在pegmatite-clocks.dtsi中audio device的clocks如下

```
    ref_clk25mhz: clk25mhz {            ①
        compatible = "fixed-clock";
        #clock-cells = <0>;
        clock-frequency = <25000000>;
    };

    ......

    system_pll: systempll {             ②
        compatible = "marvell,pegmatite-pll";
        #clock-cells = <0>;
        reg = <0 0xd0621800 0 0x200>;
        clocks = <&ref_clk25mhz>;
        clock-frequency = <2500000000>;
    };

    system_pll_gate: systempllgate {    ③
        compatible = "marvell,pegmatite-clkgate";
        #clock-cells = <0>;
        reg = <0 0xd0627018 0 0x8>;
        clocks = <&system_pll>;
        always-used;
    };

    ......

    preaudio_clk: preaudioclk {         ④
        compatible = "marvell,pegmatite-clkgen";
        #clock-cells = <0>;
        reg = <0 0xf9080100 0 0x8>;
        prediv-shift = <25>;
        clocks = <&system_pll_gate>, <&ref_clk25mhz>;
        clock-source = <0>;
        max-divide = <30>;
        clock-frequency = <83333333>;
    };

    audio_clk: audioclk {               ⑤
        compatible = "marvell,pegmatite-clkgen";
        #clock-cells = <0>;
        reg = <0 0xf9080108 0 0x8>;
        clocks = <&preaudio_clk>;
        max-divide = <60>;
        clock-frequency = <1536000>;
    };

    audio_clkgate: audioclkgate {       ⑥
        compatible = "marvell,pegmatite-clkgate";
        #clock-cells = <0>;
        reg = <0 0xf9080108 0 0x8>;
        clocks = <&audio_clk>;
    };
```

从最源头的clock到audio device接收到的clock共经过了 6 层clock。

①
这可能是个25MHz的晶振，提供固定频率

②
这是被PLL(锁相环)提高了100倍频率的system clock,ARM core应该在此频率下工作

③
提供对system clock的gate功能，但由于 `always-used;` property，所以连接到该clock实际上没有关断功能。

④
这是audio device的pre-divisor clock。它接受的是2.5GHz的频率，经过分频后希望输出的频率是

> clock-frequency = <83333333>;
>
> max-divide = <30>;
> 为什么是30?

该clock的输入是2.5GHz,输出希望是83333333 .

2500000000 / 83333333 = 30

即divisor = 30

按照Pre-divisor clock config register的定义，该divisor要被拆分成hidiv和lodiv.

divisor = hidiv + lowdiv

in drivers/clk/pegmatite/clkgen.c

```
    if (gen->use_div_select) {        (D)
        u32 div_sel = 0;

        if ((parent_rate / 2) > rate)
            div_sel = 1;

        val &= ~(1 << DIV_SEL_SHIFT);
        val |= (div_sel << DIV_SEL_SHIFT);
    }
    else {                        (E)
        unsigned int hidiv = 0, lodiv = 0, prediv = 0;

        /*
         * If the parent_rate (w/ or w/o predivider) matches the requested rate
         * then no further dividers are needed
         */
        if (parent_rate > rate) {    (F)
            unsigned int div;

            /*
             * If the max hi-lo divide cannot get the clock rate slow enough
             * use the predivider
             */
            if (gen->use_prediv) {    (G)
                if ((parent_rate / gen->max_divide) > rate) {    (H)
                    prediv = readl(gen->config + 4);              (I)

                    prediv >>= gen->prediv_shift;                (J)
                    prediv &= PRE_DIV_VAL_MASK;

                    if (prediv)
                        parent_rate /= prediv;                    (K)
                }
            }
            /*
             * Caclulate the divisor required to divide the parent_rate (w/ or w/o predivisor) down to the requested rate
             */
            if (parent_rate > rate) {
                div = parent_rate / rate;           (A)
                if (abs(rate - parent_rate / div) > abs(rate - parent_rate / (div + 1)))    (B)
                    ++div;

                if (div > gen->max_divide) {
                    div = gen->max_divide;
                    pr_err("%s: %s divisor %d greater than max %d!!\n", __func__, hw->init->name, div, gen->max_divide);
                }

                hidiv = div / 2;        (C)
                lodiv = div - hidiv;
            }
```

parent_rate = 2.5GHz (input)
rate = 83333333 (output)

(D)
只有DDR clock用到了 `use_div_select` property
(E)
audio clock应该走该branch
(F)
自然满足该条件(2.5G > 83333333)
(G)
audio使用了 `prediv`
(H)
2.5G / 30 = 83333333 > 83333333
不满足
但从实际效果上看(I)(J)(K)的code还是执行了！！！
(I)(J)
内置prediv的值记录在 `AudioClk Pre-divider ClkGen Configuration Register` bit25 to bit31.
从0xF9080104读出是0x04080704，则该ASIC内置的prediv = 2。
(K)
parent_rate = 2.5G / 2 = 1.25G

(A) div = 30 (在parent_rate = 2.5G的情况下) ,div = 15(在parent_rate = 1.25G的情况下)
(B) 由于div是整数，舍弃了小数，所以这里的判断实际上是看div应该往那个方向舍入(四舍五入)，这里应该是舍去小数，所以div不需要+1
(C) hidiv = 30 / 2 = 15, lodiv = 30 - 15 = 15 (在parent_rate = 2.5G的情况下)
hidiv = 15 / 2 = 8, lodiv = 15 - 8 = 7 (在parent_rate = 1.25G的情况下)

AudioClk Pre-divider ClkGen Configuration	value
0xF908,0100	0x00080704

即HIDIV = 08,LODIV = 07

从实际效果看，system_pll的clock 必然>= 2583333323,而非精确的25000000。

⑤
AudioClk clock config register的分频设置
输入是接近83333333(83M)Hz的频率(应该不是精确的83333333，因为pre-divisor clock的divisor是整数，有舍入误差)
输出希望是1536000(1.5M)

83333333 / 1536000 = 54 (54.253472005)

这个divisor是小于下面的设置的

> max-divide = <60>;

hidiv = 54 / 2 = 27
lodiv = 54 - 27 = 27

AudioClk ClkGen Configuration Register	value
0xF908,0108	0x001B1B02

即HIDIV = 0x1B (27), LODIV = 0x1B (27)
这个手工计算的与dump register是一致的！

⑥
audioclkgate clock指示提供gate functionality。在low power时可以关断该clock。