Beginner workshop

https://embedded-trainings.ferrous-systems.com/

Please do the setup steps

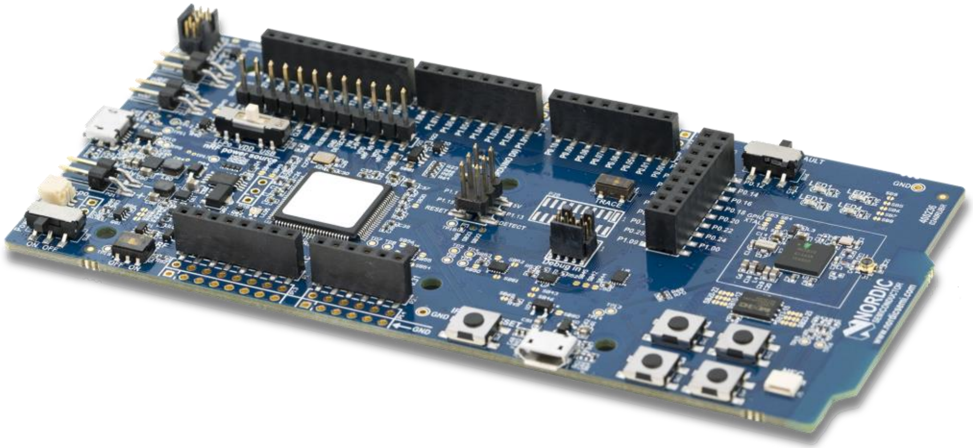
- if you haven't already
  - https://embedded-trainings.ferrous-systems.com/preparations.html
  - https://embedded-trainings.ferrous-systems.com/tooling-check.html
- starter code and slides are here
  - https://github.com/ferrous-systems/embedded-trainings-2020

Agenda

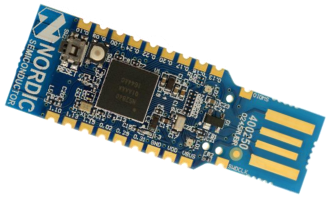
- `no_std` programs
- Embedded Rust tooling
- Using a Hardware Abstraction Layer
- Using the Radio on the nRF52840 to solve a puzzle

The hardware

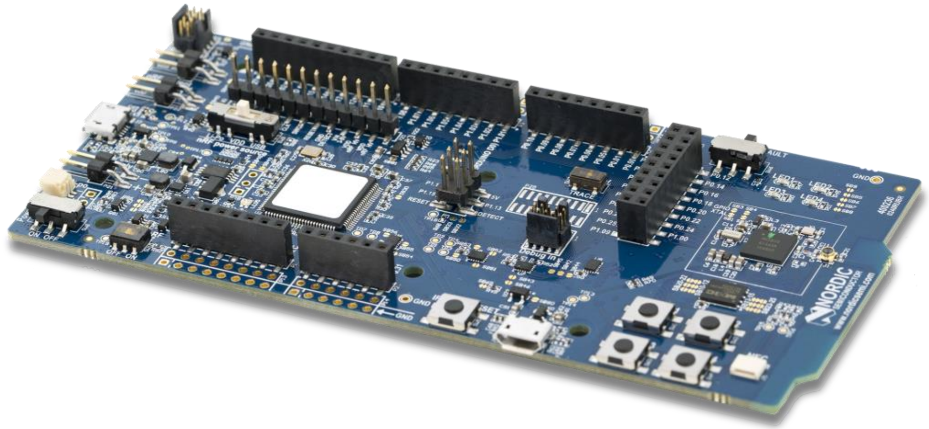
- nRF52840 Development Kit
  - USB port J2: J-Link debugger
  - Connect a cable to it



- nRF52840 Dongle
  - No on-board debugger



nRF52840



- ARM Cortex-M4F processor
- 1 MB of Flash
- 256 KB of RAM
- USBD: USB 2.0 Full-Speed device
- RADIO: IEEE 802.15.4 and Bluetooth Low Energy compatible

Parts of a `no_std` program

Training Materials: section 3.1

📁: `beginner/apps`

📝: `src/bin/hello.rs`

- `#![no_std]`: `std` API is not available but `core` is
- `#![no_main]`: custom entry point
- divergent `main` function

Cross compiling

Training Materials: section 3.2

📂: `beginner/apps`

📝: `src/bin/hello.rs`

- `cargo build --bin hello`
- Compilation target defined in `.cargo/config.toml`
- Output ELF is in `target/thumbv7em-none-eabihf/debug`

Analysis: Binary size

Training Materials: section 3.3

📁`:beginner/apps`

📝`:src/bin/hello.rs`

- Strip ELF metadata to get *program size on target*, not ELF file size
- `cargo size --bin hello -- -A`
  - First+second column is size in Flash
  - Second+third column is static RAM usage

Running a program

Training Materials: section 3.4

🗀: `beginner/apps`

📝: `src/bin/hello.rs`

- `probe-run` : Custom Cargo runner (set in config.toml)
- ☑ Click "Run" button in VS code
  OR run `cargo run --bin hello` if not using VS Code)
- On `asm::bkpt()` : Cargo runner prints stack backtrace and exits
- ☑ Try changing the log statement and re-running the program

Panicking behavior

Training Materials: section 3.5

📁: `beginner/apps`

📝: `src/bin/panic.rs`

- No default behavior in `no_std` programs
- Must pick one
  - Use a panic handler crate like `panic-log`
  - Or write a `#[panic_handler]` function
- ☑ try changing `panic_log`'s `#[panic_handler]` function

Hardware Abstraction Layer (HAL) - LED

Training Materials: section 3.6

📂: `beginner/apps`

📝: `src/bin/led.rs`

- Get HAL API documentation with `cargo doc -p dk --open`
- 👀 `Led.on()` and `Led.off()` control the LEDs
- ☑ try turning on/off different LEDs
- ☑ try uncommenting the `set_log_level` statement

- 🔎 use "leds > Go to Definition" to explore HAL internals

HAL - Timer

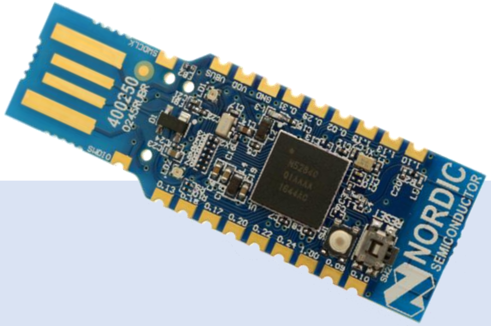
Training Materials: section 3.7

📁: `beginner/apps`

📝: `src/bin/blinky.rs`

- `Timer.wait` can be used to create delays
- ☑ try changing the delay value
- 🔎 explore the `timer.wait()` implementation in `boards/dk/src/lib.rs`

Using the Dongle



Training Materials: section 3.8

📁: `boards/dongle`

- ☑ Disconnect the DK board for now
- ☑ Press reset button on the Dongle to put it in bootloader mode
- The Dongle will pulsate its red LED in bootloader mode

-
  ```
  $ cd boards/dongle
  $ nrfdfu loopback
  $ cargo xtask serial-term
  ```
  to display the Dongle's logs
- ☑ check for interference; use `change-channel` if there is

Radio out

Training Materials: section 3.9
📁: `beginner/apps`
📝: `src/bin/radio-send.rs`

- ☑ reconnect the Development Kit & run `radio-send.rs`
- Check `serial-term` for new output
- LQI: Link Quality Indicator. Higher = better
- ☑ Try:
  - Using a different Channel
  - Changing the TX power
  - Increasing the distance between the DK and the dongle

Radio in

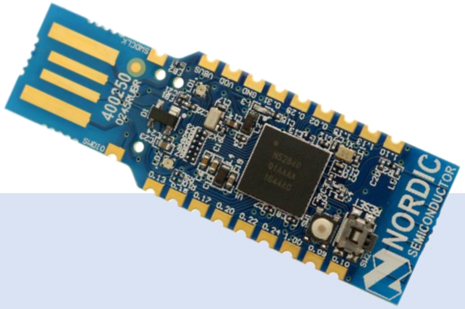
Training Materials: section 3.10

🗁: `beginner/apps`

📝: `src/bin/radio-recv.rs`

- The Dongle responds to each incoming packet
- The response contains the received data but reversed
- ☑ Try: inserting a delay between `send` and `recv_timeout`

Reflashing the Dongle



Training Materials: section 3.11

- ☑ Press the reset button on the Dongle to put it in bootloader mode

-
  ```
  $ cd boards/dongle
  $ nrfdfu puzzle
  $ cargo xtask serial-term
  ```
- Check: `serial-term` output should have ”`app=puzzle`”

- 👀 note that the channel has changed

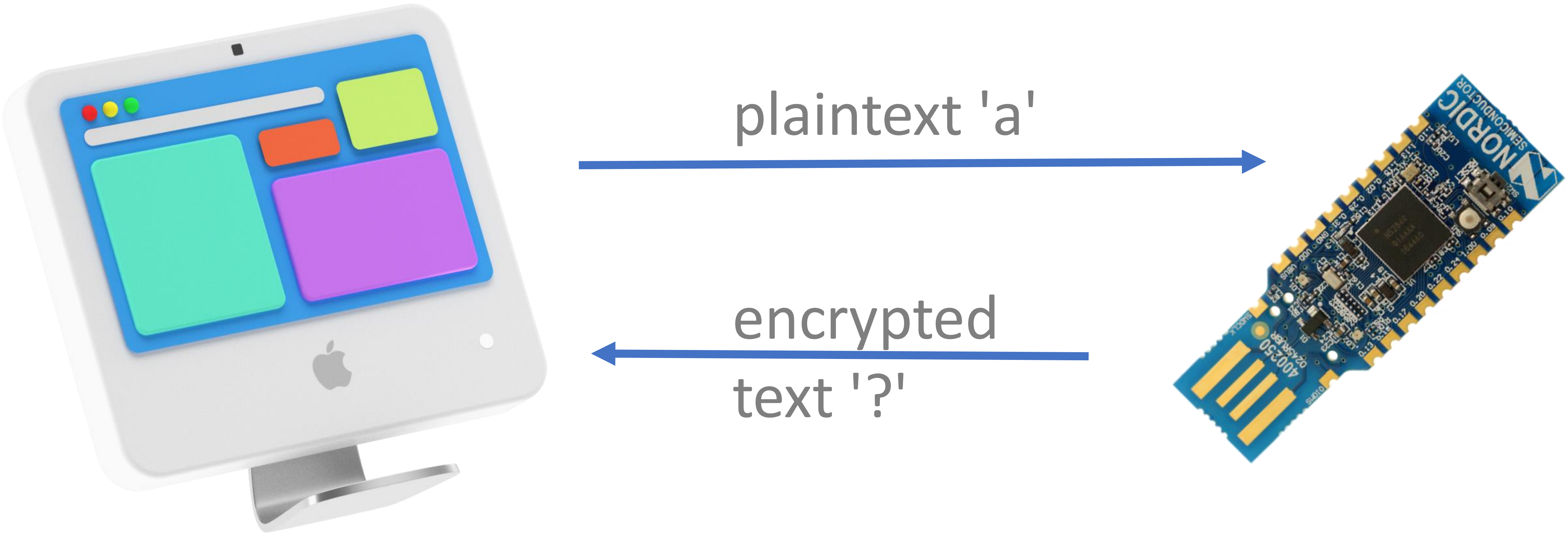
Radio puzzle

Training Materials: section 3.11

📁: `beginner/apps`

📝: `src/bin/radio-puzzle.rs`

- Dongle holds a string encrypted via single-letter substitution
- Your task is to decrypt it
- Dongle's response depends on packet size
  - 0 bytes: answers with encrypted string
  - 1 byte: mapping from plaintext letter to the ciphertext letter
  - Else: answers with "correct" if the packet contained the decrypted string



plaintext 'a'

encrypted text '?'

illustrations by craftwork.design

Radio puzzle help

Training Materials: section 3.12

- Suggested steps:
  1. Send a 1 letter packet and print response to get a feel for how the mapping works
  2. Get familiar with `heapless::LinearMap`. Do some insertions and look ups
  3. Get mappings from the radio and insert them into the dictionary
  4. Get familiar with the `heapless::Vec` API to store deciphered chars in it
  5. Retrieve the ciphertext from the Dongle; get familiar with iterating it
  6. Do the reverse mapping to decrypt the message
  7. Send plaintext to the Dongle for confirmation

🔎 You can follow incremental solutions to these steps in src/bin

Things for you to check out

Training Materials: section 3.13

- 802.15.4 experiments: energy detection, collision avoidance and WiFi coexistence
  - See section 3.14 of the workbook for details
- Memory safe interrupt handling
  - Check the concurrency chapter of the embedded Rust book
  - Check the Real-Time Interrupt-driven Concurrency (RTIC) framework