# Подключение модулей связи 2,4ГГц на базе чипов nRF24L01+ к микроконтроллеру



Автор: Погребняк Дмитрий
Самара, 2015.

Микрочипы nRF24L01+ для радиосвязи от норвежской компании Nordic Semiconductor приобрели значительную популярность.

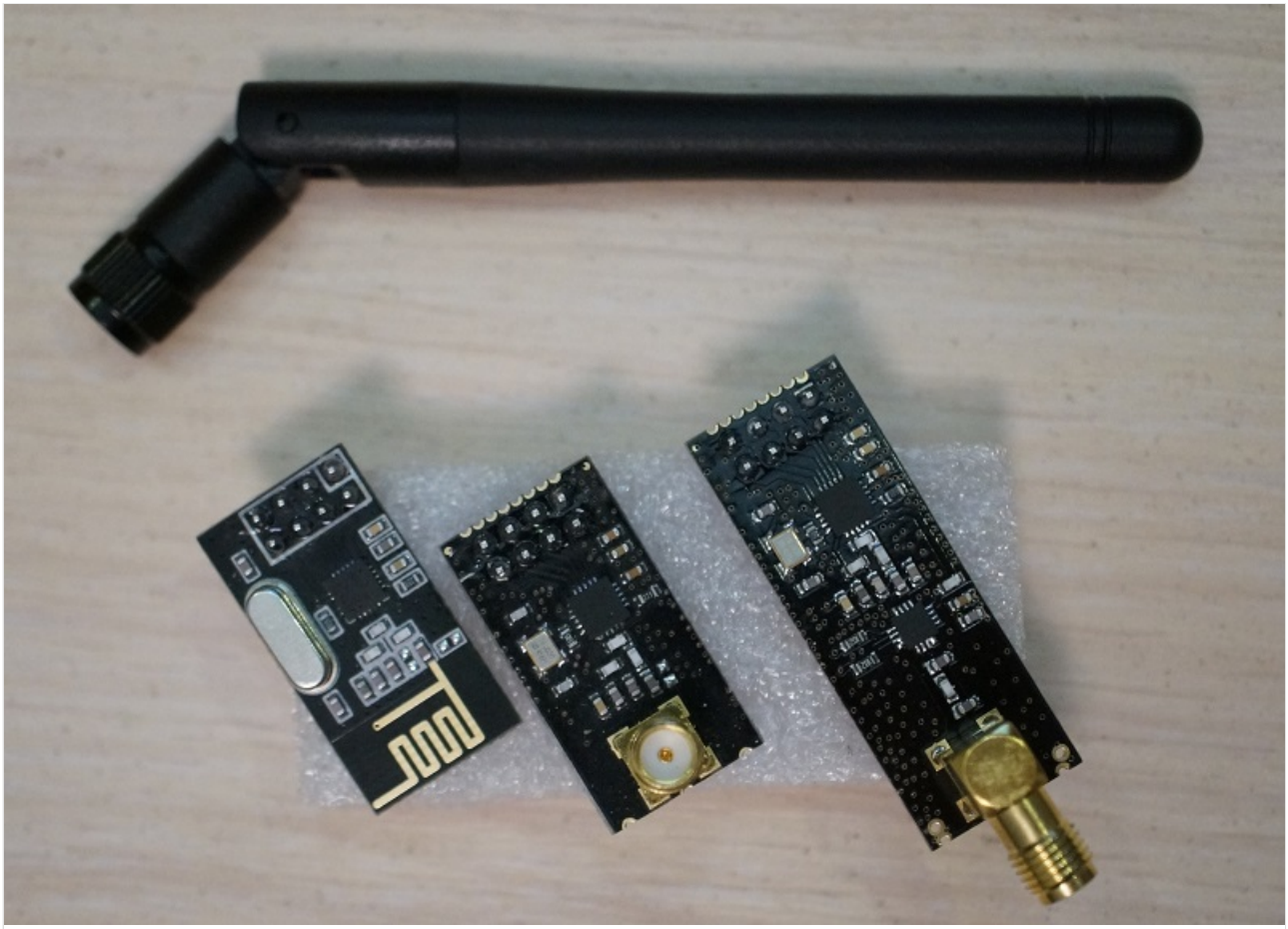
Они позволяют обеспечить коммуникацию между двумя микроконтроллерными устройствами на частоте 2,4ГГц. Чипы осуществляют обмен на скорости до 2Мбит в секунду и берут на себя такие функции как формирование пакетов, вычисление контрольной суммы, подтверждение приёма и даже автоматическую повторную передачу. При этом различные энергосберегающие режимы позволяют экономно расходовать энергию, при работе от батарей. Всё это, вкупе с небольшой стоимостью чипов и модулей на их основе и позволило им снискать популярность.

Радиолюбителям они известны в составе разнообразных радио-модулей на их основе, выпуск которых налажен в Китае. На ebay для заказа доступны разные варианты, среди них наиболее распространённый вариант, это радио-модуль 15x29мм с кварцевым резонатором и антенной,

выполненной в виде дорожки на плате. Такие модули продаются на ebay по цене около 2 долларов за пару, включая стоимость доставки в Россию.

Такие модули обеспечивают «комнатную» связь на расстоянии до 10 метров.

Более продвинутые варианты модулей имеют выход на sma антенну, которая значительно улучшает параметры связи, а также варианты, включающие в состав малошумящий усилитель и усилитель мощности, обеспечивающий выходную мощность на уровне более 20dBm, с заявленной дальностью связи до 1100 метров. Такие модули уже продаются в комплекте с антенной примерно по 5-6 долларов за штуку. Эти модули имеют одинаковую разводку разъёма для подключения, и поэтому полностью взаимозаменяемы.



Слева-направо: модуль 1мВт с антенной в виде дорожки меди на плате, модуль 1мВт с sma-выходом на антенну, модуль 100мВт с sma-выходом на антенну и чипом усиления. Сверху - sma-антенна 2,4ГГц 2dBi

В этой статье речь пойдёт о подключении таких модулей к микроконтроллерам AVR.

## Правовые нормы использования радиочастот

Диапазон частот 2400 – 2500 МГц относится к т.н. международному «промышленному научному и медицинскому диапазону радиочастот» (ISM band).

Использование радиоэлектронных средств в России регулируется Правилами регистрации радиоэлектронных средств и высокочастотных устройств. Раздел «Изъятия из перечня радиоэлектронных средств и высокочастотных устройств, подлежащих регистрации» определяет перечень радиочастотного оборудования, не подлежащего регистрации. В частности, пункт 24 разрешает использование частот 2400-2483,5 Мгц в устройствах малого радиуса действия, используемых «в сетях беспроводной передачи данных в полосе радиочастот 2400 - 2483,5 МГц, с максимальной эквивалентной изотропно излучаемой мощностью передатчика не более 100 мВт при использовании прямого расширения спектра и других отличных от псевдослучайной перестройки рабочей частоты видов модуляции ... при максимальной спектральной плотности эквивалентной изотропно излучаемой мощности 2 мВт/МГц».

Т.е. допускается использование передатчиков nRF24L01 на каналах 0...83 (2400...2483 МГц) без усилителей мощности сигнала.

Условия использования частот в диапазоне от 2500Мгц изложены в п.13:

«Пользовательское (оконечное) оборудование передающее, включающее в себя приемное устройство, работающее в полосах радиочастот 2300 - 2400 МГц, 2500 - 2690 МГц, 3400 - 3450 МГц и 3500 - 3550 МГц, с допустимой мощностью излучения передатчика не более 1 Вт, в том числе встроенное либо входящее в состав других устройств»

Т.е. допускается использование модулей nRF24L01 на каналах 100...125 (2500...2525 МГц) с усилителем мощности.

Как видно, документ не описывает использование частот в диапазоне 2484 – 2499 МГц. Т.е. использовать каналы 84...99 (2484...2499 МГц) передатчиков nRF24L01 не допускается.

# Основные сведения

Спецификацию на чип можно скачать на официальном сайте.

Чип nRF24L01+ работает при напряжении питания от 1,9 до 3,6 Вольт. Его входы толерантны к напряжению до 5,25 Вольт. Это значит, что его можно подключать к микроконтроллеру, работающему на напряжении 5 Вольт, однако в этом случае, допустимые напряжения питания чипа ограничены диапазоном от 2,7 до 3,3 Вольт.

Чип взаимодействует с микроконтроллером по 4-хпроводной шине SPI. Кроме этого используются ещё две линии: входная линия CE, сигнал высокого уровня переводит чип из ждущего режима в режим приёма, или передачи. И выходная линия IRQ, на которой устанавливается низкий уровень, когда чип принимает пакет данных, или завершает передачу.

Максимальная скорость передачи цифровых данных в радиоэфир - 2Мбит/с. Может быть установлена пониженная скорость 1Мбит/с, обеспечивающая лучшее качество связи, также доступна низкая скорость 250кбит/с для совместимости с другими версиями чипов.

> Хотя максимальная скорость передачи в радио-эфир составляет 2Мбит/с, при включенном режиме подтверждения приёма за один сеанс обмена может быть передано 32 байта полезных данных. Один сеанс обмена включает в себя 2 калибровки схемы PLL (при переходе в режим приёма и при переходе в режим передачи), каждая по 130мкс, и передачу сопутствующих данных, составляющих структуру пакета, как в прямом так и обратном направлении (в виде подтверждения).
>
> При минимальной длине поля адреса 3 байта и поля CRC 1 байт, длина вспомогательных данных при отправке и пакета подтверждения составит 49 бит. Значит, одна передача пакета с 32 байтами данных с подтверждением доставки займёт минимум 437мкс при скорости 2Мбит/с и 614мкс при скорости 1Мбит/с. Значит, теоретическая максимальная возможная скорость передачи составит 73 226.5 байта в секунду при скорости обмена 2Мбит/с, и 52 117.3 байта в секунду при скорости обмена 1Мбит/с.
>
> При отключении автоматического контроля доставки количество передаваемых полезных данных за единицу времени может быть значительно повышено. В этом случае, однако, следует помнить что хотя бы один раз в 4мс должна происходить перекалибровка PLL передатчика (TX settling), которая занимает 130мкс.
>
> Теоретическая предельная скорость передачи данных при условии отключенного автоподтверждения и контроля доставки составит 203 016 байта в секунду на скорости 2Мбит/с и 101 508 - при скорости 1Мбит/с.

Максимальная мощность передатчика самого чипа (без внешних усилителей) – 0dBm (1мВт); чувствительность приёмника -82dBm при скорости 2Мбит/с, и -85dBm при скорости 1Мбит/с.

Чип работает на частотах в диапазоне 2400-2525 МГц. Конкретная частота из этого диапазона с шагом в 1МГц задаётся настройками.

Энергопотребление чипа составляет около 0,9мкА в режиме power-down, до 11,3мА при передаче, и до 13,5мА при приёме, и 26 и 320мкА в режимах готовности (standby-I и standby-II). Использование автоматизации в передаче пакета позволяет минимизировать время нахождения в энергоёмких режимах, тем самым значительно снижая энергопотребление устройств работающих в режиме передатчика.

Чип может работать как в режиме приёмника, так и в режиме передатчика, но в каждый момент времени может находиться только в одном из этих двух состояний. Формирование пакетов и контроль передачи происходит автоматически. Максимальная длина полезного содержимого пакета – 32 байта.

И приёмник и передатчик используют очереди FIFO на три элемента каждая. В каждый элемент очередь помещается всё содержимое пакета сразу, и извлекаться из очереди должно также за одну операцию чтения.

Приёмник поддерживает получение данных по шести каналам, в зависимости от переданного адреса. Каналы 1-5 различаются только младшим байтом адреса. Адрес канала 0 может быть настроен независимо, но при передаче этот канал используется для получения подтверждений приёма.

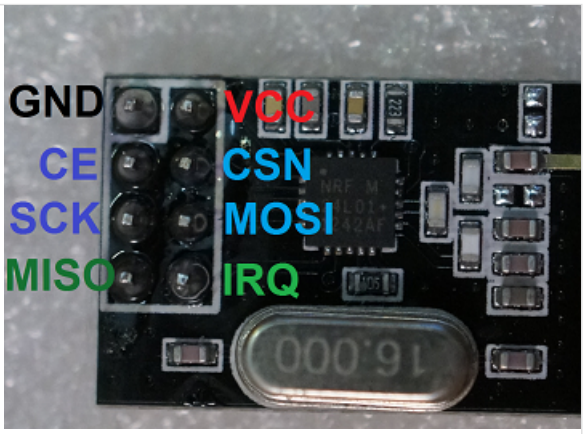
## Назначение выводов

### GND

Общий, "земля".

### VCC

Питание, от 1.9 до 3.6 Вольт. Однако, если напряжение на линиях MOSI, SCK, CE, SN более 3.6 Вольт, то напряжение питания должно быть в пределах 2.7 - 3.3 Вольт.



Назначение выводов

### CE

Входной, активный - высокий уровень. Наличие высокого уровня на этом входе активирует режим приёма. В режиме передачи импульс не менее 10мкс начинает передачу.

### CSN

Входной, активный - низкий уровень. Низкий уровень устанавливает границы сеанса обмена по шине SPI.

### SCK

Входной. Тактовый импульс для обмена по шине SPI. Данные на линиях MOSI и MISO считываются противоположной стороной по нарастающему фронту на линии SCK, и выставляются по спаду.

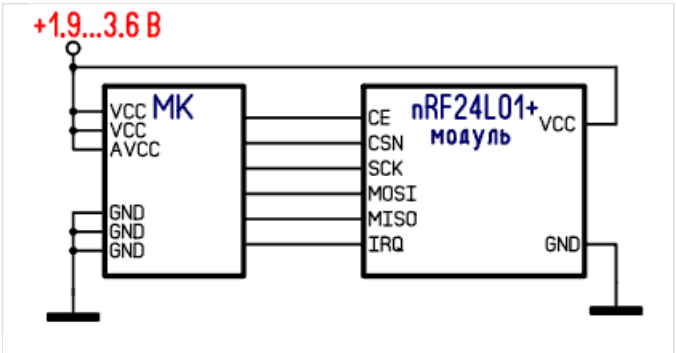
Раздельное подключение питания

### MOSI

Входной. Линия SPI, передающая данные от микроконтроллера к радио-модулю.

### MISO

Выходной. Линия SPI, передающая данные от радио-модуля к микроконтроллеру.




### IRQ

Выходной, активный - низкий уровень. При наличии одного из прерываний устанавливается

низкий уровень на этой линии.

Совмещённое подключение питания

## Команды

Сеанс обмена начинается с установки низкого уровня на линии CSN. Первый переданный после этого от микроконтроллера байт является командой. Одновременно с байтом команды от микроконтроллера, к микроконтроллеру по линии MISO передаётся байт статуса.

**Перечень команд:**

| Наименование | Двоичный код | 16-чный код | Размер данных | Описание |
|---|---|---|---|---|
| R_REGISTER | 000n nnnn | 0x00 + n | 1-5 (приём) | Прочитать регистр n |
| W_REGISTER | 001n nnnn | 0x20 + n | 1-5 (передача) | Записать регистр n |
| R_RX_PAYLOAD | 0110 0001 | 0x61 | 1-32 (приём) | Принять данные из верхнего слота очереди приёмника. После чтения данные из очереди удаляются |
| W_TX_PAYLOAD | 1010 0000 | 0xA0 | 1-32 (передача) | Записать в очередь передатчика данные для отправки |
| FLUSH_TX | 1110 0001 | 0xE1 | 0 | Сбросить очередь передатчика |
| FLUSH_RX | 1110 0010 | 0xE2 | 0 | Сбросить очередь приёмника |
| REUSE_TX_PL | 1110 0011 | 0xE3 | 0 | Использовать повторно последний переданный пакет |
| R_RX_PL_WID | 0110 0000 | 0x60 | 1 (приём) | Прочитать размер данных принятого пакета в начале очереди приёмника. Значение больше 32, означает ошибку приёма, в таком случае пакет должен быть удалён командой FLUSH_RX |
| W_ACK_PAYLOAD | 1010 1ppp | 0xA8 + p | 1-32 (передача) | Записать данные для отправки с пакетом подтверждения по каналу p. При этом бит EN_ACK_PAY в регистре FEATURE должен быть установлен |
| W_TX_PAYLOAD_NOACK | 1011 0000 | 0xB0 | 1-32 (передача) | Записать в очередь передатчика данные для отправки, для которых не требуется подтверждение приёма. |
| NOP | 1111 1111 | 0xFF | 0 | Нет операции. Может быть использовано для чтения регистра статуса |

Все байты данных, сопутствующие команде должны быть переданы/получены в течение одного сеанса обмена.

Сеанс обмена завершается установкой высокого уровня на линии CSN

## Регистры

### 0x00 CONFIG

Регистр настроек.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | MASK_RX_DR | MASK_TX_DS | MASK_MAX_RT | EN_CRC | CRCO | PWR_UP | PRIM_RX |

**MASK_RX_DR**, **MASK_TX_DS**, **MASK_MAX_RT** - маскировка источников прерываний. При установке одного из этих бит в единицу, соответствующее событие и установка соответствующего бита в регистре STATUS не будет генерировать сигнал прерывания на линии IRQ.

**EN_CRC** - включает расчёт контрольной суммы (CRC). Если включено автоподтверждение приёма путём установки хотя бы одного бита в регистре EN_AA, то значение этого бита устанавливается в единицу автоматически.

**CRCO** - Определяет размер поля контрольной суммы (CRC): 0 - 1 байт; 1 - 2 байта

**PWR_UP** - Включает питание. Если этот бит равен нулю, то чип находится в режиме отключения, и потребляет около 0,9мкА, в таком режиме радиообмен невозможен.

После включения питания до начала работы в режиме приёмника или передатчика (т.е. до выставления высокого уровня на линии CE) необходимо выдержать паузу, достаточную для выхода осциллятора на режим, типично 1.5мс (подробнее см. в разделе "Необходимые временные задержки").

**PRIM_RX** - Выбор режима: 0 - PTX (передатчик) 1 - PRX (приёмник).

Если переход в режим передатчика осуществляется сразу после завершения приёма пакета (появления прерывания RX_DR), и автоматическое подтверждение приёма включено, то необходимо обеспечить паузу с момента появления прерывания до перевода в режим PTX, во время которой чип отправит пакет подтверждения, типично не более 203 мкс (подробнее см. в разделе "Необходимые временные задержки").

### 0x01 EN_AA

Включает автоподтверждение приёма.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | ENAA_P5 | ENAA_P4 | ENAA_P3 | ENAA_P2 | ENAA_P1 | ENAA_P0 |

**ENAA_Px** - установка бита включает автоматическую отправку подтверждения приёма данных по соответствующему каналу.

### 0x02 EN_RXADDR

Выбирает активные каналы приёмника.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | ERX_P5 | ERX_P4 | ERX_P3 | ERX_P2 | ERX_P1 | ERX_P0 |

**ERX_Px** - включает приём данных по соответствующему каналу.

При использовании устройства в качестве передатчика с включенной функцией автоподтверждения, ответ от удалённого устройства принимается на канале 0. Поэтому бит ERX_P0 должен быть установлен в 1, для использования передачи с автоподтвреждением.

### 0x03 SETUP_AW

Задаёт длину поля адреса.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | AW | |

**AW** - два бита, задающие длину поля адреса: *01* - 3 байта; *10* - 4 байта; *11* - 5 байт.

Эта настройка влияет на передачу и приём пакетов по всем каналам. Данная настройка у передающего и принимающего устройств должна быть идентичной.

### 0x04 SETUP_RETR

Настройка параметров автоматического повтора отправки.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ARD | | | | ARC | | | |

**ARD** - четыре бита, задающие время ожидания подтверждения приёма перед повторной отправкой: *0000* - 250мкс; *0001* - 500мкс; *0010* - 750мкс; ... ; *1111* - 4000мкс;

Значение поля ARD необходимо выбирать в зависимости от скорости обмена и количества данных, передаваемых с пакетом подтверждения. Если данные с пакетом подтверждения не передаются, то для скоростей обмена 1 и 2 Мбит/с достаточное время ожидания 250мкс. Если вместе с пакетом подтверждения на скорости 1Мбит/с передаётся более 5 байт данных, или более 15 на скорости 2Мбит/с, или скорость обмена выбрана 250кбит/с, то необходимо выбрать паузу 500мкс.

**ARC** - четыре бита, задающие количество автоматических повторов отправки. *0000* - автоповтор отключен; *0001* - возможен однократный повтор; ... ; *1111* - возможно до 15 повторов.

### 0x05 RF_CH

Регистр задаёт номер радиоканала - частоту несущей с шагом 1Мгц. Радиочастота несущей вычисляется по формуле 2400 + RF_CH МГц.

Допустимые значения от 0 до 125. При обмене на скорости 2Мбит/с, частота должна отличатся от частоты используемой другими устройствами минимум на 2 МГц.

### 0x06 RF_SETUP

Задаёт настройки радиоканала.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CONT_WAVE | - | RF_DR_LOW | PLL_LOCK | RF_DR_HIGH | RF_PWR | | - |

**CONT_WAVE** - Непрерывная передача несущей. Предназначено для тестирования

**RF_DR_LOW** - Включает низкую скорость передачи 250кбит/с. При этом бит RF_DR_HIGH должен быть 0

**PLL_LOCK** - предназначено для тестирования.

**RF_DR_HIGH** - Выбор скорости обмена (при значении бита RF_DR_LOW = 0): *0* - 1Мбит/с; *1* - 2Мбит/с

**RF_PWR** - 2 бита, задающих мощность передатчика: *00* - -18dBm; *01* - -12dBm; *10* - -6dBm; *11* - 0dBm

### 0x07 STATUS

Регистр статуса. Его значение также передаётся на линии MISO одновременно с передачей байта команды по интерфейсу SPI.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | RX_DR | TX_DS | MAX_RT | RX_P_NO | | | TX_FULL |

**RX_DR** - Прерывание по получению пакета. Бит устанавливается в единицу, когда усройство в режиме приняло адресованый ему пакет с совпавшей контрольной суммой. Бит сбрасывается путём записи в него значения 1.

Принятый пакет доступен в очереди приёмника, и может быть прочитан командой R_RX_PAYLOAD, либо удалён командой FLUSH_RX.

**TX_DS** - Прерывание по успешной отправке пакета. Бит устанавливается в единицу, когда устройство в режиме передатчика успешно отправило пакет и, если включено автоподтверждение, приняло подтверждение получения. После успешной отправки пакет удаляется из очереди передатчика. Бит сбрасывается путём записи в него значения 1.

**MAX_RT** - Прерывание по превышению числа попыток повторной отправки. Бит устанавливается в единицу, когда устройство в режиме передатчика предприняло заданное в регистре SETUP_RETR количество попыток отправки, но так и не получило подтверждение от удалённого устройства. Передаваемый пакет остаётся в очереди передатчика. Для удаления его можно воспользоваться командой FLUSH_TX. Дальнейшая коммуникация невозможна, пока этот бит установлен. Бит сбрасывается путём записи в него значения 1.

Пока любой из бит **RX_DR**, **TX_DS**, **MAX_RT** установлен в единицу и соответствующее ему прерывание не запрещено (т.е. соответствующие из бит **MASK_RX_DR**, **MASK_TX_DS**, **MASK_MAX_RT** в регистре CONFIG равны нулю), то на выходе IRQ чипа устанавливается низкий уровень. Для сброса значений этих бит, необходимо записать регистр STATUS с этими битами, установленными в 1.

**RX_P_NO** - три бита кодирующие номер канала, пакет по которому доступен в начале очереди приёмника для чтения. Значения *000* - *101* кодируют каналы с 0 по 5, соответственно, значение *111* указывает, что очередь приёмника пуста.

**TX_FULL** - значение 1 этого бита показывает, что в очереди передатчика нет свободных слотов.

### 0x08 OBSERVE_TX

Регистр контроля передатчика

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PLOS_CNT | | | | ARC_CNT | | | |

**PLOS_CNT** - четыре бита, значение которых увеличивается, вплоть до достижения 15, при каждой отправке, на которую не получено ожидаемое подтверждение. Значение сбрасывается при записи в регистр RF_CH.

**ARC_CNT** - четыре бита, возвращающие количество предпринятых повторов отправки при передаче последнего пакета. Сбрасывается в 0, когда начинается отправка очередного пакета.

### 0x09 RPD

Оценка мощности принимаемого сигнала

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | RPD |

**RPD** - младший бит принимает значение 1, если чип находится в режиме приёмника, и уровень принимаемого сигнала превышает -64dBm

### 0x0A RX_ADDR_P0

40-битный (5 байт) регистр, используемый для указания адреса канала 0 приёмника. Этот канал используется для приёма автоподтверждений в режиме передатчика. Автоподтверждения высылаются принимающей стороной с указанием собственного адреса. Поэтому значение этого регистра должно соответствовать значению регистра TX_ADDR для корректной работы в режиме передатчика.

Реальная используемая ширина адреса задаётся в регистре SETUP_AW.

Значение регистра записывается и читается, начиная с младших байт. Если записано менее 5 байт, то старшие байты остаются неизменными.

Значение регистра по умолчанию: *0xE7E7E7E7E7*

### 0x0B RX_ADDR_P1

40-битный (5 байт) регистр, используемый для указания адреса канала 1 приёмника. Старшие 4 байта этого регистра являются общими для адресов на каналах 1 - 5.

Реальная используемая ширина адреса задаётся в регистре SETUP_AW.

Значение регистра записывается и читается, начиная с младших байт. Если записано менее 5 байт, то старшие байты остаются неизменными.

Значение регистра по умолчанию: *0xC2C2C2C2C2*

### 0x0C-0x0F RX_ADDR_P2 - RX_ADDR_P5

8-битные регистры, задающие значения младшего байта адреса для каналов 2-5. Значения старших 32 бит берутся из регистра RX_ADDR_P1.

Значение регистров по умолчанию: *0xC3*, *0xC4*, *0xC5*, *0xC6*, соответственно.

### 0x10 TX_ADDR

40-битный (5 байт) регистр, используемый в режиме передатчика в качестве адреса удалённого устройства. При включенном режиме автоподтверждения, удалённое устройство ответит подтверждением с указанием своего же адреса. Это подтверждение принимается на канале 0, поэтому для успешной передачи, значение регистра RX_ADDR_P0 должно быть идентично этому.

Реальная используемая ширина адреса задаётся в регистре SETUP_AW.

Значение регистра записывается и читается, начиная с младших байт. Если записано менее 5 байт, то старшие байты остаются неизменными.

Значение регистра по умолчанию: *0xE7E7E7E7E7*

### 0x11-0x16 RX_PW_P0 - RX_PW_P5

8-битные регистры, задающие размер данных, принимаемых по каналам, соответственно 0-5, если не включена поддержка произвольной длины пакетов в регистрах DYNPD и FEATURE. Значение *0* указывает что канал не используется. Допустимы значения длины от *1* до *32*.

### 0x17 FIFO_STATUS

Состояние очередей FIFO приёмника и передатчика

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | TX_REUSE | TX_FULL | TX_EMPTY | - | - | RX_FULL | RX_EMPTY |

**TX_REUSE** Признак готовности последнего пакета для повтрной отправки. Устанавливается командой REUSE_TX_PL.

**TX_FULL** Флаг переполнения FIFO очереди передатчика: *0* - есть свободное место в очереди; *1* - очередь переполнена.

**TX_EMPTY** Флаг освобождения FIFO очереди передатчика: *0* - в очереди есть данные; *1* - очередь пуста.

**RX_FULL** Флаг переполнения FIFO очереди приёмника: *0* - есть свободное место в очереди; *1* - очередь переполнена.

**RX_EMPTY** - Флаг освобождения FIFO очереди приёмника: *0* - в очереди есть данные; *1* - очередь пуста.

### 0x1C DYNPD

Разрешение использования пакетов произвольной длины.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | DPL_P5 | DPL_P4 | DPL_P3 | DPL_P2 | DPL_P1 | DPL_P0 |

**DPL_Px** разрешает приём пакетов произвольной длины по соответствующему каналу. При этом такая опция должна быть включена установленным битом **EN_DPL** в регистре FEATURE, а также включено автоподтверждение установкой соответствующего бита **ENAA_Px** в регистре EN_AA

В случае если бит принимает значение 0, то размер данных в принимаемых пакетах должен быть равен значению соответствующего регистра RX_PW_Px

### 0x1D FEATURE

Регистр опций

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | EN_DPL | EN_ACK_PAY | EN_DYN_ACK |

**EN_DPL** включает поддержку приёма и передачи пакетов с размером поля данных произвольной длины.

В этом случае приём пакетов произвольной длины по каналам должен быть разрешён в регистре DYNPD и включено автоподтверждение в регистре EN_AA.

Если опция отключена, то размер принимаемых данных задаётся значением регистров RX_PW_P0 - RX_PW_P5.

При передаче также передаётся поле, указывающее длительность пакета. Длина передаваемого пакета определяется размером записанных командой W_TX_PAYLOAD данных и она должна соответствовать настройкам принимающей стороны.

**EN_ACK_PAY** включает поддержку передачи данных с пакетами подтверждения.

После включения этой опции, командой W_ACK_PAYLOAD в очередь передатчика могут быть помещены данные, которые будут отправлены вместе с пакетом подтверждения приёма.

**EN_DYN_ACK** разрешает передавать пакеты, не требующие подтверждения приёма.

После включения этой опции, командой W_TX_PAYLOAD_NOACK в очередь передатчика могут быть помещены данные, на которые удалённой стороне не требуется присылать подтверждение о приёме.

## Структура пакета

Чипы nRF24L01+ автоматически производят сборку пакета при передаче, контроль адреса, проверку контрольной суммы и разбор пакета при приёме.

Поэтому передача и приёма данных осуществляется без оглядки на истинную структуру пакета, передаваемого в эфир, чья структура выглядит следующим образом:

| Преамбула | Адрес | Управляющее поле | Данные | CRC |
|---|---|---|---|---|
| 1 байт | 3-5 байт | 9 бит | 0-32 байта | 1-2 байта |

Значения всех полей передаются вначале старшим битом.

### Преамбула

Последовательность бит 01010101, или 10101010, служащая для синхронизации приёмника. Если старший бит в адресе 1, то преамбула 10101010, иначе 01010101

### Адрес

Длина поля адреса задаётся значением бит **AW** в регистре SETUP_AW. Чтобы исключить ложные обнаружения адреса (что может привести к пропуску реального адреса), желательно чтобы адрес удовлетворял следующим условиям:

- Старшие биты адреса не должны быть чередующимися значениями нулей и единиц, поскольку в таком случае часть преамбулы может быть спутана с адресом.
- Адрес должен содержать несколько переходов из нуля в единицу и обратно, иначе за адрес могут быть приняты помехи.

Пакеты подтверждения приёма высылаются приёмником с указанием собственного адреса в этом поле. Поэтому передающая сторона должна быть настроена на получение пакетов по этому же адресу на канале 0, т.е. значение регистра RX_ADDR_P0 должно быть равно значению регистра TX_ADDR.

### Управляющее поле

имеет следующую структуру

| Длина данных | PID | NO_ACK |
|---|---|---|
| 6 бит | 2 бита | 1 бит |

**Длина данных** - размер поля "Данные" в пакете. Если опция данных произвольной длины отключена, принимает значение 33 (100001), в этом случае длина данных на принимающей стороне определяется значением соответствующего регистра RX_PW_Px. Значения в диапазоне 1-32 кодируют размер данных в режиме произвольной длины, значение 0 указывает на отсутствие данных и используется в пакетах подтверждения. Режим произвольной длины должен быть включен у передающей стороны для канала 0, чтобы принимать пакеты подтверждения.

**PID** - двухбитное поле, значение которого циклически увеличивается на 1 при отправке нового пакета. В случае если принимающая сторона приняла пакет, но отправленное подтверждение о приёме не дошло до отправляющей стороны, может быть предпринята повторная отправка с таким же значением PID, как при первой попытке. Если приёмник получает пакет, где поле PID равно полученному в предыдущий раз и значение поля CRC также идентично последнему полученному, то автоматически отправляется подтверждение о получении, но полученные данные считаются повтором и игнорируются, не помещаются в очередь FIFO приёмника, и прерывание TX_DS в этом случае не появляется.

**NO_ACK** - флаг указывающий получателю на то, что подтверждение получения пакета высылать не требуется. Сами пакеты подтверждения маркируются этим флагам. Также можно отправить пакет не требующий подтверждения командой W_TX_PAYLOAD_NOACK, если в регистре FEATURE установлен бит **EN_DYN_ACK**.

### Данные

Поле данных содержит, собственно, передаваемые данные. Их длина определяется количеством данных, загруженных одной из команд W_TX_PAYLOAD, W_TX_PAYLOAD_NOACK, W_ACK_PAYLOAD.

### CRC

Контрольная сумма, размер определяется значением бита **CRCO** в регистре CONFIG. Рассчитывается по полю адреса, управляющему полю и полю данных. Если при приёме пакета контрольная сумма не совпала, то пакет игнорируется, никаких действий не предпринимается.

## Краткое описание алгоритма приёма

Переключение в режим приёмника осуществляется установкой бита **PRIM_RX** в регистре CONFIG.

Соответствующие каналы приёма должны быть разрешены в регистрах EN_AA и EN_RXADDR, и их адреса настроены в регистрах RX_ADDR_Px.

Прослушивание эфира начинается с появлением на линии CE высокого уровня. Приёмник анализирует эфир и пытается выделить адресованные ему пакеты с совпадающей контрольной суммой. Когда очередной такой пакет получен, выставляется бит **RX_DR** в регистре STATUS, и на линии прерывания появляется низкий уровень. Три бита начиная с **RX_P_NO** в регистре STATUS показывают номер канала, по которому пришёл пакет. Прочитать содержимое полученного пакета можно командой R_RX_PAYLOAD. Сбросить бит **RX_DR** в регистре STATUS можно путём записи в него единицы.

## Краткое описание алгоритма передачи

Переключение в режим передатчика осуществляется записью значения 0 в бит **PRIM_RX** в регистре CONFIG. В регистры TX_ADDR и RX_ADDR_P0 должен быть загружен адрес удалённой стороны.

После этого, данные для отправки помещаются в очередь передатчика командой W_TX_PAYLOAD. Начало передачи инициализируется кратким, но не менее 10мкс импульсом на линии CE.

Если пакет передан успешно и подтверждение получено, в регистре STATUS выставляется бит **TX_DS**, если превышено допустимое количество повторов, а подтверждение передачи не получено, выставляется бит **MAX_RT**. Обе ситуации приводят к выставлению на линии IRQ низкого уровня.

Если выставлен бит **MAX_RT**, то переданный пакет остаётся в очереди передатчика, удалить его можно командой FLUSH_TX. Сбросить биты **TX_DS** и **MAX_RT** в регистре STATUS можно путём записи в них единиц. Пока бит **MAX_RT** установлен, дальнейший радиообмен невозможен.

## Необходимые временные задержки

После включения питания (установки бита **PWR_UP** в регистре CONFIG) происходит запуск осциллятора. До начала работы в режиме приёмника или передатчика (т.е. до выставления высокого уровня на линии CE) необходимо выдержать паузу, достаточную для выхода осциллятора на режим.

Необходимое для этого время пропорционально эквивалентной индуктивности используемого резонатора. Для типичных резонаторов со значением этого параметра не превышающего 30 мГн, достаточно задержки 1.5мс. При значениях эквивалентной индуктивности больше, необходимое время задержки пропорционально возрастает

| **Эквивалентная индуктивность резонатора** | **Необходимое время задержки** |
|---|---|
| 30мГн и меньше (типичный вариант) | 1.5мс |
| 60мГн | 3мс |
| 90мГн | 4.5мс |

Если включено автоматическое подтверждение приёма, то прерывание RX_DR появляется сразу по получению пакета, после этого nRF24L01+ автоматически переходит в режим передатчика, после чего в течение 130мкс происходит выравнивание частоты блока PLL, и затем осуществляется отправка пакета подтверждения.

В этом случае, если отправка ответа предполагается сразу после получения пакета (появления прерывания RX_DR), то перед сбросом бита **PRIM_RX** и переходом в режим передатчика необходимо выдержать паузу достаточную, чтобы чип успел завершить передачу пакета подтверждения. Рассчитать это время можно по формуле:

***Время_подтверждения = 130мкс + ((длина_адреса + длина_CRC + размер_данных_в_подтверждении) * 8 + 17) / скорость_обмена***

Это время отсчитывается с момента появления прерывания RX_DR. Поскольку управляющий микроконтроллер на обработку прерывания, чтение пакета данных, подготовку ответа и прочее также затратит некоторое время, реальная необходимая задержка может быть значительно меньше.

Типичное время, необходимое для отправки подтверждений не включающих в себя передаваемых данных:

| **Настройки** | **Время, мкс** |
|---|---|

| | |
|---|---|
| 1Мбит/с, адрес 3 байта, CRC 1 байт | 179 |
| 1Мбит/с, адрес 5 байт, CRC 2 байта | 203 |
| 2Мбит/с, адрес 5 байт, CRC 2 байта | 166.5 |

## Определения для языка С

```
/* Команды */

#define R_REGISTER          0x00 // + n Прочитать регистр n
#define W_REGISTER          0x20 // + n Записать регистр n
#define R_RX_PAYLOAD        0x61 // Принять данные данные из верхнего слота очереди приёмника.
#define W_TX_PAYLOAD        0xA0 // Записать в очередь передатчика данные для отправки
#define FLUSH_TX            0xE1 // Сбросить очередь передатчика
#define FLUSH_RX            0xE2 // Сбросить очередь приёмника
#define REUSE_TX_PL         0xE3 // Использовать повторно последний переданный пакет
#define R_RX_PL_WID         0x60 // Прочитать размер данных принятого пакета в начале очереди приёмника.
#define W_ACK_PAYLOAD       0xA8 // + p Записать данные для отправки с пакетом подтверждения по каналу p.
#define W_TX_PAYLOAD_NOACK  0xB0 // Записать в очередь передатчика данные, для отправки без подтверждения
#define NOP                 0xFF // Нет операции. Может быть использовано для чтения регистра статуса

/* Регистры */

#define CONFIG      0x00 // Регистр настроек
#define EN_AA       0x01 // Выбор автоподтверждения
#define EN_RXADDR   0x02 // Выбор каналов приёмника
#define SETUP_AW    0x03 // Настройка размера адреса
#define SETUP_RETR  0x04 // Настройка повторной отправки
#define RF_CH       0x05 // Номер радиоканала, на котором осуществляется работа. От 0 до 125.
#define RF_SETUP    0x06 // Настройка радиоканала
#define STATUS      0x07 // Регистр статуса.
#define OBSERVE_TX  0x08 // Количество повторов передачи и потерянных пакетов
#define RPD         0x09 // Мощность принимаемого сигнала. Если младший бит = 1, то уровень более -64dBm
#define RX_ADDR_P0  0x0A // 3-5 байт (начиная с младшего байта). Адрес канала 0 приёмника.
#define RX_ADDR_P1  0x0B // 3-5 байт (начиная с младшего байта). Адрес канала 1 приёмника.
#define RX_ADDR_P2  0x0C // Младший байт адреса канала 2 приёмника. Старшие байты из RX_ADDR_P1
#define RX_ADDR_P3  0x0D // Младший байт адреса канала 3 приёмника. Старшие байты из RX_ADDR_P1
#define RX_ADDR_P4  0x0E // Младший байт адреса канала 4 приёмника. Старшие байты из RX_ADDR_P1
#define RX_ADDR_P5  0x0F // Младший байт адреса канала 5 приёмника. Старшие байты из RX_ADDR_P1
#define TX_ADDR     0x10 // 3-5 байт (начиная с младшего байта). Адрес удалённого устройства для передачи
#define RX_PW_P0    0x11 // Размер данных при приёме по каналу 0: от 1 до 32. 0 - канал не используется.
#define RX_PW_P1    0x12 // Размер данных при приёме по каналу 1: от 1 до 32. 0 - канал не используется.
#define RX_PW_P2    0x13 // Размер данных при приёме по каналу 2: от 1 до 32. 0 - канал не используется.
#define RX_PW_P3    0x14 // Размер данных при приёме по каналу 3: от 1 до 32. 0 - канал не используется.
#define RX_PW_P4    0x15 // Размер данных при приёме по каналу 4: от 1 до 32. 0 - канал не используется.
#define RX_PW_P5    0x16 // Размер данных при приёме по каналу 5: от 1 до 32. 0 - канал не используется.
#define FIFO_STATUS 0x17 // Состояние очередей FIFO приёмника и передатчика
#define DYNPD       0x1C // Выбор каналов приёмника для которых используется произвольная длина пакетов.
#define FEATURE     0x1D // Регистр опций


/* Биты регистров */

// CONFIG
#define MASK_RX_DR  6 // Запрещает прерывание по RX_DR (получение пакета)
#define MASK_TX_DS  5 // Запрещает прерывание по TX_DS (завершение отправки пакета)
#define MASK_MAX_RT 4 // Запрещает прерывание по MAX_RT (превышение числа повторных попыток отправки)
#define EN_CRC      3 // Включает CRC
#define CRCO        2 // Размер поля CRC: 0 - 1 байт; 1 - 2 байта
#define PWR_UP      1 // Включение питания
#define PRIM_RX     0 // Выбор режима: 0 - PTX (передатчик) 1 - PRX (приёмник)

// EN_AA
#define ENAA_P5 5 // Включает автоподтверждение данных, полученных по каналу 5
#define ENAA_P4 4 // Включает автоподтверждение данных, полученных по каналу 4
#define ENAA_P3 3 // Включает автоподтверждение данных, полученных по каналу 3
#define ENAA_P2 2 // Включает автоподтверждение данных, полученных по каналу 2
#define ENAA_P1 1 // Включает автоподтверждение данных, полученных по каналу 1
#define ENAA_P0 0 // Включает автоподтверждение данных, полученных по каналу 0

// EN_RXADDR
#define ERX_P5 5 // Включает канал 5 приёмника
#define ERX_P4 4 // Включает канал 4 приёмника
#define ERX_P3 3 // Включает канал 3 приёмника
#define ERX_P2 2 // Включает канал 2 приёмника
#define ERX_P1 1 // Включает канал 1 приёмника
#define ERX_P0 0 // Включает канал 0 приёмника

// SETUP_AW
#define AW 0 // Два бита, Выбирает ширину поля адреса: 1 - 3 байта; 2 - 4 байта; 3 - 5 байт.

#define SETUP_AW_3BYTES_ADDRESS (1 << AW)
#define SETUP_AW_4BYTES_ADDRESS (2 << AW)
#define SETUP_AW_5BYTES_ADDRESS (3 << AW)
```

```
// SETUP_RETR
#define ARD 4 // 4 бита. Задаёт значение задержки перед повторной отправкой пакета: 250 x (n + 1) мкс
#define ARC 0 // 4 битай. Количество повторных попыток отправки, 0 - повторная отправка отключена.

#define SETUP_RETR_DELAY_250MKS  (0 << ARD)
#define SETUP_RETR_DELAY_500MKS  (1 << ARD)
#define SETUP_RETR_DELAY_750MKS  (2 << ARD)
#define SETUP_RETR_DELAY_1000MKS (3 << ARD)
#define SETUP_RETR_DELAY_1250MKS (4 << ARD)
#define SETUP_RETR_DELAY_1500MKS (5 << ARD)
#define SETUP_RETR_DELAY_1750MKS (6 << ARD)
#define SETUP_RETR_DELAY_2000MKS (7 << ARD)
#define SETUP_RETR_DELAY_2250MKS (8 << ARD)
#define SETUP_RETR_DELAY_2500MKS (9 << ARD)
#define SETUP_RETR_DELAY_2750MKS (10 << ARD)
#define SETUP_RETR_DELAY_3000MKS (11 << ARD)
#define SETUP_RETR_DELAY_3250MKS (12 << ARD)
#define SETUP_RETR_DELAY_3500MKS (13 << ARD)
#define SETUP_RETR_DELAY_3750MKS (14 << ARD)
#define SETUP_RETR_DELAY_4000MKS (15 << ARD)

#define SETUP_RETR_NO_RETRANSMIT (0 << ARC)
#define SETUP_RETR_UP_TO_1_RETRANSMIT (1 << ARC)
#define SETUP_RETR_UP_TO_2_RETRANSMIT (2 << ARC)
#define SETUP_RETR_UP_TO_3_RETRANSMIT (3 << ARC)
#define SETUP_RETR_UP_TO_4_RETRANSMIT (4 << ARC)
#define SETUP_RETR_UP_TO_5_RETRANSMIT (5 << ARC)
#define SETUP_RETR_UP_TO_6_RETRANSMIT (6 << ARC)
#define SETUP_RETR_UP_TO_7_RETRANSMIT (7 << ARC)
#define SETUP_RETR_UP_TO_8_RETRANSMIT (8 << ARC)
#define SETUP_RETR_UP_TO_9_RETRANSMIT (9 << ARC)
#define SETUP_RETR_UP_TO_10_RETRANSMIT (10 << ARC)
#define SETUP_RETR_UP_TO_11_RETRANSMIT (11 << ARC)
#define SETUP_RETR_UP_TO_12_RETRANSMIT (12 << ARC)
#define SETUP_RETR_UP_TO_13_RETRANSMIT (13 << ARC)
#define SETUP_RETR_UP_TO_14_RETRANSMIT (14 << ARC)
#define SETUP_RETR_UP_TO_15_RETRANSMIT (15 << ARC)

// RF_SETUP
#define CONT_WAVE   7 // (Только для nRF24L01+) Непрерывная передача несущей (для тестов)
#define RF_DR_LOW   5 // (Только для nRF24L01+) Включает скорость 250кбит/с. RF_DR_HIGH должен быть 0
#define PLL_LOCK    4 // Для тестов
#define RF_DR_HIGH  3 // Выбор скорости обмена (при значении бита RF_DR_LOW = 0): 0 - 1Мбит/с; 1 - 2Мбит/с
#define RF_PWR      1 // 2бита. Выбирает мощность передатчика: 0 - -18dBm; 1 - -16dBm; 2 - -6dBm; 3 - 0dBm

#define RF_SETUP_MINUS18DBM (0 << RF_PWR)
#define RF_SETUP_MINUS12DBM (1 << RF_PWR)
#define RF_SETUP_MINUS6DBM  (2 << RF_PWR)
#define RF_SETUP_0DBM       (3 << RF_PWR)

#define RF_SETUP_1MBPS (0 << RF_DR_HIGH)
#define RF_SETUP_2MBPS (1 << RF_DR_HIGH)
#define RF_SETUP_250KBPS (1 << RF_DR_LOW) // этот режим не должен использоваться с контролем доставки

// STATUS
#define RX_DR   6 // Флаг получения новых данных в FIFO приёмника. Для сброса флага нужно записать 1
#define TX_DS   5 // Флаг завершения передачи. Для сброса флага нужно записать 1
#define MAX_RT  4 // Флаг превышения установленного числа повторов. Без сброса (записать 1) обмен невозможен
#define RX_P_NO 1 // 3 бита. Номер канала, данные для которого доступны в FIFO приёмника. 7 -  FIFO пусто.
#define TX_FULL_STATUS 0 // Признак заполнения FIFO передатчика: 1 - заполнено; 0 - есть доступные слоты
        // (переименовано из TX_FULL во избежание путаницы с одноимённым битом из регистра FIFO_STATUS)

// OBSERVE_TX
#define PLOS_CNT  4 // 4 бита. Общее количество пакетов без подтверждения. Сбрасывается записью RF_CH
#define ARC_CNT   0 // 4 бита. Количество предпринятых повторов при последней отправке.

// FIFO_STATUS
#define TX_REUSE      6 // Признак готовности последнего пакета для повторной отправки.
#define TX_FULL_FIFO  5 // Флаг переполнения FIFO очереди передатчика.
        // (переименовано из TX_FULL во избежание путаницы с одноимённым битом из регистра STATUS)
#define TX_EMPTY      4 // Флаг освобождения FIFO очереди передатчика.
#define RX_FULL       1 // Флаг переполнения FIFO очереди приёмника.
#define RX_EMPTY      0 // Флаг освобождения FIFO очереди приёмника.

// DYNDP
#define DPL_P5 5 // Включает приём пакетов произвольной длины по каналу 5
#define DPL_P4 4 // Включает приём пакетов произвольной длины по каналу 4
#define DPL_P3 3 // Включает приём пакетов произвольной длины по каналу 3
#define DPL_P2 2 // Включает приём пакетов произвольной длины по каналу 2
#define DPL_P1 1 // Включает приём пакетов произвольной длины по каналу 1
#define DPL_P0 0 // Включает приём пакетов произвольной длины по каналу 0

// FEATURE
#define EN_DPL      2 // Включает поддержку приёма и передачи пакетов произвольной длины
#define EN_ACK_PAY  1 // Разрешает передачу данных с пакетами подтверждения приёма
#define EN_DYN_ACK  0 // Разрешает использование W_TX_PAYLOAD_NOACK
```

# Подключение SPI

Чип nRF24L01+ обменивается данными с управляющим микроконтроллером по шине SPI в режиме 0 (CPHA=0, CPOL=0). Это значит, что на линии SCK до начала обмена выдерживается низкий уровень. Начинается обмен с установки низкого уровня на линии CSN, при этом и ведущая и ведомая сторона выставляют на линиях MOSI и MISO соответственно, значения старших бит передаваемых байт. При нарастающем фронте на линии SCK, происходит чтение выставленных значений противоположными сторонами. При спадающем фронте SCK, выставляются значения следующих бит. Передача идёт от старших бит в байте к младшим. Сеанс обмена завершается установкой высокого уровня на линии CSN.

nRF24L01+ поддерживает обмен по SPI на частоте до 10МГц. Однако, в случае большой паразитной емкости линий данных и при наличии помех, может потребоваться уменьшить частоту.

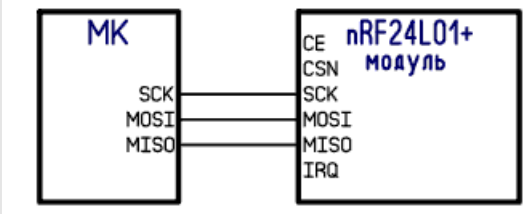
Примеры ниже показывают различные варианты подключения микроконтроллера к шине SPI и определяют две функции:

**void spi_init()** - производит первичную настройку интерфейса SPI;

**uint8_t spi_send_recv(uint8_t data)** - отправляет 1 байт по SPI, по линии MOSI, дожидается окончания обмена и возвращает байт принятый по линии MISO.

В этих примерах не определяется использование линии CSN.

## Использование аппаратного интерфейса SPI




Аппаратный интерфейс SPI микроконтроллеров AVR позволяет организовать обмен на частоте до половины частоты процессора.

При инициализации интерфейса в режиме «мастера», необходимо вручную установить соответствующие биты в регистрах направлений (DDRx). Выводы MOSI, SCK и SS должны быть настроены на выход (соответствующие биты в регистре DDR – единицы), направление вывода MISO переопределяется на «вход» самим интерфейсом.

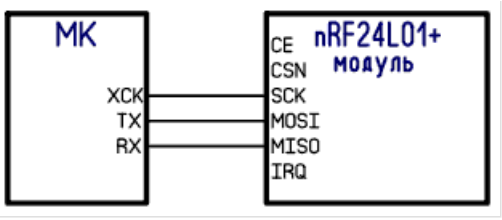
> Когда интерфейс SPI в микроконтроллерах AVR работает в режиме «мастера», вывод SS управляется программистом самостоятельно и не влияет на работу интерфейса, поэтому нет нужды подключать именно этот выход к входу CSN управляемого чипа. **Однако**, этот вывод микроконтроллера **ДОЛЖЕН** быть настроен на «выход», поскольку, в случае если он настроен на «вход», и на нём появится низкий уровень, то интерфейс SPI прервёт обмен и перейдёт в режим «ведомого». Во избежание путаницы, рекомендую использовать этот вывод для подключения к входу CSN.

**Пример кода инициализации интерфейса SPI для микроконтроллеров AVR**

```
#define SPI_DDR DDRB

#define SPI_SS 2
#define SPI_MOSI 3
#define SPI_SCK 5

// Инициализация интерфейса
void spi_init() {
  SPI_DDR |= (1 << SPI_MOSI) | (1 <<  SPI_SCK) | (1 << SPI_SS);
  SPCR = (1 << SPE) | (1 << MSTR); // режим 0, мастер, частота 1/4 от частоты ЦП
}

// Передаёт и принимает 1 байт по SPI, возвращает полученное значение
uint8_t spi_send_recv(uint8_t data) {
  SPDR = data;
  while (!(SPSR & (1 << SPIF)));
  return SPDR;
}
```

*Значения SPI_DDR, SPI_SS, SPI_MOSI, SPI_SCK в примере даны для МК ATmega8(A), ATmega48/88/168/328(P/A/PA). Для других МК подставьте нужные значения*

## Использование аппаратного интерфейса USART в режиме SPI-master




Некоторые микроконтроллеры из серии AVR (напр. ATmega48/88/168/328(P/A/PA)) позволяют переключить интерфейс USART в режим SPI-master. Вывод TX микроконтроллера подключается к линии MOSI, RX – к MISO, XCK – к SCK.

Скорость обмена задаётся значением пары регистров UBRRnL, UBBRnH. На время инициализации рекомендуется установить их значение в 0, дабы на линии XCK появился нужный уровень.

Направление выводов TX и RX переопределяется интерфейсом, однако направление вывода XCK нужно задать на «выход» явно.

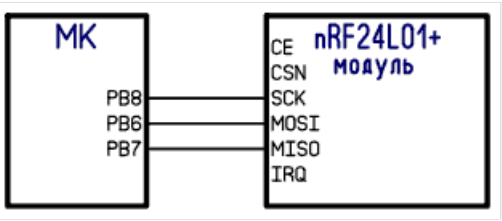
Интерфейс использует буферизацию принимаемых и отправляемых данных, однако для работы в режиме SPI удобнее использовать не буферизированный обмен, дожидаясь окончания приёма.

**Пример кода инициализации интерфейса SPI для микроконтроллеров AVR**

```
#define USART_DDR DDRD

#define USART_XCK 4

// Инициализация интерфейса USART0 в режиме SPI-master
void spi_init() {
  UBRR0 = 0;
  USART_DDR |= (1 << USART_XCK);
  UCSR0C = (1 << UMSEL01) | (1 << UMSEL00); // выбор режима SPI-master
  UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Включение приёмника и передатчика
  UBRR0 = 1;  // Выбор частоты интерфейса 1/4 от частоты ЦП
}

// Передаёт и принимает 1 байт по SPI, возвращает полученное значение
uint8_t spi_send_recv(uint8_t data) {
  UDR0 = data;
  while (!(UCSR0A & (1 << RXC0)));
  return UDR0;
}
```

*Значения USART_DDR и USART_XCK в примере даны для МК ATmega48/88/168/328(P/A/PA). Для других МК подставьте нужные значения*

## Использование программной реализации интерфейса SPI




Программная реализация интерфейса, хоть и накладывает ограничения на максимальную скорость обмена, но позволяет подключать ведомое устройство к любым доступным портам ввода/вывода.

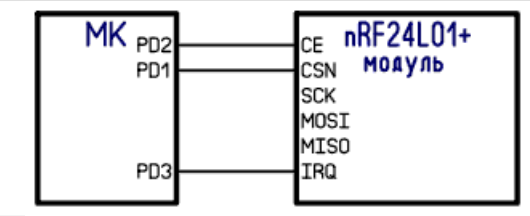
**Пример кода инициализации интерфейса SPI для микроконтроллеров AVR**

```
#define SWSPI_DDR DDRB
#define SWSPI_PORT PORTB
#define SWSPI_PIN PINB

#define SWSPI_MOSI 6
#define SWSPI_MISO 7
#define SWSPI_SCK 8

// Инициализация программного интерфейса SPI
void spi_init() {
  SWSPI_PORT &= ~((1 << SWSPI_MOSI) | (1 << SWSPI_SCK));
  SWSPI_DDR |= (1 << SWSPI_MOSI) | (1 << SWSPI_SCK);
  SWSPI_DDR &= ~ (1 << SWSPI_MISO);
  SWSPI_PORT |= (1 << SWSPI_MISO); // подтяжка на линии MISO
}

// Передаёт и принимает 1 байт по SPI, возвращает полученное значение
uint8_t spi_send_recv(uint8_t data) {
  for (uint8_t i = 8; i > 0; i--) {
    if (data & 0x80)
      SWSPI_PORT |= (1 << SWSPI_MOSI); // передача единички
    else
      SWSPI_PORT &= ~(1 << SWSPI_MOSI); // передача нулика
    SWSPI_PORT |= (1 << SWSPI_SCK);
    data <<= 1;
    if (SWSPI_PIN & (1 << SWSPI_MISO)) // Чтение бита на линии MISO
      data |= 1;
    SWSPI_PORT &= ~(1 << SWSPI_SCK);
  }
```

```
  return data;
}
```

# Взаимодействие с nRF24L01+

## Функции настройки портов и базового взаимодействия




Линия **CSN**, определяющая сеанс обмена по интерфейсу SPI, имеет активным низкий уровень. Линия **CE**, наоборот, активна по высокому уровню. Чтобы избежать путанницы определим функции **radio_assert_ce**, **radio_deassert_ce**, **csn_assert**, **csn_deassert**, которые будут переводить линии ce/csn к активному (assert) и неактивному (deassert) уровню.

Также ниже будут определены функции читающие/записывающие регистры и вызывающие выполнение команд чипа nRF24L01+.

Но для начала нужно определить порты, на которых будет происходить обмен.

```
#define RADIO_PORT PORTD
#define RADIO_DDR DDRD
#define RADIO_PIN PIND

#define RADIO_CSN 1
#define RADIO_CE 2
#define RADIO_IRQ 3

// Выбирает активное состояние (высокий уровень) на линии CE
inline void radio_assert_ce() {
  RADIO_PORT |= (1 << RADIO_CE); // Установка высокого уровня на линии CE
}

// Выбирает неактивное состояние (низкий уровень) на линии CE
inline void radio_deassert_ce() {
  RADIO_PORT &= ~(1 << RADIO_CE); // Установка низкого уровня на линии CE
}

// Поскольку функции для работы с csn не предполагается использовать в иных файлах, их можно объявить static

// Выбирает активное состояние (низкий уровень) на линии CSN
inline static void csn_assert() {
  RADIO_PORT &= ~(1 << RADIO_CSN); // Установка низкого уровня на линии CSN
}

// Выбирает неактивное состояние (высокий уровень) на линии CSN
inline static void csn_deassert() {
  RADIO_PORT |= (1 << RADIO_CSN); // Установка высокого уровня на линии CSN
}

// Инициализирует порты
void radio_init() {
  RADIO_DDR |= (1 << RADIO_CSN) | (1 << RADIO_CE); // Ножки CSN и CE на выход
  RADIO_DDR &= ~(1 < RADIO_IRQ); // IRQ - на вход
  csn_deassert();
  radio_deassert_ce();
  spi_init();
}

// Выполняет команду cmd, и читает count байт ответа, помещая их в буфер buf, возвращает регистр статуса
uint8_t radio_read_buf(uint8_t cmd, uint8_t * buf, uint8_t count) {
  csn_assert();
  uint8_t status = spi_send_recv(cmd);
  while (count--) {
    *(buf++) = spi_send_recv(0xFF);
  }
  csn_deassert();
  return status;
}

// Выполняет команду cmd, и передаёт count байт параметров из буфера buf, возвращает регистр статуса
uint8_t radio_write_buf(uint8_t cmd, uint8_t * buf, uint8_t count) {
  csn_assert();
  uint8_t status = spi_send_recv(cmd);
  while (count--) {
    spi_send_recv(*(buf++));
  }
  csn_deassert();
  return status;
}

// Читает значение однобайтового регистра reg (от 0 до 31) и возвращает его
```

```
uint8_t radio_readreg(uint8_t reg) {
  csn_assert();
  spi_send_recv((reg & 31) | R_REGISTER);
  uint8_t answ = spi_send_recv(0xFF);
  csn_deassert();
  return answ;
}

// Записывает значение однобайтового регистра reg (от 0 до 31), возвращает регистр статуса
uint8_t radio_writereg(uint8_t reg, uint8_t val) {
  csn_assert();
  uint8_t status = spi_send_recv((reg & 31) | W_REGISTER);
  spi_send_recv(val);
  csn_deassert();
  return status;
}

// Читает count байт многобайтового регистра reg (от 0 до 31) и сохраняет его в буфер buf,
// возвращает регистр статуса
uint8_t radio_readreg_buf(uint8_t reg, uint8_t * buf, uint8_t count) {
  return radio_read_buf((reg & 31) | R_REGISTER, buf, count);
}

// Записывает count байт из буфера buf в многобайтовый регистр reg (от 0 до 31), возвращает регистр статуса
uint8_t radio_writereg_buf(uint8_t reg, uint8_t * buf, uint8_t count) {
  return radio_write_buf((reg & 31) | W_REGISTER, buf, count);
}

// Возвращает размер данных в начале FIFO очереди приёмника
uint8_t radio_read_rx_payload_width() {
  csn_assert();
  spi_send_recv(R_RX_PL_WID);
  uint8_t answ = spi_send_recv(0xFF);
  csn_deassert();
  return answ;
}

// Выполняет команду. Возвращает регистр статуса
uint8_t radio_cmd(uint8_t cmd) {
  csn_assert();
  uint8_t status = spi_send_recv(cmd);
  csn_deassert();
  return status;
}

// Возвращает 1, если на линии IRQ активный (низкий) уровень.
uint8_t radio_is_interrupt() {
  return (RADIO_PIN & RADIO_IRQ) ? 0 : 1;
}
```

В крайних случаях, если нужно сэкономить линию, пин прерывания можно не подключать, а проверять наличие прерывания анализируя регистр статуса:

```
uint8_t radio_is_interrupt() {
// использовать этот вариант только в крайних случаях!!!
  return (radio_cmd(NOP) & ((1 << RX_DR) | (1 << TX_DS) | (1 << MAX_RT))) ? 1 : 0;
}
```

## Первоначальная настройка регистров

Хотя все регистры имеют определённое значение по-умолчанию, которое загружается в них после подачи питания, рекомендую явно задать значения всех необходимых регистров. Особенно это актуально, если линия CSN не притянута внешним резистором к питанию: тогда в момент сброса микроконтроллера, радио-модуль может улавливать помехи и хаотичным образом перенастроить регистры.

Пример ниже показывает инициализацию с готовностью работы в режиме приёма по каналу 1 и передачи. Выбирается длина адреса 5-байт, радио канал 3 (2403 МГц). Собственный адрес 0xE7E7E7E7E7, адрес удалённой стороны - 0xC2C2C2C2C2

```
// Функция производит первоначальную настройку устройства. Возвращает 1, в случае успеха, 0 в случае ошибки
uint8_t radio_start() {
  uint8_t self_addr[] = {0xE7, 0xE7, 0xE7, 0xE7, 0xE7}; // Собственный адрес
  uint8_t remote_addr[] = {0xC2, 0xC2, 0xC2, 0xC2, 0xC2}; // Адрес удалённой стороны
```

```
  uint8_t chan = 3; // Номер радио-канала (в диапазоне 0 - 125)

  radio_deassert_ce();
  for(uint8_t cnt = 100;;) {
    radio_writereg(CONFIG, (1 << EN_CRC) | (1 << CRCO) | (1 << PRIM_RX)); // Выключение питания
    if (radio_readreg(CONFIG) == ((1 << EN_CRC) | (1 << CRCO) | (1 << PRIM_RX)))
      break;
    // Если прочитано не то что записано, то значит либо радио-чип ещё инициализируется, либо не работает.
    if (!cnt--)
      return 0; // Если после 100 попыток не удалось записать что нужно, то выходим с ошибкой
    _delay_ms(1);
  }

  radio_writereg(EN_AA, (1 << ENAA_P1)); // включение автоподтверждения только по каналу 1
  radio_writereg(EN_RXADDR, (1 << ERX_P0) | (1 << ERX_P1)); // включение каналов 0 и 1
  radio_writereg(SETUP_AW, SETUP_AW_5BYTES_ADDRESS); // выбор длины адреса 5 байт
  radio_writereg(SETUP_RETR, SETUP_RETR_DELAY_250MKS | SETUP_RETR_UP_TO_2_RETRANSMIT);
  radio_writereg(RF_CH, chan); // Выбор частотного канала
  radio_writereg(RF_SETUP, RF_SETUP_1MBPS | RF_SETUP_0DBM); // выбор скорости 1 Мбит/с и мощности 0dBm

  radio_writereg_buf(RX_ADDR_P0, &remote_addr[0], 5); // Подтверждения приходят на канал 0
  radio_writereg_buf(TX_ADDR, &remote_addr[0], 5);

  radio_writereg_buf(RX_ADDR_P1, &self_addr[0], 5);

  radio_writereg(RX_PW_P0, 0);
  radio_writereg(RX_PW_P1, 32);
  radio_writereg(DYNPD, (1 << DPL_P0) | (1 << DPL_P1)); // включение произвольной длины для каналов 0 и 1
  radio_writereg(FEATURE, 0x04); // разрешение произвольной длины пакета данных

  radio_writereg(CONFIG, (1 << EN_CRC) | (1 << CRCO) | (1 << PWR_UP) | (1 << PRIM_RX)); // Включение питания
  return (radio_readreg(CONFIG) == ((1 << EN_CRC) | (1 << CRCO) | (1 << PWR_UP) | (1 << PRIM_RX))) ? 1 : 0;
}
```

## Отправка и приём сообщений

### Двусторонний обмен

В общем случае, алгоритм работы программы таков: всё время радио-модуль находится в режиме приёма (PRX), ожидая, когда к нему обратятся по радио-эфиру.

Когда возникает необходимость передать пакет, nRF24L01+ переводится в режим передатчика, осуществляет отправку пакета, а по завершению - удачному, или нет - возвращается в режим приёмника.

Примере ниже определены функции для обмена по такому сценарию

Вызов функции *send_data* осуществляет отправку буфера указанной длины.

Основной цикл программы периодически вызывает функцию *check_radio*, которая, в свою очередь, анализирует прерывания от радио-чипа и, в зависимости от них, вызывает:

- функцию *on_packet* когда принят новый пакет по каналу 1.
- функцию *on_send_error* когда при передаче пакета было превышено количество попыток.

```
// Вызывается, когда превышено число попыток отправки, а подтверждение так и не было получено.
void on_send_error() {
 // TODO здесь можно описать обработчик неудачной отправки
}

// Вызывается при получении нового пакета по каналу 1 от удалённой стороны.
// buf - буфер с данными, size - длина данных (от 1 до 32)
void on_packet(uint8_t * buf, uint8_t size) {
 // TODO здесь нужно написать обработчик принятого пакета

 // Если предполагается немедленная отправка ответа, то необходимо обеспечить задержку ,
 // во время которой чип отправит подтверждение о приёме
 // чтобы с момента приёма пакета до перевода в режим PTX прошло:
 // 130мкс + ((длина_адреса + длина_CRC + длина_данных_подтверждения) * 8 + 17) / скорость_обмена
 // При типичных условиях и частоте МК 8 мГц достаточно дополнительной задержки 100мкс
}



// Помещает пакет в очередь отправки.
// buf - буфер с данными, size - длина данных (от 1 до 32)
uint8_t send_data(uint8_t * buf, uint8_t size) {
  radio_deassert_ce(); // Если в режиме приёма, то выключаем его
  uint8_t conf = radio_readreg(CONFIG);
  if (!(conf & (1 << PWR_UP))) // Если питание по какой-то причине отключено, возвращаемся с ошибкой
    return 0;
```

```
  uint8_t status = radio_writereg(CONFIG, conf & ~(1 << PRIM_RX)); // Сбрасываем бит PRIM_RX
  if (status & (1 << TX_FULL_STATUS))  // Если очередь передатчика заполнена, возвращаемся с ошибкой
    return 0;
  radio_write_buf(W_TX_PAYLOAD, buf, size); // Запись данных на отправку
  radio_assert_ce(); // Импульс на линии CE приведёт к началу передачи
  _delay_us(15); // Нужно минимум 10мкс, возьмём с запасом
  radio_deassert_ce();
  return 1;
}

void check_radio() {
  if (!radio_is_interrupt()) // Если прерывания нет, то не задерживаемся
    return;
  uint8_t status = radio_cmd(NOP);
  radio_writereg(STATUS, status); // Просто запишем регистр обратно, тем самым сбросив биты прерываний

  if (status & ((1 << TX_DS) | (1 << MAX_RT))) { // Завершена передача успехом, или нет,
    if (status & (1 << MAX_RT)) { // Если достигнуто максимальное число попыток
      radio_cmd(FLUSH_TX); // Удалим последний пакет из очереди
      on_send_error(); // Вызовем обработчик
    }
    if (!(radio_readreg(FIFO_STATUS) & (1 << TX_EMPTY))) { // Если в очереди передатчика есть что передавать
      radio_assert_ce(); // Импульс на линии CE приведёт к началу передачи
      _delay_us(15); // Нужно минимум 10мкс, возьмём с запасом
      radio_deassert_ce();
    } else {
      uint8_t conf = radio_readreg(CONFIG);
      radio_writereg(CONFIG, conf | (1 << PRIM_RX)); // Устанавливаем бит PRIM_RX: приём
      radio_assert_ce(); // Высокий уровень на линии CE переводит радио-чип в режим приёма
    }
  }
  uint8_t protect = 4; // В очереди FIFO не должно быть более 3 пакетов. Если больше, значит что-то не так
  while (((status & (7 << RX_P_NO)) != (7 << RX_P_NO)) && protect--) { // Пока в очереди есть принятый пакет
    uint8_t l = radio_read_rx_payload_width(); // Узнаём длину пакета
    if (l > 32) { // Ошибка. Такой пакет нужно сбросить
      radio_cmd(FLUSH_RX);
    } else {
      uint8_t buf[32]; // буфер для принятого пакета
      radio_read_buf(R_RX_PAYLOAD, &buf[0], l); // начитывается пакет
      if ((status & (7 << RX_P_NO)) == (1 << RX_P_NO)) { // если datapipe 1
        on_packet(&buf[0], l); // вызываем обработчик полученного пакета
      }
    }
    status = radio_cmd(NOP);
  }
}


// Основной цикл
int main(void) {
  radio_init();
  while (!radio_start()) {
    _delay_ms(1000);
  }
  // Перед включением питания чипа и сигналом CE должно пройти время достаточное для начала работы осциллятора
  // Для типичных резонаторов с эквивалентной индуктивностью не более 30мГн достаточно 1.5 мс
  _delay_ms(2);

  radio_assert_ce();

  for(;;) {
    check_radio();

    // TODO здесь основной код программы
  }
}
```

## Только приём

В случае, если устройство работает только на приём, т.е. принимает пакеты, но не отправляет их, канал 0 может быть запрещён в регистрах EN_RXADDR и DYNPD в *radio_start()*.

Из функции *check_radio()* можно исключить часть, ответственную за прерывания передатчика.

## Только передача

Если вы реализуете пульт дистанционного управления, то он должен обеспечивать только передачу. А в целях экономии энергии сразу после передачи одиночной команды переводить радио-чип в спящий режим.

В этом случае в *radio_start()* канал 1 может быть запрещён в регистрах EN_RXADDR и DYNPD.

Функции передачи сообщения и обработчики прерываний могут выглядеть так:

```
// Помещает пакет в очередь отправки.
// buf - буфер с данными, size - длина данных (от 1 до 32)
uint8_t send_data(uint8_t * buf, uint8_t size) {
  radio_deassert_ce(); // Если в режиме приёма, то выключаем его
  uint8_t conf = radio_readreg(CONFIG);
  // Сбрасываем бит PRIM_RX, и включаем питание установкой PWR_UP
  uint8_t status = radio_writereg(CONFIG, (conf & ~(1 << PRIM_RX)) | (1 << PWR_UP));
  if (status & (1 << TX_FULL_STATUS))  // Если очередь передатчика заполнена, возвращаемся с ошибкой
    return 0;
  if (!(conf & (1 << PWR_UP))) // Если питание не было включено, то ждём, пока запустится осциллятор
    _delay_ms(2);
  radio_write_buf(W_TX_PAYLOAD, buf, size); // Запись данных на отправку
  radio_assert_ce(); // Импульс на линии CE приведёт к началу передачи
  _delay_us(15); // Нужно минимум 10мкс, возьмём с запасом
  radio_deassert_ce();
  return 1;
}

void check_radio() {
  if (!radio_is_interrupt()) // Если прерывания нет, то не задерживаемся
    return;
  uint8_t status = radio_cmd(NOP);
  radio_writereg(STATUS, status); // Просто запишем регистр обратно, тем самым сбросив биты прерываний

  if (status & ((1 << TX_DS) | (1 << MAX_RT))) { // Завершена передача успехом, или нет,
    if (status & (1 << MAX_RT)) { // Если достигнуто максимальное число попыток
      radio_cmd(FLUSH_TX); // Удалим последний пакет из очереди
      on_send_error(); // Вызовем обработчик
    }
    if (!(radio_readreg(FIFO_STATUS) & (1 << TX_EMPTY))) { // Если в очереди передатчика есть что передавать
      radio_assert_ce(); // Импульс на линии CE приведёт к началу передачи
      _delay_us(15); // Нужно минимум 10мкс, возьмём с запасом
      radio_deassert_ce();
    } else {
      uint8_t conf = radio_readreg(CONFIG);
      radio_writereg(CONFIG, conf & ~(1 << PWR_UP)); // Если пусто, отключаем питание
    }
  }
  uint8_t protect = 4; // В очереди FIFO не должно быть более 3 пакетов. Если больше, значит что-то не так
  while (((status & (7 << RX_P_NO)) != (7 << RX_P_NO)) && protect--) { // Пока в очереди есть принятый пакет
    radio_cmd(FLUSH_RX); // во всех случаях выкидываем пришедший пакет.
    status = radio_cmd(NOP);
  }
}
```

5 комментариев

Ваш комментарий...

поделиться Отправить

**Master Electric**
Супер!!!
5 янв в 16:33 Комментировать

**Кулибин Муковоз**
Статья хорошая, все рассказано подробно.Есть ли такая для датчика DHT11? И примеры кода на ассемблере.
5 янв в 3:58 Комментировать

**Сергей Брусницын**
Статья отличная! Сделал модули для Atmel Studio 6.2 и Keil uVision5. Но возникла проблема, не могу передать температуру otds18b20 и attiny85 на stm32.
sprintf(CTR_SENDBUFF, "+26.50");
send_data(CTR_SENDBUFF,32);
Если сделать так то в приёмнике получим +26.50 вроде нормально, только данные липовые.
Но если сделать так
Показать полностью...
27 дек в 18:12

**Дмитрий Погребняк**
Что такое converttemp и какой тип у tt? %f подразумевает что tt должно быть число с плавающей точкой.
Кстати, зачем показания температуры передаются в виде текстовой строки? Не практичнее передавать их в двоичном виде, в тех же 1/16 градуса, например?
3 янв в 18:36 Ответить

Написать комментарий...

**Maxim Filatov**
Шикарная статья!!!
17 дек в 23:03 Комментировать

**Виталий Приезжев**
Хорошая статья, всё очень подробно, но есть одна ошибка, которая отняла у меня не мало времени и нервов. При настройке Вы включаете автоподтверждения только по каналу 1 и нигде не уточняете, что если камень в режиме передатчика, то нужно включать и для канала 0, иначе он просто не будет ловить ACK пакеты.
1 ноя в 16:36 Комментировать

5 ms; mod: Sat, 06 May 2017 20:41:51 GMT; gen: Wed, 31 Jan 2018 20:09:22 GMT