ET014A/ET016A Laboratory Exercise 1: Getting Started with Contiki-NG and the Firefly node

Sebastian Bader

1 Background and Goals

This lab acts as an introduction to the programming of basic Contiki-NG[1] applications and the usage of the Firefly nodes[2]. The focus of the exercise lies on getting to know the hardware and software platform and to learn how to use its basic features.

After this exercise you should be able to:

- Create, edit, compile and upload simple Contiki-NG applications.
- Handle simple user interfaces of the Firefly node (i.e., LEDs and pushbuttons).
- Implement time-triggered software actions.

1.1 Preparation

Read the following documentation pages. You can use the quiz on Moodle for some guidance what to focus on.

- Contiki-NG build system
- Repository structure
- Processes and events
- Timers

1.2 Required Equipment

- a Zolertia Firefly node
- a computer with the Contiki-NG development environment[3]

1.3 Evaluation

This lab is evaluated through the demonstration of functional implementations.

[1]http://contiki-ng.org
[2]http://www.zolertia.io
[3]a link to a virtual machine is provided on the moodle page

Task 1: Getting Started

This task will introduce you to the build system of the Contiki operating system. You will compile, upload and run an existing example application.

For this task, you can follow the Hello World Tutorial, which you can find here. In order to run the program on the Firefly node, you have to use the `TARGET=zoul` and `BOARD=firefly-reva` parameters.

You can ignore the section on Cooja for this task. It is also not necessary to fully understand the code yet.

Task 2: LEDs

2.1: Basic LEDs

Now that you know the basics, it is time to have a look at some features of Contiki-NG and the Firefly. We will start with LED handling, providing simple output interfaces. You can find an example on how to handle basic LEDs in `examples/dev/leds/leds-example.c`. For more information you can even have a look at the LED library, which you can find at `os/dev/led.{c/h}`.

Create a new application (e.g., by copying the hello-world application folder) and empty the PROCESS_THREAD so that just the PROCESS_BEGIN and PROCESS_END structure is left.

Now adjust the process thread implementation so that the green LED turns on after startup and remains on. Compile and upload your new program. Note that you might have to adjust your `Makefile` if you have changed the name of the program (recommended) or the relative path distance to the Contiki base folder.

2.2: RGB LEDs

The Firefly actually does not contain individual LEDs, but a single RGB LED. Contiki-NG provides a driver for RGB LEDs, which is very similar to the handling of basic LEDs, but can utilize color combinations. You can find an example in `examples/dev/rgb-led/rgb-led-example.c` and the driver is located at `arch/dev/rgb-led/`.

Utilizing this information, create a new program that blinks a white light. To accomplish this in the simplest manner, you can make use of the `clock_wait` function. This function is provided by the Clock library and you can find its documentation here.

Task 3: Pushbuttons

LEDs being a simple user output, pushbuttons are equivalently simple user inputs. In Contiki-NG, buttons are handled via the button-hal (i.e., a hardware abstraction layer). You can find an example for handling buttons in `examples/dev/button-hal/button-hal-example.c` and the driver in `os/dev/button-hal.{c/h}`.

Based on this information, create a new program that turns an LED on when the button is pressed, and turns it off when the button is released. You may end up getting a compiler error complaining that a variable is set but not used. Contiki is configured to treat warnings as errors by default, which you can override by adding `CFLAGS+=-Wno-error` at the end of your Makefile.

Task 4: Timers

In Task 2, we used a wait function to create a periodic event. Wait functions, however, block program execution and are thus not very efficient. Instead, the usage of timers and time events is much more common.

Contiki-NG implements a number of timers, namely `timer`, `stimer`, `etimer`, `ctimer` and `rtimer`. Each of these timers are suitable for specific tasks, but the etimer is probably the one used the most. You have already seen an example of its usage in the Hello World example.

Create a new application that toggles an LED with a 2-second interval using the etimer.

Task 5: Processes (Optional)

Contiki allows programs to consist of multiple processes. In the simplest case, these processes will just run in parallel, but they can also interact and communicate with each other.

Try running both Task 3 and 4 in the same program as separate processes. Utilizing different LED colors it should be easy to see that the two processes seemingly run in parallel. You may even try to start and stop the blinking process by pressing the pushbutton in a separate process.