



RTOS CCU

开发指南

版本号: 1.1
发布日期: 2021.04.27

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.10.23	AWA1637	1. 初版
1.1	2021.04.27	AWA1637	1. 增加 F133 使用说明



目 录

1 前言	1
1.1 文档简介	1
1.2 目标读者	1
1.3 适用范围	1
2 模块介绍	2
2.1 模块功能介绍	2
2.2 相关术语介绍	2
2.3 模块配置介绍	2
2.3.1 设备树配置	2
2.3.2 platform 配置	2
2.4 模块源码结构	3
2.5 模块数据结构	5
2.5.1 时钟数据结构	5
2.5.2 时钟类型数据结构	5
2.5.3 时钟序号	6
2.5.4 rst 结构体	7
2.5.5 rst 类型结构体	7
2.5.6 rst 序号	7
3 模块接口说明	9
3.1 接口列表	9
4 模块使用范例	10
4.1 FAQ	11

1 前言

1.1 文档简介

介绍 RTOS 中 CCU 驱动的接口及使用方法，为 CCU 使用者提供参考。

1.2 目标读者

CCU 驱动层/应用层开发/使用/维护人员。

1.3 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
V459	Melis	hal_clk.c
R328	FreeRTOS	hal_clk.c
F133	Melis	hal_clk.c

2 模块介绍

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

表 2-1: CCU 模块相关术语介绍

术语	解释说明
SUNXI	Allwinner 一系列 SOC 硬件平台
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	驱动数字电路运转时的时钟信号，芯片内部的各硬件模块都需要时序控制，因此理解时钟信号很重要

2.3 模块配置介绍

2.3.1 设备树配置

可以到 Kernel Setup-> Driver Setup->Soc Hal Drivers->CCMU Driver 下面把 Enable ccmu driver 选项打开，此时跳出两个选项。如果是 V459 及 V459 以前的旧平台，选中 enable sunxi ccmu driver。如果是 F133 以后的平台选中 enable sunxi-ng ccmu driver。如果需要测试用 ccu-ng 的功能，可以把 enable sunxi-ng ccmu hal APIs test command 选项打开。

2.3.2 platform 配置

在不同的 Sunxi 硬件平台中，CCU 的设计有差别，但平台配置文件的信息基本类似，如下：

```
rtos-hal/source/ccmu/sunxi-ng
|   ├── ccu-sun8iw20.c
|   ├── ccu-sun8iw20.h
|   ├── ccu-sun8iw20-r.c
|   ├── ccu-sun8iw20-r.h
|   ├── rst-sun8iw20.h
|   └── rst-sun8iw20-r.h
```

其中 ccu-xxxx.c 主要包含每个平台具体的 clk 配置。ccu-xxx.h 主要包含每个时钟的 clk 序号，在调用时钟相关的配置时用到。rst-xxx.h 主要包含每个时钟的 rst 配置，在调用时钟相关的配置时用到。

2.4 模块源码结构

RTC 模块源码结构如下所示：

```
└── common_ccmu.h
└── hal_clk.c
└── hal_clk.o
└── hal_reset.c
└── hal_reset.o
└── Kconfig
└── Makefile
└── modules.order
└── platform_ccmu.h
└── platform_rst.h
└── sunxi
    ├── built-in.o
    ├── clk.c
    ├── clk_factors.c
    ├── clk_factors.h
    ├── clk_factors.o
    ├── clk.h
    ├── clk.o
    ├── clk_periph.c
    ├── clk_periph.h
    ├── clk_periph.o
    ├── Makefile
    ├── modules.order
    ├── platform_clk.h
    └── sun8iw18p1
        ├── clk_sun8iw18.c
        └── clk_sun8iw18.h
    └── sun8iw19p1
        ├── built-in.o
        ├── clk_sun8iw19.c
        ├── clk_sun8iw19.h
        ├── clk_sun8iw19.o
        └── Makefile
└── sunxi-ng
    ├── ccu.c
    ├── ccu_common.c
    ├── ccu_common.h
    ├── ccu_common.o
    └── ccu_div.c
```

```
├── ccu_div.h
├── ccu_div.o
├── ccu_frac.c
├── ccu_frac.h
├── ccu_frac.o
├── ccu_gate.c
├── ccu_gate.h
├── ccu_gate.o
├── ccu.h
├── ccu_mp.c
├── ccu_mp.h
├── ccu_mp.o
├── ccu_mult.c
├── ccu_mult.h
├── ccu_mult.o
├── ccu_mux.c
├── ccu_mux.h
├── ccu_mux.o
├── ccu_nk.c
├── ccu_nk.h
├── ccu_nkm.c
├── ccu_nkm.h
├── ccu_nkm.o
├── ccu_nkmp.c
├── ccu_nkmp.h
├── ccu_nkmp.o
├── ccu_nk.o
├── ccu_nm.c
├── ccu_nm.h
├── ccu_nm.o
├── ccu.o
├── ccu_phase.c
├── ccu_phase.h
├── ccu_reset.c
├── ccu_reset.h
├── ccu_reset.o
├── ccu_sdm.c
├── ccu_sdm.h
├── ccu_sdm.o
├── ccu-sun8iw20.c
├── ccu-sun8iw20.h
├── ccu-sun8iw20.o
├── ccu-sun8iw20-r.c
├── ccu-sun8iw20-r.h
├── ccu-sun8iw20-r.o
├── ccu-sun8iw20-rtc.c
├── ccu-sun8iw20-rtc.h
├── ccu-sun8iw20-rtc.o
├── clk.c
├── clk-divider.c
├── clk-divider.o
├── clk-fixed-factor.c
├── clk-fixed-factor.o
├── clk-fixed-rate.c
├── clk-fixed-rate.h
├── clk-fixed-rate.o
├── clk.h
├── clk.o
├── Makefile
└── rst-sun8iw20.h
```

└─ rst-sun8iw20-r.h

2.5 模块数据结构

ccu-ng 模块提供给其他模块使用的接口有使能/失能时钟，使能/失能 rst，设置父时钟，设置频率等等。下面介绍需要使用到的一些数据结构。

2.5.1 时钟数据结构

该数据结构用来保存时钟的一些信息，也是配置时钟相关配置必备的数据结构，该结构体在每个平台的定义有所不同。

在 V459 之前的平台，该结构体就是一个数，表明时钟的标号，用来传给时钟配置的各个接口，定义可以在 platform_ccmu.h 查看。

```
typedef u32 hal_clk_id_t;
typedef hal_clk_id_t hal_clk_t;
```

在 f133 之后的平台，其是一个结构体，用来存放时钟的信息，定义可以在 platform_ccmu.h 查看

```
struct clk
{
    struct clk_core *core;
    const char *name;
};

typedef struct clk *hal_clk_t;
```

2.5.2 时钟类型数据结构

该数据结构用来保存 clk 的类型，对于 V459 之前的平台和 F133 之后的平台，该数据结构的定义有所不同，下面给出相关的定义。

在 V459 之前的平台，该数据结构说明的是时钟是 fixed-clock 还是 periph-clock 之类的时钟，其定义可以在 sunxi/platform_clk.h 中找到

```
typedef enum
{
    HAL_CLK_ROOT      = -1,
    HAL_CLK_FIXED_SRC,
    HAL_CLK_FIXED_FACTOR,
    HAL_CLK_FACTOR,
    HAL_CLK_PERIPH,
} hal_clk_type_t;
```

在 F133 之后的平台，该数据结构说明的是 cpux 还是 cpus 域的时钟，其定义可以在 platform_ccmu.h 找到：

```
typedef enum
{
    HAL_SUNXI_FIXED_CCU = 0,
    HAL_SUNXI_RTC_CCU,
    HAL_SUNXI_CCU,
    HAL_SUNXI_R_CCU,
    HAL_SUNXI_CCU_NUMBER,
} hal_clk_type_t;
```

2.5.3 时钟序号

可以对 hal_clk_id_t 数据结构赋值，对于 V459 之前的平台和 F133 之后的平台，这些定义有所不同，下面给出相关的定义。

在 V459 之前的平台，数据结构定义可以在 sunxi/platform_clk.h 中找到。下面给出部分定义。

```
enum
{
    HAL_CLK_UNINITIALIZED      = -1,
    HAL_CLK_SRC_ROOT,
    /*
     *  FIXED SOURCE CLOCK 0~255
     */
    HAL_CLK_SRC_HOSC24M        = HAL_CLK_FIXED_SRC_OFFSET,
    HAL_CLK_SRC_HOSC24MD2,
    HAL_CLK_SRC_IOSC16M,
    HAL_CLK_SRC_OSC48M,
    HAL_CLK_SRC_OSC48MD4,
    HAL_CLK_SRC_LOSC,
    HAL_CLK_SRC_RC16M,
    HAL_CLK_PLL_PERI0DIV25M,
    /*
    */
};
```

在 F133 之后的平台，结构体定义可以在 sunxi-ng/ccu-xxx.h 找到，如 f133 的可以在 sunxi-ng/ccu-sun8iw20.h 找到，下面给出部分定义。

```
#define CLK_OSC12M      0
#define CLK_PLL_CPUX     1
#define CLK_PLL_DDR0      2
#define CLK_PLL_PERIPH0_PARENT 3
#define CLK_PLL_PERIPH0_2X 4
#define CLK_PLL_PERIPH0    5
#define CLK_PLL_PERIPH0_800M 6
#define CLK_PLL_PERIPH0_DIV3 7
#define CLK_PLL_VIDE00     8
#define CLK_PLL_VIDE00_2X  9
#define CLK_PLL_VIDE00_4X 10
#define CLK_PLL_VIDE01     11
#define CLK_PLL_VIDE01_2X 12
```

```
#define CLK_PLL_VIDE01_4X 13
#define CLK_PLL_VE 14
#define CLK_PLL_AUDIO0 15
#define CLK_PLL_AUDIO0_2X 16
#define CLK_PLL_AUDIO0_4X 17
#define CLK_PLL_AUDIO1 18
```

2.5.4 rst 结构体

该结构体表明了一个 rst 的信息，其与 rst 的相关配置息息相关，具体定义如下所示：

```
struct reset_control {
    struct reset_control_dev *rcdev;
    u32 enable_count;
    hal_reset_id_t id;
};
```

2.5.5 rst 类型结构体

该结构体用来保存 rst 的类型，对于 V459 之前的平台和 F133 之后的平台，该结构体的定义有所不同，下面给出相关的定义。

```
typedef enum {
    HAL_SUNXI_RESET = 0,
    HAL_SUNXI_R_RESET,
    HAL_SUNXI_RESET_NUMBER,
} hal_reset_type_t;
```

2.5.6 rst 序号

该数据结构用来保存 rst 的序号，对于 rst 的申请作用很大，其是一个数。具体的配置可以到 sunxi-ng/rst-xxx.h 或者 sunxi-ng/rst-xxx-r.h 查看，注意 V459 之前的平台无定义，所以可以随便赋值。下面给出部分参考

```
#define RST_MBUS 0
#define RST_BUS_DE0 1
#define RST_BUS_DI 2
#define RST_BUS_G2D 3
#define RST_BUS_CE 4
#define RST_BUS_VE 5
#define RST_BUS_DMA 6
#define RST_BUS_MSGBOX0 7
#define RST_BUS_MSGBOX1 8
#define RST_BUS_MSGBOX2 9
#define RST_BUS_SPINLOCK 10
#define RST_BUS_HSTIMER 11
#define RST_BUS_DBG 12
```

```
#define RST_BUS_PWM      13
#define RST_BUS_IOMMU      14
#define RST_BUS_DRAM       15
#define RST_BUS_MMC0       16
#define RST_BUS_MMC1       17
```



3 模块接口说明

3.1 接口列表

CCU 提供的主要接口列表如下，使用这些接口需要引入 hal_clk.h 头文件：

```
hal_clk_t hal_clock_get(hal_clk_type_t type, hal_clk_id_t id); //获取时钟
hal_clk_status_t hal_clock_put(hal_clk_type_t type, hal_clk_id_t id); //释放时钟
hal_clk_t hal_clk_get_parent(hal_clk_t clk); //获取父时钟
hal_clk_status_t hal_clk_set_parent(hal_clk_t clk, hal_clk_t parent); //配置父时钟
u32 hal_clk_get_rate(hal_clk_t clk); //获取时钟频率
hal_clk_status_t hal_clk_set_rate(hal_clk_t clk, u32 rate); //配置时钟频率
hal_clk_status_t hal_clock_disable(hal_clk_t clk); //失能时钟
hal_clk_status_t hal_clock_enable(hal_clk_t clk); //使能时钟
hal_clk_status_t hal_clock_is_enabled(hal_clk_t clk); //时钟是否使能
```

注意：以上接口如有返回 hal_clk_t 参数，后面需要调用 hal_clock_put 释放

rst 提供的主要接口列表如下，使用这些接口需要引入 hal_rst.h 头文件。这几个接口在 V459 (包括 V459/R328) 之前的平台无操作

```
struct reset_control *hal_reset_control_get(hal_reset_type_t type, hal_reset_id_t id); //申请reset
int hal_reset_control_put(struct reset_control *reset); //释放reset
int hal_reset_control_deassert(struct reset_control *reset); //打开reset
int hal_reset_control_assert(struct reset_control *reset); //失能reset
int hal_reset_control_reset(struct reset_control *reset); //重置reset，先使能reset，再使能reset
int hal_reset_control_status(struct reset_control *reset); //reset的状态
```

注意：以上接口如有调用 hal_reset_control_get，后面需要调用 hal_reset_control_put 释放

4 模块使用范例

模块的使用可以参考 hal/test/ccmu/test_ng_ccmu.c 文件。下面给出详解：

```
int cmd_test_ng_ccmu(int argc, char **argv)
{
    int i, j;

    hal_clk_type_t clk_type;
    hal_clk_id_t clk_id;
    hal_clk_status_t clk_status;
    hal_clk_t clk, p_clk;
    u32 old_rate, new_rate, p_rate;

    hal_reset_type_t reset_type;
    hal_reset_id_t reset_id;
    struct reset_control *reset;
    int reset_status;

    for (i = HAL_SUNXI_FIXED_CCU; i < HAL_SUNXI_CCU_NUMBER; i++)
    {
        clk_type = i;
        for (j = 0; j < clk_number[i]; j++)
        {
            clk_id = j;
            printf("get clock, type:%d, id:%d\n", clk_type, clk_id);
            clk = hal_clock_get(clk_type, clk_id);

            clk_status = hal_clock_is_enabled(clk);
            printf("clock %s status:%s\n", clk->name, clk_status? "disabled" :
enabled");
            printf("enable clock %s\n", clk->name);
            hal_clock_enable(clk);
            clk_status = hal_clock_is_enabled(clk);
            printf("clock %s status:%s\n", clk->name, clk_status? "disabled" :
enabled");

            if (is_strict_clk(clk))
                continue;

            p_clk = hal_clk_get_parent(clk);
            if (p_clk)
                printf("clock %s's parent is: %s\n", clk->name, p_clk->name);
            else
                printf("clock %s is root clk\n", clk->name);

            old_rate = hal_clk_get_rate(clk);
            printf("clock %s rate: %d\n", clk->name, old_rate);
            old_rate /= 2;
            printf("clock %s set rate: %d\n", clk->name, old_rate);
            hal_clk_set_rate(clk, old_rate);
            new_rate = hal_clk_get_rate(clk);
            printf("clock %s get rate: %d\n", clk->name, new_rate);
        }
    }
}
```

```
//printf("disable clock %s\n", clk->name);
//hal_clock_disable(clk);
//clk_status = hal_clock_is_enabled(clk);
//printf("clock %s status:%s\n", clk->name, clk_status? "disabled" :
enabled);

    hal_clock_put(p_clk);    #注意，该函数与hal_clock_get配对
    hal_clock_put(clk);
}
}

for (i = HAL_SUNXI_RESET; i < HAL_SUNXI_RESET_NUMBER; i++)
{
    reset_type = i;
    for (j = 0; j < reset_number[i]; j++)
    {
        reset_id = j;

        printf("reset: get reset control, type:%d, id: %d\n", reset_type, reset_id)
        ;
        reset = hal_reset_control_get(reset_type, reset_id);

        #printf("reset: control assert\n");
        #hal_reset_control_assert(reset);

        printf("reset: control deassert\n");
        hal_reset_control_deassert(reset);

        reset_status = hal_reset_control_status(reset);
        printf("reset status: %s", reset_status? "assert" : "deassert");

        printf("reset: put reset control, type:%d, id: %d\n", reset_type, reset_id)
        ;
        hal_reset_control_put(reset);
    }
}
return 0;
}
```

4.1 FAQ

注意，V459 与 F133 最新代码都采用 CCU 接口。区别在于时钟跟 RST 的类型定义跟 ID 定义，这些最好在平台的头文件中定义。

著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

  **全志科技**  (不完全列举) 均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。