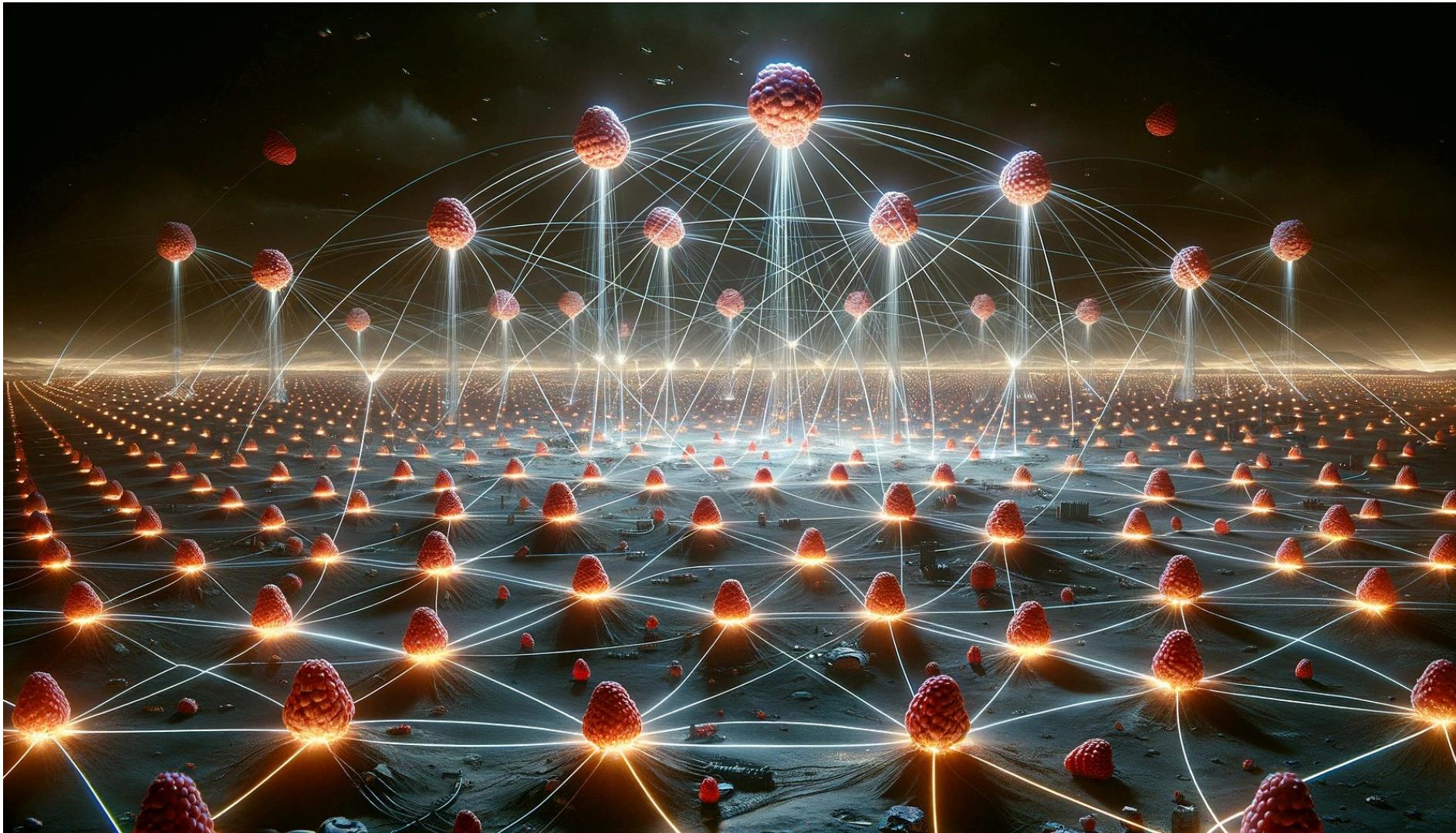


Lab 14: NRF (140e:win26)

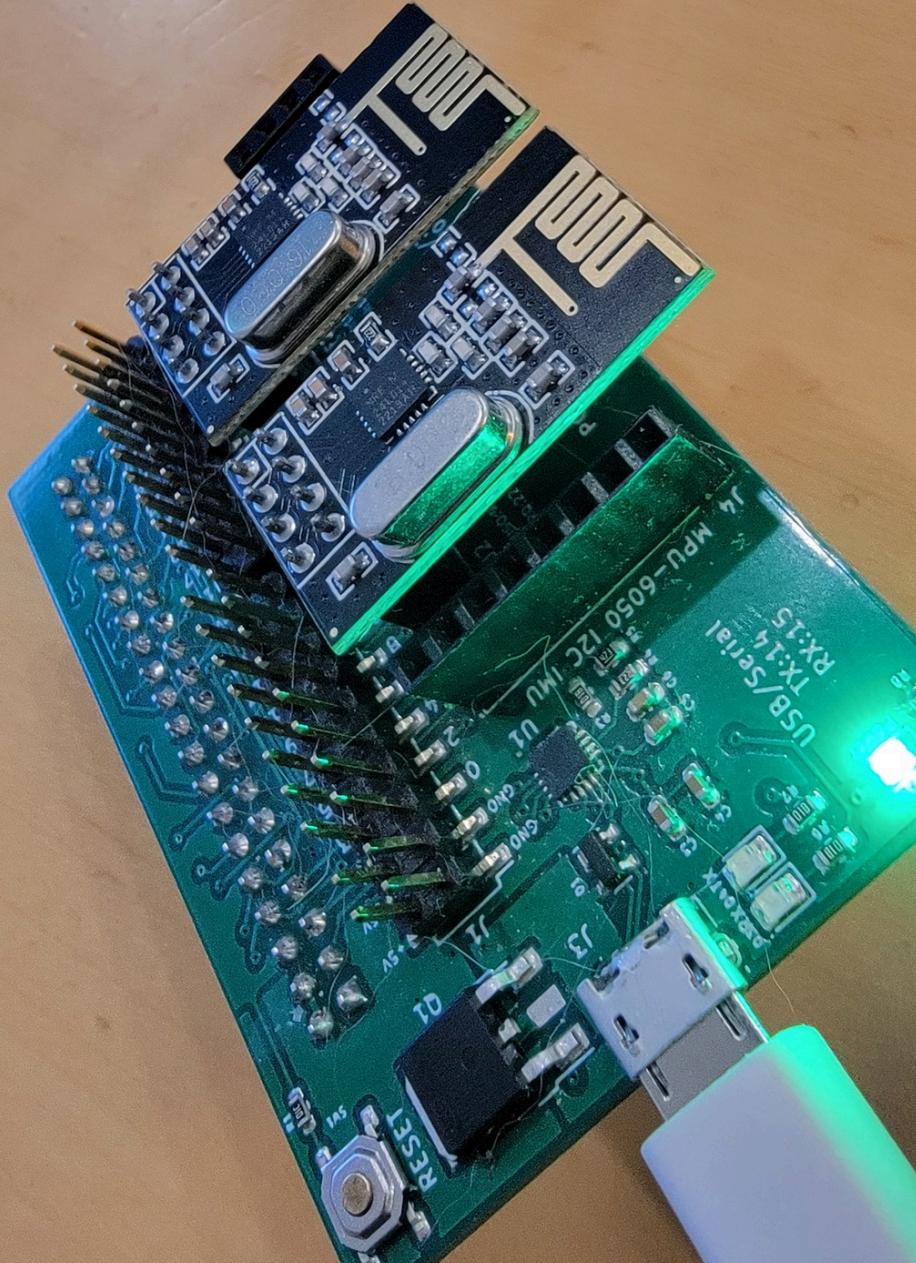


today: use two NRFs to send + receive to yourself.

- Various high bits:
 - 1-32 byte packets with CRC + dest address
 - NRF hardware can do timeout+ack retransmission
 - but if need reliable: you still have to do as well.
 - up to 2mb/s, with standard RF tradeoff:
 - faster = shorter range, but less chance for collision.
 - 2^7 different channels (+ can dynamic hop)
 - Separate device, doesn't see power cycle on reboot.
- Huge gotcha:
 - i. TX and RX are separate modes.
 - ii. Cannot RX packet if in TX mode.

Big ups to Parthiv board!

- Today is a big reason we use Parthiv
 - Pi talks to NRF using SPI
 - SPI faster than I2C, but 4 wires vs 2.
 - total wires: 4 SPI + 3v + gnd + CE
 - 7 wires x 2 ends x 2 NRF = 28 endpoints
 - $\text{Pr}(\text{misconnect} \mid \text{loose}) \sim 1$
- No Parthiv-board?
 - Then can't do lab w/ 93 students.
 - (240lx: he teaches a pcb lab.)



Big datasheet = big starter code (sorry)

- What you modify:
 - `nrf-driver.c`.
- What you have:
 - `nrf.h` : defines datastructure and configuration.
 - `nrf-hw-support.h` : many accessors for NRF registers.
 - `nrf-hw-support.c` : SPI routines, and `nrf_dump` (you'll use this alot)
 - `nrf-public.c` : wrapper over the low level nrf driver.
 - `nrf-test.h` : helpers for testing.
- Great extension is to rewrite! or do in daniel mode.

The state machine (fixed-pt: stay in RX)

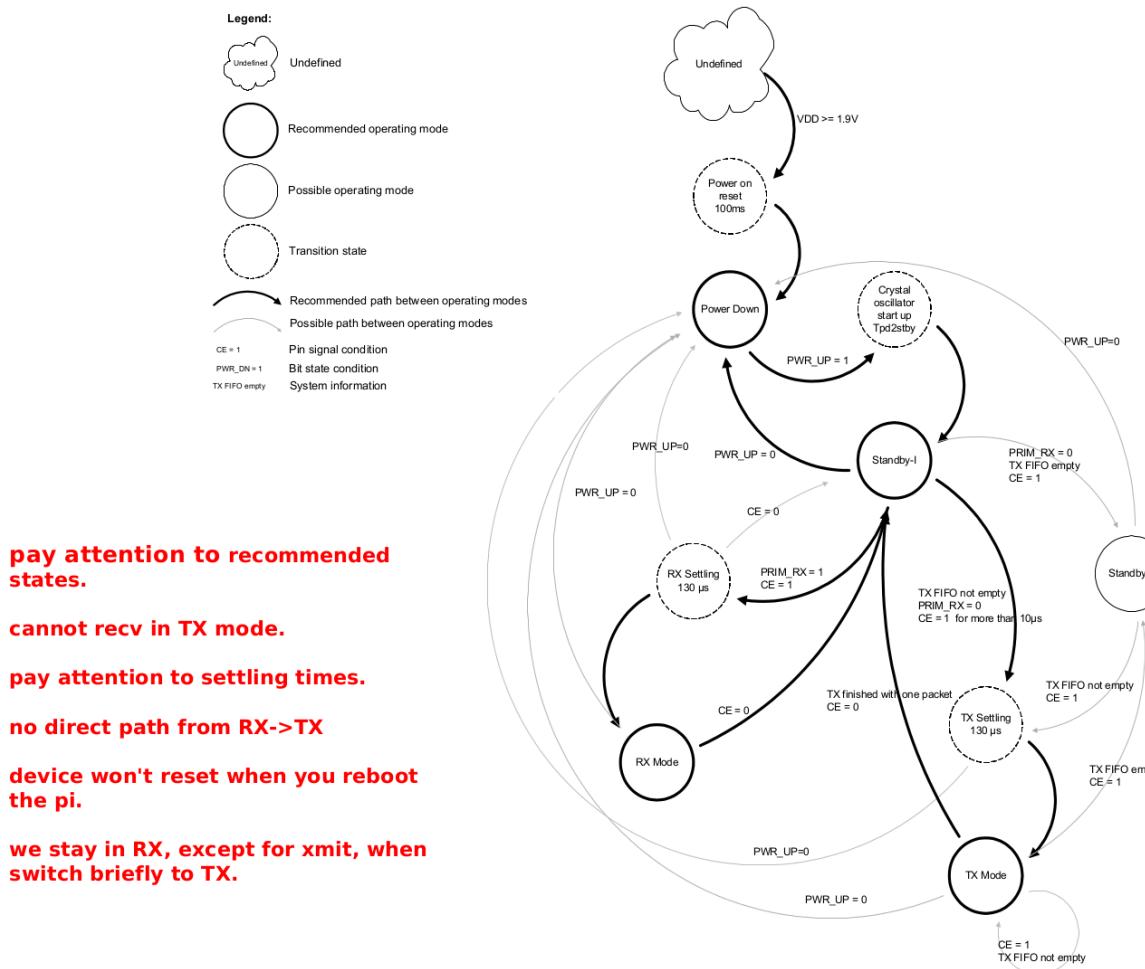
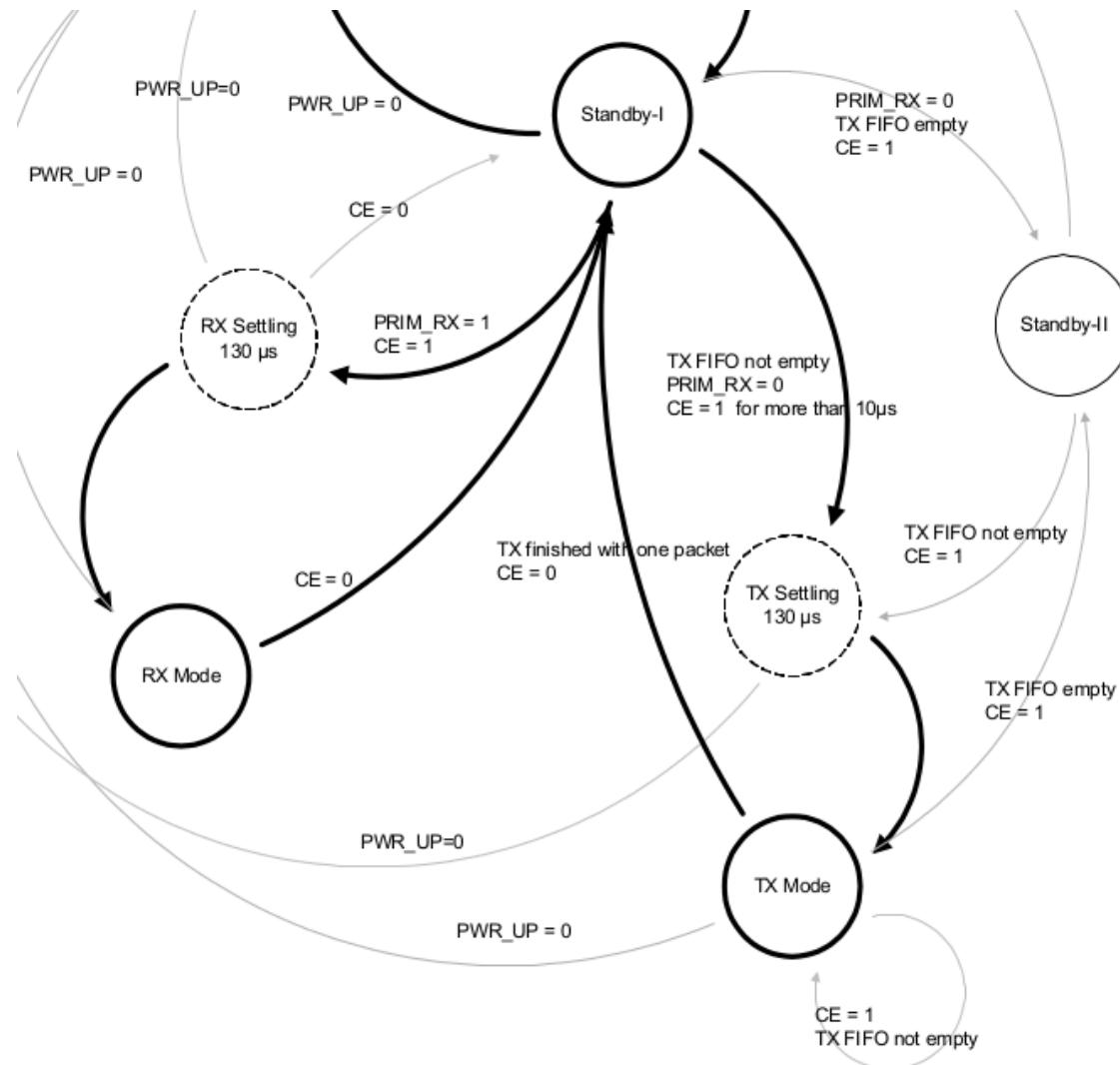


Figure 4. Radio control state diagram

To transmit: do RX->Standby-I->TX



How to SPI

- Same metaphor as GPIO: read and write "registers"
 - but instead of load/store to address
 - use SPI to read/write 8-bit registers with 1+ bytes
 - more honest: explicit i/o.
 - Read back what you wrote: Catches many errors.
 - We own device: don't RMW during init

```
// nrf-hw-support.h: has many others.  
// read 8 bits of data from <reg>  
uint8_t nrf_get8(const nrf_t *n, uint8_t reg);  
// write 8 bits of data to <reg>: returns <CONFIG>  
uint8_t nrf_put8(nrf_t *nic, uint8_t reg, uint8_t v);  
// write 8-bit <v> to <reg> and then check get8(reg) = <v>.  
uint8_t nrf_put8_chk(nrf_t *nic, uint8_t reg, uint8_t v);
```

Huge cheat code: read-back any value you set

- As with any device: if you set a deterministic register: read it back.
- Catches many human errors trivially:
 - register discards values in ways you didn't expect.
 - register uses a different size than expected
 - you misunderstood the helper routines
 - Device mis-configuration:
 - e.g., SPI clock was too fast so the device couldn't keep up.
- Catches many broken hardwares trivially:
 - if one of your NRF devices (or Parthiv board slots) is dead.

Common symptom: MAX_RT

- Seeing a lot of MAX_RT (p59)?
 - i. if consistent:
 - likely send/receive pipes don't match
 - e.g., different packet size or ack settings
 - ii. if sporadic:
 - collisions: change the channel you're using.

Mistake: NRF addresses are > than one byte!

- You must use `nrf_get_addr` and `nrf_set_addr` methods (nrf-hw-support.c)
 - They do sanity checking
 - They use the right SPI calls to set and get multiple-bytes.
- Just setting addresses with a 8-bit SPI cmd = truncate + not work.

Mistake: Hard-coding a value in `nrf_init`.

- Hard-coding value = very hard to debug
 - Will work fine on initial tests (we don't randomize).
 - Then won't work when you use different value
 - For example changing RX or TX addresses
- Each year this mistake causes groups to waste hour+.

Mistake: not setting CE

- If:
 - you finished `nrf_init`
 - your `nrf_dump` matches the staff
- But:
 - you can't TX/RX/ACK anything
- Then:
 - Make sure you're setting the CE pin correctly
 - Is a GPIO pin, so you have to control it manually.

Mistake: not waiting for settling!

- P22: During boot up: `delay_ms(100)` until in "Power down"
- P22: During Standby-I->RX: `delay_us(130)`
 - avoid forbidden state.
- P24: After setting "power up" during init: `delay_ms(5ms)`
- NOTE: Can only change config in Standby-I or Power-down

Some Rules

1. Cannot receive in TX mode.
2. NRF has own power:
 - Reboot has no power on its state!
 - Can't rely on registers having "reset values"
3. Just b/c has ACKs, doesn't mean transmit "solved"
 - Retrans can be exceeded.
4. Just b/c you see an ACK does not mean other code received.
 - it might have not pulled it off RX FIFO fast enough
5. Just b/c you see an ACK does not mean other code acted.
 - might have dropped later, or crashed, or bug.
 - For a long discussion of 4/5: the End-to-end principle

Big datasheet but mainly a few pages

- State machine figure: p22-26
- Shockwave: figure p41, TX p75, RX p76
- SPI commands: p51
- TX,RX FIFO: p56
- Registers: 57-63
- Cheatsheet: ./CHEATSHEET-nrf24l01p.md

Inf Extensions!!!

- make driver as simple as possible.
- make driver as fast as possible.
- handle multiple pipes (change `nrf_t`)
- do as many NRFS as possible (final project)
- send code

