AN50987

PSoC® 1 I²C 入门

作者：Todd Dust 与 M. Ganesh Raja
相关项目：有
相关器件系列：CY8C21x23、21x34、21x45、22x45、23x33、24x23A、24x33、24x94、27x43、28xxx、29x66
软件版本：PSoC® Designer™5.3

AN50987 介绍的是 I²C 标准概况并说明了 PSoC® 1 器件如何处理 I²C 通信。阅读完本应用笔记后，您应了解 I²C 的工作方式、如何在 PSoC 1 中实现它以及如何为设计选择正确的用户模块。示例项目演示了如何将 PSoC 1 配置为 I²C 主设备/从设备，以便与总线上的其他 I²C 器件进行通信。

目录

简介 2
I²C 的基本知识 2
物理层 2
协议层 2
PSoC 1 I²C 4
硬件 4
I²C 用户模块 5
固件 8
从设备操作 8
主设备操作 9
多主设备从设备操作 10
I²Cm 11
I²C 特别注意事项 12
I²C 寻址 12
上拉电阻 12
I²C 和 ISSP 编程冲突 12
上电时引脚短时脉冲 13
时钟速度 13
时钟延展和中断延迟 14
热插拔 14
短时脉冲过滤 14
I²C 和睡眠模式 14
I²C 和动态重配置 15
I²CHW 用户模块中的动态从设备寻址 15
SCL 线路处于低电平停滞状态 15
总结 16
附录 A 17
硬件寄存器 17
固件需求 17
仲裁 18
附录 B 20
EzI2Cs_ADC_LED_DAC 示例项目 20
I²CHW 从设备示例项目 24
I²CHW 主设备示例项目 25
移植示例项目 26
文档修订记录 27
销售、解决方案以及法律信息 28

简介

内部集成电路（IIC 或 I²C）是由飞利浦半导体公司（现为 NXP）开发的常用芯片间串行通信标准。I²C 为集成电路提供了一种在同一印刷电路板（PCB）上进行通信的简单方法。I²C 由一个简单的物理层组成，仅需要两个引脚和极少的外部组件。与 SPI 等通讯标准相比，I²C 的一个优点就是内置通信协议，使器件之间轻松进行无差错通信。

赛普拉斯 PSoC 1 提供了在设计中实现 I²C 的多种选择。这些选择以用户模块（UM）的形式出现在 PSoC Designer™ 集成开发环境（IDE）中。本应用笔记先介绍了 I²C 的基本知识，旨在帮您了解 PSoC 1 器件如何处理 I²C。如果您对 I²C 基础已经有所了解，建议您直接阅读 PSoC 1 I2C。如果您需要帮助解决 I²C 设计中的故障难题，请直接阅读 I2C 特别注意事项部分的内容。

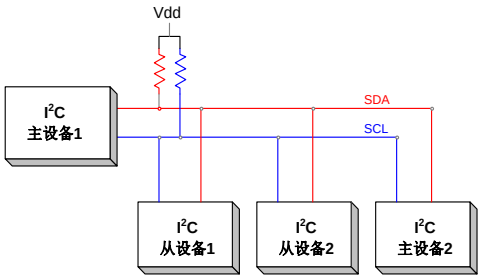
本应用笔记假定您已熟悉 PSoC 1 器件和 PSoC Designer IDE。

I²C 的基本知识

物理层

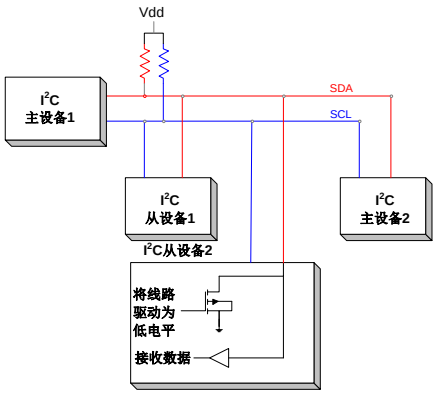
图 1 显示了 I²C 器件与 I²C 总线的连接方式。I²C 总线由两个物理线路组成，分别是串行数据（SDA）和串行时钟（SCL）。总线上的所有器件都必须连接至这两条物理线路。唯一需要的外部硬件就是 SDA 和 SCL 上配备的 V_{DD}（高轨）上拉电阻。欲了解更多有关上拉电阻的信息，请参见上拉电阻部分。

图 1. 典型 I²C 总线



SDA 和 SCL 都是双向的；数据可以在主设备和从设备之间来回传送。SDA 和 SCL 具有开漏驱动模式并被驱动为低电平。器件能够将线路驱动至逻辑低电平或高阻抗状态，也就是说，器件不能将这些线路驱动至高电平。这种配置可防止总线上发生电源和地的短路；请参见图 2。SDA 和 SCL 上的上拉电阻产生高逻辑电平。

图 2. 开漏驱动低电平引脚配置



I²C 总线上的器件之间的关系是主设备 - 从设备。主设备启动总线上的所有数据传输，并生成所有时钟信号。每个从设备都有唯一的一个地址；在开始传输数据之前，主设备必须先寻址一个特定的从设备并接收一条确认信息。同一总线上可以同时存在多个主设备和多个从设备。I²C 总线可在不同频率上运行；典型的频率是 100 kHz 和 400 kHz。I²C 规范还允许 1 MHz 和 3.4 MHz 的高频率。

根据各个器件的具体情况，I²C 总线的工作电压有所不同。要确定两个器件是否可以成功通信，请检查各自的数据手册，确保二者的逻辑电平是兼容的。

如果两个器件的电压和逻辑电平不兼容，可根据 I²C 规范中提供的解决方案，对不同电压电平的总线进行桥接。

协议层

要掌握本数字通信技术，理解 I²C 的协议层就是下一个主要步骤。

每个 I²C 数据操作都由以下元素组成：启动（或重复启动）、寻址、数据和停止。

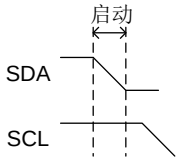
启动或重复启动

主设备控制时钟线路，因此它启动总线上的所有通信。要控制总线并启动数据传输，主设备会首先发送一个启动条件；请参见图 3。

启动条件向总线上的所有器件发送信号，通知主设备已经控制了总线，并将发送一个地址。此时，总线即被视为繁忙，其他主设备必须等待停止条件使总线恢复空闲状态后，才能启动数据传输。

启动 = SCL 为高电平时，SDA 从高电平转变为低电平

图 3. 启动条件



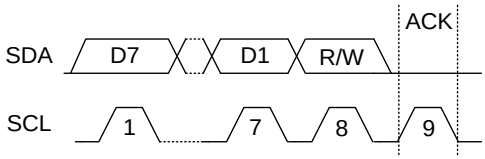
重复启动条件在物理层面上与启动条件相同。它表示主设备已经维持对总线的控制而且该总线没有处于闲置状态。

地址

主设备发出启动条件后，发送的第一个数据字节就是地址。每个从设备都有唯一的地址。大多数 I²C 地址的长度都为七个数据位。读/写（R/W）位凑满 8 位字节并指出接下来数据传输的通信方向；请参考图 4。“读”代表主设备希望从设备中读取数据。“写”表明主设备希望向从设备中写入数据。所有地址和数据字节都优先发送最高有效位（MSB）。

例如，如果一个 7 位地址为 0x20，则代表“读”的完整 8 位字节为 0x40；代表“写”的完整 8 位字节为 0x41。

图 4. 7 位地址



ACK/NAK

主设备发送地址后，等待从设备发出确认（ACK）信息。ACK 是一个附加的状态位，位于每个数据字节的末尾，因此，所有 I²C 数据传输的长度都是九位。

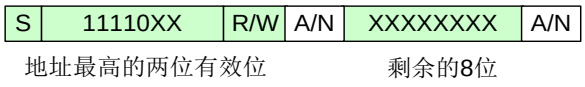
总线上所有从设备都要读取接收到的地址，并将其与自己的内部地址进行比较。如果地址相匹配，从设备必须在第九个时钟循环发出 ACK 信息。如果地址不相匹配，从设备将不响应（发出 NAK 信息）。

- ACK = 0 （低逻辑电平）
- NAK = 1 （高逻辑电平）

10 位地址

I²C 规范还允许使用 10 位地址；请参考图 5。要使用此类地址，必须向从设备发送两个地址字节。第一个字节包含的内容有：序列 11110、地址的两个最高有效位、R/W 位；从设备必须确认该字节。然后，主设备发送剩余的八位地址。最后，从设备发送 ACK/NAK。

图 5. 10 位地址

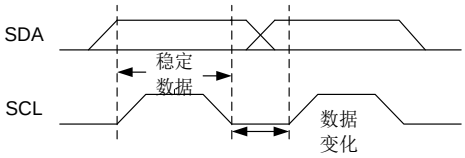


数据

主设备发送地址而且从设备确认后，每次可以传送八位数据。主设备和从设备都可以传送和接收，因此它们一起拥有 SDA 线的控制权。

SDA 线上的数据在 SCL 的上升沿前必须保持稳定状态。仅当 SCL 为低电平时，才能更改 SDA 上的数据；请参见图 6。

图 6. SDA 数据更改



主设备执行写入操作时，在 SDA 上写出八位的数据，并在 SCL 上提供八个时钟周期。主设备发送八位数据后，从设备负责在第九个时钟周期发送 ACK 或者 NAK 信号；请参考图 7。

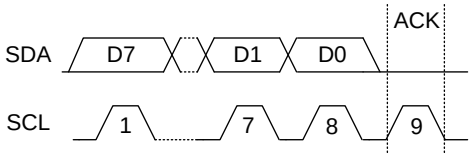
- ACK = 从设备有足够空间容纳更多数据
- NAK = 从设备无法容纳更多数据

交替条件表明主设备希望在从设备中读取数据。在这种情况下，主设备会提供八个时钟周期，但是从设备在 SDA 上传输八个数据位。传输八个数据位后，主设备必须发送 ACK 或 NAK 信号。

- ACK = 主设备希望读取更多数据
- NAK = 主设备读取完毕

注意： 从设备无法通知主设备没有要发送的数据。

图 7. 八个数据位后面是一个 ACK 位

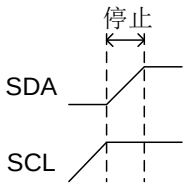


停止

发送所有字节后，主设备要发送一个停止条件。停止条件表明当前数据传输已完成，总线处于闲置状态；请参见图 8。

停止 = SCL 为高电平时，SDA 从低电平转变至高电平

图 8. 停止条件



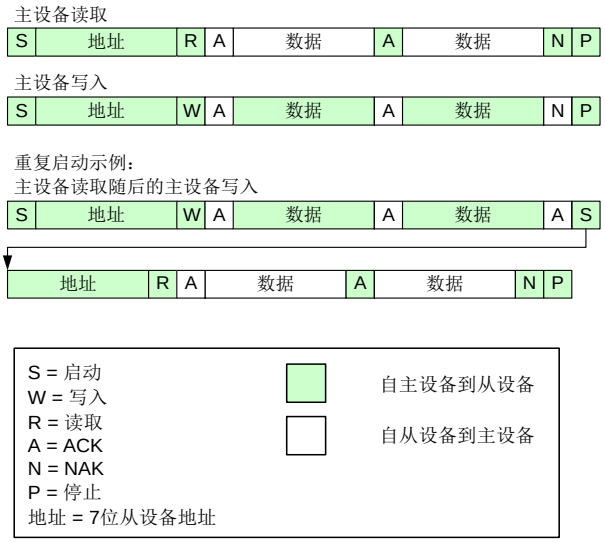
除了发送停止条件之外，主设备还可以发送重复启动条件。重复启动条件在物理层面上与启动条件相同。当主设备想要通知从设备开始新的数据传输或者更改数据流的方向时，主设备会发送重复启动条件，这样可以保持对总线的控制。

当主设备想要向特定的从设备写入数据，然后反过来从该从设备中读取数据时，需要使用重复启动条件。通过使用重复启动条件，主设备可以维持对总线的控制。如果使用停止条件，则总线可能由其他主设备控制。

全部放在一起

图 9 显示的是包含两个字节的主设备读取和包含两个字节的主设备写入的完整数据传输。另外它还显示一个先写入两个字节然后读取两个字节的重复启动示例。

图 9. 完成数据传输

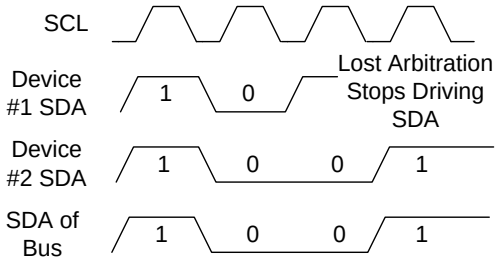


仲裁

I²C 协议允许多个主设备在同一总线上进行通信。两个主设备可以同时进行通信。为了防止数据丢失，每个 I²C 主设备都必须查看 I²C 总线，确保总线上的数据为自己所提供。如果数据不匹配，仲裁失败的 I²C 主设备必须停止驱动 SDA 线，并等待总线闲置后再重新尝试发送数据。

如图 10 所示，当一个主设备将 SDA 保持为高电平，而另一个主设备尝试将 SDA 驱动至低电平时，前者仲裁失败，必须等待。

图 10. 仲裁



更多有关 I²C 及其协议的信息，请参考飞利浦（NXP）的 I²C 规范。

PSoC 1 I²C

在 PSoC 1 中，专用 I²C 硬件模块处理 I²C 数据传输，帮助 CPU 解除处理流程产生的负担，并使 CPU 可以处理重要的实时任务。在大多数 PSoC 1 器件中，该模块的基本结构相似；请参见表 1，了解不同器件系列之间的差异。

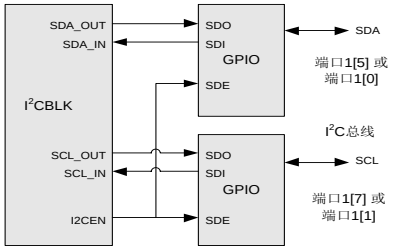
表 1. I²C 硬件模块差异

器件	主设备	从设备	硬件地址匹配	两个 I²C 模块
20x34	无	**有**	无	无
20xx6A	无	**有**	**有**	无
21x23	**有**	**有**	无	无
21x34	**有**	**有**	无	无
22xxx/21x45	**有**	**有**	无	无
23x33	**有**	**有**	无	无
24x23A	**有**	**有**	无	无
24x94	**有**	**有**	无	无
27x43	**有**	**有**	无	无
28xxx	**有**	**有**	**有**	**有**
29x66	**有**	**有**	无	无

硬件

如图 11 所示，硬件模块允许端口 1.5、1.0、1.7 和 1.1 连接到 I²C 总线上。赛普拉斯建议不要将端口 1.1 和 1.0 连接到 I²C 上，因为这些线路为编程接口。

图 11. I²C 硬件模块



硬件模块是一个简单的模块，能够处理 I²C 数据传输的所有状态和时序需求。该模块处于主设备模式时，会生成 I²C 时钟。另外它也将 I²C 数据输入和输出 PSoC 1 并记录 I²C 数据传输的状态和错误。

中断和数据传输队列

模块每次只能接收或传送一个字节的数据。另外它还会在每个字节边界上生成一个中断。CPU 必须响应中断的要求，为模块提供更多的数据，或者读取模块接收到的数据。CPU 不需要立即响应模块的需求，因为模块会将 SCL 线保持在低电平状态，直到 CPU 进行释放为止；这一过程就是时钟延展。欲了解更多信息，请参见时钟延展和中断延迟部分的内容。

模块每次仅能够将一个数据传输加入队列。不能在模块中排列多个启动。因此，用户代码必须保证当前 I²C 数据传输完成后，再启动新的数据传输。

在多主设备环境中，模块会自动探测并报告仲裁条件。出现仲裁事件时，模块会报告 CPU 仲裁失败。用户代码应查看仲裁是否失败。如果是，则该代码将重新尝试传输。

如果发生以下任意情况，I²C 硬件模块将生成中断：总线错误、停止或字节完成。

如果总线上启动或停止条件的位置出现错误，则发生总线错误。发生总线错误后，总线上的所有器件必须停止当前的传输并回到空闲状态。

如果它被使能，那么只要总线上出现停止条件，便会发生停止中断。

根据数据流的不同方向，会在不同时间点触发字节完成中断；请参见表 2。该表格适用于地址和数据传输。

表 2. 字节完成中断

模式	主设备	从设备
发送器	8 位数据及 ACK/NAK 信号后	8 位数据后
接收器	8 位数据后	8 位数据及 ACK/NAK 信号后

欲了解 I²C 硬件模块功能的更多信息，请参见附录 A 及 PSoC 技术参考手册中 I²C 部分的内容。

CY8C28xxx 中的硬件模块

CY8C28xxx 器件系列提供两个单独 I²C 硬件模块，允许硬件同时能与多个 I²C 总线相连。此外，每个模块还提供硬件地址匹配功能。仅当地址匹配时，硬件才会中断 CPU 活动。然而，不能将 PSoC 1 从睡眠状态中唤醒。中断地址匹配后，硬件会根据表 2 中列出的条件中断 CPU 活动。

CY8C28xxx 器件系列中的每个硬件模块允许其他 I²C 引脚连接至端口 1.2 和 1.6 或端口 3.0 和 3.2。

拥有两个 I²C 的硬件模块可以实现很多强大的应用。表 3 列出了可以实现的某些应用。

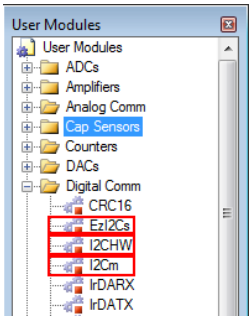
表 3. 双 I²C 配置和常见使用情况

配置	常见用法
两个 I²C 从设备	多个 I²C 总线系统 I²C 共享存储器件
一个 I²C 从设备，一个 I²C 主设备或多主设备	I²C 总线开关 I²C 热插拔控制器 I²C 缓冲区 调试
两个 I²C 主设备或多主设备	增加带宽

I²C 用户模块

赛普拉斯提供一系列预配置和预编码的 I²C 用户模块（UM），这些模块位于 UM 目录中的 PSoC Designer 内；请参考图 12。

图 12. PSoC Designer 中的 I²C 用户模块



如果将这些 UM 应用于设计中，您在 PSoC Designer 的芯片视图中无法看到它们。这是因为这些用户模块使用专用的硬件模块，或者为软件主设备。仅在 28xxx 系列中才使用数字模块；这时，芯片视图显示两个 I²C 硬件模块，因此您就能够了解哪些模块已使用，哪些未使用；请参见图 13。

图 13. 28xxx 中的两个 I²C 模块



I²C 用户模块包含了 EzI2C 和 I²CHW 的，它提供的抽象层次高于 I²C 硬件模块。本章节内容简要介绍了各种用户模块，以及设计中何时需要应用该模块。欲了解更多信息，请参见特殊用户模块的数据手册。

EzI2C

EzI2C 仅作为从设备运行；没有主设备版本。如果在一个设计中需要主设备操作，那么就必须使用 I²CHW 或 I²Cm。

顾名思义，EzI2C 是一种易于实现的 I²C 从设备接口。它是 I²C 硬件模块之上的固件层。它能够实现附录 A 中硬件模块描述内提到的所有固件要求。

EzI2C 用户模块十分特别，只需要用户了解很少的 I²C 总线工作知识即可。该模块允许用户在用户代码中建立一个数据结构，并将该结构公开给 I²C 主设备。所有 I²C 数据传输都通过中断在后台发生。用户模块启动后，您无需担心 I²C 的功能出现问题。应用代码需要完成的就是使用主设备要读取的数据更新数据结构，并检查和处理主设备写入的数据。

当 PSoC 需要持续向主设备传送数据且用户希望写入最少的 I²C 代码时，最适宜使用该用户模块。例如，该用户模块在 CapSense®应用中十分常用。PSoC 读取 CapSense 按键的状态，主设备连续与 PSoC 进行通信，旨在查看按键是否被按下。在这种情况下，将 CapSense 变量公开给 EzI2C 用户模块，主设备即可轻松读取 CapSense 按键的状态。

EzI2C 根据基于 I²C 的存储器（如 EEPROM）进行建模。它使用子地址寻址方式，在公开的 I²C 数据结构的具体数据位置内写入或读取数据。例如，要考虑下面所示的 I²C 数据结构：

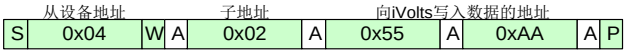
```
struct MyI2C_Regs {
    BYTE bStat;
    BYTE bCmd;
    int  iVolts;
    char cStr[6];
} MyI2C_Regs
```

上述数据将通过以下存储器映射公开给主设备。

0x00	MyI2C_Regs.bStat
0x01	MyI2C_Regs.bCmd
0x02	MyI2C_Regs.iVolts(MSB)
0x03	MyI2C_Regs.iVolts(LSB)
0x04	MyI2C_Regs.cStr[0]
0x05	MyI2C_Regs.cStr[1]
0x06	MyI2C_Regs.cStr[2]
0x07	MyI2C_Regs.cStr[3]
0x08	MyI2C_Regs.cStr[4]
0x09	MyI2C_Regs.cStr[5]

如果主设备希望写入 iVolts，则应先发送从设备地址，然后发送指示架构中 iVolts 偏移的子地址（0x02）。要计算该偏移，只需在计数 iVolts 之前先计数字节数即可；在此示例中，子地址为‘2’。还可以扩展该示例，如果子地址为‘0’，则会写入 bStat，子地址为‘1’则会写入 bCmd，子地址为‘4’则会写入 cStr 阵列的第一个元素。图 14 显示了主设备将数据写入 EzI2Cs 用户模块中 iVolts 的示例（该用户模块中具有 0x04 的从设备地址）。

图 14. EzI2Cs 示例：写入 iVolts



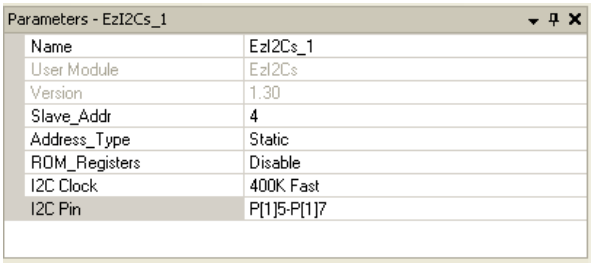
如果主设备希望读取 iVolts，则需要先 PSoC 从设备进行寻址，并写入子地址‘2’。然后再次对从设备进行寻址，并读取两个字节。后续每次读取都从 iVolts 开始，直到写入新的子地址为止。图 15 显示了主设备读取 iVolts 中数据的示例。

图 15. EzI2Cs 示例：读取 iVolts



现在请看一下如何在项目中实现 EzI2Cs。图 16 介绍了该项目的参数。

图 16. EzI2Cs UM 参数



Slave_Addr：该参数用于设置 EzI2Cs 从设备的地址。如果 ROM_Registers 参数被禁用，那么 Slave_Addr 即为 7 位地址，其取值范围为 0 到 127。如果使能了 ROM_Registers 参数，则 Slave_Addr 为 6 位地址，取值范围为 0 到 63。

Address_Type：如果地址类型被设置为静态，则 EzI2Cs 从设备的地址固定为 Slave_Addr 参数所设定的值。如果地址类型被设为动态，则可以通过 EzI2Cs_SetAddr 函数进行修改固件中的地址。在应用中，如果总线上有一个以上的 PSoC 1 EzI2Cs 从设备，则该参数很有帮助，可以通过配置 GPIO 引脚设置从设备的地址。

ROM_Registers：使能 ROM_Registers 参数，则可以向 I²C 主设备公开存储在 ROM 阵列中的数据。使能该参数后，EzI2Cs 将向主设备公开两个地址。如果主设备想要访问 ROM 储存器空间，那么需要使用一个第七位已被设置的 7 位地址。如果要访问 RAM 储存器空间，则使用一个第七位已被清空的 7 位地址。例如，如果 Slave_Addr 被设为 0x04，则主设备将使用 0x44 的地址对从设备进行寻址，以便访问 ROM 寄存器，并使用 0x04 的地址对从设备进行寻址以访问 RAM 寄存器。这就是使能 ROM 寄存器时，Slave_Addr 参数应设为 6 位地址的原因。

I2C_Clock：使用该参数设置从设备能够运行的最高速度。请注意，该最高速度是基于 24 MHz 的 SYSCLK 得出的。如果在全局资源中，将 SYSCLK 设为 6 MHz 或 12 MHz（已使能 SLIMO），最大时钟速度也将以相同的系数减少。例如，如果 I2C_Clock 参数被设为 400 kHz，SysClk 被设为 6 MHz，则从设备的实际速度将是 100 kHz。请参见时钟速度部分的内容，了解更多详细信息。

I2C_Pins：该参数用于选择 I²C 时钟和数据使用的引脚。

UM 参数配置完成后，请按照以下步骤在固件中运行 EzI2Cs。

1. 创建数据结构以公开给主设备。例如：

```
structMyI2C_Regs {
    BYTE bStat;
    BYTE bCmd;
int  iVolts;
char cStr[6];
} MyI2C_Regs
```

2. 通过调用 EzI2Cs_Start 函数启动 EzI2Cs。
3. 通过调用 EzI2Cs_EnableInt 函数启动中断。
4. 使用 EzI2Cs_RamSetBuffer 函数将数据结构公开给主设备。

```
EzI2Cs_SetRamBuffer(sizeof(MyI2C_Regs),
2, (char*)&MyI2C_Regs);
```

第一个参数用于设置缓冲区大小。第二个参数设置写入边界，第三个参数启动指向数据结构的指针。

EzI2Cs 允许通过 SetRamBuffer 函数定义数据结构上的读/写权限。例如，如步骤 1 中所显示的代码，该应用将为 bStat 和 bCmd 变量设置读/写权限，并将写入边界设置为‘2’，使其他变量成为只读变量。在这种情况下，如果主设备尝试向只读变量写入数据，则从设备会生成 NAK 消息，忽略主设备写入的数据。

5. 在主循环中，不断刷新主设备将读取的处理数据结构，并不断从主设备寻找新的数据。

使用 EzI2Cs UM 时要考虑的一个要点就是数据连贯性。在示例中，iVolts 是一个双字节变量。很可能在 CPU 开始更新该变量，但仅写入一个字节后，I²C 主设备就读取了两个字节。因此，主设备将读取到错误的数据。例如，假设变量 iVolts 的值为 0x01FF。CPU 需要将其更新为 0x0200。CPU 先将 0x02 写入到最高有效位（MSB）内，然后再将 0x00 写入到最低有效位（LSB）内。如果 I²C 仅在 CPU 写入到最高有效位内但尚未完成向最低有效位的写入操作时读取数据，则读取到的数据将是 0x2FF。

要避免发生这种情况，最好在主设备和从设备间使用标记或信号来指示数据可被读取或写入的时间。EzI2Cs 数据手册详细介绍了保证数据连贯性的其他方法。

此应用笔记包含一个简单项目，用于演示如何使用 EzI2Cs 用户模块。有关如何使用该项目的更多信息，请参见附录 B。

要详细了解该用户模块的工作方法以及使用方法，请参见 EzI2Cs 数据手册。

EzI2Cs 总结和重要说明

EzI2Cs 是一个仅作为从设备且易于实现的用户模块。

EzI2Cs 向具有明确读/写权限的主设备公开寄存器结构。

使能 ROM 寄存器后，从设备会有两个地址，一个用于 RAM 空间（已清除第七位），一个用于 ROM 空间（已设置第七位）。因此，使能 ROM 寄存器后，从设备地址仅限于 6 位。

在 EzI2Cs 中处理多字节值时，请务必在主设备和从设备之间使用标记或信号，以保证数据连贯性。

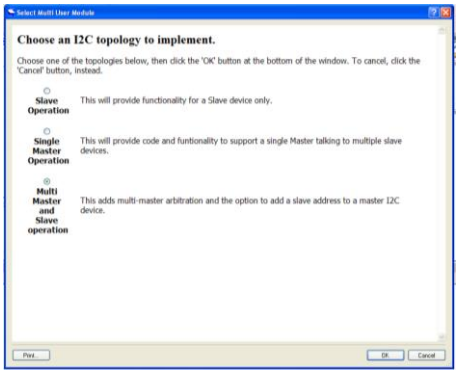
I²CHW

该用户模块是 I²C 硬件模块之上的固件层，用于实现附录 A 中所指定的全部固件任务。您可以灵活使用该用户模块，将其作为从设备、主设备或多主从设备。

与 EzI2Cs 不同，I²CHW 用户模块需要更多代码交互。必须选中状态位以查看 I²C 数据传输操作是否发生，数据传输完成后，必须重新初始化缓冲器，且清除状态位。此外，还要查看固件，检查数据传输过程中是否出现错误条件。

下面将会介绍在项目中如何使用 I²CHW。双击打开 I²CHW 拓扑结构选择窗口。请查看 图 17。

图 17. I²CHW 拓扑结构窗口



Slave Operation（从设备操作）：如需执行从设备操作，请选择此选项。

Single Master Operation（单一主设备操作）：如需在单一主设备环境中执行简单的主设备操作，请选择此选项。

主设备操作中有三个参数可用：Read_Buffer_Type、I2C_Clock 和 I2C_Pin。这三个参数的配置与下面“从设备操作”部分中所述相同。

Multimaster and Slave Operation（多主设备和从设备操作）：如需在多主设备环境中将 I²CHW 用作为主设备和从设备，请选择此选项。

根据您所选择的选项，相应的用户模块窗口将会弹出。

Slave_Addr：这是 7 位从设备地址。与 EzI2Cs 不同，I²CHW 不会对 ROM 缓冲区进行动态寻址，亦不会为其提供单独地址。在多主设备和从设备操作中，Slave_Addr 是被总线上另一主设备作为从设备进行寻址的器件的地址。

Read_Buffer_Type：选中“RAM only”（仅 RAM）时，应用程序可设置仅读取 RAM 中的缓冲区。当您选中“RAM or FLASH”（RAM 或闪存）时，应用程序可向主设备公开 RAM 缓冲区或闪存缓冲区。

Communication Service Type（通信服务类型）：选中“Interrupt”（中断）时，系统将在中断期间自动处理所有 I²C 数据传输操作。当您选中“Polled”（轮询）时，系统仅在前台应用程序调用 I2CHW_Poll 函数时才会处理 I²C 数据传输操作。硬件会持续延展 I²C 时钟，直至应用程序调用 I2CHW_Poll 为止。赛普拉斯建议您选择“Interrupt”（中断）选项。

I2C_Clock：此参数用于选择从设备的速度。 如 EzI2Cs 从设备的时钟参数部分所述，此速度基于 24 MHz 的 SYSCLK。

I2C_Pins：该参数用于选择 I²C 使用的引脚。除 CY8C28xxx 外，所有器件均可选用 P1[0]-P1[1]和 P1[5]-P1[7]。CY8C28xxx 还可选用 P1[2]-P1[6]和 P3[0]-P3[2]。

固件

配置完拓扑结构和参数后，请遵循下列步骤在固件中运行 I²CHW。

从设备操作

1. 在 RAM（或闪存）中声明一个读取缓冲区，以供主设备读取数据。例如：

```
BYTE ReadBuffer[16];
```

缓冲区可为以下结构：

```
struct ReadBuffer {
    BYTE bStatus;
    int iVolts;
}ReadBuffer;
```

2. 在 RAM 中声明一个写入缓冲区，以供主设备写入数据。例如：

```
BYTE WriteBuffer[16];
```

缓冲区可为以下结构：

```
struct WriteBuffer {
    BYTE bCmd;
    int iDACCounts;
}WriteBuffer;
```

3. 通过调用 M8C_EnableGInt 宏使能全局中断。

```
M8C_EnableGInt ;
```

请务必使能全局中断，因为所有 I²C 操作均在 I²C ISR 内部后台执行。

4. 调用 I2CHW_Start、I2CHW_EnableSlave 和 I2CHW_EnableInt 函数以启动从设备。

```
I2CHW_Start();
I2CHW_EnableSlave();
I2CHW_EnableInt();
```

5. 利用 I2CHW_InitRamRead 初始化读取缓冲区。

```
I2CHW_InitRamRead(ReadBuffer,
sizeof(ReadBuffer));
```

此函数用于初始化指向读取缓冲区的指针，并设置缓冲区的大小。每当主设备尝试从从设备读取数据时，数据都会从缓冲区自动传输至主设备。如果主设备试图读取的字节数超出了缓冲区的大小，就会重复传输缓冲区的最后一个字节。

6. 利用 I2CHW_InitWrite 函数初始化写入缓冲区。

```
I2CHW_InitWrite(WriteBuffer,
sizeof(WriteBuffer));
```

此函数用于初始化指向写入缓冲区的指针，并设置缓冲区的大小。主设备将数据写入到从设备时，该数据会自

动存入缓冲区内。如果主设备试图写入的字节数超出了缓冲区的大小，则从设备会否认并丢弃这些额外的字节数。

7. 调用函数 I2CHW_bReadI2CStatus 并检查 I2CHW_RD_COMPLETE 标志。如果已设置此标志，表示主设备已完成读取数据操作。清除状态标志，重新初始化读取缓冲区。

```
// Check if a read operation is over
if (I2CHW_bReadI2CStatus()
I2CHW_RD_COMPLETE)
{
// Prepare fresh data for Master

// Clear the flag
I2CHW_ClrRdStatus();

// Re-initialize the buffer
I2CHW_InitRamRead(ReadBuffer,
sizeof(ReadBuffer));
}
```

请务必检查 RD_COMPLETE 标志并重新初始化缓冲区。否则，日后读取主设备时，会重复传输缓冲区中的最后一个字节。

8. 调用函数 I2CHW_bReadI2CStatus 并检查 I2CHW_WR_COMPLETE 标志。如果已设置此标志，表示主设备已完成写入数据操作。处理数据，清除标志，重新初始化写入缓冲区。

```
// Check if a write operation is over
if (I2CHW_bReadI2CStatus() &
I2CHW_WR_COMPLETE)
{
// Process data from Master

// Clear the flag
I2CHW_ClrWrStatus();

// Re-initialize the buffer
I2CHW_InitWrite(WriteBuffer,
sizeof(WriteBuffer));
}
```

请务必重新初始化写入缓冲区，否则从设备将会否认主设备中的所有后续数据。

该应用笔记中的一个简单项目将演示如何将 I²CHW 用作为从设备，用于回响由 I²C 主设备写入的数据。有关该项目的更多信息，请参见附录 B。

主设备操作

1. 在 RAM（或闪存）中声明一个读取缓冲区，以供储存自从设备读取的数据。例如：

```
BYTE ReadBuffer[16];
```

缓冲区的结构也可以与上面所述的“从设备操作”部分中讨论的结构相似。

2. 在 RAM 中指明一个写入缓冲区，用于写入 I²C 从设备的数据。例如：

```
BYTE WriteBuffer[16];
```

3. 通过调用 M8C_EnableGInt 宏使能全局中断。

```
M8C_EnableGInt;
```

请务必使能全局中断，因为所有 I²C 操作均在 I²C ISR 内部后台执行。

4. 调用 I2CHW_Start、I2CHW_EnableMstr 和 I2CHW_EnableInt 函数以启动主设备。

```
I2CHW_Start();
I2CHW_EnableMstr();
I2CHW_EnableInt();
```

5. 利用 I2CHW_bWriteBytes 函数向从设备写入数据。

```
I2CHW_bWriteBytes(0x50, WriteBuffer, 16,
I2CHW_CompleteXfer);
```

第一个参数是从设备地址；使用 7 位从设备地址。在 I²C 总线上发送地址之前，bWriteBytes 函数会自动将读/写位添加至 7 位地址。第二个参数是指向存有从设备数据的缓冲区的指针。第三个参数是要写入从设备的字节数。第四个参数是数据传输类型，共有三个不同的值：I2CHW_CompleteXfer、I2CHW_NoStop 和 I2CHW_RepStart。

当您使用 I2CHW_CompleteXfer 时，系统将执行一个完整的 I²C 数据传输操作，其中包括起始位、地址字节、数据字节和停止位。如果您使用 I2CHW_NoStop，写入数据之后不会生成停止位。使用 NoStop 的数据传输操作后可以紧跟一个使用了 I2CHW_RepStart 函数的数据传输操作。通常情况下，建议在执行大多数 I2C 数据操作中使用 I2CHW_CompleteXfer。

I2CHW_bWriteByte 函数用于初始化指向缓冲区的指针、设置计数值，以及启动 I²C 硬件中的起始位。之后，所有操作均在 ISR 内部执行。每次传输完一个字节，I²C 硬件就生成一个中断，而 ISR 负责递增指向缓冲区的指针，将下一个字节传输至从设备，并在传输完所有字节后生成一个停止位。

6. 使用 I2CHW_bReadI2CStatus 函数检查写入操作是否已完成，如果完成则清除标志。例如：

```
while(!(I2CHW_bReadI2CStatus() &
    I2CHW_WR_COMPLETE));
    I2CHW_ClrWrStatus();
```

上述代码会一直等待，直到 WR_COMPLETE 标志被设置为止。除了“while”语句，也可以使用“if”条件语句；在后台执行 I²C 数据传输时，处理器还可处理其他事务。

7. 使用 I2CHW_fReadBytes 函数以便启动对从设备进行的读取操作。使用 I2CHW_bReadI2CStatus 函数并检查 I2CHW_RD_COMPLETE 标志，然后清除读取状态。

```
I2CHW_fReadBytes(0x50,  ReadBuffer,  16,
    I2CHW_CompleteXfer);
while(!(I2CHW_bReadI2CStatus() &
    I2CHW_RD_COMPLETE));
    I2CHW_ClrRdStatus();
```

多主设备从设备操作

在多主设备从设备模式下，I²CHW 可在多主设备环境中作为主设备和从设备使用。

1. 调用 I2CHW_Start 函数。
2. 调用 I2CHW_EnableMstr 和 I2CHW_EnableSlave 函数以使能主设备和从设备模式，调用 I2CHW_EnableInt 函数以使能中断。

```
I2CHW_Start();
I2CHW_EnableMstr();
I2CHW_EnableSlave();
I2CHW_EnableInt();
```

3. 从设备模式操作与单个从设备操作相似，但函数名称却有所不同。
   - 分配读和写缓冲区；调用 I2CHW_InitSlaveRamRead 和 I2CHW_InitSlaveWrite 函数以便对这些缓冲区进行初始化。
   - 调用 I2CHW_bReadSlaveStatus 函数。
   - 检查 I2CHW_RD_COMPLETE 和 I2CHW_WR_COMPLETE 标志。如果已设置这些标志，则重新初始化缓冲区。
4. 负责主设备一方操作的固件需要处理一些特殊情况。在多主设备环境中，如果一个主设备（姑且称为主设备 1）试图启动针对从设备的数据传输操作，可能有三种不同的情景，具体取决于其他主设备（假设总线中存在另一个主设备，即主设备 2）的活动。
   a. 总线处于空闲状态，主设备 1 启动并完成一个读取或写入数据传输操作。
   b. 主设备 2 已经启动了与总线上的其他从设备进行的数据传输操作，因此总线处于繁忙状态。主设备 1 必须等到总线空闲，然后才能启动数传输操作。
   c. 主设备 1 启动一个读取或写入数据传输操作，与此同时，主设备 2 也启动一个数据传输操作。这种情况会引发仲裁。如果主设备 1 赢得仲裁，则完成数据传输操作。如果主设备 1 仲裁失败，则必须在主设备 2 完成其数据操作后，再重试数据传输操作。有关仲裁的详细信息，请参阅附录 A。

   I²CHW 提供了多个单独的函数，以供主设备处理多主设备环境。
5. 要写入从设备，请调用 I2CHW_bWriteBytesNoStall。要读取从设备，请调用 I2CHW_fReadBytesNoStall。这些函数会在启动数据传输操作之前检查总线是否繁忙。如果总线繁忙，这些函数会返回 0xFF。固件应检查返回值，如果该值为 0xFF，则重试数据传输操作。例如，以下代码会循环执行操作，直到 I2CHW_bWriteBytesNoStall 返回一个不是 0xFF 的值为止。

```
while(I2CHW_bWriteBytesNoStall(0x50,
WriteBuffer, 16, I2CHW_CompleteXfer) ==
0xFF);
```

除了可以使用被阻塞的“while”循环语句外，还可以使用“if”条件语句检查返回值。如果返回值为 0xFF，您可能在一段固定时间后重试数据传输操作。

6. 如果启动读取或写入操作时未出现总线繁忙错误，固件应继续检查操作是否完成或主设备的仲裁是否失败。以下为执行此操作的示例代码。

```
while((!(I2CHW_bReadMasterStatus() &
I2CHW_WR_COMPLETE)) &&
(!(I2CHW_bReadBusStatus() &
I2CHW_LOST_ARB)));
if (I2CHW_bReadMasterStatus() &
I2CHW_WR_COMPLETE)
{
    // Write is completed successfully.
    // Clear flag
    I2CHW_ClrMasterWrStatus();
}
if (I2CHW_bReadBusStatus() &
I2CHW_LOST_ARB)
{
    // Master lost arbitration. Retry
later
}
```

第一行指令会被循环执行，直至写入完成或 LOST_ARB 错误标志被设置为止。“while”循环终止时，代码会检查是哪一条件导致循环终止。如果已设置 WR_COMPLETE 标志，则清除该标志。如果已设置 LOST_ARB 标志，则稍后重试数据传输操作。

此应用笔记包含一个简单项目，用于演示如何使用 I²CHW UM 读取标准 EEPROM。有关详细信息，请参考附录 B。

有关更多信息，请参阅 I²CHW UM 数据手册和技术参考手册的 I²C 章节。

I²CHW 总结和重要说明

I²CHW 可在单一主设备、单一从设备或多主设备从设备等模式中运行。

在从设备模式下，当主设备完成读取或写入时，缓冲区必须重新初始化，以执行下一个数据操作。

在多主设备环境的主设备模式下，固件必须检查 bWriteBytesNoStall 和 fReadBytesNoStall 函数的返回值，以确定总线是否繁忙，然后重试数据传输。

在多主设备环境的主设备模式下，启动读取或写入数据操作时，固件应检查该数据传输操作是否顺利完成，或该主设备是否在仲裁上输给了另一个主设备。如果主设备仲裁失败，则需重试数据传输操作。

I²Cm

PSoC 1 中的另一个用户模块，即 I²Cm，通过软件操作控制通用 I/O 端口引脚来实施 I²C 主设备。该用户模块不采用 I²C 硬件模块，也不存在 I²C 软件从设备用户模块。

该用户模块相对于 I²CHW 的优势在于，它可用于任何一对引脚，而非仅限于 P1.5 与 P1.7 或 P1.0 与 P1.1。 缺点是它需要更多的 CPU 开销，而且不支持多主设备操作。执行 I²C 数据传输操作期间，它需要占用 100%CPU。第二个劣势是，它仅支持 ≤ 100 kHz 的总线频率。

在单芯片需要多个主设备，或未提供连接 I²C 硬件所需引脚的情况下，I²Cm 是最佳选择。如果仅需一个主设备，且提供了 P1.5 与 P1.7 或 P1.0 与 P1.1，赛普赖斯建议您采用 I²CHW 来执行主设备操作。

请遵循下列步骤在项目中运行此用户模块。

1. 使用 I²Cm_Start 函数启动该用户模块。
2. 要将数据从 RAM 缓冲区写入到一个从设备，请使用 I2Cm_bWriteBytes 函数。此函数接受从设备地址、指向 RAM（或 ROM）中源数据的指针以及要传送的字节数。
3. 要自从设备读取数据，请使用 I2Cm_fReadBytes 函数。此函数接受从设备地址、指向保存所读数据的目标缓冲区的指针以及要读取的字节数。

该用户模块的工作方式是，操控运行其端口的驱动模式寄存器（PRTxDMx）和数据寄存器（PRTxDR）。正因如此，在用户代码中操作这些寄存器时，请务必小心谨慎，否则 I²C 通信会受到不良影响。避免出问题的最好方法是，在写入到与 I²Cm 引脚相关的驱动模式寄存器和数据寄存器内时，使用影像寄存器。

当 I²Cm 用户模块被置于项目中时，PSoC Designer 会自动创建适用于数据寄存器和两个驱动模式寄存器（PRTxDM0 和 PRTxDM1）的影像寄存器。因为 PRTxDM2 不会受该用户模块影响，PSoC Designer 不会为该寄存器创建影像寄存器。

在 *psocconfig.asm* 和 *psocgpioint.h* 文件中定义这些影像寄存器。例如，如果 I²Cm 被置于 Port0 中，将在 *psocconfig.asm* 文件中定义下列变量。

```
; write only register shadows
_Port_0_Data_SHADE:
Port_0_Data_SHADE:          BLK     1
_Port_0_DriveMode_0_SHADE:
Port_0_DriveMode_0_SHADE:   BLK     1
_Port_0_DriveMode_1_SHADE:
Port_0_DriveMode_1_SHADE:   BLK     1
```

以下定义位于 *psocgpioint.h* 文件中。

```
extern BYTE Port_0_Data_SHADE;
extern BYTE Port_0_DriveMode_0_SHADE;
extern BYTE Port_0_DriveMode_1_SHADE;
```

应用程序代码使用这些影像寄存器向数据寄存器和驱动模式寄存器进行写入。例如，如果应用程序要设置 P0[0]，则应使用下列代码：

```
// Write to PRT0DR through shadow register
Port_0_Data_SHADE |= 0x01;
PRT0DR = Port_0_Data_SHADE;
```

使用 PRT0DMx 寄存器将 P0[0]设置为强驱动模式：

```
// Write to PRT0DM0 through shadow register
Port_0_DriveMode_0_SHADE |= 0x01;
PRT0DM0 = Port_0_DriveMode_0_SHADE;

// Write to PRT0DM1 through shadow register
Port_0_DriveMode_1_SHADE &= ~0x01;
PRT0DM1 = Port_0_DriveMode_1_SHADE;

// Write to PRT0DM2 directly
PRT0DM2 &= ~0x01;
```

有关需要使用影像寄存器的情况的更多信息，请参考影像寄存器数据库.

有关此用户模块的详细信息，请参阅 I²Cm 数据手册。

I²Cm 总结和重要说明

I²Cm 是 I²C 主设备的软件实现，不会占用 I²C 硬件资源。

I²Cm 对从设备进行读取或写入时，会使用全部 CPU 资源。

如果应用程序代码需写入到 I²Cm 所在端口的数据寄存器或驱动模式寄存器内，请务必谨慎使用影像寄存器，以免 I²C 接口出现问题。

I²C 特别注意事项

I²C 为器件间相互通信提供了一种简易方式。与任何设计一样，该设计过程中也可能出现问题。本章节旨在帮您解决 I²C 设计过程中所遇到的常见问题，避免可能遇到的难题。它包括下面各主题：

- 7 位和 10 位寻址
- 上拉电阻的注意事项
- 共享 I²C 和 ISSP 引脚
- 上电时所发生的短时脉冲
- SYSCLK 和 I²C 时钟的速度
- 时钟延展和中断延迟
- 热插拔
- 短时脉冲过滤
- I²C 和睡眠模式
- I²C 和动态重配置
- I2CHW 中的动态寻址

I²C 寻址

I²C 从设备地址有两种常见的描述方法。第一种方法是使用 7 位地址，没有读/写位，例如 0x42。这是赛普拉斯的用户模块处理 I²C 从设备地址的方式。

第二种方法是在地址中加入读/写位（例如 0x84/0x85）。

请确保知悉您所用器件的寻址方式。

目前为止，赛普拉斯尚未支持 10 位从设备地址。

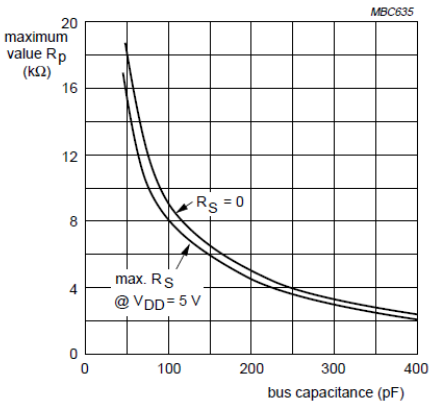
上拉电阻

I²C 设计中另一个常见注意事项是上拉电阻的值。应根据通信频率和总线电容选择该电阻值。较大的总线电容和上拉电阻会延长时钟或数据线路的上升时间。I²C 规范列明了最大上升时间。

如果总线上升时间大于该最大上升时间，I²C 通信无法正常进行。I²C 规格中提供了如图 18 所示的曲线图，帮助用户确定上拉电阻的值。Rs 是串联电阻，I²C 规范中规定了 Rs 最大值。

对于大多数系统而言，上拉电阻值的范围为 2.2 k 到 4 k。

图 18. 上拉电阻值



I²C 和 ISSP 编程冲突

将 I²C 与 PSoC 配合使用时的，一个常见注意事项是选择要用的引脚。使用硬件模块时，I²C 可使用这两对固定引脚：（P1.5，P1.7）和（P1.0，P1.1）。（在 28xxx 系列中，您还可以使用 P1.2 和 P1.6 或 P3.0 和 P3.2。）您也可以使用 P1.0 和 P1.1 引脚来编程 PSoC。这些线路上的上拉电阻会导致系统内编程出错。

编程期间，P1.0 使用电阻下拉驱动模式强制降低线路上的逻辑电平。下拉电阻约为 5.6 kΩ。内部下拉电阻和外部 I²C 上拉电阻形成了一个电压分频器。该电压分频器在线路上应用一个不确定或高逻辑电平，而编程器不会将此电平视为所需的低逻辑电平。这便会导致编程失败。如需确定合适的电

压电平，请参阅器件特定数据手册中的 DC 编程规范和通用 I/O 逻辑电平。

最佳解决方法是使用 P1.5 和 P1.7 引脚。如果需要，才能使用 P1.0 和 P1.1 这对引脚。另一个选择是，在 PSoC 中将 P1.0 与 P1.1 的驱动模式更改为上拉模式，而不是使用外部上拉电阻。内部上拉电阻约为 5.6 kΩ。

上电时引脚短时脉冲

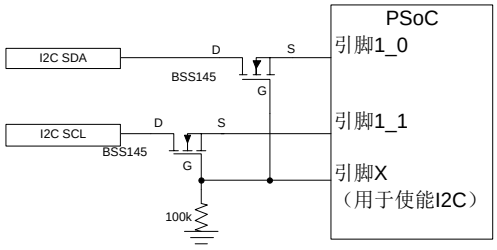
如前面所述，P1.0 和 P1.1 用于编程，也就是说，上电时这些引脚的行为不同于其他引脚。

器件从复位释放后，P1.0 会被驱动为强驱动高电平。这时，如果总线上有其他 I²C 器件驱动低电平，就会导致故障。P1.1 将被驱动为电阻式低电平。如前所述，这将在总线上形成一个电压分频器和中间电压，而其他器件可能无法识别此电压。超过设置的时间量后，P1.0 跃变为电阻式低电平，因此在该线路上引发中间电压。

PSoC 启动时，如果其他 I²C 器件被充电，且与 P1.0 和 P1.1 相连，则您需要注意上述行为及其对总线上 I²C 器件的影响。

要避免发生此问题，应在 I²C 通信中采用 P1.5 与 P1.7。或需要采取措施，尽可能地减少复位期间引脚行为对 P1.0 与 P1.1 的影响。简单来做，可以将所有其他器件保持在复位状态，直至 PSoC 启动为止。也可以采用较为复杂的解决方案，例如利用转换器选通 PSoC 的输出或逻辑门。请参见图 19。

图 19. 隔离 PSoC



注意:

- 请务必在此模块中准确放置 N 通道 FET，以便 FET 的源极连接至 PSoC GPIO 引脚。
- FET 需要采用 N 通道，并必须有 VGS_{TH} 的额定值≤ PSoC V_{dd}。举例来说，如果 PSoC V_{dd} = 3.3 V，则选择 VGS_{TH} 的额定值 ≤ 3.3 V 的一个通道 N，即 FET。
- 将引脚 X 设置为强驱动并将‘1’写入到该引脚内，从而连接到 I²C 总线。

时钟速度

如前所述，当处于主设备模式时，I²C 硬件负责在 SCL 上生成时钟。PSoC 1 中可用的 I²C 时钟频率为 50 kHz、100 kHz 和 400 kHz。这些频率是基于系统时钟（SYSCLK）固定连接的时钟分频器得到的。

此类时钟分频器生成采样时钟；这些采样时钟对 I²C 线路进行 16 倍或 32 倍过采样。表 4 列出了内部采用率。

表 4. 内部采样率

时钟频率	SYSCLK 预分频器	内部采样时钟频率（SYSCLK = 24 MHz）	样品数/位
50 kHz	/16	1.5 MHz	32
100 kHz	/4	1.5 MHz	16
400 kHz	/16	6 MHz	16

内部采样时钟假设 SYSCLK 的频率为 24 MHz。如果 SYSCLK 的频率低于 24 MHz，则 I²C 时钟会变慢。例如，如果 SYSCLK 为 12 MHz，则可用速度为 25 kHz、50 kHz 和 200 kHz。如果 SYSCLK 为 6 MHz，则 I²C 的可用速度为 12.5 kHz、25 kHz 和 100 kHz。（请参考表 5。）

表 5. 实际频率与 IMO 和用户模块设置

用户模块设置	IMO（SYSCLK）设置		
	24 MHz	**12 MHz**	**6 MHz**
400 kHz	400 kHz	200 kHz	100 kHz
100 kHz	100 kHz	50 kHz	25 kHz
50 kHz	50 kHz	25 kHz	12.5 kHz

注意： SYSCLK 独立于 CPU_Clock。

在从设备模式下操作时，此规则同样适用于 I²C 模块可读取的最高时钟速度。过采样时钟用于监控 I²C 线路。在从设备模式下所设置的速率表示硬件对时钟和数据线路进行过采样的频率。

如果 I²C 时钟频率为 400 kHz，并且硬件被配置为 100 kHz，则硬件无法正常接收数据。一个常见错误是，在 PSoC Designer 中将 I²C 从设备时钟频率设置为 100 kHz，而将 SYSCLK 设置为 6 MHz。认为从设备将在 100 kHz 的总线上运行的想法是错误的。因为 SYSCL 为 6 MHz，所以从设备仅能在 25 kHz 的总线上运行。由此可见，要在 SYSCLK 为 6 MHz 的情况下，在 100 kHz 的总线上运行，必须在用户模块属性中将 I²C 速度选为 400 kHz。

SYSCLK 的频率取决于 PSoC 的供电电压。如果供电电压超过 3 V，则大多数器件的 SYSCLK 为 24 MHz。如果供电电压小于 3 V，则大多数器件的 SYSCLK 为 6 MHz。有多个器件可选择 12 MHz 的 SYSCLK。此外，可在 PSoC Designer 中更改 SYSCLK 的值。使用 I²C 时，请勿更改

SYSCLK 或 CPU_CLK 的频率，否则会导致在 I²C 线路上发生短时脉冲。有关时钟的更多信息，请参见技术参考手册的“时钟”章节中的内容。

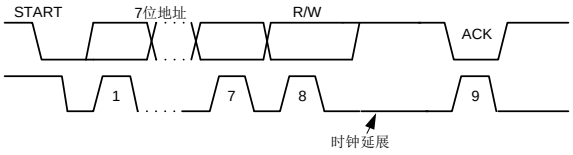
使用外部时钟时，请确保使用正确的 I²C 时钟速度。

时钟延展和中断延迟

时钟延展是从设备将时钟线路保持在低电平状态，从而停止主设备在总线上进一步通信的过程。通常情况下，从设备延展时钟是为了处理接收自主设备的信息，或准备更多数据以发送至主设备。时钟延展可在数据传输操作的任何时间点完成。PSoC 在字节完成中断后延展时钟。请参见图 20。

I²C 规范表明时钟延展为可选功能；并非所有 I²C 器件均需支持此功能。不过，所有赛普拉斯 PSoC 1 I²C 从设备用户模块都具备延展时钟功能。如果您所用的主设备不支持时钟延展，则总线会锁定且无法复位。

图 20. 时钟延展示例

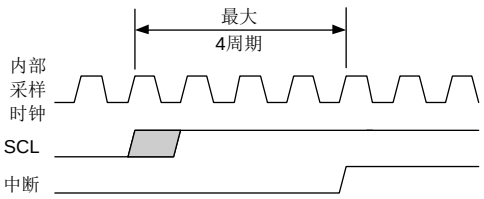


PSoC 从设备延展时钟所需时间长短取决于以下几个因素：主设备对从设备进行的是读操作还是写操作，系统中是否发生了其他中断，以及 CPU 的速度。

在 I²C 总线速度等于或高于 100 kHz 的情况下，PSoC 1 器件的时钟延展时间会增加。EzI2Cs 和 I2CHW UM 中的 ISR 代码包含 150 至 300 个 CPU 指令循环。在 CPU 时钟为 24 MHz 时，ISR（中断服务子程序）执行时间约为 6 至 13 µs。ISR 代码执行完毕后，将释放 SCL 线路。

在 SCL 上升沿后三到四个内部采样时钟周期中触发 ISR，请参见图 21。这是因为 SCL 上有一个内部短时脉冲过滤；请参见图 22。请参见表 4 以更加了解采样时钟的频率。这意味着 ISR 将在 SCL 上升沿后约 2.5 µs 被触发。100 kHz 时钟的正常周期是 10 µs。如果触发 ISR 需要 2.5 µs，且执行 ISR 需要 6 至 13 µs，则大部分情况下都会发生时钟延展。

图 21. 字节完成中断时序



传输：第九个正向沿SCL
接收：第八个正向沿SCL

通过此信息，您可以确定时钟是否将被延展及延展多长时间。不过，如果系统中有其他中断，则应将这些中断耗时考虑在内。

要最小化 PSoC 中的时钟延展时间，第一步是将 CPU 的运行速度控制在 24 MHz。接下来，I²C 时钟速度必须不超过 100 kHz。最后，尽量减少中断，以缩短时钟延展时间。

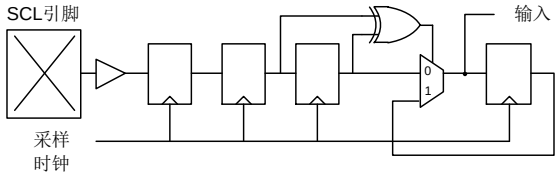
热插拔

热插拔是将已充电的器件连接至未充电的 PSoC 的过程。PSoC 中 I²C 的设计尚不支持热插拔。对 PSoC 进行热插拔时需考虑若干因素。第一个问题是反向供电。如果 PSoC 未充电，但是一个外部引脚处于高压状态，则可能向 PSoC 器件反向供电。这是我们不希望看到的情况，因为 PSoC 可能被意外执行，并且 I²C 线路可能受到负面影响。

短时脉冲过滤

SCL 的输入有一个短时脉冲过滤，如图 22 所示。

图 22. SCL 短时脉冲过滤



输入与采样时钟双重同步，然后进行短时脉冲过滤。原始输入和延迟输入必须相互匹配，这样才能传送信号。因为采样时钟为 1.5 MHz 或 6 MHz，相应地，小于 666 ns 和 166 ns 的短时脉冲将被抑制。

I²C 和睡眠模式

如果在具有进入和退出睡眠模式操作的设计中使用了 I²C，那么需要特别注意以下几项。

首先，I²C 进入睡眠模式之前，务必完成所有 I²C 数据传输操作。否则，器件被唤醒时，I²C 时钟会将数据错误解读为地址，或者将地址解读为数据。确定所有 I²C 通信均已停止后，请执行以下步骤：

- 配置 I²C 引脚为 HIGHZ 驱动模式。
- 禁用 I²C 模块。通常，可以通过调用用户模块的 Stop() API 完成此操作。
- 清除所有待处理的 I²C 中断。

满足上述条件之后，PSoC 可进入睡眠状态。当器件从睡眠状态被唤醒时，执行以下步骤以确保 I²C 在经过睡眠后可正确运行：

- 确保总线上没有任何 I²C 活动。
- 调用相应的 Start API。

- 配置 I²C 引脚为开漏低电平驱动模式。
- 使能中断。

在结合使用 I²C 和睡眠模式时，只要遵循这些步骤，便能够避免大多数错误。

I²C 和动态重配置

切勿通过动态重配置加载或卸载 I²C 用户模块。它们应始终存在，如被加载或卸载，则将发生 I²C 错误。

I²C 用户模块应位于基础配置或始终加载着的单独外覆层中。

I²CHW 用户模块中的动态从设备寻址

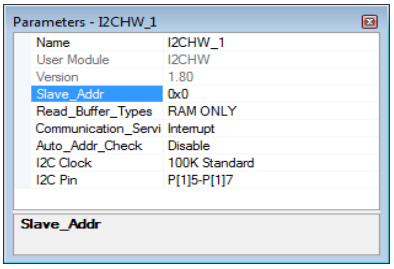
EzI2Cs UM 允许您通过编程方式随时更改 I²C 从设备地址。不过，I²CHW UM 不具备此功能；若要实现相同功能，请执行以下步骤：

在 I²CHW 用户模块中，从主设备所接收的地址将在 *I2CHW_1int.asm* 文件内部进行处理。以下代码用于执行地址匹配功能。

```
mov A, reg[I2CHW_1_DR]
and F, 0xF9
rrc A
xor A, I²CHW_1_SLAVE_ADDR
```

从主设备接收的地址包括一个 7 位地址和读/写位。“rrc A”放弃读/写位，将 7 位地址与常量 I2CHW_1_SLAVE_ADDR 进行比较，该常量由器件编辑器根据 I²CHW 的 UM 参数中所设置的 Slave_Addr 而创建。此常量位于 *I2CHW_1.inc* 文件中。

图 23. I²CHW 用户模块从设备地址



要将 RAM 变量代替该常量，请按照这些步骤执行操作：

1. 创建一个 RAM 变量以保存动态从设备地址。在 *I2CHW_1int.asm* 文件中的变量分配区域下，有一个自定义用户代码区域。在自定义声明区域，添加此代码：

   ```
   export _I2CSlaveAddress
   export I2CSlaveAddress

   在变量分配区域，添加下列代码：

   Area InterruptRAM(ram)
   _I2CSlaveAddress:
   I2CSlaveAddress:   BLK   1
   ```

   带‘_’的变量可在 C 语言代码中进行修改。

2. 修改用于比较地址的代码。

   ```
   mov A, reg[I2CHW_1_DR]
   and F, 0xF9
   rrc A
   xor A, [I2CSlaveAddress]
   ```

   因为此代码位于自定义用户代码区域内，应用程序生成期间对用户模块库文件的更改将予以保留。请注意，如果重命名了用户模块，这些更改将被丢失，需要重新进行更改。

3. 要为 *I2CHW_1int.asm* 文件中定义的 I²C Slave Address 变量添加一个引用，只需将下列代码添加至 *main.c*（或任何其他 C 语言文件）内。

   ```
   extern BYTE I2CSlaveAddress;
   ```

SCL 线路处于低电平停滞状态

使用 I²C 时遇见的一个常见问题是 SCL 处于低电平停滞状态。以下内容可指导您调试和解决 PSoC 1 器件中的这个问题。

1. 您是否在 PSoC Designer 5.0 SP5 中使用 EzI2Cs？如果为“是”，该版本的用户模块中存在一个已知缺陷，它会导致 SCL 线路处于低电平停滞状态。更新至 PSoC Designer 的最新版本即可修复此问题。请参见 I²C 时钟始终为逻辑低电平停滞状态文章。
2. 使用 PD5.0 之前，您是否使用 PSoC Designer 版本？如果为“是”，则 P1.5 与 P1.7 在上电时会有短时脉冲。此问题已在 PSoC Designer 的最新版本中妥当解决。更多有关信息，请参考上电时 I²C 线路上发生短时脉冲文章。
3. 主设备是否支持时钟延展？如果为“否”，从设备和主设备之间的通信会不同步，而且可能发生意外行为，包括 SCL 处于低电平停滞状态。如果主设备不支持时钟延展，则无法保证 PSoC 在总线上正常运行。
4. 代码执行期间，CPU 时钟或 SYSCLK 是否会动态更改？如果为“是”，这会导致在 SCL 和 SDA 线路上发生短时脉冲。为避免该问题，请勿在代码中更改时钟速度。
5. 您是否在代码中是能和禁用了 I²C 中断？如果为“是”，请务必使用 ResumeInt API 而非 EnableInt API。如果有待处理的 I²C 中断而且被您清除了，SCL 将永远停滞在低电平状态。
6. PSoC 启动时总线上是否有其他 I²C 通信？如果为“是”，则表示存在一个已知缺陷，不过最新版本的 PSoC Designer 已修复该缺陷。如果您没有最新版本，且无法更新，可采用以下解决方法：在芯片编辑器中，将 I²C 引脚的驱动模式设为模拟 High-Z 模式。在主代码中使能 I²C 用户模块。然后将 I²C 引脚的驱动模式设为开漏低电平驱动模式。

7. 项目中是否使用了睡眠模式？如果为“是”，请遵循“睡眠模式”章节中讨论的步骤，以避免发生任何问题。

最新版本的 PSoC Designer 已修复和解决了上述大部分问题。赛普拉斯始终建议使用 PSoC Designer 的最新版本。

总结

I²C 是一个简单的两线制芯片对芯片数字通信协议。该协议面向主设备，但允许在两条通信线路上进行双向通信。

赛普拉斯 PSoC 提供了多个用户模块，以便在设计中实施 I²C，其中包括从设备、主设备和多主设备配置。如果您严格遵守了赛普拉斯所建议的注意事项，那么在 PSoC 中实现 I²C 通信会变得非常简单可靠。

关于作者

姓名： Todd Dust

职务： 应用工程师

背景： 西雅图太平洋大学电气工程学（BSEE） 学士学位

联系地址： tdu@cypress.com

姓名： M. Ganesh Raaja

职务： 首席应用工程师

背景： Ganesh 毕业于印度本地治里中央直辖区的莫逖拉尔·尼赫鲁理工学院（Motilal Nehru Govt. Polytechnic）电子与通信工程专业。 他在模拟电路设计和微控制器方面拥有 20 年经验。他还在赛普拉斯公司网站上开通了名为“PSoC Hacker”的博客。

联系地址： graa@cypress.com

附录 A

本附录是针对希望详细了解 PSoC 中 I²C 硬件模块的操作的用户。

硬件寄存器

有多个寄存器用于控制 I²C 硬件模块和报告其状态。

I2C_CFG 寄存器：控制 I²C 硬件模块的配置。它控制时钟速度和所用的引脚，并且指示是否使能了从设备或主设备的功能，同时还支持出现停止条件和总线错误时启动中断。

I2C_SCR 寄存器：返回 I²C 硬件模块中的状态标志。它报告是否发送或接收到一个完整字节（字节完成）。该寄存器指示是否发生了总线错误还是仲裁失败。它还可确定最后一个数据操作是否为一个地址。

I2C_DR 寄存器：该寄存器保存已发送或接收的数据值。仅在下面三种情况下，它才对数据进行移位：数据是一个地址；将硬件模块配置为从设备并对它进行寻址；主设备初始化读取操作。

I2C_MSCR 寄存器：该寄存器用于控制 I²C 数据操作的主设备部分，并保存允许生成“启动”条件的位。必须设置 I2C_CFG 寄存器中的使能主设备的位，该寄存器才可用；否则寄存器会保持在复位状态。

固件需求

本章节介绍了不同配置下固件模块的操作。请注意，文中无论何处提及固件需求，所有 PSoC I²C 用户模块均运行该代码。

启动生成

如果在主设备状态和控制寄存器中设置 start gen（启动生成）位，硬件模块会生成一个启动条件，然后发送数据寄存器中的地址。但是，该硬件可识别是否有其他器件在控制总线。如果检测到外部启动条件，硬件会将当前启动列入到队列中，直到总线进入空闲状态为止。直到硬件成功发出启动条件，才清除该启动位；或者使用固件清除它。

每一次只能将一个启动条件列入队列。总线繁忙时，如果主设备试图发送两个启动条件，硬件仅会发送最后一个地址。如前所述，用户代码必须确保不会发生这种情况。

主设备操作

在主设备模式下，发送启动条件和地址后，硬件会一直等到收到从设备发出的 ACK/NAK 位。接收到 ACK/NAK 位时，硬件会中断 CPU。随后，固件将确定从设备是否已确认该地址。固件需要通过读取 I2C_SCR 寄存器中最后接收到的数据位（LRB）来实现该操作。如果该数据位的值为‘0’，则表示从设备已确认了该地址；如果该数据位为‘1’，则表示从设备未确认该地址。应根据这种情况采取相应的措施。

固件还通过设置 SCR 寄存器中的传输位来设定通信的方向。 如果将‘0’写入该位，则模块将被置于接收模式。在该模式下，硬件会在接收到八个数据位时中断 CPU。中断后，固件必须确定是否需要发送 ACK/NAK。

如果将‘1’写入传输位，则模块将被置于传输模式。在该模式下，硬件会在接收到从设备发出的 ACK/NAK 后中断 CPU。然后固件将再次处理这些情况。

写入到 SCR 寄存器时，主设备将开始生成时钟以发送或接收更多数据。

从设备操作

在从设备模式下，硬件在检测到启动条件时，会将下一个八位数据移入 I2C_DR 寄存器内。接收到第八位数据时，硬件模块会中断 CPU，并使 I2C_SCR 寄存器中的地址位变为高电平。

发布中断后，硬件将时钟线路保持在低电平。随后，固件负责读取输入的地址，并确认该地址是否为自己的地址。固件必须适当地设置 SCR 寄存器中的 ACK 位。固件还需要读取该地址的读/写位。如果该位为‘1’，则 SCR 寄存器中的传输位需要改为‘1’。CPU 写入至 SCR 寄存器后，硬件释放时钟线路，这样能够继续进行数据操作。

如果将从设备配置为发送器，那么固件必须将有效的数据加载到 DR 寄存器内，然后再写入到 SCR 寄存器并释放总线。总线释放后，硬件在主设备提供的时钟沿上将数据移出至 SDA 线路。硬件会一直等到接收到主设备发出的 ACK/NAK，然后将中断 CPU。这时，固件必须加载新数据或不执行任何操作。

如果将从设备配置为接收器，硬件会在收到第八位数据后中断 CPU。固件将决定是否能够接收更多数据。然后，必须设置相应的 ACK 位。

停止条件

在主设备模式下，当完成了某个数据操作时，硬件模块将生成停止条件，以指示总线处于空闲状态。

在从设备模式下，收到停止条件后，硬件模块会进入闲置模式，直至接收到新的启动条件。

中断源

在之前的示例中，硬件模块在收到字节完成条件后会中断 CPU。此类中断的发生取决于通信的方向。

- 发送：字节完成中断 = 第九位
- 接收：字节完成中断 = 第八位

硬件模块支持其他两个中断源。通过设置 I2C_CFG 寄存器中的相应位，即可启用这些中断。硬件可在收到停止条件后进行中断。在从设备模式下运行时，该功能非常有用；该中断可提醒固件当前数据操作已完成。还可以在发生总线错误（即启动或停止条件在总线上出现了错误）时生成中断。如果硬件检测到该中断，那么它将停止当前的操作，并发出中断请求；此时固件会决定如何处理这种情况。

仲裁

硬件模块还可以指示主设备的仲裁是否失败。在多主设备模式下操作时，该功能非常重要。当两个主设备同时开始进行写操作时，便会发生仲裁。硬件监控 SDA 线路。如果主设备试图将总线保持为高电平，而另一个主设备则将其拉低，那么表示前者仲裁失败。仲裁条件发生后，硬件不再控制 SDL 线路，但会继续给 SCL 线路提供时钟脉冲。在后续字节完成中断时，固件将检查仲裁失败位，以查看上一传输过程中，仲裁是否失败。

I²C 的基本流程

图 24 和图 25 显示了 I²C 数据操作的基本流程。这是在 PSoC 1 中成功实施 I²C 应遵循的基本流程。所提供的用户模块遵循该流程时会增加开销。

图 24. 用于成功配置从设备发送器/接收器的流程图

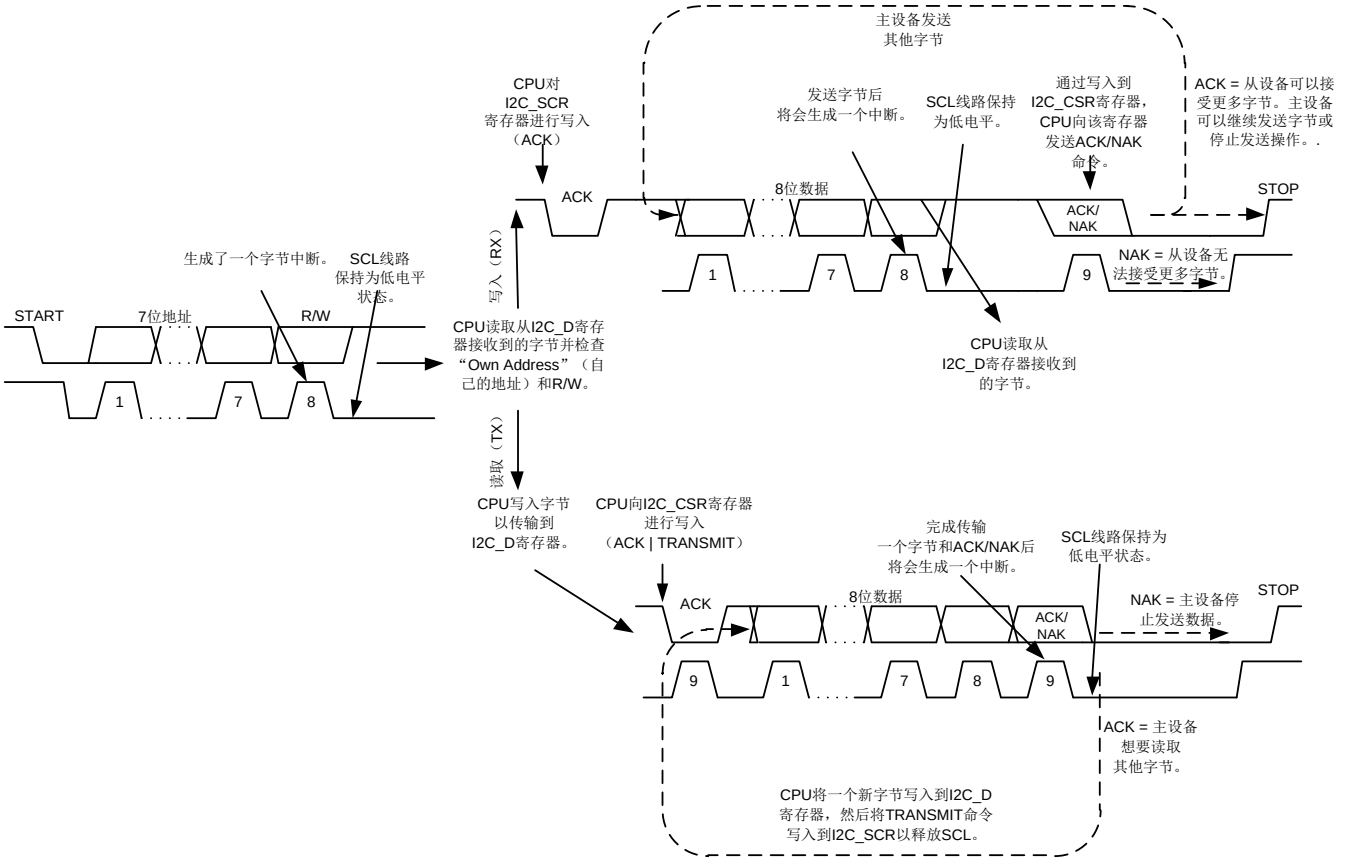
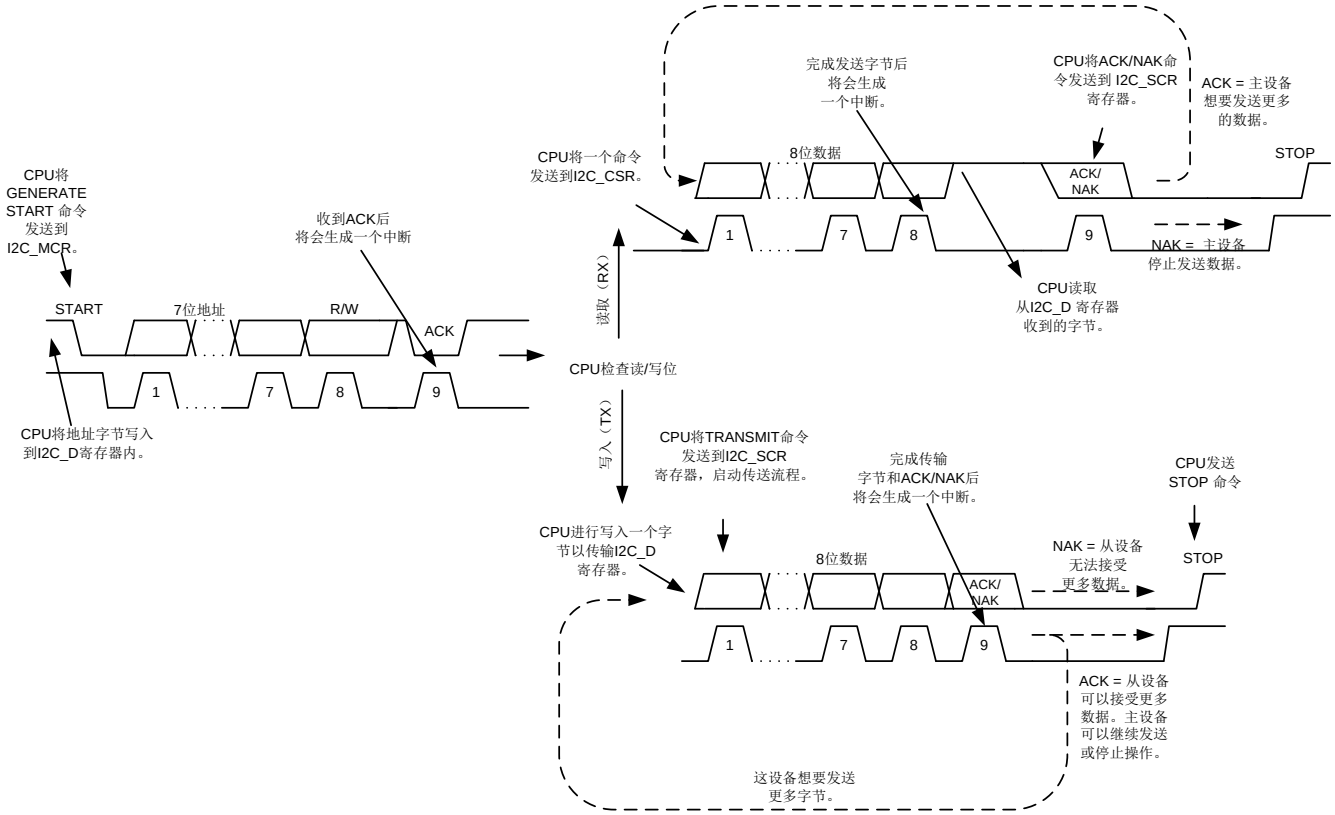


图 25. 用于成功配置主设备发送器/接收器的流程图



附录 B

该附录介绍了多种示例项目， 通过这些项目演示了 I²C 用户模块（UM）的用途。

EzI2Cs_ADC_LED_DAC 示例项目

该项目用于演示如何配置 EzI2Cs UM。该项目将 PSoC 配置为 I²C 从设备，并且可将其作为一个实现了由 I²C 控制的模拟外设器件以及端口扩展器。

该项目创建了下面的数据结构：

```
struct I2CRegs
{
    BYTE LEDValue;/* Updates LEDs */
    BYTE DACValue;/* Updates DAC */
    BYTE ADCValue;/* Reads ADC value */
}I2CRegs;
```

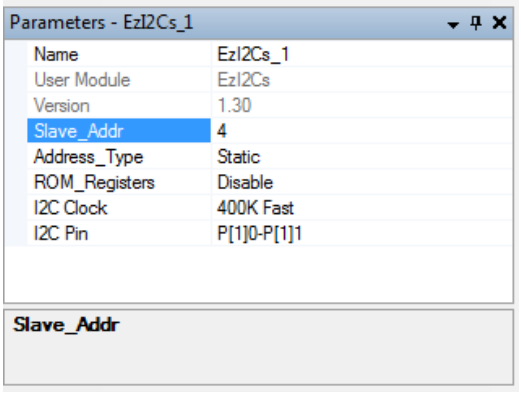
通过调用以下 API，可以将该结构通知给 I²C 主设备：

```
EzI2Cs_1_SetRamBuffer(sizeof(I2CRegs),    2,
(BYTE *) &I2CRegs);
```

第一个参数设置数据结构的大小。第二个参数则用于设置可读/写参数的数量。不属于该范围的参数均为只读参数的。在该结构中，LEDValue 和 DACValue 都是读/写参数的，并且 ADCValue 为只读参数。最后的参数为指向数据结构自身的指针。

根据图 26 配置芯片中的 EzI2Cs UM。

图 26. EzI2Cs UM 配置



依次进行下列操作：将从设备地址设置为 0x04、地址类型为静态；禁用 ROM 寄存器；将时钟速度设为 400 kHz，快速模式；并将 P1[0]和 P1[1]作为 I²C 引脚使用。使用 P1[0]和 P1[1]，以便使 CY3210 PSoC 1 评估板上的 ISSP 端口还可用于连接 I²C 主设备。

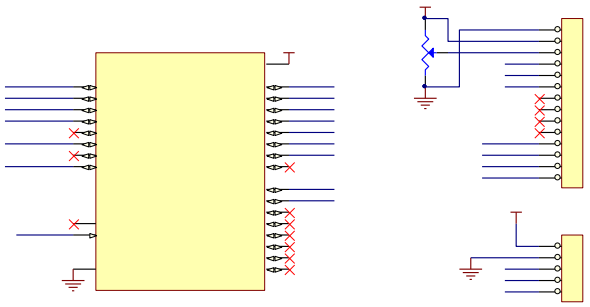
在固件中，ADC 用于读取 P0.7 上的电压。通过在该引脚上连接一个电位器，您可以仿真各个不同的电压。ADC 读取的值会被写入至变量 ADCValue。可用一个主设备读取该变量。

在固件中创建了 LEDValue 和 DACValue 的局部副本。将 I²C 寄存器结构中的 LEDVAlue 和 DACValue 值连续同这些参数的局部副本进行比较。当主设备写入另一个 DAC 值或 LED 值时，将会使用新的值来更新 LED 端口（P0[0]到 P0[3]）和 DAC。DA 输出可用于 P0[5]。

测试项目

图 27 显示了项目测试所需的设置。使用一个 CY3210 PSoC 1 评估板对设置的对象进行布线。

图 27. EzI2Cs_ADC_LED_DAC_Project 的原理图



将 P0[0]至 P0[3]分别连接到 J5 上的 LED1 至 LED4 信号。将 P0[7]连接到可变电阻 VR。将数字万用表连接到 P0[5]，以监控 DAC 输出。将 LCD 连接到 LCD 连接器（J9）。

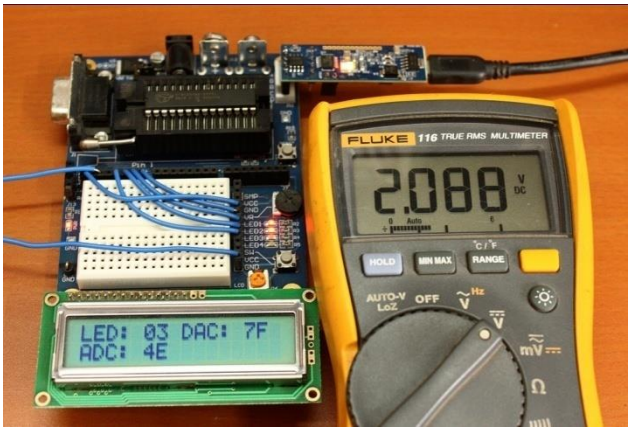
通过 CY3217 MiniProg1 或 CY8CKIT-002 MiniProg3，可以编程使用 CY3210 评估板上的 ISSP 插头的器件。清参见表 6。

可将 CY3240 USB-I²C 桥接器 PSoC 开发套件或 MiniProg3 作为 I²C 主设备使用。图 28 显示的是进行测试项目时的 CY3210 评估板。此处使用了 CY3240 I²C-USB 桥接。

表 6. CY3210 评估板的设置情况

PSoC 1 的引脚	CY3210 的连接	说明
P0[0]到 P0[3]	LED1 到 LED4	将 P0[3:0]分别连接至四个 LED
P0[7]	VR	电位器输入
P0[5]	–	连接万用表
–	ISSP 插头（J11）	连接 MiniProg1 或 MiniProg3 以进行编程
–	ISSP 插头（J11）	连接 Cy3240/MiniProg3， 以通过 I2C 与 PC 进行通信

图 28. EzI2Cs 示例项目的测试设置



可通过以下两种方法测试该项目。您可以使用桥接控制面板软件，也可以使用外部 MCU。

使用桥接控制面板软件进行测试

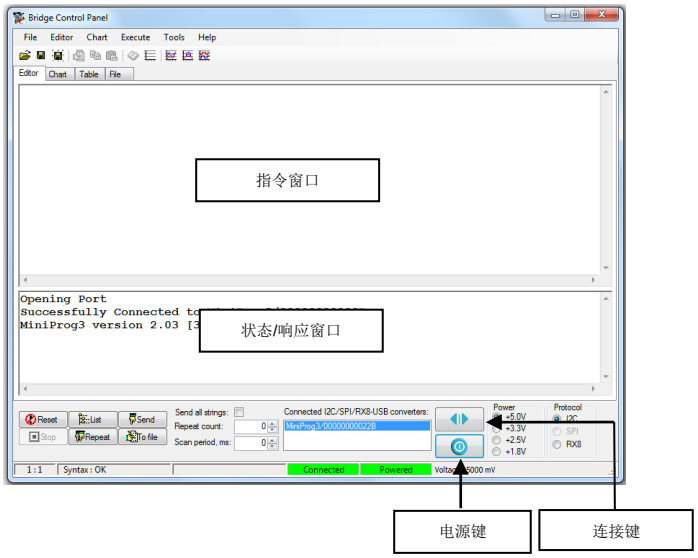
赛普拉斯的桥接控制面板软件可作为图形前端使用，从而同 PSoC I²C 从设备进行通信。它有助于进行测试、调节以及调试具有 I²C 从设备接口的程序。您可以将桥接控制面板与 PSoC Designer 和 PSoC Programmer™一起安装。

编程该器件后，继续将 CY3240 或 MiniProg3 连接至 ISSP 连接器。CY3240 和 MiniProg3 都使用了 I²C 的 SDA 和 SCL 线所需的上拉电阻。

下面的流程介绍了如何使用桥接控制面板读取 ADC 值并控制 EzI2Cs_ADC_LED_DAC 项目的 LED 和 DAC 输出。

1. 从 Windows 的启动菜单中打开 Bridge Control Panel。它位于赛普拉斯文件夹中。
2. 选择器件列表中的 MiniProg3 或 CY3240，并点击 Connect（连接）按键。
3. 随后，点击 Power（电源）按键，实现为 CY3210 测试设置供电。（请参见图 29。）

图 29. 桥接控制面板



4. 为了写入 LED 和 DAC 参数，在 Bridge Control Panel（桥接控制面板）的指令窗口中，请键入以下指令：

   w 04 00 03 80 p

   其中，‘w’表示写入指令，‘04’表示从设备地址，‘00’表示数值被写入到的辅助地址内，‘03’为 LED 的值。数值为‘03’时，LED1 和 LED2 均被打开。‘80’为 DAC 输出，这时，DAC 的输出大概为 2.09 V。

   在指令窗口中输入该指令并按下 Enter 键时，I²C 主设备会将该指令发送给 EzI2Cs 从设备。请观察结果窗口中 LED 的状态和 P0[5]上的输出。（请参见图 30。）

   ```
   w 04+ 00+ 03+ 80+ p
   ```

   每个字节后面的‘+’符号表示 EzI2Cs 从设备已经确认了该字节。‘–’ 符号则表示 EzI2Cs 从设备尚未确认该字节。

   例如，尝试输入以下指令：

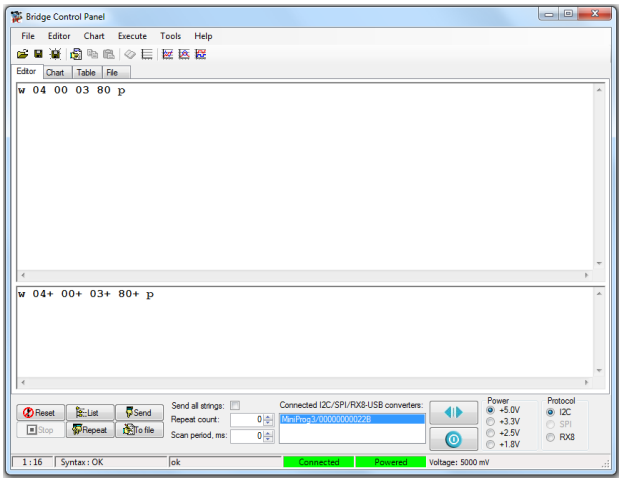
   w 04 00 03 80 55 p

   响应为：

   w 04+ 00+ 03+ 80+ 55– p

   因为已经将 EzI2Cs 寄存器结构中的第三个字节设置为只读字节，所以从设备不会应答（NAK）对该寄存器进行的写操作。

图 30. 写入 LED 和 DAC 值



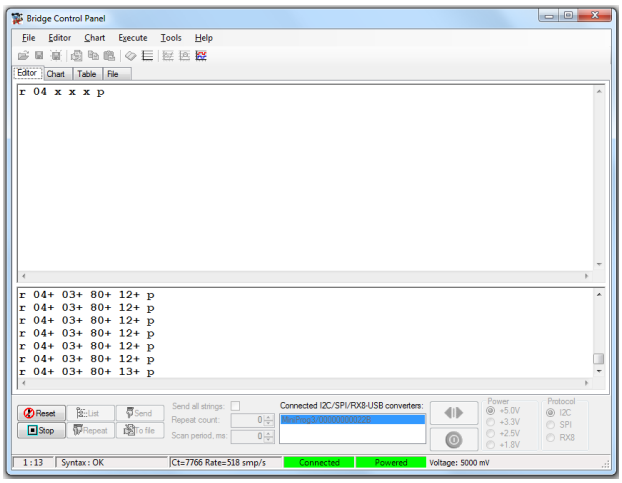
5. 为了读取 ADC 结果，要在指令窗口中键入以下指令：

   r 04 x x x p

   其中，‘r’ 表示读取指令，‘04’为从设备地址，三个‘*x*’表示读取从设备中的字节数量。

   现在，点击桥接控制面板中的 Repeat（重复）按键，并观察结果窗口中显示的结果。（请参见图 31。）

图 31. 读取 I²C 寄存器的所有值。



   响应中最后一个字节是 ADC 结果。第二和第三个字节分别表示 LED 和 DAC 参数的响应值。

6. 如果只想读取 ADC 值，不读取 LED 和 DAC 值，（请参见图 32），首先要执行下面的指令：

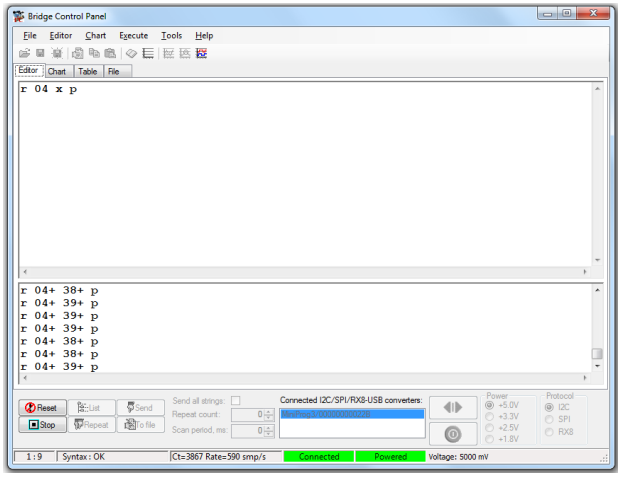
   w 04 02 p

   该写入指令将设置 EzI2Cs 中的辅助地址。任何其他读取操作都发生在辅助地址‘0x0’中，该辅助地址中存有 ADC 值。

   然后请键入以下指令，然后选中 Repeat 按键。只观察读取的 ADC 值。

   r 04 x p

图 32. 仅读 ADC 寄存器的值



使用一个外部 MCU 进行测试

也可以通过将其他微控制器作为 I²C 主设备来测试该项目。本节通过使用被配置为 I²C 主设备的 Arduino Duemilanove 板来演示 I²C 接口，并且假设您已熟悉了 Arduino 原型平台。有关该板和 Arduino 平台的更多信息，请访问 www.arduino.cc 网站。想要了解如何将 Arduino 板连接到 PC 以及上载该程序的指导信息，请参见 Arduino 入门。

配置 Arduino 器件，使之具有各个专用引脚上（AIN4、AIN5）的 I²C 主设备接口以及引脚 11 上的 PWM 输出。

将 CY3210 配置为一个 I²C 从设备，该器件具有一个 4 位数字显示屏（四个 LED）、一个模拟输出（DAC）、一个模拟输入（已连接到模拟输入引脚的电位器）以及一个 LCD 显示屏。

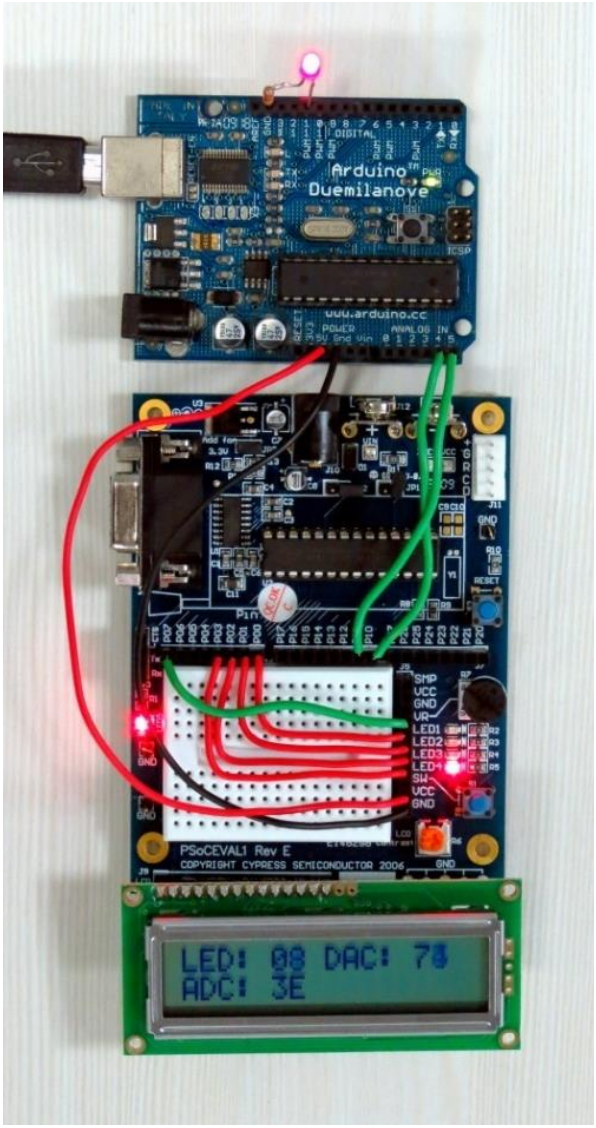
1. 将示例项目编程到 CY3210 评估板上。
2. 使用 Arduino 软件打开所包含的 Arduino 项目文件“Arduino_I2C_Master_PSoC1_Slave.ino”。
3. 在 Tools（工具）选项卡上，请选择“Board”，然后选择“Arduino Duemilanove”选项。
4. 从 Tools 选项卡内，选择“Serial Port”（串行端口）后再选择该板连接上的端口。
5. 使用一个 USB A-B 线缆将程序下载到该板上。编程完成后，拔掉 USB 线缆。
6. 按照表 7 建立硬件连接。

表 7. 硬件连接

Arduino Duemilanove 上的连接		
引脚	**连接**	**说明**
引脚 11（PWM）	连接一个 LED	Arduino 上的 PWM 输出
CY3210-PSoC 评估板上的连接		
引脚	**连接**	**说明**
P00	J5 上的 LED1	数字输出位 0
P01	J5 上的 LED2	数字输出位 1
P02	J5 上的 LED3	数字输出位 2
P03	J5 上的 LED4	数字输出位 3
P05	万用表/示波器	模拟输出（DAC）
P07	J5 上的 VR	模拟输入（ADC）
CY3210 与 Arduino 的连接		
Arduino 的引脚	**PSoC 1 的引脚**	**说明**
Analog in 4（模拟输入 4 - A4）	J7 上的 P10	I²C 的 SDA
Analog in 5（模拟输入 5 - A5）	J7 上的 P11	I²C 的 SCL
5 V	J5 上的 VCC	使用 Arduino 上的 5 V 引脚为 PSoC 1 电路板供电
GND	J5 上的 GND	接地

7. 将 CY3217 MiniProg1 或 CY8CKIT-002 MiniProg3 连接到 CY3210 评估板上的 ISSP 插头。编程完成后，移除 ISSP 插头上的该编程器。
8. 将 USB 线缆重新连接到 Arduino 板上。这样会同时给 Arduino 板和 CY3210 评估板供电。图 33 展示了设置的具体情况。

图 33. Arduino 与 PSoC 连接

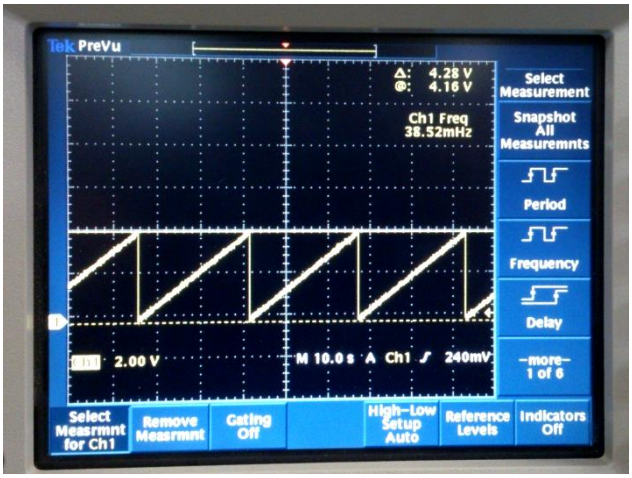


PSoC 读取 P07（已连接到电位器）上的电压，并将该值存储在 ADCValue 中。Arduino 使用了 I²C 来读取该值，并控制着引脚 11 上的 PWM 输出。调试 CY3210 板上的电位器会使 Arduino 板上 LED 的亮度发生变化。

Arduino 通过 I²C 接口连续发送一个 4 位的模块。PSoC 将读取该值，并将它写到各个 Port0 引脚内，这些引脚已经连接了四个 LED。Arduino 还会连续发送一个 8 位的数字值，PSoC DAC 将该值转换为模拟电压。被发送的 DAC 代码是一个递增值，其取值范围为 0 到 255。图 34 显示的是使用示波器观察到的 PSoC 引脚（P05）上 DAC 输出的波形。

该示例说明了一个外部 MCU 如何通过使用 I²C 来连接到 PSoC 器件。

图 34. P05 上的 DAC 输出波形



I²CHW 从设备示例项目

该项目演示了如何将 I²CHW 配置为从设备。

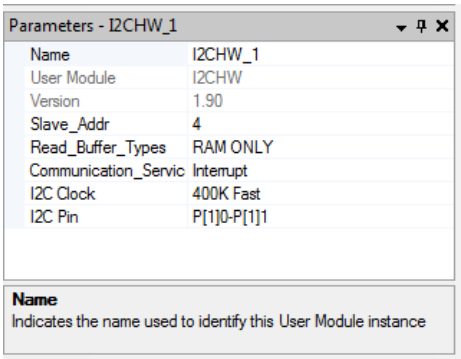
在该项目中，将写入到 I²CHW 从设备的数据回送给主设备。将读取与写入缓冲区配置为同一缓冲区，即可实现该操作。具体操作，请使用下面的 API：

```
/*When master writes data it will write to
rxtxBuffer*/ I2CHW_1_InitWrite(rxtxBuffer,
10);
/*When master reads data it will read from
rxtxBuffer*/
I2CHW_1_InitRamRead(rxtxBuffer, 10);
```

主代码检查主设备是否已经对 I²CHW 从设备进行读/写操作。如果进行了该操作，代码将重置缓冲区，并清除相应的状态标志。

图 35 显示了 I²CHW UM 的配置情况。器件的从设备地址为 0x04。

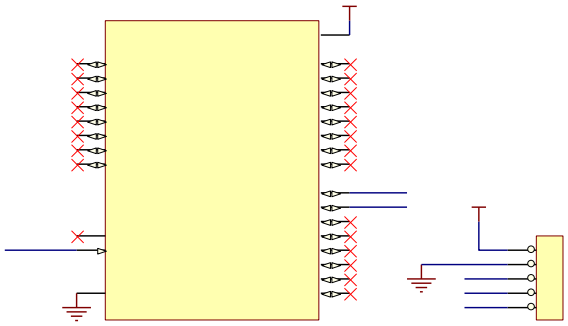
图 35. I²CHW UM 从设备的配置



测试项目

图 36 显示了项目测试所需的设置。

图 36. I²CHW_Slave 项目的原理图



测试该项目时只需要 ISSP 连接。同样可以使用 ISSP 连接器建立同 I²C 主设备的连接。（请参见表 8。）

表 8. 用于测试项目的连接

PSoC 1 的引脚	CY3210 的连接	说明
–	ISSP 插头（J11）	用于进行编程的 MiniProg1 或 MiniProg3 连接
–	ISSP 插头（J11）	连接 Cy3240 或 MiniProg3，以通过 I²C 与 PC 进行通信。

通过使用 CY3210 PSoC 1 评估板，可以测试 I²CHW 从设备项目。

通过 CY3210 评估板上的 ISSP 插头，可以使用 CY3217 MiniProg1 或 CY8CKIT-002 MiniProg3 编程器件。

将 CY3240 USB-I²C 桥接器或 MiniProg3 作为 I²C 主设备使用。

图 37 显示的是如何使用桥接控制面板对 I²CHW 从设备进行读/写操作。欲了解设置桥接控制面板的简要说明，请参见 EzI2Cs_ADC_LED_DAC 示例项目。

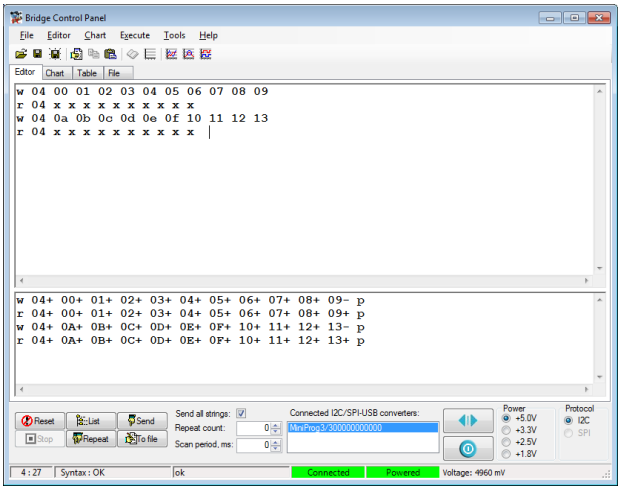
1. 在桥接控制面板的指令窗口中，请键入以下指令：

   ```
   w 04 00 01 02 03 04 05 06 07 08 09 p

   r 04 x x x x x x x x x x p

   w 04 0a 0b 0c 0d 0e 0f 10 11 12 13

   r 04 x x x x x x x x x x p
   ```

2. 选择“Send all strings”（发送所有字符串）选项，然后点击“Send”按键，实现依次执行所有指令。

将在结果窗口中观察到：执行读指令时，使用写指令写入的相同值均被回读。

图 37. I²CHW 从设备的写入和读取操作

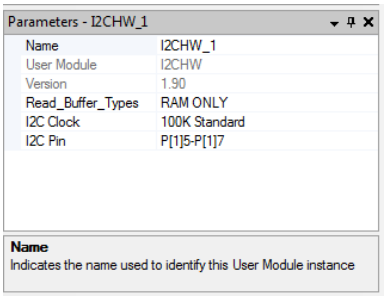


I²CHW 主设备示例项目

该项目演示了如何在主设备模式下使用 I²CHW UM。并且它还专门演示了如何使用 I²CHW UM 对外部 I²C EEPROM 进行读/写操作。

图 38 显示了 I²CHW UM 的配置情况。

图 38. I²CHW 主设备的配置



在主代码中，I²CHW UM 被初始化。然后，它从 RAMBuffer 阵列中将 66 个字节写入到 EEPROM 内。前两个字节表示写入到 EEPROM 中的数据所在的辅助地址。该项目将使用页面大小为 64 个字节的 32 Kb EEPROM 进行测试。当写入到一个更小的 EEPROM 时，写入值被限制为特定 EEPROM 的页面大小。

向 EEPROM 中写入数据时，EEPROM 会进入一个写周期。在此期间，从设备不会对任何 I²C 数据操作生成 ACK。只有 EEPROM 完成该写周期后，才能执行下一步操作。为了进行检测，主设备会进入“while”循环，持续发送“启动”指令并检查 ACK 状态。

```
while(!(I2CHW_fSendStart(0x50,
I2CHW_READ)))
{
I2CHW_SendStop();
}
```

从设备生成 ACK 时，主设备退出“while”循环。

接下来，主设备将读取刚刚写入到 EEPROM 中的数据。为了完成上述操作，首先要写入两个辅助地址字节，然后从 EEPROM 中读取 64 字节的数据。随后，主设备会将被读取的数据与被写入的数据进行比较。如果所有 64 个字节均匹配，LED 灯将亮起。

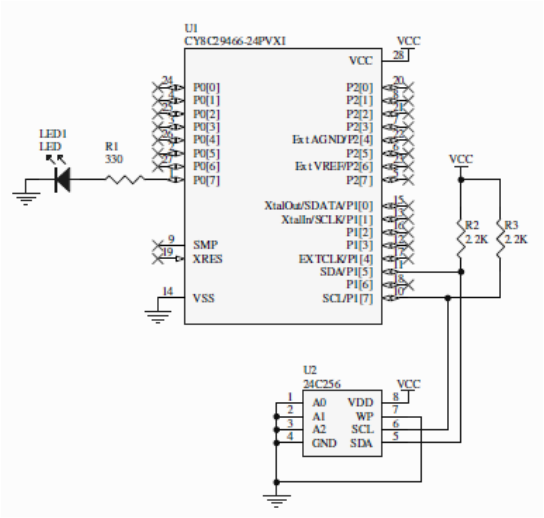
为了测试该项目，需要将 P1.5 和 P1.7 分别连接到外部 I²C EEPROM 的 SDA 和 SCL 线路。请确保在这些线路上已经安装了外部上拉电阻。将带有串行电阻的 LED 连接至 P0_7。可以使用一个 CY3210 PSoC 评估板测试该项目。（请参见表 9。）

表 9. 用于测试项目的连接

PSoC 1 的引脚	CY3210 的连接	说明
P1[5]	2.2 kΩ 的电阻被上拉到 V_{DD}	连接到 24C256 IC 的 SDA
P1[7]	2.2 kΩ 的电阻被上拉到 V_{DD}	连接到 24C256 IC 的 SCL
P0[7]	LED1	LED 指示灯

图 39 显示了项目测试所需的设置

图 39. I2CHW_Master 测试原理图

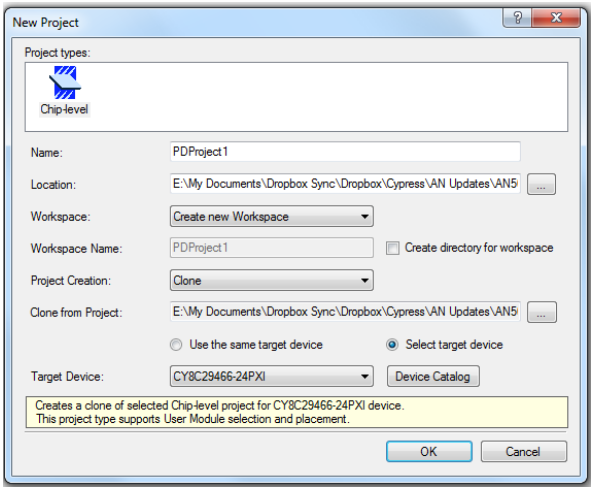


移植示例项目

利用 PSoC Designer 的内置复制功能，可轻松将这些项目移植到其他器件中。例如，要想移植 EzI2Cs_ADC_LED_DAC 项目，请执行下面操作：

1. 打开 PSoC Designer，创建一个新项目。
2. 在 Project Creation（项目创建）参数中，选择 Clone（复制）项。
3. 在“Clone from Project”（从项目中复制）参数中，浏览到 EzI2Cs_ADC_LED_DAC 项目的项目文件夹，然后选择 EzI2Cs_ADC_LED_DAC.cmx 文件。接着，点击 Device Catalog（器件目录）按键，选择要运行的器件。
4. 选择器件后，请点击 OK；PSoC Designer 会将该项目移植到新的器件内。（请参见图 40。）

图 40. 移植项目



文档修订记录

文档标题：AN50987 — PSoC® 1 I²C 入门

文档编号：001-82514

版本	ECN	变更者	提交日期	变更说明
**	3732096	QDGU	09/03/2012	本文档版本号为 Rev**，译自英文版 001-50987 Rev*C。
*A	4592688	QDGU	12/09/2014	本文档版本号为 Rev*A，译自英文版 001-50987 Rev*D。
*B	4992976	RING	11/02/2015	本文档版本号为 Rev*B，译自英文版 001-50987 Rev*E。
*C	6065079	XITO	02/09/2018	本文档版本号为 Rev. *C，译自英文版 001-50987 Rev. *F。

销售、解决方案以及法律信息

全球销售和设计支持

赛普拉斯公司具有一个由办事处、解决方案中心、厂商代表和经销商组成的全球性网络。要想查找离您最近的办事处，请访问赛普拉斯所在地。

产品

Arm® Cortex® 微控制器	cypress.com/arm
汽车级产品	cypress.com/automotive
时钟与缓冲器	cypress.com/clocks
接口	cypress.com/interface
物联网	cypress.com/iot
存储器	cypress.com/memory
微控制器	cypress.com/mcu
PSoC	cypress.com/psoc
电源管理 IC	cypress.com/pmic
触摸感应	cypress.com/touch
USB 控制器	cypress.com/usb
无线连接	cypress.com/wireless

PSoC®解决方案

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

赛普拉斯开发者社区

社区 | 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

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。

赛普拉斯半导体公司
198 Champion Court
San Jose, CA 95134-1709

© 赛普拉斯半导体公司，2012-2018 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。没有任何电子设备是绝对安全的。因此，尽管赛普拉斯在其硬件和软件产品中采取了必要的安全措施，但是赛普拉斯并不承担任何由于使用赛普拉斯产品而引起的安全问题及安全漏洞的责任，例如未经授权的访问或使用赛普拉斯产品。此外，本材料中所介绍的赛普拉斯产品有可能存在设计缺陷或设计错误，从而导致产品的性能与公布的规 格不一致。（如果发现此类问题，赛普拉斯会提供勘误表）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。