GigaDevice Semiconductor Inc.

GD32H77x_78x Series
Software Development Guide

Application Note
AN310

Revision 1.0

(Nov. 2026)

Table of Contents

Table of Contents 1
List of Figures 3
List of Tables 4
1. Introduction 5
2. Module usage instructions 6
2.1. System and memory architecture 6
2.1.1. Selection and configuration of Boot modes 6
2.1.2. Instruction of cache use 7
2.1.3. Bootloader operation notes 9
2.2. Non-volatile memory controller (NVMC) 9
2.2.1. Non-volatile memory structure and configuration recommendations 9
2.2.2. Error check and correction (ECC) 9
2.2.3. Erase characteristics of CNVM and ECNVM 11
2.2.4. Option bytes 11
2.2.5. Precautions for using OTP 14
2.3. Power Management Unit (PMU) 15
2.3.1. The POR_ON pin 16
2.3.2. Sleep mode downloader connection issue 16
2.3.3. Link issue between Pxx pins and Pxx_C pins in standby mode 16
2.3.4. Software configuration 16
2.4. Reset and clock unit (RCU) 16
2.5. Debug (DBG) 18
2.5.1. Secure JTAG usage 18
2.5.2. Jlink debugging issue 20
2.6. Analog-to-digital converter (ADC) 20
2.6.1. Two methods for monitoring junction temperature 20
2.6.2. Method for monitoring junction temperature with ADC 21
2.6.3. JTM alarm function 21
2.6.4. Advices for high-temperature and high-speed conditions 21
2.7. DSI host controller (DSI) 21
2.7.1. PMU configuration 21
2.7.2. Clock configuration 22
2.8. Resolver-to-Digital Conversion Module (RDCM) 22
2.8.1. RDCM Sine Wave Exciter Configuration 22
2.8.2. RDCM mixer configuration 23
2.8.3. RDCM demodulator configuration 26

2.8.4. RDCM group delay compensation configuration. 28
2.9. Controller area network (CAN) 29
2.9.1. Use of CAN filter 29
2.9.2. CANFD data segment communication parameters and TDCO configuration 31
2.10. Secure digital input/output interface (SDIO) 32
2.10.1. SDIO clock configuration 32
2.10.2. SDIO power-on initialization 32
2.10.3. Bus width configuration 34
2.10.4. Accessing eMMC Boot partition data 37
2.10.5. MDMA configuration 37
2.10.6. IDMA configuration 39
2.11. External memory controller (EXMC) 39
2.11.1. Hardfault issue arising from non-aligned access to EXMC SDRAM 39
2.12. Serial Audio Interface (SAI) 40
2.12.1. Precautions for SAI using DMA burst transfer to send data 40
2.13. Universal serial bus High-Speed interface (USBHS) 41
2.13.1. Precautions for the Use of USBHS 41
2.14. Ethernet (ENET) 42
2.14.1. Enabling situation of ENET cache 42
2.15. EtherCAT SubDevice controller (ESC) 44
3. Revision history 45

List of Figures

Figure 2-1. DCache invalidation 7
Figure 2-2. 32-byte aligned destination array 7
Figure 2-3. Clear cache 8
Figure 2-4. 32-byte aligned welcome array 8
Figure 2-8. PMU mode configuration 16
Figure 2-9. RDCM analog sine wave output mode configuration 23
Figure 2-10. Example of RDCM PWM mode output configuration 23
Figure 2-11. RDCM mixer configuration example 26
Figure 2-12. Example of RDCM demodulator configuration 28
Figure 2-13. Configuration of the format of RxFIFO identifier filtering table elements 31
Figure 2-14. Configuration of the private filters 31
Figure 2-15. SDIO pin initialization configuration 32
Figure 2-16. Processes of power-on initialization and voltage switch of SD card 33
Figure 2-17. Processes of power-on initialization and voltage switch of eMMC 34
Figure 2-18. CPDM configuration 36
Figure 2-19. Tuning processes of SD card 36
Figure 2-20. Flow chart of MDMA controlling the sending of CMD12 38
Figure 2-21. Flow chart of MDMA controlling RAM data transfer 38
Figure 2-22. USBHS clock tree 41
Figure 2-23. ENET DMA descriptors and buffers definition 42
Figure 2-24. ENET DMA descriptors and buffers MPU configuration 43
Figure 2-25. ENET project configuration 43

List of Tables

Table 1-1. Applicable product 5
Table 2-1. Selection of boot mode 6
Table 2-2. API for cache operation 8
Table 2-4. Option bytes 12
Table 2-5. SecureJTAG.JlinkScript 18
Table 2-6. secureJTAG.bat 20
Table 2-7. Configuration of MIPI DSI PMU 21
Table 2-8. High-pass filter cutoff frequency 24
Table 2-9. CIC3 filter properties (N = I/QCICDFAC[8:0] + 1) 26
Table 2-10. Default coefficients for FIR0 27
Table 2-11. Default coefficients for FIR1 27
Table 2-12. Group delay of filter elements 28
Table 2-13. Configuration of the number of Rx FIFO identifier filtering table elements 30
Table 2-14. Differences in CANFD data segment communication parameters and TDCO bit-field width 31
Table 2-15. Configuring bus width of SD card 35
Table 2-16. Configuring bus speed of SD card 35
Table 2-17. Configuring bus mode of eMMC 35
Table 2-18. Configuring bus speed of eMMC 35
Table 2-19. Accessing partition commands and parameters 37
Table 2-20. ARMv7-M address mapping 39
Table 2-21. Configuration of EXMC SDRAM MPU 40
Table 2-22. DMA burst transfer and SAI FIFO threshold 41
Table 3-1. Revision history 45

1. Introduction

This application note is specifically written for the GD32H77x_78x series MCUs, aiming to provide functional examples and usage precautions for the peripheral resources of this series of chips, helping users quickly master the software development methods and practices for GD32H77x_78x series MCUs. The applicable products are as shown in ***Table 1-1. Applicable product***.

Table 1-1. Applicable product

Product series	Model
GD32H77x	GD32H779, GD32H77D, GD32H77E, GD32H77R series
GD32H78x	GD32H789, GD32H78D, GD32H78E series

Note: This application note is for reference only. In case of any conflict with the user manual or datasheet, the user manual or datasheet shall prevail.

2. Module usage instructions

2.1. System and memory architecture

2.1.1. Selection and configuration of Boot modes

GD32H77x_78x provides different Boot modes which are SECURITY BOOT, USER BOOT, SRAM BOOT, and SYSTEM BOOT. The value on the BOOT pin is latched on the 4th rising edge of CK_SYS after a reset. It is up to the user to set the BOOT pin after a power-on reset or a system reset to select the required boot source. Once the BOOT pin has been sampled, it is free and can be used for other purposes. The BOOT_ADDR0[15:0] and BOOT_ADDR1[15:0] address allows to configure the boot memory address to any address from 0x0000 0000 to 0x9000 0000. The boot mode can be obtained from the BOOT_MODE[2:0] bits in the SYSCFG_USERCFG register. For details, please refer to the table below.

Table 2-1. Selection of boot mode

SCR	SPC[7:0]	OTA(FW_UPDATE)	BOOT_ADDRESS (configured in BOOT_ADDRx(x = 0,1))	BOOT MODE	Boot From
1	x	x	XXXX	SECURITY BOOT	ROM
0	Protection level high	x	0x0800_0000~max ECNVM	USER BOOT	BOOT_ADDRESS
			0x7000_0000	USER BOOT	OSPI1
			0x9000_0000	USER BOOT	OSPI0
			other	USER BOOT	0x0800_0000
0	No protection / Protection level low	1	x	USER BOOT	Check_bootloader(1)
		0	0x1FF1_0000	USER BOOT	Check_bootloader(1)
		0	0x1FEF_0000	SYSTEM BOOT	BootLoader
		0	0x0800_0000~max ECNVM	USER BOOT	BOOT_ADDRESS
		0	0x7000_0000	USER BOOT	OSPI1
		0	0x9000_0000	USER BOOT	OSPI0
		0	0x0000_0000	SRAM BOOT(ITCM)	0x0000_0000
		0	0x2000_0000	SRAM BOOT(DTCM)	0x2000_0000
		0	0x2400_0000~ max AXI	SRAM BOOT(AXI	BOOT_ADDRES

SCR	SPC[7:0]	OTA(FW_UPDATE)	BOOT_ADDRESS (configured in BOOT_ADDRx(x = 0,1))	BOOT MODE	Boot From
			SRAM(ITCM/DTCM/AXI)	SRAM)	S
		0	0x2408_0000~ max RAM shared	SRAM BOOT(RAM shared)	BOOT_ADDRESS
		0	Other	USER BOOT	0x0800_0000
		0		SYSTEM BOOT	0x1FEF_0000

Note: REPAIR_BOOT_SYS = !HWREPMOD & BLREPMOD & !BLREPDONE.

(1) The GD32H77xxI/GD32H78xxI series does not have Check_bootloader.

2.1.2. Instruction of cache use

Data consistency issues when using Dcache and DMA simultaneously

GD32H77x_78x 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.

Figure 2-1. Dcache invalidation

```
    /* 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.

Figure 2-2. 32-byte aligned destination array

```
__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.

Figure 2-3. Clear cache

```
/* 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.

Figure 2-4. 32-byte aligned welcome array

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

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.

2.1.3. Bootloader operation notes

For detailed information, refer to ***AN298 Precautions for GD32H77x_78x BootLoader Operation***.

2.2. Non-volatile memory controller (NVMC)

2.2.1. Non-volatile memory structure and configuration recommendations

Non-volatile memory structure

Different devices correspond to different memory structures, as detailed below:

GD32H77xxW/GD32H77xxP/GD32H77xxI/GD32H78xxW/GD32H78xxP/GD32H78xxI support different capacities of CNVM (code non-volatile memory) and ECNVM (extend code non-volatile memory): (MB)

- GD32H77xxW/GD32H78xxW: 2 + 7.5
- GD32H77xxP/GD32H78xxP: 2 + 3.5
- GD32H77xxI/GD32H78xxI: 1.625 + 0

2.2.2. Error check and correction (ECC)

ECNVM ECC

The ECNVM ECC mechanism supports:

- One error detection and correction
- Two errors detection

When two errors are detected:

The ECCDET bit in NVMC_ECNVM_STAT register will be set. If the ECCDETIE bit in NVMC_ECNVM_CTL register is set, an interrupt is generated. The ECCADDR bits in NVMC_ECNVM_ECCADDR register notice the address where the error occurred. The ECCDET bit in NVMC_ECNVM_STAT register can be cleared by writing 1 to ECCDET bit in NVMC_ECNVM_STAT register.

Note:

1. Data in ECNVM memory are 72-bits words: 8 bits ECC are added per double word (64 bits), but the added 8bits ECC are calculated by hardware and cannot be accessed by user.
2. Only the 7.5MB ECNVM has ECC functionality.
3. The ECNVM ECC mechanism supports one error detection and correction, but there is no corresponding one bit error flag.
4. When the original data programmed is all F, the ECC calculation will bypass this operation. Erasing is not required, and programming other values again can be written normally.
5. When an ECC error is reported, a new read at the error address may not generate an ECC error if the data is still present in the current buffer / prefetch buffer, even if ECCDET bit is cleared.

CNVM ECC

The CNVM ECC mechanism supports:

- One error detection and correction
- Two errors detection and correction

When two error is detected:

The AUTOECCDERRDET bit in NVMC_CNVM_STAT register will be set. If the AUTOECCDERRDETIE bit in NVMC_CNVM_CTL register is set, an interrupt is generated. The NVMC_CNVM_AUTOERR2ADDRx register records the address where the error occurred and state of address. The AUTOECCDERRDET bit in NVMC_CNVM_STAT register can be cleared by clearing all AUTOERR2ADDRSTAT of NVMC_CNVM_AUTOERR2ADDRx (x = 0…3).

Note:

1. Data in CNVM memory are 144-bits words: 16 bits are added per 128 bits, but the added 16bits are calculated by hardware and cannot be accessed by user.
2. Support four error detected / corrected entirety for 2*128-bit CNVM word using 2*16 ECC check bits (For 2*128-bit CNVM word, high 128 bits, ECC can detected / corrected double bit errors at most, low 128 bits, ECC can detected / corrected double bit errors at most).
3. The CNVM ECC function can be turned on or off by the CNVM_ECC_OVRD and CNVM_ECC_MODE[1:0]. When CNVM_ECC_OVRD is set to 0 and CNVM_ECC_MODE[1:0] is set to 1 / 2 / 3, CNVM ECC is turned off. Otherwise, CNVM ECC is turned on. This function can be used to test the CNVM ECC feature.
4. When the same address is read twice, the second read is from the buffer. When using

CNVM ECC to test related functions, the second read of the same address may not yield the expected value under the current ECC state. Therefore, it is recommended to clear the buffer by setting EMPTY_CNVM_BUF after each ECC switch.

5. When an ECC error is reported, a new read at the error address may not generate an ECC error if the data is still present in the CPU buffer / general buffer, even if AUTOECCDERRDET bit is cleared.

2.2.3. Erase characteristics of CNVM and ECNVM

After erase, CNVM is in the 0 state, while ECNVM is in the 1 state.

2.2.4. Option bytes

Option bytes description

NVMC provides two groups of option byte registers. The option byte register is divided into two categories: opt and opt + reg.

For opt option byte register:

■ "_EFT" registers group (read-only)

These registers store the effective value of the option bytes. After system reset or wakeup from standby mode, the registers values will be automatically loaded from non-volatile memory.

■ "_MDF" registers group (read / write)

These registers store the modification of the option bytes. When user wants to modify the option byte, the desired option byte value needs to be written to the register group.

The option bytes block is reloaded to "_EFT" registers after each system reset, and the option bytes take effect.

For opt + reg option byte register:

■ "_EFT" registers group (read-only)

After system reset or wakeup from standby mode, the registers values will be automatically loaded from non-volatile memory.

■ "_MDF" registers group (read / write)

These registers store the modification of the option bytes. These registers store the effective value. When user wants to modify the option byte, the desired option byte value needs to be written to the register group.

If the option byte function is triggered during the reset phase, the corresponding configuration is the value of the option byte that was previously programmed. After reset, if

the software enables the corresponding function of the option byte, the configuration will be the value in the "_MDF" registers. The software can set these "_MDF" registers value before enabling the corresponding function. The option bytes block is reloaded to "_EFT" and "_MDF" registers after each system reset.

Note: For opt + reg option byte register, the values in "_MDF" registers take effect without the need to program them into the NVM.

The option bytes description is shown in the ***Table 2-3. Option bytes***. The option bytes are configured according to the requirements of the application.

Table 2-3. Option bytes

Name	Factory value	Class	Register mapping
[31]: VFIMG	0x0	opt	NVMC_OBSTAT0_EFT / NVMC_OBSTAT0_MDF
[30]: NOPIPE_MODE	0x1	opt	
[29]: IOSPDOPEN	0x0	opt	
[28]: CNVM_ECC_OVRD	0x1	opt	
[27]: R0_SRAM_ECCEN	0x1	opt	
[26]: R1_SRAM_ECCEN	0x1	opt	
[25]: R2_SRAM_ECCEN	0x1	opt	
[24]: DTCM1ECCEN	0x1	opt	
[23]: DTCM0ECCEN	0x1	opt	
[22]: ITCMECCEN	0x1	opt	
[21]: SCR	0x0	opt	
[20]: SWAP	0x0	opt	
[19]: FW_UPDATE	0x0	opt	
[18]: FWDGSPD_STDBY	0x1	opt	
[17]: FWDGSPD_DPSLP	0x1	opt	
[16]: OTA_CHECK	0x0	opt	
[15:8]: SPC[7:0]	0xAA	opt	
[7]: nRST_STDBY	0x1	opt	
[6]: nRST_DPSLP	0x1	opt	
[5]: VFALL	0x0	opt	
[4]: nWDG_HW	0x1	opt	
[3:2]: BOR_TH[1:0]	0x0	opt	
[1]: NOPRINT	0x0	opt	
[0]: DISLFI	0x0	opt	
[31]: CNVM_DCRP_EREN	0x0	opt	NVMC_DCRPADDR_EFT / NVMC_DCRPADDR_MDF
[30]: ECNVM_BANK0_DCRP_EREN	0x0	opt	
[29]: ECNVM_BANK1_DCRP_EREN	0x0	opt	
[24:16]: DCRP_AREA_END[8:0]	0x0	opt	
[15]: CNVM_DCRP_EN	0x0	opt	
[14]: ECNVM_BANK0_DCRP_EN	0x0	opt	

Name	Factory value	Class	Register mapping
[13]: ECNVM_BANK1_DCRP_EN	0x0	opt	
[8:0]: DCRP_AREA_START[8:0]	0x1FF	opt	
[31]: CNVM_SCR_EREN	0x0	opt	NVMC_SCRADDR_EFT / NVMC_SCRADDR_MDF
[30]: ECNVM_BANK0_SCR_EREN	0x0	opt	
[29]: ECNVM_BANK1_SCR_EREN	0x0	opt	
[24:16]: SCR_AREA_END[8:0]	0x0	opt	
[15]: CNVM_SCR_EN	0x0	opt	
[14]: ECNVM_BANK0_SCR_EN	0x0	opt	
[13]: ECNVM_BANK1_SCR_EN	0x0	opt	
[8:0]: SCR_AREA_START[8:0]	0x1FF	opt	
[15:0]: WP[15:0]	0xFFFF	opt	NVMC_WP_CNVM_EFT / NVMC_WP_CNVM_MDF
[31:16]: BOOT_ADDR1[15:0]	0x1FEF	opt	NVMC_BTADDR_EFT / NVMC_BTADDR_MDF
[15:0]: BOOT_ADDR0[15:0]	0x080X	opt	
[31]: CPUBUSTO	0x1	opt	NVMC_OBSTAT1_EFT / NVMC_OBSTAT1_MDF
[30:16]: DATA[14:0]	0x7FFF	opt	
[6:4]: ITCM_DTCM_SHARED[2:0]	0x0	opt	
[2:0]: ITCM_AXI_SHARED[2:0]	0x0	opt	
[31:0]: WP[31:0]	0xFFFFFFFF	opt	NVMC_WP0_ECNVM_EFT / NVMC_WP0_ECNVM_MDF
[29:0]:WP[59:32]	0x0FFFFFFF	opt	NVMC_WP1_ECNVM_EFT / NVMC_WP1_ECNVM_MDF
[31:26]: UPDATESZ[5:0]	0x3F	opt	NVMC_OBREPCTL_EFT NVMC_OBREPCTL_MDF
[15]: PORRSTREPDIS	0x0	opt	
[14]: SYSRSTREPDIS	0x1	opt	
[13]: BBMODE	0x1	opt + reg	
[12:8]: DIESZ[4:0]	0x1F	opt	
[5]: BBREPLEN	0x1	opt + reg	
[4:3]: REPUNLK	0x3	opt	
[1]: HWREPMOD	0x1	opt	
[0]: DIEEN	0x1	opt	
[20:5]: STARTADDR[15:0]	0x0	opt + reg	NVMC_OBREPSTADDR_EFT NVMC_OBREPSTADDR_MDF
[20:5]: ENDADDR[15:0]	0xFFFF	opt + reg	NVMC_OBREPENDADDR_EFT NVMC_OBREPENDADDR_MDF
[20:5]: OBBBADDR[15:0]	0x0	opt	NVMC_OBBBADDRx_EFT / NVMC_OBBBADDRx_MDF
[0]: OBBBADDRVLD	0x0	opt + reg	

Option Bytes modification

The following steps show the erase sequence.

1. Unlock the NVMC_OBCTL register if necessary.
2. Check the BUSY bit in NVMC_ECNVM_STAT register to confirm that no ECNVM memory operation is in progress (BUSY equals to 0). Otherwise, wait until the operation has finished. (For GD32H77xxI/GD32H78xxI, check the BUSY bit in NVMC_CNVM_STAT register to confirm that no CNVM memory operation is in progress (BUSY equals to 0). Otherwise, wait until the operation has finished.)
3. Write the desired option byte value in NVMC_XXX_MDF.
4. Send the option bytes change command to the NVMC by setting the OBSTART bit in NVMC_OBCTL register.
5. Wait until all the operations have been finished by checking the value of the BUSY bit in NVMC_ECNVM_STAT register. (For GD32H77xxI/GD32H78xxI, wait until all the operations have been finished by checking the value of the BUSY bit in NVMC_CNVM_STAT register.)
6. Launch a system reset to start option bytes loading.
7. Read from NVMC_XXX_EFT register and verify the option bytes if required.

Note: "XXX" includes OBSTAT0, DCRPADDR, SCRADDR, WP_CNVM, BTADDR, OBSTAT1, WP0_ECNVM, WP1_ECNVM, OBREPCTL, OBREPSTADDR, OBREPENDADDR or OBBBxADDR.

When the operation is executed successfully, the ENDF in NVMC_ECNVM_STAT register is set, and an interrupt will be triggered by NVMC if the ENDIE bit in the NVMC_ECNVM_CTL register is set. (For GD32H77xxI/GD32H78xxI, when the operation is executed successfully, the ENDF in NVMC_CNVM_STAT register is set, and an interrupt will be triggered by NVMC if the ENDIE bit in the NVMC_CNVM_CTL register is set.)

If OBLK bit in the NVMC_OBCTL register is set, the _MDF register is forbidden to modify. When the OBSTART bit is set to 1, NVMC will compare the effective value (_EFT) with a new value (_MDF) to check whether have option byte needs to be changed.

2.2.5. Precautions for using OTP

Introduction to OTP Parameter Properties

The NVMC_OTP_USER_CTLx (x = 0-5) parameter has two sets of registers: hardware registers and software registers. Users can read and write hardware registers via the AXI bus and read software registers via the AHB bus.

The parameters in NVMC_OTP_USER_CTLx can be divided into three categories:

Security-related registers: SPC_H, SPC_L, DCRP_EFFCTL, SCR_EFFCTL, SCR, SCR_AREA_START, SCR_AREA_END, DCRP_AREA_START, DCRP_AREA_END, ECNVM_BANK1_SCR_EN, ECNVM_BANK0_SCR_EN, CNVM_SCR_EN,

ECNVM_BANK1_DCRP_EN, ECNVM_BANK0_DCRP_EN, CNVM_DCRP_EN, and all lock bits. For security-related registers, they are reset on power reset, and loaded after power reset; retained during system reset, and loaded after system reset.

Debug-related registers: NDBG and JTAGNSW. For debug-related registers, they are reset on power reset and loaded after power reset; retained during system reset and loaded after system reset. The NDBG configuration takes effect after power reset. Additionally, JTAGNSW configured from 0 to 1 takes effect after power reset, and JTAGNSW configured from 1 to 0 takes effect after system reset.

Non-security-related registers: Registers in NVMC_OTP_USER_CTLx excluding security-related registers and debug-related registers are referred to as non-security-related registers. For non-security-related registers, they are reset on power reset and loaded after power reset; reset on system reset and loaded after system reset.

For DBGPSWD_DATA and OTP_AES_KEY: There are no AHB software registers, and users can only write via the AXI bus. Loaded after system reset.

For the user data area, there are no AHB software registers, and users can only perform read/write operations via the AXI bus. For data bits and lock bits in the user data area, writes to the user data area take effect immediately without requiring a reset operation.

Introduction to the OTP Parameter Rollback Function

In the NVMC_OTP_USER_CTLx parameters, once VFALL, VFIMG, DISLFI, DCRP_EFFCTL, SCR_EFFCTL, SCR, SPC_H, SPC_L, NDBG, ECNVM_BANK1_SCR_EN, ECNVM_BANK0_SCR_EN, CNVM_SCR_EN, ECNVM_BANK1_DCRP_EN, ECNVM_BANK0_DCRP_EN, and CNVM_DCRP_EN are configured to 1, the corresponding lock bits will be automatically set to 1. When reading software registers, they will display as 1, but the stored values in OTP will not be modified. The effective values are those in the software registers. For the remaining configuration bits not mentioned above, if they are not locked, they can roll back to 0 after being written to 1. However, if they are locked, they cannot be modified.

For DBGPSWD_DATA and OTP_AES_KEY: If not locked, they can roll back to 0 after being written to 1; if locked, they cannot be modified.

For data bits in the user data area: If not locked, they can roll back to 0 after being written to 1; if locked, they cannot be modified.

2.3. Power Management Unit (PMU)

Power consumption design is one of the key concerns for the GD32H77x_78x series products. The GD32H77x_78x supports features such as low-power switching DC-DC converters, USB power regulators, and low-power modes. Certain aspects of these functionalities require attention and additional explanation during use.

2.3.1. The POR_ON pin

The POR_ON pin must be connected to VDD; otherwise, some registers may not reset properly after a POR reset.

2.3.2. Sleep mode downloader connection issue

After the chip enters sleep mode, it is currently impossible to download and debug using the downloader. Exiting sleep mode is required before using the downloader for downloading and debugging.

2.3.3. Link issue between Pxx pins and Pxx_C pins in standby mode

In standby mode, the chip can be woken up via the rising edge of the WKUP pin. It is worth noting that for GD32H77x_78x, the WKUP pin corresponds to an analog input, meaning there will be a Pxy_C and Pxy pin pair. When the chip enters standby mode, the Pxy_C and Pxy pin pair will be short-circuited.

2.3.4. Software configuration

To configure the PMU mode correctly, users need to select the appropriate PMU mode in the firmware library system_gd32h77x_h78x.c to match the external circuit. For example, if the user is using the GD32H77EI chip and the external circuit adopts the SMPS power supply mode, and SMPS output is required, the user needs to uncomment line 63 of the code `#define SEL_PMU_SMPS_MODE PMU_BYPASS` to configure the PMU correctly, as shown below.

Figure 2-5. PMU mode configuration

```
61   */
62   //#define SEL_PMU_SMPS_MODE    PMU_DIRECT_SMPS_SUPPLY
63   //#define SEL_PMU_SMPS_MODE    PMU_BYPASS
```

2.4. Reset and clock unit (RCU)

Some peripheral clocks can be configured which user can 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 of clocks are as below.

1. ADC clock can be selected 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 clock switch dynamically.

3. The clock of USART can be selected as CK_APBx (0, 1), CK_AHB, CK_LXTAL or CK_IRC64MDIV. The USART supports clock switch dynamically.
4. The clock of I2C can be selected as CK_APB1, CK_PLL2R, CK_IRC64MDIV or CK_LPIRC4M. The I2C supports clock switch dynamically.
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 clock switch dynamically.
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 clock switch dynamically.
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 clock switch dynamically.
8. The OSPI is clocked by the clock of CK_AHB or CK_PLL0Q which defined by OSPISEL bits in RCU_CFG4 register. The OSPI supports clock switch dynamically.
9. The clock of LPDTS can be selected as CK_APB4 or CK_LXTAL.
10. The clock of CAN can be selected as CK_HXTAL, CK_APB2, CK_APB2/2 or CK_IRC64MDIV. The CAN supports clock switch dynamically.
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 clock switch dynamically.
12. The clocks of SAI0, SAI1 and SAI3 can be selected as CK_PLL0Q, CK_PLL1P, CK_PLL2P, I2S_CKIN or CK_PER. The SAI0, SAI1 and SAI3 supports clock switch dynamically.
13. The clock of HPDF can be selected as CK_AHB or CK_APB2. The HPDF supports clock switch dynamically.
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 clock switch dynamically.
16. The clock of SDIO can be selected as CK_PLL0Q and CK_PLL1R. The SDIO supports clock switch dynamically.
17. The TLI is clocked by the clock of PLL2R divided by 2, 4, 8, 16 which defined by PLL2RDIV bits in RCU_CFG1 register.
18. The clock of RSPDIF can be selected as CK_PLL0Q, CK_PLL1R, CK_PLL2R or CK_IRC64MDIV. The RSPDIF supports clock switch dynamically.
19. The ESC is clocked by the clock of CK_AHB, CK_PLL0R, CK_PLL1P or CK_PLL2Q which defined by ESCSEL bits in RCU_CFG1 register and then divided by ESCDIV. The ESC supports clock switch dynamically.
20. The NVMC is clocked by the clock of CK_AHB, CK_AHB / 2, CK_PLL1R or CK_PLL2R which defined by NVMC_ECNVMSEL bits in RCU_CFG1 register. The NVMC supports clock switch dynamically.
21. The ENCODER (AFMT, TFMT, ENDAT) is clocked by the clock of CK_SYS, CK_PLL0Q, CK_PLL1R or CK_PLL2R which defined by ENCODERSEL bits in RCU_CFG6 register and then divided by AFMTDIV, TFMTDIV or ENDATDIV. The ENCODER (AFMT, TFMT,

ENDAT) supports clock switch dynamically.

22. The HDSL is clocked by the clock of CK_SYS, CK_PLL0R, CK_PLL1P or CK_PLL2Q which defined by HDSLSEL bits in RCU_CFG6 register and then divided by HDSLDIV. The HDSL supports clock switch dynamically.
23. The clock of RTC can be selected as LXTAL clock, IRC32K clock, or HXTAL clocks divided by 2-63.

2.5. Debug (DBG)

2.5.1. Secure JTAG usage

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 OTP can't be modified once it is written into OTP.

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

Modify JTAGNSW and NDBG[1:0] bit fields of user control register (NVMC_OTP_USER_CTL0) of OTP 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 (NVMC_OTP_USER_CTL0) of OTP module and DBGPSWD_DATA[63:0] of password register .by writing security key into DBGPSWD_DATA [31:0] and DBGPSWD_DATA [63:32] 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:

Table 2-4. SecureJTAG.JlinkScript

```
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 DBGPSWD_DATA [31:0] when users set the key. 0X55667788 is the value written into DBGPSWD_DATA [63:32] 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 secureJTAG.bat file in the same path as that of SecureJTAG1.JlinkScript and copy the contents below to the file:

Table 2-5. secureJTAG.bat

```
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.

2.5.2. Jlink debugging issue

When using KEIL IDE to debug GD32H77x_78x chips, if the EXMC driver's address memory interface is opened during debugging, KEIL IDE will freeze the next time it is reopened. A window settings reset is required.

2.6. Analog-to-digital converter (ADC)

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.

2.6.1. Two methods for monitoring junction temperature

GD32H77x_78x 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).

2.6.2. 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:

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

V_{25}: internal temperature sensor output voltage at 25°C, 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.

2.6.3. 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.

2.6.4. 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.

2.7. DSI host controller (DSI)

2.7.1. PMU configuration

Before enabling the DSI clock, the PMU needs to be configured first. The specific configuration is as follows.

Table 2-6. Configuration of MIPI DSI PMU

```
pmu_mipi_ldo_enable();
while(1) {
    if(RESET != pmu_flag_get(PMU_FLAG_RDYF)) {
```

```
            break;
        }
};
pmu_mipi_power_enable();
pmu_mipi_output_isolation_enable();
```

2.7.2. Clock configuration

When the LCD screen is a video screen, the TLI clock configuration needs to be consistent with dsi_lcd_cfg. Among them, PIXEL_CLK is the TLI pixel clock frequency, LP_CLK is the DSI clock, and HS_RATE is the MIPI DSI frequency.

2.8. Resolver-to-Digital Conversion Module (RDCM)

2.8.1. RDCM Sine Wave Exciter Configuration

RDCM has two excitation signal output modes: analog sine wave output mode and PWM output mode. The configuration methods and precautions for each mode are introduced below.

Using Analog Sine Wave Output Mode

When using the analog sine wave output mode, the trigger source of the exciter must first be selected as either an external trigger signal or an internal clock by configuring the NCOETM[1:0] bits in the RDCM_CTL register.

When NCOETM[1:0] = 2'b00, the exciter's trigger source is the internal clock. The frequency of the excitation signal output is (CK_RDCM / (NCODIV[13:0] + 1)) * FEXC[13:0] / 16384, the amplitude is 0~(AEXC[12:0] / 2) transmitted to the DAC output register, and the initial phase is 2 * pi * PEXC[13:0] / 16384.

When NCOETM[1:0] is not 0, the exciter's trigger source is an external signal, and the signal selection is configured through the TRIGSEL module. The frequency of the excitation signal output is (external trigger frequency / (NCODIV[13:0] + 1)) * FEXC[13:0] / 16384, the amplitude is 0~(AEXC[12:0] / 2) transmitted to the DAC output register, and the initial phase is 2 * pi * PEXC[13:0] / 16384.

Figure 2-6. RDCM analog sine wave output mode configuration

```
/* configure RDCM trigger */
trigger_init.cal_trigger_edge = RDCM_CALETM_RISING;
trigger_init.cal_trigger_sel = RDCM_CALSEL_EXTERNAL_TRIGGER;
trigger_init.nco_trigger_sel = RDCM_NCOETM_RISING;
rdcm_trigger_config(&trigger_init);

/* configure RDCM excitor: frequence is 2MHz*128/16384 = 15625Hz, initial phase is 0 */
exc_init.exc_amp = 4095;
exc_init.exc_phase = 0;
exc_init.exc_fre = 128;
exc_init.exc_div = 0;
exc_init.exc_en = RDCM_EXC_ENABLE;
rdcm_excitor_output_config(&exc_init);
```

Use PWM output mode

When using PWM output mode, the configuration of trigger source, frequency, amplitude, and phase is consistent with the analog sine wave output mode.

In addition, the RDCM_PWMCFG register needs to be configured. When BREVM = 1, it is recommended to set PWMGAIN[13:0] to 2 * AEXC[12:0] and PWMOFFSET[13:0] to 8192. When BREVM = 0, it is recommended to set PWMGAIN[13:0] to roundup(2 * AEXC[12:0] / (NCODIV[13:0] + 1)) and PWMOFFSET[13:0] to AEXC[12:0].

When configuring the PWM excitation frequency, it is necessary to consider the balance between the clock frequency (internal/external) and NCODIV[13:0]. NCODIV[13:0] represents the number of PWM outputs within one sine wave cycle. The larger the value of NCODIV[13:0], the smoother the sine wave curve output after filtering through the peripheral circuit. It is recommended to set NCODIV[13:0] to roundup(16384 / FEXC[13:0] - 1).

Figure 2-7. Example of RDCM PWM mode output configuration

```
/* configure RDCM trigger */
trigger_init.cal_trigger_edge = RDCM_CALETM_DISABLE;
trigger_init.cal_trigger_sel = RDCM_CALSEL_ADC0;
trigger_init.nco_trigger_sel = RDCM_NCOETM_DIVISION;
rdcm_trigger_config(&trigger_init);

/* configure RDCM excitor: the frequence is 75MHz/64*256/16384 = 18310Hz, the initial phase is 0 */
exc_init.exc_amp = 4095;
exc_init.exc_phase = 0;
exc_init.exc_fre = 256;
exc_init.exc_div = 63;
exc_init.exc_en = RDCM_EXC_ENABLE;
rdcm_excitor_output_config(&exc_init);

/* configure RDCM output PWM */
pwm_init.pwm_en = RDCM_PWM_ENABLE;
pwm_init.bit_reverse = RDCM_BREVM_NORMAL;
pwm_init.pwm_gain = 128;
pwm_init.pwm_offset = 4095;
rdcm_excitor_pwm_config(&pwm_init);
```

2.8.2. RDCM mixer configuration

The RDCM mixer requires separate configuration for the I/Q channels, mainly including high-pass filter, mixing signal configuration, and zero-crossing detection configuration.

High-pass filter

The high-pass filter is used to eliminate the DC component of the I/Q channels and can be

configured to select manual compensation or automatic compensation mode through I/QHFM[2:0] in the RDCM_HFCFG0/1 registers.

When I/QHFM[2:0] = 3'b000, offset compensation can remove the DC offset component by subtracting a predefined value from the raw result. In this mode, the high-pass filter is disabled. The I/QOFFSET[14:0] in the RDCM_HFCFG0/1 registers needs to be written by user software as the predefined value.

When I/QHFM[2:0] is not 0 and set to other values, offset compensation can automatically remove the DC component of the differential signal. This is accomplished through a high-pass filter. The I/QOFFSET[14:0] compensation value in the RDCM_HFCFG0/1 registers is automatically updated by the filter. This attenuates frequency signals below the selected cutoff frequency by at least -3dB.

Table 2-7. High-pass filter cutoff frequency

I/QHFM[2:0]	Cutoff frequency	Note
000	-	High-pass filter disable
001	$1 * 10^{-5} * f_d$	Lowest cutoff frequency, rate 1
010	$4 * 10^{-5} * f_d$	Rate 2
011	$16 * 10^{-5} * f_d$	Rate 3
100	$62 * 10^{-5} * f_d$	Rate 4
101	$25 * 10^{-4} * f_d$	Rate 5
110	$1 * 10^{-2} * f_d$	Rate 6
111	$3.5 * 10^{-2} * f_d$	Rate 7

Note: f_d is the data rate output from the ADC, meaning the sampling rate of the ADC.

Mixing signal

In practical applications, it should always be ensured that the returned I/Q carrier signal is frequency and phase synchronized with the mixing signal.

To ensure frequency synchronization, the I/QFRE[13:0] in RDCM_RFMCFG0/1 registers must remain consistent with FEXC[13:0].

To ensure phase synchronization, phase offset compensation is required. RDCM provides three adjustment modes, selectable via the I/QZCDMS[1:0] in the RDCM_CTL register.

Manual mode: When I/QZCDMS[1:0] = 2'b00, the user can configure the initial phase of the I/Q reference frequency mixer in I/QPHASE[13:0] of RDCM_RFMCFG0/1 registers to manually compensate for phase error. Full-range adjustment of phase error compensation is achievable.

Calibration mode: When I/QZCDMS[1:0] = 2'b01 or 2'b10, the user can select the direction (positive/negative) of the zero-crossing detection mode in calibration. Additionally, the reference excitation source for mixer phase adjustment can be selected via the I/QSIGNSEL bit in the RDCM_DISWSEL register.

When I/QSIGNSEL = 0, the symbol of the on-chip exciter is used as the reference for the I/Q channel RDCM frequency mixer. In this mode, the resolver shaft should be fixed at the positive half-cycle of the envelope position. The hardware will capture and calculate the average positive/negative zero-crossing mismatch counter. The zero-crossing detection mode is configured in I/QZCDMS[1:0] of the RDCM_CTL register. Phase error extraction supports full-range adjustment. The calibrated I/Q phase value will be updated in I/QPHASE[13:0] of the RDCM_RFMCFG0/1 registers, and the mixer phase will be automatically adjusted.

Note: If the resolver shaft angle is within the negative half-cycle of the envelope, the calibrated phase will have a pi phase error.

When I/QSIGNSEL = 1, the symbol of an external chip exciter is used as the reference for the I/Q channel RDCM frequency mixer. The hardware will capture and calculate the average positive/negative zero-crossing mismatch counter. Phase error extraction and adjustment support full-range. The calibrated I/Q phase value will be updated in I/QPHASE[13:0] of the RDCM_RFMCFG0/1 registers, and the mixer phase will be automatically adjusted.

Automatic mode: When I/QZCDMS[1:0] = 2'b11, an internally synthesized reference signal is generated in the I/Q channels synchronized with the reference frequency carrier. The synthesized reference signal is derived from internally filtered sine and cosine envelope signals. It is generated by evaluating the statistical average zero-crossing delay of the sine, cosine, and reference frequency mixers. Automatic phase error extraction is enabled by configuring different initial phases for the exciter and reference frequency mixer, with an adjustment range of ϕ ~ ϕ + pi (where ϕ is configurable). The calibrated I/Q phase will be updated in I/QPHASE[13:0] of the RDCM_RFMCFG0/1 registers, and the mixer phase will be automatically adjusted.

Zero-crossing detection

To detect the zero-crossing delay between the carrier portion of the modulated signal (after offset compensation) and the reference frequency mixer, a mismatch timer is used to generate the counter value during the NCO calculation cycle (exciter and reference frequency mixer). To eliminate the envelope impact on zero-crossing detection in the modulated signal, the mismatch counter value threshold should be configured in the I/QZCTH[13:0] bits of the RDCM_ZTCFG0 register.

In calibration mode, the recommended value is (16384 / FEXC[13:0]).

In automatic mode, the recommended value is (8192 / FEXC[13:0]).

Counter values within [I/QZCTH[13:0] - I/QZTVAR[13:0], I/QZCTH[13:0] + I/QZTVAR[13:0]] will be captured for evaluation.

The zero-crossing detection method is sensitive to signal quality and has limited precision. The precision of phase error adjustment is pi * FEXC[13:0] / 8192. The captured mismatch counter value may be unstable. Using hardware statistical averaging can provide a stable mismatch counter value.

When I/QAVEM[3:0] = 4'b0000 in the RDCM_MCACFG register, the statistical averaging function is disabled, and each captured mismatch counter value will update the phase adjustment of the reference frequency mixer. The larger the I/QAVEM[3:0] value, the greater the statistical averaging strength, but the convergence time becomes longer.

Figure 2-8. RDCM mixer configuration example

```
/* configure RDCM reference frequence mixer */
rfm_init.i_hf_mode = RDCM_IHFM_RATE5;
rfm_init.i_rfm_fre = 256;
rfm_init.i_rfm_phase = 12288;
rfm_init.i_zc_mode = RDCM_IZCDMS_BOTH;
rfm_init.i_zc_th = 32;
rfm_init.i_zc_th_var = 2;
rfm_init.i_average = 7;
rfm_init.i_rfm_en = RDCM_RFM_ENABLE;

rfm_init.q_hf_mode = RDCM_QHFM_RATE5;
rfm_init.q_rfm_fre = 256;
rfm_init.q_rfm_phase = 12288;
rfm_init.q_zc_mode = RDCM_QZCDMS_BOTH;
rfm_init.q_zc_th = 32;
rfm_init.q_zc_th_var = 2;
rfm_init.q_average = 7;
rfm_init.q_rfm_en = RDCM_RFM_ENABLE;
rdcm_reference_frequency_mixer_config(&rfm_init);
```

2.8.3. RDCM demodulator configuration

The RDCM demodulator configuration consists of three parts: CIC3 filter, gain adjustment, and compensation filter configuration.

CIC3 filter

Users configure the decimation rate (decimation rate) of the CIC3 filter to range from 4 to 512 through the I/QCICDFAC[8:0] settings in the RDCM_CICCFG0/1 registers.

Table 2-8. CIC3 filter properties (N = I/QCICDFAC[8:0] + 1)

Filter	Maximum delay	Max output value gain
CIC3	3N	$[-N^3, N^3]$

The valid data bits of the CIC3 filter output are partially affected by the CIC3 decimation factor I/QCICDFAC[8:0] and the modulator used. The data shifter is controlled by the I/QCICSHIFT[5:0] settings in the RDCM_CSGCFG0/1 registers.

The value of I/QCICSHIFT[5:0] is determined by the following formula:

$$I/QCICSHIFT[5:0] = roundup(11 - \log_2(A_s / (N^3 * A_t)))$$

N: This value equals I/QCICDFAC[8:0] + 1.

A_s: Amplitude value of the signal after high-pass filtering.

A_t: Target amplitude value after calibration.

Gain adjustment

The rounding error in the above formula can be corrected by calculating the gain correction factor I/QGAIN[12:0], which is stored in the RDCM_CSGCFG0/1 registers.

$$I/QGAIN[12:0] = truncate(((A_s / (N^3 * A_t)) * 2^{(I/QCICSHIFT[5:0] - 11)}) * 4096)$$

Example:

For N=15, the signal's amplitude value As after passing through the high-frequency filter is 7000. We expect the amplitude At of the signal after CIC3 gain adjustment to also be 7000.

The value of I/QCICSHIFT[5:0] is:

$I/QCICSHIFT[5:0] = roundup(11 - \log_2(7000 / (N^3 * 7000))) = 23.$

The gain correction factor is:

$I/QGAIN[12:0] = truncate(((7000 / (N^3 * 7000)) \times 2^{(I/QCICSHIFT[5:0]-11)}) * 4096) = 4971.$

Compensation filter

The compensation filter consists of two FIR modules. The decimation rate of FIR0 is 2, and the decimation rate of FIR1 can be set to 1 or 2. The decimation rate of FIR1 is controlled by the I/QFIR1DEC bit in the RDCM_DEMCFG register. The total decimation rate of the phase-sensitive demodulator is 2 * N or 4 * N.

The coefficients of the two FIR filters (signed 10-bit, must be symmetric) are fixed. These filters are symmetric and have linear phase. FIR0 is an 8th-order symmetric filter, while FIR1 is a 28th-order symmetric filter.

The default coefficients of FIR0/FIR1 are shown in the table below.

Table 2-9. Default coefficients for FIR0

a0	a1	a2	a3	a4	a5	a6	a7
-29	-43	143	441	441	143	-43	-29

Table 2-10. Default coefficients for FIR1

a0, a27	a1, a26	a2, a25	a3, a24	a4, a23	a5, a22	a6, a21	a7, a20	a8, a19	a9, a18	a10, a17	a11, a16	a12, a15	a13, a14	a0, a27
-5	-3	3	14	-2	-22	-12	30	39	-20	-86	-19	196	399	-5

Figure 2-9. Example of RDCM demodulator configuration

```
/*configure RDCM demodulator */
dem_init.i_cic3_gain = 4295;
dem_init.i_cic3_shift = 31;
dem_init.i_cic3_dec = 99;
dem_init.i_cic3_val = 99;
dem_init.i_fir1_dec = RDCM_FIR1_DEC_2;
dem_init.i_cic3_en = RDCM_CIC3_ENABLE;
dem_init.i_dem_en = RDCM_DEM_ENABLE;
dem_init.i_fir0_en = RDCM_FIR0_ENABLE;
dem_init.i_fir1_en = RDCM_FIR1_ENABLE;
dem_init.q_cic3_gain = 4295;
dem_init.q_cic3_shift = 31;
dem_init.q_cic3_dec = 99;
dem_init.q_cic3_val = 99;
dem_init.q_fir1_dec = RDCM_FIR1_DEC_2;
dem_init.q_cic3_en = RDCM_CIC3_ENABLE;
dem_init.q_dem_en = RDCM_DEM_ENABLE;
dem_init.q_fir0_en = RDCM_FIR0_ENABLE;
dem_init.q_fir1_en = RDCM_FIR1_ENABLE;
rdcm_demodulator_config(&dem_init);
```

2.8.4. RDCM group delay compensation configuration.

When the digital filter chain reaches a steady state, the delay is referred to as group delay, determined by the filter configuration. These delays are measured in RDCM calculation cycles, corresponding to ADC data update cycles.

Table 2-11. Group delay of filter elements

Filter element	Delay	Description
High-pass filter	0	Delay of high-pass filter
Gain stage	1	Pineline delay of gain stage calculation
Non-orthogonal phase mismatch compensation	0	Pineline delay of non-orthogonal phase mismatch compensation
CIC3	3 * (N - 1) / 2	Delay of CIC3 stage in PSD (N is the decimation ratio of CIC3)
FIR0	3.5 * N	Delay of FIR0 stage in PSD
FIR1	13.5 * N * 2^{F0}	Delay of FIR1 stage in PSD F0 = 0 for FIR0 is off, F0 = 1 for FIR0 is on
Zero-order holding	N * 2^{F0} * 2^{F1} / 2	Delay of zero-order holding after PSD F0 = 0 for FIR0 is off, F0 = 1 for FIR0 is on, F1 = 0 for FIR1 is off or FIR1 decimation is off, F1 = 1 for FIR1 on and FIR1 decimation is on

In practical use, the higher the speed, the greater the angle error. At this time, users need to perform real-time software compensation in the code. The angle error formula is as follows:

$$\Phi_{error} = \Phi_{installation} + \Phi_{demodulation}$$

$\Phi_{installation}$ is the installation offset of the resolver, determined by the actual structure. It can be considered a fixed value and needs to be measured by the user. $\Phi_{demodulation}$ is the angle caused by demodulation delay, which is related to the group delay in the table above. The calculation formula is as follows:

$$\Phi_{demodulation} = \text{Group delay} / \text{ADC sampling rate} * 16384 * n$$

Where n is the actual speed of the resolver, in units of revolutions per second, the calculation formula is as follows:

$$n = (\text{ADC sampling rate} * \text{VELOCITY[15:0]} / (16384 * 1024))$$

VELOCITY[15:0] is the speed value demodulated by RDCM, stored in the RDCM_VELDATA register.

The compensated angle is as follows:

$$\Phi_{compensation} = \text{POSITION[15:0]} + \Phi_{error}$$

POSITION[15:0] is the position value demodulated by RDCM, stored in the RDCM_POSDATA register. The radian angle (rad) conversion formula is as follows:

$$\theta = 2pi * \Phi_{compensation} / 16384$$

Note: If the calculated $\Phi_{compensation}$ is greater than 16384, it needs to be mapped to the range of 0-16384. If the calculated θ is greater than 2pi, it needs to be mapped to the range of 0-2pi.

2.9. Controller area network (CAN)

2.9.1. 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 configured 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:

Table 2-12. Configuration of the number of Rx FIFO identifier filtering table elements

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
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.

Figure 2-10. Configuration of the format of RxFIFO identifier filtering table elements

```
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:

Figure 2-11. Configuration of the private filters

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

2.9.2. CANFD data segment communication parameters and TDCO configuration

Compared with the GD32H759 and GD32H757 series, the GD32H77x_78x series exhibits differences in the configuration of CANFD data segment communication parameters. The GD32H77x_78x series has been designed with increased bit-field widths for DPBS2, DPBS1, DPTS, and DSJW in the data segment to meet the configuration requirements of data segment communication parameters when the CAN module reselects a high-speed clock source.

The CAN module in the GD32H77x_78x series has also been designed with an increased bit-field width for Transmission Delay Compensation Offset (TDCO) to ensure that the TDCO configuration meets the expected requirements.

The comparison of CANFD data segment communication parameters and TDCO bit-field widths is shown as *Table 2-13. Differences in CANFD data segment communication parameters and TDCO bit-field width*.

Table 2-13. Differences in CANFD data segment communication parameters and TDCO bit-field width

Function Name	Bit-field Width Comparison	
	GD32H77x_78x series	GD32H759/H757 series
Resynchronization jump width	DSJW[3:0]	DSJW[2:0]
Propagation time segment	DPTS[5:0]	DPTS[4:0]
Phase buffer segment 1	DPBS1[4:0]	DPBS1[2:0]
Phase buffer segment 2	DPBS2[4:0]	DPBS2[2:0]

Function Name	Bit-field Width Comparison	
	GD32H77x_78x series	GD32H759/H757 series
Transmitter delay compensation offset	TDCO[6:0]	TDCO[4:0]

2.10. Secure digital input/output interface (SDIO)

2.10.1. SDIO clock configuration

CK_SDIO is the kernel clock of the SDIO module. The SDIO_CLK provided to the SD/eMMC card is derived by dividing the frequency of CK_SDIO. When configuring CK_SDIO through the RCU, it is recommended to set the working clock frequency of CK_SDIO to no more than 208 MHz.

When configuring pins for Alternate Function (AF), it is recommended to modify GPIOx_AFSELx (x = 0 or 1) first, and then set the GPIO mode to AF.

Figure 2-12. SDIO pin initialization configuration

```
static void gpio_config(void)
{
    /* configure the SDIO_DAT0(PG9), SDIO_DAT1(PG10), SDIO_DAT2(PG11), SDIO_DAT3(PG12), SDIO_CLK(PD6) and SDIO_CMD(PD7) */
    gpio_af_set(GPIOG, GPIO_AF_11, GPIO_PIN_9 | GPIO_PIN_10);
    gpio_af_set(GPIOG, GPIO_AF_10, GPIO_PIN_11 | GPIO_PIN_12);
    gpio_af_set(GPIOD, GPIO_AF_11, GPIO_PIN_6 | GPIO_PIN_7);

    gpio_mode_set(GPIOG, GPIO_MODE_AF, GPIO_PUPD_PULLUP, GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12);
    gpio_output_options_set(GPIOG, GPIO_OTYPE_PP, GPIO_OSPEED_100_220MHZ, GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12);

    gpio_mode_set(GPIOD, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO_PIN_6);
    gpio_output_options_set(GPIOD, GPIO_OTYPE_PP, GPIO_OSPEED_100_220MHZ, GPIO_PIN_6);

    gpio_mode_set(GPIOD, GPIO_MODE_AF, GPIO_PUPD_PULLUP, GPIO_PIN_7);
    gpio_output_options_set(GPIOD, GPIO_OTYPE_PP, GPIO_OSPEED_100_220MHZ, GPIO_PIN_7);
}
```

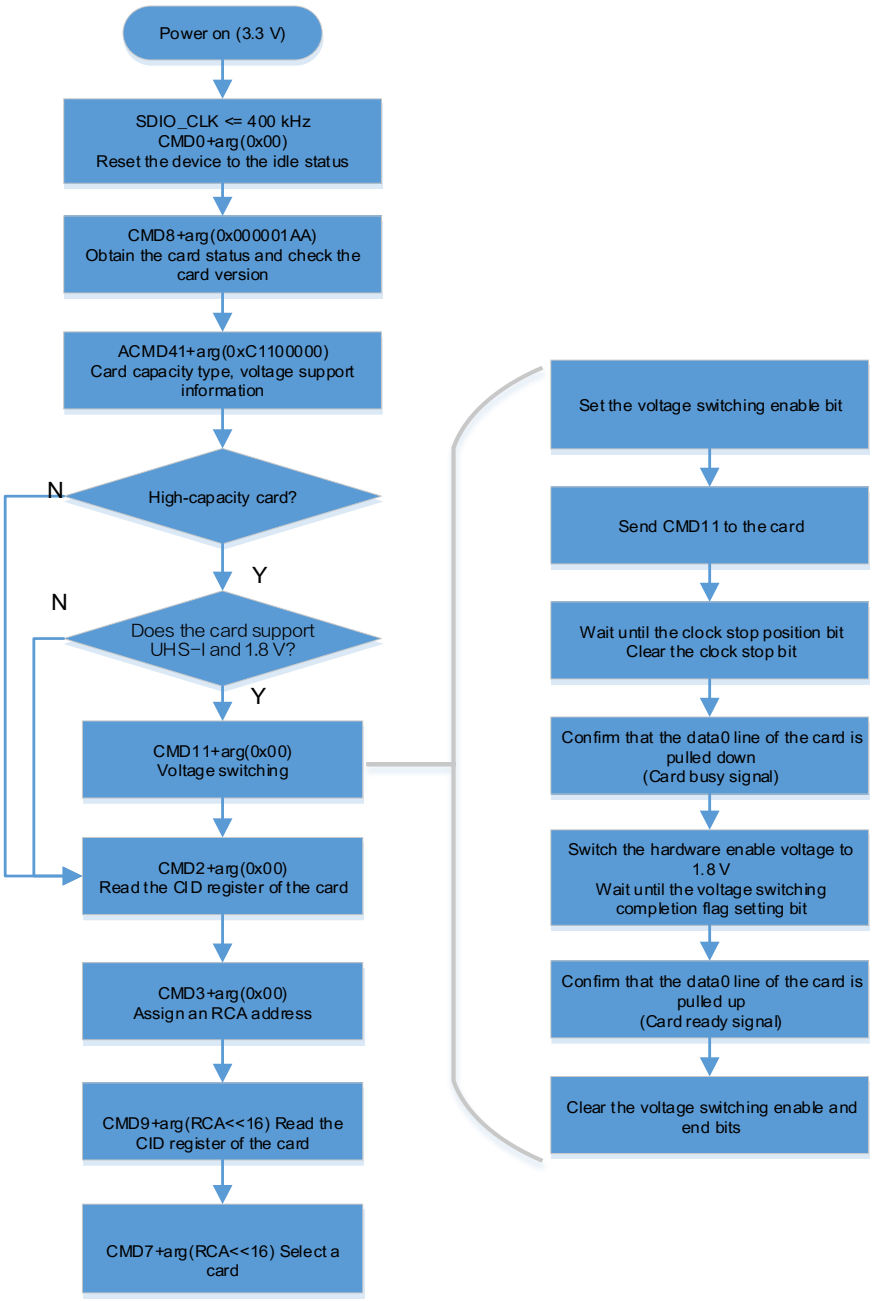
2.10.2. SDIO power-on initialization

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

SD 3.0 cards support the UHS-I (Ultra High Speed Bus Speed Mode Phase I) speed mode, which includes SDR12, SDR25, SDR50, SDR104, and DDR50. UHS-I operates at a voltage of 1.8 V, while the card powers on at 3.3 V. Therefore, UHS-I mode requires support for voltage switching from 3.3 V to 1.8 V. When the voltage switching sequence is successfully completed, the card will enter UHS-I mode with SDR12 as the default.

Power-on initialization of the card and the capture of card status and information can be performed for UHS-I cards. When responding to ACMD41, the card will indicate support for a 1.8 V voltage switch and then proceed with the 1.8 V voltage switching operation. The main processes are shown in the figure below. In addition, please note that SDIO0 supports UHS-I, but SDIO1 does not. In other words, SDIO1 cannot generate signals to control external voltage switching chips.

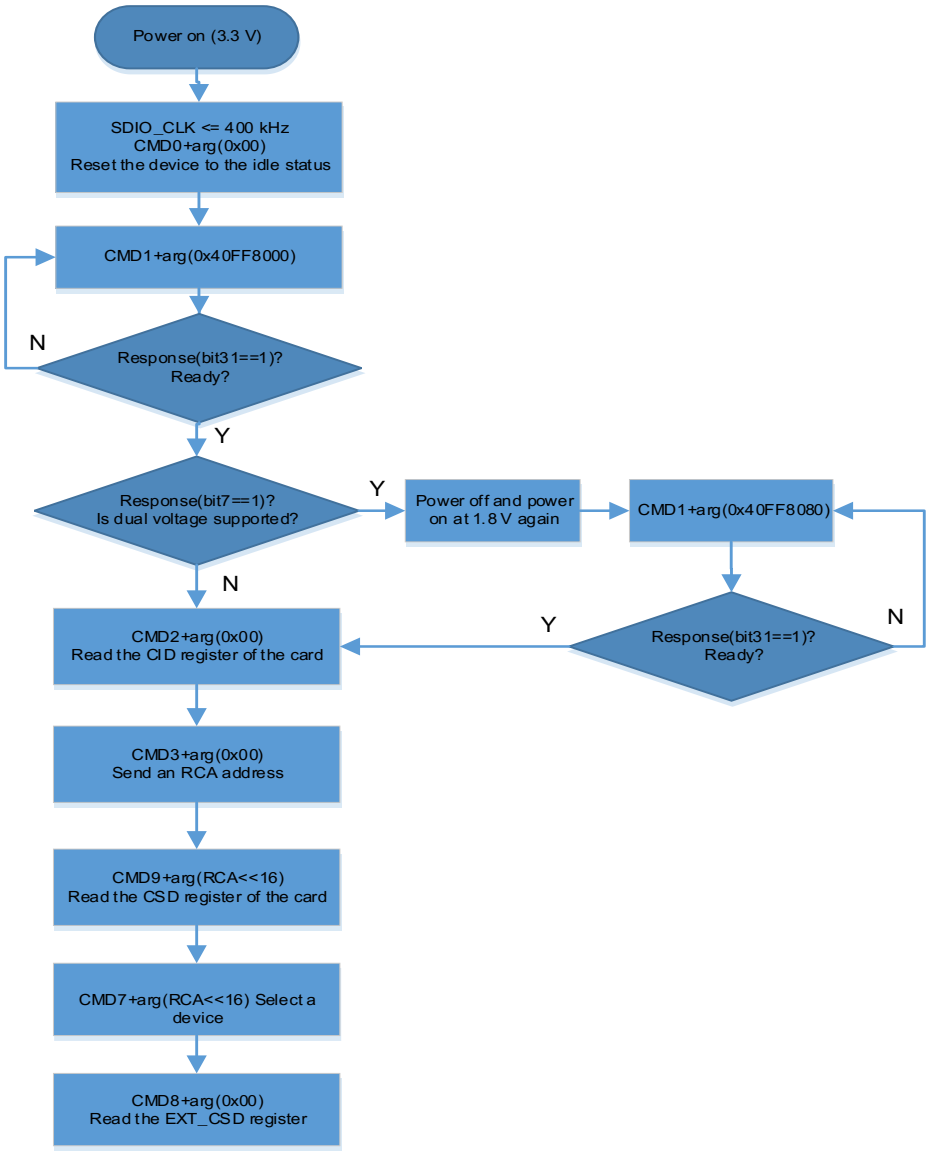
Figure 2-13. Processes of power-on initialization and voltage switch of SD card



Power-on initialization of eMMC

The power-on process of eMMC is shown in the figure below. First, the host determines whether the eMMC can operate at 1.8 V by checking the OCR register, and then reconfigures the power supply to 1.8 V to power on the eMMC.

Figure 2-14. Processes of power-on initialization and voltage switch of eMMC



2.10.3. Bus width configuration

When the host sends a commands 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 2-14. Configuring bus width of SD card

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

Table 2-15. 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.

Table 2-16. 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

Table 2-17. 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 or SDR104 that can work in UHS-I are in bus speed mode, CPDM module can be used for tuning of sampling points of receiving data. After using CPDM, hardware flow control must be disabled. Before sending CMD19 for tuning, the delay line length must be configured with CPDM.

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

Figure 2-15. CPDM configuration

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

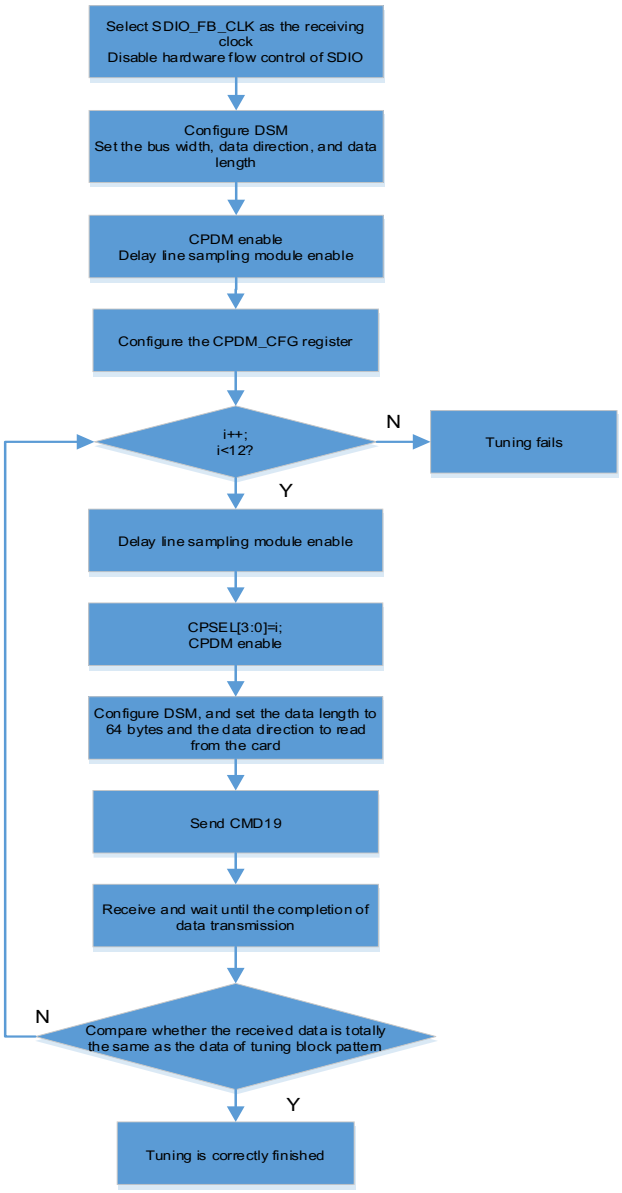
    for(i = 0; i < ((uint32_t)0x00000080U); i++) {
        CPDM_CFG(CPDM_SDIO0) = temp | (i << 8);

        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 2-16. Tuning processes of SD card



2.10.4. 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.

Table 2-18. Accessing partition commands and parameters

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

2.10.5. MDMA configuration

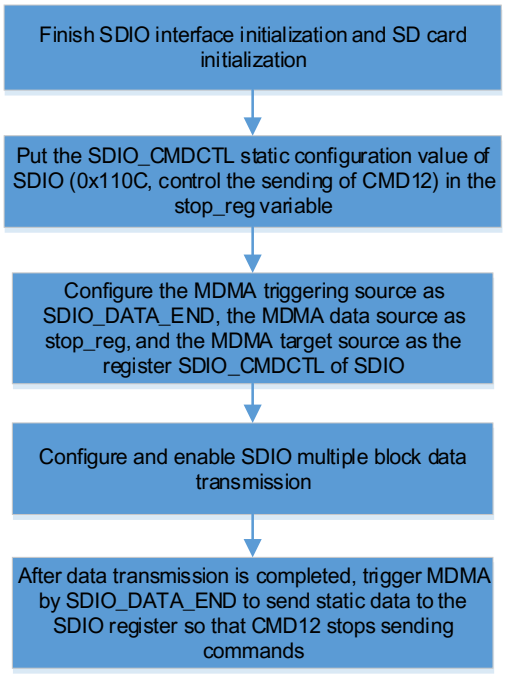
Controlling SDIO transfer process with MDMA

When MDMA is not used, CPU will control SDIO transfer process. For example, after transferring multiple block 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 take over SDIO transfer control from the CPU during three specific events: the end of data transfer (SDIO_DATA_END), the end of a command sequence (SDIO_CMD_END), and the end of buffer transfer (SDIO_BUF_END). When MDMA is triggered at these events, it will transfer pre-configured static data to the SDIO register to manage SDIO operations.

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

Figure 2-17. Flow chart of MDMA controlling the sending of CMD12

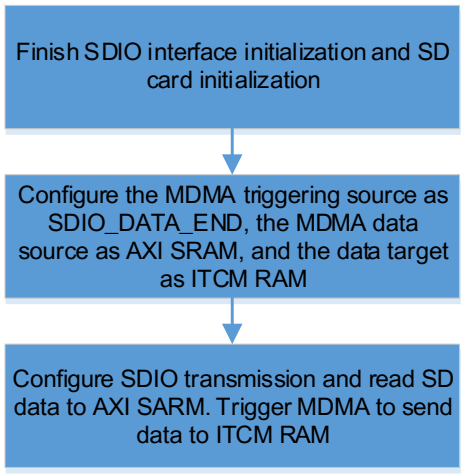


In addition, MDMA supports the linked list, so it is possible to conduct transfer control on SDIO for many times with multi-node linked list.

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 2-18. Flow chart of MDMA controlling RAM data transfer



2.10.6. 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 alternately read or write data in two buffers. When SDIO is using one buffer, the CPU can read or write data in the other buffer. It is necessary to configure the SDIO register SDIO_IDMACTL in dual buffer mode and enable IDMA, set SDIO_IDMASIZE as the size of each buffer, configure SDIO_IDMAADDR0 and SDIO_IDMAADDR1 as the addresses of the two external buffers, and optionally enable the IDMAEND interrupt. By enabling SDIO data transfer, SDIO will alternately access the two buffers. When the transfer in one buffer is completed, the IDMAEND interrupt will be triggered. The CPU can determine which buffer is not being used by SDIO based on BUFSEL in SDIO_IDMACTL and access it.

2.11. External memory controller (EXMC)

2.11.1. 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.

Table 2-19. 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.

Table 2-20. Configuration of EXMC SDRAM MPU

```
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) ;
```

2.12. Serial Audio Interface (SAI)

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

Each audio sub-module of GD32H77x_78x 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.

Table 2-21. 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.

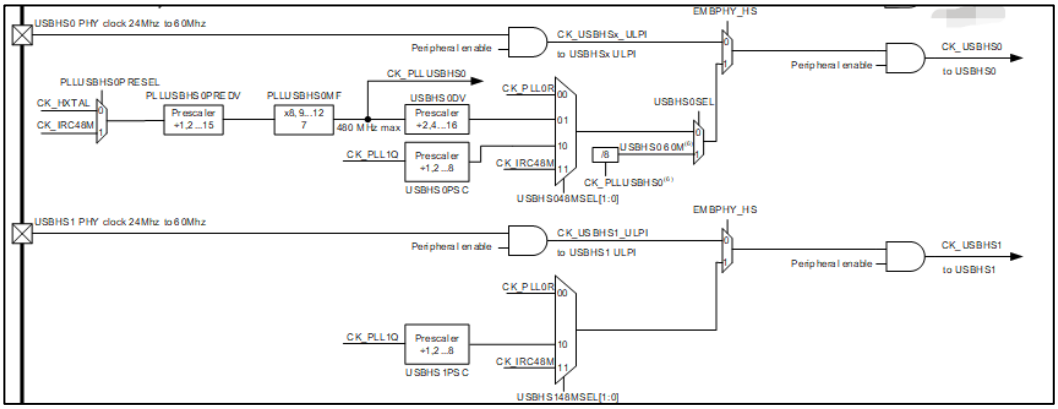
2.13. Universal serial bus High-Speed interface (USBHS)

2.13.1. Precautions for the Use of USBHS

The precautions for the USBHS module of the GD32H77x_78x series are as follows:

(1) Clock configuration of USBHS

Figure 2-19. USBHS clock tree



The clock configurations of USBHS0 and USBHS1 are different. The USBHS048MSEL selector can choose from four clock sources (CK_PLL0R, CK_PLLUSBHS0, CK_PLL1Q and CK_IRC48M), while USBHS148MSEL has only three clock sources (CK_PLL0R, CK_PLL1Q and CK_IRC48M).

USBHS0SEL selector is used to select the clock source USBHS060M or USBHS048MSEL. USBHS060M is obtained by dividing the 480M PHY clock. When selecting with software, it is required to set USBHS0SWEN bit of RCU_USBCLKCTL register. If USBHS0SWEN bit of

RCU_USBCLKCTL register is 0, the clock source will be selected by the hardware by default. Hardware will select USBHSx60M as the clock source when chirp handshake results in high speed.

USBHS0 uses the above configuration to enable the embedded PHY for USB high-speed applications and can also use EMBPHY_HS to select the ULPI clock for USB high-speed applications. However, USBHS1 does not have these clock options and can only use EMBPHY_HS to select the ULPI clock for USB high-speed applications.

(2) Configuration of internal and external PHY for USBHS

The embedded USB PHY of USBHS0 can be configured as either high-speed or full-speed. However, the embedded USB PHY of USBHS1 can only be configured as full-speed.

Both USBHS0 and USBHS1 support high-speed USB applications through an external ULPI PHY.

(3) Other features of USB HS

USBHS0 supports OTG mode, and the operation of full-speed devices also supports battery charging detection (BCD), attach detection protocol (ADP). However, USBHS1 does not support OTG mode, nor does it support BCD and ADP.

The other operations of the USBHS peripheral in the GD32H77x_78x series, such as USBHS GPIO configuration and USBHS power configuration, are the same as those in the USBHS of the GD32H7xx series. For details, please refer to ***AN117 Precautions for the use of USBHS of GD32H7xx series***.

2.14. Ethernet (ENET)

2.14.1. 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 and transmit / receive buffer.

Figure 2-20. ENET DMA descriptors and buffers definition

```
#elif defined (__clang__)                                        /*! ARM clang compiler */
volatile enet_descriptors_struct  rxdesc_tab0[ENET_RXBUF_NUM] __attribute__((section(".ARM.__at_0x30000000")));      /*!< ENET0 RxDMA descriptor */
volatile enet_descriptors_struct  txdesc_tab0[ENET_TXBUF_NUM] __attribute__((section(".ARM.__at_0x30000160")));      /*!< ENET0 TxDMA descriptor */
volatile uint8_t rx_buff0[ENET_RXBUF_NUM][ENET_RXBUF_SIZE]    __attribute__((section(".ARM.__at_0x30000300")));      /*!< ENET0 receive buffer */
volatile uint8_t tx_buff0[ENET_TXBUF_NUM][ENET_TXBUF_SIZE]    __attribute__((section(".ARM.__at_0x30002100")));      /*!< ENET0 transmit buffer */

volatile enet_descriptors_struct  rxdesc_tab1[ENET_RXBUF_NUM] __attribute__((section(".ARM.__at_0x30004000")));      /*!< ENET1 RxDMA descriptor */
volatile enet_descriptors_struct  txdesc_tab1[ENET_TXBUF_NUM] __attribute__((section(".ARM.__at_0x30004160")));      /*!< ENET1 TxDMA descriptor */
volatile uint8_t rx_buff1[ENET_RXBUF_NUM][ENET_RXBUF_SIZE]    __attribute__((section(".ARM.__at_0x30004300")));      /*!< ENET1 receive buffer */
volatile uint8_t tx_buff1[ENET_TXBUF_NUM][ENET_TXBUF_SIZE]    __attribute__((section(".ARM.__at_0x30006100")));      /*!< ENET1 transmit buffer */
```

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.

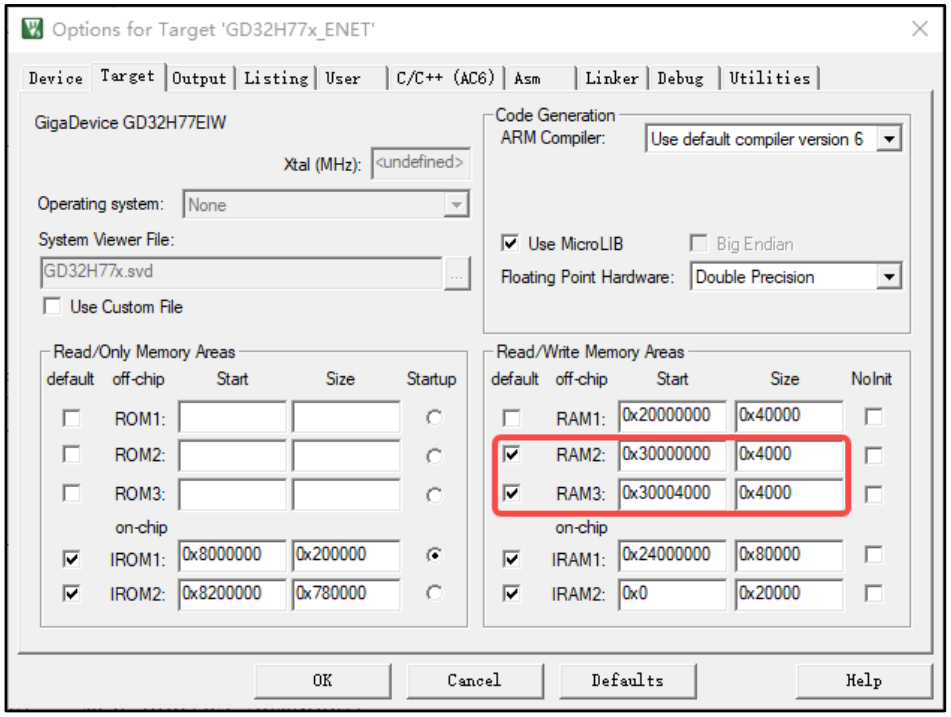
Figure 2-21. ENET DMA descriptors and buffers MPU configuration

```
/* Configure the ENET0 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_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_TYPE0;
mpu_region_config(&mpu_init_struct);
mpu_region_enable();

/* Configure the ENET1 DMA descriptors and Rx/Tx buffer */
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_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_NUMBER2;
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();
```

At the same time, it is required to select the set RAM area in related project settings. Otherwise, the above configuration will be invalid.

Figure 2-22. ENET project configuration



2.15. EtherCAT SubDevice controller (ESC)

For details, refer to ***AN246 Reference Manual for EtherCAT Protocol Stack Code Porting***.

3. Revision history

Table 3-1. Revision history

Revision No.	Description	Date
1.0	Initial release	Nov. 09, 2026

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.

© 2026 GigaDevice Semiconductor Inc. - All rights reserved