Rockchip Developer Guide RTOS Clock

文件标识：RK-KF-YF-055

发布版本：V1.2.2

日期：2021-04-28

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

产品版本

芯片名称	版本
PISCES	RT-THREAD&HAL
RK2108	RT-THREAD&HAL
RV1108	RT-THREAD&HAL
RK1808	RT-THREAD&HAL
RK2206	FreeRTOS V10.0.1&HAL

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	张晴	2019-05-21	第一次临时版本发布
V1.1.0	黄涛	2019-09-19	修订 clk dump 相关实现
V1.2.0	张晴	2019-09-19	修订 接口修改，test 相关实现
V1.2.1	张晴	2019-12-06	增加文件标识和免责声明
V1.2.2	黄莹	2021-04-28	修改格式

目录

1. CLK 配置

1.1 HAL CLK 配置

1.1.1 HAL 层 CLK 头文件

cru 的工具会自动生成头文件，里面包含 GATE_ID、SOFTRST_ID、DIV_ID、MUX_ID、CLK_ID。 GATE_ID： 包含 CON 和 SHIFT，CON = GATE_ID / 16, SHIFT = GATE_ID % 16 SOFTRST_ID: 包含 CON 和 SHIFT，CON = SOFTRST_ID / 16, SHIFT = SOFTRST_ID % 16 DIV_ID: 包含 CON、SHIFT、WIDTH MUX_ID： 包含 ON、SHIFT、WIDTH CLK_ID： 包含 DIV 和 MUX 的信息

e.g:

```
#define ACLK_VPU_CLK_PLL_SEL 0x0206000a
Con = 10;Shift = 6;Width = 2;
#define ACLK_VPU_CLK_DIV 0x0500000a
Con = 10;Shift = 0;Width = 5;
```

1.1.2 常用 API

```
uint32_t HAL_CRU_GetPllFreq(struct PLL_SETUP *pSetup);
HAL_Status HAL_CRU_SetPllFreq(struct PLL_SETUP *pSetup, uint32_t rate);
HAL_Check HAL_CRU_ClkIsEnabled(uint32_t clk);
HAL_Status HAL_CRU_ClkEnable(uint32_t clk);
HAL_Status HAL_CRU_ClkDisable(uint32_t clk);
HAL_Check HAL_CRU_ClkIsReset(uint32_t clk);
HAL_Status HAL_CRU_ClkResetAssert(uint32_t clk);
HAL_Status HAL_CRU_ClkResetDeassert(uint32_t clk);
HAL_Status HAL_CRU_ClkSetDiv(uint32_t divName, uint32_t divValue);
uint32_t HAL_CRU_ClkGetDiv(uint32_t divName);
HAL_Status HAL_CRU_ClkSetMux(uint32_t muxName, uint32_t muxValue);
uint32_t HAL_CRU_ClkGetMux(uint32_t muxName);
HAL_Status HAL_CRU_FracdivGetConfig(uint32_t rateOut, uint32_t rate,
                                    uint32_t *numerator,
                                    uint32_t *denominator);
uint32_t HAL_CRU_ClkGetFreq(eCLOCK_Name clockName);
HAL_Status HAL_CRU_ClkSetFreq(eCLOCK_Name clockName, uint32_t rate);
HAL_Status HAL_CRU_ClkNp5BestDiv(eCLOCK_Name clockName, uint32_t rate, uint32_t pRate,
uint32_t *bestdiv);
```

1.1.3 CLK 开关

```
HAL_Check HAL_CRU_ClkIsEnabled(uint32_t clk);
HAL_Status HAL_CRU_ClkEnable(uint32_t clk);
HAL_Status HAL_CRU_ClkDisable(uint32_t clk);
```

参数是 GATE_ID(在 soc.h 中，详细解释见本文 1.1.1)。

备注：

（1）HAL 中没有 CLK 的完整架构，没有时钟树的概念，每个 CLK 都是单独的，没有父子关系。

（2）没有引用计数的概念，写开就会开，写关就会关，对于很多模块共用的 CLK，关闭需谨慎。

1.1.4 CLK 频率设置

```
uint32_t HAL_CRU_ClkGetFreq(eCLOCK_Name clockName);
HAL_Status HAL_CRU_ClkSetFreq(eCLOCK_Name clockName, uint32_t rate);
```

这个是封装好的，参数是 CLK_ID(在 soc.h 中，详细解释见本文 1.1.1)。

如果有其他需求可以通过 DIV 和 MUX 接口，去实现 CLK 的设置。参数是 DIV_ID 和 MUX_ID（在 soc.h 中，详细解释见本文 1.1.1）。

```
HAL_Status HAL_CRU_ClkSetDiv(uint32_t divName, uint32_t divValue);
uint32_t HAL_CRU_ClkGetDiv(uint32_t divName);
HAL_Status HAL_CRU_ClkSetMux(uint32_t muxName, uint32_t muxValue);
uint32_t HAL_CRU_ClkGetMux(uint32_t muxName);
```

1.1.5 CLK SOFTRESET

```
HAL_Check HAL_CRU_ClkIsReset(uint32_t clk);
HAL_Status HAL_CRU_ClkResetAssert(uint32_t clk);
HAL_Status HAL_CRU_ClkResetDeassert(uint32_t clk);
```

参数是 SFRST_ID(在 soc.h 中，详细解释见本文 1.1.1)。

1.2 RT-THREAD CLK 配置

1.2.1 RT-THREAD CLK 接口

```
rt_err_t clk_enable_by_id(int gate_id);
rt_err_t clk_disable_by_id(int gate_id);
struct clk_gate *get_clk_gate_from_id(int gate_id);
void put_clk_gate(struct clk_gate *gate);
rt_err_t clk_enable(struct clk_gate *gate);
rt_err_t clk_disable(struct clk_gate *gate);
int clk_is_enabled(struct clk_gate *gate);
uint32_t clk_get_rate(eCLOCK_Name clk_id);
rt_err_t clk_set_rate(eCLOCK_Name clk_id, uint32_t rate);
```

在 RT-THREAD 中封装接口的原因： 1、增加互斥锁机制，对于公共 CLK，两个模块都在使用的，最好能有锁，这样更安全。 2、增加引用计数，对于公共 CLK，两个模块都在使用的，同时开关的时候有引用计数，这样更安全。

1.2.2 RT-THREAD 开关 CLK

使用示例：

1、对于复用时钟，如hclk_audio是hclk_audio、hclk_vad、hclk_i2s、hclk_pdm等模块复用的，所以开关的时候需要使用下面带有引用计数和锁的接口。

```
struct clk_gate *aclk_vio0 = get_clk_gate_from_id(ACLK_VIO0_GATE);

clk_enable(aclk_vio0);/* clk enable */
clk_disable(aclk_vio0);/* clk disable */

put_clk_gate(aclk_vio0);
```

备注： 因为有引用计数，所以使用的时候注意开关要成对。

2、对于模块专有时钟，如aclk_dsp。开关的时候可以直接通过ID开关，使用如下直接调用HAL的方式：

```
clk_enable_by_id(ACLK_DSP_GATE);/* clk enable */
clk_disable_by_id(ACLK_DSP_GATE);/* clk disable */
```

1.3 RT-THREAD 设置频率

使用示例：

```
clk_set_rate(clk_id, init_rate_hz);
rt_kprintf("%s: rate = %d\n", __func__, clk_get_rate(clk_id));
```

1.3.1 RT-THREAD 设置初始化频率及 CLK DUMP

(1)在 board.c 中初始化时钟使用示例如下：

```
static const struct clk_init clk_inits[] =
{
    INIT_CLK("PLL_GPLL", PLL_GPLL, 1188000000),
    INIT_CLK("PLL_CPLL", PLL_CPLL, 1000000000),
    INIT_CLK("HCLK_M4", HCLK_M4, 400000000),
    INIT_CLK("ACLK_DSP", ACLK_DSP, 300000000),
    INIT_CLK("ACLK_LOGIC", ACLK_LOGIC, 300000000),
    INIT_CLK("HCLK_LOGIC", HCLK_LOGIC, 150000000),
    INIT_CLK("PCLK_LOGIC", PCLK_LOGIC, 150000000),
};
```

```
void rt_hw_board_init()
{
.....
    clk_init(clk_inits, HAL_ARRAY_SIZE(clk_inits), true);
.....
}
```

(2) CLK DUMP

CLK DUMP 只能 DUMP 部分在 clk_inits[]结构中的时钟和所有的寄存器，如果需要增加时钟请按照 clk_inits[]结构添加。

CLK DUMP使用是用FINSH_FUNCTION_EXPORT，直接敲clk_dump()就可以。

1.4 FreeRTOS CLK 配置

1.4.1 FreeRTOS CLK 接口

```
rk_err_t ClkEnableById(int gateId);
rk_err_t ClkDisableById(int gateId);
rk_err_t ClkEnable(CLK_GATE *gate);
rk_err_t ClkDisable(CLK_GATE *gate);
int ClkIsEnabled(CLK_GATE *gate);
CLK_GATE *GetClkGateFromId(int gateId);
void PutClkGate(CLK_GATE *gate);
uint32_t ClkGetRate(eCLOCK_Name clkId);
rk_err_t ClkSetRate(eCLOCK_Name clkId, uint32_t rate);
uint32 GetHclkSysCoreFreq(void);
rk_err_t ClkDevInit(void);
rk_err_t ClkDevDeinit(void);
void ClkInit(const CLK_INIT *clkInits, uint32 clkCount, bool clkDump);
void ClkDisableUnused(const CLK_UNUSED *clksUnused, uint32 clkUnusedCount);
void ClkDump(void);
```

在 FreeRTOS 中封装接口的原因： 1、增加互斥锁机制，对于公共 CLK，两个模块都在使用的，最好能有锁，这样更安全。 2、增加引用计数，对于公共 CLK，两个模块都在使用的，同时开关的时候有引用计数，这样更安全。

1.4.2 FreeRTOS 开关 CLK

使用示例：

1、对于复用时钟，如hclk_audio是hclk_audio、hclk_vad、hclk_i2s、hclk_pdm等模块复用的，所以开关的时候需要使用下面带有引用计数和锁的接口。

```
CLK_GATE *aclk_vio0 = GetClkGateFromId(ACLK_VIO0_GATE);

ClkEnable(aclk_vio0);/* clk enable */
ClkDisable(aclk_vio0);/* clk disable */

PutClkGate(aclk_vio0);
```

备注： 因为有引用计数，所以使用的时候注意开关要成对。

2、对于模块专有时钟，如aclk_dsp。开关的时候可以直接通过ID开关，使用如下直接调用HAL的方式：

```
ClkEnableById(ACLK_DSP_GATE);/* clk enable */
ClkDisableById(ACLK_DSP_GATE);/* clk disable */
```

1.5 FreeRTOS 设置频率

使用示例：

```
ClkSetRate(clkId, rate);
rk_printfA("%s: rate = %d\n", __func__, ClkGetRate(clk_id));
```

1.5.1 FreeRTOS 设置初始化频率及 CLK DUMP

(1)在 board_config.c 中初始化时钟使用示例如下：

```
static const CLK_INIT clkInits[] =
{
    INIT_CLK("PLL_GPLL", PLL_GPLL, 384000000),
    INIT_CLK("PLL_VPLL", PLL_VPLL, 491520000),
    INIT_CLK("CLK_HIFI3", CLK_HIFI3, 164000000),
    INIT_CLK("HCLK_MCU_BUS", HCLK_MCU_BUS, 200000000),
    INIT_CLK("PCLK_MCU_BUS", PCLK_MCU_BUS, 100000000),
    INIT_CLK("SCLK_M4F0", SCLK_M4F0, 200000000),
    INIT_CLK("ACLK_PERI_BUS", ACLK_PERI_BUS, 200000000),
    INIT_CLK("HCLK_PERI_BUS", HCLK_PERI_BUS, 100000000),
    INIT_CLK("HCLK_TOP_BUS", HCLK_TOP_BUS, 100000000),
    INIT_CLK("PCLK_TOP_BUS", PCLK_TOP_BUS, 100000000),
};
```

```
void ClkDevHwInit(void)
{
    ClkDevInit();
    ClkInit(clkInits, HAL_ARRAY_SIZE(clkInits), true);
}

void ClkDevHwDeInit(void)
{
    ClkDevDeinit();
}
```

(2) CLK DUMP

CLK DUMP 只能 DUMP 部分在 clkInits[]结构中的时钟和所有的寄存器，如果需要增加时钟请按照 clkInits[]结构添加。

CLK DUMP有支持test命令。详细见第3章TEST。

2. PD 配置

2.1 HAL PD 配置

2.1.1 HAL 层 PD 头文件

PD 的 ID 需要手动填写一下，如下：

```
#ifndef __ASSEMBLY__
typedef enum PD_Id {
    PD_DSP              = 0x80000000U,
    PD_LOGIC            = 0x80011111U,
    PD_SHRM             = 0x80022222U,
    PD_AUDIO            = 0x80033333U,
} ePD_Id;
#endif
```

按照下面定义，对应填写 PWR_SHIFT, ST_SHIFT, REQ_SHIFT, ACK_SHIFT。

```
#define PD_PWR_SHIFT  0U
#define PD_PWR_MASK   0x0000000FU
#define PD_ST_SHIFT   4U
#define PD_ST_MASK    0x000000F0U
#define PD_REQ_SHIFT  8U
#define PD_REQ_MASK   0x00000F00U
#define PD_IDLE_SHIFT 12U
#define PD_IDLE_MASK  0x0000F000U
```

```
#define PD_ACK_SHIFT  16U
#define PD_ACK_MASK   0x000F0000U

#define PD_GET_PWR_SHIFT(x) (((uint32_t)(x)&PD_PWR_MASK) >> PD_PWR_SHIFT)
#define PD_GET_ST_SHIFT(x)  (((uint32_t)(x)&PD_ST_MASK) >> PD_ST_SHIFT)
#define PD_GET_REQ_SHIFT(x) (((uint32_t)(x)&PD_REQ_MASK) >> PD_REQ_SHIFT)
#if defined(RKMCU_RK1808)
#define PD_GET_IDLE_SHIFT(x) ((((uint32_t)(x)&PD_IDLE_MASK) >> PD_IDLE_SHIFT) + 16)
#else
#define PD_GET_IDLE_SHIFT(x) (((uint32_t)(x)&PD_IDLE_MASK) >> PD_IDLE_SHIFT)
#endif
#define PD_GET_ACK_SHIFT(x) (((uint32_t)(x)&PD_ACK_MASK) >> PD_ACK_SHIFT)
```

2.1.2 常用 API

```
HAL_Status HAL_PD_On(ePD_Id pd);
```

2.1.3 PD 开关

```
HAL_Status HAL_PD_Off(ePD_Id pd);
```

参数是 PD_ID(在 soc.h 中，详细解释见本文 2.1.1)。

备注：

（1）HAL 中没有 PD 的完整架构，没有电源树的概念，每个 PD 都是单独的，没有父子关系。

（2）没有引用计数的概念，写开就会开，写关就会关，对于很多模块共用的 PD，关闭需谨慎。

2.2 RT-THREAD PD 配置

2.2.1 RT-THREAD 接口

```
struct pd *get_pd_from_id(ePD_Id pd_id);
void put_pd(struct pd *power);
rt_err_t pd_on(struct pd *power);
rt_err_t pd_off(struct pd *power);
```

在 RT 中封装接口的原因： 1、增加互斥锁机制，对于公共 PD，两个模块都在使用的，最好能有锁，这样更安全。2、增加引用计数，对于公共 PD，两个模块都在使用的，同时开关的时候有引用计数，这样更安全。

2.2.2 RT-THREAD 开关 PD

使用示例：

1、对于复用PD，如PD_AUDIO是PDM、VAD、I2S等模块复用的，所以开关的时候需要使用下面带有引用计数和锁的接口。

```
struct pd *pd_audio = get_pd_from_id(PD_AUDIO);

pd_on(pd_audio);/* power on */
pd_off(pd_audio);/* power off */

put_pd(pd_audio);
```

备注： 因为有引用计数，所以使用的时候注意开关要成对。

2、对于模块专有PD，如PD_DSP。开关的时候可以直接通过ID开关，使用如下直接调用HAL的方式：

```
HAL_PD_On(PD_DSP);/* power on */
HAL_PD_Off(PD_DSP);/* power off */
```

2.3 FreeRTOS PD配置

2.3.1 FreeRTOS 接口

```
rk_err_t PdPowerOn(PD *power);
rk_err_t PdPowerOff(PD *power);
PD *GetPdFromId(int pdId);
void PutPd(PD *power);
```

在 FreeRTOS 中封装接口的原因： 1、增加互斥锁机制，对于公共 PD，两个模块都在使用的，最好能有锁，这样更安全。 2、增加引用计数，对于公共 PD，两个模块都在使用的，同时开关的时候有引用计数，这样更安全。

2.3.2 FreeRTOS 开关 PD

使用示例：

1、对于复用PD，如PD_AUDIO是PDM、VAD、I2S等模块复用的，所以开关的时候需要使用下面带有引用计数和锁的接口。

```
PD *pd_audio = GetPdFromId(PD_AUDIO);

PdPowerOn(pd_audio);/* power on */
PdPowerOff(pd_audio);/* power off */

PutPd(pd_audio);
```

备注： 因为有引用计数，所以使用的时候注意开关要成对。

2、对于模块专有PD，如PD_DSP。开关的时候可以直接通过ID开关，使用如下直接调用HAL的方式：

```
HAL_PD_On(PD_DSP);/* power on */
HAL_PD_Off(PD_DSP);/* power off */
```

3. TEST

3.1 RT-THREAD

3.1.1 CONFIG配置

```
RT-Thread bsp test case  --->
    RT-Thread Common Test case  --->
        [*] Enable BSP Common TEST
            [*]   Enable BSP Common PM TEST
```

3.1.2 USAGE

```
clk -w <id|name> <rate_hz>    set clk rate;
clk -r <id|name>              get clk rate;
clk -e <id>                   enable clk;
clk -d <id>                   disable clk;
clk_dump                      print clk id;
```

使用示例：

```
/* 设置GPLL频率 594M, GPLL的ID是0 */
clk -w 0 594000000
/* 获取GPLL频率 */
clk -r 0
/* 打印时钟树和部分id */
clk_dump
```

3.2 FreeRTOS

3.2.1 CONFIG配置

```
Components Config  --->
    Command shell  --->
        [*]     Enable PM_TEST Shell
```

3.2.2 USAGE

```
"    clk -w <id> <rate_hz>    set clk rate\r\n"
"    clk -r <id>              get clk rate\r\n"
"    clk -e <id>              enable clk\r\n"
"    clk -d <id>              disable clk\r\n"
"    clk dump                 print clk id\r\n"
```

使用示例:

```
/* 设置GPLL频率 594M, GPLL的ID是0 */
clk -w 0 594000000
/* 获取GPLL频率 */
clk -r 0
/* 打印时钟树和部分id */
clk_dump
```