PSoC® Creator™

System Reference Guide
（《系统参考指南》）

cy_boot 组件 v2.40

Document Number：001-80265 Rev. **

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，2012。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品的内嵌电路之外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于可能发生运转异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统中，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC® 是赛普拉斯半导体公司的注册商标，PSoC Creator™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

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

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

产品使用可能受适用的赛普拉斯软件许可协议限制。

目录

1 简介 5
格式 6
参考 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

3 时钟 10
PSoC Creator 时钟实施 10
API 18

4 电源管理 28
时钟配置 28
唤醒时间配置 29
唤醒源配置 29
API 30
实例低功耗 API 36

5 中断 37
API 37

6 缓存 41
PSoC 3 缓存功能 41
PSoC 5 缓存功能 41

7 引脚 42
API 42

8 寄存器访问 45
API 45
9 DMA 49
10 闪存和 EEPROM 50
API 51
11 引导加载程序系统 57
引导加载程序组件 57
通信组件 58
自定义引导加载程序组件 59
引导加载程序项目类型 60
存储器使用情况 62
引导加载程序参数 64
Bootloadable Parameters（引导加载程序参数） 65
引导加载程序 API 66
引导加载程序命令 66
引导加载程序数据包 71
引导加载程序状态/错误代码 72
引导加载应用程序和代码数据文件格式 72
引导加载主机工具 73
12 系统函数 75
通用 API 75
CyDelay API 76
13 启动和连接 78
PSoC 3 78
PSoC 5 78
CMSIS 支持（适用于 PSoC 5） 79
复位状态的保留（适用于 PSoC 3 和 PSoC 5） 79
14 看门狗定时器 80
API 80
15 cy_boot 组件更改 82
2.40 版 82
2.30 版 82
2.21 版 83
2.20 版 83
2.10 版 84
2.0 版 84

1 简介

本系统参考文档介绍了 PSoC Creator cy_boot 组件所提供的函数。cy_boot 组件为项目提供系统功能，以方便其更好地获得芯片资源。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函数。

cy_boot 组件的独特之处在于：

- 自动添加到每个项目
- 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函数：

- DMA
- Flash（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]**
文件 > 新建项目	表示菜单路径： **文件 > 新建项目 > 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 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 组件作者指南
- PSoC 3 和 PSoC 5 技术参考手册 (TRM)

修订记录

文档标题：PSoC® Creator™ 系统参考指南，cy_boot 组件 v2.40 **文档编号：001-80265**		
修订版	**日期**	**变更说明**
**	05/06/2012	本文档版本号为Rev**，译自英文版 001-73816 Rev**

2 标准类型和定义

为了让带有多个编译器的多个 CPU 中支持使用相同的代码，cy_boot 组件提供了可以在多个平台之间均等使用的类型和定义。

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

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。

Define（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__)
```

Define（定义）	说明
__C51__	Keil 8051 编译器
__GNUC__	ARM GCC 编译器

Define（定义）	说明
__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。

Define（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返回 8 位无符号值类型：cystatus。标准返回值为：

Define（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

Define（定义）	说明
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)
{
      /* 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

Define（定义）	说明
CY_NOP	处理器 NOP 指令

设备版本定义

Define（定义）	说明
CY_PSOC3	任何 PSoC 3 器件
CY_PSOC5	任何 PSoC 5 器件
CY_PSOC3_ES3	PSoC 3 ES3 或更高版本
CY_PSOC3_ES2	PSoC 3 ES2

3 时钟

PSoC Creator 时钟实施

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。

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。

BUS_CLK

这是一种特殊的时钟。它与 MASTER_CLK 密切相关。在大多数设计中，MASTER_CLK 和 BUS_CLK 具有相同的频率，并且被视作相同的时钟。它们应该是系统中速度最高的时钟。CPU 会脱离 BUS_CLK 运行，并且所有外设都会使用 BUS_CLK 与 CPU 和 DMA 进行通信。同步某时钟时，该时钟将与 MASTER_CLK 同步。同步某引脚时，该引脚将与 BUS_CLK 同步。

全局时钟

这种时钟置于全局低时滞数字时钟线上。BUS_CLK 也属于这种时钟的范畴。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。这与 BUS_CLK 和 MASTER_CLK 有关。PSoC 旨在用作同步系统。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

同步时钟示例：

- 已设置“与 MASTER_CLK 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

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）
  - 固定函数模块输出（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

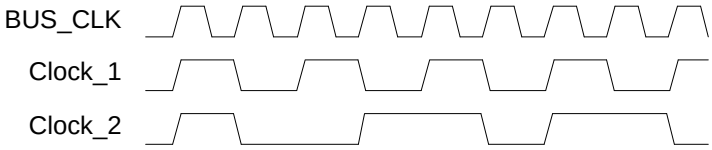
根据时钟信号源的不同，可使用以下不同方法同步信号：

- 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。

以下示例显示了已与 BUS_CLK 同步的两个时钟。Clock_1 的周期恰好是 BUS_CLK 的周期的两倍。Clock_2 的周期大约是 BUS_CLK 的周期的三倍。这导致了 BUS_CLK 的一个周期和两个周期之间的高低脉冲的宽度出现差异。在这两种情况下，所有切换均发生在 BUS_CLK 的上升沿。



路由时钟实施

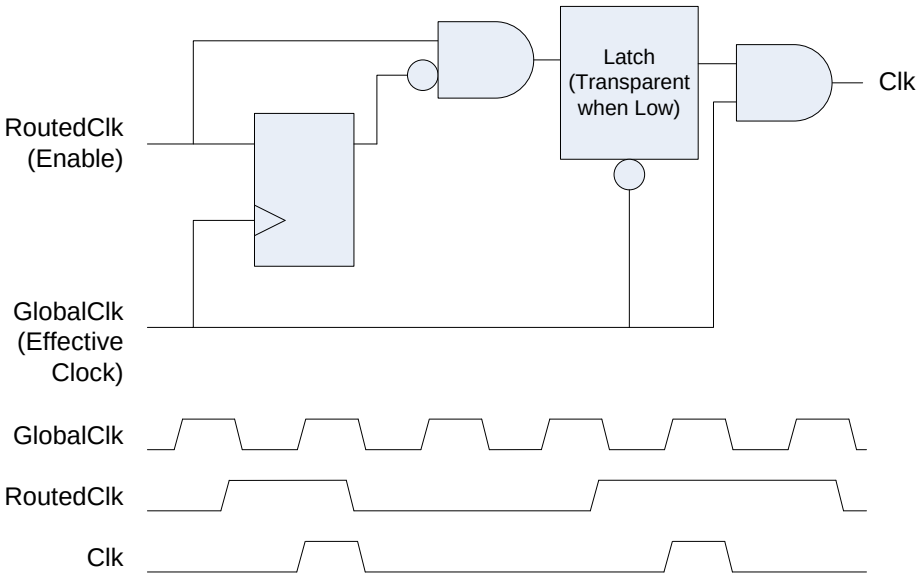
PSoC 中的时钟实施会将全局时钟信号直接连接至计时数字逻辑的时钟输入。这对于同步时钟和异步时钟均适用。因为全局时钟分布在低时滞时钟线上，所以连接至相同全局时钟的计时元素都将在相同的时间计时。

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 BUS_CLK 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。在这种情况下，通用全局时钟是 BUS_CLK。

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。



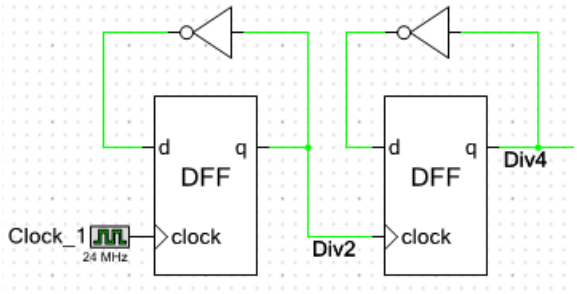
此图显示，生成的时钟在路由时钟的上升沿后于第一时钟上的全局时钟同步发生。

在分析此设计，确定路由时钟源时，可以考虑另一个转换的路由时钟。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在报告文件中报告。成功构建后，此文件位于**结果**选项卡下的“工作区浏览器”中。详细信息显示在“初始映射”标题下。每个路由时钟都将显示“有效时钟”和“使能信号”。“有效时钟”是使用的全局时钟，“使能信号”是检测到沿并用作时钟使能的路由时钟。

以分频时钟为例

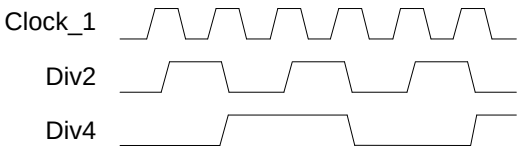
简单的分频时钟电路可用于观察该转换的完成方法。下面的电路使用全局时钟对第一个触发器 (cydff_1) 进行计时。这将生成一个二分频时钟。该信号用作路由时钟，对下一个触发器 (cydff_2) 进行计时。



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>
```

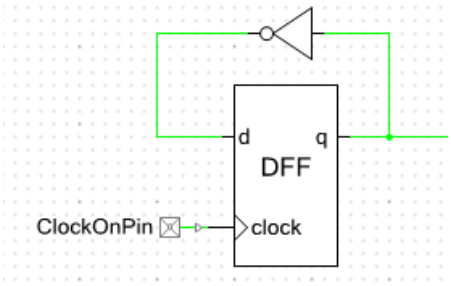
此电路生成的结果信号如下。



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处引入。因为引脚同步使用 BUS_CLK 完成，所以转换的电路将 BUS_CLK 用作有效时钟，并将引脚的上升沿用作使能信号。



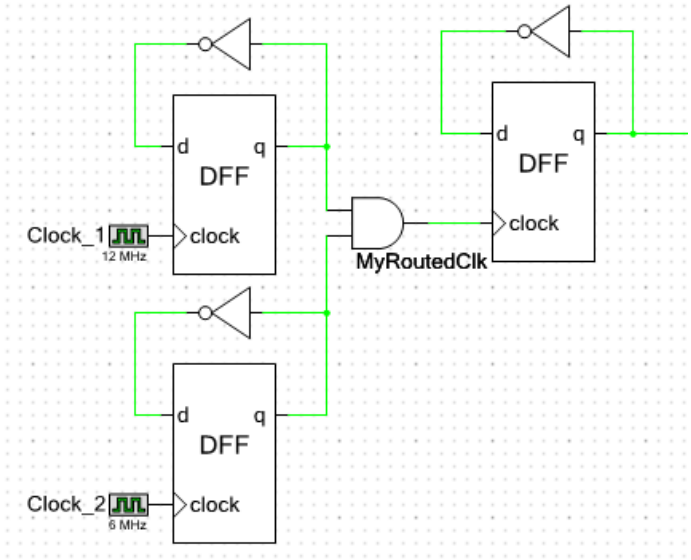

```
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
(Global Clock Selection)
<CYPRESSTAG name="UDB Routed Clock Assignment">
    Routed Clock: ClockOnPin(0):iocell.fb
        Effective Clock: BUS_CLK
        Enable Signal: ClockOnPin(0):iocell.fb
</CYPRESSTAG>
```

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。这两个时钟都是同步时钟，所以 BUS_CLK 是将成为有效时钟的通用全局时钟。

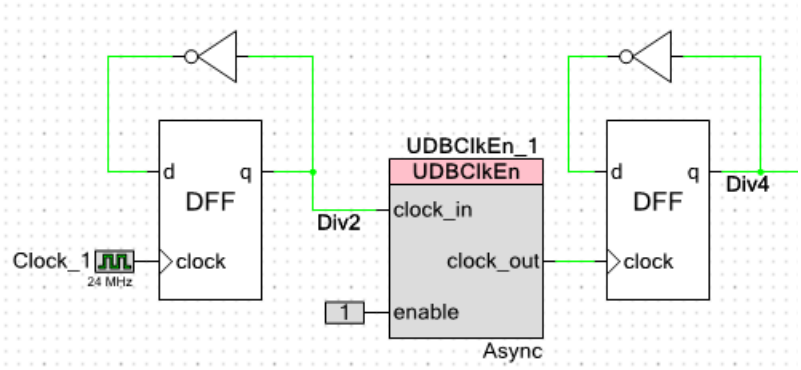


```
<CYPRESSTAG name="Tech mapping">
<CYPRESSTAG name="Initial Mapping" icon="FILE_RPT_TECHM">
<CYPRESSTAG name="Global Clock Selection" icon="FILE_RPT_TECHM">
    Digital Clock 0: Automatic-assigning  clock 'Clock_1'. Fanout=1, Signal=tmp__cydff_1_clk
    Digital Clock 1: Automatic-assigning  clock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

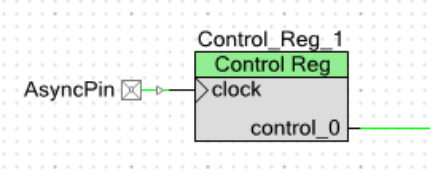
PSoC Creator 在路由时钟上执行的自动转换通常是应该使用的实施。但是，也有可强制直接使用路由时钟的方法。在异步模式中配置的 UDBClkEn 组件会将使用的时钟强制变为路由时钟，如下方电路所示。

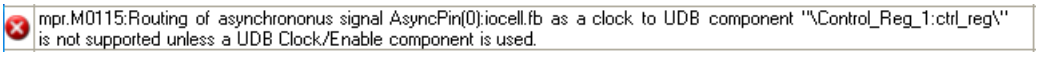


使用异步时钟

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。





如错误消息中所述，可通过在异步模式中使用 UDBClkEn 组件删除此错误。这并不会删除基础同步问题，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：两个时钟域相互不同步；两个时钟域都与 BUS_CLK 同步。

当两个时钟都与 BUS_CLK 同步时，来自较慢时钟域的信号可以在另一个时钟域中自由使用。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，必须满足的时序限制基于 BUS_CLK 的速度，而非其中任何一个时钟域的速度。

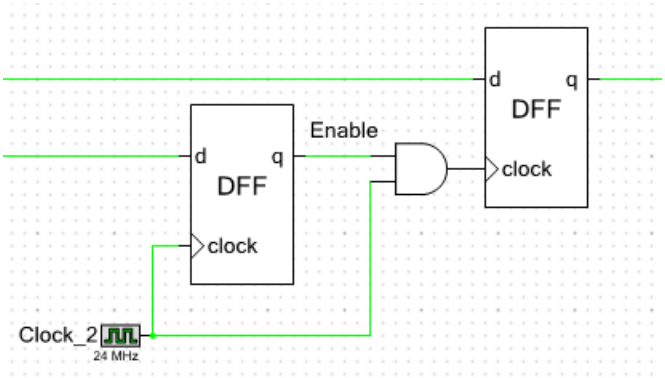
两个时钟域之间的唯一可以确定的是，它们的沿将始终发生在 BUS_CLK 的上升沿。这意味着两个时钟域的上升沿的距离和单个 BUS_CLK 循环之间的距离一样近。即使时钟域是对方的倍数仍然如此，因为它们的时钟分频器不必对齐。如果两个时钟域之间存在组合逻辑，则可能需要插入触发器，以防止限制 BUS_CLK 操作的频率。通过插入触发器，一个时钟域到另一个时钟域的跨越是一个直接的触发器到触发器路径。

当时钟域相互不关联时，必须在时钟域之间使用同步器。同步组件可用于实施同步函数。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。

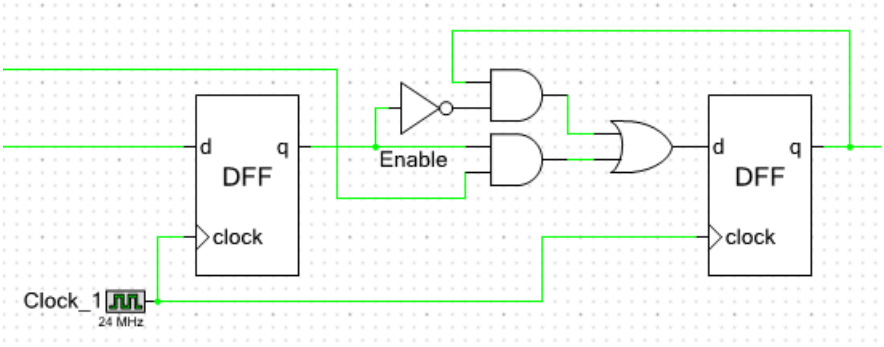
门控时钟

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。

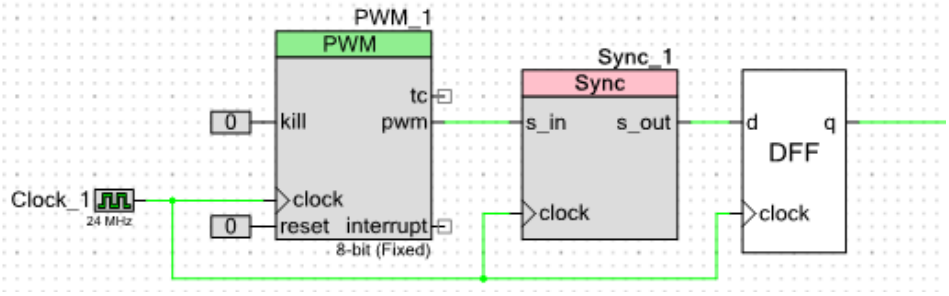
以下电路实施相同的函数，由时序分析支持，仅使用全局时钟，且没有可靠性问题。此电路不对时钟进行门控，而是逻辑启用新数据计时或逻辑维护当前数据。



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。



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： wait：

- 0：配置后立即返回
- 1：等待 PLL 锁定或超时

Return Value（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。

副作用和限制： 如果启用等待：
使用快速时轮对等待进行计时。快速时轮的其他任何使用都将在此函数周期停止，然后再恢复。
则使用 100 KHz ILO。如果未启用，该函数将在函数周期启用 100 KHz ILO。
此函数周期内中断子程序不得更改 ILO、快速时轮、中央时轮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、中央时轮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： None（无）

Return Value（返回值）： None（无）

void CyPLL_OUT_SetPQ(uint8 P, uint8 Q, 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 [4 - 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。67 MHz 或更低的输出频率建议使用 2uA，更高的输出频率建议使用 3uA。

Return Value（返回值）： None（无）

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	Define（定义）	源
0	CY_PLL_SOURCE_IMO	IMO
1	CY_PLL_SOURCE_XTAL	MHz 晶振
2	CY_PLL_SOURCE_DSI	DSI

Return Value（返回值）： None（无）

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。

Return Value（返回值）： None（无）

副作用和限制： 如果启用等待：
使用快速时轮对等待进行计时。快速时轮的其他任何使用都将在此函数周期停止，然后再恢复。
则使用 100 KHz ILO。如果未启用，该函数将在函数周期启用 100 KHz ILO。
此函数周期内中断子程序不得更改 ILO、快速时轮、中央时轮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、中央时轮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： None（无）

Return Value（返回值）： None（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。如果选择 USB 设置，则将启用 USB 时钟锁定电路。否则将禁用此电路。必须驱动 USB 模块，然后方可选择 USB 设置。如果 IMO 当前正在驱动主控时钟，则必须先正确设置闪存等待状态，然后再使用 CyFlash_SetWaitCycles() 作出这一更改。

参数： freq：IMO 操作的频率

值	Define（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	62.6 MHz
6	CY_IMO_FREQ_74MHZ	74.7 MHz (PSoC 5)
8	CY_IMO_FREQ_USB	24 MHz（已针对 USB 操作进行调整）

Return Value（返回值）： None（无）

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。如果 IMO 当前正在驱动主控时钟，则必须先正确设置闪存等待状态，然后再使用 CyFlash_SetWaitCycles() 作出这一更改。

参数： source：三个可用 IMO 输出源中的一个

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

Return Value（返回值）： None（无）

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： None（无）

Return Value（返回值）： None（无）

void CyIMO_DisableDoubler()

说明： 禁用 IMO 倍频器。

参数： None（无）

Return Value（返回值）： None（无）

void CyMasterClk_SetSource(uint8 source)

说明： 设置主控时钟的源。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。必须先正确设置闪存等待状态，然后再使用 CyFlash_SetWaitCycles() 作出这一更改。

参数： source：四个可用主控时钟源中的一个

值	Define（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

Return Value（返回值）： None（无）

void CyMasterClk_SetDivider(uint8 divider)

说明： 设置用于生成主控时钟的分频器值。必须先正确设置闪存等待状态，然后再使用 CyFlash_SetWaitCycles() 作出这一更改。

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，则此参数应设置为 1。

Return Value（返回值）： None（无）

void CyBusClk_SetDivider(uint16 divider)

说明： 设置用于生成总线时钟的分频器值。必须先正确设置闪存等待状态，然后再使用 CyFlash_SetWaitCycles() 作出这一更改。

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，则此参数应设置为 1。

Return Value（返回值）： None（无）

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，则此参数应设置为 1。

Return Value（返回值）： None（无）

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	Define（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

Return Value（返回值）： None（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： None（无）

Return Value（返回值）： None（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： None（无）

Return Value（返回值）： None（无）

void CyILO_Start100K()

说明： 启用 ILO 100 KHz 振荡器。

参数： None（无）

Return Value（返回值）： None（无）

void CyILO_Stop100K()

说明： 禁用 ILO 100 KHz 振荡器。

参数： None（无）

Return Value（返回值）： None（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： None（无）

Return Value（返回值）： None（无）

void CyILO_Disable33K()

说明： 禁用 ILO 33 KHz 分频器。请注意，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： None（无）

Return Value（返回值）： None（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	Define（定义）	源
0	CY_ILO_SOURCE_100K	ILO 100 KHz
1	CY_ILO_SOURCE_33K	ILO 33 KHz
	CY_ILO_SOURCE_1K	ILO 1 KHz

Return Value（返回值）： None（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	Define（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。

Return Value（返回值）： 先前功耗模式

void CyXTAL_32KHZ_Start()

说明： 启用 32 KHz 晶体振荡器。

参数： None（无）

Return Value（返回值）： None（无）

void CyXTAL_32KHZ_Stop()

说明： 禁用 32 KHz 晶体振荡器。

参数： None（无）

Return Value（返回值）： None（无）

uint8 CyXTAL_32KHZ_ReadStatus()

说明： 读取 32 KHz 振荡器的两个状态位。

参数： None（无）

Return Value（返回值）： 状态

值	Define（定义）	源
0x20	CY_XTAL32K_ANA_STAT	模拟测量 1：稳定 0：不稳定
0x10	CY_XTAL32K_DIG_STAT	数字测量（需要使用 33 KHz ILO 进行该测量） 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

Return Value（返回值）： 先前功耗模式

uint8 CyXTAL_Start(uint8 wait)

说明： 启用此 MHz 晶体。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。由晶振指定合适的值。

Return Value（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： 如果启用了等待（非零等待）：
使用快速时轮对等待进行计时。快速时轮的其他任何使用都将在此函数周期停止，然后再恢复。
则使用 100 KHz ILO。如果未启用，该函数将在函数周期启用 100 KHz ILO。
此函数周期内中断子程序不得更改 ILO、快速时轮、中央时轮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、中央时轮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： None（无）

Return Value（返回值）： None（无）

void CyXTAL_EnableErrStatus()

说明： 启用兆赫兹晶振的 XERR 状态位的生成。

参数： None（无）

Return Value（返回值）： None（无）

void CyXTAL_DisableErrStatus()

说明： 禁用兆赫兹晶振的 XERR 状态位的生成。

参数： None（无）

Return Value（返回值）： None（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： None（无）

Return Value（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。

参数： None（无）

Return Value（返回值）： None（无）

void CyXTAL_DisableFaultRecovery()

说明： 禁用故障恢复电路，其在兆赫兹晶振电路发生故障的情况下可切换到 IMO。

参数： None（无）

Return Value（返回值）： None（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。有关特定晶振的相应值，请参考 TRM。

Return Value（返回值）： None（无）

void CyXTAL_SetFbVoltage(uint8 setting)

说明： 仅对于 PSoC 3 ES3 器件，此函数可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]。有关特定电压设置值的映射的详细信息，请参考 TRM。

Return Value（返回值）： None（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： 仅对于 PSoC 3 ES3 器件，此函数可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]。有关特定电压设置值的映射的详细信息，请参考 TRM。

Return Value（返回值）： None（无）

副作用和限制： 反馈参考电压必须高于看门狗参考电压。

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电源模块。PSoC 3 和 PSoC 5 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。

注意 当调试器在运行时，PSoC 5 不会进入低功耗模式。

注意 对于 PSoC 3 和 PSoC 5，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。

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。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。

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 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。
- 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，以在睡眠和休眠低功耗模式下进行正确操作。对于 PSoC 3 ES3 器件，1 KHz ILO 用于测量复位后休眠/睡眠电压调节器建立时间。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设置。

警告 请勿修改此寄存器。

对于 PSoC 5 器件，需要中断以唤醒 CPU。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。有关更多信息，请参见下节。

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：中央时轮 (CTW)、快速定时器轮 (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 和数据表。

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（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

对于 PSoC 3 ES3 器件，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。对于 PSoC 5 和 PSoC 3 ES2 芯片，唤醒源不可选。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 芯片，仅支持 CTW 作为睡眠模式的唤醒源。

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。

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： None（无）

Return Value（返回值）： None（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。

PSoC 3 ES3： 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。保持关系超时后不验证其是否就绪。

参数： None（无）

Return Value（返回值）： None（无）

void CyPmAltAct(uint16 wakeupTime, uint16 wakeupSource)

说明： 将部件置于“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，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 SleepTimer 或 RTC 组件），则将 wakeupTime 指定为 NONE 并为 wakeupSource 提供合适的源。

PSoC 5：两个参数都不用于 PSoC 5。器件将进入“备用活动”模式，直至发生了已启用的中断。

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

值	Define（定义）	时间
0	PM_ALT_ACT_TIME_NONE	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

CyPmAltAct（续）

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

值	Define（定义）	时间
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 ES2，值的有效范围为 1 至 32。对于 PSoC 3 ES3，值的有效范围为 1 至 256。

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	Define（定义）	源
0	PM_ALT_ACT_SRC_NONE	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
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 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、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。

Return Value（返回值）： None（无）

副作用和限制： 对于 PSoC 3 ES2 和 PSoC 5 芯片，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： 将部件置于“睡眠”状态。

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“睡眠”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，此时定时器为启用状态且中断为禁用状态。如果已经为唤醒配置了 CTW 或 One PPS（例如，使用 SleepTimer 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。器件将进入睡眠模式，直至其被基于中央时轮 (CTW) 的中断唤醒。必须已配置 CTW 以生成中断。使用 SleepTimer 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 8 ms。4 ms、8 ms 或 16 ms 的 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。

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

值	Define（定义）	时间
0	PM_SLEEP_TIME_NONE	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

CyPmSleep（续）

参数： wakeupTime（续）

值	Define（定义）	时间
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	Define（定义）	源
0	PM_SLEEP_SRC_NONE	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
2048*	PM_SLEEP_SRC_CTW	中央时轮
2048*	PM_SLEEP_SRC_ONE_PPS	One PPS
409	PM_SLEEP_SRC_LCD	LCD

注 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。

Return Value（返回值）： None（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。

PSoC 3 ES2 芯片存在缺陷，如果将器件置于低功耗模式，会造成连接至多个模拟资源时不可靠。有关详细信息，请参见芯片勘误表。

应针对 PSoC3 ES3 芯片启用 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。

void CyPmHibernate()

说明： 将部件置于“休眠”状态。

PSoC 3：切换到休眠状态之前，保存 PICU 唤醒源位的当前状态，然后设置状态。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 行为。在 TRM 中，此寄存器为“PICU[0..15]_INTTYPE[0..7]”。在引脚组件数据表中，此寄存器也称为 IRQ 选项。一旦发生唤醒，将恢复 PICU 唤醒源位，并且 PSoC 返回活动状态。

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： None（无）

Return Value（返回值）： None（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。利用 20 µs，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。没有符合该要求的硬件检查。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。

PSoC 3 ES2 芯片存在缺陷，如果将器件置于低功耗模式，会造成连接至多个模拟资源时不可靠。有关详细信息，请参见芯片勘误表。

应针对 PSoC3 ES3 芯片启用 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的中央时轮和快速时轮定时器的中断状态。该硬件寄存器在读取后清除。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	Define（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

Return Value（返回值）： 状态。与用于掩码参数的枚举位的值相同。

实例低功耗 API

大多数组件具有特定于实例的低功耗 API 集，可使用这些 API 将组件置于低功耗状态（“睡眠”或“休眠”）。下文大致列出了这些函数。如果适用，有关寄存器保留信息的具体信息，请参考各数据手册。

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。
在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 _Sleep() 函数。

参数： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（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： None（无）

Return Value（返回值）： None（无）

Side Effects（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： None（无）

Return Value（返回值）： None（无）

Side Effects（副作用）： None（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： None（无）

Return Value（返回值）： None（无）

Side Effects（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。有关中断的更多信息，另请参见中断组件数据表。

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： 禁用为每个中断启用的中断。

参数： None（无）

Return Value（返回值）： 之前启用中断的 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

Return Value（返回值）： None（无）

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

void CyIntEnable(uint8 number)

说明： 启用指定的中断编号。

参数： number：中断编号。Valid range（有效范围）：[0-31]

Return Value（返回值）： None（无）

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

void CyIntDisable(uint8 number)

说明： 禁用指定的中断编号。

参数： number：中断编号。Valid range（有效范围）：[0-31]

Return Value（返回值）： None（无）

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

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。Valid range（有效范围）：[0-31]

Return Value（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。Valid range（有效范围）：[0-31]

address：中断服务子程序指针

Return Value（返回值）： 上一个中断矢量值

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。Valid range（有效范围）：[0-31]

Return Value（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。它设置指定异常的中断矢量。ARM 架构中的这些异常的操作类似于用户中断，但由处理器的系统架构指定。各个异常的编号是固定的。注意，这些异常的编号与用于用户中断的编号是不同的。

参数： number：异常编号。Valid range（有效范围）：[0-15]。

address：中断服务子程序指针

Return Value（返回值）： 上一个中断矢量值

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。它设置指定异常的中断矢量。ARM 架构中的这些异常的操作类似于用户中断，但由处理器的系统架构指定。各个异常的编号是固定的。注意，这些异常的编号与用于用户中断的编号是不同的。

参数： number：异常编号。Valid range（有效范围）：[0-15]。

Return Value（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。Valid range（有效范围）：[0-31]

priority（优先级）：中断优先级。0 的优先级最高。Valid range（有效范围）：[0-7]

Return Value（返回值）： None（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。Valid range（有效范围）：[0-31]

Return Value（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。Valid range（有效范围）：[0-31]

Return Value（返回值）： None（无）

void CyIntClearPending(uint8 number)

说明： 清除指定中断编号的所有待处理中断。

参数： number：中断编号。Valid range（有效范围）：[0-31]

Return Value（返回值）： None（无）

6 缓存

PSoC 3 缓存功能

默认启用 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 5 缓存功能

void CyFlushCache()

说明： 通过使所有条目失效清理 PSoC 5 缓存。

参数： None（无）

Return Value（返回值）： None（无）

7 引脚

除了提供作为引脚组件一部分的引脚功能，*cypins.h* 文件中还提供了引脚宏的库。这些宏全部利用端口引脚配置寄存器，该寄存器可用于器件上的每个引脚。该寄存器的地址在 *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 是端口编号，y 是端口内部的引脚编号。

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)

Return Value（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)

Return Value（返回值）： None（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)

Return Value（返回值）： None（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

Define（定义）	源
PIN_DM_ALG_HIZ	模拟 HiZ
PIN_DM_DIG_HIZ	数字 HiZ
PIN_DM_RES_UP	电阻上拉
PIN_DM_RES_DWN	电阻下拉
PIN_DM_OD_LO	开漏驱低
PIN_DM_OD_HI	开漏驱高
PIN_DM_STRONG	强 CMOS 输出
PIN_DM_RES_UPDWN	电阻上拉/下拉

Return Value（返回值）： None（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)

Return Value（返回值）： 引脚的当前驱动模式

Define（定义）	源
PIN_DM_ALG_HIZ	模拟 HiZ
PIN_DM_DIG_HIZ	数字 HiZ
PIN_DM_RES_UP	电阻上拉
PIN_DM_RES_DWN	电阻下拉
PIN_DM_OD_LO	开漏驱低
PIN_DM_OD_HI	开漏驱高
PIN_DM_STRONG	强 CMOS 输出
PIN_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)

Return Value（返回值）： None（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)

Return Value（返回值）： None（无）

8 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.h*、*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 5 处理器结构使用小端模式排列。

PSoC 3 和 PSoC 5 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，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 将使用正确的 8 位、16 位和 32 位访问权限进行访问。PSoC 5 不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 5)

Return Value（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 5)

value：要写入的值

Return Value（返回值）： None（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 5)

Return Value（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 5)

value：要写入的值

Return Value（返回值）： None（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 5)

Return Value（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 5)

value：要写入的值

Return Value（返回值）： None（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 5)

Return Value（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 5)

value：要写入的值

Return Value（返回值）： None（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 5 的 CY_GET_REG8 功能相同。

参数： reg：寄存器地址

Return Value（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 5 的 CY_SET_REG8 功能相同。

参数： reg：寄存器地址

value：要写入的值

Return Value（返回值）： None（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 5 的 CY_GET_REG16 功能相同。

参数： reg：寄存器地址

Return Value（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 5 的 CY_SET_REG16 功能相同。

参数： reg：寄存器地址

value：要写入的值

Return Value（返回值）： None（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 5 的 CY_GET_REG24 功能相同。

参数： reg：寄存器地址

Return Value（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 5 的 CY_SET_REG24 功能相同。

参数： reg：寄存器地址

value： 要写入的值

Return Value（返回值）： None（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 5 的 CY_GET_REG32 功能相同。

参数： reg：寄存器地址

Return Value（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 5 的 CY_SET_REG32 功能相同。

参数： reg：寄存器地址

value：要写入的值

Return Value（返回值）： None（无）

9 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 组件数据表。

10 闪存和 EEPROM

可使用一组普通的函数对闪存和 EEPROM 编程。有关更多信息，请参考 EEPROM 组件数据表。

闪存和 EEPROM 通过系统性能控制器 (SPC) 调用编程。特定于闪存/EEPROM 的 API 进行了简化处理。

只有 PSoC 3 部件有错误校正代码 (ECC) 存储。这可配置为真正的 ECC 或用于数据存储。ECC 存储器作为数据存储的最常见用途是存储直接受 PSoC Creator 支持的配置数据。

在禁用 ECC 的情况下，在对闪存编程时（可用于数据存储），有多种方法可写入数据行：

- 整行包括 ECC
- 整行不包括 ECC
- 仅 ECC 存储器。

如果 ECC 存储器用于存储配置数据，则确保您覆盖用于配置数据的 ECC 存储器区域。

调用程序必须首先调用 CySetTemp 和 CySetFlashEEBuffer 函数。需要合适的温度调整写入闪存的时间以实现最佳性能。在与 SPC 通信时，缓冲区用来存储中间数据。该 SPC 为推挽式，带有一个可以向其发送命令并从其读回数据的寄存器。

闪存或 EEPROM 可通过调用同一函数“CyWriteRowData”每次写入到一行。第一个参数将确定闪存或 EEPROM 阵列。闪存的阵列数和 EEPROM 的阵列数特定于所选确切部件。检查您的部件以了解哪些阵列 ID 是有效的。

闪存阵列最多有 64 KB 和 ECC 字节之和。PSoC 3 结构有一个闪存阵列，大小为 16 KB、32 KB 或 64 KB 和 ECC 字节之和。因此，唯一的有效阵列 ID 为 0x00。

EEPROM 阵列最大为 2 KB。PSoC 3 和 PSoC 5 器件有一个 EEPROM 阵列，大小为 512 字节、1 KB 或 2 KB。

API

cystatus CySetTemp()

说明： 获取 die 的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： None（无）

Return Value（返回值）： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	已使用闪存/EEPROM 写入
CYRET_UNKNOWN	失败

副作用和限制： 执行该函数需要较长时间。
函数不返回，直至 SPC 返回到空闲状态。

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：分配的大小为 (SIZEOF_FLASH_ROW + SIZEOF_ECC_ROW) 字节的缓冲区。

Return Value（返回值）： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	已使用闪存/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。如果是闪存阵列并且 ECC 正用于配置存储，则函数期望行数据和 ECC 数据都已作为 rowData 的一部分提供。

参数： uint8 arrayId：要写入的阵列的 ID。写入的类型（闪存或 EEPROM）由阵列 ID 决定。部件中的阵列是连续的，从特定存储器类型的第一个 ID 开始。

类型	第一个 ID	阵列大小
Flash（闪存）	FIRST_FLASH_ARRAYID	64 K 字节
EEPROM	FIRST_EE_ARRAYID	2 K 字节

uint16 rowAddress：指定 arrayId 中的行地址。

类型	每个阵列的行数	行大小（字节）
闪存（ECC 已启用）	256	SIZEOF_FLASH_ROW (256)
闪存（ECC 已禁用）	288	SIZEOF_FLASH_ROW + SIZEOF_ECC_ROW (288)
EEPROM	128	SIZEOF_EEPROM_ROW (16)

uint8 *rowData：要编程的数据的地址。该行的大小为 SIZEOF_FLASH_ROW 或 SIZEOF_EEPROM_ROW（取决于“arrayId”）。

注 这不能是作为 SPC 传递的同一缓冲区

uint16 rowSize：行数据的字节数

Return Value（返回值）： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	已使用闪存/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。对于闪存，如果 ECC 启用，则 ECC 将自动写入。如果 ECC 禁用，则当前 ECC 存储器的内容将保留。

参数： uint8 arrayId：要写入的阵列的 ID。写入的类型（闪存或 EEPROM）由阵列 ID 决定。部件中的阵列是连续的，从特定存储器类型的第一个 ID 开始。

类型	第一个 ID	阵列大小
Flash（闪存）	FIRST_FLASH_ARRAYID	64K 字节
EEPROM	FIRST_EE_ARRAYID	2K 字节

uint16 rowAddress：指定 arrayId 中的行地址。

类型	每个阵列的行数	行大小（字节）
Flash（闪存）	256	SIZEOF_FLASH_ROW (256)
EEPROM	128	SIZEOF_EEPROM_ROW (16)

uint8 *rowData 要写入的字节阵列。

Return Value（返回值）： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	已使用闪存/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： 写入闪存行的 ECC 部分。该函数仅在 ECC 禁用时使用，且不用于存储配置数据。该函数仅对闪存阵列 ID 有效（对 EEPROM 无效）。

参数： arrayId：要写入的阵列的 ID。部件中的阵列是连续的，从特定存储器类型的第一个 ID 开始。

类型	第一个 ID	ECC 阵列大小
Flash（闪存）	FIRST_FLASH_ARRAYID	8K 字节

rowAddress：指定 arrayId 中的行地址。

类型	每个阵列的行数	ECC 的行大小（字节）
Flash（闪存）	256	SIZEOF_ECC_ROW (32)

rowData：要写入的字节阵列。

Return Value（返回值）： 状态

值	说明
CYRET_SUCCESS	成功
CYRET_LOCKED	已使用闪存/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。默认情况下闪存为启用状态。
对于 PSoC 3 ES2、更早的版本和 PSoC 5，相同的位同时控制 EEPROM 和闪存。启动或终止都会导致二者进行相应的操作。

参数： None（无）

Return Value（返回值）： None（无）

void CyFlash_Stop()

说明： 禁用闪存。只要 CPU 当前正在运行，该设置就会被忽略。该设置仅在 CPU 禁用后生效。
对于 PSoC 3 ES2、更早的版本和 PSoC 5，相同的位同时控制 EEPROM 和闪存。启动或终止都会导致二者进行相应的操作。

参数： None（无）

Return Value（返回值）： None（无）

void CyFlash_SetWaitCycles(uint8 freq)

说明： 基于器件运行频率设置正确的闪存等待周期数。该函数应在增加时钟频率之前调用。可在降低时钟频率后选择性调用此函数，以提高 CPU 性能。

参数： freq：操作频率（以兆赫兹为单位）

Return Value（返回值）： None（无）

void CyEEPROM_Start()

说明： 启用 EEPROM。
对于 PSoC 3 ES2、更早的版本和 PSoC 5，相同的位同时控制 EEPROM 和闪存。启动或终止都会导致二者进行相应的操作。另外，对于这些芯片版本，EEPROM 在默认情况下是启用的。对于更高版本的芯片，EEPROM 由单独的位控制，在使用之前必须先启动。

参数： None（无）

Return Value（返回值）： None（无）

void CyEEPROM_Stop()

说明： 禁用 EEPROM。
对于 PSoC 3 ES2、更早的版本和 PSoC 5，相同的位同时控制 EEPROM 和闪存。启动或终止都会导致二者进行相应的操作。

参数： None（无）

Return Value（返回值）： None（无）

void CyEEPROM_ReadReserve()

说明： 请求访问 EEPROM 进行读取，并等待获取访问权限。对 EEPROM 的访问在写入 EEPROM 的控制器和对 EEPROM 执行常规读取访问之间进行仲裁。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却尝试进行读取，则会产生错误并返回错误数据。

参数： None（无）

Return Value（返回值）： None（无）

void CyEEPROM_ReadRelease()

说明： 释放对 EEPROM 的读取保留。如果 EEPROM 已保留用于读取，那么必须对其进行释放，才能进一步对 EEPROM 执行写入操作。

参数： None（无）

Return Value（返回值）： None（无）

11 引导加载程序系统

在 PSoC Creator 中，引导加载程序系统管理使用新应用代码和/或数据（“代码”）对器件闪存进行更新的处理。这可以通过以下部件来完成：

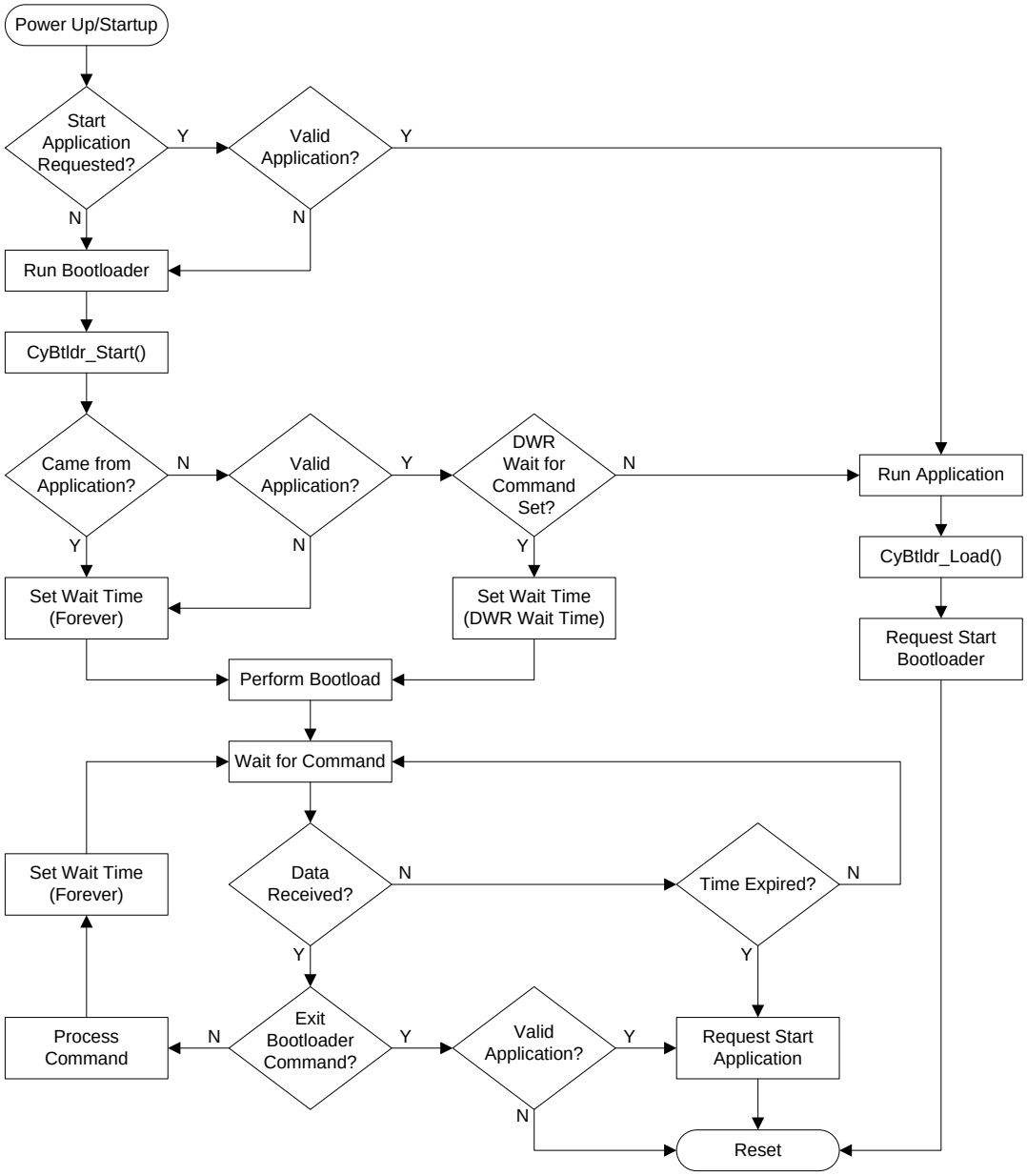
- 引导加载程序组件
- 通信组件
- 引导加载程序项目，用于创建引导加载程序组件
- 可引导加载项目，用于创建代码

以下章节对引导加载程序处理的各个方面进行了详细介绍。

引导加载程序组件

引导加载程序组件允许您使用新代码更新器件闪存。引导加载程序接受并执行命令，然后将这些命令的响应传递回通信组件。引导加载程序收集并整理接收到的数据，并通过一个简单的命令/状态寄存接口管理对闪存的实际写入操作。引导加载程序组件不是典型的组件。它不在“组件目录”中。相反，如果您有一个引导加载程序类型的项目，则该组件始终存在于后台中。

下图显示了引导加载程序的工作原理。



通信组件

通信组件管理通信协议以从外部系统接收命令，然后将这些命令传递到引导加载程序。它还将引导加载程序的命令响应传递回片外系统。

请注意：引导加载程序仅正式支持 USB 和 I^2C 这两种通信方法。有关适当通信方法的详情，请根据需要参考 USBFS 或 I^2C 组件数据手册。还可使用“Custom”（自定义）选项添加对任何现有通信组件的引导加载程序支持。请参见“自定义引导加载程序组件”。

您还可以为任意数量的通信方法创建自己的引导加载程序组件。有关如何执行此操作的信息和指令，请参考《组件作者指南》(*Component Author Guide*)。

自定义引导加载程序组件

可使用任何需要的通信组件创建自定义引导加载程序组件。在 DWR 系统编辑器“IO Component”（IO 组件）下，选择“Custom_Interface”选项。另请参见“IO 组件”（IO 组件）参数。这可以在您的应用代码中以任何必要的方式实现所需的函数。以下示例是针对 SPI 通信组件将代码插入到 *main.c* 文件中。有关 CyBtldr_API 的更多信息，请参见*组件作者指南*。

```
void CyBtldrCommStart(void)
{
    SPIS_1_Start();
}

void CyBtldrCommStop (void)
{
    SPIS_1_Stop();
}

void CyBtldrCommReset(void)
{
}

cystatus CyBtldrCommWrite(uint8* buffer, uint16 size, uint16* count, uint8
timeOut)
{
    uint16 i;
    cystatus status = CYRET_EMPTY;

    uint8 intStatus = CyEnterCriticalSection();

    SPIS_1_ClearRxBuffer();
    SPIS_1_ClearTxBuffer();

    for (i = 0; i < size; i++)
    {
        SPIS_1_WriteTxData(buffer[i]);
    }

    CyExitCriticalSection(intStatus);

    while (timeOut-- > 0)
    {
        if ((SPIS_1_ReadTxStatus() & SPIS_1_STS_SPI_DONE) !=
           SPIS_1_STS_SPI_DONE)
        {
            *count = size;
            status = CYRET_SUCCESS;
            break;
        }

        CyDelay(10);
    }

    return status;
```

```
}

cystatus CyBtldrCommRead (uint8* buffer, uint16 size, uint16* count, uint8
timeOut)
{
    uint16 i = 0;
    cystatus status = CYRET_EMPTY;

    uint8 validData = 0;
    uint8 dataByte;

    while (timeOut-- > 0)
    {
        if (SPIS_1_GetRxBufferSize() > 0 && SPIS_1_GetTxBufferSize() == 0)
        {
            while (!validData && SPIS_1_GetRxBufferSize() > 0)
            {
                dataByte = SPIS_1_ReadRxData();

                validData = (1 == dataByte);
            }

            if (validData)
            {
                buffer[0] = dataByte;
                i = 1;
            }

            CyDelay(10);
            while (SPIS_1_GetRxBufferSize() > 0)
            {
                buffer[i++] = SPIS_1_ReadRxData();
            }

            if (i > 0)
            {
                while(buffer[--i] != 0x17);
                *count = i+1;
                status = CYRET_SUCCESS;
                break;
            }
        }
        CyDelay(10);
    }
    return status;
}
```

引导加载程序项目类型

为了同时实施引导加载程序组件和代码，您必须创建特殊的 PSoC Creator 项目类型：引导加载程序项目和可引导加载项目。

引导加载程序项目

引导加载程序组件仅存在于类型为“引导加载程序”的 PSoC Creator 设计项目中。创建引导加载程序项目时，引导加载程序组件自动在项目中生成。

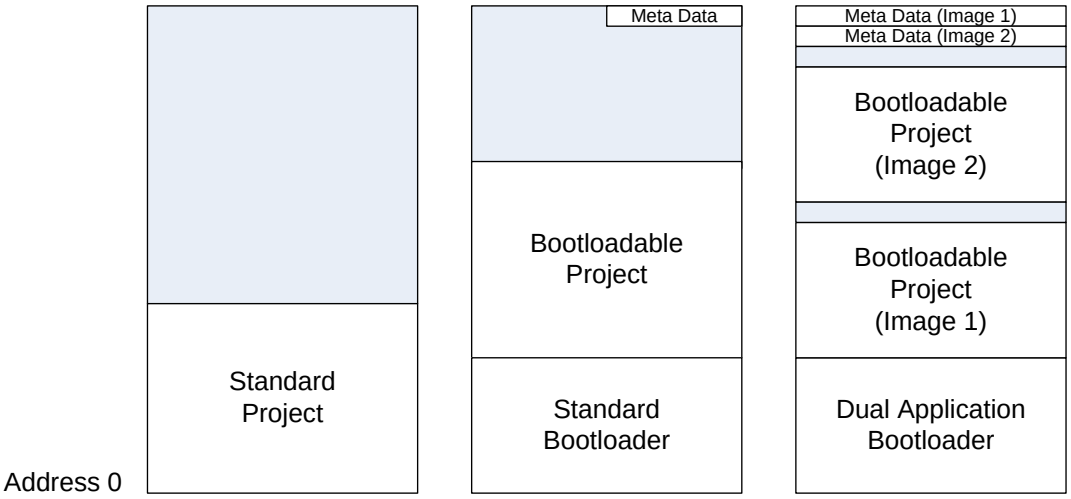
有两种类型的引导加载程序可用：“Standard Bootloader”（标准引导加载程序）和“Dual Application Bootloader”（双应用引导加载程序）。“Standard Bootloader”（标准引导加载程序）选项允许使用单个应用代码，而“Dual Application Bootloader”（双应用引导加载程序）允许闪存中有两个应用。对于要保证始终有可运行的有效应用的设计，“Dual Application Bootloader”（双应用引导加载程序）是非常有用的。此保证有一个限制，即各个应用拥有“标准引导加载程序”项目可用闪存的一半闪存。

通常，您可以通过将通信组件拖到原理图上、将 I/O 路由至引脚、设置时钟等操作来完成引导加载程序设计项目。带有通信组件的引导加载程序项目通过接收新代码并将其写入闪存来实施基本的引导加载程序函数。您可以通过将其他组件拖到原理图上或添加源代码来向基本的引导加载程序项目添加自定义函数。

可引导加载项目

可引导加载项目实际上就是代码。它非常类似于“标准”设计项目；在设计阶段，项目类型很容易在这两者之间发生改变。主要区别在于：可引导加载项目始终与引导加载程序项目相关联，而标准项目却从来不会与引导加载程序项目相关联。

标准项目驻留在以地址 0 开始的闪存中。可引导加载项目占据地址为 0 以上的闪存；关联的引导加载程序项目占据以地址 0 开始的闪存，如下所示：



引导加载程序和可引导加载项目的函数

引导加载程序项目通过引导加载程序项目的通信组件可将引导加载项目（或新代码）整体传输到闪存中。传输后，会始终对处理器进行复位。引导加载程序项目还负责在复位时测试特定条件，并有可能在可引导加载项目不存在或损坏的情况下自动启动传输操作。

启动时，引导加载程序代码加载其自己配置的配置字节。它还必须对堆栈、其他资源及外设进行初始化，然后才能执行传输。传输完成后，控件通过软件复位传递到可引导加载项目中。

然后，可引导加载项目加载其自己配置的配置字节，并对堆栈及其函数的其他资源和外设重新进行初始化。可引导加载项目会调用引导加载程序项目中的 CyBtldr_Load() 函数来启动传输操作（该操作会再次造成软件复位）。

PSoC Creator 项目输出文件

在创建引导加载程序项目或可引导加载项目时，都会为该项目创建输出文件。

此外，在构建可引导加载项目时，还会为这两种项目创建一个输出文件，即“组合”文件。该文件同时包含引导加载程序项目和可引导加载项目。它通常用于方便在生产环境中将两种项目下载（通过 JTAG/SWD）到器件闪存中。

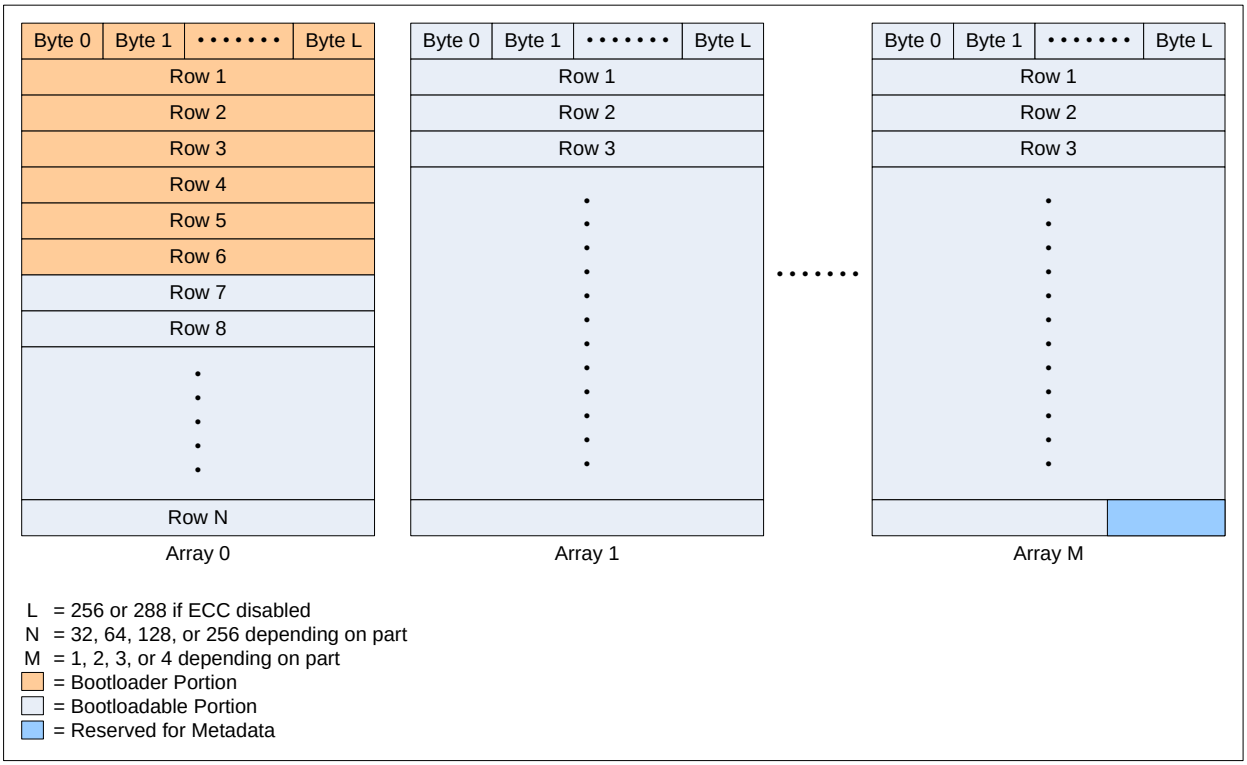
对于引导加载程序项目，配置字节始终存储在引导加载程序占据的主闪存中，从来不会在 ECC 闪存中。

可引导加载项目的配置字节存储在主闪存或 ECC 闪存中。可引导加载项目输出文件的格式有以下特点：当器件具有已被禁用的 ECC 字节时，执行传输操作的时间将减少。该操作通过交错可引导加载的主闪存地址空间中的记录以及 ECC 闪存地址空间中的记录来完成。引导加载程序通过对关联的闪存行进行一次编程来利用此交错的结构，该闪存行同时包含主闪存和 ECC 闪存的字节。

每个项目都有其自己的校验和。在项目构建期间，校验和包含在输出文件中。

存储器使用情况

下图显示 PSoC 3 和 PSoC 5 中器件的闪存布局。



引导加载程序项目始终占据闪存底部的 N 256 字节模块。对 N 进行如此设置，从而以下内容具有足够的闪存：

- 该项目的矢量表，以地址 0 开始（除了 PSoC 3），
- 引导加载程序项目配置字节，
- 引导加载程序项目代码和数据，
- 闪存引导加载程序部分的校验和。

请注意，引导加载程序项目的配置字节始终存储在主闪存中，从来不会在 ECC 闪存中。相关选项会从项目的.cydwr 文件中删除。

闪存的引导加载程序部分将受到保护；只有通过 JTAG / SWD 进行下载才能将其覆盖。

可引导加载项目紧随引导加载程序，占据从首个 256 字节边界开始的闪存，包含以下内容：

- 该项目的矢量表（除了 PSoC 3），
- 可引导加载项目的代码和数据，
- 64 字节数据保留在最后一个闪存阵列的末尾，以存储由引导加载程序和可引导加载的项目共同使用的元数据。

可引导加载项目的配置字节和标准项目中配置字节的存储方式相同，即存储在主闪存或 ECC 闪存中，具体取决于项目.cydwr 文件中的设置。

闪存最高 64 位模块用作两种项目的公共区域。该模块存储了各种参数，包括：

- 可引导加载项目在闪存中的条目（4 字节地址）
- 可引导加载项目占据的闪存量（闪存行的数量）
- 闪存可引导加载部分的校验和（1 字节）
- 闪存可引导加载部分的大小（4 字节，以字节为单位）

8051 详细信息 (PSoC 3)

在 PSoC 3 中，唯一“异常矢量”是地址 0 处的 3 字节指令，在处理器复位时执行。（中断矢量不在闪存中，而是由中断控制器 [IC] 提供）。因此，复位时，8051 引导加载程序代码仅从闪存地址 0 开始执行。

ARM Cortex-M3 详细信息 (PSoC 5)

在 PSoC 5 中，地址 0 处必须存在异常矢量表。（该表位于矢量表偏移寄存器指向的地址 0xE000ED08，其值在复位时设置为 0。） 引导加载程序代码紧跟着该表之后。

该表包含引导加载程序项目的初始堆栈指针 (SP) 值以及引导加载程序项目代码的起始地址。它还包含用于引导加载程序的异常矢量和中断矢量。

可引导加载项目还有其自己的矢量表，其中包含项目的起始 SP 值和第一个指令地址。传输完成后，作为将控件传递到可引导加载项目的一部分，矢量表偏移寄存器中的值更改为可引导加载项目表的地址。

引导加载程序参数

要访问引导加载程序参数，请打开 DWR 系统编辑器，然后展开编辑器的引导加载程序部分。

等待命令

说明： 复位时，如果引导加载程序检测到可引导加载项目闪存中的校验和有效，则会在跳转到可引导加载项目代码之前选择性等待命令，以启动传输操作。

设置： 是否等待。

默认： 是

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 如果选择“是”，则“等待命令的时间”参数可编辑。如果选择“否”，则“等待命令的时间”参数变为灰色。在这种情况下，外部系统通常无法启动传输操作，但可引导加载项目代码仍可以通过调用 Bootloader_Start() 来启动传输操作。

等待命令的时间

说明： 复位时，如果引导加载程序检测到可引导加载项目闪存中的校验和有效，则会在跳转到可引导加载项目代码之前选择性等待命令，以启动传输操作。该参数表示等待的超时时间。

设置： 1 — 255，以 10 毫秒为单位。

默认： 10，或 100 毫秒。

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 如果“等待命令”参数设置为“是”，则此参数可编辑；如果“等待命令”参数设置为“否”，则此参数变为灰色。

IO 组件

说明： 这是引导加载程序用于接收命令并发送响应的通信组件。必须选择且仅选择一个通信组件。仅使用双向通信组件。例如，UART 必须同时启用 RX 和 TX，且不可使用红外 (IrDA) 组件。存在设计规则检查 (DRC)，以免出现引导加载程序项目原理图中没有放置双向通信组件的情况。

设置： 该属性是原理图上具有引导加载程序支持的可用 IO 通信协议列表。在任何情况下，无论原理图上有何内容，都有一个“Custom”（自定义）选项可用于直接实现引导加载程序函数。

默认： 如果原理图上没有通信组件，则将选择“Custom”（自定义）选项。这可以任何方式实现通信。

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 无。

Fast Application Verification（快速应用验证）

说明： 如果启用了此参数，引导加载程序将计算应用代码的校验和。如果计算成功，将存储此信息用于未来的启动操作，以免除在每次启动时验证应用代码的需要。

设置： 是否记住验证。

默认： 无

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 无。

Checksum Type（校验和类型）

说明： 提供两个校验和类型选项，在主机和引导加载程序之间传输数据包时使用这两个选项。

设置： 基本求和，将所有字节相加，取其二进制的补码计算得到。使用 CCITT 算法的 16 位 CRC。

默认： Basic Sum（基本求和）。

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 无。

版本

说明： 提供一个 2 字节数字以代表引导加载程序的版本。

设置： 任意 2 字节数字。

默认： 0x0000

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 无。

Bootloadable Parameters（引导加载程序参数）

版本

说明： 提供一个 2 字节数字以代表引导可引导加载的应用的版本。

设置： 任意 2 字节数字。

默认： 0x0000

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 无。

Bootloadable ID（可引导加载 ID）

说明： 提供一个 2 字节数字以代表可引导加载应用的 ID。

设置： 任意 2 字节数字。

默认： 0x0000

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 无。

Custom ID（自定义 ID）

说明： 提供一个 4 字节自定义 ID 数字，以代表应用中的任意项。

设置： 任意 4 字节数字。

默认： 0x00000000

是否可由 API 或驱动程序固件修改： 无

与其他参数的关系 无。

引导加载程序 API

引导加载程序提供一个公开 API 调用，专为从可引导加载项目代码中启动传输操作使用。一旦调用，就会执行软件复位，然后引导加载程序会接管 CPU。但不会执行可引导加载项目代码（包括中断处理程序）。

传输操作开始后，资源和外设会根据需要重新配置。所有其他资源和外设都将处于禁用状态。

传输操作完成后，CPU 就会复位。

void CyBtldr_Load(void)

说明： 启动传输操作。根据引导加载程序项目配置对器件重新进行配置。

参数： void

Return Value（返回值）： 无。传输完成后处理器复位。

Side Effects（副作用）： None（无）

引导加载程序命令

引导加载程序支持以下命令。对于所有接收到的字节，如果没有以一组命令字节之一开始，则该字节会被丢弃，且不会生成响应。所有多字节字段都是输出最低有效位优先。

注意 引导加载程序执行任何命令所需的时间是以器件的设置为基础的。一些因素会影响时序，包括：

- 部件运行的时钟速度
- 用于构建项目的工具链

- 构建过程中使用的优化设置
- 后台运行的中断数

进入引导加载程序

在接收到此命令之前，所有其他命令都将被忽略。

输入

- 命令字节：0x38
- 数据字节：NA

输出

- 状态/错误代码：
  - 成功
  - 错误命令
  - 错误数据
  - 错误长度
  - 错误校验和
- 数据字节：
  - 4 字节 - 芯片 ID
  - 1 字节 - 芯片修订版
  - 3 字节 - 引导加载程序版本

获取闪存大小

使用所选闪存阵列中第一个以及最后一个可用行进行响应。

输入

- 命令字节：0x32
- 数据字节：
  - 1 字节 - 闪存阵列 ID

输出

- 状态/错误代码：
  - 成功
  - 错误命令
  - 错误数据
  - 错误长度
  - 错误校验和
- 数据字节：
  - 2 字节 - 第一个可用行
  - 2 字节 - 最后一个可用行

编程行

将一行闪存数据写入器件。

输入

- 命令字节：0x39
- 数据字节：
  - 1 字节 - 闪存阵列 ID
  - 2 字节 - 闪存行数
  - n 字节 - 要写入闪存行的数据

输出

- 状态/错误代码：
  - 成功
  - 错误命令
  - 错误数据
  - 错误长度
  - 错误校验和
  - 错误闪存行
  - 错误活动
- 数据字节：NA

擦除行

擦除所提供的闪存行的内容。

输入

- 命令字节：0x34
- 数据字节：
  - 1 字节 - 闪存阵列 ID
  - 2 字节 - 闪存行数

输出

- 状态/错误代码：
  - 成功
  - 错误命令
  - 错误数据
  - 错误长度
  - 错误校验和
  - 错误闪存行
  - 错误活动

- 数据字节：NA

验证行

获取所提供的闪存行内容的 1 字节校验和

输入

- 命令字节：0x3A
- 数据字节：
  - 1 字节 - 闪存阵列 ID
  - 2 字节 - 闪存行数

输出

- 状态/错误代码：
  - 成功
  - 错误命令
  - 错误数据
  - 错误长度
  - 错误校验和
- 数据字节：
  - 1 字节 - 行校验和

验证校验和

获取 1 字节值，用于指示闪存的校验和是否与预期的校验和值匹配。返回值 1 表示校验和匹配，且应用被视为良好。返回值 0 表示校验和不匹配，且应用无效。在尝试运行应用代码之前，引导加载程序也会运行相同检查。

输入

- 命令字节：0x31
- 数据字节：N/A

输出

- 状态/错误代码：
  - 成功
  - 错误命令
  - 错误数据
  - 错误长度
  - 错误校验和
- 数据字节：
  - 1 字节 - 应用校验和无效

发送数据

将一个数据模块发送到器件。此数据将进行缓冲，等待另一个命令，该命令会告知引导加载程序如何处理此数据。如果连续发出多个“发送数据”命令，则此数据将附在上一模块后。此命令用于将大量的传输数据分为适当大小，以防止某些协议中出现总线枯竭的情况。

输入

- 命令字节：0x37
- 数据字节：n 字节 - 要保存在器件中的数据

输出

- 状态/错误代码：
  - 成功
  - 错误命令
  - 错误数据
  - 错误长度
  - 错误校验和
- 数据字节：NA

同步引导加载程序

将引导加载程序复位至空白状态，准备接受新命令。所有缓冲的数据都将被删除。只有主机和客户端之间不再同步时才有必要使用。

输入

- 命令字节：0x35
- 数据字节：NA

输出

- NA - 该数据包被否认

退出引导加载程序

通过触发器件的软件复位退出引导加载程序。执行软件复位之前，验证可引导加载的应用。如果应用通过了验证，将在软件复位之后执行此应用。如果应用未通过验证，则将在软件复位之后再次用引导加载程序执行此应用。

输入

- 命令字节：0x3B
- 数据字节：NA

输出

- NA - 该数据包被否认

获取应用状态（仅适用于双应用引导加载程序）

输入

- 命令字节：0x33
- 数据字节：
  - 1 字节 — 应用编号

输出

- 状态/错误代码：
  - 成功
  - 错误长度
  - 错误校验和
  - 错误数据
- 数据字节：
  - 1 字节 — 应用编号有效
  - 1 字节 — 应用编号活动

设置活动的应用（仅适用于双应用引导加载程序）

输入

- 命令字节：0x36
- 数据字节：
  - 1 字节 — 应用编号

输出

- 状态/错误代码：
  - 成功
  - 错误应用
  - 错误长度
  - 错误数据
  - 错误校验和
- 数据字节：NA

引导加载程序数据包

发送到引导加载程序的数据包具有以下结构：

- 1 字节数据包开始 (0x01)
- 1 字节命令
- 2 字节数据长度
- n 字节数据

- 2 字节校验和
- 1 字节数据包结束 (0x17)

引导加载程序的数据包输出具有以下结构：

- 1 字节数据包开始 (0x01)
- 1 字节状态/错误代码
- 2 字节数据长度
- n 字节数据
- 2 字节校验和
- 1 字节数据包结束 (0x17)

引导加载程序状态/错误代码

引导加载程序中可能的状态/错误代码输出如下所示：

引导加载应用程序

- 成功 — 成功接收并执行命令。Value = 0x00
- 错误长度 (CYRET_ERR_LENGTH) — 现有数据量超出预期范围。Value = 0x03
- 错误数据 (CYRET_ERR_DATA) — 数据形式错误。Value = 0x04
- 错误命令 (CYRET_ERR_CMD) — 无法识别命令。Value = 0x05
- 错误校验和 (ERR_CHECKSUM) — 校验和与预期值不匹配。值 = 0x08
- 错误闪存行 (ERR_ROW) — 闪存行无效。值 = 0x0A
- 错误未知 (ERR_UNK) — 发生未知错误。值 = 0x0F
- 错误应用 (CYRET_ERR_APP) — 应用无效，且不可设为活动。值 = 0x0C。（仅适用于双应用引导加载程序）
- 错误活动 (CYRET_ERR_ACTIVE) — 应用当前被标记为活动状态。值 = 0x0D。（仅适用于双应用引导加载程序）

引导加载程序主机

- 错误器件 (CYRET_ERR_DEVICE) — 预期器件与检测到的器件不匹配。Value = 0x06
- 错误版本 (CYRET_ERR_VERSION) — 检测到的引导加载程序版本不受支持。Value = 0x07

引导加载应用程序和代码数据文件格式

引导加载应用程序和代码数据 (.cyacd) 文件格式用于存储设计中可引导加载的部分。文件包括后面接有若干行闪存数据的标题。除标题以外，.cyacd 文件中的每一行均表示闪存数据的一整行。数据以 Big Endian 的格式存储为 ASCII 数据。

标题记录格式为：

```
[4-byte SiliconID][1-byte SiliconRev][1-byte Checksum Type]
```

数据记录格式为：

```
[1-byte ArrayID][2-byte RowNumber][2-byte DataLength][N-byte Data]
[1-byte Checksum]
```

标题中的校验和类型指示用于在引导加载程序主机和引导加载程序自身之间发送的数据包的校验和类型。数据记录中的校验和是基本求和，通过对所有字节求和（不包括校验和自身），然后取其二进制补码计算得到的。

引导加载主机工具

PSoC Creator 附带有引导加载主机工具 (*bootloader_host.exe*)，可用于对在 PSoC 芯片上运行的引导加载程序进行充分测试。引导加载主机工具是一种应用程序，它与引导加载程序自身进行通信，以发送可引导加载的新图像。所提供的引导加载主机工具仅拟用作开发和测试工具。

源代码

除本身可执行的主机外，还提供了所使用的大部分源代码。该源代码可重新用于创建您自己的引导加载主机应用程序。源代码位于下列目录中：

<Install Dir>\cybootloaderutils

默认情况下，此目录为：

C:\Program Files\Cypress\PSoC Creator\<Release Version>\PSoC Creator\cybootloaderutils

该源代码分为四个不同的模块。这些模块用于实现引导加载主机所需的各种功能。根据所需的控制级别，部分或所有这些模块可用于开发自定义引导加载主机应用程序。

cybtldr_command.c/h

该模块用于构建发送至引导加载程序的数据包，并解析从引导加载程序接收到的数据包。该模块具有构建引导加载程序能够理解的各类数据包的 1 次函数，并具有解析引导加载程序返回结果的 1 次函数。

cybtldr_parse.c/h

该模块用于解析 *.cyacd 文件，该文件包含要发送至器件的可引导加载图像。该模块具有用于设置文件访问、读取标题、读取行数据和关闭文件的函数。

cybtldr_api.c/h

该模块是行级 API，允许每次向使用所提供的通信机制的引导加载程序发送单行数据。该模块具有用于设置引导加载操作、编程行、擦除行、验证行和结束引导加载操作的函数。

cybtldr_api2.c/h

该模块是更高级别的 API，处理整个引导加载过程。该模块具有用于编程器件、擦除器件、验证器件和终止当前操作的函数。

版本

以下是不同版本的引导加载主机工具具备的功能：

1.0.0 版 (PSoC Creator 1.0, Beta5)

最初版本；提供具有以下功能的 API：

- 解析 *.cyacd 文件
- 构建发送至引导加载程序的数据包
- 具有用于编程、擦除和验证数据行的函数
- 具有用于执行整个引导加载操作的函数

1.1.0 版 (PSoC Creator 1.0, Production)

提供引导加载程序主机 1.0.0 版中包含的所有功能。支持使用基本求和（用于 1.0.0）或新的 16 位 CCITT 校验和，以在与引导加载程序通信时，确保数据包的完整性。

1.2.0 版本 (PSoC Creator 2.0)

提供引导加载程序主机 1.1.0 版中包含的所有功能。添加对用“Standard Bootloader”（标准引导加载程序）或“Dual Application Bootloader”（双应用引导加载程序）进行通信的支持。

12 系统函数

这些函数适用于所有结构。

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（实际值取决于器件是 PSoC 3 还是 PSoC 5）。

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 PSoC 3 和 PSoC 5 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： None（无）

Return Value（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 5 — 如果之前已启用中断，则返回 1；如果之前已禁用中断，则返回 0。

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。

Return Value（返回值）： None（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。

Return Value（返回值）： None（无）

void CyHalt(uint8 reason)

说明： 停止 CPU。

参数： reason：要通过的值，用于调试。此值可用于了解调用 CyHalt() 的原因。

Return Value（返回值）： None（无）

void CySoftwareReset(void)

说明： 强制对器件进行软件复位。

参数： None（无）

Return Value（返回值）： None（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。

Return Value（返回值）： None（无）

副作用和限制： 通过假设启用了指令缓存，已实现 CyDelay。PSoC 5 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：延迟的微秒数。

Return Value（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。PSoC 5 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。

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

Return Value（返回值）： None（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。

Return Value（返回值）： None（无）

13 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
- 与引导加载程序集成
- 调用 main() C 入口点

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。

PSoC 5

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。

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。

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（适用于 PSoC 5）

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。提供以下支持：

- 核心外设接入层
  - core_cm3.c
  - core_cm3.h

无需修改便可使用这些文件。相同的文件适用于所有我们的支持平台。

复位状态的保留（适用于 PSoC 3 和 PSoC 5）

无论何时，当设备启动时，都应读取和清除复位状态寄存器的值，且该值应保存为全局 SRAM 变量。此寄存器位 PSoC 3 的 RESET_SR0。应提供该变量以及寄存器中字段的定义。

uint8 CyResetStatus

Name（名称）	说明
CY_RESET_LVID	低压检测数字
CY_RESET_LVIA	低压检测模拟
CY_RESET_HVIA	高压检测模拟
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

14 看门狗定时器

API

void CyWdtStart(uint8 ticks, uint8 lpMode)

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

看门狗启用后将无法禁用。每次中央时轮 (CTW) 达到指定时期，看门狗都将计数。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

值	Define（定义）	时间
0	CYWDT_2_TICKS	2 个 CTW 计时单元
1	CYWDT_16_TICKS	16 个 CTW 计时单元
2	CYWDT_128_TICKS	128 个 CTW 计时单元
3	CYWDT_1024_TICKS	1024 个 CTW 计时单元

lpMode：低功耗模式配置

值	Define（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

Return Value（返回值）： None（无）

副作用和限制 所有“lpMode”参数值都受 PSoC 3 production 芯片支持。PSoC 5 和 PSoC 3 ES2 仅支持 NOCHANGE。

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。

void CyWdtClear()

说明： 清除（馈送）看门狗定时器。

参数： None（无）

Return Value（返回值）： None（无）

15 cy_boot 组件更改

2.40 版

本节列出并说明了 2.40 版 cy_boot 组件的主要更改：

2.30 版的更改说明	更改/影响原因
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/影响原因
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，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。
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。	禁用中断以在恢复器件状态之前出现中断。

2.30 版的更改说明	更改/影响原因
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。中央时轮 (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 SleepTimer 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 器件上非 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。保持关系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/影响原因
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 器件添加了一个解决方法，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

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/影响原因
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。

2.20 版的更改说明	更改/影响原因
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/影响原因
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/影响原因
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/影响原因
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/影响原因
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。