Visualizing Device Trees

Daniel Maslowski

Agenda

- Device Tree History
- Design of dtvis
- Rust and WebAssembly

Thanks for a start!

The development of `dtvis` has had strong support from our hackerspace.

Device Tree History

… so what is a Device Tree?

IEEE 1275[1] Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices / Open Firmware[2]

[1] https://standards.ieee.org/ieee/1275/1932/
[2] https://www.openfirmware.info/data/docs/of1275.pdf

… so what is a Device Tree?

- IEEE 1275[1] Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices / Open Firmware[2]
- describing hardware topology for non-discoverable devices

[1] https://standards.ieee.org/ieee/1275/1932/
[2] https://www.openfirmware.info/data/docs/of1275.pdf

… so what is a Device Tree?

- IEEE 1275[1] Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices / Open Firmware[2]
- describing hardware topology for non-discoverable devices
- Linux, U-Boot, Zephyr, OLPC, FreeBSD and other projects use it
  - including Apple back in the days

[1] https://standards.ieee.org/ieee/1275/1932/
[2] https://www.openfirmware.info/data/docs/of1275.pdf

… so what is a Device Tree?

- IEEE 1275[1] Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices / Open Firmware[2]
- describing hardware topology for non-discoverable devices
- Linux, U-Boot, Zephyr, OLPC, FreeBSD and other projects use it
  - including Apple back in the days
- not too elegant, attached to kernel via bindings

[1] https://standards.ieee.org/ieee/1275/1932/
[2] https://www.openfirmware.info/data/docs/of1275.pdf

… so what is a Device Tree?

- IEEE 1275[1] Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices / Open Firmware[2]
- describing hardware topology for non-discoverable devices
- Linux, U-Boot, Zephyr, OLPC, FreeBSD and other projects use it
  - including Apple back in the days
- not too elegant, attached to kernel via bindings
- only few specified fields, most are “as someone wrote them”
  - compare e.g. Amlogic vs Allwinner SoC based trees

[1] https://standards.ieee.org/ieee/1275/1932/
[2] https://www.openfirmware.info/data/docs/of1275.pdf

… so what is a Device Tree?

- IEEE 1275[1] Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices / Open Firmware[2]
- describing hardware topology for non-discoverable devices
- Linux, U-Boot, Zephyr, OLPC, FreeBSD and other projects use it
  - including Apple back in the days
- not too elegant, attached to kernel via bindings
- only few specified fields, most are “as someone wrote them”
  - compare e.g. Amlogic vs Allwinner SoC based trees
- can range from a few hundred to a thousand nodes

[1] https://standards.ieee.org/ieee/1275/1932/
[2] https://www.openfirmware.info/data/docs/of1275.pdf

… so what is a Device Tree?

- IEEE 1275[1] Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices / Open Firmware[2]
- describing hardware topology for non-discoverable devices
- Linux, U-Boot, Zephyr, OLPC, FreeBSD and other projects use it
  - including Apple back in the days
- not too elegant, attached to kernel via bindings
- only few specified fields, most are “as someone wrote them”
  - compare e.g. Amlogic vs Allwinner SoC based trees
- can range from a few hundred to a thousand nodes
- the tree is a lie; there are cycles, e.g., power supplies and clocks

[1] https://standards.ieee.org/ieee/1275/1932/
[2] https://www.openfirmware.info/data/docs/of1275.pdf

Previous work

There were discussions on tooling at Linux Plumbers[3], partially stalled.

[3] https://elinux.org/images/8/83/Plumbers_2016_dt_device_tree_tools.pdf

Previous work

There were discussions on tooling at Linux Plumbers[3], partially stalled.

- Component Inspector (by Freescale, now NXP)
  - proprietary, closed source Eclipse plugin
  - was part of QorIQ Configuration Suite, no longer available

[3] https://elinux.org/images/8/83/Plumbers_2016_dt_device_tree_tools.pdf

Previous work

There were discussions on tooling at Linux Plumbers[3], partially stalled.

- Component Inspector (by Freescale, now NXP)
  - proprietary, closed source Eclipse plugin
  - was part of QorIQ Configuration Suite, no longer available
- https://github.com/dev-0x7C6/fdt-viewer
  - mixed tree + hex/text viewer, C++ + Qt
  - supports dtb, dtbo (overlay) and itb (FIT image)

[3] https://elinux.org/images/8/83/Plumbers_2016_dt_device_tree_tools.pdf
[4] https://www.spinics.net/lists/devicetree-spec/msg00950.html

Previous work

There were discussions on tooling at Linux Plumbers[3], partially stalled.

- Component Inspector (by Freescale, now NXP)
  - proprietary, closed source Eclipse plugin
  - was part of QorIQ Configuration Suite, no longer available
- https://github.com/dev-0x7C6/fdt-viewer
  - mixed tree + hex/text viewer, C++ + Qt
  - supports dtb, dtbo (overlay) and itb (FIT image)
- https://github.com/bmx666/dtv-demo
  - “RFC - DTV (Device Tree Visualiser)” on mailing list[4]
  - dt_s_ only, more of a text editor, Python + Qt6

[3] https://elinux.org/images/8/83/Plumbers_2016_dt_device_tree_tools.pdf
[4] https://www.spinics.net/lists/devicetree-spec/msg00950.html
[5] https://marketplace.visualstudio.com/items?itemName=plorefice.devicetree

Previous work

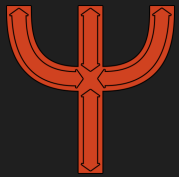
There were discussions on tooling at Linux Plumbers[3], partially stalled.

- Component Inspector (by Freescale, now NXP)
  - proprietary, closed source Eclipse plugin
  - was part of QorIQ Configuration Suite, no longer available
- https://github.com/dev-0x7C6/fdt-viewer
  - mixed tree + hex/text viewer, C++ + Qt
  - supports dtb, dtbo (overlay) and itb (FIT image)
- https://github.com/bmx666/dtv-demo
  - “RFC - DTV (Device Tree Visualiser)” on mailing list[4]
  - dt_s_ only, more of a text editor, Python + Qt6
- VS Code plugin `plorefice.devicetree`[5]
  - syntax highlighting + collapsing
  - could be enhanced with *dtvis* :-)

[3] https://elinux.org/images/8/83/Plumbers_2016_dt_device_tree_tools.pdf
[4] https://www.spinics.net/lists/devicetree-spec/msg00950.html
[5] https://marketplace.visualstudio.com/items?itemName=plorefice.devicetree

Design of dtvis

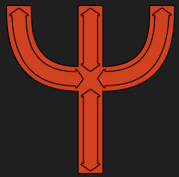
Recap: Platform System Interface



The *Platform System Interface* project (PSI) is a collection of design ideas, specifications, tools and other resources all around hardware platforms, firmware, bootloaders, OS interfacing and user experience.

https://github.com/platform-system-interface

Recap: Platform System Interface



The *Platform System Interface* project (PSI) is a collection of design ideas, specifications, tools and other resources all around hardware platforms, firmware, bootloaders, OS interfacing and user experience.

https://github.com/platform-system-interface

Talk: Platform System Interface - Design and Evaluation of Computing as a Whole

in-depth discussion of design paradigms and complexity in computing
https://metaspora.org/platform-system-interface-computing-as-whole.pdf

What is dtvis?

[6] https://devicetree.org

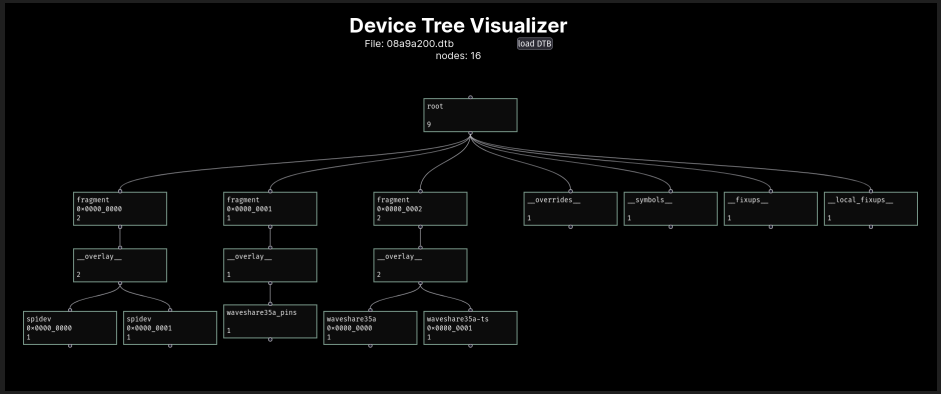
What is dtvis?

dtvis is a DeviceTree[6] visualizer.

[6] https://devicetree.org

What is dtvis?

dtvis is a DeviceTree[6] visualizer.



https://github.com/platform-system-interface/dtvis

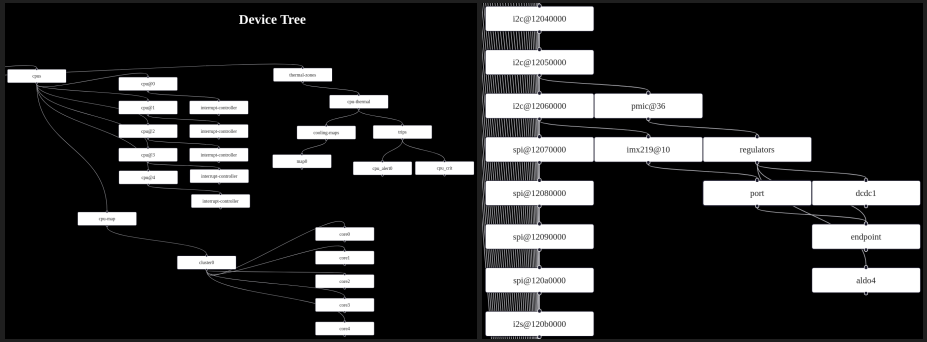
[6] https://devicetree.org

Challenges when Drawing Trees

- How do we distribute nodes without overlaps?
- How do we make “good” use of space?
- Nodes can have few to dozens of properties.

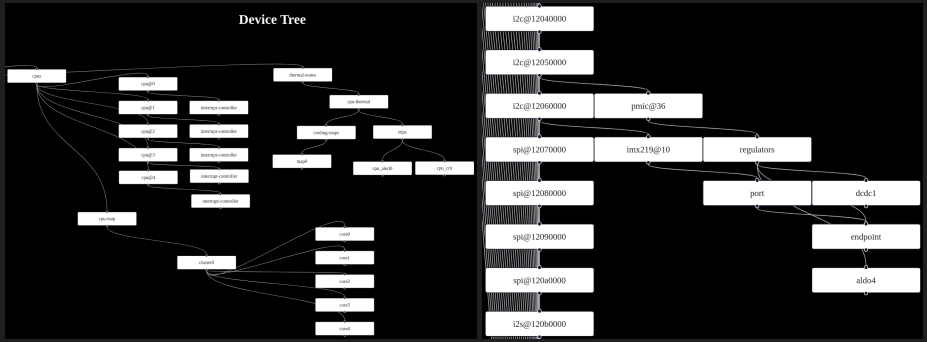
Challenges when Drawing Trees

- How do we distribute nodes without overlaps?
- How do we make “good” use of space?
- Nodes can have few to dozens of properties.



Challenges when Drawing Trees

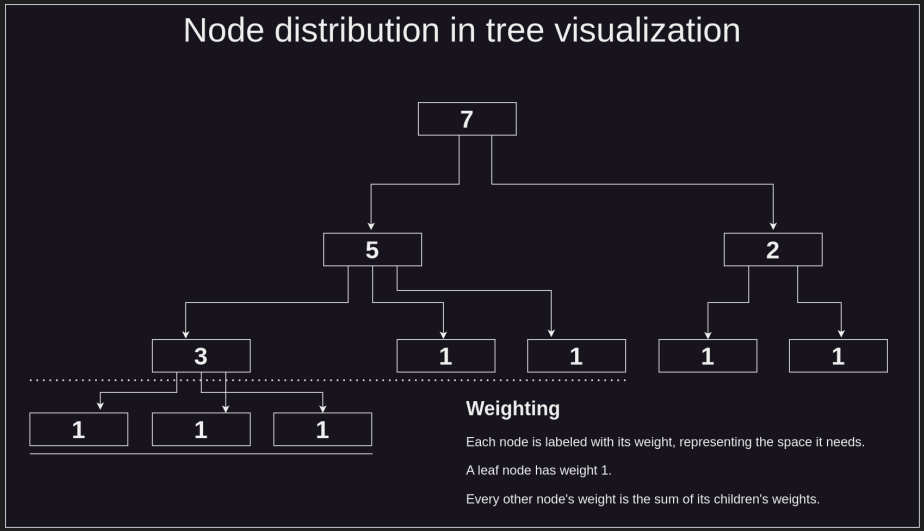
- How do we distribute nodes without overlaps?
- How do we make “good” use of space?
- Nodes can have few to dozens of properties.



Strategies

- Even distribution
- Collapsing

dtvis Design



Currently, we have no overlaps, but trees may grow very wide.

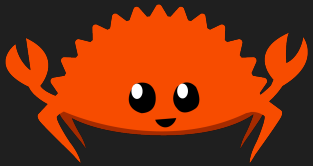
DEMO

Rust and WebAssembly

What if…

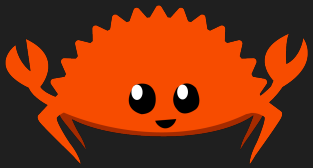
What if…

… we compile Rust…



What if…

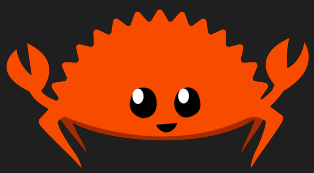
… we compile Rust…



… to Wasm…

What if…

… we compile Rust…



… to Wasm…

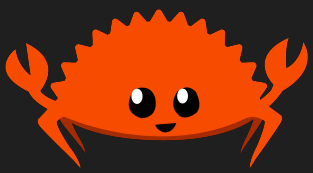
… and use it in an app?

What if…

… we compile Rust…

… to Wasm…

… and use it in an app?



Magic happens - we can use native code on web platforms!

Howto

Howto

Getting started

https://lannonbr.com/blog/2020-01-07-rust-wasmpack/
https://rustwasm.github.io/docs/wasm-pack/

Howto

Getting started

https://lannonbr.com/blog/2020-01-07-rust-wasmpack/
https://rustwasm.github.io/docs/wasm-pack/

TL;DR

```
cargo install wasm-pack
wasm-pack new my-rust-wasm-foo
```

Howto

Getting started

https://lannonbr.com/blog/2020-01-07-rust-wasmpack/
https://rustwasm.github.io/docs/wasm-pack/

TL;DR

```
cargo install wasm-pack
wasm-pack new my-rust-wasm-foo
```

The glue

https://github.com/wasm-tool/wasm-pack-plugin
https://rustwasm.github.io/docs/wasm-pack/tutorials/hybrid-applications-with-webpack/using-your-library.html

Howto

Getting started

https://lannonbr.com/blog/2020-01-07-rust-wasmpack/
https://rustwasm.github.io/docs/wasm-pack/

TL;DR

```
cargo install wasm-pack
wasm-pack new my-rust-wasm-foo
```

The glue

https://github.com/wasm-tool/wasm-pack-plugin
https://rustwasm.github.io/docs/wasm-pack/tutorials/hybrid-applications-with-webpack/using-your-library.html

More glue

```
cargo add gloo-utils
```

The Rust side

The Rust side

```rust
extern crate wasm_bindgen;
use gloo_utils::format::JsValueSerdeExt;
use serde::{Deserialize, Serialize};
use wasm_bindgen::prelude::*;

/// ...

#[derive(Serialize, Deserialize)]
struct Foo {
    bar: u32,
    baz: String,
}

#[wasm_bindgen]
pub fn some_fun(data: JsValue) -> JsValue {
    /// ...
    let foo = Foo::new { bar: 42, baz: "Rust Wasm" };
    JsValue::from_serde(&foo).unwrap()
}
```

The JavaScript side

```
import { some_fun } from "./rs/pkg";

/* ... */
    const res = some_fun({ woopWoop: 1337 });
    console.info(res);
/* ... */
```

The JavaScript side

```
import { some_fun } from "./rs/pkg";

/* ... */
    const res = some_fun({ woopWoop: 1337 });
    console.info(res);
/* ... */
```

But that is synchronous and blocking!

The JavaScript side

```
import { some_fun } from "./rs/pkg";

/* ... */
    const res = some_fun({ woopWoop: 1337 });
    console.info(res);
/* ... */
```

But that is synchronous and blocking!

https://rustwasm.github.io/wasm-bindgen/reference/js-promises-and-rust-futures.html

https://rustwasm.github.io/wasm-bindgen/api/wasm_bindgen_futures/

Thanks! :)

Follow Me

Daniel Maslowski

https://github.com/orangecms
https://twitter.com/orangecms
https://mastodon.social/@cyrevolt
https://twitch.tv/cyrevolt
https://youtube.com/@cyrevolt

https://github.com/platform-system-interface

https://metaspora.org/visualizing-device-trees.pdf