Device Tree Made Easy

EUF-DES-T1465

Massimo Bonazzi

M A Y . 2 0 1 5

External Use

Agenda

- What’s the device tree?
- How was it working up to now?
- What’s changed?
- Device Tree syntax
- Bindings
- Bindings GIC example
- Bindings I2C example
- Gpio keys example
- i.MX 6Q device tree files
- How to build the dtb
- References

What’s the Device Tree ?

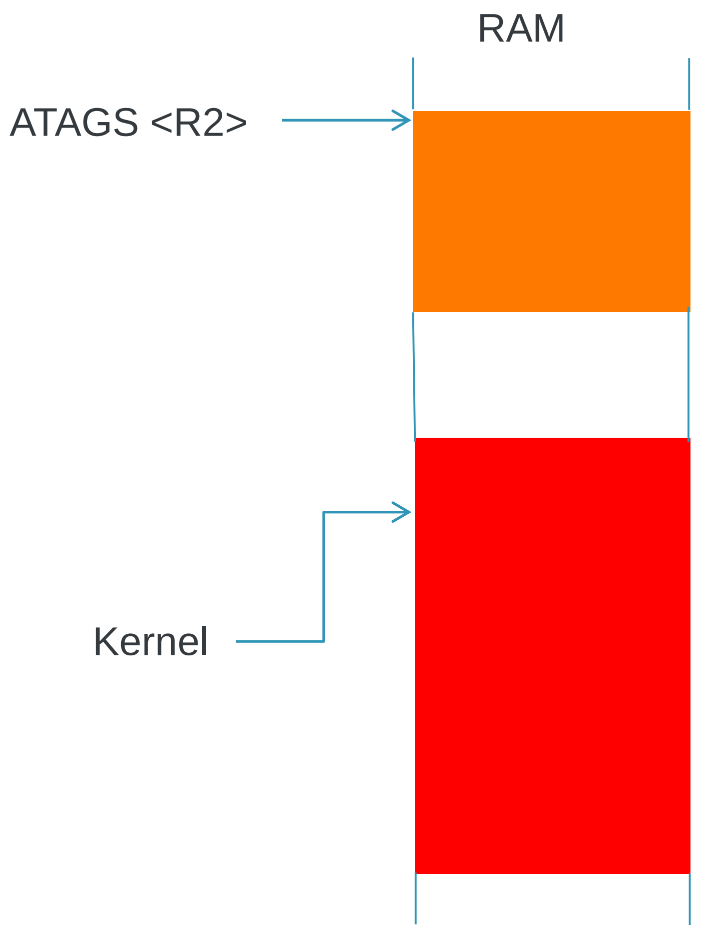
- Literally from Wikipedia:
- The **device tree** is a data structure for describing hardware, which originated from Open Firmware. The data structure can hold any kind of data as internally it is a tree of named nodes and properties. Nodes contain properties and child nodes, while properties are name–value pairs.
- Given the correct device tree, the same compiled kernel can support different hardware configurations within a wider architecture family. The Linux® kernel can read device tree information in the ARM®, x86, MicroBlaze, Power Archicture®, and SPARC architectures. For ARM, use of device trees has become mandatory for all new SoCs. This can be seen as a remedy to the vast number of forks (of Linux and Das U-boot) that has historically been created to support (marginally) different ARM boards.

How Did it Work Until Now?

- The kernel was dependent on the hardware description.
- The bootloader was loading kernel image, and then was executing it.
- Machine type is placed in R1
- The bootloader was also setting up the ATAGS pointer in R2.
- Tha ATAGS is a list of tagged elements, each one starts with length and a tag (ATAG_CORE, ATAG_CMDLINE , ATAG_MEM, ATAG_NONE)
- Old style U-Boot command was:
- bootm <kernel addr>

How Did it Work Until Now?

- ATAGS has to be in RAM
- R2 should contain ATAG address in recent kernels
- ATAGS must not extend beyond the 0x4000 boundary
- 32 bit aligned
- Starts with ATAG_CORE, ends with ATAG_NONE
- Must contain at least one ATAG_MEM

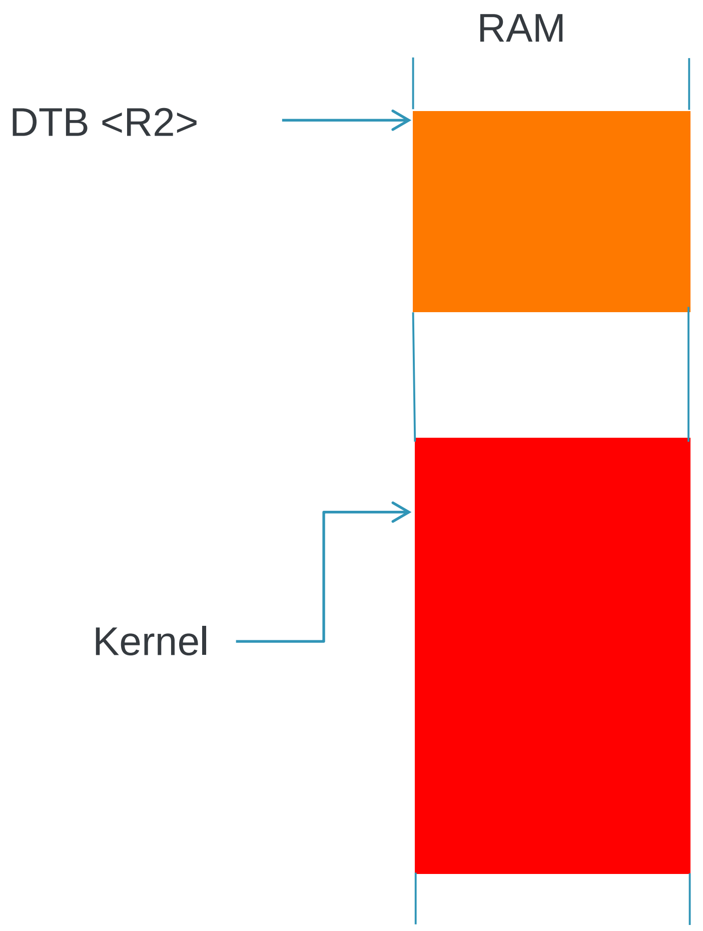


What’s Changed?

- Hardware description is now in the DTB which is a separate binary
- The bootloader loads kernel and DTB as well
- DTB can be found in arch/arm/boot/dts/imx6q-sabresd.dtb
- No more machine type
- R2 now contains DTB address.
- U-Boot command is now changed:
- bootm <kernel img addr> - <dtb addr>

What's hanged?

- R2 now points to the DTB
- R1 is not used anymore
- The same kernel can be used for more than one board



Device Tree Syntax

```
/ {
   cpus {
              #address-cells = <1>;
              #size-cells = <0>;

              cpu0: cpu@0 {
                    compatible = "arm,cortex-a9";
                    device_type = "cpu";
                    reg = <0>;
                    next-level-cache = <&L2>;
                    operating-points = <
                          /* kHz    uV */
                          1200000 1275000
                          996000  1250000
                          852000  1250000
                          792000  1150000
                          396000  975000
                    >;
                    ..................
```

Node name

Unit address

Property value

Label

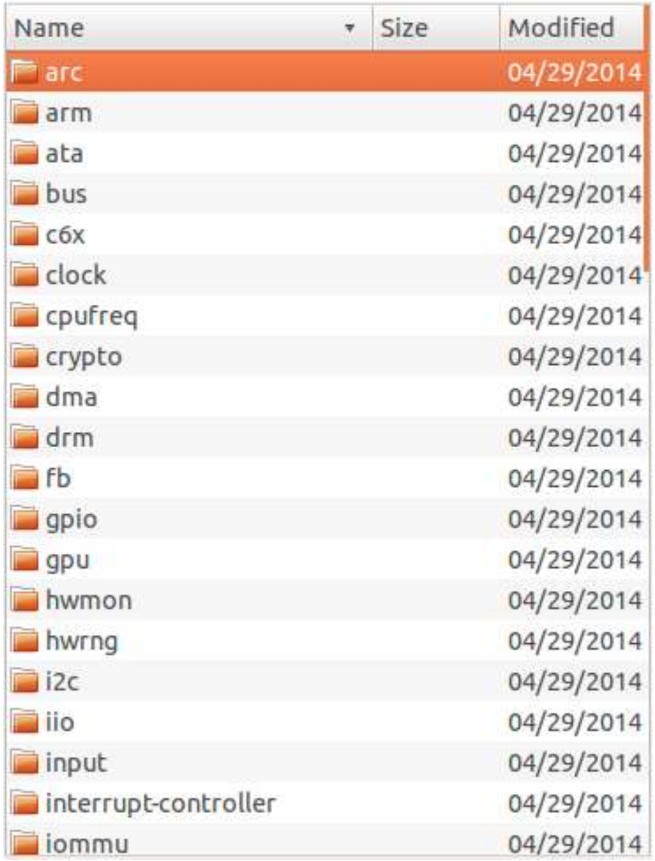
Property name

phandle

Cell property

Bindings

- What are bindings?
  - A "bindings" is a description of how a device is described in the device tree. Bindings for a lot of devices are well established and documented. You can read about them in the existing ePAPR and IEEE 1275 (OpenFirmware) documentation.
  - Bindings documentation can be found in /Documentation/devicetree/bindings



Name	Size	Modified
arc		04/29/2014
arm		04/29/2014
ata		04/29/2014
bus		04/29/2014
c6x		04/29/2014
clock		04/29/2014
cpufreq		04/29/2014
crypto		04/29/2014
dma		04/29/2014
drm		04/29/2014
fb		04/29/2014
gpio		04/29/2014
gpu		04/29/2014
hwmon		04/29/2014
hwrng		04/29/2014
i2c		04/29/2014
iio		04/29/2014
input		04/29/2014
interrupt-controller		04/29/2014
iommu		04/29/2014

Bindings GIC Example

* ARM Generic Interrupt Controller

ARM SMP cores are often associated with a GIC, providing per processor interrupts (PPI), shared processor interrupts (SPI) and software generated interrupts (SGI).

Primary GIC is attached directly to the CPU and typically has PPIs and SGIs. Secondary GICs are cascaded into the upward interrupt controller and do not have PPIs or SGIs.

Main node required properties:

Legal properties

- compatible : should be one of:
    "arm,cortex-a15-gic"
    "arm,cortex-a9-gic"
    "arm,cortex-a7-gic"
    "arm,arm11mp-gic"
- interrupt-controller : Identifies the node as an interrupt controller
- #interrupt-cells : Specifies the number of cells needed to encode an interrupt source. The type shall be a <u32> and the value shall be 3.

And values…

The 1st cell is the interrupt type; 0 for SPI interrupts, 1 for PPI interrupts.

The 2nd cell contains the interrupt number for the interrupt type. SPI interrupts are in the range [0-987]. PPI interrupts are in the range [0-15].

The 3rd cell is the flags, encoded as follows:

bits[3:0] trigger type and level flags.

1 = low-to-high edge triggered
2 = high-to-low edge triggered
4 = active high level-sensitive
8 = active low level-sensitive

bits[15:8] PPI interrupt cpu mask. Each bit corresponds to each of the 8 possible cpus attached to the GIC. A bit set to '1' indicated the interrupt is wired to that CPU. Only valid for PPI interrupts.

Valuable information here!

Bindings I2C Example (I2C-imx.txt)

* Freescale Inter IC (I2C) and High Speed Inter IC (HS-I2C) for i.MX

Required properties:
- compatible : Should be "fsl,<chip>-i2c"
- reg : Should contain I2C/HS-I2C registers location and length
- interrupts : Should contain I2C/HS-I2C interrupt

Required properties

Optional properties:
- clock-frequency : Constains desired I2C/HS-I2C bus clock frequency in Hz.
  The absence of the propoerty indicates the default frequency 100 kHz.

Optional properties

Examples:

```
i2c@83fc4000 { /* I2C2 on i.MX51 */
    compatible = "fsl,imx51-i2c", "fsl,imx21-i2c";
    reg = <0x83fc4000 0x4000>;
    interrupts = <63>;
};

i2c@70038000 { /* HS-I2C on i.MX51 */
    compatible = "fsl,imx51-i2c", "fsl,imx21-i2c";
    reg = <0x70038000 0x4000>;
    interrupts = <64>;
    clock-frequency = <400000>;
};
```

Hw description

Bindings I2C Example

```
i2c1: i2c@021a0000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "fsl,imx6q-i2c", "fsl,imx21-i2c";
    reg = <0x021a0000 0x4000>;
    interrupts = <0 36 0x04>;
    clocks = <&clks 125>;
    status = "disabled";
};

i2c2: i2c@021a4000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "fsl,imx6q-i2c", "fsl,imx21-i2c";
    reg = <0x021a4000 0x4000>;
    interrupts = <0 37 0x04>;
    clocks = <&clks 126>;
    status = "disabled";
};
```

Required properties

Status disabled

Address space

Interrupts GIC's style

What clock?

Later on in imx6qdl-sabresd.dtsi

```
&i2c1 {
    clock-frequency = <100000>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_i2c1_2>;
    status = "okay";

    codec: wm8962@1a {
            compatible = "wlf,wm8962";
            reg = <0x1a>;
```

Here it goes to okay

Bindings I2C Example Adding a Device

```
mma8451@1c {
            compatible = "fsl,mma8451";
            reg = <0x1c>;
            position = <1>;
            vdd-supply = <&reg_sensor>;
            vddio-supply = <&reg_sensor>;
};
```

Node and address

Compatible

I2C address

What regulator?

Bindings for mma8450

* Freescale MMA8450 3-Axis Accelerometer

Required properties:
- compatible : "fsl,mma8450".
- reg: the I2C address of MMA8450

Example:

```
accelerometer: mma8450@1c {
    compatible = "fsl,mma8450";
    reg = <0x1c>;
};
```

```
static const struct of_device_id mma8450_dt_ids[] = {
                { .compatible = "fsl,mma8450", },
                { /* sentinel */ }
};
MODULE_DEVICE_TABLE(of, mma8450_dt_ids);
```

It doesn't matter cause the driver is not really using the device tree information.

8450 or 51?

Gpio Keys Example

```
gpio-keys {
            compatible = "gpio-keys";
            power {
                        label = "Power Button";
                        gpios = <&gpio3 29 1>;
                        linux,code = <116>; /* KEY_POWER */
                        gpio-key,wakeup;
            };

            volume-up {
                        label = "Volume Up";
                        gpios = <&gpio1 4 1>;
                        linux,code = <115>; /* KEY_VOLUMEUP */
            };

            volume-down {
                        label = "Volume Down";
                        gpios = <&gpio1 5 1>;
                        linux,code = <114>; /* KEY_VOLUMEDOWN */
            };
  };
```

Driver compatibility bind

Phandle to the pin (bank number flags)

A name for convenience

Need to wakup?

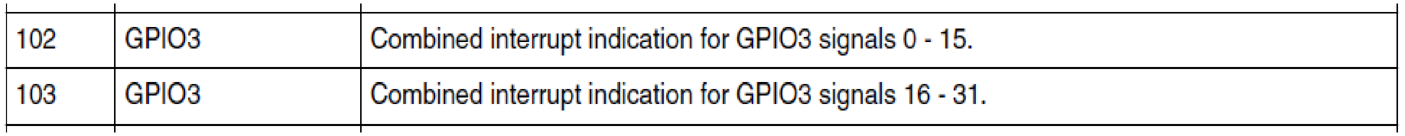
The key number

```
./include/linux/input.h
#define KEY_VOLUMEDOWN        114
#define KEY_VOLUMEUP          115
#define KEY_POWER             116
```

Here they are…

Gpio Keys Example

020A_4000	020A_7FFF	GPIO3	16KB



102	GPIO3	Combined interrupt indication for GPIO3 signals 0 - 15.
103	GPIO3	Combined interrupt indication for GPIO3 signals 16 - 31.

Bank 3 address

```
gpio3: gpio@020a4000 {
        compatible = "fsl,imx6q-gpio", "fsl,imx35-gpio";
        reg = <0x020a4000 0x4000>;
        interrupts = <0 70 0x04 0 71 0x04>;
        gpio-controller;
        #gpio-cells = <2>;
        interrupt-controller;
        #interrupt-cells = <2>;
};
```

16K

Shared 102-32=70 Trigger

Marks

Documentation/devicetree/bindings/arm/gic.txt

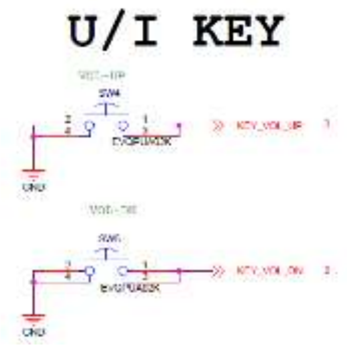
- #interrupt-cells : Specifies the number of cells needed to encode an interrupt source. The type shall be a <u32> and the value shall be 3.

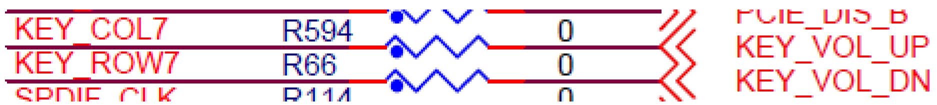
The 1st cell is the interrupt type; 0 for SPI interrupts, 1 for PPI interrupts.

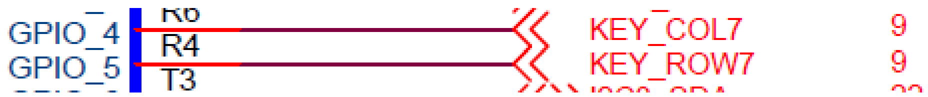
The 2nd cell contains the interrupt number for the interrupt type. SPI interrupts are in the range [0-987]. PPI interrupts are in the range [0-15].

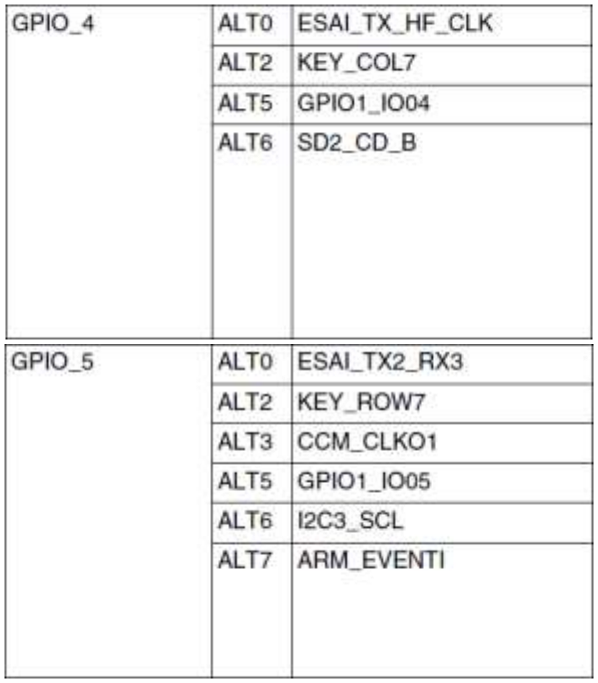
The 3rd cell is the flags, encoded as follows:
    bits[3:0] trigger type and level flags.
        1 = low-to-high edge triggered
        2 = high-to-low edge triggered
        4 = active high level-sensitive
        8 = active low level-sensitive
    bits[15:8] PPI interrupt cpu mask. Each bit corresponds to each of the 8 possible cpus attached to the GIC. A bit set to '1' indicated the interrupt is wired to that CPU. Only valid for PPI interrupts.

Gpio Keys Example









GPIO_4	ALT0	ESAI_TX_HF_CLK
	ALT2	KEY_COL7
	ALT5	GPIO1_IO04
	ALT6	SD2_CD_B
GPIO_5	ALT0	ESAI_TX2_RX3
	ALT2	KEY_ROW7
	ALT3	CCM_CLKO1
	ALT5	GPIO1_IO05
	ALT6	I2C3_SCL
	ALT7	ARM_EVENTI

```
volume-up {
		label = "Volume Up";
		gpios = <&gpio1 4 1>;
		linux,code = <115>; /* KEY_VOLUMEUP */
		};
volume-down {
		label = "Volume Down";
		gpios = <&gpio1 5 1>;
		linux,code = <114>; /* KEY_VOLUMEDOWN */
		};
```

Driver Interface to the DTB

```
static struct gpio_keys_platform_data * gpio_keys_get_devtree_pdata(struct device *dev)
{
    struct device_node *node, *pp;
    struct gpio_keys_platform_data *pdata;
    struct gpio_keys_button *button;
    int error,nbuttons;
    node = dev->of_node;
    nbuttons = of_get_child_count(node);
    pdata = kzalloc(sizeof(*pdata) + nbuttons * (sizeof *button),GFP_KERNEL);
    pdata->buttons = (struct gpio_keys_button *)(pdata + 1);
    pdata->nbuttons = nbuttons;
    pdata->rep = !!of_get_property(node, "autorepeat", NULL);
    for_each_child_of_node(node, pp) {
            int gpio;
            enum of_gpio_flags flags;
            if (!of_find_property(pp, "gpios", NULL)) {
                    pdata->nbuttons--;
                    dev_warn(dev, "Found button without gpios\n");
                    continue;
            }
            gpio = of_get_gpio_flags(pp, 0, &flags);
            button = &pdata->buttons[i++];
            button->gpio = gpio;
            button->active_low = flags & OF_GPIO_ACTIVE_LOW;
```

All done here!

How many?

The node

Get autorepeat

Iterate…

Is this property there?

Get flags

Driver Interface to the DTB

Get Key code

Get label

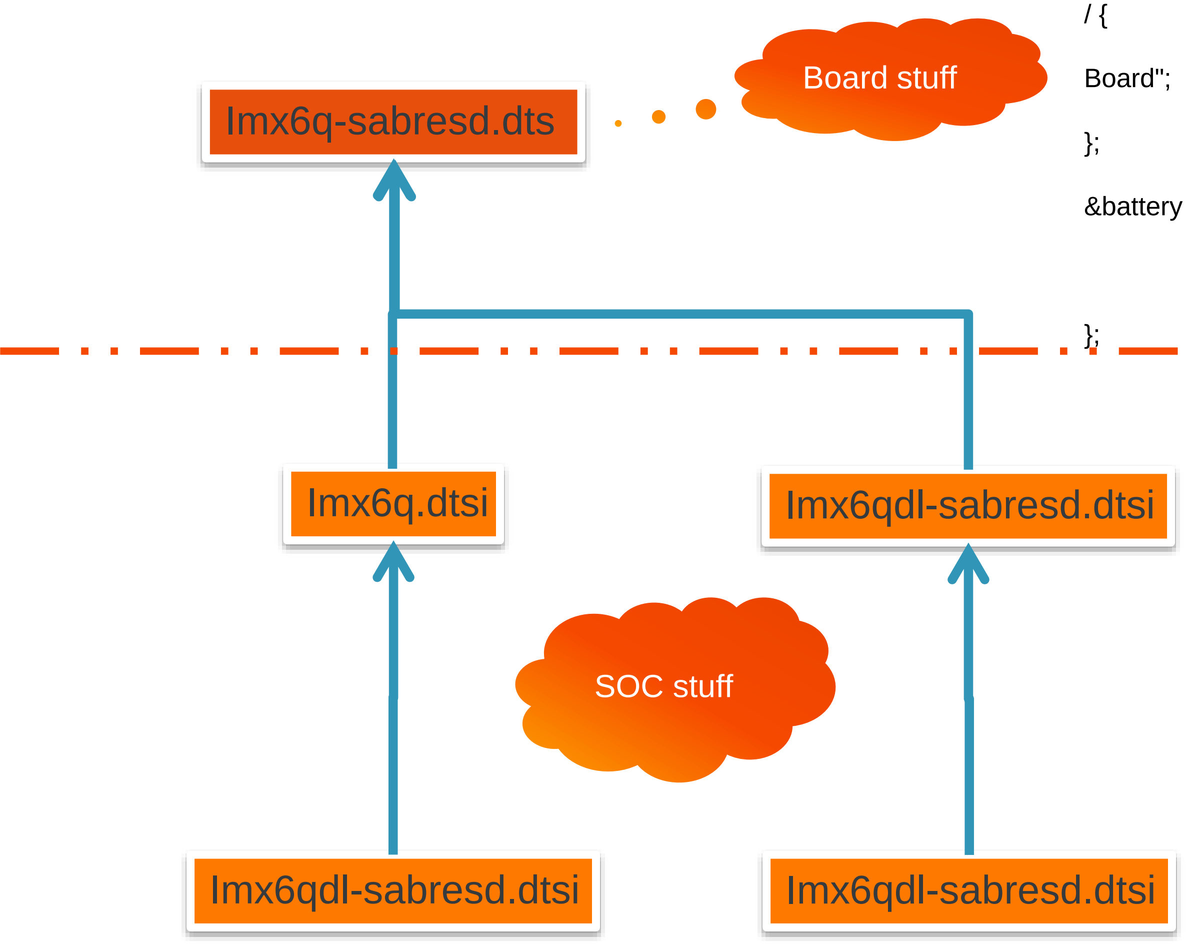
Is it wakeup?

```
                if (of_property_read_u32(pp, "linux,code", &button->code)) {
                        dev_err(dev, "Button without keycode: 0x%x\n",button->gpio);
                        error = -EINVAL;
                        }
                button->desc = of_get_property(pp, "label", NULL);
                if (of_property_read_u32(pp, "linux,input-type", &button->type))
                        button->type = EV_KEY;

                button->wakeup = !!of_get_property(pp, "gpio-key,wakeup", NULL);

                if (of_property_read_u32(pp, "debounce-interval",  &button->debounce_interval))
                        button->debounce_interval = 5;
    }
return pdata;
}

static struct of_device_id gpio_keys_of_match[] = {
    { .compatible = "gpio-keys", },
    { },
};
MODULE_DEVICE_TABLE(of, gpio_keys_of_match);
```

Need debounce?

What I am compatible with

i.MX 6Q Device Tree Files



```
#include "imx6q.dtsi"
#include "imx6qdl-sabresd.dtsi"

/ {
                model = "Freescale i.MX6 Quad SABRE Smart Device
Board";
                compatible = "fsl,imx6q-sabresd", "fsl,imx6q";
};

&battery {
                offset-charger = <1900>;
                offset-discharger = <1694>;
                offset-usb-charger = <1685>;
};
```

```
cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        cpu0: cpu@0 {
                compatible = "arm,cortex-a9";
                device_type = "cpu";
                reg = <0>;
                next-level-cache = <&L2>;
                operating-points = <
                        /* kHz    uV */
                        1200000 1275000
                        996000  1250000
                        852000  1250000
                        792000  1150000
                        396000  975000
```

How to Build the DTB

- i.Mx6 DTS is located in:
  - fsl-bsp/build/tmp/work/imx6qsabresd-poky-linux-gnueabi/linux-imx/3.10.17-r0/git/arch/arm/boot/dts
  - dtsi files are like .h files for the DTS
- The DTC (Device Tree Compiler) can be found here:
  - fsl-bsp/build/tmp/work/imx6qsabresd-poky-linux-gnueabi/linux-imx/3.10.17-r0/git/scripts/dtc/dtc
- The DTB is produced by the DTC and it can be rebuilt like this:
  - export PATH=$PATH:~/yocto/fsl-bsp/build-fb/tmp/sysroots/x86_64-linux/usr/bin/cortexa9hf-vfp-neon-poky-linux-gnueabi
  - cd yocto/fsl-bsp/build-fb/tmp/work/imx6qsabresd-poky-linux-gnueabi/linux-imx/3.10.17-r0/git/
  - make ARCH=arm CROSS_COMPILE=~/yocto/fsl-bsp/build-fb/tmp/sysroots/x86_64-linux/usr/bin/cortexa9hf-vfp-neon-poky-linux-gnueabi/arm-poky-linux-gnueabi- imx6q-sabresd.dtb

References

- Wikipedia.org
- Linux Kernel Doc’s in Documentation/devicetree
- Thomas Petazzoni’s preso in http://www.freeelectrons.org
- Power.orgTM Standard for Embedded Power Architecture Platform Requirements (ePAPR), http://www.power.org/
- http://www.devicetree.org

freescale™

www.Freescale.com