# DEVICE TREE

-easy to read hardware description file.

-JSON-like formatting

Properties can be empty or in the form of key-value pairs.

CONFIG\_OF=y

DT API headers.

<linux/of.h>

<linux/of\_device.h>

node\_label: nodename@reg{

string-property = “a string”;

string-list = “str1” “str2”

one-int-property = <197>;

int-list-property = <0xaf 0xbe 121>;

mixed-list-property = “string1” , <0xaf> , <35>, [0x01 0x21 0x41]

byte-array-property = [0x01 0x23 0x45 0x67];

boolean-property;

};

Cells - 32-bit unsigned integers delimited by angle brackets

Boolean data is empty property. If property is there, its true. Otherwise false.

Every node must have a name in the form <name>[@address] - upto 31 characters long.

<address> - primary address used to access the device.

Eg:

i2c@021a0000 {

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

reg = <0x021a0000 0x4000>;

[...]

};

**Label** - labelling a node is useful when a node is referenced from property of another node.

Label is a pointer to a node.

Eg:

gpio1: gpio@0209c000 {

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

[...]

};

A **phandle** is a 32-bit value associated with a node that is used to uniquely identify that node.

By using <&mylabel>, you point to the node whose label is mylabel.

**Aliases**

Aliases are like lookup table.

find\_node\_by\_alias() can be used to find a node given its alias. The aliases are dereferenced in the linux kernel.

Eg:

aliases {

ethernet0 = &fec;

gpio0 = &gpio1;

gpio1 = &gpio2;

mmc0 = &usdhc1;

[...]

};

**DT Compiler**

DTS - textual - sources (.dts) (.dtsi)

DTB - binary blob (.dtb)

Run-time representation of Device Tree in /proc/device-tree. CONFIG\_PROC\_DEVICETREE - to expose DT to user space.

DTC - Device Tree Compiler

For compiling, give:

ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- make imx6dl-sabrelite.dtb

To extract source (.dts) from binary (.dtb), give:

dtc -I dtb -O dts arch/arm/boot/dts imx6dl-sabrelite.dtb >path/to/my\_devicetree.dts

**Addressing Devices**

Each device - atleast one node.

Common properties:

reg

#address-cells

#size-cells

Device addressing on the bus they sit on.

Each addressable device gets a <reg> property.

reg = <address0 size0> [address1 size1] [address2 size2]..

#size-cells indicates how many 32-bit cells are used to represent size.

#address-cells indicates how many 32-bit cells are used to represent the address.

The presence of #size-cells and #address-cells in a given device does not affect the device itself, but its children.

SPI and I2C devices are not memory mapped.

The bus controller driver will perform indirect access on behalf of the CPU.

For spi and i2c nodes, #size-cells is 0(zero) and #address-cells is 1(one).

For I2C device, <reg> property is used to specify the device’s address on the bus.

&i2c3 {

[...]

status = "okay";

temperature-sensor@49 {

compatible = "national,lm73";

reg = <0x49>;

};

pcf8523: rtc@68 {

compatible = "nxp,pcf8523";

reg = <0x68>;

};

};

For SPI device, <reg> property represents the index of the chip select line assigned to the device from all the chip-select lines available to the controller.

&ecspi1 {

fsl,spi-num-chipselects = <3>;

cs-gpios = <&gpio5 17 0>, <&gpio5 17 0>, <&gpio5 17 0>;

status = "okay";

[...]

ad7606r8\_0: ad7606r8@1 {

compatible = "ad7606-8";

reg = <1>;

spi-max-frequency = <1000000>;

interrupt-parent = <&gpio4>;

interrupts = <30 0x0>;

convst-gpio = <&gpio6 18 0>;

};

};

I2C/SPI devices should be declared in a board-level file (.dts) while I2C/SPI bus controllers must be declared in SoC level file (.dtsi).

**Platform Device Addressing**

Memory is accessible to the CPU (memory-mapped devices).

<reg> property defines the list of memory regions on which you can access the device.

Each region is represented by a tuple of cells where:

- first cell is the base address.

- second cell is the size of the region.

reg = <base0 length0> [base1 length1] [base2 length2]…

These type of devices should be declared within a node with a special value,

compatible= “simple-bus”

meaning no specific driver.

Eg of a device node - iMX6’s UART device node:

uart1: serial@02020000 {

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

reg = <0x02020000 0x4000>;

interrupts = <0 26 IRQ\_TYPE\_LEVEL\_HIGH>;

clocks = <&clks IMX6QDL\_CLK\_UART\_IPG>,

<&clks IMX6QDL\_CLK\_UART\_SERIAL>;

clock-names = "ipg", "per";

dmas = <&sdma 25 4 0>, <&sdma 26 4 0>;

dma-names = "rx", "tx";

status = "disabled";

};

Named resources:

reg-names

clock-names

interrupt-names

dma-names

So that a given name will always match the resource.

In code, the resource can be obtained from name using the function:

struct resource \*res1;

res1=platform\_get\_resource\_byname(pdev, IORESOURCE\_MEM, “name”);

In the device driver code, the driver will take ownership of the memory region and map it into virtual address space.

Eg:

struct resource \*res1;

void \_\_iomem \*base;

res1 = platform\_get\_resource(pdev,IORESOURCE\_MEM,0);

base = devm\_ioremap\_resource(&pdev->dev,res);

platform\_get\_resource() will set the start and end fields of struct res according to the memory region present in the first (index 0) reg assignment.

**Handling Interrupts**

Four properties are used to describe interrupt connections in DT:

On the controller side:

interrupt-controller

#interrupt-cells - how many cells are used to specify an interrupt for that controller

On the consumer side:

interrupt-parent - contains a phandler pointer to the controller

interrupts

Interrupt binding and interrupt specifiers are tied to the interrupt controller device.

Interrupt handling

-fetch the IRQ number from DT and map to Linux IRQ, thus registering a callback function for it.

int irq = platform\_get\_irq(pdev, 0);

ret = request\_irq(irq, imx\_rxint, 0, dev\_name(&pdev->dev), sport);

platform\_get\_irq() call will return the irq number.

In DT, interrupts are specified like below:

interrupts = <0 66 IRQ\_TYPE\_LEVEL\_HIGH>;

According to ARM GIC, first cell is the type of interrupt

0 - Shared peripheral interrupt, can be routed by GIC to any core.

1 - Private peripheral interrupt, private to an individual core.

The second cell holds the interrupt number - 66 in the example.

The third cell represents the sense level. (include/linux/irq.h)