GigaDevice Semiconductor Inc.

GD32 Clock Switching Configuration User Guide

Application Note:
AN250

Version 1.0

(December 2024).

Table of Contents

List of Tables

1. Introduction

This topic describes how to reconfigure the system clock when the GD32 series needs clock switching.

When the system clock frequency is directly switched from high to low or from low to high frequency, such as in the Boot+APP architecture, or the clock needs to be reconfigured in and out of deepsleep, etc., in such scenarios, it is necessary to add a phased up-frequency or down-frequency operation, first reduce the frequency in stages, and then switch the system clock source to other clock sources, such as the internal high-speed crystal oscillator, and then modify PLL, the final stage rises to the target frequency.

2. Development environment

2.1. Hardware environment

- Hardware development board: GD32F307C-EVAL development board
- Program tool: Onboard GD-Link

2.2. Software environment

- Operating system: Win10-64-bit
- Development environment: KEIL 5.29
- Firmware library: GD32F30x_Firmware_Library

The GD32F30x_Firmware_Library can be downloaded from the website at **www.gd32mcu.com**.

3. Clock frequency switching configuration

In system-level applications, there will be application scenarios such as the architecture design of Boot+APP. In the boot program, the change of the system clock is the process of switching from the internal high-speed crystal oscillator to the highest frequency of the PLL, and the application layer needs to pay attention to adding a phased frequency cutting code, or in the APP project, if the target frequency requirements do not change, there is no need to reconfigure the clock.

3.1. Phased frequency reduction

3.1.1. Downscaling code

The application layer needs to avoid switching directly from high to low frequencies (e.g. IRC8M). ***Table 3-1. Sample code for three-stage frequency reduction*** gives an example code for three-stage frequency reduction. Assuming that the current clock is 120MHz, the frequency changes to 120MHz after calling the following three-stage downscaling code→60MHz→30MHz→15MHz, and then switch the system clock to the internal high-speed crystal.

Table 3-1. Sample code for three-stage frequency reduction

```
#define RCU_MODIFY_DE_3(__delay)    do{                                 \
                                        volatile uint32_t i,reg;        \
                                        if(0 != __delay){ \
                                            /* Insert a software delay */   \
                                            for(i=0; i<__delay; i++){       \
                                            }                               \
                                            reg = RCU_CFG0;                 \
                                            reg &= ~(RCU_CFG0_AHBPSC);      \
                                            reg |= RCU_AHB_CKSYS_DIV2;      \
                                            /* AHB = SYSCLK/2 */            \
                                            RCU_CFG0 = reg;                 \
                                            /* Insert a software delay */   \
                                            for(i=0; i<__delay; i++){       \
                                            }                               \
                                            reg = RCU_CFG0;                 \
                                            reg &= ~(RCU_CFG0_AHBPSC);      \
                                            reg |= RCU_AHB_CKSYS_DIV4;      \
                                            /* AHB = SYSCLK/4 */            \
                                            RCU_CFG0 = reg;                 \
                                            /* Insert a software delay */   \
                                            for(i=0; i<__delay; i++){       \
```

```
                                    }                                   \
                                    reg = RCU_CFG0;                     \
                                    reg &= ~(RCU_CFG0_AHBPSC);          \
                                    reg |= RCU_AHB_CKSYS_DIV8;          \
                                    /* AHB = SYSCLK/8 */                \
                                    RCU_CFG0 = reg;                     \
                                    /* Insert a software delay */       \
                                    for(i=0; i<__delay; i++){           \
                                    }                                   \
                                }                                       \
                            }while(0)
```

3.1.2. Specific examples of frequency reduction

Table 3-2. Sample code for frequency reduction is the sample code of the frequency reduction application is given, and the system works at a high frequency before the switch_system_clock_to_72m_irc8m function is called.

Table 3-2. Sample code for frequency reduction

```
.....
/* function declaration */
static void switch_system_clock_to_72m_irc8m(void);
.....
int main(void)
{
    .....
    /* clock switching */
    switch_system_clock_to_72m_irc8m();
    .....
}
.......

static void switch_system_clock_to_72m_irc8m(void)
{
    uint32_t timeout = 0U;
    uint32_t stab_flag = 0U;

    /* enable IRC8M */
    RCU_CTL |= RCU_CTL_IRC8MEN;

    /* wait until IRC8M is stable or the startup time is longer than IRC8M_STARTUP_TIMEOUT */
    do{
        timeout++;
        stab_flag = (RCU_CTL & RCU_CTL_IRC8MSTB);
```

```
    }while((0U == stab_flag) && (IRC8M_STARTUP_TIMEOUT != timeout));

    /* if failed */
    if(0U == (RCU_CTL & RCU_CTL_IRC8MSTB)){
      while(1){
      }
    }

    /* add frequency reduction code to avoid switching directly from high frequency to the lowest frequency */
    RCU_MODIFY_DE_3(0x50);
    /* select IRC8M as system clock source, deinitialize the RCU */
    rcu_system_clock_source_config(RCU_CKSYSSRC_IRC8M);
    rcu_deinit();
    /* AHB = SYSCLK */
    RCU_CFG0 |= RCU_AHB_CKSYS_DIV1;
    /* APB2 = AHB */
    RCU_CFG0 |= RCU_APB2_CKAHB_DIV1;
    /* APB1 = AHB */
    RCU_CFG0 |= RCU_APB1_CKAHB_DIV1;
    /* PLL = (IRC8M/2) * 18 = 72 MHz */
    RCU_CFG0 &= ~(RCU_CFG0_PLLSEL | RCU_CFG0_PLLMF);
    RCU_CFG0 |= (RCU_PLLSRC_IRC8M_DIV2 | RCU_PLL_MUL18);

    /* enable PLL */
    RCU_CTL |= RCU_CTL_PLLEN;

    /* wait until PLL is stable */
    while(0 == (RCU_CTL & RCU_CTL_PLLSTB));

    /* select PLL as system clock */
    RCU_CFG0 &= ~RCU_CFG0_SCS;
    RCU_CFG0 |= RCU_CKSYSSRC_PLL;

    /* wait until PLL is selected as system clock */
    while(0 == (RCU_CFG0 & RCU_SCSS_PLL));
}
```

3.2. Phased upscaling

3.2.1. Upscaling code

The application layer needs to avoid directly switching from the low frequency (such as IRC8M)

to the highest frequency, so it is recommended to add it. ***Table 3-3. Three-stage upscaling sample code*** is the code listed.

Table 3-3. Three-stage upscaling sample code

```
#define RCU_MODIFY_UP_3(__delay)    do{                                     \
                                        volatile uint32_t i,reg;            \
                                        if(0 != __delay){ \
                                            /* Insert a software delay */   \
                                            for(i=0; i<__delay; i++){       \
                                            }                               \
                                            reg = RCU_CFG0;                 \
                                            reg &= ~(RCU_CFG0_AHBPSC);      \
                                            reg |= RCU_AHB_CKSYS_DIV4;      \
                                            /* AHB = SYSCLK/4 */            \
                                            RCU_CFG0 = reg;                 \
                                            /* Insert a software delay */   \
                                            for(i=0; i<__delay; i++){       \
                                            }                               \
                                            reg = RCU_CFG0;                 \
                                            reg &= ~(RCU_CFG0_AHBPSC);      \
                                            reg |= RCU_AHB_CKSYS_DIV2;      \
                                            /* AHB = SYSCLK/2 */            \
                                            RCU_CFG0 = reg;                 \
                                            /* Insert a software delay */   \
                                            for(i=0; i<__delay; i++){       \
                                            }                               \
                                            reg = RCU_CFG0;                 \
                                            reg &= ~(RCU_CFG0_AHBPSC);      \
                                            reg |= RCU_AHB_CKSYS_DIV1;      \
                                            /* AHB = SYSCLK/1 */            \
                                            RCU_CFG0 = reg;                 \
                                        }                                   \
                                    }while(0)
```

3.2.2. A specific example of upscaling

Table 3-4. Example code for upscaling is the sample code for an upscaling application is given. After deepsleep wakes up, the frequency is doubled from the internal high-speed crystal oscillator to a specific frequency, and the change of the system frequency is first divided by 8 by the internal high-speed crystal oscillator, and then PLL is configured and switch the clock source to the PLL, and then start the step-by-step upscaling.

Table 3-4. Example code for upscaling

```
......
```

```
/* function declaration */
static void switch_system_clock_reconfig(void);
static void __system_clock_120m_hxtal(void);
......
int main(void)
{
    ......
    /* enter deepsleep mode */
    pmu_to_deepsleepmode(PMU_LDO_LOWPOWER,          PMU_LOWDRIVER_DISABLE,
WFI_CMD);
    /* clock switching */
    switch_system_clock_reconfig();
    ......
}
.......

static void switch_system_clock_reconfig(void)
{
    /* FPU settings */
#if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
    SCB->CPACR |= ((3UL << 10*2)|. 3UL << 11*2));    /* set CP10 and CP11 Full Access */
#endif
    /* reset the RCU clock configuration to the default reset state */
    /* Set IRC8MEN bit */
    RCU_CTL |= RCU_CTL_IRC8MEN;
    while(0U == (RCU_CTL & RCU_CTL_IRC8MSTB)){
    }
    /* add frequency reduction code to avoid switching directly from high frequency to the lowest
frequency */
    if(((RCU_CFG0 & RCU_CFG0_SCSS) == RCU_SCSS_PLL)){
        RCU_MODIFY_DE_3(0x50);
     }

    RCU_CFG0 &= ~RCU_CFG0_SCS;

#if (defined(GD32F30X_HD) || defined(GD32F30X_XD))
    /* reset HXTALEN, CKMEN and PLLEN bits */
    RCU_CTL &= ~(RCU_CTL_PLLEN | RCU_CTL_CKMEN | RCU_CTL_HXTALEN);
    /* disable all interrupts */
    RCU_INT = 0x009f0000U;
#elif defined(GD32F30X_CL)
    /* reset HXTALEN, CKMEN, PLLEN, PLL1EN and PLL2EN bits */
    RCU_CTL    &=    ~(RCU_CTL_PLLEN    |RCU_CTL_PLL1EN    |    RCU_CTL_PLL2EN    |
```

```
RCU_CTL_CKMEN | RCU_CTL_HXTALEN);
    /* disable all interrupts */
    RCU_INT = 0x00ff0000U;
#endif

    /* reset HXTALBPS bit */
    RCU_CTL &= ~(RCU_CTL_HXTALBPS);

    /* reset CFG0 and CFG1 registers */
    RCU_CFG0 = 0x00000000U;
    RCU_CFG1 = 0x00000000U;
    /* configure the system clock source, PLL Multiplier, AHB/APBx prescalers and Flash settings */
    __system_clock_120m_hxtal();
}

static void __system_clock_120m_hxtal(void)
{
    uint32_t timeout = 0U;
    uint32_t stab_flag = 0U;

    /* enable HXTAL */
    RCU_CTL |= RCU_CTL_HXTALEN;

    /* wait until HXTAL is stable or the startup time is longer than HXTAL_STARTUP_TIMEOUT */
    do{
        timeout++;
        stab_flag = (RCU_CTL & RCU_CTL_HXTALSTB);
    }while((0U == stab_flag) && (HXTAL_STARTUP_TIMEOUT != timeout));

    /* if fail */
    if(0U == (RCU_CTL & RCU_CTL_HXTALSTB)){
        while(1){
        }
    }

    RCU_APB1EN |= RCU_APB1EN_PMUEN;
    PMU_CTL |= PMU_CTL_LDOVS;

    /* HXTAL is stable */
    /* AHB = SYSCLK/8 */
    RCU_CFG0 |= RCU_AHB_CKSYS_DIV8;
    /* APB2 = AHB/1 */
    RCU_CFG0 |= RCU_APB2_CKAHB_DIV1;
```

```
    /* APB1 = AHB/2 */
    RCU_CFG0 |= RCU_APB1_CKAHB_DIV2;

#if (defined(GD32F30X_HD) || defined(GD32F30X_XD))
    /* select HXTAL/2 as clock source */
    RCU_CFG0 &= ~(RCU_CFG0_PLLSEL | RCU_CFG0_PREDV0);
    RCU_CFG0 |= (RCU_PLLSRC_HXTAL_IRC48M | RCU_CFG0_PREDV0);

    /* CK_PLL = (CK_HXTAL/2) * 30 = 120 MHz */
    RCU_CFG0 &= ~(RCU_CFG0_PLLMF | RCU_CFG0_PLLMF_4 | RCU_CFG0_PLLMF_5);
    RCU_CFG0 |= RCU_PLL_MUL30;

#elif defined(GD32F30X_CL)
    /* CK_PLL = (CK_PREDIV0) * 30 = 120 MHz */
    RCU_CFG0 &= ~(RCU_CFG0_PLLMF | RCU_CFG0_PLLMF_4 | RCU_CFG0_PLLMF_5);
    RCU_CFG0 |= (RCU_PLLSRC_HXTAL_IRC48M | RCU_PLL_MUL30);

    /* CK_PREDIV0 = (CK_HXTAL)/5 *8 /10 = 4 MHz */
    RCU_CFG1 &= ~(RCU_CFG1_PLLPRESEL | RCU_CFG1_PREDV0SEL |
RCU_CFG1_PLL1MF | RCU_CFG1_PREDV1 | RCU_CFG1_PREDV0);
    RCU_CFG1 |= (RCU_PLLPRESRC_HXTAL | RCU_PREDV0SRC_CKPLL1 |
RCU_PLL1_MUL8 | RCU_PREDV1_DIV5 | RCU_PREDV0_DIV10);

    /* enable PLL1 */
    RCU_CTL |= RCU_CTL_PLL1EN;
    /* wait till PLL1 is ready */
    while((RCU_CTL & RCU_CTL_PLL1STB) == 0U){
    }
#endif /* GD32F30X_HD and GD32F30X_XD */

    /* enable PLL */
    RCU_CTL |= RCU_CTL_PLLEN;

    /* wait until PLL is stable */
    while(0U == (RCU_CTL & RCU_CTL_PLLSTB)){
    }

    /* enable the high-drive to extend the clock frequency to 120 MHz */
    PMU_CTL |= PMU_CTL_HDEN;
    while(0U == (PMU_CS & PMU_CS_HDRF)){
    }

    /* select the high-drive mode */
```

```
    PMU_CTL |= PMU_CTL_HDS;
    while(0U == (PMU_CS & PMU_CS_HDSRF)){
    }

    /* select PLL as system clock */
    RCU_CFG0 &= ~RCU_CFG0_SCS;
    RCU_CFG0 |= RCU_CKSYSSRC_PLL;

    /* wait until PLL is selected as system clock */
    while(0U == (RCU_CFG0 & RCU_SCSS_PLL)){
    }
    /* add frequency escalation code to avoid switching directly from lowest frequency to the high frequency */
    RCU_MODIFY_UP_3(0x50);
}
```

3.3. Other considerations

In addition to the macros provided above, you can also call the rcu_ahb_clock_config function provided in gd32f30x_rcu.c to change the AHB clock, note that after each AHB frequency change, you need to add a little delay to ensure the stability of the clock.

In order to change the frequency, you need to switch the clock from the PLL to another clock source, then modify the PLL, and then switch the system clock source back to PLL.

For other series, a similar approach can be used to avoid anomalies when switching clock speeds.

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	First release	December 17, 2024

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company according to the laws of the People’s Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.