Advanced workshop

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

Oxidize Global – 17.07.2020

Do the setup steps (if you haven't already)

→ https://oxidizeconf.com/oxidize-global-setup/

Install some more tools

- Folder: `tools`
- Run: `cargo install --path usb-list`
- Run: `cargo install --path dk-run`
- Leave those processes running in the background

Agenda

- How to work with hardware registers
- How to handle with 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-M4 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: `usb-list`
- Output: "J-Link on the nRF52840 Development Kit"
- ✅ Goal: "nRF52840 on the nRF52840 Development Kit"

Hello, world!

- Folder: `advanced/firmware`
- File: `src/bin/hello.rs`
- ✅ Click the "Run" button within VS code
  - (if not using VS code, run `cargo run --bin hello`)

API documentation

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

Hello from RTIC

- Folder: `advanced/firmware`
- File: `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

- Folder: `advanced/firmware`
- File: `src/bin/events.rs`
- Peripherals are structs e.g. `POWER`
- Registers are struct fields e.g. `intenset`
- API generated by `svd2rust` from SVD file

Event handling

- Folder: `advanced/firmware`
- File: `src/bin/events.rs`
- Run: with J3 cable disconnected
- Then: connect cable to port J3
- ☑ try adding `loop { }` at the end of `init`

Task state

- Folder: `advanced/firmware`
- File: `src/bin/resource.rs`
- Use an RTIC resource to add state to add task
- Resource in initialized in `#[init]` - POWER is moved into task
- Task can access the resource by reference on each invocation

USB enumeration

- 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

- Folder: `advanced/firmware`
- File: `src/bin/usb-1.rs`
- USBD events: USBRESET, EP0SETUP, EP0DATADONE
- Run: with cable connected to J3
- ✅ Goal: reach EP0SETUP case

USB endpoints

- 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

- 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

- Folder: `advanced/firmware`
- File: `src/bin/usb-2.rs`
- Run: with cable connected to J3
- EP0SETUP 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

- Folder: `common/usb`
- File: `src/lib.rs`
- SETUP data parser
- "Test" button in VS code
- (or run `cargo test` if not using VS code)

Device descriptor

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

Configurations

- 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

- Folder: `advanced/firmware`
- File: `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

Direct Memory Access (DMA)

- Folder: `advanced/firmware`
- File: `src/bin/usb-3.rs`
- Go to definition: `dk::usb::Ep0In::start()`
- Need: move data from RAM to USBD peripheral
- Lifetime requirement: copy to internal buffer
- Compiler fences are required!

Supporting more standard requests

- Folder: `advanced/common/usb`
- File: `src/lib.rs`
- TODO: GET_DESCRIPTOR Configuration
- TODO: SET_CONFIGURATION
- Solutions in `advanced/common/usb/`

Error handling

- Folder: `advanced/firmware`
- File: `src/bin/usb-4.rs`
- `ep0setup` refactored: returns `Result`
- On `Err`: stall the endpoint

Device state

- Folder: `advanced/firmware`
- File: `src/bin/usb-4.rs`
- `State` is now part of the task state
- ✅ update handling of USBRESET event

Stalling the endpoint

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

SET_ADDRESS

- Folder: `advanced/firmware`
- File: `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

- *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

- 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

- 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

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

GET_DESCRIPTOR/CONFIGURATION

- Folder: `advanced/firmware`
- File: `src/bin/usb-4.rs`
- ✅ Check requested index
- ✅ Respond with a single packet that contains
  - Configuration descriptor
  - Interface descriptor

SET_CONFIGURATION (Linux & Mac OS)

- Folder: `advanced/firmware`
- File: `src/bin/usb-4.rs`
- May be sent after SET_ADDRESS
- OK to stall for now

Idle state

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

Inspecting the descriptors

- Folder: `advanced/host/print-descs`
- File: `src/main.rs`
- Run: `cargo run`
- Output: Device, configuration and interface descriptors

Getting it configured (Windows)

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

SET_CONFIGURATION

- Folder: `advanced/firmware`
- File: `src/bin/usb-4.rs`
- Section 9.4.7 of the USB spec explains how to handle this

Final checkpoint

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

Things for you to check out

- 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