EastonWoo的专栏

一天50行代码 一天改一个bug 一天一笔记

基于linux 3.10.49内核的pinctrl流程分析

2016-05-23 14:18 219人阅读 评论(0) 收藏 举报

分类：

linux内核驱动（6）

基于linux 3.10.49内核的pinctrl流程分析

linux kernel 3.10.49+

pinctrl驱动的platform注册就不说了,

无非就是platform_driver_register这个入口,

最后匹配到合适的设备后调用struct platform_driver 的probe函数.

这里说说, pinctl io复用关系(pinmux)的是怎么通过device tree source(dts)设置的.

1. 首先,当然是看pinctrl驱动的probe函数(这相当于驱动初始化的入口):

   drivers/pinctrl/pinctrl-xxxxxx.c : xxxxxx_pinctrl_probe(...)

   xxxxxx_pinctrl_probe(...) --> pinctrl_register(...)

2. drivers/pinctrl/core.c : pinctrl_register(...)

   pinctrl_register(...) --> pinctrl_get(...)

3. drivers/pinctrl/core.c : pinctrl_get(...)

   pinctrl_get(...) --> create_pinctrl(...)

4. drivers/pinctrl/core.c : create_pinctrl(...)

   create_pinctrl(...) --> pinctrl_dt_to_map(...)

5. drivers/pinctrl/devicetree.c : pinctrl_dt_to_map(...)

   pinctrl_dt_to_map(...) --> dt_to_map_one_config(...)

6. drivers/pinctrl/devicetree.c : dt_to_map_one_config(...)

   dt_to_map_one_config(...) --> ops->dt_node_to_map(...) // 回调函数. 第7~10步是进入到回调后的一系列初始化.

7. ops->dt_node_to_map 就是 drivers/pinctrl/pinctrl-xxxxxx.c 中 struct pinctrl_ops 的 dt_node_to_map成员函数指针,

   也就是struct pinctrl_ops xxxxxx_pinctrl_ops->dt_node_to_map = xxxxxx_pinctrl_dt_node_to_map;

8. drivers/pinctrl/pinctrl-xxxxxx.c : xxxxxx_pinctrl_dt_node_to_map(...)

   xxxxxx_pinctrl_dt_node_to_map(...) --> 轮询调用 xxxxxx_pinctrl_dt_subnode_to_map(...)

   轮询的内容: struct device_node *np 其中一个就是 dts文件里pinmuxing node设备, 而pinmuxing node设备有7个子node(看例子):

   如: arm926u, i2c0, i2c1等.

```
state_default: pinmuxing {
    arm926u {
```

- QT5 qtcreator 加入qwt画图插件(7534)
- Linux命令之ar - 创建静态库.a文件和动态库.so(6525)
- arm汇编编程(示例)(5545)
- linux 远程本地端口映射(4818)
- s3c2440 地址分配硬件连接及其启动原理分析(3985)
- gcc 编译的四大过程（预处理-编译-汇编-链接 ）(3009)
- namespace命名空间成员类的声明(2642)
- arm汇编编程 简单例子(2606)

评论排行

- QT5 r 加入qwtplot3d 三维库(6)
- VS2008 动态库和静态库的生成和加载(6)
- s3c2440 地址分配硬件连接及其启动原理分析(4)
- QT5 qtcreator 加入qwt画图插件(1)
- Linux命令之ar - 创建静态库.a文件和动态库.so(1)
- 公用规则用升级版[3]Makefile 适用于中大型工程(1)
- 理解LDM和STM多寄存器寻址 堆栈指针sp例子(1)
- namespace命名空间成员类的声明(1)
- vfork 例子祥解(0)
- x86_64下多平台编译qt4.8.6(0)

推荐文章

- * 致JavaScript也将征服的物联网世界
- * 从苏宁电器到卡巴斯基：难忘的三年硕士时光
- * 作为一名基层管理者如何利用情商管理自己和团队（一）
- * Android CircleImageView圆形ImageView
- * 高质量代码的命名法则

最新评论

- Linux命令之ar - 创建静态库.a文件和动态库.so
  qq_34890287: 写的很好。很是感动。加油。拿出更好的心得。
- VS2008 动态库和静态库的生成和加载
  EastonWoo: @u010141025:谢谢
- VS2008 动态库和静态库的生成和加载
  EastonWoo: @wenzhang912418283:谢谢
- VS2008 动态库和静态库的生成和加载
  EastonWoo: @zkl99999:谢谢
- VS2008 动态库和静态库的生成和加载
  zkl99999: 不错
- QT5 r 加入qwtplot3d 三维库
  yuerongzhong: @yuerongzhong:这个是因为你没加qglu的头文件和库，库在D:\qt\Qt5.3.1\T...
- QT5 r 加入qwtplot3d 三维库
  scgaluo: @flyingknight:自己找到解决办法了才看到你这一段
- QT5 r 加入qwtplot3d 三维库


```
            xxx,function = "arm926u";
            xxx,group = "arm926u";
        };
        i2c0 {
            xxx,function = "i2c";
            xxx,group = "i2c0_pos_0";
        };
        i2c1 {
            xxx,function = "i2c";
            xxx,group = "i2c1";
        };
        i2c2 {
            xxx,function = "i2c";
            xxx,group = "i2c2";
        };
        uart0 {
            xxx,function = "uart";
            xxx,group = "uart0_pos_0";
        };
        uart1 {
            xxx,function = "uart";
            xxx,group = "uart1_pos_0";
        };
        wdt {
            xxx,function = "wdt";
            xxx,group = "wdt";
        };
    };
```

9. drivers/pinctrl/pinctrl-xxxxxx.c : xxxxxx_pinctrl_dt_subnode_to_map(...)
   在这里有:
   ret = of_property_read_string(np, "xxx,function", & function); // 哈哈, 解析dts node设备属性
   ret = of_property_read_string(np, "xxx,group", &group);
   reserve_map(...) // allocate map内存
   最后调用 add_map_mux(...);

10. drivers/pinctrl/pinctrl-xxxxxx.c : add_map_mux(...)
   (*map)[*num_maps].type = PIN_MAP_TYPE_MUX_GROUP; //特别注意这个type, 后续用到
   (*map)[*num_maps].data.mux.group = group; // group只是个字符串, 如:i2c0_pos_0
   (*map)[*num_maps].data.mux.function = function; // function只是个字符串, 如:i2c
   (*num_maps)++;

11. 回调ops->dt_node_to_map完成, 回到第6步继续运行.
   drivers/pinctrl/devicetree.c : 运行dt_remember_or_free_map(...);

12. drivers/pinctrl/devicetree.c : dt_remember_or_free_map(...)
   dt_remember_or_free_map(...) --> pinctrl_register_map(...)

13. drivers/pinctrl/core.c : pinctrl_register_map(...)
   判断一下PIN_MAP_TYPE_MUX_GROUP
   pinctrl_register_map(...) --> list_add_tail(&maps_node->node, &pinctrl_maps) // pinctrl_maps 是全局变量:
LIST_HEAD(pinctrl_maps);

######### 到这里, create_pinctrl(...) 的 pinctrl_dt_to_map(...) 函数已运行完成. #########

14. 回调pinctrl_dt_to_map完成, 回到第4步继续运行. 还是在create_pinctrl(...)函数里.
   create_pinctrl(...) --> add_setting(...)

15. drivers/pinctrl/core.c : add_setting(...)

```
switch (map->type) {
case PIN_MAP_TYPE_MUX_GROUP:
    ret = pinmux_map_to_setting(map, setting);
    break;
    }
```

add_setting(...) --> pinmux_map_to_setting(...)

16. drivers/pinctrl/pinmux.c : pinmux_map_to_setting(...)

pinmux_map_to_setting(...) --> pmxops->get_function_groups(...)

17. pmxops->get_function_groups(...) 就是 drivers/pinctrl/pinctrl-xxxxxx.c 中 struct pinmux_ops 的 get_function_groups成员函数指针,

也就是struct pinmux_ops xxxxxx_pinmux_ops->get_function_groups = xxxxxx_get_groups;

在xxxxxx_get_groups(...)函数里:可以取到 xxxxx_groups[] = {...};的字符串数值.

```
ret = pinmux_func_name_to_selector(pctldev, map->data.mux.function);
// pinmux_func_name_to_selector(...) {  // 函数实现
// ...
//     while (selector < nfuncs) {
//         const char *fname = ops->get_function_name(pctldev,
//                 selector);          // ops->get_function_name 就是struct pinmux_ops的get_function_name 取到 i2c

//         if (!strcmp(function, fname))   // function: map->data.mux.function这个就是dts的 xxx,function = "i2c";
//             return selector;  // fname:得到const struct xxxxxx_function xxxxxx_functions[] 第几个是i2c,
                                  // 所以, 代码里的 function 与 group 必须是一一对应的.

//         selector++;
//     }
// }
// ...
setting->data.mux.func = ret;
ret = pmxops->get_function_groups(pctldev, setting->data.mux.func,
            &groups, & num_groups);
    // groups 等于 fname##_groups 如:const char * const i2c_groups[] = { "i2c0_pos_0", "i2c0_pos_1", "i2c1", "i2c2"}
    group = map->data.mux.group;   // xxx,group = "i2c0_pos_0";   //注意: 凡是map相关的, 很可能是dts的内容
    for (i = 0; i < num_groups; i++) {
        if (!strcmp(group, groups[i])) {
            found = true;
            break;
        }
    }
// dts的 类似于xxx,group = "i2c0_pos_0" 与通过get_function_groups获取到代码的 const struct xxxxxx_function xxxxxx_functions[]全局变量.
// 两个作对比, 得到setting->data.mux.group;

ret = pinctrl_get_group_selector(pctldev, group);
// pinctrl_get_group_selector(struct pinctrl_dev *pctldev,
            const char *pin_group) {  // 函数实现
// ...
// while (group_selector < ngroups) {
//     const char *gname = pctlops->get_group_name(pctldev,
//                 group_selector);  // pctlops->get_group_name 就是 struct pinctrl_ops的get_group_name
//     if (!strcmp(gname, pin_group)) {  // pin_group: 是 xxx,group = "i2c0_pos_0";
```

```
                                // gname: 是 代码里的全局变量 const struct xxxxxx_group xxxxxx_groups[];
    //        dev_dbg(pctldev->dev,
    //                "found group selector %u for %s\n",
    //                group_selector,
    //                pin_group);
    //        return group_selector;
    //    }

    //    group_selector++;
    // }
    // ...

    setting->data.mux.group = ret;  // 得到setting->data.mux.group,后面给xxxxxx_enable使用(设置控制寄存器)
    // 相当于, 我只要在dts中设置xxx,group = "i2c0_pos_0", pinctrl子系统就会找到对应的寄存器设置.


######### 到这里, create_pinctrl(...) 的 add_setting(...) 函数已运行完成. #########


######### 到这里, pinctrl_register(...) 的 pinctrl_get(...) 函数已运行完成. #########

18. 回调pinctrl_get完成, 回到第2步继续运行. 还是在pinctrl_register(...)函数里.
    pinctrl_register(...) --> pinctrl_select_state(...)
    switch (setting->type) {
        case PIN_MAP_TYPE_MUX_GROUP:   // 用到了.
            ret = pinmux_enable_setting(setting);
            break;
            ...
            }

19. drivers/pinctrl/pinmux.c : pinmux_enable_setting(...)
    pinmux_enable_setting(...) --> ops->enable(...) // 哈哈, 又是ops,回调函数,第19~20步是进入到回调后的一系列初始化.
    ret = ops->enable(pctldev, setting->data.mux.func,
              setting->data.mux.group);


20. ops->enable 就是 drivers/pinctrl/pinctrl-xxxxxx.c 中 struct pinmux_ops 的 enable成员函数指针,
    也就是struct pinmux_ops xxxxxx_pinmux_ops->enable  = xxxxxx_enable;
    在xxxxxx_enable(...)函数里:可以设置控制寄存器的值, 作为IO复用设置(这函数是内核运行完成后,应用程序修改时用到的.), 用作gpio, 还是i2c等.

######### 到这里, pinctrl_register(...) 的 pinctrl_select_state(...) 函数已运行完成. #########




##################################################################################################
重要结构体:
1). struct pinctrl_desc: struct platform_driver的probe函数的pinctrl_register需要使用到
                    struct pinctrl_dev *pinctrl_register(struct pinctrl_desc *pctldesc,
                                struct device *dev, void *driver_data);

2). struct pinctrl_ops: struct pinctrl_desc结构体需要使用到.
    .get_groups_count = 需赋函数指针, --|
    .get_group_name = 需赋函数指针,    | 这三个函数, 需要生成一个全局变量的结构体, 指定gpio引脚控制寄存器的位移和组需要用到的引脚号.
```

.get_group_pins = 需赋函数指针, --|
.dt_node_to_map = 需赋函数指针, ------|
.dt_free_map = 需赋函数指针, ------| 这两个是解析dts文件的复用信息pinmuxing到内存map 和 析构map

3). struct pinmux_ops: struct pinctrl_desc结构体需要使用到.
.gpio_request_enable = 需赋函数指针,
.gpio_disable_free = 需赋函数指针,
.request = 需赋函数指针,
.free = 需赋函数指针,
.get_functions_count = 需赋函数指针, --|
.get_function_name = 需赋函数指针, --| 这三个函数, 是直接解析dts, 用到的function 和 group, 通过function名字, 找到 group的名字(可能有多个)
.get_function_groups = 需赋函数指针, --| 再通过 group 的名字去找到gpio引脚控制寄存器的信息, 最后通过enable回调函数设置.
.enable = 需赋函数指针,
.disable = 需赋函数指针,

4). struct pinctrl_pin_desc 所有引脚号.
5). 自定义group结构体:
5-1). group的名字(也就是io复用的名字)(dts编写时, 需要在这里找得到),
5-2). 复用io数(如i2c, 需要使用两个io).
5-3). 控制寄存器地址, 和使用哪几位.
6). 自定义function结构体:
6-1). function的名字(dts编写时, 需要在这里找得到),
6-2). group的名字(可以有很多个, 如i2c1, i2c2等, 但也必须在自定义group结构体里找得到)

##
pinctrl 运行原理:
1) 读取dts: 先读入dts的pinmuxing节点的信息到map;
2) dts的子节点的function 和 自定义function结构体的function的名字匹配, 得到自定义function结构体的下标, 放入Setting变量的func:
3) 由自定义function结构体的下标, 得到自定义function结构体的group的名字和数量;
4) 判断 dts的子节点的group 是否在自定义function结构体的group的名字里面, 如果是, 运行第5步, 否就运行第二步匹配dts下一个子节点的function;
5) dts的子节点的group 和 自定义group结构体的group的名字匹配, 得到自定义group结构体的下标, 放入Setting变量的group;
6) 通过Setting变量的func和group这两个下标调用struct pinmux_ops的enable回调函数. 哈哈, 络于可以设置寄存器了.