第27章　FSMC—扩展外部 SRAM

本章参考资料：《STM32F10X-中文参考手册》FSMC 章节。

关于 SRAM 存储器，请参考“常用存储器介绍”章节，实验中 FLASH 芯片的具体参数，请参考其规格书《IS62WV51216》来了解。

27.1 SRAM 控制原理

STM32 控制器芯片内部有一定大小的 SRAM 及 FLASH 作为内存和程序存储空间，但当程序较大，内存和程序空间不足时，就需要在 STM32 芯片的外部扩展存储器了。

扩展内存时一般使用 SRAM 和 SDRAM 存储器，但 STM32F1 系列的芯片不支持扩展 SDRAM(STM32F429 系列支持)，它仅支持使用 FSMC 外设扩展 SRAM，我们以 SRAM 为例讲解如何为 STM32 扩展内存。由于引脚数量的限制，只有 STM32F103ZE 或以上型号的芯片才可以扩展外部 SRAM。

给 STM32 芯片扩展内存与给 PC 扩展内存的原理是一样的，只是 PC 上一般以内存条的形式扩展，内存条实质是由多个内存颗粒(即 SDRAM 芯片)组成的通用标准模块，而 STM32 直接与 SRAM 芯片连接。见图 27-2，这是我们实验板上使用的型号为 IS62WV51216 的 SRAM 芯片内部结构框图，以它为模型进行学习。

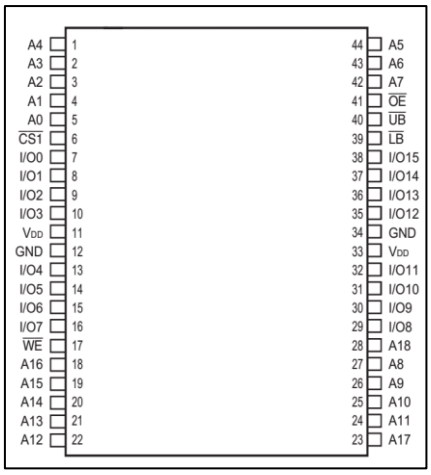

图 27-1 SRAM 芯片外观

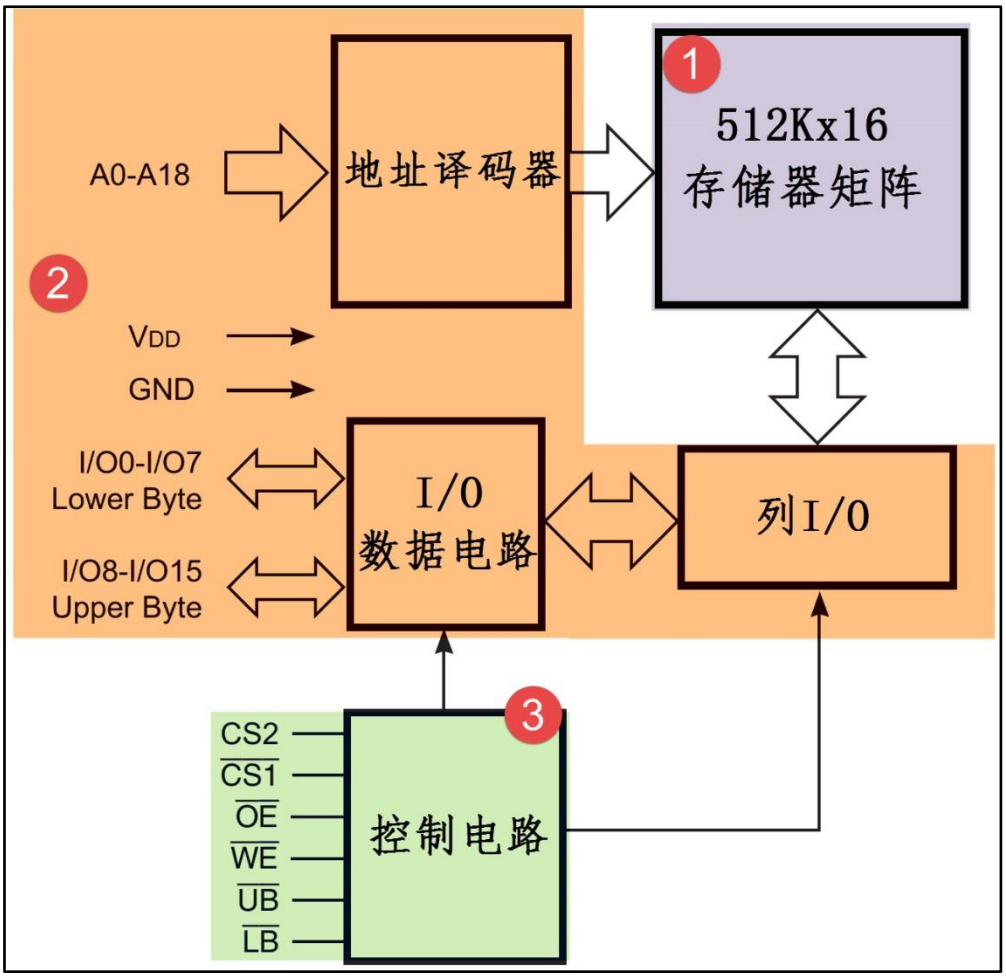

图 27-2 一种 SRAM 芯片的内部结构框图

27.1.1 SRAM 信号线

图 27-2 中左侧引出的是 SRAM 芯片的控制引脚，其说明见表 27-1。

表 27-1 SRAM 控制引脚说明

信号线	类型	说明
A0-A18	I	地址输入
I/O0-I/O7	I/O	数据输入输出信号，低字节
I/O8-I/O15	I/O	数据输入输出信号，高字节
CS 和 CS1#	I	片选信号，CS2 高电平有效，CS1#低电平有效，部分芯片只有其中一个引脚
OE#	I	输出使能信号，低电平有效
WE#	I	写入使能，低电平有效
UB#	I	数据掩码信号 Upper Byte，高位字节允许访问，低电平有效
LB#	I	数据掩码信号 Lower Byte，低位字节允许访问，低电平有效

SRAM 的控制比较简单，只要控制信号线使能了访问，从地址线输入要访问的地址，即可从 I/O 数据线写入或读出数据。

27.1.2 存储器矩阵

框图中标号①处表示的是存储器矩阵，这个 SRAM 芯片的空间大小为 512Kx16(bits)，见图 27-3。

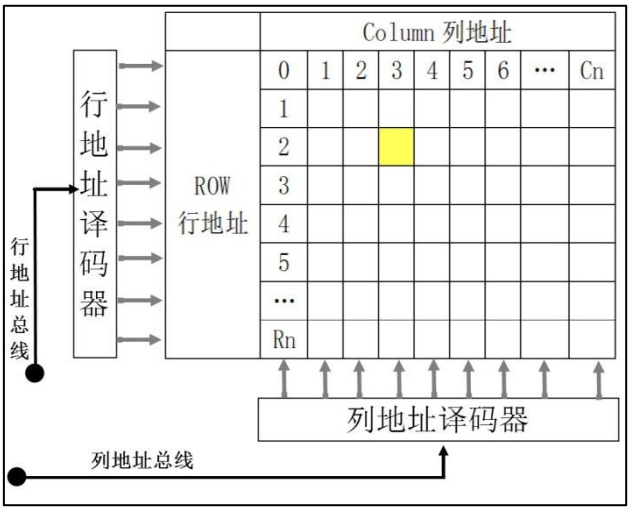

图 27-3 SRAM 存储阵列模型

SRAM 内部包含的存储阵列，可以把它理解成一张表格，数据就填在这张表格上。和表格查找一样，指定一个行地址和列地址，就可以精确地找到目标单元格，这是 SRAM 芯片寻址的基本原理。这样的每个单元格被称为存储单元，而这样的表则被称为存储矩阵。

27.1.3 地址译码器、列 I/O 及 I/O 数据电路

地址译码器把 N 根地址线转换成 2^N 根信号线，每根信号线对应一行或一列存储单元，通过地址线找到具体的存储单元，实现寻址。如果存储阵列比较大，地址线会分成行和列地址，或者行、列分时复用同一地址总线，访问数据寻址时先用地址线传输行地址再传输列地址。本实例中的 SRAM 比较小，没有列地址线，它的数据宽度为 16 位，即一个行地址对应 2 字节空间，框图中左侧的 A0-A18 是行址信号，18 根地址线一共可以表示 $2^{18}=2^8$x1024=512K 行存储单元，所以它一共能访问 512Kx16bits 大小的空间。访问时，使用 UB#或 LB#线控制数据宽度，例如，当要访问宽度为 16 位的数据时，使用行地址线指出地址，然后把 UB#和 LB#线都设置为低电平，那么 I/O0-I/O15 线都有效，它们一起输出该地址的 16 位数据(或者接收 16 位数据到该地址)；当要访问宽度为 8 位的数据时，使用行地址线指出地址，然后把 UB#或 LB#其中一个设置为低电平，I/O 会对应输出该地址的高 8 位和低 8 位数据，因此它们被称为数据掩码信号。

27.1.4 控制电路

控制电路主要包含了片选、读写使能以及上面提到的宽度控制信号 UB#和 LB#。利用 CS2 或 CS1#片选信号，可以把多个 SRAM 芯片组成一个大容量的内存条。OE#和 WE#可以控制读写使能，防止误操作。

27.1.5 SRAM 的读写流程

对 SRAM 进行读写数据时，它各个信号线的时序流程见图 27-4 及图 27-5。

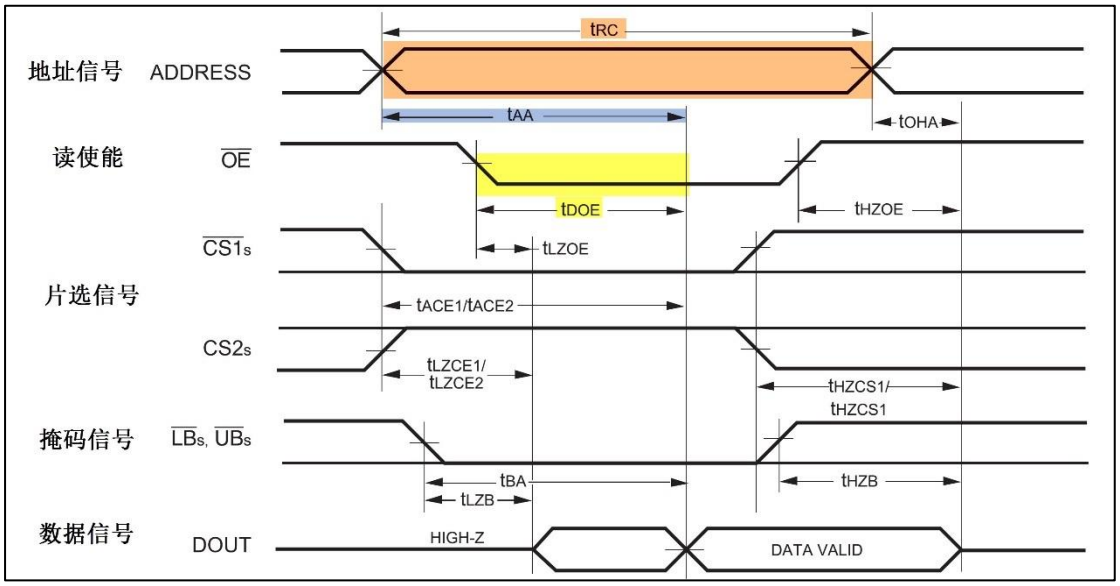

图 27-4 SRAM 的读时序

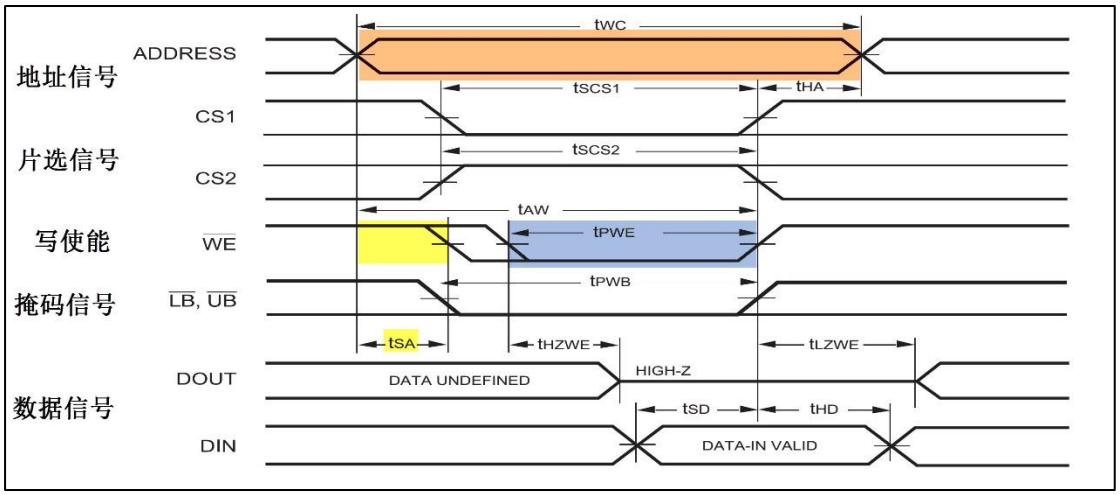

图 27-5 SRAM 的写时序

读写时序的流程很类似，下面我们统一解说：

(1) 主机使用地址信号线发出要访问的存储器目标地址；

(2) 控制片选信号 CS1#及 CS2#使能存储器芯片；

(3) 若是要进行读操作，则控制读使能信号 OE#表示要读数据，若进行写操作则控制写使能信号 WE#表示要写数据；

(4) 使用掩码信号 LB#与 UB#指示要访问目标地址的高、低字节部分；

(5) 若是读取过程，存储器会通过数据线向主机输出目标数据，若是写入过程，主要使用数据线向存储器传输目标数据。

在读写时序中，有几个比较重要的时间参数，在使用 STM32 控制的时候需要参考，它们的介绍见表 27-2。

表 27-2 IS62WV51216BLL-55ns 型号 SRAM 的时间参数

时间参数	IS62WV51216BLL-55ns 型号的时间要求	说明
t_{RC}	不小于 55ns	读操作的总时间
t_{AA}	最迟不大于 55ns	从接收到地址信号到给出有效数据的时间
t_{DOE}	最迟不大于 25ns	从接收到读使能信号到给出有效数据的时间
t_{WC}	不小于 55ns	写操作的总时间
t_{SA}	大于 0ns	从发送地址信号到给出写有使能信号的时间
t_{PWE}	不小于 40ns	从接收到写使能信号到数据采样的时间

27.2 FSMC 简介

STM32F1 系列芯片使用 FSMC 外设来管理扩展的存储器，FSMC 是 Flexible Static Memory Controller 的缩写，译为灵活的静态存储控制器。它可以用于驱动包括 SRAM、NOR FLASH 以及 NAND FLSAH 类型的存储器，不能驱动如 SDRAM 这种动态的存储器而在 STM32F429 系列的控制器中，它具有 FMC 外设，支持控制 SDRAM 存储器。

27.3 FSMC 框图剖析

STM32 的 FSMC 外设内部结构见图 27-6。

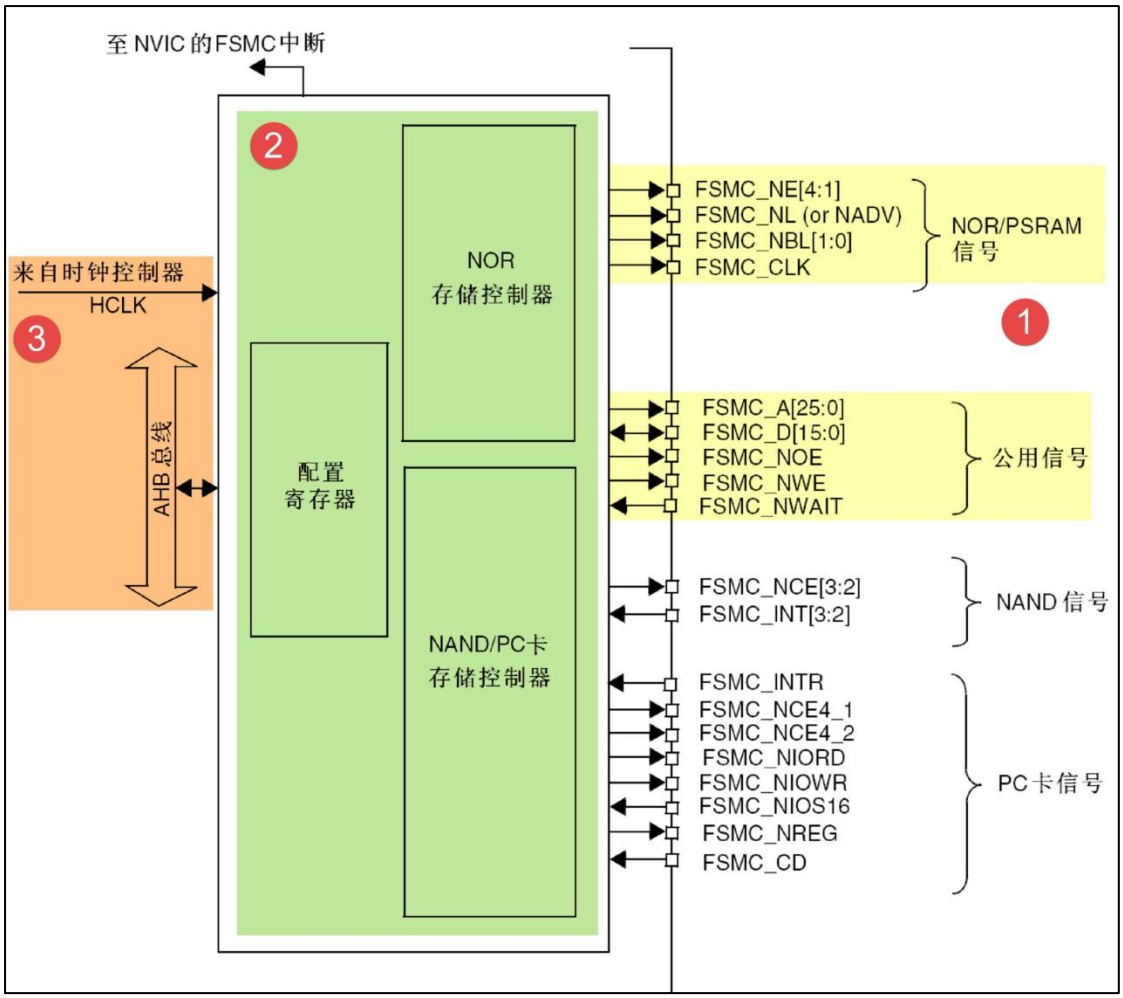

图 27-6 FSMC 控制器框图

1. 通讯引脚

在框图的右侧是 FSMC 外设相关的控制引脚，由于控制不同类型存储器的时候会有一些不同的引脚，看起来有非常多，其中地址线 FSMC_A 和数据线 FSMC_D 是所有控制器都共用的。这些 FSMC 引脚具体对应的 GPIO 端口及引脚号可在《STM32F103 规格书》中搜索查找到，不在此列出。针对本示例中的 SRAM 控制器，我们整理出以下的 FSMC 与 SRAM 引脚对照表 27-3。

表 27-3 FSMC 中的 SRAM 控制信号线

FSMC 引脚名称	对应 SRAM 引脚名	说明
FSMC_NBL[1:0]	LB#、UB#	数据掩码信号
FSMC_A[18:0]	A[18:0]	行地址线
FSMC_D[15:0]	I/O[15:0]	数据线
FSMC_NWE	WE#	写入使能
FSMC_NOE	OE#	输出使能(读使能)
FSMC_NE[1:4]	CE#	片选信号

注：由于大部分地址线 FSMC_A 都是使用到了 GPIOF、GPIOG 端口，而 STM32VET6 等 144 脚以下型号的芯片不具有这些端口，因此对于 144 脚以下的 STM32 芯片，无法扩展

外部 SRAM，STM32ZET6 或以上型号的芯片具有 GPIOF、GPIOG 等端口，因此想要扩展外部 SRAM 的时候，需要注意 STM32 芯片的选型。

其中比较特殊的 FSMC_NE 是用于控制 SRAM 芯片的片选控制信号线，STM32 具有 FSMC_NE1/2/3/4 号引脚，不同的引脚对应 STM32 内部不同的地址区域。例如，当 STM32 访问 0x68000000-0x6BFFFFFF 地址空间时，FSMC_NE3 引脚会自动设置为低电平，由于它连接到 SRAM 的 CE#引脚，所以 SRAM 的片选被使能，而访问 0x60000000-0x63FFFFFF 地址时，FSMC_NE1 会输出低电平。当使用不同的 FSMC_NE 引脚连接外部存储器时，STM32 访问 SRAM 的地址不一样，从而达到控制多块 SRAM 芯片的目的。各引脚对应的地址会在后面“FSMC 的地址映射”小节讲解。

2. 存储器控制器

上面不同类型的引脚是连接到 FSMC 内部对应的存储控制器中的。NOR/PSRAM/SRAM 设备使用相同的控制器，NAND/PC 卡设备使用相同的控制器，不同的控制器有专用的寄存器用于配置其工作模式。

控制 SRAM 的有 FSMC_BCR1/2/3/4 控制寄存器、FSMC_BTR1/2/3/4 片选时序寄存器以及 FSMC_BWTR1/2/3/4 写时序寄存器。每种寄存器都有 4 个，分别对应于 4 个不同的存储区域，各种寄存器介绍如下：

- FSMC_BCR 控制寄存器可配置要控制的存储器类型、数据线宽度以及信号有效极性能参数。
- FMC_BTR 时序寄存器用于配置 SRAM 访问时的各种时间延迟，如数据保持时间、地址保持时间等。
- FMC_BWTR 写时序寄存器与 FMC_BTR 寄存器控制的参数类似，它专门用于控制写时序的时间参数。

3. 时钟控制逻辑

FSMC 外设挂载在 AHB 总线上，时钟信号来自于 HCLK(默认 72MHz)，控制器的同步时钟输出就是由它分频得到。例如，NOR 控制器的 FSMC_CLK 引脚输出的时钟，它可用于与同步类型的 SRAM 芯片进行同步通讯，它的时钟频率可通过 FSMC_BTR 寄存器的 CLKDIV 位配置，可以配置为 HCLK 的 1/2 或 1/3，也就是说，若它与同步类型的 SRAM 通讯时，同步时钟最高频率为 36MHz。本示例中的 SRAM 为异步类型的存储器，不使用同步时钟信号，所以时钟分频配置不起作用。

27.4 FSMC 的地址映射

FSMC 连接好外部的存储器并初始化后，就可以直接通过访问地址来读写数据，这种地址访问与 I2C EEPROM、SPI FLASH 的不一样，后两种方式都需要控制 I2C 或 SPI 总线给存储器发送地址，然后获取数据；在程序里，这个地址和数据都需要分开使用不同的变量存储，并且访问时还需要使用代码控制发送读写命令。而使用 FSMC 外接存储器时，其存储单元是映射到 STM32 的内部寻址空间的；在程序里，定义一个指向这些地址的指针，

然后就可以通过指针直接修改该存储单元的内容，FSMC 外设会自动完成数据访问过程，读写命令之类的操作不需要程序控制。FSMC 的地址映射见图 27-7。

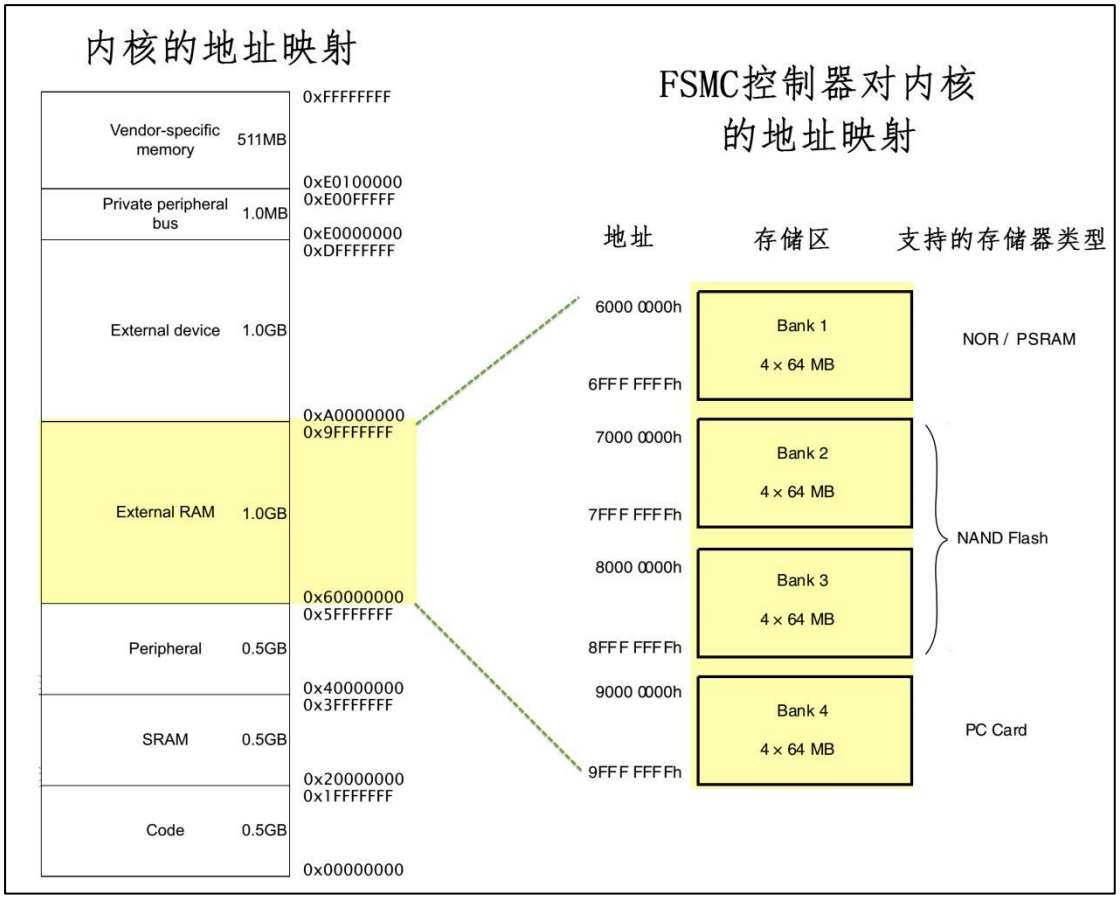

图 27-7 FSMC 的地址映射

图中左侧的是 Cortex-M3 内核的存储空间分配，右侧是 STM32 FSMC 外设的地址映射。可以看到 FSMC 的 NOR/PSRAM/SRAM/NAND FLASH 以及 PC 卡的地址都在 External RAM 地址空间内。正是因为存在这样的地址映射，使得访问 FSMC 控制的存储器时，就跟访问 STM32 的片上外设寄存器一样(片上外设的地址映射即图中左侧的“Peripheral”区域)。

FSMC 把整个 External RAM 存储区域分成了 4 个 Bank 区域，并分配了地址范围及适用的存储器类型，如 NOR 及 SRAM 存储器只能使用 Bank1 的地址。在每个 Bank 的内部又分成了 4 个小块，每个小块有相应的控制引脚用于连接片选信号，如 FSMC_NE[4:1]信号线可用于选择 BANK1 内部的 4 小块地址区域，见图 27-8，当 STM32 访问 0x68000000-0x6BFFFFFF 地址空间时，会访问到 Bank1 的第 3 小块区域，相应的 FSMC_NE3 信号线会输出控制信号。

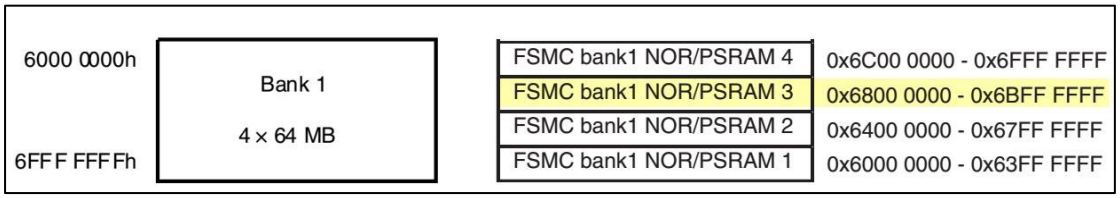

图 27-8 Bank1 内部的小块地址分配

27.5 FSMC 控制 SRAM 的时序

FSMC 外设支持输出多种不同的时序以便于控制不同的存储器，它具有 ABCD 四种模式，下面我们仅针对控制 SRAM 使用的模式 A 进行讲解，见图 27-9 及图 27-10。

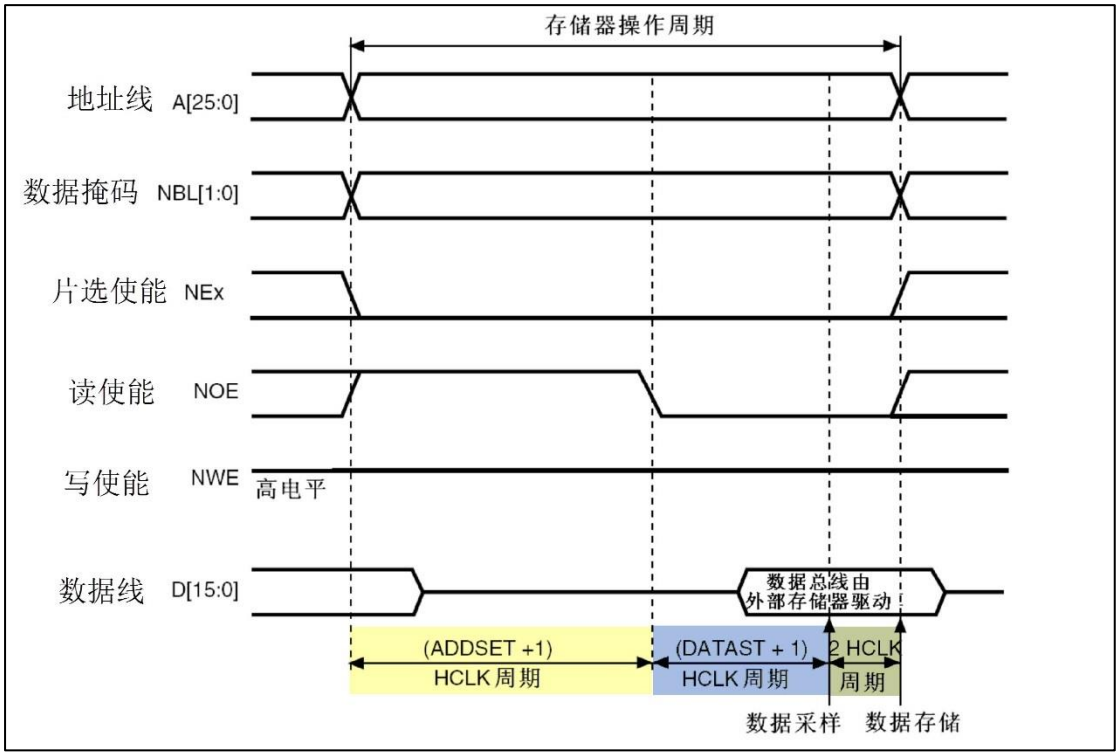

图 27-9 FSMC 模式 A 的读时序

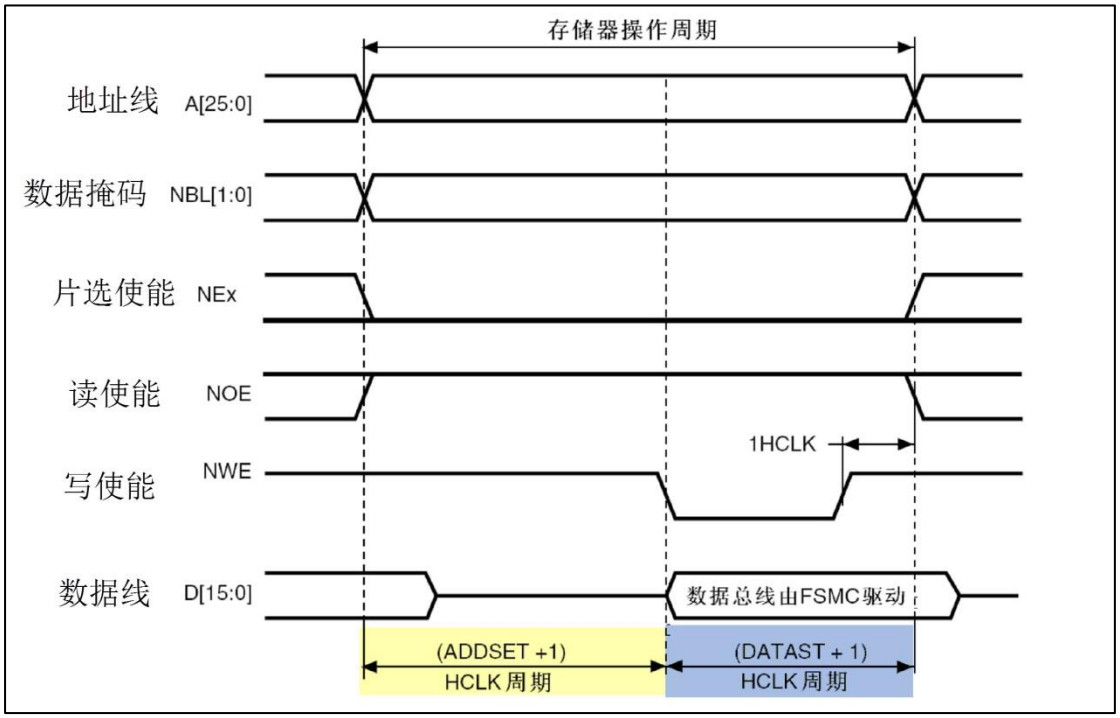

图 27-10 FSMC 模式 A 的写时序

当内核发出访问某个指向外部存储器地址时，FSMC 外设会根据配置控制信号线产生时序访问存储器，上图中的是访问外部 SRAM 时 FSMC 外设的读写时序。

以读时序为例，该图表示一个存储器操作周期由地址建立周期(ADDSET)、数据建立周期(DATAST)以及 2 个 HCLK 周期组成。在地址建立周期中，地址线发出要访问的地址，数据掩码信号线指示出要读取地址的高、低字节部分，片选信号使能存储器芯片；地址建立周期结束后读使能信号线发出读使能信号，接着存储器通过数据信号线把目标数据传输给 FSMC，FSMC 把它交给内核。

写时序类似，区别是它的一个存储器操作周期仅由地址建立周期(ADDSET)和数据建立周期(DATAST)组成，且在数据建立周期期间写使能信号线发出写信号，接着 FSMC 把数据通过数据线传输到存储器中。

27.6 SRAM 时序结构体

控制 FSMC 使用 SRAM 存储器时主要是配置时序寄存器以及控制寄存器，利用 ST 标准库的 SRAM 时序结构体以及初始化结构体可以很方便地写入参数。

SRAM 时序结构体的成员见代码清单 27-1。

代码清单 27-1 SRAM 时序结构体 FSMC_NORSRAMTimingInitTypeDef

```
1 typedef struct
2 {
3   uint32_t FSMC_AddressSetupTime;        /*地址建立时间，0-0xF 个 HCLK 周期*/
4   uint32_t FSMC_AddressHoldTime;         /*地址保持时间，0-0xF 个 HCLK 周期*/
5   uint32_t FSMC_DataSetupTime;           /*地址建立时间，0-0xF 个 HCLK 周期*/
6   uint32_t FSMC_BusTurnAroundDuration;/*总线转换周期,0-0xF 个 HCLK 周期，在 NOR FLASH */
7  uint32_t FSMC_CLKDivision;/*时钟分频因子,1-0xF，若控制异步存储器，本参数无效 */
8   uint32_t FSMC_DataLatency;     /*数据延迟时间，若控制异步存储器，本参数无效 */
9   uint32_t FSMC_AccessMode;              /*设置访问模式 */
10 }FSMC_NORSRAMTimingInitTypeDef;
```

这个结构体成员定义的都是 SRAM 读写时序中的各项时间参数，这些成员的的参数都与 FSMC_BRT 及 FSMC_BWTR 寄存器配置对应，各个成员介绍如下：

(1) FSMC_AddressSetupTime

本成员设置地址建立时间，即 FSMC 读写时序图 27-9 中的 ADDSET 值，它可以被设置为 0-0xF 个 HCLK 周期数，按 STM32 标准库的默认配置，HCLK 的时钟频率为 72MHz，即一个 HCLK 周期为 1/72 微秒。

(2) FSMC_AddressHoldTime

本成员设置地址保持时间，它可以被设置为 0-0xF 个 HCLK 周期数。

(3) FSMC_DataSetupTime

本成员设置数据建立时间，即 FSMC 读写时序图 27-10 中的 DATAST 值，它可以被设置为 0-0xF 个 HCLK 周期数。

(4) FSMC_BusTurnAroundDuration

本成员设置总线转换周期，在 NOR FLASH 存储器中，地址线与数据线可以分时复用，总线转换周期就是指总线在这两种状态间切换需要的延时，防止冲突。控制其它存储器时这个参数无效，配置为 0 即可。

(5) FSMC_CLKDivision

本成员用于设置时钟分频，它以 HCLK 时钟作为输入，经过 FSMC_CLKDivision 分频后输出到 FSMC_CLK 引脚作为通讯使用的同步时钟。控制其它异步通讯的存储器时这个参数无效，配置为 0 即可。

(6) FSMC_DataLatency

本成员设置数据保持时间，它表示在读取第一个数据之前要等待的周期数，该周期指同步时钟的周期，本参数仅用于同步 NOR FLASH 类型的存储器，控制其它类型的存储器时，本参数无效。

(7) FSMC_AccessMode

本成员设置存储器访问模式，不同的模式下 FSMC 访问存储器地址时引脚输出的时序不一样，可选 FSMC_AccessMode_A/B/C/D 模式。一般来说控制 SRAM 时使用 A 模式。

这个 FSMC_NORSRAMTimingInitTypeDef 时序结构体配置的延时参数，将作为下一节的 FSMC SRAM 初始化结构体的一个成员。

27.7 SRAM 初始化结构体

FSMC 的 SRAM 初始化结构体见代码清单 27-2。

代码清单 27-2 SRAM 初始化结构体 FSMC_NORSRAMInitTypeDef

```
/**
  * @brief  FSMC NOR/SRAM Init structure definition
  */
typedef struct
{
  uint32_t FSMC_Bank;                  /*设置要控制的 Bank 区域 */
  uint32_t FSMC_DataAddressMux;        /*设置地址总线与数据总线是否复用 */
  uint32_t FSMC_MemoryType;            /*设置存储器的类型 */
  uint32_t FSMC_MemoryDataWidth;       /*设置存储器的数据宽度*/
  uint32_t FSMC_BurstAccessMode; /*设置是否支持突发访问模式，只支持同步类型的存储器 */
  uint32_t FSMC_AsynchronousWait;      /*设置是否使能在同步传输时的等待信号，*/
  uint32_t FSMC_WaitSignalPolarity;   /*设置等待信号的极性*/
  uint32_t FSMC_WrapMode;              /*设置是否支持对齐的突发模式 */
  uint32_t FSMC_WaitSignalActive; /*配置等待信号在等待前有效还是等待期间有效 */
  uint32_t FSMC_WriteOperation;        /*设置是否写使能 */
  uint32_t FSMC_WaitSignal;            /*设置是否使能等待状态插入 */
  uint32_t FSMC_ExtendedMode;          /*设置是否使能扩展模式 */
  uint32_t FSMC_WriteBurst;            /*设置是否使能写突发操作*/
   /*当不使用扩展模式时，本参数用于配置读写时序，否则用于配置读时序*/
  FSMC_NORSRAMTimingInitTypeDef* FSMC_ReadWriteTimingStruct;
   /*当使用扩展模式时，本参数用于配置写时序*/
  FSMC_NORSRAMTimingInitTypeDef* FSMC_WriteTimingStruct;
}FSMC_NORSRAMInitTypeDef;
```

这个结构体，除最后两个成员是上一小节讲解的时序配置外，其它结构体成员的配置都对应到 FSMC_BCR 中的寄存器位。各个成员意义介绍如下，括号中的是 STM32 标准库定义的宏：

(1) FSMC_Bank

本成员用于选择 FSMC 映射的存储区域，它的可选参数以及相应的内核地址映射范围见表 27-4。

表 27-4 可以选择的存储器区域及区域对应的地址范围

可以输入的宏	对应的地址区域
FSMC_Bank1_NORSRAM1	0x60000000-0x63FFFFFF
FSMC_Bank1_NORSRAM2	0x64000000-0x67FFFFFF
FSMC_Bank1_NORSRAM3	0x68000000-0x6BFFFFFF
FSMC_Bank1_NORSRAM4	0x6C000000-0x6FFFFFFF

(2) FSMC_DataAddressMux

本成员用于设置地址总线与数据总线是否复用(FSMC_DataAddressMux_Enable /Disable)，在控制 NOR FLASH 时，可以地址总线与数据总线可以分时复用，以减少使用 STM32 信号线的数量。

(3) FSMC_MemoryType

本成员用于设置要控制的存储器类型，它支持控制的存储器类型为 SRAM、PSRAM 以及 NOR FLASH(FSMC_MemoryType_SRAM/PSRAM/NOR)。

(4) FSMC_MemoryDataWidth

本成员用于设置要控制的存储器的数据宽度，可选择设置成 8 或 16 位(FSMC_MemoryDataWidth_8b /16b)。

(5) FSMC_BurstAccessMode

本成员用于设置是否使用突发访问模式(FSMC_BurstAccessMode_Enable/Disable)，突发访问模式是指发送一个地址后连续访问多个数据，非突发模式下每访问一个数据都需要输入一个地址，仅在控制同步类型的存储器时才能使用突发模式。

(6) FSMC_AsynchronousWait

本成员用于设置是否使能在同步传输时使用的等待信号(FSMC_AsynchronousWait_Enable/Disable)，在控制同步类型的 NOR 或 PSRAM 时，存储器可以使用 FSMC_NWAIT 引脚通知 STM32 需要等待。

(7) FSMC_WaitSignalPolarity

本成员用于设置等待信号的有效极性，即要求等待时，使用高电平还是低电平(FSMC_WaitSignalPolarity_High/Low)。

(8) FSMC_WrapMode

本成员用于设置是否支持把非对齐的 AHB 突发操作分割成 2 次线性操作(FSMC_WrapMode_Enable/Disable)，该配置仅在突发模式下有效。

(9) FSMC_WaitSignalActive

本成员用于配置在突发传输模式时，决定存储器是在等待状态之前的一个数据周期有效还是在等待状态期间有效(FSMC_WaitSignalActive_BeforeWaitState/DuringWaitState)。

(10) FSMC_WriteOperation

这个成员用于设置是否写使能(FSMC_WriteOperation_ Enable /Disable)，禁止写使能的话 FSMC 只能从存储器中读取数据，不能写入。

(11) FSMC_WaitSignal

本成员用于设置当存储器处于突发传输模式时，是否允许通过 NWAIT 信号插入等待状态(FSMC_WaitSignal_Enable/Disable)。

(12) FSMC_ExtendedMode

本成员用于设置是否使用扩展模式(FSMC_ExtendedMode_Enable/Disable)，在非扩展模式下，对存储器读写的时序都只使用 FSMC_BCR 寄存器中的配置，即下面的 FSMC_ReadWriteTimingStruct 结构体成员；在扩展模式下，对存储器的读写时序可以分开配置，读时序使用 FSMC_BCR 寄存器，写时序使用 FSMC_BWTR 寄存器的配置，即下面的 FSMC_WriteTimingStruct 结构体。

(13) FSMC_ReadWriteTimingStruct

本成员是一个指针，赋值时使用上一小节中讲解的时序结构体 FSMC_NORSRAMInitTypeDef 设置，当不使用扩展模式时，读写时序都使用本成员的参数配置。

(14) FSMC_WriteTimingStruct

同样地，本成员也是一个时序结构体的指针，只有当使用扩展模式时，本配置才有效，它是写操作使用的时序。

对本结构体赋值完成后，调用 FSMC_NORSRAMInit 库函数即可把配置参数写入到 FSMC_BCR 及 FSMC_BTR/BWTR 寄存器中。

27.8 FSMC—扩展外部 SRAM 实验

本小节以型号为“IS62WV51216”的 SRAM 芯片为 STM32 扩展内存。它的地址线宽度为 19 位，数据线宽度为 16 位，容量大小为 1MB。

学习本小节内容时，请打开配套的“FSMC—外部 SRAM”工程配合阅读。本实验仅讲解基本的外部 SRAM 驱动，不涉及内存管理的内容，在本书的《MDK 编译过程及文件类型全解》章节将会讲解使用更简单的方法从外部 SRAM 中分配变量，以及使用 C 语言标准库的 malloc 函数来分配外部 SRAM 的空间。

27.8.1 硬件设计

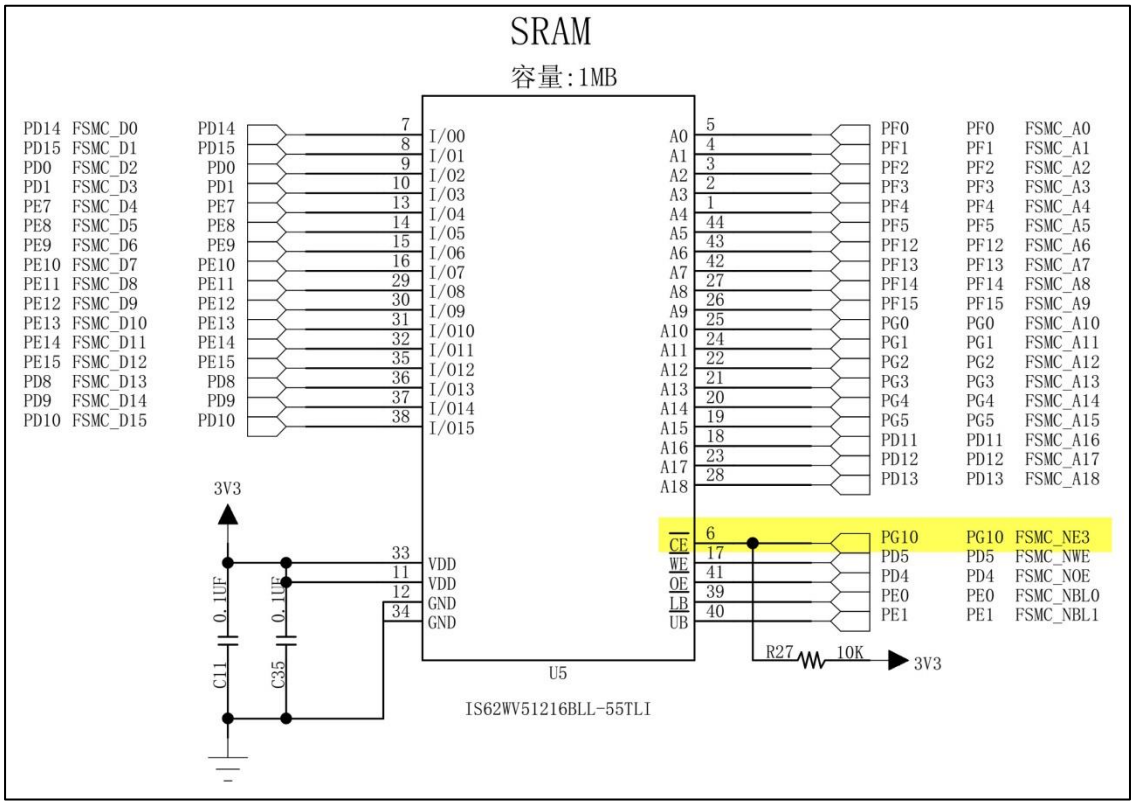

图 27-11 外部 SRAM 硬件连接图

外部 SRAM 芯片与 STM32 相连的引脚非常多，主要是地址线和数据线，这些具有特定 FSMC 功能的 GPIO 引脚可查询《STM32F10x 规格书》中的说明来了解。

关于该 SRAM 芯片的更多信息，请参考其规格书《IS62WV51216》了解。若您使用的实验板 SRAM 的型号或控制引脚不一样，可在我们工程的基础上修改，程序的控制原理相同。

根据本硬件设计，SRAM 芯片的使能信号与 FSMC_NE3 连接，所以它会被映射到 STM32 中的 BANK1 NOR/SRAM 3 区域，该区域的地址范围为 0x68000000-0x6BFFFFFF，因此，当内核访问从基地址 0x68000000 开始的 1MB 空间时，FSMC 外设会自动控制原理图中的引脚产生访问时序，访问这个外部 SRAM 存储器。

27.8.2 软件设计

为了使工程更加有条理，我们把 SRAM 初始化相关的代码独立分开存储，方便以后移植。在“工程模板”之上新建“sram.c”及“sram.h”文件，这些文件也可根据您的喜好命名，它们不属于 STM32 标准库的内容，是由我们自己根据应用需要编写的。

1. 编程要点

(1) 初始化通讯使用的目标引脚及端口时钟；

(2) 使能 FSMC 外设的时钟；

(3) 配置 FSMC SRAM 的时序、工作模式；

(4) 建立机制访问外部 SRAM 存储器；

(5) 编写测试程序，对读写数据进行校验。

2. 代码分析

FSMC 硬件相关宏定义

我们把 FSMC SRAM 硬件相关的配置都以宏的形式定义到 “sram.h”文件中，见代码清单 27-3。

代码清单 27-3 SRAM 硬件配置相关的宏(省略了大部分数据线，sram.h 文件)

```
/*A 地址信号线*/
#define FSMC_A0_GPIO_PORT         GPIOF
#define FSMC_A0_GPIO_CLK          RCC_APB2Periph_GPIOF
#define FSMC_A0_GPIO_PIN          GPIO_Pin_0
/*……省略 A1-A18 地址线*/

/*D 数据信号线*/
#define FSMC_D0_GPIO_PORT         GPIOD
#define FSMC_D0_GPIO_CLK          RCC_APB2Periph_GPIOD
#define FSMC_D0_GPIO_PIN          GPIO_Pin_14
/*……省略 D1-D15 地址线*/

/*控制信号线*/
/*CS 片选*/
/*NE3 ,对应的基地址 0x68000000*/
#define FSMC_CS_GPIO_PORT         GPIOG
#define FSMC_CS_GPIO_CLK          RCC_APB2Periph_GPIOG
#define FSMC_CS_GPIO_PIN          GPIO_Pin_10
/*WE 写使能*/
#define FSMC_WE_GPIO_PORT         GPIOD
#define FSMC_WE_GPIO_CLK          RCC_APB2Periph_GPIOD
#define FSMC_WE_GPIO_PIN          GPIO_Pin_5
/*OE 读使能*/
#define FSMC_OE_GPIO_PORT         GPIOD
#define FSMC_OE_GPIO_CLK          RCC_APB2Periph_GPIOD
#define FSMC_OE_GPIO_PIN          GPIO_Pin_4
/*LB 数据掩码*/
#define FSMC_UDQM_GPIO_PORT         GPIOE
#define FSMC_UDQM_GPIO_CLK          RCC_APB2Periph_GPIOE
#define FSMC_UDQM_GPIO_PIN          GPIO_Pin_1
/*UB 数据掩码*/
#define FSMC_LDQM_GPIO_PORT         GPIOE
#define FSMC_LDQM_GPIO_CLK          RCC_APB2Periph_GPIOE
#define FSMC_LDQM_GPIO_PIN          GPIO_Pin_0
```

以上代码根据硬件的连接，把与 SRAM 通讯使用的引脚端口、引脚号以及时钟都以宏封装起来。其中 FSMC_CS 作为片选引脚对应的是 FSMC_NE3，所以后面我们对 SDRAM 的寻址空间也是要指向存储区域 BANK1 NOR/SRAM 3 的。

初始化 FSMC 的 GPIO

利用上面的宏，编写 FSMC 的 GPIO 引脚初始化函数，见代码清单 27-4。

代码清单 27-4 FSMC 的 GPIO 初始化函数(省略了大部分数据线，sram.c 文件)

```
/**
  * @brief   初始化控制 SRAM 的 IO
```

```
  * @param  无
  * @retval 无
  */
static void SRAM_GPIO_Config(void)
{
  GPIO_InitTypeDef  GPIO_InitStructure;

  /* 使能 SRAM 相关的 GPIO 时钟 */

                          /*地址信号线,部分省略*/
  RCC_APB2PeriphClockCmd(FSMC_A0_GPIO_CLK |
                          /*数据信号线，部分省略*/
                          FSMC_D0_GPIO_CLK |
                          /*控制信号线*/
                FSMC_CS_GPIO_CLK  | FSMC_WE_GPIO_CLK | FSMC_OE_GPIO_CLK |
                          FSMC_UDQM_GPIO_CLK|FSMC_LDQM_GPIO_CLK, ENABLE);

  /*使能 FSMC 外设时钟*/
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC,ENABLE);

 /*-- GPIO 配置 -----------------------------------------------------*/
/*--所有 GPIO 的配置都相同，此处省略大量引脚初始化，具体请查看工程中的代码*/
  /* 通用 GPIO 配置 */
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;        //配置为复用功能
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

  /*A 地址信号线 针对引脚配置*/
  GPIO_InitStructure.GPIO_Pin = FSMC_A0_GPIO_PIN;
  GPIO_Init(FSMC_A0_GPIO_PORT, &GPIO_InitStructure);
      /*...*/
  /*DQ 数据信号线 针对引脚配置*/
  GPIO_InitStructure.GPIO_Pin = FSMC_D0_GPIO_PIN;
  GPIO_Init(FSMC_D0_GPIO_PORT, &GPIO_InitStructure);
     /*...*/
  /*控制信号线*/
  GPIO_InitStructure.GPIO_Pin = FSMC_CS_GPIO_PIN;
  GPIO_Init(FSMC_CS_GPIO_PORT, &GPIO_InitStructure);
    /*...*/
}
```

与所有使用到 GPIO 的外设一样，都要先把使用到的 GPIO 引脚模式初始化，以上代码把 FSMC SRAM 的所有信号线全都初始化为复用功能，所有引脚配置都是一样的。

配置 FSMC 的模式

接下来需要配置 FSMC SRAM 的工作模式，这个函数的主体是根据硬件连接的 SRAM 特性，对时序结构体以及初始化结构体进行赋值。见代码清单 27-5。

代码清单 27-5 配置 FSMC 的模式(sram.c 文件)

```
/**
  * @brief  初始化 FSMC 外设
  * @param  None.
  * @retval None.
  */
void FSMC_SRAM_Init(void)
{
  FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
  FSMC_NORSRAMTimingInitTypeDef  readWriteTiming;

  /*初始化 SRAM 相关的 GPIO*/
  SRAM_GPIO_Config();
 /*使能 FSMC 外设时钟*/
 RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC,ENABLE);
```

```
13    /*********************时序**************************/
14    //地址建立时间（ADDSET）为 1 个 HCLK 1/72M=14ns
15    readWriteTiming.FSMC_AddressSetupTime = 0x00;
16
17    //地址保持时间（ADDHLD）模式 A 未用到
18    readWriteTiming.FSMC_AddressHoldTime = 0x00;
19
20    //数据保持时间（DATAST）为 3 个 HCLK 3/72M=42ns(对 EM 的 SRAM 芯片)
21    readWriteTiming.FSMC_DataSetupTime = 0x02;
22
23    //设置总线转换周期，仅用于复用模式的 NOR 操作
24    readWriteTiming.FSMC_BusTurnAroundDuration = 0x00;
25
26    //设置时钟分频，仅用于同步类型的存储器
27    readWriteTiming.FSMC_CLKDivision = 0x00;
28
29    //数据保持时间，仅用于 NOR
30    readWriteTiming.FSMC_DataLatency = 0x00;
31
32    //选择匹配 SRAM 的模式
33    readWriteTiming.FSMC_AccessMode = FSMC_AccessMode_A;
34
35  /*************************控制***********************************/
36    // 选择 FSMC 映射的存储区域： Bank1 sram3
37    FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM3;
38
39    //设置地址总线与数据总线是否复用，仅用于 NOR
40    FSMC_NORSRAMInitStructure.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable;
41
42    //设置要控制的存储器类型：SRAM 类型
43    FSMC_NORSRAMInitStructure.FSMC_MemoryType =FSMC_MemoryType_SRAM;
44
45    //存储器数据宽度：16 位
46    FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_16b;
47
48    //设置是否使用突发访问模式，仅用于同步类型的存储器
49    FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode =FSMC_BurstAccessMode_Disable;
50
51    //设置是否使能等待信号，仅用于同步类型的存储器
52    FSMC_NORSRAMInitStructure.FSMC_AsynchronousWait=FSMC_AsynchronousWait_Disable;
53
54    //设置等待信号的有效极性，仅用于同步类型的存储器
55    FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity = FSMC_WaitSignalPolarity_Low;
56
57    //设置是否支持把非对齐的突发操作，仅用于同步类型的存储器
58    FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
59
60    //设置等待信号插入的时间，仅用于同步类型的存储器
61    FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive = FSMC_WaitSignalActive_BeforeWaitState;
62
63    //存储器写使能
64    FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable;
65
66    //不使用等待信号
67    FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
68
69    // 不使用扩展模式，读写使用相同的时序
70    FSMC_NORSRAMInitStructure.FSMC_ExtendedMode = FSMC_ExtendedMode_Disable;
71
72    //突发写操作
73    FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
74
75    //读写时序配置
76    FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &readWriteTiming;
```

```
77
78    //读写同样时序，使用扩展模式时这个配置才有效
79    FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &readWriteTiming;
80
81    FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);  //初始化 FSMC 配置
82
83    FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM3, ENABLE);  // 使能 BANK
84 }
```

这个函数的执行流程如下：

(1) 初始化 GPIO 引脚以及 FSMC 时钟

函数开头调用了前面定义的 SRAM_GPIO_Config 函数对 FSMC 用到的 GPIO 进行初始化，并且使用库函数 RCC_AHBPeriphClockCmd 使能 FSMC 外设的时钟。

(2) 时序结构体赋值

接下来对时序结构体 FSMC_NORSRAMTimingInitTypeDef 赋值。在这个时序结构体配置中，由于我们要控制的是 SRAM，所以选择 FSMC 为模式 A，在该模式下配置 FSMC 的控制时序结构体中，实际上只有地址建立时间 FSMC_AddressSetupTime（即 ADDSET 的值）以及数据建立时间 FSMC_DataSetupTime（即 DATAST 的值）成员的配置值是有效的，其它 SRAM 没使用到的成员值全配置为 0 即可。而且，这些成员值使用的单位为：1 个 HCLK 的时钟周期，而 HCLK 的时钟频率为 72MHz，对应每个时钟周期为 1/72 微秒。

由图 27-12 的 FSMC 时序和 SRAM 时序对比及 SRAM 时间参数要求可总结出表 27-5 最右侧的计算表达式。

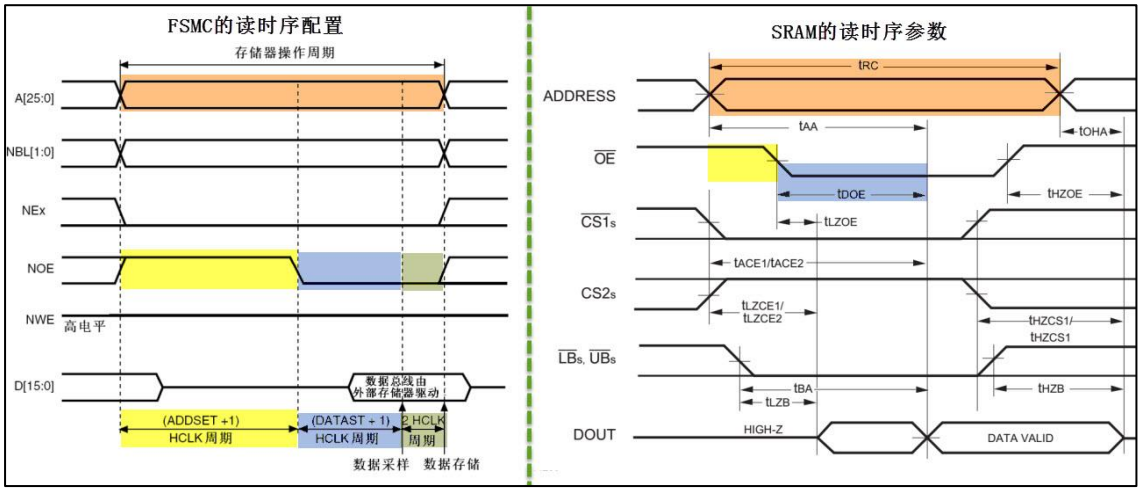

图 27-12 FSMC 时序配置与 SRAM 时序参数要求对比(读)

表 27-5 SRAM 的读操作参数(时间要求摘自《IS62WV51216》规格书)

时间参数	SRAM 要求	说明	FSMC 配置要求表达式
t_{RC}	不小于 55ns	读操作周期	ADDSET+1+DATAST+1+2 > 55ns
小于 t_{LZCE}	SRAM 无给出要求	从发出地址到给出读使能信号的时间	ADDSET+1 > 0 ns
t_{DOE}	最迟不大于 25ns	从接收到读使能信号至给出有效数据的时间	DATAST+1 > 25 ns

根据 FSMC 配置表达式的配置要求把时间单位 1/72 微秒(即 1000/72 纳秒)代入，可求得 ADDSET = 0，DATAST=1 时即可符合要求。如：

t_{RC}=ADDSET+1+DATAST+1+2 =(0+1+1+1+2)*1000/72 = 69 ns > 55 ns

t_{DOE}=DATAST+1 = (1+1)*1000/72 = 27.7 > 25 ns

不过，经实验总结出该配置在连续读取多个数据的时候会出现问题，更正确的配置为 ADDSET=0，DATAST=2，而这样的配置与写时序是一致的，下面再来列出写时序的计算过程：

由图 27-13 的 FSMC 时序和 SRAM 时序对比及 SRAM 时间参数要求可总结出表 27-5 最右侧的计算表达式。

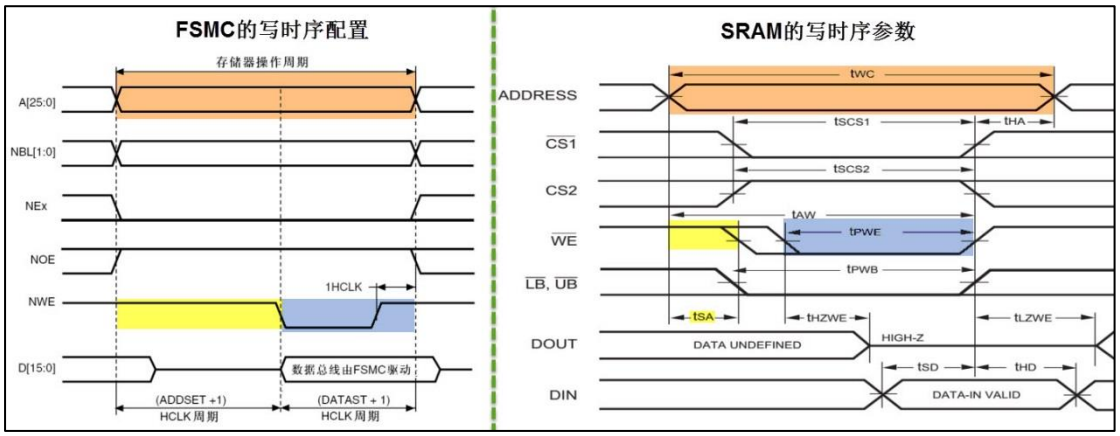

图 27-13 FSMC 时序配置与 SRAM 时序参数要求对比(写)

表 27-6 SRAM 的写操作参数(时间要求摘自《IS62WV51216》规格书)

时间参数	SRAM 要求	说明	FSMC 配置要求表达式
t_{WC}	大于 55ns	写操作周期	ADDSET+1+DATAST+1 > 55ns
t_{SA}	无要求	地址建立时间	ADDSET+1 > 0 ns
t_{PWB}	最早最不小于 40ns	从接收到写使能信号到对数据采样的时间	DATAST+1 > 40 ns

根据 FSMC 配置表达式的配置要求把时间单位 1/72 微秒(即 1000/72 纳秒)代入，可求得 ADDSET = 0，DATAST=2 时即可符合要求。如：

t_{WC} = ADDSET+1+DATAST+1 =(0+1+2+1)*1000/72 = 55.555 ns > 55 ns

t_{PWB} = DATAST+1 = (2+1) *1000/72 = 41.6666 > 40 ns

把计算得的参数赋值到时序结构体中的 FSMC_AddressSetupTime（即 ADDSET 的值）及 FSMC_DataSetupTime（即 DATAST 的值）中，然后再把时序结构体作为指针赋值到下面的 FSMC 初始化结构体中，作为读写的时序参数，最后再调用 FSMC_NORSRAMInit 函数即可把参数写入到相应的寄存器中。

(3) 配置 FSMC 初始化结构体

函数接下来对 FSMC SRAM 的初始化结构体赋值。主要包括存储映射区域、存储器类型以及数据线宽度等，这些是根据外接的 SRAM 的电路设置的。

- 设置存储区域 FSMC_Bank

  FSMC_Bank 成员设置 FSMC 的 SRAM 存储区域映射选择为 FSMC_Bank1_NORSRAM3，这是由于我们的 SRAM 硬件连接到 FSMC_NE3 和 NOR/PSRAM 相关引脚，所以对应到存储区域 Bank1 SRAM3，对应的基地址为 0x68000000；

- 存储器类型 FSMC_MemoryType

  由于我们控制的是 SRAM 类型存储器，所以 FSMC_MemoryType 成员要选择相应的 FSMC_MemoryType_SRAM；

- 数据线宽度 FSMC_MemoryDataWidth

  根据硬件的数据线连接，数据线宽度被配置为 16 位宽 FSMC_MemoryDataWidth_16b；

- 写使能 FSMC_WriteOperation

  FSMC_WriteOperation 用于设置写使能，只有使能了才能正常使用 FSMC 向外部存储器写入数据；

- 扩展模式以及读写时序

  在 FSMC_ExtendedMode 成员中可以配置是否使用扩展模式，当设置扩展模式时，读时序使用 FSMC_ReadWriteTimingStruct 中的配置，写时序使用 FSMC_WriteTimingStruct 中的配置，两种配置互相独立，可以赋值为不同的读写时序结构体。在本实例中不使用扩展模式，即读写时序使用相同的配置，都是赋值为前面的 readWriteTiming 结构体；

- 其它

  配置 FSMC 还涉及到其它的结构体成员，但这些结构体成员与 SRAM 控制不相关，都被设置为 Disable 了；

赋值完成后调用库函数 FSMC_NORSRAMInit 把初始化结构体配置的各种参数写入到 FSMC_BCR 控制寄存器及 FSMC_BTR 时序寄存器中。最后调用 FSMC_NORSRAMCmd 函数使能要控制的存储区域 FSMC_Bank1_NORSRAM3。

使用指针的方式访问 SRAM 存储器

完成初始化 SRAM 后，我们就可以利用它存储数据了，由于 SRAM 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 SRAM，访问这些地址时，FSMC 外设自动读写 SRAM，程序上无需额外操作。

通过地址访问内存，最直接的就是使用 C 语言的指针方式，见代码清单 27-6。

代码清单 27-6 使用指针的方式访问 SRAM

```
1  /*SRAM 起始地址 存储空间 2 的起始地址*/
2  #define Bank1_SRAM3_ADDR     ((uint32_t)0x68000000)
3  /*SRAM 大小，8M 字节*/
4  #define IS62WV51216_SIZE 0x100000
5
6  uint32_t temp;
7
8  /*向 SRAM 写入 8 位数据*/
9  *( uint8_t*) (Bank1_SRAM3_ADDR ) = (uint8_t)0xAA;
10 /*从 SRAM 读取数据*/
11 temp =  *( uint8_t*) (Bank1_SRAM3_ADDR );
12
13 /*写/读 16 位数据*/
14 *( uint16_t*) (Bank1_SRAM3_ADDR +10 ) = (uint16_t)0xBBBB;
15 temp =  *( uint16_t*) (Bank1_SRAM3_ADDR+10 );
16
17 /*写/读 32 位数据*/
18 *( uint32_t*) (Bank1_SRAM3_ADDR +20 ) = (uint32_t)0xCCCCCCCC;
```

```
19 temp =  *( uint32_t*) (Bank1_SRAM3_ADDR+20 );
```

为方便使用，代码中首先定义了宏 Bank1_SRAM3_ADDR 表示 SRAM 的起始地址，该地址即 FSMC 映射的存储区域 Bank SRAM3 的首地址；宏 IS62WV51216_SIZE 表示 SRAM 的大小，所以从地址(Bank1_SRAM3_ADDR)到(Bank1_SRAM3_ADDR+IS62WV51216_SIZE)都表示在 SRAM 的存储空间，访问这些地址，直接就能访问 SRAM。

配合这些宏，使用指针的强制转换以及取指针操作即可读写 SRAM 的数据，使用上跟普通的变量无异。

直接指定变量存储到 SRAM 空间

每次存取数据都使用指针来访问太麻烦了，为了简化操作，可以直接指定变量存储到 SRAM 空间，见代码清单 27-7。

代码清单 27-7 直接指定变量地址的方式访问 SRAM

```
1 /*SRAM 起始地址 存储空间 2 的起始地址*/
2 #define Bank1_SRAM3_ADDR     ((uint32_t)0x68000000)
3 /*绝对定位方式访问 SRAM,这种方式必须定义成全局变量*/
4 uint8_t testValue __attribute__((at(SDRAM_BANK_ADDR)));
5 testValue = 0xDD;
```

这种方式使用关键字“__attribute__((at()))”来指定变量的地址，代码中指定 testValue 存储到 SRAM 的起始地址，从而实现把变量存储到 SRAM 上。要注意使用这种方法定义变量时，必须在函数外把它定义成全局变量，才可以存储到指定地址上。

更常见的是利用这种方法定义一个很大的数组，整个数组都指定到 SRAM 地址上，然后就像使用 malloc 函数一样，用户自定义一些内存管理函数，动态地使用 SRAM 的内存，我们在使用 emWin 写 GUI 应用的时候就是这样做的。参考我们配套的“FSMC—外部 SRAM（内存管理）”实验可以了解如何自行管理内存。。

然而，我们更推荐另一种方法，在本书的《MDK 编译过程及文件类型全解》章节将会讲解使用更简单的方法从 SRAM 中分配变量，以及使用 C 语言标准库的 malloc 函数来分配 SRAM 的空间，更有效地进行内存管理。

3. main 函数

最后我们来编写 main 函数，进行 SRAM 芯片读写校验，见代码清单 25-14。

代码清单 27-8 main 函数

```
1 /**
2  * @brief  主函数
3  * @param  无
4  * @retval 无
5  */
6 int main(void)
7 {
8     /* LED 端口初始化 */
9     LED_GPIO_Config();
10    /* 初始化串口 */
11    USART_Config();
12    printf("\r\n 秉火 iSO SRAM 读写测试例程\r\n");
13    /*初始化 SRAM 模块*/
14    FSMC_SRAM_Init ();
15
```

```
16      /*蓝灯亮，表示正在读写 SRAM 测试*/
17      LED_BLUE;
18      /*对 SRAM 进行读写测试，检测 SRAM 是否正常*/
19      if (SRAM_Test()==1)
20      {
21          //测试正常 绿灯亮
22          LED_GREEN;
23      }
24      else
25      {
26          //测试失败 红灯亮
27          LED_RED;
28      }
29      while (1);
30  }
```

函数中初始化了 LED、串口，接着调用前面定义好的 SRAM_Init 函数初始化 FSMC 及 SRAM，然后调用自定义的测试函数 SRAM_Test 尝试使用 SRAM 存取 8、16 及 32 位数据，并进行读写校验，它就是使用指针的方式存取数据并校验而已，此处不展开。

注意对 SRAM 存储空间的数据操作都要在 FSMC_SRAM_Init 初始化 FSMC 之后，否则数据是无法正常存储的。

下载验证

用 USB 线连接开发板“USB TO UART”接口跟电脑，在电脑端打开串口调试助手，把编译好的程序下载到开发板。在串口调试助手可看到 SRAM 测试的调试信息。

27.9 课后练习

5. 如果要把 SRAM 映射到 FSMC SRAM 的存储区域 Bank1 SRAM1，需要如何修改 STM32 与 SRAM 的硬件连接？程序上需要修改哪些内容？