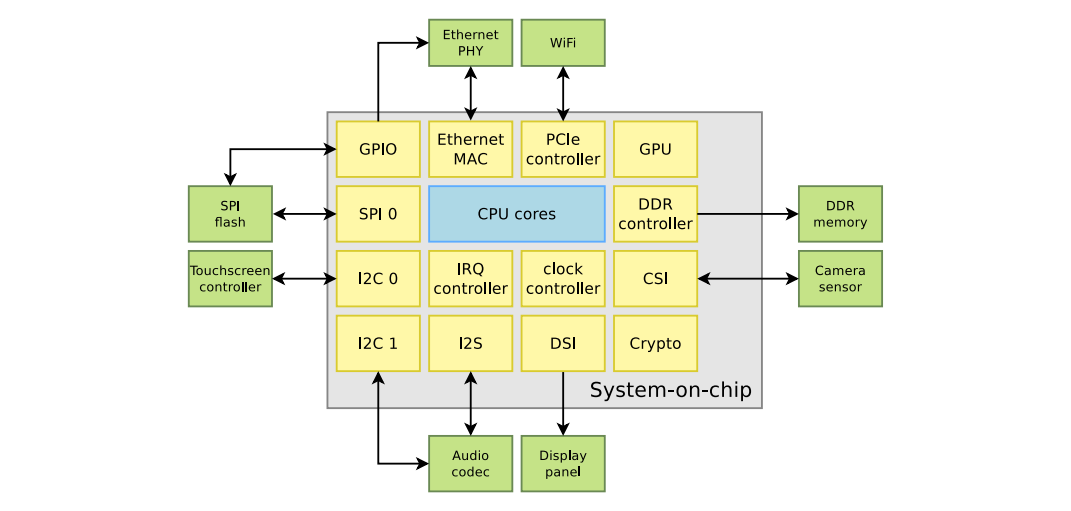
**Device tree**



**System-on-a-Chip (SoC)**

The image depicts a block diagram of a System-on-a-Chip (SoC) system. This system comprises the following components:

1. **Central Processing Unit (CPU):** This is the central processing unit of the computer, responsible for executing all the programs that the computer needs to run.
2. **Memory:** It includes DDR (Double Data Rate) memory and memory controller. DDR memory stores data and programs currently being used by the CPU.
3. **Interfaces:**
   * Ethernet: Enables wired network connections.
   * WiFi: Allows wireless network connections.
   * PCIe: Permits connection to high-speed peripheral devices such as graphics cards.
   * GPIO (General Purpose Input/Output): Allows control of simple peripheral devices like LEDs or buttons.
   * SPI (Serial Peripheral Interface): Facilitates connection to low-speed peripheral devices such as sensors.
   * I2C (Inter-Integrated Circuit): Enables connection to low-speed peripheral devices such as flash memory.
   * Touchscreen: Enables control of the computer via a touch-sensitive screen.
   * Camera: Allows capturing images and recording videos.
   * Display: Enables the display of images and videos.
4. **Cryptographic Engine (Crypto):** Provides security to the system by encrypting data.
5. **Controllers:**
   * LCD Controller: Enables control of the LCD screen.
   * Audio Controller: Enables audio playback and recording.
6. **Other Components:**
   * Clock: Provides clock signals for the system.
   * Sensors: Provide information about the surrounding environment to the system.

**Discoverable and non-discoverable hardware**

**Some hardware busses provide discoverability mechanisms**

* E.g: PCI(e), USB
* One does not need to know ahead of time what will be connected on these busses
* Devices can be enumerated and identified at runtime

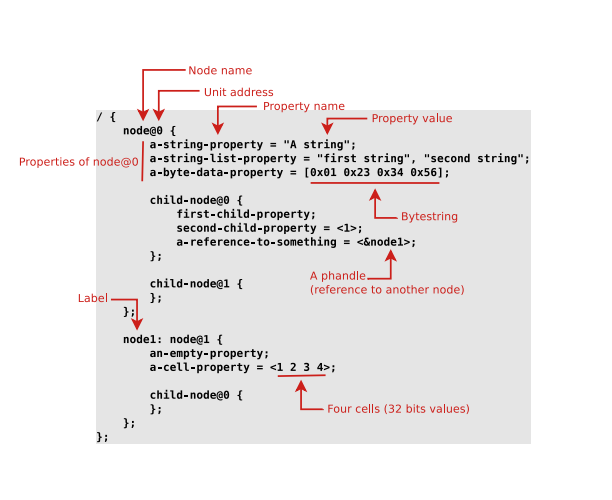
**But many hardware busses do not provide discoverability mechanisms**

* E.g: I2C, SPI, 1-wire, memory-mapped, etc.
* One needs to know what is connected on those busses, and how they are connected to the rest of the system
* Embedded systems typically make extensive use of such busses

With non-discoverable hardware we have 3 methods to describe:

* Directly in the OS/bootloader code, using compiled data structures, typically in C
* Using ACPI tables
* ‘device tree’ (good choice for embedded system) is to describe what and how that hardware connect to system including hardware info, connection method and other resources. (strength of device tree is modularity, flexibility, cross-platform support)

**Device tree syntax**



<https://beanredarmy.github.io/2018-12-01-Device-Tree/>

Some data types used in DT include:

* String, enclosed in quotes. We can use commas to create a list of strings.
* 32-bit positive integer, enclosed in curly braces.
* Boolean data, which is an empty property. If it is declared in the device tree, the value is true, if not declared, it is false.
* **Naming convention**
* Each node is represented as <name>[@address], where <name>is a string up to 31 characters long, and [@address]is the address used to access the node. <@address>May or may not. For example:

expander@20 {

compatible = "microchip,mcp23017";

reg = <20>;

[...]

};

or

i2c@021a0000 {

compatible = "fsl,imx6q-i2c", "fsl,imx21-i2c";

reg = <0x021a0000 0x4000>;

[...]

};

* In addition, the label may or may not be present. We only need to label a node if we intend to reference it as an attribute of another node.

#### Alias, lable and and phhandle

The device tree below explains how they work:

aliases {

ethernet0 = &fec;

gpio0 = &gpio1;

gpio1 = &gpio2;

mmc0 = &usdhc1;

[...]

};

gpio1: gpio@0209c000 {

compatible = "fsl,imx6q-gpio", "fsl,imx35-gpio";

[...]

};

node\_label: nodename@reg {

[...];

gpios = <&gpio1 7 GPIO\_ACTIVE\_HIGH>;

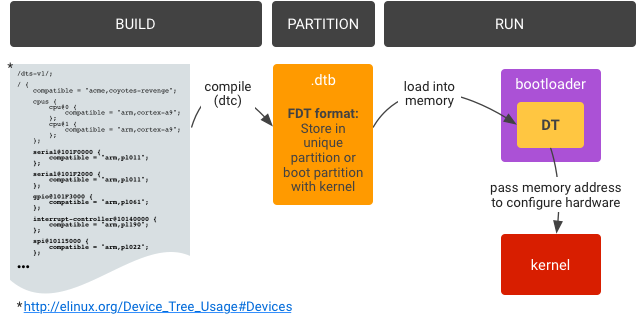
};

Lable is a way to identify a node with a unique name. In fact, this name will be converted to a single 32-bit value by the DT compiler. In the example above, gpio1and node\_labelare labels. Because the label is unique to a node, it is used to designate the node.

Phandle (pointer handle) is a 32-bit value associated with a node, and is used to identify a node so that it can be referenced from an attribute of another node. Using <&mylabel>, we can point to a node named mylabel. In the above example, &gpio1is a phhandle pointing to node gpio1.

In order for the kernel to traverse the entire tree to find a node, the concept of alias was born. Alias ​​is not used directly in DT but is used by the Linux kernel. We can use the function find\_node\_by\_alias()and pass in the alias to find a node.

**Device tree compiler (DT Compiler)**



Device tree has two main forms: source code as we know, also known as DTS and has the suffix .dts; The compiled form is also known as DTB and has the suffix .dtb. In addition, the source code can also have the suffix .dtsi, representing the DT at the SoC level, and be included in a file .dts(Like the C source code .cinclude header .h).

To compile the source file .dtsto a .dtb, we use DT compiler (dtc).

**Address the device.**

Each device is assigned to at least one node on the DT. There are some common properties that all types of devices have, especially devices located on the bus that the kernel knows about (like SPI, I2C, platform, MDIO...). Those attributes are reg, #address-cellsand #size-cells. The purpose of these three attributes is to assign their addresses on the bus. Where the address attribute is reg. #size-cellsIndicates how many 32-bit cells are used to represent the device's address size. #address-cellsIndicates how many 32-bit cells are used to represent the address.

The attribute regis a list of value pairs of the form reg = <adress0 size0 [address1 size1] [address2 size2] ...>, each pair of address and cell values ​​represents the address range used by the device.

Devices will inherit #size-cellsfrom #address-cellsparent nodes (usually nodes representing bus controllers). In fact, these two parameters do not affect the node that declares it, but only affect its child nodes. That means, before declaring the properties regof a node, you must know the two properties #address-cellsof #size-cellsthe parent node.

Example: #address-cells and #size-cells is <3> và <6> means we have 3 cells for address and 6 cell for size

Example 2:

In most cases the reg property describes the offset and the quantity of the memory or the device mapped. So

reg = <0x20000000 0x80000000>

means that: "Starting from the register 0x20000000, next 0x80000000 bytes are mapped for this device".

#address-cells specifies the number of 32 bit registers that will be required to specify the starting address. Since, 0x20000000 is a 32 bit value,

#address-cells = <1>;

Similarly, #size-cells specifies the number of bytes required to encode the size field. Since 0x80000000 is a u32 value, #size-cells = <1>;

Do refer the website: [devicetree.org](https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.4)

**Platform device address**

reg = <[base0 length0] [base1 length1] [address2 length2] ... >

List of address that we can access to that device, base is starting address and length is the length between starting addr

**Resources are named**

Driver can do : access register, interrupt control, interrupt handler

<https://devicetree-specification.readthedocs.io/en/stable/devicetree-basics.html>

**Generic name**

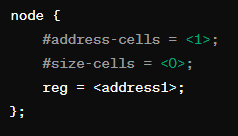
|  |  |  |
| --- | --- | --- |
| * adc * accelerometer * atm * audio-codec * audio-controller * backlight * bluetooth * bus * cache-controller * camera * can * charger * clock * clock-controller * compact-flash * cpu * cpus * crypto * disk * display * dma-controller * dsi * dsp * eeprom * efuse * endpoint * ethernet * ethernet-phy * fdc * flash * gnss * gpio | * gpu * gyrometer * hdmi * hwlock * i2c * i2c-mux * ide * interrupt-controller * iommu * isa * keyboard * key * keys * lcd-controller * led * leds * led-controller * light-sensor * magnetometer * mailbox * mdio * memory * memory-controller * mmc * mmc-slot * mouse * nand-controller * nvram * oscillator * parallel * pc-card * pci | * pcie * phy * pinctrl * pmic * pmu * port * ports * power-monitor * pwm * regulator * reset-controller * rng * rtc * sata * scsi * serial * sound * spi * sram-controller * ssi-controller * syscon * temperature-sensor * timer * touchscreen * tpm * usb * usb-hub * usb-phy * video-codec * vme * watchdog * wifi |

**Size and address cell**

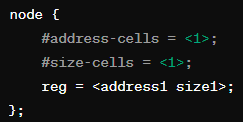
The #address-cells and #size-cells properties may be used in any device node that has children in the devicetree hierarchy and describes how child device nodes should be addressed.

[**https://stackoverflow.com/questions/74732961/incorrect-values-for-address-cells-and-size-cells-in-device-tree**](https://stackoverflow.com/questions/74732961/incorrect-values-for-address-cells-and-size-cells-in-device-tree)

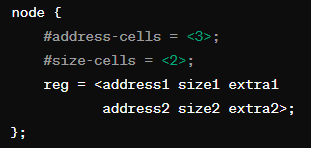
When #address-cells = <1>; and #size-cells = <0>;



When #address-cells = <1>; and #size-cells = <1>;



When #address-cells = <3>; and #size-cells = <2>;



<https://www.nxp.com/docs/en/application-note/AN5125.pdf>

<https://elinux.org/images/f/f9/Petazzoni-device-tree-dummies_0.pdf>

<https://bootlin.com/pub/conferences/2021/webinar/petazzoni-device-tree-101/petazzoni-device-tree-101.pdf>

Ex2