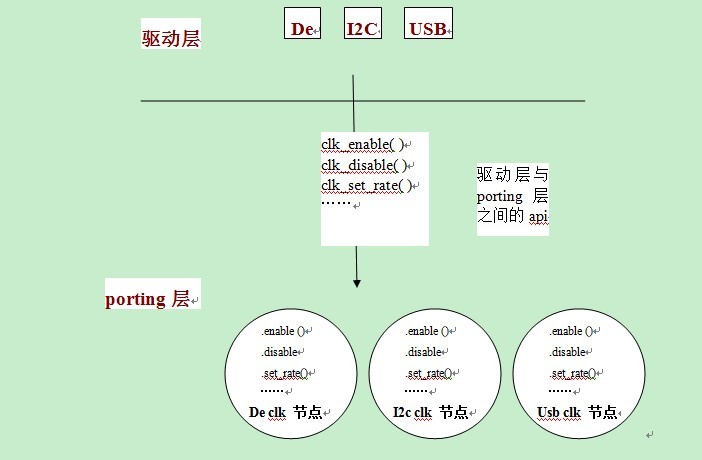
**Linux clk 模型**

Linux clk模型采用面向对象的思想来设计实现的。

在**porting层**创建一个一个的clk节点对象，然后将所有的clk节点对象连成一个list。

当**驱动层**需要设置时钟的时候，通过**porting层**与驱动层直接的api函数进行操作。首先通过clk\_get函数，根据clk节点的名字，获取clk节点。然后，使用clk\_set\_rate()函数设置clk节点的时钟。clk\_set\_rate() 函数最终将会调用clk节点对象的成员函数 clk->set\_rate() 设置时钟。

层次关系如下图所示：



**1. 驱动层与porting层之间的api**

驱动层与porting层之间的api，定义在include/linux/clk.h文件中

        // clk的结构体，空的，

// 正真的实现在arch\arm\plat-vc088x\include\plat\clock.h

struct clk;

        // 根据name获得注册过的clk

struct clk \*clk\_get(struct device \*dev, constchar \*id);

// 释放clk

void clk\_put(struct clk \*clk);

//打开和关闭clk

int clk\_enable(struct clk \*clk);

void clk\_disable(struct clk \*clk);

// 设置clk的频率，获得clk的频率

int clk\_set\_rate(struct clk \*clk, unsignedlong rate);

long clk\_round\_rate(struct clk \*clk, unsignedlong rate); //无法设置精确clk

unsigned long clk\_get\_rate(struct clk\*clk);

// 设置clk的parent，获取clk的parent

int clk\_set\_parent(struct clk \*clk, structclk \*parent);

struct clk \*clk\_get\_parent(struct clk\*clk);

clk结构体，在arch\arm\plat-vc088x\include\plat\clock.h实现。

其它函数，在arch\arm\plat-vc088x\ clock.c 中实现，供驱动层模块调用。

**2. clk porting层**

**2.1 clk porting 层包含3个文件**

clk porting 层主要包括3个文件：

arch\arm\plat-vc088x\include\plat\clock.h  // clk结构体的实现

arch\arm\plat-vc088x\clock.c   // 驱动层与porting层之间的api的实现

arch\arm\mach-vc0882\clock-vortex.c // 建立clk 节点链表

**2.2 clk的描述**

在arch\arm\mach-vc0882\clock-vortex.c文件中定义了下面这个数组：

      static struct clk\* sys\_clks[];

这个数组使用struct clk结构体描述了所有的时钟节点。

**clk节点的[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)，采用面向对象的思想。**

                     下面以snr为例，介绍如何描述clk节点的。

static struct clk snr\_clk = {

       .name  = "snr\_clk", // 驱动层通过clk\_get函数获取clk使用

       .flags = CLK\_FLAG\_DIV |CLK\_FLAG\_GATE | RECALC\_ON\_ENABLE,

       .parent = &XCLK,

       .max\_dividor = 64,

       .private\_data =&snr\_clkdata,

**.enable = v8clk\_clear\_gatebit**, // clk\_enable 最终的实现代码

**.disable = v8clk\_set\_gatebit**, // clk\_ disable 最终的实现代码

       .recalc =vc88x\_clk\_recalc,

**.****set\_rate = v8clk\_setrate**,// clk\_ set\_rate 最终的实现代码

**.init = vc88x\_clk\_init**,

};

static struct clk\_data snr\_clkdata = {

       .clk\_cfg\_reg =V8REG\_CLKRST\_CIF\_MCLK\_CFG,

       .cfg\_data =&snr\_clkcfg,

       .cfg\_mask = 0x3F00,

       .pfnCfg =div\_calculator,

       .pfnRate = div\_parser,

       .clk\_ctrl\_reg =V8REG\_CLKRST\_CIF\_MCLK\_CTRL,

       .gate\_bit =(1<<1),

};

struct clk\_data 这个结构体，用来保存clk节点的寄存器以及相关的bit信息的。这些成员变量在clk\_set\_rate、clk\_enable、clk\_ disable等api中被使用的。比较重要的，有如下成员变量：

clk\_ctrl\_reg

gate\_bit

bypass\_bit

clk\_status\_reg

clk\_sw\_rst\_reg

clk\_cfg\_reg

div\_max

div\_min

div\_shift

**2.3 clk的注册以及初始化过程**

              1.建立clk 节点list

              2.初始化所有clk

在时钟初始化的过程中，通过clk\_register函数，将sys\_clks数组中所有的时钟节点，都注册到一个list中。函数调用过程如下：

->init\_machine()(vortex\_init ())

                  -> vc088x\_register\_baseclocks ()

                     -> clk\_register ()

                                   ->list\_add () // 建立clk 节点list

->clk->init(clk); // 这个时候会初始化clk

       -> clk->set\_rate()

**3. 驱动程序如何使用clk模型的api**

根据clk节点的name，通过clk\_get 获取时钟节点。

clk1 = clk\_get(&dev, " snr\_clk");

clk2 = clk\_get(&dev, " dpi\_pixel\_clk ");

clk3 = clk\_get(&dev, " cvbs\_pixel\_clk ");

clk4= clk\_get(&dev, " vdac\_pixel\_clk ");

……

然后，就可以通过驱动层与porting层之间的api来设置各个时钟节点。

clk\_disable(clk1);

clk\_set\_rate(clk2, 24\*1000000);

clk\_enable(clk3);

**4. cpu切频如何使用clk模型的api**

       cpu切频驱动程序包括以下几个文件：

drivers\cpufreq\

cpufreq.c

cpufreq\_conservative.c     // 按次序切频

cpufreq\_ondemand.c        // 按命令切频

cpufreq\_performance.c     // 最高频率

cpufreq\_powersave.c        // 最低频率

cpufreq\_stats.c

cpufreq\_userspace.c

freq\_table.c

       cpu切频 porting 层包括以下几个文件：

arch\arm\plat-vc088x\cpu.c      // structcpufreq\_driver 结构体



其中，cpufreq\_driver 结构体的成员函数，基于linux clk 模型的驱动层和porting层的api来实现的。

static struct cpufreq\_driver v8\_driver = {

       .flags             = CPUFREQ\_STICKY,

       .verify            = v8\_verify\_speed,

       .target            = v8\_target,    // 切频最终调用的函数

       .get         = v8\_getspeed,

       .init        = v8\_cpu\_init,

       .exit        = v8\_cpu\_exit,

       .name             = "v8cpu",

       .attr        = v8\_cpufreq\_attr,

};

v8\_driver .target->

       v8\_target()

clk\_set\_rate(cpu\_clk, freqs.new \* 1000);

v8\_driver .get->

       v8\_getspeed()

rate = clk\_get\_rate(cpu\_clk) / 1000;