S32G Software Frequency Change and Spread Spectrum for EMI Optimization

by John Li (nxa08200)

Ver.	Comments	Author
V1	● Create the doc	● John.Li
V2	● Modified according to the comments of datasheet and chipset reference manual V4	● John.Li
V4	● Translate to English	● John.Li

The S32G supports the spread spectrum function of the internal accelerator PLL, ARM_PLL and DDR_PLL in addition to the peripheral PLL. Under normal circumstances, the PLL of the accelerator module and CPU does not need to spread spectrum, because it is an internal module, which is packaged inside the chip. The peripheral PLL does not support spread spectrum. Therefore, the method to actively improve EMI that can be provided is mainly the spread spectrum function of DDR_PLL, because it is possible for DDR to radiate into the space through, for example, termination resistors and surface traces.

In addition to the spread spectrum, it may also be necessary to adjust the frequency and the center point of its harmonics to avoid some sensitive frequencies such as navigation satellite signals, so it is also necessary to adjust the frequency of the DDR_PLL. In fact, because S32G only supports center spread spectrum, in order not to exceed the highest required frequency, it will first reduce the frequency and then spread the spectrum.

The default BSP of S32G initializes the DDRC controller through Uboot that has been loaded into the internal SRAM, so the above functions can be achieved by setting the relevant PLL registers through C code.

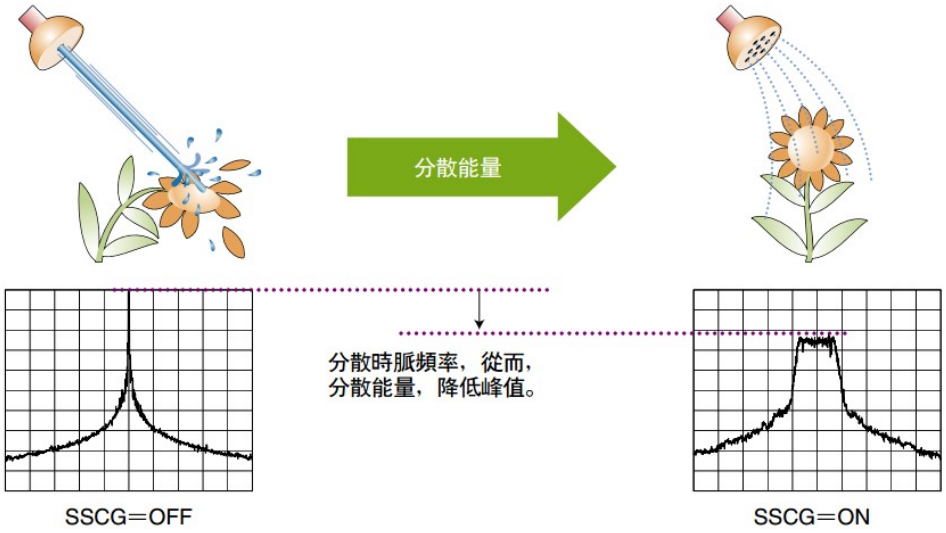
The self-designed bootloader may directly set the DDRC controller through the M7 code, and the related clock register settings can be set according to the method described in this doc.

Contents

1 The Basic Concept of Spread Spectrum

EMI, that is, electromagnetic interference, refers to the influence of the electronic circuit system on the surrounding electronic circuit system through conduction or radiation. The clock signal is often the signal with the highest frequency and the steepest edge in the circuit system, and most EMI problems are related to the clock signal. There are many ways to reduce EMI, including shielding, filtering, isolation, ferrite rings, signal edge control, and adding power and GND planes to the PCB. The above methods can be used flexibly in applications. Shielding is a relatively simple mechanical method with high cost and is not suitable for handheld and portable devices. Filtering and signal edge control are effective for low-frequency signals, but not suitable for high-speed signals that are currently widely used. In addition, the use of passive components such as EMI/RFI filters will increase the cost; it is obviously time-consuming to reduce EMI through the LAYOUT technique, and the means are not the same due to different designs.

Spread Spectrum Clocking (Spread Spectrum Clocking) is another method to effectively reduce EMI. Clock spreading disperses the energy concentrated in a narrow frequency range to a set wide frequency range by means of frequency modulation. and the amplitude (energy) of the odd harmonic frequency to achieve the purpose of reducing the peak value of the electromagnetic radiation of the system.



Frequency spreading modulates the original clock signal in a specific way. Linear and Hershey Kiss are commonly used modulations. Generally, only linear spread spectrum is supported on embedded chips.

SSCG achieves the purpose of suppressing EMI peaks by modulating the frequency of the internal integrated circuit of the clock. SSCG not only modulates the clock source, but other data, address and control signals synchronized with the clock source are also modulated at the same time as the clock spread, and the overall EMI peak will be reduced accordingly. Therefore, the clock spread is a system level. s solution. This is the biggest advantage of SSCG over other EMI suppression measures.

There are three main control parameters for clock spreading: Modulation Rate, Modulation Depth, and Modulation Profile.

1. Modulation speed

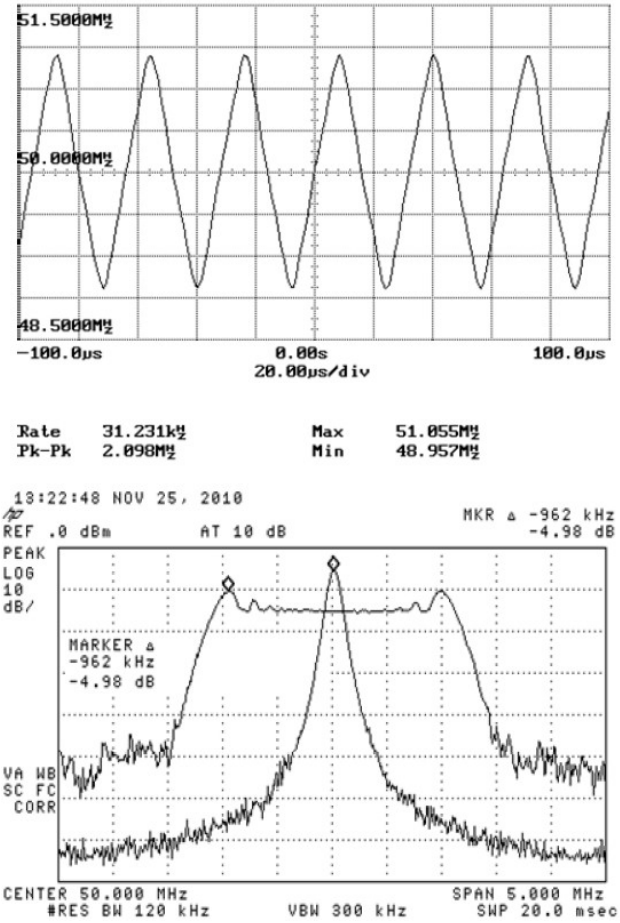
Modulation speed (MR) refers to the change speed of the output clock frequency fo within the set modulation frequency range. The modulation speed should be much smaller than the frequency fc of the source clock to avoid timing problems (setup/hold time, etc.), and should be higher than the frequency range (20Hz~20KHz) of the sound recognizable by the human ear to avoid noise. In practical applications, the modulation speed is generally selected from 30KHz to 120KHz.

2. Modulation depth

The modulation depth refers to the size by which the output clock frequency fo is shifted from the source clock frequency fc at the modulation speed MR after frequency spreading. The modulation depth is expressed as a percentage (%) of the offset (Δf) source clock frequency. The modulation depth determines how much the EMI peak is reduced. Generally, the greater the modulation depth, the lower the EMI peak. In application, it is necessary to reasonably predict the acceptable frequency modulation range of the system.

3. Modulation method

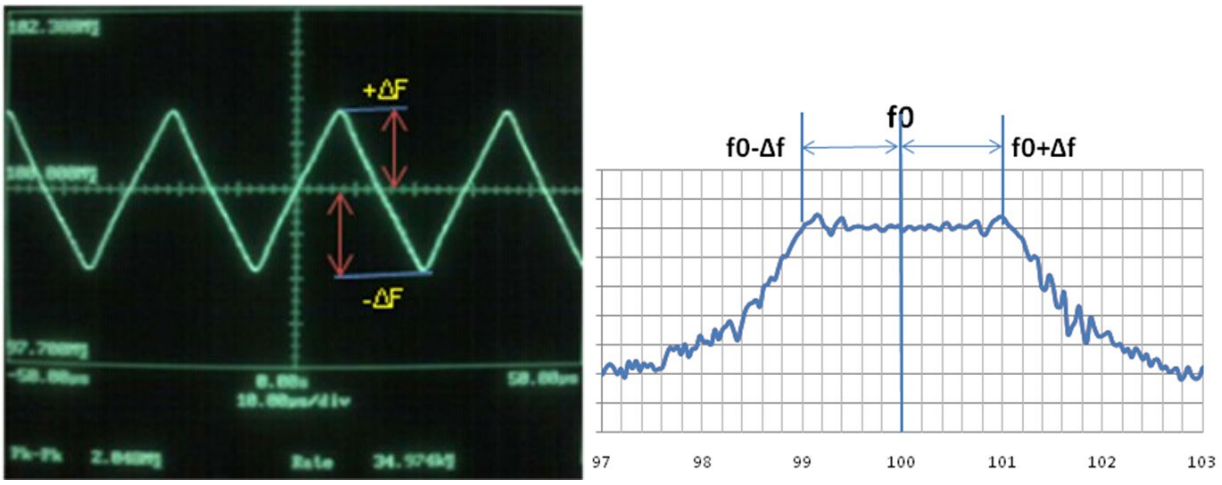
The modulation method (Modulation Profile) determines the manifestation of the EMI peak. Linear and Hershey Kiss are two modulations commonly used in SSCG. Linear modulation is relatively simple. As the name implies, the output clock frequency after linear modulation changes linearly. The disadvantage of this modulation method, as shown in the figure below, is that the output spectral side lobes are 1-2dB higher than the mid-frequency amplitude. As discussed earlier, the failure of EMI at any frequency means the failure of the entire EMI test. Sidelobe radiation peaks may exceed the SPEC range, the designer needs to take this into account



According to the difference of the offset of the spread spectrum clock relative to the source clock, the spread spectrum is divided into three categories: middle spread spectrum; downward spread spectrum; upward spread spectrum. The center spread refers to: the center spread refers to the spread spectrum mode in which the average frequency of the spread spectrum clock is the same as the frequency of the source clock. In a system where the unmodulated output clock frequency is equal to the input clock frequency, the output clock frequency fo after spread spectrum is determined by the frequency modulation method (Linear or Hershey Kiss) at the MR speed, in the range of (fc-Δf) to (fc+Δf) changes within:

$$fo = fc \pm \Delta f$$

For example, after a 100MHz clock is modulated at a depth of ±1% with mid-spread, the output clock varies from 99MHz to 101MHz.



A major disadvantage of clock spreading is that it cannot be used in applications that are sensitive to clock accuracy, such as Ethernet and CAN buses. When choosing clock spreading and modulation depth, designers need to pay special attention to the additional jitter introduced by spreading and possible setup/hold time issues, high bit error rates, and PLL loss of lock.

When the spread spectrum clock is output to the downstream PLL, note that the PLL behaves as a low-pass filter, that is, it allows the low-frequency part of the input to pass, while attenuating the high-frequency part. When the spread spectrum clock is input to the PLL, the PLL may not be able to lock the frequency. It is important to ensure that the PLL must be able to detect frequency changes of the spread spectrum clock and allow the spread spectrum clock to pass. The above depends on the bandwidth of the PLL. If the bandwidth is too low, the PLL may not be able to detect the input clock reliably, causing detection deviation and introducing a larger jitter to the system.

2 Git the Uboot Source Codes for Testing

Obtain the source code of Uboot and create a standalone compilation environment according to the document <<S32G_Uboot_BSPxx_Vx-xxxxxxxx.doc>>(Owner: JohnLi Chinese Version). This doc uses Uboot of BSP30.

Note that there is another way to initialize the DDRC controller with M7, and then put Uboot directly from the storage device in the external LPDDR4, instead of placing Uboot in the internal SRAM of the default BSP, in that case, the spread spectrum and frequency change code needs to be in the codes of M7, which is not in the scope of this doc. This doc discusses the method used by the default BSP, which is placed in the Uboot of the internal SRAM after startup to initialize the DDRC.

3 Change the DDR_PLL's Frequency

The purpose of frequency change is mainly to avoid overlapping with some sensitive signals such as navigation satellites, as follows:

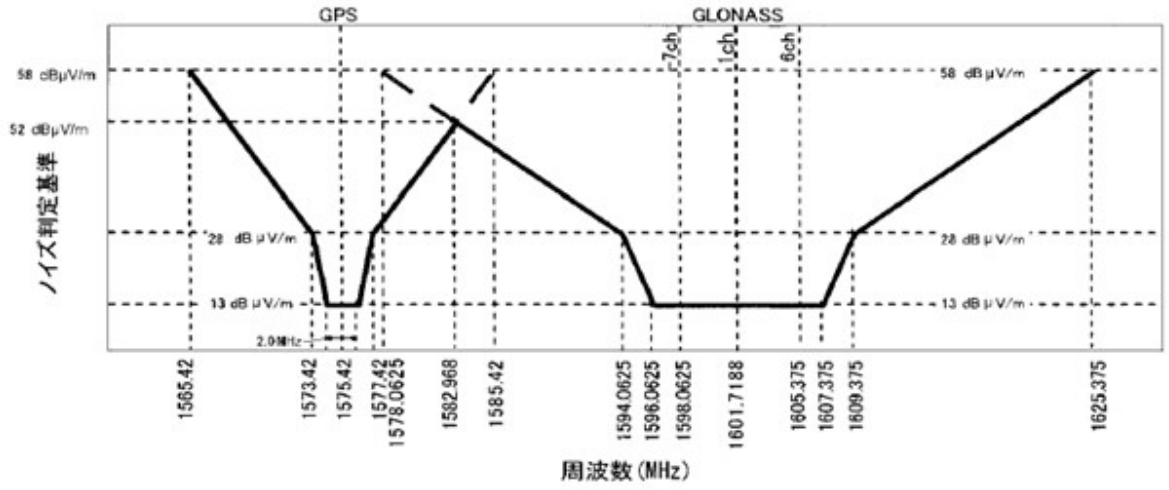


図3 GPS-GLONASS帯の目標値（電界アンテナ法 アベレージ検波）（補正値を用いた場合）

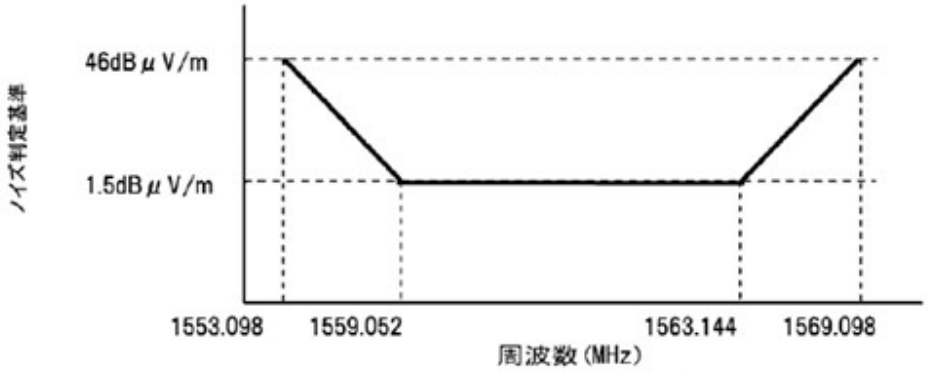


図4 中国BDS帯の目標値（電界アンテナ法 アベレージ検波）（補正値用いた場合）

- GPS frequency band is:(1575.42 +/-10 MHz 1565.42~1585.42)。
- BeiDou frequency band is: (1561.098 +/-8 MHz 1553.098~1569.098)。
- GLONASS frequency band is: (1578.0625~1625.375)。

It can be seen that Beidou, GPS and grid Glonass are coincident and cover the frequency band from 1553 to 1625 (the edge of the inverted trapezoid has some triangular areas). So if you want to avoid this frequency band perfectly, the DDR clock needs to be set to be smaller than 1553, (larger is generally not supported, exceeding the maximum frequency required by the spec).

The crystal frequency of S32G is 40M, and the default DDR clock is 40X40=1600Mhz, so it needs to be changed to 40X38=1520Mhz (note that only the integer multiplication is considered here, if you add the decimal multiplication, it can be closer to 1553M , but considering that after the frequency is changed, there are often requirements for spreading, so it is more reasonable to take the integer multiplier. In fact, in Uboot, the current BSP30 code support for fractional multipliers still has problems).

In addition, please note that the spectrum requirement of the satellite signal is an inverted trapezoid, so there is a certain space between the spectrum requirements of each positioning satellite signal, and the frequency change and spread spectrum can be adjusted to meet the requirements closer to 1600Mhz.

So the DDR frequency can be modified as follows:

```
#define S32G274A_REV2_FC_DDR_PLL_VCO_FREQ                (1520 * MHZ)
#define S32G274A_REV2_FC_DDR_FREQ                        (760 * MHZ)
```

After modification and compilation, the following test results after Uboot starts:

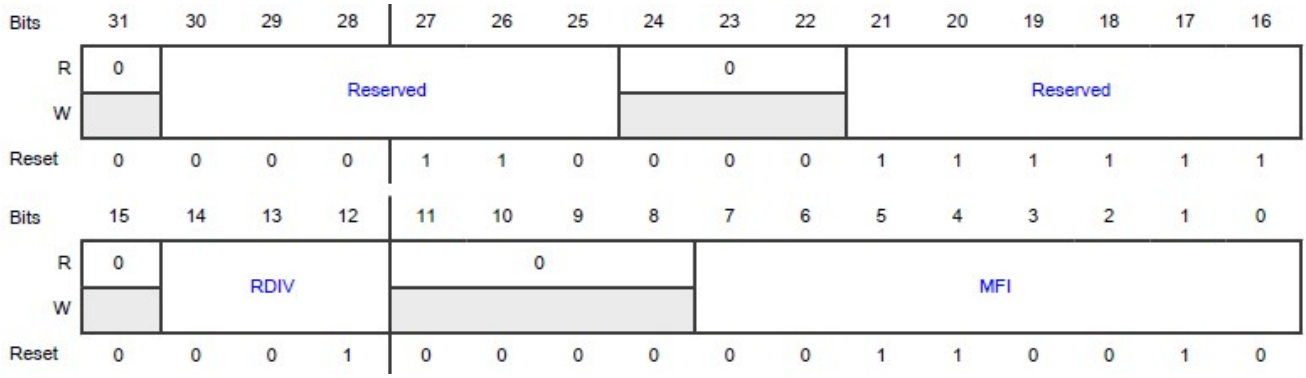
```
=> clk dump
…
DDR_PLL_MUX                 : 40000000 Hz
DDR_PLL_VCO                 : 1520000000 Hz
DDR_PLL_PHI0                : 760000000 Hz
MC_CGM5_MUX0                : 760000000 Hz
DDR                         : 760000000 Hz
…
=>md 0x40044008 1
40044008: 00001026
```

So MFI changed from 40 (0x28) to 38 (0x26):

DDR_PLL base address: 4004_4000h

Offset	Register	Width (In bits)	Access	Reset value
0h	PLL Control (PLLCR)	32	RW	8000_0000h
4h	PLL Status (PLLSR)	32	W1C	0000_0300h
8h	PLL Divider (PLLDV)	32	RW	0C3F_1032h
Ch	PLL Frequency Modulation (PLLFM)	32	RW	4000_0000h
10h	PLL Fractional Divider (PLLFD)	32	RW	0000_0000h
20h	PLL Clock Multiplexer (PLLCLKMUX)	32	RW	0000_0000h
80h	PLL Output Divider (PLLODIV_0)	32	RW	0000_0000h

Register	Offset
PLLDV	8h

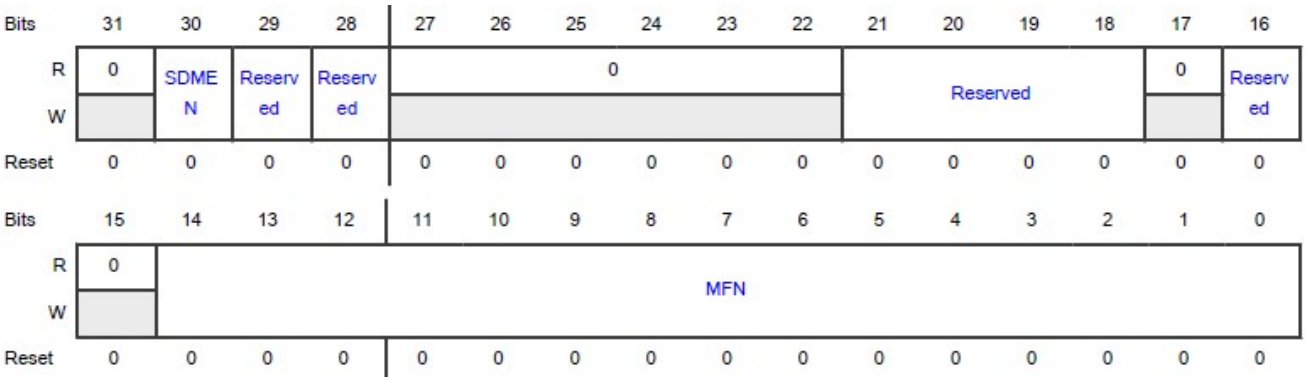


=> md 0x40044010 1

40044010: 40000000

But MFN still=0.

Register	Offset
PLLFD	10h



Field	Function
14-0 MFN	Numerator Of Fractional Loop Division Factor Sets the numerator of the fractional loop division factor. You must write a value of less than 18432 to this field. When Fractional mode is disabled, you must write 000_0000_0000_0000b to this field.

According to the frequency multiplication formula of PLL:

- Integer-only mode:

— When PLLDV[RDIV] is 0:

$$f_{pll\_VCO} = f pll\_ref \times PLLDV[MFI]$$

Equation 1. PLL VCO frequency in integer-only mode when PLLDV[RDIV] is 0

— When PLLDV[RDIV] is not 0:

$$f_{pll\_VCO} = \frac{f_{pll\_ref}}{PLLDV[RDIV]} \times PLLDV[MFI]$$

Equation 2. PLL VCO frequency in integer-only mode when PLLDV[RDIV] is not 0

Fractional mode:

— When PLLDV[RDIV] is 0:

$$f_{pll\_VCO} = f_{pll\_ref} \times \left(PLLDV[MFI] + \frac{PLLFD[MFN]}{18432}\right)$$

Equation 3. PLL VCO frequency in Fractional mode when PLLDV[RDIV] is 0

— When PLLDV[RDIV] is not 0:

$$f_{pll\_VCO} = \frac{f_{pll\_ref}}{PLLDV[RDIV]} \times \left(PLLDV[MFI] + \frac{PLLFD[MFN]}{18432}\right)$$

Equation 4. PLL VCO frequency in Fractional mode when PLLDV[RDIV] is not 0

So the frequency fpll_vco=((fpll_ref_40M/(rdiv=1))*(MFI=38))=40*38=1520.

The test results after the kernel is started are as follows:

```
root@s32g274ardb:~# cat /sys/kernel/debug/clk/clk_summary |grep ddr
   ddrpll_sel                0        0      0    40000000          0     0  50000
      ddr_part_block         0        0      0    40000000          0     0  50000
         ddrpll_vco          0        0      0  1520000000          0     0  50000
            ddrpll_phi0      0        0      0   760000000          0     0  50000
               ddr           0        0      0   760000000          0     0  50000
```

Actually use a spectrum analyzer (such as agilent E4404B 9KHz~6.7GHz ESA-E serials spectrum analyzer).

(to be added in the future)

4 Spread Spectrum of DDR_PLL

Refer to the S32G chip manual description:

26.7.4 Frequency modulation

In Frequency Modulation mode, PLL generates a frequency-modulated clock. The modulation depth and modulation frequency are calculated using the equations shown in Frequency modulation programming.

Write 0 to PLLFM[SPREADCTL] to select center-spread modulation. See Figure 130 that shows an example of center-spread modulation.

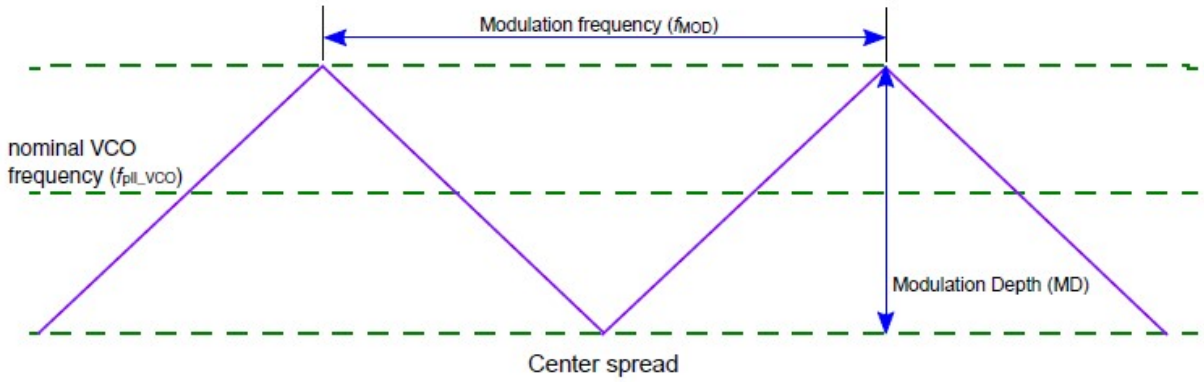


Figure 130. Frequency modulation

26.7.4.1 Frequency modulation programming

Modulation depth and modulation frequency programming uses step number (PLLFM[STEPNO]) and step size (PLLFM[STEPSIZE]). The table below shows variables used during calculations when programming PLL for frequency modulation.

Table 113. Variables for configuring modulation depth and frequency

Variable	Description
f_{REF}	Input clock frequency
f_{MOD}	Expected modulation frequency
MD	Expected modulation depth in percentage
LDF	Loop division factor

Use the following equations to configure PLL for frequency modulation.

$$\mathrm{LDF} = \mathrm{PLLDV[MFI]} + \frac{\mathrm{PLLFD[MFN]}}{18432}$$

Equation 6. LDF

$$\mathrm{PLLFM[STEPNO]} = \frac{f_{REF}}{\left(2 \times f_{MOD} \times \mathrm{PLLDV[RDIV]}\right)}$$

Equation 7. Step number

$$\mathrm{PLLFM[STEPSIZE]} = \frac{\mathrm{MD} \times \mathrm{LDF}}{100 \times \mathrm{PLLFM[STEPNO]}} \times 18432$$

Equation 8. Step size

Frequency modulation is only possible if the condition shown in Equation 9 is met.

$$(\text{PLLFM[STEPSIZE]} \times \text{PLLFM[STEPNO]}) < 18432$$

Equation 9. Requirement to achieve FM

You must write 0 to PLLFM[SSCGBYP] and write 1 to PLLFD[SDMEN] to enable frequency modulation.

$$\text{Max (MD \%)} = \frac{(\text{Fref} \times 100)}{\text{PLLDV[RDIV]} \times \text{Fpll\_vco}}$$

Equation 10. Maximum possible modulation depth

CAUTION

The effective modulation depth may differ from the intended modulation depth because of rounding operations applied to PLLFM[STEPSIZE] and PLLFM[STEPNO].

26.8 Initialization information

Perform the following steps to initialize PLL:

1. Confirm that PLLODIV_*n*[DE] is 0 for all dividers.
2. Confirm that PLLCR[PLLPD] is 1.
3. Program PLLCLKMUX to select the appropriate reference clock.
4. Program the following as needed:
   - PLLDV
   - PLLFD
   - PLLFM to the desired value
5. Program PLLODIV_*n*[DIV] to the desired value.
6. Wait for the PLL reference clock to be stable.
7. Write 0 to PLLCR[PLLPD].
8. Wait for PLLSR[LOCK] to be 1.
9. Write 1 to PLLODIV_*n*[DE].

Perform the following steps to shut down PLL:

1. Write 0 to PLLODIV_n[DE] for all dividers.
2. Write 1 to PLLCR[PLLPD].

Since S32G only supports center spread spectrum, it is necessary to move the center frequency point to lower at least half of the spread spectrum depth before spreading, so as to prevent exceeding the specified range when spreading upward. As follows, the frequency of DDR_VCO cannot be higher than 1600Mhz.

fPLL_DDR_VCO	DDR PLL VCO Frequency Range [6, 7]	1300	—	1600	MHz	without center-spread SSCG enabled	—
fPLL_DDR_PHI0	DDR PLL PHI0 Frequency [6, 8]	800	—	800	MHz	DDR_CLK (3200 MT/s), without center-spread SSCG enabled	—

At present, the uboot code automatically calculates the multiplier coefficient. In the actual test, it is found that if the multiplier with decimals is used, uboot of BSP30 will start the halt. So here is still an integer multiplier to illustrate this situation:

39X40Mhz=1560Mhz，38X40Mhz=1520Mhz。

So the parameters we need to set are:

- fMOD: modulation frequency.

Refer to the S32G datasheet description:

fPLL_MOD	Spread Spectrum Clock Modulation Frequency [13]	30	—	64	KHz	—	—

Between: 30~64KHZ.

- MD: percentage of modulation depth.

Refer to the S32G chip manual description:

$$\text{Max (MD \%)} = \frac{(\text{Fref} \times 100)}{\text{PLLDV[RDIV]} \times \text{Fpll\_vco}}$$

Equation 10. Maximum possible modulation depth

So for:

- 1560Mhz: Max(MD %)=40*100/1560=2.564%, then the spread spectrum depth is 1560X2.564%=40Mhz (peak-to-peak, then the maximum frequency is 1560+40/2=1580Mhz<1600Mhz)
- 1520Mhz: Max(MD %)=40*100/1520=2.6316%, then the spread spectrum depth is 1560X2.6316%=40Mhz (peak-to-peak value, then the maximum frequency is 1520+40/2=1540Mhz<1600Mhz)

Note that in the chip manual, the MD is peak-to-peak according to the diagram, so the calculation of the highest frequency should be divided by 2. In addition, if the frequency is higher than 1600Mhz in the case of the maximum spread spectrum depth, the maximum spread spectrum needs to be reduced to prevent exceeding the specification requirements.

So the modulation frequency is from 30~64Khz, we choose the integer division multiple of fref=40Mhz fmod=40K.

For spread spectrum after frequency change, change the frequency to 1520Mhz (mfi=38)

ldf=mfi(38)+(mfn(0)/18432)=mfi=38;

stepno=Fref(40*MHZ)/(2*S32G274A_REV2_SSC_MOD(40Khz)*rdiv(1))=500

stepsize=(S32G274A_REV2_SSC_MD(26)*ldf(38)*18432)/(1000*stepno(500))~=36(36.421632) // Maximum spread of 26/1000, guaranteed integer multiplication.

Since stepno/stepsize can only take an integer value, when the configuration is MD=2.6%, Fmod=40K, it can be seen that for a VCO of 1520 (mfi=38), the rounding is closer to stepsize=36.

For the spread spectrum of 1560, the frequency is 1576 (mfi=39)

ldf=mfi(39)+mfn(0)/18432=39

stepno=Fref(40*MHZ)/(2*S32G274A_REV2_SSC_MOD(40Khz)*rdiv(1))=500

stepsize=(S32G274A_REV2_SSC_MD(25)*ldf(39)*18432)/(1000*stepno(500))~=35(35.9424)

Relatively less close to the integer (note that because of the rounding of the C language, for values greater than 0.5, the actual value is still an integer value, and the fractional part is rounded, so +1 may be needed to get closer to the required value, but pay attention to the maximum The spread spectrum value cannot be out of bounds and needs to be verified).

Notice：

- In fact, setpno/setpsize is used to derive MD and Fmod, so the setting of the integer of the register will affect the value of MD and Fmod, for example:

For 1520Mhz+2.57% spread spectrum:
S32G274A_REV2_SSC_MD=36X100X500/(18432*38)=2.57, and the 1520 spread spectrum depth requirement is 2.6316% as above, which meets the requirements.

For 1560Mhz+2.43% spread spectrum:
S32G274A_REV2_SSC_MD=35X100X500/(18432*39)=2.43, and the 1560 spread spectrum depth requirement is 2.564% as above, which meets the requirements.

Therefore, when stepno and stepsize are integer values, make MD and Fmod meet the spec requirements, and when they meet the requirements, try to make Fmod an integer.

- How to select the spread spectrum depth and the number of spread spectrum steps should be matched with the actual EMI test results. There is no certain rule. It needs to be tested and determined by itself. This doc does not make any rules.

The test results of the spread spectrum+frequency change to 1520Mhz+2.57% are as follows:

- Uboot Command:

```
=> md 0x40044000 7
40044000: 00000000 00000704 00001026 002401f4    .....&.....
40044010: 40000000 00000000 00000000            ...@.....
```

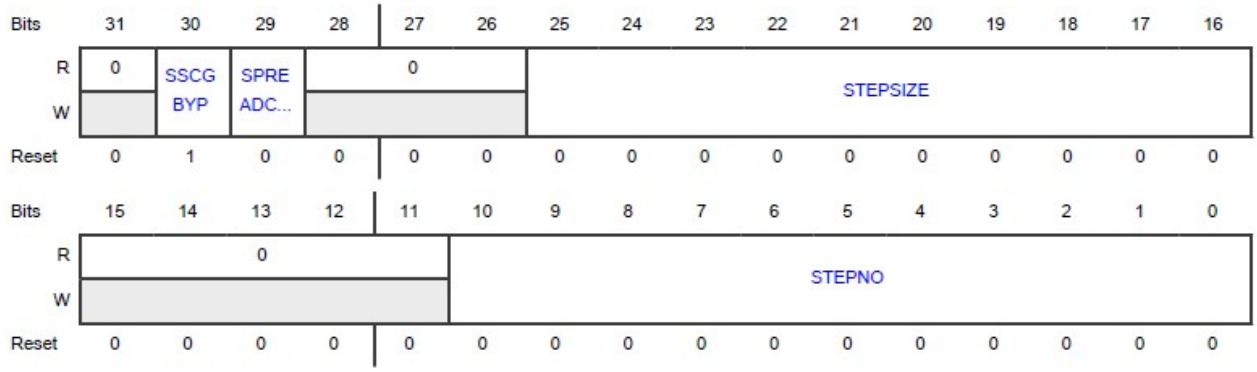
So it should be:

Register	Offset
PLLFM	Ch

Function

Configures PLL frequency modulation parameters.

Diagram



Field	Function
30 SSCGBYP	Frequency Modulation (Spread Spectrum Clock Generation) Bypass Bypasses frequency modulation. 0b - Not bypassed 1b - Bypassed
29 SPREADCTL	Modulation Type Selection Indicates that the modulation is centered around the nominal frequency. 0b - Centered around nominal frequency 1b - Reserved

SSCGBYP=0: Spread spectrum function does not bypass.

SPREADCTL=0: Set as center spread spectrum.

25-16 STEPSIZE	Frequency Modulation Step Size Provides the step size for modulation depth and frequency in Frequency Modulation mode (see Frequency modulation).
15-11 —	Reserved
10-0 STEPNO	Number Of Steps Of Modulation Period Or Frequency Modulation Provides the number of steps to achieve modulation depth in Frequency Modulation mode (see Frequency modulation).

SETPNO=0x1f4=500 steps

STEPSIZE=0x24=36:

Compared with the PLL-related register values before frequency conversion without spreading, the values are as follows:

```
=> md 0x40044000 7
40044000: 00000000 00000704 00001028 40000000    ........(......@
40044010: 40000000 00000000 00000000             ...@........
=>
```

The test results of the spread spectrum and the frequency change to 1560Mhz+2.43% are as follows:

The code is modified to:

```
#ifdef CONFIG_EMI_FC
#define S32G274A_REV2_FC_DDR_PLL_VCO_FREQ (1560 * MHZ) //(1560 * MHZ)
#define S32G274A_REV2_FC_DDR_FREQ (780 * MHZ) //(780 * MHZ)
#ifdef CONFIG_EMI_SSC
#define KHZ (1000UL)
#define S32G274A_REV2_SSC_MD (25) //26%0 for 1520Mhz, 25%0 for 1560Mhz,
#define S32G274A_REV2_SSC_MOD (40 * KHZ)
#endif
#endif
```

- Uboot command:

```
=> clk dump
…
DDR_PLL_MUX                    : 40000000 Hz
DDR_PLL_VCO                    : 1560000000 Hz
```

```
DDR_PLL_PHI0                    : 780000000 Hz
MC_CGM5_MUX0                    : 780000000 Hz
DDR                             : 780000000 Hz
=> md 0x40044000 7
40044000: 00000000 00000704 00001027 002301f4    ........'.....
40044010: 40000000 00000000 00000000             ...@........
```

Actually use a spectrum analyzer (such as agilent E4404B 9KHz~6.7GHz ESA-E serials spectrum analyzer).

(to be added in the future)

5 Summarize the Modified Source Code

Uboot's ddr clock initialization process is:

```
init_sequence_f(common/board_f.c)
|->arch_cpu_init(arch/arm/cpu/armv8/s32/cpu.c)
|   |-> enable_early_clocks(drivers/clk/s32/early_clocks.c)
|   |   |->enable_ddr_clock
|   |   |   |->s32gen1_enable
```

Eventually the PLL will be found from source to end:

```
|   |   |   |   |->enable_module
|   |   |   |   |   |->enable_pll
|   |   |   |   |   |   |->program_pll
```

The modified source code is as follows:

Configs/s32g274ardb2_defconfig

```
CONFIG_EMI_FC=y //Controls whether to turn on frequency conversion
CONFIG_EMI_SSC=y // Control whether to open frequency, we all open by default.
```

Drivers/clk/Kconfig

```
config EMI_FC
        bool "enable S32G DDR PLL frequency changing "
        help
          Enable S32G DDR PLL frequency changing.

config EMI_SSC
        bool "enable S32G DDR PLL frequency spread spectrum clocking "
        help
          Enable S32G DDR PLL frequency spread spectrum clocking.

config EMI_DEBUG
```

```
    bool "enable S32G DDR PLL EMI DEBUG "
    help
      Enable S32G DDR PLL EMI DEBUG.
```

uboot/include/dt-bindings/clock/s32gen1-clock-freq.h

```
#ifdef CONFIG_EMI_FC
#define S32G274A_REV2_FC_DDR_PLL_VCO_FREQ (1520 * MHZ) //(1560 * MHZ)
#define S32G274A_REV2_FC_DDR_FREQ (760 * MHZ) //(780 * MHZ)
#ifdef CONFIG_EMI_SSC
#define KHZ (1000UL)
#define S32G274A_REV2_SSC_MD (26) //26%0 for 1520Mhz, 25%0 for 1560Mhz,
#define S32G274A_REV2_SSC_MOD (40 * KHZ)//modulation frequency range is 30K to 64K from datasheet
#endif
#endif
```

Uboot/drivers/clk/s32/early_clocks.c

```
static int enable_ddr_clock(void)
{…
        ddr_pll_freq = S32GEN1_DDR_PLL_VCO_FREQ;
        ddr_freq = S32GEN1_DDR_FREQ;
    }
#if defined(CONFIG_EMI_FC)
  ddr_pll_freq = S32G274A_REV2_FC_DDR_PLL_VCO_FREQ; //改频或改频作为展频基础
  ddr_freq = S32G274A_REV2_FC_DDR_FREQ;
#else
…
```

Uboot/arch/arm/include/asm/arch-s32/s32-gen1/mc_cgm_regs.h

```
//johnli for emi //设置FM寄存器的宏
//johnli for emi
#if defined(CONFIG_EMI_SSC)

#define PLLDIG_PLLFM_SSCGBYP_SET(val )(PLLDIG_PLLFM_SSCGBYP_MASK & \
                                        ((val)<< PLLDIG_PLLFM_SSCGBYP_OFFSET))


#define PLLDIG_PLLFM_SPREADCTL_OFFSET       (29)
#define PLLDIG_PLLFM_SPREADCTL_MASK  (0x20000000)
#define PLLDIG_PLLFM_SPREADCTL_SET(val)     (PLLDIG_PLLFM_SPREADCTL_MASK & \
                                        ((val)<< PLLDIG_PLLFM_SPREADCTL_OFFSET))

#define PLLDIG_PLLFM_STEPSIZE_OFFSET   (16)
#define PLLDIG_PLLFM_STEPSIZE_MASK     (0x03FF0000)
#define PLLDIG_PLLFM_STEPSIZE_SET(val)  (PLLDIG_PLLFM_STEPSIZE_MASK & \
                                        ((val)<< PLLDIG_PLLFM_STEPSIZE_OFFSET))


#define PLLDIG_PLLFM_STEPNO_OFFSET     (0)
#define PLLDIG_PLLFM_STEPNO_MASK       (0x000007FF)
#define PLLDIG_PLLFM_STEPNO_SET(val)    (PLLDIG_PLLFM_STEPNO_MASK & \
```

```
                                                        ((val)<< PLLDIG_PLLFM_STEPNO_OFFSET))
#endif
//end
```

Uboot\drivers\clk\s32\enable_clk.c

```
static int program_pll(struct s32gen1_pll *pll, void *pll_addr,
                       struct s32gen1_clk_priv *priv, u32 clk_src)
{
        …
        u32 rdiv = 1, mfi, mfn;
        //johnli for emi
#if defined(CONFIG_EMI_SSC)
        u32 stepno, stepsize,;
        float ldf;
#endif
        //end
…
        writel(PLLDIG_PLLFD_MFN_SET(mfn) |
             PLLDIG_PLLFD_SMDEN, PLLDIG_PLLFD(pll_addr));  // PLLDIG_PLLFD_SMDEN already set to 1
        //johnli for emi
        #if defined(CONFIG_EMI_SSC)

if(0x40044000 == pll_addr)//Spread spectrum for DDR PLL only
        {

    ldf=(float)mfi+((float)mfn/18432); //for 1520Mhz clock, mfi=38(38X40=1520), mfn=0, so ldf=38
    stepno=(40*MHZ)/(2*S32G274A_REV2_SSC_MOD*rdiv); // 40M/(2X40K*1)=500
    stepsize=(uint)((S32G274A_REV2_SSC_MD*ldf*18432)/(1000*stepno)); // mdX38X18432/(100X500)= ?
//md/1000

        writel(PLLDIG_PLLFM_SSCGBYP_SET(0)|PLLDIG_PLLFM_SPREADCTL_SET
(0)|PLLDIG_PLLFM_STEPSIZE_SET (stepsize)|PLLDIG_PLLFM_STEPNO_SET(stepno),
PLLDIG_PLLFM(pll_addr));
}
        #endif
        //end
        ret = adjust_odiv_settings(pll, pll_addr, priv, odivs_mask, old_vco);
…
```