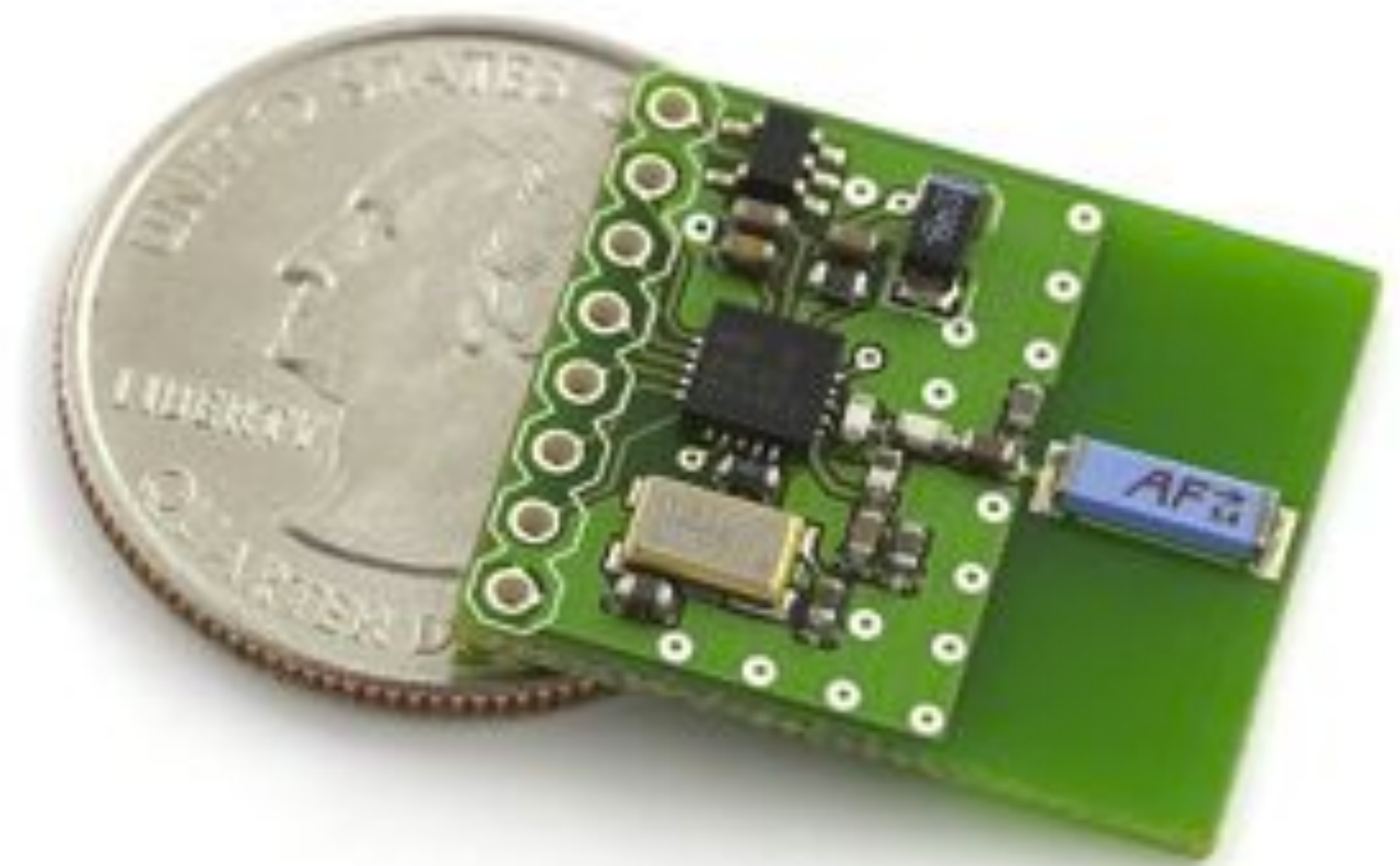
NRF24L01+ Introduction

Wireless Communication

Andrew Kessler
Yuchen Yang

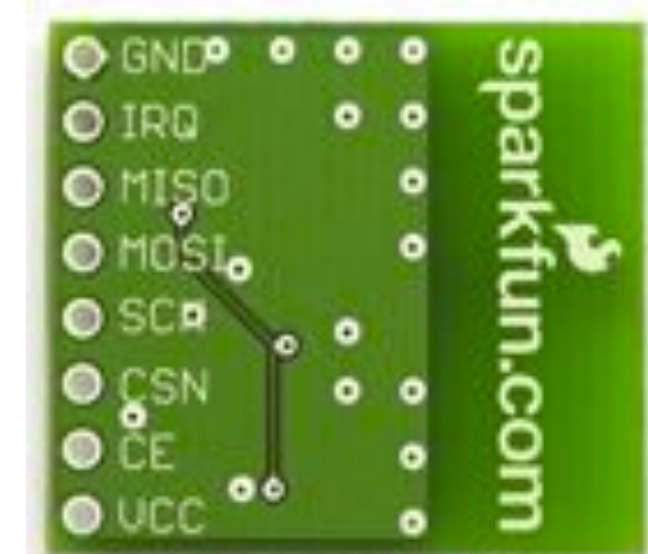
nrf24L01+ Module with Chip Antenna

- breakout of nrf24L01+ 2.4 GHz transceiver
- built in antenna
- SPI compatible
- 125 selectable channels
- 1-32 byte payload
- 6 Receiving Data Pipes



Pin Overview

- CSN - tells chip to begin listening (active low)
- SCK - SPI serial clock
- MISO - Master In, Slave Out
- MOSI - Master Out, Slave In
- IRQ - Interrupt (active low)
- CE - Chip Enable (different meanings depending on mode)
- VCC - Supply Voltage (+3.3 V)
- GND- Ground

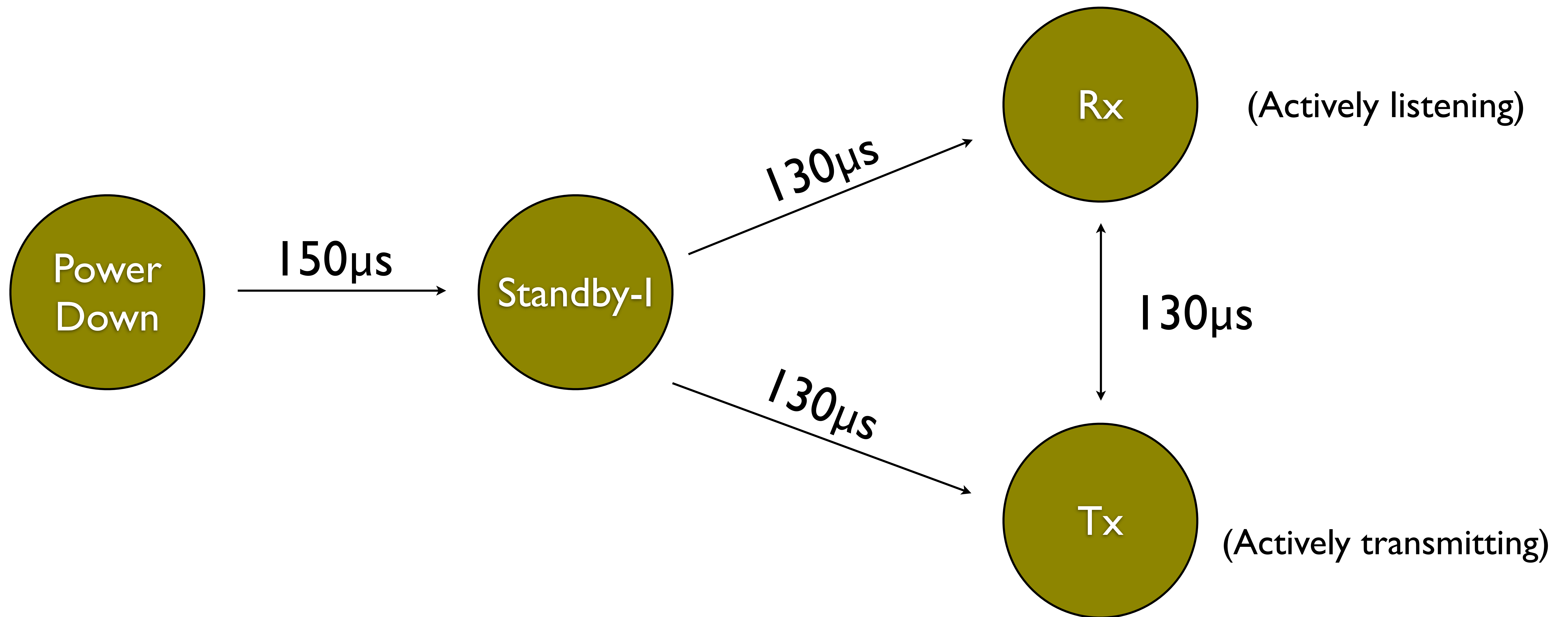


Operating Modes

1. Power Down Mode
    - least current consumption
    - relatively long start up time
2. Standby Mode
    - minimize current consumption while maintaing short start up times
3. Transmit Mode
    - only used when transmitting a packet
    - never stay in longer than 4 ms
4. Receiver Mode
    - constantly reads on-air data searching for a transmission

Mode Transition Times

(Hardware)



To go from Power Down mode to Rx/Tx mode, the nrf24L01+ **must** pass through Standby.

Air Data Rates & Channels

- The air data rate (how fast the chip sends/receives) can be **250kbps**, **1Mbps,** or **2 Mbps**
  - lower data rates give a better receiver sensitivity and allow closer spacing between channel frequencies
  - higher data rates have less current consumption but require a larger spacing
- The nrfL2401+ can operate on channels between **2.4 GHz** and **2.525 GHz**
  - Channels while running at <1Mbps require minimum spacing of **1 MHz**
  - Channels while running at 2 Mbps require minimum spacing of **2 MHz**

Air Data Rates & Channels

- So this allows for:
    - about 125 channels at <1 Mbps
    - about 62 channels at < 2 Mbps
- Two nrf24L01+ chips must have the same channel and air data rate in order to communicate.

Addresses

- Every wireless transmission is preceded by the address of the receiver it is intended for
- A receiver will ignore transmissions that do not contain its address
- Addresses can be 3, 4 or 5 bytes long
- Addresses are user definable, so different receivers can share the same address if desired

Sample Transmission

Actual Data

Intended receiver

Cyclic Redundancy Check

Signal start of transmission

Payloads

- The payload is the actual data you are trying to send in your wireless transmission
- The wireless chips can handle payload sizes from 1-32 bytes
- A receiver will not recognize a transmission unless it is set to the correct payload

Setting up the nrf24L01.c library

- Available from www.DIYembedded.com
- To use:
  - define necessary pin registers and masks in nrf24L01.h
  - implement spi_send_read_byte( )

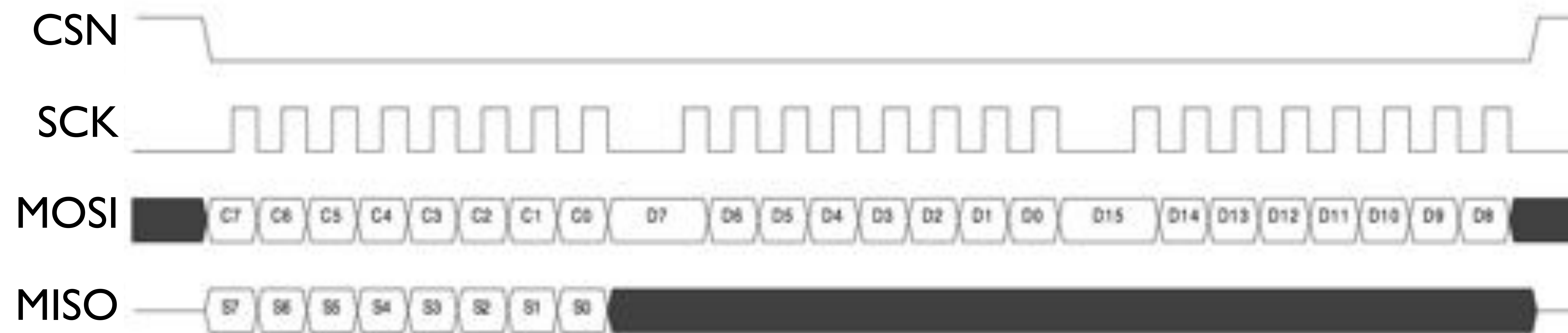
Initialize SPI

```
SpiInitDevice(1,1,0,0);

void SpiInitDevice(int chn, int isMaster, int frmEn, int frmMaster) {

  unsigned int config = SPI_CON_MODE8|SPI_CON_SMP|SPI_CON_ON;  // SPI configuration byte
  if(isMaster)
  {
     config|=SPI_CON_MSTEN;
  }
  if(frmEn)
  {
     config|=SPI_CON_FRMEN;
     if(!frmMaster) {
        config|=SPI_CON_FRMSYNC;
     }
  }
  SpiChnOpen(chn, config, 4); // divide fpb by 4, configure the I/O ports.
}
```

Setting up the nrf24L01.c library

```
unsigned char spi_send_read_byte(unsigned char byte) {

    char  txData, rxData; // transmit, receive characters
    int chn = 1; // SPI channel to use (1 or 2)

    txData = byte; // take inputted byte and store into txData
    SpiChnPutC(chn, txData);     // send data
    rxData = SpiChnGetC(chn);    // retrieve over channel chn the
received data into rxData

    return rxData;
 }
```

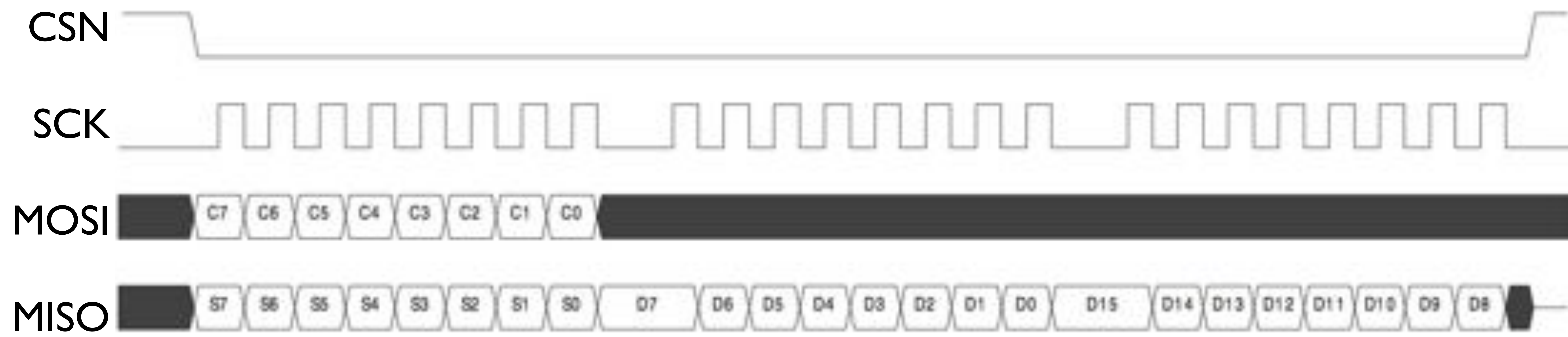
Sending an SPI Command



SPI Write

Cx- command bit
Sx- status register bit
Dx - data bit

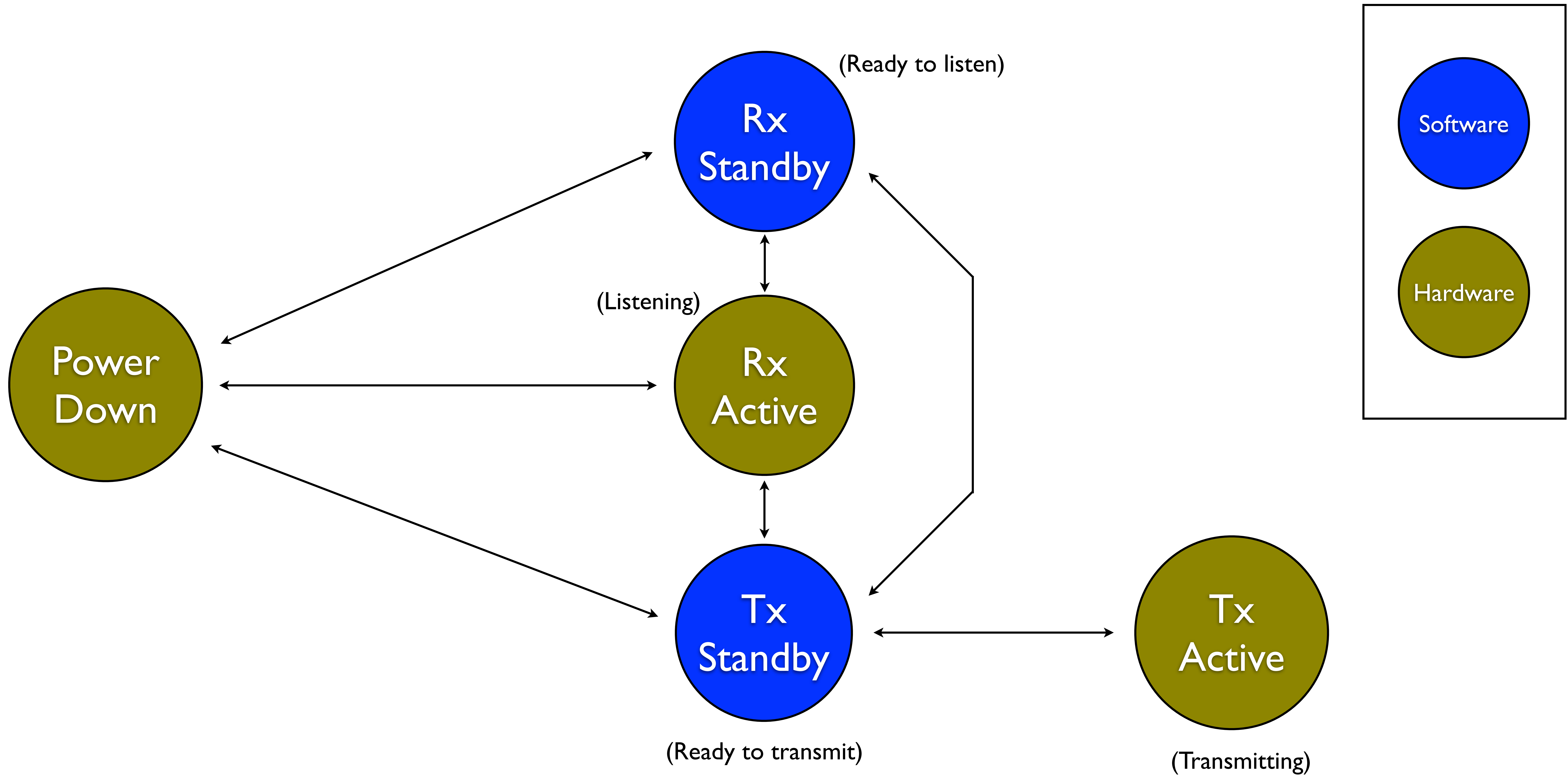
Sending an SPI Command



SPI Read

Cx- command bit
Sx- status register bit
Dx - data bit

Modes in the nrf24L01+ Library



Initializing the nrf24L01+

1. **`nrf24L01_intialize(...)`**
    - most control; lets you specify every register
    - 21 arguments
2. **`nrf24L01_intialize_debug(`**`bool rx, unsigned char payload_width, bool aa_on`**`)`**
    - lets you specify Tx/Rx, payload width, and whether or not to turn on auto-acknowledge
    - everything else is set to default
    - limits you to using pipe 0
    - Channel = 2.403 GHz

Using the Library (Tx)

```
bool is_rx = false; //receiver?
int width= 1; //payload width
bool auto_ack = false; //auto acknowledge on?
nrf24l01_initialize_debug(is_rx, payload_width, auto_ack);
char data = ‘a’;
nrf24l01_write_tx_payload(&data, width, true); //transmit


//wait until it has been transmitted
while(!(nrf24l01_irq_pin_active() && nrf24l01_irq_tx_ds_active()));
nrf24l01_irq_clear_all(); //clear all interrupts
```

Using the Library (Tx)

Transmitting Payloads >1

```
bool is_rx = false; //receiver?

int width= 10; //payload width

bool auto_ack = false; //auto acknowledge on?

nrf24l01_initialize_debug(is_rx, payload_width, auto_ack);

char data[width];

/... Fill Array .../

nrf24l01_write_tx_payload(data, width, true); //transmit


//wait until it has been transmitted

while(!(nrf24l01_irq_pin_active() && nrf24l01_irq_tx_ds_active()));

nrf24l01_irq_clear_all(); //clear all interrupts
```

Using the Library (Tx)

Transmitting Payloads >1

- Your payload width can be shorter than size of the data array, but not bigger
- So if you plan on changing the payload, best practice is to initialize data array to a length of 32
- If you change the payload, you must re-initialize the nrf24L01+ before the new payload is recognized

Using the Library (Rx)

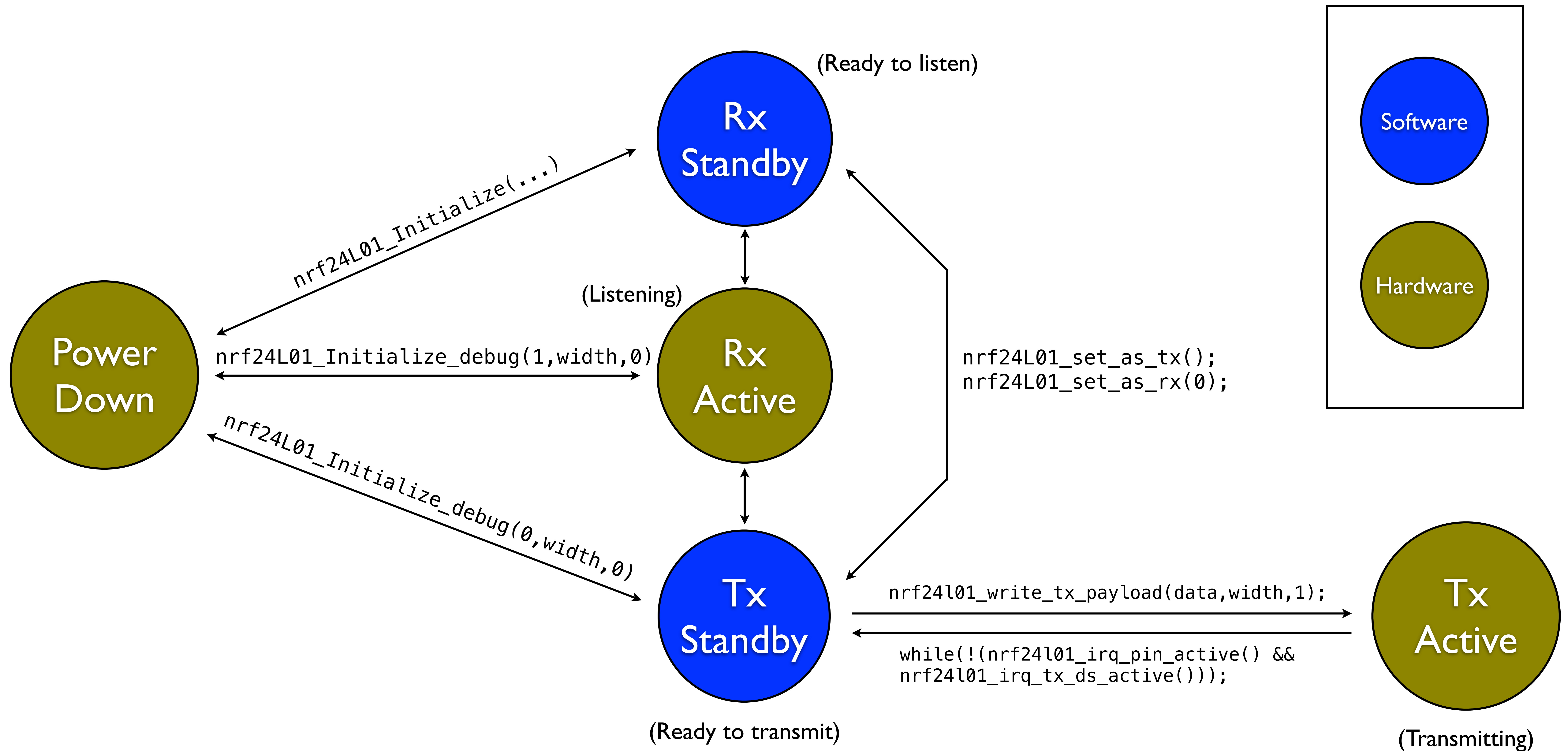
```
bool is_rx = true; //receiver?

int width= 1; //payload width

bool auto_ack = false; //auto acknowledge on?

nrf24l01_initialize_debug(is_rx, payload_width, auto_ack);

//wait until a packet has been received

while(!(nrf24l01_irq_pin_active() && nrf24l01_irq_rx_dr_active()));


nrf24l01_read_rx_payload(&data, width); //read into data variable

nrf24l01_irq_clear_all(); //clear all interrupts
```

Using the Library (Rx)

Receiving Payloads >1

```
bool is_rx = true; //receiver?

int width= 10; //payload width

bool auto_ack = false; //auto acknowledge on?

nrf24l01_initialize_debug(is_rx, payload_width, auto_ack);

char data[width];

//wait until a packet has been received

while(!(nrf24l01_irq_pin_active() && nrf24l01_irq_rx_dr_active()));


nrf24l01_read_rx_payload(data, width); //read into data array

nrf24l01_irq_clear_all(); //clear all interrupts
```

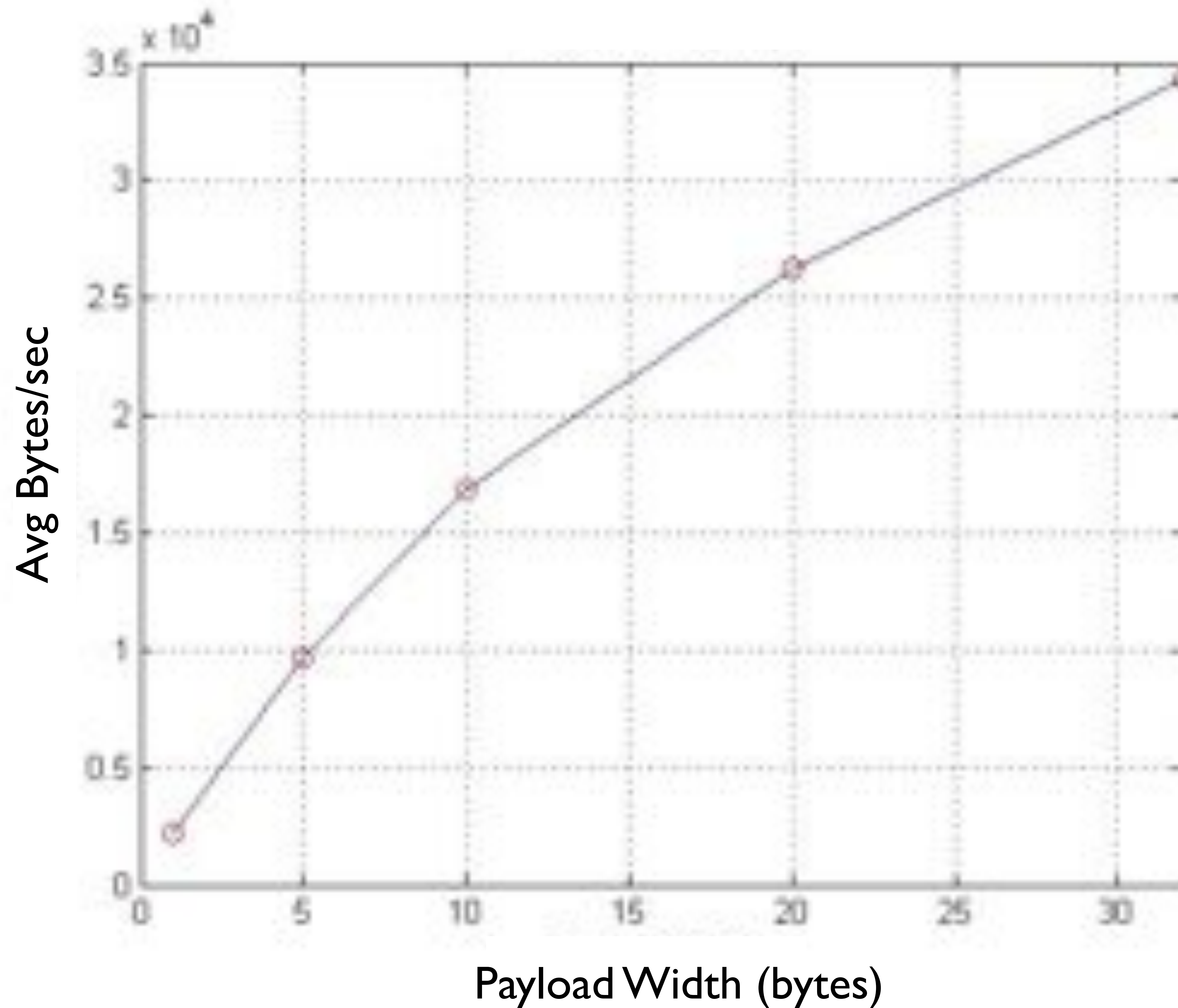
Modes in the nrf24L01+ Library



Helpful Functions

- `nrf24l01_power_down();`
- `nrf24l01_set_as_rx(bool rx_active_mode);`
    - change to Rx mode (active if true), keeping everything else constant
- `nrf24l01_set_as_tx();`
    - change to Tx Standby mode (keeping everything else constant)
- `nrf24l01_set_rf_ch(unsigned char channel);`
    - set RF channel to new value
- `nrf24l01_rx_pipe_enable(unsigned char rxpipenum);`
    - open a new receiver pipe
- Most functions (receive_payload, transmit_payload, etc.) return the current value of the status register

Payload Efficiency



Using Multiple Pipes

- Pipes essentially allow one nrf24L01+ chip to act as (up to) 6 different receivers at once
- Each active pipe needs a different address
- Different pipes may have a different payload width
- But all pipes must operate on the same channel (frequency)
- All pipes feed into the same 3 packet deep queue

Using Multiple Pipes

- Pipes 0 and 1 may have addresses completely different from each other
- But the addresses for pipes 2, 3 and 4 may only differ from that of Pipe 1 by the Least Significant Byte.

	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Data pipe 0 (RX_ADDR_P0)	0xE7	0xD3	0xF0	0x35	0x77
Data pipe 1 (RX_ADDR_P1)	0xC2	0xC2	0xC2	0xC2	**0xC2**
Data pipe 2 (RX_ADDR_P2)	0xC2	0xC2	0xC2	0xC2	**0xC3**
Data pipe 3 (RX_ADDR_P3)	0xC2	0xC2	0xC2	0xC2	**0xC4**
Data pipe 4 (RX_ADDR_P4)	0xC2	0xC2	0xC2	0xC2	**0xC5**
Data pipe 5 (RX_ADDR_P5)	0xC2	0xC2	0xC2	0xC2	**0xC6**

Figure 13. Addressing data pipes 0-5

Using Multiple Pipes

```
status = nrf24l01_read_rx_payload(&data, 1);
pipe = nrf24l01_get_rx_pipe_from_status(status);

switch(pipe) {
case 0:
//process case 0
break;

case 1:
//process case 1
break;

case 2:
//process case 2
break;

default:
//process any other pipe
}
```