna like this one: http://martybugs.net/wireless/biquad/ And make sure you are tilting them the right way!
Looking forward to hear how your setup comes out in the end! =)

Kalle Löfgren, 18 november 2013 23:28

Another solution would be to put a transceiver in between your transmitter and your receiver, and have it resend the data it receives from the transmitter, to the receiver, that way there is no limit in how far you can reach! Good thing these little gadgets are so cheap =)

Svara

Syed Muhammad Ali 25 november 2013 10:05

Thanks for your suggestion sir,
Actually I m facing problem with obstacle, pillar . Range is too good of nrf24l01 which i have tested up to 400 m in air without obstacle. But in shed of textile loom machines it sometimes work or sometimes not due to iron stands, pillar, and other industrial equipment in the shed. Range is not issue, obstacles are creating problem.
My query is how mobile signal works between lot of walls, conjusted places. Why we cannot use same frequency to as mobile phone. Which module will be better for this purpose.

Svara

Syed Muhammad Ali 25 november 2013 10:08

I don`t have nrf905 module, if you suggest then will purchase.
thanks

Svara

Svar

Kalle Löfgren, 25 november 2013 19:10

Hi!
200 meters indoor is quite allot! Did you say the transmission succeeds when the machines are not in use? if so, i'm impressed!

As i said, i have worked with both the 24l01+ and the 905 (488MHz), but have not tested the range difference... If you decide to try with the 905, it works on almost the same way as the 24L01, even if you need to connect a few more pins to get the 905 working. This shouldn't bee a problem if you understand the code for 24L01 and read the manual (https://www1.elfa.se/data1/wwwroot/assets/datasheets/07300809.pdf), but if you need help, i have a working code...

Anyway my best advice would be, as i said, to use a third of your 24L01+ and put it as a node in between the two you want to connect, and have it receive the payload from the transmitter, and forward the payload to the receiver. This would half the distance, and hopefully be enough. If not, use a forth node, or a cable ;)
Good luck

Svara

Unknown 3 december 2013 10:51

Hi! Very nice tutorial!

I would like to make a low consume repeater (a device that sends to other devices whatever he receives from another). I did it by using an arduino pro mini (atmega328) and MIRF library but now I want to try using atmega88. Thanks to your tutorial I learn a lot about the internal operation of the nrf and now I'm wondering some things:

1. Can I use one channel for read and other channel for write at the same time?
2. In my actual design with atmega328 I use the IRQ signal to wake up the atmega and when message is repeated go to sleep. Do you think that this behavior could affect your code?
3. I need to save as much energy as I can (my devices are solar powered) but reading the datasheet I didnt found anything to save energy in RX mode. Do you know something about that?

Thank you!

Svara

Svar

Kalle Löfgren, 3 december 2013 15:16

Hi, and thanks!
I'll try and answer your questions, but remember that i'm not an expert in these modules, and havn't been working with them for a while ;)

1. Yes that should work, just use different addresses for receiver and transmitter. Make sure you setup the Auto ACK the correct way (read page 13-14 in the data-sheet, especially the picture on top of page 13 and nr 2 on page 14: https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf)

2. That would definently bee a good idea! set the Atmega to wake on interrupt caused by the IRQ pin on the nRF.

3. Ok, if you don't need the range, go with the older version of the nRF and set it to the 1000 kbps mode. it takes 11,8 mA when set to 1Mbps and 12,3 mA in 2Mbps mode, see page 5. (The nrf+ version takes 12,6 in 250kbps, 13,1 in 1Mbps, and 13,5 in 2Mbps mode, see page 14 in its datasheet)

an alternative option would be to have the receiver in powered down mode ~95% of the time (guessed this value), and have it started by the atmega on a short intervall only to listen for data... this might work if you set the transmitters to try and resend failed package for maximum amount of time, and with long time in between every retry... but you would have to calculate on it to see that the wakeup-time for the receiver is less than the full cycle of transmitt-retries (or have a loop on the transmitter that manually retransmitts the data untill it succeeds)

Good luck, please let me know how it turns out, and if you need more help, just post the question here.
/Kalle

Svara

Friezca vebby 9 december 2013 05:55

sir may I ask for your email?
I need your help, I am still confused what is the difference between NFR905 with NRF24L01 ..
and if your program that are above it can be used in nfr905??
please answer ..
thanks

Svar

Kalle Löfgren, 9 december 2013 10:58

Hi! Sure you can have my email, but i prefere not to put it on my blog due to the risk of getting lots of mail with questions... If you post your mail, i will respond!
The difference with the two rf-modules are that one uses 488MHz, and the other 2,9GHz to send data. You need to modify the code above slightly to get it to work on the 905, but not much!

Svara

jensgulow 9 december 2013 20:05

Hi Kalle,
today I found this tutorial and would be happy to find a working solution for the nRF905. Would you be as kind as to send the code to me?

Thanks

Jens from germany

Svara

Svar

Kalle Löfgren, 9 december 2013 21:11

Here is code for the 905, First included h-file, then receiver, last transmitter... (use translate.google.com or something to understand Swedish comments): https://gist.github.com/klalle/83ec2a1a691523f2829f

Can't promise they will work as is, as you see i was mocking with them august 2012, but i remember that i had no problem getting them to work with this code!

Notice the TXE-pin (start SPI) and the PWR (Power) pin, which the 24L01 doesn't have, otherwise you use the same code as above!

jensgulow 10 december 2013 17:48

Thx a lot!

Kalle Löfgren, 11 december 2013 21:50

I see now that the code i sent you is an older version of my program... When you have got that to work, use the "WriteByreSPI"-function from the blogpost above to easily send/receive array of bytes!

Svara

Friezca vebby 12 december 2013 16:07

can you tell me the point that needs to be replaced ... because I am a new beginner and not very familiar with NRF905 but this relates to my final project on campus

Svara

Svar

Kalle Löfgren, 12 december 2013 16:26

Well, see the code i just posted to "jensgulow" with the code to the 905: https://gist.github.com/klalle/83ec2a1a691523f2829f
This is an older version of the code, so change the "WriteByteSPI"-function to the version i use on the 24L01 code above!
/Kalle

Svara

Friezca vebby 15 december 2013 11:22

If I want to try to do my work, NRF905 what should I do.?

Svara

Svar

Kalle Löfgren, 17 december 2013 08:23

Sorry, i don't quite understand your question there...?
Do you want me to give you school project examples of what is possible to do with the nRF?

Svara

Cristi Petrescu 30 december 2013 17:44

Hi there. Nice tutorial.

I am desperatly tring to make the NRF to work. I am using 2 pic 16f877a. I wrote my program in CCS C and i am able to read, write the internal registers of the nrf but i cant transfer any data between the PICs. Do you have any idea or hint?
I posted here the source code:
http://www.sourcepod.com/kvrgni03-21521
and here the header for the nrf
http://www.sourcepod.com/agmjuo15-21520

This is a school project and i would be very grateful if you could help me!
Thanks in advance!

Svara

Svar

Kalle Löfgren, 2 januari 2014 16:26

Your links does not seem to work... anyway, if you have got a working setup where you can transfer bytes to and from the nRF, you should have no problems following my examples above... here is a link to full code

Svara

Comp123 13 januari 2014 14:16

Is there any input voltage difference between nrf24L01 and nrf24L01+ . The nrf24L01 chip itself should be powered at 3.3V.At present for nrf24L01+ also I am providing same range in between 1.9V-3.3V. Does nrf24L01+ demands change in supply input voltage. nRF gets peak power on transmission start so by trying for electrolytic capacitor to nRF power line will it be helpful.

Svara

Svar

Kalle Löfgren, 13 januari 2014 14:20

I have no idea... see datasheet for that!

Svara

Comp123 13 januari 2014 14:27

Range is the concern in my project so I am using nrf24L01+ advanced RF. Do you have any nrf24L01+ RF specific examples for reference posted anywhere. Thanks

Svara

Svar

Kalle Löfgren, 13 januari 2014 15:07

Don't really know what you are talking about!?
As i have said in the blog post, there is not much difference at all with the + version except that you can us the 250kbps instead of 1Mbps which is the lowest speed the old version can manage. This increases the maximum length dramatically! But you can use exactly the same code if you're not using the 250kbps setting!

Kalle Löfgren, 13 januari 2014 15:12

See this datasheet on bottom of page 55 to set the 250kbps. Set the RF_DR_LOW bit in the RF_SETUP registry

Svara

Friezca vebby 16 januari 2014 11:44

Can you make articles about nrf905 thank you, i really need your help

Svara

Svar

Kalle Löfgren, 16 januari 2014 12:49

Well, no, because it is almost exactly the same as for the 24L01!
I get the impression that you just want a working code, without understanding it!? A search which will give you this as first hit, which is exactly what you want!

Svara

Marcin Dudek 22 februari 2014 15:55

Hallo Kalle!
First of all i have to admit that this is a great tutorial:) You help me a lot but i still have some problem with communication:(Maybe you can help me becouse i have no idea what to do now. As i noticed nRF does not request interrupt so as i guess it has to be problem with transmission. Moreover, once I managed to get a stable connection but only once... Please could you look at my code and help somehow? SPI communication is OK... Please help if you can:)

transmiter code: http://wklej.org/id/1279926/

Svara

Marcin Dudek 22 februari 2014 18:27

i checked STATUS register of transmitter, right after power up status is 0x0E, when transmission is over STATUS register is 0x2E and after reset again 0x0E. Everything seems to be ok but nRF does not generate interrupt signal on IRQ PIN...

Svara

Kalle Löfgren, 24 februari 2014 19:18

You seem to have understood the concept!
I might be wrong, but in your main while loop you try and transmit one time, and if that fails, you go into this funcion:
while(!(GetReg(STATUS)&(1<<5)))
SETBIT(PORTD,6);
Which is a never ending loop!

Because your code never jups out of the loop you will never try a secound time, isn't that a bit useless?
I would start by just transmitting your payload every 1000ms or something, (remember to reset in between) and reading out the STATUS registry after every time!
something like this in the main while loop:

```
while(1){
reset();

transmit_payload(Buffer);

_delay_ms(100); //whait a while before you check the status, give the receiver som time...

if(!(GetReg(STATUS)&(1<<5))==0){ //checks if TX_DS is set to "1" => data sent
SETBIT(PORTD,6); //LED on
_delay_ms(5000); //indicate successfull transmission with a looooong LED
}
CLEARBIT(PORTD,6); //LED off
_delay_ms(1000); //Whait 1s before next transmission
}
```

Post your receiver code and i'll have a look!
Good luck

Svara

and i can still see, change not happened at status what i trying to read...
RX led is high all the time
TX led is low all the time

have you ideas how i test the spi communication to the nrf? is there possibly some standby-power-status register?

Kalle Löfgren, 8 april 2014 12:46

Well, to test if the SPI is working, read out the status registry bits one by one, and see what it sais... if its 0x00 or 0xff than you don't have a working setup...

One way of showing the byte could be to connect 8 LED's to the chip (PORTB 0-7 for example) and set the PORTB = GetReg(STATUS);
Otherwise you can loop through the bites in the STATUS and let the LED show each bit at a time:

```
While(1){
for(i=0; i<8; i++){
if((GetReg(STATUS) & (1<<i)) != 0){
SETBIT(PORTB, 0);
_delay_ms(1000);
{
else{
CLEARBIT(PORTB, 0);
_delay_ms(1000);
}
}

SETBIT(PORTB, 0); //to get LED-ON for five seconds in between every read out
_delay_ms(5000);
}
```

asdfgh 8 april 2014 13:28

my atmel studio give a many warnings with that integer test and i cant make that code relevant.. so i was made this other PORTB test.

i get next ressult. PB1, PB3 and PB5 was high and rest of portb was low.. mean working?

Kalle Löfgren, 8 april 2014 13:47

Good your SPI seams to be working!
Bit nr 5 in status reg => TX_DS high (data is sent)...
Bit 1-3 =>data pipe number for payload, means you have data to read..

asdfgh 8 april 2014 18:06

Sorry this flooding but there is somethin what i dont absolutely understand..

1. when i cheking the rx status 1<<6 at the following code, led is blinkin (i think this is ok but..).. The big suprice is that, when i take the tx side power off the led will blinkin ater that... so is that possible, the rx side 1<<6 register will switching between 1 and 0 even without there is no packets for receiving..

2. wich register i must read, when i want read i.e receiving that "0x93", is it at getreg funkition 'reg' register or spi SPDR?

here is the question 1 code:

```
while(1)
{
reset();
receive_payload();

if (((GetReg(STATUS) & (1<<6)) == 1));
{
SETBIT(PORTB, 0);
_delay_ms(200);
}

if (((GetReg(STATUS) & (1<<6)) == 0));
{
CLEARBIT(PORTB, 0);
_delay_ms(100);
}
}
```

and thaks you very much Kalle, this wreally do not so simple..
I try to this message is last question..

Kalle Löfgren, 9 april 2014 08:09

I think you ar doing the check wrong... it should be "!=0" instead of "==1"

Svara

goutham b r 10 april 2014 08:36

Hi,
Can this code be used for interfacing NRF24L01+ with ATmega8L?
Will these nrf modules work in mesh topology?
i'm using this for a short range i.e about less than 50 meter wireless communication!
Thanks in advance!
-Goutham

Svara

Svar

Kalle Löfgren, 10 april 2014 09:39

Hi!
Yes, should not be any problem using this code with the Atmega8L!
What kind of mesh topology are you planing to use? yes you could send data to several nodes and receive from several nodes with one nRF, so should not be a problem!
/Kalle

Svara

Swapnil Tandel 17 april 2014 17:12

Hello Sir your tutorial is very good..!
but i have a question that whether calling reset() resets TX_ADDR..?

Svara

Svar

Kalle Löfgren, 18 april 2014 00:18

Thanks!
No, it only resets the bits in the STATUS-registry (see RX_DR, TX_DS and MAX_RT)

Svara

Swapnil Tandel 18 april 2014 08:43

Thanx for your reply sir...

Sir i need your help, i was trying to interface two NRF24L01+ with two different micros since last weak but don't know whats wrong with my code..
If possible have a look at my code and please correct me...

Transmitter Code: http://pastebin.com/vzYvkiXx
Receiver Code: http://pastebin.com/i9dKX4F2

I am not able to receive a single char in above code.

Svara

Svar

Kalle Löfgren, 18 april 2014 13:48

I think you need to set the RX_ADDR_Po = TX_ADDR if the EN_AA is enabled (Which it is!)
This is because the receiver will receive the package, and return it to the same address...
/Kalle

Svara

Swapnil Tandel 18 april 2014 08:45

But my SPI communication with NRF is working..!

Svara

Swapnil Tandel 19 april 2014 19:15

Thanx for your reply sir... :D

Sir but I am still not able to receiver data at receiver side.
My SPI communication is working and i have checked it with GetReg() function by reading the seated values like CONFIG reg....
Please sir help me out sir please....

My code is given bellow..
Receiver: http://pastebin.com/u8EfmTdu
Transmitter: http://pastebin.com/cbYtiT00

Svara

Svar

Kalle Löfgren, 20 april 2014 01:46

Try remove the 1000ms delay at the end of the while-loop on the receiver code... during that time, the transmitter wont be able to send you anything because you are not listening!

Swapnil Tandel 20 april 2014 15:26

Sir, Now my NRFs are communicating and the problem was the bad power supply...
Thanx sir for helping me ... :D

But, Sir how to switch from Transmitter to Receiver ...?

Kalle Löfgren, 20 april 2014 17:56

Congrats! =)
It's easy, just change the CONFIG-registry from 0x1E to 0x1F (this have to be done when in power down mode => 100ms after last transmission or reseiving-session)

Svara

Piotr Kaczmarczyk 30 april 2014 21:44

Hello and greetings from Poland :) I am trying to communicate betweend two nRF24L01+ with ATmega8A as transmitter and ATmega32A as receiver. I am going to use it in RGB propeller display in future to send an image wireless. Right now I just want to send one byte stored from the beginning in program memory (I am not using USART) and put this byte on LED port.
I am generally using your code, with just a little change on pins. However I can't make my transmission success. I did checking the registers like you wrote in one of your posts like this:

```
check=GetReg(CONFIG);
reset();
if(check==0x1E)
{
while(i<5)
{
SETBIT(PORTB,0);
_delay_ms(100);
CLEARBIT(PORTB,0);
_delay_ms(100);
i++;
}
}
```

It looks OK for the first but it seems that I can't read back RF_SETUP and SETUP_RETR registers (LED is not blinkig) or these registers have other values then I have stored in them. I laso tried to check if sending a message was succeed with this code:

```
if(!(GetReg(STATUS)&(1<<5))==0)
{
SETBIT(PORTB,0);
_delay_ms(1000);
}
CLEARBIT(PORTB,0);
_delay_ms(1000);
```

but it also failed (LED was not blinkig). I checked power supply and it is very stable so it isn't this. Can you please check my code? I will be very greatful. Here are my codes:
http://wklej.org/id/1348350/
http://wklej.org/id/1348352/
Please help me.
Here is my email if you would like to write to me directly: xadrez@o2.pl

Svara

Svar

Kalle Löfgren, 5 maj 2014 20:18

Hi!
Your code is a bit hard to read due to formating, but i cant find anything wrong with it...
First of all make sure your SPI is working both ways, it seams that you only succeed in reading from the registers, not writning to them!?
Make sure your cables are wired correctly, and try and get this to work before you put too much work on the code!
/Kalle

Svara

Kojot 18 maj 2014 14:18

Hi Kalle,
thank you for this post. Currently I'm using nrf24l01+ in my home alarm project. I'm using board with atmega16 and your code with little modifications. Here is the source: http://pastebin.com/aZUwMFn1 . I got a problem. When I'm checking STATUS register, the board send to me 0xFF, not 0x0E. Could you tell me where the problem is?

With regards
Pawel

Svara

Svar

Kalle Löfgren, 18 maj 2014 17:08

When the Status returns 0xFF it means the SPI is not working!
Check your cables and voltages!

Kojot 18 maj 2014 23:06

I forgotten to supply +3V3 regulator. Now I see in terminal 0x00.

Svara

Kumar Karan Jain 30 maj 2014 11:36

Hello,
thanks for nice tutorial.
I m using NRF24L01+ for first time , with the ATmega328p ,
I not able to read the STATUS register

I m only using following functions just to read STATUS register so that I can confirm SPI is working,

```
USART_Init();
USART_Transmit();
USART_receive();
SPI_Init();
WriteByteSPI();
GetReg();
int main(void)
{
int main(void)
{
clockprescale();
InitSPI();
usart_init();
reset();


char sta = 'D';

while(1)
{

USART_Transmit('R');
```

```
USART_Transmit(GetReg(STATUS));

_delay_ms(100);

}
return 0;
}
```

but not getting STATUS value on Serial monitor.

plz help me..

Svara

Kumar Karan Jain 5 juni 2014 09:01

Hello kalle.....struggling for past 2 days didnt get any sucess....plz give some specific suggestion to do..

Svara

Svar

Kalle Löfgren, 5 juni 2014 10:36

Hi!
Why do you have double main-functions?
Does your USART work when sending the "B"?
Are you sure your serial program is setup so it can show the STATUS byte as decimal or hex and not ACSII (http://www.jimprice.com/ascii-0-127.gif)
Your SPI is obviously not working! check you cables and try again! are you supplying a stable 3,3V to the nrf?

I'm not much help before you have a working SPI, try a different tutorial on how to setup the SPI on a Atmega328p!

Good luck
/Kalle

Kalle Löfgren, 5 juni 2014 10:37

This ASCII-table is easier to read: http://www.asciichars.com/_site_media/ascii/ascii-chars-landscape.jpg

Svara

Irud 6 juni 2014 02:26

Hi Kalle!
Great blog and big respect for spreading knowledge and helping everyone!
I'm trying to communicate between 2 Atmega8 chips, like hello world. Just to send bytes from one and recieve on the other. SPI is working.
What I did is took your code, modded the config register (1E/1F) and added transmit/recieve loops for transmitter and reciever. Is that all?
It's not working and it's hard to debug deeper without USART (still incoming from ebay)...
If you have time please check code here https://github.com/jivkovic/RF/ I really don't know what could be the problem, if code is ok I will try to add capacitor after linear regulator, I'm out of ideas..
After this works I'd like to tidy up code so you can add it as the most basic hello world version (without interrupts, usart,..) if you want. I think it's much easier to learn for people new to uC world that way.

Thank you for your time

Svara

Svar

Kalle Löfgren, 6 juni 2014 18:49

Hi!
You seam to have understood the code and your code seems to be ok!
Try and read out some other register in your test-code to se if you actually succeded to write something to the nrf:

```
if (GetReg(SETUP_RETR) == 0x2F)
{
blinky();
}
```

My experience is that i do not need the capacitors on the linear regulators... maybe it depends on wich one you are using!?

About the tidy hello world-code: I have thought of doing that, but have never got so far... and i think there is a point of letting people read this blog-post very carefully and learn the code instead of getting a working hello-world code. I think the working code would lead to me getting a million questions on how to modify the code to there needs, which is very easy if they have read the blog post and managed to get there own code working!
/Kalle

Irud 8 juni 2014 00:30

It seems that writing succeeds after i started using 5V regulator on one, and 5V from usb on second chip.. makes no sense for me, I think regulator should handle both..
Now only one more problem, verifying recieved data..
Code above always go on the "else" part:

```
if (((GetReg(STATUS) & (1 << 6)) != 0 ))
{
data=WriteToNrf(R, R_RX_PAYLOAD, data, dataLen);
if (data[0] == 0x93)
blinky();
else
blinky2();
}
```

tried comparing it to 0x00, 0xFF, and array of 5 0x93's sent from transmitter, but no success. Any idea?

Irud 8 juni 2014 19:54

Wow how frustrating :) Regulator IS working for both chips. I was getting blinks on reciever because USB voltage was messed. No data was actually recieved. I checked if (GetReg(SETUP_RETR) == 0x2F) after every sending, and it's ok. Tried removing the auto_ack and flush_tx from send function and still no luck. Also, tried with one nRF on raspberry like on your other tutorial, and everything seems fine, but without any sign of successful transmission. Do you have any other idea what else should I check? I really think it's about code.