Device Tree:

What's the Device tree?

- **The device tree is just a way to describe the HW components of computer system, It's a static data not an executable code**

Introducing device trees

If you are working with Arm or PowerPC SoCs, you are almost certainly going to encounter device trees at some point. This section aims to give you a quick overview of what they are and how they work. We will revisit the topic of device trees repeatedly throughout the course of this book.

A device tree is a flexible way of defining the hardware components of a computer system. Bear in mind that a device tree is just static data, not executable code. Usually, the device tree is loaded by the bootloader and passed to the kernel, although it is possible to bundle the device tree with the kernel image itself to cater for bootloaders that are not capable of loading them separately.

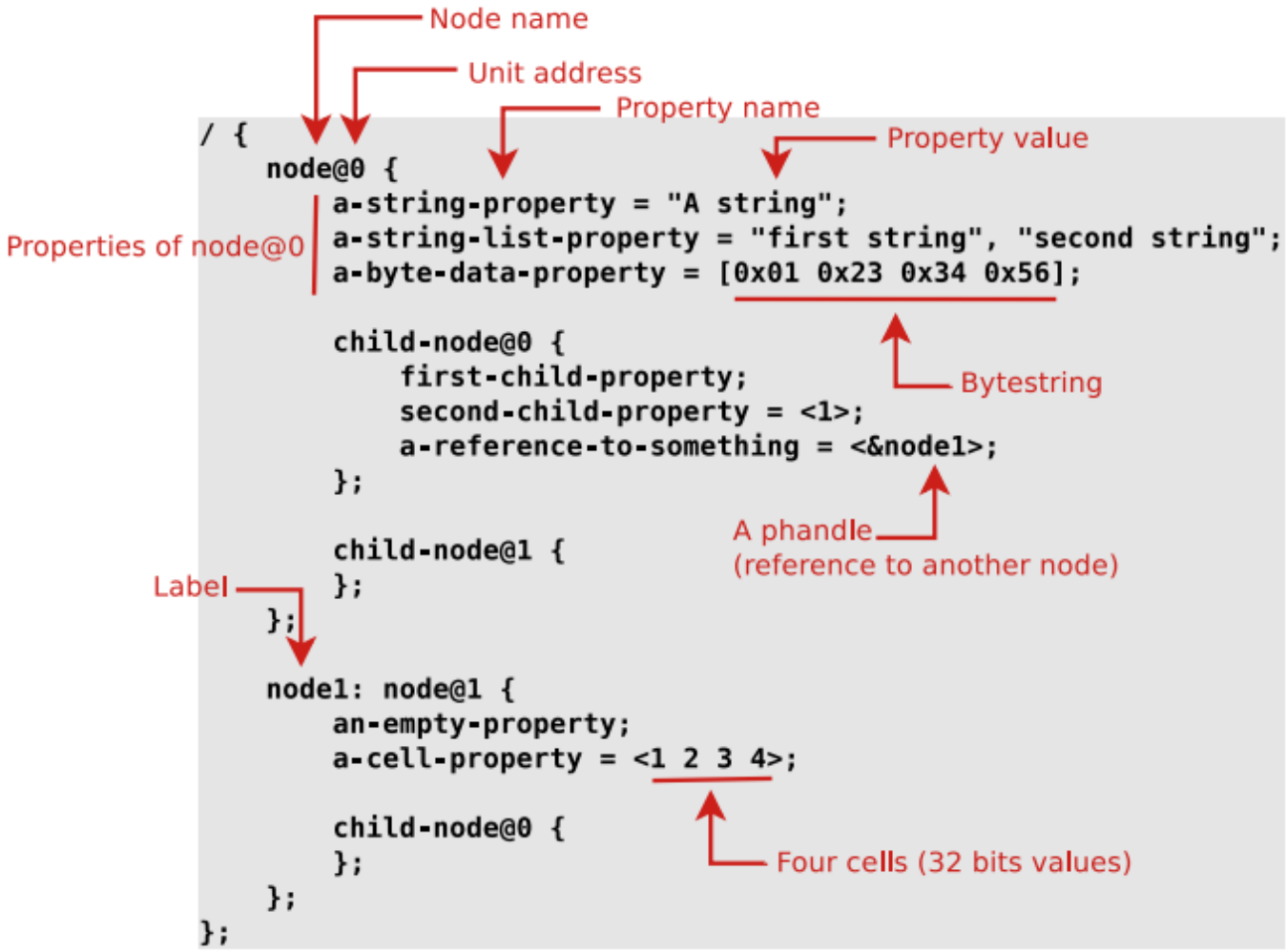
Where is the device tree located?

- **The Device tree is stored in the bootloader's firmware and then it's passed from the bootloader to the kernel.**
- **Or it can be stored in the kernel image itself.**

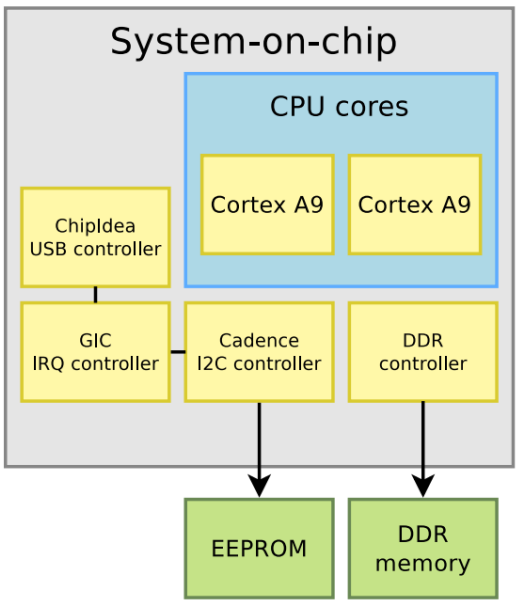
- A tree data structure describing the hardware is written by a developer in a *Device Tree Source* file, `.dts`
- Gets compiled to a more efficient *Device Tree Blob* representation, `.dtb` by the *Device Tree Compiler*, `dtc`
- The resulting `.dtb` accurately describes the hardware platform in an **OS-agnostic** way and:
  - Can be **linked directly** inside a bootloader binary (U-Boot, Barebox)
  - Can be **passed** to the operating system by the bootloader (Linux)
  - U-Boot: `bootz <kernel-addr> - <dtb-addr>`

How the Device tree code is written ?

- **It consits of nodes and under each node there are properties**
- **The nodes are the HW components like the CPU, the memory, the peripherals**
- **It starts with the root node and it's written in the form of `/dts-v1/`**
- **The properties are written in the form of `key = value;`**
- **Every node represents a device or IP block**
- **Note that the CPU and the memory**



Real Example :



- **Note that the CPU has a property called `Compatible` this specifies the hardware device driver,it makes the kernel identify the CPU model and load the appropriate drivers and firmware.**

The reg property:

- **The reg property is used to specify the address and the size of the memory mapped registers of the device.**
- The memory and cpu nodes shown earlier have a reg property, which refers to a range of units in a register space. A reg property consists of two values representing the real physical address and the size (length) of the range. Both are written as zero or more 32-bit integers, called cells. Hence, the previous memory node refers to a single bank of memory that begins at 0x80000000 and is 0x20000000 bytes long.
- **The first value is the address and the second value is the size.**
- ```
  reg = <0x3F000000 0x100000>; //This means a single bank of memory
  that begins at 0x3F000000 and is 0x100000 bytes long.
  ```
- **But what if we want to make more that one bank memory ? We will have to assign values to `#address_cells` and `#size_cells`**
- ```
  /{
      #address-cells = <2>;
      #size-cells = <2>;
      memory@80000000 {
          device_type = "memory";
          reg = <0x00000000 0x80000000 0 0x80000000>;
      };
  };
  ```

labels:

- **The labels are used to refer to the nodes in the device tree(Every node might has a label), we use it if we want to make connections between the components**
- **We can reference the node from another node by the label**
- **Lables can be named `Phandles`**

```
    intc: interrupt-controller@48200000 {
        compatible = "ti,am33xx-intc";
        interrupt-controller;
        #interrupt-cells = <1>;
        reg = <0x48200000 0x1000>;
    };
    lcdc: lcdc@4830e000 {
        compatible = "ti,am33xx-tilcdc";
        reg = <0x4830e000 0x1000>;
        interrupt-parent = <&intc>;
        interrupts = <36>;
        ti,hwmods = "lcdc";
        status = "disabled";
    };
};
```

Includes in Device tree :

- **The device tree can be splitted out into sections using the include files**
- **There's a file with `.dtsi` extention, this is the include file of the device tree**
- **This include can be done with the normal include in C `#include "file.dtsi"`**

```
#include "am33xx.dtsi"
#include "am335x-bone-common.dtsi"
```

Compiling the Device tree :

- **The bootloader and the kernel needs a binary file to deal with so we have to compile the `.dtc` file to generate a binary called `.dtb`**
- **The figure below shows , we will use the `dtsi` header file and the `dts` file to generate the `.dtb` file , Very easy right 😃 ?**

Definition of the AM33xx SoC

```
/ {
     compatible = "ti,am33xx";
     [...]
     ocp {
          uart0: serial@44e09000 {
              compatible = "ti,omap3-uart";
              reg = <0x44e09000 0x2000>;
              interrupts = <72>;
              status = "disabled";
          };
     };
};
```

am33xx.dtsi

+

Definition of the BeagleBone board

```
#include "am33xx.dtsi"

/ {
     compatible = "ti,am335x-bone", "ti,am33xx";
     [...]
     ocp {
          uart0: serial@44e09000 {
              pinctrl-names = "default";
              pinctrl-0 = <&uart0_pins>;
              status = "okay";
          };
     };
};
```

am335x-bone.dts

=

Compiled DTB

```
/ {
     compatible = "ti,am335x-bone", "ti,am33xx";
     [...]
     ocp {
          uart0: serial@44e09000 {
              compatible = "ti,omap3-uart";
              reg = <0x44e09000 0x2000>;
              interrupts = <72>;
              pinctrl-names = "default";
              pinctrl-0 = <&uart0_pins>;
              status = "okay";
          };
     };
};
```

am335x-bone.dtb

Note: the real DTB is in binary format. Here we show the text equivalent of the DTB contents;