AN73212

使用 PSoC® 1 进行调试

作者：Dan Sweet
相关工程项目：无
相关器件系列：所有 PSoC 1 系列
软件版本：PSoC Designer 5.1
相关应用笔记：无

摘要

AN73212 将介绍 PSoC® 1 调试器系统的各个组件，并说明如何配置并高效地使用这些组件。本文档还将介绍几种通用调试技巧，帮助您解决堆栈溢出和内存泄露等常见问题。此外，本文档亦提供了故障排除指南。

目录

简介 1
调试硬件 1
在线仿真器 (ICE) 1
片上调试器 (OCD) 部件 2
调试转接板 2
调试硬件环境建立 4
调试环境 – PSoC Designer IDE 6
启动调试器 6
调试控件 7
追踪 9
事件查看器 10
映射文件 (.mp) 16
列表文件 (.lst) 17
PSoC 1 调试提示和技巧 17
在“停止”期间硬件仍然运行 17
闪存写入调试 18
在睡眠模式下使用调试器 18
其他调试方法 18
使用 I2C-USB 桥接器进行调试 18
使用 UART 接口调试 19
引脚触发 19
故障排除 19
总结 21
附录 A – 向 PCB 添加 OCD 部件 21
附录 B – 堆栈溢出 23
不使用 ICE 检查堆栈溢出 23
避免堆栈溢出的方法 23
避免使用 CASE 语句 23
避免使用复杂的 IF 语句 23
限制传递变量的数量和大小 24
限制局部变量的数量 24
附录 C – 旧版硬件 25
关于作者 25
产品 27
PSoC 解决方案 27
赛普拉斯开发者社区 27

简介

本应用笔记宗旨在于介绍 PSoC 1 中有效的硬件和软件调试器工具，并将提供几种常用的调试技巧。

调试系统的主要硬件元件为：一个在线仿真器 (ICE)、一个调试转接板及一个支持片上调试器 (OCD) 的 PSoC 1 器件。这些元件及其配置的说明将在本应用笔记的调试硬件部分中予以说明。

软件工具则以 PSoC Designer® 为主。这一集成开发环境可提供很多调试工具，包括断点、监视变量、内存查看器、追踪、事件和输出文件（列表和映射）。在本应用笔记的调试环境 – PSoC Designer IDE 部分将介绍这些工具。

此外，本文档还将讨论以下主题：

- PSoC 1 调试提示和技巧。
- 其他调试方法。
- 故障排除。

调试硬件

本部分将介绍构建 PSoC 1 调试系统所需的硬件，包括一个 ICE、一个调试转接板和相应的连接器。以下各部分将对每个工具进行具体描述。

在线仿真器 (ICE)

在线仿真器 (ICE) 或 ICE-Cube，是 PSoC 1 调试系统中的关键工具。ICE 负责管理在计算机上运行的调试器软件 (PSoC Designer) 和 支持片上调试(OCD)的 PSoC 芯片之间的所有仿真通信。ICE 如图 1 所示。

图 1. 在线仿真器 (ICE)

ICE 包含在 CY3215-DK 开发工具包中，该工具包还提供启用调试所需的全部附件。

片上调试器 (OCD) 部件

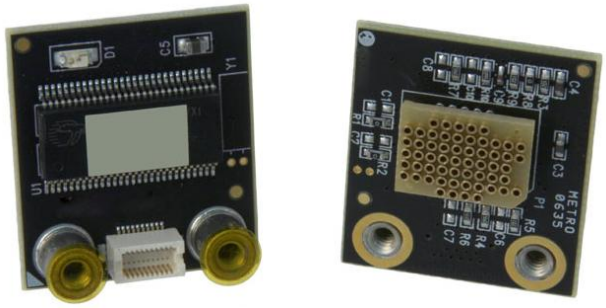
并非所有的 PSoC 1 芯片都支持调试功能。但是每个 PSoC 1 器件系列至少均有一个版本的器件可以支持调试。这些支持调试的 PSoC 器件称为 OCD 部件。一个器件系列的 OCD 部件可以对该系列的所有部件进行仿真。例如，CY8C27002-24PVXI OCD 部件可以对 CY8C27xxx 系列的所有部件进行仿真。

有些器件系列包含多个 OCD 部件，这些部件可以提供不同的封装；其他的器件系列则只有一个面向整个系列的 OCD 部件。请参阅器件数据手册，了解每个 PSoC 1 器件系列 OCD 部件的部件编号和引脚分布。

调试转接板

调试转接板是 PSoC 1 调试系统的另一重要工具。一般来讲，转接板是一个小的 PCB，包含一个支持片上调试器 (OCD) 的 PSoC 1 芯片、一个 ICE 连接器以及一个将 PCB 连接至现有电路板或开发工具上的连接器。调试转接板有两种主要类型：CY3250 转接板（如图 2 所示）和 CY3210 评估转接板（如图 3 所示）。

图 2. CY3250 转接板示例



CY3250 转接板最适合用于为新的或现有的客户 PCB 添加临时调试能力。CY3250 转接板包含一个 OCD PSoC 部件、一个 ICE 连接器、一个可通过支脚工具安装至电路板的连接器以及几个被动供电器件。

如需获取可用的 CY3250 转接板列表，请参考所用 PSoC 1 器件的器件数据手册，查看其中的仿真和编程附件表。

CY3210-EvalPod 最适合用于以评估工具或其他具有 28 引脚 DIP 连接器为原型而设计的电路板调试。大部分 PSoC 1 器件系列均支持 CY3210-EvalPod。每个 EvalPod 均配有一个支持 OCD 的 PSoC 1 器件、一个安装 28 引脚的 PDIP 连接器电路板、一个连接 ICE 的 RJ-45 连接器、一个 5 引脚编程头和若干个用于便捷访问引脚的连接器。一般来讲，在与现有开发工具包、实验板或者其他配有 28 引脚 PDIP 连接器的电路板进行调试时，EvalPods 表现最佳。

图 3. CY3210 EvalPod 示例

表 1 列出了可用的 CY3210 EvalPod。大多数常用系列中均有一个可用的 CY3210 EvalPod。

表 1. CY3210-EvalPod 列表

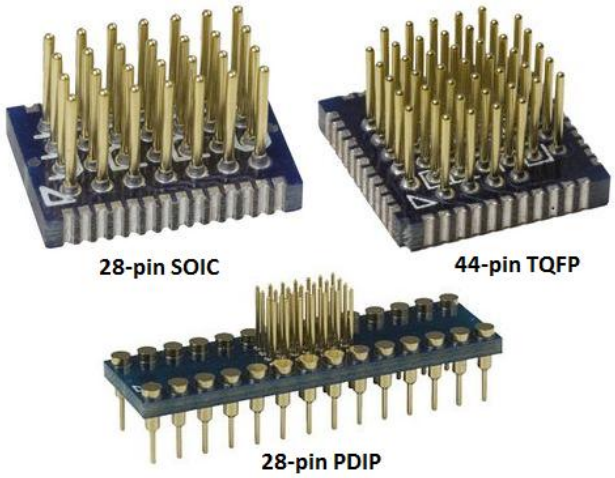
部件系列	EvalPod 部件编号
CY8C20x24/34	CY3210-20x34
CY8C20xx6	CY3210-20x36/46/66
CY8C21x23	CY3210-21x23
CY8C21x34	CY3210-21x34
CY8C24x94	CY3210-24x94
CY8C27xxx	CY3210-27x43
CY8C28xxx	CY3210-28xxx
CY8C29xxx	CY3210-29x66

除了可以使用转接板，还可以直接将 OCD 部件和 ICE 连接器放置到新的 PCB 上。如果转接板不适用于 PCB 或者目标 PSoC 1 器件没有适用的转接板，这种直接安装的方式将非常实用。有关直接在 PCB 上构建调试能力的更多信息，请参阅附录 A – 向 PCB 添加 OCD 部件。

支脚(测试引脚)

要将 CY3250 型转接板正确安装在电路板上，需要使用一个支脚套件和屏蔽模具。使用支脚套件可以将转接板安装至特定的封装，这样，单一的转接板便可安装在几乎所有的电路板上。使用配有支脚套件的 CY3250 转接板和一个 ICE 将可更改现有 PCB，使其支持调试。封装恰当的支脚套件可连接到现有 PCB，则转接板可安装至支脚上。图 4 为三种支脚的图样。

图 4. 支脚图样



如图 4 所示，支脚的底部分别适用于不同的 PCB 封装。图 4 仅显示了 28 引脚 SOIC、 44 引脚 TQFP 和 28 引脚 PDIP，所有的 PSoC 1 封装尺寸和类型均有可用的支脚套件。

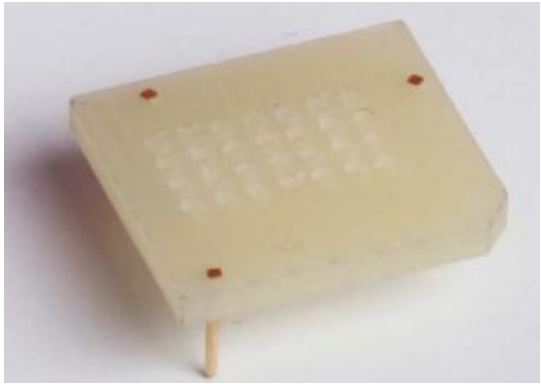
支脚套件的上半部分提供了连接到转接板的相应引脚。封装不同，引脚的数目也不同，因此，每个支脚套件应与 OCD 部件的必要引脚相连。例如，转接板上的某个 OCD 部件可能采用了 56-QFN 封装，但要对仅有 28 引脚的部件进行仿真时，支脚套件可以仅连接 56 个可用引脚中的 28 个。

每种 PSoC 器件封装适用的支脚套件列于器件数据手册中的仿真和编程附件表。

屏蔽模具

屏蔽模具安装于转接板和支脚套件之间，用于将支脚套件的引脚对齐至转接板相应的孔中，并可罩住转接板上无需使用的接孔。一些转接板上的连接点很多，而有些支脚的连接点则很少（例如 8 引脚封装）。因此，屏蔽模具能够保证支脚套件和转接板连接器正确对齐。屏蔽模具如图 5 所示。

图 5. 屏蔽模具示例



并非所有的支脚和转接板连接都需要屏蔽模具，一些转接板仅适用于某一定向的支脚套件。例如，所有基于 QFN 的转接板在连接至 QFN 支脚套件时，均不需要使用屏蔽模具。屏蔽模具有两种常用尺寸：一种兼容 28 引脚封装（DIP、SOIC 和 SSOP 封装）的屏蔽模具,以及另一种适用于任何 8 引脚或 20 引脚封装的屏蔽模具。44 引脚或 48 引脚封装的支脚套件不需要使用屏蔽模具。

如有需要，CY3250 转接板或者支脚套件中会提供相应的屏蔽模具，您无需单独购买。

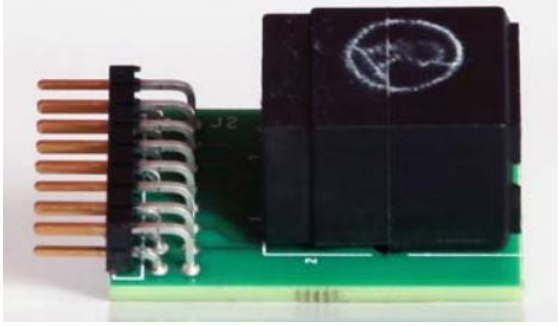
调试硬件环境建立

本部分将说明在构建功能性调试系统时各个不同硬件的连接方法。根据转接板、支脚套件和屏蔽模具的不同，配置亦可能略有区别。下面将介绍几种常用的配置。

ICE + CY3210 EvalPod

连接 ICE 和 CY3210 EvalPod 需要使用以下部件：

1. ICE
2. 面向目标 PSoC 1 系列的 CY3210 EvalPod
3. RJ-45 ICE 适配器，也被称为“向后兼容适配器”，因为某些旧有转接板亦使用 RJ-45 连接器。



4. RJ-45/以太网线缆（12 英寸或更短），CY3215-DK 套件中附带。*不能*使用长线缆代替，因为长线缆会干扰 ICE 与连接板之间的通信。

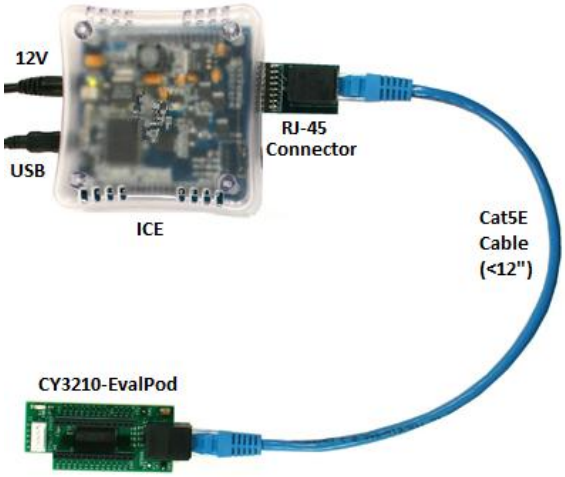
如需使用替代线缆，可以选用 12 英寸或更短的 Cat5E 标准线缆。

注 ICE 通信不是标准的以太网协议。其使用的是定制通信协议。

5. USB 线缆和 12V 电源（ICE CY3215-DK 中附带）。

以上部件的正确连接方法如图 6 所示。可以选择将 EvalPod 插入带有 28 引脚 DIP 连接器的电路板中。

图 6. ICE + CY3210 EvalPod 配置



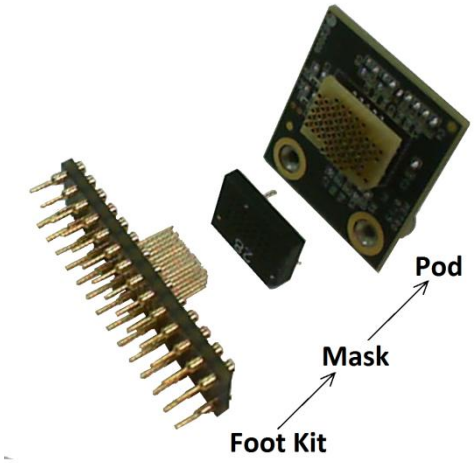
ICE + CY3250 Pod

要将 ICE 连接至 CY3250 连接板，应采取以下步骤：

1. 选择一个支脚，该支脚应与目标电路中的 PSoC 器件引脚分布相匹配。
2. 如有需要，请选择与所需支脚匹配的屏蔽模具。通常，只有 8 引脚、20 引脚和 28 引脚的支脚需要屏蔽模具。
3. 将屏蔽模具插入转接板底部，将屏蔽模具的斜切角与转接板上的引脚 1 的三角形对齐。
4. 通过屏蔽模具插入支脚。如果未使用掩模，则支脚将直接连接至转接板基座。

   步骤 2 到 4 如图 7 所示。

图 7. 支脚套件 + 屏蔽模具 + 转接板

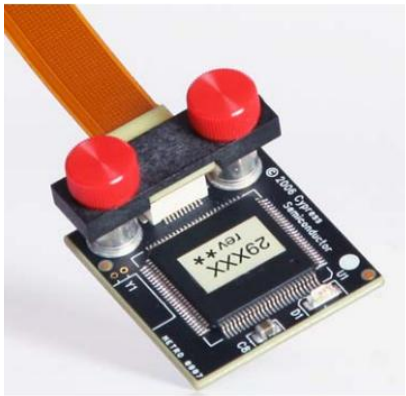


5. 将装好的转接板连接至目标电路板（如果支脚套件尚未安装于电路板上）。

6. 将转接板与 CY3250-Flex 线缆较小的一端相连。转接板、线缆和连接器如图 **8** 所示。完成的连接如图 **9** 所示。

图 8. CY3250 转接板、柔性线缆和连接器

图 9. 柔性线缆连接



7. 将柔性线缆的另一端插入 ICE，如图 10 所示。

图 10. ICE 连接至 CY3250 转接板

8. 最后，用一条 USB 线缆将 ICE 连接至计算机并通过提供的 12V 直流电源为 ICE 供电。

ICE + OCD 器件的板上安装

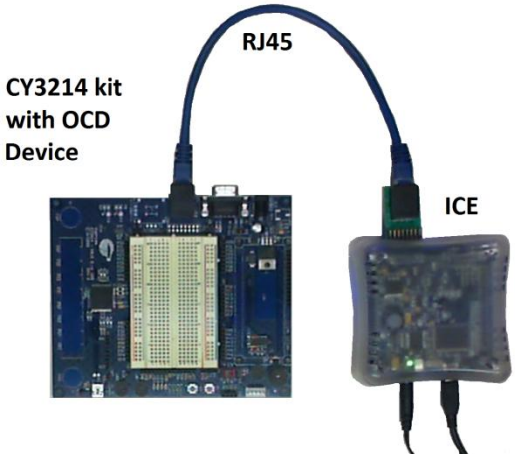
在某些情况下，直接将 OCD 部件安装在目标电路板上可不使用转接板进行调试。将相应的调试引线引出并连接到同样安装于电路板上的连接器上。这种方法适用于当目标 PSoC 1 器件没有相应的转接板或者转接板不适用于目标 PCB 时。

有时，将 OCD 器件直接放置在电路板上是更符合成本效益的调试方式。有些 PSoC 1 开发板使用这种方法来启用调试。这些开发板通常可被片上 RJ-45 连接器识别。

有关如何在电路板上直接添加 OCD 部件的更多信息，请参阅附录 A – 向 PCB 添加 OCD 部件。

使用已安装的 OCD 器件将 ICE 连接至电路板类似于上文所述的将 ICE 连接至 CY3210-EvalPod 的过程。惟一的区别是线缆将连接至目标电路板 RJ-45 连接器，而不是 EvalPod。请参考图 11 中的示例：

图 11. ICE + CY3214 及板上 OCD 器件



使用 ICE 进行器件编程

ICE 也可以使用 CY3215-DK 附带的 5 引脚在线串行编程 (ISSP) 线缆进行 PSoC 器件编程。按以下步骤操作：

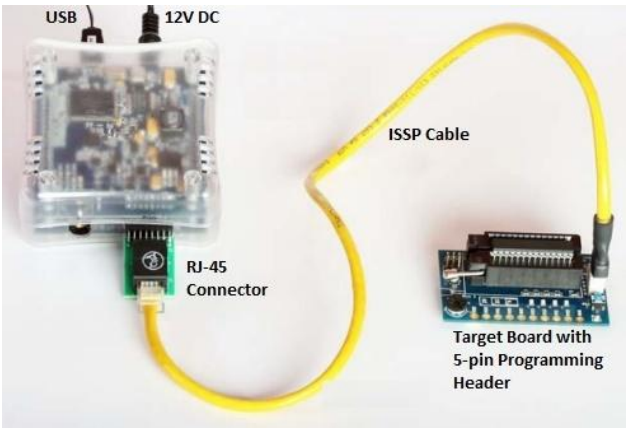
1. 将 ICE 连接至 RJ-45 适配器。
2. 将 ISSP 线缆连接至 RJ-45 适配器。线缆通常为黄色或黑色，一端为 RJ-45 连接器，另一端为白色 5 引脚连接器。

图 12 - ICE ISSP 线缆

3. 将 5 引脚连接器母头与目标电路板上的 5 引脚编程头相连。

图 13 显示的是一个 ICE 和为 ISSP 配置的目标电路板。

图 13. ICE 编程配置



注 此配置无法进行调试。5 引脚 ISSP 线缆连接的是器件的编程引脚，而调试用引脚与编程引脚并不相同。

调试环境 – PSoC Designer IDE

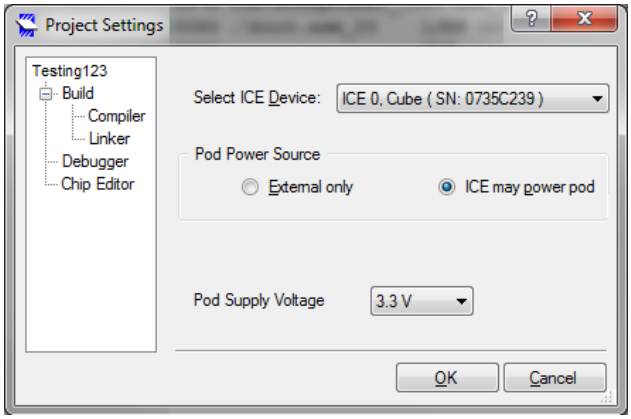
PSoC Designer 拥有一套功能强大的代码开发和调试工具。这些调试工具可用于 ICE 及本应用笔记调试硬件环境建立部分中提及的所有硬件配置。

下面将简要介绍 PSoC Designer 中的关键调试步骤。要了解关于各种特性的更多信息，请参阅 IDE 用户指南中的调试器部分。

启动调试器

要将 PSoC Designer 连接至 ICE 并开始调试，请确保 ICE 和电路板均已供电，且 ICE 已通过 USB 线缆与计算机相连。目标电路板可独立供电，也可通过 ICE 供电。要控制目标电路板的供电方式，可以在 PSoC Designer 中进行调试器设置。调试器设置可通过 **Project（项目） -> Settings（设置） -> Debugger（调试器）** 窗口访问，如图 14 所示。

图 14. 调试器项目设置



如果同一计算机连接了多个 ICE 器件，则您可以通过图 14 中的设置选择连接至哪个 ICE。此设置还可以配置 ICE 为转接板供电。如果用 ICE 为转接板供电，则必须指明转接板的供电电压。仅可使用 5.0 V 和 3.3 V 的供电电压。

在将 ICE 连接至计算机，并应用了适当的设置后，系统便可进行调试了。PSoC Designer ICE 工具栏提供了各种控件，用于进行 ICE 连接，其功能如表 2 所示。

表 2. ICE 连接控制

图标	功能	说明
	Connect（连接）	将 PSoC Designer 连接至已安装的 ICE Cube。
	Download（下载）	将编译代码下载至 ICE 和 OCD 部件。
	Refresh M8C Views（刷新 M8C 视图）	更新 PSoC Designer 中显示的内存和变量。
	Get Next Trace Data（获取下一追踪数据）	在追踪窗口中添加 64 行代码执行信息。请参阅 **追踪**部分以获取更多信息。

如果在向 ICE 连接或下载程序的过程中出现问题，请参阅本文档后面的故障排除部分。

调试控件

PSoC Designer 提供了一系列命令，可以控制在所连接的 ICE 和 OCD PSoC 上的代码执行。调试器支持若干种控制程序流的标准方法，如表 3 所示。

表 3. 基本调试控件

图标	功能	说明
	Go（启动）	启动调试器。
	Run to Cursor（运行至光标处）	在光标当前所处位置为源代码创建一个临时（不可见）断点，并运行应用程序到该断点处暂停。
	Halt（终止调试）	停止调试器。
	Reset（复位）	将器件的 PC 值复位至“0”并重启调试器。
	Step Into（单步执行）	单步执行下一语句。
	Step Out（单步跳出）	跳出当前函数。
	Step Over（单步跳过）	跳过下一语句。
	Step ASM（单步 ASM）	执行一行汇编代码。如果当前代码行为 C 代码，则将打开列表文件并执行一行汇编代码。
	Execute Program（执行程序）	连接至 ICE，下载程序并开始运行。

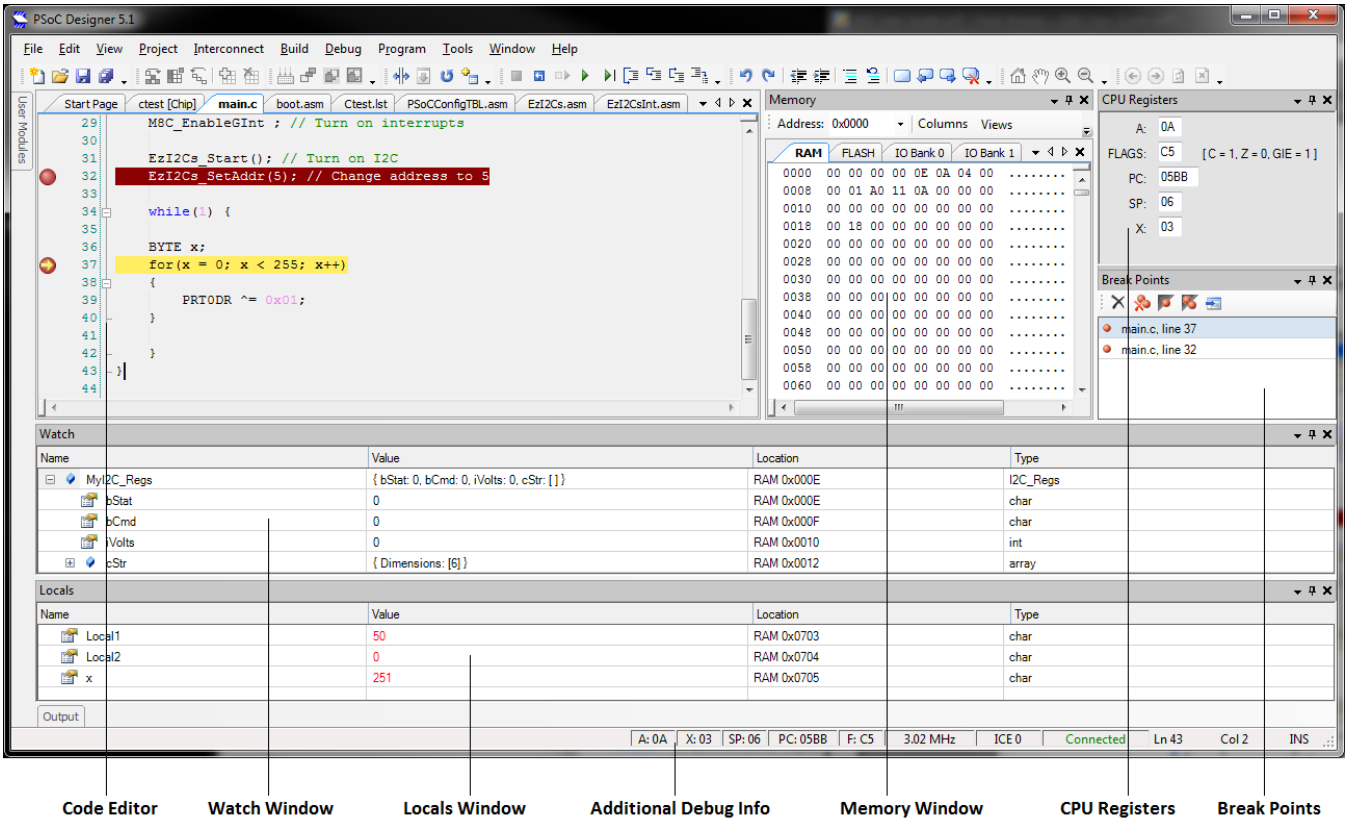
断点是 PSoC Designer 和 ICE 的另一个调试工具。一个断点即是代码中的一个标志，在运行到标志行时，将暂停执行。左键单击代码行旁的边缘即可添加断点，也可以在代码编辑器中右键单击相应行，并选择“Insert Breakpoint”（插入断点）。插入了断点的代码行如图 15 所示。图中亦显示了一个“书签”（第 15 行），您可以右键单击代码行旁边的边缘添加书签。书签用于标记代码中需要注意的地方，并不影响代码的执行。

图 15. 书签和断点示例

```
12      int i,j=0;
13      while(1){
14          for (i=0;i<256;i++)
15          {
16              for (j=0;j<256;j++)
17              {
18                  asm("nop");
19                  asm("nop");
20              }
21          }
22      }
```

第 8 页的图 16 显示了在一个调试会话期间 PSoC Designer 中可用的众多调试窗口。图中所示的全部窗口均为可选的，可以根据您的喜好移动并改变大小。下面将简要描述每个窗口，关于各个窗口的更多信息，请参阅 IDE 用户指南。

图 16. PSoC Designer 调试窗口



*Code Editor（代码编辑器）窗口：*代码编辑器窗口在调试会话期间显示代码。该窗口还显示断点位置以及代码暂停执行的位置（如果程序未运行）。调试过程中不能编辑代码。如果在调试器运行时编辑代码，PSoC Designer 将会退出调试模式，并允许对代码进行修改。

Watch（监视）窗口： 监视窗口将在调试会话期间显示用户添加的所有监视变量。要向监视窗口中添加变量，可以在调试会话期间，于代码编辑器窗口右键单击该变量，然后单击“Add Watch”（添加监视）。

*Locals（本地）窗口：*本地窗口显示程序暂停时处于范围内的局部变量。当前范围中的局部变量将会自动添加。

*其他调试信息：*PSoC Designer 总是会在屏幕下方显示有用的调试信息。这些信息包括：

- 累加器的当前值 (A)
- X 寄存器 (X)
- 堆栈指针 (SP)
- 程序计数器 (PC)
- 标志寄存器 (F)

此外，ICE 的连接速度和连接状态信息亦将在此显示。

Memory（内存）窗口： 程序暂停时，用户可以在内存窗口中查看芯片上的各个存储器区域。在内存调试窗口中，每个区域（RAM、FLASH、IO Bank 0、IO Bank 1）均有独立标签。在激活调试会话期间，可以编辑 RAM、IO Bank 0 和 IO Bank 1。

CPU Rigisters（CPU 寄存器）窗口： CPU 寄存器调试窗口显示 PSoC 器件的内部各 CPU 寄存器。这些寄存器为累加器 (A)、标志寄存器、程序计数器 (PC)、堆栈指针 (SP) 和 X 寄存器。在激活调试会话期间，CPU 寄存器窗口显示的值可以进行编辑。

*Breakpoints（断点）窗口：*断点窗口显示项目中所有已放置的断点，包括文件名称和每个断点所处的行号。可以在该窗口单独删除、禁用或启用某个或全部断点。

追踪

用户可以通过 PSoC Designer 和 ICE 调试器中的 Trace（追踪）窗口追踪和记录每条指令的活动。该窗口将显示连续可配置的器件内部操作列表。每次开始执行程序时，均会清空追踪缓冲区。缓冲区在写满后，将继续运行并覆盖之前的数据。

共有三种可用的追踪模式，描述如下。

追踪模式 1 - 仅 PC – 列出程序计数器 (PC) 以及 PSoC 执行的每条指令的汇编指令。

图 17. 追踪模式 1 - 仅 PC

Trace.txt

	PC / SYMBOL	INSTRUCTION
1	PC / SYMBOL	INSTRUCTION
2	0456	NOP
3	0455	NOP
4	0454	NOP
5	0453	NOP
6	039C	RETI
7	0390	JNC 039Ch
8	038E	INC [0006h]
9	038A	JZ 038Eh
10	0387	TST [0001h], FFh
11	0383	JZ 0387h
12	0380	TST [0002h], FFh

追踪模式 2 - PC/寄存器 – 列出 PC、指令、数据、累加器 (A)、X 寄存器、堆栈指针 (SP)、标志寄存器和 SRAM 分页。

图 18. 追踪模式 2 - PC/寄存器

Trace.txt

	PC / SYMBOL	INSTRUCTION	DATA	A	X	SP	FLAGS	SRAM PAGE
1	PC / SYMBOL	INSTRUCTION	DATA	A	X	SP	FLAGS	SRAM PAGE
2	0456	NOP	18	18	5B	02	C5	00
3	0455	NOP	18	18	5B	02	C5	00
4	0454	NOP	18	18	5B	02	C5	00
5	0453	NOP	18	18	5B	02	C5	00
6	039C	RETI	18	18	5B	02	C5	07
7	0390	JNC 039Ch	18	18	5B	05	00	00
8	038E	INC [0006h]	03	18	5B	05	00	00
9	038A	JZ 038Eh	18	18	5B	05	02	00
10	0387	TST [0001h], FFh	18	18	5B	05	02	00
11	0383	JZ 0387h	18	18	5B	05	02	00
12	0380	TST [0002h], FFh	18	18	5B	05	02	00

追踪模式 3 - PC/时间统计 – 列出 PC、指令、累加器 (A)、SRAM 分页和时间统计。时间统计会持续记录程序执行的 CPU 周期，能够有效地对一段代码所需执行的 CPU 周期进行计数。

图 19. 追踪模式 3 - PC/时间统计

Trace.txt

	PC / SYMBOL	INSTRUCTION	A	SRAM PAGE	TIMESTAMP
1	PC / SYMBOL	INSTRUCTION	A	SRAM PAGE	TIMESTAMP
2	0456	NOP	18	00	262
3	0455	NOP	18	00	258
4	0454	NOP	18	00	254
5	0453	NOP	18	00	250
6	039C	RETI	18	07	246
7	0390	JNC 039Ch	18	00	236
8	038E	INC [0006h]	18	00	231
9	038A	JZ 038Eh	18	00	224
10	0387	TST [0001h], FFh	18	00	219
11	0383	JZ 0387h	18	00	211
12	0380	TST [0002h], FFh	18	00	206

追踪缓冲区的大小为 256 KB，在追踪模式 1 下，可追踪 128 K 的指令，在追踪模式 2 和 3 下，则可追踪 32 K 的指令。

可以通过 **Debug（调试）->Windows（窗口）->Trace（追踪）**菜单打开追踪窗口。可以使用 **Debug（调试）->Trace Mode（追踪模式）**菜单修改追踪模式。打开追踪窗口后，在程序暂停时，追踪窗口将会加载最近 64 条追踪条目。要再加载 64 个条目，可使用“Get Next Trace Data”（获取下一追踪数据）工具栏或菜单选项。此外，也可以使用“Get all Trace Data”（获取所有追踪数据）菜单选项，加载所有可用的追踪数据。

事件查看器

您可以使用事件查看器定义条件或者条件序列，在满足这些条件时，触发断点或追踪。这样，您便可以检查并调试更为复杂的代码序列，而使用标准调试工具则无法实现这种功能。

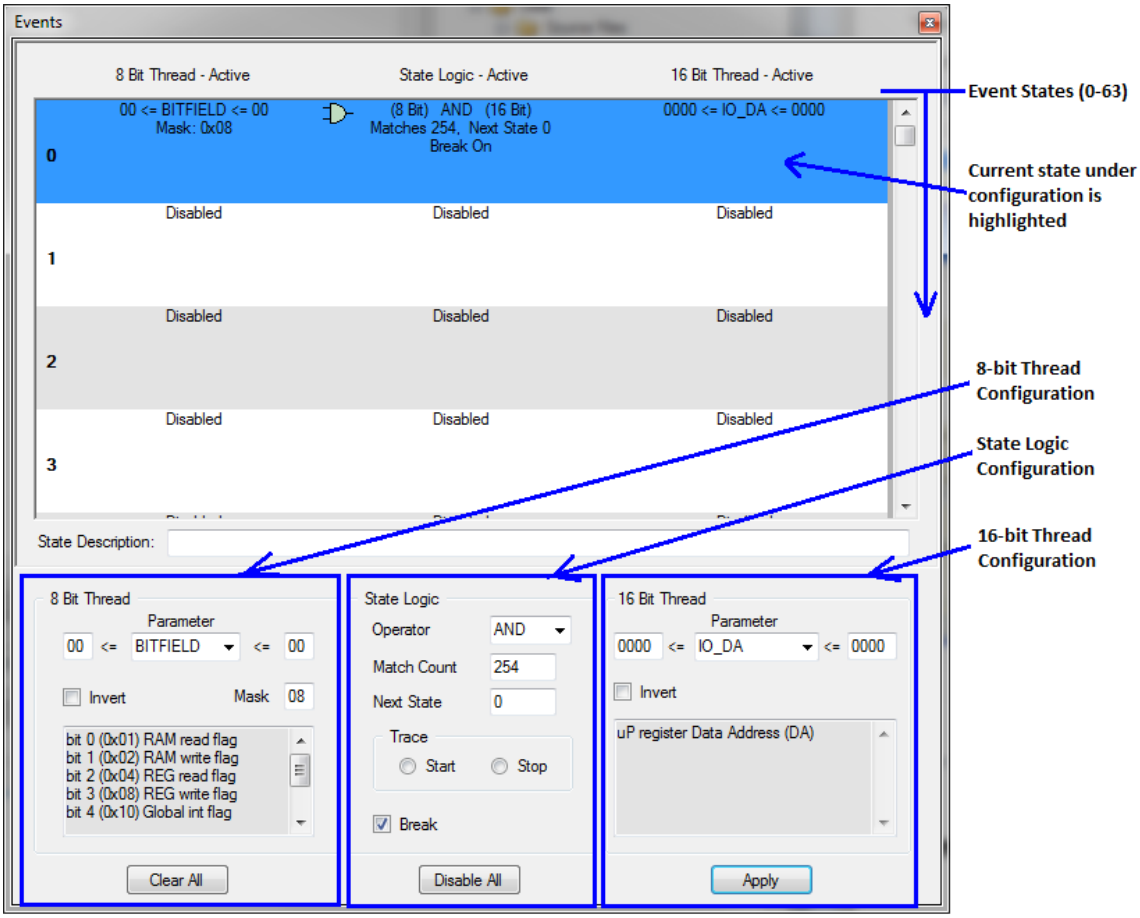
可以对事件进行配置，使其适用于多种调试情况。设置一个事件相当于编写一个标准的 if-then 语句。首先，进行比较。如果比较经计算为真，则执行一项操作。这个操作可以是设置断点、启动/停止跟踪缓冲区或者继续运行至另一 if-then 语句。您最多可定义 64 条语句。

“if”语句由 8 位或者 16 位线程组成。每个线程可以对若干个可用的 8 位或 16 位源进行比较，例如 RAM 地址、PC 值或者堆栈值。

要增加“if”语句的灵活性，可以独立使用 8 位和 16 位比较或者通过逻辑函数（OR、AND、NOR、NAND）进行组合。此外，您还可以反转每个 8 位或者 16 位线程的逻辑输出，以便更好地帮助检测低电平有效数据或已清空的数据。

Events（事件）调试窗口如第 11 页图 **20** 所示，可以在 PSoC Designer 中通过 **Debug（调试）-> Windows（窗口）-> Events（事件）**菜单访问。该窗口中的一部分可用于选择和显示全部 64 条可用事件的状态。要配置某一事件状态，只需在窗口的上半部分选择该事件，然后在下半部分的状态设置中设置配置参数即可。每种有效设置将于后面的章节详细介绍。

图 20. Events（事件）调试窗口



下面将说明可用的事件 8 位线程、16 位线程、逻辑比较和输出选项。

8 Bit Thread（8 位线程）选项

8 位线程提供了几种对 PSoC 内部宽 8 位参数进行计算的方法。可以针对具体数值或介于 0x00 到 0xFF 的数值范围进行比较。

*Invert（取反）：*可以使用该选项反转 8 位参数的比较结果。

Mask（掩码）： 通过掩码字段可以在参数比较之前对输入数据的指定位进行屏蔽。如果您仅关注所选择参数值的某些具体位，则此选项将非常有用。

*Parameter（参数）：*您可以通过参数下拉菜单选择用于 8 位比较的字段。可用的 8 位参数比较选项如下所示。

*NONE：*禁用 8 位线程比较。

A： 累加器比较。计算累加器中的数据。

*IO_DA：*寄存器地址比较。计算寄存器存储器空间中的数据地址。

IO_DB： 寄存器数据比较。计算寄存器存储器空间中数据总线上的数据。

*IR：*指令寄存器比较。计算指令寄存器中的指令操作码。

MEM_DA： RAM 地址比较。计算 RAM 存储器空间中的一个地址。

*MEM_DB：*RAM 数据比较。计算 RAM 存储器空间中数据总线上的数据。

*SP：*堆栈指针比较。计算堆栈指针当前的地址（并非 SP 指向的数据）。

*X：*X 寄存器比较。计算 X 寄存器当前的值。

BITFIELD： 允许对芯片内部操作进行具体的标志检查，例如发生写入或读取操作时，或置位了零标志或进位位标志时。可用的 BITFIELD 检查如下。可以同时检查多个标志。

位 0 (0x01)： RAM 读取标志。

位 1 (0x02)： RAM 写入标志。

*位 2 (0x04)：*寄存器读取标志。

位 3 (0x08)： 寄存器写入标志。

位 4 (0x10)： 全局中断标志。

*位 5 (0x20)：*零标志。

位 6 (0x40)： 进位位标志。

位 7 (0x80)： 扩展 IO 标志。

BITFIELD 比较选项通常会与一个附加比较（或一序列比较）结合使用。例如，位 1（RAM 写入标志）通常与 RAM 地址和/或 RAM 数据比较（16 位线程）结合使用，以了解特定的值在何时写入某个 RAM 位置。RAM 地址和 RAM 数据比较会自行执行这些任务，在初始写入操作发生后，此类比较仍会继续直到为真（根据调试情况有些比较可能无需进行）。

BITFIELD 中的位为低电平有效。要检查 BITFIELD 中的具体位，可以进行以下操作：

- 使用 Mask（掩码）字段检查您关注的位。例如，要检查零标志，可将 Mask（掩码）值设为 0x20。这样就可以屏蔽不需要的位，因此，参数比较将不会对这些位进行计算。
- 将参数比较的高低数值均设为 0。由于掩码已经筛选了关注位，所以不需要进行参数比较。如果比较的高低值均为 00，则所有通过掩码的位都将进行比较计算（因为 BITFIELD 位为低电平有效）。

要获取 BITFIELD 比较参数的具体示例，请参考示例 1：具体地址的写入中断和示例 2：具体值和地址的写入中断。

16 Bit Thread（16 位线程）选项

16 位线程提供了几种对 PSoC 内部宽 16 位参数进行计算的方法。可以针对具体数值或介于 0x0000 到 0xFFFF 的数值范围进行比较。请注意，16 位线程中的有些可用比较选项只有 8 位宽，针对这种选项，将对 0x00 到 0xFF 之间的数值进行计算。

*Invert（取反）：*可以使用该选项反转 16 位参数的比较结果。

Parameter（参数）： 参数下拉菜单提供了很多 8 位线程中可用的比较，此外，还含有对若干附加 16 位宽参数进行比较的选项。一些 16 位比较允许在一次比较中同时检查两个 8 位参数。可用的 16-bit 位比较选项如下所示。

*NONE：*选中该选参数将禁用 16 位线程比较。

A[1]*：*累加器比较所提供的功能与 8 位线程版本的功能相同。

IO_DA[1]*：* 寄存器地址比较的功能与 8 位线程版本的功能相同。

*IO_DA_DB：*结合了寄存器地址和数据比较。输入的高 8 位有效位将用于寄存器地址比较（类似于 IO_DA）；输入的低 8 位有效位则用于寄存器数据比较（类似于 IO_DB）。

IO_DB[1]*：*寄存器数据比较的功能与 8 位线程版本的功能相同。

IR[1]*：*指令寄存器比较所提供的功能与 8 位线程版本的功能相同。

*IR_SP：*结合了指令寄存器和堆栈指针比较。高 8 位有效位执行一次指令寄存器比较；低 8 位有效位则执行一次堆栈指针比较。

MEM_DA[1]*：*RAM 地址比较的功能与 8 位线程版本的功能相同。

*MEM_DA_DB：*结合了 RAM 地址和 RAM 数据比较。高 8 位有效位执行一次 RAM 地址比较（类似于 MEM_DA）；低 8 位有效位则执行一次 RAM 数据比较（类似于 MEM_DB）。

MEM_DB[1]*：* RAM 数据比较的功能与 8 位线程版本的功能相同。

*PC16：*程序计数器比较。使用该参数，您将可以计算程序计数器的当前值。

SP[1]*：*堆栈指针比较所提供的功能与 8 位线程版本的功能相同。

X[1]*：*X 寄存器比较的功能与 8 位线程版本的功能相同。

*X_A：*结合了 X 寄存器和累加器比较。高 8 位有效位执行一次 X 寄存器比较；低 8 位有效位则执行一次累加器比较。

[1] *在进行 A、IO_DA、IO_DB、IR、MEM_DA、MEM_DB、SP 和 X 比较时，仅计算 16 位宽输入比较字段中的低 8 位有效位。对于超过 8 位的 RAM 或寄存器地址比较，将不会自动检查分页指针寄存器。*

State Logic（状态逻辑）

Operator（运算符）： 如果 8 位线程和 16 位线程同时启用，则运算字段可将两者按照一定的逻辑结合起来。可用的逻辑运算符为 AND、OR、NAND、NOR。如果 8 位线程或 16 位被禁用，则将忽略运算符。

*Match Count（匹配计数）：*用户可通过匹配计数，指定在启用追踪或断点或移至其他事件状态前，事件应发生的次数。匹配计数值可介于 1 至 32,767 之间。如果需要超过 32,767 次匹配，则可以将多个事件状态链接起来。

*Next State（下一状态）：*通过此参数可以将多个事件状态链接起来，创建更为复杂灵活的事件序列。如果符合当前事件状态，达到指定的匹配次数，将会执行下一状态。如果不想创建状态链，则下一状态应设置为当前状态编号。

并非同时计算所有状态。始终先对状态 0 进行计算。如果状态 0 禁用或计算为假，则将不会对其余状态进行计算。

因此，所有事件或事件链必须以状态 0 开始，即使不需要链接亦是如此。

Trace（追踪）：如果事件状态输出和设置相匹配，则可能启动或停止追踪缓冲区。如果希望仅追踪代码的某个部分，该选项将非常实用。

Break（断点）：通过断点复选框，您可以指定在满足事件条件时，是否允许事件触发一个断点。

事件注释：

- 触发事件条件中断后，PSoC Designer 会在事件发生后暂停两条指令周期的时间。暂停的原因是，事件参数的 ICE 计算比实际的转接板执行滞后两个指令周期。有时，在 C 语言调试中，这一现象并不明显，因为执行每行 C 代码可能需要多行汇编代码。
- 在事件窗口中做出的修改不能应用到正在运行的器件中。对一个事件或者一系列事件做出修改时，必须在器件暂停时单击应用按钮，以确保 ICE 识别更新后的事件。
- 使用 8 位线程中的 BITFIELD 比较参数时，应该完成以下两项任务：将比较的高低值均设置为 0x00，并使用 Mask（掩码）字段选中恰当的 BITFIELD 参数。要获取 BITFIELD 参数的设置和使用示例，请参考示例 1：具体地址的写入中断。
- 在某个事件配置判断为真并触发了一个断点后，配置会继续计算为真（并继续触发断点）直至芯片状态变更为不会引发事件判断为真的状态。例如，如果将一个事件配置为在将具体值加载至累加器时暂停，但是此条件后的指令并不修改累加器，那么在累加器变更前，中断将会继续发生（每次 1 条指令）。

事件常见用法：

事件适用于很多调试情况，但是其可能常见的用法包括：

- 查找何时寄存器或 RAM 地址读取或写入（参见下方事件示例 1 和示例 2）。
- 检查堆栈溢出（参见下方示例 3）。
- 检测一个跳转或者一次函数调出。
- 追踪具体的一段代码。
- 测量中断延迟。
- 在一行代码执行到第“n”次中断（匹配计数，参见下方示例 4）。
- 进位位标志状态中断。
- 等待一确定的指令集。
- 累加器中的具体数据匹配中断。

以下将详细说明如何为几种常见的用法实例设置并使用事件工具。

示例 1：具体地址的写入中断

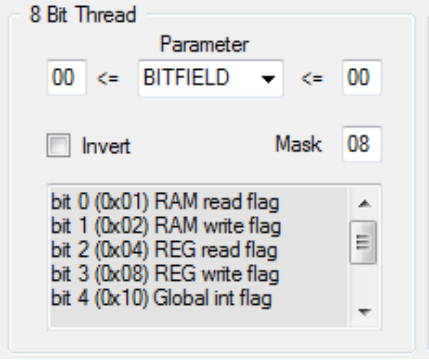
在某些情况下，了解具体地址的写入时间很有价值。

例如，I2C 从器件出现了间歇式问题，主控会周期性地在从器件中读取一个不正确的值。您可能无法知道何时 I2C 数据寄存器加载这一不正确的值。这时，可以在 I2C 输出数据寄存器发生写入操作时使用事件触发中断。

要实现这一操作，可以使用 8 位线程检查寄存器地址写入操作的发生时间。这项检查还可以结合一个检查 I2C 数据寄存器地址的 16 位线程。当该地址发生写入操作时，会产生一个断点。

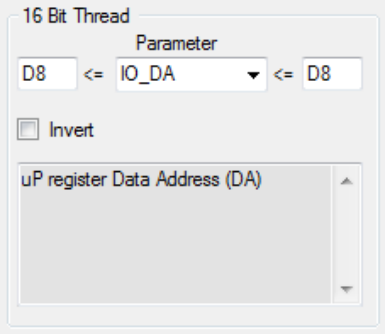
要产生对具体地址的写入操作的中断，需配置以下参数：

1. 设置 8 位线程的各项参数。



   a. Parameter（参数）= “BITFIELD”。通过这一设置，您可以进行一系列检查，包括在本示例中用到的“寄存器写入”检查。
   b. 由于 BITFIELD 为低电平有效，将参数比较的高低值均设置为“00”，这样将会检查所有位。Mask（掩码）值用于检查关注的具体位。
   c. 将掩码值设置为 0x08，这样，只要执行寄存器写入操作，BITFIELD 比较便会计算为真。

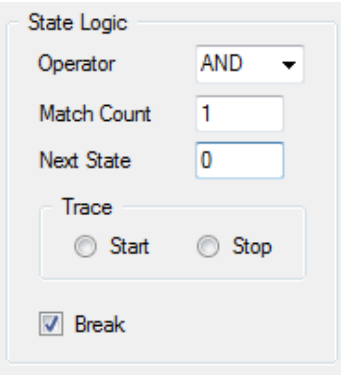
2. 设置 16 位线程的各项参数。



   a. Parameter（参数）= “IO_DA”。通过这一设置，在具体寄存器地址位于总线时，将发生事件。

b. 将比较的高低值均设为“D8”。这是 I2C 模块的数据寄存器。可以在用户模块数据手册或者技术参考手册中找到此寄存器或任意其他模块的寄存器信息。

3. 设置状态逻辑的各字段。



a. 将 Operator（运算符）字段设置为“AND”。此字段将配置事件状态，因此，仅在 8 位线程与 16 位线程所定义的条件同时满足后，事件才会计算为真。在这种情况下，事件将仅在 I2C 数据寄存器发生写入操作时才会计算为真（读取操作将被忽略）。

b. 将 Match Count（匹配计数）设置为 1。该事件仅需发生一次，就会触发断点。

c. 将 Next State（下一状态）设置为 0。在这种情况下，只需一个事件状态即可对所需条件进行计算，因此，不需要状态链。

d. 不需配置 Trace（追踪）中的“Start”（启动）和“Stop”（停止）选项。在本示例中未进行追踪。

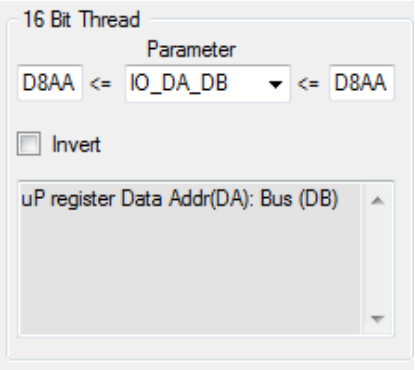
e. 选中“Break”（中断）复选框。这样，只要为整个事件状态定义的条件计算为真时，即会产生断点。

示例 2：具体值和地址的写入中断

为了扩展事件示例 1，设计者可能还希望了解具体值何时写入寄存器。示例 1 中所示的事件会在每次有值写入 I2C 数据寄存器时中断，但是，仅在错误数值写入寄存器时才产生中断，可能会更为实用。

要增加这一功能，必须对 16 位线程稍作修改。事件状态的其他部分则应保持不变，如示例 1 所指定。

1. 设置 16 位线程的各项参数。



a. Parameter（参数）=“IO_DA_DB”。通过这项设置，在具体值写入总线上某一指定寄存器地址时，将发生事件。

将比较的高低值均设为“D8AA”。这样，只要 0xAA 写入 I2C 数据寄存器 (0xD8) 就会发生中断。请注意，通过修改比较的高低位值，可以检查某个范围内的数值，而不仅仅是 0xAA。

示例 3：堆栈溢出

事件查看器的另一有效应用时机是检查堆栈溢出，堆栈溢出可能引发意外行为，有时还会增加调试的困难程度。但所幸，使用事件查看器，可轻松检测堆栈溢出，监控堆栈使用情况。

要使用事件对程序的堆栈溢出进行监控，必须配置一个 8 位线程，用于监视堆栈指针。

要在堆栈溢出将要发生前产生中断，应配置以下参数。

1. 设置 8 位线程的各项参数：

   a. Parameter（参数）=“SP”。通过这一设置，可以监控堆栈指针值。

   b. 将低比较值设为 FD，高比较值设为 FF。这样，当堆栈指针达到 FD 时，将触发中断。

   c. 将 Mask（掩码）值设置为 FF。该设置不会屏蔽任何位。

2. 将 16 位参数值设为 NONE，以确保禁用 16 位线程。

3. 设置状态逻辑的各字段。

   a. 运算符的选择并不重要，因为已禁用 16 位线程。在禁用 16 位线程的情况下，会忽略运算符选择。

   b. 将 Match Count（匹配计数）设置为 1。这样，事件只需发生一次，即会触发断点。

   c. 将 Next State（下一状态）设置为 0。在这种情况下，只需一个事件状态即可对所需条件进行计算，因此，不需要状态链。

   d. 无需配置 Trace（追踪）中的“Start”（启动）和“Stop”（停止）选项。本示例未进行追踪，但是在尝试确定堆栈溢出的原因时，追踪功能非常实用。

e. 选中“Break”（中断）复选框。这样，只要为整个事件状态定义的条件计算为真时，即会产生断点。

关于堆栈溢出的更多信息，包括不用调试器检查堆栈溢出的方法以及避免堆栈溢出的技巧，请参考附录 B – 堆栈溢出。

示例 4：事件发生 X 次后中断

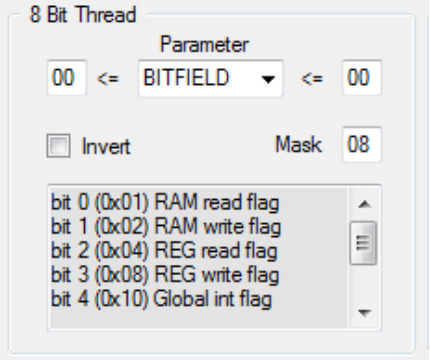
这一示例将描述如何在事件条件已多次计算为真后设置一个断点。这一操作能够帮助开发人员更为精确地控制程序何时停止执行。

在本示例中，事件查看器设置为在第 254 次进入以下代码中的 for 循环时产生中断：

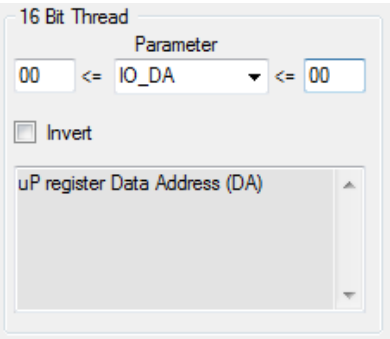
```
BYTE x;
for(x = 0; x < 255; x++)
{
   PRT0DR ^= 0x01;
}
```

这一事件状态的配置与示例 1 中的状态类似。BITFIELD 参数用于检测寄存器写入。该参数与检查 PRT0DR 寄存器的 16 位线程配合实用，在每次进行循环时，递增事件计数。要配置这一事件状态，需对事件做出以下变更：

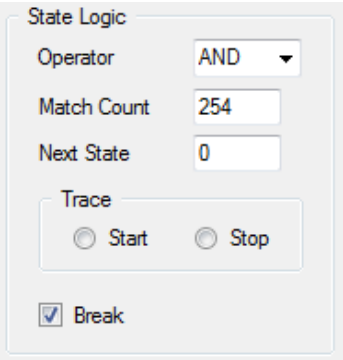
1. 设置 8 位线程的各项参数。这些设置与示例 1 所用设置相同。请参考示例 1 了解每个设置的更多详细描述。



2. 设置 16 位线程的各项参数。这些设置类似于示例 1 中的设置，仅需将地址比较修改为 00（PRT0DR 地址）。



3. 设置状态逻辑的各字段。与示例 1 的唯一区别是 Match Count（匹配计数）的值变更为“254”。这样，在循环了 254 次之后，事件会评估为真并触发一个断点。



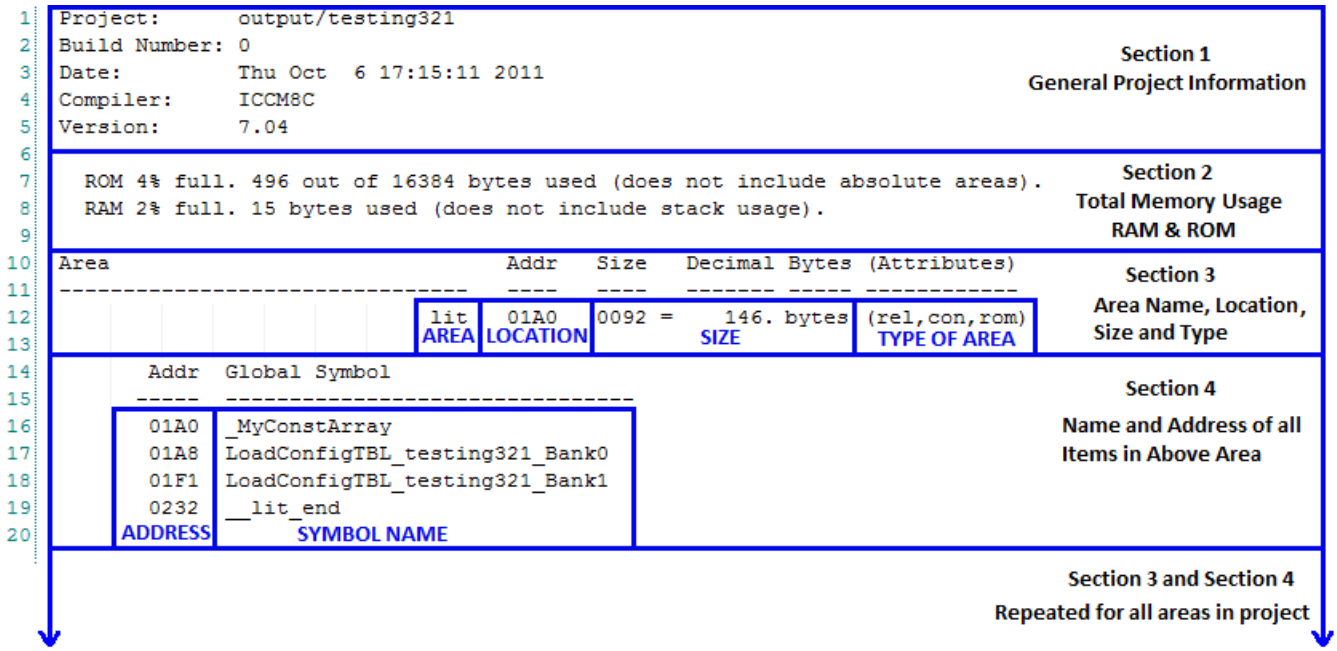
映射文件 (.mp)

映射文件在程序构建过程中生成，包含全局符号地址以及内存中所定义之区域的信息。映射文件是一个实用的调试工具，可用于确定变量或者一段代码在内存中的位置或者某串代码所占用的存储器空间。

映射文件中将不会显示变量、函数或标签。

映射文件的各个部分如图 21 所示。图中的布局和描述均针对于 ImageCraft C 编译器和 v7.0.4 连接器，编译器版本如有不同，映射文件也可能略有差异。映射文件位于 PSoC Designer Workspace Explorer（工作区浏览器）的“Output Files”（输出文件）文件夹中。

图 21. 映射文件及相关注释



第 1 部分：基本信息

映射文件的开头（1-5 行）列出了项目的基本信息，包括项目名称、日期、最后构建的时间记录以及编译器版本。

第 2 部分：总内存的使用情况

映射文件的下一部分（7-8 行）列出了 ROM 使用总量（减去绝对区域）以及 RAM 大小（减去堆栈使用量）。ROM 中的绝对区域不能通过连接器计算，因为这些区域通常由用户（使用具体命令）放置，而非由连接器确定。在运行期间，堆栈大小会发生改变，并且不能在构建过程中通过连接器计算，因此在计算 RAM 时将不考虑堆栈使用量。但是，报告的 RAM 大小能够帮助说明剩余多少 RAM 可供堆栈使用或添加其他变量。

第 3 部分：区域名称、位置、大小和类型

第 3 部分（10-13 行）是下一内存区域的头。这一部分显示的是区域名称、位置（内存地址）、大小和区域类型。

区域名称： 区域名称在声明区域时进行分配。区域是内存的一部分，编译器用其存放代码和数据。用户可以指定自己的内存区域，但是 ImageCraft 编译器会使用几个预定义的区域。

top：包含中断向量和 boot.asm 代码。这一区域位于 ROM 的地址 0。

lit：包含常量数据，位于 ROM。

idata：储存全局数据的初始化数值。

text：包含用户代码。

psoc_config：包含用于配置器件的函数（以及用于对器件进行动态重新配置的函数）。

usermodules(用户模块)：包含用户模块 API 程序。

区域位置： 此位置为区域起始地址。对于在 ROM 中定义的区域，这是一个 ROM 地址；对于在 RAM 中定义的区域，这是一个 RAM 地址。

区域大小： 这是区域的总大小，提供十六进制和十进制两种数值。区域末尾为区域起始位置加上区域大小。

区域类型： 这是给定区域的已定义属性。这些属性定义了区域中存放的数据类型，以及连接器对该区域的放置方法。以下为可用于定义区域类型的有效关键字完整列表：

REL – 可重定位区域。这一关键字允许连接器将区域重新定位至其选择的地址。可重定位区域没有预定义的位置，其地址在构建连接阶段确定。

ABS – 绝对区域，或不可重定位区域。这一关键字的含义是，区域地址已在代码中定义，且不能被连接器移动。

区域可以是可重定位的，或是绝对的，但不能兼具两种属性。

CON – 联结区域。这一关键字表明在内存中各区域将顺序相连。

OVR – 覆盖区域。这一关键字表明多个覆盖区域可以具有相同的起始地址，并可以占用相同的内存空间。它仅适用于 RAM 区域。

区域可以是联结的，也可以是覆盖的，但不能兼具两种属性。

RAM – 指明数据存储在 RAM 中，仅用于变量存储。

ROM – 指明数据存储在闪存中。代码和常量数据均可存储在 ROM 区域中。

区域可以是 RAM，也可以是 ROM，但不能兼具两种属性。

在本示例中，所示区域为“lit”，即文字数据区域。这一区域存储项目中使用的所有常量数据。总大小为 146 字节，区域类型“rel、con、rom”指程序 (ROM) 内存空间中可重定位的联结区域。

第 4 部分：所有符号的名称和地址。

映射文件的这一部分描述了第 3 部分定义和说明的区域中的内容。此部分依次列出了区域中每个符号的地址和名称。由于所有符号均是依次排列的，因此，每个符号的大小均可通过比较给定符号与下一符号的地址而确定。

列出的地址与给定区域的内存类型相对应（对于 RAM 区域，该地址将为 RAM 地址；对于 ROM 区域，该地址将为 ROM 地址）。

在此区域中，并非所有符号都会占用内存。全局符号标签亦出现在列表中。例如，图 21 中的“__lit_end”符号仅用于指明 lit 区域结束的地址（地址 0x0232），而不占用任何物理内存。

列表文件 (.lst)

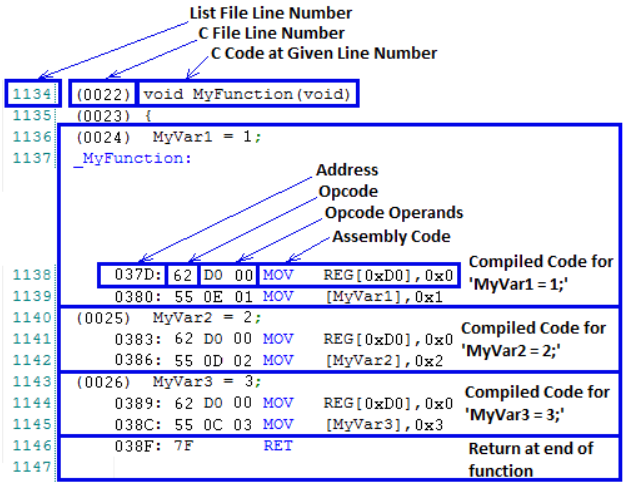
PSoC Designer 生成的另一有用文档为列表 (.lst) 文件，包含 C 编译器生成的汇编代码。在调试函数未按预期运算或者尝试用一段 C 代码与一段汇编代码混编时，请参考列表文件。调试器的“Step ASM”（单步 ASM）功能也可以引导用户转至列表文件。

图 22 列出了列表文件中的一段，并重点标识出了关键信息。对于 C 语言源文件中的每一行，列表文件将重新列出了该行 C 代码，并依次列出给定行所需的所有已编译汇编代码。

列表文件的组织方式是从闪存的 0x00 地址开始，按顺序列出所有已编译代码，直至内存存满。因此，在较大的项目中，列表文件会很长。使用搜索功能可以轻松地在大型列表文件中找到某一函数或一段代码。

每行汇编代码还会列出指令的代码地址和具体操作码以及汇编指令操作码的操作数。关于汇编操作码的更多信息，请参考汇编语言用户指南。

图 22. 列表文件及相关注释



选取的这段列表文件展示了函数“MyFunction”，这一函数将常量值赋给三个全局变量（MyVar1、MyVar2、MyVar3）。如图所示，C 语言的每个赋值操作均被编译成两行汇编代码；一行设置分页寄存器，一行为变量赋值。

PSoC 1 调试提示和技巧

本部分记录了调试 PSoC 1 器件时需要注意的一些技巧。

在“停止”期间硬件仍然运行

ICE 可以模仿并控制 PSoC 器件内部的代码执行状态，但不能对器件的数字和模拟部分进行显式控制。因此，在调试器暂停时，其他硬件元件仍将继续运行。调试器在某一断点处所报告的硬件值是在暂停时刻所有寄存器数值的快照。

在处理计数器和定时器时，或者在处理使用计数器和定时器的用户模块（例如一些 ADC 用户模块）时，这种现象尤为明显。在调试器暂停了 CPU 的执行后，计数器仍将继续运行。因此，在程序恢复调试后，定时器/计数器/PWM（以及使用计数器的任意 ADC）报告的第一个值可能是不正确的。在报告了第一个错误值后，定时器/计数器/PWM（以及使用定时器的任意 ADC）会重新进行同步，并开始报告正确数值。

如果调试取决于 ADC 读数的精度，则可以应用以下两种解决方法确保调试器报告的 ADC 数值准确无误：

1. 在程序运行时，左键单击代码窗口左侧边缘（行号左侧），在读取 ADC 结果的指令后面动态设定断

点。 这样，程序暂停时，从 ADC 中读取的值将是正确的。 要获得另一读数，可清除断点、运行程序，并重新动态放置断点。

2. 如果所选的 ADC 支持单独采样模式（例如增量型 ADC ），那么还可以使用另一种方法。在单独采样模式下运行 ADC，而不以连续采样模式运行。在单独采样模式下，每次完成转换后，计数器和定时器都会停止，这样便可确保 ADC 返回的数值正确（前提是转换开始至结果返回期间无断点）。请注意，这种方法并不适用于基于 Delta-sigma 的 ADC。

闪存写入调试

在开发一个包含内部闪存写入的项目时，可以使用 ICE 和调试器。比如一个引导加载程序或其他将数据保存至闪存的应用。在使用 ICE 调试器包含闪存写入程序时，请注意以下几点：

- 要允许 ICE 仿真闪存写入，则要写入的闪存模块必须处于无保护状态。无保护闪存是允许外部器件写入闪存的唯一安全设置。虽然 ICE 对外部闪存写入进行仿真，但从技术角度来讲，ICE 仍然是 PSoC 器件内部请求闪存更新的外部器件。
- 与独立 PSoC 器件的闪存写入相比，使用 ICE 仿真闪存写入的耗时将显著增加。在闪存写入期间，ICE 和 PSoC 器件之间需要进行的通信活动会耗费一定时间。ICE 的写入次数各不相同，但通常为 PSoC 器件数据手册中定义的典型写入次数的 10-20 倍。
- 在不使用 m8ssc.inc（所有的 PSoC Designer 项目均还有此文件）中的宏 SSC_Action (OpCode) 的情况下，请不要尝试执行系统管理调用程序 (SSC) 指令。这些指令导致闪存写入操作或其他 SSC 指令必须遵循特定的顺序，这样，ICE 才能检测并仿真 SSC 指令。SSC_Action 宏执行正确的指令序列来使 SSC 操作期间启用仿真功能。用户模块及赛普拉斯提供的库（例如 Flashblock 写入库）均使用了正确的 SSC 宏。
- E2PROM 用户模块和 Flashblock API 库闪存写入例程可作为预编译库供用户使用。在调试过程中，这部分源代码不予提供且调试器的“Step Into”（单步执行）功能，亦不可用。但是，在执行“Step Over”（单步跳过）时，函数仍会正确执行。

在睡眠模式下使用调试器

在调试使用睡眠功能的项目时，请注意以下几点：

- 不能在 M8C_Sleep 指令的下一行放置断点。在进入睡眠状态之前，睡眠操作会预提取下一条指令。这样，如果在睡眠指令后立即触发断点，则 ICE 便与 PSoC 不再同步并会断开。
- 调试器不允许单步执行或单步跳过睡眠指令；这么做会使 ICE 与器件断开。要避免这种情况，请在睡眠指令后间隔两行再设置断点，并允许调试器自由运行至断点，而不是精确地单步跳过睡眠指令。
- 如果在手动暂停调试器时，连接的 PSoC 1 器件正处于睡眠状态，则 ICE 将会断开。要避免这种情况，使用睡眠功能的项目应始终通过断点或者事件暂停正在运行的程序，而不是直接单击暂停命令。

其他调试方法

本部分将介绍其他的调试工具和技巧。这些技巧和方法没有标准调试设置的功能集，但在标准 ICE 调试设置不可用或不适用的情况下，也是极具成本效益的选择。此外，在需要实时调试支持的情况下，这些方法非常适用，可以不用物理停止器件运行而进行调试。

使用 I2C-USB 桥接器进行调试

I2C-USB 桥接器可以将计算机的 USB 接口桥接至 PSoC 的 I2C 用户模块，以用于测试、调整和调试 PSoC 应用中的硬件和软件。桥接器的作用相当于一个 I2C 主机，通过 USB 接口向总线上的任意 I2C 从器件发出命令。

赛普拉斯提供两种类型的 I2C-USB 桥接器：

- CY8CKIT-002 PSoC MiniProg3 编程和调试套件。MiniProg3 具备内置的 I2C-USB 桥接器功能，可以对所有 PSoC 1、3、和 5 器件进行编程。
- CY3240 I2C-USB 桥接器套件。CY3240 套件支持 I2C 至 USB 的桥接，但不提供任何编程功能。赛普拉斯软件工具仍支持这一套件，但是新用户最好选购 CY8CKIT-002 MiniProg3 套件。MiniProg3 支持 I2C 至 USB 的桥接，是 CY3240 所有功能的超级集合。

MiniProg3 和 CY3240 套件如图 23 所示。

图 23. MiniProg3 和 CY3240 I2C-USB 桥接器

在调试实时应用时，或在 ICE/OCD 设置不可用的情况下，I2C-USB 桥接器非常实用。在这种方式下，ICE 和支持 OCD 的 PSoC 均不需要进行调试。

典型的用法实例中含有以下工具：

- 一个 I2C-USB 桥接器。建议采用 MiniProg3 解决方案，亦可使用 CY3240。
- 目标 PSoC 器件，带有一个运行的 EZI2C 从设备用户模块，连接至 P1[5] 和 P1[7] 或 P1[0] 和 P1[1]。
- 运行桥接器控制面板软件的计算机，用于向桥接器发出指令。

之后，I2C-USB 桥接器即可作为 I2C 主器件，连接到主机计算机和目标 PSoC 器件之间。桥接器控制面板用于周期性查询 PSoC 器件以获取信息。轮询率和传递数据量完全可配置，能够帮助您查看正在运行的 PSoC 器件上的实时调试信息。

常见 I2C-USB 桥接器调试用法实例：

- 传输实时 Capsense 按钮调校数据。桥接器控制面板亦可支持输入 I2C 数据的实时图样化。在该用法实例中，PSoC 程序定期将按钮信息加载至 I2C 阵列，而 I2C 桥接器则周期性地查询此类数据。
- 监视预定义“监视”变量集。在此用法实例中，PSoC 程序会定期将所关注的任意变量复制到 I2C 阵列，而 I2C 桥接器则周期性地查询并监视此类数据。
- 测试 I2C 从器件的功能。I2C-USB 桥接器的作用相当于一个 I2C 主控，可以测试总线上任意 I2C 从器件的功能。
- 测试基于 I2C 引导加载程序的功能。I2C-USB 桥接器常用于与基于 I2C 的引导加载程序进行通信。PSoC Designer 可支持生成下载文件，桥接器控制面板可以直接使用这些文件进行 I2C 引导加载操作。

要获取桥接器套件的更多文档以及说明其用法的相关文档，请点击以下链接：

CY8CKIT-002 PSoC MiniProg3 编程和调试

CY3240 I2C-USB 桥接器

使用 UART 接口调试

使用 UART 接口进行调试是针对嵌入式处理器的常用策略。所幸，向 PSoC 1 器件添加 UART 接口十分容易，很多 PSoC 1 开发工具包均带有可供使用的 RS-232 收发器。

此外，您还可以在几乎所有的计算机上查看 UART 器件发送的信息。串行端口已逐渐变得不再常用，但 USB-UART 桥接器件则非常普遍。用于查看数据的计算机软件亦被广泛地使用，例如 Hyperterminal(超级终端)。

UART 调试方法的常见用法实例类似于上文列出的 I2C-USB 桥接器用法实例。UART 用户模块提供的 API 让 PSoC 器件的调试信息传入和传出变得更加简单。可供选择的 API 包括 PutString、PutChar、GetString 和 GetChar。

关于在 PSoC 1 上配置和使用 UART 的更多信息，请参考 PSoC Designer 中提供的 UART 用户模块数据手册。

引脚触发

引脚触发虽然功能基础，但仍是一种很好的调试手法，特别适用于需要实时调试的情况。引脚触发通常用于以下情况：

- 用两个引脚触发所关注的一段代码，以便计算该代码运行时长。这一技巧常用于为中断服务例程 (ISR) 的执行时间计时。
- 检查是否正在运行某段代码。此任务还可通过断点完成，但有时，使用这种方法有助于在不干扰代码执行的情况下完成检查。

故障排除

本部分将列出在使用 PSoC 1 调试器时会遇到的常见问题，并将讨论可能的解决方案。

问题	可能的解决方案
ICE 无法检测或连接，或与转接板不兼容	转接板与项目所选的器件不匹配。每种转接板都支持其所属系列中的一些器件；请参考 PSoC 器件数据手册，以确保正在使用的转接板或者 OCD 部件与项目所选的具体部件型号兼容。
	请确保 PSoC 器件已正常供电。PSoC 可能通过外部供电或者通过 ICE 供电。Project（项目）->Settings（设置）->Debugger（调试器）设置（如图 12 所示）中所选供电方法必须与实际供电方法相匹配。
	ICE 未供电。应通过 12-V/1-A 直流电源为 ICE 供电。如果使用的是较低电压的电源，则 ICE 可能会亮起，但不会正常运行。
	转接板可能未正确连接。请参考 调试硬件环境建立部分，了解常见调试硬件配置。
	请确保只有一个 PSoC Designer 或者 PSoC Programmer 实例已开启并已连接至目标器件。
	可能将过期版本的转接板连接至 ICE。请确保使用现行版本的转接板。

ICE 在调试会话期间断开连接	ICE 可能未正常供电。请确保已使用 12V/1A 直流电源为 ICE 供电。如果使用电压电流不符合规格的电源供电，器件也许可以连接，但在调试开始后可能会失败。
	请检查代码中是否存在闪存写入或者其他 SSC 指令。如果有，请参考闪存写入调试部分，了解如何使用包含闪存写入程序的 ICE 调试器。
	请检查代码中是否存在任何睡眠操作。如果有，请参考在睡眠模式下使用调试器部分，了解如何在正确使用 ICE 在调用了睡眠功能的项目中。
	请避免在调试过程中使用锁相环 (PLL)。要防止这一问题的产生，请在调试时关闭锁相环，并将所有外部晶振硬件从 PSoC 上断开。
	请避免在调试过程中使用外部晶体振荡器 (ECO)。要防止这一问题的产生，请在调试时关闭 ECO，并启动内部主振荡器 (IMO)。
	请确认计算机、ICE 和 PSoC 之间已正确紧密相连。请参考调试硬件环境建立部分，了解常用的调试硬件设置。
调试会话期间产生无效参考内存	无效参考内存 (IMR) 产生的原因与导致 ICE 在调试会话期间断开的原因相同。请参见“ICE 在调试会话期间断开连接”部分，了解解决这一问题的建议。
无法找到选定的 ICE 端口	USB 线缆可能从 ICE 端或者计算机端脱落。请确保 USB 线缆正确连接，并等候几秒钟再尝试重新连接。
	ICE 可能未正常供电。请确保已使用 12V/1A 直流电源为其供电。
	请确保只有一个 PSoC Designer 或者 PSoC Programmer 实例已开启并已连接至目标器件。
项目执行于非预期位置暂停	可能触发了欠压检测 (LVD)，这一操作会在 boot.asm （LVD 中断矢量）的地址 0x04 处暂停程序，或在 LVD 中断服务例程（如果已定义）中暂停程序。如果是这种情况： - 请检查 PSoC 供电状况。如果 PSoC 的电压波动至低于全局参数设置中的 LVD 激发电压，则会发生 LVD 中断。 - 请检查并确保 ICE 正常供电。应使用 12-V/1-A 直流电源为 ICE 供电，使用其他供电方式可能会引发意外行为。 - 如果 ICE 为目标器件或电路板供电，则请确保目标器件或电路板的所需电流小于 250 mA。
	请确保未发生堆栈溢出。如果在函数调用期间，加载至堆栈的程序计数器 (PC) 值被覆盖，则堆栈溢出会干扰 PC。在这种情况下，程序将会执行至代码中的非预期位置。请参考本应用笔记的附录 B – 堆栈溢出部分，了解检测和预防堆栈溢出的技巧。
超出 USB 集线器的供电能力	如果电路板通过 ICE 供电，则其所需的电流可能会超出 ICE（及 USB 集线器）的供电能力。请使用直接外部电源（而不是通过 ICE 供电）或者更靠近主 PC USB 端口的 USB 端口。
未检测到任何事件	请确保事件已正确应用。要使 ICE 能够识别事件状态，则在调试器暂停时，必须点击事件窗口内的“Apply”（应用）按钮。如果在调试器运行或者断开连接时点击应用事件状态，则事件信息将不会应用至 ICE。
	请确保已为所关注事件选中“Break”（中断）复选框。
如果所有方法均失败：	如有可能，请尝试使用另一套硬件： - 首先，更换一个不同的（最好是新的）ICE，查看能否解决问题。如果这一方法有效，则问题可能出在 ICE 单元。 - 用 PSoC 单元替换转接板或电路板，以使用新的或不同的硬件集。如果问题得以解决，则可能是转接板、PCB 或者 ICE 连接器出现故障。
	卸载 PSoC Programmer 和 PSoC Designer 并重新安装这两个工具的最新版本。如果驱动或者其他调试相关的工具软件没有正确安装，则会导致连接问题。重新安装可确保所需的工具已正确安装并完成了最新的错误修复集。

总结

赛普拉斯提供很多硬件和软件工具，帮助您调试 PSoC 1 项目。本应用笔记应该可以帮助您更有信心地了解在调试项目或解决问题时会用到的硬件、软件和调试技巧。

附录 A – 向 PCB 添加 OCD 部件

如果支脚套件和转接板不适用于目标 PCB 或选定的部件型号没有适用的转接板，则一种可行的调试解决方案是直接向 PCB 布局中添加一个支持片上调试器 (OCD) 的 PSoC 1 器件。

与标准调试转接板相比，直接向 PCB 添加一个 OCD 部件具有以下优势：

- 部件成本更低，因为不需要支脚套件和完整的转接板装配。
- 比起为每个需要调试的电路板分别购买一个转接板，创建多个具备调试能力的电路板更易实现。
- 由于不需要支脚套件，因此，PSoC 芯片所需的 PCB 空间更少。连接器需要板上空间，但其可位于距 OCD 部件 4 英寸远的位置。

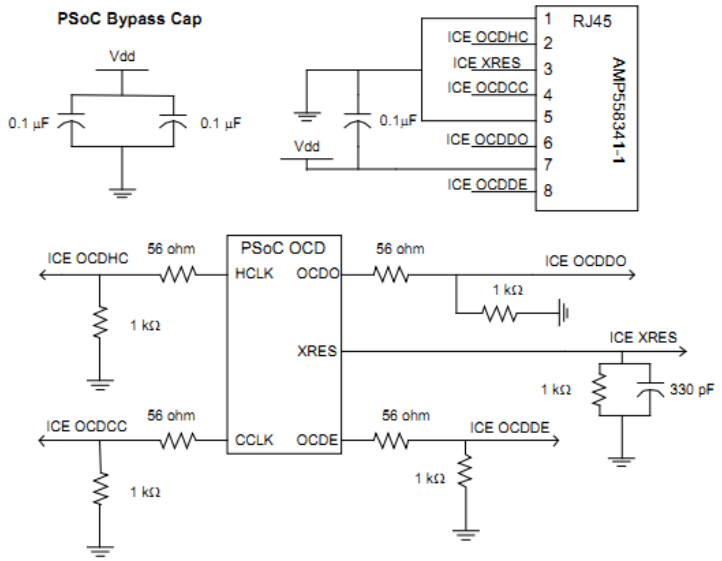
向 PCB 添加 OCD 部件所需的器件列于表 4 中。

表 4. 板上 OCD 调试器所需的器件

说明	数量	注
RJ-45 连接器	1	AMP/Tyco Electronics 部件编号 5557785-1 DigiKey 部件编号 A31457-ND
PSoC OCD 部件	1	参见片上调试器 (OCD) 部件。
56 Ω 电阻	4	1/16 W，5%
1 kΩ 电阻	4	1/16 W，1%
330 pF 电容	1	5-V 陶瓷 NPO
0.1-µF 电容	3	5-V 陶瓷 Y5V

板上 OCD 调试器所需的原理图如图 24 所示。

图 24. 板上 OCD 调试器原理图



所有串联电阻均作为终端用于信号线上的阻抗匹配。

图中包含的旁路电容用于过滤出电路中关键元件的交流电噪声。

OCDHC、OCDCC、OCDDO 和 OCDDE 的追踪长度应限制在 4 英寸以内。

附录 B – 堆栈溢出

不使用 ICE 检查堆栈溢出

本应用笔记在之前的内容中，我们讨论了使用 ICE 和事件检测堆栈溢出的方法（示例 3：堆栈溢出）。此方法可以有效地捕获并中断堆栈溢出。但并不适用于没有调试能力的部件。

所幸，堆栈溢出还可以通过固件进行检测。此类检测方法之一就是使用已知值预填充堆栈页面的末端。如果堆栈到达堆栈页面的末端并使用新值覆盖预填充的值，则表示可能存在堆栈溢出问题。

要在堆栈页面中设置预填充值，可以使用以下 C 代码：

```
char* StackPtr = (char*) 0x7FF;

*StackPtr = 0xAA;
```

上述代码创建了一个指向堆栈分页（这里为页面 7）最终 RAM 位置的指针，然后在该位置放置一个预填充值 0xAA。带有单页 RAM 的 PSoC 1 器件应使用预填充地址 0x0FF。第三个数字应始终与项目所用的堆栈分页匹配。

然后，可以使用以下代码检查代码在代码执行过程中是否有堆栈填充至该位置。

```
if (*StackPtr != 0xAA)
{
    //Stack overflow
}
```

堆栈应该在代码初始化序列中及早填充预填充的值，以免堆栈过大。在这之后，可以随时检查预填充的值。最好将此段代码放置于一段定期运行的代码中，例如主循环。如果堆栈在程序运行过程的任意时间到达页面底端，则会被此项检查操作检测出来。

请注意，如果填充值（在本示例中为“0xAA”）与预填充位置的进栈值相同，则不会检测出溢出。如果您担心这一问题，可以使用不同的预填充值多次运行测试，也可以对一个以上的堆栈位置进行预填充和检查。还需注意，StrackPtr 位置可以修改为任一堆栈分页或者某一堆栈分页内的任一位置。这样，您便可以确定给定项目堆栈的可用空间。例如，StackPtra 位置可以置于 0x780 或者 0x7C0，以查看分页 7 中堆栈的使用率是否已达到了 50% 或 75%。

避免堆栈溢出的方法

以下部分将详细介绍几种不同的方法，通过编写和修改代码避免或修复堆栈溢出。并非每个方法都适用于所有项目。因此，请从列表中选择满足您需求的方法。

- 由于编译器有所限制，请避免使用 CASE 语句。
- 如果可能，请避免使用复杂的 IF 语句。
- 限制传递变量的数量和大小。
- 限制本地变量的数量。

避免使用 CASE 语句

CASE 语句并非最有效的代码实践。使用 CASE 语句会增加堆栈空间的使用量，以存储每个可能的 SWITCH 语句的中间结果。代码示例参见代码 1，该段代码可以替换为代码 2 中所示代码。

代码 1. 会增加堆栈空间使用率的代码示例

```
switch ( bSwitchVariable )
{
   case (0x01):    // do something
      break;
   case (0x02):    // do something else
      break;
   default:        // do third thing
      break;
}
```

代码 2. 不会增加堆栈空间使用的代码示例

```
If ( bSwitchVariable == 0x01 )
{
   // do something
}
else if (bSwitchVariable == 0x02 )
{
   // do something else
}
else
{
   // do third thing
}
```

避免使用复杂的 IF 语句

导致在堆栈中存储额外字节的另一原因是复杂的 IF 语句。与前一部分中说明的 CASE 语句问题类似，IF 语句中如果含有若干个决策点，则在每次进行比较时，都会有一个字节进栈。因此，请尽量将这类语句替换为最简单的语句类型。代码示例参见代码 3，该段代码可以替换为代码 4 中所示代码。

代码 3. 复杂 IF 语句的代码示例

```
if (!(( bVariable == 0x00 )
    || ( bVariable == 0x10 )
    || ( bVariable == 0x20 )
    || ( bVariable == 0x30 )
    || ( bVariable == 0x40 )
    || ( bVariable == 0x50 )
    || ( bVariable == 0x60 )
    || ( bVariable == 0x70 )))
{
   // do something

}
```

代码 4. 简单 IF 语句的代码示例

```
if (!(( bVariable & 0x8F ) == 0x00))
{
   // do something
}
```

限制传递变量的数量和大小

在 C 语言的函数调用过程中，所有的传递变量均会进栈存储。因此，要减少堆栈的使用，则应限制传递变量的数量和大小。一种解决方法是使用全局变量。全局变量储存在固定的 RAM 位置中，所有函数均可对其进行访问，且不会向堆栈添加任何内容。您可以将指针指向局部变量，而不用传递变量或声明全局变量，但是在 PSoC 1 器件中，指针的大小为 2 个字节，反而加倍了堆栈的使用量。指针仅应该用于传递缓冲区或者阵列的首个地址，这样，置于堆栈中的仅为 2 字节的地址，而不是阵列的全部内容。

限制局部变量的数量

函数使用的每个局部变量在进入函数时均会在堆栈中分配到 1 个字节。因此，请务必限制函数所用的局部变量的数量。

代码示例参见代码 5，该段代码可以替换为代码 6 中所示代码。

代码 5. 含有很多局部变量的代码示例

```
void SampleFunction ( * abMessageBuffer,
bMessageLength )
{
   // local variables
   BYTE bMessageAddress;
   BYTE bMessageType;
   BMessageAddr = abMessageBuffer[0];

   if ( bMessageAddr == 0x01 )
   // do something

   bMessageType = abMessageBuffer[1];
   if ( bMessageType == 0x01 )
   // do something else

   return;

}
```

代码 6. 含有有限局部变量的代码示例

```
void SampleFunction ( * abMessageBuffer,
bMessageLength )
{
   if ( abMessageBuffer[0] == 0x01 )
   // do something

   if ( abMessageBuffer[1] == 0x01 )
   // do something else

   return;
}
```

附录 C – 旧版硬件

本部分记录了一些旧版调试硬件，赛普拉斯现已不再主动支持这类硬件。赛普拉斯建议您将旧版硬件升级至本应用笔记中描述的最新版本硬件。

ICE-4000

ICE-4000 是 ICE-Cube 的前身，自 PSoC Designer 4.4 和 PSoC Programmer 2.3 以后的版本均不支持这一硬件。

ICE-4000 通过并行端口与计算机连接，主要用于调试旧版的 CY8C25xxx 和 CY26xxx 部件（不建议用于新设计）。

图 25. ICE-4000

Flex 转接板

Flex Pod 是另一种常见的旧版调试硬件。在柔性线缆的一端配有标准的 ICE 连接器，线缆本身装有一个 OCD 部件，线缆的另一端则是一个 PDIP 连接器（用于原型设计）。

ICE 和 PSoC Designer 已不再支持该硬件。应将 Flex Pod 升级至最新版本的 CY3250 转接板或者 CY3210-EvalPods。

图 26. 旧版 Flex 转接板

关于作者

姓名：	Dan Sweet
职位：	高级应用工程师
背景：	Dan Sweet 毕业于西华盛顿大学，获得电气工程技术理学学士学位。
联系：	drsw@cypress.com

文档修订记录

文档标题：使用 PSoC® 1 进行调试

文档编号：001-79031

修订版	ECN	变更人	提交日期	变更说明
**	3610419	VLX	2012-05-09	本文档版本号为 Rev**，译自英文版 001-73212 Rev**
*A	3888973	VLX	2013-01-29	本文档版本号为 Rev*A，译自英文版 001-73212 Rev**. AN number error in Page 1 corrected.

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、工厂和经销商组成的全球性网络。要找到距您最近的办事处，请访问赛普拉斯所在地。

产品

汽车	cypress.com/go/automotive
时钟与缓冲器	cypress.com/go/clocks
接口	cypress.com/go/interface
照明和电源控制	cypress.com/go/powerpsoc cypress.com/go/plc
内存	cypress.com/go/memory
光学导航传感器	cypress.com/go/ons
PSoC	cypress.com/go/psoc
触摸传感	cypress.com/go/touch
USB 控制器	cypress.com/go/usb
无线/射频	cypress.com/go/wireless

PSoC 解决方案

psoc.cypress.com/solutions
PSoC 1 | PSoC 3 | PSoC 5

赛普拉斯开发者社区

社区 | 论坛 | 博客 | 视频 | 培训

PSoC 是赛普拉斯半导体公司的注册商标。“可编程片上系统”、PSoC Designer 和 PSoC Express 是赛普拉斯半导体公司的商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

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

电话 ：408-943-2600
传真 ：408-943-4730
网站 ：www.cypress.com

© 赛普拉斯半导体公司，2013。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品的内嵌电路之外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

该源代码（软件和/或固件）均归赛普拉斯半导体公司（赛普拉斯）所有，并受全球专利法规（美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和/或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定的用途之外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对此材料提供任何类型的明示或暗示保证，包括（但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不做出通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用的赛普拉斯软件许可协议限制。