AN89056

PSoC 4 — IEC 60730 Class B 和 IEC 61508 SIL 安全软件库

作者：Vasyl Parovinchak，Taras Kuzo

相关项目：有

相关器件系列：PSoC 4000, PSoC 4100, PSoC 4200, PSoC 4200M

软件版本：PSoC Creator™ 3.1

相关应用笔记：AN78175

AN89056 介绍了 PSoC® 4 MCU IEC 60730 Class B 和 IEC 61508 SIL 安全软件库，并包含带有自检测子程序的示例项目，用于确保可靠安全地进行操作。您可以将这些库子程序和示例项目中所包含的示例集成到您的应用中。本应用笔记还说明了库中所提供的 API 函数。

目录

1 简介 1
2 IEC 60730-1 附录 H 概览 2
3 IEC 61508-2 附录 A 概览 2
4 IEC 60730 ClassB 和 IEC 61508 标准要求 3
5 安全软件库 4
6 PSoC 4 的 API 函数 4
6.1 CPU 寄存器测试 4
6.2 程序计数器测试 5
6.3 程序流测试 6
6.4 中断处理和执行测试 6
6.5 时钟测试 8
6.6 闪存（恒量存储器）测试 8
6.7 SRAM（变量存储器）测试 9
6.8 堆栈溢出测试 12
6.9 数字 I/O 测试 13
6.10 ADC 和 DAC 测试 13
6.11 电压比较器测试 15
6.12 运算放大器测试 15
6.13 通信 UART 测试 16
6.14 通信 SPI 测试 17
6.15 UDB 配置寄存器测试 18
6.16 启动配置寄存器测试 19
6.17 看门狗测试 20
6.18 窗口模式下的看门狗定时器 20
6.19 UART 数据传输通信协议示例 24
7 总结 27
8 参考文档 27
附录 A：进行测试闪存（恒量存储器）时设置校验和 28
附录 B：IECEE CB 计划测试证书 31
附录 C：受支持的芯片型号列表 32
附录 D：MISRA 合规性 33
文档修订记录 36
销售、解决方案以及法律信息 37

1 简介

当前，大多数内置在家庭应用和工业产品中的自动电子控制器都使用了单芯片微控制器单元（MCU）。制造商开发的实时嵌入式固件在 MCU 内执行，并提供了用于智能控制家庭应用和工业机器的功能。由过热、静态放电、过压或其他因素造成的 MCU 损坏会使终端产品进入未知或不安全状态。

国际电工委员会（IEC）制定的 60730-1 安全标准对各种家庭应用中的机械、电气、电子、环境承受力、电磁兼容性（EMC）和不正常操作等因素进行了相关讨论。IEC 61508 规范列出了针对工业设计中的电气、电子、可编程电子（E/E/PE）安全相关系统的详细要求。这两种规范的测试要求十分相似，可以在本文档中或者安全软件库中找到。

本应用笔记着重介绍了 IEC 60730-1 规范中的附录 H（电子控制控制要求）和 IEC 61508-2 规范的附录 A（E/E/PE 安全相关系统的技术和测量：控制操作过程中的故障）。这些章节说明的测试和诊断方法可提高家庭应用和工业机器中嵌入式控制硬件和软件的操作安全。

2 IEC 60730-1 附录 H 概览

IEC 60730-1 标准附录 H 将应用软件分成以下几类：

- Class A 控制功能：这种功能不影响设备的安全情况。例如，潮湿控制器、照明控制器、定时器和开关器件。
- Class B 控制功能：可用于防止受控设备发生不安全操作。例如，洗衣设备的热保护和门锁功能。
- Class C 控制功能：用于防止发生特殊的危险情况（如由受控设备引起的爆炸）。例如：热水器被关闭、无排气口等情况下的燃料自动控制和热气流断流器功能。

大型电器产品（如洗衣机、洗碗机、干燥机、冰箱、冰柜和炊具/炉）经常使用 Class B 的控制功能。其中可能会引起爆炸的电器设备（如燃气控制机）使用的是 Class C 的控制功能。

Class B 安全软件库以及本应用笔记所提供的示例项目实现了 Class B 控制功能类型中所规定的自检测和自诊断方法。这些方法使用不同方式进行检测与软件相关的故障和错误，并做出响应。按照 IEC 60730-1 标准，自动电子控制技术制造商必须使用下面结构类型中的一种来设计自己的 Class B 软件：

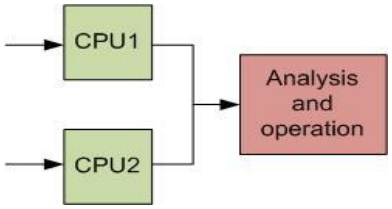
- 具有功能测试性能的单通道
- 具有定期自检测性能的单通道
- 不具有比较功能的双通道（如图 1 所示）

在带有功能测试性能的单通道结构中，软件使用单个 CPU 来执行各项指令功能。将在应用程序启动后执行功能测试操作，从而确保所有主要功能可以可靠地发挥作用。

在带有定期自检测性能的单通道结构中，软件使用单个 CPU 来执行各项指令功能。各项定期自检测性能被嵌入在软件中，并且自检测操作会在软件处于执行模式期间定期执行。在这种结构中，会要求 CPU 定期检查电子控制器的主要功能，并要确保该工作不能同终端应用执行发生冲突。

在不具有比较功能的双通道结构中，软件使用两个 CPU 来执行各项主要功能。在执行某项主要功能前，需要确保两个 CPU 已经完成了各自相关的任务。例如，当解除洗衣机的门锁时，一个 CPU 会停止旋转滚筒电机，另一个 CPU 会检测滚筒速度，从而确认它已经停止，如图 1 所示。

图 1. 不具有比较功能的双通道结构



由于需要具备两个 CPU（或两个 MCU），因此实现双通道结构所需费用会更大。此外，这种应用的执行也更为复杂，因为两个器件间需要定期进行通信。因此，带有定期自检测性能的单通道结构是最通用的执行情况。

3 IEC 61508-2 附录 A 概览

IEC 61508-2 附录 A 介绍的最大诊断范围可应用于工业设计中的相关技术和措施。本应用笔记中未涵盖的其他要求可应用于特殊的工业领域，如铁路、处理过程控制、汽车级产品、核电以及机械装置。对于每个安全完整性级别（SIL），该附录推荐了用于解决控制随机硬件、系统、环境和操作等故障的技术和措施。更多有关架构和措施，请参考 IEC 61508-6 标准附录 B 和 IEC 61508-7 标准附录 A。

要想避免这些故障或发生时对其进行控制，通常需要采取几项措施。IEC 61508 标准的附录 A 和附录 B 所涵盖的要求可分为两类：用于避免 E/E/PE 系统安全生命周期不同阶段的故障（附录 B）；用于控制操作期间的故障（附录 A）。各项用于控制故障的措施都是 E/E/PE 安全相关系统的内置性能。

该程序开始于对受控设备上每个危险事件进行风险评估。通常会根据每种故障发生的可能性以及故障的影响来判定诊断范围和安全故障比例。这样会加大风险，使一个远程灾难性故障会与常见的可忽略故障存在相似的风险。因此，风险评估过程得出的目标 SIL 便是对终端系统的要求。

SIL 等级的含义会根据设备操作的使用频率而存在差异。几乎所有设备均被归类为“高要求”，因为它们的使用频率高于一次/年。根据“高要求”类型中不同 SIL 级别的危险故障概率（使用期间每小时发生的次数）分为如下几类：

- SIL 1：≥10-6 ~ < 10-5 （11 年发生一次）
- SIL 2：≥10-7 ~ < 10-6 （114 年发生一次）
- SIL 3：≥10-8 ~ < 10-7 （1144 年发生一次）
- SIL 4：≥10-9 ~ < 10-8 （11446 年发生一次）

4 IEC 60730 ClassB 和 IEC 61508 标准要求

按照 IEC 60730-1 Class B 附录 H 中表 H.11.12.7 以及 IEC 61508-2 附录 A 中表 A.1~A.14 的内容，根据软件类型，必须测试某些组件。通常，每个组件已经提供的各项可选措施用于验证或测试相关组件，因此制造商可灵活进行操作。

为了遵守单通道结构的 Class B IEC 60730 和 IEC 61508 规范，电子控制技术制造商需要对表 1 中列出的各个组件进行测试。

表 1. 单通道结构中需要测试的组件

需要进行电子控制测试的 Class B IEC 60730 组件（附录 H 中的表 H.11.12.7）	需要进行测试的 IEC 61508 组件（附录 A 中的表 A.1–A14）	故障/错误
1.1 CPU 寄存器	A.4、A.10 CPU 寄存器	卡住
1.3 CPU 程序计数器	A.4、A.10 程序计数器	卡住
2. 中断处理和中断执行	A.4 中断处理	无中断或过于频繁中断
3. 时钟	A.11 时钟	错误频率
4.1 恒量存储器	A.5 恒量存储器	所有单位故障
4.2 变量存储器	A.6 变量存储器	直流故障
4.3 寻址（相关于变量/恒量存储器）	A.4、A.10 地址计算	卡住
5.1 内部数据路径的数据	A.8 数据路径（内部通信）	卡住
5.2 内部数据路径寻址 （仅用于扩展存储器 MCU 系统）	–	错吴地址
6.1 外部通信数据	A.7 I/O 单元和接口	汉明距离 = 3
6.2 外部通信寻址	A.7 I/O 单元和接口	汉明距离 = 3
6.3 时序	–	时间/时序中的错误点
7.1 I/O 外围	A.7 I/O 单元和接口	附录 B 和“IEC 60730-1，H.27”中说明的故障状态
7.2.1 模拟 A/D 和 D/A 转换器	A.3 模拟信号监控	附录 B 和“IEC 60730-1，H.27”中说明的故障状态
7.2.2 模拟复用器	–	错误寻址

在执行 Class B 软件库期间，用户应用必须确定需要使能还是禁用中断。例如，如果在执行 CPU 自检测子程序期间发生某个中断，那么某个寄存器中可能会发生一个意外的更改。因此，当中断服务子程序（ISR）被执行时，寄存器的内容将与预期值互相不匹配。

Class B 安全软件库中的示例项目显示了需要禁用和使能中断的位置，从而正确执行自检测操作。

5 安全软件库

可将本应用笔记中介绍的安全软件库应用于 PSoC 4 MCU 器件。该库包含了各个 API，用于在故障检测过程中最大限度地提高应用的可靠性。

某些自检测操作只会通过将适当的 API 函数添加到 Class B 安全软件库的 **.c* 和**.h* 文件上的方式得以实现。其他自检测操作则可以通过将适当的 API 函数添加到 **.c* 和**.h* 文件中，并通过修改项目原理图的方式得以实现。

本应用笔记描述并实现了这两种自检测功能：

- 自检测功能有助于符合 IEC 60730-1 Class B 和 IEC 61508-2 标准。
  - CPU 寄存器：测试被卡位
  - 程序计数器：进行测试，从而跳转到准确的地址。
  - 程序流程：进行测试，从而检查准确的固件程序流程
  - 中断处理和执行：测试准确的中断调用和周期性
  - 时钟：测试错误频率
  - 闪存（恒量存储器）：测试存储器的损坏情况
  - SRAM（变量存储器）：测试被卡的位和合适的存储器寻址
  - 堆栈过流：进行测试，以检查程序执行期间编程数据存储器中所发生的堆栈过流情况
  - 数字 I/O：测试引脚短路
  - 模数转换器（ADC）和数模转换器（DAC）：测试正常的功能
  - 比较器：测试正常的功能
  - 通信层（UART、SPI）：测试数据接收是否准确
- 由于具有可编程互联性能，因此 PSoC 4 期间可支持额外的自检测功能。虽然 IEC 60730-1 的附录 B 或 IEC 61508-2 标准没有包含终端应用，但通常它们也需要进行这些自检测操作。
  - 运算放大器测试
  - Universal Digital Block (UDB)配置寄存器测试
  - 启动配置寄存器测试
  - 看门狗测试：测试芯片复位情况
  - 其他窗口看门狗定时器（WDT），用于监控固件的执行情况

设备启动后可以立即执行所有自检测操作，并在设备执行期间持续执行。设备启动时执行自检测有助于决定芯片是否符合在执行应用固件前进行的操作。在正常操作期间执行自检测操作可允许执行连续损坏检测以及用户定义的纠正措施。

下面章节介绍了每种测试执行的详情，并列出了执行相关测试所需要的 API。

6 PSoC 4 的 API 函数

6.1 CPU 寄存器测试

带有 Cortex-M0 CPU 的 PSoC 4 器件具有 16 位和 32 位两种寄存器：

- R0 到 R12 — 通用寄存器。
- R13 — 堆栈指针（SP）：存在两个堆栈指针，但一次只有一个指针可用。该 SP 始终是 32 位字对齐；所有位[1:0]始终被忽略，并且它们的值被认定为“0”。
- R14 — 链接寄存器：该寄存器在调用函数期间会存储返回的程序计数器。
- R15 — 程序计数器：通过写入该寄存器可以控制程序流。

通过棋盘检测方式，CPU 寄存器可以检测 CPU 寄存器中的卡住故障。这种测试可确保寄存器中的各位不被卡在数值“0”或数值“1”。它是一种无损测试，用于执行以下主要任务：

1. 执行该子程序前，CPU 寄存器中已经经过测试的内容将被保存在堆栈上。

2. 依次通过将二进制序列 01010101 和跟随在它后面的 10101010 写入到寄存器内，然后读取这些寄存器中的值进行验证；然后测试寄存器。
3. 如果返回值不匹配，那么该测试将返回错误代码。

棋盘法适用于所有 CPU 寄存器（程序计数器除外）。

函数

```
uint8 SelfTest_CPU_Registers(void)

Returns:  0   No error
          1   Error detected

Located in:  SelfTest_CPU.c
             SelfTest_CPU.h
```

调用 `SelfTest_CPU_Registers` 函数，从而执行 CPU 检测。

如果发生错误，那么 PSoC 器件将不再继续执行，因为这时它的操作是不可预知的，因此可能潜在不安全因素。

6.2 程序计数器测试

PSoC 4 CPU 程序计数器 R15 寄存器是 CPU 寄存器集的一部分。通常需要使用棋盘法来测试这些寄存器，并且要求为该测试分配地址 0x5555 和 0xAAAA。其中，地址 0x5555 和 0xAAAA 用于表示棋盘位的图像。

程序计数器（PC）测试实现了 IEC 60730 标准，即 H.2.16.5 一节中规定的功能测试。该 PC 保持了被执行的下一条指令的地址。该测试执行了下述主要任务：

1. 调用闪存存储器中不同位置的函数。对于 PSoC 4 器件，可以通过使用 **.ld* 文件中的链接器脚本实现。

```
NV_CONFIG1 0x5555 :
{
    . = 0x00;
    *(PC5555);
} >rom =0

NV_CONFIG2 0xAAAA :
{
    . = 0x00;
    *(PCAAAA);
} >rom =0
```

   在该示例项目中，它已经被添加到 *custom_cm0gcc.ld* 文件中。要想添加链接器文件，请依次选择 **Project** > **Build Setting** > **Linker** > **Custom Linker Script**。
2. 各个函数都将返回唯一的一个数值。
3. 通过使用 PC 测试功能可以验证返回的值。
4. 如果这些值互相匹配，那么 PC 会分支到正确的位置，或一个 WDT 会触发一个复位，因为程序执行超出了范围。

函数

```
uint8 SelfTest_PC(void)
Returns:  0   No error
          1   Error detected
Located in:  SelfTest_CPU.c
             SelfTest_CPU.h
```

注意：对于 PSoC 4，必须将 *custom_cm0gcc.ld* 文件添加到链接器内。

`SelfTest_PC()` 函数被调用，用于执行 PC 测试。

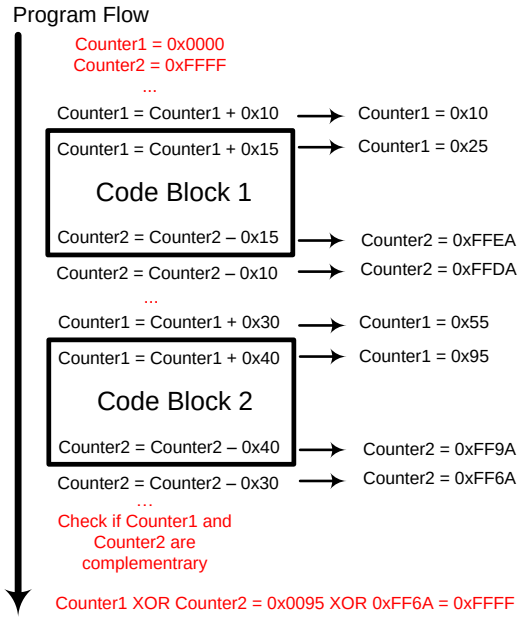
6.3 程序流测试

使用一种特定的方法来检查程序执行流程。对于每一个关键执行代码模块，模块执行前和执行后唯一的数值会被添加到互补计数器内或从互补计数器中踢出。通过这些流程，您可以查看是否正确从主程序流程中调用了代码模块，并检查该模块是否被正确执行。

只要退出点和输入点的数量相同，那么每次测试模块后计数器对都互补。请参见图 2。

任何意外的值都被视为程序流执行错误。

图 2. 程序流测试



6.4 中断处理和执行测试

借助 PSoC 4 中断控制器，硬件资源可以独立于主代码当前正在执行的任务，使程序地址跳转到一个新的地址位置。另外，完成 ISR 后它们将继续处理中断代码。

中断测试会实现 IEC 60730 标准 H.2.18.10.4 部分中所定义的独立时隙（time-slot）监控。它将检查所发生的中断数量是否位于预定的范围内。

中断测试是为了验证中断是否经常发生。通过使用定时器 UM 驱动的中断源，测试则会检查中断控制器。

函数

```
uint8 SelfTest_Interrupt(void)

Returns:  0   No error
          1   Error detected

Located in:  SelfTest_Interrupt.c
             SelfTest_Interrupt.h
```

注意：组件名称如图 3 所示。进行该测试时必须使能全局中断，但要禁用所有其他中断（isr_1 除外）。

调用 `SelfTest_Interrupt()`函数，以检查中断控制器操作。调用函数会启动定时器。

配置 Timer_1，使其在 1 ms 时间内生成 13 个中断。isr_1 会计算已发生的中断数量。如果 isr_1 的计数值为≥ 9，且不大于 15，那么测试成功。测试成功所需要的具体中断数量取决于应用，并可以根据需要进行修改。

图 4 显示的是中断自检测流程图。

图 3. 中断自检测 PSoC Creator 原理图

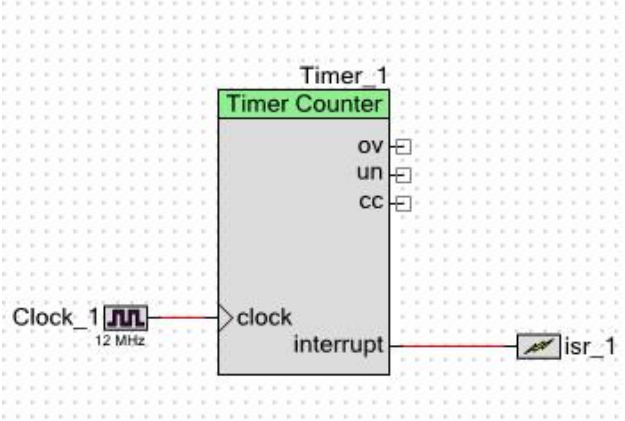
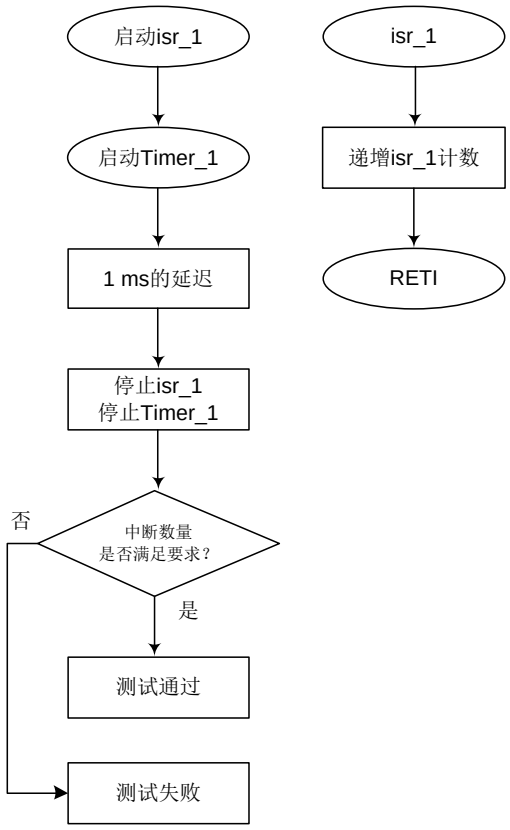


图 4. 中断自检测流程图



6.5 时钟测试

时钟测试可实现 IEC 60730 标准 H.2.18.10.4 部分所定义的独立时隙（time-slot）监控。它验证了内部主振荡器（IMO）系统时钟的可靠性，特别是要保证该系统时钟速度在内部低速振荡器（ILO）的容限内既不能太快又不能太慢。ILO 时钟的准确度为± 60%。如果要求准确度高于 60%，那么可以使用精度系统级信号或量产测试来将 ILO 调整为更准确的数值 。可使用 CLK_ILO_TRIM 寄存器来调整 ILO。

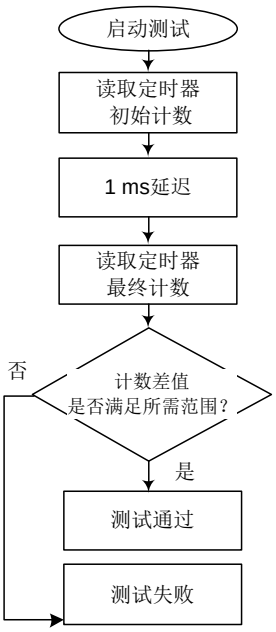
函数

```
uint8 SelfTest_Clock(void)

Returns:  0   No error
   Not 0  Error detected
Located in:  SelfTest_Clock.h
             SelfTest_Clock.c
```

注意：*SelfTest_Clock.h* 文件中定义了所测试时钟的准确度。可根据终端系统的要求更改时钟准确度。

时钟测试使用 16 位定时器 0。该定时器被集成到 WDT 上，并且由 32.768 kHz ILO 提供时钟脉冲。WDT 定时器 0 是一个带溢出的连续递增计数 16 位定时器。通过读取定时器的当前计数开始执行测试，然后通过软件延迟等待 1 毫秒，最后执行第二次读定时器计数。那么可以在一个定时器溢出中期测试的特殊情况下减去和修正这两个计数值。然后测试所测周期（额定值为 33 次计数）。如果该周期处于预定范围内，则测试成功。图 5 显示的是时钟自检测流程图。

图 5．时钟自检测流程图



6.6 闪存（恒量存储器）测试

PSoC 4 器件包含一个高达 128 KB 的片上闪存存储器。闪存存储器按照行的形式进行组织，其中每一行包含 128 个数据字节。

6.6.1 校验和方法

为了完成闪存存储器完整的诊断，需要计算所有被使用的闪存的校验和。

当前库使用了一个 Fletcher 的 64 位校验和。由于 Fletcher 的 64 位校验和完全可靠，所以选择该方法。

您可以将 Fletcher 的校验和方法更改为 *SelfTest_Flash.c* 文件中 SelfTest_CheckSum_Formula()函数的任意校验和方法。

SelfTest_Flash.h 文件中的 PSOC_FLASH_SIZE 定义了监控时所需要的闪存大小。

所提出的校验和闪存测试读取 ROM 或闪存位置，并堆积 64 位变量中存储的值来计算整个闪存存储器的运行校验和。闪存实际的 64 位校验和被存储在闪存自身最后 8 个字节中。当测试到闪存终端地址减去 8 个字节（32 KB 器件上的 0x8FF8）所在的位置时，便停止。然后将计算的校验和值与闪存最后 8 个字节中存储的实际值进行比较。如果比较后相互不匹配，则表示闪存失败，这时代码执行被冻结，从而避免尝试执行无效代码。

6.6.2 编程步骤

开始测试前，您需要正确设置预计算校验和，如附录 A：进行测试闪存（恒量存储器）时设置校验和章节中所介绍的内容。

函数

```
uint8 SelfTest_FlashCheckSum()
      Returns:     1    Error detected
         2   Checksum for one block calculated, but end of Flash was not reached
         3   Pass, Checksum of all flash is calculated and checked
      Located in: SelfTest_Flash.c
                  SelfTest_Flash.h
```

调用 SelfTest_FlashCheckSum()函数，以便通过使用校验和方法来执行闪存存储器损坏测试。在调用过程中，该函数会计算一个闪存模块的校验和。通过使用 *SelfTest_Flash.h* 文件中各个参数可以设置该模块的大小。

```
/*Set size of one block in Flash test*/
       #define FLASH_DOUBLE_WORDS_TO_TEST (512u)
```

必须多次调用该函数，直到测试完整个闪存域为止。每次调用函数都将自动跳转到到下一个测试模块。如果计算好模块校验和，并达到测试闪存的最后地址，测试会返回 0x03。如果计算好模块校验和，但未达到闪存的最后地址，那么测试会返回 0x02。如果检测到一个错误，那么测试会返回 0x01。

注意：如果在运行过程中闪存中发生了变化，那么不会进行检测。调用测试前，需要更新校验和。其他可更改闪存数据的测试需要在测试闪存前调用。

6.7 SRAM（变量存储器）测试

注意：PSoC 4 器件包含一个高达 16 KB 的片上 SRAM。该 SRAM 的一部分包含位于存储器终端内的堆栈。

变量存储器测试会实现 IEC 60730 标准 H.2.19.6 一节中所定义的定期静态存储器测试。它会检测变量存储器中的单比特故障。 变量存储器测试可以是毁坏性的，也可能是非毁坏性的。毁坏性测试会在测试期间破坏存储器中的内容，而非毁坏性测试则能够保存存储器的内容。对于本库中所使用的毁坏性测试算法，它被封装成代码。首先，该代码会在进行测试前保存存储器中的数据，完成测试后恢复这些数据。

变量存储器包含在执行程序中发生变化的数据。RAM 存储器测试用于确定 RAM 存储器的任意位卡在‘1’还是‘0’处。通过使用 March 存储器测试和棋盘法测试是最广泛用于检查 DC 故障的静态存储器算法。

March 测试包括一系列相似的测试，它们间的算法存在少量变化。March 测试的变量使用大写字母表示，允许测试符合具体架构的测试要求。March C 测试是为实现 PSoC 4 安全软件库，因为它提供了与棋盘法更好的测试范畴，并且是该器件的最佳 March 方法。单独运行“变量 SRAM”和“堆栈 SRAM”区域，以确保数据在测试过程中不被损坏。

6.7.1 March C 测试

March 测试对存储器阵列中所有存储器单元执行有限的一组操作。March C 测试用于检测变量存储器中下面各类型的故障：

- 固定故障
- 查找故障
- 切换故障
- 耦合故障

测试复杂性为 11n，其中“n”表示存储器中的位数量，这是因为测试每个位置时需要进行 11 项操作。当该测试为毁坏性操作时，赛普拉斯通过在每个测试期间只测试一个小的存储器模块来提供 March C 测试，从而避免数据破坏，使模块的数据得到保存和恢复。

6.7.2 March C 算法

March 测试符号：

>	地址顺序按升序排列
<	地址顺序按降序排列
<>	地址顺序按升序或降序排列
r0	表示读操作（从某个存储器单元中读‘0’）
r1	表示读操作（从某个存储器单元中读‘1’）
w0	表示写操作（从某个存储器单元中写‘0’）
w1	表示写操作（从某个存储器单元中写‘1’）

```
MarchC
{
    <> (w0)
    > (r0 then w1)
    > (r1 then w0)
    <> (r0)
    < (r0 then w1)
    < (r1 then w0)
    > (r0)
}
```

函数

```
uint8 SelfTests_SRAM_March(void)
Returns:  1   Error detected
          2   Still testing
          3   Pass, all RAM is tested
Located in:  SelfTest_RAM.c
             SelfTest_RAM.h
```

调用该函数，以便在运行时进行 March SRAM 测试，并不会导致数据损坏。

通过使用起始地址和结束地址指针，该函数会将正在测试的 SRAM 区域备份到 SRAM 的其他保留部分中，然后恢复数据，从而执行运行期间发生的 March C 测试。

复制数据前，先要使用 March 测试来检验 SRAM 中保留的部分。该存储器区域被损坏。因此，不能在该存储器中存储或放置变量。用户负责控制该存储器。建议您将变量存入到未使用的阵列中，并设置一个禁止优化该阵列的编译器提示。*AN_89056_Cpu* 项目中显示该问题。

SRAM 的保留区域位于 SRAM 终端区域内（就在堆栈区域前），并使用 *SelfTest_SRAM_March.s* 文件中下述各参数进行设置：

```
MARCH_BUFF_ADDR_START: .word  (CYDEV_SRAM_BASE  +  CYDEV_SRAM_SIZE  -  CYDEV_STACK_SIZE  -
RESERVE_BLOCK_SIZE)

MARCH_BUFF_ADDR_END:   .word (CYDEV_SRAM_BASE + CYDEV_SRAM_SIZE - CYDEV_STACK_SIZE)

.if (TEST_BLOCK_SRAM_SIZE>TEST_BLOCK_STACK_SIZE)
    .equ RESERVE_BLOCK_SIZE,    TEST_BLOCK_SRAM_SIZE
.else
    .equ RESERVE_BLOCK_SIZE,    TEST_BLOCK_STACK_SIZE
.endif
```

后面是 SRAM 内其他部分位置的区域。例如：

```
CYDEV_SRAM_BASE = 0x20000000;
CYDEV_SRAM_SIZE = 0x00001000;
```

[0x20000000; (0x20001000-CYDEV_STACK_SIZE- RESERVE_BLOCK_SIZE)]	变量 SRAM
[(0x20001000-CYDEV_STACK_SIZE- RESERVE_BLOCK_SIZE); (0x20001000-CYDEV_STACK_SIZE)]	March C 测试的缓冲区（被保留的 SRAM 部分）
[(0x20001000-CYDEV_STACK_SIZE); 0x20001000]	堆栈 SRAM

调用期间，该函数会测试 SRAM 的一个模块。该模块大小与 SRAM 中保留缓冲区的模块大小相同。通过使用 *SelfTest_SRAM_March.s* 文件中以下各参数，可以设置模块大小：

```
.equ TEST_BLOCK_SRAM_SIZE, 0x00000400
```

只针对变量 SRAM 区域进行测试，而不是测试堆栈区域。如果成功测试模块，并达到保留区域的起始地址，那么测试会返回 0x03。如果成功测试模块，但并未达到保留区域的起始地址，那么测试会返回 0x02。如果检测到一个错误，那么测试会返回 0x01。

函数

```
void SelfTests_Init_March_SRAM_Test(uint8 shift)
Located in:  SelfTest_RAM.c
SelfTest_RAM.h
```

该函数用于初始化 SRAM 的基本地址，在下述两种情况中应该调用该函数：

- 在第一次调用 SelfTests_SRAM_March()之前。在这种情况下，该函数会第一次初始化测试起始地址。
- 当全部 SRAM 被测试，并且 SelfTests_SRAM_March()返回“Pass”状态（0x02u）时。在这种情况下，该函数会初始化测试起始地址。

参数“shift”可设置从测试起始地址进行的移位。该参数用于设置各种不同的起始地址，并允许测试覆盖掉所有先前测试的模块边界。如果没有使用参数“shift”，那么不能对每个模块的第一个字节和最后一个字节进行与相邻测试模块互动的测试。通常，参数“shift”将在 0 和每个完整测试序列的半个测试模块大小间进行交替。

例如：

实例 1：

```
TEST_BLOCK_SRAM_SIZE = 10;
CYDEV_SRAM_SIZE      = 100;
SelfTests_Init_March_SRAM_Test(0x00u);
```

在每个 SelfTests_SRAM_March()调用过程中，测试范围为：

[0-9]; [10-19]; [20-29] …..[80-99]; [90-99]

实例 2：

```
TEST_BLOCK_SRAM_SIZE = 10;
CYDEV_SRAM_SIZE      = 100;
SelfTests_Init_March_SRAM_Test(0x05u);
```

每次调用 SelfTests_SRAM_March()过程中，测试范围为：

[5-14]; [15-24]; [25-34] …..[85-94]; [95-99]

每次完成完整的 SRAM 测试后，可以更改参数“shift”。

函数

```
uint8 SelfTests_Stack_March (void)
Returns:  1   Error detected
          2   Still testing
          3   Pass, all RAM is tested
Located in:  SelfTest_RAM.c
             SelfTest_RAM.h
```

该函数被调用，以便在运行时进行 March 堆栈测试而不会导致数据损坏。

通过使用起始地址和结束地址指针，该函数会将被测试的堆栈部分备份到保留的 SRAM 部分中，然后恢复数据，从而执行运行期间发生的 March C 测试。

复制数据前，先使用 March 测试来测试 SRAM 中保留的部分。该函数使用所保留的 SRAM 部分与 SRAM March 测试所使用的部分相同。

保留的 SRAM 部分被损坏。因此，不能在该部分存储器中存储或放置变量。用户负责控制该存储器。建议您将变量存入到未使用的阵列中，并设置一个禁止优化该阵列的编译器提示。

调用期间，该函数会测试 SRAM 的一个模块。该模块大小与 SRAM 中保留缓冲区的模块大小相同。通过使用 *SelfTest_SRAM_March.s* 文件中以下各参数，可以设置模块大小：

```
.equ TEST_BLOCK_STACK_SIZE, 0x00000040
```

如果成功测试该模块，并且达到 SRAM 的结束地址（意味着所有堆栈 RAM 都通过了测试），那么测试会返回 0x03。如果成功测试该模块，但未达到 SRAM 的结束地址，那么测试会返回 0x02。如果检测到一个错误，那么测试会返回 0x01。

函数

```
void SelfTests_Init_March_Stack_Test (uint8 shift)
Located in:  SelfTest_RAM.c
             SelfTest_RAM.h
```

该函数用于初始化堆栈 SRAM 的基本地址，在下述两种情况下应该调用该函数：

- 第一次调用 SelfTests_Stack_March()前。在这种情况下，该函数会第一次初始化测试起始地址。
- 当全部堆栈 SRAM 均被测试，并且 SelfTests_Stack_March()返回“Pass”状态（0x02u）时。在这种情况下，该函数会初始化测试起始地址。

参数“shift”可设置从测试起始地址进行的移位。它用于设置各种不同的起始地址并覆盖地址的所有形式。

用于该函数的示例与用于介绍 SelfTests_Init_March_SRAM_Test(uint8 shift)函数的示例相同。

6.8 堆栈溢出测试

堆栈是 RAM 的一部分，CPU 使用堆栈暂时存储信息。该信息可以是数据或地址。由于寄存器数量有限，因此 CPU 需要使用该存储空间。

在 PSoC 4 中，堆栈位于 RAM 末端并向下增长。堆栈指针的宽度为 32 位，可以使用所有 PUSH 指令递减该指针，并使用 POP 指令递增它。

堆栈溢出测试是为了确保在程序执行期间堆栈不会与编程数据存储器发生重叠。例如，若使用了递归函数，可能发生这种情况。

要想执行该测试，需要将一个预定义模型填入到位于堆栈末端的保留固定存储器模块内，并且定期调用测试函数以验证该模型。如果发生堆栈溢出，损坏的数据字节将覆盖掉该保留模块，这种情况被视为溢出错误。

函数

```
void SelfTests_Init_Stack_Test(void)
Returns: NONE
Located in:  SelfTest_Stack.c
             SelfTest_Stack.h
```

一旦调用该函数，预定义模型便会填入到保留存储器模块内。

函数

```
uint8 SelfTests_Stack_Check(void)
Returns:  0   No error
          1   Error detected

Located in:  SelfTest_Stack.c
             SelfTest_Stack.h
```

运行时，将定期调用该函数，以对堆栈溢出进行测试。模块大小应该是一个偶数值，通过使用位于 *SelfTest_Stack.h* 的宏可以修改该数值：

```
#define STACK_TEST_BLOCK_SIZE 0x08u
```

通过使用位于 *SelfTest_Stack.h* 的宏可以修改模型：

```
#define STACK_TEST_PATTERN 0x55AAu
```

6.9 数字 I/O 测试

PSoC 4 提供了多达 36 个可编程 GPIO 引脚。任何 GPIO 引脚均可作为 CapSense®、LCD、模拟或数字引脚使用。可编程驱动模式、强度和转换速率。

数字 I/O 被安排到各个端口，每个端口最多可安排八个引脚。一些 I/O 引脚被复用为特殊功能（USB、调试端口、晶体振荡器）。通过使用与指定功能相关的控制寄存器，可以使能这些特殊功能。

该测试的目标是确保 I/O 引脚不会与 GND 或 Vcc 短接。

在正常的操作条件下，引脚与地面间以及引脚与 VCC 间的电阻较高。要想检测短接情况，需要将电阻值与 PSoC 内部上拉电阻进行比较。

要想检测引脚与地面间的短接情况，需要将引脚配置为电阻上拉驱动模式。在正常条件下，CPU 会读取某个逻辑引脚的状态，因为该引脚通过电阻上拉。如果引脚通过一个小电阻被接地，那么输入电平被视为逻辑 0。

要想检测传感器与 VCC 间的短接情况，需要将传感器引脚配置为电阻上拉驱动模式。在正常条件下，输入电平为零。

重要提示：该测试取决于应用程序，可能需要定制。默认测试值可能会暂时将引脚配置为终端应用程序的某个错误状态。

函数

```
uint8 SelfTests_IO()
Returns:  0   No error
          1   Error detected (Short to VCC)
          2   Error detected (Short to GND)
Located in:  SelfTest_IO.c
             SelfTest_IO.h
```

调用 `SelfTests_IO()` 函数以检查 I/O 引脚被短路接地还是与 Vcc 短接。`SelfTests_IO()` 函数中的 `PintToTest` 阵列用于设置需要检测的引脚。

例如：

```
static const uint8 PinToTest[] =
{
   0b11011111,/* PORT0 mask */
   0b00111111,/* PORT1 mask */
   0b11111111,/* PORT2 mask */
   0b11110011,/* PORT3 mask */
   0b00000000,/* PORT4 mask */
};
```

每个引脚由 PinToTest 表端口掩码中的相应位表示。LSB 表示引脚 0，MSB 表示引脚 7。如果检测某个引脚， 相应位应被设置为‘1’。

6.10 ADC 和 DAC 测试

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

ADC 测试实现了独立输入比较，如 IEC 60730 标准的 H.2.18.8 章节中的定义。它提供了故障/错误控制技术，通过该技术可以对满足特定容差范围的输入/输出进行比较。

该测试用于检查 ADC 和 IADC 模拟功能。由于 PSoC 4 没有参考电压，所以要使用某个带有外部电阻的 IADC。

通过使用 PSoC 器件的可重新配置硬件和带有外部电阻的 IADC 构成一个参考电压（例如，大小为 4.99 kΩ 且容差为 1%的电阻），这样便能够很轻松地实现该测试。通过该配置，可以提供高达 3.05 V 的参考电压，其步长为 12 mV（相当于 8 位

分辨率）。图 6 显示的是基于 PSoC 4 的 ADC 和 DAC 测试的实现原理图。额外的模拟系统组件提供给可选的电压比较器和运算放大器测试使用。Pin_Opamp1、Pin_Opamp2、Pin_ADC1 和 Pin_ADC2 都是用户使用的引脚。配置 ADC 以便扫描三个通道。前两个通道用于测试，第三个通道供用户使用。可以添加其他用户通道。ADC 通道“0”通过一个运算放大器连接到 DAC，用于运算放大器测试。通道“1”则被直接连接到 DAC，用于进行 ADC 和 DAC 测试。

函数

```
uint8 SelfTest_ADC(void)
Returns:  0   No error
          1   Error detected
Located in:  SelfTest_Analog.h
             SelfTest_Analog.c
```

ADC 准确度定义在 *SelfTest_Analog.h* 中。12 位 ADC 用于测试：

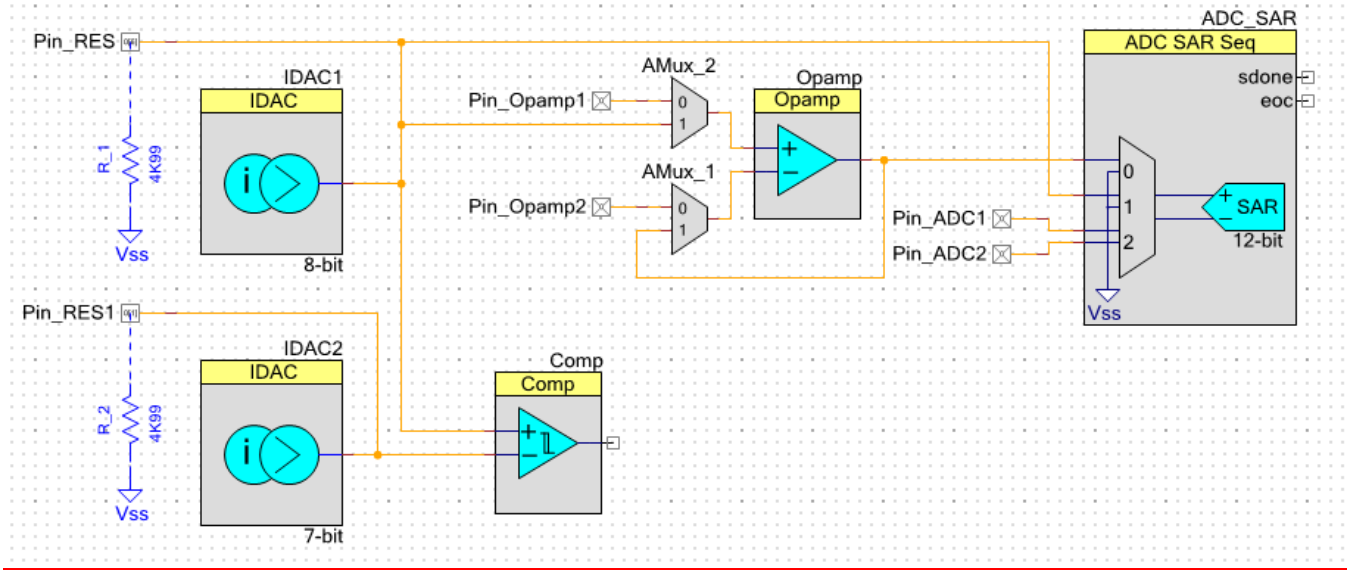
```
#define ADC_TEST_ACC 12                          // +/- ADC result value
```

要想执行该测试，需要配置 ADC 扫描通道“1”。使用 IDAC 来生成预定义（参考）电压，该电压由 ADC 采样。

如果数字化的输入电压值等于满足定义准确度范围的所需参考电压值，那么可以成功实现该测试。如果测试成功，该函数将返回‘0’；否则，它会返回‘1’。

测试前，测试函数会保存全部组件配置， 测试结束后，它将恢复这些配置。

图 6. ADC 和 DAC 测试的 PSoC 实现



6.11 电压比较器测试

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

使用带有外部电阻的 IDAC1 和 IDAC2 执行电压比较器功能测试，以便生成两个参考电压。这些参考电压将被连续放置在模拟电压比较器的输入端，从而强制修改该电压比较器的输出值。输出状态将被读取，然后将该值与预期值进行比较。如果测试成功，该函数将返回‘0’；否则，它会返回‘1’。

函数

```
uint8 SelfTest_Comparator(void)
Returns:  0   No error
          1   Error detected
Located in:  SelfTest_Analog.h
             SelfTest_Analog.c
```

图 6 显示了电压比较器测试中 PSoC 实现的原理图，并包含可选的 ADC、DAC 和运算放大器等测试。在输入信号的不同极性上分析电压比较器的输出值。如果在执行该操作过程中输出信号发生变化，那么可通过测试。

测试前，测试函数会保存 ADC 和 DAC 配置， 测试结束后，它会恢复这些配置。

6.12 运算放大器测试

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

在 IEC 60730 标准中没有提供运算放大器，但在关键模块中使用运算放大器会更加有用。

运算放大器测试遵守“独立输出比较”测试的原则（在 IEC 60730 标准的 H.2.18.8 章节中进行了定义）。它提供了故障/错误控制技术，通过该技术可以对满足特定容差范围的输入/输出进行比较。

该测试用于检测运算放大器模拟功能。通过使用 PSoC 器件的可重新配置硬件，可以轻松实现该测试。另外，还可以使用带有外部电阻的 IADC 构成一个预定义的参考电压来执行该测试。ADC 可以定期转换运算放大器的输出信号，从而测试运算放大器功能。图 6 显示了基于 PSoC 4 的运算放大器测试的实现原理图，并且包含了可选的 ADC、DAC 和电压比较器测试。Pin_Opamp1 和 Pin_Opamp2 都是用户使用的引脚。测试前，按照运算放大器自测试的要求，模拟复用器会断开与用户引脚的运算放大器的连接，并重新配置内部连接。完成测试后，用户引脚的连接将得到恢复。

函数

```
uint8 SelfTest_Opamp(void)
Returns:  0   No error
          1   Error detected
Located in:  SelfTest_Analog.h
             SelfTest_Analog.c
```

运算放大器的准确度定义在 *SelfTest_Analog.h* 中。12 位 ADC 用于测试：

```
#define OPAMP_TEST_ACC 12                // +/- Opamp result value
```

该函数用于实现运算放大器测试，在以下两种情况中使用 ADC 来测量运算放大器的输出信号：

- IDAC1 输出电压为 0.096 V；预期的 ADC 结果为 0.096 V
- IDAC1 输出电压为 1.52 V；预期的 ADC 结果为 1.52 V

如果输入 ADC 电压值等于满足定义准确度范围的预期电压值，那么能够成功实现该测试。 如果测试成功，则该函数会返回‘0’；否则，返回‘1’。

测试结束后，测试函数会使 ADC 和 DAC 配置恢复为默认情况。

6.13 通信 UART 测试

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

该测试实现了 UART 内部数据回送测试。如果传输字节等于接收字节，将返回‘2’，那么测试成功。每次函数调用都会递增测试字节。经过 256 次函数调用，当成功测试完全部 256 个数值时，函数会返回 3。

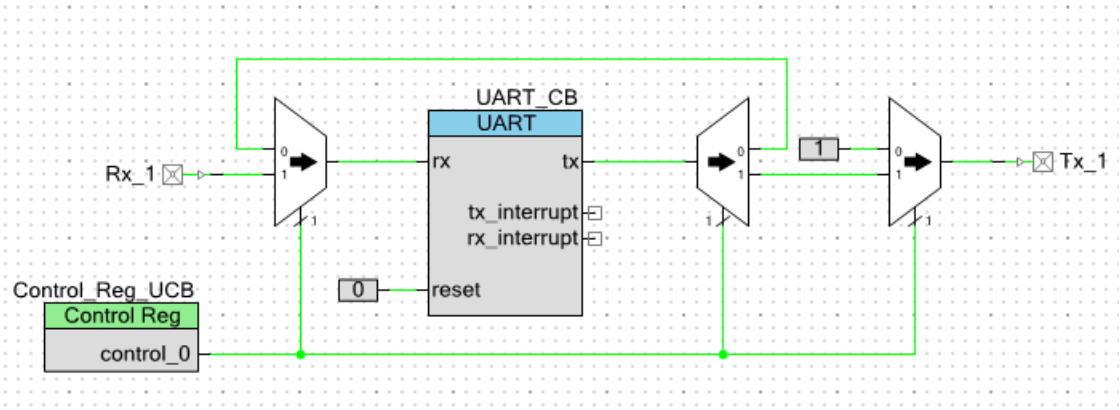
函数

```
uint8 SelfTest_UART(void)
Returns:  1   Error detected
          2   Pass test with current value, but test is not complete testing full range
              from 0x00 to 0xFF
          3   Pass, completed testing full range from 0x00 to 0xFF
          4   ERROR_TX_NOT_EMPTY
          5   ERROR_RX_NOT_EMPTY
          6   ERROR_TX_NOT_ENABLE
          7   ERROR_RX_NOT_ENABLE
Located in:   SelfTest_UART.h
              SelfTest_UART.c
```

图 7 显示的是 UART 测试的 PSoC 实现原理图。输入和输出终端在相应引脚间切换，并循环进行，以便通过使用 UART 复用器和解复用器来提供内部回送测试。如果测试前接收缓冲区或传输缓冲区非空，便不再执行测试，并会返回 ERROR_RX_NOT_EMPTY 或 ERROR_TX_NOT_EMPTY 状态。

测试前，测试函数会保存组件配置， 测试结束后，它将恢复这些配置。在调用期间，函数传输一个字节。每次函数调用后，传输值都会递增。测试值的范围为 0x00 到 0xFF。

图 7. UART 测试的 PSoC 实现



6.14 通信 SPI 测试

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

该测试实现了 SPI 内部数据回送测试。如果传输字节等于接收字节，将返回‘2’，那么表示测试成功。每次函数调用都会递增测试字节。经过 256 次函数调用，当成功测试完全部 256 个数值时，函数会返回 3。

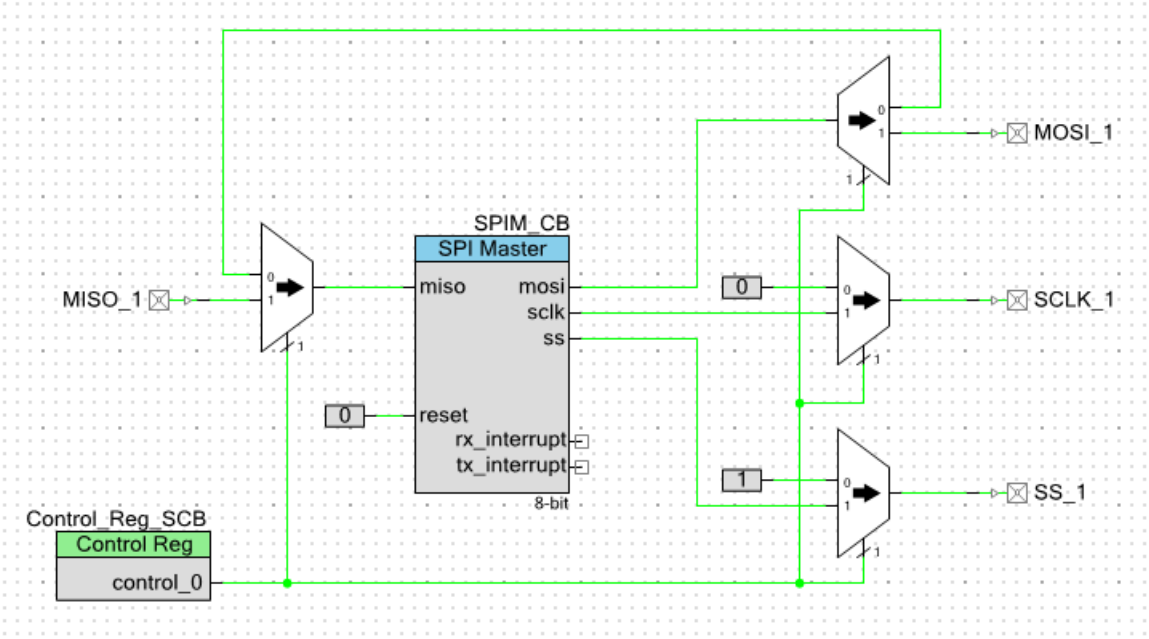
函数

```
uint8 SelfTest_SPI (void)
Returns:  1   Error detected
          2   Pass test with current values, but not all tests in range from 0x00 to 0xFF
              have completed
          3   Pass, tested with all values in range from 0x00 to 0xFF
          4   ERROR_TX_NOT_EMPTY
          5   ERROR_RX_NOT_EMPTY
Located in:   SelfTest_SPI.h
              SelfTest_SPI.c
```

图 8 显示的是 SPI 测试的 PSoC 实现原理图。SPI 输入和输出终端在相应引脚间切换，并循环进行，以便通过使用复用器和解复用器来提供内部回送测试。如果测试前接收缓冲区或传输缓冲区非空，便不再执行测试，并返回 ERROR_RX_NOT_EMPTY 或 ERROR_TX_NOT_EMPTY 状态。

测试前，测试函数会保存全部组件配置， 测试结束后，它将恢复这些配置。调用期间，函数传输了一个字节。每次函数调用后，传输值都会递增。测试值的范围为 0x00 到 0xFF。

图 8. SPI 测试的 PSoC 实现



6.15 UDB 配置寄存器测试

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

UDB 配置寄存器是静态寄存器，在设计构建期间配置它们。在器件进行操作过程中，这些寄存器不会发生变化，可以根据初始配置对它们进行检查。

通过下面各函数，您能够在设计中实现 UDB 配置寄存器测试。它们实现了以下两种测试模式：

- 器件启动后，将 UDB 配置寄存器的副本存入到闪存存储器内。每经过一段时间，都会将配置寄存器与所存副本进行比较。检查后，被破坏的寄存器可以通过闪存中所存的副本进行恢复。
- 如果设置了 CRC 状态信号量，那么将计算得出的 CRC 与先前存储在闪存中的 CRC 进行比较。如果未设置该状态信号量，则必须要计算 CRC，并将该值存入到闪存内，并要设置状态信号量。

函数

```
cystatus SelfTests_Save_UDB_Cfg(void)
Returns:  0   write in flash is successful
          1   error detected during flash writing
Located in:  SelfTest_UDB_CfgReg.c
             SelfTest_UDB_CfgReg.h
```

该函数可将 UDB 配置寄存器存入到闪存存储器内。UDB 配置寄存器位于从 0x400F0000 到 0x40100000 的存储器地址范围内，占用 65K 个字节。但这些寄存器不会占用存储器的全部空间，在该存储器中还存在未使用的间隙。这便是使用位于 `SelfTest_UDB_CfgReg` 的阵列从而获取需要测试的寄存器地址的原因。通过复制项目中 *cydevice_trm.h* 文件内的所有 UDB 寄存器，可生成寄存器阵列。

```
const uint16 UDB_ConfRegs[UDB_REGS_COUNT]

* Number of UDB configuration registers to be counted */
#define UDB_REGS_COUNT 1551
```

需要测试的寄存器数量定义在 SelfTest_UDB_CfgReg.h 中。`UDB_REGS_COUNT` 是从 `cydevice_trm.h` 文件中复制的 UDB 寄存器总数。

注意：在初始 PSoC 上电并初始化后和进入主程序前，应该调用一次该函数。它将 UDB 配置寄存器的正确值写入到闪存内。执行该初始写入操作（通常在制造过程中使用一个测试指令执行的）后，寄存器值已经得到保存，无需再次调用该函数。

注意：存储 UDB 配置寄存器或计算得出 CRC 后，才能调用闪存测试，否则闪存测试会失败。

函数

```
uint8 selfTests_UDB_ConfigReg(uint16 RegsToTest)
Parameters: RegsToTest - number of blocks to be tested per 1 function call. 128 - used in
demo project.
Returns:  1   Error detected
          2   Test in progress
          3   Test completed OK

Located in:  SelfTest_UDB_CfgReg.c
             SelfTest_UDB_CfgReg.h
```

该函数用于检查 UDB 配置寄存器。通过使用 *SelfTest_UDB_CfgReg.h* 文件中的 UDB_REGISTERS_PER_TEST 常量，可以设置每次进行函数调用时需要测试的寄存器数量。应将该常量作为一个参数传输给 `selfTests_UDB_ConfigReg` 函数。

```
#define UDB_REGISTERS_PER_TEST (128u)
```

一共存在两个检查模式：

- CRC-16 计算和验证
- 寄存器与其副本进行比较

通过将 UDB_CFG_REGS_MODE 定义为 *SelfTest_UDB_CfgReg.h* 文件中以下某个常量，您可以定义检查模式：

```
#define UDB_CFG_REGS_MODE CFG_REGS_TO_FLASH_MODE / CFG_REGS_CRC_MODE
#define CFG_REGS_TO_FLASH_MODE (1u)
```

在该模式下，寄存器的副本被存入到闪存内，寄存器将同其副本进行比较。如果收到的数值不同，该函数将返回一条失败信息。在该模式下，可以恢复寄存器。SelfTests_Save_UBD_Cfg()函数用于将寄存器的副本存入到闪存内。寄存器将自动被存入到闪存的最后几行内。

```
#define CFG_REGS_CRC_MODE (0u)
```

在该模式下，该函数将计算寄存器的 CRC-16，并将 CRC 存入到闪存内。然后，函数调用会重新计算 CRC-16，并将计算出的值与所存值进行比较。如果收到的数值不同，它将返回一条失败信息。

6.16 启动配置寄存器测试

该测试介绍并显示了如何检查启动配置寄存器的示例：

1. 测试数字时钟配置寄存器。
2. 测试模拟配置寄存器（启动后，设置为默认值）。
3. 测试 cyfitter 配置寄存器。

这些启动配置寄存器通常是静态的，并且在设计完成之后位于 *cyfitter_cfg.c* 文件中。在极少数用例中，其中一些寄存器可能会动态更新。 动态更新的寄存器必须从此测试中排除。动态寄存器在知道当前正确值的应用程序中进行测试。

两种测试模式都在函数中实现：

- 器件启动后，将启动配置寄存器的副本存入到闪存存储器内。每经过一段时间，将配置寄存器与所存的副本进行比较。检查后，被破坏的寄存器可以通过闪存中的副本进行恢复。
- 如果设置了 CRC 状态信号量，那么将计算得出的 CRC 与先前存储在闪存中的 CRC 进行比较。如果未设置该状态信号量，则必须要计算 CRC，并将该值存入到闪存内，并要设置状态信号量。

注意：以下各函数作为示例，并且仅适用于示例项目。如果您修改了原理图或配置情况，其他配置寄存器可能生成在 *cyfitter_cfg.c* 文件中。您必须修改所需寄存器的列表（推荐的寄存器将生成在 cyfitter_cfg()函数中）。

函数

```
cystatus SelfTests_Save_StartUp_ConfigReg(void)
Returns:  0  write in flash is successful
          1  error detected during flash     writing
Located in:  SelfTest_ConfigRegisters.c
             SelfTest_ConfigRegisters.h
```

该函数会按照需要存储的寄存器数量复制列表中的全部启动配置寄存器，并将它存入到闪存的最后（几）行内。如果为 CFG_REGS_TO_FLASH_MODE 配置 UDB 配置测试，那么启动配置寄存器将立即被存入到闪存（几）行内（位于 UDB 配置寄存器前）。**注意：**初始 PSoC 上电后，应调用一次该函数。进入主程序前，应该初始化该函数。它会将启动配置寄存器的值写入到闪存内。执行该初始写入操作（通常在制造期间）后，寄存器值已经得到保存，无需再次调用该函数。

函数

```
uint8 SelfTests_StartUp_ConfigReg(void)
Returns:     0No error
             1Error detected
Located in:  SelfTest_ConfigRegisters.c
             SelfTest_ConfigRegisters.h
```

该函数用于检查所列出的启动配置寄存器。一共存在两个检查模式：

- CRC-16 计算和验证
- 寄存器与其复制本相比较

可以使用 SelfTest_ConfigRegisters.h 文件中的参数定义模式：

```
#define STARTUP_CFG_REGS_MODE                CFG_REGS_CRC_MODE / CFG_REGS_TO_FLASH_MODE
```

```
#define CFG_REGS_TO_FLASH_MODE (1u)
```

在该模式下，寄存器的副本被存入到闪存内，寄存器将同其副本进行比较。如果收到的数值不同，它将返回一条失败信息（1）。在该模式下，可以恢复寄存器。SelfTests_Save_cfg 函数用于将寄存器的副本存入到闪存内。CONF_REG_FIRST_ROW 参数用于定义启动配置寄存器在闪存储存器中的位置，该参数被自动计算在 SelfTests_Save_cfg.h 中

```
#define CFG_REGS_CRC_MODE (0u)
```

在该模式下，该函数将计算寄存器的 CRC-16，并将 CRC 存入到闪存内。然后，函数调用会重新计算 CRC-16，并将计算出的值与所存值进行比较。如果收到的数值不同，它将返回一条失败信息（1）。

6.17 看门狗测试

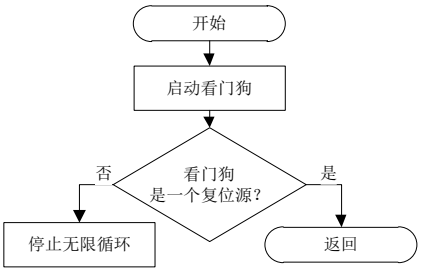
该函数实现了看门狗功能测试。该函数可启动 WDT，并执行无限循环。如果 WDT 正常操作，它会生成一个复位。复位后，该函数会分析复位源。如果看门狗是复位源，那么该函数会返回；否则，它会无限进行循环。

函数

```
void  SelfTest_WDT(void)
Returns:  None
Located in:  SelfTest_WDT.h
             SelfTest_WDT.c
```

图 9 显示的是测试流程图。

图 9. WDT 测试的 PSoC 实现



6.18 窗口模式下的看门狗定时器

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

WDT 提高了基于微处理器系统的可靠性。窗口可选的 WDT 使调整看门狗超时周期成为可能，从而提供更大的灵活性，以满足不同处理器的时序要求。窗口模式下的看门狗电路可以防止系统运行速度过快或过慢。

执行重要功能或有关安全的函数的微处理器要求高级的监控，从而确保能够正确执行故障检测和纠正操作。可以将一个重要函数定义为不允许停机的函数，（多种情况下）它被定义为维修成本较高的函数。可以在微处理器市场的多数分类中发现这种函数：患者监护系统、过程控制 plants 和有关安全的汽车应用。

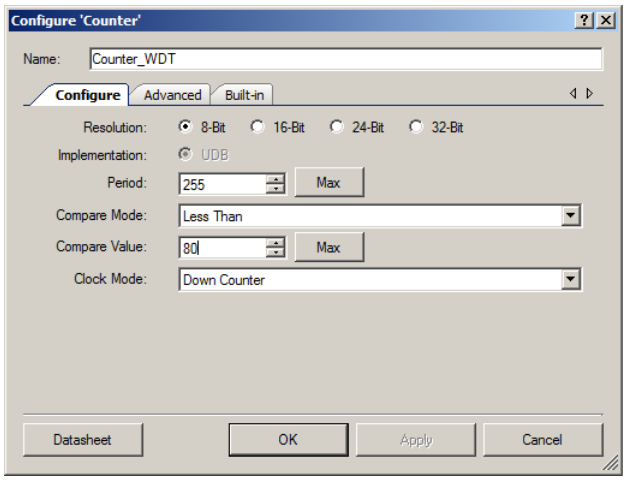
PSoC 4 中不存在窗口模式下的 WDT，但可以通过 PSoC 可重新配置硬件实现。窗口模式下的 WDT 提供了要求执行 ClearWDT 指令的方法，例如，只在看门狗超时周期的最后四分之一时间内提出要求。基本上，这样能够更好地监控代码流，从而检测出固件错误。例如，一个应用程序错误会导致代码流开头的 ClearWDT 指令被重复执行，可以将该错误视为在无窗口 WDT 模式下的正常操作。大多数用户只使用无窗口 WDT 模式。

请注意，在窗口 WDT 模式下，必须在一个规定的窗口中调用 ClearWDT 指令，从而限制驱动 WDT 的时钟源容差。时钟源容差还定义了看门狗周期的最小和最大额定值。

计数器和触发器用于实现 PSoC 4 中窗口模式下的 WDT。

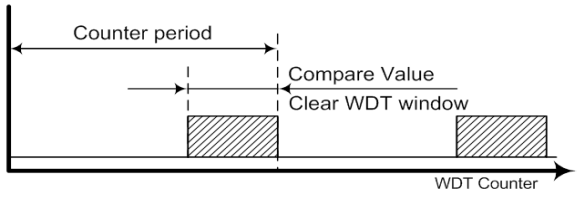
图 10 显示的是窗口模式下 WDT 计数器配置。**Clock_Count**（图 12 中的 Clock_2）和 **Period**（图 10 中所示）用于定义清除指令的最长等待时间（固件执行的最长时间）。**Compare Value**（图 10 所示）用于定义清除窗口的宽度。

图 10. 窗口模式下 WDT 计数器配置



递减计数器输出“comp”将在计数周期内按照下面情况被更改（请参考图 11）：

图 11. 窗口模式下 WDT 时序图



- 第一个窗口：从（255u）到（80u），输出“comp” 为“0”。
- 第二个窗口：从（80u）到（0u），输出“comp” 为“1”。

从（80u）到（0u）的第二个窗口用于检测窗口 WDT 清除是否正确。

如果窗口模式下的 WDT 清除发生在第一个窗口内或并未发生在计数周期内，那么将发生不正确的固件操作，并且必须将 PSoC 复位。

窗口模式下的清除 WDT 意味着在 Control_Reg_ClearWDT 控制寄存器中触发“1”。

图 12 显示的是基于 PSoC 4 的窗口模式下 WDT 测试的实现原理图，一共分三个阶段：

1. 触发器 DFF1 检测清除操作是否在第二个窗口中发生。
2. 触发器 DFF2 检测清除操作是否在第一个窗口中发生。
3. 触发器 DFF3 检测清除操作是否并非发生在计数周期内。

ISR isr_Windowed_WDT 在阶段 2 或 3 中得到触发。它用于检测示例项目中的复位情况，以便进行示范。在客户设计中，可以使用连接到硬件复位的引脚来替代 ISR。

此外，可以对窗口模式下的 WDT 输出以及关键输出进行 ANDed/ORed 运算，从而能够在发生 WDT 事件时禁用这些关键输出。

集成安全输入（如窗口模式下的 WDT 或带有输出逻辑和引脚的互锁输入）会导致 100%的硬件安全控制，而无需 CPU 的处理。

图 13 显示的是窗口模式下的 WDT 操作的流程图。

函数

一共有两个函数用于运行窗口模式下的 WDT：

- `void Windowed_WDT_Start(void)`：This function starts operation of the windowed WDT.
- `void Windowed_WDT_Clear(void)`：This function clears the windowed WDT.

这些函数都位于以下文件内：

- SelfTest_Windowed_WDT.c
- SelfTest_Windowed_WDT.h

图 12. PSoC 4 中的窗口模式下 WDT 实现

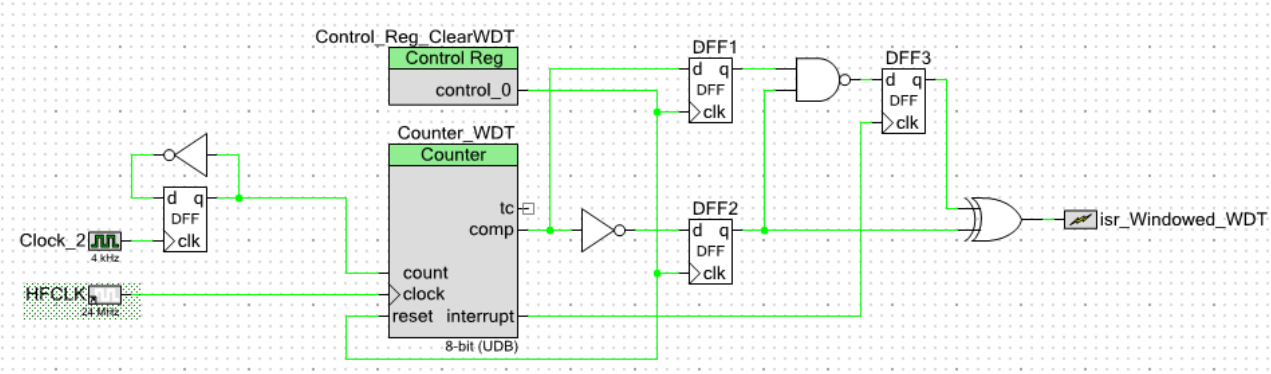
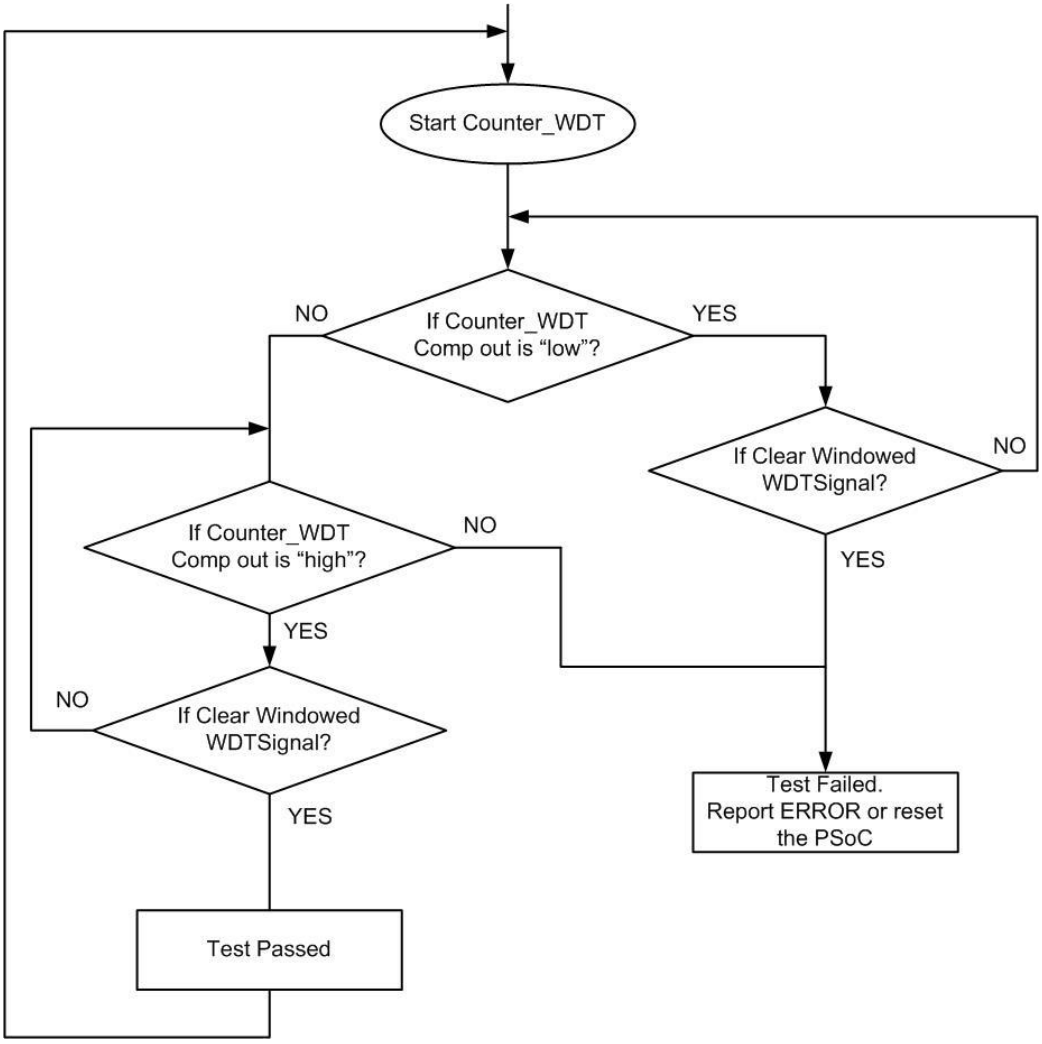


图 13. 窗口模式下的 WDT 操作流程图



6.19 UART 数据传输通信协议示例

注意：仅适用于 PSoC 4100 和 PSoC 4200 系列。

在系统组件间传输数据时，可以使用带有 CRC 和数据包处理的通信协议来确保其他系统安全。下面显示的是安全通信示例。

使用某个传输的定义结构将数据放置在数据包内。所有数据包都有一个使用数据包数据计算得出的 CRC，用以确保数据包的安全传输。图 14 显示了数据包的格式。

图 14. 数据包结构

STX	ADDR	DL	D0	……..（数据字节）	Dn	CRCH	CRCL

要想传输数据包起始标记（STX）的保留值，需要使用一个通用的 Escape 方法。当数据包中所有字节都等于 STX 或 ESC 时，它会变为一个 2 字节的序列。如果数据包字节等于 ESC，那么请用两个字节（ESC，ESC + 1）替换掉它。如果所有数据包字节等于 STX，那么请使用两个字节（ESC，STX + 1）替换掉它。这个程序提供了一个独特的数据包起始符。ESC 字节始终等于 0x1B。它不是数据包的一部分，并且始终在发送（数据包字节+ 1）或（ESC，STX + 1）前发送。表 2 显示的是数据包字段说明。

表 2. 数据包字段说明

名称	长度	数值	说明
STX	1 个字节	0x02	独特的数据包起始标记 = 0x02。
地址	1 个或 2 个字节	0x00…0xFF（0x02 除外）	从设备地址。如果该字节等于 STX，它会变为一个 2 字节的序列：（ESC）+（STX + 1）。如果该字节等于 ESC，它会变为一个 2 字节的序列：（ESC）+（ESC + 1）。
DL	1 个或 2 个字节	0x00…0xFF（0x02 除外）	数据包的数据长度（不带协议字节）。如果该字节等于 STX，它会变为一个 2 字节的序列：（ESC）+（STX + 1）。如果该字节等于 ESC，它会变为一个 2 字节的序列：（ESC）+（ESC + 1）。
D0…Dn（数据）	1…510 个字节	0x00…0xFF（0x02 除外）	数据包的数据。如果数据中任何字节等于 STX，它会变为一个 2 字节的序列：（ESC）+（STX + 1）。如果数据中任何字节等于 ESC，它会变为一个 2 字节的序列：（ESC）+（ESC + 1）。
CRCH	1 个或 2 个字节	0x00…0xFF（0x02 除外）	数据包 CRC 的 MSB。使用了 CRC-16。为所有数据包字节（从 ADDR 到最后数据字节）计算 CRC. 在 ESC 更改程序后，CRC 将被计算。如果该字节等于 STX，它会变为一个 2 字节的序列：（ESC）+（STX + 1）。如果该字节等于 ESC，它会变为一个 2 字节的序列：（ESC）+（ESC + 1）。
CRCL	1 个或 2 个字节	0x00…0xFF（0x02 除外）	数据包 CRC 的 LSB。使用了 CRC-16。为所有数据包字节（从 ADDR 到最后数据字节）计算 CRC. 在 ESC 更改程序后，CRC 将被计算。如果该字节等于 STX，它会变为一个 2 字节的序列：（ESC）+（STX + 1）。如果该字节等于 ESC，它会变为一个 2 字节的序列：（ESC）+（ESC + 1）。

6.19.1 数据供应控制

通信程序可分为以下三个阶段：

- 发送请求（与接收请求相对应）
- 等待响应（与分析请求相对应）
- 接收响应（与发送响应相对应）

“发送请求”阶段包括发送 STX、通过使用字节更改程序发送数据长度和数据、计算 CRC 以及发送 CRC。

“接收响应”阶段包括查找 STX 和开始计算 CRC。如果已接收的地址失效，将重复进行查找 STX 字节。如果该地址有效，那么数据长度和数据字节被接收。然后，CRC 计数器停止，两个 CRC 字节被接收。这些字节同计算得出的 CRC 值进行比较。

发送一个请求后，保护定时器开始检测是否在超时周期内没有收到响应。

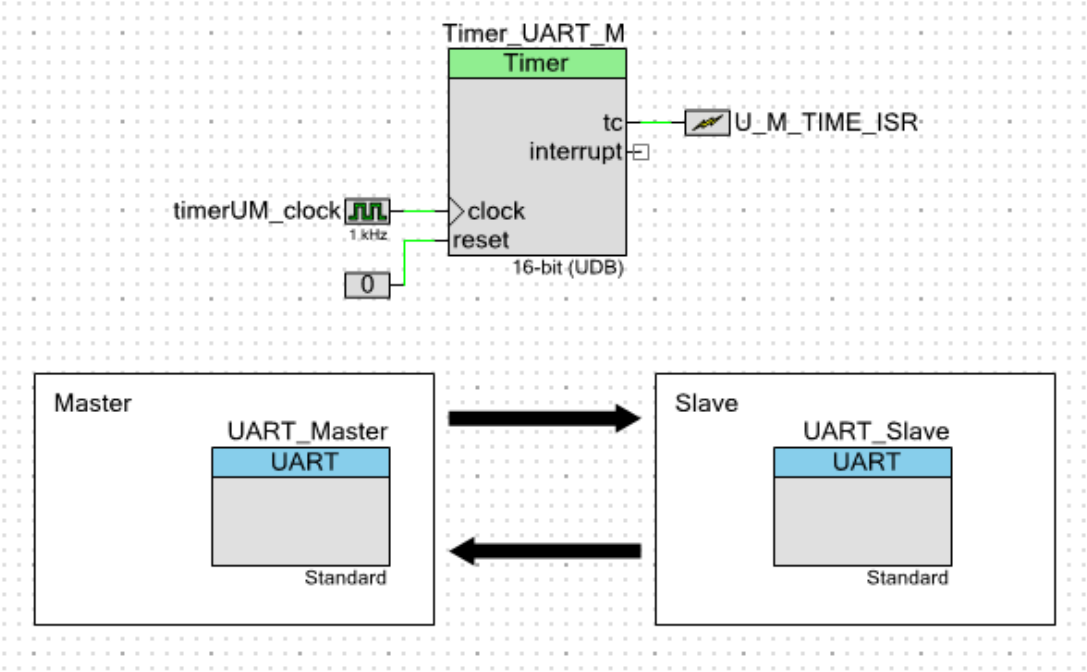
6.19.2 PSoC 实现

图 15 显示的是通过使用 PSoC 进行的协议实现。UART SCB 组件用于物理性生成信号。软件 CRC-16 计算适用于所有发送/接收字节（STX 和 CRC 本身除外）。定时器用于检测不成功的数据包数据传输。

该项目中实现的三个中断提供了一个全中断驱动的后台进程：

- UART 中的传输中断被配置为一个 FIFO 未满事件，用于从 RAM 提取新数据，并将该数据放置在 TX 缓冲区内。此外，该中断还被配置为一个传输完成事件，用以启动或停止 CRC 计算。
- UART 中的接收中断被配置为一个 FIFO 未满的事件，用于分析接收到的数据、计算 CRC 并将接收到的数据存到 RAM 内。
- 定时器中断用于检测没有成功完成数据传输的终端。

图 15. PSoC 协议实现



该软件单元作为一个中断驱动的驱动器得以实现。更确切地说，用户仅启动该流程并检查单元的状态。全部操作都是在后台中完成的。

四个函数用于配合同主设备的协议单元使用

函数 1

```
void UartMesMaster_Init(void)
Returns:  None
Located in:  UART_master_message.h
             UART_master_message.c
```

该函数用于初始化 UART 信息单元。

函数 2

```
uint8 UartMesMaster_DataProc(uint8 address, char * txd, uint8 tlen, char * rxd, uint8
rlen)
Returns:  0   No error
          1   Error detected
Located in:  UART_master_message.h
             UART_master_message.c
```

该函数会启动传输和接收信息流程并返回流程启动的结果：0 = 成功，1 = 错误。因为单元当前正在发送信息或检测出一个空传输长度，因此会发生错误。

输入参数如下：

- 地址：用于进行通信的从设备地址
- txd：指向传输数据（请求数据）的指针
- tlen：所请求的长度，以字节为单位
- rxd：指针，指向用于存储接收数据（接收数据）的缓冲区
- rlen：接收缓冲区的长度，以字节为单位

函数 3

```
uint8 UartMesMaster_State(void)
Returns:
      0  (UM_ERROR)  -  the  last  transaction    process  finished  with  an  error  and  the
         unit is ready to start a new process
      1  (UM_COMPLETE)  -  the  last  transaction  process  finished  successfully,  the
         received buffer contains a response. The unit is ready to start a new process
      2 (UM_BUSY) - the unit is busy with an active transaction operation.
Located in:  UART_master_message.h
             UART_master_message.c
```

该函数可返回 UART 信息单元的当前状态。

结果可能为：

- UM_ERROR：最终数据传输流程以一个错误结束，单元已经准备好启动一个新的流程。
- UM_COMPLETE：最终数据传输流程成功结束，接收缓冲区包含一个响应。单元准备好启动一个新的流程。
- UM_BUSY：单元正在执行有效数据传输操作。

函数 4

```
uint8 UartMesMaster_GetDataSize(void)
Returns: returns data size
Located in:  UART_master_message.h
             UART_master_message.c
```

该函数会返回存储在接受缓冲区中的接收数据大小。如果单元繁忙或最终流程生成了一个错误，它将返回 0。

五个函数用于配合与从设备的协议单元使用

函数 1

```
void UartMesSlave_Init(uint8 address)
Returns: None
Located in:  UART_slave_message.h
             UART_slave_message.c
```

该函数用于初始化 UART 信息单元。输入参数如下：

- 地址：从设备地址

函数 2

```
uint8 UartMesSlave_Respond(char * txd, uint8 tlen)
```

```
Returns:  0   No error
          1   Error detected
Located in:  UART_slave_message.h
             UART_slave_message.c
```

该函数会启动响应。它会返回流程启动结果。成功时，结果为 0；错误则为 1（该单元未收到标记）。

输入参数如下：

- Txd：指向传输数据（请求数据）的指针
- tlen：所请求的长度，以字节为单位

函数 3

```
uint8 UartMesSlave_State(void)
Returns:
   0 (UM_IDLE) – the last transaction process is finished

   1 (UM_PACKREADY) – the unit has received a marker and there is received data in the
      buffer. The master waits for a response.
   2 (UM_RESPOND) – the unit is busy with sending a response.
Located in:  UART_slave_message.h
             UART_slave_message.c
```

该函数可返回 UART 信息单元的当前状态。结果可能为：

- UM_IDLE：最终数据传输流程结束。
- UM_PACKREADY：单元收到一个标记，缓冲区中存在接收数据。主设备等待一个响应。
- UM_RESPOND：单元忙于发送一个响应。

函数 4

```
uint8 UartMes_GetDataSize(void)
Returns:  returns data size
Located in:  UART_slave_message.h
             UART_slave_message.c
```

该函数用于获取存储在接受缓冲区中的接收数据大小。如果单元状态不为 UM_PACKREADY，它会返回 0。

函数 5

```
uint8 * UartMesSlave_GetDataPtr(void)
Returns: return pointer to data
Located in:  UART_slave_message.h
             UART_slave_message.c
```

该函数包含指向接收数据的指针。

7 总结

本应用笔记介绍了如何实现 IEC 60730 和 IEC 61508 标准定义的诊断测试。在白家电和其他家电产品的设计中，结合这些标准可将消费者的安全性提高到一个新的水平。

通过采取 PSoC 4 所提供的独特硬件配置性优势，设计师可以遵守法规，同时维持或降低电子系统的成本。通过使用 PSoC 和安全软件库能够建立一个强大的系统级开发平台，从而可以实现卓越的性能、快速的上市时间以及能源效率。

8 参考文档

- IEC 60730 标准，“普通家用和类似用途的自动电气控制”，IEC 60730-1 版本 3.2（2007 年 03 月）
- IEC 61508 标准，“电气/电子/可编程电子安全相关系统的功能安全”，IEC 61508-2 版本 2.0（2010 年 04 月）

附录 A：进行测试闪存（恒量存储器）时设置校验和

下面各指令可帮助您编程芯片，以获得正确的闪存和 ROM 诊断测试。

1. 使用校验和值（在 *SelfTest_Flash.c* 文件中被设置为 0x0000）来编译 PSoC Creator 中的项目。

对于 GCC 编译器：

```
volatile const uint64 flash_StoredCheckSum __attribute__ ((used, section ("CheckSum"))) =
0x0000000000000000u;
```

对于 MDK 或 RVD 编译器：

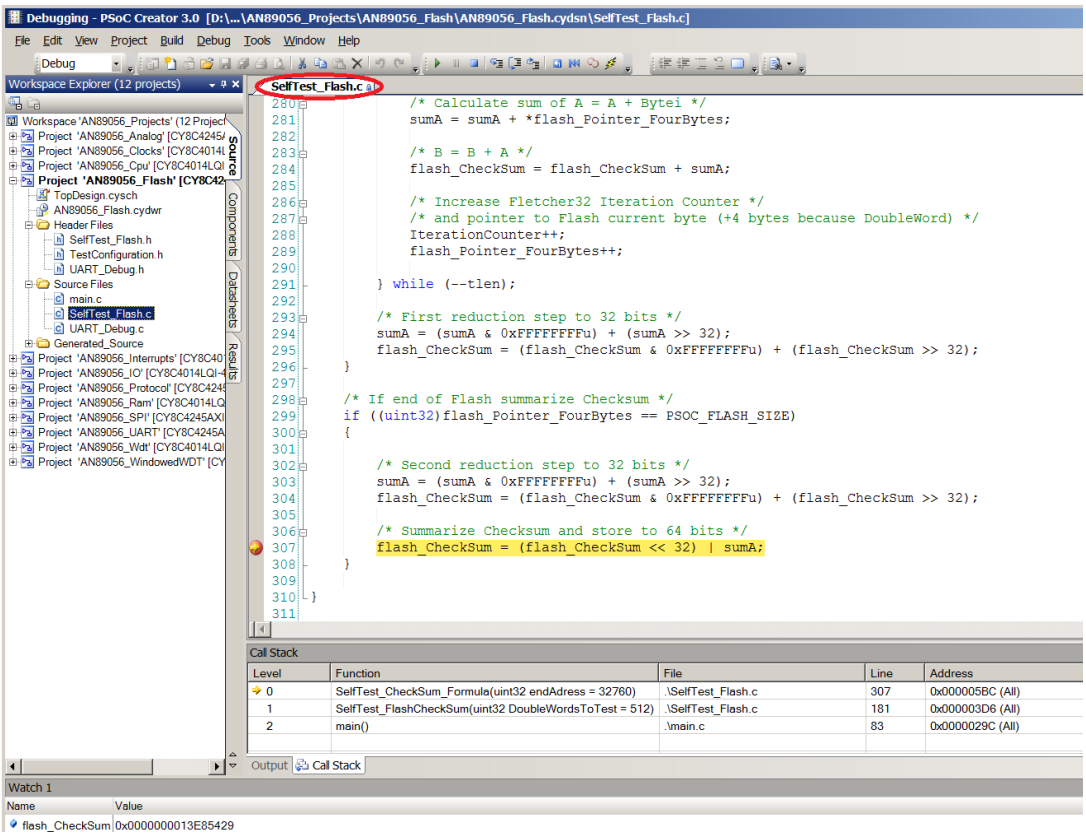
```
#if (CYDEV_CHIP_MEMBER_USED == CYDEV_CHIP_MEMBER_4A)
    volatile const uint64 flash_StoredCheckSum __attribute__ ((at(0x00007FF8))) =
0x0000000000000000;
#endif

   /* CY8C40XX */
#if (CYDEV_CHIP_MEMBER_USED == CYDEV_CHIP_MEMBER_4D)
    volatile const uint32 flash_StoredCheckSum __attribute__ ((at(0x00003FF8))) =
0x0000000000000000;
```

2. 读取所计算的闪存校验和。可通过下述两种方法：

a. 在调试模式中读取校验和。

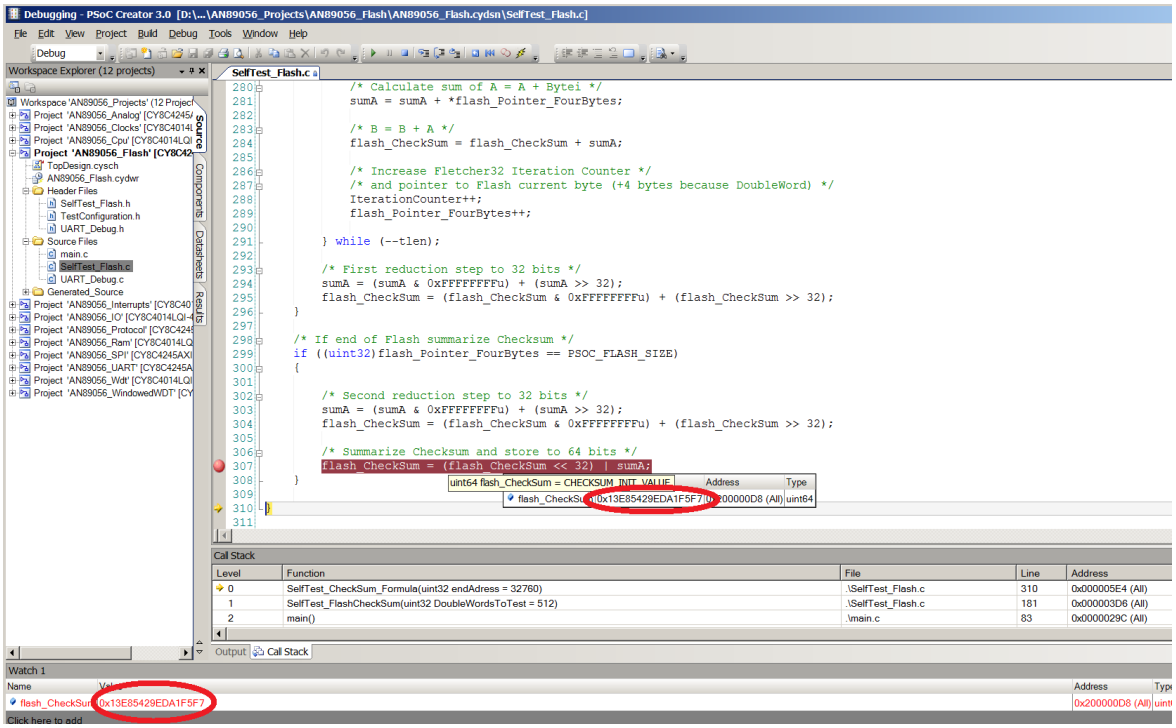
i. 打开项目文件 *SelfTest_Flash.c*，并将调试模式中的断点设置为图 16 所示的行。

图 16. 存储调试模式中的校验和。



ii. 按下**[F10]**，以单步通过断点位置，并将鼠标悬停在变量“flash_CheckSum”。这时会出现该变量中所存储的值，如图 17 所示。

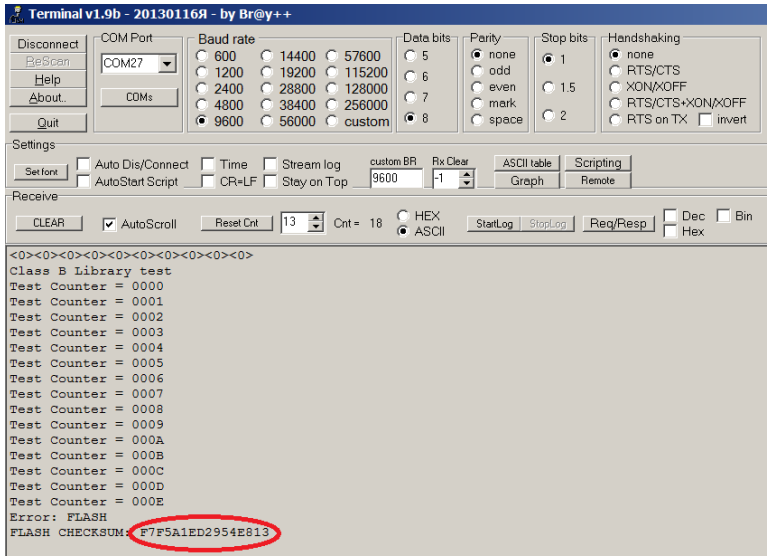
图 17.存储调试模式中的校验和。



b. 使用通信协议读取校验和：

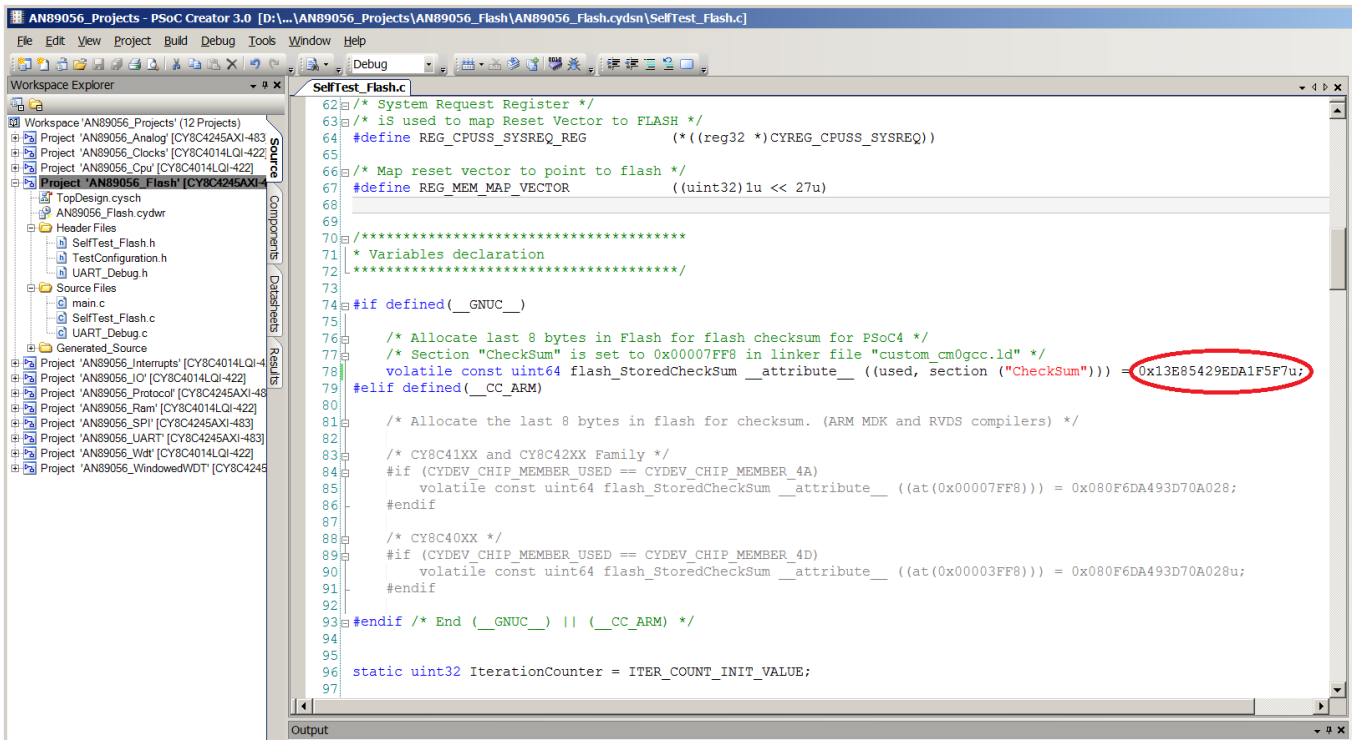
i. 通过使用一个 UART 可加快测试闪存校验和输出的过程。在 B 类固件中实现该特性。当所存储的闪存校验和与计算出的闪存校验和互相不匹配时，它将显示计算出的校验和。要使用该项目，需要设置 UART 参数，如图 18 所示。

图 18. 使用 UART 的校验和输出



3. 复制该校验和值，并将它存储在校验和位置上，*但要注意 PSoC 4 使用了低位优先格式*。GCC 编译器的项目如图 19 所示。

图 19. 使用实际的校验和更新校验和常数



4. 编译项目并编程 PSoC。

附录 B：IECEE CB 计划测试证书

IEC IECEE CB SCHEME

Ref. Certif. No.

US-24905-UL

IEC SYSTEM FOR MUTUAL RECOGNITION OF TEST CERTIFICATES FOR ELECTRICAL EQUIPMENT (IECEE) CB SCHEME

SYSTEME CEI D'ACCEPTATION MUTUELLE DE CERTIFICATS D'ESSAIS DES EQUIPEMENTS ELECTRIQUES (IECEE) METHODE OC

CB TEST CERTIFICATE

CERTIFICAT D'ESSAI OC

Product Produit	Integrated, Protective Control with Type 2 action (Self-Test Software Library – Safety Control)
Name and address of the applicant Nom et adresse du demandeur	Cypress Semiconductor 2700 162nd St. SW, Lynnwood, CA 98087 USA
Name and address of the manufacturer Nom et adresse du fabricant	Cypress Semiconductor 2700 162nd St. SW, Lynnwood, CA 98087 USA
Name and address of the factory Nom et adresse de l'usine Note: When more than one factory, please report on page 2 Note: Lorsque il y plus d'une usine, veuillez utiliser la 2ème page	Cypress Semiconductor 2700 162nd St. SW Lynnwood, WA 98087 USA ☐ Additional Information on page 2
Ratings and principal characteristics Valeurs nominales et caractéristiques principales	N/A
Trademark (if any) Marque de fabrique (si elle existe) Type of Manufacturer's Testing Laboratories used Type de programme du laboratoire d'essais constructeur	Cypress Semiconductor
Model / Type Ref. Ref. De type	PSoC 4 IEC 60730 Class B Safety Software Library (AN89056), version 1.0
Additional Information (if necessary may also be reported on page 2) Les informations complémentaires (si nécessaire,, peuvent être indiqués sur la 2ème page	☐ Additional Information on page 2
A sample of the product was tested and found to be in conformity with Un échantillon de ce produit a été essayé et a été considéré conforme à la	IEC 60730-1(ed.4)
As shown in the Test Report Ref. No. which forms part of this Certificate Comme indiqué dans le Rapport d'essais numéro de référence qui constitue partie de ce Certificat	4786782869-20150325 issued on 2015-03-25

This CB Test Certificate is issued by the National Certification Body
Ce Certificat d'essai OC est établi par l'Organisme **National de Certification**

UL

☒ UL (US), 333 Pfingsten Rd IL 60062, Northbrook, USA
☐ UL (Demko), Borupvang 5A DK-2750 Ballerup, DENMARK
☐ UL (JP), Marunouchi Trust Tower Main Building 6F, 1-8-3 Marunouchi, Chiyoda-ku, Tokyo 100-0005, JAPAN
☐ UL (CA), 7 Underwriters Road, Toronto, M1R 3B4 Ontario, CANADA

For full local entity names see www.ul.com/ncbnames

Date: 2015-03-30

Signature:

Jolanta M. Wroblewska

1/1

附录 C：受支持的芯片型号列表

注意：SPI、UART 以及窗口模式下的 WDT 自测试使用的是 UDB 模块，并且只支持 UDB 配备的芯片。PSoC 4200M 系列器件需要 PSoC Creator 3.2 或更高版本，这些在认证库中不支持。

PSoC 4	
CY8C4013LQI-411	CY8C4244LQI-443
CY8C4013SXI-400	CY8C4244LQQ-443
CY8C4013SXI-410	CY8C4244PVI-432
CY8C4013SXI-411	CY8C4244PVI-442
CY8C4014LQI-412	CY8C4244PVQ-432
CY8C4014LQI-421	CY8C4244PVQ-442
CY8C4014LQI-422	CY8C4245AXI-473
CY8C4014LQI-SLT1	CY8C4245AXI-483
CY8C4014LQI-SLT2	CY8C4245AXQ-473
CY8C4014SXI-411	CY8C4245AXQ-483
CY8C4014SXI-420	CY8C4245LQI-483
CY8C4014SXI-421	CY8C4245LQQ-483
CY8C4124AXI-433	CY8C4245PVI-482
CY8C4124AXQ-433	CY8C4245PVQ-482
CY8C4124LQI-433	CY8C4245AZI-M433 (库项目不支持。)
CY8C4124LQQ-433	CY8C4245AZI-M443 (库项目不支持。)
CY8C4124PVI-432	CY8C4245AZI-M445 (库项目不支持。)
CY8C4124PVI-442	CY8C4245LTI-M445 (库项目不支持。)
CY8C4124PVQ-432	CY8C4245AXI-M445 (库项目不支持。)
CY8C4124PVQ-442	CY8C4246AZI-M443 (库项目不支持。)
CY8C4125AXI-473	CY8C4246AZI-M445 (库项目不支持。)
CY8C4125AXI-483	CY8C4246AZI-M475 (库项目不支持。)
CY8C4125AXQ-473	CY8C4246LTI-M445 (库项目不支持。)
CY8C4125AXQ-483	CY8C4246LTI-M475 (库项目不支持。)
CY8C4125LQI-483	CY8C4246AXI-M445 (库项目不支持。)
CY8C4125LQQ-483	CY8C4247LTI-M475 (库项目不支持。)
CY8C4125PVI-482	CY8C4247AZI-M475 (库项目不支持。)
CY8C4125PVQ-482	CY8C4247AZI-M485 (库项目不支持。)
CY8C4244AXI-443	CY8C4247AXI-M485 (库项目不支持。)
CY8C4244AXQ-443	

附录 D：MISRA 合规性

本附录中的各个表格提供了用于测试项目的 MISRA-C:2004 合规性和偏差的详细信息。

汽车工业软件可靠性协会（MISRA）规范包括一套 122 个强制性规则和 20 个参考规则，这些内容适用于固件设计。汽车产业对它进行编译，从而能够提高汽车级设备中所嵌入的固件代码的质量和稳定性。

表 3. 验证环境

组件	名称	版本
测试规则	针对 C 语言在关键系统中所使用的 MISRA-C:2004 准则。	2004 年 10 月
目标器件	PSoC 4	量产
	PSoC 4	量产
目标编译器	PK51	9.03
	GCC	4.8.4
生成工具	PSoC Creator	3.1
MISRA 检查工具	Windows 编程研究 QA C 源代码分析器	8.1-R
	编程研究 QA C MISRA-C:2004 合规性模块（M2CM）	3.2

表 4. 偏差规则

MISRA-C:2004 规则	规则类别（R/A）	规则说明	偏差说明
3.1	R	所有实现定义行为的使用情况都应被记录下来。	对于 PK51 和 GCC 编译器的文档，请分别参考“PSoC Creator 帮助”菜单、“文档”子菜单、“Keil”和“GCC”指令。
8.7	R	如果只能从单个函数中访问对象，那么应该在模块范围内定义这些对象。	在 ISR 程序中访问易失性的全局变量。
8.8	R	外部对象或函数只能在一个文件中声明。	对于 PSoC 4，通过*.c 文件中的外部链接可声明某些对象，这些声明并不在头文件内。
8.10	R	除非要求外部链接，否则文件范围内的对象或函数的所有声明和定义都应具有内部链接。	库 API 设计用于某个用户应用，不可用于其它库 API。
10.1	R	在下面情况中，整型表达式的数值不应该毫无保留地转换为不同的底层类型：a) 没有转换到更宽的符号相同的整数类型，或 b) 表达式是复合的，或 c) 表达式并不是一个常量，是一个函数参数，或 d) 表达式不是一个常量，是一个返回表达式。	*SelfTest_Analog.c* 文件：在算术运算中使用了 uint16 `DAC_Offset` 变量。
11.3	A	不应对一种指针类型和一种整数类型进行转换。	请参见表 5。
11.4	A	指向对象类型的不同指针之间不应进行转换。	请参见表 5。
11.5	R	不被执行的转换，该转换会清除指针寻址类型中的任何常量或易失性资质。	请参见表 5。
13.6	R	在“for”循环中未被用于迭代计数的数字变量在循环体（loop body）中不会被修改。	文件 *SelfTest_UDB_CfgReg.c*、函数 `SelfTests_UDB_ConfigReg`:在循环体中 iblock 变量为零，以便开始测试下一个闪存模块。

MISRA-C:2004 规则	规则类别（R/A）	规则说明	偏差说明
13.7	R	不允许结果不变的 Boolean 运算。	分析器会将某些 Boolean 运算错误地视为“不变”。这些都是误报的。库 API 设计用于用户应用，不可用于库演示代码。
14.1	R	没有不可访问的代码。	库 API 设计用于用户应用，不可用于库代码。
14.3	R	预处理前，空语句只能单独一行出现；可以在它后面的一个空格后添加注释。	CyGlobalIntEnable 宏包含紧挨着其他代码的空语句。
15.2	R	无条件的 break 语句会终止所有非空的 switch 子句。	文件 *SelfTest_Clock.c*、函数 `SelfTests_Clock`：状态机实现需要使用 switch case，而无需“break”语句。
16.8	R	具有非 void 返回类型的函数的所有退出路径要包含一个明确的“return”语句，该语句可返回一个表达式。	文件 *SelfTest_CPU_Regs.c*、函数 `SelfTest_CPU_Regs`：通过 ASM 指令返回测试结果。
16.9	R	一个函数标识符应该与一个“&”参数或与带挂号参数表（该参数表可以为空）一起使用。	文件 *uart_master_message.c* 和 *uart_slave_message.c* `UTX_M_ISR_SetVector`, `URX_M_ISR_SetVector`, `U_M_TIME_ISR_SetVector`, `UTX_S_ISR_SetVector`,以及 `URX_S_ISR_SetVector` 等函数会将函数名称作为输入参数。
16.10	R	如果某个函数返回错误信息，那么会对该错误信息进行测试。	库函数返回在演示项目中没有被使用但可在用户项目中使用的值。
17.4	R	阵列索引是唯一一种允许的指针运算形式。	请参见表 5。
21.1	R	应至少通过使用以下某项，从而确保最大程度地减少运行时发生的故障： a) 静态分析工具/技术 b) 动态分析工具/技术 c) 用于处理运行时故障的检查的明确编码	由于所执行的是广义的实现方法，在某些特定的配置中，PSoC Creator 生成的一些代码可能包含冗余操作。

表 5. 演示项目中的指针违规

指针（变量名称）	MISRA 规则	文件	说明
**.h* 文件中的所有寄存器定义都被用在库 API 中。	11.3 不应对一种指针类型和一种整数类型进行转换。	**AN78175_Memory 项目：** *Main.c*、*SelfTest_cpu_asm.c*、*SelfTest_crc_calc.c*、*SelfTest_CustomFlash.c*、*SelfTest_EEPROM.c*、*SelfTest_Ram.c*、*SelfTest_Flash.c*、*SelfTest_Stack.c*、*SelfTest_UDB_CfgReg.c* **AN78175_Analog 项目：** *Main.c*、*idac8_cb.c*、*elfTest_Analog.c*、*SelfTest_Analog_Calibration.c*、*vdac8_cb.c* **AN78175_Digital 项目：** *Main.c*、*SelfTest_IO.c* **AN78175_Protocol 项目：** *Main.c*、*uart_master_message.c*、*Uart_slave_message.c* **AN78175_Wdt 项目：** *Main.c* **WDT_Window 项目：** *Main.c*	通过指向这些寄存器的指针来访问硬件寄存器。
***SelfTest_Flash.c*：** Flash_Pointer ***Main.c*：** rxd	11.4 指向不同对象类型的指针间不能相互转换。	**AN78175_Memory 项目：** *SelfTest_Flash.c* **AN78175_Protocol 项目：** *main.c*	将 **static uint8 CYCODE *Flash_Pointer** 转换为**(*((uint16 code *)Flash_Pointer))**，以便对每个指令进行 uint16 字的访问。 **LCD_Char_PrintString** 库函数需要时，将 **uint8 rxd[16u]**转换为 **(char8*) rxd)**。
***uart_slave_message.c*：** UMS.message;	11.5 未被执行的转换，该转换会清除指针寻址类型中的所有常量或易失性资质。	**AN78175_Protocol 项目：** *uart_slave_message.c*	将 **uint8 message[MAX_MESSAGE_SIZE]** 转换为**(uint8 *)UMS.message**。
***SelfTest_CRC_calc.c*：** RegPointer ***SelfTest_CustomFlash.c*：** rowData ***SelfTest_EEPROM.c*：** RegPointer、RegPointer1 ***SelfTest_Flash.c*：** Flash_Pointer ***SelfTest_Stack.c*：** stack ***SelfTest_UDB_CfgReg.c*：** CfgRegPointer ***uart_master_message.c*：** UM.rxptr ***uart_slave_message.c*：** UMS.txptr	17.4 阵列索引是唯一允许的指针运算形式。	**AN78175_Memory 项目：** *SelfTest_CRC_calc.c*、*SelfTest_CustomFlash.c*、*SelfTest_EEPROM.c*、*SelfTest_Flash.c*、*SelfTest_Stack.c*、*SelfTest_UDB_CfgReg.c* **AN78175_Protocol 项目：** *uart_master_message.c*、*uart_slave_message.c*	**uint8 * rowData** 作为一个参数被传输给作为索引数组访问的函数。 **reg8 * RegPointer** 和 **reg8 * RegPointer1** 作为索引数组使用，这样能够访问寄存器集。 **static uint8 CYCODE *Flash_Pointer** 作为闪存数据的数组使用，并通过“++”运算得到索引。 **uint16 *stack** 作为一个数组使用，并通过一个“++”运算得到索引。 **uint8 CYCODE *** CfgRegPointer 作为一个索引数组使用，用于访问闪存中存储的数据。 **uint8 *txptr** 作为一个数组使用，并通过一个“++”运算得到索引。

文档修订记录

文档标题：AN89056 — PSoC 4 – IEC 60730 Class B 和 IEC 61508 SIL 安全软件库

文档编号：001-98000

版本	ECN	变更者	提交日期	变更说明
**	4802492	LISZ	07/03/2015	本文档版本号为 Rev**，译自英文版 001-89056 Rev**。
*A	6241840	SSAS	07/12/2018	本文档版本号为 Rev*A，译自英文版 001-89056 Rev*C。

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

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，2015-2018 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。