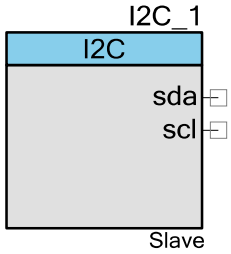
I²C 主控/多主控/从器件

3.1



特性

- 行业标准 NXP® I²C 总线接口
- 支持从器件、主控、多主控和多主控从器件操作
- 只需要两个引脚（SDA 和 SCL）与 I²C 总线连接
- 支持 100/400/1000 kbps 标准数据速率
- 高级 API 只需少量用户编程

概述

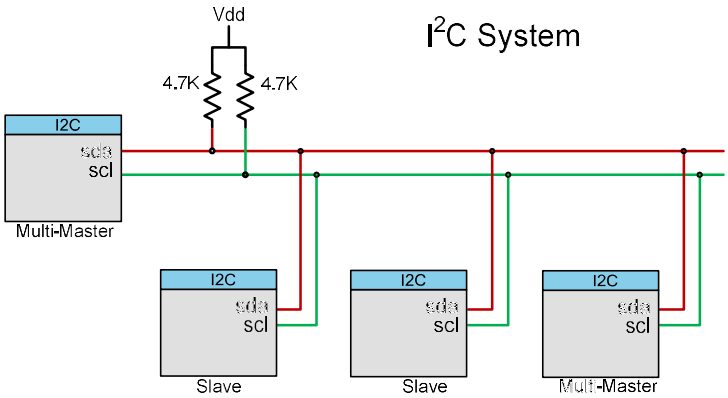
I²C 组件支持 I²C 从器件、主控和多主控配置。I²C 总线是由 Philips 开发的基于行业标准的两线硬件接口。主控在 I²C 总线上启动所有通信，并为所有从器件提供时钟。

I²C 组件支持的标准时钟速率高达 1000 kbps。I²C 组件与其他第三方从器件设备和主控设备相兼容。

注 此版本的组件数据手册涵盖了固定的硬件 I²C 模块和 UDB 版本。

在何种情况下使用 I²C 组件

当您对单一板或小系统中的多个设备进行联网时，使用 I²C 组件是最佳的解决方案。您可以将系统设计为单一主控和多从器件、多主控或主控和从器件的组合。



输入/输出连接

本节介绍 I²C 组件的各种输入和输出连接。I/O 列表中的星号 (*) 表示该 I/O 可能在 I/O 说明中列出的情况下隐藏在符号中。

sda — 输入/输出

串行数据 (SDA) 是 I²C 数据信号。这种双向数据信号用于传输或接收所有总线数据。应该将连接至 sda 的引脚配置为开漏驱动低电平。

SCL — 输入/输出

串行时钟 (SCL) 是主控生成的 I²C 时钟。虽然从器件从不会生成时钟信号，但它可使时钟保持在低电平状态使总线停顿，直至它准备发送数据或确认/否认[1]最新数据或地址为止。应该将连接至 scl 的引脚配置为开漏驱动低电平。

时钟 — 输入*

当 **Implementation**（实现）参数设置为 **UDB** 时，可以使用时钟输入。UDB 版本需要一个时钟才能提供 16 倍的过采样。

总线	时钟
50 kbps	800 kHz
100 kbps	1.6 MHz
400 kbps	6.4 MHz
1000 kbps	16 MHz

复位 — 输入*

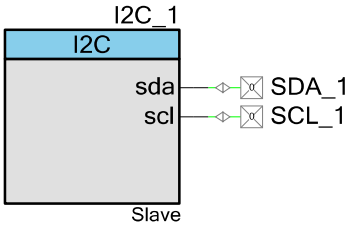
当 **Implementation**（实现）参数设置为 **UDB** 时可以使用复位输入。如果复位引脚保持在逻辑高电平状态，则 I²C 模块将处于复位状态，并且通过 I²C 进行的通信会停止。这仅适用于硬件复位。而软件必须使用 I2C_Stop() 和 I2C_Start() API 进行单独复位。复位输入在无外部连接的情况下应悬空。如果复位线路无任何连接，则组件将为其分配常数逻辑 0。

[1] NAK 为否定确认或未确认的缩写。I²C 文档中通常使用 NACK，而其他网络中使用 NAK。它们的含义相同。

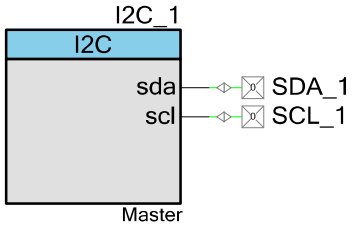
原理图宏信息

默认情况下，PSoC Creator 组件目录为 I²C 组件提供了四个原理图宏实现。这些宏包含已连接和已配置的引脚，并根据需要提供了时钟源。原理图宏使用配置有固定功能和 UDB 硬件的 I²C 从器件和主控组件，如下图所示。

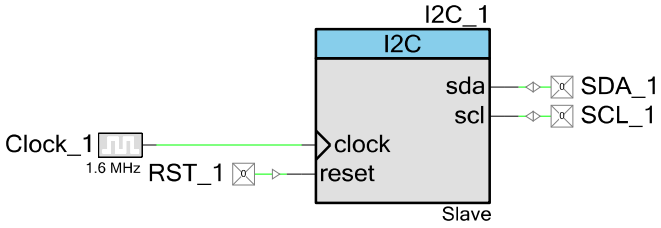
带有引脚的固定功能 I²C 从器件



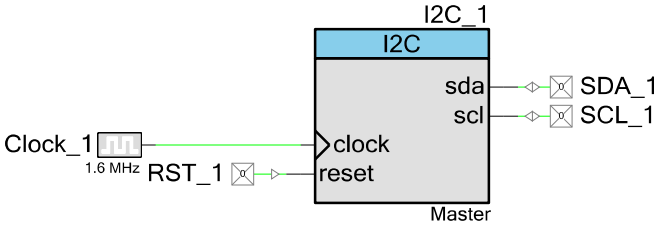
固定功能 I²C 主控引脚



带有时钟和引脚的 UDB I²C 从器件

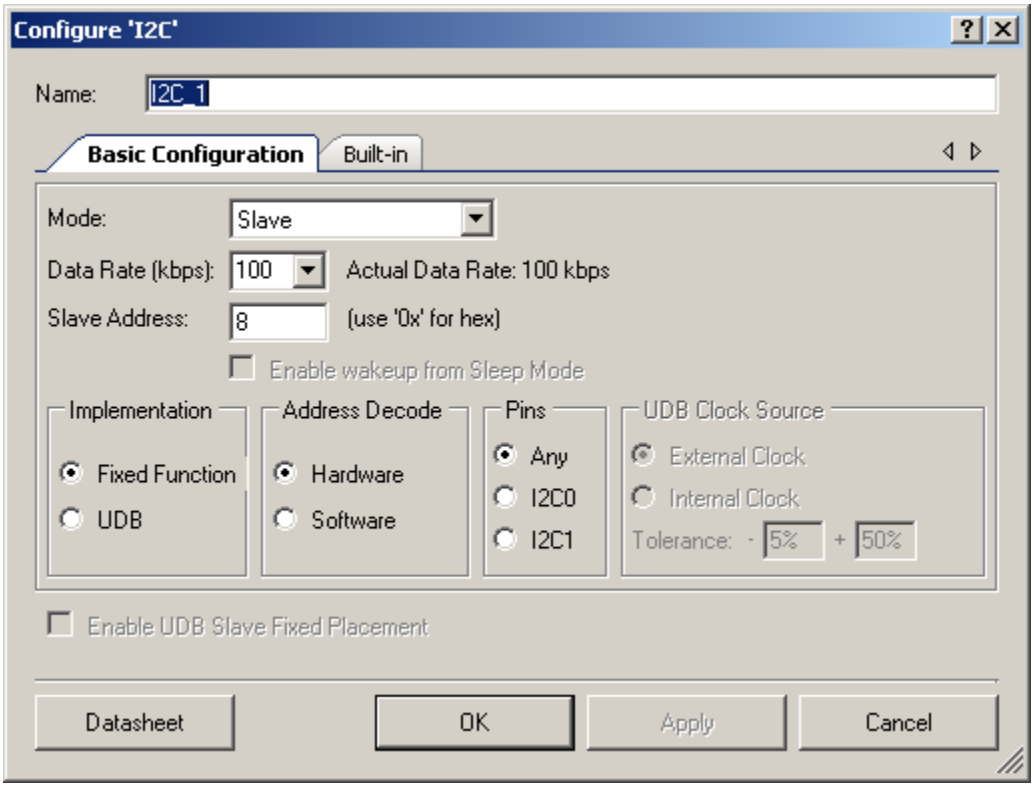


带有时钟和引脚的 UDB I²C 主控



元件参数

将 I²C 组件拖放到您的设计上，双击它以打开 **Configure**（配置）对话框。



I²C 组件提供了以下参数。

Mode（模式）

此选项可确定所支持的模式：从器件、主控、多主控或多主控从器件。

Mode（模式）	说明
从器件	仅适用于从器件的操作（默认）。
Master（主控）	仅适用于主控的操作。
Multi-Master（多主控）	支持总线上使用多主控。
Multi-Master-Slave（多主控从器件）	从器件和多主控的同步操作。

Data Rate（数据速率）

此参数用于设置 I²C 数据速率值（可达 1000 kbps）；实际速度可能会基于可用的时钟速度和分频器范围而有所不同。标准数据速率[2]为 50、100（默认）、400 以及 1000 kbps。如果将 **Implementation**（实现）设为 **UDB** 且 UDB Clock Source（UDB 时钟源）参数设为 **External Clock**（外部时钟），则可忽略 **Data Rate**（数据速率）参数；数据速率将由 16x 输入时钟确定。

从器件地址

这是从器件将识别的 I²C 地址。如果不选择从器件操作，则会忽略此参数。您可以选择介于 0 至 127（0x00 和 0x7F）之间的从器件地址，默认值为 **8**。此地址为右对齐的 7 位从器件地址，它不包括读/写位。您可以输入十进制或十六进制的值，对于十六进制数值，请在地址之前键入“0x”。如果需要 10 位从器件地址，则设计者必须使用软件地址解码并为 ISR 中的 10 位地址的第二个字节提供解码支持。

实现

此选项可确定如何在设备上实现 I²C 硬件。

实现	说明
固定功能	在器件上使用固定功能模块（默认）。
UDB	在 UDB 阵列中实现 I²C。

地址解码

通过此参数，您可以选择软件地址解码或硬件地址解码。对于提供了足够的 API 并且只需一个从器件地址的大部分应用程序来说，将首选硬件地址解码。而对于希望修改源代码以检测多个从器件地址的应用程序而言，则必须使用软件地址检测。**硬件**是默认设置。如果使能了硬件地址解码，该模块会自动否认不属于自己的地址，而无需 CPU 干预。它会在收到正确地址后自动中断 CPU 的运行，并将 SCL 线保持在低电平，直至 CPU 干预为止。

引脚

此参数可确定用于 SDA 和 SCL 信号连接的引脚类型。此参数包含三个可选值：**Any**、**I2C0** 和 **I2C1**。默认值为 **Any**。

Any 表示通用 I/O（GPIO 或 SIO）。如果不需要**从睡眠模式使能唤醒**，则应将 **Any** 用于 SDA 和 SCL。如果需要**从睡眠模式使能唤醒**，则必须使用 **I2C0** 或 **I2C1**；使用 **I2C0** 或 **I2C1**，您可以将该器件配置为在 I²C 地址匹配时唤醒。

[2] 固定功能实现仅支持 PSoC 3 ES2 和 PSoC 5 器件使用标准数据速率 50、100 或 400 kbps。而对最高可达 1000 kbps 的不同数据速率使用基于 UDB 的实现。

I²C 组件不负责检查引脚分配是否正确。

值	引脚
Any	通过原理图连接的任意 GPIO 或 SIO 引脚
I2C0	SCL = SIO 引脚 P12[4]，SDA = SIO 引脚 P12[5]
I2C1	SCL=SIO 引脚 P12[0]，SDA=SIO 引脚 P12[1]

从睡眠模式使能唤醒

此选项会在地址匹配时从睡眠模式唤醒系统。此选项仅在 **Address Decode**（地址解码）设为 **Hardware**（硬件），并且 SDA 和 SCL 信号连接至 SIO 引脚（**I2C0** 或 **I2C1**）时才有效。默认情况下，禁用此选项。PSoC 3 ES2 和 PSoC 5 器件不支持此选项。

在切换到睡眠模式后，必须使能在从器件地址匹配时可以使用 I²C 来唤醒设备。这一操作可通过调用 I2C_Sleep() API 来完成；另请参考 *System Reference Guide*（《系统参考指南》）中的在硬件地址匹配时唤醒（在硬件地址匹配时唤醒）和“Power Management APIs”（电源管理 API）章节。

UDB 时钟源

该参数允许您针对数据速率生成在内部配置的时钟和外部配置的时钟之间进行选择。考虑到 16 倍过采样，当设置为 **Internal Clock**（内部时钟）时，PSoC Creator 将基于 **Data Rate**（数据速率）参数计算和配置所需时钟频率。在 **External Clock**（外部时钟）模式下，组件不控制数据速率而是基于用户连接的时钟源显示实际速率。如果将此参数设置为 **Internal Clock**（内部时钟），则符号上将不显示时钟输入。

您可以为内部时钟输入所需容差值。时钟容差可指定为百分比。从器件模式的默认范围为 **-5% 至 +50%**。时钟在此模式下可快速运行。对于其他模式，默认范围为 -25% 至 +5%。此外，在主控模式下运行较慢。在最大数据速率 (1000 kbps) 时，时钟应等于或慢于预期速率，而不应超过预期速率。否则，可能会导致意外行为。

使能 UDB 从器件固定放置

您可以通过此参数选择一个固定组件放置，以提供高于无限制放置下的组件性能。当设置此参数后，全部组件资源都将固定在器件的右上角。此参数可控制连接至组件的引脚分配。引脚分配的选择并非组件性能的决定因素。此选项仅在 **Mode**（模式）设置为 **Slave**（从器件）且 **Implementation**（实现）设置为 **UDB 时才有效。** 默认情况下，禁用此选项。

组件的固定放置方面可消除在处理“所有路由的最大值”案例（请参见“直流电和交流电电气特性（UDB 实现）（DC 和 AC 电气特征（UDB 实现）了解详细信息）时要考虑的多变性。此外，还可以在一个完全空白的设计中按照与非固定放置设计相同的方式继续运行固定放置。

时钟选择

当选择内部时钟配置时，PSoC Creator 计算所需的频率和时钟源，并生成实现要用的资源。否则，您必须提供时钟组件并计算所需的时钟频率。该频率是可提供的所需数据速率的 16 倍。例如，需要使用 1.6 MHz 的时钟才能使数据速率达到 100 kbps。

固定功能模块使用 BUS_CLK，并通过定制器分频器来计算该模块以对 16/32 过采样率（50 kbps 过采样率为 32，任何其他过采样率为 16）进行存档。

注 请查看勘误表中的第 49 项。I²C 时钟可以为早期芯片版本上的 I²C 固定功能模块提供所需的时钟。

资源

以下配置设置用于生成资源使用信息：(1) **Address Decode**（地址解码）设置为 **Software**（软件）；(2) 取消选中 **Enable wakeup from Sleep Mode**（从睡眠模式使能唤醒）；**UDB Clock Source**（UDB 时钟源）设置为 **External Clock**（外部时钟）。

固定 I²C 模块用于固定功能的实现。

模式	资源类型	API 存储器（字节）		引脚（每个外部 I/O）
	I²C 固定模块	闪存	RAM	
从器件	1	916	22	2
主控	1	1737	20	2
多主控	1	1889	20	2
多主控从器件	1	2550	34	2

对于 UDB 实现，请参见下表。

模式	资源类型				API 存储器（字节）		引脚（每个外部 I/O）
	数据路径	PLD	状态单元	Control/ Count7 单元	闪存	RAM	
从器件	1	12	1	2	962	18	4
主控	2	14	1	1	1834	17	4
多主控	2	18	1	1	2007	17	4
多主控从器件	2	32	1	2	2754	30	4

应用程序编程接口

您可以使用应用程序编程接口 (API) 子程序在运行时配置组件。下表列出了每个函数的接口，并进行了说明。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 会将实例名称“I2C_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。出于可读性考虑，下表中使用的实例名称为“I2C”。

所有 API 函数都假设基于 I²C 主控进行数据定向。当数据从主控写入到从器件时会发生写入事件。当主控由从器件读取数据时会发生读取事件。

通用函数

此节包含了 I²C 从器件或主控操作通用的函数。

函数	说明
I2C_Start()	初始化并使能 I²C 组件。使能 I²C 中断后，该组件可响应 I²C 通信。
I2C_Stop()	停止响应 I²C 通信（禁用 I²C 中断）。
I2C_EnableInt()	使能中断，大部分 I²C 操作都需要使能中断。
I2C_DisableInt()	禁用中断。I2C_Stop() API 会自动执行此操作。
I2C_Sleep()	停止 I²C 操作，并保存 I²C 非保留配置寄存器（禁用中断）。如果使能了从睡眠模式唤醒功能（禁用 I²C 中断），则可以准备执行在地址匹配时唤醒操作。
I2C_Wakeup()	恢复 I²C 非保留配置寄存器，并使能 I²C 操作（使能 I²C 中断）。
I2C_Init()	使用定制器提供的初始值来初始化 I²C 寄存器。
I2C_Enable()	激活 I²C 硬件并开始执行组件操作。

函数	说明
I2C_SaveConfig()	保存 I²C 非保留配置寄存器（禁用 I²C 中断）。
I2C_RestoreConfig()	恢复由 I2C_SaveConfig() 或 I2C_Sleep() 保存的 I²C 非保留配置寄存器（使能 I²C 中断）。

全局变量

在执行一般操作时，无需了解这些变量。

变量	说明
I2C_initVar	I2C_initVar 表示 I²C 组件是否已完成初始化。变量将初始化为 0，并在第一次调用 I2C_Start() 时设置为 1。这样，在第一次调用 I2C_Start() 子程序后，组件无需重新初始化便可重启。 如需重新初始化组件，则可在调用 I2C_Start() 或 I2C_Enable() 函数前调用 I2C_Init() 函数。
I2C_state	表示 I²C 状态机的当前状态。
I2C_mstrStatus	表示 I²C 主控的当前状态。
I2C_mstrControl	通过生成或不生成“停止”来控制操作的主控端。
I2C_mstrRdBufPtr	主控读取缓冲区的指针。
I2C_mstrRdBufSize	主控读取缓冲区的大小。
I2C_mstrRdBufIndex	主控读取缓冲区中的当前索引。
I2C_mstrWrBufPtr	主控写入缓冲区的指针。
I2C_mstrWrBufSize	主控写入缓冲区的大小。
I2C_mstrWrBufIndex	主控写入缓冲区中的当前索引。
I2C_slStatus	I²C 从器件的当前状态。
I2C_slAddress	I²C 从器件的软件地址。
I2C_slRdBufPtr	从器件读取缓冲区的指针。
I2C_slRdBufSize	从器件读取缓冲区的大小。
I2C_slRdBufIndex	从器件读取缓冲区中的当前索引。
I2C_slWrBufPtr	从器件写入缓冲区的指针。
I2C_slWrBufSize	从器件写入缓冲区的大小。
I2C_slWrBufIndex	从器件写入缓冲区中的当前索引。

通用函数

void I2C_Start(void)

说明： 这是开始执行组件操作的首选方法。I2C_Start() 会首先调用 I2C_Init() 函数，然后调用 I2C_Enable() 函数。您必须在执行 I²C 总线操作之前调用 I2C_Start()。

此 API 可使能 I²C 中断。大部分 I²C 操作都需要中断。

在调用此函数之前，必须设置 I²C 从器件缓冲区，以避免在设置缓冲区时读取或写入部分数据。

在处于使能状态且未设置缓冲区的情况下，I²C 从器件的行为将如下所示：

I²C 读取传输 — 返回 0xFF，直到设置读取缓冲区为止。使用 I2C_SlaveInitReadBuf() 函数设置读取缓冲区；

I²C 写入传输 — 因没有空间来存储接收到的数据，故将发送 NAK。使用 I2C_SlaveInitWriteBuf() 函数设置读取缓冲区；

参数： 无

返回值： 无

副作用： 无

void I2C_Stop(void)

说明： 此函数可禁用 I²C 硬件和中断。

FF 实现 （**仅限于 Production PSoC 3**）：如果 I²C 总线被器件锁定且被设为空闲状态，则系统会将其释放。

UDB 实现：如果 I²C 总线被器件锁定且被设为空闲状态，则系统会将其释放。

参数： 无

返回值： 无

副作用： 无

void I2C_EnableInt(void)

说明： 此函数可使能 I²C 中断。大部分操作都需要中断。

参数： 无

返回值： 无

副作用： 无

void I2C_DisableInt(void)

说明： 此函数可使能 I²C 中断。通常情况下，在 I2C_Stop() 函数禁用中断后就不需要此函数了。

参数： 无

返回值： 无

副作用： 如果在运行 I²C 的同时禁用 I²C 中断，则可能造成 I²C 总线锁定。

void I2C_Sleep(void)

说明： 这是准备组件进入睡眠的首选 API。I²C 中断会在函数调用后被禁用。

使能在地址匹配时唤醒： 如果预设在此 API 调用过程中对器件执行某项数据操作，则该操作将等到当前操作完成之后才可执行。在此器件进入睡眠状态之前，将会否认对此器件预设的所有后续 I²C通信。地址匹配事件将唤醒该芯片。

禁用在地址匹配时唤醒： 如果当前已使能 API，则 API 将检查当前的 I²C 组件状态，并保存该组件，以及调用 I2C_Stop() 来禁用该组件。然后会调用 I2C_SaveConfig() 来保存 I²C 非保留配置寄存器。

在调用 CyPmSleep() 或 CyPmHibernate() 函数之前应先调用 I2C_Sleep() 函数。有关电源管理函数的更多信息，请参考 PSoC Creator *System Reference Guide*（《系统参考指南》）。

参数： 无

返回值： 无

副作用： 无

void I2C_Wakeup(void)

说明： 这是将组件恢复到上次调用 I2C_Sleep() 时状态的首选 API。I²C 中断会在函数调用后使能。

使能在地址匹配时唤醒： 此 API 会使能 I²C 主控功能（如果在睡眠之前它处于使能状态），并禁用 I²C 备份调节器。在使能 I²C 中断后将继续进行输入的数据操作。

禁用在地址匹配时唤醒： 此 API 会调用 I2C_RestoreConfig() 来恢复 I²C 非保留配置寄存器。如果在调用 I2C_Sleep() 函数之前已使能该组件，I2C_Wakeup() 将重新使能该组件。

参数： 无

返回值： 无

副作用： 在调用 I2C_Wakeup() 函数前未调用 I2C_Sleep() 或 I2C_SaveConfig() 函数，可能会产生意外行为。

void I2C_Init(void)

说明： 此函数根据定制器 Configure（配置）对话框设置来初始化或恢复组件。您无需调用 I2C_Init()，因为 I2C_Start() API 会调用此函数，该函数是开始执行组件操作的首选方法。

参数： 无

返回值： 无

副作用： 所有寄存器将设置为定制器“配置”对话框中的值。

void I2C_Enable(void)

说明： 此函数激活硬件并开始执行组件操作。您无需调用 I2C_Enable()，因为 I2C_Start() API 会调用此函数，该函数是开始执行组件操作的首选方法。如果要调用此 API，则必须首先调用 I2C_Start() 或 I2C_Init()。

参数： 无

返回值： 无

副作用： 无

void I2C_SaveConfig(void)

说明： 此函数会保存 I²C 组件非保留配置寄存器，并禁用 I²C 中断

使能在地址匹配时唤醒： 此 API 将禁用 I²C 主控，如果此前已使能 I2C，则可使能 I²C 备份调节器。如果预设在此 API 调用过程中对器件执行某项操作，则该操作将等到当前的数据操作完成并且 I²C 做好进入睡眠的准备之后才可执行。在此器件进入睡眠状态之前，将会否认所有后续的 I²C 通信。

禁用在地址匹配时唤醒： 请参考主要说明。

禁用 I²C 中断时并不取决于是否使能地址匹配时唤醒。

参数： 无

返回值： 无

副作用： 无

void I2C_RestoreConfig(void)

说明： 此函数会将 I²C 组件非保留配置寄存器恢复到调用 I2C_Sleep() 或 I2C_SaveConifg() 之前的状态。使能 I²C 中断。

使能在地址匹配时唤醒： 此 API 会使能 I²C 主控功能（如果在睡眠之前它处于使能状态），并禁用 I²C 备份调节器。

禁用在地址匹配时唤醒： 请参考主要说明。

使能 I²C 中断时并不取决于是否使能地址匹配唤醒。

参数： 无

返回值： 无

副作用： 在调用此函数前未调用 I2C_Sleep() 或 I2C_SaveConfig() 函数，可能会产生意外行为。

从器件函数

此节列出了用于 I²C 从器件操作的函数。在从器件已使能的情况下可以使用这些函数。

函数	说明
I2C_SlaveStatus()	返回从器件状态标志。
I2C_SlaveClearReadStatus()	返回读取状态标志并清除从器件读取状态标志。
I2C_SlaveClearWriteStatus()	返回写入状态并清除从器件写入状态标志。
I2C_SlaveSetAddress()	设置从器件地址，该值介于 0 至 127 (0x00 - 0x7F) 之间。
I2C_SlaveInitReadBuf()	设置从器件接收数据缓冲区。（主控 <- 从器件）
I2C_SlaveInitWriteBuf()	设置从器件写入缓冲区。（主控 -> 从器件）
I2C_SlaveGetReadBufSize()	返回缓冲区复位后主控读取的字节数。
I2C_SlaveGetWriteBufSize()	返回缓冲区复位后主控写入的字节数。
I2C_SlaveClearReadBuf()	将读取缓冲区计数器复位至零。
I2C_SlaveClearWriteBuf()	将写入缓冲区计数器复位至零。

uint8 I2C_SlaveStatus(void)

说明： 此函数返回从器件通信状态。

参数： 无

返回值： uint8：I²C 从器件的当前状态。

从器件状态常量	说明
I2C_SSTAT_RD_CMPLT[3]	从器件读取传输已完成。在主控发送 NAK 响应通知已完成读取时设置。
I2C_SSTAT_RD_BUSY	正在执行从器件读取传输。在主控读取以寻址从器件时设置，并在设置 RD_CMPLT 时清除。
I2C_SSTAT_RD_ERR_OVFL	主控尝试读取超过缓冲区字节数的字节。
I2C_SSTAT_WR_CMPLT[4]	从器件写入传输已完成。在收到“停止”条件时设置。
I2C_SSTAT_WR_BUSY	正在执行从器件写入传输。在主控写入以寻址从器件时设置，并在设置 WR_CMPLT 时清除。
I2C_SSTAT_WR_ERR_OVFL	主控试图在缓冲区结束时写入内容。接收的字节被从器件否认。

副作用： 无

uint8 I2C_SlaveClearReadStatus(void)

说明： 此函数可清除读取状态标志并返回其值。其他状态标志不会受影响。

参数： 无

返回值： uint8：从器件的当前读取状态。有关常量，请参考 I2C_SlaveStatus() 函数。

副作用： 无

---

[3] — 定义由 I2C_SSTAT_RD_CMPT 更改为 I2C_SSTAT_RD_CMPLT，以符合主控读取完成定义。该组件同时支持两种定义，但是 I2C_SSTAT_RD_CMPT 将取消使用。

[4] — 定义由 I2C_SSTAT_WR_CMPT 更改为 I2C_SSTAT_WR_CMPLT，以符合主控写入完成定义。该组件同时支持两种定义，但是 I2C_SSTAT_WR_CMPT 将取消使用。

uint8 I2C_SlaveClearWriteStatus(void)

说明： 此函数清除写入状态标志并返回其值。其他状态标志不会受影响。

参数： 无

返回值： uint8：从器件的当前写入状态。有关常量，请参见 I2C_SlaveStatus() 函数。

副作用： 无

void I2C_SlaveSetAddress(uint8 address)

说明： 此函数可设置 I²C 从器件地址

参数： uint8 address：主设备的 I²C 从器件地址。此值可以为 0 至 127 (0x00 - 0x7F) 之间的任意地址。此地址为右对齐的 7 位从器件地址，它不包括读/写位。

返回值： 无

副作用： 无

void I2C_SlaveInitReadBuf(uint8 * rdBuf, uint8 bufSize)

说明： 此函数可设置缓冲区指针并设置读取缓冲区的大小。此函数还会复位 I2C_SlaveGetReadBufSize() 函数所返回的传输计数。

参数： uint8* rdBuf：主控读取的数据缓冲区的指针。

uint8 bufSize：公开给 I²C 主控的缓冲区的大小。

返回值： 无

副作用： 如果在总线数据操作期间调用此函数，则可能传输先前缓冲区位置的数据以及当前缓冲区开始位置的数据。

void I2C_SlaveInitWriteBuf(uint8 * wrBuf, uint8 bufSize)

说明： 此函数可设置缓冲区指针并设置写入缓冲区的大小。此函数还会复位 I2C_SlaveGetWriteBufSize() 函数所返回的传输计数。

参数： uint8* wrBuf：主控写入的数据缓冲区的指针。

uint8 bufSize：公开给 I²C 主控的缓冲区的大小。

返回值： 无

副作用： 如果在总线数据操作期间调用此函数，则可能会在先前的缓冲区和当前的缓冲区位置接收数据。

uint8 I2C_SlaveGetReadBufSize(void)

说明： 返回执行 I2C_SlaveInitReadBuf() 或 I2C_SlaveClearReadBuf() 函数后 I²C 主控读取的字节数。

最大返回值为读取缓冲区的大小。

参数： 无

返回值： uint8：主控读取的字节。

副作用： 无

uint8 I2C_SlaveGetWriteBufSize(void)

说明： 此函数返回执行 I2C_SlaveInitWriteBuf() 或 I2C_SlaveClearWriteBuf() 函数后 I²C 主控写入的字节数。

最大返回值为写入缓冲区的大小。

参数： 无

返回值： uint8：主控写入的字节。

副作用： 无

void I2C_SlaveClearReadBuf(void)

说明： 此函数将读取指针复位至读取缓冲区中的第一个字节。主控读取的下一个字节将是读取缓冲区中的第一个字节。

参数： 无

返回值： 无

副作用： 无

void I2C_SlaveClearWriteBuf(void)

说明： 此函数将写入指针复位至写入缓冲区中的第一个字节。主控写入的下一个字节将是写入缓冲区中的第一个字节。

参数： 无

返回值： 无

副作用： 无

主控和多主控函数

这些函数仅在使能了主控或多主控模式时才可用。

函数	说明
I2C_MasterStatus()	返回主控状态。
I2C_MasterClearStatus()	返回主控状态并清除状态标志。
I2C_MasterWriteBuf()	将引用的数据缓冲区写入到指定的从器件地址。
I2C_MasterReadBuf()	从指定的从器件地址读取数据，并将数据放入引用的缓冲区中。
I2C_MasterSendStart()	只向特定的地址发送“启动”。
I2C_MasterSendRestart()	只向指定的地址发送“重启”。
I2C_MasterSendStop()	生成“停止”条件。
I2C_MasterWriteByte()	写入单个字节。这是一个手动命令，它只应与 I2C_MasterSendStart() 或 I2C_MasterSendRestart() 函数一同使用。
I2C_MasterReadByte()	读取单个字节。这是一个手动命令，它只应与 I2C_MasterSendStart() 或 I2C_MasterSendRestart() 函数一同使用。
I2C_MasterGetReadBufSize()	返回调用 I2C_MasterClearReadBuf() 函数后读取的数据字节计数。
I2C_MasterGetWriteBufSize()	返回调用 I2C_MasterClearWriteBuf() 函数后写入的数据字节计数。
I2C_MasterClearReadBuf()	将读取缓冲区指针复位至缓冲区的开始位置。
I2C_MasterClearWriteBuf()	将写入缓冲区指针复位至缓冲区的开始位置。

uint8 I2C_MasterStatus(void)

说明： 此函数可返回主控通信状态。

参数： 无

返回值： uint8：I²C 主控的当前状态。可以将所有的 I²C 主控状态常量一起运行“或”运算。

主控状态常量	说明
I2C_MSTAT_RD_CMPLT	读取传输已完成。 必须检查错误状况位，以确保读取传输顺利完成。
I2C_MSTAT_WR_CMPLT	写入传输已完成。 必须检查错误状况位，以确保写入传输顺利完成。
I2C_MSTAT_XFER_INP	正在进行传输
I2C_MSTAT_XFER_HALT	传输已终止。I²C 总线目前处于等待状态，等待主控生成“重启”或“停止”条件。
I2C_MSTAT_ERR_SHORT_XFER	错误状况：在传输完所有字节之前写入传输已结束。
I2C_MSTAT_ERR_ADDR_NAK	错误状况：从器件无法确认该地址。
I2C_MSTAT_ERR_ARB_LOST	错误状况：主控在与从器件进行通信时仲裁失败。
I2C_MSTAT_ERR_XFER	错误状况：这是表中所示错误状况的“或”运算值。 如果错误状况位都已清除却设置了此位，则传输会因从器件操作而中断。

副作用： 无

uint8 I2C_MasterClearStatus(void)

说明： 此函数可清除所有状态标志并返回主控状态。

参数： 无

返回值： uint8：主控的当前状态。有关常量，请参见 I2C_MasterSendStart() 函数。

副作用： 无

uint8 I2C_MasterWriteBuf(uint8 slaveAddress, uint8 * wrData, uint8 cnt, uint8 mode)

说明： 此函数自动将整个缓冲数据写入从器件设备中。此函数启动数据传输之后，附带的 ISR 在逐字节模式下管理后续的数据传输。使能 I²C 中断。

参数： uint8 slaveAddress：右对齐的 7 位从器件地址（有效范围介于 0 至 127 之间）。

uint8 wrData：要发送的数据缓冲区指针。

uint8 cnt：要发送的缓冲区字节数。

uint8 mode：传输模式会定义：(1) 在传输开始时是生成“启动”条件还是生成“重启”条件，以及 (2) 在总线上生成“停止”条件之前是完成该传输还是停止该传输。

在传输模式中，可以对所有模式常量执行“或”运算。

模式常量	说明
I2C_MODE_COMPLETE_XFER	从头至尾执行完整的传输。
I2C_MODE_REPEAT_START	发送“重复启动”而不是“启动”。
I2C_MODE_NO_STOP	执行传输，而不停止。

返回值： uint8：错误状态。请参见 I2C_MasterSendStart() 函数获取常量信息。

副作用： 无

uint8 I2C_MasterReadBuf(uint8 slaveAddress, uint8 * rdData, uint8 cnt, uint8 mode)

说明： 此函数自动读取从器件设备中的整个缓冲数据。此函数启动数据传输后，附带的 ISR 将在逐字节模式下管理接下来的数据传输。使能 I²C 中断。

参数： uint8 slaveAddress：右对齐的 7 位从器件地址（有效范围介于 0 至 127 之间）。

uint8 rdData：从器件向其发送数据的缓冲区的指针。

uint8 cnt：要读取的缓冲区字节数。

uint8 mode：传输模式会定义：(1) 在传输开始时是生成“启动”条件还是生成“重启”条件，以及 (2) 在总线上生成“停止”条件之前是完成该传输还是停止该传输。

在传输模式中，可以对所有模式常量执行“或”运算

模式常量	说明
I2C_MODE_COMPLETE_XFER	从头至尾执行完整的传输。
I2C_MODE_REPEAT_START	发送重复启动而不是启动。
I2C_MODE_NO_STOP	执行传输，而不停止。

返回值： uint8：错误状态。有关常量，请参见 I2C_MasterSendStart() 函数。

副作用： 无

uint8 I2C_MasterSendStart(uint8 slaveAddress, uint8 R_nW)

说明： 此函数可生成启动条件并发送带有读/写位的从器件地址。禁用 I²C 中断。

参数： uint8 slaveAddress：右对齐的 7 位从器件地址（有效范围介于 0 至 127 之间）。

uint8 R_nW：设置为 0：发送写入命令；设置为非 0 值：发送读取命令。

返回值： uint8：错误状态。

模式常量	说明
I2C_MSTR_NO_ERROR	正确无误地完成函数操作。
I2C_MSTR_BUS_BUSY	总线繁忙，未生成启动条件。
I2C_MSTR_NOT_READY	总线上的主控无效，或者正在进行从器件操作。
I2C_MSTR_ERR_LB_NAK	已否认最后一个字节。
I2C_MSTR_ERR_ARB_LOST	在生成启动时主控仲裁失败。（此状态仅在使能多主控时才有效。）
I2C_MSTR_ABORT_XFER	由于启动从器件操作而终止生成启动条件。（此状态仅在多主控从器件模式下有效。）

副作用： 此函数正在执行阻止操作，并且在 I2C_CSR 寄存器中设置 byte_complete 位之前不会退出。

uint8 I2C_MasterSendRestart(uint8 slaveAddress, uint8 R_nW)

说明： 此函数生成重启条件并发送带有读/写位的从器件地址。

参数： uint8 slaveAddress：右对齐的 7 位从器件地址（有效范围介于 0 至 127 之间）。

uint8 R_nW：设置为 0：发送写入命令；设置为非 0 值：发送读取命令。

返回值： uint8：错误状态。有关常量，请参见 I2C_MasterSendStart() 函数。

副作用： 此函数正在执行阻止操作，并且在 I2C_CSR 寄存器中设置 byte_complete 位之前不会退出。

uint8 I2C_MasterSendStop(void)

说明： 此函数在总线上生成 I²C “停止”条件。如果在调用此函数之前“启动”或“重启”条件失败，则此函数将不执行任何操作。

参数： 无

返回值： uint8：错误状态。有关常量，请参见 I2C_MasterSendStart() 命令。

副作用： 此函数正在执行阻止操作，在符合下列条件之前不会退出：

主控： 此函数不会等待生成“停止”条件。

多主控和多主控从器件： 在生成停止条件或在 ACK/NAK 位上仲裁失败时，此函数会进行等待。

uint8 I2C_MasterWriteByte(uint8 theByte)

说明： 由此函数向从器件发送一个字节。在调用此函数之前必须生成有效的“启动”或“重启”条件。如果在调用此函数之前“启动”或“重启”条件失败，则此函数将不执行任何操作。

参数： uint8 theByte：发送至从器件的数据字节。

返回值： uint8：错误状态。

模式常量	说明
I2C_MSTR_NO_ERROR	正确无误地完成函数操作。
I2C_MSTR_NOT_READY	总线上的主控无效，或者正在执行从器件操作。
I2C_MSTR_ERR_LB_NAK	已否认最后一个字节。
I2C_MSTR_ERR_ARB_LOST	在生成“启动”时主控仲裁失败。（此状态仅在使能多主控时才有效。）

副作用： 此函数正在执行阻止操作，并且在 I2C_CSR 寄存器中设置字 byte_complete 之前不会退出。

uint8 I2C_MasterReadByte(uint8 acknNak)

说明： 此函数可读取从器件中的一个字节，并确认或否认该传输。在调用此函数之前必须生成有效的“启动”或“重启”条件。如果在调用此函数之前“启动”或“重启”条件失败，则此函数将不执行任何操作，并返回一个为零的值。

参数： uint8 acknNak：若为 0，则发送 NAK；若为非 0，则发送 ACK。

返回值： uint8：自从器件中读取字节

副作用： 此函数正在执行阻止操作，并且在 I2C_CSR 寄存器中设置 byte_complete 位之前不会退出。

uint8 I2C_MasterGetReadBufSize(void)

说明： 此函数返回使用 I2C_MasterReadBuf() 函数传输的字节数。

参数： 无

返回值： uint8：传输的字节计数。如果传输尚未完成，此函数将返回截至到目前已传输的字节计数。

副作用： 无

uint8 I2C_MasterGetWriteBufSize(void)

说明： 此函数将返回使用 I2C_MasterWriteBuf() 函数传输的字节数。

参数： 无

返回值： uint8：传输的字节计数。如果传输尚未完成，它将返回截至到目前已传输的字节计数。

副作用： 无

void I2C_MasterClearReadBufSize(void)

说明： 此函数将读取缓冲区指针复位到缓冲区中的第一个字节。

参数： 无

返回值： 无

副作用： 无

void I2C_MasterClearWriteBufSize(void)

说明： 此函数将写入缓冲区指针复位到缓冲区中的第一个字节。

参数： 无

返回值： 无

副作用： 无

多主控从器件函数

多主控从器件是将从器件函数和多主控函数合并在一起。

引导加载程序支持

I²C 组件可用作引导加载程序的通信组件。使用以下配置可为外部系统到引导加载程序的通信协议提供支持：

- **模式**：从器件
- **实现**：可以是固定功能，也可以是基于 UDB
- **数据速率**：必须与主机（引导设备）数据速率一致。
- **从器件地址**：必须与主机（引导设备）所选的从器件地址一致。
- **地址匹配**：硬件是首选项，但不是必须项

有关引导加载程序的更多信息，请参考 *System Reference Guide*（《系统参考指南》）中的 “Bootloader System”（引导加载程序系统）一节。

有关 I²C 通信组件实现的其他信息，请参考引导加载程序协议与 I2C 通信组件的交互（引导加载程序协议与 I2C 通信组件相交互）一节。

I²C 组件为使用引导加载程序提供了一组 API 函数。

函数	说明
I2C_CyBtldrCommStart	使能 I²C 组件，并使其中断。
I2C_CyBtldrCommStop	禁用 I²C 组件并禁用其中断。
I2C_CyBtldrCommReset	将读取和写入 I²C 缓冲区设置为初始状态，并使从器件状态复位。
I2C_CyBtldrCommWrite	允许调用程序将数据写入引导加载程序主机中。该函数将处理轮询以便让数据块完全发送至主机器件。
I2C_CyBtldrCommRead	允许调用程序读取引导加载程序主机中的数据。该函数将处理轮询以便从主机器件完全接收到数据块。

void I2C_CyBtldrCommStart(void)

说明： 此函数使能 I²C 组件，并使其中断。

每个输入的 ²C 写入数据操作都将视为引导加载程序的一个命令。

在引导加载程序向执行命令发送响应之前，每个输入的 I²C 读取数据操作都将返回 OxFF：

参数： 无

返回值： 无

副作用： 无

void I2C_CyBtldrCommStop(void)

说明： 此函数禁用 I²C 组件并禁用其中断。

参数： 无

返回值： 无

副作用： 无

void I2C_CyBtldrCommReset(void)

说明： 此函数将 I²C 缓冲区的读取和写入操作设为初始状态，并将从器件状态复位。

参数： 无

返回值： 无

副作用： 无

cystatus I2C_CyBtldrCommRead(uint8 * Data, uint16 size, uint16 * count, uint8 timeOut)

说明： 此函数允许调用程序读取引导加载程序主机中的数据。该函数将处理轮询以便从主机器件完全接收到数据块。

参数： uint8 *Data: 发送至器件的数据块的指针

uint16 size: 要写入的字节数

uint16 *count: 用于写实际写入字节数的变量指针

uint8 timeOut: 等待的单位数（时间为 10 毫秒），之后会因超时而返回

返回值： cystatus: 如果未遇到任何问题将返回 CYRET_SUCCESS，或返回可详尽描述该问题的值。有关更多信息，请参考 *System Reference Guide*（《系统参考指南》）中的“Return Codes”（返回代码）一节。

副作用： 无

cystatus I2C_CyBtldrCommWrite(uint8 * Data, uint16 size, uint16 * count, uint8 timeOut)

说明： 此函数允许调用程序将数据写入引导加载程序主机中。该函数将处理轮询以便让数据块完全发送至主机器件。

参数： uint8 *Data: 发送至器件的数据块的指针

uint16 size: 要写入的字节数

uint16 *count: 用于写实际写入字节数的变量指针

uint8 timeOut: 等待的单位数（时间为 10 毫秒），之后会因超时而返回

返回值： cystatus: 如果未遇到任何问题将返回 CYRET_SUCCESS，或返回可详尽描述该问题的值。有关更多信息，请参考 *System Reference Guide*（《系统参考指南》）中的“Return Codes”（返回代码）一节。

副作用： 无

固件源代码示例

PSoC Creator 在 Find Example Project（查找示例项目）对话框中提供了大量包括原理图和代码示例的示例代码。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开开始页（Start Page）或 **File**（文件）菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（滤波器选项）可缩小可选项目的列表。

有关更多信息，请参考 PSoC Creator 帮助中的“查找示例项目”主题。

功能描述

此组件支持 I²C 从器件、主控、多主控和多主控从器件配置。以下各节概述了如何使用从器件、主控和多主控组件的信息。

由于 I²C 硬件是由中断驱动的，因此此组件需要您使能全局中断。即使此组件需要中断，您也不需要向 ISR（中断服务子程序）添加任何代码。该组件为所有中断（数据传输）提供服务，与您的代码无关。为此接口分配的存储器缓冲区类似于您的应用程序与 I²C 主控/从器件之间的简单双端口存储器。

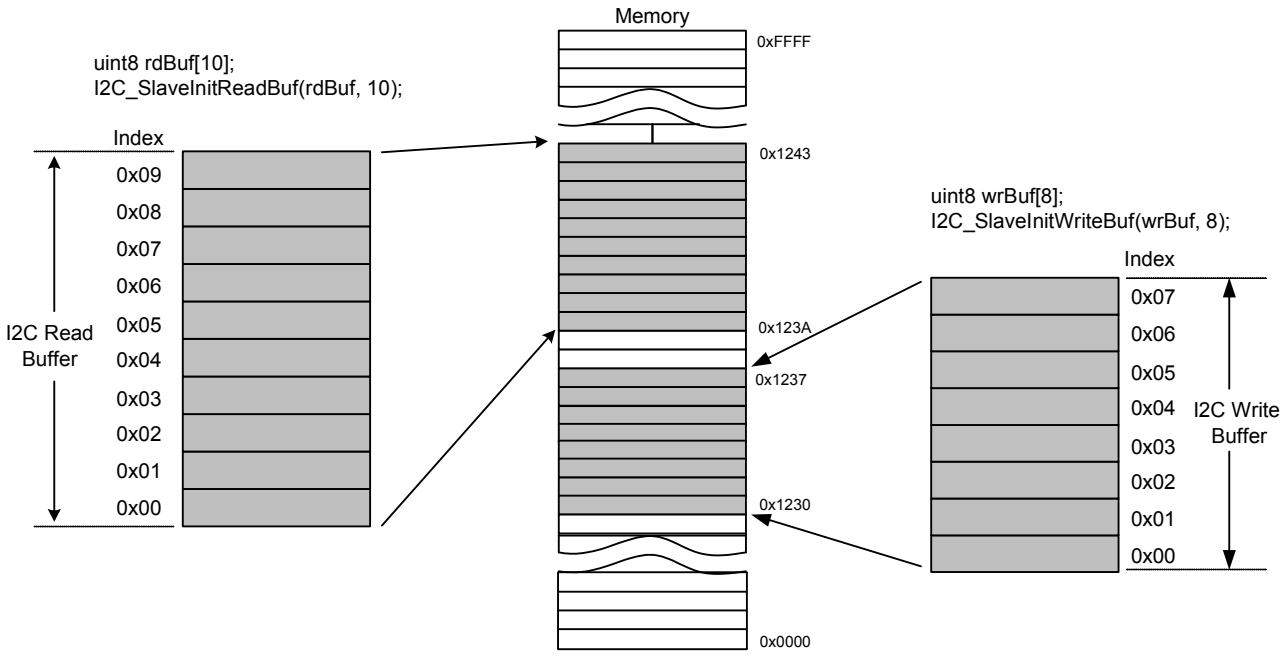
从器件操作

从器件接口是由存储器中的两个缓冲区组成的，一个缓冲区用于保存主控写入到从器件的数据，另一个缓冲区用于保存主控从从器件中读取的数据。请记住，应基于 I²C 主控来执行读取和写入操作。I²C 从器件读取和写入缓冲区是由以下初始化命令来设置的。这些命令不会分配存储器，但会将数组指针和大小复制到内部组件变量中。用于缓冲区的数组必须由设计者对其进行实例化，这是因为组件不会自动生成这些数组。相同的缓冲区可同时用作读取和写入缓冲区，但请务必正确管理数据。

```
void I2C_SlaveInitReadBuf(uint8 * rdBuf, uint8 bufSize)
void I2C_SlaveInitWriteBuf(uint8 * wrBuf, uint8 bufSize)
```

使用上述函数会为读取和写入缓冲区设置指针和字节计数。这些函数的缓冲区大小可能小于或等于实际的数组大小，但是它们不应大于 rdBuf 或 wrBuf 指针指向的可用存储器的大小。

图 1. 从器件缓冲区结构



在调用 I2C_SlaveInitReadBuf() 或 I2C_SlaveInitWriteBuf() 函数时，内部索引将设置为 rdBuf 和 wrBuf 分别指向的数组中的第一个值。当 I²C 主控读取或写入字节时，索引会累计增加，直至偏移小于字节计数。您可以随时针对读取缓冲区和写入缓冲区分别调用 I2C_SlaveGetReadBufSize() 或 I2C_SlaveGetWriteBufSize() 来查询所传输的字节数。读取或写入超过缓冲区字节数的字节时，将会造成溢出错误。该错误将设置在从器件状态字节中，可以使用 I2C_SlaveStatus() API 来读取该错误。

要将索引复位至数组的开始位置，请使用以下命令。

```
void I2C_SlaveClearReadBuf(void)
void I2C_SlaveClearWriteBuf(void)
```

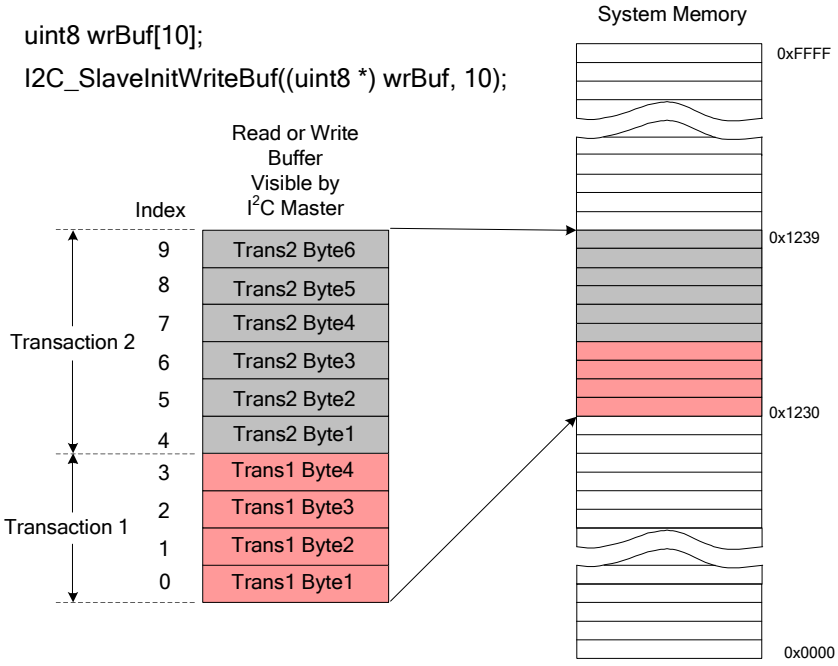
这会将索引复位至零。I2C 主控读取或写入的下一个字节即为数组中的第一个字节。在使用缓冲区的清除命令之前，应先读取或更新数组中的数据。

I²C 主控的多次读取或写入将继续使数组索引增量，直至使用清除缓冲区命令，否则数组索引会试图超过数组大小。图 2 显示的是 I²C 主控执行了两次写入数据操作的示例。第一次写入操作为 4 个字节，第二次写入操作为 6 个字节。第二次数据操作中的第 6 个字节已被从器件否认，表明了

缓冲区操作已结束。如果主控试图在第二次数据操作中写入第 7 个字节，或开始在第三次数据操作时写入其他字节，则每个字节都会被否认并丢弃，直至缓冲区复位为止。

在第一次数据操作将索引复位至零并造成第二次数据操作覆盖第一次数据操作中的数据后，请使用 I2C_SlaveClearWriteBuf() 函数。请务必确保缓冲区溢出时不会丢失数据。在缓冲区索引复位之前，从器件应先处理缓冲区中的数据。

图 2. 系统存储器



读取和写入缓冲区有四个状态位可表示传输完成、正在传输中、缓冲区溢出。当传输开始时，将会设置繁忙标志。当传输完成时，将会设置传输完成标志，并会清除繁忙标志。如果开始第二次传输，则可能会同时设置繁忙标志和传输完成标志。下表所示为读取和写入状态标志。

从器件状态常量	值	说明
I2C_SSTAT_RD_CMPLT	0x01	从器件读取传输已完成
I2C_SSTAT_RD_BUSY	0x02	正在执行从器件读取传输（繁忙）
I2C_SSTAT_RD_OVFL	0x04	主控尝试读取超过缓冲区限制的字节。
I2C_SSTAT_WR_CMPLT	0x10	从器件写入传输已完成
I2C_SSTAT_WR_BUSY	0x20	正在执行从器件写入传输（繁忙）
I2C_SSTAT_WR_OVFL	0x40	主控试图在缓冲区结束之后写入内容。

以下代码示例对写入缓冲区进行了初始化，然后等待传输的完成。在传输完成之后，数据将复制到一个工作数组中。在许多应用程序中，数据不必复制到第二个位置，而是可以在原始缓冲区中

进行处理。读取缓冲区示例与写入缓冲区示例几乎是一致的，不同之处在于需要使用读取函数和常量来替换写入函数和常量。处理数据可能意味着新数据将传入到从器件缓冲区中，而不是从从器件缓冲区中传出。

```
uint8 wrBuf[10];
uint8 userArray[10];
uint8 byteCnt;

/* Initialize write buffer before call I2C_Start */
I2C_SlaveInitWriteBuf((uint8 *) wrBuf, 10);

/* Start I2C Slave operation */
I2C_Start();

/* Wait for I2C master to complete a write */

for(;;)  /* loop forever  */
{
   /* Wait for I2C master to complete a write */
   if(0u != (I2C_SlaveStatus() & I2C_SSTAT_WR_CMPLT))
   {
      byteCnt = I2C_SlaveGetWriteBufSize();
      I2C_SlaveClearWriteStatus();
      for(i=0; i < byteCnt; i++)
      {
         userArray[i] = wrBuf[i]; /* Transfer data */
      }
      I2C_SlaveClearWriteBuf();
   }
}
```

主控/多主控操作

主控和多主控[5,6]除了两点不同，两者的操作基本上相同。在多主控模式下操作时，程序应时刻检查总线，查看总线是否繁忙。另一个主控可能已与另一个从器件进行通信。在这种情况下，该程序必须等到当前操作完成，然后才能启动启动数据操作。该程序会查看返回值，如果另一个主控控制了总线，则该值会设置错误。

第二个不同之处是在多主控模式下，可以同时启动两个主控。如果发生这种情况，其中一个主控将仲裁失败。必须在每个字节传输完成后检查是否出现这种情况。组件会自动检查这种情况，并在仲裁失败后发出错误响应。

[5]如果软件在“启动”条件之后立即设置“停止”条件，则在主控或多主控模式下，PSoC 3 ES2 和 PSoC 5 的固定功能实现中的模块将生成“停止”条件。上述情况将在地址字段（如果写入数据，发送 0xFF）之后发生，此时时钟线保持为低电平。为避免这种情况，请勿在启动之后立即设置“停止”条件；至少传输一个字节，并在发送 NAK 或 ACK 响应之前设置停止条件。

[6]固定功能实现不支持未定义的总线条件。为防止上述情况，可代替采用基于 UDB 的实现。

在执行 I²C 主控操作时可进行如下选择：手动和自动。在自动模式中，会创建缓冲区来保存所有传输数据。在执行写入操作时，缓冲区将预填充要发送的数据。如果自从器件读取数据，则至少需要分配一个具有数据包大小的缓冲区。要在自动模式中将字节数组写入从器件，请使用以下函数。

```
uint8 I2C_MasterWriteBuf(uint8 slaveAddress, uint8 * xferData, uint8 cnt, uint8
mode)
```

slaveAddress 变量是一个右对齐的 7 位从器件地址，其值介于 0 至 127 之间。组件 API 会自动将写入标志附加到地址字节的 LSb 中。第二个参数 xferData 指向要传输的数据数组。Cnt 参数表示要传输的字节数。最后一个参数“mode（模式）”将确定传输开始和结束的方式。可以使用“Restart（重启）”而不是“Start（启动）”来开始数据操作，或在“停止”序列之前终止数据操作。这些选项将允许执行背对背传输，其中最后一个传输不会发送“Stop（停止）”命令，而下一个传输会发送“Restart（重启）”而不是“Start（启动）”命令。

读取操作几乎与写入操作一样。使用带有相同常量的相同参数。

```
uint8 I2C_MasterReadBuf(uint8 slaveAddress, uint8 * xferData, uint8 cnt, uint8
mode);
```

这两个函数都会返回状态。有关 I2C_MasterStatus() 函数返回值的信息，请参见状态表。由于在 I²C 中断代码期间读取和写入传输会在后台完成，因此可以使用 I2C_MasterStatus() 函数来确定该传输何时完成。以下所示代码段是对从器件的典型写入。

```
I2C_MasterClearStatus(); /* Clear any previous status */
I2C_MasterWriteBuf(0x08, (uint8 *) wrData, 10, I2C_MODE_COMPLETE_XFER);
for(;;)
{
   if(0u != (I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT))
   {

      /* Transfer complete. Check Master status to make sure that transfer
         completed without errors. */

      break;
   }
}
```

还可以使用手动方式执行 I²C 主控操作。在此模式中，将使用单个命令来执行写入数据操作的每个部分。

```
status = I2C_MasterSendStart(0x08, I2C_WRITE_XFER_MODE);
if(status == I2C_MSTR_NO_ERROR)   /* Check if transfer completed without errors */
{
   /* Send array of 5 bytes */
   for(i=0; i<5; i++)
   {
      status = I2C_MasterWriteByte(userArray[i]);
      if(status != I2C_MSTR_NO_ERROR)
      {
```

```
            break;
        }
    }
}
I2C_MasterSendStop();      /* Send Stop */
```

手动读取数据操作类似于写入数据操作，但不同之处在于是否应否认最后一个字节。以下示例描述了一个典型的手动读取数据操作。

```
status = I2C_MasterSendStart(0x08, I2C_READ_XFER_MODE);
if(status == I2C_MSTR_NO_ERROR)   /* Check if transfer completed without errors */
{
   /* Read array of 5 bytes */
   for(i=0; i<5; i++)
   {
      if(i < 4)
      {
         userArray[i] = I2C_MasterReadByte(I2C_ACK_DATA);
      }
      else
      {
         userArray[i] = I2C_MasterReadByte(I2C_NAK_DATA);
      }
   }
}
I2C_MasterSendStop();      /* Send Stop */
```

多主控从器件模式操作

在此模式中，可以执行多主控和从器件操作。该组件可以作为从器件被寻址，但固件还会启动主控模式传输。在此模式中，当主控在地址字节期间仲裁失败时，硬件将还原到从器件模式，并且接收的字节会生成从器件地址中断。

有关主控和从器件操作的示例，请参见从器件操作和主控/多主控操作。

```
uint8 userArray[10];
uint8 byteCnt;

/* Initialize write buffer before call I2C_Start */
I2C_SlaveInitWriteBuf((uint8 *) wrBuf, 10);

/* Start I2C Slave operation */
I2C_Start();

/* Wait for I2C master to complete a write */

for(;;)  /* loop forever  */
{
   /* Wait for I2C master to complete a write */
   if(0u != (I2C_SlaveStatus() & I2C_SSTAT_WR_CMPLT))
   {
      byteCnt = I2C_SlaveGetWriteBufSize();
```

```
            I2C_SlaveClearWriteStatus();
            for(i=0; i < byteCnt; i++)
            {
               userArray[i] = wrBuf[i]; /* Transfer data */
            }
            I2C_SlaveClearWriteBuf();
        }
    }
```

主控/多主控操作两个章节。

在使能硬件地址匹配时地址字节仲裁受限：如果主控在地址字节期间仲裁失败，从器件地址中断仅在从器件寻址后生成。在其他情况下，基于中断的函数将找不到仲裁失败状态。软件地址检测可以消除这种可能性，但也会排除在硬件地址匹配时唤醒功能。

基于手动的函数 I2C_MasterSendStart() 在上述案例中提供了正确的状态信息。

开始多主控从器件传输

使用多主控从器件时，可以在任意时刻对从器件寻址。当总线处于闲置状态时，多主控必须花费时间做好充足准备才能生成“启动”条件。在此期间，系统将对从器件寻址，并因此停止多主控数据操作，而继续执行从器件操作。请务必避免中断从器件操作；在生成启动条件防止数据操作进入寻址阶段之前，必须禁用 I²C 中断。此操作使您可以中止多主控数据操作，并正确启动从器件操作。禁用 I²C 中断，可能会发生下列状况：

- 生成“启动”之前总线繁忙（从器件操作正在进行中，或者其他通信正占用总线）。多主控无法生成“启动”条件。使能 I²C 中断后，从器件操作仍在继续进行。I2C_MasterWriteBuf()、I2C_MasterReadBuf() 或 I2C_MasterSendStart() 的调用返回状态 **I2C_MSTR_BUS_BUSY**。
- 生成“启动”之前，总线处于闲置状态。使能 I²C 中断后，多主控在总线上生成“启动”条件，但操作仍在继续。I2C_MasterWriteBuf()、 I2C_MasterReadBuf() 或 I2C_MasterSendStart() 的调用返回状态 **I2C_MSTR_NO_ERROR**。
- 生成“启动”之前，总线处于闲置状态。多主控尝试生成“启动”条件，但在此之前另一多主控对从器件寻址，此时总线将进入繁忙状态。“启动”条件生成将列入队列。由于禁用了 I²C 中断，从器件操作停止于寻址阶段。. 使能 I²C 中断后，队列中的多主控数据操作将被取消，从器件操作将继续进行。I2C_MasterWriteBuf() 或 I2C_MasterReadBuf() 的调用为对此作出响应，而是返回 **I2C_MSTR_NO_ERROR**。多主控数据操作被取消后，I2C_MasterStatus() 返回 **I2C_MSTAT_WR_CMPLT** 或一同返回 **I2C_MSTAT_RD_CMPLT** 和 **I2C_MSTAT_ERR_XFER**（其他所有错误状况位都已清除）。I2C_MasterSendStart() 的调用返回错误状态 **I2C_MSTR_ABORT_XFER**。

中断函数操作

- I2C_MasterWriteBuf();

- I2C_MasterReadBuf();

```
I2C_MasterClearStatus();    /* Clear any previous status */

I2C_DisableInt();           /* Disable interrupt */

status = I2C_MasterWriteBuf(0x08, (uint8 *) wrData, 10, I2C_MODE_COMPLETE_XFER);
/* Try to generate, start. The disabled I2C interrupt halt the transaction on
address stage in case of Slave addressed or Master generates start condition */

I2C_EnableInt();            /* Enable interrupt and proceed Master or Slave
transaction */

for(;;)
{
   if(0u != (I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT))
   {
     /* Transfer complete. Check Master status to make sure that transfer
        completed without errors. */
      break;
   }
}

if (0u != (I2C_MasterStatus() & I2C_MSTAT_ERR_XFER))
{
      /* Error occurred while transfer, clean up Master status and
         retry the transfer */
}
```

手动函数操作

手动多主控操作假设 I²C 中断已被禁用，但其实最好的做法是采用以下预防措施：

```
I2C_DisableInt();        /* Disable interrupt */
status = I2C_MasterSendStart(0x08, I2C_WRITE_XFER_MODE);;    /* Try to generate
start condition */
if (status == I2C_MSTR_NO_ERROR)   /* Check if start generation completed without
errors */
{
   /* Proceed the write operation */
   /* Send array of 5 bytes */
   for (i=0; i<5; i++)
   {
      status = I2C_MasterWriteByte(userArray[i]);
      if (status != I2C_MSTR_NO_ERROR)
      {
         break;
      }
   }
    I2C_MasterSendStop();      /* Send Stop */
}
I2C_EnableInt();         /* Enable interrupt, if it was enabled before */
```

在硬件地址匹配时唤醒

如果满足以下条件，则有可能发生在 I²C 地址匹配时从睡眠模式唤醒事件：

- I²C 从器件已使能。选择从器件模式或多主控从器件模式。
- 选择 I²C 硬件地址检测。
- SIO 对连接至 SCL 和 SDA，并且在定制器中选择正确的对：I2C0 — SCL P12[4]，SDA P12[5] 且 I2C1 — SCL P12[0]，SDA P12[1]。

I²C 组件定制器可控制上述情况，**不包括引脚分配正确的情况**。

工作原理

I²C 模块将对睡眠模式下的 I²C 总线的数据操作进行响应。如果输入的地址与从器件地址相匹配，I²C 将唤醒该系统。当该地址匹配时，会激活唤醒中断来唤醒系统，并且 SCL 将置于低电平状态。当系统唤醒并且 CPU 确定了数据操作中的下一个操作后，将会发出 ACK 应答。

唤醒和时钟延展

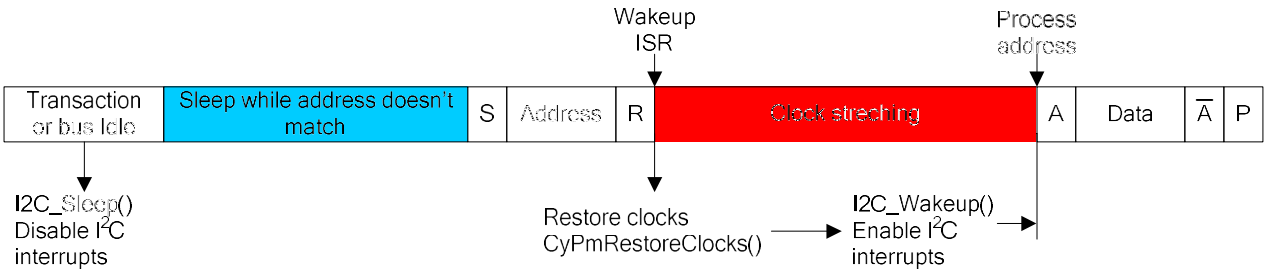
I²C 从器件将在退出睡眠模式时使时钟延展。您必须恢复系统中的所有时钟，然后才能继续 I²C 数据操作。在进入睡眠状态之前将会禁用 I²C 中断，并且仅在调用 I2C_Wakeup() 函数后才会使能该中断。在唤醒和结束调用 I2C_Wakeup() 的期间，SCL 线将置于低电平状态。

代码示例：

```
...
I2C_Sleep();            /* Go to Sleep and disable I2C interrupt */
CyPmSaveClocks();       /* Save clocks settings */

CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_I2C);

CyPmRestoreClocks();    /* Restore clocks */
I2C_Wakeup();           /* Wakeup, enable I2C interrupt and ACK the address, until
end of this call the SCL is pulled low */
...
```



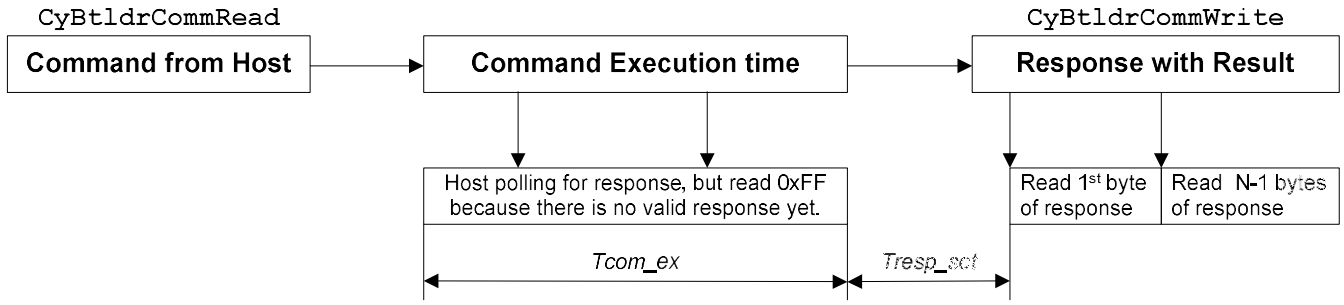
引导加载程序协议与 I²C 通信组件的交互

引导加载程序协议作为命令（写入数据操作）和响应（读取数据操作）进行实施。

从主机发出命令到引导加载程序发回应答的时间段是命令的执行时间。引导加载程序的 I²C 通信组件是按如下方式进行设计的：当主机请求应答并且引导加载程序仍在执行命令时，该应答为 0xFF。

使能：I²C 引导加载程序通信组件准备接收该命令，并且尚未得到一个有效的应答。主机执行的所有读取数据操作都将返回 0xFF。所有写入数据操作都将视为命令。

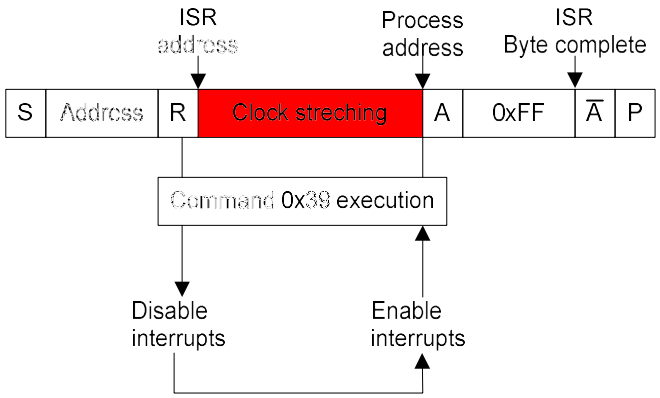
引导加载程序流程：主机将通过一个写入数据操作发出命令，并启动轮询来获取应答。在引导加载程序传递有效应答之前，都将使用 0xFF 来应答 I²C 通信组件。接收 0x01 之后，主机必须再执行一次读取才可获得响应的剩余 N – 1 个字节。在两次读取完成之后，结果将进行合并，形成完整的响应包。



主机必须读取一个字节来执行轮询，读取多个字节可能会破坏应答。以 0xFF 0x01 0x03 为例（主机所读取的是响应中的两个字节，而是不是一个字节），由于 0x01 和 0x03 已读取完毕，完整响应的下一次读取会返回两个无效值。

如何避免轮询：应根据系统设置（CPU 速度、编译器和编译器优化级别）来测量命令执行时间 (Tcom_ex) 以及应答设置时间 (Tans_set)。主机必须在该时间过后请求响应。命令执行时间会因不同的命令而有所变化，因此应该选择较长的时间。

在轮询的同时进行时钟延展：I²C 通信组件在进行操作时要求使能中断。将一行闪存数据写入器件的命令程序行 (0x39) 要求禁用中断。如果在禁用中断的同时 I²C 通信组件接受了该地址，则会发生时钟延展。

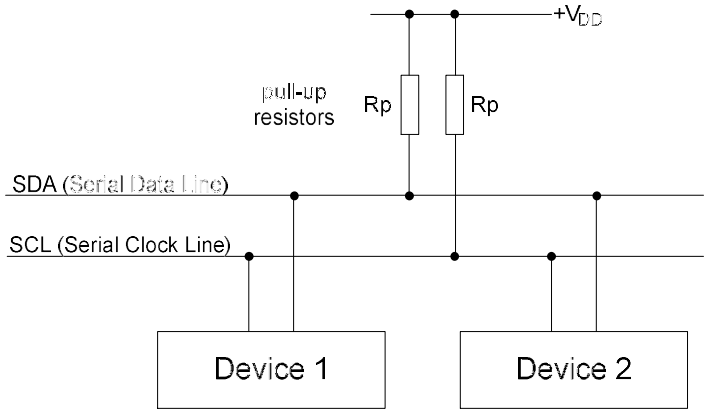


如何避免时钟延展：要避免时钟延展，应根据系统设置（CPU 速度、编译器和编译器优化级别）来测量命令程序行 (0x39) 执行时间 (Tcom_ex)。主机必须在该时间过后请求响应。

外部电气连接

如图 3 所示，I²C 总线需要外部上拉电阻。上拉电阻 (R_P) 取决于供电电压、时钟频率和总线电容。将输出阶段的任何器件（主控或从器件）的最小灌电流设置为不小于 3 mA（在 V_{OLmax} = 0.4V 的条件下）。这会将 5V 系统的最小上拉电阻值限制在 1.5 kΩ 左右。R_P 的最大值取决于总线电容和时钟速度。对于总线电容为 150 pF 的 5V 系统，上拉电阻不超过 6 kΩ。有关调整上拉电阻和其他物理总线规范大小的更多信息，请参见 NXP 网站 (www.nxp.com) 上的 *The I²C -Bus Specification*（I²C — 总线规范）。

图 3. 器件与 I²C 总线的连接



注 从赛普拉斯或其获得分许可的其中一个联营公司处购买 I²C 组件，即可根据 Philips I²C 专利权获得一份许可，以便在 I²C 系统中使用这些组件，但前提是该系统符合 Philips 定义的 I²C 标准规范。自 2006 年 10 月 1 日起，飞利浦半导体开始采用一个新的商标名称 — NXP 半导体。

中断服务子程序

中断服务子程序是组件代码本身使用的程序，您不应该对其进行修改。

从器件操作具备下列用户选择：

- 定制涵盖内容和定义
- 添加的地址比较
- 准备读取缓冲区

主控操作未系统任何用户选择：

I²C 组件可对大部分操作应用中断；数据操作状态将在此时更新。不对状态读取和清除函数提供防中断保护。下文大致列出了这些函数：

主控或多主控：

- I2C_MasterStatus()
- I2C_MasterClearStatus()
- I2C_MasterGetReadBufSize()
- I2C_MasterGetWriteBufSize()
- I2C_MasterClearReadBuf()
- I2C_MasterClearWriteBuf()

从器件：

- I2C_SlaveStatus()
- I2C_SlaveClearReadStatus()
- I2C_SlaveClearWriteStatus()
- I2C_SlaveInitReadBuf()
- I2C_SlaveInitWriteBuf()
- I2C_SlaveGetReadBufSize()
- I2C_SlaveGetWriteBufSize()
- I2C_SlaveClearReadBuf()
- I2C_SlaveClearWriteBuf()

寄存器

所提供的函数支持大部分应用程序所需的通用运行时函数。以下寄存器参考信息为高级用户提供了简要的说明。I2C_Data 寄存器无需使用该 API，便可直接将数据写入到总线中。这可能对使用 CPU 或 DMA 非常有帮助。

可用于每个 I²C 组件配置的寄存器将根据作为固定功能还是 UDB 进行实现来分组。

固定功能主控/从器件寄存器

有关这些寄存器的更多信息，请参考该芯片的技术参考手册 (TRM)。在下列定义中，将使用星号 (*) 来表示添加到 Production PSoC3 芯片中的所有位。

I2C_XCFG

可以使用固定功能硬件模块中的扩展配置寄存器来配置硬件地址模式和时钟源。

位	7	6	5	4	3	2	1	0
值	csr_clk_en	i2c_on*	ready_to_sleep*	force_nak*	RSVD			hw_addr_en

- csr_clk_en：用于使能固定功能模块核心逻辑的关断。
- i2c_on*：用于选择 I²C 模块作为唤醒源。
- ready_to_sleep*：用于通知模块正准备进入睡眠。
- force_nak*：用于强制否认该数据操作。
- hw_addr_en：用于使能硬件地址比较模式。

I2C_ADDR

可以使用固定功能硬件模块中的从器件地址寄存器为硬件比较模式（如果上述 XCFG 寄存器中使能了该模式）配置从器件器件地址。

位	7	6	5	4	3	2	1	0
值	RSVD	slave_address						

- slave_address：用于为硬件地址比较模式定义 7 位从器件地址。

I2C_CFG

可以使用固定功能硬件模块中的配置寄存器来配置基本功能。

位	7	6	5	4	3	2	1	0
值	sio_select	pselect	bus_error_ie	stop_ie	clock_rate[1:0]		en_mstr	en_slave

- sio_select：用于为 SCL 和 SDA 选择 SIO1 和 SIO2 线，必须为此位设置 pselect 才有效。
- pselect：用于为 SCL 和 SDA 线选择 SIO 直连或 DSI 路由的 GPIO/SIO 引脚。
- bus_error_ie：用于为 bus_error 使能中断生成。
- stop_ie：用于在停止位检测上使能中断生成。
- clock_rate：用于选择 16 位或 32 位过采样。Production PSoC 3 仅使用 bit2。
- en_mstr：用于使能主控模式。
- en_slave：用于使能从器件模式。

I2C_CSR

可以使用固定功能硬件模块中的控制和状态寄存器进行运行时控制和状态反馈。

位	7	6	5	4	3	2	1	0
值	bus_error	lost_arb*	stop_status	ack	address	transmit	lrb	byte_complete

- bus_error：总线错误检测状态位。这必须将“0”写入到此位位置来清除。
- lost_arb*：仲裁失败检测状态位。
- stop_status：停止检测状态位。这必须将“0”写入到此位置来清除。
- ack：确认控制位。要确认最后接收的一个字节，必须将此位设置为“1”，要否认最后接收的一个字节，必须将其设置为“0”。
- address：如果上述收到的字节是地址字节，则会设置该值。
- transmit：固件使用该值来定义字节传输的方向。
- lrb：最后接收的一个位的状态。此位表示接收器对最后传输的一个字节所作的第 9 位 (ACK/NACK) 响应的状态。
- byte_complete：传输或接收最后一次读取此寄存器以来的状态。在传输模式中，此位表示自最后一次读取以来已传输了 8 位数据和 ACK/NAK 应答。在接收模式中，此位表示自最后一次读取此寄存器以来已接收了 8 位数据。

I2C_DATA

可以使用固定功能硬件模块的数据寄存器进行运行时传输和数据接收。

位	7	6	5	4	3	2	1	0
值	data							

- data：在传输模式中，会使用传输的数据来写入此寄存器。在接收模式中，将在收到 byte_complete 的状态下读取此寄存器。

I2C_MCSR

可以使用固定功能硬件模块中的主控控制和状态寄存器进行主控模式操作的运行时控制和状态反馈。

位	7	6	5	4	3	2	1	0
值	RSVD			stop_gen*	bus_busy	master_mode	restart_gen	start_gen

- stop_gen*：如果设置该值，当字节传输结束时将在主控发送器模式中生成“停止”
- bus_busy：表示总线状态。0 表示已检测到“停止”条件，1 表示已检测到“启动”条件。
- master_mode：表示已生成有效的“启动”条件，硬件器件作为总线主控正在运行。
- restart_gen：控制寄存器以在总线上创建“重启”条件。硬件将在“重启”实现后清除此位（可能在设置该条件完成状况轮询之后作为读取状态）。
- start_gen：控制寄存器以在总线上创建“启动”条件。硬件将在“启动”实现后清除此位（可能在设置该条件完成状况轮询之后作为读取状态）。

UDB 主控

UDB 寄存器定义是从 I²C 的 Verilog 实现派生的，有关这些寄存器定义的更多信息，请参考特定模式实现 Verilog。

I2C_CFG

可以使用 UDB 实现中的控制寄存器对硬件进行运行时控制。

位	7	6	5	4	3	2	1	0
值	start_gen	stop_gen	restart_gen	ack	RSVD	transmit	en_master	RSVD

- start_gen：设为 1 可在总线上生成“启动”条件。必须在初始化下一个数据操作之前由固件清除此位。

- stop_gen：设为 1 可在总线上生成“停止”条件。必须在初始化下一个数据操作之前由固件清除此位。
- restart_gen：设为 1 可在总线上生成“重启”条件。必须在生成“重启”条件后由固件清除此位。
- ack：设为 1 会拒绝下一个读取字节。清除以确认下一个读取字节。必须由介于字节之间的固件来清除此位。
- transmit：设为 1 会将当前模式设置为传输，或清零以接收数据字节。必须在启动下一个传输或接收数据操作之前由固件清除此位。
- en_master：设为 1 会使能主控功能。

I2C_CSR

在 UDB 实现中可以使用状态寄存器由硬件获取实时状态反馈。在计数器的输入时钟沿，系统将寄存其配置为 mode = 1 的全部位的状态数据。这些位是粘滞的并在某次读取状态寄存器后清除。所有其他位都配置为 mode = 0，并且可以直接将这些位从输入读取到状态寄存器中。它们不是粘滞位，因此在读取时不会清除。在下列定义中，所有配置为 mode = 1 的位都用星号 (*) 表示。

位	7	6	5	4	3	2	1	0
值	RSVD	lost_arb*	stop_status*	bus_busy	address	master_mode	lrb	byte_complete

- lost_arb*：如果设置该值，则表示仲裁失败（多主控和多主控从器件模式）。
- stop_status*：如果设置该值，则表示已在总线上检测到“停止”条件。
- bus_busy：如果设置该值，则表示总线繁忙。目前正在传输或接收数据。
- address：地址检测。如果设置该值，则表示已发送地址字节。
- master_mode：表示生成一个有效“启动”条件，同时硬件设备运行为总线主控。
- lbr：最后接收的一个位。表示最后接收的一个位的状态（为最后传输的一个字节而收到的 ACK/NAK）。Cleared = ACK 和 set = NAK。
- byte_complete：传输或接收最后一次读取此寄存器以来的状态。在传输模式中，此位表示自最后一次读取以来已传输了 8 位数据和 ACK/NAK 应答。在接收模式中，此位表示自最后一次读取此寄存器以来已接收了 8 位数据。

I2C_INT_MASK

可以使用 UDB 实现中的中断屏蔽寄存器来配置将哪些状态位使能为中断源。可以使用与上述 I2C_CSR 中的状态寄存器位域定义进行 1 对 1 位关联，将任何状态寄存器位使能为中断源。

I2C_ADDRESS

可以使用 UDB 实现中的从器件地址寄存器为硬件比较模式配置从器件设备地址。

位	7	6	5	4	3	2	1	0
值	RSVD	slave_address						

- slave_address：用于为硬件地址比较模式定义 7 位从器件地址。

I2C_DATA

可以使用 UDB 实现模块中的数据寄存器进行运行时传输和数据接收。

位	7	6	5	4	3	2	1	0
值	data							

- data：在传输模式中，会使用传输的数据来写入此寄存器。在接收模式中，将在收到 byte_complete 的状态下读取此寄存器。

I2C_GO

在主控执行传输操作时，Go 寄存器会强制传输数据寄存器中的数据。在主控执行接收操作时，Go 寄存器会强制接收数据寄存器中的数据。对此寄存器执行的任何写入都会强制执行此操作，无论将写入哪个值。

UDB 从器件

UDB 寄存器定义是从 I²C 的 Verilog 实现派生的。有关这些寄存器定义的更多信息，请参考特定模式实现 Verilog。

I2C_CFG

可以使用 UDB 实现中的控制寄存器对硬件进行运行时控制。

位	7	6	5	4	3	2	1	0
值	RSVD	RSVD	RSVD	nak	any_address	transmit	RSVD	en_slave

- nak：如果设置该值，它将用于否认最后接收的一个字节。必须由介于字节之间的固件来清除此位。
- any_address：如果设置该值，它将用于使能器件以响应它接收的任何器件地址，而不仅是 I2C_ADDRESS 中提供的单一地址。
- transmit：用于设置传输或接收数据的模式。必须由介于字节之间的固件来清除此位。Set = transmit 和 cleared = receive。

- en_slave：设为 1 会使能从器件功能。

I2C_CSR

可以使用固定功能硬件模块中的控制和状态寄存器进行运行时控制和状态反馈。在计数器的输入时钟沿，系统将寄存其配置为 mode = 1 的全部位的状态数据。这些位是粘滞的并在某次读取状态寄存器后清除。所有其他位都配置为 mode = 0，并且可以直接将这些位从输入读取到状态寄存器中。它们不是粘滞位，因此在读取时不会清除。在下列定义中，所有配置为 mode = 1 的位都用星号 (*) 表示。

位	7	6	5	4	3	2	1	0
值	RSVD	RSVD	stop*	RSVD	address	RSVD	lrb	byte_complete

- stop*：如果设置该值，则表示已在总线上检测到“停止”条件。
- 地址：地址检测。如果设置该值，则表示已接收地址字节。
- lbr：最后接收的一个位。表示最后接收的一个位的状态（为最后传输的一个字节而收到的 ACK/NAK）。Cleared = ACK 和 set = NAK。
- byte_complete：传输或接收最后一次读取此寄存器以来的状态。在传输模式中，此位表示自最后一次读取以来已传输了 8 位数据和 ACK/NAK。在接收模式中，此位表示自最后一次读取此寄存器以来已接收了 8 位数据。

I2C_INT_MASK

可以使用 UDB 实现中的中断屏蔽寄存器来配置将哪些状态位使能为中断源。可以使用与 I2C_CSR 中的状态寄存器位域定义进行 1 对 1 位关联，将任何状态寄存器位使能为中断源。操作期间所用的中断源：byte_complete 和停止。

I2C_ADDRESS

可以使用 UDB 实现中的从器件地址寄存器为硬件比较模式配置从器件设备地址。

位	7	6	5	4	3	2	1	0
值	RSVD	slave_address						

- slave_address：用于为硬件地址比较模式定义 7 位从器件地址。

I2C_DATA

可以使用 UDB 实现模块中的数据寄存器进行运行时传输和数据接收。

位	7	6	5	4	3	2	1	0
值	data							

- data：在传输模式中，会使用传输的数据来写入此寄存器。在接收模式中，将在收到 byte_complete 的状态下读取此寄存器。

I2C_GO

在主控执行传输操作时，Go 寄存器会强制传输数据寄存器中的数据。在主控接收时，Go 寄存器强制数据寄存器接收数据。对此寄存器执行的任何写入都会强制执行此操作，无论写入哪个值。

直流电和交流电电气特性（FF 实现）

下面的值表示了预计性能，它们基于初始特性数据。

I²C 直流电规范

参数	说明	条件	最小值	典型值	最大值	单位
	模块电流消耗	已使能，针对 100 kbps 进行配置	--	--	250	µA
		已使能，针对 400 kbps 进行配置	--	--	260	µA
		从睡眠模式唤醒	--	--	30	µA

I²C 交流电规范

参数	说明	条件	最小值	典型值	最大值	单位
	比特率		--	--	1	Mbps

直流电和交流电电气特性（UDB 实现）

下面的值表示了预计性能，它们基于初始特性数据。

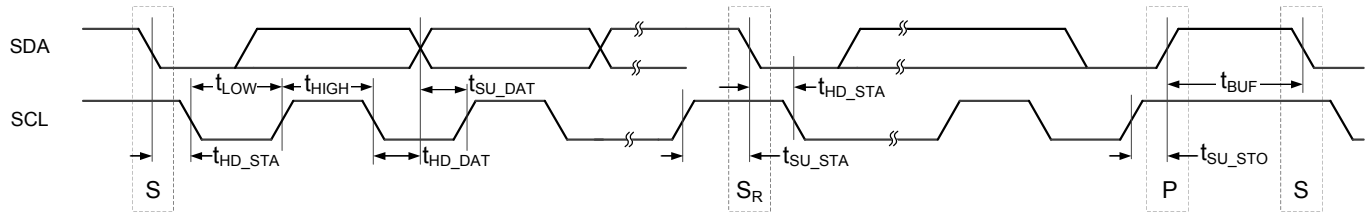
时序特性“所有路由的最大值”

参数	说明	最小值	典型值	最大值	单位
f_{SCL}	SCL 时钟频率				

参数	说明		最小值	典型值	最大值	单位
		标准模式	–	100	–	kHz
		快速模式	–	400	–	kHz
		增强型快速模式	–	1000	–	kHz
f_{CLOCK}	组件输入时钟频率		–	16 × f_{SCL}	–	kHz
t_{LOW}	SCL 时钟的低电平周期		–	8	–	t_{CY_clock} [7]
t_{HIGH}	SCL 时钟的高电平周期		–	8	–	t_{CY_clock} [7]
t_{HD_STA}	保留时间（重复）启动条件		–	15	–	t_{CY_clock}
t_{SU_STA}	重复 START 条件的建立时间		–	9	–	t_{CY_clock}
t_{HD_DAT}	数据保留时间		–	1	–	t_{CY_clock}
t_{SU_DAT}	数据建立时间		–	7	–	t_{CY_clock}
t_{SU_STO}	STOP 条件的建立时间		–	9	–	t_{CY_clock}
t_{BUF}	STOP 和 START 条件之间的总线空闲时间		–	32	–	t_{CY_clock}
t_{RESET}	复位脉冲宽度		–	2	–	t_{CY_clock}

[7] $t_{CY_clock} = 1/f_{CLOCK}$。上述是一个时钟周期的循环时间

图 4. 数据转换时序图



如何将 STA 结果用于特性数据

额定路由最大值是通过使用静态时序分析 (STA) 进行多次测试而收集的。您可以结合使用 STA 结果与下列方法为你的设计计算最大值。

在所命名的组件时钟的时钟汇总中的时序结果中提供了 **f_{CLOCK}** 最大组件时钟频率。最大组件时钟限制为 f_{CLOCK} = 16 × f_{SCL} = 16 × 1000 kHz = 16 MHz，因此 STA 报告只能用于检查是否违反了报告的最大时钟频率（最大频率）。下面是 ***_timing.html*** 文件中组件时钟限制的示例：

+Clock Summary

Clock	Actual Freq	Max Freq	Violation
BUS_CLK	48.000 MHz	112.664 MHz	
Clock	16.000 MHz	20.571 MHz	

其余参数特定于实现，在时钟循环中进行计算。I²C 组件与 2007 年 6 月的 I²C 总线规范修订版 3 兼容。

t_{SCL} 最高可将 I²C 数据速率值定义为 1000 kbps；标准数据速率为 50、100、400 和 1000 kbps。需要 16x 输入时钟才能获取所需的数据速率。

t_{LOW} SCL 时钟的低电平周期。组件生成 50 % 占空比时钟循环。

t_{HIGH} SCL 时钟的高电平周期。组件生成 50 % 占空比时钟循环。

t_{HD_STA} 在 SDA 为了生成“启动”条件而从高电平切换到低电平之后 SCL 信号为高电平的最小时间量。经过此时间段之后，会生成第一个时钟脉冲。

t_{SU_STA} 在 SDA 为了生成“启动”条件而从高电平切换到低电平之前 SCL 信号为高电平的最小时间量。

t_{HD_DAT} SCL 信号的下降沿后数据应当有效的最小时间量。

t_{SU_DAT} SCL 信号的上升沿前数据应当有效的最小时间量。

t_{SU_STO} 在 SDA 信号为了生成停止条件而从低电平切换到高电平之前 SCL 信号应当为高电平的最小时间量。

t_{BUF} 在停止条件后总线视为空闲的时间段。

t_{RESET} 组件实现需要两循环宽度的复位信号。

组件更改

本节介绍组件与以前版本相比的主要更改。

版本	更改说明	更改/影响原因
3.1	将定义由 I2C_SSTAT_RD_CMPT 更改为 I2C_SSTAT_RD_CMPLT 将定义由 I2C_SSTAT_WR_CMPL 更改为 I2C_SSTAT_WR_CMPLT	旨在符合读取和写入完成标志的主控定义。该组件同时支持两种定义，但是 I2C_SSTAT_RD_CMPT 和 I2C_SSTAT_WR_CMPT 不再使用。
	为 .cyre 中的所有的 API 添加 CYREENTRANT 关键字。	并非所有的 API 都是可重载入的。组件 API 源文件中的注释表示哪些函数不建议使用。 在安全方式下，消除不建议再次使用的函数的编译器警告所需的更改：通过标志或关键区段防止并行调用的发生。
3.0.a	对数据表进行了少量编辑和更新	
3.0	更改了定制器的外观	更加直观且易于使用
	增加了 UDB 时钟容差设置。	避免针对多个配置显示时钟警告。
	在退出休眠模式后，FF 实现中的组件通过从“Enable from Sleep”（从睡眠中使能）选项正确恢复配置。	修复休眠模式下的组件行为。
	在调用 I2C_Start() 后使能 I²C 中断。	在从器件模式下，用户忘记在 I2C_Start() 之后使能中断时未提示错误。
	添加了对 UDB 实现的内部时钟支持。	增强功能。
	删除了函数 I2C_SlaveGetWriteByte() 和 I2C_SlavePutReadByte()	无法再使用上述函数。
2.20	添加了对组件 UDB 实现的引导加载程序通信支持。	允许在设计中使用支持引导加载的多个 I²C 组件。这可与 cy_boot v2.21 附带的定制加载程序功能结合使用。
	修复了在基于零数据保持时间的数据操作过程中放置错误的启动条件检测。	由于主控的零数据保持时间，从器件正常运行。
2.10	添加“多主控 — 从器件”模式	支持将“多主控 — 从器件”功能添加到组件中。
	定制器标签和说明编辑	改善了组件定制器的外观和内容。
	更改了 I²C 引导加载程序通信组件行为以抑制读取时的时钟伸展。	如果在启动引导进程之前发出读取命令，则 I²C 引导加载程序通信组件永远将 SCL 保持低电平。
	向数据表中添加了特性数据。	
	对数据表进行了少量编辑和更新	
2.0.a	将组件移动到组件目录的子文件夹中	

版本	更改说明	更改/影响原因
	对数据表进行了少量编辑和更新	
2.0	添加了睡眠/唤醒和初始化/使能 API。	为支持低功耗模式并提供常用接口，以单独控制大多数组件的初始化和使能。
	更新了组件，使之可以支持 Production PSoC 3 及其更高版本。更新了 Configure（配置）对话框。	新增了支持 Production PSoC 3 器件的要求，因此创建了新的 2.0 版。 版本 1.xx 支持 PSoC 3 ES2 和 PSoC 5 芯片版本
	为“在 I²C 地址匹配时唤醒”功能添加了 I2C 引脚连接端口的配置。	I²C 组件将能够在 I²C 地址匹配时从睡眠模式唤醒器件。
	更新了数据手册。	更新了参数和设置、时钟选择以及资源选择，以体现 UDB 实现。 解决了示例代码中的错误。
	添加了针对组件的重新进入支持。	允许用户进行特定的 API 重新进入（如果需要重新进入）。

© 赛普拉斯半导体公司，2012。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品的内嵌电路之外，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。也不根据专利或其他权利以明示或暗示的方式授予任何许可。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。

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。此处引用的所有其他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。除上述指定的用途之外，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的赛普拉斯软件许可协议限制。