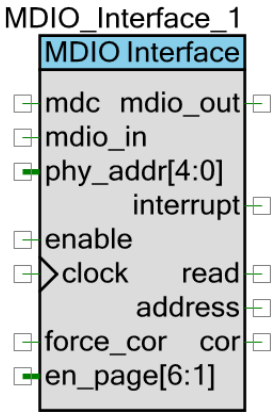
MDIO 接口

1.20



特性

- 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 45 条款
- 自动将存储器分配给寄存器空间。通过直观且易用的图形配置 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 45 条款。

何时使用 MDIO 接口

在 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，允许用户能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/输出接口

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 说明部分中所列出的情况下，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，将引脚组件“Configure”对话框中的 **Sync Mode**（同步模式）参数设置为“Transparent”（透明）。

clock — 输入

该时钟供给内部逻辑时序使用。有关对组件时钟的要求的详细信息，请参考时钟选择一节。

phy_addr — 输入*

物理端口地址。若将 Physical （物理地址）参数设置为 **Hardware**（硬件），将会显示该地址。地址的宽度可以是 3 位或 5 位，具体情况取决于 Address width（地址宽度）参数的设置。如果将地址宽度设置为 3 位，那么组件便能够响应来自某个 MDIO 帧的四个不同物理地址。因此，组件将忽略来自 MDIO 帧的两个最高有效地址位。例如，如果将 3 位物理地址设置为 0x4，则组件将对下面各地址值作出响应：0x04、0x0C、0x14 和 0x1C。

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, drains low”（开漏驱动低电平），并将 **Sync Mode**（同步模式）设为“Transparent”（透明）。如果取消选中 Enable external OE（使能外部 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（使能外部 OE），则将会显示该信号。

mdio_out — 输出*

由 MDIO 接口驱动的 MDIO 信号。如果选中 Enable external OE（使能外部 OE），则将会显示该信号。

enable — 输入

同步高电平有效使能信号。

中断 — 输出

在**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**高级模式**下，当 MDIO 主机结束写入操作时，并配置了相关的寄存器，使之在写操作发生时触发中断后，该输出会生成脉冲。

force_cor — 输入*

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

en_page[x] — 输入*

使能/禁用相应的寄存器空间。禁用该页后，组件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。如果将 Configuration （配置）参数设置为 **Advanced**，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，该输出会生成脉冲。如果将 Configuration （配置）参数设置为 **Advanced**，会显示该信号。

address — 输出*

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

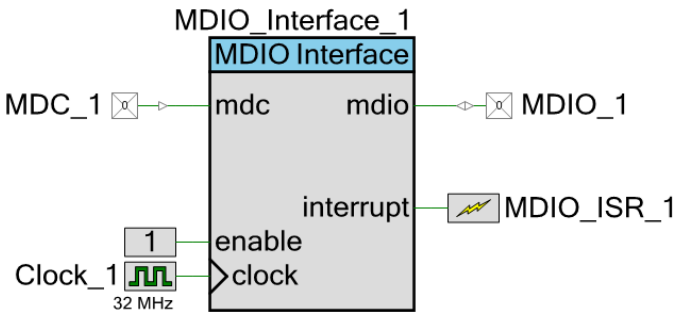
aux[4:1] — 输出*

该信号输出与所访问的寄存器相关的补助位。如果将 Configuration （配置）参数设置为 **Advanced** 并选中 Enable auxiliary bits（使能辅助位）选项，则会显示该信号。

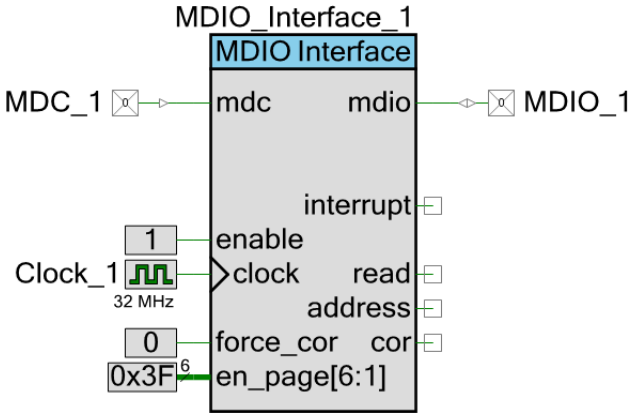
原理图宏

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，将引脚组件“Configure”对话框中的 **Sync mode** 参数设置为“Transparent”。

基本模式



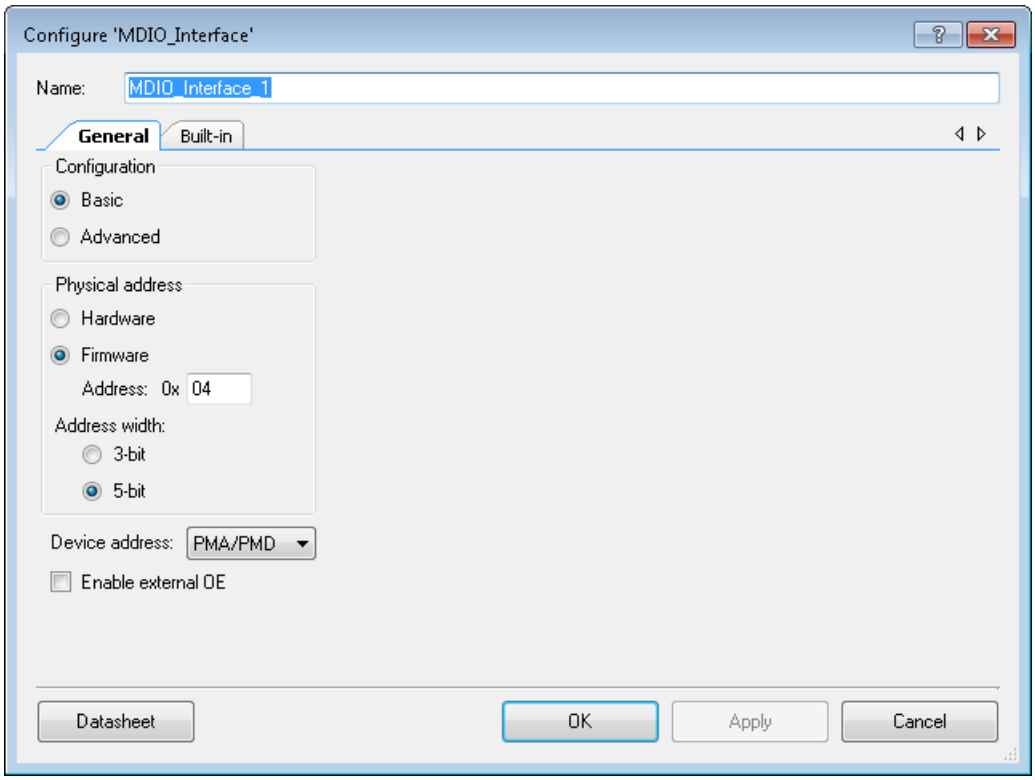
高级模式



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。该对话框包含下列各个选项卡和参数。

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）

决定通过 phy_addr 端口地址总线还是固件 API 更新物理地址。如果选中 **Firmware** 选项，需要确定默认的物理地址。地址模式的选项包括 **Firmware**（默认）或者 **Hardware**。

可以将地址宽度设置为 **3** 位或 **5** 位（默认值）。选择 5 位的物理地址时，地址的取值范围为 0 ~ 0x1F；选择 3 位地址时，该范围则为 0 ~ 7。默认设置值为 **0x04**。

Device address（器件地址）

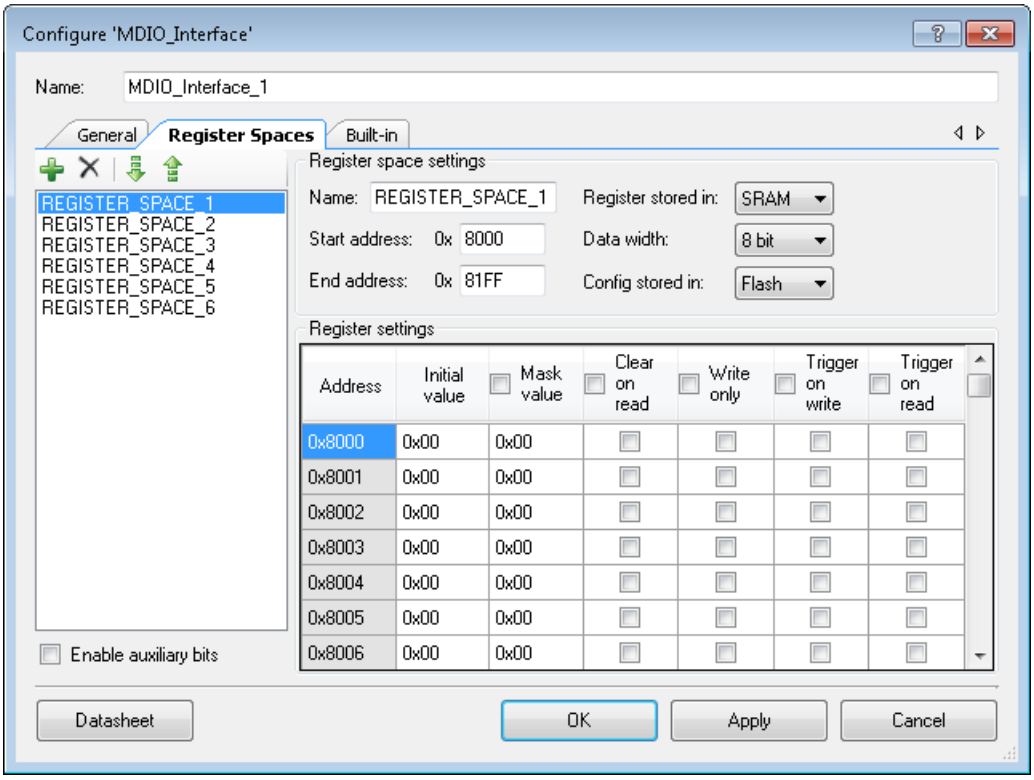
指定 MDIO 接口的器件类型。可将该值设置为 **PMA/PMD**、**WIS**、**PCS**、**PHY XS** 或 **DTE XS**。默认设置为 **PMA/PMD**。

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。使能该选项，可在符号上将 **mdio** inout（双向）信号公开为 **mdio_in** 输入和 **mdio_out** 输出。（默认设置 = 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 寄存器设置的规定。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字段。选项 = 选中或取消选中。（默认设置 = 取消选中）。

Name（名称）

是文本字段，包括 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 C 宏。该字段的默认标签为 **REGISTER_SPACE_N**（其中，N=1 到 **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（寄存器的存储位置）

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中的位置。寄存器空间的配置可存储在 **Flash** 或 **SRAM**（默认）中。

注意：每个寄存器空间的配置都包含 4 个参数字节，供给该寄存器空间的每个寄存器使用。

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）

对读取时被清除的寄存器进行定义。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）。

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‘读取时清除’，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的。选项 = 选中或取消选中。如果 MDIO 主机读取某个只写寄存器，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。（默认设置 = 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，该项用于指定中断输出是否生成脉冲。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。（默认设置 = 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，该项用于指定读取输出是否生成脉冲。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。（默认设置 = 取消选中）。

Aux bits（辅助位）

当 MDIO 主机访问相应的寄存器时，这些位定义了公开给 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。

应用编程接口

通过应用编程接口（API），您可以使用软件对组件进行配置。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了每个函数。

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。可以将该实例重新命名为符合标识符语法规则的任意唯一值。实例名称会成为与该组件相关的每个全局函数名称、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

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。

函数	说明
MDIO_Interface_EnableInt()	使能中断输出端。
MDIO_Interface_DisableInt()	禁用中断输出端。
MDIO_Interface_SetPhyAddress()	设置MDIO接口的物理地址。
MDIO_Interface_UpdatePhyAddress()	更新MDIO接口的物理地址。
MDIO_Interface_SetDevAddress()	设置MDIO接口的器件地址。
MDIO_Interface_GetData()	返回存储在给定地址的值。
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。

void MDIO_Interface_Start(void)

说明： 这是开始执行组件操作的首选方法。该函数将设置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： 初始化或恢复随自定义程序所提供的默认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： 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，可通过该函数使能该中断。

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从设备的5位或3位物理地址。对于3位地址，来自MDIO帧的两个最高有效地址位将被忽略。例如，如果将3位物理地址设置为0x4，则组件将对来自MDIO帧的下面各个物理地址作出响应：0x04、0x0C、0x14和0x1C。

参数： phyAddr：物理地址值。

返回值： 无

其他影响： 覆盖自定义程序中所定义的默认物理地址。

void MDIO_Interface_UpdatePhyAddress(void)

说明： 根据phy_addr输入信号的当前值更新物理地址。如果将 Physical address参数设置为**Firmware**，会设置地址值，使之等于自定义程序中的默认值。

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： devAddr：器件地址值。

返回值： 无

其他影响： 覆盖自定义程序中指定的默认器件地址。

cystatus MDIO_Interface_GetData (uint16 address, const uint16 *regData, uint16 numWords)

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

参数： address：需要访问的地址。

regData：包含读取数据的阵列。

numWords：需要读取的字数。

返回值：

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

其他影响： 无

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

说明： 自特定地址开始写入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，该API才可用。

参数： address：需要访问的地址。

regData：需要写入的数据阵列。

numWords：需要写入的字数。

返回值：

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

其他影响： 无

cystatus 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，该API才可用。

参数： address：需要访问的地址。

regBits：需要设置的位。

返回值：

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

其他影响： 无

uint16 MDIO_Interface_GetAddress(void)

说明： 返回MDIO主机所写入的最后地址。

参数： 无

返回值： 最后写入的地址。

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配置阵列的指针。

参数： regSpace：寄存器空间索引。

返回值： 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： 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： opCode：操作代码。

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时的首选子程序。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()函数仍重新启用组件。

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： 该函数会保存组件配置和非保留的寄存器。它还会保存“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： 该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。

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()后，组件不用重新初始化仍可重启。 如需重新初始化组件，可在调用MDIO_Interface_Start()或MDIO_Interface_Enable()函数前先调用MDIO_Interface_Init()函数。

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均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：表示‘读’，1表示‘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，会根据自定义程序中的寄存器设置初始化MDIO_Interface_registerConfig_N[]。一般情况下，不需要对该些阵列进行访问或更改。

变量	说明
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。访问上述阵列的首选方法是，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。

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。

示例固件源代码

PSoC Creator 在“Find Example Project”（查找示例项目）对话框中提供了很多包括原理图和代码示例的示例项目。要查看特定组件示例，请打开“Component Catalog”中的对话框或原理图中的组件实例。要查看通用示例，请打开起始页或 **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。

更多信息，请参考《PSoC Creator 帮助》部分中主题为“查找示例项目”一节的内容。

1. RegSpaceName — 自定义程序中输入给特定寄存器空间的名称。

MISRA 合规性

本节介绍了 MISRA-C:2004 规范以及本组件的偏差情况。定义了下面两种类型的偏差：

- 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。

该 MDIO 接口组件具有如下特定偏差：

MISRA-C:2004规则	规则类别（必须（R）/建议（A））	规则说明	偏差说明
11.4	建议	指向对象类型的不同指针之间不应进行转换。	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。
19.7	建议	函数应优先于类函数宏使用。	函数宏的偏差允许得到效率更高的代码。 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。

该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，请参见相应组件数据手册。

API 的内存使用量

根据编译器、器件、所使用的 API 数量以及组件的配置情况不同，组件的内存使用量也不一样。下表提供了给定组件配置中的所有 API 所使用存储空间。

下表中的存储器大小是在将相应编译器设置为‘Release’模式并且优化选项为‘Size’的情况下测得的。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的使用情况。

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 [2]	4.3K + R_{FLS}	53 + R_{SRAM}	N/A	N/A	2.8K + R_{FLS}	86 + R_{SRAM}

功能说明

定义

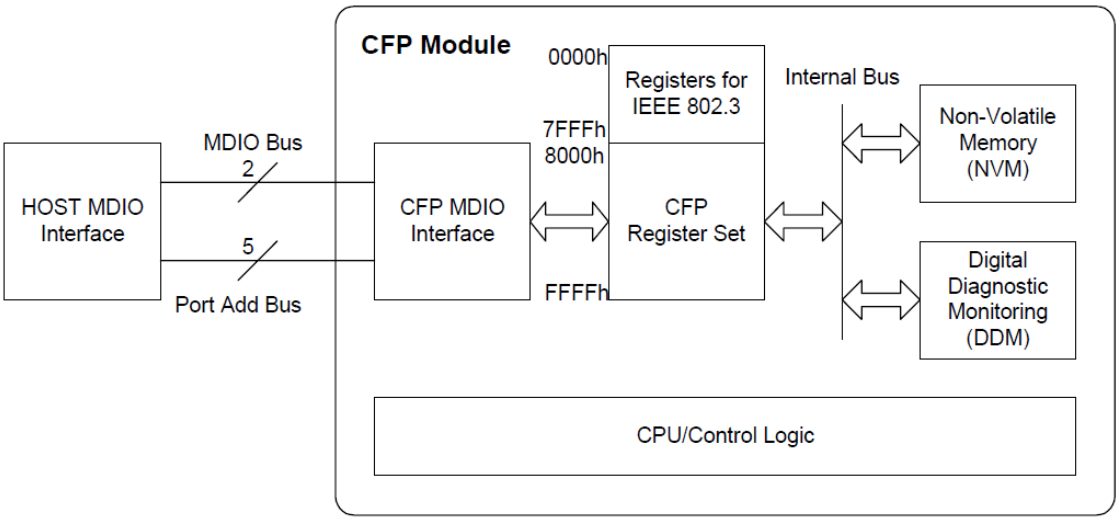
- **CFP Module**（CFP 模块）— 40 Gbps 和 100 Gbps 以太网中的外置 C 型可插拔光学收发器（CFP）模块。
- **MDIO Host**（MDIO 主机）— 用于控制和监控（各个）CFP 光学模块的启动、关闭和运行等操作的主系统处理器。
- **CFP Management Interface**（CFP 管理接口）— 主机和 CFP 模块间用于进行控制和诊断的管理接口。
- **Management Data Input/Output (MDIO) Bus**（管理数据输入/输出（MDIO）总线）— 带有 MDC 和 MDIO 硬件信号的工业标准（IEEE 802.3）串行总线。MDC 是由主机驱动的 MDIO 时钟线，MDIO 是主机和模块根据数据方向进行驱动的双线数据线。
- **CFP Register Set**（CFP 寄存器集）— 该寄存器集用于存储 CFP 光学模块 ID 信息、配置、诊断数据以及被配置为 16 位字的供应商特定数据，每个对象占用一个 MDIO 地址。

---

2. 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，其中：
   - 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

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 主机通过该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。

该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，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 组件会根据自定义程序设置自动分配 CFP 寄存器表。该组件支持以下寄存器访问类型：

- 只读寄存器
- 读取/写入寄存器
- 在 16 位字工作环境下的只读和读/写混合访问寄存器
- “读取时清除”寄存器
- 只写寄存器

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。

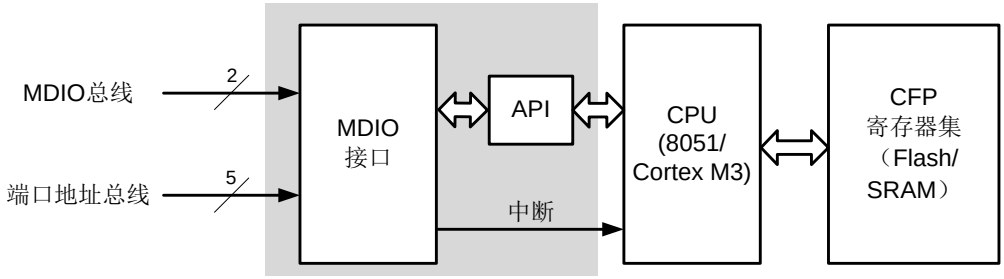
对 0x0000 到 0x7FFF 地址范围进行的所有访问都被忽略（MDIO 总线上无活动）。

框图和配置

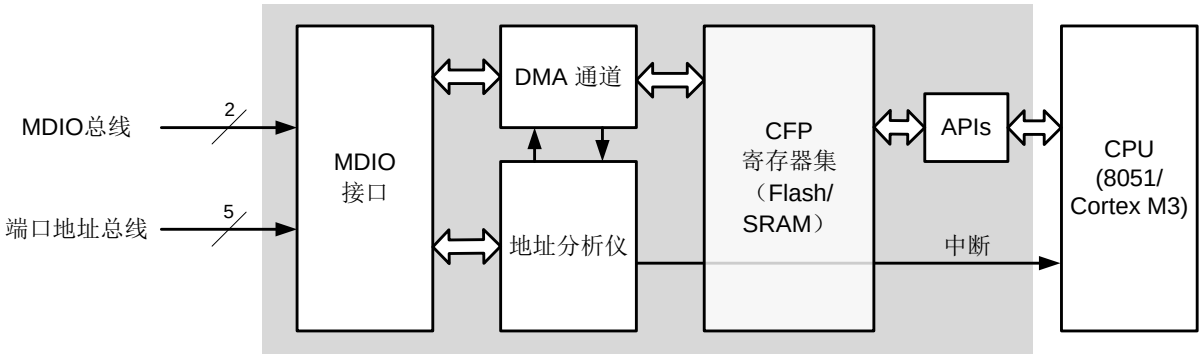
MDIO 接口组件作为一组已配置的 UDB 实现。在基本模式下，它会触发中断给 MDIO 总线上接收到的任何帧，这样用户能够实现 CFP 寄存器集以及数据处理算法固件。

在高级模式下，可以配置该组件，使之自动分配和处理硬件中的 CFP 寄存器集，而无需 CPU 干预。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）



时钟选择

当组件在**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**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。为了确保满足 CFP MSA 规范中指定的时序要求，组件必须运行于 40 MHz 的时钟频率。

寄存器

MDIO 接口组件有几个控制和状态寄存器。固件 API 使用这些寄存器来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。

使用资源

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。该组件利用以下资源。

配置	资源类型					
	数据路径单元	宏单元	状态寄存器	控制寄存器	DMA通道	中断
基本模式	2	16	2	2	–	–
高级模式	8	59	3	5	10	–

直流和交流电气特性

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

直流特性

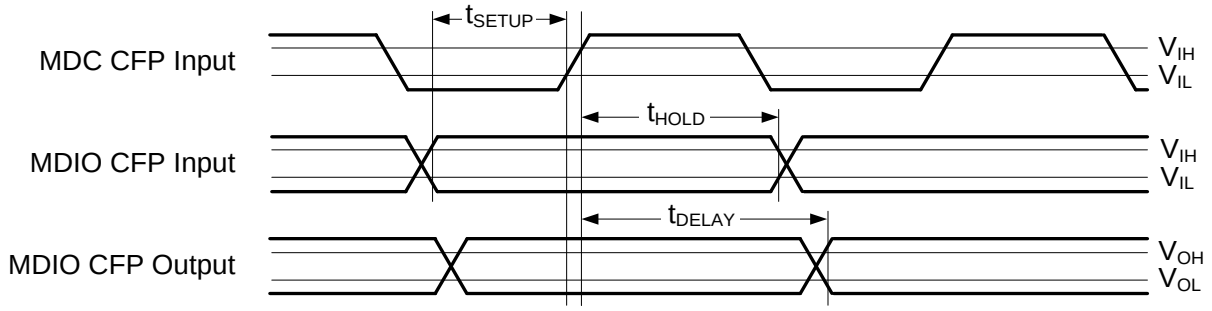
参数	说明	最小值	典型值	最大值	单位
V_{IH}	输入高电压	0.8	–	5.5	V
V_{IL}	输入低电压	-0.5	–	0.42	V
V_{OH} [3]	输出高电压（I_{OH} = -100 µA）	–	–	–	V
V_{OL}	输出低电压（I_{OL} = 100 µA）	0.001	–	0.002	V
I_{OH} [4]	输出高电流（V_{OH} = 1.0 V）	–	–	–	mA
I_{OL}	输出低电流（V_{OL} = 0.2 V）	10	–	–	mA
C_i	输入电容	–	–	7	pF

3. 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。
4. I_{OH}不适用于漏极开路驱动器。

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}	40	–	MHz
t_{SETUP}	主机MDIO建立时间	10	–	–	ns
t_{HOLD}	主机MDIO保持时间	10	–	–	ns
t_{DELAY}	CFP MDIO延迟时间	–	–	150	ns

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 路径延时之间的差异

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 宏单元的设置时间要求（3.51 ns）

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 路径延时之间的差异（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。

参考

- *CFP MSA 管理接口规范版本 2.0*
- *CFP MSA 硬件规范版本 1.4*

组件更改

本节列出了该组件各版本中的主要更改内容。

版本	更改内容	更改/影响的原因
1.20	纠正了当接收到的MDIO帧数据部分与特定的数据部分相匹配时将器件地址设置为PMA/PD（0x01）、PCS （0x03）或DTE XS （0x05）而导致组件在基本模式下不能正常工作的问题。 高级操作模式以及器件地址WIS （0x02）和PHY XS （0x04）均不受影响。	当MDIO帧的数据位与下面各器件地址的特定数据部分相匹配时，组件在基本模式下不能正常工作： PMA/PD X 1 1 0 0 X X PHYADDR 0 0 0 0 PCS X 1 1 0 0 X X PHYADDR 0 0 0 1 X X 1 1 0 0 X X PHYADDR 0 0 0 DTE XS X 1 1 0 0 X X PHYADDR 0 0 1 0 例如，如果器件地址 = PMA/PD和物理地址（PHYADDR） = 0x04，则下面各数据会发生故障：0x6040、0x6240、0x6440、0x6640、0xE040、0xE240、0xE440、0xE640。
	添加了用于设置5位和3位物理地址的选项。将地址宽度设置为3位时，组件会对来自MDIO帧的4个不同物理地址做出响应。	根据可用性反馈对组件进行改进。现有设计不受任何影响。
	当尝试对只写寄存器进行读操作时，转换（TA）位会返回逻辑低电平。前一个返回值为逻辑高电平。	符合CFP MSA规范的要求。MDIO主机将检查TA位，并希望它们在读取帧中均为逻辑低电平。
	更正了SetBits() API函数的检测DMA完成状态的问题。	函数可能在操作完成前被返回。
	更新并更正了数据手册。	反映版本1.20中的所有更改。
1.10	在自定义程序中添加了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硬规范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。请注意，必须配置连接至mdc/mdio终端的引脚，从而能够将**Sync mode**参数设置为“Transparent”。
	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规范的要求。
	删除了PSoC 5支持	PSoC Creator 3.0版本和更高版本不再支持PSoC 5。

版本	更改内容	更改/影响的原因
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，2014。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。也不会以明示或暗示的方式授予任何专利许可或其他权利。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证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，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。

PSoC®是赛普拉斯半导体公司的注册商标，PSoC Creator™和 Programmable System-on-Chip™是赛普拉斯半导体公司的商标。此处引用的所有其它商标或注册商标归其各自所有者所有。

所有源代码（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，未经赛普拉斯的明确书面许可，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（但不仅限于）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。对于合理预计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器件。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。