Prelab: Zephyr and Renode

General Instructions

In this series of labs, we will look at advanced concepts of communication in the context of the Internet of Things. We will focus on short-range wireless (mesh) networks for small, embedded platforms. We will target resource-constrained hardware, in our case featuring a 64 MHz CPU and 256 kB of RAM. All implementations will be done in C.

We divided this course into three main blocks: a mandatory prelab, three labs during which you will implement wireless protocols for IoT devices, and a project during which you will be given more freedom. Please check the lab instructions slides for a complete view on how the labs will be performed. This prelab is **mandatory**, and is a **Pass or Fail** examination. You have one week to submit your written answers and screenshots as a PDF document. Your document **must be submitted on Moodle.** You cannot take part of the final exam if you did not deliver your prelab on time. In the document, all exercises in the instructions must be answered unless stated otherwise. Your answers should be concise; you don't need more than a few lines to answer each question. Questions that need to be answered are within boxes.

You should complete the labs in groups of two persons — use the group you've created in Moodle! If you don't have a lab partner, please use this specific Mattermost channel to find one: `https://wetalk.informatik.uni-kiel.de/internet-of-things-ss24/channels/labs--projects`.

Internet of Things

The Internet of Things is a new paradigm having gained a spectacular momentum since the past decade. The Internet of Things, also shortened as IoT, is an umbrella term covering many applications:

- Smart homes: window shutters, thermostat and heaters, and door locks are connected and controllable from your smartphone, anywhere on the planet;
- Smart factories: industrial robots are equipped with sensors and actuators, allowing the factory to be fully automated and quickly adapt the supply chain to new requirements in a matter of minutes;
- Smart cities: connected meters allow electricity and water providers to adapt energy production to the actual demand of the population, your car can communicate with red lights to determine the most efficient path to your destination;
- Smart textiles: Clothing items incorporate sensors to monitor your physical activity.

When we refer to IoT devices, we often speak of small, resource-constrained devices. These platforms often operate with limited energy, and are usually powered by batteries or small

solar panels. Computing power is limited, with microcontrollers often integrating a single core CPU running at a few MHz. Memory is also scarce, with flash and RAM capped to a few MB.

With such limited hardware, most applications must run bare-metal (without an underlying operating system), or must use specific OSes with limited functionalities. Thus, programmers have to use low-level languages to take full control of the platform.

Part 1 - C Programming

This course assumes that you have an advanced understanding of programming. All implementations will be done in C. We require you to have taken a course featuring C programming in your previous studies, or that you've been using the C language in past personal projects. Additionally, you must have taken a course on operating systems, to fully grasp memory management.

In this first part of the prelab, we test your knowledge of C.

Question 1. Answer the first 10 questions of this online test: `https://www.geeksforgeeks.org/c-language-2-gq/pointers-gq/`
Report your score in your report.

If you scored less than 50%, we highly recommend you to take a C course (or check the videos linked on Mattermost). Programming embedded programs requires a deep understanding of memory management and computation complexity. In the following labs, you will have to constantly deal with pointer operations, C structures, integer overflows, and value padding and packing.

Question 2. What is the stack? The heap? What happens when the stack overflows?
Question 3. What does the **static** keyword do in C?
Question 4. Assume a 16-bit unsigned integer K. Using only binary operators (>>, <<, &, |), how can you write the integer operation $K/16$? How can you write the operation $K\%16$?

Part 2 - Installing Zephyr

During the labs, we will use a special operating system called Zephyr (`https://docs.zephyrproject.org/latest/`). Zephyr is an open-source RTOS specially designed for resource-constrained devices, typically used in the IoT ecosystem.

Since IoT devices and your personal desktop or laptop do not share the same architecture, we will need to install some tools. Compiling code for different hardware (also known as

cross-compiling) requires additional software, commonly referred as toolchain. Zephyr is supported by PlatformIO (`https://platformio.org/`) containing the entire toolchain and the Zephyr Operating System.

Note: While there are other ways to use Zephyr (e.g., the Python tool `west`), we recommend using PlatformIO for its stability and simplicity to use.

Instructions. Install VSCode and follow this page (`https://platformio.org/platformio-ide`) and the lab introduction slides to set up PlatformIO.
If you nonetheless want to use Zephyr with `west`, follow this getting started guide: `https://docs.zephyrproject.org/latest/develop/getting_started/index.html`

Part 3 - Installing Renode and Wireshark

For the labs, we will use a simulator called Renode (`https://renode.io`), that emulates full IoT devices and simulates a wireless medium for testing and evaluating protocols.
Moreover, we will use Wireshark (`https://www.wireshark.org`) to inspect the communication between devices in Renode.

Instructions. Install Renode following these instructions (`https://github.com/renode/renode/blob/master/README.rst#installation`). Install Wireshark if you do not already have it installed.
Note: Check that you installed all dependencies (`https://github.com/renode/renode/blob/master/README.rst#installing-dependencies`) for Renode to work, especially, Mono/.NET, depending on your local Operating System.

Part 4 - Running BLE Example in Renode

At this point, you should have Zephyr/PlatformIO and Renode installed on your machine. You should be able to start Renode and see the *monitor window*. You will start discovering the functionalities of Renode by running a predefined BLE example included in Renode.

Instructions. Run a demo application in Renode: `https://renode.readthedocs.io/en/latest/tutorials/ble-simulation.html`

At this point, you should see several windows in the simulator, running the application and generating output.

Question 5. Did you install everything correctly? Take a screenshot showing that Renode works.

Part 5 - Basic Networking

As you might recall from the lectures, an IoT device is defined as a resource-constrained device, augmented with computing and communication capabilities, sometimes able to sense or interact with its environment. During these labs, we will use the Bluetooth Low Energy (BLE) standard for short-range communication. As example, every day devices around you, like your phone, wireless headphones, keyboards, smart speakers, and many others use BLE.

In these labs, we build our communication protocols on the application layer and use the Zephyr BLE stack as is.

To get you started with Zephyr, you will run an example with a BLE beacon and a BLE advertisement scanner.

Instructions. In the enclosed material, you find two PlatformIO projects. Study these examples, compile them, and modify the enclosed Renode script to run them in Renode.
Note: You can find the compiled firmware in `.pio/build/nrf52840_dk/firmware.elf`.
Question 6. How is a BLE simulation configured in Renode? Be brief.
Question 7. What does the application do? Take a look at the linked code and briefly explain what it does.
Question 8. Run both the advertiser and the scanner in Renode and study the output in Renode and in Wireshark. Especially, take a look at the "Bluetooth Low Energy Link Layer" information in Wireshark. Can you identify the advertising data we send? Please document it.
Question 9 ***(optional).*** Upload the BLE beacon firmware (here: the `.hex`-file) to a physical device, e.g., the BBC micro:bit v2. Use an app on your phone to scan for BLE beacons, for example the nRF Connect App (`https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-mobile`), and document what you see, including a screenshot. Were you able to find your beacon?

Optional: To Go Further

Important: This part is not mandatory to pass the prelab. This is the kind of question we will ask you in the following labs. You can take a look and see if you believe this course is for you.

It is now time to design your first, simple wireless communication between multiple devices.

We will start with a simple network composed of **4 (four) BLE nodes**, placed in a line. A company wants you to design a decentralized, global counter, to count the number of people entering a shopping mall through a set of four doors next to each other. Each node must have a 16-bit unsigned integer counter (tip: set the counter variable as static). Every **10 seconds**, a node increments its counter by one and broadcasts its value. When a node receives a counter value, it selects the **maximum** between the received and local counter value, and sets its local value accordingly.
Optional question. Starting from the code, you have seen so far, imagine how you could implement the simple protocol presented above. **You do not need to implement and run the protocol!** Simply, explain in your report how this protocol could be implemented (list the functions you would need to use, and the important parts your code would require). What problems could arise? Will your counter be consistent everywhere?

During the following labs, you will need to design, justify, implement and evaluate such protocols in Renode.