```c

#define UCPHA0 1
#define F_CPU 8000000UL
#define BAUD_RATE 9600UL
#define UBRR_VALUE ((F_CPU)/(2UL*BAUD_RATE))-1

#include "nrf24.h"
#include <avr/io.h>

uint8_t payload_len;

void nrf24_init()
{
    nrf24_setupPins();
    nrf24_ce_digitalWrite(LOW);
    nrf24_csn_digitalWrite(HIGH);
}

void nrf24_config(uint8_t channel, uint8_t pay_length)
{
    /* Use static payload length ... */
    payload_len = pay_length;

    // Set RF channel
    nrf24_configRegister(RF_CH,channel);

    // Set length of incoming payload
    nrf24_configRegister(RX_PW_P0, 0x00); // Auto-ACK pipe ...
    nrf24_configRegister(RX_PW_P1, payload_len); // Data payload pipe
    nrf24_configRegister(RX_PW_P2, 0x00); // Pipe not used
    nrf24_configRegister(RX_PW_P3, 0x00); // Pipe not used
    nrf24_configRegister(RX_PW_P4, 0x00); // Pipe not used
    nrf24_configRegister(RX_PW_P5, 0x00); // Pipe not used

    // 1 Mbps, TX gain: 0dbm
    nrf24_configRegister(RF_SETUP, (0<<RF_DR)|((0x03)<<RF_PWR));

    // CRC enable, 1 byte CRC length
    nrf24_configRegister(CONFIG,nrf24_CONFIG);

    // Auto Acknowledgment
    nrf24_configRegister(EN_AA,(1<<ENAA_P0)|(1<<ENAA_P1)|(0<<ENAA_P2)|
      (0<<ENAA_P3)|(0<<ENAA_P4)|(0<<ENAA_P5));

    // Enable RX addresses
    nrf24_configRegister(EN_RXADDR,(1<<ERX_P0)|(1<<ERX_P1)|(0<<ERX_P2)|
      (0<<ERX_P3)|(0<<ERX_P4)|(0<<ERX_P5));

    // Auto retransmit delay: 1000 us and Up to 15 retransmit trials
    nrf24_configRegister(SETUP_RETR,(0x04<<ARD)|(0x0F<<ARC));

    // Dynamic length configurations: No dynamic length
    nrf24_configRegister(DYNPD,(0<<DPL_P0)|(0<<DPL_P1)|(0<<DPL_P2)|(0<<DPL_P3)|
      (0<<DPL_P4)|(0<<DPL_P5));

    // Start listening
    nrf24_powerUpRx();
}

/* Set the RX address */
void nrf24_rx_address(uint8_t * adr)
{
    nrf24_ce_digitalWrite(LOW);
    nrf24_writeRegister(RX_ADDR_P1,adr,nrf24_ADDR_LEN);
    nrf24_ce_digitalWrite(HIGH);
}

/* Returns the payload length */
uint8_t nrf24_payload_length()
{
    return payload_len;
}

/* Set the TX address */
void nrf24_tx_address(uint8_t* adr)
{
    /* RX_ADDR_P0 must be set to the sending addr for auto ack to work. */
    nrf24_writeRegister(RX_ADDR_P0,adr,nrf24_ADDR_LEN);
    nrf24_writeRegister(TX_ADDR,adr,nrf24_ADDR_LEN);
}

/* Checks if data is available for reading */
/* Returns 1 if data is ready ... */
uint8_t nrf24_dataReady()
{
    // See note in getData() function - just checking RX_DR isn't good enough
    uint8_t status = nrf24_getStatus();

    // We can short circuit on RX_DR, but if it's not set, we still need
    // to check the FIFO for any pending packets
    if ( status & (1 << RX_DR) )
    {
        return 1;
    }

    return !nrf24_rxFifoEmpty();;
}

/* Checks if receive FIFO is empty or not */
uint8_t nrf24_rxFifoEmpty()
{
    uint8_t fifoStatus;

    nrf24_readRegister(FIFO_STATUS,&fifoStatus,1);
```

```c

    return (fifoStatus & (1 << RX_EMPTY));
}

/* Returns the length of data waiting in the RX fifo */
uint8_t nrf24_payloadLength()
{
    uint8_t status;
    nrf24_csn_digitalWrite(LOW);
    spi_transfer(R_RX_PL_WID);
    status = spi_transfer(0x00);
    nrf24_csn_digitalWrite(HIGH);
    return status;
}

/* Reads payload bytes into data array */
void nrf24_getData(uint8_t* data)
{
    /* Pull down chip select */
    nrf24_csn_digitalWrite(LOW);

    /* Send cmd to read rx payload */
    spi_transfer( R_RX_PAYLOAD );

    /* Read payload */
    nrf24_transferSync(data,data,payload_len);

    /* Pull up chip select */
    nrf24_csn_digitalWrite(HIGH);

    /* Reset status register */
    nrf24_configRegister(STATUS,(1<<RX_DR));
}

/* Returns the number of retransmissions occured for the last message */
uint8_t nrf24_retransmissionCount()
{
    uint8_t rv;
    nrf24_readRegister(OBSERVE_TX,&rv,1);
    rv = rv & 0x0F;
    return rv;
}

// Sends a data package to the default address. Be sure to send the correct
// amount of bytes as configured as payload on the receiver.
void nrf24_send(uint8_t* value)
{
    /* Go to Standby-I first */
    nrf24_ce_digitalWrite(LOW);

    /* Set to transmitter mode , Power up if needed */
    nrf24_powerUpTx();

    /* Do we really need to flush TX fifo each time ? */
    #if 1
        /* Pull down chip select */
        nrf24_csn_digitalWrite(LOW);

        /* Write cmd to flush transmit FIFO */
        spi_transfer(FLUSH_TX);

        /* Pull up chip select */
        nrf24_csn_digitalWrite(HIGH);
    #endif

    /* Pull down chip select */
    nrf24_csn_digitalWrite(LOW);

    /* Write cmd to write payload */
    spi_transfer(W_TX_PAYLOAD);

    /* Write payload */
    nrf24_transmitSync(value,payload_len);

    /* Pull up chip select */
    nrf24_csn_digitalWrite(HIGH);

    /* Start the transmission */
    nrf24_ce_digitalWrite(HIGH);
}

uint8_t nrf24_isSending()
{
    uint8_t status;

    /* read the current status */
    status = nrf24_getStatus();

    /* if sending successful (TX_DS) or max retries exceded (MAX_RT). */
    if((status & ((1 << TX_DS)  | (1 << MAX_RT))))
    {
        return 0; /* false */
    }

    return 1; /* true */

}

uint8_t nrf24_getStatus()
{
    uint8_t rv;
    nrf24_csn_digitalWrite(LOW);
    rv = spi_transfer(NOP);
    nrf24_csn_digitalWrite(HIGH);
    return rv;
}

uint8_t nrf24_lastMessageStatus()
{
    uint8_t rv;

    rv = nrf24_getStatus();

    /* Transmission went OK */
    if((rv & ((1 << TX_DS))))
    {
        return NRF24_TRANSMISSON_OK;
    }
    /* Maximum retransmission count is reached */
    /* Last message probably went missing ... */
    else if((rv & ((1 << MAX_RT))))
    {
        return NRF24_MESSAGE_LOST;
    }
    /* Probably still sending ... */
    else
    {
        return 0xFF;
    }
}

void nrf24_powerUpRx()
{
    nrf24_csn_digitalWrite(LOW);
    spi_transfer(FLUSH_RX);
    nrf24_csn_digitalWrite(HIGH);

    nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));

    nrf24_ce_digitalWrite(LOW);
    nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(1<<PRIM_RX)));
    nrf24_ce_digitalWrite(HIGH);
}

void nrf24_powerUpTx()
{
    nrf24_configRegister(STATUS,(1<<RX_DR)|(1<<TX_DS)|(1<<MAX_RT));

    nrf24_configRegister(CONFIG,nrf24_CONFIG|((1<<PWR_UP)|(0<<PRIM_RX)));
}

void nrf24_powerDown()
{
    nrf24_ce_digitalWrite(LOW);
    nrf24_configRegister(CONFIG,nrf24_CONFIG);
}
```

```c

uint8_t spi_transfer(uint8_t tx)
{
    uint8_t i = 0;
    uint8_t rx = 0;

    nrf24_sck_digitalWrite(LOW);

    for(i=0;i<8;i++)
    {

        if(tx & (1<<(7-i)))
        {
            nrf24_mosi_digitalWrite(HIGH);
        }
        else
        {
            nrf24_mosi_digitalWrite(LOW);
        }

        nrf24_sck_digitalWrite(HIGH);

        rx = rx << 1;
        if(nrf24_miso_digitalRead())
        {
            rx |= 0x01;
        }

        nrf24_sck_digitalWrite(LOW);

    }

    return rx;
}

/* send and receive multiple bytes over SPI */
void nrf24_transferSync(uint8_t* dataout,uint8_t* datain,uint8_t len)
{
    uint8_t i;

    for(i=0;i<len;i++)
    {
        datain[i] = spi_transfer(dataout[i]);
    }

}

/* send multiple bytes over SPI */
void nrf24_transmitSync(uint8_t* dataout,uint8_t len)
{
    uint8_t i;

    for(i=0;i<len;i++)
    {
        spi_transfer(dataout[i]);
    }

}

/* Clocks only one byte into the given nrf24 register */
void nrf24_configRegister(uint8_t reg, uint8_t value)
{
    nrf24_csn_digitalWrite(LOW);
    spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
    spi_transfer(value);
    nrf24_csn_digitalWrite(HIGH);
}

/* Read single register from nrf24 */
void nrf24_readRegister(uint8_t reg, uint8_t* value, uint8_t len)
{
    nrf24_csn_digitalWrite(LOW);
    spi_transfer(R_REGISTER | (REGISTER_MASK & reg));
    nrf24_transferSync(value,value,len);
    nrf24_csn_digitalWrite(HIGH);
}

/* Write to a single register of nrf24 */
void nrf24_writeRegister(uint8_t reg, uint8_t* value, uint8_t len)
{
    nrf24_csn_digitalWrite(LOW);
    spi_transfer(W_REGISTER | (REGISTER_MASK & reg));
    nrf24_transmitSync(value,len);
    nrf24_csn_digitalWrite(HIGH);
}

#define RF_DDR  DDRC
#define RF_PORT PORTC
#define RF_PIN  PINC

#define set_bit(reg,bit) reg |= (1<<bit)
#define clr_bit(reg,bit) reg &= ~(1<<bit)
#define check_bit(reg,bit) (reg&(1<<bit))

/* ------------------------------------------------------------------------- */

void nrf24_setupPins()
{
    set_bit(RF_DDR,0); // CE output
    set_bit(RF_DDR,1); // CSN output
    set_bit(RF_DDR,2); // SCK output
    set_bit(RF_DDR,3); // MOSI output
    clr_bit(RF_DDR,4); // MISO input
}
```

```c
/* ------------------------------------------------------------------------ */
void nrf24_ce_digitalWrite(uint8_t state)
{
    if(state)
    {
        set_bit(RF_PORT,0);
    }
    else
    {
        clr_bit(RF_PORT,0);
    }
}
/* ------------------------------------------------------------------------ */
void nrf24_csn_digitalWrite(uint8_t state)
{
    if(state)
    {
        set_bit(RF_PORT,1);
    }
    else
    {
        clr_bit(RF_PORT,1);
    }
}
/* ------------------------------------------------------------------------ */
void nrf24_sck_digitalWrite(uint8_t state)
{
    if(state)
    {
        set_bit(RF_PORT,2);
    }
    else
    {
        clr_bit(RF_PORT,2);
    }
}
/* ------------------------------------------------------------------------ */
void nrf24_mosi_digitalWrite(uint8_t state)
{
    if(state)
    {
        set_bit(RF_PORT,3);
    }
    else
    {
        clr_bit(RF_PORT,3);
    }
}
/* ------------------------------------------------------------------------ */
uint8_t nrf24_miso_digitalRead()
{
    return check_bit(RF_PIN,4);
}
/* ---------------------------------------------------------------------- */

```