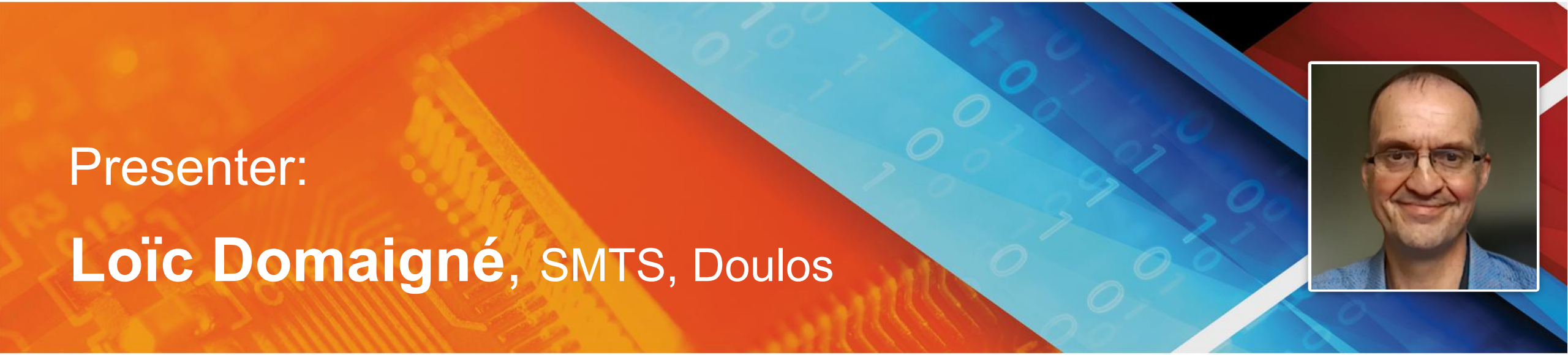
Zephyr Optimisations

Through the looking ~~glass~~ code

Zephyr®

Delivering KnowHow™ www.doulos.com



Presenter:

Loïc Domaigné, SMTS, Doulos

DOULOS

Zephyr Project Meetup: Winterthur, Switzerland

NOSER ENGINEERING

zhaw School of Engineering
Institute of Embedded Systems

loïc@doulos

30 years software / IT experience

» HPC → Telecom → Air Traffic Control → Airline IT

» Medical → Automotive → Technical Training

Delivering Know-How™

DOULOS

www.doulos.com

» SoC Design & Verification

» FPGA & Hardware Design

» Embedded software

» Python, Edge AI and Deep Learning

Zephyr®

» TSC member

» Release WG

» Contributor

Agenda

1. Motivation
2. Kconfig, autoconf.h and conditional compilation
3. Size optimisations using SysV voodoo
4. DT macro, the good part
5. Driver API: abstraction and optimisations

DOULOS

Zephyr®

Blinky: from Bare Metal to Zephyr

- Classic Embedded Programming:
  - Use header file and #define macro
  - Configure the peripheral
  - Access the registers directly

Could be encapsulated
in a file or static library

→ **"driver"**

- Zephyr:
  - Configure project to enable GPIO
  - Query the devicetree
  - Use GPIO driver to blink LED

Zephyr:

- common APIs for GPIO
- API realized by SoC-specific drivers

- Zephyr feels very high level…
- Impact on code speed or size?

How does Zephyr handle this ?

Agenda

1. Motivation
2. Kconfig, autoconf.h and conditional compilation
3. Size optimisations using SysV voodoo
4. DT macro, the good part
5. Driver API: abstraction and optimisations

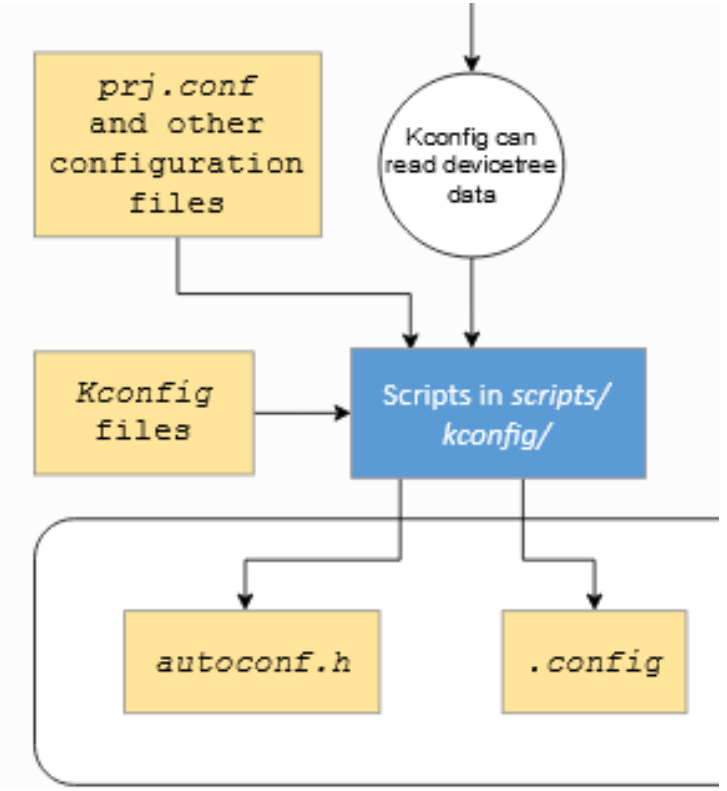
DOULOS

Zephyr®

Kconfig

- Kconfig: text file using a DSL to describe
  - Configuration items in a hierarchical manner
  - For each config item:
    - Type (boolean, int, string, choice)
    - Evt. Valid range
    - direct / reverse dependencies to other config options

- Customize zephyr to match application/platform requirements:
  - Select what feature to enable/disable
  - CONFIG_xxx in prj.conf (or board overlay)



Zephyr Build and Configuration Phases

```
# prj.conf
CONFIG_GPIO=y
CONFIG_BLINKY=y
```

```
$ find -name "autoconf.h" -o -name ".config"
./build/zephyr/include/generated/zephyr/autoconf.h
./build/zephyr/.config
```

Compile Time Modularity

main.c

```
#if CONFIG_BLINKY
#include "blinky.h"
#define TEN_SECONDS_IN_MS 10000

int main()
{
    return blinky(TEN_SECONDS_IN_MS/CONFIG_BLINKY_SPEED);
}

#else /*BLINKY demo is not enabled*/
...
#endif
```

autoconf.h

```
#define CONFIG_SERIAL 1
...
#define CONFIG_BLINKY 1
#define CONFIG_BLINKY_SPEED 10
```

Automatically included

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(blinky)

target_sources(app PRIVATE src/main.c)
target_sources_ifdef(CONFIG_BLINKY app PRIVATE src/blinky.c)
```

Add source to project if set

Agenda

1. Motivation
2. Kconfig, autoconf.h and conditional compilation
3. Size optimisations using SysV voodoo
4. DT macro, the good part
5. Driver API: abstraction and optimisations

DOULOS

Zephyr®

Everything is a Library

```
$ cd build; find . -name "*.a"
./modules/hal_nxp/hal_nxp/lib..__modules__hal__nxp.a
./zephyr/arch/arch/arm/core/…/libarch__arm__core__cortex_m__cmse.a
./zephyr/arch/arch/arm/core/…/libarch__arm__core__cortex_m.a
./zephyr/arch/arch/arm/core/mpu/libarch__arm__core__mpu.a
./zephyr/arch/arch/arm/core/libarch__arm__core.a
./zephyr/arch/common/libisr_tables.a
./zephyr/arch/common/libarch__common.a
./zephyr/lib/libc/picolibc/liblib__libc__picolibc.a
./zephyr/lib/libc/common/liblib__libc__common.a
./zephyr/libzephyr.a
./zephyr/drivers/mfd/libdrivers__mfd.a
./zephyr/drivers/clock_control/libdrivers__clock_control.a
./zephyr/drivers/gpio/libdrivers__gpio.a
./zephyr/drivers/console/libdrivers__console.a
./zephyr/drivers/serial/libdrivers__serial.a
./zephyr/drivers/pinctrl/libdrivers__pinctrl.a
./zephyr/drivers/timer/libdrivers__timer.a
./zephyr/kernel/libkernel.a
./zephyr/boards/nxp/frdm_mcxn947/libboards__nxp__frdm_mcxn947.a
./app/libapp.a
```

Vendor HAL

arch. specific

libc hooks

drivers

kernel

our app

Objects Size

```
$ find build/ -name "*.a" -exec du -bc {} +

414672  ./modules/hal_nxp/…/lib..__modules__hal__nxp.a
 22630  ./zephyr/arch/arch/arm/…/libarch__arm__core__cortex_m__cmse.a
139138  ./zephyr/arch/arch/arm/…/libarch__arm__core__cortex_m.a
 52084  ./zephyr/arch/arch/arm/core/mpu/libarch__arm__core__mpu.a
 27518  ./zephyr/arch/arch/arm/core/libarch__arm__core.a
 13072  ./zephyr/arch/common/libisr_tables.a
  3370  ./zephyr/arch/common/libarch__common.a
 27996  ./zephyr/lib/libc/picolibc/liblib__libc__picolibc.a
 23328  ./zephyr/lib/libc/common/liblib__libc__common.a
345430  ./zephyr/libzephyr.a
 18524  ./zephyr/drivers/mfd/libdrivers__mfd.a
 27402  ./zephyr/drivers/clock_control/libdrivers__clock_control.a
 40094  ./zephyr/drivers/gpio/libdrivers__gpio.a
 10544  ./zephyr/drivers/console/libdrivers__console.a
 50790  ./zephyr/drivers/serial/libdrivers__serial.a
 20130  ./zephyr/drivers/pinctrl/libdrivers__pinctrl.a
 35374  ./zephyr/drivers/timer/libdrivers__timer.a
739624  ./zephyr/kernel/libkernel.a
 50974  ./zephyr/boards/nxp/…/libboards__nxp__frdm_mcxn947.a
 19004  ./app/libapp.a

2081698 total
```

"Objects"	Size
Code	~2.0 Mb
Picolibc	~1.6 Mb
Total	~3.6 Mb

Image Size

```
$ find . -name "zephyr*" -type f -executable -exec du {} +
```

objcopy

Size Kb	Path (relative to build)	Comments
668	./zephyr/zephyr.elf	Binary using ELF format
24	./zephyr/zephyr.bin	Binary flash on target
664	./zephyr/zephyr_pre0.elf	Intermediary build file / internal use only

Image flashed on target is 24 Kb

sysV voodoo

Question? How do we go from 3.6 MB → 668 Kb → 24 Kb

Build post-processing

Ancient SysV Voodoo

- Archive is a collection of objfiles (compilation unit)

```
$ ar -t zephyr/kernel/libkernel.a
main_weak.c.obj
…
sem.c.obj
timer.c.obj
```

- Linker discards unreferenced .o files if no symbol is needed.

```
$ nm zephyr/kernel/libkernel.a
sem.c.obj:
00000000 b lock
00000000 T z_impl_k_sem_give
00000000 T z_impl_k_sem_init
00000000 T z_impl_k_sem_reset
00000000 T z_impl_k_sem_take
         U z_pend_curr
         U z_ready_thread
         U z_reschedule
         U z_unpend_first_thread
```

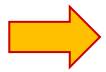
It's ancient SysV voodoo, basically

Andy Ross
Kernel dev.

Discarded in zephyr.elf when semaphore is not used

Agenda

1. Motivation
2. Kconfig, autoconf.h and conditional compilation
3. Size optimisations using SysV voodoo
4. DT macro, the good part
5. Driver API: abstraction and optimisations



DOULOS

Zephyr®

Devicetree

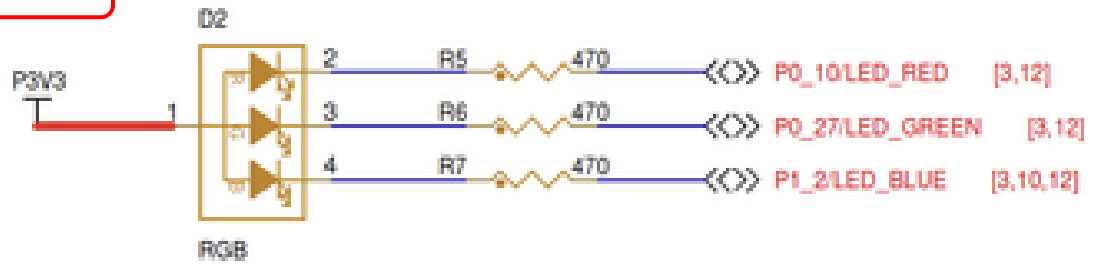
Devicetree: tree-like data structure using a DSL to describe:

- Hardware: SoC, CPUs, Memory, Buses, Peripherals,…
- Hardware Boot-time configuration

devicetree specification

GPIO0, PIN 10, ACTIVE_LOW

```
red_led: led_3 {
    gpios = < &gpio0 0xa 0x1 >;
    label = "Red LED";
    status = "okay";
};
```



The devicetree contains the information from the reference manual / schematic needed by the corresponding drivers!

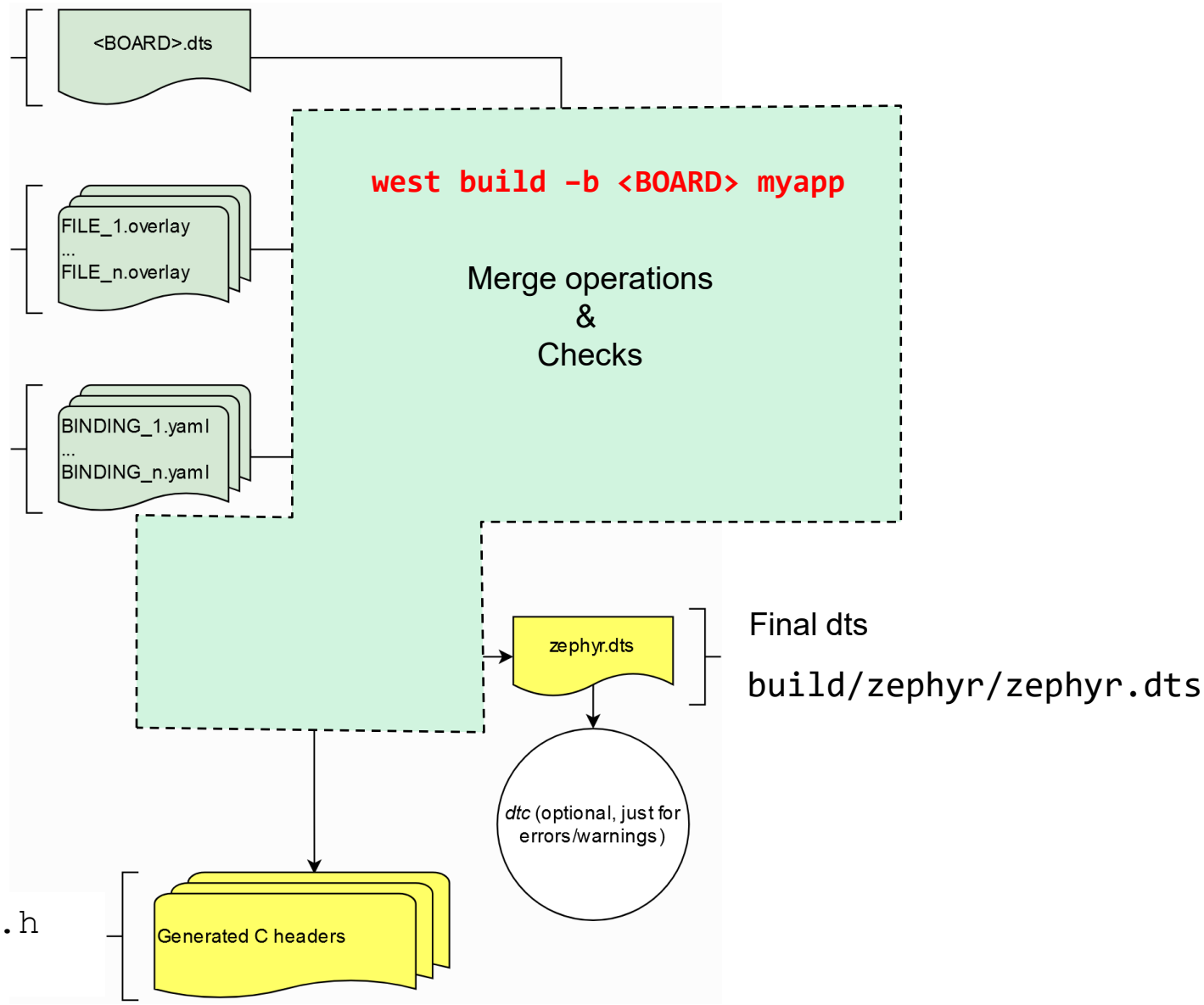
Devicetree Files

Board DT files

coming from Zephyr

```
myapp
├── board/
│   └── <BOARD>.overlay
├── CMakeLists.txt
...
```

Contains rules for devicetree contents



devicetree_generated.h

Use <devicetree.h>

DT macros – looking through the glass

```
11  /* The devicetree node identifier for the "led0" alias. */
12  #define LED0_NODE DT_ALIAS(led0)
13
14  /*
15   * A build error on this line means your board is unsupported.
16   * See the sample documentation for information on how to fix this.
17   */
18  static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
19
```

```
aliases {
    led0 = &red_led;
    …
};                    zephyr.dts
```

```
red_led: led_3 {
    gpios = < &gpio0 0xa 0x1 >;
    label = "Red LED";
    status = "okay";
};                              zephyr.dts
```

```
gpio0: gpio@96000 {
     compatible = "nxp,kinetis-gpio";
     status = "okay";
     reg = < 0x96000 0x1000 >;
     gpio-controller;
…
};                              zephyr.dts
```

expands to:

```
{ .port = (&__device_dts_ord_11),
  .pin = 10,
  .dt_flags = 1,
}
```

Provided by:

```
/* GPIO SoC/board driver:
 *   zephyr/driver/gpio/gpio_mcux.c
 */
#define DT_DRV_COMPAT nxp_kinetis_gpio
...
DT_INST_FOREACH_STATUS_OKAY(GPIO_DEVICE_INIT_MCUX)
```

DT Macros – The Good Parts

- "DT APIs" are just C-macros
  - Expand to scalar value or struct/array initializer
- Devicetree is queried during build time
  - no runtime penalty (speed/size)
- When it works, relatively easy to use from an application point of view
  - like APIs
- Bad part: when it doesn't work
  - Example: typo in property name
  - Driver for device is not enabled: __device_ord_dts_XX symbol not found
- Ugly part: in the driver

since v4.3: dtdoctor

Agenda

1. Motivation
2. Kconfig, autoconf.h and conditional compilation
3. Size optimisations using SysV voodoo
4. DT macro, the good part
5. Driver API: abstraction and optimisations

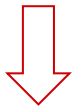
DOULOS

Zephyr®

Toggle LED… How many Calls?

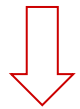
```
ret = gpio_pin_toggle_dt(&led);
```

```
static inline int gpio_pin_toggle_dt(const struct gpio_dt_spec *spec)
{
    return gpio_pin_toggle(spec->port, spec->pin);
}
```



```
static inline int gpio_pin_toggle(const struct device *port, gpio_pin_t pin)
{   ...   }
```

```
static inline int gpio_port_toggle_bits(const struct device * port, gpio_port_pins_t pins)
{   ...   }
```



```
static inline int z_impl_gpio_port_toggle_bits(const struct device *port, gpio_port_pins_t pins)
{   ...   }
```

```
static int gpio_mcux_port_toggle_bits(const struct device *dev, uint32_t mask)
{ ... gpio_base->PTOR = mask; ... }
```

GPIO driver

Toggle LED… looking through the asm glass

- 5 or 6 level of indirections… But the functions are:
  - declared **static inline**
  - in header file

Good candidates for compiler optimization

```
ret = gpio_pin_toggle_dt(&led);
```

```
=> 0x100012d8 <+68>:    ldr     r3, [r6, #8]
   0x100012da <+70>:    mov.w   r1, #1024       ; 0x400
   0x100012de <+74>:    mov     r0, r6
   0x100012e0 <+76>:    ldr     r3, [r3, #20]
   0x100012e2 <+78>:    blx     r3
```

```
static int gpio_mcux_port_toggle_bits(const struct device *dev, uint32_t mask)

Dump of assembler code for function gpio_mcux_port_toggle_bits:
   0x100044c2 <+0>:     ldr     r3, [r0, #4]
   0x100044c4 <+2>:     movs    r0, #0
   0x100044c6 <+4>:     ldr     r3, [r3, #4]
=> 0x100044c8 <+6>:     str     r1, [r3, #76]   ; 0x4c
   0x100044ca <+8>:     bx      lr
```

Production code

Conclusions

- Zephyr uses:
  - Kconfig and Devicetree
  - Good'old linker techniques to minimize the code size
  - Hints to compilers for optimizing indirections from abstractions

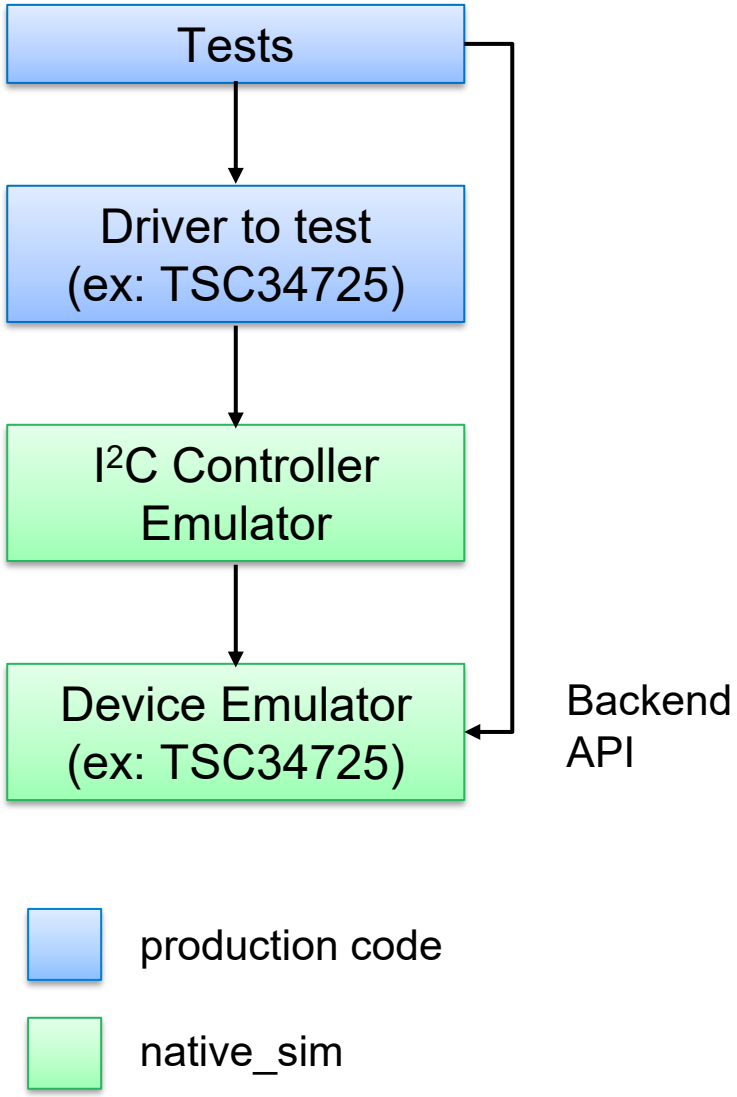
Zephyr removes the strong dependencies to hardware!

- Example:

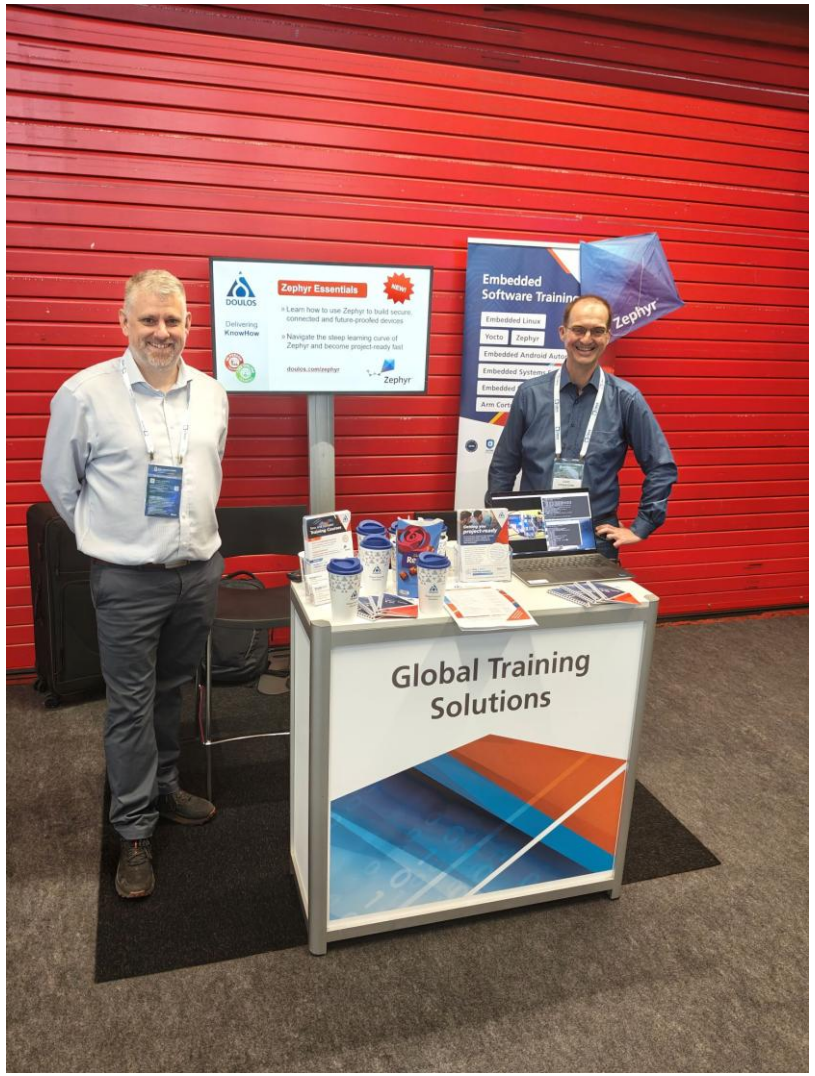
  native_sim can be used to develop and test without hardware

https://github.com/NXPHoverGames/Doom-MCX

https://zswatch.dev/



Q&A and looking forward!



Doulos Zephyr Essential Training

https://www.doulos.com/zephyr

Connect with Loïc

https://www.linkedin.com/in/loicdomaigne/

https://chat.zephyrproject.org/

ljd42_63516

Q&A time (now or later)

THE END.

DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification	» SystemVerilog » UVM » Formal » SystemC » TLM-2.0
FPGA & Hardware Design	» VHDL » Verilog » SystemVerilog » Tcl » AMD » cocotb
Embedded Software	» Emb C/C++ » Emb Linux » Yocto » RTOS » Security » Android » Arm » Rust » Zephyr
Python, Edge AI & Machine Learning	 

SystemVerilog OUT OF THE BOX · UVM · SYSTEMC · yocto PROJECT PARTICIPANT · Zephyr · freeRTOS · SECURE EMBEDDED · arm APPROVED TRAINING PARTNER