GigaDevice Semiconductor Inc.

GD32H77x_78x 软件开发指南

应用笔记

AN310

1.0 版本

(2026 年 11 月)

目录

目录 2

图索引 5

表索引 6

1. 简介 7

2. 模块使用说明 8

2.1. 系统及存储架构 8

2.1.1. Boot 方式选择和配 8
2.1.2. Cache 使用说明 9
2.1.3. Bootloader 操作注意事项 11

2.2. 非易失性存储器控制器(NVMC) 11

2.2.1. 非易失性存储器结构与配置建议 11
2.2.2. 错误检查与纠正(ECC) 11
2.2.3. CNVM 和 ECNVM 的擦除特性 12
2.2.4. 选项字节 12
2.2.5. OTP 使用注意事项 15

2.3. 电源管理单元(PMU) 17

2.3.1. POR_ON 引脚 17
2.3.2. 睡眠模式下载器链接问题 17
2.3.3. 待机模式下 Pxx 引脚和 Pxx_C 引脚链接问题 17
2.3.4. 软件配置 17

2.4. 复位和时钟单元(RCU) 17

2.5. 调试(DBG) 19

2.5.1. Secure jtag 使用 19
2.5.2. Jlink 调试问题 21

2.6. 模数转换器(ADC) 21

2.6.1. 结温监控的两种方法 21
2.6.2. ADC 监控结温方法 21
2.6.3. JTM 报警功能 22
2.6.4. 高温，高速下的建议 22

2.7. DSI 主机控制器(DSI) 22

2.7.1. PMU 配置 22
2.7.2. 时钟配置 22

2.8. 旋变数字转换模块(RDCM) 22

2.8.1. RDCM 正弦波激励器配置 22
2.8.2. RDCM 混频器配置 24
2.8.3. RDCM 解调器配置 26
2.8.4. RDCM 群延迟补偿配置 28

2.9. 控制器局域网络(CAN) 29

2.9.1. CAN 过滤器的使用 29
2.9.2. CANFD 数据段通信参数和 TDCO 配置 30

2.10. SDIO 接口(SDIO) 31

2.10.1. SDIO 时钟配置 31
2.10.2. SDIO 上电初始化 31
2.10.3. 总线宽度配置 33
2.10.4. 访问 eMMC Boot 分区数据 36
2.10.5. MDMA 配置 37
2.10.6. IDMA 配置 38

2.11. 外部存储器控制器(EXMC) 38

2.11.1. EXMC SDRAM 非对齐访问进 hardfault 问题 38

2.12. 串行音频接口(SAI) 40

2.12.1. SAI 使用 DMA 突发传输发送数据注意事项 40

2.13. 通用串行总线高速接口(USBHS) 40

2.13.1. USBHS 使用注意事项 40

2.14. 以太网(ENET) 41

2.14.1. ENET cache 开启的情况 41

2.15. EtherCAT 从站控制(ESC) 43

3. 版本历史 44

图索引

图 2-1. Dcache 使无效 9
图 2-2. 32 字节对齐的 destination 数组 9
图 2-3. Cache 清理操作 10
图 2-4. 32 字节对齐的 welcome 数组 10
图 2-8. PMU 模式配置 17
图 2-9. RDCM 输出模拟正弦波模式配置 23
图 2-10. RDCM 输出 PWM 模式配置示例 23
图 2-11. RDCM 混频器配置示例 26
图 2-12. RDCM 解调器配置示例 27
图 2-13. RxFIFO 标识符过滤表元素的格式配置 30
图 2-14. 私有过滤配置 30
图 2-15. SDIO 引脚初始化配置 31
图 2-16. SD 卡上电初始化和电压切换流程 32
图 2-17. eMMC 上电初始化和电压切换流程 33
图 2-18. CPDM 配置 35
图 2-19. SD 卡 tuning 流程 36
图 2-20. MDMA 控制 CMD12 流程图 37
图 2-21. MDMA 控制 RAM 数据传输流程图 38
图 2-22. USBHS 时钟树图 41
图 2-23. ENET DMA 描述符和缓存定义 42
图 2-24. ENET DMA 描述符和缓存 MPU 配置 42
图 2-25. ENET 工程配置 43

表索引

表 1-1. 适用产品 7
表 2-1. Boot 模式选择 8
表 2-2. Cache 操作 API 10
表 2-4. 选项字节 13
表 2-5. SecureJTAG.JlinkScript 19
表 2-6. secureJTAG.bat 21
表 2-7. MIPI DSI PMU 配置 22
表 2-8. 高通滤波器截止频率 24
表 2-9. CIC3 滤波器属性(N = I/QCICDFAC[8:0] + 1) 26
表 2-10. FIR0 默认系数 27
表 2-11. FIR1 默认系数 27
表 2-12. 滤波器元件的群延迟 28
表 2-13. Rx FIFO 标识符过滤表元素数目配置 30
表 2-14. CANFD 数据段通信参数和 TDCO 差异 31
表 2-15. SD 卡总线宽度配置 34
表 2-16. SD 卡总线速度配置 34
表 2-17. eMMC 总线模式配置 34
表 2-18. eMMC 总线速度配置 34
表 2-19. 访问分区的命令和参数 36
表 2-20. ARMv7-M 地址映射 39
表 2-21. EXMC SDRAM MPU 配置 39
表 2-22. DMA Burst 传输和 SAI FIFO 阈值 40
表 3-1. 版本历史 44

1. 简介

本应用笔记专为 GD32H77x_78x 系列 MCU 编写，旨在对该系列芯片的外设资源进行功能示例和使用注意事项说明，帮助用户快速掌握 GD32H77x_78x 系列 MCU 的软件开发方法与实践。本文档适用型号参考*表 1-1. 适用产品*。

表 1-1. 适用产品

产品系列	型号
GD32H77x	GD32H779、GD32H77D、GD32H77E、GD32H77R 系列
GD32H78x	GD32H789、GD32H78D、GD32H78E 系列

注意：本应用手册仅作参考，若与用户手册或数据手册内容有冲突，以用户手册或数据手册为准。

2. 模块使用说明

2.1. 系统及存储架构

2.1.1. Boot 方式选择和配

GD32H77x_78x 提供不同的 Boot 方式，主要是 SECURITY BOOT，USER BOOT，SRAM BOOT 和 SYSTEM BOOT。BOOT 引脚的电平状态会在复位后的第四个 CK_SYS(系统时钟)的上升沿进行锁存。用户可自行选择所需要的引导源，通过设置上电复位和系统复位后的 BOOT 的引脚电平。一旦这个引脚电平被采样，它们可以被释放并用于其他用途。BOOT_ADDR0[15:0]和 BOOT_ADDR1[15:0]地址允许将引导内存地址配置为 0x0000 0000 到 0x9000 0000 之间的任何地址。引导模式可由 SYSCFG_USERCFG 寄存器的 BOOT_MODE[2:0]位域中获取。具体请参考下表。

表 2-1. Boot 模式选择

SCR	SPC[7:0]	OTA(FW_UPDATE)	BOOT_ADDRESS (在 BOOT_ADDRx(x = 0,1) 配置)	BOOT MODE	启动地址
1	x	x	XXXX	SECURITY BOOT	ROM
0	安全保护等级高	x	0x0800_0000~max ECNVM	USER BOOT	BOOT_ADDRESS
			0x7000_0000	USER BOOT	OSPI1
			0x9000_0000	USER BOOT	OSPI0
			其他地址	USER BOOT	0x0800_0000
0	无保护状态/安全保护等级低	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 (在 BOOT_ADDRx(x = 0,1) 配置)	BOOT MODE	启动地址
			SRAM(ITCM/DTCM/AXI)	SRAM)	S
		0	0x2408_0000~ max RAM shared	SRAM BOOT(RAM shared)	BOOT_ADDRESS
		0	其他地址	USER BOOT	0x0800_0000
		0		SYSTEM BOOT	0x1FEF_0000

注意：REPAIR_BOOT_SYS = !HWREPMOD & BLREPMOD & !BLREPDONE。

(1) GD32H77xxI/GD32H78xxI系列没有Check_bootloader。

2.1.2. Cache 使用说明

Dcache 和 DMA 同时使用时的数据一致性问题

Dcache 和 DMA 同时使用时的数据一致性问题；GD32H77x_78x 提供支持读取分配的高速缓存(cache)，在 cache 未命中时将为数据分配高速缓存行，将 32 字节数据从主存储器存放至 cache，后续访问这些主存储器地址时，cache 会被命中，直接从 cache 中读取数据。当 SRAM 上启用了 cache 时，CPU 和 DMA 同时访问 SRAM 会存在一致性问题。

CPU 读取 DMA 写入到 SRAM 的数据：

当 DMA 从外设读取数据时并更新到 SRAM 中的接收缓冲区 destination[]时，此时 CPU 尝试读取 destination[]，它将从 cache 中存在的数据，而不是 SRAM 中的新数据。

解决办法如下：

1、 在 DMA 接收数据完成之后，对 cache 中的 destination[]进行无效操作。执行 cache 无效操作后使得 cache 中的 destination[]无效，CPU 在尝试读取 destination[]时，cache 不会被命中。

图 2-1. Dcache 使无效

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

2、 为保证 cache line 边界对齐，destination[]的大小必须是 32 字节对齐。

图 2-2. 32 字节对齐的 destination 数组

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

DMA 读取 CPU 写入到 SRAM 的数据：

当 CPU 在更新传输缓冲区 welcome[]中要待传输数据时，将仅更新 cache 中数据而不会更新

SRAM 中数据，当 DMA 读取 welcome[]中数据时，会从 SRAM 中读取，而不是 CPU 更新的 cache 中的新数据。

解决办法如下：

1、 在启用 DMA 传输之前，需执行 cache 清理操作，将 cache 中 welcome[]刷新到 SRAM 中。

图 2-3. Cache 清理操作

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

2、 welcome[]存放的地址必须是 32 字节对齐。

图 2-4. 32 字节对齐的 welcome 数组

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

Cache 的使用和数据对齐配置

以下表格中列出了由官网推荐的几种ARM CMSIS Dcache操作函数。

表 2-2. Cache 操作 API

操作 Cache 的 API	描述
void SCB_EnableDCache (void)	使整个数据 cache 无效后，启用数据 cache
void SCB_DisableDCache (void)	清理整个数据 cache 后，禁用数据 cache
void SCB_CleanDCache(void)	清理整个数据 cache
void SCB_CleanDCache_by_Addr (uint32_t *addr, int32_t dsize)	按地址清理数据 cache 行
void SCB_InvalidateDCache(void)	使整个数据 cache 无效
void SCB_InvalidateDCache_by_Addr (uint32_t * addr, int32_t dsize)	按地址使数据 cache 行无效
void SCB_CleanInvalidateDCache(void)	清理并使整个数据 cache 无效
void SCB_CleanInvalidateDCache_by_Addr(uint32_t *addr, int32_t dsize)	按地址清理并使数据 cache 行无效

注意：数据cache的行大小为32字节，cache的读写操作都是以行为单位，所以addr必须32字节边界对齐，dsize必须是32字节的整数倍。

Clean cache作用是把cache上标记脏(被改写)的行，强制写到SRAM上，并清除脏的标记，clean cache可以重建cache和SRAM的一致性。防止SRAM被DMA读取时，而最新的数据在cache上，没有及时被获取到。

Invalidate cache是作用是使cache行无效，只是将cache行的有效位清零，没有真正的清除cache行数据。可以防止SRAM数据被更新后，CPU获取的是cache上的数据，不是最新的数据。

另外，为了避免一致性问题的出现，也可以使用存储器保护单元(Memory Protection Unit，MPU)将CPU和DMA共享的SRAM区通过定义为不使用数据cache，同时将只能由 CPU 访问

的SRAM保留为使用数据cache的方式来操作，该方法可以参考ARM的MPU配置使用文档。

2.1.3. Bootloader 操作注意事项

具体内容参见*《AN298 GD32H77x_78x_BootLoader 操作注意事项》*。

2.2. 非易失性存储器控制器(NVMC)

2.2.1. 非易失性存储器结构与配置建议

非易失性存储器结构

不同的设备对应不同的存储器结构，具体情况如下：

GD32H77xxW/GD32H77xxP/GD32H77xxI/GD32H78xxW/GD32H78xxP/GD32H78xxI 支持不同容量的程序存储器(Code Non-Volatile Memory，CNVM)和扩展程序存储器(Extend Code Non-Volatile Memory，ECNVM)：(MB)

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

2.2.2. 错误检查与纠正(ECC)

ECNVM 纠错码

ECNVM ECC机制支持：

- 单个错误检测和纠正
- 双位错误检测

当检测到双位错误时：

NVMC_ECNVM_STAT寄存器中ECCDET位将置1。如果NVMC_ECNVM_CTL寄存器中的ECCDETIE置1，将产生中断。NVMC_ECNVM_ECCADDR寄存器中ECCADDR位域表示出现错误的地址。在NVMC_ECNVM_STAT寄存器中的ECCDET位写入1，ECCDET位将清零。

注意：

1. ECNVM中的数据是72位存储的，每个双字(64位)后加8位纠错码。8位纠错码由硬件自动计算，用户不可访问。
2. 只有7.5MB的ECNVM具有ECC功能。
3. ECNVM ECC机制支持单个错误检测和纠正，但是没有相应的单个错误标志位。
4. 当编程原始数据为全F时，ECC计算bypass本次操作，不需要擦除再次编程其他值可正常

写入。

5. 当上报ECC错误时，如果数据仍然存在于当前缓冲区/预取缓冲区中，即使清除ECCDET，重新读取该错误地址也不会产生ECC错误。

CNVM 纠错码

CNVM ECC机制支持：

- 单个错误检测和纠正
- 双位错误检测和纠正

当检测到双位错误时：

NVMC_CNVM_STAT寄存器中的AUTOECCDERRDET位将置1。如果NVMC_CNVM_CTL寄存器中的AUTOECCDERRDETIE置1，将产生中断。NVMC_CNVM_AUTOERR2ADDRx寄存器记录了错误发生的位置和地址状态。通过清除NVMC_CNVM_AUTOERR2ADDRx中的所有AUTOERR2ADDRSTAT位，将AUTOECCDERRDET位将清零。

注意：

1. CNVM中的数据是144位存储的：每128位后加16位纠错码。16位纠错码由硬件自动计算，用户不可访问。
2. CNVM支持对2*128位数据进行四位错误检测和纠正，使用2*16位ECC校验码(对于CNVM的2*128位，高128位，ECC最多可以检测和纠正双位错误；低128位，ECC最多可以检测和纠正双位错误)。
3. 当CNVM_ECC_OVRD设置为0且CNVM_ECC_MODE[1:0]设置为1 / 2 / 3时，CNVM ECC被关闭。否则，CNVM ECC被打开。此功能可以用来测试CNVM ECC功能。
4. 当同一个地址被读取两次，第二次是从缓冲区中读取。当使用CNVM ECC测试相关功能时，第二次读取同一地址可能不会得到当前ECC状态下预期值。因此，在每次ECC切换后，最好通过置位EMPTY_CNVM_BUF来清除缓冲区。
5. 当上报ECC错误时，如果数据仍然存在于当前的CPU缓冲区/通用缓冲区中，即使清除AUTOECCDERRDET，重新读取该错误地址也不会产生ECC错误。

2.2.3. CNVM 和 ECNVM 的擦除特性

CNVM擦除后状态为0，ECNVM擦除后状态为1。

2.2.4. 选项字节

选项字节说明

对于opt选项字节寄存器：

- "_EFT"寄存器组(只读)

该寄存器组包含选项字节的生效值。在从系统复位或待机模式唤醒后，该选项字节组的值都会被自动重加载。

■ "_MDF"寄存器组(可读写)

该寄存器组包含选项字节的修改值。当用户想要修改选项字节时，需要往该寄存器组里写入修改值。

每次系统复位后，ECNVM的选项字节被重加载到"_EFT"寄存器后，选项字节生效。

对于opt + reg选项字节寄存器：

■ "_EFT"寄存器组(只读)

在从系统复位或待机模式唤醒后，该选项字节组的值都会被自动重加载。

■ "_MDF"寄存器组(可读写)

该寄存器组包含选项字节的修改值。同时，该寄存器组包含选项字节的生效值。当用户想要修改选项字节时，需要往该寄存器组里写入修改值。

如果在复位阶段触发了选项字节功能，相应的配置将是之前已编程的选项字节的值。复位后，如果软件启用了选项字节的相应功能，配置将为"_MDF"寄存器中的值。软件可以在启用相应功能之前设置这些"_MDF"寄存器的值。每次系统复位后，选项字节块会被重新加载到"_EFT"和"_MDF"寄存器中。

注意：对于opt + reg选项字节寄存器，"_MDF"寄存器中的值在不编程到NVM的情况下即可生效。

选项字节详情见*表2-3. 选项字节*。选项字节最终根据应用程序的要求来配置。

表 2-3. 选项字节

名称	出厂值	分类	寄存器映射
[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	

名称	出厂值	分类	寄存器映射
[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	
[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	

名称	出厂值	分类	寄存器映射
[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	

选项字节修改

修改选项字节的操作步骤如下：

1. 确保NVMC_OBCTL寄存器不处于锁定状态；
2. 等待NVMC_ECNVM_STAT寄存器的BUSY位变为0来确保没有ECNVM操作在进行，否则等待该操作完成(对于GD32H77xxI/GD32H78xxI，等待NVMC_CNVM_STAT寄存器的BUSY位变为0来确保没有CNVM操作在进行，否则等待该操作完成)；
3. 在"_MDF"寄存器组(NVMC_XXX_MDF)中，写入想要修改的选项字节寄存器值；
4. 通过将NVMC_OBCTL寄存器的OBSTART位置1来发送选项字节编程命令到NVMC；
5. 通过检查NVMC_ECNVM_STAT寄存器的BUSY位是否清0，来确定编程指令执行完毕(对于GD32H77xxI/GD32H78xxI，通过检查NVMC_CNVM_STAT寄存器的BUSY位是否清0，来确定编程指令执行完毕)；
6. 启动一次系统复位以加载选项字节，使之生效；
7. 从NVMC_XXX_EFT寄存器读取数据，并在需要时验证选项字节。

注意："XXX"包括OBSTAT0、DCRPADDR、SCRADDR、WP_CNVM、BTADDR、OBSTAT1、WP0_ECNVM、WP1_ECNVM、OBREPCTL、OBREPSTADDR、OBREPENDADDR或OBBBxADDR。

当操作成功执行时，NVMC_ECNVM_STAT寄存器中的ENDF被置位，如果NVMC_ECNVM_CTL寄存器中的ENDIE位被设置，NVMC将触发一个中断。(对于GD32H77xxI/GD32H78xxI，当操作成功执行时，NVMC_CNVM_STAT寄存器中的ENDF被置位，如果NVMC_CNVM_CTL寄存器中的ENDIE位被设置，NVMC将触发一个中断)

当NVMC_OBCTL寄存器中的OBLK为1时，禁止修改"_MDF"寄存器。当OBSTART位被设置为1时，NVMC会将生效值(_EFT)与修改值(_MDF)进行比较，以检查是否需要修改选项字节。

2.2.5. OTP 使用注意事项

OTP 参数属性介绍

NVMC_OTP_USER_CTLx(x = 0-5)参数具有两组寄存器，即硬件寄存器和软件寄存器。用户可以基于AXI总线读写硬件寄存器，基于AHB总线读取软件寄存器。

NVMC_OTP_USER_CTLx内参数可以分为三类:

安全相关寄存器：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以及所有锁定位。对于安全相关寄存器，电源复位时复位，电源复位后load；系统复位时保持，系统复位后load。

Debug相关寄存器：NDBG和JTAGNSW。对于Debug相关寄存器，电源复位时复位，电源复位后load；系统复位时保持，系统复位后load。NDBG的配置在电源复位后生效。另外，JTAGNSW从0配置为1，在电源复位后生效；JTAGNSW从1配置为0，在系统复位后生效。

非安全相关寄存器：NVMC_OTP_USER_CTLx中除安全相关寄存器和Debug相关寄存器以外的寄存器被称为非安全相关寄存器。对于非安全相关寄存器，电源复位时复位，电源复位后load；系统复位时复位，系统复位后load。

对于DBGPSWD_DATA和OTP_AES_KEY：没有AHB软件寄存器，用户只能通过AXI总线进行写。系统复位后load。

对于用户数据区，没有AHB软件寄存器，用户只能通过AXI总线进行读写，对于用户数据区的数据位和lock位，写进用户数据区就会生效，不需要复位操作。

OTP 参数回退功能介绍

在NVMC_OTP_USER_CTLx参数中，一旦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和CNVM_DCRP_EN被配置为1，对应的锁定位也会自动置1。读取软件寄存器时将显示为1，但OTP中存储值不会被修改。实际生效的值是软件寄存器中的值。除了上面提到的配置位以外，剩余配置位在没有被锁定的前提下，在写1之后，可以回退为0；剩余配置位在被锁定的前提下，不可以被修改。

对于DBGPSWD_DATA和OTP_AES_KEY：在没有被锁定的前提下，在写1之后，可以回退为0；在被锁定的前提下，不可以被修改。

对于用户数据区中的数据位：在没有被锁定的前提下，在写1之后，可以回退为0；在被锁定的前提下，不可以被修改。

2.3. 电源管理单元(PMU)

功耗设计是 GD32H77x_78x 系列产品比较注重的问题之一，GD32H77x_78x 能够支持低功率开关电源降压稳压器、USB 电源调节器、省电模式等特性，特定的功能在使用时有些问题是比较值得关注和需要说明的。

2.3.1. POR_ON 引脚

POR_ON 引脚需要接 VDD，否则 POR 复位后，有些寄存器可能不在复位状态。

2.3.2. 睡眠模式下载器链接问题

在芯片进入睡眠模式后，目前是无法通过下载器下载和调试的。需要先退出睡眠模式再使用下载器下载和调试。

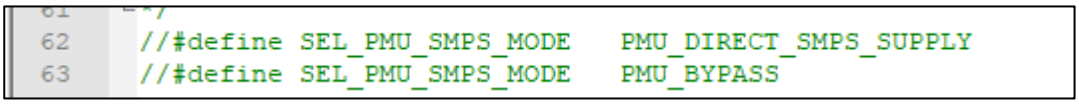
2.3.3. 待机模式下 Pxx 引脚和 Pxx_C 引脚链接问题

在待机模式下可以通过 WKUP 引脚的上升沿唤醒芯片，值得注意的是 GD32H77x_78x 的 WKUP 引脚会有对应的模拟输入端，即会存在 Pxy_C 和 Pxy 引脚对，当芯片进入待机模式时 Pxy_C 和 Pxy 引脚对将会短接。

2.3.4. 软件配置

为正确配置 PMU 模式，用户需要在固件库 system_gd32h77x_h78x.c 中选择正确的 PMU 模式用来匹配外部电路。例如，用户使用 GD32H77EI 芯片，客户需要使用 BYPASS 模式，为正确配置 PMU，用户需要取消注释 63 行代码 "#define SEL_PMU_SMPS_MODE PMU_BYPASS"。如下图所示。

图 2-5. PMU 模式配置



2.4. 复位和时钟单元(RCU)

有些外设的时钟是可配置的，用户可以根据需要选择指定的时钟进行配置。但在配置外设时钟之前，应保证相应的时钟已打开且稳定运行。对于支持时钟动态切换的外设，也应保证在切换目标时钟之前，目标时钟已经稳定运行。时钟可配置的外设如下：

1. ADC 时钟由 PLL1P、PLL2R、CK_PER 或由 AHB 时钟经 2、4、6、8、10、12、14、16 分频获得；

2. TRNG 的时钟可以选择由 CK_PLL0Q、CK_PLL2P 或 IRC48M 时钟提供，TRNG 支持时钟动态切换；
3. USART 时钟可以选择由 CK_APBx(0，1)、CK_AHB、CK_LXTAL 或 CK_IRC64MDIV 时钟提供，USART 支持时钟动态切换；
4. I2C 时钟可以选择由 CK_APB1、CK_PLL2R、CK_IRC64MDIV 或 CK_LPIRC4M 时钟提供，I2C 支持时钟动态切换；
5. SPI0(I2S0)、SPI1(I2S1)和 SPI2(I2S2)时钟可以选择由 CK_PLL0Q，CK_PLL1P，CK_PLL2P，I2S_CKIN 或 CK_PER 时钟提供，SPI0(I2S0)、SPI1(I2S1)和 SPI2(I2S2)支持时钟动态切换；
6. SPI3 和 SPI4 时钟可以选择由 CK_APB2，CK_PLL1Q，CK_PLL2Q，CK_IRC64MDIV，CK_LPIRC4M 或 CK_HXTAL 时钟提供，SPI3 和 SPI4 支持时钟动态切换；
7. SPI5(I2S5)时钟可以选择由 CK_APB2，CK_PLL1Q，CK_PLL2Q，CK_IRC64MDIV，CK_LPIRC4M，CK_HXTAL 或 I2S_CKIN 时钟提供，SPI5 支持时钟动态切换；
8. OSPI 时钟可以选择由 CK_AHB，CK_PLL0Q， CK_PLL1R 或 CK_IRC64MDIV 时钟提供，OSPI 支持时钟动态切换；
9. LPDTS 时钟可以选择由 CK_APB4 或 CK_LXTAL 时钟提供；
10. CAN 时钟可以选择由 CK_HXTAL、CK_APB2、CK_APB2 / 2 或 CK_IRC64MDIV 时钟提供，CAN 支持时钟动态切换；
11. SAI2 时钟可以选择由 CK_PLL0Q、CK_PLL1P、CK_PLL2P、I2S_CKIN、CK_PER 或 CK_RSPDIF_SYMB 时钟提供，SAI2 支持时钟动态切换；
12. SAI0，SAI1 和 SAI3 时钟可以选择由 CK_PLL0Q、CK_PLL1P、CK_PLL2P、I2S_CKIN 或 CK_PER 时钟提供，SAI0，SAI1 和 SA23 支持时钟动态切换。
13. HPDF 时钟可以选择由 CK_AHB 或 CK_APB2 时钟提供，HPDF 支持时钟动态切换；
14. HPDF_AUDIO 时钟可以选择由 CK_PLL0Q，CK_PLL1P，CK_PLL2P，I2S_CKIN 或 CK_PER 时钟提供；
15. EXMC 时钟可以选择由 CK_AHB，CK_PLL0Q，CK_PLL1R 或 CK_PER 时钟提供，EXMC 支持时钟动态切换；
16. SDIO 时钟可以选择由 CK_PLL0Q 或 CK_PLL1R 时钟提供，SDIO 支持时钟动态切换；
17. TLI 时钟可以选择由 PLL2R 时钟的 2、4、8、16 分频提供；
18. ESC 时钟可以选择由 CK_AHB，CK_PLL0R,CK_PLL1P 或 CK_PLL2Q 时钟经过 ESCDIV 分频后提供；ESC 支持时钟动态切换；
19. RSPDIF 时钟可以选择由 CK_PLL0Q,CK_PLL1R,CK_PLL2R 或 CK_IRC64MDIV 时钟提供。RSPDIF 支持时钟动态切换；
20. NVMC 时钟可以选择由 CK_AHB，CK_AHB/2，CK_PLL1R 或 CK_PLL2R 时钟提供；NVMC 支持时钟动态切换；
21. 编码器 ENCODER(AFMT、TFMT、ENDAT)时钟可以选择由 CK_SYS，CK_PLL0Q，CK_PLL1R 或 CK_PLL2R 时钟分别经过 AFMTDIV、TFMTDIV、ENDATDIV 分频后提供；编码器 ENCODER(AFMT、TFMT、ENDAT)支持时钟动态切换；
22. HDSL 时钟可以选择由 CK_SYS，CK_PLL0R，CK_PLL1P 或 CK_PLL2Q 时钟经过 HDSLDIV 分频后提供；HDSL 支持时钟动态切换；
23. RTC 时钟可以选择由 LXTAL 时钟、IRC32K 时钟或 HXTAL 时钟的 2-63 分频提供。

2.5. 调试(DBG)

2.5.1. Secure jtag 使用

安全 JTAG 功能只适用于 JTAG 接口，SWD 接口不支持安全功能。另外调试接口由 SWD 接口修改为 JTAG 接口后，不能再由 JTAG 接口恢复为 SWD 调试接口，因为通过修改 OTP 来实现的，写入 OTP 不能再修改。

调试接口 SWD 切换为普通 JTAG 接口的方法：

修改 OTP 模块的用户控制寄存器(NVMC_OTP_USER_CTL0)的 JTAGNSW 和 NDBG 位域，JTAGNSW 修改为 1，NDBG[1:0]修改为 00。电源复位后只能找到 JTAG 接口。

调试接口 SWD 切换为安全 JTAG 接口的方法：

配置为安全 JTAG 接口调试方法时，需要配置用户密钥，使用时需要密钥解锁。修改 OTP 模块的用户控制寄存器(NVMC_OTP_USER_CTL0)的 JTAGNSW 和 NDBG 位域，以及调试密码寄存器 DBGPSWD_DATA[63:0]，先向 DBGPSWD_DATA [31:0]和 DBGPSWD_DATA [63:32]写入安全密钥，并且 JTAGNSW 修改为 1，NDBG[1:0]修改为 01。电源复位后此时无法找到 JTAG 接口，需要上位机使用脚本给 MCU 发送密钥。

脚本使用方法：

首先新建 SecureJTAG.JlinkScript 文件，复制下面内容到该文件中：

表 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("I secure jtag key1.....");
    JLINK_JTAG_WriteDREnd(0x11223344, 32);

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

```
    JLINK_SYS_Report("I 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;
}
```

文件中的 0x11223344 为用户设置密钥时写入 DBGPSWD_DATA [31:0]中的值，0x55667788

为用户设置密钥时写入 DBGPSWD_DATA [63:32]中的值。文件中的 0x15 和 0x16 指令代表写密钥指令，0x18 和 0x19 指令代表读取写入的密钥，0x1e 指令是读取 mcu device ID，0x1a 指令是读取 secure jtag 和 wrong_seq 标志的状态。

然后在 SecureJTAG1.JlinkScript 同路径下新建一个 secureJTAG.bat 文件，复制下面内容到该文件中：

表 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
```

其中 D:\Keil_v528\ARM\Segger 为 jlink.exe 的目录。

最后执行 secureJTAG.bat 即可解锁使用 JTAG 调试。

注意:

1. 如果密码输入错误，则需电源复位。
2. 发生任何错误的输入序列后，若想重新解密都需要电源复位。
3. 输入正确密码打开 debug，只限于 SPC_L 及以下，不会打开 ROM、内存安全访问模式和 SPC_H。

2.5.2. Jlink 调试问题

使用KEIL IDE调试GD32H77x_78x芯片时，如果调试过程中开启了exmc驱动的地址memory界面，下次重新打开时，KEIL IDE 会卡死，需要复位窗口设置。

2.6. 模数转换器(ADC)

在芯片运行过程中 PN 结温度(结温)过高或过低将会导致性能下降，更加严重的情况将会导致芯片毁坏。

2.6.1. 结温监控的两种方法

GD32H77x_78x 可以通过读 ADC 高精度温度传感器和 PMU 中的温度监控(JTM 报警功能)两种方法用于监控结温。

2.6.2. ADC 监控结温方法

可以通过 ADC 的高精度温度传感器监测结温的实时温度，具体流程如下:

1. 配置温度传感器通道(ADC2_CH20)的转换序列和采样时间为 t_{s_temp} us，t_{s_temp} 典型值请参考数据手册;
2. 置位 ADC_CTL1 寄存器中的 TSVEN2 位，使能温度传感器;
3. 置位 ADC_CTL1 寄存器的 ADCON 位，或者由外部触发 ADC 转换;
4. 从 ADC 数据寄存器中读取并计算温度传感器数据 $V_{temperature}$，并由下面公式计算出实际温度:

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

V_{25}：内部温度传感器在 25° C 下的电压，典型值请参考数据手册。

Avg_Slope：温度与内部温度传感器电压曲线的均值斜率，典型值请参考数据手册。

2.6.3. JTM 报警功能

当 PMU_CTL1 寄存器中的 VBTMEN 位置位，温度阈值监测将开启，通过与温度高、低两个阈值水平比较可以来监测结温，PMU_CTL1 寄存器中 TEMPH 和 TEMPL 标志能够指示设备温度是否高于或低于阈值。并且 TEMPH 和 TEMPL 唤醒中断可用于 RTC 触发信号。

2.6.4. 高温，高速下的建议

当芯片在高速，高温下运行时应该实时监控结温和 JTM 中断，如果超过阈值需要降低时钟频率。

2.7. DSI 主机控制器(DSI)

2.7.1. PMU 配置

使能 DSI 时钟前，需要先配置 PMU，具体配置如下。

表 2-6. MIPI DSI PMU 配置

```
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. 时钟配置

LCD 屏为 video 屏时，TLI 时钟配置需要和 dsi_lcd_cfg 保持一致。其中 PIXEL_CLK 为 TLI 像素时钟频率，LP_CLK 为 DSI 时钟，HS_RATE 为 MIPI DSI 频率。

2.8. 旋变数字转换模块(RDCM)

2.8.1. RDCM 正弦波激励器配置

RDCM 具有两种激励信号输出模式，分别是模拟正弦波输出模式，PWM 输出模式。下面分别介绍两种模式配置方式和注意事项。

使用模拟正弦波输出模式

当使用模拟正弦波输出模式时，需要先选择激励器的触发源是外部触发信号还是内部时钟，即配置 RDCM_CTL 寄存器的 NCOETM[1:0]。

当 NCOETM[1:0] = 2'b00 时，激励器的触发源是内部时钟，输出激励信号的频率是(CK_RDCM / (NCODIV[13:0] + 1)) * FEXC[13:0] / 16384，幅值是 0~AEXC[12:0] / 2 传输到 DAC 的输出寄存器中，初始相位是 2 * pi * PEXC[13:0] / 16384。

当 NCOETM[1:0]不为 0，激励器的触发源是外部信号触发，信号选择通过 TRIGSEL 模块进行配置，输出激励信号的频率是(外部触发频率 / (NCODIV[13:0] + 1)) * FEXC[13:0] / 16384，幅值是 0~AEXC[12:0] / 2 传输到 DAC 的输出寄存器中，初始相位是 2 * pi * PEXC[13:0] / 16384。

图 2-6. RDCM 输出模拟正弦波模式配置

```
/* 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);
```

使用 PWM 输出模式

当使用 PWM 输出模式时，触发源、频率、幅值、相位的配置和模拟正弦波输出模式一致。

除此之外，需要额外配置 RDCM_PWMCFG 寄存器，在 BREVM = 1 时，建议将 PWMGAIN[13:0] 设置为 2 * AEXC[12:0]，PWMOFFSET[13:0]设置为 8192。在 BREVM = 0 时，建议将 PWMGAIN[13:0]设置为 roundup(2 * AEXC[12:0] / (NCODIV[13:0] + 1))，PWMOFFSET[13:0] 设置为 AEXC[12:0]。

在配置 PWM 激励频率时需要注意平衡时钟频率(内部/外部)与 NCODIV[13:0]的关系。NCODIV[13:0]代表着一个正弦波周期内 PWM 输出数，NCODIV[13:0]值越大，输出的 PWM 经过外围电路滤波后的正弦波曲线越平滑。建议将 NCODIV[13:0]设置为 roundup(16384 / FEXC[13:0] - 1)。

图 2-7. RDCM 输出 PWM 模式配置示例

```
/* 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 混频器配置

RDCM 的混频器需要为 I/Q 两路单独配置，主要包括高通滤波器、混频信号配置，过零点检测配置。

高通滤波器

高通滤波器用于消除 I/Q 通道的直流分量，可通过配置 RDCM_HFCFG0/1 寄存器的 I/QHFM[2:0]来选择手动补偿或者自动补偿模式。

当 I/QHFM[2:0] = 3'b000 时，时，偏移补偿可以通过从原始结果中减去一个预定义的值来移除直流信号的偏移分量。在此模式下，高通滤波器被禁用。RDCM_HFCFG0/1 寄存器的 I/QOFFSET[14:0]需要用户软件写入作为预定义值。

当 I/QHFM[2:0]不为 0 时，设置为其他值时，偏移补偿可以自动移除差分信号的直流分量。这是通过高通滤波器完成的。RDCM_HFCFG0/1 寄存器中的 I/QOFFSET[14:0]补偿值由滤波器自动更新。这至少可以将低于选定截止频率的频率信号衰减-3dB。

表 2-7. 高通滤波器截止频率

I/QHFM[2:0]	截止频率	注意
000	-	高通滤波器禁能
001	$1 * 10^{-5} * f_d$	最低的截止频率，速率1
010	$4 * 10^{-5} * f_d$	速率2
011	$16 * 10^{-5} * f_d$	速率3
100	$62 * 10^{-5} * f_d$	速率4
101	$25 * 10^{-4} * f_d$	速率5
110	$1 * 10^{-2} * f_d$	速率6
111	$3.5 * 10^{-2} * f_d$	速率7

注意：f_d 是从 ADC 输出的数据速率，即 ADC 的采样率。

混频信号

实际应用中，应始终保证返回的 I/Q 路载频信号与混频信号的频率、相位同步。

为了保证频率同步，RDCM_RFMCFG0/1 寄存器的 I/QFRE[13:0]需保持和 FEXC[13:0]一致。

为了保证相位同步，需要补偿相位偏移，RDCM 提供了三种调整模式，通过 RDCM_CTL 寄存器的 I/QZCDMS[1:0]进行选择。

手动模式：当 I/QZCDMS[1:0] = 2'b00 时，用户可以在 RDCM_RFMCFG0/1 寄存器的 I/QPHASE[13:0]中配置 I/Q 参考频率混频器的初始相位，以手动补偿相位误差。相位误差补偿可以实现全范围调整。

校准模式：当 I/QZCDMS[1:0] = 2'b01, 2'b10 时，用户可选择校准模式下过零点检测模式的方向(正向/负向)。除此之外，该模式中可通过在 RDCM_DISWSEL 寄存器的 I/QSIGNSEL 位选择混频器相位调整的参考激励源。

当 I/QSIGNSEL = 0 时，使用片上激励器的符号作为 I/Q 通道 RDCM 频率混频器的参考。在此模式下，应将旋转变压器轴固定在包络位置的正半周期。硬件将捕获并统计平均正/负过零点失配计数器。过零点检测模式设置在 RDCM_CTL 寄存器的 I/QZCDMS[1:0]中。相位误差提取可以实现全范围。I/Q 校准后的相位值将在 RDCM_RFMCFG0/1 寄存器的 I/QPHASE[13:0]中更新，并且混频器相位将自动调整。

注意：如果旋转变压器轴角度在包络的负半周期内，校准后的相位将存在 pi 相位误差。

当 I/QSIGNSEL = 1 时，使用外部芯片激励器的符号作为 I/Q 通道 RDCM 频率混合器的参考。硬件将捕获并统计平均正/负过零点失配计数器。相位误差提取和调整可以实现全范围。I/Q 校准的相位值将更新 RDCM_RFMCFG0/1 寄存器的 I/QPHASE[13:0]，并且混合器相位将自动调整。

自动模式：当 I/QZCDMS[1:0] = 2'b11 时，在 I/Q 通道中与参考频率载波同步生成内部合成参考信号。合成参考信号是利用内部滤波后的正弦和余弦包络信号派生出来的。它通过分别评估正弦、余弦和参考频率混频器的过零点延迟的统计平均值来生成。通过配置激励器和参考频率混频器的不同初始相位，可以实现自动相位误差提取，调整范围为Φ~ Φ + pi(Φ是可配置的)。I/Q 校准后的相位将在 RDCM_RFMCFG0/1 寄存器的 I/QPHASE[13:0]中更新，并且混频器相位将自动调整。

过零点检测

为了检测调制信号(补偿偏移后)的载频部分与参考频率混频器的过零点延迟，使用失配定时器生成 NCO(激励器和参考频率混频器)计算周期中的计数器值。为了消除调制信号中包络对过零点检测的影响，应在 RDCM_ZTCFG0 寄存器的 I/QZCTH[13:0]位中配置失配计数器值阈值。

在校准模式中，推荐的值是(16384 / FEXC[13:0])。

在自动模式中，推荐的值是(8192 / FEXC[13:0])。

[I/QZCTH[13:0] - I/QZTVAR[13:0], I/QZCTH[13:0] + I/QZTVAR[13:0]]之间的计数器值将被捕

获以进行评估。

过零点检测方法对信号质量敏感，且精度有限。相位误差调整的精度为 pi * FEXC[13:0] / 8192。捕获的失配计数器值可能不稳定。使用硬件统计平均值可以获得稳定的失配计数器值。

当 RDCM_MCACFG 寄存器中的 I/QAVEM[3:0] = 4'b0000 时，统计平均功能被禁用，每次捕获的失配计数器值将更新参考频率混频器的相位调整。I/QAVEM[3:0]的值越大，统计平均强度越大，但收敛时间越长。

图 2-8. RDCM 混频器配置示例

```
/* 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 解调器配置

RDCM 的解调器分为三部分的配置，分别是 CIC3 滤波器、增益调整和补偿滤波器的配置。

CIC3 滤波器

用户通过 RDCM_CICCFG0/1 寄存器的 I/QCICDFAC[8:0]的配置，将 CIC3 滤波器的抽取率(降采样率)设置在 4 到 512 之间。

表 2-8. CIC3 滤波器属性(N = I/QCICDFAC[8:0] + 1)

滤波器	最大延迟	最大输出值增益
CIC3	3N	$[-N^3, N^3]$

CIC3 滤波器输出的有效数据位部分受 CIC3 降采样因子 I/QCICDFAC[8:0]和所采用的调制器的影响。数据移位器由 RDCM_CSGCFG0/1 寄存器的 I/QCICSHIFT[5:0]控制。

I/QCICSHIFT[5:0]的值由以下公式确定：

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

N：这个值等于 I/QCICDFAC[8:0] + 1。

A_s：信号经过高通滤波器后的振幅值。

A_t：目标校准后的振幅值。

增益调整

上述公式中的舍入误差可以通过计算增益校正因子 I/QGAIN[12:0]来修正，该因子存储在 RDCM_CSGCFG0/1 寄存器中。

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

例如：

对于 N=15，信号经过高频滤波器后的振幅值 A_s 为 7000，我们期望信号经过 CIC3 增益调整后的振幅 A_t 也是 7000。

I/QCICSHIFT[5:0]的值是：

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

增益校正因子是：

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

补偿滤波器

补偿滤波器由两个 FIR 模块组成。FIR0 的降采样率为 2，FIR1 的降采样率可以设置为 1 或 2。FIR1 的降采样率由 RDCM_DEMCFG 寄存器中的 I/QFIR1DEC 位控制。相位敏感解调器的总降采样率为 2 * N 或 4 * N。

两个 FIR 滤波器的系数(有符号的 10 位，必须是对称的)是固定的。这些滤波器是对称的，并且具有线性相位。FIR0 是一个 8 阶对称滤波器，而 FIR1 是一个 28 阶对称滤波器。

FIR0/FIR1 的默认系数如下表所示。

表 2-9. FIR0 默认系数

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

表 2-10. FIR1 默认系数

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

图 2-9. RDCM 解调器配置示例

```
/*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 群延迟补偿配置

当数字滤波器链达到稳态时，延迟被称为群延迟，由滤波器配置决定。这些延迟以 RDCM 计算周期为单位进行测量，对应于 ADC 数据更新周期。

表 2-11. 滤波器元件的群延迟

滤波器	延迟	描述
高通滤波器	0	高通滤波器延迟
增益阶段	1	增益级计算的流水线延迟
非正交相位失配补偿	0	非正交相位失配补偿的流水线延迟
CIC3	3 * (N - 1) / 2	相位敏感解调器中CIC3滤波器的延迟 (N 是 CIC3 的降采样率)
FIR0	3.5 * N	相位敏感解调器中 FIR0 滤波器的延迟
FIR1	13.5 * N * 2^{F0}	相位敏感解调器中FIR1滤波器的延迟 F0 = 0 (FIR0旁置) F0 = 1 (FIR0 使能)
零阶保持	N * 2^{F0} * 2^{F1} / 2	相位敏感解调器后零阶保持的延迟 F0 = 0 (FIR0旁置) F0 = 1 (FIR0使能) F1 = 0 (FIR1旁置或抽取功能禁能) F1 = 1 (FIR1 使能且 FIR1 抽取功能使能)

实际使用时，转速越大，角度误差越大，这时需要用户在代码中做一个实时软件补偿，角度误差公式如下：

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

$\Phi_{installation}$ 为旋变安装的偏置的根据实际结构而定，可以认为是一个固定量，需要用户自己测量。$\Phi_{demodulation}$ 为解调延时造成的角度，和上表中的群延迟相关，计算公式如下：

$$\Phi_{demodulation} = \text{群延迟} / \text{ADC 采样率} * 16384 * n$$

其中 n 为旋变的实际速度，单位为圈数/s，计算公式如下：

$$n = (ADC\ 采样率\ * VELOCITY[15:0] / (16384 * 1024))$$

其中，VELOCITY[15:0]为 RDCM 解调出的速度值，存储在 RDCM_VELDATA 寄存器中。

补偿后的角度如下：

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

其中，POSITION[15:0]为 RDCM 解调出的位置值，存储在 RDCM_POSDATA 寄存器中，弧度制角度(rad)转换公式如下：

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

注意：如果计算出的$\Phi_{compensation}$大于 16384，需要映射到 0-16384 之间，计算出的 θ 大于 2pi，需要映射到 0-2pi 范围内。

2.9. 控制器局域网络(CAN)

2.9.1. CAN 过滤器的使用

CAN 总线控制器集成了可灵活配置的邮箱系统用于 CAN 帧的发送和接收。邮箱系统包含一组邮箱，用于存储控制数据，时间戳，消息标识符和消息数据，最大支持 32 个邮箱。每个 CAN 控制器内有 512 字节 RAM 空间用于邮箱或 FIFO 描述符的配置。

当 CAN 开启 FIFO 接收时，邮箱占用的 RAM 空间将用于接收 FIFO 描述符。接收 FIFO 具有标识符过滤的功能，最大支持 104 个扩展标识符的过滤，或者 208 个标准标识符的过滤，或者 416 个对标识符部分 8 位的过滤，最多有 32 个标识符过滤表元素可通过接收 FIFO/邮箱私有过滤寄存器进行配置。

当禁能Rx FIFO时：

- 如果CAN_CTL0寄存器的RPFQEN位为0，则使用CAN_RMPUBF寄存器来配置所有接收邮箱的过滤数据配置。
- 如果CAN_CTL0寄存器的RPFQEN位为1，则使用CAN_RFIFOMPFx(x = 0..31)寄存器来分别配置接收邮箱的过滤数据配置。

当使能Rx FIFO时：

- 如果CAN_CTL0寄存器的RPFQEN位为0，则使用CAN_RMPUBF寄存器来配置所有接收邮箱的过滤数据配置，使用CAN_RFIFOPUBF和CAN_RFIFOMPFx(x = 0..31)寄存器来配置所有Rx FIFO标识符过滤表元素，并且所有这些寄存器的值的配置必须相同。
- 如果CAN_CTL0寄存器的RPFQEN位为1，则使用CAN_RFIFOMPFx(x=0..31)寄存器来配置由CAN_CTL2寄存器RFFN[3:0]位域设置的Rx FIFO标识符过滤表元素以及接收邮箱(由于接收邮箱描述符和Rx FIFO描述符不能同时占用同一个区域的RAM，因此用一组寄存器进行独立控制过滤数据的配置)，由CAN_RFIFOPUBF寄存器来配置剩余所有的Rx FIFO标识符过滤表元素。

Rx FIFO标识符过滤表元素数目配置如下表示:

表 2-12. Rx FIFO 标识符过滤表元素数目配置

RFFN[3:0]	Rx FIFO标识符过滤表元素数目	Rx FIFO占用的空间	可用的邮箱
0000	8	邮箱描述符0 - 7	邮箱8 - 31
0001	16	邮箱描述符0 - 9	邮箱10 - 31
0002	24	邮箱描述符0 - 11	邮箱12 - 31
0003	32	邮箱描述符0 - 13	邮箱14 - 31
0004	40	邮箱描述符0 - 15	邮箱16 - 31
0005	48	邮箱描述符0 - 17	邮箱18 - 31
0006	56	邮箱描述符0 - 19	邮箱20 - 31
0007	64	邮箱描述符0 - 21	邮箱22 - 31
0008	72	邮箱描述符0 - 23	邮箱24 - 31
0009	80	邮箱描述符0 - 25	邮箱26 - 31
000A	88	邮箱描述符0 - 27	邮箱28 - 31
000B	96	邮箱描述符0 - 29	邮箱30 - 31
000C	104	邮箱描述符0 - 31	无
其他	104	邮箱描述符0 - 31	无

以 RFFN[3:0] = 3 为例，此时 Rx FIFO 标识符过滤表元素数目为 32，邮箱描述符 0~13 的空间被 FIFO 描述符占用，剩余邮箱 14~31 可用于邮箱发送或接收。Rx FIFO 标识符过滤元素有 3 种格式，可通过 CAN_CTL0 寄存器 FS[1:0]位域配置 Rx FIFO 标识符过滤表元素的格式。当 FS[1:0] = 0 时，为格式 A，此时最大支持 32 个完整的标准或扩展标识符；当 FS[1:0] = 1 时，为格式 B，此时最大支持 64 个完整的标准标识符或扩展标识符中的 14 个 bit；当 FS[1:0] = 2 时，为格式 C，此时最大支持 128 个标准标识符或扩展标识符中的 8 个 bit。

Rx FIFO 标识符过滤表元素的格式配置如下:

配置为格式 B，支持 64 个完整的标准标识符或扩展标识符中的 14 个 bit。

图 2-10. RxFIFO 标识符过滤表元素的格式配置

```
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);
```

因 FS[1:0] = 3，邮箱 0~13 被 FIFO 占用，故接收 FIFO/邮箱私有过滤器 0~13 用于 FIFO 接收过滤。其代码配置如下:

图 2-11. 私有过滤配置

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

2.9.2. CANFD 数据段通信参数和 TDCO 配置

GD32H77x_78x 系列与 GD32H759、GD32H757 系列对比，在 CANFD 的数据段通信参数配置存在差异，GD32H77x_78x 系列在设计上增加了数据段 DPBS2、DPBS1、DPTS 和 DSJW 的位域宽度，以满足在 CAN 模块再选择高速时钟源时数据段的通信参数配置需求。

GD32H77x_78x 系列的 CAN 模块在设计上也增加了传输延迟补偿偏置(TDCO)的位域宽度，以保证传输延迟补偿偏置(TDCO)的配置能达到预期的需求。

CANFD 数据段通信参数和 TDCO 位域宽度对比如*表 2-13. CANFD 数据段通信参数和 TDCO 差异*所示。

表 2-13. CANFD 数据段通信参数和 TDCO 差异

功能名称	位域宽度对比	
	GD32H77x_78x 系列	GD32H759/H757 系列
再同步段	DSJW[3:0]	DSJW[2:0]
传播时间断	DPTS[5:0]	DPTS[4:0]
相位缓冲段 1	DPBS1[4:0]	DPBS1[2:0]
相位缓冲段 2	DPBS2[4:0]	DPBS2[2:0]
传输延迟补偿偏置	TDCO[6:0]	TDCO[4:0]

2.10. SDIO 接口(SDIO)

2.10.1. SDIO 时钟配置

CK_SDIO 是 SDIO 模块的内核时钟，提供给 SD/eMMC 卡的时钟 SDIO_CLK 由此时钟分频获取，通过 RCU 配置 CK_SDIO 时，工作时钟 CK_SDIO 建议配置不大于 208MHz。

引脚配置 AF 功能时，建议先修改 GPIOx_AFSELx(0, 1)，再改变 GPIO 模式为复用功能。

图 2-12. SDIO 引脚初始化配置

```
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 上电初始化

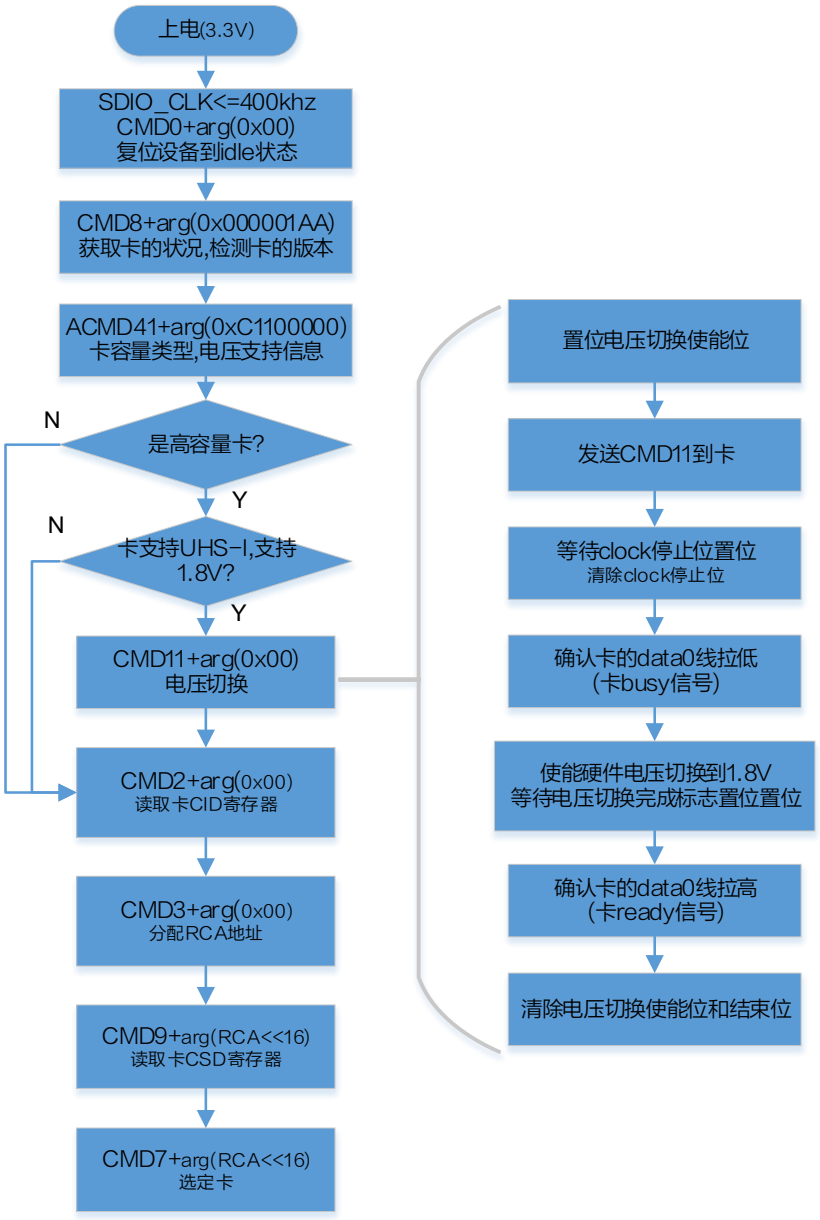
UHS-I SD 卡的上电初始化和电压切换

SD3.0 版本的卡支持 UHS-I(即超高速总线速度 I 相)速度模式，具体包括：SDR12、SDR25、

SDR50、SDR104 和 DDR50。UHS-I 工作在 1.8V 电压下，而卡上电时是 3.3V 的电压启动，所以 UHS-I 模式需要支持 3.3V 到 1.8V 的电压切换。当电压切换序列成功完成时，卡将以默认 SDR12 进入 UHS-I 模式。

卡上电初始化以及卡的状态和信息的读取，支持 UHS-I 的卡，在响应 ACMD41 时，会回复支持 1.8V 的电压切换，然后再进行 1.8V 的电压切换操作，主要流程如下图所示，另外，需要注意 SDIO0 支持 UHS-I，SDIO1 不支持 UHS-I，即 SDIO1 无法生成控制外部电压转换芯片的信号。

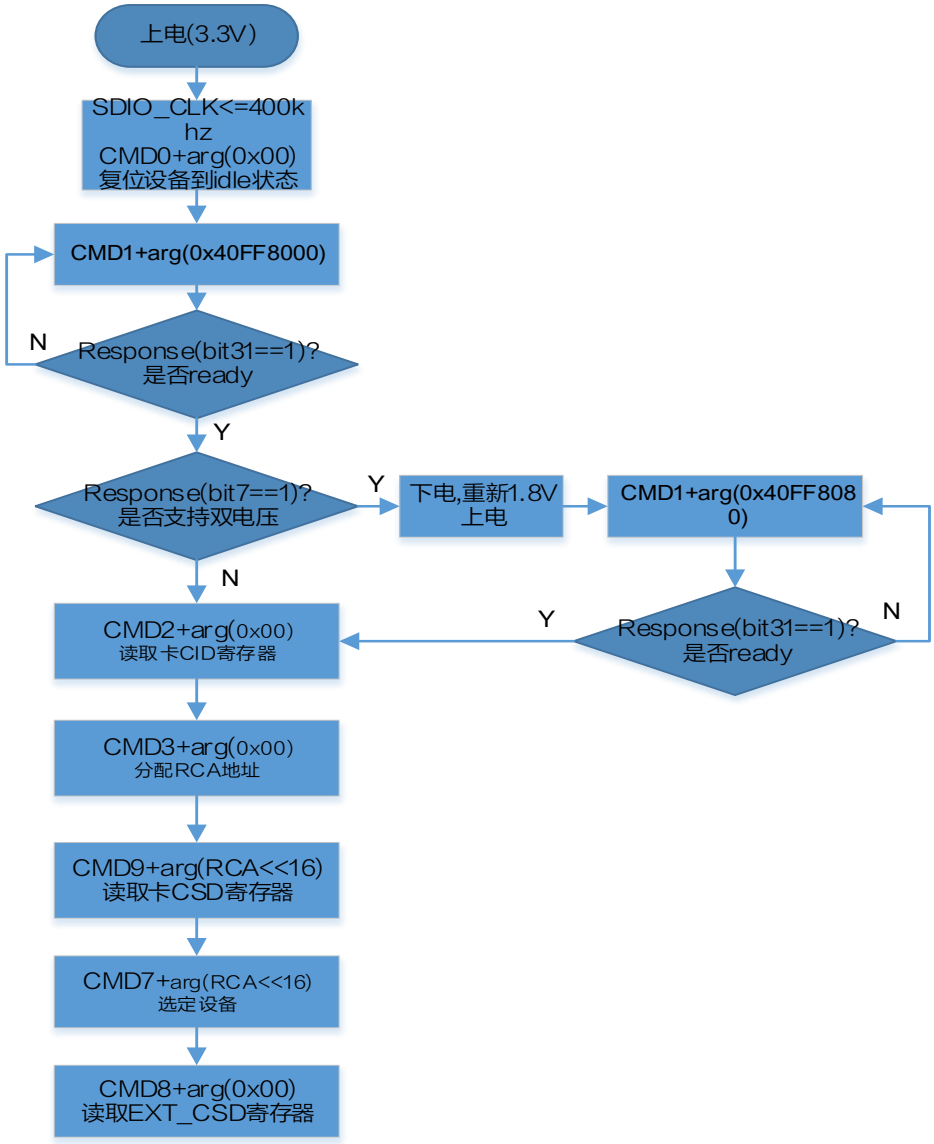
图 2-13. SD 卡上电初始化和电压切换流程



eMMC 的上电初始化

eMMC 的上电流程如下图所示，首先 host 通过 OCR 寄存器判断 eMMC 是否支持 1.8V，确定支持双电压后，然后再重新选择 1.8V 上电。

图 2-14. eMMC 上电初始化和电压切换流程



2.10.3. 总线宽度配置

Host 使用命令配置设备的总线速度，配置总线时需要在接收完 64 字节数据后，再改变主机的宽度、速度模式和 DDR。

SD 卡在上电初始化完成后，可以通过发送命令进行总线宽度和速度配置，具体依据以下表格进行，需要注意的是 UHS-I 模式下不支持 1bit 总线。

表 2-14. SD 卡总线宽度配置

选择宽度	命令	参数
1bit	ACMD6	0x00
4bit		0x02

表 2-15. SD 卡总线速度配置

选择速度模式	电压/V	命令	参数
SDR12	1.8	CMD6	0x80FFFF00
SDR25	1.8		0x80FFFF01
SDR50	1.8		0x80FF1F02
SDR104	1.8		0x80FF1F03
DDR50	1.8		0x80FF1F04

eMMC 在上电初始化完成后，可以通过发送命令进行总线宽度和速度配置，具体依据以下表格进行：

表 2-16. eMMC 总线模式配置

选择宽度	命令	参数
1BIT SDR	CMD6	0x03B70000
4BIT SDR		0x03B70100
8BIT SDR		0x03B70200
4BIT DDR		0x03B70500
8BIT DDR		0x03B70600

表 2-17. eMMC 总线速度配置

选择速度模式	电压/V	命令	参数
DS	1.8/3.3	CMD6	0x03B90000
HS	1.8/3.3		0x03B90100
HS200	1.8		0x03B90200

SD 卡的高速总线 tuning

支持 UHS-I 的 SD 卡，在 SDR50 或 SDR104 总线速度模式时，可以使用 CPDM 模块对接收数据的采样点进行 tuning，使用 CPDM 后，需要关闭硬件流控制。在发送 CMD19 进行 tuning 前，需要先进行 CPDM 配置延迟线长度。

CPDM 配置延迟线长度方式参考代码如下。

图 2-15. CPDM 配置

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

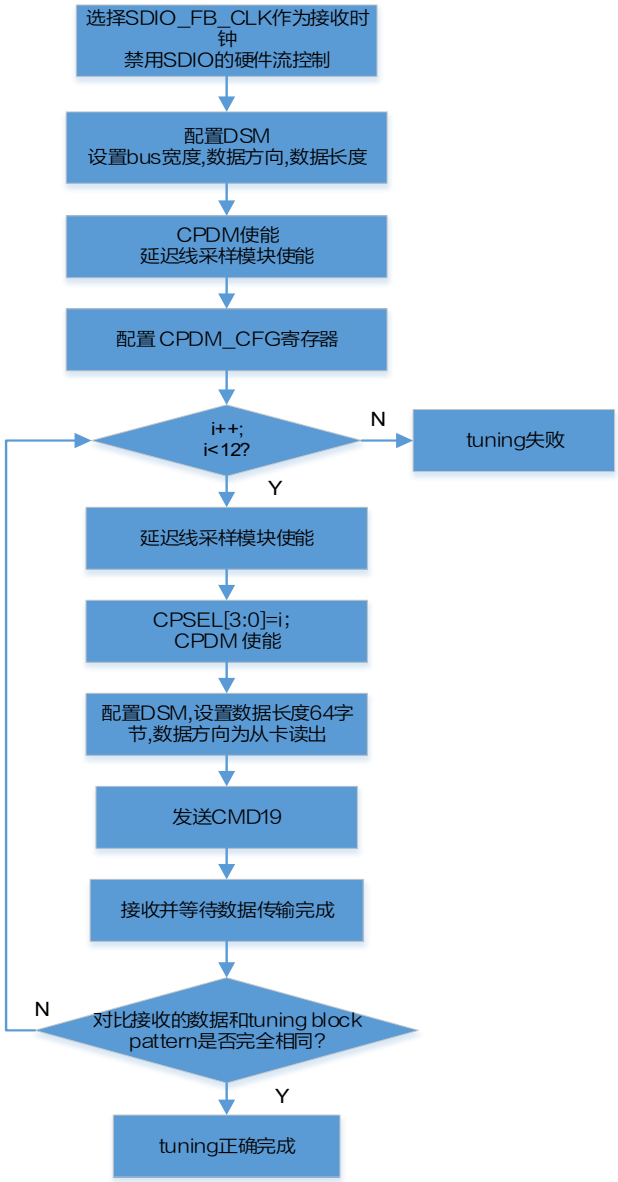
    for(i = 0; i < ((uint32_t)0x00000080U); i++) {
        CPDM_CFG(CPDM_SDIO0) = temp | (i << 8);

        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;
        }
    }
}
```

图 2-16. SD 卡 tuning 流程



2.10.4. 访问 eMMC Boot 分区数据

eMMC 有两个相同大小的 Boot 分区，Boot 分区的 size， host 通过发送 CMD8 获取的 EXT_CSD 寄存器计算得到，计算公式为 EXT_CSD[226]*128K byte。Boot 分区的数据读写和其他分区一样，只是 host 在发送读写命令前，需要发送 CMD6 切换到 boot 分区，具体命令如下表格所示，Boot 分区的数据在写入后，可以在上电初始化时自动读取到，具体操作需要依据协议进行，读取 Boot 分区的内容总线配置也需要提前依据协议配置完成。

表 2-18. 访问分区的命令和参数

切换分区	命令	参数

用户分区	CMD6	0x03B30000
boot 分区 1		0x03B30100
boot 分区 2		0x03B30200

2.10.5. MDMA 配置

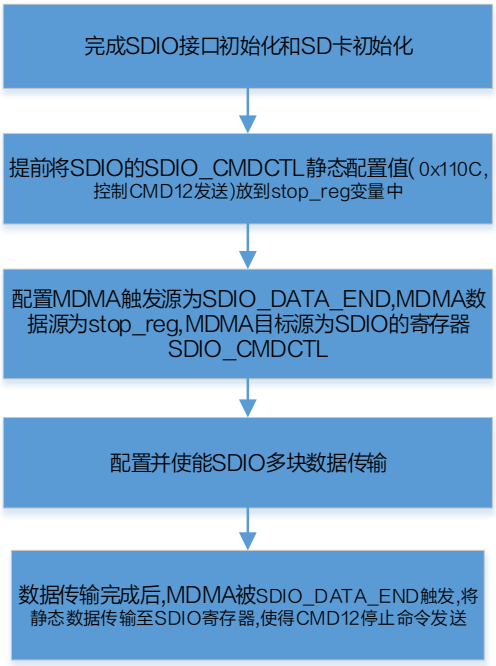
使用 MDMA 控制 SDIO 传输过程

不使用 MDMA 时，CPU 会对 SDIO 传输过程进行控制，例如：多块数据传输完成后，发送 CMD12 停止命令；双缓冲传输完成一次缓冲区传输后，改变 IDMA 中缓冲区地址配置等。

如果使用 MDMA，可以在三个时间阶段代替 CPU 对 SDIO 传输进行控制：数据传输结束时刻(SDIO_DATA_END)、命令时序结束时刻(SDIO_CMD_END)、缓冲区传输完成时刻(SDIO_BUF_END)。当 MDMA 在这些时刻被触发时，会将提前配置好的静态数据传输到 SDIO 寄存器中，实现对 SDIO 的控制。

以下流程图举例为 SDIO 多块传输完成后，MDMA 控制 CMD12 的发送。

图 2-17. MDMA 控制 CMD12 流程图

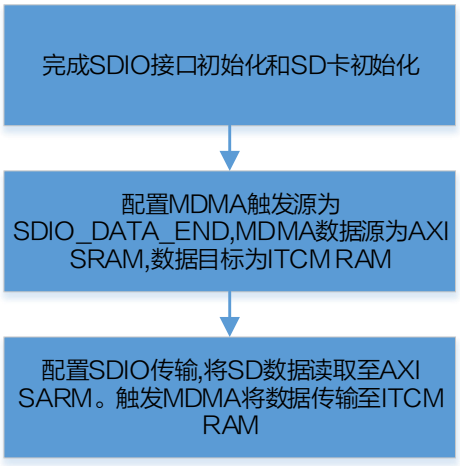


另外，MDMA 支持链表，所以可使用多节点链表对 SDIO 进行多次传输控制。

使用 MDMA 传输数据至 DTCM/ITCM

下图是 SDIO 和 MDMA 一起使用的例程，目的是配置 SDIO_DATA_END 作为 MDMA 触发源，触发将 SD 卡数据传输至 ITCM。

图 2-18. MDMA 控制 RAM 数据传输流程图



2.10.6. IDMA 配置

SDIO 有内部 DMA(IDMA)，支持单缓冲传输和双缓冲传输。

在单缓冲模式，只需在传输前配置 SDIO 寄存器 SDIO_IDMACTL 为单缓冲模式和 IDMA 使能，从而使用 IDMA 进行单缓冲传输。

在双缓冲模式，SDIO 会轮流在两个缓冲区读取或者写入数据，当 SDIO 在使用一个缓冲区时，CPU 可以对另一个缓冲区进行数据读写。需配置 SDIO 寄存器 SDIO_IDMACTL 为双缓冲模式和 IDMA 使能，配置 SDIO_IDMASIZE 为每个缓冲区的大小，配置 SDIO_IDMAADDR0 和 SDIO_IDMAADDR1 为外部两个双缓冲区地址，可选择开启 IDMAEND 中断。使能 SDIO 数据传输，SDIO 会轮流访问两个缓冲区，当一个缓冲区传输完成后，IDMAEND 中断触发，根据 SDIO_IDMACTL 中的 BUFSEL，判断哪个缓冲区未被 SDIO 使用，可由 CPU 访问。

2.11. 外部存储器控制器(EXMC)

2.11.1. EXMC SDRAM 非对齐访问进 hardfault 问题

在默认情况下，即未对 SDRAM 的访问属性做修改，采用非对齐的方式访问 SDRAM 会进 hardfault，原因是 0xC0000000-0xDFFFFFFF 默认为 Device 类型，不支持非对齐访问，具体参考下表。

表 2-19. ARMv7-M 地址映射

地址	名称	内存类型(s)	XN	Cache	描述 / 支持的存储器
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

如果需要对 SDRAM 非对齐访问，则需要使用 MPU 修改 SDRAM 所在地址空间的类型，如改为 RAM 一样的类型，例如将 0xC0000000 配置为 Write Through 的属性。

表 2-20. EXMC SDRAM MPU 配置

```
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. 串行音频接口(SAI)

2.12.1. SAI 使用 DMA 突发传输发送数据注意事项

GD32H77x_78x SAI 每个音频子模块都具有一个 8 字大小的 FIFO，并且每个音频子模块都拥有自己的 DMA 接口。DMA 访问的使能通过 SAI_BxCFG0 寄存器的 DMA 使能位(DMAEN)进行配置。DMA 请求和 FIFO 请求(FFREQ)一起产生，而 FIFO 请求产生状态取决于 FIFO 阈值(FFTH)和 FIFO 状态(FFSTAT)，这在使用 DMA 突发传输时是非常重要的。

下表列出了推荐的 DMA Burst 传输和 SAI FIFO 阈值之间的关系。

表 2-21. DMA Burst 传输和 SAI FIFO 阈值

DMA 存储器传输宽度	DMA 突发传输类型	SAI 数据宽度	SAI FIFO 阈值
16	单一	16	空、1/4 满、半满或 3/4 满
	4 拍增量突发传输		空或 1/4 满
32	单一	32	空、1/4 满、半满或 3/4 满
	4 拍增量突发传输		空或 1/4 满

当音频子模块配成发送模式时，FIFO 阈值必须设成一个特定的值，以保证在最坏的情况下也有足够的剩余空间来实现一个完整的 DMA 突发写操作，否则有可能出现 FIFO 上溢错误。当音频子模块配成接收模式时，FIFO 阈值必须设成一个特定的值，以保证 FIFO 中有足够的数据来实现一个完整的 DMA 突发读操作，从而避免出现 FIFO 下溢错误。

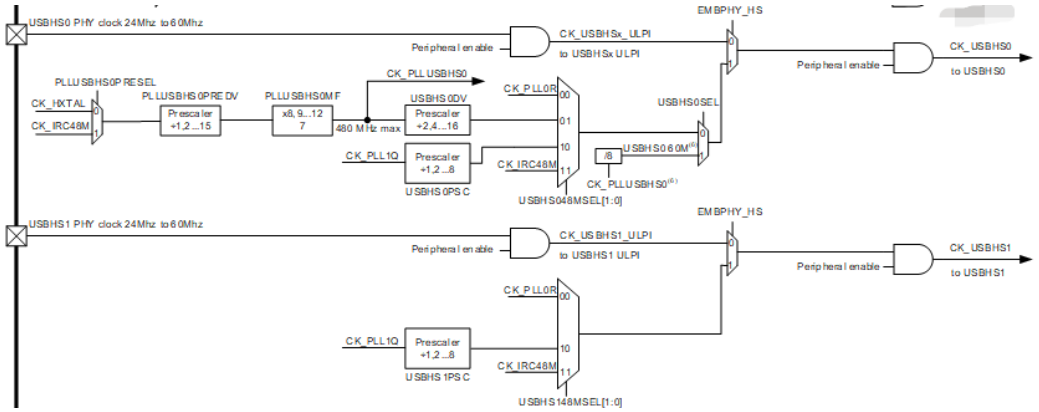
2.13. 通用串行总线高速接口(USBHS)

2.13.1. USBHS 使用注意事项

GD32H77x_78x 系列的 USBHS 模块的注意事项如下：

（0） USBHS 的时钟配置

图 2-19. USBHS 时钟树图



USBHS0 和 USBHS1 的时钟配置有区别。USBHS048MSEL 选择器可以选择四种时钟源(CK_PLL0R,CK_PLLUSBHS0,CK_PLL1Q 和 CK_IRC48M)，而 USBHS148MSEL 只有三种时钟源(CK_PLL0R,CK_PLL1Q 和 CK_IRC48M)。

USBHS0SEL 的选择器可以选择 USBHS048MSEL 或者 USBHS60M,其中 USBHS060M 由 480M 的 PHY 时钟分频所得。若通过软件进行选择，则需要置位。RCU_USBCLKCTL 寄存器的 USBHS0SWEN 位。若 RCU_USBCLKCTL 寄存器的 USBHS0SWEN 位为 0，则默认由硬件选择，硬件会在 chirp 握手为高速时，将时钟源选择为 USBHS060M。

USBHS0 通过上述配置来启动嵌入式 PHY 的 USB 高速应用,并且可以通过 EMBPHY_HS 选择 ULPI 的时钟来启动 USB 高速应用。而 USB HS1 没有这些时钟选择，只能通过 EMBPHY_HS 选择 ULPI 的时钟来启动 USB 高速应用。

(2)USBHS 的内外部 phy 的配置

USBHS0 的嵌入式 USB PHY 可以配置成高速或者全速。但是 USBHS1 的嵌入式 USB PHY 只能配置成全速。

USBHS0 和 USBHS1 都支持通过外部 ULPI PHY 完成高速 USB 应用。

(3)USBHS其他特性

USBHS0支持OTG模式，全速设备的操作也支持电池充电检测(BCD)、附加检测协议(ADP)。而USBHS1不支持OTG模式，也不支持BCD和ADP。

GD32H77x_78x 系列的 USBHS 外设的其他操作如 USBHS 的 GPIO 配置，USBHS 的电源配置，其与 GD32H7xx 系列的 USBHS 相同。具体内容参见*《AN117 GD32H7xx 系列 USBHS 使用注意事项》*。

2.14. 以太网(ENET)

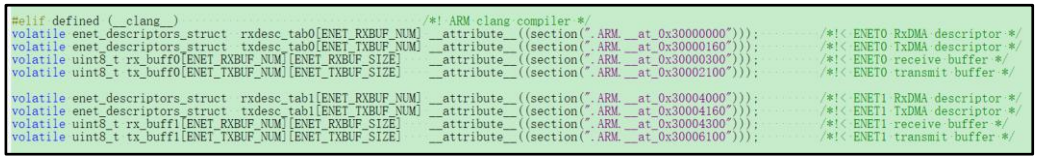
2.14.1. ENET cache 开启的情况

ENET 模块在使能 cache 后,同时以太网模块会使用 DMA 机制,以太网数据由于数据流量大，

会通过 DMA 快速更新内存，此时可能会导致 DMA 获取数据和 cache 中的数据不一致。

因此我们首先需要指定太网接收/发送描述符以及接收/发送缓存所用的内存区域地址。

图 2-20. ENET DMA 描述符和缓存定义



```
#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 */
```

并通过使用 MPU 模块对所用的内存区域的 cache 访问权进行设置，配置为不缓冲和不缓存，来保证数据的实时性和准确性。

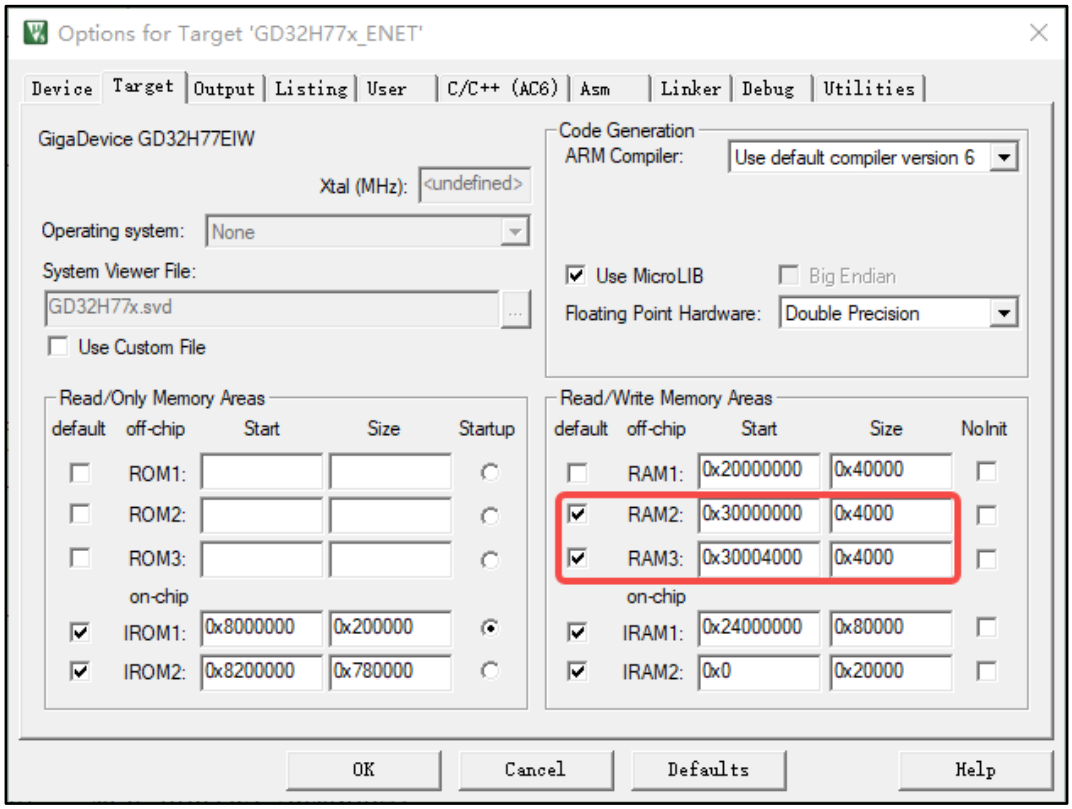
图 2-21. ENET DMA 描述符和缓存 MPU 配置

```
/* 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();
```

同时，在工程的相关设置中需要选中所设置的 RAM 区域，否则无效。

图 2-22. ENET 工程配置



2.15. EtherCAT 从站控制(ESC)

具体内容参见*《AN246 EtherCAT 协议栈代码移植参考手册》*。

3. 版本历史

表 3-1. 版本历史

版本号	说明	日期
1.0	首次发布	2026 年 11 月 09 日

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