

Application note, Nordic nRF24L01 with Bascom-Avr

Getting a Nordic nRF24L01 single chip 2.4GHZ radio transceiver up and running with Bascom-Avr.

Written by Evert Dekker 2007.

What's an nRF24L01

The nRF24L01 is a single chip transceiver that's operate in the 2.4Ghz band with a maximum data rate of 2Mbps. It has also 6 so-called data pipes that allow to connect 6 devices together. In Enhanced shockburst mode the nRF24L01 take's care over the complete packet transmission, including the ACK and retry's. No need for Manchester coding etc.

What do we need for testing

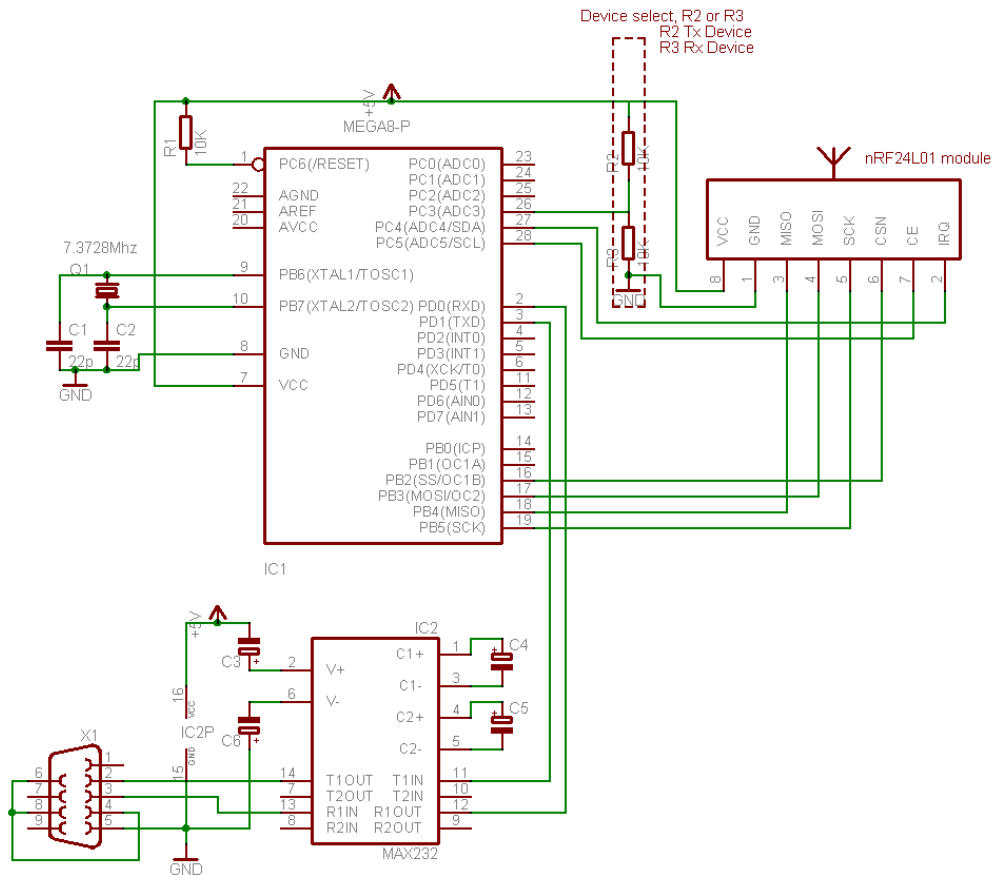
- First of all the datasheet;
http://www.nordicsemi.no/files/Product/data_sheet/Product_Specification_nRF24L01_1.0.pdf all significant information is there.
- To understand the working and functions of the chip you must read the 4 tutorials that Brennen Ball has written ; www.diyembedded.com His sample codes are written in C for the PIC processor, but the rest of the tutorials explains clearly how this chip works.
- We also need 2x the nRF24L01 chip. For most of us the chip is to small to solder, it's much easier when we buy some modules that are complete with the external components. I bought them from www.sparkfun.com . Search for *Transceiver MiRF nRF24L01* or *Transceiver Olimex nRF24L01*.



(Pics from Sparkfun.com)

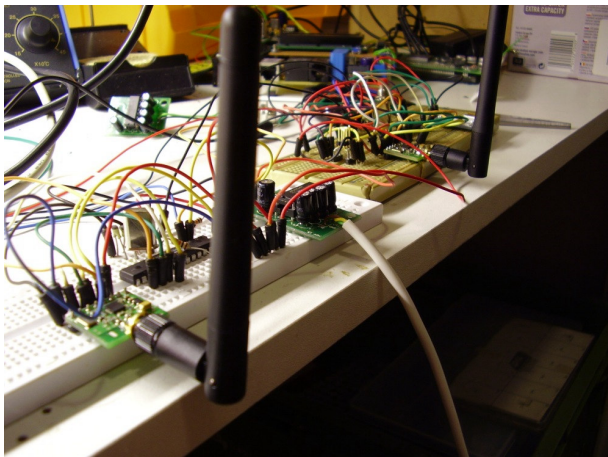
- Then we need 2 test boards with an AVR that's support hardware SPI, soft SPI is not working. The AVR needs for the test program at least 5K flash memory and Rs-232 connection with a pc.
- And of course we need Bascom-avr (tested with 1.11.8.3) This application note is too large to use with the demo version, so you need the paid version www.mcselec.com . If you strip the program or split up the TX and RX part it will maybe fit, but I didn't tried it.
It should also work with Bascom-8051 with some small modifications but that I did also not tried.

Test circuit



This is the circuit that I used for testing and where the code is adapted to.
The code is the same for the RX and TX device, with R2 or R3 you choose witch device it need to be.

Keep in mind that the nRF24L01 runs at 3.3V and that the i/o is 5V tolerant. The MiRF module from Sparkfun.com has it's own 3.3V regulator but the Olimex not.



Test circuit

The code

In the code is enough help to understand it, but here is some additional help.

You must use a chip with hardware spi, the software spi isn't working. We need the control over the SS (CSN) line our self to get the nRF24L01 working properly.

The nRF24L01 can work in different mode's. In this sample we choose to use Enhanced shockburst mode so we can demonstrate the benefits in comparing with "regular" transmitters.

In this sample we send a 5 bytes pload with auto ACK, 3x re-transmit, 2Mbps, 0dbm output trough pipe0 on channel 40.

The code is not optimised for speed so it will not reach the 2Mbps. For example, the Spi bus of the nRF24L01 can run max 8Mbps and we are using now 2Mbps, that's $7372800 / 4$ (Clock divided Spi Clockrate). Further there are some delay's that can be removed if your not using serial communication with a pc that's very slow in comparison with the air speed. Please read Brennen his tutorials how to calculate maximum air speed.

```
'-----
'   Nordic nRF24L01 data link demo in Enhanced Shockburst mode
'   By Evert Dekker 2007 nRF24L01@Evertdekker dotje com
'   Created with Bascom-Avr: 1.11.8.3
'-----

$regfile = "M8def.dat"
$crystal = 7372800
$baud = 19200
$hwstack = 40
$swstack = 20
$framesize = 40

'=== Declare sub routines
Declare Sub R_register(byval Command As Byte , Byval C_bytes As Byte)
Declare Sub W_register(byval C_bytes As Byte)

'=== Constante ===
'Define nRF24L01 interrupt flag's
Const Idle_int = &H00
Const Max_rt = &H10
Const Tx_ds = &H20
Const Rx_dr = &H40
'SPI(nRF24L01) commands
Const Read_reg = &H00
Const Write_reg = &H20
Const Rd_rx_pload = &H61
Const Wr_tx_pload = &HA0
Const Flush_tx = &HE1
Const Flush_rx = &HE2
Const Reuse_tx_pl = &HE3
Const Nop_comm = &HFF
'SPI(nRF24L01) registers(addresses)
Const Config_nrf = &H00
Const En_aa = &H01
Const En_rxaddr = &H02
Const Setup_aw = &H03
Const Setup_retr = &H04
Const Rf_ch = &H05

'Idle, no interrupt pending
'Max #of Tx Retrans Interrupt
'Tx Data Sent Interrupt
'Rx Data Received

'Define Read Command To Register
'Define Write Command To Register
'Define Rx Payload Register Address
'Define Tx Payload Register Address
'Define Flush Tx Register Command
'Define Flush Rx Register Command
'Define Reuse Tx Payload Register Command
'Define No Operation , Might Be Used To Read Status Register

'Config' register address
'Enable Auto Acknowledgment' register address
'Enabled RX addresses' register address
'Setup address width' register address
'Setup Auto. Retrans' register address
'RF channel' register address
```

```

Const Rf_setup = &H06
Const Status = &H07
Const Observe_tx = &H08
Const Cd = &H09
Const Rx_addr_p0 = &H0A
Const Rx_addr_p1 = &H0B
Const Rx_addr_p2 = &H0C
Const Rx_addr_p3 = &H0D
Const Rx_addr_p4 = &H0E
Const Rx_addr_p5 = &H0F
Const Tx_addr = &H10
Const Rx_pw_p0 = &H11
Const Rx_pw_p1 = &H12
Const Rx_pw_p2 = &H13
Const Rx_pw_p3 = &H14
Const Rx_pw_p4 = &H15
Const Rx_pw_p5 = &H16
Const Fifo_status = &H17
'Various
Const True = 1
Const False = 0

'=== Config hardware ===
Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes , Polarity = Low , Phase = 0 , Clockrate = 4 , Noss = 1
'Software SPI is NOT working with the nRF24L01, use hardware SPI only, but the SS pin must be controlled by our self
Config Pinc.5 = Output
Config Pinb.2 = Output
Config Pinc.4 = Input
Config Pinc.3 = Input
Ce Alias Portc.5
Ss Alias Portb.2
Irq Alias Pinc.4
Txrx_device Alias Pinc.3
Spiinit
Set Ce
Waitms 10
Reset Ce
Reset Ss
Dim D_bytes(33) As Byte , B_bytes(33) As Byte
Dim Temp As Byte , W As Word
Dim Packet_count As Byte

If Txrx_device = True Then Goto Main_tx
'===Main rx=====
Main_rx:
Call R_register(status , 1)
Print "Rx_device"
Reset Ce
Gosub Setup_rx
Waitms 2
Set Ce
Do
If Irq = 0 Then
Reset Ce
Do
Call R_register(rd_rx_pload , 5)
Print "Pload : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(fifo_status , 1)
Loop Until B_bytes(1).0 = True
D_bytes(1) = Write_reg + Status

'RF setup' register address
'Status' register address
'Observe TX' register address
'Carrier Detect' register address
'RX address pipe0' register address
'RX address pipe1' register address
'RX address pipe2' register address
'RX address pipe3' register address
'RX address pipe4' register address
'RX address pipe5' register address
'TX address' register address
'RX payload width, pipe0' register address
'RX payload width, pipe1' register address
'RX payload width, pipe2' register address
'RX payload width, pipe3' register address
'RX payload width, pipe4' register address
'RX payload width, pipe5' register address
'FIFO Status Register' register address

'init the spi pins

'Wait a moment until all hardware is stable
'Set CE pin low
'Set SS pin low (CSN pin)
'Dim the bytes use for SPI, D_bytes = outgoing B_bytes = Incoming

'Is this the RX or TX device?
'====
'Read STATUS register
'Send to terminal who i'm
'Set CE low to access the registers
'Setup the nRF24L01 for RX
'Add a delay before going in RX
'Set nRF20L01 in RX mode
'Main loop for RX
'Wait until IRQ occurs, pin becomes low on interrupt
'Receiver must be disabled before reading pload
'Loop until all 3 fifo buffers are empty
'Read 5 bytes RX pload register
'Read FIFO_STATUS
'Test or RX_EMPTY bit is true, RX FIFO empty
'Reset the RX_DR status bit
'Print the pload

```

```

    D_bytes(2) = &B01000000
    Call W_register(2)
    Set Ce
    Waitms 2
End If
'Gosub Dump_registers
Loop
Return

'====Main tx=====
Main_tx:
Print "TX_device"
D_bytes(1) = Flush_tx
Call W_register(1)
D_bytes(1) = Write_reg + Status
D_bytes(2) = &B00110000
Call W_register(2)
Do
Incr Packet_count
If Packet_count > 254 Then Packet_count = 0
Gosub Setup_tx
D_bytes(1) = Wr_tx_pload
D_bytes(2) = &HAA
D_bytes(3) = &HBB
D_bytes(4) = &HCC
D_bytes(5) = &H11
D_bytes(6) = Packet_count
Call W_register(6)
Waitms 2
Set Ce
Waitms 1
Reset Ce
Waitms 100
W = 0
Do
If Irq = 0 Then
    Call R_register(status, 1)
    Temp = B_bytes(1) And &B01110000
    Select Case Temp
        Case Max_rt
            Print "Maximum number of TX retries, Flushing the TX buffer now !"
            D_bytes(1) = Flush_tx
            Call W_register(1)
            D_bytes(1) = Write_reg + Status
            D_bytes(2) = &B00010000
            Call W_register(2)
            Exit Do
        Case Tx_ds
            Print "Packet " ; Packet_count ; " send and ACK received."
            D_bytes(1) = Write_reg + Status
            D_bytes(2) = &B00100000
            Call W_register(2)
            Exit Do
        Case Else
            Print "Other irq " ; Bin(temp)
            D_bytes(1) = Flush_tx
            Call W_register(1)
            D_bytes(1) = Write_reg + Status
            D_bytes(2) = &B00110000
            Call W_register(2)
    End Select
End If
'Write 1 to RX_DR bit to reset IRQ
'Enable receiver again
'Unremark me for debugging
'Send to terminal who i'm
'Flush the TX_fifo buffer
'Reset the IRQ bits
'Main loop for TX
'Increase the send packet counter, for test only
'Setup the nrf240101 for TX
'Put 5 bytes in the TX pload buffer
'Byte 1
'Byte 2
'Byte 3
'Byte 4
'Byte 5 will be increase every loop
'Write 6 bytes to register
'Set CE for a short moment to transmit the fifo buffer
'
'Some delay to read the output on the terminal, line can be removed for max. speed
'Counter for time out
'Mask the IRQ bits out the status byte
'Which IRQ occurs
'MAX_RT
'Flush the TX buffer
'Clear the MAX_RT IRQ bit
'TX_DS
'Clear the TX_DS IRQ bit
'Other IRQ ??
'Flush the TX buffer
'Clear both MAX_RT, TX_DS bits

```

```

        End Select
    End If
    Waitms 1
    Incr W
    If W > 100 Then
        Print "No irq response from RF20L01 within 100ms"
        Exit Do
    End If
Loop
Loop
Return

'=== Sub routines ===
Sub W_register(byval C_bytes As Byte)
Reset Ss
    Spiout D_bytes(1) , C_bytes
Set Ss
End Sub

Sub R_register(byval Command As Byte , Byval C_bytes As Byte) As Byte
Reset Ss
    Spiout Command , 1
    Spiin B_bytes(1) , C_bytes
Set Ss
End Sub

Setup_rx:
D_bytes(1) = Write_reg + Rx_addr_p0
D_bytes(2) = &H34
D_bytes(3) = &H43
D_bytes(4) = &H10
D_bytes(5) = &H10
D_bytes(6) = &H01
Call W_register(6)
D_bytes(1) = Write_reg + En_aa
D_bytes(2) = &H01
Call W_register(2)
D_bytes(1) = Write_reg + En_rxaddr
D_bytes(2) = &H01
Call W_register(2)
D_bytes(1) = Write_reg + Rf_ch
D_bytes(2) = 40
Call W_register(2)
D_bytes(1) = Write_reg + Rx_pw_p0
D_bytes(2) = 5
Call W_register(2)
D_bytes(1) = Write_reg + Rf_setup
D_bytes(2) = &H0F
Call W_register(2)
D_bytes(1) = Write_reg + Config_nrf
D_bytes(2) = &H0F
Call W_register(2)
Return

Setup_tx:
D_bytes(1) = Write_reg + Tx_addr
D_bytes(2) = &H34
D_bytes(3) = &H43

'Time out waiting for IRQ lms * 100
'Increment W
'Waited for 100ms
'Exit the wait loop

'Write register with SPI
'Manual control SS pin, set SS low before shifting out the bytes
'Shiftout the data bytes trough SPI , C_bytes is the amount bytes to be written
'Set SS high

'C_bytes = Count_bytes, number off bytes to be read
'Manual controle SS pin, set low before shifting in/out the bytes
'First shiftout the register to be read
'Read back the bytes from SPI send by nRF20L01
'Set SS back to high level

'Setup for RX
'RX address for pipe0

'Send 6 bytes to SPI
'Enable auto ACK for pipe0

'Enable RX address for pipe0

'Set RF channel

'Set RX pload width for pipe0

'Setup RF-> Output power 0dbm, datarate 2Mbps and LNA gain on

'Setup CONFIG-> PRX=1(RX_device), PWR_UP=1, CRC 2bytes, Enable CRC

'Setup for TX
'TX address

```

```

D_bytes(4) = &H10
D_bytes(5) = &H10
D_bytes(6) = &H01
Call W_register(6)
D_bytes(1) = Write_reg + Rx_addr_p0          'RX address for pipe0
D_bytes(2) = &H34
D_bytes(3) = &H43
D_bytes(4) = &H10
D_bytes(5) = &H10
D_bytes(6) = &H01
Call W_register(6)
D_bytes(1) = Write_reg + En_aa                'Enable auto ACK for pipe0
D_bytes(2) = &H01
Call W_register(2)
D_bytes(1) = Write_reg + En_rxaddr            'Enable RX address for pipe0
D_bytes(2) = &H01
Call W_register(2)
D_bytes(1) = Write_reg + Rf_ch                'Set RF channel
D_bytes(2) = 40
Call W_register(2)
D_bytes(1) = Write_reg + Rf_setup             'Setup RF-> Output power 0dbm, datarate 2Mbps and LNA gain on
D_bytes(2) = &H0F
Call W_register(2)
D_bytes(1) = Write_reg + Config_nrf           'Setup CONFIG-> PRX=0(TX_device), PWR_UP=1, CRC 2bytes, Enable CRC
D_bytes(2) = &H0E
Call W_register(2)
Return

Dump_registers:                               'Dumps all nRF24L01 registers to the terminal, handy for debugging
Print "* Dump nRF24L01 Registers *"
Call R_register(config_nrf , 1)
Print "CONFIG      : " ; Bin(b_bytes(1))
Call R_register(en_aa , 1)
Print "EN_AA       : " ; Bin(b_bytes(1))
Call R_register(en_rxaddr , 1)
Print "EN_RXADDR    : " ; Bin(b_bytes(1))
Call R_register(setup_aw , 1)
Print "SETUP_AW     : " ; Bin(b_bytes(1))
Call R_register(setup_retr , 1)
Print "SETUP_RETR   : " ; Bin(b_bytes(1))
Call R_register(rf_ch , 1)
Print "RF_CH       : " ; B_bytes(1)
Call R_register(rf_setup , 1)
Print "RF_SETUP    : " ; Bin(b_bytes(1))
Call R_register(status , 1)
Print "STATUS      : " ; Bin(b_bytes(1))
Call R_register(observe_tx , 1)
Print "OBSERVE_TX   : " ; Bin(b_bytes(1))
Call R_register(cd , 1)
Print "CD          : " ; Bin(b_bytes(1))
Call R_register(rx_addr_p0 , 5)
Print "RX_ADDR_P0   : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_addr_p1 , 5)
Print "RX_ADDR_P1   : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_addr_p2 , 5)
Print "RX_ADDR_P2   : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_addr_p3 , 5)
Print "RX_ADDR_P3   : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_addr_p4 , 5)
Print "RX_ADDR_P4   : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))

```

```
Call R_register(rx_addr_p5 , 5)
Print "RX_ADDR_P5 : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(tx_addr , 5)
Print "TX_ADDR : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_pw_p0 , 5)
Print "RX_PW_P0 : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_pw_p1 , 5)
Print "RX_PW_P1 : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_pw_p2 , 5)
Print "RX_PW_P2 : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_pw_p3 , 5)
Print "RX_PW_P3 : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_pw_p4 , 5)
Print "RX_PW_P4 : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(rx_pw_p5 , 5)
Print "RX_PW_P5 : " ; Hex(b_bytes(1)) ; Hex(b_bytes(2)) ; Hex(b_bytes(3)) ; Hex(b_bytes(4)) ; Hex(b_bytes(5))
Call R_register(fifo_status , 1)
Print "FIFO_STATUS : " ; Bin(b_bytes(1))
Return
```