应用笔记

ACM32H5 系列芯片

FMC-SDRAM 应用笔记

版本：V1.1

日期：2024-12-11

上海航芯电子科技股份有限公司

1. 概述

本应用手册适用于 ACM32H5 系列芯片 FMC-SDRAM 模块。它描述了与 FMC-SDRAM 模块相关的设置和功能使用方法，以便在应用程序中进行优化设计。

本应用说明应与相关的用户手册、数据手册一同阅读。

2. SDRAM 模块初始化

2.1. 主要特性

- 两个 SDRAM 存储区域，可独立配置
- 16 位和 32 位数据总线宽度
- 13 位行地址，11 位列地址，4 个内部存储区域：4x16Mx32bit (256 MB)、4x16Mx16bit(128 MB)
- 支持字、半字和字节访问
- SDRAM 时钟可以是 HCLK/2 、HCLK/3 或 HCLK/4
- 自动进行行和 Bank 边界管理
- 多 Bank 乒乓访问
- 可编程时序参数
- 支持自动刷新操作，可编程刷新速率
- 自刷新模式
- 掉电模式
- 通过软件进行 SDRAM 上电初始化
- CAS 延迟 1,2,3
- 读 FIFO 可缓存，支持 8 行 x 32 位深度（ 8 x14 位地址标记）

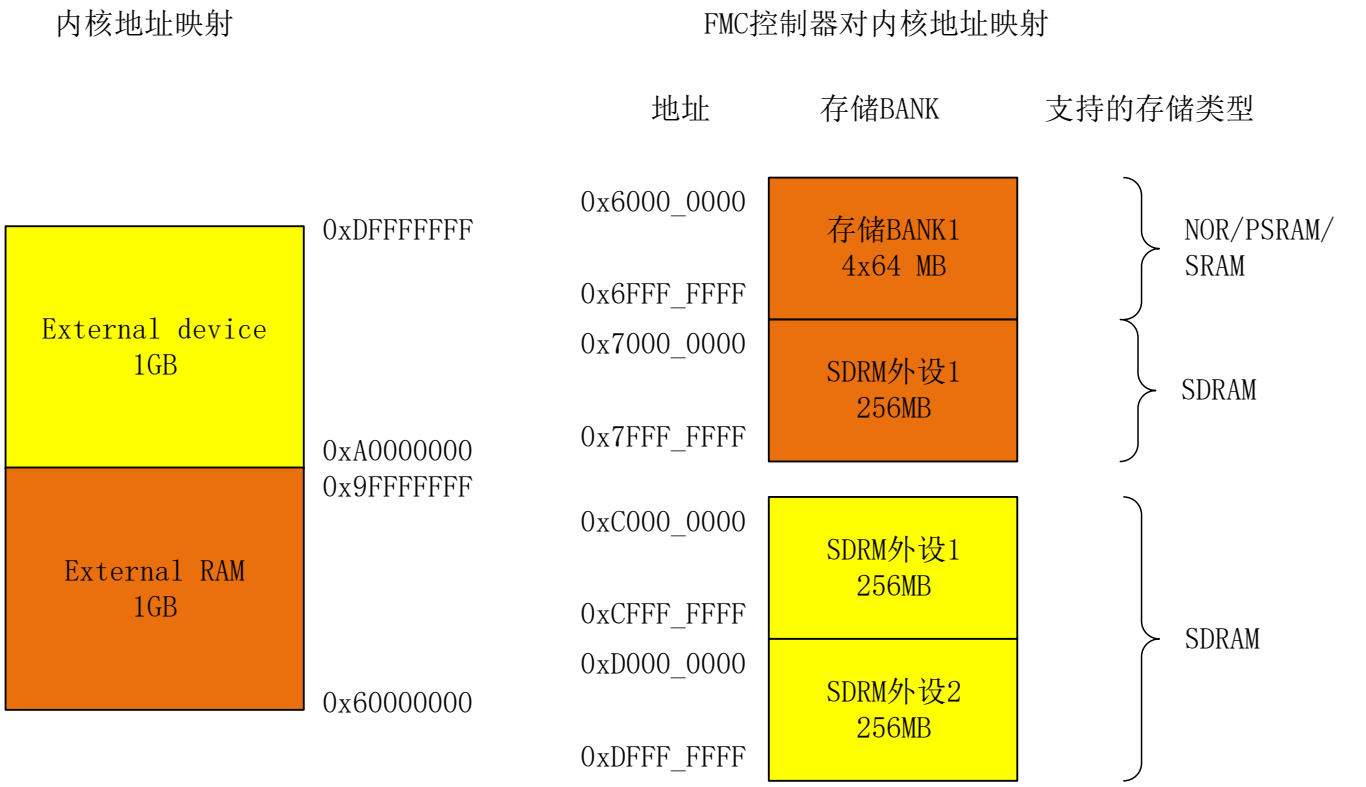
2.2. 接口信号

FMC 引脚名称	对应 SDRAM 引脚名称	说明
FMC_NBL[3:0]	DQM[3:0]	数据掩码
FMC_A[12:0]	A[12:0]	地址线
FMC_A[15:14]	BA[1:0]	Bank 地址线
FMC_D[31:0]	DQ[31:0]	数据线
FMC_SDCLK	CLK	SDRAM 时钟
FMC_SDNWE	WE#	写入使能
FMC_SDCKE[1:0]	CKE	SDCKE0：SDRAM 存储区域 1 时钟使能 SDCKE1：SDRAM 存储区域 2 时钟使能
FMC_SDNE[1:0]	-	SDNE0：SDRAM 存储区域 1 芯片使能 SDNE1：SDRAM 存储区域 2 芯片使能
FMC_NRAS	RAS#	行地址选通
FMC_NCAS	CAS#	列地址选通

其中比较特殊的是 FMC_A[15:14]引脚用作 Bank 的寻址线；而 FMC_SDCKE 线和 FMC_SDNE 都各有 2 条，FMC_SDCK 用于控制 SDRAM 的时钟使能，FMC_SDNE 用于控制 SDRAM 芯片的片选使能。它们用于控制

ACM32H5 使用不同的存储区域驱动 SDRAM，使用编号为 0 的信号线组会使用 ACM32H5 的存储器区域 1（SDRAM 外设 Bank1），使用编号为 1 的信号线组会使用存储器区域 2（SDRAM 外设 Bank2）。使用不同存储区域时，ACM32H5 访问 SDRAM 的地址不一样，具体将在“2.3 地址映射”小节讲解。

2.3. 地址映射



起始地址-结束地址	FMC_SWP[1:0]=00 默认映射	FMC_SWP[1:0]=01 交换 NOR/PSRAM 与 SDRAM	FMC_SWP[1:0]=10 重映射 SDRAM 存储区域 2
0x60000000-0x6FFFFFFF	NOR/PSRAM 区域	SDRAM 外设 1	NOR/PSRAM 区域
0x70000000-0x7FFFFFFF	SDRAM 外设 1	SDRAM 外设 2	SDRAM 外设 2
0xC0000000-0xCFFFFFFF	SDRAM 外设 1	NOR/PSRAM 区域	SDRAM 外设 1
0xD0000000-0xDFFFFFFF	SDRAM 外设 2	SDRAM 外设 2	SDRAM 外设 2

注意：External RAM 区可以直接执行代码，而 External device 区却不支持这个功能。

2.4. 结构体说明

控制 FMC 使用 SDRAM 存储器时主要配置控制寄存器以及时序寄存器。利用 HAL 库的 SDRAM 初始化结构体、时序结构体以及命令结构体可以很方便地写入参数。下面将描述具体的配置方法和过程。

首先定义一个 SDRAM 的初始化结构体变量，例如：

```
FMC_SDRAMInitTypeDef hsdram_Init;
```

定义一个 SDRAM 时序结构体变量，例如：

```
FMC_SDRAMTimingInitTypeDef hsdram_Timing;
```

以及定义一个 SDRAM 命令结构体变量，例如：

```
FMC_SDRAMCommandTypeDef hsdram_Command;
```

2.4.1. SDRAM 初始化结构体

```
typedef struct
{
    uint32_t SDBank;                /* 选择 FMC 映射的 SDRAM 存储区域 */
    uint32_t ColumnBitsNumber;      /* 定义 SDRAM 的列地址宽度 */
    uint32_t RowBitsNumber;         /* 定义 SDRAM 的行地址宽度 */
    uint32_t MemoryDataWidth;       /* 定义 SDRAM 的数据宽度 */
    uint32_t InternalBankNumber;    /* 定义 SDRAM 内部的 Bank 数量 */
    uint32_t CASLatency;            /* 定义 CASLatency 的时钟个数 */
    uint32_t WriteProtection;       /* 定义是否使能写保护 */
    uint32_t SDClockPeriod;         /* 配置同步时钟 SDCLK 的分频因子 */
    uint32_t ReadBurst;             /* 定义是否使能突发读模式 */
    uint32_t ReadPipeDelay;         /* 定义在 CAS 个延迟后等待多少个 HCLK 时钟才读取数据 */
}FMC_SDRAMInitTypeDef;
```

- SDBank：

用于选择 FMC 映射的 SDRAM 存储区域，可选择存储区域 1 或 2(FMC_SDRAM_BANK1/2)。

```
hsdram_Init.SDBank = FMC_SDRAM_BANK1;  //这里选择 FMC 的 SDRAM 存储区域 1，对应 0xC0000000~0xCFFFFFFF。
```

- ColumnBitsNumber：

用于设置 SDRAM 的列地址宽度，可选择 8-11 位(FMC_SDRAM_COLUMN_BITS_NUM_8/9/10/11)。

```
hsdram_Init.ColumnBitsNumber = FMC_SDRAM_COLUMN_BITS_NUM_9;  //SDRAM 的列地址宽度为 9 位。
```

- RowBitsNumber：

用于设置 SDRAM 的行地址宽度，可选择 11-13 位(FMC_SDRAM_ROW_BITS_NUM_11/12/13)。

```
hsdram_Init.RowBitsNumber = FMC_SDRAM_ROW_BITS_NUM_13;  //SDRAM 的行地址宽度为 13 位。
```

- MemoryDataWidth：

用于设置 SDRAM 的数据宽度，可选择 8、16 或 32 位(FMC_SDRAM_MEM_BUS_WIDTH_8/16/32)。

```
hsdram_Init.MemoryDataWidth = FMC_SDRAM_MEM_BUS_WIDTH_32;  //SDRAM 的数据宽度位 32 位
```

- InternalBankNumber：

用于设置 SDRAM 的内部 Bank 数量，可选择 2 或 4 个 Bank (FMC_SDRAM_INTERN_BANKS_NUM_2/4)。

```
hsdram_Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;  //SDRAM 的内部有 4 个 Bank
```

- CASLatency：

用于设置 CAS 即 CL 的时钟数量，可选择 1、2 或 3 个时钟周期(FMC_SDRAM_CAS_LATENCY_1/2/3)。

```
hsdram_Init.CASLatency = FMC_SDRAM_CAS_LATENCY_2;  //CASLatency 即 CL 的时钟数目为 2。
```

- WriteProtection：

用于设置是否使能写保护模式，如果使能了写保护则不能向 SDRAM 写入数据，正常使用都是禁止写保护的。

```
hsdram_Init.WriteProtection = FMC_SDRAM_WRITE_PROTECTION_DISABLE;  // 禁止写保护。
```

● SDClockPeriod：

用于设置 FMC 与外部 SDRAM 通讯时的同步时钟参数，可以设置成 HCLK 时钟频率的 1/2、1/3、1/4 或禁止输出时钟(FMC_SDRAM_CLOCK_PERIOD_2/3/4 或 FMC_SDRAM_CLOCK_DISABLE)。

```
hsdram_Init.SDClockPeriod = FMC_SDRAM_CLOCK_PERIOD_2;  //设置 SDCLK 为 HCLK 的 1/2
```

● ReadBurst：

用于设置是否使能突发读取模式，禁止时等效于 BL=1，使能时 BL 的值等于控制寄存器 BDEPTH 位的配置。

```
hsdram_Init.ReadBurst = FMC_SDRAM_RBURST_ENABLE;  //使能突发读取模式，默认缓存深度 BL=1
```

● ReadPipeDelay：

用于配置在 CASLatency 个时钟周期后，再等待多少个 HCLK 时钟周期才进行数据采样，在确保正确的前提下，这个值设置为越短越好，可选择设 0、1 或 2 个 HCLK 时钟周期(FMC_SDRAM_RPIPE_DELAY_0/1/2)。

```
hsdram_Init.ReadPipeDelay = FMC_SDRAM_RPIPE_DELAY_1;  //在 CAS 个时钟周期后，再等待 1 个 HCLK 才进行数据采样
```

2.4.2. SDRAM 时序结构体

```
typedef struct
{
    uint32_t LoadToActiveDelay;        /* TMRD：加载模式寄存器命令后的延迟 */
    uint32_t ExitSelfRefreshDelay;     /* TXSR：退出自刷新命令后的延迟*/
    uint32_t SelfRefreshTime;          /* TRAS：自刷新时间 */
    uint32_t RowCycleDelay;            /* TRC：行循环延迟 */
    uint32_t WriteRecoveryTime;        /* TWR：恢复延迟 */
    uint32_t RPDelay;                  /* TRP：行预充电延迟 */
    uint32_t RCDDelay;                 /* TRCD：行到列延迟 */
}FMC_SDRAMTimingInitTypeDef;
```

时序结构体各个成员值的单位为时钟 SDCLK 的周期数，可在使用的 SDRAM 芯片规格书中查询这些参数。

● LoadToActiveDelay：

设置 TMRD 延迟(Load Mode Register to Active)，即发送加载模式寄存器命令后要等待的时间，过了这段时间才可以发送行有效或刷新命令。

```
hsdram_Timing.LoadToActiveDelay = 2;  //TMRD：发送加载模式寄存器命令后要等待 2*SDCLK
```

● ExitSelfRefreshDelay：

设置退出 TXSR 延迟(Exit Self-refresh delay)，即退出自我刷新命令后要等待的时间，过了这段时间才可以发送行有效命令。

```
hsdram_Timing.ExitSelfRefreshDelay = 7;  //TXSR：退出自我刷新命令后要等待 7*SDCLK
```

● SelfRefreshTime：

设置自我刷新时间 TRAS(Self refresh time)，即发送行有效命令后要等待的时间，过了这段时间后才执行预充电命令。

```
hsdram_Timing.SelfRefreshTime = 5;  //TRAS：发送行有效命令后要等待 5*SDCLK
```

● RowCycleDelay：

设置 TRC 延迟(Row cycle delay)，即两个行有效命令之间的延迟，以及两个相邻刷新命令之间的延迟。

```
hsdram_Timing.RowCycleDelay = 7;  //TRC：两个行有效命令之间延迟，以及两个相邻刷新命令之间的延迟为 7*SDCLK
```

● WriteRecoveryTime：

设置 TWR 延迟(Recovery delay)，即写命令和预充电命令之间的延迟，过了这段时间后才执行预充电命令。

```
hsdram_Timing.WriteRecoveryTime = 2;  //TWR：写命令和预充电命令之间的延迟为 2*SDCLK
```

● RPDelay：

设置 TRP 延迟(Row precharge delay)，即预充电命令与其它命令之间的延迟。

```
hsdram_Timing.RPDelay = 3;  //TRP：预充电和其他命令之间的延迟为 3*SDCLK
```

● RCDDelay：

设置 TRCD 延迟(Row to column delay)，即行有效命令到列读写命令之间的延迟。

```
hsdram_Timing.RCDDelay = 3;  //TRCD：行有效命令到列读命令之间的延迟为 3*SDCLK
```

2.4.3. SDRAM 命令结构体

```
typedef struct
{
    uint32_t CommandMode;            /* 要发送的命令 */
    uint32_t CommandTarget;          /* 目标存储器区域 */
    uint32_t AutoRefreshNumber;      /* 若发送的是自动刷新命令，此处为发送的刷新次数，其它命令时无效 */
    uint32_t ModeRegisterDefinition; /* 若发送的是加载模式寄存器命令，此处为要写入 SDRAM 模式寄存器的参数 */
}FMC_SDRAMCommandTypeDef;
```

● CommandMode：

用于配置将要发送的命令。

● 宏	● 命令说明
● FMC_SDRAM_CMD_NORMAL_MODE	● 正常模式命令
● FMC_SDRAM_CMD_CLK_ENABLE	● 使能 CLK 命令
● FMC_SDRAM_CMD_PALL	● 对所有 Bank 预充电命令
● FMC_SDRAM_CMD_AUTOREFRESH_MODE	● 自动刷新命令
● FMC_SDRAM_CMD_LOAD_MODE	● 加载模式寄存器命令
● FMC_SDRAM_CMD_SELFREFRESH_MODE	● 自我刷新命令
● FMC_SDRAM_CMD_POWERDOWN_MODE	● 掉电命令

```
hsdram_Command.CommandMode = FMC_SDRAM_CMD_CLK_ENABLE;  //使能 SDRAM 的 CLK
```

- CommandTarget：

用于选择要控制的 FMC 存储区域，可选择 1、2 或 1 和 2(FMC_SDRAM_CMD_TARGET_BANK1/2/1_2);

```
hsdram_Command.CommandTarget = FMC_COMMAND_TARGET_BANK;  //SDRAM 存储区域 1
```

- AutoRefreshNumber：

有时需要连续发送多个 “自动刷新”(Auto Refresh)命令时，配置本成员即可控制它发送多少次，可输入参数值为 1-16，若发送的是其它命令，本参数值无效。如 CommandMode 成员被配置为宏 FMC_SDRAM_CMD_AUTOREFRESH_MODE，而 AutoRefreshNumber 被设置为 2 时，FMC 就会控制发送 2 次自动刷新命令。

```
hsdram_Command.AutoRefreshNumber = 4;  //如发送的命令为 FMC_SDRAM_CMD_SELFREFRESH_MODE，表示 FMC 会控制发送 4 次自动刷新命令
```

- ModeRegisterDefinition：

当向 SDRAM 发送加载模式寄存器命令时，这个结构体成员的值将通过地址线发送到 SDRAM 的模式寄存器中，这个成员值长度为 13 位，各个位一一对应 SDRAM 的模式寄存器。

```
hsdram_Command.ModeRegisterDefinition = 0;
```

3. 代码分析

3.1. 编程流程

（1）初始化通讯使用的目标引脚及端口时钟；

（2）使能 FMC 外设时钟；

（3）配置 FMC SDRAM 的时序、工作模式；

（4）根据 SDRAM 的初始化流程编写初始化函数；

（5）建立机制访问外部 SDRAM 存储器；

（6）编写测试程序，对 SDRAM 进行读写测试，并进行校验。

3.2. 初始化 FMC 的 GPIO

```
void HAL_FMC_SDRAM_MspInit(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    __HAL_RCC_GPIOL_CLK_ENABLE();//开启 GPIOL 时钟
    GPIO_InitStructure.Mode      = GPIO_MODE_AF_PP;//配置为复用功能
    GPIO_InitStructure.Pull      = GPIO_PULLUP;
    GPIO_InitStructure.Alternate = GPIO_FUNCTION_15;//AF15
    GPIO_InitStructure.Drive     = GPIO_DRIVE_LEVEL3;
    GPIO_InitStructure.Pin       = SDRAM_A0_GPIO_PIN;
    HAL_GPIO_Init(SDRAM_A0_GPIO_PORT, &GPIO_InitStructure);
    /* 所有 GPIO 的配置都相同，此处省略大量引脚初始化，
    具体请查看工程 acm32h5xx_hal_msp.c 中的代码 */

    SYSCFG->SYSCR |= BIT7;//SDRAM 专用 IO 映射
}
```

3.3. 配置 FMC 的模式

```
void SDRAM_Init(void)
{
    FMC_SDRAMInitTypeDef hsdram_Init;
    FMC_SDRAMTimingInitTypeDef hsdram_Timing;

    hsdram_Init.SDBank = FMC_SDRAM_BANK1;
    hsdram_Init.ColumnBitsNumber = FMC_SDRAM_COLUMN_BITS_NUM_9;
    hsdram_Init.RowBitsNumber = FMC_SDRAM_ROW_BITS_NUM_13;
```

```
    hsdram_Init.MemoryDataWidth = FMC_SDRAM_MEM_BUS_WIDTH_32;
    hsdram_Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
    hsdram_Init.CASLatency = FMC_SDRAM_CAS_LATENCY_2;
    hsdram_Init.WriteProtection = FMC_SDRAM_WRITE_PROTECTION_DISABLE;
    hsdram_Init.SDClockPeriod = FMC_SDRAM_CLOCK_PERIOD_2;
    hsdram_Init.ReadBurst = FMC_SDRAM_RBURST_ENABLE;
    hsdram_Init.ReadPipeDelay = FMC_SDRAM_RPIPE_DELAY_1;

    hsdram_Timing.LoadToActiveDelay = 2;
    hsdram_Timing.ExitSelfRefreshDelay = 7;
    hsdram_Timing.SelfRefreshTime = 5;
    hsdram_Timing.RowCycleDelay = 7;
    hsdram_Timing.WriteRecoveryTime = 2;
    hsdram_Timing.RPDelay = 3;
    hsdram_Timing.RCDDelay = 3;

    HAL_FMC_SDRAM_Init(&hsdram_Init);
    HAL_FMC_SDRAM_Timing_Init(&hsdram_Timing, hsdram_Init.SDBank);

    /* FMC SDRAM device initialization sequence */
    SDRAM_InitSequence();
}
```

这个函数的执行流程如下：

（1）使能 FMC 时钟，初始化 GPIO 引脚；

（2）配置 SDRAM 初始化结构体；

（3）配置 SDRAM 时序结构体；

（4）调用 HAL_FMC_SDRAM_Init 和 HAL_FMC_SDRAM_Timing_Init 把初始化结构体和时序结构体的各种参数写入到 FMC_SDRCR 控制寄存器及 FMC_SDRTR 时序寄存器中；

（5）调用 SDRAM_InitSequence 函数实现执行 SDRAM 的上电初始化时序。

3.4. SDRAM 的初始化时序

```
void SDRAM_InitSequence(void)
{
    uint32_t tmpr = 0;
    FMC_SDRAMCommandTypeDef hsdram_Command;
    /* 第一步：配置命令：开启提供给 SDRAM 的时钟 */
    hsdram_Command.CommandMode = FMC_SDRAM_CMD_CLK_ENABLE;
```

```
    hsdram_Command.CommandTarget = FMC_COMMAND_TARGET_BANK;
    hsdram_Command.AutoRefreshNumber = 1;
    hsdram_Command.ModeRegisterDefinition = 0;
    HAL_FMC_SDRAM_SendCommand(&hsdram_Command, SDRAM_TIMEOUT);
    /* 第二步: 插入 100us 最小延时 */
    HAL_SimpleDelay(50000);
    /* 第三步：配置命令：对所有的 bank 预充电 */
    hsdram_Command.CommandMode = FMC_SDRAM_CMD_PALL;
    hsdram_Command.CommandTarget = FMC_COMMAND_TARGET_BANK;
    hsdram_Command.AutoRefreshNumber = 1;
    hsdram_Command.ModeRegisterDefinition = 0;
    HAL_FMC_SDRAM_SendCommand(&hsdram_Command, SDRAM_TIMEOUT);
    /* 第四步：配置命令：自动刷新 */
    hsdram_Command.CommandMode = FMC_SDRAM_CMD_AUTOREFRESH_MODE;
    hsdram_Command.CommandTarget = FMC_COMMAND_TARGET_BANK;
    hsdram_Command.AutoRefreshNumber = 4;
    hsdram_Command.ModeRegisterDefinition = 0;
    HAL_FMC_SDRAM_SendCommand(&hsdram_Command, SDRAM_TIMEOUT);
    /* 第五步：设置加载模式寄存器配置 */
    tmpr = (uint32_t)FMC_SDRAM_LOAD_MODE_BURST_LENGTH_1             |
                     FMC_SDRAM_LOAD_MODE_BURST_TYPE_SEQUENTIAL    |
                     FMC_SDRAM_LOAD_MODE_CAS_LATENCY_2            |
                     FMC_SDRAM_LOAD_MODE_OPERATING_MODE_STANDARD  |
                     FMC_SDRAM_LOAD_MODE_WRITEBURST_MODE_SINGLE;

    hsdram_Command.CommandMode = FMC_SDRAM_CMD_LOAD_MODE;
    hsdram_Command.CommandTarget = FMC_COMMAND_TARGET_BANK;
    hsdram_Command.AutoRefreshNumber = 1;
    hsdram_Command.ModeRegisterDefinition = tmpr;
    HAL_FMC_SDRAM_SendCommand(&hsdram_Command, SDRAM_TIMEOUT);
    /* 第六步：设置刷新时间 */
    /* (7.8125 us x Freq) - 50 */
    HAL_FMC_SDRAM_ProgramRefreshRate(800);
}
```

3.5. 使用指针的方式访问 SDRAM 存储器

完成初始化 SDRAM 后，我们就可以利用它存储数据了，由于 SDRAM 的存储空间是被映射到内核的寻址区

域的，我们可以通过映射的地址直接访问 SDRAM，访问这些地址时， FMC 外设自动读写 SDRAM，程序上无需额外操作。通过地址访问内存，最直接的方式就是使用 C 语言的指针方式了。

```
#define SDRAM_BANK_ADDR                 ((uint32_t)0xC0000000)
#define SDRAM_SIZE                      (0x2000)
void SDRAM_32168Write_Read_Test(void)
{
    uint32_t counter, err_cnt;
    uint8_t u8WriteData = 0x12, u8ReadData = 0;
    uint16_t u16WriteData = 0x1234, u16ReadData = 0;
    uint32_t u32WriteData = 0x12345678, u32ReadData = 0;
    err_cnt = 0;
    /* 清零 */
    for (counter = 0; counter < SDRAM_SIZE; counter++)
    {
        *(__IO uint8_t*) (SDRAM_BANK_ADDR + counter) = (uint8_t)(0x0);
    }
    /* 向整个 SDRAM 写入数据 8 位 */
    for (counter = 0; counter < SDRAM_SIZE; counter++)
    {
        *(__IO uint8_t*) (SDRAM_BANK_ADDR + counter) = (uint8_t)(u8WriteData + counter);
    }
    /* 读取 SDRAM 数据并检测*/
    for(counter = 0; counter < SDRAM_SIZE; counter++ )
    {
        u8ReadData = *(__IO uint8_t*)(SDRAM_BANK_ADDR + counter);  //从该地址读出数据
        if(u8ReadData != (uint8_t)(u8WriteData + counter))
            err_cnt++;
    }
    HAL_DelayMs(100);
    /* 清零 */
    for (counter = 0; counter < SDRAM_SIZE; counter += 2)
    {
        *(__IO uint16_t*) (SDRAM_BANK_ADDR + counter) = (uint16_t)(0);
    }
    /* 向整个 SDRAM 写入数据 16 位 */
    for (counter = 0; counter < SDRAM_SIZE; counter += 2)
    {
```

```
        *(__IO uint16_t*) (SDRAM_BANK_ADDR + counter) = (uint16_t)(u16WriteData + counter/2);
    }
    /* 读取 SDRAM 数据并检测*/
    for(counter = 0; counter < SDRAM_SIZE; counter += 2)
    {
        u16ReadData = *(__IO uint16_t*)(SDRAM_BANK_ADDR + counter);  //从该地址读出数据
        if(u16ReadData != (uint16_t)(u16WriteData + counter/2))
            err_cnt++;
    }
    HAL_DelayMs(100);
    /* 清零 */
    for (counter = 0; counter < SDRAM_SIZE; counter += 4)
    {
        *(__IO uint32_t*) (SDRAM_BANK_ADDR + counter) = (uint32_t)(0);
    }
    /* 向整个 SDRAM 写入数据 32 位 */
    for (counter = 0; counter < SDRAM_SIZE; counter += 4)
    {
        *(__IO uint32_t*) (SDRAM_BANK_ADDR + counter) = (uint32_t)(u32WriteData + counter/4);
    }
    /* 读取 SDRAM 数据并检测*/
    for(counter = 0; counter < SDRAM_SIZE; counter += 4)
    {
        u32ReadData = *(__IO uint32_t*)(SDRAM_BANK_ADDR + counter);  //从该地址读出数据
        if(u32ReadData != (uint32_t)(u32WriteData + counter/4))
            err_cnt++;
    }
    if(err_cnt)
    {
        printfS("SDRAM_32168Write_Read_Test fail\r\n");
    }
    else
    {
        printfS("SDRAM_32168Write_Read_Test success\r\n");
    }
}
```

3.6. 直接指定变量存储到 SDRAM 空间

为了简化操作，可以直接指定变量存储到 SDRAM 空间，这种方式使用关键字“__attribute__((at()))”来指定变量的地址

```
uint8_t gu8sdram_buff[SDRAM_SIZE]          __attribute__((section(".ARM.__at_0xC0010000")));
uint16_t gu16sdram_buff[SDRAM_SIZE]        __attribute__((section(".ARM.__at_0xC0020000")));
uint32_t gu32sdram_buff[SDRAM_SIZE]        __attribute__((section(".ARM.__at_0xC0030000")));
void SDRAM_Attribute_Test(void)
{
    uint32_t counter, err_cnt;
    uint8_t u8WriteData = 0x12, u8ReadData = 0;
    err_cnt = 0;
    /* 向整个 SDRAM 写入数据 8 位 */
    for (counter = 0; counter < SDRAM_SIZE; counter++)
    {
        gu8sdram_buff[counter] = (uint8_t)(u8WriteData + counter);
    }
    /* 读取 SDRAM 数据并检测*/
    for(counter = 0; counter<SDRAM_SIZE;counter++ )
    {
        if(gu8sdram_buff[counter] != (uint8_t)(u8WriteData + counter))
            err_cnt++;
    }
    if(err_cnt)
    {
        printfS("SDRAM_Attribute_Test fail\r\n");
    }
    else
    {
        printfS("SDRAM_Attribute_Test success\r\n");
    }
}
```

4. 常见问题汇总

4.1. SDCLK 时钟分频

其它芯片 SDCLK 只有 2 个档位（1/2、1/3 HCLK）。

ACM32H5 SDCLK 档位扩充至 3 档（FMC 外设挂载在 AHB 总线上，时钟信号来自于 HCLK（默认 220M），它的时钟频率可通过 FMC_SDCR1 寄存器的 SDCLK 位配置，可以配置为 HCLK 的 1/2、1/3、1/4，也就是说 SDRAM 通讯的的同步时钟最高频率为 110MHz）。

4.2. RPIPE 读延迟

其它芯片读延迟只能通过设置 RPIPE 来决定，只有 3 个档位（0、1、2 HCLK）。

ACM32H5 在 RPIPE 基础上，新增 2 个 SDRSMA 采样微调寄存器，用于设置提前 0.5 个或 1 个 HCLK 采样，每一个 bit 对应一个 DATA 线，使读延迟档位扩充至 5 档（0、0.5、1、1.5、2 HCLK）。SDRSMA2 优先级高于 SDRSMA1。

例如：

当 RPIPE=2，SDRSMA1=0，SDRSMA2=0 时，即在 CAS 延迟后等待 2 个 HCLK 时钟才读取数据。

当 RPIPE=2，SDRSMA1=1，SDRSMA2=0 时，即在 CAS 延迟后等待 1 个 HCLK 时钟才读取数据。

当 RPIPE=2，SDRSMA1=0，SDRSMA2=1 时，即在 CAS 延迟后等待 1.5 个 HCLK 时钟才读取数据。

当 RPIPE=2，SDRSMA1=1，SDRSMA2=1 时，即在 CAS 延迟后等待 1.5 个 HCLK 时钟才读取数据。

4.3. SDRAM 读写速率说明

其它芯片读 FIFO 缓存深度 = CAS + 1 + （RPIPE / 2）。

ACM32H5 可通过 FMC_SDRCR 寄存器中的 BDEPTH 位设置读 FIFO 缓存深度。默认为 1，最大为 7。

为了加快读写速率，新增预读期间和读 FIFO 期间提前释放总线功能，可通过 FMC_SDRCR 寄存器中的 RBUSRLS、RBUSRLS2 位设置，默认开启；此功能仅在 RBURST 模式下有效。新增写期间提前一个时钟周期释放总线功能，可通过 FMC_SDRCR 寄存器中的 WBUSRLS 位设置，默认开启。

测试环境	SDRAM 读写速率
以 DMA（32bit）方式连续读写 SDRAM，数据大小为 20K Words 编译环境： ARM Compiler V6 ，优化等级-O0，关闭指令和数据加速器 SDRAM 配置： SDCLK = HCLK/2 = 110MHZ，CAS = 2，RPIPE = 1，总线宽度为 16 位，刷新计数器 = 800； TMRD = 2，TXSR = 7，TRAS = 5，TRC = 7，TWR = 2，TRP = 2，TRCD = 2。	写：571.34 us，速率 143.38 M/B 正常模式读：1137.49 us，速率 72.02 M/B 突发模式读（FIFO 长度为 1）：1328.28 us，速率 61.67 M/B 突发模式读（FIFO 长度为 2）：906.15 us，速率 90.40 M/B 突发模式读（FIFO 长度为 3）：1001.89 us，速率 81.77 M/B 突发模式读（FIFO 长度为 4）：830.43 us，速率 98.65 M/B 突发模式读（FIFO 长度为 5）：894.25 us，速率 91.61 M/B 突发模式读（FIFO 长度为 6）：786.61 us，速率 104.14 M/B 突发模式读（FIFO 长度为 7）：835.67 us，速率 98.03M/B

4.4. SDRAM 专用 IO 映射

注意 SDRAM 专用 IO 只有部分型号支持，请仔细对照数据手册。

通过设置系统控制寄存器 SYSCFG_SYSCR 中的 SDRAM_IO_SWP 位可以选择 SDRAM 专用 IO。

SDRAM	引脚名称	复用
SDRAM_D0	PQ8	AF15
SDRAM_D1	PQ9	AF15
SDRAM_D2	PM14	AF15
SDRAM_D3	PM15	AF15
SDRAM_D4	PQ10	AF15
SDRAM_D5	PQ11	AF15
SDRAM_D6	PQ12	AF15
SDRAM_D7	PQ13	AF15
SDRAM_D8	PM4	AF15
SDRAM_D9	PM5	AF15
SDRAM_D10	PM6	AF15
SDRAM_D11	PM7	AF15
SDRAM_D12	PP6	AF15
SDRAM_D13	PP7	AF15
SDRAM_D14	PP8	AF15
SDRAM_D15	PP9	AF15
SDRAM_D16	PQ0	AF15
SDRAM_D17	PQ1	AF15
SDRAM_D18	PQ2	AF15
SDRAM_D19	PQ3	AF15
SDRAM_D20	PQ4	AF15
SDRAM_D21	PQ5	AF15
SDRAM_D22	PQ6	AF15
SDRAM_D23	PQ7	AF15
SDRAM_D24	PN0	AF15
SDRAM_D25	PN1	AF15

SDRAM_D26	PN2	AF15
SDRAM_D27	PN3	AF15
SDRAM_D28	PN4	AF15
SDRAM_D29	PN5	AF15
SDRAM_D30	PN6	AF15
SDRAM_D31	PN7	AF15
SDRAM_A0	PN12	AF15
SDRAM_A1	PN13	AF15
SDRAM_A2	PN14	AF15
SDRAM_A3	PN15	AF15
SDRAM_A4	PO6	AF15
SDRAM_A5	PO7	AF15
SDRAM_A6	PO8	AF15
SDRAM_A7	PO9	AF15
SDRAM_A8	PO10	AF15
SDRAM_A9	PO11	AF15
SDRAM_A10	PL9	AF15
SDRAM_A11	PL10	AF15
SDRAM_A12	PL11	AF15
SDRAM_A14	PL7	AF15
SDRAM_A15	PL8	AF15
SDRAM_SDCLK	PL13	AF15
SDRAM_SDCKE0	PL12	AF15
SDRAM_SDNE0	PL6	AF15
SDRAM_NCAS	PL4	AF15
SDRAM_NRAS	PL5	AF15
SDRAM_SDNWE	PO5	AF15
SDRAM_NBL0	PO4	AF15
SDRAM_NBL1	PL14	AF15
SDRAM_NBL2	PQ14	AF15

SDRAM_NBL3	PQ15	AF15

5. 版本历史

版本	日期	作者	描述
V1.0	2024-11-07	Aisinochip	初始版
V1.1	2024-12-11	Aisinochip	修改地址映射框图，增加 External RAM 区和 External device 区说明

版权声明

本文档的所有部分，其著作产权归上海航芯电子科技股份有限公司（简称航芯科技）所有，未经航芯科技授权许可，任何个人及组织不得复制、转载、仿制本文档的全部或部分组件。本文档没有任何形式的担保、立场表达或其他暗示，若有任何因本文档或其中提及的产品所有资讯所引起的直接或间接损失，航芯科技及所属员工恕不为其担保任何责任。除此以外，本文档所提到的产品规格及资讯仅供参考，内容亦会随时更新，恕不另行通知。

联系我们

公司：上海航芯电子科技股份有限公司

地址：上海市闵行区合川路 2570 号科技绿洲三期 2 号楼 702 室

邮编：200241

电话：+86-21-6125 9080

传真：+86-21-6125 9080-830

Email： service@AisinoChip.com

Website： www.AisinoChip.com