AN64574

使用串行外设接口（SPI） nvSRAM 进行设计

作者： Shivendra Singh

相关项目：有

相关器件系列：CY14C101PA/QxA、CY14B101PA/QxA、CY14E101PA/QxA

软件版本：PSoC® Creator™ — 3.0 或更高版本
PSoC Designer™ — 5.2 或更高版本

相关应用笔记：无

赛普拉斯串行外设接口（SPI） nvSRAM 是高性能的非易失性串行存储器，它的写操作周期延迟为 0，并且它能够提供无限次数的 SRAM 写耐久性。SPI nvSRAM 是一个 SPI 从设备，在系统中要通过 SPI 主控制器访问 nvSRAM。本应用笔记提供了一些关键的设计注意事项和固件提示，这样是为了引导用户使用 SPI nvSRAM 进行设计。另外，还提供了一个示例项目，包括 PSoC 1 的相关项目和 PSoC 3 和 PSoC 4 的库组件，这样可以演示标准 SPI 主控制器访问 SPI nvSRAM 的情况。

目录

简介 1
SPI nvSRAM 配置 2
输入引脚配置 4
RTC 引脚配置 5
SPI 工作模式 5
SPI nvSRAM 操作码 6
在 SPI nvSRAM 中进行寻址 8
nvSRAM 操作 9
状态寄存器的操作 9
nvSRAM 中的 SRAM 写/读操作 10
示例项目设置 12
总结 13
全球销售和设计支持 15

简介

赛普拉斯 nvSRAM 将一个快速 SRAM 单元和一个非易失性单元集成为单一的 nvSRAM 单元。在正常运行模式下，可以直接对 nvSRAM 的 SRAM 部分进行读和写操作。与现有的非易失性存储器技术（如 EEPROM、闪存、FRAM、MRAM 和电池供电 SRAM）相比，这样作能够提供更快的读和写访问。系统掉电时，通过使用存储在较小电容中的电能（该电容与器件的 V_{CAP} 引脚相连），SRAM 中的数据被自动传输到非易失性单元内。在下次通电期间，非易失性单元中的数据被自动回读到 SRAM 阵列内，以便提供给用户。在普通操作模式下，与 nvSRAM 的 V_{CAP} 引脚相连的电容被充电。

nvSRAM 为它的非易失性单元指定了一百万次的耐久性周期。只有在存储操作期间，SRAM 单元中的数据被传输到非易失性单元时，nvSRAM 耐久性才被消耗。当器件的电源电压下降到低于预定义的阈值级别（V_{SWITCH}）时，或者可以通过设定一个操作码，或（$\overline{HSB}$）设定硬件引脚要求启动该非易失性存储时，nvSRAM 中的非易失性存储将自动启动；请注意，SRAM 单元为读和写操作提供了无限次的耐久性，因此在普通的操作模式下，nvSRAM 将不消耗任何耐久性周期。仅在检测到系统掉电时，才会发生非易失性存储，并且如果要将数据安全传输到非易失性单元，需要使用该非易失性存储。这表示 nvSRAM 的耐久性周期等于系统掉电总数或系统关闭事件的总和。在所有实时应用中，该数值最大不会达到一百万个周期。

SPI nvSRAM 在符合工业标准的 8 引脚 SOIC 和 16 引脚 SOIC 封装中提供了速度高，功耗低的串行 nvSRAM。nvSRAM 允许在数十微秒的时间内写入数百字节的数据量；但 EEPROM 和闪存存储器需要数十毫秒。许多数据记录应用要求在掉电时立即存储运行中的关键信息。这些关键信息包括控制器运行过程中执行状态或暂存器数据、参数设置和控制器测量的其他环境向量。

本应用笔记详细说明了 SPI nvSRAM 连接和适用于所有标准 SPI 主控制器的功能。本应用笔记提供的硬件建议不是必要的条件；但采用这些建议可使整个设计功能更健壮。为了说明系统电平的 SPI nvSRAM 性能，某些操作码已通过时序框图和基于 PSoC 1 的伪代码介绍：

本应用笔记包含以下主题。

- SPI nvSRAM 连接
- SPI 操作模式
- nvSRAM 操作

本应用笔记还提供了一个基于 PSoC 1 项目（相关项目 1）和 PSoC 3、PSoC 4 nvSRAM SPI 库组件（相关项目 2 和相关项目 3）。

SPI nvSRAM配置

赛普拉斯在各种配置和封装选项中支持 SPI nvSRAM，具体如下表所示。

表 1. SPI nvSRAM 配置

nvSRAM 器件编号	工作电压（典型值）	封装	$\overline{WP}$ 引脚	V_{CAP} 引脚/自动存储	$\overline{HSB}$引脚/HW 存储	RTC
CY14CXXXQ1A	2.5 V	8 引脚 SOIC	有	无/无	无/无	无 RTC
CY14BXXXQ1A	3.0 V					
CY14EXXXQ1A	5.0 V					
CY14CXXXQ2A	2.5 V	8 引脚 SOIC	无	有/有	无/无	无 RTC
CY14BXXXQ2A	3.0 V					
CY14EXXXQ2A	5.0 V					
CY14CXXXQ3A	2.5 V	16 引脚 SOIC	有	有/有	有	无 RTC
CY14BXXXQ3A	3.0 V					
CY14EXXXQ3A	5.0 V					
CY14CXXXPA	2.5 V	16 引脚 SOIC	有	有/有	有	RTC
CY14BXXXPA	3.0 V					
CY14EXXXPA	5.0 V					

表 1 中的“XXX”表示 nvSRAM 器件编号中用于提供密度选项的空间大小。XXX = 064 表示 64 Kbit；XXX = 256 表示 256 Kbit；XXX = 512 表示 512 Kbit；XXX = 101 表示 1 Mbit；并且 XXX = 102 表示 2 Mbit nvSRAM 密度。

根据 nvSRAM 的器件配置和封装选项，SPI 主控制器和 nvSRAM 器件之间的连接会存在差异。图 2 到图 4 分别显示了 SPI SRAM 在各种配置和封装选项情况的详细原理图连接情况。对于特殊的配置和封装选项，SPI 主机控制器和 SPI nvSRAM 之间的硬件连接在所有密度上是相同的。

SPI nvSRAM 器件的典型系统电平配置如图 1 所示。对于没有专用 SPI 总线的微控制器，可以使用通用的 I/O 端口。

图 1. 典型的 SPI nvSRAM 连接

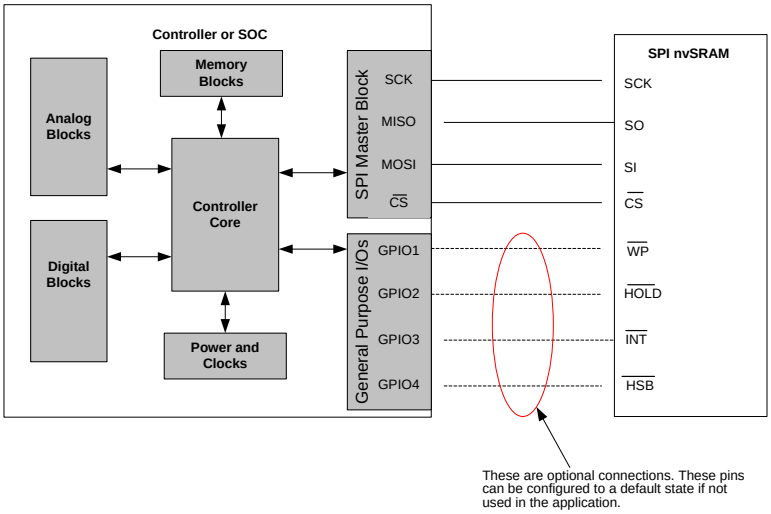


图 2. 8 引脚 SPI nvSRAM 与控制器之间的连接（无 V_{CAP}）

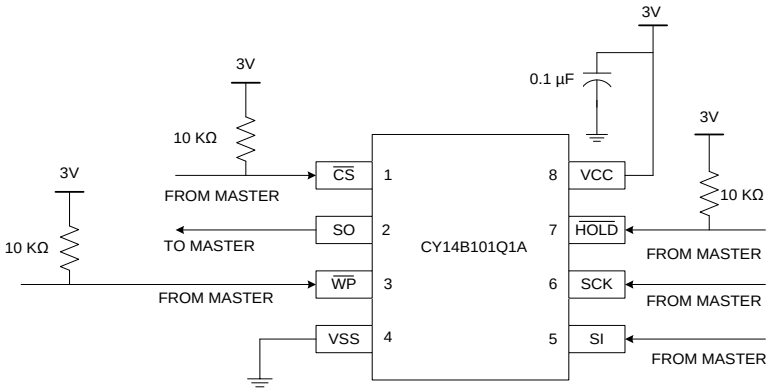


图 3. 8 引脚 SPI nvSRAM 与控制器之间的连接（有 V_{CAP}）

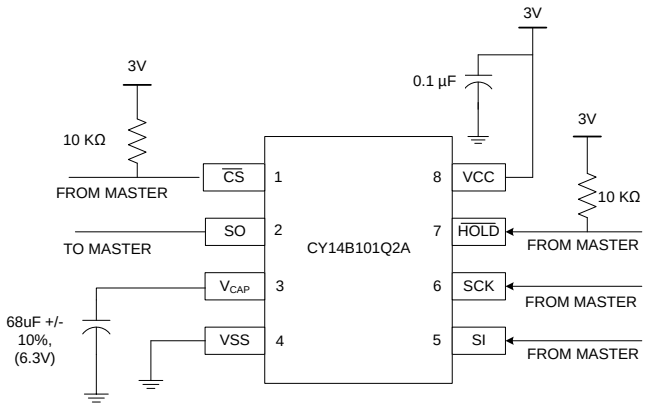
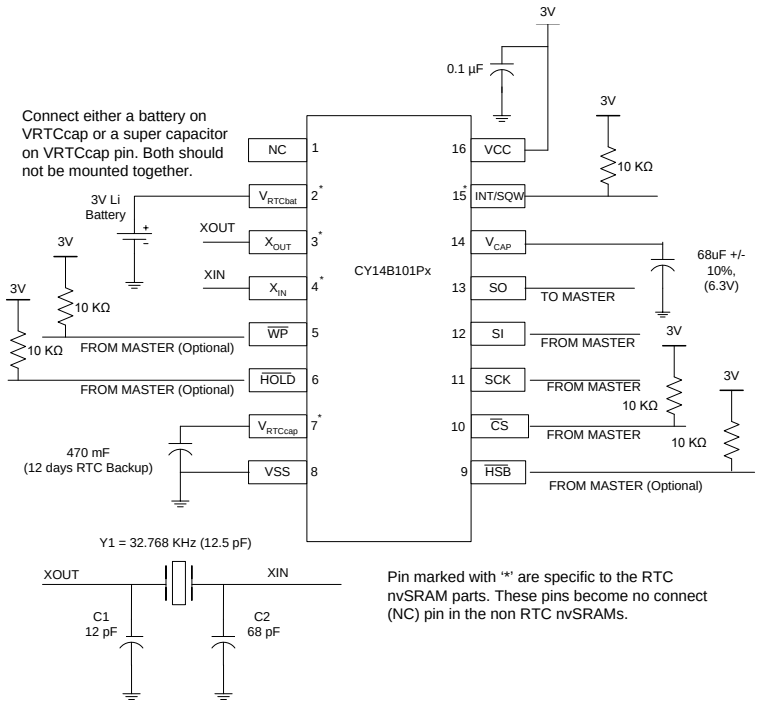


图 4. 16 引脚（RTC） SPI nvSRAM 与控制器之间的连接



输入引脚配置

SPI nvSRAM 具有多个控制输入引脚，通过将这些引脚正确设置为一个固定的逻辑状态（低电平或高电平）可执行正确的操作。如果某个引脚悬空，则它将假设中级电平导致高待机电流或它可以进入逻辑的低电平或高电平。信号移动的方向取决于多个因素，其中包括系统中的噪声、电容耦合和漏电流。因此，输入电路的电平相对随机，并且在操作期间会发生变化。这种不可预测的输入电平会严重影响器件的操作。因而，需要始终将未使用的所有输入引脚连接到合适的逻辑电平，例如，将低电平有效输入连接至高电平。可以使用大小为 10 kΩ 的电阻上拉或下拉未使用的输入引脚。

$\overline{\text{HOLD}}$引脚：SPI nvSRAM 具有$\overline{\text{HOLD}}$引脚所具有的特性。可将该引脚作为低电平有效输入使用，并且用户可以停止时钟，这样可以暂停正在进行的通信。如果该引脚进入低电平状态，则器件将不再对时钟脉冲发生反应，通信被中断并且数据会被丢失或损坏。如果不被使用，它的上拉电阻大小应该为 10 kΩ，这样是为了避免噪声在这些条件下造成的不必要的事件。

$\overline{\text{CS}}$引脚：在普通操作模式下，微控制器始终驱动芯片选择引脚（$\overline{\text{CS}}$）。在控制器断电或上电期间，该引脚被悬空。如果未被使用，则它的上拉电阻大小应该为 10 kΩ，这样是为了避免噪声在这些条件下造成的不需要的指令。

$\overline{\text{WP}}$引脚：写保护（$\overline{\text{WP}}$）引脚是一个低电平有效的输入信号。通过从外部将该引脚置于低电平可以保护对主存储器和状态控制器进行的写操作。通过使用状态寄存器中的 WPEN 位可以确定$\overline{\text{WP}}$引脚的功能。如果将 WPEN 位置‘1’，则它将使能$\overline{\text{WP}}$引脚的控制；如果将该位置‘0’，则会禁用$\overline{\text{WP}}$引脚。该引脚上应该有一个大小为 10 kΩ 的上拉电阻，这样可以避免噪声在这些条件下造成的不必要事件。

$\overline{\text{HSB}}$引脚：$\overline{\text{HSB}}$ 引脚是 nvSRAM 上的双向引脚。该引脚作为输出使用时，它将在非易失性的存储操作期间提供 nvSRAM 的就绪/繁忙状态。nvSRAM 进入就绪状态时，主控制器可以访问器件的所有功能。nvSRAM 处于繁忙状态时，将禁止所有指令（读状态寄存器除外），并且器件将设置状态寄存器中的$\overline{\text{RDY}}$位。通过使用读状态寄存器指令可以检索该寄存器。作为输入引脚使用时，控制器可通过使用$\overline{\text{HSB}}$引脚从外部启动硬件存储。如果不与任何 GPIO 相连，则该引脚被悬空。另外，通过内部弱上拉电阻可以在正常运行模式下使$\overline{\text{HSB}}$保持为高电平。如果控制器从外部控制$\overline{\text{HSB}}$，则需要使用 10 kΩ 外部上拉电阻，这样可以避免噪声在该线上引起不需要的触发。

V_{CAP}：与 V_{CAP} 相连的电容在掉电期间为 nvSRAM 供电，这样可以将 SRAM 中的数据存储在非易失性元素内。在正常运行模式下，器件从 V_{CC} 获得电流，用以向电容充电。芯片使用该存储的电荷执行单个存储操作。如果 V_{CC} 引脚的

电压下降到 V_{SWITCH} 以下，器件会自动将 V_{CAP} 引脚与 V_{CC} 的连接断开。通过 V_{CAP} 电容所提供的电源触发存储操作。

始终将合适的电容值连接到 V_{CAP} 引脚，用以成功执行自动存储操作。选择的电容值应该位于器件数据手册中所规定的范围内。如果选择的电容值不准确，将会导致器件故障。欲了解更多有关 nvSRAM 产品的电容选择指南，请参考应用笔记赛普拉斯 nvSRAM 的存储电容选项 — AN43593。

RTC引脚配置

下面引脚是特定于 RTC 特性的。如果未使用 RTC 功能，则这些引脚将在电路板上悬空。

INT 引脚：该引脚是 RTC 器件中的输出引脚。INT 输出被复用，以便可以在 RTC nvSRAM 器件中提供各种功能。根据 RTC 寄存器的设置以及定义在 nvSRAM 中的优先级，可以将 INT 引脚配置为提供警报状态、看门狗定时器状态和方波输出。通过设置中断状态/控制寄存器中的 H/L 位，可以将 INT 引脚设置为可配置的驱动器输出。将 H/L 位置‘1’时，INT 引脚为高电平有效，并且驱动器模式为推挽式。将 H/L 位置‘0’时，则 INT 引脚将为低电平的开漏输出，并且需要将外部上拉电阻驱动为逻辑高状态。因此，必须通过一个大小为 10 kΩ 外部上拉电阻将 INT 引脚上拉至 V_{CC}，同时使用低电平有效模式的 INT。

V_{RTCbat} 和 V_{RTCcap} 引脚：通过这些引脚可以为 nvSRAM 器件的 RTC 电路提供备用电源，以便在系统电源（V_{CC}）掉电时，RTC 时钟仍会正常运行。V_{RTCbat} 和 V_{RTCcap} 引脚应该与 V_{RTCbat} 上的非可再充电电池相连或与 V_{RTCcap} 引脚上的超级电容相连。如果不被使用，这些引脚应该悬空。

注意：V_{RTCcap} 引脚不会直接短接 V_{SS}。这是因为在正常运行模式下，该引脚用于为同它相连的超级电容充电。因此，V_{RTCcap} 引脚与地面（V_{SS}）直接相连时，nvSRAM 会消耗非常大的电流。

欲了解有关 nvSRAM RTC 设计指南和最佳实践的详细信息，请参考应用笔记非易失性静态随机存取存储器（nvSRAM）实时时钟（RTC）的设计指南和最佳实践 — AN61546。

SPI工作模式

SPI nvSRAM 支持 SPI 模式 0 （CPOL = 0、CPHA = 0）和 SPI 模式 3 （CPOL = 1、CPHA = 1）的操作，这些模式取决于 SPI 主设备在开始 SPI 通信时设置的时钟极性（CPOL）和时钟相位（CPHA）。表 2 显示的是与 SPI 时钟和数据相关的所有 SPI 模式（SPI 主设备和从设备分别将 SPI 时钟和数据驱动到 MOSI 和 MISO 线上）。nvSRAM 中的 SPI 模式根据主控制器的 SPI 模式进行自动配置。

表 2. SPI 工作模式

	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
nvSRAM 在 MOSI 线上锁存的数据	SCK 上升沿 （↑）	SCK 下降沿 （↓）	SCK 下降沿 （↓）	SCK 上升沿 （↑）
nvSRAM 在 MISO 线上输出的数据	SCK 下降沿 （↓）	SCK 上升沿 （↑）	SCK 上升沿 （↑）	SCK 下降沿 （↓）
SPI nvSRAM 的支持	有	不支持	不支持	有

SPI nvSRAM操作码

所有 SPI 操作码、地址和数据为 8 位数据；因此所有内部操作本质上进行的是字节宽度的操作。进行所有数据操作时，$\overline{CS}$始终保持低电平。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。SPI nvSRAM 为所有读和写操作支持工业标准操作码。另外，它还为 nvSRAM 的特定 NV 操作支持特殊操作码并且也支持高速（104 MHz）的 SPI 访问。唯一一个操作码将被分配给 SPI nvSRAM 中的每个特殊操作。表 3 显示的是 SPI nvSRAM 指令和相应的操作码。

表 3. SPI nvSRAM 操作码

指令类别	指令名	操作码	CY14B101P/Qx		CY14C101PA/QxA、CY14B101PA/QxA、CY14E101PA/QxA	
			操作码支持	SPI 频率	操作码支持	SPI 频率
状态寄存器控制指令	WREN	06H (0000 0110)	√	可达 40 MHz	√	可达 104 MHz
	WRDI	04H (0000 0100)	√	可达 40 MHz	√	可达 104 MHz
	RDSR	05H (0000 0101)	√	可达 40 MHz	√	可达 40 MHz
	FAST_RDSR	09H (0000 1001)	X	N/A	√	可达 104 MHz
	WRSR	01H (0000 0001)	√	可达 40 MHz	√	可达 104 MHz
SRAM 读取和写入指令	READ	03H (0000 0011)	√	可达 40 MHz	√	可达 40 MHz
	FAST_READ	0BH (00001011)	X	N/A	√	可达 104 MHz
	WRITE	02H (0000 0010)	√	可达 40 MHz	√	可达 104 MHz
RTC 指令[注意 1]	WRTC	12H (0001 0010)	√	可达 40 MHz	√	可达 104 MHz
	RDRTC	13H (0001 0011)	√	可达 25 MHz	√	可达 25 MHz
	FAST_RDRTC	1DH (00011101)	X	N/A	√	可达 104 MHz
NV 指令[注意 2]	STORE	3CH (0011 1100)	√	可达 40 MHz	√	可达 104 MHz
	RECALL	60H (0110 0000)	√	可达 40 MHz	√	可达 104 MHz
	ASENB	59H (0101 1001)	√	可达 40 MHz	√	可达 104 MHz
	ASDISB	19H (0001 1001)	√	可达 40 MHz	√	可达 104 MHz
睡眠模式	SLEEP	B9H (1011 1001)	X	N/A	√	可达 104 MHz
序列号	WRSN	C2H (1100 0010)	X	N/A	√	可达 104 MHz
	RDSN	C3H (1100 0011)	X	N/A	√	可达 40 MHz
	FAST_RDSN	C9H (1100 1001)	X	N/A	√	可达 104 MHz
器件 ID 读取	RDID	9FH (1001 1111)	X	N/A	√	可达 40 MHz
	FAST_RDID	99H (1001 1001)	X	N/A	√	可达 104 MHz

注意 1：RTC 指令特定于 RTC nvSRAM 器件（CY14C101P/PA、CY14B101P/PA 和 CY14E101P/PA）。这些指令不适用于非 RTC 的器件。

注意 2 ：这些指令特定于 nvSRAM 器件，以执行 NV 操作。

表 4 显示的是每个操作码以及它们的相关数据字节，以正确进行操作。

表 4. SPI nvSRAM 的数据流格式

指令名	操作码	SI 上的设备传输	SO 上的 nvSRAM 传输	注释
WREN	06H	06H	–	通过该指令可设置状态寄存器中的 WEN 位。
WRDI	04H	04H	–	清除状态寄存器中的 WEN 位（若被设置）
RDSR	05H	05H	StatusReg_Data	读取状态寄存器的内容
FAST_RDSR	09H	09H、Dummy_Byte	01H、StatusReg_Data	
WRSR	01H	01H、StatusReg_Data	-	写入状态寄存器前必须设置 WEN 位。$\overline{CS}$为高电平时，将清除 WEN 位。
READ [注意 3]	03H	03H、Add1、Add2、Add3	Data1、Data2、Data3、……、DataN	读取从 1 到 N 的数据长度，其中 N 可以为任意的整数值。nvSRAM 的内部地址计数器将自动增 1。当 nvSRAM 计数达到最大地址限制时，它将翻转到起始地址并继续读取数据。$\overline{CS}$为高电平时，将禁止读操作。
FAST_READ [注意 3]	0BH	0BH、Add1、Add2、Add3、Dummy_Byte	Data1、Data2、Data3,.,., DataN	
WRITE [注意 3]	02H	02H、Add1、Add2、Add3、Data1、Data2、Data3.,.,., DataN	–	写入 nvSRAM 存储器前必须设置 WEN 位。写入从 1 到 N 的数据长度，其中 N 可以为任意的整数值。nvSRAM 的内部地址计数器将自动增 1。当 nvSRAM 计数达到最大地址限制时，它将翻转到起始地址并通过重叠先前写入的数据继续进行编写。由于进行批量写操作时，存储器计数器会翻转，因此固件必须处理覆盖数据。$\overline{CS}$为高电平时，将禁止写操作。
WRTC [注意 4]	12H	12H、 Addr Data	–	必须在 RTC 标志寄存器中将‘W’位置‘1’，并且在状态寄存器中将 WEN 位置‘1’。$\overline{CS}$为高电平时，将清除 WEN 位。
RDRTC [注意 4]	13H	13H、Addr	数据	
FAST_RDRTC [注意 4]	1DH	1DH、Addr Dummy_Byte	数据	
STORE	3CH	3CH	–	
RECALL	60H	60H	–	
ASENB	59H	59H	–	
ASDISB	19H	19H	–	
SLEEP	B9H	B9H	–	在启动睡眠指令前，必须设置 WEN 位。$\overline{CS}$为高电平时，器件将注册睡眠指令。$\overline{CS}$为高电平时，将清除 WEN 位。
WRSN	C2H	C2H、Data1、Data2、……、Data8	–	必须设置 WEN 位。写入 8 字节的串行编号。$\overline{CS}$为高电平时，将清除 WEN 位。
RDSN	C3H	C3H	Data1、Data2、Data3,., Data8	读取 8 字节的串行编号
FAST_RDSN	C9H	C9H、Dummy_Byte	Data1、Data2、Data3、……、Data8	读取 8 字节的串行编号
RDID	9FH	9FH	Data1、Data2、Data3、Data4	器件 ID （4 字节）

指令名	操作码	SI 上的设备传输	SO 上的 nvSRAM 传输	注释
FAST_RDID	99H	99H、Dummy_Byte	Data1、Data2、Data3、Data4	器件 ID （4 字节）

注意 3：1 Mb 和密度更高的 SPI nvSRAM 使用 3 字节地址；密度更低的 nvSRAM （512 Kbit 或更小值）使用 2 字节地址。

注意 4：RTC 指令特定于 RTC nvSRAM 器件（CY14C101PA、CY14B101PA 和 CY14E101PA）。这些指令不适用于非 RTC 器件。

在SPI nvSRAM中进行寻址

SPI 主机控制器逐字节与 SPI nvSRAM 通信。在字节传输期间，它始终传输第一个时钟周期中最高的有效位和第八个时钟周期中最低的有效位。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 在读取操作期间传输数据字节时，将始终先传输最高有效位，最后才传输最低有效位。图 5 显示的是一个示例，说明了在 SPI 主设备传输三个地址字节时，地址位通过 SPI MOSI（主出从入）线传输。

图 5．SPI nvSRAM 中的地址位传输

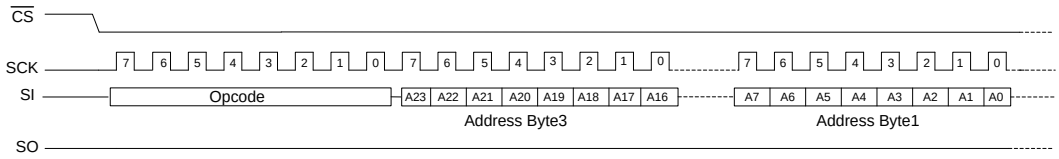


图 6 显示的是各种 nvSRAM 密度的寻址方案。A0 是地址中的最低有效位（LS 位）。

图 6。SPI nvSRAM 操作码和寻址

Density	Opcode								Address Byte3 (MSB)								Address Byte2 (Intermediate Byte)								Address Byte1 (LSB)							
	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
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

无需关注最高有效地址字节（MSB）中未使用的位，并且 nvSRAM 会将忽略这些位的状态。但在固件中应该将未使用地址位置‘0’。这样，在同一个插座中移动到更高密度器件时可更容易更新固件。

nvSRAM操作

本节介绍的是如何使用时序框图和特定于 PSoC 1 的伪代码进行 nvSRAM 操作。前缀为‘SPIM_SPIM_’的所有函数都是特定于 PSoC 1 的函数。需要修改它们的展示和实施，以便可将控制器作为 SPI 主设备使用。

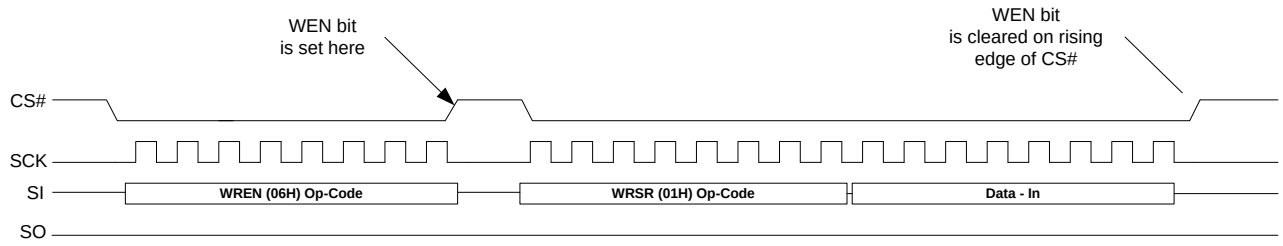
本节没有介绍 SPI nvSRAM 的所有操作码。有关每一个操作码的详细说明，请参考器件的数据手册。

状态寄存器的操作

写入状态寄存器：要想写入状态寄存器，需要发送状态寄存器的写操作码（WRSR），然后才能写入数据字节。

- 通过发送 WREN 操作码设置 WEN 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。更多有关状态寄存器的详细信息，请参见“器件数据手册”中介绍的内容。图 7 显示的是写入状态寄存器的时序框图。

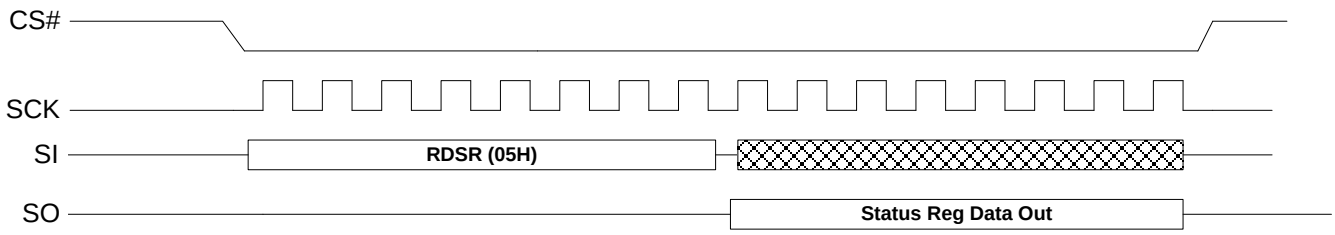
图 7. 写入状态寄存器




```
/*******************PSoC1 Based Pseudo Code for Status Register Write***************/
#define CS_HI Port0_0(1)
#define CS_LO Port0_0(0)

void WRSR(BYTE data1) //User Define Function
{
    BYTE WREN=0x06;
    BYTE OPCODEWRSR=0x01;

    CS_LO;
    while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
    SPIM_SendTxData(WREN); //This will set WEN='1'
    CS_HI;

    CS_LO;
    while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
    SPIM_SendTxData(OPCODEWRSR);//Send OPCODE for Status Register write

    while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
    SPIM_SendTxData(data1); //Send the data to be written into Status Register
    CS_HI;
}
```

读取状态寄存器：要想读取状态寄存器中的内容，需要发送读状态寄存器的操作码（RDSR），然后 nvSRAM 才会开始将状态寄存器中的内容发送到 SO 线上。必须通过将芯片选择引脚置于低电平保持选择 SPI nvSRAM，并且执行 RDSR 指令后，SPI 时钟才能读取状态寄存器中的内容。图 8 显示的是读取状态寄存器的时序框图。

图 8。读取状态寄存器




```
/******** PSoC1 Based Pseudo Code for Status Register Read************************/
BYTE RDSR ()// User Define Function

{
    BYTE OPCODERDSR=0x05;
    BYTE data;

    CS_LO;
    while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
    SPIM_SendTxData(OPCODERDSR); //Send instruction

    while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
    SPIM_SendTxData(0x01); //Dummy write to generate CLK and read data

    while(!(SPIM_bReadStatus() & SPIM_SPIM_RX_BUFFER_FULL));
    data = SPIM_bReadRxData(); //Read Byte from Status Register
    CS_HI;

    return(data);
}
```

nvSRAM中的SRAM写/读操作

SRAM 写操作：要想写入 SPI nvSRAM 的 SRAM 阵列，控制器必须通过以下的方式启动写指令。

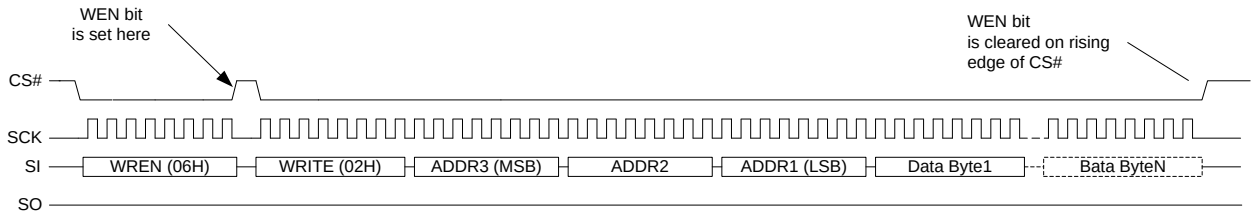
- 通过发送 WREN 操作码设置写使能锁存（WEN）位
- 发送 WRITE 操作码
- 发送最高有效地址字节
- 发送（3 字节地址内的）中间地址字节
- 发送低位地址字节
- 逐字节发送数据

必须在发送所有 nvSRAM 写指令前发送写使能（WREN）指令。如果器件未处于写入使能状态（WEN = “0”），则会忽略写入指令并在$\overline{CS}$处于高电平状态时返回到待机状态。要重新启动 SPI 串行通信，需要一个新的$\overline{CS}$下降沿。

注意：在完成写入指令（WRSR、WRITE 或 WRTC）或 nvSRAM 特殊指令（存储、回读、ASENB 和 ASDISB）后，写周期结束时，状态寄存器的 WEN 位将在$\overline{CS}$的上升沿上被清除为“0”。这样可以防止意外写入。

另外，请注意在 WREN 和任何写指令之间读取状态寄存器（RDSR 操作码）不会清除 WEL 位。发送 WREN 操作码后，用户将立即读取状态寄存器，以保证在进行写操作前设置了 WEL 位。图 9 显示的是写入 SRAM 存储器的时序框图。

图 9。写入 SRAM




```
/* PSoC1 Based Pseudo Code for nvSRAM write in burst mode. By sending tot_cnt =1, user
can write only 1 byte at a given address location*/

void nvSRAMBURSTWRITE(BYTE addr1, BYTE addr2, BYTE addr3, DWORD tot_cnt, BYTE*data) //
User Define Function
    {   BYTE WREN=0x06;
        BYTE OPCODEWRITE=0x02;
        DWORD count=0;

        CS_LO;
        while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
        SPIM_SendTxData(WREN); //Set WEN='1' prior to write
        CS_HI;

        CS_LO;
        while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
        SPIM_SendTxData(OPCODEWRITE); //Send OPCODE for Write into main memory
        while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
        SPIM_SendTxData(addr1); //Send MS Byte
        while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
        SPIM_SendTxData(addr2); //Send Intermediate Address Byte
        while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
        SPIM_SendTxData(addr3); //Send LS Byte of Address

        for(count=0; count< tot_cnt; count++)
        {
        while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
        SPIM_SendTxData(data[count+4]); //Byte written into main memory}
        CS_HI;
}
```

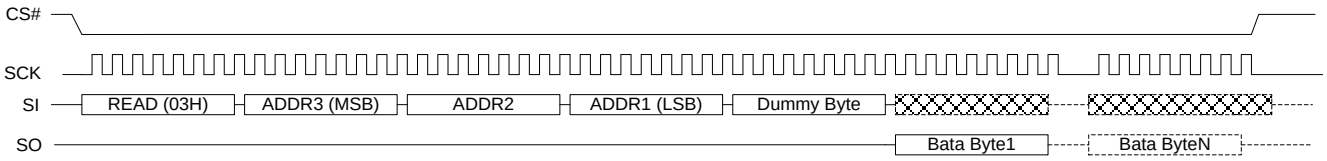
SRAM 读操作：要想读取 SPI nvSRAM 的 SRAM 阵列，控制器必须使用以下格式发送操作码和地址。

- 发送 READ 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
- 发送低位地址字节

nvSRAM 一直将数据发送到 SO 线上，直到通过将芯片选择信号置于低电平选择器件和 SPI 时钟有效为止。

要启动 SRAM 读取操作，控制器必须先发送 READ 操作码，然后读取地址字节。注册读请求和地址后，nvSRAM 会将数据发送到 SO 引脚上。当连续传送数据字节时将$\overline{CS}$置于低电平便可以访问后续的数据。该过程称为突发模式读取，并且 SPI nvSRAM 器件将自动递增地址。当$\overline{CS}$被取消激活为高电平时，将停止输出数据，并且 SO 处于高阻状态（HI-Z）。图 10 显示的是读取 SRAM 的时序框图。

图 10. 读取 SRAM




```
/********************nvSRAM Read Burst Data*************************/
void  nvSRAMBURSTREAD(BYTE  addr1,  BYTE  addr2,  BYTE  addr3,  DWORD  tot_cnt,BYTE  *
readDataArr) User Define Function
   {
       BYTE readdata;
       BYTE data;
       BYTE OPCODEREAD=0x03;
       DWORD count=0;

       CS_LO;
       while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
       SPIM_SendTxData(OPCODEREAD); //Send Read Opcode
       while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
       SPIM_SendTxData(addr1); // Send MS Byte
       while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
       SPIM_SendTxData(addr2); // Send Intermediate Address Byte
       while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
       SPIM_SendTxData(addr3); // Send LS Byte of Address

       for(count=0; count<tot_cnt; count++)
       {
       while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
       SPIM_SendTxData(addr1); //Dummy write to generate CLK and read data
       while(!(SPIM_bReadStatus() & SPIM_SPIM_RX_BUFFER_FULL));
       readDataArr[count] = SPIM_bReadRxData();// Read data from nvSRAM
       }
       CS_HI;
}
```

示例项目设置

根据目标应用可以修改本应用笔记中附带的基于 PSoC 项目。除了 SPI 主设备的功能外，通过 PSoC 的可编程性和灵活性还可以将更多功能添加到器件内。随附在本应用的项目显示的是一个带有 PSoC 的 SPI nvSRAM 接口的示例。

要想运行和执行附加源代码（而不需要修改它），需要进行下列设置。表 5 显示的是示例项目中的连接详细信息。

- PSoC Designer™ — 5.2 或更高版本。
- 硬件套件 — CY8CKIT-001 （DVK1 开发电路板）
- CY8C29 处理器模块（CY8CKIT-008）
- 用于编译项目的 PSoC 1 器件编号：CY8C29866-24AXI
- UART 波特率设置 — 19200 波特/秒
- PSoC 1 控制器的 V_{DD} 电源 — 3.3 V

表 5. 示例项目中的 PSoC 端口配置

nvSRAM 信号名称	PSoC（主设备）信号名称	PSoC 1 I/O 分配	信号方向
$\overline{CS}$	CS_n	Port0_1	PSoC 1 输出
SI	MOSI	Port0_1	PSoC 1 输出
SO	MISO	Port0_2	PSoC 1 输入
SCK	SCK	Port0_3	PSoC 1 输出
$\overline{WP}$	WP_n	Port0_4	PSoC 1 输出
$\overline{HOLD}$	HOLD_n	Port0_5	PSoC 1 输出
INT/SQW	INT	Port0_6	PSoC 1 输入
$\overline{HSB}$	HSB_n	Port0_7	PSoC 1 输入/输出（示例项目将该 PSoC 1 引脚配置为输出，通过设置合适的驱动模式寄存器会以动态方式将该引脚配置为输出/输入）
	UART TX	Port2_5	PSoC 1 输出（输出到 UART 终端）
	UART RX	Port2_6	PSoC 1 输入（从 UART 终端输入）

总结

同其他非易失性 SPI 存储器产品相似，赛普拉斯 SPI nvSRAM 支持标准的 SPI 访问协议。这样能使 nvSRAM 与所有 SPI 主控制器相兼容，并缩短系统开发周期。所有 SPI 操作码（特定于 nvSRAM 的操作码除外）都与标准 SPI 存储器产品的操作码匹配。这样 SPI nvSRAM 可以替换所有其他非易失性存储器，并且能够提供相同的功能和外形。本应用笔记演示了如何使用原理图、时序框图和示例代码在一个应用中配置 SPI nvSRAM。

文档修订记录

文档标题：使用串行外设接口（SPI） nvSRAM 进行设计 — AN64574

文档编号：001-92137

版本	ECN	变更者	提交日期	变更说明
**	4345888	LISZ	05/16/2014	本文档版本号为 Rev**，译自英文版 001-64574 Rev*G。
*A	4718351	LISZ	04/09/2015	本文档版本号为 Rev*A，译自英文版 001-64574 Rev*H。

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

PSoC 是赛普拉斯半导体公司的注册商标，PSoC Designer 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
网站地址 ：www.cypress.com

© 赛普拉斯半导体公司，2010-2015。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。