linux驱动开发—基于Device tree机制的驱动编写-iARM-ChinaUnix博客

分类： 其他平台

摘要：媒介 Device Tree是一种用去描绘硬件的数据布局，类似板级描绘说话，发源于OpenFirmware(OF)。正在现在遍及应用的linux kernel 2.6.x版本中，对分歧仄台、分歧硬件，往]

前言

Device Tree是一种用来描述硬件的数据结构，类似板级描述语言，起源于OpenFirmware(OF)。在目前广泛使用的Linux kernel 2.6.x版本中，对于不同平台、不同硬件，往往存在着大量的不同的、移植性差的板级描述代码，以达到对这些不同平台和不同硬件特殊适配的需求。但是过多的平台、过的的不同硬件导致了这样的代码越来越多，最终引发了Linux创始人Linus的不满，以及强烈呼吁改变。Device Tree的引入给驱动适配带来了很大的方便，一套完整的Device Tree可以将一个PCB摆在你眼前。Device Tree可以描述CPU，可以描述时钟、中断控制器、IO控制器、SPI总线控制器、I2C控制器、存储设备等任何现有驱动单位。对具体器件能够描述到使用哪个中断，内存映射空间是多少等等。

关于Device Tree的数据结构和详细使用方法，请大家查看宋宝华老师的一篇博客：

http://blog.csdn.net/airk000/article/details/2

1 基于Device Tree机制内核的驱动开发—实例讲解

这个章节，作者来讲讲基于Linux-3.2.X之后使用device tree机制的内核的驱动开发案例。本文的驱动开发案例是作者工作期间亲自写的键盘驱动代码。CPU平台使用的是NXP（freescale）的i.MX6ul。概要信息描述如下：

硬件平台：NXP（freescale）—i.MX6ul

软件开发平台：ubuntu-12.04

内核版本：Linux-3.14.38

编译环境：yocto project

1.1 基于Device Tree机制的驱动开发—系统如何加载和解析dtb文件

基于Device Tree机制的驱动开发，在驱动当中所使用到的硬件资源都在对应的CPU平台的dts文件上进行配置，然后编译生成dtb文件，放在u-boot分区之后，内核分区之前。这里顺便讲一下，内核是如何解析dtb文件的。其大致过程如下：

系统上电启动之后，u-boot加载dtb，通过u-boot和Linux内核之间的传参操作将dtb文件传给内核，然后内核解析dtb文件，根据device tree中的配置（dtb文件）去初始化设备的CPU管脚、各个外设的状态。device tree中的配置主要是起到了初始化硬件资源的作用，后期可以在驱动中修改设备的硬件资源的状态，比如在device tree中初始化某个GPIO的管脚为上拉状态，可以在驱动加载之后修改这个管脚的状态。

1.2 基于Device Tree机制的驱动开发—dts文件的配置和编译

本节开始以具体的驱动例子讲解如何在驱动开发中配置dts文件。这里使用i.MX6ul平台下的矩阵键盘驱动中使用到的几个GPIO口讲解如何配置dts文件和编译。本次讲解案例用于编译驱动的内核是Linux-3.14.38。首先我们先来看看如何在内核中找到自己相应CPU平台的dts文件：

1.dts文件位于内核的arch/arm/boot/dts/$(board).dts，其中的$(board)指的是对应的CPU平台，比如i.MX6ul平台的dts文件如下：

imx6ul/linux-3.14.38-v2$ vim arch/arm/boot/dts/imx6ul-14x14-evk.dts（部分内容）

```
#include
#include "imx6ul.dtsi"

/ {
        model = "Freescale i.MX6 UltraLite Newland Board";
        compatible = "fsl,imx6ul-14x14-evk", "fsl,imx6ul";

        chosen {
                stdout-path = &uart1;
        };

        memory {
```

```
        reg = <0x80000000 0x20000000>;
};

pxp_v4l2 {
        compatible = "fsl,imx6ul-pxp-v4l2", "fsl,imx6sx-pxp-v4l2", "fsl,imx6sl-pxp-v4l2";
        status = "okay";
};

keyboard {
      compatible = "max-keypad";
      pinctrl-names = "default";
      pinctrl-0 = <&pinctrl_keypad>;
      in-gpios = <&gpio2 3 GPIO_ACTIVE_HIGH>,      //key_in0
                 <&gpio2 4 GPIO_ACTIVE_HIGH>,      //key_in1
                 <&gpio2 5 GPIO_ACTIVE_HIGH>;      //key_in2

      out-gpios = <&gpio2 6 GPIO_ACTIVE_HIGH>,    //key_out0
                  <&gpio2 2 GPIO_ACTIVE_HIGH>,    //key_out1
                  <&gpio2 7 GPIO_ACTIVE_HIGH>,    //key_out2
                  <&gpio4 25 GPIO_ACTIVE_HIGH>,   //key_out3
                  <&gpio4 26 GPIO_ACTIVE_HIGH>;   //key_out4
                  status = "okay";
         };
};

&cpu0 {
        arm-supply = <®_arm>;
        soc-supply = <®_soc>;
};

&clks {
        assigned-clocks = <&clks IMX6UL_CLK_PLL4_AUDIO_DIV>;
        assigned-clock-rates = <786432000>;
};

&tsc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_tsc>;
        status = "okay";
        xnur-gpio = <&gpio1 3 0>;
        measure_delay_time = <0xffff>;
        pre_charge_time = <0xfff>;
};

&gpmi {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_gpmi_nand_1>;
        status = "okay";
        nand-on-flash-bbt;
};

&lcdif {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_lcdif_dat
                     &pinctrl_lcdif_ctrl>;
    lcd_reset = <&gpio3 14 GPIO_ACTIVE_HIGH>;
        display = <&display0>;
        status = "okay";

        display0: display {
                bits-per-pixel = <16>;
                bus-width = <8>;

                display-timings {
```

```
                        native-mode = <&timing0>;
                        timing0: timing0 {
                        clock-frequency = <9200000>;
                        hactive = <240>;
                        vactive = <320>;
                        hfront-porch = <8>;
                        hback-porch = <4>;
                        hsync-len = <41>;
                        vback-porch = <2>;
                        vfront-porch = <4>;
                        vsync-len = <10>;

                        hsync-active = <0>;
                        vsync-active = <0>;
                        de-active = <1>;
                        pixelclk-active = <0>;
                        };
                };
        };
};



&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_uart1>;
        imx6ul-evk {
                pinctrl_uart1: uart1grp {
                        fsl,pins = <
                                MX6UL_PAD_UART1_TX_DATA__UART1_DCE_TX 0x1b0b1
                                MX6UL_PAD_UART1_RX_DATA__UART1_DCE_RX 0x1b0b1
                        >;
                };

                pinctrl_tsc: tscgrp {
                        fsl,pins = <
                                MX6UL_PAD_GPIO1_IO01__GPIO1_IO01        0xb0
                                MX6UL_PAD_GPIO1_IO02__GPIO1_IO02        0xb0
                                MX6UL_PAD_GPIO1_IO03__GPIO1_IO03        0xb0
                                MX6UL_PAD_GPIO1_IO04__GPIO1_IO04        0xb0
                        >;
                };

                pinctrl_lcdif_dat: lcdifdatgrp {
                        fsl,pins = <
                                MX6UL_PAD_LCD_DATA00__LCDIF_DATA00  0x79
                                MX6UL_PAD_LCD_DATA01__LCDIF_DATA01  0x79
                                MX6UL_PAD_LCD_DATA02__LCDIF_DATA02  0x79
                                MX6UL_PAD_LCD_DATA03__LCDIF_DATA03  0x79
                                MX6UL_PAD_LCD_DATA04__LCDIF_DATA04  0x79
                                MX6UL_PAD_LCD_DATA05__LCDIF_DATA05  0x79
                                MX6UL_PAD_LCD_DATA06__LCDIF_DATA06  0x79
                                MX6UL_PAD_LCD_DATA07__LCDIF_DATA07  0x79
                        >;
                };

                pinctrl_lcdif_ctrl: lcdifctrlgrp {
                        fsl,pins = <
                                MX6UL_PAD_LCD_CLK__LCDIF_WR_RWN     0x79
                                MX6UL_PAD_LCD_ENABLE__LCDIF_RD_E    0x79
                                MX6UL_PAD_LCD_HSYNC__LCDIF_RS       0x79
                                MX6UL_PAD_LCD_RESET__LCDIF_CS       0x79
                                /* used for lcd reset */
```

```
                        MX6UL_PAD_LCD_DATA09__GPIO3_IO14    0x79
                >;
        };

        pinctrl_keypad: keypadgrp {
     fsl,pins = <
        MX6UL_PAD_ENET1_RX_EN__GPIO2_IO02       0x70a0
        MX6UL_PAD_ENET1_TX_DATA0__GPIO2_IO03    0x70a0
        MX6UL_PAD_ENET1_TX_DATA1__GPIO2_IO04    0x70a0
        MX6UL_PAD_ENET1_TX_EN__GPIO2_IO05       0x70a0
        MX6UL_PAD_ENET1_TX_CLK__GPIO2_IO06      0x70a0
        MX6UL_PAD_ENET1_RX_ER__GPIO2_IO07       0x70a0
        MX6UL_PAD_CSI_DATA04__GPIO4_IO25        0x70a0
        MX6UL_PAD_CSI_DATA05__GPIO4_IO26        0x70a0
    >;
};

        pinctrl_gpmi_nand_1: gpmi-nand-1 {
                fsl,pins = <
                        MX6UL_PAD_NAND_CLE__RAWNAND_CLE         0xb0b1
                        MX6UL_PAD_NAND_ALE__RAWNAND_ALE         0xb0b1
                        MX6UL_PAD_NAND_WP_B__RAWNAND_WP_B       0xb0b1
                        MX6UL_PAD_NAND_READY_B__RAWNAND_READY_B 0xb000
                        MX6UL_PAD_NAND_CE0_B__RAWNAND_CE0_B     0xb0b1
                        MX6UL_PAD_NAND_CE1_B__RAWNAND_CE1_B     0xb0b1
                        MX6UL_PAD_NAND_RE_B__RAWNAND_RE_B       0xb0b1
                        MX6UL_PAD_NAND_WE_B__RAWNAND_WE_B       0xb0b1
                        MX6UL_PAD_NAND_DATA00__RAWNAND_DATA00   0xb0b1
                        MX6UL_PAD_NAND_DATA01__RAWNAND_DATA01   0xb0b1
                        MX6UL_PAD_NAND_DATA02__RAWNAND_DATA02   0xb0b1
                        MX6UL_PAD_NAND_DATA03__RAWNAND_DATA03   0xb0b1
                        MX6UL_PAD_NAND_DATA04__RAWNAND_DATA04   0xb0b1
                        MX6UL_PAD_NAND_DATA05__RAWNAND_DATA05   0xb0b1
                        MX6UL_PAD_NAND_DATA06__RAWNAND_DATA06   0xb0b1
                        MX6UL_PAD_NAND_DATA07__RAWNAND_DATA07   0xb0b1
                >;
        };
};
```

2.根据自己的开发需求配置dts文件，本文矩阵键盘驱动所使用到的GPIO管脚资源为：gpio2-2、gpio2-3、gpio2-4、gpio2-5、gpio2-6、gpio2-7、gpio4-25、gpio4-26。dts文件配置如下：

~/yangfile/imx6ul/linux-3.14.38-v2$ vim arch/arm/boot/dts/imx6ul-newland.dts

2.1 在dts文件中添加一个设备节点，比如我们是矩阵键盘驱动，那么就添加一个名为”keyboard“的设备节点；

2.2 compatible属性用于of_find_node_compatible函数获取设备节点用的，这个函数的通过”max-keypad“字符串去遍历device tree，查找匹配的设备节点；

2.3 pinctrl-0 = <&pinctrl_keypad>主要用于说明设备硬件资源在哪里获取，比如这里就是到iomuxc里面去获取IO资源

2.4 iomuxc设备节点里面定义了CPU所有的IO资源，包括每个IO口的初始化状态都定义好了，比如：MX6UL_PAD_ENET1_RX_EN_GPIO2_IO02 0x70a0，这里的MX6UL_PAD_ENET1_RX_EN_GPIO2_IO02宏表示的是GPIO2-2这个IO口的寄存器组（IO复用寄存器、IO方向控制寄存器、IO输入输出值设置寄存器），0x70a0这个值根据自己的驱动开发需求，查阅CPU手册定义，不唯一。

```
keyboard {
    compatible = "max-keypad";
    pinctrl-names = "default";//这个设置成默认default就可以了，没什么特别要求
    pinctrl-0 = <&pinctrl_keypad>;//到iomuxc里面去获取相应IO资源的初始化状态
    in-gpios = <&gpio2 3 GPIO_ACTIVE_HIGH>,      //“in-gpios”字符串可以自己随便定义，主要是为了获取gpio资源的时候匹配用的
               <&gpio2 4 GPIO_ACTIVE_HIGH>,      //GPIO_ACTIVE_HIGH：逻辑高电平有效
               <&gpio2 5 GPIO_ACTIVE_HIGH>;      //key_in2

    out-gpios = <&gpio2 6 GPIO_ACTIVE_HIGH>,    //“out-gpios”字符串可以自己随便定义，主要是为了获取gpio资源的时候匹配用的
```

```
                <&gpio2 2 GPIO_ACTIVE_HIGH>,    //key_out1
                <&gpio2 7 GPIO_ACTIVE_HIGH>,    //key_out2
                <&gpio4 25 GPIO_ACTIVE_HIGH>,   //key_out3
                <&gpio4 26 GPIO_ACTIVE_HIGH>;   //key_out4
                status = "okay";//使能要使用的gpio资源
    };
};
```

```
&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_uart1>;
。。。。。。。。
pinctrl_keypad: keypadgrp {
             fsl,pins = <
                MX6UL_PAD_ENET1_RX_EN__GPIO2_IO02       0x70a0
                MX6UL_PAD_ENET1_TX_DATA0__GPIO2_IO03    0x70a0
                MX6UL_PAD_ENET1_TX_DATA1__GPIO2_IO04    0x70a0
                MX6UL_PAD_ENET1_TX_EN__GPIO2_IO05       0x70a0
                MX6UL_PAD_ENET1_TX_CLK__GPIO2_IO06      0x70a0
                MX6UL_PAD_ENET1_RX_ER__GPIO2_IO07       0x70a0
                MX6UL_PAD_CSI_DATA04__GPIO4_IO25        0x70a0
                MX6UL_PAD_CSI_DATA05__GPIO4_IO26        0x70a0
            >;
    };
```

3.编译dts文件，在内核根目录下执行以下命令：

~/yangfile/imx6ul/linux-3.14.38-v2$ make ARCH=arm CROSS_COMPILE=arm-linux-gcc imx6ul-newland.dtb

(这里的arm-Linux-gcc只是个代表交叉编译器的标识，具体的根据实际情况而定)

4.将配置、编译后的dtb文件烧录到设备flash(或者SD卡)的dtb分区中。

2 驱动代码中如何注册dts文件中的设备

接触了device tree机制的驱动开发后，其实device tree机制就是Linux-2.6.x中的platform 总线机制的优化版本。OK,我们来说说基于device tree机制的驱动开发中注册设备的过程，这里以我写的矩阵键盘驱动代码的设备注册过程为例： 1.在probe函数中调用of_get_**或者of_find_**函数从dtb中获取设备资源：

```
static int max_keypad_probe(struct platform_device *pdev)
{
    int i,ret;
    struct device *dev;
    struct device_node *dev_node = NULL;       //add by zengxiany

    dev = &pdev->dev;
    。。。。。。
    //省略部分代码
    dev_node = of_find_compatible_node(NULL,NULL,"fsl,imx6ul-gpio");
    if(!of_device_is_compatible(dev_node,"fsl,imx6ul-gpio"))
    {
        printk("get keypad device node error!\n");
        return -EINVAL;
    }
    dev_node = of_find_compatible_node(dev_node,NULL,"max-keypad");
    if(!of_device_is_compatible(dev_node,"max-keypad"))
    {
        printk("failure to find max-keypad device node!\n");
        return -EINVAL;
    }

    for(i=0; i< KEYPAD_ROWS; i++)
    {
        gpio_map_rowkey[i] = of_get_named_gpio(dev_node,"in-gpios",i);
```

```
                set_key_input(gpio_map_rowkey[i]);
    }

        for(i=0; i< KEYPAD_COLS; i++)
    {
        gpio_map_colkey0[i] = of_get_named_gpio(dev_node,"out-gpios",i);
                set_key_input(gpio_map_colkey0[i]);
        }
}
```

2.在init函数中注册设备：

```
//add by zengxiany for platform device register
static struct of_device_id max_keypad_of_match[] = {
        { .compatible = "max-keypad", },
        { },
};

static struct platform_driver max_keypad_device_driver = {
        .probe          = max_keypad_probe,
        .remove         = max_keypad_remove,
        .driver         = {
                .name   = "max-keypad",
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(max_keypad_of_match),
        }
};
```

```
static int __init keypad_module_init(void)
{
        int ret;
        ret = platform_driver_register(&max_keypad_device_driver);//modify by zengxiany
        if(ret < 0)
        {
                printk("max_keypad_device driver init error!\n");
                return -ENODEV;
        }
    return 0;
}

static void __exit keypad_module_exit(void)
{
        platform_driver_unregister(&max_keypad_device_driver);
}
```

OK,这样就完成了设备的注册!