PCI Subsystem -3- (Host Controller)

2018-10-17 (http://jake.dothome.co.kr/pci-3/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

PCI Host Controller Device Tree Nodes

Identify the following PCI nodes:

```
pci {
    compatible = "company,foo"
    device_type = "pci";        /* always be "pci" */
    #address-cells   = <3>;     /* always be 3 */
    #size-cells      = <2>;     /* 32bit=1, 64bit=2 */
    #interrupt-cells = <1>;     /* always be 1 for legacy pci irq */

    bus-range = <0x0 0x1>;

    /*    CPU_PHYSICAL(P#a)     SIZE(P#s) */
    reg = <0x0 0x40000000    0x0 0x1000000>;

    /*             PCI_PHYSICAL(#a)      CPU_PHYSICAL(P#a)   SIZE(#s) */
    ranges = <0x81000000 0x0 0x00000000   0x0 0x48000000    0x0 0x00010000>,  /* no
n-relocatable IO */
             <0x82000000 0x0 0x40000000   0x0 0x40000000    0x0 0x40000000>;  /* no
n-relocatable MEM */

    /*               PCI_DEVICE(#a)  INT#(#i)   IRQ-Controller   IRQ-UNIT#   IRQ_AR
G(PI#i) */
    interrupt-map = <  0x0 0x0 0x0    0x0           &gic            0x0       0x4 0
x1
                       0x800 0x0 0x0    0x1           &gic            0x0       0x5 0
x1
                      0x1000 0x0 0x0    0x2           &gic            0x0       0x6 0
x1
                      0x1800 0x0 0x0    0x4           &gic            0x0       0x7 0
x1>;

    /*                     PCI_DEVICE(#a)    INT#(#i) */
    interrupt-map-mask = <0xf800 0x0 0x0     0x7>;
}
```

- reg
  - The physical addresses and sizes to be used for access to the registers that will be used for the operation of the PCI host controller are specified.
- device_type
  - You should always substitute "pci".

- #address-cells
  - It must be 3 because it is used for PCI addresses.
- #size-cells
  - Memory sizes used by PCI and physical addresses
    - 1=32 bit memory size
    - 2=64 bit memory size
- interrupt-cells
  - Used when using legacy interrupts, which must be 1.
- bus-range
  - Start Bus Number ~ End Bus Number to Use
- ranges
  - The range in which the PCI addressing scheme must be translated to the physical addressing scheme used by the CPU is specified.
- interrrupt-map
  - Configure irq mapping information for up to four irq pins for PCI devices that use legacy interrupts.
- interrupt-map-mask
  - This is the mask information used to filter the valid value of the legacy interrupt value above.

Cells used by each property

The symbols used in the annotations indicate where the number of cells is known.

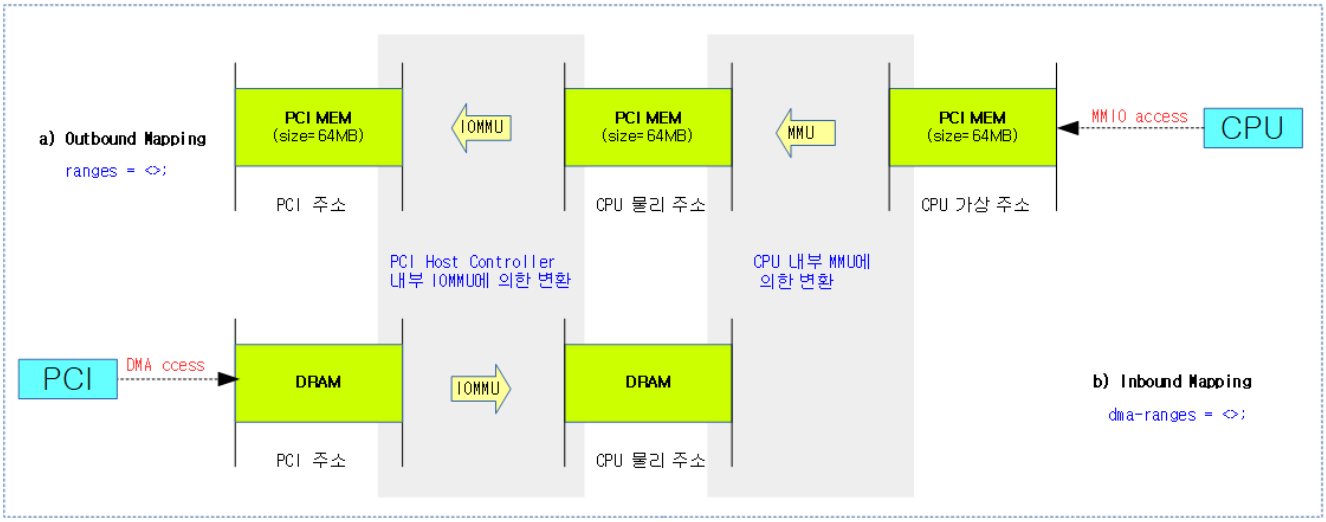
- #a
  - #address-cells value of PCI nodes
- #s
  - #size-cells value of PCI nodes
- #i
  - #interrupt-cells value of PCI nodes
- **P#a**
  - #address-cells value of the parent node
- **P#s**
  - #size-cells value of the parent node
- **PI#i**
  - #interrupt-cells value of parent interrupt controller node

```
pci {
    compatible = "company,foo"
    device_type = "pci";        /* always be "pci" */
    #address-cells   = <3>;     /* always be 3 */
    #size-cells      = <2>;     /* 32bit=1, 64bit=2 */
    #interrupt-cells = <1>;     /* always be 1 for legacy pci irq */

    bus-range = <0x0 0x1>;

    /*    CPU_PHYSICAL(P#a)     SIZE(P#s) */
    reg = <0x0 0x40000000    0x0 0x1000000>;

    /*             PCI_PHYSICAL(#a)      CPU_PHYSICAL(P#a)   SIZE(#s) */
    ranges = <0x81000000 0x0 0x00000000   0x0 0x48000000    0x0 0x00010000>,  /* no
n-relocatable IO */
             <0x82000000 0x0 0x40000000   0x0 0x40000000    0x0 0x40000000>;  /* no
n-relocatable MEM */

    /*               PCI_DEVICE(#a)  INT#(#i)   IRQ-Controller   IRQ-UNIT#   IRQ_AR
G(PI#i) */
    interrupt-map = <  0x0 0x0 0x0    0x0           &gic            0x0       0x4 0
x1
                       0x800 0x0 0x0    0x1           &gic            0x0       0x5 0
x1
                      0x1000 0x0 0x0    0x2           &gic            0x0       0x6 0
x1
                      0x1800 0x0 0x0    0x4           &gic            0x0       0x7 0
x1>;

    /*                    PCI_DEVICE(#a)    INT#(#i) */
    interrupt-map-mask = <0xf800 0x0 0x0     0x7>;
}
```

PCI Address Space Mapping

In order for the PCI device and CPU to access each other's areas, they need to be mapped according to the direction of their respective approaches.

- outbound mapping
  - Mapping for the CPU to access memory attached to a PCI device
- inbound mapping
  - Mapping for PCI devices to access host memory via DMA



(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-53b.png)

a) Outbound Mapping

One or more PCI addresses (I/O, MEM) used by a PCI device cannot be accessed and used by the CPU immediately. First, you need to use the IOMMU device that exists inside the PCI host controller to map the area of the PCI memory outbound to the empty space of the CPU's physical address. This mapped memory to the CPU physical address can be mapped back to the CPU's virtual address in the PCI device driver, allowing the CPU to access this area.

- Information such as the address ranges that the PCI host controller outbounds maps is expressed in the ranges attribute of the PCI node in the device tree.

Ranges Property

The host controller has a description for the area that maps the IO address space, the MEM32 space, and the MEM64 space outbound. If you have MEM32 and MEM64 in the pre-fetchable zone, you should configure them as separate zones. The cell values that make up the area are largely composed of 3 parts as follows.

- PCI address (source)
  - Specify the type and address of the PCI address space to be converted.
  - Use the number of cells equal to the value specified by the #address-cells value of the PCI host controller node. (This entry always uses a 3-digit cell.)
- CPU Physical Address (Destination)
  - Specify the starting physical address of the dest region that is translated to the PCI host controller.
  - Use the number of cells equal to the value specified by the #address-size value listed by the parent node.
- size
  - The size of the area that the host controller converts.
  - Use the number of cells equal to the value specified by the #size-cells value of the PCI host controller node. (32bit=1, 64bit=2)

The following ranges attribute tells the kernel that the two PCI memory regions will be converted to physical addresses. The PCI address is made up of 2 digit cells, and we'll see how to interpret them.

```
pci {
    reg = <0x0 0x50020000 0x0 0x1000>;
    ranges = <0x81000000 0x0 0x00000000 0x0 0x48000000 0x0 0x00010000>,
             <0x82000000 0x0 0x40000000 0x0 0x40000000 0x0 0x40000000>;
```

PCI 주소 cpu 물리 주소 사이즈

(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-49.png)

PCI Address

A PCI address always consists of three cells.

```
phys.hi cell:  npt000ss bbbbbbbb ddddd fff rrrrrrrr
phys.mid cell: hhhhhhhh hhhhhhhh hhhhhhhh hhhhhhhh
phys.low cell: llllllll llllllll llllllll llllllll
```

(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-50.png)

phys.hi cell

- n
  - Non-relocatable region flag
  - Non-relocatable flag (1=non-relocatable, 0=relocatable)
- p
  - Prefetchable (cacheable) zone flag (1=prefetchable, 0=non-prefetchable)
- t
  - Ten bits aliased address flag (1=Ten bits aliased, 0=otherwise)
  - Allows the use of hard-decoding with the lower 10 bits removed for the I/O and memory regions.
  - In other words, there is no need to relocate through the BAR.
- ss
  - Address Area Type
    - 00: Configuration space
    - 01: I/O space
    - 10: 32 bit Memory space
    - 11: 64 bit Memory space
- bbbbbbbb
  - PCI Bus Number
- ddddd
  - PCI device number (or slot number)
- fff
  - PCI Function Numbers
- rrrrrrrr
  - Register offset

phys.mid cell

- Top 64-bit 32-bit PCI addresses

phys.low cell

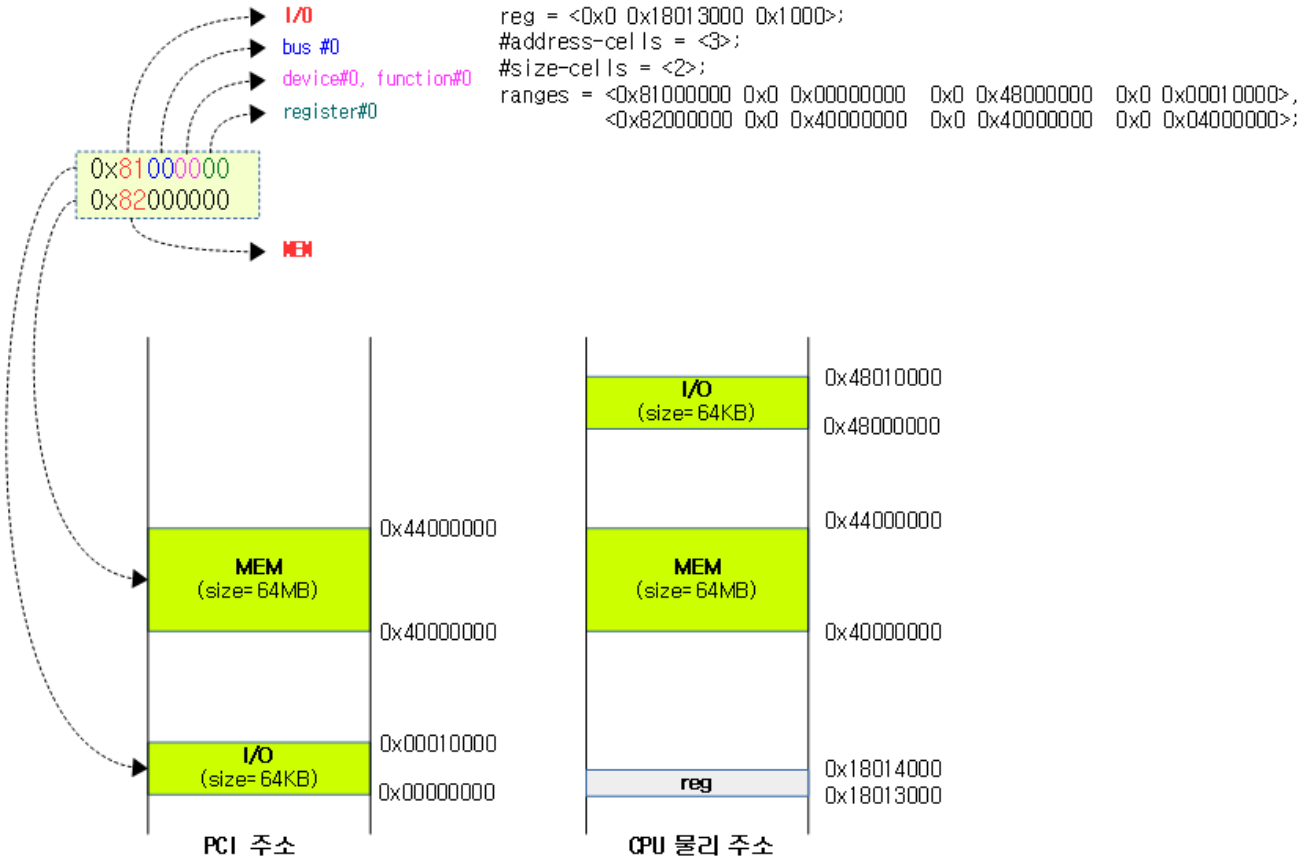
- Lower 64 bits of 32-bit PCI addresses

How to use according to SS type

Learn about configuration, I/O, and PCI address configuration by memory device type.

- ss=00: Configuration space
  - n, p, and t should all be zero.
  - Specify the address of the Configuration zone in bbbbbbbb, ddddd, fff, rrrrrrr.
  - h[31:0] must be all zeros.
  - l[31:0] must be all zeros.
- ss=01: I/O space
  - p should be 0.
  - 10-bit aliasing(t=1) is used for early PCs that use only 10-bit, but not for ARM and ARM64.
  - bbbbbbbbbb,ddddd,fff,rrrrrrrr specifies, i.e., the address that the BAR points to.
    - rrrrrrrr=0x10, 0x14, 0x18, 0x1c, 0x20, 0x24
    - rrrrrrrr=0x00 (non-relocatable)
  - h[31:0] must be all zeros.
  - n
    - If relocatable(n=0), specify a 31-bit offset of the relocatable start zone in the I/O region at l[0:32].
    - If non-relocatable(n=1), specify the 31-bit IO zone address in l[0:32].
  - ARM64 Use Cases: 0x01, 0x81
- ss=10: 32 bit memory space
  - p can be 0 or 1.
  - 10bit aliasing(t=1) is used for early PCs that use memory area up to 1M memory, but not for ARM and ARM64.
  - bbbbbbbbbb,ddddd,fff,rrrrrrrr specifies, i.e., the address that the BAR points to.
    - rrrrrrrr=0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x30
    - rrrrrrrr=0x00 (non-relocatable)
  - h[31:0] must be all zeros.
  - n
    - If relocatable(n=0), specify a relative offset to l[31:0].
      - 32bit PCI Momory start address = BAR + offset(l[31:0])
    - If non-relocatable(n=1), specify a 32-bit memory address.
      - 32bit PCI Momory start address = l[31:0]
  - arm64 use cases: 0x02, 0x42, 0x82
- ss=11: 64 bit memory space
  - ss=10, and h[31:0] is also used to use the parent PCI address.
  - arm64 use cases: 0x03, 0x43, 0x83

The following shows an example of the Ranges property for the I/O address space and the 32-bit memory address space on a PCI device.



(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-52.png)

b) inbound mapping

If you configure inbound mapping in the opposite direction to outbound, the PCI device will be able to access the host memory directly through the DMA device rather than through the CPU.

- Information such as address ranges inbound mapping by the PCI host controller is expressed in the device tree as the dma-ranges attribute of the PCI node
- It uses the same configuration as the ranges property used by the outbound mapping.

Let's take a look at the following example.

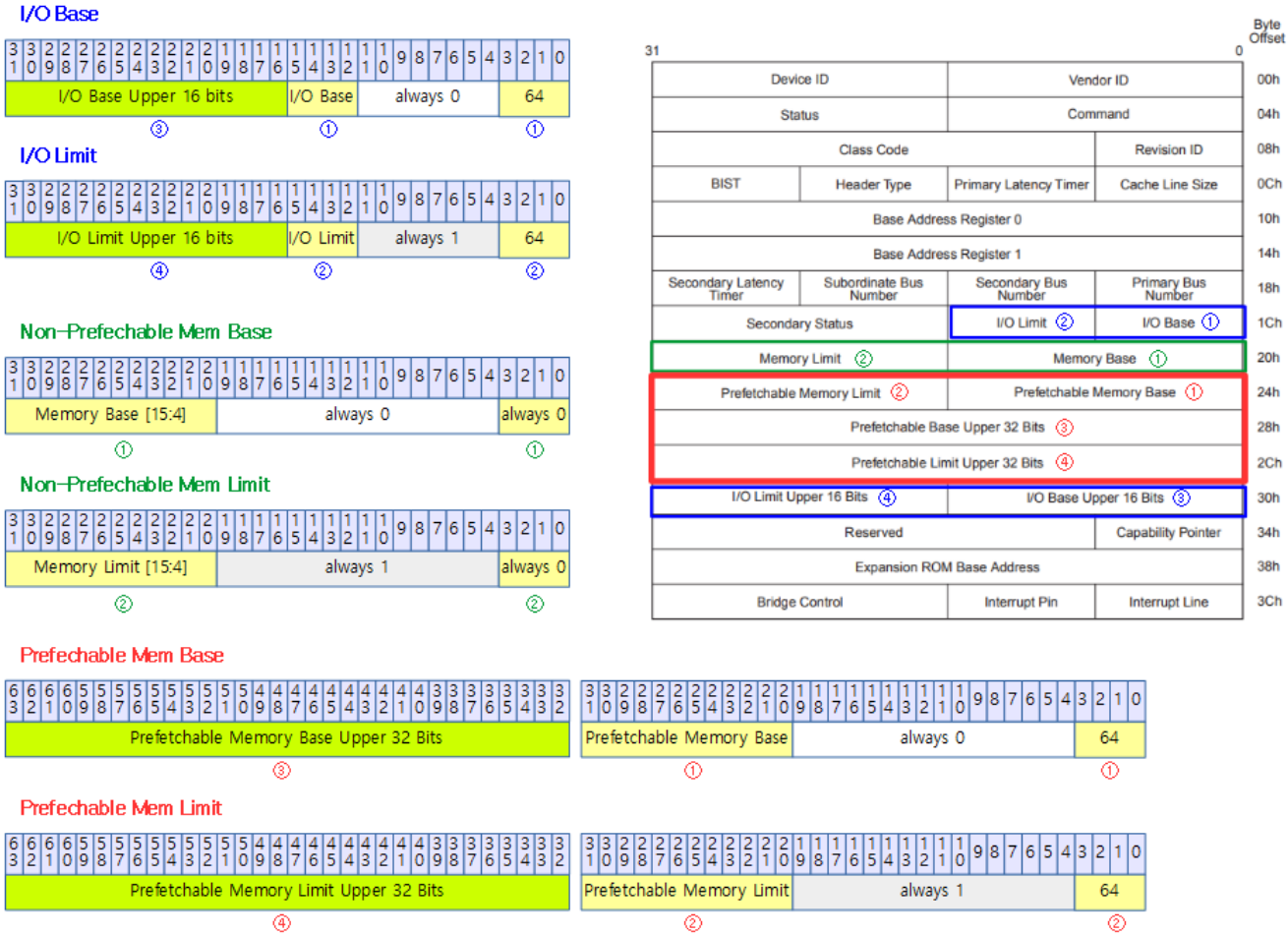
- e.g. dma-ranges = <0x43000000 0x0 0x0 0x0 0x0 0x100 0x0>;
  - The relocatable, pre-fetchable Mem64 region inbounds the PCI address and the host CPU physical address to the same space by 0x100_0000_0000 size (512G area).

PCI Address Zones and Limits on Root Complex PCIe Bridge Ports

In the configuration space of the Root Complex PCIe bridge port, you will see that the start addresses and limit ranges for the three types of memory mapping zones are specified.

- In the figure below, the value corresponding to number 1 is the lower 4 bits, and it is 32-bit and 64-bit.

    - 0b0000=32bit, 0b0001=64bit
- When constructing a base address, 1~2 values are concatenated according to each type, and the lower bits of several bits are not specified, but they are all filled with zeros. (always 0)
    - e.g. Pre-Fetchable Memory Base=0x4561, Pre-Fetchable Base Upper 32 bits=0x0000_1234
        - Base=0x0000_1234_4560_0000 (64bit)
- When configuring a limit address, 1~2 values are concatenated according to each type, and the lower bits of several bits are not specified, but they are all filled with zeros. (always 0)
    - e.g. Pre-Fetchable Memory Limit=0x4561, Pre-Fetchable Limit Upper 32 bits=0x0000_1234
        - Limit=0x0000_1234_456F_FFFF (64bit)



(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-59a.png)

PCI Legacy Interrupt Mapping

Represents the mapping configuration information of legacy interrupts used by PCI devices connected to the PCI bus. From this information, it is possible to determine which interrupt signal from INTA ~ INTD on each PCI bus is mapped to which interrupt number of the parent interrupt controller. For reference, if you are a PC that does not use the device tree, read the ACPI BIOS information and map it. In addition, PCI express does not have a legacy IRQ, but if you connect a PCI bridge to a PCI express port, you may need to configure legacy interrupt mapping information because PCI devices can be connected.

interrupt-map attribute

It consists of five parts:

- Child Unit Address
  - The #address-cells value uses 32-bit addresses equal to the number of cells specified.
  - The example below uses three 3-bit address values.
- Child Interrupt Specified Number
  - Specify the interrupt pin number 1~4.
- interrupt parent
  - The phandle value that corresponds to the parent interrupt controller node
- Parent Unit Address
  - The #address-cells value in the parent interrupt node uses 32-bit addresses equal to the number of cells specified.
    - Below is a single unit address.
- Parent interrupt number and trigger type
  - The #interrupt-cells value of the parent interrupt node uses a 32-bit value equal to the specified number of cells. (Usually 1~4 cells)
  - Below are two cells.
    - If you genericly use two cells, the first cell will specify the interrupt number, and the second will specify the interrupt type.
      - PCI uses IRQ_TYPE_LEVEL_HIGH for interrupt types.

interrupt-map-mask attribute

This is the mask information used to filter the valid value of the legacy interrupt value above, and consists of two parts:

- Mask for Child Unit Address
- Mask for child interrupt specified numbers

The cells in boxes 4 and 5 of the interrupt-map attribute below consist of the number of cells equal to the value of the parent interrupt controller's #interrupt-cells property.

- For reference, the GIC controller of the ARM64 consists of three cells.
  - The first cell is the irq-unit number, which uses GIC_SPI(0) or GIC_PPI(1).
  - The second cell is the interrupt number, which yields the hwirq number based on the value of the first argument.
    - If you use GIC_PPI(1), this value plus + 16 is the HWIRQ number.
    - If you use GIC_SPI(0), this value plus + 32 is the hwirq number.
  - The third cell specifies the interrupt trigger type.
    - IRQ_TYPE_NONE(0)
    - IRQ_TYPE_EDGE_RISING(1)
    - IRQ_TYPE_EDGE_FALLING(2)
    - IRQ_TYPE_LEVEL_HIGH(4)
    - IRQ_TYPE_LEVEL_LOW(8)

```
/*                PCI_DEVICE(#a)  INT#(#i)   IRQ-Controller   IRQ-UNIT#   IRQ_ARG(PI#i) */
interrupt-map = <  0x0 0x0 0x0    0x0          &gic             0x0        0x4 0x1
                  0x800 0x0 0x0   0x1          &gic             0x0        0x5 0x1
                 0x1000 0x0 0x0   0x2          &gic             0x0        0x6 0x1
                 0x1800 0x0 0x0   0x4          &gic             0x0        0x7 0x1>;

/*                      PCI_DEVICE(#a)    INT#(#i) */
interrupt-map-mask = <0xf800 0x0 0x0      0x7>;
```

(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-51a.png)

PCI Host Controller (PCIe RC) Implementation

Prepare the platform driver as follows. There is no particular difference between him and other bus drivers.

```
static struct platform_driver foo_pci_driver = {
        .driver = {
                .name = "foo-pci",
                .of_match_table = foo_pci_of_match,
        },
        .probe = foo_pci_probe,
        .remove = foo_pci_remove,
};
module_platform_driver(foo_pci_driver);
```

Specify the compatible attribute value so that it can be matched with the device tree node.

```
static const struct of_device_id foo_pci_of_match[] = {
        { .compatible = "company,foo" },
        { /* sentinel */ }
};
MODULE_DEVICE_TABLE(of, foo_pci_of_match);
```

foo_pcie Struct

The following shows a sample structure that can manage the PCIe Host Controller.

```
struct foo_pcie {
        struct device *dev;
        void __iomem *base;
        phys_addr_t base_addr;
        struct resource mem;
        struct pci_bus *root_bus;
        struct phy *phy;
        int (*map_irq)(const struct pci_dev *, u8, u8);
        bool need_ib_cfg;
        int nr_ib_regions;
};
```

- *dev
  - A device that points to a PCIe host controller
- *base
  - PCIe Host Controller Register Virtual Address base
- base_addr

    - PCIe Host Controller Register Physical Address base
- mem
    - Memory Resources
- *root_bus
    - PCI Root Bus
- *phy
    - PHY device that controls Serdes
- (*map_irq)
    - Hook function to get the interrupt number
- *msi
    - MSI Data

foo_pci_probe()

Learn the following sample driver code and what it means.

```
01 static int foo_pci_probe(struct platform_device *pdev)
02 {
03         struct device *dev = &pdev->dev;
04         struct foo_pcie *pcie;
05         struct device_node *np = dev->of_node;
06         struct resource reg;
07         resource_size_t iobase = 0;
08         LIST_HEAD(resources);
09         struct pci_host_bridge *bridge;
10         int ret;
11
12         bridge = devm_pci_alloc_host_bridge(dev, sizeof(*pcie));
13         if (!bridge)
14                 return -ENOMEM;
15
16         pcie = pci_host_bridge_priv(bridge);
17         platform_set_drvdata(pdev, pcie);
18         pcie->dev = dev;
19
20         ret = of_address_to_resource(np, 0, &reg);
21         if (ret < 0) {
22                 dev_err(dev, "unable to obtain controller resources\n");
23                 return ret;
24         }
25         pcie->base = devm_pci_remap_cfgspace(dev, reg.start,
26                                              resource_size(&reg));
27         if (!pcie->base) {
28                 dev_err(dev, "unable to map controller registers\n");
29                 return -ENOMEM;
30         }
31         pcie->base_addr = reg.start;
32
33         pcie->need_ib_cfg = of_property_read_bool(np, "dma-ranges");
34
35         /* PHY use is optional */
36         pcie->phy = devm_phy_get(dev, "pcie-phy");
37         if (IS_ERR(pcie->phy)) {
38                 if (PTR_ERR(pcie->phy) == -EPROBE_DEFER)
39                         return -EPROBE_DEFER;
40                 pcie->phy = NULL;
41         }
42
43         ret = of_pci_get_host_bridge_resources(np, 0, 0xff, &resources,
44                                                &iobase);
```

```
45          if (ret) {
46                  dev_err(dev, "unable to get PCI host bridge resources
    \n");
47                  return ret;
48          }
49
50          /* for legacy irq */
51          pcie->map_irq = of_irq_parse_and_map_pci;
52
53          ret = foo_pcie_setup(pcie, &resources);
54          if (ret) {
55                  dev_err(dev, "PCIe controller setup failed\n");
56                  pci_free_resource_list(&resources);
57                  return ret;
58          }
59
60          return 0;
61  }
```

This probe function drives the PCI host controller driver.

- In lines 12~14 of code, create a struct for the driver to be managed. Functions that start with devm_ are automatically released when the driver is unloaded.
  - e.g. struct pci_host_bridge and struct foo_pci two structs are created.
- In line 16~17 of the code, we get the struct foo_pci address that the bridge created above points to and store it for use as private driver data.
- Read the "reg" attribute in lines 20~24 of the code to find out as a resource.
- In lines 25~31 of code, map the PCIe host controller register resource to a virtual address, store the virtual address in the base member variable, and the physical address base_addr member variable.
- In line 33 of the code, check if there is a value for the "dma-ranges" property to determine if inbound mapping is required.
- In line 36~41 of the code, if the value of the phy-names property is "pcie-phy", the PHY handle corresponding to the node that the phandle value of the phys property points to is known.
- In lines 43~48 of code, we know the host bridge resource.
- In line 51 of code, specify a hook function that assigns the legacy irq used by the PCI device.
  - The of_property_match_string() function performs the following actions:
    - Read the PCI_INTERRUPT_PIN register (1 byte) of the PCI configuration space to find out the PCI pin number (0, 1~4).
    - Use the "interrupt-map" and "interrupt-mask" attributes of the device tree node to determine the hwirq corresponding to the PCI pin number.
    - Map hwirq and virq in irq domain.
    - Write the VIRQ to the PCI_INTERRUPT_LINE register (1 byte) in the PCI configuration space.
    - VIRQ.
- In code lines 53~58, perform the PCIe HW configuration to use the PCIe host controller.

foo_pcie_setup()

```
01  static int foo_pcie_setup(struct foo_pcie *pcie, struct list_head *res)
02  {
03          struct device *dev;
```

```
04          int ret;
05          void *sysdata;
06          struct pci_bus *child;
07          struct pci_host_bridge *host = pci_host_bridge_from_priv(pcie);
08
09          dev = pcie->dev;
10
11          ret = devm_request_pci_bus_resources(dev, res);
12          if (ret)
13                  return ret;
14
15          ret = phy_init(pcie->phy);
16          if (ret) {
17                  dev_err(dev, "unable to initialize PCIe PHY\n");
18                  return ret;
19          }
20
21          ret = phy_power_on(pcie->phy);
22          if (ret) {
23                  dev_err(dev, "unable to power on PCIe PHY\n");
24                  goto err_exit_phy;
25          }
26
27          /* pcie hw clock & regulator control */
28          /* ...(ignore)... */
29
30          /* pcie hw outbound map(ranges) control */
31          /* ...(ignore)... */
32
33          /* pcie hw inbound map(dma-ranges) control */
34          /* ...(ignore)... */
35          sysdata = pcie;
36
37          ret = foo_pcie_check_link(pcie);
38          if (ret) {
39                  dev_err(dev, "no PCIe EP device detected\n");
40                  goto err_power_off_phy;
41          }
42
43          /* pcie hw enable */
44          /* ...(ignore)... */
45
46          /* pcie msi related */
47          /* ...(ignore)... */
48
49          list_splice_init(res, &host->windows);
50          host->busnr = 0;
51          host->dev.parent = dev;
52          host->ops = &foo_pcie_ops;
53          host->sysdata = sysdata;
54          host->map_irq = pcie->map_irq;
55          host->swizzle_irq = pci_common_swizzle;
56
57          ret = pci_scan_root_bus_bridge(host);
58          if (ret < 0) {
59                  dev_err(dev, "failed to scan host: %d\n", ret);
60                  goto err_power_off_phy;
61          }
62
63          pci_assign_unassigned_bus_resources(host->bus);
64
65          pcie->root_bus = host->bus;
66
67          list_for_each_entry(child, &host->bus->children, node)
68                  pcie_bus_configure_settings(child);
69
70          pci_bus_add_devices(host->bus);
71
```

```
72          return 0;
73
74  err_power_off_phy:
75          phy_power_off(pcie->phy);
76  err_exit_phy:
77          phy_exit(pcie->phy);
78          return ret;
79  }
```

Perform the pcie hw configuration to use the pcie host controller (Root Complex).

- Lines 11~13 of the code request to use PCI resources. If a resource area already exists in the physical area, it returns an error.
  - You can check the address range of the requested resource with cat /proc/iomem.
  - It's a good idea to make a request to check for duplicate areas before ioremapping resources. It's also easy to manage via /proc/iomem.
- In code lines 15~19, initialize the PHY device connected to the transmission line of the PCI device.
  - The PHY is a device located at the end of the physical layer of the transmitting device and contains the following two parts:
    - The Physical Coding Sublayer (PCS) part responsible for coding
    - Physical Medium Dependent (PMD) parts connected with optical fiber cable or cooper cable
  - PHY is not used to communicate with chips that are connected to the immediate vicinity, but several technologies are used to communicate with devices that are slightly more than a few tens of centimeters away.
    - layer: Increases the signal voltage and current, modulates and encodes it, and outputs it.
    - Technologies such as differential are also used to remove noise during high-speed transmission.
    - It is mostly used for communication with almost all devices that are connected through slots, connectors, etc.
      - SDRAM, Flash Memory, SATA, PCIe, USB, Ethernet, Wifi, Wimax, ...
  - drivers/phy directory contains phy-related drivers.
- Turn on PHY on code lines 21~25.
- On line 28 of the code, perform the HW configuration related to the PCI host controller's clock and power supply (regulator).
  - Each PCI device has a different standard for operating registers, so you should refer to the chip's data sheet and programmer's manual.
- In line 31 of the code, set the registers for the PCI host controller's outbound mapping.
  - If the outbound mapping is provided fixed, it cannot be controlled.
- In line 34 of code, the PCI host controller sets the inbound mapping related registers for the connection of PCI devices that use DMA devices.
  - If the inbound mapping is provided fixed, it cannot be controlled.
- Check the link connection with the PCIe end point device connected to the pcie root complex in line 37~41. (Implementation functions omitted)
  - Note: iproc_pcie_check_link()
- In line 44 of code, set the register to enable the PCI host controller.
- On code line 47, prepare the MSI on the PCI host controller.

- After preparing the PCIe operation on code lines 49~61, scan the root bus bridge.
  - It scans subbuses and devices connected to this bus.
- In line 63 of the code,
- In code lines 67~68
- At line 70 of the code, register the PCI devices connected to the bus.

Specifying a PCI operation

```
1 static struct pci_ops foo_pcie_ops = {
2         .map_bus = foo_pcie_bus_map_cfg_bus,
3         .read = foo_pcie_config_read32,
4         .write = foo_pcie_config_write32,
5 };
```

- (*map_bus)
  - Specifies a hook function that sets the relevant registers of the PCI host controller so that it can be accessed from the PCI configuraton space corresponding to the requesting bus and devfn.
- (*read)
  - Specify a hook function to read data from the desired location to the PCI configuraton space corresponding to the requesting bus and devfn.
  - 1, 2, 4 bytes
- (*write)
  - Specify a hook function that writes data to the desired location in the PCI configuraton space corresponding to the requesting bus and devfn.
  - 1, 2, 4 bytes

Device Tree Case (for arm64)

1) arm/juno-base.dtsi

```
ranges = <0x01000000 0x00 0x00000000 0x00 0x5f800000 0x0 0x00800000>,  /* relocatab
le I/O */
         <0x02000000 0x00 0x50000000 0x00 0x50000000 0x0 0x08000000>,  /* relocatab
le MEM */
         <0x42000000 0x40 0x00000000 0x40 0x00000000 0x1 0x00000000>;  /* relocatab
le, pre-fetchable MEM */
interrupt-map-mask = <0 0 0 7>;
interrupt-map = <0 0 0 1 &gic 0 0 0 136 4>,
                <0 0 0 2 &gic 0 0 0 137 4>,
                <0 0 0 3 &gic 0 0 0 138 4>,
                <0 0 0 4 &gic 0 0 0 139 4>;
```

2) marvell/armada-37xx.dtsi

```
ranges = <0x82000000 0 0xe8000000   0 0xe8000000 0 0x1000000 /* non-relocatable MEM
*/
          0x81000000 0 0xe9000000   0 0xe9000000 0 0x10000>; /* non-relocatable I/O
*/
interrupt-map-mask = <0 0 0 7>;
interrupt-map = <0 0 0 1 &pcie_intc 0>,
                <0 0 0 2 &pcie_intc 1>,
                <0 0 0 3 &pcie_intc 2>,
                <0 0 0 4 &pcie_intc 3>;
```

3) marvell/armada-cp110-master.dtsi

```
ranges = <0x81000000 0 0xf9000000 0 0xf9000000 0 0x10000     /* non-relocatable I/O
*/
          0x82000000 0 0xf6000000 0 0xf6000000 0 0xf00000>;  /* non-relocatable, ME
M */
interrupt-map-mask = <0 0 0 0>;
interrupt-map = <0 0 0 0 &cpm_icu ICU_GRP_NSR 22 IRQ_TYPE_LEVEL_HIGH>;

ranges = <0x81000000 0 0xf9010000 0 0xf9010000 0 0x10000     /* non-relocatable I/O
*/
          0x82000000 0 0xf7000000 0 0xf7000000 0 0xf00000>;  /* non-relocatable MEM
*/
interrupt-map-mask = <0 0 0 0>;
interrupt-map = <0 0 0 0 &cpm_icu ICU_GRP_NSR 24 IRQ_TYPE_LEVEL_HIGH>;
```

...

4) amd/amd-seattle-soc.dtsi

```
ranges = <0x01000000 0x00 0x00000000 0x00 0xefff0000 0x00 0x00010000>,   /* relocat
able I/O (size=64K) */
         <0x02000000 0x00 0x40000000 0x00 0x40000000 0x00 0x80000000>,   /* relocat
able MEM (size=2G) */
         <0x03000000 0x01 0x00000000 0x01 0x00000000 0x7f 0x00000000>;   /* relocat
able MEM64 (size= 124G) */
interrupt-map-mask = <0xf800 0x0 0x0 0x7>;
interrupt-map = <0x1000 0x0 0x0 0x1 &gic0 0x0 0x0 0x0 0x120 0x1>,
                <0x1000 0x0 0x0 0x2 &gic0 0x0 0x0 0x0 0x121 0x1>,
                <0x1000 0x0 0x0 0x3 &gic0 0x0 0x0 0x0 0x122 0x1>,
                <0x1000 0x0 0x0 0x4 &gic0 0x0 0x0 0x0 0x123 0x1>;
dma-ranges = <0x43000000 0x0 0x0 0x0 0x0 0x100 0x0>;
```

5) freescale/fsl-ls1043a.dtsi

```
ranges = <0x81000000 0x0 0x00000000 0x40 0x00010000 0x0 0x00010000   /* non-relocat
able I/O */
          0x82000000 0x0 0x40000000 0x40 0x40000000 0x0 0x40000000>; /* non-relocat
able MEM */
interrupt-map-mask = <0 0 0 7>;
interrupt-map = <0000 0 0 1 &gic 0 110 0x4>,
                <0000 0 0 2 &gic 0 111 0x4>,
                <0000 0 0 3 &gic 0 112 0x4>,
                <0000 0 0 4 &gic 0 113 0x4>;
```

6) renesas/r8a7795.dtsi

```
ranges = <0x01000000 0 0x00000000 0 0xfe100000 0 0x00100000    /* relocatable I/O
*/
          0x02000000 0 0xfe200000 0 0xfe200000 0 0x00200000    /* relocatable MEM
*/
          0x02000000 0 0x30000000 0 0x30000000 0 0x08000000    /* relocatable MEM
*/
          0x42000000 0 0x38000000 0 0x38000000 0 0x08000000>;  /* relocatable, pre-
fetchable MEM */
interrupt-map-mask = <0 0 0 0>;
interrupt-map = <0 0 0 0 &gic GIC_SPI 116 IRQ_TYPE_LEVEL_HIGH>;
dma-ranges = <0x42000000 0 0x40000000 0 0x40000000 0 0x40000000>;
```

7) broadcom/northstar2/ns2.dtsi

```
ranges = <0x83000000 0 0x00000000 0 0x30000000 0 0x20000000>;   /* non-relocatable
MEM64 */
interrupt-map-mask = <0 0 0 0>;
interrupt-map = <0 0 0 0 &gic 0 GIC_SPI 305 IRQ_TYPE_NONE>;

brcm,pcie-ob;
brcm,pcie-ob-oarr-size;
brcm,pcie-ob-axi-offset = <0x30000000>;
brcm,pcie-ob-window-size = <256>;
```

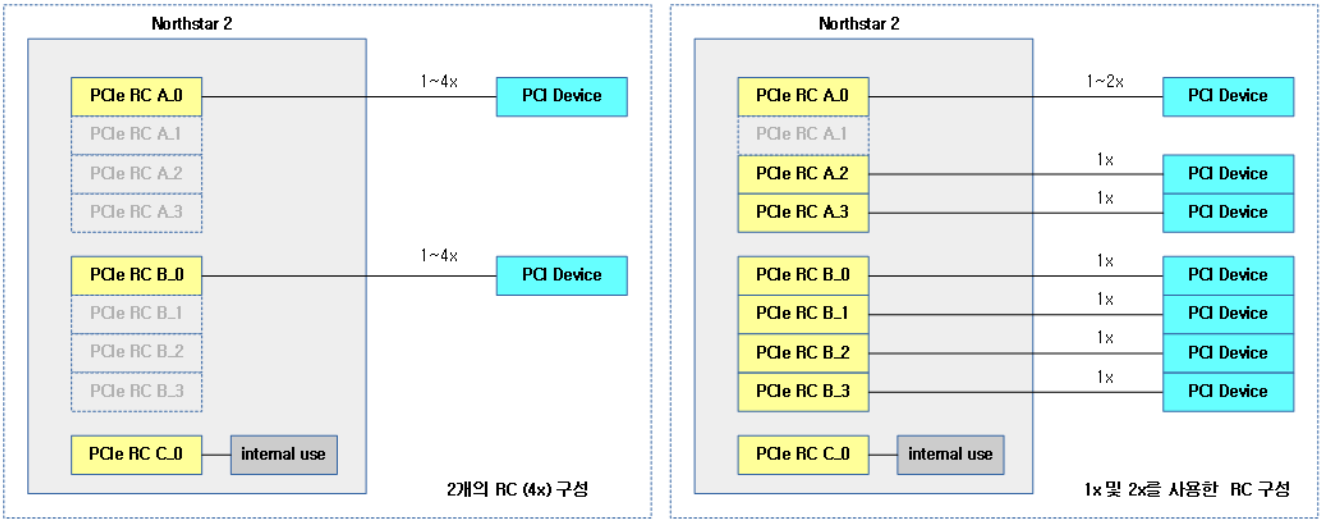
You can use the four northstar2-specific properties to make some changes to the outbound mapping address of the PCI address to the AXI bus.

- pcie-ob
  - Allows you to change the default outbound mapping of the PCIe Root Complex of ns2.
  - ns2 default Oubound mapping:
    - PCI Address: 0x0 ~ 0x1fffffff -> Host CPU Physical Address: 0x30000000 ~ 0x4fffffff (Size: 512MB)
- pcie-ob-oarr-size
  - Allows you to change the default outbound mapping size of the PCIe Root Complex of ns2.
  - ns2 default Oubound mapping size: 512MB
- pcie-ob-axi-offset
  - Specifies the offset of the host CPU physical address of the PCIe Root Complex in ns2.

    - In the example, the offset from the AXI physical address corresponding to the PCI address 0x0 is 0x30000000.
    - Note that the difference between the value of the ranges property and the CPU physical address minus this value is recorded in the OARR register.
        - CPU Physical Address (0x30000000) – pcie-ob-axi-offset attribute value (0x30000000) = 0x0 (reflected in OARR registers)
    - Note that the PCI address value of the ranges attribute is recorded in the OMAP register.
        - Reflect PCI address values (0x0) in OMAP registers
- pcie-ob-window-size
    - Specifies the size that the PCIe Root Complex of ns2 will use when changing the outbound mapping.
        - Only 128, 256, and 512 are available. (Unit: MB)

Here is a real-life example:

- NS2 allows you to change the PCIe root complex from the default 2 to up to 8 depending on the situation.
    - A port and B port can be configured as 1 RC using 4~1x lanes each. (default)
    - A port and B port can be configured as 1 RCs that use only 2~2x lanes each.
    - The A port and B port can be configured with 1 RC using 2~1x lanes and 1 RCs using 2x lanes, respectively.
    - The A and B ports can be configured as four RCs, each using only 1x lanes.



(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-58.png)

The example below shows a device tree setup using port B as a 1 RC configuration with only 4x lanes.

- The default outbound mapping is as follows: (Use 1~4x lanes)
    - ranges = <0x83000000 0 0x0 0 0x30000000 0 0x20000000>;
    - PCI Address: 0x0 ~ 0x1fffffff -> Host CPU Physical Address: 0x30000000 ~ 0x4fffffff (Size: 512MB)
- If you are using port B as 4 RCs, you can map the outbound by dividing the area by 128 MB as shown in the configuration below.

```
        pcie4: pcie@50020000 {
                compatible = "brcm,iproc-pcie";
                reg = <0 0x50020000 0 0x1000>;
                dma-coherent;

                #interrupt-cells = <1>;
                interrupt-map-mask = <0 0 0 0>;
                interrupt-map = <0 0 0 0 &gic 0 GIC_SPI 305 IRQ_TYPE_NONE>;

                linux,pci-domain = <4>;

                bus-range = <0x0 0xff>;

                #address-cells = <3>;
                #size-cells = <2>;
                device_type = "pci";
                ranges = <0x83000000 0 0x0 0 0x30000000 0 0x08000000>;

                brcm,pcie-ob;
                brcm,pcie-ob-oarr-size;
                brcm,pcie-ob-axi-offset = <0x30000000>;
                brcm,pcie-ob-window-size = <128>;

                phys = <&pci_phy1>;
                phy-names = "pcie-phy";

                msi-parent = <&v2m0>;
                brcm,pcie-num-lanes = <1>;
                status = "ok";
        };
```

```
        pcie6: pcie@50040000 {
                compatible = "brcm,iproc-pcie";
                reg = <0 0x50040000 0 0x1000>;
                dma-coherent;

                #interrupt-cells = <1>;
                interrupt-map-mask = <0 0 0 0>;
                interrupt-map = <0 0 0 0 &gic 0 GIC_SPI 317 IRQ_TYPE_NONE>;

                linux,pci-domain = <6>;

                bus-range = <0x00 0xff>;

                #address-cells = <3>;
                #size-cells = <2>;
                device_type = "pci";
                ranges = <0x83000000 0 0x10000000 0 0x40000000 0 0x08000000>;

                brcm,pcie-ob;
                brcm,pcie-ob-oarr-size;
                brcm,pcie-ob-axi-offset = <0x30000000>;
                brcm,pcie-ob-window-size = <128>;

                phys = <&pci_phy1>;
                phy-names = "pcie-phy";

                msi-parent = <&v2m0>;
                brcm,pcie-num-lanes = <1>;
                status = "ok";
        };
```

```
        pcie5: pcie@50030000 {
                compatible = "brcm,iproc-pcie";
                reg = <0 0x50030000 0 0x1000>;
                dma-coherent;

                #interrupt-cells = <1>;
                interrupt-map-mask = <0 0 0 0>;
                interrupt-map = <0 0 0 0 &gic 0 GIC_SPI 311 IRQ_TYPE_NONE>;

                linux,pci-domain = <5>;

                bus-range = <0x00 0xff>;

                #address-cells = <3>;
                #size-cells = <2>;
                device_type = "pci";
                ranges = <0x83000000 0 0x08000000 0 0x38000000 0 0x08000000>;

                brcm,pcie-ob;
                brcm,pcie-ob-oarr-size;
                brcm,pcie-ob-axi-offset = <0x30000000>;
                brcm,pcie-ob-window-size = <128>;

                phys = <&pci_phy1>;
                phy-names = "pcie-phy";

                msi-parent = <&v2m0>;
                brcm,pcie-num-lanes = <1>;
                status = "ok";
        };
```

```
        pcie7: pcie@50050000 {
                compatible = "brcm,iproc-pcie";
                reg = <0 0x50050000 0 0x1000>;
                dma-coherent;

                #interrupt-cells = <1>;
                interrupt-map-mask = <0 0 0 0>;
                interrupt-map = <0 0 0 0 &gic 0 GIC_SPI 323 IRQ_TYPE_NONE>;

                linux,pci-domain = <7>;

                bus-range = <0x00 0xff>;

                #address-cells = <3>;
                #size-cells = <2>;
                device_type = "pci";
                ranges = <0x83000000 0 0x18000000 0 0x48000000 0 0x08000000>;

                brcm,pcie-ob;
                brcm,pcie-ob-oarr-size;
                brcm,pcie-ob-axi-offset = <0x30000000>;
                brcm,pcie-ob-window-size = <128>;

                phys = <&pci_phy1>;
                phy-names = "pcie-phy";

                msi-parent = <&v2m0>;
                brcm,pcie-num-lanes = <1>;
                status = "ok";
        };
};
```

(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-57.png)

Let's take a look at how the address area of each port in the actual ns2 root complex is restricted.

- For reference, it is specified in 128M increments, but the actual PCI address area of the RC is limited to 20MB and 28M.

```
# lspci -s 0000:00:00.0 -x
0000:00:00.0 Class 0604: Device 14e4:d712 (rev 11)
00: e4 14 12 d7 06 00 10 00 11 00 04 06 00 00 01 00
10: 00 00 00 00 00 00 00 00 00 01 01 00 00 00 00 00
20: f0 ff 00 00 01 00 31 01 00 00 00 00 00 00 00 00
30: 00 00 00 00 48 00 00 00 00 00 00 00 81 01 00 00

# lspci -s 0004:00:00.0 -x
0004:00:00.0 Class 0604: Device 14e4:d712 (rev 11)
00: e4 14 12 d7 06 00 10 00 11 00 04 06 00 00 01 00
10: 00 00 00 00 00 00 00 00 00 01 01 00 00 00 00 00
20: f0 ff 00 00 01 00 b1 01 00 00 00 00 00 00 00 00
30: 00 00 00 00 48 00 00 00 00 00 00 00 82 01 00 00

# lspci -s 0005:00:00.0 -x
0005:00:00.0 Class 0604: Device 14e4:d712 (rev 11)
00: e4 14 12 d7 06 00 10 00 11 00 04 06 00 00 01 00
10: 00 00 00 00 00 00 00 00 00 01 01 00 00 00 00 00
20: f0 ff 00 00 01 08 b1 09 00 00 00 00 00 00 00 00
30: 00 00 00 00 48 00 00 00 00 00 00 00 83 01 00 00

# lspci -s 0006:00:00.0 -x
0006:00:00.0 Class 0604: Device 14e4:d712 (rev 11)
00: e4 14 12 d7 06 00 10 00 11 00 04 06 00 00 01 00
10: 00 00 00 00 00 00 00 00 00 01 01 00 00 00 00 00
20: f0 ff 00 00 01 10 b1 11 00 00 00 00 00 00 00 00
30: 00 00 00 00 48 00 00 00 00 00 00 00 84 01 00 00

# lspci -s 0007:00:00.0 -x
0007:00:00.0 Class 0604: Device 14e4:d712 (rev 11)
00: e4 14 12 d7 06 00 10 00 11 00 04 06 00 00 01 00
10: 00 00 00 00 00 00 00 00 00 01 01 00 00 00 00 00
20: f0 ff 00 00 01 18 b1 19 00 00 00 00 00 00 00 00
30: 00 00 00 00 48 00 00 00 00 00 00 00 85 01 00 00
```

I/O		Non-Prefechable Mem		Prefechable Mem	
Base	Limit	Base	Limit	Base	Limit
0x00000000	0x00000fff (1KB)	0xfff00000	0x000fffff (1MB)	0x00000000_00000000	0x00000000_013fffff (20MB)
0x00000000	0x00000fff (1KB)	0xfff00000	0x000fffff (1MB)	0x00000000_00000000	0x00000000_01bfffff (28MB)
0x00000000	0x00000fff (1KB)	0xfff00000	0x000fffff (1MB)	0x00000000_08000000	0x00000000_09bfffff (28MB)
0x00000000	0x00000fff (1KB)	0xfff00000	0x000fffff (1MB)	0x00000000_10000000	0x00000000_11bfffff (28MB)
0x00000000	0x00000fff (1KB)	0xfff00000	0x000fffff (1MB)	0x00000000_18000000	0x00000000_19bfffff (28MB)

(http://jake.dothome.co.kr/wp-content/uploads/2018/10/pci-60a.png)

8) cavium/thunder2-99xx.dtsi

```
ranges = <0x02000000    0 0x40000000    0 0x40000000    0 0x20000000       /* reloc
atable MEM */
          0x43000000 0x40 0x00000000 0x40 0x00000000 0x20 0x00000000>;     /* reloc
atable, pre-fetchable MEM64 */
interrupt-map-mask = <0 0 0 7>;
interrupt-map = /* addr  pin  ic   icaddr            icintr */
                 <0 0 0  1  &gic   0 0    GIC_SPI 0 IRQ_TYPE_LEVEL_HIGH
                  0 0 0  2  &gic   0 0    GIC_SPI 1 IRQ_TYPE_LEVEL_HIGH
                  0 0 0  3  &gic   0 0    GIC_SPI 2 IRQ_TYPE_LEVEL_HIGH
                  0 0 0  4  &gic   0 0    GIC_SPI 3 IRQ_TYPE_LEVEL_HIGH>;
```

9) nvidia/tegra132.dtsi

```
ranges = <0x82000000 0 0x01000000 0x0 0x01000000 0 0x00001000   /* non-relocatable
MEM (port 0 cfg space) */
          0x82000000 0 0x01001000 0x0 0x01001000 0 0x00001000   /* non-relocatable
MEM (port 1 cfg space) */
          0x81000000 0 0x0        0x0 0x12000000 0 0x00010000   /* non-relocatable
I/O (64 KiB) */
          0x82000000 0 0x13000000 0x0 0x13000000 0 0x0d000000   /* non-relocatable
MEM (208 MiB) */
          0xc2000000 0 0x20000000 0x0 0x20000000 0 0x20000000>; /* non-relocatable,
pre-fetchable MEM (512 MiB) */
interrupt-map-mask = <0 0 0 0>;
interrupt-map = <0 0 0 0 &gic GIC_SPI 98 IRQ_TYPE_LEVEL_HIGH>;

pci@1,0 {
        assigned-addresses = <0x82000800 0 0x01000000 0 0x1000>; /* non-relocatable
MEM (4K) */
        ...
};

pci@2,0 {
        assigned-addresses = <0x82001000 0 0x01001000 0 0x1000>; /* non-relocatable
MEM (4K) */
        ...
};
```

You can grant a range for a subbus.

consultation

- PCI Subsystem -1- (Basic) (http://jake.dothome.co.kr/pci-1) | Qc
- PCI Subsystem -2- (Core) (http://jake.dothome.co.kr/pci-2) | Qc
- PCI Subsystem -3- (Host Controller) (http://jake.dothome.co.kr/pci-3) | Question C – Current Article

- PCI Bus Binding to: IEEE Std 1275-1994 | www.devicetree.org – Download PDF (https://www.devicetree.org/open-firmware/bindings/pci/pci2_1.pdf)
- Open Firmware Recommended Practice: Interrupt Mapping | www.devicetree.org – Download PDF (http://www.devicetree.org/open-firmware/practice/imap/imap0_9d.pdf)

LEAVE A COMMENT

Your email will not be published. Required fields are marked with *

Comments

name *

email *

Website

WRITE A COMMENT

PCI Subsystem -2- (Core) (http://jake.dothome.co.kr/pci-2/)

SPI Subsystem -1- (Basic) (http://jake.dothome.co.kr/spi-1/)

Munc Blog (2015 ~ 2023)