

# Rockchip Developer Guide RTOS Clock

---

文件标识: RK-KF-YF-055

发布版本: 1.2.1

日期: 2019-12-06

文件密级: 公开资料

## 免责声明

本文档按“现状”提供，福州瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

## 商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

版权所有© 2019福州瑞芯微电子股份有限公司

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

福州瑞芯微电子股份有限公司

Fuzhou 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](http://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](mailto: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

日期	版本	作者	修改说明
2019-05-21	V1.0.0	Elaine	第一次临时版本发布
2019-09-19	V1.1.0	Tao Huang	修订 clk dump 相关实现
2019-09-19	V1.2.0	Elaine	修订 接口修改，test 相关实现
2019-12-06	V1.2.1	Elaine	增加文件标识和免责声明

# 目录

## Rockchip Developer Guide RTOS Clock

### 1 CLK 配置

#### 1.1 HAL CLK 配置

##### 1.1.1 HAL 层 CLK 头文件

##### 1.1.2 常用 API

##### 1.1.3 CLK 开关

##### 1.1.4 CLK 频率设置

##### 1.1.5 CLK SOFTRESET

#### 1.2 RT-THREAD CLK 配置

##### 1.2.1 RT-THREAD CLK 接口

##### 1.2.2 RT-THREAD 开关 CLK

#### 1.2.3 RT-THREAD 设置频率

##### 1.2.4 RT-THREAD 设置初始化频率及 CLK DUMP

#### 1.3 FreeRTOS CLK 配置

##### 1.3.1 FreeRTOS CLK 接口

##### 1.3.2 FreeRTOS 开关 CLK

#### 1.3.3 FreeRTOS 设置频率

##### 1.3.4 FreeRTOS 设置初始化频率及 CLK DUMP

### 2 PD 配置

#### 2.1 HAL PD 配置

##### 2.1.1 HAL 层 PD 头文件

##### 2.1.2 常用 API

##### 2.1.3 PD 开关

#### 2.2 RT-THREAD PD 配置

##### 2.2.1 RT-THREAD 接口

##### 2.2.2 RT-THREAD 开关 PD

#### 2.3 FreeRTOS PD配置

##### 2.3.1 FreeRTOS 接口

##### 2.3.2 FreeRTOS 开关 PD

### 3 TEST

#### 3.1 RT-THREAD

##### 3.1.1 CONFIG配置

##### 3.1.2 USAGE

#### 3.2 FreeRTOS

##### 3.2.1 CONFIG配置

##### 3.2.2 USAGE

# 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、SOFTTRST\_ID、DIV\_ID、MUX\_ID、CLK\_ID。  
GATE\_ID: 包含 CON 和 SHIFT,  $CON = GATE\_ID / 16$ ,  $SHIFT = GATE\_ID \% 16$   
SOFTTRST\_ID: 包含 CON 和 SHIFT,  $CON = SOFTTRST\_ID / 16$ ,  $SHIFT = SOFTTRST\_ID \% 16$   
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:

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

### 1.1.2 常用 API

```
1 uint32_t HAL_CRU_GetPllFreq(struct PLL_SETUP *pSetup);
2 HAL_Status HAL_CRU_SetPllFreq(struct PLL_SETUP *pSetup, uint32_t rate);
3 HAL_Check HAL_CRU_ClkIsEnabled(uint32_t clk);
4 HAL_Status HAL_CRU_ClkEnable(uint32_t clk);
5 HAL_Status HAL_CRU_ClkDisable(uint32_t clk);
6 HAL_Check HAL_CRU_ClkIsReset(uint32_t clk);
7 HAL_Status HAL_CRU_ClkResetAssert(uint32_t clk);
8 HAL_Status HAL_CRU_ClkResetDeassert(uint32_t clk);
9 HAL_Status HAL_CRU_ClkSetDiv(uint32_t divName, uint32_t divValue);
10 uint32_t HAL_CRU_ClkGetDiv(uint32_t divName);
11 HAL_Status HAL_CRU_ClkSetMux(uint32_t muxName, uint32_t muxValue);
12 uint32_t HAL_CRU_ClkGetMux(uint32_t muxName);
13 HAL_Status HAL_CRU_FracdivGetConfig(uint32_t rateOut, uint32_t rate,
14                                     uint32_t *numerator,
15                                     uint32_t *denominator);
16 uint32_t HAL_CRU_ClkGetFreq(eCLOCK_Name clockName);
17 HAL_Status HAL_CRU_ClkSetFreq(eCLOCK_Name clockName, uint32_t rate);
18 HAL_Status HAL_CRU_ClkNp5BestDiv(eCLOCK_Name clockName, uint32_t rate,
19                                  uint32_t pRate, uint32_t *bestdiv);
19
```

### 1.1.3 CLK 开关

```
1 HAL_Check HAL_CRU_ClkIsEnabled(uint32_t clk);
2 HAL_Status HAL_CRU_ClkEnable(uint32_t clk);
3 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

```
1  uint32_t HAL_CRU_ClkGetFreq(eCLOCK_Name clockName);
2  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）。

```
1  HAL_Status HAL_CRU_ClkSetDiv(uint32_t divName, uint32_t divValue);
2  uint32_t HAL_CRU_ClkGetDiv(uint32_t divName);
3  HAL_Status HAL_CRU_ClkSetMux(uint32_t muxName, uint32_t muxValue);
4  uint32_t HAL_CRU_ClkGetMux(uint32_t muxName);
```

### 1.1.5 CLK SOFTRESET

```
1  HAL_Check HAL_CRU_ClkIsReset(uint32_t clk);
2  HAL_Status HAL_CRU_ClkResetAssert(uint32_t clk);
3  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

```
1  rt_err_t clk_enable_by_id(int gate_id);
2  rt_err_t clk_disable_by_id(int gate_id);
3  struct clk_gate *get_clk_gate_from_id(int gate_id);
4  void put_clk_gate(struct clk_gate *gate);
5  rt_err_t clk_enable(struct clk_gate *gate);
6  rt_err_t clk_disable(struct clk_gate *gate);
7  int clk_is_enabled(struct clk_gate *gate);
8  uint32_t clk_get_rate(eCLOCK_Name clk_id);
9  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。

```
1  struct clk_gate *aclk_vio0 = get_clk_gate_from_id(ACLK_VIO0_GATE);
2
3  clk_enable(aclk_vio0); /* clk enable */
4  clk_disable(aclk_vio0); /* clk disable */
5
6  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：

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

## 1.2.3 RT-THREAD 设置频率

使用示例：

```
1  clk_set_rate(clk_id, init_rate_hz);
2  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：

```
1  static const struct clk_init clk_inits[] =
2  {
3      INIT_CLK("PLL_GPLL", PLL_GPLL, 1188000000),
4      INIT_CLK("PLL_CPLL", PLL_CPLL, 1000000000),
5      INIT_CLK("HCLK_M4", HCLK_M4, 400000000),
6      INIT_CLK("ACLK_DSP", ACLK_DSP, 300000000),
7      INIT_CLK("ACLK_LOGIC", ACLK_LOGIC, 300000000),
8      INIT_CLK("HCLK_LOGIC", HCLK_LOGIC, 150000000),
9      INIT_CLK("PCLK_LOGIC", PCLK_LOGIC, 150000000),
10 };
```

```
1  void rt_hw_board_init()
2  {
3      .....
4      clk_init(clk_inits, HAL_ARRAY_SIZE(clk_inits), true);
5      .....
6  }
```

(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。

## 1.3 FreeRTOS CLK 配置

### 1.3.1 FreeRTOS CLK 接口

```
1  rk_err_t ClkEnableById(int gateId);
2  rk_err_t ClkDisableById(int gateId);
3  rk_err_t ClkEnable(CLK_GATE *gate);
4  rk_err_t ClkDisable(CLK_GATE *gate);
5  int ClkIsEnabled(CLK_GATE *gate);
6  CLK_GATE *GetClkGateFromId(int gateId);
7  void PutClkGate(CLK_GATE *gate);
8  uint32_t ClkGetRate(eCLOCK_Name clkId);
```

```

9   rk_err_t ClkSetRate(eCLOCK_Name clkId, uint32_t rate);
10  uint32 GetHclkSysCoreFreq(void);
11  rk_err_t ClkDevInit(void);
12  rk_err_t ClkDevDeinit(void);
13  void ClkInit(const CLK_INIT *clkInits, uint32 clkCount, bool clkDump);
14  void ClkDisableUnused(const CLK_UNUSED *clksUnused, uint32 clkUnusedCount);
15  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。

### 1.3.2 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。

```

1   CLK_GATE *aclk_vio0 = GetClkGateFromId(ACLK_VIO0_GATE);
2
3   ClkEnable(aclk_vio0);/* clk enable */
4   ClkDisable(aclk_vio0);/* clk disable */
5
6   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：

```

1   ClkEnableById(ACLK_DSP_GATE);/* clk enable */
2   ClkDisableById(ACLK_DSP_GATE);/* clk disable */

```

### 1.3.3 FreeRTOS 设置频率

使用示例：

```

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

```

### 1.3.4 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：

```

1  static const CLK_INIT clkInits[] =
2  {
3      INIT_CLK("PLL_GPLL", PLL_GPLL, 384000000),
4      INIT_CLK("PLL_VPLL", PLL_VPLL, 491520000),
5      INIT_CLK("CLK_HIFI3", CLK_HIFI3, 164000000),
6      INIT_CLK("HCLK_MCU_BUS", HCLK_MCU_BUS, 200000000),
7      INIT_CLK("PCLK_MCU_BUS", PCLK_MCU_BUS, 100000000),
8      INIT_CLK("SCLK_M4F0", SCLK_M4F0, 200000000),
9      INIT_CLK("ACLK_PERI_BUS", ACLK_PERI_BUS, 200000000),
10     INIT_CLK("HCLK_PERI_BUS", HCLK_PERI_BUS, 100000000),
11     INIT_CLK("HCLK_TOP_BUS", HCLK_TOP_BUS, 100000000),
12     INIT_CLK("PCLK_TOP_BUS", PCLK_TOP_BUS, 100000000),
13 };

```

```

1  void ClkDevHwInit(void)
2  {
3      ClkDevInit();
4      ClkInit(clkInits, HAL_ARRAY_SIZE(clkInits), true);
5  }
6
7  void ClkDevHwDeInit(void)
8  {
9      ClkDevDeinit();
10 }

```

## (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：

```

1  #ifndef __ASSEMBLY__
2  typedef enum PD_Id {
3      PD_DSP                = 0x80000000U,
4      PD_LOGIC              = 0x80011111U,
5      PD_SHRM               = 0x80022222U,
6      PD_AUDIO              = 0x80033333U,
7  } ePD_Id;
8  #endif

```

按照下面定义，对应填写 PWR\_SHIFT, ST\_SHIFT, REQ\_SHIFT, ACK\_SHIFT。

```

1  #define PD_PWR_SHIFT      0U
2  #define PD_PWR_MASK      0x0000000FU
3  #define PD_ST_SHIFT      4U
4  #define PD_ST_MASK      0x000000F0U

```



```

5 | #define PD_REQ_SHIFT 8U
6 | #define PD_REQ_MASK 0x00000F00U
7 | #define PD_IDLE_SHIFT 12U
8 | #define PD_IDLE_MASK 0x0000F000U
9 | #define PD_ACK_SHIFT 16U
10 | #define PD_ACK_MASK 0x000F0000U
11 |
12 | #define PD_GET_PWR_SHIFT(x) (((uint32_t)(x) & PD_PWR_MASK) >> PD_PWR_SHIFT)
13 | #define PD_GET_ST_SHIFT(x) (((uint32_t)(x) & PD_ST_MASK) >> PD_ST_SHIFT)
14 | #define PD_GET_REQ_SHIFT(x) (((uint32_t)(x) & PD_REQ_MASK) >> PD_REQ_SHIFT)
15 | #if defined(RKMCU_RK1808)
16 | #define PD_GET_IDLE_SHIFT(x) (((uint32_t)(x) & PD_IDLE_MASK) >>
    PD_IDLE_SHIFT) + 16)
17 | #else
18 | #define PD_GET_IDLE_SHIFT(x) (((uint32_t)(x) & PD_IDLE_MASK) >> PD_IDLE_SHIFT)
19 | #endif
20 | #define PD_GET_ACK_SHIFT(x) (((uint32_t)(x) & PD_ACK_MASK) >> PD_ACK_SHIFT)

```

## 2.1.2 常用 API

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

## 2.1.3 PD 开关

```
1 | 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

```

1 | struct pd *get_pd_from_id(ePD_Id pd_id);
2 | void put_pd(struct pd *power);
3 | rt_err_t pd_on(struct pd *power);
4 | 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。

```

1 struct pd *pd_audio = get_pd_from_id(PD_AUDIO);
2
3 pd_on(pd_audio);/* power on */
4 pd_off(pd_audio);/* power off */
5
6 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：

```

1 HAL_PD_On(PD_DSP);/* power on */
2 HAL_PD_Off(PD_DSP);/* power off */

```

## 2.3 FreeRTOS PD配置

### 2.3.1 FreeRTOS 接口

```

1 rk_err_t PdPowerOn(PD *power);
2 rk_err_t PdPowerOff(PD *power);
3 PD *GetPdFromId(int pdId);
4 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。

```

1 PD *pd_audio = GetPdFromId(PD_AUDIO);
2
3 PdPowerOn(pd_audio);/* power on */
4 PdPowerOff(pd_audio);/* power off */
5
6 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：

```

1 HAL_PD_On(PD_DSP);/* power on */
2 HAL_PD_Off(PD_DSP);/* power off */

```

## 3 TEST

### 3.1 RT-THREAD

#### 3.1.1 CONFIG配置

```

1 RT-Thread bsp test case --->
2     RT-Thread Common Test case --->
3     [*] Enable BSP Common TEST
4     [*] Enable BSP Common PM TEST

```

### 3.1.2 USAGE

```

1 clk -w <id|name> <rate_hz>    set clk rate;
2 clk -r <id|name>              get clk rate;
3 clk -e <id>                   enable clk;
4 clk -d <id>                   disable clk;
5 clk_dump                      print clk id;

```

使用示例:

```

1 /* 设置GPLL频率 594M, GPLL的ID是0 */
2 clk -w 0 594000000
3 /* 获取GPLL频率 */
4 clk -r 0
5 /* 打印时钟树和部分id */
6 clk_dump

```

## 3.2 FreeRTOS

### 3.2.1 CONFIG配置

```

1 Components Config --->
2     Command shell --->
3     [*] Enable PM_TEST Shell

```

### 3.2.2 USAGE

```

1 "    clk -w <id> <rate_hz>    set clk rate\r\n"
2 "    clk -r <id>              get clk rate\r\n"
3 "    clk -e <id>              enable clk\r\n"
4 "    clk -d <id>              disable clk\r\n"
5 "    clk dump                  print clk id\r\n"

```

使用示例:

```

1 /* 设置GPLL频率 594M, GPLL的ID是0 */
2 clk -w 0 594000000
3 /* 获取GPLL频率 */
4 clk -r 0
5 /* 打印时钟树和部分id */
6 clk_dump

```