



Linux CCU 开发指南

版本号: 2.1
发布日期: 2021.04.22

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.06.29	AWA1440	添加初版
2.0	2020.11.19	AWA1527	for linux-5.4
2.1	2021.04.22	XAA0191	增加部分使用描述



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	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 源码结构介绍	2
2.4 模块配置介绍	4
2.4.1 kernel menuconfig 配置	4
2.4.2 device tree 源码结构和路径	4
2.4.2.1 linux4.9	4
2.4.2.2 linux5.4	5
2.4.3 模块时钟在 dts 的配置	5
2.4.4 PLL 小数分频配置	6
2.5 系统时钟结构	8
2.6 模块时钟结构	9
3 模块接口说明	10
3.1 时钟 API 定义	10
3.2 时钟 API 说明	10
3.2.1 clk_get	10
3.2.2 devm_clk_get(推荐使用)	11
3.2.3 clk_put	11
3.2.4 of_clk_get(推荐使用)	11
3.2.5 clk_set_parent	12
3.2.6 clk_get_parent	12
3.2.7 clk_prepare	12
3.2.8 clk_enable	13
3.2.9 clk_prepare_enable (推荐使用)	13
3.2.10 clk_disable	13
3.2.11 clk_unprepare	14
3.2.12 clk_disable_unprepare (推荐使用)	14
3.2.13 clk_get_rate	14
3.2.14 clk_set_rate	15
3.2.15 sunxi_periph_reset_assert	15
3.2.16 sunxi_periph_reset_deassert	15
4 模块使用范例	17
5 FAQ	18

5.1 常用 debug 方法说明	18
5.1.1 clk tree	18
5.1.1.1 clk debugfs	20
5.1.1.2 利用 sunxi_dump 读写相应寄存器	23



插 图

2-1 内核 menuconfig 菜单	4
2-2 系统时钟结构图	8
2-3 总线时钟结构图	9
2-4 模块时钟结构图	9
5-1 内核 menuconfig 根菜单	21
5-2 Common clock framework 菜单	22
5-3 clk DebugFS 菜单	22



1 概述

1.1 编写目的

本文档对 Sunxi 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

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
全志所有产品	Linux-4.9 及以上	drivers/clock/sunxi*/*.c

1.3 相关人员

本文档适用于所有需要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

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 源码结构介绍

CCU 的源码结构如下图所示：

```
linux
|
|-- drivers
|   |-- clk
|       |-- sunxi //适用于linux-4.9
|           |-- clk-periph.h
|           |-- clk-periph.c
|           |-- clk-factors.h
|           |-- clk-factors.c
|           |-- clk-cpu.h
|           |-- clk-cpu.c
```


2.4 模块配置介绍

2.4.1 kernel menuconfig 配置

在 sunxi linux-4.9 平台上，目前 ccu 驱动是依赖 CONFIG_ARCH_SUNXI 这个宏的，因此在内核 menuconfig 菜单下，目前没有提供配置菜单。

Linux-5.4 内核版本执行：./build.sh menuconfig 进入配置主界面，并进入如下目录勾选对应驱动：

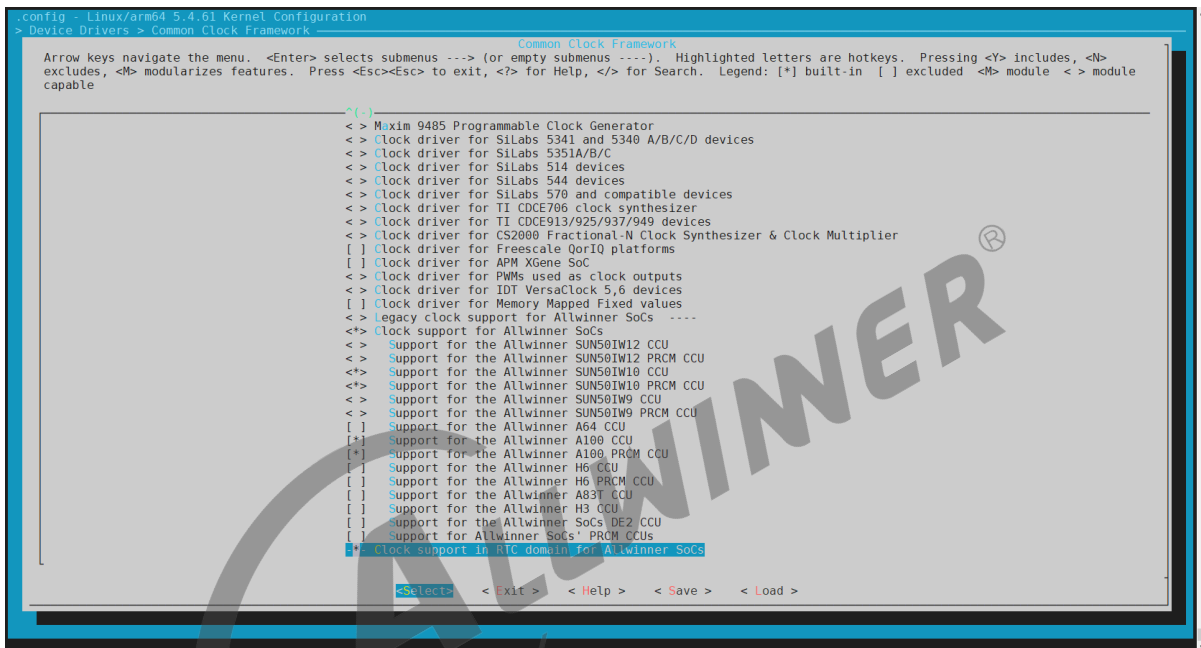


图 2-1: 内核 menuconfig 菜单

2.4.2 device tree 源码结构和路径

2.4.2.1 linux4.9

- 设备树文件的配置是该 SoC 所有方案的通用配置，对于 ARM64 CPU 而言，设备树的路径为：kernel/{KERNEL}/arch/arm64/boot/dts/sunxi/sun*-clk.dtsi。
- 设备树文件的配置是该 SoC 所有方案的通用配置，对于 ARM32 CPU 而言，设备树的路径为：kernel/{KERNEL}/arch/arm/boot/dts/sun*-clk.dtsi。
- 板级设备树 (board.dts) 路径：/device/config/chips/{IC}/configs/{BOARD}/board.dts

device tree 的源码结构关系如下：

```
board.dts
|-----sun*.dtsi
|-----sun*-pinctrl.dtsi
|-----sun*-clk.dtsi
```

2.4.2.2 linux5.4

- linux5.4 的时钟树描述已从 dts 中移除，只保留必要的 ccu 设备节点

device tree 的源码结构关系如下：

```
board.dts
|-----sun*.dtsi
```

2.4.3 模块时钟在 dts 的配置

对于 CLK 子系统的使用者，模块时钟的配置，一般都在模块框架节点下配置模块使用的时钟，模块使用的时钟频率，同时也可以通过在模块驱动匹配阶段，通过时钟的 API 对时钟进行进一步的配置（配置父时钟，时钟频率调整等）。

对于 CLK 子系统本身，也可以在 dts 中对系统时钟进行描述，通过时钟子系统框架进行时钟的注册。

说明如下：

1. 时钟在模块的配置（CLK 使用者配置）

```
对于linux-4.9:
g2d: g2d@01480000 {
    compatible = "allwinner,sunxi-g2d";
    reg = <0x0 0x01480000 0x0 0x3ffff>;
    interrupts = <GIC_SPI 90 IRQ_TYPE_LEVEL_HIGH>;
    /* 配置模块使用的clk */
    clocks = <&clk_g2d>;
    iommu = <&mmu_aw 6 1>;
};
对于linux-5.4:
g2d: g2d@6480000 {
    compatible = "allwinner,sunxi-g2d";
    reg = <0x0 0x06480000 0x0 0x3ffff>;
    interrupts = <GIC_SPI 91 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&ccu CLK_BUS_G2D>, <&ccu CLK_G2D>, <&ccu CLK_MBUS_G2D>;
    clock-names = "bus", "g2d", "mbus_g2d";
    resets = <&ccu RST_BUS_G2D>;
    iommu = <&mmu_aw 5 1>;
    assigned-clocks = <&ccu CLK_G2D>; /* 指定CLK_G2D这个时钟 */
    assigned-clock-rates = <300000000>; /* 指定时钟频率 */
    assigned-clock-parents = <&ccu xxx>; /* 指定CLK_G2D的父时钟 */
```

```
};
```

2. 时钟在 sun*-clk.dtsi 的配置 (CLK 提供者配置)

- linux-4.9

linux-4.9 中支持在 dts 中时钟树进行描述，由时钟子系统进行解析，注册到时钟框架中

```
clk_g2d: g2d {
    #clock-cells = <0>;
    compatible = "allwinner,periph-clock";
    /* 指定clk_g2d 这个clk的父时钟 */
    assigned-clock-parents = <&clk_pll_periph0x2>;
    /* 指定clk_g2d 这个clk的频率 */
    assigned-clock-rates = <300000000>;
    assigned-clocks = <&clk_g2d>;
    clock-output-names = "g2d";
};
```

- linux5.4

linux-5.4 中仅支持对“fixed-clock”类型时钟的描述，其余的时钟描述放到了 c 代码中

```
dcxo24M: dcxo24M-clk {
    #clock-cells = <0>;
    compatible = "fixed-clock";
    clock-frequency = <24000000>;
    clock-output-names = "dcxo24M";
};
```

2.4.4 PLL 小数分频配置

PLL 小数分频，一般情况下是没有打开的，但当产品面临 EMI 问题时，可能需要配置相关 PLL 进行小数分频，在 sunxi 平台的 SDK 中，小数分频不支持通过 dts 等方便的配置文件进行配置，只能修改代码。

对于 linux-4.9: 1. 找到所使用方案的 clk-sun*.c(如 clk-sun50iw9.c)，并打开该文件。

2. 找到 PLL 表，如下所示

```
1 /*          out  mode}  en-s  sdmss      ns  nw  ks  kw  ms  mw  ps  pw  d1s  d1w  d2s  d2w {frac
2  SUNXI_CLK_FACTORS(pll_cpu,      8,  8,  0,  0,  0,  2, 16, 2,  0,  0,  0,  0,  0,
   0,  0,  27,  0,  0,  0,  0,  0);
3  SUNXI_CLK_FACTORS(pll_ddr0,     8,  8,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  0,  1,  0,
```

```

0, 0, 27, 24, 1, PLL_DDR0PAT, 0xd1303333);
4 SUNXI_CLK_FACTORS(pll_dds1, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 27, 24, 1, PLL_DDR0PAT, 0xd1303333);
5 SUNXI_CLK_FACTORS(pll_periph0, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 27, 24, 1, PLL_PERI0PAT0, 0xd1303333);
6 SUNXI_CLK_FACTORS(pll_periph1, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 27, 24, 1, PLL_PERI1PAT0, 0xd1303333);
7 SUNXI_CLK_FACTORS(pll_gpu, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 27, 24, 0, PLL_GPUPAT0, 0xd1303333);
8 SUNXI_CLK_FACTORS(pll_video0x4, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
0, 0, 27, 24, 0, PLL_VIDEO0PAT0, 0xd1303333);
9 SUNXI_CLK_FACTORS(pll_video1, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
0, 0, 27, 24, 0, PLL_VIDEO1PAT0, 0xd1303333);
10 SUNXI_CLK_FACTORS(pll_video2, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
0, 0, 27, 24, 0, PLL_VIDEO2PAT0, 0xd1303333);
11 SUNXI_CLK_FACTORS(pll_ve, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 27, 24, 0, PLL_VEPAT0, 0xd1303333);
12 SUNXI_CLK_FACTORS(pll_de, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 27, 24, 0, PLL_DEPAT0, 0xd1303333);
13 SUNXI_CLK_FACTORS(pll_audiox4, 8, 8, 0, 0, 0, 0, 16, 6, 1, 1, 0, 1, 0,
0, 0, 27, 24, 1, PLL_AUDIOPAT0, 0xc00121ff);
14 SUNXI_CLK_FACTORS(pll_csi, 8, 8, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 27, 24, 0, 0, 0);
15
16 说明:
17
18 sdmss:这一列表示小数分频控制寄存器的使能bit, sunxi平台目前都在bit 24
19 sdmssw:这一列表示是否使能小数分频, 1表示使能, 0表示关闭
20 sdmssw:这一列表示小数分频的配置寄存器, 请根据spec或data sheet进行配置
21 sdmssw:这一列表示小数分频配置寄存器的值, 根据自己需要分频的情况, 可以自行调整。

```

对于 linux-5.4: 1. 找到所使用方案的 clk-sun*.c(如 ccu-sun50iw10.c), 并打开该文件。

2. 找到对应时钟, 如下所示

```

1 #define SUN50IW10_PLL_COM_REG 0x060
2 static struct ccu_sdm_setting pll_com_sdm_table[] = {
3     { .rate = 451584000, .pattern = 0xc0014396, .m = 2, .n = 37 },
4 };
5
6 static struct ccu_nm pll_com_clk = {
7     .enable = BIT(27),
8     .lock = BIT(28),
9     .n = _SUNXI_CCU_MULT_MIN(8, 8, 12),
10    .m = _SUNXI_CCU_DIV(0, 1),
11    .sdm = _SUNXI_CCU_SDM(pll_com_sdm_table, BIT(24),
12        0x160, BIT(31)),
13    .common = {
14        .reg = 0x060,
15        .features = CCU_FEATURE_SIGMA_DELTA_MOD,
16        .hw.init = CLK_HW_INIT("pll-com", "dcxo24M",
17            &ccu_nm_ops,
18            CLK_SET_RATE_UNGATE),
19    },
20 };
21 pll_com_sdm_table描述了小数分频的实际频率、pattern寄存器值、分频系数配置。
22 _SUNXI_CCU_SDM宏描述了, sdm寄存器以及相关使能位。

```

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。

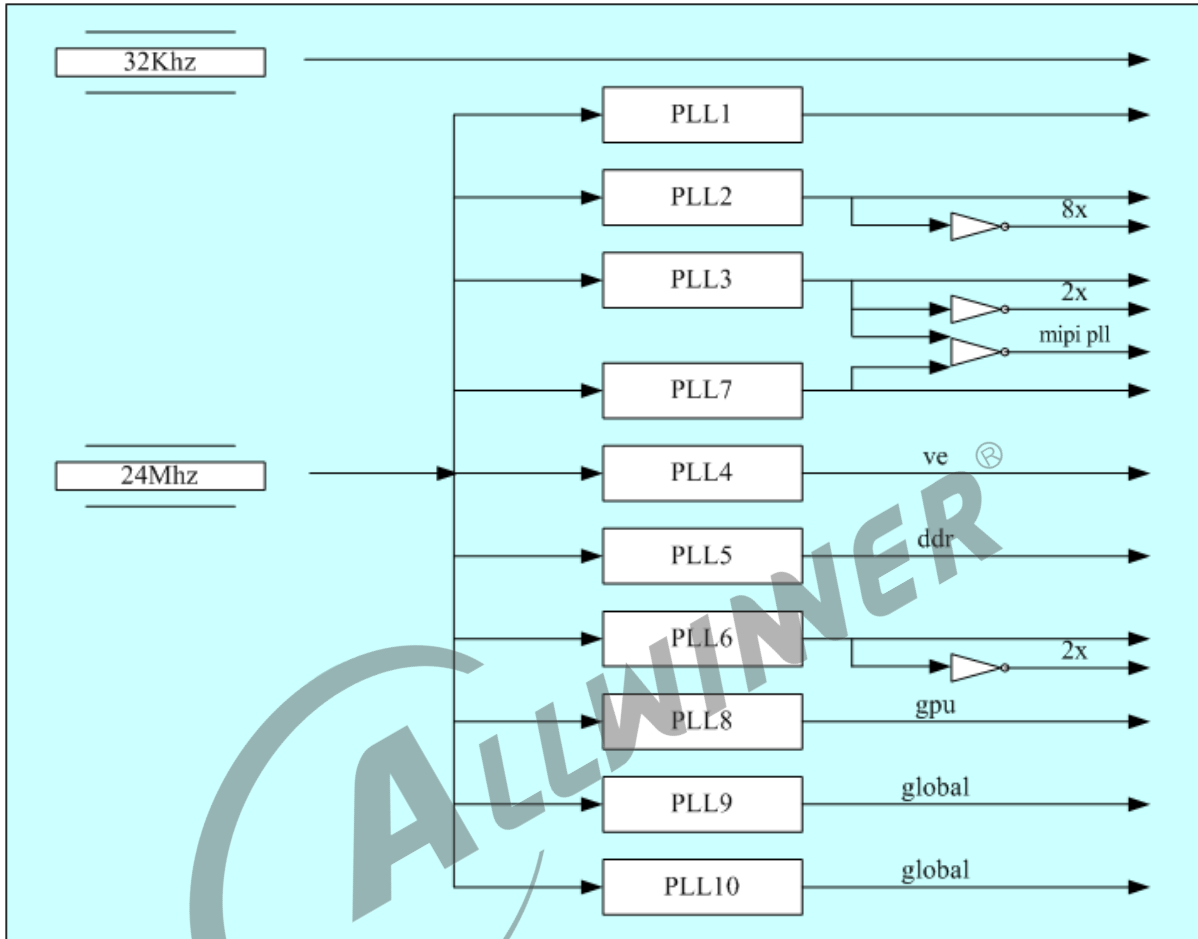


图 2-2: 系统时钟结构图

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，其工作时钟也经常输出作为某些其它模块的时钟源，因此，也将此类时钟归结为系统时钟。其结构图如下：

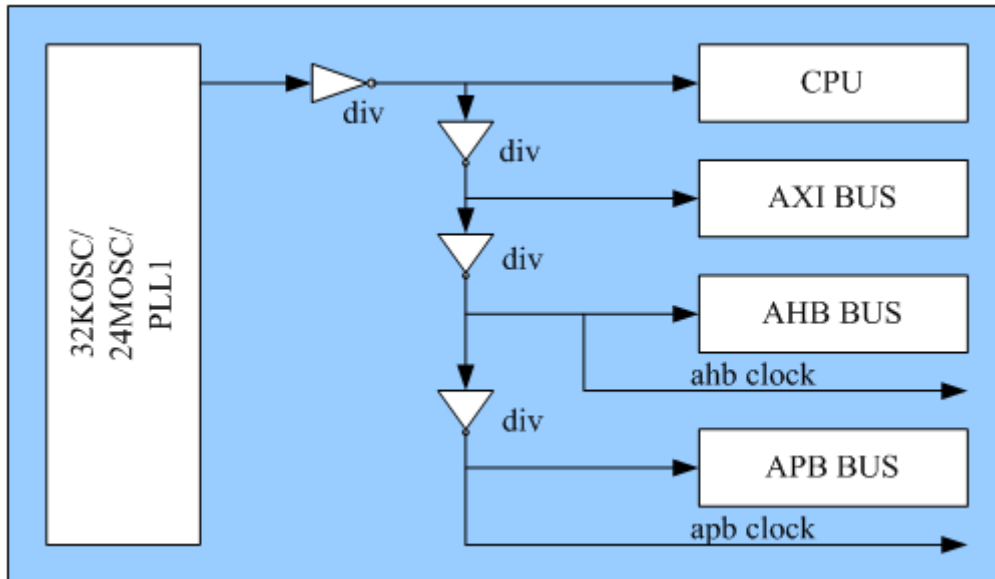


图 2-3: 总线时钟结构图

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 gating、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。

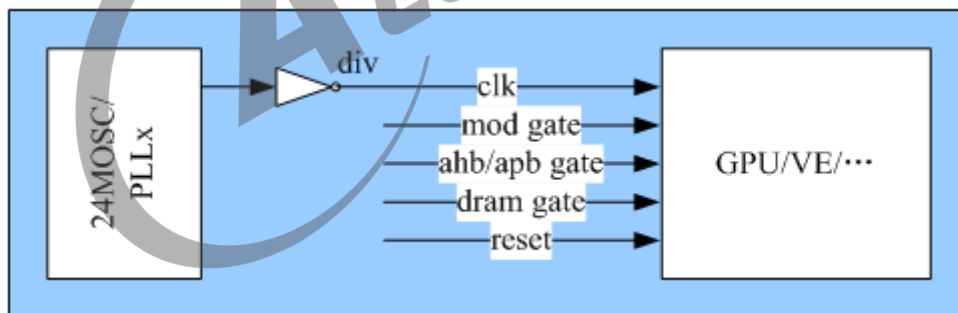


图 2-4: 模块时钟结构图

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。

3 模块接口说明

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

3.1 时钟 API 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的 API，Sunxi 平台的时钟 API 遵循该 API 规范。

3.2 时钟 API 说明

3.2.1 clk_get

- 作用：获取管理时钟的时钟句柄。
- 参数：
 - dev: 指向申请时钟的设备句柄。
 - id: 指向要申请的时钟名，可以为 NULL。
- 返回：
 - 成功，返回时钟句柄。
 - 失败，返回 NULL。

警告

该接口要与 clk_put 成对使用。

说明

该函数用于申请指定时钟名的时钟句柄，所有的时钟操作都基于该时钟句柄来实现。

3.2.2 devm_clk_get(推荐使用)

- 作用：获取管理时钟的时钟句柄。
- 参数：
 - dev: 指向申请时钟的设备句柄。
 - id: 指向要申请的时钟名（字符串），可以为 NULL。
- 返回：
 - 成功，返回时钟句柄。
 - 失败，返回 NULL。

📖 说明

该函数用于申请指定时钟名的时钟句柄，所有的时钟操作都基于该时钟句柄来实现。和 `clk_get` 的区别在于：一般用在 `driver` 的 `probe` 函数里申请时钟句柄，而当 `driver probe` 失败或者 `driver remove` 时，`driver` 会自动释放对应的时钟句柄（即相当于系统自动调用 `clk_put`）。

3.2.3 clk_put

- 作用：释放管理时钟的时钟句柄。
- 参数：
 - clk: 指向申请时钟的设备句柄。
- 返回：
 - 没有返回值。

⚠ 警告

该接口要与 `clk_get` 成对使用。

📖 说明

该函数用于释放成功申请到的时钟句柄，当不再使用时，需要释放时钟句柄。

3.2.4 of_clk_get(推荐使用)

- 作用：获取管理时钟的时钟句柄。
- 参数：
 - np: 指向设备的 `device_node` 结点的指针。
 - index: 在 `dts` 中属性的索引值。
- 返回：
 - 成功，返回时钟句柄。
 - 失败，返回 NULL。

3.2.5 clk_set_parent

- 作用：用于设定指定时钟的父时钟。
- 参数：
 - clk: 待操作的时钟句柄。
 - parent: 父时钟的时钟句柄。
- 返回：
 - 成功，返回 0。
 - 失败，返回负值的错误码。

3.2.6 clk_get_parent

- 作用：用于获取指定时钟的父时钟。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 成功，返回父时钟句柄。
 - 失败，返回 NULL。

3.2.7 clk_prepare

- 作用：用于 prepare 指定的时钟。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 成功，返回 0。
 - 失败，返回错误码。

📖 说明

旧版本 *kernel* 的 *clk_enable* 在新 *kernel* 中分解成不可在原子上下文调用的 *clk_prepare*（该函数可能睡眠）和可以在原子上下文调用的 *clk_enable*。而 *clk_prepare_enable* 则同时完成 *prepare* 和 *enable* 的工作，只能在可能睡眠的上下文调用该 *API*。

3.2.8 clk_enable

- 作用：用于 enable 指定的时钟。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 成功，返回 0。
 - 失败，返回错误码。

📖 说明

旧版本 *kernel* 的 *clk_enable* 在新 *kernel* 中分解成不可在原子上下文调用的 *clk_prepare*（该函数可能睡眠）和可以在原子上下文调用的 *clk_enable*。因此在 *clk_enable* 之前至少调用了一次 *clk_prepare*，也可用 *clk_prepare_enable* 同时完成 *prepare* 和 *enable* 的工作，只能在可以睡眠的上下文调用该 *API*。

3.2.9 clk_prepare_enable（推荐使用）

- 作用：用于 prepare 并使能指定的时钟。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 成功，返回 0。
 - 失败，返回错误码

📖 说明

旧版本 *kernel* 的 *clk_enable* 在新 *kernel* 中分解成不可在原子上下文调用的 *clk_prepare*（该函数可能睡眠）和可以在原子上下文调用的 *clk_enable*。因此在 *clk_enable* 之前至少调用了一次 *clk_prepare*，也可用 *clk_prepare_enable* 同时完成 *prepare* 和 *enable* 的工作，只能在可以睡眠的上下文调用该 *API*。

3.2.10 clk_disable

- 作用：用于关闭指定的时钟。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 无返回值

⚠️ 警告

该接口要与 *clk_enable* 成对使用。

3.2.11 clk_unprepare

- 作用：用于 unprepare 指定的时钟。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 无返回值

警告

该接口要与 `clk_prepare` 成对使用。

说明

旧版本 `kernel` 的 `clk_disable` 在新 `kernel` 中分解成可以在原子上下文调用的 `clk_disable` 和不可在原子上下文调用的 `clk_unprepare` (该函数可能睡眠) 和 `clk_disable_unprepare` 同时成 `disable` 和 `unprepare` 的工作, 只能在可能睡眠的上下文调用该 `API`。

3.2.12 clk_disable_unprepare (推荐使用)

- 作用：用于 unprepare 和关闭指定的时钟。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 无返回值

警告

该接口要与 `clk_prepare_enable` 成对使用。

说明

旧版本 `kernel` 的 `clk_disable` 在新 `kernel` 中分解成可以在原子上下文调用的 `clk_disable` 和不可在原子上下文调用的 `clk_unprepare` (该函数可能睡眠) 和 `clk_disable_unprepare` 同时完成 `disable` 和 `unprepare` 的工作, 只能在可能睡眠的上下文调用该 `API`。

3.2.13 clk_get_rate

- 作用：用于获取指定时钟当前的频率, 无论时钟是否已经使能。
- 参数：

- clk: 待操作的时钟句柄。
- 返回：
 - 成功，返回指定时钟的频率。
 - 失败，返回 0。

3.2.14 clk_set_rate

- 作用：用于设置时钟频率成功，无论时钟是否已经使能。
- 参数：
 - clk: 待操作的时钟句柄。
 - rate: 希望设置的频率。
- 返回：
 - 成功，返回 0。
 - 失败，返回错误码。

3.2.15 sunxi_periph_reset_assert

- 作用：用于 assert 模块。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：
 - 成功，返回 0。
 - 失败，返回 1。

警告

该接口是 allwinner 私有 API，不建议驱动调用。(linux-5.4 上已被废弃。) linux5.4 中建议使用内核通用接口代替：reset_control_assert

3.2.16 sunxi_periph_reset_deassert

- 作用：用于 deassert 模块。
- 参数：
 - clk: 待操作的时钟句柄。
- 返回：

- 成功，返回 0。
- 失败，返回 1。

警告

该接口是 allwinner 私有 API，不建议驱动调用。(linux-5.4 上已被废弃。)
linux5.4 中建议使用内核通用接口代替：`reset_control_deassert`

说明

该接口和 `sunxi_periph_reset_assert` 一起使用，用于模块 `reset` 操作。



4 模块使用范例

以 spi 模块的时钟处理部分作为 demo 分析：

```
1 static int sunxi_spi_clk_init(struct sunxi_spi *sspi, u32 mod_clk)
2 {
3     int ret = 0;
4     long rate = 0;
5
6     /* 获取index = 0的clk句柄 */
7     sspi->pclk = of_clk_get(sspi->pdev->dev.of_node, 0);
8     /* 对clk句柄的有效性进行判断 */
9     if (IS_ERR_OR_NULL(sspi->pclk)) {
10        SPI_ERR("[spi-%d] Unable to acquire module clock '%s', return %x\n"②,
11              sspi->master->bus_num, sspi->dev_name, PTR_RET(sspi->pclk));
12        return -1;
13    }
14    /* 获取index = 1的clk句柄并判断有效性 */
15    sspi->mclk = of_clk_get(sspi->pdev->dev.of_node, 1);
16    if (IS_ERR_OR_NULL(sspi->mclk)) {
17        SPI_ERR("[spi-%d] Unable to acquire module clock '%s', return %x\n",
18              sspi->master->bus_num, sspi->dev_name, PTR_RET(sspi->mclk));
19        return -1;
20    }
21    /* 设置clk的父时钟并判断有效性 */
22    ret = clk_set_parent(sspi->mclk, sspi->pclk);
23    if (ret != 0) {
24        SPI_ERR("[spi-%d] clk_set_parent() failed! return %d\n",
25              sspi->master->bus_num, ret);
26        return -1;
27    }
28
29    rate = clk_round_rate(sspi->mclk, mod_clk);
30    /* 设置clk的频率并判断有效性 */
31    if (clk_set_rate(sspi->mclk, rate)) {
32        SPI_ERR("[spi-%d] spi clk_set_rate failed\n", sspi->master->bus_num);
33        return -1;
34    }
35
36    SPI_INF("[spi-%d] mclk %u\n", sspi->master->bus_num, (unsigned)clk_get_rate(sspi->mclk)
37          );
38    /* 使能clk并判断有效性 */
39    if (clk_prepare_enable(sspi->mclk)) {
40        SPI_ERR("[spi-%d] Couldn't enable module clock 'spi'\n", sspi->master->bus_num);
41        return -EBUSY;
42    }
43    /* 获取clk的频率值 */
44    return clk_get_rate(sspi->mclk);
45 }
```

5 FAQ

5.1 常用 debug 方法说明

5.1.1 clk tree

- 1. 查看 clk tree 看 clk 频率和父时钟是否正确，按以下步骤操作：

```
mount -t debugfs none /sys/kernel/debug
console:/ # ls sys/kernel/debug/clk/
clk_dump          clk_orphan_dump   clk_orphan_summary clk_summary
console:/ # cat sys/kernel/debug/clk/clk_summary
```

clock	enable_cnt	prepare_cnt	rate	accuracy	phase
pll_periph0div25m	0	0	25000000	0 0	
ephy_25m	0	0	25000000	0 0	
hoscdiv32k	1	1	32768	0 0	
hosc32k	1	1	32768	0 0	
losc_out	2	2	32768	0 0	
osc48m	0	0	48000000	0 0	
osc48md4	0	0	12000000	0 0	
usbohci3_12m	0	0	12000000	0 0	
usbohci2_12m	0	0	12000000	0 0	
usbohci1_12m	0	0	12000000	0 0	
usbohci0_12m	0	0	12000000	0 0	
hosc	20	21	24000000	0 0	
sdmmc0_mod	0	0	8000000	0 0	
cpurcir	1	1	24000000	0 0	
dcxo_out	0	0	24000000	0 0	
cpurapbs2	0	0	24000000	0 0	
cpurcan	0	0	24000000	0 0	
cpurcpus	1	1	24000000	0 0	
cpurahbs	1	1	24000000	0 0	
cpurapbs1	2	2	24000000	0 0	
stwi	1	1	24000000	0 0	
cpurpio	1	1	24000000	0 0	
csi_master1	0	0	24000000	0 0	
csi_master0	0	0	24000000	0 0	
hdmi_slow	1	1	24000000	0 0	
usbphy3	2	2	24000000	0 0	
usbphy2	2	2	24000000	0 0	
usbphy1	2	2	24000000	0 0	
usbphy0	1	1	24000000	0 0	
ths	1	1	24000000	0 0	
ts	0	0	24000000	0 0	
gpadc	0	0	24000000	0 0	
spi1	0	0	24000000	0 0	
spi0	0	0	24000000	0 0	
sdmmc2_rst	1	1	24000000	0 0	

sdmmc2_bus	1	1	24000000	0 0
sdmmc1_rst	1	1	24000000	0 0
sdmmc1_bus	1	1	24000000	0 0
sdmmc0_rst	0	0	24000000	0 0
sdmmc0_bus	0	0	24000000	0 0
nand1	0	0	24000000	0 0
nand0	0	0	24000000	0 0
avs	0	0	24000000	0 0
apb2	1	1	24000000	0 0
scr0	0	0	24000000	0 0
twi4	0	0	24000000	0 0
twi3	0	0	24000000	0 0
twi2	0	0	24000000	0 0
twi1	0	0	24000000	0 0
twi0	0	0	24000000	0 0
uart5	0	0	24000000	0 0
uart4	0	0	24000000	0 0
uart3	0	0	24000000	0 0
uart2	0	0	24000000	0 0
uart1	0	0	24000000	0 0
uart0	1	1	24000000	0 0
hoscd2	0	0	12000000	0 0
pll_audiox4	9	9	90316800	0 0
ahub	3	3	90316800	0 0
codec_1x	1	1	45158400	0 0
spdif	1	1	90316800	0 0
pll_audiox2	0	0	45158400	0 0
pll_audio	5	5	22579200	0 0
codec_4x	0	0	22579200	0 0
dmic	0	0	22579200	0 0
pll_csi	0	0	432000000	0 0
pll_de	1	1	696000000	0 0
de	3	4	696000000	0 0
pll_ve	0	1	576000000	0 0
ve	0	1	576000000	0 0
pll_video2	2	2	594000000	0 0
tcon_tv	1	1	297000000	0 0
hdmi	1	1	297000000	0 0
pll_video2x4	0	0	2376000000	0 0
pll_video1	3	3	432000000	0 0
tve	1	1	216000000	0 0
tve_top	1	1	432000000	0 0
tcon_tv1	1	1	432000000	0 0
tcon_lcd1	0	0	432000000	0 0
pll_video1x4	0	0	1728000000	0 0
pll_video0x4	0	0	1188000000	0 0
tcon_lcd	0	0	1188000000	0 0
lvds	0	0	1188000000	0 0
pll_video0	0	0	297000000	0 0
csi_top	0	0	297000000	0 0
pll_gpu	0	0	306000000	0 0
gpu0	0	0	306000000	0 0
pll_periph1	2	2	600000000	0 0
hdmi_hdcp	1	1	300000000	0 0
pll_periph1x2	2	2	1200000000	0 0
sdmmc1_mod	1	1	100000000	0 0
sdmmc2_mod	1	1	200000000	0 0
pll_periph0	4	4	600000000	0 0
cpurapbs2_pll	0	0	600000000	0 0
cpurcpus_pll	0	0	600000000	0 0

apb1	3	3	100000000	0 0
pio	1	1	100000000	0 0
lradc	1	1	100000000	0 0
pwm	1	1	100000000	0 0
ahb3	9	9	200000000	0 0
display_top	2	3	200000000	0 0
usbotg	1	1	200000000	0 0
usb3_0_host	0	0	200000000	0 0
usbhci3	1	1	200000000	0 0
usbhci2	1	1	200000000	0 0
usbhci1	1	1	200000000	0 0
usbhci0	0	0	200000000	0 0
usbhci3	1	1	200000000	0 0
usbhci2	1	1	200000000	0 0
usbhci1	1	1	200000000	0 0
usbhci0	0	0	200000000	0 0
gmac1	1	1	200000000	0 0
gmac0	0	0	200000000	0 0
psi	1	1	200000000	0 0
ahb2	0	0	200000000	0 0
ahb1	2	2	200000000	0 0
iommu	1	1	200000000	0 0
dbgsys	0	0	200000000	0 0
hstimer	0	0	200000000	0 0
hwspinlock_bus	0	0	200000000	0 0
hwspinlock_rst	0	0	200000000	0 0
msgbox	0	0	200000000	0 0
dma	1	1	200000000	0 0
pll_periph0d6	0	0	100000000	0 0
periph32k	0	0	32768	0 0
pll_periph0x4	0	0	240000000	0 0
pll_periph0x2	2	2	120000000	0 0
ce	0	0	300000000	0 0
gpu1	1	1	600000000	0 0
di	1	1	300000000	0 0
g2d	0	0	300000000	0 0
mbus	0	0	400000000	0 0
pll_dds1	0	0	432000000	0 0
pll_dds0	0	0	1584000000	0 0
sdram	0	0	1584000000	0 0
pll_cpu	0	0	816000000	0 0
cpu	0	0	816000000	0 0
cpuapb	0	0	204000000	0 0
axi	0	0	272000000	0 0
iosc	0	0	160000000	0 0
losc	1	1	32768	0 0
hdmi_cec	1	1	32768	0 0

5.1.1.1 clk debugfs

利用 debugfs 提供的结点测试 clk 接口是否存在问题 (linux-5.4 暂不支持)，按以下步骤操作：

- 1. 在内核菜单项打开 clk debugfs 的配置，如下图所示：

在命令行中进入内核根目录 (kernel/linux-4.9)，执行 `make ARCH=arm64(arm) menucon-`

fig 进入配置主界面，并按以下步骤操作：首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

```
ration
Linux/arm64 4.9.170 Kernel Configuration
selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <
ded <M> module <> module capable

General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
Platform selection --->
Bus support --->
Kernel Features --->
Boot options --->
Userspace binary formats --->
Power management options --->
CPU Power Management --->
[*] Networking support --->
Device Drivers --->
Firmware Drivers --->
File systems --->
[ ] Virtualization ----
Kernel hacking --->
Security options --->
-* Cryptographic API --->
Library routines --->
```

图 5-1: 内核 menuconfig 根菜单

选择 Common Clock Framework，进入下级配置，如下图所示：

```

Device Drivers
Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <
] excluded <M> module < > module capable

<(-)
< > Serial ATA and Parallel ATA drivers (libata) ----
[ ] Multiple devices driver support (RAID and LVM) ----
< > Generic Target Core Mod (TCM) and ConfigFS Infrastructure ----
[*] Network device support --->
[ ] Open-Channel SSD target support ----
Input device support --->
Character devices --->
I2C support --->
[*] SPI support --->
< > SPMI support ----
< > HSI support ----
FPS support --->
PTP clock support --->
Pin controllers --->
[*] GPIO Support --->
< > Dallas's 1-wire support ----
[ ] Adaptive Voltage Scaling class support ----
-* Board level reset or power off --->
-* Power supply class support --->
<*> Hardware Monitoring support --->
< > Generic Thermal sysfs driver ----
[ ] Watchdog Timer Support ----
Sonnics Silicon Backplane --->
Broadcom specific AMBA --->
Multifunction device drivers --->
[*] Voltage and Current Regulator Support --->
<*> Multimedia support --->
Graphics support --->
<*> Sound card support --->
HID support --->
[*] USB support --->
< > Ultra Wideband devices ----
<*> MMC/SD/SDIO card support --->
< > Sony MemoryStick card support ----
[ ] LED Support ----
[ ] Accessibility support ----
[ ] EDAC (Error Detection And Correction) reporting ----
[*] Real Time Clock --->
[*] LMA Engine support --->
IMABUF options --->
[ ] Auxiliary Display support ----
< > Userspace I/O drivers ----
< > VFIO Non-Privileged userspace driver framework ----
[ ] Virtualization drivers ----
Virtio drivers --->
Microsoft Hyper-V guest support ----
[*] Staging drivers --->
[ ] Platform support for Goldfish virtual devices ----
[ ] Platform support for Chrome hardware ----
[*] Common Clock Framework --->
Hardware Spinlock drivers --->
Clock Source drivers --->

```

图 5-2: Common clock framework 菜单

选择 DebugFS representation of clock tree, 如下图所示:

```

Common Clock Framework
ts submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes featur
<M> module < > module capable

[*] DebugFS representation of clock tree
[ ] Clock driver for ARM Reference designs
< > Clock driver for SiLabs 5351A/B/C
< > Clock driver for SiLabs 514 devices
< > Clock driver for SiLabs 570 and compatible devices
< > Clock driver for TI CDCE706 clock synthesizer
< > Clock driver for TI CDCE925 devices
< > Clock driver for CS2000 Fractional-N Clock Synthesizer & Clock Multiplier
[ ] Clock driver for Freescale QorIQ platforms
[ ] Clock driver for APM XGene SoC
< > Clock driver for PWMs used as clock outputs
[ ] Clock support for Allwinner SoCs

```

图 5-3: clk DebugFS 菜单

- 2. 利用 clk debugfs 提供的结点测试

通过步骤 1 中在内核菜单项打开 CONFIG_COMMON_CLK_DEBUG 这个配置项后, 挂载上 debugfs, 可以看到 debugfs 目录下存在 ccudbg 目录, 则可以进行 debug 了, 如下所示:

```

mount -t debugfs none /sys/kernel/debug
console:/ # ls sys/kernel/debug/ccudbg
command info name param start

/*
 * debug clk_get_parent()接口
 */
echo getparent > sys/kernel/debug/ccudbg/command
echo cpuapb > sys/kernel/debug/ccudbg/name /*cpuapb从 clk_summary结点获取*/
echo 1 > sys/kernel/debug/ccudbg/start
cat sys/kernel/debug/ccudbg/info /*查看返回的父时钟*/
结果如下:
console:/ # cat sys/kernel/debug/ccudbg/info
cpu

/*
 * debug clk_set_rate()接口
 */
echo setrate > sys/kernel/debug/ccudbg/command
echo pll_csi > sys/kernel/debug/ccudbg/name /* pll_csi从 clk_summary结点获取 */
echo 600000000 > sys/kernel/debug/ccudbg/param /* 设置期望设置的频率 */
echo 1 > sys/kernel/debug/ccudbg/start

查看结果如下:
console:/ # cat sys/kernel/debug/ccudbg/info
600000000

console:/ # cat sys/kernel/debug/clk/clk_summary | grep "pll_csi"
    pll_csi                0          0 600000000          0 0

clk debugfs提供的常用测试命令如下所示:
getparents: 获取某个时钟的所有父时钟
getparent: 获取某个时钟当前的父时钟
setparent: 设置某个时钟的父时钟
getrate: 获取某个时钟的频率
setrate: 设置某个时钟的频率
is_enabled: 判断某个时钟是否enable
enable: 使能某个时钟
disable: 关闭某个时钟

```

5.1.1.2 利用 sunxi_dump 读写相应寄存器

```

cd /sys/class/sunxi_dump/
1. 查看一个寄存器, 如查看DE时钟寄存器, 根据spec, 看寄存器含义
echo 0x03001600 > dump ;cat dump

结果如下:
cupid-p1:/sys/class/sunxi_dump # echo 0x03001600 > dump ;cat dump
0x80000000

2. 写值到寄存器上, 如关闭DE时钟
echo 0x03001600 0x00000000 > write ;cat write

结果如下:
reg                to_write  after_write
0x0000000003001600  0x00000000  0x00000000

```

3. 查看一片连续寄存器

```
echo 0x03001000,0x03001fff > dump;cat dump
```

结果如下：

```
ccupid-pl:/sys/class/sunxi_dump # echo 0x03001000,0x03001fff > dump;cat dump
```

```
0x0000000003001000: 0x8a003a00 0x00000000 0x00000000 0x00000000
0x0000000003001010: 0xb8003900 0x00000000 0x08002301 0x00000000
0x0000000003001020: 0xb8003100 0x00000000 0x89003100 0x00000000
0x0000000003001030: 0x80003203 0x00000000 0x00000000 0x00000000
0x0000000003001040: 0x88006203 0x00000000 0x88004701 0x00000000
0x0000000003001050: 0x88006213 0x00000000 0x80001700 0x00000000
0x0000000003001060: 0x88001c00 0x00000000 0x00000000 0x00000000
0x0000000003001070: 0x00000000 0x00000000 0x89021501 0x00000000
0x0000000003001080: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001090: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030010a0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030010b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030010c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030010d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030010e0: 0x88002301 0x00000000 0x00000000 0x00000000
0x00000000030010f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001100: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001110: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001120: 0x00000000 0x00000000 0xd1303333 0x00000000
0x0000000003001130: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001140: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001150: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001160: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001170: 0x00000000 0x00000000 0xc001288d 0x00000000
0x0000000003001180: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001190: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030011a0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030011b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030011c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030011d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030011e0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030011f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001200: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001210: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001220: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001230: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001240: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001250: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001260: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001270: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001280: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001290: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030012a0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030012b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030012c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030012d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030012e0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030012f0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001300: 0x80100000 0x00000000 0x00000000 0x00000000
0x0000000003001310: 0x00030000 0x00000000 0x00030000 0x00000000
0x0000000003001320: 0x00030000 0x00000000 0x00030000 0x00000000
0x0000000003001330: 0x00030000 0x00000000 0x00000000 0x00000000
0x0000000003001340: 0x00030000 0x00000000 0x00030000 0x00000000
0x0000000003001350: 0x00030000 0x00000000 0x00030000 0x00000000
```

```
0x0000000003001360: 0x00030000 0x00000000 0x00000000 0x00000000
0x0000000003001370: 0x00000000 0x00000000 0x00030000 0x00000000
0x0000000003001380: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000003001390: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030013a0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030013b0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030013c0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030013d0: 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000030013e0: 0x00030000 0x00000000 0x00000000 0x00000000
0x00000000030013f0: 0x00000000 0x00000000 0x00000000 0x00000000
```

通过上述方式，可以查看，从而发现问题所在。






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。