[L20]Device Tree 快速入门

之前我们已经做过一个简单的驱动,不过对Device Tree似乎还是有点疑问,实际上Device Tree除了内核用,U-Boot也会读取,各种驱动也要用,而且可以让内核变干净,真的是一举N得.

最简单的入门就是实践,我们用来实践的源:

https://github.com/nickfox-taterli/xlnx-eco/blob/4f5a6abb8d661ba7d01ef14aa22c9b09520dd264/arch/arm/boot/dts/zynq-7000.dtsi

https://github.com/nickfox-taterli/xlnx-eco/blob/4f5a6abb8d661ba7d01ef14aa22c9b09520dd264/arch/arm/boot/dts/zynq-zc702.dts

为什么有dtsi和dts,dtsi可以理解成是C语言的.h文件,用来被别人包含的,当然实际上在dts上没那么严格,dtsi描述的和dts其实描述的也是差不多的东西,如果熟悉JSON一定会感觉到,这东西真像JSON,分级分层,每个节点的格式其实都是这样的.

```
[label:]node-name[@unit-address] {
  [some prop(s)]
  [child node(s)]
};
```

其中[]是可选的,比如PMU节点,他位于f8891000地址,看UG585能验证到,其中interrupts是两个数组,第一个数组有三个元素0,5,4,第二个数组有三个元素0,6,4.interrupt-parent引用intc这个lable,pmu这个节点就没有lable,就并不好引用.

```
pmu@f8891000 {
    compatible = "arm,cortex-a9-pmu";
    interrupts = <0 5 4>, <0 6 4>;
    interrupt-parent = <&intc>;
    reg = <0xf8891000 0x1000>,
          <0xf8893000 0x1000>;
  };
```

继续翻到intc的label,看到他定义的是中断控制器.

```
intc: interrupt-controller@f8f01000 {
      compatible = "arm,cortex-a9-gic";
      #interrupt-cells = <3>;
      interrupt-controller;
      reg = <0xF8F01000 0x1000>,
            <0xF8F00100 0x100>;
    };
```

还有一些引用带有index,类似一个数组一样定义.

```
uart0: serial@e0000000 {
      compatible = "xlnx,xuartps", "cdns,uart-r1p8";
      status = "disabled";
      clocks = <&clkc 23>, <&clkc 40>;
      clock-names = "uart_clk", "pclk";
      reg = <0xE0000000 0x1000>;
      interrupts = <0 27 4>;
    };
```

他引用的节点在slcr下的clkc,第23个元素正好是uart0,40正好是uart0_aper,推测可知uart0工作需要依赖这两个时钟.这里有个比较特别的情况,clock-names刚好也是个数组,也是2个元素,而clocks刚好也是两个元素(每个元素又包含2个元素),所以程序中只要获取uart_clk这个时钟,就自动对应clocks的时钟.

```
clkc: clkc@100 {
        u-boot,dm-pre-reloc;
        #clock-cells = <1>;
        compatible = "xlnx,ps7-clkc";
        fclk-enable = <0xf>;
        clock-output-names = "armpll", "ddrpll", "iopll", "cpu_6or4x",
            "cpu_3or2x", "cpu_2x", "cpu_1x", "ddr2x", "ddr3x",
            "dci", "lqspi", "smc", "pcap", "gem0", "gem1",
```

```
            "fclk0", "fclk1", "fclk2", "fclk3", "can0", "can1",
            "sdio0", "sdio1", "uart0", "uart1", "spi0", "spi1",
            "dma", "usb0_aper", "usb1_aper", "gem0_aper",
            "gem1_aper", "sdio0_aper", "sdio1_aper",
            "spi0_aper", "spi1_aper", "can0_aper", "can1_aper",
            "i2c0_aper", "i2c1_aper", "uart0_aper", "uart1_aper",
            "gpio_aper", "lqspi_aper", "smc_aper", "swdt",
            "dbg_trc", "dbg_apb";
        reg = <0x100 0x100>;
      };
```

pinctrl在Zynq是特殊的存在,因为实际上他不需要,因为fsbl已经配置好了,但是这里也配置了,因为他同时需要slcr管理,在之前的教程中有看到这样的代码片段,define在dts中也是宏定义.

```
/include/ "system-conf.dtsi"

#define GPIO_ACTIVE_HIGH 0
#define GPIO_ACTIVE_LOW  1

/ {
    model = "Navigator Development Board";
    compatible = "zynq7010,zynq-7020","xlnx,zynq-7000";
```

那如何覆盖和继承配置呢,比如我这里system-user.dtsi实际上在文件最后声明的,所以他将覆盖全部配置,即后面覆盖前面,追加也是一样道理,当然依然要遵守从根节点开始往下寻址,相信通过几个简单的例子,应该就能理解各种dts文件了,那么现在看看设备树常用的函数,还是看之前我们自己写的LED内核驱动代码.

```
// ... 省略
static int __init led_init(void){
    const char *str;
    int ret;

    // 下面两个用的不多但是要了解一下
    // of_get_parent("/led"); // 查找父节点
    // of_get_parent("/","/led"); // 查找/led的下一个子节点

    // 下面这个用于总线类设备
    // of_address_of_resource => 以后用到总线时候会讲!

    /* 新增的从dts获取数据的过程 */
    dev.nd = of_find_node_by_path("/led"); // 按路径找,精确.
    // of_find_node_by_name("led"); // 防范按名字找,方便.
    // of_find_node_by_type("iodev"); // 按device_type属性找.
    if(dev.nd == NULL){
        return -EINVAL;
    }

    ret = of_property_read_string(dev.nd,"status",&str);
    // of_find_property("/led","status",sizeof("status")); // 确定能否找到属性
    // of_property_count_elems_of_size("/led","gpios",elem_size); // 获取数组大小
    // of_property_read_u32_index("/led","gpios",0,&out); // 读数组Index 0的U32格式数值.(其他类似函数能推导出来!)
    // of_property_read_u32_array("/led","gpios",&out,5); // 读取这个数组5个元素并存out指针内.(其他类似函数能推导出来!)
    if(ret < 0){
        return -EINVAL;
    }

    if(strcmp(str,"okay")){
        return -EINVAL;
    }

    ret = of_property_read_string(dev.nd,"compatible",&str);
    // 这里例子中的方法并不好,建议用下面的方法.
    // of_find_compatible_node("/led","iodev","taterli,led"); // 找/led节点下,device_type为iodev,且compatible为taterli,led,如果找不到返回NULL,前
    // of_find_matching_node_and_match("/led",&my_dev_ids,&matched); // 用兼容表找,比如你的驱动兼容很多个不同的compatible.
    if(ret < 0){
        return -EINVAL;
    }

    if(strcmp(str,"taterli,led")){
        return -EINVAL;
    }

    /* IO当然也可以是一个数组 */
    dev.gpio = of_get_named_gpio(dev.nd,"led-gpio",0);
```

```
    if(!gpio_is_valid(dev.gpio)){
        /* IO是独占资源,因此可能申请失败! */
        return -EINVAL;
    }

    /* 申请IO并给一个名字 */
    ret = gpio_request(dev.gpio,"taterli-kernel-led");
    if(ret < 0){
        /* 除了返回EINVAL,也可以返回上一层传递的错误. */
        return ret;
    }

    ret = of_property_read_string(dev.nd,"default-state",&str);
    if(ret < 0){
        return -EINVAL;
    }

    if(!strcmp(str,"on")){
         /* 设置输出和默认电平 */
        gpio_direction_output(dev.gpio,1);
    }else if(!strcmp(str,"off")){
        gpio_direction_output(dev.gpio,0);
    }else{
        return -EINVAL;
    }

// ... 省略

}
```