**NRF24L01 BASICS**

**NRF24L01 Hardware Interface:**

Eight pins to interface with, and these are Vcc, GND, IRQ, CE, and the four SPI-related pins (CSN, SCK, MISO, and MOSI)

* CE is used to control data transmission and reception when in TX and RX modes, respectively. IRQ is the interrupt pin, and is active-low. There are three internal interrupts that can cause this pin to go low when they are active. Each of these bits can be masked out such that when the bit’s respective interrupt becomes active, the status of the IRQ pin is not changed.
* The second pin down is the CE pin. This pin is always an input with respect to the 24L01. This pin has different meanings depending on what mode the 24L01 is in at a given time. First we assume that the device is powered-up internally (by setting the PWR\_UP bit in the CONFIG register…details on how to do that later). If the device is a receiver, having CE high allows the 24L01 to monitor the air and receive packets. CE being low puts the chip in standby and it no longer monitors the air. If the device is a transmitter, CE is always held low except when the user wants to transmit a packet. This is done by loading the TX FIFO and then toggling the CE pin (low-to-high transition, leave high for at least 10 uS, then back to low).

**Interfacing the nRF24L01 via SPI**

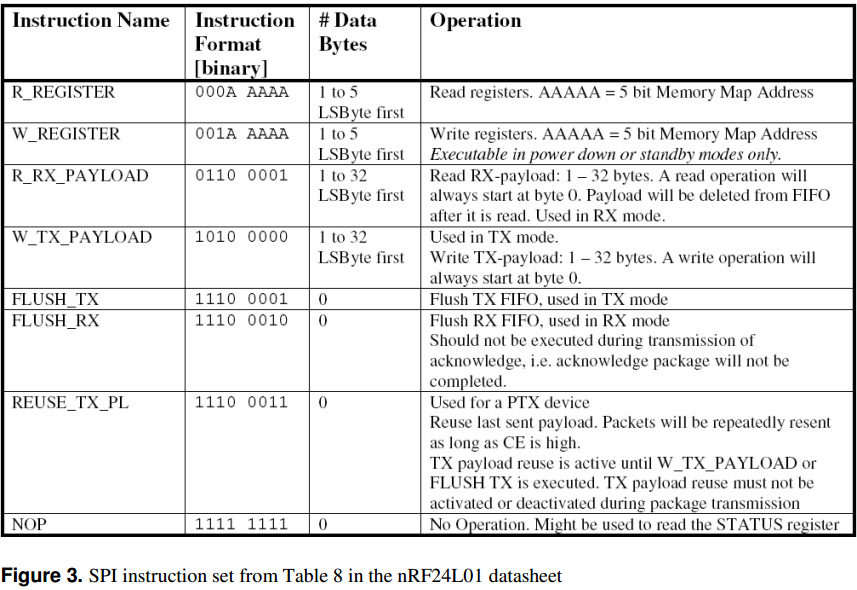
On the p.1, it states that the max is 8 Mbps. However, on p.19, a max of 10 Mbps is stated.

Set your MCU as a master device, 8-bit data transfer.

**SPI Instruction Set Summary**

In order to send data to or receive data from the SPI port on the 24L01, you have to do a few things.

* The CSN pin on the 24L01 must be high to start out with.
* Then, you bring the CSN pin low to alert the 24L01 that it is about to receive SPI data. (Note: this pin will stay low throughout the entire transaction.)
* Then, you will transmit thecommand byte of the instruction you wish to send. If you are receiving data bytes for this instruction, you must then send one byte to the 24L01 for every one byte that you wish to get out of the 24L01. If you are just sending the 24L01 data, you simply send your data bytes and generally don’t worry about what it sends back to you. When receiving data from the 24L01, it makes absolutely no difference what is contained in the data bytes you send after the command byte, just so long as you send the correct number of them.
* Once you have transmitted and/or read all of the bytes that you need, you bring CSN back high.
* Finally, you can process this data in your micro at will.

****For example, let’s say you are going to execute the R\_REGISTER instruction on TX\_ADDR register, which will read the contents of the TX address register outof the 24L01 and into your micro (more on the instruction set next). The TX\_ADDR register is 5 bytes wide, and we’ll assume that you are using 5-byte addresses.

Solution is: First, you would bring CSN low and then send the command byte ‘00010000’ to the 24L01. This instructs the 24L01 that you want to read register 0x10, which is the TX\_ADDR register. Then you would send five dummy data bytes (it makes absolutely no difference what the data bytes contain), and the 24L01 will send back to you the contents of the TX\_ADDR register. Finally, you would bring the CSN pin back high.

Note: When you send any command byte, the 24L01 always returns to you the STATUS register. After that, you will have received the five bytes that are contained in the TX\_ADDR register.

* Next is the W\_REGISTER instruction, which allows you to write most any of the registers in the 24L01.
* Now for the first of the two “big-money” operations – R\_RX\_PAYLOAD. This operation allows you to read the contents of the RX FIFO if you have received a packet (when you are in RX mode), which is generally signaled by the RX\_DR interrupt. Using this operation is a little more involved than the others, because it requires you to also use CE.

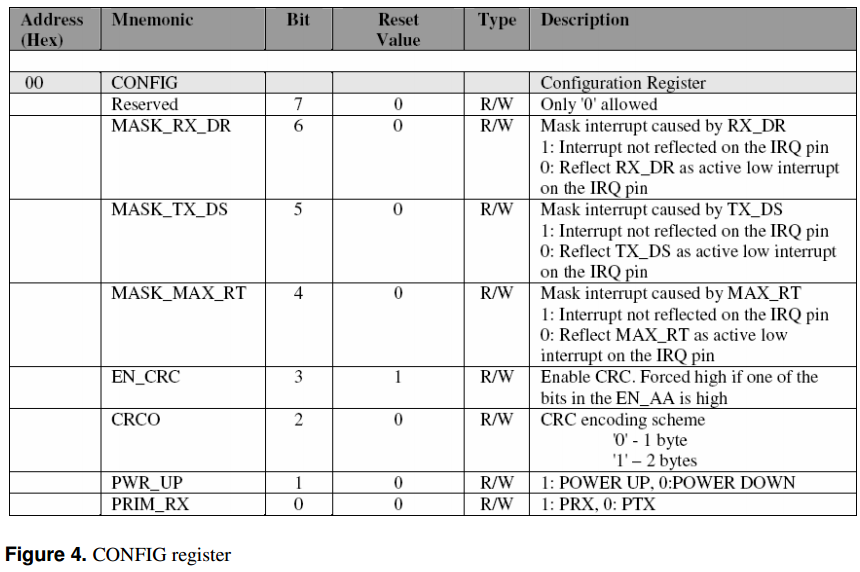
When you are receiving packets, CE is held high. Once you have received a packet you MUST bring CE low to disable the receiver, and then you execute the R\_RX\_PAYLOAD operation. You send the command byte (0x61), and then send the same amount of dummy bytes as your payload width is for the pipe you received data on. The 24L01 will automatically delete the payload that you read out from the top of the FIFO. If there are more payloads in the FIFO (the device can hold 3 payloads in the RX FIFO – see the “FIFO Info” section for more details), then you should continue reading them out until they are all read. At this point, you would clear the RX\_DR interrupt, bring CE high again to monitor for packets, and then process the data you received.

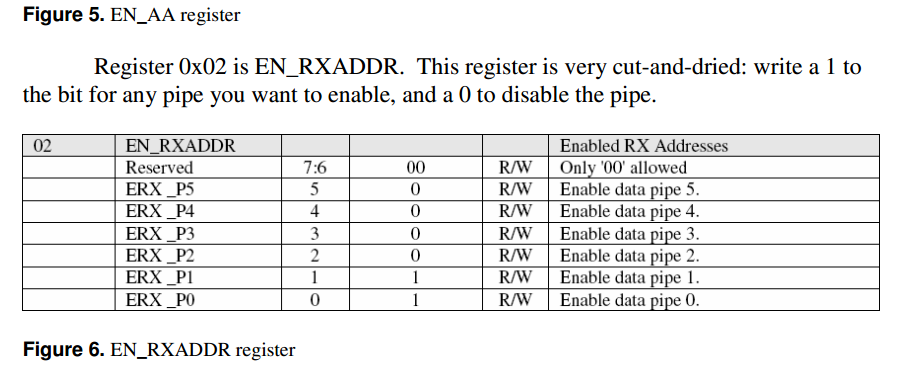
* The second of the “big-money” operations is W\_TX\_PAYLOAD. This guy is used when you are in TX mode and wish to send a packet. In normal TX operation, the CE pin is held low. You first send the command byte (0xA0), and then the payload. The number of payload bytes you send MUST match the payload length of the receiver you are sending the payload to.

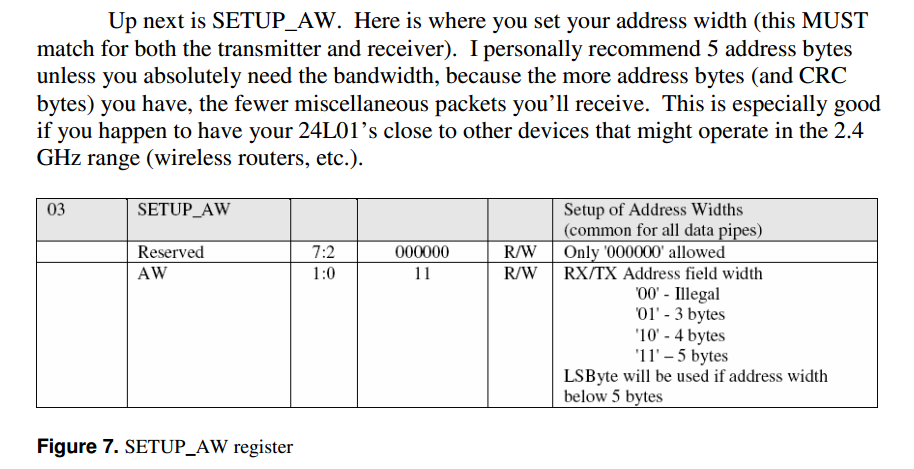
Once you load the packet, you toggle the CE pin to send the packet on its way (keeping it high for at least 10 us). If the packet was sent, the TX\_DS interrupt will occur. This is to signal to you that the 24L01 has successfully sent your packet. If you had auto-ack enabled on this pipe, the TX\_DS flag will only be set if the packet actually gets through. If the maximum amount of retries is hit, then the MAX\_RT interrupt will become active. At this point, you should clear the interrupts and continue based on which interrupt was asserted. Also remember that, like the RX FIFO, the TX FIFO is three levels deep. This means that you can load up to three packets into the 24L01’sTX FIFO before you do the CE toggle to send them on their way.

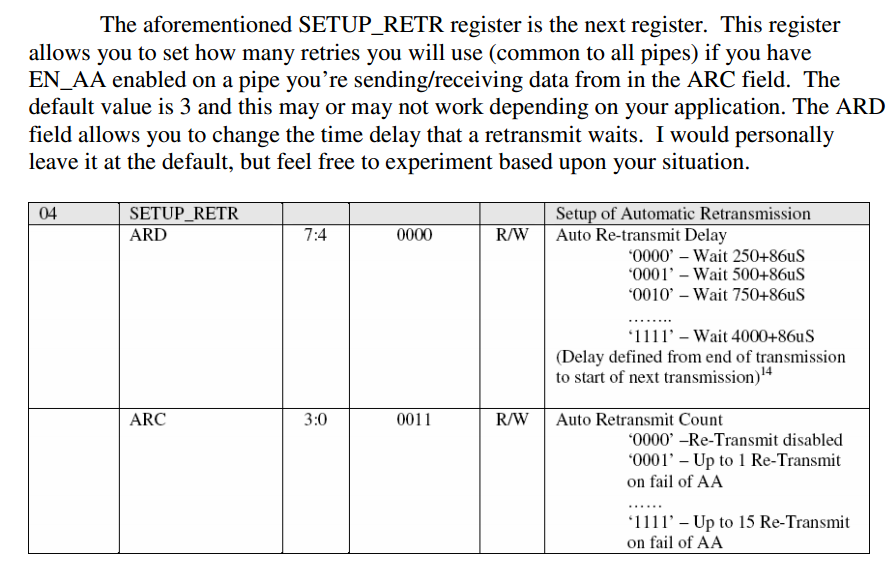
* The FLUSH\_TX and FLUSH\_RX operations (opcodes 0xE1 and 0xE2, respectively) are used to clear out any data in the TX and RX FIFOs, respectively. This can be useful if you should happen to receive a packet that you know is not valid without even checking it (like during setup, etc.). These operations have no data bytes. An operation that I have never found a real use for is REUSE\_TX\_PL. This allows you to constantly send a packet as long as CE is held high for a TX device(the status of this operation is reflected in the TX\_REUSE field of the FIFO\_STATUS register). The opcodes for this guy is 0xE3 and it has no data bytes. If you should decide to execute this operation, you can stop it temporarily by bringing CE low. To stop it permanently, you can stop it by issuing either the W\_TX\_PAYLOAD operation or the FLUSH\_TX operation. You should be sure to not execute this instruction while the 24L01 is sending a packet though, because it could cause strange stuff to happen.
* The final instruction the 24L01 knows is the NOP. It has an opcode of 0xFF and no data bytes. The only real use for this operation is to read the STATUS registerquickly, but this does indeed come in handy, especially if you’re not using the IRQ pin.

**A Quick Overview of the nRF24L01’s Internal Registers**

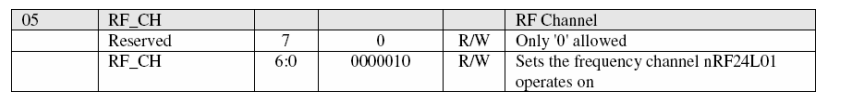
* ****The first register (well, the zeroth technically, but I digress) is the CONFIG register. It is the most important register as far as getting the 24L01 to do something goes. This register contains the PWR\_UP bit, which MUST be set in order to make the device do anything. Therefore, if you want to get the 24L01 talking, you have to write to this register at least once. Also important in this register is PRIM\_RXbit, which, if set, makes the device a receiver, and if cleared, makes the device a transmitter. You configure your interrupts here as well, and you also set up your CRC scheme. I personally recommend a 2-byte CRC, but it depends a lot on how much data rate you need. The register description can be found on the following page in Figure 4.
* The next register is EN\_AA, or the enable auto-ack register. This register allows you to enable or disable auto-acknowledgements on a per-pipe basis. If you are at all familiar with TCP, you know what acknowledgements are. Essentially, the transmitting 24L01 will send a packet and then switch momentarily for a receiver. If the receiving 24L01 gets the packet, it will send back an acknowledgement. If the transmitting 24L01 gets the acknowledgement, it changes back to a transmitter and all is happy. If the transmitting 24L01 doesn’t get the acknowledgement back in the specified window, it sends the packet again and waits for an ack. It will do this the number of times allowed in the SETUP\_RETR register for the pipe being used, and then if no acknowledgement is received it will assert the MAX\_RT interrupt. This is an extremely useful feature and I would recommend using it if possible. One caveat is that you must have the receive address of pipe 0 be the same as the transmit address if you have auto-ack enabled on the pipe you’re sending data to/from (see p.14 of the datasheet). The register description can be found on the following page in Figure 5.





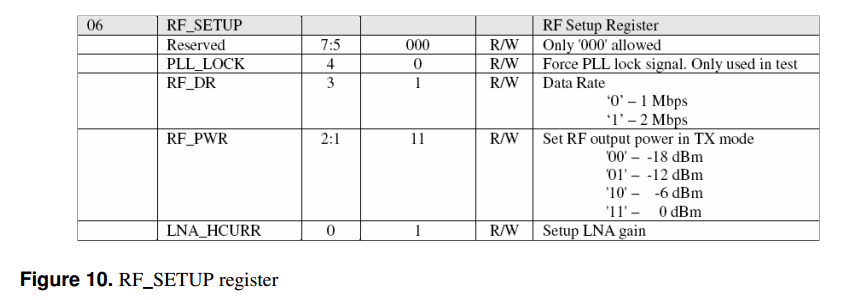


* Register 0x05 is the RF\_CH register, which allows you to set your RF channel.



The next register is the RF\_SETUP register. This guy contains all the necessary parameters (besides the channel number) to set up the RF section of the 24L01. Be sure to leave PLL\_LOCK and LNA\_HCURR at their defaults at all times because changing them could cause stuff to go weird.

The RF\_DR field allows you to change your data rate, and I would recommend using 2 Mbps as long as you have a decent quality link. Going to 1 Mbps will allow you to get better range and a larger percentage of packets though, with the obvious tradeoff of lower bandwidth. You can also change the RF power level using the RF\_PWR field. I would recommend leaving it at its highest value of binary 11, which is also the default. You can lower this if you need to save power and/or have devices really close together.

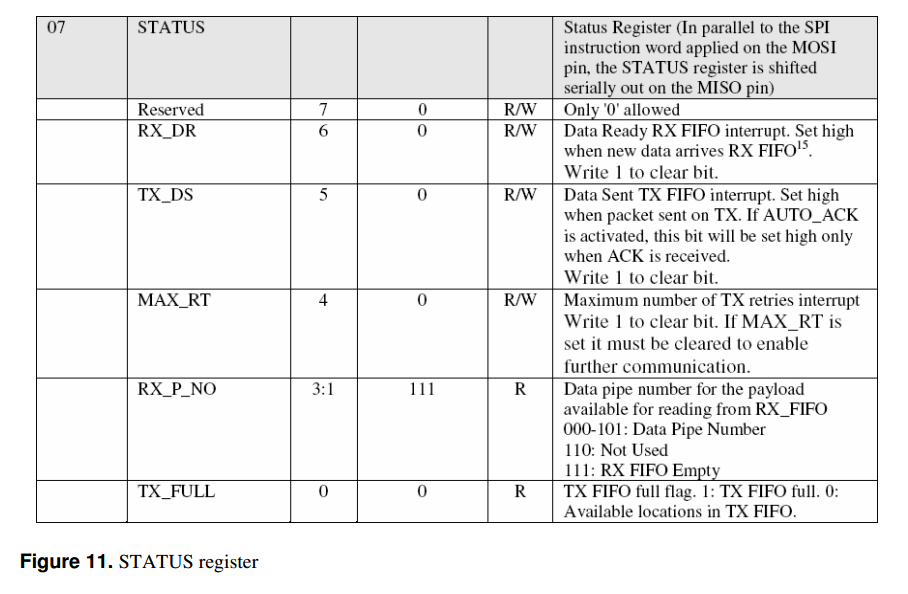


* Your new best friend is up next – the STATUS register. This is the guy you will interface with most when you’re using the chip (other than the TX/RX FIFOs – more on them later).

+ When the IRQ pin is asserted (goes low), this is the register you check to see what has happened. It tells you the interrupts that are currently asserted, the data pipe for a payload that has been received for a receiver (RX\_P\_NO field).

+ and if the TX FIFO is full for a transmitter (TX\_FULL field). If you aren’t using the IRQ pin, this is the register you poll to see if anything has happened.

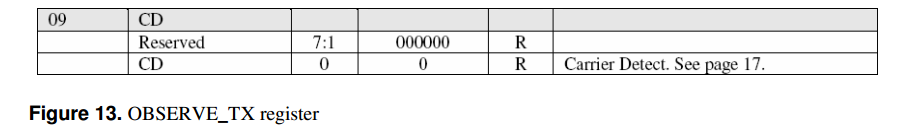
+ When an interrupt is asserted, you MUST write a 1 to the proper bit in this register to clear it. The register description can be found on the following page in Figure 11.



* The OBSERVE\_TX register is a bit complicated to describe.

+ The PLOS\_CNT field describes how many packets have been lost since the last reset. This means that for all the channels that have or have had auto-ack enabled, if a packet has been retransmitted the maximum number of retries without success, this field increments. It is limited to 15 however. You reset this counter by making a write to RF\_CH. I would presume they did it this way to tell you that you should change the RF channel because you are in an area that is not giving you good quality.

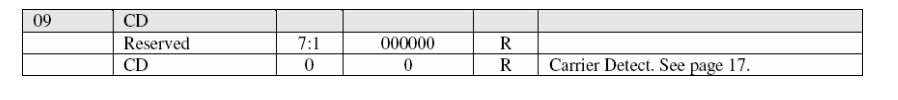
+ The other field in this register is ARC\_CNT, and this is basically the number of resent packets for a given data payload. It resets itself when a new packet is transmitted.



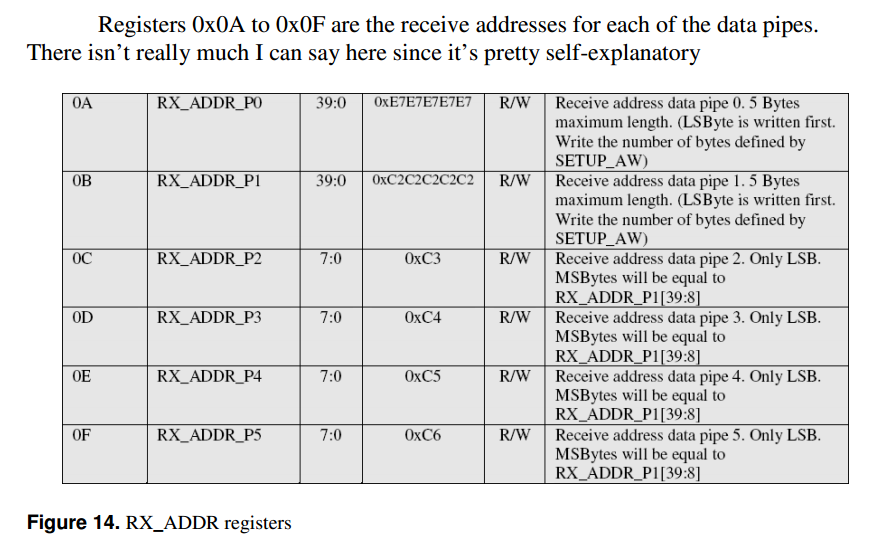
* Register 0x09 is the CD register. This register essentially monitorsthe air to see if there is anything broadcasting in your RF channel.

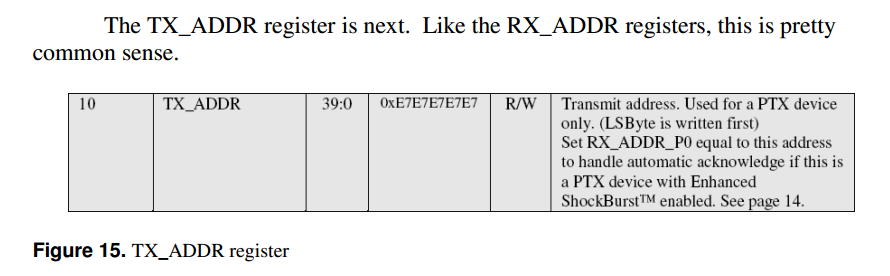
+ Its value is 0 if nothing is detected and 1 if there is something going on. I can honestly say I’ve never used it, but it would probably be useful for situations where you have multiple 24L01s that are trying to transmit at the same time (mesh networks that aren’t master/slave-based, for example).

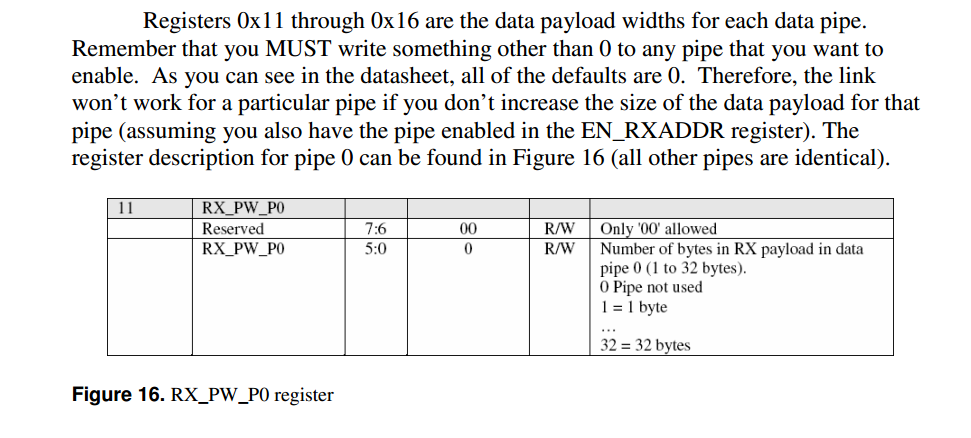
+ One thing to note with this function that is stated in the datasheet is that the 24L01 has to see an in-band signal for at least 128 uS before it will set up the value in the CD register.



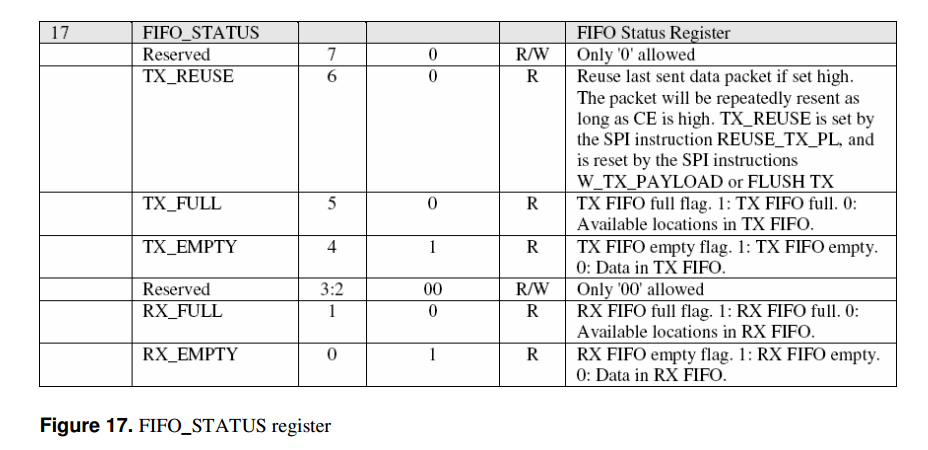
**CD register**







The final register is the FIFO\_STATUS register (all non-reserved bits are read-only). This guy has bits that tell you if the TX and the RX FIFOs are full or empty, which are very useful when you’re sending data at high rates and want to max out the FIFO use. The TX\_REUSE bit, as mentioned above, indicates when the REUSE\_TX\_PL instruction has been executed.



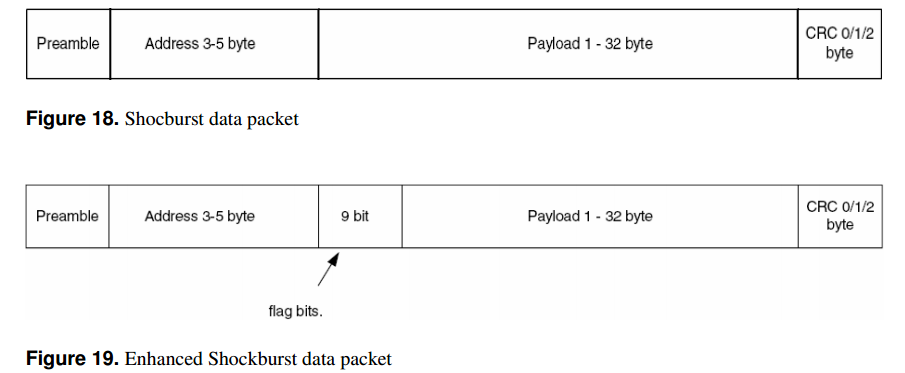
**FIFO Info**

There are FIFOs for both TX and RX modes (as mentioned in the “SPI Instruction Set” section). If you don’t know what a FIFO is, it means “first in, first out,” just like a line at the grocery store works. Both of the FIFOs are three levels deep, meaning that they will hold the three newest packets that have been put into them. Essentially, if you receive three packets in RX mode and you don’t read the payloads, the first (oldest) packet received is pushed out by the newest packet received. The same goes for the TX payload – if you load three packets and don’t transmit them (by executing the aforementioned CE toggle), then the fourth packet will push out the first packet you loaded.

**Data Packet Format**

You might wonder why you would even care about what goes into the air between the 24L01s. For those who have low data rates, this doesn’t really matter too much. However, if you’re trying to get a certain actual on-air data rate, this is very important to know because the amount of overhead bytes that are in the packet will affect greatly your achievable data rate.

To see a drawing of the message format for both Shockburst and Enhanced Shockburst modes, see page 27 of the datasheet, or Figures 18 and 19, respectively, on the following page.

* In both modes, the preamble is sent first, and it is one byte long. It is composed of alternating zero and one bits, which is used to allow the receiver to know that what it is hearing is the beginning of a packet and not just on-air noise. Also in both modes, the next bytes sent are the address bytes. This is set by you, the user, and is between three and five bytes long.

The next thing sent is different between the two modes. In Enhanced Shockburst only, a flag word of nine bits is sent to indicate the message status concerning retransmissions. Only two bits are currently used (a count of resent packets), and the other seven are reserved for future use.

The last half of the packet that is sent is the same in both modes. The first of the fields to be sent in both modes is the payload data. The length of the payload is also set by the user and can be from one to 32 bytes long. The final part of the packet to be sent is the CRC, which is user-settable to zero, one, or two bytes.

**Link Integrity and On-Air Data Rate Calculations**

Now that you know the format of the data packet, we can make some calculations on how much real payload data you can squeeze through your link.

Some people will want to push as much data as possible through the link with less worry about error checking and missed packets (similar to streaming audio over the internet). Other users will want to have error-free packets, but will have to live with a slower data rate. Most users who are reading this will likely be of the persuasion of fewer errors.

The trade-off here is that you need more overhead (longer address, more CRC bytes, and/or acknowledgements/retransmissions) to ensure that you aren’t receiving garbage data packets at the receiver. The problem is that with more overhead, you get a lower payload data rate (assuming the size of the payload stays constant).

Taking this into consideration, here’s an example data rate calculation that shows the maximum possible data rate you can squeeze out of the 24L01. Assume that you can constantly send packets (no delay in between), 100% of packets reach the receiver, the data rate is 2 Mbps, and Shockburst mode is being used. With this configuration, it is possible to have between four and eight bytes of overhead (one byte of preamble, threeto five bytes of address, and zero to two bytes of CRC). Therefore, at the absolutebest utilization, you can send 32 data bytes with 4 bytes of overhead. This gives 88.9% utilization, or 1.78 Mbps of actual data rate (32 bytes of data/36 bytes in total packet).

You can use this type of calculation for any combination of overhead and data bytes to see what you can actually push through your link (at the theoretical maximum, that is). Also remember that the previous calculation is very ideal – in the real world you won’t get 100% packet success rate and there will be a finite delay between packets.