TECHNICAL SESSION / ZEPHYR RTOS

Device Tree Beyond Hardware in Zephyr RTOS

Saravanan Sekar



A position from real work, and its boundaries

Linumiz
open-source powered

WHOAMI

Saravanan Sekar

Embedded Software Engineer at Linumiz

- Embedded Linux and Zephyr RTOS development, consulting, training
- Embedded Linux: BSP, U-Boot, Linux kernel, Yocto Project, Buildroot
- Zephyr: SoC and board support, drivers

Linumiz
open-source powered

SESSION MAP

Where we are going

01 **One description, resolved at build time**

The RTOS turn: compiled in, never parsed

02 **More than hardware**

Device tree as a typed build-time data model

03 **Presence vs structure**

Where Kconfig ends and device tree begins

04 **Where config belongs**

The boundary: static in, field-mutable out

05 **Already upstream**

binman, /config, sysbuild - in production

Linumiz
open-source powered

DEVICE TREE

A device tree, one node

Device Tree is a structured description of a system’s hardware configuration, used by software to discover and initialize hardware devices.

MINIMAL .dts

```
i2c1: i2c@40005400 {
    compatible = "vendor,i2c";
    reg = <0x40005400 0x400>;
    clock-frequency = <400000>;
    sensor@1d {
        compatible = "vendor,accel";
        reg = <0x1d>;
    };
};
```

MAPS TO HARDWARE

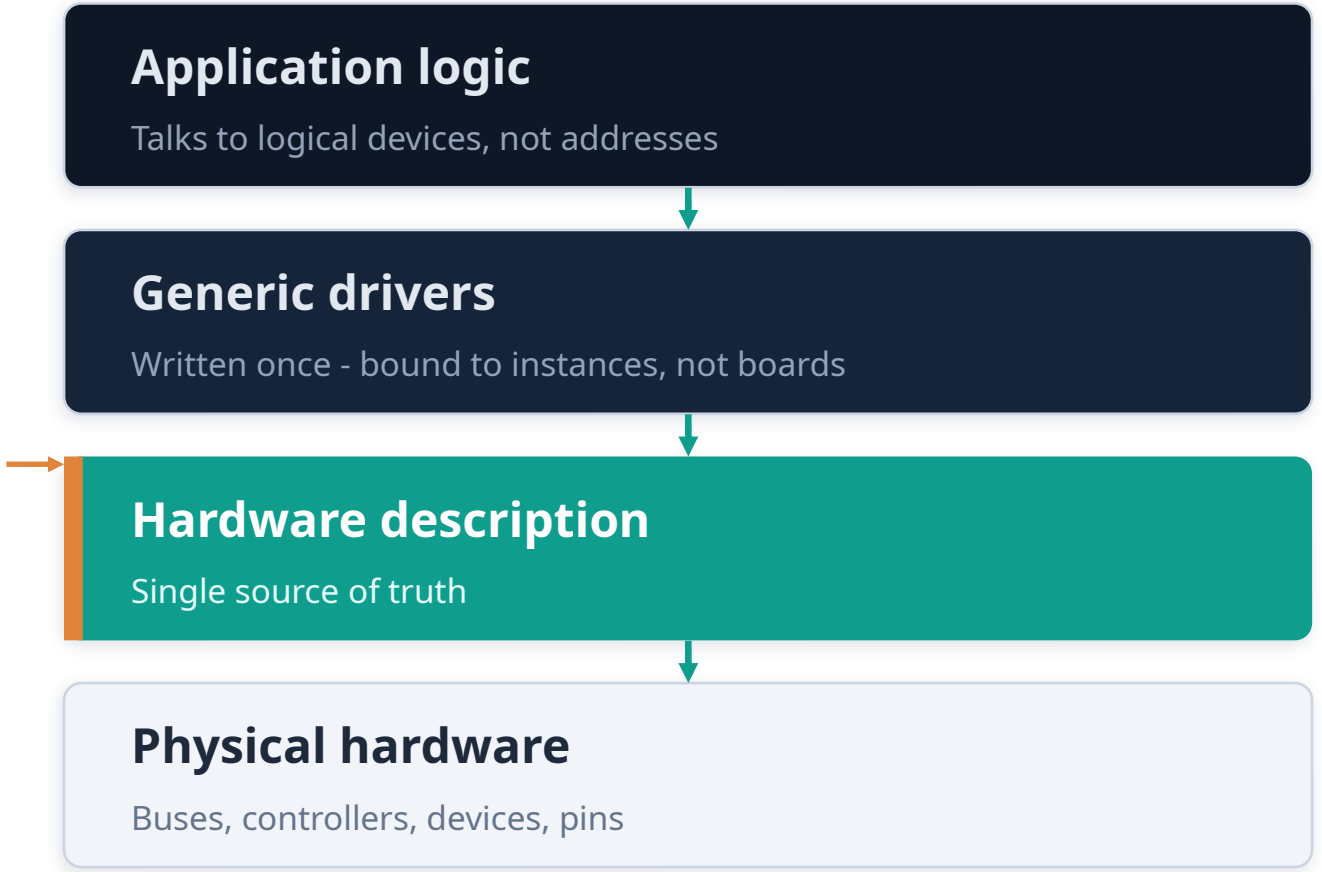


Linumiz
open-source powered

THE SINGLE MOVE

Insert one clean layer of truth

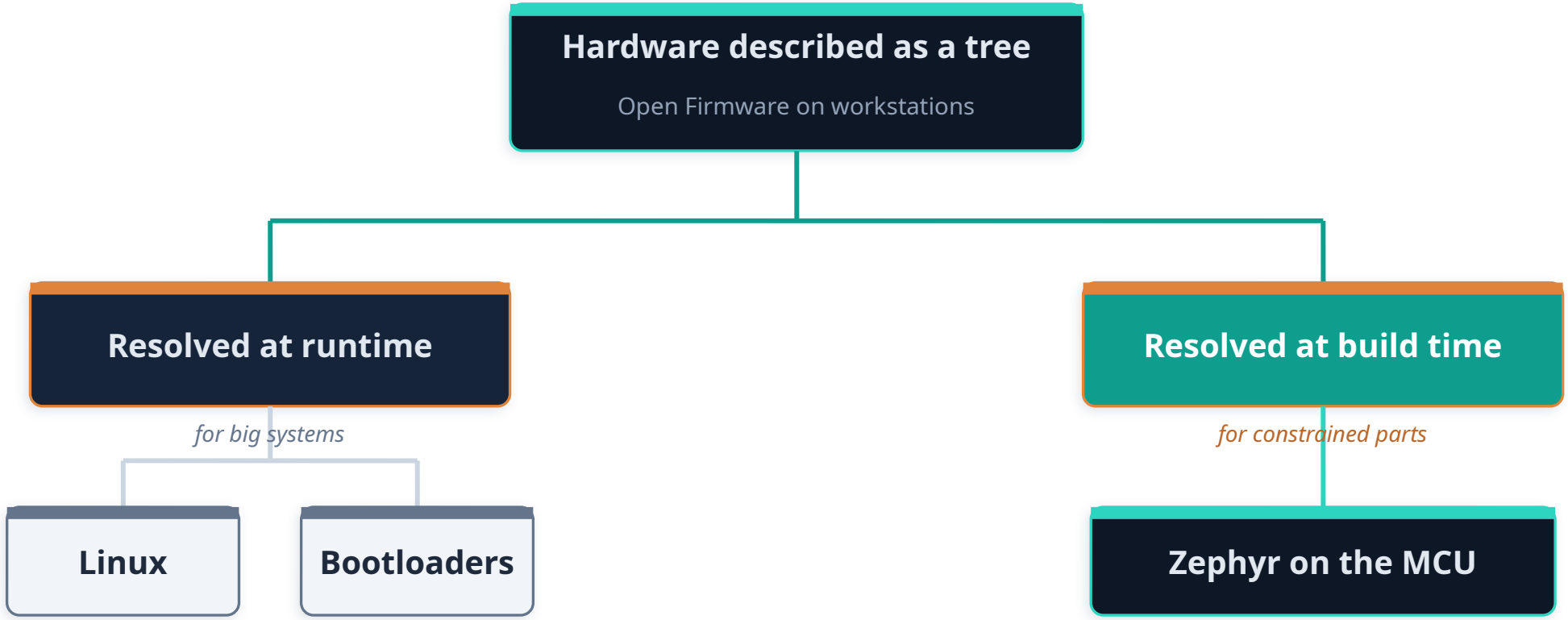
Single source of truth



Linumiz
open-source powered

ONE LINEAGE

The same idea, inherited down the stack



Not a Zephyr invention - an inheritance. The same descriptive model flows down the stack; only the moment it is resolved changes.

Linumiz
open-source powered

THE MCU ADAPTATION

Same description - resolved at a different moment

LINUX / U-BOOT

Resolved at runtime

- Description ships as a blob, separate from the image
- Kernel parses the tree while booting
- One image serves many boards

Suitable for big systems

ZEPHYR

Resolved at build time

- Description is compiled into the image itself
- Nothing is parsed on the device - ever
- Zero runtime cost, fully deterministic boot

Suitable for the microcontroller

Linumiz
open-source powered

THE ACTUAL FILES

From device tree to a generated header

INPUTS

`.dts`	board sources
`.dtsi`	SoC / shared includes
`.overlay`	per-app / per-board tweaks
`.yaml`	bindings — the rules

TOOLS

`scripts/dts/`

`gen_defines.py`
emits the C macros

`edtlib.py`
applies the bindings

`dtlib.py`
low-level DTS parser

`dtc`
sanity check only

OUTPUTS - in <build>/zephyr/

`zephyr.dts.pre`
preprocessed source (intermediate)

`devicetree_generated.h`
the generated macros

`zephyr.dts`
final merged tree (for debugging)

The rule: code never includes `devicetree_generated.h` directly - it goes through **`devicetree.h`**, the stable API over the generated macros.

Linumiz
open-source powered

THE REFRAME

03

More than hardware

The device-tree machinery is a typed, build-time data model. Nothing in it is specific to hardware.





Linumiz

A DATA MODEL

Typed, validated, resolved at build time

What device tree is

Typed nodes and properties, hierarchy, cross-references by phandle - checked against schemas, frozen into constants at build.

Why that matters here

None of that machinery is specific to hardware. The same model can carry any structured, build-time-known configuration.



The reframe

Not: a hardware format that happens to be data.

But: a data model whose first use was hardware.

Hardware description is one use of device tree, not its definition.

Linumiz
open-source powered

THE AXIS

04

Presence vs structure

Where Kconfig ends and device tree begins.





Linumiz

THE BOTTOM LINE

System Level config: scattered, or declared?

Scattered today

- Split across Kconfig, overlays, and C
- Instances hand-indexed, invalidated
- A typo is a silent wrong value
- No single place to read the config
- Duplicated across boards

Declared in the tree

- One typed, validated place for structured config
- Repeating instances as nodes, not #ifdef
- Per-board override by overlay
- Cross-references by phandle, not manual indices
- Reviewable data, compiled in at zero cost
- Field-mutable values stay in settings, by design

Linumiz
open-source powered

THE BOUNDARY

05

Extend DTS for system configs

Not everything structured belongs in the tree. The line is static versus field-mutable.





Linumiz

THE BOUNDARY

Static in, field-mutable out

BELONGS ELSEWHERE

Field-mutable, per-unit

- IP, gateway, routing - per deployment
- Credentials and keys
- User-tunable settings
- Goes to settings / NVS, not the tree

Changes in the field

BELONGS IN DEVICE TREE

Static, product-fixed

- CAN FD virtual channels - count and roles
- Channel and interface maps
- Calibration tables
- Fixed peripheral parameters

Fixed by the product

Linumiz
open-source powered

A CONCRETE CASE

VLANs today: all in app defconfig

prj.conf TODAY

Flat Kconfig scalars

- CONFIG_NET_VLAN=y
- CONFIG_NET_VLAN_COUNT=2
- ..IFACE2_VLAN_TAG=20
- ..IFACE3_VLAN_TAG=40

Per-interface, hand-indexed

AS STRUCTURED CONFIG

One node per VLAN

```
zephyr,user {
    eth0 {
        vlan20: vlan20 {
            tag = <20>;
            ipaddr = <10.10.20.1>
        };
        vlan40: vlan40 {
            tag = <40>;
            ipaddr = <10.10.40.1>
        };
    };
};
```

Structure in tree, IP in settings

Linumiz
open-source powered

FROM REAL WORK

Starting with /zephyr,user - and where it bit

What we tried

- Needed structured, repeating config - several channels
- Used zephyr,user with one child node per instance
- Walked them with DT_FOREACH_CHILD
- It compiled. It worked.

WHAT IT COST

No binding on the children, so no validation.

No compatible, no instance model, no type checking. A missing property fails at macro expansion, not with a clear error.

Lesson: structured config wants a real binding.
zephyr,user is the not-ready-yet layer, not the destination.

Linumiz
open-source powered

PRECEDENT

06

Already upstream

Not a proposal. Device tree already carries non-hardware config in production.





Linumiz

THREE PRECEDENTS

Three places it already happens

binman

U-Boot describes flash image layout - entries, offsets, sizes - as a device-tree node. Build-time, structured, not hardware. It targets Zephyr too.

+

/config

U-Boot reads runtime policy from a /config node. Some values override Kconfig. Non-hardware config, in the tree, for years.

+

sysbuild

U-Boot already builds multi-image layout from device tree. The principle is here - for infrastructure.

What is new: not the principle, but the consumer - application C code, not a build tool. Same idea, one step further.

Linumiz
open-source powered

Thank You

contact@linumiz.com

