

i.MX6Q Plugin 启动方式典型应用： PLL2 改频与展频

by John Li (R64710/nxa08200)
GSM CAS
恩智浦半导体有限公司
上海,

i.MX6Q 支持两种启动方式，DCD和Plugin。用于在正式启动前，由Rom代码来初始化某些寄存器，如初始化外存DDR配置寄存器。我们BSP默认使用DCD启动方式，但是在新的uboot代码中，也支持Plugin启动模式，只需要打开Plugin模式编译宏。本文使用i.MX6Q Uboot 源代码，版本 2015_04_r0(匹配4.1.15_1.0.0 内核代码)，来使用Plugin启动。使用Plugin启动有两个典型应用如下：

- 改变PLL2 时钟频率.
- 使用PLL2 展频功能(spread spectrum)

本文会详细说明为什么使用Plugin启动，以及具体的应用场景。

本文默认使用i.MX6Q SDP板测试，关于i.MX6DL的情况，在第7章进行了描述。

本文及相关源代码可以在以下连接下载：

<https://community.nxp.com/docs/DOC-33423>

1/

History	
V1	● 创建本文
V2	● 加入 EMC 测试结果(测试数据来源于 Lambert.Zhang)
V3	● 加入第 7 章 i.MX6DL 支持

V4	● 翻译为中文版本
V5	● 内部版本
V6	● 增加第 8 章：3.0.35 uboot/bsp 支持
V7	● 改频增加北斗频段说明

目录

1	获取测试用uboot源代码	3
2	测试Plugin启动并iomux PLL2到CLKO1	6
3	将PLL2改为520MHz.....	10
4	展频(spread spectrum)	17
5	Uboot源代码修改总结	24
6	修改内核代码来支持520Mhz PLL2	29
7	i.MX6DL支持说明	36
8	3.0.35 uboot/bsp支持	41

i.MX6Q Plugin启动方式典型应用：改频与展频

1 获取测试用 uboot 源代码

根据恩智浦<<Freescale_Yocto_Project_User's_Guide.pdf>> Yocto 编译环境搭建文档，用于下载 uboot 代码和搭建编译环境，相关文档下载地址为：www.nxp.com/imx->i.MX6
Series->i.MX6Quad->DOCUMENTATION->Supporting Information

- [L4.1.15_1.0.0_LINUX_DOCS](#)

具体步骤如下：

1. 搭建 PC 交叉编译环境(测试使用版本为 ubuntu 14.04 64bit PC)。

//在用户目录下操作，测试时使用/home/vmuser/imx6x

```
sudo apt-get update
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat libssl1.2-dev
sudo apt-get install libssl1.2-dev xterm sed cvs subversion coreutils texi2html docbook-utils python-pysqlite2 help2man make
gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev mercurial autoconf automake groff curl lzop asciidoc
sudo apt-get install u-boot-tools
```

2. 获得 BSP 源代码

```
mkdir bin
```

//repo 工具无法从 google 网站下载，可尝试使用以下连接：

```
curl http://raw.githubusercontent.com/android/tools_repo/stable/repo > ./bin/repo
chmod a+x ./bin/repo
export PATH=~/imx6x/bin/:$PATH
git config --global user.name "xxxx"
git config --global user.email "xxxx@xxxx.com"
git config --global --list
```

```
user.name=xxxx@xxxx.com
user.email=vmuser@vmuser.com
mkdir fsl-release-bsp
cd fsl-release-bsp/
```

// repo 工具无法使用 google 网站，可尝试使用以下连接：

```
repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1.15-1.0.0_ga
--repo-url=http://github.com/android/tools_repo
repo sync
```

3. 编译

i.MX6Q Plugin启动方式典型应用：改频与展频

```
DISTRO=fsl-imx-fb MACHINE=imx6qsabresd source fsl-setup-release.sh -b fsl-imx-fb
```

```
bitbake fsl-image-machine-test
```

//根据编译 PC 配置不同，需要一定的时间完成

将整个镜像烧录到测试使用的 SD 卡命令如下：

```
pwd
```

```
~/imx6x /fsl-release-bsp/fsl-imx-fb/tmp/deploy/images/imx6qsabresd
```

```
cat /proc/partitions
```

```
major minor #blocks name
```

```
...
```

```
8 32 7761920 sdc
```

```
...
```

```
sudo dd if=fsl-image-machine-test-imx6qsabresd.sdcards of=/dev/sdc bs=1M && sync
```

可能从 fsl-imx-fb 目录的以下位置找到 uboot 源代码

```
~/imx6x/fsl-release-bsp/fsl-imx-fb/tmp/work/imx6dsabresd-poky-linux-gnueabi/u-boot-imx/2015.04-r0/git$
```

4. 编译及安装 SDK

```
bitbake fsl-image-machine-test -c populate_sdk
```

//根据编译 PC 配置不同，需要一定的时间完成

```
cd tmp/deploy/sdk/
```

```
./fsl-imx-fb-glibc-x86_64-fsl-image-machine-test-cortexa7hf-vfp-neon-toolchain-4.1.15-1.1.1.sh
```

//键入”enter” 和 “Y”，选择安装目录

5. 编译 uboot

使用 Yocto bitbake 命令编译

```
pwd
```

```
/home/vmuser/imx6x /fsl-release-bsp/fsl-imx-fb
```

```
bitbake u-boot-imx -c listtasks
```

```
bitbake u-boot-imx -c compile -f
```

或者使用 standalone 直接命令编译

```
pwd
```

```
/home/vmuser/imx6x /fsl-release-bsp/fsl-imx-fb/tmp/work/imx6qsabresd-poky-linux-gnueabi/u-boot-imx/2015.04-r0/git
```

```
export ARCH=arm
```

```
export
```

```
CROSS_COMPILE=/opt/fsl-imx-fb/4.1.15-1.1.1/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-
```

i.MX6Q Plugin启动方式典型应用：改频与展频

```
export
CC="/opt/fsl-imx-fb/4.1.15-1.1.1/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g
cc --sysroot=/opt/fsl-imx-fb/4.1.15-1.1.1/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi"
make distclean
make clean
make mx6qsabresd_config
make CC="$CC" u-boot.imx
or
source /opt/fsl-imx-fb/4.1.15-1.1.1/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi
make clean
make mx6qsabresd_config
make
```

将 uboot 烧录到测试使用的 SD 卡命令如下：

```
cat /proc/partitions
major minor #blocks name
...
8      16    7761920 sdb
...
sudo dd if=u-boot.imx of=/dev/sdb bs=512 seek=2 conv=fsync
```

编译内核与 DTB 以及烧录到 SD 卡的命令如下：

```
pwd
/home/vmuser/imx6x/fsl-release-bsp/fsl-imx-fb-imx6qsabresd/tmp/work/imx6qsabresd-poky-linux-gnueabi/linux-imx/4.1.15
-r0/git
make distclean
make clean
make imx_v7_defconfig
make zImage
make imx6q-sabresd.dtb
sudo cp arch/arm/boot/zImage /media/vmuser/Boot\ imx6qs/
sudo cp arch/arm/boot/dts/imx6q-sabresd.dtb /media/vmuser/Boot\ imx6qs/
```

i.MX6Q Plugin启动方式典型应用：改频与展频

2 测试 Plugin 启动并 iomux PLL2 到 CLKO1

2015_04_r0(匹配 4.1.15_1.0.0 内核) uboot, 会直接编译出 u-boot.bin 镜像, 然后使用 imximage 工具加上 DCD 头或是 Plugin 头, 生成 u-boot.imx 正式镜像。

修改如下代码可以编译为 Plugin 启动 include\configs\mx6sabre_common.h:

```
/* uncomment for PLUGIN mode support */  
/* #define CONFIG_USE_PLUGIN */  
  
#define CONFIG_USE_PLUGIN //johnli open the plug in support.
```

如果使用 Plugin 启动模式, 将使用汇编语言来初始化 DDR 配置寄存器, 而不是使用 DCD 数据模式, 相关代码在: board\freescale\mx6sabresd\plugin.s

```
.macro imx6_ddr_setting  
....  
#elif defined(CONFIG_MX6Q)  
imx6dqsabresd_ddr_setting  
#else  
#error "SOC not configured"  
#endif  
.endm  
  
.macro imx6dqsabresd_ddr_setting  
... //the DDR initial code.  
.endm
```

所以当我们使用 Plugin 启动时, DDR 配置寄存器设置需要在这儿修改(关于如何配置与校准 DDR 参数, 请参考文档: <<MX6X_DDR3_调校_应用手册_V4_20150730.doc>>)。 imx6dqsabresd_ddr_setting 函数会被以下代码调用:

arch\arm\include\asm\arch-mx6\mx6_plugin.S

```
plugin_start:  
push {r0-r4, lr}  
imx6_ddr_setting  
imx6_clock_gating  
imx6_qos_setting
```

此代码文件也是 plugin 启动文件头的主文件, 如果希望了解 Plugin 启动的文件头格式可以认真读一下这个代码文件。

i.MX6Q Plugin启动方式典型应用：改频与展频

因为在 Plugin 的汇编代码中，没有办法通过串口来检查代码运行情况，所以我们可以采用拉 GPIO LED 灯的调试方式。此文中，我们使用将 PLL2 的时钟 iomux 到 CLKO1 输出的 GPIO 管脚，然后量测相关 GPIO 来判断 Plugin 工作，并且确认 PLL2 改频与展频是否工作。

如下寄存器，设置了 CLKO1 的 GPIO 输出时钟源，分频系数和使能控制 bit。

18.6.21 CCM Clock Output Source Register (CCM_CCOSR)

The figure below represents the CCM Clock Output Source Register (CCOSR). The CCOSR register contains bits to control the clocks that will be generated on the output ipp_do_clko1 (CCM_CLKO1) and ipp_do_clko2 (CCM_CLKO2). The table below provides its field descriptions.

Address: 20C_4000h base + 60h offset = 20C_4060h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									CLKO2_EN			CLKO2_DIV		CLKO2_SEL		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									CLK_OUT_SEL	CLK1_EN		CLKO1_DIV		CLKO1_SEL		
W												0	0	0	0	1
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

8 CLK_OUT_SEL	CCM_CLKO1 output to reflect CCM_CLKO1 or CCM_CLKO2 clocks 0 CCM_CLKO1 output drives CCM_CLKO1 clock 1 CCM_CLKO1 output drives CCM_CLKO2 clock
7 CLKO1_EN	Enable of CCM_CLKO1 clock 0 CCM_CLKO1 disabled. 1 CCM_CLKO1 enabled.
6-4 CLKO1_DIV	Setting the divider of CCM_CLKO1 000 divide by 1 001 divide by 2 010 divide by 3

CLKO1_SEL	Selection of the clock to be generated on CCM_CLKO1 0000 pll3_sw_clk (this inputs has additional constant division /2) 0001 pll2_main_clk (this inputs has additional constant division /2) 0010 pll1_main_clk (this inputs has additional constant division /2) 0011 pll5_main_clk (this inputs has additional constant division /2) 0100 video_27M_clk_root 0101 axi_clk_root 0110 enfc_clk_root 0111 ipu1_dif_clk_root 1000 ipu2_dif_clk_root 1001 ipu2_dif1_clk_root 1010 ipu2_dif1_clk_root 1011 ahb_clk_root 1100 ipg_clk_root 1101 perclk_root 1110 ckil_sync_clk_root 1111 pll4_main_clk
-----------	--

所以我们设置为：

- CLKO1_SEL=0b1: PLL2_main_clk(设置 CLKO1 的时钟源是 PLL2，他已经默认有一个固定的除 2 的系数): 528Mhz/2=264Mhz
- CLKO1_DIV=0b000: 分频系统为 1: 264Mhz/1=264Mhz
- CLKO1_EN=0b1: CCM_CLKO1 使能

i.MX6Q Plugin启动方式典型应用：改频与展频

- CLK_OUT_SEL=0b0: CCM_CLKO1 输出为 CCM_CLKO1 时钟

如下设置值：

Hex	0				8				1			
Bits	11	10	9	8	7	6	5	4	3	2	1	0
Binary	0	0	0	0	1	0	0	0	0	0	0	1

所以我们如下代码设置 CLKO1 的 iomux 和 CCOSR 寄存器：

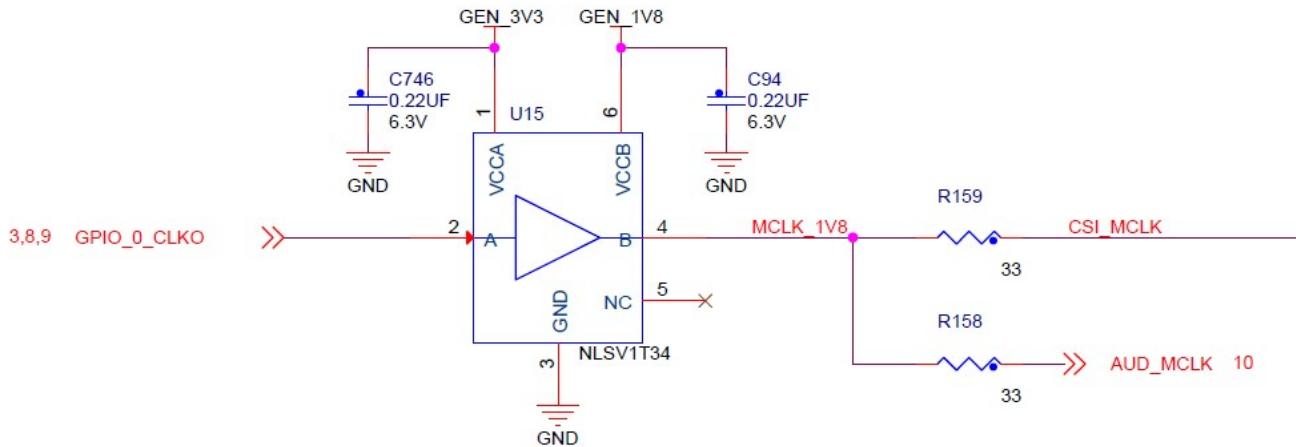
```
.macro imx6_pll2_clk01
    /*johnli add iomux pll2*/
    /*iomux gpio0 to ccm_clk01*/
    ldr r0, =IOMUXC_BASE_ADDR
    ldr r1, =0x0
    str r1, [r0, #0x220]
    /*clk01 enable, source pll2/2, drivers clk01 divider=1, (528/2)=256Mhz */
    ldr r0, =CCM_BASE_ADDR
    ldr r1, =0x81
    str r1, [r0, #0x060]
    /*end*/
.endm
```

我们使用 i.MX6Q sabresdb 板来验证，此板上我们用 GPIO_0 来 iomux 为 CCM_CLKO1：

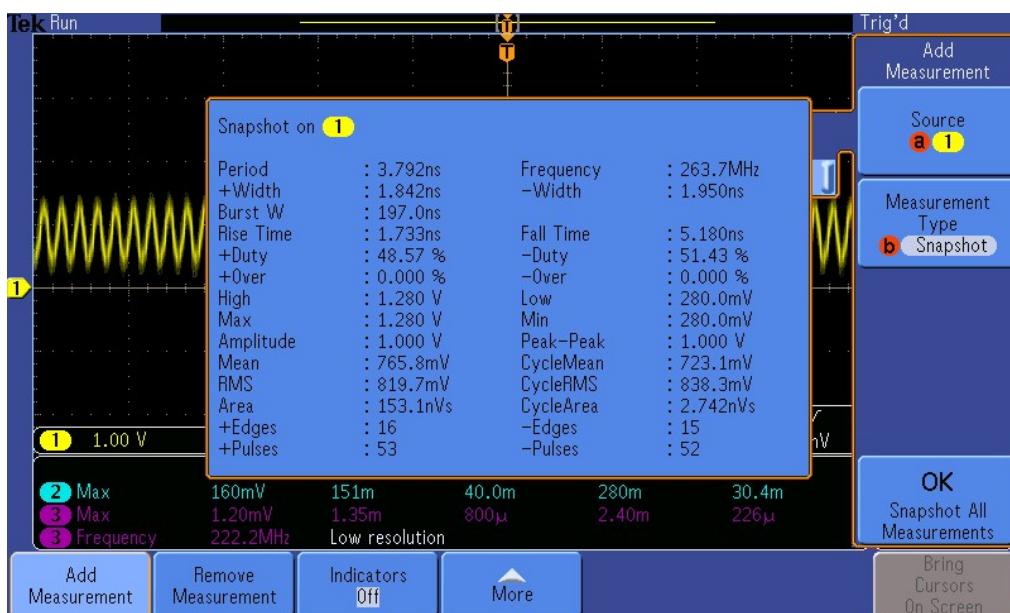


所以测试点为 R159 或者 R158：

i.MX6Q Plugin 启动方式典型应用：改频与展频

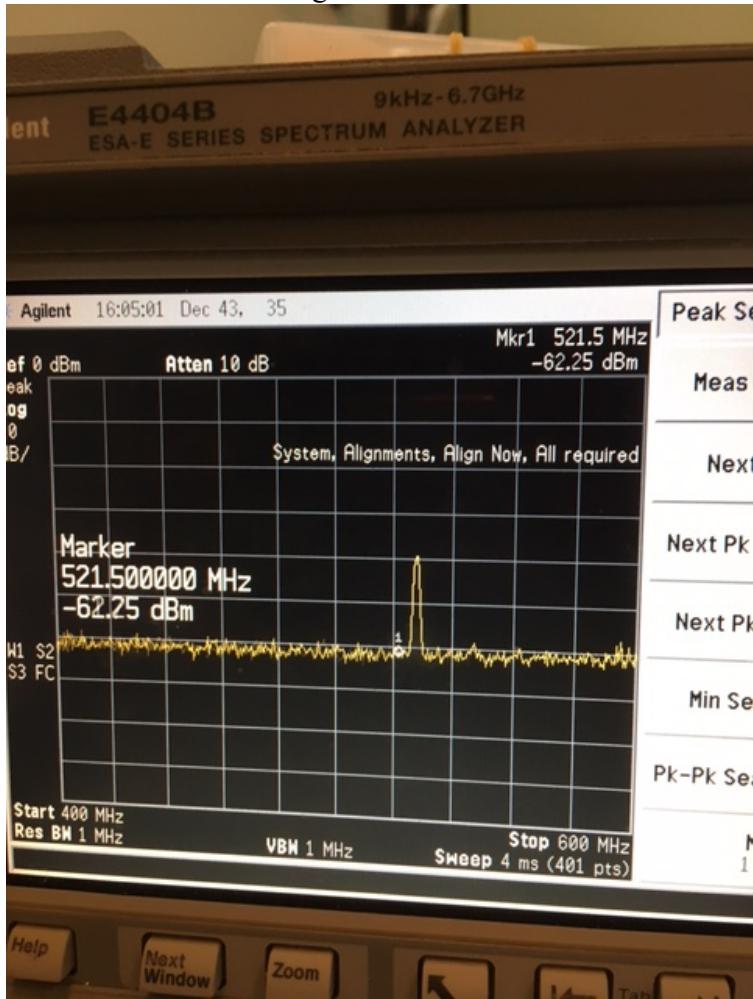


启动后键入”space”来停下 uboot,然后使用示波器量测 R159。如下测试结果,量测到的频率近似于 $528\text{Mhz}/2=264\text{Mhz}$ 。



i.MX6Q Plugin启动方式典型应用：改频与展频

所以可以确认 Plugin 代码已经工作，以下为 EMC 测试结果：



3 将 PLL2 改为 520MHz

i.MX6Q 有几个主要的应用市场，如汽车娱乐系统或虚拟仪表，一般来讲，有可能在同一块板上会设计有 GPS 模块，所以有很多客户会反映说我们的 PLL2 528MHz 会干扰 GPS 频段，因为 GPS 频段为 (L1 :1575.42 +/-10 MHz 1565.42~1585.42)。

我们的 PLL2 启动默认为 528Mhz，我们的三倍频 $3 \times 528\text{Mhz} = 1584\text{Mhz}$ 。刚好在 GPS 的频段范围内。

所以我们需要修改 PLL2 频率，但是 PLL2 本身是很多系统与外设设备的时钟源，比如说：DDR/eMMC/IPU 显示 Pixel 时钟

(注意如下：

```
Kernel\arch\arm\boot\dts\imx6qdl-sabresd.dtsi
```

i.MX6Q Plugin 启动方式典型应用：改频与展频

```

mxcfb3: fb@2 {
    compatible = "fsl,mxc_sdc_fb";
    disp_dev = "lcd";
    interface_pix_fmt = "RGB565";
    mode_str ="CLAA-WVGA";
    default_bpp = <16>;
    int_clk = <0>; // int_clk=1 则 pixel clock 来源于 ipu root clock, 而不是 PLL5
    late_init = <0>;
    status = "disabled";
};)

```

所以我们不能在系统运行起来后去修改，因为改频产生的瞬间噪声有可能导致系统死机。

所以我们只能在 DDR 初始化之前，在 Plugin 中实现。

如下分析 PLL2 时钟寄存器设置：

PLL output frequency = $F_{ref} * (\text{DIV_SELECT} + \text{NUM}/\text{DENOM})$

DIV_SELECT 寄存器值为：

18.7.4 Analog System PLL Control Register (CCM_ANALOG_PLL_SYSn)

The control register provides control for the 528MHz PLL.

Address: 20C_8000h base + 30h offset + (4d × i), where i=0d to 3d

0 DIV_SELECT	This field controls the PLL loop divider. 0 - $F_{out}=F_{ref}*20$; 1 - $F_{out}=F_{ref}*22$.
-----------------	---

默认值是 1，所以 DIV_SELECT=22，如下使用 uboot 的寄存器读命令读出的结果：

```
=> md.w 020c8030 1
020c8030: 2001 1: means DIV_SELECT=22
```

NUM 寄存器值为：

i.MX6Q Plugin启动方式典型应用：改频与展频

18.7.6 Numerator of 528MHz System PLL Fractional Loop Divider Register (CCM_ANALOG_PLL_SYS_NUM)

This register contains the numerator of 528MHz PLL fractional loop divider (signed number).

Absoulte value should be less than denominator

Address: 20C_8000h base + 50h offset = 20C_8050h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	-																															
W																																

CCM_ANALOG_PLL_SYS_NUM field descriptions	
Field	Description
31–30	Always set to zero (0).
-	
A	30 bit numerator (A) of fractional loop divider (signed integer).

使用 uboot 读寄存器命令读出来为 0:

```
=> md.w 020c8050 1
```

```
020c8050: 0000
```

DENOM 寄存器值为:

18.7.7 Denominator of 528MHz System PLL Fractional Loop Divider Register (CCM_ANALOG_PLL_SYS_DENOM)

This register contains the Denominator of 528MHz PLL fractional loop divider.

Address: 20C_8000h base + 60h offset = 20C_8060h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	-																															
W																																

CCM_ANALOG_PLL_SYS_DENOM field descriptions

Field	Description
31–30	Always set to zero (0).
-	
B	30 bit Denominator (B) of fractional loop divider (unsigned integer).

默认值是 18, uboot 读寄存器命令读出值为:

```
=> md.w 0x020c8060 1
```

```
020c8060: 0012
```

所以如下公式:

$$\text{PLL output frequency} = \text{Fref} * (\text{DIV_SELECT} + \text{NUM/DENOM})$$

i.MX6Q Plugin启动方式典型应用：改频与展频

$24\text{Mhz} * (22 + 0/18) = 528\text{Mhz}$

我们需要配置 PLL2 时钟避开 GPS 频段，由于 GPS 的最低频点是 1565Mhz。

$1565/3 = 521\text{Mhz.}$

所以我们需要配置 Pll2 为 $520\text{Mhz} = 24 * (20 + 20/12)$

如下代码：

```
/*528=24*(20+20/12) change to 520Mhz=24*(20+20/12)*/
```

```
/*set pll2 div_select=20*/
ldr r1, =0x1
str r1, [r0,#0x38]
```

```
/*set pll2 num=20*/
ldr r1, [r0,#0x50]
orr r1, r1, #0x14
str r1, [r0,#0x50]
```

```
/*set pll2 denom=12*/
ldr r1, [r0,#0x60]
bic r1, r1, #0x3FFFFFFF
orr r1, r1, #0xC
str r1, [r0,#0x60]
```

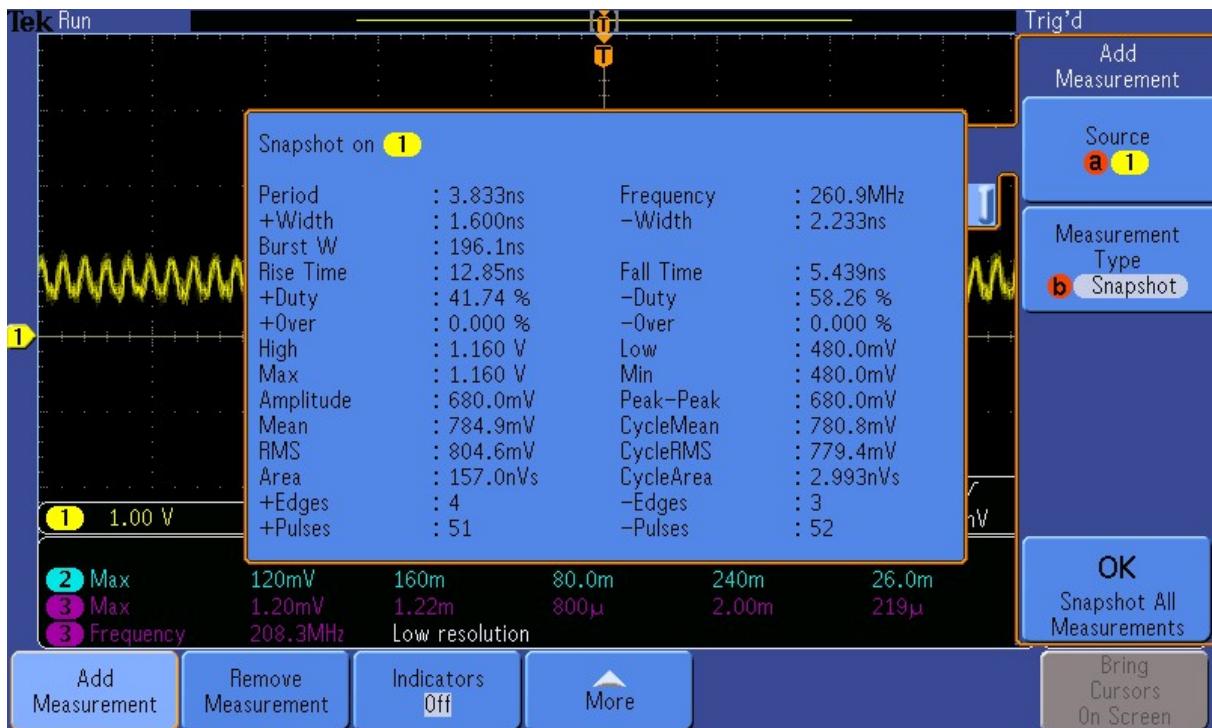
加入到 Plugin 代码后，如下为 uboot 读寄存器命令读出来的验证结果：

```
=> md.w 020c8030 1
020c8030: 2000 ..
=> md.w 020c8050 1
020c8050: 0014 ..
=> md.w 020c8060 1
020c8060: 000c ..
=>
```

如下为 CLKO1 的示波器测试结果：

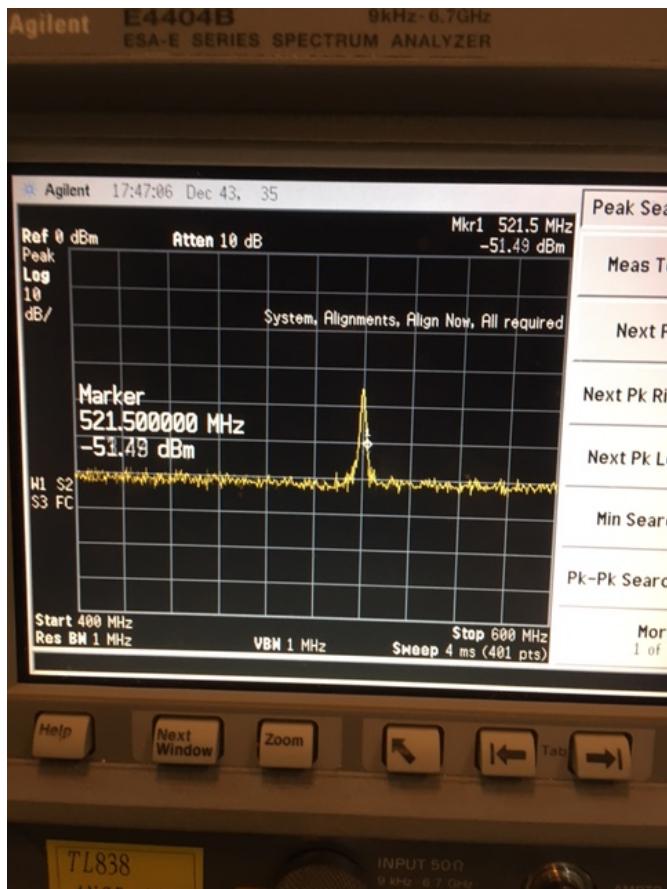
测试结果接近于 $520\text{Mhz}/2 = 260\text{Mhz.}$

i.MX6Q Plugin启动方式典型应用：改频与展频



EMC 测试结果为：

i.MX6Q Plugin启动方式典型应用：改频与展频



注意

- 修改 PLL2 需要一些保护代码，详情请参考 第 5 章
- CCM_ANALOG_PLL_SYSn 寄存器有专门的读，写，清位与 toggle 访问寄存器，如下：

20C_8030	Analog System PLL Control Register (CCM_ANALOG_PLL_SYS)	32	R/W	0001_3001h	18.7.4/911
20C_8034	Analog System PLL Control Register (CCM_ANALOG_PLL_SYS_SET)	32	R/W	0001_3001h	18.7.4/911
20C_8038	Analog System PLL Control Register (CCM_ANALOG_PLL_SYS_CLR)	32	R/W	0001_3001h	18.7.4/911

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/page
20C_803C	Analog System PLL Control Register (CCM_ANALOG_PLL_SYS_TOG)	32	R/W	0001_3001h	18.7.4/911

所以如果我们要设置或清除此寄存器某些位，可以直接操作这些寄存器。

注意，以上是以 GPS 频段来说明的，但是如下北斗频段：

i.MX6Q Plugin启动方式典型应用：改频与展频

Beidou	Freq.	BW
B1I	1561.098MHz	$\pm 2.046\text{MHz}(1\text{dB})/\pm 8\text{MHz}(3\text{dB})$

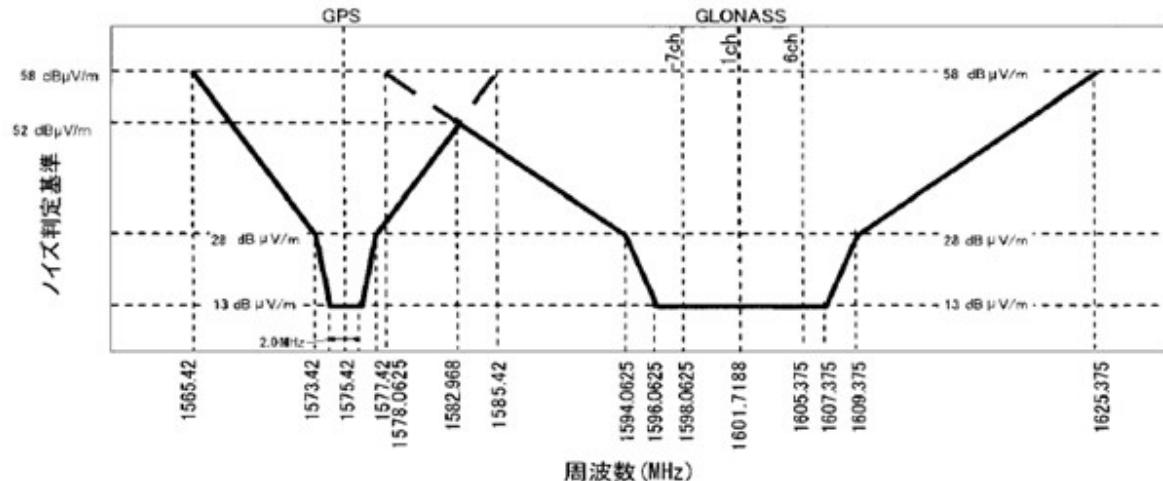


図3 GPS-GLONASS帯の目標値（電界アンテナ法 アベレージ検波）（補正值を用いた場合）

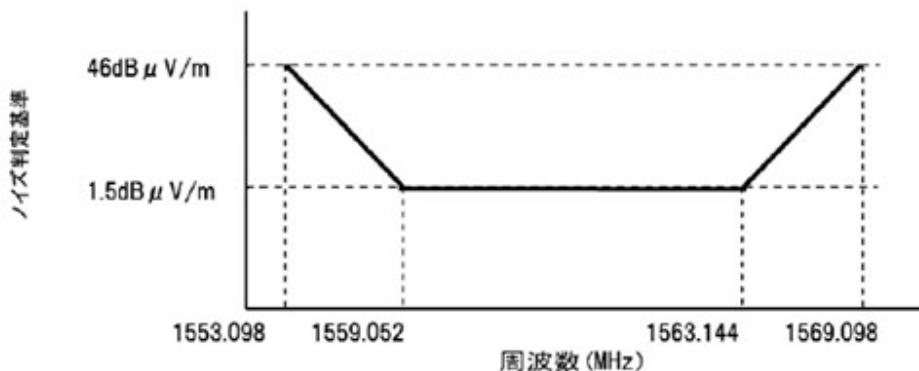


図4 中国BDS帯の目標値（電界アンテナ法 アベレージ検波）（補正值用いた場合）

则原 GPS 频段的 520Mhz，会有影响：

520Mhz X 3=1560MHz

我们需要配置 PLL2 时钟避开北斗频段，由于北斗的最低频点是 1553Mhz。

$1553/3=517\text{Mhz}$.

所以公式更新为：

所以我们需要配置 Pll2 为 $516\text{Mhz}=24*(20+18/12)$

i.MX6Q Plugin启动方式典型应用：改频与展频

所以 NUM 寄存器值应该从 20 修改为 18:

代码中下一段应该修改为:

```
/*set pll2 num=18*/  
ldr r1, [r0,#0x50]  
orr r1, r1, #0x12  
str r1, [r0,#0x50]
```

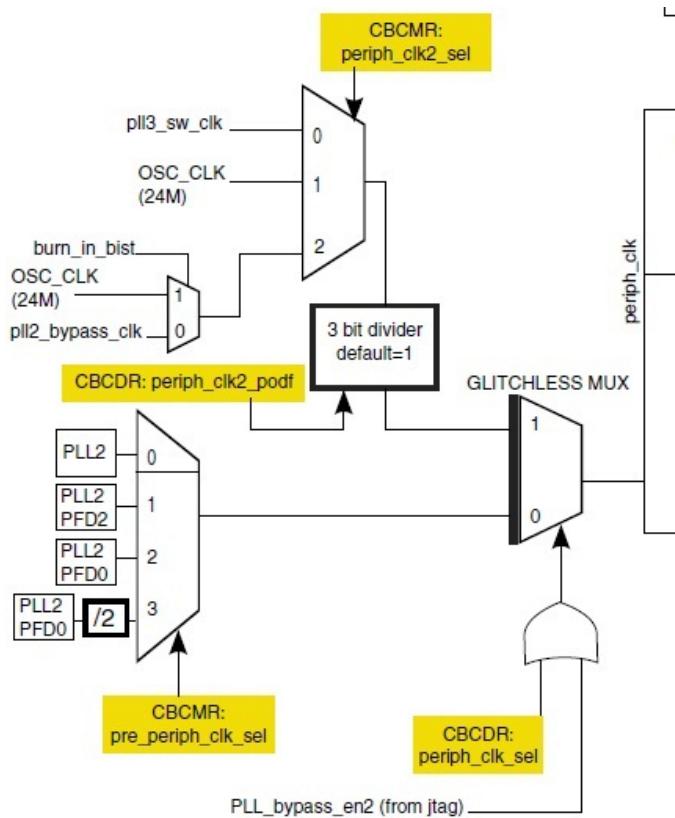
所以

- 如果是要 PLL2 完全避开 GPS/GLONASS 和北斗的频段，则需要将 PLL2 移动到 516MHz。
- 如果只是要避开 GPS 频段，则需要将 PLL2 移动到 520MHz。
- 如果只是要避开北斗频段，则不需要移动频率，因为原来的 $528\text{MHz} \times 3 = 1584\text{MHz} > (1561.098\text{MHz} + 8\text{MHz} = 1569.098\text{MHz})$ 。但是这个频点会影响到 GPS(L1 :1575.42 +/-10 MHz 1565.42~1585.42)和 GLONASS(1578MHz~1625MHz)。

4 展频(spread spectrum)

PLL2 支持展频，从而减少辐射能量聚集，通过调制让频率散布一个比较宽的频带中，从而减少辐射能力峰值，同时，展频的PLL也需要比不展频的PLL更长时间来锁定。

作为系统外设时钟，很多外设如 DDR/IPU 显示 Pixel 时钟/eMMC SD_CLK 都是来自于 PLL2，由于这些时钟都是裸露在芯片外部的，有可能导致 EMI 问题。所以除了调节相关管脚的驱动能力，多数时候，也需要使能展频来通过 EMI 测试。i.MX6Q 仅支持 PLL2 的展频功能（因为 PLL2 是系统外设主时钟），并且只支持向下展频，比如说我们需要展 6MHz，则展频范围是 $528\text{MHz} - 6\text{MHz} = 522\text{MHz} \sim 528\text{MHz}$ 。在 PLL2 展频后，以 PLL2 为源的外设时钟都会相应比例展频，所以客户需要确认其外设可以支持展频后的时序。另外，在设置展频相关寄存器时，需要关掉 PLL2 并在设置完成后再打开，所以在关掉 PLL2 时，需要使用临时时钟，我们使用 OSC_CLK(24MHz) 如下 PLL2 框图：



图中黄色部分用于控制 Periph_clk(它是 DDR 的时钟源)的时钟源，我们需要先设置 periph_clk2_sel 来自于 OSC_CLK，使用 Periph_clk 来自于 periph_clk2_sel，也就是 OSC_CLK，然后关闭 PLL2，设置展频配置，再打开，再将 Periph_clk 切换回 PLL2。这些操作都需要在 DDR 初始化之前，也就是 Plugin 中完成。

展频由CCM_ANALOG_PLL_SYS_SS 寄存器配置，使能后，PLL2输出频率会按照STEP域定义的频点往下变化，直到STOP定义的频率，然后掉头升回到以前频率。如下公式说明：

$$\text{Spectrum spread range} = (\text{ssc_stop})/\text{DENOM} * \text{Fref}$$

$$\text{Modulation frequency} = \text{Fref} * (\text{ssc_step})/(2*\text{ssc_stop})$$

注意：

- 我们验证过的最大展频范围是 30MHz，以上没有验证过，当然在 30MHz 范围内，客户也需要确认其外设是否满足时序要求。
- 我们要求的调制频率为 48K~64K。我们使用这个值倒推出相关 ssc_step/ssc_stop 寄存器值设定。

所以 $(\text{ssc_stop}/\text{ssc_step}) = 24000K/(2*(48K~64K)) = 250 \sim 187.5$.

所以如果我们设置 $\text{ssc_step}=0x6$ (双变量需要约束一个), 那 $\text{ssc_stop}=1500 \sim 1125$

i.MX6Q Plugin启动方式典型应用：改频与展频

然后如果要求 spectrum spread range =6Mhz

那么(DENOM/ ssc_stop)=4 so DENOM=6000~4500

如果要求spectrum spread range =24Mhz.(我们使用24MHz为例，而不是最大值30MHz)

那么(DENOM/ ssc_stop)=1 so DENOM=1500~1125

所以如果我们选择 DENOM=4800, spectrum spread range=6M, ssc_stop=1200, SSC_step=6, Modulation frequency=60K.

CCM_ANALOG_PLL_SYS_SS=0x4B08006

CCM_ANALOG_PLL_DENOM=0x12C0

18.7.5 528MHz System PLL Spread Spectrum Register (CCM_ANALOG_PLL_SYS_SS)

This register contains the 528 PLL spread spectrum controls.

Address: 20C_8000h base + 40h offset = 20C_8040h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	STOP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	STEP															
W	ENABLE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CCM_ANALOG_PLL_SYS_SS field descriptions

Field	Description
31–16 STOP	Frequency change = stop/CCM_ANALOG_PLL_SYS_DENOM[B]*24MHz.
15 ENABLE	0 — Spread spectrum modulation disabled 1 — Spread spectrum modulation enabled
STEP	Frequency change step = step/CCM_ANALOG_PLL_SYS_DENOM[B]*24MHz.

然后如果再展频后 还需要改频率为 520MHz PLL。 520=24*(20+num/4800), So num =8000

CCM_ANALOG_PLL_NUM=0x1F40

i.MX6Q Plugin启动方式典型应用：改频与展频

而如果选择 DENOM=1200, spectrum spread range=24M, ssc_stop=1200, SSC_step=6, Modulation frequency=60K.

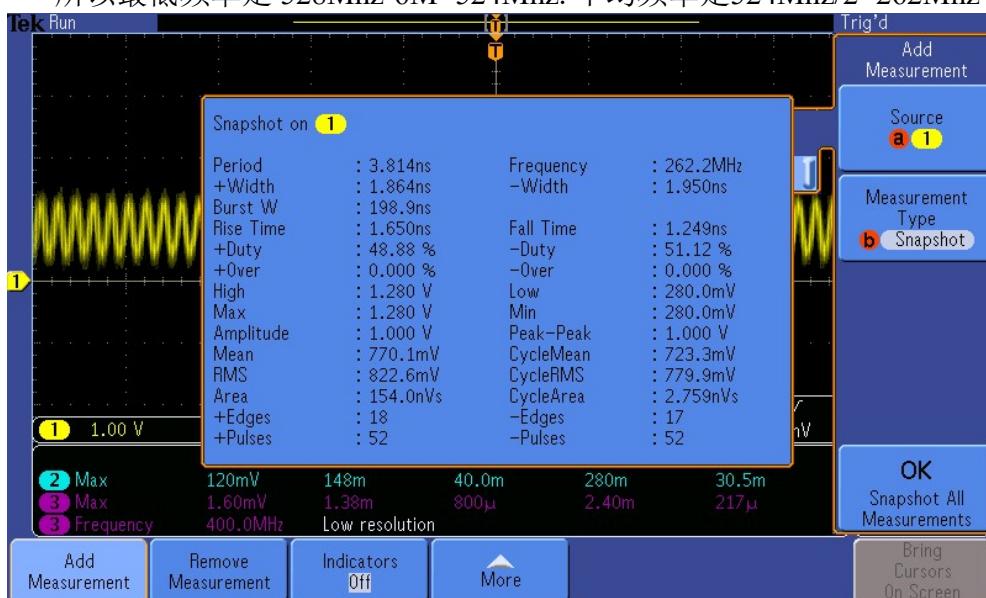
CCM_ANALOG_PLL_SYS_SS=0x4B08006
CCM_ANALOG_PLL_DENOM=0x4B0

然后如果再展频后 还需要改频率为 520Mhz PLL. $520 = 24 * (20 + \text{num}/1200)$, So num =2000
CCM_ANALOG_PLL_NUM =0x7D0

我们可以使用uboot md寄存器读命令来检查Plugin代码寄存器设置，并量测CLKO1来确认结果：

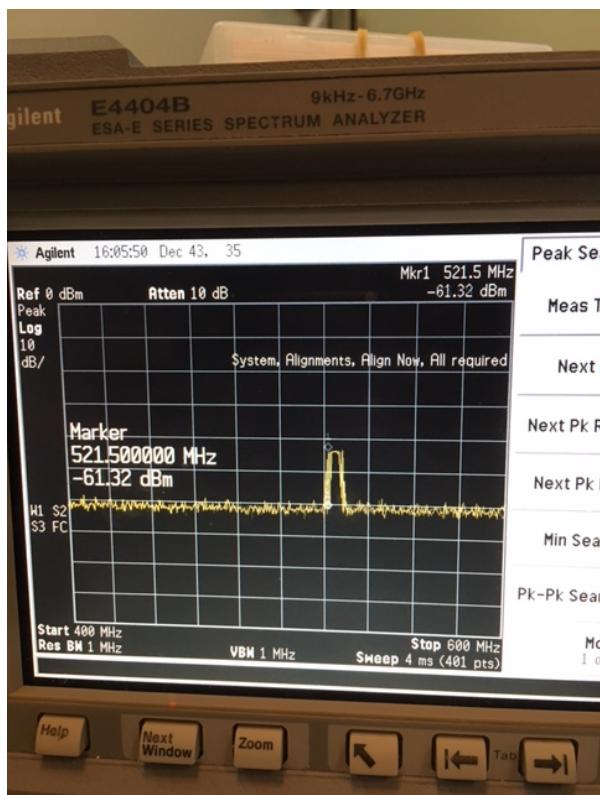
- CONFIG_PLL2_6M_SSC

所以最低频率是 528Mhz-6M=524Mhz. 平均频率是524Mhz/2=262Mhz

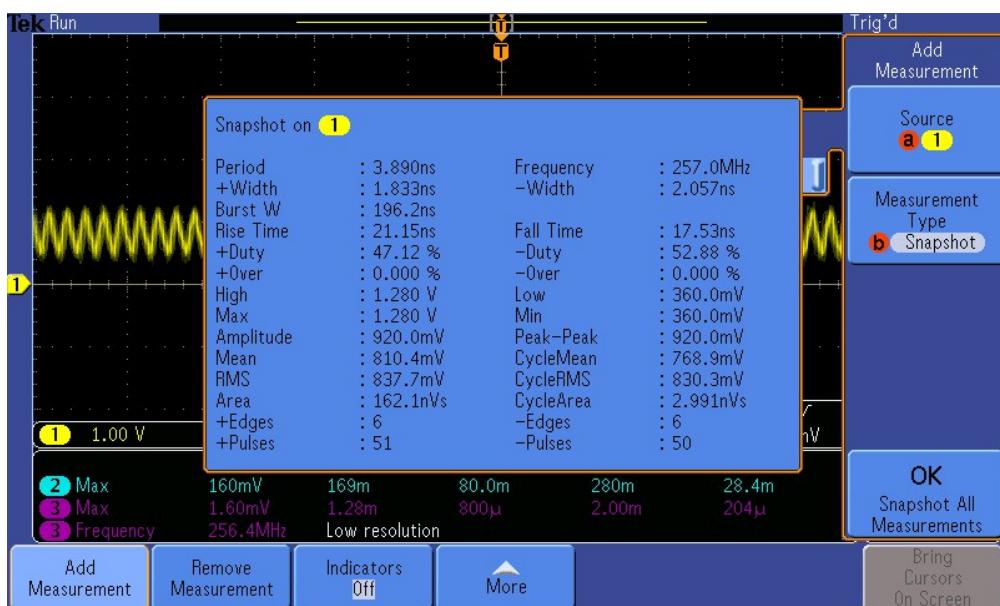


如下 EMC 测试结果：

i.MX6Q Plugin启动方式典型应用：改频与展频

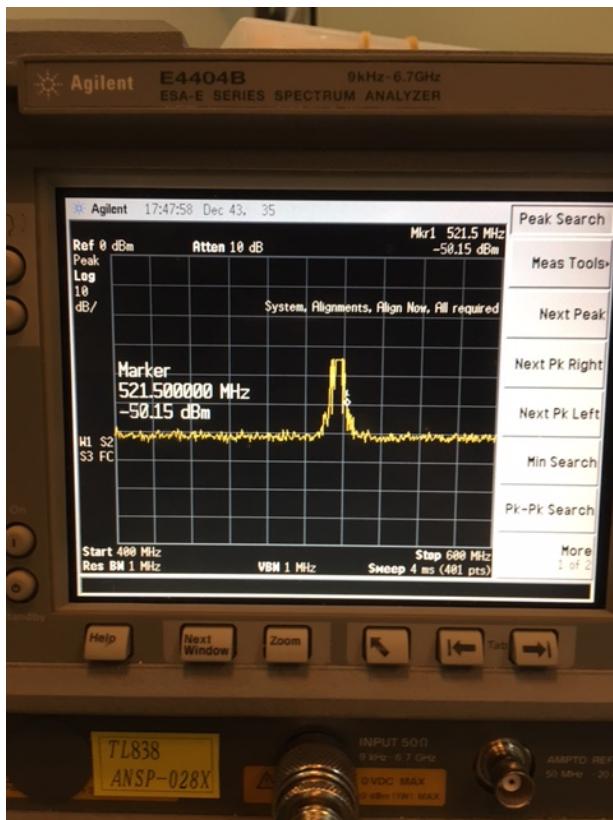


- CONFIG_PLL2_520_6M_SSC
所以最低频率是520Mhz-6M=514Mhz. 平均频率是514Mhz/2=257Mhz



如下 EMC 测试结果:

i.MX6Q Plugin启动方式典型应用：改频与展频



- CONFIG_PLL2_24M_SSC

所以最低频率是 $528\text{MHz} - 24\text{MHz} = 504\text{MHz}$, 平均频率是 $504\text{MHz}/2 = 252\text{MHz}$

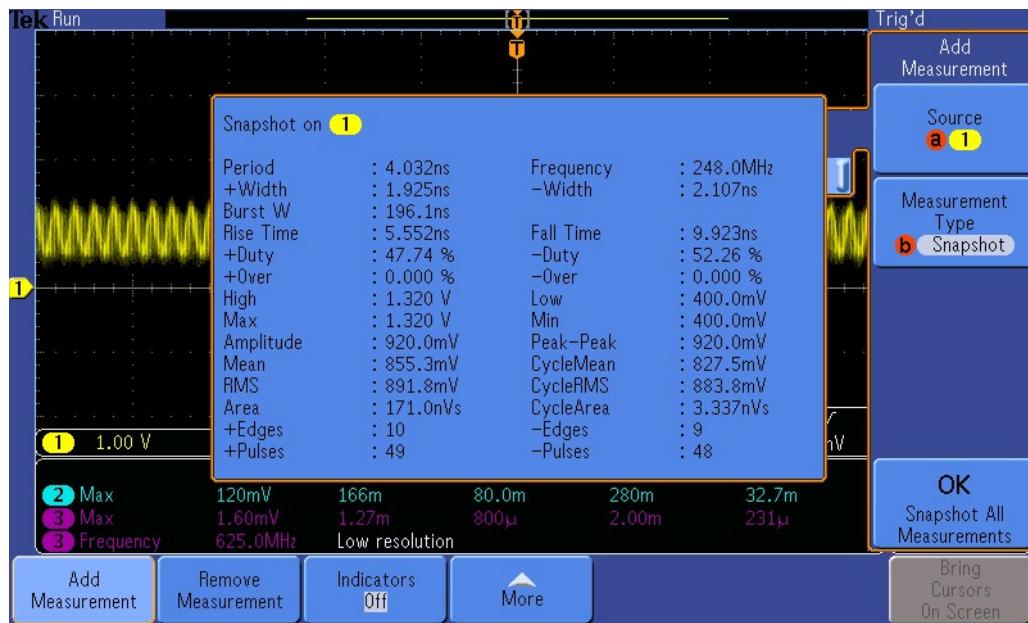


如下 EMC 测试结果: (N/A)

- CONFIG_PLL2_520_24M_SSC

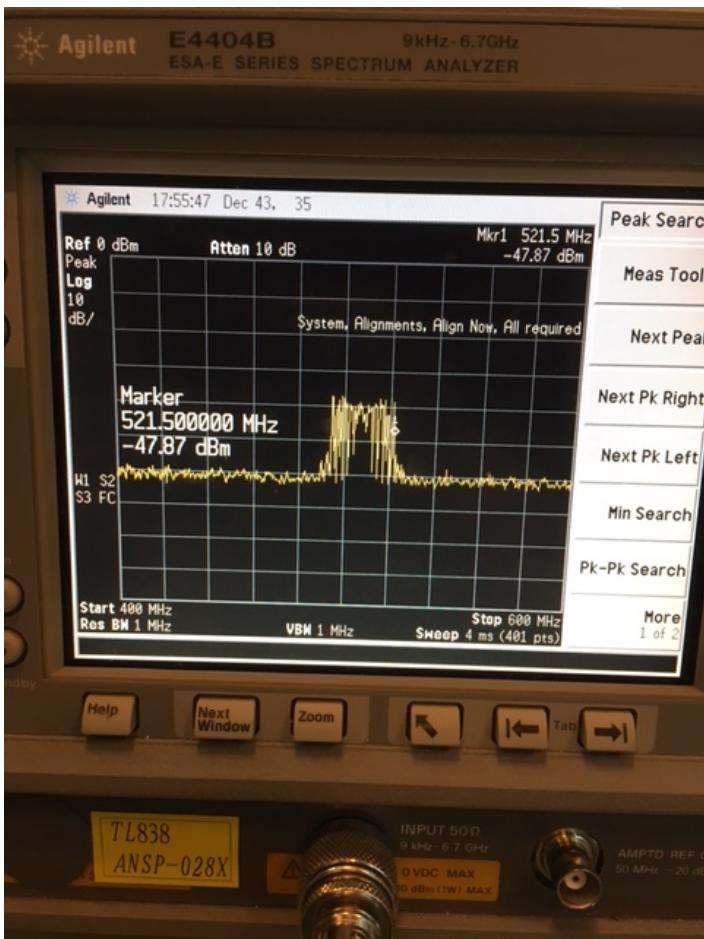
i.MX6Q Plugin启动方式典型应用：改频与展频

所以最低频率是 $520\text{Mhz}-24\text{M}=496\text{Mhz}$. 平均频率是 $496\text{Mhz}/2=248\text{Mhz}$



如下 EMC 测试结果:

i.MX6Q Plugin启动方式典型应用：改频与展频



5 Uboot 源代码修改总结

修改的源代码如下:

```

Include\configs\mx6sabre_common.h
/* uncomment for PLUGIN mode support */
/* #define CONFIG_USE_PLUGIN */
#define CONFIG_USE_PLUGIN //johnli
#ifndef CONFIG_USE_PLUGIN
#define CONFIG_PLUGIN_ADD
#ifndef CONFIG_PLUGIN_ADD
#define CONFIG_PLL2_CLKO1 //iomux pll2 to probe for test purpose, can open or close singly
#define CONFIG_PLL2_520 //if test this item, close CONFIG_PLL2_6M_SSC and CONFIG_PLL2_24M_SSC, normally,
just for test purpose
/* #define CONFIG_PLL2_6M_SSC //if test this item, close CONFIG_PLL2_520 and CONFIG_PLL2_24M_SSC*/
#ifndef CONFIG_PLL2_6M_SSC
#define CONFIG_PLL2_520_6M_SSC /*add to support 520Mhz*/
#endif
/* #define CONFIG_PLL2_24M_SSC //if test this item, close CONFIG_PLL2_520 and CONFIG_PLL2_6M_SSC*/
#ifndef CONFIG_PLL2_24M_SSC

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```
#define CONFIG_PLL2_520_24M_SSC /*add to support 520Mhz*/
#endif
#endif
#endif
```

```
Arch\arm\include\asm\arch-mx6\mx6_plugin.S
plugin_start:
    push {r0-r4, lr}
    //johnli add
#ifndef CONFIG_PLUGIN_ADD
imx6_plugin_add //because pll2 is also the DDR clock source, so we need add the code before DDR initial.
#endif
    imx6_ddr_setting
```

board\freescale\mx6sabresd\plugin.S

```
#ifdef CONFIG_PLUGIN_ADD
#ifndef CONFIG_PLL2_520
.macro imx6_pll2_520
/*johnli change pll2 to 520Mhz*/
    ldr r0, =CCM_BASE_ADDR
    ldr r1, [r0,#0x18]
    ldr r3, [r0,#0x18]

/* set periph_cclk_sel to select OSC_CLK */
    and r1, r1, #0xfffffcfff
    orr r1, r1, #0x00001000
    str r1, [r0,#0x18]

/* set periph_clk_set to switch to OSC_CLK */
    ldr r1, [r0,#0x14]
    ldr r2, [r0,#0x14]
    orr r1, r1, #0x02000000
    str r1, [r0,#0x14]

/* Switch to pll2 bypass clock*/
    ldr r0, =ANATOP_BASE_ADDR
    ldr r1, =0x10000
    str r1, [r0,#0x34]

/*528=24*(20+20/12)*/
/*set pll2 div_select=20*/
    ldr r1, =0x1
    str r1, [r0,#0x38]

/*set pll2 num=20*/
    ldr r1, [r0,#0x50]
    orr r1, r1, #0x14
    str r1, [r0,#0x50]

/*set pll2 denom=12*/
    ldr r1, [r0,#0x60]
```

i. MX6Q Plugin启动方式典型应用：改频与展频

```

bic r1, r1, #0x3FFFFFFF
orr r1, r1, #0xC
str r1, [r0,#0x60]

// Delay 300ns at least
ldr r4, =0x0
pu delay:
add r4, r4, #0x1
cmp r4, #0x200000
bne pu_delay

/* Switch to pll2 main clock*/
ldr r1, =0x10000
str r1, [r0,#0x38]

/* recovery the previous PLL source setting */
ldr r0, =CCM_BASE_ADDR
str r2, [r0,#0x14]
str r3, [r0,#0x18]
.endm
#endif

#endif CONFIG_PLL2_6M_SSC
.macro imx6_pll2_6m_ssc
/*johnli set pll2 to 6Mhz SSC*/
ldr r0, =CCM_BASE_ADDR
ldr r1, [r0,#0x18]
ldr r3, [r0,#0x18]

/* set periph_cclk2_sel to select OSC_CLK */
and r1, r1, #0xffffcff
orr r1, r1, #0x00001000
str r1, [r0,#0x18]

/* set periph_clk_set to switch to OSC_CLK */
ldr r1, [r0,#0x14]
ldr r2, [r0,#0x14]
orr r1, r1, #0x02000000
str r1, [r0,#0x14]

/* Switch to pll2 bypass clock*/
ldr r0, =ANATOP_BASE_ADDR
ldr r1, =0x10000
str r1, [r0,#0x34]

#endif CONFIG_PLL2_520_6M_SSC
/*528=24*(20+8000/4800)*/

/*set pll2 div_select=20*/
ldr r1, =0x1
str r1, [r0,#0x38]

/*set pll2 num=8000*/
ldr r1, [r0,#0x50]

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

        orr r1, r1, #0x1F40
        str r1, [r0,#0x50]
#endif

/*set pll2 ssc stop=1200/step=6 enable=1*/
ldr r1,=0x4B08006
str r1, [r0,#0x40]

/*set pll2 denom=4800*/
ldr r1, [r0,#0x60]
bic r1, r1, #0x3FFFFFFF
orr r1, r1, #0x12C0
str r1, [r0,#0x60]

// Delay 300ns at least
ldr r4, =0x0
pu_delay:
add r4, r4, #0x1
cmp r4, #0x200000
bne pu_delay

/* Switch to pll2 main clock*/
ldr r1, =0x10000
str r1, [r0,#0x38]

/* recovery the previous PLL source setting */
ldr r0, =CCM_BASE_ADDR
str r2, [r0,#0x14]
str r3, [r0,#0x18]
.endm
#endif

#ifndef CONFIG_PLL2_24M_SSC
.macro imx6_pll2_24m_ssc
/*johnli set pll2 to 24Mhz SSC*/
ldr r0, =CCM_BASE_ADDR
ldr r1, [r0,#0x18]
ldr r3, [r0,#0x18]

/* set periph_clik2_sel to select OSC_CLK */
and r1, r1, #0xffffcff
orr r1, r1, #0x00001000
str r1, [r0,#0x18]

/* set periph_clk_set to switch to OSC_CLK */
ldr r1, [r0,#0x14]
ldr r2, [r0,#0x14]
orr r1, r1, #0x02000000
str r1, [r0,#0x14]

/* Switch to pll2 bypass clock*/
ldr r0, =ANATOP_BASE_ADDR
ldr r1, =0x10000
str r1, [r0,#0x34]

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

#ifndef CONFIG_PLL2_520_24M_SSC
/*528=24*(20+2000/1200)*/

/*set pll2 div_select=20*/
ldr r1, =0x1
str r1, [r0,#0x38]

/*set pll2 num=2000*/
ldr r1, [r0,#0x50]
orr r1, r1, #0x7D0
str r1, [r0,#0x50]
#endif

/*set pll2 ssc stop=1200/step=6 enable=1*/
ldr r1,=0x4B08006
str r1, [r0,#0x40]

/*set pll2 denom=1200*/
ldr r1, [r0,#0x60]
bic r1, r1, #0x3FFFFFFF
orr r1, r1, #0x4B0
str r1, [r0,#0x60]

// Delay 300ns at least
ldr r4, =0x0
pu_delay:
add r4, r4, #0x1
cmp r4, #0x200000
bne pu_delay

/* Switch to pll2 main clock*/
ldr r1, =0x10000
str r1, [r0,#0x38]

/* recovery the previous PLL source setting */
ldr r0, =CCM_BASE_ADDR
str r2, [r0,#0x14]
str r3, [r0,#0x18]
.endm
#endif

#endif CONFIG_PLL2_CLKO1
.macro imx6_pll2_clk01
/*johnli add iomux pll2*/
/*iomux gpio0 to ccm_clk01*/
ldr r0, =IOMUXC_BASE_ADDR
ldr r1, =0x0
str r1, [r0, #0x220]
/*clk01 enable, source pll2/2, drivers clk01 divider=8, (528/2)=264Mhz */
ldr r0, =CCM_BASE_ADDR
ldr r1, =0x81
str r1, [r0, #0x060]
/*end*/

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

.endm
#endif

.macro imx6_plugin_add
#if defined(CONFIG_PLL2_520)
    imx6_pll2_520
#elif defined(CONFIG_PLL2_6M_SSC)
    imx6_pll2_6m_ssc
#elif defined(CONFIG_PLL2_24M_SSC)
    imx6_pll2_24m_ssc
#endif

#ifndef CONFIG_PLL2_CLKO1
    imx6_pll2_clko1
#endif
.endm
#endif

```

6 修改内核代码来支持 520Mhz PLL2

我们默认的 BSP 使用 PLL2=528MHz. 所以没有支持相应的 PLL2 频率设置 API, 需要增加 API 来支持, 实际上由于不可以在内核运行时修改 PLL2, 所以只有读函数有意义, 而读函数主要用于支持/unit_test/dump_clocks.sh 调试使用:

Arch/arm/mach-imx/clk.h

```

enum imx_pll3_type {
    IMX_PLLV3_GENERIC,
    IMX_PLLV3_BUS, //johnli add for pll2=520Mhz add a new macro to support pll2 bus pll
    ...
};

```

Arch/arm/mach-imx/clk-imx6q.c

```

clk[IMX6QDL_CLK_PLL2] = imx_clk_pll3(IMX_PLLV3_BUS/* johnli for pll2=520Mhz
IMX_PLLV3_GENERIC*/, "pll2", "pll2_bypass_src", base + 0x30, 0x1); //change to call the function with new macro

```

Arch/arm/mach-imx/clk-pllv3.c

```

//johnli add to support pll2 520Mhz.
#define PLL_BUS_NUM_OFFSET          0x20 //add the SYS_PLL2 num/denom offset value
#define PLL_BUS_DENOM_OFFSET        0x30

//end

struct clk *imx_clk_pll3(enum imx_pll3_type type, const char *name,
                        const char *parent_name, void __iomem *base,
                        u32 div_mask)

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

{
...
//johnli for pll2=520Mhz //call the new APIs
    case IMX_PLLV3_BUS:
        ops = &clk_pll3_bus_ops;
        break;
//end
...
    if (cpu_is_imx7d() && type == IMX_PLLV3_ENET)
        pll->powerdown = ENET_PLL_POWER;
    else if (cpu_is_imx7d() && type == IMX_PLLV3_AV) {
        pll->num_offset = PLL_AV_NUM_OFFSET;
        pll->denom_offset = PLL_AV_DENOM_OFFSET;
    }
//johnli for pll2=520Mhz , fix the num/denom offset value
    else if (type == IMX_PLLV3_BUS) {
        pll->num_offset = PLL_BUS_NUM_OFFSET;
        pll->denom_offset = PLL_BUS_DENOM_OFFSET;
    }
//end
...
}

//add the new APIs
//johnli add pll2 520Mhz support
static unsigned long clk_pll3_bus_recalc_rate(struct clk_hw *hw,
                                              unsigned long parent_rate)
{
    struct clk_pll3 *pll = to_clk_pll3(hw);
    u32 mfn = readl_relaxed(pll->base + pll->num_offset);
    u32 mfd = readl_relaxed(pll->base + pll->denom_offset);
    u32 div = (1 == (readl_relaxed(pll->base) & pll->div_mask))?22:20;
    u64 temp64 = (u64)parent_rate;
    temp64 *= mfn;
    do_div(temp64, mfd);
}

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

    return (parent_rate * div) + (u32)temp64;
}

static long clk_pll3_bus_round_rate(struct clk_hw *hw, unsigned long rate,
                                    unsigned long *prate)
{
    unsigned long parent_rate = *prate;
    unsigned long min_rate = parent_rate * 20;
    unsigned long max_rate = parent_rate * 22;
    u32 div;
    u32 mfn, mfd = 1000000;
    s64 temp64;
    if (rate > max_rate)
        rate = max_rate;
    else if (rate < min_rate)
        rate = min_rate;
    div = rate / parent_rate;
    temp64 = (u64) (rate - div * parent_rate);
    temp64 *= mfd;
    do_div(temp64, parent_rate);
    mfn = temp64;
    return parent_rate * div + parent_rate / mfd * mfn;
}

static int clk_pll3_bus_set_rate(struct clk_hw *hw, unsigned long rate,
                                 unsigned long parent_rate)
{
    struct clk_pll3 *pll = to_clk_pll3(hw);
    unsigned long min_rate = parent_rate * 20;
    unsigned long max_rate = parent_rate * 22;
    u32 val, div;
    u32 mfn, mfd = 1000000;
    s64 temp64;
    if (rate < min_rate || rate > max_rate)
        return -EINVAL;
    div = rate / parent_rate;
    temp64 = (u64) (rate - div * parent_rate);
}

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

temp64 *= mfd;
do_div(temp64, parent_rate);
mfn = temp64;
val = readl_relaxed(pll->base);
val &= ~pll->div_mask;
val |= ((22 == div)?1:0);
	writel_relaxed(val, pll->base);
	writel_relaxed(mfn, pll->base + pll->num_offset);
	writel_relaxed(mfd, pll->base + pll->denom_offset);
return clk_pll3_wait_lock(pll);
}

```

```

static const struct clk_ops clk_pll3_bus_ops = {
    .prepare = clk_pll3_prepare,
    .unprepare = clk_pll3_unprepare,
    .is_prepared = clk_pll3_is_prepared,
    .recalc_rate = clk_pll3_bus_recalc_rate,
    .round_rate = clk_pll3_bus_round_rate,
    .set_rate = clk_pll3_bus_set_rate,
};

//end

```

另外，BSP 里还有一些写死的代码，需要手动全文搜索修改：

```

Arch/arm/mach-imx/clk_mx6q.c
#endif //johnli change pll2=520Mhz.

if (cpu_is_imx6dl()) {
    imx_clk_set_parent(clk[IMX6QDL_CLK_GPU3D_SHADER_SEL],
clk[IMX6QDL_CLK_PLL2_PFD1_594M]);
    imx_clk_set_rate(clk[IMX6QDL_CLK_GPU3D_SHADER], 520000000);
    /* for mx6dl, change gpu3d_core parent to 594_PFD*/
    imx_clk_set_parent(clk[IMX6QDL_CLK_GPU3D_CORE_SEL],
clk[IMX6QDL_CLK_PLL2_PFD1_594M]);
    imx_clk_set_rate(clk[IMX6QDL_CLK_GPU3D_CORE], 520000000);
} else if (cpu_is_imx6q()) {
    if (imx_get_soc_revision() == IMX_CHIP_REVISION_2_0) {

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

...
    imx_clk_set_parent(clk[IMX6QDL_CLK_GPU3D_CORE_SEL],
clk[IMX6QDL_CLK_MMDC_CH0_AXI]);

    imx_clk_set_rate(clk[IMX6QDL_CLK_GPU3D_CORE], 520000000);

...
}

}

#else

...

Arch/arm/mach-imx/ddr3-freq-imx6.S

poll_dvfs_set_2:

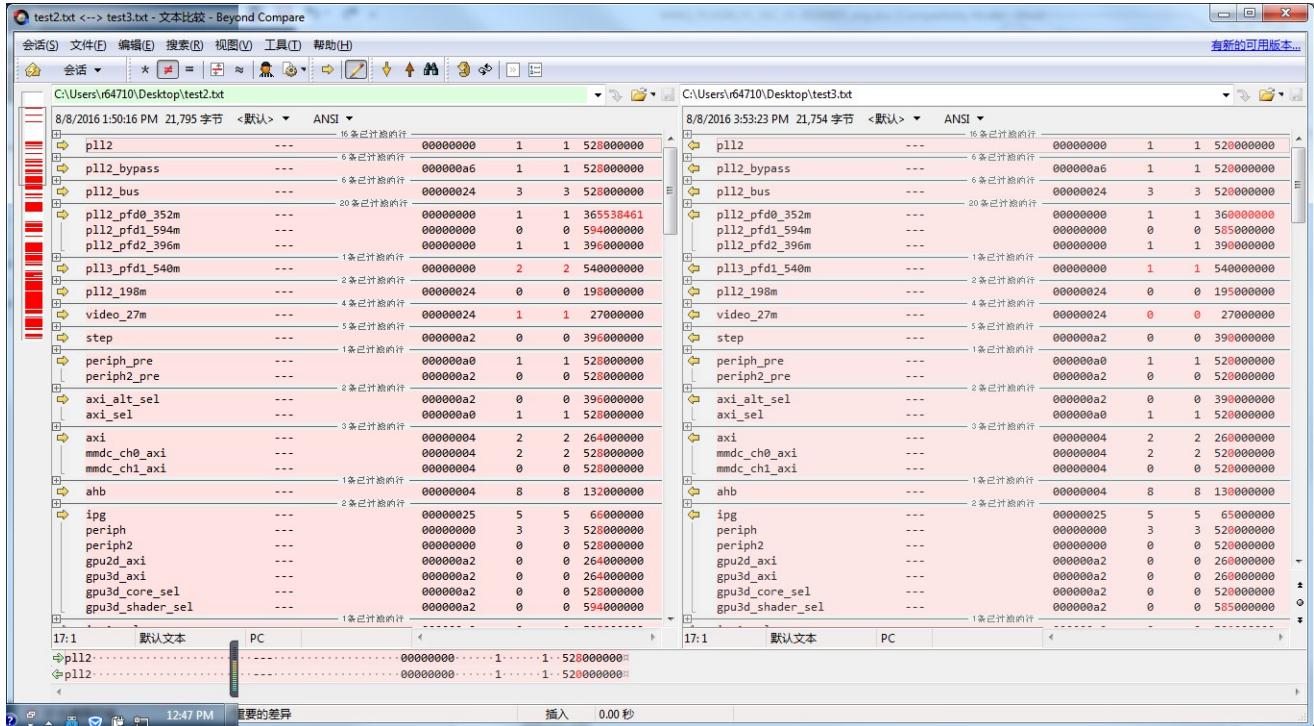
...
ldr      r1, =520000000 /*johnli change pll2=520mhz 528000000*/

```

Arch/arm/boot/dts/imx6q.dtsi

```
fsl,max_ddr_freq = <520000000>; /*johnli change pll2=520Mhz <528000000>;*/
```

使用/unit_test/dump-clocks.sh 命令，可以打印出所有的 clock 树，再改动了 PLL2 的频率后，可以发现相应的外设时钟频率也修改了：



i.MX6Q Plugin启动方式典型应用：改频与展频

test2.txt <-> test3.txt - 文本比较 - Beyond Compare

会话(5) 文件(E) 编辑(E) 搜索(V) 视图(V) 工具(I) 帮助(H)

C:\Users\r64710\Desktop\test2.txt C:\Users\r64710\Desktop\test3.txt

8/8/2016 1:50:16 PM 21,795 字节 <默认> ANSI 1条已计数的行

ipu1_sel --- 000000a2 0 0 528000000
ipu2_sel --- 000000a2 1 1 528000000
ldb_dil_sel --- 000000a4 1 1 365538461
ldb_dil_div_sel --- 000000a4 0 0 365538461
ldb_dil_div_sel --- 000000a6 1 1 104439560
ldb_dil_div_sel --- 000000a6 0 0 52219788

hs1_tx_sel --- 000000a2 1 1 396000000
pcie_axi_sel --- 000000a2 0 0 264000000

ipu2_di0_sel --- 000000a6 1 1 104439560

usdhc1_sel --- 00000000 0 0 396000000
usdhc2_sel --- 00000000 0 0 396000000
usdhc3_sel --- 00000000 0 0 396000000
usdhc4_sel --- 00000000 0 0 396000000
enfc_sel --- 000000a2 0 0 396000000
emi_sel --- 00000000 0 0 396000000
emi_slow_sel --- 00000000 1 1 264000000
vdo_axi_sel --- 000000a2 0 0 264000000
vpu_axi_sel --- 000000a2 0 0 264000000
cko1_sel --- 00000002 0 0 528000000

ipg_per --- 00000004 1 1 66000000

ldb_dil_div_3_5 --- 00000024 1 1 104439560
ldb_dil_div_7 --- 00000024 0 0 52219788
ldb_dil_div_3_5 --- 00000024 0 0 104439560
ldb_dil_div_7 --- 00000024 0 0 52219788

gpu3d_core_podf --- 00000025 0 0 520000000
gpu3d_shader --- 00000025 0 0 594000000
ipu1_podf --- 00000025 0 0 264000000
ipu2_podf --- 00000025 1 1 264000000

4条已计数的行 -----

17:1 默认文本 PC 12:47 PM 重要的差异 插入 0.00 秒

8/8/2016 3:53:23 PM 21,754 字节 <默认> ANSI 1条已计数的行

ipu1_sel --- 000000a2 0 0 528000000
ipu2_sel --- 000000a2 1 1 528000000
ldb_dil0_sel --- 000000a4 1 1 360000000
ldb_dil1_sel --- 000000a4 0 0 360000000
ldb_dil0_div_sel --- 000000a6 1 1 102857142
ldb_dil1_div_sel --- 000000a6 0 0 51428571

hs1_tx_sel --- 000000a2 1 1 396000000
pcie_axi_sel --- 000000a2 0 0 264000000

ipu2_di0_sel --- 000000a6 1 1 102857142

usdhc1_sel --- 00000000 0 0 390000000
usdhc2_sel --- 00000000 0 0 390000000
usdhc3_sel --- 00000000 0 0 390000000
usdhc4_sel --- 00000000 0 0 390000000
enfc_sel --- 000000a2 0 0 390000000
emi_sel --- 00000000 0 0 390000000
emi_slow_sel --- 00000000 1 1 264000000
vdo_axi_sel --- 000000a2 0 0 264000000
vpu_axi_sel --- 000000a2 0 0 264000000
cko1_sel --- 000000a2 0 0 528000000

ipg_per --- 00000004 1 1 65000000

ldb_dil0_div_3_5 --- 00000024 1 1 102857142
ldb_dil0_div_7 --- 00000024 0 0 51428571
ldb_dil1_div_3_5 --- 00000024 0 0 102857142
ldb_dil1_div_7 --- 00000024 0 0 51428571

gpu3d_core_podf --- 00000025 0 0 520000000
gpu3d_shader --- 00000025 0 0 585000000
ipu1_podf --- 00000025 0 0 264000000
ipu2_podf --- 00000025 1 1 264000000

4条已计数的行 -----

17:1 默认文本 PC 12:47 PM 重要的差异 插入 0.00 秒

test2.txt <-> test3.txt - 文本比较 - Beyond Compare

会话(5) 文件(E) 编辑(E) 搜索(V) 视图(V) 工具(I) 帮助(H)

C:\Users\r64710\Desktop\test2.txt C:\Users\r64710\Desktop\test3.txt

8/8/2016 1:50:16 PM 21,795 字节 <默认> ANSI 10条已计数的行

cko1_sel --- 000000a2 0 0 528000000
ipg_per --- 00000004 1 1 66000000

ldb_dil0_div_3_5 --- 00000024 1 1 104439560
ldb_dil0_div_7 --- 00000024 0 0 52219788
ldb_dil1_div_3_5 --- 00000024 0 0 104439560
ldb_dil1_div_7 --- 00000024 0 0 52219788

gpu3d_core_podf --- 00000025 0 0 520000000
gpu3d_shader --- 00000025 0 0 594000000
ipu1_podf --- 00000025 0 0 264000000
ipu2_podf --- 00000025 1 1 264000000

4条已计数的行 -----

hs1_tx_podf --- 00000025 1 1 198000000

usdhc1_podf --- 00000025 0 0 198000000
usdhc2_podf --- 00000025 0 0 198000000
usdhc3_podf --- 00000025 0 0 198000000
usdhc4_podf --- 00000025 0 0 198000000
enfc_pred --- 00000025 0 0 79200000
enfc_podf --- 00000025 0 0 198000000
emi_podf --- 00000004 0 0 198000000
emi_slow_podf --- 00000004 1 1 132000000
vpu_axi_podf --- 00000025 0 0 264000000
cko1_podf --- 00000025 0 0 66000000

apbh_dma --- 00000005 0 0 198000000

asrc_ipg --- 00000005 0 0 132000000
asrc_mem --- 00000005 0 0 132000000
caam_mem --- 00000005 1 1 132000000
caam_clk --- 00000005 1 1 132000000
caam_ipg --- 00000005 1 1 66000000
can1_ipg --- 00000005 0 0 65000000

6条已计数的行 -----

17:1 默认文本 PC 12:48 PM 重要的差异 插入 0.00 秒

8/8/2016 3:53:23 PM 21,754 字节 <默认> ANSI 10条已计数的行

cko1_sel --- 000000a2 0 0 528000000
ipg_per --- 00000004 1 1 65000000

ldb_dil0_div_3_5 --- 00000024 1 1 102857142
ldb_dil0_div_7 --- 00000024 0 0 51428571
ldb_dil1_div_3_5 --- 00000024 0 0 102857142
ldb_dil1_div_7 --- 00000024 0 0 51428571

gpu3d_core_podf --- 00000025 0 0 520000000
gpu3d_shader --- 00000025 0 0 585000000
ipu1_podf --- 00000025 0 0 264000000
ipu2_podf --- 00000025 1 1 264000000

4条已计数的行 -----

hs1_tx_podf --- 00000025 1 1 195000000

usdhc1_podf --- 00000025 0 0 195000000
usdhc2_podf --- 00000025 0 0 195000000
usdhc3_podf --- 00000025 0 0 195000000
usdhc4_podf --- 00000025 0 0 195000000
enfc_pred --- 00000025 0 0 78000000
enfc_podf --- 00000025 0 0 195000000
emi_podf --- 00000004 0 0 195000000
emi_slow_podf --- 00000004 1 1 130000000
vpu_axi_podf --- 00000025 0 0 264000000
cko1_podf --- 00000025 0 0 65000000

apbh_dma --- 00000005 0 0 195000000

asrc_ipg --- 00000005 0 0 130000000
asrc_mem --- 00000005 0 0 130000000
caam_mem --- 00000005 1 1 130000000
caam_clk --- 00000005 1 1 130000000
caam_ipg --- 00000005 1 1 65000000
can1_ipg --- 00000005 0 0 65000000

6条已计数的行 -----

17:1 默认文本 PC 12:48 PM 重要的差异 插入 0.00 秒

i.MX6Q Plugin启动方式典型应用：改频与展频

test2.txt <--> test3.txt - Beyond Compare

会话(S) 文件(E) 编辑(E) 搜索(B) 视图(V) 工具(I) 帮助(H)

C:\Users\l64710\Desktop\test2.txt C:\Users\l64710\Desktop\test3.txt

8/8/2016 1:50:16 PM 21,795 字节 <默认> ANSI

8/8/2016 3:53:23 PM 21,754 字节 <默认> ANSI

```

can2_ipg --- 1条已计数的行 00000005 0 0 65000000
dcic1 --- 1条已计数的行 00000005 0 0 264000000
dcic2 --- 5条已计数的行 00000005 0 0 264000000
enet --- 5条已计数的行 00000005 2 2 65000000
esai_ipg --- 1条已计数的行 00000005 0 0 132000000
esai_mem --- 00000005 0 0 132000000
gpt_ipg --- 00000005 1 1 65000000
random: nonblocking pool is initialized
gpt_ipg_per --- 1条已计数的行 00000005 0 0 65000000
gpu3d_core --- 00000005 0 0 520000000
hdmi_iahb --- 00000005 1 1 132000000
i2c1 --- 1条已计数的行 00000005 0 0 65000000
i2c2 --- 00000005 0 0 65000000
i2c3 --- 00000005 0 0 65000000
iim --- 00000005 0 0 65000000
enfc --- 00000005 0 0 198000000
vdoa --- 00000005 0 0 264000000
ipu2 --- 00000005 1 1 104439560
ipu2_di0 --- 00000005 1 1 104439560
ldb_di0 --- 00000005 1 1 102857142
ldb_dil --- 00000005 0 0 52219780
hs1_tx --- 1条已计数的行 00000005 2 2 198000000
mlb --- 00000005 0 0 264000000
ocram --- 00000005 2 2 132000000
openvg_axi --- 00000005 0 0 264000000
pcie_axi --- 00000005 0 0 264000000
per1_bch --- 00000005 0 0 198000000
pwm1 --- 00000005 1 1 65000000
pwm2 --- 00000005 0 0 65000000
17:1 默认文本 PC 插入 0.00 秒
```



```

can2_ipg --- 1条已计数的行 00000005 0 0 65000000
dcic1 --- 1条已计数的行 00000005 0 0 260000000
dcic2 --- 5条已计数的行 00000005 0 0 260000000
enet --- 5条已计数的行 00000005 2 2 65000000
esai_ipg --- 1条已计数的行 00000005 0 0 130000000
esai_mem --- 00000005 0 0 130000000
gpt_ipg --- 00000005 1 1 65000000
gpt_ipg_per --- 1条已计数的行 00000005 0 0 65000000
gpu3d_core --- 00000005 0 0 520000000
hdmi_iahb --- 00000005 1 1 130000000
i2c1 --- 1条已计数的行 00000005 0 0 65000000
i2c2 --- 00000005 0 0 65000000
i2c3 --- 00000005 0 0 65000000
iim --- 00000005 0 0 65000000
enfc --- 00000005 0 0 195000000
vdoa --- 00000005 0 0 260000000
ipu2 --- 00000005 1 1 102857142
ipu2_di0 --- 00000005 1 1 102857142
ldb_di0 --- 00000005 1 1 102857142
ldb_dil --- 00000005 0 0 51428571
hs1_tx --- 1条已计数的行 00000005 1 1 195000000
mlb --- 00000005 0 0 260000000
ocram --- 00000005 2 2 130000000
openvg_axi --- 00000005 0 0 260000000
pcie_axi --- 00000005 0 0 260000000
per1_bch --- 00000005 0 0 195000000
pwm1 --- 00000005 1 1 65000000
pwm2 --- 00000005 0 0 65000000
17:1 默认文本 PC 插入 0.00 秒
```


test2.txt <--> test3.txt - Beyond Compare

会话(S) 文件(E) 编辑(E) 搜索(B) 视图(V) 工具(I) 帮助(H)

C:\Users\l64710\Desktop\test2.txt C:\Users\l64710\Desktop\test3.txt

8/8/2016 1:50:16 PM 21,795 字节 <默认> ANSI

8/8/2016 3:53:23 PM 21,754 字节 <默认> ANSI

```

pwm2 --- 00000005 0 0 65000000
pwm3 --- 00000005 0 0 65000000
pwm4 --- 00000005 0 0 65000000
gpmi_bch_app --- 00000005 0 0 198000000
gpmi_bch --- 00000005 0 0 198000000
gpmi_io --- 00000005 0 0 198000000
gpmi_apb --- 00000005 0 0 198000000
rom --- 00000005 1 1 132000000
sata --- 00000005 1 1 132000000
sdma --- 00000005 8 8 132000000
spba --- 00000005 0 0 65000000
spdif_gclk --- 1条已计数的行 00000005 0 0 65000000
ssi1_ipg --- 00000005 0 0 65000000
ssi2_ipg --- 00000005 0 0 65000000
ssi3_ipg --- 00000005 0 0 65000000
uart_ipg --- 1条已计数的行 00000005 1 1 65000000
usbh3 --- 00000005 0 0 65000000
usdhc1 --- 00000005 0 0 198000000
usdhc2 --- 00000005 0 0 198000000
usdhc3 --- 00000005 0 0 198000000
usdhc4 --- 00000005 0 0 198000000
eim_slow --- 00000005 2 2 132000000
vdo_axi --- 00000005 0 0 264000000
vpu_axi --- 00000005 0 0 264000000
cko1 --- 00000024 0 0 65000000
ipu1_pclk0_sel --- 1条已计数的行 00000000 0 0 264000000
ipu1_pclk1_sel --- 00000000 0 0 264000000
ipu2_pclk0_sel --- 4条已计数的行 00000000 1 1 104439560
ipu2_pclk1_sel --- 00000000 0 0 264000000
ipu2_pclk0_div --- 00000004 1 1 104439560
ipu2_pclk0 --- 1条已计数的行 00000004 1 1 104439560
17:1 默认文本 PC 插入 0.00 秒
```



```

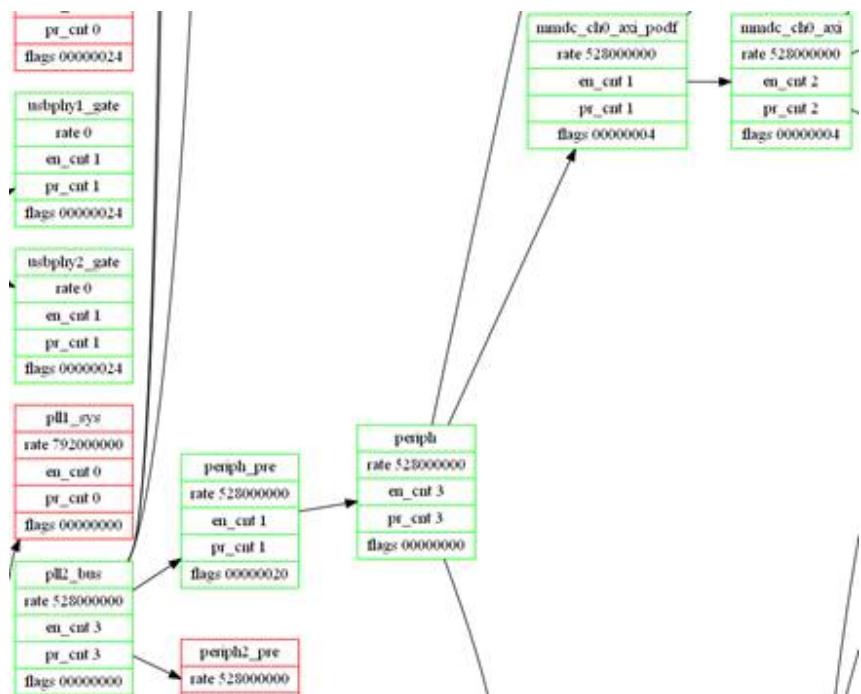
pwm2 --- 00000005 0 0 65000000
pwm3 --- 00000005 0 0 65000000
pwm4 --- 00000005 0 0 65000000
gpmi_bch_app --- 00000005 0 0 195000000
gpmi_bch --- 00000005 0 0 195000000
gpmi_io --- 00000005 0 0 195000000
gpmi_apb --- 00000005 0 0 195000000
rom --- 00000005 1 1 130000000
sata --- 00000005 1 1 130000000
sdma --- 00000005 8 8 130000000
spba --- 00000005 0 0 65000000
spdif_gclk --- 1条已计数的行 00000005 0 0 65000000
ssi1_ipg --- 00000005 0 0 65000000
ssi2_ipg --- 00000005 0 0 65000000
ssi3_ipg --- 00000005 0 0 65000000
uart_ipg --- 1条已计数的行 00000005 1 1 65000000
usbh3 --- 00000005 0 0 65000000
usdhc1 --- 00000005 0 0 195000000
usdhc2 --- 00000005 0 0 195000000
usdhc3 --- 00000005 0 0 195000000
usdhc4 --- 00000005 0 0 195000000
eim_slow --- 00000005 2 2 130000000
vdo_axi --- 00000005 0 0 260000000
vpu_axi --- 00000005 0 0 260000000
cko1 --- 00000024 0 0 65000000
ipu1_pclk0_sel --- 1条已计数的行 00000000 0 0 260000000
ipu1_pclk1_sel --- 00000000 0 0 260000000
ipu2_pclk0_sel --- 4条已计数的行 00000000 1 1 102857142
ipu2_pclk1_sel --- 00000000 0 0 260000000
ipu2_pclk0_div --- 00000004 1 1 102857142
ipu2_pclk0 --- 1条已计数的行 00000004 1 1 102857142
17:1 默认文本 PC 插入 0.00 秒
```

i.MX6Q Plugin启动方式典型应用：改频与展频

7 i.MX6DL 支持说明

i.MX6Q 与 i.MX6D 是同样的芯片，而 i.MX6DL 与 i.MX6S 的情况如下：

与 i.MX6Q/D 不同，i.MX6DL/S 仅支持 400MHz DDR 时钟频率，如下为 i.MX6DL 的 DDR 时钟源：



所以 MMDC0 来源于 periph，而 periph 来源于 periph_pre，MMDC1 来源于 periph2，而 periph2 来源于 periph2_pre。而在 i.MX6Q 中，都来源于 PLL2_bus。

如下 i.MX6DL rom 代码，其中 DDR 的时钟切换为 pll2_pfd2_396m:

```
hapi_clock_init
...
if (HAPI_SRC_SBMR_BT_FREQ0 == 1)
{
}
else
{
...
}
```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

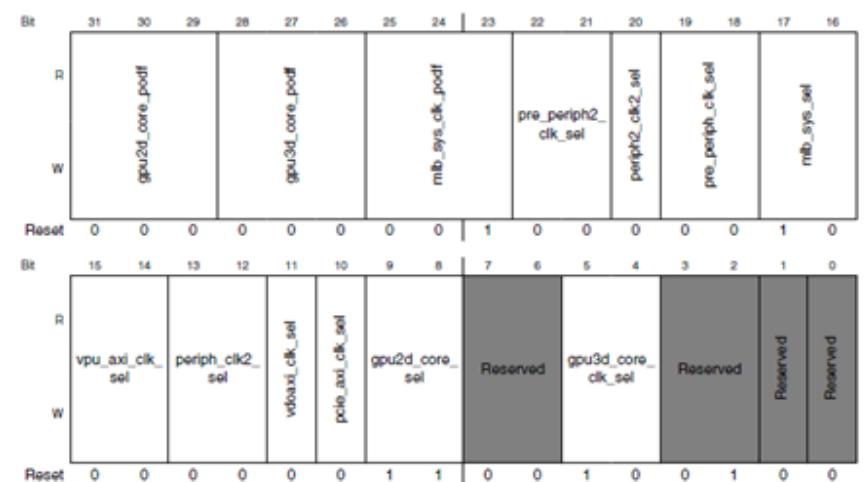
    hapi_clock_reg.cbcmr &= ~(CBCMR_PRE_PERIPH_CLK_SEL_MASK);
    hapi_clock_reg.cbcmr |= CBCMR_PRE_PERIPH_CLK_SEL_PFD400;

    ...
    hapi_clock_reg.cbcmr &= ~(CBCMR_PRE_PERIPH2_CLK_SEL_MASK);
    hapi_clock_reg.cbcmr |= CBCMR_PRE_PERIPH2_CLK_SEL_PFD400;
    ...
}

//#define CBCMR_PRE_PERIPH_CLK_SEL_PFD400 0x000040000
//#define CBCMR_PRE_PERIPH2_CLK_SEL_PFD400 0x002000000

```

Address: 20C_4000h base + 18h offset = 20C_4018h



22-21 pre_periph2_clk_sel	Selector for pre_periph2 clock multiplexer
00	derive clock from PLL2 main 528MHz clock
01	derive clock from 396MHz PLL2 PFD
10	derive clock from 307M PFD
11	derive clock from 198MHz clock (divided 396MHz PLL2 PFD)

00 Selector for pre_periph2 clk2 clock multiplexer

19-18 pre_periph_clk_sel	Selector for pre_periph clock multiplexer
00	derive clock from PLL2 main 528MHz clock
01	derive clock from 396MHz PLL2 PFD
10	derive clock from 307M PFD
11	derive clock from 198MHz clock (divided 396MHz PLL2 PFD)

i.MX6Q Plugin启动方式典型应用：改频与展频

以下为 rom 代码将 DDR 时钟从 pll2 528MHz 更变为 pll2_pfd2_396MHz 的说明,而 pll2_pfd2_396MHz 依然源自于 PLL2, 所以也会相应展频。

而另一方面, GPS 频率范围是:
(L1 :1575.42 +/-10 MHz 1565.42~1585.42).

所以如果 PLL2 改频为 520MHz. 则 pll2_pfd0 变化为

PLL2_PFD2 = 528MHz * 18 / 24 = 396MHz

PLL2_pfd2=520Mhz*18/24=390Mhz. 的以其四倍频为 4*390Mhz=1560, 仍然不在 GPS 频率范围内。

另外注意一下:

Arch\arm\mach-imx\clk-imx6q.c

```
/* ipu clock initialization */

imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_DI0_PRE_SEL], clk[IMX6QDL_CLK_PLL5_VIDEO_DIV]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_DI1_PRE_SEL], clk[IMX6QDL_CLK_PLL5_VIDEO_DIV]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_DI0_PRE_SEL], clk[IMX6QDL_CLK_PLL5_VIDEO_DIV]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_DI1_PRE_SEL], clk[IMX6QDL_CLK_PLL5_VIDEO_DIV]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_DI0_SEL], clk[IMX6QDL_CLK_IPU1_DI0_PRE]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_DI1_SEL], clk[IMX6QDL_CLK_IPU1_DI1_PRE]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_DI0_SEL], clk[IMX6QDL_CLK_IPU2_DI0_PRE]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_DI1_SEL], clk[IMX6QDL_CLK_IPU2_DI1_PRE]);
if (cpu_is_imx6dl()) {
    imx_clk_set_rate(clk[IMX6QDL_CLK_PLL3_PFD1_540M], 540000000);
    imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_SEL], clk[IMX6QDL_CLK_PLL3_PFD1_540M]);
    imx_clk_set_parent(clk[IMX6QDL_CLK_AXI_ALT_SEL], clk[IMX6QDL_CLK_PLL3_PFD1_540M]);
    imx_clk_set_parent(clk[IMX6QDL_CLK_AXI_SEL], clk[IMX6QDL_CLK_AXI_ALT_SEL]);
    /* set epdc/pxp axi clock to 200Mhz */
    imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_SEL], clk[IMX6QDL_CLK_PLL2_PFD2_396M]);
    imx_clk_set_rate(clk[IMX6QDL_CLK_IPU2], 200000000);
} else if (cpu_is_imx6q()) {
    imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_SEL], clk[IMX6QDL_CLK_MMDC_CH0_AXI]);
    imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_SEL], clk[IMX6QDL_CLK_MMDC_CH0_AXI]);
    /* set eim_slow to 132Mhz */
```

i.MX6Q Plugin启动方式典型应用：改频与展频

```
imx_clk_set_rate(clk[IMX6QDL_CLK_EIM_SLOW], 132000000);
```

所以默认 i.MX6Q 的 IPU root clock 来自于 IMX6QDL_CLK_MMDC_CH0_AXI 528Mhz。而 i.MX6DL 的 ipu root clock 来自于 IMX6QDL_CLK_PLL3_PFD1_540M。（这个可以理解，因为 i.MX6DL 的 DDR clock 只有 400MHz。所以如果 IPU root clock 也来自于 DDR clock，就太低了，所以需要切换到一个高一点的频率）而 DI clock 都来自于 IMX6QDL_CLK_PLL5_VIDEO_DIV。

那么如果我们希望 IPU 显示的 DI clock，也就是 Pixel clock 也要展频，对 i.MX6Q 来讲，就需要设置 DI clock 来自于 IPU 的 root clock。这个前面已经讲到了。

但是对于 i.MX6DL，这样显然就不行，因为 i.MX6DL 的 root clock 是 PLL3，而不是 DDR clock，也就是源自 PLL2。

所以我们的做法是，仍然使用 external clock 源，然后把 BSP 默认的 external clock 源从 IMX6QDL_CLK_PLL5_VIDEO_DIV 改成 PLL2.

如下 patch:

But if it is parallel LCD or HDMI, the customer should change the code in clk-imx6q.c

For example:

```
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_DI0_PRE_SEL], clk[IMX6QDL_CLK_PLL2_PFD0_352M]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU1_DI1_PRE_SEL], clk[IMX6QDL_CLK_PLL5_VIDEO_DIV]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_DI0_PRE_SEL], clk[IMX6QDL_CLK_PLL5_VIDEO_DIV]);
imx_clk_set_parent(clk[IMX6QDL_CLK_IPU2_DI1_PRE_SEL], clk[IMX6QDL_CLK_PLL5_VIDEO_DIV]);
```

这样就 DI clock 就来自于 PLL2 了。

然后在我们的 DI clock 设置代码中：

Drivers\mxc\ipu3\ipu_disp.c

```
ipu_init_sync_panel
{
...
    if (!strcmp(__clk_get_name(di_parent), __clk_get_name(ldb_di0_clk)) ||
        !strcmp(__clk_get_name(di_parent), __clk_get_name(ldb_di1_clk))) {
        /* if di clk parent is tve/ldb, then keep it; */
        ...
    } else {
        /* try ipu clk first*/ //BSP 默认先使用 ipu root clock 做为 ipu di clock 源。
        dev_dbg(ipu->dev, "try ipu internal clk\n");
        ret = clk_set_parent(ipu->pixel_clk_sel[disp], ipu->ipu_clk);
        if (ret) {
            dev_err(ipu->dev, "set pixel clk error:%d\n", ret);
        }
    }
}
```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

        return ret;
    }

    rounded_pixel_clk = clk_round_rate(ipu->pixel_clk[disp], pixel_clk);
    dev_dbg(ipu->dev, "rounded pix clk:%d\n", rounded_pixel_clk);
    /*
     * we will only use 1/2 fraction for ipu clk,
     * so if the clk rate is not fit, try ext clk.
     */
    if (!sig.int_clk && //如果使用 external clock,也就是
/*
Kernel\arch\arm\boot\dts\imx6qdl-sabresd.dtsi
mxcfb3: fb@2 {
...
int_clk = <0>; // int_clk=1 则 pixel clock 来源于 ipu root clock, 而不是 PLL5
...
};

*/

```

//的情况下，如果 external clock 源除下来的 pixel clock 足够精确，则使用 external clock. 精确度在+-1/200 中。

```

((rounded_pixel_clk >= pixel_clk + pixel_clk/200) ||
(rounded_pixel_clk <= pixel_clk - pixel_clk/200)) {
    dev_dbg(ipu->dev, "try ipu ext di clk\n");

    rounded_pixel_clk =
        clk_round_rate(ipu->di_clk[disp], pixel_clk);
    ret = clk_set_rate(ipu->di_clk[disp],
        rounded_pixel_clk);
    if (ret) {
        dev_err(ipu->dev,
            "set di clk rate error:%d\n", ret);
        return ret;
    }
    dev_dbg(ipu->dev, "di clk:%d\n", rounded_pixel_clk);
    ret = clk_set_parent(ipu->pixel_clk_sel[disp],
        ipu->di_clk[disp]);
    if (ret) {


```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

        dev_err(ipu->dev,
                "set pixel clk parent error:%d\n", ret);
        return ret;
    }
}
}
}

```

所以在 i.MX6DL 中我们是希望使用 external clock,但是因为是使用的 IMX6QDL_CLK_PLL2_PFD0_352M 而不是原来的 IMX6QDL_CLK_PLL5_VIDEO_DIV。所以很可能不会满足我们要求的+/-1/200 的条件，所以代码可能会退出而继续使用 internal clock。所以我们可以把这个判断去掉。

```

if (!sig.int_clk && /*((rounded_pixel_clk >= pixel_clk + pixel_clk/200) ||
    (rounded_pixel_clk <= pixel_clk - pixel_clk/200))*/ {

```

这样可能实际的 pixel clock 会和要求的有所差距，而一般的屏的 pixel clock 支持有一个范围，所以只要在范围之类，可以通过调节 H/V 的前后扫数值来匹配新 pixel clock 值。

8 3.0.35 uboot/bsp 支持

目前依然有一些客户在使用比较老久的 3.0.35_4.1.0_130816 ltib BSP，其 uboot 版本为：

u-boot-2009.08\ Makefile

VERSION = 2009

PATCHLEVEL = 08

本节说明如何在此 uboot 版本上支持改频与展频，我们可以将 L4.1.15_1.0.0 的 plugin 代码修改到 3.0.35 的 uboot 中去。前面说过需要参考的代码有：

board\freescale\mx6sabresd\plugin.S
arch\arm\include\asm\arch-mx6\mx6_plugin.S

其中 plugin 的主要代码是在 mx6_plugin.S 中，我们可以将相应代码修改到 3.0.35 中，注意 4.1.15_1.0.0 所使用的 2015.04 uboot 版本是使用工具来将 plugin 头或 dcd 头加在标准 uboot 前面的，这个与 2009.08 uboot 有所不同。所以最主要的工作是自己实现 plugin 的 ivt 启动 flash header。

DCD 只需要实现一个 IVT 头，在 ROM 代码使用 DCD 段的数据结构初始化外存后，ROM 代码再根据 IVT 定义的 uboot 镜像信息，把 uboot 镜像拷贝到外存中，所以 IVT 信息指向的地址是 uboot 镜像链接到外存的地址。

而 Plugin 需要实现两个 IVT 头，第二个用于告诉 ROM 代码如何拷贝 Uboot，与 DCD 的 IVT 头类似。但第一个 IVT 头是要将 Plugin 代码拷贝到内部 iRam 中运行，所以相关地址要指定为内部 iRam 地址，内部 iRam 的空余地址起点为 0x00907000，如下：

```

u-boot-2009.08/board/freescale/mx6q_sabresd/flash_header.S
#endif CONFIG_USE_PLUGIN /*johnli plugin*/

```

i.MX6Q Plugin 启动方式典型应用：改频与展频

```

#define IRAM_FREE_SPACE_START 0x00907000 /*rom codes will copy image to this address*/
/*rom codes会将ivt头开始的数据拷贝到此地址，所以从这儿可以找到ivt头*/
...
#endif
.section ".text.flasheader", "x"
    b      _start
    /*boot device type  image vectore table offset initial load region size*/
    /*sd/mmc/esd/emmc/sdxc/nand 1Kbyte=0x400 bytes 4Kbyte*/
    .org   CONFIG_FLASH_HEADER_OFFSET

ivt_header: .word 0x402000D1 /* Tag=0xD1, Len=0x0020, Ver=0x40 */
#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
app_code_jump_v: .word IRAM_FREE_SPACE_START+CONFIG_FLASH_HEADER_OFFSET+22*4 /* Plugin entry
point, address after the second IVT table //absolute address of the first instruction to execute from the image iRam中
plugin_start的地址，从IRAM_FREE_SPACE_START 开头偏移CONFIG_FLASH_HEADER_OFFSET， 再加两个ivt头
的大小，每个ivt中一个成员占4字节*/
#else
app_code_jump_v: .word _start
#endif
reserv1: .word 0x0
#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
dcd_ptr: .word 0 /*absolute address of the image DCDC, the DCD is optional so this field maybe set to NULL if no DCD
is required 不使用DCD启动，把此地址设置为0*/
#else
dcd_ptr: .word dcd_hdr
#endif
#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
boot_data_ptr: .word IRAM_FREE_SPACE_START+CONFIG_FLASH_HEADER_OFFSET+8*4 /*iRam中boot_data
的地址 */
self_ptr: .word IRAM_FREE_SPACE_START+CONFIG_FLASH_HEADER_OFFSET /* iRam中ivt_header的地址*/
#else
boot_data_ptr: .word boot_data
self_ptr: .word ivt_header
#endif
#endif CONFIG_SECURE_BOOT
app_code_csf: .word __hab_data
#else
app_code_csf: .word 0x0
#endif
reserv2: .word 0x0

#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
boot_data: .word IRAM_FREE_SPACE_START /*iRam中代码开始位置，包括ivt头*/
#else
boot_data: .word TEXT_BASE
#endif
#endif CONFIG_SECURE_BOOT
image_len: .word __hab_data_end - TEXT_BASE + CONFIG_FLASH_HEADER_OFFSET
#else
#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
image_len: .word MAX_PLUGIN_CODE_SIZE /* plugin can be upto 16KB in size plugin代码最大可以拷贝16KB*/
#else

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

image_len: .word _end_of_copy - TEXT_BASE + CONFIG_FLASH_HEADER_OFFSET
#endif
#endif
#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
plugin: .word 0x1 /* Enable plugin flag //Plugin flag plugin=1 means use plugin 表示使用plugin启动*/
#else
plugin: .word 0x0
#endif

#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
/* Second IVT to give entry point into the bootloader copied to DDR */
/*ivt header 2*/
ivt2_header: .word 0x402000D1 /*Tag=0xD1, Len=0x0020, Ver=0x40 (address is 0x0090742c)*/
app2_code_jump_v: .word _start /* Entry point for uboot 第二个ivt的跳转执行地址为_start*/
reserv3: .word 0x0
dcd2_ptr: .word 0x0 /*no dcd仍然无DCD*/
boot_data2_ptr: .word boot_data2
self_ptr2: .word ivt2_header
app_code_csf2: .word 0x0
reserv4: .word 0x0

/*boot data 2*/
boot_data2: .word TEXT_BASE
/*
//uboot/board/freescale/mx6q_sabreauto/config.mk TEXT_BASE = 0x27800000 which is ddr memory address
//uboot/u-boot.map
//Name Origin Length Attributes
/.text 0x0000000027800000 0x25e7c
// 0x000000002786cfb4 _end_of_copy = .
*/
image_len2: .word _end_of_copy - TEXT_BASE + CONFIG_FLASH_HEADER_OFFSET
plugin2: .word 0x0 /*plugin=0 means no plug in 此为uboot,不是plugin*/
/* Here starts the plugin code */
/*
The plugin function must follow the API described below:
*/
plugin_start:
.....
#else
/*DCD*/
#endif

```

之后， plugin_start 可以从 4.1.15_1.0.0 的 2015_04 uboot 中
arch\arm\include\asm\arch-mx6\mx6_plugin.S 拷贝过来：

```

plugin_start:
    push {r0-r4, lr}
    /*johnli add */
    #ifdef CONFIG_PLUGIN_ADD
        imx6_plugin_add
    #endif

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

#endif
    imx6_ddr_setting
    imx6_clock_gating
    imx6_qos_setting
...
/* return back to ROM code */
bx lr

/* make the following data right in the end of the output*/
.ltorg

/*
 * second_ivt_offset is the offset from the "second_ivt_header" to
 * "image_copy_start", which involves FLASH_OFFSET, plus the first
 * ivt_header, the plugin code size itself recorded by "ivt2_header"
 */
second_ivt_offset: .long (0x2C + CONFIG_FLASH_HEADER_OFFSET)
/*源文件中此处定义为 second_ivt_offset: .long (ivt2_header + 0x2C + FLASH_OFFSET), 由于
ivt2_header: .long 0x0。 而
#ifndef CONFIG_SYS_BOOT_EIMNOR || defined(CONFIG_SYS_BOOT_QSPI)
#define FLASH_OFFSET 0x1000
#else
#define FLASH_OFFSET 0x400
#endif
所以我们改为以上值 second_ivt_offset: .long (0x2C + CONFIG_FLASH_HEADER_OFFSET)
*/
#ifndef DCD
#endif

```

然后4.1.15_1.0.0的2015_04 uboot中arch\arm\include\asm\arch-mx6\mx6_plugin.S用到的一些宏常量要加上：

```

#ifndef CONFIG_USE_PLUGIN /*johnli plugin*/
#define IRAM_FREE_SPACE_START 0x00907000 /*rom codes will copy image to this address*/
#define MAX_PLUGIN_CODE_SIZE (16*1024) /*这个是单独增加的常量, 表示plugin要拷贝的大小*/
#ifndef CONFIG_ROM_UNIFIED_SECTIONS
#define ROM_API_TABLE_BASE_ADDR_LEGACY          0x180
#define ROM_VERSION_OFFSET                     0x80

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

#else
#define ROM_API_TABLE_BASE_ADDR_LEGACY 0xC0
#define ROM_VERSION_OFFSET 0x48
#endif
#define ROM_API_TABLE_BASE_ADDR_MX6DQ_TO15 0xC4
#define ROM_API_TABLE_BASE_ADDR_MX6DL_TO12 0xC4
#define ROM_API_HWCNFG_SETUP_OFFSET 0x08
#define ROM_VERSION_TO10 0x10
#define ROM_VERSION_TO12 0x12
#define ROM_VERSION_TO15 0x15

```

再后， plugin 代码调用 的代码需要从 board\freescale\mx6sabresd\plugin.S 中拷贝过来，要放在 plugin 的代码前面，来保证编译通过：

```

.macro imx6_pll2_520
.macro imx6_pll2_6m_ssc
.macro imx6_pll2_24m_ssc
.macro imx6_pll2_clk01
.macro imx6_plugin_add
.macro imx6dqpssabresd_ddr_setting
...
.macro imx6_clock_gating
.macro imx6_qos_setting
.macro imx6_ddr_setting

```

最后在配置文件中加上编译宏：

Include\configs\mx6q_sabresd.h

```

/* uncomment for PLUGIN mode support */
/* #define CONFIG_USE_PLUGIN */

#define CONFIG_USE_PLUGIN /*johnli enable plugin*/
#ifndef CONFIG_USE_PLUGIN
#define CONFIG_PLUGIN_ADD /*johnli test it*/
#ifndef CONFIG_PLUGIN_ADD
#define CONFIG_PLL2_CLK01
/*#define CONFIG_PLL2_520 */
/* #define CONFIG_PLL2_6M_SSC */
#ifndef CONFIG_PLL2_6M_SSC
#define CONFIG_PLL2_520_6M_SSC
#endif
#define CONFIG_PLL2_24M_SSC
#ifndef CONFIG_PLL2_24M_SSC
#define CONFIG_PLL2_520_24M_SSC

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

#endif
#endif
#endif
/*johnli end */

```

3.0.35 的 uboot 2009_04 版本对 PLL2 并没有考虑会改动频率，所以如果我们改成了 520MHz，需要修改其中的 hard coding 的代码，如下：

```

cpu\arm_cortexa8\mx6\generic.c
#ifndef JOHNLI_FC
//pll2由528Mhz改成会520Mhz后，相应的PFD都会发生变化，所以下写死的值都需要相应修改
#define PLL2_PFD0_FREQ 346666666
#define PLL2_PFD1_FREQ 585000000
#define PLL2_PFD2_FREQ 390000000
#define PLL2_PFD2_DIV_FREQ 195000000

#else
#define PLL2_PFD0_FREQ 352000000
#define PLL2_PFD1_FREQ 594000000
#define PLL2_PFD2_FREQ 396000000
#define PLL2_PFD2_DIV_FREQ 198000000
#endif

...
#endif CONFIG_CMD_CLOCK
...

#define PLL2_FREQ_MAX 520000000 /*johnli for fc528000000 */

...
//__decode_pll 函数，原本只考虑了 528MHz 的情况，现在要对 520Mhz 的情况做修改，注意一下由于变量是整数，所以对除法要先乘分子，再除分母，以避免产生小数，另外就是要用双精度整数，以防止越界。
static u32 __decode_pll(enum pll_clocks pll, u32 infreq)
{
    u32 div;
#ifndef JOHNLI_FC
    u32 num;
    u32 denom;
    double rel;
    double infreq_double;
    infreq_double=infreq;
#endif

case BUS_PLL2:
    div = REG_RD(ANATOP_BASE_ADDR, HW_ANADIG_PLL_528) &
          BM_ANADIG_PLL_528_DIV_SELECT;
    if(1 == div)
    {
        return infreq * (20 + (div << 1));
    }
}

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

        }
    else
    {
        num = REG_RD(ANATOP_BASE_ADDR, HW_ANADIG_PLL_528_NUM) &
        0x3FFFFFFF;
        denom = REG_RD(ANATOP_BASE_ADDR, HW_ANADIG_PLL_528_DENOM) &
        0x3FFFFFFF;
        rel=(infreq_double * (((20 + (div << 1))*denom)+num))/denom;
        return rel;
    }
}

```

特别需要注意以下初始化函数调用：

```

start_armboot
|-> stdio_init
| | |-> drv_lcd_init
| | | |-> lcd_init
| | | | |-> lcd_enable
board\freescale\mx6q_sabresd\mx6q_sabresd.c

```

原生代码本身是设置了 PLL2 的 divider 值，对于 528MHz 时，divider 值本来就是 1,所以不会产生问题，而在 520MHz 的情况下，这个值被 plugin 代码改成了 0，所以如果这儿强制设 1,会使 PLL2 时钟在运行时变化成不正确的值，导致 uboot 死机，所以需要去掉。

```

void lcd_enable(void)
{
#if defined CONFIG_MX6Q
...
#endif //johnli for fc
    writel(0x1, ANATOP_BASE_ADDR + 0x34);
#endif
...
#endif //elif defined CONFIG_MX6DL /* CONFIG_MX6Q */
...
#endif //if 0 //johnli for fc
    writel(0x1, ANATOP_BASE_ADDR + 0x34);
#endif
...
#endif
}
```

i. MX6Q Plugin启动方式典型应用：改频与展频

另如第 6 章所说，如果将 PLL2 修改为了 520MHz。则对内核的修改如下：

Arch\arm\mach-mx6\crm_regs.h

```
/* PLL2_528 defines */
#define ANADIG_PLL_528_DIV_SELECT          (1)
#if 1 //johnli for fc
#define ANADIG_PLL_528_NUM_SELECT          (0xFFFFFFFF)
#define ANADIG_PLL_528_DENOM_SELECT        (0xFFFFFFFF)
#endif
```

Arch\arm\mach-mx6\clock.c

```
static unsigned long _clk_pll2_main_get_rate(struct clk *clk)
{
    unsigned int div;
    unsigned long val;
    #if 1 //johnli for fc
    unsigned long num=0;
    unsigned long denom=0;
    #endif
    div = __raw_readl(PLL2_528_BASE_ADDR) & ANADIG_PLL_528_DIV_SELECT;
    #if 1 //johnli for fc
    num= __raw_readl(PLL2_528_BASE_ADDR+PLL_528_NUM_DIV_OFFSET) &
ANADIG_PLL_528_NUM_SELECT;
    denom= __raw_readl(PLL2_528_BASE_ADDR+PLL_528_DENOM_DIV_OFFSET) &
ANADIG_PLL_528_DENOM_SELECT;;
    #endif
    if (div == 1)
        val = clk_get_rate(clk->parent) * 22;

    else
        val = clk_get_rate(clk->parent) * 20;
    #if 1 //johnli for fc
    //实际测试中发现此内核在双精度支持上出现了编译错或是计算 错的情况，所以此处根据plugin代码设置的值来直接
    //返回520MHz，请注意如果pluing设置值不同，要相应修改。
    if(((denom == 12)||(denom == 1200)||(denom == 4800))&&((num == 20)||(num == 2000)||(num == 8000)))
        {
        val=520000000;
        }
    #endif
    #if 0 //johnli for fc
```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

        if(denom!=0)
        {
            val=val+((clk_get_rate(clk->parent) * num)/denom);
        }
#endif
return val;
}

static int _clk_pll2_main_set_rate(struct clk *clk, unsigned long rate)
{
    unsigned int reg, div;
#ifndef johnli for fc
    unsigned int num=0;
    unsigned int denom=0;
#endif
    if (rate == 528000000)
        div = 1;
    else if (rate == 480000000)
        div = 0;
#ifndef johnli for fc
    else if (rate == 520000000)
    {
        div = 0;
#endif
//need sync with uboot setting, but this function should useless
        num=20;
        denom=12;
    }
#endif
    else
        return -EINVAL;
    reg = __raw_readl(PLL2_528_BASE_ADDR);
    reg &= ~ANADIG_PLL_528_DIV_SELECT;
    reg |= div;
    __raw_writel(reg, PLL2_528_BASE_ADDR);
#ifndef johnli for fc
    reg = __raw_readl(PLL2_528_BASE_ADDR+PLL_528_NUM_DIV_OFFSET);

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```

    reg &= ~ANADIG_PLL_528_NUM_SELECT;
    reg |= num;
    __raw_writel(reg, PLL2_528_BASE_ADDR+PLL_528_NUM_DIV_OFFSET);
    reg = __raw_readl(PLL2_528_BASE_ADDR+PLL_528_DENOM_DIV_OFFSET);
    reg &= ~ANADIG_PLL_528_DENOM_SELECT;
    reg |= denom;
    __raw_writel(reg, PLL2_528_BASE_ADDR+PLL_528_DENOM_DIV_OFFSET);
#endif
    return 0;
}

```

内核中 hard coding 的代码修改如下：

arch\arm\mach-mx6\clock.c

```

#if 1 //johnli for fc
clk_set_rate(&gpu3d_core_clk[0], 520000000);
#else
clk_set_rate(&gpu3d_core_clk[0], 528000000);
#endif

```

arch\arm\mach-mx6\bus_freq.c

```

#if 1 //johnli for fc
#define DDR3_NORMAL_CLK      520000000
#else
#define DDR3_NORMAL_CLK      528000000
#endif

```

arch\arm\mach-mx6\mx6_ddr_freq.S

```

#if 1 /*johnli for fc*/
ldr r1, =520000000
#else
ldr r1, =528000000
#endif

```

以下为测试验证结果：

uboot:

```

MX6Q SABRESD U-Boot > md.w 020c8030 1
020c8030: 2000 . //divider=0

```

i.MX6Q Plugin启动方式典型应用：改频与展频

```
MX6Q SABRESD U-Boot > md.w 020c8050 1
```

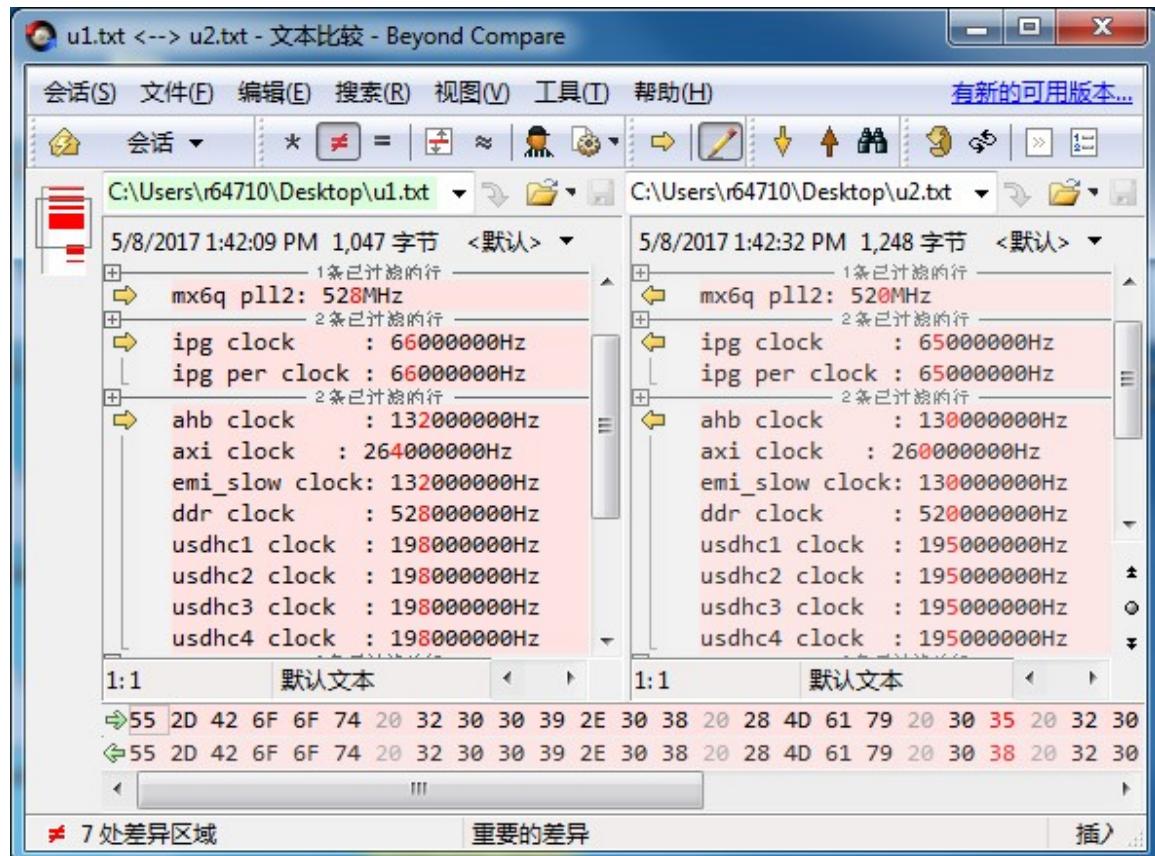
```
020c8050: 07d0 ...//num=2000
```

```
MX6Q SABRESD U-Boot > md.w 020c8060 1
```

```
020c8060: 04b0 ...//denom=1200
```

```
MX6Q SABRESD U-Boot >
```

加了 520MHz 改频和 24MHz 展频后的 uboot 打印区别如下：



kernel:

测试方法与第 6 章相同：使用/unit_test/dump-clocks.sh 命令，可以打印出所有的 clock 树。

i.MX6Q Plugin启动方式典型应用：改频与展频



i.MX6Q Plugin启动方式典型应用：改频与展频

