Getting started

If not done already, please ensure you followed the **prerequisites**

Otherwise move to next slide

Structure of a RIOT application

A minimal RIOT application consists in:

- A `Makefile`

```
APPLICATION = example

BOARD ?= native

RIOTBASE ?= $(CURDIR)/../../../RIOT

DEVELHELP ?= 1

include $(RIOTBASE)/Makefile.include
```

- A C-file containing the main function

```c
#include <stdio.h>

int main(void)
{
    puts("My first RIOT application");
    return 0;
}
```

Build a RIOT application

- The build system of RIOT is based on **make** build tool

Build a RIOT application

- The build system of RIOT is based on **make** build tool
- To build an application, **make** can be called in 2 ways:
  - From the application directory:

```
$ cd <application_directory>
$ make
```

  - From anywhere, by using the `-C` to specify the application directory:

```
$ make -C <application_directory>
```

Build a RIOT application

- The build system of RIOT is based on **make** build tool
- To build an application, **make** can be called in 2 ways:
  - From the application directory:

```
$ cd <application_directory>
$ make
```

  - From anywhere, by using the `-C` to specify the application directory:

```
$ make -C <application_directory>
```

- Use the **BOARD** variable to specify the target at build time

```
$ make BOARD=<target> -C <application_directory>
```

`BOARD` can be any board supported by RIOT
⇒ see the **RIOT/boards** directory for the complete list

Build a RIOT application

- The build system of RIOT is based on **make** build tool
- To build an application, **make** can be called in 2 ways:
  - From the application directory:

```
$ cd <application_directory>
$ make
```

  - From anywhere, by using the `-C` to specify the application directory:

```
$ make -C <application_directory>
```

- Use the **BOARD** variable to specify the target at build time

```
$ make BOARD=<target> -C <application_directory>
```

  `BOARD` can be any board supported by RIOT
  ⇒ see the **RIOT/boards** directory for the complete list

- Use the **RIOTBASE** variable to specify the RIOT source base directory

Run a RIOT application

This depends on the target board:

- Running on **native**: the RIOT application executed is a simple Linux process

```
$ make BOARD=native -C <application_dir>
$ <application_dir>/bin/native/application.elf
```

- Running on **hardware**: the RIOT application must be *flashed* first on the board

Run a RIOT application

This depends on the target board:

- Running on **native**: the RIOT application executed is a simple Linux process

```
$ make BOARD=native -C <application_dir>
$ <application_dir>/bin/native/application.elf
```

- Running on **hardware**: the RIOT application must be *flashed* first on the board

⇒ use the **flash** and **term** targets with make

- **flash**: build and write the firmware on the MCU flash memory
- **term**: opens a terminal client connected to the serial port of the target

All this can be done in one command:

```
$ make BOARD=<target> -C <application_dir> flash term
```

Note: the last command can also be used with **native** target

Exercise: your first RIOT application

Let's build and run our first RIOT application !

You just need to follow the instructions in this exercise README

```
$ cd ~/riot-course/exercises/getting-started/first-app
$ make
Building application "example" for "native" with MCU "native".

"make" -C /home/user/RIOT/boards/native
"make" -C /home/user/RIOT/boards/native/drivers
"make" -C /home/user/RIOT/core
"make" -C /home/user/RIOT/cpu/native
"make" -C /home/user/RIOT/cpu/native/periph
"make" -C /home/user/RIOT/cpu/native/vfs
"make" -C /home/user/RIOT/drivers
"make" -C /home/user/RIOT/drivers/periph_common
"make" -C /home/user/RIOT/sys
"make" -C /home/user/RIOT/sys/auto_init
 text    data  bss    dec    hex   filename
 20206   568   47652  68426  10b4a .../getting-started/first-app/bin/native/example
```

How to extend the application

⇒ by adding modules in the application `Makefile` or from the command line:

- Add extra modules with **USEMODULE**
  ⇒ `xtimer`, `fmt`, `shell`, `ps`, etc
- Include external packages with **USEPKG**
  ⇒ `lwip`, `semtech-loramac`, etc
- Use MCU peripherals drivers with **FEATURES_REQUIRED**:
  ⇒ `periph_gpio`, `periph_uart`, `periph_spi`, `periph_i2c`

How to extend the application

⇒ by adding modules in the application `Makefile` or from the command line:

- Add extra modules with **USEMODULE**
  ⇒ `xtimer`, `fmt`, `shell`, `ps`, etc
- Include external packages with **USEPKG**
  ⇒ `lwip`, `semtech-loramac`, etc
- Use MCU peripherals drivers with **FEATURES_REQUIRED**:
  ⇒ `periph_gpio`, `periph_uart`, `periph_spi`, `periph_i2c`

Example in a `Makefile`:

```
USEMODULE += xtimer shell

USEPKG += semtech-loramac

FEATURES_REQUIRED += periph_gpio
```

Example from the command line:

```
$ USEMODULE=xtimer make BOARD=b-l072z-lrwan1
```

Exercise: write an application with a shell

Follow the instructions in the exercise README

Interaction with the hardware

Interaction with the hardware can be performed at 3 levels:

- At board level by using predefined macros for controlling LEDs and buttons

  ⇒ just include `board.h` to use them

Interaction with the hardware

Interaction with the hardware can be performed at 3 levels:

- At board level by using predefined macros for controlling LEDs and buttons

  ⇒ just include `board.h` to use them

- At cpu level by using the MCU peripheral drivers APIs

  ⇒ periph_gpio periph_i2c periph_uart, etc

  This level is considered as the **HAL** of RIOT since it provides a common API for all types/architectures of CPU

Interaction with the hardware

Interaction with the hardware can be performed at 3 levels:

- At board level by using predefined macros for controlling LEDs and buttons

  ⇒ just include `board.h` to use them

- At cpu level by using the MCU peripheral drivers APIs

  ⇒ periph_gpio periph_i2c periph_uart, etc

  This level is considered as the **HAL** of RIOT since it provides a common API for all types/architectures of CPU

- At driver level by using high level driver APIs for specific external sensors/actuators/radios

  ⇒ bmp180, hts221, sx1276, etc

Exercise: interaction with the hardware

- Follow the instructions of the led exercise README to toggle LEDs from shell commands
- Follow the instructions of the sensor exercise README to read values from a sensor with shell commands

Going further: read existing applications source code

The RIOT source directory contains applications that can be used as examples for almost all features provided by RIOT.

- See applications in the `examples` directory
- Test applications in `tests` directory also provides good examples to start the RIOT

Summary

- Build & run your first RIOT application, native and on hardware
- How to extend an application, the `shell`
- Basic interaction with the hardware
- Read sensor values

Back to the course