

PSoC® 4 – 使用 GPIO 引脚

作者: **Rajiv Badiger**

相关项目: 有

相关器件系列: **PSoC 4**软件版本: **PSoC Creator™ 4.2 或更高版本**相关应用笔记: 要获取完整的应用笔记列表, 请点击[此处](#)。

更多代码示例? 我们明白。

如需寻找包含上百 PSoC 代码示例并有不断更新的网上资源, 请浏览我们的[代码示例网页](#)。您还可以在[此处](#)观看 PSoC 4 视频库。

AN86439 通过各种使用示例介绍了如何有效使用 PSoC® 4 GPIO 引脚并阐述了它们的各项功能。本文档的主要主题包括 GPIO 基础、设置选项、混合信号使用、中断和以及低功耗的特性。

目录

1 简介	2	6 GPIO 提示和技巧	34
2 PSoC 资源	2	6.1 切换 LED	35
2.1 PSoC Creator	3	6.2 读取输入并写入输出	36
2.2 代码示例	4	6.3 使用数字逻辑门驱动输出信号	37
2.3 PSoC Creator 帮助	5	6.4 使用双向引脚	38
2.4 技术支持	5	6.5 GPIO 输入/输出同步设置	39
3 GPIO 引脚的基本知识	6	6.6 通过数据寄存器能更快切换 GPIO	44
3.1 GPIO 引脚的物理结构	6	6.7 配置 GPIO 输出使能逻辑	47
3.2 引脚路由情况	9	6.8 引脚中断	49
3.3 启动和低功耗操作	19	6.9 使用固件配置 GPIO 中断设置	51
3.4 GPIO 中断	20	6.10 在 GPIO 上同时使用模拟和数字功能	53
4 过压容差 (OVT) 引脚	22	6.11 组合引脚以获得更大的源电流/灌电流	56
5 PSoC Creator 的 GPIO 引脚	22	6.12 深度睡眠模式下的控制寄存器处理	59
5.1 引脚组件符号	22	7 相关应用笔记	62
5.2 引脚组件自定义程序	23	8 总结	62
5.3 引脚组件中断	25	9 关于作者	62
5.4 引脚的手动分配	28	附录 A. PSoC 4 GPIO 与 PSoC 1、PSoC 3 和 PSoC 5LP GPIO 相比较	63
5.5 PSoC Creator API	28	附录 B. PSoC 4 开发板	64
5.6 GPIO 引脚上的调试逻辑	29	文档修订记录	65
5.7 添加多个 GPIO 引脚, 作为逻辑端口	29		
5.8 表示片外组件	32		

1 简介

PSoC 具有灵活的通用 I/O (GPIO) 引脚架构；与传统 MCU 相比，它能提供更多功能。在 PSoC 中，不仅可以通过固件对寄存器进行配置来控制 GPIO（类似于传统的 MCU），还可以通过自定义数字逻辑和模拟模块的信号驱动它们。本应用笔记介绍了 PSoC 4 的 GPIO 引脚的基本知识，并显示了如何将这些引脚有效地配置为不同功能。

本应用笔记假设您已经熟悉了 PSoC Creator™ 和 PSoC 4 架构。如果您对 PSoC 4 还不太熟悉，请参见 [AN79953 — PSoC 4 入门](#) 中的内容。如果您尚未了解 PSoC Creator，请参考 [PSoC Creator 主页](#)。更多有关器件封装或 GPIO 规范的信息，请参见 [PSoC 4 数据手册](#)。如果已经了解器件¹和 PSoC Creator，您可以跳转到 [GPIO 提示和技巧](#) 章节。

¹**注意：**除非另有说明，否则所谓‘PSoC’或‘器件’此后指的是 PSoC 4。

2 PSoC 资源

在赛普拉斯网站 www.cypress.com 上提供了大量数据，有助您正确选择 PSoC 器件来进行设计，从而使您能够快速并有效地将器件集成到设计中。有关完整的资源列表，请参考 [KBA86521 — 如何使用 PSoC 3、PSoC 4 和 PSoC 5LP 进行设计](#)。下面提供了 PSoC 4 的简要列表：

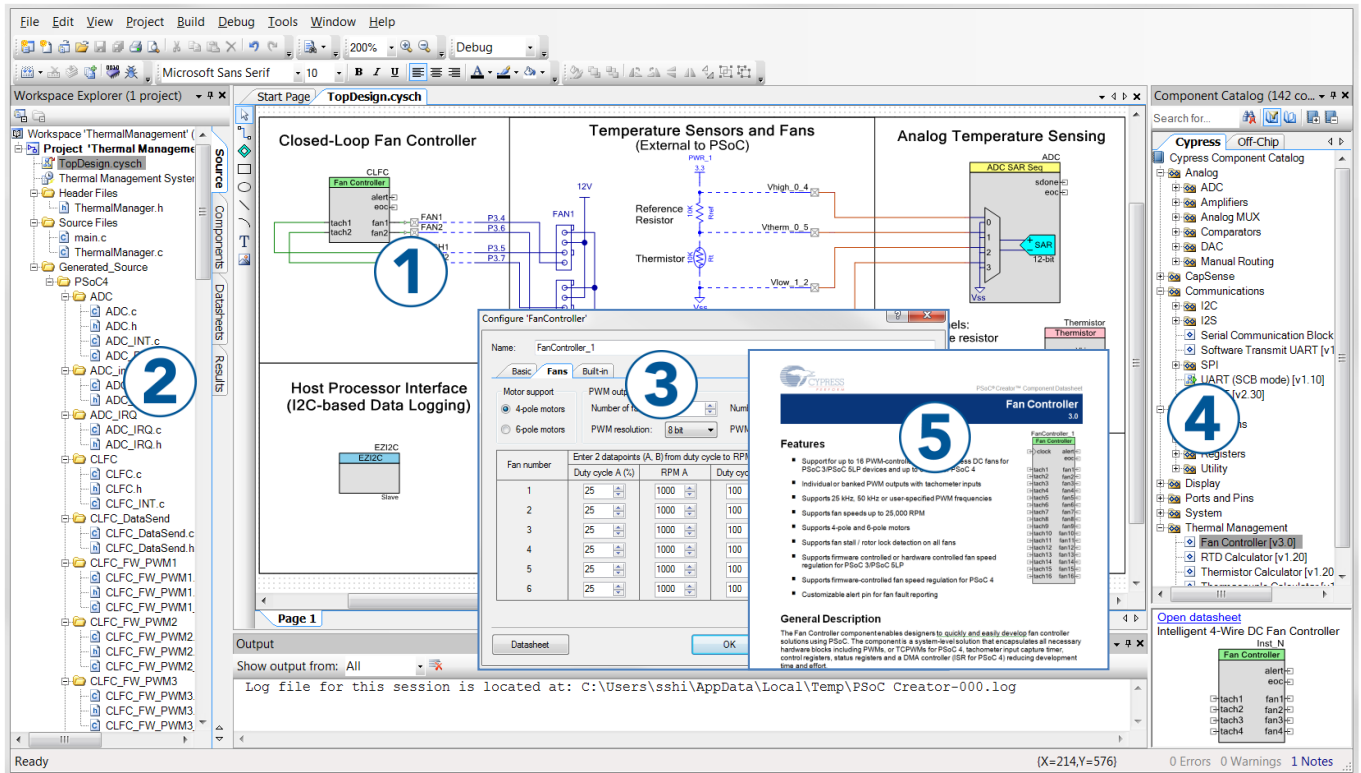
- **概况：** [PSoC 产品系列](#)、[PSoC 产品路线图](#)
- **产品选型器：** [PSoC 1](#)、[PSoC 3](#)、[PSoC 4](#)、[PSoC 5LP](#)。此外，[PSoC Creator](#) 还包含一个器件选择工具。
- **数据手册** 说明并提供了适用于 PSoC 3、PSoC 4 和 PSoC 5LP 器件系列的电气规格。
- **CapSense®设计指南：** 了解如何使用 [PSoC 3](#)、[PSoC 4](#)、[PSoC 5LP](#) 系列的资源来设计电容式触摸感应的应用。
- **应用笔记和代码示例：** 包含了从基本到高级的广泛主题。许多应用笔记还提供了代码示例。
- **技术参考手册 (TRM)：** 对 PSoC 3、PSoC 4 和 PSoC 5LP 器件系列中所使用的架构和寄存器进行了详细说明。
- **开发套件：**
 - [CY8CKIT-040](#)、[CY8CKIT-041](#)、[CY8CKIT-042](#)、[CY8CKIT-042-BLE](#)、[CY8CKIT-044](#) 以及 [CY8CKIT-046](#) 等 Pioneer 套件均为易于使用且廉价的开发平台。这些套件包括用于 [Arduino™](#) 兼容屏蔽和 [Digilent® Pmod™](#) 子卡的连接器。
 - [CY8CKIT-043](#)、[CY8CKIT-049](#)、[CY8CKIT-145](#) 和 [CY8CKIT-149](#) 都是成本非常低的原型平台，用于选择 PSoC 4 器件。[CY8CKIT-001](#) 是所有 PSoC 器件系列经常使用的开发平台。
- [MiniProg3](#) 器件提供一个用于进行闪存编程和调试的接口。

2.1 PSoC Creator

PSoC Creator 是一个基于 Windows 的免费集成开发环境 (IDE)。通过它可以同时对 PSoC 3、PSoC 4 和 PSoC 5LP 器件进行硬件和固件设计。如图 1 所示：通过 PSoC Creator，您可以进行以下操作：

1. 将组件图标拖放到主要设计工作区中，以进行您的硬件系统设计。
2. 对您的应用固件和 PSoC 硬件进行协同设计。
3. 使用配置工具配置各组件。
4. 了解包含一百多个组件的库。
5. 查看组件数据手册

图 1. PSoC Creator 特性



2.2 代码示例

PSoC Creator 包含了多个代码示例项目。可以从 PSoC Creator 的“Start Page”（起始页）上获取这些项目，如图 2 所示。

这些示例项目为您提供完整的设计（并非一个空白页），从而可以加快您的设计过程。示例项目还介绍了如何将 PSoC Creator 组件使用于不同应用中。此外，还包含了多个代码示例和数据手册，如图 3 所示。

在图 3 所示的 **Find Example Project**（查找示例项目）对话框中，您可以选择以下选项：

- 根据 **device family**（器件系列）（例如：PSoC 3、PSoC 4 或 PSoC 5LP）、**category**（类型）或 **keyword**（关键词）等选项对示例进行筛选
- 从 **Filter Options**（滤波选项）的示例菜单中选择。
- 通过 **Documentation**（文档）选项卡，查看选出的数据手册。
- 查看所选的代码示例。您可以复制该窗口中的代码，然后将其粘贴到您的项目内，从而加快代码的开发过程，或
- 根据已选项目创建一个新的项目（若需要，可添加新的工作区）。通过为您提供一个完整的基本设计，它可以加快您的设计进程。然后，您可以根据自己的应用来调整该设计。

图 2. PSoC Creator 中的代码示例

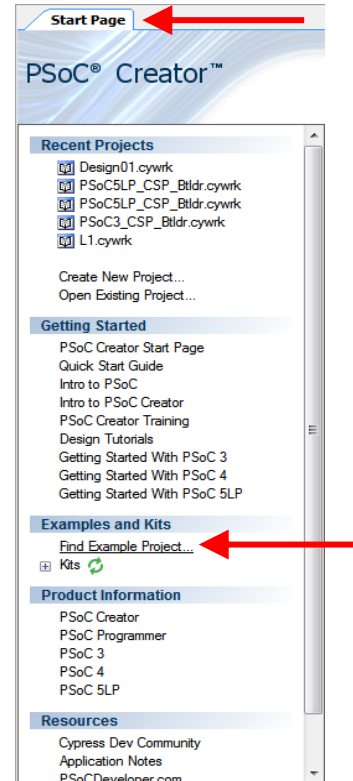
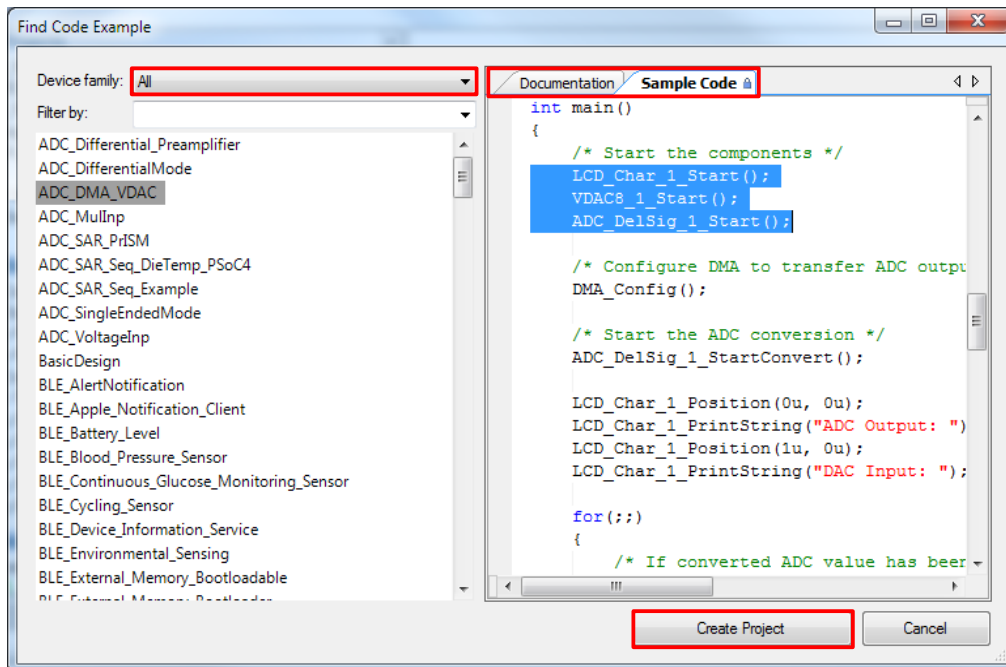


图 3. 带样本代码的示例项目



2.3 PSoC Creator 帮助

请访问 [PSoC Creator 主页](#) 下载 PSoC Creator 的最新版本。启动 PSoC Creator，并导航到下列各项：

- **快速入门指南：**依次选择 **Help > Documentation > Quick Start Guide**。本指南提供了开发 PSoC Creator 项目的基本知识。
- **系统参考指南：**依次选择 **Help > System Reference Guides**。该指南列出并描述了 PSoC Creator 所提供的系统功能。
- **组件数据手册：**右击组件，然后选择“Open Datasheet”项。请访问 [PSoC 4 组件的数据手册](#) 网页，获取所有 PSoC 4 组件的数据手册列表。
- **文档管理工具：**PSoC Creator 提供了文档管理工具，有助您查找和查看文件资源。要想打开文档管理工具，请选择菜单项 **Help > Document Manager**。

2.4 技术支持

若有任何疑问，我们的技术支持团队很乐意为您提供帮助。您可以在[赛普拉斯技术支持](#)页面上创建一个技术支持请求。

如果您在美国，可以通过拨打我们的免费电话，直接与技术支持团队联系：**+1-800-541-4736**。选择提示符处的第 8 项。

若想快速获得支持，您同样可以使用下面的支持资源。

- [自助](#)
- [所在地销售办事处](#)

3 GPIO 引脚的基本知识

PSoC 的 GPIO 引脚提供以下功能：

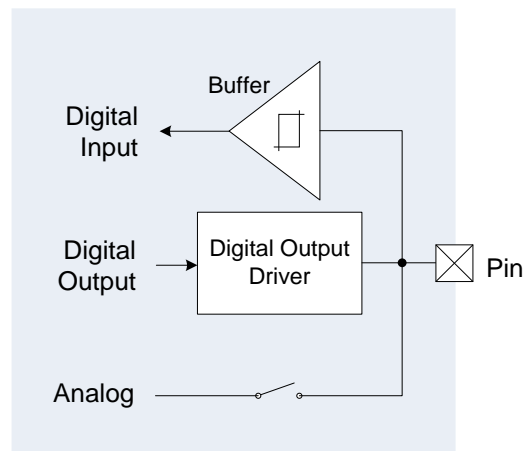
- 模拟和数字输入和输出功能
- 支持 LCD segment 驱动（PSoC 4000 不支持这项功能）
- 支持 CapSense®功能
- 可在上升沿、下降沿或双边沿上触发中断
- 转换速率控制
- 可以选择输入阈值（CMOS / LVTTL / 1.8 V CMOS）
- 具有热插拔功能的过压容差引脚（仅限于 PSoC 4 BLE、PSoC 4 M 系列以及 PSoC 4 L 系列）

GPIO 功能取决于 PSoC 4 器件中可用的外设。有关各种 PSoC 4 系列所支持的特性比较情况，请参见 [AN79953 — PSoC 4 入门应用笔记](#) 中的表 1。

3.1 GPIO 引脚的物理结构

图 4 显示的是 PSoC 器件中引脚与资源之间的连接。

图 4. 简化的 GPIO 框图

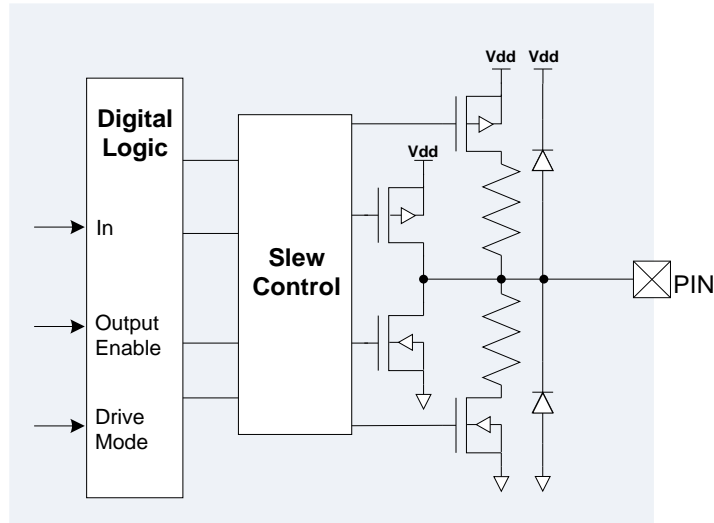


欲了解 GPIO 结构的模块框图的详细信息，请参阅 [PSoC 4 架构技术参考手册](#) 中“[I/O 系统](#)”章节。每个引脚都可以作为 CPU 和数字外设（如定时器、PWM 或 I²C 等）的出入或输出。它也可以作为运算放大器和 ADC 的模拟引脚使用。你可以根据需要将引脚作为数字输入、数字输出，模拟引脚或者这三种引脚的组合使用。例如，如果您同时使能了数字输出和输入，则可以得到一个数字双向引脚。数字输入缓冲器为外部输入提供了高阻抗。可以将该输入阈值配置为 CMOS、LVTTL 和 1.8 V CMOS。除 PSoC 4100/4200 和 PSoC 4000 外，1.8-V CMOS 模式可用于所有 PSoC 4 器件。

请参见 [器件数据手册](#)，了解该输入阈值。

数字输出驱动器支持多种驱动模式和转换速率控制（请参见图 5）。

图 5. 数字输出驱动器



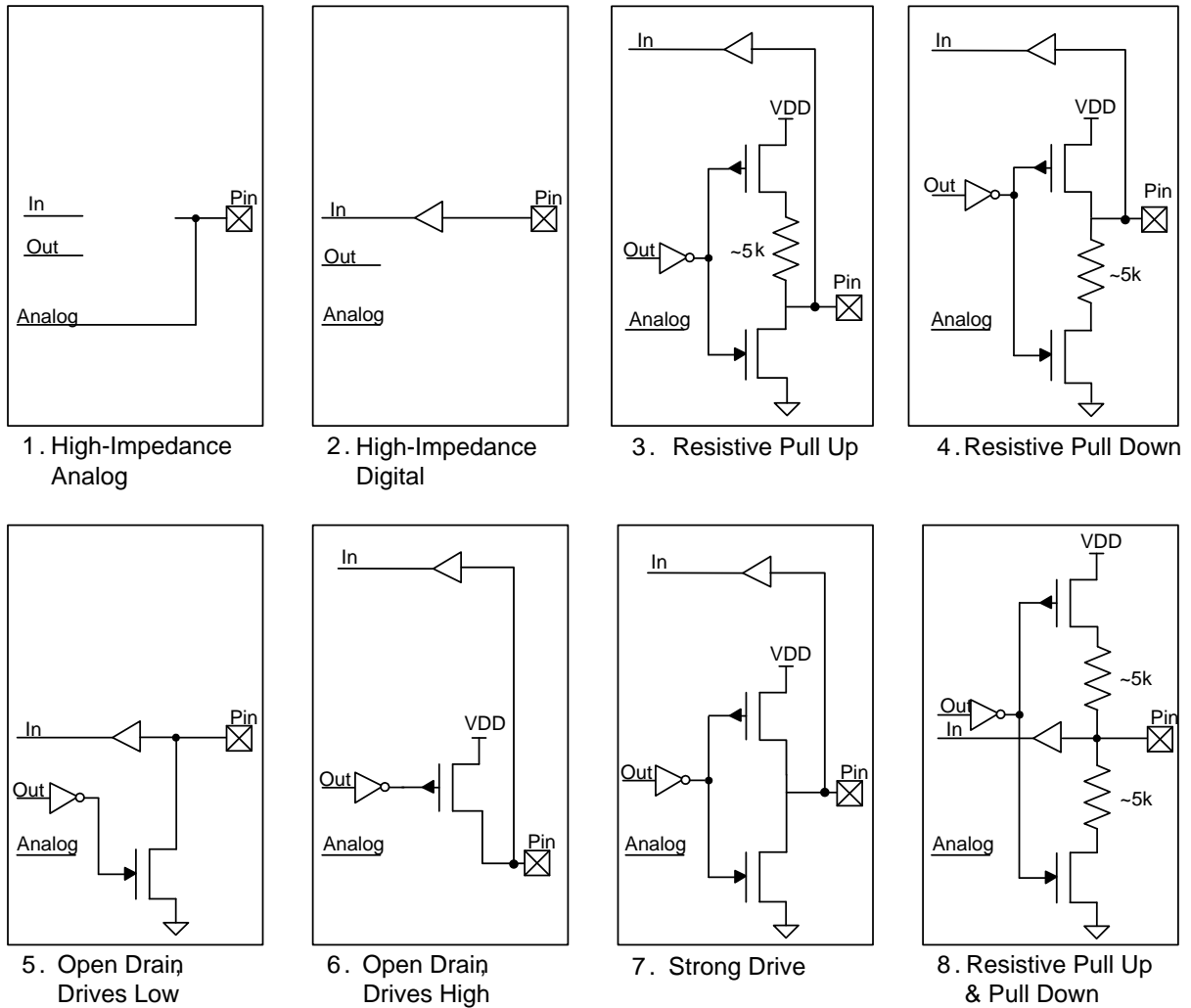
通过转换速率控制功能，可降低 EMI 和串扰信号。转换速率有两个选项，分别为快速和慢速。转换速率默认情况下被设置为快速。当信号对速度没有严格要求时，可以使用慢速选项。

图 5 中显示的电路支持 PSoC 4 的 8 种驱动模式，如表 1 所列。

表 1. 驱动模式和应用

序号	驱动模式	应用示例
1	模拟高阻态	模拟输入/输出
2	数字高阻态	数字输入
3	电阻上拉 (~5 kΩ)	用于连接一个开漏低电平输入（如来自马达的转速计输出）或连接一个与地相连的开关。还可以用于驱动 LED。
4	电阻下拉 (~5 kΩ)	用于连接开漏高电平输入，或连接与 VDD 相连的开关。此外，还可以将其作为一个输出，用于连接灌电流模式下的各个 LED。
5	漏极开路，驱动低电平	在高电平状态中提供高阻抗，且在低电平状态中提供强驱动；这种配置适用于 I²C 引脚。该模式可以与外部上拉电阻结合使用。
6	漏极开路，驱动高电平	在高电平状态中提供强驱动，而在低电平状态中则提供高阻抗。该模式可以与外部下拉电阻结合使用。
7	强驱动	在低电平和高电平状态中提供 CMOS 输出驱动。
8	电阻上拉和电阻下拉 (~5 kΩ)	在高电平和低电平状态下添加了串联电阻。

图 6. 驱动模式



注意 1: 上拉和下拉驱动模式下的电阻值（如图 6 所示）是近似值；请参见[器件数据手册](#)，了解电阻值规范。如果要求准确度更高，请使用外部电阻。在这种情况下，必须将引脚配置为开漏高电平驱动模式或开漏低电平驱动模式。

注意 2: 任何时候都要避免器件 VDD 电源通过 ESD 二极管从引脚的外部获取电压。如果不给 PSoC 4 器件供电，并且通过 GPIO 提供了外部电压，或者 GPIO 的外部电压大于器件的 VDD 时，会发生这种现象。但由于[过压容差 \(OVT\)](#)引脚上没有钳位二极管，因此在这些引脚上不会发生这种现象。

3.2 引脚路由情况

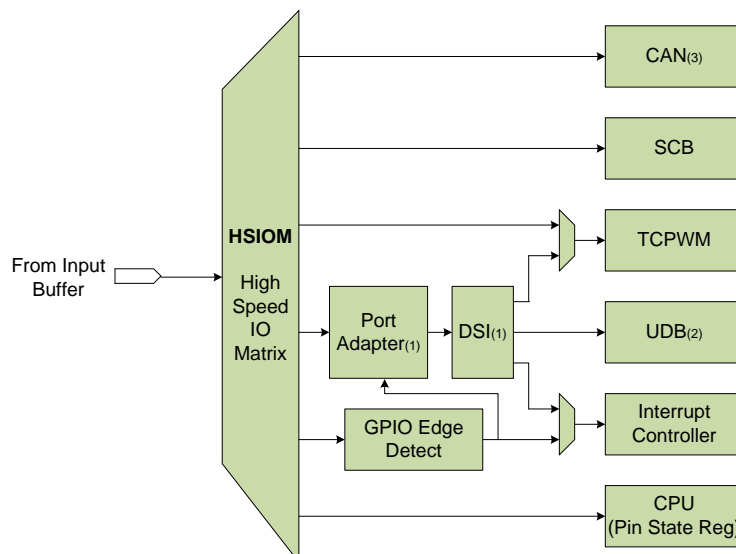
3.2.1 数字路由

可以将某个引脚路由到多个不同的数字外设，如：通用数字模块（**UDB**）、串行通信模块（**SCB**）、定时器/计数器/脉冲宽度调制器（**TCPWM**）模块、**LCD** 驱动器、**CAN** 模块、中断控制器以及由 **CPU** 读/写的寄存器。图 7 和图 8 分别显示了输入引脚和输出引脚的路由方法。

通过使用高速 I/O 矩阵（**HSIOM**），可以建立外设和引脚之间的连接，如图 7 和图 8 所示。该矩阵复用了各个外设的信号，以便连接到某个特定引脚。

在 **PSoC** 中，存在下面两种路由方法：通过 **HSIOM** 实现专用 I/O 的路由；以及使用数字系统互联（**DSI**）灵活进行路由。**DSI** 不仅可以用于将外设输入和输出路由到各个引脚上，还可以用于在数字资源之间进行信号路由。通过端口适配器，可将 **HSIOM** 与 **DSI** 相连。另外，它还提供了硬件用于将引脚输入和输出的信号进行同步。

图 7. 数字引脚输入路径

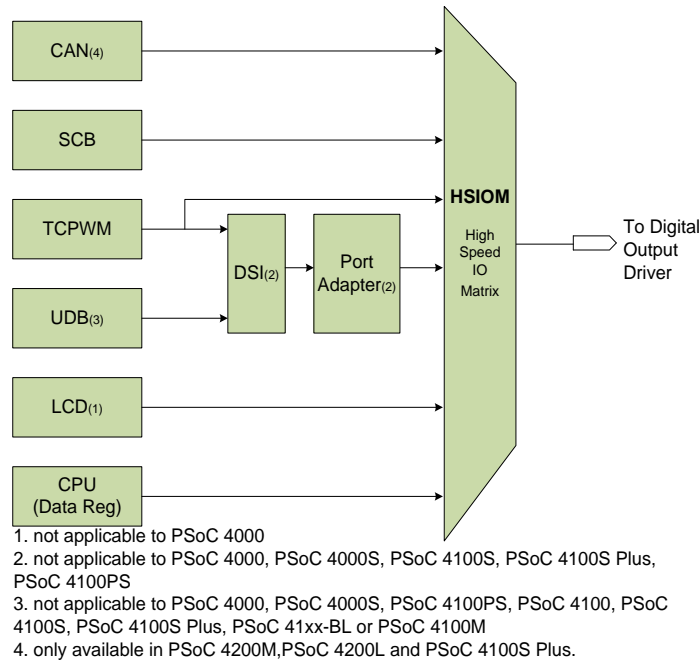


1. not applicable to PSoC 4000, PSoC 4000S, PSoC 4100S, 4100S Plus, PSoC 4100PS
2. not applicable to PSoC 4000, PSoC 4000S, PSoC 4100PS, PSoC 4100, PSoC 4100S, PSoC 4100S Plus, PSoC 41xx-BL or PSoC 4100M
3. only available in PSoC 4200M, PSoC 4200L and 4100S Plus devices

SCB（如：**I²C**、**UART** 和 **SPI**）和 **TCPWM** 路由到一些特定 I/O。对于 **UDB** 输入和输出，引脚上产生的中断，甚至是 **TCPWM** 信号，都可以灵活进行路由。在 **PSoC** 器件系列（除 **PSoC 4000** 外）的所有 I/O 引脚上都有 **LCD** 驱动器，其中任意 I/O 引脚均作为 **LCD** 显示屏的 **segment** 或 **common** 驱动器使用。

GPIO 边沿检测模块支持在上升沿、下降沿或双边沿上触发中断。有关详细信息，请参考 [GPIO 中断](#) 一节。

图 8. 数字引脚输出路径



注意： PSoC 4 具有多个端口，每个端口最多有 8 个引脚。在 PSoC 4200L 器件中，端口 7 到端口 10 上的引脚没有端口适配器；在其他器件中，端口 4 和更高端口上没有端口适配器。这些端口受到以下限制：

- 不能通过 DSI 路由。因此，基于 UDB 的数字信号不能路由到这些端口的引脚上。
- 不适用于模拟模块，如：SAR ADC、运算放大器-微型连续时间模块（CTBm）、低功耗比较器（仅限于 PSoC 4100、PSoC 4100PS 和 PSoC 4200）以及连续时间模块（仅限于 PSoC 4100PS）
- 无输入/输出同步

但是，这些端口在下面各种情况下很有用：

- 作为由固件控制的 GPIO 引脚使用
- 直接连接到 TCPWM、SCB 或 CAN
- 作为 LCD 和 CapSense 引脚使用
- 中断生成

注意： 分配 PSoC 器件的引脚，以专门用于连接到不同外设。为了了解每个引脚的功能，请参见相应器件数据手册中的“引脚布局”一节。

3.2.2 模拟路由

通过直接连接或通过模拟开关和模拟复用（AMUX）总线，可以将高阻抗模拟（HI-Z）模式中配置的 GPIO 引脚连接到模拟资源，如图 9 到图 14 所示。

以下是 PSoC 4000 器件的模拟走线的要点，如图 9 所示：

- 所有引脚（端口 3 除外）都可被连接到 AMUX 总线，该连接由固件控制。共有两个总线：AMUXBUS_A 和 AMUXBUS_B。
- CapSense IDAC0 连接到 AMUXBUS_A，IDAC1 则连接到 AMUXBUS_B。
- CapSense CMOD 连接到 P0[4]，Ctank 则连接到 P0[2]。
- 由于 CapSense 模块通过 AMUX 总线连接到传感器，任意引脚（端口 3 除外）均可连接到电容式触摸传感器。

注意：将 CMOD 电容放置在靠近引脚的位置上。有关布局指南的信息，请参见 [AN85951 — PSoC 4 CapSense 设计指南](#)。

以下是其他 PSoC 4 器件的模拟路由的要点，如图 11 到图 15 所示：

- 有两种 AMUX 总线。所有引脚都能连接到 AMUXBUS_A 和 AMUXBUS_B。AMUX 总线连接可由固件控制或通过 DSI 信号控制。请注意：对于端口 4 和更高端口的引脚，DSI 连接不可用，仅能通过固件建立 AMUX 连接。
- 可对运算放大器输入和输出进行直接连接，由于电阻和电容较小，因此这些连接能提供更好的性能。直接连接也可以用于低功耗比较器（LPCOMP）输入，而不需要使用各种开关。
- CapSense CMOD 和 Ctank 也有各自的专用引脚。请参见图 11 到图 15，以了解这些引脚。
- CapSense IDAC0 连接到 AMUXBUS_A，IDAC1 则连接到 AMUXBUS_B。
- 由于 CapSense 模块通过 AMUX 总线连接到传感器，任意引脚均可连接到电容触摸传感器。
- 可以使用蓝色标记的开关使 AMUXBUS_A 和 AMUXBUS_B 总线分开，如图 12 到图 15 所示。当非 CapSense 应用需要使用 AMUX 总线，如进行运算放大器/比较器输入和输出路由（在该系统中，同时使用该运算放大器/比较器和 CapSense）时，该操作很有用。
- 通过 SAR 序列发生器将 SAR ADC 输入连接到：
 - PSoC 4100、PSoC 4100S、PSoC 4200、PSoC 4100S Plus、PSoC 4100M、PSoC 4200M 和 PSoC 4200L 的端口 2
 - PSoC 41xx-BL、PSoC 42xx-BL 和 PSoC 41xxPS 的端口 3
 - CTBm、CTB 输出
 - 温度传感器输出

通过控制图 11 到图 15 中红色显示的开关，可以进行复用。请注意，SAR ADC 也可以通过 AMUXBUS（而不需要使用序列发生器）连接到任何引脚输入。

注意：可将运算放大器输出连接到专用引脚上（无需任何开关）。如果需要连接到 AMUX 总线，则专用引脚相对应的 AMUX 开关被激活。同样，通过 AMUX 总线，可将其他引脚作为运算放大器的输出引脚使用。

注意：当 SAR ADC 的输入在序列发生器模式下为差分输入时，正向输入可以只是偶数引脚，而负向输入则是相邻的奇数引脚。例如，在 PSoC 4200 中，P2[0]和 P2[1]是一对引脚，P2[0]作为正向输入，P2[1]作为负向输入。它们在模拟路由图中使用各个环来标识。

图 9. PSoC 4000 模拟路由图

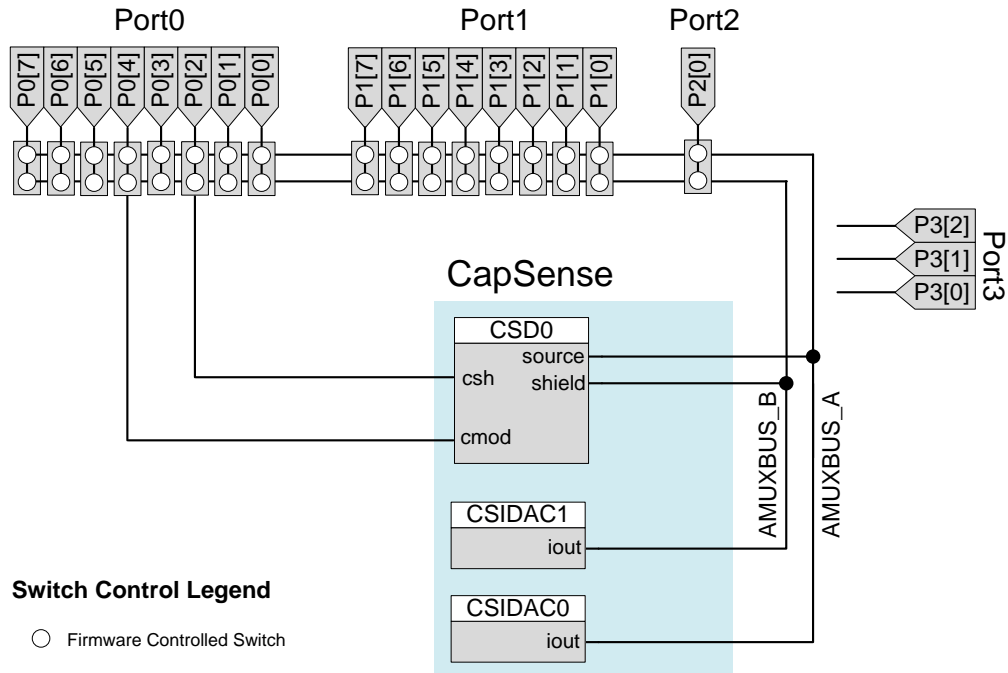


图 10. PSoC 4000S 模拟路由图

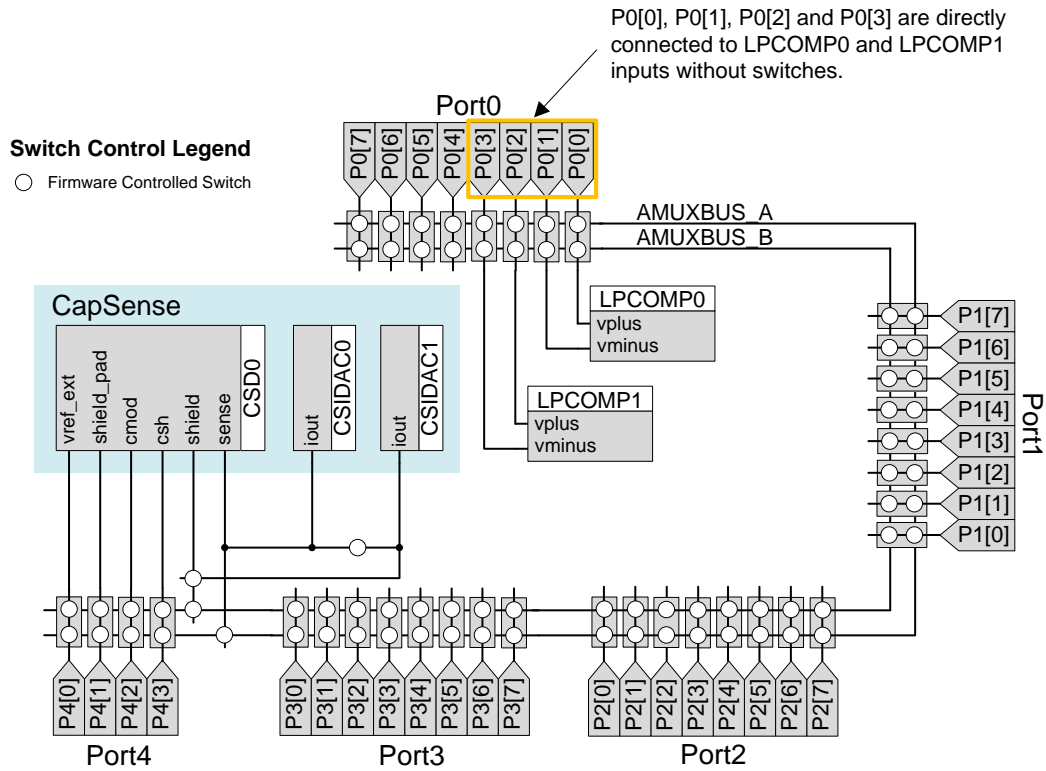


图 11. PSoC 4200/PSoC 4100 模拟路由图

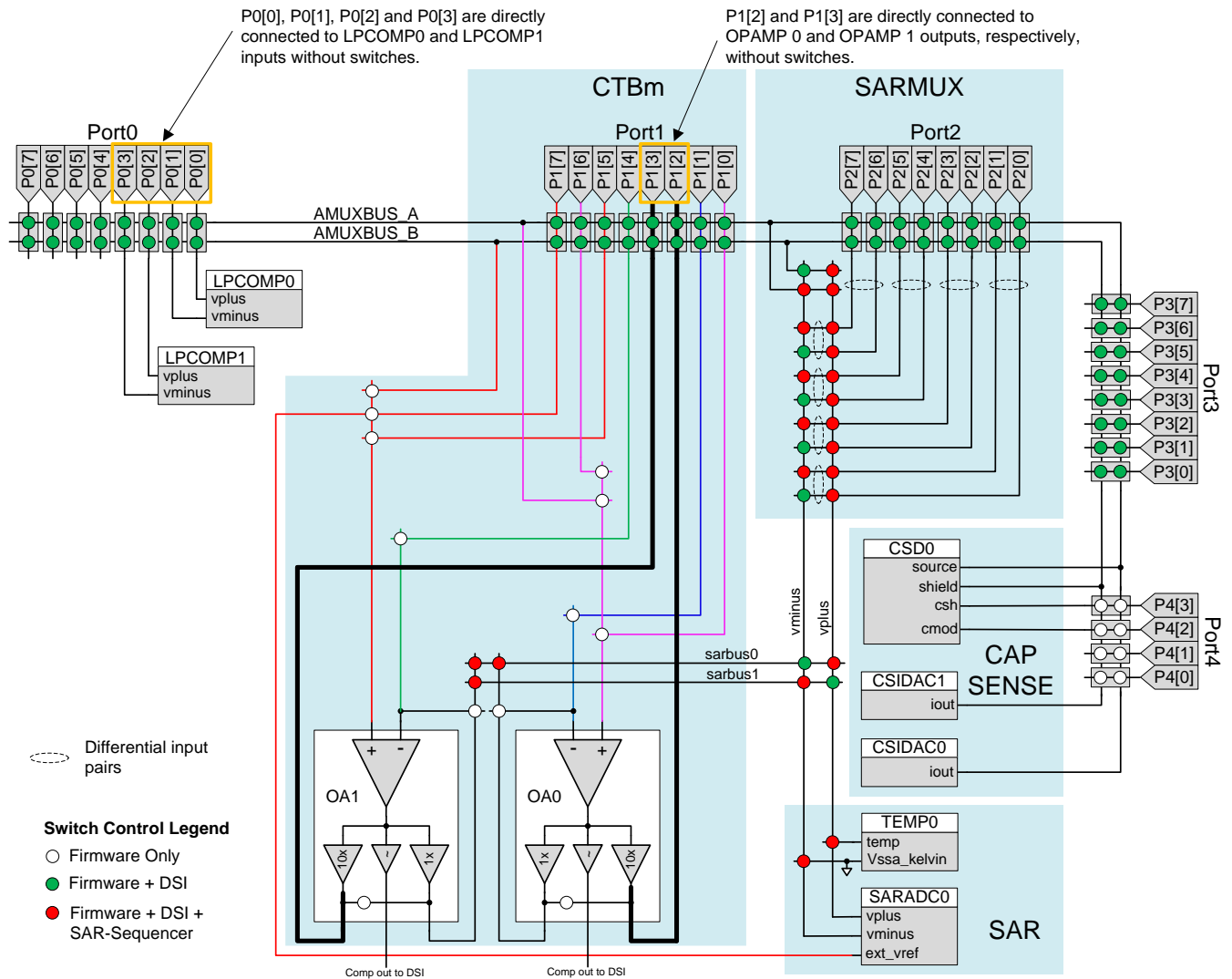
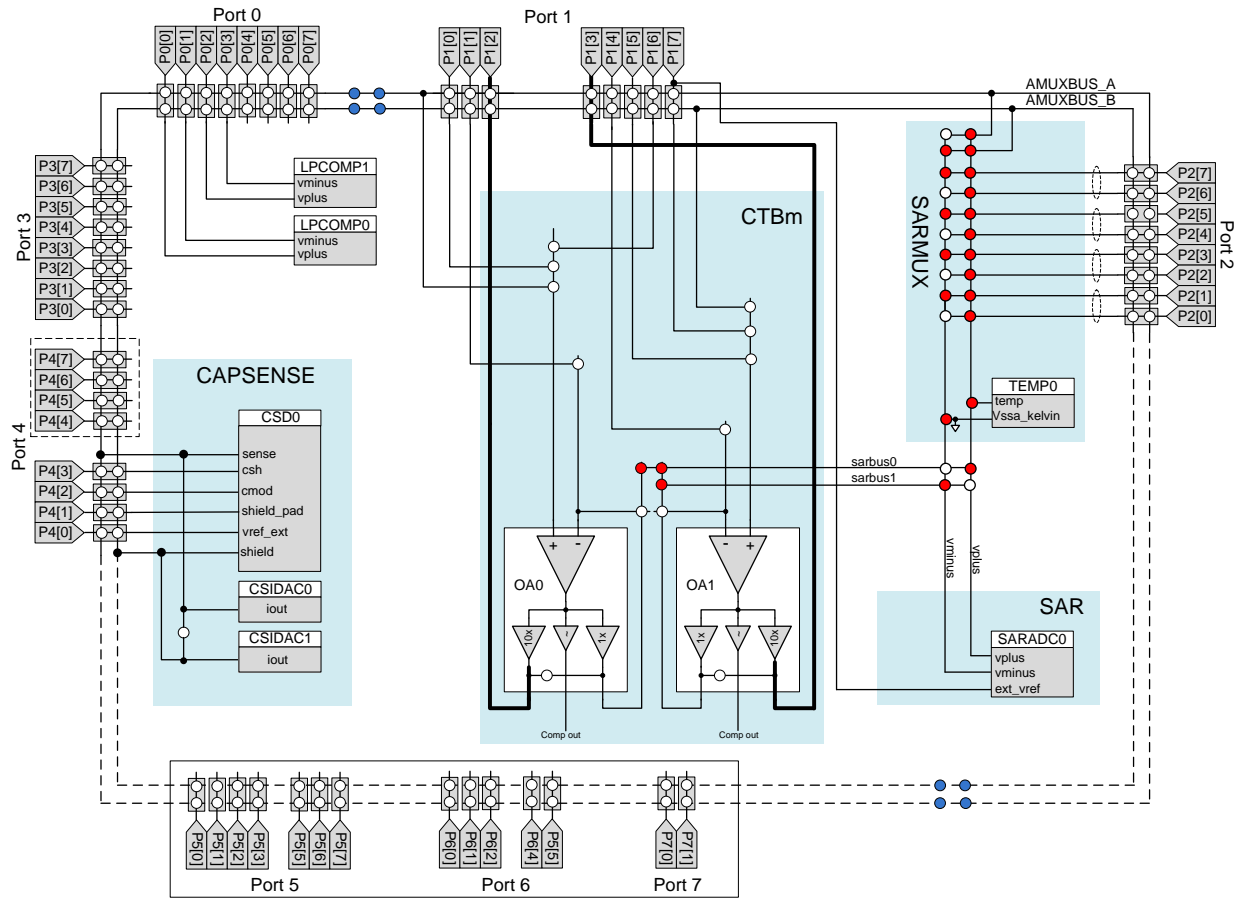


图 12. PSoC 4100S/4100S Plus 模拟路由图



----- Available only in PSoC
4100S Plus Family

Switch Control Legend

- Firmware Only
- Firmware + SAR-Sequencer
- AMUX Splitter (Firmware Only)

图 13. PSoC 41xx-BL/PSoC 42xx-BL 模拟路由图

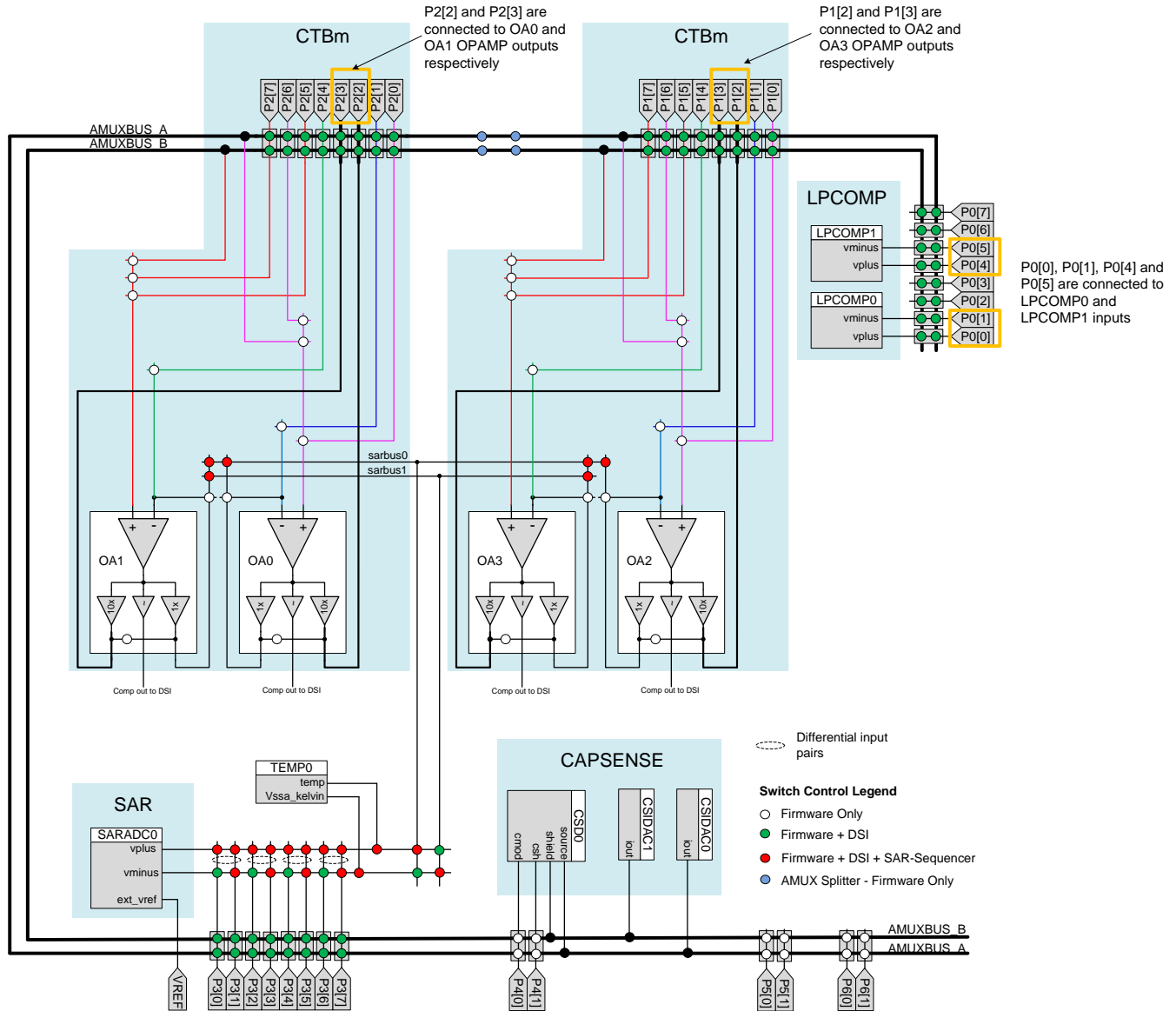


图 14. PSoC 4100M/PSoC 4200M 模拟路由图

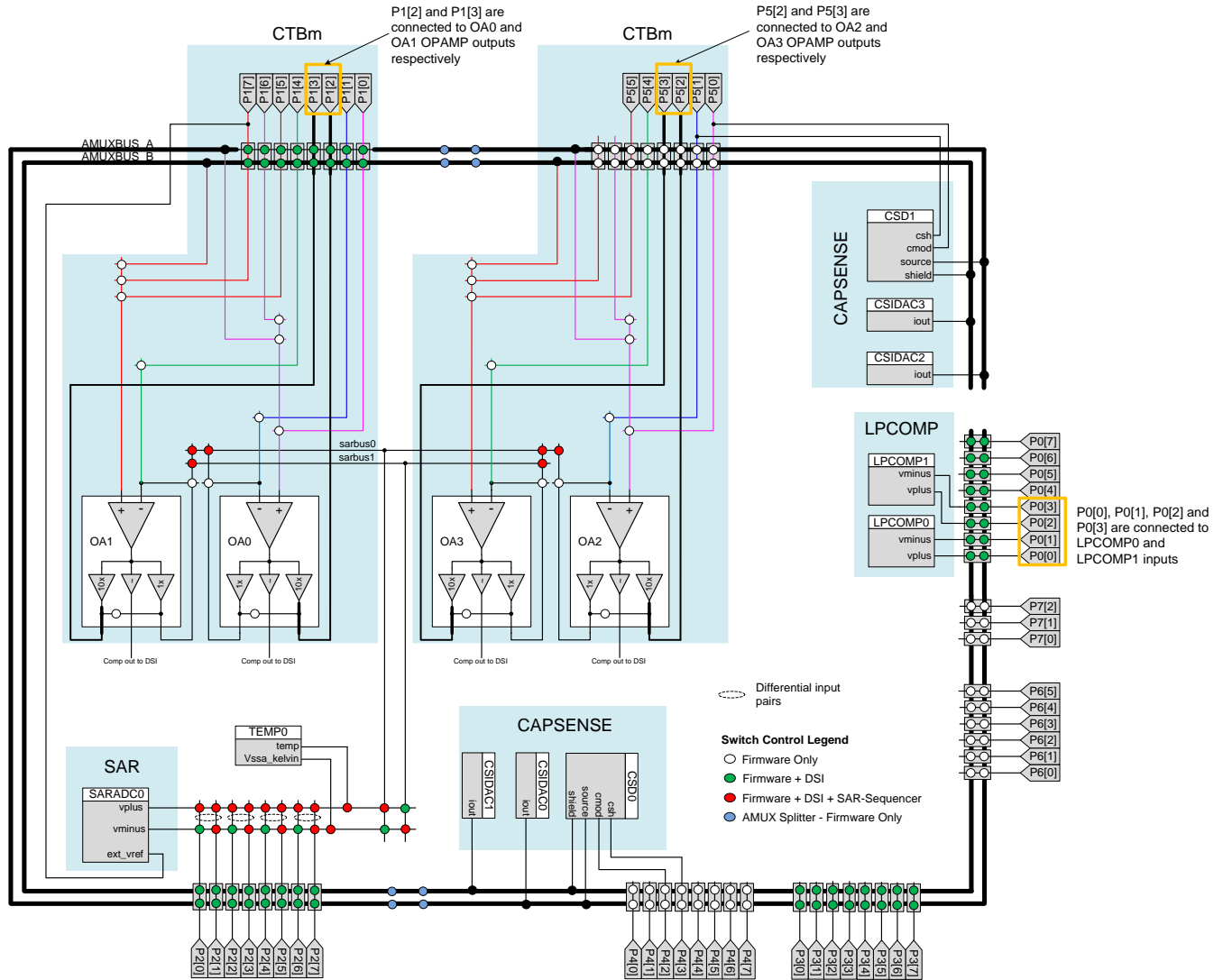


图 15. PSoC 4200L 模拟路由图

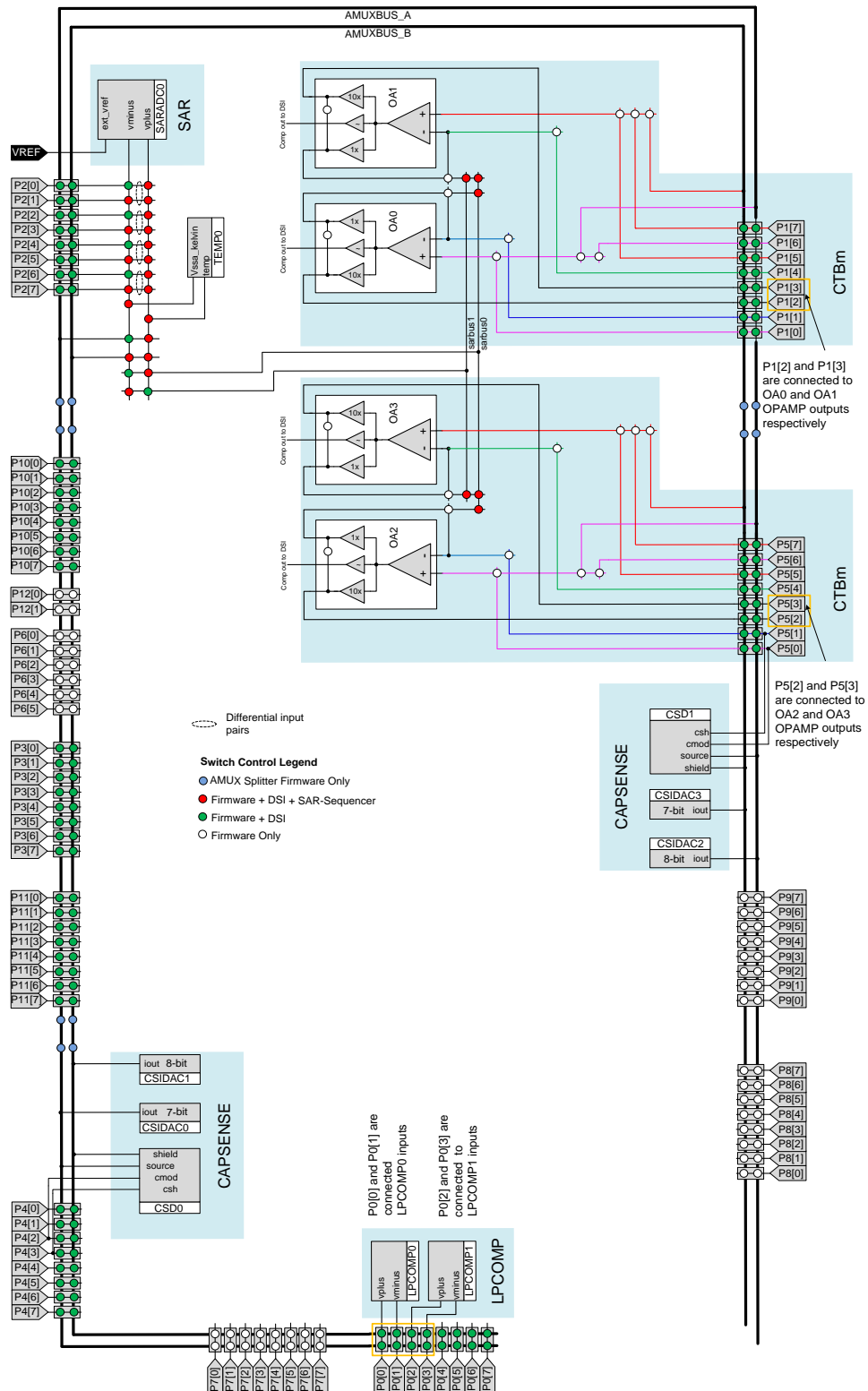
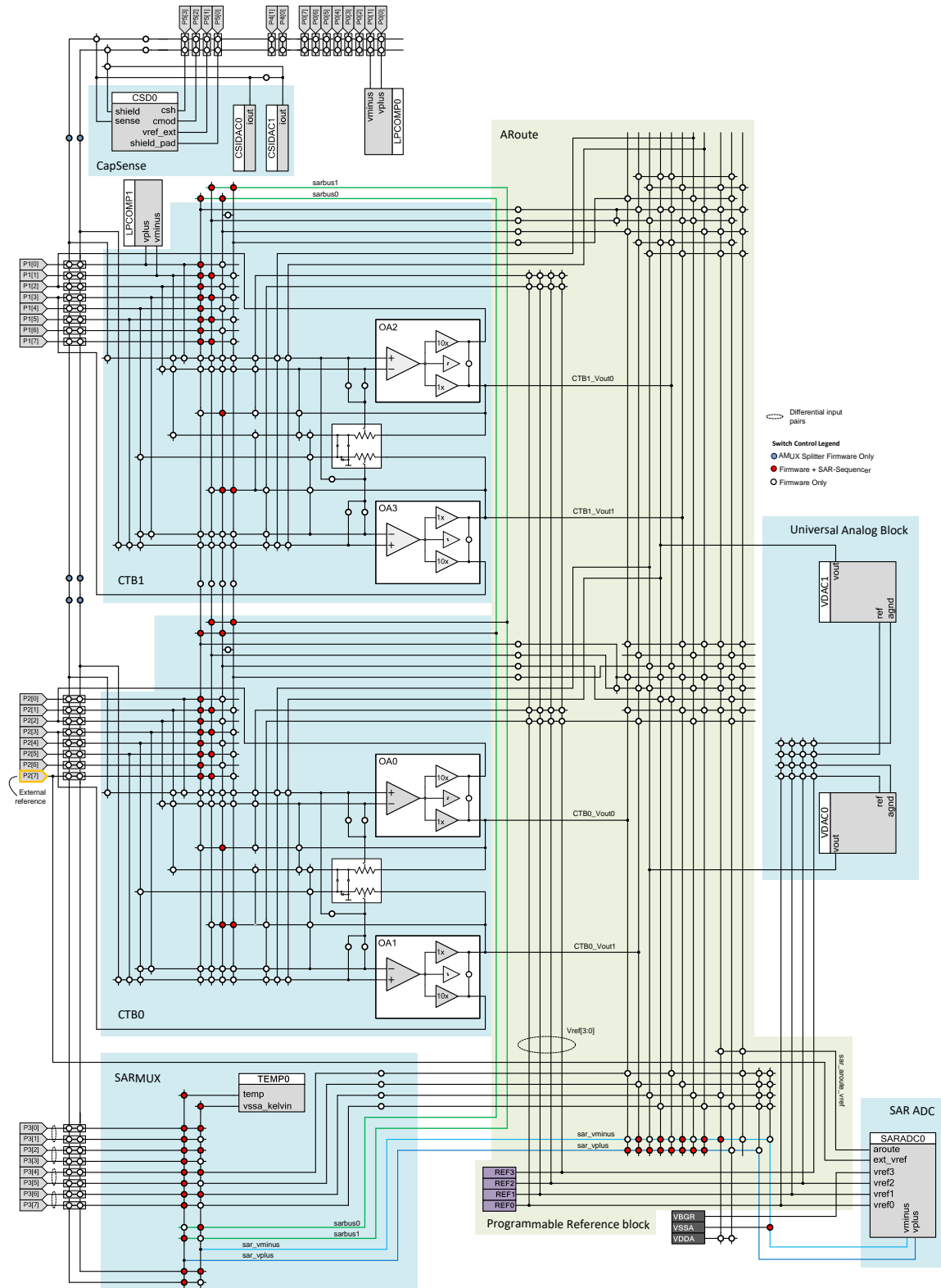


图 16. PSoC 4100PS 模拟路由图



注意：PSoC Creator IDE 工具为进行设计提供了模拟路由图，如图 9 到图 16 所示。请在 PSoC Creator 中查看项目的 .cydwr 文件的 Analog 选项卡。

3.3 启动和低功耗操作

复位/上电时，所有 GPIO 引脚将在模拟高阻抗模式下启动，此时，输入缓冲器和输出驱动器被禁用。退出复位前，这些 GPIO 引脚仍处于该模式；然后在引导过程中对每个 GPIO 引脚的相应寄存器加载初始操作配置，并立即生效。器件运行期间，可以通过写入到相应的寄存器内来配置 GPIO。

注意：在所有 PSoC 4000 器件的上电过程中，P1[6]引脚会暂时被配置为 XRES 引脚，直到器件执行启动代码为止。上电时，请勿下拉该引脚，因为这样会使器件处于复位状态中。请注意，通过 P1[6]的复位仅用于测试目的，不应将其用于用户应用。

请参见 [KBA91258 — PSoC 4000 系列的 I/O 系统限制](#)，了解更多信息。

PSoC 具有四种功耗模式。表 2 显示的是 PSoC 4 系列所支持的功耗模式。

表 2. 低功耗模式

器件	睡眠模式	深度睡眠模式	休眠模式	停止模式
PSoC 4000	✓	✓	✗	✗
PSoC 4000S	✓	✓	✗	✗
PSoC 4100/4200	✓	✓	✓	✓
PSoC 4100S	✓	✓	✗	✗
PSoC 4100S Plus	✓	✓	✗	✗
PSoC 4 BLE	✓	✓	✓	✓
PSoC 4 M	✓	✓	✓	✓
PSoC 4 L	✓	✓	✓	✓
PSoC 4100PS	✓	✓	✗	✗

在睡眠模式下，各个 GPIO 都处于活动状态，并且可由外设有效驱动。在该模式下，只有 CPU 被关闭。在深度睡眠模式下，由外设（如 I²C、LCD 驱动器、运算放大器以及比较器）驱动的各个引脚都能运行。I²C 引脚可以通过地址匹配事件唤醒器件。甚至在深度睡眠模式下还会定期刷新与器件引脚相连的段式 LCD。

PSoC 4 器件（PSoC 4000 除外）还有另一个功能，即：在深度睡眠、休眠和停止模式下冻结 GPIO。当退出低功耗模式时，会自动解冻 GPIO。但请注意，由深度睡眠外设驱动的 GPIO 在深度睡眠模式下仍然有效，并不会被冻结。

在休眠和停止模式下，复位会唤醒器件。这样将清除 GPIO 配置和状态。为了保持引脚状态，请调用 `CySysPmFreezeIo()` 和 `CySysPmUnfreezeIo()` API 函数。请注意，在停止模式下，不需调用 `CySysPmFreezeIo()` 函数，因为当用户使用 `CySysPmStop()` API 函数调用停止模式时，会自动调用它。但是，调用函数进入休眠模式前，应先调用 `CySysPmFreezeIo()` API。通过调用 `CySysPmUnfreezeIo()` API，可以对 GPIO 进行解锁。退出停止模式时，也要调用该 API。注意：发生外部复位（XRES）事件时不能保持冻结引脚的状态和配置。

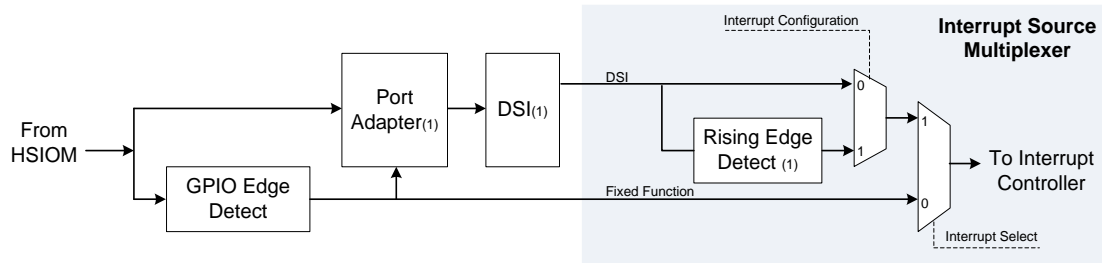
在深度睡眠模式下，`CySysPmFreezeIo()` 和 `CySysPmUnfreezeIo()` API 很有用。[深度睡眠模式下的控制寄存器处理](#)一节显示了一个关于使用该特性的示例。基于 UDB 的组件（如控制寄存器）在深度睡眠、休眠以及停止模式下无效，并且丢失数据。如果控制寄存器正在驱动一个引脚，且引脚的最终状态为‘1’，那么 PSoC 进入或退出这些模式时会引起窄脉冲。为了避免这种窄脉冲，在进入低功耗模式前需要先冻结 GPIO。

更多有关低功耗模式的信息，请参阅 [AN86233 — PSoC 4 低功耗模式和降低功耗技术](#)。

3.4 GPIO 中断

图 17 的是从 HSIOM 到中断控制器的信号路径。

图 17. GPIO 中断信号路由



(1) not applicable to PSoC 4000, PSoC 4000S, PSoC 4100S, PSoC 4100S Plus, PSoC 4100PS

在处理器内核中，中断控制器的每一组 32 个中断线中都有一个“中断源复用器”。该复用器模块选定了中断源，并提供了上升沿检测或直接连接到中断控制器等选项。有两个中断源，分别为：

1. 固定功能源
2. DSI 源

“中断选择”线会选择 DSI 或固定功能源。“中断配置”选择了直接连接或上升沿检测逻辑布线，以便连接到中断控制器。

固定功能中断源有一个固定中断向量；因此中断源可专门连接到 Cortex® M0/Cortex M0+CPU 中 32 个中断线中的某一个。这种路由的中断源直接连接着中断控制器。当通过 DSI 路由中断源时，向量选择并不是固定的。除了直接连接外，该路由方式还可提供另一个选择，即上升沿检测。

注意：更多有关中断向量的信息，请参考[技术参考手册](#)中的“中断”一节。

中断源复用器不仅限于 GPIO 中断；它还适用于所有其他中断源。欲详细了解有关其他中断源的信息，请参阅 [AN90799 — PSoC® 4 中断](#)中的“中断源”一节。

除了中断源复用器中的现有资源，GPIO 中断还可以使用自己的 GPIO 边沿检测模块，如图 17 所示。

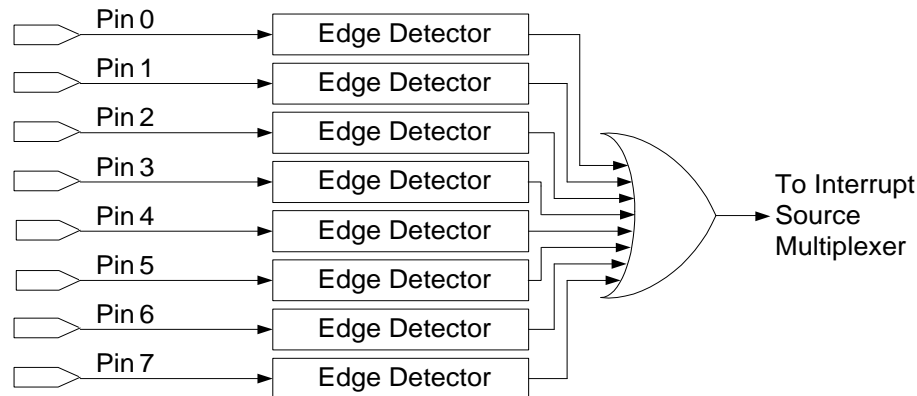
可以通过以下方式对 HSIOM 的 GPIO 中断信号进行布线：

- 第一种方法：在中断源复用器被配置为直接连接逻辑的情况下，使用 GPIO 边沿检测模块进行固定功能源布线
- 第二种方法：在中断源复用器被配置为上升沿的情况下，使用 GPIO 边沿检测模块进行 DSI 源布线
- 第三种方法：在中断源复用器被配置为直接连接逻辑的情况下，使用 GPIO 边沿检测模块进行 DSI 源布线
- 第四种方法：在中断源复用器被配置为上升沿的情况下，可以进行 DSI 源布线而不使用 GPIO 边沿检测模块
- 第五种方法：在中断源复用器被配置为直接连接逻辑的情况下，可以进行 DSI 源布线而不使用 GPIO 边沿检测模块

更多有关不同布线方式的配置，请参考[引脚组件中断](#)一节中所讲述的内容。

图 18 显示的是 GPIO 边沿检测模块。该模块检测下一个 GPIO 信号的上升沿、下降沿或者双边沿。每个端口内的各个独立 GPIO 中断信号被进行“OR”运算，从而生成单个中断请求。因此，每个端口只有一个中断向量。

图 18. GPIO 边沿检测



如图 18 所示，中断被触发时，将要求识别中断源。PSoC 4 提供了一个状态寄存器，用于确定中断引脚。读取状态寄存器后，应立即清除它（若在使用 GPIO 沿检测逻辑的情况下），以避免发生下面行为：

1. 中断源复用器被配置为上升沿时，会触发单个中断，并且不响应其它请求。使用第二种布线方法时，会发生这种情况。
2. 中断源复用器被配置为直接连接时，会为单个请求重复触发中断。使用第一种第三种布线方法时，会发生这种情况。
3. GPIO 中断进行布线（而不使用 GPIO 边沿检测模块）时，便不需要清除中断。但如果中断源复用器中的上升沿检测逻辑也被旁路，则会导致电平触发中断（第五种布线方法）。这时，将在引脚信号为高电平期间重复触发中断。因此，不使用 GPIO 边沿检测模块（第四种方法）时，建议将中断源复用器配置为上升沿触发中断。

注意：GPIO 中断逻辑在睡眠、深度睡眠和休眠模式下仍起作用；因此可以将任意引脚作为唤醒源。在 PSoC 4200/PSoC 4100、PSoC 4 M 和 PSoC 4 L 器件中，专用唤醒引脚（P0[7]）用于使器件从停止模式唤醒。对于 PSoC 4 BLE 器件，停止模式的唤醒引脚为 P2[2]。

3.4.1 GPIO 中断的限制

1. 端口 4 和更高的端口没有端口适配器。因此，这些端口不支持使用 DSI 布线进行引脚中断。
2. PSoC 4000 和 PSoC 4100/PSoC 4200 器件的每个端口都有一个中断向量。PSoC 4 BLE 端口 5 以上的端口和 PSoC 4 M 端口 4 以上的端口都没有专用的中断向量。但可以给这些端口分配一个公用的中断向量。当使能任意端口中断时，该中断向量会被触发。请参考[引脚组件数据手册](#)，了解如何使用这个公用的端口中断。
 1. 请参考[架构技术参考手册（TRM）](#)中的“中断”一节，以了解有关公用中断向量的端口。

[引脚中断](#)一节中提供了一个示例项目，详细说明了 GPIO 中断的用法。请参考应用笔记 [AN90799 — PSoC 4 中断](#)，了解中断概述。

4 过压容差（OVT）引脚

PSoC 4 BLE 的引脚 P5[0]和 P5[1]以及 PSoC 4 M 的端口 6 都是 OVT 引脚。在 PSoC 4 L 中，端口 6 和端口 8 都有 OVT 引脚。它们与通用的 GPIO 相同，但具有以下各附加特性：

1. 过压容差功能 — OVT 引脚和电源范围间没有 ESD 钳位二极管。这样能使 OVT 引脚承受高于 VDDIO、VDDD 或 VDDA 电压，数值高达 5.5 V。
2. 与通用 GPIO 相比，OVT 引脚提供了更好的下拉驱动强度。
3. 当被配置为 I²C，并且它的各条信号线被路由到 OVT 引脚时，则作为串行通信模块（SCB）使用；能够满足以下 I²C 规范：
 - a. 加强型快速模式的低电平输出电流（IOL）规范
 - b. 快速模式和加强型快速模式的迟滞以及最小下降时间规范

欲详细了解有关 I/O 硬件的信息，请参见[技术参考手册](#)中“[I/O 系统](#)”一节。

5 PSoC Creator 的 GPIO 引脚

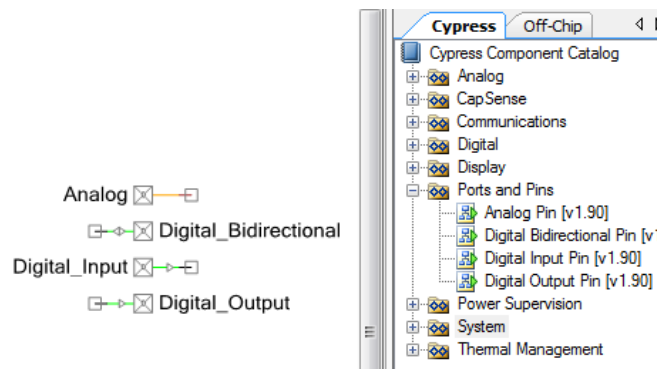
本节介绍了如何使用 PSoC Creator 进行配置和操作 GPIO 引脚。

5.1 引脚组件符号

建议使用引脚组件将 PSoC 的内部资源连接到物理引脚。这样，PSoC Creator 能够根据所选的引脚配置，在 PSoC 器件内自动分布和路由各信号。

标准的赛普拉斯组件类别下的端口和引脚类符号包括四种预定义 GPIO 配置，分别为：模拟、数字双向、数字输入和数字输出。将其中的一个组件拖放到原理图中即可为项目添加引脚，如图 19 所示。

图 19. PSoC Creator 中的引脚组件符号类型



5.2 引脚组件自定义程序

PSoC Creator 中的每个组件都有一个自定义程序，用于配置组件。图 20 显示的是引脚组件的自定义程序，可通过双击组件来访问它。

图 20. 引脚组件自定义程序

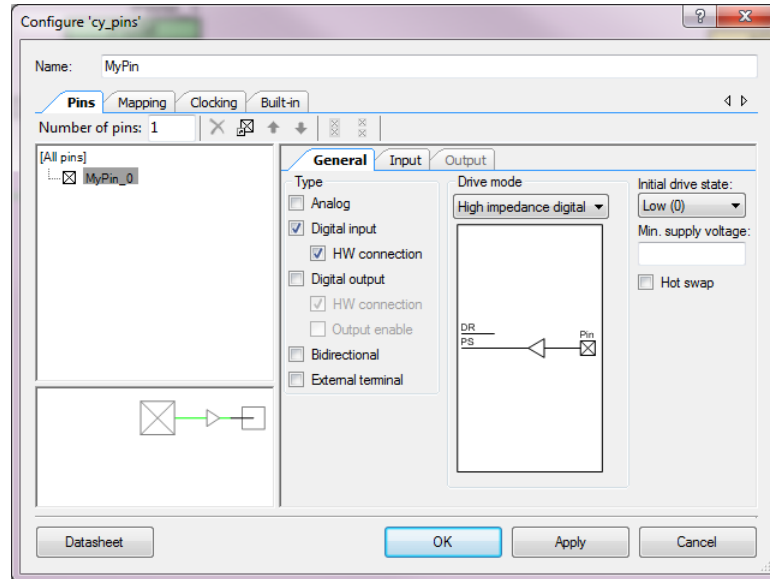


表 3 介绍了引脚组件自定义程序中的部分参数。要想查寻所有参数的信息，请参考[引脚组件数据手册](#)。

表 3. 引脚组件设置

设置	说明
依次点击 General 选项卡 > Type 项	<p>通过该设置，可以置配引脚类型。可以选择下面的类型：</p> <ul style="list-style-type: none"> ■ 模拟功能 ■ 有/无硬件（HW）连接的数字输入 ■ 有/无 HW 连接和输出使能的数字输出 ■ 双向引脚 <p>如果将数字输入或输出配置为没有 HW 连接，则引脚状态由 CPU 控制。请注意，一次可选择多种类型。例如，一个引脚可以同时被配置为模拟和数字输入。</p>
依次点击 General 选项卡 > Drive Mode	<p>该设置将引脚配置为 GPIO 引脚的基本知识 一节中所介绍的 8 种驱动模式中的一种。图 21 显示了引脚自定义程序中的驱动模式选项。</p>
依次点击 General 选项卡 > Initial Drive State	<p>“Initial drive state” 参数用于设置数据寄存器值。软件驱动引脚时，该值会显示在引脚上，以表明使用正确的驱动模式设置引脚。如果勾选引脚输出的“HW connection”项并取消选择它的“Output enable”项，那么初始驱动状态作为使能控制。将初始状态设置为“1”（PSoC Creator 的默认值）时，会使能引脚，如图 21 所示。如果引脚被配置为输入，则可以保持初始驱动状态。例如，输入引脚需要被电阻上拉时，应将驱动模式和初始状态分别配置为“Resistive pull up”（电阻上拉）模式和“HIGH”（高电平），以激活电阻上拉路径。同样，输入引脚需要被电阻下拉时，应将初始驱动状态设置为“LOW”（低电平），以使能电阻下拉路径。</p>
依次点击 Input 选项卡 > Threshold	<p>CMOS 和 LVTTTL 输入阈值设置用于整个端口。有三种阈值选项，如图 22 所示。“CMOS or LVTTTL”选项允许 PSoC Creator 工具根据端口中其他引脚的阈值设置来选择使用 CMOS 或者 LVTTTL。</p>
依次点击 Input 选项卡 > Interrupt	<p>该设置用于配置 GPIO 边沿检测模块，如 GPIO 中断 一节中所述。更多有关信息，请参考引脚组件中断一节。</p>

图 21. 引脚驱动模式设置和初始驱动状态

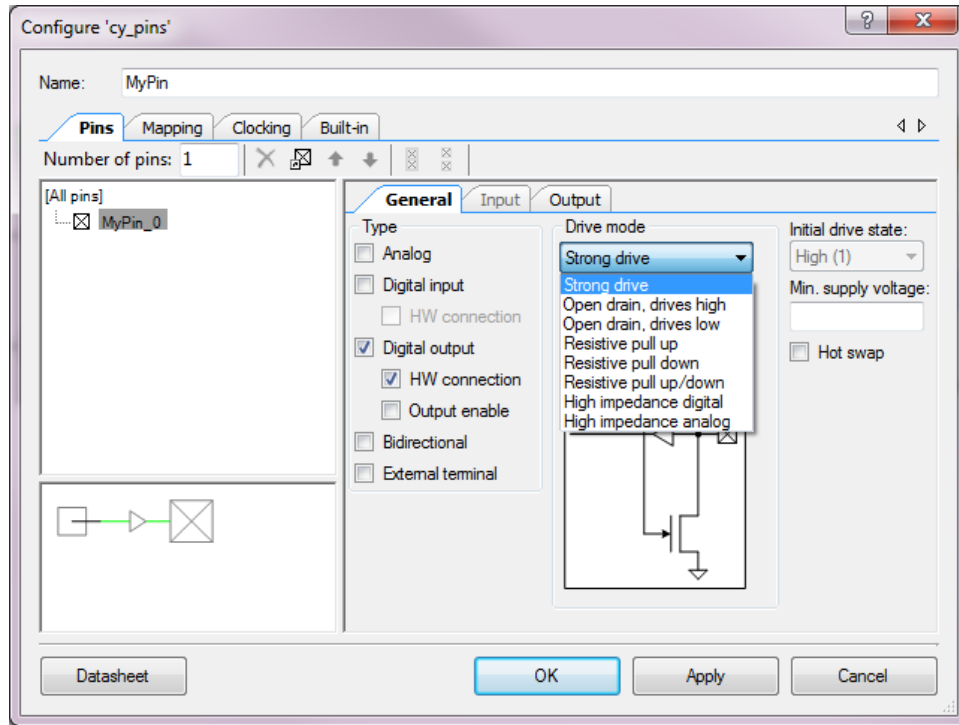
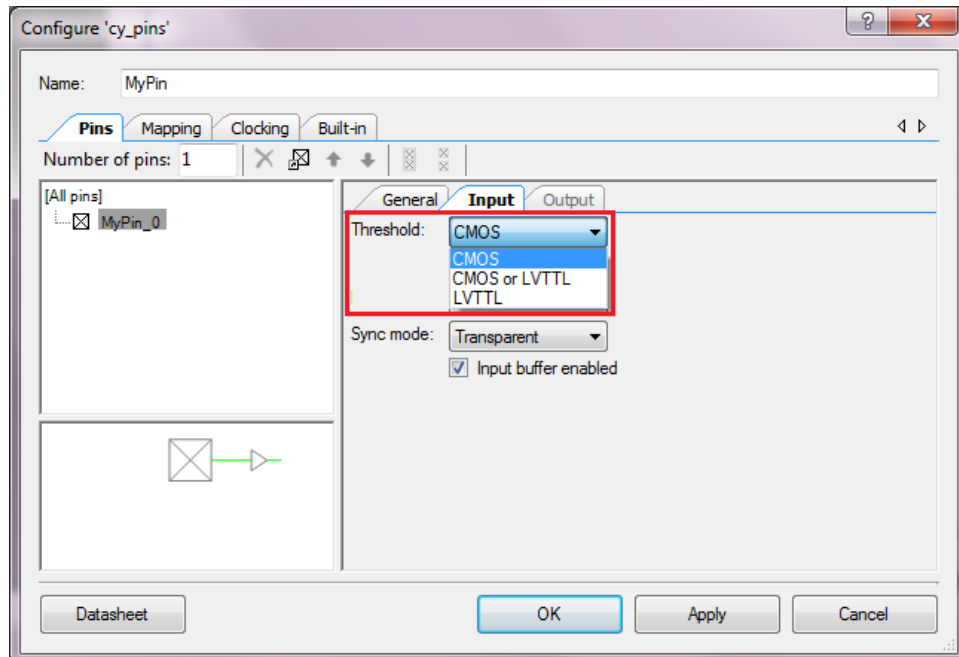


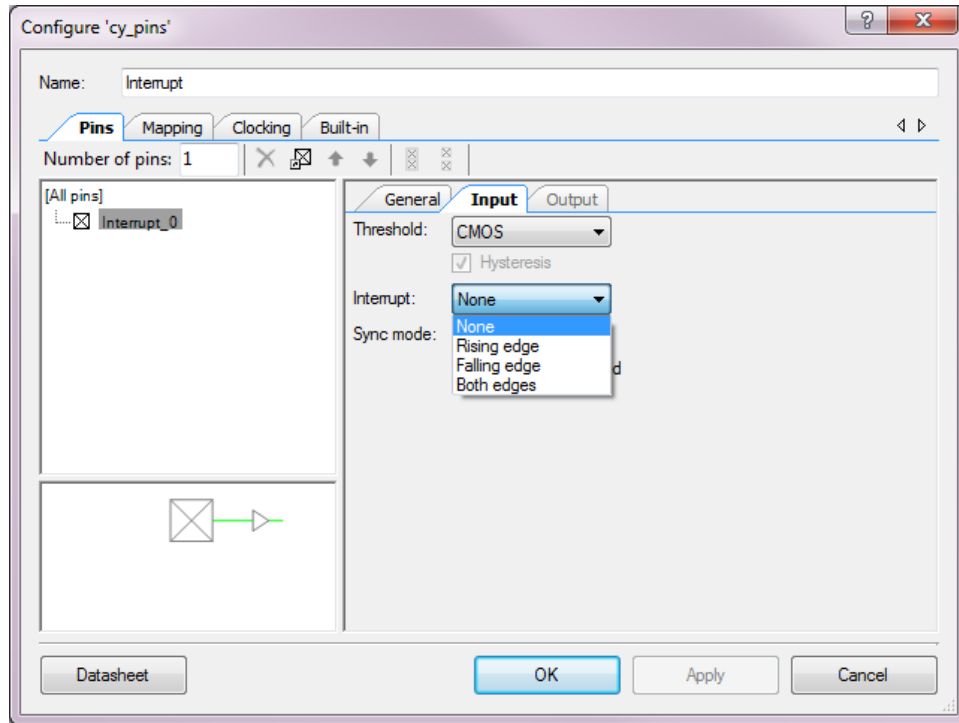
图 22. 引脚输入阈值选项



5.3 引脚组件中断

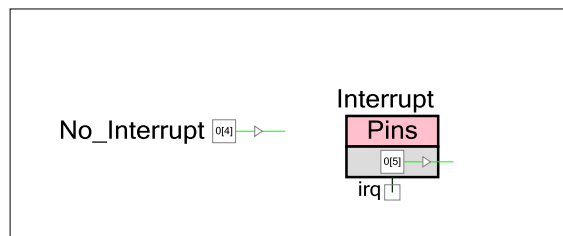
引脚自定义程序中的中断参数用于配置 GPIO 边沿检测模块，具体情况请参考 [GPIO 中断](#) 一节的内容。

图 23. PSoC Creator 的中断配置



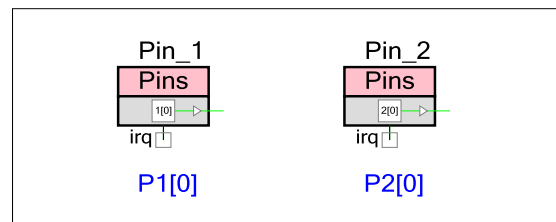
当使能中断时，引脚组件的符号会发生改变，如图 24 所示。

图 24. 使能中断时的引脚组件符号



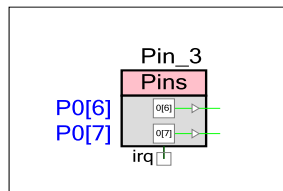
请注意，如果使能了中断，每个物理 GPIO 端口上只能使用一个引脚组件。该限制是由于端口内的所有引脚中断都被进行了“OR”（或）运算，如 [GPIO 中断](#) 一节中所述。因此，每个端口的原理图中只能显示一个 IRQ 信号。例如，请查看中断使能时的两个引脚组件，如图 25 所示。这些组件不能被映射到相同物理端口的引脚上。

图 25. 中断使能时的两个引脚组件



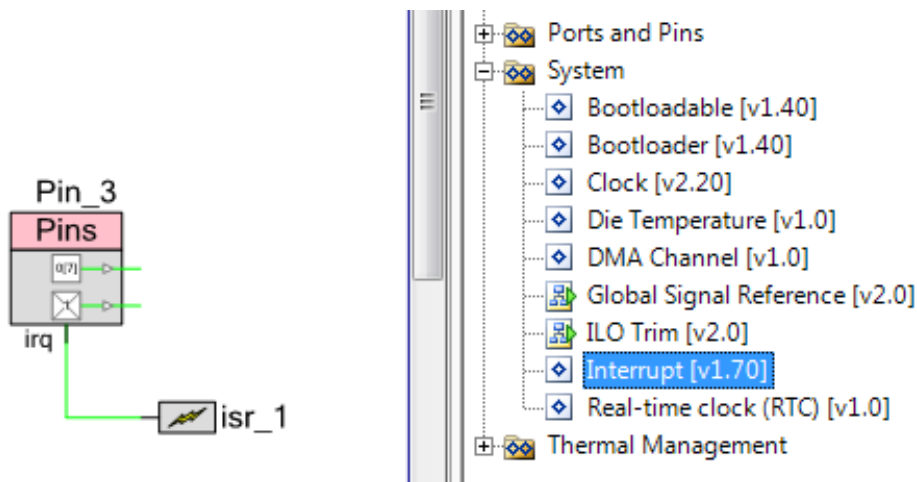
PSoC Creator 不允许将这两个组件分配给同一个端口。但可以将多个引脚分配给同一个组件，如图 26 所示。这样能确保在原理图中的物理端口上只有一个 IRQ 信号。您仍然可以为每个引脚指定各自的中断边沿触发类型。唯一限制是这些引脚在同一个端口上的位置必须是连续的。应将该中断源定义于 ISR 中；请参考[引脚中断](#)一节的内容。

图 26. 同一端口上多个带有中断的引脚



应将引脚组件的 IRQ 信号连接至中断组件。这样可以将 GPIO 中断信号路由到中断控制器。要想查找中断组件，请参考[图 27](#) 中的组件目录。

图 27. 目录中的中断组件



中断组件将“中断源复用器”配置为直接连接（在中断组件自定义程序中显示为“电平”）或上升沿。[GPIO 中断](#)一节详细说明了 GPIO 中断架构以及适用于中断信号的不同布线方式。这些布线方式是通过引脚组件和中断组件自定义程序设置进行配置的，如[表 4](#) 所示。

表 4. GPIO 中断配置

原理图	引脚组件的中断设置	中断组件设置	布线方式	说明
	上升沿、下降沿或双边沿	电平	布线 1	在各边沿上触发中断取决于相应的引脚组件设置。它使用由所选端口规定的一个固定中断向量。在该配置中，应清除 GPIO 中断状态寄存器；否则，在发送单个中断请求时，将重复触发中断。改配置可用于从低功耗模式唤醒器件。但请注意，从停止模式唤醒需要使用特定的引脚，具体取决于所选的器件。
	上升沿、下降沿或双边沿	上升沿	布线 2	在各边沿上触发中断取决于相应的引脚组件设置。在该配置中，应清除 GPIO 中断寄存器；否则，只能触发一次中断。中断向量并非固定。该配置只能从睡眠模式唤醒 CPU；它在其他低功耗模式是不可用的。
	上升沿、下降沿或双边沿	电平	布线 3	这种布线方式与第一种方式相同。但仅在中断向量被强制到某个想要的 DSI 向量线时，才会采用第三种布线方式。请参考 AN90799 — PSoC 4 中断应用笔记 ，了解如何强制中断向量。该配置只能从睡眠模式唤醒 CPU；它在其他低功耗模式是不可用的。
	禁用	上升沿	布线 4	该配置支持在上升沿上触发中断。在这种情况下，不需清除中断。该配置只能从睡眠模式唤醒 CPU；它在其他低功耗模式是不可用的。
	禁用	电平	布线 5	该配置支持电平触发中断。请注意，当引脚信号为高电平时，中断将被重复触发。这种配置也不需清除中断。该配置只能从睡眠模式唤醒 CPU；它在其他低功耗模式是不可用的。

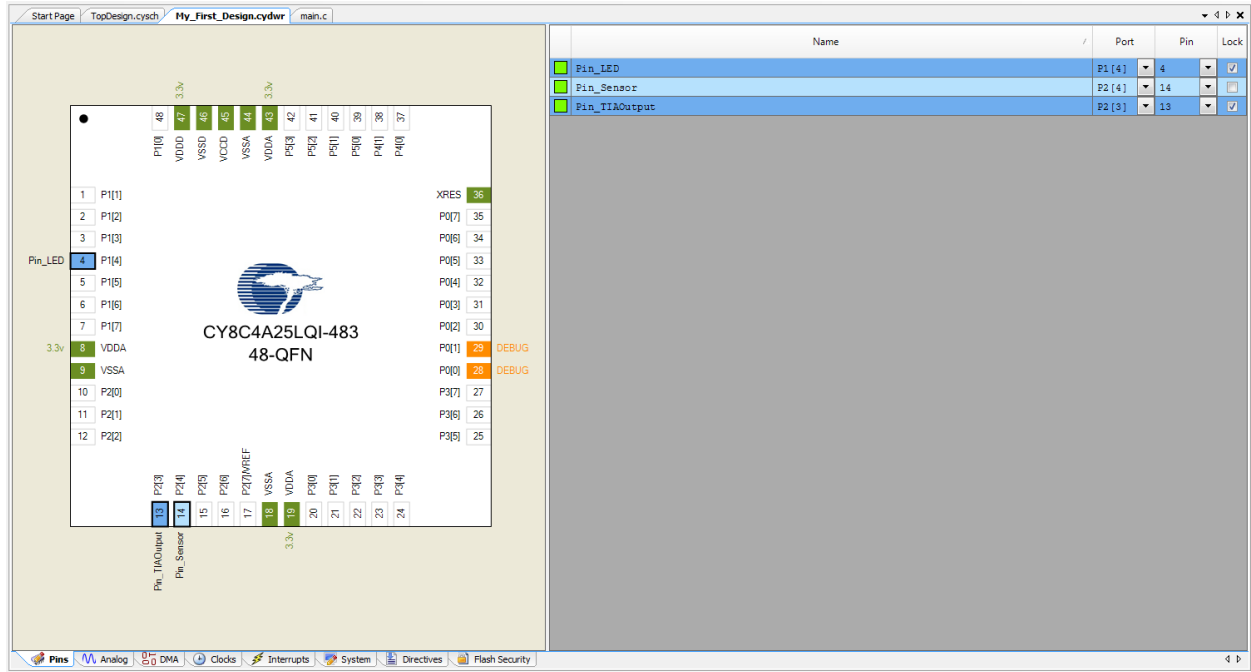
5.4 引脚的手动分配

使用设计范围资源（DWR）窗口中的 **Pins**（引脚）选项卡，可以将引脚组件分配给物理引脚。用户未选择任何引脚时，PSoC Creator 将自动分配引脚，但这种引脚分布可能会使 PCB 上的布线变得更为复杂。

图 28 显示的是三个分配引脚。灰色高亮显示的引脚是手动分配的，蓝色高亮显示的引脚是自动分配的。选择 **Lock**（锁定）选项卡可防止 PSoC Creator 重新分配引脚。

使用 PSoC Creator，您可以更加轻松地重新分配引脚（若需要），但建议在设计电路板前要考虑引脚的选择情况。

图 28. DWR 窗口中的引脚分配



注意： PSoC Creator 可以使用各个未使用的引脚开关来路由模拟信号。该配置是通过.cydwr 文件中 **System** 选项卡的 **Unused Bonded I/O** 参数实现的。更多有关信息，请参考 PSoC Creator 助手。

5.5 PSoC Creator API

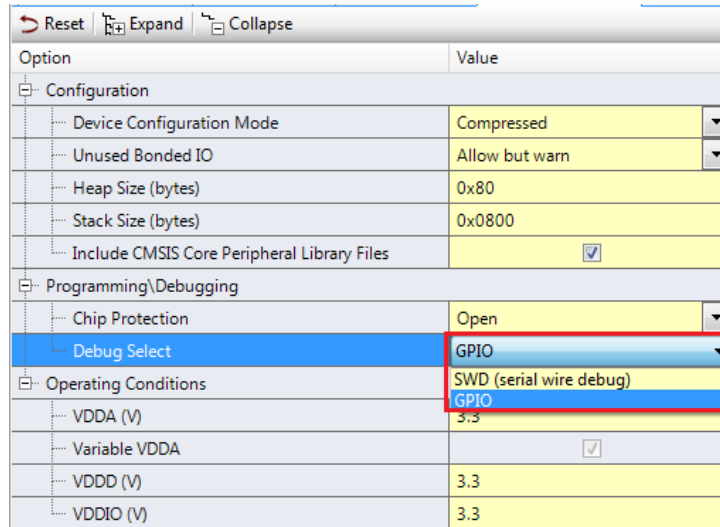
赛普拉斯提供了一组 API 函数，固件可调用它们来动态控制 GPIO。引脚组件的 API 在组件范围和引脚基础上提供访问功能。更多有关信息，请参考 [引脚数据手册](#) 中“API”一节的内容。

cypins.h 文件中提供了使用于引脚的 API（作为 cy_boot 的一部分）；PSoC Creator 系统参考指南（依次打开 **Help > Documentation > System Reference**）中“引脚”一节对这些函数进行了描述。您可以使用这些函数控制每个物理引脚的配置寄存器。

5.6 GPIO 引脚上的调试逻辑

可以在端口引脚上共享 PSoC 4 串行线调试 (SWD) 引脚。更多有关调试端口引脚的信息，请参考其相应的[器件数据手册](#)。但可以禁用该调试功能，并将引脚作为通用的 GPIO 使用（即在 DWR 窗口的 **System** 选项卡中将 “Debug Select” 选项设置为 “GPIO”，如图 29 所示）。

图 29. 禁用调试端口

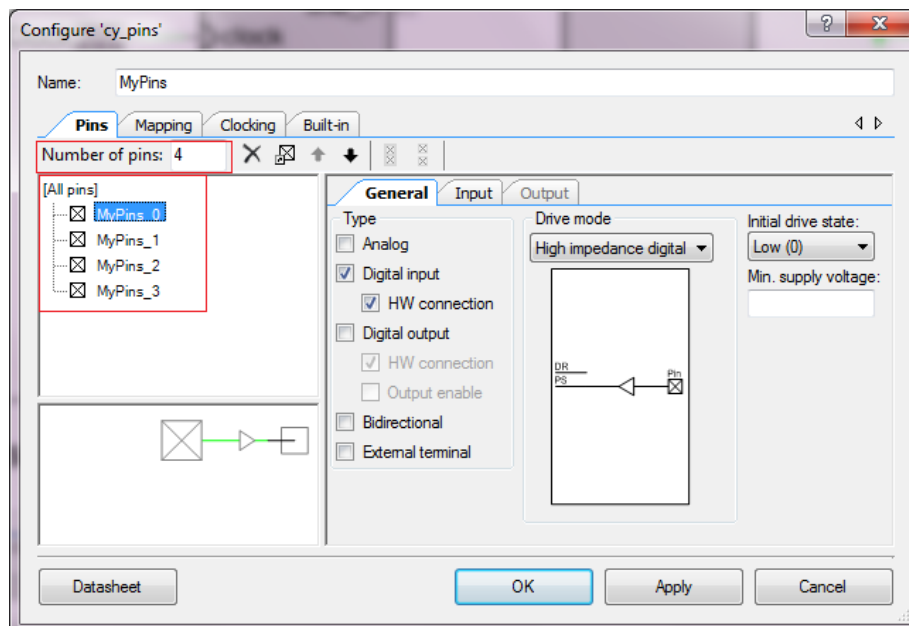


请注意，禁用调试接口不会影响对该器件的编程能力。

5.7 添加多个 GPIO 引脚，作为逻辑端口

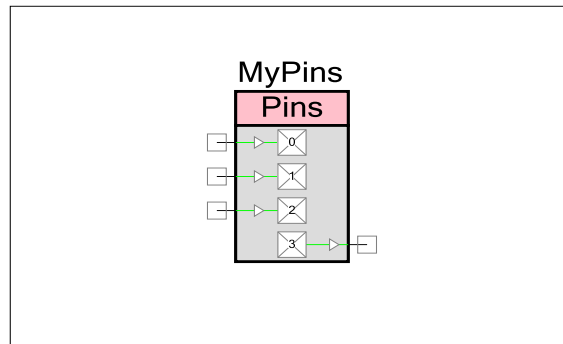
在 PSoC Creator 中，您最多可以将 36 个引脚组合成一个逻辑端口，然后通过端口的定义名称引用到代码中。所有引脚可能来自同一物理端口，也可能来自不同的物理端口。请在引脚组件自定义程序中设置端口所需的 **Number of Pins**（引脚数量）。这些引脚将出现于字段下的列表中（如图 30 所示）。可以单独对每个引脚进行配置。要想同步组件中所有引脚的配置，请选择 **[All Pins]**（所有引脚）项。

图 30. 四个引脚中的一个被配置为数字输入



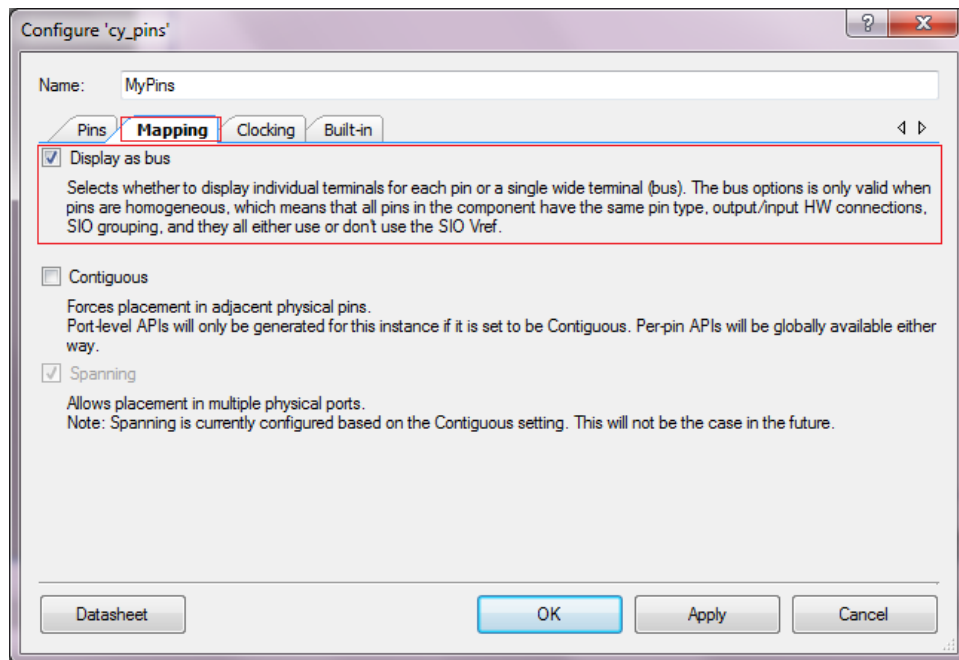
如果将引脚数量配置为‘4’（其中有 3 个数字输入和一个数字输出），原理图符号会如图 31 显示。

图 31. 端口配置窗口中的引脚组件



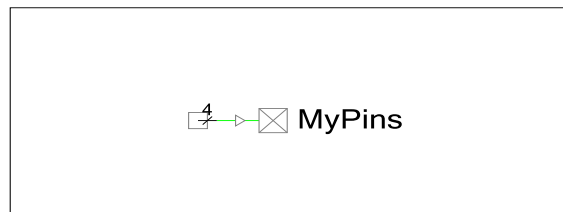
若不想采用单独引脚终端，您可以将端口显示为总线。在引脚配置窗口的 **Mapping** 选项卡中勾选 **Display as Bus** 项，使该端口显示为总线，如图 32 所示。请注意，只有所有引脚类型一样时，才能显示为总线。

图 32. 显示为总线选项



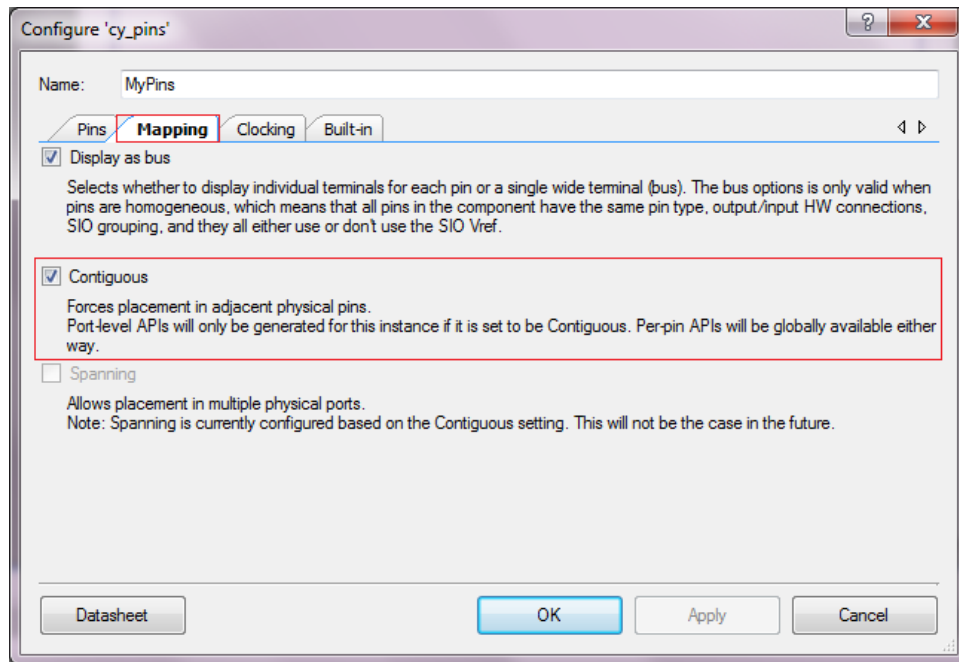
将 **Number of Pins** 选项设置为 4 个数字输出时，原理图符号将如图 33 显示。

图 33. 四个引脚被显示为总线



通过选择 **Mapping** 选项卡中的 **Contiguous** 复选框，可以强制引脚（即总线终端）映射至相邻的引脚上（如图 34 所示）。

图 34. 连续引脚放置选项



当选择 **Contiguous** 时，PSoC Creator 会修改可用的引脚选项，用以使端口设置相互匹配，如图 35 所示。禁用“Contiguous”选项时，可以选择任意引脚。如果使能了“Contiguous”选项，则只能选择相邻的引脚（如图 35 所示）。

图 35. 禁用/使能“Contiguous”选项时的引脚放置

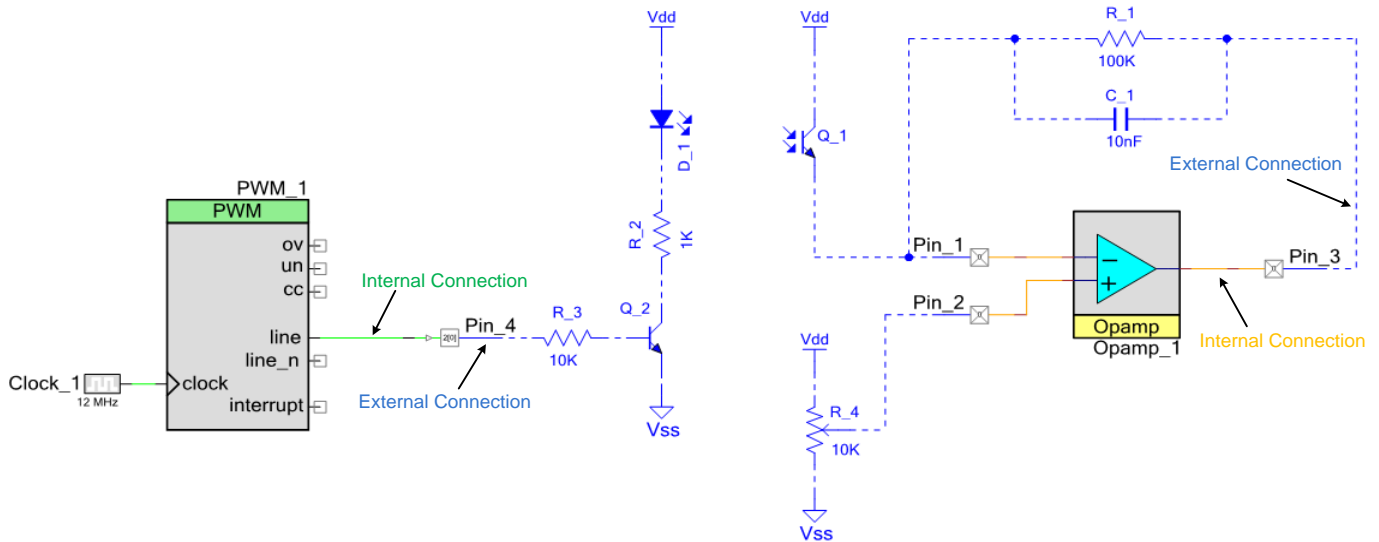
Contiguous Disabled					Contiguous Enabled				
Alias	Name	Port	Pin	Lock	Alias	Name	Port	Pin	Lock
	MyPins [0]		▼	▼		\MyPins[3:0]\		▼	▼
	MyPins [1]		▼	▼		P0 [3:0]			
	MyPins [2]		▼	▼		P0 [4:1]			
	MyPins [3]		▼	▼		P0 [5:2]			
			▼	▼		P0 [6:3]			
			▼	▼		P0 [7:4]			
			▼	▼		P1 [3:0]			
			▼	▼		P1 [4:1]			
			▼	▼		P1 [5:2]			
			▼	▼		P1 [6:3]			
			▼	▼		P1 [7:4]			

在引脚配置窗口和引脚组件数据手册中详细介绍了这些功能。

5.8 表示片外组件

片外组件目录提供了在同一原理图上融合外部和内部组件的方法，如图 36 所示。这样可以优化文档，更详细地演示了内部原理图非常适合整个设计。片外组件包含了如代码中所注释的功能 — 它们不改变 PSoC 设计的性能，而又为整个系统提供更加清晰的画面。

图 36. 设计片外组件



在图 36 显示的设计中，PWM_1 和 Opamp_1 都是器件的内部模块。通过使用 Pin_1 到 Pin_4，可以将这些模块连接至各个外部组件。绿色和橙色线表示器件内部连接（绿色：数字信号线，橙色：模拟信号线）；蓝色线和各组件都是通过外部连接的。如果在原理图中要建立与外部组件的连接，请在引脚组件自定义程序中使能“External Terminal”（外部终端）参数，如图 37 所示。这样会向原理图内添加额外的终端，如图 38 所示。

图 37. 使能外部终端

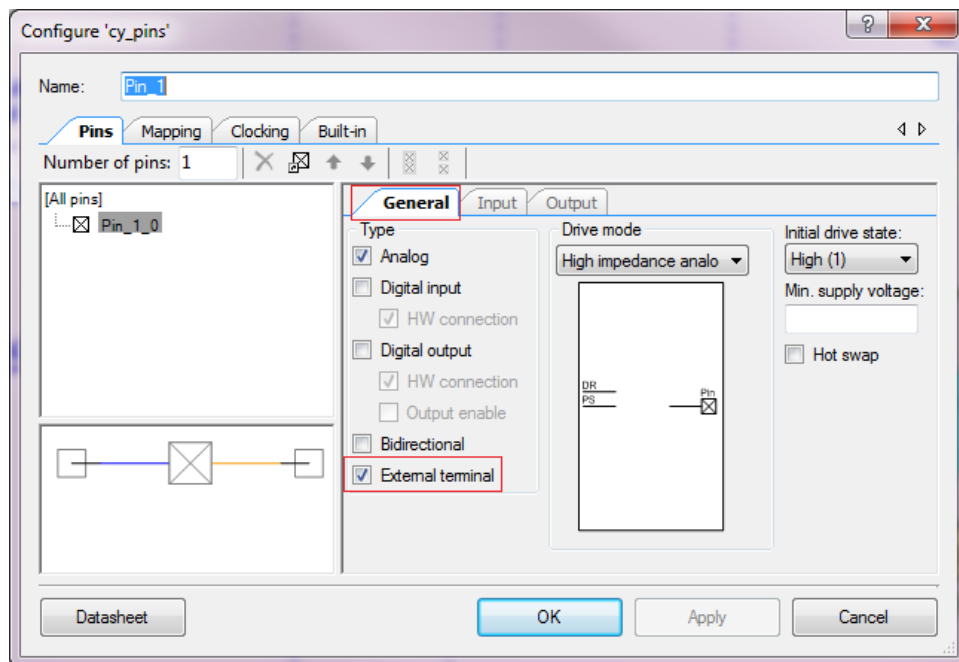
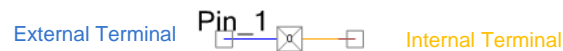
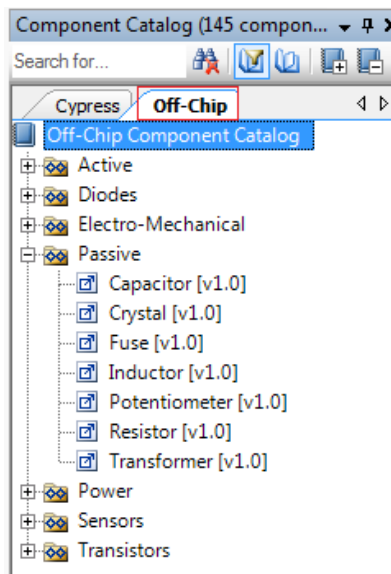


图 38. 引脚组件与内部和外部终端的连接



可以在组件目录的“Off-Chip”选项卡中寻找在原理图中需要连接到外部终端的组件，如图 39 所示。

图 39. 片外组件目录



该目录中介绍的是在电路板上通常被连接到 PSoC 器件引脚的组件，包括电阻、电容、晶体管、电感、开关和其他组件。将组件拖放到原理图中（和内部组件情况一样）。

6 GPIO 提示和技巧

本节提供的实际示例介绍了如何使用 GPIO 引脚。您可以在本应用笔记中所提供的 PSoC Creator 项目中查找这些示例详情。

表 5. PSoC Creator 项目

序号	章节	项目	PSoC 4000、 4000S、4100S、 4100S Plus、 4100PS	PSoC 41x7_BLE、 4100、4100M	PSoC 4200、 4200M、4200L、 42x7_BLE
1	切换 LED	Project01_ToggleLED	✓	✓	✓
2	读取输入并写入输出	Project02_ReadingPin	✓	✓	✓
3	使用数字逻辑门驱动输出信号	Project03_HWDrive			✓
4	使用双向引脚	Project04_BidirectionalPin		✓	✓
5	GPIO 输入/输出同步设置	Project05_GPIOSynchronization		✓	✓
6	通过数据寄存器能更快切换 GPIO	Project06_FastGPIOUpdate	✓	✓	✓
7	配置 GPIO 输出使能逻辑	Project07_OutputEnable			✓
8	引脚中断	Project08_PinInterrupt	✓	✓	✓
9	使用固件配置 GPIO 中断设置	Project09_GPIOIntConfig	✓	✓	✓
10	在 GPIO 上同时使用模拟和数字功能	Project10_GPIOAnalogDigital		✓	✓
11	组合引脚以获得更大的源电流/灌电流	Project11_GangingPins			✓
12	深度睡眠模式下的控制寄存器处理	Project12_ControlRegInDeepSleep			✓

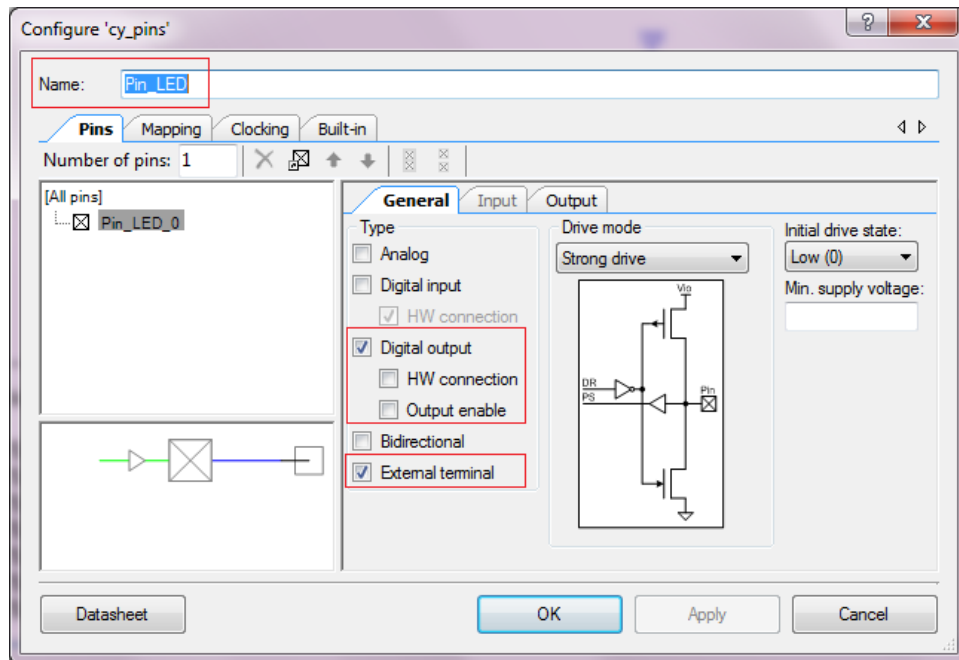
可以使用赛普拉斯的开发套件（[PSoC 4 开发板](#)）来测试这些项目。

6.1 切换 LED

GPIO 的最简单用法是通过固件将引脚输出设置为高电平或低电平。本示例演示了如何使用引脚组件 API 来设置用于切换 LED 的输出。

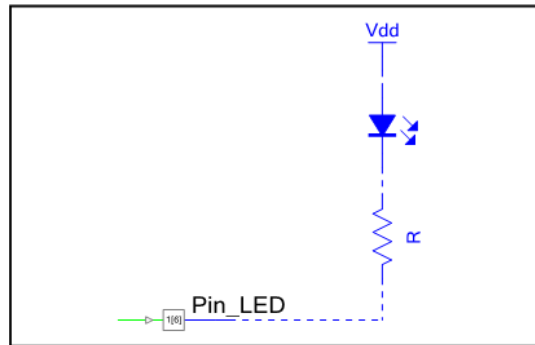
1. 将一个数字输出引脚组件放置在项目原理图中。
2. 将该组件命名为“Pin_LED”，并禁用硬件连接，如图 40 所示。

图 40. Pin_LED 配置



3. 使能外部引脚，以便在原理图中与各个外部组件建立连接。
4. 将该引脚分配给 DWR 窗口中 Pins 选项卡内的某个物理引脚（该示例中使用的是 P1[6]）。
5. 将该物理引脚连接着一个 LED。请注意，该 LED 和电阻 ‘R’ 均为片外组件。请参考图 41，了解相关的 TopDesign 原理图。

图 41. 切换 LED 的示例原理图



在 *main.c* 中，使用组件 API 来设置输出，如下所示：

```
for(;;)
{
    /* Set LED output to logic HIGH */
    Pin_LED_Write(1u);

    /* Delay of 500 ms */
    CyDelay(500u);

    /* Set LED output state to logic LOW */
    Pin_LED_Write(0u);

    /* Delay of 500 ms */
    CyDelay(500u);
}
```

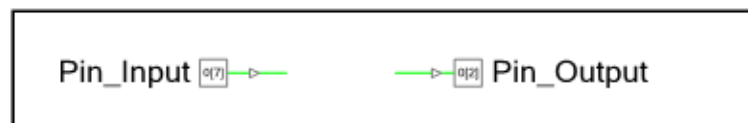
6. 编译项目并将其编程到 PSoC 4 器件内。
这样会使 LED 按照 1 Hz 的频率闪烁发光。

6.2 读取输入并写入输出

本示例介绍了如何通过组件 API 读取和写入一个 GPIO 引脚。输出引脚会将驱动的输入引脚状态反转。

1. 在项目原理图中放置两个引脚：一个数字输入引脚和一个数字输出引脚（硬件连接已被禁用），如图 42 所示。

图 42. 输入和输出示例原理图



2. 在 .cydwr 窗口中为 Pin_Input 和 Pin_Output 分配引脚。
3. 根据 Pin_Input，使用组件 API 设置 Pin_Output 的状态，如下所示：

```
for(;;)
{
    /* Set the output pin with an inverted value of input pin */
    Pin_Output_Write(~Pin_Input_Read());
}
```

4. 编译项目并将其编程到 PSoC 4 器件内。

通过将信号生成器的一个方波提供给 Pin_Input，可以对该项目进行测试。Pin_Output 上的信号将为 Pin_Input 信号的反向形式。

6.3 使用数字逻辑门驱动输出信号

前面的示例介绍了如何使用处理器内核来读取某个引脚，并将另一个引脚设置为所读取值取反后的值。该示例演示了与其相同的操作，但使用的是可配置的数字资源（即通用数字模块 — UDB）。在该示例中，一个输入引脚信号被路由至 NOT 门控，以及 NOT 门控的输出被路由至另一个引脚。请按照以下步骤创建项目：

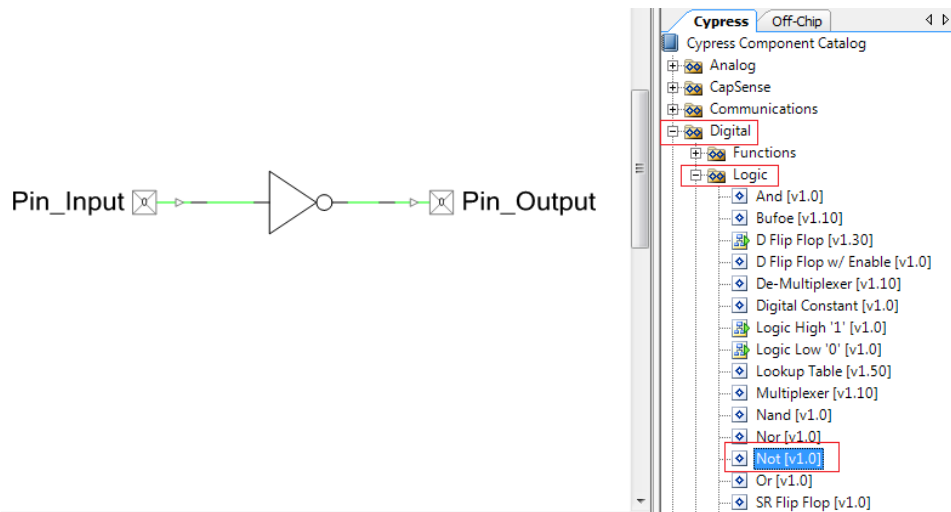
1. 在项目原理图中放置两个引脚：一个数字输入引脚和一个数字输出引脚（使能了硬件连接），如图 43 示。

图 43. 使能硬件连接的输入和输出引脚



2. 放置一个 NOT 门控，并将其连接到引脚上，如图 44 所示。

图 44. NOT 门控连接



3. 在.cydwr 窗口中为 Pin_Input 和 Pin_Output 分配引脚。
4. 该项目不需任何代码。编译项目并将其编程到 PSoC 4 器件内。
5. 与前面项目相同，通过将信号生成器的一个方波提供给 Pin_Input，可以对该项目进行测试。Pin_Output 上的信号将为 Pin_Input 信号的反向形式。

6.4 使用双向引脚

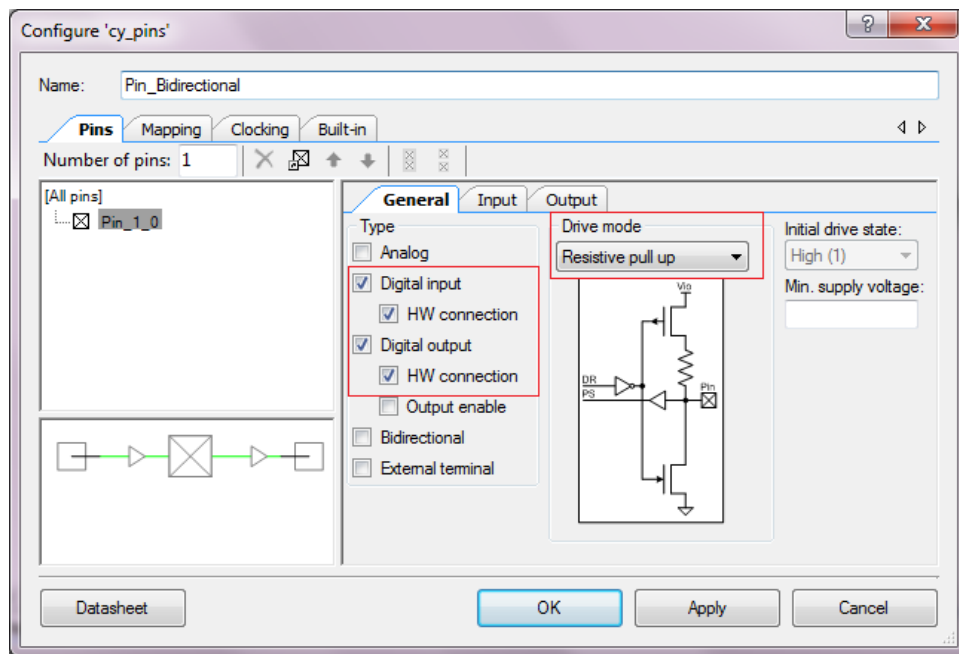
该示例说明了如何使用双向引脚，即数字输入和数字输出都被使能。PSoC Creator 提供了一个具有双向配置的引脚组件：“数字双向引脚”。



但是该引脚组件为输入和输出只显示单个终端。其用法仅限于 I²C SDA 和 SCL 信号线。在多种应用中，使用两个终端（一个用作输入，一个用作输出）是非常有用。通过勾选引脚组件自定义程序中的“Digital Input”和“Digital Output”选项，便可以实现该操作。以下是配置这样引脚的示例，其中输入端连接着一个开关，用于将该引脚下拉为逻辑低。继续通过电阻将该引脚上拉到逻辑‘1’。为了检查双向引脚的状态，需要将引脚信号路由到另一个引脚上。请按照以下步骤创建项目：

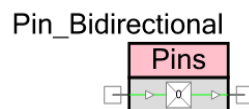
1. 将一个数字输入引脚放置在原理图中。
2. 使能“Digital Output”选项，并将“Drive mode”设置为“Resistive pull up”，如图 45 所示。

图 45. Pin_Bidirectional 配置



3. 请确保在原理图中，组件会如图 46 显示。

图 46. 原理图中的 Pin_Bidirectional



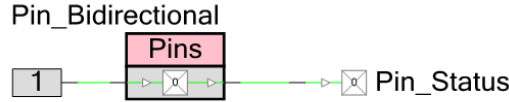
4. 继续使用电阻将该引脚上拉到逻辑高电平，如图 47 所示。

图 47. 将逻辑高连接到 Pin_Bidirectional 引脚



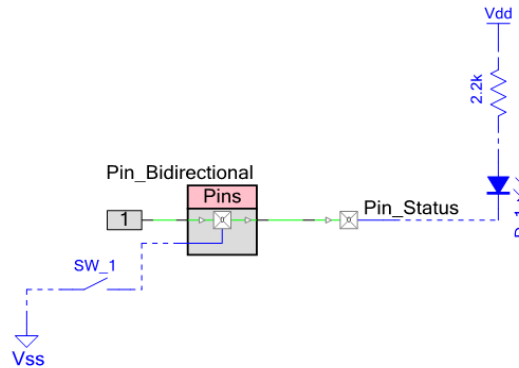
- 现在，使用连接着输入端的另一个引脚（Pin_Status）可以检查到该双向引脚的状态。放置一个数字输出引脚，并将其连接到 Pin_Bidirectional 的输入缓冲器上，如图 48 所示。

图 48. 将 Pin_Status 连接到 Pin_Bidirectional



进行外部连接后，原理图会如图 49 所示。

图 49. 完整的原理图



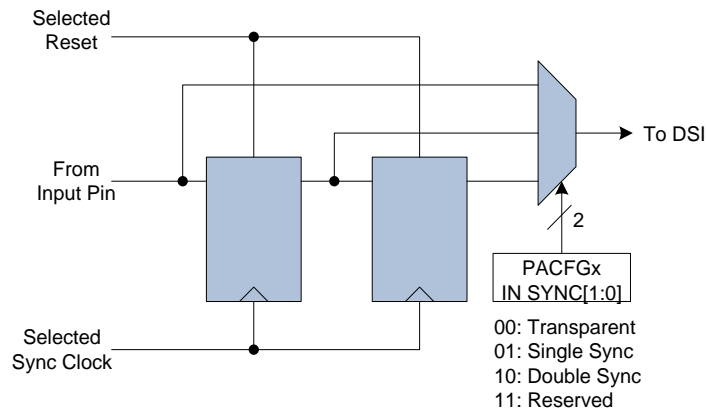
- 在.cydw 窗口中分配各引脚。
- 该项目不需要任何用户代码。编译项目并将其编程到 PSoC 4 器件内。

连接到 Pin_Status 引脚的 LED 指出 Pin_Bidirectional 引脚的输入缓冲器状态。未按下开关时，会通过电阻将 Pin_Bidirectional 引脚上拉到逻辑 '1'。这样会关闭 LED（因为该 LED 仅在驱动引脚处于低电平时才点亮）。按下开关时，Pin_Bidirectional 引脚会处于强驱动逻辑 '0' 状态。这样会打开 LED。因此，该项目阐述了同一个引脚（Pin_Bidirectional）上的两个驱动器：一个内部（逻辑 1）驱动器，一个需要使用输入的外部驱动器（开关）。

6.5 GPIO 输入/输出同步设置

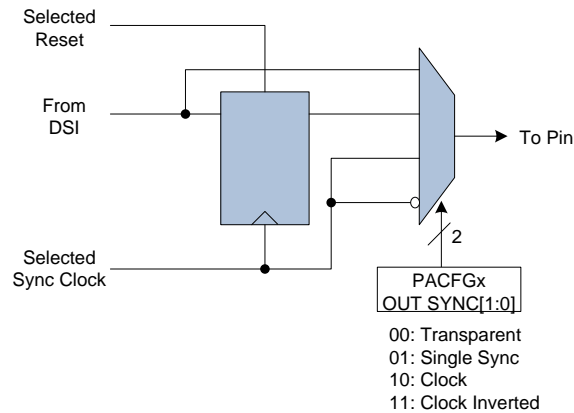
对于数字输入和输出信号，GPIO 会通过使用内部时钟（HFCLK）或一个用作 PSoC 4 系列器件（PSoC 4000 除外）时钟的数字信号进行同步。另外，它还提供了用于使能时钟和同步逻辑复位的配置。端口适配器逻辑用于实现 PSoC 4 器件内部的输入和输出同步，如图 50 和图 51 所示。

图 50. PSoC 4 器件内的输入同步



如图 50 所示，输入同步电路提供了不同步、单同步和双同步等选项。

图 51. PSoC 4 器件内的输出同步



而输出同步电路提供了不同步、单同步、时钟和时钟反转等选项，如图 51 所示。其中，时钟和时钟反转选项会将同步时钟连接到输出引脚上。这些选项是在引脚组件自定义程序中设置的，如图 52 所示。

可以将同步器时钟配置为 HFCLK、（DSI 的）外部信号或某个引脚信号。而同步器模块复位信号可被配置为（DSI 的）外部信号或某个引脚信号。这些选项也是在引脚组件自定义程序的“Clocking”选项卡中设置，如图 53 所示。

更多有关引脚组件自定义程序中“Clocking”选项的参数信息，请参考引脚组件数据手册。

图 52. 同步模式设置

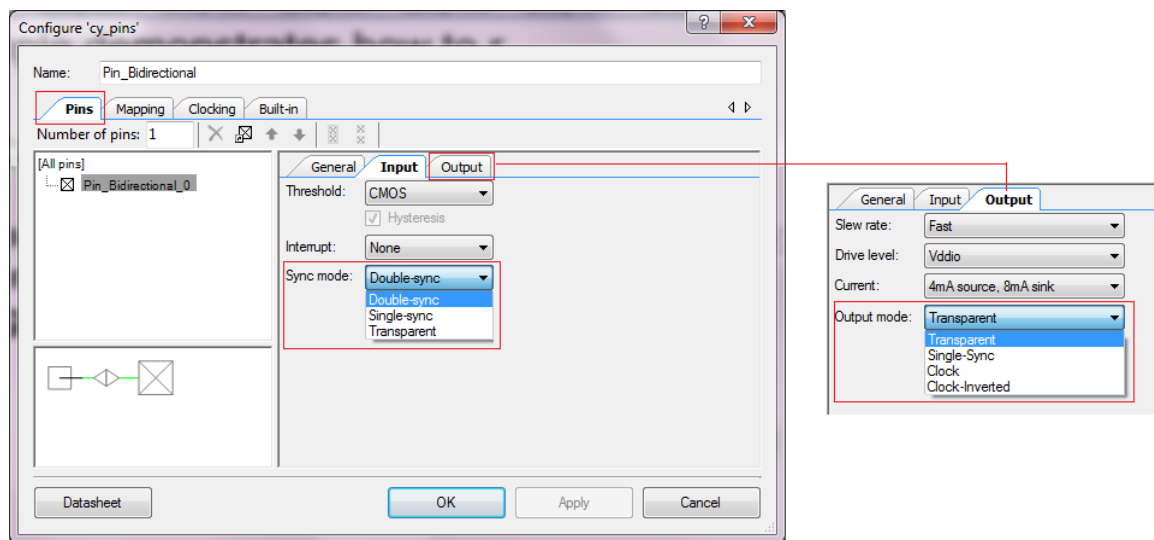
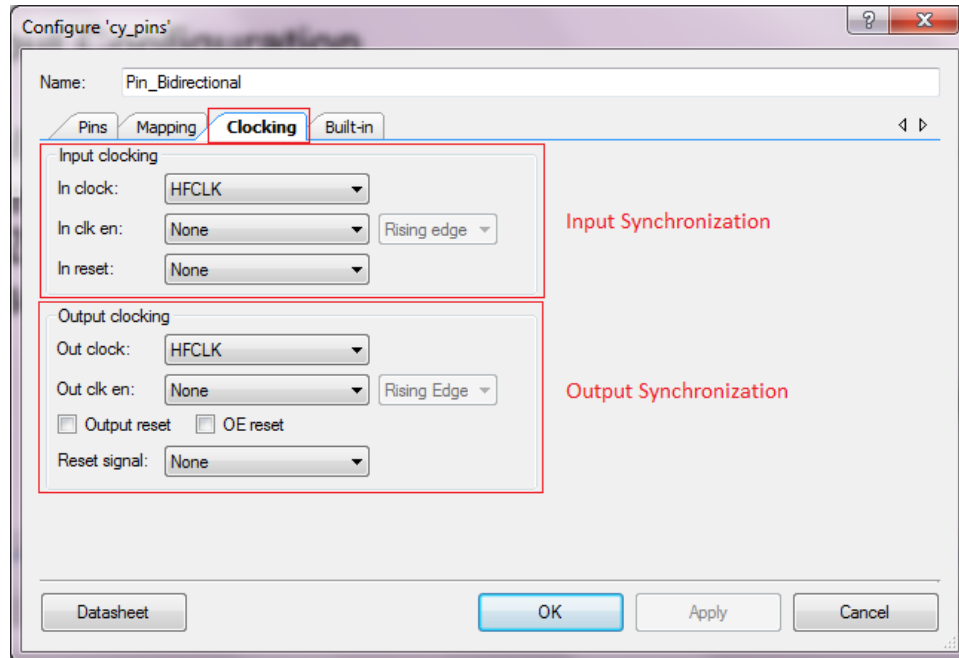


图 53. 时钟设置



可以通过组合使用 UDB 端口适配器和 GPIO 模块使各引脚信号同步。同样，在内部时钟同步情况下，端口中的所有引脚也共用时钟。更多有关信息，请参考 [PSoC 4 架构技术参考手册](#)。

注意： 由于端口 4 和更高的端口没有 UDB 端口适配器，所以不能对这些端口的信号进行同步处理。因而，这些端口引脚始终被用于不同步模式，以避免编译期间发生的错误。

下面两个示例演示了如何设置输入/输出同步。

6.5.1 GPIO 输入同步

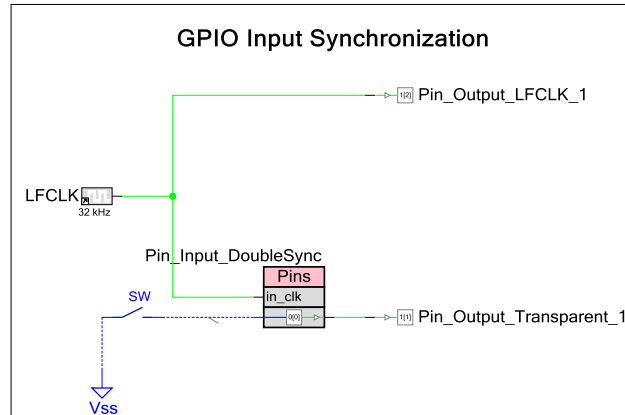
1. 将一个数字输入引脚、两个数字输出引脚和一个时钟组件放置到项目原理图中。根据表 6 配置它们。

表 6. 组件配置

组件	名称	配置
数字输入引脚	Pin_Input_DoubleSync	驱动模式：电阻上拉 同步模式：双同步 输入时钟（用于输入同步）：外部
数字输出引脚	Pin_Output_LFCLK_1	输出模式：不同步
数字输出引脚	Pin_Output_Transparent_1	输出模式：不同步
时钟	LFCLK	时钟类型：现有 时钟源：LFCLK

2. 连接各引脚并添加各片下组件，如图 54 所示。

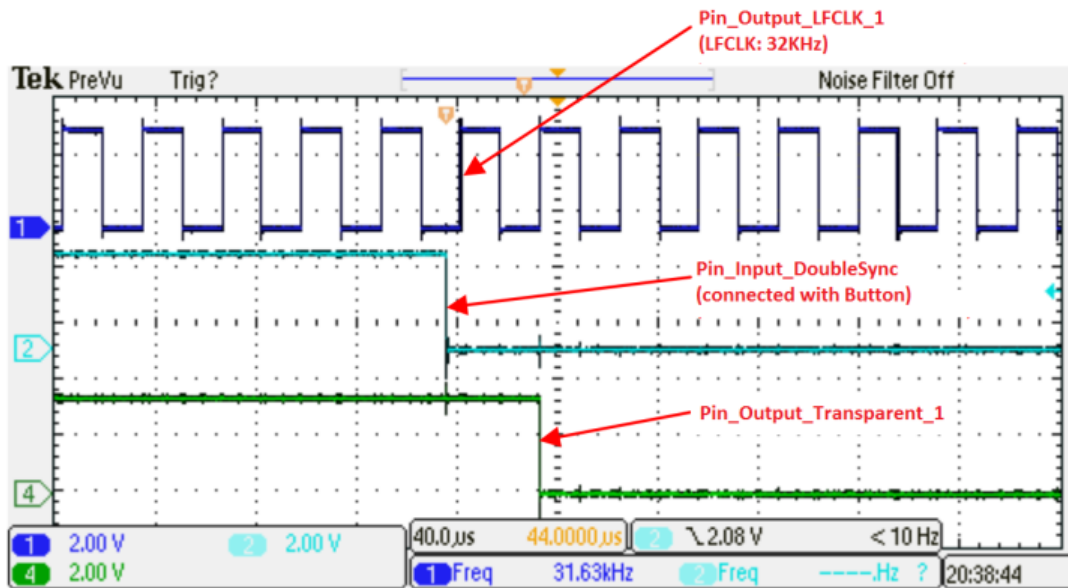
图 54. GPIO 输入同步的示例原理图



注意： 不能将 PSoC 4 中的时钟（SYSCLK 和 LFCLK 除外）直接连接到任何引脚终端。更多相关信息，请参考 [PSoC 4 架构技术参考手册](#) 中“时钟系统”一节。

3. 分配各引脚，并将 Pin_Input_DoubleSync 引脚连接到一个接地的开关。
4. 编译项目并将其编程到 PSoC 4 器件内。
5. 按下连接到 Pin_Input_DoubleSync 引脚的按键时，将出现信号波形，如图 55 所示。由于输入与 LFCLK 双重同步，因此 Pin_Output_Transparent_1 引脚将在 LFCLK 的第二个上升沿上变成低电平。

图 55. 输入/输出信号波形



6.5.2 GPIO 输出同步

6. 将一个数字输入引脚、三个数字输出引脚和一个时钟组件放置在项目原理图中，如图 56 所示，然后按照表 7 的内容进行配置。

图 56. GPIO 输出同步的示例原理图

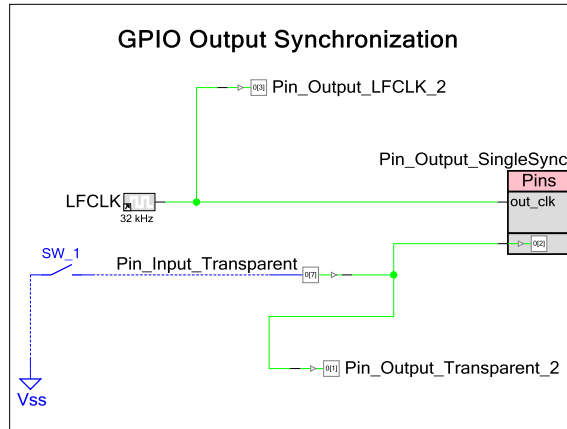


表 7. 引脚配置

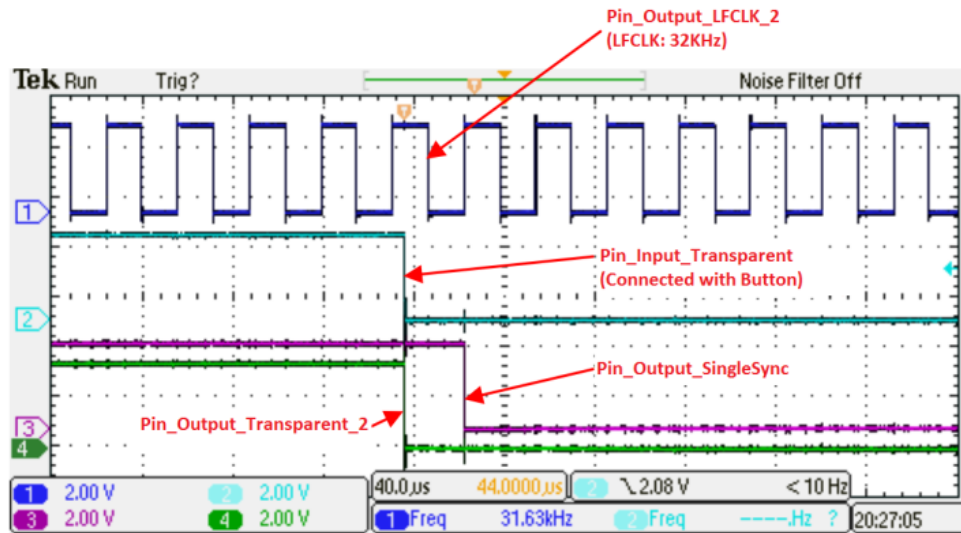
组件	名称	配置
数字输入引脚	Pin_Input_Transparent	驱动模式：电阻上拉 同步模式：不同步
数字输出引脚	Pin_Output_LFCLK_2	输出模式：不同步
数字输出引脚	Pin_Output_SingleSync	输出模式：单同步 输出时钟（用于输出同步）：外部
数字输出引脚	Pin_Output_Transparent_2	输出模式：不同步
时钟	LFCLK	时钟类型：现有 时钟源：LFCLK

7. 连接引脚，如图 56 所示。
8. 分配各引脚，并将 Pin_Input_Transparent 引脚连接到一个接地的开关。请注意，您不能为 Pin_Output_SingleSync 选择端口 4 和更高的端口引脚，因为它们不支持同步。
9. 编译项目并将其编程到 PSoC 4 器件内。

图 57 显示的是每次按下按键时出现的相应波形。

由于 Pin_Output_SingleSync 引脚与 LFCLK 同步，因此它将在 LFCLK 的下一个上升沿上变成低电平。由于不具备同步性，Pin_Output_Transparent_2 引脚将与 Input_Transparent 引脚同时变成低电平。

图 57. 输入/输出信号波形



6.6 通过数据寄存器能更快切换 GPIO

使用组件 API 是控制 GPIO 引脚的最简便方法；但并非最快的方法。

例如，将逻辑 ‘1’ 写入被映射到 P5[2] 的 Pin_1 引脚。API 函数调用如下：

```
Pin_1_Write(1);
```

可在项目工作区 “Results” 选项卡中列表文件 (*main.lst*) 内看到等效的汇编代码。

```
mov    r0, #1      ;load the value in r0
bl     Pin_1_Write ;call Pin_1_Write
```

Pin_1_Write 函数的组件 API 的汇编代码也能在列表文件中找到。

```
ldr    r3, .L2      ;load the address of Pin_1_DR into r3
ldr    r1, [r3]     ;load the value of Pin_1_DR into r1
mov    r2, #251     ;load 251 into r2 (value depends on location of pin
                    ;in 8 bit wide port, in this case, Pin_1 is on port P5[2])
and    r2, r1       ;AND the values of r2 and r1 and load result back in r2

lsl    r0, r0, #2    ;left shift r0 by two bits and load the result back in
                    ;r0 (this instruction is not present for the pin on LSB)
mov    r1, #4       ;load value of 4 into r1 (depends on the location of
                    ;pin in 8 bit wide port)
and    r0, r1       ;and the value of r0 (contains "value") and r1 and load
                    ;the result in r0
orr    r0, r2       ;or the value of r0 with r2 and load the result back in
                    ;r0
str    r0, [r3]     ;store the result back in Pin_1_DR
bx     lr           ;return to calling function
```

该代码需要 20 个 CPU 周期来将逻辑 ‘1’ 写入 Pin_1。

寄存器的定义和掩码位于每个引脚组件的 `<pin_name>.h` 文件中，使用它们可以快捷更新引脚。

下面语句将被映射到 P5[2] 的 Pin_1 设为逻辑 ‘1’。Pin_1_DR 是 Pin_1 的数据寄存器。

```
Pin_1_DR |= Pin_1_MASK
```

在列表文件 (`main.lst`) 中，上面指令转换成如下汇编代码：

```
ldr    r3, .L3      ;load the address of Pin_1_DR into r3
ldr    r1, [r3]     ;load value of Pin_1_DR into r1
mov    r2, #4       ;move value of 4 (Pin_1_MASK) into r2
orr    r2, r1       ;Set the bit in Pin_1_DR
strb   r2, [r3]     ;Store it back into Pin_1_DR
```

与组件 API（需要 20 个周期）相比，该代码仅要 8 个周期便可将逻辑 ‘1’ 写入 Pin_1。

执行组件 API 时，需要固件开销来实现下面各项操作（而在直接写入寄存器时，则不需要固件开销）：

- 调用函数
- 检查函数的参数，将引脚设为逻辑 ‘1’ 或 ‘0’
- 从函数返回

为了用户能够通过直接写入寄存器来设置、复位和读取引脚，PSoC Creator 中提供了下面各个宏：

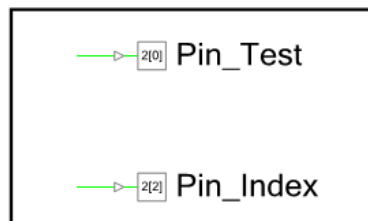
宏	说明
<code>CY_SYS_PINS_SET_PIN(portDR, pin)</code>	设置引脚的输出值，以将该引脚置为逻辑高电平 portDR 是端口数据寄存器的地址 pin 是引脚编号（0 到 7）
<code>CY_SYS_PINS_CLEAR_PIN(portDR, pin)</code>	清零引脚的输出值，以将该引脚置为逻辑低电平。 portDR 是端口数据寄存器的地址 pin 是引脚编号（0 到 7）
<code>CY_SYS_PINS_READ_PIN(portPS, pin)</code>	读取引脚值 portPS 是端口状态寄存器的地址 pin 是引脚编号（0 到 7）

有关这些宏的更多信息，请参见 *系统参考指南*（位于 PSoC Creator 帮助菜单中）。

请按照这些指导创建 PSoC Creator 项目（该项目可用于对调用 API 函数与直接写入寄存器的性能进行比较）：

1. 将两个数字输出引脚（硬件连接被禁用）放置在项目原理图中，并将其命名为 “Pin_Test” 和 “Pin_Index”，如图 58 所示。

图 58. 使用数据寄存器切换 GPIO 的示例原理图



2. 在.cydw 窗口中分配各引脚。
3. 将下面的代码添加到 *main.c* 文件中。该代码将 *Pin_Index* 设置为高电平，并使用组件 API 切换 *Pin_Test*。然后，它会将 *Pin_Index* 置低并通过数据寄存器（DR）切换 *Pin_Test*。

```

for(;;)
{
    /* Set IndexPin */
    Pin_Index_Write(1);

    /****** API Call *****/

    /* Set TestPin */
    Pin_Test_Write(1u);

    /* Clear TestPin */
    Pin_Test_Write(0u);

    /* do it again */
    Pin_Test_Write(1u);
    Pin_Test_Write(0u);

    /******

    /* Clear IndexPin */
    Pin_Index_Write(0);

    /****** Direct Register Writes *****/

    /* Set TestPin */
    CY_SYS_PINS_SET_PIN(Pin_Test__DR, Pin_Test_SHIFT);

    /* Clear TestPin */
    CY_SYS_PINS_CLEAR_PIN(Pin_Test__DR, Pin_Test_SHIFT);

    /* do it again */
    CY_SYS_PINS_SET_PIN(Pin_Test__DR, Pin_Test_SHIFT);
    CY_SYS_PINS_CLEAR_PIN(Pin_Test__DR, Pin_Test_SHIFT);

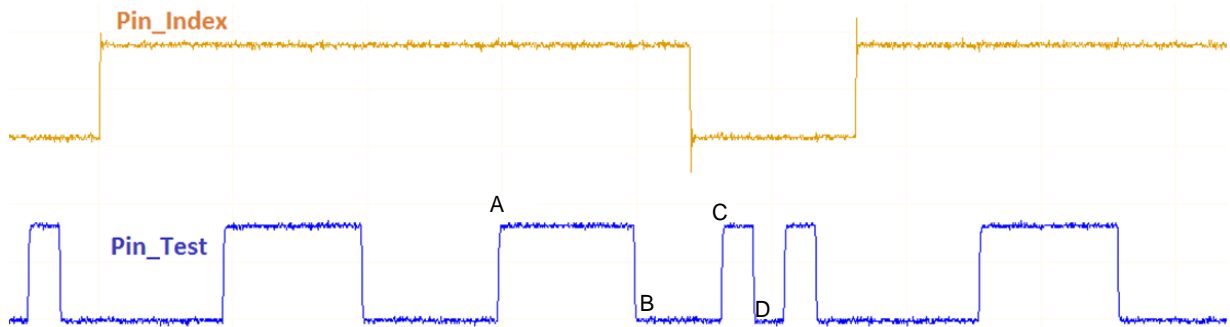
    /******
}
  
```

注意：*Pin_Test__DR* 是数据寄存器的地址；*Pin_Test_DR* 则是数据寄存器的值。为了了解数据寄存器地址的宏，请参考项目工作区的源文件中引脚组件.h 文件（在此为 *Pin_Test.h*）。

写入到数据寄存器的代码同 API 调用一样可以移植，因此如果在开发期间改变了引脚分配，便不需更改代码。

- 通过一个示波器观察这两个引脚的波形，如图 59 所示。

图 59. 输出信号波形



- A. Pin_Test_Write(1u)
- B. Pin_Test_Write(0u)
- C. CY_SYS_PINS_SET_PIN(Pin_Test__DR, Pin_Test_SHIFT)
- D. CY_SYS_PINS_CLEAR_PIN(Pin_Test__DR, Pin_Test_SHIFT)

图 59 显示的是与调用 API 函数相比，直接写入数据寄存器可以更快地切换引脚。

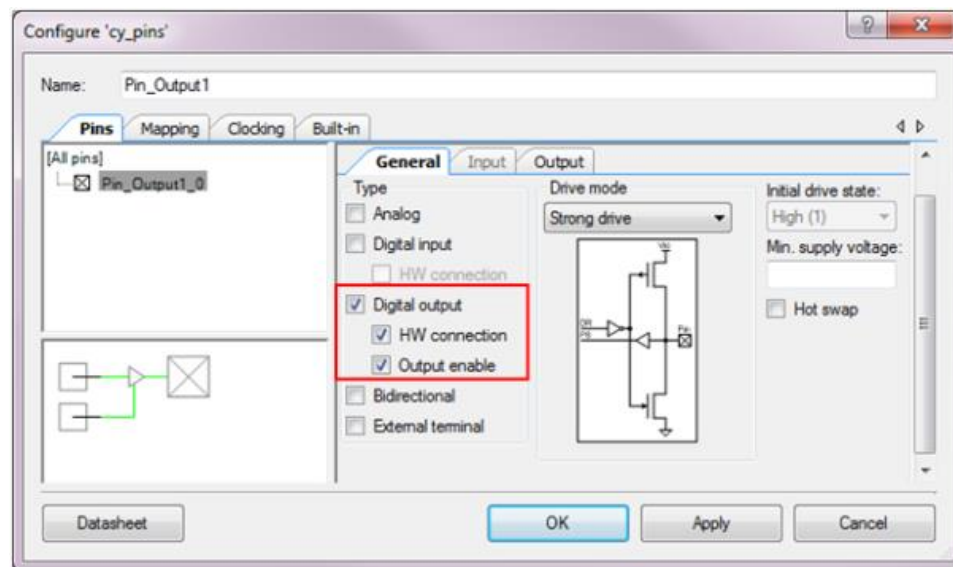
有关优化时间的编码技术，请参考应用笔记 [AN89610 — PSoC 4 和 PSoC 5LP ARM Cortex 代码优化](#)。

6.7 配置 GPIO 输出使能逻辑

本示例演示了如何配置和使用 GPIO 引脚的输出使能逻辑。该项目仅适用于 PSoC 4200、PSoC 42xx_BL、PSoC 4200M 和 PSoC 4200L 器件。

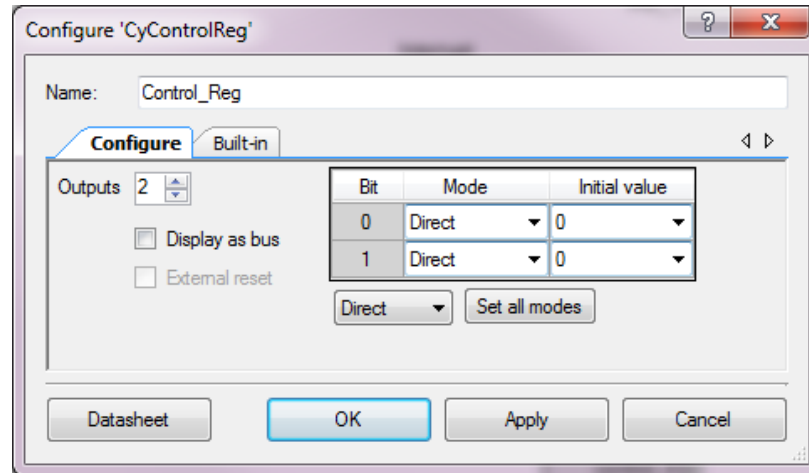
- 将两个数字输出引脚放置到项目原理图中。
- 打开每个引脚的配置对话框并勾选 **Output Enable** 选项，如图 60 所示。

图 60. 输出使能选择



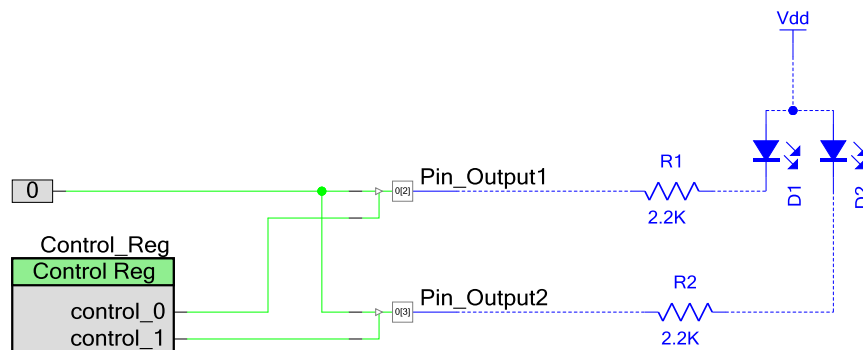
3. 在原理图中放置控制寄存器。
4. 为两个输出配置控制寄存器，如图 61 所示。

图 61. 配置两个输出的控制寄存器



5. 添加一个逻辑低 ‘0’ 组件。
6. 将该逻辑低组件连接到各引脚并为各个 LED 添加片下组件，如图 62 所示。

图 62. 控制寄存器驱动引脚的输出使能



7. 分配各引脚并将其连接到各个 LED。
8. 将下面的代码添加到 *main.c* 文件中。

```
uint8 count;

for(;;)
{
    for(count = 0u; count < 4u; count++)
    {
        /* Set Control_Reg Value */
        Control_Reg_Write(count);

        /* Delay for 500ms */
        CyDelay(500u);
    }
}
```

9. 编译项目并将其编程到 PSoC 4 器件内。

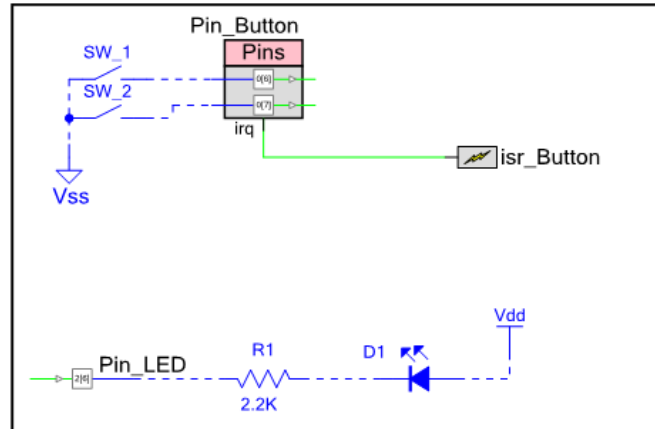
结果是两个引脚经过 Control_Reg 状态门控得到的输出，它能使各个 LED 闪烁发亮，直到 “count” 达到 3 为止。

6.8 引脚中断

该示例演示了如何通过组件 API 使用由同一个端口上的两个引脚所生成的中断。这两个引脚仅能使用一个 IRQ 终端。因此，必须将中断源定义在 ISR 中。

1. 在项目原理图中放置两个引脚：一个数字输入引脚和一个数字输出引脚（分别命名为 Pin_Button 和 Pin_LED）。
2. 将 Pin_Button 的引脚数量设置为 2、驱动模式置为电阻上拉、中断置为下降沿。这样便显示了 IRQ 终端。
3. 将中断组件连接到 IRQ 终端，如图 63 所示。

图 63. 引脚中断的示例原理图



4. 在.cydwr 窗口中分配各引脚。
5. 使用组件 API 来根据 Pin_Button 设置 LED 引脚的状态。复制下面的 *main.c* 代码：

```
#define LED_ON (0u)
#define LED_OFF (1u)

/* The flag to enter ISR_Button */
uint8 isrFlag = 0u;

/* The LED state */
uint8 ledState = LED_OFF;

/* ISR for ISR_Button */
CY_ISR(INT_ISR_Button)
{
    /* Set the flag */
    isrFlag = 1u;

    /* Check which pin caused interrupt by reading interrupt status register */
    if(Pin_Button_INTSTAT & (0x01u << Pin_Button_SHIFT))
    {
        /* Triggered by Pin_Button_0 */
        ledState = LED_OFF;
    }
    else
    {
        /* Triggered by Pin_Button_1 */
        ledState = LED_ON;
    }

    /* Clear interrupt */
    Pin_Button_ClearInterrupt();
}
```

```

}

int main()
{
    /* Start Pin ISR */
    isr_Button_StartEx(INT_ISR_Button);

    /* Enable global interrupt */
    CyGlobalIntEnable;

    for(;;)
    {
        /* Check the flag */
        if(0u != isrFlag)
        {
            /* Clear the flag */
            isrFlag = 0u;

            /* Drive the LED with ledState. Led State is updated in ISR */
            Pin_LED_Write(ledState);
        }

        /* Delay 1ms */
        CyDelay(1u);
    }
}

```

6.

在 *main.c* 代码中，`CY_ISR(INT_ISR_Button)` 是引脚中断的中断服务子程序。

7. 编译项目并将其编程到 PSoC 4 器件内。

结果为：在按下连接到 `Pin_Button_0` 的按键时，LED 关闭；按下连接到 `Pin_Button_1` 的按键时，LED 打开。但释放各按键时，LED 的状态将不变。（请注意，按键上发生的开关抖动可能导致按键按下一次会重复触发中断的现象；更多有关信息，请参考 [AN60024 — PSoC 3、PSoC 4 和 PSoC 5LP](#) 的开关去抖动和窄脉冲滤波。）

在 *main.c* 代码中，`Pin_Button_INTSTAT` 和 `Pin_Button_SHIFT` 是引脚组件所提供的函数和常量宏。它们用于确定引起中断的引脚。

`Pin_Button_ClearInterrupt()` 函数清除中断状态寄存器。

注意：并非所有端口都有专用中断。对于更高端口，将生成一个公用中断的信号。

请参考 [架构技术参考手册 \(TRM\)](#) 中的“中断”一节。

有关中断和写入中断处理程序的详细信息，请参考 [AN90799 — PSoC 4 中断](#)。

6.9 使用固件配置 GPIO 中断设置

GPIO 中断的动态配置是通过向中断配置寄存器的以下两位进行写操作来完成的。

- 对于 PSoC 4000 器件：GPIO_PRTx_INTR_CFG[2y+1:2y]
- 对于其他 PSoC 4 器件：PRTx_INTCFG[2y+1:2y]

其中，“x”表示端口编号，“y”表示引脚编号（请参考表 8）。您可以随时更改这些配置，以使能或禁用引脚中断。

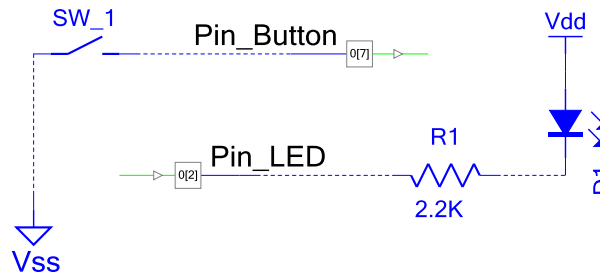
表 8. GPIO 中断类型和位设置

PRTx_INTCFG [2y+1:2y]	边沿类型	说明
0	禁用	中断被禁用
1	上升沿	在上升沿上触发中断
2	下降沿	在下降沿上触发中断
3	双边沿	在任意边沿上触发中断

在该示例中，Pin_Button 配置为在上升沿上触发中断。生成中断时，它会被配置为在下降沿上触发中断。某个中断被触发时，将切换一个 LED。

1. 将一个数字输入引脚和一个数字输出引脚放置在项目原理图中。为该 LED 和按键添加各个片下组件，如图 64 所示。

图 64. 示例原理图



2. 在.cydwf 窗口中为 Pin_Button 和 Pin_LED 分配引脚。
3. 将 Pin_Button 配置为电阻上拉引脚，并将它连接到一个按键。
4. 将 Pin_LED 配置为一个强驱动引脚，并将它连接到一个外部 LED。
5. 将下面的代码添加到 main.c 文件中。请注意，该项目使用了由 PSoC Creator 提供的 Pin_Button_INTCFG（在 cyfitter.h 文件中）进行中断配置，并未使用器件寄存器的名称。您不需考虑特定器件中寄存器的正确名称。这样有助于将项目转移到另一个 PSoC 4 器件时，无需进行任何代码更改。

```
#define INTERRUPT_MASK 0x03
#define RISING_EDGE 0x01
#define FALLING_EDGE 0x02

int main()
{
    /* Variable to save temporary data */
    uint32 regVal = 0x00u;

    /* Flag to switch interrupt type */
    uint8 edgeFlag = 0x00u;

    for(;;)
    {
```

```

/* Get value of port interrupt configuration register */
regVal = CY_GET_REG32(Pin_Button__INTCFG);

/* Clear the configuration bits for the Pin_Button.
Pin_Button_SHIFT is multiplied by 2 as two bits of the interrupt
configuration register sets the configuration for one pin */
regVal &= ~(INTERRUPT_MASK << (Pin_Button_SHIFT * 2));

if(edgeFlag)
{
    /* Set P0[7] to GPIO interrupt rising-edge trigger.
    Pin_Button_SHIFT is multiplied by 2 as two bits of the
    interrupt configuration register sets the configuration for
    one pin */
    CY_SET_REG32(Pin_Button__INTCFG, regVal | (RISING_EDGE <<
(Pin_Button_SHIFT * 2)));
}
else
{
    /* Set P0[7] to GPIO interrupt falling-edge trigger.
    Pin_Button_SHIFT is multiplied by 2 as two bits of the
    interrupt configuration register sets the configuration for
    one pin */
    CY_SET_REG32(Pin_Button__INTCFG, regVal | (FALLING_EDGE <<
(Pin_Button_SHIFT * 2)));
}

/* Toggle edgeFlag */
edgeFlag ^= 0x01u;

/* Wait for Interrupt */
while(!(CY_GET_REG32(Pin_Button__INTSTAT) & (0x01u <<
Pin_Button_SHIFT))) {}

/* Clear interrupt */
CY_SET_REG32(Pin_Button__INTSTAT, (0x01u << Pin_Button_SHIFT));

/* Toggle LED */
Pin_LED_Write(~Pin_LED_Read());
}
}

```

6. 编译项目并将其编程到 PSoC 4 器件内。

无论是按下还是释放按键，都会切换该 LED。按下按键时，会在下降沿触发中断；释放按键时，则会在上升沿触发中断。

PSoC 4 架构技术参考手册涵盖了更多有关 GPIO 中断的信息，包括模块框图和功能说明。另一个很好的参考资源是应用笔记 [AN90799 — PSoC 4 中断](#)。

6.10 在 GPIO 上同时使用模拟和数字功能

本示例演示了如何将引脚配置为模拟和数字功能，以及如何使用它。在本示例中，输出引脚交替受 IDAC 和固件的控制。由固件控制时，LED 将闪烁。由 IDAC 控制时，LED 将逐渐点亮。

如果您需要在同一个引脚上使用模拟和数字功能，该复用类型便非常有用。同时，它还可以减少设计中所需的 GPIO 引脚数量。

另外，您可以使用硬件连接取代固件，来控制数字输出。请参考本章节最后部分介绍的内容，了解需要对项目进行的修改。

为了配置引脚信号源，需要更新 HSIOM_PORT_SELx 寄存器。与前面示例一样，本项目使用了引脚组件中所定义的寄存器名称，这样便于在 PSoC 4 器件系列间实现移植。

请按照下面步骤创建原理图和固件：

1. 将一个模拟引脚和一个电流 DAC 放置在原理图中。
2. 将引脚组件分配给一个物理引脚（本示例中使用了 P0[2]）。
3. 同时将引脚配置为 **Analog** 和 **Digital Output**，如图 65 所示。
4. 将 IDAC 的 **Polarity** 设置为 **Negative (Sink)**，如图 66 所示。将 IDAC 连接到模拟引脚，如图 67 所示。
5. 编译项目，从而创建所需 API。

图 65. LED 引脚被配置为模拟和数字功能

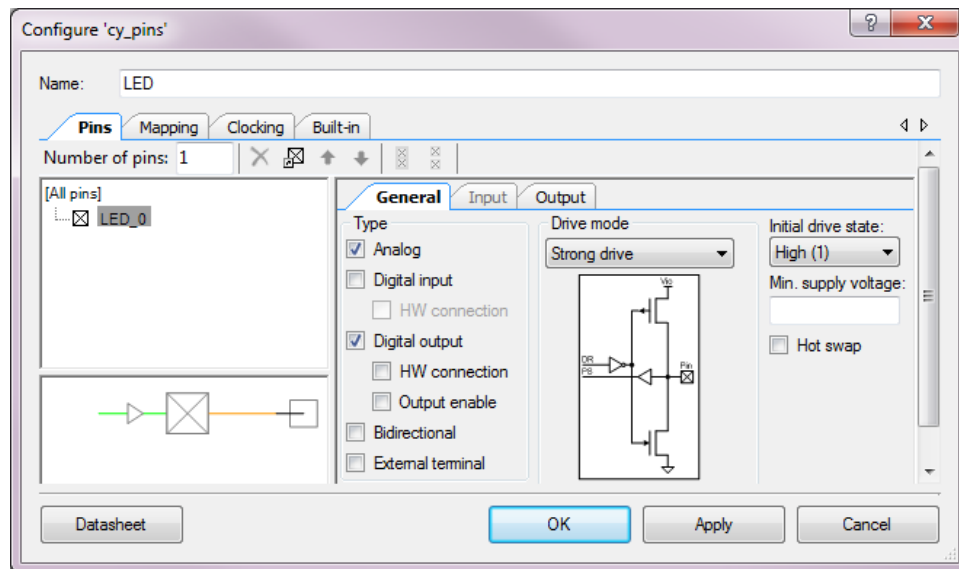


图 66. IDAC 设置

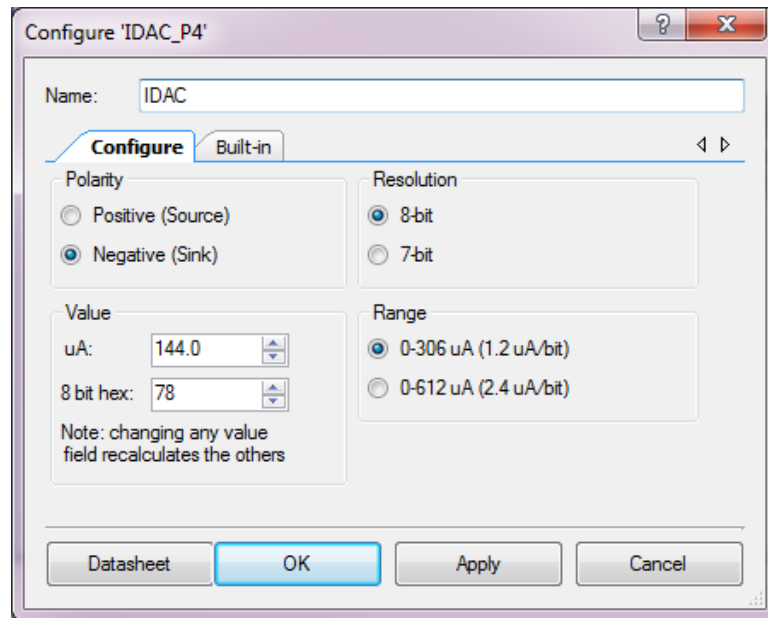
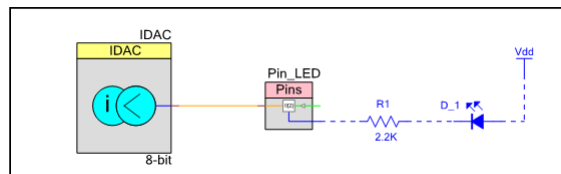


图 67. PSoC Creator 的模拟和数字切换原理图



- 将下面代码添加到 **main.c** 文件中并重新编译项目。使用所生成的十六进制文件编程器件。请注意，下面代码中也使用了定义在引脚组件和 **Cyfitter.h** 中的宏。

```
#define HSIOM_SW_GPIO 0x00
#define HSIOM_AMUX_BUS_A 0x06

int main()
{
    uint32 i = 0u;
    uint32 regVal = 0x00u;

    /* Disable Input Buffer */
    Pin_LED_INP_DIS |= (0x01u << Pin_LED_SHIFT);

    /* Start IDAC */
    IDAC_Start();

    for(;;)
    {
        /* Get the current value of HSIOM_PORT_SEL0 register */
        regVal = CY_GET_REG32(Pin_LED__0__HSIOM);
        regVal &= ~Pin_LED__0__HSIOM_MASK;

        /* Set LED Pin as GPIO controlled by firmware */
        regVal = CY_SET_REG32(Pin_LED__0__HSIOM, regVal | (HSIOM_SW_GPIO <<
        Pin_LED__0__HSIOM_SHIFT));
    }
}
```

```

    /* Set LED Pin to Strong Drive Mode */
    Pin_LED_SetDriveMode(Pin_LED_DM_STRONG);

    for(i= 0u; i < 5u; i++)
    {
        /* Toggle LED with 100-ms delay */
        Pin_LED_Write(0u);
        CyDelay(100u);
        Pin_LED_Write(1u);
        CyDelay(100u);
    }

    /* Get the current value of HSIOM_PORT_SEL0 register */
    regVal = CY_GET_REG32(Pin_LED__0__HSIOM);
    regVal &= ~Pin_LED__0__HSIOM_MASK;

    /* Connect LED Pin to AMUXBUS-A */
    CY_SET_REG32(Pin_LED__0__HSIOM, regVal | (HSIOM_AMUX_BUS_A <<
Pin_LED__0__HSIOM_SHIFT));

    /* Set LED Pin to High Impedance-Analog Drive Mode */
    Pin_LED_SetDriveMode(Pin_LED_DM_ALG_HIZ);

    for(i = 0u; i < 0x7fu; i++)
    {
        /* Adjust LED brightness */
        IDAC_SetValue(i);

        /* Delay 20 ms */
        CyDelay(20u);
    }
}

```

这样会使输出交替受固件和 IADC 的控制。

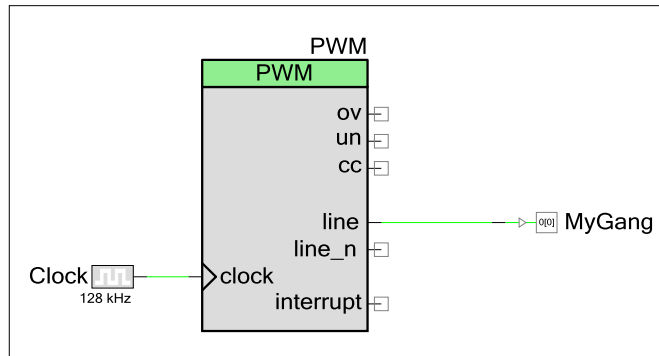
您可以很轻松地修改该项目，从而可以使用硬件连接取代固件，以控制数字输出。为了实现该操作，应在引脚配置窗口中使能 **HW connection** 项（步骤 3）。然后，您可以将一个数字资源连接到该引脚上。为了选择该数字资源作为引脚输出，建议您使用 **HSIOM_PORT_SEL** 寄存器将该引脚设置为由 **DSI** 路由的 **GPIO** 或一个引脚专用数字资源连接。更多有关信息，请参考 [PSoC 4 架构技术参考手册](#)。

6.11 组合引脚以获得更大的源电流/灌电流

为了提高电路的总源电流/灌电流，可以将各 GPIO 引脚组合使用（短接在一起）。本示例演示了如何使用四个 GPIO 引脚路由 PWM 信号。该项目仅适用于 PSoC 4200、PSoC 42xx_BL、PSoC 4200M 和 PSoC 4200L 器件。

1. 在原理图中，放置并配置一个 PWM（TCPWM 模式）和一个时钟组件。
2. 放置一个数字输出引脚组件。
3. 连接各组件，如图 68 所示。

图 68. PWM 信号路由到一个引脚



4. 打开引脚配置对话框，并设置相应的引脚数量，如图 69 所示。本示例使用了四个 GPIO 引脚。将 **Output Mode** 设置为 **Single-Sync**，并将 **Out Clock** 置为 **External**，如图 70 和图 71 所示。

注意： 通过同步输出可以避免不同引脚上发生不同的输出信号延迟。

图 69. 在组件中配置多个引脚

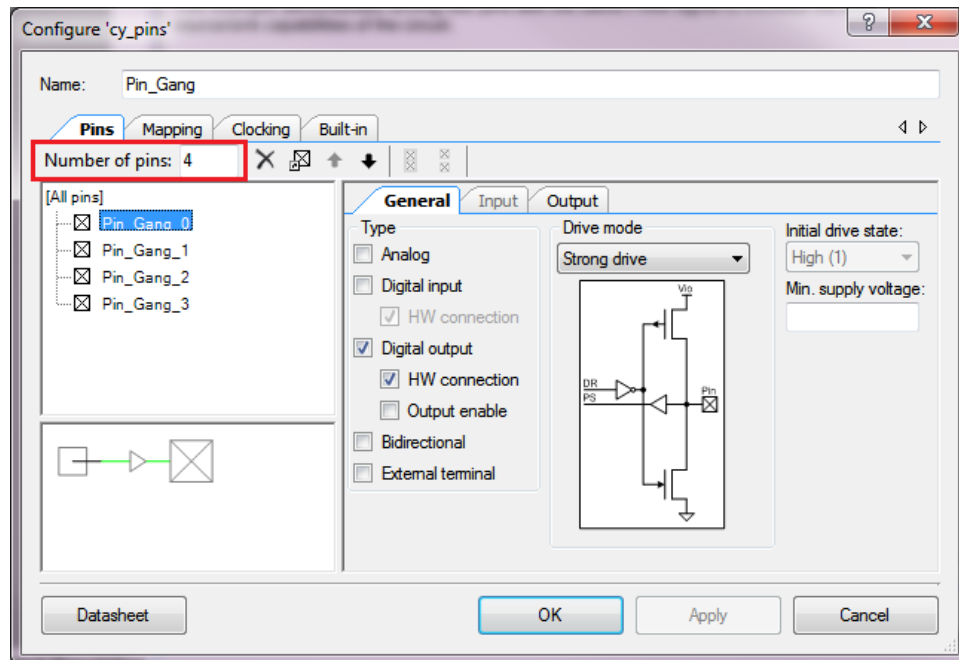


图 70. 输出模式设置

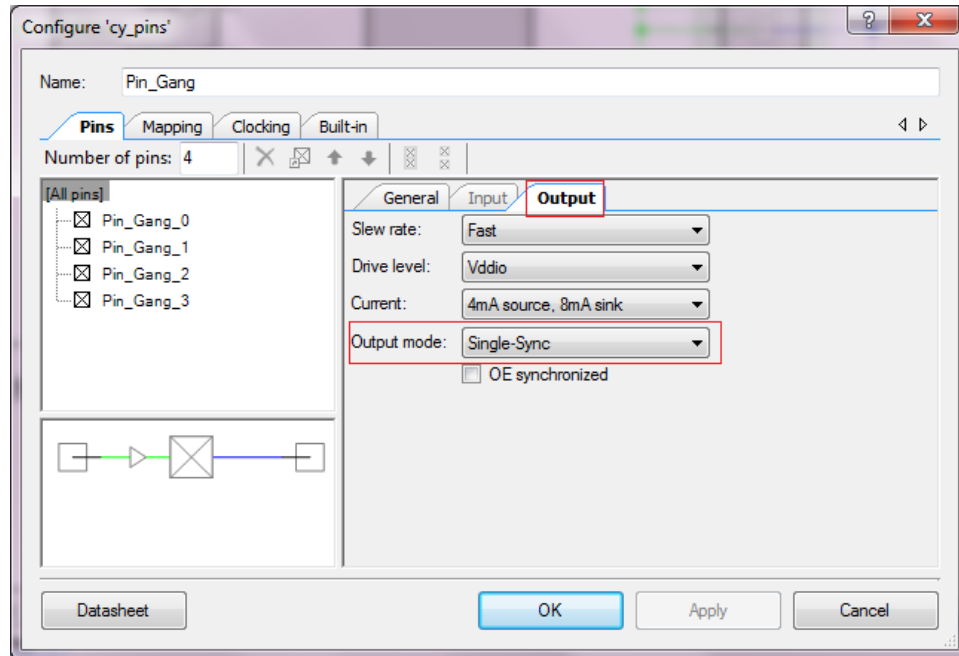
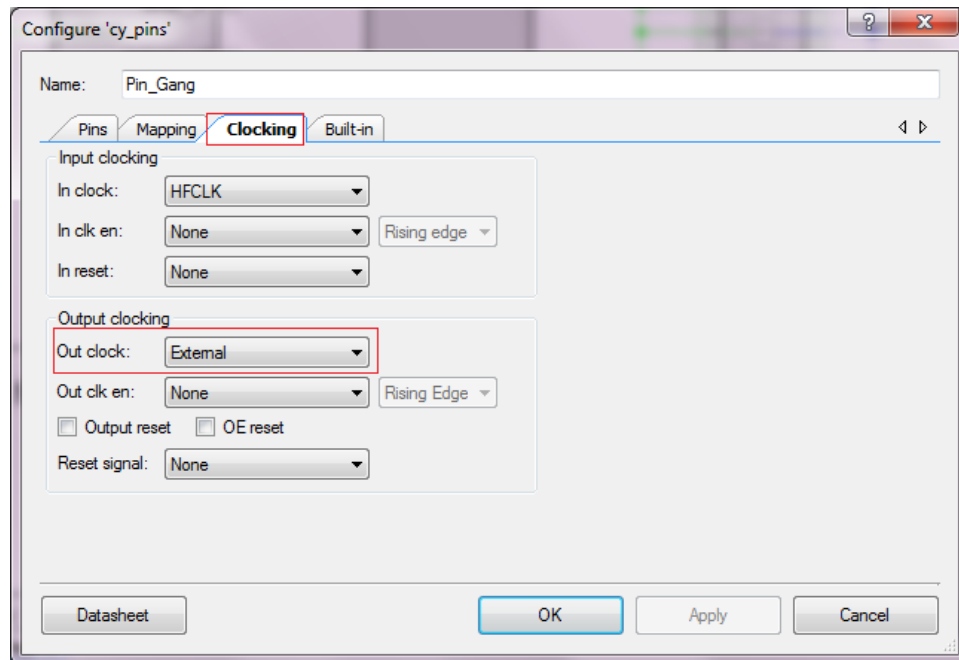
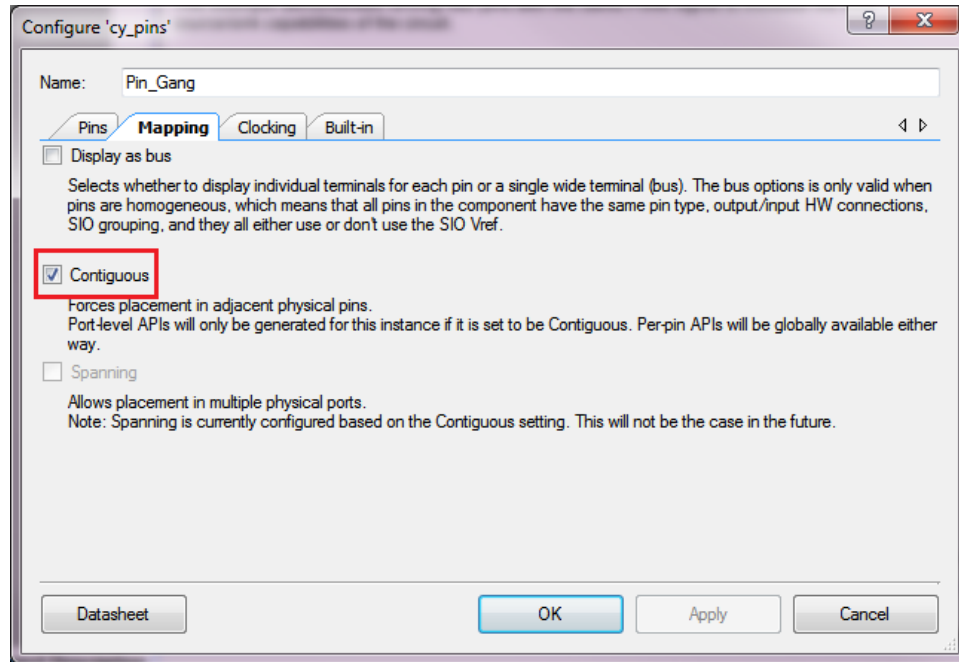


图 71. 输出时钟设置



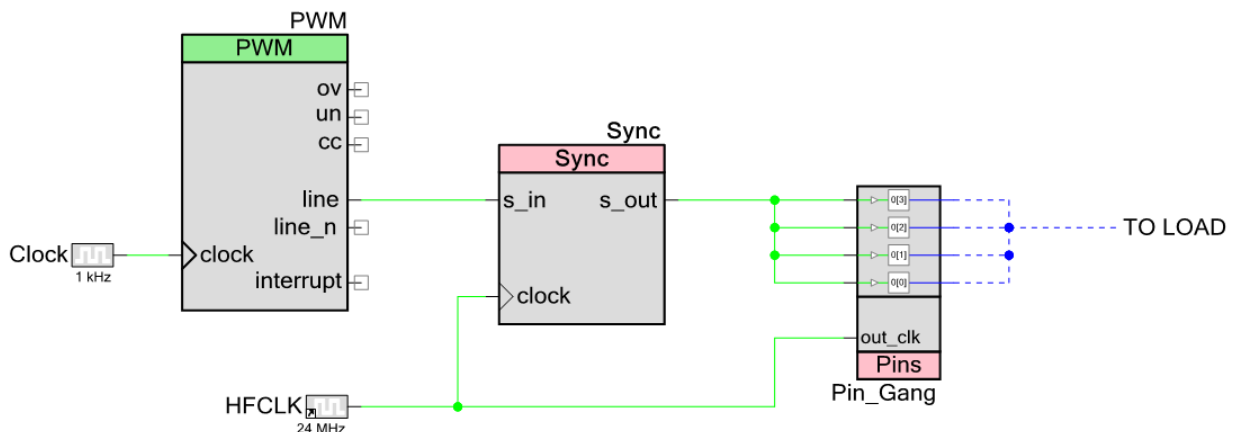
5. (可选的) 将引脚映射设置为 **Contiguous**，以便于进行 PCB 路由，如图 72 所示。

图 72. 使能连续映射



6. 将引脚组件分配给物理引脚。
 7. 放置一个同步组件，并通过该组件将信号源（在本示例中为 PWM）连接至每一个引脚终端。放置另一个时钟组件，并将高频时钟（HFCLK）设置为它的时钟源。将引脚组件的 `out_clk` 终端和同步组件的时钟终端连接到 HFCLK。
- 请注意，使用高频同步时钟有助于降低引脚信号延迟的差异。需要使用同步组件对跨时钟域传输的信号进行同步。在本示例中，PWM 输出从时钟（1 kHz）域传输到 HFCLK。

图 73. PWM 信号路由到四个引脚



8. 编译项目并将其编程到 PSoC 4 器件内。
9. 在全部四个 GPIO 上驱动 PWM 的输出。可以在 PCB 板上对这些引脚进行外部短接，并根据要求将其连接到外部电路。

6.12 深度睡眠模式下的控制寄存器处理

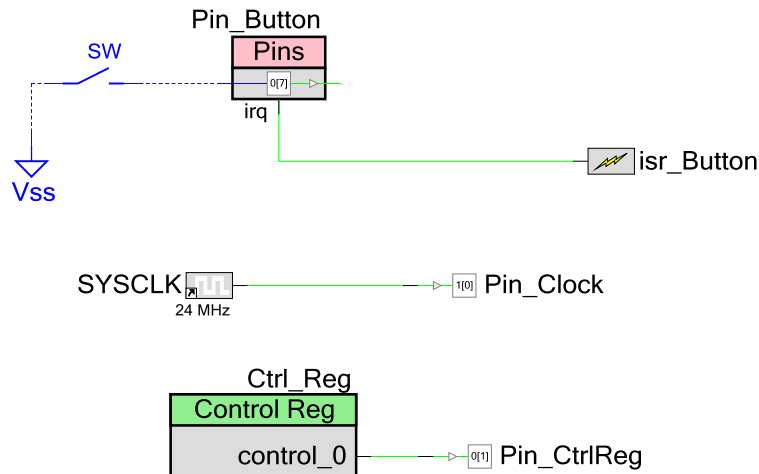
该示例演示了在低功耗模式下对 GPIO 进行冻结，以防止输出发生窄脉冲。例如，假设一个控制寄存器驱动一个引脚。器件处于深度睡眠模式时，所有 I/O 将在没有用户干涉的条件下被冻结。器件唤醒时，所有 I/O 都将自动恢复到原始的配置状态。然而，控制寄存器在深度睡眠模式下将丢失它的数据。I/O 解冻前，需要将其恢复。否则，在输出上将生成窄脉冲。PSoC 4 通过 `CySysPmFreezeIo()` 和 `CySysPmUnfreezeIo()` API 对 GPIO 的冻结和解冻情况提供了控制。请实现以下各步骤，来创建 PSoC Creator 项目。该项目仅适用于 PSoC 4200、PSoC 42xx_BL、PSoC 4200M 和 PSoC 4200L 器件。

1. 将一个数字输入引脚、两个数字输出引脚、一个时钟、一个控制寄存器和一个中断组件放置在原理图中。
2. 按照表 9 配置各组件。连接各组件，如图 74 所示。

表 9. 组件配置

组件	名称	配置
数字输入引脚	Pin_Button	驱动模式：电阻上拉 中断：上升沿
数字输出引脚	Pin_Clock	默认配置
数字输出引脚	Pin_CtrlReg	默认配置
时钟	SYSCLK	时钟源：SYSCLK
中断	isr_Button	默认配置
控制寄存器	Ctrl_Reg	输出数量：1 初始值：1

图 74. 避免退出深度睡眠模式时发生的窄脉冲



3. 将下面的代码添加到 `main.c` 文件中。

```

/* Set FREEZE_IO to 0x01 to avoid glitch by enabling the GPIO freeze */
/* else set it to 0 */
#define FREEZE_IO 0x01

/* The flag to enter ISR */
uint8 isrFlag = 0u;
CY_ISR(ISR_Handle)
{
    /* Set the flag */
    isrFlag = 1u;

    /* Clear pin interrupt */

```

```

    Pin_Button_ClearInterrupt();
}

int main()
{
    /* This variable is used as backup for Control register value */
    uint8 ctrlRegVal = 0u;

    /* Clear the flag */
    isrFlag = 0u;

    /* Start the ISR */
    isr_Button_StartEx(ISR_Handle);

    CyGlobalIntEnable;

    /* Set Control register output as high */
    Ctrl_Reg_Write(1u);

    for(;;)
    {
        /* If freeze flag is set */
        if(0u != isrFlag)
        {
            /* Clear isr flag set in GPIO Interrupt Handler */
            isrFlag = 0u;

            /* Rewrite the value */
            Ctrl_Reg_Write(ctrlRegVal);

            #if(FREEZE_IO)
                /* Unfreeze I/O */
                CySysPmUnfreezeIo();
            #endif
        }

        /* Delay 200us */
        CyDelayUs(200u);

        /* Store the value of Control register */
        ctrlRegVal = Ctrl_Reg_Read();

        #if(FREEZE_IO)
            /* Freeze I/O */
            CySysPmFreezeIo();
        #endif

        /* Enter Deep-Sleep mode */
        CySysPmDeepSleep();
    }
}

```

为了禁用冻结选项，将 **FREEZE_IO** 设置为 0。编译项目并将其编程到器件内。按下和释放按键时，可在 **Pin_CtrlReg** 引脚上看到窄脉冲，如图 75 所示。为了使能冻结选项，将 **FREEZE_IO** 置为 1。编译项目并将其编程到器件内。此时，I/O 被冻结，并且没有发生任何窄脉冲，如图 76 所示。

注意：并非所有端口都有专用中断。对于更高端口，将生成一个公用中断的信号。

请参考架构技术参考手册（TRM）中的“中断”一节。

图 75. 输出信号波形（不冻结 I/O）

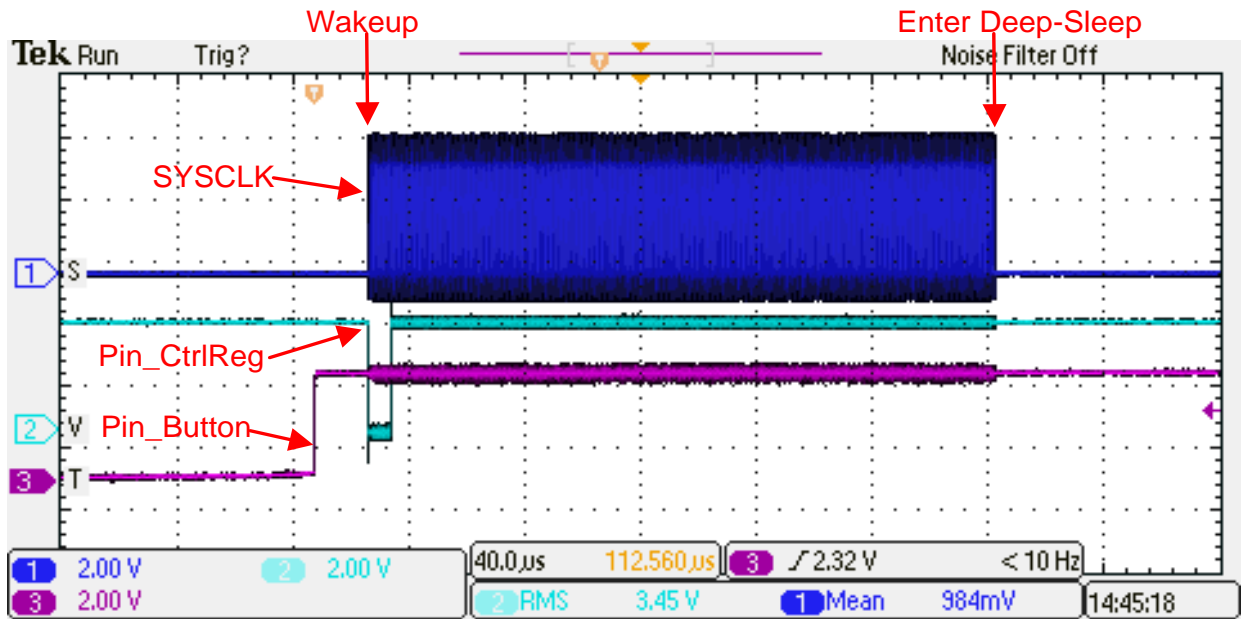
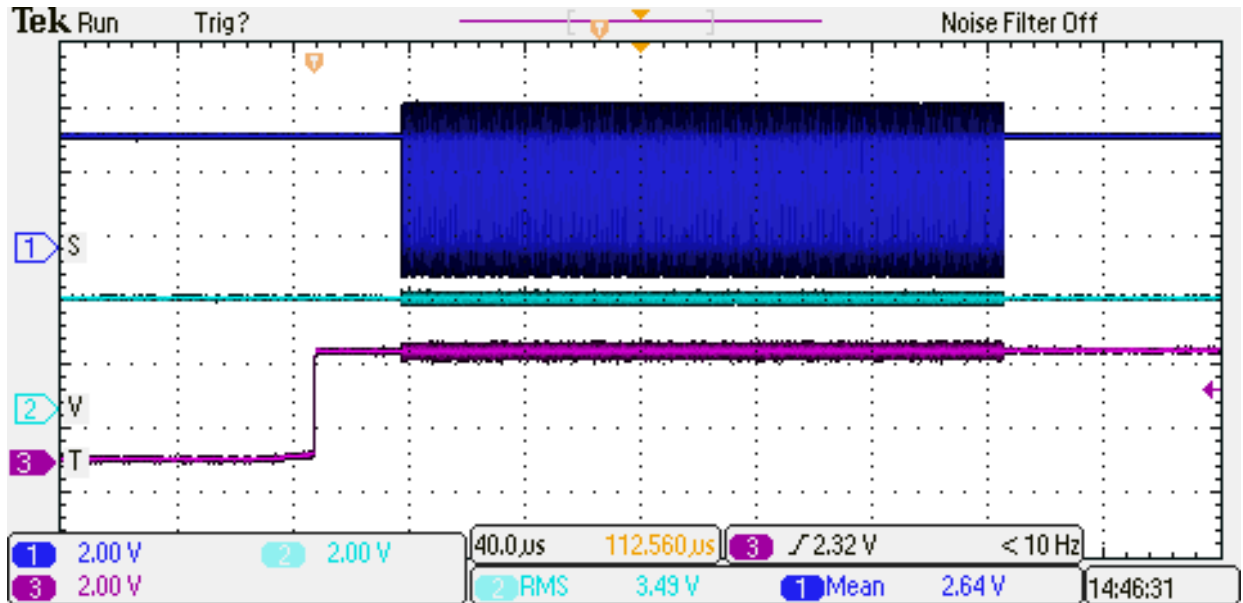


图 76. 输出信号波形（冻结 I/O）



7 相关应用笔记

- [AN79953 — PSoC 4 入门](#)
- [AN86233 — PSoC 4 低功耗模式和降低功耗技术](#)
- [AN60024 — PSoC 3、PSoC 4 和 PSoC 5LP 的开关去抖动和窄脉冲滤波](#)
- [AN72382 — 使用 PSoC 3 和 PSoC 5LP GPIO 引脚](#)
- [AN90799 — PSoC 4 中断](#)
- [AN2094 — PSoC 1 GPIO 入门](#)
- [AN89610 — PSoC® 4 和 PSoC 5LP 的 ARM Cortex 代码优化](#)

8 总结

本应用笔记介绍了如何通过配置 PSoC Creator 中的引脚组件来有效使用 GPIO 引脚的各种基本功能。

9 关于作者

姓名: Charles Cheng

职务: 应用工程师

背景: Charles 是赛普拉斯半导体可编程系统部的应用工程师，重点工作领域是 PSoC 应用。

姓名: Rajiv Badiger

职务: 应用工程师

背景: Rajiv 是赛普拉斯半导体可编程系统部的应用工程师，重点工作领域是 PSoC 应用。

附录A. PSoC 4 GPIO 与 PSoC 1、PSoC 3 和 PSoC 5LP GPIO 相比较

PSoC 4 GPIO 与 PSoC 1、PSoC 3 和 PSoC 5LP 的 GPIO 不一样；更多有关信息请参考[表 10](#)。

表 10. PSoC 4 GPIO 与 PSoC 1、PSoC 3 和 PSoC 5LP GPIO 相比较

GPIO 功能	PSoC 1	PSoC 3	PSoC 4	PSoC 5LP
CapSense	√	√	√	√
LCD Segment 驱动	√	√	√*	√
八种驱动模式	√	√	√	√
POR 状态配置	×	√	×	√
单独端口 DR 和 PS	×	√	√	√
输入/输出同步	×	Bus_clk	HFCLK, 外部时钟*	Bus_clk

*PSoC 4000 器件不支持

附录B. PSoC 4 开发板

您可以在以下的赛普拉斯开发电路板上测试本应用笔记所附带的 PSoC Creator 项目。

器件系列	开发电路板
PSoC 4000	CY8CKIT-040 PSoC 4000 Pioneer 开发套件
PSoC 4000S / 4100S	CY8CKIT-041 PSoC® 4 S 系列 Pioneer 套件
PSoC 4100S Plus	CY8CKIT-149 PSoC 4100S Plus 原型开发套件
PSoC 4100PS	CY8CKIT-147 PSoC 4100PS 原型开发套件
PSoC 4100/PSoC 4200	CY8CKIT-042 PSoC® 4 Pioneer 套件 PSoC 4 CY8CKIT-049 4xxx 原型开发套件
PSoC 42x7_BL	CY8CKIT-042-BLE 低功耗蓝牙 (BLE) Pioneer 套件
PSoC 4200M	CY8CKIT-044 PSoC® 4 M 系列 Pioneer 套件
PSoC 4200L	CY8CKIT-046 PSoC® 4 L 系列 Pioneer 套件

文档修订记录

文档标题: AN86439 — PSoC® 4 — 使用 GPIO 引脚

文档编号: 001-97880

版本	ECN	提交日期	变更说明
**	4802437	07/01/2015	本文档版本号为 Rev**, 译自英文版 001-86439 Rev*A。
*A	5012994	11/27/2015	本文档版本号为 Rev*A, 译自英文版 001-86439 Rev*B。
*B	5255824	05/10/2016	本文档版本号为 Rev*B, 译自英文版 001-86439 Rev*D。
*C	5716244	05/25/2017	更新徽标和版权。
*D	6651885	03/31/2020	本文档版本号为 Rev*D, 译自英文版 001-86439 Rev*I。

全球销售和设计支持

赛普拉斯公司拥有一个由办事处、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/pmuc
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](#) | [代码示例](#) | [项目](#) | [视频](#) | [博客](#) | [培训](#) | [组件](#)

技术支持

cypress.com/support

此处引用的所有其它商标或注册商标都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，2014-2020 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归赛普拉斯所有。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。如果软件没有附带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适用于个人的、非独占性、不可转让的许可（无转授许可权）（1）在版权保护下的软件（a）以源代码形式提供的软件，只能是在组织内部为了使用赛普拉斯的硬件去修改和复制。（b）以二进制代码形式从外部发到终端用户（直接或间接通过经销商和分销商），仅用于赛普拉斯硬件产品单元。（2）在软件（由赛普拉斯公司提供，且未经修改）侵犯赛普拉斯专利的权利主张下，仅许可在赛普拉斯硬件产品上制造、使用、提供和导入软件。禁止对软件的任何其他使用、复制、修改、翻译或编译。

赛普拉斯不在此材料提供任何类型的明示或暗示保证，包括但不限于针对特定用途的适销性和适用性的暗示保证。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。在适用法律允许的范围内，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。贵方应赔偿并保护赛普拉斯免受所有索赔的损害，包括因人身伤害或死亡引起的索赔、费用、损失和其它责任。

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。