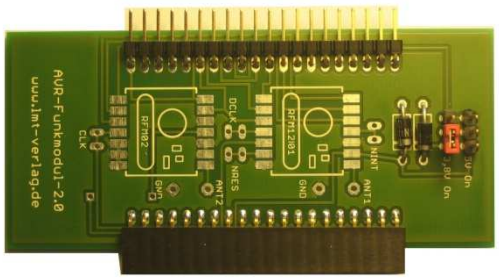
4.18. Ein Funkmodul

Das FSK-Modul RFM12B ist ein Sende- und Empfangsmodul bei einer Frequenz von 433 MHz. Das Modul besitzt eine SPI-Schnittstelle und ist damit für den Anschluss an den Mikrocontroller ATmega8 bestens geeignet. Für eine vollständige Beschreibung wird auf die Herstellerseite *www.hoperf.com* verwiesen.

Für den Anschluss dieses und anderer Module dieser Serie an das myAVR Board steht ein Adapter zur Verfügung.

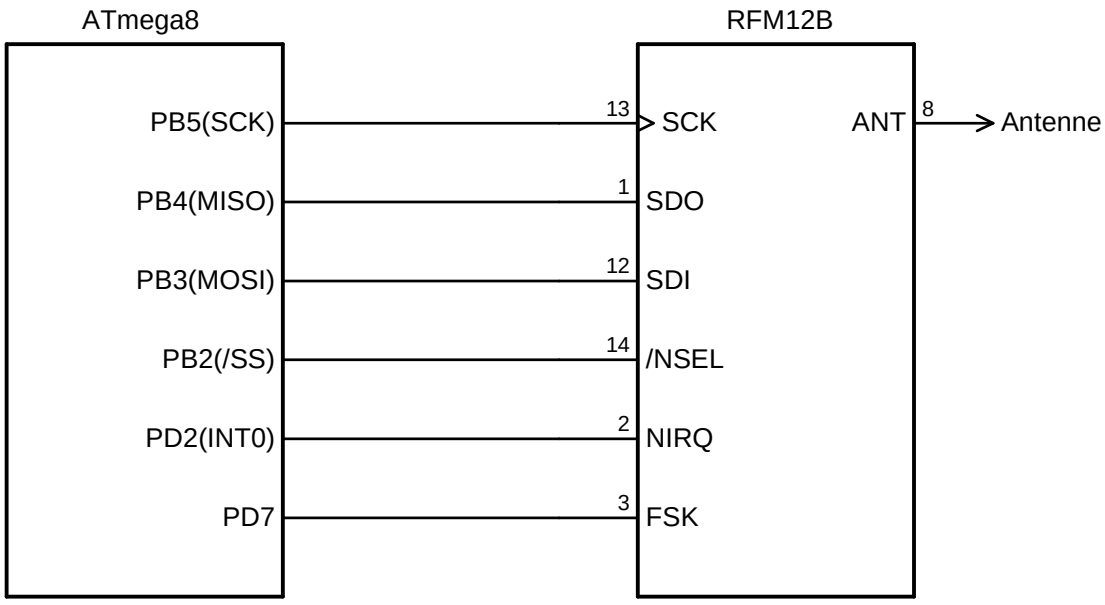
Funkmodul RFM12B

Funkmodul-Adapter



Das folgende Bild zeigt die notwendigen Anschlüsse des RFM12B an den Mikrocontroller. Das RFM-Modul arbeitet im sogenannten FIFO-Mode (first in first out). Dabei werden die Empfangsdaten in einem FIFO-Speicher abgelegt, wenn die Bitfolge 0x2DD4 empfangen wurde.

Beschaltung:



Die folgende Tabelle beschreibt die Bedeutung der Anschlüsse des RFM-Moduls. Die Aufgaben der weiteren Anschlüsse, die in dem folgenden Beispiel nicht benötigt werden, sind dem Datenblatt zu entnehmen.

Anschlüsse des RFM12B:

Anschluss	Bedeutung
SCK	SPI Takt Eingang
SDO	SPI Daten Ausgang
SDI	SPI Daten Eingang
/NSEL	Chip Auswahl (aktiv LOW)
NIRQ	Unterbrechung Anforderung Ausgang (aktiv LOW)
FSK	Auf ‚1' setzen für FIFO-Mode

Die Anschlüsse SCK, SDO, SDI und /NSEL ermöglichen die Verbindung mit der SPI-Schnittstelle des Mikroprozessors. Die Initialisierung und der Datenaustausch des Moduls mit dem Mikroprozessor erfolgt durch Kommandoworte mit einer Breite von 16 Bit. Deswegen sind für den Zugriff immer zwei SPI-Zyklen erforderlich. Die Beschreibung der möglichen Kommandos (z.B. für die Einstellungen der Frequenz, der Baudrate, des FIFO-Modes, ...) würde den Rahmen dieses Kapitels überschreiten. Deswegen wird auch hier auf das Datenblatt verwiesen. Die notwendigen Kommandos gehen aus dem unten beschriebenen Beispiel hervor.

Für die Übertragung eines Kommandos in das RFM-Modul wird die folgende Funktion verwendet. Der Rückgabewert ist ein 16-Bit-Datenwort mit Statusbits.

```
uint16_t RFM12_WRT_CMD(uint16_t aCmd)
{
      uint16_t temp=0x0000;
      uint8_t CmdHi=(aCmd>>8)&0x00ff;     //Commmand High-Byte
      uint8_t CmdLow=aCmd&0x00ff;         //Command Low-Byte
      uint16_t StatHi=0x00;               //Status High-Byte
      uint16_t StatLow=0x00;              //Status Low-Byte
      PORTB&=~(1<<PB2);                   //SS=0
      SPDR=CmdHi;                         //Daten zum Senderegister
      while(!(SPSR&1<<SPIF));             //warte auf Übertragungsende
      StatHi=SPDR;                        //Status High-Byte
      SPDR=CmdLow;                        //Daten zum Senderegister
      while(!(SPSR&1<<SPIF));             //warte auf Übertragungsende
      StatLow=SPDR;                       //Status Low-Byte
      PORTB|=(1<<PB2);                    //SS=1
      temp=(StatHi<<8)|StatLow;           //Statuswort
      return(temp);
}
```

Für die Programmierung ist das Verständnis für den Anschluss NIRQ erforderlich. Eine Unterbrechungsanforderung tritt u.a. dann auf, wenn ein Zeichen empfangen und im FIFO-Speicher abgelegt ist oder wenn ein Zeichen in den FIFO-Speicher zum Senden abgelegt werden kann. Diese Leitung wird für den Polling-Betrieb zum Senden und Empfangen der Daten benutzt.

Die folgende Funktion wartet beispielsweise auf eine Unterbrechung bis das Senderegister frei ist und sendet dann ein 16-Bit-Kommando an das RFM Modul. Das Kommando (0xB0xx) bewirkt ein Senden des Zeichens.

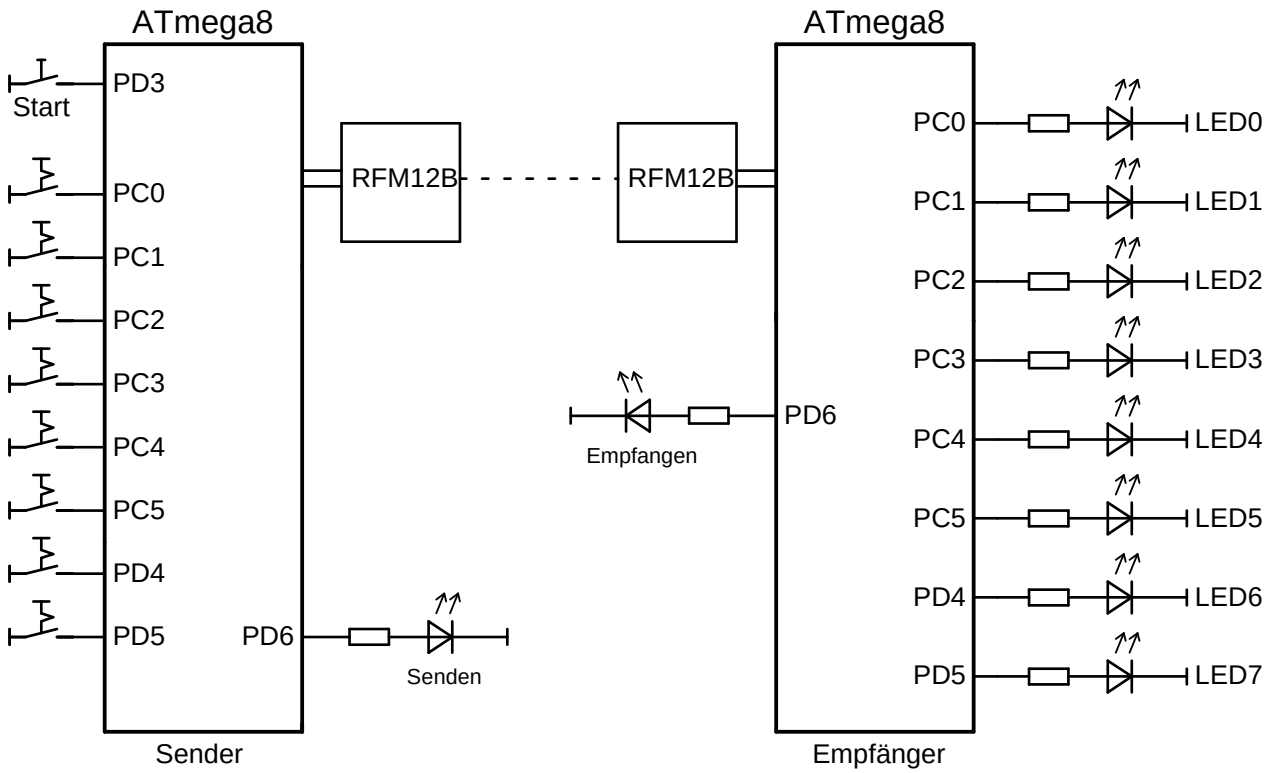
```
void RFM12_SEND(uint8_t aByte)
{
      while(PIND&(1<<PD2));         //warte auf freies Senderegister
      RFM12_WRT_CMD(0xB800+aByte);
}
```

Eine entsprechende Funktion zum Empfang der Daten hat die folgende Form. Das Kommando 0xB000 bewirkt das Auslesen aus dem FIFO-Speicher. Der Rückgabewert ist das empfangene Zeichen.

```
uint8_t RFM12_RECV()
{
uint16_t FIFO_data;                     //Empfangsdaten
while(PIND&(1<<PD2));                   //warte auf nIRQ
RFM12_WRT_CMD(0x0000);                  //Status lesen
FIFO_data=RFM12_WRT_CMD(0xB000);        //Daten lesen
return((uint8_t)FIFO_data&0x00FF);
}
```

Beispiel :

Gemäß dem folgenden Bild ist ein Funksystem bestehend aus Sender und Empfänger aufgebaut.



Wenn der Start-Taster betätigt wird (fallende Flanke) sollen die anliegenden 8-Bit-Daten gesendet und auf den LEDs am Empfänger angezeigt werden.

Zur eindeutigen Identifizierung erhalten Sender und Empfänger eine feste Adresse (z.B. Sender=0x30 – Empfänger=0x31). Gesendet werden die Adresse des Senders, die Adresse des Empfängers und das eigentliche Datenbyte. Um fehlerhafte Übertragungen zu minimieren, wird zusätzlich eine Checksumme gesendet. Insgesamt werden also 4 gültige Bytes übertragen.

Senden und Empfangen werden durch kurzes Aufblinken einer LED (an PD6) angezeigt.

Das Programm für den Sender:

```
/*********************************************************************
Beispiel für RFM12B-Funkmodul (Senden)
Dateiname:  RFM12B_Sender.c
Prozessor:  ATMEGA8 - 4,6864 MHz-Quartz
Frequenz:   434MHz
Datenrate:  4.8kbps
Pin-Belegung:
-----------------------
ATMEGA8           RFM12/IO-Modul
-------------------------------------------
SCK (PB5)   A     SCK
MISO (PB4)  E     SDO
MOSI (PB3)  A     SDI
SS   (PB2)  A     nSEL
INT0 (PD2)  E     nIRQ
PD3         E     START-Taster
PD6         A     Senden-LED
PD7         A     FSK
PC0...PC5   E     Datenbyte Bit0...Bit5
PD4...PD5   E     Datenbyte Bit6...Bit7
*********************************************************************/
#include<avr/io.h>
#define F_CPU 3686400UL
#include<util/delay.h>
#define Sendadress 0x30 //bitte anpassen
#define Recvadress 0x31 //bitte anpassen
void PORT_INIT(void)
{
      DDRC=0b00000000;        //PC0...PC7 zur Eingabe
      PORTC=0b11111111;       //Pull-up für PORTC
      DDRD=0b11000000;        //PD6...PD7 zur Ausgabe
      PORTD=0b00111000;       //Pull-up für PORTD
      DDRB=0b00101100;        //PB2,PB3,PB5 zur Ausgabe
      SPCR=0b01010000;        //SPI,MSTR,CLK=f/4
      PORTB|=(1<<PB2);        //SS=1
      PORTD|=(1<<PD7);        //setze FSK auf High
}

uint16_t RFM12_WRT_CMD(uint16_t aCmd)
{
      uint16_t temp=0x0000;
      uint8_t CmdHi=(aCmd>>8)&0x00ff;     //Commmand High-Byte
      uint8_t CmdLow=aCmd&0x00ff;         //Command Low-Byte
      uint16_t StatHi=0x00;               //Status High-Byte
      uint16_t StatLow=0x00;              //Status Low-Byte
      PORTB&=~(1<<PB2);                   //SS=0
      SPDR=CmdHi;                         //Daten zum Senderegister
      while(!(SPSR&1<<SPIF));             //warte auf Übertragungsende
      StatHi=SPDR;                        //Status High-Byte
      SPDR=CmdLow;                        //Daten zum Senderegister
      while(!(SPSR&1<<SPIF));             //warte auf Übertragungsende
```

```
      StatLow=SPDR;                     //Status Low-Byte
      PORTB|=(1<<PB2);                  //SS=1
      temp=(StatHi<<8)|StatLow;         //Statuswort
      return(temp);
}

void RFM12_INIT(void)
{
      RFM12_WRT_CMD(0x80D7);  //EL,EF,12.0pF
      RFM12_WRT_CMD(0x8239);  //!er,!ebb,ET,ES,EX,!eb,!ew,DC
      RFM12_WRT_CMD(0xA640);  //A640=430.8MHz
      RFM12_WRT_CMD(0xC647);  //4.8kbps
      RFM12_WRT_CMD(0xCC77);  //PLL Setting
      RFM12_WRT_CMD(0x94A0);  //VDI,FAST,134kHz,0dBm,-103dBm
      RFM12_WRT_CMD(0xC2AC);  //AL,!ml,DIG,DQD4
      RFM12_WRT_CMD(0xCA81);  //FIFO8,SYNC,!ff,DR
      RFM12_WRT_CMD(0xC49B);  //AFC Command
      RFM12_WRT_CMD(0x9850);  //!mp,9810=30kHz,MAX OUT
      RFM12_WRT_CMD(0xE000);  //Nicht benötigt
      RFM12_WRT_CMD(0xC800);  //Nicht benötigt
      RFM12_WRT_CMD(0xC000);  //1.66MHz,2.2V
}

void RFM12_SEND(uint8_t aByte)
{
      while(PIND&(1<<PD2));         //warte auf freies Senderegister
      RFM12_WRT_CMD(0xB800+aByte);
}

void DATA_SEND()
{
      uint8_t ChkSum;         //Checksumme
      uint8_t wert=0x00;      //Datenbyte
      wert=(PINC&0x3f)|((PIND&0x30)<<2); //Datenbyte bilden
      PORTD|=(1<<PD6);        //Senden-LED ein
      RFM12_WRT_CMD(0x0000);  //lese Status Register
      RFM12_WRT_CMD(0x8229);  //!er,!ebb,ET,ES,EX,!eb,!ew,DC
      ChkSum=0;
      RFM12_SEND(0xAA);       //PREAMBLE
      RFM12_SEND(0xAA);       //PREAMBLE
      RFM12_SEND(0xAA);       //PREAMBLE
      RFM12_SEND(0x2D);       //SYNC HI BYTE
      RFM12_SEND(0xD4);       //SYNC LOW BYTE
      RFM12_SEND(Sendadress); //Sender-Adresse
      ChkSum+=Sendadress;
      RFM12_SEND(Recvadress); //Empfänger-Adresse
      ChkSum+=Recvadress;
      RFM12_SEND(wert);       //Datenbyte
      ChkSum+=wert;
      RFM12_SEND(ChkSum);     //sende ChkSumme
      RFM12_SEND(0xAA);       //DUMMY BYTE
      RFM12_SEND(0xAA);       //DUMMY BYTE
      RFM12_SEND(0xAA);       //DUMMY BYTE
```

```
      RFM12_WRT_CMD(0x8201);  //Übertragungsende
      PORTD&=~(1<<PD6);       //Senden_LED aus
}

int main(void)
{
      uint8_t i;
      uint8_t fm=0x00;        //Bit0 = Flankenmerker
      PORT_INIT();
      RFM12_INIT();
/********* blinke 3 mal bei Power-On **********/
      for(i=0;i<3;i++)
      {
      _delay_ms(200);
      PORTD|=(1<<PD6);
      _delay_ms(200);
      PORTD&=~(1<<PD6);
      }
      _delay_ms(200);
      while(1)
      {
            if(!(PIND&0x08))        //PD3 = 0 ?
            {
            if(!(fm&0x01))          //Flankenmerker=0 ?
                  {
                  fm|=(1<<0);       //Flankenmerker setzen
                  _delay_ms(50);    //Prellzeit
                  DATA_SEND();
                  }
            }
      else                          //wenn PD3 =1
            {
            if(fm&0x01)             //Flankenmerker=1 ?
                  {
                  fm&=~(1<<0);      //Flankenmerker zurücksetzen
                  _delay_ms(50);    //Prellzeit
                  }
            }
      }
}
```

Das Programm für den Empfänger:

```
/*****************************************************************
Beispiel für RFM12B-Funkmodul (Empfangen)
Dateiname:  RFM12B_Empfaenger.c
Prozessor:  ATMEGA8 - 4,6864 MHz-Quartz
Frequenz:   434MHz
Datenrate:  4.8kbps
Pin-Belegung:
-----------------------------------------
ATMEGA8      RFM12/IO-Modul
-----------------------------------------
SCK (PB5)   A   SCK
MISO (PB4)  E   SDO
MOSI (PB3)  A   SDI
SS   (PB2)  A   nSEL
INT0 (PD2)  E   nIRQ
PD6         A   Empfangen-LED
PD7         A   FSK
PC0...PC5   A   Datenbyte Bit0...Bit5
PD4...PD5   A   Datenbyte Bit6...Bit7
*****************************************************************/
#include<avr/io.h>
#define F_CPU 3686400UL
#include<util/delay.h>
#define Sendadress 0x30
#define Recvadress 0x31

void PORT_INIT(void)
{
      DDRD=0b11110000;  //PD4...PD7 zur Ausgabe
      DDRC=0b00111111;  //PC0...PC5 zur Ausgabe
      DDRB=0b00101100;  //PB2,PB3,PB5 zur Ausgabe
      SPCR=0b01010000;  //SPI,MSTR,CLK=f/4
      PORTB|=(1<<PB2);  //SS=1
      PORTD|=(1<<PD7);  //setze FSK auf High
}
uint16_t RFM12_WRT_CMD(uint16_t aCmd)
{
      uint16_t temp=0x0000;
      uint8_t CmdHi=(aCmd>>8)&0x00ff;     //Commmand High-Byte
      uint8_t CmdLow=aCmd&0x00ff;         //Command Low-Byte
      uint16_t StatHi=0x00;               //Status High-Byte
      uint16_t StatLow=0x00;              //Status Low-Byte
      PORTB&=~(1<<PB2);                   //SS=0
      SPDR=CmdHi;                         //Daten zum Senderegister
      while(!(SPSR&1<<SPIF));             //warte auf Übertragungsende
      StatHi=SPDR;                        //Status High-Byte
      SPDR=CmdLow;                        //Daten zum Senderegister
      while(!(SPSR&1<<SPIF));             //warte auf Übertragungsende
      StatLow=SPDR;                       //Status Low-Byte
      PORTB|=(1<<PB2);                    //SS=1
      temp=(StatHi<<8)|StatLow;           //Statuswort
```

```
      return(temp);
}

void RFM12_INIT(void)
{
RFM12_WRT_CMD(0x80D7); //EL,EF,12.0pF
RFM12_WRT_CMD(0x82D8); //enable receive, !PA
RFM12_WRT_CMD(0xA640); //A640=430.8MHz
RFM12_WRT_CMD(0xC647); //4.8kbps
RFM12_WRT_CMD(0x94A0); //VDI,FAST,134kHz,0dBm,-103dBm
RFM12_WRT_CMD(0xC2AC); //AL,!ml,DIG,DQD4
RFM12_WRT_CMD(0xCC77); //PLL Setting
RFM12_WRT_CMD(0xCA80); //FIFO8,SYNC,!ff,DR
RFM12_WRT_CMD(0xCA83); //FIFO8, SYNC
RFM12_WRT_CMD(0xC49B); //AFC Command
RFM12_WRT_CMD(0x9850); //!mp,9810=30kHz,MAX OUT
RFM12_WRT_CMD(0xE000); //Nicht benötigt
RFM12_WRT_CMD(0xC800); //Nicht benötigt
RFM12_WRT_CMD(0xC000); //1.66MHz,2.2V
}

uint8_t RFM12_RECV()
{
uint16_t FIFO_data;               //Empfangsdaten
while(PIND&(1<<PD2));             //warte auf nIRQ
RFM12_WRT_CMD(0x0000);            //Status lesen
FIFO_data=RFM12_WRT_CMD(0xB000);  //Daten lesen
return((uint8_t)FIFO_data&0x00FF); //Empfangsdaten zurückgeben
}

int main(void)
{
uint8_t i;
uint8_t ChkSum;        //Checksumme
uint8_t wert[3];       //Empfangsdaten
PORT_INIT();
/***************** blinke 3 mal bei Power-On *****************/
for(i=0;i<3;i++)
{
_delay_ms(200);
PORTD|=(1<<PD6);
_delay_ms(200);
PORTD&=~(1<<PD6);
}
_delay_ms(200);
RFM12_INIT();
RFM12_WRT_CMD(0xCA80); //FIFO initialisieren
```

```
while(1)
{
      RFM12_WRT_CMD(0xCA83);       //FIFO aktivieren
      ChkSum=0;
      wert[0]=RFM12_RECV();        //Sender-Adresse
      wert[1]=RFM12_RECV();        //Empfänger-Adresse
      wert[2]=RFM12_RECV();        //Daten
      for(i=0;i<3;i++)
      {
      ChkSum+=wert[i];             //Checksumme bilden
      }
      i=RFM12_RECV();              //Checksumme
      RFM12_WRT_CMD(0xCA81);       //FIFO deaktivieren
      if(ChkSum==i)                      //richtige Checksumme?
      {
//Sender- und Empfängeradresse korrekt?
        if((wert[0]==Sendadress)&&(wert[1]==Recvadress))
        {
        PORTC=wert[2]&0x3f;        //Daten an LEDs ausgeben
        PORTD=((wert[2]&0xc0)>>2)|0x80;
        PORTD|=(1<<PD6);           //Empfangen-LED ein
       _delay_ms(200);
        PORTD&=~(1<<PD6);          //Empfangen-LED aus-
        }
      }
    }
}
```