GigaDevice Semiconductor Inc.

GD32H7xx Series
Software Development Guide

Application Notes
AN111

Version 1.0

(May 2023)

Table of Contents

List of Figures

List of Tables

1. Overview

This document is intended for GD32H7xx MCU, introducing how to build and debug GD32H7xx chip based projects and how to use each module. This application note aims to give an exemplary introduction to peripheral resources on GD32H7xx MCU so that users can know how to develop software rapidly with GD32H7xx chips.

Table 1-1. Applicable product

Type	Model
MCU	GD32H7xx Series

2. Development of software functions

2.1. Selection and configuration of Boot modes

GD32H7xx provides different Boot modes whose difference mainly lies in booting location. In general, it includes secure Boot mode and standard Boot mode.

Secure Boot mode can only be initiated from ROM. For details, please refer to secure storage management of AN113 GD32H7xx series. After properly configuring option byte or SCR in Efuse and register in the secure area, users can initiate secure Boot mode. At the next start-up, SECURITY BOOT will be directly initiated regardless of other Boot-related configurations.

Standard Boot mode allows three Boot modes, including USER BOOT, SRAM BOOT, and SYSTEM BOOT. Boot mode can be selected through hardware BOOT pin together with Efuse and option bytes registers. When configuring, EFUSE is superior to option bytes. BOOT pin is adopted to select Boot address 0/1. When the BOOT pin level is low, the high bit of BOOT address is defined by BOOT_ADDR0[15:0]. When the BOOT pin level is high, the high bit of BOOT address is defined by BOOT_ADDR1[15:0]. High-level read protection configured in standard Boot mode is prohibited in some boot sectors. For details, please refer to the table below.

Table 2-1. Selection of Boot mode

SCR	SPC[7:0]	BOOT_ADDRESS (configured in BOOT_ADDRx (x = 0,1))	BOOT mode	Boot address
1	x	XXXX	SECURITY BOOT	ROM
0	High protection level	0x9000_0000	USER BOOT	OSPI0
		0x7000_0000	USER BOOT	OSPI1
		0x0800_0000~max user flash	USER BOOT	BOOT_ADDRESS
		other	USER BOOT	0x0800_0000
	No protection level / Low protection level	0x9000_0000	USER BOOT	OSPI0
		0x7000_0000	USER BOOT	OSPI1
		0x2408_0000 ~ max RAM shared (ITCM/DTCM/AXI)	SRAM BOOT (RAM shared)	BOOT_ADDRESS
		0x2400_0000~ max AXI SRAM	SRAM BOOT (AXI SRAM)	BOOT_ADDRESS
		0x2000_0000	SRAM BOOT (DTCM)	0x2000_0000
		0x0800_0000~max user flash	USER BOOT	BOOT_ADDRESS
		0x0000_0000	SRAM BOOT (ITCM)	0x0000_0000

SCR	SPC[7:0]	BOOT_ADDRESS (configured in BOOT_ADDRx (x = 0,1))	BOOT mode	Boot address
		0x1FF0_0000	SYSTEM BOOT	BootLoader
		Others	USER BOOT	0X0800_0000 (when the BOOT pin level is low)
			SYSTEM BOOT	BootLoader (When the BOOT pin level is high)

2.2. Instruction of PMU use-related issues

Power consumption design is one of the most highlighted issues for GD32H7xx products. GD32H7xx allows the switch mode power supply low dropout regulator (SMPS low dropout regulator), USB power regulator, power-saving mode, and other features. Instructions are required for some issues that are worth attention when using specific functions.

2.2.1. SMPS initialization configuration

The SMPS low dropout (LDO) regulator can be used to set the power supply of 0.9 V power domain. With different configurations, six valid power supply modes of 0.9V power domain can be achieved. Different power supply modes can make the chips more balanced in performance and power consumption. However, the following issues require attention in terms of SMPS configuration.

■ Configuration of power supply mode must be prior to PLL configuration. PMU in default status can't drive application of high basic frequency or high load.

■ Configuration of power supply mode should match external circuits. For example,
   – When the external circuit is solely powered by SMPS, SMPS output is connected to V_{CORE} to power $V_{0.9V}$ directly. If software configuration is such that LDO is powered by SMPS and powers $V_{0.9V}$, SMPS will output 1.8 V or 2.5 V to $V_{0.9V}$ to burn out the chip.
   – When the external circuit is so configured that LDO is powered by SMPS and powers $V_{0.9V}$ power domain, SMPS output will be connected to V_{DDLDO} to power LDO. If the power supply mode solely by SMPS is configured for the software, LDO will be turned off which will cause power outage of $V_{0.9V}$.

■ Some packages don't have SMPS-related pins, so some power supply modes are not available. However, it is still required to configure LDO power supply mode or bypass mode before PLL configuration.

2.2.2. POR_ON pin

POR_ON pin should be connected to VDD. Otherwise, after POR is reset, some registers may not in reset status.
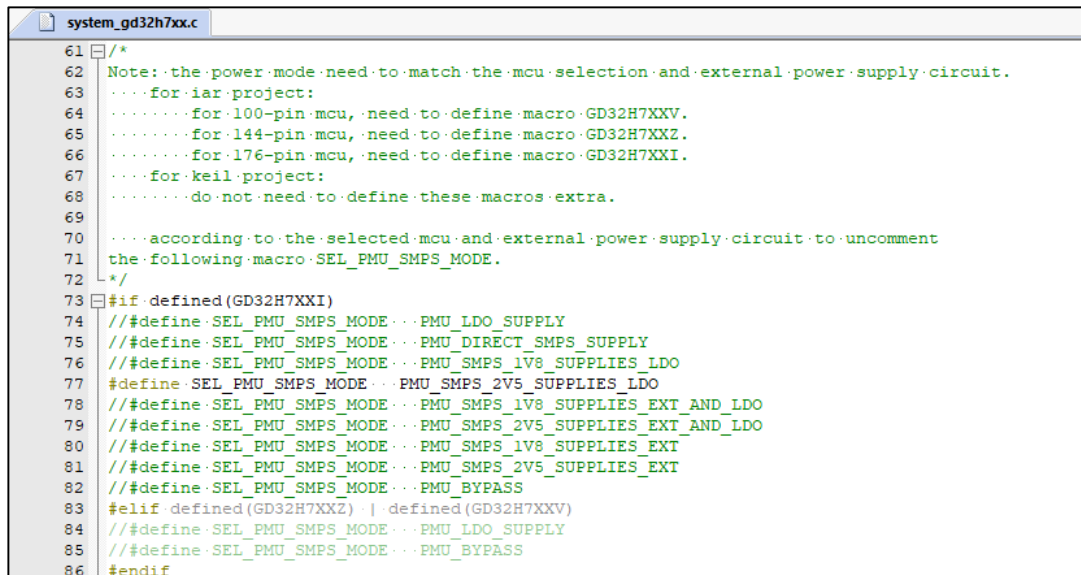
2.2.3. Programmer link issues in sleep mode

At present, the chip in sleep mode can not be downloaded or debugged through the programmer. It should exit from the sleep mode before being downloaded and debugged with the downloader.

2.2.4. Pxx pin and Pxx_C pin link issues in standby mode

Chips in standby mode can be awoken through the rising edge of WKUP pins. It is worth attention that WKUP pins of GD32H7xx have simulation input end. In other words, there are Pxy_C and Pxy pin pairs. When the chip is in standby mode, Pxy_C and Pxy pins will be short circuited.

2.2.5. Software configuration

To correctly configure the PMU mode, users need to select the correct PMU mode in the firmware library system_gd32h7xx.c to match the external circuit. For example, given that users use GD32H759I chip whose external circuit is powered by SMPS&LDO, to properly configure PMU to obtain 2.5 V SMPS output, users need to uncomment the codes in line 77 "#define SEL_PMU_SMPS_MODE   PMU_SMPS_2V5_SUPPLIES_LDO", as shown in the figure below.

```
    system_gd32h7xx.c
61 ⊟/*
62 |Note:·the·power·mode·need·to·match·the·mcu·selection·and·external·power·supply·circuit.
63 |····for·iar·project:
64 |········for·100-pin·mcu,·need·to·define·macro·GD32H7XXV.
65 |········for·144-pin·mcu,·need·to·define·macro·GD32H7XXZ.
66 |········for·176-pin·mcu,·need·to·define·macro·GD32H7XXI.
67 |····for·keil·project:
68 |········do·not·need·to·define·these·macros·extra.
69 |
70 |····according·to·the·selected·mcu·and·external·power·supply·circuit·to·uncomment
71 |the·following·macro·SEL_PMU_SMPS_MODE.
72 └*/
73 ⊟#if·defined(GD32H7XXI)
74 |//#define·SEL_PMU_SMPS_MODE···PMU_LDO_SUPPLY
75 |//#define·SEL_PMU_SMPS_MODE···PMU_DIRECT_SMPS_SUPPLY
76 |//#define·SEL_PMU_SMPS_MODE···PMU_SMPS_1V8_SUPPLIES_LDO
77 |#define·SEL_PMU_SMPS_MODE···PMU_SMPS_2V5_SUPPLIES_LDO
78 |//#define·SEL_PMU_SMPS_MODE···PMU_SMPS_1V8_SUPPLIES_EXT_AND_LDO
79 |//#define·SEL_PMU_SMPS_MODE···PMU_SMPS_2V5_SUPPLIES_EXT_AND_LDO
80 |//#define·SEL_PMU_SMPS_MODE···PMU_SMPS_1V8_SUPPLIES_EXT
81 |//#define·SEL_PMU_SMPS_MODE···PMU_SMPS_2V5_SUPPLIES_EXT
82 |//#define·SEL_PMU_SMPS_MODE···PMU_BYPASS
83 |#elif·defined(GD32H7XXZ)·|·defined(GD32H7XXV)
84 |//#define·SEL_PMU_SMPS_MODE···PMU_LDO_SUPPLY
85 |//#define·SEL_PMU_SMPS_MODE···PMU_BYPASS
86 |#endif
```

2.3. Instruction of RCU use

Some peripheral clocks can be configured to allow the user to select required clock for configuration. However, before configuring peripheral clocks, users should ensure that the corresponding clock is stable and running. For peripheral devices that support clock dynamic switch, user should ensure that the target clock has been stable before being switched. The

configurable peripheral clocks are as below.

1. ADC clock can be obtained by dividing the frequency of PLL1P, PLL2R, CK_PER, or AHB clock by 2, 4, 6, 8, 10, 12, 14, and 16.

2. The clock of TRNG can be selected as CK_PLL0Q, CK_PLL2P or IRC48M. The TRNG supports dynamic clock switch.

3. The clock of USART can be selected as CK_APBx (0, 1), CK_AHB, CK_LXTAL or CK_IRC64MDIV. The USART supports dynamic clock switch.

4. The clock of I2C can be selected as CK_APB1, CK_PLL2R, CK_IRC64MDIV or CK_LPIRC4M. The I2C supports dynamic clock switch.

5. The clocks of SPI0 (I2S0), SPI1 (I2S1), and SPI2 (I2S2) can be selected as CK_PLL0Q, CK_PLL1P, CK_PLL2P, I2S_CKIN or CK_PER. The SPI0(I2S0), SPI1(I2S1) and SPI2(I2S2) support dynamic clock switch.

6. The clocks of SPI3 and SPI4 can be selected as CK_APB2, CK_PLL1Q, CK_PLL2Q, CK_IRC64MDIV, CK_LPIRC4M or CK_HXTAL. The SPI3/SPI4 supports dynamic clock switch.

7. The clock of SPI5 (I2S5) can be selected as CK_APB2, CK_PLL1Q, CK_PLL2Q, CK_IRC64MDIV, CK_LPIRC4M, CK_HXTAL or I2S_CKIN. The SPI5/I2S5 supports dynamic clock switch.

8. The clock of LPDTS can be selected as CK_APB4 or CK_LXTAL.

9. The clock of CAN can be selected as CK_HXTAL, CK_APB2, CK_APB2/2 or CK_IRC64MDIV. The CAN supports dynamic clock switch.

10. The clock of RSPDIF can be selected as CCK_PLL0Q, CK_PLL1R, CK_PLL2R or CK_IRC64MDIV. The RSPDIF supports dynamic clock switch.

11. The clock of SAI2 can be selected as CK_PLL0Q, CK_PLL1P, CK_PLL2P, I2S_CKIN, CK_PER or CK_RSPDIF_SYMB. The SAI2 supports dynamic clock switch.

12. The clocks of SAI0 and SAI1 can be selected as CK_PLL0Q, CK_PLL1P, CK_PLL2P, I2S_CKIN or CK_PER. The SAI0 and SAI1 support dynamic clock switch.

13. The clock of HPDF can be selected as CK_AHB or CK_APB2. The HPDF supports dynamic clock switch.

14. The clock of HPDF_AUDIO can be selected as CK_PLL0Q, CK_PLL1P, CK_PLL2P, I2S_CKIN or CK_PER.

15. The clock of EXMC can be selected as CK_AHB, CK_PLL0Q, CK_PLL1R or CK_PER. The EXMC supports dynamic clock switch.

16. The clock of SDIO can be selected as CK_PLL0Q and CK_PLL1R. The SDIO supports dynamic clock switch.

17. The clock of RTC can be selected as LXTAL clock, IRC32K clock, or HXTAL clocks divided by 2-63.

2.4. Monitoring junction temperature

During operation of chips, excessively high or low PN junction temperature will lead to declination of performance or even cause damage to the chips in more serious cases.

2.4.1. Two methods for monitoring junction temperature

GD32H7xx can have junction temperature monitored in two ways, including reading the value of ADC high precision temperature senor or temperature monitor in PMU (JTM alarm function).

2.4.2. Method for monitoring junction temperature with ADC

Real-time junction temperature can be monitored with ADC high precision temperature sensor. Detailed processes are as below:

1. Configure conversion sequence and sampling time (t_{s_temp} us) of the temperature sensor channel (ADC2_CH20). For typical value of t_{s_temp}, please refer to the datasheet.
2. Set TSVEN2 bit of ADC_CTL1 register to enable the temperature sensor.
3. Set ADCON bit of ADC_CTL1 register, or externally trigger ADC to convert.
4. Read the temperature sensor data $V_{temperature}$ from ADC data register, and calculate the actual temperature by using the following formula:

$$\text{Temperature (°C)} = \{(V_{temperature} - V_{25}) / Avg\_Slope\} + 25$$

V_{25}: internal temperature sensor output voltage at 25°C. For the typical value, please refer to the datasheet.

Avg_Slope: Average Slope for curve between Temperature vs. $V_{temperature}$, the typical value please refer to the datasheet.

2.4.3. JTM alarm function

When VBTMEN bit of PMU_CTL1 register is set, monitoring of temperature threshold will be initiated. Junction temperature is monitored by comparing high and low temperature thresholds. TEMPH and TEMPL signs in PMU_CTL1 register can indicate whether the instrument temperature is higher or lower than the threshold. In addition, TEMPH and TEMPL wake-up suspension can be used for RTC to trigger signal.

2.4.4. Advices for high-temperature and high-speed conditions

When chips run at high speed or high temperature, it is necessary to monitor junction temperature and JTM suspension in real time. If they are over thresholds, it is required to lower clock frequency.

2.5. Use of Secure JTAG

Secure JTAG function is only applicable to JTAG interfaces rather than SWD interfaces. In addition, the debugging interface is irreversibly switched from SWD interface to JTAG interface because the switch achieved by modifying EFUSE can't be modified once it is written into EFUSE.

The method of switching SWD debugging interface to normal JTAG interface:

Modify JTAGNSW and NDBG[1:0] bit fields of user control register (EFUSE_USER_CTL) of EFUSE module to 1 and 00 respectively. After the power supply is reset, only JTAG interface can be found.

The method of switching SWD debugging interface to secure JTAG interface:

When configuring as secure JTAG interface for debugging, it is required to configure the user key which should be used for unlocking when this method is used. Modify JTAGNSW and NDBG bit fields of user control register (EFUSE_USER_CTL) of EFUSE module and EFUSE_DP0 and EFUSE_DP0 of password register by writing security key into DP0[31:0] and DP1[31:0] and modifying JTAGNSW to 1 and NDBG[1:0] to 01. After the power supply is reset, JTAG interface can not be found. In such case, the host computer should send key to MCU with scripts.

How to use scripts:

First create SecureJTAG.JlinkScript file and copy the contents below to the file:

```
int InitTarget(void)
{
    int v;
    int v1;
    int v2;
    int v3;
    int v4;
    JLINK_CORESIGHT_Configure("IRPre=0;DRPre=0;IRPost=0;DRPost=0;IRLenDevice=5");
    JLINK_JTAG_Reset();

    JLINK_JTAG_WriteIR(0x15);
    JLINK_JTAG_StartDR();
    JLINK_JTAG_WriteDRCont(0xffffffff, 32);

    JLINK_SYS_Report("Writting secure jtag key1.................................");
    JLINK_JTAG_WriteDREnd(0x11223344, 32);

    JLINK_JTAG_WriteIR(0x16);
    JLINK_JTAG_StartDR();
    JLINK_JTAG_WriteDRCont(0xffffffff, 32);

    JLINK_SYS_Report("Writting secure jtag key2.................................");
    JLINK_JTAG_WriteDREnd(0x55667788, 32);

    JLINK_JTAG_WriteIR(0x18);
    JLINK_JTAG_StartDR();
    JLINK_SYS_Report("Reading secure jtag key1.................................");
```

```
        JLINK_JTAG_WriteDREnd(0xffffffff, 32);
        v1 = JLINK_JTAG_GetU32(0);
        JLINK_SYS_Report1("secure jtag key1:", v1);


        JLINK_JTAG_WriteIR(0x19);
        JLINK_JTAG_StartDR();
        JLINK_SYS_Report("Reading secure jtag key2................................");
        JLINK_JTAG_WriteDREnd(0xffffffff, 32);
        v2 = JLINK_JTAG_GetU32(0);
        JLINK_SYS_Report1("secure jtag key2:", v2);


        JLINK_JTAG_WriteIR(0x1e);
        JLINK_JTAG_StartDR();
        JLINK_SYS_Report("Reading boundary scan ID................................");
        JLINK_JTAG_WriteDREnd(0xffffffff, 32);
        v3 = JLINK_JTAG_GetU32(0);
        JLINK_SYS_Report1("boundary scan ID:", v3);


        JLINK_JTAG_WriteIR(0x1a);
        JLINK_JTAG_StartDR();
        JLINK_SYS_Report("Reading secure jtag state................................");
        JLINK_JTAG_WriteDREnd(0xffffffff, 32);
        v4 = JLINK_JTAG_GetU32(0);
        JLINK_SYS_Report1("secure jtag state:", v4);


        return 0;
    }
```

0X11223344 in the file is the value written into DP0[31:0] when users set the key. 0X55667788 is the value written into DP1[31:0] when users set the key. 0x15 and 0x16 commands in the file represent commands of writing keys. 0x18 and 0x19 commands represent reading written keys. 0X1e command represents reading mcu device ID. 0X1a command represents reading the status of secure jtag and wrong_seq signs.

Then create secure JTAG.bat file in the same path as that of SecureJTAG1.JlinkScript and copy the contents below to the file:

```
PATH=%PATH%;D:\Keil_v528\ARM\Segger;
jlink.exe -JLinkScriptFile SecureJTAG1.JlinkScript -device Cortex-M7 -if JTAG -speed 100 -autoconnect 1 -JTAGConf -1,-1
pause
```

D:\Keil_v528\ARM\Segger is the directory of jlink.exe.

Finally, execute secureJTAG.bat and JTAG debugging is unlocked and ready to use.

Note:

1. If the password is entered in error, it is required to reset the power supply.

2. If there is any error in entering the sequence, it is required to reset the power supply for decryption.

3. Entering correct password to open debugging only applies to SPC_L or below. ROM, RAM secure access mode, and SPC_H would not be opened through this operation.

2.6. Jlink debug issues

When debugging GD32H7XX chips with KEIL IDE, if address memory interface driven by EXMC is initiated in the course, KEIL IDE will be stuck when the interface is opened again and it is required to reset window settings.

2.7. Instruction of cache use

2.7.1. Data consistency issues when using Dcache and DMA simultaneously

GD32H7xx provides high-speed cache that allows read assignment. When cache is not hit, data will be assigned with high-speed cache lines and 32-byte data is saved from the master storage to the cache. Subsequently, when accessing these master storage addresses, cache will be hit to read the data directly. When cache is enabled on SRAM, there will be some consistency-related issues when CPU and DMA access SRAM simultaneously.

CPU reads the data DMA writes into SRAM.

When DMA reads data from peripheral and updates into the receive buffer destination[] in SRAM, CPU attempts to read the data in destination[]. It will read the existing data in cache rather than new data in SRAM.

Solutions are as below:

1. After DMA receives data, conduct invalid operation on destination[] in cache. destination[] in cache will be invalid through the operation below. When CPU attempts to read destination[], cache will not be hit.

```
    /* invalidate the cache line */
    SCB_InvalidateDCache_by_Addr((uint32_t *)destination, BUFFER_SIZE * 2);
```

2. To ensure that cache line margins are aligned, destination[] must be 32 bytes aligned.

```
__attribute__((aligned(32))) uint16_t destination[BUFFER_SIZE];
```

DMA reads the data CPU writes into SRAM.

When CPU is updating the data to be transmitted in the transmit buffer welcome[], it will update the data in cache only rather than the data in SRAM. When DMA reads the data in welcome[], it will read the data from SRAM rather than new data in cache updated by CPU.

Solutions are as below:

1. Before initiating DMA transmission, it is required to clear cache and refresh welcome[] in cache to SRAM.

```
/* clean the cache lines */
SCB_CleanDCache_by_Addr((uint32_t *)welcome, BUFFER_SIZE);
/* configure DMA1 */
dma_config();
```

2. The address saved in welcome[] must be 32 bytes aligned.

```
__attribute__((aligned(32))) uint8_t welcome[BUFFER_SIZE] = "Hi, this is an example: RAM_TO_USART by DMA !\n";
```

2.7.2. Use of cache and data alignment configuration

The table below lists some ARM CMSIS Dcache operating functions recommended on the official website.

Table 2-2. API for cache operation

API for cache operation	Description
void SCB_EnableDCache (void)	After invalidating the whole data cache, enable data cache.
void SCB_DisableDCache (void)	After cleaning the whole data cache, disable data cache.
void SCB_CleanDCache(void)	Clean the whole data cache.
void SCB_CleanDCache_by_Addr (uint32_t *addr, int32_t dsize)	Clean data cache lines by address.
void SCB_InvalidateDCache(void)	Invalidate the whole data cache.
void SCB_InvalidateDCache_by_Addr (uint32_t * addr, int32_t dsize)	Invalidate data cache lines by address.
void SCB_CleanInvalidateDCache(void)	Clean and invalidate the whole data cache.
void SCB_CleanInvalidateDCache_by_Addr(uint32_t *addr, int32_t dsize)	Clean and invalidate data cache lines by address.

Note: Since the data cache line has 32 bytes and the cache is read and written in lines, addr must have 32 bytes with margin aligned and dsize must be an integer multiple of 32 bytes.

Cleaning cache aims to forcibly write lines marked as dirty (rewritten) in cache to SRAM, clean the sign of dirty lines, and rebuild consistency between cache and SRAM. This avoids failing to timely capture the latest data in cache when SRAM is read by DMA.

Invalidating cache aims to invalidate cache lines by clearing the significant bit in cache lines only rather than clearing the data in cache lines. This avoids CPU from capturing data in cache instead of the latest data after SRAM data is updated.

In addition, to avoid consistency issues, the Memory Protection Unit (MPU) can be used to define SRAM area shared by CPU and DMA as not using the data cache but keep SRAM which can only be accessed by CPU using the data cache. For this method, please refer to

MPU configuration use document of ARM.

2.8. Use of CAN filter

The CAN bus controller is integrated with a flexibly configured mailbox system for sending and receiving CAN frame. The mailbox system consists of one set of mailboxes (maximumly 32 mailboxes) for storing control data, time stamps, message identifiers, and message data. Each CAN controller has 512-byte RAM space for configuration of mailbox or FIFO descriptor.

When receive FIFO is enabled in CAN, the RAM space occupied by the mailbox will be used for receiving FIFO descriptors. With the identifier filtering function, the receive FIFO can maximumly filter 104 extended identifiers, 208 standard identifiers, or 8 bits of 416 identifiers. Maximum 32 identifier filtering table elements can be configured through the private filtering register of receive FIFO/mailbox.

When Rx FIFO is disabled:

- If RPFQEN bit of CAN_CTL0 register is 0, then CAN_RMPUBF register is used for all receive mailboxes.
- If RPFQEN bit of CAN_CTL0 register is 1, then CAN_RFIFOMPFx (x = 0..31) register is used for receive mailboxes individually.

When Rx FIFO is enabled:

- If RPFQEN bit of CAN_CTL0 register is 0, then CAN_RMPUBF register is used for all receive mailboxes, and CAN_RFIFOPUBF and CAN_RFIFOMPFx (x = 0..31) registers are used to configure all Rx FIFO identifier filtering table elements. Values of all these registers must be configured the same.
- If RPFQEN bit of CAN_CTL0 register is 1, then CAN_RFIFOMPFx (x=0..31) register is used to configure Rx FIFO identifier filtering table elements set by RFFN[3:0] bit field of CAN_CTL2 register and receive mailboxes (as descriptors of receive mailboxes and Rx FIFO can not occupy RAM of the same area at the same time, a set of registers are used to configure filtering data under separate control), and CAN_RFIFOPUBF register is used to configure all remaining Rx FIFO identifier filtering table elements.

Configuration of the number of Rx FIFO identifier filtering table elements is as listed in the table below:

RFFN[3:0]	Number of Rx FIFO identifier filtering table elements	Space occupied by Rx FIFO	Mailbox available
0000	8	Mailbox descriptor 0 - 7	Mailbox 8- 31
0001	16	Mailbox descriptor 0 - 9	Mailbox 10- 31
0002	24	Mailbox descriptor 0 - 11	Mailbox 12- 31
0003	32	Mailbox descriptor 0 - 13	Mailbox 14- 31
0004	40	Mailbox descriptor 0 - 15	Mailbox 16- 31

RFFN[3:0]	Number of Rx FIFO identifier filtering table elements	Space occupied by Rx FIFO	Mailbox available
0005	48	Mailbox descriptor 0 - 17	Mailbox 18- 31
0006	56	Mailbox descriptor 0– 19	Mailbox 20- 31
0007	64	Mailbox descriptor 0– 21	Mailbox 22- 31
0008	72	Mailbox descriptor 0– 23	Mailbox 24- 31
0009	80	Mailbox descriptor 0– 25	Mailbox 26- 31
000A	88	Mailbox descriptor 0– 27	Mailbox 28- 31
000B	96	Mailbox descriptor 0– 29	Mailbox 30- 31
000C	104	Mailbox descriptor 0– 31	None
Others	104	Mailbox descriptor 0 - 31	None

By taking RFFN[3:0] = 3 as an example, at this time, the number of Rx FIFO identifier filtering table elements is 32, the space of mailbox descriptor 0-13 is occupied by FIFO descriptor, and the remaining mailboxes14-31 can be used for sending and receiving for mailbox. There are three formats for Rx FIFO identifier filtering table elements, which can be configured by FS[1:0] bit field of CAN_CTL0 register. When FS[1:0] is 0, Rx FIFO identifier filtering table elements are in format A and allow maximum 32 complete standard or extended identifiers. When FS[1:0] is 1, they are in format B and allow maximum 64 complete standard or 14 bits of extended identifiers. When FS[1:0] is 2, they are in format C and allow maximum 128 complete standard or 8 bits of extended identifiers.

Configuration of the format of Rx FIFO identifier filtering table elements is as below:

The Rx FIFO identifier filtering table elements configured in format B allows 64 complete standard or 14 bits of extended identifiers.

```
can_fifo_parameter.filter_format_and_number = CAN_RXFIFO_FILTER_B_NUM_64;
/* used for 26-63nd IDs filtering configuration */
can_fifo_parameter.fifo_public_filter = 0xFFFFFFFF;
can_rx_fifo_config(CAN0, &can_fifo_parameter);
```

Because FS[1:0] is 3 and mailboxes 0-13 are occupied by FIFO, private filters 0-13 of receive FIFO/mailbox are used to receive and filter FIFO. The code configuration is as below:

```
for(i=0; i<=13; i++){
    can_private_filter_config(CAN0, i, 0xFFFFFFFF);
}
```

2.9. Hardfault issue arising from non-aligned access to EXMC SDRAM

In default situation, when access property of SDRAM has not been modified, hardfault will occur when accessing SDRAM in non-aligned way for 0xC0000000-0xDFFFFFFF defaults to device type and does not allow non-aligned access. For details, please refer to the table below.

Table2-3. ARMv7-M address mapping

Address	Name	RAM type (s)	XN	Cache	Description/ supported storage
0xE0000000- 0xFFFFFFFF	System	Device & Strongly Ordered	XN	-	Vendor system region (VENDOR_SYS) Private Peripheral Bus (PPB)
0xC0000000- 0xDFFFFFFF	Device	Device	XN	-	Non-shareable memory
0xA0000000- 0xBFFFFFFF	Device	Device, Shareable	XN	-	Shareable memory
0x80000000- 0x9FFFFFFF	RAM	Normal	-	WT	Memory with WT cache attributes
0x60000000- 0x7FFFFFFF	RAM	Normal	-	WBWA	Write-back, Write-allocate L2/L3
0x40000000- 0x5FFFFFFF	Peripheral	Device	XN	-	On-chip peripheral address space
0x20000000- 0x3FFFFFFF	SRAM	Normal	-	WBWA	SRAM On-chip RAM
0x00000000- 0x1FFFFFFF	Code	Normal	-	WT	ROM Flash Memory

To access SDRAM in non-aligned way, it is required to modify the type of the address space where SDRAM is located with MPU, like the same type as that of RAM. For example, users can configure Write Through property for 0xC0000000.

```
mpu_region_init_struct mpu_init_struct;
mpu_region_struct_para_init(&mpu_init_struct);

/* disable the MPU */
ARM_MPU_Disable();
ARM_MPU_SetRegion(0, 0);

/* configure the MPU attributes for SDRAM */
mpu_init_struct.region_base_address = 0xC0000000;
mpu_init_struct.region_size          = MPU_REGION_SIZE_32MB;
mpu_init_struct.access_permission    = MPU_AP_FULL_ACCESS;
mpu_init_struct.access_bufferable    = MPU_ACCESS_NON_BUFFERABLE;
mpu_init_struct.access_cacheable     = MPU_ACCESS_CACHEABLE;
mpu_init_struct.access_shareable     = MPU_ACCESS_NON_SHAREABLE;
mpu_init_struct.region_number        = MPU_REGION_NUMBER0;
mpu_init_struct.subregion_disable    = MPU_SUBREGION_ENABLE;
mpu_init_struct.instruction_exec     = MPU_INSTRUCTION_EXEC_PERMIT;
mpu_init_struct.tex_type             = MPU_TEX_TYPE0;
mpu_region_config(&mpu_init_struct);
mpu_region_enable();
```

```
/* enable the MPU */
ARM_MPU_Enable(MPU_MODE_PRIV_DEFAULT);
```

2.10. Precautions for SAI using DMA burst transfer to send data

Each audio sub-module of GD32H7xx SAI has an 8-byte FIFO and DMA interface. Enabling DMA access is configured with DMA enable bit (DMAEN) of SAI_BxCFG0 register. DMA request and FIFO request (FFREQ) are generated together. The generation status of FIFO request depends on FIFO threshold (FFTH) and FIFO status (FFSTAT), which is important when using DMA burst transfer.

The table below lists the recommended relationship between DMA burst transfer and SAI FIFO threshold.

DMA transfer width	DMA burst transfer type	SAI data width	SAI FIFO threshold
16	Single	16	Empty, 1/4 full, half-full, or 3/4 full
	4-beat incremental burst transfer		Empty or 1/4 full
32	Single	32	Empty, 1/4 full, half-full, or 3/4 full
	4-beat incremental burst transfer		Empty or 1/4 full

When the audio sub-module is configured in transmitting mode, FIFO threshold must be set as a designated value to ensure that there is enough remaining space to achieve a complete DMA burst writing in the worst situation. Otherwise, FIFO overflow error might happen. When the audio sub-module is configured in receiving mode, FIFO threshold must be set as a designated value to ensure that there is enough remaining space to achieve a complete DMA burst reading in the worst situation to prevent FIFO underflow error.

2.11. Enabling situation of ENET cache

ENET module uses DMA mechanism while cache of ENET module is enabled. Due to high data traffic, ENET module will update RAM through DMA rapidly, which may lead to inconsistency between the data captured by DMA and the data in cache.

Therefore, we first need to specify the address of the relevant memory area, including the ENET RxDMA /TxDMA descriptor, LWIP RAM heap, and transmit / receive buffer.

```
#if defined(GD32H7XX)

#if defined  (__CC_ARM)                                    /*!< ARM compiler */
__attribute__((section(".ARM.__at_0x30000000"))) enet_descriptors_struct  rxdesc_tab[ENET_RXBUF_NUM];      /*!< ENET RxDMA descriptor */
__attribute__((section(".ARM.__at_0x30000160"))) enet_descriptors_struct  txdesc_tab[ENET_TXBUF_NUM];      /*!< ENET TxDMA descriptor */
__attribute__((section(".ARM.__at_0x30000300"))) uint8_t rx_buff[ENET_RXBUF_NUM][ENET_RXBUF_SIZE];         /*!< ENET receive buffer */
__attribute__((section(".ARM.__at_0x30002100"))) uint8_t tx_buff[ENET_TXBUF_NUM][ENET_TXBUF_SIZE];         /*!< ENET transmit buffer */

#elif defined ( __ICCARM__ )                               /*!< IAR compiler */
#pragma location=0x30000000
enet_descriptors_struct  rxdesc_tab[ENET_RXBUF_NUM];       /*!< ENET RxDMA descriptor */
#pragma location=0x30000160
enet_descriptors_struct  txdesc_tab[ENET_TXBUF_NUM];       /*!< ENET TxDMA descriptor */
#pragma location=0x30000300
uint8_t rx_buff[ENET_RXBUF_NUM][ENET_RXBUF_SIZE];          /*!< ENET receive buffer */
#pragma location=0x30002100
uint8_t tx_buff[ENET_TXBUF_NUM][ENET_TXBUF_SIZE];          /*!< ENET transmit buffer */

#elif defined (__GNUC__)        /* GNU Compiler */
enet_descriptors_struct  rxdesc_tab[ENET_RXBUF_NUM] __attribute__((section(".ARM.__at_0x30000000")));       /*!< ENET RxDMA descriptor */
enet_descriptors_struct  txdesc_tab[ENET_TXBUF_NUM] __attribute__((section(".ARM.__at_0x30000160")));       /*!< ENET TxDMA descriptor */
uint8_t rx_buff[ENET_RXBUF_NUM][ENET_RXBUF_SIZE] __attribute__((section(".ARM.__at_0x30000300")));          /*!< ENET receive buffer */
uint8_t tx_buff[ENET_TXBUF_NUM][ENET_TXBUF_SIZE] __attribute__((section(".ARM.__at_0x30002100")));          /*!< ENET transmit buffer */

#endif /* __CC_ARM */
```

```
/* Relocate the LwIP RAM heap pointer */
#define LWIP_RAM_HEAP_POINTER    (0x30004000)
```

In addition, we need to set cache access permit of the RAM area in use with MPU module as no buffer and no cache to ensure real-time property and accuracy of data.

```c
void mpu_config(void)
{
    mpu_region_init_struct mpu_init_struct;
    mpu_region_struct_para_init(&mpu_init_struct);

    /* disable the MPU */
    ARM_MPU_SetRegion(0U, 0U);

    /* Configure the DMA descriptors and Rx/Tx buffer*/
    mpu_init_struct.region_base_address = 0x30000000;
    mpu_init_struct.region_size = MPU_REGION_SIZE_16KB;
    mpu_init_struct.access_permission = MPU_AP_FULL_ACCESS;
    mpu_init_struct.access_bufferable = MPU_ACCESS_BUFFERABLE;
    mpu_init_struct.access_cacheable = MPU_ACCESS_NON_CACHEABLE;
    mpu_init_struct.access_shareable = MPU_ACCESS_NON_SHAREABLE;
    mpu_init_struct.region_number = MPU_REGION_NUMBER0;
    mpu_init_struct.subregion_disable = MPU_SUBREGION_ENABLE;
    mpu_init_struct.instruction_exec = MPU_INSTRUCTION_EXEC_PERMIT;
    mpu_init_struct.tex_type = MPU_TEX_TYPE0;
    mpu_region_config(&mpu_init_struct);
    mpu_region_enable();

    /* Configure the LwIP RAM heap */
    mpu_init_struct.region_base_address = 0x30004000;
    mpu_init_struct.region_size = MPU_REGION_SIZE_16KB;
    mpu_init_struct.access_permission = MPU_AP_FULL_ACCESS;
    mpu_init_struct.access_bufferable = MPU_ACCESS_NON_BUFFERABLE;
    mpu_init_struct.access_cacheable = MPU_ACCESS_NON_CACHEABLE;
    mpu_init_struct.access_shareable = MPU_ACCESS_SHAREABLE;
    mpu_init_struct.region_number = MPU_REGION_NUMBER1;
    mpu_init_struct.subregion_disable = MPU_SUBREGION_ENABLE;
    mpu_init_struct.instruction_exec = MPU_INSTRUCTION_EXEC_PERMIT;
    mpu_init_struct.tex_type = MPU_TEX_TYPE1;
    mpu_region_config(&mpu_init_struct);
    mpu_region_enable();

    /* enable the MPU */
    ARM_MPU_Enable(MPU_MODE_PRIV_DEFAULT);
}
```
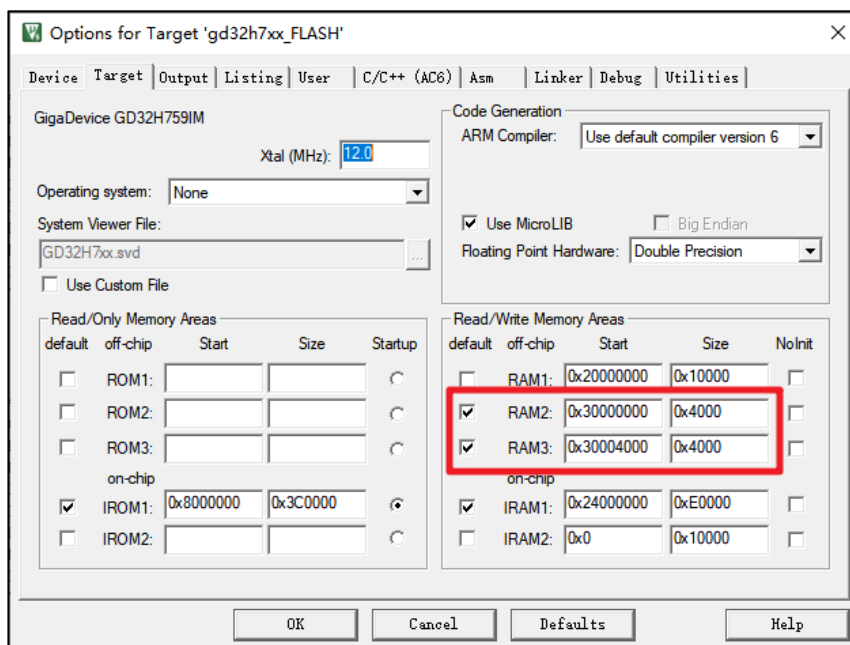
At the same time, it is required to select the set RAM area in related project settings. Otherwise, the above configuration will be invalid.

2.12. Precautions for Bootloader operation

For details, please refer to ***Precautions for AN126_GD32H7xx_BootLoader Operations.***

2.13. Precautions for the Use of USBHS

For details, please refer to ***Precautions for the Use of USBHS of AN117 GD32H7xx Series***.

2.14. Precautions for the Use of SDIO

2.14.1. SDIO clock configuration

CK_SDIO is a kernel clock of SDIO module. SDIO_CLK provided to SD/eMMC card is obtained through frequency division of the clock. When configuring CK_SDIO through RCU, it is recommended to configure the frequency of the working clock CK_SDIO to up to 208 Mhz.

When configuring AF function with pins, it is recommended to modify GPIOx_AFSELx(x=0, 1) first and then switching GPIO mode into AF.

```
static void gpio_config(void)
{
    /* configure the SDIO_DAT0(PB13), SDIO_DAT1(PC9), SDIO_DAT2(PC10), SDIO_DAT3(PC11), SDIO_CLK(PC12) and SDIO_CMD(PD2) */
    gpio_af_set(GPIOB, GPIO_AF_12, GPIO_PIN_13);
    gpio_af_set(GPIOC, GPIO_AF_12, GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12);
    gpio_af_set(GPIOD, GPIO_AF_12, GPIO_PIN_2);

    gpio_mode_set(GPIOB, GPIO_MODE_AF, GPIO_PUPD_PULLUP, GPIO_PIN_13);
    gpio_mode_set(GPIOC, GPIO_MODE_AF, GPIO_PUPD_PULLUP, GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11);

    gpio_output_options_set(GPIOB, GPIO_OTYPE_PP, GPIO_OSPEED_100_220MHZ, GPIO_PIN_13);
    gpio_output_options_set(GPIOC, GPIO_OTYPE_PP, GPIO_OSPEED_100_220MHZ, GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11);

    gpio_mode_set(GPIOC, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO_PIN_12);
    gpio_output_options_set(GPIOC, GPIO_OTYPE_PP, GPIO_OSPEED_100_220MHZ, GPIO_PIN_12);

    gpio_mode_set(GPIOD, GPIO_MODE_AF, GPIO_PUPD_PULLUP, GPIO_PIN_2);
    gpio_output_options_set(GPIOD, GPIO_OTYPE_PP, GPIO_OSPEED_100_220MHZ, GPIO_PIN_2);
}
```
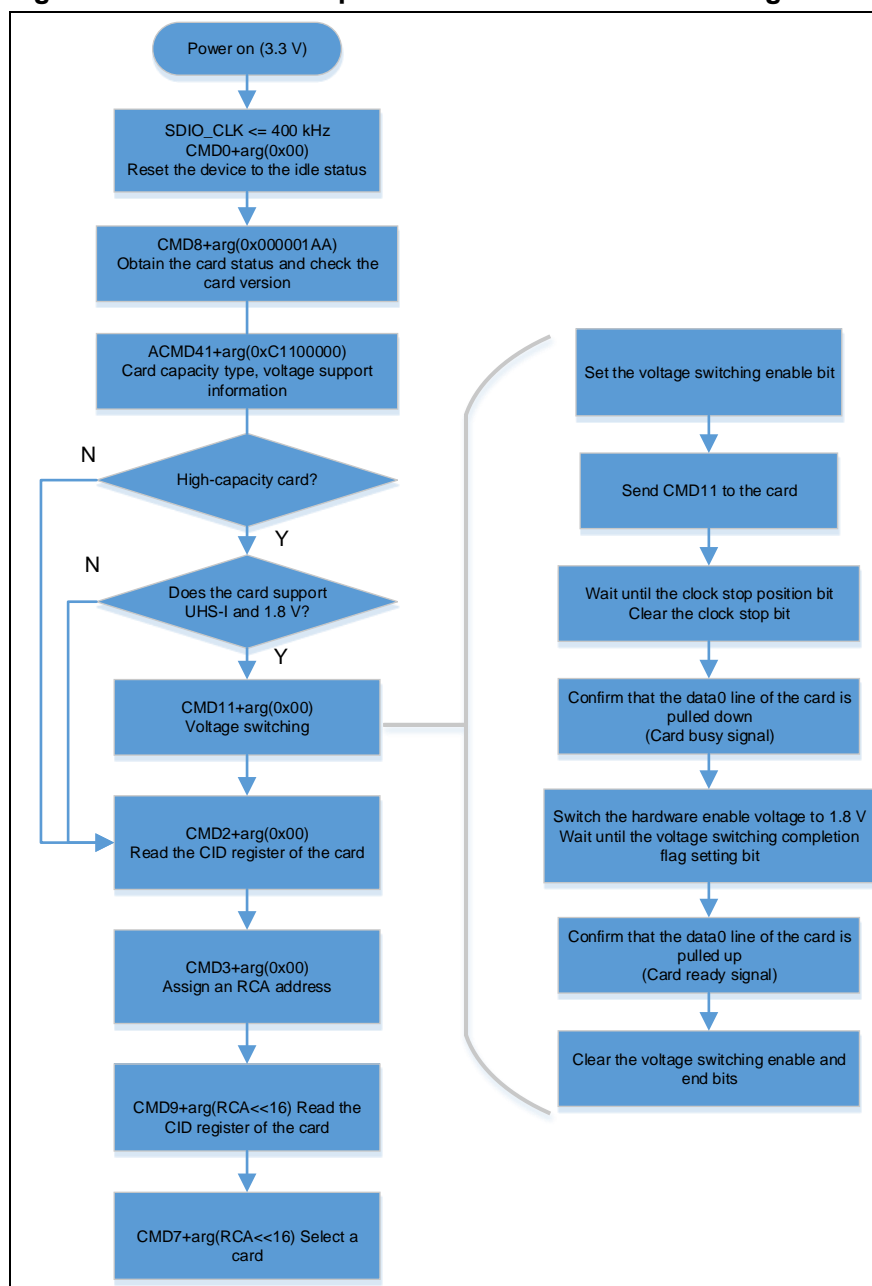
2.14.2.   SDIO power-on initialization

Power-on initialization and voltage switch of UHS-I SD card

SD 3.0 card allows UHS-I (Ultra High Speed Bus Speed Mode phase I) speed mode, including SDR12, SDR25, SDR50, SDR104, and DDR50. The card works in UHS-I at 1.8 V while the card is powered on at 3.3 V, so UHS-I mode allows the voltage switch from 3.3 V to 1.8 V. When the voltage switch sequence is completed successfully, SDR12 card will enter into UHS-I mode by default.
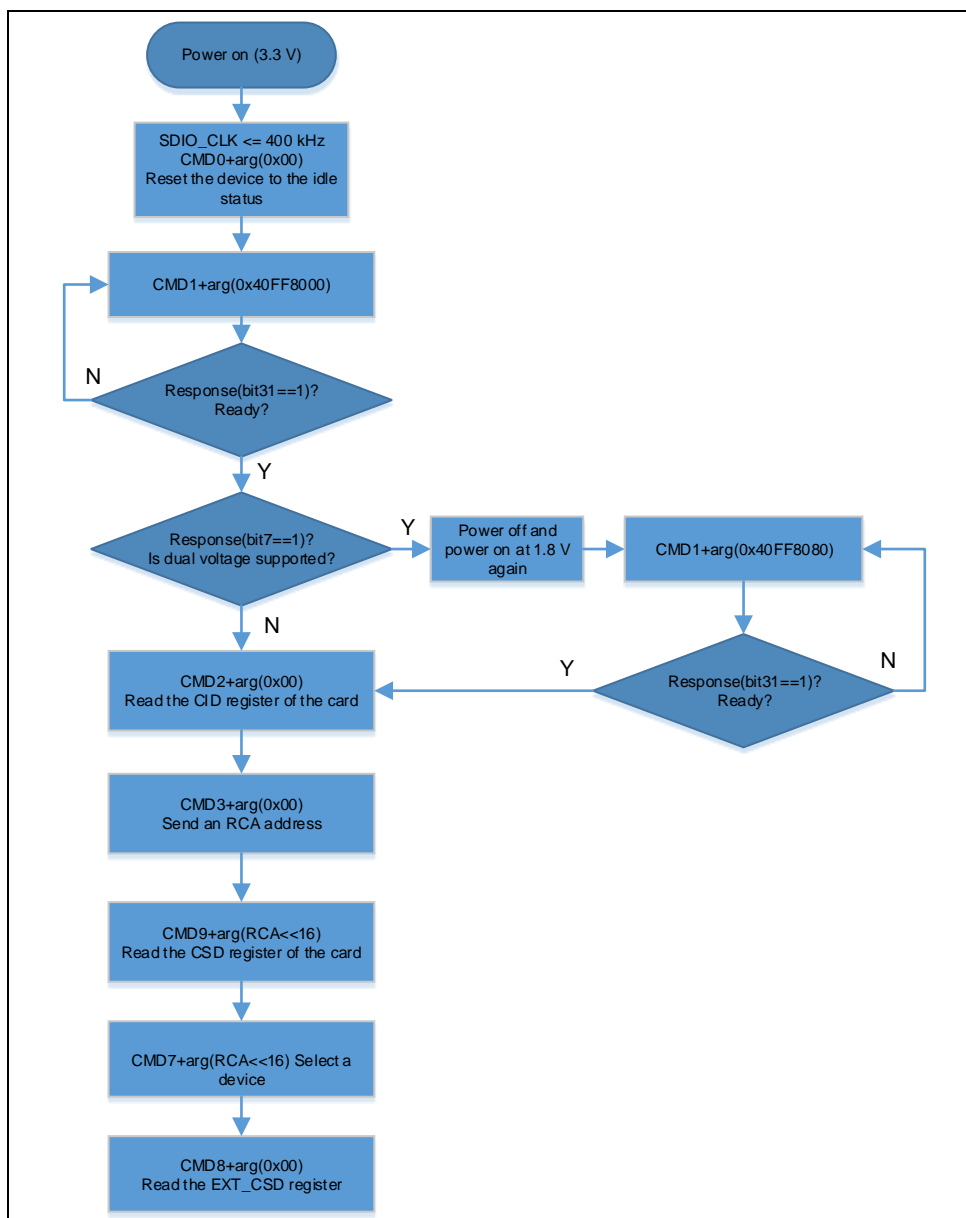
Power-on initialization of the card and capture of card status and information can be completed for cards that can work in UHS-I. When responding to ACMD41, the card will reply with the support of 1.8 V voltage switch and then proceed with 1.8 V voltage switch operation. Main processes are shown in the figure below. In addition, please note that SDIO0 can work in UHS-I but SDIO1 can not work in UHS-I. In other words, SDIO1 can not generate the chip signal to control external voltage switch.

Figure 2-1. Processes of power-on initialization and voltage switch of SD card

Power-on initialization of eMMC

Power-on processes of eMMC are shown in the figure below. First, the host determines whether eMMC can work at 1.8 V with OCR register, and then the host will select 1.8 V to power on eMMC again.

Figure 2-2. Processes of power-on initialization and voltage switch of eMMC

2.14.3. Bus width configuration

When the host sends a command to set device bus speed, it is required to change the width, speed mode, and DDR of the host after 64 byte-data is received.

After power-on initialization of SD card is completed, configuration of bus width and speed can be done by sending commands according to the table below. Please note that 1-bit bus is not allowed in UHS-I mode.

Table 2-4. Configuring bus width of SD card

Selected width	Command	Parameters
1bit	ACMD6	0x00
4bit		0x02

Table2-5. Configuring bus speed of SD card

Selected speed mode	Voltage/V	Command	Parameters
SDR12	1.8	CMD6	0x80FFFF00
SDR25	1.8		0x80FFFF01
SDR50	1.8		0x80FF1F02
SDR104	1.8		0x80FF1F03
DDR50	1.8		0x80FF1F04

After power-on initialization of eMMC is completed, configuration of bus width and speed can be done by sending commands according to the table below.

Table2-6. Configuring bus mode of eMMC

Selected width	Command	Parameters
1BIT SDR	CMD6	0x03B70000
4BIT SDR		0x03B70100
8BIT SDR		0x03B70200
4BIT DDR		0x03B70500
8BIT DDR		0x03B70600

Table2-7. Configuring bus speed of eMMC

Selected speed mode	Voltage/V	Command	Parameters
DS	1.8/3.3	CMD6	0x03B90000
HS	1.8/3.3		0x03B90100
HS200	1.8		0x03B90200

High-speed bus tuning of SD card

When SDR50 and SDR104 that can work in UHS-I is in bus speed mode, CPDM module can be used for tuning of sampling points of receiving data. After CPDM is used, it is required to disable hardware flow control. Before sending CMD19 for tuning, it is required to configure delay line length with CPDM.

Reference codes for configuring delay line length with CPDM are as below.

```
void cpdm_config(void)
{
    /* apply the delay settings */
    CPDM_CTL(CPDM_SDIO0) = CPDM_CTL_CPDMEN | CPDM_CTL_DLSEN;
    temp = CPDM_MAX_PHASE ;

    for(i = 0; i < ((uint32_t)0x00000080U); i++) {
        CPDM_CFG(CPDM_SDIO0) = temp | (i << 8);

        while((CPDM_CFG(CPDM_SDIO0) & CPDM_CFG_DLLENF) == SET);
        while((CPDM_CFG(CPDM_SDIO0) & CPDM_CFG_DLLENF) == RESET);

        if((((CPDM_CFG(CPDM_SDIO0) >> 16) & 0x7FF) > 0) &&
           (((CPDM_CFG(CPDM_SDIO0) & 0x04000000) == RESET) || ((CPDM_CFG(CPDM_SDIO0) & 0x08000000) == RESET))) {
            temp = CPDM_CFG(CPDM_SDIO0);
            temp &= 0xFFFFFFF0;

            CPDM_CFG(CPDM_SDIO0) = temp | 0x0A ;
            CPDM_CTL(CPDM_SDIO0) = CPDM_CTL_CPDMEN;
            break;
        }
    }
}
```
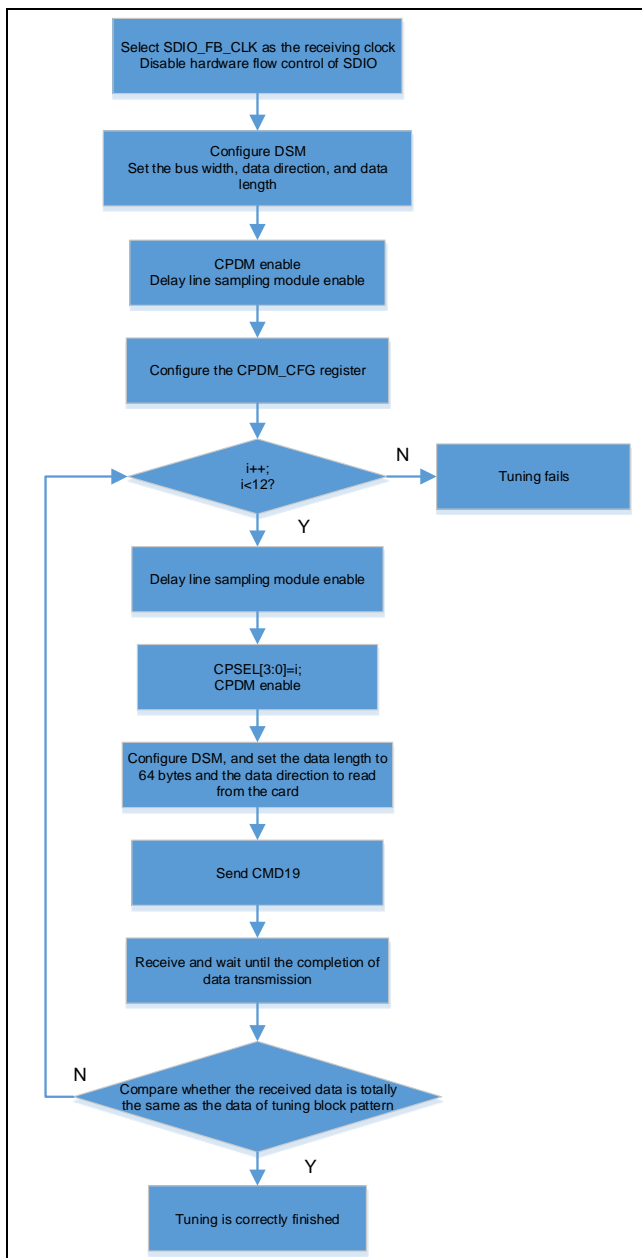
Figure 2-3. Tuning processes of SD card

2.14.4. Accessing eMMC Boot partition data

eMMC has two Boot partitions in the same size, which is obtained by calculating EXT_CSD register according to the calculation formula EXT_CSD[226]*128K byte. EXT_CSD register is obtained by the host sending CMD8. Data reading and writing in Boot partition is the same as that in other partitions, but before sending reading and writing commands, the host should send CMD6 access to Boot partition. Detailed command is listed in the table below. The data written into Boot partition can be read automatically upon power-on initialization. Detailed operations are conducted according to protocols. Configuration of reading content bus of Boot partition is also required to be done according to protocols beforehand.

Table2-8. Accessing partition commands and parameters

| Switched partition	Command	Parameters
User partition		0x03B30000
Boot partition 1	CMD6	0x03B30100
Boot partition 2		0x03B30200

2.14.5. MDMA configuration

Controlling SDIO transfer process with MDMA

When MDMA is not used, CPU will control SDIO transfer process. For example, after transferring multiple blocks of data, send CMD12 stop command. After one dual-buffer transfer in buffer is done, modify buffer address configuration in IDMA.

If MDMA is used, it can control SDIO transfer in place of CPU in three time periods, including the end of data transfer (SDIO_DATA_END), the end of command sequence (SDIO_CMD_END), and the end of buffer transfer (SDIO_BUF_END). When MDMA is triggered at these moments, it will transfer pre-configured static data to SDIO register to achieve SDIO control.

The flow chart below shows the example of MDMA controlling the sending of CMD12 after SDIO transfers multiple blocks of data.

Figure 2-4. Flow chart of MDMA controlling the sending of CMD12
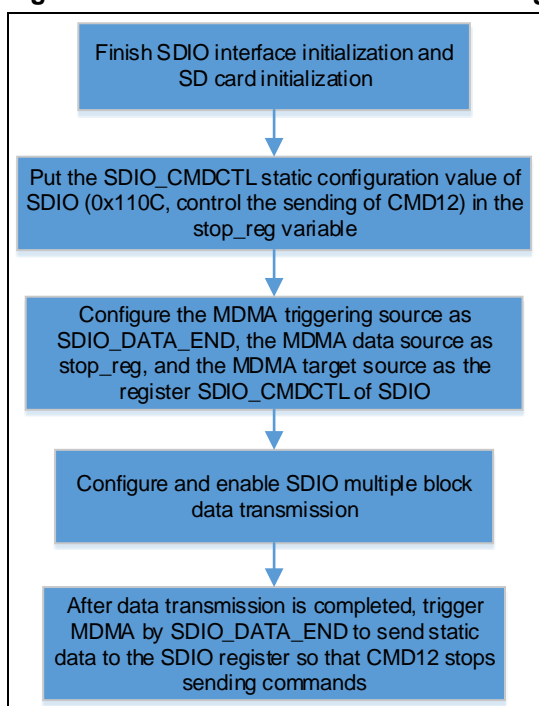
In addition, MDMA supports the linked list, so it is possible to conduct transfer control on SDIO for many times with multi-node linked list.

Transferring data to DTCM/ITCM with MDMA

The figure below is an example of using SDIO and MDMA at the same time. It aims to configure SDIO_DATA_END as a source to trigger MDMA to transfer SD card data to ITCM.

Figure 2-5. Flow chart of MDMA controlling RAM data transfer
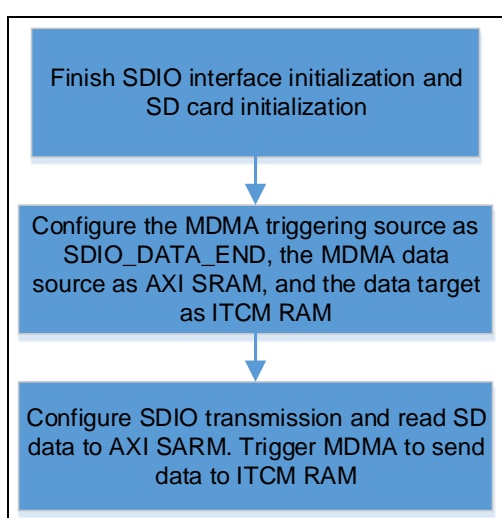
2.14.6. IDMA configuration

There is DMA (IDMA) in SDIO which supports single buffer transfer and dual buffer transfer.

In single buffer transfer mode, it is only required to configure SDIO register SDIO_IDMACTL in single buffer mode and enable IDMA before transfer so that single buffer transfer can be achieved with IDMA.

In dual buffer transfer mode, SDIO will read or write data in two buffers in turn. When SDIO is using one buffer, CPU can read or write data in the other buffer. It is required to configure SDIO register SDIO_IDMACTL in dual buffer mode and enable IDMA, configure SDIO_IDMASIZE as the buffer size, configure SDIO_IDMAADDR0 and SDIO_IDMAADDR1 as the addresses of the two external buffers, and choose to enable IDMAEND interruption. By enabling SDIO data transfer, SDIO will access two buffers in turn. When transfer is done in one buffer, IDMAEND interruption will be triggered, and CPU can access the other buffer that is not used by SDIO by judging it according to BUFSEL in SDIO_IDMACTL.

3. Revision history

Table 3-1. Revision history

Revision No.	Description	Date
1.0	Initial release	May 9, 2023

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 under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. 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.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which have been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do 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 Products. Customers shall indemnify and hold the Company 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 Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.