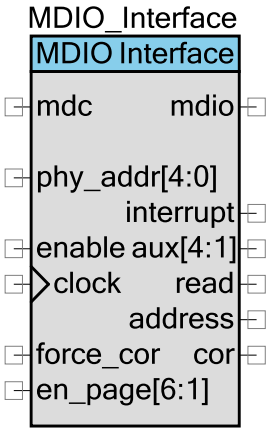
MDIO 接口

1.10



性能

- 与以太网产品结合使用
- 可配置的物理地址
- 支持高达 4.4 MHz 的总线时钟（mdc）
- 符合 IEEE 802.3 标准中的第 45 条
- 自动将存储器分配给寄存器空间。通过直观且易于使用的图形配置 GUI 可以配置该寄存器空间。

概述

MDIO 接口组件支持管理数据输入/输出。该输出/输出是用于媒体独立接口（MII）的 IEEE 802.3 标准中以太网系列所定义的串行总线。该 MII 将媒体访问控制（MAC）器件连接到以太网物理层（PHY）电路。该组件符合 IEEE 802.3 标准中的第 45 条。

何时使用 MDIO 接口

在一个 PHY 管理接口中，使用 MDIO 接口组件读取和写入 PHY 控制和状态寄存器。它在运行之前对每个 PHY 进行配置，并在运行期间监控链接的状态。

可配置该组件，使之为来自 MDIO 总线的所有帧生成中断。在基本模式下，允许用户在固件中进行数据处理算法。在高级模式下，可配置该组件，使之自动处理硬件中的所有寄存器，而无需 CPU 干预。

输入/输出连接

本节介绍了 MDIO 接口组件的各种输入和输出连接。I/O 列表中的星号（*）表示，在 I/O 说明部分所列出的内容中，该 I/O 可能不可见。

mdc — 输入

MDC 是由 MDIO 主机提供的总线时钟。它直接连接到物理 MDC 输入引脚。对于连接到 mdc 输入的引脚，将 **Sync Mode**（同步模式）参数（在引脚组件的“Configure”对话框中）设置为“Transparent”（透明）。

clock — 输入

该时钟供给内部逻辑时序使用。当组件在 **Basic mode**（基本模式）下运行时，需要设置时钟速率至少为 MDC 总线时钟的 8 倍。而在 **Advanced mode**（高级模式）下运行时，建议使用最低配置频率为 40 MHz 的 BUS_CLK 作为组建的时钟。

phy_addr[4:0] — 输入*

物理端口地址。若将 **Physical address**（物理地址）参数设置为 **Hardware**（硬件），将会显示该地址。

mdio — Inout(双向) *

用于传输或接收数据的双向引脚。对于连接到 mdio 的引脚，应将 **Drive Mode**（驱动模式）设置为“Open Drain Low”（漏极开路），并将 **Sync Mode** 设为“Transparent “（透明）。如果取消选中 **Enable external OE** 参数，则将会显示 mdio 终端。

mdio_in — 输入*

由主机驱动的 MDIO 信号。如果选中 **Enable external OE**，则将会显示该信号。

mdio_out — 输出*

由 MDIO 接口驱动的 MDIO 信号，在选中 **Enable external OE** 选项时将被显示出来。

enable — 输入

同步高电平有效使能信号。

interrupt — 输出

在 **Basic mode**（基本模式）下进行配置时，只有物理地址和器件地址与先前配置的值相匹配时，该输出才会在帧结束后生成脉冲。

但在 **Advanced mode**（高级模式）下，当 MDIO 主机结束写入操作时，以及配置了相关的寄存器，使之在写操作发生时触发中断后，该输出会生成一个脉冲。

force_cor — 输入*

对当前的 MDIO 地址进行读取时强制进行清除操作。如果将 **Configuration** 参数设置为 **Advanced**，将会显示该信号。

en_page[x] — 输入*

启用/禁用相应的寄存器空间。禁用此页后，组建会在读取帧的数据部分为高电平，同时忽略写入帧的数据。该信号的宽度取决于组件所管理的寄存器空间的编号。如果将 **Configuration** 参数设置为 **Advanced**，将会显示该信号。

read — 输出*

当 MDIO 主机结束一个读取操作，并且配置了相关寄存器，使之在读取操作发生时触发中断后，该输出会生成一个脉冲。如果将 **Configuration** 参数设置为 **Advanced**，将会显示该信号。

address — 输出*

每当 MDIO 主机发送地址帧结束时，该输出将生成一个脉冲。如果将 **Configuration** 参数设置为 **Advanced**，将会显示该信号。

cor — 输出*

当接收到一个读取帧，且将寄存器配置为“读取时清除”时，该输出将生成一个脉冲。如果将 **Configuration** 参数设置为 **Advanced**，将会显示该信号。

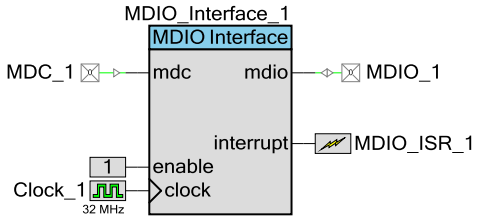
aux[4:1] — 输出*

该信号输出与所访问的寄存器相关的补助位，并在选中 **Enable auxiliary bits** 选项时显示出来。

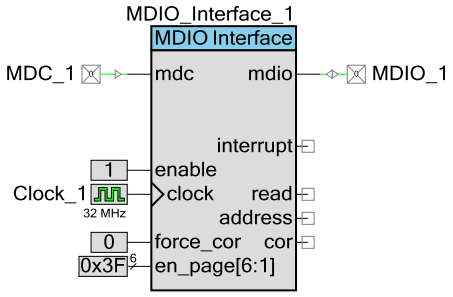
原理图宏信息

PSoC Creator 组件目录提供了用于基本和高级操作模式的两个宏，如下图所示。这两个宏含有了已与数字引脚、时钟和一个中断相连的 MDIO 接口组件。对于 mdc 和 mdio 引脚，**Sync Mode** 参数（在引脚组件的“Configure”对话框中）被设置为“Transparent”（透明）。

基本模式



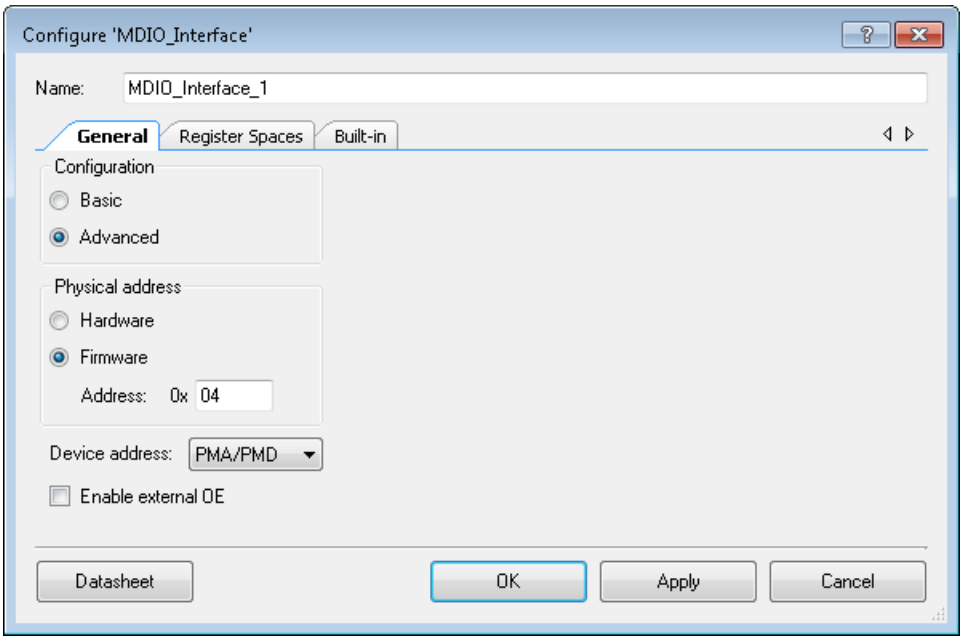
高级模式



组件参数

将一个 MDIO 接口组件拖入您的设计中，然后双击它以打开 **Configure** 对话框。该对话框中有两个选项卡：Basic（基本）和 Register Spaces（寄存器空间）（当选中“Advanced”配置时出现）。

Basic（基本）选项卡



Configuration （配置）

确定组件运作的模式：**Basic**（默认）或 **Advanced**。

Physical Address（物理地址）

确定通过 **phy_addr[4:0]**端口地址总线还是固件 API 更新物理地址。如果选中 **Firmware** 选项，则用来确定默认的物理地址。地址模式的选项包括 **Firmware** 或者 **Hardware**。此地址值范围介于 0 到 0x1F 间。默认设置值为 **0x04**。

Device Address（器件地址）

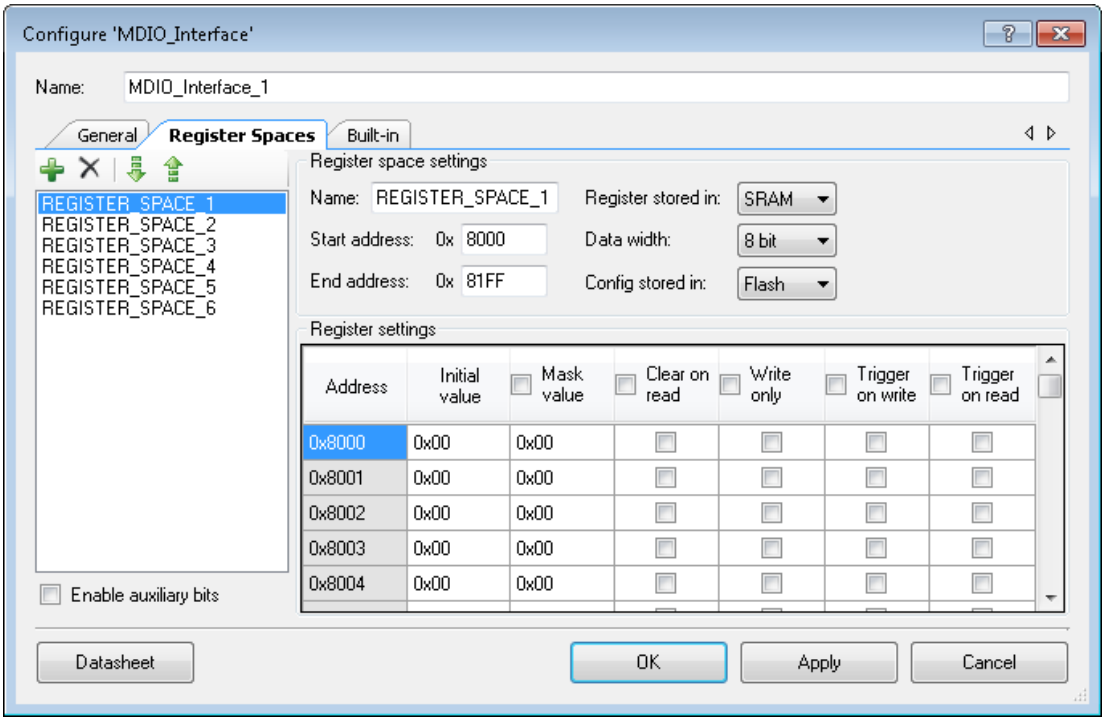
指定 MDIO 接口的器件类型。此值可以设置为 **PMA/PMD**、**WIS**、**PCS**、**PHY XS** 或 **DTE XS**。默认设置为 **PMA/PMD**。

Enable external OE（使能外部 OE）

允许您将 **mdio** 双向信号分开为 **mdio_in** 输入和 **mdio_out** 输出信号。选项 = 选中或取消选中。使能此选项，可在符号上将 **mdio** inout 信号公开为 **mdio_in** 输入和 **mdio_out** 输出。（默认值 = 取消选中）。

Register Spaces（寄存器空间）选项卡

该选项卡只在 **Advanced mode**（高级模式）下可用，同时也要遵循 CFP MSA 规格中有关 CFP 寄存器分配的规定。通过该选项卡，可对各个 CFP 寄存器空间进行分配，并配置分配寄存器空间所涉及的每个单一寄存器的参数。



Enable auxiliary bits（使能辅助位）

全局使能所有寄存器空间的辅助位域。选项 = 选中或取消选中。（默认值 = 取消选中）

Name（名称）

文本字段，24 个字符，用于为快速访问寄存器空间的起始地址和结束地址创建各个宏。该字段的默认标签为 **REGISTER_SPACE_N**（其中，N=1…**寄存器空间编号**）。

Start address（起始地址）

给每个分配寄存器空间确定十六进制的起始地址。其取值范围为 0x8000 到 0xFFFF。

End address（结束地址）

给每个分配寄存器空间确定十六进制的结束地址。该地址值介于 0x8000 到 0xFFFF 之间，并且必须大于相应寄存器空间的起始地址。

Register stored in（寄存器的存储区）

指定寄存器空间在存储器中的位置。寄存器空间的数据可以存储在 **Flash** 或 **SRAM**（默认）内。

Data width（数据宽度）

指定数据宽度为 8 位或 16 位。

Config stored in（配置存储位置）

指定与寄存器空间相应的配置数据所在存储器的位置。寄存器空间的配置可存储在 **Flash** 或 **SRAM**（默认）中。

请注意，每个寄存器空间的配置包含四个参数字节，用于该寄存器空间的每个寄存器。

Initial value（初始值）

指定寄存器的十六进制初始值。根据相应寄存器空间的 **Data width**（数据宽度）参数，该值可以是 8 位或 16 位。（默认值 = 0）

Mask value（掩码值）

指定寄存器的可写位掩码（十六进制）。根据相应寄存器空间的 **Data width**（数据宽度）参数，该值可以是 8 位或 16 位。若相应的寄存器空间位于闪存内，将忽略该值。（默认值 = 0）

Clear on read（读取时清除）

指定读取时清除(COR)的寄存器。选项 = 选中或取消选中。若选中该项，则在 MDIO 主机读取时将寄存器清除为 0。若相应的寄存器空间位于闪存内，将忽略该值。（默认值 = 取消选中）

请注意，读取时清除操作的宽度为 16 位。如果将 8 位寄存器配置为 COR，则对该寄存器进行读取操作将清除其内部的值，以及存储在后续存储器地址的值。

Write only（只写）

将寄存器定义为只写(WO)。选项 = 选中或取消选中。如果 MDIO 主机读取某个 WO 寄存器，则 MDIO 帧的数据部分的返回值将为 0xFFFF。若相应的寄存器空间位于闪存内，将忽略该值。（默认值 = 取消选中）。

Trigger on write（写入时触发）

当 MDIO 主机对相应的寄存器进行的写操作完成时，该项用于指定 **interrupt** 输出是否生成脉冲。选项 = 选中或取消选中。若相应的寄存器空间位于闪存内，将忽略该值。（默认值 = 取消选中）

Trigger on read（读取时触发）

当 MDIO 主机对相应的寄存器进行的读取操作完成时，该项用于指定 **read** 输出是否生成脉冲。选项 = 选中或取消选中。若相应的寄存器空间位于闪存内，将忽略该值。（默认值 = 取消选中）

Aux bits（辅助位）

当 MDIO 主机访问相应的寄存器时，定义了公开给 **aux[4:1]**输出的 4 位二进制数值。当需要直接向其他硬件指示特定的寄存器访问时，这些位是必有的。若选中 **Enable auxiliary bits**（使能辅助位）选项，将显示该值。若相应的寄存器空间位于闪存（默认值 = 0）内，该值将被忽略。

时钟选择

该器件中没有内部时钟。所以您必须添加一个时钟源。当组件在 **Basic mode**（基本模式）下运行时，所提供的时钟速率至少要等于所需 MDC 时钟速率的 8 倍。在 **Advanced mode**（高级模式）下，建议使用最低配置频率为 40 MHz 的 BUS_CLK 作为组建的时钟。

应用编程接口

通过应用编程接口（API），您可以使用软件进行配置组件。下表列出并说明了每个函数的接口。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 将实例名称“MDIO_Interface_1”分配给指定设计中组件的第一个实例。您可以将该实例重命名为符合标识符语法规则的任意唯一值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。为便于阅读，下表使用的实例名称为“MDIO_Interface”。

函数	说明
MDIO_Interface_Start()	初始化并使能MDIO接口。
MDIO_Interface_Stop()	禁用MDIO接口。
MDIO_Interface_Init()	初始化随自定义程序所提供的默认配置。
MDIO_Interface_Enable()	使能MDIO接口。
MDIO_Interface_EnableInt()	使能中断输出终端。
MDIO_Interface_DisableInt()	禁用中断输出终端。
MDIO_Interface_SetPhyAddress()	给MDIO接口设置5位物理地址。
MDIO_Interface_UpdatePhyAddress()	更新给MDIO接口的物理地址。
MDIO_Interface_SetDevAddress()	给MDIO接口设置5位器件地址。
MDIO_Interface_GetData()	返回存储在给定地址的值。

函数	说明
MDIO_Interface_SetData()	设置给定地址中的参数值。
MDIO_Interface_SetBits()	在指定的地址中设置指定位。
MDIO_Interface_GetAddress()	返回MDIO主机所写入的最后地址。
MDIO_Interface_GetConfiguration()	返回指向给定寄存器空间中配置阵列的指针。
MDIO_Interface_PutData()	设置将要传输给MDIO主机的数据。
MDIO_Interface_ProcessFrame()	处理从MDIO主机接收的最后一帧数据。
MDIO_Interface_Sleep()	停止MDIO接口并保存用户配置。
MDIO_Interface_Wakeup()	恢复用户配置并使能MDIO接口。
MDIO_Interface_SaveConfig()	保存当前的用户配置。
MDIO_Interface_RestoreConfig()	恢复用户配置。

全局变量

变量	说明
MDIO_Interface_initVar	MDIO_Interface_initVar指出MDIO接口是否已被初始化。该变量将被初始化为0，并在第一次调用MDIO_Interface_Start()时将其设置为1。这样，第一次调用MDIO_Interface_Start()例程后，组件不用重新初始化即可重启。 如需重新初始化器件，可在MDIO_Interface_Start()或MDIO_Interface_Enable()函数前调用MDIO_Interface_Init()函数。

变量	说明

MDIO_Interface_registerConfig_N[]
（仅适用于高级模式）
	保存与寄存器空间N相关的寄存器配置。N=1…8 — 组件所分配的寄存器空间编号。可以通过本文档后面部分API常量一节中所描述的相应API常量内容，对上述任何阵列进行直接访问。
	请注意，每个寄存器空间的配置军包含了四个参数字节，用于该寄存器空间的每个寄存器。按照下面内容将这些参数存储到C结构内：
	`typedef struct`
	`{`
	`uint16 mask; /* 16 bit writable mask */`
	`uint8 ctrlReg; /* Register configuration */`
	`uint8 reserved; /* Reserved */`
	`}`
	掩码：确定主机的寄存器位的访问类型。0 — 读。1 — 读/写。组件API的所有位都是读/写位。
	ctrlReg：寄存器配置位
	位0：读取时生成脉冲
	位1：写入时生成脉冲
	位2：只写
	位3：读取时清除
	位7-4：辅助位（aux[4:1]）
	对于每个已分配的寄存器空间，会根据定制器中的寄存器设置初始化MDIO_Interface_registerConfig_N[]。一般情况下，不需要对此些阵列进行访问或更改。

MDIO_Interface_registerSpace_N[] （仅适用于高级模式）	保存寄存器空间N的寄存器值。N=1…8 — 组件分配的寄存器空间编号。可以通过本文档后面部分API常量一节中所说明的相应API常量内容，对上述任何阵列进行直接访问。访问上述阵列的首先方法是，通过MDIO_GetData()函数获取寄存器值，并通过MDIO_SetData()函数修改MDIO_Interface_registerSpace_N[]阵列中的寄存器值。

void MDIO_Interface_Start(void)

说明： 这是开始执行组件操作的首选方法。此函数设置initVar变量，调用MDIO_Interface_Init()函数，然后调用MDIO_Interface_Enable()函数。

参数： 无

返回值： 无

其他影响： 如果已经设置了initVar变量，则该函数仅调用MDIO_Interface_Enable()函数。

void MDIO_Interface_Stop(void)

说明： 禁用MDIO接口。如果将该组件配置为高级模式，则该函数将禁用所有内部DMA通道。

参数： 无

返回值： 无

其他影响： 无

void MDIO_Interface_Init(void)

说明： 初始化或恢复随定制器所提供的默认MDIO接口配置。如果将该组件配置为高级模式，则该参数会初始化内部的DMA通道。通常不需要调用MDIO_Interface_Init()，因为MDIO_Interface_Start()子程序会调用此函数，这是开始组件操作的首选方法。

参数： 无

返回值： 无

其他影响： 无

void MDIO_Interface_Enable(void)

说明： 激活硬件，并开始执行组件操作。无需调用MDIO_Interface_Enable()，因为MDIO_Interface_Start()子程序会调用该函数，这是开始组件操作的首选方法。

参数： 无

返回值： 无

其他影响： 如果MDIO_Interface_DisableInt() API禁用了终端输出中断，可通过此函数使能该中断。

void MDIO_Interface_EnableInt(void)

说明： 使能终端输出中断。

参数： 无

返回值： 无

其他影响： 无

void MDIO_Interface_DisableInt(void)

说明： 禁用终端输出中断。

参数： 无

返回值： 无

其他影响： 无

void MDIO_Interface_SetPhyAddress(uint8 phyAddr)

说明： 设置MDIO接口的5位物理地址。

参数： uint8 phyAddr：物理地址值。

返回值： 无

其他影响： 覆盖定制器中所定义的默认物理地址。

void MDIO_Interface_UpdatePhyAddress(void)

说明： 根据组件的当前phy_addr[4:0]更新物理地址。如果将物理地址的模式参数设置为“Firmware”，会设置地址值，使之等于定制器中的默认值。

参数： 无

返回值： 无

其他影响： 无

void MDIO_Interface_SetDevAddress(uint8 devAddr)

说明： 设置MDIO接口的5位器件地址。

参数： uint8 devAddr：器件地址值。

返回值： 无

其他影响： 覆盖定制器中指定的默认器件地址。

uint8 MDIO_Interface_GetData(uint16 address, uint16 *regData, uint16 numWords)

说明： 自给定地址开始返回N值。如果出现不属于所分配的寄存器空间的任何地址，此函数将返回一个错误。此API只在高级模式下可用。

参数： uint16 address：需要访问的地址。

uint16 *regData：包含读取数据的阵列。

uint16 numWords：要读取的字数。

返回值： uint8 status

值	说明
CYRET_SUCCESS	成功
CYRET_BAD_PARAM	一个或多个无效参数
CYRET_TIMEOUT	操作超时

其他影响： 无

uint8 MDIO_Interface_SetData(uint16 address, const uint16 *regData, uint16 numWords)

说明： 自给定地址开始写入N值。如果出现不属于所分配的寄存器空间的地址，或寄存器空间位于闪存中，则该函数会返回一个错误。仅当至少一个寄存器空间位于SRAM，并在高级模式下运行时，此API才可用。

参数： uint16 address：需要访问的地址。

const uint16 *regData：要写入的数据阵列。

uint16 numWords：要写入的字数。

返回值： uint8 status

值	说明
CYRET_SUCCESS	成功
CYRET_BAD_PARAM	一个或多个无效参数
CYRET_TIMEOUT	操作超时

其他影响： 无

uint8 MDIO_Interface_SetBits(uint16 address, uint16 regBits)

说明： 设置给定地址中的位。如果出现分配寄存器空间外的地址，或寄存器空间位于闪存中，则此函数会返回一个错误。仅当至少一个寄存器空间位于SRAM内，并在高级模式下运行时，此API才可用。

参数： uint16 address：需要访问的地址。

uint16 regBits：要设置的位。

返回值： uint8 status

值	说明
CYRET_SUCCESS	成功
CYRET_BAD_PARAM	一个或多个无效参数
CYRET_TIMEOUT	操作超时

其他影响： 无

uint16 MDIO_Interface_GetAddress(void)

说明： 返回MDIO主机所写入的最后的地址。

参数： 无

返回值： uint16：地址。

其他影响： 无

uint8 MDIO_Interface_GetConfiguration(uint8 regSpace)

说明： 返回指向给定寄存器空间中所配置阵列的指针。

参数： uint8 regSpace：寄存器空间索引。

返回值： uint8*：指向配置阵列的指针。

其他影响： 无

uint8 MDIO_Interface_PutData(uint16 regData)

说明： 将给定数据写入内部FIFO内，并在下一帧将其传送到主机上。只在基本模式下可用。

参数： uint16 regData：需要发送的数据。

返回值： 无

其他影响： 无

void MDIO_Interface_ProcessFrame(uint8* opCode, uint16* regData)

说明： 处理和解析从主机接收到的最后一帧数据。只在基本模式下可用。

参数： uint8* opCode：操作代码。

uint16* regData：接收到的寄存器数据。

返回值： 无

其他影响： 无

void MDIO_Interface_Sleep(void)

说明： 这是组件准备进入睡眠模式的首选子程序。MDIO_Interface_Sleep()子程序保存当前的组件状态。然后它调用MDIO_Interface_Stop()函数和 MDIO_Interface_SaveConfig()来保存硬件配置。

在调用CyPmSleep()或CyPmHibernate()函数之前先调用MDIO_Interface_Sleep()函数。有关功耗管理函数的详细信息，请参考PSoC Creator *《系统参考指南》*。

参数： 无

返回值： 无

其他影响： 无

void MDIO_Interface_Wakeup(void)

说明： 该函数是将组件恢复到调用MDIO_Interface_Sleep()时状态的首选子程序。MDIO_Interface_Wakeup()函数调用 MDIO_Interface_RestoreConfig()函数以恢复配置。如果组件在调用MDIO_Interface_Sleep()函数前已启用，则MDIO_Interface_Wakeup()函数将重新启用组件。

参数： 无

返回值： 无

其他影响： 调用MDIO_Interface_Wakeup()函数前未调用MDIO_Interface_Sleep()或MDIO_Interface_SaveConfig()函数可能会产生意外行为。

void MDIO_Interface_SaveConfig(void)

说明： 此函数会保存组件配置以及非保留寄存器。它还保存“Configure”对话框中定义的或通过相应API修改的当前组件参数值。该函数由MDIO_Interface_Sleep()函数调用。

参数： 无

返回值： 无

其他影响： 无

void MDIO_Interface_RestoreConfig(void)

说明： 此函数会恢复组件配置以及非保留寄存器。它还将组件参数值恢复为在调用MDIO_Interface_Sleep()函数之前的值。

参数： 无

返回值： 无

其他影响： 如果在调用MDIO_Interface_SaveConfig()之前调用了API，则组件配置将恢复为其默认设置。

API 常量

变量[1]	说明
MDIO_Interface_ADDRESS	地址帧的操作代码
MDIO_Interface_WRITE	写入帧的操作代码
MDIO_Interface_READ	读取帧的操作代码
MDIO_Interface_POS_READ	发布读取帧的操作代码
MDIO_Interface_NUMBER_OF_PAGES	已分配的寄存器空间编号
MDIO_Interface_RegSpaceName_IDX	寄存器空间索引
MDIO_Interface_RegSpaceName_START	寄存器空间的起始地址
MDIO_Interface_RegSpaceName_END	寄存器空间的结束地址
MDIO_Interface_RegSpaceName_PTR	阵列的外部参考，该阵列用于保存相应寄存器空间内的寄存器数据。
MDIO_Interface_RegSpaceName_CONFIG_PTR	阵列的外部参考，该阵列用于保存相应寄存器空间的配置数据。

1. RegSpaceName — 定制器中输入给定寄存器空间的名称。

MISRA 合规性

本节介绍了 MISRA-C:2004 合规性和本组件的偏差情况。定义了两种类型的偏差：

- 项目偏差 — 适用于所有 PSoC Creator 组件的偏差
- 特定偏差 — 仅适用于该组件的偏差

本节介绍了有关组件特定偏差的信息。在 *《系统参考指南》* 的“MISRA 合规性”章节中介绍了项目偏差以及有关 MISRA 合规性验证环境的信息。

此 MDIO 接口组件具有如下特定偏差：

MISRA-C:2004 规则	规则类别（必须/建议）	规则说明	偏差说明
11.4	建议	不能对不同对象指针类型进行转换。	不同对象指针类型间的转换用于配置DMA数据移动时将一个CFP寄存器空间映射到组件内部存储器中。
16.7	建议	如果没有使用指针进行更改寻址的对象，则应将函数原型中的指针参数声明为指向常量的指针。	该指针寻址该API中配置的DMA传输所更改的对象。
19.7	建议	函数应优先于类函数宏使用。	由于使用了类函数以提高代码生成效率，出现了偏差。 组件在闪存或SRAM中分配了8位或16位寄存器。通过使用带参数的宏，可以快速确定寄存器宽度和存储器位置。

此组件具有以下嵌入式组件 — DMA。MISRA 合规性与特定偏差的相关信息，请参见相应的组件数据手册。

样例固件源代码

PSoC Creator 在“Find Example Project”对话框中提供了包括原理图和代码示例的许多示例项目。要查看特定组件实例，请打开“Component Catalog”中的对话框或者原理图中的组件样例。要查看通用示例，请打开“Start Page”或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多有关信息，请参见《PSoC Creator 帮助》中的“Find Example Project”（查找示例项目）主题。

参考

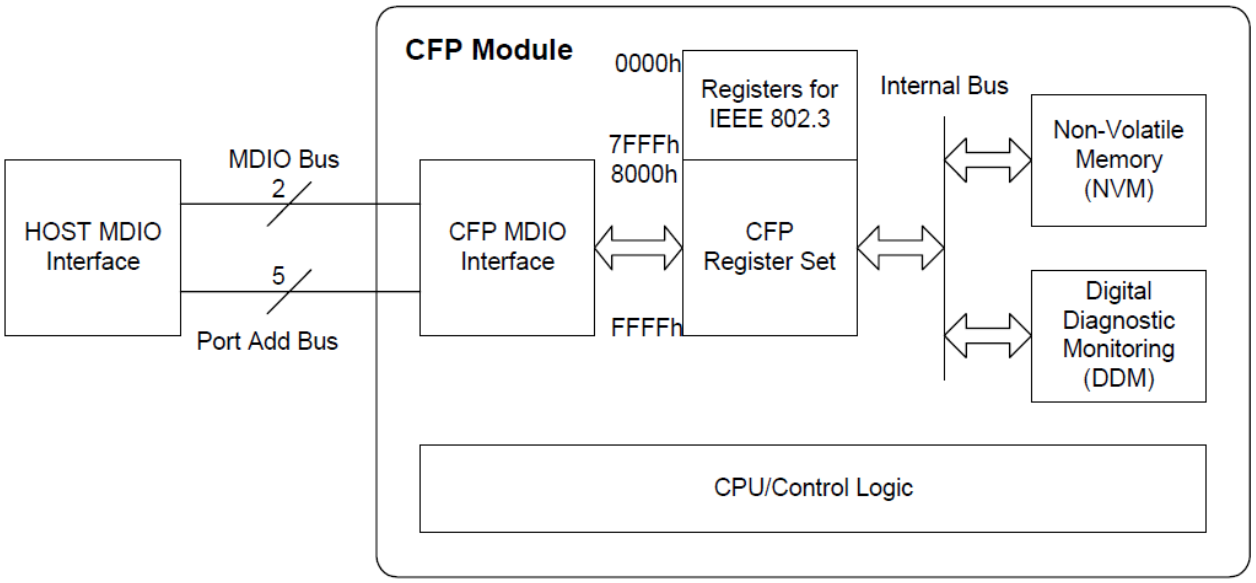
CFP MSA 管理接口规格版本 2.0

CFP MSA 硬件规格版本 1.4

功能说明

CFP 模块概述

MDIO_Slave 组件支持规定媒体独立（MII）特性的 IEEE 802.3 标准中的第 45 条。该组件能与 CFP 管理接口结合使用。该接口将 MDIO 总线指定为主机与 CFP 模块间的管理接口。CFP 模块框图如下图所示。



MDIO 主机通过此接口控制和监控所管理的 CFP 模块的启动、关闭以及正常运行等操作。在同一个总线上，多个从器件可以通过不同的总线地址连接至同一个主机。可通过端口地址总线配置该模块的地址。

此组件接口通过两条导线连接到主机：即作为 MDIO 总线使用的数据线和作为 MDC 使用的时钟线。MDIO 总线是双向的，并按照 MDC 中指定的速率传输数据。而 MDC 时钟线则由主机驱动，以便控制通信速率。

一旦被使能，此 MDIO 接口组件将一直监控 MDIO 主机器件，并与其进行通信。在基本模式下，可在每个 MDIO 帧结束时触发一个中断，从而允许用户通过所需的 API 将 FIFO 中的该捕获数据传输到系统存储器内。高级模式对数据包的信息进行检查，并允许将数据包传入/传出系统存储器。

在高级模式下运行时，MDIO 组件会根据定制器设置自动分配 CFP 寄存器表。该组件支持以下寄存器访问类型：

- 支持只读（RO）寄存器
- 支持可读/写（R/W）寄存器
- 在 16 位字工作环境下，支持 RO 和 R/W 的混合访问
- 支持读取时清除（COR）
- 支持只写（WO）寄存器

当 MDIO 主机发送一个读/写帧时，这些操作均由硬件执行。

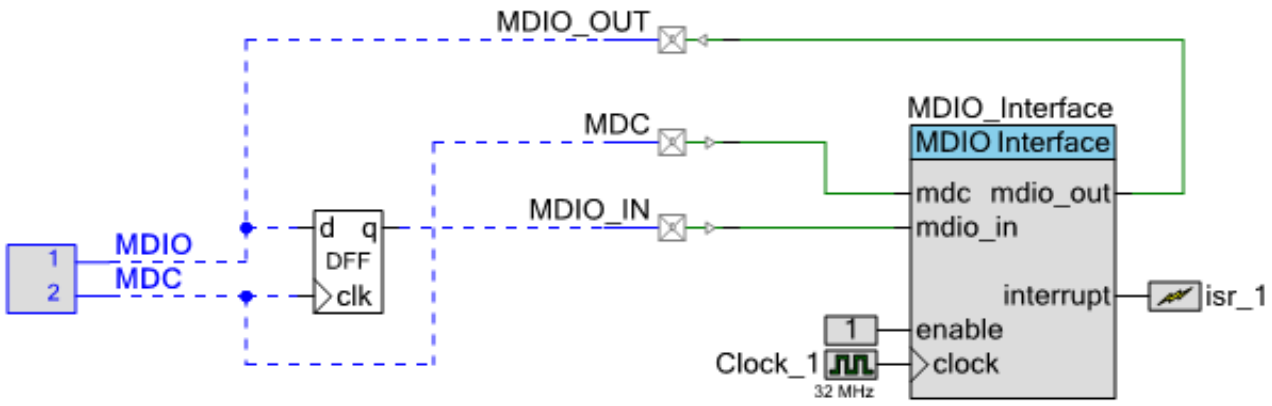
如果 MDIO 主机读取一个只写寄存器，则会向主机返回 0xFFFF。如果 MDIO 主机写入一个只读寄存器或某些只读位，MDIO 接口组件将忽略整个帧或该只读位。

对 0x0000 到 0x7FFF 地址范围进行的所有访问都被忽略（MDIO 总线上无活动）。

将双向 mdio 引脚分成 mdio_in 和 mdio_out 终端

通过配置该组件，可以将 **mdio** 双向终端分为 **mdio_in** 和 **mdio_out** 两个终端。当多个 MDIO 从器件复用 MDIO 总线时，此性能非常有用。为了实现该操作，请在 **Configure** 对话框中的 **General** 选项卡下，选中 **Enable external OE** 复选框。

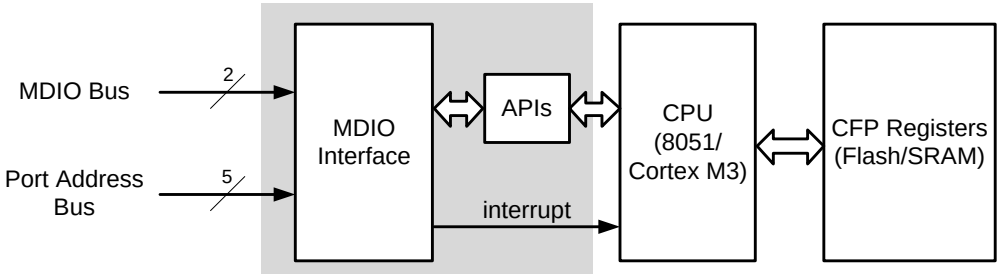
使用外部 DFF 时，应该使用一个额外引脚，以便将 MDIO 输入和输出信号分开。下图显示的是 DFF 连接到 PSoC 引脚的情况。请注意，应将 MDIO_OUT 引脚配置为 **Open Drain, Drives Low**（开漏驱动低电平） 数字输出。同时，将 MDIO_IN 和 MDC_IN 配置为 **High Impedance**（高阻抗）数字输入。



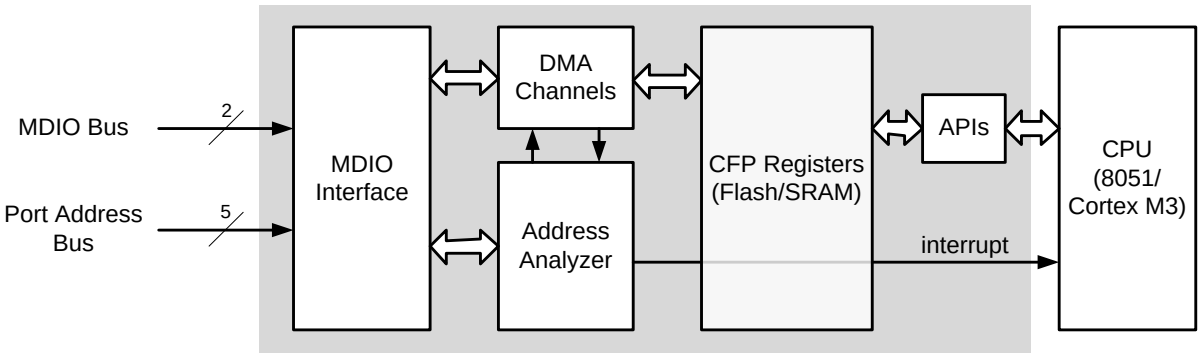
框图和配置

MDIO 接口组件作为一组已配置的 UDB 实现。当配置为高级模式时，该组件会通过 DMA 传入/传出系统 SRAM/闪存。下面框图显示了实现过程。

MDIO 接口（基本模式）



MDIO 接口（高级模式）



寄存器

MDIO 接口组件有几个控制和状态寄存器。固件 API 使用这些寄存器控制操作和监控状态。用户固件不能直接访问其中任何寄存器。

资源

MDIO 接口组件放置在整个 UDB 阵列中。当配置为高级模式时，该组件会通过 DMA 传入/传出系统 SRAM/闪存。该器件使用以下资源。

配置	资源类型					
	数据路径单元	宏单元	状态单元	控制单元	DMA通道	中断
基本	2	16	2	2	–	–
高级	8	59	2	5	9 [2]	–

API 存储器占用

根据不同编译程序、器件、所使用的 API 数量以及组件的配置情况，组件所用的存储空间大小也不一样。下表提供了给定组件配置中的所有 API 存储器使用。

通过发布模式中所配置的相应编译程序来完成测量操作。在发布模式下，可以得到最优化的尺寸。有关特定的设计，可分析编译器生成的映射文件以确定存储器使用情况。

配置	PSoC 3 （Keil_PK51）		PSoC 4（GCC）		PSoC 5LP（GCC）	
	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节	闪存 字节	SRAM 字节
基本	365	5	N/A	N/A	460	7
高级 [3]	4.3K + R_{FLS}	53 + R_{SRAM}	N/A	N/A	2.8K + R_{FLS}	86 + R_{SRAM}

2. PSoC 5L 的实现要求一个额外的 DMA 通道。共有 10 个 DMA 通道供给 PSoC 5LP 使用。

3. 在高级模式下，组件将自动分配和管理CFP寄存器表。R_{FLS}和R_{SRAM}分别是闪存和SRAM存储寄存器数据和配置所使用的空间。该空间可通过以下公式计算得出：
   R_{SRAM} = 16*PAGE_NUM + REG_NUM_SRAM$_{8\text{-}BIT}$ + 2*REG_NUM_SRAM$_{16\text{-}BIT}$ + 4*REG_NUM_CFG_SRAM;
   R_{FLS} = 8*PAGE_NUM + REG_NUM_FLS$_{8\text{-}BIT}$ + 2*REG_NUM_FLS$_{16\text{-}BIT}$ + 4*REG_NUM_CFG_FLS, where:
   PAGE_NUM — 已分配的寄存器页数；
   REG_NUM_SRAM$_{8\text{-}BIT}$和REG_NUM_SRAM$_{16\text{-}BIT}$ — 存储在SRAM中的8位和16位寄存器数量；
   REG_NUM_FLS$_{8\text{-}BIT}$和REG_NUM_FLS$_{16\text{-}BIT}$ — 存储在闪存中的8位和16位寄存器数量；
   REG_NUM_CFG_SRAM — 存储在SRAM中并带配置的寄存器数量；
   REG_NUM_CFG_FLS — 存储在闪存中并带配置的寄存器数量；

直流和交流电的电气特性

除非另有说明，否则这些规范的适用条件是：-40°C ≤ T_A ≤ 85 °C 和 T_J ≤ 100 °C。除非另有说明，否则这些规范的适用范围为 1.71 V 到 5.5 V。

直流电特性

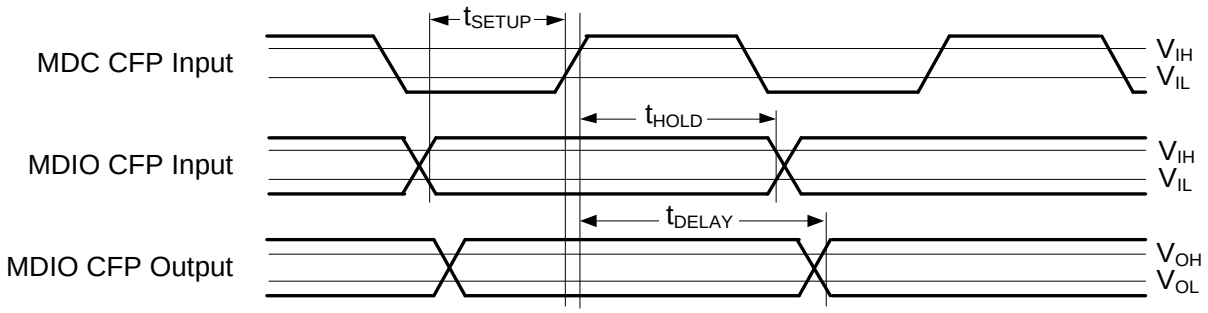
参数	说明	最小值	典型值	最大值	单位
V_{IH}	输入高电平电压	0.8	–	5.5	V
V_{IL}	输入低电平电压	-0.5	–	0.42	V
V_{OH} [4]	输出高电平电压（I_{OH} = -100 µA）	–	–	–	V
V_{OL}	输出低电平电压（I_{OL} = 100 µA）	0.001	–	0.002	V
I_{OH} [5]	输出高电流（V_{OH} = 1.0 V）	–	–	–	mA
I_{OL}	输出低电流（V_{OL} = 0.2 V）	10	–	–	mA
C_i	输入电容	–	–	7	pF

交流电特性

参数	说明	最小值	典型值	最大值	单位
f_{MDC}	MDC时钟频率	–	–	4.4	MHz
f_{CLOCK}	组件时钟频率	8 * f_{MDC}	42	–	MHz
t_{SETUP}	主机MDIO建立时间				
	取消选中外部OE	10	–	–	ns
	选中外部OE [6]	2.4	–	–	ns
t_{HOLD}	主机MDIO保持时间				
	取消选中外部OE	10	–	–	ns
	选中外部OE [6]	0	–	–	ns
t_{DELAY}	CFP MDIO延迟时间	–	–	150	ns

4. PSoC输出驱动器为漏极开路，因此MDIO总线的上拉电阻决定V_{OH}电压大小。
5. I_{OH}不适用于漏极开路驱动器。
6. 该参数适用于MDIO总线的一个外部D触发器。在默认情况下，当选中**external OE**（外部OE）时，组件所需的t_{HOLD}为85 ns。为满足CFP MSA硬件规格中所指定的保持时间，需要通过外部D触发器（DFF）（如Texas Instruments的 SN74AUP1G79或Fairchild Semiconductor的NC7SP74）捕获总线上的MDIO数据，并将其传输到PSoC。有关组件与外部DFF的正确连接的信息，请参考**将双向mdio引脚分为mdio_in和mdio_out两个终端**部分。

MDC/MDIO 时序图



如何将 STA 结果使用于特性数据

MDIO 设置和保持时间

STA 不直接提供设置和保持时间。不过，STA 结果中提供的数据指示了某些可用于计算时序的内部逻辑时序限制。为符合您的系统对时序的需求，MDC 和 MDIO 间的关系要满足以下公式：

$$-t_{H_CFP} + t_{S_MC} \le t_{DELAY} \le t_{S_CFP}$$

其中：

t_{DELAY} = 组件输入上 MDIO 路径延时与 MDC 路径延时间的差异

t_{S_CFP} = MDIO 主机器件提供的设置时间要求

t_{H_CFP} = MDIO 主机器件提供的保持时间要求

t_{S_MC} = PSoC 宏单元（3.51 ns）的设置时间要求

可在下面 STA 报告的“输入到时钟”部分中查找 MDIO 路径延时与 MDC 路径延时间的差异。（连接到 mdc 和 mdio 组件输入的引脚名称分别为 MDC 和 MDIO）：

- **Input To Clock Section**
  - **MDC(0)_PAD**

Source	Destination	Delay (ns)
MDIO(0)_PAD:in	\MDIO_Interface:bMDIO:mdio_reg\/main_0	1.377

设置时间的富裕空间为（t_{S_CFP} - t_{DELAY}），保持时间的富裕空间为（t_{H_CFP} + t_{DELAY} - t_{S_MC}）。例如，如果 t_{DELAY} = 1.377 ns，t_{S_CFP} = 10 ns 和 t_{H_CFP} = 10 ns，那么设置时间的富裕空间约达 8.6 ns，保持时间的富裕空间为 7.9 ns 左右。

MDIO 延迟时间

MDIO 延迟时间等于组件的所有延时（总共五个时钟周期）和器件的时钟到输出时间（25 ns 左右）之和。为满足 CFP MSA 规格中指定的延迟时间，需要将最低操作频率为 40 MHz 的 BUS_CLK 作为时钟源。这样会产生 150 ns 的延时。

组件更改

本节列出了各版本组件的主要更改。

版本	更改内容	更改/影响的原因
1.10	在定制器中添加了aux[4:1]输出和辅助位支持。	允许直接访问其他硬件指示特定的寄存器。
	改善了逻辑时序模式以满足CFP MSA硬件规格中所规定的设置和保持时间要求。	CFP MSA硬件规格中所指定的时序要求的合规性。
	执行了255 us超时并改善了用于报告SetData()、GetData()和SetBits() API的状态。	提高了SetData()、GetData()和SetBits() API的可靠性。
	当尝试读取无效的寄存器页面时，读取帧数据部分的返回值将为0xFFFF。前一个返回值为0x0000。	符合CFP MSA规格的要求
	删除了PSoC 5支持	PSoC Creator 3.0版本和更高版本不再支持PSoC 5。
1.0.a	更新和更正了数据手册	
1.0	版本1.0是MDIO接口组件的首次发行版	

赛普拉斯半导体公司，2013-2016 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可权）（1）在赛普拉斯特软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。