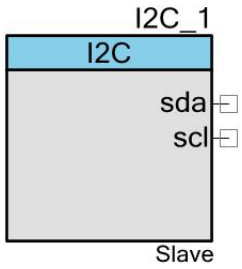
I2C 主设备/多主设备/从设备

3.30



特性

- 工业标准 NXP® I²C 总线接口
- 支持从设备、主设备、多主设备和多主设备从设备操作
- 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 从设备、主设备和多主设备配置。I²C 总线是 Philips 基于行业标准开发的两线硬件接口。主设备在 I²C 总线上启动所有通信，并为所有从设备提供时钟。

I²C 组件支持的标准时钟速率高达 1000 kbps。它与 I²C 标准、快速和超快速模式的器件兼容[1]，装置见 NXP I²C 总线规范。I²C 组件与其他第三方从设备器件和主设备器件相兼容。

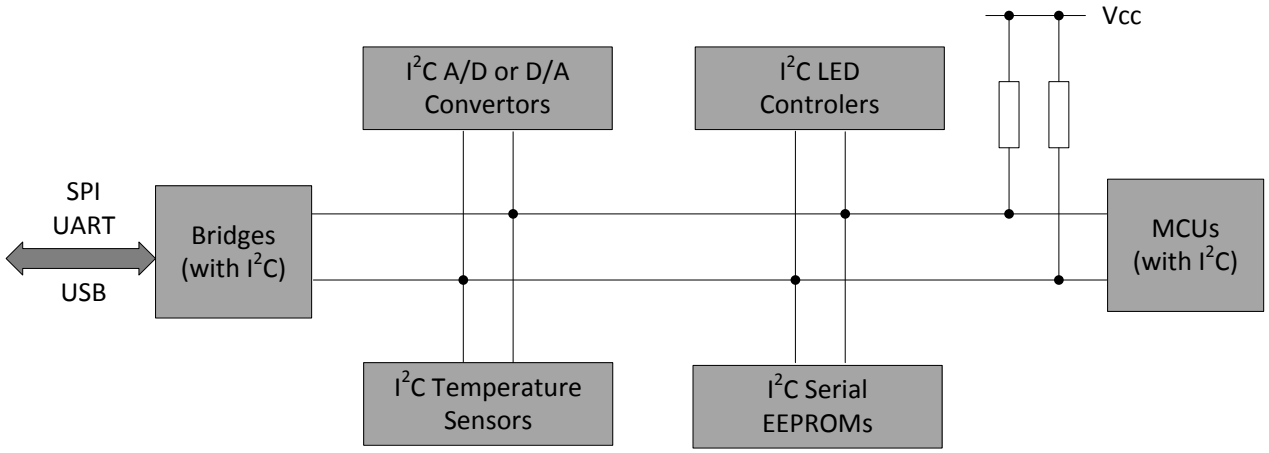
注意：此版本的组件数据手册涵盖了固定的硬件 I²C 模块和 UDB 版本。

---

[1] I²C 外设在以下范围内不符合 NXP I²C 规范：模拟短时脉冲滤波器、I/O V_{OL}/I_{OL}、I/O 迟滞。I²C 模块带有数字毛刺滤波器（在睡眠模式下无效）。通过将各个 I/O 设置为慢速可以达到组件在快速工作模式下的最小下降时间。更多详细信息，请参考器件数据手册的“输入和输出”一节中的 I/O 电气指标。

在何种情况下使用 I²C 组件

当您将单一电路板或小系统中的多个器件进行联网时，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设备和多从设备、多主设备或主设备和从设备的组合。



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，在 I/O 说明部分中所列出的特定条件下，该 I/O 可能不可见。

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。将该引脚连接至 sda，并配置为开漏状态，然后驱动设置为低电平。

SCL — 输入/输出

串行时钟（SCL）是来自主 I²C 的时钟。虽然从设备从不会生成时钟信号，但它能够使时钟保持在低电平的状态，并使总线停顿，直至它准备发送数据或确认/否认（ACK/NACK）[2] 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（Open-Drain-Drives-Low）。

[2] NAK 是“negative acknowledgment”（否定确认）或“not acknowledged”（未确认）的缩写。I²C 文档中通常用“NACK”来表示，网络上的其它位置则用“NAK”来表示。两者的意思相同。

clock — 输入*

将 **Implementation**（实现）参数设置为 **UDB** 时，可以使用时钟输入。UDB 版本需要一个时钟以提供 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*

将 **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。这种情况仅适用于硬件复位。而软件必须使用 I2C_Stop()和 I2C_Start() API 进行单独复位。复位输入可以保持悬空，而不与外部连接。如果复位线上无任何连接，则组件会给它分配一个常量逻辑 0。

I²C 总线复用

只在**外部 OE 缓冲**选项（位于 **Advanced** 选项卡下）被选中时，下面的输入和输出才可用。内部 OE 缓冲被移除，并且双向 scl 和 sda 终端被单独的输入（sda_i 和 scl_i）和输出（sda_o 和 scl_o）替换。这样便允许内部 I²C 总线复用。

sda_i — 输入*

串行数据输入信号用于接收数据。对于 SDA 双向引脚，sda_i 终端应该连接到一个三态缓冲反馈信号。

scl_i — 输入*

串行时钟输入时钟信号用于在从模式中接收时钟，并在主模式中实现时钟同步化。对于 SCL 双向引脚，scl_i 终端应该连接到一个三态缓冲反馈信号。

sda_o — 输出*

串行数据输出信号用于传输数据。对于 SDA 双向引脚，sda_o 终端应该连接到一个三态缓冲直接输入信号。

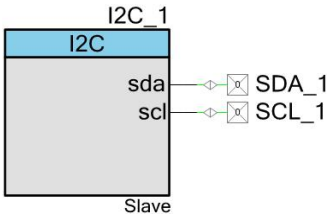
scl_o — 输出*

从设备或主设备使用串行时钟输出信号使时钟线保持在低电平并使总线停顿，直至总线准备好操作为止。对于 SCL 双向引脚，scl_o 终端应该连接到一个三态缓冲直接输入信号。

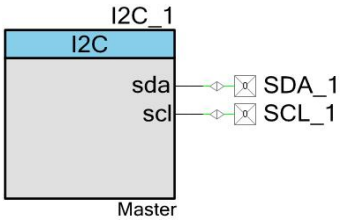
原理图宏信息

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 从设备和主设备组件，如下图所示。

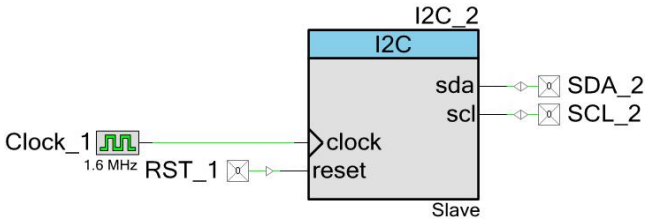
带有引脚的固定功能I²C从设备



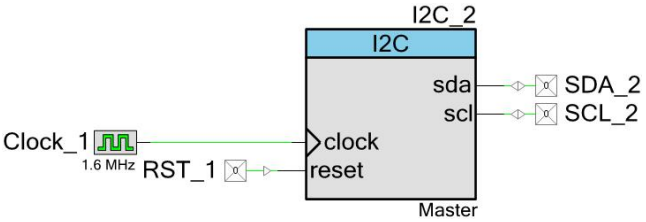
固定功能的I²C主设备引脚



带有时钟和引脚的UDB I²C从设备



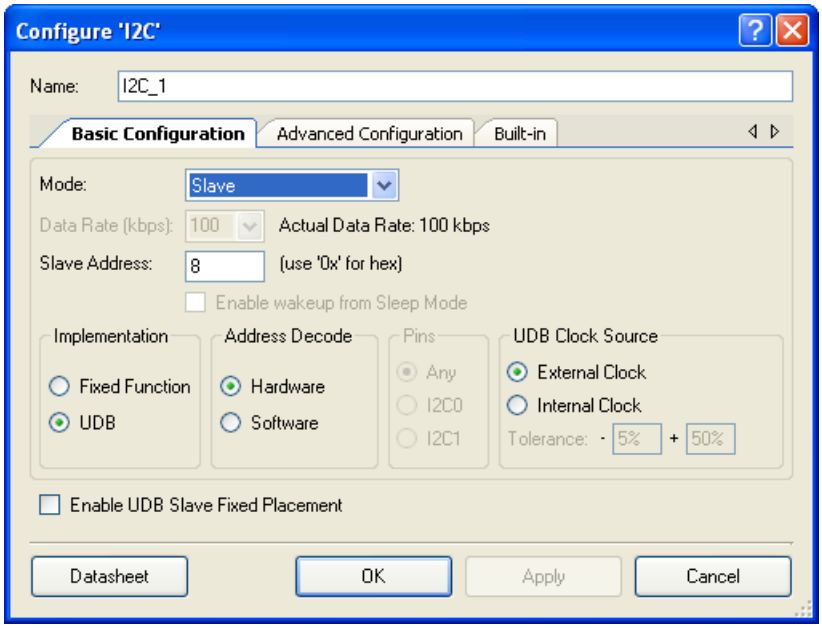
带有时钟和引脚的UDB I²C主设备



组件参数

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

Basic Configuration（基本配置）选项卡



I²C 组件提供了以下参数。

模式

请使用此选项来选择 I²C 模式：从设备、主设备、多主设备或多主从设备。

模式	说明
从设备	仅适用于从设备的操作（默认）。
主设备	仅适用于主设备的操作。
多主设备	支持总线上使用多主设备。
多主从设备	从设备和多主设备的同步操作。

数据速率

此参数用于设置 I²C 数据速率值（可达 1000 kbps）；实际速率可能会基于可用的时钟速度和分频器的分频范围而有所不同。标准数据速率[3]为 50、100（默认值）、400 和 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。

注意： 如果将 **Implementation**（实现）设置为 **UDB**，并且 **Mode** （模式）参数设置为 **Master**、**Multi-Master** 或 **Multi-Master-Slave**，**Data Rate** 超过 400 kbps 的实时主设备速度会根据 **BUS_CLK** 的值、f_{SCL}[4]的上升和下降时间以及组件的位置不同而有所变化。

从设备地址

这是从设备的 I²C 识别地址。如果不选择从设备操作，则会忽略该参数。您可以选择介于 0 至 127（0x00 和 0x7F）之间的从设备地址，默认值为 **8**。该地址为右对齐的 7 位从设备地址，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 位从设备地址，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 并且只需一个从设备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设备地址或地址为 10 位的应用程序而言，则必须使用软件地址检测。**硬件**是默认的设置。如果使能硬件地址解码，该模块会自动不响应不属于自己的地址，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 线保持为低电平，直至 CPU 干预为止。

---

[3] 固定功能实施仅支持适用于 PSoC 5 器件的标准数据速率 50、100 或 400 kbps。而对于速率最高可达 1000 kbps 的数据，则使用基于 UDB 的实现。

[4] 请参考 *I²C 总线规格（修订版* 3，2007 年 6 月）中第 *7.2.1 节：降低 f_{SCL}*。

引脚

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

数值	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]

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。

I²C 组件不负责检查引脚分配是否正确。

使能从睡眠模式唤醒

此选项会在地址匹配时将系统从睡眠模式唤醒。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 和 PSoC 5LP 器件不支持此选项。

必须使能 I²C，以便在从设备地址匹配时可唤醒设备，同时切换至睡眠模式。这一操作可通过调用 I2C_Sleep() API 函数来完成；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和“Power Management APIs”（电源管理 API）章节。

UDB 时钟源

通过该参数您可以选择内部配置的时钟或者外部配置的时钟，以生成数据速率。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** 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。可将时钟容差指定为百分比值。从设备模式的默认范围为**-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，在主设备模式下运行较慢。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。

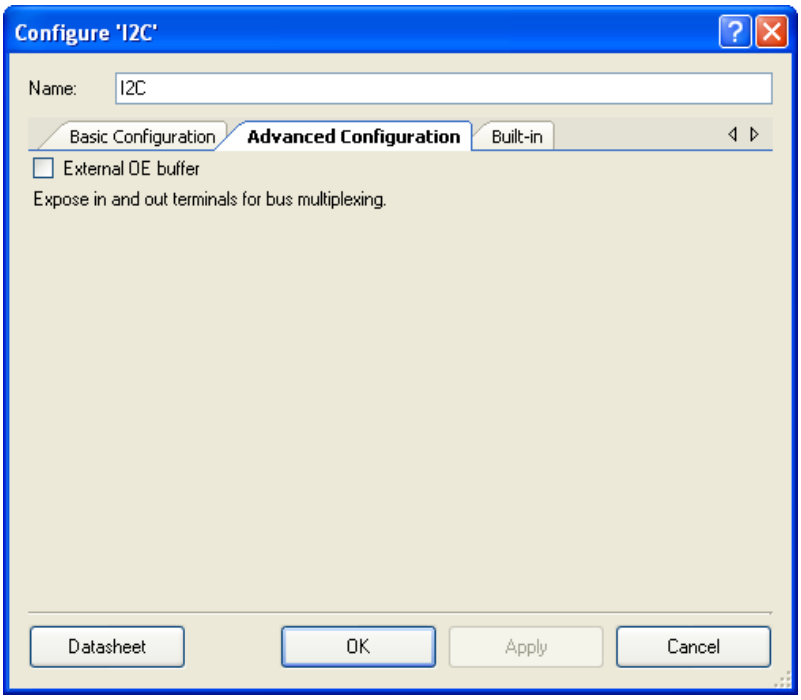
使能 UDB 从设备固定放置

通过此参数您可以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**（从设备）且 **Implementation**（实现）设置为 **UDB** 时才有效。 默认情况下，禁用该选项。

此组件的固定放置状态消除了路由可变性。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。

Advanced Configuration（高级配置）选项卡



External OE Buffer（外部 OE 缓冲区）

此参数允许内部 I²C 总线复用。内部 OE 缓冲被移除，并且双向 scl 和 sda 终端被单独的输入（sda_i 和 scl_i）和输出（sda_o 和 scl_o）替换。

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，用户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。

应用编程接口

您可以使用应用编程接口（API）子程序在运行时配置组件。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了每个函数。

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。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 主设备进行数据定向。当数据从主设备写入到从设备时会发生写入事件。当主设备由从设备读取数据时会发生读取事件。

通用函数

此节包含了 I²C 从设备或主设备操作通用的函数。

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函数可自动执行此操作。
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。
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

正常运行时，不需获得这些变量。

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	I²C主设备的当前状态。
I2C_mstrControl	通过生成或不生成“停止”来控制操作的主设备端。
I2C_mstrRdBufPtr	主设备读取缓冲区的指针。
I2C_mstrRdBufSize	主设备读取缓冲区的大小。
I2C_mstrRdBufIndex	主设备读取缓冲区中的当前索引。
I2C_mstrWrBufPtr	主设备写入缓冲区的指针。
I2C_mstrWrBufSize	主设备写入缓冲区的大小。
I2C_mstrWrBufIndex	主设备写入缓冲区中的当前索引。
I2C_slStatus	I²C从设备的当前状态。
I2C_slAddress	I²C从设备的软件地址。
I2C_slRdBufPtr	从设备读取缓冲区的指针。
I2C_slRdBufSize	从设备读取缓冲区的大小。
I2C_slRdBufIndex	从设备读取缓冲区中的当前索引。
I2C_slWrBufPtr	从设备写入缓冲区的指针。
I2C_slWrBufSize	从设备写入缓冲区的大小。
I2C_slWrBufIndex	从设备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()。

这API使能了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从设备缓冲区，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从设备的行为将如下所示：

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()函数来设置读取缓冲区；

参数： 无

返回值： 无

其他影响： 无

void I2C_Stop(void)

说明： 此函数可禁用I²C硬件和中断。

固定功能（FF）实现（**PSoC 3以及PSoC5 LP**）：如果I²C总线被器件锁定且被设为空闲状态，则它会将其释放。

UDB实现：如果I²C总线被器件锁定且被设为空闲状态，则它会将其释放。

参数： 无

返回值： 无

其他影响： 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

其他影响： 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

其他影响: 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： 这是让组件准备进入睡眠状态的首选API。

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 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。**使能在地址匹配时唤醒：**此API会使能I²C主设备功能（如果在睡眠之前它处于使能状态），并禁用I²C备份调节器。在设置了常规I²C中断处理程序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

其他影响： 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： 此函数根据配置窗口Configure（配置）对话框设置来初始化或恢复器件。您无需调用I2C_Init()，因为I2C_Start() API会调用该函数，该函数是开始执行组件操作的首选方法。

参数： 无

返回值： 无

其他影响： 根据自定义程序“Configure”对话框中的内容设置所有寄存器。

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

其他影响： 无

void I2C_SaveConfig(void)

说明： 此函数会保存I²C组件配置非保留寄存器。

使能在地址匹配时唤醒：此API会使能I²C主设备（如果先前它处于使能状态），并禁用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。

参数： 无

返回值： 无

其他影响： 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。

使能在地址匹配时唤醒：此API会使能I²C主设备功能（如果之前它处于使能状态），并禁用I²C备份调节器。如果I²C作为释放总线并处理到来的I²C数据操作的唤醒源，将生成I²C中断。

禁用在地址匹配时唤醒：请参考主要说明。

参数： 无

返回值： 无

其他影响： 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。

从设备函数

此节列出了用于 I²C 从设备操作的函数。在从设备已使能的情况下可以使用这些函数。

函数	说明
I2C_SlaveStatus()	返回从设备状态标志。
I2C_SlaveClearReadStatus()	返回读取状态标志并清除从设备读取状态标志。
I2C_SlaveClearWriteStatus()	返回从设备写状态并清除写状态标志位。
I2C_SlaveSetAddress()	设置从设备地址，该值的范围介于0至127（0x00 - 0x7F）之间。
I2C_SlaveInitReadBuf()	设置从设备接收数据缓冲区（主设备<-从设备）
I2C_SlaveInitWriteBuf()	设置从设备写缓冲区（主设备 -> 从设备）
I2C_SlaveGetReadBufSize()	返回缓冲区复位后主设备读取的字节数。
I2C_SlaveGetWriteBufSize()	返回缓冲区复位后主设备写入的字节数。
I2C_SlaveClearReadBuf()	将读取缓冲区计数器复位为零。
I2C_SlaveClearWriteBuf()	将写入缓冲区计数器复位为零。

uint8 I2C_SlaveStatus(void)

说明： 此函数返回从设备通信状态。

参数： 无

返回值： uint8：EZ I²C从设备的当前状态。本状态包含读取和写入的状态标志。

每个常量都是一个位字段值。返回值可能包含多个已设置的位，用于指示读取或写入传输的状态。

从设备状态常数	说明
I2C_SSTAT_RD_CMPLT[5]	从设备读取传输完成。在主设备发送 NAK 响应通知已完成读取时设置。
I2C_SSTAT_RD_BUSY	正在进行从设备读取传输操作。在主设备读取以寻址从设备时设置，并在设置 RD_CMPLT 时清除。
I2C_SSTAT_RD_ERR_OVFL	主设备尝试读取超过缓冲区字节数的字节。
I2C_SSTAT_WR_CMPLT[6]	从设备写入传输完成。在收到“停止”条件时设置。
I2C_SSTAT_WR_BUSY	正在进行从设备写入传输操作。在主设备写入以寻址从设备时设置，并在设置WR_CMPLT时清除。
I2C_SSTAT_WR_ERR_OVFL	主设备试图在缓冲区结束时写入内容。接收的字节被从设备否认。

其他影响： 无

uint8 I2C_SlaveClearReadStatus(void)

说明： 此函数可清除读取状态标志并返回其值。

调用该函数不会影响I2C_SSTAT_RD_BUSY标志。

参数： 无

返回值： uint8：EZ I²C从设备的当前读取状态。有关常量，请参见I2C_SlaveStatus()函数。

其他影响： 无

---

[5] 定义由 I2C_SSTAT_RD_CMPT 更改为 I2C_SSTAT_RD_CMPLT，以符合主设备读取完成定义。该组件同时支持两种定义，但是 I2C_SSTAT_RD_CMPT 将取消使用。

[6] 定义由 I2C_SSTAT_WR_CMPT 更改为 I2C_SSTAT_WR_CMPLT，以符合主设备写入完成定义。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。
调用该函数不会影响I2C_SSTAT_WR_BUSY标志。

参数： 无

返回值： uint8：EZ I²C从设备的当前写入状态。有关常量，请参见 I2C_SlaveStatus() 函数。

其他影响： 无

void I2C_SlaveSetAddress(uint8 address)

说明： 此函数可设置I²C从设备地址

参数： uint8 address：主设备的I²C从设备地址。可将此值设置为0至127（0x00 - 0x7F）之间的任意地址。此地址为右对齐的7位从设备地址，它不包括读/写位。

返回值： 无

其他影响： 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：指向主设备读取的数据缓冲区的指针。

uint8 bufSize：I²C主设备可读取的缓冲区的大小。

返回值： 无

其他影响： 如果在总线数据操作期间调用此函数，则可能传输前一个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：指向主设备写入的数据缓冲区的指针。

uint8 bufSize：I²C主设备可写入的缓冲区的大小。

返回值： 无

其他影响： 在总线数据操作期间调用该函数时，可接收前一个缓冲区和当前缓冲位置的数据。

uint8 I2C_SlaveGetReadBufSize(void)

说明： 返回执行I2C_SlaveInitReadBuf()或I2C_SlaveClearReadBuf()函数后I²C主设备读取的字节数。

最大返回值是读取缓冲区的大小。

参数： 无

返回值： uint8：主设备读取的字节。

其他影响： 无

uint8 I2C_SlaveGetWriteBufSize(void)

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

最大返回值为写入缓冲区的大小。

参数： 无

返回值： uint8：主设备写入的字节。

其他影响： 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。主设备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

其他影响： 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。主设备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

其他影响： 无

主设备和多主设备函数

这些函数仅在使能了主设备或多主设备模式时才可用。

函数	说明
I2C_MasterStatus()	返回主设备状态。
I2C_MasterClearStatus()	返回主设备状态并清除状态标志。
I2C_MasterWriteBuf()	将引用数据缓冲区写入到指定的从设备地址内。
I2C_MasterReadBuf()	从指定的从设备地址读取数据，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到缓冲区的起点。
I2C_MasterClearWriteBuf()	将写缓冲区指针复位到缓冲区的起点。

uint8 I2C_MasterStatus(void)

说明： 此函数可返回主设备通信状态。

参数： 无

返回值： uint8：I²C主设备的当前状态。每个常量是一个位字段值。返回值可能包含多个已设置的位，用于指示指示传输的状态和错误条件的生成。

主设备状态常量	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	传输正在进行
I2C_MSTAT_XFER_HALT	传输已停止。I²C总线目前处于等待状态，等待主设备生成“重启”或“停止”条件。
I2C_MSTAT_ERR_SHORT_XFER	错误条件：在传输完所有字节之前写入传输已经结束。
I2C_MSTAT_ERR_ADDR_NAK	错误条件：从设备尚未确认地址。
I2C_MSTAT_ERR_ARB_LOST	错误条件：主设备在与从设备进行通信时仲裁失败。
I2C_MSTAT_ERR_XFER	错误条件：这是表中所示错误状况进行“或”运算后得到的值。 如果错误状况位都已清除，但设置了此位，则传输会因从设备操作而中断。

其他影响： 无

uint8 I2C_MasterClearStatus(void)

说明： 此函数可清除所有状态标志并返回主设备状态。

参数： 无

返回值： uint8：主设备的当前状态。有关常量的信息，请参见I2C_MasterSendStart()函数。

其他影响： 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设备设备中。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位从设备地址（有效范围为0至127）。

uint8 wrData：指向发送数据的缓冲区的指针。

uint8 cnt：要发送的缓冲区的字节数。

uint8 mode：传输模式定义：（1）传输开始时是否生成了开始或重新开始条件，（2）总线生成停止条件之前，传输是否完成或停止。

在传输模式下，可对所有模式常量执行“或”（OR）运算。

模式常量	说明
I2C_MODE_COMPLETE_XFER	执行从开始到停止的完整传输过程。
I2C_MODE_REPEAT_START	发送“重复开始”，并非“开始”。
I2C_MODE_NO_STOP	在没有“停止”操作下执行传输。

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

其他影响： 无

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

说明： 此函数自动读取从设备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位从设备地址（有效范围为0至127）。

uint8 rdData：从设备向其发送数据的缓冲区的指针。

uint8 cnt：要读取的缓冲区的字节数。

uint8 mode：传输模式定义：（1）传输开始时是否生成了开始或重新开始条件，（2）总线生成停止条件之前，传输是否完成或停止。

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“重复开始”，并非“开始”。
I2C_MODE_NO_STOP	在没有“停止”操作下执行传输。

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

其他影响： 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/写位的从设备地址。禁用I²C中断。

参数： uint8 slaveAddress：右对齐的7位从设备地址（有效范围为0至127）。

uint8 R_nW：其值为零时将发送写入指令，其值为非零数值时则发送读取指令。

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	总线上的主设备无效，或者正在进行从设备操作。
I2C_MSTR_ERR_LB_NAK	已否认最后一个字节。
I2C_MSTR_ERR_ARB_LOST	在生成“启动”条件时主设备仲裁失败。（此状态仅在使能多主设备时才有效。）
I2C_MSTR_ERR_ABORT_START_GEN	由于启动从设备操作而终止生成启动条件。（此状态仅在多主设备从设备模式下有效。）

其他影响： 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_MasterSendRestart(uint8 slaveAddress, uint8 R_nW)

说明： 此函数生成重启条件并发送带有读/写位的从设备地址。

参数： uint8 slaveAddress：右对齐的7位从设备地址（有效范围为0至127）。

uint8 R_nW：其值为零时将发送写入指令，其值为非零数值时则发送读取指令。

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

其他影响： 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_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。有关常量的信息，请参见I2C_MasterSendStart()指令。

其他影响： 此函数正在执行阻止操作，在符合下列条件之前不会退出：

Master：此函数不会等待生成“停止”条件。

Multi-Master, Multi-Master-Slave：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： 该函数向从设备发送一个字节。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：发送到从设备的数据字节。

返回值： uint8：错误状态。

模式常量	说明
I2C_MSTR_NO_ERROR	函数完整、无误。
I2C_MSTR_NOT_READY	总线上的主设备无效，或者正在执行从设备操作。
I2C_MSTR_ERR_LB_NAK	已否认最后一个字节。
I2C_MSTR_ERR_ARB_LOST	主设备仲裁失败。（此状态仅在使能多主设备时才有效。）

其他影响： 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： 此函数可读取从设备中的一个字节，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：值为零时发送NAK，值为非零时则发送ACK。

返回值： uint8：从从设备中读取字节

其他影响： 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。

其他影响： 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。

其他影响： 无

void I2C_MasterClearReadBuf (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

其他影响： 无

void I2C_MasterClearWriteBuf (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

其他影响： 无

多主设备从设备函数

多主设备从设备是将从设备函数和多主设备函数合并在一起。

Bootloader 支持

I²C 组件可用作引导加载程序的通信组件。使用以下配置可支持从外部系统到 Bootloader 的通信协议：

- **Mode**：从设备
- **Implementation**：可以是固定功能，也可以是基于 UDB
- **Data Rate**：必须与主机（引导设备）数据速率一致。
- **Slave Address**：必须与所选主机（引导设备）的从设备地址保持一致。
- **Address Match**：硬件是首选项，但不是必须项

更多有关 Bootloader 的信息，请查阅*《系统参考指南》*中“Bootloader 系统”一节的内容。

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	启动I²C组件并使能其中断。
I2C_CyBtldrCommStop	禁用I²C组件并禁用其中断。
I2C_CyBtldrCommReset	将读取和写入I²C缓冲区设置为其初始状态，然后重置从设备状态。
I2C_CyBtldrCommRead	通过此函数，可调用程序读取Bootloader主机中的数据。该函数将处理轮询，以便从主机器件完整接收到数据块。
I2C_CyBtldrCommWrite	允许调用程序将数据写入Bootloader主机。该函数将处理轮询以便让数据块完全发送至主机器件。

void I2C_CyBtldrCommStart(void)

说明： 此函数使能I²C组件，并使其中断。

每个I²C写数据操作都将视为Bootloader的指令。

每个输入I²C读数据操作都将在Bootloader响应已执行的指令前返回0xFF。

参数： 无

返回值： 无

其他影响： 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

其他影响： 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，并将从设备状态复位。

参数： 无

返回值： 无

其他影响： 无

cystatus I2C_CyBtldrCommRead(uint8 pData[], uint16 size, uint16 * count, uint8 timeOut)

说明： 此函数允许调用程序读取Bootloader主机中的数据。该函数将处理轮询，以便接收来自Bootloader主机的完整数据块。

参数： uint8 pData[]：指向要从Bootloader主机读取数据块的存储区的指针

uint16 size：需要读取的字节数

uint16 *count：指向用于写实际读取字节数的变量的指针

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则返回CYRET_SUCCESS，或是返回对该问题描述最准确的值。更多有关信息，请参考*《系统参考指南》*中“返回代码”一节中的内容。

其他影响： 无

cystatus I2C_CyBtldrCommWrite(const uint8 pData[], uint16 size, uint16 * count, uint8 timeOut)

说明： 此函数允许调用程序将数据写入Bootloader主机中。该函数将处理轮询，以便将完整的数据块发送到Bootloader主机。

参数： const uint8 pData[]：指向要写入Bootloader主机的数据块的指针

uint16 size：需要写入的字节数

uint16 *count：指向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则返回CYRET_SUCCESS，或是返回对该问题描述最准确的值。更多有关信息，请参考*《系统参考指南》*中“返回代码”一节。

其他影响： 无

MISRA 合规性

本节介绍了 MISRA-C:2004 合规性和本器件的偏差情况。有两种偏差类型，如下定义：

- 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。*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。

I²C 组件具有以下特定偏差：

MISRA-C: 2004规则	规则类别（必须/建议）	规则说明	偏差说明
10.1	R	在下面情况下，整型表达式的数值不应隐式转换为不同的底层类型。 a）没有转换到相同符号的更宽的整数类型，或 b）表达式是复合的，或 c）表达式不是一个常量，而是函数参数，或 d）表达式不是一个常量，而是返回表达式	库函数memcpy具有一个通用int参数，用于设置要复制的字节数。 将无符号的16位整数作为一个参数，并将其传输到此函数。 该操作不会造成任何负面影响，因为要复制的字节数始终小于256。
11.5	R	不被执行的转换，该转换将清除指针寻址类型中的任何常量或易失性资质。	库函数memcpy具有指向源和目标的void参数的指针。 指向常量数组的指针作为源参数传输出去，并且资质常量被丢失。 函数memcpy实现永远不会改变该源，因此可以安全使用一个恒定的参数。
17.4	R	数组索引是唯一允许的指针运算形式。	该应用程序分配缓冲区，并根据组件提供的指针和大小设置它们。组件使用阵列索引操作访问这些缓冲区。访问缓冲区前，先检查它们的大小。 只要由应用程序提供的缓冲区大小正确，该实现是安全的。
19.7	A	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。 固件采用了固定功能和UDB实现。带有参数的宏用于以灵活的方式支持这两种实现。

该组件具有以下嵌入式组件：时钟。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。

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。要查看特定组件实例，请打开“Component Catalog”中的对话框或原理图中的组件示例。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，请参考《PSoC Creator 帮助》部分中主题为“查找示例项目”的内容。

功能说明

此组件支持 I²C 从设备、主设备、多主设备和多主设备从设备配置。以下各节概述了如何使用从设备、主设备和多主设备组件的信息。

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。该组件将独立于代码之外为所有中断（数据传输）提供服务程序。为此接口分配的存储器缓冲区类似应用程序与 I²C 主设备/从设备之间的简单双端口存储器。

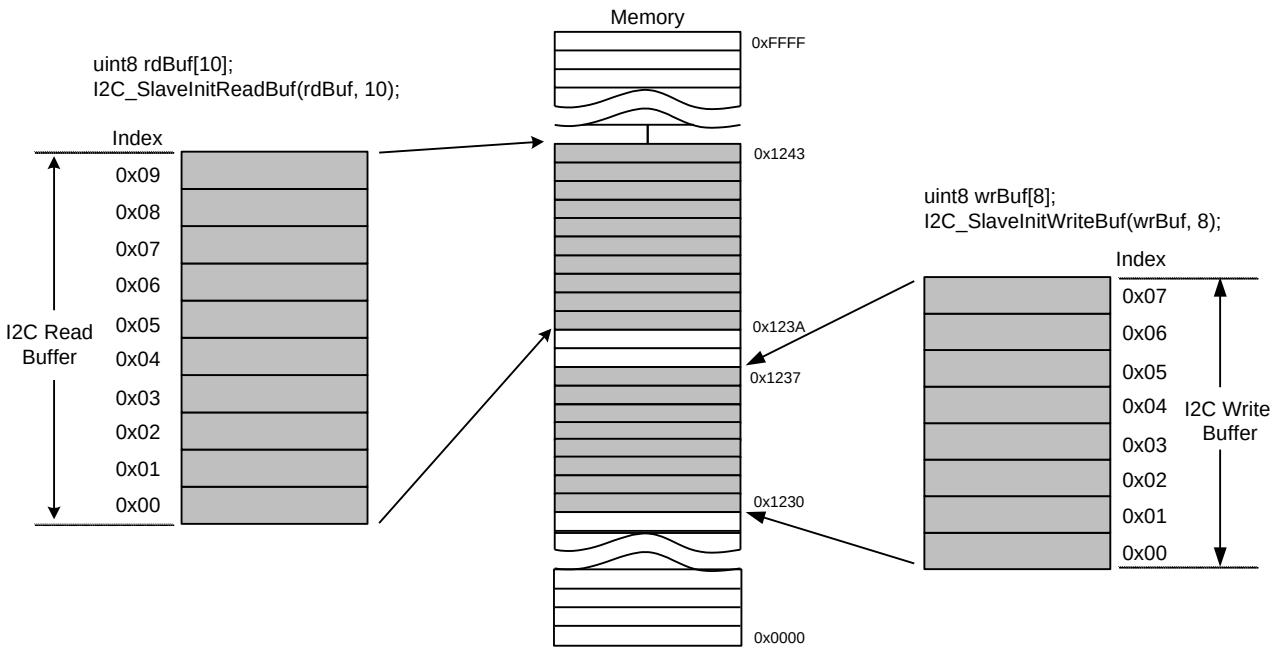
从设备操作

从设备接口由存储器中的两个缓冲区组成。其中一个用于由主设备写入到从设备的数据，另一个则用于主设备从从设备中读取的数据。请牢记，此处的读取和写入均是针对 I²C 主设备的。I²C 从设备的读写缓冲区由下列初始化指令完成设置。这些指令不会分配存储器，而是将阵列指针和大小复制到内部组件变量中。由于组件不会自动生成这些阵列，必须对用于缓冲区的阵列进行实例化。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)
```

使用上述功能为读取和写入缓冲区设置一个指针和字节计数。这些函数的“bufSize”可能不大于实际的阵列大小，但也不应大于“rdBuf”或“wrBuf”指针指向的可用存储器大小。

图 1. 从设备缓冲区结构



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 主设备读取或写入字节时，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。该错误将设置在从设备状态字节中，可以使用 I2C_SlaveStatus() API 来读取该错误。

要将索引重置为阵列的开始位置，请使用以下指令。

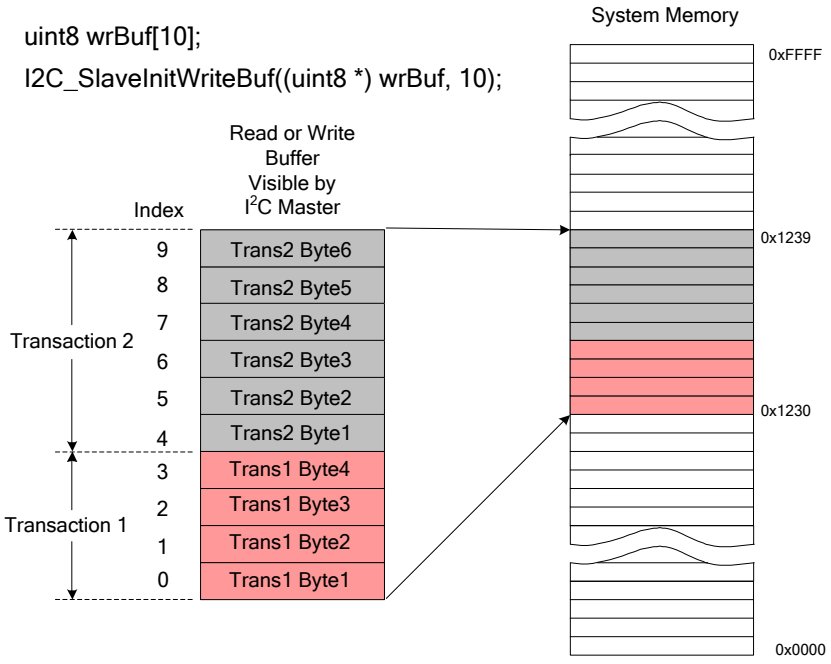
```
void I2C_SlaveClearReadBuf(void)
void I2C_SlaveClearWriteBuf(void)
```

此操作会将索引重置为零。I²C 主设备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 主设备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。在图 2 的示例中，I²C 主设备已经执行了两次写入数据操作。第一次写入了 4 个字节，第二次写入了 6 个字节。由于缓冲区中有一个空间用于存储字节，因此第二次数据操作中的第 6 个字节已经被从设备确认。如果主设备试图在第二次操作中写入第 7 个字节，或在第三次操作时写入其他字节，则在重置缓冲区之前， 每个字节都会被否认并丢弃。

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。请务必确保在缓冲区溢出时不会丢失数据。重置缓冲区索引之前，从设备应先处理缓冲区中的数据。

图 2. 系统存储器



读取和写入缓冲区均有四个状态位用以指示传输完成、正在传输以及缓冲区溢出等状态。当传输开始时，将会设置繁忙标志。传输完成时，将设置传输完成标志并清除繁忙标志。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。读写状态标志如下表所示。

从设备状态常数	数值	说明
I2C_SSTAT_RD_CMPLT	0x01	从设备读取传输已完成
I2C_SSTAT_RD_BUSY	0x02	正在执行从设备读取传输（繁忙）
I2C_SSTAT_RD_OVFL	0x04	主设备尝试读取超过缓冲区限制的字节。
I2C_SSTAT_WR_CMPLT	0x10	从设备写入传输已完成
I2C_SSTAT_WR_BUSY	0x20	正在执行从设备写入传输（繁忙）
I2C_SSTAT_WR_OVFL	0x40	主设备试图在缓冲区结束之后写入内容。

以下示例代码对写入缓冲区初始化后等待传输完成。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，不必复制数据到另一个地方，但可在原始缓冲区中进行处理。您可以通过使用读取函数和常量来替换写入函数和常量，创建一个与写入缓冲区示例一样的读取缓冲区示例。处理数据可能意味着新数据将传入到从设备缓冲区中，而不是从从设备缓冲区中传出。

```
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();
   }
}
```

主设备和多主设备操作

主设备和多主设备[7, 8]操作基本上是相同的，但有两个例外。在多主设备模式下操作时，程序应时刻检查返回的是否为开始进行数据操作的状态。另一个多主设备可能已与另一个从设备进行通信。这时，程序必须保持等待状态，直到通信完成，并且总线进入空闲状态为止。该程序有两种等待方式：生成一个开始进行数据操作，直到返回的状态显示成功完成操作为止；或者检查总线的状态，直到总线进入空闲状态然后生成一个开始进行数据操作为止。如果其他的多主设备生成的开始数据操作更快，则多主设备数据操作可被列入队列。这时，不会返回错误状态，并且会发生多主设备数据操作。只要总线进入空闲状态，将生成该数据操作。

第二个不同之处是在多主设备模式下，可以同时启动两个主设备。如果发生这种情况，其中一个主设备将仲裁失败。

- 多主设备数据自动操作：组件会自动检查这种情况，并在仲裁失败后发出错误响应。当仲裁失败时，多主设备数据操作被认为是完整的（适用的完成状态标志被设置）。
- 手动多主设备数据操作：每个字节传输后，您需要检查返回的状态。

在执行 I²C 主设备操作时可进行如下选择：手动和自动。在自动模式中，将创建缓冲区以保持整个传输。在执行写入操作时，缓冲区预填充要发送的数据。如果自从设备读取数据，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阵列写入到从设备，请使用以下函数。

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

SlaveAddr 变量是一个右对齐的 7 位从设备地址，其值介于 0 至 127 之间。组件 API 将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。参数“cnt”为要传输的字节数。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。

读取操作与写入操作方法几乎一致。使用带有相同常量的相同参数。

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

---

[7] 如果软件在“启动”条件之后立即设置“停止”条件，则在主设备或多主设备模式下，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。

[8] 固定函数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。

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。后续代码段显示对从设备的典型写入操作。

```
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;
   }
}
```

I²C 主设备可手动操作。在此模式中，将使用单个指令来执行写入数据操作的每个部分。

```
status = I2C_MasterSendStart(0x08, I2C_WRITE_XFER_MODE);
if(I2C_MSTR_NO_ERROR == status)   /* 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 */
```

除最后一个字节应该否认外，手动读取数据操作与写入数据操作类似。以下示例显示的是一个典型的手动读取数据操作。

```
status = I2C_MasterSendStart(0x08, I2C_READ_XFER_MODE);
if(I2C_MSTR_NO_ERROR == status)   /* 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 */
```

多主从设备模式操作

在此模式中，可以执行多主设备和从设备操作。该组件可以作为从设备被寻址，但固件还会启动主设备模式传输。在此模式中，当主设备在地址字节期间仲裁失败时，硬件将还原到从设备模式，并且接收的字节会生成从设备地址中断。

有关主设备和从设备操作的示例，请参见从设备操作和主设备和多主设备操作两个章节。

在使能硬件地址匹配时地址字节仲裁受限：如果主设备在地址字节期间仲裁失败，从设备地址中断仅在从设备寻址后生成。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。

开始多主设备从设备传输

使用多主设备从设备时，可以在任意时刻对从设备寻址。当总线处于闲置状态时，多主设备必须花费时间做好充足准备才能生成“启动”条件。在此期间，系统将对从设备寻址，并因此停止多主设备数据操作，而继续执行从设备操作。请务必避免中断从设备操作；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设备数据操作，并正确启动从设备操作。禁用 I²C 中断，可能会发生下列状况：

- 生成“启动”之前总线繁忙（从设备操作正在进行中，或者其他通信正占用总线）。多主设备无法生成“启动”条件。使能 I²C 中断后，从设备操作仍在继续进行。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，多主设备在总线上生成“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。多主设备尝试生成“启动”条件，但在此之前另一多主设备对从设备寻址，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，从设备操作停止于寻址阶段。使能 I²C 中断后，队列中的多主设备数据操作将被取消，从设备操作将继续进行。I2C_MasterWriteBuf()或 I2C_MasterReadBuf()的调用为对此作出响应，而是返回 **I2C_MSTR_NO_ERROR**。多主设备数据操作被取消后，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_ERR_ABORT_START_GEN**。

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

手动多主设备操作假设 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 */
/* Try to generate start condition */
status = I2C_MasterSendStart(0x08, I2C_WRITE_XFER_MODE);
/* Check if start generation completed without errors */
if (I2C_MSTR_NO_ERROR ==status)
{
   /* 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 从设备已使能。选择从设备模式或多主设备从设备模式。
- 选择 I²C 硬件地址检测。
- 将 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]以及 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设备地址相匹配，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

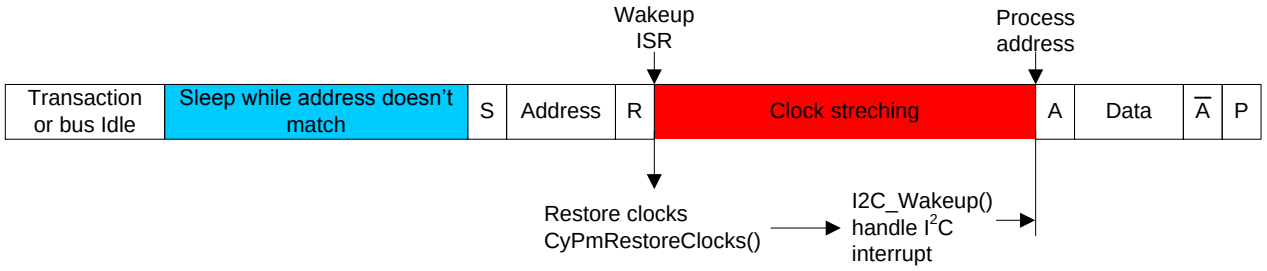
I²C 从设备将在退出睡眠模式时使时钟延展。您必须恢复系统中的所有时钟，然后在唤醒后才能继续 I²C 数据操作。在进入睡眠状态前，需要保持 I²C 中断的有效状态，但必须修改中断处理程序。地址匹配时唤醒触发 I²C 中断，并且 I²C 唤醒标志被设置，以通知该事件。调用 I2C_Wakeup()函数后，中断处理程序将修改为常规 I²C ，另外将根据 I²C 唤醒标志生成中断，以进行输入的数据操作。唤醒后，SCL 线保持低电平，直到调用 I2C_Wake()为止。

代码示例：

```
I2C_Sleep();            /* Prepare I2C to be able wake up from Sleep mode*/
CyPmSaveClocks();       /* Save clocks settings */

CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_I2C);

CyPmRestoreClocks();    /* Restore clocks */
I2C_Wakeup();           /* Retuns I2C for operation in Active mode */
...
```



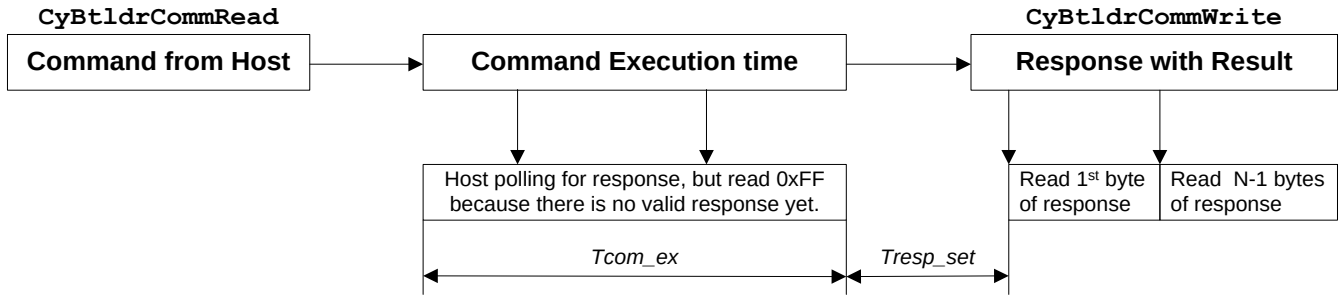
引导加载程序协议与 I²C 通信组件的交互

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。

启动：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。

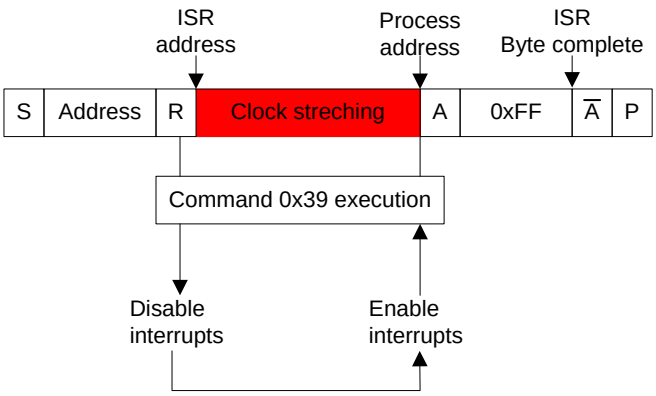
引导加载程序流程：主机将通过一个写入数据操作发出命令，并启动轮询来获取应答。在引导加载程序传递有效应答之前，都将使用 0xFF 来应答 I²C 通信组件。接收 0x01 之后，主机必须再执行一次读取才可获得响应的剩余 N – 1 个字节。在两次读取完成之后，结果将进行合并，形成完整的响应包。



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。

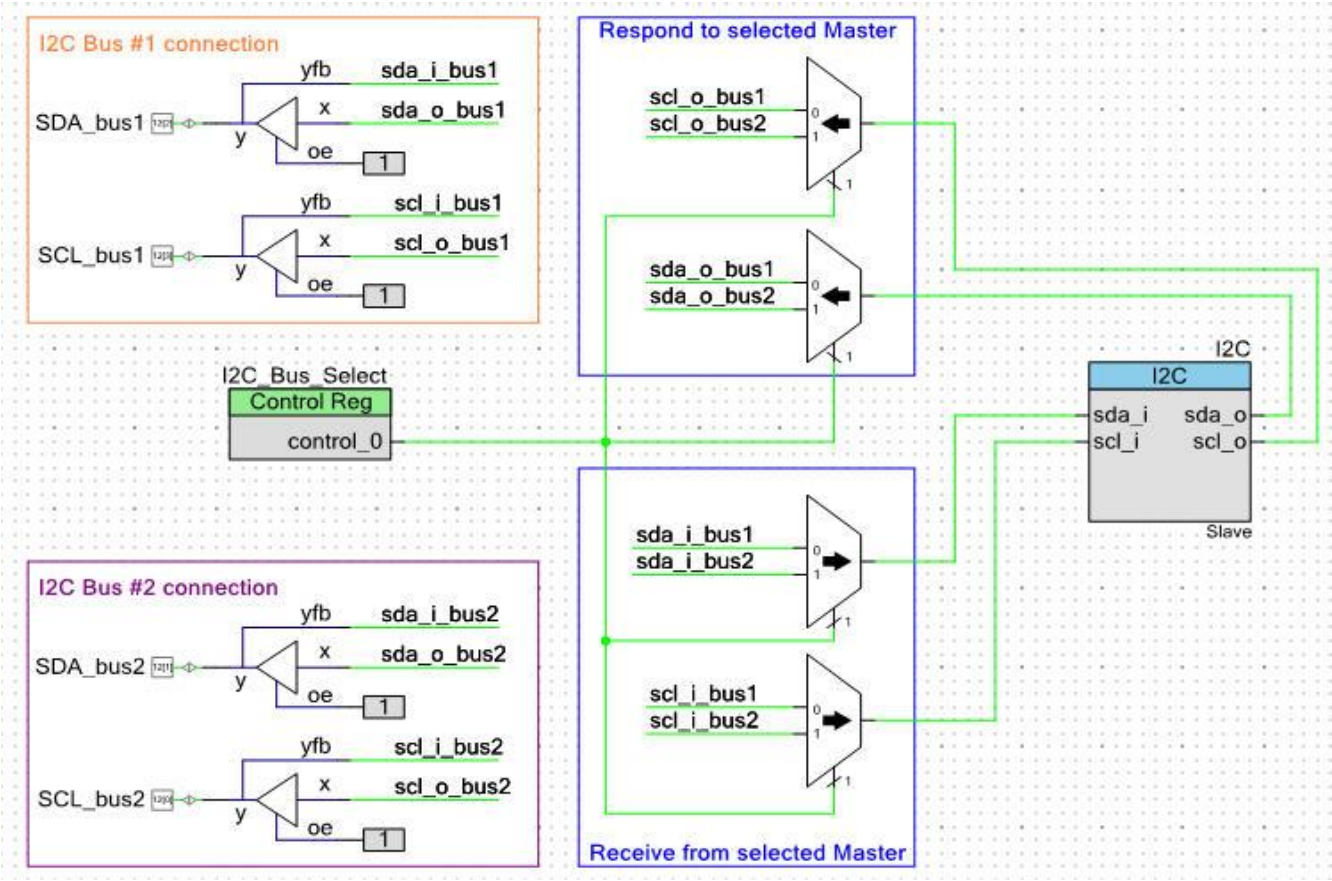
内部 I²C 总线复用

选择 **External OE buffer** 项，以允许内部 I²C 总线复用。内部 OE 缓冲被移除，并且双向 scl 和 sda 终端被单独的输入（sda_i 和 scl_i）和输出（sda_o 和 scl_o）替换。下图通过使用原理图显示了一个 2:1 I²C 从设备复用实现的示例。它可以很容易被扩张以支持某个 N:1 的复用。该方法同样也可以将某个 I²C 主设备连接至多个外部的下游 I²C 总线。

数字复用器用于从不同的 I²C 总线中选出 sda_o 和 scl_o。通过执行“与”运算，将 sda_i 和 scl_i 信号结合起来。三态缓冲区用于使控制 I²C 接口的双向特性控制可用。

要想选择 I²C 总线，必须设置相应的控制寄存器（I2C_Bus_Select）。

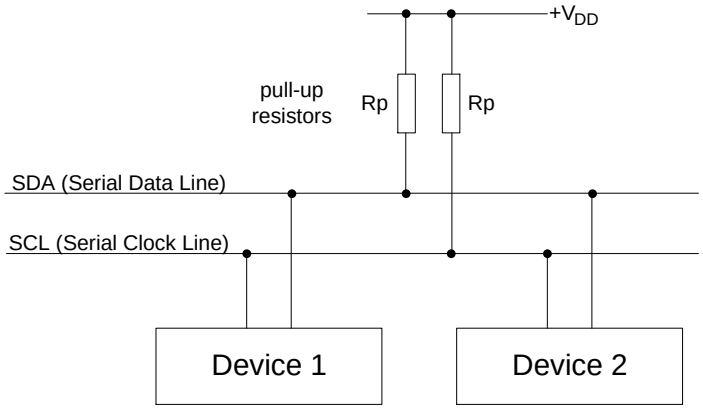
图 3. 内部 I²C 总线复用



外部电气连接

如图 4 所示，I²C 总线上要有外部上拉电阻。上拉电阻（R_P）由供电电压、时钟速度和总线电容确定。将输出阶段的任何器件（主设备或从设备）的最小灌电流设置为不超过 3 mA（在 V_{OLmax} = 0.4 V 的条件下）。这会将 5 V 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率。对于总线电容为 150 pF 的 5 V 系统，上拉电阻不能超过 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 4. 器件与 I²C 总线的连接情况



注意：从赛普拉斯或获得许可的其中一个联营公司处购买 I²C 器件，即可根据 Philips I²C 专利获得一份使用许可，以在符合 Philips 定义的 I²C 标准规范的 I²C 系统中使用这些器件。自 2006 年 10 月 1 日起，Philips 半导体就采用一个新的商标名称 — NXP Semiconductors。

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。您不应该对它进行修改。

从设备操作具备下列用户选项：

- 定制涵盖内容和定义
- 添加的地址比较
- 准备读取缓冲区

主设备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：

主设备或多主设备：

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

从设备：

- 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。

固定功能主设备/从设备寄存器

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 PSoC 3 和 PSoC 5LP 设备中的所有位。

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
数值	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

可以使用固定函数硬件模块中的从设备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）配置从设备器件地址。

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 位从设备地址。

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
数值	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。PSoC 3 和 PSoC 5LP 仅使用位 2。
- en_mstr：用于使能主设备模式。
- en_slave：用于使能从设备模式。

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
数值	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
数值	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设备控制和状态寄存器进行主设备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设备发送器模式中生成“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：表示生成一个有效“启动”条件，同时硬件设备作为总线主设备运行。
- 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 主设备

UDB 寄存器定义来自 I²C 的 Verilog 实现。更多有关这些寄存器的定义，请参考特定模式的 Verilog 实现。

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
数值	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 会使能主设备功能。

I2C_CSR

可以使用固定函数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。对于所有配置为 mode = 1 的位，状态数据寄存在计数器的输入时钟沿处，这些位是粘滞位，并且会在读取状态寄存器时清除。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（多主设备和多主设备从设备模式）。
- 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设备。
- lbr：最后接收的一位。表示最后接收的一位的状态（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 实现中的从设备地址寄存器为硬件比较模式配置从设备设备地址。

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

- slave_addressslave_address：用于为硬件地址比较模式定义 7 位从设备地址

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
数值	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

在主设备执行传输操作时，Go 寄存器会强制传输数据寄存器中的数据。在主设备执行接收操作时，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，无论写入什么值。

UDB 从设备

UDB 寄存器定义来自 I²C 的 Verilog 实现。更多有关这些寄存器的定义，请参考特定模式的 Verilog 实现。

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 会使能从设备功能。

I2C_CSR

可以使用固定函数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。对于所有配置为 mode = 1 的位，状态数据寄存在计数器的输入时钟沿处，这些位是粘滞位，并且会在读取状态寄存器时清除。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。
- address：地址检测。如果设置该值，则表示已接收地址字节。

- lbr：最后接收的一位。表示最后接收的一位的状态（为最后传输的一个字节而收到的 ACK/NAK。Cleared = ACK 和 set = NAK。
- byte_complete：最后一次读取此寄存器以来的发送或接收状态。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 实现中的从设备地址寄存器为硬件比较模式配置从设备设备地址。

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

- slave_addressslave_address：用于为硬件地址比较模式定义 7 位从设备地址

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
数值	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

在主设备执行传输操作时，Go 寄存器会强制传输数据寄存器中的数据。在主设备接收时，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。

使用资源

固定 I²C 模块和一个中断用于固定功能的实现。

该组件的 UDB 版本使用以下资源。

配置	资源类型					
	Datapath 单元	宏单元	状态单元	控制单元	DMA通道	中断
从设备	1	25	1	2	–	1
主设备	2	33	1	1	–	1
多主设备	2	36	1	1	–	1
多主从设备	2	65	1	2	–	1

API 存储器使用

根据编译器、器件、所使用的 API 数量以及组件的配置情况的不同，组件所用的存储器使用情况也不一样。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。对于特定的设计，分析编译器生成的映射文件后可以确定组件占用存储器的大小。

API 存储器使用（FF 实现）

配置	PSoC 3（Keil_PK51）		PSoC 5（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
从设备	1155	20	1155	20	1368	27
主设备	1754	19	1902	22	2018	26
多主设备	1866	19	2026	22	2138	26
多主从设备	2707	32	2719	32	3054	27

API 存储器使用（UDB 实现）

配置	PSoC 3（Keil_PK51）		PSoC 5（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
从设备	927	16	1176	23	1104	23
主设备	1703	16	2034	22	1962	22
多主设备	1857	16	2190	22	2114	22
多主从设备	2640	28	3098	23	2974	23

直流和交流的电气特性

除非另有说明，否则这些规范的适用条件是：–40 °C ≤ T_A ≤ 85 °C 且 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。

直流电的特性（FF 实现）

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

直流电的特性（UDB 实现）

参数	说明			最小值	典型值[9]	最大值	单位[10]
I_{DD}（从设备）	组件电流消耗（从设备）		标准模式	–	200	–	µA
			快速模式	–	290	–	µA
			增强型快速模式	–	335	–	µA
I_{DD}（主设备）	组件电流消耗（主设备）		标准模式	–	210	–	µA
			快速模式	–	305	–	µA
			增强型快速模式	–	465	–	µA
I_{DD}（多主设备）	组件电流消耗（多主设备）		标准模式	–	215	–	µA
			快速模式	–	320	–	µA
			增强型快速模式	–	515	–	µA
I_{DD}（多主从设备）	组件电流消耗（多主从设备）	从设备操作	标准模式	–	200	–	µA
			快速模式	–	290	–	µA
			增强型快速模式	–	335	–	µA
		多主设备操作	标准模式	–	215	–	µA
			快速模式	–	320	–	µA
			增强型快速模式	–	515	–	µA

9 未包括设备 IO 和时钟分配的电流。这些都是在温度为 25 °C 时的值。

10 根据新的组件时钟指定电流消耗。

交流电的特性（FF 实现）

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

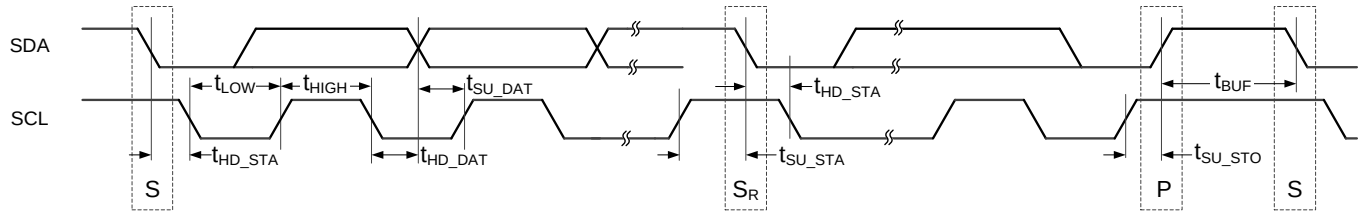
交流电的特性（UDB 实现）

参数	说明	最小值	典型值	最大值	单位
f_{SCL}	SCL时钟频率	– – –	– – –	100 400 1000	kHz
f_{CLOCK}	组件输入时钟频率	–	16 × f_{SCL}	–	kHz
t_{RESET}	复位信号脉冲宽度	–	2	–	t_{CY_clock}[11]
t_{LOW}	SCL时钟的低电平周期	4.7 1.3 0.5	– – –	– – –	µs
t_{HIGH}	SCL时钟的高电平周期	4.0 0.6 0.26	– – –	– – –	µs
t_{HD_STA}	保留时间（重复）启动条件	4.0 0.6 0.26	– – –	– – –	µs
t_{SU_STA}	重复启动条件的建立时间	4.7 0.6 0.26	– – –	– – –	µs
t_{HD_DAT}	数据保持时间	5.0 – –	– – –	– – –	µs
t_{SU_DAT}	数据建立时间	250 100 50	– – –	– – –	ns

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

参数	说明	最小值	典型值	最大值	单位
t_{SU_STO}	停止条件的建立时间	4.0 0.6 0.26	– – –	– – –	µs
t_{BUF}	停止和启动之间的总线空闲时间	4.7 1.3 0.5	– – –	– – –	µs

图 5. 数据转换时序图



组件勘误表

本节列出了组件的已知问题。

赛普拉斯ID	组件版本	问题	解决方案
191257	v3.30	在没有修正PSoC Creator 3.0 SP1中的版本编号时进行更改这组件。更多信息，请参见基础知识文章KBA94159（网页地址：www.cypress.com/go/kba94159）。	解决方案是不必要的。设计不受任何影响。

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	更改说明	更改原因/影响
3.30.b	编辑数据手册并将其添加到组件勘误章节。	文档的组件被更改，但设计不受任何影响。
3.30.a	编辑了数据手册。	更新了“在何种情况下使用I²C组件”章节中框图。 阐明了使用一个中断的固定功能实现。
3.30	添加了MISRA合规性章节。	此组件具有特定的描述偏差。
	当仲裁失败时，修正了主设备手动API的不正确操作。	在某个仲裁失败事件中，主设备手动API返回正确的状态，但没有释放I²C总线。当发生仲裁失败事件时，添加了代码以释放总线。
	在某些地方添加了表示不符合NXP I²C规格的脚注。	文档改进。
	数据规格书的微小编辑和更新。	
3.20	添加了新的特性。移除了内部OE缓冲区，并裸露了输入和输出端子。	通过该特性，I²C总线可以在PSoC内进行复用。
	修改了唤醒过程的控制流量，以避免禁用I²C中断。	PSoC 5LP要求使能一个I²C中断，以便在发生地址匹配时可以唤醒器件。
	移除了Stop（停止）中断，以在开始一个新的数据操作时对其进行处理。	当下一个数据操作开始时，会发生Stop（停止）中断。这样会使中断代码进入一个不正确的状态，并且它不能捕捉停止中断。该问题仅针对从器件设备。
3.10	添加了PSoC 5LP支持。	
	修正了在收到地址字节后错误的SDA行为（线路驱动为低）。	在预期的从设备地址被否认的情况下，当主设备开始执行数据操作时，可能发生错误的停止检测。 该问题仅在带软件地址解码和基于UDB实现的从设备模式下出现。
3.1.a	文档的更改内容描述了有效数据速率如何变化。	对于超过400 kbps的数据速率，有效的时钟速率可能有所不同。
	文档的更改内容描述了主设备和多主设备模式的不同。	在多主设备模式下操作时，需要考虑若干特殊因素，以正确处理同其他主设备的交互。
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设备定义。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。

版本	更改说明	更改原因/影响
	为.cyre 中的所有的API添加CYREENTRANT关键字。	并非所有API都是真正可重入的函数。组件API源文件中的注释表示哪些函数不建议使用。 对于采用了安全方式并且是不可重入的函数，则需要该项变更，这样可以消除编译器警告：通过标志或关键节防止同时调用。
3.0.a	对数据手册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。	在从设备模式下，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。	由于主设备的零数据保持时间，从设备正常运行。
2.10	添加了“多主设备 — 从设备”模式	将“多主设备 — 从设备”功能支持添加到组件中。
	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 Bootloader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指令，则I²C Bootloader通信组件永远将SCL保持低电平。
	向数据手册中添加了特性数据。	
	对数据手册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	
	对数据手册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芯片的修订版

版本	更改说明	更改原因/影响
	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，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。