Linux common clock framework(1)_概述

作者：wowo 发布于：2014-10-20 23:06 分类：电源管理子系统

1. 前言

common clock framework是用来管理系统clock资源的子系统，根据职能，可分为三个部分：

1）向其它driver提供操作clocks的通用API。

2）实现clock控制的通用逻辑，这部分和硬件无关。

3）将和硬件相关的clock控制逻辑封装成操作函数集，交由底层的platform开发者实现，由通用逻辑调用。

因此，蜗蜗会将clock framework的分析文章分为3篇：

第一篇为概述和通用API的使用说明，面向的读者是使用clock的driver开发者，目的是掌握怎么使用clock framework（就是本文）；

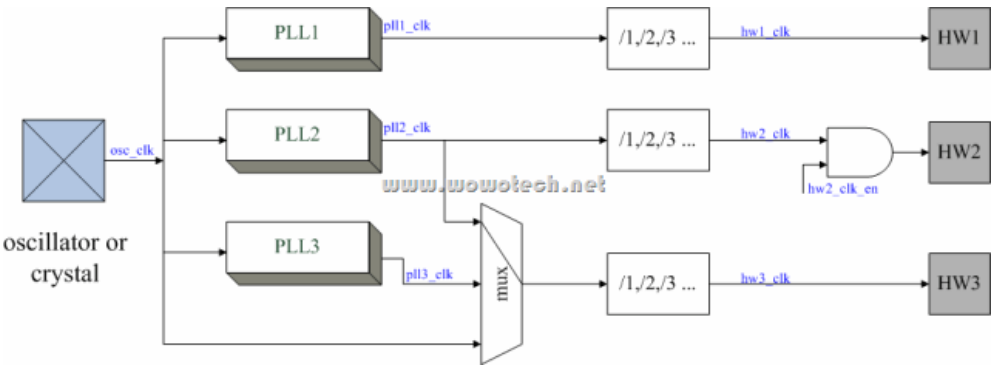
第二篇为底层操作函数集的解析和使用说明，面向的读者是platform clock driver的开发者，目的是掌握怎么借助clock framework管理系统的时钟资源；

第三篇为clock framework的内部逻辑解析，面向的读者是linux kernel爱好者，目的是理解怎么实现clock framework。

注1：任何framework的职能分类都是如此，因此都可以按照这个模式分析。

2. 概述

如今，可运行Linux的主流处理器平台，都有非常复杂的clock tree，我们随便拿一个处理器的spec，查看clock相关的章节，一定会有一个非常庞大和复杂的树状图，这个图由clock相关的器件，以及这些器件输出的clock组成。下图是一个示例：



clock相关的器件包括：用于产生clock的Oscillator（有源振荡器，也称作谐振荡器）或者Crystal（无源振荡器，也称晶振）；用于倍频的PLL（锁相环，Phase Locked Loop）；用于分频的divider；用于多路选择的Mux；用于clock enable控制的与门；使用clock的硬件模块（可称作consumer）；等等。

站内搜索

请输入关键词 搜索

功能

留言板
评论列表
支持者列表

最新评论

tigerbear
请教个问题， 1. 你提到kswapd会周期性的回收，并且...
fengzi
大侠，求教一下。linux待机恢复的时候是从UBOOT哪里跳...
Nicole
正在学习。。。
泥人
资料不公开，是否可以问个问题？！呵呵 AD6900有片上B...
Physh
大侠，求教一个问题：我用perf top分析一个arm-li...
ailon
在多路服务器中，每个CPU都有一个uart控制器，在CPU角...

文章分类

Linux内核分析(11)
统一设备模型(15)
电源管理子系统(42)
中断子系统(15)
进程管理(17)
内核同步机制(18)
GPIO子系统(5)
时间子系统(14)
通信类协议(7)
内存管理(27)
图形子系统(1)
文件系统(4)
TTY子系统(6)
u-boot分析(3)
Linux应用技巧(13)
软件开发(6)
基础技术(13)
蓝牙(16)
ARMv8A Arch(13)
显示(3)

common clock framework的管理对象，就是上图蓝色字体描述的clock（在软件中用struct clk抽象，以后就简称clk），主要内容包括（不需要所有clk都支持）：

1）enable/disable clk。

2）设置clk的频率。

3）选择clk的parent，例如hw3_clk可以选择osc_clk、pll2_clk或者pll3_clk作为输入源。

3. common clock framework提供的通用API

管理clock的最终目的，是让device driver可以方便的使用，这些是通过include/linux/clk.h中的通用API实现的，如下：

1）struct clk结构

一个系统的clock tree是固定的，因此clock的数目和用途也是固定的。假设上面图片所描述的是一个系统，它的clock包括osc_clk、pll1_clk、pll2_clk、pll3_clk、hw1_clk、hw2_clk和hw3_clk。我们完全可以通过名字，抽象这7个clock，进行开/关、rate调整等操作。但这样做有一个缺点：不能很好的处理clock之间的级联关系，如hw2_clk和hw3_clk都关闭后，pll2_clk才能关闭。因此就引入struct clk结构，以链表的形式维护这种关系。

同样的道理，系统的struct clk，也是固定的，由clock driver在系统启动时初始化完毕，需要访问某个clock时，只要获取它对应的struct clk结构即可。怎么获取呢？可以通过名字索引啊！很长一段时间内，kernel及driver就是使用这种方式管理和使用clock的。

最后，设备（由struct device表示）对应的clock（由struct clk表示）也是固定的啊，可不可以找到设备就能找到clock？可以，不过需要借助device tree。

注2：对使用者（device driver）来说，struct clk只是访问clock的一个句柄，不用关心它内部的具体形态。

2）clock获取有关的API

device driver在操作设备的clock之前，需要先获取和该clock关联的struct clk指针，获取的接口如下：

```
1: struct clk *clk_get(struct device *dev, const char *id);
2: struct clk *devm_clk_get(struct device *dev, const char *id);
3: void clk_put(struct clk *clk);
4: void devm_clk_put(struct device *dev, struct clk *clk);
5: struct clk *clk_get_sys(const char *dev_id, const char *con_id);
6:
7: struct clk *of_clk_get(struct device_node *np, int index);
8: struct clk *of_clk_get_by_name(struct device_node *np, const char *name);
9: struct clk *of_clk_get_from_provider(struct of_phandle_args *clkspec);
```

> a）clk_get，以device指针或者id字符串（可以看作name）为参数，查找clock。
>
> a1）dev和id的任意一个可以为空。如果id为空，则必须有device tree的支持才能获得device对应的clk；
> a2）根据具体的平台实现，id可以是一个简单的名称，也可以 是一个预先定义的、唯一的标识（一般在平台提供的头文件中定义，如mach/clk.h）；
> a3）不可以在中断上下文调用。
>
> b）devm_clk_get，和clk_get一样，只是使用了device resource management，可以自动释放。
>
> c）clk_put、devm_clk_put，get的反向操作，一般和对应的get API成对调用。
>
> d）clk_get_sys，类似clk_get，不过使用device的name替代device结构。
>
> e）of_clk_get、of_clk_get_by_name、of_clk_get_from_provider，device tree相关的接口，直接从相应的DTS node中，以index、name等为索引，获取clk，后面会详细说明。

3）clock控制有关的API

```
1: int clk_prepare(struct clk *clk)
2: void clk_unprepare(struct clk *clk)
```


随机文章

文章存档

```
3:
4: static inline int clk_enable(struct clk *clk)
5: static inline void clk_disable(struct clk *clk)
6:
7: static inline unsigned long clk_get_rate(struct clk *clk)
8: static inline int clk_set_rate(struct clk *clk, unsigned long rate)
9: static inline long clk_round_rate(struct clk *clk, unsigned long rate)
10:
11: static inline int clk_set_parent(struct clk *clk, struct clk *parent)
12: static inline struct clk *clk_get_parent(struct clk *clk)
13:
```

> a）clk_enable/clk_disable，启动/停止clock。不会睡眠。
>
> b）clk_prepare/clk_unprepare，启动clock前的准备工作/停止clock后的善后工作。可能会睡眠。
>
> c）clk_get_rate/clk_set_rate/clk_round_rate，clock频率的获取和设置，其中clk_set_rate可能会不成功（例如没有对应的分频比），此时会返回错误。如果要确保设置成功，则需要先调用clk_round_rate接口，得到和需要设置的rate比较接近的那个值。
>
> d）获取/选择clock的parent clock。
>
> e）clk_prepare_enable，将clk_prepare和clk_enable组合起来，一起调用。clk_disable_unprepare,将clk_disable和clk_unprepare组合起来，一起调用。

注2：prepare/unprepare，enable/disable的说明。

这两套API的本质，是把clock的启动/停止分为atomic和non-atomic两个阶段，以方便实现和调用。因此上面所说的“不会睡眠/可能会睡眠”，有两个角度的含义：一是告诉底层的clock driver，请把可能引起睡眠的操作，放到prepare/unprepare中实现，一定不能放到enable/disable中；二是提醒上层使用clock的driver，调用prepare/unprepare接口时可能会睡眠哦，千万不能在atomic上下文（例如中断处理中）调用哦，而调用enable/disable接口则可放心。

另外，clock的开关为什么需要睡眠呢？这里举个例子，例如enable PLL clk，在启动PLL后，需要等待它稳定。而PLL的稳定时间是很长的，这段时间要把CPU交出（进程睡眠），不然就会浪费CPU。

最后，为什么会有合在一起的clk_prepare_enable/clk_disable_unprepare接口呢？如果调用者能确保是在non-atomic上下文中调用，就可以顺序调用prepare/enable、disable/unprepared，为了简单，framework就帮忙封装了这两个接口。

4）其它接口

```
1: int clk_notifier_register(struct clk *clk, struct notifier_block *nb);
2: int clk_notifier_unregister(struct clk *clk, struct notifier_block *nb);
```

> 这两个notify接口，用于注册/注销 clock rate改变的通知。例如某个driver关心某个clock，期望这个clock的rate改变时，通知到自己，就可以注册一个notify。后面会举个例子详细说明。

```
1: int clk_add_alias(const char *alias, const char *alias_dev_name, char *id,
2:                   struct device *dev);
```

> 这是一个非主流接口，用于给某个clk起个别名。无论出于何种原因，尽量不要它，保持代码的简洁，是最高原则！

4. 通用**API**的使用说明

结合一个例子（摘录自“Documentation/devicetree/bindings/clock/clock-bindings.txt”），说明driver怎么使用clock通用API。

1）首先，在DTS（device tree source）中，指定device需要使用的clock，如下：

2014年5月(9)
2014年4月(9)
2014年3月(7)
2014年2月(3)
2014年1月(4)

```
/* DTS */
device {
    clocks = <&osc 1>, <&ref 0>;
    clock-names = "baud", "register";
};
```

> 该DTS的含义是：
>
> device需要使用两个clock，“baud”和“regitser”，由clock-names关键字指定；
>
> baud取自“osc”的输出1，register取自“ref”的输出0，由clocks关键字指定。
>
> 那么问题来了，clocks关键字中，<&osc 1>样式的字段是怎么来的？是由clock的provider，也就是底层clock driver规定的（具体会在下一篇文章讲述）。所以使用clock时，一定要找clock driver拿相关的信息（一般会放在“Documentation/devicetree/bindings/clock/”目录下）。

2）系统启动后，device tree会解析clock有关的关键字，并将解析后的信息放在platform_device相关的字段中。

3）具体的driver可以在probe时，以clock的名称（不提供也行）为参数，调用clk get接口，获取clock的句柄，然后利用该句柄，可直接进行enable、set rate等操作，如下：

```
/* driver */
int xxx_probe(struct platform_device *pdev)
{
        struct clk *baud_clk;
        int ret;

        baud_clk = devm_clk_get(&pdev->dev, “baud”);
        if (IS_ERR(clk)) {
                …
        }

        ret = clk_prepare_enable(baud_clk);
        if (ret) {
```

原创文章，转发请注明出处。蜗窝科技，www.wowotech.net。

标签: **Linux framework clock API**

« linux内核同步 | Linux内核同步机制之（二）：Per-CPU变量»

评论：

lolo
2016-12-05 10:56

Dear sir:
我最近也需要在linux下配置一个时钟，找了好多有关如何调用clk配置的api，发现现在linux下面针对所有的clk都是用同一的接口去管理了，难怪底层的驱动都不提供用户调用接口了，而是都封装成 common clock framework调用的统一接口了；
我现在在做这个的时候碰到了一个问题，就是只要程序包含 #include <linux/clk.h> 就会报错，我找了下我装的交叉编译工具 arm-linux-gnueabihf-gcc 工具下的include 目录确实没有包含 clk.h这个文件，但是我参考的xilinx的内核文件下是有这一系列文件的，请问下我该如何处理呢？ 我尝试着把文件直接copy到交叉编译工具的目录下面，但是还是会报其他错误，因为内核结构又很多都不一样；

请问楼主有没有碰到这样的问题？

回复

wowo
2016-12-05 11:21

@lolo：为什么要放到交叉编译工具目录呢？这个头文件是kernel的啊

回复

lolo84
2016-12-06 12:57

@wowo：但是怎么解决写应用程序包含头文件报错的问题呢？ 我现在不是要编内核 我是想要写应用程序 。我包含 linux/fb.h 就不会报错 因为我找交叉编译工具的include目录下面确实有fb.h ,但是我包含clk.h 它就会报没有这个文件 显然是编译器没有找到这个文件。

回复

wowo

2016-12-06 13:23

@lolo84：用户空间的代码为什么要包含clk.h呢？这是不允许的。
用户空间能够包含的头文件，都在include/uapi/linux/里面，你可以看看。

回复

lolo84

2016-12-08 17:53

@wowo：你好，我现在就是想在linux下动态的配置一个时钟输出，频率为， 一直在器件驱动下面找可以直接调用的api，最后看了您的文章才知道所有的时钟都是在common clock framework框架下去做了;
那么我就想知道标准的用户接口放在那个头文件里面了呢？ 我以为是要包含clk.h , 但是clk.h确实不在 include/uapi/linux 路径下面;
楼主有应用层配置的代码可以发给我参考下吗？ 回头一定打赏下～

回复

wowo

2016-12-08 18:30

@lolo84：Linux kernel是不允许用户空间代码直接控制clock的（太危险了），因此没有提供用户空间的访问API。
如果有在用户空间动态配置时钟输出的需求，你可以试着将需求转换一下：不是直接控制clock，而是控制使用clock的那个设备，例如：
如果设备是cpu，则使用cpufreq framework提供的接口（博客里面有相关的文章）；
如果是外部设备的，则使用devfreq（drivers/devfreq）提供的接口（和cpufreq类似）。

回复

lolo84

2016-12-06 13:00

@wowo：我网上找到有直接找到交叉编译工具的源码直接包含kernel 然后编译针对kernel的交叉编译工具 ， 不知道我现在碰到的问题是不是也要这样去解决呢，但是那个过程貌似太复杂了

回复

vic

2015-10-12 17:32

请问您的 hw3_clk 要如何定义？

它的clock包括osc_clk、pll1_clk、pll2_clk、pll3_clk、hw1_clk、hw2_clk和hw3_clk。我们完全可以通过名字，抽象这7个clock，进行开/关、rate调整等操作。但这样做有一个缺点：不能很好的处理clock之间的级联关系，如hw2_clk和hw3_clk都关闭后，pll2_clk才能关闭。因此就引入struct clk结构，以链表的形式维护这种关系。

我想pll1_clk, pll2_clk, pll3_clk 应该是如下：

```
struct pll1_clk {
   struct clk_hw    hw;
   void __iomem    *reg;
};

struct pll2_clk {
   struct clk_hw    hw;
   void __iomem    *reg;
};

struct pll3_clk {
   struct clk_hw    hw;
   void __iomem    *reg;
};
```

回复

wowo

2015-10-12 21:34

@vic：抱歉，不是很明白您的意思，你能稍微详细的描述一下吗？

回复

vic

2015-10-13 09:39

@wowo：hw3_clk 的 device tree 要如何写， 才能表达 “如hw2_clk和hw3_clk都关闭后，pll2_clk才能关闭。因此就引入struct clk结构，以链表的形式维护这种关系”。

是如下吗?

```
struct hw3_clk {
    struct clk_hw    hw;
    void __iomem    *reg;
};

struct pll2_clk {
    struct clk_hw    hw;
    void __iomem    *reg;
    struct clk      *hw3_clk;
};
```

回复

wowo
2015-10-13 13:38

@vic：您可以参考“http://www.wowotech.net/pm_subsystem/clock_provider.html”，在注册clock的时候，指定parent即可。

回复

vic
2015-10-14 14:35

@wowo：我知道了。例如我在device tree定义：

```
hw3_clk: hw3_clk {
    #clock-cells = <0>;
    compatible = "hw3-clk";
    clocks = <&pll2 0>,<&pll3 0>;
    clock-output-names = "hw3_clk";
};
```

然后：

```
of_property_read_string(node, "clock-output-names", &clk_name);
parent_name[0] = of_clk_get_parent_name(node, 0);
parent_name[1] = of_clk_get_parent_name(node, 1);

clk = clk_register_mux(NULL, clk_name, parent_name,
      ARRAY_SIZE(parent_name), 0,
      regs_base , offset_bit, one_bit, 0, NULL);
```

应该是这样吧。谢谢。

回复

kkajm
2015-08-26 16:14

终于找到大神了，对芯片级的分析超透彻，本人也是做芯片级bringup，但相对楼主的深度 这是自愧不如啊

回复

wowo
2015-08-26 21:19

@kkajm：您这真是过奖了的，不敢当。欢迎常来，多交流。

回复

lucky
2015-09-15 20:02

@wowo：大家建一个群怎样？

回复

lucky
2015-09-15 20:02

@kkajm：大家建一个群怎样？

回复

wowo
2015-09-15 22:44

@lucky：其实现在有一个群：http://wowotech.net/contact_us.html
只是很少在里面回答大家的问题，总觉得不甚好意思，呵呵……

回复

linuxer
2015-09-15 22:48

@wowo：悄悄问一句：没有QQ的能加入群吗？

回复

wowo
2015-09-15 22:50

@linuxer：晕……，linuxer同学你暴露了你的“年龄”了啊，恐龙级的人物啊，哈哈。

回复

wzw200
2015-03-06 08:47

一口气把这3篇文章看完了，以前只知道有一个时钟树，用GET SET方法，跟硬件联系之类的，很模糊！
现在算是看明白了，真是感谢！

回复

xiaogao
2015-02-02 10:38

谢谢作者，分析的太棒了，已转载并注明原文出处

回复

xvzhaobin
2014-11-05 15:09

你好，注2中的“另外，clock的enable/disable为什么需要睡眠呢？”是不是写错了？enable/disable是不能睡眠的，prepare/unprepare是可以睡眠的；这个地方不理解

回复

wowo
2014-11-05 15:44

@xvzhaobin：不好意思，我想说的是一个泛指（指clock的开关），而不是具体的enable/disable接口。修改一下表述会好一些。谢谢提醒。

回复

xvzhaobin
2014-11-05 16:16

@wowo：谢谢哈

回复

xvzhaobin
2014-11-05 16:27

@wowo：Linux common clock framework第三篇文章什么时候出啊？期待

回复

wowo
2014-11-05 16:30

@xvzhaobin：该我谢谢您才是。
我还正犹豫要不要立即写第三篇呢，因为我觉得90%的人应该不关心第三篇，所以想稍微往后推推。不过我会考虑您的建议。谢谢。

回复

xvzhaobin
2014-11-05 16:35

@wowo：很是期待，

回复

heavenward
2016-03-01 10:17

@wowo：我们还是很关心第三篇的，我觉得第三部分才是精华，能更清楚的明白透彻

回复

wowo
2016-03-01 10:43

@heavenward：第三篇已经有了，请参考：
http://www.wowotech.net/pm_subsystem/clock_framework_core.html，谢谢

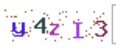
回复

发表评论：

昵称

邮件地址 (选填)

个人主页 (选填)



发表评论