Qualcomm

Open Source Summit India 2026

DTS 101 From Roots to Trees, aka Devicetree for Beginners

Krzysztof Kozlowski

Senior Staff Engineer
Qualcomm Wireless GmbH

krzk@kernel.org, @krzk@social.kernel.org

Introduction

- Krzysztof Kozlowski
- I work for Qualcomm

Introduction

- Krzysztof Kozlowski
- I work for Qualcomm
- I am a co-maintainer of a few subsystems in the Linux kernel
  - SoC subsystem (formerly arm-soc)
  - Devicetree bindings
  - Memory controller drivers
  - Samsung Exynos SoC ARM/ARM64 architecture

Introduction

- Krzysztof Kozlowski
- I work for Qualcomm
- I am a co-maintainer of a few subsystems in the Linux kernel
  - SoC subsystem (formerly arm-soc)
  - Devicetree bindings
  - Memory controller drivers
  - Samsung Exynos SoC ARM/ARM64 architecture
- I contributed a lot to the Linux kernel, although now I mostly read the code
  - git shortlog -s -n --no-merges | head -n 5
  - Basically, since v3.11, I have contributed to every Linux kernel release

Caveats

- Through the presentation some parts will be **marked with bold and color font**. These represent common issues observed in DTS and things to watch out.
- For simplification I will use only “Linux kernel” term, but this should be understood as any upstream OS or software using the kernel’s Devicetree
  - E.g. OpenBSD, U-Boot
- Devicetree bindings are not limited to the Linux kernel and other upstream users should be considered as well
  - However the talk will not cover non-Linux kernel DTS style / rules
- The talk is considered introductory, thus obviously incomplete
  - For more rules please refer to writing-bindings.rst, writing-schema.rst and DT submitting-patches.rst in the Linux kernel sources
- Most of the code examples use 4-space indentation for slides readability
  - This is not the Linux nor DTS kernel coding style

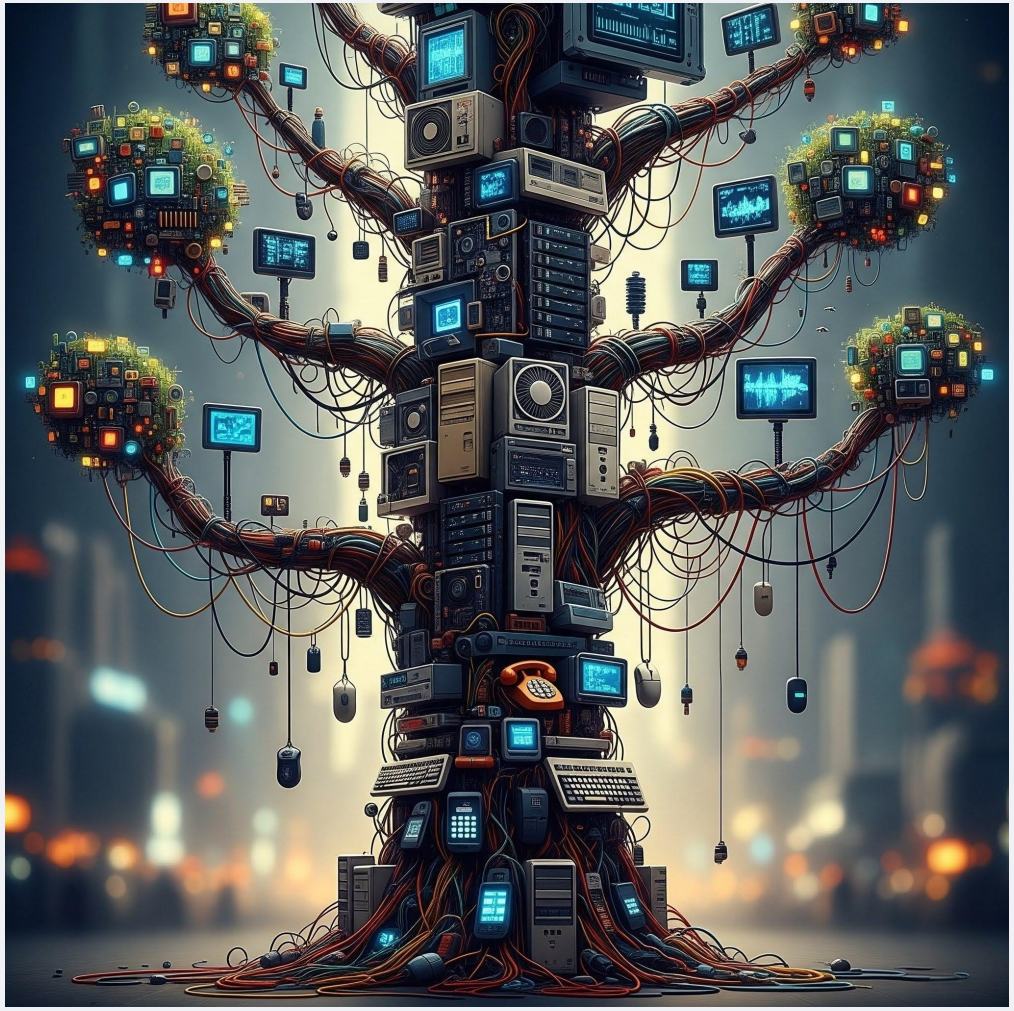
Agenda

- What is the Devicetree?
- How does a DTS look like?
- What does compatibility between devices mean and how to express it?
- How does a DT binding look like?
- What can be in DTS and what cannot
- Validate your DTS and live error-free ever after

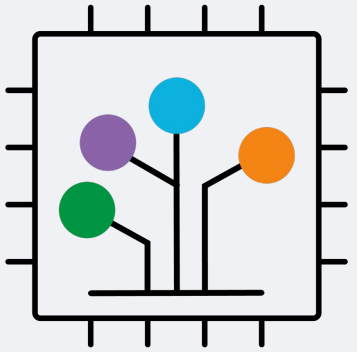
What Is the Devicetree?

What Is the Devicetree (DT)?

- Tree of devices?



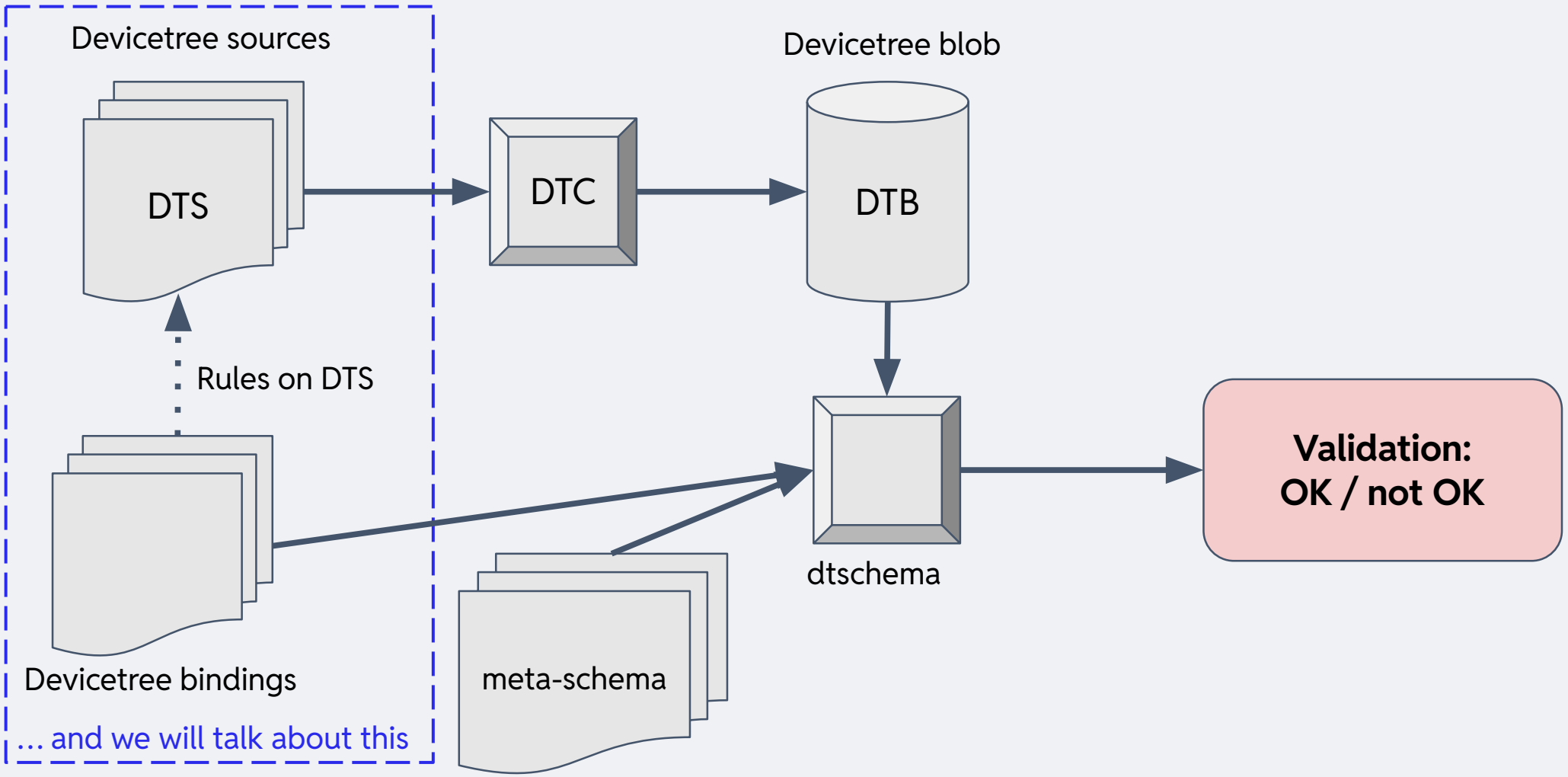
What Is the Devicetree (DT)?



Source: https://devicetree-specification.readthedocs.io
(trademark and service mark licensed by Linaro Ltd.)

- Data format to describe the non-discoverable hardware
- Used by the software, like Linux kernel or U-boot, to understand the hardware it is running on
  - ACPI-based systems use… ACPI tables for that!
- DTS - Devicetree Sources are textual representation of Devicetree, to be consumed by the Devicetree compiler (DTC)
- DTB - Devicetree Blob is a compact binary representation of the DTS
- Devicetree bindings - rules how DTS should be constructed and documenting the ABI between software and DTS
  - … and the toolset
  - … and a project: https://github.com/devicetree-org/dt-schema/

What Is the Devicetree (DT)?



How Does a DTS Look Like?

Device Nodes

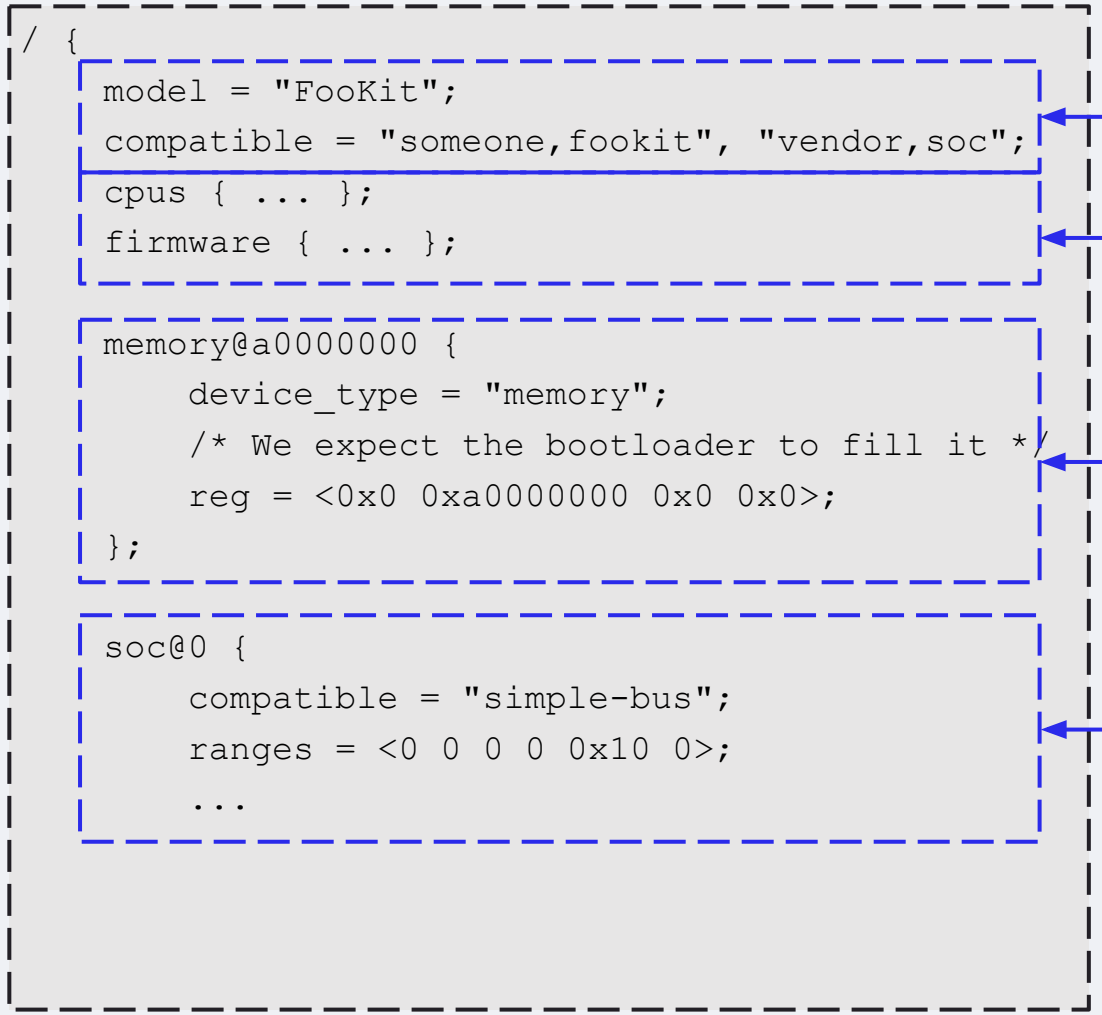
- As a tree, it starts with root
- Each node is a called “device node”
  - Usually should correspond to a “device”

How Does a DTS Look Like?

```
/ {
    model = "FooKit";
    compatible = "someone,fookit", "vendor,soc";
    cpus { ... };
    firmware { ... };

    memory@a0000000 {
        device_type = "memory";
        /* We expect the bootloader to fill it */
        reg = <0x0 0xa0000000 0x0 0x0>;
    };

    soc@0 {
        compatible = "simple-bus";
        ranges = <0 0 0 0 0x10 0>;
        ...
```

```
    soc@0 {
        compatible = "simple-bus";
        ranges = <0 0 0 0 0x10 0>;
        ...

        clock-controller@100000 {
            compatible = "vendor,soc-clock-ctrl";
            reg = <0x0 0x00100000 0x0 0x1f4200>;
            #clock-cells = <1>;
        };

        spi@880000 {
            compatible = "vendor,soc-spi-ctrl";
            reg = <0x0 0x00880000 0x0 0x4000>;
            ...
        };
    };
```

How Does a DTS Look Like?



- Root node with model and compatible describing the board (don't forget bindings for the "compatible"!)
- List of the CPUs
- Some services provided by firmware
- Total physical memory installed, often placeholder for whatever is provided by the bootloader
- I omitted here many more typical device nodes
- Node grouping all MMIO (memory mapped IO) nodes, usually representing the System on Chip
- By convention that's just a simple-bus
  - simple means: no resources, nothing to do here

How Does a DTS Look Like?

- The SoC node has many children representing particular devices
- These devices will have their own compatible and MMIO address
- They might have certain resources (more on that later) and more children

```
soc@0 {
    compatible = "simple-bus";
    ranges = <0 0 0 0 0x10 0>;
    ...

    clock-controller@100000 {
        compatible = "vendor,soc-clock-ctrl";
        reg = <0x0 0x00100000 0x0 0x1f4200>;
        #clock-cells = <1>;
    };

    spi@880000 {
        compatible = "vendor,soc-spi-ctrl";
        reg = <0x0 0x00880000 0x0 0x4000>;
        ...
    };
};
```

How Does a DTS Look Like?

- How an SPI controller could look like?

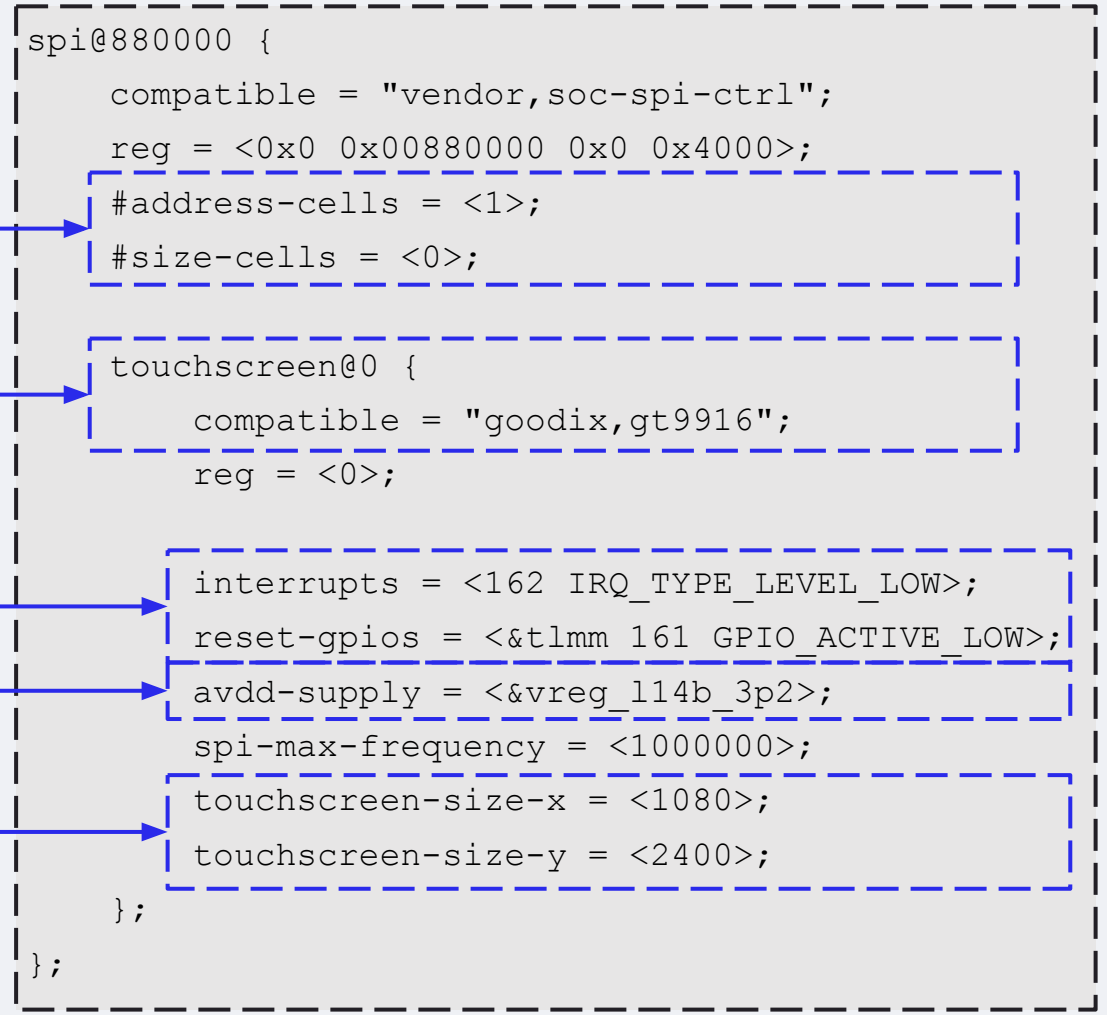
```
spi@880000 {
    compatible = "vendor,soc-spi-ctrl";
    reg = <0x0 0x00880000 0x0 0x4000>;
```

How Does a DTS Look Like?

- How an SPI controller could look like?
- Describes how the child nodes should be addressed
  - One u32 number for address
  - No size for the addresses, why?
  - Because that's SPI Chip Select!
- Device is a Goodix touchscreen
- It has an interrupt line, one reset GPIO
- It cannot operate without power - thus a supply

- And some more device-specific properties

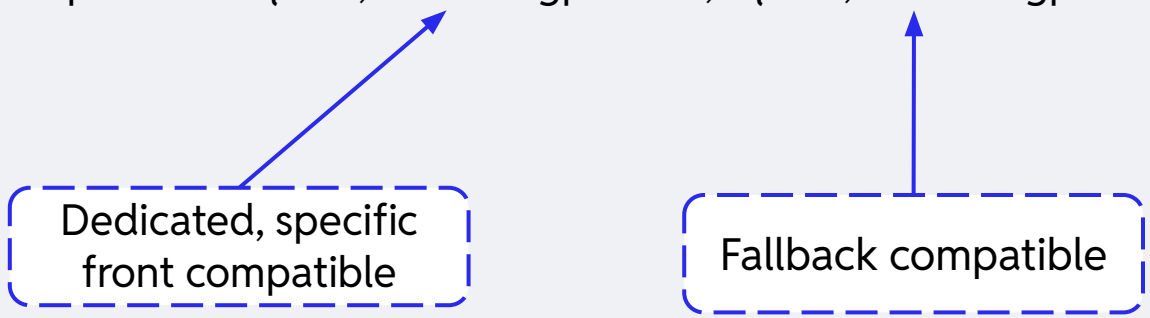
- **Unless you say otherwise with "status" property, each node is enabled by default**



Compatibility

Compatible Property

- Probably the most important property in DTS and bindings: compatible
- List of strings “vendor,device” and usually each string represents one device
  - **No wildcards, no family names**
  - Avoid versions (there are exceptions), but use actual device model name
  - Must be specific: applying to one specific hardware
    - A bit different meaning is for Linux-specific compatibles like syscon or simple-mfd
- It is a list, which means devices could be compatible with each other
  - For example on a Qualcomm SM8750 SoC the GPI DMA controller is:
    compatible = "qcom,sm8750-gpi-dma", "qcom,sm6350-gpi-dma";



Compatible Devices

- **Each device must have its own, specific compatible**
  - If you use fallback, you need the front compatible
  - If devices are identical, a dedicated front compatible is still necessary
- Compatible tells Linux which driver to bind with the device
- Two devices are compatible when the new device works with Linux drivers bound via fallback (old) compatible
  - Some new features might not be available in such case, that's fine
  - Dedicated, front compatible might not be ever used by Linux drivers, that's fine
- If for the new device the **fallback cannot be used by the Linux kernel, devices are most likely not compatible**

Why Do You Need a DT Binding?

Purpose of DTS

- DTS is created for a specific purpose:
  To be consumed by some software which needs to work with the hardware
- Our software - the Linux kernel - is generic purpose
  - We build one arm64 defconfig
  - Will be running on different arm64 boards from different vendors
  - The difference will be DTB (compiled DTS) matching hardware passed to the kernel
- How does the Linux know what to look for in DTS?
  - What device node properties to expect?
  - When a property is optional and when required?
- Are you sure your out of tree DTS will work fine?
  - Not every product ends with upstreamed DTS

Why Do You Need a DT Binding?

- There is a contract between the software (Linux kernel) and the DTS
- We define ABI (Application Binary Interface)
- Software and DTS both adhere to the ABI rules
- Here come Devicetree Bindings
  - Describe that contract, that ABI
  - Precise rules how DTS should be constructed so software can use it

How Does a DT Binding Look Like

How to Start with a DT Binding?

- Choose the example-schema.yaml or some recent, reviewed binding of similar device as a template
- Filename: matching the compatible or one of its fallbacks

```
# SPDX-License-Identifier: (GPL-2.0-only OR BSD-2-Clause)
%YAML 1.2
---
$id: http://devicetree.org/schemas/iio/adc/rohm,bd79104.yaml#
$schema: http://devicetree.org/meta-schemas/core.yaml#

# Short title describing the hardware (not drivers)
title: ROHM Semiconductor BD79104 ADC

# Person(s) caring about this particular hardware (not subsystem maintainers)
maintainers:
  - Matti Vaittinen <mazziesaccount@gmail.com>
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

How to Start with a DT Binding?

```
...
maintainers:
  - Matti Vaittinen <mazziesaccount@gmail.com>

description:
  Say something more about hardware. Simple devices as such SPI ADC could have
  just one sentence. More complex hardware, some groups of devices or
  anything unusual usually deserves more explanation. Remember about wrapping
  your lines at current coding style (at 80, even though the example here
  for the purpose of presentation does not fit).

properties:
  # Compatible should be always the first property, see DTS coding style document.
  #
  # Simple cases - one device:
  compatible:
    const: rohm,bd79104
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

How to Start with a DT Binding?

```
...

properties:
  # Or use enumeration of few incompatible devices, but similar enough to be
  # put in the same binding. Such enumerations or lists are usually ordered
  # alphanumerically:
  compatible:
    enum:
      - rohm,bd79104
      - rohm,bd79115
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

How to Start with a DT Binding?

```
...

properties:
  # Or express compatibility between devices with fallbacks:
  compatible:
    oneOf:
      - const: rohm,bd79104

      # "items" means a list:
      - items:
          # We expect more devices to be compatible with bd79104, thus enumeration:
          - enum:
              - rohm,bd79115
          - const: rohm,bd79104

  # Above maps to DTS, either:
  # 1. compatible = "rohm,bd79104";
  # 2. compatible = "rohm,bd79115", "rohm,bd79104";
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

How to Start with a DT Binding?

```
properties:
  compatible:
    const: rohm,bd79104

  # No need to write descriptions for obvious fields and cases (SPI chip select):
  reg:
    maxItems: 1

  "#io-channel-cells":
    const: 1

  spi-max-frequency:
    maximum: 20000000

  # Regulator supplies, based on datasheet:
  iovdd-supply: true
  vdd-supply: true
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

How to Start with a DT Binding?

```
properties:
  ...

# List which properties are required by the hardware or by drivers:
required:
  - compatible
  - reg
  # Devices rarely operate without power, even if Linux will provide "dummies":
  - iovdd-supply
  - vdd-supply
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

How to Start with a DT Binding?

```
properties:
  ...

# List which properties are required by the hardware or by drivers:
required:
  ...

# This is an SPI device, so needs to reference the schema for SPI devices.
# Often there are other schemas with common properties for devices of similar type.
allOf:
  - $ref: /schemas/spi/spi-peripheral-props.yaml#

# This binding referenced other schema (spi-peripheral-props.yaml) and all properties
# from there apply, thus we want to disallow properties not evaluated so far.
# Not evaluated here, not in referenced schemas, not in dtschema itself.
# If we did not reference other schema, this would be "additionalProperties: false".
unevaluatedProperties: false
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

How to Start with a DT Binding?

```
# Examples are used in validation of the binding itself, so they should be complete and correct.
# Do not add more examples than needed. Also no need for consumer examples in provider bindings.
examples:
  - |
    spi {
        #address-cells = <1>;
        #size-cells = <0>;

        adc@0 {
            compatible = "rohm,bd79104";
            reg = <0>;
            #io-channel-cells = <1>;
            spi-max-frequency = <4000000>;
            iovdd-supply = <&iovdd_supply>;
            vdd-supply = <&vdd_supply>;
            # Important: No "status" property in examples
        };
    };
```

Code snippet: SPDX-License-Identifier: GPL-2.0-only OR BSD-2-Clause

What Could You Put into DTS?

What Could You Put into DTS?

- TLDR: only hardware
  - or underlying parts of non-discoverable firmware
- Device nodes represent real hardware devices
  - **Not the Linux Device Driver model**
- Properties represent observable hardware characteristics or board wiring
  - No Linux driver choices
    - Usually **don't even reference drivers** in the commit messages or the DT binding description
  - No OS policies
  - No runtime knobs

What Could You Put into DTS?

- I need a child node to instantiate Linux driver (e.g. from MFD driver)
  - NO (usually)
  - Is the hardware piece of your child device re-usable (e.g. RTC block in PMIC)?
  - Does the child node have distinctive resources (e.g. address space, clocks, pins, resets, supplies)?
  - Instead: Fold the child properties into the parent node
- My driver needs to notify others for hot plug events
  - NO
  - Instead: Describe the actual hardware problem
- My driver should not register power supply class for this device
  - NO
  - Instead: Compatible already implies that (e.g. “this variant does / does not have charger”)
- My driver needs to program register “foo”, so I need “vendor,foo-value”
  - DEPENDS
  - Instead: Please rephrase the problem and the property, to describe the actual hardware feature

What Could You Put into DTS?

- “vendor,allowed-voltages-microvolt”, because my new device, which is compatible with an older one, does not support all voltages
  - NO
  - Instead: Your front, dedicated compatible defines that
- “clock-frequency” property, because I need to configure registers based on some clock
  - NO (usually)
  - Instead: Your device has clock input, thus needs “clocks” property. Your driver should use clk_get_rate().
  - Instead: If you need to configure clock to given frequency: git grep assigned-clock-rates
- I need new “vendor,foo-prop” property
  - WAIT
  - Please look at existing common properties from common schemas or devices representing similar class

What Could You Put into DTS?

- My SoC has two, almost the same PCIe PHYs with slight differences, I need “instance-id” property or custom OF alias to figure out which one is which
  - NO
  - Instead: Maybe other existing, common properties express it, e.g. “num-lanes”
  - Instead: If they have different programming model, then different compatible
  - Instead: Maybe the phandle (cell) arguments from consumer node expresses it, e.g.:

```
phys = <&socphy PHY_TYPE_PCIE>;
```

Validate Your DTS and Live Error-free Ever After

Validate DTS and Bindings

- Validate your binding

```
$ make dt_binding_check DT_SCHEMA_FILES=trivial-devices.yaml
```

- Validate all DTS files in given ARCH against your binding
  - Remember about cross compile options and ARCH=foo
  - This might not show all warnings, because it skips other schema files

```
$ make dtbs_check DT_SCHEMA_FILES=trivial-devices.yaml
```

- Validate one DTS
  - You pass the DTB Makefile target, not DTS

```
$ make CHECK_DTBS=y qcom/sm8450-hdk.dtb
```

- Validate one DTS against your binding
  - You pass the DTB Makefile target, not DTS

```
$ make CHECK_DTBS=y qcom/sm8450-hdk.dtb DT_SCHEMA_FILES=trivial-devices.yaml
```

Thank you

Krzysztof Kozlowski
krzk@kernel.org, @krzk@social.kernel.org

Thank you

For more information, visit us at qualcomm.com & qualcomm.com/blog