SPI 从器件数据表 SPIS V 2.60

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

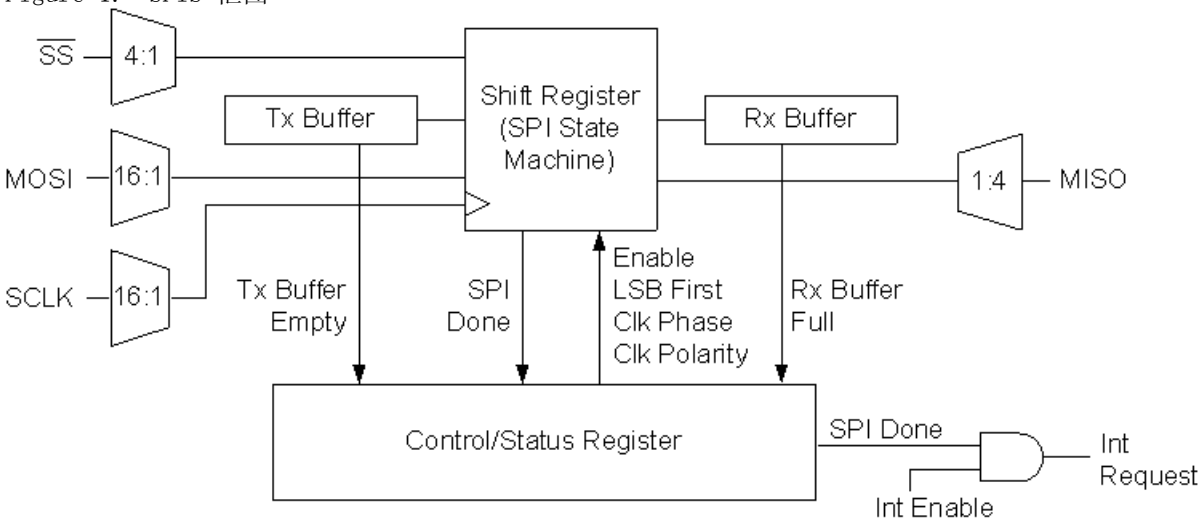
资源	PSoC® 模块			API 内存 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22/21、CY8C23x33、CY7C603xx、CY7C64215、CYWUSB6953、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTMA30xx、CY8C28x45、CY8CPLC20、CY8CLED16P01、CY8C28xxx、 CY8C21x12						
	1	0	0	43	0	4
CY8C26/25xxx	1	0	0	37	0	4

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
- MOSI、SCLK 和 ˜SS 可选择输入源
- MISO 可选择输出路由
- SPI 完成条件下的可编程中断
- SS 可能由固件控制 （不包括 PSoC 器件的 CY8C26/25xxx 系列）

SPIS 用户模块是一种串行外设互连从器件。 它执行全双工同步 8-bit 数据传输。 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 是一个实施串行外设互连从器件的用户模块。 该模块使用数字通信型 PSoC 模块的 Tx 缓冲区寄存器、Rx 缓冲区寄存器、控制寄存器和移位寄存器。

控制寄存器通过器件编辑器和 / 或 SPIS 用户模块固件应用程序编程接口 (API) 子程序进行初始化和配置。 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。 为了正常通信，必须为 SPI 主控和 SPI 从器件设置相同的模式和位配置。 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 输入信号是从 SPI 主控接收数据的主出从入 (Master-Out-Slave-In) 数据信号。

MISO 输出信号是将数据从移位寄存器传输到 SPI 主控器件的主入从出 (Master-In-Slave-Out) 数据信号。 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，用户可将此功能轻松添加到用户模块。

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 协议是仅主控起始响应协议。 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。

当使用 API 子程序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。

要传输到 SPI 主控的数据会被写入到 Tx 缓冲区寄存器。 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。 要传输的数据字节的第一位随后将置入到 MISO 输出信号上。 此时可将另一个要发送的数据字节写入至 Tx 缓冲区寄存器。 当前字节传输完毕后，即可将此数据发送到 SPI 主控。

激活每个 SCLK 输入信号后，该数据将一边从移位寄存器移出到 MISO 输出，一边从 MOSI 输入移入到移位寄存器。 SCLK、MOSI 和 MISO 信号的具体时序取决于 SPI 模式的配置。

当所有位已完成传输并同时完成接收后，接收的数据将从移位寄存器传输到 Rx 缓冲区寄存器，Tx 缓冲区寄存器中的数据将传输到移位寄存器。 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。 此中断可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将准备发送。

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，则应轮询控制寄存器来监控 “Rx 缓冲区满 ”(Rx Buffer Full) 状态位。 在完全接收下一个数据字节或设定 “ 过速错误 ” 状态位前，必须从 Rx 缓冲区寄存器中读取已接收的数据。

[+] Feedback

应对 “SPI 完成 ” 位进行监控，以确定禁用 SPIS 用户模块的时间。 这可以保证所有时钟信号已经在 SPI 主控与从器件之间完成。

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。

时序

SPIS 传输的典型时序如下图所示。 更多 SPIS 时序信息请参见 《技术参考手册》。

Figure 2. 模式 0 和 1 的典型 SPIS 时序

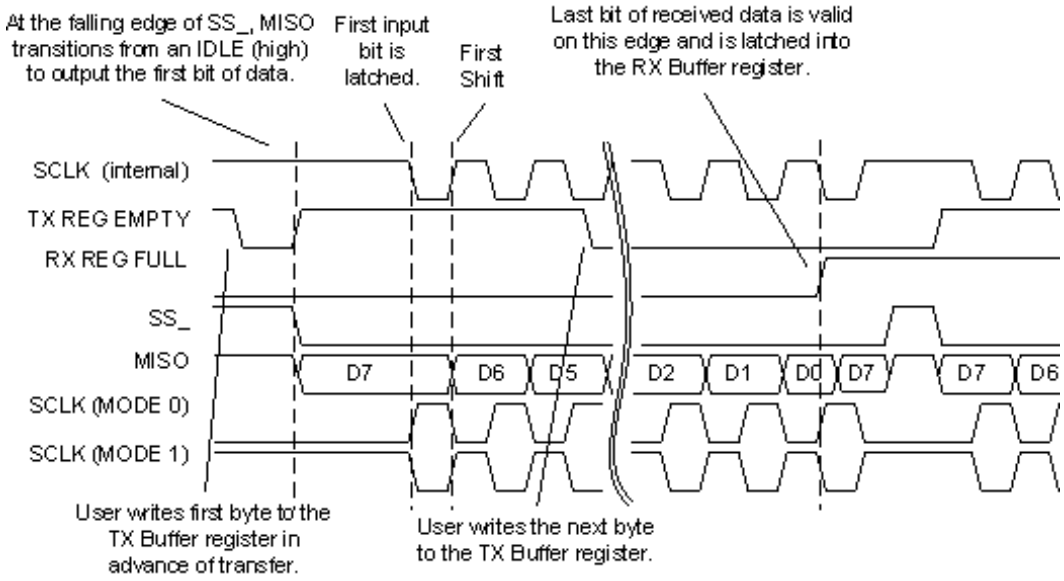
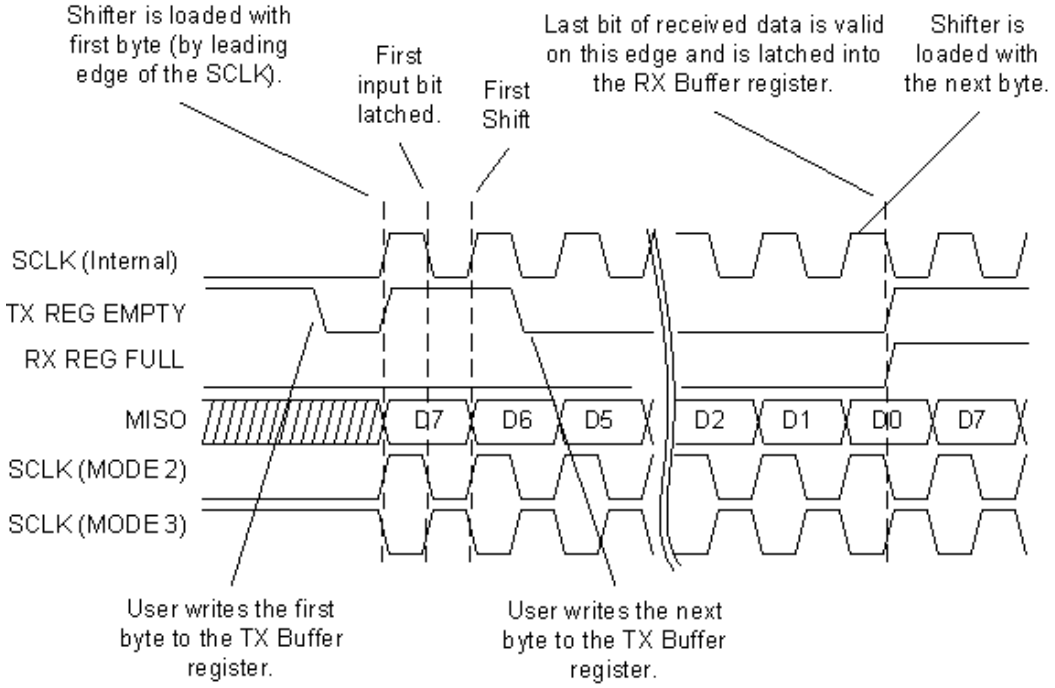


Figure 3. 模式 2 和 3 的典型 SPIS 时序



[+] Feedback

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	限值	单位
F_{InMax}	最大输入 SCLK 频率	4.1	8.2	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。 它可置于任意数字通信模块内。

参数和资源

SCLK

SPIS 的时钟由 SPI 主控生成的 SCLK 信号提供。 此信号来自可用的源之一。 提供 SCLK 输入时可指定高、低、全局 I/O 总线、模拟比较器总线或另一个 PSoC 模块。 此时钟定义了有效位传送速率。 默认情况下，SCLK 与 SysClk 同步。 如果 SCLK 要与 SysClk*2 同步或不同步，则应直接写入输出寄存器的 ClockSync 位域。 可用的时钟和时钟频率因 PSoC 器件而异。

MOSI

主出从入输入信号来自 16 个可能的源之一。 提供 MOSI 输入时可指定高、低、全局 I/O 总线、模拟比较器总线或另一个 PSoC 模块。

~SS

从器件选择输入信号来自 4 个可能的全局输入之一。 此外，~SS 同时也可能受固件控制。 使用 SW_SlaveSelect 时，从器件选择信号将会置入到用户模块的启动上。 使用硬件 SS 时不会监控此实例。

MISO

主入从出输出信号可路由到一条全局输出总线。 然后，该全局输出总线可连接至外部引脚或另一个 PSoC 模块，以进行下一步处理。

中断模式

此选项决定了何时为 TX 模块产生中断。 “TxRegEmpty” 选项能够让中断在数据从数据寄存器传输至移位寄存器后立刻发生。 选择第二个选项 “TxComplete” 将会延迟中断，直到最后位移出移位寄存器。 如果需要知道字符在何时全部发送，第二个选项会很有帮助。 第一个选项 “TxRegEmpty” 最适合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。

InvertMOSI

使用此参数，用户可以反转 MOSI 输入。

中断产生控制 (Interrupt Generation Control)

当选中 PSoC Designer 中的 **“ 启用中断产生控制 ”** 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：

- 中断 API (Interrupt API)
- IntDispatchMode

[+] Feedback

InterruptAPI

InterruptAPI 参数允许有条件地生成一个用户模块的中断处理程序和中断矢量表入口。选择 “ 启用 ”(Enable) 以生成中断处理程序和中断矢量表条目。选择 “ 禁用 ”(Disable) 以取消生成中断处理程序和中断矢量表条目。在那些拥有多个程序层而且有多个程序层使用同一模块资源的项目中，特别推荐要正确地选择是否要生成中断 API。仅在必要时选择生成中断 API，这样可以避免生成中断调度代码，从而减少开销。

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程序层正处于活动状态。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” 参数会导致固件只在最初已经有一个程序层运行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

应用程序编程接口 (API) 子程序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。本部分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 API 中的一样，A 和 X 寄存器的值可能因调用 API 函数而更改。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）。选择这种 “ 寄存器易变 ” 策略是为了提高效率，并且自从 PsoC Designer 的 1.0 版本起使用。C 编译器自动遵循此要求。汇编语言程序员也必须确保其代码遵守这一策略。虽然有些用户模块的 API 函数可能保留 A 和 X 不变，但并不保证在未来也将如此。

本节列出了 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，应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。此操作应在用户提供的子程序中完成。

C 原型：

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

汇编：

```
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及其相关值。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。而且还要注意，用户模块的实例名称加在下表列出的符号名称前面。例如，如果在放置用户模块时将其命名为 SPIS1，那么第一个模式的符号名称为 SPIS1_SPIM_MODE_0.

[+] Feedback

符号名	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：

此函数可能更改 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 原型：

```
void SPIS_Stop(void)
```

汇编：

```
lcall  SPIS_Stop
```

参数：

无

返回值：

无

副作用：

此函数可能更改 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 原型：

```
void SPIS_EnableInt(void)
```

汇编：

```
lcall  SPIS_EnableInt
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

SPIS_DisableInt

说明：

在 “SPI 完成 ” 条件下禁用 SPIS 中断。

C 原型：

```
void SPIS_DisableInt(void)
```

汇编：

```
lcall  SPIS_DisableInt
```

参数：

无

返回值：

无

副作用：

此函数可能更改 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 原型：

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

汇编：

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

参数：

bTxData： 要发送到 SPI 主控器件并传送到累加器的数据。

返回值：

无

副作用：

此函数可能更改 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。 调用此子程序之前应检查 “Rx 缓冲区已满 ” 标志，确认已收到数据字节。

C 原型：

```
BYTE SPIS_bReadRxData(void)
```

汇编：

```
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：

此函数可能更改 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 原型：

```
BYTE SPIS_bReadStatus(void)
```

汇编：

```
lcall  SPIS_bReadStatus
and   A, SPIS_DONE | 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。 利用定义的掩码来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
SPI_COMPLETE	0x20
RX_OVERRUN_ERROR	0x40
BUFFER_EMPTY	0x10
RX_BUFFER_FULL	0x08

[+] Feedback

副作用：

调用此函数之后将会清除状态位。 此函数可能更改 A 和 X 寄存器。

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，必须将名为 “ 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，则此函数可能无法生效。

C 原型：

```
void SPIS_DisableSS(void)
```

汇编：

```
lcall SPIS_DisableSS
```

参数：

无

返回值：

无

副作用：

此函数可能更改 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，必须将名为 “ 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，则此函数可能无法生效。

C 原型：

```
void SPIS_EnableSS(void)
```

汇编：

```
lcall SPIS_EnableSS
```

参数：

无

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

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 to be performed in the non-interrupt
;  processing.  This code could easily be written to be
;  interrupt driven.
;
;  It is assumed that the SPI Slave User Module is setup properly
;  and started.
;
;*****************************************************************
include  "SPIS.inc"
export   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);
    }
}
```

配置寄存器

用于配置此用户模块的 A 型数字通信 PSoC 模块寄存器描述如下。 仅解释参数化符号。

Table 3. SPIS 模块： 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	0	0	1	1	1	0

此寄存器定义将此 “A” 型数字通信模块配置为 SPIS 用户模块。

Table 4. SPIS 模块： 寄存器输入

位	7	6	5	4	3	2	1	0
值	MOSI				SCLK			

MOSI 是主出从入输入信号。 SCLK 是来自 SPI 主控的时钟信号。 二者均在参数选择过程中使用器件编辑器进行设置。

Table 5. SPIS 模块： 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	ClockSync		~SS		MISO		

~SS 是从器件选择输入信号。 MISO 是主入从出输出信号。 二者均在参数选择过程中使用器件编辑器进行设置。

Table 6. SPIS 模块： 移位寄存器 DR0

位	7	6	5	4	3	2	1	0
值	移位寄存器							

此移位寄存器是 SPI 移位寄存器。

Table 7. SPIS 模块： TX 数据缓冲区寄存器 DR1

位	7	6	5	4	3	2	1	0
值	TX 缓冲区寄存器							

Tx 缓冲区寄存器： 启用 PSoC 模块后，写入此缓冲区的数据将传输到移位寄存器。

Table 8. SPIS 模块： RX 数据缓冲区寄存器 DR2

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 9. SPIS 模块： 控制寄存器 CR0

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 缓冲区为空	Rx 缓冲区已满	时钟相位	时钟极性	SPIS 启用

最低有效位优先指定最低有效位应优先发送。

“Rx 过速错误 ” 标志表示接收到新字节时尚未读取之前接收到的数据字节。

“SPI 完成 ” 标志表示完整的 SPI 传输 / 接收循环已完成。

“Tx 缓冲区为空 ” 标志表示 Tx 缓冲区为空。

“Rx 缓冲区已满 ” 标志表示已从移位寄存器接收到数据字节。

“ 时钟相位 ” 表示 SCLK 信号的相位。 它是定义 SPI 模式的参数之一。

“ 时钟极性 ” 表示 SCLK 信号的极性。 它是定义 SPI 模式的参数之一。

“SPIS 启用 ” 经设置后可启用 SPIS PSoC 模块。

版本历史记录

版本	创作者	说明
2.5	DHA	更新的从器件选择参数。
2.60	DHA	添加了对 CY8C21x12 器件的支持。

Note PSoC Designer 5.1 在所有用户模块数据表中引入了版本历史。 此部分记录了当前用户模块版本和以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.