

How I fell in Love with Zephyr

Dr. Tobias Kästner

ESE Kongress, 02-05.12.2024, Sindelfingen

Team inovex

*Karlsruhe · Köln · München · Hamburg
Berlin · Stuttgart · Pforzheim · Erlangen*

Dr. Tobias Kästner



Tobias Kaestner



@tobiaskaestner



@tobiaskaestner

Solution Architect Medical IoT

#FOSS4MEDICAL

- PhD in Physics
- 15+ years SW/System Architect
 - mainly Medical Devices
- Trainer & Technical Consultant
 - SW-Architecture, Zephyr, Yocto
- In Love w/ Zephyr since 2016
 - realised several prototype projects for life-science R&D
 - Maintainer of TiacSys-Bridle Project
 - Participant Zephyr Safety-WG & Zephyr TSC
- Inovex Zephyr Project Silver Member since Nov 2024

Agenda

- Challenges of Embedded Systems Development
- The Zephyr (RTOS) Framework
- SW-Features and Variant Management
- Modern Build System capabilities
- Hardware Abstraction as dependency injection
- Conclusion

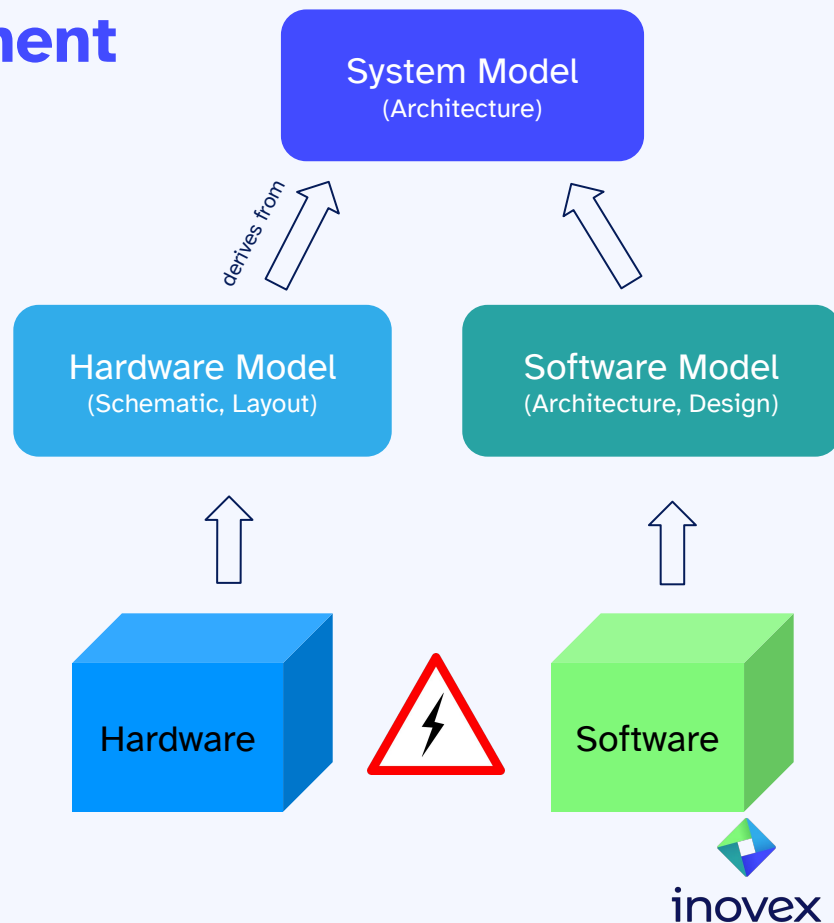


Challenges of Embedded Systems Development

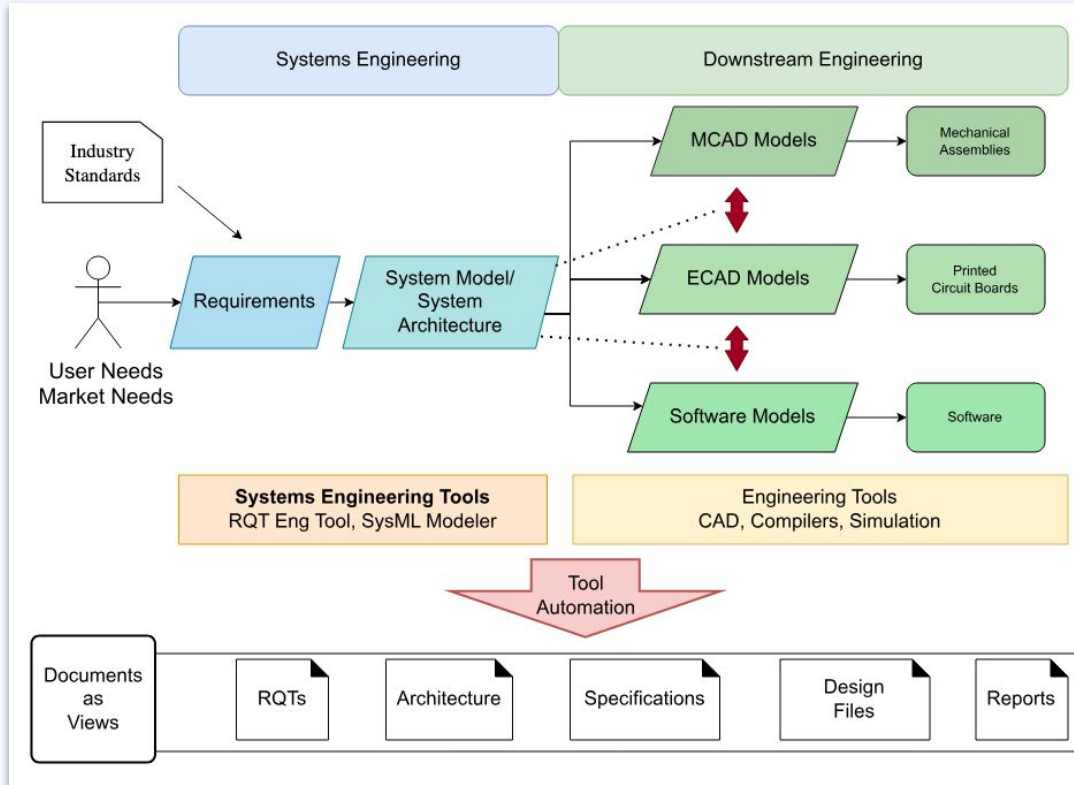
Embedded Systems Development

Spans multiple domains:

- System Development
 - functional decomposition
- Hardware Development
 - electrical/mechanical
- Software Development
 - information processing/exchange



Model based (embedded) Systems Engineering

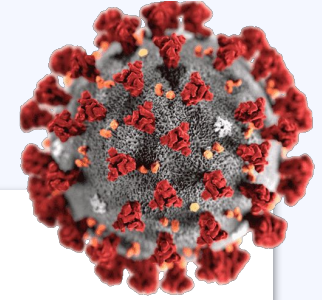


Key Challenges of Embedded Development

- how to make sure **models do align**
- propagate **changes** consistently
- deal with **implicit** models

Introducing ACME-NG

Grafik: Center for Disease Control and Prevention, public domain



A couple of years ago ...

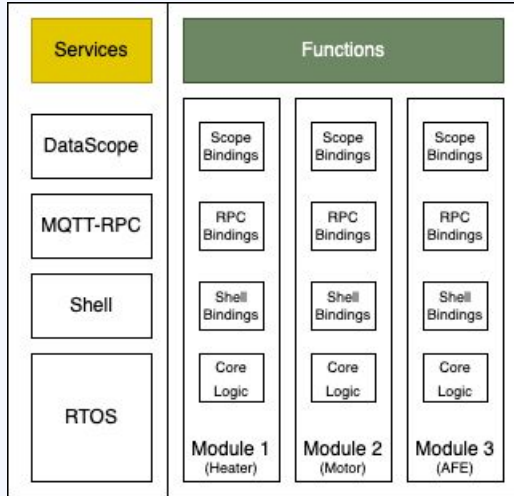
Goals of ACME : develop new type of high-performance test to diagnose Covid-19

Iterative system design - basic system functions known but specific details dependent on reagent chemistry developed simultaneously

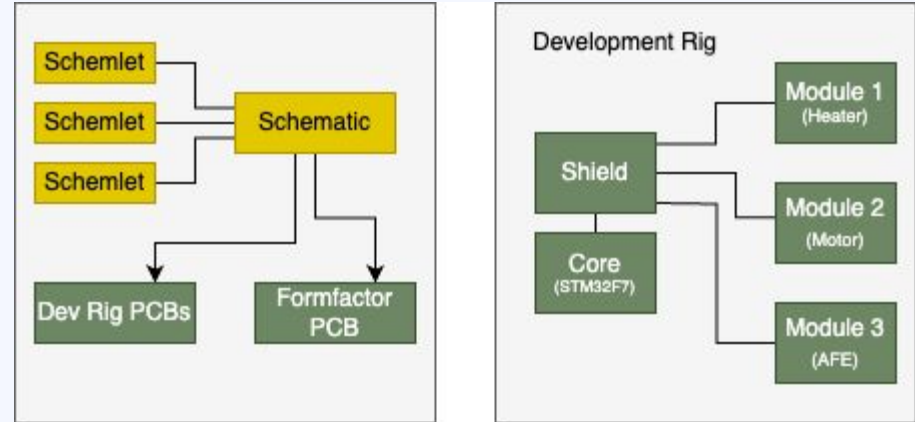
Time to Market - extremely time-sensitive due to ongoing pandemic

Supply-Chain-Risks - Availability of HW components worsened dramatically during project time

Introducing ACME-NG



Software Architecture



(Agile) Hardware Architecture

The Zephyr (RTOS) Framework

A first look at Zephyr



Source: wikipedia

Zephyr[®]

- launched in 2016 as an RTOS specifically built for the IoT
- fully open source (Apache 2.0) project run by the Linux Foundation
- in 2024 most active open source RTOS project reaching 100,000 contributions
- strong focus on security and connectivity (safety coming)

More info <https://www.zephyrproject.org/wp-content/uploads/2024/10/Zephyr-Overview-20241017.pdf>

Zephyr as an RTOS Kernel

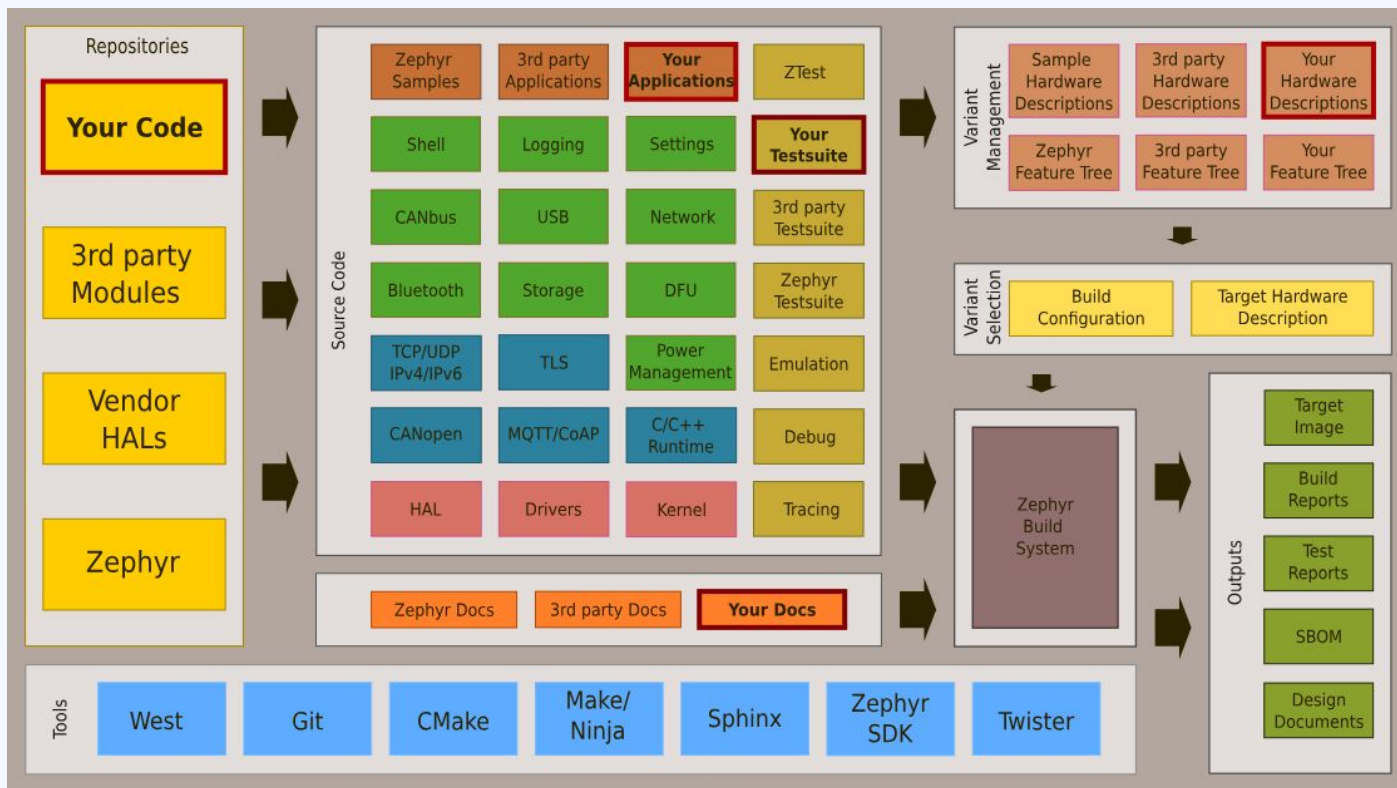


Source: wikipedia

Zephyr[®]

- supports cooperative and pre-emptive thread scheduling for uni-core and SMP
- Kernel mostly used in tickless mode today (Power management)
- supports most standard RTOS primitives for synchronization and data passing
- includes higher level abstractions for advanced data passing
 - event driven pub/sub topologies (ZBus)
 - message-passing for heterogeneous multi-core systems (openAMP)
- OS support for tracing and profiling

Zephyr as an Embedded Development Framework



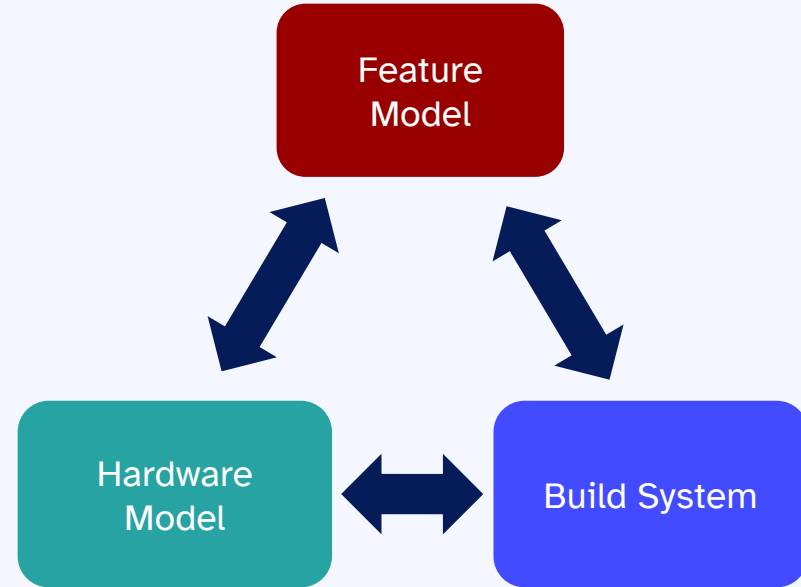
Models in Zephyr

3 domain-specific models at play

Feature Model: select desired functionality

Hardware Model: to describe hardware properties

Build System: to describe build process

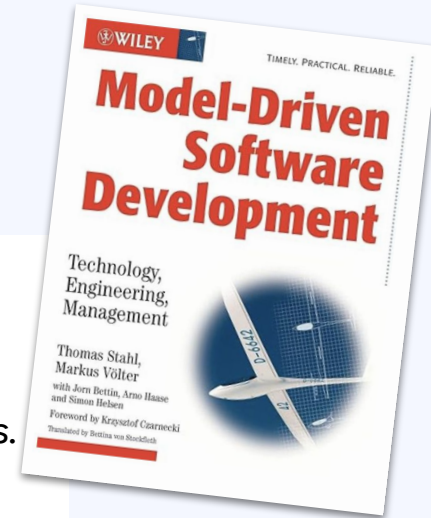


Model Driven Software Development

“ To successfully apply the ‘domain-specific model’ concept, **three requirements** must be met:

- **Domain-specific languages** are required to allow the [...] formulating of models.
- Languages that can express [...] **model-to-code transformations** are needed.
- **Compilers, generators or transformers** are required that can run the transformations to generate code executables.”

MDSD by Stahl and Voelker, 2006

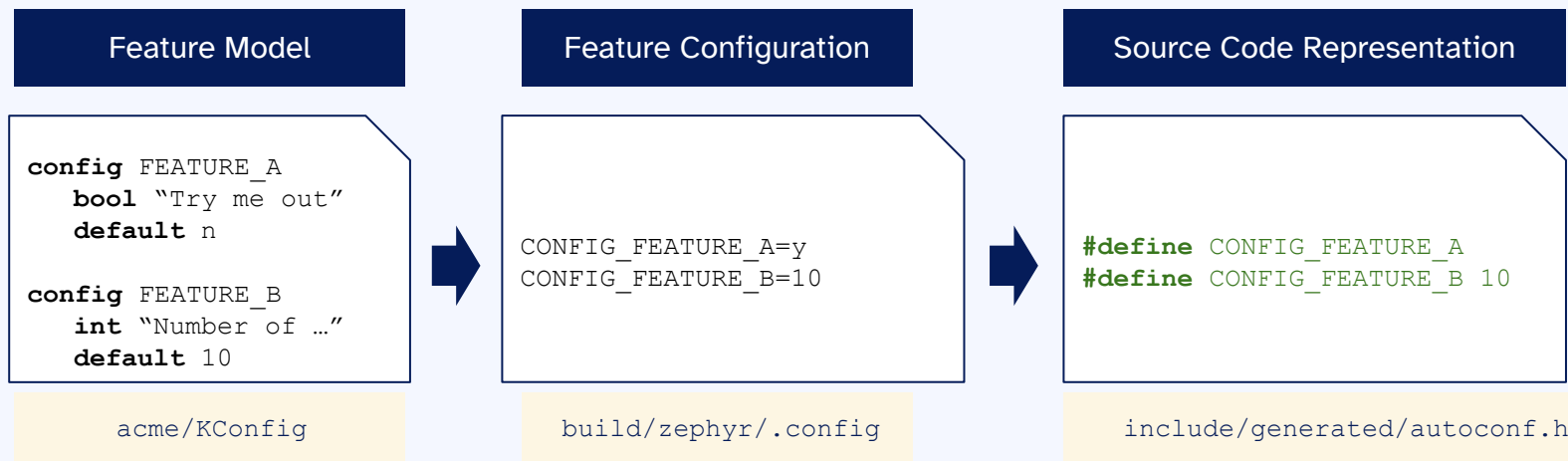


SW Features & Variant Management

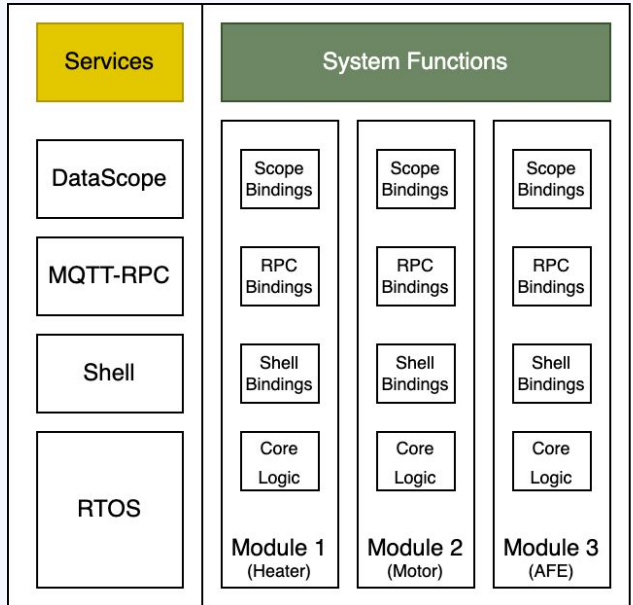
Modeling Software Features w/ Kconfig

Kconfig is a domain-specific language to describe software feature models

- features are typed & can relate to each other (**select**, **depend**, **imply**)
- models can be composed from smaller models (**[or] source**)
- models are transformed at build time into C language constructs



Modeling Software Features w/ Kconfig



```
1 menu "ACME Subsystems"
2
3 menu "Modules"
4     rsource "m_heater/Kconfig"
5     rsource "m_motor/Kconfig"
6     rsource "m_afe/Kconfig"
7 endmenu #Modules
8
9 menu "Core Services"
10     rsource "s_mqtt_rpc"
11     rsource "s_datascope"
12 endmenu #Core Services
13 endmenu #ACME Subsystems
```

System Model
(Architecture)

Software Model
(Architecture, Design)

Modeling Software Features w/ Kconfig

```
1 menu "ACME Subsystems"
2
3 menu "Modules"
4 rsource "m_heater/Kconfig"
5 rsource "m_motor/Kconfig"
6 rsource "m_afe/Kconfig"
7 endmenu #Modules
8
9 menu "Core Services"
10 rsource "s_mqtt_rpc"
11 rsource "s_datascopes"
12 endmenu #Core Services
13 endmenu #ACME Subsystems
```

```
1 menuconfig ACME_SUBSYS_HEATER # option to toggle the entire subsystem on/off
2 bool "Heater subsystem"
3 help
4 | The Heater subsystem is responsible for measuring and controlling
5 | the temperature.
6
7 if ACME_SUBSYS_HEATER
8
9 config ACME_SUBSYS_HEATER_THREAD_STACK_SIZE
10 | int "Stack size of subsystem thread"
11 | default 2048
12
13 config ACME_SUBSYS_HEATER_MQTT_RPC
14 | bool "Enable MQTT-RPC bindings for $(subsys-str) subsystem"
15 | depends on ACME_MQTT_RPC
16
17 config ACME_SUBSYS_HEATER_SHELL
18 | bool "Enable shell bindings for $(subsys-str) subsystem"
19 | depends on SHELL
20
21 config ACME_SUBSYS_HEATER_SCOPE
22 | bool "Enable data scope bindings for $(subsys-str) subsystem"
23 | depends on ACME_SCOPE #only selectable if datascopes is enabled
24
25 endif
```

Modern Build System Capabilities

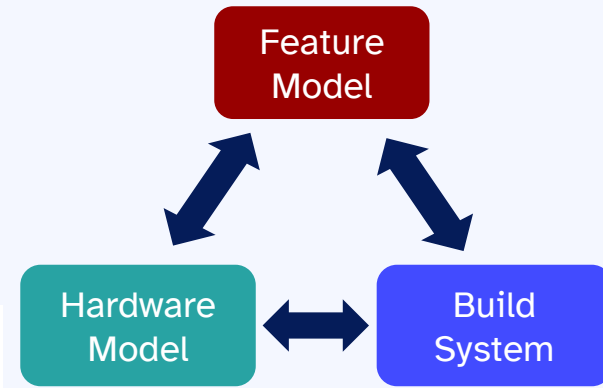
The Zephyr Build System

Built on-top of CMake by Kitware

- cross-platform
- support for multiple build systems

Custom extensions add additional features:

- Responsive to Feature tree selected for build
- Hardware model aware:
 - Selection of appropriate cross-compiler
 - Construction of Linker file



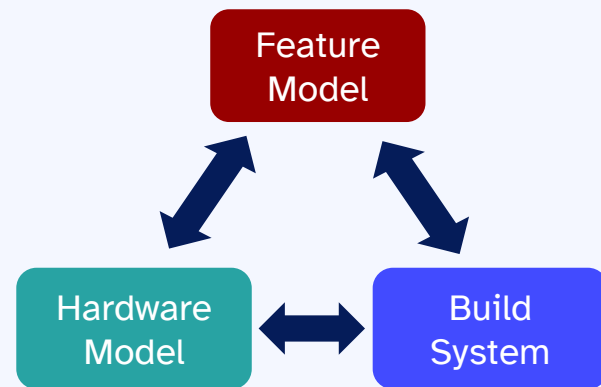
The Zephyr Build System

Convention over Configuration

Complexity hidden by default
to reduce cognitive load

yet all variation points fully exposed

Overwrite with overlays and during command invocation



```
west build -b nucleo_g474re samples/hello_world
```

invoke CMake build
system

select hardware
& features
<BOARD>.dts
<BOARD>_defconfig

select build system entry point
CMakeLists.txt
and application & features
prj.conf

The Zephyr Build System

Integration with Feature model allows conditional compilation

- Uniform pattern to integrate middleware and optional components
- Key to keep 9+ architectures and 280+ SoCs in single source tree
- Extensible by downstream applications

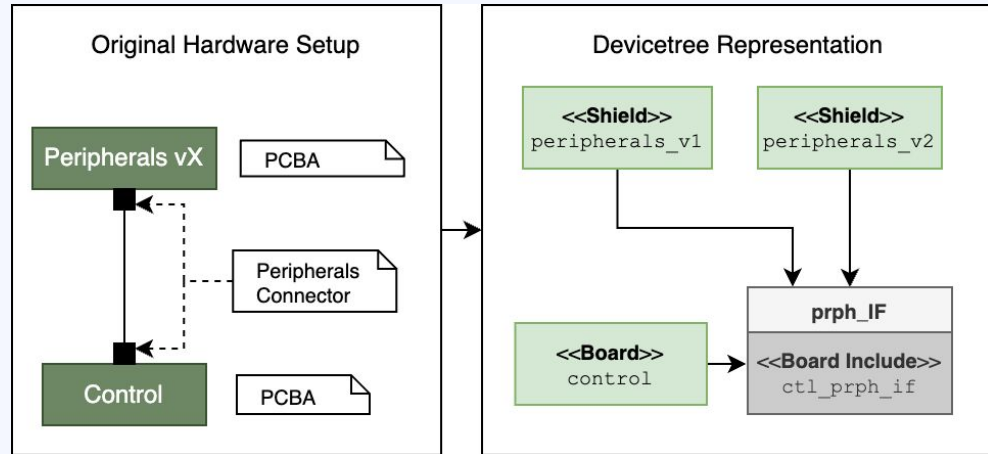
```
1  zephyr_library_named(acme-heater)
2
3  zephyr_library_sources(heater.c)
4  zephyr_library_sources_ifdef(CONFIG_ACME_SUBSYS_HEATER_SHELL heater_shell.c)
5  zephyr_library_sources_ifdef(CONFIG_ACME_SUBSYS_HEATER_MQTT_RPC heater_mqtttrpc.c)
6  zephyr_library_sources_ifdef(CONFIG_ACME_SUBSYS_HEATER_SCOPE heater_scope.c)
7
```

Hardware abstraction as Dependency injection

Modeling Hardware Features w/ Devicetree

Devicetree is a domain-specific language to describe **hardware properties** which are **software relevant**

If used correctly, HW setups can be mapped faithfully to devicetree models including hardware interfaces aka board interconnects



Modeling Hardware Features w/ Devicetree

```
/ {
    model = "Nordic nRF52840 DK NRF52811";
    compatible = "nordic,nrf52840-dk-nrf52811";

    aliases {
        led0 = &led0;
    };

    leds {
        compatible = "gpio-leds";
        led0: led_0 {
            gpios = <&gpio0 13 GPIO_ACTIVE_LOW>;
            label = "Green LED 0";
        };
    };

    gpio0: gpio@50000000 {
        gpio-controller;
        compatible = "nordic,nrf-gpio";
        reg = < 0x50000000 0x200 0x50000500 0x300 >;
        #gpio-cells = < 0x2 >;
        status = "okay";
        ...
    };
}
```

scripts/dts/gen_defines.py

#include <generated/devicetree_generated.h>

```
#include <zephyr/devicetree.h>

#define LED0_NODE DT_ALIAS( led0)

static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE,
gpios);

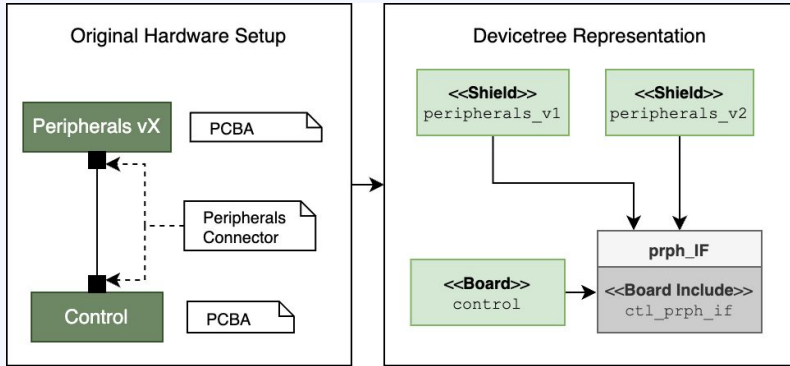
int main(void)
{
    int ret;
    bool led_state = true;

    if (!gpio_is_ready_dt(&led)) {
        return 0;
    }

    ret = gpio_pin_configure_dt(&led, GPIO_OUTPUT_ACTIVE);
    ...
}
```

Pre-processor injects
correct address, pin

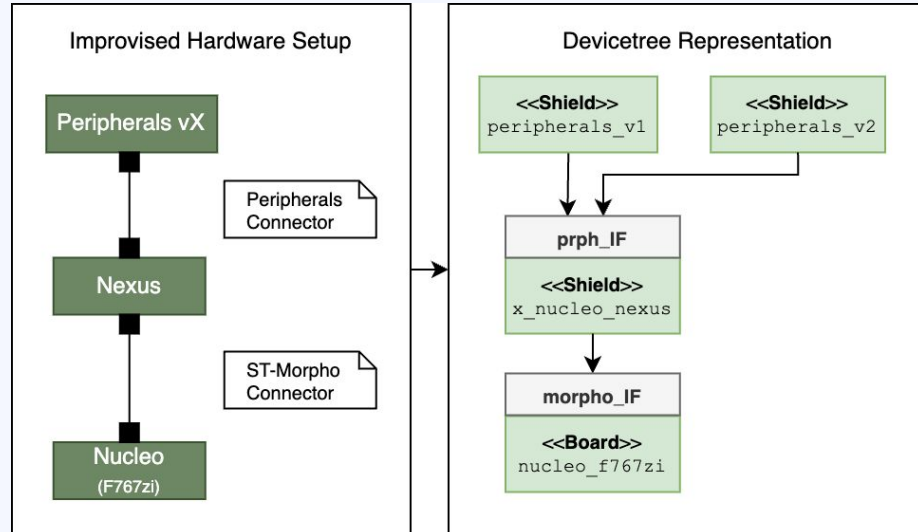
When supply chains fall apart ...



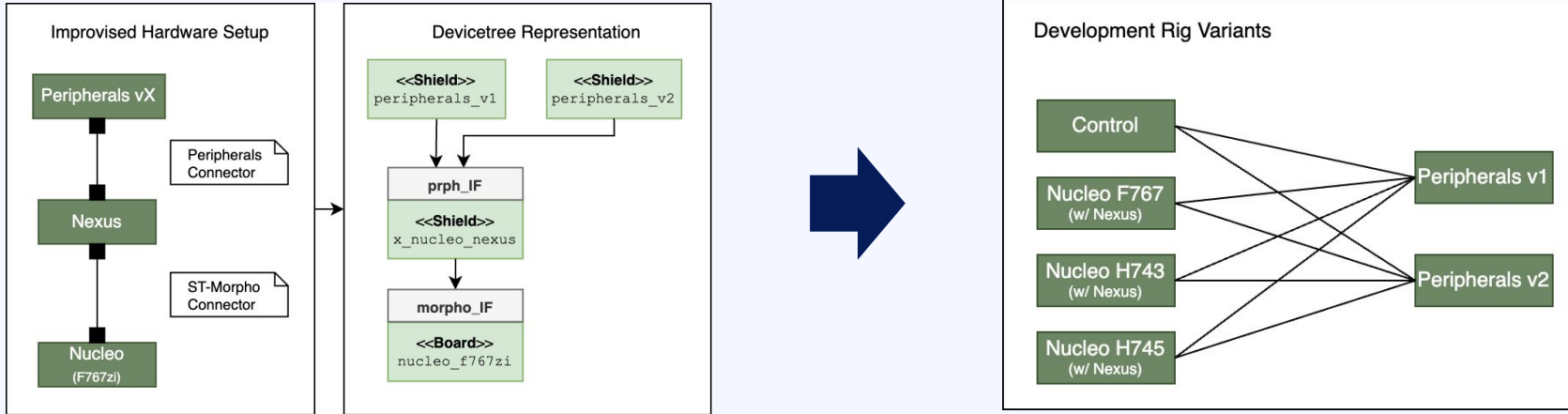
When ACME-NG needed parts to make hardware the most, the parts had disappeared ...

... and all ACME-NG could do, was to buy existing devkit boards

devicetree models allowed ACME-NG to **compensate for all HW changes** without touching a single line of source code



When supply chains fall apart ...



```
west build -b core -shield peripherals_v1 acme_app  
west build -b core -shield peripherals_v2 acme_app
```

```
west build -b nucleo_f767zi -shield x_nucleo_nexus -shield peripherals_v1 acme_app  
west build -b nucleo_h743zi -shield x_nucleo_nexus -shield peripherals_v1 acme_app
```

Conclusion

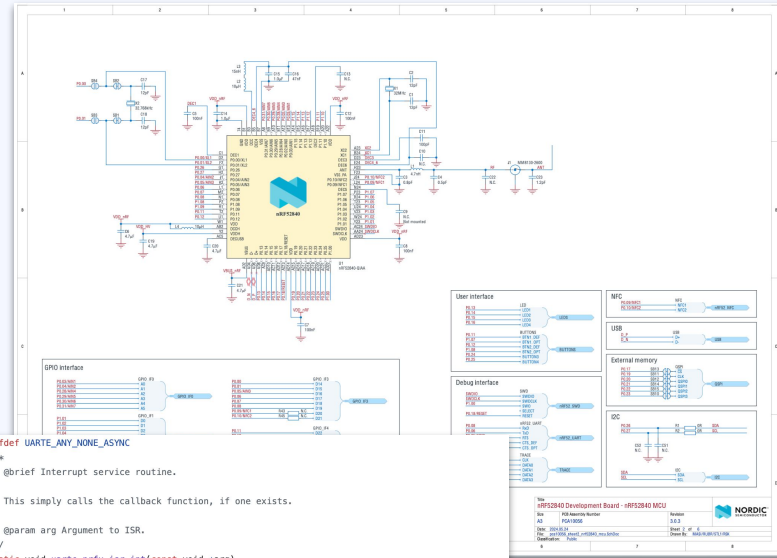
The many forms of models

Models are

- unavoidable
- abstractions
- potentially disagree with each other

Explicit models are

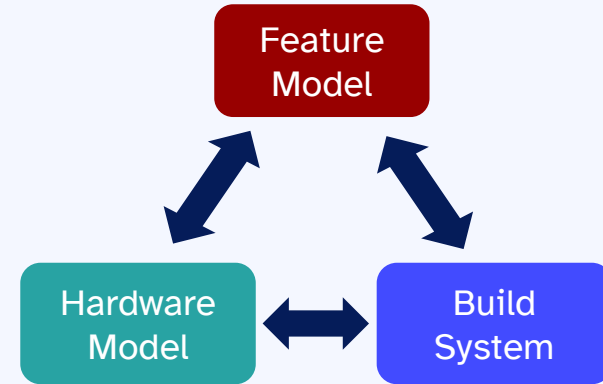
- more expressive
- at the “right” abstractions
- domain-specific
- viable ways to improve productivity



```
327 #ifndef UARTE_ANY_NONE_ASYNC
328 /*
329  * @brief Interrupt service routine.
330  *
331  * This simply calls the callback function, if one exists.
332  *
333  * @param arg Argument to ISR.
334  */
335 static void uarte_nrfx_isr_int(const void *arg)
336 {
337     const struct device *dev = arg;
338     const struct uarte_nrfx_config *config = dev->config;
339     struct uarte_nrfx_data *data = dev->data;
340     NRF_UART_Type *uarte = get_uarte_instance(dev);
341
342     /* If interrupt driven and asynchronous APIs are disabled then UART
343      * interrupt is still called to stop TX. Unless it is done using PPI.
344      */
345     if ((IS_ENABLED(UARTE_HAS_ENDTX_STOPPTX_SHORT) &&
346         nrf_uarte_int_enable_check(uarte, NRF_UART_EVENT_ENDTX_MASK) &&
347         nrf_uarte_event_check(uarte, NRF_UART_EVENT_ENDTX)) {
348         endtx_isr(dev);
349     }
350
351     bool txstopped = nrf_uarte_event_check(uarte, NRF_UART_EVENT_TXSTOPPED);
352
353     if (txstopped && IS_ENABLED(CONFIG_PM_DEVICE_RUNTIME) || LOW_POWER_ENABLED(config)) {
354         unsigned int key = irq_lock();
355
356         if (IS_ENABLED(CONFIG_PM_DEVICE_RUNTIME) &&
357             (data->flags & UARTE_FLAG_POLL_OUT)) {
358             data->flags &= ~UARTE_FLAG_POLL_OUT;
359             pm_device_runtime_put(dev);
360         } else {
361             nrf_uarte_disable(uarte);
362         }
363     }
364
365     #ifndef UARTE_INTERRUPT_DRIVEN
366     if (!data->int_driven || data->int_driven->fifo_fill_lock == 0)
367     #endif

```

What makes us go fast w/ models?



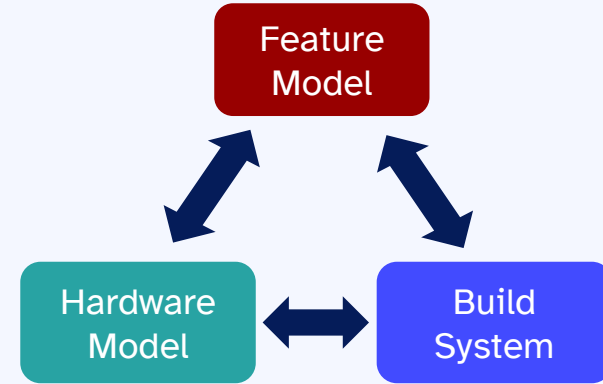
Appropriate: Devicetree, Kconfig and CMake established and mature

Textual DSL: easy to diff and version control, models as code

Automated: model transformations happen as part of software build process

Transparent: generated expressions consumable by standard C compiler

What makes us go fast w/ models?

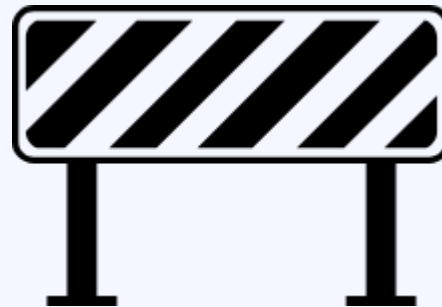


Integrable: Models can interact with each other to further increase usefulness

Extensible: Model languages can be extended with new constructs

Open: Technologies open-source -> no limitations to use or future development

No model is perfect - never



Existing models as expressed by Kconfig, Devicetree and Zephyr CMake functions already extremely powerful ...

... however, not without limitations:

- missing abstractions: connectors (interfaces & multi-instance)
- missing concepts: multi-board setups (only via `--shield ... --shield ...`)
- missing composability: CS-lines of SPI devices

Final thoughts



- Zephyr showcases MDSD techniques, not through intent but by convergence
- Productivity gains explainable through this modeling approach
- Still plenty of space for improvements:
 - What other transformations could be looked at?
 - What other domains could be modeled?

Thank You

Zephyr Hands-On Trainings

starting 2025: Jan 22/23, Apr 02/03, Jul 02/03

Find out more

<https://www.inovex.de/de/training/zephyr-basic-training/>



Dr. Tobias Kästner
Solution Architect Medical IoT

tobias.kaestner@inovex.de

+49 152 3314 8940

Allee am Röthelheimpark 11,
91052 Erlangen



Tobias Kaestner



@tobiaskaestner



@tobiaskaestner

