请注意赛普拉斯已正式并入英飞凌科技公司。
此封面页之后的文件标注有“赛普拉斯”的文件即该产品为此公司最初开发的。请注意作为英飞凌产品组合的部分，英飞凌将继续为新的及现有客户提供该产品。

文件内容的连续性
事实是英飞凌提供如下产品作为英飞凌产品组合的部分不会带来对于此文件的任何变更。未来的变更将在恰当的时候发生，且任何变更将在历史页面记录。

订购零件编号的连续性
英飞凌继续支持现有零件编号的使用。下单时请继续使用数据表中的订购零件编号。

AN72382

使用 PSoC® 3 和 PSoC 5LP GPIO 引脚

作者：Greg Reynolds

相关项目：无

相关器件系列：所有 PSoC® 3 和 PSoC 5LP 器件

软件版本：PSoC Creator™ 2.1 SP1 及更高版本

相关应用笔记：要想获取相关应用笔记的完整列表，请点击此处。

AN72382 向您介绍了如何有效地使用 PSoC® 3 和 PSoC 5LP 中的 GPIO 引脚。主要主题包括 GPIO 基础、设置、混合信号使用、寄存器、中断和以及低功耗的特性。

目录

简介

PSoC® 3 和 PSoC 5LP 的 GPIO 可以将任意信号路由至任意一个引脚，这样有助于优化 PCB 布局，缩短设计时间，并能实现无焊料返工。然而，由于其灵活性，造成学习这种器件比学习传统的微控制器更难。此应用笔记向您介绍了 PSoC 3 和 PSoC 5LP GPIO 的基本知识，并演示了其有效的使用方法。

该文档假设您已经熟悉了 PSoC Creator™、PSoC 3 和 PSoC 5LP 系列器件架构。如果您刚刚接触 PSoC，则可以参阅 AN54181 — PSoC 3 入门、以及 AN77759 — PSoC 5LP 入门，以了解这些产品。如果您尚未了解 PSoC Creator，请参考 PSoC Creator 主页中的内容。

要获取相关 PSoC 设计资源的列表，请查看相关应用笔记一节。

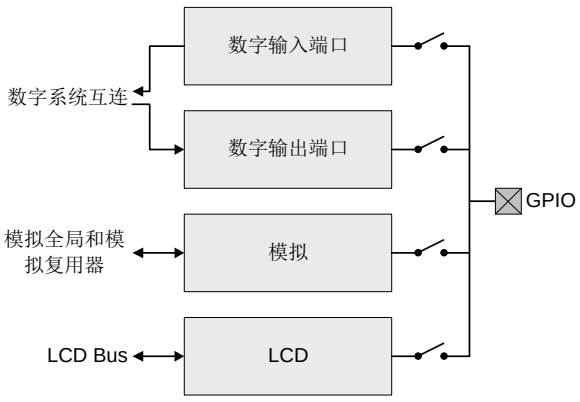
GPIO 引脚的基本知识

在 PSoC 3 和 PSoC 5LP 器件中，GPIO、SIO 和 USB 引脚是相似的。与 GPIO 引脚不同，SIO 和 USB 引脚有具有不同的驱动强度和特定于应用的特性。有些 GPIO 引脚还有另外专用功能，例如运算放大器的输入和输出、编程和调试接口，或者 DAC 输出。如果不使用特殊功能，则所有 GPIO 引脚的行为均相同。根据封装类型，PSoC 器件最多可具有 62 个 GPIO 引脚。

GPIO 引脚的物理结构

GPIO 引脚有八种驱动模式，用以支持 PSoC 提供的多种模拟和数字 I/O 功能。PSoC 3 架构技术参考手册（TRM）以及 PSoC 3 和 PSoC 5LP 系列的数据手册中提供了 GPIO 结构的详细框图。图 1 显示的是一个简化的版本。

图 1. 简化的 GPIO 框图

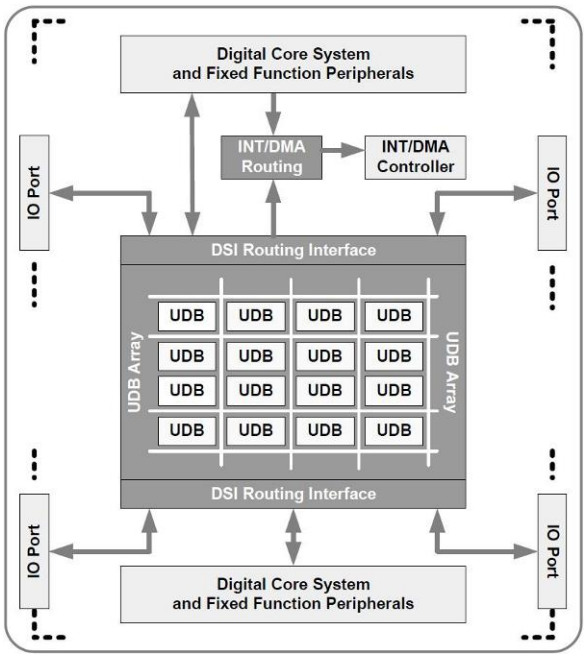


引脚组件数据手册详细描述了各种驱动模式以及它们的自定义设置。本手册由 PSoC Creator 提供，您也可以通过赛普拉斯网站下载。

数字系统互连概述

PSoC 3 和 PSoC 5LP 数字子系统拥有可编程的互联性能，因此能够在内置的外围设备、定制逻辑功能（通用数字模块，又称为 UDB）以及任意 I/O 引脚间进行连接。通过数字系统互连（DSI）的布线接口可使 GPIO 引脚与芯片上任何数字资源相连，如图 2 所示。

图 2. DSI 框图

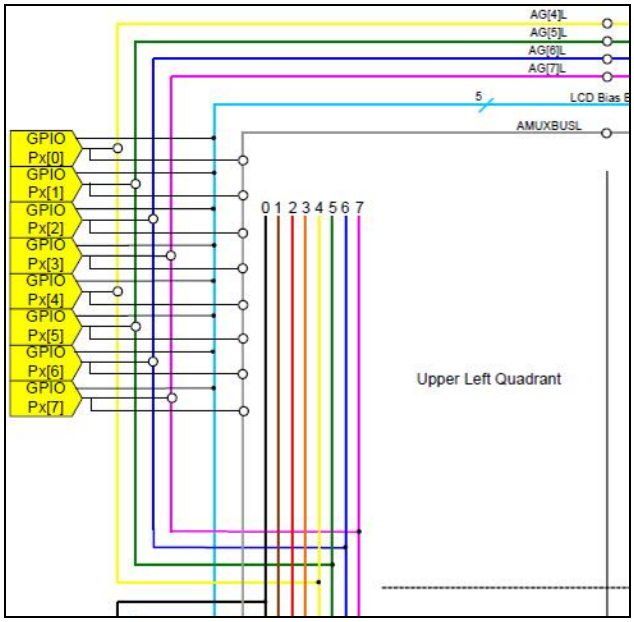


所有数字资源均被路由至 DSI，以进行互相连接或与系统内核连接。如需了解关于 DSI 操作的更多详细信息，请查看技术参考手册中的“UDB 阵列和数字系统互联”一节。

模拟布线概述

通过由开关和多路复用器构成的一系列模拟布线总线，GPIO 引脚被连接到模拟资源，或互相连接。两个主要的模拟布线总线分别为模拟全局（AG）总线和模拟复用（AMUX）总线。AG 总线被分为四个象限（AGL0-4、AGL4-7、AGR0-4 和 AGR4-7），而 AMUX 总线被分为两个部分（AMUXL 和 AMUXR）。图 3 显示的是 TRM 中模拟布线图的一部分。

图 3. 左上角模拟布线象限



每个 AGx 均能同每个象限相关部分中的两个引脚相连，而每个 AMUX 能与自己所在芯片范围内的所有引脚相连。模拟总线也连接了各个模拟资源的输入和/或输出端，例如比较器、DAC 和 ADC。此外，可以通过开关将左右总线相互连接起来。

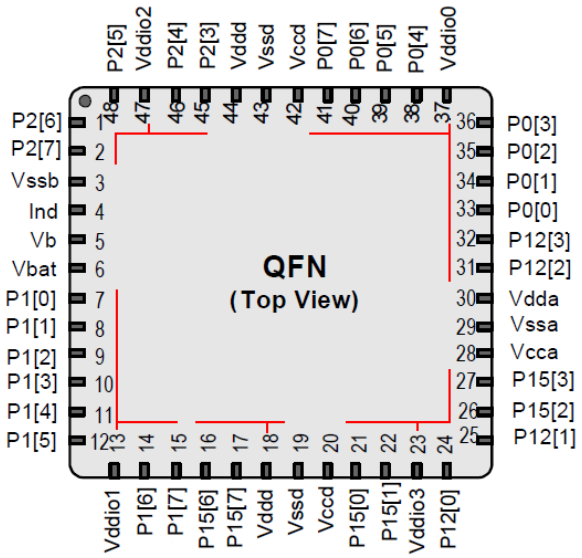
技术参考手册中的“模拟布线”章节进一步介绍了 PSoC 3 和 PSoC 5LP 器件的模拟布线系统。应用笔记 AN58304 和 AN58827 详细讨论了模拟布线和引脚选择内容。

GPIO 功率结构和限制

一般来说，GPIO 引脚的拉电流是 4 mA，灌电流是 8 mA。可以将 GPIO 引脚组合在一起（短路），用以提供比单个引脚的更大的拉电流或灌电流，但是您需要考虑其它功率限制。

PSoC 3 和 PSoC 5LP 器件通过 V_{DDIO} 引脚可以提供四个单独的 I/O 电压阈。在 PSoC 3 和 PSoC 5LP 数据手册的引脚分布图中，用实线标出了为特定引脚组供电的 V_{DDIO} 引脚。图 4 显示的是 48 引脚 PSoC 3 器件，其中 V_{DDIO} 象限指示器以红色高亮显示。

图 4. 红色高亮显示的 V_{DDIO} 象限示例



一般情况下，V_{DDIO} 引脚连接的电源电压范围与 V_{DD} 一样。不需要特别注意单个 V_{DDIO} 象限的拉电流和灌电流，但是这些数值仍受限。表 1 显示的是每个 PSoC 系列和封装类型的限制。

表 1. V_{DDIO} 象限的电流限制

系列	封装	拉电流	灌电流
PSoC 3 PSoC 5LP	100 引脚 68 引脚	对于 V_{DDIO}： 为 100 mA	对于 V_{DDIO}： 为 100 mA
	48 引脚	对于 V_{DDIO0}+V_{DDIO2}： 为 100 mA 对于 V_{DDIO1}+V_{DDIO3}： 为 100 mA	对于 V_{DDIO0}+V_{DDIO2}： 为 100 mA 对于 V_{DDIO1}+V_{DDIO3}： 为 100 mA
PSoC 5	100 引脚 68 引脚	对于 V_{DDIO}： 为 20 mA	对于 V_{DDIO}： 为 20 mA
	48 引脚	对于 V_{DDIO0}+V_{DDIO2}： 为 20 mA 对于 V_{DDIO1}+V_{DDIO3}： 为 20 mA	对于 V_{DDIO0}+V_{DDIO2}： 为 20 mA 对于 V_{DDIO1}+V_{DDIO3}： 为 20 mA
注意：每个 V_{DDIO} 象限（或 48 引脚封装的每一对 V_{DDIO}），拉电流和灌电流的总和不得超出 100 mA。			

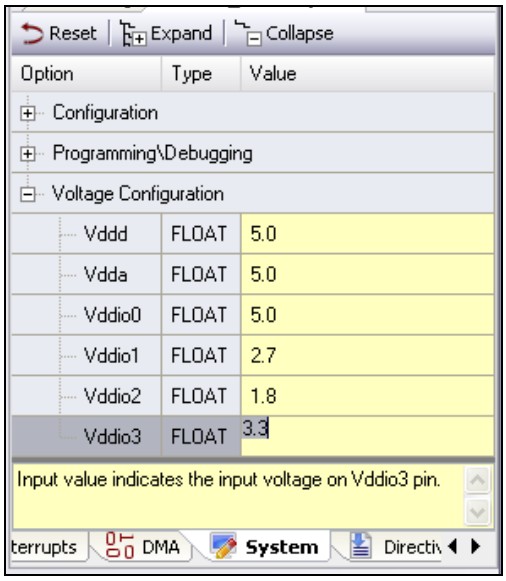
对于 GPIO 引脚提供的拉电流和灌电流预期超出该限值80%的应用，要确保在最坏的情况下 GPIO 引脚的每个象限都不会超出最大值。上述内容表明为了扩大电流，需要使用不同 V_{DDIO} 象限的引脚。

V_{DDA}、V_{DDD} 和 V_{DDIO} 的相对电压

V_{DDA} 必须是 PSoC 3 或 PSoC 5LP 器件上的最高电压。所有其他供电引脚的电压不能大于 V_{DDA}。V_{DDD} 和 V_{DDIO} 引脚电压可能相同，也可能不同。

Design Wide Resources 文件中的 **System** 选项卡包含 Voltage Configuration（电压配置）部分，通过该部分您可以定义每个电压域将运行的工作电压。如果组件或功能依赖于工作电压，则 PSoC Creator 将使用这些字段中输入的数值（如图 5 所示）。

图 5．Design Wide Resources 中的 Voltage Configuration（电压配置）



无论使用哪一种组件或功能，都建议为 PSoC Creator 设置正确的电压。

启动和低功耗特性

启动后，所有 GPIO 引脚都默认进入模拟高阻状态，并在复位前保持该状态。在启动过程中会加载每个引脚的初始操作设置，并会立即生效。对 PSoC 器件进行编程时，您可以使用非易失性锁存器阵列的 PRTxRDM 字段来更改 GPIO 的复位行为。

在所有低功耗模式下，GPIO 引脚都会保持其状态，直到器件被复位或被唤醒为止。端口中断逻辑在所有低功耗模式下仍起作用，所以可将引脚作为唤醒源。

注意：基于 UDB 的组件（如控制寄存器）通常在睡眠或休眠模式下无效。当 PSoC 器件进入或退出这些模式时，它们可能产生干扰。干扰会使 GPIO 进入意外状态。为了避免发生这种情况，应在 PSoC 器件进入低功耗模式之前，将这些引脚明确设置为高或低逻辑状态。

DMA 访问 GPIO 引脚

PSoC 器件带有与不同内部外设（包括 I/O 接口）相连的 DMA 控制器。由于 GPIO 寄存器是内存寻址，可以使用 DMA 传输来配置 GPIO 引脚并将数据写入到数字输出路径，而不需要 CPU 干涉。

DMA 配置和数据传输过于复杂，本应用笔记中难以对其进行详细说明。我们还提供了其他几本应用笔记和示例代码用作这些内容的参考，包括 AN52705 — PSoC 3 和 PSoC 5LP — DMA 入门指南。

端口中断控制单元

PSoC 3 和 PSoC 5LP 有管理 I/O 中断的端口中断控制单元（PICU）。每个 GPIO 引脚都能在上升沿和/或下降沿上产生中断。通过将 cy_isr 组件与引脚组件的中断端连接起来，可以实现电平触发中断。

当 GPIO 中断被触发时，GPIO 状态寄存器中的相应位被设置为‘1’。在读取该寄存器或复位芯片前，该位保持为‘1’。PSoC Creator 提供的 API 会管理 GPIO 中断的配置和报告。

每个端口内的独立 GPIO 中断信号被“OR”运算，并且单独的 PICU 请求被发送到中断控制器内。端口中断请求以菊链式连接以产生单一的唤醒信号，该信号被发送给 PSoC 电源管理器。在所有低功耗模式下，PICU 仍保持激活状态，但唤醒后仍须对单独的 GPIO 中断进行管理。

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。PSoC Creator 结合了文字和图形编辑界面，因此设计人员可以设置自己的硬件配置，同时编程固件。

PSoC Creator API

赛普拉斯提供了一组 API，用于动态控制 GPIO。引脚组件的 API 在引脚和组件范围基础上提供访问功能。

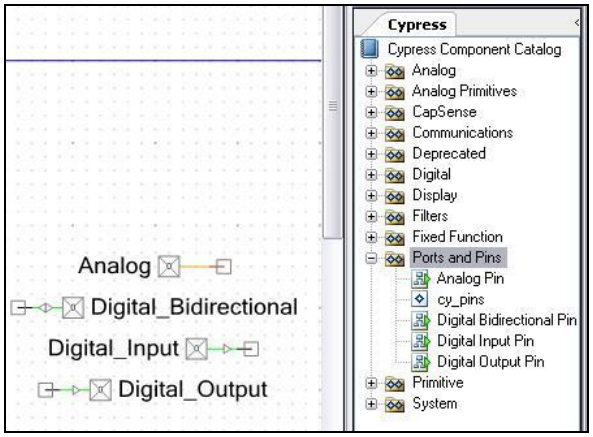
cy_boot 组件也提供了用于访问芯片资源的函数。虽然 cy_boot 的函数不是单独组件库的一部分，但是这些库仍可以使用这些函数。在 PSoC Creator 系统参考指南的“引脚”一节中描述了每个引脚的 API，并作为 *cypins.h* 文件中 cy_boot 的一部分。您可以使用这些 API 控制每个物理引脚的配置寄存器。

要想了解 GPIO API 的总结和简单的代码示例，请参考附录 A：GPIO API 和寄存器参考。

引脚组件的符号和宏

建议使用 cy_pins 组件将内部 PSoC 资源连接到物理引脚。它使 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类型的预定义 GPIO 配置（宏），分别为：模拟、数字双向、数字输入和数字输出。将其中一个组件宏拖动到原理图内，可在项目中添加一个引脚，如图 6 所示。

图 6. PSoC Creator 中的引脚组件符号类型

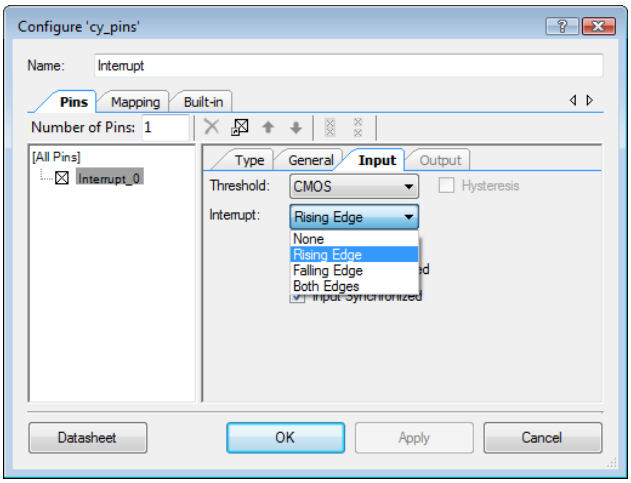


您不受所选宏符号所代表引脚配置的限制。将引脚符号放置在原理图上后，您可以通过本内容中说明的组件自定义程序选项来配置其行为。

引脚组件中断

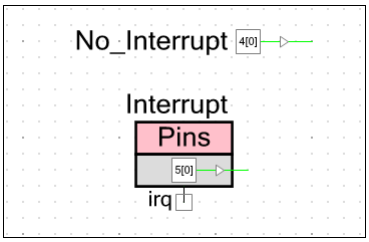
您可以在 PSoC Creator 中通过 cy_pins 配置对话框使能引脚组件中断，如图 7 所示。要想打开引脚组件，请双击它。

图 7. 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 8 所示。触发引脚中断时，引脚组件的 IRQ 信号将被切换。要使能某个引脚中断，您不必将 irq 终端连接至 ISR 组件。

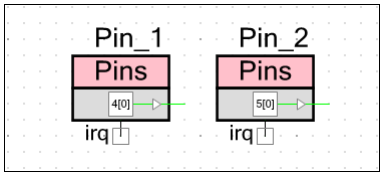
图 8. 使能中断时，引脚组件的符号会发生改变



如果使能了中断，则每个物理 GPIO 端口上只能使用一个引脚组件。该限制是由于对端口内的所有引脚中断进行了“OR”运算，因此只有一个 IRQ 信号能被显示在原理图上。

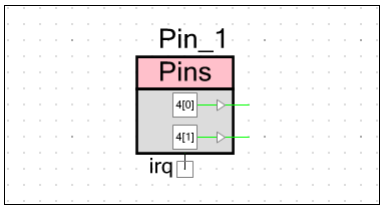
例如，考虑到中断使能时的两个引脚组件，如图 9 所示。这些组件不能被映射到同一个物理端口的引脚，因为 Psoc Creator 原理图中有两个单独的 IRQ 信号，实际上一个端口只能产生一个物理 PICU 中断。

图 9. 带有中断的两个引脚组件



如果试图将这两个组件分配给同一个端口，则 PSoc Creator 将生成一个错误。但能够将多个引脚分配给同一个组件，如图 10 所示。这样可以确保在原理图上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类型。唯一的限制是这些引脚必须是连续的。

图 10. 不同中断边沿类型的引脚选择

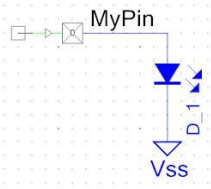


无论组件的设置如何，都可以通过 PICUx_INTTYPEy 寄存器使能或更改任何 GPIO 引脚上的中断。有关这些寄存器的详细信息，请参考文档附录。

外部终端

Cy_pins 配置对话框中提供一个选项用于显示外部终端。这样，您可以将片外组件添加到原理图中并显示它与引脚的连接情况。图 11 显示的是一个用引脚组件驱动片外 LED 的示例。

图 11. 片外组件连接示例

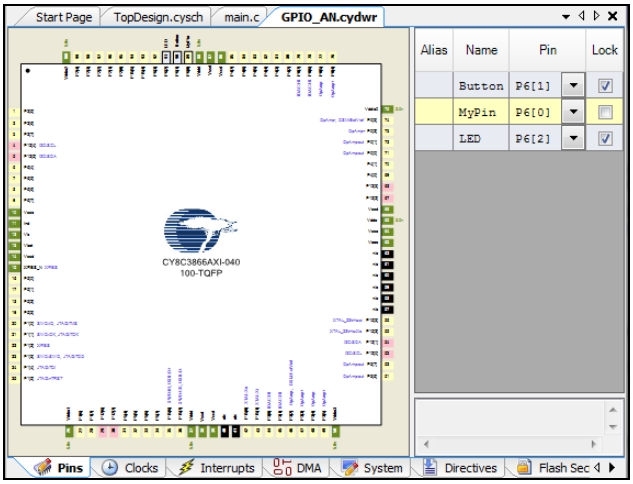


引脚的手动分配

通过 Design-wide Resources 界面（*cydwr*）的 **Pins** 选项卡，将引脚组件分配给物理引脚。如果用户没有选择任何引脚，则 PSoC Creator 将自动指定引脚，但它指定的引脚位置可能在 PCB 上布线更困难。另外，有些 GPIO 引脚直接与模拟或数字资源连接在一起。

图 12 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，黄色高亮显示的引脚是自动分配的。选择 **Lock**（锁定）选项可防止 PSoC Creator 重新分配引脚。

图 12. cydwr 窗口中的引脚分配



通过 PSoC Creator 可以根据需要轻松重新分配引脚，但设计人员在设计电路板前应考虑引脚选择情况。AN58304 和 AN58827 技术参考手册中的“模拟互联”图对于确定选择最佳模拟引脚非常有用。

GPIO 示例、提示和技巧

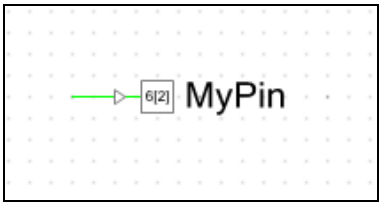
本节提供的实际示例介绍了如何使用 GPIO 引脚。这些示例都是针对 PSoC 3 器件的，但是 PSoC 5LP 可以采用同样的技术。同时介绍了基本的示例和更高级的技术。

GPIO “Hello World”项目

GPIO 的最简单用法是将引脚输出状态设置为高电平或低电平。本示例演示了如何通过引脚组件 API 设置输出状态。

1. 将一个数字输出引脚组件（被配置为强驱动模式）放置在项目原理图中，如图 13 所示。
2. 将组件命名为“MyPin”，并将其分配到 P6[2]。

图 13. Hello World 示例原理图



3. 在 *main.c* 中，使用组件 API 来切换输出，如下所示：

```
for(;;)
{
   /* Set MyPin output state to HIGH */
   MyPin_Write(1);

   /* Delay for 500 ms */
   CyDelay(500);

   /* Set MyPin output state to LOW */
   MyPin_Write(0);

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

4. 编译项目并编程 PSoC 器件。

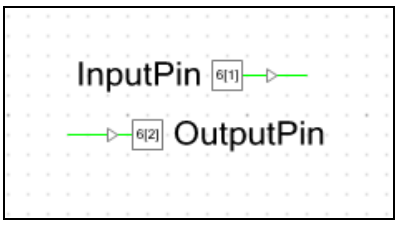
输出会每经过 500 ms 在高/低电平间进行一次切换。

读取输入并写入输出

本示例演示了如何通过组件 API 读取和写入 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，如图 14 所示。

图 14. 输入和输出示例原理图



2. 根据 InputPin，使用组件 API 设置 OutputPin 的状态，如下所示：

```
for(;;)
{
   /* Set OutputPin state to the
      inverse of the InputPin state */
   OutputPin_Write( ~InputPin_Read() );
}
```

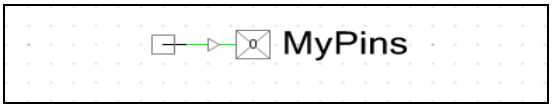
这样会使 OutputPin 的状态始终与 InputPin 的相反。

添加多个 GPIO 引脚，作为逻辑端口

在 PSoC Creator 中，您最多可以将 64 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。

1. 放置单个引脚符号，如图 15 所示。

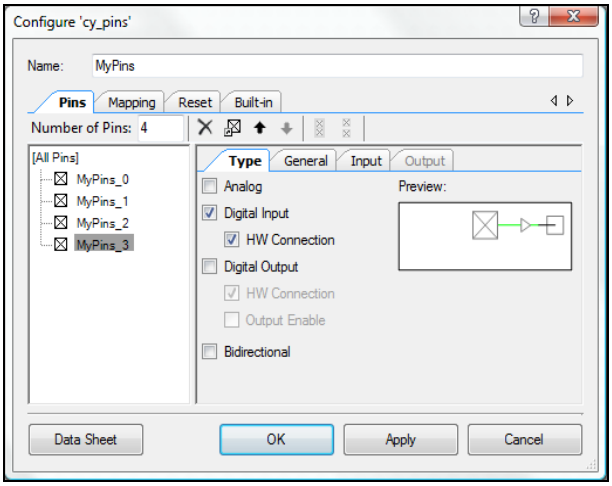
图 15. 在原理图中放置的单引脚符号



2. 双击引脚符号，打开引脚自定义程序窗口。
3. 在配置窗口的 **Number of Pins**（引脚数量）字段中输入引脚数量。

   这些引脚将出现在字段下的列表中。在列表中选择单个引脚，对其进行单独设置。如果选择**[All Pins]**，设置内容会影响该端口上的所有引脚。
4. 对于本示例，将三个引脚设置为数字输出引脚。将最后一个引脚设置为数字输入引脚，如图 16 所示。

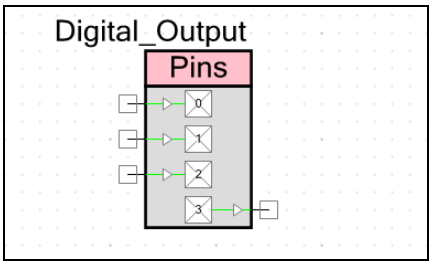
图 16. 四个引脚中的一个被配置为数字输入



5. 单击 **OK**，应用更改内容。

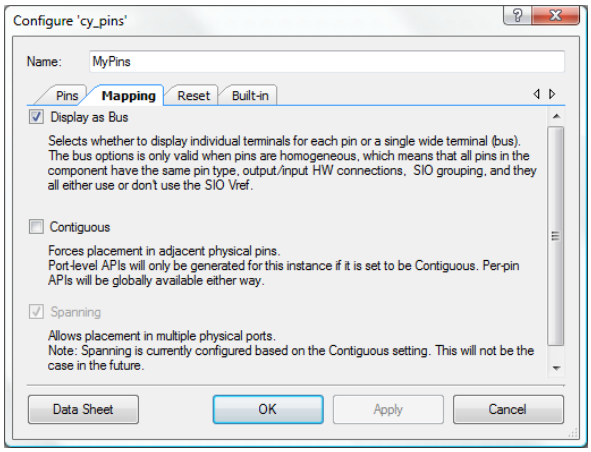
   定义引脚数量及其类型后，原理图的符号同图 17 所示的相似。

图 17. 端口配置窗口中的引脚组件



6. （可选）在引脚配置窗口的 **Mapping** 选项卡中选择 **Display as Bus**（显示为总线）项，使该端口显示为总线符号，如图 18 和图 19 所示。

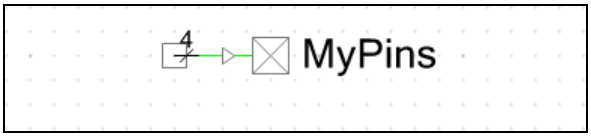
图 18. ‘Display as Bus’选项



该功能不会影响端口的性能。

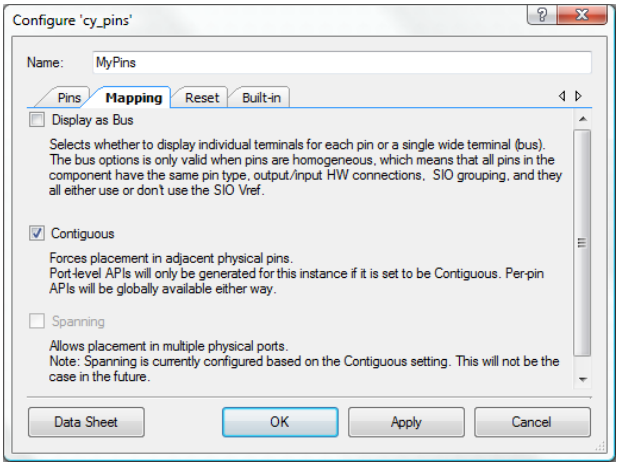
注意： 只有所有引脚类型一样时，才能显示为总线。

图 19. 四个引脚显示为一个端口总线符号



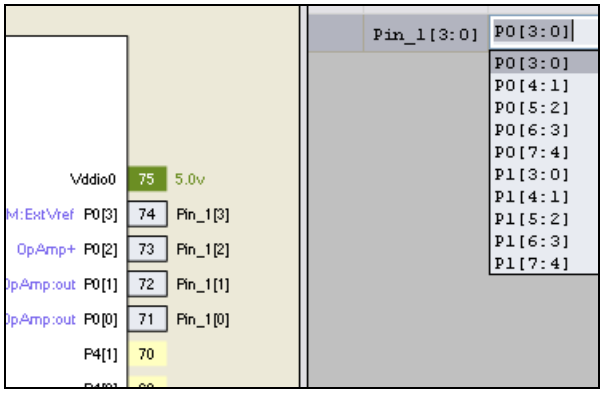
7. （可选）在 **Mapping** 选项卡中选择 **Contiguous** 项，强制将引脚放置在连续的位置上，如图 20 所示。

图 20. 连续引脚放置选项



当选择 **Contiguous** 时，PSoC Creator 会修改可用的引脚分布选项，用以匹配端口配置，如图 21 所示。

图 21. 连续端口引脚的放置



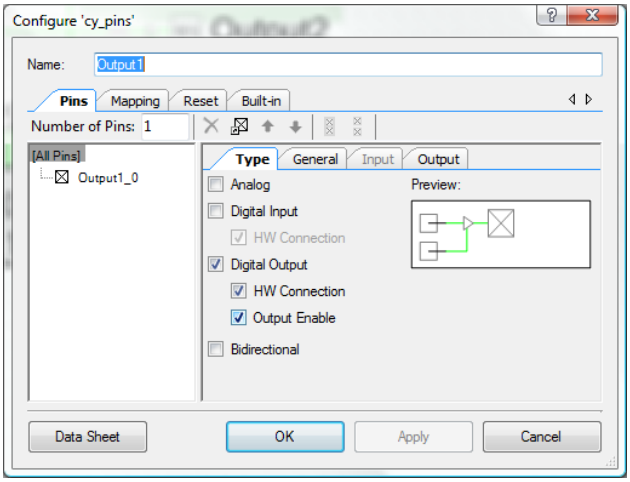
在引脚配置窗口和引脚组件数据手册中详细介绍了这些功能。

配置 GPIO 输出使能逻辑

本示例演示了配置和使用 GPIO 引脚的输出使能逻辑的情况。

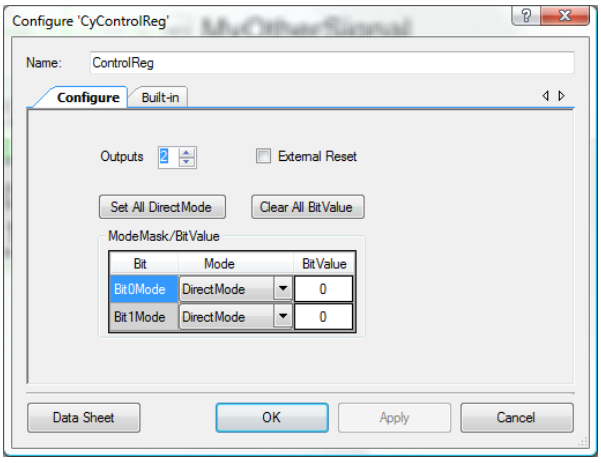
1. 向项目原理图中放置两个数字输出引脚。
2. 打开每个引脚的配置对话框，勾选 **Output Enable**（输出使能）选框，如图 22 所示。

图 22. 输出使能选择



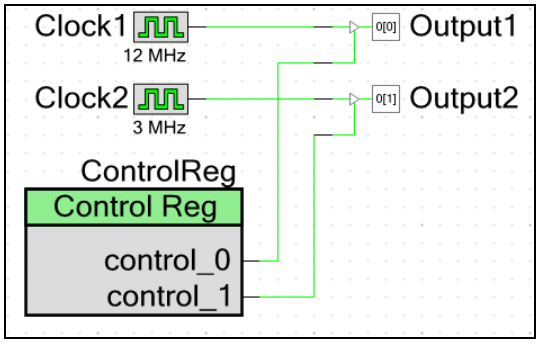
3. 在原理图中放置控制寄存器。
4. 为两个输出配置控制寄存器，如图 23 所示。

图 23. 配置带有两个输出的控制寄存器



5. 添加两个时钟组件，随意对其进行配置。
6. 将时钟连接到引脚，如图 24 所示。

图 24. 控制寄存器驱动引脚的输出使能



7. 将下面的代码添加到 *main.c* 文件中：

```
for(;;)
{
    for( i=0; i<=3; i++ )
    {
        ControlReg_Write(i);
        CyDelay(500);
    }
}
```

8. 编译并编程 PSoC 3 或 PSoC 5LP 器件。

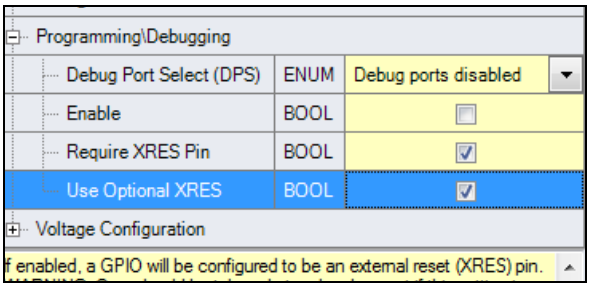
结果为由 ControlReg 状态门控的两个引脚的输出。

使能可配置 XRES 功能

本示例演示了如何使能可配置 XRES 功能。您可以将引脚 P1[2]配置为可选的 XRES 引脚，以支持对小型封装进行外部复位。该功能也适用于更大的封装类型。

1. 在 Design Wide Resources 文件中，打开 **System** 选项卡，如图 25 所示。
2. 选择 **Use Optional XRES**（使用可选 XRES）选项，使能可选的 XRES 逻辑。如果选中该选框，则 P1[2]不再作为 GPIO 引脚使用，并且被配置为带有内部上拉电阻的低电平输入。

图 25. 可选的 XRES 引脚使能



3. 对 PSoC 器件进行编程，将设置内容写入到非易失性阵列内。在下次上电后它会生效。

4. 取消 **Use Optional XRES** （使用可选 XRES）选项并重新编程 PSoC 器件，以恢复正常的 GPIO 功能。

请注意，在出厂时所有 PSoC 3 和 PSoC 5LP 器件的可选 XRES 功能被默认禁用。使用可配置的 XRES 引脚不会改变专用 XRES 引脚的功能。

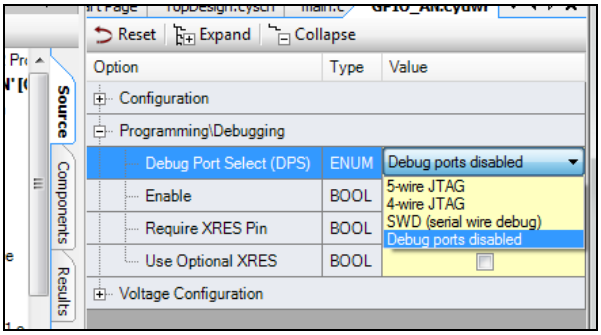
禁用 GPIO 引脚的调试逻辑

本示例演示了如何禁用与端口 1 引脚相关的调试逻辑。使能调试端口功能时，如果 PSoC 器件在启动时检测到这些引脚上的活动，它将进入调试模式。

1. 打开 Design-wide Resources 文件，然后单击 **System** 选项卡。

2. 从下拉菜单中选择 **Debug ports disabled**（调试端口禁用），如图 26 所示。

图 26. 调试端口禁用



3. 编译并编程 PSoC 3 或 PSoC 5LP 器件。

请注意，如果需要进行调试，必须再次手动使能调试端口。禁用调试接口不会影响对该器件的编程能力。

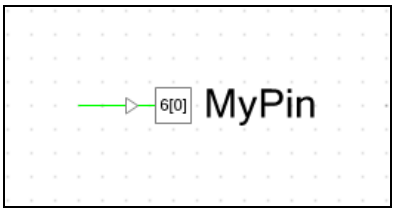
通过数据寄存器能更快切换 GPIO

本示例演示了如何使用端口数据寄存器和掩码快速切换引脚。虽然组件 API 是控制 GPIO 引脚的最简单方法，但更新引脚时所需的处理器周期数量会影响切换速度。寄存器的定义和掩码位于每个组件的*<pin_name>.h* 文件中，使用它们可以加快更新各个引脚。

1. 在原理图中放置数字输出引脚。为便于使用，将其命名为“MyPin”。

2. 配置组件为无硬件连接，将其分配到某个物理引脚（本示例使用 P6[0]），如图 27 所示。

图 27. 原理图中放置的引脚

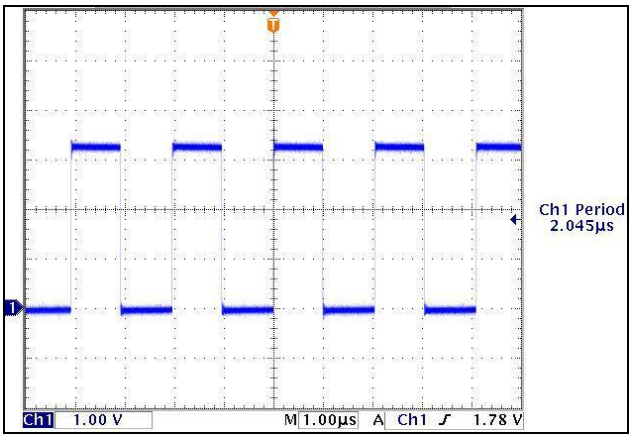


3. 将下面的代码添加到 *main.c* 文件中：

```
for(;;)
{
    // These are API functions
    MyPin_Write(1); //set MyPin output
    MyPin_Write(0); //clear MyPin output
}
```

4. 使用 API 观察 P6[0]的输出，如图 28 所示。

图 28. 使用 API 开关法切换引脚

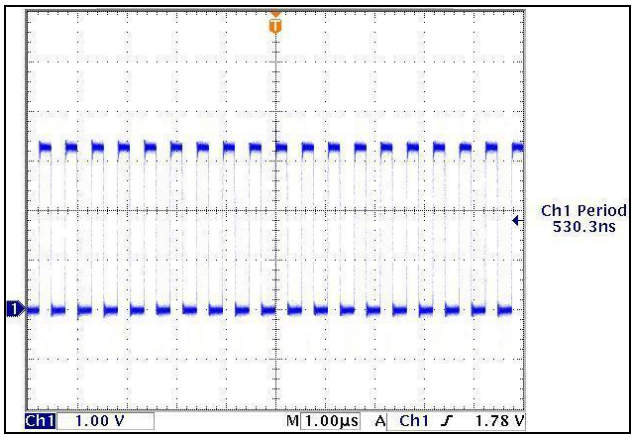


5. 使用下面的代码替换 *main.c* 中原来的代码：

```
for(;;)
{
    MyPin_DR |= MyPin_MASK; //Set MyPin
    MyPin_DR &= ~MyPin_MASK; //Clear
}
```

6. 使用快速切换法观察 P6[0]的输出，如图 29 所示。

图 29. 使用快速切换法切换引脚



使用快速切换法进行引脚切换的速度几乎是 API 函数法的四倍。使用这段代码也非常方便。如果引脚分配在开发过程中发生了变化，您不需要对特定的物理引脚寄存器进行写操作。

使用 8051 特殊功能寄存器

PSoC 3 中的 8051 有一组特殊功能寄存器（SFR），通过它您能够更快地访问受限的一组 PSoC 寄存器。您可以使用其中两个寄存器快速切换 GPIO 引脚。

1. 在项目原理图中放置一个数字输出引脚组件，并将其分配到物理引脚，如前一示例中所介绍的。本示例也使用 P6[0]。
2. 将下面的代码添加到 *main.c* 文件中：

```
/* Enable SFR access for P6[0]. */
/* Only done once in the beginning. */
SFRPRT6SEL |= 0x01;
/* Toggle GPIO pin. */
for(;;)
{
      /* Switch on P6[0] */
      SFRPRT6DR |= 0x01;
      /* Switch off P6[0] */
      SFRPRT6DR &= ~0x01;
}
```

3. 另外，使用以下方法：

```
for(;;)
{
      /* Toggle P6[0] */
      SFRPRT6DR ^= 0x01;
}
```

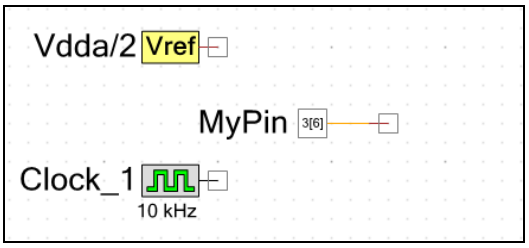
任何一种方法都能够实现快速切换引脚。有关 SFR 的详细信息，请参考 PSoC 3 架构 TRM。

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。假定 GPIO 引脚需要在短时间内输出 10 kHz 的时钟信号，快速切换到参考电压，然后切换回 10 kHz 信号。

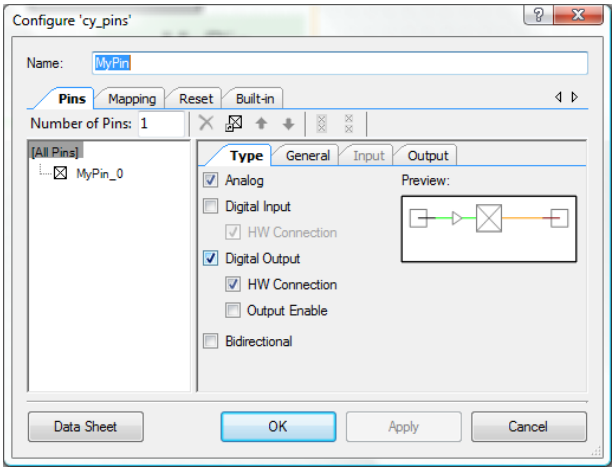
1. 在原理图中放置一个模拟引脚（V_{REF}）和一个时钟。
2. 将引脚组件分配到某个物理引脚（本示例使用 P3[6]），如图 30 所示。

图 30. 原理图中放置的基本组件



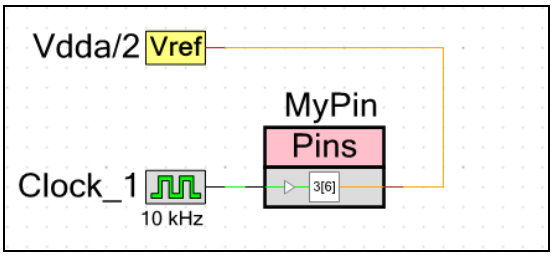
3. 同时将引脚配置为模拟和数字输出，如图 31 所示。

图 31. MyPin 被配置为模拟和数字功能



4. 将时钟连接到数字终端，并将 V_{REF} 连接到模拟终端，如图 32 所示。

图 32. PSoC Creator 的模拟和数字切换原理图



5. 编译项目，创建用于确定模拟布线所需的 API，PSoC Creator 使用了该模拟布线。

6. 打开 *cyfitter_cfg.c* 文件，寻找 CYREG_PRT3_AG（模拟全局使能）或 CYREG_PRT3_AMUX（模拟多路复用器总线使能）。在这种情况下，布线工具选择使用端口 0 的 AG 总线，如下所示。

```
CY_SET_REG8(CYREG_PRT3_AG, 0x40);
```

注意，每次重编译项目时都可能更改模拟布线。如果对该项目进行了任何更改，必须检查布线情况。

7. 将下面的代码添加到 *main.c* 文件中：

```
for(;;)
{
    /* Set pin to Analog */
    // Set P3[6] to Analog HI-Z
    CyPins_SetPinDriveMode(CYREG_PRT3_PC
    6, PIN_DM_ALG_HIZ);

    // Make AG connection for P3[6]
    CY_SET_REG8(CYREG_PRT3_AG,
    CY_GET_REG8(CYREG_PRT3_AG) | 0x40);

    // Wait for 100 ms while driving
    signal
    CyDelay(100);

    /* Set pin to digital */
    // Break AG connection for P3[6]
    CY_SET_REG8(CYREG_PRT3_AG,
    CY_GET_REG8(CYREG_PRT3_AG) & 0xBF);

    //Set P3[6] to Strong Drive mode
    CyPins_SetPinDriveMode(CYREG_PRT3_PC
    6, PIN_DM_STRONG);

    // Wait for 100 ms while driving
    signal
    CyDelay(100);
}
```

8. 编译并编程 PSoC 3 或 PSoC 5LP 器件。

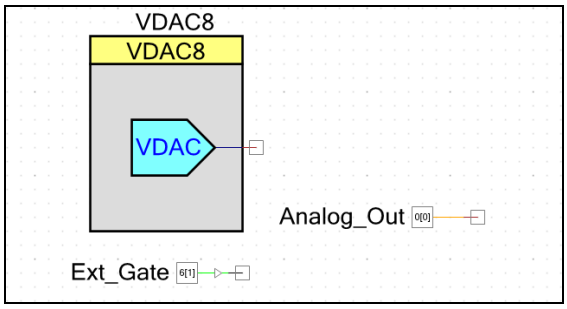
结果是每 100 ms 在时钟信号和参考电压之间进行交替的输出。

用硬件控制模拟开关

本示例演示了在没有 CPU 干预下如何使用外部信号控制模拟引脚输出。

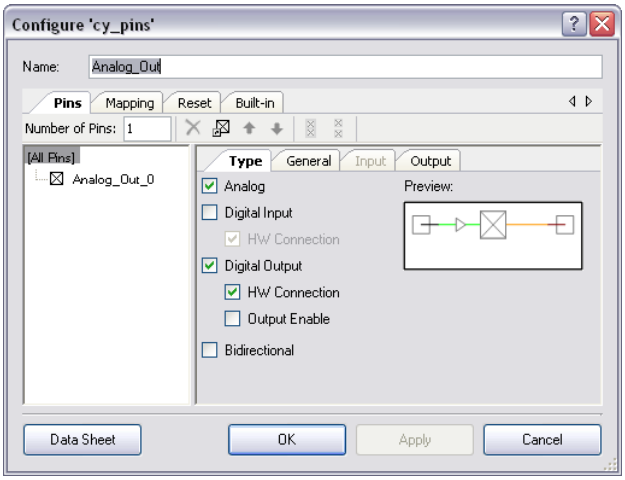
1. 在项目原理图中放置一个数字输入引脚（在本示例中为 Ext_Gate）、模拟引脚（Analog_Out）和模拟源（VDAC8），如图 33 所示。

图 33. 硬件控制门的组件



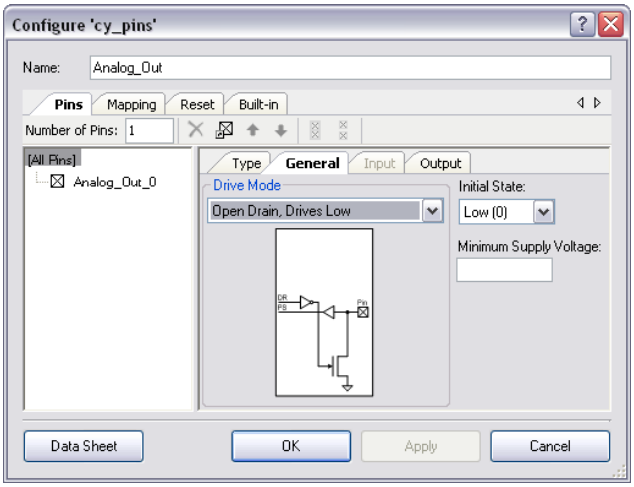
2. 同时将 Analog_Out 引脚配置为模拟和数字输出，如图 34 所示。

图 34. Analog_Out 引脚配置



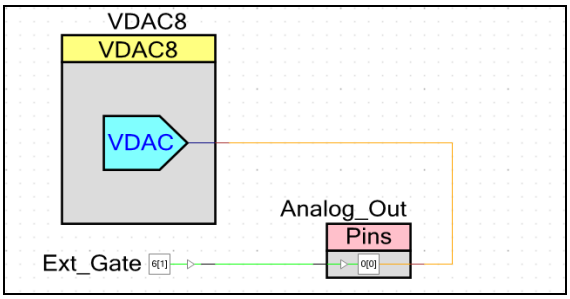
3. 将 Analog_Out 引脚的驱动模式配置为“开漏低电平驱动模式”，如图 35 所示。

图 35. Analog_Out 引脚的驱动模式



4. 连接组件，如图 36 所示。

图 36. 带有硬件门控的模拟引脚



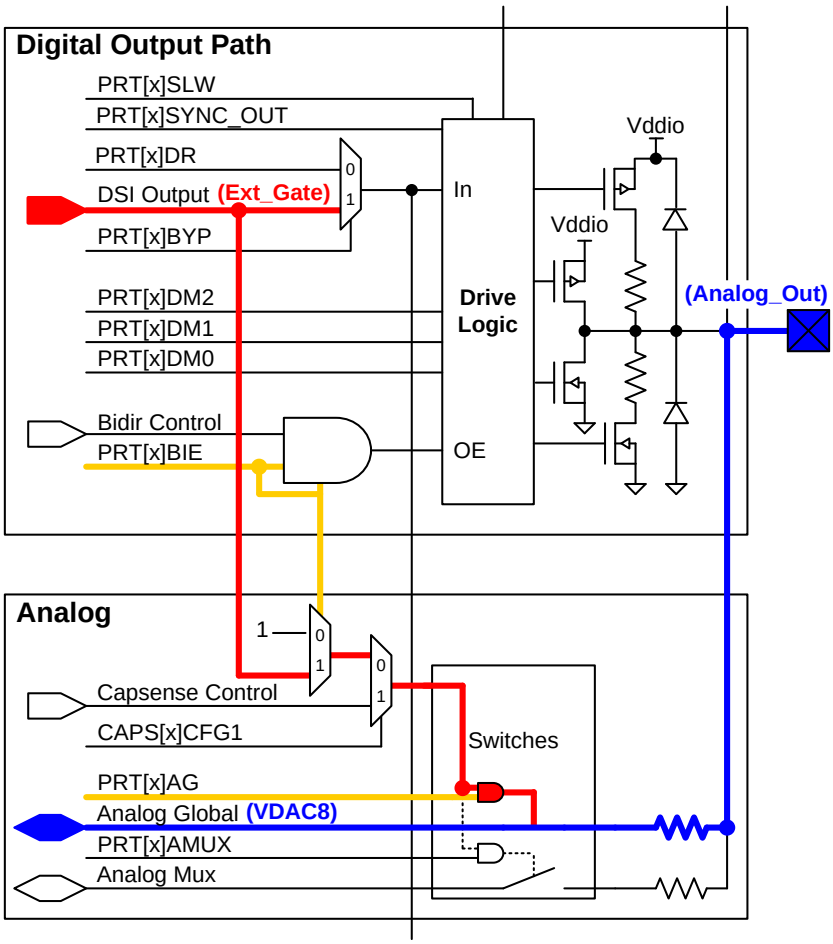
5. 将下面的代码添加到 *main.c* 文件中：在本示例中，设置了端口 0 配置寄存器中引脚 0 上的双向位：

```
// Set P0[0] to bidirectional mode
CY_SET_REG8(CYDEV_IO_PRT_PRT0_BIE,
0x01);
```

6. 将 Analog_Out 引脚分配给 P0[0]，以匹配前面的代码。
7. 编译并编程 PSoC 3 或 PSoC 5LP 器件。

图 37 取自 PSoC 3 和 PSoC 5LP 数据手册中的 GPIO 详细框图，它显示了如何使用 GPIO 控制逻辑来实现该技术。

图 37. 高亮显示的 GPIO 框图取自 PSoC 3 和 PSoC 5LP 数据手册



Ext_Gate 信号通过 DSI 连接到 Analog_Out 引脚的数字部分。来自 DSI 的信号（红色）被连接到模拟开关，因为已经设置了端口双向位和模拟全局选择位（黄色）。根据 Ext_Gate 信号的逻辑状态，打开或关闭 VDAC 输出（蓝色）。

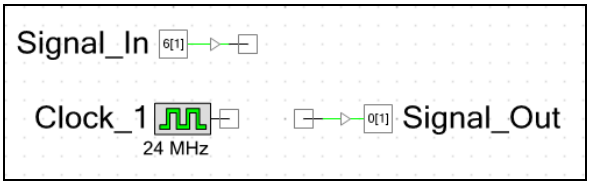
如需了解有关 PSoC 3 和 PSoC 5LP 器件的模拟切换功能，请参阅技术参考手册中的“模拟布线”章节。另一个优势资源是应用笔记 AN58827 — PSoC 3 和 PSoC 5LP 内部模拟布线的注意事项。

将 DSI 作为时钟源使用

本示例演示了如何将经过 DSI 的数字信号作为时钟源使用。通过任意一个 DSI 信号，最多可以创建 8 个数字时钟和 4 个模拟时钟。此外，PSoC 器件可以将任意的数字信号作为 PLL 的输入源使用。

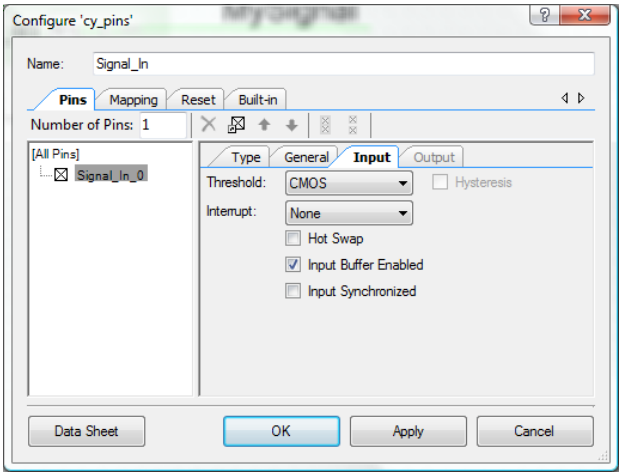
1. 在原理图上放置一个数字输入引脚、一个数字输出引脚和一个时钟，如图 38 所示。

图 38. DSI 时钟的基本组件示例



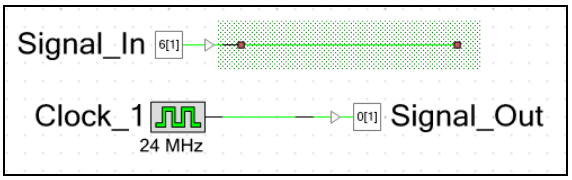
2. 在引脚配置窗口中，打开 Signal_In 的配置对话框，然后取消选中 **Input Synchronized**（输入同步）选项，如图 39 所示。该操作对防止信号与自身同步非常必要。

图 39. 输入同步的设置被禁用



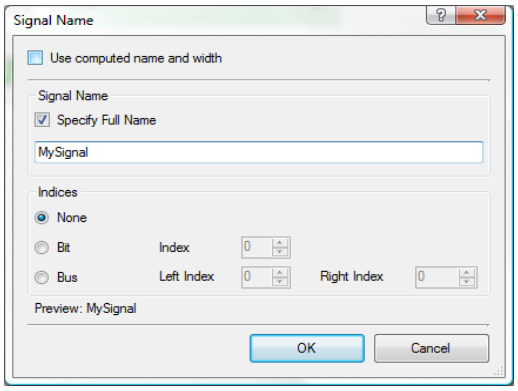
3. 使用连线将时钟连接到 Signal_Out。
4. 使用连线工具为 DSI 源添加一个信号，并且该信号仅连接到 Signal_In。从 Signal_In 终端开始拉线，如图 40 所示。

图 40. DSI 时钟源信号



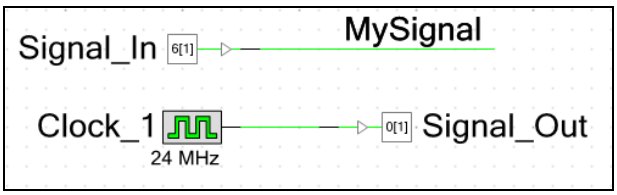
5. 右键单击该连线，从出现的弹出菜单中选择 **Edit Name And Width**（编辑名称和宽度）项。
6. 取消选择 **Use computed name and width**（使用计算的名称和宽度）选项，在 **Signal Name**（信号名称）字段中输入唯一的名称（在本示例中为“MySignal”），如图 41 所示。

图 41. 信号名称配置窗口



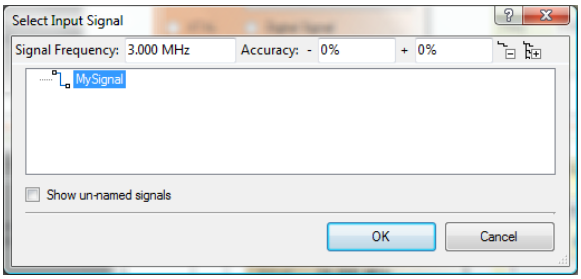
设置完这些配置后，原理图同图 42 所示的情况相似。

图 42. DSI 时钟的修改组件示例



7. 打开项目中的 Design-wide Resources 文件（*<Project name>.cydwr*），然后选择 **Clocks** 选项卡。
8. 双击任意一个系统时钟，打开 **Configure Systems Clocks**（配置系统时钟）窗口。
9. 选中 **Digital Signal**（数字信号）选项，单击“**...**”按键以打开 **Select Input Signal**（选择输入信号）窗口。
10. 选择“MySignal”，输入信号频率（在本示例中为 3 MHz）和精确度，如图 43 所示。

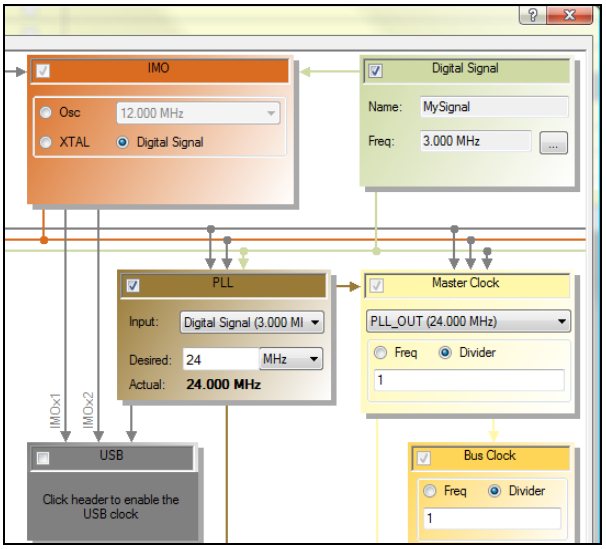
图 43. 数字信号配置窗口



注意：PSoC Creator 使用该信息完成配置系统时钟所需的计算。应确保信号信息的准确性。

11. 将 **IMO** 和 **PLL** 的输入源设置为“Digital Signal”（数字信号），如图 44 所示。

图 44. DSI 信号作为 IMO 和 PLL 的源



PLL 使用 3 MHz 的输入生成 24 MHz 的输出，该输出被连接到主设备和总线时钟，用以生成 Signal_Out 时钟。

如需了解有关 DSI 系统时钟的详细信息，请参阅数据手册、技术参考手册和应用笔记 AN60631 — PSoC 3 和 PSoC 5LP 时钟资源。

通过固件更改 PICU 设置

通过对 PICUx_INTTYPEy 寄存器的位[1:0]进行写操作完成 PICU 的动态配置，“x”为相应端口的编号，“y”为相应引脚的编号（请参见表 2）。您可以随时更改这些配置项，以使能或禁用引脚中断。

表 2. PICU 中断类型和位设置

位 1:0	名称	说明
00	禁用	中断被禁用
01	上升沿	在上升沿上触发
10	下降沿	在下降沿上触发
11	更改模式	在任意边沿上触发

在本示例中，引脚 P6[0]被配置为上升沿中断， P6[1]被配置为下降沿中断。

1. 向项目原理图中放置两个数字输入引脚。
2. 将引脚指定给 P6[0]和 P6[1]。
3. 将 P6[0]配置为电阻下拉引脚，或添加外部下拉电阻。
4. 将 P6[1]配置为电阻上拉引脚，或添加外部上拉电阻。
5. 将下面的代码添加到 *main.c* 文件中：

```
//Set P6[0] to PICU rising-edge trigger
CY_SET_REG8(CYREG_PICU6_INTTYPE0,
0x01);

//Set   P6[1]   to   PICU   falling-edge
trigger
CY_SET_REG8(CYREG_PICU6_INTTYPE1,
0x02);

//Sleep and wait for PICU interrupt
//Sleep again if not P6[1] PICU wakeup
do
{
    //Save clocks and enter sleep
    CyPmSaveClocks();
    CyPmSleep(PM_SLEEP_TIME_NONE,
PM_SLEEP_SRC_PICU);
    CyPmRestoreClocks();

    //Stay awake for two seconds
    CyDelay(2000);
}
while
(!(CY_GET_REG8(CYREG_PICU6_INTSTAT)     &
0x02));

//Disable P6[1] PICU trigger
CY_SET_REG8(CYREG_PICU6_INTTYPE1,
0x00);
```

6. 编译并编程 PSoC 3 或 PSoC 5LP 器件。

PSoC 器件会被任何 PICU 中断唤醒，但它将重新返回到睡眠状态，除非 P6[1]是触发器。唤醒后，您不再需要禁用中断。正常操作期间可以将他们作为中断源使用。

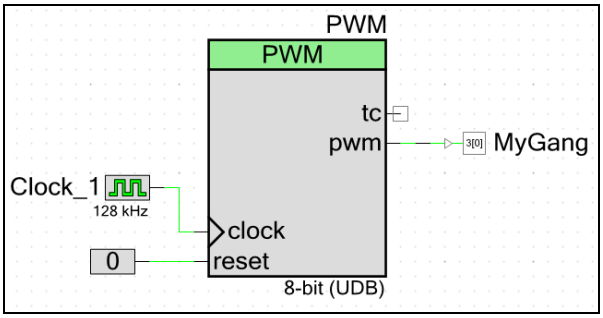
PSoC 3 或 PSoC 5LP 技术参考手册中包含更多有关 PICU 的信息，包括框图和功能说明描述。另一个很好的参考资源是应用笔记 AN54460 — PSoC 3 和 PSoC 5LP 中断。

组合引脚以获得更大的拉电流/灌电流

为了提高电路的总拉电流/灌电流，可以组合使用 GPIO 引脚（短接在一起）。V_{DDIO} 象限的限制仍然适用。本示例演示了如何使用四个 GPIO 引脚驱动 PWM 信号。

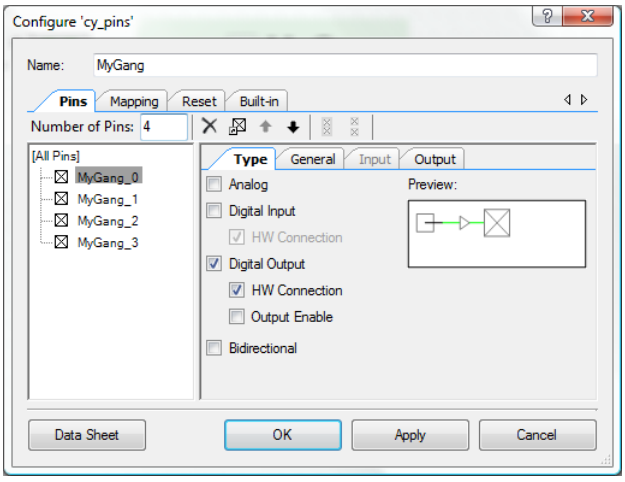
1. 放置并配置 PWM 和时钟组件。
2. 放置单个数字输出引脚组件，并将其连接到 PWM 输出终端，如图 45 所示。

图 45. 在原理图中放置的单个引脚



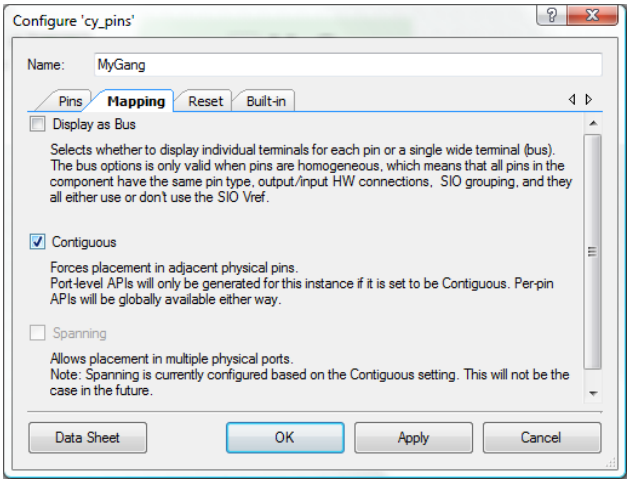
3. 打开引脚配置对话框，设置所需引脚数量，如图 46 所示。本示例使用了四个 GPIO 引脚。

图 46. 在组件中配置多个引脚



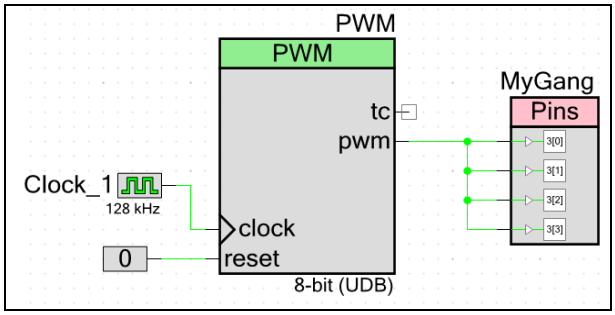
4. 或者，将引脚的映射设置为 **Contiguous**，便于进行 PCB 布线，如图 47 所示。

图 47. 使能连续映射



5. 将引脚组件分配给物理引脚。
6. 将信号源（在本示例中为 PWM）连接至组件的每个引脚终端， 如图 48 所示。

图 48. 组合引脚的符号



7. 编译并编程 PSoC 3 或 PSoC 5LP 器件。

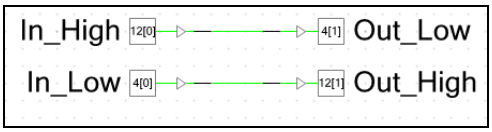
PWM 输出由全部四个 GPIO 驱动。可以在 PCB 板上对这些引脚进行外部短接，并根据需要连接到外部电路。

电平漂移信号

通过为 V_{DDIO} 引脚施加不同的电压，GPIO 引脚可用于信号的电平漂移。唯一的限制是，V_{DDIO} 象限不能超过 V_{DDA} 的电压。本示例演示了在 PSoC 3 或 PSoC 5LP 器件中如何实现简单的 5 V/1.8 V 电平漂移。

1. 在项目原理图中放置两个高阻态的数字输入引脚和两个强驱动数字输出引脚。
2. 将其中一个输入与另一个输出相连接。将剩下的输入与其余输出相连。
3. 为了容易实现，给引脚符号起一个有意义的名字，类似于图 49 所示。

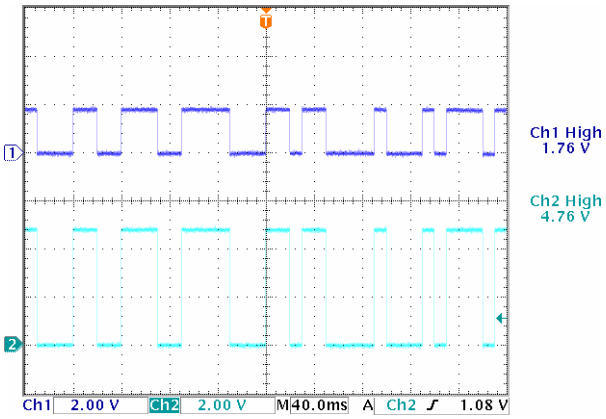
图 49. GPIO 引脚的电平漂移



4. 将 5 V 信号分配给 V_{DDIO} 象限内的引脚，将 1.8 V 信号分配给另一象限中的引脚。在本示例中，选定了 V_{DDIO3}（5.0 V – P12[1:0]）和 V_{DDIO0}（1.8 V – P4[1:0]）。请参阅 V_{DDIO} 分布的器件数据手册。
5. 在开发电路板上，将 V_{DDD}、V_{DDA}、V_{DDIO1}、V_{DDIO2} 和 V_{DDIO3} 连接到一个 5 V 的电压。
6. 将 V_{DDIO0} 与 1.8 V 电压连接。
7. 编译并编程 PSoC 3 或 PSoC 5LP 器件。

在高侧输入端输入任何 5 V 信号，都会在低侧输出 1.8 V 的电压。同样，在低侧输入端输入 1.8 V 信号，那么会在高侧输出 5 V 的电压，如图 50 所示。

图 50. 电平漂移信号



除了使信号电压漂移外，PSoC 器件还可以读取和操作所传输的数据。请注意，V_{DDIO} 象限中的所有 GPIO 引脚的电压均相同。

相关应用笔记

- AN54181 — PSoC 3 入门
- AN77759 — PSoC 5LP 入门
- AN60631 — PSoC 3 和 PSoC 5LP 时钟资源
- AN58304 — PSoC 3 和 PSoC 5LP — 模拟设计中的引脚选择
- AN58827 — PSoC 3 和 PSoC 5LP 内部模拟布线的注意事项
- AN54460 — PSoC 3 和 PSoC 5LP 中断
- AN60580 — PSoC 3/PSoC 5LP 中 SIO 提示和技巧
- AN77900 — PSoC 3 和 PSoC 5LP 低功耗模式和降低功耗技术
- AN52705 — PSoC 3 和 PSoC 5LP — DMA 入门

关于作者

姓名：	Greg Reynolds
背景：	Greg Reynolds 在赛普拉斯公司供职已经超过十年，担任过多个职务。

附录 A：GPIO API 和寄存器参考

组件 API

引脚组件 API 用于访问所有与组件相关联的物理引脚。该组件的实例名称是由用户指定或者由 PSoC Creator 生成的，作为函数名称的前缀。引脚组件数据手册中详细介绍了组件 API。表 3 列出了组件的函数。

表 3. 组件 API 参考

API	说明	示例（使用名为 MyPin 的组件）
<Pin_Name>_Read	为组件中所有引脚返回当前值。	`myVar = MyPin_Read();`
<Pin_Name>_Write	将数值写入到组件引脚。	`MyPin_Write(1);`
<Pin_Name>_ReadDataReg	为组件中所有引脚返回当前值。	`myVar = MyPin_ReadDataReg();`
<Pin_Name>_SetDriveMode	为该组件的每一个引脚设置驱动模式。	`MyPin_SetDriveMode(MyPin_DM_ALG_HIZ);`
<Pin_Name>_ClearInterrupt	清除映射组件各端口上的所有有效中断。返回中断状态寄存器中的数值。	`myVar = MyPin_ClearInterrupt();`

引脚 API

通过全局单引脚 API 宏，可以访问单个物理引脚。物理引脚不需要与引脚组件相关，因为宏可以直接访问引脚配置寄存器。由于与组件配置相冲突，所以使用引脚 API 会在与引脚组件相关的所有物理引脚上引发未定义行为。PSoC Creator 系统参考指南中详细介绍了引脚 API。表 4 列出了引脚函数。

表 4. 引脚 API 参考

API	说明	示例（使用 P1[2]）
CyPins_ReadPin	读取引脚上的当前值。	`myVar = CyPins_ReadPin(CYREG_PRT1_PC2);`
CyPins_SetPin	将引脚的输出值设置为逻辑高电平。	`CyPins_SetPin(CYREG_PRT1_PC2);`
CyPins_ClearPin	将引脚的输出值设置为逻辑低电平。	`CyPins_ClearPin(CYREG_PRT1_PC2);`
CyPins_SetPinDriveMode	设置引脚的驱动模式。	`CyPins_SetPinDriveMode(CYREG_PRT1_PC2, PIN_DM_ALG_HIZ);`
CyPins_ReadPinDriveMode	读取引脚的驱动模式。	`myVar = CyPins_ReadPinDriveMode(CYREG_PRT1_PC2);`
CyPins_FastSlew()	将引脚的斜率设置为快速沿速率。	`CyPins_FastSlew(CYREG_PRT1_PC2);`
CyPins_SlowSlew()	将引脚的斜率设置为慢速沿速率。	`CyPins_SlowSlew(CYREG_PRT1_PC2);`

GPIO 寄存器

下列寄存器用于配置 GPIO。在正常操作期间，可以通过固件访问它们。PSoC 3 和 PSoC 5LP 寄存器的技术参考手册文档中提供了更多的详细信息和寄存器映射图，可以免费从赛普拉斯网站下载这些文档。表 5 列出了 GPIO 寄存器。

表 5. GPIO 寄存器

寄存器	名称	说明
PRTx_PCy[7:0]	端口引脚配置	该寄存器一次访问单个 I/O 端口引脚的多个配置或状态位。
PRTx_DR[7:0]	端口数据输出	该寄存器用来设置相应 GPIO 端口的输出数据状态。
PRTx_PS[7:0]	端口引脚状态	该寄存器保持相应 GPIO 端口的逻辑引脚状态。如果引脚的驱动模式被设置为高阻态模拟，则读取状态始终返回 0。
PRTx_DM2[7:0] PRTx_DM1[7:0] PRTx_DM0[7:0]	端口驱动模式	可使用这些寄存器的组合值来确定 GPIO 端口中每个位的独特驱动模式。
PRTx_SLW[7:0]	端口斜率控制	该寄存器用于为所有强驱动模式的 GPIO 引脚设置一个快速边沿速率或慢速边沿速率。
PRTx_BYP[7:0]	端口旁路使能	该寄存器会选择来自 DSI 还是选择来自端口逻辑数据寄存器的相应 GPIO 输出数据。
PRTx_BIE[7:0]	端口双向使能	该寄存器用于使能通过 DSI 的动态双向控制。
PRTx_INP_DIS[7:0]	输入缓冲区禁用覆盖	该寄存器用于强制关闭输入缓冲区。
PRTx_CTL[0]	端口宽度控制信号	该寄存器用于选择内部缓冲区的激发点。
PRTx_PRT[7:5,3:1]	端口宽度配置	根据端口宽度，该寄存器选择访问几个可用配置寄存器。
PRTx_BIT_MASK[7:0]	别名寄存器访问的位掩码	该寄存器允许或阻止从别名寄存器地址空间访问数据寄存器。
PRTx_AMUX[7:0]	端口模拟全局复用总线使能	该寄存器控制相应 GPIO 端口上的模拟全局多路复用开关。
PRTx_AG[7:0]	端口模拟全局使能	该寄存器控制着相应 GPIO 端口的模拟全局开关。
PICUx_INTTYPE[1:0]	端口中断控制类型	该寄存器会配置相应 GPIO 端口的中断类型。
PICUx_INTSTAT[7:0]	端口中断控制状态	该寄存器显示了相应 GPIO 端口上已发生的中断。
PICUx_SNAP[7:0]	端口中断控制快照	该寄存器显示了最后一次读取 PICUx_INTSTAT 时输入引脚上的状态。
PICUx_DISABLE_COR[0]	禁用状态寄存器“Clear on Read”（读取时清除）功能	该寄存器禁用了 PICUx_INTSTAT 寄存器的“Clear on Read”功能。

非易失性锁存器

PSoC 3 和 PSoC 5LP 具有一个用于复位时配置设备的非易失性锁存器（NVL） 阵列。下列锁存器用于配置 GPIO。在正常操作期间，无法通过固件访问它们。PSoC 3 和 PSoC 5LP 数据手册以及技术参考手册中的“非易失性锁存器”章节提供了更多信息以及 NVL 阵列的寄存器映射状况。表 6 列出了 NVL 锁存器。

表 6. 与 GPIO 相关的 NVL 锁存器

NVL 锁存器	名称	说明
PRTxRDM[1:0]	端口复位驱动模式	用于控制相应 I/O 端口的复位驱动方式。
XRESMEN[0]	可选的 XRES 使能	用于控制将引脚 P1[2]用作 GPIO 还是外部复位。
DPS[1:0]	调试端口选择	控制将各个 Port1 引脚作为调试端口使用。

附录 B：PSoC Creator 设置和寄存器

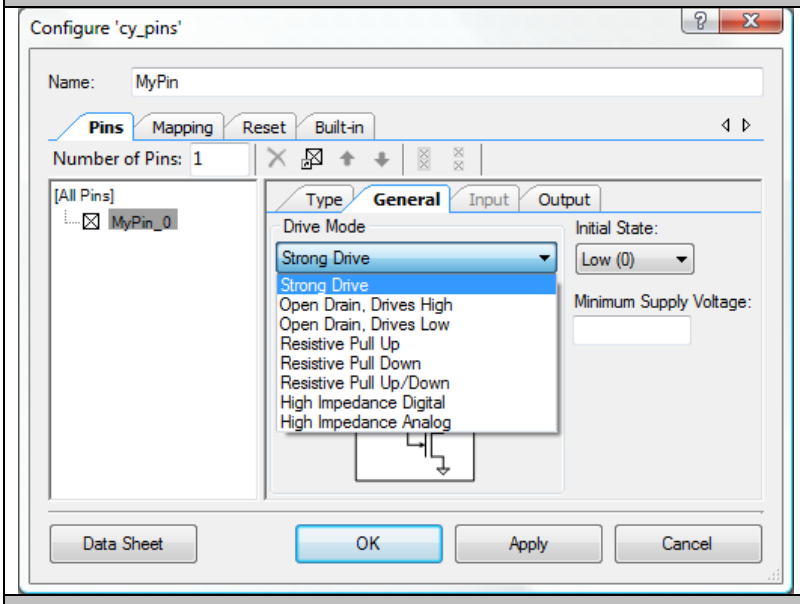
PSoC Creator 设定的 GPIO 设置是 cy_boot 启动代码的一部分，在初始器件配置过程中生效。本附录中的表格显示的是引脚组件配置窗口中的设置与 GPIO 寄存器之间的关系。此外，通过简单的代码示例演示了如何在正常运行期间在固件中执行相同的功能（若适用）。配置窗口看起来不太一样，具体情况取决于您正在使用的 PSoC Creator 版本。

表 7. 驱动模式参数

驱动模式

该参数将引脚配置为可用的 8 种引脚驱动模式中的一种。引脚类型将确定默认的设置。

PSoC Creator 配置窗口



组件 API

<PinName>_SetDriveMode()

引脚 API

CyPins_SetPinDriveMode()
CyPins_ReadPinDriveMode()

相关寄存器

PRTx_PCy[3:1]
PRTx_DMy[7:0]
PRTx_BIE[7:0]

示例代码

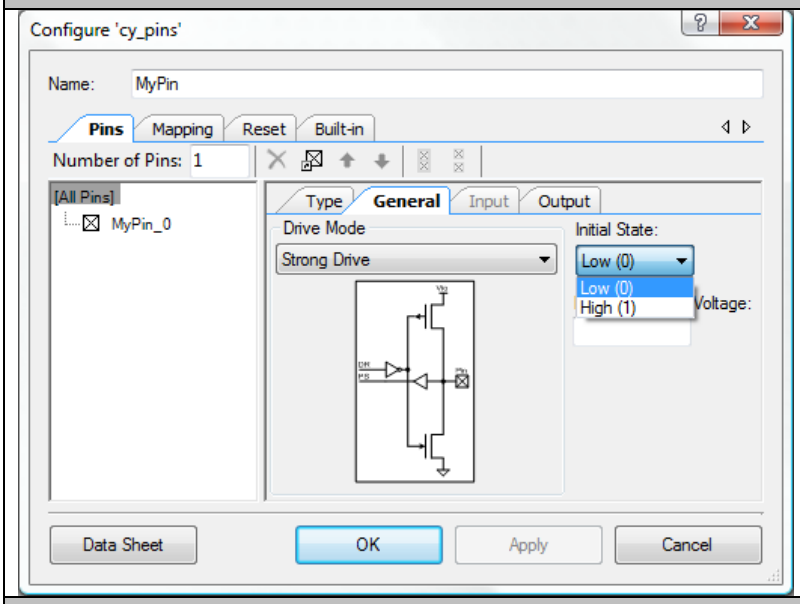
```
/* Set pin to Resistive Pull-up Using Component API */
MyPin_SetDriveMode(MyPin_DM_RES_UP);

/* Set pin to Resistive Pull-up Using Per-Pin API */
CyPins_SetPinDriveMode(CYREG_PRT1_PC2,PIN_DM_RES_UP);

/* Read Drive Mode Using Per-Pin API */
myVar = CyPins_ReadPinDriveMode(CYREG_PRT1_PC2);

/* Set pin to Resistive Up/Down Using Register Write */
CY_SET_REG8(CYREG_PRT1_PC2, CY_GET_REG8(CYREG_PRT1_PC2) | 0x07);
```

表 8. 初始状态参数

初始状态	
该参数指定了上电复位（POR）后写入引脚数据寄存器中的数值。	
PSoC Creator 配置窗口	**组件 API**
	<PinName>_Write() **引脚 API** CyPins_SetPin() CyPins_ClearPin() **相关寄存器** PRTx_DR[7:0]
示例代码	

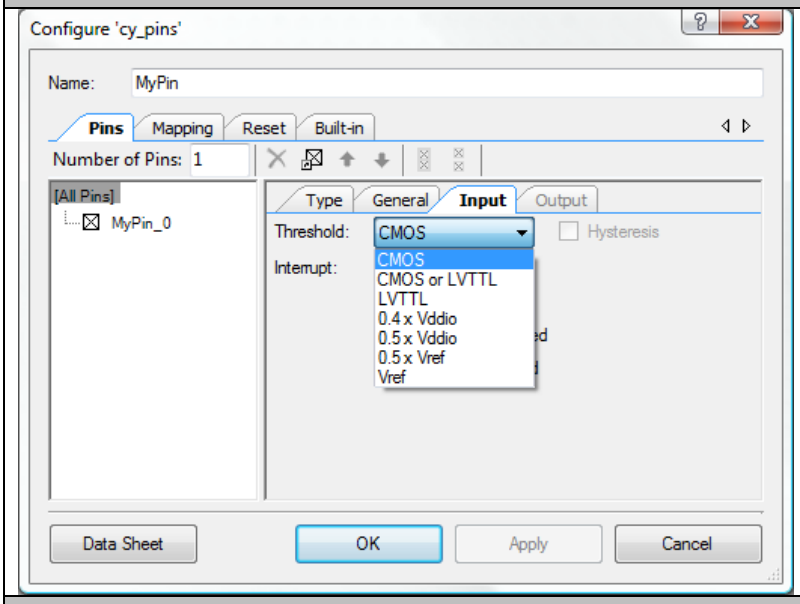
```
/* Set pin to logic state HIGH output */
MyPin_Write(1);

/* Set pin P1[2] to logic state HIGH output */
CyPins_SetPin(CYREG_PRT1_PC2);

/* Set pin P1[2] to logic state LOW output */
CyPins_ClearPin(CYREG_PRT1_PC2);

/* Set pin P1[2] output to logic state LOW using a register write */
CY_SET_REG8(CYREG_PRT1_DR, CY_GET_REG8(CYREG_PRT1_DR) | 0xFB);
```

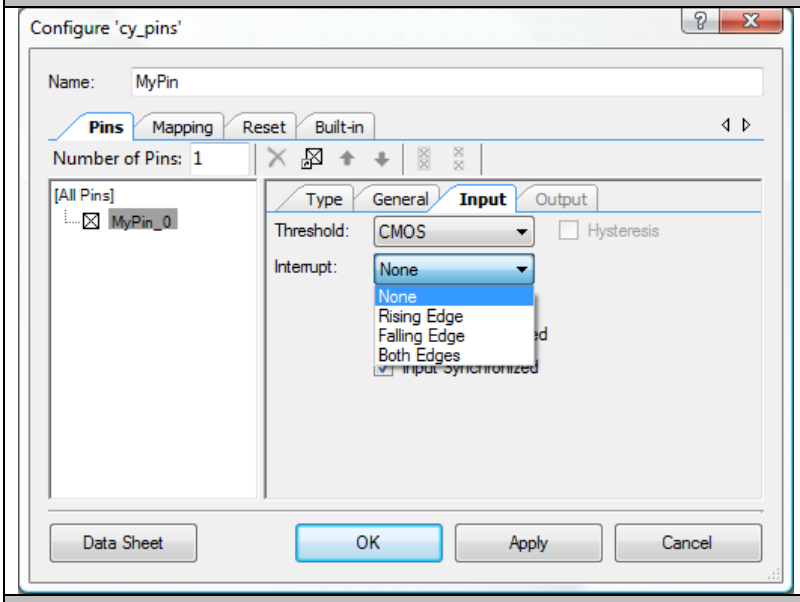
表 9. 阈值参数

阈值	
通过该参数可选择某个阀值用于定义逻辑高电平(1)和逻辑低电平(0)。该设置适用于端口上的所有物理引脚。只有 CMOS 和 LVTTL 设置适用于 GPIO 引脚。	
PSoC Creator 配置窗口	**组件 API**
	N/A **引脚 API** N/A **相关寄存器** PRTx_CTL[0]
示例代码	

```
/* Set port 1 logic threshold to CMOS using a register write */
CY_SET_REG8(CYREG_PRT1_CTL, 0);

/* Set port 1 logic threshold to LVTTL using a register write */
CY_SET_REG8(CYREG_PRT1_CTL, 1);
```

表 10. 中断参数

中断	
该参数用于选择 GPIO 的中断类型。	
PSoC Creator 配置窗口	**组件 API**
	N/A **引脚 API** N/A **相关寄存器** PICUx_INTTYPEy[1:0]
示例代码	

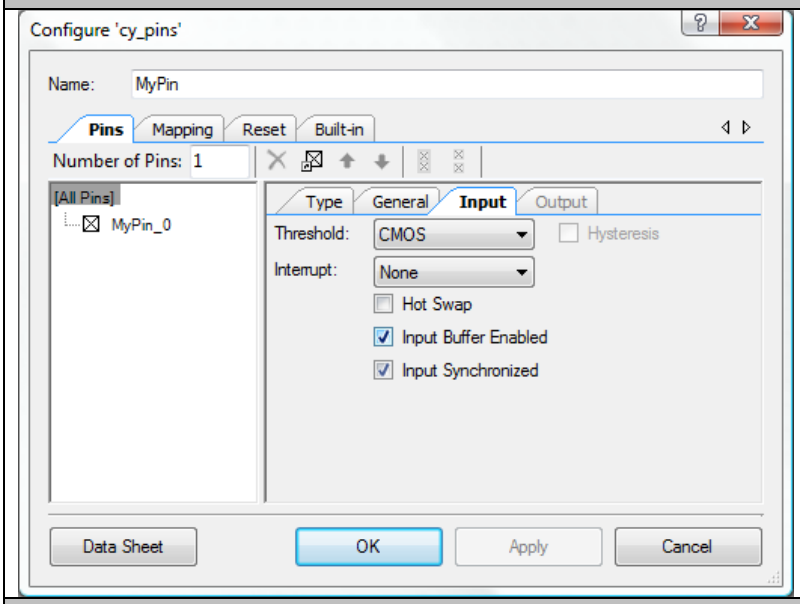
```
/* Disable interrupt on P1[2] */
CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x00);

/* Enable rising-edge interrupt on P1[2] */
CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x01);

/* Enable falling-edge interrupt on P1[2] */
CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x02);

/* Enable any edge interrupt on P1[2] */
CY_SET_REG8(CYREG_PICU1_INTTYPE2, 0x03);
```

表 11. 输入缓冲区使能参数

输入缓冲区使能	
该参数用于确定是否使能引脚的数字输入缓冲区。	
PSoC Creator 配置窗口	**组件 API**
	N/A **引脚 API** N/A **相关寄存器** PRTx_INP_DIS[7:0]
示例代码	

```
/* Disable Input Buffer on P1[2] using register write. */
CY_SET_REG8(CYREG_PRT1_INP_DIS, CY_GET_REG8(CYREG_PRT1_INP_DIS) | 0x04);
```

表 12. 输入同步参数

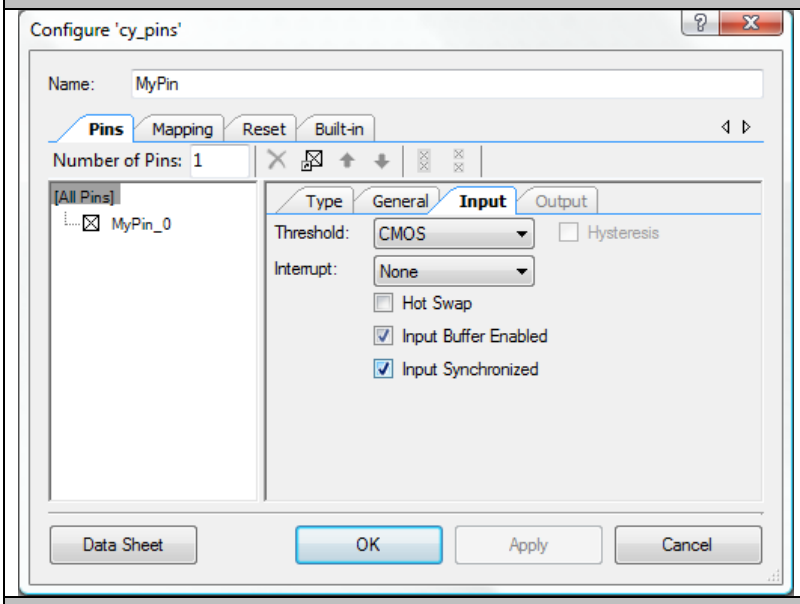
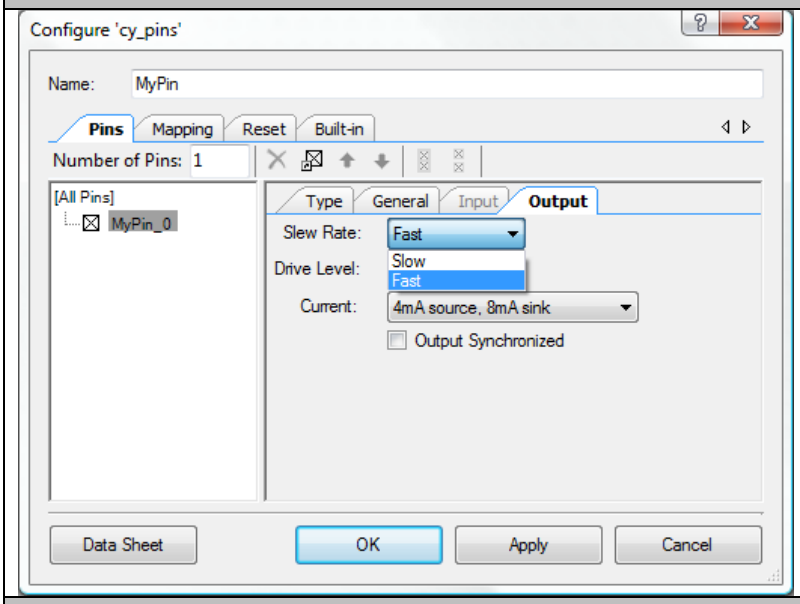
输入同步	
该参数用于使能引脚输入与总线时钟间的同步。	
PSoC Creator 配置窗口	**PSoC Creator 配置窗口**
	N/A **引脚 API** N/A **相关寄存器** PRTx_DBL_SYNC_IN[7:0]
示例代码	
`/* Sync input of P1[2] to bus_clk using register write. */` `CY_SET_REG8(CYREG_PRT1_DBL_SYNC_IN, CY_GET_REG8(CYREG_PRT1_DBL_SYNC_IN) | 0x04);`	

表 13. 斜率参数

斜率	
该参数用于确定随着输出逻辑电平变化，引脚上升和下降的速率。	
PSoC Creator 配置窗口	**组件 API**
	N/A **引脚 API** CyPins_FastSlew(CYREG_PRTx_PCy) CyPins_SlowSlew(CYREG_PRTx_PCy) **相关寄存器** PRTx_SLW[7:0]
示例代码	

```
/* Set fast edge rate for P1[2] */
CyPins_FastSlew(CYREG_PRT1_PC2);

/* Set slow edge rate for P1[2] */
CyPins_SlowSlew(CYREG_PRT1_PC2);

/* Set slow edge rate for P1[2] */
CY_SET_REG8(CYREG_PRT1_SLW, CY_GET_REG8(CYREG_PRT1_SLW) |= 0x02);
```

表 14. 输出同步参数

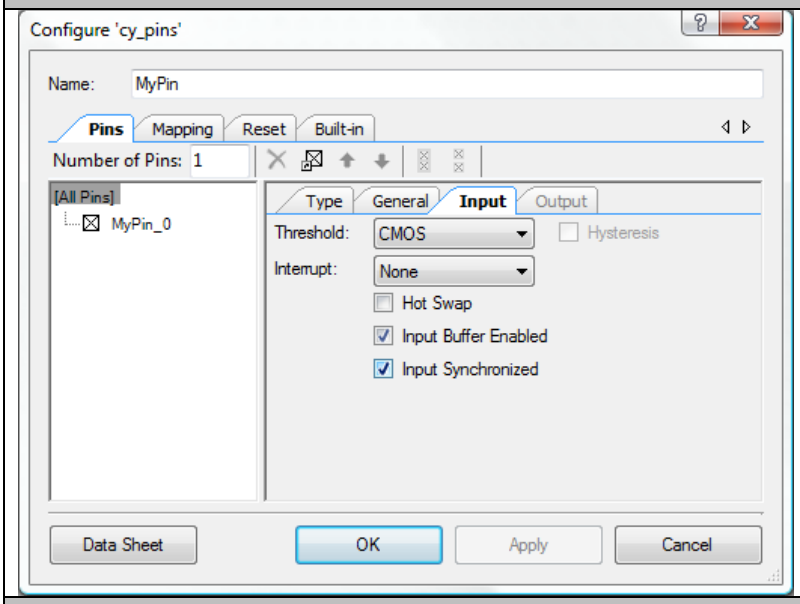
输出同步	
该参数用于使总线时钟和引脚的输出驱动器同步。	
PSoC Creator 配置窗口	**组件 API**
	N/A **引脚 API** N/A **相关寄存器** PRTx_SYNC_OUT[7:0]
示例代码	
`/* Sync output of P1[2] to bus_clk using register write. */` `CY_SET_REG8(CYREG_PRT1_SYNC_OUT, CY_GET_REG8(CYREG_PRT1_SYNC_OUT) | 0x04);`	

表 15. POR 参数

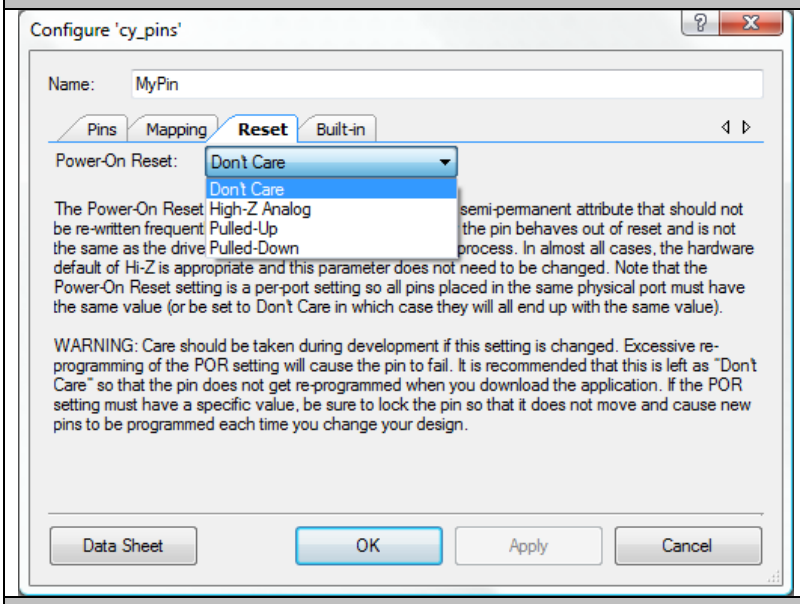
上电复位（NVL 阵列）	
该参数用于确定引脚在复位过程中的行为。它与操作驱动模式不同，后者是在启动过程中被配置的。请注意，POR 设置是针对每个端口进行的，要求相同物理端口内的所有引脚具有相同的值。该寄存器是 NVL 阵列的一部分，在正常操作期间不能被更改。	
PSoC Creator 配置窗口	**PSoC Creator 配置窗口**
	N/A
	引脚 API
	N/A
	相关寄存器
	PRTxRDM[1:0]
示例代码	
关于编程 NVL 阵列的详细信息和说明，请参阅 PSoC 3 器件编程规范或 PSoC 5LP 器件编程规范文档。	

表 16. 使能可选的 XRES 参数

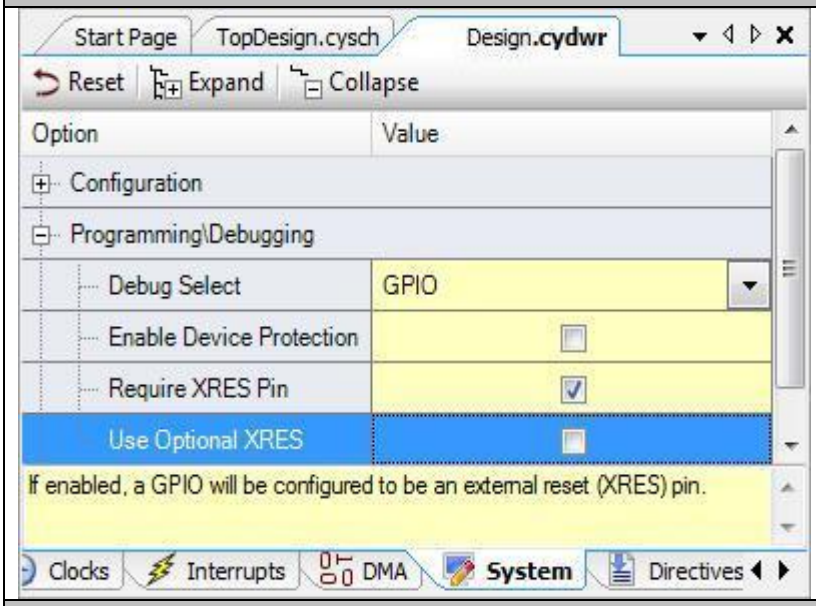
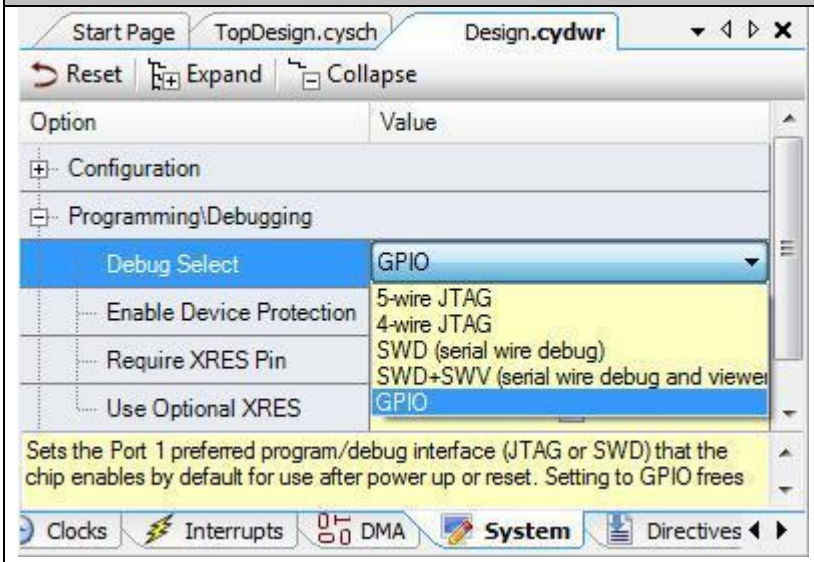
<table>
<tr><th colspan="2">使能可选的 XRES（NVL 阵列）</th></tr>
<tr><td colspan="2">该参数用于确定是否将 P1[2]配置为外部复位（XRES）引脚。该寄存器是 NVL 阵列的一部分，在正常操作期间不能被更改。</td></tr>
<tr><th>PSoC Creator 配置窗口</th><th>组件 API</th></tr>
<tr><td rowspan="5"> 
</td><td>N/A</td></tr>
<tr><th>引脚 API</th></tr>
<tr><td>N/A</td></tr>
<tr><th>相关寄存器</th></tr>
<tr><td>XRESMEN[0]</td></tr>
<tr><th colspan="2">示例代码</th></tr>
<tr><td colspan="2">关于编程 NVL 阵列的详细信息和说明，请参阅 PSoC 3 器件编程规范或 PSoC 5LP 器件编程规范文档。</td></tr>
</table>

表 17. 调试端口选择参数

调试端口选择（NVL 阵列）	
该参数用于设置首选的编程和调试接口。该寄存器是 NVL 阵列的一部分，在正常操作期间不能被更改。	
PSoC Creator 配置窗口	**组件 API**
	N/A **引脚 API** N/A **相关寄存器** DPS[1:0]
示例代码	
关于 NVL 阵列编程的详细信息和说明，请参阅 PSoC 3 器件编程规范或 PSoC 5LP 器件编程规范文档。	

文档修订记录

文档标题：AN72382 — 使用 PSoC® 3 和 PSoC 5LP GPIO 引脚

文档编号：001-93053

版本	ECN	变更者	提交日期	变更说明
**	4521562	MSON	10/15/2014	本文档版本号为 Rev**，译自英文版 001-72382 Rev*E。
*A	4802348	YANS	06/18/2015	本文档版本号为 Rev*A，译自英文版 001-72382 Rev*F。
*B	5967726	GJV	11/15/2017	文档无更改。本文档版本号为 Rev*B，译自英文版 001-72382 Rev*G。

销售、解决方案以及法律信息

全球销售和设计支持

赛普拉斯公司拥有一个由办事处、解决方案中心、原厂代表和经销商组成的全球性网络。如欲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

赛普拉斯开发者社区

论坛 | WICED IoT 论坛| 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709