

## SPI F-RAM 与 PSoC® 4 的连接指南

作者: Shivendra Singh

相关工程: 有

相关器件系列: SPI F-RAM – FM25XXX

软件版本: 3.0 组件包 7 或更高版本

相关应用笔记: AN304、AN79953、AN87352

AN89659 通过提供示例电路、时序图和伪代码介绍如何将串行外设接口 (SPI) F-RAM 同赛普拉斯的 PSoC® 4 (可编程片上系统) 器件相连。您还可以将本应用笔记作为参考设计指南使用, 以达到将 SPI F-RAM 连接至其他标准 SPI 控制器的目的。本笔记包括了一个相关的 PSoC 4 示例工程。

## 目录

SPI F-RAM 与 PSoC® 4 的连接指南.....	1
目录 .....	1
简介 .....	1
SPI F-RAM 接口 .....	2
SPI 工作模式 .....	3
使用 PSoC 4 实现 SPI 主设备 .....	3
使用 UDB 实现 SPI 主设备 .....	3
使用 SCB 实现 SPI 主设备 .....	3
使用 Bit Banging 实现 SPI 主设备配置 .....	5
SPI F-RAM 输入引脚配置 .....	5
SPI F-RAM 的工作电压 .....	6
SPI F-RAM 操作码 .....	6
SPI F-RAM 的地址 .....	7
存储器空间升级 .....	8
SPI F-RAM 操作示例 .....	8
状态寄存器操作 .....	8
F-RAM 写和读操作 .....	10
总结 .....	13
附录 A: PSoC 4 示例工程 .....	14
示例工程引脚分配 .....	14
将 SPI F-RAM 组件集成到工程中 .....	14
文档修订记录 .....	19
全球销售和设计支持 .....	20
产品 .....	20
PSoC® 解决方案 .....	20

## 简介

SPI F-RAM 是使用高级铁电工艺的串行非易失性存储器。铁电随机存储器 (F-RAM) 是非易失性 RAM, 它能够解决串行 EEPROM 和其他非易失性存储器遇到的复杂性、开销和系统级可靠性等问题。与串行 EEPROM 和闪存存储器不同, F-RAM 可以在总线速度下执行写操作而不会引起任何写延迟 (NoDelay™)。数据可以被直接写入到存储器阵列中, 然后立即开启新的一个总线周期, 而不需要轮询数据。F-RAM 产品还提供了一个高达  $10^{14}$  次的写入次数。与串行 EEPROM 和闪存存储器相比, 该次数被提高了好几个数量级。另外, 比串行 EEPROM 或闪存存储器相比, F-RAM 耗能更低。欲了解串行 F-RAM 与串行 EEPROM 的优势, 请参考应用笔记 AN87352 — 用于智能电子仪表的 F-RAM。

赛普拉斯 PSoC 是一个真正的可编程嵌入式片上系统, 它在单芯片上集成了可配置的模拟和数字外设功能、存储器和微控制器。PSoC 4 是基于 ARM Cortex-M0 的 PSoC 系列器件。更多 PSoC 4 的详细信息, 请查看 AN79953 — PSoC® 4 入门。

本应用笔记详细说明了连接至 PSoC 4 的 SPI F-RAM 的连接和性能。这里提出的硬件建议不是必要的, 但采用这些建议可使整个设计功能更强大。

为了说明 SPI F-RAM 在系统级的运行方式, 本应用笔记通过所提供时序图和基于 PSoC 4 的伪代码来介绍一些操作码。

本笔记还提供了一个在 PSoC Creator™ 环境中创建的相关工程 PSoC 4 SPI F-RAM 组件。附录 A 介绍如何在 PSoC Creator 中将 SPI F-RAM 组件集成到一个新的项目中。PSoC Creator 是赛普拉斯的一个强大的集成开发环境，用于 PSoC 3、PSoC 4 和 PSoC 5LP。

SPI F-RAM 的所有连接详情和建议可适用于与 SPI F-RAM 连接的其他 SPI 主控制器。

本应用笔记包含以下主题：

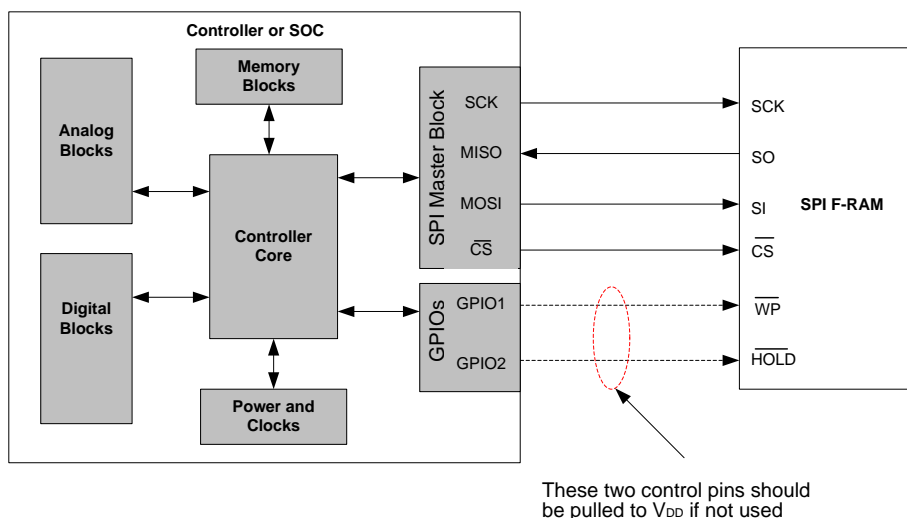
- SPI F-RAM 接口
- SPI 操作模式
- 使用 PSoC 4 实现 SPI 主设备
- SPI F-RAM 操作码
- SPI F-RAM 的地址
- SPI F-RAM 操作示例

## SPI F-RAM 接口

不管密度如何，SPI 主控制器（主设备）和 SPI F-RAM（从设备）之间的硬件连接均保持不变。但 SPI 器件所提供的密度选项和特性不同会引起固件发生变化。例如，密度为 1 Mb 和更高密度的 SPI F-RAM 要求 3 字节的地址，而密度为 512 Kb 和更低的 SPI F-RAM 仅需要 2 字节的地址。在这种情况下，固件的差异表现在包含 3 字节或 2 字节的地址，而硬件连接保持不变。同样，并非所有 SPI F-RAM 器件都具备序列号读取（SNR）特性。

图 1 展示了 SPI F-RAM 器件的典型系统级配置。

图 1. SPI F-RAM 与控制器的典型连接



## SPI 工作模式

标准的 SPI 支持 4 种不同的操作模式，它们由 SPI 时钟（SCK）极性（CPOL）和时钟相位（CPHA）决定。SPI 时钟（SCK）极性（CPOL）是 SPI 时钟（低电平或高电平）的基值，而时钟相位（CPHA）是 SPI 时钟沿（上升沿或下降沿）。执行 SPI 通信前，SPI 主设备将设置 SPI 模式。表 1 汇总了全部 4 种 SPI 模式以及通过主出从入（MOSI）和主入从出（MISO）线的 SPI 时钟、数据驱动和捕获沿。

表 1. SPI 工作模式

SPI 的数据驱动和捕获沿	模式 0 (CPOL=0; CPHA=0)	模式 1 (CPOL=0; CPHA=1)	模式 2 (CPOL=1; CPHA=0)	模式 3 (CPOL=1; CPHA=1)
SPI 时钟 (SCK) 启动逻辑电平	低电平	低电平	高电平	高电平
在 MOSI 线上被 F-RAM 锁存的数据	SCK 上升沿 (↑)	SCK 下降沿 (↓)	SCK 下降沿 (↓)	SCK 上升沿 (↑)
由 F-RAM 通过 MISO 线发送的数据	SCK 下降沿 (↓)	SCK 上升沿 (↑)	SCK 上升沿 (↑)	SCK 下降沿 (↓)
支持 SPI F-RAM	有	无	无	有

## 使用 PSoC 4 实现 SPI 主设备

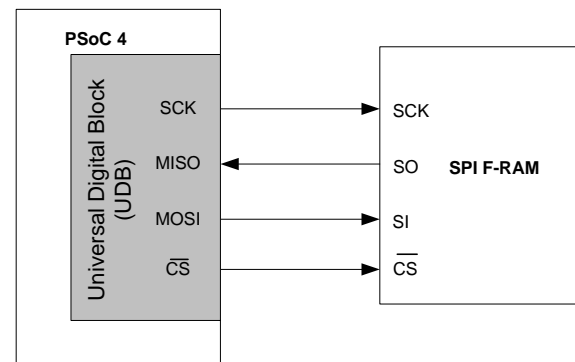
通过能够灵活自动布局的 PSoC 4 内的可编程和可重新配置数字块，用户可以选择 4 个通用 I/O（GPIO）引脚中的任何一个作为 SPI 接口使用。采取下面方法中的任意一个，PSoC 4 均能够实现 SPI 主设备：

### 使用 UDB 实现 SPI 主设备

可将 PSoC 4 上的通用数字块（UDB）配置成 SPI 主设备。由于所有 GPIO 都可以对 UDB 进行访问，所以可以将 PSoC 4 上的 4 个 GPIO 引脚中的任意一个作为 SPI 主设备控制引脚使用。更多 I/O 引脚配置和 UDB 的详细信息，请参阅 [PSoC 4 架构 TRM](#) 和器件数据手册。图 2 显示的是使用 UDB 实现的 PSoC 4 SPI 主设备及 SPI F-RAM 的连接。

本笔记所随附的示例工程使用 UDB 来实现 PSoC 4 上的 SPI 主设备。

图 2. 使用 PSoC 4 UDB 实现 SPI 主设备



### 使用 SCB 实现 SPI 主设备

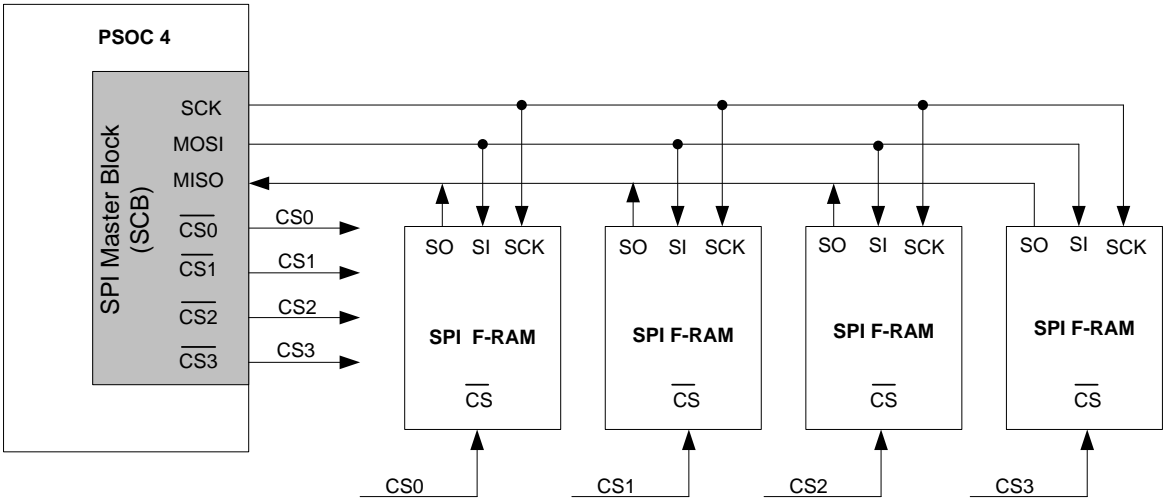
PSoC 4 配备了两个能够配置成 SPI 主设备的专用串行通信模块（SCB）。但每个 SCB 都分配了一组固定的 GPIO 给指定的控制引脚，但应注意，仅能使用这些引脚构建 PSoC 4 和外设器件之间的串行通信链接。表 2 显示的是每个 SCB 的专用 PSoC 4 SPI 控制引脚。

表 2. PSoC 4 SCB 内的 SPI 引脚配置

PSoC 4 引脚		SCB 模块	引脚功能	引脚说明
名称	类型			
P4.0	GPIO	SCB0	MOSI	SCB0 的主出从入线
P4.1	GPIO	SCB0	MISO	SCB0 的主入从出线
P4.2	GPIO	SCB0	SCK	SCB0 的 SP 时钟
P4.3	GPIO	SCB0	CS0	SCB0 的 SPI 从设备选择 0
P0.0	GPIO	SCB0	CS1	SCB0 的 SPI 从设备选择 1
P0.1	GPIO	SCB0	CS2	SCB0 的 SPI 从设备选择 2
P0.2	GPIO	SCB0	CS3	SCB0 的 SPI 从设备选择 3
P0.4/P3.0	GPIO	SCB1	MOSI	SCB1 的主出从入线
P0.5 / P3.1	GPIO	SCB1	MISO	SCB1 的主入从出线
P0.6 / P3.2	GPIO	SCB1	SCK	SCB1 的 SPI 时钟
P0.7 / P3.3	GPIO	SCB1	CS0	SCB1 的 SPI 从设备选择 0
P3.4	GPIO	SCB1	CS1	SCB1 的 SPI 从设备选择 1
P3.5	GPIO	SCB1	CS2	SCB1 的 SPI 从设备选择 2
P3.6	GPIO	SCB1	CS3	SCB1 的 SPI 从设备选择 3

每个 SCB 均能够实现一个 SPI 主设备模块。SPI 主设备可以与总线上一共连接的 4 个从设备 SPI 器件进行通信。从设备选择（CS）引脚用于选择连接至 SPI 总线的单独 SPI 从设备。图 3 介绍了使用 SCB 配置的 PSoC 4 SPI 和 SPI F-RAM 的连接。该图还介绍了如何在同一个 SPI 总线上连接到某个 SPI 从设备上。

图 3. 使用 SCB 实现 PSoC 4 上的 SPI 主设备



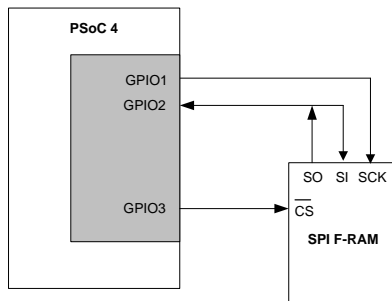
## 使用 Bit Banging 实现 SPI 主设备配置

第三种（同时也是最不推荐的）方法是使用 I/O bit-banging 实现 PSoc 4 上的 SPI 主设备。使用该方法时，将通过固件驱动 I/O 引脚来生成 SPI 协议。与使用 UDB 或 SCB 实现硬件相比，固件实现需要添加大量的代码开销。另外使用 SPI 接口的 bit-banging 方法还减少了数据吞吐量，并占用了更多 CPU 资源，这样会减少提供其他功能的 CPU 资源。

Bit-banging 方法的唯一优势是它仅需要 3 个 GPIO 引脚（而不是使用 4 个 GPIO 引脚）来实现 SPI 接口。这也被称为 SPI 接口的半双工模式。SPI 接口的半双工模式要求 PSoc 4 控制器在任意给定的时间内执行一个写操作或一个读操作。因此，与可以同时传输和接收数据的标准 SPI 接口不同，可以使用单一的 GPIO 来发送或接收数据。

图 4 显示的是使用 PSoc 4 上的 GPIO 引脚实现的 PSoc 4 和 SPI F-RAM 的连接。本应用笔记不讨论使用 bit-banging 方法实现 PSoc 4 内的 SPI 主控制器的话题。

图 4. 使用 PSoc 4 bit-banging 实现半双工 SPI 主设备。

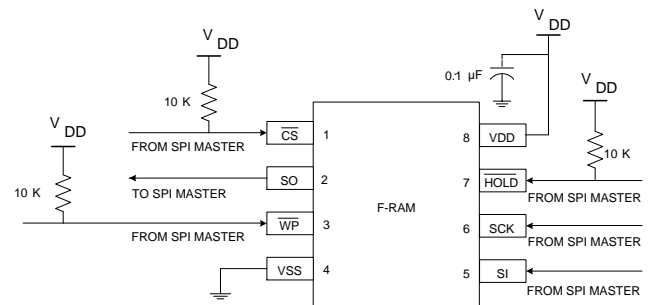


## SPI F-RAM 输入引脚配置

SPI F-RAM 配备几个控制引脚。应该正确将这些引脚偏置为固定的逻辑状态（高电平或低电平），以正确执行器件。如果没有正确偏置输入引脚，并保持为悬空，则它会假设中间逻辑状态导致高器件电流，或它可悬空至能够触发不良操作的逻辑低电平或高电平。悬空输入信号假设的逻辑状态方向取决于多个因素，如系统内的噪声、电容耦合和漏电压等。

因此，悬空输入电路观察到的逻辑电平比较是随机的，并且它在执行操作器件时会存在改变。这种出乎意料的输入电平会严重影响器件的操作。因此，应该始终将未用的输入引脚保持为合适的逻辑电平，例如，将有效输入低电平输入引脚保持高电平。此外，可以使用大小为 10 kΩ 的电阻上拉或下拉未用的输入引脚。

图 5. SPI F-RAM 连接至标准的 SPI 主设备



**HOLD 引脚：** HOLD 引脚是低电平输入有效的引脚。通过该引脚，可以挂起时钟中流，从而暂停正在进行的通信操作。如果该引脚为低电平，则器件将不对任何收到的时钟脉冲做出反应，并且通信被中断，另外数据很有可能被丢失或损坏。如果不使用该引脚，应该将它连接至大小为 10 kΩ 的上拉电阻，以避免发生不希望的事件。

**WP 引脚：** 写保护 ( $\overline{WP}$ ) 引脚是低电平有效的输入引脚。当外部将它拉低为有效低电平，可以阻止写入至状态寄存器的操作。WP 引脚的功能是由状态寄存器内的 WPEN 位决定的。

如果 WPEN 位被置为 ‘1’，则它将使能  $\overline{WP}$  引脚控制；如果它被置为 ‘0’，则  $\overline{WP}$  引脚将被禁用。应该将该引脚连接至大小为 10 kΩ 的上拉电阻，以避免发生不希望的事件。

**CS 引脚：** 正常操作期间，SPI 主设备驱动芯片选择 ( $\overline{CS}$ ) 引脚。CS 上的外部上拉是可选的。然而，当 SPI 主设备不通过此线驱动时，可以使用 10 kΩ 的上拉电阻将  $\overline{CS}$  引脚保持为高电平。

欲详细了解上电、断电和正常操作期间输入控制引脚的功能和运行方式，请参考器件数据手册。

## SPI F-RAM 的工作电压

SPI F-RAM 器件提供了一个宽广的工作电压 ( $V_{DD}$ ) 范围：2.0 V 至 5.5 V。但该范围并不适用于所有 F-RAM 器件。F-RAM 的最典型工作电压分别为 2.0 V 至 3.6 V、2.7 V 至 3.6 V 及 4.5 V 至 5.5 V。请参阅器件数据手册，了解器件的  $V_{DD}$  和输入引脚的允许电压，确保  $V_{DD}$  引脚和 I/O 引脚上的电压不超过数据手册建议的限定值。

## SPI F-RAM 操作码

所有 SPI 操作码、地址和数据均是 8 位的数据。执行所有操作时， $\overline{CS}$  始终为低电平。操作码、地址和数据均输入通过 SI 引脚传输；数据输出则通过 SO 引脚输出。通过这些操作码，可以控制器件。SPI F-RAM 支持所有读和写操作的行业标准操作码。唯一一个操作码被分配给 SPI F-RAM 中每一个特定操作。表 3 提供了操作码列表。表 4 说明了正确执行操作要求时每个操作码和相关字节。

表 3. SPI F-RAM 操作码

指令类别	指令名	说明	操作码	
			十六进制	二进制
状态寄存器控制指令	WREN	设置状态寄存器内的写使能锁存 (WEL) 位。	06H	0000 0110b
	WRDI	清除状态寄存器内的 WEL 位；这样可禁用所有写操作。	04H	0000 0100b
	RDSR	读取状态寄存器	05H	0000 0101b
	WRSR	写入状态寄存器	01H	0000 0001b
F-RAM 读取和写入指令	READ (16 Kb 和更高)	读取存储器数据	03H	0000 0011b
	READ (4Kb) <sup>(1)</sup>	读取存储器数据	03H or 0BH	0000 A011b
	FSTRD <sup>(2)</sup>	快速读存储器数据	0BH	0000 1011b
	WRITE	写入存储器数据	02H	0000 0010b
	WRITE (4Kb) <sup>(1)</sup>	写入存储器数据	02H or 0AH	0000 A010b
Sleep 指令	SLEEP	进入睡眠模式	B9H	1011 1001b
器件 ID 和序号指令	RDID	读取器件 ID	9FH	1001 1111b
	信噪比	读取序号	C3H	1100 0011b

**注意 1：**4-Kb SPI F-RAM 采用了单字节地址来执行写操作和读操作，使用最低有效字节 (LSB) (位 3) 中的第 4 位，可通过操作码字节发送地址位的最高有效字节 (MSB)

**注意 2：**SPI FSTRD 类似于 SPI READ 指令，不同的是，FSTRD 指令需要一个额外的虚拟地址周期。通过 FSTRD 指令，SPI F-RAM 可以替换支持 FSTRD 指令的 SPI 闪存存储器。在 SPI F-RAM 器件内，FSTRD 指令和 READ 指令的运行频率相同。更多关于 FSTRD 的详细信息，请参考 SPI F-RAM 数据手册。



表 4. SPI F-RAM 数据流

指令名	操作码	SI 线上的主设备传输	F-RAM 通过 SO 线传输	注释
WREN	06H	06H	—	该指令用于设置状态寄存器中的 WEL 位。执行该指令后 CS，WEL 位在芯片选择 ( $\overline{CS}$ ) 的上升沿上被设置。
WRDI	04H	04H	—	WRDI 指令用于清除状态寄存器内的 WEL 位（如果被设置）。
RDSR	05H	05H	01H、StatusReg_Data	RDSR 指令用于读取状态寄存器中的内容。
WRSR	01H	01H、01H、StatusReg_Data	—	写入状态寄存器前必须设置 WEL 位。 $\overline{CS}$ 变为高电平时，WEN 位被清除。
READ	03H	03H、Add1、Add2、Add3	Data1、Data2、Data3、.....、DataN	可将数据长度设置为 1 至 N 间的数值以执行读操作，其中，N 可以是任意一个整数值。开始执行 READ 指令后，F-RAM 内部地址将自动递增 1，并且器件会从递增的存储器地址准备好传输下一个数据字节。
FSTRD	0BH	0BH、Add1、Add2、Add3、Dummy_Byte	Data1、Data2、Data3,..., DataN	当内部计数器达到最大的可读地址时，它将转至启发地址 00H，并在新位置继续读取数据，但前提是器件处于读取模式，并且串行时钟可用。 $\overline{CS}$ 到高电平时，也将退出读取模式。 FSTRD 指令要求使用一个额外的虚拟地址周期。虚拟周期后，会通过 SO 线驱动有效的数据。
WRITE	02H	02H、Add1、Add2、Add3、Data1、Data2、Data3,..., DataN	—	发送 WRITE 操作码前，必须设置 WEL 位。可将数据长度设置为 1 至 N 间的数值以执行写操作，其中，N 可以是任意一个整数值。开始执行 WRITE 指令后，F-RAM 内部地址将自动加 1，并且器件会准备好接收下一个数据字节，以写入到递增后的存储器位置。 当 nvSRAM 内部计数器达到最大的可写地址时，它将转至启发地址，并通过覆盖先前写入的数据来继续写入数据。只要器件处于写入模式，并且串行时钟可用，就可以继续执行数据写入操作。 执行写操作时，由于存储器计数器会翻转，因此控制器固件必须观察数据覆盖问题。 $\overline{CS}$ 切换到高电平后，将结束写操作，并且 WEL 被清除。
SLEEP	B9H	B9H	—	当 $\overline{CS}$ 变为高电平时，器件将进入睡眠模式，此时器件将消耗睡眠模式电流 (IZZ)。WEL 位必须在启动 SLEEP 前被设置。 $\overline{CS}$ 变为高电平时，WEN 位将被清除。
RDID	9FH	9FH	Data1、Data2、Data3、..、Data9	RDID 指令用于读取器件 ID（9 个字节）。
信噪比	C3H	C3H	Data1、Data2、Data3、..、Data8	SNR 指令用于读取序号。（8 个字节）。

**注意：**密度为 1 Mb 或更高的 SPI F-RAM 使用的是 3 字节的地址；密度更低的 F-RAM（512 Kb 或更低（降到 16 Kb）使用的是 2 字节的地址。4 Kb F-RAM 仅使用 1 字节的地址。

## SPI F-RAM 的地址

执行字节传输时，SPI 主机控制器将逐字节与 SPI nvSRAM 通信，并始终在第一个时钟周期内传输最高有效位，在第八个时钟周期内传输最低有效位。这种方法适用于所有 SPI 通信，包括指令、地址和数据字节。

同样，当 SPI nvSRAM 在读取操作过程中传输数据字节时，它将始终先传输最高有效位，最后传输最低有效位。

图 6 显示的是一个示例，演示了在 SPI 主器件传输三个地址字节时，地址位通过 SPI MOSI 线传输。

最高有效字节的未用位是“无需关注”（don't care）位。F-RAM 将忽略该位但应该在固件中将未用的地址位设置为‘0’。这样，在同一个插座中移动到密度更高的器件时更容易更新固件。

图 7 显示了各种不同密度的 F-RAM 的地址方案。在地址中，A0 是最低有效位。

图 6. SPI F-RAM 中的地址位传输

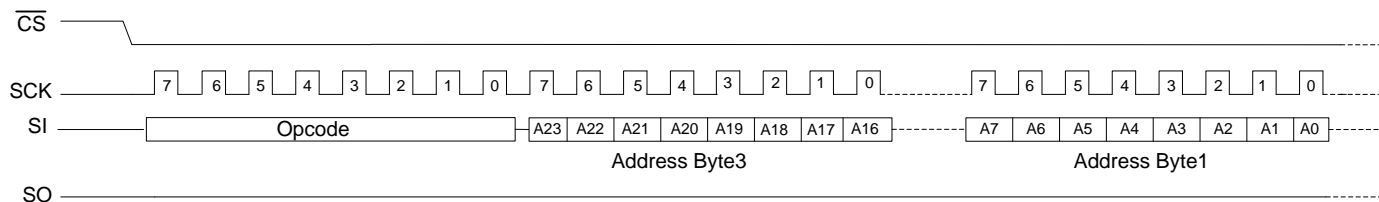


图 7. SPI F-RAM 操作和地址

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
4 Kbit <sup>[Note 3]</sup>	op	op	op	op	A	op	op	op	Not Applicable (1 Byte Addressing Only)										A7	A6	A5	A4	A3	A2	A1	A0						
16 Kbit	op	op	op	op	op	op	op	op	Not Applicable (2 Byte Addressing Only)							0	0	0	0	0	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
64 Kbit	op	op	op	op	op	op	op	op	Not Applicable (2 Byte Addressing Only)							0	0	0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
128 Kbit	op	op	op	op	op	op	op	op	Not Applicable (2 Byte Addressing Only)							0	0	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
256 Kbit	op	op	op	op	op	op	op	op	Not Applicable (2 Byte Addressing Only)							0	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
512 Kbit	op	op	op	op	op	op	op	op	Not Applicable (2 Byte Addressing Only)							A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
1 Mbit	op	op	op	op	op	op	op	op	0	0	0	0	0	0	0	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
2 Mbit	op	op	op	op	op	op	op	op	0	0	0	0	0	0	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
4 Mbit	op	op	op	op	op	op	op	op	0	0	0	0	0	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

注意：4 Kb SPI F-RAM 仅使用 1 字节的地址。最高有效地址位 A8（第 9 位）是 READ 和 WRITE 操作码的一部分。READ 操作码和 WRITE 操作码的位 3 可以作为存储器地址的 MSb 使用。

## 存储器空间升级

标准的 SPI 支持一主多从的拓扑结构。因此可以将多个 SPI 从设备连接至同一个 SPI 总线。需要与 SPI 从设备进行通信时，SPI 首先发送 SPI 指令，然后通过将其从设备选择（CS）引脚拉为低电平选择 SPI 从设备。如需支持多个 SPI 从设备，则主设备应具备一个专用的从设备选择控制引脚，以提供给每一个 SPI 从设备。图 3 显示的是多个 SPI 从设备连接至同一个 SPI 主设备。该拓扑结构用来控制多个 SPI 从设备，例如，使用多个 SPI 存储器来扩展系统存储器空间。

## SPI F-RAM 操作示例

通过提供时序图和 PSoC 4 特定伪代码，本节进一步说明了 F-SRAM 操作。前缀为 NVRAM\_SPI\_1 的所有函数均是 PSoC 4 的特定函数。NVRAM\_SPI\_1 是 PSoC Creator 示例项目中实例化的 SPI F-RAM 组件名称。

本部分仅介绍了一些操作码，目的是展示 SPI 主设备和 SPI F-RAM 之间的 SPI 通信过程中涉及的 SPI F-RAM 数据流。请参考 SPI F-RAM 数据手册，以获取更多操作码的详细信息。

## 状态寄存器操作

通过所提供的时序图和 PSoC 4 代码示例，本部分说明了写操作和读操作。

### 写入状态寄存器

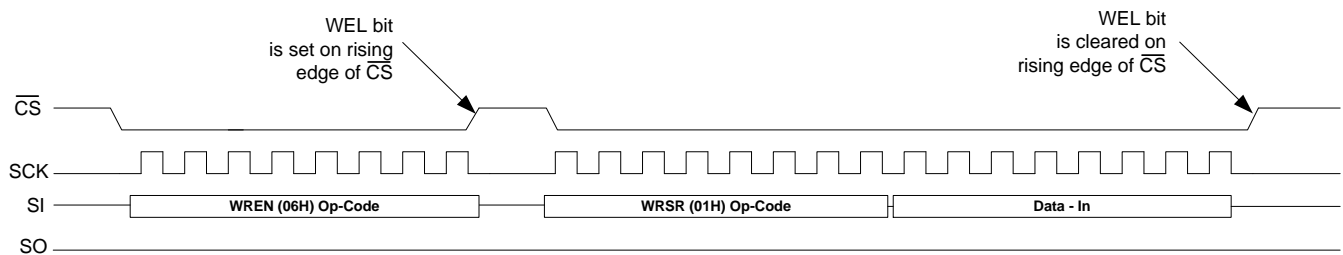
WRSR 指令用于设置状态寄存器内的用户可写位。其他无法写入的位由器件内部设置，或者被预留和返回固定值（‘0’或‘1’）。更多状态寄存器的详细信息，请参见器件数据手册。

执行 F-RAM 状态寄存器的写操作需要遵循下面的序列：

- 发送 WREN 操作码，从而设置写使能锁存（WEL）位。
- 发送写状态寄存器的操作码（WRSR）后，可以将数据字节写入到状态寄存器内。请注意，状态寄存器中的只读位不受 WRSR 操作的影响。有关使用情况的详细信息，请参见 TRM 或器件数据手册。
- 图 8 显示的是写入至状态寄存器的时序图。



图 8. 写入至状态寄存器 (WRSR 操作码)



```

/*****PSoc 4 pseudocode for Status Register write*****/
void NVRAM_SPI_1_Status_Reg_Write ( uint8 data_byte )
{
    NVRAM_SPI_1_CS_Reg_Write(0); // Enable the SPI slave by toggling chip select LOW
    NVRAM_SPI_1_SPIM_ClearTxBuffer(); //Clear SPI transmit buffer before sending command
    NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_WREN); // Set the write enable (WEL) bit prior to write

    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) // Wait till SPI_DONE flag is
        != NVRAM_SPI_1_SPIM_STS_SPI_DONE); // cleared

    NVRAM_SPI_1_CS_Reg_Write(1); // WEL is set high when CS is switched high
    NVRAM_SPI_1_CS_Reg_Write(0); // Re-enable the SPI slave
    NVRAM_SPI_1_SPIM_ClearTxBuffer();
    NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_WRSR_CMD); //Send Write Status Register instruction
    NVRAM_SPI_1_SPIM_WriteTxData(data_byte); //Send data

    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE)
        != NVRAM_SPI_1_SPIM_STS_SPI_DONE);
    NVRAM_SPI_1_CS_Reg_Write(1); // Terminate the write operation by toggling chip select HIGH
}

```

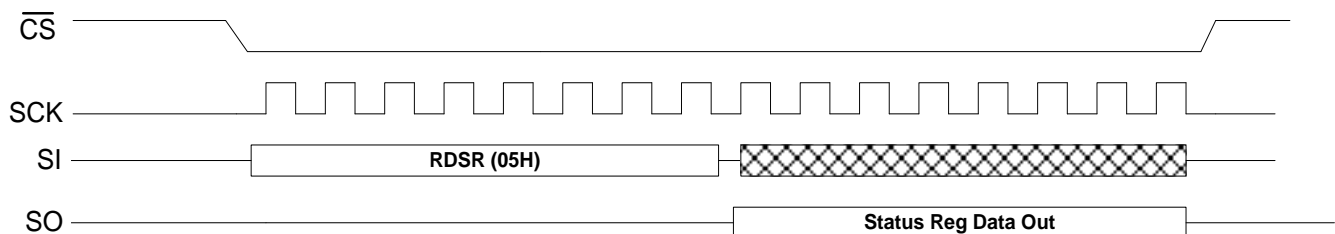
### 读取状态寄存器

RDSR 指令用于读取状态寄存器中的当前值。例如，WREN 指令后，执行 RDSR 指令会返回 WEL 位设置为“1”的状态寄存器值。

执行从 F-RAM 状态寄存器读取的操作需要遵循下面的序列：

- 发送 RDSR 操作码
- 读取 SO 线上的状态寄存器值。
- 通过将芯片选择切到高电平，主机可以停止状态寄存器指令。图 9 显示的是读取 SPI F-RAM 状态寄存器的时序图。

图 9. 读取状态寄存器 (RDSR 操作码)



```

/***** PSoC 4 pseudocode for Status Register read *****/

uint8 NVRAM_SPI_1_Status_Reg_Read ( void )
{
    uint8 data_byte;

    NVRAM_SPI_1_CS_Reg_Write(0); // Enable the SPI slave by toggling chip select LOW
    NVRAM_SPI_1_SPIM_ClearTxBuffer(); //Clear SPI transmit buffer before sending command
    NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_RDSR_CMD); //Send read status register command

    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) // Wait till SPI_DONE flag is
        != NVRAM_SPI_1_SPIM_STS_SPI_DONE); // cleared

    NVRAM_SPI_1_SPIM_ClearRxBuffer();
    NVRAM_SPI_1_SPIM_WriteTxData(0x00); //Dummy write for reading the status register data byte

    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE)
        != NVRAM_SPI_1_SPIM_STS_SPI_DONE);
    while(!NVRAM_SPI_1_SPIM_GetRxBufferSize()); //Wait until there is data in the read buffer
    data_byte = NVRAM_SPI_1_SPIM_ReadRxData();
    NVRAM_SPI_1_CS_Reg_Write(1); // Terminate the read operation by toggling chip select HIGH

    return(data_byte);
}

```

## F-RAM 写和读操作

通过所提供的时序图和 PSoC 4 代码的示例，本部分说明了 F-RAM 的写操作和读操作。

### F-RAM 写操作

执行写入 F-RAM 的操作需要遵循下面的序列：

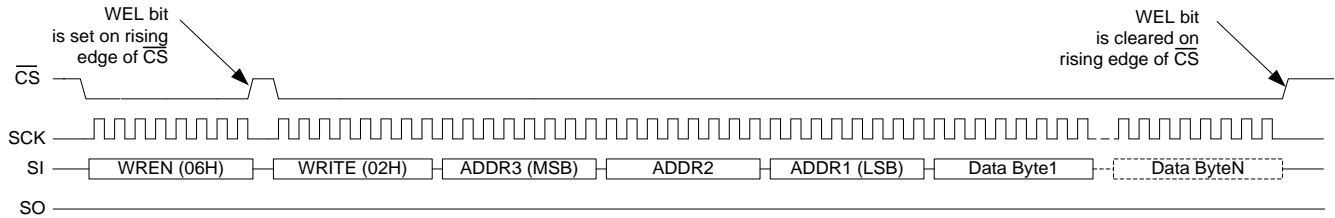
- 发送 WREN 操作码，从而设置写使能锁存（WEL）位。
- 发送 WRITE 操作码
- 发送最高有效地址字节。
- 发送（3 字节地址内的）中间地址字节
- 发送最高有效地址字节。
- 发送（各）数据字节

必须在发送所有 F-RAM 写指令前发送写使能（WREN）指令。如果未启用器件的写入功能（WEL = ‘0’），它将忽略写指令。这时，需要在新的  $\overline{\text{CS}}$  下降沿上重新启动 SPI 通信。

完成写指令（WRSR 或 WRITE）后，状态寄存器的 WEL 位将在芯片选择（ $\overline{\text{CS}}$ ）的上升沿上被清除为 ‘0’。这样可以确保 SPI F-RAM 已退出写入模式，从而防止发生意外的写入操作。

请注意，读取状态寄存器（RDSR 操作码）不是说要清除 WEL 位。一旦完成执行 WREN，一些用户读取状态寄存器来确认启动写操作前 WEL 位是否被置位。图 10 显示的是写入 SRAM 存储器的时序图。

图 10. 写入至 F-RAM (WRITE 操作码)



/\* PSoc 4 pseudocode for 1 Mb (128kx8) F-RAM write in burst mode. By passing in total\_data\_count =1, the user can write 1 byte at a given address location\*/

```
void NVRAM_SPI_1_Write ( uint32 addr, uint8 *data_write_ptr, uint32 total_data_count )
{
    uint32 i;

    NVRAM_SPI_1_CS_Reg_Write(0); // Enable the SPI slave by toggling chip select LOW
    NVRAM_SPI_1_SPIM_ClearTxBuffer(); // Clear SPI transmit buffer
    NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_WREN); // Set the write enable (WEN) bit prior to write

    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE) // Wait till SPI_DONE flag is
        != NVRAM_SPI_1_SPIM_STS_SPI_DONE); // cleared

    NVRAM_SPI_1_CS_Reg_Write(1); // WEL is set here
    NVRAM_SPI_1_CS_Reg_Write(0);
    NVRAM_SPI_1_SPIM_ClearTxBuffer();
    NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_SRAM_WRITE_CMD); //Send memory write command

    if(NVRAM_SPI_1_spi_density >= SPI_1MBit)
    {
        NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>16)); //Transmits most significant address byte (1 Mb and above)
    }
    NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>8)); // Transmits intermediate address byte in 3 byte addressing,
        //or, Transmits most significant address byte in 2 byte addressing
    NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr)); // Transmits least significant address byte in 2/3 byte addressing,
    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE)
        !=NVRAM_SPI_1_SPIM_STS_SPI_DONE);

    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);
    }
    NVRAM_SPI_1_CS_Reg_Write(1); // Terminate the write operation by toggling chip select HIGH
}
```

## F-RAM 读操作

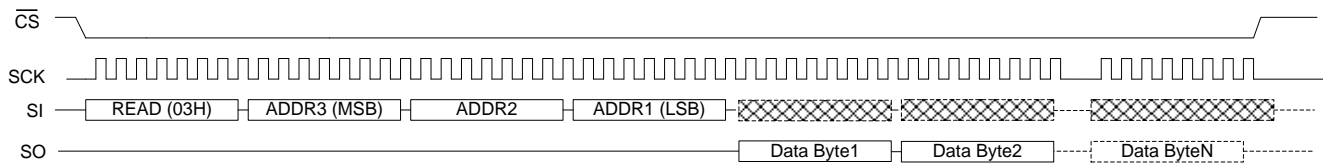
执行从 F-RAM 读取的操作需要遵循下面序列:

- 发送 READ 操作码
- 地址周期期间, 需要先发送最高有效的地址字节, 然后发送最低有效的地址字节。
- 一旦 SPI F-RAM 器件收到 READ 指令, 它将启动通过 SO 线发送数据字节。主控制器可以启动一个单字节读取或一个突发读取 (一个字节以上) 操作。

通过启动单一的读取指令, 突发读取操作可允许读取下一个存储器的位置。执行突发读取时, F-RAM 器件将自动递增内部地址计数器, 并且继续通过 SO 线发送数据字节。只要芯片选择CS保持低电平, 并且存在 SPI 时钟, 则该操作将持续进行。突发读取操作将以循环方式通过存储区连续循环。

当芯片选择CS被取消激活, 则数据输出将被停止, 并且 SO 也变为高阻态状态。图 11 显示的是读取 F-RAM 的时序图。

图 11. 读取 F-RAM (READ 操作码)



```
/* PSoc 4 pseudocode for 1 Mb (128kx8) F-RAM Read in burst mode. By passing in total_data_count =1, user can read only 1 byte from a given address location*/
```

```
void NVRAM_SPI_1_Read ( uint32 addr, uint8 *data_read_ptr, uint32 total_data_count )
{
    uint32 i;

    NVRAM_SPI_1_CS_Reg_Write(0); // Enable the SPI slave by toggling chip select LOW
    NVRAM_SPI_1_SPIM_ClearTxBuffer(); // Clear SPI transmit buffer
    NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_SRAM_READ_CMD); // Send memory read command

    if(NVRAM_SPI_1_spi_density >= SPI_1MBit)
    {
        NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>16)); //Transmits most significant address byte (1 Mb and above)
    }
    NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>8)); // Transmits intermediate address byte in 3 byte addressing,
                                                    //or, Transmits most significant address byte in 2 byte addressing
    NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr)); // Transmits least significant address byte in 2/3 byte addressing,

    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE)
           !=NVRAM_SPI_1_SPIM_STS_SPI_DONE);

    for(i = 0; i < total_data_count; i++)
    {
        NVRAM_SPI_1_SPIM_ClearRxBuffer();
        NVRAM_SPI_1_SPIM_WriteTxData((uint8) 0x00); //dummy write for reading the F-RAM memory data byte
        while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE)
               !=NVRAM_SPI_1_SPIM_STS_SPI_DONE);

        while(!NVRAM_SPI_1_SPIM_GetRxBufferSize());
        data_read_ptr[i] = NVRAM_SPI_1_SPIM_ReadRxData();
    }
    NVRAM_SPI_1_CS_Reg_Write(1); // Terminate the read operation by toggling chip select HIGH
}
```

## F-RAM 快速读取操作

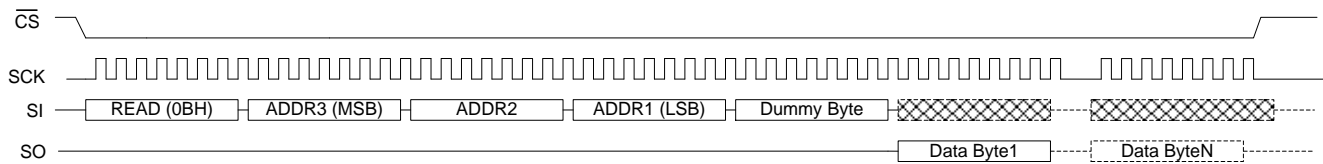
执行快速读取 F-RAM 操作需要遵循下面的序列:

- 发送 FSTRD 操作码
- 地址周期期间, 需要先发送最高有效地址字节, 然后发送最低有效地址字节。
- 发送一个虚拟地址字节。
- 一旦 SPI F-RAM 器件收到 FSTRD 指令, 它将启动通过 SO 线发送数据字节。主控制器可以启动一个单字节的读取或一个突发读取 (一个字节以上) 操作。

通过启动单一的读取指令, 突发读取操作可允许读取下一个存储器的位置。执行突发读取时, F-RAM 器件将自动递增内部地址计数器, 并且继续通过 SO 线发送数据字节。只要芯片选择  $\overline{CS}$  保持为低电平, 并且存在 SPI 时钟, 则该操作将持续进行。突发读取操作将以循环方式通过存储区连续循环。

当芯片选择  $\overline{CS}$  被取消激活, 则数据输出将被停止, 并且 SO 也变为高阻状态。图 12 显示的是使用快速读取 (FSTRD) 指令读取 F-RAM 的时序图。

图 12. 读取 F-RAM (FSTRD 操作码)



```

/* PSoC 4 pseudocode for 1 Mb (128kx8) F-RAM fast read in burst mode. By passing in total_data_count =1, the user
can read only 1 byte from a given address location*/

void NVRAM_SPI_1_FastRead ( uint32 addr, uint8 *data_read_ptr, uint32 total_data_count )
{
    uint32 i;

    NVRAM_SPI_1_CS_Reg_Write(0); // Enable the SPI slave by toggling chip select LOW
    NVRAM_SPI_1_SPIM_ClearTxBuffer(); // Clear SPI transmit buffer
    NVRAM_SPI_1_SPIM_WriteTxData(NVRAM_SRAM_READ_CMD); // Send memory read command

    if(NVRAM_SPI_1_spi_density >= SPI_1MBit)
    {
        NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>16)); //Transmits most significant address byte (1 Mb and above)
    }
    NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr>>8)); // Transmits intermediate address byte in 3 byte addressing,
    //or, Transmits most significant address byte in 2 byte addressing
    NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr)); // Transmits least significant address byte in 2/3 byte addressing,

    NVRAM_SPI_1_SPIM_WriteTxData((uint8)(addr)); // Transmits least significant address byte as dummy byte for fast
    // read operation

    while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE)
    !=NVRAM_SPI_1_SPIM_STS_SPI_DONE);
    for(i = 0; i < total_data_count; i++)
    {
        NVRAM_SPI_1_SPIM_ClearRxBuffer();
        NVRAM_SPI_1_SPIM_WriteTxData((uint8) 0x00); //dummy write for reading the F-RAM memory data byte
        while((NVRAM_SPI_1_SPIM_ReadTxStatus() & NVRAM_SPI_1_SPIM_STS_SPI_DONE)
        !=NVRAM_SPI_1_SPIM_STS_SPI_DONE);
        while(!NVRAM_SPI_1_SPIM_GetRxBufferSize());
        data_read_ptr[i] = NVRAM_SPI_1_SPIM_ReadRxData();
    }
    NVRAM_SPI_1_CS_Reg_Write(1); // Terminate the read operation by toggling chip select HIGH
}

```

## 总结

与其他非易失性 SPI 存储器产品相似（如 SPI EEPROM、闪存和 MRAM），赛普拉斯的 SPI F-RAM 也支持行业标准 SPI 访问协议。这样，F-RAM 能够与所有标准 SPI 主控制器相兼容。由于 SPI F-RAM 操作码与标准 SPI 存储器产品兼容，因此很容易使用 SPI F-RAM 进行替换。本应用笔记通过使用原理图、时序图和示例代码介绍了如何将 SPI F-RAM 连接至赛普拉斯 PSoC 4 控制器。

## 附录 A: PSoC 4 示例工程

本应用笔记所附带的工程是介绍 SPI F-RAM 与 PSoC 4 器件的连接示例工程。您可以从本应用笔记提供的链接下载名为 *AN89659.zip* 的工程文件。根据目标应用，用户可以利用 PSoC 4 的可编程性和灵活性等优势来添加更多功能，并修改工程。需要在硬件设计中执行示例工程，用户要备有连接至 SPI F-RAM 的 PSoC 4。

### 示例工程引脚分配

表 5 显示的是示例工程中的连接详细信息。本示例使用了以下软件和硬件组件：

- PSoC Creator 3.0 组件包 7。
- 硬件套件 — CY8CKIT-042。
- 用于构建工程的 PSoC 4 部件编号 CY8C4245AXI-483。
- 通过 USB 给 PSoC 4 控制器供电的  $V_{DD}$  电源。

表 5. 示例工程中的 PSoC 4 端口配置。

F-RAM 信号名称	PSoC (主器件) 信号名称	PSoC 4 I/O 分配	信号方向
$\overline{CS}$	CS	P2[0]	PSoC 4 输出引脚
SI	MOSI	P2[3]	PSoC 4 输出引脚
SO	MISO	P1[4]	PSoC 4 输入引脚
SCK	SCK	P2[2]	PSoC 4 输出引脚
$\overline{HOLD}$	HOLD	P2[1]	PSoC 4 输出引脚

示例工程将执行下面的操作：

1. 将 4 字节的数据写入至存储器内，并读回 4 字节的写入数据。数据将存储在内部寄存器中。
2. 将 1 字节的数据写入到值状态寄存器中，然后读回该数据。读取数据字节存储在内部寄存器中。

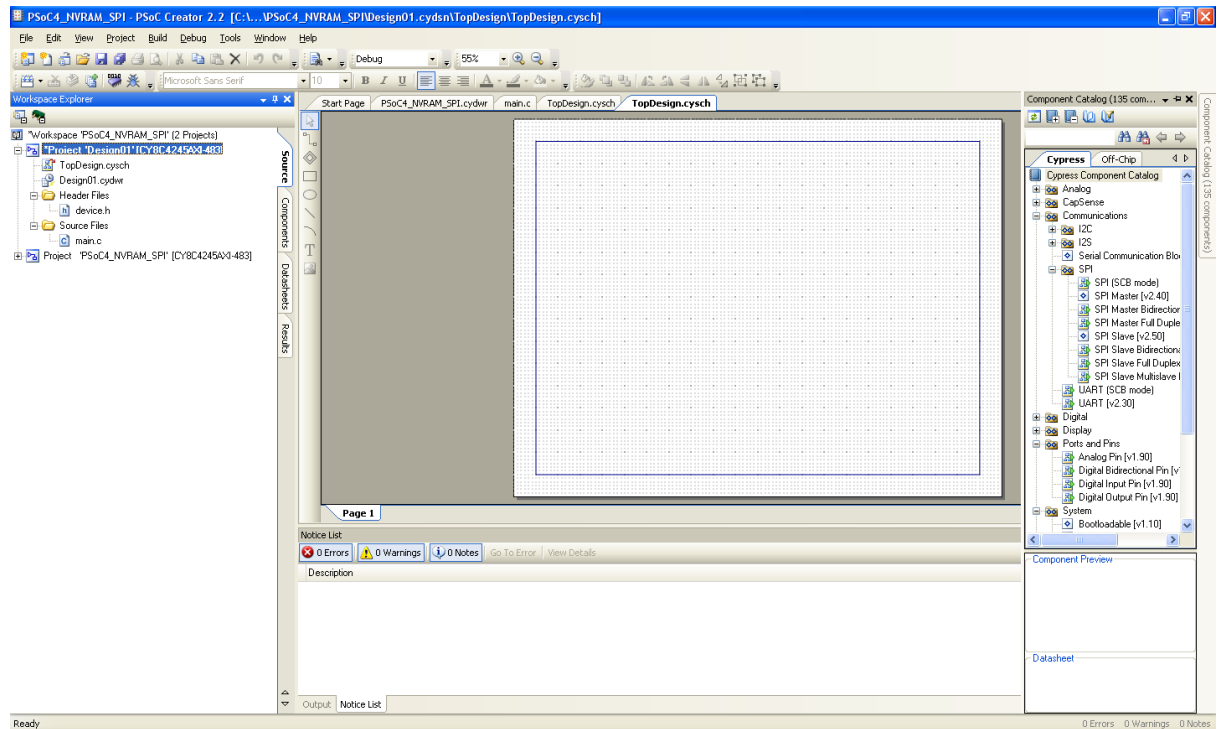
### 将 SPI F-RAM 组件集成到工程中

本部分描述了如何将 F-RAM 组件集成到新的 PSoC 4 工程中。将压缩的 *AN89659.zip* 文件解压为名字为 “PSoC4\_NVRAM\_SPI,” 的新文件夹。该文件夹包含了示例工程和 NVRAM\_SPI 组件。下面各步骤介绍的是如何将 NVRAM\_SPI 组件集成到新的 PSoC 4 设计，并介绍了使用方法。

1. 打开 PSoC Creator，然后依次选择 **File > New > Project**，以创建新的工程。选择 **Empty Template**（空模板），然后选择 **Empty PSoC 4 Design**（空 PSoC 4 设计）项。请在 **Name**（名称）字段上输入工程的名称。创建了名为 “Design01” (workspace) 的新工程，如图 13 所示。

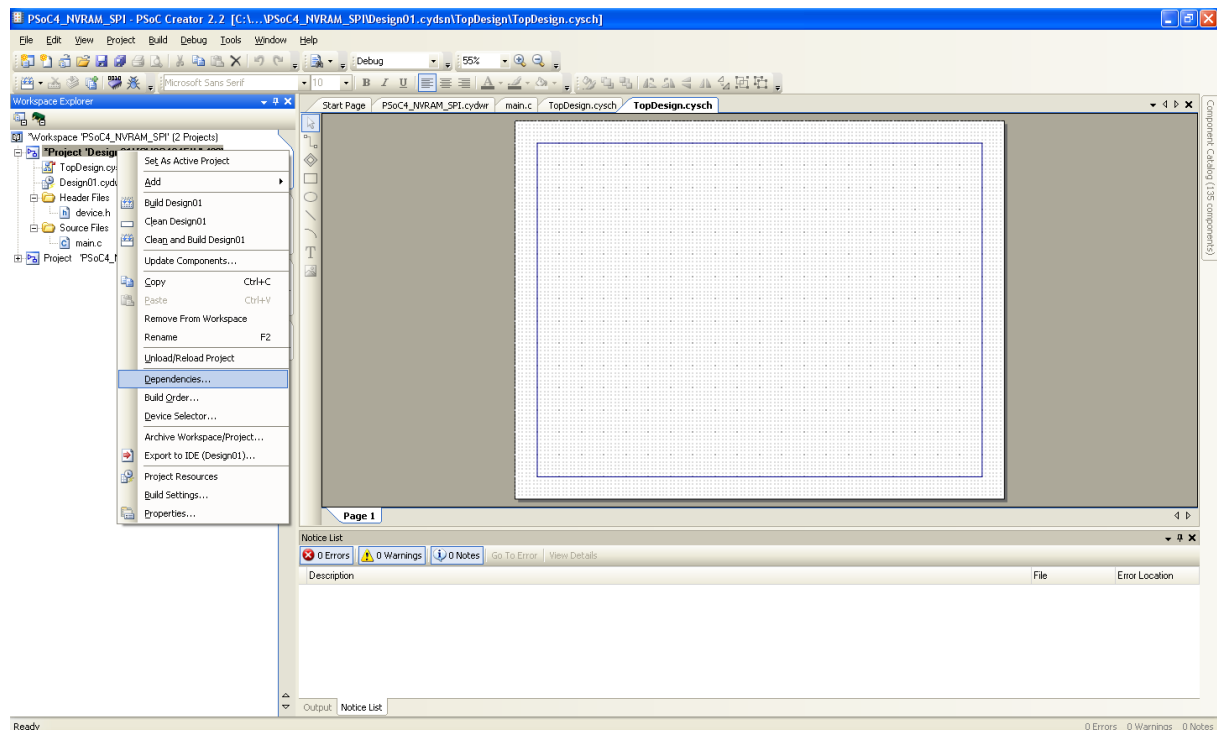


图 13. 创建工程 “Design01”



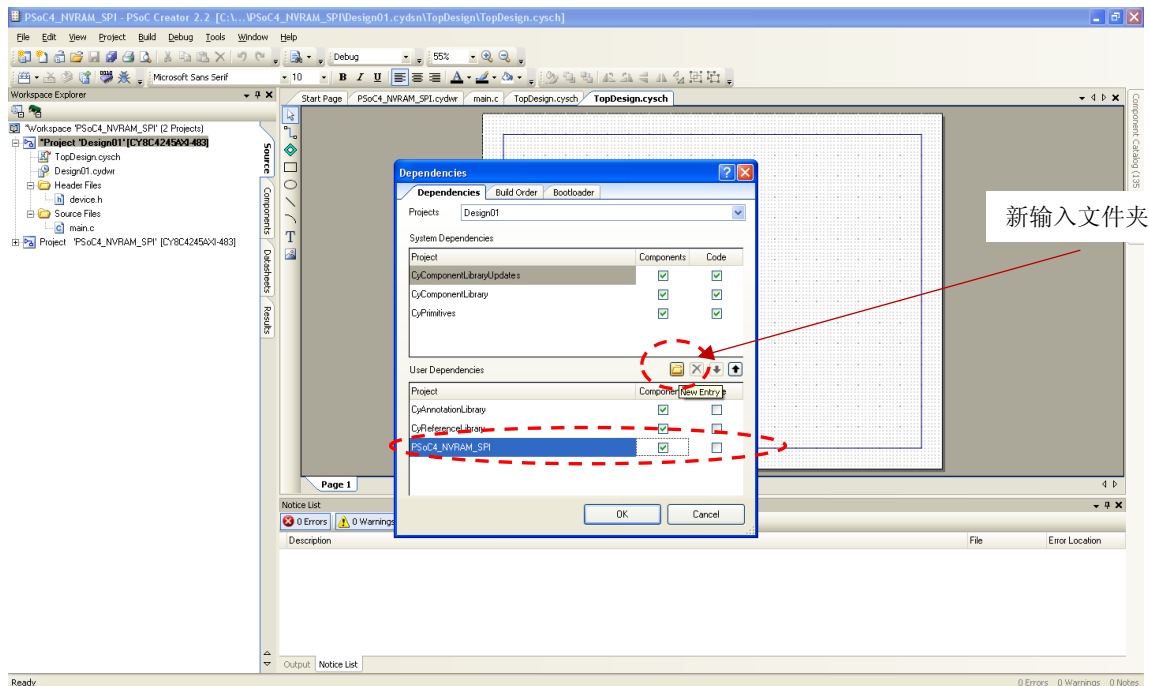
- 在 **Workspace Explorer**（工作区管理器）选项卡上，右击工程，然后选择 **Dependencies** 选项。将 NVRAM\_SPI 组件集成到您的设计中，如图 14 所示。

图 14. 打开 Dependencies



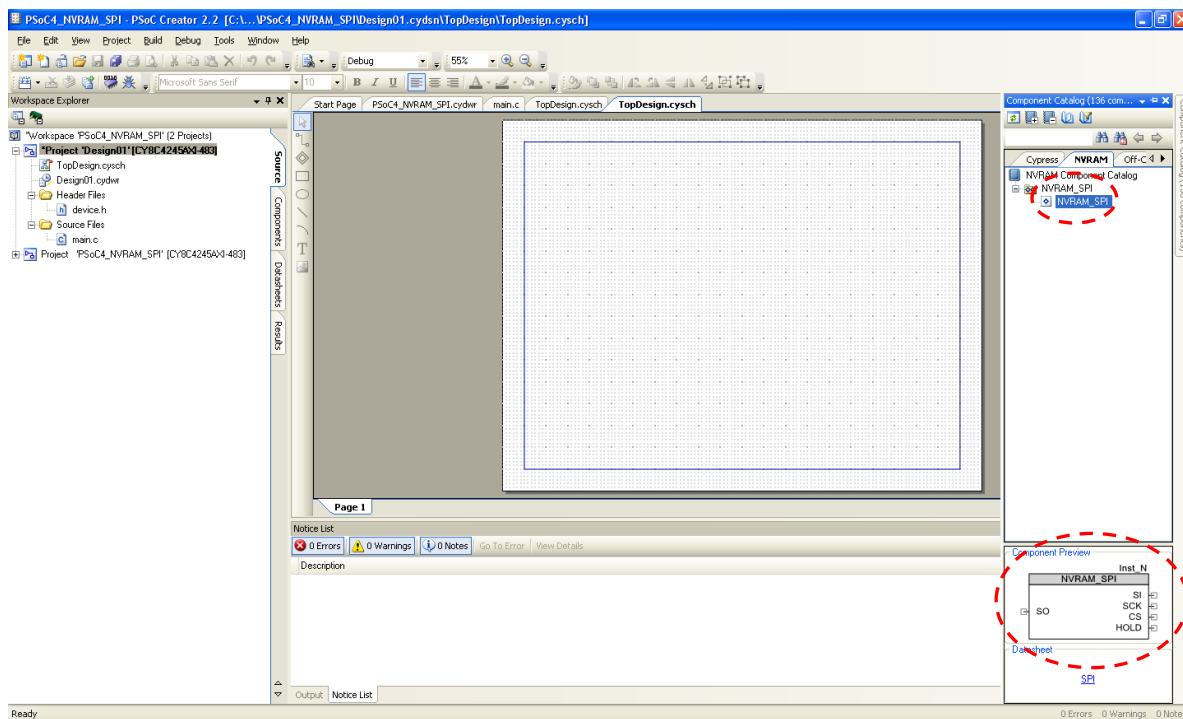
3. 点击 **New Entry (User Dependencies)**，并从 *PSoc4\_NVRAM\_SPI.cydsn* 文件夹中选择 *PSoc4\_NVRAM\_SPI.cypri*，如图 15 所示。

图 15. 添加 Dependencies



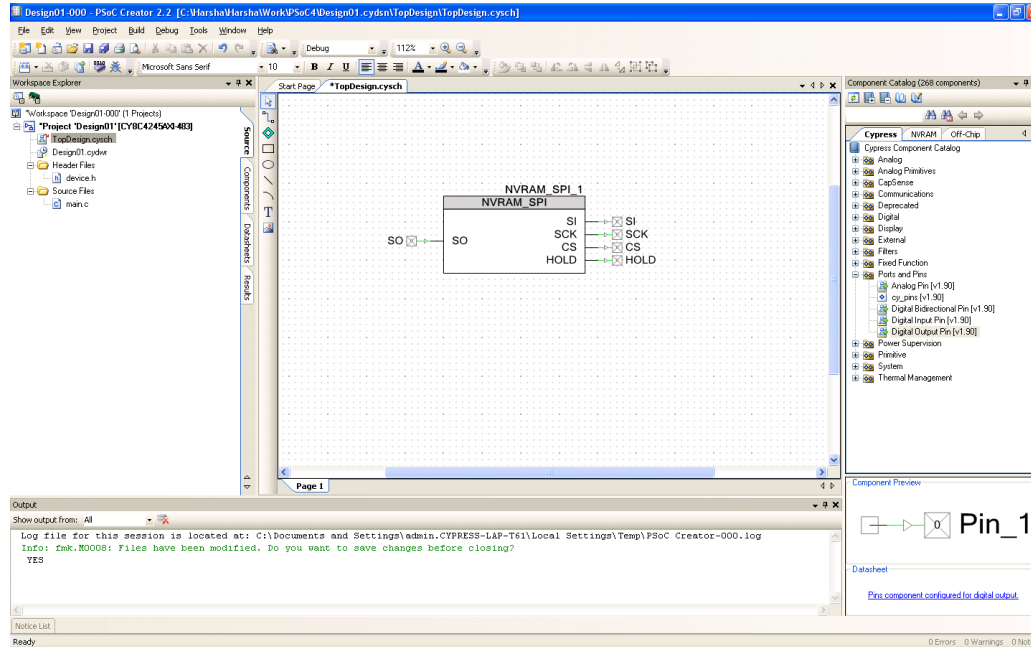
4. NVRAM\_SPI 组件显示在 **NVRAM** 选项卡上和 NVRAM\_Component Catalog 上，如图 16 所示。

图 16. 目录中的 NVRAM\_SPI 组件



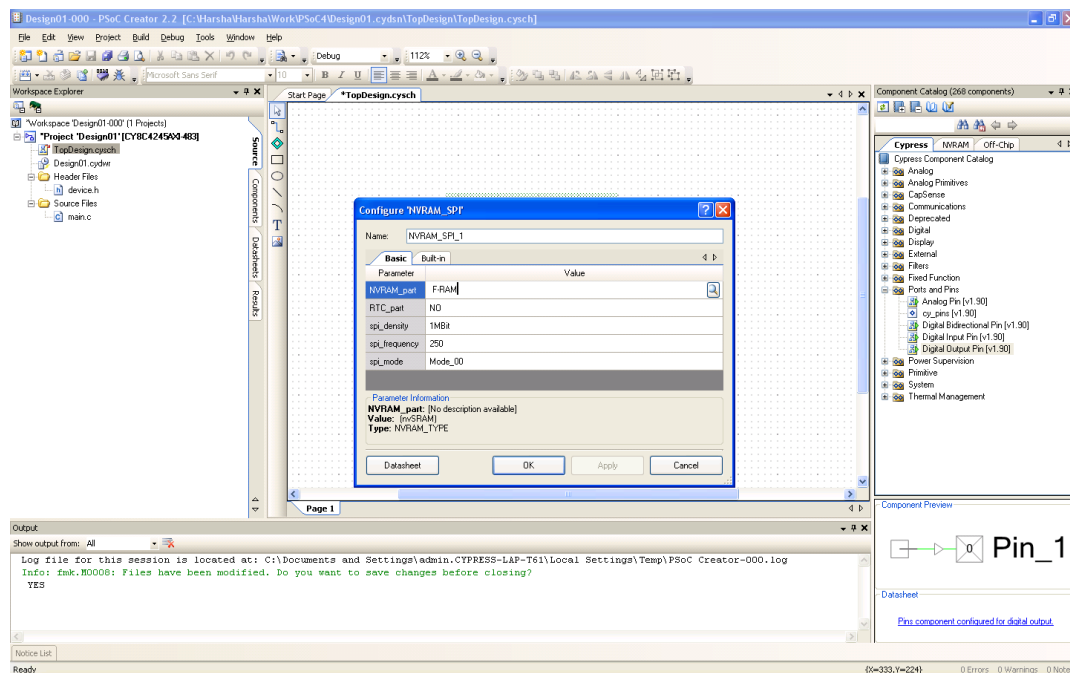
- 将 NVRAM\_SPI 组件拖放到 *TopDesign.cysch* 内，并分配数字 I/O。将组件的输入/输出引脚连接至 PSoC 4 上适当的端口引脚。表 5 指出了示例工程中各引脚的分配情况。

图 17. 使用 NVRAM\_SPI 组件创建设计原理图



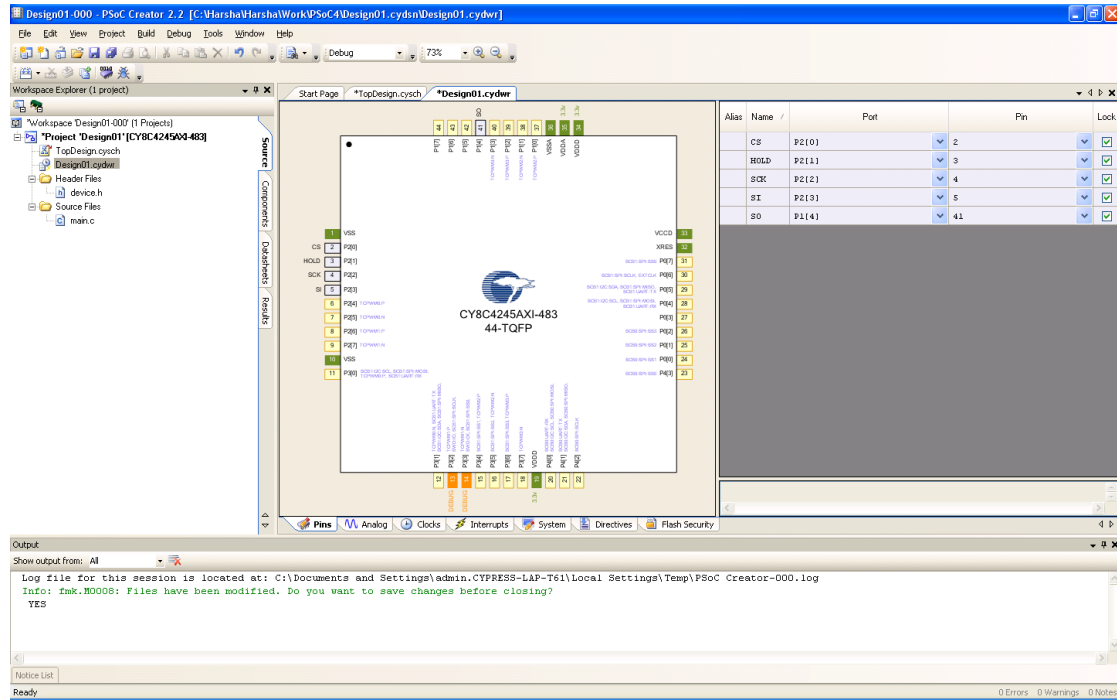
- 双击 NVRAM\_SPI 组件，然后按照以下顺序配置组件参数：将 NVRAM\_Part 选为 F-RAM，RTC 选为 NO，并选择用于应用中的每个 F-RAM 密度的 SPI 密度。然后，根据应用的要求进行设置 SPI 频率和模式（模式 0 或者模式 3），如图 18 所示。

图 18. 配置 NVRAM\_SPI 组件参数

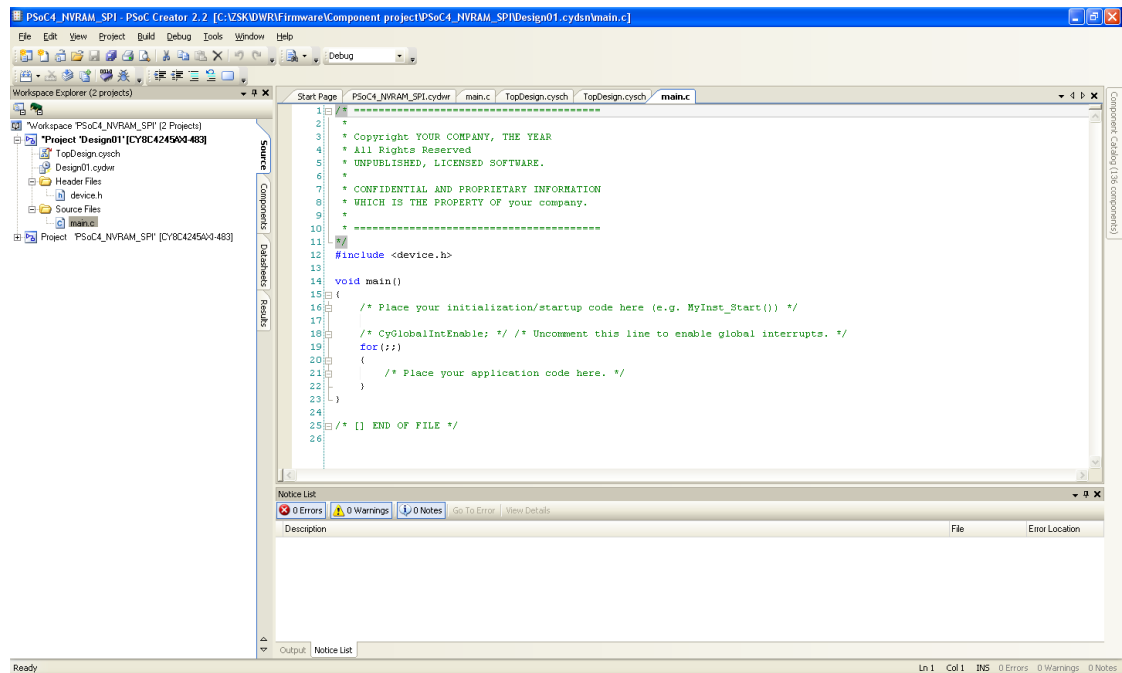


- 在您的设计中，分配适当的输入/输出引脚，如图 19 所示，然后创建工程。

图 19. 分配 PSoC 4 引脚



在 `main.c` 文件中进行开发您的代码。您可以将 API 直接调用至您的程序，并执行 F-RAM 功能。

图 20. 工程中的 `main.c` 文件


更多 PSoC 4 和 PSoC Creator 的信息，请参阅 [AN79953 —PSoC® 4 入门](#)和相关链接。

## 文档修订记录

文档标题: SPI F-RAM 和 PSoC® 4 的连接指南 – AN89659

文档编号: 001-92144

修订版	ECN	原始变更	提交日期	变更说明
**	4345974	LISZ	05/22/2014	本文档版本号为 Rev**, 译自英文版 001-89659 Rev**。

## 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、工厂代表和经销商组成的全球性网络。要找到离您最近的办事处，请访问[赛普拉斯所在地](#)。

## 产品

汽车	<a href="http://cypress.com/go/automotive">cypress.com/go/automotive</a>
时钟与缓冲区	<a href="http://cypress.com/go/clocks">cypress.com/go/clocks</a>
接口	<a href="http://cypress.com/go/interface">cypress.com/go/interface</a>
照明和电源控制	<a href="http://cypress.com/go/powerpsoc">cypress.com/go/powerpsoc</a> <a href="http://cypress.com/go/plc">cypress.com/go/plc</a>
存储器	<a href="http://cypress.com/go/memory">cypress.com/go/memory</a>
PSoC	<a href="http://cypress.com/go/psoc">cypress.com/go/psoc</a>
触摸感应	<a href="http://cypress.com/go/touch">cypress.com/go/touch</a>
USB 控制器	<a href="http://cypress.com/go/usb">cypress.com/go/usb</a>
无线/射频	<a href="http://cypress.com/go/wireless">cypress.com/go/wireless</a>

## PSoc®解决方案

[psoc.cypress.com/solutions](http://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](http://cypress.com/go/support)

PSoc 是赛普拉斯半导体公司的注册商标且 PSoc Creator 是赛普拉斯半导体公司的商标。此处引用的所有其他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 : [www.cypress.com](http://www.cypress.com)

©赛普拉斯半导体公司，2014。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，否则赛普拉斯产品不保证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，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准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，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。除上述指定用途外，未经赛普拉斯的明确书面许可，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：赛普拉斯不针对该材料提供任何类型的明示或暗示保证，包括（但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不另行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。对于合理预计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的赛普拉斯软件许可协议限制。