请注意赛普拉斯已正式并入英飞凌科技公司。
此封面页之后的文件标注有“赛普拉斯”的文件即该产品为此公司最初开发的。请注意作为英飞凌产品组合的部分，英飞凌将继续为新的及现有客户提供该产品。

文件内容的连续性
事实是英飞凌提供如下产品作为英飞凌产品组合的部分不会带来对于此文件的任何变更。未来的变更将在恰当的时候发生，且任何变更将在历史页面记录。

订购零件编号的连续性
英飞凌继续支持现有零件编号的使用。下单时请继续使用数据表中的订购零件编号。

PSoC 4000 系列

PSoC® 4 技术参考手册（TRM）

文档编号：001-92060 修订版 *A

May 30, 2017

赛普拉斯半导体
198 Champion Court
San Jose， CA 95134-1709
http://www.cypress.com

许可证

© 赛普拉斯半导体公司，2013-2017 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。

目录概况

Section A: 概述 **11**

1. 简介 13
2. 入门 17
3. 文档结构 19

Section B: CPU系统 **23**

4. Cortex-M0 CPU 25
5. 中断 31

Section C: 存储器系统 **39**

6. 存储器映射 41

Section D: 系统资源 **43**

7. I/O系统 45
8. 时钟系统 53
9. 电源与电源监测 59
10. 芯片工作模式 63
11. 功耗模式 65
12. 看门狗定时器 69
13. 复位系统 73
14. 器件安全 75

Section E: 数字系统 **77**

15. 内部集成电路（I2C） 79
16. 定时器、计数器和PWM 95

Section F: 模拟系统 **115**

17. CapSense 117

Section G: 编程和调试 **127**

18. 编程与调试接口 129
19. 非易失性存储器编程 135

术语表 **149**

索引 163

目录

Section A: 概述 **11**

文档修订记录 11

1. 简介 **13**

1.1 顶层架构 14
1.2 特性 14
1.3 CPU系统 14
1.3.1 处理器 14
1.3.2 中断控制器 15
1.4 存储器 15
1.5 系统范围资源 15
1.5.1 时钟系统 15
1.5.2 电源系统 15
1.5.3 GPIO 15
1.6 固定功能数字模块 15
1.6.1 定时器/计数器/PWM模块 15
1.6.2 I2C模块 15
1.7 特殊功能的外设 15
1.7.1 CapSense 15
1.8 编程和调试 16

2. 入门 **17**

2.1 支持 17
2.2 产品升级 17
2.3 开发套件 17

3. 文档结构 **19**

3.1 主要部分 19
3.2 文档规范 19
3.2.1 寄存器规定 19
3.2.2 数字规范 19
3.2.3 测量单位 20
3.2.4 缩略语 20

Section B: CPU系统 **23**

顶层架构 23

4. Cortex-M0 CPU **25**

4.1 特性 25
4.2 框图 26
4.3 工作原理 26
4.4 寄存器 26
4.4.1 工作模式 28

4.4.2 指令集 28
4.4.3 Systick定时器 29
4.4.4 调试 29

5. 中断 31

5.1 特性 31
5.2 工作原理 31
5.3 中断和异常事件 — 操作 32
5.3.1 PSoC 4中的中断/异常事件处理 32
5.3.2 电平与脉冲中断 32
5.3.3 异常事件向量表 32
5.4 异常事件源 33
5.4.1 复位异常事件 33
5.4.2 不可屏蔽中断（NMI）异常事件 33
5.4.3 HardFault异常事件 33
5.4.4 管理程序调用（SVCall）异常事件 33
5.4.5 PendSV异常事件 34
5.4.6 SysTick异常事件 34
5.5 中断源 34
5.6 异常事件优先级 34
5.7 使能/禁用中断 35
5.8 异常事件的状态 35
5.8.1 挂起异常事件 35
5.9 异常事件的堆栈使用情况 36
5.10 中断和低功耗模式 36
5.11 异常事件 — 初始化和配置 36
5.12 寄存器 37
5.13 相关文档 37

Section C: 存储器系统 39

顶层架构 39

6. 存储器映射 41

6.1 特性 41
6.2 工作原理 41

Section D: 系统资源 43

顶层架构 43

7. I/O系统 45

7.1 特性 45
7.2 框图 46
7.3 GPIO驱动模式 47
7.3.1 高阻模拟 48
7.3.2 高阻数字 48
7.3.3 电阻上拉或电阻下拉 48
7.3.4 开漏高电平驱动和开漏低电平驱动 48
7.3.5 增强驱动 48
7.3.6 电阻上拉和电阻下拉 48
7.4 摆率控制 48
7.5 CMOS LVTTL电平控制 48
7.6 高速输入/输出矩阵 49
7.7 固件控制GPIO 49

7.8 CapSense 49
7.9 I/O端口的重新配置 50
7.10 上电时的I/O状态 50
7.11 低功耗模式中的行为 50
7.12 GPIO中断 50
7.12.1 特性 50
7.12.2 中断控制器框图 50
7.12.3 功能和配置 50
7.13 寄存器 51

8. 时钟系统 53

8.1 框图 53
8.2 时钟源 55
8.2.1 内部主振荡器 55
8.2.2 内部低速振荡器 55
8.2.3 外部时钟 55
8.3 时钟分布 55
8.3.1 HFCLK输入选择 55
8.3.2 HFCLK预分频器的配置 55
8.3.3 SYSCLK预分频器的配置 56
8.3.4 外设时钟分频器的配置 56
8.4 低功耗操作模式 56
8.5 寄存器列表 57

9. 电源与电源监测 59

9.1 框图 59
9.2 供电电压方案 60
9.2.1 电压范围为1.8 V至5.5 V的单线未调节电源 60
9.2.2 电压范围为1.71 V到1.89 V的调节电源直接供电 60
9.2.3 VDDIO电源。 61
9.3 工作原理 61
9.3.1 电压调压器汇总 61
9.3.2 电压监控 61
9.4 寄存器列表 62

10. 芯片工作模式 63

10.1 引导模式 63
10.2 用户 63
10.3 特权级别 63
10.4 调试 63

11. 功耗模式 65

11.1 活动模式 66
11.2 睡眠模式 66
11.3 深度睡眠模式 66
11.4 功耗模式总汇 66
11.5 低功耗模式进入和退出 67
11.6 寄存器列表 67

12. 看门狗定时器 69

12.1 特性 69
12.2 框图 69
12.3 工作原理 70

12.3.1 使能和禁用WDT 70
12.3.2 WDT中断和低功耗模式 70
12.3.3 WDT复位模式 70
12.4 寄存器列表 71

13. 复位系统 73

13.1 复位源 73
13.1.1 上电复位 73
13.1.2 欠压复位 73
13.1.3 看门狗复位 73
13.1.4 软件复位 73
13.1.5 外部复位 74
13.1.6 特权保护复位 74
13.2 识别复位源 74
13.3 寄存器列表 74

14. 器件安全 75

14.1 特性 75
14.2 工作原理 75

Section E: 数字系统 77

顶层架构 77

15. 内部集成电路（I2C） 79

15.1 特性 79
15.2 概述 79
15.2.1 术语和定义 80
15.3 I2C工作模式 80
15.3.1 写数据 81
15.3.2 读数据 81
15.4 Easy I2C（EZI2C）协议 81
15.4.1 存储器阵列的写操作 82
15.4.2 存储器阵列的读操作 82
15.5 I2C寄存器 83
15.6 I2C中断 83
15.7 使能和初始化I2C 83
15.8 I2C中的内部和外部时钟模式 84
15.8.1 I2C非EZ工作模式 85
15.8.2 EZ工作模式 85
15.8.3 从睡眠模式中唤醒 86
15.8.4 主设备模式的传输示例 87
15.8.5 从设备模式的传输示例 89
15.8.6 EZ从设备模式的传输示例 91
15.8.7 多个主设备模式的传输示例 93

16. 定时器、计数器和PWM 95

16.1 特性 95
16.2 框图 96
16.2.1 使能和禁用TCPWM模块中的计数器 96
16.2.2 时钟 96
16.2.3 基于触发输入的事件 96
16.2.4 输出信号 97
16.2.5 功耗模式 98

16.3 各种操作模式 99
16.3.1 定时器模式 100
16.3.2 捕获模式 102
16.3.3 正交解码器模式 104
16.3.4 脉冲宽度调制模式 107
16.3.5 带死区模式的脉冲宽度调制 110
16.3.6 脉冲宽度调制伪随机模式 111
16.4 TCPWM寄存器 114

Section F: 模拟系统 115

顶层架构 115

17. CapSense 117

17.1 特性 117
17.2 框图 117
17.3 工作原理 118
17.4 CapSense CSD感应 119
17.4.1 GPIO单元中电容-电流转换器 119
17.4.2 CapSense时钟发生器 121
17.4.3 Sigma Delta转换器 121
17.5 CapSense CSD屏蔽 122
17.5.1 CMOD预充电 123
17.6 通用资源 — IDAC和比较器 124
17.7 寄存器列表 125

Section G: 编程和调试 127

顶层架构 127

18. 编程与调试接口 129

18.1 特性 129
18.2 功能说明 129
18.3 串行线调试（SWD）接口 130
18.3.1 SWD时序的详细信息 131
18.3.2 ACK的详细信息 131
18.3.3 反转（Trn）周期的详细信息 131
18.4 Cortex-M0调试和访问端口（DAP） 131
18.4.1 调试端口（DP）寄存器 131
18.4.2 访问端口（AP）寄存器 132
18.5 编程PSoC 4器件 132
18.5.1 获取SWD端口 132
18.5.2 SWD编程模式入口 133
18.5.3 SWD编程子程序执行 133
18.6 PSoC 4 SWD调试接口 133
18.6.1 调试控制和配置寄存器 133
18.6.2 断点单元（BPU） 133
18.6.3 数据观察点（DWT） 133
18.6.4 调试PSoC 4器件 133
18.7 寄存器 134

19. 非易失性存储器编程 135

19.1 特性 135
19.2 功能说明 135
19.3 系统调用实现 136

19.4 阻塞和非阻塞的系统调用 136
19.4.1 执行系统调用 136
19.5 系统调用 136
19.5.1 芯片ID 137
19.5.2 配置时钟 138
19.5.3 加载闪存字节 138
19.5.4 写入行 139
19.5.5 编程行 140
19.5.6 擦除所有 141
19.5.7 校验和 141
19.5.8 写保护 142
19.5.9 非阻塞写入行 142
19.5.10 非阻塞编程行 143
19.5.11 恢复非阻塞 144
19.6 系统调用状态 145
19.7 非阻塞系统调用伪代码 145

术语表 **149**

索引 163

Section A: 概述

本章节包括下面章节：

- 第 13 页上的简介章节
- 第 17 页上的入门章节
- 第 19 页上的文档结构章节

文档修订记录

版本	发布日期	变更人	修订说明
**	04/09/2014	NIDH	本文档版本号为 Rev*A，译自英文版 001-89309 Rev**。
*A	05/30/2017	NIDH	Updated logo and copyright

1. 简介

PSoC® 4 是基于 ARM® Cortex™-M0 CPU 的可编程嵌入式系统控制器的的家族，为嵌入式应用提供了一个强大的可编程平台。

CY8C4000 系列是 PSoC 4 器件系列中最小的成员，并与 PSoC 4 中的更大成员向上兼容。

PSoC 4 器件具有以下特性：

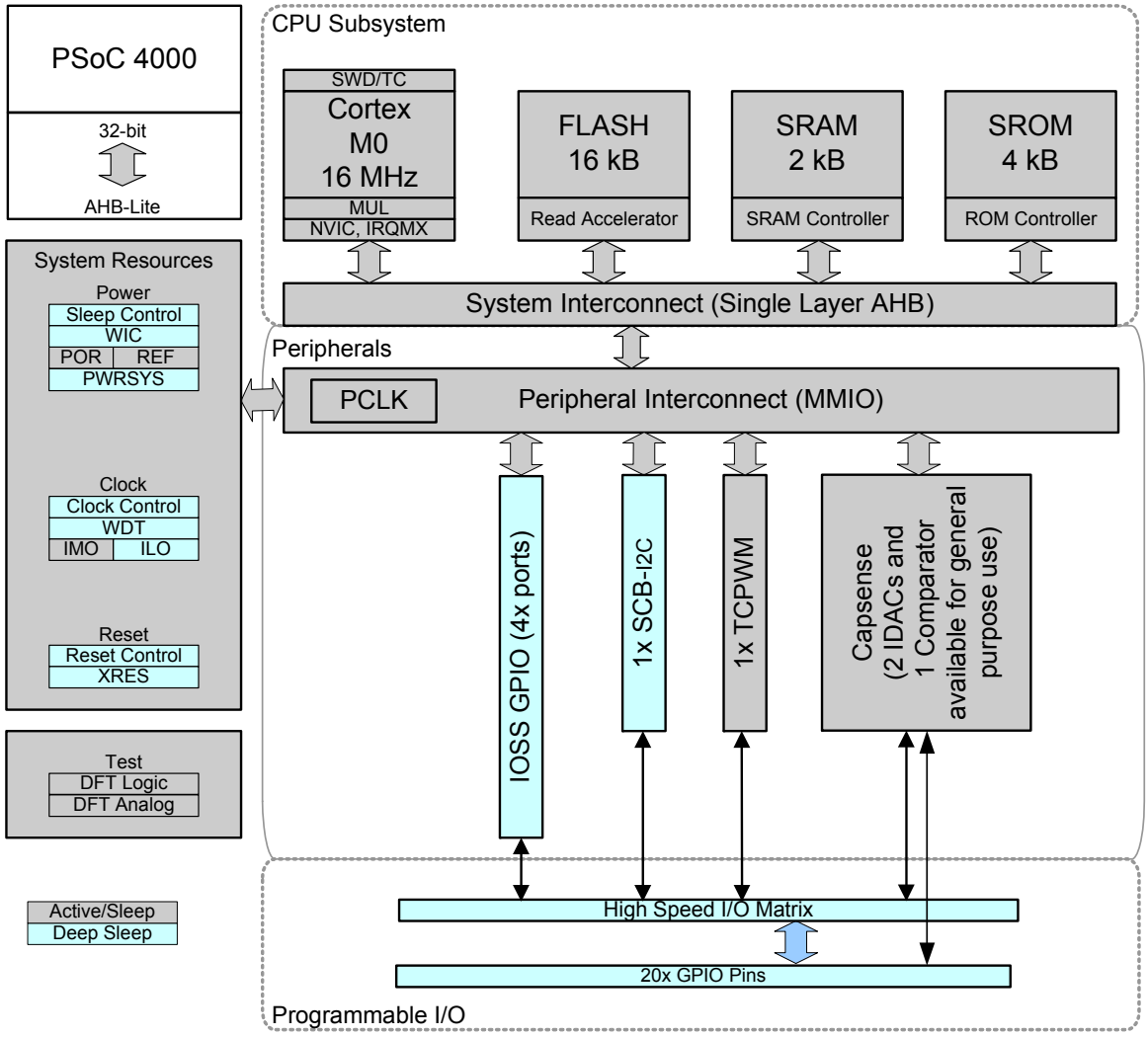
- 具有单周期乘法功能的高性能 32 位 Cortex-M0 CPU 内核
- 电容式触摸传感 （CapSense®）
- 可配置定时器 / 计数器 /PWM 模块
- 两个支持拉电流 / 灌电流模式的 DAC （IDAC）
- 参考电压为 1.2 V 的比较器
- 可配置为主设备、从设备和多主设备操作模式的 I2C 模块
- 低功耗模式包括睡眠模式和深度睡眠模式

本文档详细介绍了 PSoC 4 器件中每个功能模块的相关信息。从而帮助设计员创建系统级的设计。

1.1 顶层架构

图 1-1 显示的是 PSoC 4000 架构内的主要组件。

图 1-1. PSoC 4000 系列的框图



1.2 特性

PSoC 4000 系列具有下面的主要组件：

- 带单周期乘法的 32 位 Cortex-M0 CPU，其工作频率为 16 MHz，且它的指令执行速度高达 14 DMIPS
- 闪存空间高达 16 KB， SRAM 大小为 2 KB
- 带互补死区可编程输出的中心对齐脉冲宽度调制器（PWM）
- 可配置为主设备、从设备和多主设备操作模式的 I2C 模块
- CapSense
- 低功耗运行模式：睡眠模式和深度睡眠模式
- 串行线调试 （SWD）编程和调试系统
- 两个支持拉电流 / 灌电流模式的 DAC （IDAC）
- 参考电压为 1.2 V 的比较器
- 全面支持 PSoC Creator™ IDE 工具

1.3 CPU 系统

1.3.1 处理器

PSoC 4 的主要组件是 32 位的 Cortex-M0 CPU 内核，其工作频率高达 16 MHz。该内核通过扩展的时钟门控来优化低功耗操作。它使用了 16 位指令并执行 Thumb-2 指令子集。这样

能够将完全兼容的二进制代码导入更高性能的处理器，如 Cortex M3 和 M4。

PSoC 4 同时包含一个输出 32 位结果的单周期硬件乘法器。

1.3.2 中断控制器

PSoC 4 中的 CPU 子系统包括一个带有 9 个中断输入的嵌套向量中断控制器（NVIC）和一个用于从深度睡眠模式唤醒处理器的唤醒中断控制器（WIC）。

1.4 存储器

PSoC 4 存储器子系统由一个带有闪存加速器的 16 KB 闪存模块、一个 2 KB SRAM 和 4 KB 管理 ROM 选项组成。闪存加速器加快了对闪存模块的访问速度，这样可以得到单周期 SRAM 访问 85 % 的性能。通过功能强大且非常灵活的保护模型，您可以锁定选定的存储器模块以便实现读写保护，保护您的敏感信息。此外，如果担心因器件恶意重新编程而造成欺诈性攻击的应用，或通过启动和中断闪存编程序列来击败安全性的尝试，则可以永久性禁用所有器件接口。如果需要，闪存模块的一部分空间可以用于模仿 EEPROM 操作。只读管理存储器用于存储引导和配置子程序。

1.5 系统范围资源

1.5.1 时钟系统

PSoC 4 器件的时钟系统既包括内部时钟（内部主要振荡器（IMO）和内部低速振荡器（ILO））以及一个外部时钟接口。

CPU 系统需要的系统时钟（SYSCLK）以及外设需要的高频率时钟（HFCLK）的频率均可达 16 MHz。这些时钟均由 IMO 生成。

在 PSoC 4 中， IMO 时主要的内部时钟源。IMO 的默认频率为 24 MHz，并可在 24 MHz 到 48 MHz 的范围内对其进行调整，调整步长为 4 MHz。 IMO 和赛普拉斯提供的校准之间的容差为 ±2%（24 MHz 和 32 MHz）。可以从主时钟频率生成多个分频时钟，以满足应用需求。

ILO 是一个准确度较低的低功耗振荡器，用于为使外设能在深度睡眠模式下运行生成时钟。它的时钟频率为 32 kHz。

可以通过将频率范围上拉为 0 MHz 到 16 MHz 的外部时钟源（而不是使用 IMO），为 PSoC 4 的功能模块生成分频时钟。

1.5.2 电源系统

PSoC 4 的单一外部电源工作电压范围为 1.71 V 到 5.5 V。除了默认的活动模式外，PSoC 4 还能在两个低功耗模式（即睡眠 和深度睡眠模式）下运行。

在活动模式下，CPU 中所有的逻辑均被供电。在睡眠模式下，CPU 不运行，并且其工作时钟被关闭。在深度睡眠模式下，CPU、 SRAM 和高速度逻辑均被保留；同时，主要系统时钟关闭，低频时钟打开并且低频外设将会运行。

系统中的多个内部调压器能满足不同功耗模式的供电要求。

1.5.3 GPIO

PSoC 4 中的每个 GPIO 均有以下特性：

- 八种驱动模式
- 输入和输出禁用的单独控制
- 用于锁存前一状态的保持模式
- 可选的摆率
- 中断生成：边沿触发

PSoC 4 20 个 GPIO 引脚中有 17 个支持 CapSense 功能。各引脚集中在个 8 位宽的端口中。高速度 I/O 矩阵用于复用连接至一个 I/O 引脚的多个信号。固定功能外设的引脚有固定的位置。

1.6 固定功能数字模块

1.6.1 定时器 / 计数器 /PWM 模块

定时器 / 计数器 /PWM 模块包含一个用户可编程周期长度的 16 位计数器。 TCPWM 模块由捕获寄存器、周期寄存器和比较寄存器组成。该模块支持互补死区可编程输出。它还提供用于强制输出进入未确定状态的终止（Kill）输入。该模块的其他特性包括中心对齐 PWM、时钟预分频、伪随机 PWM 和正交解码器。

1.6.2 I2C 模块

PSoC 4 具有一个功能固定的 I2C 接口。可将 I2C 接口用于通用的 I2C 通信以及调试 CapSense 组件，以优化其操作。

I2C 模块具有下面特性：

- 标准的 I2C 多主设备和从设备功能
- 支持带 32 字节缓存的 EZ 功能模式

1.7 特殊功能的外设

1.7.1 CapSense

PSoC 4 器件具有 CapSense 性能，允许您能够通过使用手指的电容属性切换按键、滑条和滚轮。PSoC 4 通过 CapSense Sigma-Delta（CSD）模块，在除 3 个 GPIO 引脚外的所有引脚上支持 CapSense 功能。 CSD 还提供了防水功能。

1.7.1.1 IDAC 和比较器

CapSense 模块包括两个 IDAC 和一个比较器，其参考电压为 1.2 V；在不使用CapSense的情况下，它们可用于通用目的。

1.8 编程和调试

PSoC 4器件通过片上SWD接口支持器件的编程和调试性能。PSoC Creator 集成开发环境 （IDE）软件能够为 PSoC 4 器件提供全面集成的编程和调试支持。SWD 接口与行业标准的第三方工具全面兼容。

2. 入门

2.1 支持

可在线访问 http://www.cypress.com 来获取 PSoC® 4 产品的免费支持。资源包括培训讲座、论坛、应用笔记、 PSoC 顾问、CRM 技术支持电子邮件、知识库以及应用支技术人员。

有关应用支持，请访问 http://www.cypress.com/support/ 或电话联系 1-800-541-4736。

2.2 产品升级

赛普拉斯免费提供 PSoC Creator 的定期升级以及版本更新。可以从分销商提供的光盘中找到升级版本，也可以直接从 http://www.cypress.com 网页上 ‘ 软件 ’ 部分下载升级版本。文档目录下同时提供系统文档的重要更新。

2.3 开发套件

可在 Digi-Key、Avnet、Arrow 以及 Future 中获得开发套件。赛普拉斯在线商店包含所有成功开发 PSoC 项目所需的开发套件、C 编译器和附件。请访问赛普拉斯的在线商店 http://www.cypress.com/shop/。在产品目录下，通过点击 **Programmable System-on-Chip** 可以查找当前可用的器件系列。

3. 文档结构

本文档中的以下部分包括这些主题：

- 第 23 页上的 Section B:CPU 系统
- 第 39 页上的 Section C: 存储器系统
- 第 43 页上的 Section D: 系统资源
- 第 77 页上的 Section E: 数字系统
- 第 115 页上的 Section F: 模拟系统
- 第 127 页上的 Section G: 编程和调试

3.1 主要部分

为便于使用，本文档中的信息被整理为各部分和章节。这些部分和章节根据器件的功能来划分。

- 部分 — 介绍顶级架构、如何入门以及任何特别区域的规范和概述信息。该区域可通知读者该产品的结构和组织。
- 章节 — 介绍用于指定部分主题的单独方面的章节。针对集成电路的某些方面，详细执行和使用信息。
- 术语表 — 定义了本技术参考手册中使用的专业术语。术语表使用粗体字、斜体字来显示。
- PSoC® 4 寄存器技术参考手册 — 提供技术参考手册中总结的所有器件寄存器的详细信息。这些文档是额外的。

3.2 文档规范

除了标题中使用的某些字体类型以外，本文档只使用四个明显的字体类型。

- 第一个是当参考文档标题或名称时使用的*斜体字*。
- 第二个是当参考本文档中术语表描述的术语时使用的***粗体斜体***。
- 第三个是使用 Times New Roman 字体，以区分公式的例子。
- 第四个是使用 Courier New 字体，以区分代码的例子。

3.2.1 寄存器规定

PSoC® 4 寄存器技术参考手册显示的是寄存器规范。

3.2.2 数字规范

十六进制数字中的所有字母均为大写，结尾带小写的 “h” （例如，“14h” 或 “3Ah”），并*十六进制*数字可由 ‘0x’ 前缀和 *C* 代码规范所示。二进制数字在结尾带小写的 ‘b’ （例如，“01010100b” 或 “01000011b”）。不带 ‘h’ 或 ‘b’ 的数字是*十进制数字*。

3.2.3 测量单位

下表列出了本文档中使用的测量单位。

表 3-1. 测量单位

符号	测量单位
°C	摄氏度
dB	分贝
fF	飞法
Hz	赫兹
k	公斤， 1000
K	公斤， 2^10
KB	1024 个字节或约一千个字节
Kbit	1024 位
kHz	千赫 （32.000）
kW	千欧
MHz	兆赫兹
MW	兆欧
µA	微安
µF	微法
µs	微秒
µV	微伏
µVrms	微伏的均方根
mA	毫安
ms	毫秒
mV	毫伏
nA	纳安
ns	纳秒
nV	纳伏
Ω	欧姆
Ω	皮法
pp	峰峰值
ppm	百万分率
SPS	每秒采样数
s	sigma：一个标准差
V	伏特

3.2.4 缩略语

下表列出的是本文档中使用的缩略语

表 3-2. 缩略语

符号	测量单位
ABUS	模拟输出总线
AC	交流
ADC	模数转换器
AHB	AMBA （先进微控制器总线结构）高性能总线，一种 ARM 数据传输总线
API	应用编程接口
APOR	模拟上电复位
BC	广播时钟
BOM	材料表
BR	比特率
BRA	总线请求确认
BRQ	总线请求
CAN	控制器区域网络
CI	移入
CMP	比较
CO	移出
CPU	中央处理单元
CRC	循环冗余校验
CT	连续时间
DAC	数模转换器
DC	直流
DI	数字或数据输入
DNL	微分非线性
DO	数字或数据输出
DSI	数字信号接口
ECO	外部晶体振荡器
EEPROM	电可擦可编程只读存储器
EMIF	外部存储器接口
FB	反馈
FIFO	先进先出
FSR	全量程
GPIO	通用 I/O
I^2C	内部集成电路
IDE	集成开发环境
ILO	内部低速振荡器
IMO	内部主振荡器
INL	积分非线性度
I/O	输入 / 输出
IOR	I/O 读取
IOW	I/O 写入

表 3-2. 缩略语 *(continued)*

符号	测量单位
IRES	初次加电复位
IRA	中断请求确认
IRQ	中断请求
ISR	中断服务子程序
IVR	中断矢量读取
LRb	最后接收的一个位
LRB	最后接收的一个字节
LSb	最低有效位
LSB	最低有效字节
LUT	查询表
MISO	主入从出
MMIO	存储器映射输入 / 输出
MOSI	主出从入
MSb	最高有效位
MSB	最高有效字节
PC	程序计数器
PCH	程序计数器的高字节
PCL	程序计数器的低字节
PD	断电
PGA	可编程增益放大器
PM	电源管理
PMA	PSoC 存储器仲裁器
POR	加电复位
PPOR	精确上电复位
PRS	伪随机序列
PSoC®	可编程片上系统
PSRR	电源抑制比
PSSDC	电源系统睡眠占空比
PWM	脉冲宽度调制器
RAM	随机存取存储器
RETI	从中断返回
ROM	只读存储器
RW	读 / 写
SAR	逐次逼近寄存器
SC	开关电容
SCB	串行通信模块
SIE	串行接口引擎
SIO	特殊 I/O
SE0	单端零
SNR	信噪比
SOF	帧起始
SOI	指令起始
SP	堆栈指针

表 3-2. 缩略语 *(continued)*

符号	测量单位
SPD	顺序相位检测器
SPI	串行外设互连
SPIM	串行外设互连主设备
SPIS	串行外设互连从设备
SRAM	静态随机存取存储器
SROM	只读管理存储器
SSADC	单斜 ADC
SSC	管理系统调用
SWD	单线调试
TC	终端计数
TD	事务描述符
UART	通用异步接收器 / 发送器
UDB	通用数字模块
USB	通用串行总线
USBIO	USB 输入 / 输出
WDT	看门狗定时器
WDR	看门狗复位
XRES_N	外部复位，低电平有效

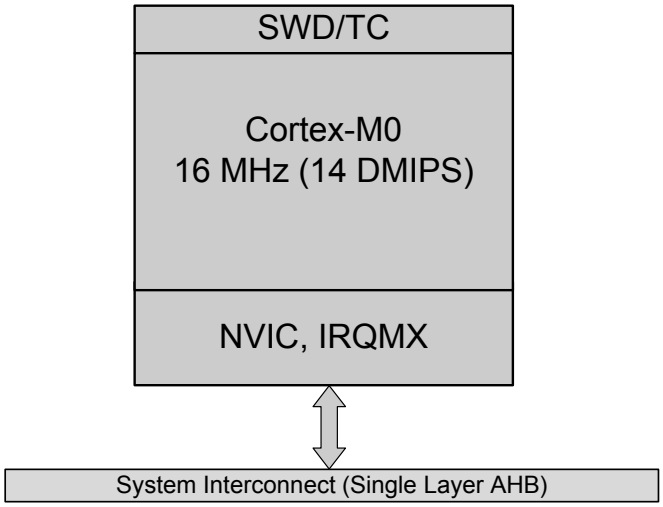
Section B: CPU 系统

本章节包括以下小节：

- 第 25 页上的 Cortex-M0 CPU 章节
- 第 31 页上的中断章节

顶层架构

CPU 系统框图



4. Cortex-M0 CPU

PSoC® 4 ARM Cortex-M0 内核是一个低功耗的 32 位 CPU。它拥有一个高效率的三段流水线式、一个固定的 4 GB 存储器映射，另外还支持 ARMv6-M Thumb 指令集。Cortex-M0 还提供一个单周期的乘法指令和低延迟的中断服务子程序（ISR）进入和退出。

Cortex-M0 处理器包含与 CPU 内核紧密相连的许多其他组件。这些组件具有一个嵌套向量中断控制器（NVIC）、一个 SYSTICK 定时器和调试。

本节介绍了 Cortex-M0 处理器的概述。更多有关信息，请查看 http://www.arm.com 网站中提供的 ARM Cortex-M0 用户指南或技术参考手册。

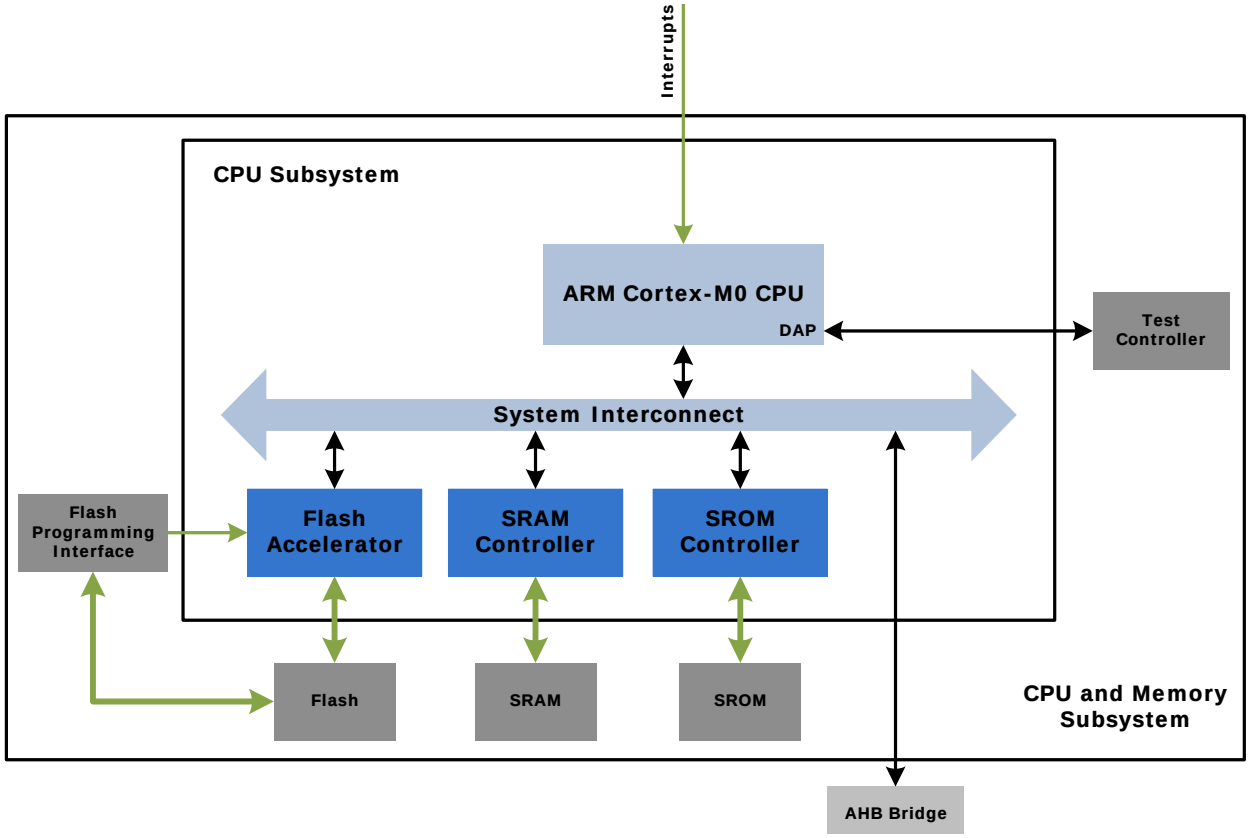
4.1 特性

PSoC 4 Cortex-M0 具有以下特性：

- 工作速度可达 0.9 DMIPS/MHz ；这有助于提高程序执行速度、降低功耗
- 支持 Thumb 指令集，这可提高代码密度和对存储器的利用率
- 支持中断和异常的 NVIC 单元，用于确保快速和确定的中断响应
- 广泛的调试支持包括：
  - 串行线调试 （SWD）端口
  - 断点
  - 观察点

4.2 框图

图 4-1. PSoC 4 CPU 子系统框图



4.3 工作原理

Cortex-M0 是一个 32 位处理器，包含一个 32 位数据路径、 32 位寄存器和一个 32 位存储器接口。它支持 Thumb 指令集中的大多数 16 位指令和 Thumb-2 指令集中的一些 32 位指令。

该处理器支持两种工作模式和一个单周期的 32 位乘法指令。

4.4 寄存器

Cortex-M0 具有 16 个 32 位寄存器，如表 4-1 中所示：

- R0 到 R12 — 通用寄存器。 R0 到 R7 可由所有指令访问，其他寄存器可由指令子集访问。
- R13 — 堆栈指针 （SP）。有两个堆栈指针，一次只有一个指针可用。在线程模式下， CONTROL （控制）寄存器指出将使用的堆栈指针：主堆栈指针 （MSP）还是进程堆栈指针 （PSP）。
- R14 — 链接寄存器。在调用函数时存储返回程序计数器。
- R15 — 程序计数器。通过写入该寄存器可以控制程序流。

表 4-1. Cortex-M0 寄存器

名称	类型[a]	复位值	说明
R0-R12	RW	未知	R0-R12 是用于数据操作的 32 位通用寄存器。
MSP PSP	RW	[0x00000000]	堆栈指针 （SP）寄存器是寄存器 R13。在线程模式中， CONTROL （控制）寄存器的位 [1] 指出被使用的堆栈指针： 0 = 主堆栈指针 （MSP）。这是复位值。 1 = 进程堆栈指针 （PSP）。 复位时，处理器将地址 0x00000000 中的值加载到 MSP。
LR	RW	未知	链接寄存器 （LR）是寄存器 R14。它存储子程序、函数调用和异常情况的返回信息。
PC	RW	[0x00000004]	程序计数器 （PC）为寄存器 R15。它包含当前程序地址。复位时，处理器将地址 0x00000004 中的值加载到 PC。复位时，该值的位 [0] 必须为 1 并被加载到 EPSR T 位。
PSR	RW	未知[b]	程序状态寄存器 （PSR）结合： 应用程序状态寄存器 （APSR）。 执行程序状态寄存器 （EPSR）。 中断程序状态寄存器 （IPSR）。
APSR	RW	未知	APSR 包含前指令执行程序中条件标志的当前状态。
EPSR	RO	未知[b]	EPSR 包含 Thumb 状态位。
IPSR	RO	0	IPSR 包含当前 ISR 的异常编号。
PRIMASK	RW	0	PRIMASK 寄存器防止激活可配置优先级的异常情况。
CONTROL	RW	0	CONTROL （控制）寄存器控制处理器在线程模式下使用的堆栈。

a. 说明程序在线程模式和处理程序模式运行时的访问类型。调试访问会有所不同。
b. 位 [24] 是从复位向量的位 [0] 加载的 T 位。

表 4-2 显示的是 PSR 位的分配方式。

表 4-2. Cortex-M0 PSR 位分配

位	PSR 寄存器	名称	使用情况
31	APSR	N	负标志
30	APSR	Z	零标志
29	APSR	C	进位标志或借位标志
28	APSE	V	溢出标志
27 – 25	–	–	保留
24	EPSR	T	Thumb 状态位。必须始终为 1。 T 位为 0 时尝试执行指令将导致 HardFault 异常。
23 – 6	–	–	保留
5 – 0	IPSR	N/A	当前 ISR 的异常编号： 0 = 线程模式 1 = 保留 2 = NMI 3 = HardFault 4 – 10 = 预留 11 = SVCall 12、 13 = 保留 14 = PendSV 15 = SysTick 16 = IRQ0 … 24 = IRQ8

使用 MSR 或 CPS 指令来设置或清除 PRIMASK 寄存器的位 0。如果该位为 0，将使能异常。如果该位为 1，将禁用所有可配置优先级的异常，即所有异常，HardFault、NMI 和 Reset（复位）除外。第 31 页上的中断章节显示的是异常列表。

4.4.1 工作模式

Cortex-M0 处理器支持两种工作模式：

- 线程模式 — 由所有普通应用使用。在线程模式下，可以使用 MSP 或 PSP。 CONTROL （控制）寄存器的位 1 确定将使用的堆栈指针：
  - 0 = 当前的堆栈指针为 MSP
  - 1 = 当前的堆栈指针为 PSP
- 处理模式 — 用于执行异常处理程序。一直使用 MSP。

在线程模式下，使用 MSR 指令来设置 CONTROL（控制）寄存器中的堆栈指针位。修改堆栈指针时，在使用 MSR 指令后立即使用 ISB 指令。这样确保 ISB 指令后的指令使用新的堆栈指针执行。

在处理模式下，对 CONTROL （控制）寄存器进行写操作无效，因为 MSP 一直在使用。异常进入和返回机制会自动更新 CONTROL （控制）寄存器。

4.4.2 指令集

Cortex-M0 执行 Thumb 指令集版本。有关详细内容，请查看 Cortex-M0 通用用户指南。

指令操作数可以是一个 ARM 寄存器、一个常数或另一个指令特定参数。在操作数上执行指令并将该结果存储在目标寄存器内。将 PC 或 SP 作为操作数或目标寄存器使用，许多指令无法使用或有所限制。

表 4-3. Thumb 指令集

助记符	简要说明
ADCS	进位加
ADD{S}	添加
ADR	PC 相对地址到寄存器
ANDS	位元 AND
ASRS	算术右移
B{cc}	分支 { 条件 }
BICS	位清除
BKPT	断点
BL	分支连接
BLX	分支间接连接
BX	分支间接
CMN	负数比较
CMP	比较
CPSID	修改处理器状态，禁用中断
CPSIE	修改处理器状态，使能中断
DMB	数据存储器屏障
DSB	数据同步屏障

表 4-3. Thumb 指令集

助记符	简要说明
EORS	逻辑异或
ISB	指令同步屏障
LDM	多寄存器数据装载指令
LDR	从 PC 相对地址加载寄存器
LDRB	将字加载到寄存器
LDRH	将半字加载到寄存器
LDRSB	将带符号的字节加载到寄存器
LDRSH	将带符号的半字加载到寄存器
LSLS	逻辑左移
LSRS	逻辑右移
MOV{S}	移动
MRS	从通用寄存器移动到特殊寄存器
MSR	从特殊寄存器移动到通用寄存器
MULS	乘法，结果为 32 位
MVNS	位元 NOT
NOP	无操作
ORRS	逻辑 OR
POP	从堆栈弹出寄存器
PUSH	将寄存器推入堆栈
REV	字节反转字
REV16	字节反转紧密式半字
REVSH	字节反转有符号半字
RORS	向右旋转
RSBS	反向减法
SBCS	进位减
SEV	发送事件
STM	多寄存器数据存储指令
STR	存储一个字
STRB	存储一个字节
STRH	存储半字
SUB{S}	减
SVC	系统管理调用程序
SXTB	符号扩展字节
SXTH	符号扩展半字
TST	基于逻辑 AND 的测试
UXTB	对字节进行零扩展
UXTH	对半字进行零扩展
WFE	等待事件
WFI	等待中断

4.4.2.1 地址对齐

对齐访问是一个操作，其中字对齐地址用于一个字或许多字

访问，或半字对齐地址用于半字访问。字节访问始终对齐。

Cortex-M0 处理器上的非对齐访问不受任何支持。尝试执行非对齐存储器访问操作会导致 HardFault 异常。

4.4.2.2 存储器字节顺序

PSoC 4 Cortex-M0 使用低位优先格式，其中最低有效字节被存储在最低地址，最高有效字节被存储在最高地址。

4.4.3 Systick 定时器

Systick 定时器与 NVIC 集成并生成 SYSTICK 中断。该中断可用于实时系统中的任务管理。该定时器拥有一个重新加载寄存器，其中 24 位可作为一个倒计时的值使用。 Systick 定时器将 Cortex-M0 内部时钟作为源时钟使用。

4.4.4 调试

PSoC 4 包含一个基于 SWD 的调试接口；它具有四个断点（地址）比较器和两个观察点 （数据）比较器。

5. 中断

PSoC® 4 中的 ARM Cortex-M0（CM0）CPU 支持中断和异常事件。中断是指由 CPU 之外的设备（如定时器、串行通信模块和端口引脚信号）生成的事件。异常事件是指由 CPU （如存储器访问故障和内部系统定时器事件）生成的事件。中断和异常事件都会中止当前的程序流程，同时， CPU 会执行与该中断 / 异常事件相应的处理程序（即为中断服务子程序）。 PSoC 4 为中断处理程序和异常情况处理程序提供了统一的异常向量表。

5.1 特性

PSoC 4 支持下面的中断特性：

- 支持 9 个中断
- CPU 内核中集成了嵌套向量中断控制器 （NVIC），从而得到较短的中断延迟
- 可将向量表保存在闪存或 SRAM 内
- 每个中断的可配置优先级范围为 0 到 3
- 支持电平触发和边沿触发

5.2 工作原理

图 5-1. PSoC 4 中断框图

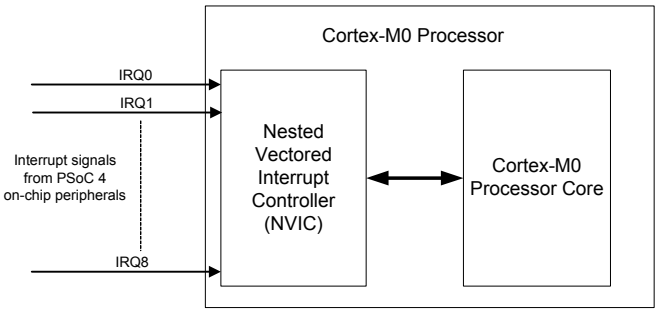


图 5-1 显示的是中断信号与 Cortex-M0 CPU 间的交互。 PSoC 4 一共有 9 个中断；这些中断信号均由 NVIC 来处理。 NVIC 分别用于使能 / 禁用单独中断、优先级处理，连接 CPU 内核。与 CPU 的外部设备所生成的中断不同，异常事件是由 CM0 内核生成的，因此，它们不会在图 5-1 中显示。

5.3 中断和异常事件 — 操作

5.3.1 PSoC 4 中的中断 / 异常事件处理

在触发某个中断或异常事件时所发生的一系列事件包括：

1. 假设所有的中断信号的初始状态均为低电平 （闲置或非活动状态），处理器正在执行主代码，并且 NVIC 会记录任何一个中断线上的上升沿。这时，该中断线处于挂起状态，直至 CPU 开始处理该中断为止。
2. 当检测到来自 NVIC 的中断请求信号时， CPU 会将其寄存器中的内容推放到堆栈上，以保存它的当前上下文内容。
3. 同时， CPU 还接收来自 NVIC 的触发中断的异常事件编号。 PSoC 4 中的所有中断和异常事件均有唯一的异常事件编号，如表 5-1 所示。通过使用该异常事件编号，CPU 可以从向量表中加载特定异常事件处理器的地址。
4. 然后， CPU 转至该地址，并执行相应的异常事件处理器。
5. 在完成执行异常事件处理器后， CPU 寄存器会通过堆栈弹出操作恢复到它的原始状态，同时 CPU 也恢复执行主代码。

如果在执行某个中断时 NVIC 接收到另一个中断请求，或当它同时接收到多个中断请求时， NVIC 会评估这些中断的优先级，然后向 CPU 发送优先级最高的中断异常事件的编号。因此，优先级高的中断可以随时中断执行优先级低的中断处理器。

异常事件的处理方法与中断的处理方法相同。每个异常事件均有唯一的异常事件编号。 CPU 使用该编号执行相应的异常事件处理器。

5.3.2 电平与脉冲中断

PSoC 4 NVIC 支持中断线（IRQ0 至 IRQ8）上的电平和脉冲信号。中断类型 （电平或边沿）由中断源决定。

图 5-2. 电平中断

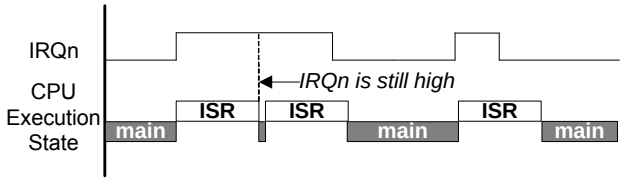


图 5-3. 边沿中断

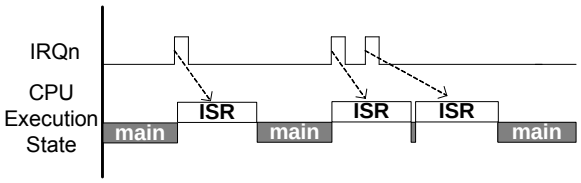


图 5-2 和图 5-3 分别显示的是工作中的电平中断和边沿中断。假设中断信号最初的状态是无效的（逻辑低），下面各事件序列说明了如何处理电平中断和脉冲中断：

1. 在中断信号的上升沿事件中， NVIC 将记录中断请求。此时中断处于挂起状态，即 CPU 尚未处理该中断请求。
2. NVIC 会将异常事件编号和中断请求信号一同发送到 CPU。当 CPU 开始执行中断服务子程序 （ISR）时，将清除中断的挂起状态。
3. 当 CPU 执行 ISR 时，一个或多个中断信号的上升沿将被记录，并被视为单一的挂起请求。在完成执行当前的 ISR 后，挂起的中断将再次被处理 （请查看图 5-3，了解边沿中断的相关信息）。
4. 执行完 ISR 后，如果中断信号仍为高电平，则该信号会处于挂起状态，并且 ISR 被重新执行。图 5-2 说明了电平触发中断的情况，其中只要中断信号为高电平，则始终执行 ISR。

5.3.3 异常事件向量表

异常事件向量表（表 5-1）保存的是 PSoC 4 中所有异常事件处理器的入口点地址。 CPU 会根据异常事件编号加载相应的地址。

表 5-1. PSoC 4 异常向量表

异常编号	异常事件	异常优先级	向量地址
–	初始堆栈指针值	不适用 （NA）	Base_Address — 可以为 0x00000000 （闪存存储器的起始地址），或者为 0x20000000 （SRAM 的起始地址）
1	复位	–3，最高优先级	Base_Address + 0x04
2	不可屏蔽中断 （NMI）	–2	Base_Address + 0x08
3	硬故障	–1	Base_Address + 0x0C
4-10	保留	NA	Base_Address + 0x10 - Base_Address + 0x28
11	管理程序调用 （SVCall）	可配置 （0 - 3）	Base_Address + 0x2C
12-13	保留	NA	Base_Address + 0x30 - Base_Address + 0x34
14	管理程序挂起 （PendSV）	可配置 （0 - 3）	Base_Address + 0x38

表 5-1. PSoC 4 异常向量表

异常编号	异常事件	异常优先级	向量地址
15	系统定时器 （SysTick）	可配置 （0 - 3）	Base_Address + 0x3C
16	外部中断 （IRQ0）	可配置 （0 - 3）	Base_Address + 0x40
…	…	可配置 （0 - 3）	…
24	外部中断 （IRQ8）	可配置 （0 - 3）	Base_Address + 0x52

在表 5-1 中，没有将第一个字（4 字节）标记为异常事件编号零。其原因是异常事件表中的第一个字用于在器件复位时初始化主堆栈指针（MSP）的值；并不将其视为一个异常事件。在 PSoC 4 中，可通过配置向量表，使其位于闪存存储器（起始地址为 0x00000000），或位于 SRAM （起始地址为 0x20000000）。通过对 CPUSS_CONFIG 寄存器中的 VECT_IN_RAM 位字段 （位 0）进行写操作，可以实现该配置。当 VECT_IN_RAM 位字段为 ‘1’ 时，CPU 可从 SRAM 中的向量表位置加载异常事件处理器的地址。当该位字段为 ‘0’ （复位状态）时，可通过使用闪存存储器中的向量表加载异常事件的地址。您必须将 VECT_IN_RAM 位字段设置为器件引导代码的一部分，以通过配置使向量表转移到SRAM。将向量表转移到 SRAM 中的优点是：通过修改 SRAM 向量表的内容，可以动态更改异常事件处理器的地址。然而，必须通过对闪存存储器进行的写操作才能更改非易失性闪存存储器的向量表。

5.4 异常事件源中说明了异常事件源 （异常事件编号 1 到 15）。虽然在表 5-1 中标记为保留的异常事件都有自己的保留地址，但在 PSoC 4 中不会使用它们。5.5 中断源说明了各个中断源 （异常事件编号为 16 到 24）。

5.4 异常事件源

本节介绍了表 5-1 中所列出的不同异常事件源 （异常事件编号 1 到 15）。

5.4.1 复位异常事件

在 PSoC 4 中，器件复位被视为一个异常事件。该异常始终被使能，固定优先级为 –3 （即最高优先级）。器件复位可以由不同的原因 （如：上电复位 （POR）、 XRES 引脚上的外部复位信号以及看门狗复位）引起。在复位器件时，将在监控只读存储器 （SROM）范围外执行用于配置器件的初始引导代码。 SROM 存储器中的引导代码和其他数据均由赛普拉斯编程，并且外部用户不能对其进行读 / 写访问。在完成 SROM 引导序列后， CPU 代码执行将跳转到闪存存储器。闪存存储器地址 0x00000004 （表 5-1 中的异常事件编号 1）是启动代码在闪存存储器中的位置。 CPU 从该地址开始执行代码。注意，不能使用 SRAM 向量表中的复位异常事件地址，因为器件退出了复位后会选择闪存向量表。只有在取消激活复位后将寄存器配置作为闪存中的一部分启动代码，才能通过该寄存器配置选用 SRAM 向量表。

5.4.2 不可屏蔽中断 （NMI）异常事件

除复位外，不可屏蔽中断（NMI）的优先级最高的异常事件。该异常始终被使能，固定优先级为 2。在 PSoC 4 中，有两种触发 NMI 异常事件的方法：

- **通过置位 NMIPENDSET 位引起的 NMI 异常事件 （用户 NMI 异常事件）：** 通过置位中断控制状态寄存器（CM0_ICSR 寄存器）中的 NMIPENDSET 位，可以使用软件触发 NMI 异常事件。通过置位该位，可以执行运行向量表 （闪存或 SRAM 向量表）所指向的 NMI 处理器。
- **系统调用 NMI 异常事件：** 该异常事件用于 PSoC 4 中的非易失性编程操作，如闪存写入操作和闪存校验和操作。该异常事件是通过置位 CPUSS_SYSREQ 寄存器中的 SYSCALL_REQ 位触发的。通过 SYSCALL_REQ 位触发的 NMI 异常事件始终执行 SROM 中的 NMI 异常事件处理器代码 — 闪存或 SRAM 异常事件向量表不适用于系统调用的 NMI 异常事件。不能对 SROM 中的 NMI 处理器代码进行读 / 写操作，因为用户不能修改该代码包含的非易失性编程子程序。

5.4.3 HardFault 异常事件

HardFault 是正常或异常处理过程中因发生错误而导致的异常事件。该异常事件始终被使能。HardFault 的固定优先级为 1，因此它的优先级高于其他所有异常事件可配置的优先级。HardFault 异常事件是包含了不同故障条件类型（包括执行未定义的指令和访问无效的存储器地址）的综合异常事件。CM0 CPU 虽然不为 HardFault 异常处理器提供故障状态的信息，但在软件能从故障恢复的情况下，它允许处理器返回异常事件，并持续运行。

5.4.4 管理程序调用 （SVCall）异常事件

当 SVC 被视为应用代码的一部分并被 CPU 执行时，会引起管理程序调用（SVCall）异常事件。该异常事件始终被使能。应用软件使用 SVC 指令来调用操作系统。该调用操作被称为管理程序调用。 SVC 指令允许应用生成一个管理程序调用，用以请求对系统的访问。注意， PSoC 4 中的 CM0 将一个特权模式用于系统调用 NMI 异常事件 （与 SVCall 异常事件无关）。（有关特权模式的详细信息，请参考第 63 页上的芯片工作模式章节。）在 PSoC 4 中的架构级别上，不支持 SVCall 的其他任何特权模式。因此，应用开发者必须根据最终应用要求来定义 SVCall 异常处理器。

通过写入到系统处理器优先级寄存器 2 （SHPR2）中的两个位字段 PRI_11[31:30]，来配置 SVCall 异常的优先级 （范围为 0 到 3）。当执行 SVC 指令时， SVCall 异常将进入挂起状态，并等待 CPU 进行处理。通过系统处理器控制和状态寄存器 （SHCSR）中的 SVCALLPENDED 位，可以检查或修改 SVCall 异常的挂起状态。

5.4.5 PendSV 异常事件

PendSV 是与 SVCall 相似的另一类管理程序调用异常事件。PendSV 始终被使能，并且其优先级是可配置的。通过置位中断控制状态寄存器 CM0_ICSR 中的 PENDSVSET 位，可以触发 PendSV 异常。当置位该位时， PendSV 异常会进入挂起状态，并等待 CPU 的处理。通过清除中断控制状态寄存器（ CM0_ICSR）中的 PENDSVCLR 位，可以清除 PendSV 异常挂起状态。通过写入到系统处理器优先级寄存器 3（CM0_SHPR3）中的两个位字段 PRI_14[23:22]，可以配置 PendSV 异常的优先级（范围为 0 到 3）。更多有关信息，请参考 ARMv6-M 架构参考手册。

5.4.6 SysTick 异常事件

PSoC 4 中的 CM0 CPU 支持系统定时器（亦即 SysTick）作为其内部架构的一部分。SysTick 为 RTOS 计时定时器、高速警报定时器、简单计数器等计时工具提供了一个简单的 24 位递减计数器。可配置 SysTick 定时器，使其计数到零时生成一个中断，亦即 SysTick 异常事件。通过置位 SysTick 控制和状态寄存器 （CM0_SYST_CSR）中的 TICKINT 位，可以使能该异常事件。通过写入到系统处理器优先级寄存器 3（SHPR3）中的两个位字段 PRI_15[31:30]，可以配置 SysTick 异常的优先级（范围为 0 到 3）。可随时通过向中断控制状态寄存器 CM0_ICSR 中的 PENDSTSET 位写入 1，用以在软件中生成 SysTick 异常。同样，也可通过将 1 写入到中断控制状态寄存器 CM0_ICSR 中的 PENDSTCLR 位来清除 SysTick 异常的挂起状态。

5.5 中断源

PSoC 4 支持来自外设的 9 个中断（IRQ0 - IRQ8 或异常编号 16 到 24）。表 5-3 介绍了所有中断源。 PSoC 4 为所有 9 个中断线提供了灵活的源选项。中断包括来自片上外设的标准中断，如 TCPWM、串行通信模块、CSD 模块以及端口的中断。通常通过对外设的不同状态进行 “ 或 ” 逻辑运算来生成中断。需要在执行 ISR 时读取外设状态寄存器，以检测生成中断的条件。这些中断通常是由电平触发的，所以需要通过在执行 ISR 时读取外设状态寄存器来清除该中断。如果执行 ISR 时未读取状态寄存器，该中断将保持激活状态，并且 CPU 将连续执行 ISR。

表 5-2. PSoC 4 中断源的列表

中断编号	Cortex-M0 异常编号	中断源
NMI （请参见第 33 页上的异常事件源 ）	2	–
IRQ0	16	GPIO 中断 - 端口 0
IRQ1	17	GPIO 中断 - 端口 1
IRQ2	18	GPIO 中断 - 端口 2
IRQ3	19	GPIO 中断 - 端口 3
IRQ4	20	WDT （看门狗定时器）或温度
IRQ5	21	SCB （串行通信模块）
IRQ6	22	SPC （系统性能控制器）
IRQ7	23	CSD （Capsense 模块计数器溢出中断）
IRQ8	24	TCPWM0 （定时器 / 计数器 /PWM 0）

请参考第 45 页上的 I/O 系统章节，了解 GPIO 中断的详细信息。

5.6 异常事件优先级

当 CPU 需要处理多个异常时，可通过异常优先级实现异常仲裁。在 PSoC 4 中，能够灵活地为不同异常选择优先级值。除复位、 NMI 和硬故障外，可对其他所有异常分配可配置的优先级别。复位、 NMI 和硬故障异常有自己的固定优先级，分别为 –3、–2 和 –1。在 PSoC 4 中，编号越低，优先级越高。因此复位、 NMI 和硬故障异常的优先级最高。其他异常的优先级可配置范围为 0~3。

PSoC 4 支持嵌套异常，这样优先级更高的异常可优先执行于（中断）当前运行的异常处理器。如果下一个异常与当前运行的异常有相同的优先级，那么该优先执行性能无效。处理完优先级高的异常后，CPU 会恢复执行优先级低的异常。PSoC 4 中的 CM0 CPU 最多允许嵌套四个异常。当 CPU 接收到两个或多个优先级相同的异常请求，将先执行编号最低的异常。

第33页上的异常事件源 中介绍了用于配置优先级的异常编号 1 到 15 的寄存器。

通过对中断优先级寄存器 （CM0_IPR）进行写操作，可以配置 9 个中断（IRQ0 - IRQ8）的优先级。该组寄存器包括四个 32 位寄存器，每个寄存器用于保存四个中断的优先级值，如

表 5-3 所介绍。该寄存器的其他位字段不被使用。

表 5-3. 中断优先级寄存器位的定义

位	名称	说明
7:6	PRI_N0	中断编号 N 的优先级。
15:14	PRI_N1	中断编号 N+1 的优先级。
23:22	PRI_N2	中断编号 N+2 的优先级。
31:30	PRI_N3	中断编号 N+3 的优先级。

5.7 使能 / 禁用中断

NVIC 提供了寄存器来使用软件单独使能和禁用 9 个中断。如果未使能某个中断，则 NVIC 将不会处理该中断线上的中断请求。中断设置使能寄存器 （CM0_ISER）和中断清除使能寄存器 （CM0_ICER）分别用于使能和禁用中断。这些寄存器的宽度为 32 位，其中每个位相应于编号与之相同的中断。还可以通过软件读取该寄存器，以获取中断的使能状态。表 5-4 显示了两个寄存器的访问属性。注意，向这些寄存器写入零不会产生任何影响。

表 5-4. 中断使能 / 禁用寄存器

寄存器	操作	位值	注释
中断设置使能寄存器（CM0_ISER）	写	1	使能中断
		0	不受影响
	读	1	中断使能
		0	中断禁用
中断清除使能寄存器（CM0_ICER）	写	1	禁用中断
		0	不受影响
	读	1	中断使能
		0	中断禁用

CM0_ISER 和 CM0_ICER 寄存器仅适用于中断 （IRQ0 - IRQ8）。不能使用它们使能或禁用编号为 1 到 11 的异常 15 个异常具有它们自己的使能和禁用支持，如 第 33 页上的异常事件源 所述。

Cortex-M0 （CM0） CPU 中的 PRIMASK 寄存器可作为全局异常使能寄存器使用。无论异常事件的使能情况如何，该寄存器都可以对所有可配置优先级的异常事件进行掩码。除表 5-1 中介绍的复位、 NMI 和硬故障等异常事件外，其他所有的异常事件的优先级都可配置。可将它们的优先级配置为 0 到 3，其中 0 的优先级最高，3 的优先级最低。当置位 PRIMASK 寄存器中的 PM 位 （位 0）时， CPU 不会处理任何可配置优先级的异常。但这些异常能处于挂起状态，等到 PM 位被清除后， CPU 将对其进行处理。

5.8 异常事件的状态

每个异常都能处在下面各状态之一。

表 5-5. 异常事件的状态

异常事件的状态	含义
非活动	该异常事件即处于非活动状态又不被挂起。该异常事件被禁用，或者使能后的异常事件未被触发。
挂起	CPU/NVIC 已接收到异常事件请求，且该异常事件正在等待 CPU 对其进行处理。
活动	CPU 正在执行异常事件，但该事件的处理器操作尚未完成。优先级更高的异常事件可以中断优先级更低的异常事件的执行。在这种情况下，该两个异常事件都处于活动状态。
活动和挂起	处理器正在处理异常事件。与此同时，另一个来自相同源的异常事件请求正在挂起。

中断控制状态寄存器 （CM0_ICSR）包含用于描述各种异常状态的状态位。

- CM0_ICSR 中的 VECTACTIVE 位（[8:0]）用于存储当前执行的异常事件的编号。如果 CPU 尚未执行任何异常处理器 （CPU 处于线程模式），则该值为零。注意， VECTACTIVE 位字段中的值与中断编程状态寄存器 （IPSR）中位 [8:0] 内的值相同。 IPSR 还用来保存有效异常的编号。
- CM0_ICSR 寄存器中的 VECTPENDING 位（[20:12]）保存了优先级最高的挂起异常的编号。如果没有任何挂起异常，则该值为零。
- CM0_ICSR 寄存器中的 ISRPENDING 位（位 22）表示由 NVIC 生成的中断 （IRQ0 到 IRQ8）是否处于挂起状态。

5.8.1 挂起异常事件

当外设为 NVIC 生成一个中断请求信号，或发生某个异常事件时，相应的异常事件会进入挂起状态。当 CPU 开始执行相应的异常处理器子程序时，该异常会从挂起状态转为活动状态。

通过使用独立的寄存器位可设置和清除中断的挂起状态， NVIC 允许软件挂起 9 个中断线。中断设置挂起寄存器（CM0_ISPR）和中断清除挂起寄存器 （CM0_ICPR）分别用于设置和清除中断线的挂起状态。这些寄存器的宽度为 32 位，每个位对应于编号与之相同的中断。表 5-6 显示了这两个寄存器的访问属性。注意，向这些寄存器写入零不会产生任何影响。

表 5-6. 中断设置挂起 / 清除挂起寄存器

寄存器	操作	位值	注释
中断设置挂起寄存器（CM0_ISPR）	写	1	使某个中断进入挂起状态
		0	不受影响
	读	1	中断正在挂起
		0	中断没有挂起

表 5-6. 中断设置挂起 / 清除挂起寄存器

中断清除挂起寄存器（CM0_ICPR）	写	1	清除挂起的中断。
		0	不受影响
	读	1	中断正在挂起
		0	中断没有挂起

在挂起位已被置位时再次对该位进行设置，但中断处理器只会执行一次。不管相应的中断是否被使能，仍会更新挂起位。如果中断未被使能，则中断线不会转换为挂起状态，直到该中断通过对 CM0_ISER 寄存器进行写操作得到使能为止。

注意，CM0_ISPR 和 CM0_ICPR 寄存器仅用于 9 个外设中断（异常编号为 16 到 47）。不能使用它们来挂起编号为 1 到 11 的异常。这 15 个异常具有自己的挂起支持，如 第 33 页上的异常事件源 所述。

5.9 异常事件的堆栈使用情况

当 CPU 执行主代码（在线程模式中）并且某个异常请求出现时，CPU 会将其通用寄存器的状态信息存储到堆栈内。然后，CPU 开始执行相应的异常处理器（在处理器模式中）。CPU 将八个 32 位内部寄存器的内容推移到堆栈上。这些寄存器包括编程和状态寄存器（PSR）、返回地址、链接寄存器（LR 或 R14）、R12、R3、R2、R1 和 R0。Cortex-M0 有两个堆栈指针 — MSP 和 PSP。一次只能激活一个堆栈指针。在线程模式下，控制寄存器中的活动堆栈指针位用于定义当前的活动堆栈指针。在处理器模式下， MSP 始终作为堆栈指针使用。Cortex-M0 中的堆栈指针始终向下移动并指向最后转移的数据的地址。

当 CPU 处在线程模式中，并某个异常请求出现时，CPU 将使用控制寄存器中定义的堆栈指针保存通用寄存器的内容。堆栈推移操作完成后， CPU 将进入处理器模式，以执行异常处理器。执行当前异常期间，如果出现比该优先级更高的异常，会使用 MSP 实现堆栈推移 / 弹出操作，这是因为 CPU 已经处于处理器模式。有关详细信息，请参考第 25 页上的 Cortex-M0 CPU 章节中介绍的内容。

Cortex-M0 使用两个技术 （即末尾连锁和迟到）来缩短服务异常的延迟。对于外部用户，这些技术是不可见的。它们仅作为内部处理器架构的一部分（http://infocenter.arm.com/help/topic/com.arm.doc.ddi0419c/index.html）。

5.10 中断和低功耗模式

在 PSoC 4 中，当生成某个外设中断请求时，器件可以从低功耗模式唤醒。当一个或更多唤醒源生成中断信号时，唤醒中断控制器 （WIC）模块会生成一个唤醒信号，以使器件进入活动模式。器件进入活动模式后，将执行外设中断的中断处理器。

由 CM0 CPU 执行的等待中断 （WFI）指令可使器件进入睡眠模式 和深度睡眠模式。第 65 页上的功耗模式章节中说明了进入不同低功耗模式的序列。芯片的低功耗模式有两类功能固定的中断源：

- 固定功能中断源（例如，看门狗定时器中断， I2C 中断和 GPIO 中断）仅用于活动模式和深度睡眠模式
- 仅用于活动模式的功能固定中断源 （其他所有的功能固定中断）

5.11 异常事件 — 初始化和配置

本节介绍了 PSoC 4 中初始化和配置异常事件的不同步骤。

1. 配置异常向量表位置：使用异常事件的第一步是配置向量表位置 — 确定该向量表是在闪存存储器中还是在 SRAM 中。通过将 ‘1’（SRAM 向量表）或 ‘0’（闪存向量表）写入到 CPUSS_CONFIG 寄存器中的 VECT_IN_RAM 位字段 （位 0）内，可以实现该配置。对该寄存器进行写操作被视为器件初始化代码的一部分。

   如果应用程序需要动态改变向量地址，建议该向量表在 SRAM 中可用。如果该表位于 Flash 内，则需要通过对 Flash 进行写操作来修改向量表中的内容。默认情况下，PSoC Creator IDE 使用的是 SRAM 中的向量表。
2. 配置单独异常：下一步是对应用所需要的单独异常进行配置。
   a. 配置异常或中断源；配置操作包括设置中断生成条件。根据特定异常的需要，寄存器的配置会不一样。
   b. 定义异常处理器功能并将该功能的地址写入异常向量表。表 5-1 介绍了异常向量表的格式；需要将异常处理器地址写入表中合适的异常编号入口。
   c. 设置异常的优先级，如第 34 页上的异常事件优先级所述。
   d. 使能该异常，如第 35 页上的使能 / 禁用中断 所述。

5.12 寄存器

表 5-7. 寄存器列表

寄存器名称	说明
CM0_ISER	中断设置使能寄存器
CM0_ICER	中断清除使能寄存器
CM0_ISPR	中断设置挂起寄存器
CM0_ICPR	中断清除挂起寄存器
CM0_IPR	中断优先级寄存器
CM0_ICSR	中断控制状态寄存器
CM0_AIRCR	应用中断和复位控制寄存器
CM0_SCR	系统控制寄存器
CM0_CCR	配置和控制寄存器
CM0_SHPR2	系统处理器优先级寄存器 2
CM0_SHPR3	系统处理器优先级寄存器 3
CM0_SHCSR	系统处理器控制和状态寄存器
CM0_SYST_CSR	Systick 控制和状态
CPUSS_CONFIG	CPU 子系统配置
CPUSS_SYSREQ	系统请求寄存器

5.13 相关文档

- ARMv6-M 架构参考手册 — 本文档介绍了 ARM Cortex-M0 架构，包括指令集、 NVIC 架构和 CPU 的寄存器说明。

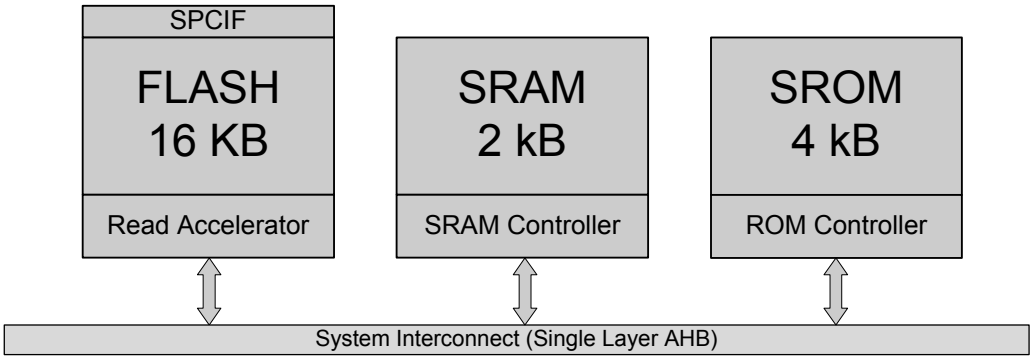
Section C: 存储器系统

本章节包括以下小节：

■ 第 41 页上的存储器映射章节

顶层架构

存储器系统框图



6. 存储器映射

通过 CPU 或 （大部分情况下）通过调试系统，可以访问所有的 PSoC® 4 存储器 （包括闪存、 SRAM 和 SROM）和寄存器。本节介绍了存储器和寄存器地址的总体映射情况。

6.1 特性

PSoC 4 存储器系统的特性包括：

- 16K 字节闪存、 2K 字节 SRAM
- 包含引导和配置子程序的 4K 字节 SROM
- ARM Cortex-M0 32 位的线性地址空间，包括用于代码、 SRAM、外设以及 CPU 内部寄存器的地址范围
- 闪存被映射到 Cortex-M0 代码区
- SRAM 被映射到 Cortex-M0 SRAM 区
- 外设寄存器被映射到 Cortex-M0 外设区
- Cortex-M0 专用外设总线 （PPB）区域包括 CPU 内核中执行的寄存器，如 NVIC、 SysTick 定时器和固定功能模块（SCB）。更多有关信息，请参阅第 25 页上的 Cortex-M0 CPU 章节 。

6.2 工作原理

以下各表详细介绍了 PSoC 4 的存储器映射情况。更多有关信息，请参考 PSoC 4 寄存器技术参考手册 （TRM）。

Cortex-M3 具有固定的地址映射，因此可通过简单的存储器访问指令来访问存储器和外设。 32 位 （4 GB）地址空间分为多个范围，如表 6-1 所示。注意，可从代码区和 SRAM 区执行代码。

表 6-1. Cortex-M0 地址映射情况

地址范围	名称	使用说明
0x00000000 – 0x1FFFFFFF	代码	编程代码的可执行范围。可以在此范围内放置数据。它包括从地址 0 开始的异常向量表。
0x20000000 – 0x3FFFFFFF	SRAM	数据的可执行范围。可以在此范围内放置代码。
0x40000000 – 0x5FFFFFFF	外设	所有的外设寄存器。不能在此范围外执行代码。
0x60000000 – 0xDFFFFFFF	–	未使用
0xE0000000 – 0xE00FFFFF	PPB	CPU 内核的外设寄存器。
0xE0100000 – 0xFFFFFFFF	芯片	PSoC 4 的专属空间

表 6-2 显示了 PSoC 4 的地址映射情况。

表 6-2. PSoC 4 地址映射情况

地址范围	使用说明
0x00000000 - 0x00003FFF	16 KB 闪存
0x0FFFF000 - 0x10000000	4 KB 只读管理闪存
0x20000000 - 0x200007FF	2 KB SRAM
0x40100000 - 0x4011FFFF	CPU 子系统寄存器
0x40020000 - 0x40023FFF	I/O 端口控制 （高速 I/O 矩阵）寄存器
0x40040000- -0x40043FFF	I/O 端口寄存器
0x40050000- -0x4005FFFF	TCPWM 寄存器
0x40060000- -0x4006FFFF	固定功能 I2C 寄存器
0x40080000- -0x4008FFFF	CapSense 寄存器
0x40030000- -0x4003FFFF	电源、时钟、复位控制寄存器
0xE0000000 - 0xE00FFFFF	Cortex-M0 PPB 寄存器
0xF0000000 - 0xF0000FFF	CoreSight ROM

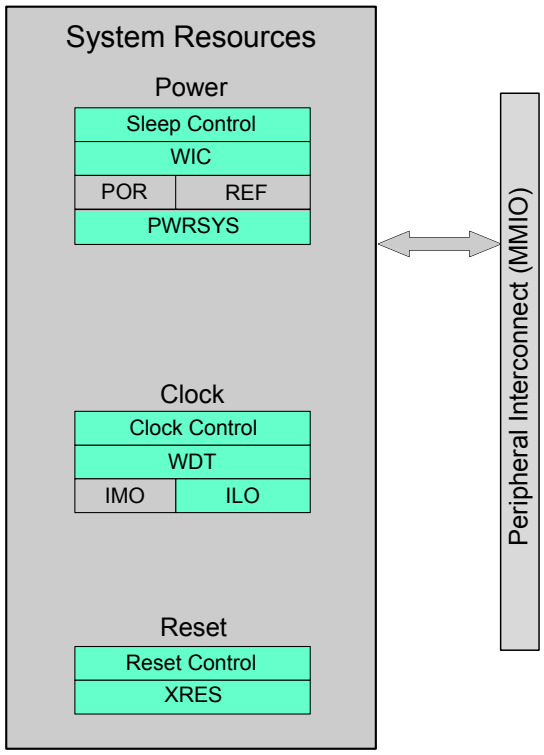
Section D: 系统资源

本章节包括以下小节：

- 第 45 页上的 I/O 系统章节
- 第 53 页上的时钟系统章节
- 第 59 页上的电源与电源监测章节
- 第 63 页上的芯片工作模式章节
- 第 65 页上的功耗模式章节
- 第 69 页上的看门狗定时器章节
- 第 73 页上的复位系统章节
- 第 75 页上的器件安全章节

顶层架构

系统资源框图



7. I/O 系统

本章节介绍了 PSoC® 4 I/O 系统以及其特性、架构、工作模式和中断。PSoC 4 中的各通用 I/O（GPIO）引脚被分为不同端口；每端口可包含最多 8 个 GPIO。 CY8C4000 系列最多有 20 个 GPIO，分成四个端口。

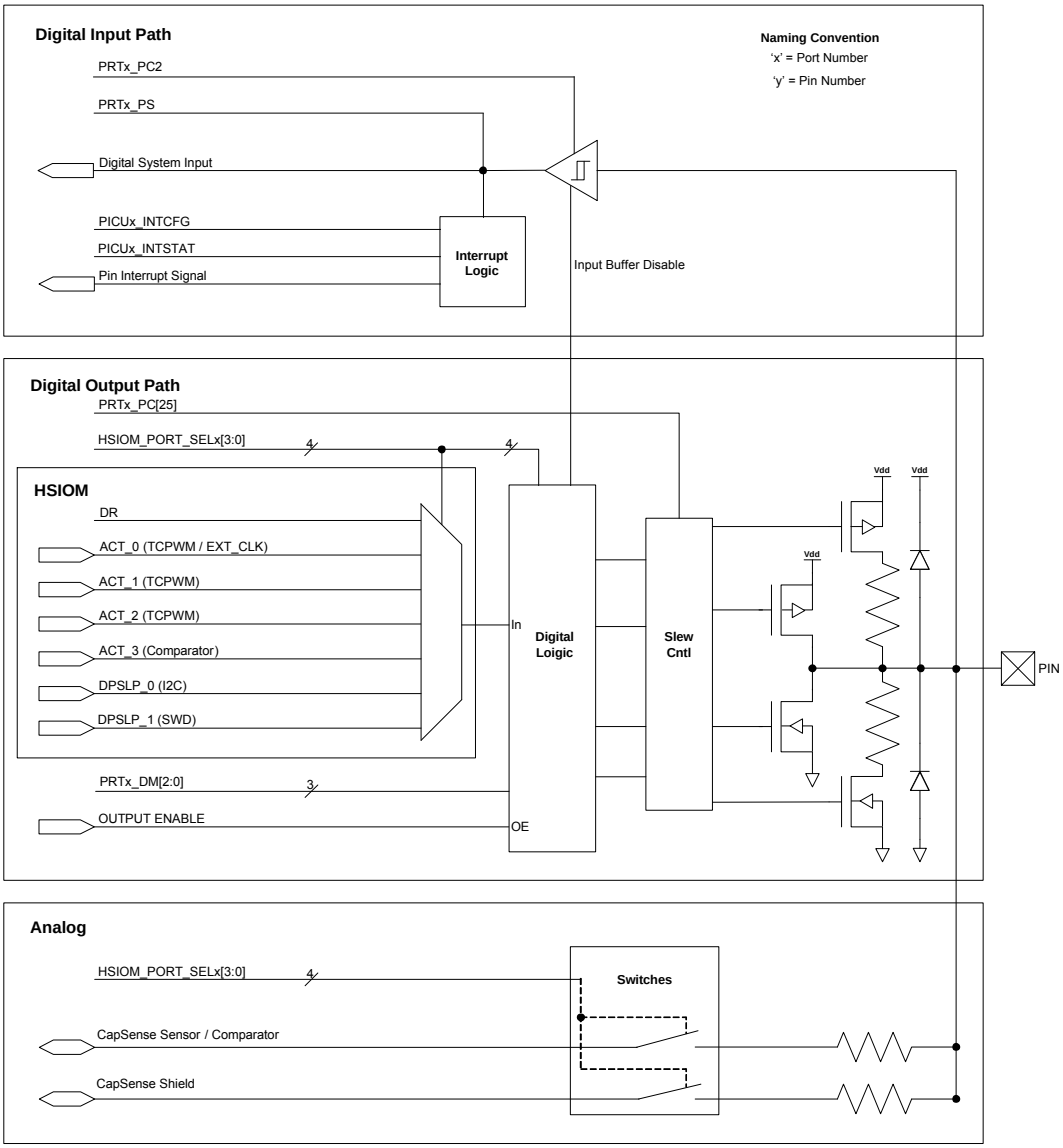
7.1 特性

PSoC 4 GPIO 具有以下特性：

- 支持模拟 / 数字输入输出
- 支持 CapSense
- 在数字模式下，灌电流可达 8 mA，拉电流可达 4 mA
- 单独的端口读 （PS）和写 （DR）数据寄存器，能够避免发生 “ 读取 - 修改 - 写入 ” 错误
- 支持边沿触发中断，包括上升沿、下降沿和双边沿等触发方式
- 摆率控制
- 可选择输入阈值：CMOS 输入或 LVTTL 输入

7.2 框图

图 7-1. GPIO 框图



注意： 该图像中显示的 GPIO 特性并非适用于所有引脚。更多信息，请参考表 7-3 的内容。

7.3 GPIO 驱动模式

可将每个 I/O 单独配置为表 7-1 中罗列的八种驱动模式中的一种。图 7-2 为简便的引脚框图，它显示的是基于八个驱动模式的引脚视图。

使用两个端口配置寄存器来配置 PSoC 4 中的 GPIO：端口配置寄存器（GPIO_PRTx_PC）和端口辅助配置寄存器（GPIO_PRTx_PC2）。所有 PSoC 4 端口均有专用的 GPIO_PRTx_PC 和 GPIO_PRTx_PC2 寄存器。要想了解每个端口的寄存器，请参考表 7-5。

GPIO_PRTx_PC 可用于配置下述的每个端口的属性：

- 每个引脚的输出驱动模式
- 每个引脚的输入缓冲器模式
- 端口的摆率（请参考第 48 页上的摆率控制 中的内容）
- 端口的输入阈值选择（请参考第 48 页上的 CMOS LVTTL 电平控制 中的内容）

GPIO_PRTx_PC2 用于配置端口上每个引脚的输入缓冲器，无论 GPIO_PRTx_PC 中的驱动模式配置如何。当模拟信号位于该引脚上时，可以使用 GPIO_PRTx_PC2 禁用输入缓冲器。

图 7-2. I/O 驱动模式框图

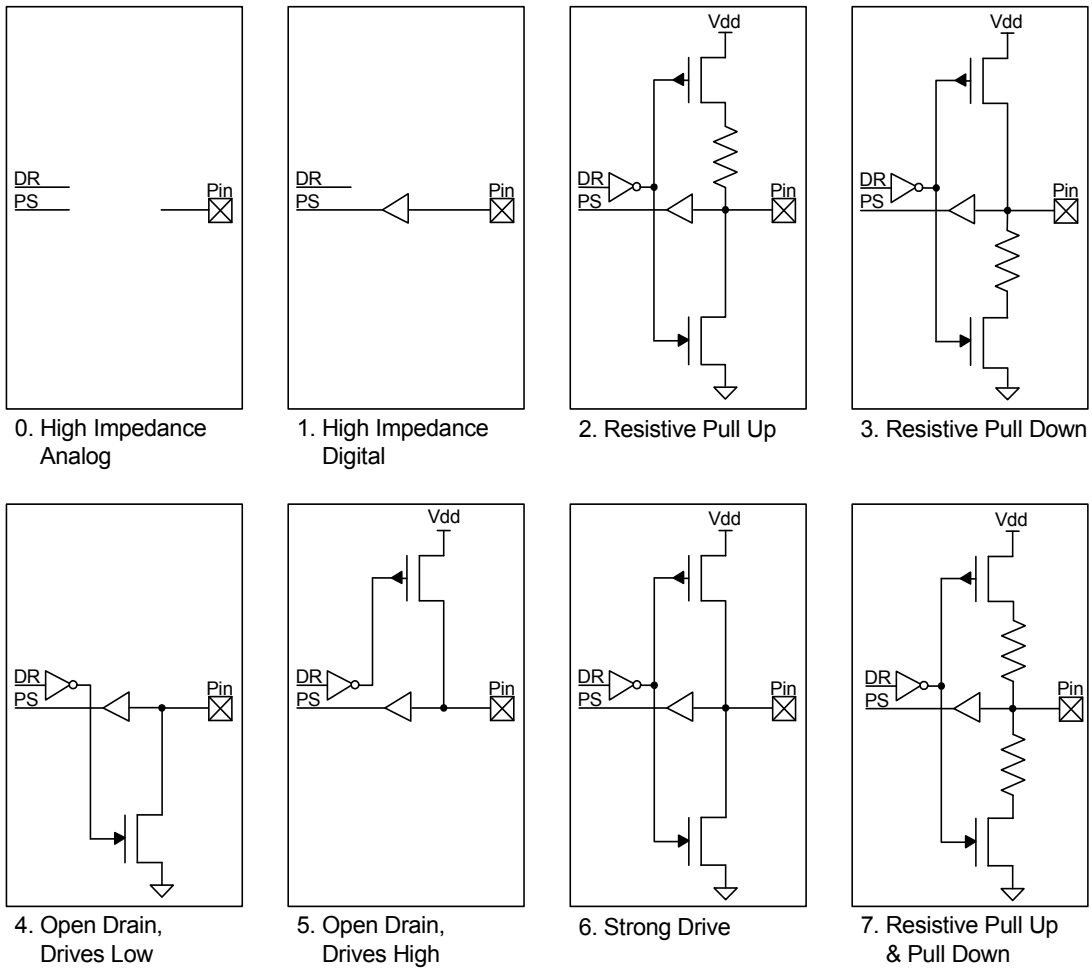


表 7-1. 驱动模式设置

GPIO_PRTx_PC （‘x’ 表示端口编号， ‘y’ 表示引脚编号）				
位	驱动模式	数值	数据 = 1	数据 = 0
3y+2: 3y	SEL’ y’	选择引脚 ‘y’ 的驱动模式 （0≤ y ≤ 7）		
	高阻模拟	0	高阻态	高阻态
	高阻数字	1	高阻态	高阻态
	电阻上拉	2	弱 1	强 0
	电阻下拉	3	强 1	弱 0
	开漏低电平驱动	4	高阻态	强 0
	开漏高电平驱动	5	强 1	高阻态
	增强驱动	6	强 1	强 0
	电阻上拉和下拉	7	弱 1	弱 0

表 7-2. 输入缓冲器禁用 （端口配置 2）

GPIO_PRTx_PC2 （‘x’ 表示端口编号， ‘y’ 表示引脚编号）		
位	名称	说明
7:0	INP_DIS	禁用独立于端口控制驱动模式的输入缓冲器。当模拟信号位于引脚上时，该位应置位。

7.3.1 高阻模拟

高阻模拟模式是默认的复位状态；输出驱动器和数字输入缓冲器均关闭。这种状态可以防止外部电压产生流入数字输入缓冲器的电流。对于悬空引脚或支持模拟电压的引脚，推荐使用该驱动模式。高阻模拟引脚不可用于数字输入。无论数据寄存器值如何，读取该引脚状态寄存器均返回 0X00 值。

为了实现在低功耗模式中的最低器件电流，必须配置未使用的 GPIO 为高阻模拟模式。

7.3.2 高阻数字

高阻数字模式是建议用于数字输入的标准高阻抗 （High Z） 状态。在这种状态中，会针对数字输入信号使能输入缓冲器。

7.3.3 电阻上拉或电阻下拉

电阻模式是指在某一种数据状态下提供串联电阻，并在另一种数据状态下提供增强驱动。在这些模式下，引脚可用于数字输入或数字输出。如果要求电阻上拉，请将 ‘1’ 写入引脚的数据寄存器位。如果要求电阻下拉，请将 ‘0’ 写入引脚的数据寄存器位。机械开关是这些驱动模式的常见应用。当 PSoC 与开漏的驱动线相连接时，可使用这些驱动模式。当输入为开漏低电平或开漏高电平时，分别使用电阻上拉或电阻下拉。

7.3.4 开漏高电平驱动和开漏低电平驱动

开漏模式在一种数据状态下提供高阻抗，在另一种数据状态下提供增强驱动。在这些模式下，引脚可作为数字输入或输出使用。因此，这些模式广泛使用于双向数字通信。当信号采用外部上拉时，使用开漏高电平驱动模式；当信号采用外部下拉时，则使用开漏低电平驱动。

开漏低电平驱动模式的常见应用是驱动 I^2C 总线信号线。

7.3.5 增强驱动

增强驱动模式是各引脚的标准数字输出模式；在高电平和低电平状态下它都可提供增强型 CMOS 的输出驱动。一般情况下，增强驱动模式的引脚不能作为输入使用。通常将这种模式用于数字输出信号或驱动外部晶体管。

7.3.6 电阻上拉和电阻下拉

该模式类似于7.3.3 电阻上拉或电阻下拉部分中所述的驱动模式。在电阻上拉和电阻下拉模式下， GPIO 的逻辑 1 和逻辑 0 输出状态均有串联电阻。高电平数据被上拉，而低电平数据被下拉。当其它信号驱动总线可能导致短路时，可使用该模式。

7.4 摆率控制

GPIO引脚的增强驱动模式提供了快速和慢速摆率选项；通过 GPIO_PRTx_PC[25] 位可配置该选项。可以单独配置每个端口的摆率。默认情况下，该位被清除，并且端口运行在快速摆率模式。如果需要较低的摆率，则可以设置该位。对于频率超过 1 MHz 的信号，推荐使用快速摆率。摆率越慢， EMI 和串扰越少；因此对速度要求不是很关键 （通常小于 1 MHz） 的信号，推荐使用该选项。

7.5 CMOS LVTTL 电平控制

I/O 引脚可在两种电压电平模式下运行。通过写入 GPIO_PRTx_PC[24] 位，可选择这些电压电平。

每个端口可以单独配置输入电平。默认情况下，该位被清除并

端口在 CMOS 模式下运行。可以设置该位，重新配置端口为 LVTTL 模式。

大多数情况下，可使用 CMOS 模式，而 LVTTL 可用于自定义接口要求，可运行于低电平模式。

7.6 高速输入 / 输出矩阵

高速输入 / 输出矩阵（HSIOM）是一组将 GPIO 路由到 PSoC 中的资源的高速开关。这些资源包括 CapSense、TCPWM、I2C 以及 CPU。HSIOM 为引脚选择活动和深度睡眠的电压范围。 HSIOM_PORT_SELx 是宽度为 32 位的寄存器，控制 GPIO 的路由。每个寄存器控制一个端口；端口中每个 GPIO 占用四个专用位。它提供最多 16 个不同的 GPIO 路由选项。不同的选择提供不同的引脚功能，如表 7-3 中所示。

表 7-3. HSIOM 端口设置

HSIOM_PORT_SELx （'x' 表示端口编号， 'y' 表示引脚编号）			
位	名称 （SEL 'y'）	数值	说明 （选择引脚 'y' 的源 0 £ y £ 7））
4y+3 : 4y	DR	0	引脚为可软件控制的常规 GPIO。
	CSD_SENSE	4	引脚为 CSD 检测引脚 （模拟模式）。
	CSD_SHIELD	5	引脚为 CSD 屏蔽引脚 （模拟模式）。
	AMUXA	6	引脚被连接至 AMUXBUS-A。
	AMUXB	7	引脚被连接至 AMUXBUS-B。该模式也可用于 GPIO 的槽电容预充电。
	ACT_0 (TCPWM/ EXT_CLK)	8	活动模式源的专用引脚 #0：用于外部时钟输入和 TCPWM 溢出输出。
	ACT_1 (TCPWM)	9	活动模式源的专用引脚 #1：用于 TCPWM 输入 / 输出信号。
	ACT_2 (TCPWM)	10	活动模式源的专用引脚 #2：用于 TCPWM 下溢输出。
	ACT_3 (CSD_COMP)	11	活动模式源的专用引脚 #3：用于直接连接到 CSD 比较器。
	DPSLP_0	14	深度睡眠模式源的专用引脚 #0：用于 I2C 连接。
	DPSLP_1	15	深度睡眠模式源的专用引脚 #1：用于 SWD 连接。

注意： 这些特性并非适用于所有引脚。更多有关每个引脚所支持的特性信息，请查看 PSoC 4 数据手册中的内容。

7.7 固件控制 GPIO

请查看表 7-3，了解如何配置固件控制 GPIO 的 HSIOM 设置。GPIO_PRTx_DR 为数据寄存器，用于读写 GPIO 输出数据。对该寄存器进行写操作可修改 GPIO 输出为所写入的值。请注意，读操作反映的是写入该寄存器的输出数据，并不是 GPIO 的当前状态。使用该寄存器，在具有输入和输出 GPIO 端口上，可以安全执行读 - 修改 - 写序列。

除了数据寄存器外，还有两个其它寄存器，即 GPIO_PRTx-_DR_SET 和 GPIO_PRTx_DR_CLR。这两个寄存器用于设置或清除端口中特定 GPIO 的输出数据。此外，可以使用 GPIO_PRTx_INV 寄存器对特定 GPIO 的输出数据进行反转。

GPIO_PRTx_PS 为端口 I/O 焊盘寄存器；对它进行读操作时，它将提供 GPIO 的状态。无法对该寄存器进行写操作。

有关这些寄存器的更多信息，请查看 PSoC 4 寄存器的技术参考手册。

7.8 CapSense

可将支持 CSD 的引脚配置为 CapSense Widget，如按键、滑条元件、触摸板元件或接近传感器。CapSense 还需要外部槽电容和屏蔽线路。表 7-3 显示的是 CapSense 所需的 GPIO 和 HSIOM 设置。有关详细信息，请参见第 117 页上的 CapSense 章节 。

表 7-4. CapSense 设置

CapSense 引脚	GPIO 驱动模式 （GPIO_PRTx_PC）	数字输入缓冲器设置 （GPIO_PRTx_PC2）	HSIOM 设置
传感器	高阻模拟	禁用缓冲器	CSD_SENSE
屏蔽	高阻模拟	禁用缓冲器	CSD_SHIELD
CMOD （正常操作）	高阻模拟	禁用缓冲器	AMUXBUS A 或 CSD_-COMP
CMOD （GPIO 预充电，仅适用于选择 GPIO）	高阻模拟	禁用缓冲器	AMUXBUS B 或 CSD_-COMP
CSH TANK （GPIO 预充电，仅适用于选择 GPIO）	高阻模拟	禁用缓冲器	AMUXBUS B 或 CSD_-COMP

7.9 I/O 端口的重新配置

通过更改GPIO_PRTx_PC和HSIOM_PORT_SELx寄存器的值，可在运行时重新配置驱动模式和 GPIO。当引脚直接连接到数字外设时，请保持在重新配置引脚过程中各引脚的状态。如果端口由数据寄存器驱动，会自动保持状态。在配置端口过程中，当前的配置应如下保存：

1. 通过软件读取 GPIO_PRTx_PS 中的 GPIO 引脚状态。
2. 将 GPIO_PRTx_PS 值写入到 GPIO_PRTx_DR 数据寄存器。
3. 更高 HSIOM_PORT_SELx 中的相应字段，以通过 GPIO_PRTx_DR 数据寄存器驱动引脚。

注意: 配置 GPIO 连接到 AMUXBUS A/AMUXBUS B 时，使用 DSI 信号来控制 GPIO 的切换。因此，在 DSI 和 AMUXBUS 模式中，无法重新配置并使用 GPIO。

7.10 上电时的 I/O 状态

默认情况下，在上电过程中所有 GPIO 均处于高阻模拟状态，并输入缓冲器被禁用。当芯片通电时，可以根据所需应用的要求通过写入相应的寄存器来配置 GPIO。

7.11 低功耗模式中的行为

在睡眠模式下，GPIO 保持当前引脚状态。在睡眠模式下，所有 GPIO 都活动，并可由活动外设驱动，如 CapSense、TCPWM 和 I2C。

在深度睡眠模式下，除 I2C 引脚外，所有其它引脚的状态均被锁存，并保持引脚上的信号。但 I2C 引脚仍然工作，并在发生 I2C 地址匹配事件时唤醒处理器。GPIO 中断在深度睡眠模式下仍具有唤醒功能。

更多信息，请参考第 65 页上的功耗模式章节 、第 31 页上的中断章节 和第79页上的内部集成电路（I2C）章节 中的内容。

为了实现在低功耗模式中的最低器件电流，必须配置未使用的 GPIO 为高阻模拟模式。

7.12 GPIO 中断

本节介绍了 PSoC 4 GPIO 的中断功能。

7.12.1 特性

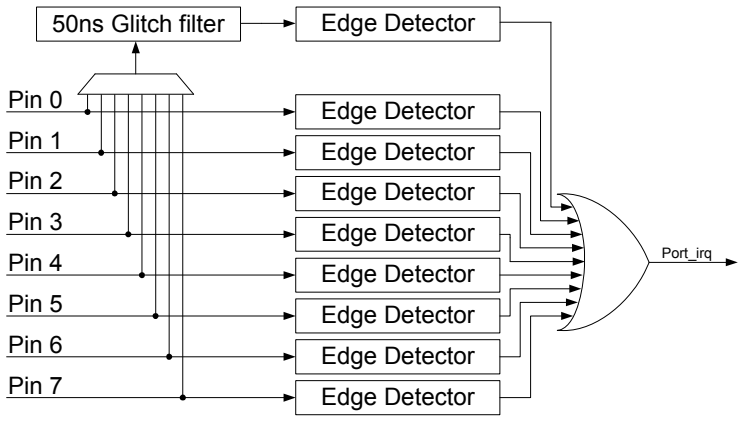
GPIO 中断的特性如下：

- 每个端口上的所有八个引脚均有专用中断和相应的中断向量。
- 通过引脚状态位，可以轻松确定中断源来自哪个引脚
- 上升沿、下降沿或两个边沿触发中断均被处理
- 可以单独使能或禁用引脚中断
- AHB 接口用于对其寄存器进行读和写操作
- 将端口中的所有 GPIO 驱动的单一端口中断请求（PIRQ）信号发送给中断控制器

7.12.2 中断控制器框图

每个端口均有其各自单独的中断请求和相应的中断请求（IRQ）以及中断服务子程序 （ISR）。另外，在每个端口上可以选择一个引脚。通过 50 ns 的干扰滤波器路由此引脚以形成端口的防干扰中断。详细信息如图 7-3 中所示。

图 7-3. 中断发生器



7.12.3 功能和配置

通过对 GPIO_PRTx_INTR_CFG 寄存器进行写操作，可以单独配置每个端口引脚，以便生成上升沿、下降沿和两个边沿上触发的中断。不支持电平敏感型中断。 GPIO_PRTx_INTR_CFG 还用于将特定的通道路由到干扰滤波器，以生成第九个防干扰中断。

当信号在中断使能端口引脚上触发 GPIO 中断时，将会更新 GPIO_PRTx_INTR 寄存器（中断状态寄存器）。固件可通过读取该寄存器来确定触发中断的 GPIO。然后，固件可通过将‘1’ 写入其相应位来清除 IRQ 位。

请按照下面步骤生成引脚中断：

1. 当在引脚上发生中断时，状态寄存器中的相应位便被设置为 ‘1’，并向中断控制器发生中断请求。
2. 向该位写入 ‘1’ 时，数值为 ‘1’ 的状态位将被清除。状态寄存器的其他位还可以对输入中断源发出响应。
3. 如果中断等待处理，并读取了状态寄存器，相同的中断源的所有输入事件 （I/O）均被阻塞，直到完成读取操作为止。

但是状态寄存器中无挂起的中断不会被阻塞。

此外，读取端口中断控制状态寄存器的同时，如果在相应端口上发生了中断，则可能不会正确检测该中断。因此，当使用 GPIO 中断时，推荐仅在相应的中断服务子程序中读取状态寄存器，并不会读取代码的任何其他部分。

更多详细信息，请参考 PSoC 4 寄存器的技术参考手册中 GPIO_PRTx_INTR_CFG 和 GPIO_PRTx_INTR 的内容。

7.13 寄存器

表 7-5. I/O 寄存器

名称	说明
GPIO_PRTx_DR	端口输出数据寄存器
GPIO_PRTx_DR_SET	端口输出数据设置寄存器
GPIO_PRTx_DR_CLR	端口输出数据清除寄存器
GPIO_PRTx_DR_INV	端口输出数据反转寄存器
GPIO_PRTx_PS	端口引脚状态寄存器 — 用于读取 I/O 的逻辑引脚状态
GPIO_PRTx_PC	端口配置寄存器 — 用于配置输出驱动模式、输入阈值以及转换速率
GPIO_PRTx_PC2	端口辅助配置寄存器 — 用于配置 I/O 引脚的输入缓冲器
GPIO_PRTx_INTR_CFG	端口中断配置寄存器
GPIO_PRTx_INTR	端口中断状态寄存器
HSIOM_PORT_SELx	HSIOM 端口选择寄存器

注意： 寄存器名称中的 ‘x’ 表示端口编号。例如， GPIO_PTR1_DR 为端口 1 的输出数据寄存器。

8. 时钟系统

PSoC® 4 时钟系统包括以下时钟资源：

- 两个内部时钟源：
  - 24 到 48 MHz 的内部主要振荡器 （IMO）IMO 和赛普拉斯提供的校准之间的容差为 ±2% （24 MHz 和 32 MHz）。
  - 32 kHz 的内部低速振荡器 （ILO）
- 通过使用 I/O 引脚的信号来生成的外部时钟 （EXTCLK）
- 从 IMO 或外部时钟选中的高达 16 MHz 的高频率时钟 （HFCLK）
  - HFCLK 专用预分频器
- 来自 ILO 的低频率时钟 （LFCLK）
- 来自 HFCLK 的频率高达 16 MHz 的系统时钟 （SYSCLK）专用预分频器
- 四个外设时钟，每个时钟都有一个 16 位分频器

8.1 框图

图 8-1 提供了 PSoC 4 器件中时钟系统的通用视图。

图 8-1. 时钟系统框图

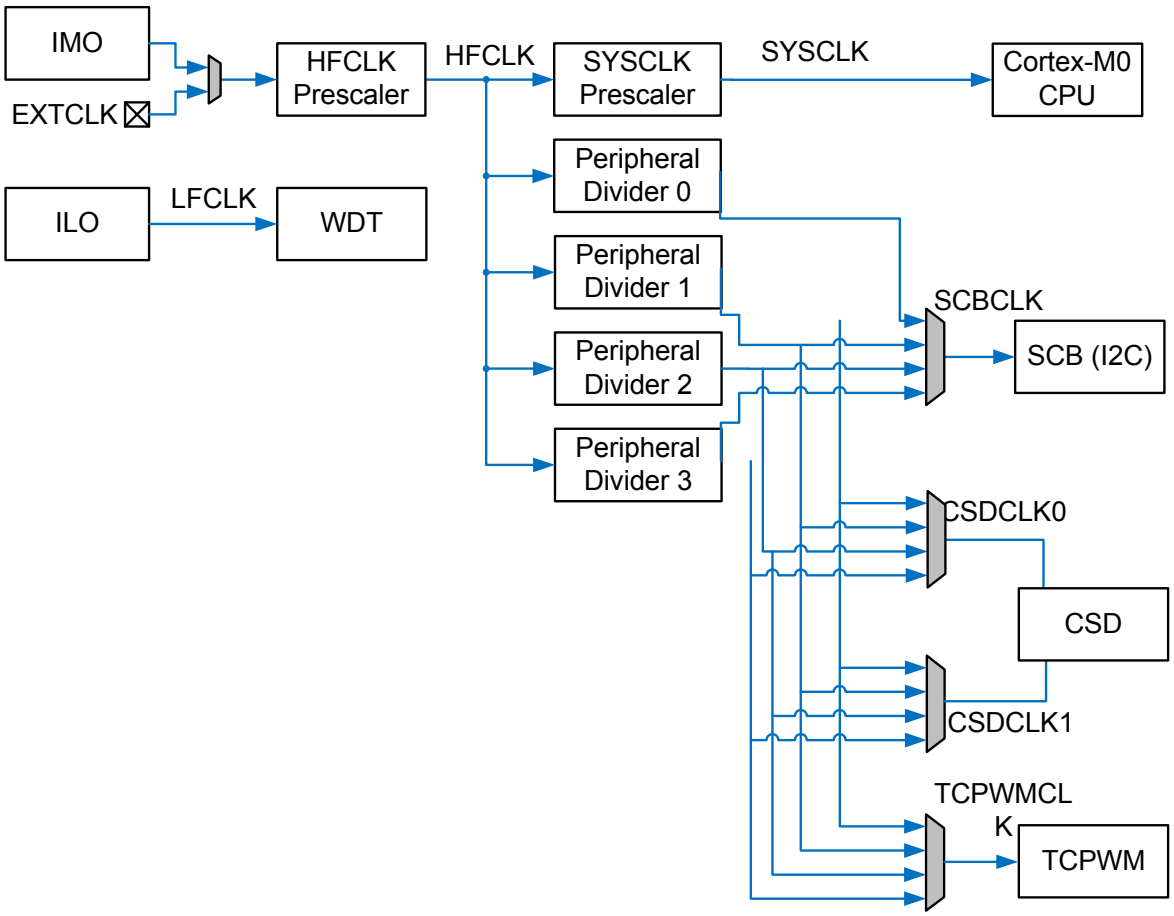


图 8-1 的左侧显示的是器件中的三个时钟源。HFCLK 复用器将从外部时钟源或 IMO 选择 HFCLK。ILO 来自 LFCLK。SYSCLK 预分频器生成 SYSCLK，则外设分频器生成单独的外设时钟。

8.2 时钟源

8.2.1 内部主振荡器

内部主振荡器无需外部组件仍可运行，并输出各种频率的稳定时钟，频率范围为 24 至 48 MHz，它的步长为 4 MHz。IMO 和赛普拉斯提供的校准之间的容差为 ±2%（24 MHz 和 32 MHz）。不能保证在其他频率设置下的容差。因此，推荐将使用的 IMO 频率设置为 24 MHz 或 32 MHz。有关详细信息，请参见器件数据手册中介绍的内容。通设置寄存器 CLK_IMO_TRIM2 中的频率范围 以及寄存器 CLK_IMO_TRIM1 中的 IMO 调整，可以选择频率。每个器件都有在制造过程中测量的 IMO 调整值以满足数据手册的规范；此调整被存储在 SFLASH 中的制造配置数据中。可能在启动时加载这些值，以实现所需的配置。固件可检索这些调整值，并重新配置器件以修改运行时的频率。

8.2.1.1 启动行为

复位之后，IMO 被配置为 24 MHz。在启动的 “ 引导 ” 部分中，调整值从闪存读取并配置 IMO 以获取数据手册指定的准确度。最初时，将 HFCLK 预分频器设置为 4 的分频值，用以降低启动时消耗的电流。

8.2.2 内部低速振荡器

内部低速振荡器无需外部组件仍可运行，并输出 32 kHz 的稳定时钟。ILO 是相对低功耗和低准确度的。它在所有功率模式下均可用。在 PSoC 4 中， ILO 始终作为系统低频率时钟 LFCLK 使用。推荐始终使能 ILO，因为它是 WDT 的源，这样能够保证系统可靠地运行。通过清除CLK_ILO_CONFIG 寄存器中的 ENABLE 位，可以禁用 ILO。禁用 ILO 前，必须先禁用 WDT 复位。否则，所有用以禁用 ILO 的寄存器写入操作均被忽略。使能 WDT 复位可自动使能 ILO。

注意： 如果发生下面条件，不推荐禁用 ILO 复位：

- 需要 WDT 保护，来避免固件崩溃
- 需要 WDT 保护来防止电源突然断电事件，因为该事件可能会返过来损害 CPU 的功能。

有关详细信息，请参见第 53 页上的时钟系统章节。

8.2.3 外部时钟

外部时钟是 MHz 范围的时钟，它可由设计 PSoC 4 引脚上的信号生成。使用该时钟取代 IMO 作为系统高频率时钟 HFCLK 的源。外部时钟频率的可用范围为 0–16 MHz。 PSoC 4 始终使用 IMO 启动，并且外部时钟必须在用户模式下使能。因此，不可通过使用外部时钟给复位信号提供时钟脉冲的复位信号来启动器件。

8.3 时钟分布

PSoC 4 时钟在整个器件中开发和分布，如图 8-1 所示。分布配置有以下的选项：

- HFCLK 输入选择
- HFCLK 预分频器的配置
- SYSCLK 预分频器的配置
- 外设分频器的配置

8.3.1 HFCLK 输入选择

PSoC 4 中的 HFCLK 有两个输入选项：IMO 和 EXTCLK。通过使用 CLK_SELECT 寄存器的 DIRECT_SEL 位，可以选中 HFCLK 输入，如表 8-1 中所述。

表 8-1. HFCLK 输入选择位 DIRECT_SEL

名称	说明
DIRECT_SEL[2:0]	HFCLK 输入时钟选择 0：IMO。 IMO 作为 HFCLK 源使用 1：EXTCLK。 EXTCLK 作为 HFCLK 源使用 2–7：保留。请勿使用

当手动将引脚配置为 EXTCLK 的输入时，引脚的驱动模式必须设置为高阻抗数字，以使能数字输入缓冲区。有关更多信息，请参考第 45 页上的 I/O 系统章节。

8.3.2 HFCLK 预分频器的配置

SYSCLK 预分频器允许器件在将 HFCLK 选择复用器输入作为 HFCLK 使用前对它进行分频。预分频器可以对 HFCLK 进行 2 的幂 （从 1 到 8）分频。可以使用 CLK_SELECT 寄存器中的 HFCLK_DIV 位来设置预分频器的分频值，如表 8-2 所示。在引导时将 HFCLK 预分频器设置为 4 的分频值，用以降低电流消耗。

注意： HFCLK 的频率不能超过 16 MHz。

表 8-2. HFCLK 预分频器的分频值位 HFCLK_DIV

名称	说明
SYSCLK_DIV[1:0]	SYSCLK 预分频器的分频值 0：1。 SYSCLK = HFCLK 1：2。 SYSCLK = HFCLK / 2 2：4。 SYSCLK = HFCLK / 4 3：8。 SYSCLK = HFCLK / 8

8.3.3 SYSCLK 预分频器的配置

SYSCLK 预分频器允许器件在将 HFCLK 作为 SYSCLK 使用之前对它进行分频，这样允许外设时钟和系统时钟之间的非整数关系。 SYSCLK 的时钟频率需要大于或者等于各个外设的时钟频率。预分频器可对 HFCLK 进行 2 次方 （1 到 8）的分频。使用 CLK_SELECT 寄存器中的 SYSCLK_DIV 位设置预分频器的分频值，如表 8-3 所示。预分频器最初被配置为 1 分频。

注意： SYSCLK 频率不能超过 16 MHz。

表 8-3. SYSCLK 预分频器的分频值位 SYSCLK_DIV

名称	说明
SYSCLK_DIV[1:0]	SYSCLK 预分频器的分频值 0：1。 SYSCLK = HFCLK 1：2。 SYSCLK = HFCLK / 2 2：4。 SYSCLK = HFCLK / 4 3：8。 SYSCLK = HFCLK / 8

8.3.4 外设时钟分频器的配置

通过使用 16 位外设时钟分频器对 HFCLK 进行分频后，可得到四个外设时钟。每个分频器都能够对输入时钟进行 1~65,536 分频。四个分频器中的每一个均由 PERI_DIV_16_CTL 寄存器控制，其映射情况如表 8-4 所示。

表 8-4. 外设时钟分频器的控制寄存器 PERI_DIV_16_CTLx

位	名称	说明
0	EN	使能或禁用分频器 0：分频器被禁用 1：分频器被使能
23:8	INT16_DIV	分频器的分频值。输出 = 输入 / (INT16_DIV + 1) 它的取值范围为 0 到 65,536。

PERI_DIV_CMD 寄存器可用于使能、禁用并选择所有外设时钟分频器的类型。请参阅技术参考手册 （TRM）中相关 PERI_DIV_CMD 寄存器的内容，了解更详细的信息。

外设的输入时钟由 PERI_PCLK_CTLx 寄存器选中。表 8-5 显示的是外设时钟以及它们相应的寄存器。请参阅寄存器的技术参考手册，了解更详细信息。

表 8-5. 选择外设时钟

时钟	寄存器
SCB （I2C）	PERI_PCLK_CTL0
CSD0	PERI_PCLK_CTL1
CSD1	PERI_PCLK_CTL2
TCPWM	PERI_PCLK_CTL3

8.4 低功耗操作模式

PSoC 4 时钟的操作在不同功耗模式下会不一样。MHz 频率时钟，包括 IMO、EXTCLK、HFCLK、SYSCLK 并外设时钟，仅在活动模式和睡眠模式下运行。则 ILO 和 LFCLK 可以在所有功耗模式下运行。

8.5 寄存器列表

表 8-6. 时钟系统寄存器列表

寄存器名称	说明
CLK_IMO_TRIM1	IMO 微调寄存器 — 该寄存器包含 IMO 的调整，允许其频率细调。
CLK_IMO_TRIM2	IMO 频率选择寄存器 — 该寄存器控制 IMO 的频率范围，允许其频率粗调。
CLK_ILO_CONFIG	ILO 配置寄存器 — 该寄存器控制 ILO 的配置。
CLK_SELECT	时钟选择 — 该寄存器控制时钟树配置，用以选择不同的系统时钟源。
DIVIDER_x	外设时钟分频器控制寄存器 — 这些寄存器可配置外设时钟分频器，选择时钟源、设置整数分频值及使能或禁用分频器。
DIVIDER_FRAC_x	外设时钟小数分频器控制寄存器 — 这些寄存器可配置外设时钟分频器，选择时钟源、设置小数分频值以及使能或禁用分频器。
SELECT_x	外设时钟选择寄存器 — 这些寄存器可配置外设时钟分频器的输出，选择某个特定行中特定分频器的源。
PERI_DIV_16_CTLx	外设时钟分频器控制寄存器 — 这些寄存器可配置每个外设时钟分频器，使能或禁用分频器以及设置整数分频值。
PERI_PCLK_CTLx	可编程时钟控制寄存器 — 这些寄存器用以选择外设的时钟输入。

9. 电源与电源监测

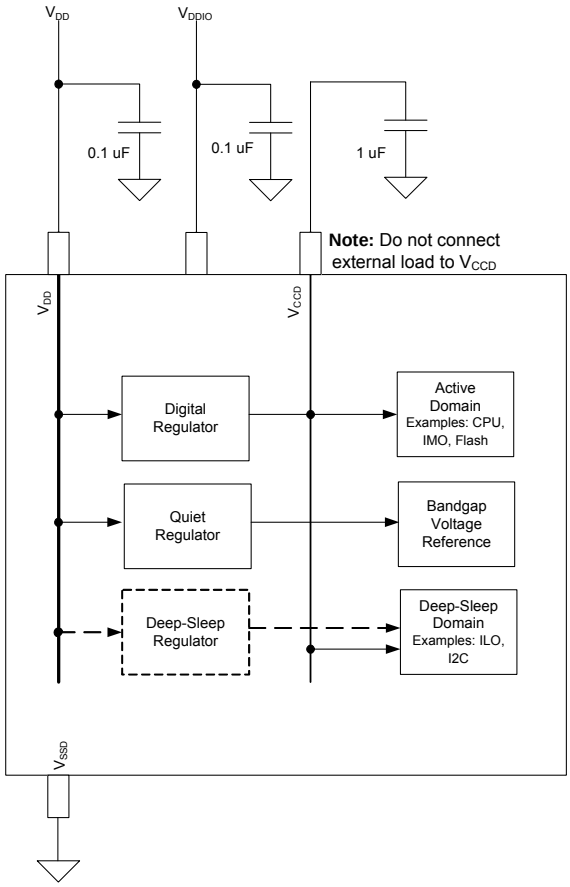
PSoC® 4 可在电压大小为 1.71 V 到 5.5 V 的单电源外部供电情况下运行。支持的工作电压范围如下：

- 内部电压调压器的输入电压范围为 1.80 V 至 5.50 V
- 直接供应的电源电压范围为 1.71 V 至 1.89 V

PSoC 4 器件具有多种内部电压调压器以支持不同的功耗模式。这些电压调压器包括：活动状态数字调压器、低噪声调压器，和深度睡眠状态调压器 。

9.1 框图

图 9-1. 电源系统框图



通过一个活动状态数字调压器，可将外部 V_{DD} 供电电压调整为数字内核所需的额定电压 1.8 V。该调压器的输出引脚满足电容的特定要求，如图 9-1 所示。该活动状态数字调压器专门为内部电路供电，**不能连接外部负载**。

初级稳压电源（即 V_{CCD}）可以是内部调节配置得到的，也可以由引脚直接供给。在内部调节模式中， V_{DD} 的变化范围为 1.8 V 到 5.5 V，且片上电压调压器会生成其他低供电电压。

在直接供电配置中，必须将 V_{CCD} 和 V_{DD} 互相短接，并且要将其连接到电压范围为 1.71 V 到 1.89 V 的电源。活动状态数字调压器仍被供电，并默认被使能。必须通过固件禁用它，以降低功耗；请参考 9.3.1.1 活动状态数字调压器。

在特定的封装类型中，V_{DDIO} 引脚可为 I2C 引脚提供单独的电压域。这样，芯片可以与 I2C 系统进行通信，并且能够在不同的电压范围下运行（其中 $V_{DDIO} \le V_{DD}$）。例如，V_{DD} 可以等于 3.3 V，V_{DDIO} 可以等于 1.8 V。请参考 “ 器件数据手册 ” 的内容，了解详细信息。

使用一个额外的电压调压器用于为工作的深度睡眠模式补充电源。另外，低噪声调压器可供电给敏感的模拟电路，包括带隙参考和电容触摸传感子系统。

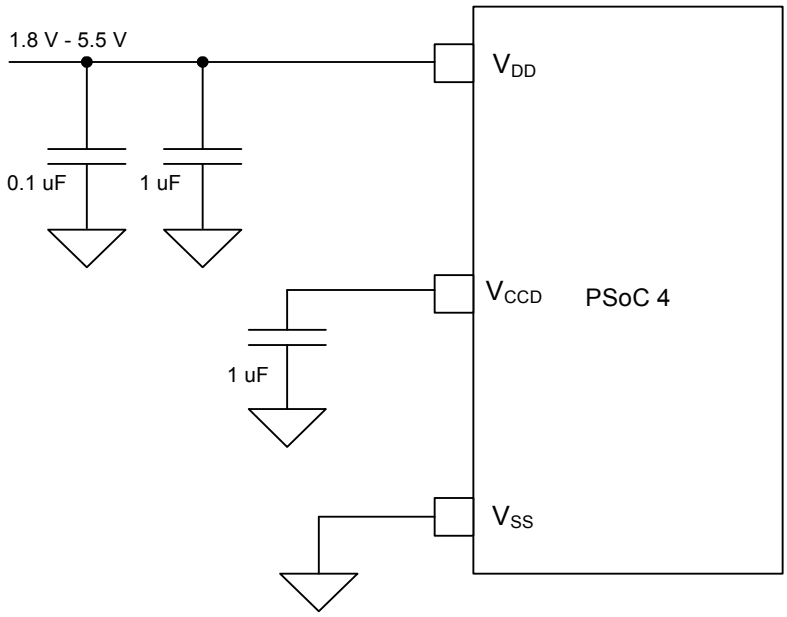
9.2 供电电压方案

下图描述了给 PSoC 4 器件供电的不同方式。

9.2.1 电压范围为 1.8 V 至 5.5 V 的单线未调节电源

如果大小为 1.8 V 到 5.5 V 的电源作为 PSoC 4 的调节供电电压输入使用，应连接它，如图 9-2 所示。

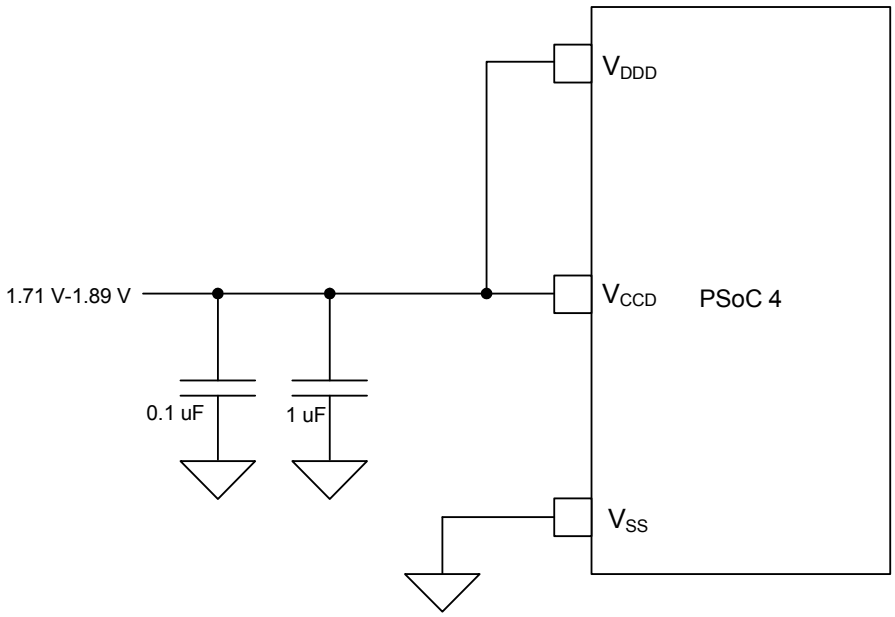
图 9-2. 单独调节 V_{DD} 电源



9.2.2 电压范围为 1.71 V 到 1.89 V 的调节电源直接供电

在直接供电配置中，V_{CCD} 和 V_{DD} 被短接，并连接一个电压范围为 1.71 V 至 1.89 V 的电源。应将该未调节的电源连接至 PSoC 4，如图 9-3 所示。

图 9-3. 单独的未调节 V_{DD} 电源



9.2.3 V_{DDIO} 电源。

V_{DDIO} 引脚 （在特定的封装类型中可用）为 I2C 引脚提供单独的电压域。请参见 “ 器件数据手册 ” 中的内容，了解当存在 V_{DDIO} 时的供电电压的连接情况。

9.3 工作原理

图 9-1 中的电压调压器将为器件的不同区域供电。所有电压调压器的输入均来自 V_{DD} 引脚上的电源。数字 I/O 是由 V_{DD} 供电。 V_{DDIO} 引脚 （在特定的封装类型中可用）为 I2C 引脚提供单独的电压域。有关详细信息，请参见 “ 器件数据手册 ” 中介绍的内容。

9.3.1 电压调压器汇总

在活动或睡眠功耗模式中，活动状态数字调压器和低噪声调压器均被使能。在深度睡眠模式中，它们被禁用 （请参见表 11-1 和图 9-1）。

9.3.1.1 活动状态数字调压器

对于大小属于 1.8 V 到 5.5 V 范围的外部电源，在活动模式和睡眠模式下，活动状态数字电压调压器提供主数字逻辑。该调压器的输出被连接至 V_{CCD} 引脚，并且需要连接一个外部去耦电容 （1µF X5R）。

对于低于 1.8 V 的电源，必须直接为 V_{CCD} 供电。在这种情况下， V_{CCD} 和 V_{DD} 必须互相短接，如图 9-3 所示。

通过置位 PWR_CONTROL 寄存器中的 EXT_VCCD 位，可以禁用活动状态数字调压器。这样可以降低直接供电模式中的功耗。活动状态数字调压器仅在活动和睡眠功耗模式中可用。

9.3.1.2 低噪声电压调压器

在活动和睡眠模式中，该电源调压器用于为模拟电路 （如带隙参考和电容式感测子系统）供电。这些模拟电路需要低噪声电源、无数字切换噪声以及无电源噪声。该调压器具有高电源抑制比。低噪声电压调压器仅在活动和睡眠功耗模式中可用。

9.3.1.3 深度睡眠状态电压调压器

该电压调压器将供电给在深度睡眠模式下保持上电的电路，如 ILO 和 SCB。深度睡眠状态电压调压器在所有功耗模式中均可用。在活动和睡眠功耗模式中，该调压器的主输出被连接至数字电压调压器的输出 （V_{CCD}）。该调压器也有独立的副本输出，它为 ILO 提供稳定的电压。在活动和睡眠模式下，该输出未连接至 V_{CCD}。

9.3.2 电压监控

电压监控系统包括上电复位 （POR）和欠压检测 （BOD）。

9.3.2.1 上电复位 （POR）

在初始电源上升阶段，上电复位电路提供了一个复位脉冲。POR 电路将监控 V_{CCD} 电压。通常，在触发点上，POR 电路的监控不是很准确。

POR 在芯片的初次上电期间中被启用，然后被禁用。

9.3.2.2 欠压检测（BOD）

通过对器件进行复位， BOD 电路可使处于工作 / 保持状态的逻辑避免可能发生的无安全供电状态。BOD 电路将监控 V_{CCD} 电压。如果电压偏移低于最低安全电压 V_{CCD}（欲了解详细信息，请参考 PSoC 4 数据手册），则 BOD 电路将生成一个复位。系统不会退出复位状态，直到供电电压再次进入有效范围为止。有一些电压状态未被 BOD 电路检测到，但它们会阻止器件正常运行。出于该原因，应在所有设计中使用看门狗定时器，以确保操作可靠。

9.4 寄存器列表

表 9-1. 电源与电源监控寄存器列表

寄存器名称	说明
PWR_CONTROL	功耗模式控制寄存器 — 通过该寄存器可以配置器件的功耗模式，并能够激活电压调压器。

10. 芯片工作模式

PSoC® 4 可以在四种不同模式下运行固件。在不同模式下，闪存和 ROM 内的运行位置以及硬件的特权级别也不一样。终端应用只使用其中的三种模式。在固件开发过程中，调试模式专门用来进行调试。

PSoC 4 的工作模式包括：

- 引导模式
- 用户模式
- 特权模式
- 调试模式

10.1 引导模式

引导模式是一种运行 ROM 器件中的硬编码指令对器件进行配置的工作模式。复位结束后如果没有接收到调试指令，芯片将进入此模式。引导模式是特权模式。在这个模式下，中断被禁止，以免引导固件进行器件设置时被打断。在上电期间，从 NV 锁存加载硬件调整设置，以保证上电期间的正常运行。引导结束时，器件将进入用户模式，并开始执行闪存的代码。闪存内代码可能包括从 PSoC Creator IDE 自动生成的指令，该指令将进一步配置器件。

有关器件启动的详细信息，请参考第 73 页上的复位系统章节。

10.2 用户

用户模式是一种可执行正常用户固件的工作模式。用户模式不能运行 ROM 中的代码。用户模式下的固件运行包括以下两个操作：PSoC Creator IDE 自动生成的固件，以及用户编写的固件。自动生成的固件可以控制固件的启动以及部分正常操作。完成自己的任务后，引导过程会控制此模式。

10.3 特权级别

特权模式是一种操作模式，它允许执行存储于器件 ROM 中的特别子程序。这些子程序不能被用户更改，并且用于执行不被中断或观察的专有代码。在特权模式中，不允许进行调试。退出该模式后，器件将返回用户模式。

10.4 调试

调试模式是一种允许检测 PSoC 4 操作参数的工作模式。在开发过程中，此模式用于调试固件。在指定时间窗口（器件复位过程中所发生的）内，将 SWD 调试器连接至器件时，将进入该模式。调试模式允许使用 IDE（如 PSoC Creator 或 ARM MDK）调试固件。调试模式只适用于开放模式的器件。更多有关调试接口的信息，请查阅第 129 页上的编程与调试接口章节。

更多有关保护模式的信息，请查阅第 75 页上的器件安全章节。

11. 功耗模式

PSoC® 4 提供了多种功耗模式，从而最小化某个具体应用的平均功耗。按功率消耗降低的顺序，功耗模式包括：

- 活动模式
- 睡眠模式
- 深度睡眠模式

活动、睡眠及深度睡眠模式是标准的 ARM 定义的功耗模式。 ARM CPU 和指令集架构 （ISA）支持这些模式。

可通过下列方法来控制不同功耗模式中的功率消耗：

- 使能 / 禁用外设的时钟
- 开启 / 关闭内部电压调压器
- 开启 / 关闭时钟发生器
- 开启 / 关闭 PSoC 4 部分组件

图 11-1 描述了各种功耗模式以及他们之间的可能转换

图 11-1. 功耗模式转换的状态框图

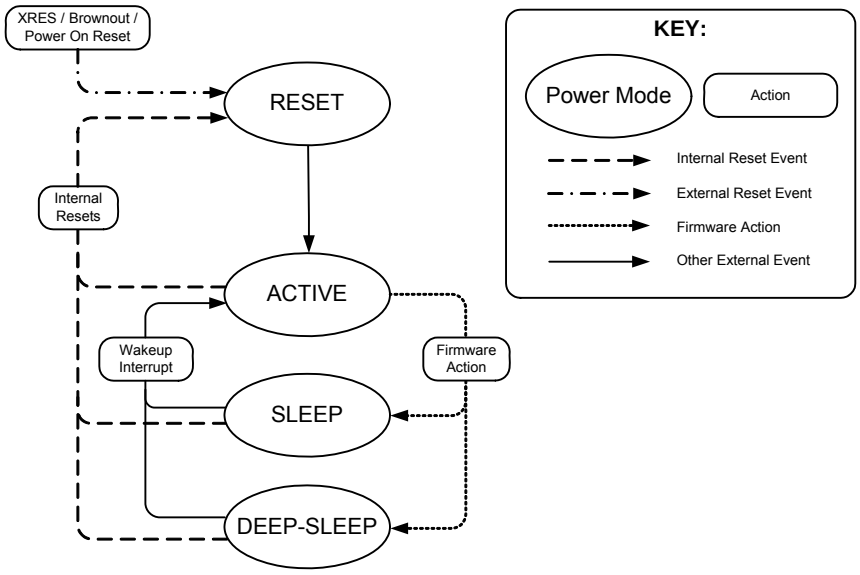


表 11-1 描述了 PSoC 4 所提供的各种功耗模式

表 11-1. PSoC 4 功耗模式

功耗模式	说明	进入条件	唤醒源	活动时钟	唤醒事件	可用电压调压器
活动	主要工作模式，所有外设均可用 （可编程）。	从其他功耗模式唤醒、内部和外部复位、欠压、上电复位	不适用	任意 （可编程）	中断	所有电压调压器均可用。如果使用外部稳压调节，则可以禁用活动状态数字电压调压器。
睡眠模式	CPU 进入睡眠模式，SRAM 被保留；所有外设均可用 （可编程）	通过寄存器手动写入	任意中断	任意 （可编程）	中断	所有电压调压器均可用。如果使用外部稳压调节，则可以禁用活动状态数字电压调压器。
深度睡眠	所有内部供电电源来自深度睡眠电压调压器。 IMO 和高速外设处于关闭状态。只有低频 （32 kHz） 时钟可用。通过低速、异步或低功耗模拟外设的中断可以实现唤醒操作。	通过寄存器手动写入	GPIO 中断、I2C、看门狗定时器	ILO （32 kHz）	中断	深度睡眠状态电压调压器

除了表 11-1 中讲述的唤醒源，可通过外部复位 （XRES） 和欠压复位使器件从任何功耗模式转换为活动模式。

11.1 活动模式

活动模式是 PSoC 器件的主要工作模式。在该模式下，允许使用器件中所有可能的子系统 / 外设。在该模式下， CPU 正在执行且所有外设被供电。通过配置固件可以禁用未使用的特定外设，从而能够降低功耗。

11.2 睡眠模式

该功耗模式以 CPU 为中心。在该模式下，Cortex-M0 CPU 进入睡眠模式， CPU 时钟被禁用。为降低功耗，通常 PSoC 4 要保持为睡眠模式，或 CPU 空闲后立即进入该模式。对于外设来说，该模式相当于活动模式。

所有已使能的中断均会使其从睡眠模式唤醒。

11.3 深度睡眠模式

在深度睡眠模式下， CPU、 SRAM 以及高速逻辑均被保留。 MHz 范围时钟 （包括 HFCLK 和 SYSCLK） 被禁用。可选用内部低频 （32 kHz） 振荡器，使低频外设继续正常操作。无需时钟或可接收外部时钟 （例如， I2C 从设备） 的数字外设仍持续运行。来自低速、异步或低功耗模拟外设的中断可以使其从深度睡眠模式唤醒。

表 11-3 中列出了可用的唤醒源。

11.4 功耗模式总汇

表 11-2 说明了每个低功耗模式下可用的外设；表 11-3 描述了每个功耗模式下可用的唤醒源。

表 11-2. 可用外设

外设	活动	睡眠	深度睡眠
CPU	开启	保留[a]	保留
SRAM	开启	保留	保留
高速外设	开启	开启	保留
低速外设	开启	开启	开启 （可选）
内部主振荡器 （IMO）	开启	开启	关闭
内部低速振荡器 （ILO， 32 kHz）	开启	开启	开启 （可选）
异步外设	开启	开启	开启
上电复位，欠压检测	开启	开启	开启
常规模拟外设	开启	开启	关闭
GPIO 输出状态	开启	开启	开启

a. 外设的配置和状态被保留。当器件进入活动模式时，外设继续进行它的操作。

表 11-3. 唤醒源

功耗模式	唤醒源	唤醒事件
睡眠	任何中断源	中断
	任何复位源	复位
深度睡眠	GPIO 中断	中断
	I2C 地址匹配	中断
	看门狗定时器	中断 / 复位
	XRES （外部复位引脚）[a]，欠压	复位

a. XRES 将触发一个系统复位。所有状态 （包括冰结 GPIO）均被丢失。在此情况下，器件重启后，无法读取唤醒原因。

11.5 低功耗模式进入和退出

Cortex-M0 （CM0）的 “Wait For Interrupt” （等待中断，WFI）指令能够触发切换到睡眠 和深度睡眠模式的事件。Cortex-M0 可以维持事件进入低功耗模式，直到退出最低优先级的 ISR 为止 （如果置位了 CM0 系统控制寄存器中的 SLEEPONEXIT 位）。

通过 CM0 系统控制寄存器 （SCR）中的 SLEEPDEEP 标志可以控制切换到睡眠 和深度睡眠模式。

- 当 WFI 指令被执行，且 SLEEPDEEP = 0 时，将进入睡眠模式。
- 当 WFI 指令被执行，且 SLEEPDEEP = 1 时，将进入深度睡眠模式。

PWR_CONTROL 寄存器中的 LPM READY 位显示的是深度睡眠模式中电压调压器的状态。如果在电压调压器准备好前固件尝试进入深度睡眠模式，然后 PSoC 4 先进入睡眠模式，并且电压调压器准备就绪后，器件将进入深度睡眠模式。在硬件中，该操作会自动完成。

在睡眠和深度睡眠模式下，允许选择外设（请参见表 11-3），和固件可以使能或禁用与其相应的中断。已使能的中断可以导致从低功耗模式中唤醒后再进入活动模式。另外，所有 RESET 都会使系统返回活动模式。

进入深度睡眠模式前，将 IMO 频率修改为 12 MHz。从深度睡眠模式中唤醒后，恢复以前的IMO频率。有关如何修改IMO频率的详细信息，请参见第 53 页上的时钟系统章节的内容。

11.6 寄存器列表

表 11-4. 功耗模式寄存器列表

寄存器名称	说明
SCR	系统控制寄存器 — 设置或返回系统控制数据。
PWR_CONTROL	功耗模式控制 — 控制器件的功耗模式选项，且允许观察当前状态。

12. 看门狗定时器

当固件执行异常或欠压导致 CPU 功能异常时，可使用看门狗定时器（WDT）进行自动复位。WDT 的工作频率为 32 kHz，工作时钟是 ILO。程序必须定期刷新定时器，以避免发生复位。否则，定时器将停止并产生复位。WDT 还可作为低功耗模式中的中断源或唤醒源使用。

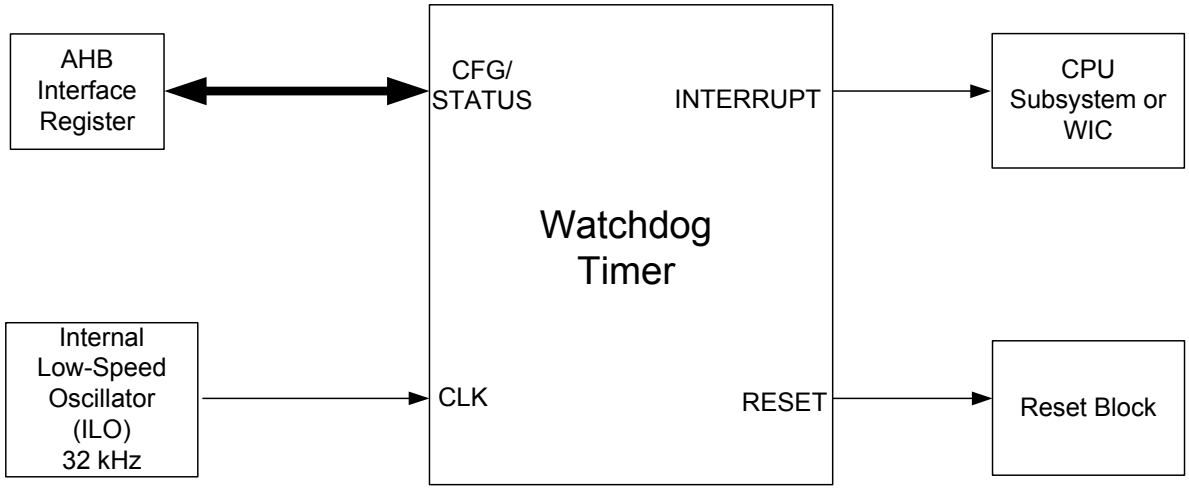
12.1 特性

WDT 的特性包括：

- 可配置的定时器周期
- 通过生成中断，可从睡眠或深度睡眠模式唤醒器件
- 经过指定的时间间隔后，可在活动模式中生成中断
- 用于防止意外破坏寄存器的保护设置

12.2 框图

图 12-1. 看门狗定时器框图



12.3 工作原理

经过最多可达 2048 毫秒的可编程时间间隔后，WDT 将激活器件的中断或硬件复位，除非程序周期性对 WDT 刷新。WDT 是一个自由运行的 16 位向上计数器。

WDT_COUNTER 寄存器提供了 WDT 的计数值。WDT_COUNTER 寄存器中的计数值等于存储在 WDT_MATCH 寄存器中的匹配值时，WDT 会生成中断。但它不会将计数值重置为 ‘0’。WDT 却会保持计数值，直到该值溢出并返回 0 为止。当计数值再次达到匹配值时，会生成下一个中断。

每当发生 WDT 中断时，都会置位 SRSS_INTR 寄存器中的 WDT_MATCH 位。通过向 SRSS_INTR 中的 WDT_MATCH 位内写入 ‘1’，可以清除该中断，以刷新看门狗定时器。如果连续两个中断里固件没有刷新 WDT，则第三次匹配事件发生时将生成硬件复位。

有关详细信息，请参考 PSoC 4 寄存器 TRM 中介绍的 WDT_COUNTER、WDT_MATCH 和 SRSS_INTR 寄存器。

当使用 WDT 防止系统崩溃时，必须使用与 WDT 中断没有直接关联的代码来清除 WDT 中断位来刷新 WDT。否则，就算固件的主要函数崩溃或处于无限循环状态，WDT 中断向量仍完整并定期刷新 WDT。

使用 WDT 防止系统崩溃的最安全方法是：

- 通过在固件代码的主体中定期清除中断位刷新看门狗。
- 确保在每个 WDT 周期内至少清除一次中断。
- 将 WDT 中断服务子程序（ISR）作为定时器使用，以触发特定的行动，并修改下一个 WDT_MATCH 值。在 ISR 中请勿刷新 WDT。

按照以下各步骤使用 WDT 作为一个周期性的中断发生器：

1. 将所需的匹配值写入到 WDT_MATCH 寄存器内
2. 使能 SRSS_INTR 中的 WDT 中断
3. 在 ISR 中，清除 WDT 中断，并使用所需的匹配值替换现有的匹配值。这样，当该计数器达到新的匹配值时，将生成下一个 WDT 中断。
4. WDT_MATCH 寄存器中的 IGNORE_BITS 可用于减少整个 WDT 计数器周期。使用忽略的位可以指定需要清除的 MSB 数量。例如，如果 IGNORE_BITS 的值为 3，则 WDT 计数器变为 13 位的计数器。

12.3.1 使能和禁用 WDT

WDT 是一个不可禁止的自由运行的计数器。但通过向 WDT_DISABLE_KEY 寄存器中写入 ‘0xACED8865’，可禁用 WDT 复位。将任意其他值写入到寄存器中均可以使能 WDT 复位。如果禁用 WDT 复位，则固件不需定期刷新 WDT，这样可以避免产生复位。WDT 还可以作为中断源或唤醒源使用。阻止 WDT 生成中断和唤醒事件的唯一方法是通过清除 CLK_ILO_CONFIG 寄存器中的 ENABLE 位来禁用 ILO。禁用 ILO 前，必须禁用 WDT 复位。否则，会忽略任何用以禁用 ILO 的寄存器写入操作。使能 WDT 复位会自动使能 ILO。

注意： 如果发生下述条件，不推荐禁用 WDT 复位：

- 需要保护，防止系统崩溃
- 电源突然断电，可能会损坏 CPU 的功能

12.3.2 WDT 中断和低功耗模式

在活动模式下，WDT 的中断请求被发送到 CPU；在睡眠和深度睡眠模式下，WDT 的中断被发送到唤醒中断控制器（WIC）其工作原理如下：

- **活动模式：**在活动模式下，WDT 发送中断给 CPU。CPU 确认中断请求，并执行 ISR。进入 ISR 后，必须在固件中清除该中断。
- **睡眠或深度睡眠模式：**在该模式中，CPU 子系统被断电。因此，来自 WDT 的中断请求被直接发送给 WIC 用于唤醒 CPU。CPU 确认中断请求，并执行 ISR。进入 ISR 后，必须在固件中清除该中断。

更多信息，请参考第 65 页上的功耗模式章节。

12.3.3 WDT 复位模式

RES_CAUSE 寄存器中的 RESET_WDT 位表示 WDT 生成了复位。该位保持置位状态，直到对其进行清除，或发生上电复位（POR）或欠压复位为止。所有其它复位操作都不会对该位产生任何影响。

更多信息，请参考第 73 页上的复位系统章节。

12.4 寄存器列表

表 12-1. WDT 寄存器

寄存器名称	说明
WDT_DISABLE_KEY	当写入 0XACED8865 时，禁用 WDT，写入其他任何值 , WDT 均正常工作。
WDT_COUNTER	提供 WDT 的计数值
WDT_MATCH	存储 WDT 的匹配值
SRSS_INTR	刷新 WDT 以避免复位

13. 复位系统

PSoC® 4 支持多种类型的复位，这些复位可保证器件上电时无错误运行，并且允许器件根据用户提供的外部硬件或内部软件复位信号进行复位。 PSoC 4 还包括了使能复位检测的硬件。

复位系统包括以下复位源：

- 上电复位 （POR）— 供电电压上升时，器件将处于复位状态
- 欠压复位 （BOD）— 在操作过程中，如果供电电源电压低于规定的范围，将产生复位
- 看门狗复位 （WRES）— 如果固件没有正常复位看门狗定时器，将产生复位
- 软件复位 （SRES）— 如果需要，可以使用固件来产生复位
- 外部复位 （XRES）— 使用 PSoC 4 的外部信号复位器件
- 特权保护复位 （PROT_FAULT）— 如果用户尝试进行特权操作时，将产生复位

13.1 复位源

下述内容将描述 PSoC 4 中的各种复位源 。

13.1.1 上电复位

上电复位用于在上电时系统复位。 POR 会将器件锁定于复位状态，直到电压 （即 V_{DDD}）满足数据手册中的规范为止。 POR 在上电时自动被激活。

POR 事件不会设置复位源的状态位；但是，通过观察其它复位源的状态，可以推断出部分原因。如果未检测到其它复位事件，那么，复位是由 POR、 BOD 或 XRES 引起的。

13.1.2 欠压复位

欠压复位将监控数字电源电压 V_{CCD} ； 如果 V_{CCD} 低于器件数据手册中指定的最小逻辑工作电压值，将产生复位。BOD 在所有功耗模式中均可用。

BOD 事件将不设置复位源状态位，但是在某些情况下，可以检测到。在某些 BOD 事件中，V_{CCD} 会低于最小的逻辑工作电压，但仍大于最小逻辑保持电压。因此，可以通过检查逻辑保持来区分这些 BOD 事件和 POR 事件。

13.1.3 看门狗复位

如果看门狗定时器未在用户指定的时限内复位计数器，看门狗 （WRES）将产生复位，这表明用户程序发生了错误。通过置位 WDT_CONTROL[0] 寄存器位来使能该功能。

发生看门狗复位时， RES_CAUSE[0] 状态位被置位。该位保持置位状态，直到它被清除或发生 POR 或 BOD 复位为止，例如在器件的上电周期。所有其它复位不会对该位产生任何影响。

更多信息，请参考 第 69 页上的看门狗定时器章节

13.1.4 软件复位

软件复位（SRES）是一个允许软件产生复位的机制。向位 2 写入 ‘1’ 时，Cortex-M0 应用中断与复位控制寄存器（AIRCR）将强制器件复位。 AIRCR 需要将数值 A05F 写到前两个半字节内。因此，复位操作需要写入 A05F0004。

发生软件复位时， RES_CAUSE[4] 状态位将被置位。该位保持置位状态，直到被清除或发生 POR 或 BOD 复位为止，例如在器件的供电周期。所有其它复位不会对该位产生任何影响。

13.1.5 外部复位

外部复位（XRES）是一种由用户提供的复位，产生后立即导致系统复位。XRES_N 引脚是***低电平有效*** — 该引脚上的高电平电压不起任何作用，而低电平电压会导致复位。在器件中，引脚电平被拉高。在所有器件中，XRES_N 均作为专用引脚。

XRES 引脚处于活动状态时，将保持器件的复位状态。释放引脚后，器件将进行正常的启动操作。器件数据手册中的电气规范一节列出了各 XRES 和其它电气特性的逻辑阈值。

XRES事件将不设置复位源的状态位；但是，通过观察其它复位源的缺失情况，可推断出部分原因。如果未检测到其它复位事件，那么，复位是由 POR、 BOD 或 XRES 引起的。

13.1.6 特权保护复位

特权保护复位（PROT_FAULT）将检测未经授权的特权操作；如果检测到此类行为的发生，它将产生复位。例如，执行特权代码时，遇到了调试断点。

发生保护故障时， RES_CAUSE[3] 位被置位。该位保持置位状态，直到它被清除或发生 POR 或 BOD 复位为止，例如器件的供电周期。所有其它复位不会对该位产生任何影响。

13.2 识别复位源

器件退出复位状态时，知道最近或更早的复位源是很有用的。通常，通过器件的 RES_CAUSE 寄存器，可以获得此信息。该寄存器具有与几种复位源相应的特定状态位。 RES_-CAUSE 寄存器支持看门狗复位、软件复位以及特权保护复位等复位源检测。它不会记录是否发生 POR、BOD 或 XRES。当发生复位时，相应的状态位将被置位，并且在复位后仍保持设置状态，直到被清除或丢失保留为止，如 POR 复位或低于逻辑保持电压的欠压复位。

如果 RES_CAUSE 寄存器不能检测到复位原因，那么复位原因可能是下面的无记录和无保持复位类型的一种：BOD、POR、 XRES 或 XRES。通过器件的资源无法区分这些复位源。

13.3 寄存器列表

表 13-1. 复位系统寄存器列表

寄存器名称	说明
WDT_CONTROL	看门狗定时器控制寄存器 — 通过该寄存器，可配置器件看门狗定时器。
AIRCR	Cortex-M0 应用中断和复位控制寄存器 — 除初始化软件复位的功能外，寄存器还提供 Cortex-M0 的其他功能 。
RES_CAUSE	复位源寄存器 — 该寄存器捕捉最近发生的复位的原因。

14. 器件安全

PSoC® 4 给用户提供了多种防止未授权访问和复制的选项。禁止调试性能以及强大的闪存保护提供了较高级别的安全性。

默认情况下，调试电路处于使能状态，并且只能在固件中被禁用。如果被禁用，唯一的使能调试方法是擦除整个器件，清除闪存保护，然后用新固件对器件进行重新编程。此外，对于担心因器件恶意重新编程而造成欺诈性攻击的应用或通过启动和中断闪存编程序列来击败安全性的尝试，可以永久禁用所有器件接口。在大多数应用中，不建议永久禁用接口，因为这样一来，设计人员将无法对器件进行访问。

注意：由于使能最高安全级别时将禁用所有编程、调试和测试接口，因此已启用全器件安全性的 PSoC 4 器件将不能退回进行故障分析。

14.1 特性

PSoC 4 器件安全系统具有以下特性：

- 用户可选的保护级别
- 在最高安全级别的模式中，芯片被 “ 锁定 ”。这时，无法对芯片进行测试或调试，也无法进行擦除操作。中断擦除操作是黑客使芯片处于未定义状态并打开观察的一种方式。
- 使用不可屏蔽中断（NMI）将使 CPU 在特权模式下运行。当处于特权模式下，NMI 保持确认状态，以避免中断指令可能导致安全漏洞的意外返回。

14.2 工作原理

CPU 在通用用户模式或特权模式下运行，而器件则在 BOOT、OPEN、PROTECTED 以及 KILL 等四种保护模式下运行。每种模式提供特定的功能用于 CPU 软件和调试 （通过调试访问端口 DAP）：

- **BOOT 模式**：在此模式下，器件将退出复位过程。器件处于此模式直到其保护状态从监控闪存被复制到保护控制寄存器（CPUSS_PROTECTION）内。此操作完成前，调试访问端口仍被禁止。 BOOT 是一种过渡模式。此模式要求器件处于已配置的保护状态。在 BOOT 模式下， CPU 时钟在特权模式下运行。
- **OPEN 模式**：这是出厂的默认模式。 CPU 可以于用户模式或特权模式下运行。在用户模式下，可以对闪存进行编程并支持调试器性能。特权模式的访问被限制。
- **PROTECTED 模式**：用户可以将 OPEN 模式转换成 PROTECTED 模式。在此模式下，不能对用户代码或存储器进行调试访问。只能访问用户寄存器，不允许重新编程闪存。只在完成擦除闪存后才能此模式转换回 OPEN 模式。
- **KILL 模式**：用户可以将 OPEN 模式转换成 KILL 模式。此模式删除所有用户代码或存储器的调试访问，同时不能对闪存进行擦除。只能访问用户寄存器，不允许重新编程闪存。器件无法退出 KILL 模式。处于此模式的器件不能退回进行故障分析。

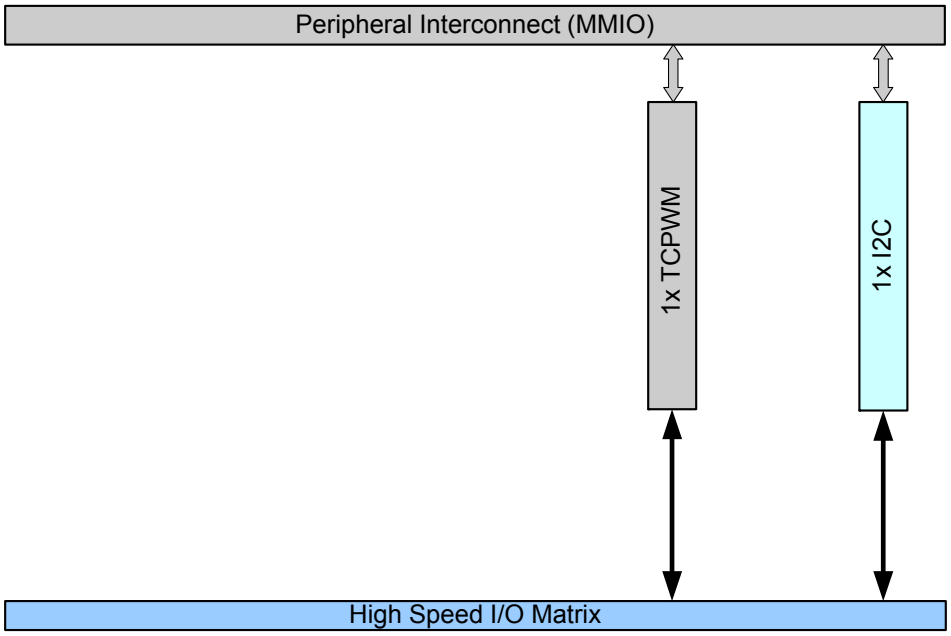
Section E: 数字系统

本章节包括以下小节：

- 第 79 页上的内部集成电路 （I2C）章节
- 第 95 页上的定时器、计数器和 PWM 章节

顶层架构

数字系统框图



15. 内部集成电路 （I2C）

PSoC 4 包含一个串行通信模块（SCB），只能将该模块配置为固定功能的 I2C 模块。本节介绍了 PSoC 4 中 I2C 的实现情况。更多有关 I2C 协议规范的信息，请查阅 NXP 网站上的 I2C 总线规范。

15.1 特性

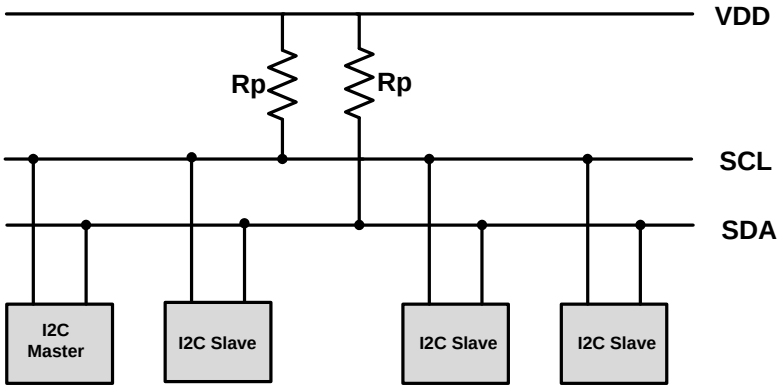
PSoC 4 支持下面的 I2C 特性：

- 主设备、从设备和主设备 / 从设备模式
- 慢速模式 （50 kbps）、标准模式 （100 kbps）和快速模式 （400 kbps）
- 7 位或 10 位从设备寻址 （10 位寻址需要固件支持）
- 时钟延长和冲突检测
- I2C 时钟信号 （SCL）的可编程过采样
- 在 I2C 数据信号 （SDA）的输入路径上使用数字中值滤波器来减少错误
- 使用模拟干扰滤波器传输无干扰的信号
- 中断或轮循 CPU 接口

15.2 概述

图 15-1 介绍了一个 I2C 通信网络的示例。

图 15-1. I2C 接口框图



标准 I2C 总线是一个双线接口，包括：

- 串行数据 （SDA）
- 串行时钟 （SCL）

I2C 器件通过集电极开路或开漏输出级和上拉电阻 （Rp）连接到这两线。各器件间有简单的主设备 / 从设备的关系。可将各主设备和从设备作为发送器或接收器运行。通过使用唯一的 7 位地址可以寻址总线上连接的每一个从设备。 PSoC 4 还为 I2C 的 10 位地址匹配提供了固件支持。

15.2.1 术语和定义

表 15-1 介绍了 I2C 通信网络中常用的术语。

表 15-1. I2C 总线术语的定义

术语	说明
发送器	是指将数据发送给总线的器件。
接收器	是指从总线接收数据的器件。
主设备	是指启动某个传输操作、生成时钟信号并终止传输的设备。
从设备	由主设备寻址的设备
多个主设备	多个主设备可同时尝试控制总线而不会破坏信息。
仲裁	是指如果多个主设备同时尝试控制总线，确保只有其中一个能实现，并且获得仲裁的信息不被损坏的过程。
同步	是指对两个或更多器件的时钟信号进行同步的过程。

总线停止 （时钟延长）

当从设备仍未准备好进行处理数据时，它可能在 SCL 线上驱动 ‘0’，以保持其低电平状态。由于启用了输入 / 输出信号接口，因此 SCL 线的值为 ‘0’，该值完全独立于其它主设备或从设备在 SCL 线上驱动的值。该情况被视为时钟延长，另外，它是唯一一个从设备驱动 SCL 线的情况。主设备监控 SCL 线。在 SCL 线上无法生成正时钟脉冲 （ ‘1’ ）时将检测该线路。主设备将在 SCL 线上延迟上升沿的生成，从而有效与延展时钟的从设备同步。

总线仲裁

I2C 协议是一个多主设备、多从设备的接口。通过监控 SDA 线在主设备上实现总线仲裁。当主设备发现 SDA 线路的值与当前SDA线路上驱动的值不一样时，将检测总线冲突。例如，主设备 1 在 SDA 线上驱动数值 ‘1’，且主设备 2 在 SDA 线上驱动数值 ‘0’，那么由于执行了输入 / 输出信号接口，所以实际线路值将为 ‘0’。主设备 1 检测到存在不一致情况，会丢失总线控制权。主设备 2 没有检测到任何不一致情况并保持总线控制权。

15.3 I2C 工作模式

I2C 是一个同步的单主设备、多主设备、多从设备串行接口。器件可以在主设备、从设备或主设备 / 从设备模式下运行。在主设备 / 从设备模式下，寻址器件时，它将从主设备模式切换到从设备模式。在数据传输期间，只有一个主设备有效。有效的主设备将驱动 SCL 线上的时钟。

表 15-2 显示的是 I2C 工作模式。

表 15-2. I2C 模式

模式	说明
从设备	仅限从设备操作 （默认）
主设备	仅限于主设备操作
多个主设备	支持总线上多个主设备
多个主设备 - 从设备	可以同时执行从设备和多个主设备的操作

使用 I2C 总线进行的数据传输要遵循特定的格式。表 15-3 列出了参与 I2C 数据传输的常见总线事件。写数据和读数据两节说明了数据传输中 I2C 总线上的位格式。

表 15-3. I2C 总线事件术语

总线事件	说明
START	SCL 为高电平时， SDA 线的电平状态由高转为低。
STOP	SCL 为高电平时， SDA 线的电平状态由低转为高。
ACK	发送器发送每个字节后，接收器将 SDA 线拉低，并在时钟脉冲高电平期间保持低电平状态。
NACK	发送器发送每个字节后，接收器并没有将 SDA 线拉低，并在时钟脉冲高电平期间保持它的高电平状态。
重复 START	传输过程结束时，主设备将生成 START 事件，而不是 STOP 事件

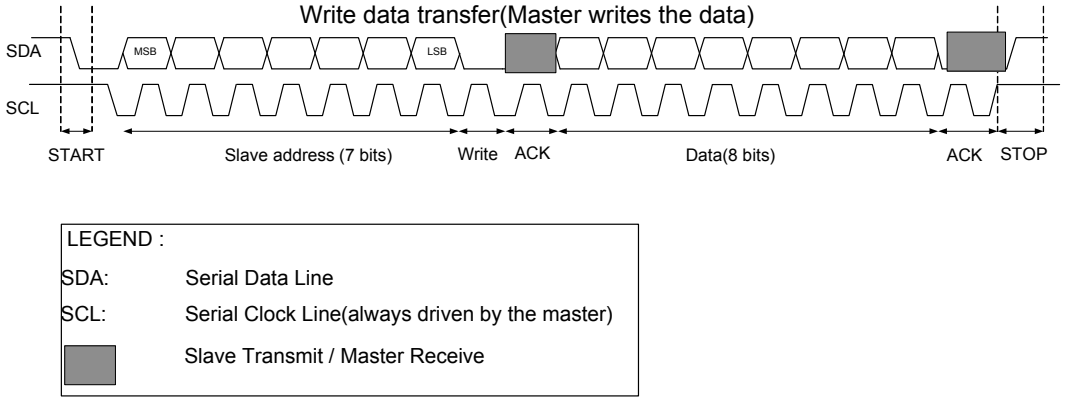
在多个主设备模式下操作时，应始终检查总线是否处于繁忙状态；其它主设备能够准备好与从设备进行通信。在此情况下，主设备必须等待完成当前的操作才能发出 START 信号（请查阅表 15-3、图 15-2 和图 15-3）。主设备寻找 STOP 信号，该信号表示主设备可以启动其数据传输。

在多个主设备 - 从设备模式下，如果主设备在数据传输期间失去了仲裁，则硬件将转到从设备模式，并且它所收到的字节会生成从设备地址中断。

对于所有这些模式，有两种传输，即读数据和写数据。在写数据中，主设备将数据发送给从设备；在读数据中，主设备接收来自从设备的数据。第 87 页上的主设备模式的传输示例 、第 89 页上的从设备模式的传输示例 和第 93 页上的多个主设备模式的传输示例 介绍的是写数据和读数据示例。

15.3.1 写数据

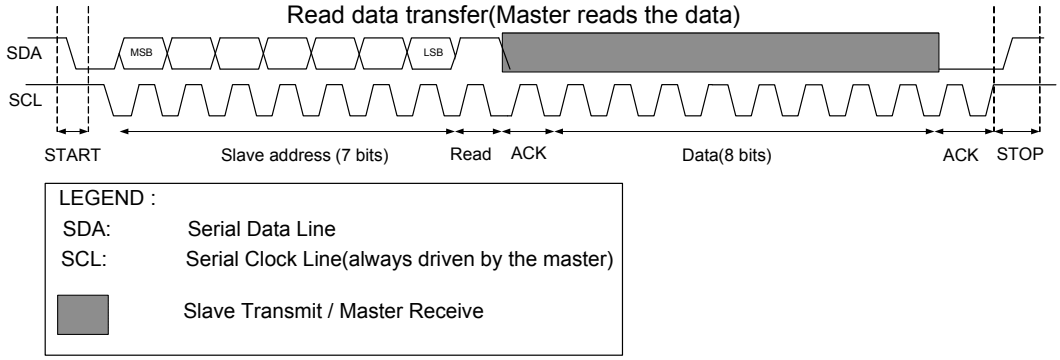
图 15-2. 主设备写数据



- 通常主设备在I2C总线上生成START状态来启动写数据。在 START 事件后，主设备将写入 7 位 I2C 从设备地址和写指示 （‘0’）。通过在第九位时间段内下拉数据线，被寻址的从设备将发送确认字节。
- 如果从设备地址没有与任何从设备匹配或寻址设备不想确认请求，它将发送无应答信号 （NACK）。这样，由于实现了上拉电阻， SDA 线的值将为 ‘1’ 。
- 如果从设备不发送任何确认，主设备通过 STOP 事件结束写传输。主设备也可生成重复 START 条件，重试传输。
- 如果主设备接收到确认信号，它会将写数据发送给总线。寻址的从设备将发送一个确认信号来确认写数据的接收。收到该确认信号时，主设备将传输其它数据字节。
- 传输完成时，主设备将生成 STOP 条件。

15.3.2 读数据

图 15-3. 主设备读数据



- 通常主设备会在 I2C 总线上生成 START 条件来启动读操作。在 START 事件后，主设备将写入 7 位 I2C 从设备地址和读指示 （‘0’）。通过在第九位时间段内将数据线下拉为低电平，寻址的从设备会传输确认信号。
- 如果从设备地址与连接从设备的地址不匹配或寻址器件不想确认请求，那么将发送无应答信号 （NACK）。这样，由于实现了上拉电阻， SDA 线的值将为 ‘1’ 。
- 如果从设备不发送任何确认信号，主设备会通过 STOP 事件结束读传输。主设备也可以生成重复 START 条件，重试传输。
- 如果从设备确认了地址，它将在确认信号后开始发送数据。主设备发送一个确认信号来确认接收到从设备发送的每个数据字节。收到该确认信号时，寻址的从设备将传输其它数据字节。
- 主设备可以发送 NACK 信号给从设备，指示从设备停止发送数据字节。这样便完成了读数据。
- 传输完成时，主设备将生成 STOP 事件。

15.4 Easy I2C （EZI2C）协议

Easy I2C (EZI2C) 协议是赛普拉斯在 I2C 协议上建立的独特的通信方案。它使用了标准 I2C 协议的软件封装器，并通过索引存储器传输与 I2C 从设备进行通信。这样， CPU 不需要干预每一个帧的操作。

EZI2C 协议定义了一个 8 位 EZ 地址（8 位宽和 32 入口深度）用于索引存储器阵列，该存储器阵列位于从设备上。EZ 地址的低 5 位用于寻址 32 个入口。通过比较 START 事件中的 EZ

地址和 STOP 事件中的地址，可以计算出发送给 EZI2C 存储器阵列或从 EZI2C 存储器阵列收到的字节的数量。

注意 ： 固定功能 I2C 模块具有一个硬件 FIFO 存储器，该存储器具有 16 位宽度和 16 入口深度，并具有字节写使能性能。EZ 和非 EZ 功能的访问方式有所不同。在非 EZ 模式下，FIFO 被分开为 TXFIFO 和 RXFIFO，各有 16 位宽和 8 入口深度。而在 EZ 模式下，FIFO 作为单存储器单元（8 位宽和 32 入口深）使用。

EZI2C 包括两种传输类型 ：将主设备中的 EZ 数据写入到所寻址的从设备存储器空间内，以及主设备从所寻址的从设备存储器空间内读取数据。

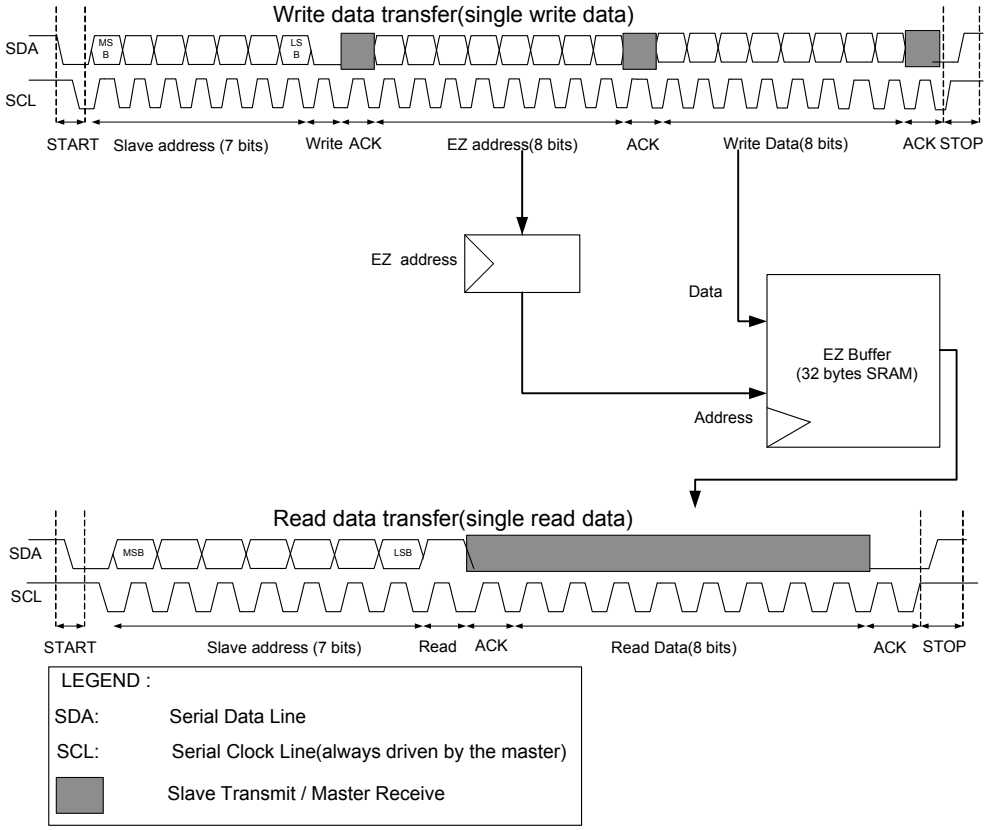
15.4.1 存储器阵列的写操作

通过 I2C 写传输对存储器阵列索引进行 EZ 写操作。使用被传输的第一个写数据可以将主设备中的 EZ 地址发送到从设备内。写数据的 5 个最低有效位可作为从设备中 “ 新 ” 的 EZ 地址使用。写传输中的其它写数据元素是写入存储器阵列中的字节。各字节被写入到存储器阵列时，从设备自动递增 EZ 地址。当 EZ 地址超过 32 个存储器位置时，它将返回 0。

15.4.2 存储器阵列的读操作

通过 I2C 读传输对存储器阵列索引进行 EZ 读操作。 EZ 读操作依赖之前进行的 EZ 写操作（EZ 写操作设置从设备上的 EZ 地址）。第一个接收到的读数据是 EZ 地址存储器位置上存储器阵列中的字节。读取存储器阵列中的各字节时， EZ 地址将自动递增。当 EZ 地址超过 32 个存储器位置时，它将返回 0。

图 15-4. EZI2C 写和读数据传输



请参考第 91 页上的 EZ 从设备模式的传输示例 中所示的示例。

15.5 I2C 寄存器

通过对表 15-4 内显示的配置、控制和状态寄存器集进行读 / 写来控制 I2C 接口。

表 15-4. I2C 寄存器

寄存器	函数
SCB_CTRL	用于使能固定功能 I2C 模块并选择串行接口的类型 （SPI、 UART 或 I2C）。还可用于选择内部、外部时钟模式以及 EZ 或非 EZ 操作模式。
SCB_I2C_CTRL	选择模式 （主设备 / 从设备）并根据接收器 FIFO 状态发送 ACK 或 NACK 信号。
SCB_I2C_STATUS	总线繁忙状态标志、主 / 从设备的读 / 写状态位， EZ 从设备地址存储。
SCB_I2C_M_CMD	使能主设备用以生成 START、 STOP 和 ACK/NACK 等信号。
SCB_I2C_S_CMD	使能从设备生成 CK/NACK 信号。
SCB_STATUS	指示外部时钟逻辑是否正在使用 EZ 存储器。该位可用于确定通过软件对 EZ 存储器进行访问是否安全。
SCB_I2C_CFG	滤波器配置，用于去除 SDA 和 SCL 线上的干扰。
SCB_TX_CTRL	用于使能发送器并指定数据帧的宽度；通过它还可以指定第一个传输位是 MSB （最高有效位）还是 LSB （最低有效位）。
SCB_TX_FIFO_CTRL	用于指定触发电平，从而可清除发送器 FIFO 和移位寄存器，并冻结 （FREEZE）发送器 FIFO。
SCB_TX_FIFO_STATUS	指示存储在发送器 FIFO 中的字节数、硬件读取数据帧的位置 （读取指针）、新数据帧被编写的位置 （写入指针），并决定发送器 FIFO 是否保存有效的数据。
SCB_TX_FIFO_WR	保存被写入到发送 FIFO 中的数据帧。该寄存器的功能类似于 PUSH 操作。
SCB_RX_CTRL	它的功能与 SCB_TX_CTRL 寄存器相同，但它是用于接收器的。
SCB_RX_FIFO_CTRL	它的功能与 SCB_TX_FIFO_CTRL 寄存器相同，但它是用于接收器的。
SCB_RX_FIFO_STATUS	它的功能与 SCB_TX_FIFO_STATUS 寄存器相同，但它是用于接收器的。
SCB_RX_FIFO_RD	保存从接收器 FIFO 中读取的数据。取某个数据帧会移除 FIFO 中的数据帧；类似于 POP 操作。当通过软件读取数据帧时，会影响寄存器，即 FIFO 中的数据帧会被移除。
SCB_RX_FIFO_RD_SILENT	保存从接收器 FIFO 中读取的数据。读取某个数据帧并不会移除 FIFO 中的数据帧；类似于 PEEK 操作。
SCB_RX_MATCH	存储从设备地址，还可以将其作为从设备地址 MASK 使用。
SCB_EZ_DATA	存储 EZ 寄存器中的数据。

注意： PSoC 4 寄存器技术参考手册中的内容详细说明了各个 I2C 寄存器位。

15.6 I2C 中断

在下面条件下，固定功能 I2C 模块将生成中断。

- 仲裁失败
- 从设备地址匹配
- 检测到 I2C 总线的停止 / 启动事件
- 检测到 I2 总线错误
- 完成了 I2C 字节 / 字传输
- I2C TX FIFO 未满
- I2C TX FIFO 为空
- I2C RX FIFO 为空
- I2C RX FIFO 非空
- I2C RX FIFO 溢出
- I2C RX FIFO 已满

I2C 中断信号硬连线到 Cortex-M0 NVIC，并不能路由到外部引脚。

中断输出是所有可能中断源的逻辑 “ 或 ” （OR）。当满足任何已使能的中断条件时，将触发中断。通过使用中断状态寄存器可以确定实际的中断源。欲了解更多有关中断寄存器的信息，请参见 PSoC 4 寄存器的技术参考手册。

15.7 使能和初始化 I2C

下面内容说明了如何配置 I2C 模块的标准 （非 EZ）模式和 EZI2C 模式。

配置 I2C 标准 （非 EZ）模式

必须按下列步骤编程 I2C 接口。

1. 根据表 15-5，使用 SCB_I2C_CTRL 寄存器编程协议特殊信息，包括选择主设备 - 从设备的功能。

2. 使用 SCB_TX_CTRL 和 SCB_RX_CTRL 寄存器编程通用的发送器和接收器信息，如表 15-6 中所示。
   a. 指定数据帧的宽度。
   b. 使能发送器和接收器。
3. 分别使用 SCB_TX_FIFO_CTRL 和 SCB_RX_FIFO_CTRL 寄存器对发送器和接收器 FIFO 进行编程，如表 15-7 中介绍的内容：
   a. 设置触发电平。
   b. 清除发送器、接收器 FIFO 以及移位寄存器。
4. 通过编程 SCB_CTRL 寄存器使能固定功能模块，并选择 I2C 模式。这些寄存器位显示在表 15-8 中。欲了解 I2C 寄存器的完整说明内容，请查阅 PSoC 4 寄存器的技术参考手册。

表 15-5. SCB_I2C_CTRL 寄存器

位	名称	数值	说明
30	SLAVE_MODE	1	从设备模式
31	MASTER_MODE	1	主设备模式

表 15-6. SCB_TX_CTRL/SCB_RX_CTRL 寄存器

位	名称	说明
[3:0]	DATA_ WIDTH	‘DATA_WIDTH + 1’ 是指所传输或接收到的数据帧中位的数量。唯一的有效值为 7。

表 15-7. SCB_TX_FIFO_CTRL/ SCB_RX_FIFO_CTRL

位	名称	说明
[2:0]	TRIGGER_LEVEL	触发电平。当发送器 FIFO 所输出的数据低于此字段的值或接收器 FIFO 所接收到的数据多于此字段的值时，则在相应情况下产生一个发送器或接收器的触发事件。
16	CLEAR	该位被设为 ‘1’ 时，发送器或接收器 FIFO 和移位寄存器都被清除。
17	FREEZE	该位被设为 ‘1’ 时，硬件对发送器或接收器 FIFO 进行的读 / 写操作不产生任何影响。冻结操作不会增加 TX 或 RX FIFO 的读 / 写指针。

表 15-8. SCB_CTRL 寄存器

位	名称	数值	说明
[25:24]	MODE	00[a]	I2C 模式
31	ENABLED	0	固定功能的 I2C 模块被使能
		1	固定功能的 I2C 模块被禁用

a. 其它值不适用

配置 EZI2C 模式

要想将固定功能 I2C 模块配置为 EZI2C 模式，请设置下面各个 I2C 寄存器位

1. 通过将 ‘1’ 写入 SCB_CTRL 寄存器的 EZ_MODE 位（位 10）选择 EZI2C 模式。
2. 执行配置 I2C 标准 （非 EZ）模式中所描述的第 2 到第 4 步。
3. 设置 SCB_I2C_CTRL 寄存器中的 S_READY_ADDR_ACK （位 12）和 S_READY_DATA_ACK （位 13）。

15.8 I2C 中的内部和外部时钟模式

固定功能I2C支持内部和外部时钟模式进行数据速率的生成。内部时钟模式使用来自 PSoC 系统总线时钟的时钟信号。外部时钟模式使用用户提供的时钟。在深度睡眠功耗模式下 (片上时钟无效)，外部时钟模式仅支持有限的功能。更多有关系统时序的信息，请参考第 53 页上的时钟系统章节。

外部时钟模式仅适用于下面各场合：

- 从设备功能。
- EZ 功能。由于 TX 和 RX FIFO 不支持外部时钟操作，因此该操作不适用于非 EZ 功能。

内部和外部时钟操作由 SCB_CTRL 寄存器的两个寄存器字段决定：

- **EC_AM_MODE （外部时钟模式地址匹配模式）**：表示 I2C 地址匹配操作为内部时钟模式 （ ‘0’ ）还是外部时钟模式 （ ‘1’ ）。
- **EC_OP_MODE （外部时钟模式）**：表示剩下的协议操作（除 I2C 地址匹配外）为内部时钟模式 （ ‘0’ ）还是外部时钟模式 （ ‘1’ ）。如上所述，外部时钟模式不支持非 EZ 功能。

通过下面两个寄存器字段，可以确定 I2C 的功能。需要根据活动、睡眠和深度睡眠系统功耗模式下所需要的行为来设置寄存器字段。设置不正确会引起具体功耗模式中的错误行为。表15-10和表15-10介绍的是I2C在EZ和非EZ模式下的设置。

15.8.1 I2C 非 EZ 工作模式

非 EZ 功能不支持外部时钟操作，这是因为该模式并不支持 FIFO。所以，在非 EZ 模式中， EC_OP_MODE 应始终设置为‘0’。但仍可以将EC_AM_MODE设置为‘0’或‘1’。表 15-9 显示的是各种可能情况的概述。EC_AM_MODE=0 和 EC_OP_MODE=1 的组合无效，并且模块不会响应。

EC_AM_MODE 为 ‘0’ 和 EC_OP_MODE 为 ‘0’. 只在活动 / 睡眠系统功耗模式下，该设置才有效。所有固定功能 I2C 的功能均为内部时钟模式。

EC_AM_MODE 为 ‘1’ 和 EC_OP_MODE 为 ‘0’. 只有在活动和深度睡眠系统功耗模式下，该设置才有效。在这两个模式中，I2C 地址匹配均由外部定时逻辑执行。外部时钟模式逻辑进行地址匹配时，它置位一个唤醒中断源位。通过该位，可以生成用于唤醒 CPU 的中断。

表 15-9. I2C 的非 EZ 模式

SPI 标准 （非 EZ）模式				
系统功耗模式	EC_OP_MODE = 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE=1	EC_AM_MODE=0
活动和睡眠	使用内部时钟匹配地址 使用内部时钟运行	使用外部时钟匹配地址 使用内部时钟运行	不支持	无效配置
深度睡眠	不支持	使用外部时钟匹配地址 使用内部时钟运行	不支持	

- 在活动系统功耗模式下， CPU 有效，并且唤醒中断源位被禁用 （相应掩码位为 ‘0’）。外部时钟模式逻辑执行地址匹配，而内部时钟模式逻辑执行余下的 I2C 传输过程。
- 在睡眠模式下，根据应用要求，唤醒中断源位可以被使能或禁用。其余的操作与活动模式中的操作相同。
- 在深度睡眠模式下， CPU 被关闭；如果使能了唤醒中断源位，当发生 I2C 活动时将唤醒 CPU。 CPU 唤醒会占用一段时间，正在进行的 I2C 传输可能被否定确认（NACK）或时钟被延长。在否定确认情况下，内部时钟逻辑将执行唤醒后的第一个 I2C 传输。在时钟延长情况下，内部时钟逻辑执行唤醒时正在进行 / 延长的传输。通过 SCB_I2C_CTRL 寄存器中的 S_NOT_READY_ADDR_NACK （位 14），可以确定外部时钟逻辑执行否定确认 （ ‘1’ ）还是时钟延长 （ ‘0’ ）。

15.8.2 EZ 工作模式

EZ 模式具有三种设置情况。当 EC_OP_MODE 为 ‘0’, EC_AM_MODE 可以设置为 ‘0’ 或 ‘1’ ；当 EC_OP_MODE 为 ‘1’ 时， EC_AM_MODE 必须设置为 ‘1’ 。表 15-10 提供了可能情况的概述。灰色显示的各单元格表示可能发生的设置。但并不推荐该设置，因为这些设置需要使用从外部时钟逻辑（从设备选择）切换为内部时钟逻辑（剩下操作）的开关。 EC_AM_MODE=0 和 EC_OP_MODE=1 的组合无效，并且模块不会响应。

表 15-10. I2C EZ 模式

I2C、 EZ 模式				
系统功耗模式	EC_OP_MODE= 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
活动和睡眠	使用内部时钟匹配地址 使用内部时钟运行	使用外部时钟匹配地址 使用内部时钟运行	使用外部时钟匹配地址 使用外部时钟运行	无效配置
深度睡眠	不支持	使用外部时钟匹配地址 使用内部时钟运行	使用外部时钟匹配地址 使用外部时钟运行	

- **EC_AM_MODE 为 ‘0’ 和 EC_OP_MODE 为 ‘0’**。只在活动 / 睡眠系统功耗模式下，该设置才有效。
- **EC_AM_MODE 为 ‘1’ 和 EC_OP_MODE 为 ‘0’**。该设置与 I2C 非 EZ 模式相同。
- **EC_AM_MODE 为 ‘1’ 和 EC_OP_MODE 为 ‘1’**。在活动和深度睡眠系统功耗模式下，该设置有效。

固定功能 I2C 的功能为外部时钟模式。请注意，该设置会导致对模块 SRAM 进行外部时钟访问的结果。这些访问会与器件中的内部时钟访问冲突。它会导致等待状态或总线错误。

SCB_CTRL 寄存器中的 FIFO_BLOCK 字段（位 17）能够确定生成等待状态 （ ‘1’ ）还是总线错误 （ ‘0’ ）。

15.8.3 从睡眠模式中唤醒

发生 I2C 地址匹配时，系统从睡眠或深度睡眠系统功耗模式中唤醒。固定功能 I2C 模块在地址匹配后将执行下面两个操作中的某一个：地址 ACK （应答）或 NACK （无应答）。

地址 ACK — I2C 从设备执行时钟延长，并等待，直到唤醒器件并应答地址为止。

地址 NACK — I2C 从设备立即对地址发生无应答 （NACK）信号。器件唤醒时间结束后，主设备必须再次轮询从设备。只有在从设备或多个主设备 - 从设备模式下，该选项才有效。

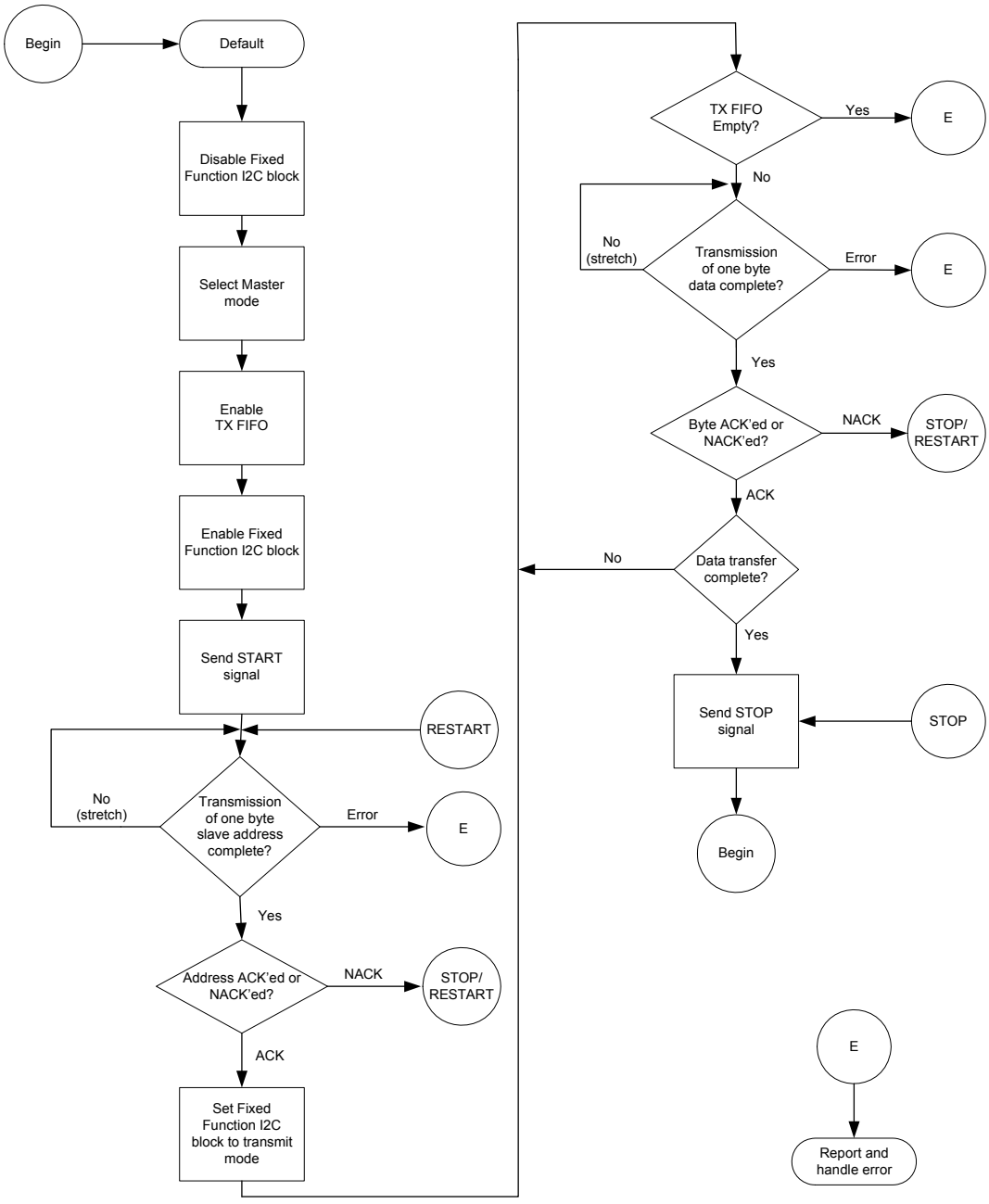
注意: 必须使能 INTR_I2C_EC 寄存器中的中断位 WAKEUP （位 0），以便切在换为睡眠模式时， I2C 通过从设备地址匹配来唤醒器件。

15.8.4 主设备模式的传输示例

主设备模式传输或接收数据。

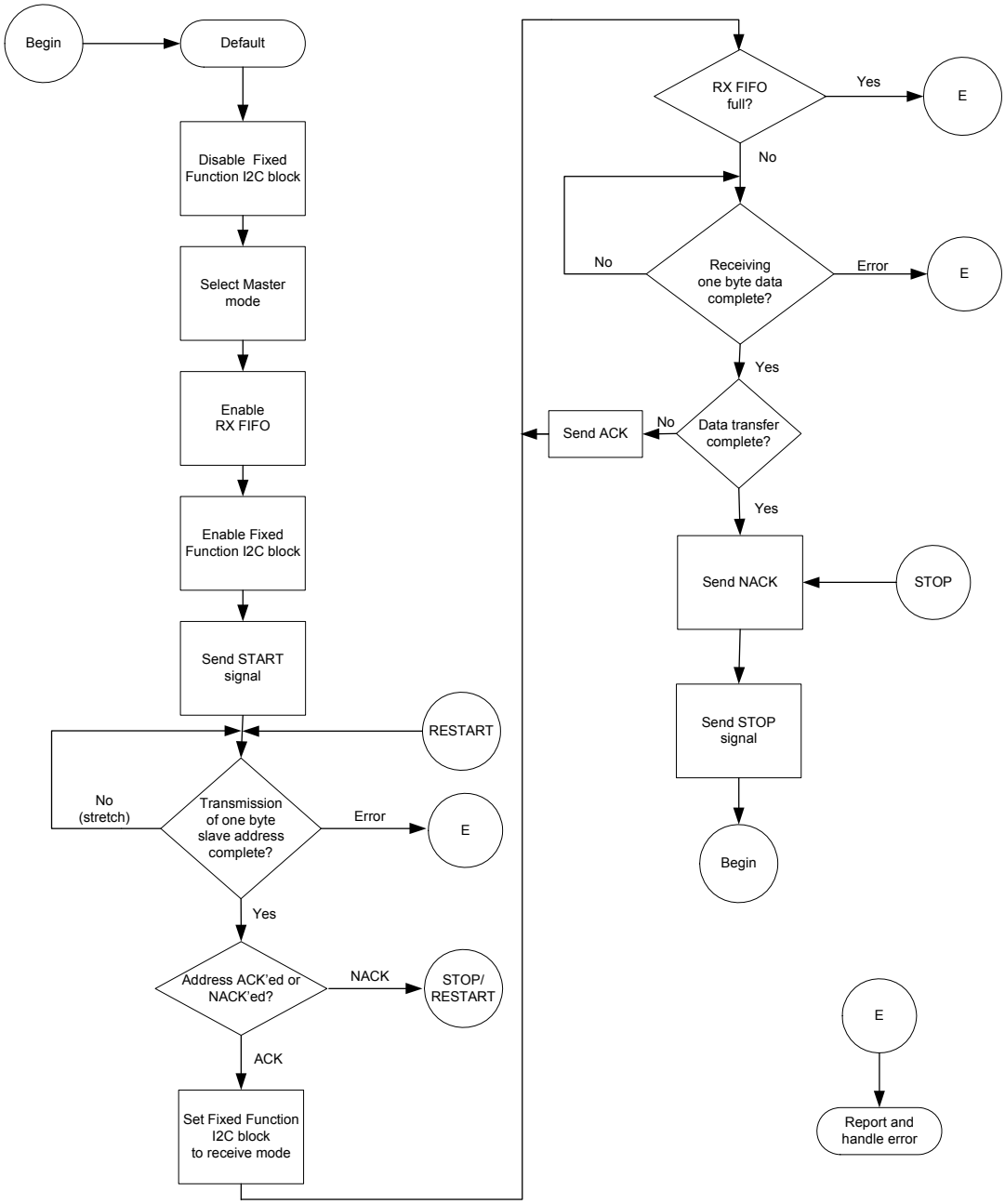
15.8.4.1 主设备传输

图 15-5. 单主设备模式写操作的流程图



15.8.4.2 主设备接收

图 15-6. 单主设备模式读操作的流程图

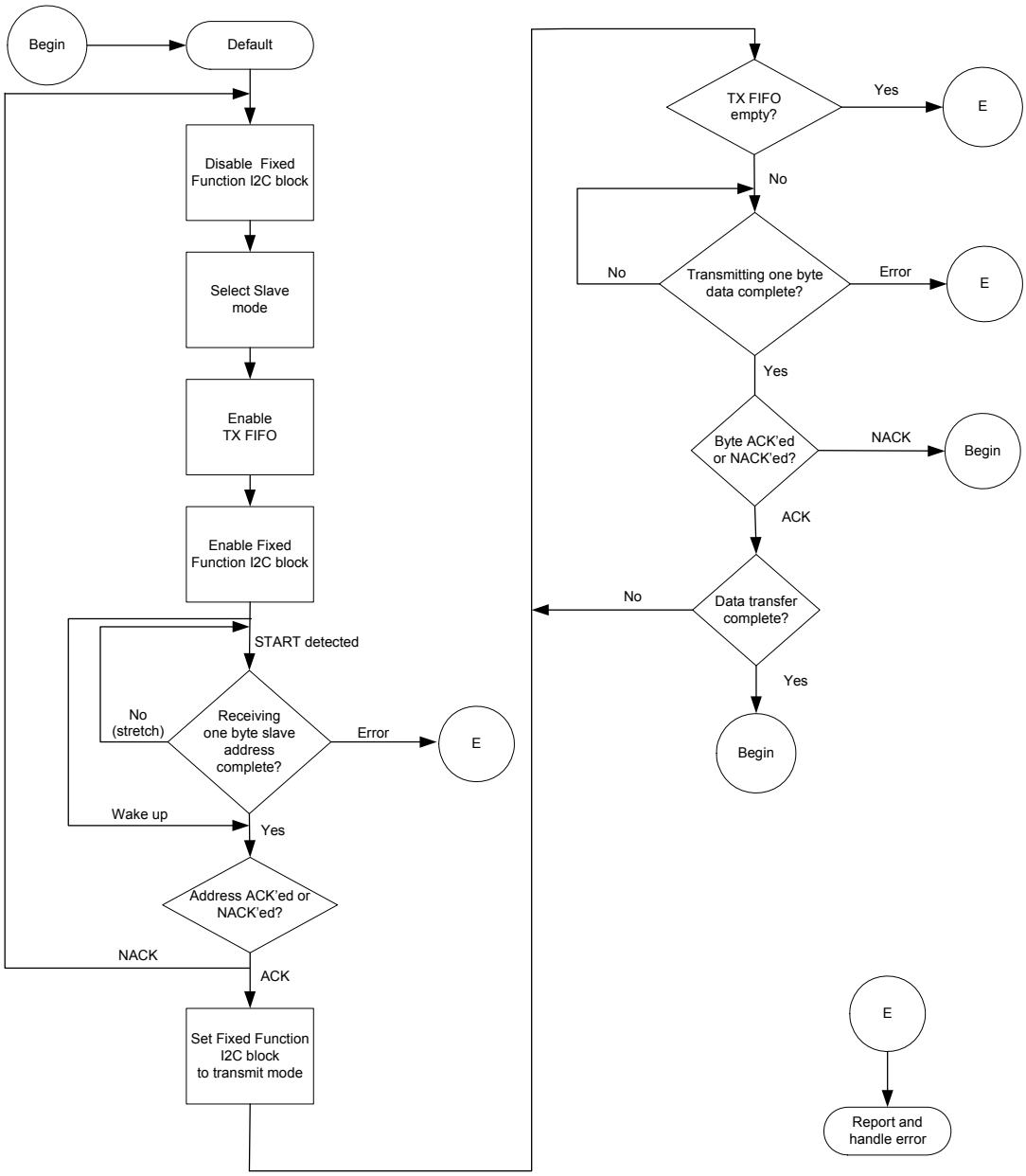


15.8.5 从设备模式的传输示例

从设备模式传输或接收数据。

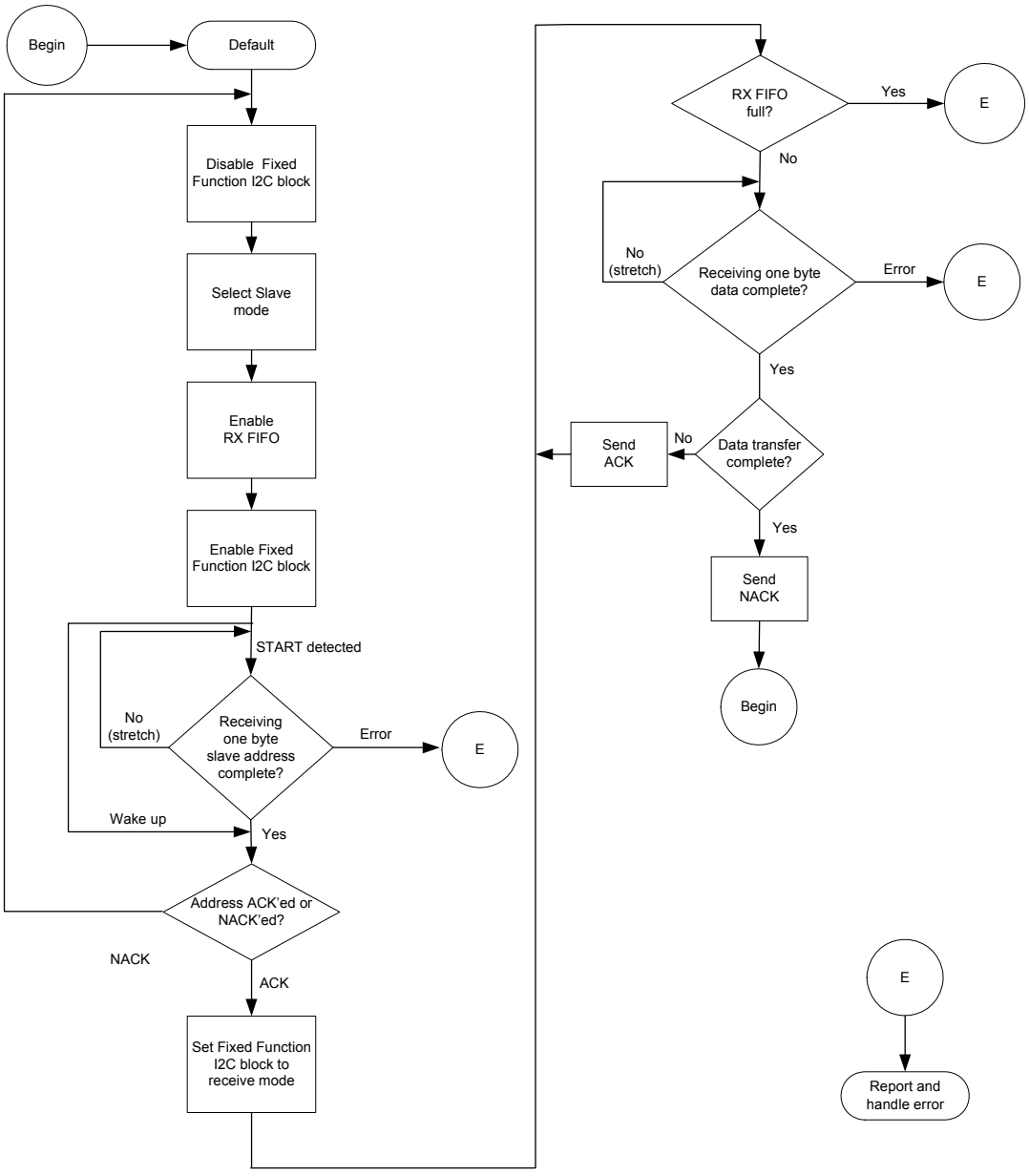
15.8.5.1 从设备传输

图 15-7. 从设备模式写操作的流程图



15.8.5.2 从设备接收

图 15-8. 从设备模式读操作的流程图

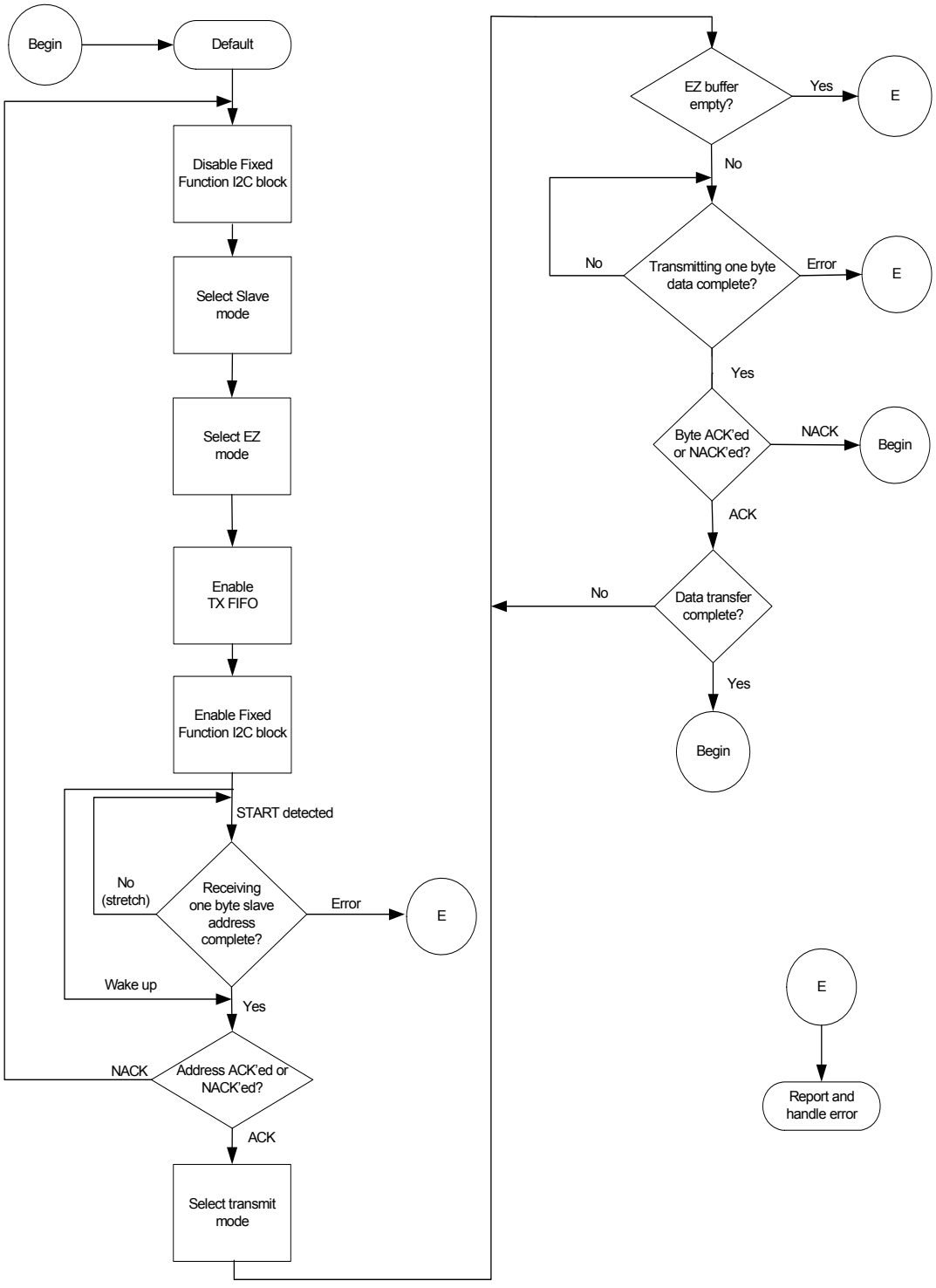


15.8.6 EZ 从设备模式的传输示例

EZ 从设备模式传输或接收数据。

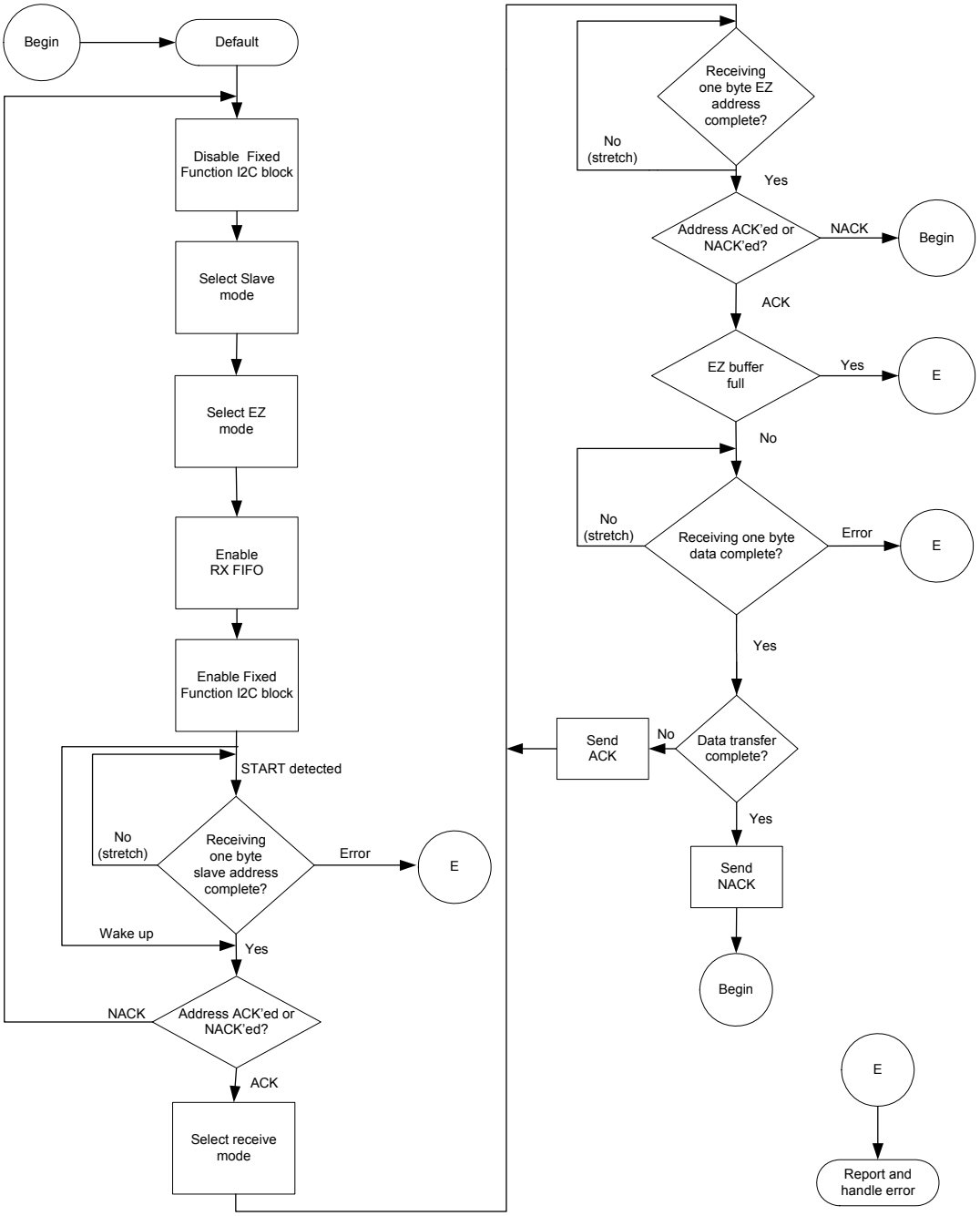
15.8.6.1 EZ 从设备传输

图 15-9. EZI2C 从设备模式写操作的流程图



15.8.6.2 EZ 从设备接收

图 15-10. EZI2C 从设备模式读操作的流程图

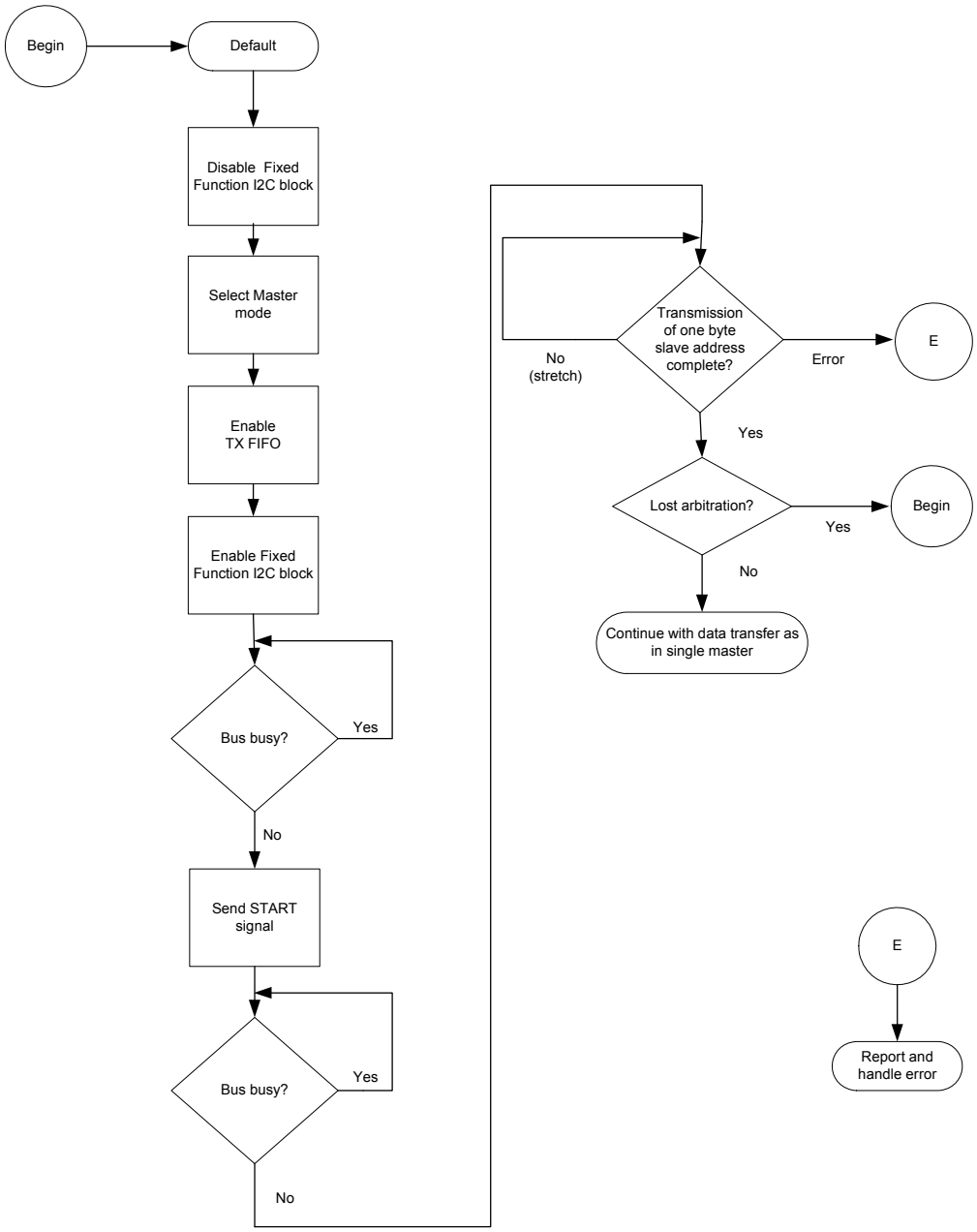


15.8.7 多个主设备模式的传输示例

在多个主设备模式下，使能或禁用从设备都能够传输数据。

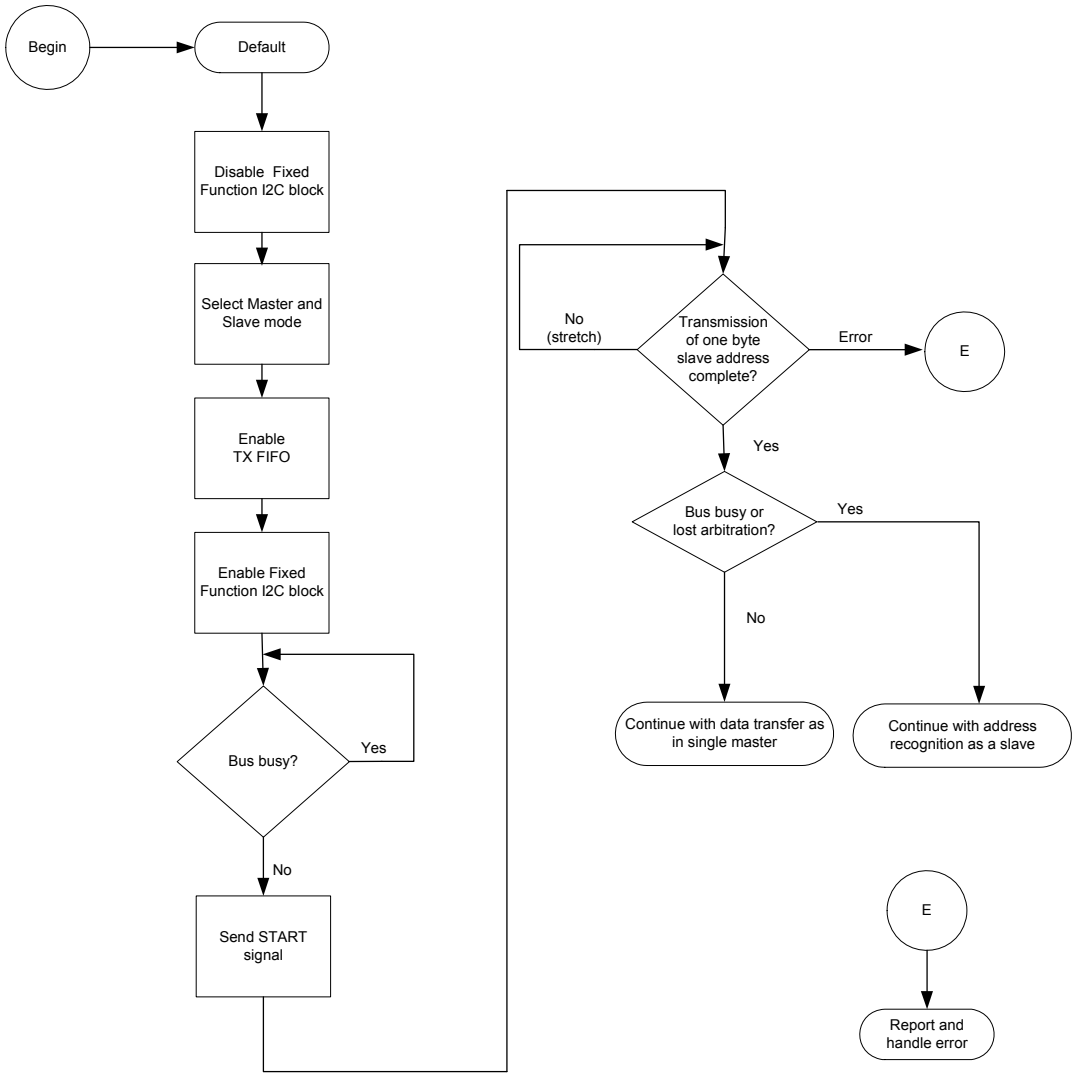
15.8.7.1 多个主设备 — 从设备不被使能

图 15-11. 多个主设备、从设备不被使能的流程图



15.8.7.2 多个主设备 — 从设备被使能

图 15-12. 多个主设备、从设备被使能的流程图



16. 定时器、计数器和 PWM

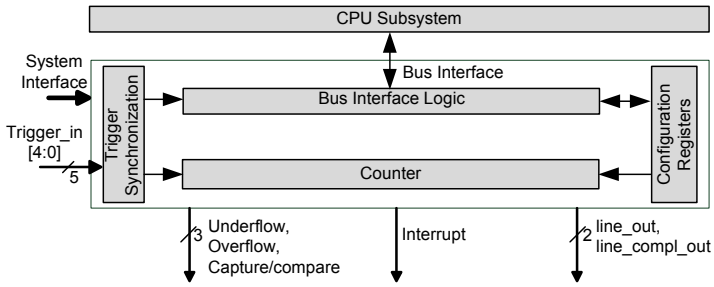
PSoC® 4 中的定时器、计数器和脉冲宽度调制器 （TCPWM）模块能够实现 16 位定时器、计数器、脉冲宽度调制器 （PWM）和正交解码器等功能。该模块用于测量输入信号的周期和脉冲宽度（定时器），捕获特定事件发生的次数（计数器）生成 PWM 信号或解码正交信号。本节介绍的是 TCPWM 模块的特性、实现以及操作模式。

16.1 特性

- 一个 16 位的定时器、计数器或脉冲宽度调制器 （PWM）
- TCPWM 模块支持下面各操作模式：
  - 定时器
  - 计数器
  - 捕获
  - 正交解码
  - 脉冲宽度调制
  - 伪随机 PWM
  - 带死区的 PWM
- 多个计数模式 — 向上、向下和向上 / 向下
- 时钟预分频 （1、 2、 4 ... 64、 128 分频）
- 比较 / 捕获值和周期值的双缓冲
- 支持以下中断：
  - 计数终值 — 计数器达到的最终值
  - 捕获 / 比较 — 计数值被捕获到捕获 / 比较寄存器或计数器的值等于比较值
- 下溢、溢出和捕获 / 比较输出信号可路由到专用的 GPIO
- PWM 的互补线路输出
- 根据上升沿、下降沿、双边沿和电平触发器等选项，可从专用 GPIO 选择 TCPWM 的启动、重载、停止、计数和捕获事件信号

16.2 框图

图 16-1. TCPWM 框图



该模块具有以下接口：

- 总线接口：将该模块连接至 CPU 子系统
- I/O 信号接口：将输入触发器 （如重载、启动、停止、计数和捕获）和输出信号 （如溢出 （OV）、下溢 （UN）和捕获、比较 （CC）连接至专用的 GPIO。
- 中断：根据计数终值 （TC）或 CC 条件为计数器提供中断请求信号。
- 系统接口：包括控制信号，如来自系统资源子系统的时钟和复位。

通过对 TCPWM 寄存器进行写操作，可以配置 TCPWM 模块。有关该模块所需要的所有寄存器的全部信息，请参见第 114 页上的 TCPWM 寄存器。

16.2.1 使能和禁用 TCPWM 模块中的计数器

通过设置控制寄存器 TCPWM_CTRL 中的 COUNTER_ENABLED 字段 （位 0），可以使能计数器。

注意： 使能计数器前，可先对它进行配置。如果配置后该计数器被使能，则寄存器将被更新为新的配置值。禁用计数器后，寄存器中的值仍被保留，直到再次使能 （重新配置）该计数器为止。

16.2.2 时钟

TCPWM 通过系统接口接收 HFCLK，用以对模块中所有的事件进行同步化。使能计数器时所生成的计数器使能信号（counter_en）对 HFCLK 进行门控，以提供一个计数器特定时钟 （counter_clock）。此外，还会对输出触发器 （本节中后面介绍）和 HFCLK 进行同步化。

时钟预分频：可对 counter_clock 进行预分频，分频值可分别为 1、2、4… 64、128。通过修改计数器控制（TCPWM_CNT_CTRL）寄存器中的 GENERIC 字段，可以实现该操作，如表 16-1 所示。

表 16-1. 预分频计数器时钟的位字段设置

GENERIC[10:8]	说明
0	1 分频
1	2 分频
2	4 分频
3	8 分频
4	16 分频
5	32 分频
6	64 分频
7	128 分频

注意 ：在正交模式和脉冲宽度调制模式下，如果存在死区时间 （PWM-DT），则不能对时钟进行预分频。

16.2.3 基于触发输入的事件

这些事件可由硬件或软件触发。

- 重载
- 启动
- 停止
- 计数
- 捕获 / 切换

硬件触发是指电平信号、上升沿、下降沿或双边沿。

图 16-2. TCPWM 触发选择和事件检测

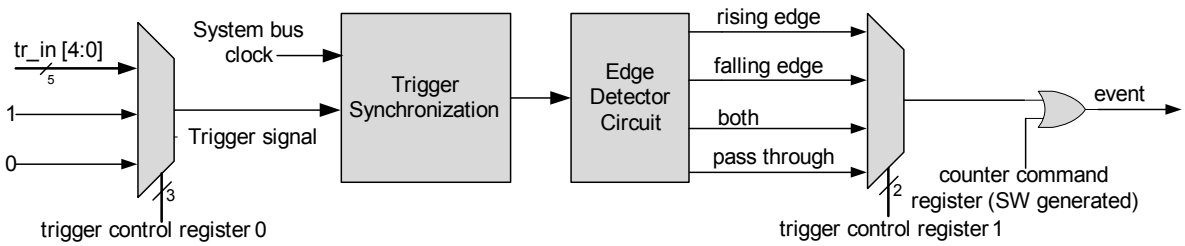


图 16-2 显示了 TCPWM 模块中的触发选择和事件检测。触发控制寄存器 0 （TCPWM_CNT_TR_CTRL0）选择 五个触发输入中的某一个作为事件信号。此外，也可以将常量 ‘0’ 和 ‘1’ 信号作为事件信号使用。

通过配置触发控制寄存器 1（TCPWM_CNT_TR_CTRL1）可以选择发生事件是由边沿 （上升沿、下降沿或双边沿）还是电平 （高电平或低电平）引发。可分别为每个触发事件选择边沿 / 电平配置。此外，通过写入到计数器指令寄存器（TCPWM_CMD）内，固件可生成一个事件，如图16-2所示。

在 TCPWM 模块的各种模式下，由这些触发引起的事件可能有不同的定义。

- **重载：**一个重载事件初始化并启动计数器。
  - 在向上计数模式下，计数寄存器（TCPWM_CNTx_COUNTER）被初始化为 ‘0’。
  - 在向下计数模式下，计数器使用存储在 TCPWM_CNTx_PERIOD 寄存器中的周期值进行初始化。
  - 在向上 / 向下计数模式下，计数寄存器被初始化为 ‘0’。
  - 在正交模式下，重载事件当做正交索引事件。索引 / 重载事件表示一个完整的旋转，并且可用于同步化正交解码。
- **启动：**启动事件用于启动计数。在发生停止事件后，或软件将计数寄存器重新初始化为某个值后，可使用启动事件。请注意，发生该事件时，计数寄存器不会初始化。
  - 在正交模式下，启动事件作为正交相位输入 phiB，这在第 104 页上的正交解码器模式中有详细说明。
- **计数：**根据计数器的配置，计数事件使它递增或递减。
  - 在正交模式下，计数事件作为正交相位输入 phiA。
- **停止：**停止事件停止计数器递增或递减。启动事件会再次启动计数操作。
  - 在 PWM 模式下，停止 （stop）事件作为禁止 （ kill）事件。禁止 （kill）事件会禁用所有的 PWM 输出线路。
- **捕获：**捕获事件会将计数器寄存器中的值复制到捕获寄存器内，并将捕获寄存器中的值复制到缓存捕获寄存器内。在 PWM 模式下，捕获事件作为切换事件。它将捕获 / 比较寄存器和周期寄存器中的值切换到相应的缓存寄存器内。通过该特性，可调制脉冲的宽度和频率。

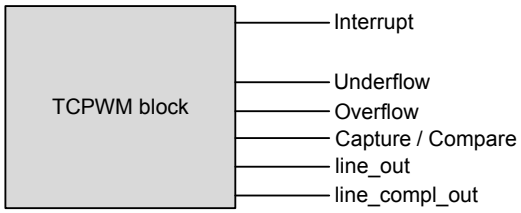
注意：

- 将同步化所有触发器的输入和 HFCLK。
- 当在同一个计数器时钟周期内发生多个事件时，可能会丢失一个或多个事件。对于高频率事件 （接近计数器频率）和使用预分频 （分频）计数器时钟的定时器配置，这种情况更容易发生。

16.2.4 输出信号

TCPWM 模块生成某些输出信号，如图 16-3 所示。

图 16-3. TCPWM 输出信号



16.2.4.1 发生触发事件时的信号

- 当向上计数和计数寄存器达到周期值时，计数器会生成内部上溢出 （OV）状态。
- 当向下计数和计数寄存器达到零值时，计数器会生成内部下溢出 （UN）状态。
- 当计数器运行并发生下面任何一个条件时，TCPWM 将生成捕获 / 比较 （CC）状态：
  - 计数器值等于比较值。
  - 发生捕获事件 — 发生捕获事件时， TCPWM_CNT_COUNTER 寄存器中的值将被复制到捕获寄存器内，捕获寄存器中的值将被复制到缓存捕获寄存器内。

注意： 发生该事件时，这些信号在 两个 HFCLK 周期内保持为逻辑高的电平状态。为保证可靠性，该触发事件的频率应该小于 HFCLK 频率的四分之一。例如，如果 HFCLK 的工作频率为 24 MHz，则引起发生触发的信号频率需要小于 6 MHz。

16.2.4.2 中断

TCPWM 模块从计数器提供一个专用的中断输出信号。在发生 TC 条件或 CC 条件时会生成中断。根据特定模式，这些条件的准确定义会不相同。

该模块使用了四个寄存器处理中断，如表 16-2 所示。

表 16-2. 中断寄存器

中断寄存器	位	名称	说明
TCPWM_CNT_INTR（中断请求寄存器）	0	TC	当到达计数终值时，该位被设置为 ‘1’。写 ‘1’ 清零。
	1	CC_MATCH	当计数值与捕捉 / 比较寄存器中的值相匹配时，该位将被置为 ‘1’。写 ‘1’ 清零。
TCPWM_CNT_INTR_SET（中断设置请求寄存器）	0	TC	写 ‘1’ 可设置中断请求寄存器中的对应位。读取该寄存器时，它反映的是中断请求寄存器的状态。
	1	CC_MATCH	写 ‘1’ 可以设置中断请求寄存器中的对应位。读取该寄存器时，它反映的是中断请求寄存器的状态。
TCPWM_CNT_INTR_MASK（中断屏蔽寄存器）	0	TC	中断请求寄存器中同 TC 位相对应的屏蔽位。
	1	CC_MATCH	中断请求寄存器中同 CC_MATCH 位相对应的屏蔽位。
TCPWM_CNT_INTR_MASKED（中断屏蔽请求寄存器）	0	TC	对相应的 TC 请求和屏蔽位进行逻辑 “ 与 ” 运算。
	1	CC_MATCH	对相应的 CC_MATCH 请求和屏蔽位进行逻辑 “ 与 “ 运算。

16.2.4.3 输出

TCPWM 有两个输出，line_out 和 line_compl_out（ line_out 的互补）。请注意，根据需求，通过配置 TCPWM_CNT_TR_CTRL2 寄存器，可使用 OV、 UN 和 CC 条件驱动 line_out 和 line_compl_out （请查看 表 16-3）。

表 16-3. 配置 OV、 UN 和 CC 条件的输出线路

字段	位	值	事件	说明
CC_MATCH_MODE 默认值 = 3	1:0	0	将 line_out 设置为 ‘1’	在发生比较匹配 （CC）事件时配置输出线路
		1	将 line_out 清零	
		2	反转 line_out	
		3	无变化	
OVERFLOW_MODE 默认值 = 3	3:2	0	将 line_out 设置为 ‘1’	在发生溢出 （OV）事件时配置输出线路
		1	将 line_out 清零	
		2	反转 line_out	
		3	无变化	
UNDERFLOW_-MODE 默认值 = 3	5:4	0	将 line_out 设置为 ‘1’	在发生下溢 （UN）事件时配置输出线路
		1	将 line_out 清零	
		2	反转 line_out	
		3	无变化	

16.2.5 功耗模式

TCPWM 模块可在活动模式和睡眠模式中工作。 TCPWM 模块由 V_{CCD} 供电。在深度睡眠模式中，仍供电给配置寄存器和其他逻辑，以保持配置寄存器的状态。请参见表 16-4。

表 16-4. TCPWM 模块的功耗模式

功耗模式	模块状态
活动	在活动模式下，模块正常供电，时钟正常运行，支持所有的操作。
睡眠	所有的计数器时钟都存在，但不能访问总线接口。
深度睡眠	在这种模式下，该模块仍然被供电，但并未提供总线时钟，无法工作。所有的配置寄存器将保持其状态。

16.3 各种操作模式

计数器模块可在六种操作模式下工作，如 表 16-5 所示。计数器控制寄存器 （TCPWM_CNTx_CTRL）中的 MODE [26:24] 字段，用于配置处在特定操作模式中的计数器。

表 16-5. 操作模式配置

模式	MODE 字段 [26:24]	说明
定时器	000	实现一个定时器或计数器在每个计数器时钟周期内，如果检测到计数事件，则计数器将加 1 或减 1。
捕获	010	使用捕获输入实现定时器或计数器。在每个计数器时钟周期内，如果检测到计数事件，则计数器将加 1 或减 1。当发生捕获事件时，会将计数器值复制到捕获寄存器中。
正交解码器	011	根据选定 （X1、 X2 或 X4）编码结构的两个相位输入执行正交解码器。执行解码器时可使计数器递增或递减。
PWM	100	通过 8 位时钟预分频器和缓存比较 / 周期寄存器实现边沿 / 中心对齐 PWM。
PWM-DT	101	通过可配置的 8 位死区时间 （在两个输出上）和缓存比较 / 周期寄存器实现边沿 / 中心对齐 PWM。
PWM-PR	110	使用一个 16 位线性反馈移位寄存器执行一个伪随机 PWM。

通过在 TCPWM_CNTx_CTRL 中设置 UP_DOWN_MODE[17:16] 字段，将模块配置为计数使用向上、向下、向上 / 向下计数模式，如表 16-6 所示。

表 16-6. 计数模式配置

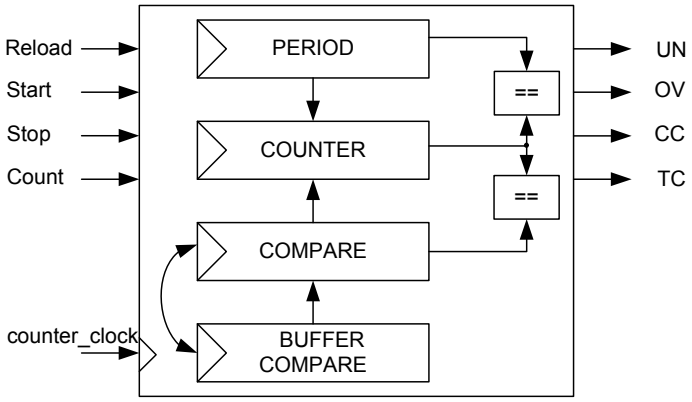
计数模式	UP_DOWN_MODE[17:16]	说明
UP 计数模式	00	计数器递增，直到周期值为止。当计数器达到周期值时，将生成计数终值 （TC）状态。
向下计数模式	01	计数器从周期值开始递减，直到等于数值 0 为止。当计数器的值为 0 时，将生成 TC 状态。
向上 / 向下计数模式 0	10	计数寄存器的值累加，当达到周期寄存器中的值时，计数寄存器的值递减，直到达到 “0” 只有到达 “0”，才生成 TC 状态。
向上 / 向下计数模式 1	11	与向上 / 向下计数模式 0 相似，但在计数器等于 “0” 和计数器达到周期值时，都会生成 TC 状态。

16.3.1 定时器模式

定时器模式通常用于测量某个事件发生的时长或测量两个事件间的时间差。

16.3.1.1 框图

图 16-4. 定时器模式框图



16.3.1.2 工作原理

可配置定时器计数，以在向上、向下和向上 / 向下计数模式中进行计数。也可以将其配置为连续模式或单触发模式。

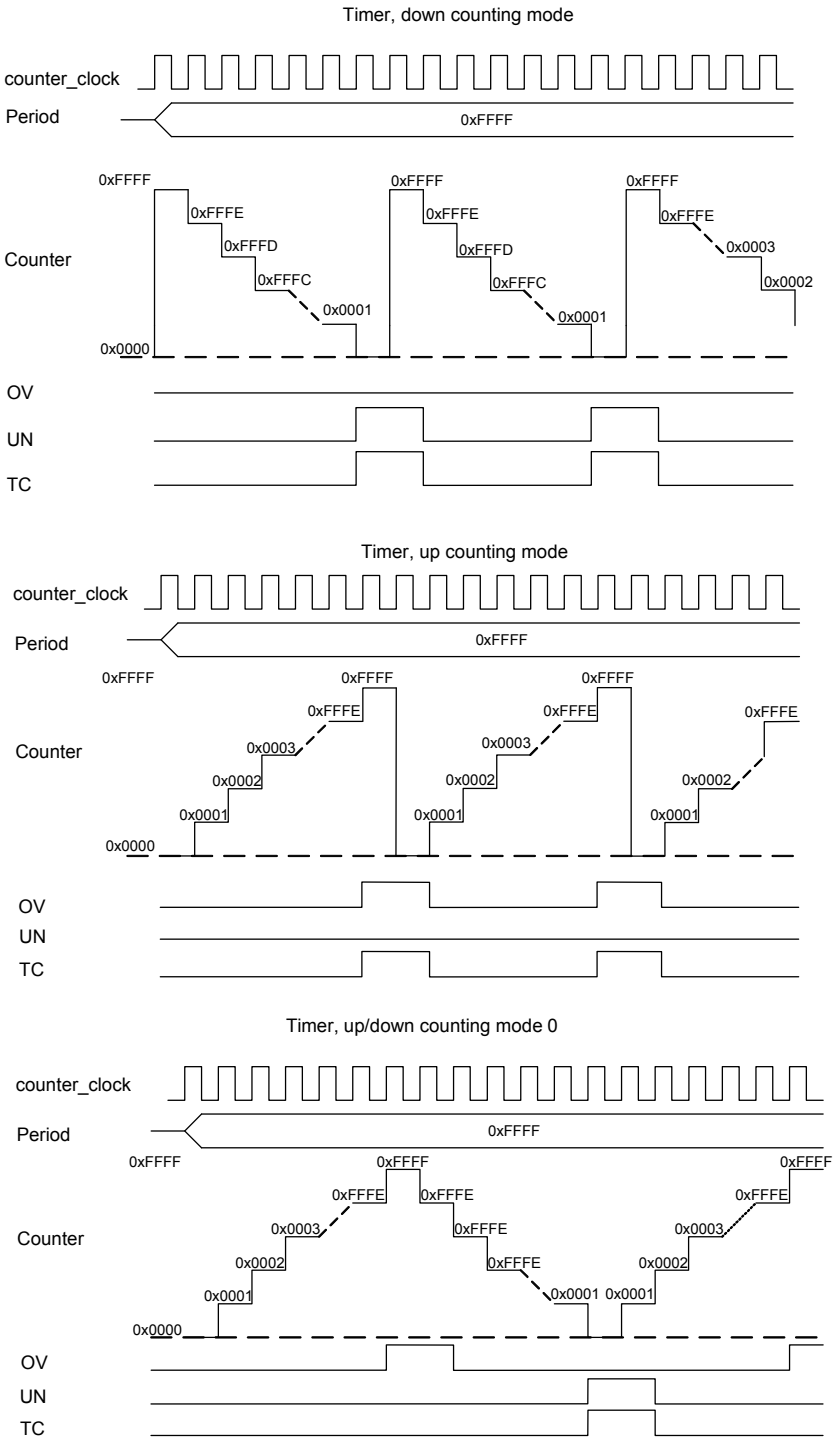
下面介绍的是定时器的工作原理：

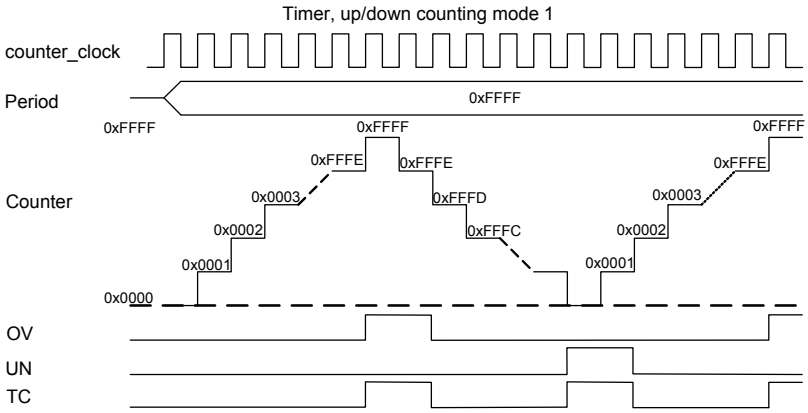
- 定时器是一个向上、向下或向上 / 向下计数器。
  - 当前的计数值被存储在计数寄存器（TCPWM_CNTx_COUNTER）中。**注意：** 不推荐在计数器运行过程中对该寄存器进行任何写操作。
  - 定时器的周期值被保存在周期寄存器内。
- 可在不同的计数模式下重新初始化计数器：
  - 在向上计数模式下，计数达到周期值之后，计数寄存器会自动重新加载 '0'。
  - 在向下计数模式下，计数寄存器达到零之后，计数寄存器会重新加载周期寄存器中的值。
  - 在向上 / 向下计数模式下，计数寄存器值不会更新，直到达到终值。当计数值等于零或周期值时，计数方向才发生改变。
- 当计数寄存器中的值等于比较寄存器中的值时，会生成 CC 状态。发生该条件时，如果通过计数器控制（TCPWM_CNT_CTRL）寄存器的 AUTO_RELOAD_CC 位字段使能了比较寄存器和缓存比较寄存器，这两个寄存器的值将被切换。可通过该条件生成中断请求。

图 16-5 显示的是在四种不同的计数模式下，计数器的定时器操作模式。周期寄存器包含最大计数器值。

- 在向上计数模式中， A 的周期值会导致 A+1 计数器周期（0 到 A）。
- 在向下计数模式中， A 的周期值会导致 A+1 计数器周期（A 到 0）。
- 在两种向上 / 向下计数模式（两者模式 0 和 1）中，A 的周期值导致 2*A 计数器周期 （0 到 A 并 A 回至 0）。

图 16-5. 各种计数模式下的定时器时序图





注意： OV 和 UN 信号在 HFCLCK 的 两个周期内保持为逻辑高电平状态，如 第 97 页上的发生触发事件时的信号所示。本节中的各个图表都假设 HFCLK 与计数器时钟相同。

16.3.1.3 配置定时器模式的计数器

下面介绍的是配置定时器操作模式的计数器以及受影响的寄存器位的各个步骤。

1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段写入零，可以禁用计数器。
2. 通过将 '000' 写入到 TCPWM_CNTx_CTRL 寄存器的 MODE[26:24] 字段内，可以选择定时器模式。
3. 设置 TCPWM_CNT_PERIOD 寄存器中所需要的 16 位周期。
4. 在 TCPWM_CNT_CC 寄存器中设置 16 位比较值，并在 TCPWM_CNT_CC_BUFF 寄存器中设置缓冲器比较值。如果需要每次发生 CC 条件时切换寄存器的值，请设置计数器控制寄存器中的 AUTO_RELOAD_CC 字段。
5. 通过对计数器控制 （TCPWM_CNT_CTRL）寄存器中的 GENERIC[10:8] 字段进行写操作，可设置时钟预分频值，如 表 16-1 所示。
6. 通过对 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_-MODE[17:16] 字段进行写操作，可以设置计数的方向，如 表 16-6 所示。
7. 分别将 0 和 1 写入到 TCPWM_CNT_CTRL 寄存器的 ONE_SHOT[18] 字段，可分别将定时器配置为连续模式或单触发模式。
8. 设置 TCPWM_CNT_TR_CTRL0 寄存器选择引起事件（重载、启动、停止、捕获和计数）的触发器。
9. 设置 TCPWM_CNT_TR_CTRL1 寄存器选择引起事件（重载、启动、停止、捕获和计数）的触发器边沿。
10. 根据需求，根据 TC 或 CC 条件设置中断，如第 97 页上的中断所示。
11. 通过将 '1' 写入到 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段使能计数器。如果硬件启动信号未被使能，则固件 （TCPWM_CMD 寄存器）必须提供一个启动触发信号，用以启动计数器。

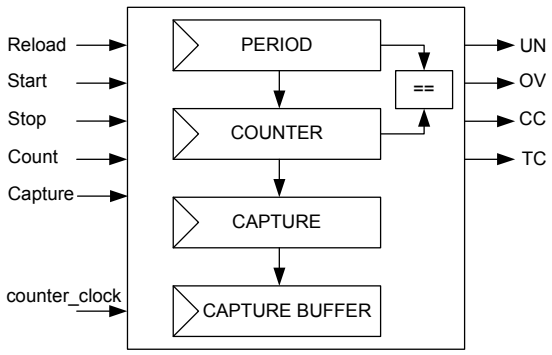
图 16-7 显示了在向上计数模式中的捕获行为。

16.3.2 捕获模式

在捕获模式下，通过固件对指令寄存器（TCPWM_CMD）进行写操作或通过捕获触发输入，都可随时捕获计数器的值。该模式用于测量周期和脉冲宽度。

16.3.2.1 框图

图 16-6. 捕获模式框图



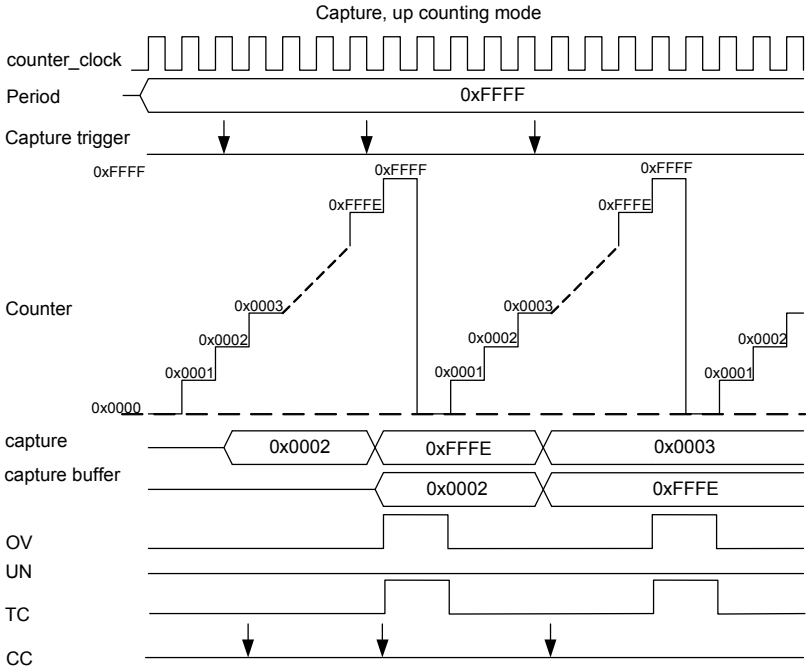
16.3.2.2 工作原理

通过配置计数器控制寄存器（TCPWM_CNT_CTRL）的 UP_DOWN_MODE[17:16] 位字段，可将计数器设置为向上、向下和向上 / 向下计数模式。

在捕获模式下可进行以下操作：

- 发生由硬件或软件生成的捕获事件时，当前的计数寄存器值被复制到捕获寄存器 （TCPWM_CNT_CC）内，并且捕获寄存器中的值被复制到缓冲器捕获寄存器（TCPWM_CNT_CC_BUFF）内。
- 将计数器的值复制到捕获寄存器内时，将在 CC 输出信号上生成一个脉冲。此状态还可用于生成中断请求。

图 16-7. 在捕获模式、向上计数模式中的计时器时序图



在该图中可以观察到：

- 周期寄存器包含最大计数值。
- 当计数器到达周期值时，会生成内部溢出 （OV）和 TC 状态。
- 只有在边沿上或通过软件，才能生成捕获事件。使用触发器控制寄存器 1 配置边沿检测。
- 按照下面的设置处理单个时钟周期内的多个捕获事件：
  - 偶数捕获事件 — 尚未观察到任何事件
  - 奇数捕获事件 — 观察到单一事件

当捕获信号频率超过了 counter_clock 频率时，可观察到上述现象。

16.3.2.3 *配置捕获模式下的计数器*

下面各步骤介绍的是配置捕获操作模式中的计数器以及受影响的寄存器位。

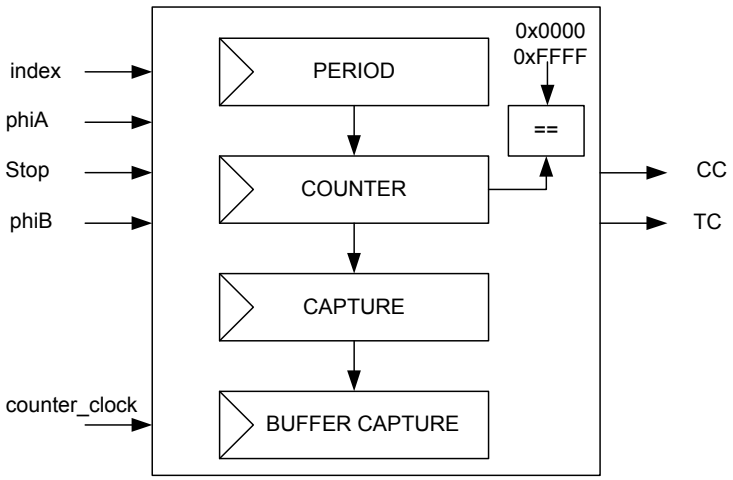
1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段写入 “0” 禁用计数器。
2. 通过将 ‘010’ 写入到 TCPWM_CNT_CTRL 寄存器的 MODE[26:24] 字段内设置捕获模式。
3. 设置 TCPWM_CNT_PERIOD 寄存器所需要的 16 位周期值。
4. 通过对 TCPWM_CNT_CTRL 寄存器的 GENERIC[10:8] 字段进行写操作设置时钟预分频，如 表 16-1 所示。
5. 通过对 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_-MODE[17:16] 字段进行写操作设置计数方向，如 表 16-6 所示。
6. 分别将 0 和 1 写入到 TCPWM_CNT_CTRL 寄存器的 ONE_SHOT[18] 字段内，可分别将计数器配置为连续模式或单触发模式。
7. 设置 TCPWM_CNT_TR_CTRL0 寄存器选择引起事件（重载、启动、停止、捕获和计数）的触发器。
8. 设置 TCPWM_CNT_TR_CTRL1 寄存器选择引起事件（重载、启动、停止、捕获和计数）的边沿。
9. 根据需求，设置发生 TC 或 CC 条件时的中断，如第 97 页上的中断所示。
10. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段内写入 “1” 使能计数器。如果硬件启动信号未被使能，则固件 （TCPWM_CMD 寄存器）必须提供一个启动触发信号，用以启动计数器。

16.3.3 正交解码器模式

正交解码器可用于确定旋转式设备（如伺服电机、音量控制车轮和电脑鼠标）的速度和位置。正交解码器信号可作为解码器的 phiA 和 phiB 输入使用。

16.3.3.1 框图

图 16-8. 正交模式框图



16.3.3.2 工作原理

只能在 counter_clock 中执行正交解码。可在下面三个子模式下执行该操作：X1、 X2 和 X4。可以通过计数器控制寄存器（TCPWM_CNT_CTRL）中的 QUADRATURE_MODE[21:20] 字段控制这些编码模式。该模式使用了双缓冲的捕获寄存器。

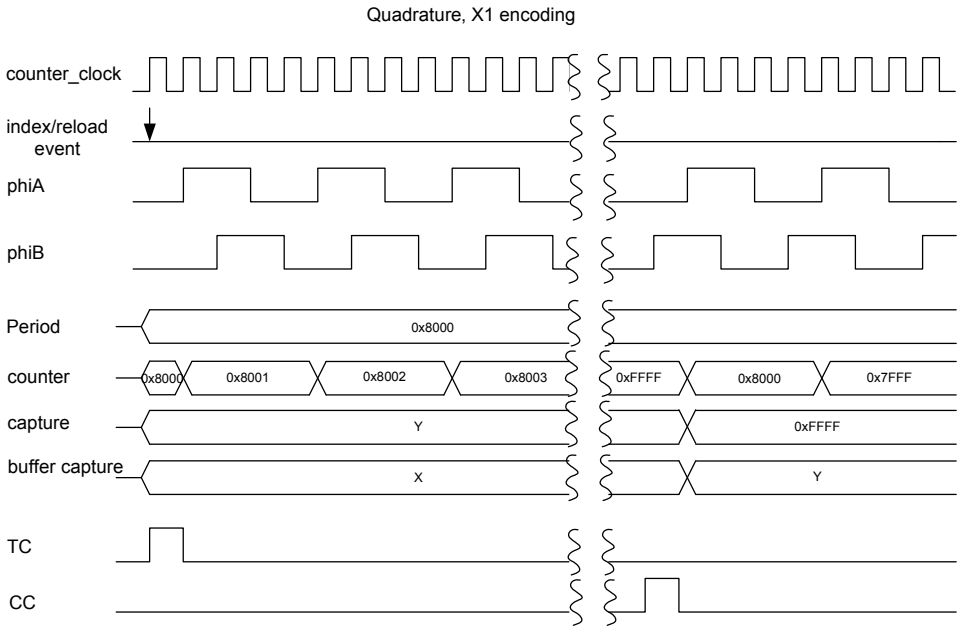
下面介绍的是正交模式下的操作：

- 正交相位 phiA 和 phiB：计数方向可通过 phiA 和 phiB 间的相位关系来确定。这些相位分别连接至计数和启动触发器输入，并作为解码器的硬件输入使用。
- 正交索引信号：该信号连接至重载信号，并作为硬件输入使用。该事件会生成 TC 状态，如 图 16-9 所示。

  在 TC 的情况下，计数器将被设置为 0x0000 （在递增计数模式中）或被设置为周期值 （在递减计数模式中）。

  注意： 在向下计数模式下，建议使用周期值 （中点值）为 0x8000。
- 当计数寄存器值达到 0x0000 或 0xFFFF 时，会在 CC 输出信号上生成一个脉冲。在 CC 的情况下，计数寄存器的值被设置为周期值 （在这种情况下，周期值为 0x8000）。
- 在 TC 或 CC 的情况下：
  - 计数寄存器的值被复制到捕获寄存器中
  - 捕获寄存器的值被复制到缓冲捕获寄存器中
  - 可以使用该条件生成中断请求
- 使用捕获寄存器中的值可确定引起事件的条件，并确定是否：
  - 发生计数器下溢出 （该值为 0）
  - 发生计数器下溢 （该值为 0xFFFF）
  - 发生索引 /TC 事件 （该值为非 0 或非 0xFFFF）
- 通过读取计数器状态 （TCPWM_CNTx_STATUS）寄存器中的 DOWN 位字段，可以确定当前计数的方向。数值 ‘0’ 表示前一个递增操作，数值 ‘1’ 表示前一个递减操作。图 16-9 说明了 X1 解码模式下的正交操作状况。
  - phiA的上升沿分别在phiB等于 ‘0’ 和phiB等于 ‘1’ 时递增和递减计数器。
  - 发生索引 / 重载事件时，使用周期值初始化计数寄存器。
  - 通过索引事件初始化计数器时，会生成计数终值事件。可以使用该事件生成中断。
  - 当计数寄存器达到 0xFFFF （最大计数寄存器值）值时，该计数寄存器的值将被复制到捕获寄存器内，并且计数寄存器将被初始化为周期值 （0x8000）。

图 16-9. X1 正交解码模式的时序图

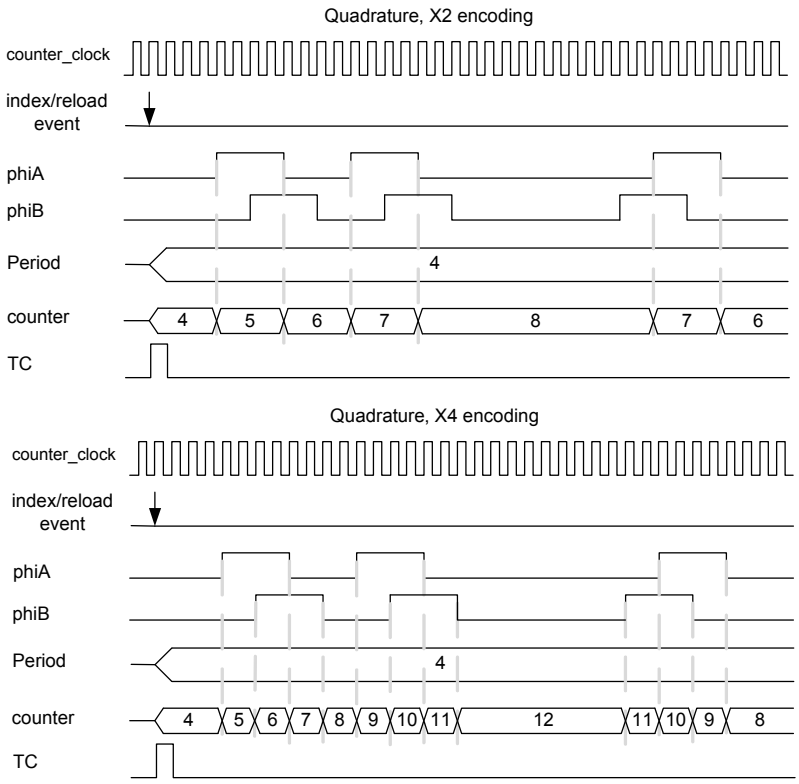


可在 counter_clock 上检测正交相位。在单一的计数器周期内，不能多次修改相位的值。

X2 和 X4 正交编码模式的计数速度分别比 X1 编码模式下的计数速度快一倍和三倍。

图 16-10 说明了 X2 和 X4 解码模式下的正交模式状况。

图 16-10. X2 和 X4 正交解码模式的时序图



16.3.3.3 配置正交模式的计数器

下面介绍的是配置正交操作模式的计数器以及受影响的寄存器位的各个步骤。

1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段写入零，可以禁用该计数器。
2. 通过将 ‘011’ 写入到 TCPWM_CNT_CTRL 寄存器的 MODE[26:24] 字段内，可以选择正交模式。
3. 设置 TCPWM_CNT_PERIOD 寄存器所需要的 16 位周期。
4. 通过对 TCPWM_CNT_CTRL 寄存器的 QUADRATURE_MODE[21:20] 字段进行写操作，可以设置所需要的编码模式。
5. 设置 TCPWM_CNT_TR_CTRL0 寄存器选择引起事件（索引和停止）的触发器。
6. 设置 TCPWM_CNT_TR_CTRL1 寄存器选择引起事件（索引和停止）的边沿。
7. 如果需要，可在发生 TC 或 CC 事件时设置中断，如 第 97 页上的中断所示。
8. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段内写入数值 ‘1’ 使能计数器。

16.3.4 脉冲宽度调制模式

PWM 模式也被称为数字比较器模式。比较输出是一个 PWM 信号，其周期取决于周期寄存器的值，它的占空比取决于比较和周期寄存器的值。

PWM 周期 = （周期值 / 计数器时钟频率）在左对齐和右对齐模式下

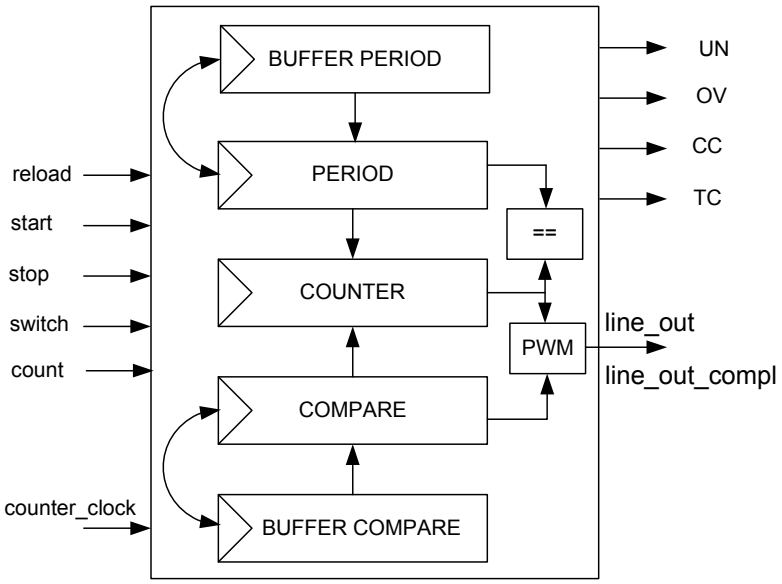
PWM 周期 = （2 × （周期值 / 计数器时钟频率）在中心对齐模式下

占空比 = （比较值 / 周期值）在左对齐和右对齐模式下

占空比 = （（周期值 - 比较值） / 周期值）在中心对齐模式下

16.3.4.1 框图

图 16-11. PWM 模式框图



16.3.4.2 工作原理

PWM模式可输出左对齐、右对齐、中心对齐或非对称的PWM信号。通过 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_MODE [17:16] 位选择计数器的递增、递减、递增 / 递减计数模式，可以得到所需要的输出对齐，如 表 16-6 所示。

CC、 OV 和 UN 信号均能够控制 PWM 输出线路。这些信号可以翻转输出线路，或可以通过配置 TCPWM_CNTx_TR_CTRL2 寄存器将该线路设置为逻辑 0 或逻辑 1。通过配置信号影响输出线路的方式，可以得到所需要的 PWM 输出对齐。

推荐的修改占空比的方法包括：

- 使用新值更新缓存周期寄存器和缓存比较寄存器。
- 在 TC 的情况下，如果出现了某个有效的切换事件，周期和比较寄存器将被自动更新为缓存周期和缓存比较寄存器。计数器控制寄存器中的 AUTO_RELOAD_CC 和 AUTO_RELOAD_PERIOD 字段均被设置为 1。当检测到某个切换事件时，该事件将被保存，直到发生下一个 TC 事件为止。通过信号 （设置事件检测时所选择的）不能触发切换事件。
- 发生下一个 TC 事件，并且出现了某个有效的切换事件前，需要完成缓存周期寄存器和缓存比较寄存器的更新；否则，切换操作将不影响寄存器的更新，如 图 16-13 所示。

在中心对齐模式下，在达到计数终值时会将输出线路设置为‘0’，并在发生 CC 条件时翻转该线路。

在发生重载事件时，计数寄存器被初始化，并在合适的模式下开始计数。每次计数，都会将计数寄存器的值和比较寄存器的值进行比较，用以在发生匹配时生成 CC 信号。

图 16-12 显示了有缓冲周期寄存器和比较寄存器的中心对齐脉冲宽度调制 （向上 / 向下计数模式 0）。

图 16-12. 中心对齐 PWM 的时序图

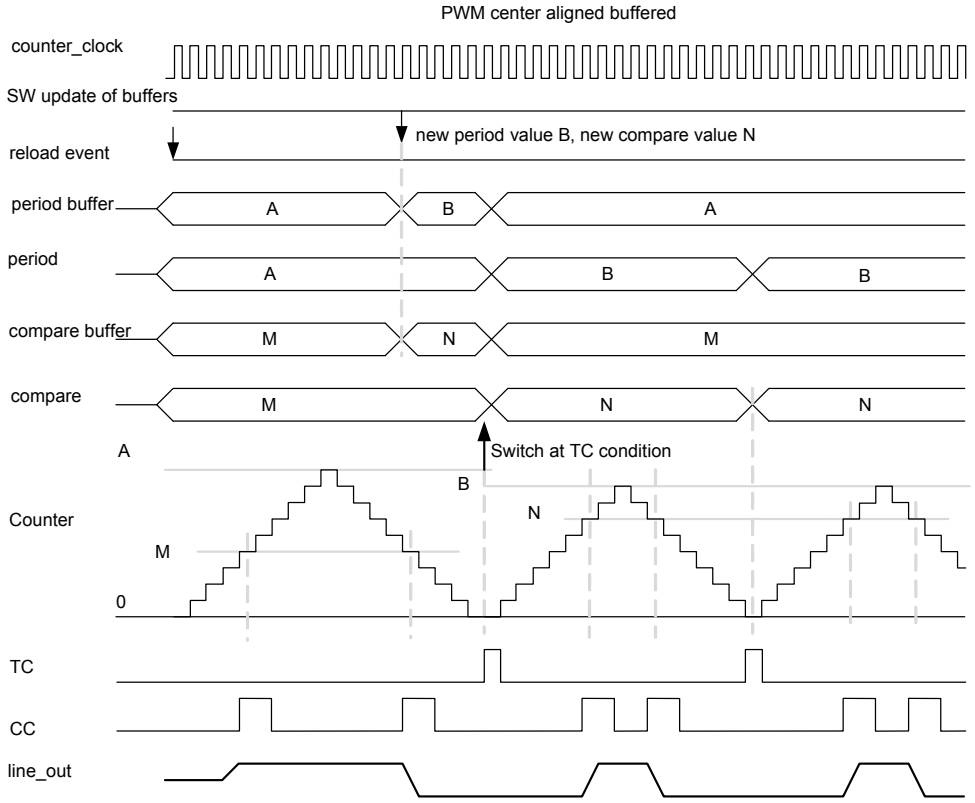
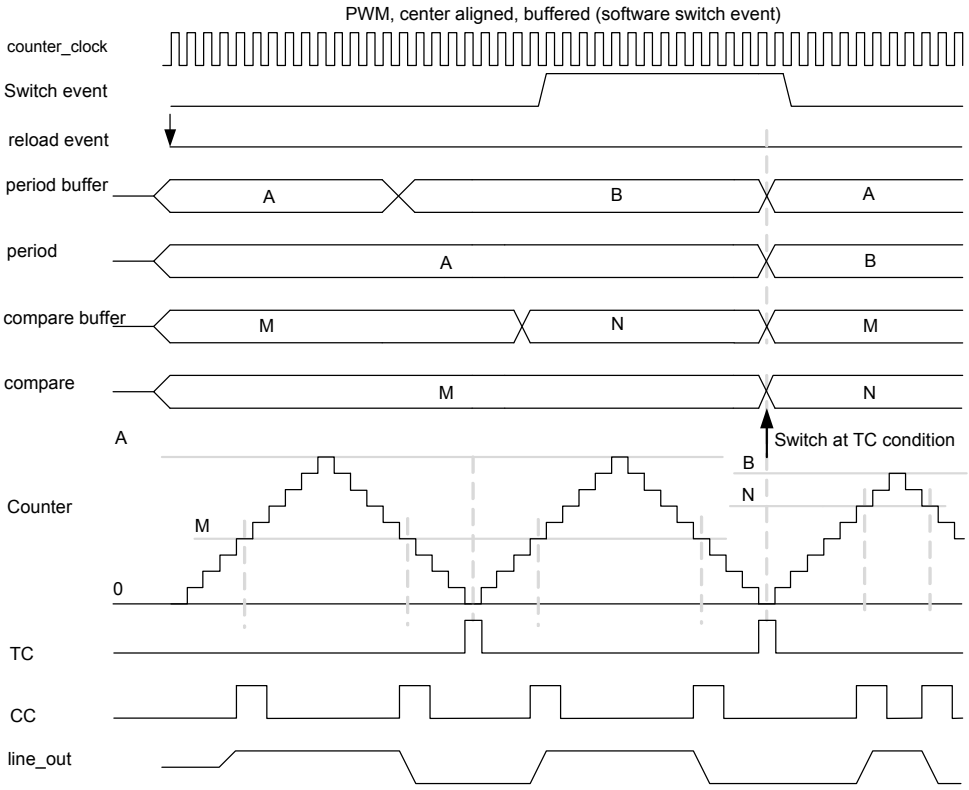


图 16-13 显示的是中心对齐 PWM，其中切换事件是由软件生成的：

- 仅在更新了周期缓存寄存器和比较缓存寄存器后，软件才会生成切换事件。
- 由于第二个 PWM 脉冲的更新被延迟 （达到计数终值后），所以第一个 PWM 将被重复。
- 注意：切换事件完成后，硬件将在 TC 条件下自动取消该事件。

图 16-13. 中心对齐 PWM （软件切换事件）的框图



16.3.4.3 *其他配置*

- 对于不对称 PWM，应使用向上 / 向下计数模式 1。当计数器达到 0 或周期值时，会发生 TC 事件。想要创建某个非对称 PWM，需要满足下面的条件：在每次发生 TC 事件（计数器达到 0 或周期值）时更改比较寄存器的值，同时，只能在发生其他 TC 事件 （只在计数器达到 0）时更改周期寄存器。
- 对于左对齐脉冲宽度调制，使用向上计数模式；配置 OV 时设置输出线为 ‘1’ 并配置 CC 时复位输出线为 ‘0’ 。请参见表 16-3。
- 对于右对齐脉冲宽度调制，使用向下计数模式；配置时复位输出线为 ‘0’ 并配置 CC 时设置输出线为 ‘1’ 。请参见表 16-3。

16.3.4.4 *终止 （kill）特性*

使用终止 （Kill）特性，可以立即禁用两个输出线路。这个事件是可编程的，通过修改计数器控制寄存器中的 PWM_STOP_ON_KILL 和 PWM_SYNC_KILL 字段即可停止计数器，如 表 16-7 所示。

表 16-7. 非同步停止 （Kill）功能的字段设置

PWM_STOP_ON_KILL 字段	注释
0	停止 （kill）事件停止 PWM 输出，但计数器仍然运行。
1	停止 （kill）事件停止 PWM 输出，并且计数器也会停止操作。

可将终止 （kill）事件编程为异步事件或同步事件，如表 16-8 所示。

表 16-8. 同步 / 异步停止 （Kill）的字段设置

PWM_SYNC_KILL 字段	注释
0	只要存在异步停止 （kill）事件，它将始终有效。此事件需要 ” 通过 “ 模式。
1	同步停止 （kill）事件禁用输出线路，直到发生下一个 TC 事件为止。此事件需要上升沿模式。

在同步终止 （kill）中，发生下一个 TC 条件前，不能启动 PWM。移除终止 （kill）输入后，如果要立即重启 PWM，那么应该使用异步终止 （kill）事件 （请参见表 16-8）。通过所

生成的停止 （stop）事件可以禁用两个输出线路。在这种情况下，重载事件应该在下降沿检测模式下使用相同的触发输入信号。

16.3.4.5 配置 PWM 模式的计数器

下面介绍的是配置PWM操作模式的计数器以及受影响的寄存器位的各个步骤。

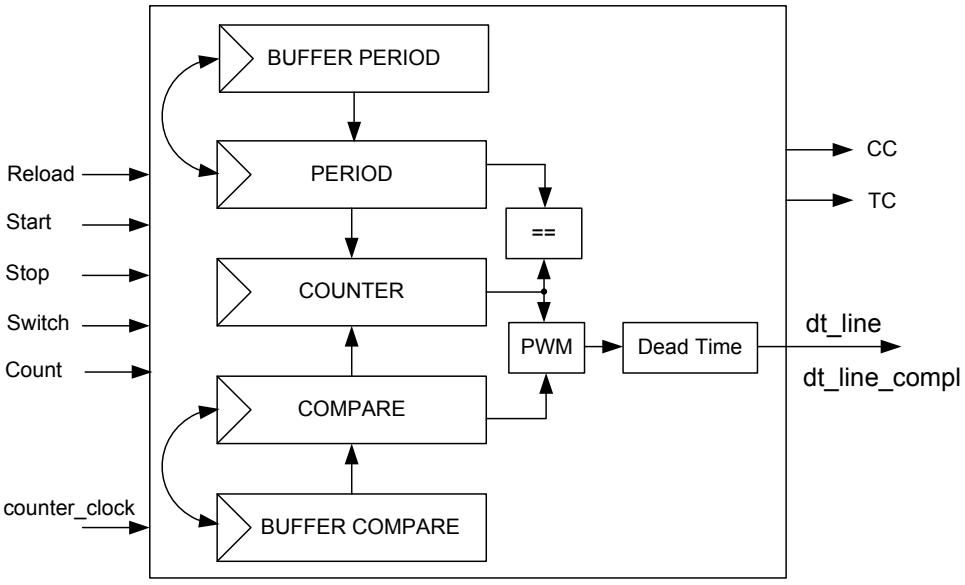
1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段写入 ‘0’ 禁用计数器。
2. 通过向 TCPWM_CNT_CTRL 寄存器的 MODE[26:24] 字段内写入 ‘100’ 选择 PWM 模式。
3. 通过对 TCPWM_CNT_CTRL 寄存器的 GENERIC[10:8] 字段进行写操作设置时钟预分频，如 表 16-1 所示。
4. 根据需求，分别将 TCPWM_CNT_PERIOD 寄存器中的 16 位值和 TCPWM_CNT_PERIOD_BUFF 寄存器中的缓存周期值设置为切换值。
5. 根据需求，分别将 TCPWM_CNT_CC 寄存器中的 16 位比较值和 TCPWM_CNT_CC_BUFF 寄存器中的缓存比较值设置为切换值。
6. 通过对 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_-MODE[17:16] 字段进行写操作设置计数方向，将其配置为左对齐、右对齐或中心对齐的 PWM，如 表 16-6 所示。
7. 根据需求，设置 TCPWM_CNT_CTRL 寄存器中的 PWM_STOP_ON_KILL 和 PWM_SYNC_KILL 字段。
8. 通过设置 TCPWM_CNT_TR_CTRL0 寄存器选择引起事件 （重载、启动、终止 （Kill）、切换和计数）的触发器。
9. 通过设置 TCPWM_CNT_TR_CTRL1 寄存器选择引起事件 （重载、启动、终止 （Kill）、切换、计数）的边沿。
10. 通过 TCPWM_CNT_TR_CTRL2 寄存器控制 line_out 和 line _out_compl 在发生 CC、 OV 和 UN 时置位、复位或翻转。
11. 如果需求，可设置发生 TC 或 CC 时产生中断，如第 97 页上的中断所示。
12. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段内写入 ‘1’ 使能计数器。如果硬件启动信号未被使能，则固件 （TCPWM_CMD 寄存器）必须提供一个启动信号启动计数器。

16.3.5 带死区模式的脉冲宽度调制

死区时间用于延迟 ‘line_out’ 和 ‘line_out_compl’ 信号的转换。它经过特定的时间间隔分离这两个信号的转换边沿。从两个互补输出线路得到这两个信号，即 “dt_line” 和 “dt_line_compl”。在死区期间，比较输出和互补比较输出均在固定时长内保持逻辑 0 状态。通过死区特性，可以生成两个非重叠的 PWM 脉冲。使用该特性，最多可生成长达 255 个时钟周期的死区时间。

16.3.5.1 框图

图 16-14. PWM-DT 模式框图



16.3.5.2 工作原理

带有死区时间模式的 PWM 操作状况如下介绍：

- 在 PWM line_out 的上升沿，根据 UN、OV 和 CC 事件，死区模块将 dt_line 和 dt_line_compl 设置为 ‘0’。
- 根据寄存器中配置的周期加载并计数死区周期。

- 死区周期完成后， dt_line 被设置为 ‘1’ 。
- 在 PWM line_out 的下降沿，根据 UN、OV 和 CC 事件，死区模块将 dt_line 和 dt_line_compl 设置为 ‘0’ 。
- 根据寄存器中所配置的周期来加载并计数死区周期。
- 死区周期完成后， dt_line_compl 被设置为 1。
- 零死区周期不会对 dt_line 产生任何影响，它与 line_out 相同。
- 当死区时间的持续时长等于或超过了脉冲的宽度时，脉冲将被移除。

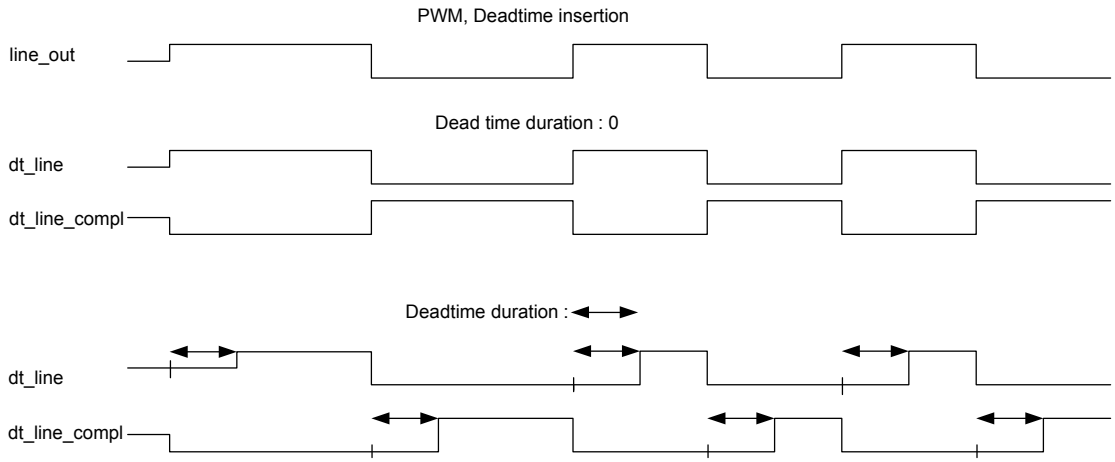
该模式遵循 PWM 模式并支持 PWM 模式的下述可用特性：

- 多种输出对齐模式
- 两个互补输出线路 （dt_line 和 dt_line_compl）分别由 PWM “line_out” 和 “line _out_compl” 生成。
  - 带有同步模式和异步模式的停止 （Stop） / 停止 （kill）事件
  - 比较和缓冲比较寄存器以及周期和缓冲周期寄存器的有条件切换事件

该模式不支持时钟预分频。

图 16-15 下图显示了如何根据 PWM 输出线路 “line_out” 生成互补输出线路 “dt_line” 和 “dt_line_compl” 。

图 16-15. 带和不带死区的 PWM 的时序图



16.3.5.3 *配置 PWM （带死区模式）的计数器*

下面各步骤介绍的是配置 PWM（带死区）操作模式的计数器以及受影响的寄存器位。

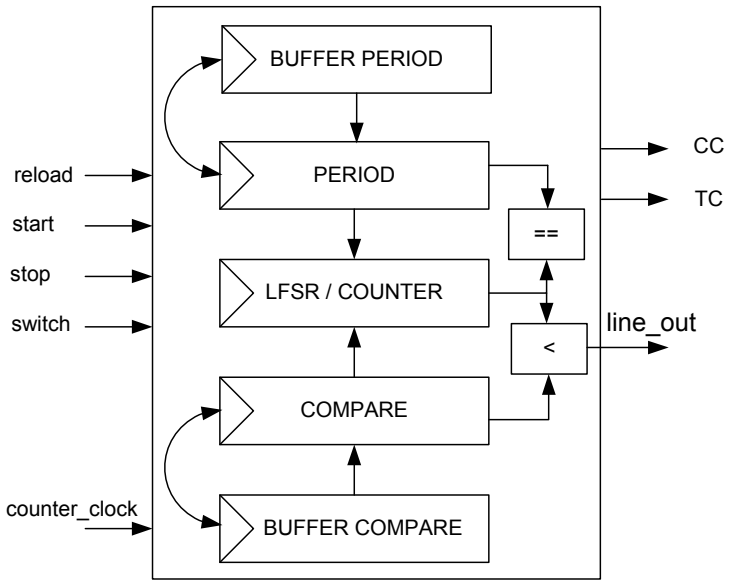
1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段写入 ‘0’ 禁用计数器。
2. 通过将 ‘101’ 写入到 TCPWM_CNT_CTRL 寄存器中的 MODE[26:24] 字段选择带死区模式的 PWM。
3. 通过对 TCPWM_CNT_CTRL 寄存器的 GENERIC[15:8] 字段进行写操作设置所需要的死区时间，如 表 16-1 所示。
4. 根据需求，分别将 TCPWM_CNT_PERIOD 寄存器中的 16 位值和 TCPWM_CNT_PERIOD_BUFF 寄存器中的缓存周期值设置为切换值。
5. 根据需求，可分别将 TCPWM_CNT_CC 寄存器中的 16 位比较值和 TCPWM_CNT_CC_BUFF 寄存器中的缓存比较值设置为切换值。
6. 通过对 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_-MODE[17:16] 字段进行写操作设置计数方向，将其配置为左对齐、右对齐或中心对齐的 PWM，如 表 16-6 所示。
7. 根据要求，设置 TCPWM_CNT_CTRL 寄存器中的 PWM_STOP_ON_KILL 和 PWM_SYNC_KILL 字段，如 第 107 页上的脉冲宽度调制模式所示。
8. 通过设置 TCPWM_CNT_TR_CTRL0 寄存器选择引起事件 （重载、启动、终止 （Kill）和切换）的触发器。
9. 通过设置 TCPWM_CNT_TR_CTRL1 寄存器选择引起事件 （重载、启动、终止 （Kill）、切换、计数）的边沿。
10. 通过 TCPWM_CNT_TR_CTRL2 寄存器控制 line_out 和 line _out_compl 在发生 CC、 OV 和 UN 时置位、复位或翻转。
11. 如果需要，可设置发生 TC 或 CC 时产生中断，如第 97 页上的中断所示。
12. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段内写入 ‘1’ 使能计数器。如果硬件启动信号未被使能，则固件 （TCPWM_CMD 寄存器）必须提供一个启动信号启动计数器。

16.3.6 脉冲宽度调制伪随机模式

该模式使用线性反馈移位寄存器 （LFSR）。 LFSR 是一个移位寄存器，其输入位是其前一个状态的线性功能。

16.3.6.1 框图

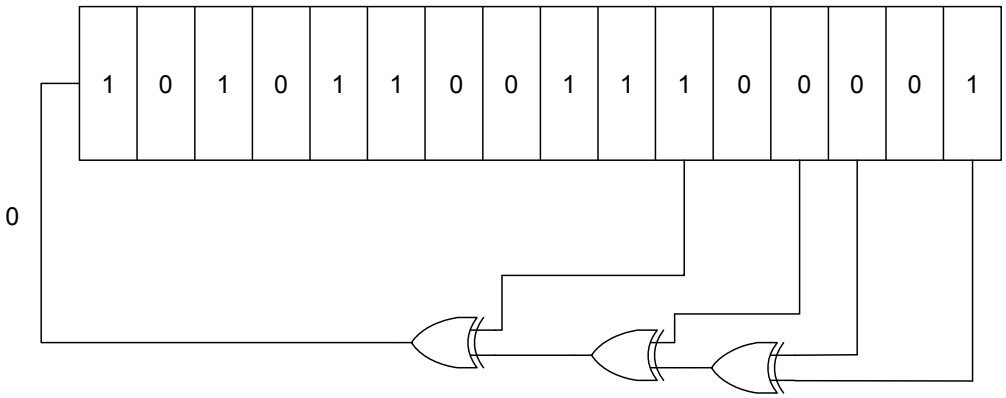
图 16-16. PWM-PR 模式的框图



16.3.6.2 工作原理

通过计数器寄存器，并使用多项式 $x^{16}+x^{14}+x^{13}+x^{11}+1$，可以执行 LFSR，如 图 16-17 所示。它以伪随机序列生成 [1, 0xFFFF] 范围内所有的数值。注意：应该使用非零值初始化计数器寄存器。

图 16-17. 使用计数器寄存器生成伪随机序列

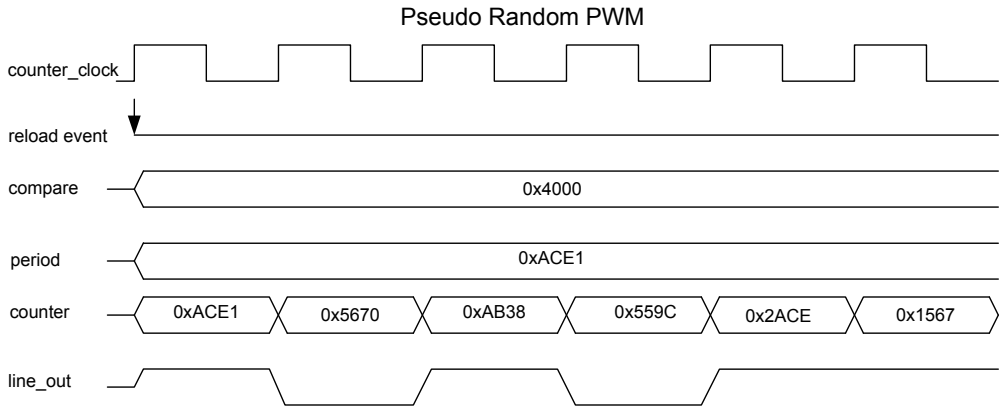


该过程包括以下各步骤：

- 当计数器寄存器中低 15 位的值小于比较寄存器中的值 （当计数器 [14:0] < 比较器 [15:0]）时， PWM 输出线路 ‘line_out’ 将被驱动为 1。比较值等于或大于 ‘0x8000’，都会使 PWM 输出线路等于 1。如果比较值等于 0，则 PWM 输出线路始终为 0。
- 重载事件与启动事件相同，但它并不会初始化计数器。
- 当计数器的值等于周期值时，将生成计数终值 （TC）状态。 LFSR 为特定的初始值生成可预测的计数器值模式。在 LFSR 的计数重复了 ‘n’ 次后，可以通过该可预测性计算计数器的值。计算得出的计数器值可作为周期值使用，并且在重复 ‘n‘ 次后将生成 TC 条件。
- 发生 TC 时，切换 / 捕获事件会有条件地切换比较和周期寄存器对 （根据计数器控制寄存器的 AUTO_RELOAD_CC 和 AUTO_RELOAD_PERIOD 字段）。
- 通过编程终止 （kill）事件可以禁止计数器，如前一节所述。
- 通过设置计数器控制寄存器中的 ONE_SHOT 字段，可以配置单触发模式。达到计数终值时，硬件将停止计数器。
- 在该模式下不会发生下溢、溢出和触发器条件等事件。
- 当计数器运行，并且它的值等于比较值时，将发生 CC 事件。图 16-18 描述了伪随机噪声的状况。

- 如果比较值等于 0x4000，则它能够生成 50% 的占空比（仅将 16 位计数器中低 15 位的值和比较寄存器的值进行比较）。

图 16-18. 伪随机 PWM 的时序图



捕获/切换输入信号可能切换比较寄存器和比较缓存寄存器间的值，也可以切换周期寄存器和周期缓存寄存器间的值。该功能可控制调制操作，它使用触发器输入信号调制两个不同的比较值。

注意 捕获 / 切换输入信号只能由某个边沿（上升沿、下降沿或两者）触发。记录该输入信号，直到到达下一个计数终值为止。

16.3.6.3 配置伪随机 PWM 模式的计数器

下面介绍的是配置伪随机PWM操作模式的计数器以及受影响的寄存器位的各个步骤。

1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段写入 ‘0’ 禁用计数器。
2. 通过向 TCPWM_CNT_CTRL 寄存器的 MODE[26:24] 字段内写入 “110” 选择伪随机 PWM 模式。
3. 根据需求，并为了切换各个值，分别设置 TCPWM_CNT_PERIOD 寄存器中所需要的周期 （16 位）和 TCPWM_CNT_PERIOD_BUFF 寄存器中的缓存周期值。
4. 为了切换各个值，分别设置 TCPWM_CNT_CC 寄存器中的 16 位比较值和 TCPWM_CNT_CC_BUFF 寄存器中的缓冲比较值。
5. 根据需求，设置 TCPWM_CNT_CTRL 寄存器中的 PWM_STOP_ON_KILL 和 PWM_SYNC_KILL 字段。
6. 通过设置 TCPWM_CNT_TR_CTRL0 寄存器选择引起事件 （重载、启动、终止 （Kill）、切换和计数）的触发器。
7. 通过设置 TCPWM_CNT_TR_CTRL1 寄存器选择引起事件 （重载、启动、终止 （Kill）、切换）的边沿。
8. 通过 TCPWM_CNT_TR_CTRL2 寄存器控制 line_out 和 line _out_compl 在发生 CC、 OV 和 UN 时置位、复位或翻转。
9. 如果需要，可设置发生 TC 或 CC 时产生中断，如第 97 页上的中断所示。
10. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段内写入 ‘1’ 使能计数器。

16.4 TCPWM 寄存器

表 16-9. TCPWM 寄存器列表

寄存器	注释	特性
TCPWM_CTRL	TCPWM 控制寄存器	使能计数器模块
TCPWM_CMD	TCPWM 指令寄存器	生成软件事件
TCPWM_INTR_CAUSE	TCPWM 计数器中断源寄存器	指定组合中断信号源
TCPWM_CNT_CTRL	计数器控制寄存器	用于配置计数器模式、编码模式、单触发模式、切换性能、停止 （kill）功能、死区时间、时钟预分频和计数方向等内容
TCPWM_CNT_STATUS	计数器状态寄存器	读取计数方向、死区时间和时钟预分频值；检查计数器是否运行
TCPWM_CNT_COUNTER	计数寄存器	包含 16 位计数器的值
TCPWM_CNT_CC	计数器比较 / 捕获寄存器	捕获计数器的值或将所需值同计数器值进行比较
TCPWM_CNT_CC_BUFF	计数器缓存比较 / 捕获寄存器	计数器 CC 寄存器的缓存寄存器；切换周期值
TCPWM_CNT_PERIOD	计数器周期寄存器	包含计数器的上限值
TCPWM_CNT_PERIOD_BUFF	计数器缓存周期寄存器	计数器周期寄存器的缓存寄存器；切换比较值
TCPWM_CNT_TR_CTRL0	计数器触发控制寄存器 0	选择特定计数器事件的触发器
TCPWM_CNT_TR_CTRL1	计数器触发控制寄存器 1	确定特定计数器输入信号的边沿检测
TCPWM_CNT_TR_CTRL2	计数器触发控制寄存器 2	在发生 CC、 OV 和 UN 条件时控制计数器输出线路
TCPWM_CNT_INTR	中断请求寄存器	检测到 TC 或 CC 条件是设置该寄存器位
TCPWM_CNT_INTR_SET	中断设置请求寄存器	设置中断请求寄存器中的相应位
TCPWM_CNT_INTR_MASK	中断掩码寄存器	中断请求寄存器的屏蔽
TCPWM_CNT_INTR_MASKED	中断屏蔽请求寄存器	对中断请求和屏蔽寄存器进行 “ 与 ” 运算

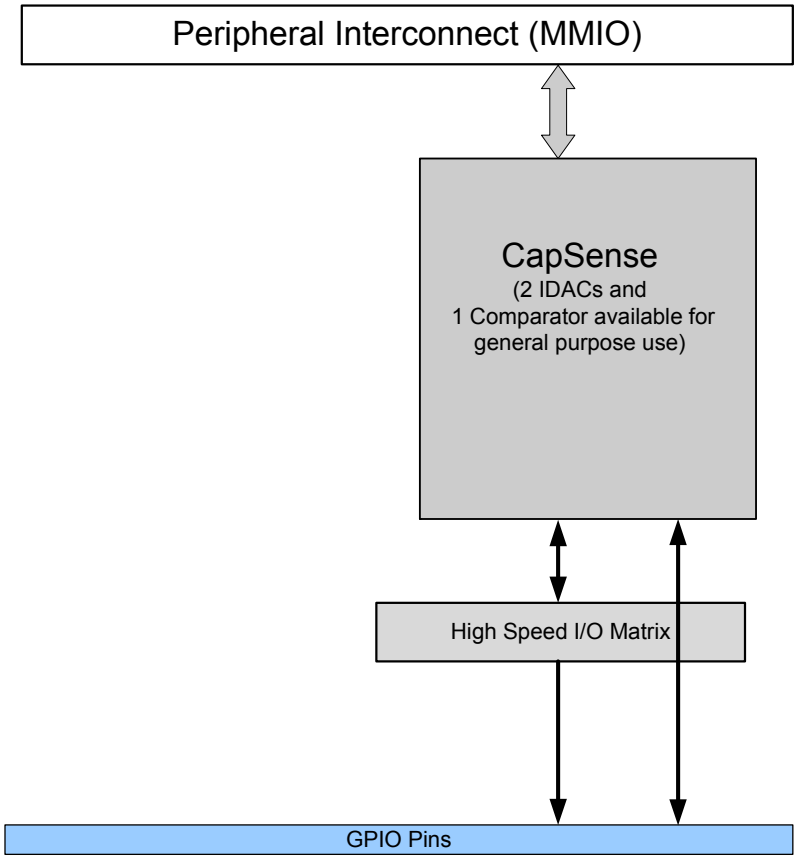
Section F: 模拟系统

本章节包括以下小节：

■ 第 117 页上的 CapSense 章节

顶层架构

模拟系统框图



17. CapSense

PSoC® 4 使用了一种电容式触摸感应方法，即 CapSense® Sigma Delta（CSD）。CapSense Sigma Delta 触摸感应方法能够提供行业中最佳的信噪比。 CSD 是硬件技术和固件技术的结合。本章节介绍的是在 PSoC4 中如何实现 CSD 硬件。

请参见PSoC 4 CapSense 设计指南的内容，了解有关基本CSD操作、有效的CapSense设计工具、容易使用的PSoC Creator™ 组件、使用调谐器 GUI 的调试功能以及设计注意事项等详细信息。

17.1 特性

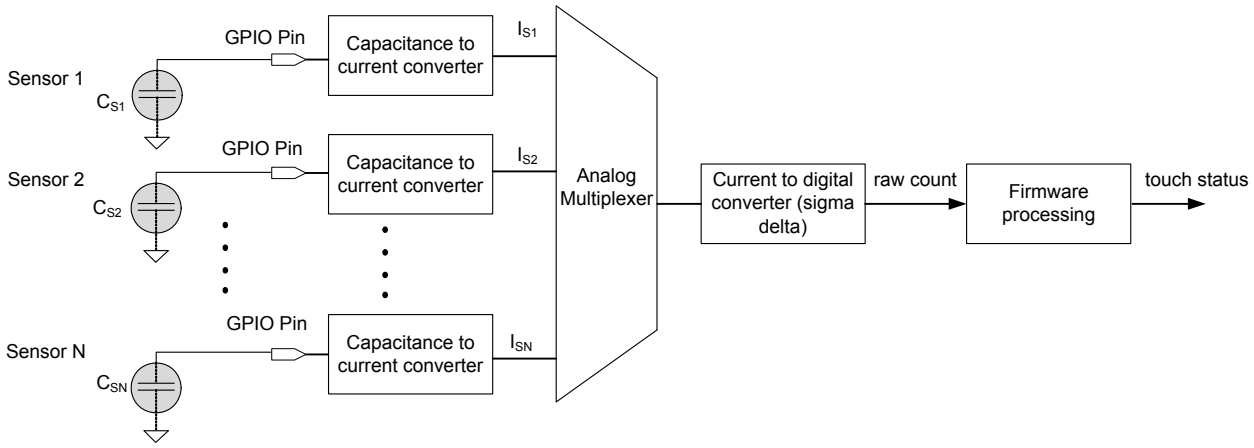
PSoC 4 CapSense 具有以下特性：

- 鲁棒的感应技术
- CSD 操作提供了最佳的 SNR
- 能够给各种的覆盖材料和厚度提供高性能感应
- SmartSense™ 自动调试技术
- 支持多达 16 个传感器
- 具有大范围的接近感应
- 16 个 GPIO 都有基于屏蔽信号的防水功能
- 低功耗
- 两个 IDAC 同时使用，提高扫描速度和 SNR
- 伪随机序列 （PRS）时钟源可降低电磁干扰 （EMI）
- 专用的充电槽电容，用来在屏蔽线路上快速转移电荷
- GPIO 单元预充电，支持快速初始化外部槽电容
- 如果无需触摸感应，则提供一个通用的比较器和两个 IDAC

17.2 框图

图 17-1 显示的是 CSD 系统框图。

图 17-1. CapSense 模块框图



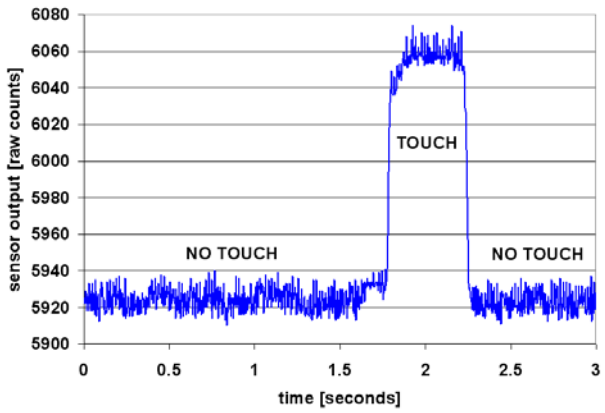
17.3 工作原理

采用 CSD 时，每个 GPIO 均有一个开关电容电路，用于将传感器电容转换为等效电流。然后，模拟复用器会选择其中一个电流信号并将其送到电流 - 数字转换器。电流 - 数字转换器的工作原理与 Delta Sigma ADC 的工作原理相似。

电流 - 数字转换器的输出 （被称为初始计数值）是一个与传感器电容成比例的数字值。

图 17-2 显示的是一段时间内的初始计数值图。当手指触摸传感器时，初始计数值会随着传感器电容值增加而增大。通过将初始计数值与某个预定的阈值进行对比，软件就可以判定手指的存在。

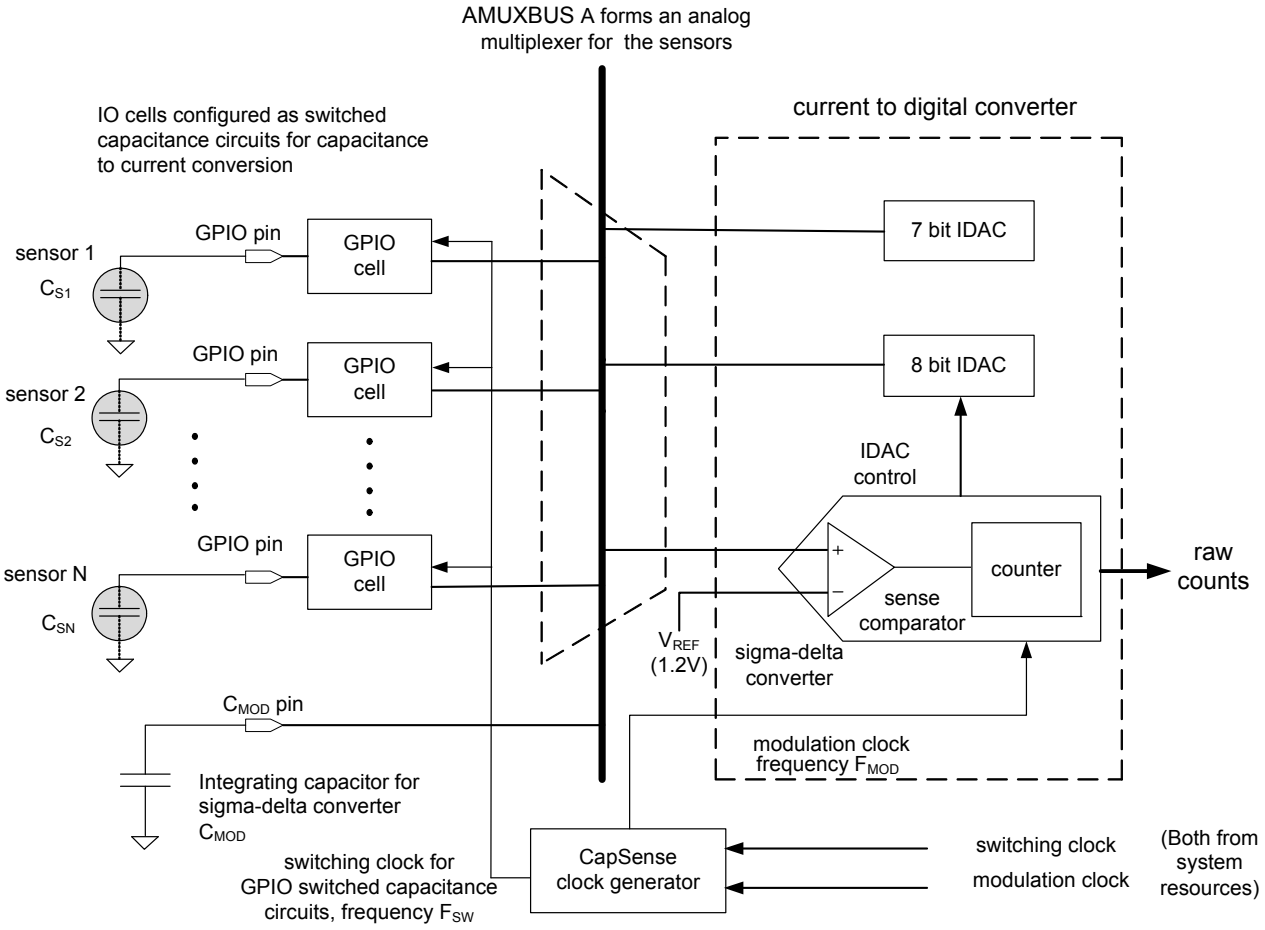
图 17-2. 初始计数值与时间



17.4 CapSense CSD 感应

图 17-3 显示的是 PSoC 4 CapSense 硬件的框图。

图 17-3. PSoC 4 CapSense CSD 感应



17.4.1 GPIO 单元中电容 - 电流转换器

在 CapSense CSD 系统中， GPIO 单元被配置为将传感器电容转换为等效电流的开关电容电路。图 17-4 显示的是 PSoC 4 GPIO 单元架构的简单框图。

PSoC 4 具备两条模拟复用器总线，其中：AMUXBUS A 适用于 CSD 感应； AMUXBUS B 适用于 CSD 屏蔽。 GPIO 开关电容电路有两种可能配置：向 AMUXBUS A 输入的拉电流或从 AMUXBUS A 接受的灌电流。图 17-5 显示的是向 AMUX-BUS A 输入的拉电流的开关电容配置。

图 17-4. PSoC 4 中的 GPIO 单元

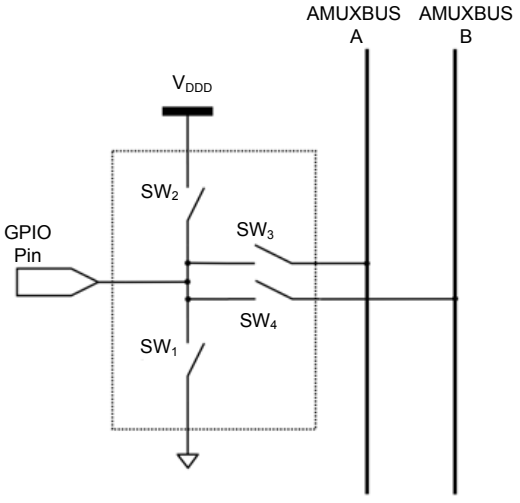
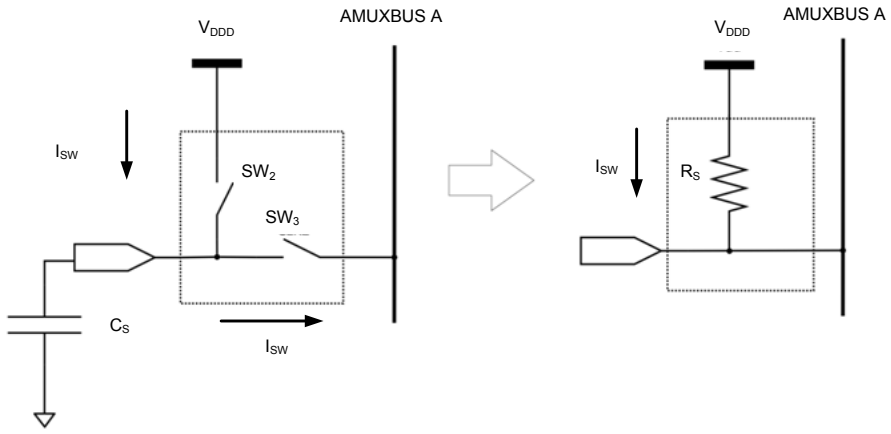


图 17-5. 输入 AMUXBUS A 的拉电流



两个非重叠、非重相位时钟的频率 F_{SW}（请参考图 17-3）控制开关 SW_2 和 SW_3。持续开关的 SW_2 和 SW_3 会构成一个等效的电阻 R_S，如图 17-5 所示。等效的 R_S 电阻值为：

$$R_S = \frac{1}{C_S F_{SW}}$$

公式 17-1

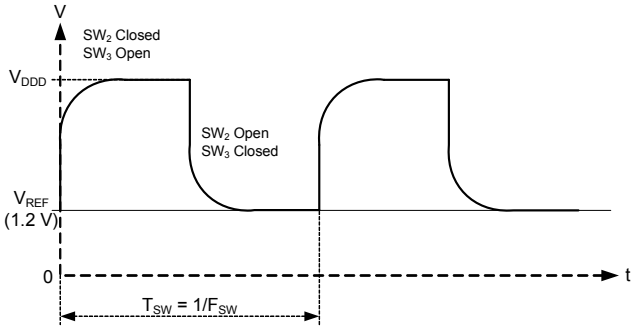
其中：

C_S = 传感器电容

F_{SW} = 开关时钟的频率

Sigma Delta 转换器将 AMUXBUS A 的电压保持为一个常量 V_{REF}（在第 121 页上的 Sigma Delta 转换器 中更详细介绍了该过程）。图 17-6 显示的是传感器电容的电压波形。

图 17-6. 传感器电容上的电压



通过公式 23-3 可计算得出供给 AMUXBUS A 的电流平均值。

$$I_S = C_S F_{SW}(V_{DDD} - V_{REF})$$

公式 17-2

图 17-7 显示了从 AMUXBUS A 输出的灌电流的开关电容配置。则图 17-8 显示的是 C_S 上的电压波形结果。

图 17-7. 来自 AMUXBUS A 的灌电流

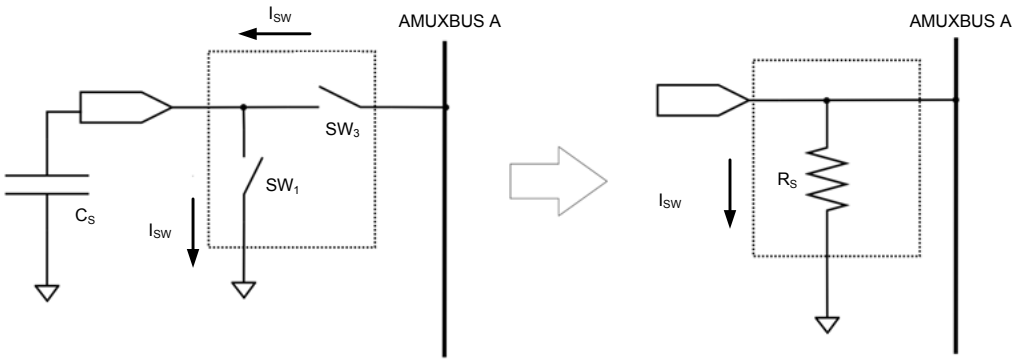
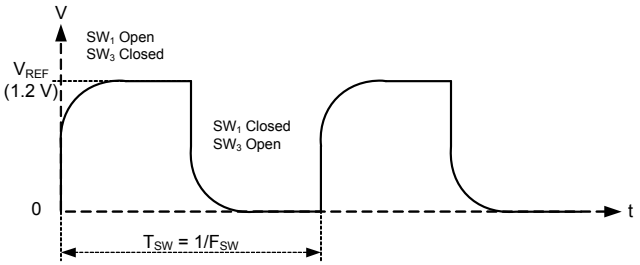


图 17-8. 传感器电容上的电压



通过公式 23-4 可计算出从 AMUXBUS A 输出电流的平均值。

$$I_S = C_S F_{SW} V_{REF}$$

公式 17-3

Sigma Delta 转换器每次扫描一个传感器。AMUXBUS A 用于选择一个 GPIO 单元，并将该单元连接至 Sigma Delta 转换器的引脚，如图 17-3 所示。AMUXBUS A 和 GPIO 单元开关共同组成该模拟复用器（请参见图17-4中的SW3）。AMUXBUS A 可连接支持 CSD 的 16 个 PSoC 4 引脚。欲了解 CSD 引脚的信息，请参考 PSoC 4 数据手册中的内容。

请参见第 45 页上的 I/O 系统章节，了解何为感应、屏蔽和连接 C_{MOD} 配置 GPIO 单元。

17.4.2 CapSense 时钟发生器

该模块和系统资源中的外设时钟生成开关时钟 F_{SW} 和调制时钟 F_{MOD}，如图 17-3 所示。有关详细信息，请参见第 53 页上的时钟系统章节。

GPIO 单元切换电容电路时，需要使用开关时钟。Sigma Delta 转换器使用调制时钟执行时序操作。

使用系统资源中的两个外设时钟（CSDCLK0 和 CSDCLK1）生成所需的频率。有关详细信息，请参见第 53 页上的时钟系统章节。CSDCLK0 生成调制时钟，另外 CSDCLK1 生成开关时钟。

然而，最终的开关时钟频率取决于 CapSense 的时钟发生器。它具有以下输出选项：

- 直接：直接使用可编程时钟分频器的输出。要选择该选项，请设置 CSD_CONFIG 寄存器 1 中的 BYPASS_SEL 位。
- 二分频。对时钟频率进行二分频。要选择该选项，请在 CSD_CONFIG 寄存器中清除 PRS_SELECT 和 BYPASS_SEL 位。
- 伪随机序列 （PRS）：通过在更大的范围内扩展开关频率可降低 CapSense 系统中的 EMI。要选择该选项，请在 CSD_CONFIG 寄存器中设置 PRS_SELECT 位并清除 BYPASS_SEL 位。通过在同一个寄存器中使用 PRS_12_8 位，可以选择 8 位伪随机序列或 12 位伪随机序列。设置 PRS_12_8 位可以选择 12 位序列；清除它以选择 8 位 PRS。

如果已经选择了 PRS，则最大开关频率为

$$F_{SW(maximum)} = \frac{F_{in}}{2}$$

公式 17-4

其中：F_{in} 是 CSDCLK1 的输出频率。最小频率为：

$$F_{SW(minimum)} = \frac{F_{in}}{\text{PRS length-1}}$$

公式 17-5

其中：PRS 的长度为 12 位或 8 位。平均开关频率为：

$$F_{SW(average)} = \frac{F_{in}}{4}$$

公式 17-6

CSD_CONFIG 寄存器中的 PRS_CLEAR 位可用于清除 PRS；设置该位时，它会强制伪随机发生器返回到初始状态。

17.4.3 Sigma Delta 转换器

Sigma Delta 转换器将输入电流转换为一个相应的数字计数。它包含一个比较器、一个参考电压 V_{REF}、一个计数器以及两个拉电流 / 灌电流数模转换器 （IDAC），如图 17-3 所示。

Sigma delta 调制器以打开 / 关闭形式来控制 8 位 IDAC 的电流。IDAC 被称为调制 IDAC。7 位 IDAC（即称为补偿 IDAC）始终为打开或关闭状态。

Sigma delta 转换器可在单 IDAC 模式或双 IDAC 模式下工作。在单 IDAC 模式下，补偿 IDAC 始终为关闭状态。在双 IDAC 模式下，补偿 IDAC 始终为打开状态。

Sigma Delta 转换器还需要一个外部积分电容 C_{MOD}，如图 17-1 所示。C_{MOD} 的推荐值为 2.2 nF。PSoC 4 有一个专用的 C_{MOD} 引脚。欲了解详细信息，请参考 PSoC4 数据手册中的 “ 器件引脚分布 ” 一节中介绍的内容。

Sigma Delta 调制器保持 C_{MOD} 上的电压为 V_{REF}。它在下列某种模式下工作：

- IDAC 拉电流模式：如果开关电容电路从 AMUXBUS A 接收电流，那么， IDAC 将为 AMUXBUS A 提供电流，以使其电压平衡。
- IDAC 灌电流模式：在此模式下， IDAC 从 C_{MOD} 吸收电流，同时开关电容电路会为 C_{MOD} 提供电流。

在这两种情况下，随着 C_{MOD} 上小电压变化，将调制 IDAC 电流切换为打开或关闭状态，以保持 C_{MOD} 的电压等于 V_{REF}。

Sigma delta 转换器的工作范围为 8 位到 16 位分辨率。在单 IDAC 模式下，初始计数值与传感器电容值成正比。如果 ‘N’ 是 Sigma Delta 转换器的分辨率，并且 I_{MOD} 是调制 IDAC 电

流的值，则使用公式 16-7 可计算出 IDAC 拉电流模式中相应的初始计数值。

$$Rawcount = 2^N \frac{V_{REF} F_{SW}}{I_{MOD}} C_S$$

公式 17-7

同样， IDAC 灌模式中初始计数的近似值为：

$$Rawcount = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{MOD}} C_S$$

公式 17-8

在这两种情况下，初始计数值均与传感器电容值 C_S 成正比。固件会处理初始计数值，用以检测触摸。您可以使用双 IDAC 模式下的两种 IDAC，以提高 CapSense 的性能。

在该双IDAC模式下，补偿IDAC始终为打开状态。如果I_{COMP}是补偿 IDAC 电流，则通过下面的公式可计算出 IDAC 拉电流模式中的初始计数值：

$$Rawcount = 2^N \frac{V_{REF} F_{SW}}{I_{MOD}} C_S - 2^N \frac{I_{COMP}}{I_{MOD}}$$

公式 17-9

IDAC 灌电流模式中的初始计数值可通过公式 16-10 得出。

$$Rawcount = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{MOD}} C_S - 2^N \frac{I_{COMP}}{I_{MOD}}$$

公式 17-10

请注意初始计数值始终为正值。

应将硬件参数（如：I_{COMP}、I_{MOD} 和 F_{SW}）调试到最佳状态，用以执行可靠的触摸测试。欲了解调试过程的详细信息，请参考 PSoC 4 CapSense 设计指南的内容。

CSD_CONFIG、CSD_COUNTER 和 CSD_IDAC 寄存器控制 Sigma delta 转换器的操作。 CSD_CONFIG 寄存器中的各重要位包括：

- CSD_CONFIG 中的 ENABLE 位：使能 CSD 模块的主设备。必须将该位设置为 1，使 CSD 操作有效。
- CSD_CONFIG 寄存器中的 POLARITY 位：用于选择 IDAC 灌电流模式还是 IDAC 拉电流模式。 0：IDAC 拉电流模式， 1：IDAC 灌电流模式。
- CSD_CONFIG 寄存器中的 SENSE_COMP_BW 位：用于选择感应比较器的带宽。设置该位可提供高带宽，如果清除它则提供的是低带宽。建议在 CSD 操作中使用高带宽。
- CSD_CONFIG 中的 SENSE_COMP_EN 位：用于启动感应比较器电路。0 表示感应比较器的电源被关闭 。1 表示感应比较器的电源被打开。
- SENSE_EN 位：使能 Sigma delta 调制器输出。另外，还会启动各 IDAC.

必须准确配置 IDAC，以用于 CSD 操作更多有关信息，请参见 PSoC 4 寄存器 TRM 中介绍 CSD_IDAC 寄存器的内容。

CSD_COUNTER 寄存器用于对当前选择传感器进行采样，并读取结果。对比较器进行采样 （以调制时钟频率进行采样）时，如果采样结果为 ‘1’ 时，则该寄存器中的 16 位计数器字段会增加。每次开始进行一个新感应操作时，固件通常会将 ‘0’ 写入到该字段。CSD_COUNTER 寄存器中的 16 位周期字段用于将电容值初始为数值转换。将一个非零值写入到该寄存器中会初始感应操作。通过固件写入到该字段的值用于确定计数器字段对比较器输出进行采样的周期。

开始进行 CSD 操作前，必须正确配置时钟、GPIO、IDAC 和 Sigma delta 调制器等项。每个调制时钟周期过后，周期字段都会减少。当它的值达到 0 时，计数器字段会停止递增。这时，字段值等于同传感器中电容值相应的初始计数值。

17.5 CapSense CSD 屏蔽

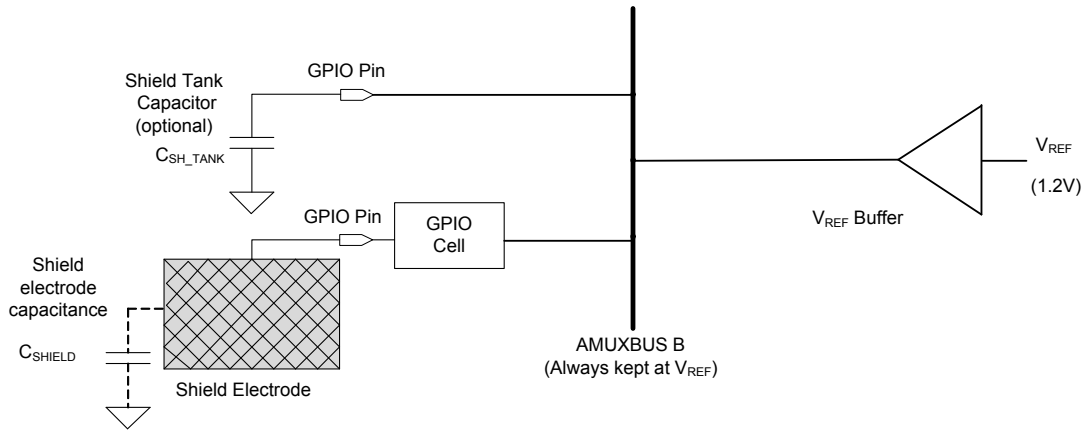
PSoC 4 CapSense 为防水和接近传感性能支持屏蔽电极。对于防水性能，屏蔽电极总是保持为与传感器相同的电位。PSoC 4 CapSense 具有一个屏蔽电路，该电路会使用传感器开关信号的副本来驱动屏蔽电极（请参考第 119 页上的 GPIO 单元中电容 - 电流转换器），这样可以避免传感器与屏蔽电极间潜在的差异。欲了解屏蔽的基本信息，请参考 PSoC 4 CapSense 设计指南一节的内容。

在感应电路中， Sigma Delta 转换器保持 AMUXBUS A 的电压等于 V_{REF}（请参考第 121 页上的 Sigma Delta 转换器）。通过切换 AMUXBUS A 与电源轨（是 V_{DD} 或接地，取决于其配置）之间的传感器，GPIO 单元可生成传感器波形。屏蔽电路也使用类似的方法来工作； AMUXBUS B 电压始终保持为 V_{REF}。GPIO 单元会切换 AMUXBUS B 与电源轨（等于 V_{DD} 或接地，该配置与传感器配置相同）之间的屏蔽。该过程会生成屏蔽电极上的传感器开关波形的副本。

根据保持 AMUXBUS B 为 V_{REF} 的方法不同，可能有以下两种不同的配置。

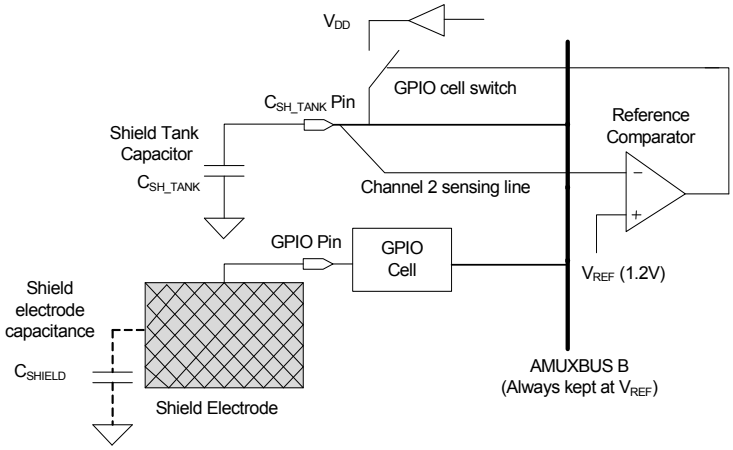
- 使用 V_{REF} 缓冲区驱动屏蔽：在此配置中，会使用一个电压缓冲区将 AMUXBUS B 驱动为 V_{REF}，如图 17-9 所示。推荐使用外部电容 C_{SH_TANK} 来降低开关跃变电压。CSD_CONFIG 寄存器中 REBUF_OUTSEL 位的设置会将缓冲区输出连接至 AMUXBUS B。同一寄存器中的 REFBUF_DRV 位字段可用来设置缓冲区的驱动能力。将 0 写入到该字段可禁用该缓冲区；将 1、 2 和 3 写入到该字段分别可以选择低电流、中电流和高电流驱动模式。

图 17-9. 使用 V_{REF} 缓冲区驱动屏蔽



- 使用 GPIO 单元预充电驱动屏蔽：该配置用到一个外部电容 C_{SH_TANK}，如图 17-10 所示。特殊的 GPIO 单元和一个参考比较器用来给电容 C_{SH_TANK} 充电，因此 AMUXBUS B 会保持其电压为 V_{REF}。参考比较器始终监控 CSH_TANK 电容的电压，并控制 GPIO 单元开关，以维持其电压为 V_{REF}。使用专用的感应线路 （称为通道 2 感应线路），将参考比较器连接至 C_{SH_TANK} 电容，如图 17-10 所示。

图 17-10. 使用 GPIO 预充电驱动屏蔽



GPIO 单元预充电功能仅在固定引脚 C_{SH_TANK} 上有效。欲了解更详细的信息，请参考 PSoC 4 数据手册中的器件引脚分布一节的内容。

CSD_CONFIG 寄存器中的 COMP_MODE 位选择参考缓冲区预充电或 GPIO 预充电； 0：表示参考缓冲区预充电； 1：表示 GPIO 预充电。

17.5.1 C_{MOD} 预充电

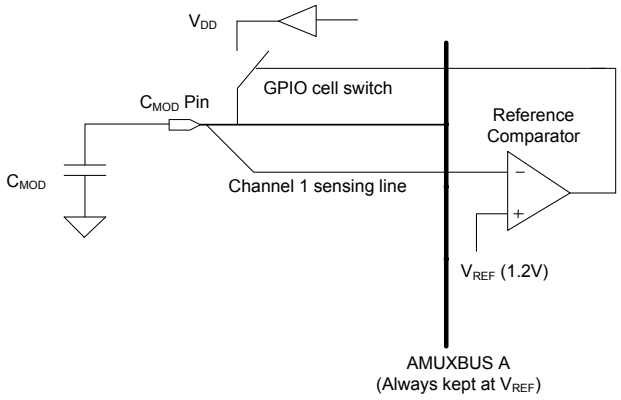
第一次使能 CapSense 硬件时， C_{MOD} 上的电压为零。然后 Sigma Delta 转换器会缓慢为 C_{MOD} 充电，使之电压达到 V_{REF}。充电电流来自各 IDAC （在 IDAC 拉电流模式中）或传感器开关电容电路（在 IDAC 灌电流模式中）。然而，由于 C_{MOD} 是一个较大的电容，因此，充电过程相当缓慢。

C_{MOD} 预充电是快速初始化 C_{MOD} 上的电压使之达到 V_{REF} 的过程。使用预充电可缩短 Sigma Delta 转换器开始工作所需要的时间。 C_{MOD} 预充电有两种选项。

- 使用 V_{REF} 缓冲区进行预充电：使能屏蔽时，V_{REF} 缓冲区输出始终连接至 AMUXBUS B （图 17-9）。为了能使用 V_{REF} 缓冲区进行预充电，应先将 C_{MOD} 连接至 AMUXBUS B。预充电结束后，将 C_{MOD} 连接至 AMUXBUS A，使 Sigma Delta 能够可正常操作。屏蔽被禁用时，V_{REF} 缓冲区输出始终连接至 AMUXBUS A 维持预充电过程，然后断开连接。
- 使用 GPIO 单元进行预充电：在此配置中，通过一个特殊的 GPIO 单元和一个参考比较器用于为电容 C_{MOD} 充电，使之电压达到 V_{REF}。仅在固定 C_{MOD} 引脚上， GPIO 单元预充电功能才有效。欲了解更详细的信息，请参考 PSoC 4 数据手册中的器件引脚分布一节中的内容。该目

的使用的比较器与预充电 CSH_TANK 的参考比较器相同。通过 CSD_CONFIG 寄存器中的 COMP_PIN 位选择连接至参考比较器的电容器。如果该位的值为 0，被指定为 “ 通道 1” 的感应线路用于将 C_{MOD} 连接至参考比较器，如图 17-11 所示；如果该位的值为 1，则通道 2 感应线路用来将 CSH_TANK 连接到参考比较器，如图 17-10 所示。请注意：必须正确配置 GPIO 单元，这样 GPIO 单元预充电才能进行。

图 17-11. GPIO 单元预充电



使用 GPIO 单元进行预充电的速度比使用 V_{REF} 缓冲区的速度快。因此，推荐使用 GPIO 进行预充电。然而，如果您对初始化 CapSense 的时效性要求不高，请使用 V_{REF} 缓冲区进行预充电。

通道 1 感应线路也用来将 C_{MOD} 连接至 Sigma delta 调制器中的感应比较器。将 CSD_CONFIG 寄存器中的 SENSE_INSEL 位设置为 1 可使能该选项。清除该位可使 C_{MOD} 通过 AMUXBUS A 连接至感应比较器。

17.6 通用资源 — IDAC 和比较器

如果触摸感应时没有使用 CapSense 模块，则感应比较器和两个 IDAC 可作为通用模拟模块使用。

您可以使用 AMUXBUS A 将任何支持 CSD 的 GPIO 连接至感应比较器的非反相输入端。反相输入被连接到 1.2 V V_{REF}（请参见图 17-3）。AMUXBUS A 还可以作为比较器输入的模拟复用器使用。可以使用 CSD_CONFIG 寄存器中的 SENSE_COMP_EN、SENSE_COMP_BW 和 ENABLE 位控制感应比较器，如第 121 页上的 Sigma Delta 转换器 中的介绍。

如果在其他使用中用到 AMUXBUS，则 CSD_CONFIG 寄存器中的 SENSE_INSEL 位可用于将感应比较器的非反相输入连接至固定的 C_{MOD} 引脚，如第 123 页上的 CMOD 预充电 中的解释。比较器的输出可以连接到多个 GPIO，请参见第 45 页上的 I/O 系统章节了解详细信息。

8 位 IDAC 的工作电流范围为 0~306 µA（1.2 µA/ 位）或 0~612 µA（2.4 µA/ 位）。7 位 IDAC 的工作电流范围为 0~152.4 µA（1.2 µA/ 位）或 0~304.8 µA （2.4 µA/ 位）。

8 位和 7 位 IDAC 都可以使用 AMUXBUS A 和 AMUXBUS B 连接到 GPIO。另外，还可以将这两个 IDAC 连接到单个 AMUXBUS。IDAC 可以在下面三个不同的模式下工作：CSD 模式、通用 （GP）模式和 CSD 与通用模式。表 17-1 说明的是在每个模式中 IDAC1 和 IDAC2 被连接到 AMUXBUS A 和 AMUXBUS B 的方式。

表 17-1. IDAC 模式

模式	AMUXBUS A	AMUXBUS B
CSD 模式	IDAC 灌 / 拉电流电压为 1.2 V	没有连接 IDAC
通用模式	8 位 IDAC 灌 / 拉电流	7 位 IDAC 灌 / 拉电流
CSD 和通用 （GP）模式	8 位 IDAC 灌 / 拉电流电压为 1.2 V	7 位 IDAC 灌 / 拉电流

有关更多信息，请参见 PSoC 4 寄存器 TRM 中介绍 CSD_IDAC 寄存器的内容。通过 CSD_CONFIG 寄存器可以使能 IDAC，并设置极性，如第 121 页上的 Sigma Delta 转换器 所介绍的。有关如何将 GPIO 连接至 AMUXBUS A 和 B 的详细信息，请参考第 45 页上的 I/O 系统章节。

17.7 寄存器列表

表 17-2. CapSense 寄存器列表

寄存器名称	说明
CSD_CONFIG	该寄存器用于配置和控制 CSD 模块及其资源。
CSD_IDAC	该寄存器用于控制 IDAC 电流的设置。
CSD_COUNTER	该寄存器用来初始化对选择电容传感器进行的采样，并读取转换器的结果。
CSD_STATUS	该寄存器允许观察 CSD 模块中的关键信号。
CSD_INTR	它是 CSD 中断请求寄存器。

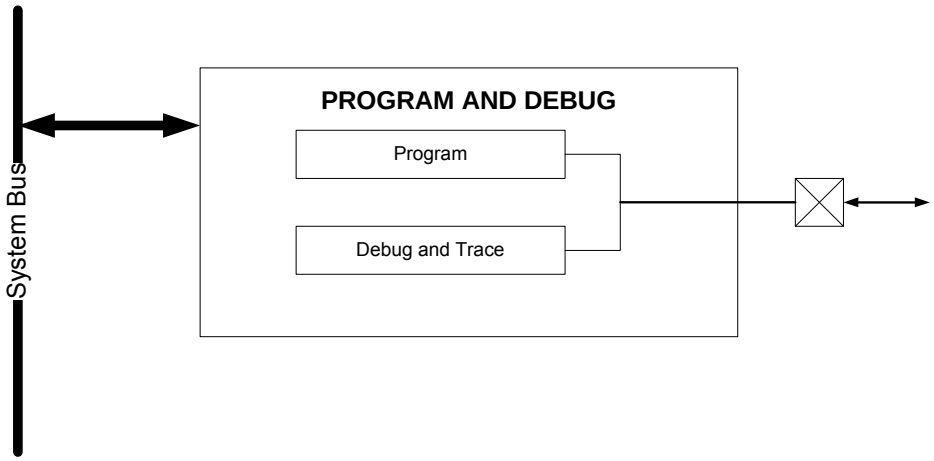
Section G: 编程和调试

本章节包括以下小节：

- 第 129 页上的编程与调试接口章节
- 第 135 页上的非易失性存储器编程章节

顶层架构

编程和调试框图



18. 编程与调试接口

PSoC® 4 编程与调试接口为外部器件提供了一个用于进行编程或调试的通信网关。外部器件可以是赛普拉斯供应的编程器和调试器，也可以是支持 PSoC 4 编程和调试功能的第三方器件。 PSoC4 的编程 / 调试采用标准的串行线调试 （SWD）接口。

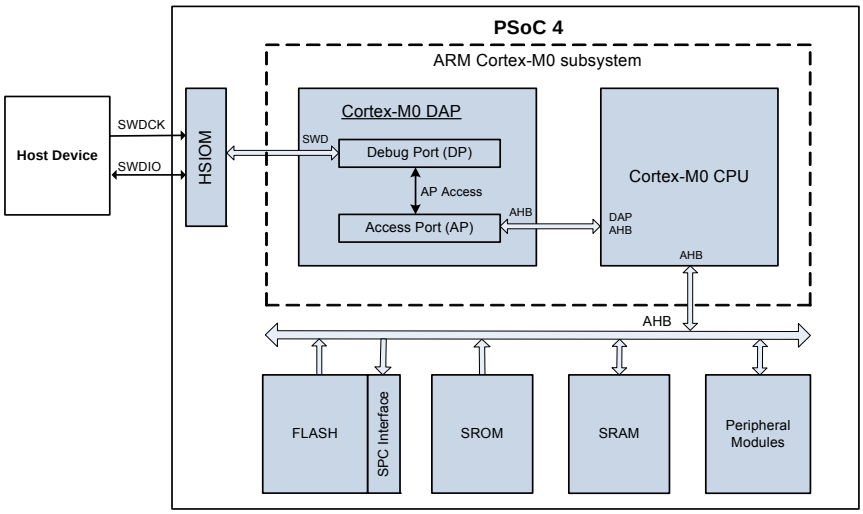
18.1 特性

- 通过 SWD 接口编程和调试
- 调试过程中能使用四个硬件断点和两个硬件观察点
- 调试过程中能对系统中所有的存储器和寄存器 （包括 Cortex-M0 寄存器组，内核运行或停止情况下）进行读和写访问

18.2 功能说明

图 18-1 显示的是 PSoC 4 中编程和调试接口的框图。 Cortex-M0 调试和访问端口 （DAP）作为编程和调试接口使用。外部编程器或调试器，亦即 “ 主机 ”，通过使用 SWD 接口的两个引脚（双向数据引脚（SWDIO）和主机驱动的时钟引脚（SWDCK））与 PSoC 4 的 DAP “ 目标 ” 通信。 SWD 物理端口引脚 （SWDIO 和 SWDCK）通过高速 I/O 矩阵 （HSIOM）与 DAP 通信。有关 HSIOM 的详细信息，请参考第 45 页上的 I/O 系统章节。

图 18-1. PSoC 4 编程和调试接口



DAP 使用 ARM 指定的高级和高性能总线 （AHB）接口与 Cortex-M0 CPU 通信。 AHB 是 PSoC 4 的内部系统互连协议，用于 AHB 主设备进行的存储器和外设寄存器访问。PSoC 4 有两个 AHB 主控 — ARM CM0 CPU 内核和 DAP。外部器件可通过 DAP 有效地控制整个器件，以实现编程和调试操作。

18.3 串行线调试 （SWD）接口

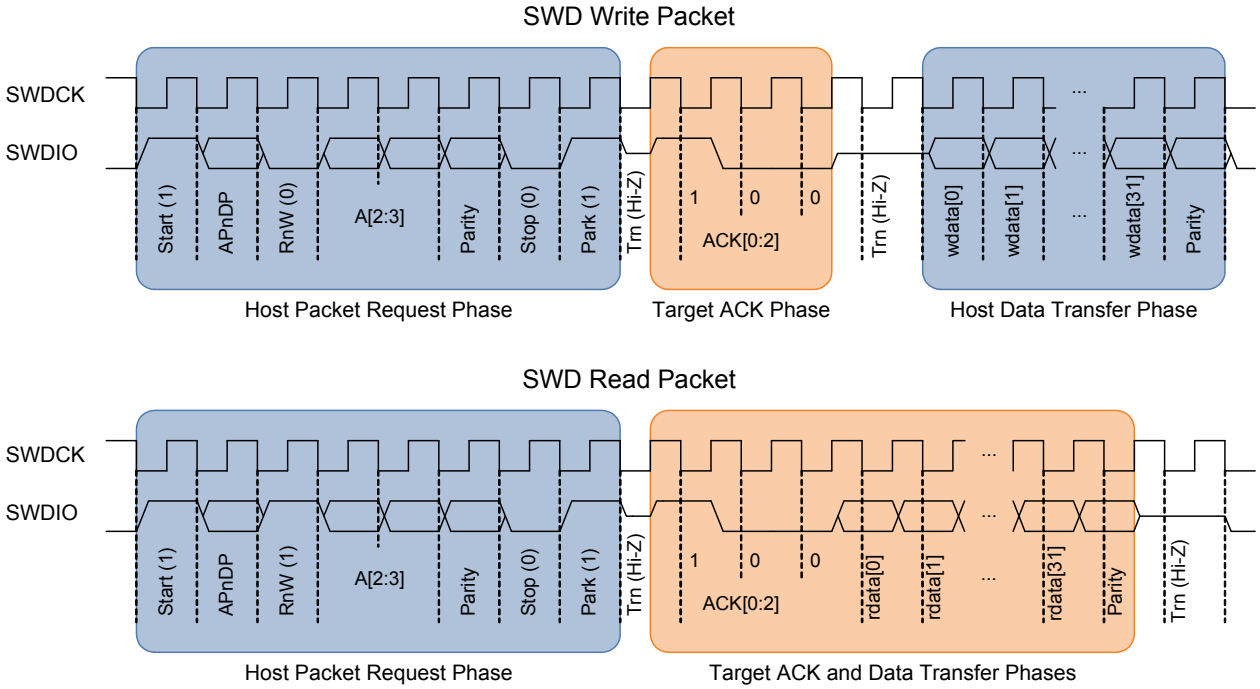
PSoC 4 的 Cortex-M0 支持通过 SWD 接口进行的编程和调试。SWD 协议是基于数据包的串行数据操作协议。在引脚等级上，它使用一个单一双向数据信号 （SWDIO）和一个单向时钟信号 （SWDCK）。主机编程器始终驱动时钟线，同时，主机或目标将驱动数据线。完整的数据传输 （一个 SWD 数据包）需要 46 个时钟周期，并包括三个阶段：

- **主机数据包请求 阶段 —** 主机向 PSoC 4 目标发送一个请求。
- **目标确认响应阶段 —** PSoC 4 目标给主机发送一个确认。
- **数据传输阶段 —** 根据传输方向，主机或目标将数据写入总线上。

当 SWDIO 线的控制权从主机转交给目标 （或反过来转交）时，在一个反转周期 （Trn）内，没有任何器件驱动该线，并且该线会浮动于高阻抗 （Hi- Z）状态。根据转交情况，反转周期可能等于 0.5 或 1.5 个时钟周期。

图 18-2 显示了读取和写入 SWD 数据包的时序框图。

图 18-2. PSoC 4 SWD 写入和读取数据包的时序框图



下面介绍的是 SWD 读取和写入数据包的传输情况：

1. 主机数据包请求阶段：SWDIO 由主机驱动
   a. 起始位启动传输；其状态始终为逻辑 1。
   b. “AP not DP” （APnDP）位确定传输类型是 AP 访问 — 1b1 还是 DP 访问 — 1b0。
   c. “Read not Write” 位 （RnW）控制着数据传输的方向。1b1 表示从目标 ‘ 读取 ’ ，1b0 表示 ‘ 写入 ’ 目标。
   d. 地址位 (A[3:2]) 是 AP 或 DP （取决于 APnDP 位值）的寄存器选择位。请参见表 18-3 和表 18-4，了解相关的定义。**注意** 地址位的最低有效位 （LSB）先被传送。
   e. 奇偶位包含 APnDP、 RnW 和 ADDR 位的奇偶校验。它是一个偶校验位，这意味着，当将该位与其他位进行 XOR 运算时，得到的结果将为 0。
      如果奇偶位不正确， PSoC 4 将忽略包头；这样，将不会有 ACK 响应 （ACK = 3b111）。应该中止该编程操作，并通过下面的器件复位过程再尝试进行编程。
   f. 停止位始终为逻辑 0。
   g. 停留位始终为逻辑 1。
2. 目标确认响应阶段：SWDIO 由目标驱动
   a. ACK[2:0] 位表示从目标到主机的响应，并指出响应结果是成功还是失败。请参见表 18-1，了解相关的定义。**注意 :** ACK 位的最低有效位 （LSB）先被传送。
3. 数据传输阶段：SWDIO 由目标或主机驱动 （取决于传输方向）
   a. 将需要读取或写入的数据写入到总线内 （先写入最低有效位 （LSB））。
   b. 数据奇偶位指示需要读取或写入的数据的奇偶情况。它是一个偶校验位，这意味着，当将该位与数据位进行 ” XOR” 运算时，得到的结果将为 0。

如果奇偶位指出一个数据错误，那么，需要进行纠正操作。对于一个读取数据包，如果主机检测到一个奇偶错误，它必须中止编程过程，然后重新启动该操作。对于写入数据包，如果目标检测到奇偶错误，它将在下一个数据包中生成 FAULT ACK 响应。

根据 SWD 协议，在 SWDIO 的状态为低电平的同时，主机可以在某两个数据包间生成不限定的 SWDCK 时钟周期数。如果时钟为非自由运行的，或要求时钟在空闲模式中自由运行，则建议在某两个SWD数据包间生成三个或更多的虚拟时钟周期。

在 SWDIO 的状态为高电平的同时，在 50 个或更多时钟周期内计时 SWDCK 线，可以复位 SWD 接口。如要返回闲置状态，只要在 SWDIO 为低电平状态的同时以一个周期计时 SWDCK 线即可。

18.3.1 SWD 时序的详细信息

根据通信的方向，SWDIO 线的写入和读取时间会有所不同。主机在主机数据包请求相位内驱动SWDIO线。如果主机向目标写入数据，它也会在数据传输相位内驱动 SWDIO 线。当主机驱动 SWDIO 线时，它会在 SWDCK 的下降沿上写入新的位，同时，目标会在 SWDCK 的上升沿上读取该位。在目标确认响应相位期间目标会驱动 SWDIO 线。如果目标读取数据，则它也会在数据传输相位期间驱动 SWDIO 线。当目标驱动 SWDIO 线时，它会在 SWDCK 的上升沿上写入新的位，同时，主机会在 SWDCK 的下降沿上读取该位。

表18-1和图18-2介绍了SWDIO位的写入和读取的时序情况。

表 18-1. SWDIO 位写入和读取时序

SWD 数据包相位	SWDIO 边沿	
	下降	上升
主机数据包请求	主机写入	目标读取
主机数据传输		
目标确认响应	主机读取	目标写入
目标数据传输		

18.3.2 ACK 的详细信息

确认 （ACK）位字段用于通信前一传输的状态。 OK ACK 表示前一数据包传输成功。一个 WAIT 响应要求一个数据阶段。FAULT 的状态表示需要立即中止编程过程。表 18-2 显示的是 ACK 位字段解码的详细信息。

表 18-2. SWD 传输 ACK 响应解码

响应	ACK[2:0]
OK （确定）	3b001
WAIT （等待）	3b010
FAULT （故障）	3b100
NO ACK （无确认）	3b111

下面介绍的是 WAIT 和 FAULT 响应行为的详情：

- 对于 WAIT 响应，如果数据操作为读取，主机应在数据阶段内忽略数据的读取。目标将不驱动该线，并且主机不能检查奇偶位。
- 对于 WAIT 响应，如果数据操作为写入，PSoC 4 会忽略数据相位。然而，主机仍要发送所需写入的数据，以完成数据包。同时，主机还要发送与该数据相应的奇偶位。
- WAIT响应表示PSoC 4正在处理前一数据操作。主机最多可以尝试四个连续的 WAIT 响应，以确定是否接收到 OK 响应。如果失败，那么需要中止编程操作，并重新启动该操作。
- 对于 FAULT 响应，也需要中止编程操作，然后通过器件复位重新启动该操作。

18.3.3 反转 （Trn）周期的详细信息

数据包请求阶段和 ACK 阶段间有一个反转周期； ACK 阶段和数据阶段 （主机写入传输情况下）间也有一个反转周期，如图 18-2 所示。根据 SWD 协议，主机和目标都使用 Trn 周期来更改其相应 SWDIO 线的驱动模式。在数据包请求阶段结束后的第一个 Trn 周期内，目标在 SWDCK 上升沿到来时开始驱动 SWDIO 线上的 ACK 数据。这样确保主机能在下一个下降沿上读取 ACK 数据。因此，第一个 Trn 周期的长度仅为半个时钟周期。SWD 数据包的第二个 Trn 周期等于一个半时钟周期。在 Trn 周期内，主机和 PSoC 4 都不能驱动 SWDIO 线。

18.4 Cortex-M0 调试和访问端口 （DAP）

Cortex-M0 编程和调试接口包括一个调试端口 (DP) 和一个访问端口 (AP)。这两个端口组合起来，形成了 DAP 端口。调试端口将执行用于使能与主机通信的 SWD 接口协议的状态机。它还包含用于配置访问端口、DAP 标识代码，等等的寄存器。访问端口包含允许外部器件访问Cortex-M0 DAP-AHB接口的寄存器。通常， DP 寄存器用于一次性配置或错误检测， AP 寄存器用于执行编程和调试操作。有关 DAP 的完整架构的详细信息，请参阅 *ARM® 调试接口 v5 架构规格*。

18.4.1 调试端口 （DP）寄存器

表 18-3 显示的是用于编程和调试的 Cortex-M0 DP 寄存器及其相应的 SWD 地址位选择。对于 DP 寄存器访问， APnDP 位始终为零。两个地址位 (A[3:2]) 用于选择不同的 DP 寄存器。请注意，根据访问操作是读取操作还是写入操作，可以通过同样的地址位访问不同的 DP 寄存器。更多有关所有 DP 寄存器的详细信息，请参阅 *ARM® 调试接口 v5 架构规格*。

表 18-3. 主要调试端口 （DP）寄存器

寄存器	APnDP	地址 A[3:2]	RnW	全名	寄存器功能
ABORT	0 (DP)	2b00	0 (W)	AP 中止寄存器	该寄存器用于强制中止一个 DAP，以及取消错误和粘滞标志条件。
IDCODE	0 (DP)	2b00	1 (R)	标识代码寄存器	该寄存器保存 Cortex-M0 CPU 的 SWD ID （0x0BB11477）。
CTRL/STAT	0 (DP)	2b01	X (R/W)	控制和状态寄存器	该寄存器用于控制 DP，并包含有关 DP 的状态信息。
SELECT	0 (DP)	2b10	0 （W）	AP 选择寄存器	该寄存器用于选择当前的 AP。在 PSoC 4 中，只有一个与 DAP AHB 相连的 AP。
RDBUFF	0 (DP)	2b11	1 (R)	读缓冲寄存器	该寄存器保存最后一次 AP 读取操作的结果。

18.4.2 访问端口 （AP）寄存器

表 18-4 介绍了用于编程和调试的主要 Cortex-M0 AP 寄存器及其相应的 SWD 地址位选择。对于 AP 寄存器访问，APnDP 位始终为零。两个地址位 (A[3:2]) 用于选择不同的 AP 寄存器。

表 18-4. 主要访问端口 （AP）寄存器

寄存器	APnDP	地址 A[3:2]	RnW	全名	寄存器功能
CSW	1 (AP)	2b00	X (R/W)	控制和状态字寄存器 (CSW)	通过该寄存器，可以配置并控制通过存储器访问端口对已连接的存储器系统 （即为 PSoC 4 存储器映像）进行的访问。
TAR	1 (AP)	2b01	X (R/W)	传输地址寄存器	该寄存器用于指定需要读取或写入的 32 位存储器地址
DRW	1 (AP)	2b11	X (R/W)	数据读 / 写寄存器	该寄存器保存需要读取或写入 TAR 寄存器所指定的地址的 32 位数据

18.5 编程 PSoC 4 器件

按照下面的顺序，可以对 PSoC 4 进行编程。请参阅 PSoC 4 器件编程规范，了解编程时所需要的编程算法、时序规范、硬件配置的详细信息。

1. 获取 PSoC 4 中的 SWD 端口。
2. 进入编程模式。
3. 执行器件编程子程序，如芯片 ID 检查、闪存编程、闪存验证以及校验和验证。

18.5.1 获取 SWD 端口

18.5.1.1 主要和辅助 SWD 引脚对

编程器件的第一步是获取 PSoC 4 中的 SWD 端口。有关 SWD 引脚的信息，请参考 PSoC 4 数据手册的内容。

如果在器件中的两对 SWD 引脚均可用，则通过监控闪存区中的 SWD_CONFIG 寄存器，可以选择两对 SWD 引脚中的一对用于编程和调试。请注意，每次编程或调试会话期间，只能使用一对 SWD 引脚。器件的默认工厂设置是主要的 SWD 引脚对。如要选用辅助 SWD 引脚对，则需要编程器件，使之同时使用主要引脚对和配置用于使能辅助引脚对的十六进制文件。这样，就能使用辅助 SWD 引脚对。

18.5.1.2 SWD 端口获取序列

编程器件的第一步是为主机获取目标的 SWD 端口。主机先通过激活外部复位 （XRES）引脚执行器件复位。移除 XRES 信号后，主机必须在获取窗口中向器件发送一个连接序列，以连接至 DAP 中的 SWD 接口。下面提供了该序列的伪代码。

代码 1. SWD 端口获取伪代码

```
ToggleXRES(); // Toggle XRES pin to reset device

//Execute ARM’s connection sequence to acquire SWD-port
do
{
        SWD_LineReset(); //perform a line reset (50+ SWDCK clocks with SWDIO high)
        ack = Read_DAP ( IDCODE, out ID); //Read the IDCODE DP register

}while ((ack != OK) && time_elapsed < 1.5 ms); // retry connection until OK ACK or timeout

if (time_elapsed >= 1.5 ms) return FAIL; //check for acquire time out

if (ID != CM0_ID) return FAIL; //confirm SWD ID
```

of Cortex-M0 CPU. (0x0BB11477)

在伪代码中， SWD_LineReset() 是用于复位调试访问端口的标准 ARM 指令。它使用超过 49 SWDCK 个时钟周期，并且还需要 SWDIO 处于高电平状态。至少在发送了一个 SWDCK 时钟周期，且 SWDIO 被置位为低电平时，必须完成数据操作。该序列将对编程器和芯片进行同步化。Read_DAP() 指的是对调试端口中的 IDCODE 寄存器进行的读取操作。需要重复线复位序列和 IDCODE 的读取操作，直至接收到 IDCODE 读取操作的 OK ACK 或发生超时 (1.5 ms) 为止。如果在时间窗口中接收到 OK ACK，并且 IDCODE 的读取操作与 Cortex-M0 DAP 的相应读取操作相匹配，这样才能获取 SWD 端口。

18.5.2 SWD 编程模式入口

获取 SWD 端口后，主机必须在特定的时间窗口中进入器件编程模式。该操作可通过置位测试模式控制寄存器 （MODE 寄存器）中的 TEST_MODE 位 （位 31）实现的。进入器件编程模式前，还要配置调试端口。有关进入编程模式的时序规范和伪代码的信息，请参考 PSoC 4 器件编程规范文档。

18.5.3 SWD 编程子程序执行

当器件进入编程模式时，外部编程器可启动发送 SWD 数据包序列，以执行编程操作，如擦除闪存、编程闪存、校验和验证，等等。第 135 页上的非易失性存储器编程章节中介绍了编程子程序。有关调用编程子程序的正确序列的信息，请参考 PSoC 4 器件编程规范文档。

18.6 PSoC 4 SWD 调试接口

Cortex-M0 DAP 有两类调试功能：侵入性调试和非侵入性调试。侵入性调试包括中止编程和逐步编程，断点以及数据观察点。非侵入性调试包括指令地址分配置和器件存储器访问。这些存储器包含闪存存储器、 SRAM 和其他外设寄存器。

DAP 有三个主要的调试子系统：

- 调试控制和配置寄存器
- 断点单元 （BPU） — 提供断点支持
- 调试观察点（DWT） — 提供观察点支持 Cortex-M0 调试中不支持跟踪功能。

请参阅 *ARMv6-M 架构参考手册*，了解有关调试架构的详细信息。

18.6.1 调试控制和配置寄存器

调试控制和配置寄存器用于执行固件的调试。下面介绍的是寄存器及其主要功能。请参阅 *ARMv6-M 架构参考手册* ，了解有关这些寄存器的完整位级定义。

- 调试中止控制和状态寄存器 （CM0_DHCSR） — 该寄存器包含用于使能调试、中止 CPU、单步调试等操作的控制位。它还有处理器调试状态的状态位。
- 调试故障状态寄存器 （CM0_DFSR ） — 该寄存器用于说明引起调试事件的来源。它包括由 CPU 中止、断点事件或观察点事件引起的调试事件。
- 调试内核寄存器选择器寄存器 （CM0_DCRSR） — 通过寄存器，可以在 Cortex-M0 CPU 中选择必须由外部调试器读取或写入的通用寄存器。
- 调试内核寄存器数据寄存器 （CM0_DCRDR） — 该寄存器用于存储对寄存器 （由 DCRSR 寄存器选择）进行读取 / 写入操作的数据。
- 调试异常和监控控制寄存器 （CM0_DEMCR) — 该寄存器包含各个使能位，分别用于全局调试观察点 （DWT） 模块使能、复位向量捕捉和硬故障异常捕捉。

18.6.2 断点单元 （BPU）

BPU 提供了提取指令时断点功能。 PSoC 4 中的 Cortex-M0 DAP 支持多达四个硬件断点。除了硬件断点外，通过使用 Cortex-M0 中的 BKPT 指令，可以创建任意软件断点的数量。BPU 有两类寄存器。

- 断点控制寄存器 CM0_BP_CTRL 用于使能 BPU 和存储受调试系统支持的硬件断点数量 （对于 PSoC 4 中的 CM0 DAP，该数量为四）。
- 每个硬件断点有一个断点比较寄存器 （CM0_BP_COMPx）。它包含断点的使能位，比较地址值以及触发断点调试事件的匹配条件。该寄存器的典型使用情况是：当某个指令加载地址与断点的比较地址相互匹配时，将生成一个断点，并中止处理器。

18.6.3 数据观察点 （DWT）

DWT 为数据地址访问或编程计数器 (PC) 指令地址提供观察点支持。 PSoC 4 中的 Cortex-M0 不支持跟踪功能。 DWT 支持两个观察点。它还通过 PC 样本寄存器提供外部编程计数器采样。该样本寄存器用于编程计数器的非侵入粗调配置。下面将介绍 DWT 中的最重要寄存器。

- 观察点比较 （CM0_DWT_COMPx）寄存器存储观察点比较器所使用的比较值，以生成观察点事件。每个观察点均有与其相应的 DWT_COMPx 寄存器。
- 观察点掩码 （CM0_DWT_MASKx）寄存器存储适用于相关观察点中地址范围匹配的忽略掩码。
- 观察点功能 （CM0_DWT_FUNCTIONx）寄存器存储触发观察点事件的条件。这些事件可以是编程计数器观察点事件，也可以是数据地址读取 / 写入访问观察点事件。当相应的观察点事件发生时，状态位将被置位。
- 观察点比较器 PC 样本寄存器 （CM0_DWT_PCSR）存储编程计数器的当前值。该寄存器用于编程计数器寄存器的粗调、非侵入配置。

18.6.4 调试 PSoC 4 器件

通过访问调试控制和配置寄存器、BPU 中的寄存器以及 DWT 中的寄存器，主机可以调试 PSoC 4 目标器件。通过 SWD 接口对所有的寄存器进行访问； Cortex-M0 DAP 中的 SWD 调试端口 （SW-DP）通过 DAP-AHB 接口将 SWD 数据包转换为适当的寄存器访问。

调试 PSoC 4 目标器件的第一步是获取 SWD 端口。获取序列

包括 SWD 线复位序列和通过 SWD 接口进行的 DAP SWDID 读取操作。从目标器件读取到正确的 CM0 DAP SWDID 后，可以获取 SWD 端口。如果要使调试数据操作发生在 SWD 接口上，请勿将相应的引脚用于其他任何目的。请参考第 45 页上的 I/O 系统章节，了解如何配置 SWD 端口引脚上的各位，从而决定将这些引脚仅使用于 SWD 接口，还是允许将它们使用于其他功能，如 GPIO。如需要调试，请勿将 SWD 端口引脚使用于其他目的。如果只需要编程，则可以将 SWD 引脚使用于其他目的。

获取了 SWD 端口后，外部调试器将通过置位 DHCSR 寄存器中的 C_DEBUGEN 位来使能调试操作。然后，通过对调试系统中的相应寄存器进行写操作，可以执行不同的调试操作，如逐步调试、中止、断点配置以及观察点配置。

对目标器件的调试过程还受整个器件的保护设置的影响，如第 75 页上的器件安全章节中所述。因此，只有 OPEN（打开）保护模式支持器件调试。当器件从活动模式转换为睡眠或深度睡眠模式时，外部调试器与目标器件间的连接不被断连。当器件从深度睡眠模式或睡眠模式恢复到活动模式时，调试器无需再次启动连接序列，就能恢复其操作。

18.7 寄存器

表 18-5. 寄存器列表

寄存器名称	说明
CM0_DHCSR	调试停止控制和状态寄存器
CM0_DFSR	调试故障状态寄存器
CM0_DCRSR	调试内核寄存器选择器寄存器
CM0_DCRDR	调试内核寄存器数据寄存器
CM0_DEMCR	调试异常和监控控制寄存器
CM0_BP_CTRL	断点控制寄存器
CM0_BP_COMPx	断点比较寄存器
CM0_DWT_COMPx	观察点比较寄存器
CM0_DWT_MASKx	观察点掩码寄存器
CM0_DWT_FUNCTIONx	观察点功能寄存器
CM0_DWT_PCSR	观察点比较器 PC 样本寄存器

19. 非易失性存储器编程

非易失性存储器编程指的是对 PSoC® 4 器件中的闪存存储器进行编程。本章节介绍了器件编程的部分功能，如擦除、写入、编程和校验等。用户可借助 Cypress 公司提供的编程器或其他第三方编程器，使用这些功能将十六进制的应用程序写入 PSoC 4 器件中。此外，在执行加载引导程序 （bootloader）时，用户也可使用这些功能来更新部分闪存。

19.1 特性

- 支持通过调试和访问端口 （DAP）及 Cortex-M0 CPU 进行编程
- 支持从 Cortex-M0 CPU 进行的阻塞和非阻塞的闪存编程和擦除操作。

19.2 功能说明

编程相关的所有操作均是通过调用 “System Call” 函数来实现的；存储在 SROM 中的 System Call 函数仅可在特权模式下才可被执行用户无权读取或修改 SROM 代码。将函数操作码和各参数写入 SPC 输入寄存器上，然后请求 SROM 执行函数。这样 DAP 或 CM0 CPU 将调用 System Call。根据函数操作码， SPC 将从 SROM 执行相应的系统调用，并更新 SPC 状态寄存器。为了能够获得函数执行的成功 / 失败结果，DAP 或 CPU 应当读取该状态寄存器。执行函数时，SROM 中的代码与 SPC 接口交互，以进行实际的闪存编程操作。

使用编程擦除编程 （PEP）序列进行编程 PSoC 4 闪存。将所有闪存单元编程为已知状态 — 擦除状态，然后对选定的位进行编程。这样可以延长闪存的使用寿命，因为存储电荷得到平衡。写入闪存时需要进行两个操作：先将数据复制到页锁存缓冲区内，然后通过使用闪存写函数将该数据传输到闪存。

通过将各指令发送到调试和访问端口 （DAP），外部编程器可以使用 SWD 协议来编程 PSoC 4 中的闪存存储器。有关带有一个外部编程器的 PSoC 4 器件的编程序列，请参考 PSoC 4 器件编程规范中介绍的内容。通过 AHB 接口访问相关寄存器，CM0 CPU 也可以对闪存存储器进行编程。这类编程在引导加载操作期间通常用于更新闪存存储器的部分，或其他应用要求，如更新存储在闪存存储器内的查找表。从 DAP 或 CPU 对闪存存储器进行的所有写操作都通过系统性能控制器 （SPC）接口实现。

注意 写入闪存的操作会需要 20 ms。在这段时间内不该复位器件，否则将导致闪存部分的意外更改。复位源（请参考第 73 页上的复位系统章节）包括 XRES 引脚、软件复位以及看门狗；需要确保不无意激活这些源。另外，低电压检测线路可以配置为生成中断而不是复位。

19.3 系统调用实现

一个系统调用包括以下项目：

- 操作码：一个唯一的 8 位操作码
- 参数：所有系统调用必须有两个 8 位参数。这些参数被称为 key1 和 key2，具体如下定义：

  key1 = 0xB6

  key2 = 0xD3 + 操作码

  两个按键被通过，以确保用户系统调用不被错误启动。如果参数 key1 和 key2 不正确， SROM 将不会执行函数并返回错误代码。除了这两个参数以外，根据被调用的特殊函数，可能还需要其他参数。
- 返回值：某些系统调用在完成执行后，还返回一个值，如芯片 ID 或一个校验和。
- 完成状态: 每个系统调用返回CPU或DAP会读取的一个32位状态，以验证成功或确定失败的原因。

19.4 阻塞和非阻塞的系统调用

根据执行性质，系统调用函数可以划分为阻塞或非阻塞。使用阻塞系统调用情况下， CPU 在执行系统调用时不能执行任何其他任务。从一个进程调用阻塞系统调用时， CPU 跳转到 SROM 中的相应代码。执行完成后，将恢复原始线程执行。非阻塞系统调用允许 CPU 同时可以执行其他代码，另外，通过一个中断向 CPU 通知中间系统调用已完成。

只在 CPU 启动系统调用时，才能使用非阻塞系统调用。在编程模式下， DAP 仅使用系统调用， CPU 在该过程中停止。

三种非阻塞系统调用分别为非阻塞写入行、非阻塞编程行以及恢复非阻塞。所有其他系统调用都是阻塞系统调用。

由于 CPU 在闪存上进行擦除或编程操作时不能执行闪存中的代码，因此只通过在 SRAM 的范围外执行代码才能调用非阻塞系统调用。如果从闪存存储器调用非阻塞函数，则结果将为未定义并会返回一个总线错误，而且执行闪存提取操作时将触发一个硬故障。

系统性能控制器（SPC）是生成合适序列高电压脉冲的模块，该脉冲用于闪存存储器的擦除和编程操作。从 SRAM 调用非阻塞函数时，写入或编程操作中的每个子操作完成后， SPC 定时器将触发其中断。调用 SPC 中断服务子程序 （ISR）中的恢复非阻塞函数，以确保系统调用中的后续步骤完成。执行非阻塞写操作或编程操作时， CPU 只能执行 SRAM 中的代码，因此应该将 SPC ISR 置位在 SRAM 内。调用非阻塞编程函数时， SPC 中断将被触发一次；调用非阻塞写操作时，SPC 中断将被触发三次。在 SPC ISR 中执行的恢复非阻塞函数调用在非阻塞编程操作中被调用一次，而在非阻塞写操作中被调用三次。

用于非阻塞写系统调用和在 SRAM 外执行用户代码的伪代码在后面内容提供。

19.4.1 执行系统调用

下面介绍了启动一个系统调用的流程：

1. 设置函数参数：准备函数参数（key1、key2、其他参数）的两种方法如下所示：
   a. 将函数参数写入 CPUSS_SYSARG 寄存器：该方法用于从 CPUSS_SYSARG 寄存器读取参数的函数。必须将相应系统调用表指定的序列中的参数写入 32 位 CPUSS_SYSARG 寄存器内。
   b. 将函数参数写入 SRAM：该方法用于从 SRAM 读取参数的函数。应该按照指定的序列将这些参数先写入到连续的 SRAM 地址内。然后将 SRAM 的起始地址（即第一个参数的地址）写入到 CPUSS_SYSARG 寄存器内。该起始地址应该始终为字对齐 （32 位）地址。系统调用使用该地址来提取参数。
2. 通过使用系统调用的操作码并启动系统调用来指定系统调用：将 8 位操作码写入 CPUSS_SYSREQ 寄存器中的 SYSCALL_COMMAND 位（[15:0]）内。该伪代码被放置在低八位 [7:0] 内和 0x00 被写入到高八位 [15:8] 内。要启动该系统调用，需要设置 CPUSS_SYSREG 寄存器中的 SYSCALL_REQ 位 （31）。设置该位会触发一个不可屏蔽中断，该中断使 CPU 跳转到操作码参数参照的 SROM 代码。
3. 等待系统调用完成执行：系统调用开始执行时，它将设置 CPUSS_SYSREQ 寄存器中的 PRIVILEGED 位。只有系统调用才能置位该位，CPU 或 DAP 则不能。DAP 应该连续轮询 CPUSS_SYSREG 寄存器中的 PRIVILEGED 位和 SYSCALL_REQ 位，以检查系统调用是否完成。系统调用完成时，这两位将被清除。最大的执行时间是 1 秒。如果超过 1 秒后这两位不被清除，该操作将被视为失败并被停止，而不进行下列步骤。请注意，不如 DAP， CPU 应用代码在系统调用执行期间不能轮询这些位。这是因为在系统调用期间， CPU 在 SROM 外执行代码。该执行从 SROM 返回后，应用代码只能检查最后函数的成功 / 失败状态。
4. 检查完成状态：清除 PRIVILEGED 位和 SYSCALL_REQ 位以指出系统调用已完成后，需要读取CPUSS_SYSARG 寄存器来检查系统调用的状态。若从 CPUSS_SYSARG 寄存器上读取的 32 位值为 0xAXXXXXXX（其中 'X' 表示无需关注的十六进制值），它表示系统调用操作已成功。如果系统调用失败，状态代码将为 0xF00000YY，其中 YY 表示失败的原因。有关状态代码的完整列表和说明，请查看表 19-1 中的内容。
5. 检索返回值：对于返回芯片 ID 和校验和值的系统调用，CPU或DAP需要读取CPUSS_SYSREG和CPUSS_SYSARG 寄存器来提取返回的值。

19.5 系统调用

表 19-1 列出 PSoC 4 支持的所有系统调用和功能说明以及其间保护模式中的可用性。有关器件保护设置的更多信息，请参考第

75 页上的器件安全章节中介绍的内容。请注意， CPU 不能调用一些系统调用，如下表所示。随后是各系统调用的详细内容。

表 19-1. 系统调用列表

系统调用	说明	DAP 访问			CPU 访问
		打开	受保护	停止	
芯片 ID	返回器件的芯片 ID、系列 ID 和修订 ID	4	4	–	4
加载闪存字节	将数据加载到页锁存缓冲区内。该数据将被编程到闪存行，其中粒度为 1 个字节，行大小为 64 个字节	4	–	–	4
写入行	擦除闪存行，然后将页锁存缓冲区中的数据编程到该闪存行内	4	–	–	4
编程行	将页锁存缓冲区中的数据编程到闪存行内	4	–	–	4
擦除所有	擦除闪存阵列中的所有用户代码和监控闪存区中的闪存行级保护数据	4	–	–	
校验和	在整个闪存存储器上（用户和监控区）计算校验和或对单个闪存行进行校验和	4	4	–	4
写保护	通过该函数可以将闪存行级保护设置和芯片级保护设置编程到管理闪存（行 0）内	4	4	–	
非阻塞写入行	擦除闪存行，然后将页锁存缓冲区中的数据编程到该闪存行内。在编程 / 擦除脉冲期间，用户可以从 SRAM 执行代码。该功能仅适用于 CPU 访问	–	–	–	4
非阻塞编程行	将页锁存缓冲区中的数据编程到闪存行内。在编程 / 擦除脉冲期间，用户可以从 SRAM 执行代码。该功能仅适用于 CPU 访问	–	–	–	4
恢复非阻塞	恢复非阻塞写入行或非阻塞编程行。该功能仅适用于 CPU 访问	–	–	–	4

19.5.1 芯片 ID

该函数返回一个 12 位系列编号、一个 16 位芯片 ID、一个 8 位修订编号和当前器件的保护模式。这些值被返回到 CPUSS_SYSARG 和 CPUSS_SYSREQ 寄存器。各参数通过 CPUSS_SYSARG 和 CPUSS_SYSREQ 寄存器被传送。

参数

地址	要写入的值	说明
CPUSS_SYSARG 寄存器		
位 [7:0]	0xB6	Key1
位 [15:8]	0xD3	Key2
位 [31:16]	0x0000	未使用
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0000	芯片 ID 操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [7:0]	芯片 ID 低	有关不同器件型号的芯片 ID 值，请查看 PSoC 4 数据手册
位 [15:8]	芯片 ID 高	
位 [19:16]	次要版本 ID	请参见 PSoC 4 器件编程规范，了解这些值
位 [23:20]	主要版本 ID	
位 [27:24]	0xXX	未使用 （无需关注）
位 [31:28]	0xA	成功状态代码

地址	返回值	说明
CPUSS_SYSREQ 寄存器		
位 [11:0]	系列 ID	PSoC 4 的系列 ID 为 0x093
位 [15:12]	芯片保护	请参阅 第 75 页上的器件安全章节
位 [31:16]	0xXXXX	未使用

19.5.2 配置时钟

该函数初始化闪存编程和擦除操作所需的时钟。该 API 用以确保在调用闪存进行写入和擦除 API 前，已经将电荷泵时钟（clk_pump）和 HF 时钟 （clk_hf）都设为 48 MHz 的 IMO。如果 IMO 是电荷泵时钟源，且它的频率不是 48 MHz，则闪存写入和擦除 API 对闪存不起任何作用，并且操作完成将返回 “ 无效泵时钟频率 ” 状态。

参数

地址	要写入的值	说明
SRAM 地址：32’hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xE8	Key2
位 [31:16]	0xXXXX	无需关注
CPUSS_SYSARG 寄存器		
位 [31:0]	32’hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数 （key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0015	配置时钟操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回值

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.5.3 加载闪存字节

通过使用该功能 可以将要编程为一个闪存行的数据加载到页锁存缓冲区内。加载大小的范围为一个闪存行中的 1 字节到最大字节数量即 64 个字节。数据被加载到页锁存缓冲区内。该缓冲器开始于 “Byte Addr” 输入参数指定的地址。加载到页锁存缓冲区的数据被保持，直到执行清除页锁存内容的编程操作为止。该功能的参数 （包括加载到页锁存的数据）被写入 SRAM 内；SRAM 数据的起始地址被写入到 CPUSS_SYSARG 寄存器内。请注意，参数的起始地址需要为字对齐的地址。

参数

地址	要写入的值	说明
SRAM 地址 — 32’hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xD7	Key2
位 [23:16]	字节地址	用于写数据的页锁存缓冲区的起始地址 0x00 — 锁存缓冲区的字节 0 0x40 — 锁存缓冲区的字节 64

地址	要写入的值	说明
位 [31:24]	闪存宏选择	0x00 — 闪存宏 0 0x01 — 闪存宏 1 （有关器件中的闪存宏数量，请参考第 25 页上的 Cortex-M0 CPU 章节）
SRAM 地址 — 32'hYY + 0x04		
位 [7:0]	装载大小	写入到页锁存缓冲区的字节数量。 0x00 — 1 个字节 0x3F — 64 个字节
位 [15:8]	0xXX	无需关注参数
位 [23:16]	0xXX	无需关注参数
位 [31:24]	0xXX	无需关注参数
SRAM 地址 — 从 （32'hYY + 0x08）到 （32'hYY + 0x08 + 装载大小）		
字节 0	数据字节 [0]	将装载的第一个数据字节
.	.	.
.	.	.
字节 （装载大小 –1）	数据字节 [装载大小 –1]	将装载的最后数据字节
CPUSS_SYSARG 寄存器		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数（key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0004	装载闪存字节操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.5.4 写入行

该功能先擦除寻址闪存行，然后将页锁存缓冲区中的数据编程到该闪存行内。如果页锁存缓冲区中的所有数据都为 0，则该程序将被跳过。该功能的参数被存储在 SRAM 内。存储参数的起始地址被写入到 CPUSS_SYSARG 寄存器内。行被编程后，该功能将清除页锁存缓冲区的内容。

使用要求：在调用该函数前，先调用配置时钟 API。通过配置时钟 API，可确保电荷泵时钟（clk_pump）和 HF 时钟（clk_hf）均被设为 48 MHz 的 IMO。在调用该函数之前，先调用加载闪存字节函数。只在相应的闪存行不是写保护的情况下，该函数才能执行写操作。

请注意，SROM 在任何闪存操作中将不会修改、使能或禁用任何时钟。更多有关信息，请参考 PSoC 4 寄存器技术参考手册中 CLK_IMO_CONFIG 寄存器的内容。

参数

地址	要写入的值	说明
SRAM 地址：32'hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xD8	Key2

位 [31:16]	行 ID	要写入的行号 0x0000 — 行 0
CPUSS_SYSARG 寄存器		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数 （key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0005	写入行的操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功的状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.5.5 编程行

该函数将页锁存缓冲区中的数据编程到寻址闪存行内。如果页锁存缓冲区中的所有数据都为 0，则该程序将被跳过。调用该函数前，必须擦除这一行。该行被编程后，该函数将清除页锁存缓冲区的内容。

使用要求：在调用该函数前，先调用配置时钟 API。通过配置时钟 API，可确保电荷泵时钟（clk_pump）和 HF 时钟（clk_hf）均被设为 48 MHz 的 IMO。

在调用该函数之前，先调用加载闪存字节函数。调用该函数前，必须擦除这一行。只在相应的闪存行不是写保护的情况下，该函数才能执行编程操作。

参数

地址	要写入的值	说明
SRAM 地址：32'hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xD9	Key2
位 [31:16]	行 ID	要编程的行号 0x0000 — 行 0
CPUSS_SYSARG 寄存器		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数 （key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0006	编程行操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功的状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.5.6 擦除所有

通过该函数可以擦除闪存主阵列中的所有用户代码和每个闪存宏的管理闪存第 0 行中的行级保护数据。

使用要求：在调用该函数前，先调用配置时钟 API。通过配置时钟 API，可确保电荷泵时钟（clk_pump）和 HF 时钟（clk_hf）均被设为 48 MHz 的 IMO。

只在芯片保护模式为 OPEN（打开），且 DAP 处于编程模式时，才能从 DAP 调用该 API。如果芯片保护模式为 PROTECTED（保护）， DAP 必须使用写保护 API 将保护设置改为 OPEN（打开）。将保护设置从 PROTECTED 修改为 OPEN 时，会自动执行擦除操作。

参数

地址	要写入的值	说明
SRAM 地址：32'hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDD	Key2
位 [31:16]	0xXXXX	无需关注
CPUSS_SYSARG 寄存器		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数 （key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x000A	擦除所有操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.5.7 校验和

通过使用该函数可以读取整个闪存存储器或一个闪存行，并返回闪存区中读取的每个字节的 24 位总和。在整个闪存上执行校验和时，也对用户代码和监控闪存代码进行校验和。在一个闪存行上执行校验和时，闪存行编号将被作为一个参数传送。通过参数的字节 2 和字节 3，可以选择在整个闪存存储器还是在一个用户代码闪存行上执行校验和。

参数

地址	要写入的值	说明
CPUSS_SYSARG 寄存器		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDE	Key2
位 [31:16]	行 ID	选择执行校验和操作的闪存行 行号 — 16 位闪存的行号 或 0x8000 — 校验和在整个闪存存储器上进行
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x000B	校验和操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:24]	0xX	未使用 （无需关注）
位 [23:0]	校验和	选定的闪存区的 24 位校验和值

19.5.8 写保护

通过该函数可以对监控闪存行中的闪存行级保护设置和器件保护设置进行编程。器件中每个闪存宏的闪存行级保护设置被独立编程的。每行具有一个单保护位。保护字节的总数等于闪存行的数量除以 8。芯片级保护设置 （1 字节）被存储在监控闪存行 0 中最后字节地址上的闪存宏 0 内。监控闪存行的大小等于用户代码闪存行的大小。

使用要求：在调用该函数前，先调用配置时钟 API。通过配置时钟 API，可确保电荷泵时钟 （clk_pump）和 HF 时钟 （clk_hf）均被设为 48 MHz 的 IMO。

通过使用加载闪存字节函数，可以将闪存宏的闪存保护字节加载到相应的页锁存缓冲区内。加载函数的起始地址参数需要为 0。需要对闪存宏编号进行编程；要加载的字节数量是该宏中的闪存保护字节数量。

然后调用写保护函数，以将页锁存中的闪存保护字节编程为相应闪存宏的监控行。在存储器件保护设置的闪存宏 0 内，器件级保护设置作为 CPUSS_SYSARG 寄存器中的参数传送。

参数

地址	要写入的值	说明
CPUSS_SYSARG 寄存器		
位 [7:0]	0xB6	Key1
位 [15:8]	0xE0	Key2
位 [23:16]	器件保护字节	仅适用于闪存宏 0 的参数 0x01 — OPEN （打开）模式 0x02 — PROTECTED （保护）模式 0x04 — KILL （停止）模式
位 [31:24]	闪存宏选择	0x00 — 闪存宏 0 0x01 — 闪存宏 1
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x000D	写保护操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:24]	0xX	未使用 （无需关注）
位 [23:0]	0x000000	

19.5.9 非阻塞写入行

CM0 CPU 需要根据非阻塞方式写入闪存行时，该函数将被使用。这样可以确保当执行写操作时，CPU 能够执行 SRAM 中的代码。有关非阻塞系统调用的说明，请参考第 136 页上的阻塞和非阻塞的系统调用 。

非阻塞写入行的系统调用具有三个阶段：预编程、擦除、编程。在预编程的步骤中，闪存行中的所有位都被写入 ‘1’，以准备执行擦除操作。擦除操作将清除该行中的所有位，然后编程操作将新数据写入该行内。

每个阶段正在执行时， CPU 可以执行 SRAM 中的代码。当启动非阻塞写入行的系统调用时，除了用于完成非阻塞写操作的恢复非阻塞函数外，用户不能调用任何其他系统调用函数。完成每个阶段后， SPC 将触发其中断。在该中断中，将调用恢复非阻塞系统调用。

使用要求：在调用该函数前，先调用配置时钟 API。通过配置时钟 API，可确保电荷泵时钟（clk_pump）和 HF 时钟（clk_hf）均被设为 48 MHz 的 IMO。

注意：在非阻塞写入行过程中，器件的固件不必使器件进入睡眠模式。这将复位页锁存缓冲区，并会将所有零值写入到闪存内。

调用该函数之前，先调用加载闪存字节函数，旨在加载用于编程行的数据字节。另外，只能从 SRAM 调用非阻塞写入行函数。这是因为执行闪存擦除编程操作时， CM0 CPU 不能执行闪存中的代码。如果从闪存存储器调用该函数，则结果为未定义并会返回一个总线错误，而且执行闪存提取操作时会触发一个硬故障。

参数

地址	要写入的值	说明
SRAM 地址：32’hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDA	Key2
位 [31:16]	行 ID	要写入的行号 0x0000 — 行 0
CPUSS_SYSARG 寄存器		
位 [31:0]	32’hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数 （key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0007	非阻塞写入行的操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.5.10 非阻塞编程行

CM0 CPU 需要根据非阻塞方式编程闪存行时，将使用该函数。这样可以确保当执行编程操作时，CPU 能够执行 SRAM 中的代码。有关非阻塞系统调用的说明，请参考第 136 页上的阻塞和非阻塞的系统调用 。执行编程操作时， CPU 不能执行 SRAM 中的代码。当调用非阻塞编程行的系统调用时，除了用于完成非阻塞写操作的恢复非阻塞函数外，用户不能调用任何其他系统调用函数。

与非阻塞写入行的系统调用不同，编程系统调用只有一个阶段。因此，当使用非阻塞编程行的系统调用时，仅需要从 SPC 中断调用恢复非阻塞函数一次。

使用要求：在调用该函数前，先调用配置时钟 API。通过配置时钟 API，可确保电荷泵时钟（clk_pump）和 HF 时钟（clk_hf）均被设为 48 MHz 的 IMO。

调用该函数之前，先调用加载闪存字节函数，旨在加载用于编程行的数据字节。另外，只能从 SRAM 调用非阻塞编程行函数。这是因为执行闪存编程操作时， CM0 CPU 不能执行闪存中的代码。如果从闪存存储器调用该函数，则结果将为未定义并会返回一个总线错误，而且执行闪存提取操作时将触发一个硬故障。

参数

地址	要写入的值	说明
SRAM 地址：32'hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDB	Key2
位 [31:16]	行 ID	要写入的行号 0x0000 — 行 0
CPUSS_SYSARG 寄存器		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数（key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0008	非阻塞编程行的操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.5.11 恢复非阻塞

使用该函数可以完成其他擦除和编程阶段。这些阶段通过使用非阻塞写入行和非阻塞编程行的系统调用来启动。调用非阻塞写入行后，需要调用该函数三次，从 SPC ISR 调用非阻塞编程行后，需要调用该函数一次。所有编程或擦除操作的阶段完成前，不能执行其他系统调用。更多有关使用非阻塞函数的流程，请参考第 136 页上的阻塞和非阻塞的系统调用 。

参数

地址	要写入的值	说明
SRAM 地址：32'hYY （32 位宽、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDC	Key2
位 [31:16]	0xXXXX	无需关注。 SROM 不使用
CPUSS_SYSARG 寄存器		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，用于存储第一个函数参数 （key1）
CPUSS_SYSREQ 寄存器		
位 [15:0]	0x0009	恢复非阻塞操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回

地址	返回值	说明
CPUSS_SYSARG 寄存器		
位 [31:28]	0xA	成功状态代码
位 [27:0]	0xXXXXXXX	未使用 （无需关注）

19.6 系统调用状态

在每个系统调用结束后，会将一个状态代码覆写 CPUSS_SYSARG 寄存器中的参数。成功状态的代码为 0xAXXXXXXX，其中 'X'表示无需关注的值或返回数据（如果系统调用返回某一值）。失败状态的代码为0xF00000XX，其中 'XX' 表示失败的代码。

表 19-2. 系统调用的状态代码

状态代码（CPUSS_SYSARG 寄存器上的 32 位值）	说明
AXXXXXXXh	成功 — "X" 表示 " 无需关注 " 的值。除非 API 直接向 CPUSS_SYSARG 寄存器返回参数，否则该值包含了由 SROM 返回的 '0'。
F0000001h	无效的芯片保护模式 — 在当前的芯片保护模式下，不能用此 API。
F0000003h	无效的页锁存地址 — 表示页锁存缓冲区超出了边界，或者为页面地址所提供过大的地址。
F0000004h	无效的地址 — 所提供的行号或字节地址不处于可用的存储器范围内。
F0000005h	受保护的行 — 所提的行号是一个受保护的行。
F0000007h	恢复过程已完成 — 所有非阻塞 API 操作已完成。不能调用恢复的 API，直到执行下一个非阻塞 API 为止。
F0000008h	挂起恢复 — 启动了一个非阻塞 API。调用任何其他 API 之前，必须通过调用一个恢复 API 完成此操作。
F0000009h	正在进行的系统调用 — 正在进行调用一个恢复的或非阻塞的 API。在尝试下一个恢复过程之前，必须触发 SPC ISR。
F000000Ah	零校验和失败 — 所计算的校验和是非零值。
F000000Bh	无效的操作码 — 操作码不是一个有效的 API 操作码。
F000000Ch	主操作码失配 — 所提供的操作码与 key1 和 key2 不匹配。
F000000Eh	无效的起始地址 — 起始地址大于所提供的结束地址。
F0000012h	无效的泵时钟频率 — 在对闪存进行写入 / 擦除操作前，必须将 IMO 设为 48 MHz， HF 时钟源设为 IMO 时钟源。

19.7 非阻塞系统调用伪代码

本节提供的伪代码演示了如何设置一个非阻塞的系统调用并在闪存编程操作期间在 SRAM 外执行代码。

```
#define REG(addr)(*((volatile uint32 *) (addr)))
#define CM0_ISER_REG REG( 0xE000E100 )
#define CPUSS_CONFIG_REGREG( 0x40100000 )
#define CPUSS_SYSREQ_REG REG( 0x40100004 )
#define CPUSS_SYSARG_REG REG( 0x40100008 )
#define ROW_SIZE              64

//Variable to keep track of how many times SPC ISR is triggered
__ram int iStatusInt = 0x00;

__flash int main(void)
{
   DoUserStuff();

   //CM0 interrupt enable bit for spc interrupt enable
   CM0_ISER_REG |= 0x00000040;

   //Set CPUSS_CONFIG.VECS_IN_RAM because SPC ISR should be in SRAM
   CPUSS_CONFIG_REG |= 0x00000001;
   //Call non-blocking write row API

   NonBlockingWriteRow();

   //End Program
   while(1);
}
```

```
__sram void SpcIntHandler(void)
{
   /* Call Resume API */
   // Write key1, key2 parameters to SRAM
   REG( 0x20000000 ) = 0x0000DCB6;
   //Write the address of key1 to the CPUSS_SYSARG reg
   CPUSS_SYSARG_REG = 0x20000000;
   //Write the API opcode = 0x09 to the CPUSS_SYSREQ.COMMAND
   //register and assert the sysreq bit
   CPUSS_SYSREQ_REG = 0x80000009;
   iStatusInt ++; // Number of times the ISR has triggered
}
 __sram void NonBlockingWriteRow(void)
 {
     int iter;
     /*Load the Flash page latch with data to write*/
     //Write key1, key2, byte address,
     //and macro sel parameters to SRAM
     REG( 0x20000000 ) = 0x0000D7B6;
     //Write load size param ( 64 bytes) to SRAM

     REG( 0x20000004 ) = 0x0000003F;
     for(i = 0; i < ROW_SIZE/4; i += 1);
     {
         REG( 0x20000008 + i*4 ) = 0xDADADADA;
     }
     //Write the address of the key1 param to CPUSS_SYSARG reg
     CPUSS_SYSARG_REG = 0x20000000;
     //Write the API opcode = 0x04 to CPUSS_SYSREQ.COMMAND
     //register and assert the sysreq bit
     CPUSS_SYSREQ_REG = 0x80000004;
     /*Perform Non-Blocking Write Row on Row 200 as an example */
     //Write key1, key2, row id to SRAM
     //row id = 0xC8 -> which is row 200
     REG( 0x20000000 ) = 0x00C8DAB6;
     //Write the address of the key1 param to CPUSS_SYSARG reg
     CPUSS_SYSARG_REG = 0x20000000;
     //Write the API opcode = 0x07 to CPUSS_SYSREQ.COMMAND
     //register and assert the sysreq bit
     CPUSS_SYSREQ_REG = 0x80000007;
     //Execute user code until iStatusInt equals 3 to signify
     //3 SPC interrupts have happened. This should be 1 in case
     // of non-blocking program System Call
     while( iStatusInt != 0x03 )
     {
         DoOtherUserStuff();
     }
     //Get the success or failure status of System Call
     syscall_status = CPUSS_SYSARG_REG;
}
```

在代码中，通过将 0x01 写入 CPUSS_CONFIG 寄存器可内，以将 CM0 异常表配置为处于 SRAM 内。 SRAM 异常表将 SPC 中断的矢量地址作为定义在 SRAM 内的 *SpcIntHandler()* 函数使用。有关将 CM0 异常表配置为处于 SRAM 内的详细信息，请参考第 31 页上的中断章节。非阻塞编程系统调用的伪代码是相同的，但函数操作码和各参数有差异，而且 iStatusInt 变量轮询的是 1 而不是 3。这是因为在非阻塞编程系统调用中，只能触发 SPC ISR 一次。

术语表

术语表这一节介绍了本技术参考手册所使用的术语。术语在整个本手册的文本使用***粗体字***、***斜体字***显示。

A

累加器	中间结果存储在位于 CPU 中的寄存器。如没有累加器，则需要将各项计算的结果（加、减、移位等等）写入到主存储器内，然后读回它们。由于累加器通常具有算术逻辑单元 （ALU）的直接输入 / 输出路径，因此访问主存储器的速度比访问累加器的速度慢。
高电平有效	1. 逻辑信号将其激活状态作为逻辑 1 状态。 2. 逻辑信号将逻辑 1 状态作为两个状态中较高的电压状态。
低电平有效	1. 逻辑信号将其激活状态作为逻辑 0 状态。 2. 逻辑信号将逻辑 1 状态作为两个状态中较低的电压状态：反转逻辑。
地址	确定存储一个信息单位的存储器位置 （RAM、 ROM 或寄存器）的标签或号码。
算法	通过某些限定步骤解决数学问题的程序，这些步骤通常涉及到一个操作的重复。
环境温度	一个指定区域的空气温度，特别是 PSoC 器件周围的区域。
模拟	请参见***模拟信号***。
模拟模块	基本的可编程运算放大器电路。它们是 SC （开关电容）和 CT （连续时间）模块。这些模块内部互联，能够提供 ADC、 DAC、多极滤波器、放大器等多种功能。
模拟输出	该输出可驱动各个电源轨之间的任何电压，而不仅在逻辑 1 或逻辑 0 的电压水平。
模拟信号	在连续时间内连续变化的信号。相反，数字信号是在连续时间内分离变化的信号。
模数转换器 （ADC）	将模拟信号转换为相应量级的数字信号的器件。通常， ADC 可以将电压转换成数字量。*数模（DAC）*转换器可以用于执行逆向操作。
AND	请参见*布尔代数*。
API （应用编程接口）	一系列的软件程序，包括计算机应用与低层服务和函数 （例如，用户模块和库）之间的接口。应用编程接口 （API）用作程序员创建软件应用使用的基本模块。

数组 数组，也称为向量或列表，是电脑编程中最简单的数据结构之一。数组存储固定数目、大小相等和通常属于相同的数据类型的数据元素。与关联数组不同，可通过使用一组连续的整数索引对单个数组进行访问。大多数高级编程语言都具有内置数据类型的数组。某些数组是多维的，也就是说它们是由固定数量的整数索引，例如，包含两个整数的一组。可是，最常见的数组是一维和二维数组。另外，数组还可以是使用几个通用形式来连接的电容或电阻的组合。

汇编 一个特定处理器的机器语言的符号表示法。使用汇编程序将汇编语言转换为机器代码。虽然通常使用宏，但每个汇编代码行一般产生一个机器指令。汇编语言被视为低级语言，相反， C 被视为高级语言。

异步 其数据被立即确认或作出响应的信号，与任何时钟信号无关。

衰减 信号的强度衰减。它是由信号在到达检测器时能量被吸收和分散而导致。它不包括由几何扩展的衰减。衰减的单位通常使用分贝 （dB）。

B

带隙参考 一个稳定电压的参考设计将 V_T 温度正系数与 V_{BE} 温度负系数相互匹配，从而生成零温度系数（理想的）参考。

带宽
1. 消息或信息处理系统的频率范围 （单位为赫兹）。
2. 放大器 （或吸收器）在其频谱区会有大量增益 （或损益）；有时，它表示更为具体，例如，半峰全宽。

偏置
1. 数值与参考值之间的系统偏差。
2. 一组值的平均值偏离参考值的幅度。
3. 针对器件建立运行该器件所需的参考电平所适用的电力、机械力、磁场或其他力 （场）。

偏置电流 用于为放大器产生稳定操作的常量低电平直流电流。在某些情况下，可调整该电流值，以更改放大器的带宽。

二进制 是以 2 为基数的数字系统名称。现在，最常用的数字系统是以 10 为基数的数字系统。数字系统的基数表示系统中一个数字里的特定位置所存在的数字数量。例如，在二进制中，每个位置包含两个值（0 或 1）的其中一个。在十进制数字系统中，每个位置包含十个值（0、1、2、3、4、5、6、 7、 8 和 9）的其中一个。

位 二进制数系统中的单一数字。因此，一位仅代表一个 ‘0’ 或 ‘1’ 。每 8 个位组成一个字节。因为 PSoC 的 M8CP 是一个 8 位微控制器，所以 PSoC 器件的原始数据块的大小是 1 字节。

比特率 （BR） 位流中每个时间单位内处理的位数，通常使用比特每秒 （bps）为单位。

模块
1. 用于执行单项功能的功能性单元，例如振荡器。
2. 用于执行某个功能而配置的功能单位，例如，数字 PSoC 模块或模拟 PSoC 模块。

布尔代数	在科学和计算机科学中，布尔代数或布尔格是捕获了集合运算交集、并集、补集和逻辑运算 AND（与）、OR（或）、NOT（非）三者的根本性质的一个代数结构。布尔代数也定义了如何操作布尔等式的一组定理。例如，通过使用这些定理来简化布尔等式。这样会降低实现等式所需的逻辑元素。 布尔代数的运算符可以使用多种方式来表示。通常，它们简写为 AND、OR 和 NOT。在描述电路的情况下，也可以使用 NAND (NOT AND)、NOR (NOT OR)、XNOR（同 NOT OR）和 XOR（同 OR）。对于 OR，数学家经常使用 ‘+’ 号（例如，A+B），而对于 AND，他们使用• 号（例如，A*B）（在某些情况下，那些运算与代数运算中的减法和乘法类似）。另外，数学家通过在被否定的表达式上面划线来表示 NOT （例如， ~A、 A_、!A）。
先开后合	是指建立新的连接状态 （“ 合 ”）前需要先进入断开状态 (“ 开 ”）的因素。
广播网络	是指在整个微控制器中布置，并可由多个模块或系统访问的信号。
缓冲区	1. 是用来补偿数据从一个器件传输至另一个器件时速度之差的数据存储区。通常是指为 I/O 操作保留的区域，可在此区域读取或写入数据。 2. 往往在将数据发送到外部器件之前或者从外部器件接收数据之前，留出一部分用于存储数据的存储器空间。 3. 用于降低系统输出阻抗的放大器。
总线	1. 网络的命名连接。将网络捆绑到总线中，便于使用类似的布线模式来布线网络。 2. 用于执行常用函数和携带类似数据的一组信号。通常使用矢量符号来表示；例如，地址 [7:0]。 3. 作为一组相关器件的通用连接的一个或多个导体。
字节	是一个等于 8 个位的数据存储单位。

C

C	是一种高级编程语言
电容	电容是由平行的、中间有绝缘介质隔离的两片导体构成，用来衡量当电势差变化时保持电荷的能力。电容的测量单位为法拉 （Farads）。
捕获	通过使用软件或硬件来提取信息，而不是将数据手工输入到计算器文件内。
链路	连接两个或两个以上的 8 位数字模块，以组成 16、 24 乃至 32 位的函数。通过链路，一个模块可以为其他模块产生比较 （Compare）、进位 （Carry)、使能 （Enable）、捕获 （Capture）和门（Gate）等信号。
校验和	可通过将每个数据字添加到总和来生成一组数据的校验和。实际的校验和可以是结果之和，或者是要添加到总和以生成预定值的值。
清除	将某一位或某一寄存器的值强制设置为逻辑 ‘0’ 。
时钟	生成具有固定频率和占空比的周期性信号的器件。有时，时钟可以用来同步化各个不同的逻辑模块。
时钟发生器	用于生成时钟信号的电路。

CMOS	使用以互补方式连接的 *MOS* 构建的逻辑门。 CMOS 是互补金属氧化物半导体的缩略语。
比较器	两个输入电平同时满足预定振幅要求时，生成输出电压或电流的电气电路。
编译器	将高级语言 （例如 C 语言）转换成机器语言的程序。
配置	在计算机系统中，功能性单元的安排取决于其性质、数量以及主要特性。该配置与硬件、软件、固件以及文档有关。它将影响系统性能。
配置空间	在 PSoC 器件中，当 CPU_F 寄存器中的 XIO 位设置为 ‘1’ 时，可以访问寄存器空间。
铁撬	是一种过压保护的电路。当输出电压超出预定的电压值，这个电路会快速将低电阻（通常为 SCR）放在信号和电源轨范围内。
CPUSS	CPU 子系统
晶体振荡器	由压电晶体控制频率的振荡器。通常情况下，压电晶体对环境温度的敏感度低于其他电路组件。
循环冗余校验 （CRC）	检测数据通迅使用的计算方法通常采用线性反馈移位寄存器（LFSR）来执行。相似计算法可用于其他多种用途，例如，数据压缩。

D

数据总线	计算机使用来从存储器位置向中央处理单元 （CPU）或反向传送信息的双向信号组。更为普遍的是，用来传送数字功能之间数据的信息组。
数据流	用于表示传输的信息的一串数字编码信号。
数据传输	使用通道的信号来将数据从一个位置发送到其他位置。
调试器	允许用户用于分析正在开发系统操作的软件和硬件系统。调试器通常允许开发人员单步执行固件，一次执行一步，设置断点和分析存储器。
死区	两个或多个信号都不处于有效状态或切换状态时的一段时间。
十进制	是以 10 为基础的数字系统。它使用符号 0、1、2、3、4、5、6、7、8 和 9（名为数字）以及小数点 ‘+’ （加号）和 ‘-’ （减号）两个符号表示数字。
默认值	适用于预定义的初始的、原始的或特定的设置、条件、数值或某系统所假设的执行、用途，或在没有用户指令的情况下执行。
器件	除非另有说明，否则该手册提及的器件是 PSoC 器件。
die	一个非封装集成电路 （IC），通常从晶圆切割。
数字	一个信号或功能，它的幅值使用两个离散值 “0” 或 “1” 中的一个来描述。
数字模块	可用作计数器、计时器、串行接收器、串行发射器、CRC 发生器、伪随机数发生器或 SPI 的 8 位逻辑模块。

数字逻辑	用于处理表达式的方法。表达式包含了描述电路或系统的行为二状态变量。
数模转换器 （DAC）	可将数字信号转换为对应量级的模拟信号的器件。*模数 （ADC）*转换器可以用来执行逆向运算。
直接访问	根据独立于它们相关位置的序列，使用表示数据的物理位置的地址来获取存储器件中的数据或将数据写入存储器件中的能力。
占空比	是时钟周期的*高电平时间*与其*低电平时间*的关系，表示为一个百分比。

E

外部复位 （XRES_N）	传入 PSoC 器件的有效高电平信号。这导致 CPU 上所有操作和模块停止，并返回到预定义状态。

F

下降沿	从逻辑 1 到逻辑 0 的跃变。它也被称为负向沿。
反馈	将一个 （通常为活动）器件的输出部分或输出的处理部分返回到输入。
滤波器	信号的某些频率组件衰减的器件或过程。
固件	指被嵌入到硬件器件并由 CPU 执行的软件。最终用户可以执行该软件，但不能修改它。
标志	确定条件或事件 （例如，一个字符通知传输操作终止）的任意指示器。
闪存	是可电编程和电擦除、*易失性*的技术，它为用户提供 EPROM 的可编程功能和数据存储，以及系统内可擦除功能。非易失性意味着断电时，数据仍被保留。
闪存组	是闪存模块在一个单一闪存组中以 ‘0’ 开始的一组闪存 ROM 模块。一个闪存组也有其自己的模块等级保护信息。
闪存模块	实现一次性程序化的闪存 ROM 最小空间及受保护的闪存最小空间。闪存模块容量为 64 个字节。
触发器	是具有两种稳定的状态和两个输入端的器件，其中一个输入端与一个状态对应。电路将在剩余的状态下保持。只有使用相应的信号时，它才转到其他状态。
频率	是指周期函数中每个时间单位内的周期数或事件数。

G

增益	输出电流、电压或功率与输入电流、电压或功率各自比率。增益的单位通常使用分贝 （dB）。
门	1. 是具有一个输出通道和一个或多个输入通道的器件。这样，除开关跃变中，输出通道状态完全取决于输入通道状态。 2. 多种组合逻辑元素的其中一个具有最少两个输入（例如， AND、 OR、 NAND 和 NOR（请参见*布尔代数*一节））。

接地

1. 与周围地面具有相同电位的电中性线。
2. 直流电源的负端。
3. 电气系统的参考点。
4. 电路或设备与地面或地面上服务的某些导电体之间的导电路径。

H

硬件

是一个使用于计算机或嵌入式系统所有物理部分的通用术语。硬件不同于它自己所包含或操作的数据和软件 （通过使用该软件提供的指令完成任务）。

硬件复位

是由电路进行的复位，例如 POR、看门狗复位或外部复位。当首次给器件加电时，硬件复位将器件恢复到以前的状态。因此，根据本文档的寄存器表所示，将所有寄存器设为 POR 值。

十六进制

是以 16 为基的数字系统（通常简写为 hex）。它常使用数字 0 到 9 和字母 A 到 F 表示。因为将四个位元化成单独的十六进制数字不太困难，所以十六进制在计算机领域是一个很有用的系统。因此，用户可以使用两个连续的十六进制数字来表示一个字节。我们一起来对二进制、十进制和十六进制表示法进行对比：

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

这样，可以将表示法为 0100 1111b 的十进制数字 79 转为表示法为 0x4F 的十六进制 4Fh

高电平时间

针对一个周期性数字信号，信号在一个周期内具有 ‘1’ 值的时长。

I

I^2C

由 Philips Semiconductors （现更名为 NXP Semiconductors）生产的两线串行计算机总线。 I^2C 是内部集成电路。它用于连接嵌入式系统中的低速外设。最初的系统创建于 20 世纪 80 年代初期，当时只作为电池控制接口，但后来被用作构建控制电子器件使用的简单的内部总线系统。I^2C 仅使用两个双向引脚时钟和数据，二者均运行在 +5 V 的电压上，采用电阻上拉。在标准模式下，总线运行 100 KB，而在快速模式下，总线运行 400 KB。

空闲状态

当用户消息不被传输时但可以立即使用服务的状态。

阻抗	1. 是电流上的电阻，该电阻由电路中的电阻、电容或电感产生。 2. 是供给电流的总负阻抗。请注意，该阻抗是由已给电路中的电阻、感抗和容抗决定的。
输入	在器件、过程、通到中用于接收数据的点。
输入 / 输出 (I/O)	将数据引入系统或从系统中提取数据的器件。
指令	是在编程语言中 （例如 C 或汇编语言）指定一个操作和确定它的操作数。
指令助记符	是表示各汇编语言指令 （例如， ADD、 SUBB、 MOV）的操作码的一组缩略语。
集成电路 （IC）	是将电阻、电容、二级管和*晶体管*等组件制造在半导体单一芯片表面上的器件。
接口	用于使两个系统或器件连接或相互连接的工具。
中断	流程外围事件导致流程暂停 （例如，执行计算机程序），并使用可以恢复流程的方法来执行。
中断服务子程序 （ISR）	M8C 收到硬件中断时常规代码执行转入的代码模块。许多中断源均有各自的优先级和单个 ISR 代码模块。各个 ISR 代码模块均以 RETI 指令结束，并将器件返回到离开常规程序执行的程序点。

J

抖动	1. 从其理想位置跃变的时序错位。在串行数据流中出现的典型的损坏。 2. 一个或多个信号特性突发的不必要变化，例如连续脉冲之间的间隔、连续周期之间的振幅、或连续周期的频率或相位。

L

延迟	信号通过一个给定的电路或网络的所用时间或延迟时间。
最低有效位 （LSb）	在二进制数中表示最低有效值 （通常为右边的位）的二进制数字或位。在 LSb 中的位使用小写字母 “b”，以区分位和字节。
最低有效字节 （LSB）	多字节中表示最低有效值 （通常为右边的字节）的字节。在 LSB 中的位使用大写字母 “B”，以区分位和字节。
线性反馈移位寄存器 （LFSR）	它是一个移位寄存器，其数据输入在寄存器链中被转换为包含两个或两个以上元素的一个 *XOR*。
负载	操作过程所需求的电力，其表现形式为功耗 （瓦特）、电流 （安培）或电阻 （欧姆）。
逻辑函数	是一个对数字数据执行数字运算，然后返回一个数字值的数学函数。

查询表 （LUT） 执行若干逻辑函数的逻辑模块。通过使用选择线选中逻辑函数，并将它应用于模块的输入。例如：可以使用具有 4 个选择线的 2 引脚 LUT 来执行两个引脚上的任一 16 逻辑函数，从而产生一个单一的逻辑输出。 LUT 是一个组合的器件，因此输入 / 输出之间是连续关系，即不被采样。

低时间 针对一个周期数字信号，信号在一个周期内具有 ‘0’ 值的时长。

低压检测 （LVD） 是指在 V_{DDD} 降低到选定阈值以下时，可检测 V_{DDD} 并实现系统中断的电路。

M

M8CP 8 位 Harvard 架构微处理器。微处理器通过连接闪存、 SRAM 和寄存器空间来协调 PSoC 器件内部的所有活动。

宏 编程语言宏是一个抽象概念。它根据一系列预定义的规则替换一定的文本模式。当遇到宏的实例名称，直译器或编译器将自动使用宏内容来替换宏的实例名称。因此，如果使用了一个宏五次，且宏的定义需要 10 个字节的代码空间，则总共需要 50 个字节的代码空间。

掩码
1. 用于掩饰、隐藏信息，或以其他方式防止信息派生自信号。掩码经常是与其他信号相交互的结果，例如噪声、静态、拥塞或干扰的其他形式。
2. 可使用位模式来保留或抑制计算和数据处理系统中其他位模式的段式。

主设备 用于控制两个器件之间数据交换时序的器件。或者，以脉冲宽度级联器件时，主设备是用来控制级联器件与外部接口之间数据交换时序的器件。受控制的器件被称作*从设备*。

微控制器 主要用于控制系统和产品的集成电路器件。除 CPU 外，微控制器通常还包含存储器、定时电路和 I/O 电路。这是为了允许执行包含最小器件数量的控制器，从而能实现最大程度的微型化。相反，这会降低控制器的体积和成本。由于微控制器是一个微处理器，它通常不用于通用计算功能。

助记符 用于协助存储器的工具。通过重复助记符不仅记住了事实，而且还创建了易记结构和数据列表之间的关联。表示微处理器指令的一个具有两到四个字符的字符串。

模式 软件或硬件的不同操作方式。例如，数字 PSoC 模块可以处于计数器模式或定时器模式。

调制 用于编码载波信号（通常为正弦波信号）上的信息的一系列技术。执行调制的器件被称为调制器。

调制器 在载波上附加信号的器件。

MOS 是金属氧化物半导体的缩略语。

最高有效位 （MSb） 在二进制数中表示最高有效值（通常为左边的位）的二进制数字或位。在 MSb 字母中的位使用小写字母 “b”，以区分位和字节。

最高有效字节 （MSB） 多字节中表示最高有效值的字节（通常为左边的字节）。在 MSB 字母中的位使用大写字母 “B”，以区分位和字节。

复用器 （mux）
1. 一个使用二进制值，或地址来选择多个输入信号中的一个，然后将选定输入信号的数据传递至输出信号中的逻辑功能。
2. 允许不同的输入 （或输出）信号在不同的时间使用同样的线路，并由外部信号控制的技术。若在线路上和 I/O 端口上使用，复用器具有保留的功能。

N

NAND	请参见 *布尔代数*。
下降沿	从逻辑 1 到逻辑 0 的跃变。它也被称为负向沿。
网络	是器件之间的走线。
半字节	包含四个位，就是一个字节的一半。
噪声	1. 会影响信号，且会使信号携带的信息失真的干扰。 2. 如电压、电流或数据等任何实体中一种或多种特性发生的随机变化。
NOR	请参见 *布尔代数*。
NOT	请参见 *布尔代数*。

O

OR	请参见 *布尔代数*。
振荡器	可受晶控，并用于生成时钟频率的电路。
输出	由模拟或数字模块生成的电子信号或信号。

P

并行	是指数字数据每次能发送多个位，其中每个同步位通过一个独立的线路进行传送的通信方式。
参数	是已给模块的特性。这些特性已经被定性，或可由设计人员定义。
参数模块	是指执行存储器中的 SSC 指令参前被存储的位置。
奇偶校验	用于测试传输数据的技术。通常，将一个二进制数字添加到数据中，以便求所有二进制数据奇数之和 （奇校验）或偶数之和 （偶校验）。
路径	1. 是由计算机执行的指令的逻辑序列。 2. 电路中的电子信号流。
待处理中断	一个被触发但未得到处理的中断，这可能是因为处理器正在忙着处理其他中断，或全局中断被禁用。
相位	是两个信号 （通常为具有相同频率的信号）之间的关系。此关系决定信号之间的延迟。信号间的此延迟可使用时间或角 （度）来测量。
引脚	是硬件组件上的终端。它也被称为接脚。

引脚分布	引脚号分配：印刷电路板 (PCB) 封装中 PSoC 器件及其物理对立方的逻辑输入与输出之间的关系。引脚分布涉及到原理图与 PCB 设计（两者均是计算机生成的文件）之间链接的引脚号，也涉及道引脚名称。
端口	一组引脚，通常有八个。
正向沿。	从逻辑 0 到逻辑 1 的跃变。它也被称为上升沿。
发布的中断	一个由硬件检测，但可能或可能不被屏蔽位使能的中断。未屏蔽的发布中断变成了挂起中断。
上电复位 （POR）	当电压下降至预设电压时强迫 PSoC 器件复位的电路。这是一种*硬件复位*的类型。
程序计数器	指令指针（又称程序计数器）是电脑处理器中的寄存器，该处理器指出在存储器中 CPU 处理指令的位置。根据特定机器的详细内容，它会存储将被执行的指令的地址，或将被执行的下一个指令的地址。
协议	是一组规则。特别是控制网路通信的规则。
PSoC®	赛普拉斯的可编程片上系统 （PSoC®）器件。
PSoC 模块	*请参见 模拟模块*和*数字模块*。
PSoC Creator™	赛普拉斯的新一代可编程片上系统技术的软件。
脉冲	是指信号 （例如，相位或频率）某些特性的快速变化。它从基线的值变为更高或更低的值，然后快速返回到基线的值。
脉冲宽度调制器（PWM）	占空比形式表示的输出，它随着应用测量对象的不同而变化。

R

RAM	随机存取存储器的缩略语。数据存储器件，可以对该器件进行读写操作。
寄存器	具有特定容量 （例如一位或字节）的存储器件。
复位	使系统返回已知状态的方法。请参见*硬件复位*和*软件复位*。
电阻	导体的电流电阻，单位为欧姆 （ohms）
版本 ID	PSoC 器件的唯一标识符。
纹波分频器	是触发器所构成的一个异步纹波计数器。时钟反馈到计数器的第一段。包含 n 触发器的 n 位二进制计数器可以从 0 到 2^n - 1 进行计数的二进制值。
上升沿	*请参见 正向沿*。
ROM	只读存储器的缩略语。数据存储器件，可以对该器件进行读操作但无法进行写操作。
子程序	是一个代码模块。它由其他代码模块调用，另外，还具有通用或频繁使用方式。

布线	是根据参考库中的设计规则而物理上连接各对象。
短脉冲	在数字电路中，窄脉冲是指由于信号的非零上升和下降时间，信号未达到一个有效高电平或低电平。例如，短脉冲可能发生在由异步时钟间进行切换，或者是信号经过两个独立的路径通过同一个电路造成竞争状态。在这些竞争状态下可能有不同的延迟，然后形成毛刺或者此时一个触发器的输出成为亚稳定状态。

S

采样	表示将模拟信号转换为一系列数字值或相反的过程。
原理图	是详细描述一个系统的元件 （例如电子电路的元件或计算机中一个逻辑图的元素）的图、形图或草图。
种子值	是加载到线性反馈移位寄存器或随机数发生器中的初始值。
串行	1. 表示所有事件在其中相继发生的流程。 2. 表示在单个器件或通道中两个或多个相关活动的连续发生。
设置	将某一位或某一寄存器的值强制设置为逻辑 1。
建立时间	输入从一个值改为另一个值后，输出信号或值变为稳定状态需要的时长。
移位	是字位置中位的移位，可移到左边或右边。例如，如果十六进制值 0x24 向左边移位，它将成为 0x48。如果十六进制值 0x24 向右边移位，则它便成为 0x12。
移位寄存器	按顺序向左或向右转移一个文字以便输出串行数据流的存储器存储器件。
符号位	是符号二进制数的最高有效二进制数字或位。如果将它设置为逻辑 1，此位将表示负值。
信号	用于传递信息的可测试传送能量。使用于各电子和任何传送电脉冲。
芯片 ID	PSoC 芯片的唯一标识符。
时滞	是在并行发送中，同时传送的位的到达时间差别。
从设备	允许另一个器件控制两个器件之间数据交换的时序的器件。或者，以脉冲宽度级联器件时，从设备是允许另一个器件控制级联器件与外部接口之间数据交换的时序的器件。控制器件被称为主设备。
软件	是一系列有关数据处理系统操作的电脑程序、过程和相关文档的结合（例如，编译器、库子程序、手册和电路图） 。通常，先将软件编写为源代码，然后将它转换为适合器件执行的二进制格式代码。
软件复位	是软件执行的部分复位，以将部分系统返回已知的状态。复位软件时，则将 M8CP 恢复到一个已知的状态，而不是恢复到 PSoC 模块、系统、外设或寄存器。复位软件时，要将 CPU 寄存器（CPU_A、CPU_F、CPU_PC、CPU_SP 和 CPU_X）的值设为 0x00。因此，将在闪存地址 0x0000 开始执行代码。

SRAM	静态随机存取存储器的缩略语。可以高速存储和检索数据的存储器器件。之所以使用术语 “ 静态 ”，是因为在将某一值加载到 SRAM 单元时，该值会保持不变，直至它被明确更改，或直至器件断电为止。
SROM	只读管理存储器的缩略语。 SROM 保留用以引导器件、校准电路和执行闪存操作的代码。使用常规用户代码访问 SROM 功能，并从闪存中运行。
堆栈	堆栈是按照后入先出 （LIFO）的原理运作的数据结构。就是说，输入堆栈的最后项目也是取出的第一项目。
堆栈指针	堆栈可能位于计算机中模块里的存储器单元。它的底部放在单元的一个固定位置上，而变量堆栈指针放在当前顶部单元上。
状态机	是一个功能的实际实现过程 （在硬件或软件中），它可以包括一组需要按序列进入的状态。
粘滞	它是指寄存器中的一位，该位能保持它的值，直到发生导致其转换的事件为止。
停止位	是特征或模块带有的信号，用于使接收器件准备好接收下一个特征或模块。
开关	电路上信号的控制或布线，用以执行逻辑或算术操作，或在网络中各个特定点之间传输数据。
开关相位	根据开关电容控制一个给定的开关（PHI1 或 PHI2）的时钟。PSoC SC 模块具有两组开关。一组开关通常在 PHI1 运行期间被关闭，则在 PHI2 运行期间被打开。相反，剩余的一组切换在 PHI1 运行期间被打开，则在 PHI2 运行期间被关闭。在正常操作中，可以控制这些开关。如果 PHI1 和 PHI2 时钟被反转，则可在反转模式下控制它们。
同步	1. 是指其数据未被确认或做出响应，直到时钟信号的下一个边沿有效为止的信号。 2. 其操作由时钟信号进行同步的系统。

T

抽头	是指器件中两个模块通过以串行方式来连接某些模块 / 组件，如移位寄存器或电阻电压分频器的连接。
计数终值	计数器倒计至零的状态。
阈值	系统或传感器在考虑下可检测的信号的最低值。
Thumb-2	Thumb-2 指令集是一组高效强大的指令集合。从易用性、代码大小和性能的角度来说，它能够带来显著利益。除 32 位指令外，通过添加 16 位指令，Thumb-2 指令集是先前的 17 位 Thumb 指令集的超级集合。
晶体管	晶体管是一个固态半导体器件，用于放大增益和切换。它具有三个终端：向一个终端提供的小电流或电压控制其他两个终端的电流。它是现代电器的重要组成部分。在数字电路，可将晶体管做为很快的电子开关使用，另外，组织的晶体管可充当逻辑门、 RAM 型和其他器件。在模拟电路，基本上将晶体管作为放大器使用。
三态	其输出可采用三种状态的功能：0、 1 和 Z （高阻抗）。该功能不驱动 Z 状态下的任何值，在许多方面，它可以被视为从其余电路断开，允许另一次输出以驱动相同*网络*。

U

UART	UART 或通用异步接收器 - 发射器在数据并行位和串行位之间转换。
用户	使用 PSoC 器件或阅读本手册的人。
用户模块	需要全面管理和配置低级模拟和数字 PSoC 模块的预构建、预测试硬件 / 固件外围功能。此外，用户模块还针对外围功能提供高级 *API （应用编程接口）*。
用户空间	寄存器映射的组 0 空间。执行常规程序期间和初始化期间，很可能对该组中的寄存器进行了修改。程序初始化阶段，很可能对组 1 中的寄存器进行了修改。

V

V_{DDD}	电力网名称，意为 “ 电压漏极 ”。最正极的电源信号。电压通常为 5 或 3.3 伏。
易失性	如果不在适用范围内，则不保证位置相同值。
V_{SS}	电力网名称，意为 “ 电压源 ”。最负极的电源信号。

W

看门狗定时器	一个必须定期刷新的定时器。如果未定期刷新，则 CPU 会在指定时间期间后复位。
波形	是一个信号的表示法，以振幅和时间图显示。

X

XOR	请参见 *布尔代数*。

索引

A

active mode
PSoC 66

B

block diagram
GPIO 46
port interrupt controller unit 50
program and debug interface 129
watchdog timer circuit 69
brownout reset 73

C

clock distribution 55
clock sources
distribution 55
clocking system
introduction 53
Cortex-M0
features 25
instruction set 28
registers 26

D

development kits 17
document
glossary 149
revision history 11

E

exception
HardFault 33
NMI 33
PendSV 34
reset 33
SVCall 33
SysTick 34
external reset 74

F

features
I/O system 45
port interrupt controller unit 50
watchdog timer 69

G

glossary 149
GPIO
block diagram 46
GPIO pins in creation of buttons and sliders 49

H

high impedance analog drive mode 48
high impedance digital drive mode 48
how it works
watchdog timer 70

I

I/O drive mode
high impedance analog 48
high impedance digital 48
open drain 48
resistive 48
strong 48
I/O system
CapSense 49
features 45
introduction 45
open drain modes 48
port interrupt controller unit pin configuration 50
register summary 51
resistive modes 48
slew rate control 48
strong drive mode 48
identifying reset sources 74
internal low speed oscillator 55
internal main oscillator 55
internal regulators 59
introduction
clock generator 53
I/O system 45
reset 73

O

oscillators
internal PSoC 55
overview, document
revision history 11

P

port interrupt controller
features 50
port interrupt controller unit
block diagram 50
pin configuration 50
power on reset 73
program and debug
PSoC 16
protection fault reset 74
PSoC
active mode 66
program and debug 16
PSoC 4
major components 14

R

register summary
I/O system 51
registers
Cortex-M0 26
regulator
internal 59
reset
identifying sources 74
introduction 73
reset sources
description 73
revision history 11

S

sleep mode 66
slew rate control in I/O system 48
software initiated reset 73
support 17
SWD interface
program and debug interface 130
system call
overview 136

U

upgrades 17

W

watchdog reset 73
watchdog timer
features 69
how it works 70
interrupts 70