PSoC® Creator™

System Reference Guide

cy_boot 组件 v4.0

Document Number: 001-89756, Rev. *A

赛普拉斯半导体
198 Champion Court
San Jose，CA 95134-1709
电话（美国）：800.858.1810
电话（国际）：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，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。

目录

1 简介 6
文档规范 7
参考 7
样例固件源代码 7
修订记录 7

2 标准类型和定义 8
基本类型 8
硬件寄存器类型 8
编译器定义 8
Keil 8051 兼容性定义 9
返回代码 9
中断类型和宏 10
内部定义 10
器件版本定义 10
变量属性 10
项目类型 11
芯片配置模式 11
调试模式 12
芯片保护模式 12
栈和堆 12
电压设置 12
系统时钟频率 13
JTAG/芯片 ID 13

3 时钟 14
PSoC Creator 时钟实现 14
API 23

4 电源管理 35
PSoC 3/PSoC 5LP 实现 35
PSoC 4 实现 43
实例低功耗 API 46

5 中断 48
API 48
6 引脚 52
PSoC 3/PSoC 5LP API 52
PSoC 4 API 54
7 寄存器访问 56
API 56
8 DMA 60
9 闪存和 EEPROM 61
PSoC 3/PSoC 5LP 实现 61
PSoC 4 实现 70
10 Bootloader 移植 72
简介 72
移植 bootloader 设计 73
移植 Bootloadable 设计 73
项目应用类型属性更改 74
移植 Bootloadable 依赖关系至 bootloader 设计 74
11 系统函数 75
通用 API 75
CyDelay API 76
PSoC 3/PSoC 5LP 电压检测 API 77
PSoC 4 电压检测 API 81
缓存功能 82
12 启动和连接 83
PSoC 3 83
PSoC 4/PSoC 5LP 84
复位状态保存（PSoC 3/PSoC 5/PSoC 5LP） 86
13 看门狗定时器 87
PSoC 3/PSoC 5LP API 87
PSoC 4 API 88
14 MISRA 合规性 94
验证环境 99
项目偏差 95
文档相关规则 95
PSoC Creator 生成源偏差 96
cy_boot 组件特定偏差 98

15 cy_boot 组件更改 100
版本 4.0 100
3.40 版 100
3.30 版 101
3.20 版 101
3.10 版 101
3.0 版 102
2.40 版和更早的版本 103

1 简介

本《系统参考指南》将介绍 PSoC Creator cy_boot 组件所提供的功能。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。虽然这些功能并非来自组件库，但也可以供其使用。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：

- 自动添加到每个项目
- 只能显示单个示例
- 无符号表示
- 不显示在组件目录中（默认情况下）

作为系统组件，cy_boot 包含了多项库功能。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
- 多种库功能
- 链接器脚本

通过该 cy_boot 组件所提供的 API，用户固件可完成本指南中所述的任务。下文对多种主要功能分别进行了介绍。

文档规范

下表列出了本指南使用的规范：

规范	用途
Courier New字体	显示文件位置和源代码： C:\ …cd\icc\，用户输入文本
斜体字	显示文件名和参考文档： *sourcefile.hex*
[方括号、粗体]	显示程序中的键盘指令： **[Enter]**或**[Ctrl]** **[C]**
File > New Project	表示菜单路径： **File > New Project > Clone**
黑体字	显示程序中的指令、菜单路径和选项以及图标名称： 单击**Debugger**图标，然后单击**Next**。
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 器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 组件创建指南
- PSoC 技术参考手册（TRM）

样例固件源代码

在“Find Example Project”对话框中，PSoC Creator 提供了大量的示例项目，包括原理图和代码的。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要查看通用示例，请打开 Start Page 或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多有关信息，请参考《PSoC Creator 帮助》中的“Find Example Project”（查找示例项目）主题。

修订记录

文档标题：PSoC® Creator™ System Reference Guide，cy_boot组件v4.0 **文档编号：001-87820**		
修订版	**日期**	**更改说明**
**	8/19/13	4.0版cy_boot组件的新文档。有关cy_boot的各版本的组件变更内容，请参考变更一节。
*A	10/25/16	Minor updates.

2 标准类型和定义

为了支持在不同平台上（不同的 CPU 和编译器）使用相同的代码，cy_boot 组件提供了可以在多个平台之间等效使用的类型和定义。

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其他影响，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 编译器支持特定于此平台的类型修饰符。在其他平台上，不能使用这些修饰符。为实现兼容性，对 Keil 进行编译时，可通过映射至适当字符串的定义来支持这些类型；而在对其他平台进行编译时，则通过映射到空字符串的定义来支持它们。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是一个 8 位无符号类型：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	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	无更多的可用对象
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)
{
      /* ISR Code here */
}
```

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

器件版本定义

定义	说明
CY_PSOC3	任何PSoC 3器件
CY_PSOC5	任何PSoC 5器件
CY_PSOC5A	PSoC 5器件
CY_PSOC5LP	PSoC 5LP器件
CY_PSOC4	PSoC 4器件

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。

项目类型

项目类型的定义如下（依次选择 **Project > Build Settings**）：

- CYDEV_PROJ_TYPE
- CYDEV_PROJ_TYPE_BOOTLOADER
- CYDEV_PROJ_TYPE_LOADABLE
- CYDEV_PROJ_TYPE_MULTIAPPBOOTLOADER
- CYDEV_PROJ_TYPE_STANDARD

芯片配置模式

芯片配置模式的定义如下所示（通过系统 DWR）。各选项可根据不同的器件而改变：

所有器件

- CYDEV_CONFIGURATION_MODE
- CYDEV_CONFIGURATION_MODE_COMPRESSED
- CYDEV_CONFIGURATION_MODE_DMA
- CYDEV_CONFIGURATION_MODE_UNCOMPRESSED
- CYDEV_DEBUGGING_ENABLE 或 CYDEV_PROTECTION_ENABLE（使能调试或保护功能。这两个功能是互斥的。）

PSoC 3

- CYDEV_CONFIGURATION_CLEAR_SRAM（启动代码是否清除 SRAM?）

PSoC 3 和 PSoC 5LP

- CYDEV_CONFIGURATION_COMPRESSED（配置数据是否压缩?）
- CYDEV_CONFIGURATION_DMA（配置数据是否通过 DMA 加载?）
- CYDEV_CONFIGURATION_ECC（配置数据是否被存储在 ECC 内?）
- CYDEV_CONFIG_FASTBOOT_ENABLED（启动时，器件的频率是否为 48 MHz?如果不是 48 MHz，那它就是 12 MHz。）
- CYDEV_INSTRUCT_CACHE_ENABLED（是否使能了指令缓存?）
- CYDEV_DMA_CHANNELS_AVAILABLE（可用于配置的 DMA 通道数量。）
- CYDEV_ECC_ENABLE（是否使能了 ECC?）
- CYDEV_DEBUGGING_XRES（可选的 XRES 引脚是否作为 XRES 引脚被使能?）

PSoC 4

- CYDEV_CONFIG_READ_ACCELERATOR（是否使能了闪存读取加速器?）

PSoC 4 和 PSoC 5LP

- CYDEV_USE_BUNDLED_CMSIS（包括 CMSIS 标准库。）

调试模式

调试模式的定义如下（通过系统 DWR）：

- CYDEV_DEBUGGING_DPS
- CYDEV_DEBUGGING_DPS_Disable
- CYDEV_DEBUGGING_DPS_JTAG_4
- CYDEV_DEBUGGING_DPS_JTAG_5
- CYDEV_DEBUGGING_DPS_SWD
- CYDEV_DEBUGGING_DPS_SWD_SWV

芯片保护模式

芯片保护模式的定义如下所示（通过系统 DWR）：

- CYDEV_DEBUG_PROTECT
- CYDEV_DEBUG_PROTECT_KILL
- CYDEV_DEBUG_PROTECT_OPEN
- CYDEV_DEBUG_PROTECT_PROTECTED

栈和堆

为栈和堆分配的字节数量的定义如下（通过系统 DWR）。这些定义仅适用于 PSoC 4 和 PSoC 5LP。

- CYDEV_HEAP_SIZE
- CYDEV_STACK_SIZE

电压设置

电压设置的定义如下（通过系统 DWR）。各选项可根据不同的器件而改变：

- CYDEV_VARIABLE_VDDA
- CYDEV_VDDA
- CYDEV_VDDA_MV
- CYDEV_VDDD
- CYDEV_VDDD_MV
- CYDEV_VDDIO0
- CYDEV_VDDIO0_MV
- CYDEV_VDDIO1
- CYDEV_VDDIO1_MV

- CYDEV_VDDIO2
- CYDEV_VDDIO2_MV
- CYDEV_VDDIO3
- CYDEV_VDDIO3_MV
- CYDEV_VIO0
- CYDEV_VIO0_MV
- CYDEV_VIO1
- CYDEV_VIO1_MV
- CYDEV_VIO2
- CYDEV_VIO2_MV
- CYDEV_VIO3
- CYDEV_VIO3_MV

系统时钟频率

系统时钟频率的定义如下（通过时钟 DWR）：

PSoC 3 和 PSoC 5

- BCLK__BUS_CLK__HZ
- BCLK__BUS_CLK__KHZ
- BCLK__BUS_CLK__MHZ

PSoC 4

- CYDEV_BCLK__HFCLK__HZ
- CYDEV_BCLK__HFCLK__KHZ
- CYDEV_BCLK__HFCLK__MHZ
- CYDEV_BCLK__SYSCLK__HZ
- CYDEV_BCLK__SYSCLK__KHZ
- CYDEV_BCLK__SYSCLK__MHZ

JTAG/芯片 ID

当前器件的 JTAG/芯片 ID 的定义如下：

- CYDEV_CHIP_JTAG_ID

3 时钟

PSoC Creator 时钟实现

PSoC Creator 所支持的 PSoC 器件具有灵活的时钟功能。这些时钟功能在 PSoC Creator 中进行控制，具体控制因素有：设计范围资源设置中的选项、设计原理图上时钟信号的连接性以及可修改运行期间时钟的 API 调用。

本部分介绍了 PSoC Creator 将时钟映射到器件的方法，并提供了针对 PSoC 结构进行优化的计时方法相关指导。

PSoC 3/PSoC 5LP 器件的时钟架构与 PSoC 4 器件的时钟架构不同。系统时钟将 PSoC 3/PSoC 5LP 器件上的总线时钟（BUS_CLK）以及 PSoC4 器件上的系统时钟（SYSCLK）整合在一起。主器件时钟将 PSoC 3/PSoC 5LP 器件上的主器件时钟（MASTER_CLK）以及 PSoC 4 器件上的 HFCLK 整合在一起。

时钟连接性

PSoC 架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》*中介绍的内容。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。它与主器件时钟密切相关。在大多数设计中，主器件时钟和系统时钟频率相同，且被视为同一个时钟。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。某时钟得到同步时，就意味着该时钟与主器件时钟同步。某引脚得到同步时，就意味着该引脚与系统时钟同步。

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，该时钟将作为全局时钟。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。它与系统时钟和主器件时钟相关。PSoC 的设计是一个同步系统。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。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 5LP）

同步时钟示例：

- 已设置“与主器件时钟同步”选项的全局时钟。该选项为默认设置，位于 Clock component Configure（时钟组件配置对话框的 **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 Pins component Configure（引脚组件配置）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 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：
  - 在作为时钟使用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 5LP）

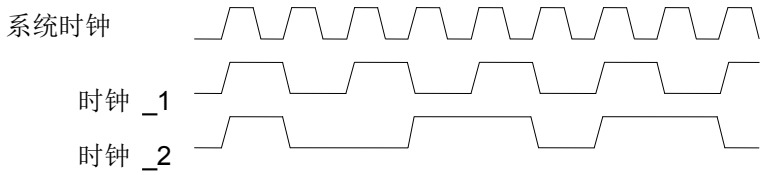
根据时钟信号源的不同，可使用以下不同方法同步信号：

- 对于异步全局时钟，可通过选中 Clock component Configure（时钟组件配置）对话框中的 **Sync with MASTER_CLK** 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** 选项卡下 Pins component Configure（引脚组件配置）对话框中的 **Input Synchronized** 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作为时钟信号，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。



路由时钟实现

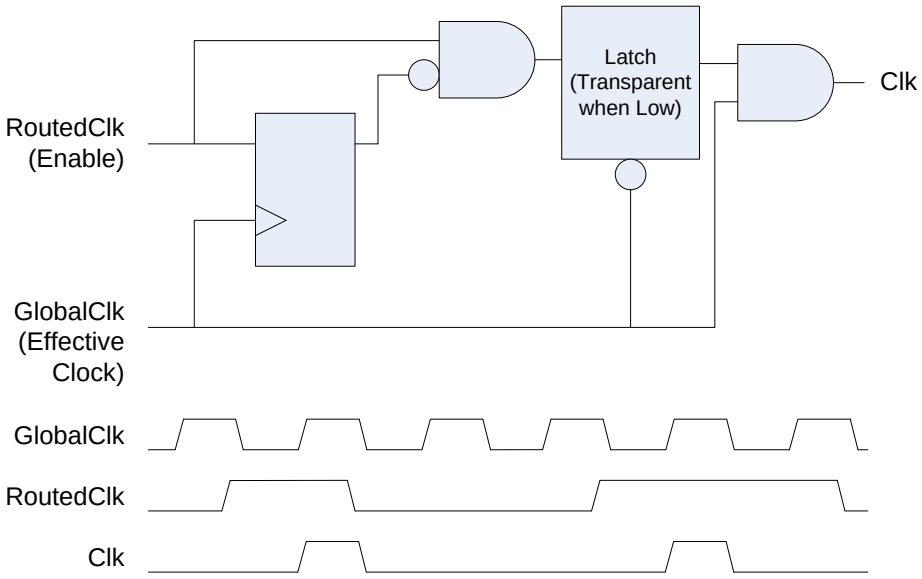
PSoC 中的时钟实现会将全局时钟信号直接连接至计时数字逻辑的时钟输入。这对于同步时钟和异步时钟均适用。因为全局时钟分布在低时滞时钟线上，所以连接至相同全局时钟的计时元素都将在相同的时间计时。

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作为时钟使用，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 中的时钟实现包括了一个在此转换中使用的内置的沿检测电路。这不会使用 PLD 资源进行实现。下图所示为逻辑实现和产生的时钟时序图。



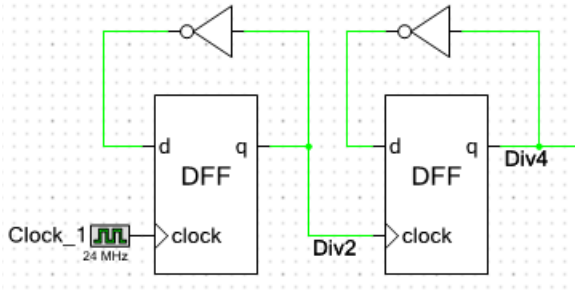
此图显示，生成的时钟在路由时钟的上升沿后于第一时钟上的全局时钟同步发生。

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 **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）”是检测到沿并作为时钟使能的路由时钟。

以分频时钟为例

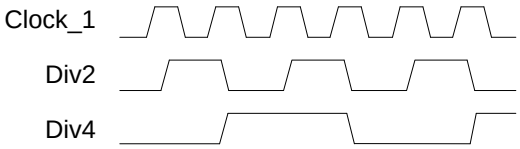
简单的分频时钟电路可用于观察该转换的完成方法。下面的电路使用全局时钟对第一个触发器（cydff_1）进行计时。这将生成一个二分频时钟。该信号作为路由时钟使用，对下一个触发器（cydff_2）进行计时。



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>
```

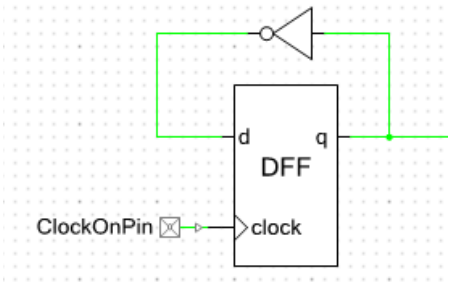
此电路生成的结果信号如下。



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处引入。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作为有效时钟使用，并将引脚的上升沿作为使能信号使用。



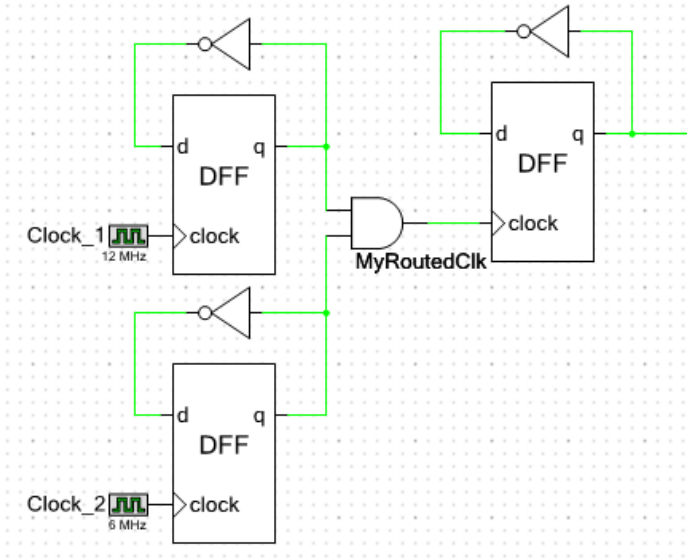

```
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
(Global Clock Selection)
<CYPRESSTAG name="UDB Routed Clock Assignment">
    Routed Clock: ClockOnPin(0):iocell.fb
        Effective Clock: BUS_CLK
        Enable Signal: ClockOnPin(0):iocell.fb
</CYPRESSTAG>
```

如果没有在引脚处使能输入同步，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。

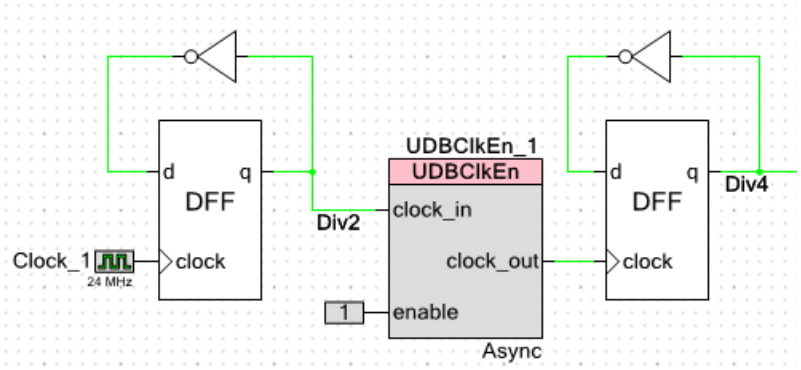


```
<CYPRESSTAG name="Tech mapping">
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
<CYPRESSTAG name="Global Clock Selection" icon="FILE_RPT_TECHM">
    Digital Clock 0: Automatic-assigning  clock 'Clock_1'. Fanout=1, Signal=tmp__cydff_1_clk
    Digital Clock 1: Automatic-assigning  clock '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

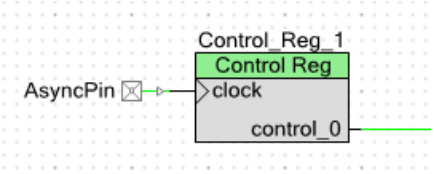
PSoC Creator 在路由时钟上执行的自动转换通常是应该使用的实现。但是，也有可强制直接使用路由时钟的方法。在异步模式中配置的 UDBClkEn 组件会将使用的时钟强制变为路由时钟，如下方电路所示。



使用异步时钟

异步时钟可以与 PLD 逻辑一起使用。但是，因为元件 CPU 之间的交互，控制寄存器、状态寄存器和数据路径元件不会自动支持异步时钟和 PLD 逻辑。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 从器件）会在数据手册中具体描述它们处理计时的方式。

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。




```
mpr.M0115:Routing of asynchrononus signal AsyncPin(0):iocell.fb as a clock to UDB component '\Control_Reg_1:ctrl_reg\'
is not supported unless a UDB Clock/Enable component is used.
```

如错误消息中所述，可通过在异步模式中使用 UDBClkEn 组件清除此错误。这并不会清除基础同步问题，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。

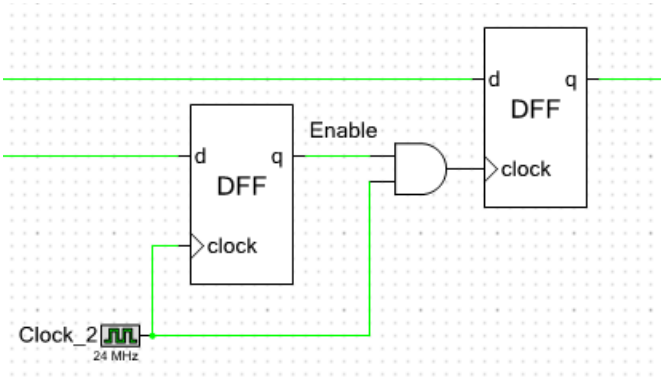
在这两个时钟域之间，唯一可确定的是它们的沿将始终出现在系统时钟的上升沿。因此，这两个时钟域的上升沿的距离与单个系统时钟循环之间的距离一样近。即使某一时钟域是对方的倍数仍然如此，因为它们的时钟分频器不必对齐。如果两个时钟域之间存在组合逻辑，则可能需要插入触发器，以防系统时钟操作的频率受限。通过插入触发器，一个时钟域到另一个时钟域的跨越是一个直接的触发器到触发器路径。

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

同步组件实现使用的是可实现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。

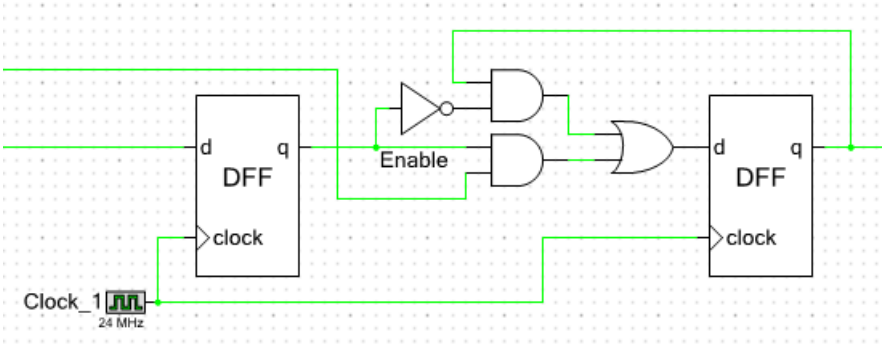
门控时钟

全局时钟不得用于直接对电路提供时钟源以外的其他目的。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。应避免如下电路，因为无法执行时序分析。



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

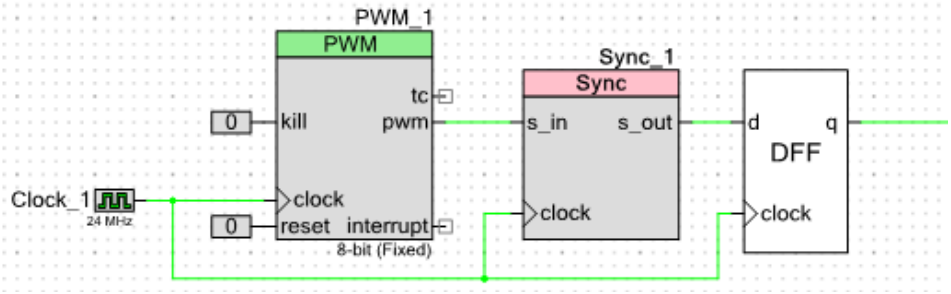
以下电路实现相同的功能，由时序分析支持，仅使用全局时钟，且没有可靠性问题。此电路不对时钟进行门控，而是逻辑使能新数据计时或逻辑维持当前数据。



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 5LP 的更多信息，请参考“CyFlash_SetWaitCycles()”。有关 PSoC 4 的更多信息，请参考“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 模块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，请勿使用升压时钟。

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()函数为所有采用的 SC 模块提供控制模拟泵和升压时钟的功能。

在 SC 模块中实现的组件（TIA、Mixer、PGA 和 PGA_Inv）将根据 Vdda 和非稳定 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()函数在运行时使能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()函数及相应参数来使能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 **Variable Vdda** 设计时间选项进行配置。

通过加载 PSoC Creator 设计范围资源（DWR）文件之“系统”选项卡中的“非稳定 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 **Variable Vdda** 值之间的依存关系。

Vdda	**< 2.7 V**	**≥ 2.7 V**	**≥ 2.7 V**
非稳定 Vdda	保持使能	使能	禁用
创建 ScBoostClk 时钟	有	有	无
复位时启动 ScBoostClk	有	无	无

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 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 和 PSoC 5LP。

PSoC 3/PSoC 5LP API

uint8 CyPLL_OUT_Start(uint8 wait)

说明： 使能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： wait：

- 0：配置后立即返回
- 1：等待PLL锁定或超时

返回值： 状态

- 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。

其他影响和限制： 如果使能等待，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。如果未使能100 KHz ILO，该函数将在函数执行期间使能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的范围必须为1MHz到3MHz。

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

返回值： 无

其他影响和限制： 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，请参考“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

其他影响和限制： 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，请参考“CyFlash_SetWaitCycles()”。

void CyIMO_Start(uint8 wait)

说明： 使能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

其他影响和限制： 如果使能等待，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。如果未使能100 KHz ILO，该函数将在函数执行期间使能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的工作频率

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
8	CY_IMO_FREQ_USB	24 MHz（已针对USB操作进行调整）

返回值： 无

其他影响和限制： 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，请参考“CyFlash_SetWaitCycles()”。

如果选择USB设置，则使能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

其他影响和限制： 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，请参考“CyFlash_SetWaitCycles()”。

void CyIMO_EnableDoubler()

说明： 使能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

其他影响和限制： 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，请参考“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

其他影响和限制： 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，请参考“CyFlash_SetWaitCycles()”。

void CyMasterClk_SetSource(uint8 source)

说明： 设置主器件时钟源。

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

其他影响和限制： 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，请参考“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

其他影响和限制： 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，请参考“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： 使能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振荡器始终被使能的。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振荡器必须被使能的。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： 使能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： 使能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时，启动时间将较长。

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： 使能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。

参数： wait：有效范围[0-255]。这是以毫秒为单位的超时值。合适的值是针对晶振的。

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。

其他影响和限制： 如果使能等待（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。如果未使能100 KHz，该函数将在该函数执行期间使能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： 使能兆赫兹晶振的XERR状态位生成。

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。

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。

参数： 无

返回值： 状态：0：无错误，1：有错误

void CyXTAL_EnableFaultRecovery()

说明： 使能故障恢复电路，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，晶振必须使能且在XERR位为0的情况下运行。

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。

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，请参考器件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。

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

返回值： 无

其他影响和限制： 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。

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

返回值： 无

其他影响和限制： 反馈参考电压必须高于看门狗参考电压。

void CyXTAL_32KHZ_Start()

说明： 使能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使能正向泵。
- 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： 使能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：使能泵。
- 0：禁用泵。

返回值： 无

PSoC 4 API

void CySysClkImoStart(void)

说明： 使能IMO。

参数： 无

返回值： 无

其他影响和限制： 无

void CySysClkImoStop(void)

说明： 禁用IMO。

参数： 无

返回值： 无

其他影响和限制： 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：运行WDT时需要ILO。

void CySysClkIloStart(void)

说明： 使能ILO。

参数： 无

返回值： 无

其他影响和限制： 无

void CySysClkIloStop(void)

说明： 禁用ILO。

参数： 无

返回值： 无

其他影响和限制： 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

其他影响和限制： 无

void CySysClkWriteSysclkDiv (uint32 divider)

说明： 从HFCLK中为SYSCLK选择预分频器的分频值。

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

其他影响和限制： 无

void CySysClkWriteImoFreq (uint32 freq)

说明： 设置IMO频率。

参数： freq：有效范围[3-48]。IMO工作频率。

返回值： 无

其他影响和限制： 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 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、睡眠和休眠。

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 ARM CPU 所支持的标准的 ARM 定义功耗模式。休眠/停止模式的功耗更低，也可以通过固件进入这些模式，同深度睡眠模式一样。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 5LP），要唤醒 CPU，需要中断。实现电源管理的前提是用单独的组件（基于组件的唤醒时间配置）配置唤醒时间，以便计数结束时将发生中断。有关详细信息，请参考“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 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 5LP，当系统性能控制器（SPC）在执行指令时，电源管理器不会将器件设为低功耗模式。器件将在 SPC 执行完指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编辑器上的“Enable Fast IMO During Startup”（在启动过程中使能快速 IMO）选项设置的。如果使能了此选项，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，要通过 CyPmSleep()/CyPmHibernate()将 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，请参考 CyFlash_SetWaitCycles()函数的说明。

必须为所有器件使能 1 KHz ILO（默认情况下，无论时钟编辑器中的选项如何，1 KHz ILO 始终是使能的），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，请参考相应的器件寄存器 TRM。

调用 CyPmSaveClocks()函数将修改器件时钟配置。因此，不会使用时钟组件，直到调用 CyPmRestoreClocks()函数，这样将恢复初始时钟配置。有关组件时钟要求的更多信息，请参考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，请参考器件 TRM 和数据手册。

有两种配置唤醒时间的方法：

- 通过调用 CyPmSleep()和 CyPmAltAct()函数以及所需参数，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 时间间隔要小于指定的时间间隔。要使后续时间间隔具有额定值，由 CyPmSleep()和 CyPmAltAct()函数使能对应的定时器，并使定时器保持使能状态。注意某些 API 也可以使用此定时器。这会导致在进入低功耗模式之前，定时器时钟始终处于使能状态（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。

唤醒之后，必须调用 CyPmReadStatus()函数以及相应的参数（例如如果器件配置为基于 CTW 唤醒，则使用 CY_PM_CTW_INT 参数），来清除中断状态位。

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。通过源可唤醒器件，但是尚未配置该源以进行此操作。必须正确配置与唤醒源相关的组件以作为唤醒源使用。

对于 PSoC 5LP 器件，还必须使能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使能，都会将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 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。使用“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。对于激活功耗模式配置，关闭所有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系统编辑器的“Enable Fast IMO During Startup”（在启动过程中使能快速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

其他影响和限制： 调用此API后，所有外设时钟都将关闭。

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。

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，必须针对作为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，且函数将返回活动状态。

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，此时定时器为使能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 5LP：两个参数都不适用。这意味着该参数被指定为NONE。器件将进入“备用活动”模式，直到发生所使能的中断为止。

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

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毫秒
3	PM_ALT_ACT_TIME_CTW_4MS	CTW：4毫秒
4	PM_ALT_ACT_TIME_CTW_8MS	CTW：8毫秒
5	PM_ALT_ACT_TIME_CTW_16MS	CTW：16毫秒
6	PM_ALT_ACT_TIME_CTW_32MS	CTW：32毫秒
7	PM_ALT_ACT_TIME_CTW_64MS	CTW：64毫秒
8	PM_ALT_ACT_TIME_CTW_128MS	CTW：128毫秒
9	PM_ALT_ACT_TIME_CTW_256MS	CTW：256毫秒
10	PM_ALT_ACT_TIME_CTW_512MS	CTW：512毫秒
11	PM_ALT_ACT_TIME_CTW_1024MS	CTW：1024毫秒
12	PM_ALT_ACT_TIME_CTW_2048MS	CTW：2048毫秒
13	PM_ALT_ACT_TIME_CTW_4096MS	CTW：4096毫秒
14 - 269	PM_ALT_ACT_TIME_FTW(1-256)	FTW：10微秒至2.56毫秒

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 5LP，此参数将被忽略。

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作为唤醒源使用时，必须在唤醒时使用对应的参数调用CyPmReadStatus函数。有关更多信息，请参考CyPmReadStatus API。

返回值： 无

其他影响和限制： 对于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，此时定时器为使能状态且中断为禁用状态。同时，ILO 1 KHz（如果CTW定时器作为唤醒定时器使用）或ILO 100 KHz（如果FTW定时器作为唤醒定时器使用）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，必须为作为唤醒定时器使用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，请参考《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，且函数将返回活动状态。

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，此时定时器为使能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 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 5LP，此参数将被忽略。

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毫秒
3	PM_SLEEP_TIME_CTW_4MS	CTW：4毫秒
4	PM_SLEEP_TIME_CTW_8MS	CTW：8毫秒
5	PM_SLEEP_TIME_CTW_16MS	CTW：16毫秒
6	PM_SLEEP_TIME_CTW_32MS	CTW：32毫秒
7	PM_SLEEP_TIME_CTW_64MS	CTW：64毫秒
8	PM_SLEEP_TIME_CTW_128MS	CTW：128毫秒
9	PM_SLEEP_TIME_CTW_256MS	CTW：256毫秒
10	PM_SLEEP_TIME_CTW_512MS	CTW：512毫秒
11	PM_SLEEP_TIME_CTW_1024MS	CTW：1024毫秒
12	PM_SLEEP_TIME_CTW_2048MS	CTW：2048毫秒
13	PM_SLEEP_TIME_CTW_4096MS	CTW：4096毫秒

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。

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。

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作为唤醒源使用时，必须在唤醒时使用对应的参数调用CyPmReadStatus函数。有关更多信息，请参考CyPmReadStatus API。

CyPmSleep（续）

返回值： 无

其他影响和限制： 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，此时定时器为使能状态且中断为禁用状态。同时，ILO 1 KHz（如果CTW定时器作为唤醒定时器使用）将保持启动状态。

使能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周期唤醒器件、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，请参考器件勘误表。

void CyPmHibernate()

说明： 将器件置于“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，就将器件配置为从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的行为。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。

参数： 无

返回值： 无

其他影响和限制： 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时长，睡眠电压调节器可在下一次休眠/睡眠事件发生之前稳定下来。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，并使能对未来事件的检测。

使能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除。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，请参考器件 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： 将器件置于“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。任何使能的中断均可导致从睡眠模式中被唤醒。

参数： 无

返回值： 无

其他影响和限制： 无

void CySysPmDeepSleep(void)

说明： 将器件置于“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，请参考对应的外设数据手册。

参数： 无

返回值： 无

其他影响和限制： 无

void CySysPmHibernate(void)

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

参数： 无

返回值： 无

其他影响和限制： 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将从复位矢量开始启动。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，或固件启动或使能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： 将器件置于“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

其他影响和限制： 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： polarity：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

其他影响和限制： 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。

其他影响和限制： 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

其他影响和限制： 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

其他影响和限制： 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使能，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 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

其他影响： 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()函数前已被使能，则_Wakeup()函数将重新使能此组件。

参数： 无

返回值： 无

其他影响： 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“Configure”对话框中所定义的或通过相应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

其他影响： 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

其他影响： 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

除了下面所述的例外，此章节中的 API 应用于所有架构。有关中断的更多信息，另请参考中断组件数据手册。

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。对 4 CPU 指令（8 个 CPU 周期）禁用中断，以调整大型的可重入堆栈。

API

CyGlobalIntEnable

说明： 使用全局中断屏蔽使能中断的宏语句。

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： 之前已使能的中断的32位掩码

void CyEnableInts(uint32 mask)

说明： 使能32位掩码中指定的所有中断。

参数： mask：要使能中断的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 寄存器位和中断使能状态（EA）恢复为在入口时的状态。只要使用了合适的嵌套 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： 使能指定的中断编号。

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 寄存器位和中断使能状态（EA）恢复为在入口时的状态。只要使用了合适的嵌套 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 寄存器位和中断使能状态（EA）恢复为在入口时的状态。只要使用了合适的嵌套 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使能状态。

参数： number：中断编号。有效范围：[0-31]

返回值： 使能状态：如果已使能，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]

priority：中断优先级。0是最高优先级。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： 强制指定中断编号处于待处理状态。

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

除了引脚组件所提供的引脚功能外，在 *cypins.h* 文档中还为 PSoC 3/PSoC 5LP 器件提供了引脚宏的程序库。尚未为 PSoC 4 器件提供引脚宏程序库。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 5LP 器件上的每个引脚。寄存器的地址在 *cydevice_trm.h* 文件中提供。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 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 5LP）

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 5LP）

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 5LP）

mode：所需的驱动模式

定义	源
CY_PINS_DM_ALG_HIZ	模拟高阻
CY_PINS_DM_DIG_HIZ	数字高阻
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 5LP）

返回值： 引脚的当前驱动模式

定义	源
CY_PINS_DM_ALG_HIZ	模拟高阻
CY_PINS_DM_DIG_HIZ	数字高阻
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 5LP）

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 5LP）

返回值： 无

PSoC 4 API

CY_SYS_PINS_READ_PIN(portPS, pin)

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对各端口进行了定义：CYREG_PRTx_PS，其中x为端口号（0至4）。
pin：引脚编号（0至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, pin)

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对各端口进行了定义：CYREG_PRTx_DR，其中x为端口号（0至4）。
pin：引脚编号（0至7）。

返回值： 无

CY_SYS_PINS_CLEAR_PIN(portDR, pin)

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对各端口进行了定义：CYREG_PRTx_DR，其中x为端口号（0至4）。
pin：引脚编号（0至7）。

返回值： 无

CY_SYS_PINS_SET_DRIVE_MODE(portPC, pin, mode)

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对各端口进行了定义：CYREG_PRTx_DR，其中x为端口号（0至4）。
pin：引脚编号（0至7）。
mode：所需的驱动模式

定义	源
CY_SYS_PINS_DM_ALG_HIZ	模拟高阻
CY_SYS_PINS_DM_DIG_HIZ	数字高阻
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, pin)

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对各端口进行了定义：CYREG_PRTx_DR，其中x为端口号（0至4）。
pin：引脚编号（0至7）。

返回值： 引脚的当前驱动模式

定义	源
CY_SYS_PINS_DM_ALG_HIZ	模拟高阻
CY_SYS_PINS_DM_DIG_HIZ	数字高阻
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

宏的库提供对器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。这可以生成与器件相关的代码。

PSoC 3 是 8 位架构，因此处理器没有字节顺序。但是，8 位架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 5LP 处理器架构使用小端模式排列。

所有架构中的 SRAM 和闪存都是通过使用架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，您必须考虑特定架构的字节顺序。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 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。

PSoC 4 要求外设寄存器访问与硬件寄存器的大小相匹配。否则，外设将忽略此数据操作，而且硬故障将被生成。

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 5LP）

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 5LP）

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 5LP）

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 5LP）

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 5LP）

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 5LP）

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 5LP）

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 5LP）

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 5LP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 5LP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 5LP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 5LP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 5LP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 5LP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 5LP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 5LP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。

9 闪存和 EEPROM

PSoC 3/PSoC 5LP 实现

闪存架构

PSoC 器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纠错码（ECC）数据提供非易失性存储空间。根据器件类型，主闪存区可以包含多达 256 KB 的用户程序存储空间。

闪存被组织为一组阵列。每个阵列由 64、128 或 256 行组成。其中每行均含有 256 个数据字节加 32 个字节的 ECC。如果未使用 ECC，则此空间可以存储器件配置数据和批量用户数据。不能在 ECC 闪存空间内执行用户代码。

闪存存储器的阵列结构

Row 0	Data (256 bytes)	ECC (32 bytes)
Row 1	Data	ECC
	⋮	
Row N	Data	ECC

PSoC 3 闪存存储器具有下列特性：

- 组织为包含 64、128 或 256 行的陈列；
- 每行均含有 256 个数据字节加 32 个字节的 ECC 或数据存储器。

PSoC 5LP 闪存存储器具有下列特性：

- 组织为包含一个 128 行或 256 行的阵列，或包含多个 256 行的阵列。
- 每行均含有 256 个数据字节加 32 个字节的 ECC 或数据存储器。

有关闪存架构的详细信息，请参考器件数据手册和 TRM。

PSoC Creator 设计范围资源（DWR）文件的 **System** 选项卡包含定义 ECC 区利用率的配置选项：

DWR选项	ECC功能
Enable Error Correcting Code（ECC）（使能纠错码（ECC））	每8个字节，ECC便可纠正一位错误并检测多位错误。ECC区存储纠错码数据。

DWR选项	ECC功能
Store Configuration Data in ECC Memory（将配置数据存储在ECC存储器中）	器件配置数据被存储在ECC区内，旨在降低主闪存存储器的使用情况。当使能该选项时，将不会使用错误纠正功能。 **注意：**对于Bootloader项目，该选项始终被禁用。这是因为ECC区专用于Bootloadable项目。
None（无）	ECC功能被禁用。可以将用户数据存储在ECC区内。

有关使用 ECC 的详细信息，请参考 TRM 的*闪存程序存储器*一章。

PSoC 器件均包括灵活的闪存保护模型，以防止访问和查看片上闪存。器件允许用户为闪存的每一行指定四个保护级别中的其中一个保护级别：

- 无保护
- 工厂升级
- 现场升级
- 完全保护

通过使用 PSoC Creator DWR 文件的 **Flash Security** 选项卡可以选择所需要的保护级别。要更改闪存保护级别，必须擦除整个闪存。闪存编程 API 无法写入完全保护级别的行。有关保护模型的更多信息，请参考“PSoC Creator 帮助”的*闪存安全编辑器*部分。

EEPROM 架构

PSoC EEPROM 存储器是一种按字节寻址的非易失性存储器。EEPROM 的结构为一组阵列。PSoC 3 和 PSoC 5LP 架构均有一个 EEPROM 阵列，其大小为 512 字节、1 KB 或 2 KB。阵列则可根据器件而包含 32、64 或 128 行。每行含 16 字节数据。

使用闪存和 EEPROM

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。

注意：当写入闪存时，指令缓存中的数据将为稳定状态。因此，缓存数据与刚写入闪存内的数据无关。通过调用 CyFlushCache()函数使缓存中的数据无效并强制加载闪存中的最新信息。

下表显示的是特定于器件的闪存参数的定义。通过这些参数可以运行闪存：

值	说明
CY_FLASH_BASE	闪存的基本地址。
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。

EEPROM API 提供下列特定于器件的定义：

值	说明
CY_EEPROM_BASE	EEPROM存储器的基本地址。
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。

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 提供用于运行闪存和 EEPROM 的统一方法，并通过提取详细信息简化与 SPC 的交互。

写入 EEPROM 或闪存可能需要 20 毫秒的时间。在这个时间内不能复位器件，否则将导致 EEPROM 或闪存的意外更改。复位源包括 XRES 引脚、软件复位以及看门狗；需要确保这些源不被无意激活。另外，低电压检测线路可以配置为生成中断而不器件复位。

PSoC 器件具有片上温度传感器对内部 Die 温度进行测量。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 Die 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。

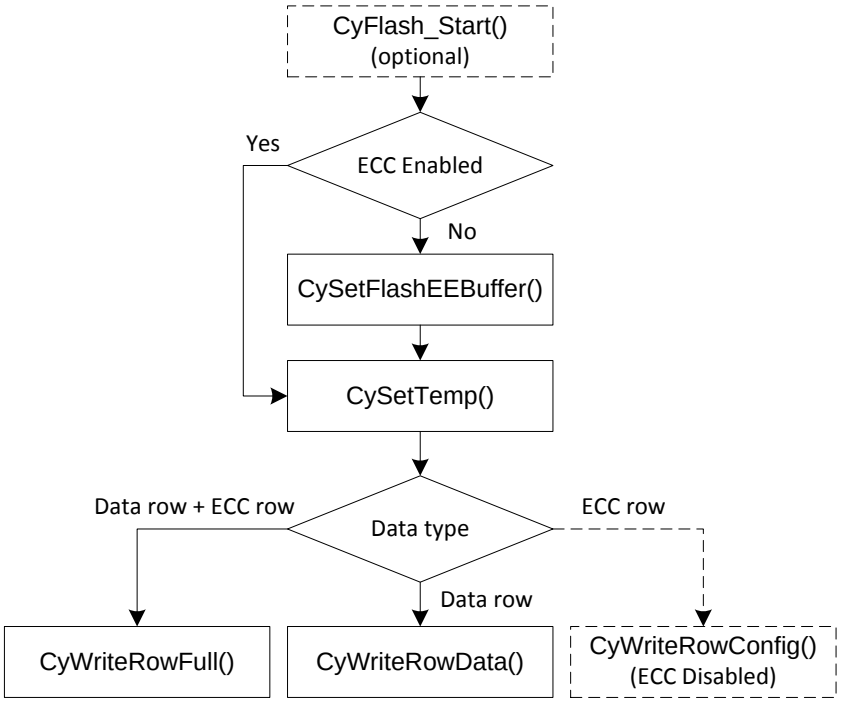
禁用错误检测或更正功能进行闪存编程时（此时 ECC 闪存空间将用于数据存储），写入一行数据有多种方法：

- 使用 CyWriteRowFull()函数写入包含 ECC 的行；
- 使用 CyWriteRowData()函数写入没有 ECC 的行；
- 使用 CyWriteRowConfig()函数写入 ECC 存储器。

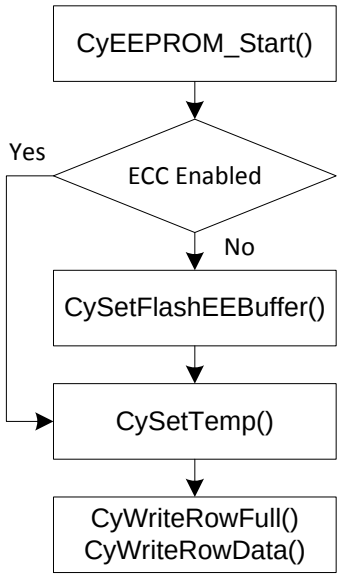
如果禁用闪存 ECC 特性，您还需分配缓冲区并将其传入用于闪存和 EEPROM 编程的 CySetFlashEEBuffer()函数。与 SPC 通信时，该缓冲区将用于存储中间数据。有关缓冲区分配的详细信息，请参考本章 API 一节的 CySetFlashEEBuffer()函数说明。

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，请参考器件 TRM。每个阵列 ID 均从 0 开始对行进行编号。

闪存编程框图



EEPROM 编程框图



闪存和 EEPROM API

cystatus CySetTemp()

说明： 更新了片上温度传感器中当前芯片温度值的稳定快照。在执行一系列闪存/EEPROM写入函数前，必须调用一次该函数。如果该应用程序使用环境中的die温度明显变化（10°C或以上），则应该确保温度快照值最新，以调整闪存写入时间从而达到最佳性能。

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

其他影响和限制： 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。

如果该阵列为闪存阵列：

DWR闪存配置	说明
使能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。
使能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。
使能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：要写入的阵列的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：要编程数据的地址。**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	未接受指令
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。

如果该阵列为闪存阵列：

DWR闪存配置	说明
使能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。
使能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：要写入的阵列的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：要编程数据的地址。

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	未接受指令
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有效（对EEPROM无效）。

DWR闪存配置	说明
使能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。
使能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，因为配置数据存储在ECC存储器中。
使能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：要写入的阵列的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：要编程数据的地址。

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	未接受指令
其他非零数据	失败

void CyFlash_Start()

说明： 使能闪存。默认情况下闪存为使能状态。

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。该设置仅在CPU禁用后生效。

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： 使能EEPROM。
EEPROM由单独的位控制，且在启动后才可以被使用。

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。

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 提供下列特定于器件的定义：

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_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	未接受指令
CYRET_BAD_PARAM	一个或多个无效参数
其他非零数据	失败

其他影响和限制： 调用此函数前必须先使能IMO。闪存写入硬件操作取决于IMO。

void CySysFlashSetWaitCycles(uint32 freq)

说明： 设置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量。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工作频率。
注意：无效的频率将被忽略。

返回值： 无

其他影响和限制： 无

10 Bootloader 移植

本章包含有关 PSoC Creator 2.1 中引入的 bootloader 架构的信息，以及在将 bootloader 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 bootloader 系统以提供更多的配置选项。在之前发布的版本中，bootloader 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 bootloader 系统包含两个独立的 bootloader 和 Bootloadable 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 bootloader 系统配置选项从 PSoC Creator 设计范围资源（DWR）文件移入相应的 bootloader 和 Bootloadable 组件配置对话框中。有关更多信息，请参考本章的移植 bootloader 设计和移植 Bootloadable 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 bootloader 系统。然而，某些更改可能影响现有 bootloader 和 Bootloadable 设计。这需要在移至新的 bootloader 系统架构时进行维护。bootloader 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
- 仅将 Bootloadable 设计移植至 PSoC Creator 2.1 或更高的版本

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 bootloader 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“Update All to Latest”（更新所有组件至最新版本）按钮以确保更新为最新的版本。
- 打开 bootloader 系统设计，并将 bootloader 组件放置在 **Bootloader** 或**多应用 bootloader** 应用类型原理图上。将可 Bootloadable 组件放置在 **Bootloadable** 应用类型原理图上。有关更多信息，请参考项目应用类型属性更改中介绍的内容。

- 将 bootloader 系统设置从 DWR 文件转为组件配置对话框。有关详细信息，请参考移植 bootloader 设计和移植 Bootloadable 设计中介绍的内容。

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 bootloader 系统设计配合使用。要继续与 bootloader 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 bootloader 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。

仅将 Bootloadable 设计移植至 PSoC Creator 2.1 或更高的版本

在这种情况下，只有 Bootloadable 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 bootloader 设计与新建或移植的 Bootloadable 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 bootloader 设计。

移植 bootloader 设计

下表显示 PSoC Creator 早期版本中的 bootloader 系统和 PSoC Creator 2.1 中引入的 bootloader 系统之间的关联。

2.1之前的PSoC Creator版本 （DWR“系统”选项卡中的“bootloader”部分）	PSoC Creator 2.1或更高版本以及cy_boot 3.0或更高版本（bootloader组件）
IO组件	通信组件
由设计的**应用类型**属性设置请参考项目应用类型属性更改一节。	多应用bootloader
等待指令	等待指令
等待指令的时间（ms）	等待指令的时间（ms）
版本	Bootloader应用版本
数据包校验和类型	数据包校验和类型
快速应用验证	快速Bootloadable应用验证
N/A	Bootloader应用的验证
N/A	可选指令

有关 PSoC Creator bootloader 系统功能的详细信息，请参考 bootloader 组件数据手册。

移植 Bootloadable 设计

下表显示 PSoC Creator 2.1 中引入的 Bootloadable 系统和 PSoC Creator 2.1 之前版本中的 Bootloadable 系统之间的关联。

2.1之前的PSoC Creator版本（DWR“System”选项卡中的“Bootloadable”部分）	PSoC Creator 2.1或更高版本以及cy_boot 3.0或更高版本（Bootloadable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
N/A	手动应用映像放置
N/A	放置地址
在项目Dependencies窗口的**Bootloader**选项卡中设置。请参考项目应用类型属性更改一节。	Bootloader十六进制文件（bootloader组件配置对话框中的**Dependencies**选项卡）。

早先在 Bootloadable 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 Bootloadable 组件的实例名。

有关 PSoC Creator bootloader 系统功能的详细信息，请参考 Bootloadable 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，**Application Type**（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，**Application Type**（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 **Application Type**（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
bootloader	禁用了多应用bootloader选项中的bootloader组件。
多应用bootloader	使能了多应用bootloader选项中的bootloader组件。
Bootloadable	Bootloadable组件。

Application Type 选项在 **New Project** 对话框中高级部分设置，以后可在 **Code Generation** 部分下面项目的 **Build Settings** 窗口中进行修改。

移植 Bootloadable 依赖关系至 bootloader 设计

在 PSoC Creator 2.1 之前的版本中，项目依赖关系窗口中的 **Bootloader** 选项卡可将 bootloadable 项目链接至 bootloader 项目。通过将 Bootloadable 组件引入具有 cy_boot version 3.0 或更高版本的 PSoC Creator 2.1，该选项被移到 Bootloadable 组件配置对话框的 **Dependencies** 选项卡中。有关详细信息，请参考组件数据手册。

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，表明先前是否使能了中断（实际值取决于器件架构）。

注意：实现CyEnterCriticalSection能够控制IRQ使能位，并使中断仍处于使能状态。对所有架构而言，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，中断已使能，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 5LP — 如果之前已使能中断，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，中断已使能，则CyExitCriticalSection将重新使能中断。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，则使能。

返回值： 无

void CyHalt(uint8 reason)

说明： 停止CPU。

参数： reason：要传入的值，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。其他开销因素，包括函数入口和出口，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

其他影响和限制： 通过假设使能了指令缓存，已实现CyDelay。PSoC 5LP上禁用指令缓存时，CyDelay导致的延迟将变为两倍。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：延迟的毫秒数。

返回值： Void

其他影响和限制： 通过假设使能了指令缓存，已实现CyDelayUs。PSoC 5LP上禁用指令缓存时，CyDelayUs导致的延迟将变为两倍。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较小的非整数，实际延迟最长等于额定值的两倍。实际延迟不会小于额定值。

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，延迟指定的周期数。

参数： cycles：延迟的周期数。

返回值： 无

PSoC 3/PSoC 5LP 电压检测 API

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。如果使能 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）”，无论何时使能的 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，请参考器件 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，由于要使能硬件繁忙，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，使能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**：使能数字LVI中断时的器件复位：

- 发生数字LVI事件时进行中断
- 非0 — 发生数字LVI事件时进行复位

threshold：设置数字低电压监控电路的激发点，其中，它的步长约为250 mV，有效范围为1.70 V （0x00）到5.45 V （0x0F）。

返回值： 无

其他影响和限制 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电路不再使能，器件退出复位。如果电源低于激发电平且固件使能LVI复位功能，器件将重新复位。只要电源低于激发电平或用户使能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等于或低于激发点时，使能模拟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**： 使能模拟LVI中断时的器件复位：

- 0 —发生模拟LVI事件时进行中断
- 非0 — 发生模拟LVI事件时进行复位

threshold：设置模拟低电压监控电路的激发点，其中，它的步长约为250 mV，有效范围为1.70 V （0x00）到5.45 V （0x0F）。

返回值： 无

其他影响和限制 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电路不再使能，器件退出复位。如果电源低于激发电平且固件使能LVI复位功能，器件将重新复位。只要电源低于激发电平或用户使能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（禁用中断和器件复位）。

参数： 无

返回值： 无

其他影响和限制 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（禁用中断和器件复位）。

参数： 无

返回值： 无

其他影响和限制 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： 通过使能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

其他影响和限制 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。

器件	值	定义	寄存器[位]
PSoC 3/ 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。

其他影响和限制 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。

器件	值	定义	寄存器[位]
PSoC 3/ 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]

其他影响和限制 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，使能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：低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： 禁用低压检测。低压中断被禁用。

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 — 表示低压检测中断

void CySysLvdClearInterrupt(void)

说明： 清除低压检测中断状态。

参数： 无

返回值： 无

缓存功能

PSoC 3

默认使能 PSoC 3 缓存。可使用 PSoC Creator 设计范围资源系统编辑器禁用它。没有用于 PSoC 3 缓存处理的定义、函数或宏。

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 器件无缓存功能。

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
- 设置执行的处理器
- 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 5LP 启动的详细信息，请参考应用手册 AN60616 中介绍的内容。

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。

硬件启动对器件进行配置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，开始执行闪存代码。

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 器件按 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。不存在特意与链接相关联的文件。但可以使用链接器指令来代替。

PSoC 4/PSoC 5LP

赛普拉斯已定制开发了启动和链接器脚本，但我们当前支持的两家工具链供货商都提供链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映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 **Build Setting** 窗口并在 **Linker**（链接器）目录下的 **Custom Linker Script** 字段中指定文件路径来使用自定义链接器描述符文件，而不是使用默认的链接接器描述符文件。

非对齐的传输（PSoC 5LP）

PSoC 5LP 支持单访问的非对齐传输，但是存在针对 Cortex-M3 平台的某些限制。当使用非对齐传输时，它们将被转换为多个对齐传输。从软件的角度来看，此转换是很透明的。发生非对齐传输时，它将被分为多个单独的传输。因此，进行单数据访问时将需要更多的时钟周期。为了获得最佳性能和确保 PSoC 处理器系列的代码兼容，需要避免非对齐的传输。

在 PSoC 5LP 器件内，SRAM 位于 Cortex-M3 代码/SRAM 区的边界。就架构而言，跨越存储器映射边界的非对齐访问是不可预测的。用于 PSoC 5LP 的链接器配置文件，不防止跨越该边界的非对齐访问。

当必须使用非对齐访问时，将使用一个函数来检查可能出现的边界问题，然后以此边界进行字节访问或修改链接器脚本，旨在强制使需要以非对齐式被访问的存储器不跨越此边界。

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，所有这些库都已链接。

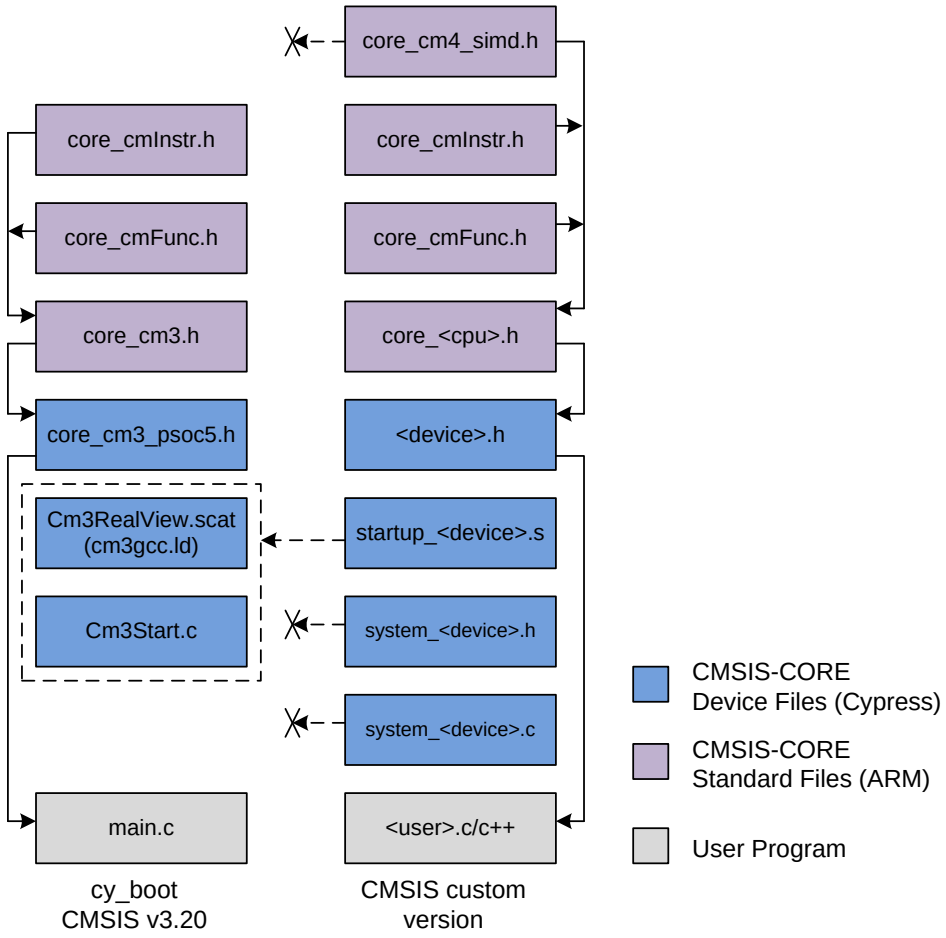
- 支持 RTOS 和用户替换子程序。因为库子程序记为“弱”，如果提供另一种实现，则允许进行替换，所以能够实现对 RTOS 和用户替换子程序的支持。
- 所提供的机制允许使用用户版本的链接器/分散文件进行替换。允许用户创建项目本地文件，且通过构建设置允许该文件规范代替自动提供的文件作为链接器/分散文件，从而实现该功能。
- 目前，堆栈大小指定为规定值（栈为 4K，堆为 1K）。如有可能，应将指定堆栈大小的要求全部删除。如果不可删除，则这些值应为默认值，并可在 DWR GUI 中选择其他数值。
- 生成源代码树中的所有代码都将编译成单一的库，作为构建过程的一部分。然后，通过最后的链接过程对已编译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

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。支持多个级别。支持核心外设访问层（CMSIS Core）。

PSoC Creator 3.0 提供对 CMSIS Core 3.20 的支持。PSoC Creator 3.0 还提供使用自定义版本 CMSIS Core 的功能。

下面的框图显示的是任何将 CMSIS Core 3.20 文件集成到 cy_boot 组件以及如何集成 CMSIS Core 自定义版本文件。



以下介绍的是图中的每个文件：

- cy_boot 组件中的 *Cm3Start.c* 和 *cm3gcc.ld* 文件含有 Cortex-M3 器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器件文件<器件>.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。
- 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（即处理器时钟和存储器总线系统）的通用文件的一部分由 *Cm3Start.c* 文件使用。

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 CMSISCore 外设库文件”。默认使能该选项，CMSIS Core 3.20 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 3.20 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

#define __CM3_REV 0x0201

#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，请参考器件数据手册和 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很透明，并且不会影响到引导加载、调试或正常器件功能。有关更多信息，请参考器件勘误表。

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。bootloader 和 Bootloadable 项目可以使用这些位，但用户不能使用。

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

13 看门狗定时器

PSoC 3/PSoC 5LP API

void CyWdtStart(uint8 ticks, uint8 lpMode)

说明： 使能看门狗定时器。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，且看门狗定时器已使能。

一旦看门狗定时器使能后，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。看门狗定时器使能后，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。

参数： ticks：四个可用定时器周期之一。一旦使能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。

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

void CyWdtClear()

说明： 清除（馈送）看门狗定时器。

参数： 无

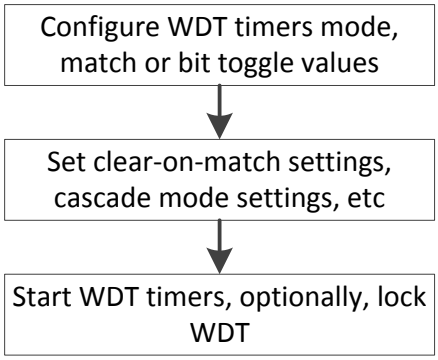
返回值： 无

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，这需要使能 ILO。因此，WDT API 默认使能 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

其他影响和限制： 如果ILO之前为禁用状态，此API会使能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

其他影响和限制： 如果ILO之前为禁用状态，此API会使能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	生成中断，并生成由于第3个未处理中断的复位。（仅对计数器0和计数器1有效）

返回值： 无

其他影响和限制： 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会使能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。

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。

enable：0为禁用，1为使能

返回值： 无

其他影响和限制： 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会使能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：1为使能，0为禁用。

void CySysWdtEnable(uint32 counterMask)

说明： 使能指定的WDT计数器。使能掩码中指定的所有计数器。

参数： counterMask：要使能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

其他影响和限制： 使能或禁用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会使能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

其他影响和限制： 使能或禁用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会使能ILO。

uint32 CySysWdtReadEnabledStatus (uint32 counterNum)

说明： 读取三个WDT计数器之一的使能状态。

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－计数器已使能

其他影响和限制： 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

其他影响和限制： 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会使能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

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。

match：有效范围[0-65535]。用于匹配计数器的值。

返回值： 无

其他影响和限制： 如果ILO之前为禁用状态，此API会使能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的模式已使能中断，则中断生成。

参数： bit：有效范围[0-31]。将检测切换的计数器2的位。

返回值： 无

其他影响和限制： 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会使能ILO。

uint32 CySysWdtReadToggleBit(void)

说明： 读取WDT计数器2中被检测以进行切换的位。

参数： 无

返回值： 将被检测到的位（范围为0到31）。

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。更改后可能需要最多三个LFCLK周期才会生效。

返回值： 16位匹配值。

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。

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

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：要使能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

其他影响和限制： 如果ILO之前为禁用状态，此API会使能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

其他影响和限制： 如果ILO之前为禁用状态，此API会使能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器件中的固件质量和鲁棒性。

定义了两种类型的偏差：

- 项目偏差 — 适用于所有 PSoC Creator 器件的偏差
- 特定偏差 — 适用于特定组件的偏差

本节提供了以下项目的相关信息：

- 验证环境
- 项目偏差
- 文档相关规则
- PSoC Creator 生成源偏差
- cy_boot 组件特定偏差

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	生产
	PSoC 4	生产
	PSoC 5LP	生产
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]	规则说明	偏差说明
1.1	R	此规则规定，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）与器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	验证没有标志符名再次被使用。	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	如果仅从单个函数中访问对象，则应在模块范围内定义这些对象。	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。

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。

[1] 必须/建议

MISRA-C: 2004规则	规则类（必须/建议）[1]	规则说明	说明
1.3	R	如果面板代码有通用定义的界面标准，而且多个语言/编译器/汇编器符合此标准，才可应用多个编译器和/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考“帮助”菜单、“文档”子菜单、“Keil”和“GCC”指令。
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。	当对两个带符号的整数进行除法计算时，并且一个为正数，另外一个为负数，编译器将对得到的负余数进行求整。
3.5	R	此规则需要记录实现的定义行为和位字段的封包。	避免使用位字段。
3.6	R	应用于生产代码中的所有库写入需要符合此文件规定，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。编译器可能也插入了对供货商特定的编译器支持库的调用。

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（必须/建议）[1]	规则说明	偏差说明
3.4	R	*#pragma*指令的所有使用情况将被记录。	#pragma指令需要保证C51编译器生成高效的代码，使用于同AMuxSeq组件相关的生成函数。
11.4	A	该规则规定了不同对象指针类型间不可进行转换。	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。
14.1	R	该规则要求不应包含得不到的代码。	通常使用CYMEMZERO、CYMEMZERO8、CYCONFIGCPY、CYCONFIGCPY8、CYCONFIGCPYCODE和CYCONFIGCPYCODE8，但不总是使用这些存储器。
15.2	R	Switch语句的结尾必须为*break*语句。	代码结构需要保证C51编译器生成高效的代码，使用于同AMuxSeq组件相关的生成函数。
15.3	R	*default*必须为*switch*语句中的最后一条。	代码结构需要保证C51编译器生成高效的代码，使用于同AMuxSeq组件相关的生成函数。

MISRA-C: 2004规则	规则类（必须/建议）[1]	规则说明	偏差说明
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宏以器件独立的方式调用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（必须/建议）[1]	规则说明	偏差说明
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 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”。
8.7	R	如果仅从单个函数中访问对象，则应在模块范围内定义这些对象。	对于特意将PSoC 4/PSoC 5LP，cySysNoInitDataValid变量声明为Cm0Start.c/Cm3Start.c文件中的全局变量，这样是为了防止链接器在CY_NOINIT部分中被清除。
8.12	R	使用外部链接声明某一阵列时，它的大小将被显式说明或通过初始化明确定义。	对于PSoC 4/PSoC 5LP（Cm0Start.c/Cm3Start.c），结构的__cy_regions阵列被声明为未知大小。
12.10	R	不使用逗号操作符。	Cm0Start.c/Cm3Start.c文件包含复杂的for()语句，该语句执行部件初始化且具有上述的偏差。
12.13	A	在一个表达式中，请勿将递增（++）和递减（--）操作符与其他操作符混合在一起。	
13.2	A	对一个非零值进行显性测试，除非操作数是有效的布尔值。	
13.5	R	“for”语句的三个表达式仅用于回路控制。	
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 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 5LP: CMSIS Core：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。
11.4	A	不同对象指针类型间不可显式转换。	DMA和中断API使用不同对象指针类型间的转换。
11.5		不被执行的转换，该转换将清除指针寻址类型中的任何常量或易失性资质。	指针被传送到memcpy()函数之前，在将此指针强制设置为“void”类型期间，将丢失易失性的性质。
14.7	R	函数应在结尾处有单一的退出点。	CyPmSleep()和CyPmHibernate()函数具有复杂的条件结构；如果器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	数组索引是唯一允许的指针运算形式。	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。

MISRA-C: 2004规则	规则类（必须/建议）[1]	规则说明	偏差说明
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	使用时，函数应优先于类函数宏。	由于使用了函数宏以实现更高效的代码，所以出现了偏差。

15 cy_boot 组件更改

版本 4.0

本节列出并说明了 4.0 版 cy_boot 组件的主要更改：

4.0版的更改说明	更改原因/影响
在闪存和EEPROM部分中添加了一个注释，表示bootloader项目类型的“ECC存储器中的存储配置数据”DWR选项不可用。	
在“使用闪存和EEPROM”部分中添加了一个注释，表示当写入闪存时，指令缓存中的数据将为过期的数据。	通过调用CyFlushCache()函数使缓存中的数据无效并强制加载闪存中的最新信息。
纠正了CyDmaChEnable()和CyDmaChDisable()函数的问题。	如果在这些函数执行期间发生DMA请求，DMA通道将被损坏。API已被修改，以解决此问题。
清除了PSoC 5器件的参考内容。	PSoC 5已被PSoC 5LP取代。
PSoC Creator生成源偏差部分更新了AMuxSeq组建相关的MISRA偏差。	

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使能，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器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，请参考器件勘误表。

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。	
已添加了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（前提条件是此中断作为唤醒事件使用，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器件，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。这意味着该参数被指定为NONE。器件将进入“备用活动”模式，直到使能中断。
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bootloader，以将bootloader和Bootloadable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，请参考第72的Bootloader移植。
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。

3.10版的更改说明	更改原因/影响
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bootloader，以将bootloader和Bootloadable项目组件分开。	有关详细信息，请参考第72的Bootloader移植。
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。
更新了CyXTAL_32KHZ_Start()、CyXTAL_32KHZ_Stop()、CyXTAL_32KHZ_ReadStatus()和CyXTAL_32KHZ_SetPowerMode() API的实现。	已添加了额外超时设定，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器件上成功启动MHZ XTAL。
对于PSoC 5部件，清除了以下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，增加器件的启动时间。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，清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CyFlash_SetWaitCycles() API的说明。	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，清除了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

3.0版的更改说明	更改原因/影响
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。
更新了*《复位状态的保留》*一节，使其内容更加详细。	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。	为器件操作提供更详细的信息。
清除了下列宏：CYWDT_TICKS CYWDT_CLEAR、CYWDT_ENABLE CYWDT_DISABLE_AUTO_FEED。	应使用CyWdtStart()和CyWdtClear()代替。
对于PSoC 5器件，清除了CyCpuClk_SetDivider()。	硬件不支持该功能。
清除了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。	如果之前已使能中断，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都是真正可重入的函数。组件API源文件中的注释指出了适用的函数。 对于采用了安全方式并且是不可重入的函数，则需要该项变更，这样可以消除编译器警告：通过标志或关键节防止同时调用。
添加了PSoC 5LP支持。	

2.40 版和更早的版本

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。	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。
通过在进入器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，用户可以配置时钟功耗模式。 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，以在使能“Enable Fast IMO during startup”（在启动过程中使能快速 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。
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，使在恢复器件状态之前不会出现中断。
对于PSoC 5器件，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。	CyPmSleep()和CyPmAltAct()函数只能使用下面各参数：CyPmSleep（PM_SLEEP_TIME_NONE、PM_SLEEP_SRC_NONE）和CyPmAltAct（PM_ALT_ACT_TIME_NONE、PM_ALT_ACT_SRC_NONE）。
更新了架构特定和芯片特定的#defines，以在整个内容中使用。	
提高了8051器件上非DMA配置的性能。	这些修改缩短了启动时间，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，必须针对作为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bootloader主机传输到bootloader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bootloader通信函数。	使其他通信协议（SPI、UART、...）的支持更容易。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bootloader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器件上，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

2.20 版和更早版本都是过期版本。