Rockchip RTOS时钟配置说明

发布版本：1.0

作者邮箱：zhangqing@rock-chips.com

日期：2019.5

文件密级：公开资料

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	RKOS&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

日期	版本	作者	修改说明
2019-05-21	V1.0	Elaine	第一次临时版本发布

---

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

```
struct clk_gate *get_clk_gate_from_id(int clk_id);
void release_clk_gate_id(struct clk_gate *gate);
rt_err_t clk_enable(struct clk_gate *gate, int on);
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：

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

clk_enable(aclk_vio0, 1);/* clk enable */
clk_enable(aclk_vio0, 0);/* clk disable */

release_clk_gate_id(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。

1.2.3 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));
```

1.2.4 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_dump clk_inits[] =
{
    DUMP_CLK("PLL_GPLL", PLL_GPLL, 1188000000),
    DUMP_CLK("PLL_CPLL", PLL_CPLL, 1000000000),
    DUMP_CLK("HCLK_M4", HCLK_M4, 400000000),
    DUMP_CLK("ACLK_DSP", ACLK_DSP, 300000000),
    DUMP_CLK("ACLK_LOGIC", ACLK_LOGIC, 300000000),
    DUMP_CLK("HCLK_LOGIC", HCLK_LOGIC, 150000000),
    DUMP_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，在shell命令行，切到finsh下，直接敲clk_dump()就可以。

1.3 RKOS CLK配置

1.3.1 RKOS CLK接口

```
rk_err_t ClkEnable(CLK_GATE *gate, int on);
int ClkIsEnabled(CLK_GATE *gate);
CLK_GATE *GetClkGateFromId(int clkId);
void ReleaseClkGateId(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 ClkDump(void);
```

在RKOS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。

1.3.2 RKOS 开关CLK

使用示例：

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

ClkEnable(aclk_vio0, 1);/* clk enable */
ClkEnable(aclk_vio0, 0);/* clk disable */

ReleaseClkGateId(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。

1.3.3 RKOS 设置频率

使用示例：

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

1.3.4 RKOS 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[] =
{
    DUMP_CLK("PLL_GPLL", PLL_GPLL, 384000000),
    DUMP_CLK("PLL_VPLL", PLL_VPLL, 491520000),
    DUMP_CLK("CLK_HIFI3", CLK_HIFI3, 164000000),
    DUMP_CLK("HCLK_MCU_BUS", HCLK_MCU_BUS, 200000000),
    DUMP_CLK("PCLK_MCU_BUS", PCLK_MCU_BUS, 100000000),
    DUMP_CLK("SCLK_M4F0", SCLK_M4F0, 200000000),
    DUMP_CLK("ACLK_PERI_BUS", ACLK_PERI_BUS, 200000000),
    DUMP_CLK("HCLK_PERI_BUS", HCLK_PERI_BUS, 100000000),
    DUMP_CLK("HCLK_TOP_BUS", HCLK_TOP_BUS, 100000000),
    DUMP_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使用目前还不支持命令，在需要的位置增加ClkDump()调用。

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：

```
#define PISCES_PD_DSP 0x00000000U
#define PISCES_PD_LOGIC 0x00011111U
#define PISCES_PD_SHRM 0x00022222U
#define PISCES_PD_AUDIO 0x00033333U
```

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_Setting(uint32_t pd, bool powerOn);
```

2.1.3 PD 开关

```
HAL_Status HAL_PD_Setting(uint32_t pd, bool powerOn);
```

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(int pd_id);
void release_pd_id(struct pd *power);
rt_err_t pd_power(struct pd *power, int on);
```

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：

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

pd_power(pd_audio, 1);/* power on */
pd_power(pd_audio, 0);/* power off */

release_pd_id(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。

2.3 RKOS PD配置

2.3.1 RKOS 接口

```
rk_err_t PdPower(PD *power, int on);
PD *GetPdFromId(int pdId);
void ReleasePdId(PD *power);
```

在RKOS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 RKOS 开关PD

使用示例：

```
PD *pd_audio = GetPdFromId(RK2206_PD_AUDIO);

PdPower(pd_audio, 1);/* power on */
PdPower(pd_audio, 0);/* power off */

ReleasePdId(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。