AN304

F-RAM™ 的 SPI 指南

作者：**Harsha Medu**
相关项目：有
相关器件系列：**FM25xxx**
软件版本：无
相关应用指南：请点击此处

AN304 介绍了 SPI F-RAM 的功能说明、时序和示例代码。此外， 还为 SPI F-RAM 提供了一个基于 PSoC 3 的用户模块。

目录

F-RAM™ 的 SPI 指南 1
目录 1
简介 1
为何使用 SPI? 1
速度优势 2
SPI 总线 2
系统结构 3
独立的 SPI F-RAM 产品简介 4
读/写操作 4
SPI F-RAM 地址 6
将一个存储密度较高的 F-RAM 器件放置到低密度的设计中 7
时钟模式 8
半双工操作 8
写保护 9
电源开关周期 9
总结 9
相关应用笔记 9
附录 A：伪代码的示例（1 字节地址，4 Kbit 器件） 10
附录 B：伪代码的示例（2 字节地址，16 Kbit 到 512 Kbit 器件） 11
附录 C：伪代码的示例（3 字节地址，1 Mbit 到 4 Mbit 器件） 12
附录 D：基于 PSoC 3 的用户模块的伪示例代码 13
文档修订记录 16
全球销售和设计支持 17

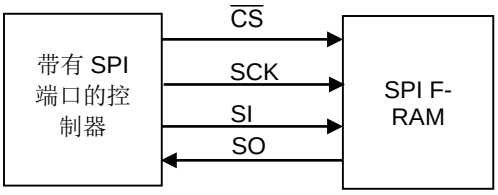
简介

FM25xxx F-RAM 产品系列采用了工业标准中 4 线 SPI 接口。这些存储器都是高速（最高达 40 MHz）低功耗非易失性器件。SPI F-RAM 的存储空间范围为：4 Kbit 至 4 Mbit。本应用笔记将讨论这些器件的功能和时序方面的内容。

为何使用 SPI?

串行外设接口（SPI）是一个由 Motorola（现为 Freescale）创建的串行总线，该接口在其公司的 MCU 产品上或其他半导体供应商（如赛普拉斯、TI、Atmel、Microchip、Analog Devices 等等）的产品上专门作为接口使用。此外，还在 DSPs、网络处理器、FPGA 等器件中使用了 SPI 端口。对于基于微控制器的需要高串行数据速率的系统来说，SPI 接口是理想的选择。串行数据吞吐量与串行时钟速度相关（请参考图 1 中所显示的 SCK 信号）。赛普拉斯大多数串行 F-RAM 的时钟频率可达 40 MHz。如果某些微控制器上没有专用的 SPI 端口，这时，可以通过 bit-banging 的方法使用 GPIO 引脚来执行 SPI 操作。该方法需要软件来控制或“bangs away”I/O 端口。图 1 显示的是基本 SPI 接口。

图 1. 基本 SPI 接口

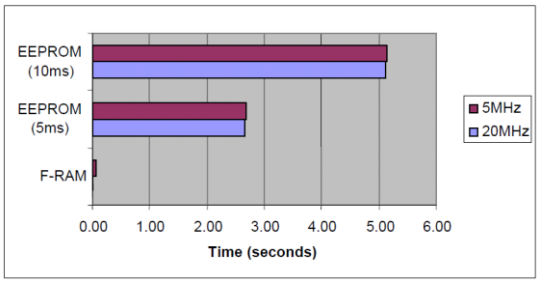


速度优势

与 EEPROM 或闪存相比，使用 F-RAM 存储器技术，写入大数据块的速度会更快。与 EEPROM 或闪存不同，F-RAM 器件没有使用页面缓冲区。收到每个字节的第 8 位后，F-RAM 会立即写入该数据字节。由于 F-RAM 同时具有 no-write（无写入）延迟和高速时钟，因此，对于需要快速写入大量数据的应用来说，它是不二选择。另外，设计师完全可以决定写入 SPI F-RAM 的字节数量。当你将一个或两个字节写入 F-RAM 内的随机位置时，写入周期时间大约为 1 µs，而将同样的内容写入到 EEPROM 或闪存内，需耗时 5 ms ~ 10 ms。此外，你也不用再担心页缓冲区大小的问题（通常，随着时间的推移，系统需要新的存储空间，引导页面缓冲区的大小也会增加）。

图 2 展示了将一个 256 Kbit 数组写入 F-RAM 和 EEPROM 内的比较图。在相同的时钟频率下， 即使与带有 64 字节页缓冲区的 EEPROM 相比，F-RAM 器件的速度也高好几个数量级。对于对烧录系统配置有时间限制的产品制造线，上述特性具有重大的意义。

图 2. 填充一个大小为 256 Kbit 的 SPI 存储数组的写入时间



请注意，就算提高了时钟频率，将其从 5 MHz 增大到 20 MHz，也不能极大地缩短写入 256 Kbit EEPROM 存储器的时间。通常每个页面写入都需要很长的延迟。对于 20 MHz 的 F-RAM 存储器，只需要 13 ms 时间用于写入整个 32 Kbyte 数组，它并不属于图 2 显示的范围内。

SPI 总线

SPI 接口有 4 个引脚，如图 1 所示。$\overline{CS}$处于低电平时，将执行所有操作。这时，地址、控制和数据会以字节为单位连续记录到器件内。地址、控制和数据输入通过 SI 被传输到器件内，数据则要通过 SO 引脚被输出。

通过这些操作码，可以控制器件。执行读取和写入操作通常按照下列序列执行：操作码、地址和数据。用来设置/清除状态寄存器中的写使能锁存位（WEL）是不参与数据传输的另外两个数据操作（表 4 显示了该状态寄存器，并且在本文档的后面章节中对其进行说明）。对于 EEPROM 和基于闪存的 SPI 存储器，状态寄存器也包含了一个重要的位，即 $\overline{RDY}$。该位用来指示 SPI 控制器某个写周期是否完成。完成写操作后，访问器件前，EEPROM 和闪存存储器通常需要 5 ms 至 10 ms 的延时。对于 F-RAM，能够完成内部写操作而无需延迟或等待，因此，$\overline{RDY}$ 位始终为逻辑‘0’（F-RAM 状态寄存器包含该 $\overline{RDY}$ = 0 位，以便包含与较慢 EEPROM 和 Flash 存储器一起工作的固件的控制器可以快速适应速度更快的 F-RAM 产品。）也就是说，控制器能以 RAM 的速度对 F-RAM 进行读和写操作。

共有六个操作码可用于控制 SPI F-RAM 器件。少数 SPI F-RAM 器件还添加了用于快速读取、进入睡眠状态、器件 ID 和序列号读取等功能的附加操作码。每个操作码都是一个 8 位指令，可命令存储器或状态寄存器执行某些操作。因此，每个有效的$\overline{CS}$周期只能发送一个操作码。表 1 介绍了所有操作码：

表 1. 操作码说明

名称	操作码	地址	Dummy（虚拟）字节	数据	操作
WREN	0000_0110b	-	-	-	设置 WEL 位
WRITE	0000_0010b[2]	3 字节[1]	-	存储器数据输入	如果 WEL=1， 数据将被写入到 F-RAM 数组内。 当$\overline{CS}$处于高电平时，WEL 被清除。
READ	0000_0011b[2]	3 字节[1]	-	存储器数据输出	读取 F-RAM 数组内的数据
WRDI	0000_0100b	-	-	-	清除 WEL
RDSR	0000_0101b	-	-	状态寄存器数据输出	读取 WPEN、BP(1:0)和 WEL 位
WRSR	0000_0001b	-	A11	状态寄存器数据输入	写入 WPEN 和 BP(1:0)位
SLEEP[3]	1011_1001b	-	-	-	进入睡眠模式

名称	操作码	地址	Dummy（虚拟）字节	数据	操作
FSTRD[3]	0000_1011b	3 字节[1]	1 字节	存储器数据输出	以频率为 40 MHz 从 F-RAM 数组读取数据
RDID	1001_1111b	-	-	9 字节器件 ID 数据输出	读取 9 字节器件 ID
SNR[3]	1100_0011b	-	-	8 字节序列号数据输出	读取 8 字节序列号

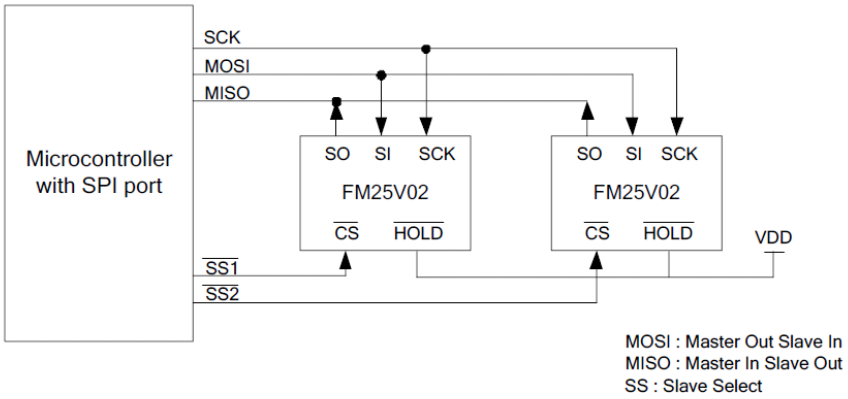
注释：

1. 根据不同的密度，某些 SPI 器件可能使用 1 位或 2 位地址。请参考表 2。
2. 对于 4 Kbit 器件，写入和读取操作码的位 3 与高地址位（A8）相对应。
3. 可能所有的 SPI 器件不支持该指令。

系统结构

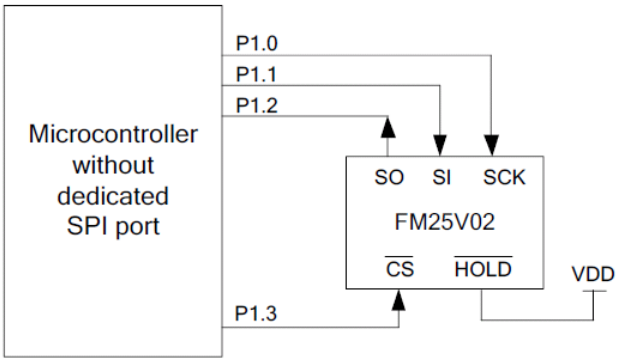
当需要连接多个 F-RAM 器件到控制器时，需要控制器使用额外的控制引脚进行片选。图 3 显示的系统配置表示连接至微控制器上标准 SPI 端口的两个 F-RAM 器件。

图 3．两个 F-RAM 器件的系统配置



对于没有专用 SPI 总线的微控制器，可以使用通用端口，如图 4 所示。Bit-banging 代码将控制该接口。

图 4．使用 GPIO 的微控制器的系统配置 — 单 F-RAM



独立的 SPI F-RAM 产品简介

下表汇总了多种独立的 SPI F-RAM 产品的基本特性。

表 2. SPI F-RAM 产品系列

		3 V									5 V			
	FM25L04B	**FM25L16B**	**FM25CL64B**	**FM25V01**	**FM25V02**	**FM25V05**	**FM25V10**	**FM25V20 FM25V20A**	**FM25H20**	**FM25V40**	**FM25040B**	**FM25C160B**	**FM25640B**	**FM25W256**
密度	4 Kbit	16 Kbit	64 Kbit	128 Kbit	256 Kbit	512 Kbit	1 Mbit	2 Mbit	2 Mbit	4 Mbit	4 Kbit	16 Kbit	64 Kbit	256 Kbit
内部组织方式	512 x 8	2K x 8	8K x 8	16K x 8	32K x 8	64K x 8	128K x 8	256K x 8	256K x 8	512K x 8	512 x 8	2K x 8	8K x 8	32K x 8
地址位数量	9	11	13	14	15	16	17	18	18	19	9	11	13	15
地址字节数量	1	2	2	2	2	2	3	3	3	3	1	2	2	2
工作电压	2.7-3.6 V	2.7-3.6 V	2.7-3.65 V	2.0-3.6 V	2.0-3.6 V	2.0-3.6 V	2.0-3.6 V	2.0-3.6 V	2.7-3.6 V	2.0-3.6 V	4.5-5.5 V	4.5-5.5 V	4.5-5.5 V	2.7-5.5 V
最大时钟频率	20 MHz	20 MHz	20 MHz	40 MHz	40 MHz	40 MHz	40 MHz	40 MHz	40 MHz	40 MHz	20 MHz	20 MHz	20 MHz	20 MHz
支持的时钟模式	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3	0, 3
睡眠模式				✓	✓	✓	✓	✓	✓	✓				
独特的 S/N							✓							
器件 ID				✓	✓	✓	✓	✓		✓				
封装	SOIC8 DFN8 (4x4.5)	SOIC8 DFN8 (4x4.5)	SOIC8 DFN8 (4x4.5)	SOIC8	SOIC8 DFN8 (4x4.5)	SOIC8	SOIC8	大型 SOIC8 DFN8[1] (5x6)	大型 SOIC8 DFN8[1] (5x6)	大型 SOIC8 DFN8[1] (5x6)	SOIC8	SOIC8	SOIC8	SOIC8

注释：

1. 5 x 6 mm DFN8 符合 SOIC8 封装。

读/写操作

SPI 接口与控制器驱动的时钟同步。所有 F-RAM SPI 器件将在 SCK 的上升沿上寄存数据输入，并在 SCK 的下降沿将数据回送给控制器。为符合时序要求，控制器通常在 SCK 的下降沿上将信号传输到存储器内，以便信号有时间进行传输并满足存储器器件的设置时序规范。

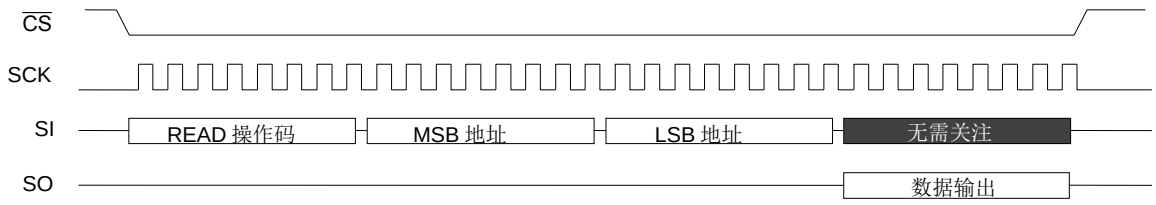
存储器读取的操作

格式：READ 操作码，MSB 地址，LSB 地址，数据输出，（数据输出，数据输出...）

读取周期期间，控制机将发送一个 READ 操作码和地址。将通过 SO 引脚传输数据，例如，data-out(0)、data-out(1)、data-out(2)等。在该周期内，$\overline{CS}$引脚必需保持为低电平。如果未将$\overline{CS}$设置为高电平，将停止输出数据，并且 SO 进入高阻态。输入的地址是第一个数据字节的起始地址。只需保持$\overline{CS}$为低电平，便可以访问后续数据字节，同时，传送完数据字节后，从地址读取的每一个字节都会被 SPI F-RAM 器件递增。

图 5 显示了一个用于存储密度为 16 Kbit 至 512 Kbit 的双字节地址。

图 5. 读取 SPI 时序



存储器写入操作

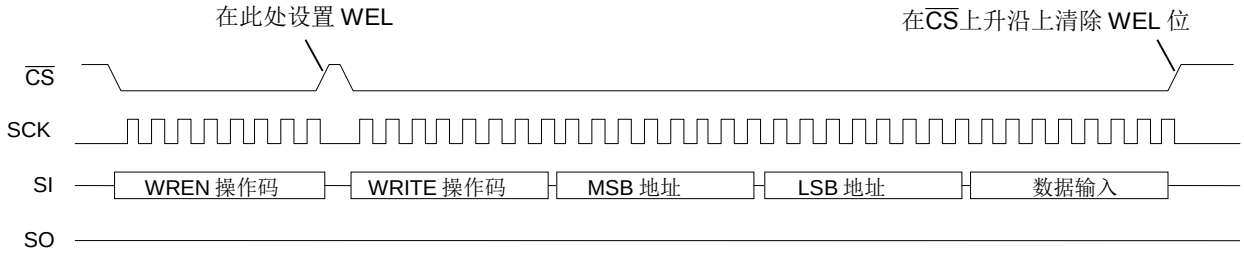
格式: WREN 操作码，WRITE 操作码，MSB 地址，LSB 地址，数据输入，（数据输入，数据输入...）

执行写入周期需要控制器按照下面的顺序发送两个操作码：WREN 和 WRITE。发送每个操作码时必须保持$\overline{CS}$为低电平。发送完 WRITE 操作码、地址和数据（如 data-in(0)、data-in(1)、data-in(2)等等）后，将发送 WREN。输入的地址是第一个数据字节的起始地址。要保持$\overline{CS}$为低电平，便能够访问后续的数据字节，同时，传输数据字节时，写入到地址内的每个字节会被 SPI F-RAM 器件递增。每个数据字节都会在它的第 8 个时钟沿上被写到 F-RAM 数组内。未发生页面缓冲区或写入延迟。

请注意，状态寄存器内的 WEL 位将被内部设置，并被 SPI F-RAM 器件清除。它在传输 WREN 操作码后被设置，并且完成写入操作后在$\overline{CS}$上升沿上被清除。读取 WREN 和 WRITE 操作码之间的状态寄存器（RDSR 操作码）不会清除 WEL 位。发送 WREN 操作码后，用户会马上读取状态寄存器，以检查 WEL 位是否被设置。但完成写入操作后便不需要读取 WEL 位。

图 6 展示了一个完整的单字节写入操作。它显示了一个用于存储密度为 16 Kbit 至 512 Kbit 的双字节地址。

图 6. 写入 SPI 时序



状态寄存器写入操作

格式：WREN 操作码，WRSR 操作码，数据输入

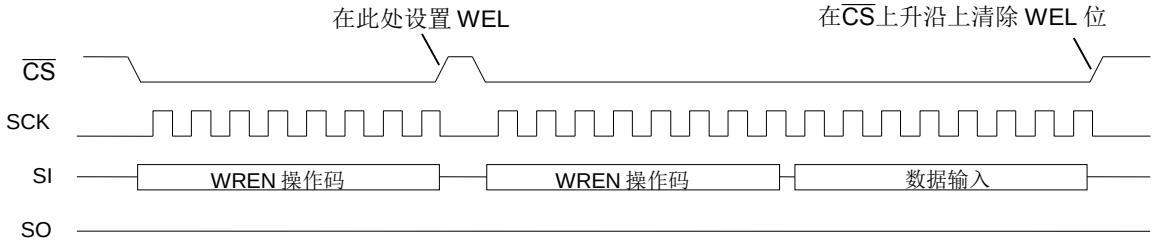
状态寄存器包含$\overline{WP}$使能位（WPEN）以及模块保护（BP1, BP0）位，如下所示。

Status Register							
7	6	5	4	3	2	1	0
WPEN	0	0	0	BP1	BP0	WEL	0

通过写入状态寄存器，用户可以对存储器模块进行写保护操作，也可以使能$\overline{WP}$引脚。共有两个模块保护位：BP1 和 BP0。通过上述两位，可以防止对数组的前四分之一、前半部分或者整个数组进行写操作。BP0、BP1 和 WPEN 位被黄色高亮显示，用于指示它们是非易失性位，即在发生电源循环事件时会保持其写入值。WPEN 用于使能或禁用外部$\overline{WP}$引脚。系统被干预时，软件可以通过该位覆盖掉$\overline{WP}$引脚。WEL 是一个只读位，用来通知用户是否已经设置好了写入使能锁存位，从而允许写入状态寄存器或存储器。

图 7 显示了一个完整的状态寄存器写入操作。

图 7. 写入状态寄存器时序

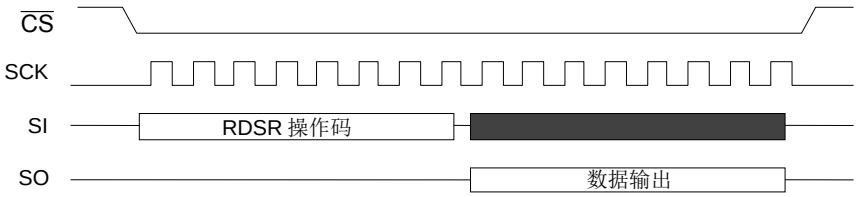


状态寄存器读取操作

格式：RDSR 操作码，数据输出

通过读取状态寄存器，用户可以查看 WPEN 位、BP (1:0)写保护位和 WEL 位的状态。图 8 显示的是一个完整的状态寄存器读取操作。

图 8。读取状态寄存器时序



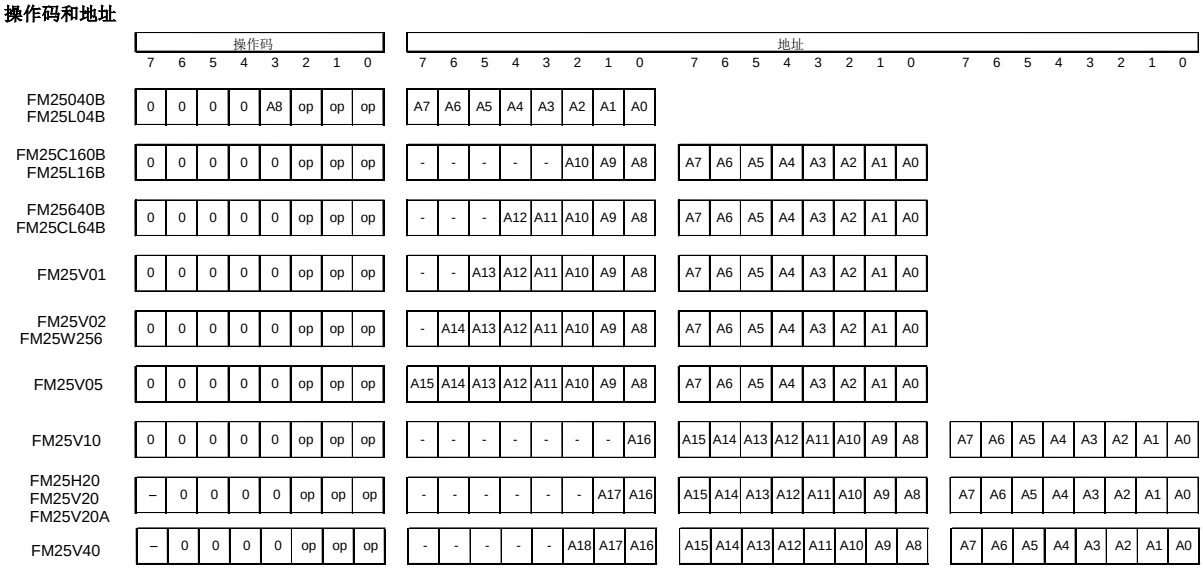
SPI F-RAM 地址

根据不同的存储密度，各个独立的 SPI F-RAM 器件需要 1 字节、2 字节或 3 字节地址。执行操作码后，会立即转移到 MSB 中的起始地址。传输最低有效地址字节后（LSB），主控会等待数据输入来执行写入操作，以及存储区将驱动发送数据输出来执行读取操作。只要 SCK 继续进行切换，那么内部地址将自动被递增，并且数据输入/输出也会持续执行，直至CS被取消激活。

注意： 4 Kbit 的器件只需要 1 字节的地址。

图 9 显示的是不同密度的 SPI F-RAM 器件的地址。

图 9。不同密度的地址差别



将一个存储密度较高的 F-RAM 器件放置到低密度的设计中

如果一个特殊的器件不可用，并且电路板或系统专用于存储密度较低的器件，这时，可以使用一个存储密度较高的器件来替换它。例如，一个设计使用 16 Kbit 器件的系统也可以使用 64 Kbit、128 Kbit、256 Kbit 或 512 Kbit 的器件。图 9 指出这些器件使用 2 字节的地址来执行读取和写入操作。器件在这些范围内通常具有相同的的引脚分布、封装（SOIC）和读/写功能，并假设它们完全符合工作电压和时序要求。存在两个潜在的问题：如果系统使用器件的地址覆盖性能；系统使用模块保护性能。一个 16 Kbit 的器件覆盖 0x800，一个 64 Kbit 的器件覆盖 0x2000，还有一个 128 Kbit 的器件覆盖 0x4000，并以此类推。在比较两个器件的密度时，其中一个是另一个的两倍，那么这两个模块保护的边缘距离也是地址的两倍。

例如，图 10 和图 11 分别显示的是 16 Kbit 和 128 Kbit 器件的串行地址流的差异。

图 9 显示的是 16 Kbit 和 128 Kbit 器件地址要求的比较情况，很明显，3 个额外地址位的位置（ 图 11 中红色加亮显示的 A13、A12 和 A11）被使用于 128 Kbit 器件。可以将一个密度较高的器件使用在密度较低器件的系统内，但前提是控制器在驱动这三个地址位来执行读取和写入操作时保持一致。由于密度为 1 Mbit（或更高）的器件使用了 3 字节地址，因此不能将它作为密度较低系统的替代品。例如，如果使用了一个三字节地址存储器，则能够发送 2 字节地址的系统便不能顺畅运行。请参考表 2，了解密度的地址字节数量。

图 10. FM25L16B 写周期（WREN 未显示）

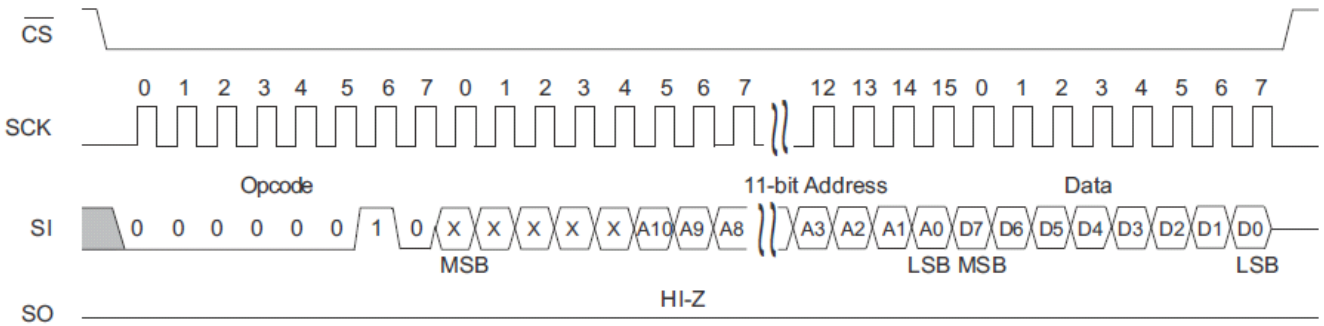
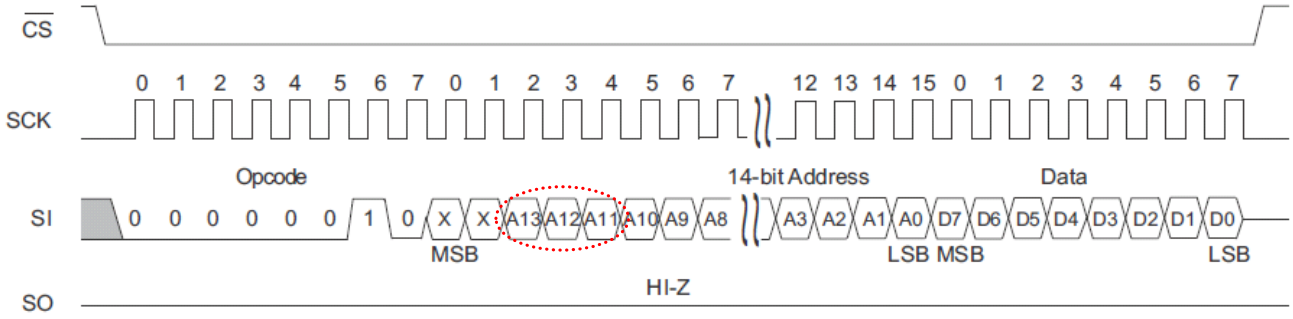


图 11. FM25V01 写周期（WREN 未显示）



时钟模式

FM25xxx 器件系列支持四个 SPI 标准时钟模式中的两个：模式 0 和模式 3。请注意，由于各个模式具有独立性，因此所有 F-RAM 器件均在 SCK 上升沿上将数据输入到器件内，在 SCK 下降沿上送出数据。模式 0 和模式 3 的区别在于：$\overline{CS}$被置为低电平时，SCK 的电平状态不同。表 3 列出了不同的 SPI 模式。

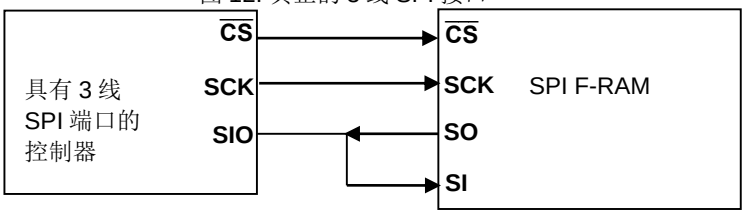
表 3.SPI 模式

	模式 0	模式 1	模式 2	模式 3
SCK 开始	低电平	低电平	高电平	高电平
SI 数据输入被锁存…	↑	↓	↓	↑
SO 数据输出驱动从…	↓	↑	↑	↓

半双工操作

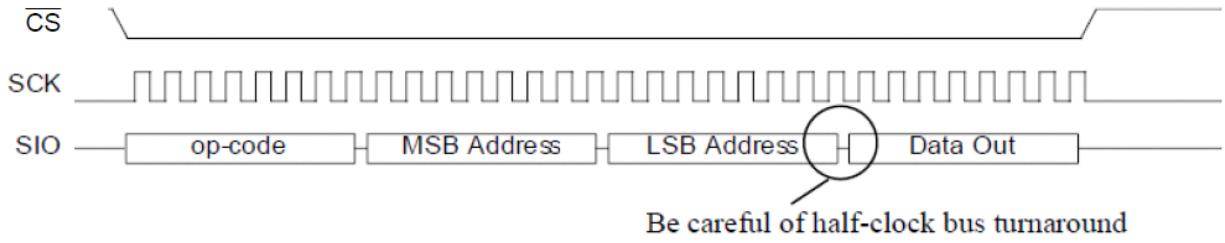
为了减少 SPI 接口上引脚的数量，可以将多个数据线连接在一起，从而形成一个共用的数据 I/O 线。这个 3 线接口是 SPI 的引脚数量最小的配置，如图 12 所示。在这里控制器必需保证在读取周期期间 SIO 线处于高阻态。否侧，会发生总线冲突。另外，由于当前数据总线是半双工的，所以它会潜在降低数据的带宽。

图 12. 真正的 3 线 SPI 接口



无论时序模式的选择如何（0 还是 3），所有 SPI F-RAM 都会在 SCK 上升沿上锁存数据输入，并在 SCK 下降沿上驱动数据输出。图 13 显示的是一个 SPI 读操作。

图 13. 通用数据 I/O 线和总线周转



写保护

可以使用$\overline{WP}$软件引脚或者编程状态寄存器对 SPI F-RAM 器件进行写保护。除了保护 F-RAM 存储器数组外，状态寄存器也可以自保护。由于状态寄存器具有非易失性模块保护位 BP (1:0)，所以可以禁用对存储器数组部分进行写操作。表 5 中的 BP0、BP1 和 WPEN 位被黄色高亮显示，以表明它们是非易失性位，因而在电源循环期间它们的设置会保持不变。WPEN 使能$\overline{WP}$硬件引脚。位置 0 的位被保留为$\overline{RDY}$位，以便使用于 EEPROM 和串行闪存。该位使用于这些器件，以便用户可以通过读取状态寄存器确定存储器是否能够用于其他指令。由于写周期后芯片始终处于就绪状态（无延时），因此在所有 SPI F-RAM 器件内，$\overline{RDY}$位均被内部固定设置为低电平。

表 4：状态寄存器和模块保护设置

Status Register							
7	6	5	4	3	2	1	0
WPEN	0	0	0	BP1	BP0	WEL	0

表 5 介绍了所有写保护状态寄存器和 F-RAM 数组的场合。 当 WEL = 0 时，对 F-RAM 数组和状态寄存器进行的所有写操作均被锁定。

表 5. 写保护

WEL	WPEN	$\overline{WP}$	Protected Blocks	Unprotected Blocks	Status Register
0	X	X	Protected	Protected	Protected
1	0	X	Protected	Unprotected	Unprotected
1	1	0	Protected	Unprotected	Protected
1	1	1	Protected	Unprotected	Unprotected

- 即使 WPEN = 0 和$\overline{WP}$引脚 = 1，F-RAM 数组仍受处于高电平的 BP 位保护。
- 即使 WPEN = 1 和$\overline{WP}$引脚 = 0，当（各个）BP 位处于低电平时，F-RAM 数组仍不受保护。
- 只有 WPEN = 1 并且$\overline{WP}$引脚 = 0 时，状态寄存器才受保护。

注意：FM25040B 和 FM25L04B 中不存在 WPEN 位。当$\overline{WP}$引脚 = 0 时，对存储器数据和状态寄存器进行的所有写操作均被锁定。

电源开关周期

F-RAM 器件是一个高速的非易失性存储器。如果在某个读或写序列中发生电源故障，可能会错误地覆盖掉（破坏）阵列中的数据。例如，芯片选择处于有效状态时（低电平），如果电源级别为中等，那么器件可能无意中写入了数据。SPI F-RAM 数据手册中规定（推荐）将器件断电前先好使芯片选择变为无效（高电平）状态。

SPI F-RAM 包括了一个简单的内部加电复位电路，而没有电源管理电路。请确保 V_{DD} 范围位于数据手册的容差范围内，以阻止发生不正确的操作。建议正确控制 V_{DD} 电源的上升和下降。当切换模式电源接通或断开时，它们都适合不受控制的输出。

系统设计员应注意在电源周期内芯片的使能和 V_{DD} 状态。更多有关数据保护的信息，请参考“*AN302 - F-RAM SPI 读和写内部操作和数据保护*”。

总结

本应用笔记介绍了不同 F-RAM SPI 器件的功能特性、时序和示例代码。

相关应用笔记

若想进一步了解 SPI F-RAM 器件，请参考下面列出的应用笔记。

- AN302 — F-RAM SPI 读和写内部操作和数据保护
- AN408 — SPI F-RAM 协处理器的设计指南—FM33256B

附录 A：伪代码的示例（1 字节地址，4 Kbit 器件）

```
#define WREN  0x06
#define WRITE 0x02 // Write opcode to access lower half of memory
#define WRITE 0x0A // Write opcode to access upper half of memory
#define READ  0x03  // Read opcode to access lower half of memory
#define READ  0x0B  // Read opcode to access upper half of memory
#define RDSR  0x05
#define WRSR  0x01
#define WRDI  0x04
```

在下面的所有场合中，括号均指示 $\overline{CS}$ 引脚变为低电平（LOW）“(“和高电平（HIGH）“)”。

```
/****** Memory Write (single byte to location 0130h) *******/
WREN (0x06)         // Sets WEL bit. WREN must precede WRITE opcode.
WRITE (0x0A,        // 0x02 is WRITE opcode and A8 bit set
0x30,               // starting address
0x55)               // 0x55 is data written to location 0130h

/****** Memory Write (multiple bytes to starting location 01FCh) *******/
WREN (0x06)         // Sets WEL bit. WREN must precede WRITE opcode.
WRITE (0x0A,        // 0x02 is WRITE opcode and A8 bit set
0xFC,               // starting address
0x55,               // 0x55 is data written to location 01FCh
0xAA,               // 0xAA is data written to location 01FDh
0x55,               // 0x55 is data written to location 01FEh
0xAA)               // 0xAA is data written to location 01FFh

/****** Memory Read (single byte from location 01D3h) *******/
READ (0x0B,         // 0x03 is READ opcode
0xD3,               // starting address
0xAA)               // 0xAA is data read from location 01D3h

/****** Memory Read (multiple bytes from starting location 01FCh) *******/
READ (0x0B,         // 0x03 is READ opcode
0xFC,               // starting address
0x55,               // 0x55 is data read from location 01FCh
0xAA,               // 0xAA is data read from location 01FDh
0x55,               // 0x55 is data read from location 01FEh
0xAA)               // 0xAA is data read from location 01FFh

/****** Write Status Register (write protect upper half of memory) *******/
WREN (0x06)         // Sets WEL bit. WREN must precede WRSR opcode.
WRSR (0x01,         // 0x01 is WRSR opcode
0xF8)               // 0xF8 sets the BP1 bit which protects the upper
                    // half of the memory array. The upper nibble
                    // set to “F” attempts to write the upper bits
                    // to 1.

/****** Read Status Register *******/
RDSR (0x05,         // 0x05 is RDSR opcode
0x08)               // 0x08 tells us that the BP1 bit is set and that
                    // the upper half of the memory array is protected.
                    // The upper nibble returns “0” since they are
                    // hardwired low.
```

注意：**蓝色**显示的文本是正在被控制器传输的数据。**红色**显示的内容是控制器正在接受的数据。

附录 B：伪代码的示例（2 字节地址，16 Kbit 到 512 Kbit 器件）

```
#define WREN 0x06
#define WRITE 0x02
#define READ 0x03
#define RDSR 0x05
#define WRSR 0x01
#define WRDI 0x04
```

在下面的所有场合中，括号均指示 $\overline{CS}$ 引脚变为低电平（LOW）“(“和高电平（HIGH）“)”。

```
/****** Memory Write (single byte to location 0F30h) *******/
WREN (0x06)  // Sets WEL bit. WREN must precede WRITE opcode.
WRITE (0x02, // 0x02 is WRITE opcode
0x0F,        // starting address MSB
0x30,        // starting address LSB
0x55)        // 0x55 is data written to location 0F30h

/****** Memory Write (multiple bytes to starting location 07FC) *******/
WREN (0x06)  // Sets WEL bit. WREN must precede WRITE opcode.
WRITE (0x02, // 0x02 is WRITE opcode
0x07,        // starting address MSB
0xFC,        // starting address LSB
0x55,        // 0x55 is data written to location 07FCh
0xAA,        // 0xAA is data written to location 07FDh
0x55,        // 0x55 is data written to location 07FEh
0xAA)        // 0xAA is data written to location 07FFh

/****** Memory Read (single byte from location 0F31h) *******/
READ (0x03,  // 0x03 is READ opcode
0x0F,        // starting address MSB
0x31,        // starting address LSB
0xAA)        // 0xAA is data read from location 0F31h

/****** Memory Read (multiple bytes from starting location 07FCh) *******/
READ (0x03,  // 0x03 is READ opcode
0x07,        // starting address MSB
0xFC,        // starting address LSB
0x55,        // 0x55 is data read from location 07FCh
0xAA,        // 0xAA is data read from location 07FDh
0x55,        // 0x55 is data read from location 07FEh
0xAA)        // 0xAA is data read from location 07FFh

/****** Write Status Register (write protect upper half of memory) *******/
WREN (0x06) // Sets WEL bit. WREN must precede WRSR opcode.
WRSR (0x01, // 0x01 is WRSR opcode
0x08)       // 0x08 sets the BP1 bit which protects the upper
            // half of the memory array.
/****** Read Status Register *******/
RDSR (0x05, // 0x05 is RDSR opcode
0x88)       // 0x88 tells us that the BP1 bit is set and that
            // the upper half of the memory array is protected.
            // The WPEN bit is also set which works with
            // the WP pin to protect the Status Register.
```

注意：**蓝色**显示的文本是正在被控制器传输的数据。**红色**显示的内容是控制器正在接受的数据。

附录 C：伪代码的示例（3 字节地址，1 Mbit 到 4 Mbit 器件）

```
#define WREN 0x06
#define WRITE 0x02
#define READ 0x03
#define RDSR 0x05
#define WRSR 0x01
#define WRDI 0x04
```

在下面的所有场合中，括号均指示 $\overline{CS}$ 引脚变为低电平（LOW）“(“和高电平（HIGH）“)”。

```
/****** Memory Write (single byte to location 1BF30h) *******/
WREN (0x06)  // Sets WEL bit. WREN must precede WRITE opcode.
WRITE (0x02, // 0x02 is WRITE opcode
0x01,        // starting address MSB
0xBF,        // starting address 2nd byte
0x30,        // starting address LSB
0x55)        // 0x55 is data written to location 1BF30h

/****** Memory Write (multiple bytes to starting location 1B7FC) *******/
WREN (0x06)  // Sets WEL bit. WREN must precede WRITE opcode.
WRITE (0x02, // 0x02 is WRITE opcode
0x01,        // starting address MSB
0xB7,        // starting address 2nd byte
0xFC,        // starting address LSB
0x55,        // 0x55 is data written to location 1B7FCh
0xAA,        // 0xAA is data written to location 1B7FDh
0x55,        // 0x55 is data written to location 1B7FEh
0xAA)        // 0xAA is data written to location 1B7FFh

/****** Memory Read (single byte from location 1BF31h) *******/
READ (0x03,  // 0x03 is READ opcode
0x01,        // starting address MSB
0xBF,        // starting address 2nd byte
0x31,        // starting address LSB
0xAA)        // 0xAA is data read from location 1BF31h

/****** Memory Read (multiple bytes from starting location 1B7FCh) *******/
READ (0x03,  // 0x03 is READ opcode
0x01,        // starting address MSB
0xB7,        // starting address 2nd byte
0xFC,        // starting address LSB
0x55,        // 0x55 is data read from location 1B7FCh
0xAA,        // 0xAA is data read from location 1B7FDh
0x55,        // 0x55 is data read from location 1B7FEh
0xAA)        // 0xAA is data read from location 1B7FFh



/****** Write Status Register (write protect upper half of memory) *******/
WREN (0x06)  // Sets WEL bit. WREN must precede WRSR opcode.
WRSR (0x01,  // 0x01 is WRSR opcode
0x08)        // 0x08 sets the BP1 bit which protects the upper
             // half of the memory array.

/****** Read Status Register *******/
RDSR (0x05,  // 0x05 is RDSR opcode
0x88)        // 0x88 tells us that the BP1 bit is set and that
```

```
// the upper half of the memory array is protected.
// The WPEN bit is also set which works with
// the WP pin to protect the Status Register.
```

注意：**蓝色**显示的文本是正在被控制器传输的数据。**红色**显示的内容是控制器正在接受的数据。

附录 D：基于 PSoC 3 的用户模块的伪示例代码

下面介绍的是用于该用户模块的代码示例。

SPI 存储器写入操作

```
/*********** Memory Write (write 0x55 0xAA 0x55 0xAA from location 1B7FCh) **********/
data[0] = 0x55;
data[1] = 0xAA;
data[2] = 0x55;
data[3] = 0xAA;

WRITE (0x01B7FC,          // Sets the address pointer to memory location 1B7FCh
           data,          // Write data
           0x04);         // Number of bytes to be written
```

SPI 存储器读取操作

```
/*********** Memory Read (read 4 bytes from location 1B7FCh) **********/
READ (0x01B7FC,           // Sets the address pointer to memory location 1B7FCh
          data,           // Pointer to store the read data
          0x04);          // Number of bytes to be read
// data buffer contains 0x55, 0xAA, 0x55, 0xAA after read
```

SPI 写入功能

```
/**********PSoC3 Based Code for SPI Write **********/
uint8 WRITE (uint32 addr, uint8 *data_write_ptr, uint32 total_data_count)
{
   uint8 i;

   // Clear the Transmit Buffer
   NVRAM_SPI_1_SPIM_ClearTxBuffer();

   // Send Write Enable WREN command
   // Make chip select LOW CS = 0
   NVRAM_SPI_1_CS_Reg_Write(0);
   // Send Write Enable Command WREN
   NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_WREN);
   // Wait for the transfer to complete
   while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) !=
   NVRAM_SPI_1_SPIM_STS_SPI_DONE);
   // Make chip select High CS = 1
   NVRAM_SPI_1_CS_Reg_Write(1);

   // Delay
   CyDelay(1);

   // Clear the Transmit Buffer
   NVRAM_SPI_1_SPIM_ClearTxBuffer();

   // Make chip select LOW CS = 0
   NVRAM_SPI_1_CS_Reg_Write(0);
   // Send NVRAM Write Command
   NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_SRAM_WRITE_CMD);
   // Send NVRAM address
   if(NVRAM_SPI_1_spi_density >= SPI_1MBit)
   {
      // Send MSB of 3-byte address for F-RAM densities of 1-Mbit and greater
      NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>16));
   }
   // Send the second byte of 3-byte address or MSB of 2-byte address
   NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>8));
   // Send LSB address byte
   NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr));

   // Wait for the transfer to complete
   while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) !=
   NVRAM_SPI_1_SPIM_STS_SPI_DONE);

   // Send NVRAM data
   for(i = 0; i < total_data_count; i++ )
   {
      NVRAM_SPI_1_SPIM_WriteTxData((uint8)(data_write_ptr[i]));
      while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) !=
      NVRAM_SPI_1_SPIM_STS_SPI_DONE);
   }
   // Make chip select HIGH CS = 1
   NVRAM_SPI_1_CS_Reg_Write(1);
}
```

SPI 读取功能

```
/*****************PSoC3 Based Code for SPI Read**************/
uint8 READ (uint32 addr, uint8 *data_read_ptr, uint32 total_data_count)
{
   uint8 i;

   // Clear the Transmit Buffer
   NVRAM_SPI_1_SPIM_ClearTxBuffer();

   // Make chip select LOW CS = 0
   NVRAM_SPI_1_CS_Reg_Write(0);
   // Send NVRAM Read Command
   NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_SRAM_READ_CMD);
   // Send NVRAM Address
   if(NVRAM_SPI_1_spi_density >= SPI_1MBit)
   {
      // Send MSB of 3-byte address for F-RAM densities of 1-Mbit and greater
      NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>16));
   }
   // Send the second byte of 3-byte address or MSB of 2-byte address
   NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>8));
   // Send LSB address byte
   NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr));

   // Wait for the transfer to complete
   while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) !=
   NVRAM_SPI_1_SPIM_STS_SPI_DONE);

   // Read the data and store in data_read_ptr
   for(i = 0; i < total_data_count; i++ )
   {
      // Clear receive buffer
      NVRAM_SPI_1_SPIM_ClearRxBuffer();
      // Send a dummy byte
      NVRAM_SPI_1_SPIM_WriteTxData((uint8)0x00);
      // Wait for the transfer to complete
      while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) !=
      NVRAM_SPI_1_SPIM_STS_SPI_DONE);
      // Wait till the receive buffer has received a byte
     while(!NVRAM_SPI_1_SPIM_GetRxBufferSize());
      // Copy the read byte to data_read_ptr
      data_read_ptr[i] = NVRAM_SPI_1_SPIM_ReadRxData();
   }

   // Make chip select HIGH CS = 1
   NVRAM_SPI_1_CS_Reg_Write(1);
}
```

文档修订记录

文档标题：F-RAM™的 SPI 指南 — AN304

文档编号：001-92104

修订版	ECN	变更者	提交日期	变更说明
**	4341524	JCUI	04/11/2014	本文档版本号为 Rev**，译自英文版 001-87196 Rev**。
*A	4697058	LISZ	03/23/2015	本文档版本号为 Rev*A，译自英文版 001-87196 Rev*C。

全球销售和设计支持

赛普拉斯公司拥有一个由办事处、解决方案中心、厂商代表和经销商组成的全球性网络。如果想要查找离您最近的办事处，请访问赛普拉斯所在地。

产品

汽车级产品	cypress.com/go/automotive
时钟与缓冲器	cypress.com/go/clocks
接口	cypress.com/go/interface
照明和电源控制	cypress.com/go/powerpsoc cypress.com/go/plc
存储器	cypress.com/go/memory
PSoC	cypress.com/go/psoc
触摸感应	cypress.com/go/touch
USB 控制器	cypress.com/go/usb
无线/射频	cypress.com/go/wireless

PSoC®解决方案

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

赛普拉斯开发者社区

社区 | 论坛| 博客 | 视频 | 培训

技术支持

cypress.com/go/support

此处引用的所有商标或注册商标归其各自所有者所有。

赛普拉斯半导体
198 Champion Court
San Jose, CA 95134-1709

电话 ：408-943-2600
传真 ：408-943-4730
网站地址 ：www.cypress.com

©赛普拉斯半导体公司，2013-2015。此处，所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯不保证产品能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。
该源代码（软件和/或固件）均归赛普拉斯半导体公司（赛普拉斯）所有，并受全球专利法规（美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和/或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定的用途外，未经赛普拉斯明确的书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。
免责声明：赛普拉斯不针对此材料提供任何类型的明示或暗示保证，包括（但不限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不做出通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于可能发生运转异常和故障，并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。
产品使用可能受适用于赛普拉斯软件许可协议的限制。