SPI 从器件数据表 SPIS V 2.5

Copyright © 2002-2010 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	CapSense®	I²C/SPI	定时器	闪存	RAM	
CY8C20x34、CY8C20x24、CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CY7C604xx、CY7C643xx、CYONS2010、CYONS2011、CYONSFN2051、CYONSFN2053、CYONSFN2061、CYONSFN2151、CYONSFN2161、CYONSFN2162、CY8CTST200、CY8CTMG2xx						
		X		43	0	4

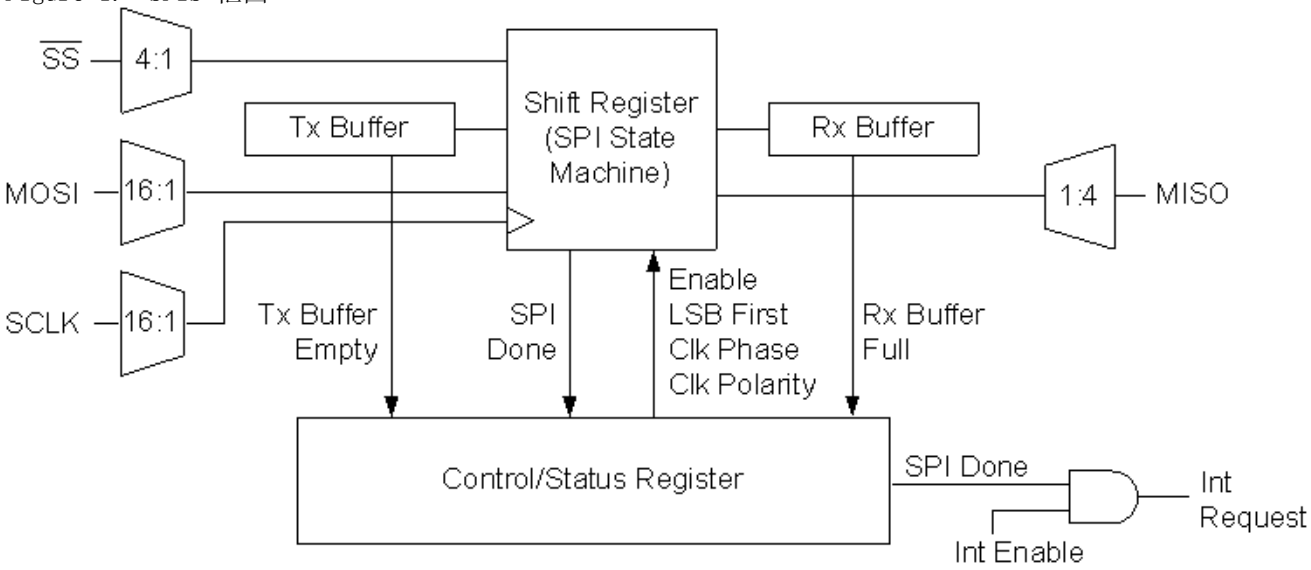
如需获取一个或多个使用此用户模块的完整配置功能性示例项目，请转到 http://www.cypress.com/psocexampleprojects.

特性与概述

- 支持串行外设互连 (SPI) 从器件协议。
- 支持协议模式 0、1、2 和 3。
- 适用于 MOSI、SCLK 和 ~SS 的可选输入源。
- 适用于 MISO 的可选输出路由。
- 基于 SPI 完成条件的可编程中断。
- SS 可能由固件控制。

SPIS 用户模块是一种串行外设互连从器件 (SPIS)。 它能够执行全双工同步 8 位数据传输。 用户可以指定 SCLK 相位、SCLK 极性和最低有效位优先，以适应大多数 SPI 协议。 SPIS PSoC 模块拥有输入和输出信号的可选路由以及可编程的中断驱动控制。 应用程序编程接口 (API) 固件为汇编语言和 C 语言应用程序软件提供了高级别的编程接口。

Figure 1. SPIS 框图



功能说明

SPIS 是一个用户模块，可实现串行外设互连从器件。 它使用 I2C/SPI PSoC 模块的 Tx 缓冲区、Rx 缓冲区、控制寄存器、配置寄存器和一个数据寄存器进行数据移位。

控制寄存器通过器件编辑器和 / 或 SPIS 用户模块固件 API 子程式进行初始化和配置。 初始化包括设置最低有效位优先以及 SPI 发送 / 接收协议模式。 支持 SPI 模式 0、1、2 和 3。 将 SPI 主控和 SPI 从器件设置为相同模式和位配置，以便正确进行通信。 SPI 模式定义如下。

Table 1. SPI 模式

模式	执行数据栓锁的 SCLK 边沿	时钟极性	注：
0	上升沿	非反相	上升沿栓锁数据。 数据在时钟下降沿上发生更改
1	上升沿	反相	
2	下降沿	非反相	下降沿栓锁数据。 数据在上升沿上发生更改。
3	下降沿	反相	

SCLK 输入信号是 SPI 主控生成的 SPI 发送 / 接收时钟。 它定义了发送 / 接收数据的位速率。

MOSI 输入信号是主出从入数据信号，它从 SPI 主控接收数据。

MISO 输出信号是主入从出数据信号，它将数据从移位寄存器发送到 SPI 主控。 通常，多个从器件的 MISO 信号捆绑在一起。 每个从器件将其各自的 MISO 信号分为三种状态，直到其从器件选择信号激活为止。 此用户模块不会将 MISO 输出信号分为三种状态。 如果需要，可将此功能添加到用户模块中。

SPIS 硬件从 MOSI 信号上的 SPI 主控中接收数据，同时将数据发送到 MOSI 信号上的 SPI 主控。 使用同一个 SCLK 信号发送和接收主控和从器件的数据。

SPI 协议是仅主控启动响应协议。 主控将从器件选择（~SS）输入信号置为低电平，以启用特定的 SPIS 器件进行主动通信。

主控的职责是确定所选从器件是否准备好接受命令或准备接收数据。

[+] Feedback

当使用 API 子程式在控制寄存器中设置 SPI 启用位时，SPIS 用户模块将启用以执行操作。

发送到 SPI 主控的数据写入 Tx 缓冲区寄存器中。 这将会清除 “Tx 缓冲区空 ” 状态位。

在从器件选择信号的下降沿，数据从 Tx 缓冲区寄存器传输到移位寄存器。 然后，数据字节首先发送的位在 MISO 输出信号上激活。 此时，另一个要发送的数据字节写入 Tx 缓冲区寄存器中。 当前字节的数据发送完成后，该数据就准备发送到 SPI 主控。

在激活每个 SCLK 输入信号的同时，数据会从移位寄存器移出到 MISO 输出，并从 MOSI 输入移入到移位寄存器。 SCLK、MOSI 和 MISO 信号的具体时序基于 SPI 模式配置。

在所有的位同时发送和接收后，接收的数据将从移位寄存器传输到 Rx 缓冲区寄存器，且 Tx 缓冲区寄存器传输到移位寄存器。 设置 “Rx 缓冲区满 ”(Rx Buffer Full) 和 “SPI 完成 ”(SPI Done) 状态位。 如果中断处于启用状态，“SPI 完成 ” 状态位将引起中断触发。 此中断用于提醒软件接收到数据字节或字节发送成功。

如果一个待处理的数据字节当前在 Tx 缓冲区寄存器中加载，那么当主控再次执行数据操作时该字节准备发送。

如果不使用 “SPI 完成 ”(SPI Done) 中断条件检索来自 Rx 缓冲区寄存器的数据字节，则轮询控制寄存器来监控 “Rx 缓冲区满 ”(Rx Buffer Full) 状态位。 在下一个数据字节完全接收或者设置过速错误状态位之前，读取来自 Rx 缓冲区寄存器的已接收数据。

监控 “SPI 完成 ” 位以确定何时禁用 SPIS 用户模块。 这样可以确保 SPI 主控和从器件之间的所有时钟信号都是完整的。

如果使用中断，则 SPIS 状态寄存器在每次中断后必须进行清除，以识别下一次中断。 读取状态寄存器会清除中断控制器识别的内部信号。 如果此信号保持高电平，则后续 SPIS 中断将被屏蔽。 这适用于 TxComplete 和 RxComplete 中断，不适用于 TxRegEmpty 和 RxRegEmpty 中断。 可使用汇编语言 **MOV** 或 **TST** 操作码读取并清除寄存器。

直流和交流电气特性

Table 2. SPIS 直流和交流电气特性

参数	条件和注释	典型值	极限值	单位
F_{max}	RX/TX 最大频率	--	12	MHz

放置

SPIS 可以映射到单个 PSoC 模块。 在 PSoC Designer 器件编辑器中，该模块名称为 SPIS。 它可以置于 I2C/SPI 模块中。

参数和资源

SS 使能

此选项决定从器件选择 (SS_) 信号是否为内部驱动。 如果选中 SW_SlaveSelect 选项，则其逻辑电平由 SPI_CFG 寄存器的 SS_ 位 （掩码 0x08）决定。 如果是外部驱动，则其逻辑电平由外部引脚决定。

中断模式

此选项决定何时对 TX 模块产生中断。 TxRegEmpty 选项可在数据从数据寄存器传输至移位寄存器后产生中断。 选择第二个选项 TxComplete 时会延迟中断，直到最终位移出移位寄存器。 第二个选项在

[+] Feedback

需要了解字符何时完全发送的情况下很有用。第一个选项 TxRegEmpty 可用于最大化发送器的输出。此选项允许在前 1 个字节正在发送时加载 1 个字节。

中断生成控制

下列参数仅在选中了 PSoC Designer 中“启用中断生成控制”(Enable interrupt generation control) 复选框时可用。该复选框位于“项目”>“设置”>“芯片编辑器”之下。**“启用中断生成控制”**复选框时，有两个附加参数将变为可用。此复选框位于 **“项目”>“设置”>“芯片编辑器”之下。**. 当拥有多个程序层而且多个用户模块在不同的程序层共享中断时，中断生成控制是非常重要的。

IntDispatchMode

IntDispatchMode 参数用于指定如何处理中断请求（当处于同一模块的多个用户模块在不同的程序层共享该中断时）选择“ActiveStatus”会导致固件在为共享的中断请求提供服务之前测试哪一个外覆层正处于活动状态。这项测试发生在每次请求共享中断的时候。这样会导致延迟增加并且会产生一个服务于共享中断请求的非确定性的程序，但并不要求任何 RAM 资源。选择“OffsetPreCalc”会导致固件在最初只有一个外覆层载入时计算共享中断请求的来源。这项计算减少了中断延迟会产生一个服务于共享中断请求的确定性程序，但其代价是 1 个字节的 RAM 资源。

应用程序编程接口

API 子程式作为用户模块的一部分提供，使设计人员能够在较高的层级处理模块。本节具体说明了每个函数对应的接口以及 include 文件所提供的相关常量。

Note 在这里，如同所有用户模块中的 API 一样，A 和 X 寄存器的值可通过调用 API 函数进行更改。发起调用 API 的函数有责任在调用之前保留 A 和 X 的值（如果调用后需要再次用到它们）。为实现高效，选择了这项“寄存器为易失性”的策略，该策略从 PSoC Designer 1.0 版本开始实施。C 语言编译器自动处理此项要求。汇编语言程序员也必须确保其代码遵守该策略。尽管有些用户模块的 API 函数会保留 A 和 X 不变，但并不保证在未来也将如此。

以下列出了 SPIS 提供的 API 函数。

SPIS_Start

说明：

通过在控制寄存器中设置正确的位来设置 SPI 接口的模块配置，并启用 SPIS 模块。

在调用此函数之前，将所有的从器件选择信号置为高电平，以取消选中连接的 SPI 从器件。在用户提供的子程式中执行此操作。

C 语言原型：

```
void  SPIS_Start(BYTE bConfiguration)
```

汇编语言：

```
mov   A,SPIS_MODE_2 | SPIS_LSB_FIRST
lcall  SPIS_Start
```

参数：

bConfiguration：用于指定 SPI 模式和“最低有效位优先”(LSB First) 配置的一个字节。下表给出了在 C 语言和汇编语言中提供的符号名称及其相关数值。请注意，符号名称可以通过“或”运算结合起来形成 SPI 接口的配置。

[+] Feedback

符号名称	值
SPIS_MODE_0	0x00
SPIS_MODE_1	0x02
SPIS_MODE_2	0x04
SPIS_MODE_3	0x06
SPIS_LSB_FIRST	0X80
SPIS_MSB_FIRST	0X00

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIS_Stop

说明：

通过在控制寄存器中清除启用位来禁用 SPIS 模块。

C 语言原型：

```
void SPIS_Stop(void)
```

汇编语言：

```
lcall  SPIS_Stop
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIS_EnableInt

说明：

在 “SPI 完成 ” 条件下启用 SPIS 中断。 SPIM 的放置位置决定了特定中断矢量和优先级。

C 语言原型：

```
void SPIS_EnableInt(void)
```

汇编语言：

```
lcall  SPIS_EnableInt
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIS_DisableInt

说明：

在 “SPI 完成 ” 条件下禁用 SPIS 中断。

C 语言原型：

```
void SPIS_DisableInt(void)
```

汇编语言：

```
lcall  SPIS_DisableInt
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIS_SetupTxData

说明：

将要发送到 SPI 主控的数据字节写入 Tx 缓冲区寄存器。

C 语言原型：

```
void SPIS_SetupTxData(BYTE bTxData)
```

汇编语言：

```
mov   A, bTxData
lcall  SPIS_SetupTxData
```

参数：

bTxData： 将要发送到 SPI 主控并在累加器内传递的数据。

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIS_bReadRxData

说明：

返回在从器件中接收到的数据字节。 在调用该子程式之前检查 “Rx 缓冲区满 ” 标志，以验证数据字节是否已接收到。

C 语言原型：

```
BYTE SPIS_bReadRxData(void)
```

汇编语言：

```
lcall  SPIS_bReadRxData
mov    bRxData, A
```

参数：

无

返回值：

在从器件 SPI 中接收并在累加器中返回的数据字节。

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIS_bReadStatus

说明：

读取并返回当前的 SPIS 控制 / 状态寄存器。

C 语言原型：

```
BYTE SPIS_bReadStatus(void)
```

汇编语言：

```
lcall  SPIS_bReadStatus
and    A, SPIS_SPI_COMPLETE | RX_BUFFER_FULL
jnz    SPI_COMPLETE_GET_RX_DATA
```

参数：

无

返回值：

返回状态字节读取内容并在累加器中返回。 使用定义的掩码测试特定的状态条件。 请注意，这些掩码可以通过 “ 或 ” 运算结合起来测试多种条件。

SPIM 状态掩码	值
SPIS_SPI_COMPLETE	0x20
SPIS_RX_OVERRUN_ERROR	0x40
SPIS_BUFFER_EMPTY	0x10
SPIS_RX_BUFFER_FULL	0x08

副作用：

调用此函数之后将会清除状态位。 此函数可能会修改 A 和 X 寄存器。

[+] Feedback

SPIS_DisableSS

说明：

当不需要外部从器件选择（~SS）信号时，使用固件将低电平有效 ~SS 信号设置为高电平状态。 要使用此函数，请将名为 “ 从器件选择输入 ” 的 SPI 参数设置为值 SW_SlaveSelect，而不是其中一个行输入。 如果已连接外部信号，则此函数无效。

C 语言原型：

```
void SPIS_DisableSS(void)
```

汇编语言：

```
lcall SPIS_DisableSS
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

SPIS_EnableSS

说明：

当不需要外部从器件选择（~SS）信号时，使用固件将低电平有效 ~SS 信号设置为低电平状态。 要使用此函数，请将名为 “SS 使能 ” 的 SPIS 参数设置为值 SW_SlaveSelect，而不是 SlaveSelect_Pin.。如果已连接外部信号，则此函数无效。

C 语言原型：

```
void SPIS_EnableSS(void)
```

汇编语言：

```
lcall SPIS_EnableSS
```

参数：

无

返回值：

无

副作用：

此函数可能会修改 A 和 X 寄存器。

[+] Feedback

固件源代码示例

本示例显示如何创建 SPI 回环。 它会回送从 SPI 主控接收的数据。 如果检测到错误状况，则发送 NAK。

```
;*****************************************************************
;  Loop Back:
;
;  This code sample illustrates how to setup a SPI Slave loop back.
;  Data received is echoed back to the SPI master.
;
;  This sample is written so that it is performed in the non interrupt
;  processing. This code is easily written as interrupt driven.
;
;  You must first properly set and start the SPI Slave user module.
;
;*****************************************************************
include  "SPIS.inc"
export   LoopBack
export   _main
_main:
        call LoopBack
LoopBack:
        ; wait for data to be received
        call  SPIS_bReadStatus
        and   A, SPIS_SPIS_SPI_COMPLETE
        jz    LoopBack

        ; read the data from the receiver
        call  SPIS_bReadRxData

        ; setup to transmit the response data
        call  SPIS_SetupTxData

        ; go wait for next byte
        jmp   LoopBack
```

以 C 语言编写的相同代码：

```
#include  "SPIS.h"

void  LoopBack(void)
{
    BYTE  bData;
    while(1)
    {
        /* wait for data to be received */
        while( !( SPIS_bReadStatus() & SPIS_SPIS_SPI_COMPLETE ) );

        /* read the received data */
        bData = SPIS_bReadRxData();


        /* setup to transmit the response data */
        SPIS_SetupTxData(bData);
    }
}
```

```
void main(void)
{
   LoopBack ();
}
```

配置寄存器

用来配置以下用户模块的数字通信 A 型 PSoC 模块寄存器在此处进行描述。 只对经过参数化的符号进行了解释。

Table 3. SPIS 模块： 配置寄存器

位	7	6	5	4	3	2	1	0
值	Clock Sel			旁路	SS_	SS_EN_	IntSel	从器件

Clock Sel – 时钟选择。 这些位决定 SPI 从器件的工作频率。

000b – SysClk / 2

001b – SysClk / 4

010b – SysClk / 8

011b – SysClk / 16

100b – SysClk / 32

101b – SysClk / 64

110b – SysClk / 128

111b – SysClk / 256

旁路 – 旁路同步。 此位决定输入是否同步到 SYSCLK。

SS_ – 从器件选择。 此位决定当 SS_EN_ 信号激活 (SS_EN_ = 0) 时，SS_ 信号的逻辑值。

SS_EN_ – 从器件选择使能。 此有效低位决定从器件选择 (SS_) 信号是否为内部驱动。 如果是内部驱动，则其逻辑电平由 SS_ 位决定。 如果是外部驱动，则其逻辑电平由外部引脚决定。

Int Sel – 中断选择。 此位选择产生中断的条件，无论它是基于 “TX 寄存器空 ” 条件还是 “SPI 完成 ” 条件。

从器件 – 此位决定模块用作主控还是从器件。

Table 4. SPIS 模块： TX 数据缓冲区寄存器

位	7	6	5	4	3	2	1	0
值	TX 缓冲区寄存器							

Tx 缓冲区寄存器： 写入此缓冲区的数据在 PSoC 模块启用时传输到移位寄存器。

Table 5. SPIS 模块： RX 数据缓冲区寄存器

位	7	6	5	4	3	2	1	0
值	RX 缓冲区寄存器							

RX 缓冲区寄存器： 在此移位寄存器中接收的数据在 SPI 发送周期结束后传输到此寄存器。

[+] Feedback

Table 6. SPIS 模块： 控制寄存器

位	7	6	5	4	3	2	1	0
值	最低有效位优先	RX 过速错误	SPI 完成	TX 缓冲区空	Rx 缓冲区满	时钟相位	时钟极性	使能

最低有效位优先 - 优先传送最低有效位。

Rx 过速错误 - 用于指示在接收下一个字节之前不能读取先前接收到的数据字节的标志。

SPI 完成 - 用于指示 SPI 传送 / 接收周期完成的标志。

Tx 缓冲区空 - 用于指示一个空的 Tx 缓冲区的标志。

Rx 缓冲区满 - 用于指示移位寄存器接收到 1 个字节数据的标志。

时钟相位 - 指示 SCLK 信号的相位。 它是定义 SPI 模式的参数之一。

时钟极性 - 指示 SCLK 信号的极性。 它是定义 SPI 模式的参数之一。

使能 - 设置时使能 SPI PSoC 模块。

版本历史记录

版本	创作者	说明
2.5	DHA	CY7C64315 和 CY7C64316 PSoC 设备的 P1[0] 设置为 SPI CLK 引脚。

Note PSoC Designer 5.1 在所有用户模块数据表中提供版本历史记录。 本部分记录了当前和先前用户模块版本之间区别的高级描述。

Copyright © 2002-2010 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.