

Embedded systems 4/7

J.-M Friedt, G. Goavec-Mérou

FEMTO-ST/time & frequency department

`jmfriedt@femto-st.fr`

slides at `jmfriedt.free.fr`

October 14, 2024

Previous session

A driver defines the low-level functionalities and is called by a device definition matching the `.name` attribute:

```
static struct platform_driver gpio_simple_driver = {
    .probe = gpio_simple_probe,
    .remove = gpio_simple_remove,
    .driver = { .name = "simple",
    },
};

static int __init gpio_simple_init(void)
{ret =platform_driver_register(&gpio_simple_driver);
 pdev=platform_device_register_simple("simple",0,NULL,0);
                                     // no argument: NULL,0
}
```

Passing parameters to the driver

On the platform device: resource¹ definition

```
static struct resource jmf_res[] = {
    {.start = 0x378,
     .end   = 0x379,
     .flags = IORESOURCE_IO, // NOT IORESOURCE_MEM
     .name  = "io-memory1"
    },
    {.start = 0x37a,
     .end   = 0x37b,
     .flags = IORESOURCE_IO,
     .name  = "io-memory2"
    },
};

static struct platform_device *pdev1;

static int __init gpio_simple_init(void)
{pdev1=platform_device_register_simple("jmf",0,jmf_res,ARRAY_SIZE(jmf_res));
...
}
```

The last two arguments of `platform_device_register_simple()` provide the arguments to the driver.

¹definition in `include/linux/ioport.h`

See include/linux/ioport.h ...

... for the allowed flag values:

```
struct resource {
    resource_size_t start;
    resource_size_t end;
    const char *name;
    unsigned long flags;
    struct resource *parent, *sibling, *child;
};

/*
 * IO resources have these defined flags.
 */
#define IORESOURCE_BITS          0x000000ff  /* Bus-specific bits */
#define IORESOURCE_TYPE_BITS    0x00001f00  /* Resource type */
#define IORESOURCE_IO           0x00000100  /* PCI/ISA I/O ports */
#define IORESOURCE_MEM          0x00000200
#define IORESOURCE_REG          0x00000300  /* Register offsets */
#define IORESOURCE_IRQ          0x00000400
#define IORESOURCE_DMA          0x00000800
#define IORESOURCE_BUS          0x00001000
```

Passing parameters to the driver module

In the driver:

```
static int gpio_simple_probe(struct platform_device *pdev)
{
    struct resource *r1,*r2;
    r1= platform_get_resource(pdev, IORESOURCE_IO, 0);
    r2= platform_get_resource(pdev, IORESOURCE_IO, 1);
    printk(KERN_ALERT "jmf %d %x %x\n",pdev->id,(int)r1->start,(int)r2->start);
    ...
}
```

leads to (dmesg)

```
[ 331.353399] jmf 0 378 37a
```

and

```
# cat /proc/ioproports
0378-0379 : io-memory1
037a-037b : io-memory2
```

⇒ multiple configurations can be provided for a same peripheral

Passing parameters to the driver module

⇒ a driver is given multiple configurations for a same peripheral type located at different addresses:

In the device module:

```
static struct resource jmf_resources1[] = {
    {.start=0x378, .end=0x379, .flags=IORESOURCE_IO, .name="io-memory1"},
    {.start=0x37a, .end=0x37b, .flags=IORESOURCE_IO, .name="io-memory2"},
};
static struct resource jmf_resources2[] = {
    {.start=0x37c, .end=0x37d, .flags=IORESOURCE_IO, .name="io-memory3"},
};

static struct platform_device *pdev1,*pdev2;

static int __init gpio_simple_init(void)
{pdev1=platform_device_register_simple("jmf",0,jmf_resources1,ARRAY_SIZE(jmf_resources1));
  pdev2=platform_device_register_simple("jmf",1,jmf_resources2,ARRAY_SIZE(jmf_resources2));
...}
```

and in the driver

```
static int gpio_simple_probe(struct platform_device *pdev)
{struct resource *r1,*r2;
  printk("resources: %d",pdev->num_resources);
  r1= platform_get_resource(pdev, IORESOURCE_IO, 0);
  printk(KERN_ALERT "start1 %d %x\n",pdev->id,(int)r1->start);
  if (pdev->num_resources>1)
    {r2= platform_get_resource(pdev, IORESOURCE_IO, 1);
     printk(KERN_ALERT "start2 %d %x\n",pdev->id,(int)r2->start);
    } // $LINUX/Documentation/driver-api/driver-model/platform.rst
...}
```

dmesg :

```
[ 187.705199] resources: 2
[ 187.705200] start1 0 378
[ 187.705653] start2 0 37a
[ 187.706333] resources: 1
[ 187.706334] start1 1 37c
```

and these entries are found in /sys/bus/platform/devices/jmf*

Challenge of peripheral description

- ▶ for enumerated buses (USB, PCI, firewire), a driver is loaded when the peripheral is detected
- ▶ for other buses (SPI, I²C, ISA/PC104), hardware must be explicitly described
- ▶ inserting hardware description in the kernel ⇒ source code growth (one driver for each peripheral implementation) and redundancy ²

```
/* SPI */
static struct spi_board_info pcm037_spi_dev[] = {
    {
        .modalias      = "dac124s085",
        .max_speed_hz  = 400000,
        .bus_num       = 0,
        .chip_select    = 0,          /* Index in pcm037_spi1_cs[] */
        .mode          = SPI_CPHA,
    },
};

/* Platform Data for MXC CSPI */
static int pcm037_spi1_cs[] = {MXC_SPI_CS(1), IOMUX_TO_GPIO(MX31_PIN_KEY_COL7)};

static const struct spi_imx_master pcm037_spi1_pdata __initconst = {
    .chipselect = pcm037_spi1_cs,
    .num_chipselect = ARRAY_SIZE(pcm037_spi1_cs),
};

int __init pcm037_eet_init_devices(void)
{[...]
    /* SPI */
    spi_register_board_info(pcm037_spi_dev, ARRAY_SIZE(pcm037_spi_dev));
    imx31_add_spi_imx0(&pcm037_spi1_pdata);
}
```

²linux-4.4.2/arch/arm/mach-imx/mach-pcm037_eet.c

Challenge of peripheral description

- ▶ for enumerated buses (USB, PCI, firewire), a driver is loaded when the peripheral is detected
- ▶ for other buses (SPI, I²C, ISA/PC104), hardware must be explicitly described
- ▶ inserting hardware description in the kernel ⇒ source code growth (one driver for each peripheral implementation) and redundancy

From Linus Torvalds <>

Date Thu, 17 Mar 2011 19:50:36 -0700

Subject Re: [GIT PULL] omap changes for v2.6.39 merge window

On Thu, Mar 17, 2011 at 11:30 AM, Tony Lindgren <tony@atomide.com> wrote:

> Please pull omap changes for this merge window from:

Gaah. Guys, this whole ARM thing is a f*cking pain in the ass.

You need to stop stepping on each others toes. There is no way that your changes to those crazy clock-data files should constantly result in those annoying conflicts, just because different people in different ARM trees do some masturbatory renaming of some random device. Seriously.

[...]

<https://lkml.org/lkml/2011/3/17/492>

Challenge of peripheral description

- ▶ for enumerated buses (USB, PCI, firewire), a driver is loaded when the peripheral is detected
- ▶ for other buses (SPI, I²C, ISA/PC104), hardware must be explicitly described
- ▶ inserting hardware description in the kernel \Rightarrow source code growth (one driver for each peripheral implementation) and redundancy
- ▶ **Solution** provided by PowerPC & SPARC architectures ported to ARM: the *devicetree*³ for describing peripheral and their dependencies

\Rightarrow a human readable ASCII file describing the peripherals and their dependencies, converted to a binary file used by the Linux kernel upon starting or while being executed (overlay)^{4 5}

³www.kernel.org/doc/Documentation/devicetree/usage-model.rst

⁴T. Petazzoni, *Introduction au "device tree" sur ARM*, Opensilicium Janvier-Février-Mars 2016, pp. 46-59, à jmfriedt.free.fr/devicetree_os19.pdf

⁵T. Petazzoni, *Device Tree for dummies*, ELC 2014
events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf, talk at www.youtube.com/watch?v=uzBwHFjJ0vU

devicetree architecture

Peripheral hierarchy: SoC=CPU+interfaces, fitted on a development board ⁶

Device Tree Implementation

Layers

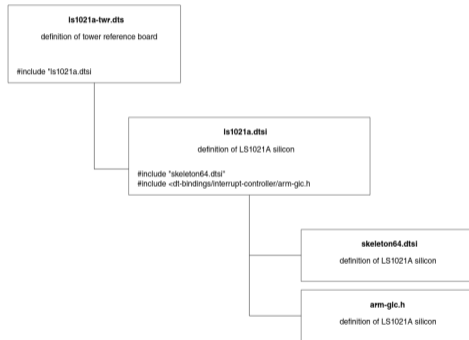
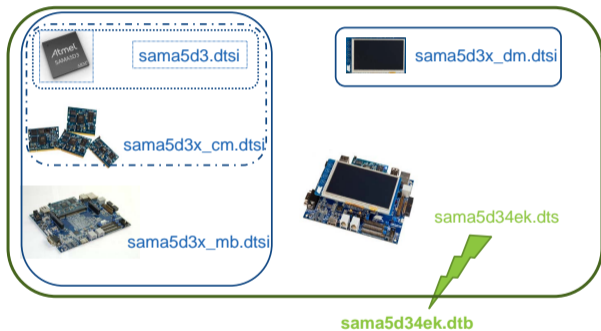


Figure 4. LS1021A DTS and DTSI structure

Not valid on a PC (x86/amd64 architectures)

⁶Atmel AN-8481, *Linux Device Tree Introduction* (2014), or AN5125 *Introduction to Device Trees*, NXP at <https://www.nxp.com/docs/en/application-note/AN5125.pdf>

Hardware description

Example of linux-4.4.2/arch/arm/boot/dts/sun5i-a13-olinuxino-micro.dts

1. in the devicetree:

```
#include "sun5i-a13.dtsi"
```

```
#include <dt-bindings/pinctrl/sun4i-a10.h>
```

```
[...]
```

```
&spi2 { // surcharge la definition de SPI precedente
```

```
    pinctrl-0 = <&spi2_pins_a>, <&spi2_cs0_pins_a>;
```

```
    status = "okay";
```

```
    ad9834@0 {
```

```
        compatible = "ad9834_2";
```

```
        spi-max-frequency = <50000000>; // 5 MHz
```

```
        spi-cpol;
```

```
        reg = <0>;
```

```
        vcc-supply = <&reg_ad9834_vref>;
```

```
        freq0 = <136727>;
```

```
    };
```

```
};
```

```
reg_ad9834_vref: vref-reg@1 {
```

```
    compatible = "regulator-fixed";
```

```
    regulator-max-microvolt = <3300000>;
```

```
    [...]
```

```
};
```

```
};
```

Link with the driver

Example of linux-4.4.2/arch/arm/boot/dts/sun5i-a13-olinuxino-micro.dts

1. in the devicetree:

```
#include "sun5i-a13.dtsi"
#include <dt-bindings/pinctrl/sun4i-a10.h>
[...]

&spi2 { // surcharge la definition de SPI precedente
    pinctrl-0 = <&spi2_pins_a>, <&spi2_cs0_pins_a>;
    status = "okay";
    ad9834@0 {
        compatible = "ad9834_2";
        spi-max-frequency = <50000000>; // 5 MHz
        spi-cpol;
        reg = <0>;
        vcc-supply = <&reg_ad9834_vref>;
        freq0 = <136727>;
    };
};
```

2. and in the driver:

```
static const struct spi_device_id ad9834_id[] = {
    {"ad9834_2", ID_AD9834}, {}
};

MODULE_DEVICE_TABLE(spi, ad9834_id);
```

Devicetree compilation

1. The “original” devicetress is located in
output/build/linux-*/arch/arm/boot/xilinx/dts/zynq-red_pitaya.dts
⇒ modify this file and compile the kernel (requires write access to the buildroot directory)
2. binary→ ASCII of an existing devicetree ⁷

```
dtc -I dtb -O dts filename.dtb
```

3. or fdt dump zynq-red_pitaya.dtb
4. ASCII → binary:

```
dtc -O dtb -o filename.dtb filename.dts
```

OF=OpenFirmware⁸

⁷script <https://git.kernel.org/cgit/utils/dtc/dtc.git> or buildroot/output/host/usr/bin/dtc :
remember to use the dtc **provided by buildroot** and not the one from the host

⁸https://www.openfirmware.info/Welcome_to_OpenBIOS

Adding a new entry to the devicetree

Loading the driver

```
# insmod composant.ko
```

does not trigger any action as long as we have not

```
# insmod dummy_platform.ko
```

which inserts the platform device requesting the driver support.

```
[ 838.108990] . Entering probe
```

```
[ 838.111930] . Registering
```

```
[ 838.114825] . Registered
```

⇒ adding an entry to call the driver from the *devicetree*

Adding a new entry to the devicetree

⇒ adding an entry to call the driver from the *devicetree*

1. fetch the current *devicetree* in the boot partition

```
# mount /dev/mmcblk0p1 /mnt/  
# ls /mnt/  
... zImage system.dtb
```

(system.dtb was renamed from zynq-red_pitaya.dtb in the Buildroot output/images)

2. binary → ASCII
3. add a node calling the driver (compatible = description that was taken care of by the device previously)

```
dummy_entry {compatible="composant1"};
```
4. ASCII → binary
5. reboot the Redpitaya to load the new configuration

Driver-devicetree link

Check that the new node has been inserted: the tree structure of the *devicetree* can be accessed in `/sys/firmware/devicetree/base` \Rightarrow `dummy_entry` is now present

```
$ cat compatible
composant1
```

In the driver, the structures associating with the *devicetree* entry is ⁹ :

```
static const struct of_device_id composant_id_of[] = { // for devicetree
    { .compatible="composant1",
      },{ } };
MODULE_DEVICE_TABLE(of, composant_id_of); // for devicetree

static struct platform_driver composant_driver = {
    .driver = {
        .name = "composant",
        .owner = THIS_MODULE,
        .of_match_table=of_match_ptr(composant_id_of), // for devicetree
    },
    .probe = composant_probe,
    .remove = composant_remove,
    .id_table = composant_id, // for platform device
};
module_platform_driver(composant_driver);
```

⁹http://events.linuxfoundation.org/sites/events/files/slides/dt_internals_0.pdf

Driver-devicetree link

This time, the messages printed by `probe()` are displayed as soon as the driver is loaded

```
[ 5804.151642] . Entering probe  
[ 5804.154560] . Registering  
[ 5804.158091] . Registered
```

without requiring explicitly loading the platform device.

⇒ the devicetree describes the hardware and the drivers are loaded dynamically while fetching their configuration

Passing parameters

Parameter passing to the driver upon being loaded by the *devicetree* to describe the peripheral configuration.

- ▶ in the devicetree:

```
dummy_entry {compatible="composant1";toto32=<56>;};
```

- ▶ in the driver:

```
static int composant_probe(struct platform_device *pdev)
{u32 val32=42;u16 val16=42;
 struct device_node *np = pdev->dev.of_node;
 of_property_read_u32(np, "toto32", &val32);
 of_property_read_u16(np, "toto16", &val16);
 printk(KERN_ALERT ". Entering probe %u %u\n",val32,val16);
 ...
}
```

- ▶ check that the variable value has been loaded

```
/sys/firmware/devicetree/base/dummy_entry
```

```
# ls
```

```
compatible  name          toto32
```

```
# hexdump toto32
```

```
00000000 0000 3800
```

```
00000004
```

```
# insmod composant_comm.ko
```

```
[ 232.968359] . Entering probe 56 42
```

Devicetree overlay

- ▶ Dynamically adding entries in the devicetree: overlays^{10 11}
- ▶ Not yet supported in the official Linux kernel
- ▶ Documentation in SRC_LINUX/Documentation/devicetree/overlay-notes.txt
- ▶ Useful when dynamically adding peripherals: FPGA IPs and associated resources described in the *devicetree*¹²
- ▶ ⇒ useful for architectures combining CPU + FPGA (Armadeus Systems, Xilinx Zynq, Intel/Altera SoC)

¹⁰M. Fischer, *FPGA Manager & devicetree overlays*, FOSDEM 2016,
https://archive.fosdem.org/2016/schedule/event/fpga_devicetree/

¹¹P. Antoniou, *Transactional Device Tree & Overlays – Making Reconfigurable Hardware Work*, ELC 2015
<http://events.linuxfoundation.org/sites/events/files/slides/dynamic-dt-elce14.pdf> et conférence
https://www.youtube.com/watch?v=3Ag7ZBC_Nts

¹²<https://git.kernel.org/cgit/linux/kernel/git/next/linux-next.git/tree/Documentation/devicetree/bindings/fpga/fpga-region.txt?id>

Devicetree overlay

Adding an overlay to the devicetree triggers the probe method of the corresponding driver:

```
/dts-v1/;
/plugin/;
/ { compatible = "xlnx,zynq-7000";
    fragment@0
        {target-path = "/";
            __overlay__
            {
                pilote_drv {compatible = "pilote_ctl"};
            };
        };
};
```

is compiled with `BUILDR00T/output/host/usr/bin/dtc -@ -I dts -o pl_ovl.dtbo pl_ovl.dts`

where “@” indicates that an overlay is being compiled

Adding the overlay

```
mkdir /sys/kernel/config/device-tree/overlays/pilote
cat pl_ovl.dtbo > /sys/kernel/config/device-tree/overlays/pilote/dtbo
cd /sys/firmware/devicetree/base/pilote_drv/
cat compatible # > pilote_ctl
```

Devicetree overlay

The driver compatible with the overlay is informed with

```
static const struct of_device_id composant_id_of[] = { // for devicetree
    { .compatible="pilote_ctl", },
    {}
};
MODULE_DEVICE_TABLE(of, composant_id_of);           // for devicetree

static struct platform_driver pilote_driver = {
    .driver = { .name = "pilote_de_jmf", .owner = THIS_MODULE,
                .of_match_table=of_match_ptr(composant_id_of),
            },
    .probe = gpio_simple_probe,
    .remove = gpio_simple_remove,
};

module_platform_driver(pilote_driver);
// creates init() and exit() including platform_register and platform_unregister

static int gpio_simple_remove(struct platform_device *pdev)
{printk("driver Good Bye\n");return 0;}

static int gpio_simple_probe(struct platform_device *pdev)
{printk("driver Hello\n");return 0;}
```

⇒ loading the driver immediately calls the probe method, without having to explicitly load a platform (the devicetree did it for us)

Removing the overlay

```
rmdir /sys/kernel/config/device-tree/overlays/pilote/
```

Linux and FPGA bitstreams

- ▶ `fpga_manager` (Linux) provides a unified interface for all SoC
- ▶ Vivado (Xilinx) provides bitstreams in `.bit` format to be converted to `.bit.bin` with the Xilinx SDK:

```
$VIVADO_SDK/bin/bootgen -image my_bif_file.bif -arch zynq \  
-process_bitstream bin
```

using the following `.bif` configuration file:

```
all: {nom_bitstream.bit}
```

→ `.bit.bin` file

- ▶ the `bit.bin` file is copied to `/lib/firmware` of the (Redpitaya) Zynq.
- ▶ Transferring to the FPGA:

```
echo "bitstream_name.bit.bin" > /sys/class/fpga_manager/fpga0/firmware
```

Overlay associated to a bitstream

- ▶ for each overlay: consistent description of the bitstream, the driver and the resources

```
/dts-v1/;
/plugin/;
/ { compatible = "xlnx,zynq-7000";
    fragment@0 {
        target = <&fpga_full>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <1>;
            firmware-name = "top_redpitaya_axi_gpio_ctl.bit.bin";
            gpio1: gpio@43C00000 {
                compatible = "gpio_ctl";
                reg = <0x43C00000 0x0001f>;
                gpio-controller;
                #gpio-cells = <1>;
                ngpio = <8>;
            };
        };
    };
};
```

Conclusion and laboratory session

Exercise: modify the devicetree so that an associated driver is automatically loaded when inserting the module

Functions related to the devicetree are declared in

```
#include <linux/of.h>
#include <linux/of_device.h>
```

Compatibility of platform device + devicetree ¹³

```
static int composant_probe(struct platform_device *pdev)
{
    struct composant_state *st;
    if (pdev->dev.of_node)
        st->chip_info=of_match_device(composant_id_of,&pdev->dev)->data; // dt
    else
        st->chip_info = // platforme
        &composant_chip_info_tbl[platform_get_device_id(pdev)->driver_data];
}
```

Adapt the driver unblocking the read function on a timer condition so that the period of the timer is given as a parameter in the *devicetree*.

Read LINUX/Documentation/devicetree

¹³https://elixir.bootlin.com/linux/latest/source/drivers/iio/adc/at91_adc.c: static int at91_adc_probe(struct platform_device *pdev)

Overlay example

```
/dts-v1/;
/plugin/;
/ { compatible = "xlnx,zynq-7000";
    fragment@0
        {target-path = "/";
            __overlay__ {
                le_timer_de_jmf {
                    mon_timer2=<2>;
                    compatible="mon_pilote";
                };
            };
        };
};
```

will generate after `cat demo.dtbo > /sys/kernel/config/device-tree/overlays/demo/dtbo`
a directory named `le_timer_de_jmf` in `/sys/firmware/devicetree/base` whose content are the “compatible” and variable fields.