Rockchip 时钟配置详细说明

发布版本：1.0

作者邮箱：zhangqing@rock-chips.com

日期：2018.6

文件密级：公开资料

前言

概述

产品版本

芯片名称	内核版本
RK3399	4.4

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
2018-06-08	V1.0	Elaine	第一次临时版本发布

---

1 时钟配置

1.1 CRU时钟配置

1.1.1 CRU时钟树

时钟树太长，不做说明，详细cat /sys/kernel/debug/clk/clk_summary

1.1.2 配置一些时钟常开

对于调试过程中，想把某些时钟设置成常开的，可以修改rk3399_cru_critical_clocks这个结构体,按照现有增加时钟名字即可：

```
drivers/clk/rockchip/clk-rk3399.c

    static const char *const rk3399_cru_critical_clocks[] __initconst = {
        "aclk_usb3_noc",
    };
```

这个结构中的clk在系统开机，clk初始化的时候会默认调用clk_set_enable接口。

注意：如果时钟不是常开的，驱动设备也没有引用这个时钟并开启，在时钟初始化完成之后，会调用clk_disable_unused_subtree（drivers/clk/clk.c）关闭没有用的时钟。如果没有用的时钟不想被关闭，可以在dts中的增加属性clk_ignore_unused：

```
    chosen {
     bootargs = "earlyprintk=uart8250-32bit,0xff690000 clk_ignore_unused";
    };
```

1.1.3 CLK ID获取

4.4的内核dts引用时钟，是根据clk id，不像3.10通过clk name索引。 CLK ID获取，详细见文档《Rockchip Clock 开发指南》中2.3.2章节。

1.1.4 PLL时钟配置

PLL锁相环详细介绍见文档《Rockchip Clock 开发指南》中1.3和2.2.1章节中。一般PLL不需要修改，尤其是下面挂了显示相关时钟的，PLL最好不要重现设置否则会有抖动问题。PLL的设置可以在UBOOT中，也可以直接在cru节点里面设置。

1. dts中设置 但是只在节点初始化的时候调用一次。

```
    cru: clock-controller@ff760000 {
        assigned-clocks =
            <&cru ARMCLKL>, <&cru ARMCLKB>,
            <&cru PLL_NPLL>, <&cru PLL_CPLL>,
            <&cru PLL_GPLL>;
        assigned-clock-rates =
            <816000000>,    <816000000>,
            <600000000>, <500000000>,
            <800000000>;
    };
```

ARMB PLL	ARML PLL	DDR PLL	GPLL	CPLL	VPLL	NPLL	USBPHYPLL	PMUPLL
1200	900	900	600	800	1200	1000	480	700

2. PLL计算公式 RK平台目前有两种类型PLL，一种是NR\NF\NO（RK3066、RK3188、RK3288），一种REF\POSTDIV1\POSTDIV2（RK3036、RK312X、RK322X、RK332X、RK336X、RK3399） 1)NR、NF、NO类型，只有整数分频 FREF = FIN / NR FOUTVCO = FREF * NF FOUTPOSTDIV = FOUTVCO / NO FREF范围： 269MHZ - 2200MHZ，VCO 范围: 440MHz – 2200MHz，输出频率范围：27.5MHz – 2200MHz。 NF_MAX：4096，NR_MAX：64，NO_MAX：16（只能偶数） 2)REF\POSTDIV1\POSTDIV2类型，支持整数分频和小数分频 整数分频： If DSMPD = 1 (DSM is disabled, "integer mode") FOUTVCO = FREF / REFDIV * FBDIV FOUTPOSTDIV = FOUTVCO / POSTDIV1 / POSTDIV2 小数分频： If DSMPD = 0 (DSM is enabled, "fractional mode") FOUTVCO = FREF / REFDIV * (FBDIV + FRAC / 2^24) FOUTPOSTDIV = FOUTVCO / POSTDIV1 / POSTDIV2 VCO 范围: 440MHz – 3200MHz，输出频率范围：27.5MHz – 2200MHz。 REFDIV_MAX：63，FBDIV_MAX：4095，POSTDIV1_MAX：7，POSTDIV1_MAX：7。

备注： 一般PLL是不需要做修改的，默认配置就可以，但是显示的时钟会对Jitter有要求，所以一般就是显示dclk独占的PLL有这种特殊的要求。可能会做一些微调。

PLL设置原则： PLL一般要求频率尽量在600-1200M，这样VCO比较合适（jitter会小）。目前的时钟频率基本都是支持自动计算的，但是自动计算只能保证VCO在上述范围内，不能保证VCO是最好的。所以如果对VCO有严格要求可以按照下面方式处理： PLL频率尽量设置大一些，然后通过后端分频。如：DCLK 50M，可以把CPLL设置成1000M，然后20分频。（这样Jitter肯定比把CPLL设置成50M或者100M的时候要好） 这种修改方式可以放在cru节点或者设备

节点中使用assigned-clock-rates的方式（本章节（1）），或者在驱动中，先设置PLL然后再设置dclk的方式。 可以在PLL频率表中增加所需要的频率，并且指定VCO。

以RK3399 4.4内核为例：

```
    static struct rockchip_pll_rate_table rk3399_vpll_rates[] = {
        /* _mhz, _refdiv, _fbdiv, _postdiv1, _postdiv2, _dsmpd, _frac */
        RK3036_PLL_RATE( 594000000, 1, 123, 5, 1, 0, 12582912),  /* vco = 2970000000 */
        RK3036_PLL_RATE( 593406593, 1, 123, 5, 1, 0, 10508804),  /* vco = 2967032965 */
        RK3036_PLL_RATE( 297000000, 1, 123, 5, 2, 0, 12582912),  /* vco = 2970000000 */
        RK3036_PLL_RATE( 296703297, 1, 123, 5, 2, 0, 10508807),  /* vco = 2967032970 */
        RK3036_PLL_RATE( 148500000, 1, 129, 7, 3, 0, 15728640),  /* vco = 3118500000 */
        RK3036_PLL_RATE( 148351648, 1, 123, 5, 4, 0, 10508800),  /* vco = 2967032960 */
        RK3036_PLL_RATE( 106500000, 1, 124, 7, 4, 0,  4194304),  /* vco = 2982000000 */
        RK3036_PLL_RATE(  74250000, 1, 129, 7, 6, 0, 15728640),  /* vco = 3118500000 */
        RK3036_PLL_RATE(  74175824, 1, 129, 7, 6, 0, 13550823),  /* vco = 3115384608 */
        RK3036_PLL_RATE(  65000000, 1, 113, 7, 6, 0, 12582912),  /* vco = 2730000000 */
        RK3036_PLL_RATE(  59340659, 1, 121, 7, 7, 0,  2581098),  /* vco = 2907692291 */
        RK3036_PLL_RATE(  54000000, 1, 110, 7, 7, 0,  4194304),  /* vco = 2646000000 */
        RK3036_PLL_RATE(  27000000, 1,  55, 7, 7, 0,  2097152),  /* vco = 1323000000 */
        RK3036_PLL_RATE(  26973027, 1,  55, 7, 7, 0,  1173232),  /* vco = 1321678323 */
        { /* sentinel */ },
    };
```

可以修改表中的频率对应的_refdiv, _fbdiv, _postdiv1, _postdiv2，以达到比较合适的VCO。 （对应3.10内核的处理方式请参考文档Rockchip Clock 开发指南中3.1章节）

1.1.5 CLK_TIMER时钟配置

Timer的时钟都是从24M直接经过gating bypass过来的。所以没有频率设置的概念，只有开关时钟的说法。如果想把clk配置成常开，请参考本文1.1.2. 如果驱动自己控制可以如下操作：

```
    struct clk *clk;
    clk = clk_get(NULL, "clk_timer00");
    clk_prepare_enable(clk);
```

1.1.6 总线时钟配置

总线时钟分为高速跟低速的， 高速时钟aclk_perihp、hclk_perihp、pclk_perihp，低速时钟是aclk_perilp0、hclk_perilp0、pclk_perilp0和hclk_perilp1、pclk_perilp1是可以配置时钟频率的，但是时钟树上这些时钟其下面的子时钟都是gating，只能开关，不能设置频率，如果希望修改频率，只能修改父时钟的频率。CCI做核间通信的总线时钟ACLK_CCI。DDR总线ACLK_CENTER。

频率设置方法

dts中设置(但是只在节点初始化的时候调用一次)

```
    cru: clock-controller@ff760000 {
        assigned-clocks =
            <&cru ACLK_PERIHP>, <&cru HCLK_PERIHP>,
            <&cru PCLK_PERIHP>,
            <&cru ACLK_PERILP0>, <&cru HCLK_PERILP0>,
            <&cru PCLK_PERILP0>,
            <&cru HCLK_PERILP1>, <&cru PCLK_PERILP1>,
            <&cru ACLK_CCI>, <&cru ACLK_CENTER>;
        assigned-clock-rates =
            <150000000>,   <75000000>,
            <37500000>,
            <100000000>,  <100000000>,
            <50000000>,
            <100000000>,   <50000000>,
            <40000000>, <40000000>;
    };
```

总线没有单独的驱动，也没有单独的节点，所以频率设置都是在cru节点中通过assigned的方式处理。

时钟频率范围

总线中aclk一般是用于数据传输，hclk跟pclk用于寄存器读写等。设计的频率：

CLK	SIZEOFF	CLK	SIZEOFF
ACLK_PERIHP	300M	ACLK_PERILP0	300M
HCLK_PERIHP	150M	HCLK_PERILP0	150M
PCLK_PERIHP	150M	PCLK_PERILP0	150M
HCLK_PERILP1	150M	PCLK_PERILP1	150M
CCI	600M	ACLK_CENTER	300M

如果超频需要考虑加压（logic路加压）

1.1.7 FCLK_CM0S时钟配置

fclk_cm0s是在cru而fclk_cm0s_src_pmu是在pmucru的，两个时钟设置是有差异的。Pmucru请看本章节中1.2.4.fclk_cm0s是可以配置时钟的，其下面的时钟都是gating(hclk_m0_perilp_noc\clk_m0_perilp_dec\dclk_m0_perilp\hclk_m0_perilp\sclk_m0_perilp)，只能开关，不能设置频率，如果希望修改频率，只能修改fclk_cm0s的频率。而且频率只能从GPLL或者CPLL分频下来。

频率设置方法

dts中设置，但是只在节点初始化的时候调用一次。

```
    cru: clock-controller@ff760000 {
        assigned-clocks = <&cru FCLK_CM0S>;
        assigned-clock-rates = <100000000>;
    };
```

可以放在cru节点，也可以放在设备的节点里面。 驱动中设置频率 总线一般没有单独的驱动，所以一般是在dts设置，如果部分驱动内部想调整这个频率也是可以的。

```
    struct clk *clk;
    clk = clk_get(NULL, "fclk_cm0s");
    clk_set_rate(clk, rate);/* rate单位hz */
```

时钟频率范围

M0设计的频率是100M，如果超频需要考虑加压（logic路加压）

1.1.8 CLK_I2C时钟配置

需要注意I2c1、2、3、5、6、7在cru模块中（I2C0、4、8在pmucru设置频率参考本章节1.2.5），3399的芯片i2c有两个时钟，一个控制时钟clk_i2c1一个配置时钟pclk_i2c1。控制时钟的频率只能从CPLL、GPLL分频，配置时钟频率是从pclk_perilp1来，自身只是gating不能修改频率，如果修改频率就要修改pclk_perilp1（见1.1.5）。

频率设置方法（以**i2c1**为例)

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    i2c1: i2c@ff110000 {
        clocks = <&cru SCLK_I2C1>, <&cru PCLK_I2C1>;
        clock-names = "i2c", "pclk";
        assigned-clocks = <&cru SCLK_I2C1>;
        assigned-clock-rates = <50000000>;
    };
```

可以放在cru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

```
    I2C的驱动文件drivers/i2c/busses/i2c-rk3x.c中：
    i2c->clk = devm_clk_get(&pdev->dev, "i2c");
    i2c->pclk = devm_clk_get(&pdev->dev, "pclk");
    clk_set_rate(i2c->clk, rate);/* rate单位hz */
```

时钟频率范围

一般使用控制时钟频率不超过100M，配置时钟不超过100M。如果超频需要考虑加压（logic路加压）

1.1.9 CLK_SPI时钟配置

需要注意spi0、1、2、4、5在cru模块中（spi3在pmucru设置频率参考本章节1.2.6），3399的芯片spi有两个时钟，一个控制时钟clk_spi0一个配置时钟pclk_spi0。控制时钟的频率只能从CPLL、GPLL分频，配置时钟频率是spi0、1、2、4是从pclk_perilp1，spi5从hclk_perilp1，自身只是gating不能修改频率，如果修改频率就要修改pclk_perilp1和hclk_perilp1（见1.1.5）。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    spi0: spi@ff1c0000 {
        clocks = <&cru SCLK_SPI0>, <&cru PCLK_SPI0>;
        clock-names = "spiclk", "apb_pclk";
        assigned-clocks = <&cru SCLK_SPI0>;
        assigned-clock-rates = <50000000>;
    };
```

可以放在cru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

SPI的驱动文件drivers/spi/spi-rockchip.c中：

```
    rs->apb_pclk = devm_clk_get(&pdev->dev, "apb_pclk");
    rs->spiclk = devm_clk_get(&pdev->dev, "spiclk");
    clk_set_rate(rs->spiclk, rate);/* rate单位hz */
```

时钟频率范围

一般使用控制时钟频率不超过50M，配置时钟不超过50M。如果超频需要考虑加压（logic路加压）

1.1.10 CLK_UART时钟配置

需要注意uart0、1、2、3在cru模块中（uart4在pmucru设置频率参考本章节1.2.8），3399的芯片uart有两个时钟，一个控制时钟clk_uart0一个配置时钟pclk_uart0。控制时钟支持小数分频和整数分频。clk_uart0_div由CPLL、GPLL、USB480M整数分频得到，clk_uart0_frac是由clk_uart0_div做输入时钟然后使用小数分频器分频的（小数分频需要注意输入时钟要是输出时钟的20倍以上，否则时钟Jitter很差）。具体使用时整数分频能满足走整数分频，整数分频不能满足走小数分频。配置时钟频率是从pclk_perilp1来，自身只是gating不能修改频率，如果修改频率就要修改pclk_perilp1（见1.1.5）。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    uart0: serial@ff180000 {
        clocks = <&cru SCLK_UART0>, <&cru PCLK_UART0>;
        clock-names = "baudclk", "apb_pclk";
        assigned-clocks = <&cru SCLK_UART0>;
        assigned-clock-rates = <24000000>;
    };
```

可以放在cru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

UART的驱动文件drivers/tty/serial/8250/8250_dw.c

```
    data->clk = devm_clk_get(&pdev->dev, "baudclk");
    data->pclk = devm_clk_get(&pdev->dev, "apb_pclk");
    clk_set_rate(data->clk, rate);/* rate单位hz */
```

时钟频率范围

这个主要看uart要求的波特率是多少，一般uart的频率是波特率 * 16（HZ），一般我们平台默认支持115200、1500000两种，其他波特率要具体看PLL的频率是否可以分到。

1.1.11 CLK_EMMC、CLK_SDIO、CLK_SDMMC时钟配置

这几个比较特殊，由于内部是双边沿采集数据，所以要求时钟的占空比是50%,也就要求必须是偶数分频。Emmc有两个时钟，clk_emmc是控制器时钟，要求偶数分频的。Aclk_emmc是数据传输和配置时钟。SDIO有有两个时钟，clk_sdio是控制器时钟，要求偶数分频的。hclk_sdio是配置时钟。SDMMC有有两个时钟，clk_sdmmc是控制器时钟，要求偶数分频的。hclk_sdmmc是配置时钟。控制时钟都是可以配置频率的，aclk_emmc、hclk_sdmmc也是可以单独配置频率，hclk_sdio是一个gating是从hclk_perilp1来，需要修改Hclk_sdio只能修改hclk_perilp1(见1.1.5).

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    sdhci: sdhci@fe330000 {
        assigned-clocks = <&cru SCLK_EMMC>;
        assigned-clock-rates = <200000000>;
        clocks = <&cru SCLK_EMMC>, <&cru ACLK_EMMC>;
        clock-names = "clk_xin", "clk_ahb";
    };
    sdio0: dwmmc@fe310000 {
        assigned-clocks = <&cru SCLK_SDIO>;
        assigned-clock-rates = <100000000>;
        clocks = <&cru HCLK_SDIO>, <&cru SCLK_SDIO>,
            <&cru SCLK_SDIO_DRV>, <&cru SCLK_SDIO_SAMPLE>;
        clock-names = "biu", "ciu", "ciu-drive", "ciu-sample";
    };
    sdmmc: dwmmc@fe320000 {
        assigned-clocks = <&cru SCLK_SDMMC>;
        assigned-clock-rates = <100000000>;
        clocks = <&cru HCLK_SDMMC>, <&cru SCLK_SDMMC>,
            <&cru SCLK_SDMMC_DRV>, <&cru SCLK_SDMMC_SAMPLE>;
        clock-names = "biu", "ciu", "ciu-drive", "ciu-sample";
    };
```

可以放在cru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

EMMC的驱动文件drivers/mmc/host/Sdhci-of-arasan.c

```
    sdhci_arasan->clk_ahb = devm_clk_get(&pdev->dev, "clk_ahb");
    clk_xin = devm_clk_get(&pdev->dev, "clk_xin");
    clk_set_rate(clk_xin, rate);/* rate单位hz */
```

时钟频率范围

时钟的频率范围如下，这个是输入频率，模块内部还有二分频，所以模块输出的实际频率是时钟树上看到频率的2分频。

CLK名称	**sizeoff频率**
EMMC	200M
ACLK_EMMC	100M（300M/b 64Bit)
SDMMC/SDIO	300/240M（<=300M)

注意　　对于频率设置需要说明，EMMC、SDIO、SDMMC的控制时钟的parent一般有CPLL、GPLL、NPLL、PPLL、UPLL。一般这些PLL中，CPLL被显示独占，如果EMMC需要200M频率，那么要求PLL频率是400M\800M\1200M,所以控制时钟能分到的频率要看PLL的频率是多少？一定是偶数分频得到的频率才可以（如果PLL只有600M和800M， 那么只能分出150\200\300\400M,实际在控制器输出频率只能那个有75、100、150、200M）。

1.1.12 显示相关VOP、HDCP、EDP跟ISP时钟配置

显示相关的时钟需求比较多，dclk一般要求任意频率，因为显示的分辨率不同dclk频率不同。而aclk跟Hclk做为数据传输和寄存器配置时钟一般是固定在一个值上，不会变化，一旦显示情况下修改aclk很hclk可能会造成显示抖动等。

DCLK　　如果要支持任意分辨率也就是要求DCLK可以出任意频率，一般这种情况DCLK要独占一个PLL，如果是双显也就是要两个PLL独立给DCLK使用（是否支持一般在芯片设计的时候就已经确定了，或者说支持双显后可以不需要支持其他特殊的功能）。RK3399的平台如果打开宏RK3399_TWO_PLL_FOR_VOP是支持双显任意频率，dclk_vop0独占CPLL，dclk_vop1独占VPLL。并且dclk_vop0和dclk_vop1都CLK_SET_RATE_PARENT的属性，dclk需要多少就会将对于的parent的Pll设置成多少。如果没有开启宏。只有dclk_vop0独占CPLL可以支持任意频率，dclk_vop1就是在当前parent可以就近分频。

```
#ifdef RK3399_TWO_PLL_FOR_VOP
    COMPOSITE(DCLK_VOP0_DIV, "dclk_vop0_div", mux_pll_src_vpll_cpll_gpll_p,
            CLK_SET_RATE_PARENT | CLK_SET_RATE_NO_REPARENT,
            RK3399_CLKSEL_CON(49), 8, 2, MFLAGS, 0, 8, DFLAGS,
            RK3399_CLKGATE_CON(10), 12, GFLAGS),
#else
    COMPOSITE(DCLK_VOP0_DIV, "dclk_vop0_div", mux_pll_src_vpll_cpll_gpll_p,
            CLK_SET_RATE_PARENT,
            RK3399_CLKSEL_CON(49), 8, 2, MFLAGS, 0, 8, DFLAGS,
            RK3399_CLKGATE_CON(10), 12, GFLAGS),
#endif

#ifdef RK3399_TWO_PLL_FOR_VOP
    COMPOSITE(DCLK_VOP1_DIV, "dclk_vop1_div", mux_pll_src_vpll_cpll_gpll_p,
            CLK_SET_RATE_PARENT | CLK_SET_RATE_NO_REPARENT,
```

```
            RK3399_CLKSEL_CON(50), 8, 2, MFLAGS, 0, 8, DFLAGS,
            RK3399_CLKGATE_CON(10), 13, GFLAGS),
#else
    COMPOSITE(DCLK_VOP1_DIV, "dclk_vop1_div", mux_pll_src_dmyvpll_cpll_gpll_p, 0,
            RK3399_CLKSEL_CON(50), 8, 2, MFLAGS, 0, 8, DFLAGS,
            RK3399_CLKGATE_CON(10), 13, GFLAGS),
#endif
```

ACLK、HCLK　　支持频率设置，但是如果有Uboot显示的时候，要求Uboot到kernel，频率不能发生变化，否则会闪屏。也就是要求uboot跟kernel的aclk hclk频率一致，其parent一致，parent频率一致。

频率设置方法

1. dts中设置

指定vop dclk的parent，因为rk3399硬件设计原因，要求给hdmi使用的那个vop一定要在vpll上（跟hdmi是同源的），另一个vop的parent是cpll。在各自的dts中，显示的节点中增加。

```
    &vopb_rk_fb {
        assigned-clocks = <&cru DCLK_VOP0_DIV>;
        assigned-clock-parents = <&cru PLL_VPLL>;
    };
    &vopl_rk_fb {
        assigned-clocks = <&cru DCLK_VOP1_DIV>;
        assigned-clock-parents = <&cru PLL_CPLL>;
    };
```

而ACLK跟HCLK只需要初始化一次：

```
    &vopb {
        clocks = <&cru ACLK_VOP0>, <&cru DCLK_VOP0>, <&cru HCLK_VOP0>, <&cru
        DCLK_VOP0_DIV>;
        clock-names = "aclk_vop", "dclk_vop", "hclk_vop", "dclk_source";
        assigned-clocks = <&cru ACLK_VOP0>, <&cru HCLK_VOP0>;
        assigned-clock-rates = <400000000>, <100000000>;
    };
```

可以放在cru节点，也可以放在设备的节点里面。HDCP、EDP跟ISP设置类似处理。

HDCP　　ACLK_HDCP 可以从CPLL、GPLL、PPLL分频得到、HCLK_HDCP和PCLK_HDCP是从ACLK_HDCP分频得到。都有独立的gating。

EDP　　只有PCLK_EDP，可以从CPLL、GPLL分频得到。其他的EDP时钟都是挂在这个下面的独立gating。

ISP　　ACLK_ISP0、ACLK_ISP1用于总线数据传输的可以从CPLL、GPLL、PPLL分频得到、HCLK_ISP0、HCLK_ISP1是寄存器配置时钟直接从ACLK分频、SCLK_ISP0、SCLK_ISP1是控制器时钟可以从CPLL、GPLL、NPLL分频得到。引用时钟的时候ACLK引用ACLK_ISP0_WRAPPER，HCLK引用HCLK_ISP0_WRAPPER,会自动打开上级的parent时钟保证ACLK跟HCLK时钟通路开启。

2. 驱动中设置频率

vop的驱动文件drivers/gpu/drm/rockchip/rockchip_drm_vop.c

```

```

```
    vop->hclk = devm_clk_get(vop->dev, "hclk_vop");
    if (IS_ERR(vop->hclk)) {
        dev_err(vop->dev, "failed to get hclk source\n");
        return PTR_ERR(vop->hclk);
    }
    vop->aclk = devm_clk_get(vop->dev, "aclk_vop");
    if (IS_ERR(vop->aclk)) {
        dev_err(vop->dev, "failed to get aclk source\n");
        return PTR_ERR(vop->aclk);
    }
    vop->dclk = devm_clk_get(vop->dev, "dclk_vop");
    if (IS_ERR(vop->dclk)) {
        dev_err(vop->dev, "failed to get dclk source\n");
        return PTR_ERR(vop->dclk);
    }
```

时钟频率范围

Dclk主要是看屏的分辨率是多少。因为PLL使用自动计算的，jitter不是最优，如果需要调整，可以在PLL表格中自己计算合适的VCO填进去(VCO的计算方式详细见RK3399 datasheet的2.6.2)。

```

Drivers/clk/rockchip/clk-rk3399.c

    static struct rockchip_pll_rate_table rk3399_vpll_rates[] = {
        /* _mhz, _refdiv, _fbdiv, _postdiv1, _postdiv2, _dsmpd, _frac */
        RK3036_PLL_RATE( 594000000, 1, 123, 5, 1, 0, 12582912),  /* vco = 2970000000
*/
        RK3036_PLL_RATE( 593406593, 1, 123, 5, 1, 0, 10508804),  /* vco = 2967032965
*/
        RK3036_PLL_RATE( 297000000, 1, 123, 5, 2, 0, 12582912),  /* vco = 2970000000
*/
        RK3036_PLL_RATE( 296703297, 1, 123, 5, 2, 0, 10508807),  /* vco = 2967032970
*/
        RK3036_PLL_RATE( 148500000, 1, 129, 7, 3, 0, 15728640),  /* vco = 3118500000
*/
        RK3036_PLL_RATE( 148351648, 1, 123, 5, 4, 0, 10508800),  /* vco = 2967032960
*/
        RK3036_PLL_RATE( 106500000, 1, 124, 7, 4, 0,  4194304),  /* vco = 2982000000
*/
        RK3036_PLL_RATE(  74250000, 1, 129, 7, 6, 0, 15728640),  /* vco = 3118500000
*/
        RK3036_PLL_RATE(  74175824, 1, 129, 7, 6, 0, 13550823),  /* vco = 3115384608
*/
        RK3036_PLL_RATE(  65000000, 1, 113, 7, 6, 0, 12582912),  /* vco = 2730000000
*/
        RK3036_PLL_RATE(  59340659, 1, 121, 7, 7, 0,  2581098),  /* vco = 2907692291
*/
        RK3036_PLL_RATE(  54000000, 1, 110, 7, 7, 0,  4194304),  /* vco = 2646000000
*/
        RK3036_PLL_RATE(  27000000, 1,  55, 7, 7, 0,  2097152),  /* vco = 1323000000
*/
```

```
        RK3036_PLL_RATE(  26973027, 1,  55, 7, 7, 0,  1173232),  /* vco = 1321678323
*/
        { /* sentinel */ },
    };
```

而对于对应的频率范围（如果做4K显示带宽不够可以相应提高ACLK的频率，但是要注意电压是不是够是否需要提压）：

CLK	SIZEOFF	CLK	SIZEOFF
ACLK_VOP0/1	400M	ACLK_VIO	400M
DCLK_VOP0	600M	ACLK_ISP0/1	400M
DCLK_VOP1	300M	CLK_ISP0/1	500M
CLK_VOP0/1_PWM	200M	CLK_EDP	200M
ACLK_HDCP	400M		

1.1.13 视频编解码VDU、RGA、CODEC、IEP相关时钟配置

主要是VDU、RGA、CODEC、IEP相关的时钟配置。

VDU ACLK_VDU可以从CPLL、GPLL、NPLL、PPLL分频得到，HCLK_VDU从ACLK分频得到。控制器时钟SCLK_VDU_CORE和SCLK_VDU_CA从CPLL、GPLL、NPLL分频得到。

RGA ACLK_RGA可以从CPLL、GPLL、NPLL、PPLL分频得到，HCLK_RGA从ACLK分频得到。

CODEC ACLK_VCODEC可以从CPLL、GPLL、NPLL、PPLL分频得到，HCLK_VCODEC从ACLK分频得到。控制器时钟SCLK_RGA_CORE从CPLL、GPLL、NPLL、PPLL分频得到。

IEP ACLK_IEP可以从CPLL、GPLL、NPLL、PPLL分频得到，HCLK_IEP从ACLK分频得到。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    rkvdec: rkvdec@ff660000 {
        clocks = <&cru ACLK_VDU>, <&cru HCLK_VDU>,
        <&cru SCLK_VDU_CA>, <&cru SCLK_VDU_CORE>;
        clock-names = "aclk_vcodec", "hclk_vcodec",
        "clk_cabac", "clk_core";
        assigned-clocks = <&cru ACLK_VDU>, <&cru HCLK_VDU>,
        <&cru SCLK_VDU_CA>, <&cru SCLK_VDU_CORE>;
        assigned-clock-rates = <400000000>, <400000000>,
        <300000000>, <300000000>;
    };
```

可以放在cru节点，也可以放在设备的节点里面(其他模块类似处理)。

2. 驱动中设置频率

VCODEC的驱动文件drivers/video/rockchip/vcodec/vcodec_service.c

```
    pservice->aclk_vcodec = devm_clk_get(dev, "aclk_vcodec");
    pservice->hclk_vcodec = devm_clk_get(dev, "hclk_vcodec");
    pservice->clk_cabac = devm_clk_get(dev, "clk_cabac");
    pservice->clk_core = devm_clk_get(dev, "clk_core");
```

时钟频率范围

而对于对应的频率范围（如果做4K视频编解码带宽不够可以相应提高ACLK的频率，但是要注意电压是不是够是否需要提压）：

CLK	SIZEOFF	CLK	SIZEOFF
ACLK_VCODEC	400M	ACLK_IEP	400M
ACLK_VDU	400M	ACLK_RGA	400M
CLK_VDU_CORE	300M	CLK_RGA_CORE	400M
CLK_VDU_CA	300M		

1.1.14 USB相关时钟配置

USB主要包括aclk、Host、otg还有就是usb内部phy。

USB3 ACLK_USB3可以从CPLL、GPLL、NPLL分频得到。

HOST HCLK_HOST0跟HCLK_HOST1都是挂在HCLK_PERI下面，只有GATING属性，如果要修改频率只能修改HCLK_PERI（详细见本文1.1.5）。

OTG ACLK_USB3OTG0、ACLK_USB3OTG1是挂在从ACLK_USB3下面，只有GATING属性，如果要修改频率只能修改ACLK_USB3。SCLK_USB3OTG0_REF直接从24M来的，只有GATING属性，SCLK_USB3OTG0_SUSPEND可以从24M或者32K上获取可以设置频率。

PHY SCLK_USBPHY1_480M是USB PHY内部PLL锁相环出的，一般是固定频率480M，可以选择内部锁相环输出可以直接选择24M输出。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    usb_host0_ohci: usb@fe3a0000 {
        clocks = <&cru HCLK_HOST0>, <&cru HCLK_HOST0_ARB>,
             <&cru SCLK_USBPHY0_480M_SRC>;
        clock-names = "hclk_host0", "hclk_host0_arb", "usbphy0_480m";
    };

    usb_host1_ehci: usb@fe3c0000 {
        clocks = <&cru HCLK_HOST1>, <&cru HCLK_HOST1_ARB>,
             <&cru SCLK_USBPHY1_480M_SRC>;
        clock-names = "hclk_host1", "hclk_host1_arb", "usbphy1_480m";
    };
    usbdrd3_0: usb@fe800000 {
        clocks = <&cru SCLK_USB3OTG0_REF>, <&cru SCLK_USB3OTG0_SUSPEND>,
             <&cru ACLK_USB3OTG0>, <&cru ACLK_USB3_GRF>;
```

```
        clock-names = "ref_clk", "suspend_clk",
                      "bus_clk", "grf_clk";
    };
```

可以放在cru节点，也可以放在设备的节点里面(其他模块类似处理)。

2. 驱动中设置频率

VCODEC的驱动文件drivers/usb/dwc3/dwc3-rockchip.c

```
    clk = of_clk_get(np, i);
    if (IS_ERR(clk)) {
        ret = PTR_ERR(clk);
        goto err0;
    }
    ret = clk_prepare_enable(clk);
    if (ret < 0) {
        clk_put(clk);
        goto err0;
    }
```

时钟频率范围

而对于对应的频率范围（如果USB有大数据拷贝等可以相应提高ACLK的频率，但是要注意电压是不是够是否需要提压）,ACLK_USB的SIZEOFF频率400M。

1.1.15 CIF相关时钟配置

Cif主要是SCLK_CIF_OUT，可能有24M或者27M这样的时钟要求。这个时钟源可以直接选择24M进行分频也可以选择CPLL、GPLL、NPLL然后再分频。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    cru: clock-controller@ff760000 {
        assigned-clocks = <&cru SCLK_CIF_OUT_SRC>, <&cru SCLK_CIF_OUT>;
        assigned-clock-rates = <800000000>, <27000000>;
    };
```

可以放在cru节点，也可以放在设备的节点里面(其他模块类似处理)。

1.2 PMUCRU时钟配置

1.2.1 PMUCRU时钟树

```
pll_ppll                          1     1    676000000    0 0
   ppll                           3     3    676000000    0 0
      pclk_pmu_src                3     3     48285715    0 0
         pclk_wdt_m0_pmu          0     0     48285715    0 0
         pclk_uart4_pmu           0     0     48285715    0 0
         pclk_mailbox_pmu         0     0     48285715    0 0
         pclk_timer_pmu           0     0     48285715    0 0
         pclk_spi3_pmu            0     0     48285715    0 0
         pclk_rkpwm_pmu           1     1     48285715    0 0
         pclk_i2c8_pmu            0     0     48285715    0 0
         pclk_i2c4_pmu            0     0     48285715    0 0
         pclk_i2c0_pmu            0     0     48285715    0 0
         pclk_noc_pmu             1     1     48285715    0 0
         pclk_sgrf_pmu            0     0     48285715    0 0
         pclk_gpio1_pmu           0     0     48285715    0 0
         pclk_gpio0_pmu           0     0     48285715    0 0
         pclk_intmem1_pmu         0     0     48285715    0 0
         pclk_pmugrf_pmu          0     0     48285715    0 0
         pclk_pmu                 0     0     48285715    0 0
      clk_i2c8_pmu                0     0     48285715    0 0
      clk_i2c4_pmu                0     0     48285715    0 0
      clk_i2c0_pmu                0     0     48285715    0 0
      clk_wifi_div                0     0     26000000    0 0
         clk_wifi_pmu             0     0     26000000    0 0
         clk_wifi_frac            0     0      1300000    0 0
      clk_spi3_pmu                0     0     96571429    0 0
      fclk_cm0s_pmu_ppll_src      1     1    676000000    0 0
         fclk_cm0s_src_pmu        2     2     84500000    0 0
            hclk_noc_pmu          1     1     84500000    0 0
            dclk_cm0s_pmu         0     0     84500000    0 0
            hclk_cm0s_pmu         0     0     84500000    0 0
            sclk_cm0s_pmu         0     0     84500000    0 0
            fclk_cm0s_pmu         0     0     84500000    0 0
```

注意　　上述时钟控制都是在pmucru寄存器。

1.2.2 配置一些时钟常开

对于调试过程中，想把某些时钟设置成常开的，

可以修改rk3399_pmucru_critical_clocks这个结构体中，按照现有增加时钟名字即可。

```
Drivers/clk/rockchip/clk-rk3399.c

  static const char *const rk3399_pmucru_critical_clocks[] __initconst = {
     "pclk_noc_pmu",
  };
```

这个结构中的clk在系统开机，clk初始化的时候会默认调用clk_set_enable接口。

1.2.3 PCLK_PMU总线时钟配置

总线时钟只有pclk_pmu_src是可以配置时钟的，其下面的时钟都是gating(pclk_wdt_m0_pmu\pclk_uart4_pmu\pclk_mailbox_pmu\pclk_timer_pmu\pclk_spi3_pmu\pclk_rkpwm_pmu\pclk_i2c8_pmu\pclk_i2c4_pmu\pclk_i2c0_pmu\pclk_noc_pmu\pclk_sgrf_pmu\pclk_gpio1_pmu\pclk_gpio0_pmu\pclk_intmem1_pmu\pclk_pmugrf_pmu\pclk_pmu)，只能开关，不能设置频率，如果希望修改频率，只能修改pclk_pmu_src的频率。而且频率只能从PPLL分频下来（676M整除出来的频率）。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    pmucru: pmu-clock-controller@ff750000 {
        assigned-clocks = <&pmucru PCLK_SRC_PMU>;
        assigned-clock-rates = <100000000>;
    };
```

可以放在pmucru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

总线一般没有单独的驱动，所以一般是在dts设置，如果部分驱动内部想调整这个频率也是可以的。

```
    struct clk *clk;
    clk = clk_get(NULL, "pclk_pmu_src");
    clk_set_rate(clk, rate);/* rate单位hz */
```

时钟频率范围

IC设计的频率是50M，如果超频需要考虑加压（logic路加压）

1.2.4 PMU_M0时钟配置

总线时钟只有PCLK_SRC_PMUfclk_cm0s_src_pmu是可以配置时钟的，其下面的时钟都是gating(hclk_noc_pmu\dclk_cm0s_pmu\hclk_cm0s_pmu\sclk_cm0s_pmu\fclk_cm0s_pmu)，只能开关，不能设置频率，如果希望修改频率，只能修改fclk_cm0s_src_pmu的频率。而且频率只能从PPLL或者24M分频下来。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    pmucru: pmu-clock-controller@ff750000 {
        assigned-clocks = <&pmucru FCLK_CM0S_SRC_PMU>;
        assigned-clock-rates = <100000000>;
    };
```

可以放在pmucru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

总线一般没有单独的驱动，所以一般是在dts设置，如果部分驱动内部想调整这个频率也是可以的。

```
    struct clk *clk;
    clk = clk_get(NULL, "fclk_cm0s_src_pmu");
    clk_set_rate(clk, rate);/* rate单位hz */
```

时钟频率范围

IC设计的频率是100M，如果超频需要考虑加压（logic路加压）

1.2.5 PMU总线时钟配置

总线时钟只有PCLK_SRC_PMU是可以配置时钟的，其下面的时钟都是gating(详细见时钟树)，只能开关，不能设置频率，如果希望修改频率，只能修改PCLK_SRC_PMU的频率。而且频率只能从PPLL分频下来。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    pmucru: pmu-clock-controller@ff750000 {
        assigned-clocks = <&pmucru PCLK_SRC_PMU>;
        assigned-clock-rates = <100000000>;
    };
```

可以放在pmucru节点，也可以放在设备的节点里面。

时钟频率范围

IC设计的频率是100M，如果超频需要考虑加压（logic路加压）

1.2.6 PMU_I2C时钟配置

需要注意I2c0\4\8在pmucru模块中，3399的芯片i2c有两个时钟，一个控制时钟clk_i2c0_pmu一个配置时钟pclk_i2c0_pmu。控制时钟的频率只能从PPLL（676M）分频，配置时钟频率是从pclk_pmu_src来，自身只是gating不能修改频率，如果修改频率就要修改pclk_pmu_src（见1.2.1）。

频率设置方法（以**i2c0**为例）

1.dts中设置，但是只在节点初始化的时候调用一次。

```
    i2c0: i2c@ff3c0000 {
        clocks =  <&pmucru SCLK_I2C0_PMU>, <&pmucru PCLK_I2C0_PMU>;
        clock-names = "i2c", "pclk";
        assigned-clocks = <&pmucru SCLK_I2C0_PMU>;
        assigned-clock-rates = <50000000>;
    }
```

可以放在pmucru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

I2C的驱动文件drivers/i2c/busses/i2c-rk3x.c中：

```
    i2c->clk = devm_clk_get(&pdev->dev, "i2c");
    i2c->pclk = devm_clk_get(&pdev->dev, "pclk");
    clk_set_rate(i2c->clk, rate);/* rate单位hz */
```

时钟频率范围

一般使用控制时钟频率不超过100M，配置时钟不超过100M。如果超频需要考虑加压（logic路加压）

1.2.7 PMU_SPI时钟配置

需要注意spi3在pmucru模块中，3399的芯片spi有两个时钟，一个控制时钟clk_spi3_pmu一个配置时钟pclk_spi3_pmu。控制时钟的频率只能从PPLL（676M）分频，配置时钟频率是从pclk_pmu_src来，自身只是gating不能修改频率，如果修改频率就要修改pclk_pmu_src（见1.2.1）。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    spi3: spi@ff350000 {
        clocks = <&pmucru SCLK_SPI3_PMU>, <&pmucru PCLK_SPI3_PMU>;
        clock-names = "spiclk", "apb_pclk";
        assigned-clocks = <&pmucru SCLK_SPI3_PMU>;
        assigned-clock-rates = <50000000>;
    };
```

可以放在pmucru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

SPI的驱动文件drivers/spi/spi-rockchip.c中：

```
    rs->apb_pclk = devm_clk_get(&pdev->dev, "apb_pclk");
    rs->spiclk = devm_clk_get(&pdev->dev, "spiclk");
    clk_set_rate(rs->spiclk, rate);/* rate单位hz */
```

时钟频率范围

一般使用控制时钟频率不超过50M，配置时钟不超过50M。如果超频需要考虑加压（logic路加压）

1.2.8 PMU_WIFI时钟配置

需要注意WIFI在pmucru模块中，3399的芯片wifi支持小数分频和整数分频。clk_wifi_div由PPLL整数分频得到，clk_wifi_frac是由clk_wifi_div做输入时钟然后使用小数分频器分频的（小数分频需要注意输入时钟要是输出时钟的20倍以上，否则时钟Jitter很差）。具体使用时整数分频能满足走整数分频，整数分频不能满足走小数分频。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    pmucru: pmu-clock-controller@ff750000 {
        compatible = "rockchip,rk3399-pmucru";
        assigned-clocks = <&pmucru SCLK_WIFI_PMU>;
        assigned-clock-rates = <26000000>;
    };
```

可以放在pmucru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

```
    struct clk *clk;
    clk = clk_get(NULL, "clk_wifi_pmu");
    clk_set_rate(clk, rate);/* rate单位hz */
```

时钟频率范围

这个主要看wifi模组使用是什么晶振，一般常见的24M 26M 37.4M 40M。

1.2.9 PMU_UART4时钟配置

需要注意uart4在pmucru模块中，3399的芯片uarti有两个时钟，一个控制时钟clk_uart4_pmu一个配置时钟pclk_uart4_pmu。控制时钟支持小数分频和整数分频。clk_uart4_div由PPLL整数分频得到，clk_uart4_frac是由clk_uart4_div做输入时钟然后使用小数分频器分频的（小数分频需要注意输入时钟要是输出时钟的20倍以上，否则时钟Jitter很差）。具体使用时整数分频能满足走整数分频，整数分频不能满足走小数分频。配置时钟频率是从pclk_uart4_pmu来，自身只是gating不能修改频率，如果修改频率就要修改pclk_pmu_src（见1.2.1）。

频率设置方法

1. dts中设置，但是只在节点初始化的时候调用一次。

```
    uart4: serial@ff370000 {
        clocks = <&pmucru SCLK_UART4_PMU>, <&pmucru PCLK_UART4_PMU>;
        clock-names = "baudclk", "apb_pclk";
        assigned-clocks = <&pmucru SCLK_UART4_PMU>;
        assigned-clock-rates = <24000000>;
    };
```

可以放在pmucru节点，也可以放在设备的节点里面。

2. 驱动中设置频率

UART的驱动文件drivers/tty/serial/8250/8250_dw.c

```
    data->clk = devm_clk_get(&pdev->dev, "baudclk");
    data->pclk = devm_clk_get(&pdev->dev, "apb_pclk");
    clk_set_rate(data->clk, rate);/* rate单位hz */
```

时钟频率范围

这个主要看uart要求的波特率是多少，一般uart的频率是波特率 * 16（HZ），一般我们平台默认支持115200、1500000两种，其他波特率要具体看PLL的频率是否可以分到。

2 时钟间依赖关系

2.1 普通的父子关系

时钟结构图及时钟树如下：

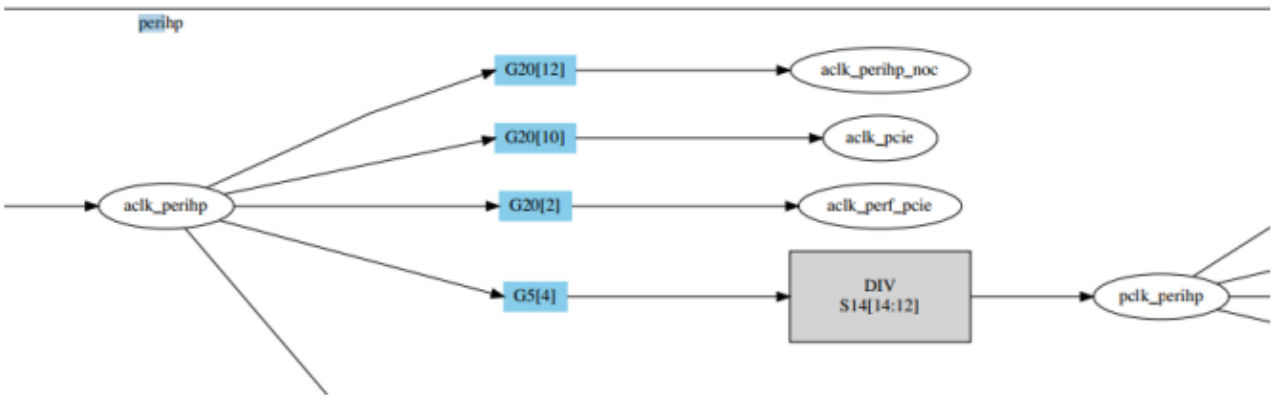

图 2-1 部分时钟结构图

时钟树的结构就是：

```
   clock                         enable_cnt  prepare_cnt        rate   accuracy   phase
----------------------------------------------------------------------------------------
      aclk_perihp                         4            5   133333334          0 0
         aclk_perihp_noc                  1            1   133333334          0 0
         aclk_pcie                        0            0   133333334          0 0
         aclk_perf_pcie                   0            0   133333334          0 0
         pclk_perihp                      3            3    33333334          0 0
            pclk_hsicphy                  0            0    33333334          0 0
            pclk_perihp_noc               1            1    33333334          0 0
            pclk_pcie                     0            0    33333334          0 0
            pclk_perihp_grf               1            1    33333334          0 0
```

普通的父子关系的依赖关系就是，子时钟开启的时候需要开启父时钟，时钟结构会保证此操作，只需要开启子时钟即可，时钟结构会自动索引其父时钟并开启。只要其子时钟有在工作，父时钟就不能关闭，正常情况时钟的开关是有引用计数，如上图中的enable_cnt，子时钟或者本身时钟被enable后计数加一，disable的时候计数减一，直到计数减为零，时钟才会被关闭。

2.2 不同模块间NOC复用

在设计NOC的时候，有一些模块之间的Noc是复用，这就要求任何一个模块在使用的时候，NOC时钟都要开启，而且NOC的父时钟的整个时钟通路都要开启。 有这种特殊要求的有如下时钟（目前代码中都已经处理，保证NOC时钟常开）：

表 2-1 NOC 依赖关系表 1

cci_noc	aclk_cci_noc0:G15[3]		
	aclk_cci_noc1:G15[4]		
	clk_dbg_noc:G15[5]		
peri_lp_noc	alive_noc（详细见下表格）		
	pmu_noc		
	emmc_noc		
	gmac_noc		
	usb3_noc		
	gic_noc		
	sdioaudio_noc		
	center_noc	msch0_noc	
		msch1_noc	
		iep_noc	
		rga_noc	
		perihp_noc	sd_noc
		vdu_noc	
		vcodec_noc	
		gpu_noc	
		edp_noc	
		vio_noc	isp0_noc
			isp1_noc
			hdcp_noc
			vopb_noc
			vopl_noc

表 2-2 NOC 依赖关系表 2

alive_noc	aclk_perilp0_noc:G25[7] hclk_perilp0_noc:G25[8] hclk_perilp1_noc:G25[9] pclk_perilp1_noc:G25[10]	alive_noc:PMUGRF0[6]	
		pclk_noc_pmu:PG1[6] hclk_noc_pmu:PG1[5]	
		aclk_emmc_noc:G32[9]	
		aclk_gmac_noc:G32[1] pclk_gmac_nocG32[3]	
		aclk_usb3_noc:G30[0]	
		aclk_gic_noc:G33[1]	
		sdioaudio_noc:G34[6]	
		aclk_center_main_noc:G19[0] aclk_center_peri_noc:G19[1] pclk_center_main_noc:G18[10]	msch0_noc:G18[4]
			msch1_noc:G18[9]
			aclk_iep_noc:G16[1] hclk_iep_noc:G16[3]
			aclk_rga_noc:G16[9] hclk_rga_noc:G16[11]
			aclk_perihp_noc:G20[12] hclk_perihp_noc:G20[13] pclk_perihp_noc:G20[14]
			aclk_vdu_noc:G17[9] hclk_vdu_noc:G17[11]
			aclk_vcodec_noc:G17[1] hclk_vcodec_noc:G17[3]
			gpu_noc
			pclk_edp_noc:G32[12]

2.3 不同模块间GRF复用

在设计GRF的时候，有一些模块之间的GRF时钟是复用，这就要求任何一个模块在GRF寄存器读写的时候，公用的GRF时钟都要开启，而且GRF的父时钟的整个时钟通路都要开启。 有这种特殊要求的有如下时钟（目前代码中都已经处理，保证GRF时钟常开）：

GRF	CON	GRF	CON
aclk_cci_grf	grf_a72_perf	pclk_grf(alive)	grf_iomux
aclk_cci_grf	grf_a53_perf	pclk_grf(alive)	grf_soc_con[0~8]
aclk_cci_grf	grf_cpu_status	pclk_grf(alive)	grf_usb3phy
aclk_cci_grf	grf_cpu_con	pclk_grf(alive)	grf_ddrc
		pclk_grf(alive)	grf_gpio2/3/4
pclk_perihp_grf	grf_hsic	pclk_grf(alive)	grf_io_vsel
pclk_perihp_grf	grf_hsicphy	pclk_grf(alive)	grf_saradc
pclk_perihp_grf	grf_usbhost0	pclk_grf(alive)	grf_tsadc
pclk_perihp_grf	grf_usbhost1	pclk_grf(alive)	grf_usb20_phy
pclk_perihp_grf	grf_usbphy	pclk_grf(alive)	grf_dll
pclk_perihp_grf	grf_usb20_host	pclk_grf(alive)	grf_alive_lf_ena
pclk_perihp_grf	grf_pcie	pclk_grf(alive)	grf_cphy
		pclk_grf(alive)	grf_uphy
pclk_vio_grf	grf_soc_con[9, 20~26]	pclk_grf(alive)	grf_pcie
pclk_vio_grf	grf_hdcp	pclk_grf(alive)	grf_sd
aclk_usb3_grf	grf_sta_usb3otg	aclk_gpu_grf	grf_gpu_perf
aclk_usb3_grf	grf_usb3_perf		

3 时钟频率值

3.1 可设置的时钟频率

时钟名称	最高频率	可以设置频率
CCI	600M	(cpll/gpll/npll/vpll) / (1~32)
ACLK_CENTER	300M	(cpll/gpll/npll) / (1~32)
DDR_PCLK	200M	(cpll/gpll) / (1~32)
ACLK_PERIHP	300M	(cpll/gpll) / (1~32)
ACLK_PERILP0	300M	(cpll/gpll) / (1~32)
AHB_PERILP1	150M	(cpll/gpll) / (1~32)
ACLK_VCODEC	400M	(cpll/gpll/npll/ppll) / (1~32)
ACLK_VDU	400M	(cpll/gpll/npll/ppll) / (1~32)
CLK_VDU_CORE	300M	(cpll/gpll/npll) / (1~32)
CLK_VDU_CA	300M	(cpll/gpll/npll) / (1~32)
ACLK_IEP	400M	(cpll/gpll/npll/ppll) / (1~32)
ACLK_RGA	400M	(cpll/gpll/npll/ppll) / (1~32)
CLK_RGA_CORE	400M	(cpll/gpll/npll/ppll) / (1~32)
EMMC	200M	(cpll/gpll/npll/ppll/upll/24M) / (1~128)
SDMMC/SDIO	400/300/240M	(cpll/gpll/npll/ppll/upll/24M) / (1~128)
CLK_PCIE_REF	24/100M	(npll/24M) / (1~128)
CLK_PCIE_CORE	250M	(cpll/gpll/npll) / (1~128)
CLK_PCIE_PM	25M/24M	(cpll/gpll/npll/24M) / (1~128)
ACLK_GMAC	300M	(cpll/gpll) / (1~32)
ACLK_EMMC	300M	(cpll/gpll) / (1~32)
M0-PERILP	300M	(cpll/gpll) / (1~32)
CRYPTO	200M	(cpll/gpll/ppll) / (1~32)
SPI	100M	(cpll/gpll) / (1~128)
I2S/SPDIF	Frac<=50M	(cpll/gpll) / (1~128)
SARADC	20M	24M/ (1~256)
TSADC	10M	24M/ (1~1024)
ACLK_VIO	400M	(cpll/gpll/ppll) / (1~32)
CLK_EDP	200M	(cpll/gpll) / (1~32)

时钟名称	最高频率	可以设置频率
ACLK_ISP0/1	400M	(cpll/gpll/ppll) / (1~32)
CLK_ISP0/1_JPE	500M	(cpll/gpll/npll) / (1~32)
ACLK_HDCP	400M	(cpll/gpll/ppll) / (1~32)
CLK_DP_CORE	200/100/10M	(cpll/gpll/npll) / (1~32)
ACLK_VOP0/1	400M	(cpll/gpll/npll/vpll) / (1~32)
DCLK_VOP0	600M	(cpll/gpll/vpll) / (1~256)
DCLK_VOP1	300M	(cpll/gpll/vpll) / (1~256)
CLK_VOP0/1_PWM	200M	(cpll/gpll/vpll/24M) / (1~32)
ACLK_USB3	300M	(cpll/gpll/npll) / (1~32)
PCLK_ALIVE	100M	(gpll) / (1~32)
PCLK_PMU	100M	(ppll) / (1~32)
GPU	500M	(cpll/gpll/npll/upll/ppll) / (1~32)
M0-EC	200M	(cpll/gpll) / (1~32)