

Advanced 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

- How to work with hardware registers
- How to handle external events
- How to debug evented applications
- How to test ``no_std`` code
- You will write USB firmware from scratch

The Hardware

- nRF52840 Development Kit
 - USB port J2: J-Link debugger
 - USB port J3: nRF52840
- 2 USB cables
- Connect both

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

Code organization

- **Folder:** advanced
- `firmware/` : `no_std` code set up for cross compilation
- `host/` : `std` code
- `common/` : **shared** `no_std` code; can be tested on the host

Listing USB devices

- Run: `cargo xtask usb-list`
- Output: "J-Link on the nRF52840 Development Kit"
- Goal: "nRF52840 on the nRF52840 Development Kit" in the output

Hello, world!

Training Materials: section 4.3

 : advanced/firmware

 : src/bin/hello.rs

-  Click the "Run" button within VS code
 - (if not using VS code, run `cargo run --bin hello`)

API documentation

Training Materials: section 4.4

 : `advanced/firmware`


- Run: `cargo doc -p dk --open`
- Also check the `log crate` (left sidebar)

Hello from RTIC

Training Materials: section 4.5

 : advanced/firmware

 : src/bin/rtic-hello.rs

- main is now split in `#[init]` and `#[idle]`
- `#[init]` code runs with interrupts disabled
-  check `target/rtic-expansion.rs`

Dealing with registers

Training Materials: section 4.6

 : `advanced/firmware`

 : `src/bin/events.rs`

- Peripherals are structs e.g. `POWER`
- Registers are struct fields e.g. `intenset`
- API generated by `svd2rust` from SVD file
- Run: with J3 cable disconnected
- try running with cable connected

Event handling

Training Materials: section 4.7

 : `advanced/firmware`


 : `src/bin/events.rs`


- Run: with J3 cable disconnected
- Then: connect cable to port J3
- try running the program with the cable initially connected
- try removing the `INTENSET` write
- try adding `loop { }` at the end of `init`

Task state

Training Materials: section 4.8

 : `advanced/firmware`


 : `src/bin/resource.rs`

- Use an RTIC resource to add state to the task
- Resource is initialized in `#[init]` - POWER is moved into task
- Task can access the resource by reference on each invocation
-  Add a counter to the task and print its current value

USB enumeration

Training Materials: section 4.9

 : `advanced/firmware`

 : `src/bin/resource.rs`


- **USB device states:** `Default`, `Address`, `Configured`
- Enumeration moves the device from the `Default` to the `Address` state
- Sequence of events:
 - USB reset
 - `GET_DESCRIPTOR` request
 - `SET_ADDRESS` request

Dealing with USB events

Training Materials: section 4.10

 : advanced/firmware

 : src/bin/usb-1.rs

- USB events: USBRESET, EPOSETUP, EPODATADONE
- Run: with cable connected to J3
-  Goal: reach EPOSETUP case

USB endpoints

Training Materials: section 4.11

- For multiplexing: like TCP ports but with direction (IN / OUT)
- Identified by address and direction e.g. EP0IN, EP2OUT
- 4 types: Control, Bulk, Isochronous, Interrupt
- Control endpoint 0 is mandatory

Control transfer

Training Materials: section 4.12

- Data transfer over a control endpoint
- 3 stages
 - SETUP: header that identifies the control request
 - DATA: optional stage
 - STATUS: device acknowledges (or not) the request

SETUP stage

Training Materials: section 4.13


 : `advanced/firmware`


 : `src/bin/usb-2.rs`

- Run: with cable connected to J3
- EPOSETUP event = SETUP data received
- SETUP data is stored in registers like `BMREQUESTTYPE`
- parse the `GET_DESCRIPTOR` request in `common/usb`
- pass SETUP data to the parser in `usb-2`

Unit testing

Training Materials: section 4.13

 : `advanced/common/usb` (open in a separate VSCode window)

 : `src/lib.rs`

- SETUP data parser
- "Test" button in VS code
- (or run `cargo test` if not using VS code)

Device descriptor

Training Materials: section 4.14

- Sent in response to GET_DESCRIPTOR/Device request
- Contains info about the device:
 - product ID
 - vendor ID
 - number of configurations, etc.

Configurations

Training Materials: section 4.14

- Configuration = operating mode
- Single configuration example: USB mouse
- Two configurations example:
 - config #0: USB printer
 - config #1: Device Firmware Update mode
- We'll report one configuration

DATA stage

Training Materials: section 4.15

 : advanced/firmware

 : src/bin/usb-3.rs

- On GET_DESCRIPTOR/Device request: send device descriptor
- Use `dk::usb::Ep0In` abstraction
 - `start()`, starts the transfer
 - `end()`, must be called on EP0DATADONE done

Supporting more standard requests

Training Materials: section 4.17

 : `advanced/common/usb`

 : `src/bin/lib.rs`

- **TODO: GET_DESCRIPTOR Configuration**
- **TODO: SET_CONFIGURATION**
- **Solutions in `advanced/common/usb/`**

Error handling

Training Materials: section 4.18

 : advanced/firmware

 : src/bin/usb-4.rs


- ep0setup refactored: returns Result
- On Err: stall the endpoint

Device state

Training Materials: section 4.19

 : advanced/firmware

 : src/bin/usb-4.rs

- State is now part of the task state
-  update handling of USBRESET event

Stalling the endpoint

Training Materials: section 4.20

- Device action to reject a host request
- Use it for invalid and unsupported requests
- API: `dk::usbd::ep0stall()`

SET_ADDRESS

Training Materials: section 4.21

 : advanced/firmware

 : src/bin/usb-4.rs

- Entirely handled by the peripheral
- No action required in software
- Section 9.4.6 of the USB spec describes how to handle this

Configuration descriptor

Training Materials: section 4.22

- *Total length* of the configuration
- Number of interfaces
- The configuration's (non-zero) value
- Described in section 9.6.3 of the USB specification

Interfaces

Training Materials: section 4.22.2

- Interface = USB function
- At least one interface per configuration
- Single interface example: USB mouse with HID interface
- Two interface example: nrf52840
 - iface #0: TTY ACM (virtual COM/serial) for logging
 - iface #1: HID to control the radio from host

Interface descriptor

Training Materials: section 4.22.3

- The interface's number (zero-based index)
- Number of endpoints
 - Does not include endpoints 0 IN or 0 OUT
- Described in section 9.6.5 of the USB specification

Endpoint descriptor

Training Materials: section 4.22.4

- Will not be used in this workshop
- Described in section 9.6.6 of the USB specification

GET_DESCRIPTOR/CONFIGURATION

Training Materials: section 4.22.5

 : advanced/firmware

 : src/bin/usb-4.rs

- Check requested index
- Respond with a single packet that contains
 - Configuration descriptor
 - Interface descriptor

SET_CONFIGURATION (Linux & Mac OS)

Training Materials: section 4.23

 : advanced/firmware

 : src/bin/usb-4.rs

- May be sent after SET_ADDRESS
- OK to stall for now

Idle state

Training Materials: section 4.24

- After you reach the Addressed state the bus will go idle
- Compare your logs
 - linux-enumeration.txt
 - macos-enumeration.txt
 - windows-enumeration.txt
- Run: `cargo xtask usb-list`
- Output: "nRF52840 on the nRF52840 Development Kit"

Inspecting the descriptors

Training Materials: section 4.25

 : `advanced/host/print-descs`

 : `src/main.rs`

- **Run:** `cargo run`
- **Output:** Device, configuration and interface descriptors

Getting it configured (Windows)

Training Materials: section 4.26

- Change driver using Zadig
- Run modified `print-descs` program
 - Uncomment `open` line

SET_CONFIGURATION

Training Materials: section 4.26

 : advanced/firmware

 : src/bin/usb-4.rs

- Section 9.4.7 of the USB spec explains how to handle this

Final checkpoint

Training Materials: section 4.26

- Should have reached the Configured state
- Compare logs
 - linux-configured.txt
 - macos-configured.txt
 - windows-configured.txt

Things for you to check out

[Training Materials](#): section 4.27

- String descriptors, how about adding string descriptor support to your firmware?
 - See the workbook for suggested steps
- The RTIC book, RTIC has many features we have not covered
- usb-device, a device-agnostic USB framework