PSoC® Creator™

系统参考指南

cy_boot 组件 v3.40

Document Number: 001-87260, Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intl): 408.943.2600
http://www.cypress.com

赛普拉斯半导体公司，2013-2016 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。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，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。

目录

1 简介 5
格式 6
参考 6
固件源代码示例 6
修订记录 6

2 标准类型和定义 7
基本类型 7
硬件寄存器类型 7
编译器定义 7
Keil 8051 兼容性定义 8
返回代码 8
中断类型和宏 9
内部定义 9
设备版本定义 9
变量属性 9

3 时钟 11
PSoC Creator 时钟实施 11
API 20

4 电源管理 32
PSoC 3/PSoC 5/PSoC 5LP 实现 32
PSoC 4 实现 41
实例低功耗 API 44

5 中断 46
API 46

6 引脚 50
PSoC 3/PSoC 5/PSoC 5LP API 50
PSoC 4 API 52

7 寄存器访问 54
API 54

8 DMA 58

9 闪存和 EEPROM 60
PSoC 3/PSoC 5/PSoC 5LP 实现 60
PSoC 4 实现 67

10 引导加载程序移植 70
简介 70
移植引导加载程序设计 71
移植可引导加载设计 71
项目应用类型属性更改 72
移植可引导加载依赖关系至引导加载程序设计 72

11 系统函数 74
通用 API 74
CyDelay API 75
PSoC 3/PSoC 5/PSoC 5LP 电压检测 APIs 76
PSoC 4 电压检测 API 80
缓存功能 81

12 启动和连接 83
PSoC 3 83
PSoC 4/PSoC 5/PSoC 5LP 84
复位状态保存 (PSoC 3/PSoC 5/PSoC 5LP) 86

13 看门狗定时器 89
PSoC 3/PSoC 5/PSoC 5LP API 89
PSoC 4 API 90

14 MISRA 合规性 96
验证环境 96
项目偏差 97
文档相关规则 98
PSoC Creator 生成源偏差 98
cy_boot 组件特定偏差 99

15 cy_boot 组件更改 101
3.40 版 101
3.30 版 101
3.20 版 102
3.10 版 102
3.0 版 102
2.40 版和较早版本 104

1 简介

本《系统参考指南》将介绍 PSoC Creator cy_boot 组件所提供的函数。cy_boot 组件可为项目提供系统功能，以便其更好地访问并利用芯片资源。虽然这些函数并非来自组件库，但也可以为其所用。您可以通过函数调用可靠地执行所需芯片功能。

cy_boot 组件的独特之处在于：

- 自动添加到每个工程
- 只可显示单个实例
- 无符号表示
- 不显示在组件目录中（默认情况下）

与系统组件一样，cy_boot 包含多项库功能。本指南主要介绍以下功能：

- DMA（不适用于 PSoC 4）
- 闪存
- 时钟
- 电源管理
- 启动代码
- 多种库函数
- 连接器脚本

通过该 cy_boot 组件提供的 API，用户固件可完成本指南中所述的任务。下文分别介绍了多种主要功能区域。

格式

下表列出了本指南使用的格式：

格式	用途
`Courier New`	显示文件位置和源代码： `C:\ …cd\icc\`，用户输入文本
斜体	显示文件名和参考文档： *sourcefile.hex*
[方括号、粗体]	显示程序中的键盘命令： **[Enter]** 或 **[Ctrl] [C]**
File(文件) > New Project(新建项目)	表示菜单路径： **File(文件) > New Project(新建项目) > Clone(复制)**
粗体	显示程序中的命令、菜单路径和选项以及图标名称： 单击 **Debugger**（调试器）图标，然后单击**下一步**。
文本显示在灰色框中	显示仅针对 PSoC Creator 或 PSoC 器件的警告和功能。

参考

本指南是关于 PSoC Creator 和 PSoC 组件的一系列文档之一。如需要，请参考以下其他文档：

- PSoC Creator 帮助
- PSoC Creator 组件数据手册
- PSoC Creator 组件作者指南
- PSoC 技术参考手册 (Technical Reference Manual, TRM)

固件源代码示例

PSoC Creator 在“查找示例工程(Find Example Project)”对话框中提供了大量包括原理图和代码的例子工程。要获取组件特定的示例工程，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开 Start Page（开始页）或 **File**（文件）菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（筛选选项）可缩小可选工程的列表。

有关更多信息，请参见 PSoC Creator 帮助中的“Find Example Project（查找示例工程）”主题。

修订记录

文档标题：PSoC® Creator™ 系统参考指南， cy_boot 组件 v3.40 **文档编号：001-85653**		
修订版	**日期**	**变更说明**
**	4/3/13	3.40 版 cy_boot 组件的新文档。有关 cy_boot 较之先前版本的组件变更，请参考变更一节。
*A	4/11/16	Minor document edits.

2 标准类型和定义

为了支持不同 CPU 所支持的不同编译器使用相同的编程代码，cy_boot 组件提供了可以在多个平台之间兼容使用的数据类型和定义。

基本类型

类型	说明
char8	8 位（有符号或无符号，具体取决于 char 的编译器选择）
uint8	8 位无符号
uint16	16 位无符号
uint32	32 位无符号
int8	8 位有符号
int16	16 位有符号
int32	32 位有符号
float32	32 位浮点
float64	64 位浮点（不适用于 PSoC 3）
int64	64 位有符号（不适用于 PSoC 3）
uint64	64 位无符号（不适用于 PSoC 3）

硬件寄存器类型

硬件寄存器通常会有副作用，因此归类为易失性类型。

定义	说明
reg8	易失性 8 位无符号
reg16	易失性 16 位无符号
reg32	易失性 32 位无符号

编译器定义

所使用的编译器可通过测试具体编译器的定义来确定。例如，要测试 PSoC 3 Keil 编译器的定义：

```
#if defined(__C51__)
```

定义	说明
__C51__	Keil 8051 编译器

__GNUC__	ARM GCC 编译器
__ARMCC_VERSION	Keil MDK 和 RVDS 工具集所使用的 ARM Realview 编译器

Keil 8051 兼容性定义

Keil 8051 编译器支持此平台独有的类型修饰符。其他平台中不得出现这些修饰符。为实现兼容性，对 Keil 进行编译时映射至适当字符串而在对其他平台进行编译时映射到空字符串的定义支持这些类型。这些定义用于创建优化的 Keil 8051 代码，但同时仍支持其他平台上进行的编译。

定义	Keil 类型	其他平台
CYBDTA	bdata	
CYBIT	bit	uint8
CYCODE	code	
CYCOMPACT	compact	
CYDATA	data	
CYFAR	far	
CYIDATA	idata	
CYLARGE	large	
CYPDATA	pdata	
CYREENTRANT	reentrant	
CYSMALL	small	
CYXDATA	xdata	

返回代码

赛普拉斯子程序的返回代码将作为 8 位无符号值返回：cystatus。标准返回值为：

定义	说明
CYRET_SUCCESS	成功
CYRET_UNKNOWN	未知故障
CYRET_BAD_PARAM	一个或多个无效参数
CYRET_INVALID_OBJECT	已指定的对象无效
CYRET_MEMORY	存储器相关故障
CYRET_LOCKED	资源锁定故障
CYRET_EMPTY	无更多可用对象
CYRET_BAD_DATA	收到错误数据（CRC 或其他错误校验）
CYRET_STARTED	操作已启动，但不一定已完成
CYRET_FINISHED	操作已完成
CYRET_CANCELED	操作已取消
CYRET_TIMEOUT	操作超时
CYRET_INVALID_STATE	操作未配置或处于错误状态

中断类型和宏

类型和宏提供了跨编译器和跨平台时保持一致的中断服务子程序定义。请注意，函数定义和函数原型所用的宏是不一样的。

函数定义示例：

```
CY_ISR(MyISR)
{
      /* 此处为 ISR 代码 */
}
```

函数原型示例：

```
CY_ISR_PROTO(MyISR);
```

中断矢量地址类型

类型	说明
cyisraddress	中断矢量（ISR 函数地址）

内部定义

定义	说明
CY_NOP	处理器 NOP 指令

芯片版本定义

定义	说明
CY_PSOC3	任何 PSoC 3 芯片
CY_PSOC5	任何 PSoC 5 芯片
CY_PSOC5A	PSoC 5 Device（PSoC 5 芯片）
CY_PSOC5LP	PSoC 5LP Device（PSoC 5LP 芯片）
CY_PSOC4	PSoC 4 Device（PSoC 4 芯片）

变量属性

定义	说明
CY_NOINIT	指定将静态变量置于未初始化数据段中，以防该变量在启动时被初始化为“0”。 对于 PSoC 3，将不对该宏生成任何代码。

此页特意留白。

3 时钟

PSoC Creator 时钟实施

PSoC Creator 所支持的 PSoC 芯片具有灵活的时钟功能。这些时钟功能在 PSoC Creator 中进行控制，具体控制方法有：配置设计范围资源（Design-Wide Resources）设置中的选项、设计原理图上时钟信号的连性以及在芯片运行期间可修改时钟周期的 API。

本部分介绍了 PSoC Creator 将时钟分配到器件的方法，并提供了针对 PSoC 结构进行优化了的时钟配置方法的相关指导。

PSoC 3、PSoC 5 和 PSoC 5LP 器件的时钟架构与 PSoC 4 器件的不同。系统时钟将 PSoC 3、PSoC 5 和 PSoC 5LP 器件上的总线时钟 (BUS_CLK) 以及 PSoC 4 器件上的系统时钟 (SYSCLK) 整合在一起。主时钟（Master Clock）将 PSoC 3、PSoC 5 和 PSoC 5LP 器件上的主时钟 (MASTER_CLK) 以及 PSoC 4 器件上的 HFCLK 整合在一起。

时钟连接性

PSoC 结构包含灵活的时钟生成逻辑。有关特定器件中所有可用时钟源的详细说明，请参考*技术参考手册*。可根据这些时钟连接到设计中的组件的方法对各种时钟源的应用进行分类。

系统时钟

这是一种特殊的时钟。它与主组件时钟密切相关。在大多数设计中，主时钟（Master Clock）和系统时钟频率相同，且被视为同一时钟。它们应该是系统中速度最高的时钟。CPU 会脱离系统时钟运行，且所有外设均会使用系统时钟与 CPU 和 DMA 通信。同步某时钟时，该时钟将与主时钟同步。同步某引脚时，该引脚将与系统时钟同步。

全局时钟

这种时钟置于全局低时滞数字时钟线上。系统时钟也属于此类时钟。使用时钟组件创建时钟时，该时钟就会创建为全局时钟。该时钟必须直接连接至时钟输入，或在连接时钟输入前可进行反相。全局时钟线仅连接至 PSoC 中数字组件的时钟输入。如果全局时钟线连接的不是时钟输入（即，组合逻辑或引脚），则不会使用低时滞时钟线发送信号。

路由时钟

除全局时钟外的所有时钟都是路由时钟。这种时钟包括逻辑（单个反相器除外）生成的时钟和来自引脚的时钟。

时钟同步

PSoC 器件中的时钟分为同步时钟和异步时钟两种。它与系统时钟和主时钟（Master Clock）相关。PSoC 运行的时候是一个同步系统。其目的在于实现可编程逻辑与 CPU 或 DMA 之间的通信。如果这几者没有与普通时钟同步，则电路中的任何通信都会需要时钟交叉电路。通常，只有不与 CPU 系统交互的 PLD 逻辑支持异步时钟，其他均不支持。

同步时钟 (PSoC 3/PSoC 5/PSoC 5LP)

同步时钟示例：

- 已设置“与主组件时钟同步”选项的全局时钟。该选项为默认设置，位于时钟组件“Configure”（配置）对话框的 **Advanced**（高级）选项卡中。
- 已选择“输入同步”选项的来自输入引脚的时钟。该选项为默认设置，位于引脚组件“Configure”（配置）对话框的 **Input**（输入）选项卡中。
- 由信号组合派生而得到的时钟，且信号全部由同步时钟计时的寄存器生成。

异步时钟 (PSoC 3/PSoC 5/PSoC 5LP)

除同步时钟外的所有时钟都是异步时钟。异步时钟示例：

- 来自数字系统互连 (DSI) 而非同步引脚的任何信号。由于这种信号的时序不确定，因此必须将其视为异步。其中包括：
  - 在用作时钟前通过逻辑馈送的全局时钟（如直接连接至时钟输入）
  - 固定功能模块输出（即，计数器、定时器、PWM）
  - 模拟模块的数字信号
- 未设置同步选项的全局时钟
- 未选择“输入同步”选项的来自输入引脚的时钟
- 使用任何异步信号组合创建的时钟

同步信号 (PSoC 3/PSoC 5/PSoC 5LP)

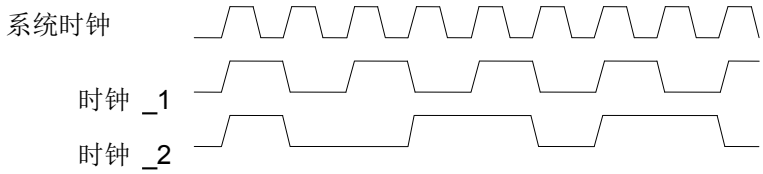
根据时钟信号源的不同，可使用以下不同方法同步信号：

- 对于异步全局时钟，可通过选中时钟组件“配置”对话框中的 **Sync with MASTER_CLK**（与 MASTER_CLK 同步）选项（此选项为默认选项）促成同步。
- 对于来自引脚的路由时钟，可通过选中 **Pins**（引脚）选项卡下引脚组件“Configure”（配置）对话框中的 **Input Synchronized**（输入同步选项）（此选项为默认选项）促成同步。
- 通过使用同步组件并将同步时钟用作时钟信号，可同步任何信号。

同步信号时：

- 相对于正进行同步的信号的频率，同步时钟的频率必须至少是其两倍。如果未满足此要求，则输入时钟沿可能会遗漏，从而未反映在生成的同步时钟内。
- 时钟信号输出的所有切换均发生在同步时钟的上升沿。
- 时钟信号输出的沿将偏离其原始时序。
- 除非输入时钟和同步时钟直接相关联，否则时钟信号输出的高脉冲和低脉冲的宽度会存在差异。

以下示例展示了已与系统时钟同步的两个时钟。Clock_1 的周期恰为系统时钟周期的两倍。Clock_2 的周期约为系统时钟周期的三倍。这导致了在系统时钟一个周期和两个周期之间的高低脉冲宽度的不同。在这两种情况下，所有切换均出现在系统时钟的上升沿。



路由时钟实施

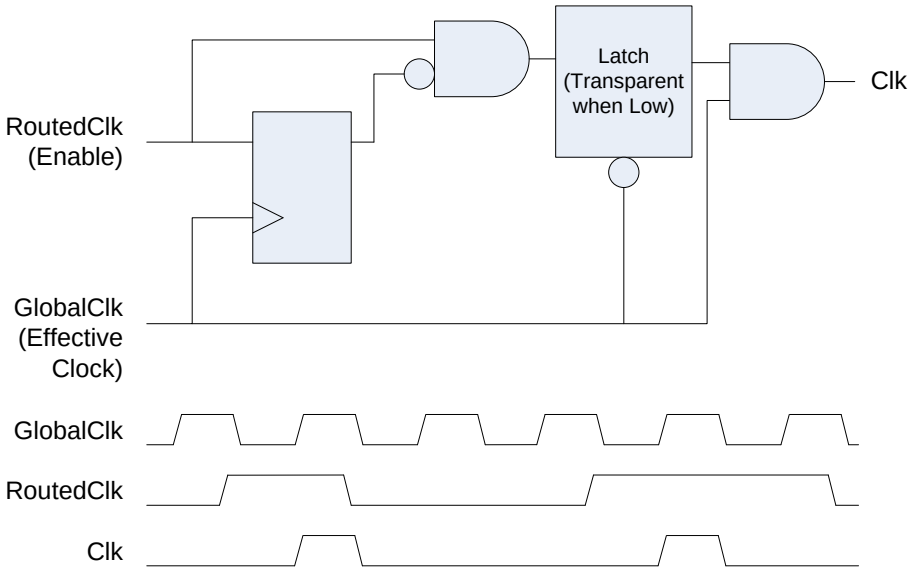
PSoC 中的时钟实施会将全局时钟信号直接连接至计时数字逻辑的时钟输入。这对于同步时钟和异步时钟均适用。因为全局时钟分布在低时滞时钟线上，所以连接至相同全局时钟的计时元素都将在相同的时间开启。

路由时钟使用通用数字路由结构进行分布。这就导致时钟到达每个目的地的时间不同。如果该时钟信号直接被用作时钟，则会强制将该时钟视为异步时钟。这是因为它无法保证在系统时钟的上升沿处跃变。如果相同时钟晚期到达版本计时的寄存器使用由早期到达时钟计时的寄存器的输出，也会导致电路失效。

在一些情况下，PSoC Creator 可以将路由时钟电路转换为使用全局时钟的电路。如果所有路由时钟的源都可以回溯到由通用全局时钟计时的寄存器输出，则 PSoC Creator 就会自动转换此电路。存在这种可能性的情况有：

- 所有信号都派生自相同的全局时钟。这一全局时钟可以是异步时钟，也可以是同步时钟。
- 所有信号派生自不止一个同步全局时钟。在此情况下，通用全局时钟为系统时钟。

PSoC 中的时钟实施包括在此转换中使用的内置沿检测电路。这不会使用 PLD 资源进行实施。下图所示为逻辑实施和产生的时钟时序图。



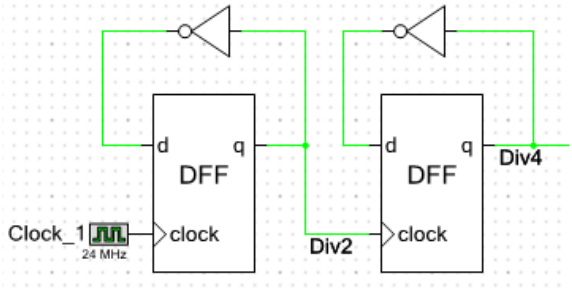
此图显示，生成的时钟在路由时钟的上升沿后于第一时钟上的全局时钟同步发生。

在分析此设计，确定路由时钟源时，可能会引入另一个经过转换的路由时钟。在这种情况下，转换过程中使用的全局时钟被视为该信号的源时钟。

用于每个路由时钟的时钟转换会在报告文件中报告。成功构建后，此文件位于**结果**（Result）选项卡下的“工作区浏览器（Workspace Explorer）”中。详细信息显示在“初始映射（Initial Mapping）”标题下。每个路由时钟都将显示“有效时钟（Effective Clock）”和“使能信号（Enable Signal）”。“有效时钟（Effective Clock）”是使用的全局时钟，“使能信号（Enable Signal）”是检测到沿并用作时钟使能的路由时钟。

以分频时钟为例

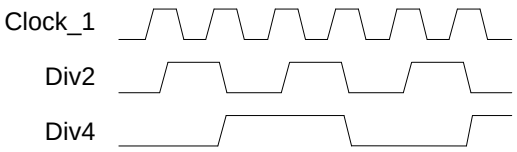
简单的分频时钟电路可用于观察该转换的完成方法。下面的电路使用全局时钟对第一个触发器 (cydff_1) 进行计时。这将生成一个二分频时钟。该信号用作路由时钟，对下一个触发器 (cydff_2) 进行计时。



报告文件表明：使用了一个全局时钟，并且单个的路由时钟已通过使用全局时钟作为有效时钟进行转换。

```
<CYPRESSTAG name="Tech mapping">
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
<CYPRESSTAG name="Global Clock Selection" icon="FILE_RPT_TECHM">
    Digital Clock 0: Automatic-assigning  clock 'Clock_1'. Fanout=1, Signal=tmp__cydff_1_clk
</CYPRESSTAG>
<CYPRESSTAG name="UDB Routed Clock Assignment">
    Routed Clock: tmp__cydff_1_reg:macrocell.q
        Effective Clock: Clock_1
        Enable Signal: tmp__cydff_1_reg:macrocell.q
</CYPRESSTAG>
```

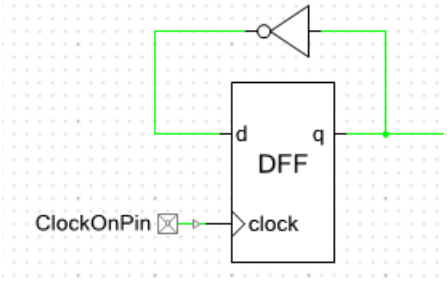
此电路生成的结果信号如下。



Div4 信号似乎是由 Div2 信号的下降沿生成。事实并非如此。Div4 信号在 Div2 上的上升沿后的第一个 Clock_1 上升沿处生成。

以引脚时钟为例

在下面的电路中，时钟在打开了同步的引脚上引入。因为引脚同步通过系统时钟完成，已转换的电路将把系统时钟用作有效时钟，并将引脚的上升沿用作使能信号。



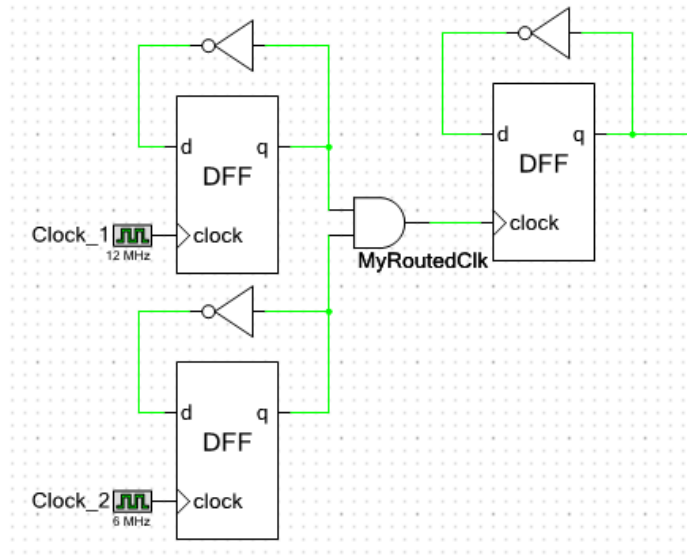

```
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
{Global Clock Selection}
<CYPRESSTAG name="UDB Routed Clock Assignment">
    Routed Clock: ClockOnPin(0):iocell.fb
        Effective Clock: BUS_CLK
        Enable Signal: ClockOnPin(0):iocell.fb
</CYPRESSTAG>
```

如果没有在引脚处启用输入同步，则将没有全局时钟可用于转换路由时钟，系统将直接使用路由时钟。

```
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
<CYPRESSTAG name="Global Clock Selection" icon="FILE_RPT_TECHM">
</CYPRESSTAG>
<CYPRESSTAG name="UDB Routed Clock Assignment">
    Routed Clock: ClockOnPin(0):iocell.fb
        Effective Clock: ClockOnPin(0):iocell.fb
        Enable Signal: True
</CYPRESSTAG>
```

以多个时钟源为例

在此示例中，路由时钟派生自通过两个不同时钟进行计时的触发器。由于这两个时钟均为同步时钟，因此系统时钟为即将成为有效时钟的通用全局时钟。

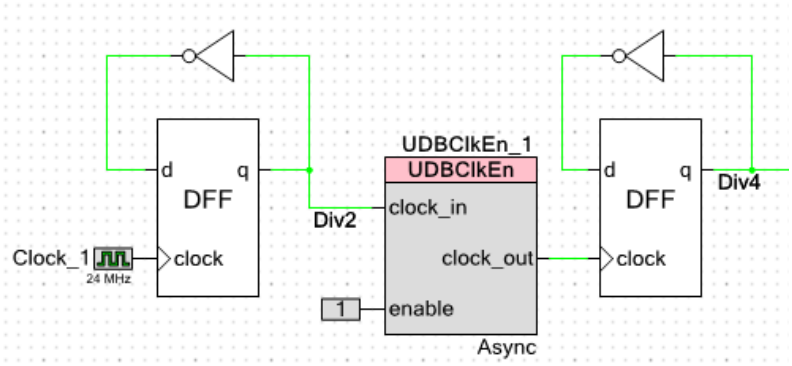


```
<CYPRESSTAG name="Tech mapping">
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
<CYPRESSTAG name="Global Clock Selection" icon="FILE_RPT_TECHM">
    Digital Clock 0: Automatic-assigning  clock 'Clock_1'. Fanout=1, Signal=tmp__cydff_1_clk
    Digital Clock 1: Automatic-assigning  clock 'Clock_2'. Fanout=1, Signal=tmp__cydff_2_clk
</CYPRESSTAG>
<CYPRESSTAG name="UDB Routed Clock Assignment">
    Routed Clock: MyRoutedClk:macrocell.q
        Effective Clock: BUS_CLK
        Enable Signal: MyRoutedClk:macrocell.q
</CYPRESSTAG>
<CYPRESSTAG name="UDB Clock/Enable Remapping Results">
</CYPRESSTAG>
```

如果这两个时钟中有任何一个是异步时钟，则系统将直接使用路由时钟。

覆盖路由时钟转换

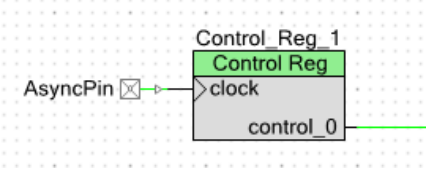
PSoC Creator 在路由时钟上执行的自动转换通常是应该使用的转换。但是，也有可强制直接使用路由时钟的方法。在异步模式中配置的 UDBClkEn 组件会将使用的时钟强制变为路由时钟，如下方电路所示。

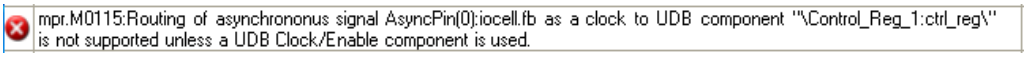


使用异步时钟

异步时钟可以与 PLD 逻辑一起使用。但是，因为与 CPU 存在交互，控制寄存器、状态寄存器和数据路径（Datapath）不会自动支持异步时钟。大多数赛普拉斯库组件仅支持同步时钟。具体来说，如果提供的时钟是异步时钟，这些组件会自动强制插入同步器。设计用于与异步时钟配合使用的组件（如 SPI 从组件）会在数据手册中具体描述它们处理时钟的方式。

如果异步时钟直接连接的不是 PLD 逻辑，则会生成设计规则检查 (DRC) 错误。例如，如果异步引脚与控制寄存器时钟连接，则会生成 DRC 错误。





如错误消息中所述，可通过在异步模式中使用 UDBClkEn 组件消除此错误。这并不会从根本上消除同步问题，但如果设计已通过其他某种方式处理了同步，这可以允许设计覆盖此错误。

跨时钟域

一个设计中通常需要多个时钟域。通常这多个域之间不会交互，因而不会发生跨时钟域现象。如果一个时钟域中生成的信号需要用于另一个时钟域，在这种情况下必须特别小心。存在两种情况：两个时钟域互不同步；两个时钟域均与系统时钟同步。

当两个时钟域的时钟均与系统时钟同步时，来自较慢时钟域的信号可在另一时钟域中任意使用。而在相反情况下，必须特别小心：来自较快时钟域的信号活动周期足够长，将被较慢时钟域采样。在这两种情况下，须满足的时序限制将基于系统时钟的速度，而非其中任一时钟域的速度。

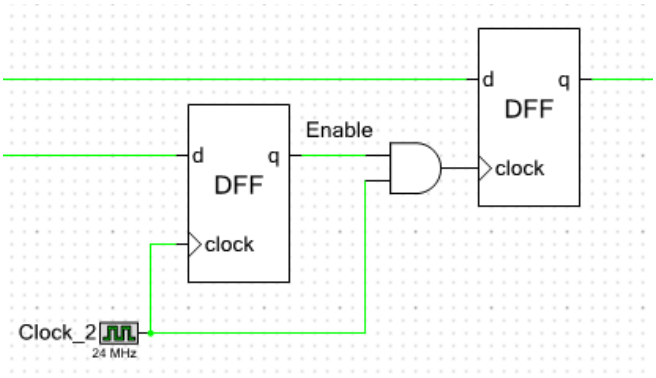
在这两个时钟域之间，唯一可确定的是它们的沿将始终出现在系统时钟的上升沿。因此，这两个时钟域的上升沿的距离与单个系统时钟循环之间的距离一样近。即使某一时钟域是对方的的情况下仍然如此，因为它们的时钟分频器不必对齐。如果两个时钟域之间存在组合逻辑，则可能需要插入触发器，以防系统时钟操作的频率受限。通过插入触发器，一个时钟域到另一个时钟域的跨越是一个直接的触发器到触发器路径。

当时钟域相互不关联时，必须在时钟域之间使用同步器。同步器件可用于实施同步功能。它应由目标时钟域进行计时。

同步组件实施使用的是可实施双同步器的状态寄存器特殊模式。输入信号的脉宽必须至少达到采样时钟的周期。通过同步器的确切延迟会有不同，具体取决于输入信号与同步时钟的对齐状况。延迟的范围在仅超过一个时钟周期到只超过两个时钟周期之间。如果正在同步多个信号，两个进入同步器的信号和输出处相同对应的两个信号之间的时间差可能会变化一个时钟周期，具体取决于同步器成功采样每个信号的时间。

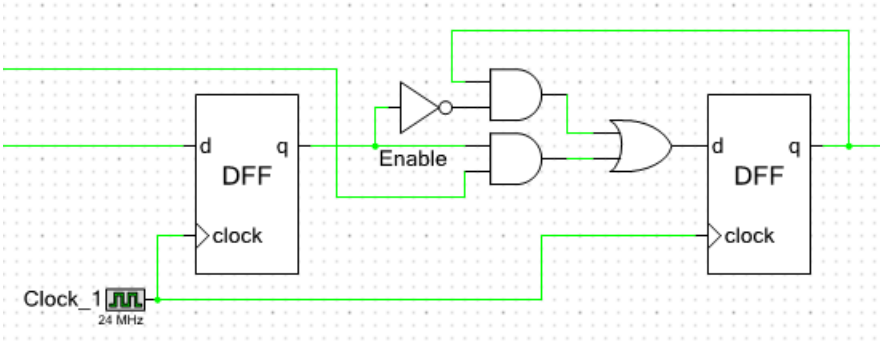
门控时钟

全局时钟不得用于直接对电路计时以外的其他目的。如果全局时钟用于逻辑功能，信号会在无保证时序的状态下使用完全不同的路径进行路由。因为无法执行时序分析，应避免如下电路。



此电路使用路由时钟实施，没有时序分析支持，在时钟启用和禁用时易受时钟信号上产生的短时脉冲影响。

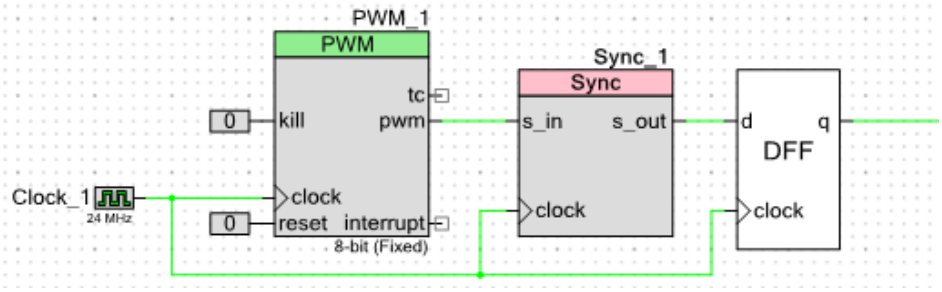
以下电路实施相同的功能，由时序分析支持，仅使用全局时钟，且没有可靠性问题。此电路不对时钟进行门控，而是逻辑启用新数据计时或逻辑维持当前数据。



如果需要访问时钟，例如为了生成时钟以发送到引脚，则应使用两倍频率时钟对反转触发器进行计时。然后，该触发器的输出可以与可用的关联时序分析配合使用。

固定功能时钟

在原理图中，发送到固定功能外设和基于 UDB 的外设上的时钟信号似乎是相同的时钟。但是，并不能保证时钟信号到达这些不同外设类型时的时序关系。此外，也不能保证数据信号的路由延迟。因此，当固定功能外设连接到 UDB 阵列中的信号时，这些信号必须按如下所示方式同步。不应对来自固定功能外设的信号做出时序假设。



缓存配置

如果 CPU 时钟频率在器件工作期间升高，应对时钟周期数进行调整，缓存在对从闪存返回的数据取样前将保持等待。如果 CPU 时钟频率降低，也可对时钟周期数进行调整以提高 CPU 性能。有关 PSoC 3/PSoC 5/PSoC 5LP 的更多信息，请参见“CyFlash_SetWaitCycles()”。有关 PSoC 4 的更多信息，请参见“CySysFlashSetWaitCycles()”。

低电压模拟升压时钟 (PSoC 3/PSoC 5LP)

当 PSoC 3 或 PSoC 5LP 器件的工作电压 (Vdda) 低于 2.7 V 时，SC-block 的模拟升压泵需要升压时钟输入以确保其性能符合标准。当该工作电压处于 2.7 V 和 4.0 V 之间时，可选择升压时钟以提高性能。当该工作电压高于 4.0 V 时，切勿使用升压时钟。

在先前发布的 PSoC Creator 中，默认为每个需要升压的组件实例（TIA、Mixer、PGA 和 PGA_Inv）保留模拟时钟资源。因此，当 Vdda 较低时，模拟设计会过早耗尽可用资源。通过使用该功能，可自动共享设计范围模拟时钟资源，并可在运行期间通过 CySetScPumps() 函数为所有 SC-block 提供控制模拟泵和升压时钟的功能。

在 SC-block 中实现的组件（TIA、Mixer、PGA 和 PGA_Inv）将根据 Vdda 和变量 Vdda 设计时间选项进行初始化。当工作电压 (Vdda) 降至 2.7 V 以下时，可通过 CySetScPumps() 函数在运行时启用或禁用升压泵和升压时钟。用户需监控 Vdda 电压。

当 PSoC 3 或 PSoC 5LP 器件的工作电压 (Vdda) 降至 4.0 V 以下时，须通过调用 SetAnalogRoutingPumps() 及相应参数以启用模拟路由开关的模拟泵。当 Vdda 升至 4.0 V 以上时，须禁用模拟路由开关的模拟泵。运行期间，用户需监控 Vdda 电压。模拟路由开关的模拟泵在器件启动时将根据 **Vdda** 和**变量 Vdda** 设计时间选项进行配置。

通过加载 PSoC Creator 设计范围资源 (Design-Wide Resources, DWR) 文件之系统（System）”选项卡中的“变量 Vdda”选项，可创建设计范围内模拟时钟资源 (ScBoostClk) 以作为模拟模块的升压时钟源。

下表描述了在 PSoC Creator 设计范围资源 (DWR) 文件之“系统”选项卡中配置的 **Vdda** 和**变量 Vdda** 值之间的依存关系。

Vdda	**< 2.7 V**	**≥ 2.7 V**	**≥ 2.7 V**
变量 Vdda	保持启用	Enabled（启用）	禁用（禁用）
创建 ScBoostClk 时钟	是	是	否
复位时启动 ScBoostClk	是	否	否

注意：先前版本的 SC 模块组件（TIA、Mixer、PGA 和 PGA_Inv）将继续使用专用的本地时钟，新的低电压模拟升压时钟 API 对这些时钟没有影响。

API

有一组用于 PSoC 3、PSoC 5 和 PSoC 5LP 器件的 API 和另一组用于 PSoC 4 器件的 API。以 CySysClk 开头的函数仅适用于 PSoC 4。所有其他函数仅适用于 PSoC 3、PSoC 5 和 PSoC 5LP。

PSoC 3/PSoC 5/PSoC 5LP API

uint8 CyPLL_OUT_Start(uint8 wait)

说明： 启用 PLL。可以选择等待其变稳定。等待至少 250 us 或直到检测到 PLL 稳定。

参数： 等待（wait）

- 0：配置后立即返回
- 1：等待 PLL 锁定或超时

返回值： 状态（status）

- CYRET_SUCCESS－成功完成
- CYRET_TIMEOUT - 发生超时，未检测到稳定时钟。如果时钟输入源抖动，则可能不会发生锁定指示。但是，超时过期后，生成的 PLL 时钟仍可使用。

副作用和限制： 如果启用等待，该函数使用“快速时轮” (FTW) 对等待进行计时。FTW 的其他任何使用都将在此函数周期停止，然后再恢复。

该函数使用 100 KHz ILO。如果未启用 100 KHz ILO，该函数将在函数执行期间启用 100 KHz ILO。

在该函数执行期间中断子程序不更改 ILO、FTW、中央时轮 (CTW) 或每秒一次中断的设置。如果电源管理器中断状态寄存器的读取仅通过使用 CyPMReadStatus() 函数完成，则 ILO、CTW 和每秒一次中断的当前操作将在此函数操作过程中得到保持。

void CyPLL_OUT_Stop()

说明： 禁用 PLL。

参数： 无

返回值： 无

void CyPLL_OUT_SetPQ(uint8 pDiv, uint8 qDiv, uint8 current)

说明： 设置 P 和 Q 分频器和电荷泵电流。输出频率等于 P/Q * 输入频率。调用此函数前必须先禁用 PLL。

参数： P：有效范围 [8 - 255]

Q：有效范围 [1 - 16]。输入频率 / Q 必须在 1MHz 到 3MHz 范围内。

current：有效范围 [1 - 7]。电荷泵电流按 uA 计算。有关更多信息，请参见器件 TRM 和数据手册。

返回值： 无

副作用和限制： 如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

void CyPLL_OUT_SetSource(uint8 source)

说明： 将输入时钟源设置为 PLL。调用此函数前必须先禁用 PLL。

参数： source：三个可用 PLL 时钟源中的一个

值	定义	源
0	CY_PLL_SOURCE_IMO	IMO
1	CY_PLL_SOURCE_XTAL	MHz 晶振
2	CY_PLL_SOURCE_DSI	DSI

返回值： 无

副作用和限制： 如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

void CyIMO_Start(uint8 wait)

说明： 启用 IMO。可以选择等待至少 6us，让其稳定。

参数： wait：

- 0：配置后立即返回
- 1：等待至少 6us，让 IMO 稳定。

返回值： 无

副作用和限制： 如果启用等待，该函数使用 FTW 对等待进行计时。FTW 的其他任何使用都将在此函数周期停止，然后再恢复。
该函数使用 100 KHz ILO。如果未启用 100 KHz ILO，该函数将在函数执行期间启用 100 KHz ILO。

在该函数执行期间中断子程序不更改 ILO、FTW、CTW 或每秒一次中断的设置。如果电源管理器中断状态寄存器的读取仅通过使用 CyPMReadStatus() 函数完成，则 ILO、CTW 和每秒一次中断的当前操作将在此函数操作过程中得到保持。

void CyIMO_Stop()

说明： 禁用 IMO。

参数： 无

返回值： 无

void CyIMO_SetFreq(uint8 freq)

说明： 设置 IMO 频率。在 IMO 运行过程中可以进行更改。

参数： freq：IMO 操作的频率

值	定义	频率
0	CY_IMO_FREQ_3MHZ	3 MHz
1	CY_IMO_FREQ_6MHZ	6 MHz
2	CY_IMO_FREQ_12MHZ	12 MHz
3	CY_IMO_FREQ_24MHZ	24 MHz
4	CY_IMO_FREQ_48MHZ	48 MHz
5	CY_IMO_FREQ_62MHZ	62 MHz（PSoC 5 组件不支持）
8	CY_IMO_FREQ_USB	24 MHz（已针对 USB 操作进行调整）

返回值： 无

副作用和限制： 如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

如果选择 USB 设置，则启用 USB 时钟锁定电路。否则将禁用此电路。必须先给 USB 模块上电，然后方可选择 USB 设置。

void CyIMO_SetSource(uint8 source)

说明： 设置 IMO 模块时钟输出的源。默认情况下，IMO 输出为 IMO 自身。MHz 晶振或 DSI 输入也可成为 IMO 输出的源。

参数： source：三个可用 IMO 输出源中的一个

值	定义	源
0	CY_IMO_SOURCE_IMO	IMO
1	CY_IMO_SOURCE_XTAL	MHz 晶振
2	CY_IMO_SOURCE_DSI	DSI

返回值： 无

副作用和限制： 如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

void CyIMO_EnableDoubler()

说明： 启用 IMO 倍频器。两倍频率时钟用于将 24 MHz 输入转换为 48 MHz 输出，供 USB 模块使用。

参数： 无

返回值： 无

void CyIMO_DisableDoubler()

说明： 禁用 IMO 倍频器。

参数： 无

返回值： 无

void CyBusClk_SetDivider(uint16 divider)

说明： 设置用于生成总线时钟的分频器值。

参数： divider：有效范围 [0-65535]。时钟将按此值 + 1 进行分频。例如，要按 2 进行分频，此参数应设置为 1。

返回值： 无

副作用和限制： 如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

void CyCpuClk_SetDivider(uint8 divider)

说明： 设置用于生成 CPU 时钟的分频器值。仅适用于 PSoC 3。

参数： divider：有效范围 [0-15]。时钟将按此值 + 1 进行分频。例如，要按 2 进行分频，此参数应设置为 1。

返回值： 无

副作用和限制： 如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

void CyMasterClk_SetSource(uint8 source)

说明： 设置主组件时钟源。

参数： source：四个可用主控时钟源中的一个

值	定义	源
0	CY_MASTER_SOURCE_IMO	IMO
1	CY_MASTER_SOURCE_PLL	PLL
2	CY_MASTER_SOURCE_XTAL	MHz 晶振
3	CY_MASTER_SOURCE_DSI	DSI

返回值： 无

副作用和限制： 在调用此函数前，当前源和新的源必须都处于运行且稳定的状态。

如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

void CyMasterClk_SetDivider(uint8 divider)

说明： 设置用于生成主控时钟的分频器值。

参数： divider：有效范围 [0-255]。时钟将按此值 + 1 进行分频。例如，要按 2 进行分频，此参数应设置为 1。

返回值： 无

副作用和限制： 将主控器或总线时钟分频器的值从 div-by-n 更改为 div-by-1 时，div-by-1 后的第一个时钟周期输出可比最终/期望的 div-by-1 周期最多短 4 ns。
如果在器件工作期间 CPU 时钟频率升高，则使用相应参数调用 CyFlash_SetWaitCycles() 以调整时钟周期数，缓存在对从闪存返回的数据采样前保持等待。如果 CPU 时钟频率降低，则可以调用 CyFlash_SetWaitCycles() 函数提高 CPU 性能。有关更多信息，请参见“CyFlash_SetWaitCycles()”。

void CyUsbClk_SetSource(uint8 source)

说明： 设置 USB 时钟的源。

参数： source：四个可用 USB 时钟源中的一个

值	定义	源
0	CY_USB_SOURCE_IMO2X	IMO 2x
1	CY_USB_SOURCE_IMO	IMO
2	CY_USB_SOURCE_PLL	PLL
3	CY_USB_SOURCE_DSI	DSI

返回值： 无

void CyILO_Start1K()

说明： 启用 ILO 1 KHz 振荡器。

注意 默认情况下，无论时钟编辑器中的选项如何，ILO 1 KHz 振荡器始终是启用的。因此，只有手动关闭此振荡器后，才需要此 API。

参数： 无

返回值： 无

void CyILO_Stop1K()

说明： 禁用 ILO 1 KHz 振荡器。

注意 如果预测要使用睡眠或休眠低功耗模式的 API，ILO 1 KHz 振荡器必须是启用的。更多信息，请参阅本文的“电源管理”部分。

参数： 无

返回值： 无

void CyILO_Start100K()

说明： 启用 ILO 100 KHz 振荡器。

参数： 无

返回值： 无

void CyILO_Stop100K()

说明： 禁用 ILO 100 KHz 振荡器。

参数： 无

返回值： 无

void CyILO_Enable33K()

说明： 启用 ILO 33 KHz 分频器。

注意：33 KHz 时钟生成自 100 KHz 振荡器，所以它也必须处于运行状态才能生成 33 KHz 输出。

参数： 无

返回值： 无

void CyILO_Disable33K()

说明： 禁用 ILO 33 KHz 分频器。

注意：33 KHz 时钟生成自 100 KHz 振荡器，但此 API 不会禁用 100 KHz 时钟。

参数： 无

返回值： 无

void CyILO_SetSource(uint8 source)

说明： 设置 ILO 模块时钟输出的源。

参数： source：三个可用 ILO 输出源中的一个

值	定义	源
0	CY_ILO_SOURCE_100K	ILO 100 KHz
1	CY_ILO_SOURCE_33K	ILO 33 KHz
2	CY_ILO_SOURCE_1K	ILO 1 KHz

返回值： 无

uint8 CyILO_SetPowerMode(uint8 mode)

说明： 设置断电期间 ILO 使用的功耗模式。允许较低的断电功耗，可实现较短的启动时间。

参数： mode：

值	定义	说明
0	CY_ILO_FAST_START	启动较快，断电时内部偏置仍然打开。
1	CY_ILO_SLOW_START	启动较慢，断电时内部偏置关闭。

返回值： 先前功耗模式

uint8 CyXTAL_Start(uint8 wait)

说明： 启用 MHz 晶振。
等待，直到 XERR 位较低（无错误）的状态持续一毫秒或直到达到等待参数指定的毫秒数。

PSoC 5（不是指 PSoC 5LP）：
等待 CY_CLK_XMHZ_MIN_TIMEOUT 毫秒（或如果超过 CY_CLK_XMHZ_MIN_TIMEOUT，则等待参数指定的毫秒数）。不对 XERR 位状态进行检查。

参数： wait：有效范围 [0-255]。这是以毫秒为单位的超时值。由晶振指定合适的值。

返回值： 状态
CYRET_SUCCESS－成功完成
CYRET_TIMEOUT - 出现超时，在 XERR 上未检测到低值。

副作用和限制： 如果启用等待（非零等待），该函数使用 FTW 对等待进行计时。FTW 的其他任何使用都将在此函数周期停止，然后再恢复。
该函数也使用 100 KHz ILO。如果未启用 100 KHz，该函数将在该函数执行期间启用 100 KHz。

在该函数执行期间中断子程序不更改 ILO、FTW、CTW 或每秒一次中断的设置。如果电源管理器中断状态寄存器的读取仅通过使用 CyPmReadStatus() 函数完成，则 ILO、CTW 和每秒一次中断的当前操作将在此函数操作过程中得到保持。

void CyXTAL_Stop()

说明： 禁用兆赫兹晶体振荡器。

参数： 无

返回值： 无

void CyXTAL_EnableErrStatus()

说明： 启用兆赫兹晶振的 XERR 状态位生成。该函数对 PSoC 5 器件不可用（对 PSoC 3 和 PSoC 5LP 器件可用）。

参数： 无

返回值： 无

void CyXTAL_DisableErrStatus()

说明： 禁用兆赫兹晶振的 XERR 状态位生成。该函数对 PSoC 5 器件不可用（对 PSoC 3 和 PSoC 5LP 器件可用）。

参数： 无

返回值： 无

uint8 CyXTAL_ReadStatus()

说明： 读取兆赫兹晶振的 XERR 状态位。该状态位是粘连的、在读取后清除的值。该函数对 PSoC 5 器件不可用（对 PSoC 3 和 PSoC 5LP 器件可用）。

参数： 无

返回值： 状态：0：无错误，1：错误

void CyXTAL_EnableFaultRecovery()

说明： 启用故障恢复电路，其在兆赫兹晶振电路发生故障的情况下可切换到 IMO。在调用该函数以防止即时故障切换之前，晶振必须启用且在 XERR 位为 0 的情况下运行。该函数对 PSoC 5 器件不可用（对 PSoC 3 和 PSoC 5LP 器件可用）。

参数： 无

返回值： 无

void CyXTAL_DisableFaultRecovery()

说明： 禁用将在兆赫兹晶振电路发生故障的情况下切换到 IMO 的故障恢复电路。该函数对 PSoC 5 器件不可用（对 PSoC 3 和 PSoC 5LP 器件可用）。

参数： 无

返回值： 无

void CyXTAL_SetStartup(uint8 setting)

说明： 设置晶振的启动设置。

参数： setting：有效范围 [0-31]。值取决于所用晶振的频率和质量。有关更多信息，请参见器件 TRM 和数据手册。

返回值： 无

void CyXTAL_SetFbVoltage(uint8 setting)

说明： 设置晶振电路要使用的反馈参考电压。该函数对 PSoC 5 器件不可用（对 PSoC 3 和 PSoC 5LP 器件可用）。

参数： setting：有效范围 [0-15]。有关更多信息，请参见器件 TRM 和数据手册。

返回值： 无

副作用和限制： 反馈参考电压必须高于看门狗参考电压。

void CyXTAL_SetWdVoltage(uint8 setting)

说明： 设置看门狗用于检测晶振电路故障的参考电压。该函数对 PSoC 5 器件不可用（对 PSoC 3 和 PSoC 5LP 器件可用）。

参数： setting：有效范围 [0-7]。有关更多信息，请参见器件 TRM 和数据手册。

返回值： 无

副作用和限制： 反馈参考电压必须高于看门狗参考电压。

void CyXTAL_32KHZ_Start()

说明： 启用 32 KHz 晶体振荡器。

参数： 无

返回值： 无

void CyXTAL_32KHZ_Stop()

说明： 禁用 32 KHz 晶体振荡器。

参数： 无

返回值： 无

uint8 CyXTAL_32KHZ_ReadStatus()

说明： 读取 32 KHz 振荡器的两个状态位。

参数： 无

返回值： 状态

值	定义	源
0x20	CY_XTAL32K_ANA_STAT	模拟测量 1：稳定 0：不稳定

uint8 CyXTAL_32KHZ_SetPowerMode(uint8 mode)

说明： 设置在睡眠模式下所用 32 KHz 振荡器的功耗模式。存在较少噪声源时，在睡眠模式下可实现较低的功耗。在活动模式下，振荡器始终以高功耗模式运行。

参数： mode：

- 0：高功耗模式
- 1：睡眠模式下的低功耗模式

返回值： 先前功耗模式

void CySetScPumps(uint8 enable)

说明： 启动/停止模拟升压时钟并配置 SC 模块正向泵。

参数： enable：

- 1：启动模拟升压时钟并启用正向泵。
- 0：如果所有 SC 模块被禁用，则禁用 SC 模块正向泵并停止模拟升压时钟。

返回值： 无

void SetAnalogRoutingPumps(uint8 enabled)

说明： 启用或禁用模拟泵馈送模拟路由开关。目的是基于 Vdda 系统配置在启动时调用；当用户通知我们 Vdda 电压超过泵阈值时，可在运行期间调用。

参数： enabled：

- 1：启用泵。
- 0：禁用泵。

返回值： 无

PSoC 4 API

void CySysClkImoStart(void)

说明： 启用 IMO。

参数： 无

返回值： 无

副作用和限制： 无

void CySysClkImoStop(void)

说明： 禁用 IMO。

参数： 无

返回值： 无

副作用和限制： 如果 WDT 锁定（调用 CySysWdtLock()），则该函数无效。调用 CySysWdtUnlock() 将 WDT 解锁，并能够停止 ILO。注意：需要 ILO 才能运行 WDT。

void CySysClkIloStart(void)

说明： 禁用 ILO。

参数： 无

返回值： 无

副作用和限制： 无

void CySysClkIloStop(void)

说明： 禁用 ILO。

参数： 无

返回值： 无

副作用和限制： 无

void CySysClkWriteHfclkDirect (uint32 clkSelect)

说明： 为 HFCLK 选择直接源。

参数： clkSelect：HFCLK 直接源之一。

值	定义	源
0	CY_SYS_CLK_HFCLK_IMO	IMO
1	CY_SYS_CLK_HFCLK_EXTCLK	外部时钟引脚

返回值： 无

副作用和限制： 无

void CySysClkWriteSysclkDiv (uint32 divider)

说明： 从 HFCLK 为 SYSCLK 选择预分频器分频数量。

参数： divider：2 的幂次预分频器选择。

值	定义	源
0	CY_SYS_CLK_SYSCLK_DIV1	1
1	CY_SYS_CLK_SYSCLK_DIV2	2
2	CY_SYS_CLK_SYSCLK_DIV4	4
3	CY_SYS_CLK_SYSCLK_DIV8	8
4	CY_SYS_CLK_SYSCLK_DIV16	16
5	CY_SYS_CLK_SYSCLK_DIV32	32
6	CY_SYS_CLK_SYSCLK_DIV64	64
7	CY_SYS_CLK_SYSCLK_DIV128	128

返回值： 无

副作用和限制： 无

void CySysClkWriteImoFreq (uint32 freq)

说明： 设置 IMO 频率。

参数： freq：有效范围 [3-48]。IMO 运行频率。

返回值： 无

副作用和限制： 无

4 电源管理

提供了受 PSoC 器件支持的一整套的用于控制功耗和可用资源量的运行模块。

PSoC 3/PSoC 5/PSoC 5LP 器件支持以下功耗模式（采用由高到低的功耗顺序）：活动、备用活动、睡眠、休眠。

PSoC 4 器件支持以下功耗模式（采用由高到低的功耗顺序）：活动、睡眠、深度睡眠、休眠、停止。活动、睡眠及深度睡眠是标准的 ARM 定义功耗模式，支持 ARM CPU。休眠/停止模式甚至功耗更低，从深度睡眠到休眠/停止是由固件完成的。。但是，一旦唤醒后，CPU（及所有外设）将完全进行重置。

对于基于 ARM 的器件 (PSoC 4/PSoC 5/PSoC 5LP)，要唤醒 CPU，需要中断。电源管理实现假设用单独的组件（基于组件的唤醒时间配置）配置唤醒时间，以针对终端计数启动中断。有关详细信息，请参见“唤醒时间配置”一节。

应清除所有待处理中断（即使其处于屏蔽状态）之后方可将器件设为低功耗模式。

PSoC 3/PSoC 5/PSoC 5LP 实现

低功耗用法

当调试器在运行时，PSoC 5 不会进入低功耗模式。

对于 PSoC 3/PSoC 5/PSoC 5LP，当系统性能控制器 (SPC) 在执行命令时，电源管理器不会将器件设为低功耗模式。器件将在 SPC 执行完命令后进入低功耗模式。

时钟配置

要正确地进入低功耗模式并进行唤醒，需要满足几个器件配置要求。

- 在进入睡眠和休眠模式之前，应准备好时钟系统，以确保按预期在活动模式与低功耗模式之间进行切换。
- CyPmSaveClocks() 和 CyPmRestoreClocks() 函数分别用于负责在进入低功耗模式之前和唤醒进入活动模式之后准备好时钟配置。通常，CyPmSaveClocks() 用于保存配置并设置进入低功耗模式的要求。CyPmRestoreClocks() 用于将时钟配置恢复到其初始状态。
- 需要 IMO 作为主控时钟的源。因此，IMO 时钟值是根据设计范围资源系统（Design-Wide Resources）编辑器上的“Enable Fast IMO During Startup”（在启动过程中启用快速 IMO）选项设置的。如果启用了此选项，IMO 时钟频率设为 48 MHz；否则设为 12 MHz。

  注意：在 PSoC 5 器件上，IMO 时钟频率始终设为 12 MHz。当主控时钟源自 IMO 时，关闭 PLL 和 MHz ECO。

IMO 值必须为 12 MHz，之后方可进入睡眠和休眠模式。在输入指定的低功耗模式之前，IMO 频率设置为 12 MHz（无需更正闪存的等待周期数）。一旦唤醒，将立即恢复 IMO 频率。

- 将总线和主控时钟分频器设为一分频值，并设置闪存等待周期的新值以匹配 CPU 频率的新值。有关更多信息，请参见 CyFlash_SetWaitCycles() 函数的说明。

必须为所有器件启用 1 KHz ILO（默认情况下，无论时钟编辑器中的选项如何，1 KHz ILO 始终是启用的），以在睡眠和休眠低功耗模式下进行正确操作。用于重置后测量休眠/睡眠电压调节器的设置时间。在这段时间内，系统忽略进入这些模式的请求。使用 1 kHz ILO 的上升沿测量保持关闭延迟。终端计数由睡眠电压调节器调整寄存器 (PWRSYS_SLP_TR) 设置。 **警告**：请勿修改此寄存器。有关详细信息，请参见相应的器件寄存器 TRM。

调用 CyPmSaveClocks() 函数将修改器件时钟配置。因此，在调用 CyPmRestoreClocks() 函数之前，任何依赖于时钟的组件不应被启用，因为调用 CyPmRestoreClocks() 函数是用来恢复时钟的初始配置的。有关组件时钟要求的更多信息，请参见相应的组件数据手册。

唤醒时间配置

有三个定时器可用于将器件从低功耗模式中唤醒：CTW、FTW 和每秒一次脉冲 (One PPS)。有关这些定时器的更多信息，请参见器件 TRM 和数据手册。

有两种配置唤醒时间的方法：

- 通过使用所需的参数调用 CyPmSleep() 和 CyPmAltAct() 函数，以完成基于参数的唤醒时间配置。此配置方法仅适用于 PSoC 3 器件。
- 基于组件的唤醒时间配置。用睡眠定时器组件配置 CTW 唤醒时间间隔。用 RTC 组件配置一秒时间间隔。

没有可用于休眠模式的唤醒时间配置。

务必记住，仅保证第一个 CTW 和 FTW 时间间隔将小于指定时间间隔。要使后续时间间隔具有额定值，由 CyPmSleep() 和 CyPmAltAct() 函数启用对应的定时器，并使定时器保持启用状态。注意一些 API 也可使用此定时器。这会导致在进入低功耗模式之前，定时器时钟始终处于启用状态（仅在禁用了对应定时器时才可更改定时器时间间隔），因此唤醒时间间隔始终小于预期时间间隔。

必须在唤醒之后就用对应的参数调用 CyPmReadStatus() 函数（例如如果器件配置为基于 CTW 唤醒，则使用 CY_PM_CTW_INT 参数），以清除中断状态位。

当 CTW 作为唤醒定时器时，在唤醒后必须始终调用 CyPmReadStatus() 函数（当以基于参数或组件的方法配置唤醒时），以清除 CTW 中断状态位。要求在 CTW 事件发生后的 1 ms（ILO 的 1 个时钟周期）内调用此函数。

唤醒源配置

可配置用哪个唤醒源将器件从“备用活动”和“睡眠”低功耗模式中唤醒。未将该源配置为唤醒器件；该源允许唤醒组件。必须正确配置与唤醒源相关的组件以用作唤醒源。

对于 PSoC 5 和 PSoC 5LP，还必须启用与唤醒源相关的中断以便也唤醒 CPU。

PSoC 3 备用活动模式特定问题

- 任何中断，无论是否已在中断控制器上启用，都将把器件从备用活动功耗模式中唤醒。

- 同时还将绕过边沿检测器，因此唤醒源始终是通过电平触发的。
- 直接连接的 DMA 中断将不会从此模式中唤醒。必须通过 DSI 将它们路由过来，以生成唤醒条件。

PSoC 5 特定问题

对于 PSoC 5，唤醒源针对所有低功耗模式都不可选。在这种情况下，唤醒源参数将被忽略，且任何可用的唤醒源都可唤醒该器件。

对于 PSoC 5，仅支持 CTW 作为睡眠模式的唤醒源。PSoC 5LP 支持所有唤醒源。

PSoC 5LP 特定问题

对于 PSoC 5LP，睡眠模式可使用唤醒源，而备用活动模式不可使用唤醒源。在备用活动模式下，唤醒源参数将被忽略，且任何可用的唤醒源都可唤醒该器件。

对于 PSoC 5LP，连接到唤醒源的中断组件不能使用“RISING_EDGE”检查选项。而应使用“LEVEL”（电平）选项代替。

电源管理 API

void CyPmSaveClocks()

说明： 调用该函数准备进入睡眠或休眠低功耗模式。保存睡眠/休眠模式中不保留的或为进入睡眠/休眠模式必须更改的时钟系统的所有状态。关闭所有仅用于激活功耗模式的数字和模拟时钟分频器。
将主时钟切换至 IMO 并关闭 PLL 和 MHz 晶振。IMO 频率设为 12 MHz 或 48 MHz，以匹配设计范围资源（Design-Wide Resources ）系统编辑器的“Enable Fast IMO During Startup”（在启动过程中启用快速 IMO）设置。ILO 和 32 KHz 振荡器不受影响。当前闪存等待状态设置已保存，且该闪存等待状态设置已针对当前 IMO 的速度进行了设置。
注意：如果主控时钟源可通过 DSI 输入路由，则在使用 CyPmSaveClocks()/CyPmRestoreClocks() 函数之前必须手动将它设置为另一个源。

参数： 无

返回值： 无

副作用和限制： 完成此 API 方法调用后，所有外设时钟将关闭。

void CyPmRestoreClocks()

说明： 恢复最后调用 CyPmSaveClocks 保留的任何状态。闪存等待状态设置也将恢复。

注意：如果主控时钟源可通过 DSI 输入路由，则在使用 CyPmSaveClocks()/CyPmRestoreClocks() 函数之前必须手动将它设置为另一个源。

如果保持关闭超时后兆赫兹晶振未就绪，合并区域可用于处理状态。

PSoC 5： 提供最高 130 ms 以便兆赫兹晶振稳定下来。保持关闭超时后不验证其是否就绪。不适用于 PSoC 5LP器件。

参数： 无

返回值： 无

void CyPmAltAct(uint16 wakeupTime, uint16 wakeupSource)

说明： 将器件置于“备用活动”（待机）状态。“备用活动”状态可允许器件的任何功能处于活动状态，但是该函数的操作具体决定于“备用活动”状态期间所禁用的 CPU。配置代码和组件 API 将为“备用活动”状态配置模板，使其与“活动”状态相同，有所区别的是，CPU 将在“备用活动”期间被禁用。

注意：调用此函数之前，必须针对用作唤醒定时器的定时器手动配置源时钟的功耗模式。

PSoC 3： 在切换至“备用活动”之前，如果 wakeupTime 没有指定为 NONE，则将根据中断指定为禁用定时器配置合适的定时器状态。唤醒源是 wakeupSource 中指定的值和 wakeupTime 参数中指定的任何定时器的组合。一旦满足唤醒条件，所有保存的状态都将恢复，该函数也将恢复为“活动”状态。

注意：如果 wakeupTime 值有所不同，则发生唤醒之前的时间将显著少于指定时间。如果使用相同的 wakeupTime 值调用下一函数，则将在前次唤醒发生后的指定时间发生唤醒。

如果 wakeupTime 没有指定为 NONE，则在退出时，指定定时器将保持 wakeupTime 指定的状态，此时定时器为启用状态且中断为禁用状态。如果已经为唤醒配置了 CTW、FTW 或 One PPS（例如，使用 Sleep Timer 或 RTC 组件），则将 wakeupTime 指定为 NONE 并为 wakeupSource 提供合适的源。

PSoC 5 和 PSoC 5LP：两个参数都不适用。这意味着参数应设为 NONE。组参数应设为将进入“备用活动”模式，直到启用的中断触发。

参数： wakeupTime：指定定时器唤醒源和该源的频率。对于 PSoC 5 和 PSoC 5LP，忽略此参数。

值	定义	时间
0	PM_ALT_ACT_TIME_NONE	无
1	PM_ALT_ACT_TIME_ONE_PPS	One PPS：1 秒
2	PM_ALT_ACT_TIME_CTW_2MS	CTW：2 ms
3	PM_ALT_ACT_TIME_CTW_4MS	CTW：4 ms
4	PM_ALT_ACT_TIME_CTW_8MS	CTW：8 ms
5	PM_ALT_ACT_TIME_CTW_16MS	CTW：16 ms
6	PM_ALT_ACT_TIME_CTW_32MS	CTW：32 ms
7	PM_ALT_ACT_TIME_CTW_64MS	CTW：64 ms
8	PM_ALT_ACT_TIME_CTW_128MS	CTW：128 ms
9	PM_ALT_ACT_TIME_CTW_256MS	CTW：256 ms
10	PM_ALT_ACT_TIME_CTW_512MS	CTW：512 ms
11	PM_ALT_ACT_TIME_CTW_1024MS	CTW：1024 ms
12	PM_ALT_ACT_TIME_CTW_2048MS	CTW：2048 ms
13	PM_ALT_ACT_TIME_CTW_4096MS	CTW：4096 ms
14 - 269	PM_ALT_ACT_TIME_FTW(1-256)	FTW：10 µs 至 2.56 ms

PM_ALT_ACT_TIME_FTW() 宏使用带有指定要延迟的 10 µs 增量数的参数。对于 PSoC 3 芯片，值的有效范围为 1 至 256。

CyPmAltAct（续）

参数： wakeupSource：指定唤醒源的位掩码。此外，如果指定了 wakeupTime，则相关定时器将作为唤醒源被包括在内。函数退出之前，将恢复唤醒源配置。对于 PSoC 5 和 PSoC 5LP，忽略此参数。

值	定义	源
0	PM_ALT_ACT_SRC_NONE	无
1	PM_ALT_ACT_SRC_COMPARATOR0	比较器 0
2	PM_ALT_ACT_SRC_COMPARATOR1	比较器 1
4	PM_ALT_ACT_SRC_COMPARATOR2	比较器 2
8	PM_ALT_ACT_SRC_COMPARATOR3	比较器 3
16	PM_ALT_ACT_SRC_INTERRUPT	中断
64	PM_ALT_ACT_SRC_PICU	PICU
128	PM_ALT_ACT_SRC_I2C	I2C
512	PM_ALT_ACT_SRC_BOOSTCONVERTER	升压转换器
1024*	PM_ALT_ACT_SRC_FTW	快速时轮
1024*	PM_ALT_ACT_SRC_VD	高低电压检测
2048*	PM_ALT_ACT_SRC_CTW	中央时轮
2048*	PM_ALT_ACT_SRC_ONE_PPS	One PPS
4096	PM_ALT_ACT_SRC_LCD	LCD

注意：CTW 和 One PPS 唤醒信号位于同一掩码位。FTW 和低电压中断 (LVI)/高电压中断 (HVI) 唤醒信号位于同一掩码位。

如果指定一个比较器作为 wakeupSource，则使用特定于实例的定义，该定义将追踪该实例的特定比较器。例如，对于名为“MyComp”的比较器实例，使用“或”运算写入掩码的值是：MyComp_ctComp__CMP_MASK。

当 CTW、FTW 或 One PPS 用作唤醒源时，必须在唤醒时使用对应的参数调用 CyPmReadStatus 函数。有关更多信息，请参见 CyPmReadStatus API。

返回值： 无

副作用和限制： 对于 PSoC 5 和 PSoC 5LP，唤醒源不可选。在这种情况下，wakeupSource 参数将被忽略，且任何可用的唤醒源都可唤醒该器件。

如果 wakeupTime 没有指定为 NONE，则在退出时，指定定时器将保持 wakeupTime 指定的状态，此时定时器为启用状态且中断为禁用状态。同时，ILO 1 KHz（如果 CTW 定时器用作唤醒定时器）或 ILO 100 KHz（如果 FTW 定时器用作唤醒定时器）将保持启动状态。

void CyPmSleep(uint8 wakeupTime, uint16 wakeupSource)

说明： 将器件置于“睡眠”状态。

注意： 调用此函数之前，必须为用作唤醒定时器的定时器手动配置源时钟的功耗模式。

注意： 调用此函数前，必须通过调用 CyPmSaveClocks() 函数为低功耗模式准备时钟树。然后，通过调用 CyPmRestoreClocks() 函数在 CyPmSleep() 执行后恢复时钟配置。有关详细信息，请参见《系统参考指南》中“电源管理”一节的“时钟配置”小节。

PSoC 3： 切换至“睡眠”状态之前，如果未将 wakeupTime 指定为 NONE，则会根据中断指定为禁用定时器配置合适的定时器状态。唤醒源是 wakeupSource 中指定的值和 wakeupTime 参数中指定的任何定时器的组合。一旦满足唤醒条件，所有保存的状态都将恢复，该函数也将恢复为“活动”状态。

注意： 如果 wakeupTime 值与其上一值不同，则唤醒开始前的时间将大大短于指定时间。如果使用同一 wakeupTime 值再次调用该函数，则会在前次唤醒发生后的指定时间内唤醒。

如果 wakeupTime 没有指定为 NONE，则在退出时，指定定时器将保持 wakeupTime 指定的状态，此时定时器为启用状态且中断为禁用状态。如果已经为唤醒配置 CTW 或 One PPS（例如，使用 Sleep Timer 或 RTC 组件），则将 wakeupTime 指定为 NONE 并为 wakeupSource 提供合适的源。

PSoC 5： 此函数的两个参数都不用于 PSoC 5。这意味着应参数应设为 NONE。该器件将进入睡眠模式，直至被基于 CTW 的中断唤醒。必须已配置 CTW 以生成中断。使用 Sleep Timer 组件配置 CTW。只可使用 CTW 将器件从睡眠模式中唤醒。此函数自动禁用其他中断源，然后在器件被 CTW 唤醒之后恢复这些中断源。

需要控制睡眠时间，以便器件进入睡眠状态后不会过早唤醒或保持睡眠的时间太长。支持的可靠睡眠时间范围为 1 ms 到 128 ms。4 ms、8 ms 、16 ms、32 ms、128 ms 或 256 ms 等 CTW 设置可满足此要求。要控制睡眠时间，在将器件转为睡眠模式之前自动复位 CTW。这导致睡眠时间为编程到 CTW 中的时间的一半，考虑到第一个 ILO 时钟沿的到达时间，会有 1 ms 的误差。例如，4 ms 的设置将导致睡眠时间为 1 ms 到 2 ms。

PSoC 5LP： 不使用 wakeupTime 参数，且仅可设置为 NONE。必须为该组件配置唤醒时间，为 Sleep Timer 配置 CTW 间隔，以及为 RTC 配置 1PPS 间隔。要生成中断，必须先配置该组件。

CyPmSleep（续）

参数： wakeupTime：指定定时器唤醒源和该源的频率。对于 PSoC 5 和 PSoC 5LP，请忽略此参数。

值	定义	时间
0	PM_SLEEP_TIME_NONE	无
1	PM_SLEEP_TIME_ONE_PPS	One PPS：1 秒
2	PM_SLEEP_TIME_CTW_2MS	CTW：2 ms
3	PM_SLEEP_TIME_CTW_4MS	CTW：4 ms
4	PM_SLEEP_TIME_CTW_8MS	CTW：8 ms
5	PM_SLEEP_TIME_CTW_16MS	CTW：16 ms
6	PM_SLEEP_TIME_CTW_32MS	CTW：32 ms
7	PM_SLEEP_TIME_CTW_64MS	CTW：64 ms
8	PM_SLEEP_TIME_CTW_128MS	CTW：128 ms
9	PM_SLEEP_TIME_CTW_256MS	CTW：256 ms
10	PM_SLEEP_TIME_CTW_512MS	CTW：512 ms
11	PM_SLEEP_TIME_CTW_1024MS	CTW：1024 ms
12	PM_SLEEP_TIME_CTW_2048MS	CTW：2048 ms
13	PM_SLEEP_TIME_CTW_4096MS	CTW：4096 ms

wakeupSource：指定唤醒源的位掩码。此外，如果指定了 wakeupTime，则相关定时器将作为唤醒源被包括在内。函数退出之前，将恢复唤醒源配置。对于 PSoC 5，此参数将被忽略。

值	定义	源
0	PM_SLEEP_SRC_NONE	无
1	PM_SLEEP_SRC_COMPARATOR0	比较器 0
2	PM_SLEEP_SRC_COMPARATOR1	比较器 1
4	PM_SLEEP_SRC_COMPARATOR2	比较器 2
8	PM_SLEEP_SRC_COMPARATOR3	比较器 3
64	PM_SLEEP_SRC_PICU	PICU
128	PM_SLEEP_SRC_I2C	I2C
512	PM_SLEEP_SRC_BOOSTCONVERTER	升压转换器
1024	PM_SLEEP_SRC_VD	高低电压检测
2048*	PM_SLEEP_SRC_CTW	中央时轮
2048*	PM_SLEEP_SRC_ONE_PPS	One PPS
4096	PM_SLEEP_SRC_LCD	LCD

注意： CTW 和 One PPS 唤醒信号位于同一掩码位。对于 PSoC 5，这些值位于不同的位（值 1024）。

如果指定一个比较器作为 wakeupSource，则使用特定于实例的定义，该定义将追踪该实例的特定比较器。例如，对于名为“MyComp”的比较器实例，使用“或”运算写入掩码的值是：MyComp_ctComp__CMP_MASK。

当 CTW 或 One PPS 用作唤醒源时，必须在唤醒时使用对应的参数调用 CyPmReadStatus 函数。有关更多信息，请参见 CyPmReadStatus API。

CyPmSleep（续）

返回值： 无

副作用和限制： 如果 wakeupTime 没有指定为 NONE，则在退出时，指定定时器将保持 wakeupTime 指定的状态，此时定时器为启用状态且中断为禁用状态。同时，ILO 1 KHz（如果 CTW 定时器用作唤醒定时器）将保持启动状态。

应针对 PSoC3 和 PSoC 5LP 启用 1 kHz ILO 时钟，以在复位后测量休眠/睡眠电压调节器建立时间。使用 1 kHz ILO 的上升沿测量保持关闭延迟。

对于 PSoC 3 芯片，进入睡眠功耗模式前应先禁用硬件蜂鸣器。启动期间，PSoC Creator 将禁用硬件蜂鸣器。在睡眠模式下，如果需要设计 LVI、HVI 或掉电检测（电源监控功能）功能，需使用 CTW 周期唤醒组件、执行软件蜂鸣并更新监控服务。如果无需设计 LVI、HVI 或掉电检测功能，则不需要使用 CTW。有关详细信息，请参见器件勘误表。

void CyPmHibernate()

说明： 将部件置于“休眠”状态。

切换到“休眠”状态之前，将保存并设置 PICU 唤醒源位的当前状态。这样，就将组件配置为从 PICU 中唤醒。确保您至少配置了一个引脚以生成 PICU 中断。对于引脚 Px.y，寄存器“PICU_INTTYPE_PICUx_INTTYPEy”控制 PICU 行为。在 TRM 中，此寄存器为“PICU[0..15]_INTTYPE[0..7]”。在引脚组件数据手册中，此寄存器也称为 IRQ 选项。一旦发生唤醒，将恢复 PICU 唤醒源位，PSoC 也将恢复为“活动”状态。

PSoC 5：支持唤醒休眠状态的唯一方法是器件的硬件复位。将器件转为休眠状态之前，此函数自动禁用 PICU 中断源。

参数： 无

返回值： 无

副作用和限制： 从休眠状态中唤醒之后，在重新进入休眠或睡眠状态之前，应用程序必须等待 20 µs。通过这 20 µs，可在下次休眠/睡眠事件发生前稳定睡眠电压调节器。当器件唤醒时，需要这 20 µs。没有硬件检查来确认这个要求是否满足。指定的延迟应在 ISR 入口上完成。

在唤醒 PICU 中断发生之后，必须调用 Pin_ClearInterrupt() 函数（其中“Pin”（引脚）是引脚组件的实例名称），以清除锁存的引脚事件。在这种情况下，可正确进入休眠模式，并启用对未来事件的检测。

应针对 PSoC3 和 PSoC 5LP 启用 1 kHz ILO 时钟，以在复位后测量休眠/睡眠电压调节器建立时间。使用 1 kHz ILO 的上升沿测量保持关闭延迟。

uint8 CyPmReadStatus(uint8 mask)

说明： 管理电源管理器中断状态寄存器。该寄存器具有每秒一次脉冲、中央时轮和快速时轮定时器的中断状态。该硬件寄存器将在读取后进行清理。为了仅清除目标位，而保存其他位，该函数使用了保留该状态的影像寄存器。该函数使用影像寄存器读取状态寄存器并对该值进行“或”运算。也就是返回的值。然后，将从该值清除掩码中的位并将其写回至影像寄存器。

注意：必须在发生 CTW 事件后的 1 ms（ILO 的一个时钟周期）内调用此函数。

参数： mask：影像寄存器中要清除的位

值	定义	源
1	CY_PM_FTW_INT	快速时轮
2	CY_PM_CTW_INT	中央时轮
4	CY_PM_ONEPPS_INT	每秒一次脉冲

返回值： 状态。与用于掩码参数的枚举位的值相同。

PSoC 4 实现

当电路板上的 Vccd 降低为 Vddd 时，该软件应在 PWR_CONTROL 寄存器中设置 EXT_VCCD 位。这会影响芯片内部状态转换。在这些转换期间，用户需了解 Vccd 电压是连接还是浮动状态，以便在低功耗模式下实现最低电流。

注意：除非 Vddd 和 Vccd 引脚均由外部供电，否则设置该位将关闭活动的电压调节器，并导致系统重置。有关详细信息，请参见器件 TRM。

从 ISR 调用 PM API 十分安全。“睡眠”和“深度睡眠”低功耗模式的唤醒条件见下表：

中断状态	条件	Wakeup（唤醒）	ISR 执行
取消屏蔽	IRQ 优先级 > 当前级	是	是
	IRQ 优先级 ≤ 当前级	否	否
屏蔽	IRQ 优先级 > 当前级	是	否
	IRQ 优先级 ≤ 当前级	否	否

电源管理 API

void CySysPmSleep(void)

说明： 将部件置于“睡眠”状态。该功耗模式以 CPU 为中心。它表示 CPU 已表明其处于“睡眠”模式并可移除其主时钟。就外设而言，它相当于“活动”模式。任何启用的中断均可导致从睡眠模式中被唤醒。

参数： 无

返回值： 无

副作用和限制： 无

void CySysPmDeepSleep(void)

说明： 将部件置于“深度睡眠”状态。

如果固件尝试在系统就绪前进入深度睡眠模式（即，当 PWR_CONTROL.LPM_READY = 0），器件将进入深度睡眠模式，并在触发抑制超时时自动进入最初预计的模式。从深度睡眠或休眠的外设收到中断后，开始唤醒。有关详细信息，请参见对应的外设数据手册。

参数： 无

返回值： 无

副作用和限制： 无

void CySysPmHibernate(void)

说明： 将部件置于“休眠”状态。仅保留 SRAM 和 UDB，且大多数内部组件供电断开。仅可通过引脚或休眠的比较器进行唤醒。

参数： 无

返回值： 无

副作用和限制： 调用此函数前固件应已使用 CySysPmFreezeIo() 函数冻结 IO 单元。如果没有冻结 IO 单元，IO 单元从”活动“状态转换为“深度睡眠”状态时将以同一方式被冻结，但 IO 单元在唤醒时将丢失（因为同时发生了重置）。

由于所有 CPU 状态均已丢失，CPU 将从矢量复位后启动。要在“休眠”低功耗模式下保存固件状态，应使用 CY_NOINIT 属性定义相应变量。如此可防止启动时将数据初始化为 0。将保留休眠状态下的外设的中断原因，例如，可能是由于固件读取，或固件启动或启用相应中断后导致中断。要区分唤醒与睡眠模式和一般复位事件，可使用 CySysPmGetResetReason() 函数。

void CySysPmStop(void)

说明： 将部件置于“停止”状态。所有内部供电断开且不保留任何状态。

通过切换唤醒引脚 (P0.7) 可从停止模式唤醒，从而执行正常的启动程序。要配置唤醒引脚，应将分配给 P0.7 的数字输入引脚组件置于原理图上，并通过电阻上拉或下拉至唤醒极性的相反状态。要区分唤醒与停止模式和一般复位事件，可使用 CySysPmGetResetReason() 函数。唤醒引脚默认为低电平有效。可通过 CySysPmSetWakeupPolarity() 函数更改此唤醒引脚极性。

参数： 无

返回值： 无

副作用和限制： 此函数将暗中冻结 IO 单元。冻结 IO 单元前无法进入“停止”模式。IO 单元从“停止”模式唤醒后将保持冻结状态，直到固件通过 CySysPmUnfreezeIo() 函数调用启动后将其解冻。

void CySysPmSetWakeupPolarity(uint32 polarity)

说明： 通过切换唤醒引脚 (P0.7) 可从停止模式唤醒，从而执行正常的启动程序。此函数用于为唤醒引脚分配有效电平。为该电平设置唤醒引脚后，可从停止模式唤醒。唤醒引脚默认为低电平有效。

参数： 极性：唤醒引脚有效电平

值	定义	说明
0	CY_PM_STOP_WAKEUP_ACTIVE_LOW	逻辑零将唤醒芯片。
1	CY_PM_STOP_WAKEUP_ACTIVE_HIGH	逻辑 1 将唤醒芯片。

返回值： 无

副作用和限制： 无

uint32 CySysPmGetResetReason(void)

说明： 获取最后复位原因－从断电/XRES/停止/休眠模式转换为“复位”状态。注意：使用 XRES 从停止模式唤醒将被视为一般复位。

参数： 无

返回值： 复位原因

值	定义	复位原因
0	CY_PM_RESET_REASON_UNKN	未知
1	CY_PM_RESET_REASON_XRES	从断电/XRES 模式转换为“复位”状态
2	CY_PM_RESET_REASON_WAKEUP_HIB	从“休眠”模式转换/唤醒为“复位”状态
3	CY_PM_RESET_REASON_WAKEUP_STOP	从“停止”模式转换/唤醒为“复位”状态。

副作用和限制： 无

void CySysPmFreezeIo(void)

说明： 对 IO 单元直接解冻，以便从“休眠”或“停止”模式唤醒时保存 IO 单元状态。

参数： 无

返回值： 无

副作用和限制： 由于 CySysPmStop() 函数暗中冻结了 IO 单元，因此进入“停止”模式前无需调用此函数。

void CySysPmFreezeIo(void)

说明： 从“休眠”或“停止”模式唤醒后，IO 单元将保持冻结，直到固件启动后对其进行解冻。调用此函数将暗中解冻 IO 单元。

参数： 无

返回值： 无

副作用和限制： 无

实例低功耗 API

大多数组件具有特定于实例的低功耗 API 集。使用这些 API，可将组件置于低功耗状态。下文大致列出了这些函数。有关寄存器保留信息的具体内容，请参考对应的数据手册（如果适用）。

void `=instance_name`_Sleep (void)

说明： _Sleep() 函数用于查看组件是否已启用，并保存该状态。然后，调用_Stop() 函数和 _SaveConfig() 函数保存用户配置。

- PSoC 3/PSoC 5/PSoC 5LP：在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 _Sleep() 函数。
- PSoC 4：在调用 CySysPmDeepSleep() 函数之前调用 _Sleep() 函数。

参数： 无

返回值： 无

副作用： 无

void `=instance_name`_Wakeup (void)

说明： _Wakeup() 函数调用 _RestoreConfig() 函数以恢复用户配置。如果组件在调用_Sleep() 函数前已启用，则_Wakeup() 函数将重新启用组件。

参数： 无

返回值： 无

副作用： 调用_Wakeup() 函数前未调用_Sleep() 或_SaveConfig() 函数可能会产生意外行为。

void `=instance_name`_SaveConfig(void)

说明： 此函数保存组件配置。它将保存非保留寄存器。此函数还将保存当前“配置”对话框中定义的或通过相应 API 修改的组件参数值。该函数由 _Sleep() 函数调用。

参数： 无

返回值： 无

副作用： 无

void `=instance_name`_RestoreConfig(void)

说明： 此函数恢复组件配置。此将恢复非保留寄存器。该函数还会将组件参数值恢复为调用 _Sleep() 函数之前的值。

参数： 无

返回值： 无

副作用： 调用该函数前未调用_Sleep() 或_SaveConfig() 函数可能会产生意外行为。

5 中断

除非另行说明，此章节中的 API 应用于所有架构的器件。有关中断的更多信息，另请参见中断组件数据手册。

注意：对于 PSoC 3，Keil C 编译器的运行时库将不禁用中断。但此特性对使用可重入的大函数的 C51 运行时库除外。中断会被禁止 4 个 CPU 指令周期（8 个 CPU 时钟周期）的时间

，以调整大的可重入堆栈。

API

CyGlobalIntEnable

说明： 使用全局中断屏蔽启用中断的宏语句。

CyGlobalIntDisable

说明： 使用全局中断屏蔽禁用中断的宏语句。

uint32 CyDisableInts()

说明： 禁用所有中断。

参数： 无

返回值： 之前已启用的中断的 32 位掩码

void CyEnableInts(uint32 mask)

说明： 启用 32 位掩码中指定的所有中断。

参数： mask：要启用中断的 32 位掩码

返回值： 无

注意：中断服务子程序必须遵守以下策略：将 CYDEV_INTC_CSR_EN 寄存器位和中断启用状态 (EA) 恢复为在入口时的状态。只要使用了合适的呼应 CyEnterCriticalSection() 与 CyExitCriticalSection() 函数调用，ISR 就不需要再做任何特殊操作。

void CyIntEnable(uint8 number)

说明： 启用指定的中断编号。

参数： number：中断编号。有效范围：[0-31]

返回值： 无

注意：中断服务子程序必须遵守以下策略：将 CYDEV_INTC_CSR_EN 寄存器位和中断启用状态 (EA) 恢复为在入口时的状态。只要使用了合适的呼应 CyEnterCriticalSection() 与 CyExitCriticalSection() 函数调用，ISR 就不需要再做任何特殊操作。

void CyIntDisable(uint8 number)

说明： 禁用指定的中断编号。

参数： number：中断编号。有效范围：[0-31]

返回值： 无

注意：中断服务子程序必须遵守以下策略：将 CYDEV_INTC_CSR_EN 寄存器位和中断启用状态 (EA) 恢复为在入口时的状态。只要使用了合适的呼应 CyEnterCriticalSection() 与 CyExitCriticalSection() 函数调用，ISR 就不需要再做任何特殊操作。

uint8 CyIntGetState(uint8 number)

说明： 获取指定中断编号的启用状态。

参数： number：中断编号。有效范围：[0-31]

返回值： 启用状态：如果已启用，则为 1；如果已禁用，则为 0

cyisraddress CyIntSetVector(uint8 number, cyisraddress address)

说明： 设置指定中断编号的中断矢量。

参数： number：中断编号。有效范围：[0-31]

address：指向中断服务子程序的指针

返回值： 之前的中断矢量值

cyisraddress CyIntGetVector(uint8 number)

说明： 获取指定中断编号的中断矢量。

参数： number：中断编号。有效范围：[0-31]

返回值： 中断矢量值

cyisraddress CyIntSetSysVector(uint8 number, cyisraddress address)

说明： 此函数仅适用于基于ARM的处理器，因此不适用于 PSoC 3 器件。它用于设置指定异常的中断矢量。ARM 架构中的这些异常按与用户中断类似的方式运行，但由处理器的系统架构指定。每个异常的编号是固定的。注意：这些异常的编号独立于用户中断所用的编号。

参数： number：异常编号。有效范围：[0-15]。

address：指向中断服务子程序的指针

返回值： 之前的中断矢量值

cyisraddress CyIntGetSysVector(uint8 number)

说明： 此函数仅适用于基于ARM的处理器，因此不适用于 PSoC 3 器件。它用于获取指定异常的中断矢量。ARM 架构中的这些异常按与用户中断类似的方式运行，但由处理器的系统架构指定。每个异常的编号是固定的。注意：这些异常的编号独立于用户中断所用的编号。

参数： number：异常编号。有效范围：[0-15]。

返回值： 中断矢量值

void CyIntSetPriority(uint8 number, uint8 priority)

说明： 设置指定中断编号的优先级。

参数： number：中断编号。有效范围：[0-31]

优先级：中断优先级。0 的优先级最高。有效范围：[0-7]

返回值： 无

uint8 CyIntGetPriority(uint8 number)

说明： 获取指定中断编号的优先级。

参数： number：中断编号。有效范围：[0-31]

返回值： 中断优先级

void CyIntSetPending(uint8 number)

说明： 迫使指定中断编号待定。

参数： number：中断编号。有效范围：[0-31]

返回值： 无

void CyIntClearPending(uint8 number)

说明： 清除指定中断编号的待处理中断。

参数： number：中断编号。有效范围：[0-31]

返回值： 无

6 引脚

除了为作为引脚组件一部分的引脚提供的功能外，还针对 PSoC 3/PSoC 5/PSoC 5LP 组件在 *cypins.h* 文档中提供了引脚宏的程序库。没有针对 PSoC 4 组件的引脚宏程序库。这些宏充分利用端口引脚配置寄存器，该寄存器可用于 PSoC 3/PSoC 5/PSoC 5LP 组件上的每个引脚。寄存器的地址列于 *cydevice_trm.h* 文档中。各个引脚配置寄存器的名称为：

```
CYREG_PRTx_PCy
```

其中，x 指端口编号，y 指端口中的引脚编号。

PSoC 4 仅包含状态寄存器、数据输出寄存器和端口配置寄存器，因此宏使用两个参数：端口寄存器和引脚号。每个端口具有如下定义的寄存器地址：

```
CYREG_PRTx_DR
CYREG_PRTx_PS
CYREG_PRTx_PC
```

x 为端口号，第二个参数为引脚号。

PSoC 3/PSoC 5/PSoC 5LP API

uint8 CyPins_ReadPin(uint16/uint32 pinPC)

说明： 读取引脚上的当前值（引脚状态，PS）。

参数： pinPC：端口引脚配置寄存器（uint16 PSoC 3、uint32 PSoC 5）

返回值： 引脚状态

0：逻辑低值

非 0：逻辑高值

void CyPins_SetPin(uint16/uint32 pinPC)

说明： 将引脚的输出值（数据寄存器，DR）设置为逻辑高。请注意，这仅对配置为不由硬件驱动的软件引脚有效。

参数： pinPC：端口引脚配置寄存器（uint16 PSoC 3、uint32 PSoC 5）

返回值： 无

void CyPins_ClearPin(uint16/uint32 pinPC)

说明： 将引脚的输出值（数据寄存器，DR）清除为逻辑低。请注意，这仅对配置为不由硬件驱动的软件引脚有效。

参数： pinPC：端口引脚配置寄存器（uint16 PSoC 3、uint32 PSoC 5）

返回值： 无

void CyPins_SetPinDriveMode(uint16/uint32 pinPC, uint8 mode)

说明： 设置引脚的驱动模式 (DM)。

参数： pinPC：端口引脚配置寄存器（uint16 PSoC 3、uint32 PSoC 5）

mode：所需的驱动模式

定义	源
CY_PINS_DM_ALG_HIZ	模拟 HiZ
CY_PINS_DM_DIG_HIZ	数字 HiZ
CY_PINS_DM_RES_UP	电阻上拉
CY_PINS_DM_RES_DWN	电阻下拉
CY_PINS_DM_OD_LO	开漏驱低
CY_PINS_DM_OD_HI	开漏驱高
CY_PINS_DM_STRONG	强 CMOS 输出
CY_PINS_DM_RES_UPDWN	电阻上拉/下拉

返回值： 无

uint8 CyPins_ReadPinDriveMode(uint16/uint32 pinPC)

说明： 读取引脚的驱动模式 (DM)。

参数： pinPC：端口引脚配置寄存器（uint16 PSoC 3、uint32 PSoC 5）

返回值： 引脚的当前驱动模式

定义	源
CY_PINS_DM_ALG_HIZ	模拟 HiZ
CY_PINS_DM_DIG_HIZ	数字 HiZ
CY_PINS_DM_RES_UP	电阻上拉
CY_PINS_DM_RES_DWN	电阻下拉
CY_PINS_DM_OD_LO	开漏驱低
CY_PINS_DM_OD_HI	开漏驱高
CY_PINS_DM_STRONG	强 CMOS 输出
CY_PINS_DM_RES_UPDWN	电阻上拉/下拉

void CyPins_FastSlew(uint16/uint32 pinPC)

说明： 将引脚的斜率设置为快速沿速率。请注意，这仅适用于强输出驱动模式下的引脚，而不适用于电阻驱动模式下的引脚。

参数： pinPC：端口引脚配置寄存器（uint16 PSoC 3、uint32 PSoC 5）

返回值： 无

void CyPins_SlowSlew(uint16/uint32 pinPC)

说明： 将引脚的斜率设置为慢速沿速率。请注意，这仅适用于强输出驱动模式下的引脚，而不适用于电阻驱动模式下的引脚。

参数： pinPC：端口引脚配置寄存器（uint16 PSoC 3、uint32 PSoC 5）

返回值： 无

PSoC 4 API

CY_SYS_PINS_READ_PIN（portPS，引脚）

说明： 读取引脚上的当前值（引脚状态，PS）。

参数： portPS：端口引脚状态寄存器地址 (uint32)。*cydevice_trm.h* 文件中按照以下格式提供了对各端口的定义：CYREG_PRTx_PS，其中 x 为端口号 0 - 4。
pin：引脚编号 0 – 7。

返回值： 引脚状态：
0：逻辑低值
非 0：逻辑高值

CY_SYS_PINS_SET_PIN（portDR，引脚）

说明： 将引脚的输出值（数据寄存器，DR）设置为逻辑高。
请注意，这仅对配置为不由硬件驱动的软件引脚有效。

参数： portDR：输出引脚数据寄存器的地址 (uint32)。*cydevice_trm.h* 文件中按照以下格式提供了对各端口的定义：CYREG_PRTx_DR，其中 x 为端口号 0 - 4。
pin：引脚编号 0 – 7。

返回值： 无

CY_SYS_PINS_CLEAR_PIN（portDR，引脚）

说明： 该宏将指定引脚的状态设置为 0.

参数： portDR：输出引脚数据寄存器的地址 (uint32)。*cydevice_trm.h* 文件中按照以下格式提供了对各端口的定义：CYREG_PRTx_DR，其中 x 为端口号 0 - 4。
pin：引脚编号 0 – 7。

返回值： 无

CY_SYS_PINS_SET_DRIVE_MODE（portPC，引脚，模式）

说明： 设置引脚的驱动模式 (DM)。

参数： portPC：端口配置寄存器的地址 (uint32)。*cydevice_trm.h* 文件中按照以下格式提供了对各端口的定义：CYREG_PRTx_DR，其中 x 为端口号 0 - 4。
pin：引脚编号 0 – 7。
mode：所需的驱动模式

定义	源
CY_SYS_PINS_DM_ALG_HIZ	模拟 HiZ
CY_SYS_PINS_DM_DIG_HIZ	数字 HiZ
CY_SYS_PINS_DM_RES_UP	电阻上拉
CY_SYS_PINS_DM_RES_DWN	电阻下拉
CY_SYS_PINS_DM_OD_LO	开漏驱低
CY_SYS_PINS_DM_OD_HI	开漏驱高
CY_SYS_PINS_DM_STRONG	强 CMOS 输出
CY_SYS_PINS_DM_RES_UPDWN	电阻上拉/下拉

返回值： 无

CY_SYS_PINS_READ_DRIVE_MODE（portPC，引脚）

说明： 读取引脚的驱动模式 (DM)。

参数： portPC：端口配置寄存器的地址 (uint32)。*cydevice_trm.h* 文件中按照以下格式提供了对各端口的定义：CYREG_PRTx_DR，其中 x 为端口号 0 - 4。
pin：引脚编号 0 – 7。

返回值： 引脚的当前驱动模式

定义	源
CY_SYS_PINS_DM_ALG_HIZ	模拟 HiZ
CY_SYS_PINS_DM_DIG_HIZ	数字 HiZ
CY_SYS_PINS_DM_RES_UP	电阻上拉
CY_SYS_PINS_DM_RES_DWN	电阻下拉
CY_SYS_PINS_DM_OD_LO	开漏驱低
CY_SYS_PINS_DM_OD_HI	开漏驱高
CY_SYS_PINS_DM_STRONG	强 CMOS 输出
CY_SYS_PINS_DM_RES_UPDWN	电阻上拉/下拉

7 寄存器访问

宏的库提供对组件寄存器的读写访问。这些宏与生成的 *cydevice_trm.h* 和 *cyfitter.h* 文件中定义的值一起使用。应使用这些宏，而不是用于实现这些宏的函数访问寄存器。这可以生成与组件相关的代码。

PSoC 3 是 8 位结构，因此处理器没有字节顺序。但是，8 位结构的编译器将实现字节顺序。对于 PSoC 3，Keil 编译器实现大端模式（MSB 在最低地址）排列。PSoC 4/PSoC 5/PSoC 5LP 处理器结构使用小端模式排列。

所有结构中的 SRAM 和闪存都是通过使用结构和编译器的字节顺序实现的。但是，所有这些芯片中的寄存器是以小端模式布置的。这些宏允许访问寄存器以匹配小端模式排列。如果在不使用这些宏的情况下在多字节寄存器上进行操作，您必须考虑特定结构的字节顺序。示例包括使用 DMA 以在存储器和寄存器之间传输，以及通过存储器中字节阵列的函数调用。

PSoC 3 是 8 位处理器，因此将以每次一字节的速度完成所有访问。PSoC 5 / PSoC 5LP 将使用正确的 8 位、16 位和 32 位访问权限进行访问。PSoC 4 要求这些访问与数据操作的宽度保持一致。

API

uint8 CY_GET_REG8(uint16/uint32 reg)

说明： 从指定的寄存器读取 8 位的值。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

返回值： 读取值

void CY_SET_REG8(uint16/uint32 reg, uint8 value)

说明： 向指定寄存器写入 8 位的值。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

value：要写入的值

返回值： 无

uint16 CY_GET_REG16(uint16/uint32 reg)

说明： 从指定的寄存器读取 16 位的值。该宏可实现正确操作所需的字节交换。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

返回值： 读取值

void CY_SET_REG16(uint16/uint32 reg, uint16 value)

说明： 向指定寄存器写入 16 位的值。该宏可实现正确操作所需的字节交换。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

value：要写入的值

返回值： 无

uint32 CY_GET_REG24(uint16/uint32 reg)

说明： 从指定寄存器读取 24 位的值。该宏可实现正确操作所需的字节交换。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

返回值： 读取值

void CY_SET_REG24(uint16/uint32 reg, uint32 value)

说明： 向指定寄存器写入 24 位的值。该宏可实现正确操作所需的字节交换。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

value：要写入的值

返回值： 无

uint32 CY_GET_REG32(uint16/uint32 reg)

说明： 从指定寄存器读取 32 位的值。该宏可实现正确操作所需的字节交换。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

返回值： 读取值

void CY_SET_REG32(uint16/uint32 reg, uint32 value)

说明： 向指定寄存器写入 32 位的值。该宏可实现正确操作所需的字节交换。对于 PSoC 3，地址必须在较低的 64K 地址范围内。

参数： reg：寄存器地址（uint16 PSoC 3、uint32 PSoC 4/PSoC 5）

value：要写入的值

返回值： 无

uint8 CY_GET_XTND_REG8(uint32 reg)

说明： 从指定的寄存器读取 8 位的值。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 get 函数多。与 PSoC 4/PSoC 5 的 CY_GET_REG8 功能相同。

参数： reg：寄存器地址

返回值： 读取值

void CY_SET_XTND_REG8(uint32 reg, uint8 value)

说明： 向指定寄存器写入 8 位的值。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 set 函数多。与 PSoC 4/PSoC 5 的 CY_SET_REG8 功能相同。

参数： reg：寄存器地址

value：要写入的值

返回值： 无

uint16 CY_GET_XTND_REG16(uint32 reg)

说明： 从指定的寄存器读取 16 位的值。该宏可实现正确操作所需的字节交换。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 get 函数多。与 PSoC 4/PSoC 5 的 CY_GET_REG16 功能相同。

参数： reg：寄存器地址

返回值： 读取值

void CY_SET_XTND_REG16(uint32 reg, uint16 value)

说明： 向指定寄存器写入 16 位的值。该宏可实现正确操作所需的字节交换。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 set 函数多。与 PSoC 4/PSoC 5 的 CY_SET_REG16 功能相同。

参数： reg：寄存器地址

value：要写入的值

返回值： 无

uint32 CY_GET_XTND_REG24(uint32 reg)

说明： 从指定寄存器读取 24 位的值。该宏可实现正确操作所需的字节交换。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 get 函数多。与 PSoC 4/PSoC 5 的 CY_GET_REG24 功能相同。

参数： reg：寄存器地址

返回值： 读取值

void CY_SET_XTND_REG24(uint32 reg, uint32 value)

说明： 向指定寄存器写入 24 位的值。该宏可实现正确操作所需的字节交换。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 set 函数多。与 PSoC 4/PSoC 5 的 CY_SET_REG24 功能相同。

参数： reg：寄存器地址

value： 要写入的值

返回值： 无

uint32 CY_GET_XTND_REG32(uint32 reg)

说明： 从指定寄存器读取 32 位的值。该宏可实现正确操作所需的字节交换。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 get 函数多。与 PSoC 4/PSoC 5 的 CY_GET_REG32 功能相同。

参数： reg：寄存器地址

返回值： 读取值

void CY_SET_XTND_REG32(uint32 reg, uint32 value)

说明： 向指定寄存器写入 32 位的值。该宏可实现正确操作所需的字节交换。支持 PSoC 3 的完整地址空间，但是需要的执行周期比标准寄存器的 set 函数多。与 PSoC 4/PSoC 5 的 CY_SET_REG32 功能相同。

参数： reg：寄存器地址

value：要写入的值

返回值： 无

8 DMA

DMA 文件向 DMA 控制器、DMA 通道和传输描述符提供 API 函数。该 API 是库版本，不是用户将 DMA 组件置于原理图时生成的代码。自动生成的代码将在该模块中使用这些 API。

有关详细信息，请参考 DMA 组件数据手册。

注意：仅当 DMA 组件被置于原理图上时，才需（通过从启动代码调用 CyDmacConfigure() 函数）创建待分配的“传输描述符”链接列表。

PSoC 4 器件无 DMA 功能。

此页特意留白。

9 闪存和 EEPROM

PSoC 3/PSoC 5/PSoC 5LP 实现

存储器架构

PSoC 组件中的闪存旨在为用户固件、用户配置数据、批量数据存储和可选 ECC 数据提供非易失性存储空间。PSoC EEPROM 存储器是一种按字节寻址的非易失性存储器。有关详细信息，请参见器件数据手册和 TRM。

闪存被组织为一组阵列。每个阵列由 64、128 或 256 行组成。PSoC 3 结构含有一个闪存阵列，其大小为 16 KB、32 KB 或 64 KB 与 ECC 字节之和。因此，闪存的唯一有效阵列 ID 为 0。PSoC 5 结构要么包含一个 128 行或 256 行的阵列，要么包含多个 256 行的阵列。闪存/EEPROM 的定义如下。这些定义可用于特定于组件的编号。某一阵列的行数也用阵列大小除以每行的大小得出。其中每行均含有 256 个数据字节外加 32 个字节。这些字节要么用于错误校正码 (ECC)（在设计范围资源编辑器中启用 ECC 选项），要么用于数据存储（禁用 ECC 时）。组件配置数据或用户数据可根据 ECC 存储器选项中的“存储配置数据”存储在此处。

EEPROM 也被组织为一组阵列。PSoC 3 和 PSoC 5 架构均有一个 EEPROM 阵列，其大小为 512 字节、1 KB 或 2 KB。阵列则可根据组件包含 32、64 或 128 行。每行含 16 字节数据。

闪存和 EEPROM API 提供下列特定于组件的定义：

值	说明
CY_FLASH_BASE	闪存的基本指针。
CY_FLASH_SIZE	闪存的大小。
CY_FLASH_SIZEOF_ARRAY	闪存阵列的大小。
CY_FLASH_SIZEOF_ROW	闪存行的大小。
CY_FLASH_SIZEOF_ECC_ROW	ECC 行的大小。
CY_FLASH_NUMBER_ROWS	闪存行的数量。
CY_FLASH_NUMBER_ARRAYS	闪存阵列的数量。
CY_EEPROM_BASE	EEPROM 存储器的基本指针。
CY_EEPROM_SIZE	EEPROM 存储器的大小。
CY_EEPROM_SIZEOF_ARRAY	EEPROM 阵列的大小。
CY_EEPROM_SIZEOF_ROW	EEPROM 行的大小。
CY_EEPROM_NUMBER_ROWS	EEPROM 行的数量。
CY_EEPROM_NUMBER_ARRAYs	EEPROM 阵列的数量。

ECC

您可为高可靠应用启用 ECC。每 8 个字节，ECC 便可纠正一位错误并检测多位错误。有关使用 ECC 的详细信息，请参见 TRM 的“闪存程序存储器”一章。

PSoC 5 组件不提供闪存错误检测或更正功能，但可用 ECC 闪存空间存储数据。ECC 用于数据存储时，组件配置或用户数据也将存储在此处。在此情况下，最常见的用法是存储配置数据。此操作直接由 PSoC Creator 支持。

禁用错误检测或更正功能进行闪存编程时（此时 ECC 闪存空间将用于数据存储），写入一行数据有多种方法：

- 包括 ECC 的整行
- 不包括 ECC 的整行
- 仅 ECC 存储器

使用闪存和 EEPROM

闪存和 EEPROM 通过系统性能控制器 (SPC) 进行编程。为与 SPC 进行交互，信息将被推送入一个寄存器然后再从中取出。特定于闪存/EEPROM 的 API 将通过提取详细信息简化与 SPC 的交互。

闪存和 EEPROM 都将被映射到存储器空间中，并可直接读取。要获取某特定阵列 ID 的首个闪存/EEPROM 行的地址，应将阵列大小乘以阵列 ID，然后加入闪存/EEPROM 基地址中。要访问阵列 ID 相同的任意行，应将行的大小乘以要访问的行数，然后加入到特定阵列的首行地址中。

要使用闪存和 EEPROM 的写入功能，至少须获取一次温度。如果该应用程序使用环境中的核心温度变化了 10°C 或以上，则应刷新该温度以调整闪存写入时间从而达到最佳性能。通过调用 CySetTemp() 函数可获得核心温度。该函数将查询 SPC 以获取核心温度并将其存储在全局变量中。后续执行闪存和 EEPROM 写入操作时便会用到此变量。

如果“启用 ECC”选项已禁用，您还需分配缓冲区并将其传入 CySetFlashEEBuffer() 函数。与 SPC 通信时，该缓冲区将用于存储中间数据。有关缓冲区分配的详细信息，请参见本章 API 一节的 CySetFlashEEBuffer() 函数说明。

闪存或 EEPROM 可通过调用 CyWriteRowData() 函数每次写入一行。首个参数用于确定对象为闪存还是 EEPROM 阵列。属于闪存的阵列数和属于 EEPROM 的阵列数特定于所选实际器件。如需查询有效阵列 ID 的信息，请参见器件 TRM。每个阵列 ID 均由 0 开始对闪存行进行编号。

闪存和 EEPROM API

cystatus CySetTemp()

说明： 获取核心温度并将结果保留在由闪存和 EEPROM 写入函数使用的静态位置。在执行一系列闪存/ EEPROM 写入函数前，必须调用一次该函数。

参数： 无

返回值： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	正在进行闪存/EEPROM 写入
CYRET_UNKNOWN	失败

副作用和限制： 在 SPC 回到空闲状态之前，此函数不会返回。

*cystatus CySetFlashEEBuffer(uint8 *buffer)*

说明： 设置用于完整闪存行的临时存储缓冲区，以及在写入闪存和 EEPROM 期间使用的相关 ECC。该缓冲区仅在闪存 ECC 禁用时需要使用。

参数： uint8 *buffer：分配的缓冲区的地址，其大小等于闪存行和 ECC 行大小的总和。

返回值： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	正在进行闪存/EEPROM 写入

*cystatus CyWriteRowFull(uint8 arrayId, uint16 rowAddress, uint8 *rowData, uint16 rowSize)*

说明： 允许对某一行进行擦除和编程。

闪存配置	说明
启用 ECC－开 将配置数据存储在 ECC 存储器中－不可用	数据被写入闪存行。自动计算和写入这些数据的 ECC。 数据大小等于闪存行大小。
启用 ECC－关 将配置数据存储在 ECC 存储器中－开	数据被写入闪存和 ECC 行。 防止覆盖存储在 ECC 闪存空间中的配置数据，数据的大小必须等于闪存行的大小。
启用 ECC－关 将配置数据存储在 ECC 存储器中－关	数据被写入闪存和 ECC 行。 数据大小等于闪存行和 ECC 行大小之和。

如果阵列为 EEPROM 阵列，则数据大小等于 EEPROM 行的大小。

参数： uint8 arrayId：待写入的阵列的 ID。写入的类型（闪存或 EEPROM）取决于阵列 ID。部件中的阵列是连续的，并从特定存储器类型的第一个 ID 开始。闪存的阵列 ID 介于 0x00 到 0x3F，而 EEPROM 的阵列 ID 介于 0x40 到 0x7F。

uint16 rowAddress：指定 arrayId 中的行地址。

uint8 *rowData：待编程的数据的地址。**注意：**该缓冲区与 CySetFlashEEBuffer() 函数分配的缓冲区不同。

uint16 rowSize：行数据的字节数

返回值： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	正在进行闪存/EEPROM 写入
CYRET_CANCELED	未接受命令
其他非零数据	失败

*cystatus CyWriteRowData(uint8 arrayId, uint16 rowAddress, uint8 *rowData)*

说明： 写入闪存行或 EEPROM 行。

闪存配置	说明
启用 ECC－开 将配置数据存储在 ECC 存储器中－不可用	数据被写入闪存行。自动计算和写入这些数据的 ECC。 传送给此函数的数据大小等于闪存行的大小。
启用 ECC－关 将配置数据存储在 ECC 存储器中－开/关	数据被写入闪存。 使用 CySetFlashEEBuffer() 函数提供的缓冲区保留存储在 ECC 存储器中的数据。

如果阵列为 EEPROM 阵列，则数据大小等于 EEPROM 行的大小。

参数： uint8 arrayId：待写入的阵列的 ID。写入的类型（闪存或 EEPROM）取决于阵列 ID。部件中的阵列是连续的，并从特定存储器类型的第一个 ID 开始。闪存的阵列 ID 介于 0x00 到 0x3F，而 EEPROM 的阵列 ID 介于 0x40 到 0x7F。

uint16 rowAddress：指定 arrayId 中的行地址。

uint8 *rowData：待编程的数据的地址。

返回值： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	正在进行闪存/EEPROM 写入
CYRET_CANCELED	未接受命令
其他非零数据	失败

*cystatus CyWriteRowConfig(uint8 arrayId, uint16 rowAddress, uint8 *rowData)*

说明： 写入闪存的 ECC 部分。该函数仅对闪存阵列 ID（而非 EEPROM）有效。

闪存配置	说明
启用 ECC－开 将配置数据存储在 ECC 存储器中－不可用	此配置无法使用这一功能，因为 ECC 存储在 ECC 存储器中。
启用 ECC－关 将配置数据存储在 ECC 存储器中－开	此配置无法使用这一功能，因为 ECC 存储在 ECC 存储器中。
启用 ECC－关 将配置数据存储在 ECC 存储器中－关	数据写入 ECC 行。 使用 CySetFlashEEBuffer() 函数提供的缓冲区保留存储在 闪存行中的数据。

参数： uint8 arrayId：待写入的阵列的 ID。部件中的阵列是连续的，并从特定存储器类型的第一个 ID 开始。闪存阵列 ID 是从 0x00 到 0x3F。

uint16 rowAddress：指定 arrayId 中的行地址。

uint8 *rowECC：待编程的数据的地址。

返回值： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	正在进行闪存/EEPROM 写入
CYRET_CANCELED	未接受命令
其他非零数据	失败

void CyFlash_Start()

说明： 启用闪存。默认情况下将启用闪存。
对于 PSoC 5，相同的位可同时控制 EEPROM 和闪存。启动或终止其中任一项均会导致二者同时发生作用。

参数： 无

返回值： 无

void CyFlash_Stop()

说明： 禁用闪存。只要 CPU 当前正在运行，便会忽略该设置。该设置仅当 CPU 禁用后有效。
对于 PSoC 5，相同的位可同时控制 EEPROM 和闪存。启动或终止其中任一项均会导致二者同时发生作用。

参数： 无

返回值： 无

void CyFlash_SetWaitCycles(uint8 freq)

说明： 设置缓存在对闪存返回的数据进行采样前，缓存需等待的时钟周期数。提高 CPU 时钟频率前必须调用此函数。可在降低 CPU 时钟频率后选择性地调用此函数，以提高 CPU 性能。

参数： freq：CPU 的工作频率单位为兆赫兹。

返回值： 无

void CyEEPROM_Start()

说明： 启用 EEPROM。
EEPROM 由单独的位控制，且必须先启动方可进行使用。
对于 PSoC 5，相同的位可同时控制 EEPROM 和闪存。启动或终止其中任一项均会导致二者同时发生作用。PSoC 5 默认启用 EEPROM。

参数： 无

返回值： 无

void CyEEPROM_Stop()

说明： 禁用 EEPROM。
EEPROM 由单独的位控制并可单独停止。
对于 PSoC 5，相同的位可同时控制 EEPROM 和闪存。启动或终止其中任一项均会导致二者同时发生作用。

参数： 无

返回值： 无

void CyEEPROM_ReadReserve()

说明： 请求访问 EEPROM 以便进行读取，并等待直至取得访问权限。对 EEPROM 的访问将在写入 EEPROM 的控制器以及对 EEPROM 执行常规读取访问之间进行仲裁。用户无需保留对 EEPROM 的读取访问权限。但如果写入操作仍处于活动状态便尝试进行读取，则会产生错误并返回不正确的数据。

参数： 无

返回值： 无

void CyEEPROM_ReadRelease()

说明： 释放对 EEPROM 的读取保留权。如果 EEPROM 已保留用于读取，则必须将其释放才能对 EEPROM 执行后续写入操作。

参数： 无

返回值： 无

PSoC 4 实现

存储器架构

PSoC 4 含有 32 KB 闪存且不支持 ECC 功能。所有 32 KB 闪存均位于同一闪存阵列中，该阵列有 256 行。因此，每行大小为 128 字节。此外，PSoC 4 不含 EEPROM 存储器，因此 PSoC 4 无法使用 EEPROM 组件。

闪存和 API 提供下列特定于组件的定义：

值	说明
CY_FLASH_BASE	闪存的基本指针。
CY_FLASH_SIZE	闪存的大小。
CY_FLASH_SIZEOF_ARRAY	闪存阵列的大小。
CY_FLASH_SIZEOF_ROW	闪存行的大小。
CY_FLASH_SIZEOF_ECC_ROW	ECC 行的大小。
CY_FLASH_NUMBER_ROWS	闪存行的数量。
CY_FLASH_NUMBER_ARRAYS	闪存阵列的数量。

闪存操作

通过 SPC 接口对闪存进行编程。为了与 SPC 接口进行通信，信息从单个寄存器中推进和拉出。闪存专用 API 通过提取详细信息简化了与 SPC 之间的交互。

闪存被直接映射到存储器空间并可直接读取。

闪存 API

cystatus CySysFlashWriteRow(uint32 rowNum, const uint8 rowData[])

说明： 写入一行闪存。

参数： uint32 rowNum：行编号。每行 128 字节，所以 32 KB 闪存存储器的有效范围为 [0-255]。

uint8 rowData：要写入的字节阵列。

返回值： 状态：

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	使用的闪存
CYRET_CANCELED	未接受命令
CYRET_BAD_PARAM	一个或多个无效参数
其他非零数据	失败

副作用和限制： 调用此函数前必须先启用 IMO。闪存写入硬件操作取决于 IMO。

void CySysFlashSetWaitCycles(uint32 freq)

说明： 设置缓存在对闪存返回的数据进行采样前，缓存需等待的时钟周期数。提高 SYSCLK 时钟频率前须调用此函数。可在降低 SYSCLK 时钟频率后选择性调用此函数，以提高 CPU 性能。

参数： freq：有效范围 [3-48]。IMO 运行频率。
注意：无效的频率将被忽略。

返回值： 无

副作用和限制： 无

此页特意留白。

10 引导加载程序移植

本章包含有关 PSoC Creator 2.1 中引入的引导加载程序架构的信息，以及在将引导加载程序系统设计从 PSoC Creator 2.0 或更早版本进行移植时可能遇到的已知问题。

简介

从 PSoC Creator 2.1 开始，已重新组织引导加载程序系统以提供更多的配置选项。在之前发布的版本中，引导加载程序系统为 cy_boot 组件的一部分（在所有设计上自动且不可见地得到了实例化的必需组件）。目前引导加载程序系统包含两个独立的引导加载程序组件和可引导加载组件。您可以在 PSoC Creator 组件目录中找到这些组件，还包括和其他组件一样的组件数据手册。

此外，已将引导加载程序系统配置选项从 PSoC Creator 设计范围资源 (DWR) 文件移入相应的引导加载程序和可引导加载组件配置对话框中。有关更多信息，请参见本章的移植引导加载程序设计 “移植引导加载程序设计”和移植可引导加载设计“移植可引导加载设计”两节。

可通过更新 cy_boot 组件至 3.0 或以上版本自动切换到新的引导加载程序系统。然而，某些更改可能影响现有引导加载程序和可引导加载设计。这需要在移至新的引导加载程序系统架构时进行维护。引导加载程序系统配置设置须手动移动。

三个关键移植备选方案包括：

- 完成移植到 PSoC Creator 2.1 或更高版本
- 移植到 PSoC Creator 2.1 或更高版本，但不进行 cy_boot 组件更新
- 仅为可引导加载设计移植至 PSoC Creator 2.1 或更高版本

完成移植到 PSoC Creator 2.1 或更高版本

在这种情况下，可将 PSoC Creator 2.1 或更高版本与 cy_boot 组件的最新版本及新的引导加载程序架构配合使用。

当您打开一个项目时，如果该项目最后保存在 PSoC Creator 早期发布的版本中，系统会提示您更新组件到最新版本。

- 使用组件更新工具并选择最新版本的 cy_boot 组件。建议一起更新所有的组件。使用“更新所有组件至最新版本（Update All to Latest）”按钮以确保选择最新的版本进行更新。
- 打开引导加载程序系统设计，并将引导加载程序组件放置在**引导加载程序**或**多应用引导加载程序**应用类型原理图上。将可引导加载组件放置在**可引导加载** 应用类型原理图上。有关更多信息，请参见项目应用类型属性更改“项目应用类型属性更改”。

- 将引导加载程序系统设置从 DWR 文件转为组件配置对话框。有关详细信息，请参见移植引导加载程序设计**“移植引导加载程序设计”和**移植可引导加载设计**“移植可引导加载设计”**。

移植到 PSoC Creator 2.1 或更高版本，但不进行 cy_boot 组件更新

在这种情况下，可将 PSoC Creator 2.1 或更高版本与在 PSoC Creator 早前版本中创建的引导加载程序系统设计配合使用。要继续与引导加载程序系统配合使用，无需其他额外关注事项。

除 cy_boot 组件外，可将在设计原理图上出现的所有组件更新至最新可用版本。如果没有同时切换至新的引导加载程序架构，不能将 cy_boot 组件更新至 3.0 或更高版本。

注意：随 PSoC Creator 2.1 或更高版本附带的某些组件最新版本要求 cy_boot 3.0 或更高版本，所以在没有更新 cy_boot 时无法对他们进行更新。

仅为可引导加载设计移植至 PSoC Creator 2.1 或更高版本

在这种情况下，只有可引导加载项目被移植至 PSoC Creator 2.1 或更高版本及 cy_boot 3.0 或更高版本。现有的引导加载程序设计与新建或移植的可引导加载设计完全兼容。不包含更新 cy_boot 组件的 PSoC Creator 2.1 或更高版本，或 PSoC Creator 的早期版本可用于更改引导加载程序设计。

移植引导加载程序设计

下表显示 PSoC Creator 早期版本中的引导加载程序系统和 PSoC Creator 2.1 中引入的引导加载程序系统之间的关联。

2.1 之前的 PSoC Creator 版本（DWR“系统”选项卡中的“引导加载程序”部分）	**PSoC Creator 2.1 或更高版本以及 cy_boot 3.0 或更高版本（引导加载程序组件）**
IO 组件	通信组件
由设计的**应用类型**属性设置 请参见项目应用类型属性更改“项目应用类型属性更改”一节。	多应用引导加载程序
等待命令	等待命令
等待命令的时间 (ms)	等待命令的时间 (ms)
版本	引导加载应用程序版本
数据包校验类型	数据包校验和类型
Fast Application Verification（快速应用验证）	快速可引导加载应用验证
无	引导加载应用程序验证
无	可选命令

有关 PSoC Creator 引导加载程序系统功能的详细信息，请参见引导加载程序组件数据手册。

移植可引导加载设计

下表显示 PSoC Creator 2.1 中引入的可引导加载系统和 PSoC Creator 2.1 之前版本中的可引导加载系统之间的关联。

2.1 之前的 PSoC Creator 版本 （DWR“系统”选项卡中的“可引导加载”部分）	PSoC Creator 2.1 或更高版本以及 cy_boot 3.0 或更高版本 （可引导加载组件）
版本	应用版本
应用 ID	应用 ID
Custom ID（自定义 ID）	应用自定义 ID
无	手动应用图像放置
无	放置地址
在项目“依赖关系”窗口的**引导加载程序**选项卡中设置请参见项目应用类型属性更改“项目应用类型属性更改”一节。	引导加载程序十六进制文件（引导加载程序组件配置对话框中的**依赖关系**选项卡）。

早先在可引导加载项目中声明的 CyBtldr_Load() 函数被移除。改用 Bootloadable_1_Load() 函数，其中 Bootloadable_1 为可引导加载组件的实例名。

有关 PSoC Creator 引导加载程序系统功能的详细信息，请参见可引导加载组件数据手册。

项目应用类型属性更改

在 PSoC Creator 2.1 之前版本中，**应用类型**选项被用于选择创建项目时生成的应用类型。从 PSoC Creator 2.1 开始，**应用类型**项目属性提供了验证设计是否按照预期创建的方法。

下表将**应用类型**选项值与放置的组件配置相互关联：

应用类型	预期组件
引导加载程序	具有多应用引导加载程序选项的引导加载程序组件禁用。
多应用引导加载程序	具有多应用引导加载程序选项的引导加载程序组件启用
可引导加载	可引导加载组件。

应用类型选项在**新建项目**对话框中高级部分设置，以后可在**代码生成**部分下面的项目**构建设置**窗口中进行修改。

移植可引导加载依赖关系至引导加载程序设计

在 PSoC Creator 2.1 之前版本中，项目依赖关系窗口中的**引导加载程序**选项卡可将可引导加载项目连接至引导加载程序项目。通过将可引导加载组件引入 PSoC Creator 2.1 及 cy_boot version 3.0 或更高版本，该选项被移到可引导加载组件配置对话框的**依赖关系**（Dependencies）选项卡中。有关详细信息，请参考组件数据手册。

此页特意留白。

11 系统函数

这些函数适用于所有架构的器件。

通用 API

uint8 CyEnterCriticalSection(void)

说明： CyEnterCriticalSection 禁用中断，并返回一个值，表明先前是否启用了中断（实际值取决于器件架构）。

注意：实现 CyEnterCriticalSection 能够控制 IRQ 使能位，并使中断仍处于启用状态。对所有架构器件而言，中断位的测试和设置并非原子操作。因此，为避免破坏处理器状态，所有中断子程序必须将中断使能位恢复为输入时的初始状态。

参数： 无

返回值： uint8

PSoC 3 — 返回值包含两位：

位 0：如果在调用 CyEnterCriticalSection 之前，中断已启用，则返回 1。

位 1：如果在调用 CyEnterCriticalSection 之前，IRQ 生成已禁用，则返回 1。

PSoC 4/PSoC 5/PSoC 5LP—如果之前已启用中断，则返回 0；如果之前已禁用中断，则返回 1。

void CyExitCriticalSection(uint8 savedIntrStatus)

说明： 如果在调用 CyExitCriticalSection 前，中断已启用，则 CyExitCriticalSection 将重新启用中断。参数应为 CyEnterCriticalSection 返回的值。

参数： uint8 savedIntrStatus：CyEnterCriticalSection 函数返回的保存中断状态。

返回值： 无

void CYASSERT(uint32 expr)

说明： 该宏用于计算表达式，如果为假（计算结果为 0），处理器将停止。除非 NDEBUG 已定义，否则需计算此宏。如果 NDEBUG 已定义，则不针对该宏生成任何代码。默认情况下，NDEBUG 定义了发布版本设置，未定义调试版本设置。

参数： expr：逻辑表达式。如果为假，则启用。

返回值： 无

void CyHalt(uint8 reason)

说明： 停止 CPU。

参数： reason：要被传递的值，用于调试。此值可用于了解调用 CyHalt() 的原因。

返回值： 无

void CySoftwareReset(void)

说明： 强制对器件进行软件复位。

参数： 无

返回值： 无

CyDelay API

有四个可实现基于软件的简单延迟循环的 CyDelay API。循环补偿总线时钟频率。

CyDelay 函数提供最小延迟。如果处理器发生中断，将延长循环的长度直至其实现中断。其他间接因素，包括函数进入和跳出，可能也会影响执行函数所花费的总时长。当额定延迟时间较小时，这种现象将尤为明显。

void CyDelay(uint32 milliseconds)

说明： 延迟指定的毫秒数。默认情况下，延迟循环次数的计算基于输入 PSoC Creator 的时钟配置。如果在运行时更改了时钟配置，则 CyDelayFreq 函数会用于表示新的总线时钟频率。因为 CyDelay 用于若干个组件，所以，如果更改了时钟频率而没有更新延迟的频率设置，就会导致这些组件发生故障。

参数： milliseconds：延迟的毫秒数。

返回值： 无

副作用和限制： 通过假设启用了指令缓存，已实现 CyDelay。在 PSoC 5/PSoC 5LP 上禁用指令缓存时，CyDelay 要大两倍。例如，如果禁用了指令缓存，CyDelay(100) 将导致大约 200 ms 的延迟，而不是 100 ms。

void CyDelayUs(uint16 microseconds)

说明： 延迟指定的微秒数。默认情况下，延迟循环次数的计算基于输入 PSoC Creator 的时钟配置。如果在运行时更改了时钟配置，则 CyDelayFreq 函数会用于表示新的总线时钟频率。因为 CyDelayUs 用于若干个组件，所以，如果更改了时钟频率而没有更新延迟的频率设置，就会导致这些组件发生故障。

参数： microseconds：延迟的微秒数。

返回值： Void

副作用和限制： 通过假设启用了指令缓存，已实现 CyDelayUs。在 PSoC 5/PSoC 5LP 上禁用指令缓存时，CyDelayUs 导致的延迟将大两倍。例如，如果禁用了指令缓存，CyDelayUs(100) 将导致大约 200 µs 的延迟，而不是 100 µs。
如果总线时钟频率为非整小数，实际延迟将和额定值一样达到两次。实际延迟不能短于额定值。

void CyDelayFreq(uint32 freq)

说明： 设置总线时钟频率，此频率用于计算通过 CyDelay 实现延迟所需的周期数。默认情况下，使用的频率基于构建时由 PSoC Creator 确定的值。

参数： freq：单位为 Hz 的总线时钟频率。

0：使用默认值

非 0：设置频率值

返回值： 无

void CyDelayCycles(uint32 cycles)

说明： 采用软件延迟循环，延迟指定的循环次数。

参数： cycles：延迟的循环次数。

返回值： 无

PSoC 3/PSoC 5/PSoC 5LP 电压检测 APIs

当 Vdda 或 Vddd 超过定义范围时，可通过配置这些器件中的电压监控电路生成中断。低电压中断对模拟和数字供电均可用，高电压中断仅对模拟供电可用。低电压检测器的激发电平是独立可配置的。高电压检测器的激发电平固定为 5.75 V。可通过配置模拟和数字低电压监控电路对器件复位，而不是生成中断。

如果供电电压超过激发电平，RESET_CR1 寄存器中的位 [2:0] 控制电压监控电路是否生成中断。如果启用 LVI 中断，RESET_CR3 中的位 [7:6] 在发生低电压事件时控制器件是否复位。LVI 复位是精确复位电路的一部分并生成瞬时硬件 POR 复位。

电压监控电路的状态存储在两个不同寄存器中。如果低电压或高电压事件发生，RESET_SR0 寄存器的位 [2:0] 被设为 1。当读取或进行 POR 复位时，寄存器被清除。RESET_SR2 寄存器的位 [2:0] 保持电压监控电路输出的实时状态，这意味着事件发生期间这些位只能设置为“1”。

GlobalSignalRef 组件可将 LVI 和 HVI 中断信号连接至工程原理图中的其他组件，或对 LVI/HVI 事件执行中断时连接至中断组件。如果在组件中选择“低/高电压检测 (LVI/HVI)”，无论何时启用的 LVI 或 HVI 电路检测

到事件，GlobalSignalRef 输出被设为 1。只要发生 LVI 或 HVI 事件，输出保持为 1。有关详细信息，请参考 GlobalSignalRef 组件数据手册。

以下 API 函数可配置和管理电压监控电路和相关的中断状态寄存器。有关电压监控电路的更多信息，请参见器件 TRM 的“电压监控”一节和器件数据手册的“供电电压电平监控器”一节。

注意：对于 PSoC 3 芯片，由于要启用硬件繁忙，LVI/HVI 应在进入睡眠模式前禁用。应在进入睡眠模式前禁用硬件繁忙（通过 PSoC Creator 启动代码禁用），因为如果与器件其他唤醒源一起使用，将导致器件被锁定并停止代码进一步执行。

void CyVdLvDigitEnable(uint8 reset, uint8 threshold)

说明： 当 Vddd 处于或低于激发点时，启用数字低电压监控器的输出，在发生低电压事件时，选择器件是否生成中断或复位该部分，并设置电压激发电平。

参数： **reset**： 对数字 LVI 中断启用器件复位（对 PSoC 5 不可用）。

器件	复位值	定义	寄存器 [位]
PSoC 3	0	数字 LVI 事件中断	RESET_CR3 [6]
	非零	数字 LVI 事件复位	
PSoC 5	任意值	数字 LVI 事件中断	
PSoC 5LP	0	数字 LVI 事件中断	
	非零	数字 LVI 事件复位	

threshold：在约 250 mV 步进间隔中，设置数字低电压监控电路的激发点。

器件	阈值	定义	寄存器 [位]
PSoC 3	0x00 至 0x0F	范围：1.70 V (0x00) 至 5.45 V (0x0F)。	RESET_CR0 [3:0]
PSoC 5	0x03 至 0x0F	范围：2.45 V (0x03) 至 5.45 V (0x0F)。	
PSoC 5LP	0x00 至 0x0F	范围：1.70 V (0x00) 至 5.45 V (0x0F)。	

注意：对于 PSoC 5，小于 0x03 的值为无效值，必须跳过。

返回值： 无

副作用和限制 LVI 复位是瞬时的。当发生 LVI 复位，RESET_CR1 和 RESET_CR3 寄存器恢复到其默认值。这意味着 LVI 电路不再启用，器件退出复位。如果电源低于激发电平且固件启用 LVI 复位功能，组件将重新复位。只要电源低于激发电平或用户启用 LVI 复位功能，复位将继续。

当发生任何 LVI 复位时，RESET_SR0 和 RESET_SR2 状态寄存器被清除。这意味着在 LVI 复位过程中，模拟 LVI、数字 LVI 和模拟 HVI 状态位不是永久的。

void CyVdLvAnalogEnable(uint8 reset, uint8 threshold)

说明： 当 Vddd 处于或低于激发点时，启用模拟低电压监控器的输出，在发生低电压事件时，选择组件是否生成中断或复位该部分，并设置电压激发电平。

参数： **reset**： 对模拟 LVI 中断启用组件复位（对 PSoC 5 不可用）。

组件	复位值	定义	寄存器 [位]
PSoC 3	0	模拟 LVI 事件中断	RESET_CR3 [7]
	非零	模拟 LVI 事件复位	
PSoC 5	任意值	模拟 LVI 事件中断	
PSoC 5LP	0	模拟 LVI 事件中断	
	非零	模拟 LVI 事件复位	

threshold：在约 250 mV 步进间隔中，设置模拟低电压监控电路的激发点。

组件	阈值	定义	寄存器 [位]
PSoC 3	0x00 至 0x0F	范围：1.70 V (0x00) 至 5.45 V (0x0F)。	RESET_CR0 [7:4]
PSoC 5	0x03 至 0x0F	范围：2.45 V (0x03) 至 5.45 V (0x0F)。	
PSoC 5LP	0x00 至 0x0F	范围：1.70 V (0x00) 至 5.45 V (0x0F)。	

注意：对于 PSoC 5，小于 0x03 的值为无效值，必须跳过。

返回值： 无

副作用和限制 LVI 复位是瞬时的。当发生 LVI 复位，RESET_CR1 和 RESET_CR3 寄存器恢复到其默认值。这意味着 LVI 电路不再启用，组件退出复位。如果电源低于激发电平且固件启用 LVI 复位功能，组件将重新复位。只要电源低于激发电平或用户启用 LVI 复位功能，复位将继续。

当发生任何 LVI 复位时，RESET_SR0 和 RESET_SR2 状态寄存器被清除。这意味着在 LVI 复位过程中，模拟 LVI、数字 LVI 和模拟 HVI 状态位不是永久的。

void CyVdLvDigitDisable(void)

说明： 禁用数字低电压监控器（禁用中断和组件复位）。

参数： 无

返回值： 无

副作用和限制 中断和 LVI 复位禁用，但待处理中断和状态位未被清除。

void CyVdLvAnalogDisable(void)

说明： 禁用模拟低电压监控器（禁用中断和组件复位）。

参数： 无

返回值： 无

副作用和限制 中断和 LVI 复位禁用，但待处理中断和状态位未被清除。

void CyVdHvAnalogEnable(void)

说明： 通过启用模拟高电压监控器在以 5.75 V 阈值对 Vdda 进行检测时生成中断。

参数： 无

返回值： 无

void CyVdHvAnalogDisable(void)

说明： 禁用模拟高压监控器（禁用中断）。

参数： 无

返回值： 无

副作用和限制 禁用中断和 HVI 复位，但不清除待处理中断和状态位。

uint8 CyVdStickyStatus(uint8 mask)

说明： 读取和清除 RESET_SR0 寄存器中的电压检测状态位。供电超出检测器激发点范围时，电压监控电路将状态位设为 1。它们将一直设置为 1，直到被读取或发生 POR/LVI/PRES 复位。此功能使用影像寄存器，因此在影像寄存器中只有在参数传递的位会被清除。

参数： mask：要清除及返回的 RESET_SR0 影像寄存器中的位。

组件	值	定义	寄存器 [位]
PSoC 3/ PSoC 5/ PSoC 5LP	CY_VD_LVID (0x01)	数字 LVI 的持续状态。	RESET_SR0 [0]
	CY_VD_LVID (0x02)	模拟 LVI 的持续状态。	RESET_SR0 [1]
	CY_VD_HVIA (0x04)	模拟 HVI 的持续状态。	RESET_SR0 [2]

返回值： 状态。与用于掩码参数的枚举位的值相同。对未用于掩码参数的位返回零值。

副作用和限制 发生 LVI 复位时，RESET_SR0 状态寄存器被清除。这意味着在 LVI 复位过程中，电压检测状态位不是永久的，不能用于确定复位源。

uint8 CyVdRealTimeStatus(void)

说明： 在 RESET_SR2 寄存器中读取实时电压检测状态位。供电超出检测器激发点范围时，电压监控电路将状态位设置为 1；当供电在检测器激发点范围以内时，设置为 0。

参数： 无

返回值： RESET_SR2 寄存器中 LVID、LVIA 和 HVIA 位的状态。

组件	值	定义	寄存器 [位]
PSoC 3/ PSoC 5/ PSoC 5LP	0x01	数字 LVI 的实时状态。	RESET_SR0 [0]
	0x02	模拟 LVI 的实时状态。	RESET_SR0 [1]
	0x04	模拟 HVI 的实时状态。	RESET_SR0 [2]

副作用和限制 发生 LVI 复位时，RESET_SR2 状态寄存器被清除。这意味着在 LVI 复位过程中，电压检测状态位不是永久的，不能用于确定复位源。

PSoC 4 电压检测 API

void CySysLvdEnable(uint32 threshold)

说明： Vddd 位于或低于激发点时，启用低压监控器的输出功能，将器件配置为生成中断并设置电压激发电平。

参数： threshold：欠压检测电路的阈值选择。阈值变化量为这些典型电压选择的 +/- 2.5%。

定义	电压阈值
CY_LVD_THRESHOLD_1_75_V	1.75 V
CY_LVD_THRESHOLD_1_80_V	1.80 V
CY_LVD_THRESHOLD_1_90_V	1.90 V
CY_LVD_THRESHOLD_2_00_V	2.00 V
CY_LVD_THRESHOLD_2_10_V	2.10 V
CY_LVD_THRESHOLD_2_20_V	2.20 V
CY_LVD_THRESHOLD_2_30_V	2.30 V
CY_LVD_THRESHOLD_2_40_V	2.40 V
CY_LVD_THRESHOLD_2_50_V	2.50 V
CY_LVD_THRESHOLD_2_60_V	2.60 V
CY_LVD_THRESHOLD_2_70_V	2.70 V
CY_LVD_THRESHOLD_2_80_V	2.80 V
CY_LVD_THRESHOLD_2_90_V	2.90 V
CY_LVD_THRESHOLD_3_00_V	3.00 V
CY_LVD_THRESHOLD_3_20_V	3.20 V
CY_LVD_THRESHOLD_4_50_V	4.50 V

返回值： 无

void CySysLvdDisable(void)

说明： 禁用欠压检测。欠压中断被禁用。

参数： 无

返回值： 无

uint32 CySysLvdGetInterruptSource(void)

说明： 得到欠压检测中断状态（未清除）。

参数： 无

返回值： 中断请求值：

CY_SYS_LVD_INT－表示欠压检测中断

void CySysLvdClearInterrupt(void)

说明： 清除欠压检测中断状态。

参数： 无

返回值： 无

缓存功能

PSoC 3

默认启用 PSoC 3 缓存。可使用 PSoC Creator 的 Design-Wide Resources System Editor 来禁用该缓存。没有用于 PSoC 3 缓存处理的定义、函数或宏。

PSoC 5/PSoC 5LP

void CyFlushCache()

说明： 通过使所有条目失效清理 PSoC 5/PSoC 5LP 缓存。

参数： 无

返回值： 无

PSoC 4

PSoC 4 组件无缓存功能。

此页特意留白。

12 启动和连接

cy_boot 组件负责启动系统。已实现了以下功能：

- 提供复位矢量
- 组织处理器的执行
- 设置中断
- 设置堆栈，包括 8051 的重新进入堆栈
- 配置器件
- 通过初始化值对静态变量和全局变量进行初始化
- 清除所有剩余的静态变量和全局变量
- 保存复位状态
- 调用 main() C 入口点

有关 PSoC 3 和 PSoC 5 启动的详细信息，请参见应用笔记 AN60616。

器件启动过程对器件进行配置以符合数据手册和 PSoC Creator 项目规范。释放复位源后或电源上升结束后，开始启动。启动有两大主要部分：硬件启动和固件启动。硬件启动过程中，CPU 暂停运行，其他资源配置器件。在固件启动过程中，CPU 运行由 PSoC Creator 生成的代码以配置器件。启动结束后，器件已完全配置，其 CPU 开始执行用户编写的 main() 代码。

硬件启动配置器件以符合数据手册中的一般性能规范。硬件启动相位在电源上升或复位事件后开始。硬件启动有两个阶段：复位和引导。硬件启动结束后，闪存代码执行开始。

固件启动对 PSoC 器件进行配置以按照 PSoC Creator 工程中所述进行运行。固件启动在硬件启动后开始。固件启动完成后，PSoC 器件的 CPU 开始执行用户编写的 main() 代码。固件启动的主要任务是填充配置寄存器，使 PSoC 组件按 PSoC Creator 工程设计的那样运行。这包括配置模拟外设和数字外设，以及系统资源，如时钟和路由。

启动过程可能会被更改以更好的满足特定应用的需求。修改器件启动有两种方法：使用 PSoC Creator 设计范围资源 (DWR) 接口和修改器件启动代码。

PSoC 3

启动均由单一的汇编文件 (KeilStart.a51) 处理，该文件基于 Keil 提供的模板。不存在特别与连接相关联的文件。使用连接器指令。

PSoC 4/PSoC 5/PSoC 5LP

赛普拉斯已定制开发了启动和连接器脚本，但我们当前支持的两家工具链供货商都提供连接器实现样本和完整的库，可解决由自定义实现引起的多种问题。

最终二进制和十六进制图像的存储器布局以及在 PSoC 5 存储器中的位置都在 PSoC Creator 构建生成的连接器描述符 (.scat) 文件中进行了说明。通过进入**构建设置**（Build Setting）窗口并在**连接器**类别下的**自定义连接器脚本**字段中指定文件路径来使用自定义连接器描述符文件，而不是使用默认连接器描述符文件。

Realview 实现（适用于 MDK 和 RVDS）

使用所有的标准库（C standardlib、C microlib、fplib、mathlib）。默认情况下，所有这些库都已连接。

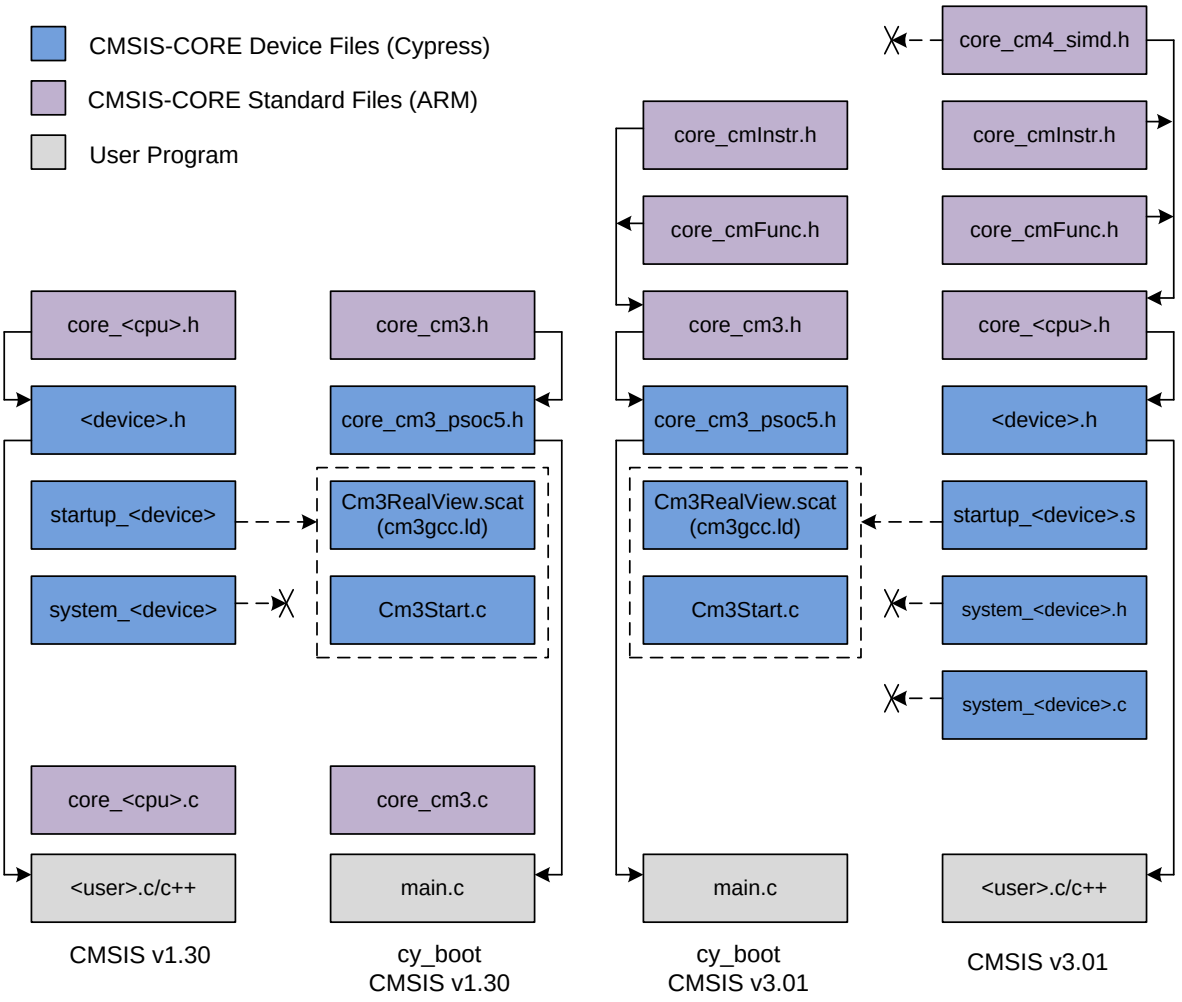
- 支持 RTOS 和用户替换子程序。因为库子程序记为“弱”，如果提供另一种实现，则允许进行替换，所以能够实现对 RTOS 和用户替换子程序的支持。
- 所提供的机制允许使用用户版本的连接器/分散文件进行替换。允许用户创建项目本地文件，且具有允许该文件规范代替自动提供的文件作为连接器/分散文件的构建设置，从而实现该功能。
- 目前，堆栈大小指定为规定值（栈为 4K，堆为 1K）。如有可能，应将指定堆栈大小的要求全部删除。如果不可删除，则这些值应为默认值，并可在 DWR GUI 中选择其他数值。
- 生成源代码树中的所有代码都将编译成单一的库，作为构建过程的一部分。然后，编译库将通过用户代码连接至最终链路。

CMSIS 支持

Cortex 微型控制器软件接口标准 (CMSIS) 是基于 ARM 的标准，用于与 Cortex M 系列处理器进行交互。支持包括多个层次。提供核心外设接入层 (CMSIS Core) 支持。

PSoC Creator 2.2 提供对 CMSIS Core 1.30 的支持。PSoC Creator 2.2 还提供使用自定义版本 CMSIS Core 的功能。

下显示 CMSIS Core 1.30 文件是如何集成到 cy_boot 组件以及 CMSIS Core 3.01 文件是如何被集成的。



以下介绍图中的每个文件：

- *Cm3Start.c* 和 *cm3gcc.ld* 文件（cy_boot 组件的一部分）含有 Cortex-M3 组件启动代码和中断矢量表并完全替代了 CMSIS startup_<device>.s 模版文件。
- 包含 CMSIS Core 标准文件的供应商特定组件文件 <组件>.h 在 cy_boot 组件中由 *core_cm3_psoc5.h* 表示。
- *core_cmInstr.h* 文件定义了访问专用 Cortex-M 指令的内在函数，*core_cmFunc.h* 文件提供了访问 Cortex-M 核心外设的函数。自 CMSIS Core 版本 2.0 以后，加入了这些文件。实际上，这些文件的内容代替了 CMSIS v1.30 中 *core_<cpu>.h* 和 *core_<cpu>.c* 文件内容。
- 加入 CMSIS SIMD 指令访问的 *core_cm4_simd.h* 文件仅与 Cortex-M4 相关。
- *system_<device>.h*、*system_<device>.c*—系统配置通用文件（即处理器时钟和存储器总线系统），在 *Cm3Start.c* 中部分涉及。

手动添加 CMSIS Core 文件

从 PSoC Creator 2.2 开始，向 DWR 文件的“系统”选项卡中添加了“包含 CMSIS Core 外设库文件”。默认启用该选项，CMSIS Core 1.30 文件被添加到项目中。如果您希望手动添加 CMSIS Core 文件，则应禁用此选项。

在 DWR 文件的“系统”选项卡上取消选中“包含 CMSIS Core 外设库文件” 选项以从 cy_boot 组件分离 CMSIS 1.30 文件。

将 CMSIS Core 文件添加到项目中：

- core_cmInstr.h
- core_cmFunc.h
- core_cm3.h

基于 CMSIS 供应商特定模板文件 (<device>.h)，创建器件头文件，从 core_cm3_psoc5.h 文件复制器件特定的定义并在文件顶部添加下列定义：

```
#include <cytypes.h>

#define __CHECK_DEVICE_DEFINES

#if CY_PSOC5A
    #define __CM3_REV              0x0101
#elif CY_PSOC5LP
    #define __CM3_REV              0x0201
#endif

#define __MPU_PRESENT              0
#define __NVIC_PRIO_BITS           3
#define __Vendor_SysTickConfig     0
```

将先前创建的供应商特定器件头文件包含到应用程序中。

GCC 实现

使用所有标准的 GCC 库（libcs3、libc、libcs3unhosted、libgcc、libcs3micro）。默认情况下，所有这些库都已连接。

复位状态保存 (PSoC 3/PSoC 5/PSoC 5LP)

只要器件启动，复位状态寄存器 (RESET_SR0) 的值就被读取并清除。该值被保存到一个全局 SRAM 变量中。请注意，IPOR、PRES 和 LVI 复位源清除 RESET_SR0 寄存器，而 WDT、软件复位和 XRES 复位源保存 RESET_SR0 寄存器。更多有关信息，请参见器件数据手册和 TRM。

一些 PSoC 3 器件执行附加软件的复位。除了之前已发生的软件器件复位外，它还会重新加载 NVL 并应用正确的设置。该操作对正常启动过程透明，并且不会干扰引导加载、调试或正常器件功能。有关更多信息，请参见器件勘误表。

为了保留在许多复位中持续的用户定义状态，可使用 RESET_SR0 寄存器中的 CY_RESET_GP0 和 CY_RESET_GP1 位。器件复位后，用 CyResetStatus 变量获取这些位值。引导加载程序和引导可加载工程可以使用这些位，用户不能使用。

uint8 CyResetStatus

名称	说明
CY_RESET_LVID	数字低压检测
CY_RESET_LVIA	模拟低压检测

CY_RESET_HVIA	模拟高压检测
CY_RESET_WD	看门狗复位
CY_RESET_SW	软件复位
CY_RESET_GP0	通用位 0
CY_RESET_GP1	通用位 1

此页特意留白。

13 看门狗定时器

PSoC 3/PSoC 5/PSoC 5LP API

void CyWdtStart(uint8 ticks, uint8 lpMode)

说明： 启用看门狗定时器。这个定时器已配置指定的计数间隔，CTW 已清除，低功耗模式设置已配置，且看门狗定时器已启用。

一旦看门狗定时器启用后，看门狗定时器的硬件实现可阻止对定时器所做的任何改动。看门狗定时器启用后，也可防止禁用定时器。这使看门狗定时器不会因错误代码而发生改变。因此，复位后，只有第一次 CyWdtStart() 函数调用才起作用。

每次 CTW 到达指定时间段时，看门狗都会计数。在看门狗计数到 3 之前，必须使用 CyWdtClear() 函数清除看门狗。CTW 自由运行，因此，将在 2 到 3 个定时器周期之后进行清除。

PSoC 5： 在睡眠模式中不应使用看门狗定时器。由于启用 WDT 后不能再禁用，因此可将 WDT 超时期设置为大于睡眠唤醒期，这样每次从睡眠模式唤醒时都会对看门狗进行馈送。

参数： ticks：四个可用定时器周期之一。一旦启用 WDT，就不能更改间隔。

值	定义	时间
0	CYWDT_2_TICKS	4 – 6 ms
1	CYWDT_16_TICKS	32 – 48 ms
2	CYWDT_128_TICKS	256 – 384 ms
3	CYWDT_1024_TICKS	2.048 – 3.072 s

void CyWdtStart(uint8 ticks, uint8 lpMode)（续）

参数 lpMode：低功耗模式配置。对于 PSoC 5，此参数将被忽略。WDT 始终表现得似乎已传递 CYWDT_LPMODE_NOCHANGE。

值	定义	作用
0	CYWDT_LPMODE_NOCHANGE	无变化
1	CYWDT_LPMODE_MAXINTER	在睡眠/休眠时切换到最长定时器模式
3	CYWDT_LPMODE_DISABLED	在睡眠/休眠时禁用 WDT

返回值： 无

副作用和限制 **PSoC 5：** 必须为适当的 WDT 操作启用 ILO 1 KHz WDT。停止 ILO 1 kHz 会破坏有效的 WDT 功能。

void CyWdtClear()

说明： 清除（馈送）看门狗定时器。

参数： 无

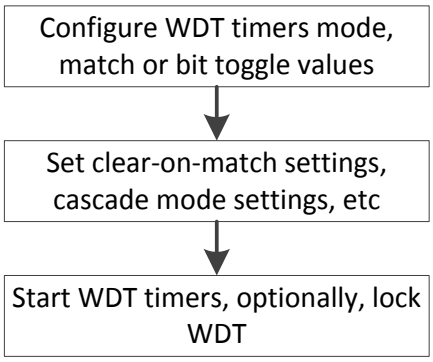
返回值： 无

PSoC 4 API

PSoC 4 有三个 WDT 定时器，在对任何看门狗配置进行更改时都应禁用这些 WDT 定时器。唯一的例外：CySysWdtWriteMatch() API（针对看门狗计时器 0 和 1）。

配置 WDT 包括在时钟域间传递控制信号，这需要启用 ILO。因此，WDT API 默认启用 ILO，将更改写入 WDT 寄存器。

下图显示了 WDT 配置步骤：



如果未配置 WDT 模式，WDT 定时器会以自由运行模式运行。

如果将 WDT 设置为生成中断，应使用相应 API 将其清除；否则持续生成中断。

void CySysWdtLock(void)

说明： 锁定看门狗定时器寄存器和 ILO 配置寄存器的配置更改。

参数： 无

返回值： 无

副作用和限制： 如果 ILO 之前为禁用状态，此 API 会启用 ILO。调用 API 后，在调用 CySysWdtUnlock() 前不能禁用 ILO。

void CySysWdtUnlock(void)

说明： 解除对看门狗定时器配置寄存器的锁定。

参数： 无

返回值： 无

副作用和限制： 如果 ILO 之前为禁用状态，此 API 会启用 ILO。

void CySysWdtWriteMode(uint32 counterNum, uint32 mode)

说明： 写入三个 WDT 计数器中其中一个的模式。

参数： counterNum：有效范围 [0-2]。WDT 计数器数量
mode：计数器运行模式：

值	定义	模式
0	CY_SYS_WDT_MODE_NONE	自由运行
1	CY_SYS_WDT_MODE_INT	对于计数器 0 和计数器 1，在匹配时生成中断，对于计数器 2，在位切换时生成中断。
2	CY_SYS_WDT_MODE_RESET	匹配时复位（仅对计数器 0 和计数器 1 有效）
3	CY_SYS_WDT_MODE_INT_RESET	在第 3 个未处理的中断生成中断并生成复位。（仅对计数器 0 和计数器 1 有效）

返回值： 无

副作用和限制： 要设置模式，应禁用 WDT 计数器的 counterNum 功能。否则此函数调用将无效。
如果 ILO 之前为禁用状态，此 API 会启用 ILO。

uint32 CySysWdtReadMode(uint32 counterNum)

说明： 读取三个 WDT 计数器中其中一个的模式。

参数： counterNum：有效范围 [0-2]。WDT 计数器的数量

返回值： 计数器的模式。和 CySysWdtWriteMode() 使用的模式参数是同样的枚举值。

副作用和限制： 无

void CySysWdtWriteClearOnMatch(uint32 counterNum, uint32 enable)

说明： 在匹配设置时，对 WDT 计数器清除进行配置。如果配置为匹配时清除，如果给计数器一个（匹配值+1）的周期，它将从 0 到匹配值计数。

参数： counterNum：有效范围 [0-1]。WDT 计数器数量。计数器 2 不支持匹配值。

启用：0 为禁用，1 为启用

返回值： 无

副作用和限制： 应禁用 WDT 计数器 counterNum。否则此函数调用将无效。
如果 ILO 之前为禁用状态，此 API 会启用 ILO。

uint32 CySysWdtReadClearOnMatch(uint32 counterNum)

说明： 为指定计数器读取匹配清除设置。

参数： counterNum：有效范围 [0-1]。WDT 计数器数量。计数器 2 不支持匹配值。

返回值： 匹配清除的状态：如果已启用，则为 1；如果已禁用，则为 0。

副作用和限制： 无

void CySysWdtEnable(uint32 counterMask)

说明： 启用指定的 WDT 计数器。启用掩码中指定的所有计数器。

参数： counterMask：要启用的所有计数器的掩码：

值	定义	计数器
1<<3	CY_SYS_WDT_COUNTER0_MASK	0
1<<11	CY_SYS_WDT_COUNTER1_MASK	1
1<<19	CY_SYS_WDT_COUNTER2_MASK	2

返回值： 无

副作用和限制： 启用或禁用 WDT 需要三个 LF 时钟周期才能生效。如果 ILO 之前为禁用状态，此 API 会启用 ILO。

void CySysWdtDisable(uint32 counterMask)

说明： 禁用指定的 WDT 计数器。禁用掩码中指定的所有计数器。

参数： counterMask：要禁用用的所有计数器的掩码：

值	定义	计数器
1<<3	CY_SYS_WDT_COUNTER0_MASK	0
1<<11	CY_SYS_WDT_COUNTER1_MASK	1
1<<19	CY_SYS_WDT_COUNTER2_MASK	2

返回值： 无

副作用和限制： 启用或禁用 WDT 需要三个 LF 时钟周期才能生效。如果 ILO 之前为禁用状态，此 API 会启用 ILO。

uint32 CySysWdtReadEnabledStatus (uint32 counterNum)

说明： 读取三个 WDT 计数器之一的启用状态。

参数： counterNum：有效范围 [0-2]。WDT 计数器数量。

返回值： WDT 计数器状态：
0－计数器已禁用，1－计数器已启用

副作用和限制： 此 API 从状态寄存器返回为实际 WDT 计数器状态。由于 WDT 计数器针对 WDT 状态寄存器已激活包含实际数据，操作可能需要 3 个 LFCLK 周期。

void CySysWdtWriteCascade(uint32 cascadeMask)

说明： 根据指定掩码值的组合写入 WDT 级联值。

参数： cascadeMask：用于设置或清除两个级联值的掩码值：

值	定义	计数器
0	CY_SYS_WDT_CASCADE_NONE	两者皆不
1<<6	CY_SYS_WDT_CASCADE_01	级联 01
1<<14	CY_SYS_WDT_CASCADE_12	级联 12

要设置两个级联模式，应对两个定义执行 OR 运算：
(CY_SYS_WDT_CASCADE_01 | CY_SYS_WDT_CASCADE_12)

返回值： 无

副作用和限制： 如果仅指定一个级联掩码，第二个级联将被禁用。要设置两个级联模式，应对两个定义执行 OR 运算：
(CY_SYS_WDT_CASCADE_01 | CY_SYS_WDT_CASCADE_12)
如果 ILO 之前为禁用状态，此 API 会启用 ILO。

uint32 CySysWdtReadCascade(void)

说明： 读取两个 WDT 级联值，返回一个位组掩码。

参数： 无

返回值： 级联值组掩码：

值	定义	级联
0	CY_SYS_WDT_CASCADE_NONE	两者皆不
1<<6	CY_SYS_WDT_CASCADE_01	级联 01
1<<14	CY_SYS_WDT_CASCADE_12	级联 12

副作用和限制： 无

void CySysWdtWriteMatch(uint32 counterNum, uint32 match)

说明： 配置 WDT 计数器匹配比较值。

参数： counterNum：有效范围 [0-1]。WDT 计数器数量。计数器 2 不支持匹配值。

匹配：有效范围 [0-65535]。用于匹配计数器的值。

返回值： 无

副作用和限制： 如果 ILO 之前为禁用状态，此 API 会启用 ILO。

void CySysWdtWriteToggleBit(uint32 bits)

说明： 配置 WDT 计数器 2 中的哪些位以检测到切换。当位切换时，如果计数器 2 的模式已启用中断，则中断生成。

参数： 位：有效范围 [0-31]。将检测切换的计数器 2 的位。

返回值： 无

副作用和限制： 应禁用 WDT 计数器 2。否则此函数调用将无效。
如果 ILO 之前为禁用状态，此 API 会启用 ILO。

uint32 CySysWdtReadToggleBit(void)

说明： 读取哪些 WDT 计数器 2 中检测的位以切换。

参数： 无

返回值： 将被检测到的位（范围为 0 到 31）。

副作用和限制： 无

uint32 CySysWdtReadMatch(uint32 counterNum)

说明： 读取 WDT 计数器匹配比较值。

参数： counterNum：有效范围 [0-1]。WDT 计数器数量。计数器 2 不支持匹配值。更改可能需要应用三个 LFCLK。

返回值： 16 位匹配值。

副作用和限制： 无

uint32 CySysWdtReadCount(uint32 counterNum)

说明： 读取当前 WDT 计数器值。

参数： counterNum：有效范围 [0-2]。WDT 计数器数量。

返回值： 实时计数器值。计数器 0 和 1 为 16 位计数器，计数器 2 为 32 位计数器。

副作用和限制： 无

uint32 CySysWdtGetInterruptSource(void)

说明： 读取一个含有所有当前设置的 WDT 中断的掩码。

参数： 无

返回值： 中断组掩码：

值	定义	计数器
1<<2	CY_SYS_WDT_COUNTER0_INT	0
1<<10	CY_SYS_WDT_COUNTER1_INT	1
1<<18	CY_SYS_WDT_COUNTER2_INT	2

副作用和限制： 无

void CySysWdtClearInterrupt(uint32 counterMask)

说明： 清除所有设置在掩码中的 WDT 计数器中断。当计数器模式设置为生成 3 个中断然后复位器件，调用此 API 也能防止复位发生。所有 WDT 中断将被固件清除，否则会连续生成中断。

参数： counterMask：要启用的所有计数器的掩码：

值	定义	计数器
1<<2	CY_SYS_WDT_COUNTER0_INT	0
1<<10	CY_SYS_WDT_COUNTER1_INT	1
1<<18	CY_SYS_WDT_COUNTER2_INT	2

返回值： 无

副作用和限制： 如果 ILO 之前为禁用状态，此 API 会启用 ILO；如果设置了看门狗锁定，暂时删除此锁定；清除了设置在掩码中的 WDT 中断后，然后恢复锁定状态。

void CySysWdtResetCounters(uint32 counterMask)

说明： 复位所有设置在掩码中的 WDT 计数器。

参数： counterMask：要复位的所有计数器的掩码：

值	定义	计数器
1<<2	CY_SYS_WDT_COUNTER0_RESET	0
1<<11	CY_SYS_WDT_COUNTER1_RESET	1
1<<19	CY_SYS_WDT_COUNTER2_RESET	2

返回值： 无

副作用和限制： 如果 ILO 之前为禁用状态，此 API 会启用 ILO。如果看门狗为锁定状态，此 API 调用不会复位计数器值。

14 MISRA 合规性

本节介绍 PSoC Creator cy_boot 组件和由 PSoC Creator 生成的代码的 MISRA-C:2004 合规性和偏差。

MISRA 代表汽车工业软件可靠性协会。MISRA 规范包含具有 122 条强制性规则和 20 条参考规则的规则集，该规则集应用于固件设计并已由汽车工业组织在一起以提高嵌入在汽车组件中的固件质量和鲁棒性。

定义了两种类型的偏差：

- 工程偏差－适用于所有 PSoC Creator 组件的偏差
- 特定偏差－适用于特定组件的偏差

本节提供以下项目的相关信息：

- 验证环境
- 项目偏差
- cy_boot 组件特定偏差

验证环境

本节提供 MISRA 合规性分析环境描述。

组件	名称	版本
测试规则	针对 C 语言在关键系统中使用的 MISRA-C:2004 准则。	2004 年 10 月
目标器件	PSoC 3	Production
	PSoC 4	Production
	PSoC 5	Production
	PSoC 5LP	Production
目标编译器	PK51	9.03
	GCC	4.4.1
	RVDS	4.1
	MDK	4.1
生成工具	PSoC Creator	2.2 SP1
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

MISRA 规则 1.5、2.4、3.3 和 5.7 未被编程研究 QA C 执行。这些规则的合规性通过代码审核手动验证。

项目偏差

项目偏差定义为允许放松应用于随 PSoC Creator 附带的源代码的 MISRA 规则要求。下表提供了偏差规则列表。

MISRA-C：2004 规则	规则类 (R/A)[1]	规则描述	偏差描述
1.1	R	此规则说明了代码必须符合 C ISO/IEC 9899:1990 标准。	有的 C 语言扩展（如中断关键词）与组件硬件功能相关并且这些扩展实际上无法避免。 在由 PSoC Creator 生成的 main.c 文件中，使用非标准 main() 声明：“void main()”。标准声明为“int main()” 宏定义数量超过 1024－项目不严格符合 ISO:C90。
5.1	R	该规则要求内部和外部标示符都不应依赖多于 31 个字符的有效性。	基于用户定义名的名称长度取决于用户定义名的长度。
5.7	A	验证没有标示符名再次使用。	具有同样名称的多局部变量可出现在不同的函数中。除常用的名称（如“i”）之外，同样组件针对多个实例生成的 API 函数可能具有相同的变量名称。
8.7	R	如果对象只能从一个单个的函数进行访问，应在模块范围内对对象进行定义。	对象“InstanceName_initVar”只由函数“InstanceName_Start”在其定义的转换单元内引用。此全局公用变量专门由用户应用程序使用。
8.10	R	除非要求外部链接，文件范围内的对象或函数的所有声明和定义应具有内部链接。	组件 API 旨在用于用户应用场合，可能不适用于组件 API。
11.3	A	该规则规定了不应将一种指针类型转换成另一种积分类型。	从无符号整数到指针的转换不会造成任何意外效果，因为它是基于对硬件寄存器结构定义的结果。
14.1	R	应无不可迭代码。	作为组件 API 一部分的一些函数不用于组件 API。组件 API 旨在用于用户应用场合，可能不适用于组件 API。
21.1	R	至少使用以下一项以保证运行时故障最小化： a) 静态分析工具/技术； b) 动态分析工具/技术； c) 处理运行时故障的供检查用的显性编码。	由于实行广义的实现方法，在一些特定的配置中的组件可以包含冗余操作介绍。

[1] 必须/参考

文档相关规则

本节提供有关 PSoC Creator 支持的工具链实现定义行为的相关信息。下表提供了偏差规则列表。

MISRA-C：2004 规则	规则类 (R/A)1	规则描述	说明
1.3	R	如果对象代码有一般定义的界面标准，而且语言/编译器/汇编器符合此标准，应只应用多个编译器和/或语言。	针对 PSoC Creator 项目，一次不能应用多个编译器和语言。 PK51 连接器产生 OMF-51 对象模块格式。GCC 连接器产生 EABI 格式文件。RVDS 和 MDK 连接器产生 ARM ELF 格式文件。
1.4	R	应检查编译器/连接器以保证 31 个字符有效性并为外部标示符支持区分大小写。	PK51 和 GCC 处理具有内部或外部标示符长度的超过 31 个字符，而且具有区分大小写功能（例如，Id 不等于 ID）。
1.5	A	规则规定浮点实现应符合定义的浮点标准。	浮点算术实现符合 IEEE-754 标准。
3.1	R	所有实现定义行为的使用应被记录。	PK51 和 GCC 编译器的文档请分别参见“帮助”菜单、“文档”子菜单、“Keil” 和“GCC”命令。
3.2	R	字符组合相应的解码应被记录。	Windows-1252 (CP-1252) 字符组编码被应用。 有些应用于 PSoC Creator 源代码生成的字符不包含在字符组中，此规定由 ISO-IEC 9899-1900“编程语言－C” 定义。
3.3	A	此规则规定整数除法的实现应被记录。	当除以有两个符号的整数时，一个符号为正而且另外一个负编译器舍入得到一个负的余数。
3.5	R	此规则需要实现定义的行为并且记录位字段的封包。	避免使用位字段。
3.6	R	应用于编码产生的库应被写入以符合此文件规定，并且已进行了适当的验证。	提供 C51、GCC 和 RVCT 的 C 标准库未经过合规性审查。有些代码应用 memset 和 memcpy。编译器可能也插入调用到供货商特定的编译器支持库中。

PSoC Creator 生成源偏差

本节提供适用于 PSoC Creator 生成代码的偏差列表。下表提供了偏差规则列表。

MISRA-C：2004 规则	规则类 (R/A)1	规则描述	偏差描述
11.4	A	该规则规定了不同对象指针类型间不可显式转换。	CYMEMZERO8 和 CYCONFIGCPY8 使用与 memset/memcpy 兼容的 void * 参数，但内部必须使用实际指针类型。

MISRA-C：2004 规则	规则类 (R/A)1	规则描述	偏差描述
14.1	R	该规则要求不应保含不可达代码。	通常使用 CYMEMZERO、CYMEMZERO8、CYCONFIGCPY、CYCONFIGCPY8、CYCONFIGCPYCODE 和 CYCONFIGCPYCODE8，但不总是使用这些存储器。
17.4	R	阵列索引应是指针运算的唯一许可形式。	CYMEMZERO8 和 CYCONFIGCPY8 具有与 memset/memcpy 兼容的 void * 参数。
19.7	A	该规则要求应使用函数，而非类函数宏。	使用 CYMEMZERO、CYMEMZERO8、CYCONFIGCPY、CYCONFIGCPY8、CYCONFIGCPYCODE 和 CYCONFIGCPYCODE8 宏以组件独立的方式调用 cymemzero、cyconfigcpy 和 cyconfigcpycode。这些宏不能转换为函数，因此不会显著提高每个函数调用所需时间和内存（仅限于 C51）。对于 GCC/RVCT，必须将宏转换为函数。

cy_boot 组件特定偏差

本节提供了 cy_boot 组件特定偏差列表。下表提供了偏差规则列表。

MISRA-C：2004 规则	规则类 (R/A)1	规则描述	偏差描述
6.3	A	在使用基本类型的地方，应用 typedef 定义的类型名指示存储大小和符号。	对于 PSoC 4/PSoC 5/PSoC 5LP，启动文件 (Cm0Start.c/Cm3Start.c) 中的 RealView C 库初始化函数 __main(void) 返回基型值“int”。
8.8	R	外部对象或函数只能在一个文件中声明。	对于 PSoC 4/PSoC 5/PSoC 5LP，在 Cm3Start.c/Cm3Start.c 文件中正在使用外部链接声明某些对象，该声明不出现在头文件中。
10.1	R	某些情况下，整型表达式的数值不应隐式转换为不同的底层类型。	PSoC 4/PSoC 5/PSoC 5LP：CMSIS 内核：在 CMSIS 内核硬件抽象层分配时，“基本无符号”型整数常数被转换为符号型常数。
10.3	R	复杂整型表达式的值只能显式转换为与表达式底层类型符号属性相同的较小的类型。	DMA API 具有复杂的“基本无符号（无符号字符）”型表达式被转换为更大的无符号类型“unsigned long”。对于 PSoC 4 cy_boot 代码，不存在这种偏差。
14.3	R	预处理前，空语句只能单独一行出现；可以在其后间隔一个空格添加注释	CYASSERT() 宏包含紧挨着其他代码的空语句。

MISRA-C：2004 规则	规则类 (R/A)1	规则描述	偏差描述
11.4	A	不同对象指针类型间不可显式转换。	DMA 和 Interrupt API 使用不同对象指针类型间的转换。
14.7	R	函数应在结尾处有单独的退出点。	CyPmSleep() 和 CyPmHibernate() 函数具有复杂的条件结构；如果组件处于非低功耗模式输入状态，为 PSoC 3/PSoC 5LP 再添加一个“返回”路径以立即返回。对于 PSoC 4 cy_boot 代码，不存在这种偏差。
17.4	R	阵列索引应是指针运算的唯一许可形式。	DMA、闪存和 Interrupt API 使用适用于指针类型对象的阵列索引以分别访问硬件寄存器、用户划分的缓冲区和矢量表。
19.4	R	C 宏只能扩展为使用一个花括号的初始化语句、常量、带括号的表达式、类型限定、存储类声明、或 do-while-zero 结构。	CYASSERT()、INTERRUPT_DISABLE_IRQ、INTERRUPT_ENABLE_IRQ、CyGlobalIntEnable 和 CyGlobalIntDisable 宏定义了使用一个花括号的代码语句块。
19.7	A	函数应优先于类函数宏使用。	由于使用了类函数以提高代码生成效率，出现了偏差。

15 cy_boot 组件更改

3.40 版

本节列出并说明了 3.40 版 cy_boot 组件的主要更改：

3.40 版的更改说明	更改原因/影响
添加了 PSoC 4器件支持。	新器件支持。
PSoC 3：更新了 CyPmSleep() 函数的描述，即新增了“进入睡眠功耗模式前，应禁用硬件蜂音器”。 由于对于 LVI 和 HVI，需要硬件蜂音器，以及进行掉电检测操作－进入睡眠功耗模式及前必须禁用，唤醒后必须恢复。如果 LVI 或 HVI 启用，且调试模式中对项目进行了编译，CyPmSleep() 将使器件停止。	使用硬件蜂音器和其他组件唤醒源可能导致器件锁定，进而使代码执行停止。有关详细信息，请参见器件勘误表。

3.30 版

本节列出并说明了 3.30 版 cy_boot 组件的主要更改：

3.30 版的更改说明	更改原因/影响
更新后可支持 PSoC Creator 2.2。	
已添加 MISRA 合规性章节。	
已添加 低电压模拟升压时钟 章节。	基于 SC（TIA、Mixer、PGA 和 PGA_Inv）的组件的新功能。
已添加中断配置相关要求（前提条件是此中断用作唤醒事件，且中断源为 PICU ）	对于 PSoC 5LP，连接到唤醒源的中断组件不能使用“RISING_EDGE”检查选项。使用“电平”选项代替。
总线时钟和模拟时钟保存/恢复之间的延迟从 CyPmSleep() 和 CyPmHibernate() 函数移至 CyPmSaveClocks() / CyPmRestoreClocks()。	这种修改减少了 CyPmSleep() 和 CyPmHibernate() 函数的执行时间。 CyPmSaveClocks() 函数执行后必须使用采用模拟时钟的组件，直到可通过 CyPmRestoreClocks() 函数恢复时钟配置。
已添加 float32 和 float64 数据类型。对于 PSoC 3 组件，float64 数据类型不可用。	

3.20 版

本节列出并说明了 3.20 版 cy_boot 组件的主要更改：

3.20 版的更改说明	更改原因/影响
对本文件进行了小的改动，以区分 PSoC 5 和 PSoC 5LP器件的功能。	完善 PSoC 5 和 PSoC 5LP 文档。
PSoC 5LP “备用活动”使用模型更改为与 PSoC 5 “备用活动”使用模型相同。	针对 CyPmAltAct()，未使用任何参数。这意味着应针对参数传递 NONE。组件将进入“备用活动”模式，直到启用中断。
针对 PSoC 5LP，更新了 CyIMO_SetFreq() 函数接口，因而可支持 62 和 72 MHz 频率。	新增了接口，可在 PSoC 5LP 上配置 62 和 72 MHz IMO。

3.10 版

本节列出并说明了 3.10 版 cy_boot 组件的主要更改：

3.10 版的更改说明	更改原因/影响
3.0 版 cy_boot 组件中，重新设计了引导加载程序，以将引导加载程序和可引导加载项目组件分开。本文列举了一些更改，供从旧版本迁移时参考。	有关详细信息，请参见第 引导加载程序移植。70 页的
对“电压检测 API”也进行了一些小的改动，例如，修复了寄存器定义的拼写错误；添加了 CyVdLvDigitEnable() 函数阈值参数掩码以防止无效参数值；更新了 CyVdLvDigitEnable() 和 CyVdLvAnalogEnable() 函数以在硬件初始化期间使用延迟而非“while”循环。	提高这些 API 的整体实现。
对 CyPmSleep() 函数进行了少量更新。	提高最新的 PSoC 3器件的支持性能。

3.0 版

本节列出并说明了 3.0 版 cy_boot 组件的主要更改：

3.0 版的更改说明	更改原因/影响
重新设计了引导加载程序，以将引导加载程序和可引导加载项目组件分开。	有关详细信息，请参见第 引导加载程序移植。70 页的
更新了 CyPmSleep() 函数实现，以在睡眠模式前/后保存/恢复 PRES 状态。新增了 HVI/LVI 支持功能。	支持新功能。
新增了以下“电压检测 API”：CyVdLvDigitEnable()、CyVdLvAnalogEnable()、CyVdLvDigitDisable()、CyVdLvAnalogDisable()、CyVdHvAnalogEnable()、CyVdHvAnalogDisable()、CyVdStickyStatus() 和 CyVdRealTimeStatus()。	新增“电压监控 API”。
由于闪存 API 中使用的 SPC API 进行了代码重构，对闪存 API 的实现稍微进行了修改。	改进实现质量。

3.0 版的更改说明	更改原因/影响
更新了 CyXTAL_32KHZ_Start()、CyXTAL_32KHZ_Stop()、CyXTAL_32KHZ_ReadStatus() 和 CyXTAL_32KHZ_SetPowerMode() APIs 的实现。	已添加额外超时设定，以确保模块正常启动。
针对 PSoC 5 器件，更改了 CyXTAL_Start() 函数的实现。有关函数的更多信息，请参考“时钟”一节。	进行了一些更改，以确保在 PSoC 5 部件上成功启动 MHZ XTAL。
对于 PSoC 5器件，删除了以下 API：CyXTAL_ReadStatus()、CyXTAL_EnableErrStatus()、CyXTAL_DisableErrStatus()、CyXTAL_EnableFaultRecovery()、CyXTAL_DisableFaultRecovery()。	PSoC 5器件不支持 API 提供的功能。
如今，只有当 DMA 组件被置于设计原理图上时通过启动代码调用 CyDmacConfigure() 函数。	如果未设计 DMA 组件，增加组件的启动时间。如果在没有 DMA 组件的情况下使用 DMA 功能，应手动调用 CyDmacConfigure() 函数。
修改了 CyXTAL_32KHZ_ReadStatus() 函数的实现，即，删除了数字测量状态返回。	模拟状态测量是唯一的可靠源。
更新了下列 API 的说明：CyFlash_SetWaitCycles()。	已进行更改，以优化功耗模式配置。
对于 PSoC 3，可重入堆栈栈顶地址从 CYDEV_SRAM_SIZE 渐减为 (CYDEV_SRAM_SIZE - 3)。	防止在可重入函数执行期间使用该可重入函数的逻辑与/或本地变量参数重写 CyResetStatus 变量。
更新了 CyIMO_SetFreq() 函数的实现，即，对于 PSoC 5器件，删除了 74 和 62 MHz 支持。	删除器件不支持的功能。
CyPLL_OUT_SetPQ() 的最低 P 分频器值从 4 提高为 8。	满足硬件要求
针对 PSoC 5LP 器件，添加了 CyXTAL_SetFbVoltage()/SetWdVoltage()。	对于 PSoC 5LP，PI 提供的功能不可用。
更新了 CyWdtStart() 的描述。	PSoC 5 低功耗模式下，添加了关于 WDT 操作的注释。
对于 PSoC 5，CyPmSleep() 的实现更改为“唤醒时不将 CTW 保持在复位状态”。	“唤醒时不将 CTW 保持在复位状态”允许 PSoC 5 在“活动”和低功耗模式下时皆可使用 CTW。
*复位状态的保留*一节进行了详细更新。	对软件重置行为进行了阐释。阐述了如何使用复位状态变量。
更新了下列 API 的说明：CyMasterClk_SetDivider()、CyWdtStart()、CyWdtStart()。	更好地反映实现。
更新了“启动”和“链接”章节。新增了有关自定义连接器脚本使用的信息。	为组件操作提供更详细的信息。
删除了下列宏：CYWDT_TICKS CYWDT_CLEAR、CYWDT_ENABLE、CYWDT_DISABLE_AUTO_FEED。	应使用 CyWdtStart() 和 CyWdtClear() 代替。

3.0 版的更改说明	更改原因/影响
对于 PSoC 5 器件，删除了 CyCpuClk_SetDivider()。	硬件不支持该功能。
删除了 cystrcpy()、cystrlen()、CyGetSwapReg16() 和 CySetSwapReg16() API。	应使用库函数。
针对 PSoC 5，更新了 CyEnterCriticalSection() 函数的返回值描述。	如果之前已启用中断，则函数返回 0；如果之前已禁用中断，则函数返回 1。
向.cyre 文件中包括的所有 API 添加了 CYREENTRANT 关键词。	并非所有 API 都是真正可重入的。组件 API 源文件中的注释指出了适用的函数。 需要此更改为采用安全方式使用并且不是可重入函数消除编译器警告：通过标志或关键节防止并发调用。
添加了 PSoC 5LP 支持	

2.40 版和较早版本

2.40 版

本节列出并说明了 2.40 版 cy_boot 组件的主要更改：

2.40 版的更改说明	更改原因/影响
更新了 CyPmSleep() 和 CyPmHibernate() API。	已进行更改，以优化功耗模式配置。

2.30 版

本节列出并说明了 2.30 版 cy_boot 组件的主要更改：

2.30 版的更改说明	更改原因/影响
CyIntEnable 和 CyIntDisable 函数已默认更改为 CYREENTRANT。	许多组件要求可重新进入 CyIntEnable 和 CyIntDisable，而这些组件无法实现这一点。这意味着您不再需要为您不会调用的这些函数填充 cyre 文件。
通过在进入组件低功耗模式之前移除 32 KHz ECO、100 KHz 和 1 KHz ILO 功耗模式配置，修改了 CyPmSleep() 和 CyPmAltActive() 函数的实现。	在“睡眠”和“备用活动”模式中，使用户负责时钟功耗模式配置。 CyILO_SetPowerMode() 和 CyXTAL_32KHZ_SetPowerMode() 可用于配置时钟功耗模式。 有关用户对时钟功耗模式配置的责任的信息已添加到 PM API 一节中。
已更新 CyPmSaveClocks() 的实现，以在启用“Enable Fast IMO during startup”（在启动过程中启用快速 IMO）时将 IMO 时钟频率设为 48 MHz，否则设为 12 MHz。在进入低功耗模式之前，IMO 频率始终设为 12 MHz，并在唤醒之后立即恢复。CyPmRestoreClocks() 恢复 IMO 时钟的原始值。	IMO 值应与 FIMO 匹配，且 FIMO 始终为 12 MHz。

2.30 版的更改说明	更改原因/影响
通过移除对 MHz ECO 和 PLL 禁用状态的恢复，更新了 CyPmRestoreClocks() 函数的实现。	应仅在调用 CyPmSaveClocks() 函数之后才调用 CyPmRestoreClocks() 函数，后一个函数始终禁用 MHz ECO 和 PLL。
全局中断在 CyPmSleep()/CyPmHibernate() 入口上被禁用，并在从函数中返回之前恢复。	禁用中断以在恢复组件状态之前出现中断。
对于 PSoC 5 组件，更新了 CyPmSleep() 和 CyPmAltAct() 函数实现，以忽略所有参数。PSoC 5 器件将进入睡眠模式，直至被三个中断源之一中的中断唤醒。CTW、每秒一次、或端口中断控制器 (PICU)。必须已配置唤醒源以生成中断。使用 Sleep Timer 组件配置了 CTW，并使用实时时钟组件配置了每秒一次中断。	必须仅通过以下参数使用 CyPmSleep() 和 CyPmAltAct() 函数：CyPmSleep(PM_SLEEP_TIME_NONE、PM_SLEEP_SRC_NONE) 和 CyPmAltAct(PM_ALT_ACT_TIME_NONE、PM_ALT_ACT_SRC_NONE)。
更新了结构特定和芯片特定的 #defines，以在整个内容中使用。	
提高了 8051 组件上非 DMA 配置的性能。	这些修改减少了启动时间，并稍微减少了代码存储器和内部数据存储器的消耗。
已针对 PSoC 5 芯片更新了 CyPmRestoreClocks() 的实现。提供 130 ms 以便兆赫兹晶振稳定下来。保持关闭超时后不验证其是否就绪。	这些修改增加了晶振启动时间，但确保晶振准备就绪。
对于作为唤醒定时器的定时器，已从 PM API 函数中移除其源时钟的功耗模式。	调用 PM API 函数之前，必须针对用作唤醒定时器的定时器手动配置源时钟的功耗模式。
更新了“PSoC Creator 电源管理”一节。	添加了更多有关电源管理 API 用途的详细信息。

2.21 版

本节列出并说明了 2.21 版 cy_boot 组件的主要更改：

2.21 版的更改说明	更改原因/影响
提供了一个新选项，用于选择如何计算从引导加载程序主机传输到引导加载程序的数据的校验和。	提供了一个更有效地在 I/O 传输过程中检查错误的方法。
提供了一个通用选项，允许用户定义其自己的自定义引导加载程序通信函数。	添加了对其他通信协议（SPI、UART、...）的支持 更简单。还提供了在同一设计中支持多个并发通信组件的方法。
更新了几个电源管理函数，以避免出现一些可能的问题。	一些括号丢失了，可能导致以错误的顺序评估项目。
添加了变量 CyResetStatus，可用于从 RESET_SR0 寄存器中获取信息。	提供该变量的原因是，RESET_SR0 寄存器中包含的许多字段处于清除读取模式中。由于引导加载程序在操作中需要访问此寄存器，因此它阻止了实际的应用代码访问值。通过使用此变量，应用仍可访问所有信息。
针对一些 PSoC3 器件添加了解决方法，以确保已正确初始化 NVL 值。	在一些 PSoC 3 组件上，NVL 信息可能未正确初始化。此解决方法用于确保在执行任何启动代码之前正确加载了 NVL。

2.20 版

本节列出并说明了 2.20 版 cy_boot 组件的主要更改：

2.20 版的更改说明	更改原因/影响
更新了 CyDelayCycles 函数，以在指令缓存启用的情况下运行。	原 CyDelayCycles 函数设计用于指令缓存禁用的情况下。对于 PSoC 3 ES3 和 Production 版本的芯片，启用指令缓存时，延迟时间不再正确。
更新了本指南中有关低功耗模式函数的代码注释和说明。	我们阐述的 CyPmSleep()、CyPmHibernate() 和 CyPmAltAct() 函数的注释和说明参考了有关 PSoC 3 ES2 和 PSoC 5 缺陷的芯片勘误表。
修复了一个与 CyBtldr_ValidateApp 函数有关的引导加载程序问题。	修复了 PSoC 5 引导加载程序中的一个问题，该问题可能会导致在应用代码大于 64 K 的情况下无法对验证应用代码。
解决了引导加载程序的通信等待问题。	修改了使引导加载程序认为存在活动和人员似乎在尝试通信所需的因素。 之前，如果引导加载程序通过所选通信组件接收到任何数据后，程序将处于永远等待数据的状态。执行了这项更改后，现在只有当接收到“进入引导加载程序”命令时，引导加载程序才会进入永久等待状态。
更新了 CySetTemp 函数，以读取模具温度两次，确保其有稳定的值。	已进行更改，以解决一个问题。不会产生影响。

2.10 版

本节列出并说明了 2.10 版 cy_boot 组件的主要更改：

2.10 版的更改说明	更改原因/影响
更新了引导加载程序中的闪存验证码	修复了引导加载程序验证子程序中的一个问题，在闪存禁用的情况下，该问题可能会导致对有效图像进行报错的情况。

2.0 版

本节列出并说明了 2.0 版 cy_boot 组件的主要更改：

2.0 版的更改说明	更改原因/影响
Keil C51 关键字可用作宏	这些宏使代码在与其他编译器保持兼容的同时，能够指定变量和指针的存储空间。最常用的关键字包括 CYCODE、CYXDATA 和 CYDATA，分别表示 C51 的代码、外部数据和数据关键字。在其他编译器上，这些宏可忽略。
CyLib API 不再具有条件性	不再需要定义 CYLIB_POWER_MANAGEMENT 等宏来将 API 函数包含到 CyLib 中。
在 RealView 连接器脚本中添加了有关 .dma_init 的单独章节。	在基于 DMA 的配置启用的情况下，在 RealView 中构建项目时，这可以防止潜在的错误和警告发生。
初始化了完整的 SRAM 中断矢量表	在以前的版本中，只初始化了表中的前 32 条。
将保留的中断矢量初始化为默认的中断处理程序	在以前的版本中，保留矢量的初值为 0，并不是有效的中断服务子程序地址。
修复了 PSoC 3 上 CyIntSetVector 返回值中错误的字节序	在以前的版本中，高字节与低字节互换。

2.0 版的更改说明	更改原因/影响
将中断属性应用到中断服务子程序声明中	通过 CY_ISR/CY_ISR_PROTO/cyisraddress 声明的中断服务子程序现在具有中断属性，使编译器发出调整堆栈以使其对齐的代码。RealView 将中断属性视为类型的一部分来进行检查，因此，没有中断属性的中断服务子程序或矢量将无法在 RealView 上进行编译。因此，在使用 RealView 时，cy_isr_v1_20 及更早的子程序或矢量与 cy_boot_v1_50 不兼容。这只会影响 PSoC 5；PSoC 3 版本中已存在中断属性。
使 CY_GET_REGxx/CY_SET_REGxx 重新进入并提高性能	已用汇编子程序替代了实现此功能的函数，即 CyGetRegXX 和 CySetRegXX。这些新的子程序可安全地用于中断服务子程序。
PSoC 3 ES3 支持 CyDelay	CPUCLK_DIV 已从 SFR 移至 ES3 中的 CLKDIST_MSTR1。
用 *limits.h* 和 *ctype.h* 替换了 *CyLib.h* 中的某些宏	LONG_MIN、LONG_MAX 和 ULONG_MAX 这三个宏现由特定于工具链的 limits.h 提供了。这可防止因编译器的 limits.h 和 CyLib.h 之间存在的差异而引起的警告。
支持 RealView 的 --gnu 模式	修正了某些会在使用 RealView 的 --gnu 选项时导致错误的预处理器条件判断。
更新了 PSoC 5 启动代码，以更好地利用标准库	PSoC 5 项目中使用的启动代码已更新，以使用标准库进行初始化。这样，就提供了标准初始化。
用 cydevice_trm.h 替换了 cydevice.h	cydevice.h 已标记为过期，因此，PSoC Creator 中的 API 和生成代码不得使用 cydevice.h。
CYCODE、CYDATA、CYXDATA 和 CYFAR 的定义已复制到 cytypes.h 中	现在，除闪存组件外，其他所有组件均可使用这些定义
修复了睡眠后缓存闪存周期的错误运行状态。	睡眠后，CYREG_CACHE_CR 不会将值保留在 PSOC 3 中
更改了多种闪存/EEPROM API。	定义了基于芯片的有效/待机功耗配置寄存器。 定义了基于芯片的有效/待机功耗配置寄存器常量。 添加了 CyFlash_Start() 和 CyFlash_Stop() API。 删除了 CyFlashEEActivePower() 和 CyFlashEEStandbyPower() API， 修改了 PSoC 3 的 CySetFlashEEBuffer() 函数，从而不对缓冲区指针是否为 0 进行测试。 将 CyWriteRowConfig() API 中的一个参数更改为 rowData。 添加了 CyEEPROM_Start()、CyEEPROM_Stop()、CyEEPROM_ReadReserve()、CyEEPROM_ReadRelease() 和 CyFlash_SetWaitCycles(uint8 freq) API。 定义了基于芯片的功耗模式寄存器和功耗模式寄存器常量。 定义了基于芯片的缓存控制寄存器。
添加了重新进入支持。	更新了 cylib.c/h、cyflash.c/h、cydmac.c/h 和 cyspc.c/h 文件中相应的 API，以支持重新进入。

2.0 版的更改说明	更改原因/影响
cymemset 和 cymemcpy 已转换为宏，删除了 cymemmove。	由于工具链供货商提供的 memset/memcpy/memmove 设计用于特定的目标平台，所以通常快于一般的实现。同时包含供货商提供的函数和一般的实现会浪费代码空间。非遗留代码应直接使用 memset 和 memcpy。
添加了许多新的时钟 API： CyPLL_OUT_Start_confirm CyMasterClock_SetSource CyMasterClock_ActivateIMOAndSet CyMasterClock_ActivatePLLAndSet CyIMO_ActivateAndWait CyIMO_SetFrequency CyIMO_EnableDisableDoubler CyIMO_EnableDoubler CyIMO_DisableDoubler CyIMO2X_SetSource CyPLL_P_SetValue CyPLL_Q_SetValue CyPLL_SetInput CyXTAL_SetGain CyXTAL_SetVref CyILO_1KHZ_Start CyILO_1KHZ_Stop CyILO_100KHZ_Start CyILO_100KHZ_Stop CyILO_33KHZ_Start CyILO_33KHZ_Stop CyILO_SelectClock CyUSB_Start CyUSB_Stop CyUSB_SetClockSource CyUSB_SetClockSource_IMO2X CyUSB_SetClockSource_IMO CyUSB_SetClockSource_PLL CyUSB_SetClockSource_DSI CyUSB_EnableDivider CyUSB_DisableDivider CyCLK_SetDividerValue CyCLK_SetBusDividerValue	用于提供附加功能。
char8 数据类型定义为字符型。	用于支持未来的新编译器。
将 CySleep 和 CyHibernate API 重新命名为 CyPmSleep 和 CyPmHibernate；添加了新的 CyPmAltAct API。	在 cy_boot 2.0 之前，低功耗 API 中存在缺陷。您必须更新您的设计以使用 cy_boot 2.0，并重新编写设计中的低功耗部分以使用新的 API。

此页特意留白。