

PSoC® 4 中断

作者: Rajiv Badiger

相关项目: 代码示例

相关器件系列: 所有 PSoC 4 器件

软件版本: PSoC Creator™ 4.2. 或更高版本

如果需要本应用笔记的最新版本, 请访问 <http://www.cypress.com/AN90799>。

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。

AN90799 介绍了 PSoC 4 中的中断架构以及它在 PSoC Creator™ 中的配置。本文档作为开发基于中断的项目的向导。本应用笔记还说明了中断相关的高级概念, 比如: 延迟、向量选择、中断代码优化以及调试技术。

目录

1 简介	1	7 高级中断主题	15
2 PSoC 中断架构	2	7.1 异常事件	15
2.1 中断源	2	7.2 中断延迟	15
2.2 电平及边沿触发中断	4	7.3 优化中断代码	16
3 PSoC Creator 的中断支持	5	7.4 带有内置中断的组件	17
3.1 中断组件配置	5	7.5 强制中断向量编号	17
3.2 中断优先级配置	7	7.6 SysTick 定时器	18
3.3 中断 API 函数	8	7.7 中断嵌套	19
4 写入一个中断服务子程序 (ISR)	9	7.8 GlobalSignal 组件的使用	19
4.1 使用自动生成的 ISR	9	7.9 易失性全局变量的使用	20
4.2 使用回调函数	11	8 总结	20
4.3 创建自定义 ISR	11	附录 A. PSoC 4 中的中断源和向量编码	22
5 代码示例	13	全球销售和设计支持	26
6 调试提示	13		

1 简介

中断是嵌入式应用的重要组成部分。它们使 CPU 不必连续轮询某个特定事件的发生情况。它仅在该事件发生时通知 CPU。在片上系统 (SoC) 的架构 (如 PSoC) 中, 经常使用中断向 CPU 报告片上外设的状态。

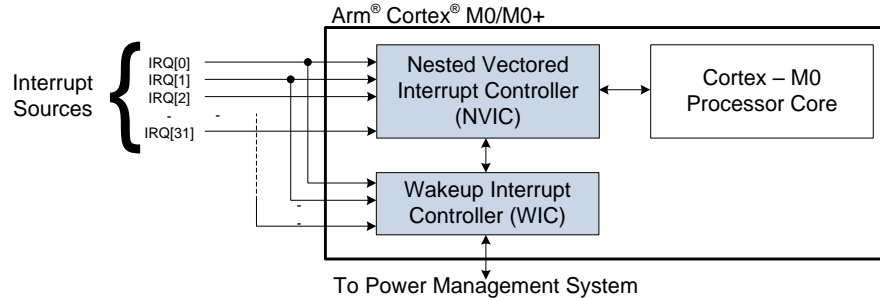
该文件先对 PSoC 4 中断结构进行了介绍。如果您想了解 PSoC Creator IDE 的中断支持的内容, 请跳到 [PSoC Creator 的中断支持](#) 一节。对于采样代码示例, 请参考 [代码示例](#)。如果您正在调试中断项目, 请转到 [调试提示](#) 部分, 该部分提供了一些用于发现和解决中断问题的提示。[高级中断主题](#) 部分提供了与中断相关的高级主题内容。

本应用笔记假定您已经熟悉了 PSoC 器件和 PSoC Creator 集成开发环境 (IDE)。如果您尚未了解 PSoC, 可通过 [AN79953 PSoC 入门应用笔记](#) 并参考 [PSoC Creator 主页](#) 来获取相关信息。

2 PSoC 中断架构

图 1 显示了 PSoC 4 的中断架构的简化框图：

图 1. PSoC 4 中断架构



一共 32 个中断线 – 从 IRQ[0] 到 IRQ[31] – 每线有四个优先级：从 0 到 3。每个中断线被分配了一个中断向量地址。接收到一个中断请求后，CPU 会跳转到该地址，并会在这里执行一个特殊函数，即中断服务子程序（ISR）。

中断信号由嵌套向量中断控制器（NVIC）接收。某个中断信号有效时，NVIC 通过中断请求信号将中断向量地址发送给处理器内核。作为回答，处理器内核在进入和离开 ISR 时都将发送一个确认信号。NVIC 负责启用/禁用用户配置所指定的中断。发生多个中断请求时，它会按中断优先级高低进行处理，支持嵌套中断，能够暂停一个优先级低的 ISR 转而执行一个优先级高的中断。

唤醒中断控制器（WIC）模块支持使用中断将器件从低功耗模式唤醒。低功耗模式包括：睡眠模式、深度睡眠模式和休眠模式。当 NVIC、处理器内核和其他外围设备掉电时，WIC 模块仍保持有效。当触发了某个中断时，WIC 将激活电源管理系统，这样会恢复 NVIC、处理器内核以及其他外设。这时 NVIC 将接管并发送向量地址到处理器内核，以执行 ISR。在 PSoC 4 器件中存在几种源，它们均能够唤醒器件。例如，图 1 显示了 IRQ[0] 和 IRQ[1] 被一同路由到了 WIC 和 NVIC。这些是来自 GPIO 的中断线路。

PSoC 4 具有以下中断特性：

- **可配置中断向量地址：**发生中断时，CPU 可以直接跳转并执行任何 ISR 代码，这样可缩短延迟。
- **灵活的中断源：**在传统的微控制器中，中断源是固定连接到各个中断线的。在 PSoC 中，您可灵活选择每条中断信号线的中断源。这种灵活的架构允许配置任何数字信号作为中断源使用。

2.1 中断源

PSoC 4 中断源分两种类型：

固定功能的中断源：这些是片上外设的一些预定义中断源。

通用数字模块（UDB）中断源（在 PSoC 4200、PSoC 42x7_BLE、PSoC 4200M 和 PSoC 4200L 中可用）：UDB 是不同数字功能（如定时器、PWM、UART、SPI 等）的基本构建模块。它包括可编程逻辑（PLD）、数据路径和灵活的路由。与功能固定的中断源相比，某个 UDB 生成的所有数字信号都能触发一个中断。信号通过一个被称为数字系统互连（DSI）的结构路由到中断控制器。请参见 [PSoC 4 技术参考手册](#)，了解更多信息。

表 1 显示了各个中断源。除非另有说明，否则表中提及的中断源可用于所有 PSoC 4。有关每个中断源的详细信息，请参考表 1 中列出的 PSoC Creator 组件数据手册。附录 A 依据器件给出了中断源的完整列表。

表 1. PSoC 4 中断源

中断源	详细说明	
GPIO	每个端口包含 8 个引脚。每个引脚可以产生一个中断，但是端口中的所有引脚共用了一个向量地址。固件必须识别引起中断的引脚。 PSoC 4 会在 GPIO 信号的上升沿、下降沿或双边沿上触发中断。该中断可将器件从睡眠、深度睡眠和休眠模式唤醒。	
低功耗比较器 (LPCOMP)	与 GPIO 相似，某个中断能够在比较器输出信号的上升沿、下降沿或双边沿上被触发。LPCOMP 也能够将器件从睡眠、深度睡眠和休眠模式唤醒。LPCOMP 在 PSoC 4000 器件中不可用	
WDT	看门狗定时器 (WDT) 是一个定时器，它可以复位器件或生成一个中断。PSoC 4000、4000S、4100S、4100S Plus 和 4100PS 器件具有一个 16 位的自由运行 WDT，而其他 PSoC 4 系列则有两个 16 位的 WDT 和一个 32 位的 WDT。WDT 可将器件从睡眠和深度睡眠模式唤醒。	
SCB	PSoC 4 具有多达 5 个串行通信模块 (SCB)，它们可被配置为 I ² C、SPI 或 UART。SCB 的准确数量取决于器件的系列。	
	I ² C	以下事件会生成一个中断：仲裁失败、从设备地址匹配、启动/停止检测、总线错误、字节/字传输完成、TX FIFO 不满、TX/RX FIFO 为空、RX FIFO 非空、RX FIFO 溢出以及 RX FIFO 已满。从设备的地址匹配事件可将器件从睡眠和深度睡眠模式唤醒。
	SPI	下列事件会生成一个中断：传输完成、闲置、TX FIFO 不满、TX/RX FIFO 为空、字节/字传输完成、RX FIFO 非空、尝试写已满的 RX FIFO 以及 RX FIFO 已满。
	UART	下列事件会生成一个中断：传输完成、UART TX 接收到 SmartCard 模式的 NACK、UART 在 LIN 或 SmartCard 模式下仲裁丢失、帧错误、奇偶校验错误、LIN 波特率检测完成以及 LIN 成功中断检测。也可以从低功耗模式唤醒器件 ⁽¹⁾ 。
系统性能控制器 (SPC)	SPC 模块控制着闪存写操作。当闪存写操作完成时，它会触发一个中断。	
SYSTICK	SysTick 是 ARM® Cortex®-M0/Cortex M0+处理器中内置的 24 位定时器。实时操作系统 (RTOS) 通常将其作为一个触发定时器使用。另外它也可作为一个通用定时器使用。请查看 Systick 定时器 章节，了解更多信息。	
电源管理 ⁽²⁾	当器件供电电压下降到某个阈值以下时，该模块会生成一个低压检测 (LVD) 中断。	
SAR ADC	当发生转换结束、数据溢出、扫描冲突、数据饱和以及数据超出范围等事件时，逐次逼近寄存器的模数转换器 (SAR ADC) 会生成中断。	
CapSense (CSD)	传感器扫描完成过程时，用于触摸应用的 CSD 会生成一个中断。	
定时器/计数器/脉宽调制器 (TCPWM)	可将 TCPWM 模块配置作为 16 位定时器、计数器或 PWM 使用。它可以在发生终端计数、输入捕获信号或某个“比较结果为真”事件时生成中断。	
控制器区域网络 (CAN)	PSoC 4200M 和 PSoC 4200L 器件具有两个 CAN 模块。PSoC 4100S Plus 器件具有一个 CAN 模块。CAN 模块能够在发生接收到信息、发送信息以及各种错误事件时生成中断。请参见 CAN 中技术参考手册 章节，了解更多信息。	
直接存储器访问 (DMA)	PSoC 4100M/4200M、PSoC 4200L、PSoC 4100S Plus 和 PSoC 4100PS 具有 DMA 用于在外设之间传输数据。数据传输完成时会生成中断。	
通用数字模块 (UDB)	通过 UDB 实现的定时器、PWM、计数器、UART 等之类的功能，可以在发生不同事件时生成中断，同固定功能的相应模块相类似。UDB 可用于 PSoC 4200、PSoC 42xx_BLE、PSoC 4200M 和 PSoC 4200L。	
USB	PSoC 4200L 中的 USB 具有帧起始 (SOF) 和通过数据端点进行的通信结束等中断。	
电压 DAC (VDAC)	它是可编程模拟子系统的一部分。在发生比较器触发器或 VDAC 结果准备就绪时，VDAC 将生成中断。该功能仅适用于 PSoC 4100PS。	
CTB/CTBm	提供连续时间模拟功能。发生时间（如比较器触发）时，它将生成中断。	
WCO WDT/WCO	PSoC 41000S 和 PSoC 4100S Plus 具有可由 WCO 提供时钟脉冲的计时器。这些计时器可生成中断。	

- (1) 因为引脚有限，并不是所有端口都有专用中断。如果 UART 选择的引脚没有专用端口中断，则无法唤醒器件。请参考架构技术参考手册 (TRM) 中的“中断”一节，以了解有关具有专用中断的端口。
- (2) 在 PSoC 4000 / PSoC 4000S / PSoC 4100S/4100S Plus 器件中不可用

2.2 电平及边沿触发中断

PSoC 4 支持电平和边沿触发的中断。图 2 显示的是用于选择触发类型的逻辑运算。NVIC 支持的所有中断线都遵循该逻辑运算。应该注意，固定功能的中断只能被配置为电平触发，但是对于包含了 UDB 的 DSI 源，中断可被上升沿触发，并且也可以通过电平触发。上升沿检测模块会在 DSI 中断信号每个上升沿上产生一个脉冲。请参考时序图（图 3 和图 4），了解 NVIC 如何响应电平和边沿触发的中断。

图 2. 电平触发和边沿触发

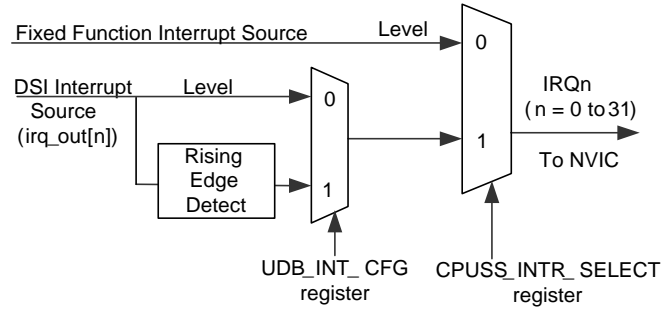


图 3. 电平触发中断

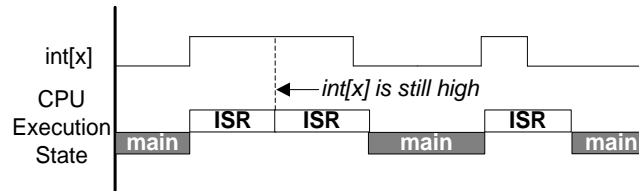
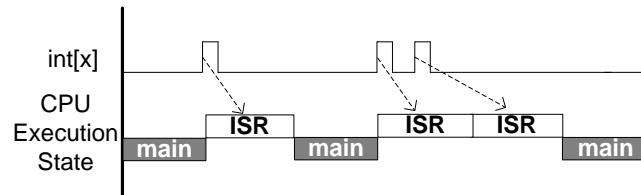


图 4. 边沿触发中断



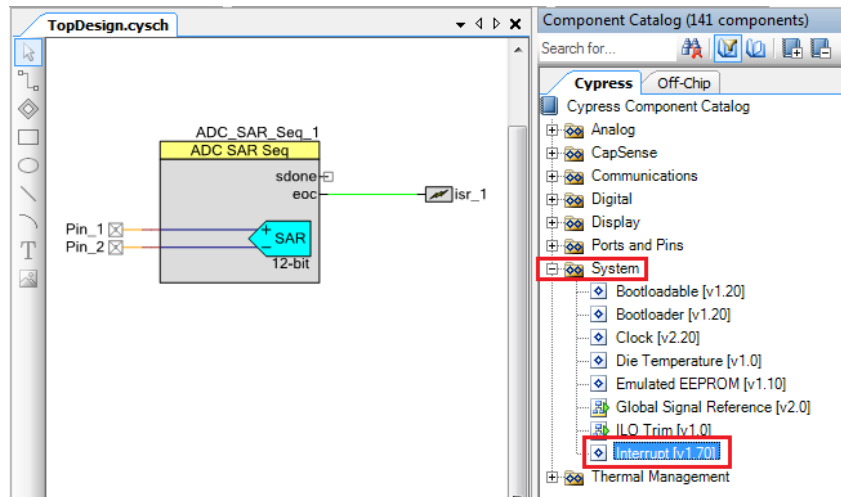
注意：GPIO 中断逻辑具有额外的电路，用于支持在上升沿、下降沿以及双沿上触发的中断。请参见 [PSoC 4 技术参考手册](#)，了解更多信息。

3 PSoC Creator 的中断支持

上述章节介绍的一些中断特性必须设置，如：电平或边沿触发、向量地址以及中断优先级。配置操作是通过 PSoC Creator 中断组件进行的。该组件位于 Component Catalog（组件目录）窗口中的 System（系统）选项卡下，如图 5 所示。

每个中断组件实例均使用了连向 NVIC 的 32 根导线中的一根中断线。图 5 所示的示例，来自 SAR ADC 的结束转换信号（EOC）被连接到中断组件“isr_1”上。SAR ADC 有一根 NVIC 的分配向量线（请参考附录 A）。例如，在 PSoC 4200 中，IRQ14 可用于 SAR ADC 中断。因此，中断组件“isr_1”通过 MUX 逻辑将 EOC 信号连接到 IRQ14 导线，如图 2 所示。

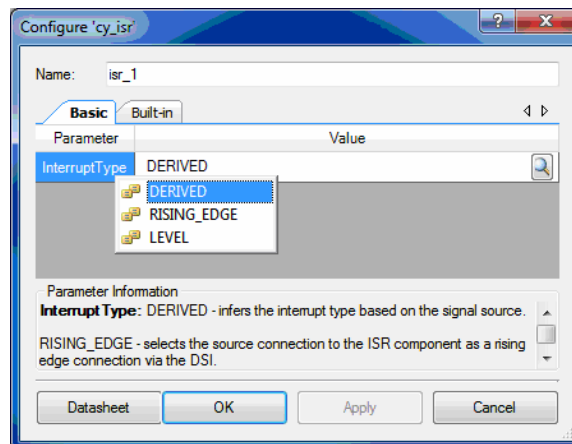
图 5. PSoC Creator 中断组件



3.1 中断组件配置

图 6 显示的是中断组件配置对话框。组件中有三个选项：DERIVED、RISING_EDGE 和 LEVEL。

图 6. 中断组件配置

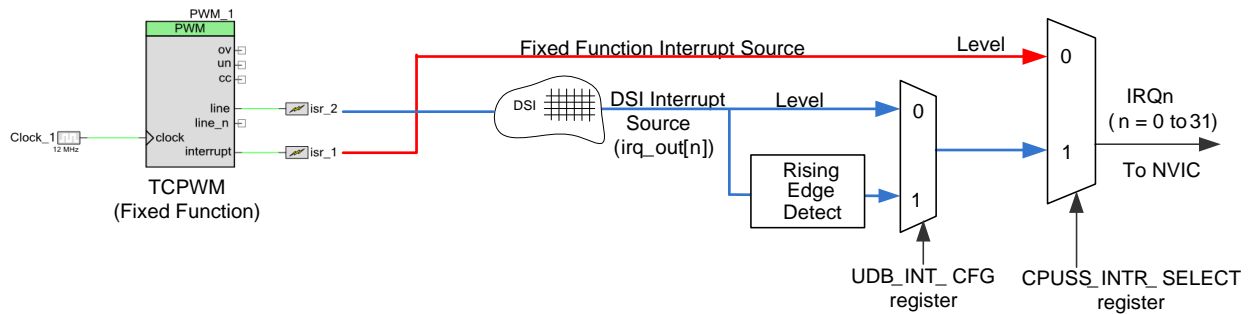


这些设置用于配置图 2 所示的复用器。根据中断源（功能固定或 UDB/DSI）和应用要求决定某个特定选项的选择情况。

固定功能模块：从固定功能模块引出的中断线通常使用“专用布线”完成，如图 7 中红线所示。配置该路径时，中断是电平触发的，并且根据正在使用的硬件模块确定向量数量。连接到中断线的中断组件（isr_1）只能被配置为电平触发。将中断设置为 RISING_EDGE 触发，会引起编译错误。配置为 DERIVED 时，工具只能将其配置为电平触发。

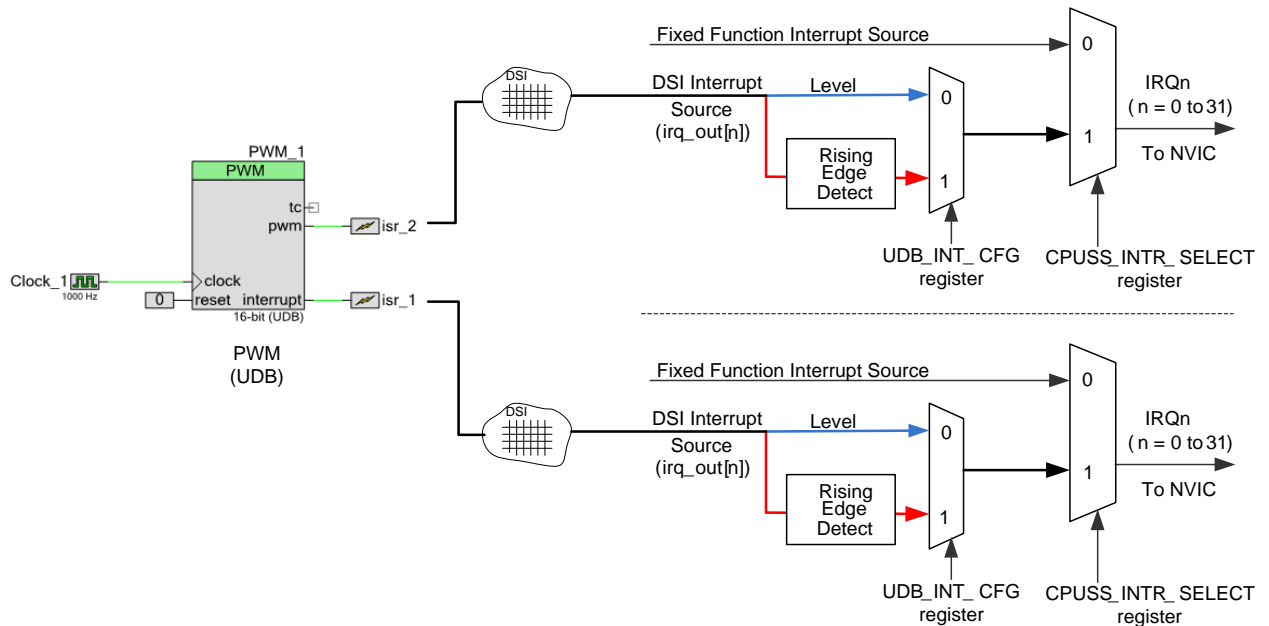
但是，在具有 DSI 的 PSoC 器件中，来自功能固定模块的其他输出信号可作为中断线使用。这样允许将来自 PWM 组件输出的线配置为 RISING_EDGE 项，如图 7 中蓝线所示。连接到来自 PWM 输出的中断组件（isr_2）可被配置为电平触发项（Level）或上升沿触发项（RISING_EDGE）。选中 DERIVED 选项时，工具会选择电平触发配置。在这种情况下，电平触发没有任何效果，这是因为它会使 ISR 在信号为高电平的情况下重复执行。因此大多情况下使用上升沿触发项（RISING_EDGE）。

图 7. 固定功能模块的中断路由



UDB：对于 UDB，使用 DSI 将信号（来自 UDB 组件的中断线或任何输出）输送给 MUX 逻辑，如图 8 所示。因此，LEVEL 和 RISING_EDGE 选项都适用于来自 UDB 的所有信号。选中中断组件（isr_1 或 isr_2）中的 DERIVED 项时，也便选中了 RISING_EDGE 项。这种情况恰好与功能固定模块输出的 DSI 信号路由情况相反。

图 8. UDB 的中断路由



注意：PSoC 4 BLE、PSoC 4200M 和 PSoC 4200L 有 8 个 DSI 通道，每个通道 4 路分解，从而为 ARM Cortex-M0 处理器提供 32 (8x4) 个中断线。因此，一个设计中 DSI 中断的最大数量被限制为 8。

表 2 提供了中断组件中设置 InterruptType 参数的指南。

表 2. 中断组件配置

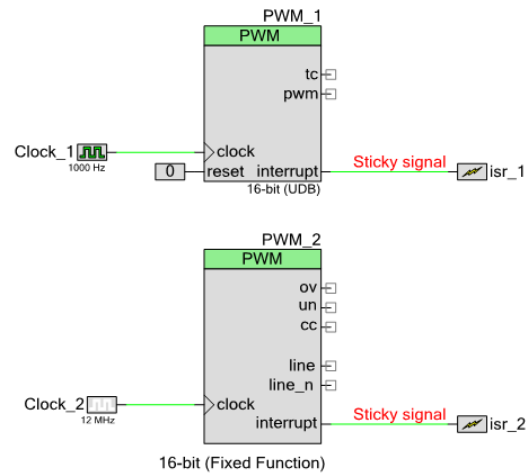
中断源	信号	中断组件配置
固定功能	中断	选择 LEVEL 或 DERIVED。不允许使用 RISING_EDGE。
	模块输出	选择 RISING_EDGE 项，否则在逻辑高电平信号期间，中断将被重复触发。
UDB 函数	中断	选择 RISING_EDGE 或 DERIVED。
	模块输出	选择 RISING_EDGE 项；如果选择 LEVEL 项，那么在逻辑高电平信号期间，中断将被重复触发。

3.1.1 粘滞（Sticky）位

中断信号可以是“粘滞”（sticky）的，这便意味着中断线保持有效状态（高电平），直到它被读取或者清除为止。在这种情况下，如果中断组件被配置为 RISING_EDGE，那么将执行一次 ISR。如果中断组件被配置为 LEVEL，那么 ISR 将重复被执行。为解决这个问题，需要使用由组件提供的 API 清除中断源。请查看中断源的组件数据手册。您也可以参考 [写入一个中断服务子程序（ISR）](#) 章节，这部分内容提供了一个使用定时器中断的例子。

请注意，当一个功能固定模块或 UDB 的输出线（例如，图 9 所示 PWM 组件的“pwm”线）被连接到中断组件，而不是中断线时，便不用清除中断。但是如果中断组件被配置为 LEVEL，那么 ISR 在信号为 HIGH 期间将重复被执行。

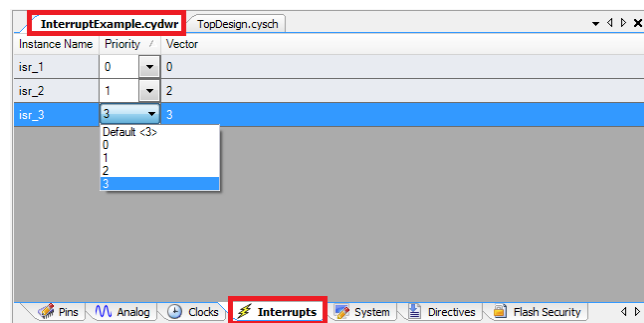
图 9. “粘滞”信号



3.2 中断优先级配置

PSoC Creator 项目的 Design Wide Resource（设计范围资源）窗口（*project_name.cydwr*）中包含一个 Interrupts 选项卡，该选项卡显示了中断组件的名称、它们的优先级以及向量编号，如图 10 所示。isr_1、isr_2 和 isr_3 为设计中使用的中断组件。

图 10. *cydwr* 窗口中的 Interrupt 选项卡



使用 *cydwr* 窗口更改中断优先级。请注意：最高优先级为 0，最低优先级为 3。Cortex-M0/Cortex M0+ CPU 支持中断嵌套，请参考 [中断嵌套](#) 了解详细信息。

构建项目时，PSoC Creator 将自动分配每个中断组件的中断向量编号，但可以手动更改。请参考 [强制中断向量编号](#) 一节中的内容，了解更详细的信息。另外，应注意 *cydwr* 窗口中显示的是向量编号的偏移量。在 Cortex-M0/Cortex M0+ 中，向量编号 0 对应异常编号 16。请参考 [异常事件](#)，了解 Cortex-M0/Cortex M0+ 异常的大概情况。

3.3 中断 API 函数

PSoC Creator 为项目中每一个组件生成一个 API `-.c` 和 `.h` 文件。这些 API 所包含的函数用于配置和使用每一个组件。下面显示的各个 API 函数同一个中断组件相关联：

- `<instance_name> Start()` 和 `<instance_name> _Stop()`
`Start()` 使能中断、将其向量设置为默认的 ISR 并设置中断优先级。
`Stop()` 禁用中断。
- `<instance_name> _StartEx()`
 它的功能与 `Start()` 相似。唯一不同的地方表现在：该函数将一个向量地址作为一个输入，这样您能够写入一个自定义 ISR，而不必使用组件生成的默认 ISR。
- `<instance_name> Enable()` 和 `<instance_name> _Disable()`
 这些函数由 `Start()` 和 `Stop()` 内部调用，用以使能和禁用中断。可以通过调用这些函数来动态使能和禁用某个中断。
- `<instance_name> SetVector()` 和 `<instance_name> SetPriority()`
 这些函数被 `Start()` 和 `Stop()` 内部调用，用以设置中断向量地址和中断优先级。也可以调用这些函数，动态设置中断的向量和优先级。请确保在调用这些函数前已经禁用了该中断。
- `<instance_name> SetPending()`
 使中断保持等待状态，而不需要硬件的中断请求。
- `<instance_name> ClearPending()`
 清除中断的挂起状态，使该中断不被服务。该函数不对中断源信号产生任何影响；它只会清除 NVIC 的中断信号线的挂起状态位。

请参见 [中断组件数据手册](#)，了解 API 的详细介绍。

3.3.1 关键部分控制函数

在 `CyLib.h` 和 `CyLib.c` 文件中，PSoC Creator 还提供了一组通用的中断函数。该项目建成时，将生成这些文件。`CyEnterCriticalSection` 和 `CyExitCriticalSection` 是关键内容。通过这两个函数可使固件变量和硬件寄存器避免发生损坏。`CyEnterCriticalSection` 可禁用中断，并返回一个中断状态值。`CyExitCriticalSection` 可恢复中断状态。

如要了解中断具体如何工作，可考虑写一个定时器控制寄存器的示例：

```
TCPWM_BLOCK_CONTROL_REG |= TCPWM_MASK;
```

执行上述语句的过程中，具体操作如下：

1. CPU 会读取 TCPWM 的控制寄存器并将数据存储在一个临时寄存器内。

CPU 对临时寄存器中的数据和它的掩码值进行逻辑或（OR）运算。

CPU 将逻辑或运算后的结果加载到控制寄存器内。

在步骤 1 和步骤 2 之间，可能会发生一个中断，并且它的 ISR 可能会将一个新值加载到相同的控制寄存器内。执行 ISR 后，当 CPU 继续执行步骤 2 时，它使用临时寄存器中的旧控制寄存器值，从而引起数据损坏。

为避免发生这种情况，添加了以下代码：

```
InterruptState = CyEnterCriticalSection();  
TCPWM_BLOCK_CONTROL_REG |= TCPWM_MASK;  
CyExitCriticalSection(InterruptState);
```

`CyEnterCriticalSection` 和 `CyExitCriticalSection` 函数在控制寄存器被写入时禁止中断，从而能够解决问题。某个共享的变量或寄存器被写入时，使用这些函数。

更多有关这些函数的信息，请参见系统参考指南（也可以在 PSoC Creator 菜单依次选择 **Help > Documentation**，找到该文档）。

4 写入一个中断服务子程序（ISR）

要了解如何编写一个 ISR，可考虑定时器中断实例。中断组件 “*isr_1*” 连接到 *Timer_1* 的中断终端，如图 11 所示。

编译项目后，PSoC Creator 生成与组件相关的所有文件，如图 12 所示。*isr_1.c* 和 *isr_1.h* 是中断组件 *isr_1* 生成的文件。这些文件提供了 API 用于配置和使用组件（包括 ISR）。

图 11. 定时器中断示例

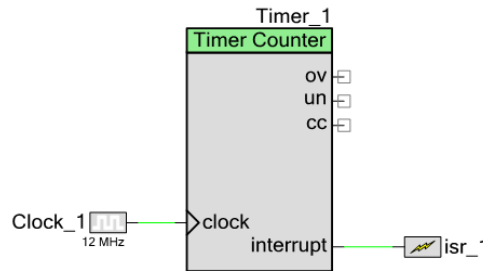
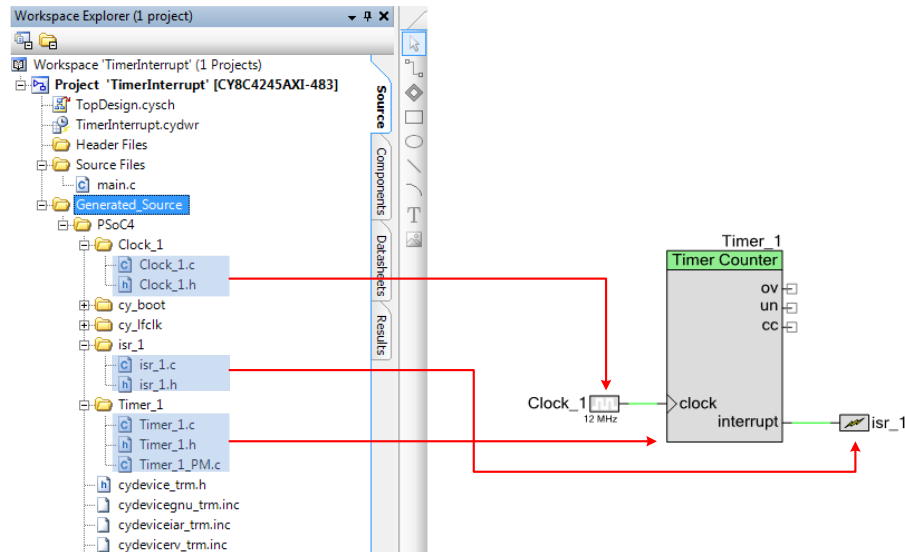


图 12. 中断组件生成的文件



有两种方式能够进行编写一个 ISR — 使用 PSoC Creator 自动生成的 ISR；创建一个自定义的 ISR 函数。

4.1 使用自动生成的 ISR

下面显示的是 *isr_1.c* 中默认生成的 ISR。ISR 函数名称的格式为 - *CY_ISR(<isr_name>_interrupt)*。

```
CY_ISR(isr_1_interrupt)
{
    #ifdef isr_1_INTERRUPT_INTERRUPT_CALLBACK
        isr_1_interrupt_interruptCallback();
    #endif /* isr_1_INTERRUPT_INTERRUPT_CALLBACK */

    /* Place your Interrupt code here. */
    /* `#START isr_1_interrupt` */

    /* `#END` */
}
```

该函数包括两部分：一个用于调用回调函数，另一个用于处理程序代码的占位符。回调函数将在下一节中介绍。您可以将处理器代码写在这个自动生成的 ISR 中标记 **#START** 和 **#END** 之间。请注意，重新生成项目文件时，写在这两个标记外的代码将被清除。

为使能中断，需要启动 **ISR** 组件。下面显示的是 *main.c* 代码，用以启动中断源，既为定时器和中断组件。

```
int main()
{
    /* Start the timer component */
    Timer_1_Start();

    /* Start the interrupt component */
    isr_1_Start();

    /* Enable global interrupt */
    CyGlobalIntEnable;

    for(;;)
    {
        /* Place your application code here. */
    }
}
```

请注意，除了需要使能中断组件外，您还要通过 **CyGlobalIntEnable** 宏使能全局中断。如粘滞（Sticky）位中的解释，在 **ISR** 内部，清除中断。在该示例中，定时器中断通过下面的 **API** 函数被清除：

```
void Timer_1_ClearInterrupt(uint32 interruptMask)
```

参数 **interruptMask** 可以是定时器组件的终端计数中断掩码，也可能是比较/捕获计数中断掩码 – 请参见定时器组件数据手册或文件 *timer_1.h*。请参阅其他组件数据手册，了解 **API** 和中断屏蔽，它通过一个特定组件来清除中断。

4.1.1 使用 **extern** 关键字

很多时候，在自动生成的 **ISR** 中，需要访问变量并调用用户源文件中定义的函数。但是要在自动生成的 **ISR** 中使用变量和函数调用，需要在 *isr_1* 文件中声明它。“**extern**”关键字是用于变量声明的。

请注意文件开头的以下代码。

```
/* *****
 * Place your includes, defines and code here
 * ***** */
/* `#START isr_1_intc` */

/* `#END` */
```

您可以直接在 **#START** 和 **#END** 标记之间声明变量和函数，也可以只要键入包含声明的头文件。下面显示的是变量和函数声明的示例。

```
/* *****
 * Place your includes, defines and code here
 * ***** */
/* `#START isr_1_intc` */

extern uint8 userVariable;
void userFunction(void);
/* `#END` */
```

4.2 使用回调函数

可以从 ISR 中调用您自己的函数，而不需要在自动生成 ISR 中编写处理器代码。这有助于区分用户代码和生成的代码。自动生成的 ISR 具有条件便宜代码。该代码由宏指令控制，用于调用回调函数。

```
CY_ISR(isr_1_Interrupt)
{
    #ifdef isr_1_INTERRUPT_INTERRUPT_CALLBACK
        isr_1_Interrupt_InterruptCallback();
    #endif /* isr_1_INTERRUPT_INTERRUPT_CALLBACK */

    /* Place your Interrupt code here. */
    /* `#START isr_1_Interrupt` */

    /* `#END` */
}
```

默认情况下，`isr_1_INTERRUPT_INTERRUPT_CALLBACK` 为被定义，因此禁用了 `isr_1_Interrupt_InterruptCallback()` 的调用。您需要在源文件中编写回调函数。

1. 使能回调函数。要实现该操作，请在项目的“Header Files”下 `cyapicallbacks.h` 文件中定义 `isr_1_INTERRUPT_INTERRUPT_CALLBACK`。另外，同时在该文件中生命下面回调函数：

```
#ifndef CYAPICALLBCKS_H
#define CYAPICALLBCKS_H

    /*Define your macro callbacks here */
    /*For more information, refer to the Writing Code topic in the PSoC
    Creator Help.*/

    #define isr_1_INTERRUPT_INTERRUPT_CALLBACK
    void isr_1_Interrupt_InterruptCallback(void);

#endif /* CYAPICALLBCKS_H */
```

请注意，在自动生成的 ISR 中使能了函数调用。

2. 与任何其他函数一样在源文件中写入回调函数。

4.3 创建自定义 ISR

可以在您自己的源文件中编写 ISR，而不一定要修改自动生成的代码。该方法在使用回调函数时有助于节省函数调用的时间为了将自己的函数（例如：MyCustomISR）当作某个 `isr_1` 中断组件的 ISR，请执行以下操作：

1. 通过使用 `CY_ISR_PROTO` 宏来声明这个自定义函数：

```
CY_ISR_PROTO(MyCustomISR);
```

使用 `CY_ISR` 宏来定义自定义函数：

```
CY_ISR(MyCustomISR)
{
    /* ISR code goes here */
}
```

在 *main.c* 文件的启动代码中，请添加对 *isr_1_StartEx()* 函数的调用，不是 *isr_1_Start()* 函数。*isr_1_StartEx()* API 函数类似于 *isr_1_Start()*，只是 *isr_1_StartEx()* 调用需要传入一个可用于 ISR 的函数名作为函数参数：*isr_1_StartEx(MyCustomISR)*；

```
CY_ISR_PROTO(MyCustomISR);
/*****
 * Function Name: MyCustomISR
 *****/
CY_ISR(MyCustomISR)
{
    /* Add code here */
}

int main()
{
    /* Start the timer component */
    Timer_1_Start();

    /* Set the custom ISR */
    isr_1_StartEx(MyCustomISR);

    /* Enable global interrupt */
    CyGlobalIntEnable;

    for(;;)
    {
        /* Place your application code here. */
    }
}
```

4.3.1 关键字 CY_ISR 的重要性

中断源文件通过使用 *CY_ISR* 宏来定义 ISR 函数。这个宏被定义在自动生成的 *cytypes.h* 文件内。将代码移植到其他 PSoC 器件系列（如 **PSoc 3** 或 **PSoc 5LP**）时，它提供兼容性，并使移植变得更加简单。

同样，*CY_ISR_PROTO* 宏声明了一个 ISR 函数原型。该声明位于中断组件的头文件中。例如，在头文件 *isr_1.h* 中，中断组件 *isr_1* 含有以下函数原型的声明：

```
CY_ISR_PROTO(isr_1_Interrupt);
```

5 代码示例

表 3 显示的是使用中断特性的代码示例清单。

表 3. 中断代码示例

代码示例	中断源
CE210557 – PSoC 4 定时器中断	定时器
CE210558 – PSoC 4 GPIO 中断	GPIO
CE95915 – PSoC® 4100/PSoC 4200 器件中 RTC 的实现	TCPWM
CE95333 – PSoC 4 的低功耗比较器	LPCOMP1:P
CE95321 – PSoC 4 的休眠和停止模式	LPCOMP、GPIO
CE95400 – PSoC 41xx/42xx 器件的看门狗定时器复位和中断	WDT
CE95275 – PSoC 4 中的序列 SAR ADC 和晶片温度传感器	SAR ADC
CE95298 – PSoC 3/4/5LP 的开关去抖动组件	GPIO、去抖动组件
CE97089 – PSoC 4 ADC 到存储器缓冲区 DMA 的传输	DMA
CE210741 – PSoC 中 UART 全双工和 printf 支持	UART

6 调试提示

本节针对调试中断项目提供了一些提示信息。下面是常会遇到的情况：

a. 中断未被触发

- 请确保中断源和全局中断被使能。
- 请检查向量是否被设置为准确的 ISR。请参考[写入一个中断服务子程序 \(ISR\)](#)，了解有关为某个中断源编写和分配处理程序的详细信息。
- 请检查是否有其他被重复触发的中断源，因而占用了整个 CPU 的带宽。
- 检查中断是否仅被触发了一次。如果中断组件被配置为上升沿触发并且中断源未被清除，那么将发生上述情况。

b. 中断被重复触发

在下面各种情况下，中断会被重复触发：

- 来自源组件的中断线连接到被配置为电平触发类型的中断组件上。可通过清除中断源来解决这个问题。
- 来自组件的数字输出（并非中断线）连接到被配置为电平触发类型的中断组件上。将中断组件配置为上升沿触发，从而在每个上升沿上触发一次中断。

请参考[粘滞 \(Sticky\) 位](#)，了解更多信息。

c. 中断仅触发一次

来自源组件的中断线连接到被配置为上升沿触发类型的中断组件上。清除中断源，从而允许在每个上升沿上触发中断。请参考[粘滞 \(Sticky\) 位](#)，了解更多信息。

d. 执行中断服务子程序 (ISR) 比预期的时间长

在 ISR 执行过程中，如果有其他优先级更高的中断被触发，会发生这种情况。可更大该中断的优先级，使它的优先级高于其他中断源的。

可通过串行线调试 (SWD) 接口调用 PSoC 4 器件的片上调试功能。通过该接口，您能够添加断点、评估和编辑变量值、检查 CPU 寄存器、观察汇编指令，以及读取和写入存储器。

调试模式有助于检查中断，具体如下：

- 要想检查中断是否被执行，将一个断点添加到 ISR 指令之一。
- 使用调试器的 **Call Stack** 窗口来检查特殊中断被执行的时间。您也可以通过该窗口检查在一个低优先级 ISR 的执行过程中是否发生了一个优先级更高的中断。

使用断点触发计数来检测中断被触发的次数。它对检查中断信号是否有干扰特别有用，若存在干扰，则这个中断可能被多次触发。

欲了解调试器的使用，请参见 PSoC Creator Help 中的“使用调试器”一节。要访问本文档，请在 PSoC Creator 中按下 **F1** 按键或者使用 **Help > Topics** 菜单。

除了使用调试器的方式外，您也可以通过引脚的“bit bang”技术实现以下操作：

- 检查 CPU 是否正在进入 ISR。
- 测量 ISR 执行时间。通过在 ISR 开始和结束时分别设置和重置引脚，可以完成这个操作。

7 高级中断主题

7.1 异常事件

使处理器暂停执行当前的代码，并转移到某个处理程序，这便是异常事件。中断是异常事件的一个子集。除中断外，操作系统应用和故障处理也是异常事件，如表 4 所示。

表 4. Arm Cortex M0 中的异常事件

异常事件	异常编号	中断优先级	说明
复位	1	-3（最高级）	在上电复位或外部复位时触发异常事件。
硬故障	3	-1	在检测未定义操作码之类的故障条件下产生异常事件。
SVCALL（管理程序调用）	11	可编程	在调用某个管理程序时（执行 SVC 指令）触发异常事件。通常在操作系统应用中使用它。
PendSV（可挂起服务调用）	14	可编程	同 SVCALL 相似，但是完成所有高优先级任务后才会转换到处理程序。
SYSTICK	15	可编程	SysTick 是 Cortex M0 中的 24 位递减计数定时器。它为操作系统应用提供周期性中断。
IRQ0 到 IRQ31	16–47	可编程	外部（引脚）或内部外设中断
复位	1	-3（最高级）	在上电复位或外部复位时触发异常事件。

请注意，异常事件编号由 Arm 定义。PSoC Creator 项目的 .cydwr 窗口内，在 **Interrupt** 选项卡显示的中断向量编号包含了一个异常偏移量。例如，中断向量 0 为异常编号 16（IRQ0）。

在器件中，复位是优先级最高的异常事件，然后是硬故障。它们的优先级是固定的，其他异常的优先级则是可编程的。PSoC Creator 为所有异常事件提供了一个默认的处理程序。对于复位，默认的处理程序为 **Cm0Start.c** 文件中的 **Reset()**。启动后首先会执行这个函数。对于所有其他异常事件，**Cm0Start.c** 文件中的函数 **IntDefaultHandler()** 是默认的处理程序。然而，所使用的包括中断的异常事件向量地址（由 PSoC Creator 组件或用户自己定义），在程序执行期间被加载到向量表内。未使用的异常事件仍使用默认的处理程序。

要确定当前正在处理的异常事件，请读取中断程序状态寄存器（IPSR）。默认处理程序正在执行时，这种方法非常有用。

有关异常事件的详细信息，请参考 <http://infocenter.arm.com/help/index.jsp>。

7.2 中断延迟

中断延迟被定义为从中断被确认到 ISR 中第一个指令被执行的延迟时间。PSoC 4 器件中的 Arm Cortex-M0 or Arm Cortex-M0+ 处理器的中断延迟分别为 16 个和 15 个 CPU 时钟周期（最坏情况），但是由于外设和 Cortex-M0/Cortex M0+ 中断线之间的同步，再需要几个 CPU 周期。表 5 提供了对于 DSI 和功能固定源中断，不同 PSoC 4 系列的同步 CPU 时钟周期延迟。

表 5. DSI 和功能固定源中断的同步时钟周期延迟

器件	DSI 中断	固定功能中断
PSoC 4000	NA	具体情况取决于外设： <ul style="list-style-type: none"> SCB-I2C、GPIO、WDT：3 个 CPU 周期 SPC、CSD、TCPWM：0 个 CPU 周期
PSoC 4100/PSoC 4200	0 个 CPU 周期	3 个 CPU 周期
PSoC 42x7 BLE / PSoC 41x7 BLE	3 个 CPU 周期	具体情况取决于外设： <ul style="list-style-type: none"> SCB-I2C、GPIO、WDT、CTBm、LPCOMP、BLE、LVD：3 个 CPU 周期 SPC、CSD、TCPWM、SAR：0 个 CPU 周期

器件	DSI 中断	固定功能中断
PSoC 4200M / PSoC 4100M / PSoC 4200L	3 个 CPU 周期	具体情况取决于外设： • SCB-I2C、GPIO、WDT、CTBm、LPCOMP、LVD：3 个 CPU 周期 • SPC、CSD、TCPWM、SAR、DMA、CAN、USB（仅使用于 PSoC 4200L）：0 个 CPU 周期
PSoC 4000S / PSoC 4100S / PSoC 4100PS	不适用	具体情况取决于外设： • SCB、GPIO、WDT、CTBm/CTB、LPCOMP：2 个 CPU 周期 • CSD、TCPWM、SAR：0 个 CPU 周期
PSoC 4100S Plus	不适用	具体情况取决于外设： • SCB、GPIO、WDT、CTBm/CTB、LPCOMP：2 个 CPU 周期 • CSD、Crypto、CAN、SAR：0 个 CPU 周期

在 Cortex M0 的 16 个周期延迟或 Cortex M0+ 的 15 个周期延迟中，会发生下列操作：

1. 处理器将当前的程序计数器（PC）、链接寄存器（LR）、编程状态寄存器（PSR）以及一些通用的寄存器推进到堆栈中。
2. 处理器从 NVIC 读出向量地址并将其更新到 PC 中。
3. 处理器更新 NVIC 寄存器。

因此，当某个 ISR 当前正在执行或即将开始的时候，Cortex M0 和 Cortex M0+ 的延迟分别为 16 个周期和 15 个周期。要使流程更有效，Cortex-M0/Cortex M0+ 处理器将实现下面两个设置：

1. **末尾连缀**：如果中断处于挂起状态，而处理器正在执行另一个中断处理，当执行完第一次中断时，将放过未堆积（un-stacking）的中断，并立即处理挂起中断。该步骤将帮您减少从堆栈恢复寄存器并将相同寄存器重新放入堆栈的时间。该方法也对减少低优先级中断的延迟起着作用。
2. **迟滞**：如果在低优先级中断的堆积期间发生更高优先级中断，处理器将优先处理更高优先级中断。堆积过程结束时，处理器将读取更高优先级中断的向量地址。一旦更高优先级中断的处理过程完成，将读取及执行被挂起的更低优先级中断处理的向量地址。该过程可排除因进入优先级更低的 ISR 并将寄存器值放入堆栈中所导致的延迟，从而减少更高优先级中断的延迟。

请注意，触发中断时，如果当前正在执行某个指令，则 ISR 的执行会产生额外的延迟。器件从中断唤醒时，执行上电序列后电压的稳定过程需要一段延迟时间。有关详细信息，请参考[器件数据手册](#)。

7.3 优化中断代码

在基于中断的应用中，ISR 代码执行时间是重要的性能要求之一。在某些应用中，当接收某个中断请求时，必须在特定的时间内执行 ISR 中的关键处理代码。此外，中断的执行时间也不能太长，否则会使主代码和其他中断的执行被停顿。要想满足这些要求，请遵循下面指南：

- **避免在 ISR 中调用冗长函数。** 诸如字符 LCD 显示子程序之类函数的执行占用更长的时间，因此阻止执行优先级更低的中断。
 推荐将不重要的函数调用移到主代码内，在 ISR 中只设置了一个标志变量。主代码将定期检查该标志。如果该标志被置位，主代码会清除它，并调用函数。
- **请为各个中断分配合适的优先级。** 在包含了多个中断的应用中，请给时间关键的中断配置更高的优先级。

7.4 带有内置中断的组件

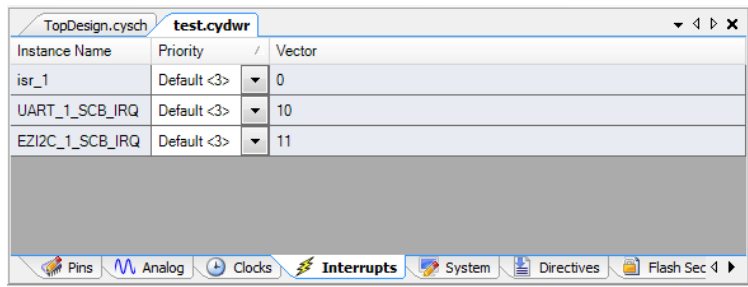
很多PSoC Creator组件在实现其功能时，已经包含了一个内部中断组件。示例包括CapSense®、SAR ADC、EZI2C和segment LCD。

与中断组件相同，这些组件中的内部ISR提供了一个用于编写用户代码的占位符区域。请参见PSoC Creator中所提供的相应组件数据手册以及相关的代码示例，了解这些组件中的中断用法。

7.5 强制中断向量编号

PSoC Creator 自动给项目中的中断组件分配向量编号。编译项目后，您可以在.cydwr窗口中的Interrupts（中断）选项卡下查看所分配的向量编号，如所示。当该中断来自DSI时，您也可以为某个中断信号选择一个特定的向量编号。本节逐步介绍了该过程。

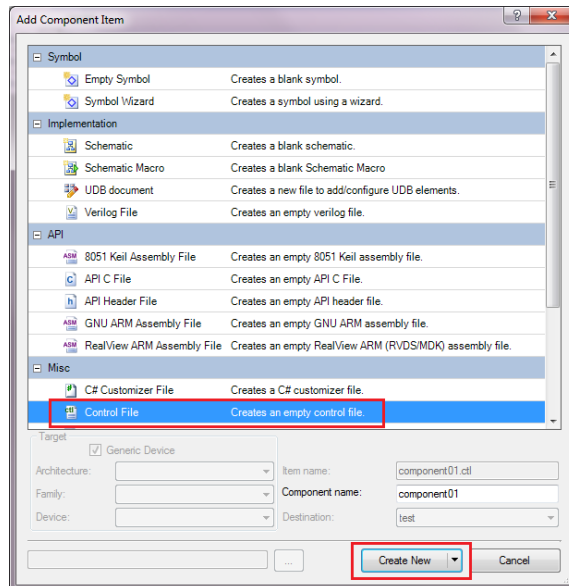
图 13. cydwr 窗口中的中断向量编号



要覆盖掉由PSoC Creator分配的向量编号，并想手动分配一个向量编号，请使用控制文件。按照下面步骤进行操作：

1. 点击‘Workspace Explorer’窗口中的**Components**选项卡。
- 右键点击**TopDesign**组件，然后点击**Add Component Item...**项。打开‘Add Component Item’对话框。
- 向下滑动到**Misc**组，选择**Control File**项，然后点击**Create New**按钮，如图14所示。

图 14. 添加控制文件



一个TopDesign.ctf文件将被创建并添加到‘Workspace Explorer’窗口内。

双击打开 *TopDesign.ctl* 文件，进行编辑。在控制文件中通过使用 **attribute** 关键词，可以指定每个中断组件的中断向量编号。指定中断向量编号的方法取决于下面两种情况：您将中断组件放在示例原理图内；或在原理图中，PSoC Creator 组件内部使用了中断组件。两种方法具体如下：

- a) 您将中断组件放在原理图内时，句法将为：

```
attribute placement_force of instance_name : label is "Intr(0, DesiredVectorNumber)";
```

这里，instance_name 是指原理图中中断组件的名称，DesiredVectorNumber 为向量编号（0 至 31）。例如，要想将向量 17 分配给中断组件 *isr_1*，该句法为：

```
attribute placement_force of isr_1 : label is "Intr(0, 17)";
```

- b) 对于内部使用中断的组件，如 EZI2C，句法将为：

```
attribute placement_force of \top_instance_name : InternalInterruptName\ : label is "Intr(0, DesiredVectorNumber)";
```

这里，top_instance_name 是指使用内部中断的组件名称。InternalInterruptName 为组件中分配给内部中断的名称。可以在 *cydwr* 窗口中的 **Interrupts** 选项卡下查找该名称。其中，中断名称是组件实例名称的前缀。在中，SCB_IRQ 是 EZI2C 组件和 UART 组件的内部中断名称。下面的语句将向量 11 分配给 EZI2C 组件。

```
attribute placement_force of \EZI2C_1:SCB_IRQ\ : label is "Intr(0,11)"
```

分配中断向量编号后，请点击 **Save** 按键，以保存控制文件中所修改的内容。

Clean and Build（清除并编译）该示例，以使新的中断向量编号分配生效。*cydwr* 窗口中的‘**Interrupts**’选项卡下显示了修改后的中断向量编号的分配情况。

7.6 SysTick 定时器

SysTick 是一个 24 位的递减计数定时器。它的中断通常用于在实时系统中进行任务切换。它使用 Cortex-M0/Cortex M0+ 内部时钟进行计数。使用下面给出的 API 配置 SysTick：

1. 设置中断处理程序

```
CyIntSetSysVector(SYSTICK_VECTOR_NUMBER, SysTick_ISR);
```

SYSTICK_VECTOR_NUMBER 是 SysTick 中断的异常编号，在 Cortex-M0 中是 15。SysTick_ISR 是中断处理程序。

配置中断周期

```
(void)SysTick_Config(CLOCK_FREQ / INTERRUPT_FREQ);
```

CLOCK_FREQ 为 CPU 时钟频率。INTERRUPT_FREQ 是由 SysTick 派生出的中断速率。

下面的内容是 SysTick 定时器用法的代码片段：

```
#define SYSTICK_INTERRUPT_VECTOR_NUMBER 15u /* Cortex-M0/M0+ hard vector */

/* clock and interrupt rates, in Hz */
#define CLOCK_FREQ 24000000u
#define INTERRUPT_FREQ 2u

CY_ISR(SysTick_ISR)
{
    /* Interrupt Handler */
}

int main()
{
    /* Point the SysTick vector to the ISR */
    CyIntSetSysVector(SYSTICK_INTERRUPT_VECTOR_NUMBER, SysTick_ISR);

    /* Set the number of ticks between interrupts */
    (void)SysTick_Config(CLOCK_FREQ / INTERRUPT_FREQ);

    /* Enable Global Interrupt */
    CyGlobalIntEnable;

    for(;;)
    {
    }
}
```

7.7 中断嵌套

NVIC 会自动处理中断嵌套，不需要任何软件开销。如果在执行优先级低的中断的过程中，有某个优先级更高的中断被触发，那么将有一些通用寄存器被推入到堆栈内，处理器从 NVIC 读取向量地址，并跳转到优先级更高的中断处理程序。完成执行后，处理器将恢复寄存器值，并执行优先级低的中断。

7.8 GlobalSignal 组件的使用

GlobalSignal 组件有助于从系统中的各种资源访问中断信号。访问的一些中断信号包括看门狗定时器中断、组合端口中断、组合低功耗比较器中断、系统性能控制器接口（SPCIF）定时器（用于闪存写入操作）等。请参考组件数据手册，了解可用中断信号的详细信息。组合端口中断如下介绍。

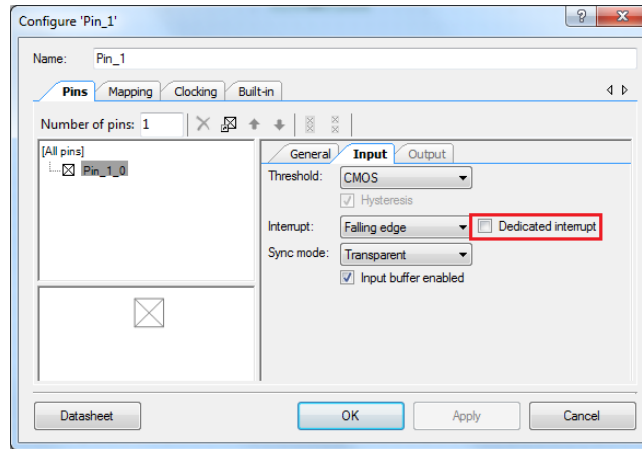
7.8.1 组合端口中断

在大多数的 PSoC 4 器件中，并非所有端口都有专用中断向量（请参考表 6）。在该情况下，建议使用 GlobalSignal 组件中的组合端口中断功能。组合端口中断通过使用一个 OR 逻辑来组合各端口中断。

例如，要在 PSoC 4100PS 器件中没有专用中断的引脚 P5 [0]上的信号生成中断，请执行以下操作：

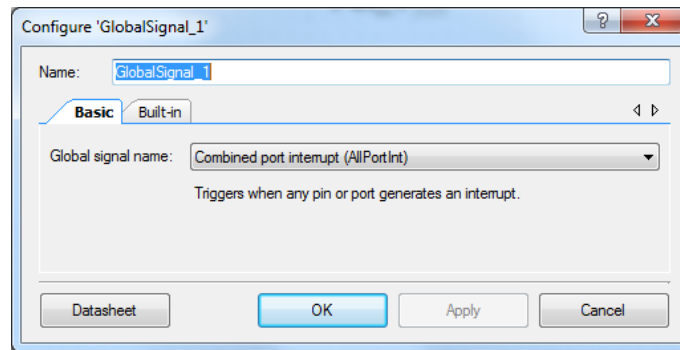
1. 将一个数字引脚组件放置到 P5[0]。
2. 配置引脚中断。确保未勾选专用中断。

图 15. 组合中断的引脚属性



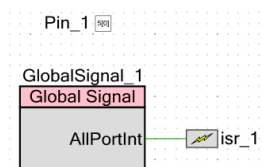
3. 将 GlobalSignal 组件放置为 “Combined Port Interrupt (AllPortInt)”。

图 16. 放置 GlobalSignal 组件



4. 将中断组件连接到 GlobalSignal 组件。

图 17. 将中断组件连接到 GlobalSignal 组件。



现在，可以对 isr_1 组件编写 ISR。请注意，如果多个端口引脚使能了中断，则应检查 GPIO_PRTx_INTR 寄存器以识别触发中断的端口引脚。请参考 *寄存器计数参考手册 (TRM)*，了解有关寄存器的详细信息。

7.9 易失性全局变量的使用

在中断中，常见的情况是使用变量作为在中断处理程序中设置并在主循环中轮询的标志。在这种情况下，编译器将通过假定程序流中没有存在代码更新标志来优化变量；这样将导致运行时发生错误。要避免此问题，请始终将在 ISR 和主循环中访问的全局变量声明为易失性变量。

8 总结

中断被广泛应用于嵌入式应用中。对于片上系统架构（如 PSoC 4），将片上外设的状态传输给 CPU 时，中断起着关键作用。通过本应用笔记所提供的信息，可以了解可用的基础设施并创建基于中断的项目。

关于作者

姓名: Rajiv Badiger
职务: 应用工程师
背景: 获得电子与通信学学士学位

附录A. PSoC 4 中的中断源和向量编码

表 6 列出了 PSoC 4 中用于 32 个中断向量的中断源列表。

表 6. PSoC 4 中断源（‘-’ 表示功能不可用）

固定功能的中断源	DSI 中断源 (不适用于 PSoC 4000/4000S/ 4100S/4100S Plus/ PSoC 4100PS)	中断向量编号								
		PSoC 4000	PSoC 4100/ 4200	PSoC 4 BLE	PSoC 4 M	PSoC 4 L	PSoC 4000S	PSoC 4100S	PSoC 4100S Plus	PSoC 4100PS
GPIO 中断 — 端口 0	DSI	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0
GPIO 中断 — 端口 1	DSI	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1
GPIO 中断 — 端口 2	DSI	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2
GPIO 中断 — 端口 3	DSI	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3
GPIO 中断 — 端口 4	DSI	-	IRQ4	IRQ4	IRQ4	IRQ4	-	-	-	-
GPIO 中断 — 端口 5	DSI	-	-	IRQ5	-	-	-	-	-	-
GPIO 中断 — 端口 13 (USB 唤醒)	DSI	-	-	-	-	IRQ5	-	-	-	-
GPIO 中断 — 所有端口*	DSI	-	-	IRQ6	IRQ5	IRQ6	IRQ4	IRQ4	IRQ4	IRQ4
-	DSI	-	IRQ5	-	-	-	-	-	-	-
-	DSI	-	IRQ6	-	-	-	-	-	-	-
-	DSI	-	IRQ7	-	-	-	-	-	-	-
LPCOMP (低功耗比较器)	DSI	-	IRQ8	IRQ7	IRQ6	IRQ7	IRQ5	IRQ5	IRQ5	IRQ5
WDT (看门狗定时器)	DSI	IRQ4	IRQ9	IRQ8	IRQ7	IRQ8	IRQ6	IRQ6	IRQ6	IRQ7
SCB0 (串行通信模块 0)	DSI	IRQ5	IRQ10	IRQ9	IRQ8	IRQ9	IRQ7	IRQ7	IRQ7	IRQ8
SCB1 (串行通信模块 1)	DSI	-	IRQ11	IRQ10	IRQ9	IRQ10	IRQ8	IRQ8	IRQ8	IRQ9
SCB2 (串行通信模块 2)	DSI	-	-	-	IRQ10	IRQ11	-	IRQ9	IRQ9	IRQ10
SCB3 (串行通信模块 3)	DSI	-	-	-	IRQ11	IRQ12	-	-	IRQ10	-
SCB3 (串行通信模块 4)	DSI	-	-	-	-	-	-	-	IRQ11	-
CTBm 中断 (所有 CTBm)	DSI	-	-	IRQ11	IRQ12	IRQ13	-	IRQ10	IRQ12	-
CTB 中断	-	-	-	-	-	-	-	-	-	IRQ6
BLE 子系统中断	DSI	-	-	IRQ12	-	-	-	-	-	-
DMA 中断	DSI	-	-	-	IRQ13	IRQ14	-	-	IRQ14	IRQ12
SPCIF 中断	DSI	IRQ6	IRQ12	IRQ13	IRQ14	IRQ15	IRQ9	IRQ10	IRQ15	IRQ13
SRSS LVD 中断	DSI	-	IRQ13	IRQ14	IRQ15	IRQ16	-	-	-	-

固定功能的中断源	DSI 中断源 (不适用于 PSoC 4000/4000S/ 4100S/4100S Plus/ PSoC 4100PS)	中断向量编号								
		PSoC 4000	PSoC 4100/ 4200	PSoC 4 BLE	PSoC 4 M	PSoC 4 L	PSoC 4000S	PSoC 4100S	PSoC 4100S Plus	PSoC 4100PS
SAR (逐次逼近 ADC)	DSI	–	IRQ14	IRQ15	IRQ16	IRQ17	–	IRQ19	IRQ25	IRQ15
CSD0 (CapSense)	DSI	IRQ7	IRQ15	IRQ16	IRQ17	IRQ18	IRQ10	IRQ13	IRQ16	IRQ14
CSD1 (CapSense)	DSI	–	–	–	IRQ18	IRQ19	–	–	–	–
VDAC 中断 (两个 VDAC)	–	–	–	–	–	–	–	–	–	IRQ16
TCPWM0 (定时器/计数器/PWM 0)	DSI	IRQ8	IRQ16	IRQ17	IRQ19	IRQ20	IRQ11	IRQ14	IRQ17	IRQ17
TCPWM1 (定时器/计数器/PWM 1)	DSI	–	IRQ17	IRQ18	IRQ20	IRQ21	IRQ12	IRQ15	IRQ18	IRQ18
TCPWM2 (定时器/计数器/PWM 2)	DSI	–	IRQ18	IRQ19	IRQ21	IRQ22	IRQ13	IRQ16	IRQ19	IRQ19
TCPWM3 (定时器/计数器/PWM 3)	DSI	–	IRQ19	IRQ20	IRQ22	IRQ23	IRQ14	IRQ17	IRQ20	IRQ20
TCPWM4 (定时器/计数器/PWM 4)	DSI	–	–	–	IRQ23	IRQ24	IRQ15	IRQ18	IRQ21	IRQ21
TCPWM5 (定时器/计数器/PWM 5)	DSI	–	–	–	IRQ24	IRQ25	–	–	IRQ22	IRQ22
TCPWM6 (定时器/计数器/PWM 6)	DSI	–	–	–	IRQ25	IRQ26	–	–	IRQ23	IRQ23
TCPWM7 (定时器/计数器/PWM 7)	DSI	–	–	–	IRQ26	IRQ27	–	–	IRQ24	IRQ24
CAN0 中断	DSI	–	–	–	IRQ27	IRQ28	–	–	IRQ26	–
CAN1 中断	DSI	–	–	–	IRQ28	IRQ29	–	–	–	–
USB 的起始帧	DSI	–	–	–	–	IRQ30	–	–	–	–
USB EP1-EP8 数据	DSI	–	–	–	–	IRQ31	–	–	–	–
Crypto 中断	–	–	–	–	–	–	–	–	IRQ27	–
WCO/WDT 中断	–	–	–	–	–	–	–	–	IRQ13	IRQ11
–	DSI	–	IRQ20	IRQ21	IRQ29	–	–	–	–	–
–	DSI	–	IRQ21	IRQ22	IRQ30	–	–	–	–	–
–	DSI	–	IRQ22	IRQ23	IRQ31	–	–	–	–	–
–	DSI	–	IRQ23	IRQ24	–	–	–	–	–	–
–	DSI	–	IRQ24	IRQ25	–	–	–	–	–	–
–	DSI	–	IRQ25	IRQ26	–	–	–	–	–	–
–	DSI	–	IRQ26	IRQ27	–	–	–	–	–	–

固定功能的中断源	DSI 中断源 (不适用于 PSoC 4000/4000S/ 4100S/4100S Plus/ PSoC 4100PS)	中断向量编号								
		PSoC 4000	PSoC 4100/ 4200	PSoC 4 BLE	PSoC 4 M	PSoC 4 L	PSoC 4000S	PSoC 4100S	PSoC 4100S Plus	PSoC 4100PS
–	DSI	–	IRQ27	IRQ28	–	–	–	–	–	–
–	DSI	–	IRQ28	IRQ29	–	–	–	–	–	–
–	DSI	–	IRQ29	IRQ30	–	–	–	–	–	–
–	DSI	–	IRQ30	IRQ31	–	–	–	–	–	–
–	DSI	–	IRQ31	–	–	–	–	–	–	–

文档修订记录

文档标题：AN90799 — PSoC® 4 中断

文档编码：002-11590

版本	ECN	提交日期	变更说明
**	5184198	03/22/2016	本文档版本号为Rev**，译自英文版 001-90799 Rev*B。
*A	5799164	07/05/2017	更新徽标和版权。
*B	6651887	03/31/2020	本文档版本号为Rev*B，译自英文版 001-90799 Rev*E。

全球销售和 design 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
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。

赛普拉斯不对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，例如未经授权的使用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、设定为或授权使用作武器操作、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。