Using Device Tree Overlays to Support Complex PCI Devices in Linux

Hervé Codina
herve.codina@bootlin.com

Hervé Codina

- Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Contributor to the Microchip LAN966x PCI driver in Linux
  - Strong open-source focus
- Open-source contributor
- Living in **Toulouse**, France

Use case

Use case

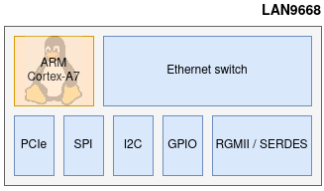
- Microchip LAN966x PCIe device
- AMD Alveo FPGA PCIe cards
  - Peripheral controllers exposed on PCIe BARs (DMA, UARTs, ...)
  - Peripherals have they own drivers available upstream
- The ASIX9100 multi purpose device (GPIO, I2C, SPI, ...)
  https://lore.kernel.org/lkml/bad63409-ed2b-4cef-988b-3c143636e9fa@alliedtelesis.co.nz/
- The RaspberryPI RP1 PCIe device
  https://lore.kernel.org/lkml/cover.1748526284.git.andrea.porta@suse.com/
- ...
- Linux Plumbers 2023 'Non-discoverable devices in PCI devices' from Rob Herring
  https://www.youtube.com/watch?v=MVGElnZW7BQ

Microchip LAN966x Chip

Microchip LAN966x chip: Two operating modes

- Traditional SoC
  - Already supported
- PCIe device
  - Our use case

Microchip LAN966x SoC



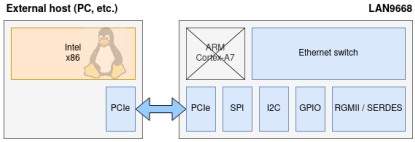
- Traditional SoC
  - Internal CPU cores
  - Set of peripherals (reset, clocks, GPIOs, I2C, ...)
  - Described by Device Tree `arch/arm/boot/dts/microchip/lan966x.dtsi`

Microchip LAN966x SoC, lan966x.dtsi

Extracted and simplified from lan966x.dtsi

```
/ {
        model = "Microchip LAN966 family SoC";
        compatible = "microchip,lan966";

        soc {
                compatible = "simple-bus";
                #address-cells = <1>;
                #size-cells = <1>;
                ranges;

                switch: switch@e0000000 {
                        compatible = "microchip,lan966x-switch";
                        reg = <0xe0000000 0x0100000>,
                              <0xe2000000 0x0800000>;
                        reg-names = "cpu", "gcb";
                        interrupts = <GIC_SPI 12 IRQ_TYPE_LEVEL_HIGH>,
                                     <GIC_SPI 9 IRQ_TYPE_LEVEL_HIGH>;
                        interrupt-names = "xtr", "ana";
                        ...
                        ethernet-ports {
                                ...
                                port0: port@0 {
                                        ...
                                };
                        };
                };

                reset: reset-controller@e200400c {
                        compatible = "microchip,lan966x-switch-reset";
                        reg = <0xe200400c 0x4>;
                };

                gpio: pinctrl@e2004064 {
                        compatible = "microchip,lan966x-pinctrl";
                        reg = <0xe2004064 0xb4>,
                              <0xe2010024 0x138>;
                        gpio-controller;
                        #gpio-cells = <2>;
                };

                mdio1: mdio@e200413c {
                        compatible = "microchip,lan966x-miim";
                        ...
                        phy0: ethernet-phy@1 {
                                ...
                        };
                };
                ...
        };
};
```

Microchip LAN966x PCIe device



- PCIe device
  - Internal CPU cores replaced by a PCIe endpoint
    - Peripherals accessed by the PCIe root complex
    - Memory-mapped I/O through PCIe BARs
    - Interrupts routed to PCIe INTx interrupt
  - Same set of peripherals
    - Drivers can be reused

LAN966x PCI driver

LAN966x PCIe device → PCI driver

- Match LAN966x PCI Vendor/Device ID
  - Loaded only for the LAN966x PCI device
- How to instantiate other drivers from the LAN966x PCI driver ?

Drivers Instantiation

Driver reuse

- How to instantiate other drivers from the LAN966x PCI driver ?
  - Drivers used in SoC: **Based on DT**
  - Do not reinvent the wheel: **Avoid drivers modifications**
  - Avoid old board.c description: **Avoid massive description** in PCI driver **C code**

Driver reuse

- How to instantiate other drivers from the LAN966x PCI driver ?
  - Drivers used in SoC: **Based on DT**
  - Do not reinvent the wheel: **Avoid drivers modifications**
  - Avoid old board.c description: **Avoid massive description** in PCI driver **C code**
- Use a **Device Tree overlay**

DT overlay

- Device Tree description
- Modify base Device Tree at runtime
  - Add DT nodes/properties when applied
  - Remove DT nodes/properties when removed
- Can be applied on a specific DT node

DT overlay for LAN966x

Extracted and simplified from lan966x.dtsi

```
/ {
        model = "Microchip LAN966 family SoC";
        compatible = "microchip,lan966";



        soc {
                compatible = "simple-bus";
                #address-cells = <1>;
                #size-cells = <1>;



                ranges;


                switch: switch@e0000000 {
                        compatible = "microchip,lan966x-switch";
                        reg = <0xe0000000 0x0100000>,
                              <0xe2000000 0x0800000>;
                        reg-names = "cpu", "gcb";


                        interrupts = <GIC_SPI 12 IRQ_TYPE_LEVEL_HIGH>,
                                     <GIC_SPI 9 IRQ_TYPE_LEVEL_HIGH>;
                        interrupt-names = "xtr", "ana";
                        ...
                        ethernet-ports {
                                ...
                        };
                };

                reset: reset-controller@e200400c {
                        compatible = "microchip,lan966x-switch-reset";
                        reg = <0xe200400c 0x4>;
                };

                gpio: pinctrl@e2004064 {
                        compatible = "microchip,lan966x-pinctrl";
                        reg = <0xe2004064 0xb4>,
                              <0xe2010024 0x13B>;
                        gpio-controller;
                        #gpio-cells = <2>;
                };

                mdio1: mdio@e200413c {
                        compatible = "microchip,lan966x-miim";
                        ...
                };

        };
};
```

Extracted and simplified from LAN966x PCI device DT overlay

```
/ {
        fragment@0 {
                target-path="";
                __overlay__ {
                        #address-cells = <1>;
                        #size-cells = <2>;

                        pci-ep-bus@0 {
                                compatible = "simple-bus";
                                #address-cells = <1>;
                                #size-cells = <1>;
                                /*
                                 * map 0@xe2000000 (32MB) to BAR0 (CPU)
                                 * map 0@xe0000000 (16MB) to BAR1 (AMBA)
                                 */
                                ranges = <0xe2000000 0x00 0x00 0x00 0x2000000
                                          0xe0000000 0x01 0x00 0x00 0x1000000>;

                                switch: switch@e0000000 {
                                        compatible = "microchip,lan966x-switch";
                                        reg = <0xe0000000 0x0100000>,
                                              <0xe2000000 0x0800000>;
                                        reg-names = "cpu", "gcb";

                                        interrupt-parent = <&oic>;
                                        interrupts = <12 IRQ_TYPE_LEVEL_HIGH>,
                                                     <9 IRQ_TYPE_LEVEL_HIGH>;
                                        interrupt-names = "xtr", "ana";
                                        ...
                                        ethernet-ports {
                                                ...
                                        };
                                };

                                reset: reset@e200400c {
                                        compatible = "microchip,lan966x-switch-reset";
                                        reg = <0xe200400c 0x4>;
                                };

                                gpio: pinctrl@e2004064 {
                                        compatible = "microchip,lan966x-pinctrl";
                                        reg = <0xe2004064 0xb4>,
                                              <0xe2010024 0x13B>;
                                        gpio-controller;
                                        #gpio-cells = <2>;
                                };

                                mdio1: mdio@e200413c {
                                        compatible = "microchip,lan966x-miim";
                                        ...
                                };
                        };
                };
        };
};
```

Attach DT overlay

Where to attach the overlay ?

The overlay describes the PCI board (internal components)

- Attach to the DT node related to the PCI board

Where to attach the overlay ?

The overlay describes the PCI board (internal components)

- Attach to the DT node related to the PCI board ?
  - DT nodes for PCI devices not present in base DT
  - PCI devices discovered at runtime (PCI enumeration)
  - PCI topology (bridges, devices) are system specific
  - Only the PCI controller (PCI root bridge) is present in a base DT

Marvell Armada base DT

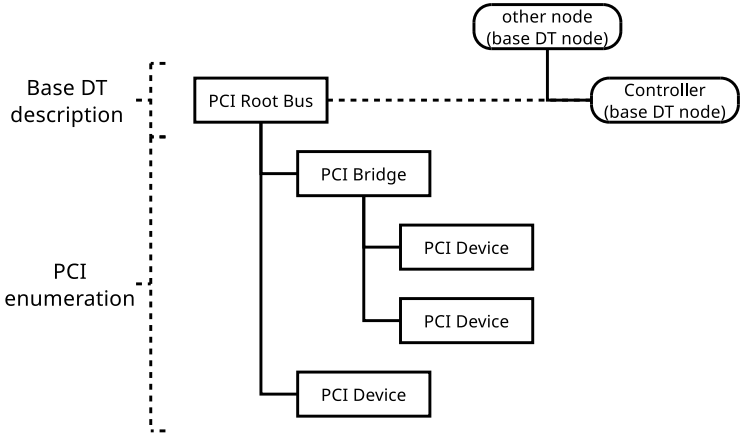
```
/ {
        model = "Marvell Armada 37xx SoC";
        compatible = "marvell,armada3700";

        soc {
                pcie0: pcie@d0070000 {
                        compatible = "marvell,armada-3700-pcie";
                        device_type = "pci";
                        reg = <0 0xd0070000 0 0x20000>;
                        ...
                        /*
                         * No sub-nodes for bridges and devices attached to the
                         * PCI bus
                         */
                };
        };
};
```

PCI enumeration

- From PCI root bridge, scan the PCI bus (heavily simplified)
  1. A PCI Device or Bridge is detected on the PCI bus
  2. Create a `struct pci_dev` for this device
  3. Compute and assign resources
  4. Bridge or Device ?
     - if Bridge: Continue enumeration scanning busses behind the bridge
     - if Device: Ok, look at next component connected to the PCI bus

PCI enumeration

- No DT nodes for PCI bridges and devices

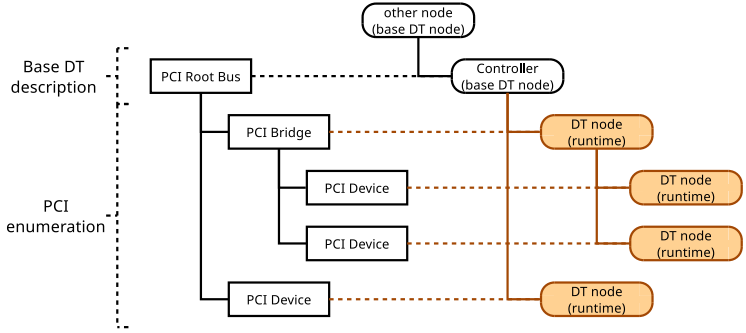


Create missing nodes

- Create missing PCI DT nodes at runtime
  - `CONFIG_PCI_DYNAMIC_OF_NODES=y`.
  - Creation done during the PCI enumeration.
- From PCI root bridge, scan the PCI bus (heavily simplified)
  1. A PCI Device or Bridge is detected on the PCI bus
  2. Create a `struct pci_dev` for this device
  3. Compute and assign resources
  4. **Create a DT node for this device**
     - **Set DT properties to value based on computed and assigned resources**
  5. Bridge or Device ?
     - if Bridge: Continue enumeration scanning busses behind the bridge
     - if Device: Ok, look at next component connected to the PCI bus

Create missing nodes

- DT nodes for PCI bridges and devices available



PCI device/bridge DT node creation

- `CONFIG_PCI_DYNAMIC_OF_NODES=y`
- Available since kernel v6.6
- `of_pci_make_dev_node()` (call from `pci_bus_add_device()`).
  - Direct call for PCI bridges
  - Using final fixup (`DECLARE_PCI_FIXUP_FINAL()` per PCI device)
  - Create node
    - Bridge node name: `pci@<slot_number>,<function_number>`
    - Device node name: `dev@<slot_number>,<function_number>`
  - Add node properties using `of_pci_add_properties()`.
  - Attach node to parent DT node
  - Attach node to `struct device` (PCI device or bridge)
- `of_pci_remove_node()` (call from `pci_stop_dev()`).
  - Detach node from `struct device`
  - Detach node from the parent DT node
  - Destroy node

PCI device bridge DT node creation

Properties added by `of_pci_add_properties()`

- For bridges
  - `compatible`
  - `reg`
  - `device_type = "pci"`
  - `#address-cells` and `#size-cells`
  - `ranges`
  - ...
- For devices
  - `compatible`
  - `reg`
  - `#address-cells` and `#size-cells`
  - `ranges`
  - `#interrupt-cells` and `interrup-controller`.
  - ...

PCI device/bridge DT node creation

Interesting DT nodes properties

- Address translations:
  - `ranges`: Used at each PCI level in the PCI tree
- Interrupt translations:
  - `interrupt-controller`: Consider the PCI device as an interrupt controller

'ranges' property, simple translation

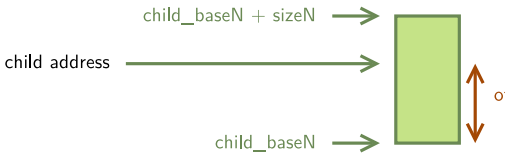
#address-cells
for child address cells

#address-cells
for parent address cells

#size-cells
for child size cells

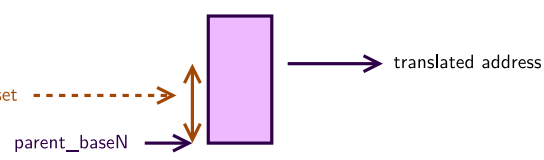
```
ranges = < child_base1 parent_base1 size1 >,
         < child_base2 parent_base2 size2 >,
         < child_base3 parent_base3 size3 >;
```

1) Find matching item



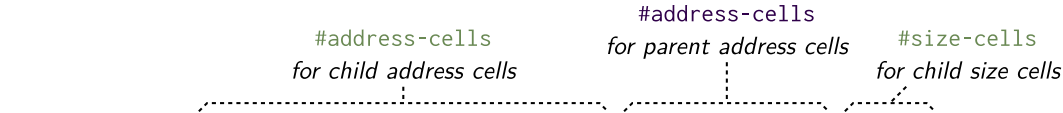
child_baseN ≤ child address < child_baseN + sizeN

2) Translate using matching item N



translated address = parent_baseN + offset
with
offset = child address - child_baseN

'ranges' property, pci translation



'ranges' property, PCI Flags

- Flags are defined only for `device_type = "pci"`.
- 32bit word: npt000**ss** bbbbbbbb ddddd fff rrrrrrrr
  - n: Relocatable region
  - p: Prefetchable region
  - t: Aliased address flag
  - **ss: Space code**
    - **00: Configuration space**
    - **01: I/O space**
    - **10: 32bit memory space**
    - **11: 64bit memory space**
  - bbbbbbbb: PCI bus number
  - ddddd: Device number
  - fff: Function number
  - rrrrrrrr: Register number

'ranges' property, pci translation

#address-cells
for child address cells

#address-cells
for parent address cells

#size-cells
for child size cells

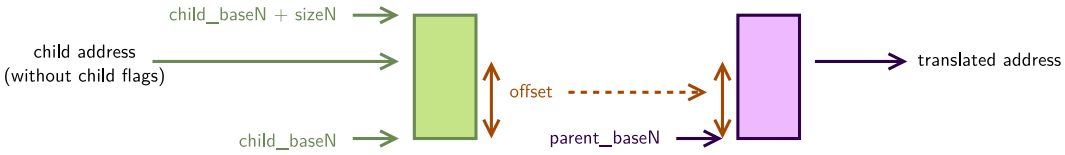
```
ranges = < child_flags1 child_base1 parent_base1 size1 >,
         < child_flags2 child_base2 parent_base2 size2 >,
         < child_flags3 child_base3 parent_base3 size3 >;

device-type = "pci";
```

1) Find matching item

child flags matches child_flagsN ('ss' flags)

&&

2) Translate using matching item N



'ranges' property

- PCI host bridge:
  - PCI translation
  - Translate from PCI addresses to host bus addresses
- PCI bridges:
  - PCI translation
  - Translate from PCI secondary busses addresses to PCI primary busses addresses
- PCI devices:
  - Simple translation
  - Translate from PCI BARs to PCI addresses
- Device-tree overlay
  - Simple translation
  - Translate from chip addresses to BARs addresses

'ranges' property, Example

```
pcie@d0070000 {
  compatible = "marvell,armada-3700-pcie";
  #address-cells = <0x03>;
  #size-cells = <0x02>;
  device_type = "pci";
  /*                  <- Child flags and addr  -> <-parent addr-> <--- size  ---> */
  ranges = /*MEM32*/ <0x82000000 0x00 0xe8000000 0x00 0xe8000000 0x00 0x7f00000>,
           /*IO   */ <0x81000000 0x00 0x00000000 0x00 0xefff0000 0x00 0x0010000>;

  pci@0,0 {
    compatible = "pci11ab,100", "pciclass,060400", "pciclass,0604";
    #address-cells = <0x03>;
    #size-cells = <0x02>;
    device_type = "pci";
    /*                  <- Child flags and addr  -> <-parent flags and addr -> <--- size  ---> */
    ranges = /*MEM32*/ <0x82000000 0x00 0xe8000000 0x82000000 0x00 0xe8000000 0x00 0x4400000>;

    dev@0,0 {
      compatible = "pci1055,9660", "pciclass,020000", "pciclass,0200";
      #address-cells = <0x03>;
      #size-cells = <0x02>;
      /*                <-child addr -> <-parent flags and addr -> <--- size  ---> */
      ranges = /*BAR0*/ <0x00 0x00 0x00 0x82010000 0x00 0xe8000000 0x00 0x2000000>,
               /*BAR1*/ <0x01 0x00 0x00 0x82010000 0x00 0xea000000 0x00 0x1000000>,
               /*BAR2*/ <0x02 0x00 0x00 0x82010000 0x00 0xeb000000 0x00 0x0800000>;

      pci-ep-bus@0 {
        compatible = "simple-bus";
        #address-cells = <0x01>;
        #size-cells = <0x01>;
        /*       <child addr> <-parent addr-> <- size -> */
        ranges = <0xe2000000  0x00 0x00 0x00  0x2000000>, /* 0xe2000000 translated using BAR 0 */
                 <0xe0000000  0x01 0x00 0x00  0x1000000>; /* 0xe0000000 translated using BAR 1 */

        reset@e200400c {
          compatible = "microchip,lan966x-switch-reset";
          reg = <0xe200400c 0x04>;
        };
      ...
```

reset controller: **reg = 0xe200400c**

- translated at pci-ep-bus
  chip addr → BAR addr
  `addr = 0x00 0x00 0x400c`
- translated at dev@0,0
  BAR addr → PCI addr
  `addr = 0x82010000 0x00 0xe800400c`
- translated at pci@0,0
  PCI addr → PCI addr
  `addr = 0x82010000 0x00 0xe800400c`
- translated at pcie@d0070000
  PCI addr → host addr
  `addr = 0x00 0xe800400c`

From CPU: **0x00000000e800400c**

'interrupt-map' property

The `interrupt-map` property is used to re-map interrupts.

```
#interrupt-cells = <1>;
#size-cells = <2>;
#address-cells = <3>;
interrupt-map-mask = <0xf800 0 0 7>;
interrupt-map = <0x9000 0 0 1 &open-pic 3 1>, /* INTA */
                <0x9000 0 0 2 &open-pic 4 1>, /* INTB */
                <0x9000 0 0 3 &open-pic 1 1>, /* INTC */
                <0x9000 0 0 4 &open-pic 2 1>; /* INTD */
```

- A phandle for the parent interrupt controller is needed

'interrupt-map' property, phandle issue

- A phandle for the parent interrupt controller is needed
- Not always available (ACPI)
- Do not use `interrupt-map`
  → Consider the **PCI device as an interrupt controller**
  (available since kernel v6.11)

PCI device as an interrupt controller

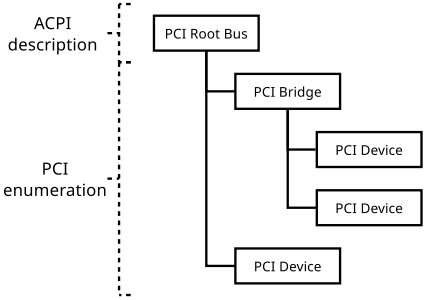
```
dev@0,0 {       /* <---- PCI device DT node */
    compatible = "pci1055,9660", "pciclass,020000", "pciclass,0200";

    /* PCI device as an interrupt controller */
    #interrupt-cells = <0x01>;
    interrupt-controller;

    pci-ep-bus@0 {
        compatible = "simple-bus";

        switch: switch@e0000000 {
            compatible = "microchip,lan966x-switch";
            reg = <0xe0000000 0x0100000>,
                  <0xe2000000 0x0800000>;
            reg-names = "cpu", "gcb";

            interrupt-parent = <&oic>;
            interrupts = <12 IRQ_TYPE_LEVEL_HIGH>,
                         <9 IRQ_TYPE_LEVEL_HIGH>;
            interrupt-names = "xtr", "ana";
            ...
        };

        oic@e00c0120 {
            /* Interrupt controller available in the LAN966x Chip */
            compatible = "microchip,lan966x-oic";
            reg = <0xe00c0120 0x190>;

            /* No interrupt-parent property */
            interrupts = <0x00>;  /* <---- Connected to PCI INTx */

            #interrupt-cells = <0x02>;
            interrupt-controller;
        };
    };
};
```

- `switch` needs interrupts 9 and 12 from `oic`
  - Classical interrupt description with `interrupts`, `interrupts-parents`
- `oic` needs interrupts 0 (PCI INTx)
  - No `interrupts-parents`
  - Walk parent nodes until an interrupt controller is found
  - Found the LAN966x PCI device node
  - Handled by the LAN966x PCI driver
    - Create an interrupt domain
    - Route interrupt 0 to the PCI INTx

ACPI

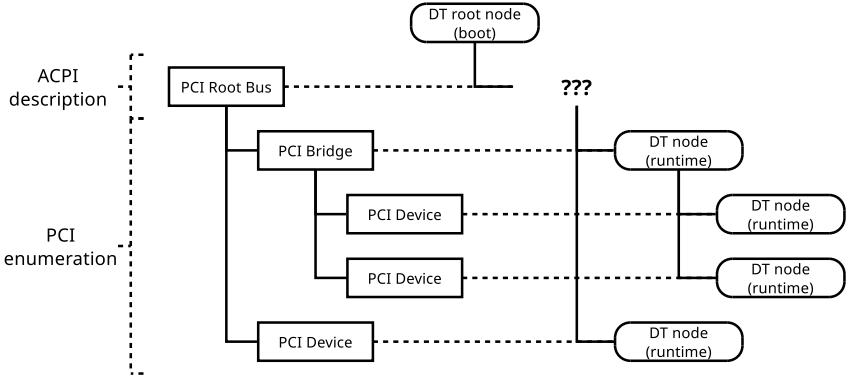
ACPI and DT

- On x86, hardware description done by ACPI.
- No Device Tree.



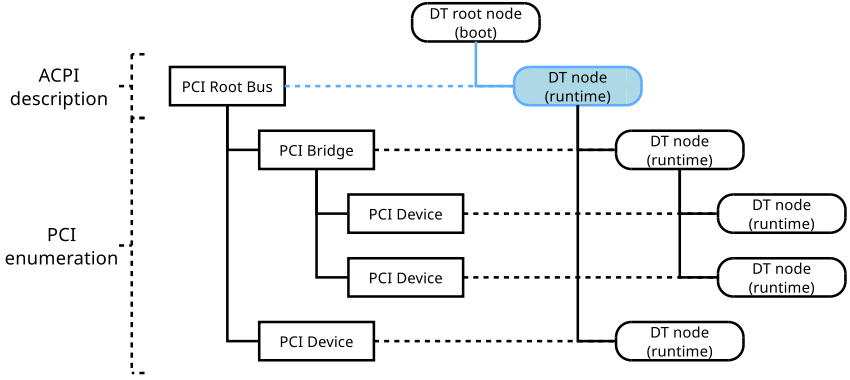
ACPI and DT

- On x86, hardware description done by ACPI.
- ~~No Device Tree~~ Device Tree available.
  - Empty DT root node created at boot
  - Create missing PCI bridge/devices DT nodes (`CONFIG_PCI_DYNAMIC_OF_NODES`)
  - No PCI root bus DT node



ACPI and DT

- On x86, hardware description done by ACPI.
- ~~No Device Tree~~ Device Tree available.
  - Empty DT root node created at boot
  - Create missing PCI bridge/devices DT nodes (`CONFIG_PCI_DYNAMIC_OF_NODES`)
  - No PCI root bus DT node → **Create a DT node when the host bridge registers**



PCI host bridge DT node creation

- `CONFIG_PCI_DYNAMIC_OF_NODES=y`
- Available since kernel v6.15
- `of_pci_make_host_bridge_node()` (call from `pci_register_host_bridge()`).
  - Create node
    - node name: `pci@<domain_number>,<bus_number>`
  - Add node properties using `of_pci_add_host_bridge_properties()`.
  - Avoid platform bus to handle this device (node attached to root node)
  - Attach created node to root DT node
  - Attach created node to `struct device` (PCI host bridge, PCI root bus)
- `of_pci_remove_host_bridge_node()` (call from `pci_stop_root_bus()`).
  - Detach node from `struct device`
  - Detach node from the parent DT node (root DT node)
  - Destroy the node

PCI host bridge DT node creation

Properties added by `of_pci_add_host_bridge_properties()`

- `device_type = "pci"`
- `#address-cells`, `#size-cells`
- `ranges`
- ...

LAN966x PCI driver

LAN966x PCI driver, PCI device as an interrupt controller

Simplified (error check and error path removed)

```
struct pci_dev_intr_ctrl {
        struct pci_dev *pci_dev;
        struct irq_domain *irq_domain;
        int irq;
};

static int pci_dev_irq_domain_map(struct irq_domain *d, unsigned int virq, irq_hw_number_t hw)
{
        irq_set_chip_and_handler(virq, &dummy_irq_chip, handle_simple_irq);
        return 0;
}

static const struct irq_domain_ops pci_dev_irq_domain_ops = {
        .map = pci_dev_irq_domain_map,
        .xlate = irq_domain_xlate_onecell,
};

static irqreturn_t pci_dev_irq_handler(int irq, void *data)
{
        struct pci_dev_intr_ctrl *intr_ctrl = data;
        int ret;

        ret = generic_handle_domain_irq(intr_ctrl->irq_domain, 0);
        return ret ? IRQ_NONE : IRQ_HANDLED;
}

static struct pci_dev_intr_ctrl *pci_dev_create_intr_ctrl(struct pci_dev *pdev)
{
        struct pci_dev_intr_ctrl *intr_ctrl;
        struct fwnode_handle *fwnode;
        int ret;

        fwnode = dev_fwnode(&pdev->dev);
        if (!fwnode)
                return ERR_PTR(-ENODEV);

        intr_ctrl = kmalloc(sizeof(*intr_ctrl), GFP_KERNEL);
        intr_ctrl->pci_dev = pdev;
        intr_ctrl->irq_domain = irq_domain_create_linear(fwnode, 1, &pci_dev_irq_domain_ops,
                                                         intr_ctrl);
        pci_alloc_irq_vectors(pdev, 1, 1, PCI_IRQ_INTX);
        intr_ctrl->irq = pci_irq_vector(pdev, 0);
        request_irq(intr_ctrl->irq, pci_dev_irq_handler, IRQF_SHARED,
                    pci_name(pdev), intr_ctrl);

        return intr_ctrl;
}
```

- Handle a dedicated IRQ domain
- 1 interrupt in the domain
- Forward the INTx interrupt to the interrupt in the domain

```
static void pci_dev_remove_intr_ctrl(struct pci_dev_intr_ctrl *intr_ctrl)
{
        free_irq(intr_ctrl->irq, intr_ctrl);
        pci_free_irq_vectors(intr_ctrl->pci_dev);
        irq_dispose_mapping(irq_find_mapping(intr_ctrl->irq_domain, 0));
        irq_domain_remove(intr_ctrl->irq_domain);
        kfree(intr_ctrl);
}

static void devm_pci_dev_remove_intr_ctrl(void *intr_ctrl)
{
        pci_dev_remove_intr_ctrl(intr_ctrl);
}

static int devm_pci_dev_create_intr_ctrl(struct pci_dev *pdev)
{
        struct pci_dev_intr_ctrl *intr_ctrl;

        intr_ctrl = pci_dev_create_intr_ctrl(pdev);
        if (IS_ERR(intr_ctrl))
                return PTR_ERR(intr_ctrl);

        return devm_add_action_or_reset(&pdev->dev, devm_pci_dev_remove_intr_ctrl, intr_ctrl);
}
```

LAN966x PCI driver, Load/Unload the DT overlay

- DT overlay (dtbo file) embedded in the driver module
- Load the overlay at the the device DT node (`dev->of_node`)

```c
/* Embedded dtbo symbols created by cmd_wrap_S_dtb in scripts/Makefile.lib */
extern char __dtbo_lan966x_pci_begin[];
extern char __dtbo_lan966x_pci_end[];

struct lan966x_pci {
        struct device *dev;
        struct pci_dev *pci_dev;
        int ovcs_id;
};

static int lan966x_pci_load_overlay(struct lan966x_pci *data)
{
        u32 dtbo_size = __dtbo_lan966x_pci_end - __dtbo_lan966x_pci_begin;
        void *dtbo_start = __dtbo_lan966x_pci_begin;
        int ret;

        ret = of_overlay_fdt_apply(dtbo_start, dtbo_size, &data->ovcs_id,
                                   data->dev->of_node);
        if (ret)
                return ret;

        return 0;
}

static void lan966x_pci_unload_overlay(struct lan966x_pci *data)
{
        of_overlay_remove(&data->ovcs_id);
}
```

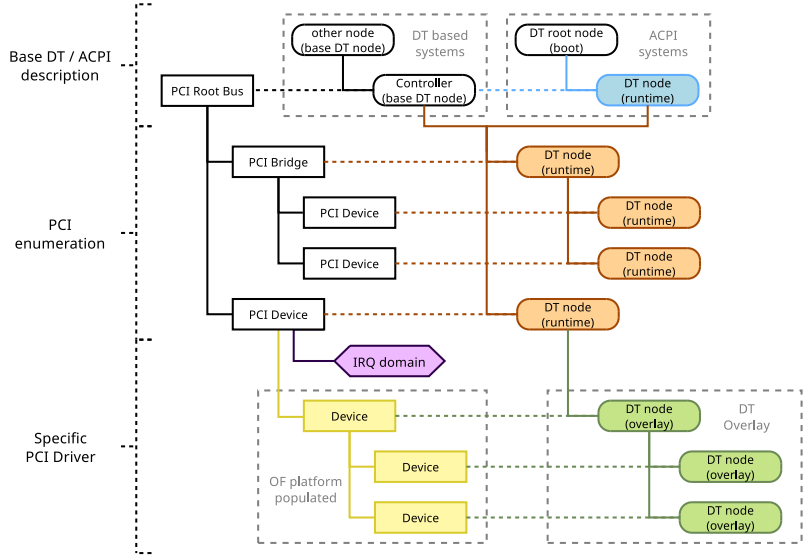
LAN966x PCI driver, Probe/Remove

```c
static int lan966x_pci_probe(struct pci_dev *pdev, const struct pci_device_id *id)
{
        struct device *dev = &pdev->dev;
        struct lan966x_pci *data;
        int ret;

        if (!dev->of_node) {
                dev_err(dev, "Missing of_node for device\n");
                return -EINVAL;
        }

        ret = pcim_enable_device(pdev);
        if (ret)
                return ret;

        ret = devm_pci_dev_create_intr_ctrl(pdev);
        if (ret)
                return ret;

        data = devm_kzalloc(dev, sizeof(*data), GFP_KERNEL);
        if (!data)
                return -ENOMEM;

        dev_set_drvdata(dev, data);
        data->dev = dev;
        data->pci_dev = pdev;

        ret = lan966x_pci_load_overlay(data);
        if (ret)
                return ret;

        pci_set_master(pdev);

        ret = of_platform_default_populate(dev->of_node, NULL, dev);
        if (ret)
                goto err_unload_overlay;

        return 0;

err_unload_overlay:
        lan966x_pci_unload_overlay(data);
        return ret;
}
```

- Create the interrupt controler
- Load the overlay
- Populate platform devices from device-tree overlay loaded at `dev->of_node`
- Driver available since kernel v6.13

```c
static void lan966x_pci_remove(struct pci_dev *pdev)
{
        struct device *dev = &pdev->dev;
        struct lan966x_pci *data = dev_get_drvdata(dev);

        of_platform_depopulate(dev);

        lan966x_pci_unload_overlay(data);

        pci_clear_master(pdev);
}

static struct pci_device_id lan966x_pci_ids[] = {
        { PCI_DEVICE(0x1055, 0x9660) },
        { 0, }
};
MODULE_DEVICE_TABLE(pci, lan966x_pci_ids);

static struct pci_driver lan966x_pci_driver = {
        .name = "mchp_lan966x_pci",
        .id_table = lan966x_pci_ids,
        .probe = lan966x_pci_probe,
        .remove = lan966x_pci_remove,
};
module_pci_driver(lan966x_pci_driver);
```

Big picture hierarchy



Specific drivers/sub-systems issues

Specific drivers/sub-systems issues

- SoC designed components, SoC use case:
  - Builtin
  - Boot time instantiation

Specific drivers/sub-systems issues

- SoC designed components, new use case:
  - ~~Builtin~~ Built as modules
  - ~~Boot time instantiation~~ Support insertions and removals

Specific drivers/sub-systems issues

- SoC designed components, new use case:
  - ~~Builtin~~ Built as modules
  - ~~Boot time instantiation~~ Support insertions and removals
- Consequences:
  - Ref counting issues
  - Registered component list issues
  - Memory leak issues
  - Dependencies issues
  - Race condition issues
  - ...

Specific drivers/sub-systems issues

- SoC designed components, new use case:
  - ~~Builtin~~ Built as modules
  - ~~Boot time instantiation~~ Support insertions and removals
- Consequences:
  - Ref counting issues
  - Registered component list issues
  - Memory leak issues
  - Dependencies issues
  - Race condition issues
  - ...
- Components impacted (sub-sytems and/or specific drivers), no blame:
  - syscon
  - reset
  - clocks
  - interrupts
  - fw_devlink
  - i2c muxes
  - ...

Thanks

- Thanks Clément Léger for:
  - Starting that work
  - Exploring several ways of doing
  - Drawing up basis
  - Fixing some issues
- Thanks Lizhi Hou for:
  - Introducing the PCI device/bridge DT node creations
- Thanks Maintainers for:
  - Reviewing, discussing, asking for improvements, ...

Questions? Suggestions? Comments?

Hervé Codina
herve.codina@bootlin.com

https://bootlin.com/pub/conferences/