Rethinking Embedded Development:

Zephyr Through the Eyes of Model Driven Engineering

Team inovex
Karlsruhe · Köln · München · Hamburg
Berlin · Stuttgart · Pforzheim · Erlangen

inovex

Dr. Tobias Kästner

Tobias Kaestner

@tobiaskaestner

@tobiaskaestner

Solution Architect Medical IoT

#FOSS4MEDICAL

- PhD in Physics (long ago)
- SW/System Architect since 15 years
  - 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

inovex

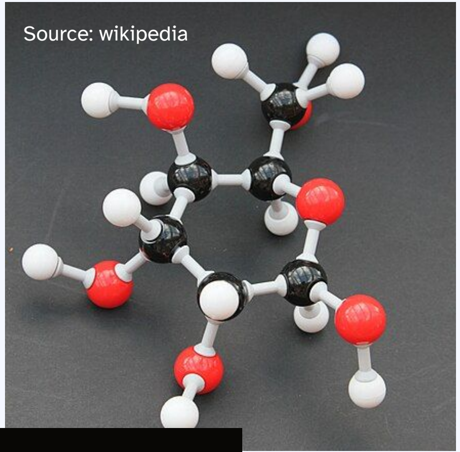
Agenda for today

- A very short 101 on MDSD
- The many DSLs of Zephyr
- The power of models

inovex

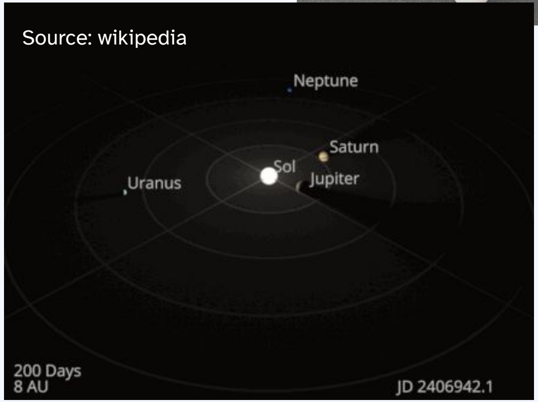
A very short 101 on MDSD

What is a model - and if, how many?



Models are sense-making devices to

- encode information about the world
- and reason about its properties
- **communicate** our understanding &
- make predictions

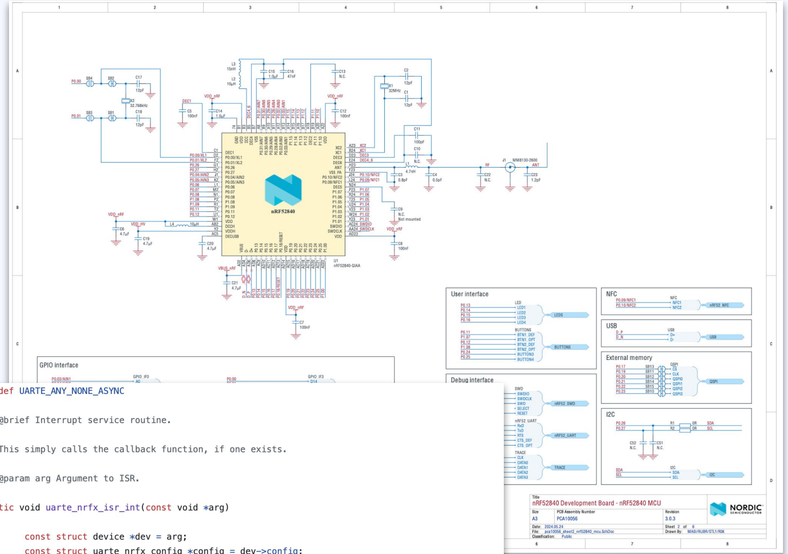


inovex

The many forms of models

Models are

- unavoidable
- abstractions
- domain-specific
- not guaranteed to agree with each other




```c
#ifdef UARTE_ANY_NONE_ASYNC
/**
 * @brief Interrupt service routine.
 *
 * This simply calls the callback function, if one exists.
 *
 * @param arg Argument to ISR.
 */
static void uarte_nrfx_isr_int(const void *arg)
{
	const struct device *dev = arg;
	const struct uarte_nrfx_config *config = dev->config;
	struct uarte_nrfx_data *data = dev->data;
	NRF_UARTE_Type *uarte = get_uarte_instance(dev);

	/* If interrupt driven and asynchronous APIs are disabled then UART
	 * interrupt is still called to stop TX. Unless it is done using PPI.
	 */
	if (!IS_ENABLED(UARTE_HAS_ENDTX_STOPTX_SHORT) &&
	    nrf_uarte_int_enable_check(uarte, NRF_UARTE_INT_ENDTX_MASK) &&
		nrf_uarte_event_check(uarte, NRF_UARTE_EVENT_ENDTX)) {
		endtx_isr(dev);
	}

	bool txstopped = nrf_uarte_event_check(uarte, NRF_UARTE_EVENT_TXSTOPPED);

	if (txstopped && (IS_ENABLED(CONFIG_PM_DEVICE_RUNTIME) || LOW_POWER_ENABLED(config))) {
		unsigned int key = irq_lock();

		if (IS_ENABLED(CONFIG_PM_DEVICE_RUNTIME) &&
		    (data->flags & UARTE_FLAG_POLL_OUT)) {
			data->flags &= ~UARTE_FLAG_POLL_OUT;
			pm_device_runtime_put(dev);
		} else {
			nrf_uarte_disable(uarte);
		}

#ifdef UARTE_INTERRUPT_DRIVEN
		if (!data->int_driven || data->int_driven->fifo_fill_lock == 0)
#endif
```


inovex

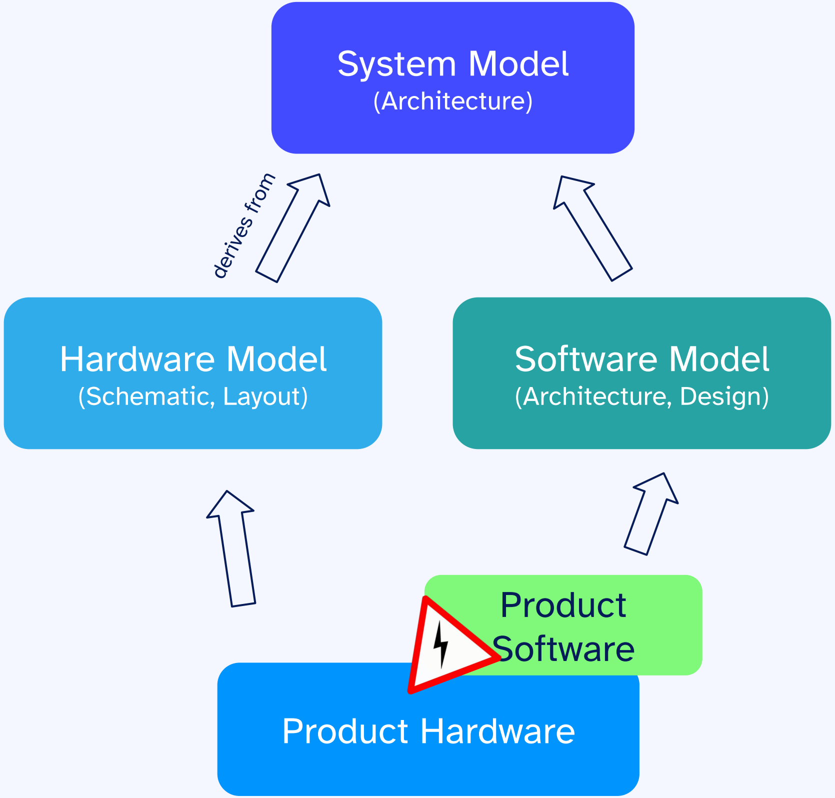
Models in Embedded

Embedded Development comprises of

- System Development Domain
- Hardware Development Domain
- Software Development Domain

Key Challenges of Embedded Development

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



inovex

Models in Embedded

System Model
(Architecture)

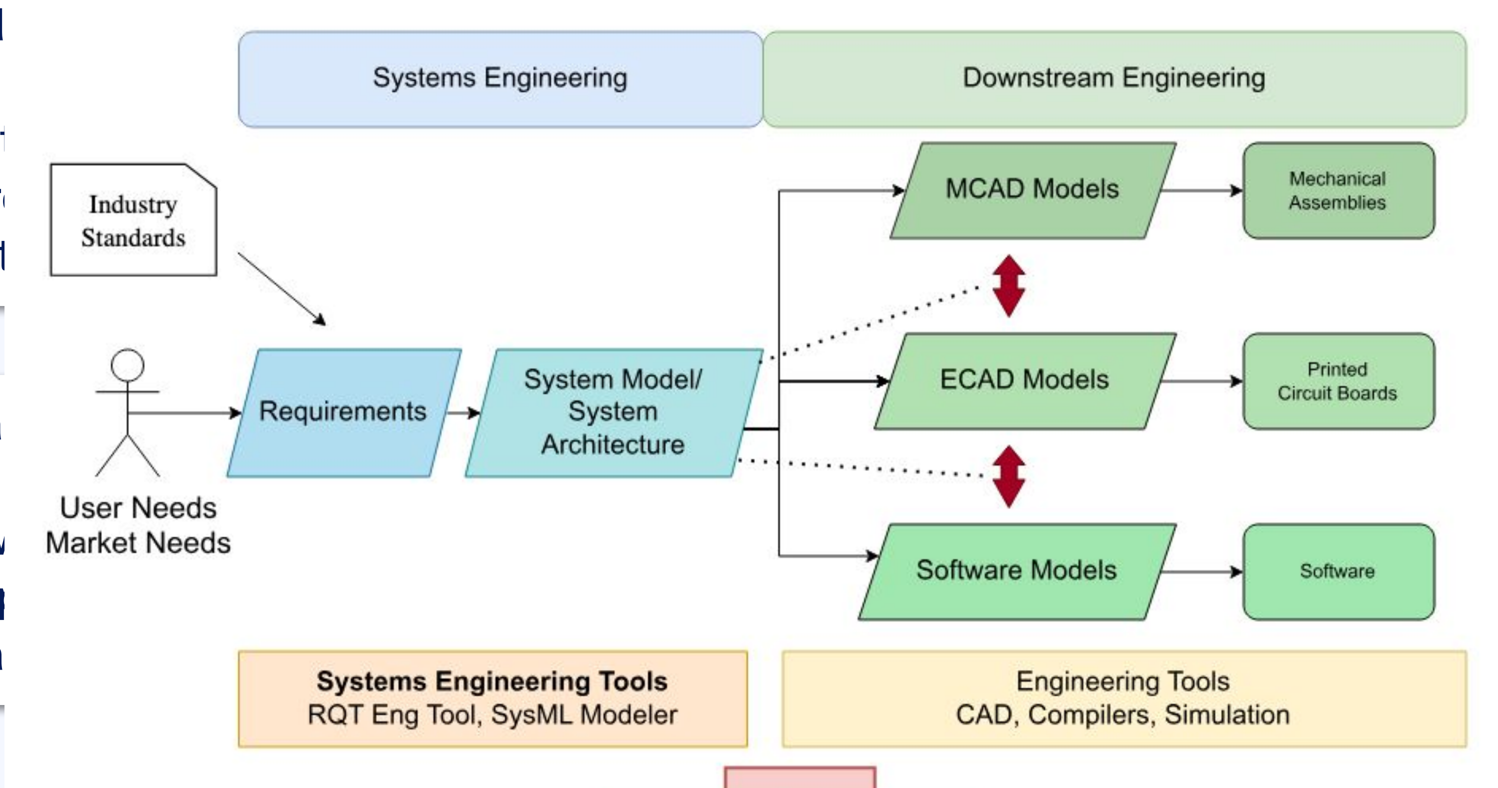
Embedd

- Syst
- Har
- Soft

Key Cha

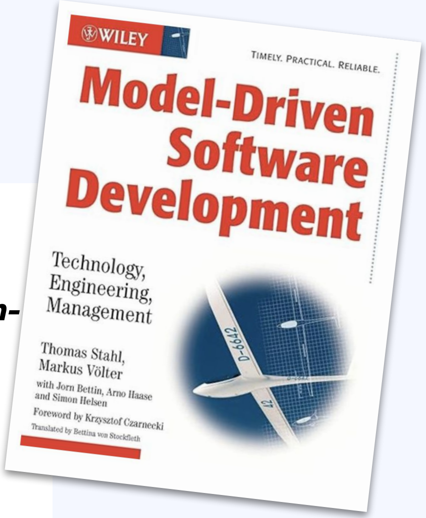
- how
- prop
- dea

Model
Design)



novex

Model Driven Software Development



*"MDSD therefore aims to find **domain-specific abstractions** and make them accessible through formal modeling. This procedure creates a great potential for **automation** of software production, which in turn leads to **increased productivity**. Moreover, both the **quality and maintainability** of software systems increase.[…] The adjective 'driven' in 'Model-Driven Software Development' […] emphasizes that this paradigm assigns models a **central** […] **role**: they are at least **as important as source code**.*

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

- ***Domain-specific languages** are required to allow the actual formulating of models.*
- *Languages that can express the necessary **model-to-code transformations** are needed.*
- ***Compilers, generators or transformers** are required that can run the transformations to generate code executable on available platforms."*

MDSD by Stahl and Voelker, 2006

https://voelter.de/data/books/mdsd-en.pdf

inovex

The many DSLs of Zephyr

inovex

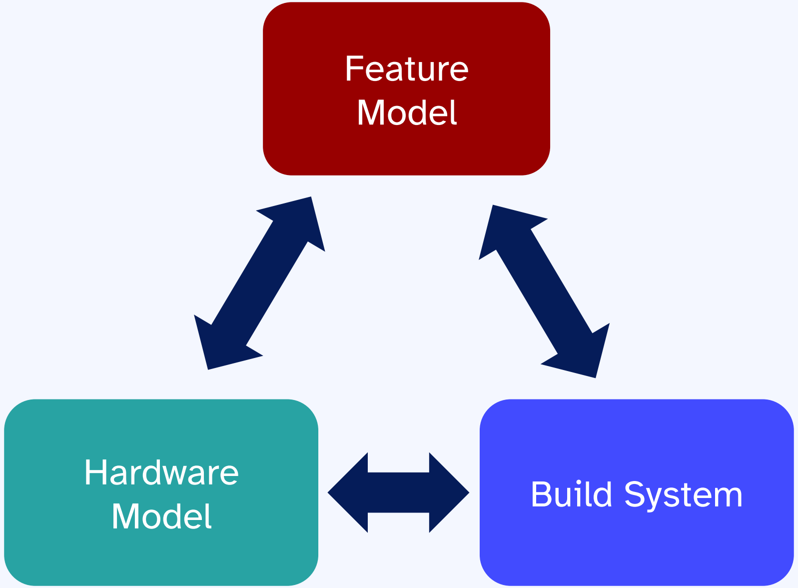
Models in Zephyr

3 domain-specific models at play

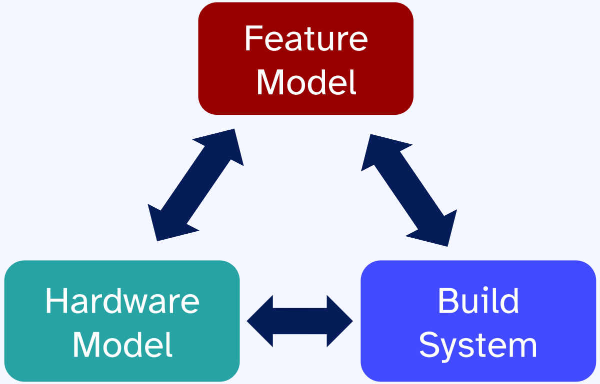
Feature Model: select desired functionality

Hardware Model: to describe hardware properties

Build System: to describe build process



Models in Zephyr

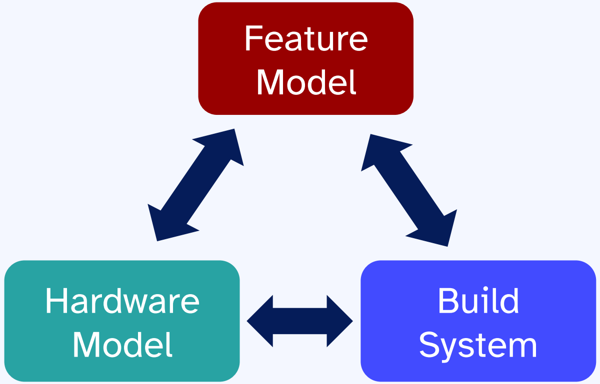


3 domain-specific models at play

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

- invoke CMake build system
- select hardware & features: <BOARD>.dts, <BOARD>_defconfig
- select application & features: prj.conf

Models in Zephyr



3 domain-specific models at play

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

invoke CMake build system

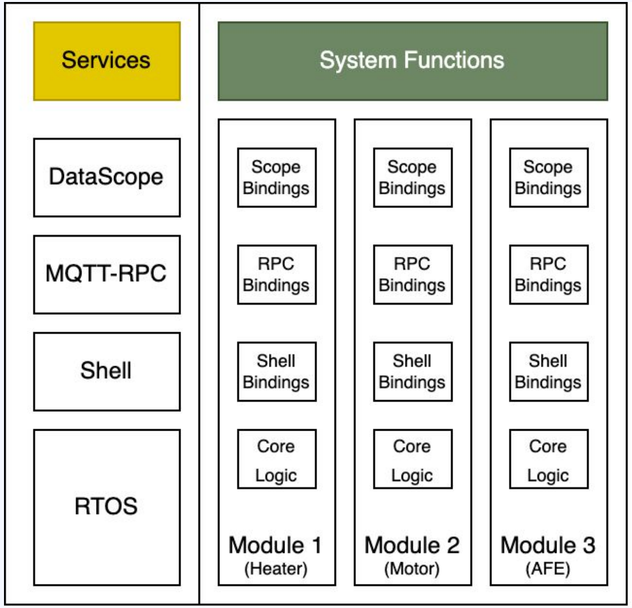
select hardware & features
<BOARD>.dts
<BOARD>_defconfig

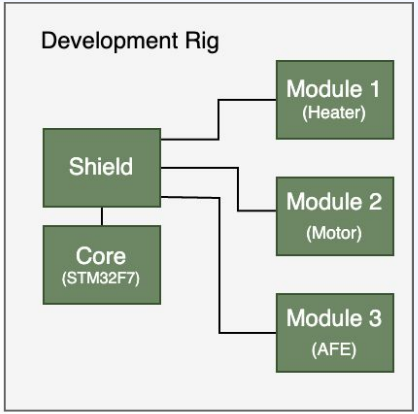
select application & features
prj.conf

models are coupled and interact with each other !!!

inovex

Introducing ACME-NG





A couple of years ago …

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

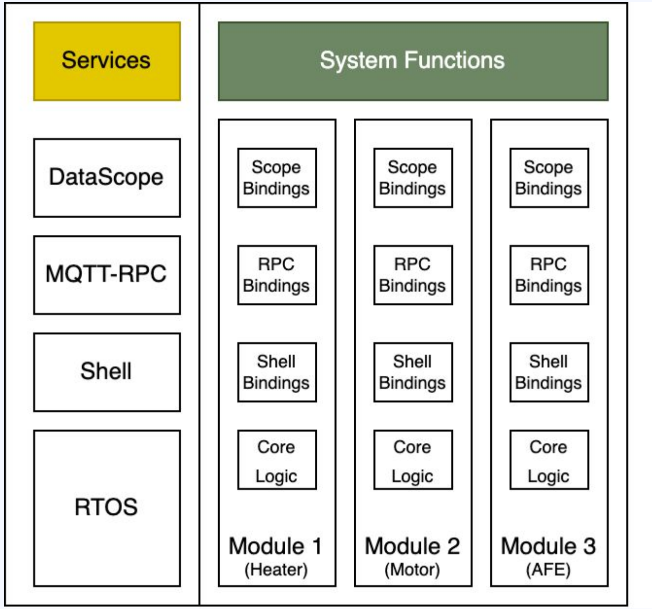
Iterative system design - basic system functions known but specific details dependent on reagent chemistry developed simultaneously

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

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

inovex

Modeling Software Features w/ Kconfig




```
menu "ACME Subsystems"

 menu "Modules"
  rsource "m_heater/Kconfig"
  rsource "m_motor/Kconfig"
  rsource "m_afe/Kconfig"
 endmenu #Modules

 menu "Core Services"
  rsource "s_mqtt_rpc"
  rsource "s_datascope"
 endmenu #Core Services
endmenu #ACME Subsystems
```

System Model
(Architecture)

Software Model
(Architecture, Design)

Modeling Software Features w/ Kconfig

```
menu "ACME Subsystems"

  menu "Modules"
    rsource "m_heater/Kconfig"
    rsource "m_motor/Kconfig"
    rsource "m_afe/Kconfig"
  endmenu #Modules

  menu "Core Services"
    rsource "s_mqtt_rpc"
    rsource "s_datascope"
  endmenu #Core Services
endmenu #ACME Subsystems
```

```
menuconfig ACME_SUBSYS_HEATER # option to toggle the entire subsystem on/off
    bool "Heater subsystem"
    help
        The Heater subsystem is responsible for measuring and controlling
        the temperature.

if ACME_SUBSYS_HEATER

    config ACME_SUBSYS_HEATER_THREAD_STACK_SIZE
        int "Stack size of subsystem thread"
        default 2048

    config ACME_SUBSYS_HEATER_MQTT_RPC
        bool "Enable MQTT-RPC bindings for $(subsys-str) subsystem"
        depends on ACME_MQTT_RPC

    config ACME_SUBSYS_HEATER_SHELL
        bool "Enable shell bindings for $(subsys-str) subsystem"
        depends on SHELL

    config ACME_SUBSYS_HEATER_SCOPE
        bool "Enable data scope bindings for $(subsys-str) subsystem"
        depends on ACME_SCOPE #only selectable if datascope is enabled

endif
```

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

```
config FEATURE_A
  bool "Try me out"
```

acme/KConfig

```
CONFIG_FEATURE_A=y
```

.config

```
#define CONFIG_FEATURE_A
```

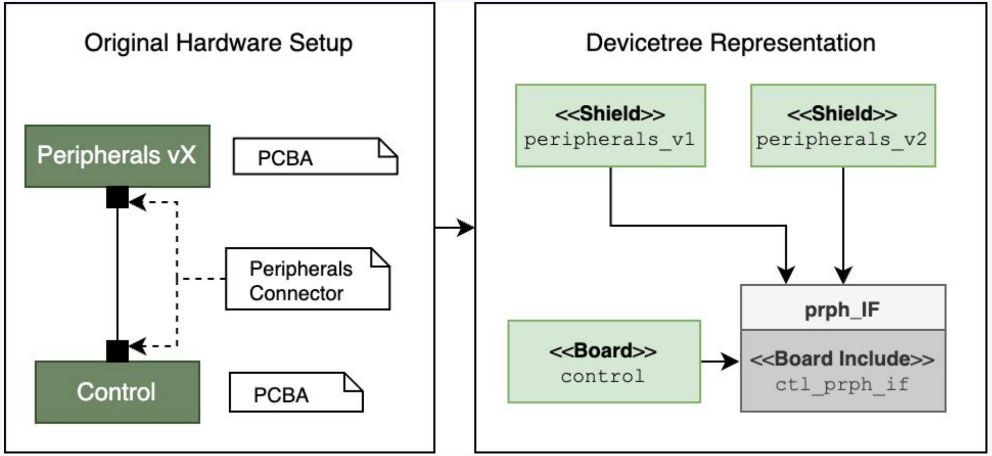
include/generated/autoconf.h

inovex

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 interface aka interconnects



inovex

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";
		…
		};
}
```

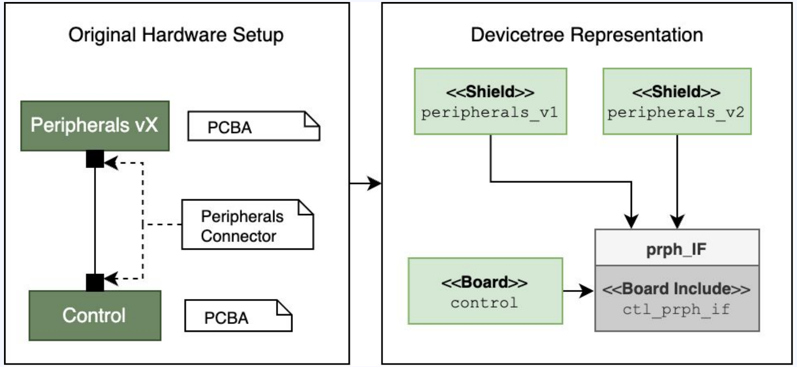
scripts/dts/gen_defines.py

```
#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);
…
```

The power of models

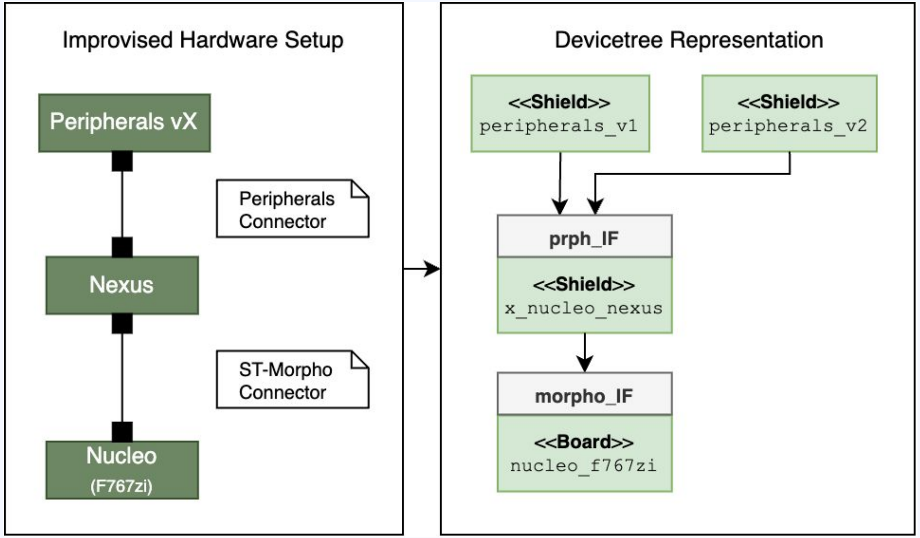
When supply chains fall apart …



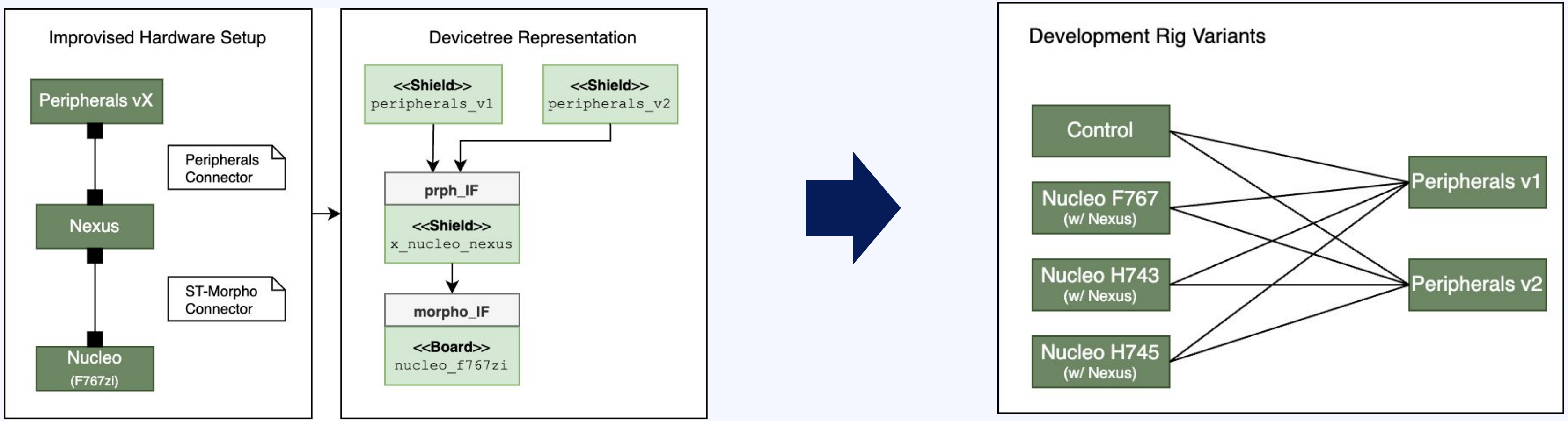
devicetree models allowed us to **compensate for all HW changes** without touching a single line of source code

When ACME needed parts to make hardware the most, the parts had disappeared …

… and all we could do, was to by existing devkit boards

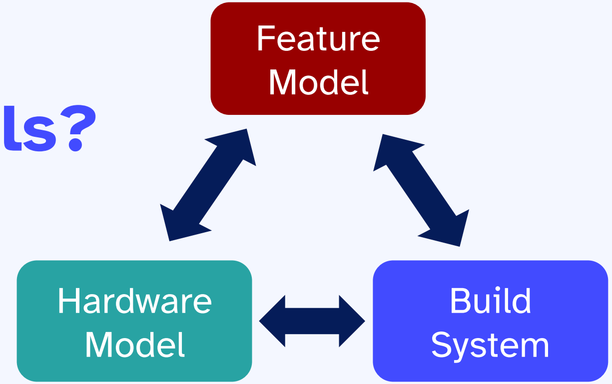


When supply chains fall apart …




```
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
```

What makes us go that fast w/ models?



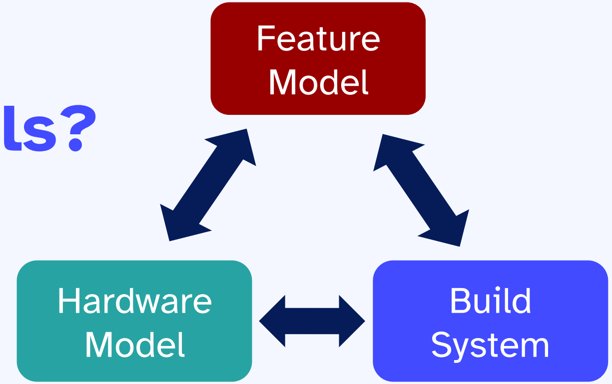
Appropriate: Devicetree, Kconfig and CMake established and mature

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 that fast w/ models?



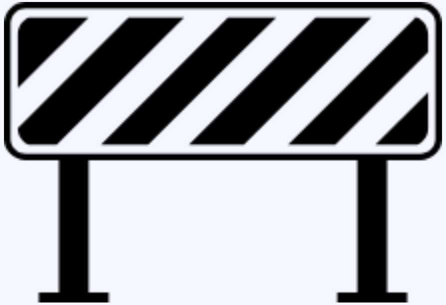
Integrable: Models can interact with each other to further increase usefulness

Extensible: Model languages can be extended with new constructs

Open: Underlying technologies open-source, no limitations to use or future development

inovex

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

inovex

Conclusion

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

inovex

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

inovex