全志科技
Allwinner Technology

V40 项目

clock 接口使用说明书 V1.0

文档履历

版本号	日期	制/修订人	内容描述
v1.0	2016-6-11		正式版本

目　　录

1. 概述

1.1.编写目的

本文档对 V40 平台的时钟管理接口使用进行详细的阐述，让用户明确掌握时钟操作的编程方法。

1.2.适用范围

本文档仅适用于 V40 sdk 内核。

1.3.相关人员

本文档适用于所有需要在 V40 sdk 上开发设备驱动的人员。

2. 模块介绍

时钟管理模块是 linux 系统为统一管理各硬件的时钟而实现管理框架，负责所有模块的时钟调节和电源管理。

2.1. 模块功能介绍

时钟管理模块主要负责处理各硬件模块的工作频率调节及电源切换管理。一个硬件模块要正常工作，必须先配置好硬件的工作频率、打开电源开关、总线访问开关等操作，时钟管理模块为设备驱动提供统一的操作接口，使驱动不用关心时钟硬件实现的具体细节。

2.2. 相关术语介绍

晶振

晶体振荡器的简称，晶振有固定的振荡频率，如 32K/24Mhz 等，是芯片所有时钟的源头。

PLL

锁相环，利用输入信号和反馈信号的差异提升频率输出。

时钟

驱动数字电路运转是的时钟信号。芯片内部的各硬件模块都需要时序控制，因此理解时钟信号对于底层编程非常重要。

2.3. 模块配置介绍

主要 device tree 配置方式。

```
pll_ve: pll_ve_clk {
    #clock-cells = <0>;
    compatible = "allwinner,sunxi-pll-clock";
    clock-output-names = "pll_ve";
    clock-frequency = <864000000>;
};
```

对于没有配置的，系统设置频率为默认值。

2.4.源码结构介绍

CCU 的源码结构如下图所示：

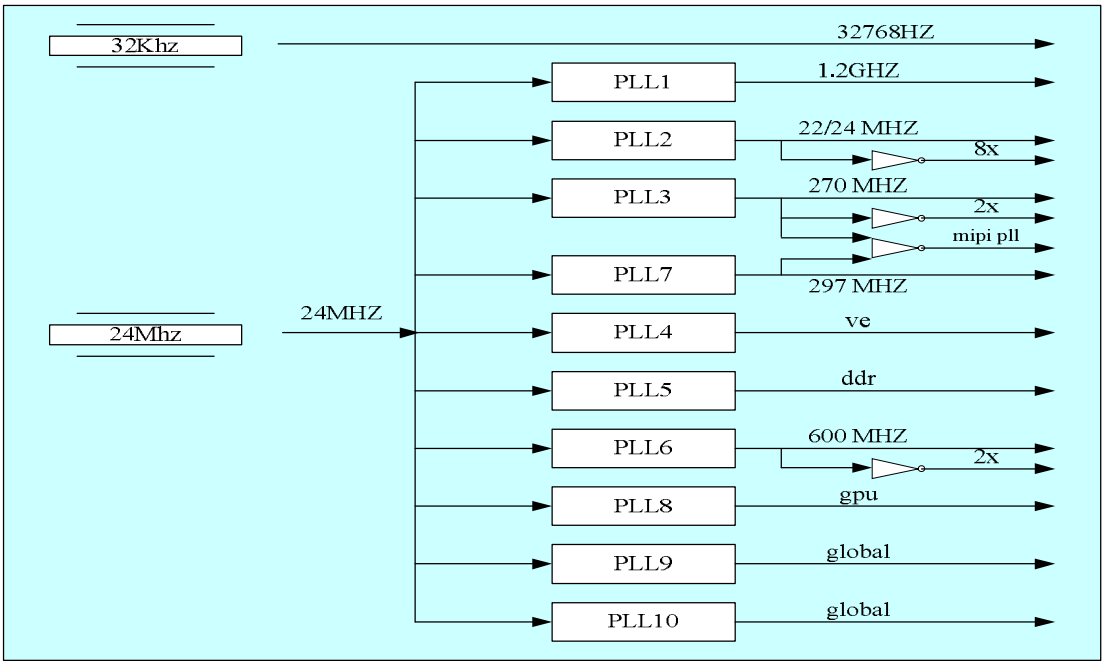
```
|---drivers
||---clk
|||---sunxi
||||---clk-sun8iw11.h
||||---clk-periph.h
||||---clk-sunxi.h
```

```
||||---clk-factors.h
||||---clk-factors.c
||||---clk-periph.c
||||---clk-sun8iw11.c
||||---clk-default.c
||||---clk-sun8iw11_tbl.c
||||---clk-debugfs.c
|---include
||---linux
|||---clk.h
|||---clk
```

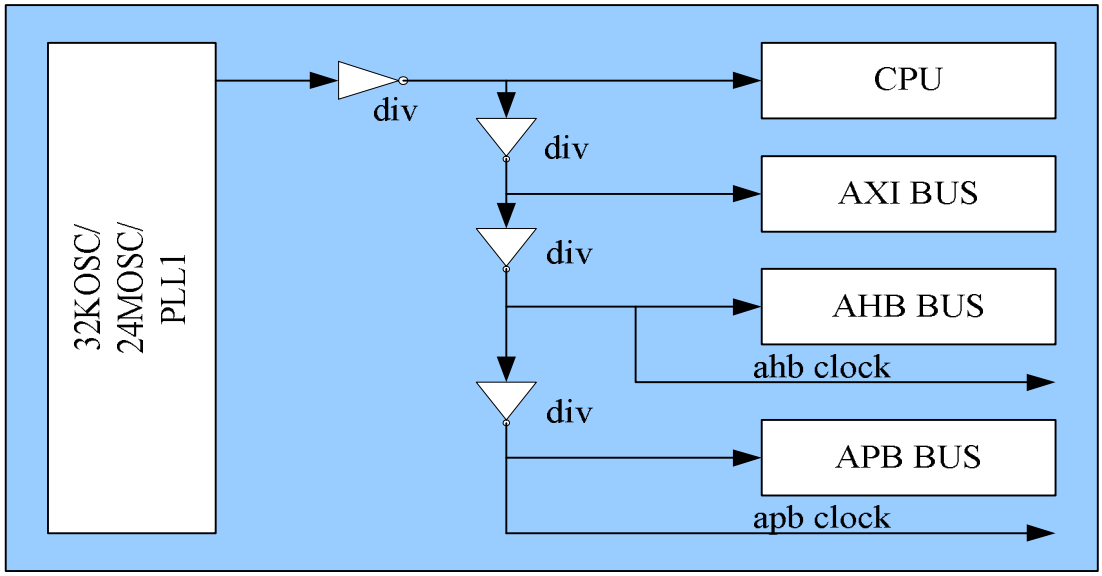
clk.h 时钟子系统提供的 API 接口定义；
clk-factors.c: 针对 PLLx 等系统时钟(以 HOSC 为 source)的公用代码
clk-periph.c:针对各模块时钟的公用代码
clk-sun8iw11.c: 针对 V40 的实现
clk-default.c: 第二级初始化操作
clk-debugfs.c: 针对 debugfs 的调试接口
clk-sun8iw11_tbl.c：针对 V40 clk table

2.5.系统时钟结构

系统时钟主要是指一些源时钟，为其它硬件模块提供时钟源输入。系统时钟一般为多个硬件模块共享，不允许随意调节。



系统上一般只有两个时源头：低频晶振（LOSC）32Khz 和高频晶振（HOSC）24Mhz，系统在 HOSC 的基础上，增加一些锁相环电路，实现更高的时钟频率输出。为了便于控制一些模块的时钟频率，系统对时钟源进行了分组，实现较多的锁相环电路，以实现分路独立调节。
由于 CPU、总线的时钟比较特殊，其工作时钟也经常会输出作为某些其它模块的时钟源，因此，我们也将此类时钟归结为系统时钟。其结构图如下：



V40 平台上，定义的系统时钟源清单 sun8iw11p1-clk.dtsi，其中 pll 的定义如下：

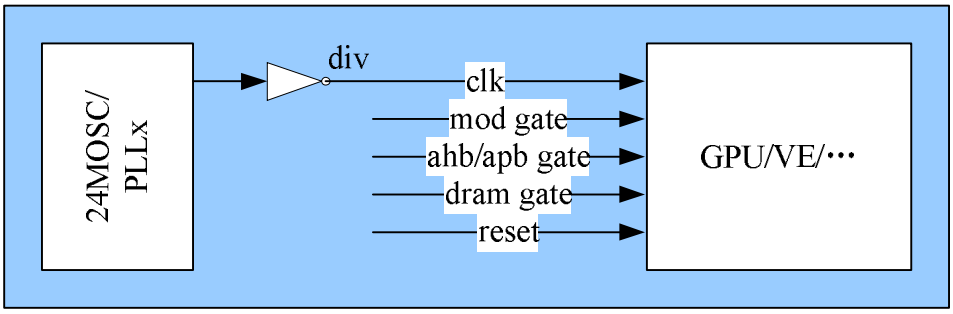
```
        clk_losc: losc {
                #clock-cells = <0>;
                compatible = "allwinner,fixed-clock";
                clock-frequency = <32768>;
                clock-output-names = "losc";
        };

        clk_iosc: iosc {
                #clock-cells = <0>;
                compatible = "allwinner,fixed-clock";
                clock-frequency = <16000000>;
                clock-output-names = "iosc";
        };

        clk_hosc: hosc {
                #clock-cells = <0>;
                compatible = "allwinner,fixed-clock";
                clock-frequency = <24000000>;
                clock-output-names = "hosc";
        };
/* register allwinner,sunxi-pll-clock */
        clk_pll_cpu: pll_cpu {
                #clock-cells = <0>;
                compatible = "allwinner,sunxi-pll-clock";
                lock-mode = "none";
                clock-output-names = "pll_cpu";
        };
        clk_pll_audio: pll_audio {
                #clock-cells = <0>;
                compatible = "allwinner,sunxi-pll-clock";
```

```
                lock-mode = "none";
                assigned-clock-rates = <24576000>;
                clock-output-names = "pll_audio";
        };
        clk_pll_video: pll_video {
                #clock-cells = <0>;
                compatible = "allwinner,sunxi-pll-clock";
                lock-mode = "none";
                assigned-clock-rates = <297000000>;
                clock-output-names = "pll_video";
        };
        clk_pll_ve: pll_ve {
                #clock-cells = <0>;
                compatible = "allwinner,sunxi-pll-clock";
                device_type = "clk_pll_ve";
                lock-mode = "none";
                assigned-clock-rates = <420000000>;
                clock-output-names = "pll_ve";
        };
```

各 PLL 的分工如下：
pll_cpu 只作为 CPU 的时钟源，不作他用；
pll_audio 只作为音频模块（如 codec、iis、spdif 等）的时钟源，不作他用；
pll_video0、pll_video1 一般作为显示相关模块（如 de、csi、hdmi 等）的时钟源；
pll_ve 一般只作为视频解码模块（ve）的时钟源；
pll_ddr0、pll_ddr1 一般只作为 DDR 的时钟源；
hosc 用作一些外设接口模块（如 nand、sdmmc、usb 等）的时钟源；
pll_gpu 一般只作为 GPU 模块的时钟源；
pll_periph0、pll_periph1 是两个通用时钟源，可以为多个模块共享；

2.6.模块时钟结构

模块时钟主要是针对一些具体模块（如：gpu、de），在时钟频率配置、电源控制、访问控制等方面进行管理。一个典型的模块如下图所示，包含 module gateing、ahb gating、dram gating，以及 reset 控制。要想一个模块能够正常工作，必须在这几个方面作好相关的配置。



硬件设计时，为每个硬件模块定义好了可选的时钟源（有些默认使用总线的工作时钟作时钟源），时钟源的定义如上节所述，模块只能在相关可能的时钟源间作选择。

模块的电源管理体现在两个方面：模块的时钟使能和模块控制器复位，相关驱动需要通过以下所列的时钟进行控制。V40 的模块时钟 sun8iw11p1-clk.dtsi 清单如下：

```
cpu: cpu_clk {
        #clock-cells = <0>;
        compatible = "allwinner,sunxi-periph-clock";
        clock-output-names = "cpu";
};
其他类似的定义如下:
"axi"
"pll_periphahb0"
"ahb1"
"apb1"
"apb2"
"ahb2"
"ths"
"nand"
"sdmmc0"
"sdmmc1"
"sdmmc2"
"ts"
"ce"
"spi0"
"spi1"
"i2s0"
"i2s1"
"i2s2"
"spdif"
"usbphy0"
"usbphy1"
"usbhsic"
"usbhsic12m"
"usbohci1"
"usbohci0"
"de"
"tcon0"
"tcon1"
"deinterlace"
"csi_s"
"csi_m"
"csi_misc"
"ve"
"adda"
"addax4"
"avs"
"hdmi"
```

```
"hdmi_slow"
"mbus"
"mipicsi"
"gpu"
"usbehci0"
"usbehci1"
"usbotg"
"gmac"
"sdram"
"dma"
"spinlock"
"msgbox"
"lvds"
"uart0"
"uart1"
"uart2"
"uart3"
"uart4"
"scr"
"twi0"
"twi1"
"twi2"
```

3. 接口描述

Linux 系统为时钟管理定义了标准的 API 接口，详见内核接口头文件《include/linux/clk.h》。

3.1. 时钟 API 接口定义

使用系统的时钟操作接口，必须引用 Linux 系统提供的时钟接口头文件，引用方式为“#include <linux/clk.h>”
Linux 系统为时钟管理定义了一套标准和 API 接口，V40 的时钟 API 遵循该 API 规范。

3.2.内部接口

目前定义了
2 个回调接口，主要用来配置 device tree 中对应的 clk.
2 个注册接口，用来将 clk 注册到 CCF 框架。
3 个初始化接口，用于按照 device tree 方式初始化所有的 clocks.

3.2.1. get_sunxi_register_periph_config

- **PROTOTYPE**

```
int get_sunxi_register_periph_config(const char* clk_name , struct sunxi_register_periph_config* config)
```

- **ARGUMENTS**

clk_name device tree 设置的 clock-output-names 名字；
config 对应 clk-name 的配置信息

- **RETURNS**

如果获取 periph-clk 参数成功，返回 0，否则返回-1。

3.2.2. get_sunxi_register_factors_config

- **PROTOTYPE**

```
int get_sunxi_register_factors_config(const char* clk_name , struct sunxi_register_factors_config* config)
```

- **ARGUMENTS**

clk_name device tree 设置的 clock-output-names 名字；
config 对应 clk-name 的配置信息

- **RETURNS**

如果获取 pll-clk 参数成功，返回 0，否则返回-1。

3.2.3. sunxi_clk_register_factors

- **PROTOTYPE**

```
struct clk *sunxi_clk_register_factors(struct device *dev,void __iomem *base,
        spinlock_t *lock,struct factor_init_data* init_data)
```

- **ARGUMENTS**

dev 设备；
base 寄存器基址
Lock 锁
init_data 初始化数据

- **RETURNS**

如果 register 成功，返回 clk；否则，返回 NULL。

3.2.4. sunxi_clk_register_periph

- **PROTOTYPE**

```
struct clk *sunxi_clk_register_periph(const char *name,
                const char **parent_names, int num_parents,unsigned long flags,
                void __iomem    *base, struct sunxi_clk_periph *periph)
```

- **ARGUMENTS**

dev 设备；
base 寄存器基址
Lock 锁
periph 初始化数据

- **RETURNS**

如果 register 成功，返回 clk；否则，返回 NULL。

3.2.5. of_sunxi_clocks_init

- **PROTOTYPE**

```
void of_sunxi_clocks_init(struct device_node *node)
```

- **ARGUMENTS**

node device tree 中的 /clocks 节点

- **RETURNS**

无。

3.2.6. of_pll_clk_setup

- **PROTOTYPE**

```
void of_pll_clk_setup(struct device_node *node)
```

- **ARGUMENTS**

node device tree 中的 /clocks/pll_cpu、 /clocks/pll_audio 等 pll-clk 相关节点

- **RETURNS**

无。

3.2.7. of_periph_clk_setup

- **PROTOTYPE**

```
void of_periph_clk_setup(struct device_node *node)
```

- **ARGUMENTS**

node　　device tree 中的 /clocks/cpu、 /clocks/axi 等 periph-clk 相关节点

- **RETURNS**

无。

3.3.外部接口

Linux 系统的时钟操作 API 接口定义如下：

- **PROTOTYPE**

```
struct clk *of_clk_get(struct device_node *np, int index);
```

- **ARGUMENTS**

np　　device tree 中引用 clocks 的节点；

index　　device tree 中定义的#clock-cells 索引号；

- **RETURNS**

如果申请时钟成功，返回时钟句柄，否则返回 NULL。

3.3.1. of_clk_get_by_name

- **PROTOTYPE**

```
struct clk *of_clk_get_by_name(struct device_node *np, const char *name);
```

- **ARGUMENTS**

np　　device tree 中引用 clocks 的节点；

name　　device tree 中定义的 clock-output-names 的时钟名字；

- **RETURNS**

如果申请时钟成功，返回时钟句柄，否则返回 NULL。

3.3.2. clk_get

- **PROTOTYPE**

```
struct clk *clk_get(struct device *dev, const char *id);
```

- **ARGUMENTS**

dev　　申请时钟的设备句柄；

id　　要申请的时钟名；

- **RETURNS**

如果申请时钟成功，返回时钟句柄，否则返回 NULL。

3.3.3. devm_clk_get

- **PROTOTYPE**

```
struct clk *devm_clk_get(struct device *dev, const char *id);
```

- **ARGUMENTS**

dev　　申请时钟的设备句柄；

id　　要申请的时钟名；

- **RETURNS**

如果申请时钟成功，返回时钟句柄，否则返回 NULL。

3.3.4. clk_put

- **PROTOTYPE**

```
void clk_put(struct clk *clk);
```

- **ARGUMENTS**

clk　　待释放的时钟句柄；

- **RETURNS**

无。

3.3.5. clk_set_parent

- **PROTOTYPE**

int clk_set_parent(struct clk *clk, struct clk *parent);

- **ARGUMENTS**

clk　　待操作的时钟句柄；

parent　父时钟的时钟句柄；

- **RETURNS**

如果设置父时钟成功，返回 0；否则，返回-1。

3.3.6. clk_get_parent

- **PROTOTYPE**

```
struct clk * clk_get_parent(struct clk *clk);
```

- **ARGUMENTS**

clk　　待操作的时钟句柄；

- **RETURNS**

如果获取父时钟成功，返回父时钟句柄；否则，返回-1。

3.3.7. clk_prepare

- **PROTOTYPE**

```
int clk_prepare(struct clk *clk);
```

- **ARGUMENTS**

clk　　待操作的时钟句柄；

- **RETURNS**

如果时钟 prepare 成功，返回 0；否则，返回-1。

3.3.8. clk_enable

- **PROTOTYPE**

```
int clk_enable(struct clk *clk);
```

- **ARGUMENTS**

clk　　待操作的时钟句柄；

- **RETURNS**

  如果时钟使能成功，返回 0；否则，返回-1。

3.3.9. clk_prepare_enable

- **PROTOTYPE**

```
int clk_prepare_enable(struct clk *clk);
```

- **ARGUMENTS**

  clk　　待操作的时钟句柄；

- **RETURNS**

  如果时钟使能成功，返回 0；否则，返回-1。

3.3.10. clk_disable

- **PROTOTYPE**

```
void clk_disable(struct clk *clk);
```

- **ARGUMENTS**

  clk　　待操作的时钟句柄；

- **RETURNS**

  无。

3.3.11. clk_unprepare

- **PROTOTYPE**

```
void clk_unprepare(struct clk *clk);
```

- **ARGUMENTS**

  clk　　待操作的时钟句柄；

- **RETURNS**

  无。

3.3.12. clk_disable_unprepare

- **PROTOTYPE**

```
void clk_disable_unprepare(struct clk *clk);
```

- **ARGUMENTS**

  clk　　待操作的时钟句柄；

- **RETURNS**

  无。

3.3.13. clk_get_rate

- **PROTOTYPE**

```
unsigned long clk_get_rate(struct clk *clk);
```

- **ARGUMENTS**

  clk　待操作的时钟句柄；

- **RETURNS**

  指定时钟的当前频率值。

3.3.14. clk_set_rate

- **PROTOTYPE**

```
int clk_set_rate(struct clk *clk, unsigned long rate);
```

- **ARGUMENTS**

  clk　　待操作的时钟句柄；

  rate　　时钟的目标频率值，以 Hz 为单位；

- **RETURNS**

  如果设置时钟频率成功，返回 0；否则，返回-1。

4. Demo

4.1. 时钟 API 调用格式要求

(1) 一定要判断 clk_get 返回句柄的有效性. 比如:
规范写法:

```
struct clk *pll = clk_get(NULL, "sys_pll3");
if(!pll || IS_ERR(pll)){
        /* 获取时钟句柄失败 */
        printk(“try to get pll3 failed!\n”);
}
...
```

不规范写法:

```
struct clk *pll = clk_get(NULL, "sys_pll3");
...(访问 pll 句柄)
```

(2) 有返回值的 clk api 一定要判断返回值, 返回失败时建议加打印, 比如:
规范写法:

```
if(clk_enable(pll)) {
        /* 使能 PLL3 输出失败 */
        printk("try to enable pll3 output failed!\n");
}
```

不规范写法:

```
clk_enable(pll);
```

(3) clk_disable 或 clk_put 之前, 先检查句柄的有效性; clk_put 之后将句柄清空. 比如:

```
if(NULL == sdc0_clk || IS_ERR(sdc0_clk)) {
        printk("sdc0_clk handle is invalid, just return!\n");
        return;
} else {
        clk_disable(sdc0_clk);
        clk_put(sdc0_clk);
        sdc0_clk = NULL;
}
```

4.2. 设置 PLL3 频率

```
struct clk *pll = clk_get(NULL, "sys_pll3");
if(!pll || IS_ERR(pll)){
        /* 获取时钟句柄失败 */
        printk(“try to get pll3 failed!\n”);
}
/* 使能时钟 */
```

```
if(clk_prepare_enable(pll)) {
	/* 使能 PLL3 输出失败 */
	printk("try to enable pll3 output failed!\n");
}
/* 设置 PLL3 的频率为 270Mhz */
if(clk_set_rate(pll, 270000000)) {
	/* 设置 PLL3 频率失败 */
	printk("try to set rate to 270000000 failed!\n");
}
```

Declaration