SPI 从器件可变长度数据表 SPISVL V 1.0

Copyright © 2009-2010 Cypress Semiconductor Corporation. All Rights Reserved.

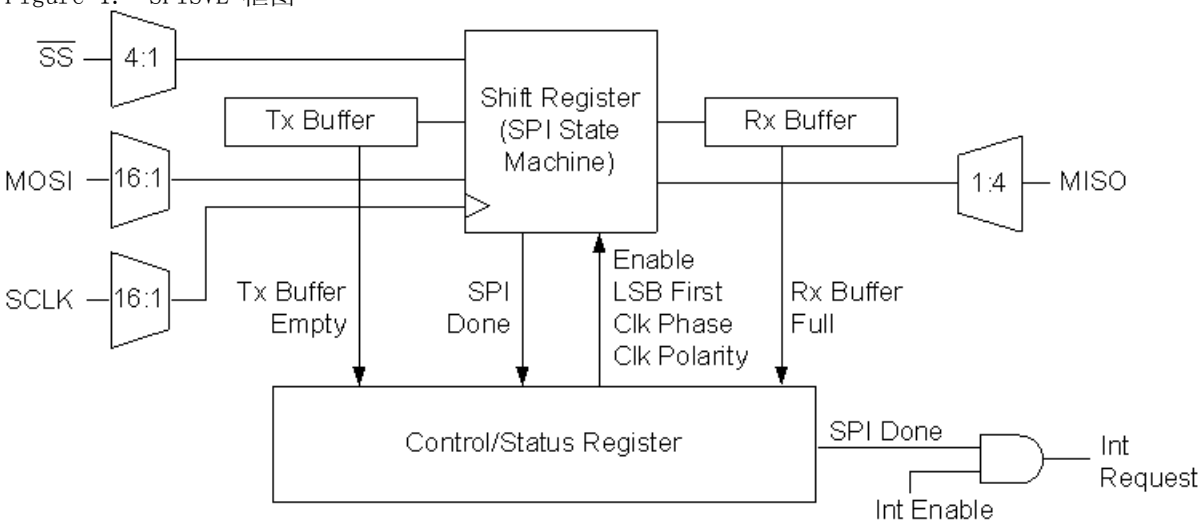
资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C21x45, CY8C22x45, CY8C28x45, CY8C28xxx						
	2	0	0	142	1	4

功能和概述

- 支持串行外设互联 (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
- 可对中断编程，使其发生在 “SPI 完成 ” 或 “TX 寄存器为空 ” 条件下
- SS 可以为受控固件
- 可选择数据长度：9 到 16 位

SPISVL 用户模块是串行外设互联从器件，可配置为可变数据长度。 该模块使用从 9 到 16 位之间的任意数据长度，执行全双工同步数据传输。 可以指定 SCLK 相位、SCLK 极性和 LSB 优先，以适应大多数 SPI 协议。 SPISVL PSoC 模块为输入与输出信号提供可选择的路由，并拥有可编程中断驱动控制。 应用程序编程接口 (API) 固件为汇编或 C 语言应用软件均提供高级编程接口。

Figure 1. SPISVL 框图



功能说明

SPISVL 是一个实施串行外设互联从器件的用户模块。 由于该模块支持 9-16 位数据长度，因此其成对使用两个数字通信类型 PSoC 模块的 Tx 缓冲区、Rx 缓冲区、Control0、Control1 和移位寄存器以及一个或多个引脚端口寄存器。

使用器件编辑器和 / 或 SPISVL 用户模块固件应用程序编程接口 (API) 例程对 Control0、Control1 寄存器进行初始化与配置。 初始化包括设置 LSB 优先的配置，设置 SPI 接收 / 传输时钟模式，以及设置数据长度。 支持 SPI 模式 0、1、2 和 3。 SPI 主控与从器件必须使用相同的时钟模式和位配置进行设置，以进行正常通信。 SPI 模式按以下方式定义。

Table 1. SPI 模式

模式	执行数据锁存的 SCLK 沿	时钟极性	注
0	前	同相	前沿锁存数据。 数据在时钟后沿更改。
1	前	反相	
2	后	同相	下降沿锁存数据。 数据在上升沿更改。
3	后	反相	

SCLK 输入信号是由 SPI 主控生成的 SPI 传输 / 接收时钟。 该信号定义已传输 / 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。 通常情况下，多个从器件的 MISO 信号将绑定在一起。 在激活从器件选择信号前，每个从器件会把其各自的 MISO 信号置为三态。 此用户模块不会把 MISO 输出信号置为三态。 如果需要，用户可将此功能轻松添加到用户模块。

SPISVL 硬件从 MOSI 信号上的主控 SPI 器件接收数据，同时向 MISO 信号上的 SPI 主控器件传输数据。 主控与从器件数据的传输和接收使用相同的 SCLK 信号。

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，以便为有效通信启用特定的 SPIS 器件。

由主控决定所选从器件是否可以执行命令或接收数据。

当使用 API 例程在 LSB 模块的 Control0 寄存器中设置 “SPI 启用 ” 位后，将启用 SPISVL 用户模块的操作功能。

要传输到 SPI 主控的数据会被写入到 Tx 缓冲区寄存器。 这将清除 LSB 模块的 “Tx 缓冲区为空 ” 状态位。

在从器件选择信号的下降沿，数据将从 Tx 缓冲区寄存器传输到移位寄存器。 要传输的数据字的第一位随后将置入到 MISO 输出信号上。 此时，另一个要传输的数据字将被写入 Tx 缓冲区寄存器。 当前字传输完毕后，就可以把此数据发送到 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。 将设定 “Rx 缓冲区已满 ” 和 “SPI 完成 ” 状态位。 如果已启用了中断，“SPI 完成 ” 状态位会触发中断。 此中断可用于提醒软件，数据的一个字节已接收，或已成功传输。

如果 Tx 缓冲区寄存器中当前装载了未处理的数据字，那么此字将可以在主控执行下一次数据操作时进行传输。

[+] Feedback

如果 “SPI 完成 ” 中断条件未用于从 Rx 缓冲区寄存器中检索数据字，应轮询 Control0 寄存器以监控 “Rx 缓冲区已满 ” 状态位。 在完全接收下一个数据字或设定 “ 溢出错误 ” 状态位前，必须从 Rx 缓冲区寄存器中读取已接收的数据。

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

如果使用中断，必须在每个中断之后读取 SPISVL 状态寄存器，以识别下一个中断。 读取状态寄存器将清除由中断控制器识别的内部信号。 如果此信号仍较高，后续 SPISVL 中断将被屏蔽。

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_{max}	最大 RX/TX 频率	--	12	MHz

放置

SPISVL 映射到两个 PSoC 模块，本文档视其为一个结构群。 SPISVL 可放置在同一行的任意一对数字通信模块中。

参数和资源

SCLK

SPISVL 的时钟由 SPI 主控生成的 SCLK 信号提供。 此信号来自 15 个可能的源之一。 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。 此时钟用于定义有效位元传输率。 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 “ 时钟同步 ” 位域。

MOSI

主出从入输入信号来自 15 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。

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。

中断类型

此选项用于确定为 TX 模块生成中断的时间。 “Tx 寄存器为空 ” 选项可在数据从数据寄存器传输到移位寄存器之后立刻生成中断。 选择第二个选项 “SPI 完成 ” 可推迟中断，直到最后一个位元从移位寄存器移出为止。 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 “Tx 寄存器为空 ” 最适合用于最大限度提高发射器的输出。 该选项允许一边发送上一字节一边加载新字节。

反转 MOSI

使用此参数，用户可以反转 MOSI 输入。

[+] Feedback

数据长度

此参数用于设置 SPI 通讯协议的数据长度。

中断生成控制

当选中 PSoC Designer 中的 **“ 启用中断生成控制 ”** 复选框时，将有另外两个参数可供使用。 此选项可通过以下路径找到： **项目 > 设置 > 芯片编辑器**. 当使用多个外覆层并且外覆层上的多个用户模块共享中断时，中断生成控制非常重要：

- 中断 API
- 中断调度模式

中断 API

“ 中断 API” 参数可允许按条件生成用户模块的中断处理程序和中断矢量表条目。 选择 “ 启用 ” 以生成中断处理程序和中断矢量表条目。 选择 “ 禁用 ” 以取消生成中断处理程序和中断矢量表条目。 对于具有多个外覆层并且不同外覆层共享一个模块源的项目，强烈建议要正确选择是否生成中断 API。 仅在必要时选择生成中断 API，可能就避免了生成中断调度码的需求，从而降低开销。

中断调度模式

“ 中断调度模式 ” 参数用于指定中断请求的处理方式，这些中断由同一模块不同外覆层中的多个用户模块共享。 选择 “ 活动状态 ”，固件会在处理共享中断请求之前检测哪个外覆层处于活动状态。 每次请求共享中断时都会执行此检测。 这样会增加延迟，还会产生一个处理共享中断请求的非决定性程序，但不需要任何 RAM。 选择 “ 计算前偏移 ”，固件仅在初次加载外覆层时计算共享中断请求的源。 这种计算可减少中断延迟，并产生一个处理共享中断请求的决定性程序，但会占用 RAM 空间。

应用程序编程接口

应用程序编程接口 (API) 例程作为用户模块的一部分提供，从而使设计人员能够采用更高级的方式处理模块。 本节指定每个函数的接口，以及引用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 寄存器的值可以通过调用 API 函数发生更改。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 此 “ 寄存器易失 ” 策略是针对提高效率的目的选择，自 PSoC Designer 1.0 版起已强制使用此策略。 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。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

另请注意，常量和函数名称中的 “SPISVL” 字样会替换为用户模块的实例名称。 例如，如果您在放置用户模块时将其命名为 SPISVL1，则第一个模块的符号名为 SPISVL_1_MODE_0，SPISVL_Start 函数变为 SPISVL_1_Start.。

下面列出了 SPISVL 提供的 API 函数。

SPISVL_Start

说明：

设置 SPI 接口的模式配置，并通过在 Control0 寄存器中设置适当的位来启动 SPISVL 模块。 调用此函数之前，所有从器件选择信号均应置为高电平，以取消选中已连接的 SPI 从器件。 此操作应在用户提供的例程中完成。

C 原型：

```
void  SPISVL_Start(BYTE bConfiguration)
```


[+] Feedback

汇编：

```
mov   A,SPISVL_MODE_2 | SPISVL_LSB_FIRST
lcall  SPISVL_Start
```

参数：

bConfiguration: 用于指定 SPI 模式和 LSB 优先配置的一个字节。 下列列出了 C 语言和汇编语言中使用的符号名及其相关值。

符号名	值
SPISVL_MODE_0	0x00
SPISVL_MODE_1	0x02
SPISVL_MODE_2	0x04
SPISVL_MODE_3	0x06
SPISVL_LSB_FIRST	0X80
SPISVL_MSB_FIRST	0X00

返回值：

无

副作用：

请参阅 API 一节开头的注意 **。

SPISVL_Stop

说明：

通过清除 Control0 寄存器中的启用位来禁用 SPISVL 模块。

C 原型：

```
void SPISVL_Stop(void)
```

汇编：

```
lcall  SPISVL_Stop
```

参数：

无

返回值：

无

副作用：

请参阅 API 一节开头的注意 **。

SPISVL_EnableInt

说明：

在 “SPI 完成 ” 条件下启用 SPISVL 中断。 SPISVL 的放置位置决定了具体的中断矢量和优先级。

[+] Feedback

C 原型：

```
void SPISVL_EnableInt(void)
```

汇编：

```
lcall  SPISVL_EnableInt
```

参数：

无

返回值：

无

副作用：

请参阅 API 一节开头的注意 **。

SPISVL_DisableInt

说明：

在 “SPI 完成 ” 条件下禁用 SPISVL 中断。

C 原型：

```
void SPISVL_DisableInt(void)
```

汇编：

```
lcall  SPISVL_DisableInt
```

参数：

无

返回值：

无

副作用：

请参阅 API 一节开头的注意 **。

SPISVL_ClearInt

说明：

清除已发布的 SPISVL 中断。

C 原型：

```
void  SPISVL_ClearInt(void);
```

汇编：

```
lcall  SPISVL_ClearInt
```

参数：

无

返回值：

无

副作用：

请参阅 API 一节开头的注意 **。

SPISVL_SetupTxData

说明：

把要传输到 SPI 主控中的数据字节写入到 Tx 缓冲区寄存器。

C 原型：

```
void SPISVL_SetupTxData(WORD wTxData)
```

汇编：

```
mov   X, [wTxData]
mov   A, [wTxData+1]
lcall  SPISVL_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：

请参阅 API 一节开头的注意 **。

SPISVL_wReadRxData

说明：

返回从从器件接收到的数据字。 调用此例程之前应检查 “Rx 缓冲区已满 ” 标志，确认已收到数据字。

C 原型：

```
WORD SPISVL_wReadRxData(void)
```

汇编：

```
lcall  SPISVL_wReadRxData
mov   [wRxData], X
mov   [wRxData+1], A
```

参数：

无

返回值：

从从器件 SPI 接收并返回到累加器的数据字。

副作用：

请参阅 API 一节开头的注意 **。

SPISVL_bReadStatus

说明：

读取并返回当前 SPISVL 控制 / 状态寄存器。

[+] Feedback

C 原型：

```
BYTE SPISVL_bReadStatus(void)
```

汇编：

```
lcall  SPISVL_bReadStatus
and    A, SPISVL_COMPLETE | SPISVL_RX_BUFFER_FULL
jnz    SPIVL_COMPLETE_GET_RX_DATA
```

参数：

无

返回值：

返回已读取的状态字节并返回至累加器中。 使用已定义的掩码测试特定状态条件。 请注意，可将掩码并列组合在一起，用于测试多个条件。

SPIM 状态掩码	值
SPISVL_COMPLETE	0x20
SPISVL_RX_OVERRUN_ERROR	0x40
SPISVL_TX_BUFFER_EMPTY	0x10
SPISVL_RX_BUFFER_FULL	0x08

副作用：

调用此函数后，状态位将被清除。 A 和 X 寄存器可能被此函数更改。

SPISVL_DisableSS

说明：

当不需要外部从器件选择信号（ ~SS ）时，使用固件将低电平有效 ~SS 信号设为高电平。 为了使用此函数，必须将 SPI 的 “ 从器件输入 ” 参数值设为 “SW_SlaveSelect”，而不能使用行输入中的值。 如果已连接外部信号，则此函数可能无法生效。

C 原型：

```
void SPISVL_DisableSS(void)
```

汇编：

```
lcall SPISVL_DisableSS
```

参数：

无

返回值：

无

副作用：

请参阅 API 一节开头的注意 **。

SPISVL_EnableSS

说明：

当不需要外部从器件选择信号（ ~SS ）时，使用固件将低电平有效 ~SS 信号设为低电平。 为了使用此函数，必须将 SPI 的 “ 从器件输入 ” 参数值设为 “SW_SlaveSelect”，而不能使用行输入中的值。 如果已连接外部信号，则此函数可能无法生效。

C 原型：

```
void SPISVL_EnableSS(void)
```

汇编：

```
lcall SPISVL_EnableSS
```

参数：

无

返回值：

无

副作用：

请参阅 API 一节开头的注意 **。

固件源代码示例

本示例展示如何创建 SPI 环路。 从 SPI 主控接收到数据后会作出回应。

```
void main(void)
{
WORD wData;

SPISVL_Start(SPISVL_MODE_0|SPISVL_MSB_FIRST);

M8C_EnableGInt;

while(1)
{
while( !(SPISVL_bReadStatus() & SPISVL_RX_BUFFER_FULL) ); // wait for received data
wData = SPISVL_wReadRxData(); // read received data from the Rx buffer

while( !(SPISVL_bReadStatus() & SPISVL_TX_BUFFER_EMPTY) ); // check for Tx buffer empty
SPISVL_SetupTxData(wData); // send echo packet back

while( !(SPISVL_bReadStatus() & SPISVL_SPI_COMPLETE) ); // encure transaction is
complete
}
}

The same example in ASM will look like following:

export _main

area bss(RAM, REL)

 wData: blk 2
```


[+] Feedback

```
area text (ROM, REL)

_main:

mov A, (SPISVL_MODE_0|SPISVL_MSB_FIRST)
lcall SPISVL_Start

M8C_EnableGInt

.nextLoop:

.WaitRx:
lcall SPISVL_bReadStatus
and A, SPISVL_RX_BUFFER_FULL
jz .WaitRx

lcall SPISVL_wReadRxData;
push A
push X

.WaitTx:
lcall SPISVL_bReadStatus
and A, SPISVL_TX_BUFFER_EMPTY
jz .WaitTx

pop X
pop A

lcall SPISVL_SetupTxData

.WaitComplete:
lcall SPISVL_bReadStatus
and A, SPISVL_SPI_COMPLETE
jz .WaitComplete
jmp .nextLoop
```

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. SPIMVL 模块，寄存器： FunctionMSB 优先配置：

位	7	6	5	4	3	2	1	0
LSB	反转 MOSI	0	0	中断类型	1	1	1	0
MSB	0	0	0	0	1	1	1	0

Table 4. SPIMVL 模块，寄存器： FunctionLSB 优先配置：

位	7	6	5	4	3	2	1	0
LSB	0	0	0	中断类型	1	1	1	0
MSB	反转 MOSI	0	0	0	1	1	1	0

[+] Feedback

此寄存器定义数字通信模块成为 SPISVL 用户模块所需的特性。

Table 5. SPISVL 模块：寄存器 InputMSB 优先配置：

位	7	6	5	4	3	2	1	0
LSB	MOSI				SCLK			
MSB					SCLK			

Table 6. SPISVL 模块：寄存器 InputLSB 优先配置：

位	7	6	5	4	3	2	1	0
LSB					SCLK			
MSB	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 7. SPISVL 模块：寄存器 OutputMSB 优先配置：

位	7	6	5	4	3	2	1	0
LSB	0	0	0	~SS				
MSB	0	0	0			MISO		

Table 8. SPISVL 模块：寄存器 OutputLSB 优先配置：

位	7	6	5	4	3	2	1	0
LSB				~SS		MISO		
MSB								

~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 9. SPISVL 模块：移位寄存器 DR0

位	7	6	5	4	3	2	1	0
LSB	移位寄存器							
MSB	移位寄存器							

Table 10. SPISVL 模块：TX 数据缓冲区寄存器 DR1

位	7	6	5	4	3	2	1	0
LSB	TX 缓冲区寄存器							
MSB	TX 缓冲区寄存器							

[+] Feedback

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 11. SPISVL 模块： RX 数据缓冲区寄存器 DR2

位	7	6	5	4	3	2	1	0
LSB	RX 缓冲区寄存器							
MSB	RX 缓冲区寄存器							

RX 缓冲区寄存器： SPI 传输循环完成后，移位寄存器中收到的数据将传输到该寄存器。

Table 12. SPISVL 模块： 控制寄存器 CR0

位	7	6	5	4	3	2	1	0
LSB	LSB 优先	RX 溢出错误	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
MSB	LSB 优先	RX 溢出错误	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

“LSB 优先 ” 指定应首先传输 LSB 位。

“Rx 溢出错误 ” 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。 该参数是用于定义 SPI 模式的参数之一。

“ 时钟极性 ” 表示 SCLK 信号的极性。 该参数是用于定义 SPI 模式的参数之一。

“SPIS 启用 ” 经设置后可启用 SPIS PSoC 模块。

Table 13. SPISVL 模块： 控制寄存器 CR1

位	7	6	5	4	3	2	1	0
LSB	1	1	0	SPI 长度				
MSB	1	0	0	SPI 长度				

“SPI 长度 ” 指定链模式中 SPI 的长度。

[+] Feedback

版本历史记录

版本	创作者	说明
1.0	DHA	初始版本

Note PSoC Designer 5.1 在所有的用户模块数据表中提供版本历史记录。 本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2009-2010 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.