```
/*************************************/
/**** NRF Send and Receive Functions ****/
/*************************************/

#include <util/delay.h>

#include "NRF24L01_Dec.h"
#include "Converter.h"

extern volatile uint8_t RFChannel;
extern volatile unsigned char GateAddress[4];

void SendBack(struct uint256_t Payload, uint8_t Pipe)
{
    uint32_t Address = 0x00000000;

    switch(Pipe)
    {
        case 0x01 :
            Address = NRF_ReadRXAddress(RX_ADDR_P1);
            break;
        case 0x02 :
            Address = NRF_ReadRXAddress(RX_ADDR_P2);
            break;
        case 0x03 :
            Address = NRF_ReadRXAddress(RX_ADDR_P3);
            break;
        case 0x04 :
            Address = NRF_ReadRXAddress(RX_ADDR_P4);
            break;
    }

    NRF_CE_Lo;
    NRF_ChangeTXAddress(Address);
    NRF_ChangeRXAddress(Address, RX_ADDR_P0);
    NRF_SetTXState();
    NRF_FlushTXBuffer();
    NRF_WriteTXPayload(Payload);

    NRF_CE_Hi;
     delay ms(1);
    NRF_CE_Lo;

    NRF_SendImpuls();

    while(NRF_IRQ_PIN & (1<<NRF_IRQ));

    uint8_t status_reg;
    status_reg = NRF_ReadRegister(STATUS);

    if((status_reg & RX_DR) == RX_DR)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    if((status_reg & TX_DS) == TX_DS)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    if((status_reg & MAX_RT) == MAX_RT)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
```

```
    }

    NRF_CE_Lo;
    NRF_SetRXState();
    NRF_CE_Hi;
}

void SendToGate(struct uint256_t Payload)
{
    Addressconvert.c[0] = GateAddress[0];
    Addressconvert.c[1] = GateAddress[1];
    Addressconvert.c[2] = GateAddress[2];
    Addressconvert.c[3] = GateAddress[3];

    NRF_CE_Lo;
    NRF_ChangeTXAddress(Addressconvert.u32i);
    NRF_ChangeRXAddress(Addressconvert.u32i, RX_ADDR_P0);
    NRF_SetTXState();
    NRF_FlushTXBuffer();
    NRF_WriteTXPayload(Payload);

    NRF_CE_Hi;
    _delay_ms(1);
    NRF_CE_Lo;

    NRF_SendImpuls();

    while(NRF_IRQ_PIN & (1<<NRF_IRQ));

    uint8_t status_reg;
    status_reg = NRF_ReadRegister(STATUS);

    if((status_reg & RX_DR) == RX_DR)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    if((status_reg & TX_DS) == TX_DS)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    if((status_reg & MAX_RT) == MAX_RT)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    NRF_CE_Lo;
    NRF_SetRXState();
    NRF_CE_Hi;
}

void SendtoLights(struct uint256_t Payload, uint32_t Address)
{
    NRF_CE_Lo;
    NRF_ChangeTXAddress(Address);
    NRF_ChangeRXAddress(Address, RX_ADDR_P0);
    NRF_SetTXState();
    NRF_FlushTXBuffer();
    NRF_WriteTXPayload(Payload);

    NRF_CE_Hi;
    _delay_ms(1);
    NRF_CE_Lo;
```

```
    NRF_SendImpuls();

    while(NRF_IRQ_PIN & (1<<NRF_IRQ));

    uint8_t status_reg;
    status_reg = NRF_ReadRegister(STATUS);

    if((status_reg & RX_DR) == RX_DR)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    if((status_reg & TX_DS) == TX_DS)
    {
        NRF WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    if((status_reg & MAX_RT) == MAX_RT)
    {
        NRF_WriteRegister(0xff, STATUS);
        NRF_FlushTXBuffer();
    }
    NRF_CE_Lo;
    NRF_SetRXState();
    NRF_CE_Hi;
}
```

```
/*********************************/
/********* NRF 24L01 *************/
/**** with Software SPI **********/
/*********************************/

#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "NRF24L01_Dec.h"
#include "SoftwareSPI_Dec.h"
#include "Converter.h"

extern unsigned char Payload[32];
extern unsigned char Received[3][32];
extern unsigned char RFChannel;
extern volatile uint8_t WorkerFlag;
extern volatile uint8_t MFlag;
extern volatile uint8_t RT;

/****** NRF Init *****************/
/*
void NRF24L01_Init(void)
{
    NRF_CE_Lo;

    NRF_CE_DDR |= (1<<NRF_CE);

//  NRF_IRQ_DDR &= ~(1<<NRF_IRQ);
//  NRF_IRQ_PORT |= (1<<NRF_IRQ);

    NRF_WriteRegister(0x0E, CONFIG);
    NRF_WriteRegister(0x1F, EN_AA);
    NRF_WriteRegister(0x1F, EN_RXADDR);
    NRF_WriteRegister(0x02, SETUP_AW);
    NRF_WriteRegister(0x04, SETUP_RETR);
    NRF_WriteRegister(0x04, RF_CH);
    NRF_WriteRegister(0x0E, RF_SETUP);
    NRF_WriteRegister(0x20, RX_PW_P0);
    NRF_WriteRegister(0x20, RX_PW_P1);

//  NRF_INT_SENS;

    NRF_CE_Lo;
}
*/
void NRF24L01_CustomInit(uint8_t RFCH, uint8_t SendRetry, uint8_t RFPWR, unsigned char Rel
ayAddress[4][4])
{
    NRF_CE_Lo;

    NRF_CE_DDR |= (1<<NRF_CE);

//  NRF_IRQ_DDR &= ~(1<<NRF_IRQ);
//  NRF_IRQ_PORT |= (1<<NRF_IRQ);

    NRF_WriteRegister(0x0F, CONFIG);
    NRF_WriteRegister(0x1F, EN_AA);
    NRF_WriteRegister(0x1F, EN_RXADDR);
    NRF_WriteRegister(0x02, SETUP_AW);
    NRF_WriteRegister(0x04, SETUP_RETR);
    NRF_WriteRegister(0x0e, RF_SETUP);
    NRF_WriteRegister(RFCH, RF_CH);
    NRF_WriteRegister(0x20, RX_PW_P0);
```

```
    NRF_WriteRegister(0x20, RX_PW_P1);
    NRF_WriteRegister(0x20, RX_PW_P2);
    NRF_WriteRegister(0x20, RX_PW_P3);
    NRF_WriteRegister(0x20, RX_PW_P4);

    for(int i=0; i<4; i++)
    {
        Addressconvert.c[i] = RelayAddress[0][i];
    }
    NRF_ChangeRXAddress(Addressconvert.u32i, RX_ADDR_P1);

    for(int i=0; i<4; i++)
    {
        Addressconvert.c[i] = RelayAddress[1][i];
    }
    NRF_ChangeRXAddress(Addressconvert.u32i, RX_ADDR_P2);

    for(int i=0; i<4; i++)
    {
        Addressconvert.c[i] = RelayAddress[2][i];
    }
    NRF_ChangeRXAddress(Addressconvert.u32i, RX_ADDR_P3);

    for(int i=0; i<4; i++)
    {
        Addressconvert.c[i] = RelayAddress[3][i];
    }
    NRF_ChangeRXAddress(Addressconvert.u32i, RX_ADDR_P4);

    NRF_CE_Lo;
}

/****** Write Register ***********/

void NRF_WriteRegister(uint8_t u8Register, uint8_t u8Cmd)
{
    uint8_t u8Command = W_REGISTER | u8Cmd;

    SCSN_Lo;

    SSPI Write byte(u8Command);
    SSPI_Write_byte(u8Register);

    SCSN_Hi;
}

/****** Read Register ************/

uint8_t NRF_ReadRegister(uint8_t u8Cmd)
{
    uint8_t u8Command = R_REGISTER | u8Cmd;
    uint8_t u8Content = 0x00;

    SCSN_Lo;

    SSPI_Read_byte(u8Command);
    u8Content = SSPI_Read_byte(0x00);

    SCSN_Hi;
    return u8Content;
}

/****** Read RX Address ***********/
```

```
uint32_t NRF_ReadRXAddress(uint8_t u8Pipe)
{
    uint8_t u8PipeReg = 0x00;
    uint8_t u8i = 0x00;

    switch (u8Pipe)
    {
        case 0x0A:
            u8PipeReg = RX_ADDR_P0;
            break;
        case 0x0B:
            u8PipeReg = RX_ADDR_P1;
            break;
        case 0x0C:
            u8PipeReg = RX_ADDR_P2;
            break;
        case 0x0D:
            u8PipeReg = RX_ADDR_P3;
            break;
        case 0x0E:
            u8PipeReg = RX_ADDR_P4;
            break;
        case 0x0F:
            u8PipeReg = RX_ADDR_P5;
            break;
    }
    uint8_t u8Command = R_REGISTER | u8PipeReg;
    SCSN_Lo;

    switch(u8Pipe)
    {
        case 0x0A:
        case 0x0B:

            SSPI_Write_byte(u8Command);
            for(u8i = 4; u8i>0; u8i--)
            {
                Addressconvert.c[u8i-1] = SSPI_Read_byte(NO_OP);
            }
            break;

        case 0x0C:
        case 0x0D:
        case 0x0E:
        case 0x0F:
            SSPI_Write_byte(R_REGISTER | RX_ADDR_P1);
            for(u8i = 4; u8i>0; u8i--)
            {
                Addressconvert.c[u8i-1] = SSPI_Read_byte(NO_OP);
            }
            SCSN_Hi;
            _delay_ms(1);
            SCSN_Lo;
            SSPI_Write_byte(u8Command);
            Addressconvert.c[3] = SSPI_Read_byte(NO_OP);
            break;
    }
    SCSN_Hi;
    return Addressconvert.u32i;
}


/****** Read TX Address ***********/
```

```
uint32_t NRF_ReadTXAddress()
{
    uint8_t u8i = 0x00;

    uint8_t u8Command = R_REGISTER | TX_ADDR;
    SCSN_Lo;
    SSPI_Write_byte(u8Command);
    for(u8i = 4; u8i>0; u8i--)
    {
        Addressconvert.c[u8i-1] = SSPI_Read_byte(NO_OP);
    }
    SCSN_Hi;
    return Addressconvert.u32i;
}

/****** Change RX Address ***********/

void NRF_ChangeRXAddress(uint32_t u32Address, uint8_t u8Pipe)
{
    uint8_t u8PipeReg = 0x00;
    uint8_t u8CEFlag = 0x00;
    uint8_t u8i = 0x00;
    Addressconvert.u32i = u32Address;

    if(PORTB & NRF_CE)
    {
        NRF_CE_Lo;
        u8CEFlag = 0x11;
    }
    _delay_us(250);
    switch (u8Pipe)
    {
        case 0x0A:
            u8PipeReg = RX_ADDR_P0;
            break;
        case 0x0B:
            u8PipeReg = RX_ADDR_P1;
            break;
        case 0x0C:
            u8PipeReg = RX_ADDR_P2;
            break;
        case 0x0D:
            u8PipeReg = RX_ADDR_P3;
            break;
        case 0x0E:
            u8PipeReg = RX_ADDR_P4;
            break;
        case 0x0F:
            u8PipeReg = RX_ADDR_P5;
            break;
    }
    uint8_t u8Command = W_REGISTER | u8PipeReg;
    SCSN_Lo;
    SSPI_Write_byte(u8Command);
    switch(u8Pipe)
    {
        case 0x0A:
        case 0x0B:
            for(u8i = 4; u8i>0; u8i--)
            {
                SSPI_Write_byte(Addressconvert.c[u8i-1]);
            }
            break;
```

```
        case 0x0C:
        case 0x0D:
        case 0x0E:
        case 0x0F:
            SSPI_Write_byte(Addressconvert.c[3]);
            break;
    }
    SCSN_Hi;

    _delay_us(10);

    if(u8CEFlag == 0x11)
    {
        NRF_CE_Hi;
        u8CEFlag = 0x00;
    }
}

/****** Change TX Address ***********/

void NRF_ChangeTXAddress(uint32_t u32Address)
{
    uint8_t u8i = 0x00;
    uint8_t u8CEFlag = 0x00;
    Addressconvert.u32i = u32Address;

    if(NRF_CE_PORT & NRF_CE)
    {
        NRF_CE_Lo;
        u8CEFlag = 0x11;
    }
    _delay_us(250);
    uint8_t u8Command = W_REGISTER | TX_ADDR;
    SCSN_Lo;
    SSPI_Write_byte(u8Command);
    for(u8i = 4; u8i>0; u8i--)
    {
        SSPI_Write_byte(Addressconvert.c[u8i-1]);
    }
    SCSN_Hi;
    _delay_us(10);

    if(u8CEFlag == 0x11)
    {
        NRF_CE_Hi;
        u8CEFlag = 0x00;
    }
}

/****** Write TX Payload ************/

void NRF_WriteTXPayload(struct uint256_t u256Payload)
{
    uint8_t u8i;
    Payloadconvert.u256i = u256Payload;
    SCSN_Lo;
    uint8_t u8Reg = NRF_ReadRegister(CONFIG);
    SCSN_Hi;

    if(!(u8Reg & 0x01))
    {
        SCSN_Lo;
        SSPI_Write_byte(W_TX_PAYLOAD);
        for(u8i=32 ; u8i>0; u8i--)
```

```
        {
            SSPI_Write_byte(Payloadconvert.c[u8i-1]);
        }
        SCSN_Hi;
    }
}

/****** Read RX Payload ************/

struct uint256_t NRF_ReadRXPayload()
{
    uint8_t u8i = 0x00;

    SCSN_Lo;
    uint8_t u8Reg = NRF_ReadRegister(CONFIG);
    SCSN_Hi;

    if((u8Reg & 0x01))
    {
        SCSN_Lo;
        SSPI_Write_byte(R_RX_PAYLOAD);
        for(u8i=32; u8i>0; u8i--)
        {
            Payloadconvert.c[u8i-1] = SSPI_Read_byte(NO_OP);
        }
        SCSN_Hi;
    }
    else
    {
        for(u8i=0; u8i<32; u8i++)
        {
            Payloadconvert.c[u8i] = 0xff;
        }
    }
    return Payloadconvert.u256i;
}

/****** Flush RX Buffer **********/

void NRF_FlushRXBuffer(void)
{
    SCSN_Lo;
    SSPI_Write_byte(FLUSH_RX);
    SCSN_Hi;
}

/****** Flush TX Buffer **********/

void NRF_FlushTXBuffer(void)
{
    SCSN_Lo;
    SSPI_Write_byte(FLUSH_TX);
    SCSN_Hi;
}

/****** Reuse TX Buffer **********/

void NRF_ReuseTXBuffer(void)
{
    SCSN Lo;
    SSPI_Write_byte(REUSE_TX_PL);
    SCSN_Hi;
}
```

```
/****** Set RX State *************/

void NRF_SetRXState()
{
    SCSN_Lo;
    uint8_t oldConfig = NRF_ReadRegister(CONFIG);

    uint8_t newConfig = oldConfig | 0x01;

    NRF_WriteRegister(newConfig, CONFIG);
    NRF_WriteRegister(RFChannel, RF_CH);
    SCSN_Hi;
}

/****** Set TX State *************/

void NRF_SetTXState()
{
    SCSN_Lo;
    uint8_t oldConfig = NRF_ReadRegister(CONFIG);

    uint8_t newConfig = oldConfig & ~(0x01);

    NRF_WriteRegister(newConfig, CONFIG);
    NRF_WriteRegister(RFChannel, RF_CH);
    SCSN_Hi;
}

/****** Send Impuls **************/

void NRF_SendImpuls()
{
    NRF_CE_Hi;
    _delay_us(20);
    NRF_CE_Lo;
}

/****** Interrupt ***************/
/*
ISR(INT0_vect)
{
    uint8_t tmp_sreg;
    uint8_t status_reg;
    uint8_t i = 0x00;

    NRF_INT_FLAG_CL;            // Clears the Interruptflag

    tmp_sreg = SREG;            // Safe SREG

    cli();                      // Disable the global Interrupt

    status_reg = NRF_ReadRegister(STATUS);

    NRF_CE_Lo;

    switch(status_reg & 0x70)
    {
        case RX_DR :
            NRF_WriteRegister(0xff, STATUS);            // clear Flags
            switch(status_reg & 0x0E)
            {
                case 0x00 :
                    Payloadconvert.u256i = NRF_ReadRXPayload();
```

```
                    for(i =0; i<32; i++)
                    {
                        Payload[i] = Payloadconvert.c[i];
                    }

                    WorkerFlag = 0x04;
                    break;

                default :
                    Payloadconvert.u256i = NRF_ReadRXPayload();
                    for( i=0; i<32; i++)
                    {
                        Received[MFlag][i] = Payloadconvert.c[i];
                    }

                    if(MFlag == 2)
                    {
                        // ToDo Error
                    }
                    NRF_SetRXState();
                    NRF_CE_Hi;
                    MFlag++;
                    break;
            }
            NRF_FlushRXBuffer();
            break;

        case TX_DS :
            NRF_WriteRegister(0xff, STATUS);            // clear Flags
            WorkerFlag = 0x03;
            NRF_FlushTXBuffer();
            NRF_SetRXState();
            break;

        case MAX_RT :
            NRF_WriteRegister(0xff, STATUS);            // clear Flags
            if(RT < 2 )
            {
                WorkerFlag = 0x08;
            }
            else
            {
                WorkerFlag = 0x20;
                NRF_FlushTXBuffer();
                RT = 0x00;
            }
            break;

        default :
            WorkerFlag = 0x00;
            break;
    }

    NRF_WriteRegister(0xff, STATUS);            // clear Flags

    SREG = tmp_sreg;                            // Restore SREG
}
*/
```