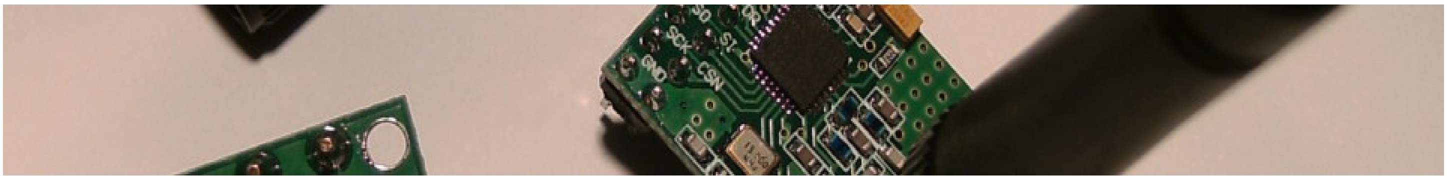


LCD Image Viewer (https://blog.zakkemble.net/lcd-image-viewer/)

Getting Bluetooth modules talking to each other (https://blog.zakkemble.net/getting-bluetooth-modules-talking-to-each-other/)

Jan 01 2013

nRF905 Radio Library for AVR and Arduino

- By Zak Kemble (https://blog.zakkemble.net/author/Zak/) in Arduino (https://blog.zakkemble.net/category/arduino/), AVR (https://blog.zakkemble.net/category/microcontroller/avr/), Libraries (https://blog.zakkemble.net/category/library/), Microcontroller (https://blog.zakkemble.net/category/microcontroller/), Radio (https://blog.zakkemble.net/category/radio/)

The nRF905 is a radio transceiver IC similar to the well known nRF24L01, but operates at 433/898/915MHz instead of 2.4GHz, has a much longer range and a few extra IO pins. However, the nRF905 data rate is only 50Kbps compared to nRF24L01's 2Mbps.

This library offers quite a bit of flexibility: Optional use of interrupts, 2 of the connections to the module are optional since their states can also be accessed by the ICs status register, and supports basic collision avoidance.

NOTE: v3.0.0 of the library was released on 12th September 2017, the default CD pin has changed and the AM pin is now used by the library.

Download:
Arduino (https://github.com/zkemble/nRF905-arduino) – Documentation (http://zkemble.github.io/nRF905-arduino/) (or use the Arduino library manager, search for "nrf905")
AVR (non-Arduino) (https://github.com/zkemble/nRF905) – Documentation (http://zkemble.github.io/nRF905/)

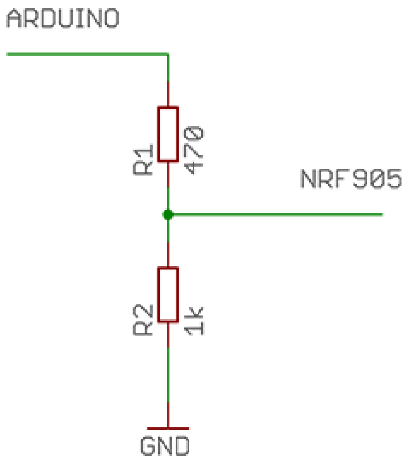
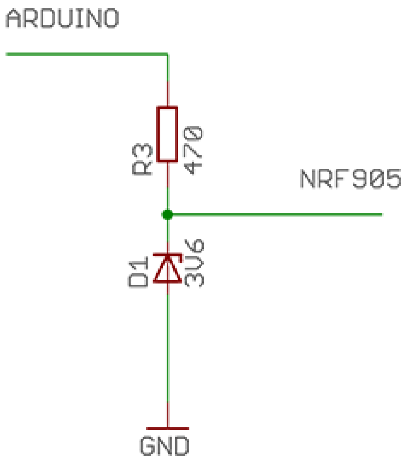
nRF905	ATmega48/88/168/328	Arduino Uno	Description
VCC	3.3V	3.3V	**Power** (3.3V)
CE	D7 (13)	7	**Standby** – High = TX/RX mode, Low = standby
TXE	B1 (15)	9	**TX or RX mode** – High = TX, Low = RX
PWR	B0 (14)	8	**Power-up** – High = on, Low = off
CD	D4 (6)	4	**Carrier detect** – High when a signal is detected, for collision avoidance
AM	D2 (4)	2	**Address Match** – High when receiving a packet that has the same address as the one set for this device, optional since state is stored in register, if interrupts are used (default) then this pin must be connected
DR	D3 (5)	3	**Data Ready** – High when finished transmitting/High when new data received, optional since state is stored in register, if interrupts are used (default) then this pin must be connected

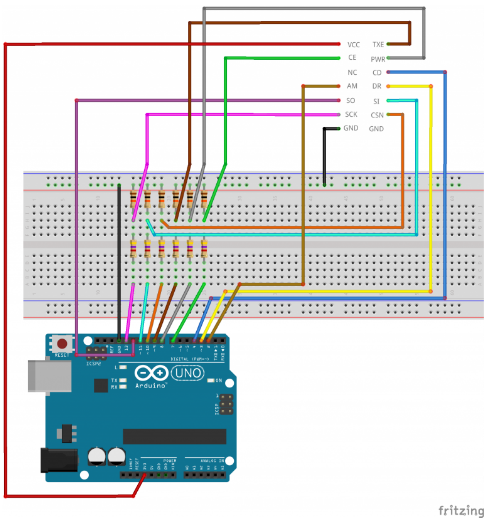
SO	B4 (18)	12	**SPI MISO** (Mega pin 50)
SI	B3 (17)	11	**SPI MOSI** (Mega pin 51)
SCK	B5 (19)	13	**SPI SCK** (Mega pin 52)
CSN	B2 (16)	10	**SPI SS**
GND	GND	GND	**Ground**

Some of the module pin names differ from the IC pin names in the datasheet:

Module	**IC**
CE	TRX_EN
TXE	TX_EN

The nRF905 is not 5V compatible, so some level conversions will need to be done with the Arduino outputs. A simple voltage divider or resistor and zener diode will do the trick. Only TXE, CE, PWR, SI, SCK and CSN pins need level conversion (not CD, AM, DR and SO).

Divider	**Zener**
 (https://blog.zakkemble.net/wp-content/uploads/2020/11/res_div.png)	 (https://blog.zakkemble.net/wp-content/uploads/2013/01/zener-1.png)



(http://blog.zakkemble.co.uk/wp-content/uploads/2013/01/nrf905_bb-1.png)

The nRF905 has 511 channels ranging 422.4MHz – 473.5MHz in 100KHz steps on the 433MHz band and 844.8MHz – 947MHz in 200KHz steps on the 868/915MHz band (remember to check which frequencies are legal in your country!), but each channel overlaps adjacent channels so there are only a total of 170 usable channels at once.

Searching for nRF905, PTR8000 and PTR8000+ should yield some results for modules on AliExpress, Ebay and DealExtreme. You should be able to get 2 for around £10.

Callbacks

Since v3.0.0 this library uses callbacks which are ran when events occur. If the option to use interrupts is enabled then the callbacks will run from the interrupt routine so be sure that any global variables you use in them are declared 'volatile', just as you would when dealing with normal ISRs. These events will wake the microcontroller if it is sleeping.

Event	Callback	Notes
New packet incoming	NRF905_CB_ADDRMATCH	
Valid packet received	NRF905_CB_RXCOMPLETE	
Invalid packet received	NRF905_CB_RXINVALID	
Packet transmission complete	NRF905_CB_TXCOMPLETE	This only works if the nextMode is NRF905_NEXTMODE_STANDBY when calling nRF905_TX()

Nitty-gritty radio stuff

The actual air data-rate of nRF905 is 100Kbps, but the data is Manchester encoded which halves the throughput to 50Kbps. The modulation is GFSK with ±50KHz deviation. The radio also adds a few extra bits of data to the address and payload; a preamble and CRC.

Transmitted packet

Preamble	Address	Payload	CRC
10 bits	**1 or 4 bytes**	**1 – 32 bytes**	**0 – 2 bytes**

The address is also used as the syncword and should have as many level shifts as possible. 10100110 01100101 00011010 11011010 (2791643866) would be a good address to use, but 00000000 00000000 00000000 0010000 (32) would not be.

The CRC is used to detect errors in the received data. Having a CRC doesn't eliminate all bad packets, there is always a small chance of a bad packet passing the CRC. Using larger CRCs helps to reduce that chance.
When the CRC is set to 16 bit the radio uses the CRC16-CCITT-FALSE (0xFFFF) algorithm. I'm not sure what it uses for 8 bit CRCs.

Transmission time

The sizes of the address, payload and CRC can be adjusted for a balance between throughput and latency.

Example configurations

Address size	Payload size	CRC size	=	Throughput (bps)	Latency (ms)
4	32	2	=	36940	6.93
1	4	1	=	17680	1.81
1	1	0	=	6838	1.17

Throughput is how much useful data can be sent (the payload part), assuming no delays for writing and reading the payload.
Latency is how long it takes for the transmission to complete.
Switching from standby mode to receive or transmit mode takes a maximum of 650us and switching between receive and transmit modes takes a maximum of 550us.

Transmission time can be worked out with:

$$t = t_{startup} + t_{preamble} + ((N_{address} + N_{payload} + N_{CRC}) / BR)$$

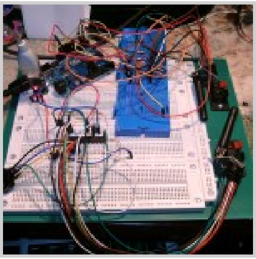
$t_{startup}$ is the time to switch mode as stated above (650us / 550us).
$t_{preamble}$ is 200us for the 10 bit preamble.
$N_{address}$, $N_{payload}$ and N_{CRC} are the address, payload and CRC sizes in bits.
BR is the bit rate which is 50,000.

For config 1, switching from standby to transmit:

$$t = 0.00065 + 0.0002 + ((32 + 256 + 16) / 50000)$$

$$t = 0.00693 \text{ seconds } (6.93\text{ms})$$

A note to developers: Before v3.0.0 this library would load the address bytes in reverse order, keep that in mind if you're using an old version of the library to interface with other radio systems!



(https://blog.zakkemble.net/wp-content/uploads/2012/12/PICT1443.jpg)

(https://blog.zakkemble.net/wp-content/uploads/2012/12/modules.jpg)

610 comments

3 pings

Skip to comment form

1.
   - Chris on July 28, 2019
   - # (https://blog.zakkemble.net/nrf905-avrarduino-librarydriver/comment-page-5/#comment-208980)
   - Reply (https://blog.zakkemble.net/nrf905-avrarduino-librarydriver/?replytocom=208980#respond)

   Hello Zak,
   First of all thank you for this library. Great work!
   The communication of my modules works flawlessly. I have also connected 2 Sevomotors to the Mega (PIN 40 and 42 with separate power supply) which permanently ticks when the communication starts. Do you have any idea why this could be?
   Many thanks and best regards,
   Chris

   1.
      - Zak Kemble () on July 28, 2019
        Author
        # (https://blog.zakkemble.net/nrf905-avrarduino-librarydriver/comment-page-5/#comment-208982)
      - Reply (https://blog.zakkemble.net/nrf905-avrarduino-librarydriver/?replytocom=208982#respond)

      Hey Chris, thanks! You'll have to post your code before I can see what's up! -> https://pastebin.com/ (https://pastebin.com/)

2.
   - Tim on September 27, 2019
   - # (https://blog.zakkemble.net/nrf905-avrarduino-librarydriver/comment-page-5/#comment-209049)
   - Reply (https://blog.zakkemble.net/nrf905-avrarduino-librarydriver/?replytocom=209049#respond)

   Hi Zak, I've built a few of these links running at 4800 baud to handle 30 bytes of data every 250mS running at 433MHz carrier.

   However on one of the links it runs ok for about 20 seconds then hangs no tx or rx activity. This problem can happen at either end and I suspect it may actually be nRf905 module related.

   I've tried disabling IRQ and other variants but always the same result.

   Pressing the reset button on the tx Arduino fixes the problem. Maybe there's an overflow problem somewhere ?

   Any pointers on how to debug ?

   I'll try slowing down the refresh rate from 250mS to 1S and see what happens