NRF24L01
About Nrf24L01

Features

The Nordic Semiconductor nRF24L01 is descended from the TXRX24G, but it offers some important features that make it much easier to deal with:

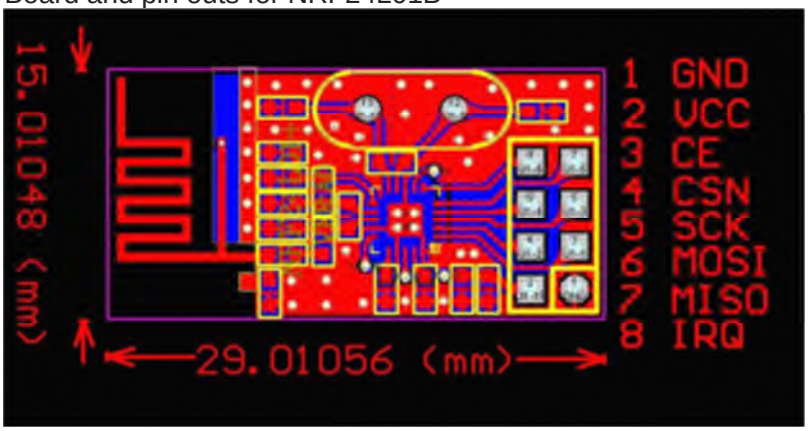
- Enhanced Shockburst™: This is Nordic's link layer protocol that gives the radio auto-acknowledgement, auto-retransmit, and packet loss detection capability. It will make the wireless link much more reliable without adding complexity to the students' application programs.
- SPI bus: The nRF24L01 implements a 4-wire Serial Peripheral Interface bus. This allows the software driver to use the microcontroller's native high speed SPI module for communications instead of having to do manual bit bashing, which is less reliable, slower, and more difficult to implement.
- External antenna: This should improve the new radio's sensitivity and range over those of the old radios.
- MultiCeiver™: Each radio can receive packets on up to six different addresses. This allows us to implement features such as selective packet broadcasting without sacrificing other functionality.

Other features: Variable packet widths, packet queueing, ack packet payload, etc.

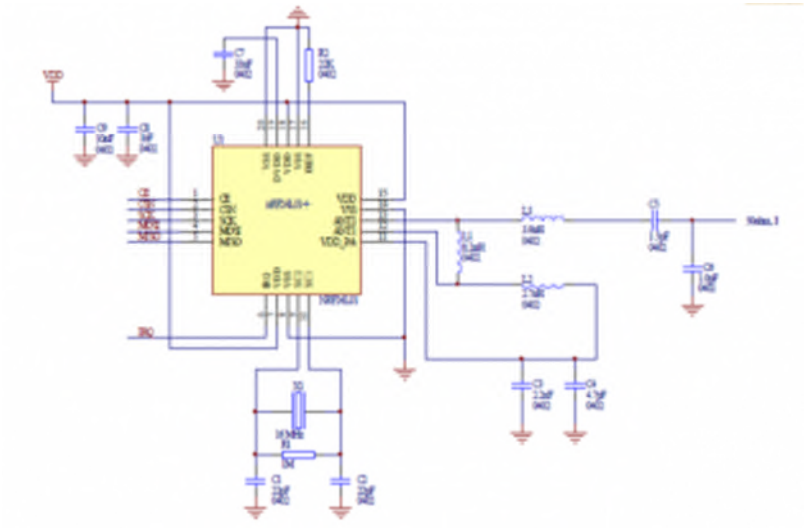
Firmware

Pin Definition

Board and pin outs for NRF24L01B



Board classical peripheral circuit



Documentation

- Datasheet

Tutorial

- Arduino official tutorial for nrf24l01
- Transfer temperature data via NRF24l01 with STC51
- sending a data array over nrf24l01 with mirf, how?
- nRF24L01 2.4GHz Radio/Wireless Transceivers How-To
- Nrf24L01
- nRF24L01-Mirf-Examples
- RF24 v1

Library and Code

- Library RF24
- Code for mega 16
- AVRLib/nRF24L01 – TinkerWiki

This page by Stefan Engekle presents code for getting the nRF24L01 working on an ATMEGA168. It was invaluable to my initial research into the radio, and gave me a clear understanding of how to interact with the registers.

- AVRLib/SPI – TinkerWiki

This is by the same author as the above link. It gives code for using SPI on the AVR. My SPI code is a copy and paste job from this page, with a few minor modifications.

- DIY Embedded

Brennen Bell provides five tutorials for the nRF24L01, four of them with code for PIC and ARM. His tutorial 0 was especially useful for me because of its commentary on the radio's registers, which provided information that was not in the product specification.

Arduino Library

Supported Arduino library

- RadioHead
- RF24
- Mirf

Use Mirf Library

- Mirf

Usage

- Download library Mirf for Arduino and install it.
- Connect nrf24l01 module to Arduino as follow:

Arduino	Nrf24L01 Module
MI 12	MISO
MO 11	MOSI
SCK 13	SCK
9	CE
10	SCN
VCC 3.3	VCC
GND	GND
NC	IRQ

- It is better to connect to 3V3 logic, instead of connecting it directly to 5V arduino board, but use 5V logic is not a problem, just 3v3 more suitable.
- VCC must connect 3V3, use 5V will be risky for long time

Demo codes

- Header (Same for sender and receiver)

```
//first included your library
#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>
```

- Setup void (Still same for sender and receiver)

```
void setup(){
   Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
```

```
    //Mirf.csnPin = 10; (This is optional to change the chip select pin)
    //Mirf.cePin = 9; (This is optional to change the enable pin)
    Mirf.init();
    Mirf.setTADDR((byte *)"serv1");
    Mirf.payload = 32;
    Mirf.config();
    Serial.println("Beginning ... "); // "Beginning ..." on sender, or "Listening ..."
on sever (Receiver)
  }
```

Codes at Transimitter

- Send string in char directly

```
void loop(){
    Mirf.send((byte *) "Hello");
    delay(500);
}
```

- Send string in char (Optional writing)

```
void loop(){
    char a[6] ="abced"; // char
    Mirf.send((byte *) a);
    delay(500);
}
```

- Or:

```
void loop(){
    int Vals[6] = {2, 4, -8, 3, 2};
    Mirf.send((byte *) a);
    delay(500);
}
```

Codes at Receiver

- Receive standard 32 Bytes

```
void loop(){
     byte data[32]; // or int data[32];
     if(!Mirf.isSending() && Mirf.dataReady()){
          Serial.println("Got packet");
          Mirf.getData((byte *) &data);
          Serial.write(byte(data[0])); //h
          Serial.write(byte(data[1])); //e
          Serial.write(byte(data[2])); //l
          Serial.write(byte(data[3])); //l
          Serial.write(byte(data[4])); //o
          Serial.println("");
     }
```

```
}
```

- Receive more than 32 bytes

```
Mirf.getData(data);
int i;
for (i = 0; i < sizeof(data); i++)
    {
    Serial.println(data[i]); //Serial.print(data[i], DEC);
    }
```

The completed Demo Code

Sender

```
 //first included your library
#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>
void setup(){
   Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
    Mirf.csnPin = 10; //(This is optional to change the chip select pin)
    Mirf.cePin = 9; //(This is optional to change the enable pin)
    Mirf.init();
    Mirf.setTADDR((byte *)"serv1");
    Mirf.payload = 32;
    Mirf.config();
    Serial.println("Beginning ... "); // "Beginning ..." on sender, or "Listening ..." on sever (Receiver)
  }

  void loop(){
    Mirf.send((byte *) "Hello");
    delay(500);
}
```

Receiver

```
 //first included your library
#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>

void setup(){
   Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
    Mirf.csnPin = 10; //(This is optional to change the chip select pin)
    Mirf.cePin = 9; //(This is optional to change the enable pin)
    Mirf.init();
    Mirf.setTADDR((byte *)"serv1");
    Mirf.payload = 32;
    Mirf.config();
    Serial.println("Beginning ... "); // "Beginning ..." on sender, or "Listening ..." on sever (Receiver)
  }

void loop(){
     byte data[32]; // or int data[32];
     if(!Mirf.isSending() && Mirf.dataReady()){
          Serial.println("Got packet");
          Mirf.getData((byte *) &data);
          Serial.write(byte(data[0])); //h
          Serial.write(byte(data[1])); //e
          Serial.write(byte(data[2])); //l
          Serial.write(byte(data[3])); //l
          Serial.write(byte(data[4])); //o
          Serial.println("");
```

```
    }
}
```

Result

Turn on the serial port on receiver Arduino, it will show up:

```
Beginning ...
Got packet
Hello
Got packet
Hello
Got packet
Hello
Got packet
Hello
```

- During the "beginning ..." line, press the restart button on the sender arduino, then reset the board and let it start to send data.

Properties:

byte	Function	Default	Note
cePin	CE Pin controls RX / TX	8	-
csnPin	CSN Pin (Chip select not)	7	-
channel	RF Channel 0 - 127 or 0 - 84 in the US	0	-
payload	Size in bytes	Default 16, max 32	Channel and payload must be the same for all nodes.

Functions

Function	Description	example
void init(void)	Initialize the module, set the pin modes for the configurable pins and initialize the SPI module.	Mirf.init();
void setRADDR(byte *addr)	Set the receiving address. Addresses are 5 bytes long.	Mirf.setRADDR((byte *)"addr1");
void setTADDR(byte *addr)	Set the sending address.	Mirf.setTADDR((byte *)"addr1");

void config(void)	Set channel and payload width. Power up in RX mode and flush RX fifo.	Mirf.payload = 32; Mirf.channel = 2; Mirf.config();

bool dataReady(void)	Is there data ready to be received?	if(Mirf.dataReady()){
		//Get the data to play with.
		}

void getData(byte *data)	Get the received data. 'data' should be an array of bytes Mirf.payload long.	byte data[Mirf.payload] Mirf.getData(data);
void send(byte *data)	Send data. 'data' should be Mirf.payload bytes long.	-

bool isSending(void)	Return true if still trying to send. If the chip is still in transmit mode then this method will return the chip to receive mode.	Mirf.send(data);
		while(Mirf.isSending()){
		//Wait.
		} //Chip is now in receive mode.
		NB: Lots more information is available from the status registers regarding acknolagement or failure status. See Mirf.cpp:218.

bool rxFifoEmpty(void)	Is the RX Fifo Empty.	-
bool txFifoEmpty(void)	Is the TX Fifo Empty.	-
byte getStatus(void)	Return the status register.	-
void powerUpRx(void)	Power up chip and set to receive mode. Also clear sending interrupts.	-
void powerUpTx(void)	Power up tx mode.	-