RF24 v1

RF24 Class Reference

Driver for nRF24L01(+) 2.4GHz Wireless Transceiver. More...

`#include <RF24.h>`

List of all members.

Public Member Functions

Primary public interface

These are the main methods you need to operate the chip

	RF24 (uint8_t _cepin, uint8_t _cspin) Constructor.
void	**begin** (void) Begin operation of the chip.
void	**startListening** (void) Start listening on the pipes opened for reading.
void	**stopListening** (void) Stop listening for incoming messages.
bool	**write** (const void *buf, uint8_t len) Write to the open writing pipe.
bool	**available** (void) Test whether there are bytes available to be read.
bool	**read** (void *buf, uint8_t len) Read the payload.
void	**openWritingPipe** (uint64_t address) Open a pipe for writing.
void	**openReadingPipe** (uint8_t number, uint64_t address) Open a pipe for reading.

Optional Configurators

Methods you can use to get or set the configuration of the chip.

*None are required. Calling **begin()** sets up a reasonable set of defaults.*

void	**setRetries** (uint8_t delay, uint8_t count) Set the number and delay of retries upon failed submit.
void	**setChannel** (uint8_t channel) Set RF communication channel.

void	**setPayloadSize** (uint8_t size) Set Static Payload Size.
uint8_t	**getPayloadSize** (void) Get Static Payload Size.
uint8_t	**getDynamicPayloadSize** (void) Get Dynamic Payload Size.
void	**enableAckPayload** (void) Enable custom payloads on the acknowledge packets.
void	**enableDynamicPayloads** (void) Enable dynamically-sized payloads.
bool	**isPVariant** (void) Determine whether the hardware is an nRF24L01+ or not.
void	**setAutoAck** (bool enable) Enable or disable auto-acknowlede packets.
void	**setAutoAck** (uint8_t pipe, bool enable) Enable or disable auto-acknowlede packets on a per pipeline basis.
void	**setPALevel** (**rf24_pa_dbm_e** level) Set Power Amplifier (PA) level to one of four levels.
rf24_pa_dbm_e	**getPALevel** (void) Fetches the current PA level.
bool	**setDataRate** (**rf24_datarate_e** speed) Set the transmission data rate.
rf24_datarate_e	**getDataRate** (void) Fetches the transmission data rate.
void	**setCRCLength** (**rf24_crclength_e** length) Set the CRC length.
rf24_crclength_e	**getCRCLength** (void) Get the CRC length.
void	**disableCRC** (void) Disable CRC validation.

Advanced Operation

Methods you can use to drive the chip in more advanced ways

void	**printDetails** (void) Print a giant block of debugging information to stdout.
void	**powerDown** (void) Enter low-power mode.
void	**powerUp** (void) Leave low-power mode - making radio more responsive.
bool	**available** (uint8_t *pipe_num) Test whether there are bytes available to be read.
void	**startWrite** (const void *buf, uint8_t len) Non-blocking write to the open writing pipe.
void	**writeAckPayload** (uint8_t pipe, const void *buf, uint8_t len) Write an ack payload for the specified pipe.
bool	**isAckPayloadAvailable** (void) Determine if an ack payload was received in the most recent call to **write()**.
void	**whatHappened** (bool &tx_ok, bool &tx_fail, bool &rx_ready) Call this when you get an interrupt to find out why.
bool	**testCarrier** (void) Test whether there was a carrier on the line for the previous listening period.

bool	**testRPD** (void) Test whether a signal (carrier or otherwise) greater than or equal to -64dBm is present on the channel.

Protected Member Functions

Low-level internal interface.

Protected methods that address the chip directly.

Regular users cannot ever call these. They are documented for completeness and for developers who may want to extend this class.

void	**csn** (int mode) Set chip select pin.
void	**ce** (int level) Set chip enable.
uint8_t	**read_register** (uint8_t reg, uint8_t *buf, uint8_t len) Read a chunk of data in from a register.
uint8_t	**read_register** (uint8_t reg) Read single byte from a register.
uint8_t	**write_register** (uint8_t reg, const uint8_t *buf, uint8_t len) Write a chunk of data to a register.
uint8_t	**write_register** (uint8_t reg, uint8_t value) Write a single byte to a register.
uint8_t	**write_payload** (const void *buf, uint8_t len) Write the transmit payload.
uint8_t	**read_payload** (void *buf, uint8_t len) Read the receive payload.
uint8_t	**flush_rx** (void) Empty the receive buffer.
uint8_t	**flush_tx** (void) Empty the transmit buffer.
uint8_t	**get_status** (void) Retrieve the current status of the chip.
void	**print_status** (uint8_t status) Decode and print the given status to stdout.
void	**print_observe_tx** (uint8_t value) Decode and print the given 'observe_tx' value to stdout.
void	**print_byte_register** (prog_char *name, uint8_t reg, uint8_t qty=1) Print the name and value of an 8-bit register to stdout.
void	**print_address_register** (prog_char *name, uint8_t reg, uint8_t qty=1) Print the name and value of a 40-bit address register to stdout.
void	**toggle_features** (void) Turn on or off the special features of the chip.

Detailed Description

Driver for nRF24L01(+) 2.4GHz Wireless Transceiver.

Examples:

GettingStarted.pde, **led_remote.pde**, **nordic_fob.pde**, **pingpair.pde**, **pingpair_irq.pde**, **pingpair_maple.pde**, **pingpair_pl.pde**, **pingpair_sleepy.pde**, **scanner.pde**, and **starping.pde**.

Constructor & Destructor Documentation

RF24::RF24 (uint8_t _cepin,
uint8_t _cspin
)

Constructor.

Creates a new instance of this driver. Before using, you create an instance and send in the unique pins that this chip is connected to.

Parameters:

_cepin The pin attached to Chip Enable on the RF module

_cspin The pin attached to Chip Select

Member Function Documentation

bool RF24::available (void)

Test whether there are bytes available to be read.

Returns:

True if there is a payload available, false if none is

bool RF24::available (uint8_t * pipe_num)

Test whether there are bytes available to be read.

Use this version to discover on which pipe the message arrived.

Parameters:

[out] **pipe_num** Which pipe has the payload available

Returns:

True if there is a payload available, false if none is

void RF24::begin (void)

Begin operation of the chip.

Call this in setup(), before calling any other methods.

void RF24::ce (int level) [protected]

Set chip enable.

Parameters:

level HIGH to actively begin transmission or LOW to put in standby. Please see data sheet for a much more detailed description of this pin.

void RF24::csn (int mode) [protected]

Set chip select pin.

Running SPI bus at PI_CLOCK_DIV2 so we don't waste time transferring data and best of all, we make use of the radio's FIFO buffers. A lower speed means we're less likely to effectively leverage our FIFOs and pay a higher AVR runtime cost as toll.

Parameters:

mode HIGH to take this unit off the SPI bus, LOW to put it on

void RF24::disableCRC (void)

Disable CRC validation.

void RF24::enableAckPayload (void)

Enable custom payloads on the acknowledge packets.

Ack payloads are a handy way to return data back to senders without manually changing the radio modes on both units.

See also:

examples/pingpair_pl/pingpair_pl.pde

void RF24::enableDynamicPayloads (void)

Enable dynamically-sized payloads.

This way you don't always have to send large packets just to send them once in a while. This enables dynamic payloads on ALL pipes.

See also:

examples/pingpair_pl/pingpair_dyn.pde

uint8_t RF24::flush_rx (void) [protected]

Empty the receive buffer.

Returns:

Current value of status register

uint8_t RF24::flush_tx (void) [protected]

Empty the transmit buffer.

Returns:

Current value of status register

uint8_t RF24::get_status (void) [protected]

Retrieve the current status of the chip.

Returns:
Current value of status register

rf24_crclength_e RF24::getCRCLength (void)

Get the CRC length.

Returns:
RF24_DISABLED if disabled or RF24_CRC_8 for 8-bit or RF24_CRC_16 for 16-bit

rf24_datarate_e RF24::getDataRate (void)

Fetches the transmission data rate.

Returns:
Returns the hardware's currently configured datarate. The value is one of 250kbs, RF24_1MBPS for 1Mbps, or RF24_2MBPS, as defined in the rf24_datarate_e enum.

uint8_t RF24::getDynamicPayloadSize (void)

Get Dynamic Payload Size.

For dynamic payloads, this pulls the size of the payload off the chip

Returns:
Payload length of last-received dynamic payload

rf24_pa_dbm_e RF24::getPALevel (void)

Fetches the current PA level.

Returns:
Returns a value from the rf24_pa_dbm_e enum describing the current PA setting. Please remember, all values represented by the enum mnemonics are negative dBm. See setPALevel for return value descriptions.

uint8_t RF24::getPayloadSize (void)

Get Static Payload Size.

See also:
setPayloadSize()

Returns:
The number of bytes in the payload

bool RF24::isAckPayloadAvailable (void)

Determine if an ack payload was received in the most recent call to **write()**.

Call **read()** to retrieve the ack payload.

Warning:
Calling this function clears the internal flag which indicates a payload is available. If it returns true, you must read the packet out as the very next interaction with the radio, or the results are undefined.

Returns:
True if an ack payload is available.

bool RF24::isPVariant (void)

Determine whether the hardware is an nRF24L01+ or not.

Returns:
true if the hardware is nRF24L01+ (or compatible) and false if its not.

void RF24::openReadingPipe (uint8_t number, uint64_t address)

Open a pipe for reading.

Up to 6 pipes can be open for reading at once. Open all the reading pipes, and then call **startListening()**.

See also:
openWritingPipe

Warning:
Pipes 1-5 should share the first 32 bits. Only the least significant byte should be unique, e.g.

```
openReadingPipe(1,0xF0F0F0F0AA);
openReadingPipe(2,0xF0F0F0F066);
```

Pipe 0 is also used by the writing pipe. So if you open pipe 0 for reading, and then **startListening()**, it will overwrite the writing pipe. Ergo, do an **openWritingPipe()** again before **write()**.

Todo:
Enforce the restriction that pipes 1-5 must share the top 32 bits

Parameters:

number	Which pipe# to open, 0-5.
address	The 40-bit address of the pipe to open.

void RF24::openWritingPipe (uint64_t address)

Open a pipe for writing.

Only one pipe can be open at once, but you can change the pipe you'll listen to. Do not call this while actively listening. Remember to **stopListening()** first.

Addresses are 40-bit hex values, e.g.:

```
openWritingPipe(0xF0F0F0F0F0);
```

Parameters:

address The 40-bit address of the pipe to open. This can be any value whatsoever, as long as you are the only one writing to it and only one other radio is listening to it. Coordinate these pipe addresses amongst nodes on the network.

void RF24::powerDown (void)

Enter low-power mode.

To return to normal power mode, either **write()** some data or startListening, or **powerUp()**.

void RF24::powerUp (void)

Leave low-power mode - making radio more responsive.

To return to low power mode, call **powerDown()**.

void RF24::print_address_register (prog_char * name, uint8_t reg, uint8_t qty = 1) [protected]

Print the name and value of a 40-bit address register to stdout.

Optionally it can print some quantity of successive registers on the same line. This is useful for printing a group of related registers on one line.

Parameters:

name Name of the register

reg Which register. Use constants from **nRF24L01.h**

qty How many successive registers to print

void RF24::print_byte_register (prog_char * name, uint8_t reg, uint8_t qty = 1) [protected]

Print the name and value of an 8-bit register to stdout.

Optionally it can print some quantity of successive registers on the same line. This is useful for printing a group of related registers on one line.

Parameters:

name Name of the register

reg Which register. Use constants from **nRF24L01.h**

qty How many successive registers to print

void RF24::print_observe_tx (uint8_t value) [protected]

Decode and print the given 'observe_tx' value to stdout.

Parameters:

value The observe_tx value to print

Warning:
Does nothing if stdout is not defined. See fdevopen in stdio.h

void RF24::print_status (uint8_t status) [protected]

Decode and print the given status to stdout.

Parameters:

status Status value to print

Warning:
Does nothing if stdout is not defined. See fdevopen in stdio.h

void RF24::printDetails (void)

Print a giant block of debugging information to stdout.

Warning:
Does nothing if stdout is not defined. See fdevopen in stdio.h

bool RF24::read (void * buf, uint8_t len)

Read the payload.

Return the last payload received

The size of data read is the fixed payload size, see **getPayloadSize()**

Note:
I specifically chose 'void*' as a data type to make it easier for beginners to use. No casting needed.

Parameters:

buf Pointer to a buffer where the data should be written

len Maximum number of bytes to read into the buffer

Returns:
True if the payload was delivered successfully false if not

uint8_t RF24::read_payload (void * buf, uint8_t len) [protected]

Read the receive payload.

The size of data read is the fixed payload size, see **getPayloadSize()**

Parameters:

buf Where to put the data

len Maximum number of bytes to read

Returns:

Current value of status register

---

uint8_t RF24::read_register (uint8_t reg) [protected]

Read single byte from a register.

Parameters:

reg Which register. Use constants from **nRF24L01.h**

Returns:

Current value of register `reg`

---

uint8_t RF24::read_register (uint8_t reg,
uint8_t * buf,
uint8_t len
) [protected]

Read a chunk of data in from a register.

Parameters:

reg Which register. Use constants from **nRF24L01.h**

buf Where to put the data

len How many bytes of data to transfer

Returns:

Current value of status register

---

void RF24::setAutoAck (bool enable)

Enable or disable auto-acknowlede packets.

This is enabled by default, so it's only needed if you want to turn it off for some reason.

Parameters:

enable Whether to enable (true) or disable (false) auto-acks

---

void RF24::setAutoAck (uint8_t pipe,
bool enable
)

Enable or disable auto-acknowlede packets on a per pipeline basis.

AA is enabled by default, so it's only needed if you want to turn it off/on for some reason on a per pipeline basis.

Parameters:

pipe Which pipeline to modify

enable Whether to enable (true) or disable (false) auto-acks

void RF24::setChannel (uint8_t channel)

Set RF communication channel.

Parameters:

channel Which RF channel to communicate on, 0-127

void RF24::setCRCLength (rf24_crclength_e length)

Set the CRC length.

Parameters:

length RF24_CRC_8 for 8-bit or RF24_CRC_16 for 16-bit

bool RF24::setDataRate (rf24_datarate_e speed)

Set the transmission data rate.

Warning:
setting RF24_250KBPS will fail for non-plus units

Parameters:

speed RF24_250KBPS for 250kbs, RF24_1MBPS for 1Mbps, or RF24_2MBPS for 2Mbps

Returns:
true if the change was successful

void RF24::setPALevel (rf24_pa_dbm_e level)

Set Power Amplifier (PA) level to one of four levels.

Relative mnemonics have been used to allow for future PA level changes. According to 6.5 of the nRF24L01+ specification sheet, they translate to: RF24_PA_MIN=-18dBm, RF24_PA_LOW=-12dBm, RF24_PA_MED=-6dBM, and RF24_PA_HIGH=0dBm.

Parameters:

level Desired PA level.

void RF24::setPayloadSize (uint8_t size)

Set Static Payload Size.

This implementation uses a pre-stablished fixed payload size for all transmissions. If this method is never called, the driver will always transmit the maximum payload size (32 bytes), no matter how much was sent to **write()**.

Todo:
Implement variable-sized payloads feature

Parameters:

size The number of bytes in the payload

void RF24::setRetries (uint8_t delay,
uint8_t count
)

Set the number and delay of retries upon failed submit.

Parameters:

delay How long to wait between each retry, in multiples of 250us, max is 15. 0 means 250us, 15 means 4000us.

count How many retries before giving up, max 15

void RF24::startListening (void)

Start listening on the pipes opened for reading.

Be sure to call **openReadingPipe()** first. Do not call **write()** while in this mode, without first calling **stopListening()**. Call isAvailable() to check for incoming traffic, and **read()** to get it.

void RF24::startWrite (const void * buf,
uint8_t len
)

Non-blocking write to the open writing pipe.

Just like **write()**, but it returns immediately. To find out what happened to the send, catch the IRQ and then call **whatHappened()**.

See also:

write()
whatHappened()

Parameters:

buf Pointer to the data to be sent

len Number of bytes to be sent

Returns:

True if the payload was delivered successfully false if not

void RF24::stopListening (void)

Stop listening for incoming messages.

Do this before calling **write()**.

bool RF24::testCarrier (void)

Test whether there was a carrier on the line for the previous listening period.

Useful to check for interference on the current channel.

Returns:
true if was carrier, false if not

bool RF24::testRPD (void)

Test whether a signal (carrier or otherwise) greater than or equal to -64dBm is present on the channel.

Valid only on nRF24L01P (+) hardware. On nRF24L01, use **testCarrier()**.

Useful to check for interference on the current channel and channel hopping strategies.

Returns:
true if signal => -64dBm, false if not

void RF24::toggle_features (void) [protected]

Turn on or off the special features of the chip.

The chip has certain 'features' which are only available when the 'features' are enabled. See the datasheet for details.

void RF24::whatHappened (bool & tx_ok,
bool & tx_fail,
bool & rx_ready
)

Call this when you get an interrupt to find out why.

Tells you what caused the interrupt, and clears the state of interrupts.

Parameters:

[out]	**tx_ok**	The send was successful (TX_DS)
[out]	**tx_fail**	The send failed, too many retries (MAX_RT)
[out]	**rx_ready**	There is a message waiting to be read (RX_DS)

bool RF24::write (const void * buf,
uint8_t len
)

Write to the open writing pipe.

Be sure to call **openWritingPipe()** first to set the destination of where to write to.

This blocks until the message is successfully acknowledged by the receiver or the timeout/retransmit maxima are reached. In the current configuration, the max delay here is 60ms.

The maximum size of data written is the fixed payload size, see **getPayloadSize()**. However, you can write less, and the remainder will just be filled with zeroes.

Parameters:

buf	Pointer to the data to be sent
len	Number of bytes to be sent

Returns:
True if the payload was delivered successfully false if not

---

uint8_t RF24::write_payload (const void * buf,
uint8_t len
) [protected]

Write the transmit payload.

The size of data written is the fixed payload size, see **getPayloadSize()**

Parameters:
buf Where to get the data
len Number of bytes to be sent

Returns:
Current value of status register

---

uint8_t RF24::write_register (uint8_t reg,
const uint8_t * buf,
uint8_t len
) [protected]

Write a chunk of data to a register.

Parameters:
reg Which register. Use constants from **nRF24L01.h**
buf Where to get the data
len How many bytes of data to transfer

Returns:
Current value of status register

---

uint8_t RF24::write_register (uint8_t reg,
uint8_t value
) [protected]

Write a single byte to a register.

Parameters:
reg Which register. Use constants from **nRF24L01.h**
value The new value to write

Returns:
Current value of status register

---

void RF24::writeAckPayload (uint8_t pipe,
const void * buf,
uint8_t len
)

Write an ack payload for the specified pipe.

The next time a message is received on `pipe`, the data in `buf` will be sent back in the acknowledgement.

Warning:
According to the data sheet, only three of these can be pending at any time. I have not tested this.

Parameters:

pipe	Which pipe# (typically 1-5) will get this response.
buf	Pointer to data that is sent
len	Length of the data to send, up to 32 bytes max. Not affected by the static payload set by **setPayloadSize()**.

The documentation for this class was generated from the following file:

- **RF24.h**