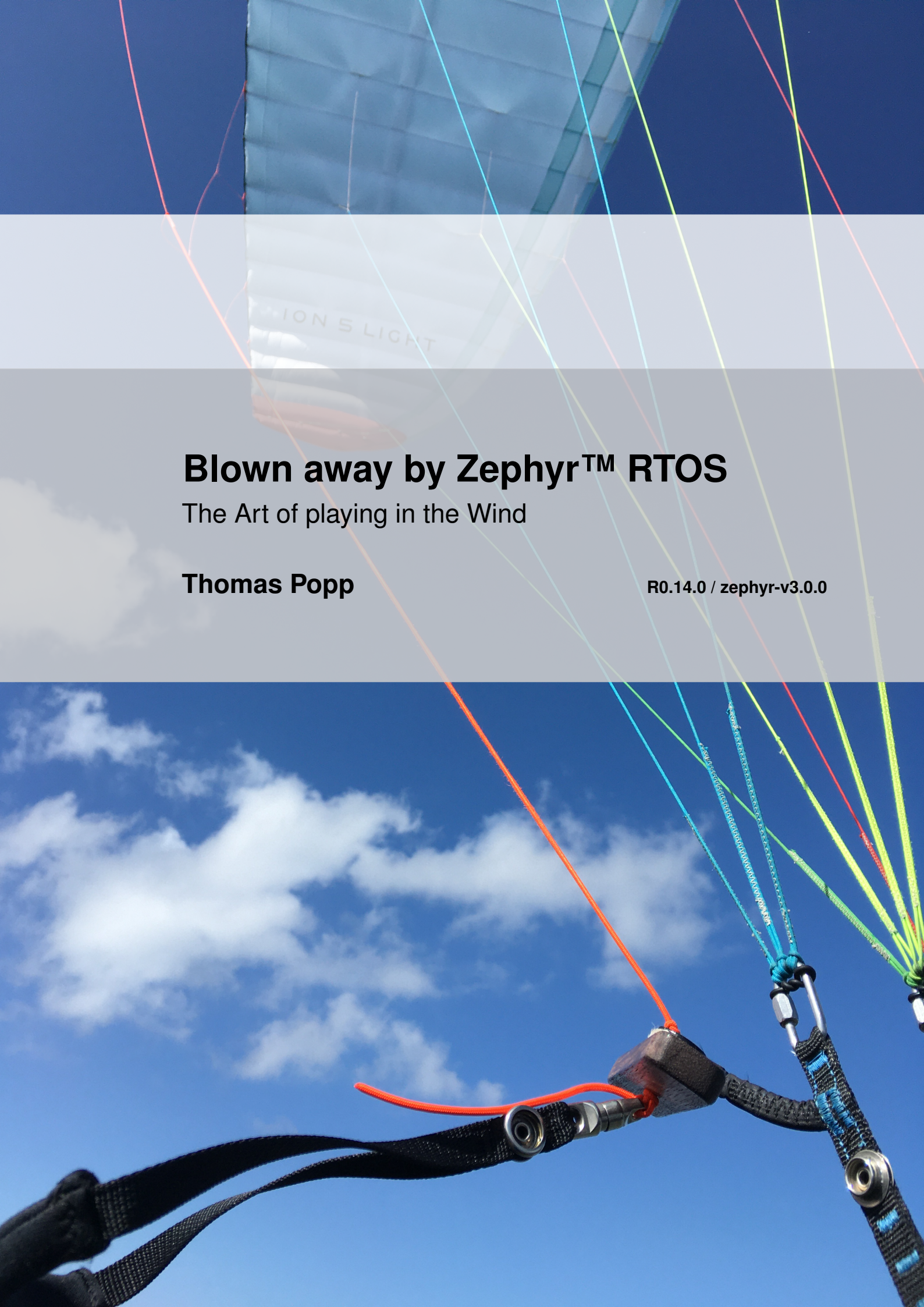


Blown away by Zephyr™ RTOS

The Art of playing in the Wind

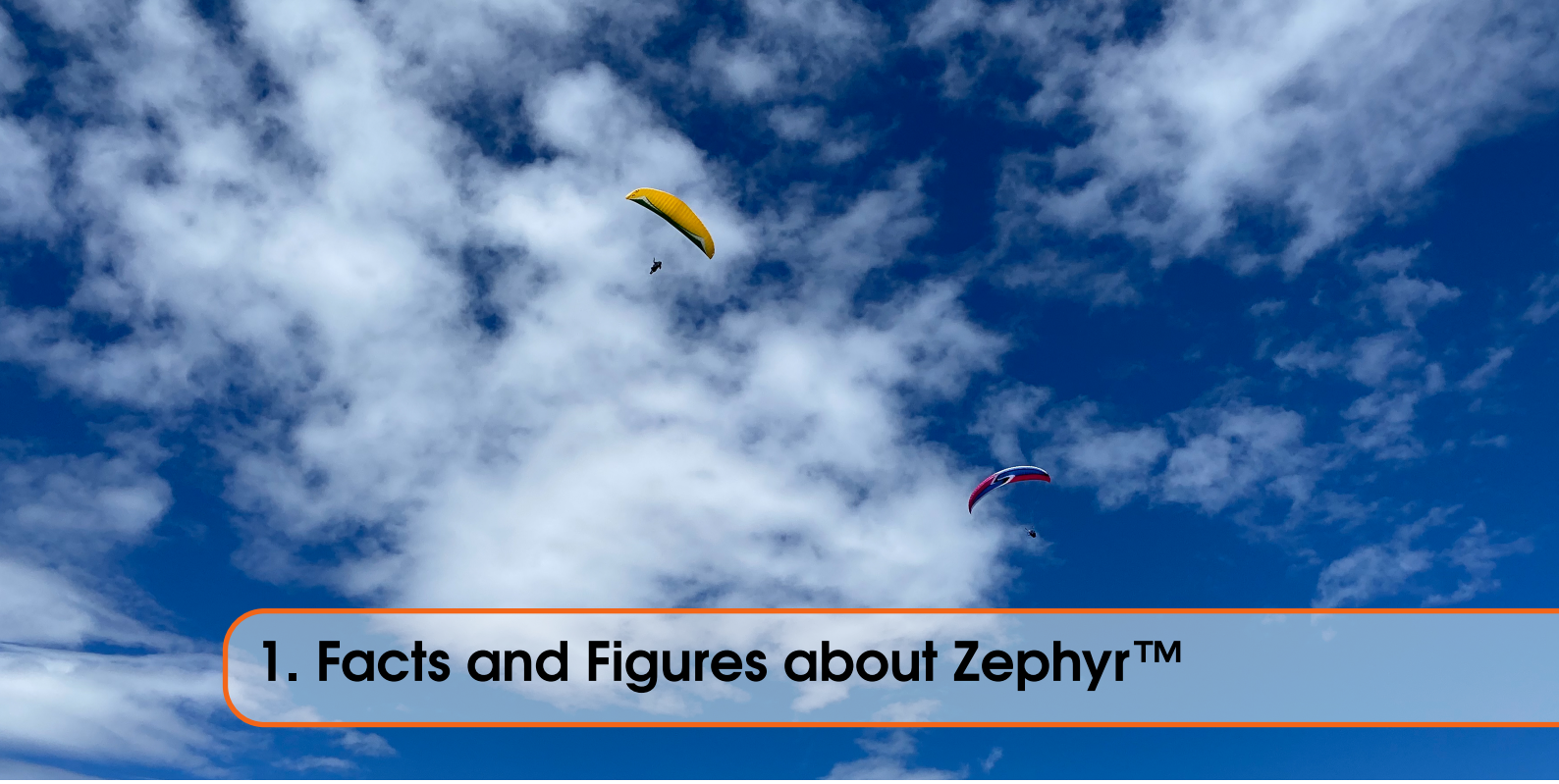
Thomas Popp

R0.14.0 / zephyr-v3.0.0

Contents

III Appendix

I Bootstrapping



1. Facts and Figures about Zephyr™

1.1 Zephyr™ in a Glimpse

Zephyr™ is an open source real-time operating system for the small, cute and resource-limited microcontrollers like derivatives from the ARM Cortex M-series. It is a project developed by de Linux Foundation for the Internet of Things. Version 1.0.0 was launched in February 2016. Currently (April 2022) version 3.0.0 is available. The project web site can be found here: https://www.zephyrproject.org/.

1.2 Supported devices

Zephyr™ RTOS supports a wide range of CPU Architectures, System on Chip's (SoC) aka Microcontrollers and HW-Boards. Almost every newer board is supported by Zephyr™ and can be used for the first steps.

1.2.1 CPU Architectures

Currently follwing CPU architectures are supported by Zephyr™ RTOS.

- arc
- arm
- nios2
- posix
- riscv
- x86
- xtensa

1.2.2 System on Chip

As of writing the following ARM SoC Families are supported by Zephyr™ RTOS.

- arm
- atmel sam
- atmel sam0
- bcm vk
- cypress
- infineon xmc
- microchip mec
- nordic nrf
- nuvoton
- nuvoton npcx
- nxp imx
- nxp kinetis
- nxp lpc
- qemu cortex a53
- silabs exx32
- st stm32
- ti lm3s6965
- ti simplelink
- xilinx zynqmp

Looking into the stm32 and atmel sam0 families folders reveals a huge amount of supported derivatives to the viewer:

- samd20
- samd21
- samd51
- same51
- same53
- same54
- samr21
- stm32f0
- stm32f1
- stm32f2
- stm32f3
- stm32f4
- stm32f7
- stm32g0
- stm32g4
- stm32h7
- stm32l0
- stm32l1
- stm32l4
- stm32l5
- stm32mp1
- stm32wb

A full list of all supported SoC's can be found in the repo: https://github.com/zephyrproject-rtos/zephyr/tree/master/soc. A list of supported boards can be in the Supported Boards section of the Zephyr™ documentation. At the end, I'm quite sure, you will find an appropriate microcontroller for your project! If not, try harder ;-).

1.3 A word about the Open Source License of Zephyr™

The Zephyr Project is licensed under Apache 2.0 and can therefore be used for your commercial projects as well to write your own open source application. For details see: https://www.zephyrproject.org/faqs/.

2. Getting Ready

This chapter shows you the first steps with Zephyr™ RTOS. At the end you will have a working installation of the development environment. You will have compiled a sample application and flashed into the target system. The final target is a flashing LED on your board of choice.

2.1 Getting Started with Zephyr™

To install the Zephyr™ environment, follow the Getting Started Guide. This will copy and install quite a bunch of files, code and tools onto your workstation. The installation should work on the following operating systems (for details have a look into the hints corner: A.0.1):

- Ubuntu 20.04 LTS (the recommended one;-)
- Ubuntu 18.04 LTS
- Max OS/X
- Windows 10

For the first installation, it is recommended to use the proposed folders and tool versions. Once the installation is up and running, you can tweak it to your needs.

If you are bound to a (windows) company network, see the hints in the appendix.

A few coffees later the environment is installed and ready for a little exercise and a basic check.

2.2 Exercises

2.2.1 Hello World

Exercise 2.1 Compile and build the HelloWorld sample for your board/machine and flash it to your hardware. ■

How-To get the hello world sample running on a STM32-board

To build the sample application use the following commands:

```
cd ~/zephyrproject/zephyr
west build -p auto -b <your-board-name> samples/basic/hello_world
```

Change `<your-board-name>` appropriately for your board. For a STM32L476 Nucleo-64 Board use `nucleo_l476rg`.

```
west build -p auto -b adafruit_itsybitsy_m4_express samples/basic/hello_world
```

```
west build -p auto -b adafruit_itsybitsy_m4_express samples/basic/hello_world
```

Some seconds later and if everything goes well, the following output will show up in your shell console:

```
...
[128/133] Linking C executable zephyr/zephyr_prebuilt.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:       14232 B         1 MB      1.36%
            SRAM:        4272 B        96 KB      4.35%
        IDT_LIST:         120 B         2 KB      5.86%
[133/133] Linking C executable zephyr/zephyr.elf
```

Connect your target with the workstation and bring it into bootloader mode. Then flash the sample application to the microcontroller by using the `west flash` command:

```
west flash
```

Some seconds later and if everything goes well, the following output will show up in your shell console:

```
Open On-Chip Debugger 0.10.0+dev-01341-g580d06d9d-dirty (2020-05-16-15:41)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results
 ↪  might differ compared to plain JTAG/SWD
Info : clock speed 500 kHz
Info : STLINK V2J28M17 (API v2) VID:PID 0483:374B
Info : Target voltage: 3.253997
Info : stm32l4x.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : Listening on port 3333 for gdb connections
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* stm32l4x.cpu       hla_target little stm32l4x.cpu       running

Info : Unable to match requested speed 500 kHz, using 480 kHz
Info : Unable to match requested speed 500 kHz, using 480 kHz
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08000f3c msp: 0x20000770
Info : device idcode = 0x10076415 (STM32L47/L48xx - Rev: 4)
Info : flash size = 1024kbytes
Info : flash mode : dual-bank
Warn : Adding extra erase range, 0x080039f8 .. 0x08003fff
auto erase enabled
wrote 14840 bytes from file
 ↪  /home/poppth/projects/zephyrproject/zephyr/build/zephyr/zephyr.hex in
 ↪  0.922547s (15.709 KiB/s)

Info : Unable to match requested speed 500 kHz, using 480 kHz
Info : Unable to match requested speed 500 kHz, using 480 kHz
target halted due to debug-request, current mode: Thread
```

```
xPSR: 0x01000000 pc: 0x08000f3c msp: 0x20000770
verified 14840 bytes in 0.508028s (28.526 KiB/s)

Info : Unable to match requested speed 500 kHz, using 480 kHz
Info : Unable to match requested speed 500 kHz, using 480 kHz
shutdown command invoked
```

Now connect the serial line with your workstation e.g. by using an USB to serial converter and open the line with your favorite terminal program (serial line parameters: 115200 8N1). My favorite terminal program is `minicom`. On a windows box `putty` will perfectly do the job. If really everything went well, the following Hello World message will apear in your terminal:

```
minicom -D /dev/ttyACM0
```

```
*** Booting Zephyr OS build zephyr-v2.3.0-2329-g9ac8dcf5a785  ***
Hello World! nucleo_l476rg
```

How-To get the hello world sample running on a SAMD21-board

To build the sample application use the following commands:

```
cd ~/zephyrproject/zephyr
west build -p auto -b <your-board-name> samples/basic/hello_world
```

Change `<your-board-name>` appropriately for your board. For an Adafruit ItsyBitsy M4 Express Board use `adafruit_itsybitsy_m4_express`.

```
west build -p auto -b adafruit_itsybitsy_m4_express samples/basic/hello_world
```

```
west build -p auto -b adafruit_itsybitsy_m4_express samples/basic/hello_world
```

Some seconds later and if everything goes well, the following output will show up in your shell console:

```
[114/119] Linking C executable zephyr/zephyr_prebuilt.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:       10404 B       232 KB      4.38%
            SRAM:        3952 B        32 KB     12.06%
        IDT_LIST:          72 B         2 KB      3.52%
[119/119] Linking C executable zephyr/zephyr.elf
```

Connect your target with the workstation and bring it into bootloader mode. Then flash the sample application to the microcontroller by using the `west flash` command:

```
west flash
```

Some seconds later and if everything goes well, the following output will show up in your shell console:

```
-- west flash: using runner bossac
Atmel SMART device 0x1001000a found
Erase flash
done in 0.825 seconds

Write 10404 bytes to flash (163 pages)
[==============================] 100% (163/163 pages)
done in 0.057 seconds

Verify 10404 bytes of flash with checksum.
Verify successful
done in 0.017 seconds
Set boot flash true
Ignoring set boot from flash flag.
CPU reset.
```

Now connect the serial line with your workstation e.g. by using an USB to serial converter and open the line with your favorite terminal program (serial line parameters: 115200 8N1). My favorite terminal program is `minicom`. On a windows box `putty` will perfectly do the job. If really everything went well, the following Hello World message will apear in your terminal:

```
minicom -D /dev/ttyUSB0
```

```
*** Booting Zephyr OS build zephyr-v2.3.0-2329-g9ac8dcf5a785  ***
Hello World! adafruit_itsybitsy_m4_express
```

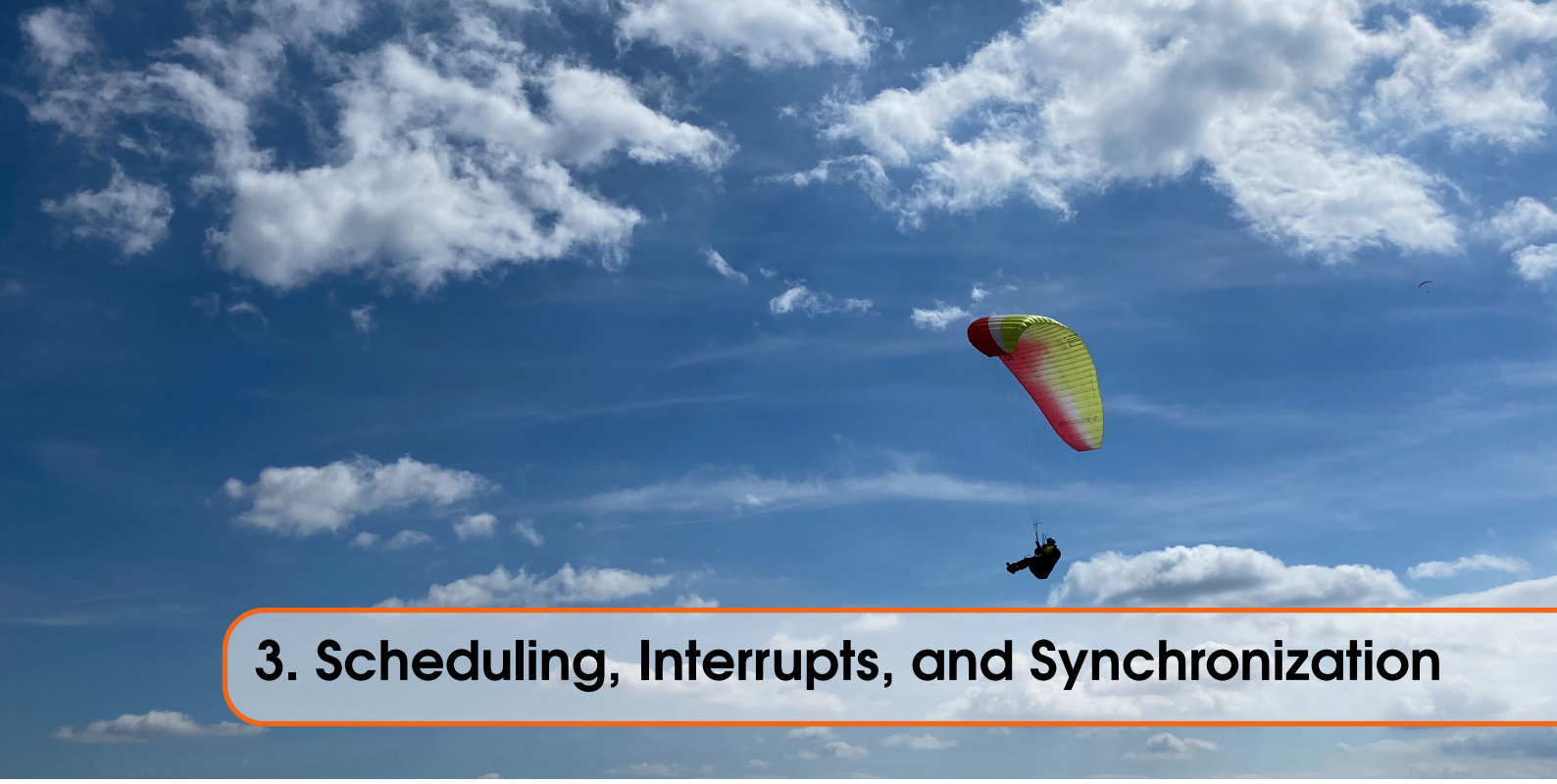
2.2.2 Hello World LED

Exercise 2.2 Compile and build the Hello World LED (Blinky) sample for your board/machine and flash it to your hardware. And yes, Lorenz, it is really amazing when a LED flashes, even if it does so on an eval board when you power it for the first time...

You will find all samples in the folder `~/zephyrproject/zephyr/samples/` of your installation. You can try them out as you like and play around until your ears are wiggling (nice german 1:1 translation;-). ■

Exercise 2.3 Analyze the sources of the examples. In particular the configuration files. ■

II The RTOS Services



3. Scheduling, Interrupts, and Synchronization

In this chapter we take a closer look at the sheduling services, interrupts and its synchronization. In Zephyr ththreads are used to execute independent tasks of an application. Plase have a look at the related Section of the Zephyr™ Docu.

3.1 Threads

Let's take a closer look at threads to start into the sheduling topic. In Zephyr threads are used to execute independent tasks of an application which should not be performed by an ISR.

Each Thread has a dedicated stack area. This must be initialized in advance. There a basicly 2 ways to declare and spawning a thread. You can define it and run it out of a funciton or alternatively, it can be declared at compile time. Plase have a look at the Threads Section of the Zephyr™ Docu. Enough theory, we learn the rest from the exercise: Learning by doing.

3.1.1 Hello World Threading

Exercise 3.1 Write an Application with a thread which makes a LED blink every second. Compile it and flash it to your microprocessor board. ■

Solution

The thread configuration needs a stack size, priority and some variables. This can be done as shown in the following code fragment.

```
30 // ---- defines, configs and static vars for the LED0 thread -------------------
31 #define LED0_THREAD_STACK_SIZE 100
32 #define LED0_THREAD_PRIORITY 5
33
34 K_THREAD_STACK_DEFINE(led0_thread_stack_area, LED0_THREAD_STACK_SIZE);
35 static struct k_thread led0_thread_data;
36 static k_tid_t led0_thread_tid;
```

The Thread can do some init work at first and then do its thask work in a endless loop. The next code snippet shows a typical thread.

```
39 // ---- the thread for the LED0 task -----------------------------------------
40 void led0_thread (void *unused0, void *unused1, void *unused2) {
41     const struct device *dev;
42     int32_t rc;
43 
44     // init the LED
45     dev = device_get_binding(LED0);
46     if (dev == NULL) {
47         return;
48     }
49 
50     rc = gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
51     if (rc < 0) {
52         return;
53     }
54 
55     // run the thread for ever
56     while (true) {
57         gpio_pin_set(dev, PIN, (int)true);
58         k_msleep(100);
59         gpio_pin_set(dev, PIN, (int)false);
60         k_msleep(900);
61     }
62 }
```

Last but not least the task is initialized and launched. The complete example is stored in `./samples/helloWorldThreadingStep0`. Compile it and have fun!.

```
65 // ---------------------------------------------------------------------------
66 void main (void) {
67     // init the thread
68     led0_thread_tid = k_thread_create(&led0_thread_data, led0_thread_stack_area,
69             K_THREAD_STACK_SIZEOF(led0_thread_stack_area),
70             led0_thread,
71             NULL, NULL, NULL,
72             LED0_THREAD_PRIORITY, 0, K_NO_WAIT);
73 
74     // that's it! Let's have a Café!
75 }
```

.

III Appendix

A. Installation Hints

A.0.1 Build Host Hint Hint #0

I highly recommend to install the zephyr development environment on a linux host. Anything else might imply some trouble. Sorry for that.

```
from sys import platform
import sys
if sys.platform == "linux" or platform == "linux2":
    print('you are fine!')
    sys.exit(0)
elif platform == "darwin":
    print('this might imply some trouble')
    sys.exit(-1000)
elif platform == "win32":
    print('sorry mate you are in trouble...')
    sys.exit(-1000000000000000000000000)
```

A.0.2 Company Network Hint #1

In a company network of my first choice I discovered by chance that the installation on a windows 10 box only works with the following tricks and this and this might not be conclusive:

Step 3 Get Zephyr and install Python dependencies

- Follow Step 1..2 of the Getting Started Guide
- Step 3.1: `pip3 install --user -U west` must be executed as administrator
- Step 3.2ff: Open a new `cmd.exe` as a regular user
- Step 3.2: create a new and empty folder `\projects\` on drive c:. Change into this folder by typing `cd c:\projects` instead of `cd %HOMEPATH%`. Then follow the remaining instructions of step 3.2.
- Step 3.4: run this step as administrator and exchange `%HOMEPATH%` with `c:\projects`.

Step 4 Install a Toolchain

At this point it gets a bit tricky...

- Download and install `gcc-arm-none-eabi-9-2019-q4-major` from GNU ARM Embedded. Choose the following installation path: `C:\gnu_arm_embedded`
- Set the following environment variables:

```
set ZEPHYR_TOOLCHAIN_VARIANT=gnuarmemb
set GNUARMEMB_TOOLCHAIN_PATH=C:\gnu_arm_embedded
```

- Install a flash and debug tool for your architecutre. For stm32 boards its openocd. Open a new `cmd.exe` as an administrator and rund the following cmd:

```
choco install openocd
```

A.0.3 Company Network Hint #2

In some company networks access to git repositories is blocked for unexplainable reasons. In these cases, a nice talk with the IT department will help. There are of course other solutions which can't be published here. In exchange for a café I could disclose them.