TECHNISCHE UNIVERSITÄT DARMSTADT

NES | NETWORKED EMBEDDED SYSTEMS LAB

Prof. Dr. Marco Zimmerling Max Granzow Jonas Kubicki
https://nes-lab.org/

Networked and Low-Power Embedded Systems

WS 2025/26

Lab 1: Embedded Platform Introduction

1 Goals and Overview of the Lab

The goal of the lab is to familiarize students with embedded systems development including programming, analysis, and debugging. We will learn to apply and implement important concepts from the lecture. To do this, we will use an embedded hardware platform development kit and explore its capabilities. In this way, we practically learn about the interaction between the different components in an embedded system and especially its peripherals such as timers, GPIOs, UART, I^2C, and others. In order to understand and correctly configure and use those peripherals, we will also study the related datasheets.

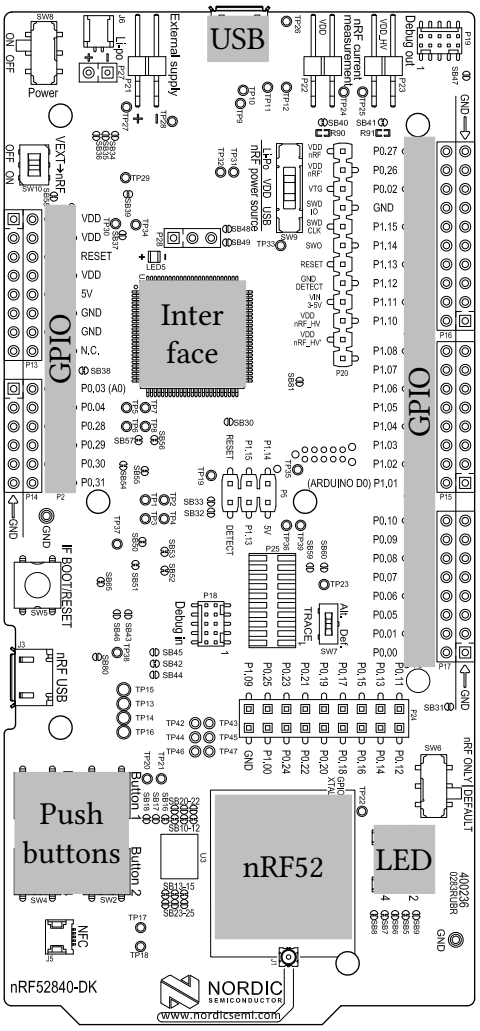
During this lab series, we will realize step by step the whole execution pipeline of a typical wireless sensor network application. We will follow the path from collecting sensor measurements, to processing the data to make useful decisions, and eventually communicating the data wirelessly to another location.

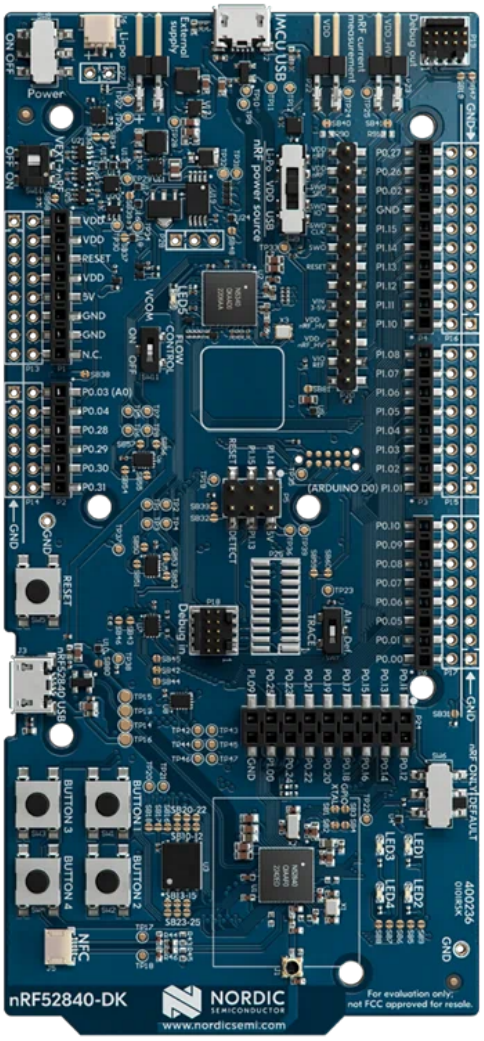
Today's lab lays the foundation for the further labs by introducing the hardware platform and setting up the development environment. For the following labs, we assume some basic C knowledge to be able to program the platform on a very low level, as close as possible to the hardware. If you want to refresh your C knowledge, please have a look at the document "ETH Zurich C companion" in Moodle, which provides an excellent overview of basic C and embedded systems terms and concepts.

2 Embedded Platform – Nordic nRF52840dk

The Nordic nRF52840dk is a development kit (devkit), designed for prototyping of applications based on the nRF52 processor. Besides the processor itself, it contains peripherals used for testing and development, like a debugger or serial interface. The GPIO pins can be easily accessed with the onboard pin headers. The development kit also features four LEDs and four buttons that can be used freely. The back of the development kit describes how they are connected and on which pins they can be accessed.

The hardware in the lab is expensive and fragile, **so please be careful when handling it!** Some wear and tear is normal, but please report any problems to the tutors as soon as possible. In case of damage caused by incorrect or improper use, the person who caused the damage will have to provide a replacement.





The development kit contains the processor and an interface chip for programming and debugging. At the core of the board is the nRF52840 MCU[1], which integrates a powerful 64 MHz ARM Cortex-M4 processor, memory, and a 2.4-GHz and BLE-compatible radio into a tiny module. Additional modules or sensors can be connected via the 48 GPIO pins and accessed via dedicated peripherals for SPI, I^2C, UART, and PWM, that can be flexibly mapped to any GPIO pin. A Micro-USB connector provides an interface to the host computer.

3 Development Setup

In the following we will setup the development toolchain and learn how to program the platform. First, download the `lab1-code.zip` source code archive from Moodle and extract its contents into a directory named `lab1`. It contains the source code required to build the executable that is deployed to the microcontroller. We use a special version of `GCC` to compile the

[1] https://www.nordicsemi.com/Products/nRF52840

program, the result can be found in `build/build.hex`. In a second step, the compiled program is flashed onto the development kit with `nrfjprog`, which stores the code in its memory. After a reset, the development kit begins executing the program's `main` function.

3.1 Compilation Toolchain

Most of the time in the lab we will be working with the Nordic nRF52840 system-on-chip (SOC), which is part of the devlopment kit. This SOC has an ARM Cortex-M4 MCU, so we will use the embedded ARM compiler (`arm-none-eabi-gcc`) to compile our source code.

First install the `nrfjprog` utility.

1. Install Segger J-Link from https://www.segger.com/downloads/jlink/. This will handle the connection to the interface chip on the development kit.
2. Install the nRF Command Line tools from https://www.nordicsemi.com/Products/Development-tools/nRF-Command-Line-Tools and add `nrfjprog` to your host path.

To avoid potential issues with different operating systems or compiler/library versions, we provide a Docker container for this lab that already contains the entire compiler toolchain.

Alternatively you can set up the toolchain yourself, which takes a few extra steps. You essentially need two programs: `make`, and `arm-none-eabi-gcc` (version 15.1.0 is used in the container), including the ARM embedded toolchain. This will allow you to skip the Docker setup below and go directly to compiling the source code in step 8 (don't forget steps 1 and 2).

3. Install Docker, either via https://www.docker.com/get-started/ or via the package manager of your distribution.
4. Start Docker.
5. Pull our Docker container image `$ docker pull neslab/nales-devbox`.

Now we compile an example application to test if the toolchain works correctly.

6. Use the console and change into the `lab1` directory.
7. Start the Docker container.

   ```
   $ docker run -it --rm -v $(pwd):/lab neslab/nales-devbox
   ```

   - `-it` starts an interactive Docker container with a terminal.
   - `--rm` automatically removes the container on exit.
   - `-v` mounts a file or directory on the host machine into the container. The location inside the container is `/lab`. File changes in the mounted directory are synchronized between host and container. On Windows replace `$(pwd)` with the current path.
8. Compile the source code *(inside the container environment)* with `$ make`.
9. If everything is ok, the compilation ends with `"Build successful!"`.

3.2 Programming the Platform

To run our compiled program, we need to copy it to the nRF52840 chip (i.e. flash the firmware). The interaction between your computer and the nRF52840 chip is handled by the interface chip using the `nrfjprog` utility.

The nRF52 chip has an access port protection mechanism that blocks debugger access. To program the development kit, the mechanism must be disabled by running `$ nrfjprog -f NRF52 --recover`, which erases the flash content. This step has to be performed, when flashing fails with a warning about the access port protection.

3.2.1 Flashing the Example Application

1. Use the console on your host *(NOT in the Docker environment)* and change into the `lab1` directory.
2. Connect the development kit to your computer via the USB cable.
3. Flash the program.

```
$ nrfjprog -f NRF52 --program build/build.hex --chiperase --verify
$ nrfjprog -f NRF52 --pinresetenable
$ nrfjprog -f NRF52 --reset
```

4. If everything went well, the LED on the development kit should now “breathe”.

Check the `Makefile` to see what commands are performed during compilation. Additionally, it contains a shorthand script to flash the application. Make sure to run the flashing command *outside the Docker container* because it cannot easily access the USB hardware from inside.

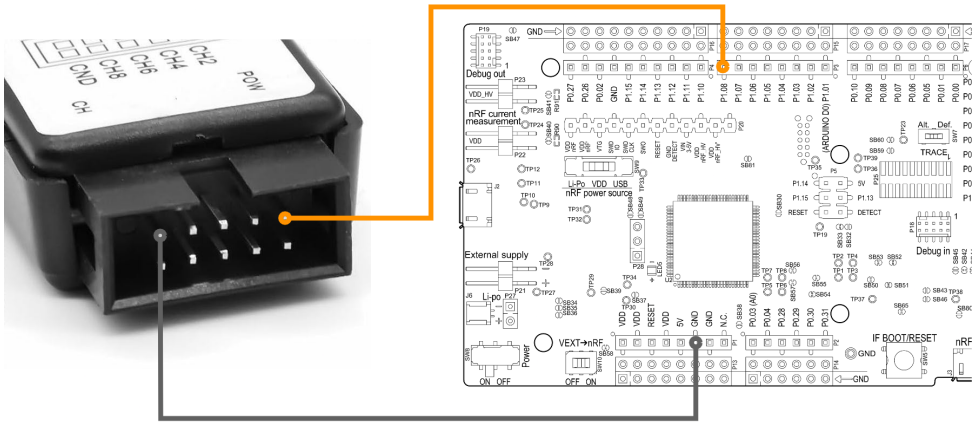
4 Logic Analyzer

In the labs, we will use a logic analyzer to capture, display, and measure electronic signals over time. This is a useful tool in embedded systems development that helps debug and test programs and compare the relation and timing among different signals.

4.1 Connecting the Logic Analyzer

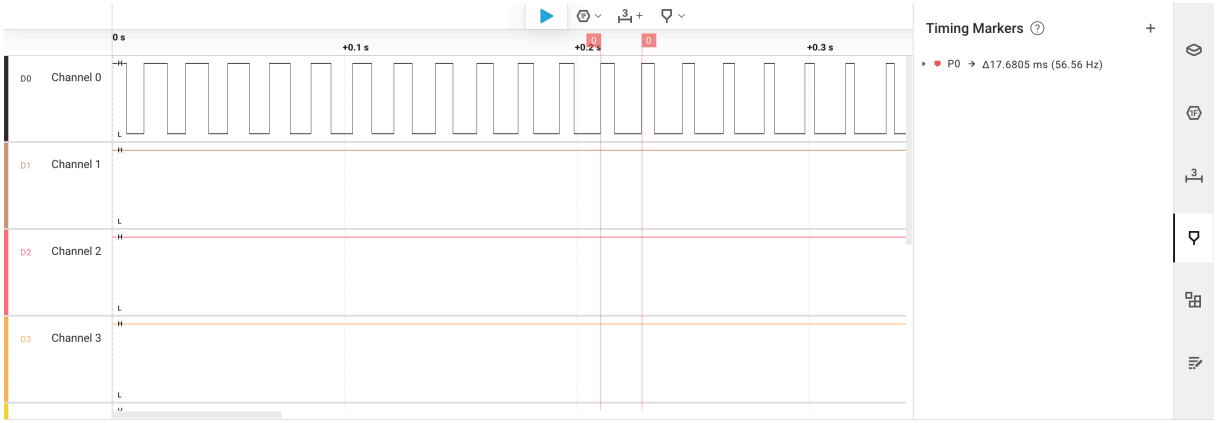
1. Download and start the logic analyzer software `Logic 2` from https://www.saleae.com/de/downloads/.
2. Connect the logic analyzer via USB with your computer and check if it gets detected in `Logic 2`.
3. Connect the logic analyzer with the development kit as depicted above. Use jumper wires to connect the ground (GND) wire on the logic analyzer to the development kit and the wire on channel 1 (see label on the logic analyzer) to pin P1.08.
4. Note that the logic analyzer is labeled starting with channel 1, while the software starts with channel 0. To find channel 1 on the logic analyzer you have to look at channel 0

in the Logic 2 software. If channel 0 is not visible in Logic 2, enable it in the “Device Settings” menu on the right.



4.2 Using the Logic Analyzer

Make sure that the development kit is connected via USB. Start recording the signal in the Logic 2 software by pressing the “play” button in the upper right corner and stop recording after a few seconds. You should now see a similar picture to the one below.



Explore the signal with your cursor, zoom in and out via mouse wheel. Also try out the “Timing Markers and Measurements” category on the right side of the Logic 2 window.

Explain what you see and how this corresponds to the behavior of the development kit!