AN2094

PSoC® 1 — GPIO 入门

作者：Meenakshi Sundaram R

相关器件系列：CY8C24x23A、CY8C24x94、CY8C21x34、CY8C20x34、CY8C21x23、CY8C21x45、CY8C22x45、 CY8C27x43、CY8C28xxx、CY8C29x66、CY7C64215、CYWUSB6953

相关应用笔记：无

要想获取本应用笔记的最新版本或相关的项目文件，请访问网站：http://www.cypress.com/AN2094。

AN2094 文档中介绍了有关通用输入和输出（GPIO）的信息，如驱动模式、映像寄存器和 GPIO 中断，以便进行 PSoC® 1 GPIO 入门操作。本文档还提供了一些提示，并对与 PSoC 1 GPIO 相关的其他资源进行了简要说明。

目录

1 简介 1
2 入门 3
2.1 PSoC Designer 3
2.2 代码示例 4
2.3 PSoC Designer 帮助 6
2.4 技术支持 6
3 GPIO 架构 7
4 GPIO 驱动模式 8
4.1 器件编辑器的配置 9
4.2 代码级配置 10
4.3 对某个端口进行读取和写入操作 11
4.4 修改某个 GPIO 引脚的驱动模式 12
5 映像寄存器 14
5.1 映像寄存器的使用情况 14
6 GPIO 中断 16
6.2 使用中断时该进行和不应进行的操作 20
7 其他 GPIO 资源和提示 22
7.1 GPIO 全局选择寄存器 22
7.2 模拟复用器（AMUX）总线控制寄存器 22
7.3 命名一个引脚 22
7.4 寄存器及其相关寄存器组 25
8 示例项目 26
8.1 项目 1：检测 LED 驱动模式 26
8.2 项目 2：映像寄存器的使用情况 29
8.3 项目 3：使用中断切换 LED 31
8.4 额外的代码示例 32
文档修订记录 33
销售、解决方案以及法律信息 34
全球销售和设计支持 34

1 简介

通用的输入和输出（GPIO）是所有微控制器（MCU）的关键部分，通过它 MCU 与外界建立连接。最终应用将决定了连接外界的该桥接器的类型和性能。PSoC 具有强大且灵活的通用 I/O（GPIO）引脚；与传统 MCU 相比，它们能提供更多功能。

例如，某个 ADC 要求将一个 GPIO 作为模拟输入，但某个 I^2C 或 SPI 数字通信模块则要求将该 GPIO 作为一个数字接口。为了正确设置该桥接器，你需要了解最终应用和 MCU 中所使用的 GPIO 系统。与其他微控制器类似，PSoC 拥有自己的 GPIO 系统。

本应用笔记介绍了 GPIO 系统特定的应用参数。有关该系统的详细技术概述，请参阅相应器件技术参考手册中“PSoC 内核”部分的通用 I/O 章节。

本应用笔记包括下面各个主题：

- **GPIO 驱动模式：**介绍了 PSoC1 中各种可用的驱动模式及其用途。此外，还描述了在固件中重新动态配置驱动模式的流程，并提供了相应的示例。

- **映射寄存器：**介绍了 GPIO 映像寄存器的概念及其用途，并提供了相应的示例。
- **GPIO 中断：**提供了一个使用中断进行 LED 切换的简单示例，用以说明 PSoC1 中的 GPIO 中断。

本文档假设您已经熟悉了 PSoC Designer™ IDE。

2 入门

赛普拉斯网站 www.cypress.com 上提供了大量资料，有助于选择符合您设计的 PSoC 器件，并能够快速有效地将该器件集成到您的设计中。有关使用资源的完整列表，请参考基础知识文章如何使用 PSoC® 1、PowerPSoC®以及 PLC 进行设计 — KBA88292。下面是 PSoC 1 的简要列表：

- 概述：PSoC 产品系列、PSoC 产品蓝图
- 产品选型器：PSoC 1、PSoC 3、PSoC 4、PSoC 5LP
- 此外，PSoC Designer 还包含了一个器件选择工具
- 应用笔记：赛普拉斯提供了大量 PSoC 应用笔记，包括了从基础到高级的广泛主题。下面列出了 PSoC 1 入门的应用笔记：
  - AN75320 — PSoC® 1 入门
  - AN2094 — PSoC® 1 — GPIO 入门
  - AN74170 — PSoC® 1 模拟结构和利用 PSoC Designer™进行的配置
  - AN2041 — 了解 PSoC® 1 开关电容模拟模块
  - AN2219 — PSoC® 1 选择模拟接地和参考电压

注意：欲了解与 CY8C29X66 器件相关的应用笔记，请点击此处。

- 开发套件：
  - 除了 CY8C25/26xxx 外，CY3210-PSoCEval1 支持所有 PSoC 1 混合信号阵列系列（包括汽车级器件）。该套件包括 LCD 模块、电位器、LED 和实验板空间。
  - CY3214-PSoCEvalUSB 可作为 CY8C24x94 PSoC 器件的开发电路板使用。该开发板的特殊之处在于它支持 USB 和 CapSense 的开发和调试功能。

注意：欲了解与 CY8C29X66 器件相关的开发套件，请点击此处。

MiniProg1 和 MiniProg3 器件提供了多个用于进行闪存编程和调试的接口。

2.1 PSoC Designer

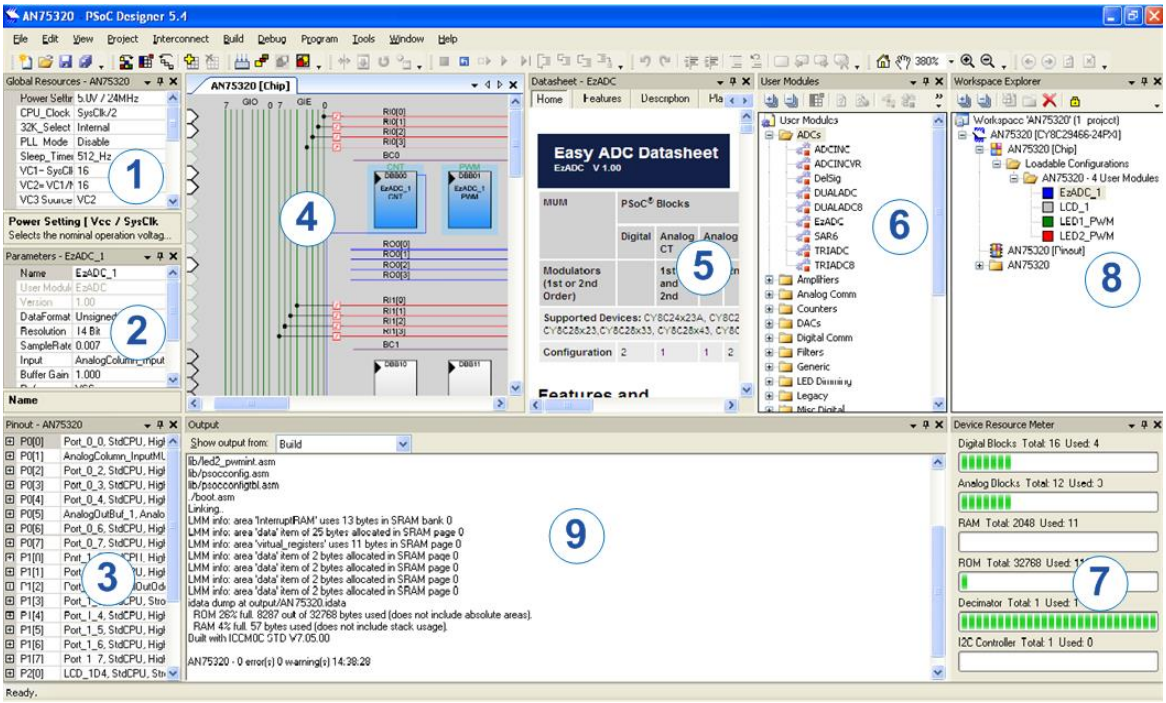
PSoC Designer 是一个基于 Windows 的免费集成开发环境（IDE）。在拖放式设计环境中使用预先设定的模拟和数字外设库来开发您的应用。然后，利用动态生成的 API 代码库来自定义您的设计。图 1 显示了 PSoC Designer 窗口。

注意：这并不是默认窗口。

1. 全局资源 — 所有器件的硬件设置。
2. 参数 — 当前选中的用户模块的参数。
3. 引脚分布 — 器件引脚的相关信息。
4. 芯片级编辑器 — 选中芯片上可用资源的框图。
5. 数据手册 —当前选中的用户模块的数据手册。
6. 用户模块 — 选中器件的所有可用的用户模块。
7. 器件资源计 — 当前项目配置的器件资源使用率。
8. 工作区 — 与项目有关的文件树级图。
9. 输出 —项目构建和调试操作的输出。

注意：欲详细了解 PSoC Designer 的信息，请依次打开 PSoC Designer IDE > **Help** > **Documentation** > “Designer Specific Documents” 文件夹 > “IDE User Guide.pdf”文档。

图 1. PSoC Designer 布局



2.2 代码示例

以下网站列出了 PSoC Designer 的代码示例。这些代码示例通过向您提供一个完整的设计（而不是空白设计）加快您的设计过程，并显示了如何将 PSoC Designer 用户模块用于各种应用。

http://www.cypress.com/documentation/code-examples/psoc-1-code-examples

要想访问集成在 PSoC Designer 中的代码示例，请依次选择 **Start Page > Design Catalog > Launch Example Browser**，如图 2 所示。

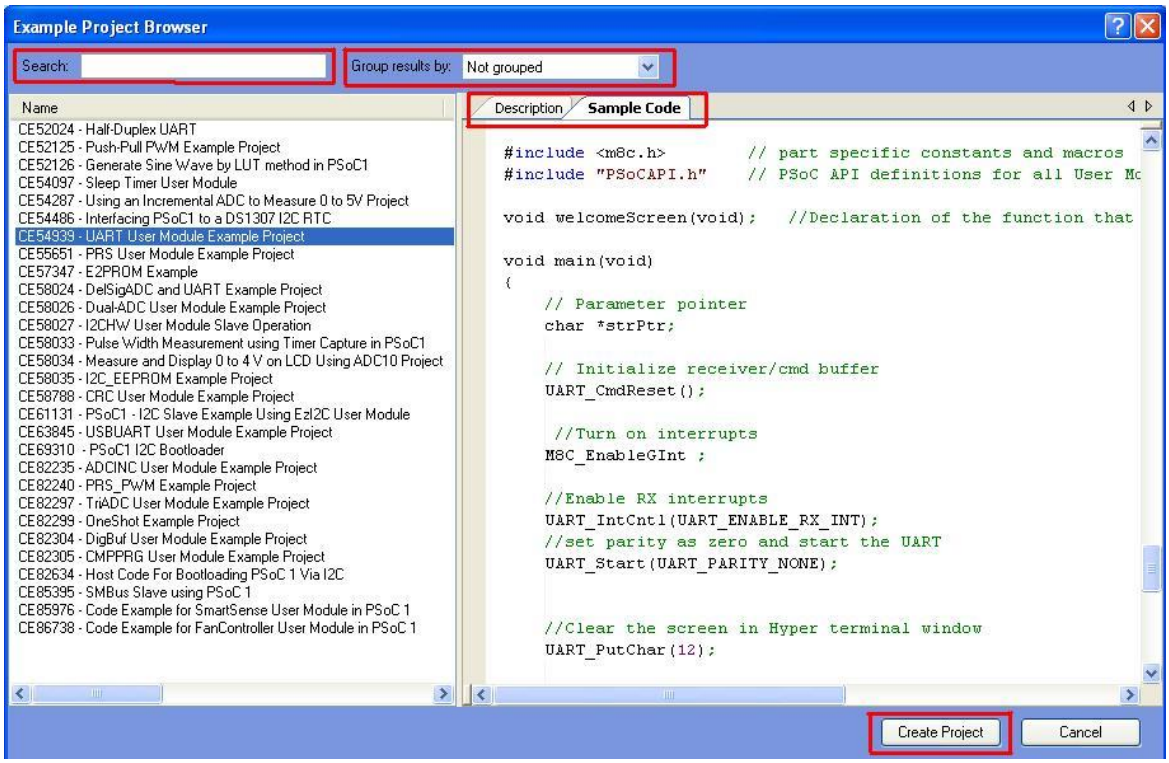
图 2. PSoC Designer 中的代码示例



在图 3 中显示的示例项目浏览器提供了以下选项内，您可以：

- 通过关键词搜索筛选项目。
- 根据类别列出项目。
- 在 Description（说明）选项卡上查看已选项目的数据手册。
- 查看已选项目的代码示例。您可以复制该窗口中的代码然后将其粘贴在您的项目内，从而加快代码的开发过程，或
- 同样可以根据所选的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，可以加快您的设计进程。然后您可以将该设计适用于您的应用中。

图 3. 带示例代码的代码示例项目



2.3 PSoC Designer 帮助

请访问 PSoC Designer 主页以下载 PSoC Designer 的最新版本。启动 PSoC Designer 并导航到下列各项：

- **IDE 用户指南**：依次选择 **Help > Documentation > Designer Specific Documents > IDE User Guide.pdf**。本指南提供了开发 PSoC Creator 项目的基本知识。
- **简单用户模块代码示例**：依次选择 **Start Page > Design Catalog > Launch Example Browser**。这些代码示例展示了如何配置及使用 PSoC Designer 用户指南。
- **技术参考手册**：依次选择 **Help > Documentation > Technical Reference Manuals**。本手册列出并阐述了 PSoC 1 器件的系统功能。
- **用户模块数据手册**：右键单击一个用户模块并选择“Datasheet”。本数据手册说明了所选用户模块的各个参数和 API。
- **器件数据手册**：依次选择 **Help > Documentation > Device Datasheets**，从而获得 PSoC 1 器件的特殊数据手册。
- **Imagecraft 编译器指南**：依次选择 **Help > Documentation > Compiler and Programming Documents > C Language Compiler User Guide.pdf**。该指南提供了有关 Imagecraft 编译器特的殊指令和函数的详细信息。

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出页面上创建一项技术支持。

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。

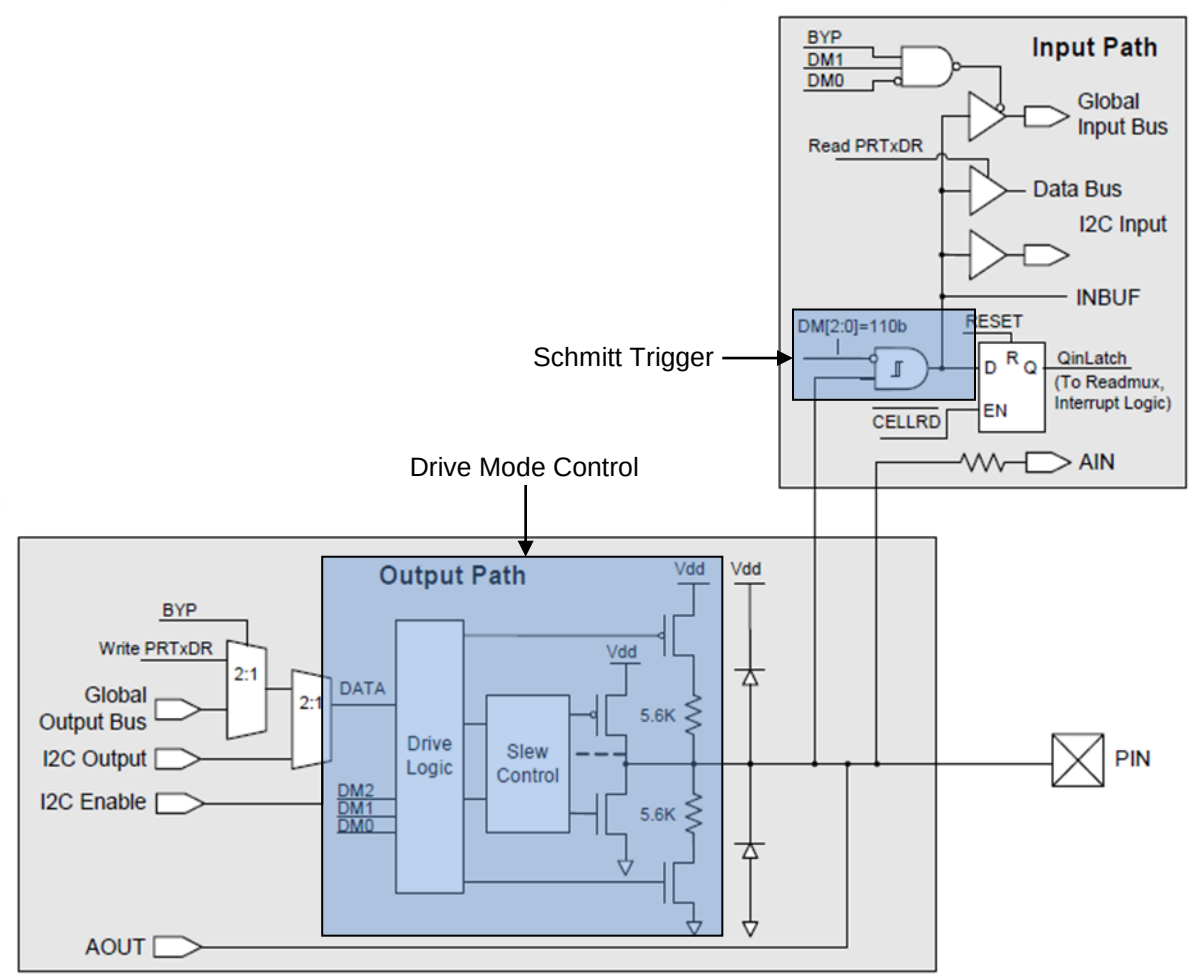
若想获得快速支持，您同样可以使用下面的支持资源。

- 自助
- 所在地销售办事处

3 GPIO 架构

图 4 显示了 PSoC1 GPIO 的架构。

图 4. PSoC1 中的 GPIO 单元架构



GPIO 单元中有两个重要部分，即输入路径和输出路径。

输入路径带有一个 Schmitt 触发器电路。对于数字信号，该电路可作为引脚和 MCU 间的接口。Schmitt 触发器的输出路径可连接至 MCU 的数据总线、全局输入总线、I²C 输入、中断控制器等。该引脚还能直接与连接至 PSoC 器件中模拟模块的内部模拟总线相连。

输出路径带有一个驱动模式选择逻辑，可以将引脚配置为 8 种驱动模式中的任何一个。驱动模式所选择的输入来自内部数据总线，这些总线包括数据寄存器、I²C 总线和全局输出总线（与数字模块的输出相连接）。此外，通过（选择引脚上的）内部模拟总线，还可以连接至模拟输出缓冲器。

4 GPIO 驱动模式

为了配置某个 GPIO 引脚，PSoC 1 提供了八种驱动模式，如表 1 所示。图 5 显示了每一种驱动模式下的 GPIO 单元配置。

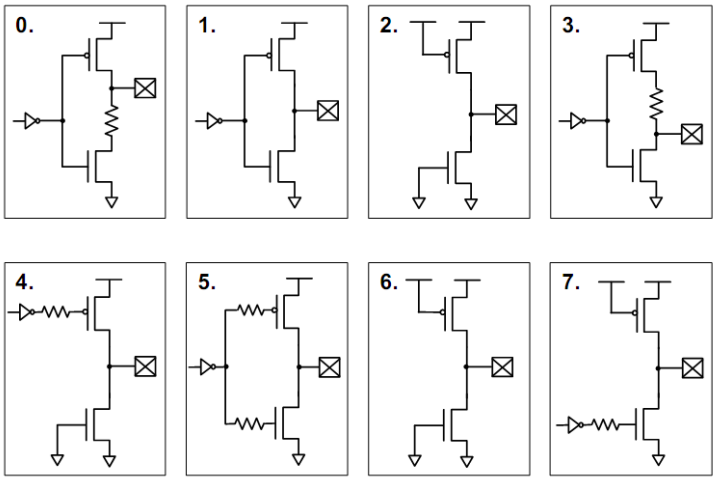
表 1. 驱动模式的详细信息

序号	驱动模式	说明	应用
1	高阻态	高阻抗数字输入模式。在该模式下，引脚作为数字输入使用。 此外，将‘1’或‘0’写入该引脚不会产生任何影响	通过结合使用数字输入和下面各部分，可以将该数字输入连接到某个信号源：一个强驱动电流、一个上拉或下拉电阻、与外部上拉电阻一起使用的集电极开路或与外部下拉电阻一起使用的发射极开路。
2	高阻模拟	高阻抗模拟模式。在这种模式下，该引脚作为模拟引脚使用。该模式与数字高阻模式相似，但 GPIO 单元中的数字输入电路（图 4 中所显示的 Schmitt 触发器）被禁用。如果您将该引脚作为数字输入使用，请确保选择的是“HighZ”驱动模式，而不是“HighZ Analog”驱动模式。	用于模拟输入引脚或被设置为模拟输入和输出的选择引脚。
3	开漏高电平（ODH）	在该模式下，写入‘1’会把该引脚上的电压驱动为 V_{DD}；写入‘0’将使该引脚进入高阻抗状态。该配置同发射极开路的配置相似。	用于提供一个带有外部下拉电阻的发射极开路接口。该驱动模式能够完成一个有线“OR”运算连接。
4	开漏低电平（ODL）	在该模式下，写入‘1’会把该引脚上的电压驱动为 GND；写入‘0’会使该引脚进入高阻抗状态。这种配置与集电极开路的配置相同。	用于提供一个带有外部上拉电阻的集电极开路接口。该驱动模式能够实现一个有线“AND”运算连接。示例 — I^2C 引脚。
5	强驱动	在这种模式下，写入‘1’会把该引脚上的电压驱动为 V_{DD}；写入‘0’会该引脚的电压驱动为 GND。	数字输出引脚。
6	下拉	在这种模式下，写入‘1’时，会把该引脚上的电压驱动为 V_{DD}；写入‘0’时，会通过一个电阻（5.6 kΩ 左右）将该引脚驱动为 GND。	用于连接一个带有发射极开路输出的信号或连接一个与 V_{DD} 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。
7	上拉	在这种模式下，写入‘1’时，可以通过一个电阻（5.6 kΩ 左右）将该引脚的电压驱动为 V_{DD}；写入‘0’时，会将该引脚驱动为 GND。	用于连接一个带有集电极开路输出的信号（如来自马达的转速计信号）或连接一个与 GND 相连的开关。此外，还可以将其作为一个输出，用于连接拉电流模式中的各个 LED。
8	慢速强驱动	该模式与强驱动模式相似，但会对输出的斜率稍微进行控制，这样在输出开关时不会出现高次谐波。	并将其作为一个辐射干扰已被降低的数字输出。

图 5. 驱动模式配置的详细信息

Drive Modes

Drive Mode	Diagram Number	Data = 0	Data = 1
Resistive Pull Down	0	Resistive	Strong
Strong Drive	1	Strong	Strong
High Impedance	2	High Z	High Z
Resistive Pull Up	3	Strong	Resistive
Open Drain, Drives High	4	High Z	Strong (Slow)
Slow Strong Drive	5	Strong (Slow)	Strong (Slow)
High Impedance Analog	6	High Z	High Z
Open Drain, Drives Low	7	Strong (Slow)	High Z

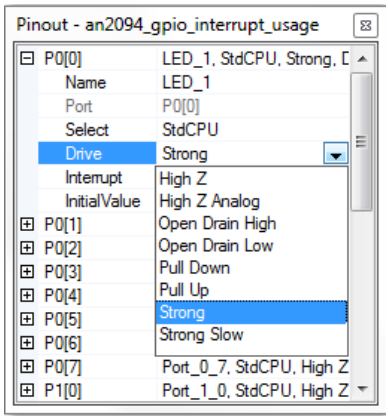


可以通过下面两种方法配置 GPIO。第一方法：将配置操作定义为 PSoC Designer 器件编辑器中初始化过程的一部分。该方法对固定的引脚配置很有用。第二种方法：配置固件中的引脚。通过该方法，可以在运行过程中灵活配置 GPIO。

4.1 器件编辑器的配置

通过使用器件编辑器的引脚分布视图模式，可以配置 I/O 引脚。在引脚分布视图模式中，PSoC Designer 接口的左下角将显示一个表格（此窗口的位置取决于各个窗口被安排的位置）。图 6 显示的是该表格。

图 6. PSoC Designer 引脚分布窗口（“驱动”列表）



下面介绍的是图 6 中所显示的各个字段：

1. **Name** 字段显示的是引脚的名称。您可以对引脚进行重命名，使其用途更加明确。对引脚进行重命名时，将在 *PSoCGPIOINT.inc* 和 *PSoCGPIOINT.h* 文件内生成该引脚的各个宏，如引脚数据寄存器、引脚屏蔽以及驱动模式寄存器。命名一个引脚一节中详细介绍了相关信息 。
2. **Port** 字段显示的是引脚的物理映射。不能对该字段进行编辑操作。
3. 通过 **Select** 字段，可以配置各引脚下面的特殊性能：
   a. **AnalogInput**：只有端口 0 和端口 2 才有额外的模拟输入和模拟输出选项。模拟输入获取来自外界的模拟信号，然后将它们连接至模拟列输入复用器或直接连接至 PSoC 模块。例如，如果您使用的是某个 ADC，则至少要将一个引脚配置为模拟输入，以获取来自外界的模拟信号。
   b. **AnalogOutputBuf**：在某些器件系列中，端口 0 上的某些引脚被连接至内部模拟输出缓冲器上。
   c. **Default**：未连接全局总线，并且驱动强度为高阻抗模拟。
   d. **StdCPU**：通过端口的普通 I/O。CPU 通过 PRTxDR 数据寄存器控制该 I/O[1]。
   e. **Global_IN, Global_OUT**：通过使用全局输入和输出，可以将时钟和数据信号路由到数字 PSoC 模块上。如果您将某个引脚配置为一个 Global_IN（输入）或 Global_OUT（输出），那么，该引脚可以与该数字模块相连。例如，如果选中 Global_IN 项，那么并能够将特定的引脚连接至 Global_INPUT 总线。总线被作为数字 PSoC 模块的输入。
   f. **AnalogMuxInput**[2]：您可以将该引脚同模拟复用器总线连接起来。可将该总线路由到 PSoC 器件中的各种模拟模块。

      除上述的各种引脚类型外，本文档还介绍并罗列了某些具有特殊功能的引脚。例如，P1[0]和 P1[1]具有 XtalOut 和 XtalIn，P1[4]具有 ExtSysClk，P1[5]和 P1[7]具有 I2C_SDA 和 I2C_SCL 等。
   g. **Drive** 字段用于设置引脚的驱动模式，如表 1 和图 5 所示。
   h. 通过引脚分布窗口中的 **Interrupt** 字段进行设置各个引脚的中断类型。这些引脚可能具有上升沿中断、下降沿中断，或两者均有。请参见 GPIO 中断。
   i. 引脚分布窗口中的 **Initial Value** 字段用于在启动时设置引脚的初始输出值。在执行自动生成的启动代码过程中，通过填入引脚的数据寄存器可以确定该初始值。
   j. **AnalogMUXBus**[2]：在芯片编辑器内使能或禁用该引脚与 AMUX 总线间的连接。如果要在固件上实现同样的操作，请参见模拟复用器（AMUX）总线控制寄存器。

4.2 代码级配置

配置 I/O 引脚的另一种方法是，通过使用汇编语言或 C 语言，直接修改固件中[3] 的相关寄存器。通过该方法，您可以在执行程序过程中动态配置各个 I/O 端口。下面各寄存器控制一个 GPIO：

1. **PRTxDR register**：它是数据寄存器，用于控制某个引脚的输出状态。每个端口均有一个相应的数据寄存器，其中每一位代表一个引脚。例如，PRT0DR 控制着端口 0；PRT0DR 中的位#0 控制着 P0[0]。通过 PRTxDR 寄存器，可以读取某个 GPIO 引脚的状态。请参见对某个端口进行读取和写入操作一节中的内容。
2. **PRTxDMx registers**：每个端口均有三个用于控制驱动模式的寄存器。通过对这些寄存器进行写操作，可以在运行时自动更改各个引脚的驱动模式。请参见修改某个 GPIO 引脚的驱动模式。
3. **PRTxICx registers**：每个端口均有两个用于控制引脚中断类型的寄存器。中断类型包括上升沿、下降沿、状态翻转以及无中断。通过对这些寄存器进行写操作，可以在运行时动态更改某个引脚的中断类型。请参见 GPIO 中断。

[1] CPU 只有在这种模式下才可以控制引脚输出的状态，即如果对端口数据寄存器进行写操作，相应的引脚上将出现更新结果。

[2] 仅在下面各器件系列中可用：CY8C21x34、21x45、22x45、 24x94 和 28xxx。与（前面所述的器件中可用的）AnalogMuxBus 字段一起使用

[3] 如果引脚配置是固定的，那么配置这些引脚时不再需要使用用户编写的代码。根据器件编辑器中的设置，PSoC Designer 将自动生成启动代码， 以配置该引脚。

4. **PRTxGS register**：该寄存器用于将 GPIO 引脚连接至全局输入或全局输出总线，或者断开它们之间的连接。请参见 GPIO 全局选择寄存器。
5. **MUX_CRx register**：在具有模拟复用器总线的器件中，使用该寄存器将某个引脚与该模拟复用器总线连接起来，或者断开它们之间的连接。每个寄存器均代表一个端口，并且该寄存器中的每一位都控制着相应端口引脚。例如，MUX_CR0 寄存器中的位#0 控制的是 P0[0]。

4.3 对某个端口进行读取和写入操作

当通过清除 PRTxGS 寄存器中的相应位（通过在 GPIO 配置窗口中将某个端口引脚配置为 StdCPU），可断开该引脚与全局总线间的连接，并且当该引脚的驱动模式为非 *HighZ* 或 *HighZ Analog* 时，通过对 PRTxDR 寄存器进行写操作，可以控制该引脚的状态。

端口 0 数据寄存器（PRT0DR，地址 = 组 0，00h）

端口 1 数据寄存器（PRT1DR，地址 = 组 0，04h）

端口 2 数据寄存器（PRT2DR，地址 = 组 0，08h）

端口 3 数据寄存器（PRT3DR，地址 = 组 0，0Ch）

端口 4 数据寄存器（PRT4DR，地址 = 组 0，10h）

端口 5 数据寄存器（PRT5DR，地址 = 组 0，14h）

端口 6 数据寄存器（PRT6DR，地址 = 组 0，18h）

端口 7 数据寄存器（PRT7DR，地址 = 组 0，1Ch）

要想对某个特定的端口引脚进行写操作，请使用相应的位掩码，并对所需位进行 AND 或 OR 运算。例如，如果要设置和清除 P0[0]：

在汇编程序中：

```
or  reg[PRT0DR],  0x01 ; Set P0[0]
and  reg[PRT0DR],  ~0x01  ;  Clear
P0[0]
```

在 C 语言程序中：

```
PRT0DR |= 0x01; // Set P0[0]
PRT0DR  &=  ~0x01;  //  Clear  P0[0]
```

要想读取某个端口引脚，则读取 PRTxDR 寄存器并使用相应的位屏蔽。例如，如要检查 P0[1]的状态并更新 P0[0]上 LED 的状态。

在汇编程序中：

```
mov A, reg[PRT0DR]
   and A, 0x02
   jnz PinHigh

   ; Code to process Pin cleared
state
   and  reg[PRT0DR],  ~0x01  ;  Set
P0[0]
   jmp Exit

PinHigh:
```

```
    ; Code to process Pin set state
    or  reg[PRT0DR],  0x01   ;  Clear
P0[0]

Exit:
```

在 C 语言程序中：

```
if (PRT0DR & 0x02)
{
    // Code to process Pin Set state
    PRT0DR |= 0x01; // Set P0[0]
}
else
{
    // Code to process Pin cleared
state
    PRT0DR &= ~0x01; // Clear P0[0]
}
```

如果在 GPIO 配置窗口中给某个端口引脚提供了一个有意义的名称，那么通过使用 PSoC Designer 所生成的引脚宏，可以访问数据寄存器和引脚屏蔽。请参见命名一个引脚。

4.4 修改某个 GPIO 引脚的驱动模式

每个端口均有三个寄存器，这三个寄存器用于设置所有端口引脚的驱动模式。它们分别为 PRTxDM0、PRTxDM1 和 PRTxDM2，其中“x”代表端口号。这三个寄存器中任何一个的某一位可以配置一个特殊的引脚。例如，PRT0DM0、PRT0DM1 和 PRT0DM2 中的位 0 控制着 P0[0]驱动模式。表 2 介绍了相应的详细信息。

表 2. 驱动模式寄存器值

PRTxDM2[n]	PRTxDM1[n]	PRTxDM0[n]	驱动模式
0	0	0	电阻下拉
0	0	1	强驱动
0	1	0	高阻抗 — 数字
0	1	1	电阻上拉
1	0	0	开漏 — 高电平
1	0	1	慢速强驱动
1	1	0	高阻抗 — 模拟
1	1	1	开漏 — 低电平

在表 2 中，‘x’表示端口编号，‘n’表示驱动模式寄存器中的相应位和要配置的端口引脚。例如，要想将端口 0 引脚 1 配置为电阻下拉，请清除 RT0DM0、PRT0DM1 和 PRT0DM2 寄存器中的位 1。更多信息，请参器件技术参考手册。

请特别注意，所有的 PRTxDM0 和 PRTxDM1 寄存器都位于寄存器组 1（有关寄存器组的详细信息，请参见技术参考手册）中，而所有的 PRTxDM2 寄存器都位于寄存器组 0 中。如果想在汇编语言程序中使用驱动模式寄存器，需要了解上述情况，因为在访问寄存器前，需要先选择寄存器组。在 C 语言程序中，编译器将根据所使用的寄存器进行分配寄存器组。

在汇编程序中：

```
M8C_SetBank1
or   reg[PRT2DM0], 0x20
and  reg[PRT2DM1], ~0x20
M8C_SetBank0
and  reg[PRT2DM2], ~0x20
```

在上述汇编示例中，第一行调用 M8C_SetBank1 宏，以将寄存器组切换为 1。之所以实现该操作，是因为 PRT2DM0 和 PRT2DM1 均位于寄存器组 1 中。通过使用 OR 指令和 0x20 屏蔽，可以设置 PRT2DM0 寄存器中的第 5 位。然后，通过使用 AND 指令和 0X20 的反转屏蔽，可以清除 PRT2DM1 寄存器中的第 5 位。

如果使用 M8C_SetBank0，变能够切回到寄存器组 0。此外，通过使用 AND 指令和 0x20 的反转屏蔽，可以清除第 5 位或 PRT2DM2 寄存器。OR 和 AND 指令为读取、修改或写入指令。首先读取寄存器中的内容，然后对其进行 OR 或 AND 运算，最后把结果写入到相同的寄存器内。通过这种方法，您可以修改某个特定位而不影响其他位。

在 C 语言程序中：

```
PRT2DM0 |= 0x20;
PRT2DM1 &= ~0x20
PRT2DM2 &= ~0x20;
```

在 C 语言程序中，由于 C 编译器负责切换各个寄存器组，因此会使代码变得更加简单。但必须同时使用“位 AND”（&=）或“位 OR”（|=）运算和寄存器中的相应屏蔽。

如果在 GPIO 配置窗口中重新对某个端口引脚进行命名，那么通过使用 PSoC Designer 所生成的引脚宏，可以访问驱动模式寄存器。请参见命名一个引脚。

示例#1 中介绍了 GPIO 驱动模式的修改情况。您可以从相应应用笔记的网页上下载该示例。有关该示例项目的详细信息，请参考项目 1：检测 LED 驱动模式。

5 映像寄存器

5.1 映像寄存器的使用情况

在众多设计中，某个端口可以同时有输入引脚（如带有上拉或下拉输入的开关）和输出引脚（如某个 LED）。在该设计中，用于更新输出引脚的指令可能永远使输入引脚锁存为 1 或 0。

例如，请考虑下面的情况：P0_1 上的某个开关输入被配置为下拉模式（该开关连接引脚至 V_{DD}）。同时，P0_0 上有一个 LED，它跟随 P0_1 的开关状态。此外，该引脚被配置作为强驱动（输出引脚）。可以通过下面的代码实现上述配置。

```
if (PRT0DR & 0x02)
{
   // Switch pressed. Turn on the LED
   PRT0DR |= 0x01; // Set P0[0]
}
else
{
   // Switch released. Turn off the LED
   PRT0DR &= ~0x01; // Clear P0[0]
}
```

现在，假设已经按下开关，因而需要打开该 LED。代码“PRT0DR |= 0X01”是一个读取-修改-写入指令，它执行以下操作：

1. 读取 — PRT0DR = x x x x x x 1 0 (位 0 = 0 ➔ LED 关闭；位 1 = 1 ➔ 开关被按下）
2. 修改 — （PRT0DR | 00000001） = x x x x x x 1 1
3. 写入 — PRT0DR = x x x x x x 1 1 （位 0 = 1 →LED 打开；设置位 1 可以将该引脚内部连接至 V_{DD}，因为在下拉驱动模式中，如果向该引脚写入 1，则可以将其连接至 V_{DD}）。
4. 在第 3 步骤中，已经将 1 写入到了 P0[1]内，这样该引脚的状态将始终为 1。即使开关被释放，数据寄存器仍会将该引脚驱动为 1，并且代码始终会在开关输入上读取得到 1。

为解决该问题，可以在这些端口（具有输入和输出的结合）上使用一个被称为映像寄存器的变量。

当您使用某个映像寄存器时，将通过该变量实现所有对引脚执行的写操作。应该在适当设置了输入引脚的状态条件下初始化该变量。对于上拉或开漏低电平输入引脚，应将该寄存器的值设置为 1；对于下拉或开漏高输入引脚，则应将该值设置为 0。需要更改输出引脚的状态时，应在映像寄存器上执行读取-修改-写入指令，然后将该寄存器的内容复制到端口数据寄存器内。在端口数据寄存器上直接执行所有的读取操作。通过下面代码可以实现映像寄存器：

```
// Use 'extern' when using ShadowRegs UM
extern BYTE Port_0_Data_SHADE;

// Using shadow variables in code
if(PRT0DR & 0x02)
{
       Port_0_Data_SHADE |= 0x01;
       PRT0DR = Port_0_Data_SHADE;
}

else
{
       Port_0_Data_SHADE &= ~0x01;
       PRT0DR = Port_0_Data_SHADE;
}
```

现在，读取-修改-写入操作如下：

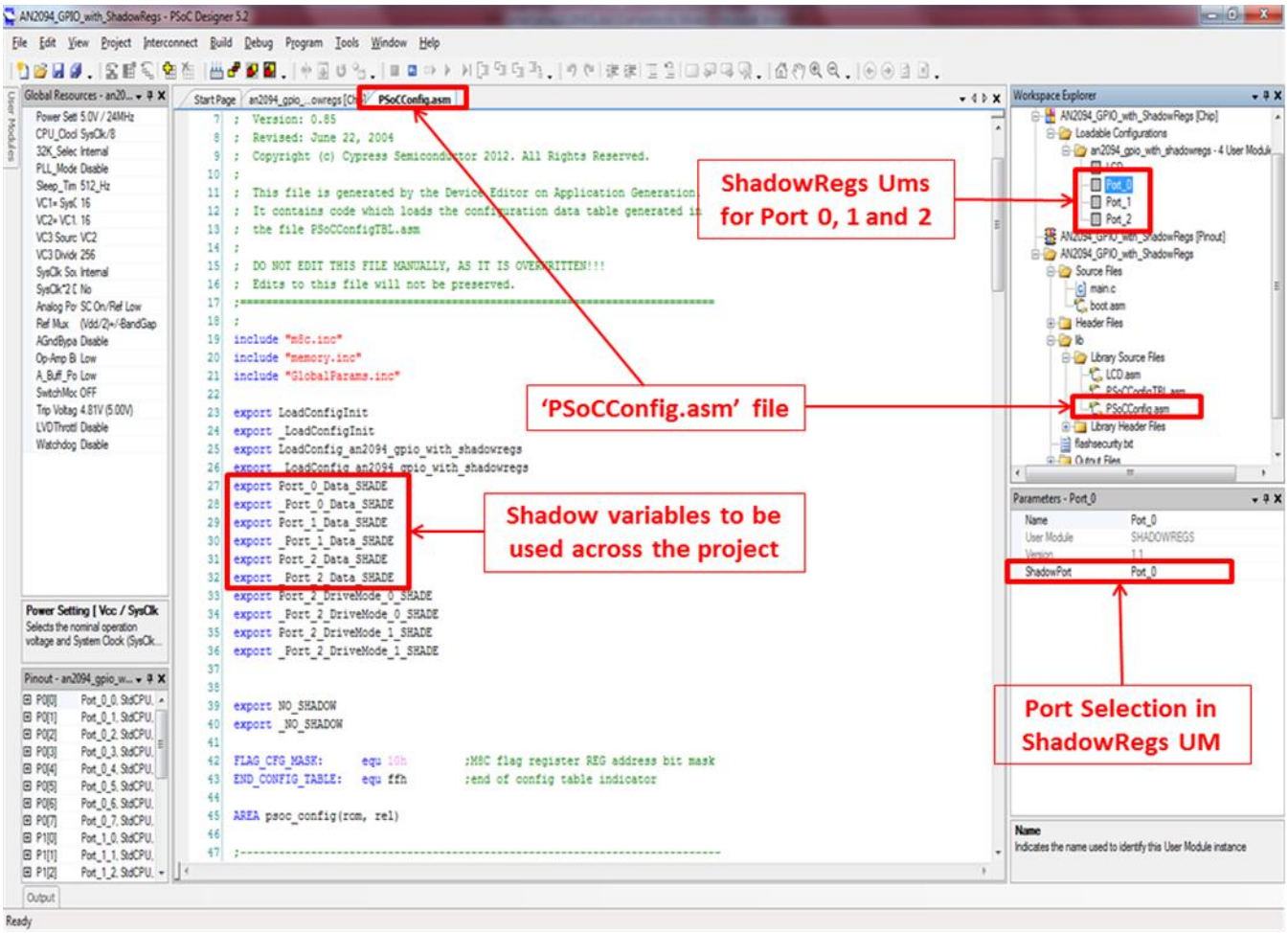
1. 读取 – Port_0_Data_Shade = x x x x x x 0 0 （位 0 = 0 → LED 关闭；根据初始设置，位 1 = 0，该数据并非直接来自该端口）
2. 修改 – (Port_0_Data_Shade | 00000001) = x x x x x x 0 1
3. 写入 – Port_0_Data_Shade = x x x x x x 0 1
4. 写入到端口 – PRT0DR = Port_0_Data_Shade

PSoC Designer 在其它数字用户模块类别中提供了一个用户模块（UM），即 ShadowRegs（映像寄存器）。如果在项目中放置了该用户模块，并且分配了一个端口，那么将在 *Library source files* 目录下的 *PSoCConfig.asm* 文件内（请参见图 7）生成一个变量（即 Port_x_Data_SHADE），其中‘x’是端口号。现在，通过使用‘extern BYTE Port_x_Data_SHADEN’，可以将该变量导入到各个文件内。

运行 GPIO 引脚的用户模块（如 TX8SW）也使用映像寄存器，因此，当其他用户模块使用已被组合的输入引脚和输出引脚时，也不会造成任何影响。

当您重命名某个引脚并在端口中放置一个映像寄存器时，*psocgpioint.h* 和 *psocgpioint.inc* 文件中将为该映像寄存器创建一个带有引脚名称的别名。请参见命名一个引脚部分。

图 7. PSoC Designer 中的映像变量位置



示例 2 中介绍了 GPIO 驱动模式的修改。您可以从相应应用笔记的网页上下载该示例。有关该示例项目的详细信息，请参考项目 2：映像寄存器的使用情况。

6 GPIO 中断

中断是 GPIO 系统的另一个重要部分，尤其是当需要优先处理数字信号时。有关 PSoC 中的中断系统的内容非常多。欲了解 PSoC 中所有可用中断及其优先级的信息，请参见相应的器件 TRM（技术参考手册）。本节仅介绍了有关 GPIO 中断的信息。

可以配置 PSoC 中的每个 GPIO 引脚，使之在发生下述事件时生成中断：上升沿、下降沿或前一次读取事件后发生了变化。 可以通过两种方法配置触发 GPIO 中断的事件：第一方法是在 GPIO 配置窗口中配置中断（请参见器件编辑器的配置）；第二个方法是在固件中配置中断。表 3 中列出了配置和启用 GPIO 中断时所涉及的寄存器。

表 3. GPIO 中断配置的相关寄存器

相关寄存器	说明	值
PRTxIE	每个端口‘x’的中断使能寄存器。设置或清除该寄存器中的某一位时，可以通过启用或禁用特定引脚上相应的中断实现。	1 — 使能 0 — 禁用中断
PRTxIC0 和 PRTxIC1	中断控制寄存器 — 用于设置触发中断的事件类型（上升沿、下降沿或读取值）的变化。	表 4
INT_MSK0	中断使能屏蔽寄存器 0	该寄存器的位 5 是全局 GPIO 中断使能/禁用位。INT_MSK0_GPIO 宏可作为一个屏蔽用于使能或禁用该寄存器中的第 5 位。
INT_CLR0	已生成的中断读取和清除寄存器 0	该寄存器中的位 5 是已生成的 GPIO 中断位。通过向该位写入 0，可以清除任何一个所生成的 GPIO 中断。如果读取该位时，返回值为 1，则表示已经生成了一个 GPIO 中断。通过软件向该位写入 1，可以生成一个 GPIO 中断[4]。

表 4. 中断控制寄存器设置

PRTxIC1 [n]	PRTxIC0 [n]	中断类型	说明
0	0	禁用	中断被禁用
0	1	下降沿	输入信号从 1 变为 0 时将发生中断
1	0	上升沿	输入信号从 0 变为 1 时将发生中断
1	1	读取值的变化	与最后读取 PRTxDR[n]上的值所代表的状态相比，引脚当前的状态发生了变化。

在表 4 中，‘x’表示端口号，‘n’表示要配置的端口寄存器/引脚中的位。

要想实现某个 GPIO 中断，请执行以下操作：

1 在 GPIO 配置窗口中，设置引脚的中断类型。也可以通过代码（即对 PRTxICx 寄存器进行写操作）实现该设置。

2 在主应用代码中，通过设置 INT_MSK0 寄存器中的位 5，可以使能 GPIO 中断。请使用 M8C_EnableIntMask 宏实现该操作。

3 通过对 PRTxIE 寄存器进行写操作来使能相应引脚的中断。如果在 GPIO 配置窗口中选择了中断类型，那么，PSoC Designer 将在启动过程中自动设置 PRTxIE 寄存器中的位。

[4] 如果ENSWINT位为0，向该寄存器写入0，将清除所生成的中断（若有）。
如果当时ENSWINT位为0，即便向该寄存器写入1，也不会产生任何影响。
如果当时ENSWINT位为1，即便向该寄存器写入0，也不会产生任何影响。
如果当时 ENSWINT 位为 1，向该寄存器写入 1，将为通用输入和输出（引脚）生成一个中断。

ENSWINT 位是 INT_MSK3 寄存器中的第 7 位。

4 为处理中断，请使用 C 语言或汇编语言编写一个中断服务子程序（ISR）。如果使用 C 语言编写 ISR，那么请使用 #pragma interrupta_handler 指令声明该 ISR，以通知编译器该函数是一个 ISR。

5 通过放置代码，将中断重定向到 ISR 函数。可以通过下面两个方法实现该操作：

a. 在 GPIO 配置窗口中启用中断时，PSoC Designer 将生成一个库文件，即 *psocgpioint.asm*。该文件包含了一个占位符函数。可以将中断重定向到该函数。

例如，如果您有一个被称为 MyGpioIsr 的 C 函数，那么请使用代码“LJMP_MyGpioIsr”（从汇编程序中调用某个 C 函数时，必须在该函数名称中添加一条下划线）。

b. 将重定向指令直接添加到启动代码内。为实现该操作，请打开项目文件夹中的 *boot.tpl* 文件。即 PSoC Designer 生成 *boot.asm* 文件时所使用的模板文件。在 *boot.tpl* 文件中的 GPIO 中断向量内，注释代码“`@INTERRUPT_7`”并将定向指令添加到 GPIO ISR 内。保存 *boot.tpl* 文件并生成应用。现在，在 *boot.asm* 文件中，已经将重定向指令添加到 GPIO ISR 内。

请注意，各个 GPIO 中断只有一个相关联的中断向量。如果在多个引脚上启用中断，则应用代码会检测引起发生中断的引脚，并处理该中断。

在下面显示的示例代码中，P0_0 和 P0_1 引脚上的中断均被使能。P0_0 的中断被配置为下降沿中断，而 P0_1 的中断被配置为‘读取值发生变化’中断。通过使用 M8C_EnableIntMask 宏使能 GPIO 中断，然后可以使能全局中断。在 GPIO_ISR 中，将放置‘if’控制结构，以检查引发该 ISR 实例的引脚，并对数据进行相应处理。

设置和启用 GPIO 中断

```
/* P0_0 configured as falling edge interrupt */
PRT0IC0 |= 0x01;
PRT0IC1 &= ~0x01;

/* P0_1 configured as Change from Read interrupt */
PRT0IC0 |= 0x02;
PRT0IC1 |= 0x02;

/* Enable P0_1 and P0_0 interrupts */
PRT0IE |= 0x03;

/* Enable GPIO interrupts */
M8C_EnableIntMask(INT_MSK0, INT_MSK0_GPIO);

/* Enable Global interrupts */
M8C_EnableGInt;
```

对 GPIO ISR 进行写操作

```
/* Function prototype for GPIO_ISR*/
#pragma interrupt_handler GPIO_ISR

/* GPIO ISR in C where GPIO interrupts are processed */
void GPIO_ISR(void)
{
/* variable to have a copy of prev P0_1 value for change from read comparison
*/
static BYTE port0_prevValue;

/* Check if interrupt because of P0_0 falling edge:
First condition checks for P0_0 to be '0'
Second condition checks if it was '1' in the last ISR */
```

```
if(((PRT0DR & 0x01) == 0) && ((PRT0DR & 0x01) == 0x01))
{
/* Process P0_0 interrupt */
}

/* Check if interrupt because of P0_1 change from read */
if ((PRT0DR ^ port0_prevValue)==0x02)
{
/* Process P0_1 interrupt */
}

/* Store values of P0_0 and P0_1 for next ISR */
port0_prevValue = PRT0DR & 0x03;
}
```

6.1.1 设置 ISR 的重定向

在 *psocgpioint.asm* 文件里面：

```
;-----------------------------------------
;  FUNCTION NAME: PSoC_GPIO_ISR
;
;  DESCRIPTION: Unless modified, this implements only a null handler stub.
;
;----------------------------------------- PSoC_GPIO_ISR:

;@PSoC_UserCode_BODY@ (Do not change this line.)
;---------------------------------------
; Insert your custom code below this banner
;---------------------------------------
   ljmp _GPIO_ISR
;---------------------------------------
; Insert your custom code above this banner
;---------------------------------------
;@PSoC_UserCode_END@ (Do not change this line.)
   reti
```

同样，下面的代码显示了汇编语言中相同的实现情况

在 *boot.tpl* 文件中：

```
org   1Ch     ;GPIO Interrupt Vector
;`@INTERRUPT_7`
ljmp _GPIO_ISR
      reti
```

进行这些更改后，请保存 *boot.tpl* 文件并生成应用。

6.1.2 设置和启用 GPIO 中断

```
; P0_1 configured as change from read
; Interrupt

M8C_SetBank1
or reg[PRT0IC0], 0x02
or reg[PRT0IC1], 0x02

; Enable P0_1 interrupt
M8C_SetBank0
or reg[PRT0IE], 0x02

;Enable GPIO interrupts
M8C_EnableIntMask INT_MSK0, INT_MSK0_GPIO

;Enable Global interrupts
M8C_EnableGInt
```

代码 1: GPIO ISR

```
export MyGpioIsr

area text
;GPIO ISR in ASM where all GPIO ISRs ;are processed
GPIO_ISR:

;Preserve CUR_PP, X and A
push A
push X
mov A, reg[CUR_PP]
push A

;Change CUR_PP to port0PrevValue's Ram ;page (port0PrevValue can be defined ;in any
'.c' file included in the ;project as a global BYTE variable)
RAM_SETPAGE_CUR >_port0PrevValue

;Read PRT0DR into A
mov A, reg[PRT0DR]

;take a copy of PRT0DR into X for ;storing in port0PrevValue at the end
mov X, A

;XOR PRT0DR value in A and PRT0DR's ;prev value
xor A, [<_port0PrevValue]

;AND with 0x02 to read P0_1's state
and A, 0x02

;If zero flag is set, no change in ;state, so not P0_1 ISR
jz .NoP0_1_ISR
;
;Comes here if the XOR operation ;resulted a non zero result
;
;Process code for P0_1 ISR
;
.NoP0_1_ISR:
;
;Process code for other GPIO ISRs
```

```
;
RAM_SETPAGE_CUR >_port0PrevValue
mov [<_port0PrevValue], X

;Restore CUR_PP, X and A in order they ;were pushed
pop A
mov reg[CUR_PP], A
pop X
pop A
reti
```

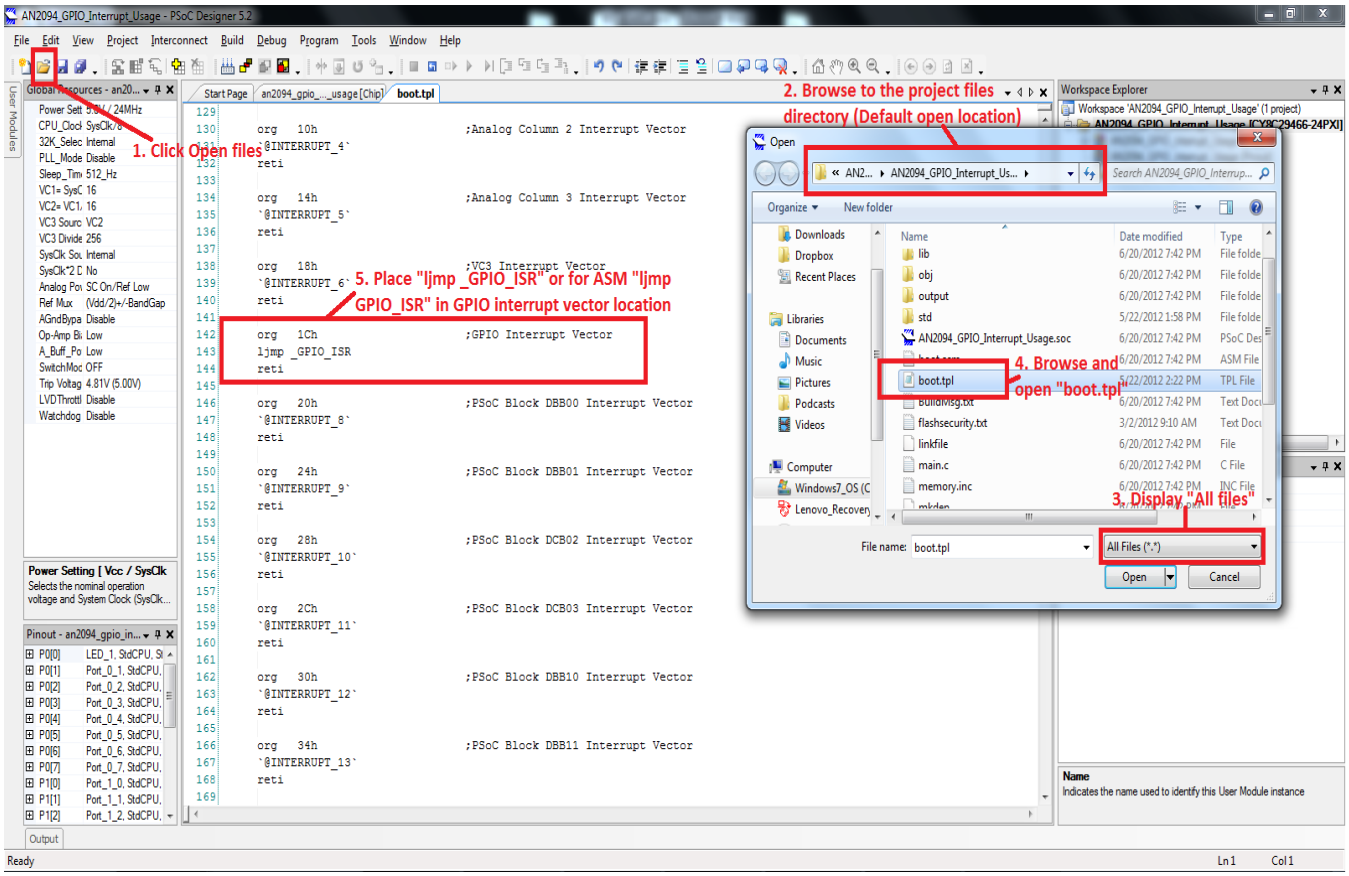
ISR 的重定向方法与在 C 语言示例中所使用的方法相同。在 *psocgpioint.asm* 文件或 *boot.tpl* 文件中所放置的 ljmp 指令内，不需在 MyGpioIsr 前面使用下划线。

示例#3 展示了实现 GPIO 中断的情况。您可以从相应的应用笔记网页上下载该示例。有关该示例项目的详细信息，请参考项目 3：使用中断切换 LED。

6.2 使用中断时该进行和不应进行的操作

- 必须通过使用 M8C_EnableGInt 宏设置全局中断使能位。
- 不应该将 GPIO 的驱动模式设置为高阻抗模拟，否则该引脚永远不会生成中断。
- 当 GPIO 中断被配置为“读取值发生变化”时，应通过 PRTxDR 寄存器读取该引脚上的值，以便触发下一个中断。仅在最后一次读取 PRTxDR 的值不同于当前的引脚状态时，才会触发该中断。
- 应该使用“#pragma interrupt_handler”指令定义 C 语言程序中的 ISR 函数，这样，该函数才能正常执行并返回控制。
- *asm* 文件中定义的 ISR 函数应保留所使用的寄存器，并在退出前恢复它们。该函数应使用 RETI 指令返回，而不是普通的 RET 指令。
- 可将在 C 语言程序或 ASM 中所定义的 ISR 函数重定向放置在 *psocgpioint.asm* 文件或 *boot.tpl* 文件内。图 8 显示了 *boot.tpl* 文件和 GPIO IRS 的位置。

图 8. *boot.tpl* 文件中 GPIO ISR 的位置



7 其他 GPIO 资源和提示

7.1 GPIO 全局选择寄存器

PRTxGS 寄存器将确定 GPIO 引脚受 CPU 的控制还是被连接至全局输入总线或全局输出总线。当 PRTxGS 寄存器中与该引脚相对应的位被清除时，通过对 PRTxGS 寄存器进行写操作，CPU 可以控制该引脚。当 PRTxGS 寄存器中的位被设置时，该引脚可以与全局总线（全局输入和全局输出）相连，也可以直接连接至某个数字模块的输入或输出。

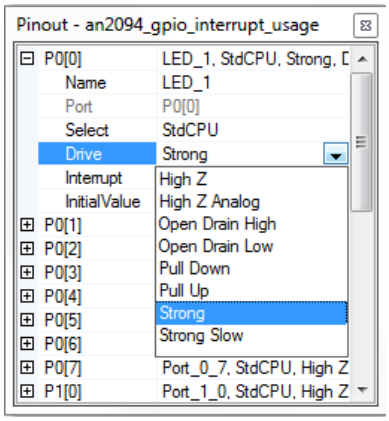
7.2 模拟复用器（AMUX）总线控制寄存器

模拟复用器（AMUX）总线控制寄存器（MUX_CRx）[5]可以启用或禁用 GPIO 引脚与内部模拟复用器总线间的连接。这样，该模拟复用器总线可作为 PSoC 器件中各种模拟模块的输入使用。例如，通过设置 MUX_CR1 的位 0，可以将 P1_0 连接至 AMuX 总线。更多有关 MUX 设置和路由的详细信息，请参考相应器件的 TRM。

7.3 命名一个引脚

在 GPIO 配置窗口中，可给每个引脚分配唯一一个有意义的名称，如图 9 所示。

图 9. 命名一个引脚



当命名完某个引脚后，为使访问操作方便，PSoC Designer 分别在 C 语言程序的 *PSoCGPIOINT.h* 文件和 ASM 语言的 *PSoCGPIOINT.asm* 文件中生成与该引脚相应的所有寄存器的宏。宏列表包括：

- 端口数据寄存器（PRTxDR）
- 端口驱动模式寄存器（PRTxDMx）
- 端口中断使能寄存器（PRTxIE）
- 端口中断设这寄存器（PRTxICx）
- 端口全局选择寄存器（PRTxGS）
- 引脚屏蔽
- 映像寄存器

表 5 提供了 *PSoCGPIOINT.h* 文件（用于 C 文件）和 *PSoCGPIOINT.inc* 文件（ASM 文件）中所生成的各个宏的概述。可以在所有函数中直接使用这些宏，以访问引脚的相关设置和信息。

[5] 仅在下面器件系列中可用：CY8C21x34、21x45、22x45、 24x94 和 28xxx。

表 5. 与一个已命名引脚相关联的各个宏

引脚名称	LED_1
端口数据寄存器	LED_1_Data_ADDR
端口驱动模式 0 寄存器	LED_1_DriveMode_0_ADDR
端口驱动模式 1 寄存器	LED_1_DriveMode_1_ADDR
端口驱动模式 2 寄存器	LED_1_DriveMode_2_ADDR
端口全局选择寄存器	LED_1_GlobalSelect_ADDR
端口中断使能寄存器	LED_1_IntEn_ADDR
端口中断控制 0 寄存器	LED_1_IntCtrl_0_ADDR
端口中断控制 1 寄存器	LED_1_IntCtrl_1_ADDR
引脚屏蔽	LED_1_MASK
映像寄存器	LED_1_DataShadow

使用引脚名称和引脚宏的好处是：如果将引脚移到其他端口，则 PSoC Designer 会自动更新与该引脚宏相关联的各个寄存器，因此不用修改应用代码。

下面的代码段显示的是这些宏的用法：

通过使用数据寄存器直接写入 LED_1 引脚：

```
/* Write 1 to LED_1 pin */
LED_1_Data_ADDR |= LED_1_MASK;

/* Write 0 to LED_1 pin */
LED_1_Data_ADDR &= ~LED_1_MASK;
```

通过使用映像寄存器写入 LED_1 引脚：

```
/* Write 1 to LED_1 */
LED_1_DataShadow |= LED_1_MASK;
LED_1_Data_ADDR = LED_1_DataShadow;

/* Write 0 to LED_1 */
LED_1_DataShadow &= ~LED_1_MASK;
LED_1_Data_ADDR = LED_1_DataShadow;
```

将 LED_1 的驱动模式更改为强驱动模式：

```
/* Set LED_1 drive mode to strong */
LED_1_DriveMode_0_ADDR |= LED_1_MASK;
LED_1_DriveMode_1_ADDR &= ~LED_1_MASK;
LED_1_DriveMode_2_ADDR &= ~LED_1_MASK;
```

将 LED_1 与全局总线连接起来，或断开它们间的连接：

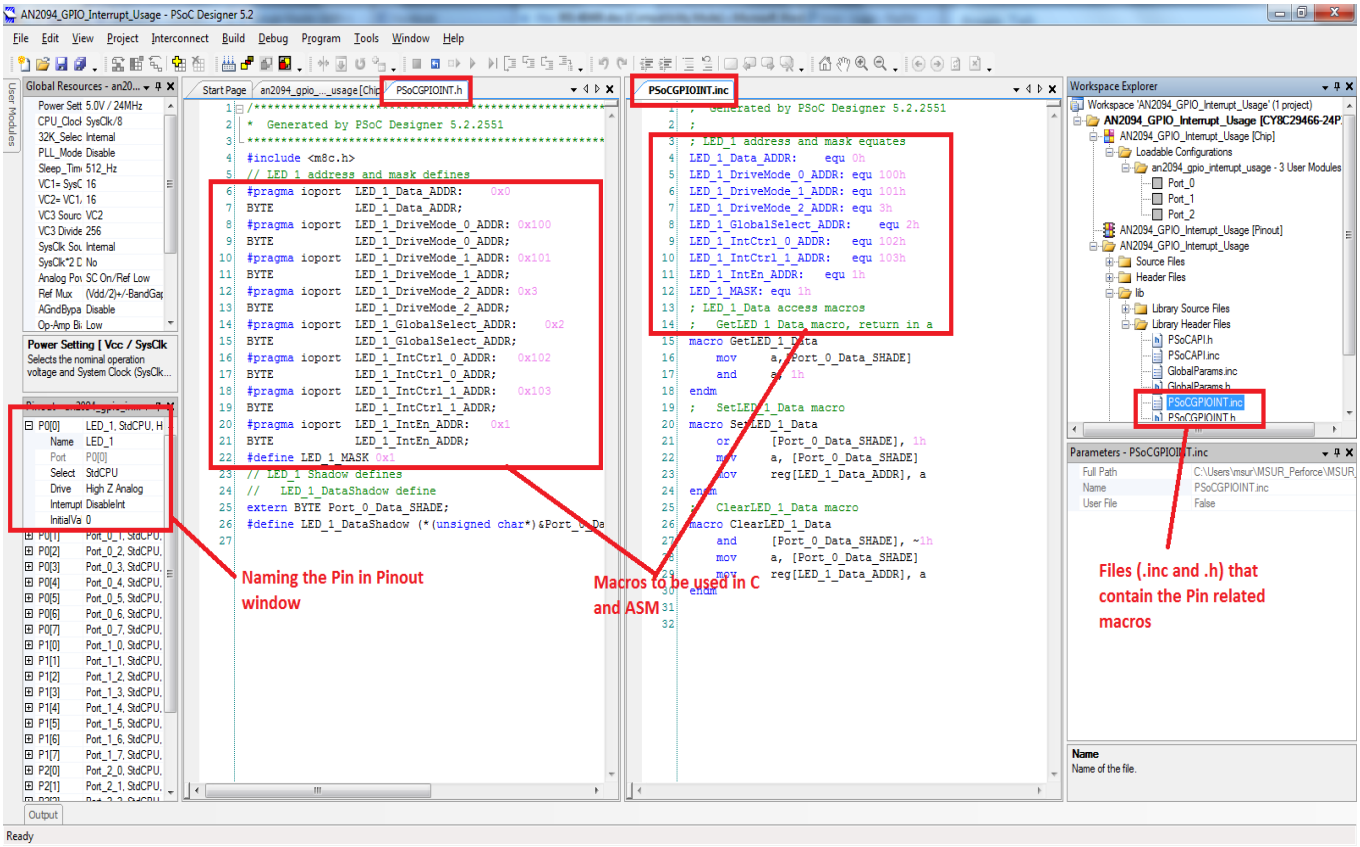
```
/* Connect LED_1 to global bus */
LED_1_GlobalSelect_ADDR |= LED_1_MASK;

/* Disconnect LED_1 from global bus */
LED_1_GlobalSelect_ADDR &= ~LED_1_MASK;
```

读取 SW 引脚，并将读取到的内容写入 LED_1 引脚：

```
if (SW_Data_ADDR & SW_MASK)
{
  /* Write 1 to LED_1 */
  LED_1_DataShadow |= LED_1_MASK;
  LED_1_Data_ADDR = LED_1_DataShadow;
}
else
{
  /* Write 0 to LED_1 */
  LED_1_DataShadow &= ~LED_1_MASK;
  LED_1_Data_ADDR = LED_1_DataShadow;
}
```

图 10 显示了 *psocgpiont.h* 和 *psocgpioint.inc* 文件中的各个引脚宏。

图 10. 相关引脚宏



7.4 寄存器及其相关寄存器组

PSoC 1 寄存器映射图中包含两个寄存器组。每个端口配置寄存器均为这些寄存器组中的某一个。要想在汇编程序中访问某个寄存器，需要清楚该寄存器所在的组。在 C 语言代码中，编译器将自动处理各组间的切换。

在 ASM 语言中，如果要将某个寄存器组更改为组 0，请使用 M8C_SetBank0 宏。同样，通过使用 M8C_SetBank1 宏，可以将该寄存器组更改为组 1。表 6 介绍的是本应用笔记中每个 GPIO 寄存器所在寄存器组的详细信息。

表 6. GPIO 的相关寄存器及其寄存器组

寄存器	寄存器组
PRTxDR	0
PRTxDM0	1
PRTxDM1	1
PRTxDM2	0
PRTxIE	0
PRTxGS	0
PRTxIC0	1
PRTxIC1	1
INT_MSK0	0
INT_CLR0	0
MUX_CRx	1

8 示例项目

8.1 项目 1：检测 LED 驱动模式

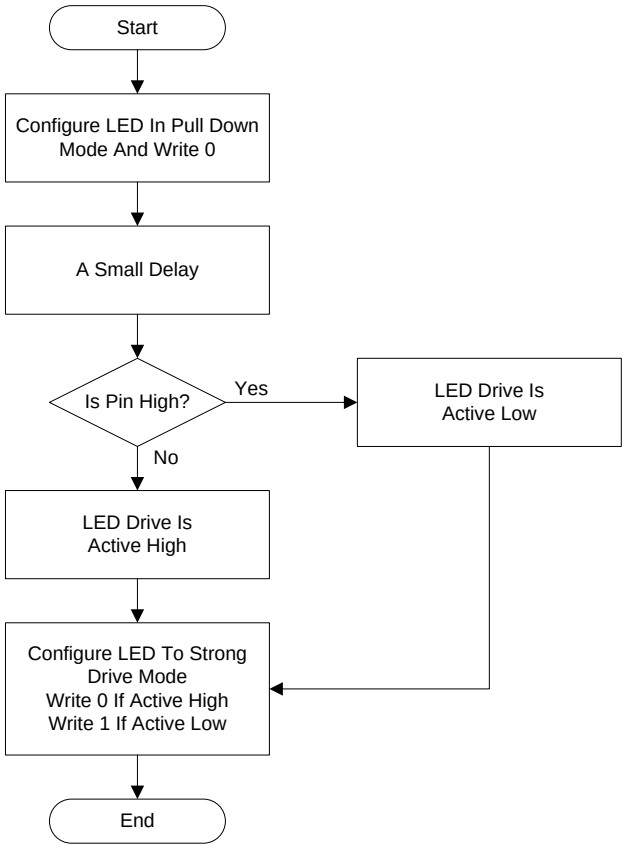
本示例项目介绍的是如何通过使用 PRTxDMx 寄存器随时重新配置某个引脚的驱动模式。

在大多数系统中，通常通过 GPIO 给 LED 提供灌电流，以便打开它们。在某些系统中，则通过 GPIO 给 LED 提供拉电流，以便打开它们。虽然在设计系统时已经明确了 LED 的实现，但您可能只要升级或更新该设计，而不想改变固件。

例如，如果你在设计中添加了一个用于为 LED 提供拉电流的 GPIO，但该 GPIO 不能提供足够的电流；或者，如果你使用一个具有更高额定电流的 LED，并将设计更改为 LED 灌电流模式，但你需要器件根据 LED 与 GPIO 间的连接模式本身进行调整。通过该示例，您可以将上述性能添加到您的设计中，即，您可以识别某个 LED 与 GPIO 引脚间的连接模式，然后相应地打开或关闭该 LED。

图 11 中的流程图说明的是如何内部检测 LED 驱动模式。

图 11. 检测 LED 驱动模式算法



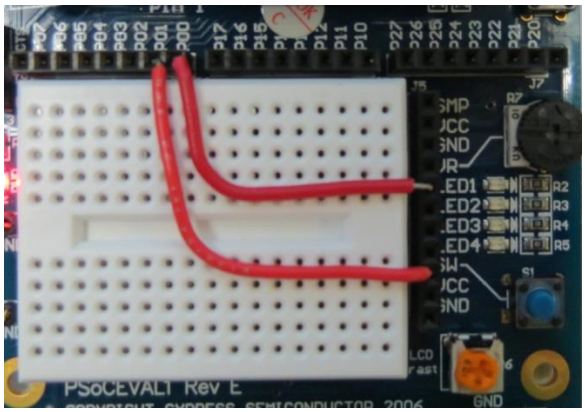
要想实现示例项目，需要使用以下硬件：

- 具有 28 引脚 CY8C29466-24PXI PDIP PSoC1 器件的 CY3210 PsoCEval1 电路板。
- 一个备用 LED 和一个 1 kΩ 的电阻，用于检查 LED 的灌电流模式
- CY3217 MiniProg1 或 CY8CKIT-002 MiniProg3 器件。
- 连接线

要想测试示例**项目**，请遵循下列步骤：

1. 将 28 引脚 CY8C29466-24PXI 器件插入 CY3210 电路上的 28 引脚 PDIP 插座内。
2. 将 MiniProg1 或 MiniProg3 连接至 CY3210 电路板上的编程插座（J11）。
3. 打开 PSoC Programmer 3.23.1 或更高版本，并连接 MiniProg1 或 MiniProg3。
4. 浏览到本应用笔记附带的 AN2094_GPIO_DM_Reconfig 项目中根目录下的 *AN2094_GPIO_DM_Reconfig.hex* 文件，即依次打开文件夹 *AN2094 > AN2094_GPIO_DM_Reconfig > AN2094_GPIO_DM_Reconfig.hex*。
5. 通过按下 PSoC Programmer 的编程按键，可以使用所选的文件编程器件。欲了解更多有关编程选项的详细信息，请参考 AN2015 — PSoC 1 读取和写入闪存和 E2PROM。
6. 将 P0_1 连接至 SW，并将 P0_0 连接至 LED1， 如图 12 所示。

图 12. 示例 1 的线连接

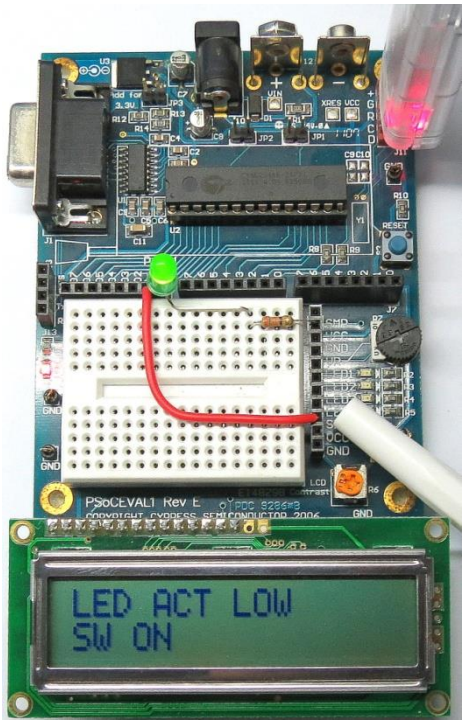


7. 将 CY3210 PSoCEval1 套件附带的字符 LCD 连接至 CY3210 电路板上的 J9 插头。
8. 将 CY3210 套件中的 JP3 拔出，以实现 5 V 操作。
9. 通过一个 MiniProg1 or MiniProg3 或 5 V 的直流适配器给电路板供电。
10. LCD 行 0 应该显示“LED ACT HI”，并且当按下 SW 时，LED1 将被点亮，并且 LCD 行 1 将显示“SW ON”，如图 13 所示。
11. 同样，如果您使用导线将一个 1 kΩ 电阻（一端连接到 LED，另一端连接到 Vcc）连接至一个灌电流模式下的备用 LED，并将其连接到 P0_0，那么，LCD 将显示“LED ACT LOW”，同时它会根据 SW ON/OFF 的情况打开/关闭 LED（LED ON/OFF），如图 14 所示。

图 13. 示例 1：输出 — LED 为高电平有效



图 14. 示例 1：输出 — LED 为低电平有效



8.2 项目 2：映像寄存器的使用情况

本文档已经通过使用 CY3210-PSoCEval1 电路板创建了一个简单的设置，以说明映像寄存器的重要性和用途，如图 15 所示。

在该示例中，通过硬件设置，可以在上电时启用或禁用映像寄存器功能。在上电过程中，如果将 P0_2 连接至 V_{DD}，映像寄存器将被禁用；如果 P0_2 被连接至 GND，则可以启用映像寄存器。该示例将描述映像寄存器的使用情况一节介绍的情况，其中某个端口同时有一个输入开关和一个输出 LED。

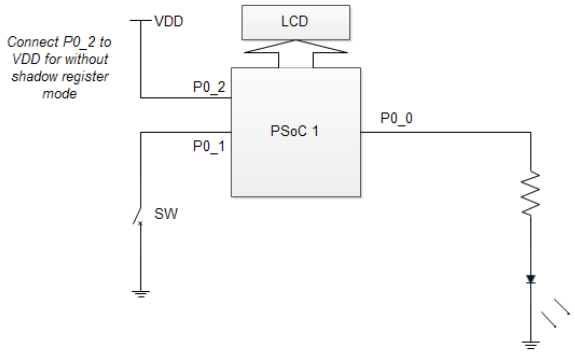
8.2.1 硬件要求

- 具有 28 引脚 CY8C29466-24PXI PDIP PSoC 1 器件的 CY3210 PsoCEval1 电路板
- CY3217 MiniProg1 器件
- 连接线

8.2.2 测试流程

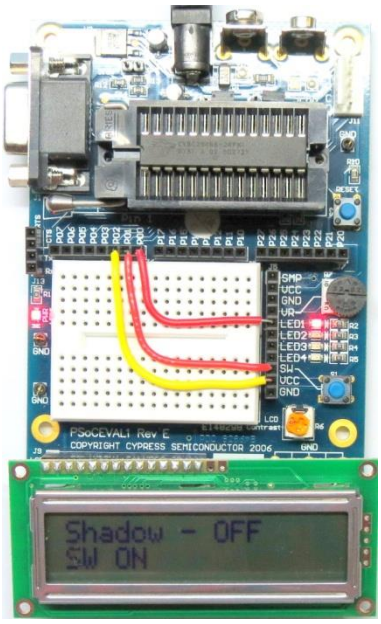
1. 将 28 引脚 CY8C29466-24PXI 器件插入到 CY3210 电路上的 28 引脚 PDIP 插座内。
2. 将 MiniProg1 或 MiniProg3 连接到 CY3210 电路板上的编程插座（J11）。
3. 打开 PSoC Programmer 3.23.1 或更高版本，并连接 MiniProg1 或 MiniProg3。
4. 浏览到本应用笔记附带的 AN2094_GPIO_with_ShadowRegs 项目中根目录下的 *AN2094_GPIO_with_ShadowRegs.hex* 文件（即依次打开 *AN2094>AN2094_GPIO_with_ShadowRegs > AN2094_GPIO_with_ShadowRegs.hex*）。
5. 通过按下 PSoC Programmer 的编程按键，可以使用所选的文件来编程器件。欲了解更多有关编程选项的详细信息，请参考 AN2015 — PSoC 1 读取和写入闪存和 E2PROM。
6. 将 CY3210 PSoCEval1 套件附带的字符 LCD 连接至 CY3210 电路板上的 J9 插头。
7. 为测试该项目而不使用映像变量，请将 P0_0 连接至 LED1，将 P0_1 连接至 SW，并将 P0_2 连接至 V_{DD}，如图 15 所示。

图 15. 示例 2 中没有使用映像寄存器的引脚连接



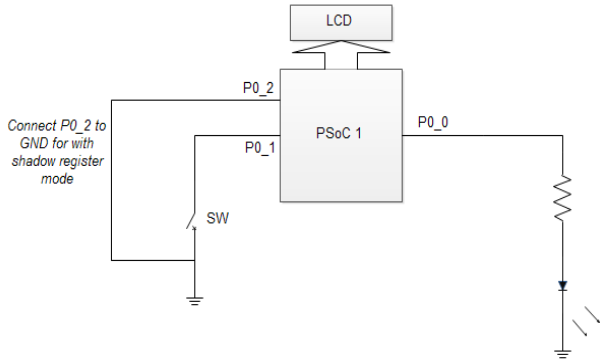
8. 将 CY3210 套件中的 JP3 拔出，以实现 5 V 操作。
9. 使用 MiniProg1、MiniProg3 或 5 V 直流适配器为电路板供电。
10. 现在，按下一次 SW 然后释放，你会观察到 LCD 显示屏的第一行始终显示“SW ON”，并且 LED1 会被打开。这是因为没有使用任何映像变量。

图 16. 示例 2：没有映像寄存器时的输出



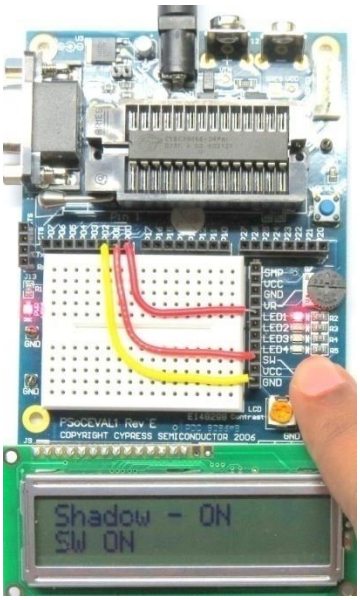
11. 现在，使电路板断电，并将 P0_2 连接至 GND，如图 17 所示。

图 17. 示例 2 中使用映像寄存器的引脚连接



12. 给电路板供电。
13. LCD 行 0 将显示“Shadow – ON”。现在，按下开关会显示“SW ON”，并且 LED 被点亮。释放开关将显示“SW OFF”，并且 LED 将被关闭，如图 18 所示。

图 18. 示例 2：使用影像寄存器时的输出



8.3 项目 3：使用中断切换 LED

为了说明 GPIO 中断的用途，已在本示例中执行了一个简单的 LED 切换算法。在与某个开关相连的引脚上启用上升沿中断。在 ISR 中，已经设置了相应的标志，用以指示该开关上的上升沿。经过一个 2 ms 的去抖动后，将在当型循环中切换该 LED。在 LCD 上显示 LED ON/OFF 状态。

8.3.1 硬件要求

- 具有 28 引脚 CY8C29466-24PXI PDIP PSoC1 器件的 CY3210 PsoCEval1 电路板。
- CY3217 MiniProg1 器件
- 连接线

8.3.2 测试流程

1. 将 28 引脚 CY8C29466-24PXI 器件插入 CY3210 电路上的 28 引脚 PDIP 插座内。
2. 将 MiniProg1 或 MiniProg3 连接至 CY3210 电路板上的编程插座（J11）。
3. 打开 PSoC Programmer 3.23.1 或更高版本，并连接 MiniProg1 或 MiniProg3。
4. 浏览到本应用笔记附带的 AN2094_GPIO_with_ShadowRegs 项目中根目录下的 *AN2094_GPIO_Interrupt_Usage.hex* 文件，即依次打开 *AN2094> AN2094_GPIO_Interrupt_Usage > AN2094_GPIO_Interrupt_Usage.hex*。
5. 通过按下 PSoC Programmer 的编程按键，可以使用所选的文件来编程器件。欲了解更多有关编程选项的信息，请参考 AN2015 — PSoC 1 读取和写入闪存和 E2PROM。
6. 将 CY3210 PSoCEval1 套件附带的字符 LCD 连接至 CY3210 电路板上的 J9 插头。
7. 为了测试该项目，请连接 P0_0 至 LED1 并连接 P0_1 至 SW，如图 19 所示。
8. 将 CY3210 套件中的 JP3 拔出，以实现 5 V 的操作。
9. 使用 MiniProg1 或 5 V 直流适配器给电路板供电。
10. 按下 SW，并观察 LED1 在每次 SW 被按下时的切换。
11. LCD 行 0 显示“GPIO ISR proj”，并且行 1 显示 LED ON/OFF 的状态，如图 19 和图 20 所示。

图 19. 示例 3：LED ON 的输出



图 20. 示例 3：LED OFF 的输出



8.4 额外的代码示例

PSoC 1 器件的更多代码示例在 PSoC Designer 5.4 SP1 或更高版本中有效。要打开它们，请依次点击：**Start Page > Design Catalog > Launch Example Browser**。

文档修订记录

文档标题：AN2094 — PSoC® 1 – GPIO 入门

文档编号：001-93056

版本	ECN	变更者	提交日期	变更说明
**	4503263	ROWA	10/28/2014	本文档版本号为 Rev**，译自英文版 001-40480 Rev*H。
*A	4669790	ROWA	03/25/2015	本文档版本号为 Rev*A，译自英文版 001-40480 Rev*I。
*B	5045445	ROWA	12/15/2015	本文档版本号为 Rev*B，译自英文版 001-40480 Rev*J。
*C	6071922	SSAS	02/15/2018	本文档版本号为 Rev. *C，译自英文版 001-40480 Rev. *K。

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 MCU

赛普拉斯开发者社区

社区 | 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

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

© 赛普拉斯半导体公司，2007-2018 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可权）（1）在赛普拉斯特软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。没有任何电子设备是绝对安全的。因此，尽管赛普拉斯在其硬件和软件产品中采取了必要的安全措施，但是赛普拉斯并不承担任何由于使用赛普拉斯产品而引起的安全问题及安全漏洞的责任，例如未经授权的访问或使用赛普拉斯产品。此外，本材料中所介绍的赛普拉斯产品有可能存在设计缺陷或设计错误，从而导致产品的性能与公布的规 格不一致。（如果发现此类问题，赛普拉斯会提供勘误表）赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。