

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 4200L 系列

PSoC[®] 4 架构 技术参考手册 (TRM)

文档编号: 002-11591 版本 **

May 17, 2016

赛普拉斯半导体公司
198 Champion Court
San Jose, CA 95134-1709
电话 (美国): 800.858.1810
电话 (国际): 408.943.2600
www.cypress.com

版权所有

许可证

© 赛普拉斯半导体公司，2015-2016。保留所有版权。此软件以及相关的文档或材料均归赛普拉斯半导体公司（赛普拉斯）所有，并受全球专利法规（美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。除非您和赛普拉斯公司达成了单独的许可协议，否则您需要同意将赛普拉斯的资料作为受版权保护的材料使用。

您需要同意保密使用赛普拉斯材料，并在未经书面授权时不可透露或使用赛普拉斯的材料。您需要遵守您和赛普拉斯公司之间的所有保密协议。

如果材料涉及到第三方许可证，您需要遵守该许可。

版权所有

版权所有 © 2015-2016 赛普拉斯半导体公司。保留所有版权。

PSoC 和 CapSense 均为赛普拉斯半导体公司的注册商标，且 PSoC Creator、Cypress® 以及 Cypress Semiconductor™ 都是赛普拉斯半导体公司的商标。此处引用的所有其他商标或注册商标均归其各自所有者所有。

从赛普拉斯或某个获得赛普拉斯授权的联营公司处购买的 I²C 组件，即可根据飞利浦 I²C 专利权获得一份许可，以便在 I²C 系统中使用这些组件，但前提要保证该系统符合飞利浦定义的 I²C 标准规范。自 2006 年 10 月 1 日起，飞利浦半导体就采用了一个新的商标名称 — NXP 半导体。

本文档中的信息如有更改，恕不另行通知，而且此文档亦不应该被视为一个承诺。尽管已采取合理的预防措施，赛普拉斯对本文档中任何可能出现的错误不承担任何责任。未经赛普拉斯书面同意，不得以任何形式或任何方式对本文档的任何部分进行复制或转载。美国制造。

免责声明

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

闪存代码保护

赛普拉斯产品符合赛普拉斯的相应数据手册中的规范。赛普拉斯坚信，不论如何使用，其 PSoC 系列产品的安全性在目前市场上的同类产品中始终名列前茅。目前可能存在一些能够破坏代码保护功能的方法。据我们所知，任何此类方法都是不正当的，并且可能是违法的。不只是赛普拉斯，任何其他半导体制造商都无法保证各自代码的安全性。代码保护并非意味着我们保证产品“坚不可摧”。

赛普拉斯非常希望能够与关注其代码完整性的客户通力合作。代码保护技术正在不断发展。持续改进产品的代码保护功能是赛普拉斯的不懈追求。

目录概览



节 A:	概述	15
	1. 简介	17
	2. 入门	21
	3. 文档结构	23
节 B:	CPU 系统	27
	4. Cortex-M0 CPU	29
	5. DMA 控制模式	35
	6. 中断	51
节 C:	系统范围资源	61
	7. I/O 系统	63
	8. 时钟系统	77
	9. 电源与电源监测	91
	10. 芯片运行模式	95
	11. 功耗模式	97
	12. 看门狗定时器	101
	13. 复位系统	105
	14. 器件安全性	109
节 D:	数字系统	111
	15. 通用数字模块 (UDB)	113
	16. 控制器区域网络 (CAN)	153
	17. 通用串行总线 (USB)	171
	18. 定时器、计数器和 PWM	189
节 E:	模拟系统	211
	20. 高精度参考	213
	21. SAR ADC	217
	22. 低功耗比较器	247
	23. 微型连续时间模块 (CTBm)	253
	24. LCD 直接驱动	263
	25. CapSense	275
	26. 温度传感器	285

节 F:	编程和调试	289
	27. 编程与调试接口	291
	28. 非易失性存储器编程	299
术语表		313
索引		329

目录



节 A:	概述	15
	文档修订记录	15
1.	简介	17
1.1	系统架构	17
1.2	特性	18
1.3	CPU 系统	18
1.3.1	处理器	18
1.3.2	中断控制器	18
1.3.3	直接存储器访问	19
1.4	存储器	19
1.4.1	闪存	19
1.4.2	SRAM	19
1.5	全局系统资源	19
1.5.1	时钟系统	19
1.5.2	供电系统	19
1.5.3	GPIO	19
1.6	可编程数字模块	19
1.7	固定功能数字模块	20
1.7.1	定时器 / 计数器 / PWM 模块	20
1.7.2	串行通信模块	20
1.7.3	控制器区域网络 (CAN)	20
1.8	模拟系统	20
1.8.1	SAR ADC	20
1.8.2	微型的连续时间模块 (CTBm)	20
1.8.3	低功耗比较器	20
1.9	特殊的功能外设	20
1.9.1	LCD SEGMENT 驱动	20
1.9.2	CapSense	20
1.10	编程和调试	20
2.	入门	21
2.1	支持	21
2.2	产品升级	21
2.3	开发套件	21
2.4	应用笔记	21
3.	文档结构	23
3.1	主要部分	23
3.2	文档规范	23
3.2.1	寄存器规定	23
3.2.2	数字命名	23
3.2.3	测量单位	24
3.2.4	缩略语	24

节 B:	CPU 系统	27
	系统架构	27
4.	Cortex-M0 CPU	29
4.1	特性	29
4.2	框图	30
4.3	工作原理	30
4.4	地址映射	30
4.5	寄存器	31
4.6	工作模式	32
4.7	指令集	32
	4.7.1 地址对齐	33
	4.7.2 存储器中的字节顺序	33
4.8	Systick 定时器	33
4.9	调试	33
5.	DMA 控制模式	35
5.1	框图说明	35
	5.1.1 触发源和复用	36
	5.1.2 挂起触发	39
	5.1.3 输出触发	39
	5.1.4 通道优先级	39
	5.1.5 数据传输引擎	39
5.2	描述符	40
	5.2.1 地址配置	40
	5.2.2 传输大小	41
	5.2.3 描述符链接	42
	5.2.4 传输模式	42
5.3	操作与时序	46
5.4	仲裁	47
5.5	寄存器列表	49
6.	中断	51
6.1	特性	51
6.2	工作原理	51
6.3	中断和异常事件 — 操作	52
	6.3.1 PSoC 4 中的中断 / 异常事件处理	52
	6.3.2 电平与脉冲中断	52
	6.3.3 异常事件向量表	53
6.4	异常事件源	53
	6.4.1 复位异常事件	53
	6.4.2 不可屏蔽中断 (NMI) 异常事件	54
	6.4.3 HardFault (硬故障) 异常事件	54
	6.4.4 管理程序调用 (SVCall) 异常事件	54
	6.4.5 PendSV 异常事件	54
	6.4.6 SysTick 异常事件	54
6.5	中断源	55
6.6	异常事件优先级	56
6.7	使能和禁用中断	56
6.8	异常事件的状态	57
	6.8.1 挂起异常事件	57
6.9	异常事件的堆栈使用情况	57
6.10	中断和低功耗模式	58
6.11	异常事件 — 初始化和配置	58
6.12	寄存器	59

6.13	相关文档	59
节 C:	系统范围资源	61
	顶层架构	61
7.	I/O 系统	63
7.1	特性.....	63
7.2	GPIO 接口概况	63
7.3	I/O 单元架构.....	64
7.3.1	数字输入缓冲器	65
7.3.2	数字输出驱动器	65
7.4	GPIO-OVT 引脚	67
7.5	特殊 I/O (SIO)	69
7.5.1	参考电压发生器	70
7.5.2	差分输入缓冲器	71
7.5.3	稳压输出驱动器	71
7.5.4	过压容差	72
7.6	高速输入 / 输出矩阵	72
7.7	上电时的 I/O 状态	73
7.8	低功耗模式下的行为	73
7.9	输入和输出的同步	73
7.10	中断.....	73
7.11	外设连接	75
7.11.1	固件控制的 GPIO	75
7.11.2	模拟 I/O	75
7.11.3	LCD 驱动.....	75
7.11.4	CapSense	75
7.11.5	串行通信模块 (SCB)	76
7.12	端口限制	76
7.13	寄存器	76
8.	时钟系统	77
8.1	框图.....	77
8.2	时钟源	78
8.2.1	内部主振荡器.....	78
8.2.2	内部低速振荡器	80
8.2.3	外部时钟 (EXTCLK)	80
8.2.4	外部晶体振荡器 (ECO)	80
8.2.5	PLL0 和 PLL1	81
8.2.6	监视晶体振荡器 (WCO)	82
8.3	时钟分布	83
8.3.1	HFCLK 和 PLL 输入选择	83
8.3.2	HFCLK 输入选择	83
8.3.3	PLL 输入选择	84
8.3.4	LFCLK 输入选择.....	84
8.3.5	SYSCLK 预分频器配置	85
8.3.6	外设时钟分频器的配置	85
8.3.7	外设时钟配置.....	87
8.4	低功耗模式操作.....	88
8.5	寄存器列表	89
9.	电源与电源监测	91
9.1	框图.....	92
9.2	工作原理	92
9.2.1	电压调节器汇总	92

9.3	电压监控	93
9.3.1	上电复位 (POR)	93
9.4	寄存器列表	94
10.	芯片运行模式	95
10.1	启动模式	95
10.2	用户模式	95
10.3	特权模式	95
10.4	调试模式	95
11.	功耗模式	97
11.1	活动模式	98
11.2	睡眠模式	98
11.3	深度睡眠模式	98
11.4	休眠模式	99
11.5	停止模式	99
11.6	功耗模式总结	99
11.7	进入和退出低功耗模式	100
11.8	寄存器列表	100
12.	看门狗定时器	101
12.1	特性	101
12.2	框图	101
12.3	工作原理	102
12.3.1	使能和禁用 WDT	103
12.3.2	WDT 操作模式	103
12.3.3	WDT 中断和低功耗模式	104
12.3.4	WDT 复位模式	104
12.4	寄存器列表	104
13.	复位系统	105
13.1	复位源	105
13.1.1	上电复位	105
13.1.2	欠压复位	105
13.1.3	看门狗复位	105
13.1.4	软件复位	106
13.1.5	外部复位	106
13.1.6	保护错误复位	106
13.1.7	休眠模式唤醒复位	106
13.1.8	停止模式唤醒复位	106
13.2	识别复位源	106
13.3	寄存器列表	107
14.	器件安全性	109
14.1	特性	109
14.2	工作原理	109
14.2.1	器件安全性	109
14.2.2	闪存安全性	110
节 D:	数字系统	111
	系统架构	111
15.	通用数字模块 (UDB)	113
15.1	特性	113
15.2	工作原理	114
15.2.1	PLD	114
15.2.2	数据路径	116

15.2.3	状态和控制模块	134
15.2.4	复位和时钟控制模块	140
15.2.5	UDB 寻址	146
15.2.6	系统总线访问一致性	147
15.3	端口适配器模块	148
15.3.1	PA 数据输入逻辑	148
15.3.2	PA 端口引脚的时钟复用器逻辑	149
15.3.3	PA 数据输出逻辑	149
15.3.4	PA 输出使能逻辑	150
15.3.5	PA 时钟复用器	151
15.3.6	PA 复位复用器	151
16.	控制器区域网络 (CAN)	153
16.1	特性	153
16.2	框图	154
16.3	CAN 中的信息帧	154
16.3.1	数据帧	154
16.3.2	远程帧	155
16.3.3	错误帧	155
16.3.4	过载帧	156
16.4	在 CAN 中发送信息	156
16.4.1	信息仲裁	156
16.4.2	信息发送过程	157
16.4.3	信息中止	157
16.4.4	一次性发送	157
16.4.5	发送扩展的数据帧	157
16.5	CAN 内接收信息	158
16.5.1	信息接收过程	158
16.5.2	验收滤波器	159
16.5.3	DeviceNet 筛选	160
16.5.4	扩展的数据帧筛选	161
16.5.5	接收信息缓冲区链接	161
16.6	远程帧	162
16.6.1	通过请求节点发送远程帧	162
16.6.2	接收一个远程帧	162
16.6.3	RTR 自动回复	163
16.6.4	扩展格式的远程帧	163
16.7	时间触发的 CAN	163
16.7.1	TTTCAN 定时器	163
16.8	位时间配置	163
16.8.1	所允许的比特率和系统时钟 (SYSCLK)	163
16.8.2	设置比特率 TSEG1 和 TSEG2	164
16.9	CAN 中的错误和中断	166
16.9.1	错误类型	166
16.9.2	错误捕获寄存器	166
16.9.3	CAN 中的错误状态	167
16.9.4	CAN 中的中断源	167
16.10	CAN 的工作模式	169
16.10.1	运行 / 停止模式	169
16.10.2	“ 仅侦听 ” 模式	169
16.10.3	环回测试模式	169
17.	通用串行总线 (USB)	171
17.1	特性	171

17.2	框图	172
17.2.1	USB 物理层 (USB PHY)	172
17.2.2	串行接口引擎 (SIE)	172
17.2.3	仲裁器	172
17.3	工作原理	173
17.3.1	USB 物理层 (USB PHY)	173
17.3.2	端点	176
17.3.3	传输类型	176
17.3.4	中断	176
17.3.5	DMA 支持	178
17.4	逻辑传输模式	178
17.4.1	存储和转发模式	180
17.4.2	直通模式	182
17.4.3	控制端点的逻辑传输	184
17.5	寄存器汇总	186
18.	定时器、计数器和 PWM	189
18.1	特性	189
18.2	框图	190
18.2.1	使能和禁用 TCPWM 模块中的计数器	190
18.2.2	时钟	190
18.2.3	基于触发输入的事件	191
18.2.4	输出信号	191
18.2.5	功耗模式	193
18.3	各种操作模式	193
18.3.1	定时器模式	194
18.3.2	捕获模式	196
18.3.3	正交解码器模式	198
18.3.4	脉宽调制模式	201
18.3.5	带死区时间模式的脉宽调制	204
18.3.6	脉宽调制伪随机模式	206
18.4	TCPWM 寄存器	209
节 E:	模拟系统	211
	系统架构	211
20.	高精度参考	213
20.1	特性	213
20.2	框图	213
20.3	工作原理	214
20.3.1	高精度带隙	214
20.3.2	调整缓冲区	214
20.3.3	低功耗缓冲区	214
20.3.4	电流镜像	215
20.3.5	温度控制的电压发生器	215
20.3.6	温度控制的电流发生器	215
20.4	配置	215
21.	SAR ADC	217
21.1	特性	217
21.2	框图	218
21.3	工作原理	218
21.3.1	SAR ADC 内核	218
21.3.2	SARMUX	222
21.3.3	SARREF	227

	21.3.4 SARSEQ	229
	21.3.5 中断	232
	21.3.6 触发器	233
	21.3.7 SAR ADC 状态	234
	21.3.8 低功耗模式	234
	21.3.9 系统操作	235
	21.3.10 寄存器模式	236
	21.3.11 DSI 模式	238
	21.3.12 模拟路由配置示例	241
	21.3.13 温度传感器配置	244
	21.4 寄存器	245
22.	低功耗比较器	247
	22.1 特性	247
	22.2 框图	247
	22.3 工作原理	248
	22.3.1 输入配置	248
	22.3.2 输出和中断配置	248
	22.3.3 功耗模式和速度配置	250
	22.3.4 迟滞	250
	22.3.5 从低功耗模式唤醒	251
	22.3.6 比较器时钟	251
	22.3.7 偏移调整	251
	22.4 寄存器汇总	252
23.	微型连续时间模块 (CTBm)	253
	23.1 特性	253
	23.2 框图	253
	23.3 工作原理	254
	23.3.1 功耗模式配置	254
	23.3.2 输出驱动配置	255
	23.3.3 补偿	256
	23.3.4 开关控制	256
	23.4 寄存器汇总	261
24.	LCD 直接驱动	263
	24.1 特性	263
	24.2 LCD segment 驱动概述	263
	24.2.1 驱动模式	264
	24.2.2 建议使用的驱动模式	272
	24.2.3 数字对比度控制	272
	24.3 框图	273
	24.3.1 工作原理	273
	24.3.2 高速和低速主发生器	273
	24.3.3 复用器及 LCD 引脚逻辑	274
	24.3.4 显示数据寄存器	274
	24.4 寄存器列表	274
25.	CapSense	275
	25.1 特性	275
	25.2 框图	275
	25.3 工作原理	276
	25.4 CapSense CSD 感应	277
	25.4.1 GPIO 单元中电容 - 电流转换器	277
	25.4.2 CapSense 时钟发生器	279
	25.4.3 Sigma Delta 转换器	279

25.5	CapSense CSD 屏蔽	281
25.5.1	CMOD 预充电	282
25.6	通用资源: IDAC 和比较器	283
25.7	寄存器列表	283
26.	温度传感器	285
26.1	特性	285
26.2	工作原理	285
26.3	温度传感器配置	286
26.4	算法	287
26.5	寄存器	287
节 F:	编程和调试	289
	系统架构	289
27.	编程与调试接口	291
27.1	特性	291
27.2	功能说明	291
27.3	串行线调试 (SWD) 接口	292
27.3.1	SWD 时序的详细信息	293
27.3.2	ACK 的详细信息	293
27.3.3	反转 (Trn) 周期的详细信息	293
27.4	Cortex-M0 调试和访问端口 (DAP)	294
27.4.1	调试端口 (DP) 寄存器	294
27.4.2	访问端口 (AP) 寄存器	294
27.5	编程 PSoC 4 器件	295
27.5.1	获取 SWD 端口	295
27.5.2	SWD 编程模式入口	295
27.5.3	SWD 编程子程序执行	295
27.6	PSoC 4 SWD 调试接口	296
27.6.1	调试控制和配置寄存器	296
27.6.2	断点单元 (BPU)	296
27.6.3	数据观察点 (DWT)	296
27.6.4	调试 PSoC 4 器件	296
27.7	寄存器	297
28.	非易失性存储器编程	299
28.1	特性	299
28.2	功能说明	299
28.3	系统调用实现	300
28.4	阻塞和非阻塞的系统调用	300
28.4.1	执行系统调用	300
28.5	系统调用	301
28.5.1	芯片 ID	301
28.5.2	配置时钟	302
28.5.3	加载闪存字节	302
28.5.4	写入行	303
28.5.5	编程行	304
28.5.6	全部擦除	304
28.5.7	校验和	305
28.5.8	写保护	306
28.5.9	非阻塞写入行	306
28.5.10	非阻塞编程行	307
28.5.11	恢复非阻塞	308
28.6	系统调用状态	309

28.7 非阻塞系统调用伪代码.....	309
术语表	313
索引	329

节 A: 概述



本部分包括以下章节:

- 第 17 页上的简介章节
- 第 21 页上的入门章节
- 第 23 页上的文档结构章节

文档修订记录

版本	提交日期	变更者	变更描述
**	05/17/2016	XZNG	本文档版本号为 Rev**, 译自英文版 001-97952 Rev*A。

1. 简介



PSoC[®] 4 是基于 ARM[®] Cortex[®]-M0 CPU 的可编程嵌入式系统控制器。它集成了可编程模拟模块、可编程互联、用户可编程的数字逻辑、常用的固定功能外设以及高性能的 ARM Cortex-M0 子系统。PSoC 4200L 是 PSoC 4200 系列的增强版本，能够与多种更高性能的 PSoC 4 器件向上兼容。

PSoC 4 器件 具有以下特性：

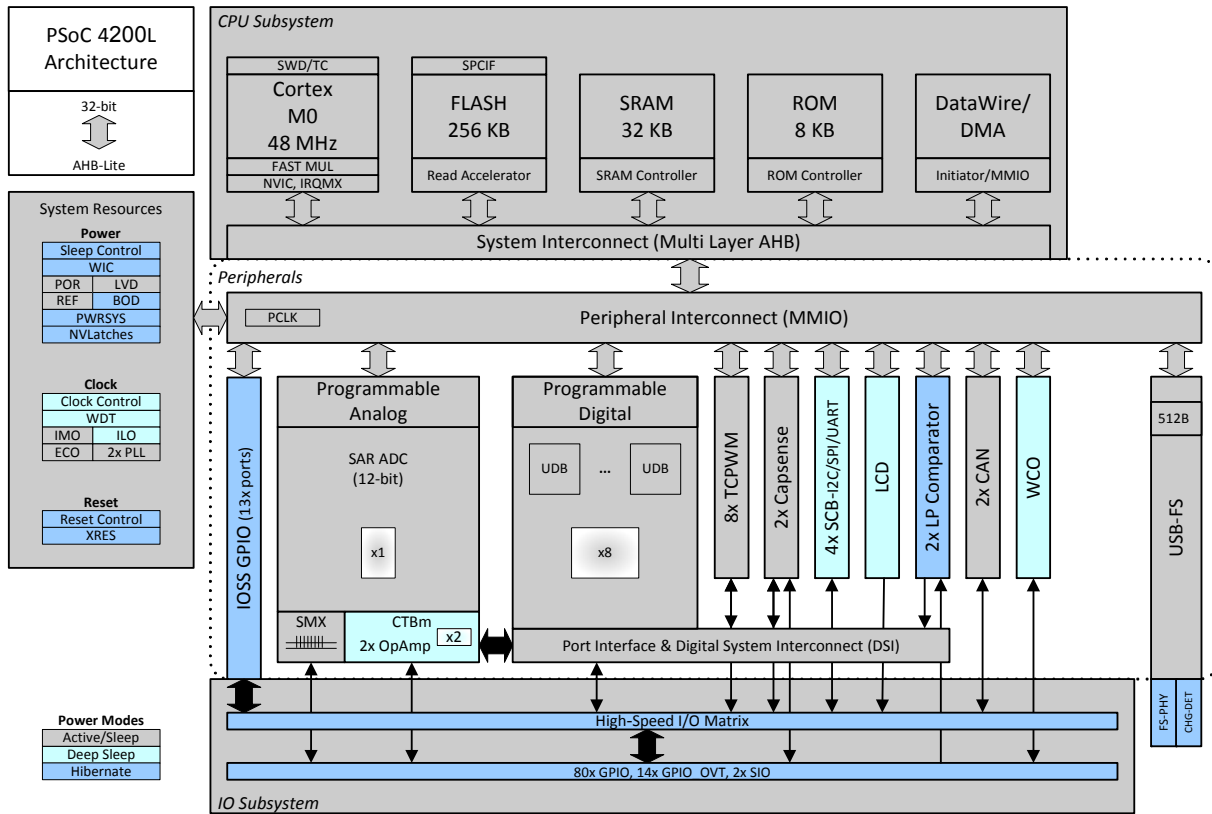
- 具有单周期乘法运算功能的高性能 32 位 Cortex-M0 CPU 内核
- 固定功能和可配置的数字模块
- 可编程数字逻辑
- 高性能模拟系统
- 灵活且可编程互联资源
- 全速 USB
- 支持电容式触摸感应（CapSense[®]）功能
- 低功耗操作模式包括：睡眠、深度睡眠、休眠和停止模式

本文档对 PSoC 4200L 器件中每个功能模块进行了详细介绍，从而帮助设计员创建系统级的设计。

1.1 系统架构

图 1-1 显示的是 PSoC 4200L 架构的主要组件。

图 1-1. PSoC 4200L 系列框图



1.2 特性

PSoC 4200L 系列具有以下主要组件：

- 支持单周期乘法功能的 32 位 Cortex-M0 CPU，运行频率 48 MHz 时，该 CPU 的计算性能高达 43 DMIPS
- 支持最大 256 KB 的闪存和 32 KB 的 SRAM
- 支持直接存储器访问（DMA）功能
- 八路支持可编程死区和互补输出的中心对齐脉冲宽度调制器（PWM）
- 12 位 SAR ADC（采样率为 1 Msps），提供了一个能够自动切换扫描多个通道的硬件定时器
- 多达四个运算放大器，用于转换模拟信号，并能配置为比较器
- 两个低功耗比较器
- 四个串行通信模块（SCB），可将其配置为 SPI、UART、I²C 和局部互连网络（LIN）从设备等各种串行通信接口
- 两个控制器区域网络（CAN）模块
- 多达八个可编程逻辑模块，即通用数字模块（UDB）
- CapSense
- SEGMENT LCD 直接驱动

- 支持多种低功耗操作模式：睡眠、深度睡眠、休眠和停止模式
- 支持使用 SWD 接口进行编程和调试
- 全面支持的 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 Cortex-M0 CPU 内核，它在 PSoC 4200L 中的工作频率高达 48 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。

CPU 还拥有 32 位的单周期硬件乘法器。

1.3.2 中断控制器

CPU 子系统包括一个带有 32 个中断输入的嵌套向量中断控制器（NVIC）和一个能够从深度睡眠模式唤醒处理器的唤醒中断控制器（WIC）。PSoC 4 的 Cortex-M0 CPU 具有一个不可屏蔽中断（NMI）输入，该输入与数字路由相连，可实现通用操作。

1.3.3 直接存储器访问

通过可编程描述符链，DMA 能够在存储器映射区域执行数据传输（外设到外设、外设到存储器或存储器到外设）。

1.4 存储器

PSoC 4 存储器子系统包括闪存和 SRAM。此外，还提供了拥有引导和配置子程序的监控 ROM。

1.4.1 闪存

PSoC 4 包含一个闪存模块，该模块的闪存加速器与 CPU 紧密耦合，以改善闪存模块的平均访问时间。通过闪存加速器，闪存的单周期访问时间平均为 SRAM 访问时间的 85%。

1.4.2 SRAM

PSoC 4 提供了能够在休眠模式中保持数据的 SRAM。

1.5 全局系统资源

1.5.1 时钟系统

PSoC 4 器件的时钟系统包括内部时钟（内部主振荡器（IMO）和内部低速振荡器（ILO）），以及一个外部时钟接口、外部晶体振荡器（ECO）和监视晶体振荡器（WCO）。

误差为 $\pm 2\%$ 的 IMO 是器件中主要的内部时钟源。IMO 的默认频率为 24 MHz；其可调整频率范围为 3 MHz 到 48 MHz，调整步长为 1 MHz。可以从主时钟频率生成多个派生时钟，以满足各种应用的要求。

ILO 是一个精度较低的低功耗振荡器，并作为 LFCLK 的时钟源使用，用以为外设的深度睡眠模式下运行时生成时钟。ILO 时钟频率为 32 kHz，精度为 $\pm 60\%$ 。

可以通过使用一个频率范围为 0 MHz 到 48 MHz 的外部时钟源（替换 IMO），为功能模块提供时钟。

通过使用外部晶体，ECO 可生成一个频率高达 33 MHz 的高精度时钟。另外，它还拥有两个锁相环（PLL），用于生成频率高达 48 MHz 的时钟。

WCO 作为 LFCLK 源使用。WCO 用于在深度睡眠模式下准确保持时间间隔。与 ILO 相同，WCO 在所有模式（休眠和停止模式除外）下均可用。

1.5.2 供电系统

PSoC 4 提供了多个电源域： V_{DDD} （给数字模块供电）， V_{DDA} （用于隔离模拟模块的噪声）以及 V_{DDIO} （给每一组 I/O 提供单独电压电平）。 V_{DDD} 和 V_{DDA} 可以在外部短接在一起，而 V_{DDIO} 可以单独供电。除了默认的活动模式外，PSoC 4 还能在四种低功耗模式（即睡眠、深度睡眠、休眠和停止模式）下运行。在活动模式下，CPU 处于运行状态，所有逻辑均被供电。在睡眠模式下，CPU 停止运行，所有其他外设仍正常工作。在深度睡眠模式下，CPU、SRAM 和高速度逻辑都被保持；主系统时钟关闭，低频时钟启用，并且低频外设仍然运行。在休眠模式下，低频时钟被关闭，低频外设停止运行。

系统中的各内部电压调节器可满足不同功耗模式的供电要求。

1.5.3 GPIO

PSoC 4 中的每个 GPIO 均具有以下特性：

- 八种驱动模式
- 能够单独控制禁用输入和输出
- 保持模式，用于锁存先前状态
- 可选的摆率
- 中断生成 — 边沿触发
- 支持 CapSense 和 LCD 驱动

PSoC 4200L 还具有 14 个过压容限引脚，这些引脚都满足 I2C 快速模式断电规范，以较低的 V_{DD} 运行时，该端口能够连接到更高的电压总线。

每个引脚都位于一个 8 位宽的端口中。多路信号通过内部高速 I/O 矩阵中的多路复用连接至一个 I/O 引脚。固定功能外设的引脚的位置是固定的。

PSoC 4200L 拥有两个特殊 I/O（SIO）。这些引脚允许进行热插拔、可配置输入阈值和输出开关电平。

1.6 可编程数字模块

PSoC 4200L 拥有多达八个 UDB。每个 UDB 均包含一个结构化的数据路径逻辑和一个互联灵活且不受限制的 PLD 逻辑。UDB 阵列提供了一个切换型路由结构，即数字信号互联（DSI）。通过 DSI，可以将信号从外设或端口路由到 UDB，也可以在各个 UDB 之间路由信号。

PSoC 4200L 中的 UDB 阵列用于设计自定义逻辑或额外定时器 / PWM 和各个通信接口，如 I²C、SPI、I2S 和 UART。

1.7 固定功能数字模块

1.7.1 定时器 / 计数器 / PWM 模块

定时器 / 计数器 / PWM 模块拥有八个用户可编程周期长度的 16 位计数器。可对这些计数器的功能进行同步化处理。每个模块都有捕获寄存器、周期寄存器和比较寄存器。该模块支持互补、死区可编程的输出。它还提供了用于强制输出进入未确定状态的非同步停止 (Kill) 输入。该模块的其他特性包括中心对齐 PWM、时钟预分频器、伪随机 PWM 和正交解码器。

1.7.2 串行通信模块

PSoC 4200L 器件拥有四个 SCB。每个 SCB 可以实现一个串行通信接口，如 I2C、UART、局部互联网络 (LIN) 从设备或 SPI。

每个 SCB 的特性包括：

- 支持标准的 I²C 多主设备和从设备功能
- 支持 Motorola、Texas Instruments 和 National (MicroWire) 模式的标准 SPI 主设备和从设备功能
- 具有标准的 UART 发送器和接收器功能，从而可以支持 SmartCard reader (ISO7816) (智能卡读卡器 (ISO7816))、IrDA 协议和 LIN
- 该 LIN 从设备符合 LIN 版本 1.3 和 LIN 版本 2.1/2.2 规范的标准
- 支持自带 32 字节缓冲区的 EzSPI 和 EzI²C 功能模式

1.7.3 控制器区域网络 (CAN)

PSoC 4200L 拥有两个 CAN 模块，它们支持 CAN 2.0A 和 2.0B。这些模块具有 16 个接收缓冲区，各自拥有一个信息滤波器，以及 8 个发送缓冲区。PHY 接口支持工业标准 Philips CAN PHY。

1.8 模拟系统

1.8.1 SAR ADC

PSoC 4200L 有一个可配置的 12 位 1 Msps SAR ADC。

ADC 提供了三个内部参考电压 (V_{DDA} 、 $V_{DDA}/2$ 和 V_{REF}) 和一个来自 GPIO 引脚的外部参考电压。可以使用 SAR 硬件定序器扫描多个通道，无需 CPU 的干预。

1.8.2 微型的连续时间模块 (CTBm)

CTBm 模块在模拟子系统的入口和出口处提供了连续时间功

能。CTBm 具有两个配置范围宽泛的高性能运算放大器以及一个切换路由矩阵。运算放大器也可以在比较器模式中运行。PSoC 4200L 有两个 CTBm 模块。

通过该模块，无需外部组件仍能够执行开环运算放大器、线性缓冲区和比较器。执行 PGA、电压缓冲区、滤波器和互阻抗放大器时，则需要使用外部组件。CTBm 模块可在活动、睡眠和深度睡眠模式下运行。

1.8.3 低功耗比较器

PSoC 4200L 有一对能在所有器件功耗模式下工作的低功耗比较器。这样，在低功耗模式下，当 CPU 和其他系统模块被禁用时，仍可以监控外部电压。两个输入电压可以均来自引脚，或一个来自引脚，另一个由内部信号通过 AMUXBUS 提供。

1.9 特殊的功能外设

1.9.1 LCD SEGMENT 驱动

PSoC 4200L 具有一个能够驱动八个 COMMON 的 LCD 控制器，此外，还可以配置每个 GPIO，使之驱动 COMMON 或 SEGMENT。该控制器使用完整的数字方法 (数字关联和 PWM) 驱动 LCD SEGMENT，而不需要生成内部 LCD 电压。

1.9.2 CapSense

PSoC 4 器件支持 CapSense 功能，通过该功能您能够使用手指的电容属性来完成对按键、滑条和滚轮的触摸操作。PSoC 4 通过 CapSense Sigma-Delta (CSD) 模块，使所有 GPIO 引脚都支持 CapSense 功能。CSD 还提供了防水功能。PSoC 4200L 器件有两个这种 CapSense 模块。

1.9.2.1 IDAC 和比较器

CapSense 模块包含两个 IDAC 和一个比较器，其参考电压为 12 V；在不使用 CapSense 的情况下，它们可用于通用目的。PSoC 4200L 具有四个 IDAC 和两个比较器，可用于通用目的。这些模块都是两个 CapSense 模块的一部分。

1.10 编程和调试

PSoC 4 器件支持通过片上 SWD 接口对器件进行编程和调试操作。PSoC Creator 集成开发环境 (IDE) 提供了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

可以访问 www.cypress.com/psoc4 以获取 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、知识库以及应用支持工程师。

有关应用支持，请访问 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。可以从分销商所提供的 DVD-ROM 中找到升级版本，也可以直接从 www.cypress.com/psoccreator 上下载升级版本。“文档”章节中也提供系统文档的重要更新。

2.3 开发套件

开发套件可由 Digi-Key、Avnet、Arrow 以及 Future 公司提供。赛普拉斯在线商店拥有成功开发 PSoC 项目所需的所有开发套件、C 编译器和附件。请访问赛普拉斯的在线商店 www.cypress.com/cypress-store。在**产品**目录下面，通过点击 **Programmable System-on-Chip** 可以查找当前可用的器件系列。

2.4 应用笔记

更多有关 PSoC 4 器件功能的信息以及如何使用 PSoC Creator 和 PSoC 4 开发套件以快速创建一个简单的 PSoC 应用，请参考应用笔记 [AN79953 — PSoC 4 入门](#)。

3. 文档结构



本文档包括以下部分：

- 第 27 页上的节 B：CPU 系统
- 第 61 页上的节 C：系统范围资源
- 第 111 页上的节 D：数字系统
- 第 211 页上的节 E：模拟系统
- 第 289 页上的节 F：编程和调试

3.1 主要部分

为了便于使用，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区分。

- 部分 — 介绍了系统架构，以及如何开始使用以及有关特定领域的约定概念和概括信息，以告知读者产品的架构。
- 章节 — 介绍了有关部分主题的某一领域的特定章节。它们是针对集成电路的某些方面的详细实现和使用信息。
- 术语表 — 定义了本技术参考手册（TRM）中使用的专业术语。术语表使用粗斜体字来显示。
- 寄存器技术参考手册 — 提供了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

除了标题中使用的字体类型以外，本文档只使用四个明显的字体类型。

- 第一类型是斜体字，在谈及某些文档标题或文件名称时使用。
- 第二类型是粗斜体字，在谈及本文中术语表所描述的某些术语时使用。
- 第三类型是 Times New Roman 字体，用以区分公式的例子。
- 第四类型是 Courier New 字体，用以区分代码的例子。

3.2.1 寄存器规定

有关寄存器规定的详细信息，请参见 [PSoC 4200L 系列：PSoC 4 寄存器技术参考手册](#)。

3.2.2 数字命名

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’），此外，还可以使用 ‘0x’ 前缀来表示十六进制数字（根据 C 代码规范）。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

缩略词	测量单位
bps	每秒位数
°C	摄氏度
dB	分贝
fF	飞法
Hz	赫兹
k	千, 1000
K	千, 2 ¹⁰
KB	1024 个字节或大约一千个字节
Kbit	1024 位
kHz	千赫 (32.000)
kΩ	千欧
MHz	兆赫兹
MΩ	兆欧
μA	微安
μF	微法
μs	微秒
μV	微伏
μVrms	微伏均方根
mA	毫安
ms	毫秒
mV	毫伏
nA	纳安
ns	纳秒
nV	纳伏
Ω	欧姆
pF	皮法
pp	峰峰值
ppm	百万分率
SPS	每秒样本数
σ	sigma: 一个标准差
V	伏特

3.2.4 缩略语

下表列出了本文档中使用的缩略语

表 3-2. 缩略语

缩略语	定义
ABUS	模拟输出总线
AC	交流
ADC	模数转换器
AHB	AMBA (高级微控制器总线结构) 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
BOD	欠压检测
BOM	物料表
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
CSD	电容触摸感应三角积分模块
CT	连续时间
CTB	连续时间模块
CTBm	微型连续时间模块
DAC	数模转换器
DAP	调试访问端口
DC	直流
DI	数字或数据输入
DMA	直接存储器访问
DNL	微分非线性
DO	数字或数据输出
DSI	数字信号接口
DSM	深度睡眠模式
DW	数据线
ECO	外部晶体振荡器
EEPROM	电可擦除可编程只读存储器
EMIF	外部存储器接口
FB	反馈
FIFO	先进先出
FSR	全量程范围
GPIO	通用输入 / 输出
HCI	主控制器的接口
HFCLK	高频率时钟

表 3-2. 缩略语 (续)

缩略语	定义
HSIOM	高速输入 / 输出矩阵
I ² C	两线式串行总线
IDE	集成开发环境
ILO	内部低速振荡器
ITO	铟锡氧化物
IMO	内部主振荡器
INL	积分非线性
I/O	输入 / 输出
IOR	I/O 读取
IOW	I/O 写入
IRES	初次上电复位
IRA	中断请求确认
IRQ	中断请求
ISR	中断服务子程序
IVR	中断向量读取
LCD	液晶显示屏
LFCLK	低频率时钟
LIN	局部互联网络
LPCOMP	低功耗比较器
LRb	最后接收位
LRB	最后接收字节
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
NMI	不可屏蔽中断
NVIC	嵌套向量中断控制器
PC	程序计数器
PCB	印刷电路板
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	上电复位
PPOR	精确上电复位
PRS	伪随机序列
PRGIO	可编程输入 / 输出
PSoC®	可编程片上系统

表 3-2. 缩略语 (续)

缩略语	定义
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	中断返回
RF	射频
ROM	只读存储器
RMS	均方根
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	特殊输入 / 输出
SE0	单端零
SNR	信噪比
SOF	帧头
SOI	指令起始
SP	堆栈指针
SPD	连续相位检测器
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	单斜模数转换器
SSC	管理系统调用
SYSCLK	系统时钟
SWD	串行线调试
TC	终端计数
TCPWM	定时器、计数器、脉冲宽度调制器
TD	事务数据操作描述符
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
WCO	监视晶体振荡器
WDT	看门狗定时器
WDR	看门狗复位
XRES	外部复位
XRES_N	外部复位，低电平有效

节 B: CPU 系统

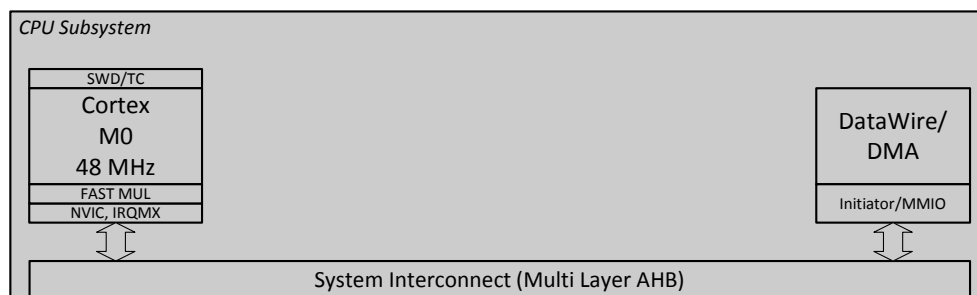


本章节包括下面小节：

- 第 29 页上的 [Cortex-M0 CPU 章节](#)
- 第 35 页上的 [DMA 控制模式章节](#)
- 第 51 页上的 [中断章节](#)

系统架构

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 32 位乘法指令和低延迟的中断处理程序。紧密与 CPU 内核相连的其他子系统包括：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，请参见 www.arm.com 网站中提供的 ARM Cortex-M0 用户指南或技术参考手册。

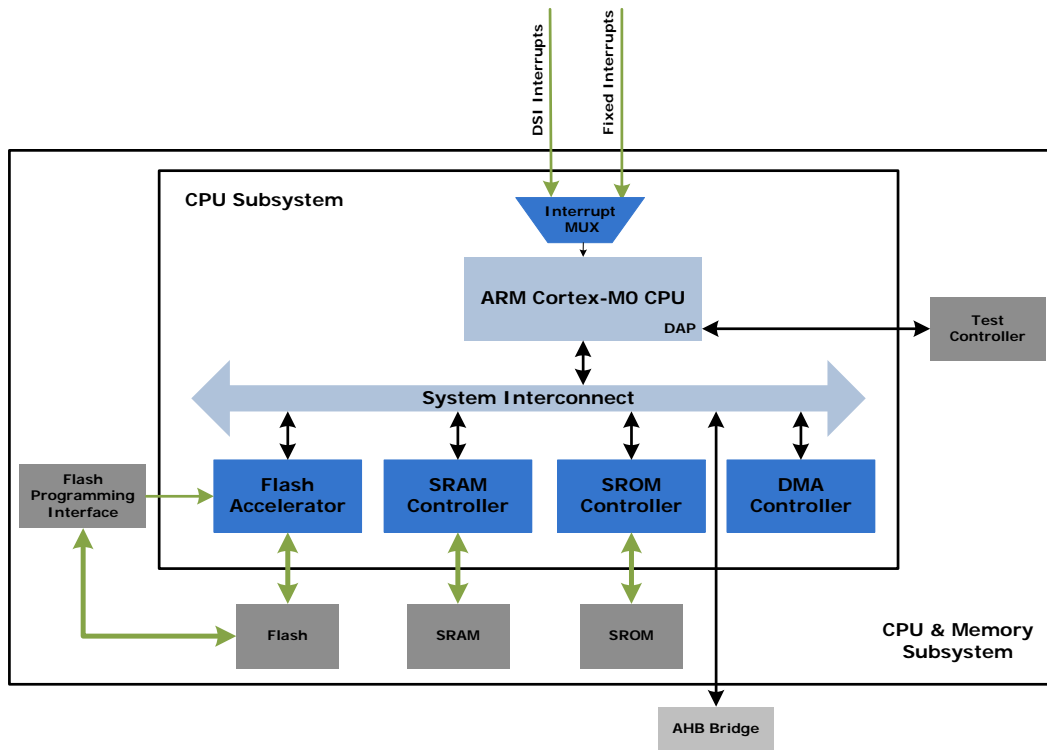
4.1 特性

PSoC 4 Cortex-M0 具有以下特性：

- 易于使用、编程和调试，因此可以从 8 位和 16 位处理器进行更简单的移植
- 工作速度可达 0.9 DMIPS/MHz；这样允许增大执行速度或降低功耗
- 支持 Thumb 指令集，这样可以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：
 - 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 位的处理器，它包含一个 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（请参见 [第 32 页上的工作模式](#)）。它有一个单周期的 32 位乘法指令。

4.4 地址映射

ARM Cortex-M0 具有固定的地址映射，因此可以通过简单的存储器访问指令访问存储器和外设。32 位（4 GB）地址空间分为多个区域，如 [表 4-1](#) 所示。请注意，可从代码区和 SRAM 区执行代码。

表 4-1. Cortex-M0 地址映射情况

地址范围	名称	使用说明
0x00000000 – 0x1FFFFFFF	代码	编程代码区域。您也可以将数据存储在此处。它包括从地址 0 开始的异常向量表。
0x20000000 – 0x3FFFFFFF	SRAM	数据区域。您也可以执行该区域中的代码。
0x40000000 – 0x5FFFFFFF	外设	所有的外设寄存器。您不能执行该区域中的代码。
0x60000000 – 0xDFFFFFFF		未使用。
0xE0000000 – 0xE00FFFFF	PPB	CPU 内核的外设寄存器。
0xE0100000 – 0xFFFFFFFF	器件	PSoC 4 的专属空间

4.5 寄存器

Cortex-M0 具有 16 个 32 位寄存器，如表 4-2 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）寄存器指出了将要使用的堆栈指针：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。通过写入到该寄存器内，可以控制程序流。

表 4-2. Cortex-M0 寄存器

名称	类型 ^a	复位值	说明
R0-R12	RW	未定义	R0-R12 是用于数据操作的 32 位通用寄存器。
MSP (R13) PSP (R13)	RW	[0x00000000]	堆栈指针（SP）寄存器是寄存器 R13。在线程模式下，CONTROL 寄存器的位 [1] 指出了所使用的堆栈指针： 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 (R14)	RW	未定义	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 (R15)	RW	[0x00000004]	程序计数器（PC）是寄存器 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 内。复位时，该值的位 [0] 必须为 1 并被加载到 EPSR T 位内。
PSR	RW	未定义	程序状态寄存器（PSR）组合： 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	未定义	APSR 包含先前指令执行程序中条件标志的当前状态。
EPSR	RO	[0x00000004].0	复位时，会将寄存器 [0x00000004] 中的位 [0] 加载到 EPSR。
IPSR	RO	0	IPSR 包含当前 ISR 的异常编号。
PRIMASK	RW	0	PRIMASK 寄存器用于防止激活所有可配置优先级的异常情况。
CONTROL	RW	0	CONTROL 寄存器用于控制处理器在线程模式下所使用的堆栈。

a. 说明程序在线程模式和处理模式下运行时的访问类型。调试访问会有所不同。

表 4-3 显示的是 PSR 位的分配方式。

表 4-3. Cortex-M0 PSR 位分配

位	PSR 寄存器	名称	使用情况
31	APSR	N	负向标志
30	APSR	Z	零标志
29	APSR	C	进位标志或借位标志
28	APSR	V	溢流标志

表 4-3. Cortex-M0 PSR 位分配

位	PSR 寄存器	名称	使用情况
27 – 25	–	–	保留
24	EPSR	T	Thumb 状态位。T 位必须为 1，如果该位为 0，则尝试执行指令会导致 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 = 保留 11 = SVCall 12、13 = 保留 14 = PendSV 15 = SysTick 16 = IRQ0 ... 47 = IRQ31

通过使用 MSR 或 CPS 指令，可以设置或清除 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（HardFault、NMI 和复位除外）。第 51 页上的中断章节显示的是异常列表。

4.6 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 用于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。在该模式下，始终使用 MSP。

在线程模式下，通过使用 MSR 指令，可以设置 CONTROL（控制）寄存器中的堆栈指针位。修改堆栈指针时，使用 MSR 指令后可立即使用 ISB 指令。这样就能确保 ISB 指令后的指令使用新的堆栈指针执行。

在处理程序模式下，对 CONTROL（控制）寄存器进行写操作无效，这是因为 MSP 一直被使用。异常进入和返回机制会自动更新 CONTROL（控制）寄存器。

4.7 指令集

Cortex-M0 执行 Thumb 指令集的一个版本，如表 4-4 中所示。有关详细内容，请参见 Cortex-M0 通用用户指南。

指令操作数可以是一个 ARM 寄存器、一个常量，或者是另一个指令特定参数。在操作数上执行指令，并经常将该结果存储在目标寄存器内。并非所有指令都能将 PC 或 SP 作为操作数或目标寄存器使用。

表 4-4. Thumb 指令集

助记符	简要说明
ADCS	进位加
ADD{S} ^a	添加
ADR	PC 相对地址到寄存器
ANDS	按位进行逻辑与（AND）运算
ASRS	算术右移
B{cc}	分支 { 条件 }
BICS	位清除
BKPT	断点
BL	分支链接
BLX	分支链接
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
EORS	逻辑异或
ISB	指令同步屏障
LDM	加载多个寄存器，并且每次加载后都会递增数据
LDR	从 PC 相对地址加载寄存器
LDRB	将字加载到寄存器内
LDRH	将半字加载到寄存器内
LDRSB	将带符号的字节加载到寄存器内
LDRSH	将带符号的半字加载到寄存器内
LSLS	逻辑左移
LSRS	逻辑右移
MOV{S} ^a	移动
MRS	从特殊寄存器移动到通用寄存器
MSR	从通用寄存器移动到特殊寄存器
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	将寄存器推入堆栈内
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

表 4-4. Thumb 指令集

助记符	简要说明
SBCS	进位减
SEV	发送事件
STM	存储多个寄存器，并且每次存储后递增地址
STR	存储寄存器的一个字
STRB	存储寄存器的一个字节
STRH	存储寄存器的半字
SUB{S} ^a	减法
SVC	管理程序调用
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

a. ‘S’ 限定符会使 ADD、SUB 或 MOV 指令更新 APSR 条件标志。

4.7.1 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，半字对齐地址用于半字访问。字节访问始终是对齐的。

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。

4.7.2 存储器中的字节顺序

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。

4.8 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。该定时器带有一个重载寄存器，其中可将 24 位作为一个倒计数值使用。SysTick 定时器将 Cortex-M0 内部时钟作为源时钟使用。

4.9 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. DMA 控制模式



DMA 控制器提供了数据线（DW）和直接存储器访问（DMA）功能。DMA 控制器具有以下特性：

- 支持多达 32 个 DMA 通道；对于某个特定器件，请查询器件数据手册，以确定它支持的通道数量
- 每个通道支持四个优先级
- 支持字节、半字（2 字节）和字（4 字节）传输
- 每个通道支持三种工作模式
- 可配置中断生成
- 传输完成触发信号输出
- 最大传输数据元素为 65,536

DMA 控制器支持三种工作模式。这些工作模式在 DMA 控制器在单个触发器信号上运行的方式方面存在差别。通过这些工作模式，用户可以实现不同的 DMA 工作模式。这些工作模式分别为：

- 模式 0：每个触发传输单数据元素
- 模式 1：每个触发传输所有数据元素
- 模式 2：每个触发传输所有数据元素，并自动触发链式描述符

数据传输的各项特性（如源地址位置、目标地址位置以及传输大小）都是由描述符结构指定的。每个通道具有独立的描述符结构。

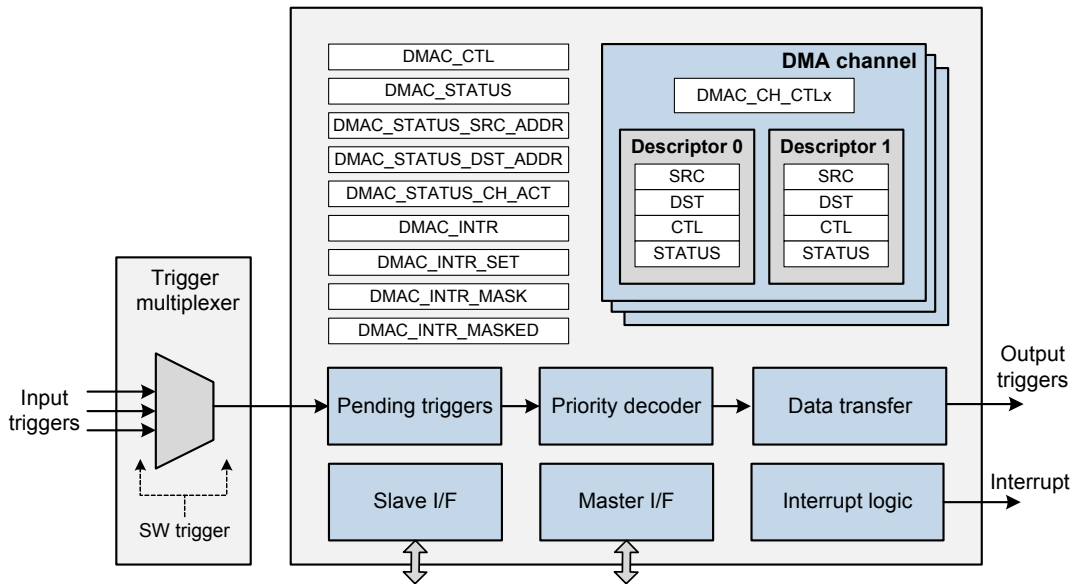
DMA 控制器支持活动 / 睡眠模式，但不支持深度睡眠和休眠功耗模式。

5.1 框图说明

DMA 组件使数据能够在存储器、外设和寄存器之间相互传输。这些传输不受 CPU 的控制。DMA 每次可传输多达 65,536 个数据元素。这些数据元素带宽可为 8 位、16 位或 32 位。通过来自 DMA 通道（包括 DMA 本身）、其他 DMA 通道、外设或 CPU 的外部触发器，DMA 可启动某个数据传输。DMA 的最大优点是可以减轻 CPU 的数据传输负担。

图 5-1 显示的是 DMA 控制器的模块级别概述。

图 5-1. DMA 控制器模块框图



每个 DMA 通道具有两个描述符，用于配置传输专用的各种参数（如源地址、目标地址和数据宽度）。DMA 通道通过触发事件启动传输。触发信号可来自器件中不同的外设（包括 DMA 本身）。

DMA 控制器具有两个总线接口：主设备接口和从设备接口。主设备 I/F 是一个 AHB-Lite 总线主设备，它允许 DMA 控制器将 AHB-Lite 数据传输到源位置和目标位置。DMA 是主设备接口中的总线主设备。通过该接口，可完成所有 DMA 传输。

通过从设备接口，可访问和重新配置 DMA 配置寄存器和描述符。从设备 I/F 是 AHB-Lite 总线从设备，它允许 PSoC 主 CPU 访问 DMA 控制器的控制 / 状态寄存器和描述符结构。一般情况下，CPU 是该总线的主设备。

接收某个触发器会激活 DMA 控制器中的状态机，经过特定的触发程序和处理过程后，它会根据描述符设置来启动数据传输操作。传输完成后会生成一个输出触发，可将它作为启动其他功能的触发条件或事件。

DMA 控制器中还包含一个中断逻辑模块。DMA 控制器中只有一个用于中断 CPU 的中断线。通过配置各个单独的 DMA 描述符，可在传输完成时激活该中断线。

5.1.1 触发源和复用

每个 DMA 通道都有一个与其相关的输入和输出触发。其中，输入触发可以来自任何外设、CPU 或 DMA 通道本身。根据传输模式小节的内容，可以使用输入触发器来触发 DMA 传输。触发器输入处于‘逻辑高电平’状态时，它将触发 DMA 通道该‘逻辑高电平’的最小宽度为两个系统时钟周期。取消激活设置配置了触发器取消激活的性质。

传输操作完成时，触发器输出会发出确认信号。可以将该信号作为 DMA 通道的触发或数字互连的数字信号。触发输入可以来自不同的源，并通过触发复用器路由。

5.1.1.1 触发复用器

DMA 通道的触发输入可来自 PSoC 的不同外设源。通过触发复用器，可将 DMA 通路由到单独的 DMA 通道的触发输入端。另外，也可以通过该触发复用器使 DMA 的输出触发路由到 PSoC 的某些外设上。

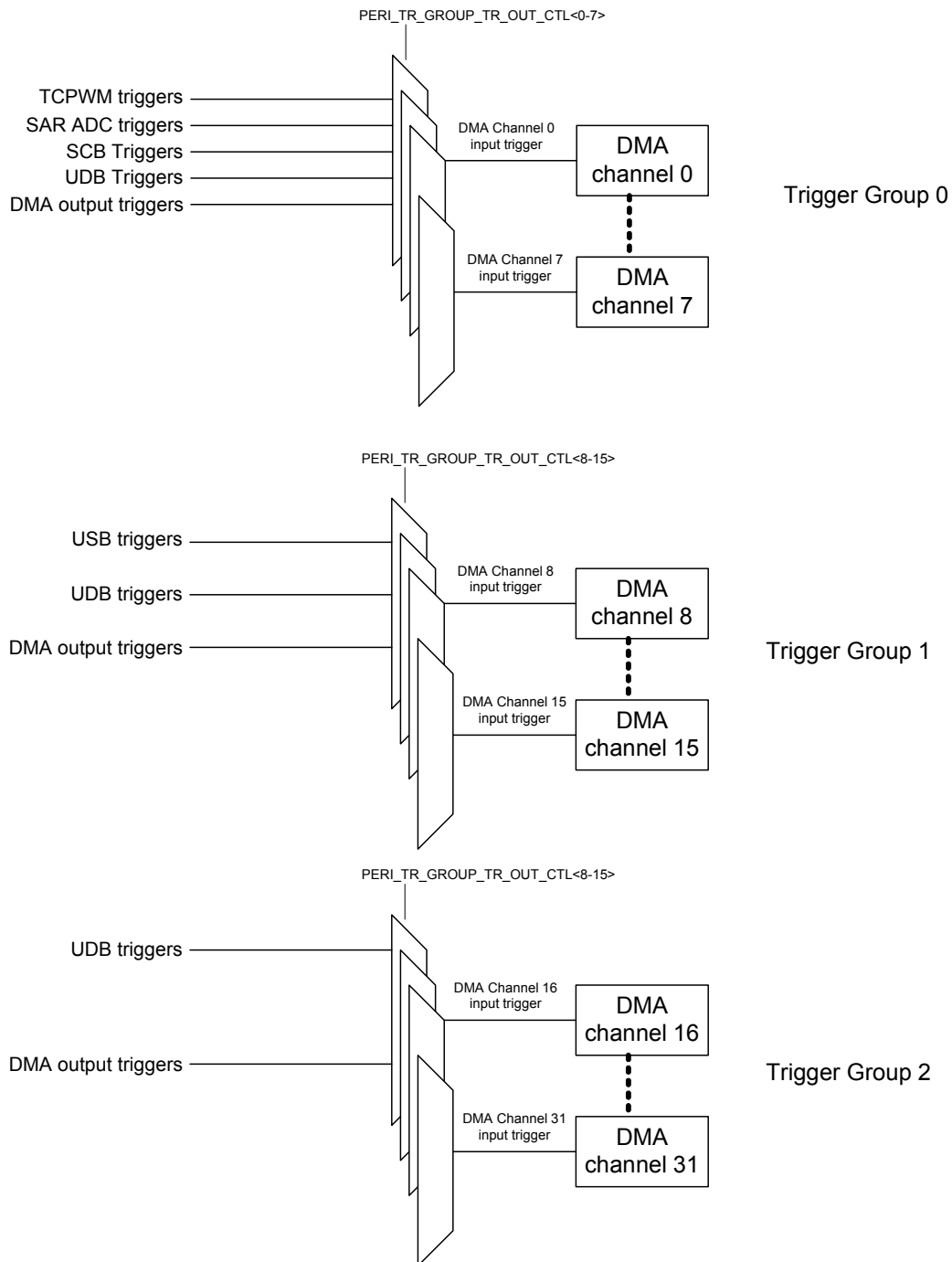
在 DMA 触发器中，复用器被分为各个触发组。每个触发组由多个复用器组成，用于将数据写入到各个 DMA 通道的触发器输入端。

PSoC 4200L 实现以下四个触发组：

- 触发组 0 复用了通用外设，如 TCPWM、SCB、UDB 和 DMA 触发源。该触发组只为 DMA 通道 0 到 7 提供触发输入。
- 触发组 1 将 USB 和 UDB 触发复用到 DMA 通道 8 到 15。
- 触发组 2 将 16 个 UDB 触发复用到 DMA 通道 16 到 31。
- 触发组 3 实现各复用操作是为了将 DMA 触发输出路由返回 USB 突发的终端信号。

图 5-2 显示的是触发组 0 到 2 的详图。

图 5-2. 触发组 0 到 2



不同 DMA 通道的输入触发器被分为四种不同的触发组。每一组都有符合复用器的触发源选项组。通过设置 PERI_TR_GROUP_TR_OUT_CTLx[5:0] 寄存器，可以为某个特定的触发组选择触发源。请参考表 5-1，了解各触发组不同源的配置信息。

表 5-1. 触发组和触发源

PERI_TR_GROUP_TR_OUT_CTLx[5:0]	触发组 0 (触发源)	触发组 1 (触发源)	触发组 2 (触发源)
0	软件触发被固定连接到逻辑 0	软件触发被固定连接到逻辑 0	软件触发被固定连接到逻辑 0
1	TCPWM 0 溢出	USB 触发 0	UDB 触发 16
2	TCPWM 1 溢出	USB 触发 1	UDB 触发 17
3	TCPWM 2 溢出	USB 触发 2	UDB 触发 18
4	TCPWM 3 溢出	USB 触发 3	UDB 触发 19
5	TCPWM 4 溢出	USB 触发 4	UDB 触发 20
6	TCPWM 5 溢出	USB 触发 5	UDB 触发 21
7	TCPWM 6 溢出	USB 触发 6	UDB 触发 22
8	TCPWM 7 溢出	USB 触发 7	UDB 触发 23
9	TCPWM 0 比较	UDB 触发 8	UDB 触发 24
10	TCPWM 1 比较	UDB 触发 9	UDB 触发 25
11	TCPWM 2 比较	UDB 触发 10	UDB 触发 26
12	TCPWM 3 比较	UDB 触发 11	UDB 触发 27
13	TCPWM 4 比较	UDB 触发 12	UDB 触发 28
14	TCPWM 5 比较	UDB 触发 13	UDB 触发 29
15	TCPWM 6 比较	UDB 触发 14	UDB 触发 30
16	TCPWM 7 比较	UDB 触发 15	UDB 触发 31
17	TCPWM 0 下溢	DMA 通道 8 触发	DMA 通道 7 触发
18	TCPWM 1 下溢	DMA 通道 9 触发	
19	TCPWM 2 下溢	DMA 通道 10 触发	
20	TCPWM 3 下溢	DMA 通道 11 触发	
21	TCPWM 4 下溢	DMA 通道 12 触发	
22	TCPWM 5 下溢	DMA 通道 13 触发	
23	TCPWM 6 下溢	DMA 通道 14 触发	
24	TCPWM 7 下溢	DMA 通道 15 触发	
25	SAR ADC EOC		
26	SCB0 TX		
27	SCB0 RX		
28	SCB1 TX		
29	SCB1 RX		
30	SCB2 TX		
31	SCB2 RX		
32	SCB3 TX		
33	SCB3 RX		
34	UDB DSI 请求 0		
35	UDB DSI 请求 1		
36	DMA 通道 0 触发		
37	DMA 通道 1 触发		
38	DMA 通道 2 触发		
39	DMA 通道 3 触发		
40	DMA 通道 4 触发		
41	DMA 通道 5 触发		
42	DMA 通道 6 触发		

表 5-1. 触发组和触发源（续）

PERI_TR_GROUP_TR_OUT_CTLx[5:0]	触发组 0（触发源）	触发组 1（触发源）	触发组 2（触发源）
43	DMA 通道 7 触发		
44	UDB 触发 0		
45	UDB 触发 1		
46	UDB 触发 2		
47	UDB 触发 3		
48	UDB 触发 4		
49	UDB 触发 5		
50	UDB 触发 6		
51	UDB 触发 7		

触发组 3 用于将 DMA 输出触发路由回 USB 模块，从而为 USB 提供突发终端信号。更多有关触发组 3 的配置信息，请参考第 171 页上的通用串行总线（USB）章节。

5.1.1.2 创建软件触发

每个 DMA 通道都有一个与其相关的触发输入和输出。该触发输入可以来自任何一个触发组，如第 36 页上的触发复用器一节所述。通过使用触发复用器设置中的触发输入选项 0，可以实现 DMA 通道的软件触发。当

PERI_TR_GROUP_TR_OUT_CTLx[5:0] 为 0 时，DMA 触发器被配置为软件触发器。然后，使用 PERI_TR_CTL 寄存器触发 DMA 通道。

5.1.2 挂起触发

如果在 DMA 通道工作时遇到触发事件，那么对应该触发的 DMA 通道将进入挂起状态。挂起触发将激活触发器局部存储在挂起位内，从而实现对它们的记录。该操作非常必要，因为可同时触发多个通道触发器，但每一次只能通过一个通道进行数据传输。通过该模块，可以使用电平敏感触发和沿触发。

挂起触发器被寄存在状态寄存器（DMAC_STATUS_CH_ACT）内。

5.1.3 输出触发

每个通道都有一个触发输出。该触发在两个系统时钟周期内处于高电平状态。在数据传输完成时将生成该触发。在系统级中，可将这些输出触发器连接到触发复用器组件上。通过该连接，可以将 DMA 控制器的输出触发连接到 DMA 控制器的输入触发上。换句话说，可通过完成单通道中的某个传输操作来激活其他通道，甚至重新激活同一通道。

请注意，DMA 输出触发器也被连接到数字系统互联（DSI），并且一些 DSI 信号被连接到触发器复用器输入。

5.1.4 通道优先级

如果有效的触发器使用了多个通道，那么可以根据通道优先级来确定可以访问数据传输引擎的通道。可以使用通道控制寄存器（DMAC_CH_CTL）的 PRIO 字段来设置每个通道的优先级，其中 ‘0’ 表示最高优先级，‘3’ 表示最低优先级。优先级解码器根据通道优先级来确定优先级最高的有效通道。如果多个有效通道的优先级都为最高，则带有最低索引 ‘i’ 的通道被视为优先级最高的有效通道。

5.1.5 数据传输引擎

数据传输引擎用于将数据从源位置传输到目标位置。闲置时，数据传输引擎可以接收优先级最高的有效通道。该数据传输的配置由描述符指定。数据传输引擎实现了一个拥有以下状态的状态机。

- 状态 0 — 默认状态：这是 DMA 控制器的闲置状态。在该状态中，控制器需要等待触发条件，从而才能进行传输。
- 状态 1 — 负载描述符：在发生触发条件并确定好优先级时，数据传输引擎将进入负载描述符状态。在该状态中，有效描述符（SRC、DST 和 CTL）将被加载到 DMA 控制器内，以开始进行传输。DMAC_STATUS、DMAC_STATUS_SRC_ADDR、DMAC_STATUS_DST_ADDR 以及 STATUS_CH_ACT 反映的是当前的有效状态。
- 状态 2 — 加载源位置的数据：数据传输引擎通过主设备 I/F 来加载源位置的数据。
- 状态 3 — 将数据存储在目标位置：数据传输引擎通过主设备 I/F 将数据存储在目标位置。
根据传输模式，可以多次执行状态 2 和 3。
- 状态 4 — 存储描述符：数据传输引擎通过更新通道的描述符结构来反映数据传输，并将其存储在描述符状态寄存器内。

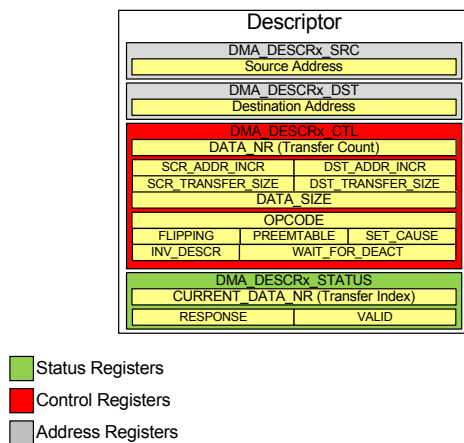
- 状态 5 — 等待触发器取消激活状态：如果触发器取消激活状态需要两个周期，那么在触发器保持有效状态两个周期后便满足该条件。如果该状态被设置为 ‘wait indefinitely’，则 DMA 控制器将保持在该状态下，直到触发信号进入低电平状态为止。
- 状态 6 — 存储描述符响应：在该相位中，将完成根据描述符的数据传输，并且生成中断（如果该中断被配置，已在该条件中生成）。DMAC_DESCR_PING_STATUS 或 DMAC_DESCR_PONG_STATUS 寄存器中的响应字段将被填充，并且其状态返回到状态 0。

5.2 描述符

通道中源地址和目标地址间的数据传输是由描述符配置的。DMA 中的每个通道都有两个称为 PING 和 PONG 的描述符（本文档将它们分别称为描述符 0 和描述符 1）。单个描述符指的是 4 个 32 位寄存器的组合，它包含相关通道的传输配置。

图 5-3 显示的是描述符的结构。

图 5-3. 描述符结构



5.2.1 地址配置

图 5-4 演示了某个传输的地址配置的描述符设置的使用情况。

Source and Destination Address: 源地址和目标地址在描述符中的相应寄存器内设置。它们为该传输的源位置和目标位置设置了基地址。通过配置描述符来传输单个元素时，该字段将保存该数据元素的源 / 目标地址。如果描述符在递增模式下使用源地址和 / 或目标地址来传输多个元素，该字段将保存被传输的第一个元素的地址。

Data Number (DATA_NR): 这是一个传输计数参数。DATA_NR 是一个 16 位的编号，它确定了在一个描述符完成前将要传输的元素数量。在典型的使用中，它设置的是传输的缓冲器大小。

Source Address Increment (SCR_ADDR_INCR): 这是在控制寄存器中设置的位，它确定了源地址是否在每个数据元素传输之间递增。当数据源为缓冲器并且需要从存储器中后续位置提取传输元素时，需要使能该特性。在这种情况下，源地址寄存器只设置基地址，并且该基地址上的后续传输均是

递增的。地址递增的大小要根据第 41 页上的 5.2.2 传输大小章节中 SCR_TRANSFER_SIZE 设置确定。

Destination Address Increment (DST_ADDR_INCR): 这是在控制寄存器中设置的位，通过它可以确定目标地址是否在每个元素传输之间递增。当数据目标为缓冲器，并且需要将每个传输元素传输到存储器中后续位置时，需要使能该特性。在这种情况下，目标地址寄存器只设置基地址，并且该基地址上的后续传输将递增。地址递增的大小要根据第 41 页上的 5.2.2 传输大小章节中的 DST_TRANSFER_SIZE 设置确定。

Invalidate Descriptor (INV_DESCR): 设置该位时，描述符将传输所有数据元素，并清除描述符中的 VALID 位，从而使其无效。该特性会对 DMA_DESCRx_STATUS 寄存器中的 VALID 位产生影响。如果用户期待描述符在传输完成后无效，可以使用该设置。通过设置描述符的 STATUS 寄存器中的 VALID 位，可以使固件中的描述符返回到有效状态。

Preemptable (PREEMPTABLE): 如果禁用该位，工作模式所定义的当前传输则能够完成而不受干扰。如果使能该位，优先级更高的 DMA 通道可以阻止 / 中断工作模式所定义的当前传输。当阻止该通道时，该位将被挂起并在下次它的优先级最高时运行。

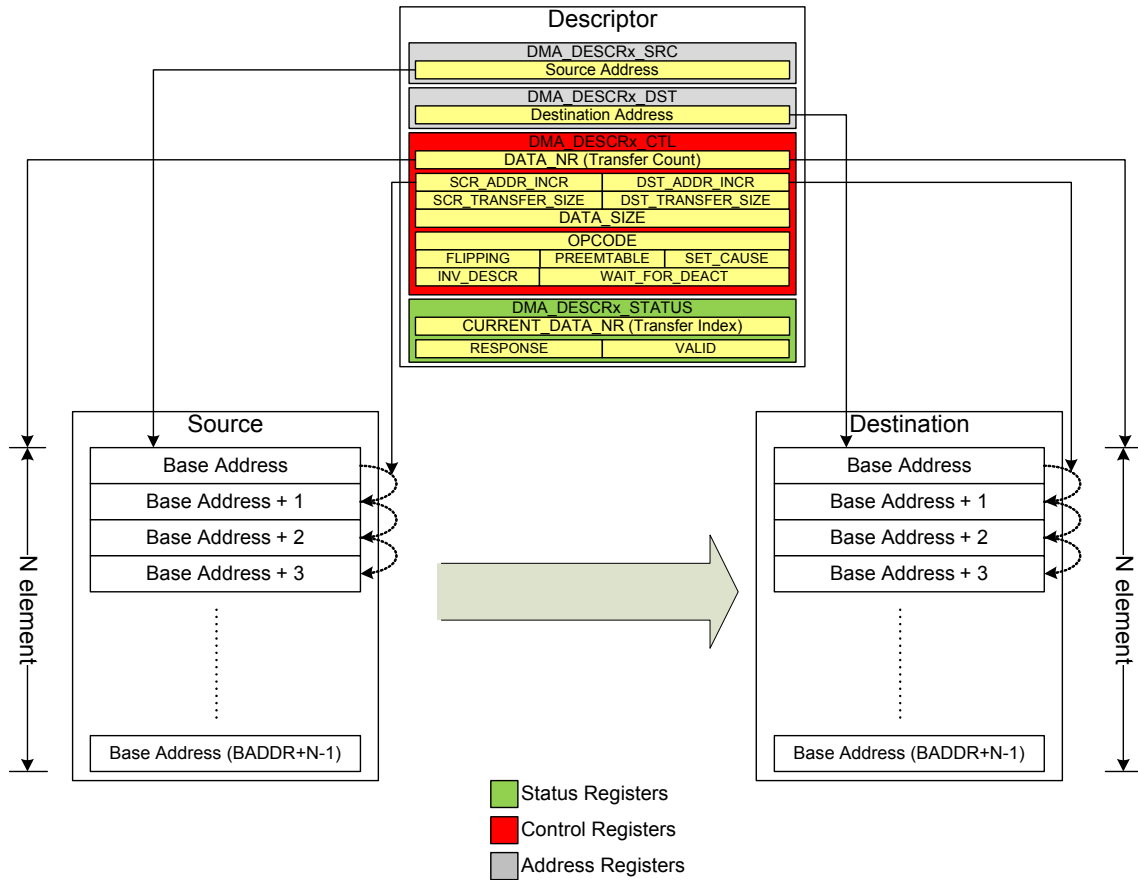
Setting Interrupt Cause (SET_CAUSE): 当描述符完成传输所有数据元素时，它将生成一个中断请求。该中断请求被所有 DMA 通道共享。设置该位会使与之相应的通道变成中断源。

Trigger Type (WAIT_FOR_DEACT): 基于描述符的 DMA 传输完成时，数据传输引擎会检测触发器是否取消激活状态。该步骤与数据传输引擎的状态 5 相对应。请参见第 39 页上的 5.1.5 数据传输引擎。根据 DMA 输入触发的类型，可以确定触发信号被取消激活的时间。触发器被激活时，DMA 传输将被激活，但这种传输只能在触发器处于取消激活状态后完成。可以使用该字段将控制器中数据的传输与生成该触发的源同步。

只有在完成一个描述符执行后使用该字段，它具有以下的四种设置值：

- 0 — 脉冲触发：无需等待取消激活。
- 1 — 电平敏感触发器等待四个 SYSCLK 周期：在四个周期内检测到电平触发器信号后，将禁用 DMA 触发。
- 2 — 电平敏感触发器等待八个 SYSCLK 周期：在八个周期内检测到电平触发器信号后，将启动 DMA 传输。
- 3 — 脉冲触发器将无限期待，直到被取消激活为止。取消激活触发器信号后，将启动 DMA 传输。

图 5-4. DMA 传输：地址配置



5.2.2 传输大小

通过使用描述符中的传输 / 数据大小参数，可以为某个传输配置传输字宽度。这些设置被多样化源传输大小、目标传输大小和数据大小。数据大小参数（DATA_SIZE）为该传输设置总线宽度。由 SCR_TRANSFER_SIZE 和 DST_TRANSFER_SIZE 设置的源和目标传输大小，该值是 DATA_SIZE 或 32 位。可以将 DATA_SIZE 设置为 32 位、16 位或 8 位。

大多数 PSoC 4 外设寄存器的数据带宽都是 4 字节（32 位），因此，当 DMA 将外设作为其源地址或目标地址时，SCR_TRANSFER_SIZE 或 DST_TRANSFER_SIZE 通常被设置为 32 位。不管需要移植的数据带宽多大，DMA 组件的源和目标地址的传输大小都必须与源和目标地址的编址宽度相匹配。DATA_SIZE 参数与实际数据的宽度相称。例如，如果将 16 位 PWM 用作 DMA 数据的目标地址，那么必须将 DST_TRANSFER_SIZE 设置为 32 位，以符合 PWM 寄存器的宽度匹配，因为 TCPWM 模块（以及大部分 PSoC 4 外设）的外设寄存器宽度始终为 32 位。但是，在该示例中，还可以将目标的 DATA_SIZE 参数设置为 16 位，因为 16 位 PWM 只使用大小为两字节的数据。SRAM 和闪存的编址宽度都是 8 位、16 位或 32 位，并且可以选择任意源和目标地址的传输大小，以满足该应用的需要。

表 5-2 汇总了传输大小设置及其说明的组合。

表 5-2. 传输字节数设置

DATA_SIZE	SCR_TRANSFER_SIZE	DST_TRANSFER_SIZE	典型应用	说明
8 位	8 位	8 位	存储器到存储器	无数据操作
8 位	32 位	8 位	外设到存储器	源位置中的高 24 位被清除
8 位	8 位	32 位	存储器到外设	目标位置中添加了高 24 位零
8 位	32 位	32 位	外设到外设	源位置中的高 24 位被清除，并且标位置中添加了高 24 位零
16 位	16 位	16 位	存储器到存储器	无数据操作
16 位	32 位	16 位	外设到存储器	源位置中的高 16 位被清除
16 位	16 位	32 位	存储器到外设	目标位置中添加了高 16 位零
16 位	32 位	32 位	外设到外设	源位置中的高 16 位被清除，并且标位置中添加了高 16 位零
32 位	32 位	32 位	外设到外设	无数据操作

5.2.3 描述符链接

每个通道具有一个 PING 和一个 PONG 描述符，它带有助于相关传输的不同设置。有效描述由单独通道控制寄存器 (DMAC_CH_CTL) 中 PING_PONG 位的设置。通过 PING 和 PONG 描述符的功能，可以创建描述符的链接列表。这样不需要 CPU 干预也可以从一个传输配置转换到另一个配置。另外，两个描述符意味着当 PONG 寄存器有效时 CPU 可以随时修改 PING 寄存器，反之亦然。

当描述符中的 FLIPPING 位被使能时，它将被链接至 PING/PONG 的副本。可以将该字段与操作码 2 传输模式一起使用。因此，使能 PING 描述符中的 FLIPPING 位并配置该位，将其用于操作码 2 后，该通道将在 PING 描述符末尾自动执行 PONG 描述符。如果配置该位，使其用于操作码 0 或操作码 1，那么需要使用新的触发器来启动 PONG 描述符。

应该在各种传输模式下使用 PING PONG。

5.2.4 传输模式

在执行描述符期间，通道的操作是由操作码设置定义的。这三个操作码适用于 DMA 控制器的所有通道。

5.2.4.1 每个触发器一个单数据元素 (操作码 0)

当配置操作码为 0 时，可以实现该模式。每次收到触发信号时，DMA 会将单个数据元素从源位置传输到目标位置中。可以将该功能与描述符中的其他设置（如源和目标递增）一起使用。

图 5-5 显示的是该传输典型的使用情况。在该图中，ADC 数据寄存器作为源地址，其目标地址是一个外设寄存器（如 PWM 比较）。该触发来自 ADC 的结束转换信号。当收到触发时，传输引擎将从 ADC 中加载数据，并将低 16 位数据存储在 PWM 寄存器内。因为描述符被重新运行，所以连续触发事件可导致相同的行为。

注意：要掌握如何将源和目标地址的数据宽度设置为 32 位。这是因为对 PSoC 中外设寄存器进行的所有访问都必须是 32 位的。由于有效的数据宽度只有 16 位，因此必须保持 DATA_SIZE 为 16 位。

图 5-5. 操作码 0: 各外设间简单的 DMA 传输

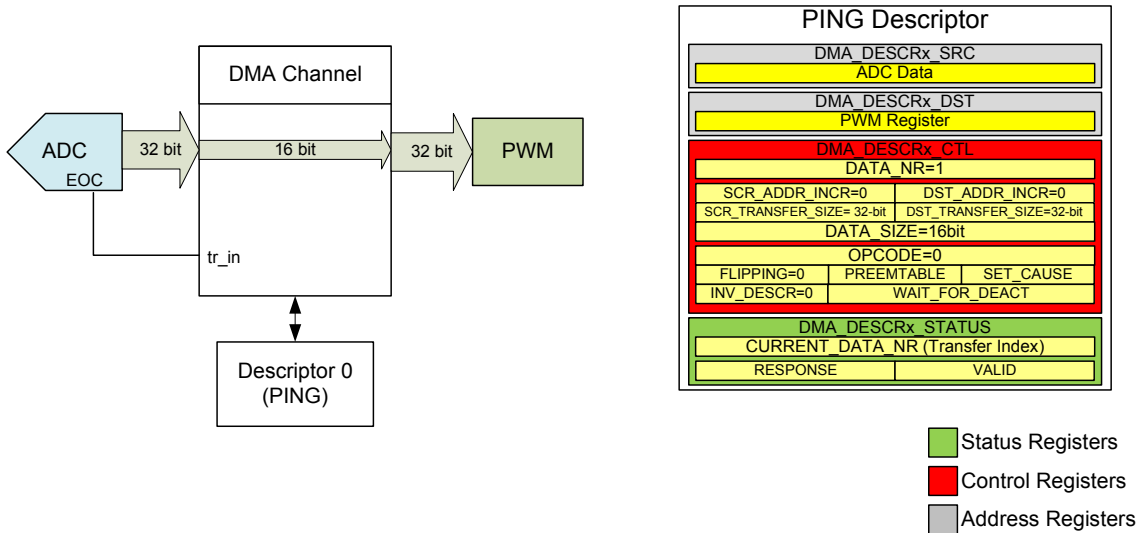


图 5-6 说明了 ADC 数据寄存器和缓冲器之间的数据传输的另一个使用情况。该情况显示的是一个 PING 描述符，从源位置（ADC）提取数据时，该寄存器会递增目标。当收到触发器时，传输引擎将从 ADC 位置加载数据，并将其存储到存储器缓冲区内（Sample 1 的存储位置）。后续触发器继续将 ADC 数据存储到 Sample 1 后连续的位置，直到 PING 描述符缓冲区（DATA_NR 字段）被填满为止。

图 5-6. 操作码 0: 使用目标地址递增功能的数据传输

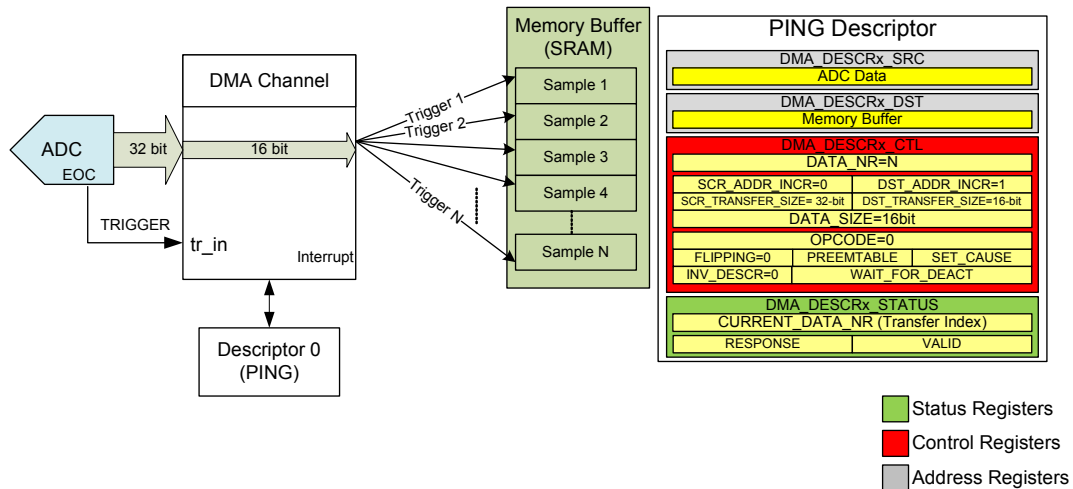
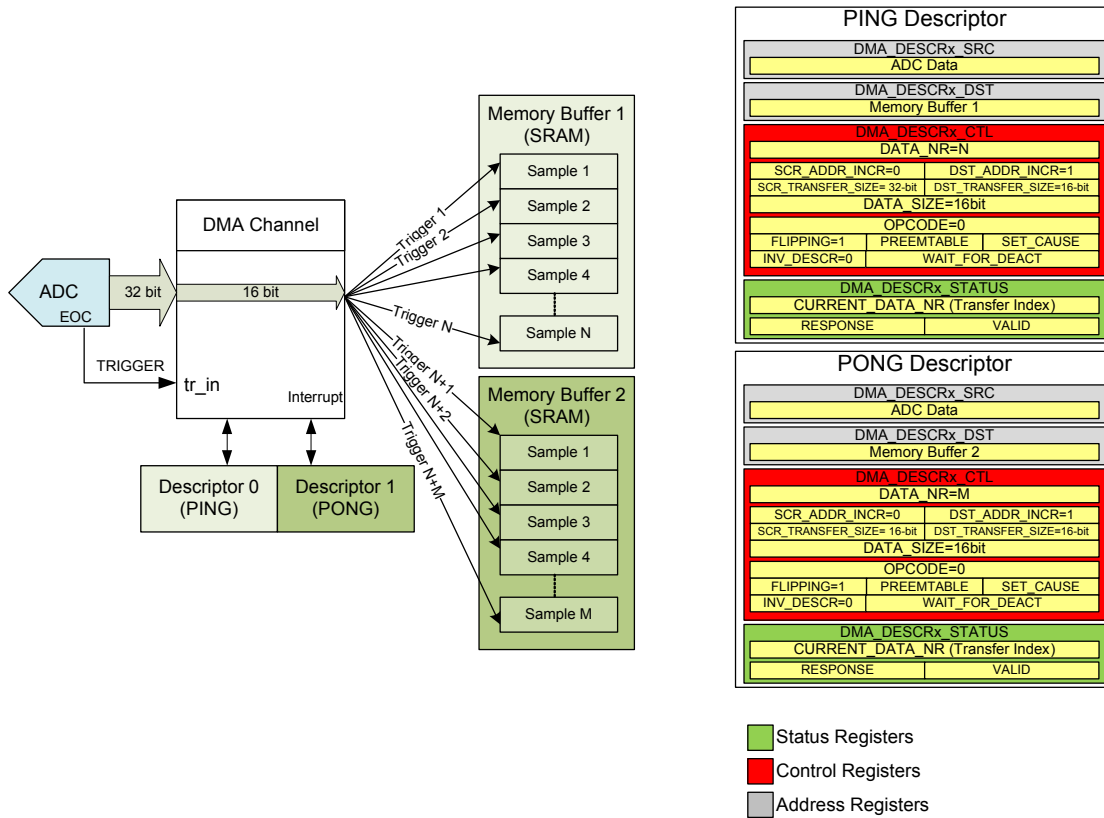


图 5-7 显示的是相同的使用情况。它演示了 PING 和 PONG 描述符的使用情况。完成 PING 描述符后，控制器将翻转，以执行 PONG 描述符。请注意，该翻转位是在描述符中被使能的；该操作使能了描述符的链接。如果翻转位未被使能，那么将重新运行相同的描述符。因此，该序列进行了两次缓冲器传输。另外请注意，传输器一次触发传输完一个元素后才能进行下一次触发传输。

图 5-7. 操作码 0：使用翻转功能的传输

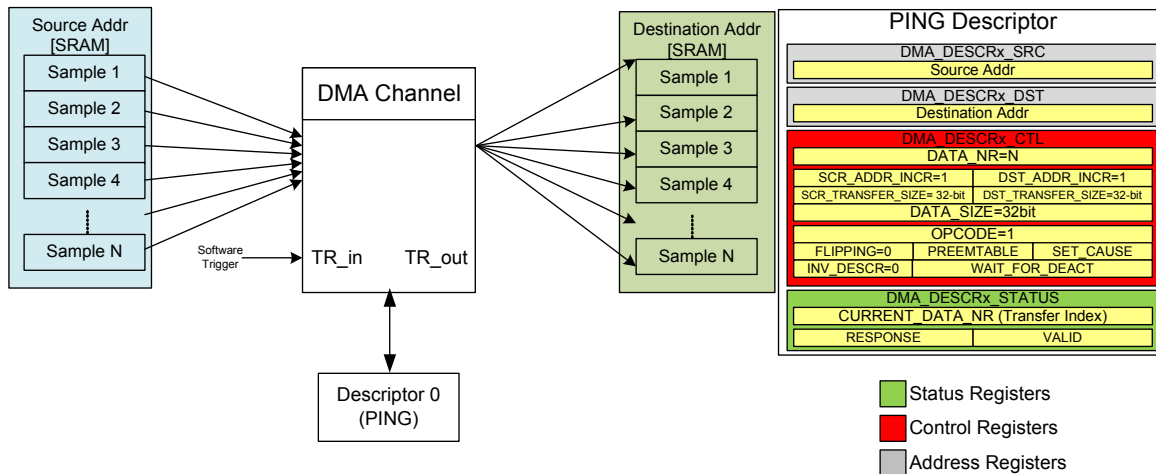


5.2.4.2 每次触发传输整个描述符（操作码 1）

在该操作模式下，在一个触发器中，DMA 能够从源位置将多个数据元素传输到目标位置。如果使用操作码 1，那么控制器将在单个触发器中执行整个描述符。在存储器至存储器缓冲器的传输中，该功能很有用。当遇到触发条件时，将连续进行传输，直到整个描述符被传输完成为止。

图 5-8 显示的是操作码 1 传输，即将源缓冲器中全部内容传输到目标缓冲器内。整个传输是单个 PING 描述符的一部分，并在执行单个触发器时完成。

图 5-8. 使用操作码 1 的 DMA 传输示例

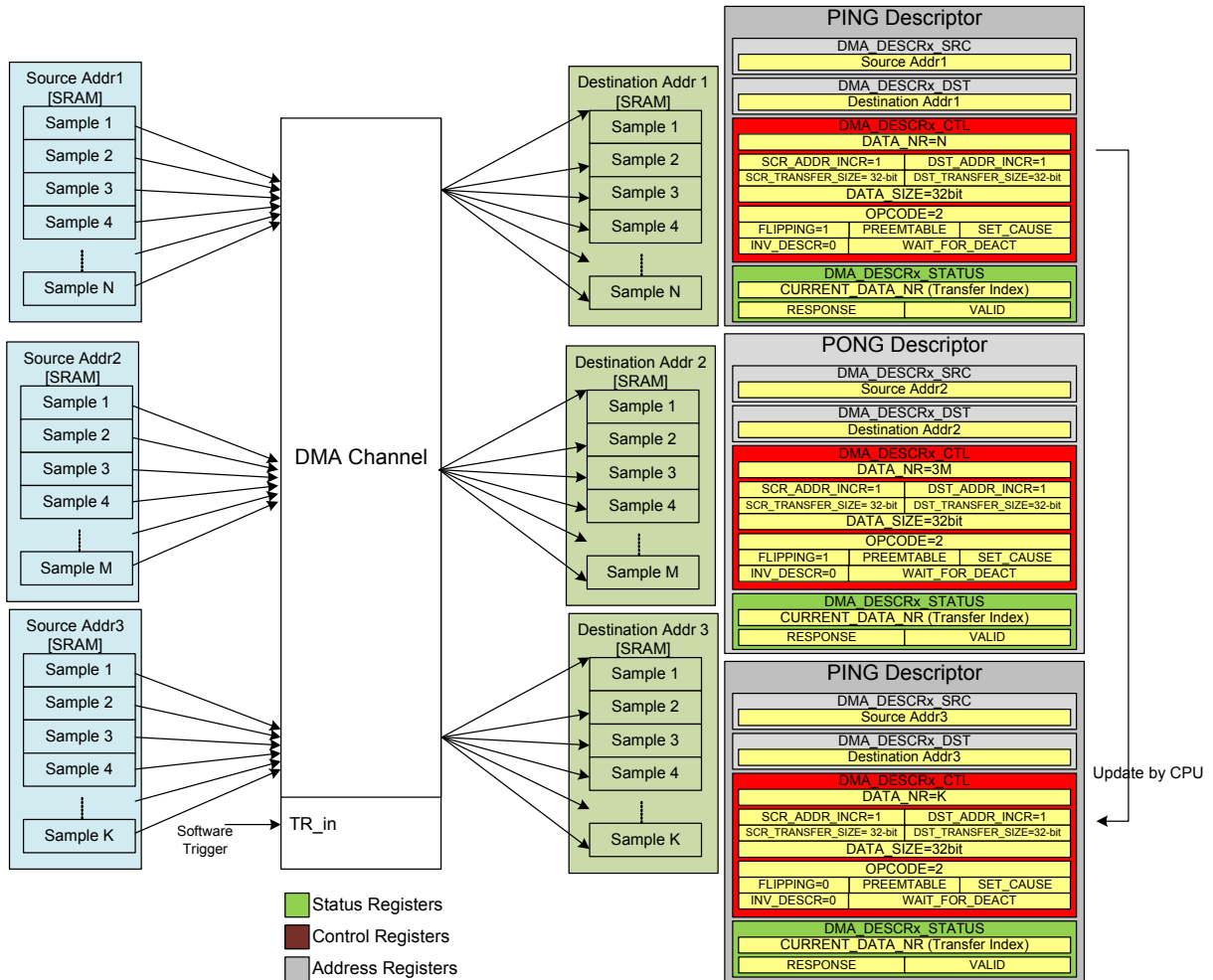


5.2.4.3 每个触发传输整个描述符链接（操作码 2）

操作码 2 始终与 **FLIPPING** 字段一起使用。将操作码 2 与 PING 描述符中的 **FLIPPING** 一起使用时，单个触发器可以执行 PING 描述符，并自动翻转到 PONG 描述符，从而能够按同样的方式执行该描述符。如果 PONG 描述符带有操作码 2，则从 PING 到 PONG 之间的周期将一直持续，直到 CPU 更改描述符或它们不再有效为止。

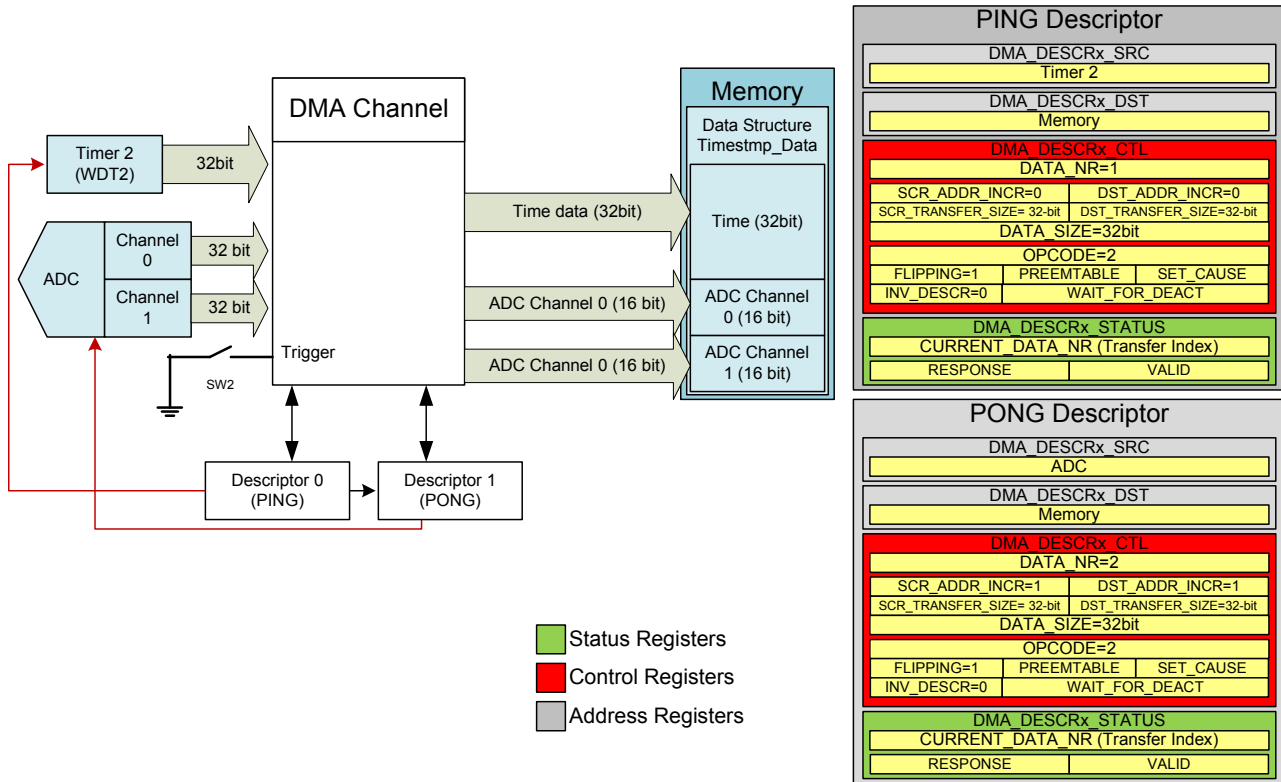
图 5-9 显示的是配置 PING 和 PONG 描述符，使之用于操作码 2 的情况。当执行 PING 寄存器的第二个迭代时，CPU 将禁用 **FLIPPING**。

图 5-9. 使用操作码 2 的 DMA 传输示例



通过制定操作码 2 的传输模式，可以实现不同的使用情况。图 5-10 显示了其中一种。在该图中，源地址数据可来自不同且不连续的存储位置。目标地址的数据结构：连续的存储位置。一个源地址为定时器 **Timer 2**（保持时序数据），另一个源地址为一个 **ADC**。这两个地址的数据被存储在存储器中连续的位置。

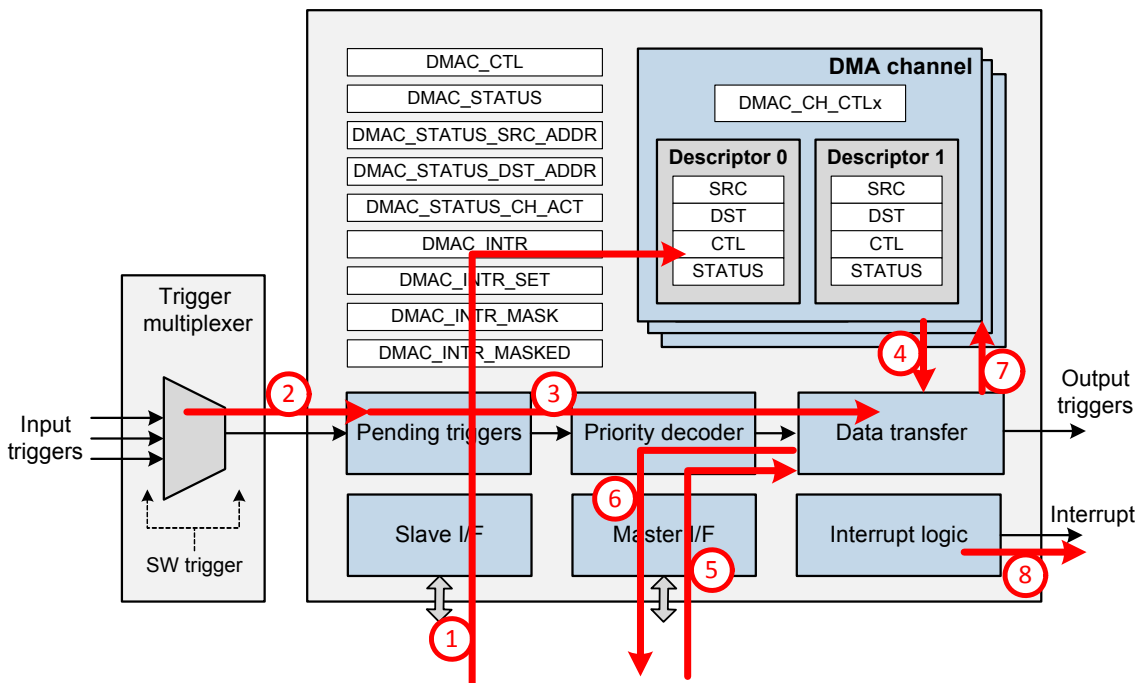
图 5-10. 操作码 2：从多个源地址传输到存储器



5.3 操作与时序

图 5-11 显示的是包含了一个触发器、数据或上面叠加了中断流的 DMA 控制器设计。

图 5-11. 操作流程



该流程图显示的是 DMA 控制器的数据传输的各个步骤。

1. 主 CPU 为特定的通道编程描述符结构。它也对 DMA 寄存器进行编程，从而为该通道选出特定的系统触发器。
2. 激活通道的系统触发器。
3. 通过优先级解码来确定优先级最高的有效通道。
4. 数据传输引擎接受有效的通道，并使用通道标识符来加载通道的描述符结构。描述符结构指定了通道的数据传输方式。
5. 数据传输引擎通过主设备 I/F 进行加载源位置中的数据。
6. 数据传输引擎通过主设备 I/F 将数据存储到目标位置。在单个元素（操作码 0）传输中，步骤 5 和 6 只能执行一次。在多个元素描述符（操作码 1 或 2）的传输中，可以按顺序地多次执行步骤 5 和 6，以实现多次数据元素传输。
7. 数据传输引擎通过更新通道的描述符结构来反映数据传输，并将其存储在描述符的 SRAM 内。
8. 如果由描述符通道结构指定的所有数据传输已经完成，会生成一个中断（这是可编程选项）。

DMA 控制器的数据传输步骤分三种：初始化、并发、或连续。

- 初始化：指的是步骤 1，用于编程描述符的结构。所有的描述符结构都执行该步骤。它由主 CPU 执行，并不能通过有效的通道触发器进行激活。
- 并发：包括步骤 2 和 3。需要对每个通道并行执行这两步。

- 连续：包括步骤 4 到 8。需要对每个有效通道连续执行这些步骤。这样，您可以通过执行这些步骤所占用的时间来确定 DMA 控制器的吞吐量。时长包含两个因素：控制器加载和存储描述符的时间以及在总线设备上传输数据的时间。后续时间则取决于总线延迟（由仲裁器和桥接器组件确定）和目标存储器 / 外设。

传输单个数据元素时，需要 12 个时钟周期来完成一个完整的传输（假定 AHB-Lite 总线不处于等待状态）。在该模式中，可按照下面的公式计算出完成一次传输所需的时钟周期：

时钟周期数量 = 12 + 加载等待状态 + 存储等待状态

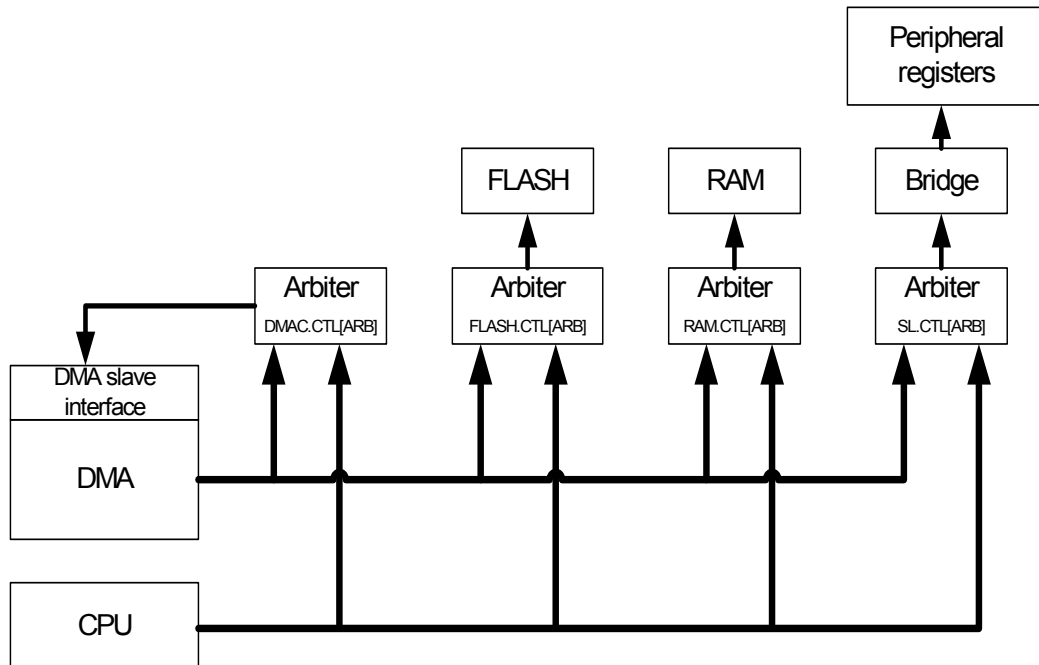
传输所有描述符或连接描述符链路时，需要 12 个时钟周期用于传输第一个数据元素。后续元素需要三个周期。该结果是在 AHB-Lite 走线不处于等待状态的条件下降得出的。对于 ‘N’ 元素的传输，请按照下面公式计算所需时钟周期：

时钟周期数量 = (12 + 加载等待状态 + 存储等待状态) + (N - 1) * (3 + 加载等待状态 + 存储等待状态)

5.4 仲裁

PSoC 4 AHB 总线包含两个主设备：CPU 和 DMA 控制器。所有外设和存储器都通过从设备接口连接到总线上。其中，闪存存储器和 RAM 具有专用的从设备接口，这些接口具有自己的仲裁器。所有外设寄存器都是通过一个专用仲裁器中的桥接器连接到单个从设备接口的。DMA 控制器的从设备接口（用于访问 DMA 控制器的控制寄存器）通过其他从设备接口进行连接。图 5-12 显示的是该架构。

图 5-12. PSoC 4 总线架构



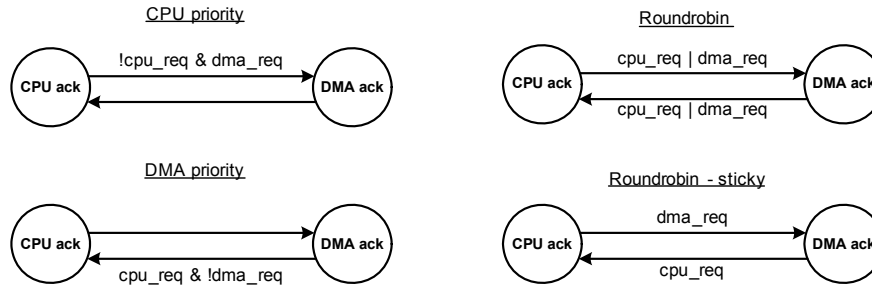
每个从设备可以使用以下任何一种仲裁方式：

- **CPU 优先：**CPU 的仲裁优先级始终最高。只有不存在 CPU 的请求时，才会允许 DMA 的访问。
- **DMA 优先级：**DMA 的仲裁优先级始终最高。只有不存在 DMA 请求时，才会允许 CPU 的访问。
- **轮循：**执行每个请求时，仲裁优先级连续在 DMA 和 CPU 之间切换。每次请求（CPU 或 DMA）中仲裁优先级都会切换。

- **粘性轮循：**该模式类似于轮循模式，但它仅在接收到更低优先级主设备的请求时切换优先级。例如，如果当前的优先级是 CPU，而 DMA 发出了一个请求，那么优先级将在下一个请求中交给 DMA。如果 DMA 没有发出任何请求，则 CPU 会保持当前的优先级。

下图详细介绍了各个仲裁模型。

图 5-13. 仲裁模型



5.5 寄存器列表

寄存器名称	说明	特性
DMAC_CTL	模块控制	用于使能 DMA 控制器的位。
DMAC_STATUS	模块状态	提供了有关 DMA 控制器的状态信息。
DMAC_STATUS_SRC_ADDR	当前源地址	提供了当前加载的源地址的相关信息。
DMAC_STATUS_DST_ADDR	当前目标地址	提供了当前加载的目标地址的相关信息。
DMAC_STATUS_CH_ACT	通道有效状态	软件通过读取该字段获取所有挂起通道的信息（该信息处于挂起状态或位于数据传输引擎内）。
DMAC_CH_CTLx	通道控制寄存器	为通道 x 提供通道使能、PING/PONG 和优先级设置等操作。
DMAC_DESCRx_PING_SRC	PING 源地址	通道 x 源位置的基地址。
DMAC_DESCRx_PING_DST	PING 目标地址	通道 x 目标位置的基地址。
DMAC_DESCRx_PING_CTL	PING 控制字	PING 描述符的所有控制设置。
DMAC_DESCRx_PING_STATUS	PING 状态字	有效性、应答和实时 Data_NR 索引状态。
DMAC_DESCRx_PONG_SRC	PONG 源地址	通道 x 源位置的基地址。
DMAC_DESCRx_PONG_DST	PONG 目标地址	通道 x 目标位置的基地址。
DMAC_DESCRx_PONG_CTL	PONG 控制字	PONG 描述符的所有控制设置。
DMAC_DESCRx_PONG_STATUS	PONG 状态字	有效性、应答和实时 Data_NR 索引状态。
DMAC_INTR	中断寄存器	
DMAC_INTR_SET	中断设置寄存器	读取时，该寄存器反映了中断请求寄存器的状态。
DMAC_INTR_MASK	中断屏蔽	INTR 寄存器中相应字段的屏蔽。
DMAC_INTR_MASKED	中断屏蔽寄存器	读取时，该寄存器反映了中断请求和屏蔽寄存器之间的按位逻辑与（AND）运算后的结果。该寄存器允许软件通过单次加载操作来读取所有屏蔽使能中断的原因，而不需要通过两次加载操作：一个用于中断原因，一个用于屏蔽。这样便简化了固件的开发过程。

6. 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进行的程序流程，同时 CPU 将执行中断服务子程序和异常事件处理程序。PSoC 4 为中断处理程序 /ISR 和异常事件处理程序提供了统一的异常向量表。

6.1 特性

PSoC 4 支持下面的中断特性：

- 支持 32 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
- 支持电平触发和脉冲触发的中断信号

6.2 工作原理

图 6-1. PSoC 4 中断框图

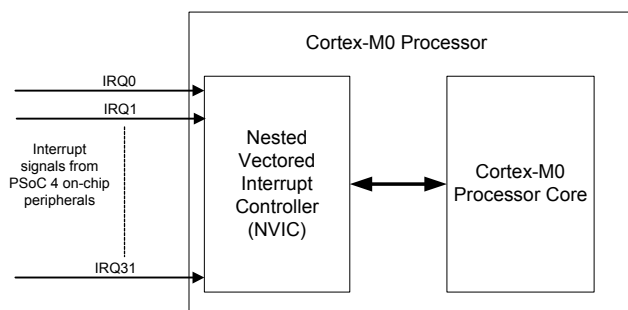


图 6-1 显示的是中断信号与 Cortex-M0 CPU 之间的交互。PSoC 4 一共有 32 个中断；这些中断均由 NVIC 处理。NVIC 分别用于使能 / 禁用单独中断，优先级处理，和与 CPU 内核通信。与 CPU 的外部设备所生成的中断不同，异常事件是由 CM0 内核生成的，因此，它们并未显示在图 6-1 中。

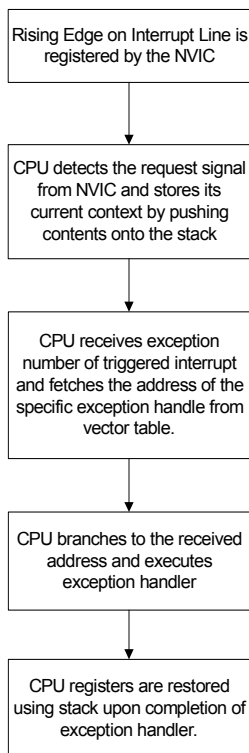
6.3 中断和异常事件 — 操作

6.3.1 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. 假设所有中断信号的初始状态均为低电平（闲置或非活动状态），且处理器正在执行主代码，NVIC 会记录所有中断线上的上升沿。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 6-1 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程序的地址。
4. 然后，CPU 会跳转到该地址，并执行相应的异常事件处理程序。
5. 在完成执行异常事件处理程序后，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 也会恢复执行主代码。

图 6-2. 被触发时的中断处理



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优先级低的 ISR 的执行。

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程序。

6.3.2 电平与脉冲中断

PSoC 4 NVIC 支持中断线（IRQ0 至 IRQ31）上的电平和脉冲信号。中断的类型（电平或脉冲）是由中断源决定的。

图 6-3. 电平中断

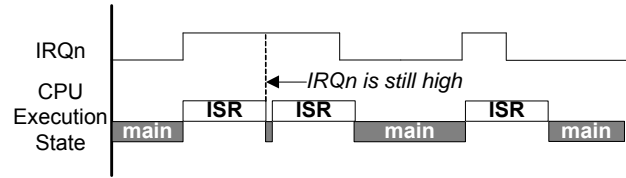


图 6-4. 脉冲中断

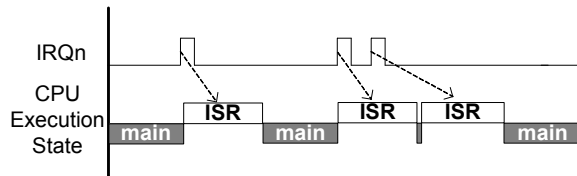


图 6-3 和图 6-4 分别显示的是工作中的电平中断和脉冲中断。假设中断信号最初为非活动状态（逻辑低），那么以下各个事件序列说明了如何处理电平中断和脉冲中断：

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 6-4，了解脉冲中断的相关信息）。
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 6-3 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。

6.3.3 异常事件向量表

异常事件向量表（表6-1）保存的是PSoC 4中所有异常事件处理程序的入口点地址。CPU会根据异常事件编号加载相应的地址。

表 6-1. 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	HardFault（硬故障）	–1	Base_Address + 0x0C
4-10	保留	NA	Base_Address + 0x10 至 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 至 Base_Address + 0x34
14	管理程序挂起（PendSV）	可配置（0 - 3）	Base_Address + 0x38
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）	...
47	外部中断（IRQ31）	可配置（0 - 3）	Base_Address + 0xBC

在表 6-1 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 能够从 SRAM 中的向量表位置加载异常事件处理程序的地址。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程序的地址。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。

如果未设置 CPUSS_SYSREQ 寄存器中的 NO_RST_OVR 位，那么在读取 0x00000000 和 0x00000004 闪存地址时会重新指向 SRAM 的前八个字节，以加载堆栈指针和复位向量。要想读取 0x00000000 和 0x00000004 闪存地址，需要将 NO_RST_OVR 位设置为 ‘1’。堆栈指针向量将保持复位时堆栈指针所加载的地址。复位向量保持引导序列的地址。当器件完成复位时，执行该映射以使用 SRAM 上堆栈指针和复位向量的默认地址。在复位过程中，先执行 SRAM 中的启动代码，然后 CPU 跳转到闪存中的 0x00000004 地址，在闪存中处理程序。复位时，VECT_IN_RAM 为 0，因此从未使用 SRAM 向量表中的复位异常事件地址。

此外，CPUSS_SYSREQ 寄存器的 SYSREQ 位被设置时，对闪存地址 0x00000008 的读取将指向 SRAM（而不是闪存），以访问 NMI 向量地址。复位 CPUSS_SYSREQ，以读取 0x00000008 闪存地址。

6.4 异常事件源中说明了各个异常事件源（异常事件编号 1 到 15）。虽然在表 6-1 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。6.5 中断源中对各个中断源（异常事件编号 16 到 47）进行了介绍。

6.4 异常事件源

本节介绍了表 6-1 中所列出的不同异常事件源（异常事件编号 1 到 15）。

6.4.1 复位异常事件

在 PSoC 4 中，器件复位被作为一个异常事件。该异常始终被使能，其优先级固定为 –3（最高优先级）。器件复位可以由不同的原因引起，如：上电复位（POR）、XRES 引脚上的外部复位信号或看门狗复位。在器件复位时，在监控只读存储器（SRAM）范围外将执行用于配置器件的初始引导代码。SRAM 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 SRAM 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 6-1 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。

6.4.2 不可屏蔽中断（NMI）异常事件

除了复位外，不可屏蔽中断（NMI）是优先级最高的异常事件。该中断始终被使能，固定优先级为 -2。在 PSoC 4 中，有三种方法用于触发 NMI 异常事件：

- **由硬件信号触发的 NMI 异常事件（用户 NMI 异常事件）：**
PSoC 4 可通过数字信号触发 NMI 异常事件。该数字信号指的是表 6-3 中的 `irq_out[0]`。由 `irq_out[0]` 触发的 NMI 异常事件将执行活动向量表（闪存或 SRAM 向量表）所指向的 NMI 处理器。
- **通过置位 NMIPENDSET 位触发的 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，可以执行活动向量表（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。通过置位 CPUSS_SYSREQ 寄存器中的 SYSCALL_REQ 位对该异常事件进行触发。通过 SYSCALL_REQ 位触发的 NMI 异常事件始终执行 SRAM 中的 NMI 异常事件处理程序代码。闪存或 SRAM 异常事件向量表不适用于系统调用 NMI 异常事件。不能对 SRAM 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。

6.4.3 HardFault（硬故障）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，即其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 异常处理程序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。

6.4.4 管理程序调用（SVCcall）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（SVCcall）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并提供一个处理操作。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 SVCcall 异常事件无关）。（有关特权模式的详细信息，请参考第 95 页上的芯片运行模式章节。）在 PSoC 4 中的架构级别上，不支持 SVCcall 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 SVCcall 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]`，可以配置 SVCcall 异常的优先级（范围为 0 到 3）。当执行 SVC 指令时，SVCcall 异常将进入挂起状态，并等待 CPU 进行处理。使用系统处理器控制和状态寄存器（SHCSR）中的 SVCALLPENDE 位来检查或修改 SVCcall 异常的挂起状态。

6.4.5 PendSV 异常事件

PendSV 是另一种与 SVCcall 相似的管理程序调用异常事件，该异常事件通常由软件生成。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 异常事件会进入挂起状态，并等待 CPU 对其进行处理。通过清除中断控制状态寄存器（CM0_ICSR）中的 PENDSVCLR 位，可以清除 PendSV 异常事件的挂起状态。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 异常事件的优先级（范围为 0 到 3）。更多有关信息，请参考 ARMv6-M 架构参考手册。

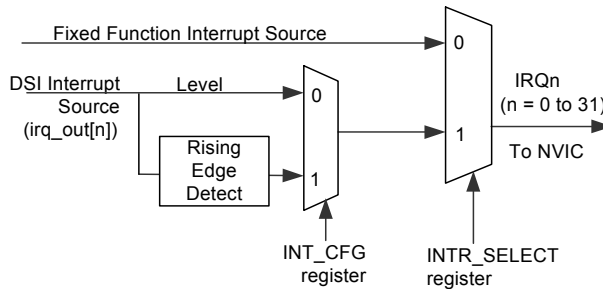
6.4.6 SysTick 异常事件

PSoC 4 中的 CM0 CPU 支持将系统定时器（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，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 异常事件的优先级（范围为 0 到 3）。随时可通过向中断控制状态寄存器 CM0_ICSR 中的 PENDSTSET 位写入 1，在软件中生成 SysTick 异常事件。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 异常事件的挂起状态。

6.5 中断源

PSoC 4 支持来自外设的 32 个中断 (IRQ0 - IRQ31 或异常编号 16 到 47)。表 6-3 介绍了所有中断源。PSoC 4 为 32 个中断线中每一个都提供了灵活的源选项。图 6-5 显示了中断源的复用选项。每个中断均有两个源：固定功能的中断源和 DSI 中断源。通过 CPUSS_INTR_SELECT 寄存器，可以选用所需的 interrupt 源。

图 6-5. 中断源复用



注意： 不能对名称为 (irq_out[n]) 的 DSI 中断信号进行访问，但通过使用 DSI 中断路径路由数字信号，PSoC Creator IDE 可以简化该访问操作。因此，您不需要手动配置 DSI 路径。

固定功能中断包括来自片上外设的标准中断，如 TCPWM、串行通信模块、和 CSD 模块。通常通过对外设的不同状态进行“或”逻辑运算来生成固定功能中断。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。固定功能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。

中断源的第二类是 DSI 中断信号。共有八个 DSI 通道，每个通道被多路分解为 4 以跨越 Cortex M0 的 32 个中断源。片上的任何数字信号，如来自 UDB 的数字输出或引脚上的数字输入信号，都能作为 DSI 中断源使用。这样便能够更加灵活地选择中断源。您可以通过上升沿检测电路路由 DSI 信号，如图 6-5 所示。该边沿检测电路将 DSI 线上的一个上升沿信号转换为宽度为两个系统时钟的脉冲信号。这样确保在 DSI 线上信号的上升沿到来时，会立即触发中断。这对不能为 NVIC 生成合适的电平中断信号的中断源很有帮助。UDB_INT_CFG 寄存器用于选择直接 DSI 路径还是边沿检测路径。由于 DSI 中断通道被多路分解，因此每次执行的 DSI 中断的最大数量被限制为 8。

请参考第 63 页上的 I/O 系统章节，了解 GPIO 中断的详细信息。

表 6-2. PSoC 4 中断源的列表

中断	Cortex-M0 异常编号	固定函数的中断源	DSI 中断源
NMI (请参见第 53 页上的异常事件源)	2	SYS_REQ	udb.interrupts[0]:4
IRQ0	16	GPIO 中断 — 端口 0	udb.interrupts[0]:0
IRQ1	17	GPIO 中断 — 端口 1	udb.interrupts[1]:0
IRQ2	18	GPIO 中断 — 端口 2	udb.interrupts[2]:0
IRQ3	19	GPIO 中断 — 端口 3	udb.interrupts[3]:0
IRQ4	20	GPIO 中断 — 端口 4	udb.interrupts[4]:0
IRQ5	21	GPIO 中断 — 端口 13 (USB 唤醒)	udb.interrupts[5]:0
IRQ6	22	GPIO 中断 — 所有端口 ^a	udb.interrupts[6]:0
IRQ7	23	LPCOMP (低功耗比较器)	udb.interrupts[7]:0
IRQ8	24	WDT (看门狗定时器)	udb.interrupts[0]:1
IRQ9	25	SCB0 (串行通信模块 0)	udb.interrupts[1]:1
IRQ10	26	SCB1 (串行通信模块 1)	udb.interrupts[2]:1
IRQ11	27	SCB2 (串行通信模块 2)	udb.interrupts[3]:1
IRQ12	28	SCB3 (串行通信模块 3)	udb.interrupts[4]:1
IRQ13	29	CTBm 中断 (所有 CTBm)	udb.interrupts[5]:1
IRQ14	30	DMA 中断	udb.interrupts[6]:1
IRQ15	31	SPCIF 中断	udb.interrupts[7]:1
IRQ16	32	LVD 中断	udb.interrupts[0]:2
IRQ17	33	SAR (逐次逼近 ADC)	udb.interrupts[1]:2
IRQ18	34	CSD0 (CapSense)	udb.interrupts[2]:2
IRQ19	35	CSD1 (CapSense)	udb.interrupts[3]:2
IRQ20	36	TCPWM0 (定时器 / 计数器 / PWM 0)	udb.interrupts[4]:2

表 6-2. PSoC 4 中断源的列表（续）

中断	Cortex-M0 异常编号	固定函数的中断源	DSI 中断源
IRQ21	37	TCPWM1（定时器 / 计数器 / PWM 1）	udb.interrupts[5]:2
IRQ22	38	TCPWM2（定时器 / 计数器 / PWM 2）	udb.interrupts[6]:2
IRQ23	39	TCPWM3（定时器 / 计数器 / PWM 3）	udb.interrupts[7]:2
IRQ24	40	TCPWM4（定时器 / 计数器 / PWM 4）	udb.interrupts[0]:3
IRQ25	41	TCPWM5（定时器 / 计数器 / PWM 5）	udb.interrupts[1]:3
IRQ26	42	TCPWM6（定时器 / 计数器 / PWM 6）	udb.interrupts[2]:3
IRQ27	43	TCPWM7（定时器 / 计数器 / PWM 7）	udb.interrupts[3]:3
IRQ28	44	CAN0 中断	udb.interrupts[4]:3
IRQ29	45	CAN1 中断	udb.interrupts[5]:3
IRQ30	46	USB — 起始帧	udb.interrupts[6]:3
IRQ31	47	USB — EP1 到 EP8 的数据	udb.interrupts[7]:3

a. 端口 5 到端口 12 没有专用的中断向量编号；这些端口使用 IRQ 6。

6.6 异常事件优先级

当 CPU 需要处理多个异常事件时，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事件选择优先级值。除复位、NMI 和 HardFault 外，可对其他所有异常事件分配可配置的优先级别。复位、NMI 和 HardFault 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 HardFault 异常的优先级最高。其他异常事件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于（中断）当前运行的异常事件处理。如果下一个异常事件与当前运行的异常事件有相同的优先级，那么下一个异常事件不会被处理。处理完成优先级更高的异常事件后，CPU 会恢复执行优先级更低的异常事件。PSoC 4 中的 CM0 CPU 最多支持嵌套四个异常事件。当 CPU 接收到两个或多个优先级相同的异常事件请求，将先执行编号最低的异常事件。

第 53 页上的异常事件源中介绍了用于配置优先级的异常事件编号 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 32 个中断（IRQ0 - IRQ31）的优先级。该组寄存器包括八个 32 位寄存器，其中每个寄存器用于保存四个中断的优先级值，如表 6-3 中的介绍。不使用该寄存器的其他位字段。

表 6-3. 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。

6.7 使能和禁用中断

NVIC 提供了各种寄存器，用于单独使能和禁用软件中的 32 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 6-4 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起任何作用。

表 6-4. 中断使能 / 禁用寄存器

寄存器	操作	位值	注意
中断设置使能寄存器（CM0_ISER）	写	1	使能中断
		0	无作用
	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	无作用
	读	1	中断使能
		0	中断禁用

CM0_ISR 和 CM0_ICER 寄存器仅适用于中断（IRQ0 - IRQ31）。不能通过这些寄存器使能或禁用编号为 1 到 11 的异常事件。这 15 个异常事件具有它们专有的使能和禁用支持，如第 53 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 6-1 中介绍的复位、NMI 和 HardFault 等异常事件外，其他所有的异常事件的优先级都是可配置的。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 不会处理任何优先级可配置的异常事件。但这些异常事件可以处于挂起状态，等到 PM 位被清除后，CPU 将处理该事件。

6.8 异常事件的状态

每个异常事件都能处在下面各状态之一。

表 6-5. 异常事件的状态

异常事件的状态	含义
非活动	该异常事件既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，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事件，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 IRQ31）是否处于挂起状态。

6.8.1 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程序时，该异常事件会从挂起状态转为活动状态。

通过使用独立的寄存器位来设置和清除中断的挂起状态，NVIC 允许软件挂起 32 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 6-6 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起任何作用。

表 6-6. 中断设置挂起 / 清除挂起寄存器

寄存器	操作	位值	注意
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	无作用
	读	1	中断正在挂起
		0	中断没有挂起
中断清除挂起寄存器（CM0_ICPR）	写	1	清除挂起的中断
		0	无作用
	读	1	中断正在挂起
		0	中断没有挂起

在挂起位被置位时再次对该位进行设置，但 ISR 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。如果未使能中断，则中断线不会转入挂起状态，直到通过写入 CM0_ISR 寄存器使能该中断为止。

注意，CM0_ISPR 和 CM0_ICPR 寄存器仅用于 32 个外设中断（异常编号 16-47）。不能使用它们来挂起编号为 1 到 11 的异常事件。这 15 个异常事件具有自己的挂起支持，如第 53 页上的异常事件源所述。

6.9 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，在处理器模式中，CPU 开始执行相应的异常处理器。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 具有两个堆栈指针 — 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，以执行异常处理程序。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 29 页上的 Cortex-M0 CPU 章节中介绍的内容。

Cortex-M0 使用两项技术（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（infocenter.arm.com/help/topic/com.arm.doc.ddi0419c/index.html）。

6.10 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。当一个或多个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 ISR。

由 CM0 CPU 执行的等待中断（WFI）指令可使器件进入睡眠、深度睡眠和休眠等模式。第 97 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三类固定功能的中断源：

- 可用于活动、深度睡眠和休眠等模式的固定功能中断源（例如，GPIO 中断，低功耗比较器）。
- 仅用于活动模式和深度睡眠模式的固定功能中断源（例如，看门狗定时器中断、串行通信模块中断）
- 仅用于活动模式的固定功能中断源（其他所有的固定功能中断）

6.11 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。如果该列表位于闪存内，则需要通过对闪存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 6-1 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 56 页上的异常事件优先级所述。
 - d. 使能该异常，如第 56 页上的使能和禁用中断所述。

6.12 寄存器

表 6-7. 寄存器列表

寄存器名称	说明
CM0_IUSER	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
CPUSS_INTR_SELECT	中断复用器选择寄存器
UDB_INT_CFG	UDB 子系统中断配置

6.13 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。

中断

节 C: 系统范围资源

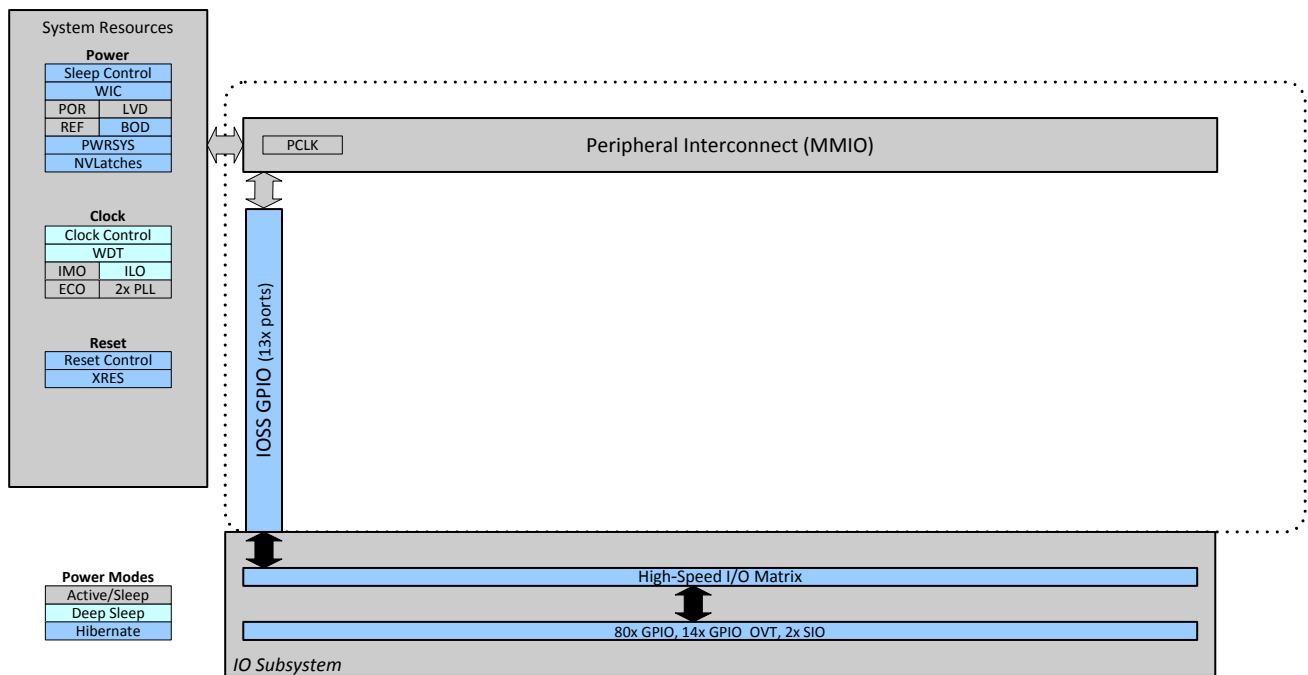


这部分包括以下章节:

- 第 63 页上的 I/O 系统章节
- 第 77 页上的时钟系统章节
- 第 91 页上的电源与电源监测章节
- 第 95 页上的芯片运行模式章节
- 第 97 页上的功耗模式章节
- 第 101 页上的看门狗定时器章节
- 第 105 页上的复位系统章节
- 第 109 页上的器件安全性章节

顶层架构

系统范围资源的框图



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 中的通用 I/O（GPIO）引脚被分为不同的端口；每个端口最多可包含 8 个 GPIO。PSoC 4200L 系列最多有 98 个 GPIO（其中有两个 USB 引脚），它们被分成 14 个端口。

7.1 特性

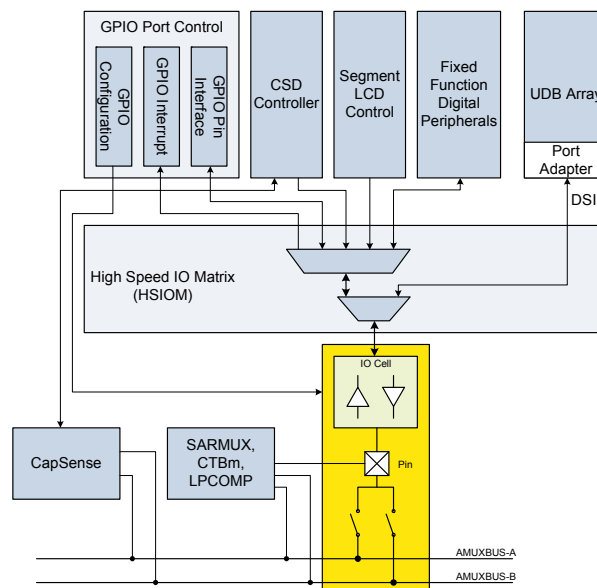
PSoC 4 GPIO 具有以下特性：

- 支持模拟 / 数字输入和输出
- 支持八种驱动强度模式
- 具有 14 个过压容差（OVT-GPIO）引脚
- 具有两个特殊 I/O（SIO）
- 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
- 转换速率控制
- 保持模式，用于锁存前一状态（在深度睡眠模式下保持 I/O 状态）
- 可选择输入缓冲器模式：CMOS 输入或低电压 LVTTTL 输入
- 支持 Capsense 功能
- 支持段式 LCD 驱动

7.2 GPIO 接口概况

PSoC 4 由多个模拟和数字外设组成。图 7-1 显示的是各个外设和引脚间的连接概况。

图 7-1. GPIO 接口概况

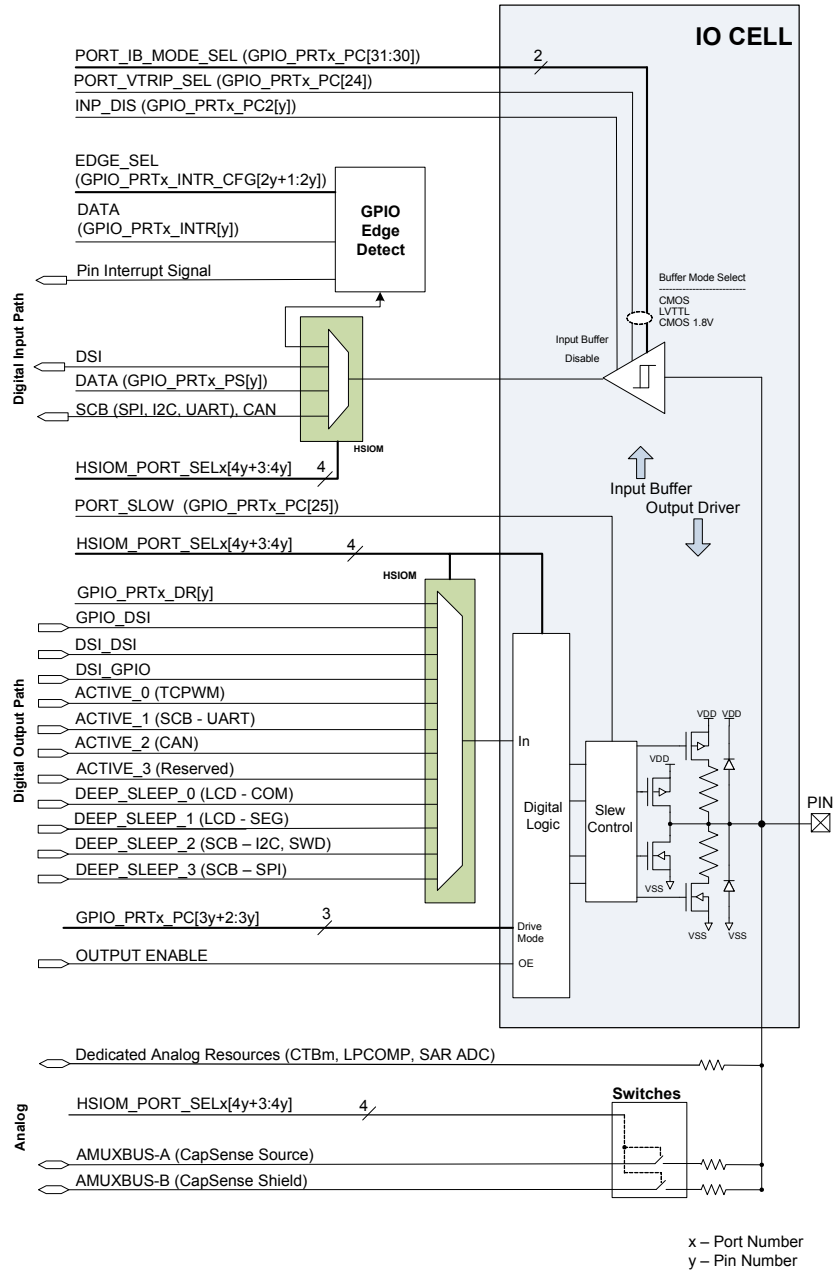


GPIO 引脚被连接至 I/O 单元。这些单元都配备了数字输入的缓冲器（这样可提供高阻抗输入），以及数字输出信号的驱动器。数字外设通过高速 I/O 矩阵（HSIOM）可连接至 I/O 单元。HSIOM 包含多个复用器，用于将用户选定的器件连接到引脚上。HSIOM 还建立了数字系统互连（DSI）与引脚间的连接。这样可以将引脚信号路由到已连接了 DSI 的外设（如 UDB）上。模拟外设（如 SAR ADC、微型连续时间模块（CTBm）、低功耗比较器（LPCOMP）和 CapSense）可直接或通过 AMUX 总线连接到 GPIO 引脚上。

7.3 I/O 单元架构

图 7-2 显示的是 I/O 单元架构。它包括一个输入缓冲器和一个输出驱动器。所有 GPIO 单元都有这种架构。它连接着数字输入和输出信号的 HSIOM 复用器。模拟外设直接连接到引脚上。

图 7-2. PSoC 4200L 中的 I/O 单元架构



7.3.1 数字输入缓冲器

数字输入缓冲器为外部数字输入提供了高阻抗缓冲器。可通过端口配置寄存器 2（GPIO_PRTx_PC2，其中 x 是端口编号）的 INP_DIS 位使能 / 禁用该缓冲器。可以将该缓冲器配置为以下几种模式：

- CMOS
- LVTTL
- 1.8 V CMOS

这些缓冲器模式可通过端口配置寄存器的 PORT_VTRIP_SEL 位（GPIO_PRTx_PC[24]）和 PORT_IB_MODE_SEL 位（GPIO_PRTx_PC[31:30]）选定。

表 7-1. 输入缓冲器模式

PORT_VTRIP_SEL	PORT_IB_MODE_SEL	输入缓冲器模式
0b	0b 或 10b	CMOS
1b	0b 或 10b	LVTTL
x	1b 或 11b	1.8 V CMOS

欲了解每种模式的阈值，请参见器件数据手册。输入缓冲器的输出被连接至 HSIOM，以便路由到所选外设。通过对 HSIOM 端口选择寄存器（HSIOM_PORT_SELx）进行写操作，可以选择所需外设。HSIOM 中的数字输入外设（如图 7-2 中所示）都依赖于引脚。请参阅器件数据手册，了解每个引脚的功能。

7.3.2 数字输出驱动器

引脚是通过数字输出驱动器驱动的。该驱动器中包含的电路

表 7-3. 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’（0 ≤ y ≤ 7）的驱动模式		
	模拟高阻态	0	高阻态	高阻态
	数字高阻态	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
	漏极开路，驱动低电平	4	高阻态	强 0
	漏极开路，驱动高电平	5	强 1	高阻态
	强驱动	6	强 1	强 0
	电阻上拉和下拉	7	弱 1	弱 0

用于为数字输出信号实现各种不同的驱动模式和转换速率控制。外设通过 HSIOM 连接到数字输出驱动器；通过写入到 HSIOM 端口选择寄存器（HSIOM_PORT_SELx），可以选择特定的外设。三个 I/O 组由 V_{DDP}、V_{DDA} 或 V_{DDIO} 电源供电。下表显示的是各个 I/O 组中的端口以及 I/O 电源。

表 7-2. I/O 组

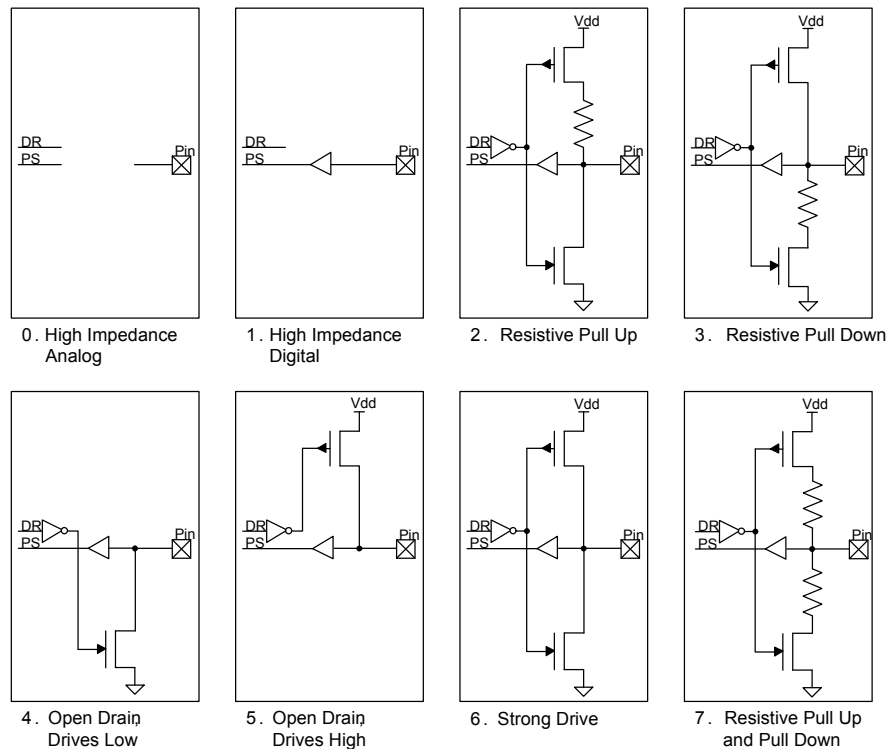
端口	IO 供电电压
端口 0、端口 7、端口 8、端口 9	V _{DDP} /V _{SSD}
端口 1、端口 2、端口 5	V _{DDA} /V _{SSA}
端口 3、端口 4、端口 6、端口 10、端口 11、端口 12	V _{DDIO} /V _{SSIO}

每个 GPIO 引脚都有 ESD 二极管，用于维持引脚电压不超过 I/O 电源电压。请确保在任何情况下，引脚电压都不能超过 I/O 电源电压 V_{DDIO}/V_{DDP}/V_{DDA}，也不能低于 V_{SSIO}/V_{SSD}/V_{SSA}。有关 GPIO 电压的最大和最小绝对值，请参阅器件数据手册。通过外设的 DSI 信号或与输出引脚相关联的数据寄存器（GPIO_PRTx_DR），可以启用或禁用数字输出驱动器。请参考 7.6 高速输入 / 输出矩阵一节，了解有关数据的外设源选择以及启用 / 禁用控制源选择的详细信息。

7.3.2.1 驱动模式

通过使用端口配置寄存器（GPIO_PRTx_PC），可以单独将每个 I/O 配置为八种驱动模式中的任意一种。表 7-3 中列出了各种模式。图 7-2 是简便的输出驱动器框图，它显示了基于八种驱动模式的引脚视图。

图 7-3. I/O 驱动模式框图



■ 模拟高阻态

模拟高阻态模式是默认的复位状态：在这种模式下，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。模拟高阻态引脚不能作为数字输入。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下的最低器件电流，必须将未使用的 GPIO 配置为模拟高阻态模式。

■ 数字高阻态

数字高阻态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，会使能数字输入信号的输入缓冲器。

■ 电阻上拉或电阻下拉

电阻模式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强驱动。在这些模式下，引脚可作为数字输入或输出使用。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’ 写到该引脚的数据寄存器位上。机械开关接口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，可使用这些电阻模式。输入为开漏低电平时，使用电阻上拉；输入为开漏高电平时，则使用电阻下拉。

■ 开漏高电平驱动和开漏低电平驱动

开漏模式在某一种数据状态下提供高阻抗，在另一种数据状态下提供强驱动。在这些模式下，引脚可作为数字输入或输出使用。因此，这些模式广泛用于双向数字通信。当信号采用

外部电阻下拉时，使用开漏高电平驱动模式；信号采用外部电阻上拉时，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 I²C 总线信号线。

■ 强驱动

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，强驱动模式的引脚不能作为输入使用。这种模式通常适用于数字输出信号或驱动外部晶体管。

■ 电阻上拉和电阻下拉

在电阻上拉和电阻下拉模式下，GPIO 在逻辑 1 和逻辑 0 输出状态下均有串联电阻。高电平数据被上拉，而低电平数据被下拉。当其他可能导致短路的信号驱动总线时，可使用此模式。

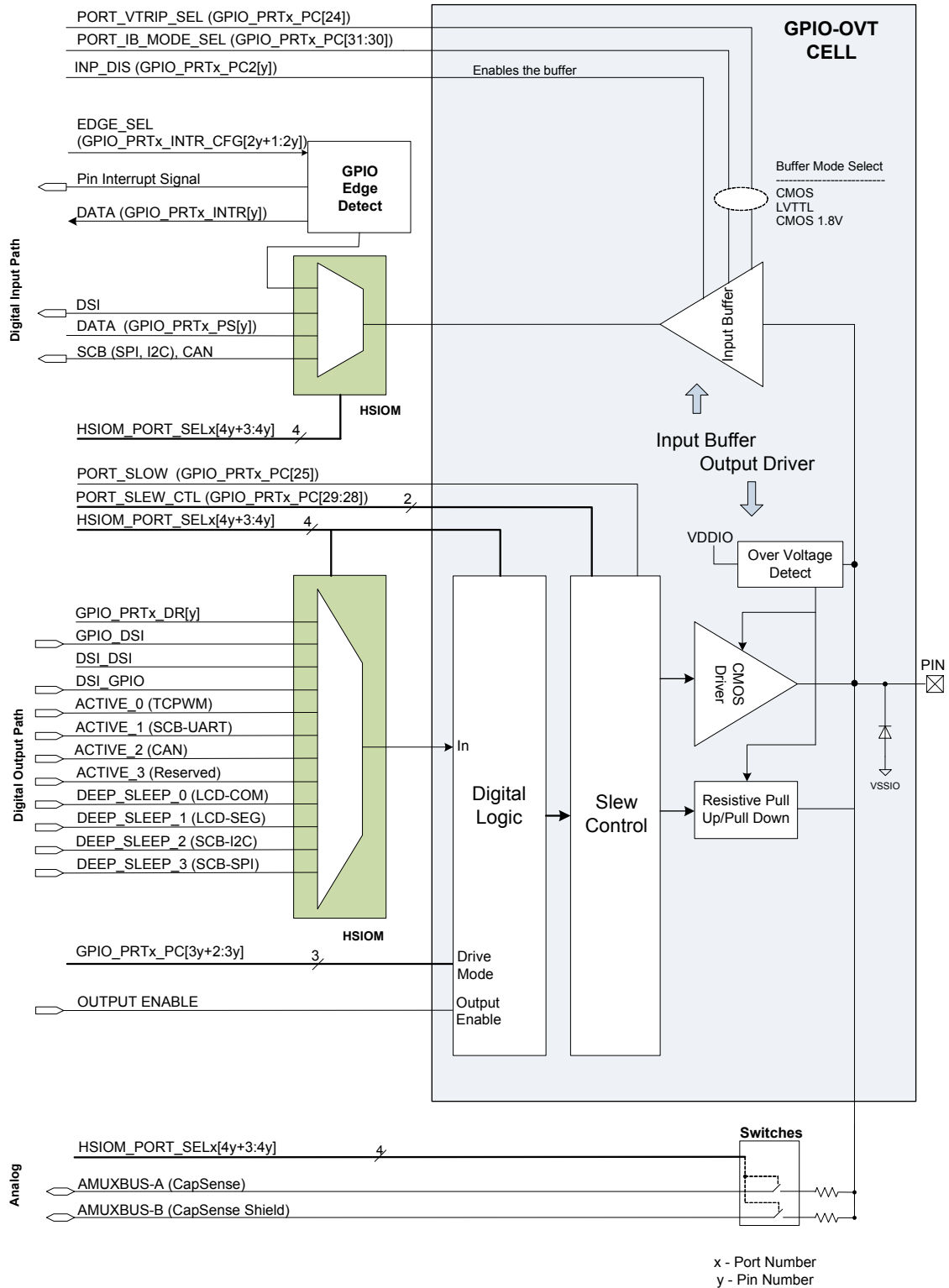
7.3.2.2 转换率控制

GPIO 引脚在强驱动模式下具有快速和慢速输出转换速率选项；通过使用端口配置寄存器（GPIO_PRTx_PC[25]）的 PORT_SLOW 位，可以配置该选项。可以单独配置每个端口的转换速率。默认情况下，该位被清除，并且端口在快速转换速率模式下运行。如果需要较低的转换速率，则可以设置该位。转换速率越慢，EMI 和串扰则越少。因此，对低频信号或没有严格时序限制的信号，推荐使用慢速转换选项。

7.4 GPIO-OVT 引脚

器件中的端口 6 和端口 8 都有过压容差 (OVT) 引脚。图 7-4 显示的是 GPIO-OVT 引脚架构。

图 7-4. PSoC 4200L 中的 GPIO-OVT 架构



它与通用的 GPIO 相似，但具有以下各附加特性：

- 过压容差 — GPIO-OVT 单元使用硬件（图 7-4 中即为“过压检测”模块）来比较 VDDIO 和引脚电压。如果引脚电压超过 VDDIO 电压，则输出驱动器被禁用，并且该引脚变成三态。这样会生成引脚上可忽略的灌电流。此外，引脚和 VSSIO 间还有 ESD 钳位二极管，可以限制负电压尖峰。

请注意，引脚上发生过压时，如果外部源的 VOH 和 VOL 规格不符合输入缓冲器的跳变点，则输入缓冲器数据无效。

- 与通用 GPIO 相比，OVT 引脚提供了更好的下拉驱动强度。
- 被配置为 I²C，并且它的各条信号线被路由到 GPIO-OVT 引脚时，则作为串行通信模块（SCB）使用。并且满足以下 I²C 规范：
 - 加强型快速模式的 IOL 规范
 - 快速模式和加强型快速模式的迟滞以及最小下降时间规范

通过额外的转换速率控制功能，可以获得最小下降时间规范。通过使用端口配置寄存器（GPIO_PRTx_PC[29:28]）的 PORT_SLEW_CTRL 位来配置该转换速率控制功能。表 7-4 显示的是 PORT_SLEW_CTRL 位的设置情况。

表 7-4. 转换速率控制

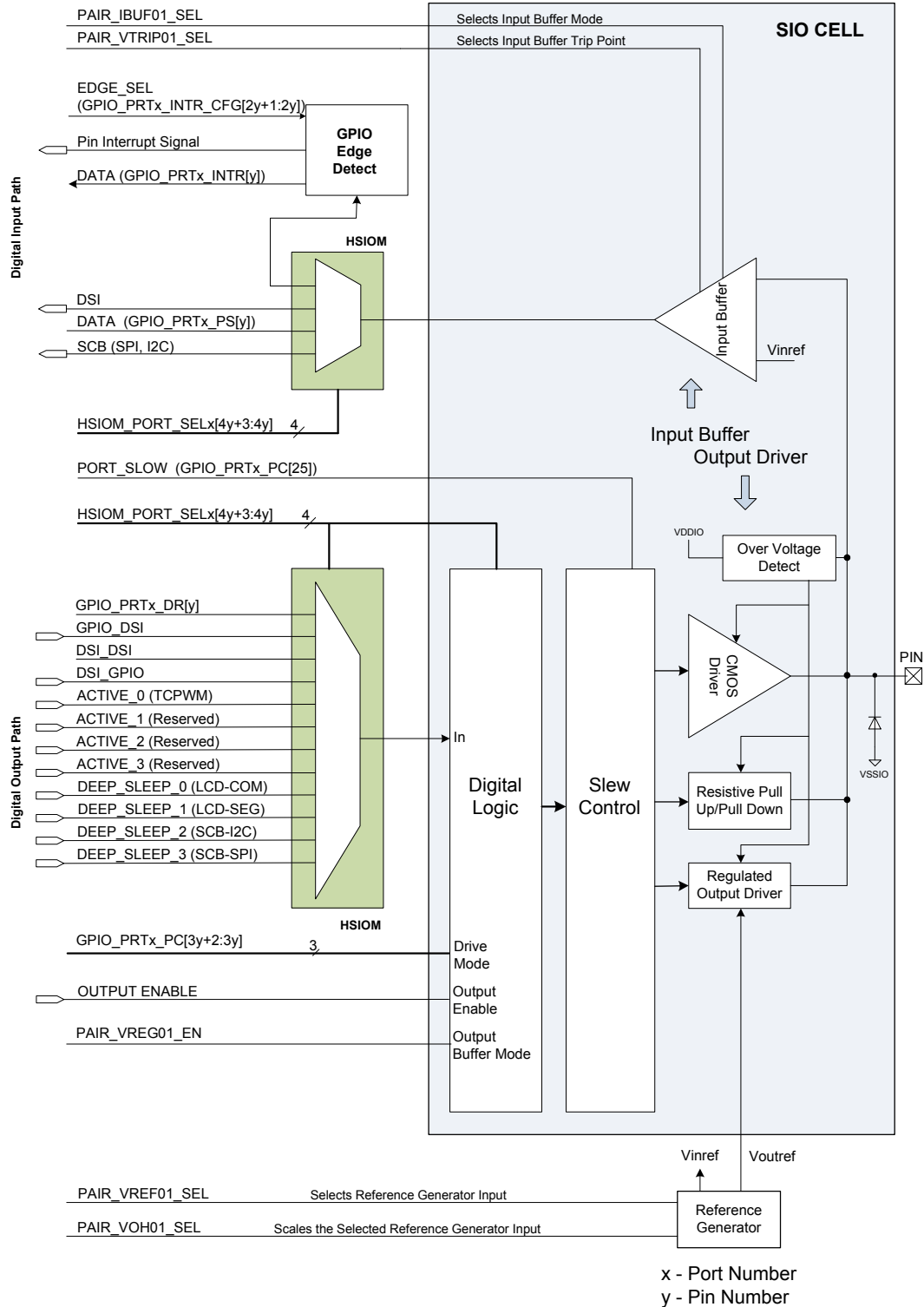
PORT_SLEW_CTRL (GPIO_PRTx_PC[29:28])	使用说明
00b	保留
01b	I ² C 增强型快速模式（FM+）适用于外部电源电压超过 2.8 V 的 I ² C 总线
10b	保留
11b	I ² C 增强型快速模式（FM+）适用于外部电源电压低于 2.8 V 的 I ² C 总线

请注意，要想使用 PORT_SLEW_CTRL 位，需要将驱动模式配置为开漏低电平驱动模式。如果选中其它驱动模式，则 PORT_SLEW_CTRL 不起任何作用。

7.5 特殊 I/O (SIO)

PSoC 4200L 提供了两个 SIO 引脚 (P12[0] 和 P12[1])。图 7-5 显示了 SIO 的架构。

图 7-5. SIO 架构



SIO 与通用的 GPIO 相似，但它具有以下附加特性：

- 参考电压发生器
- 差分输入缓冲器
- 稳压输出驱动器
- 过压容差

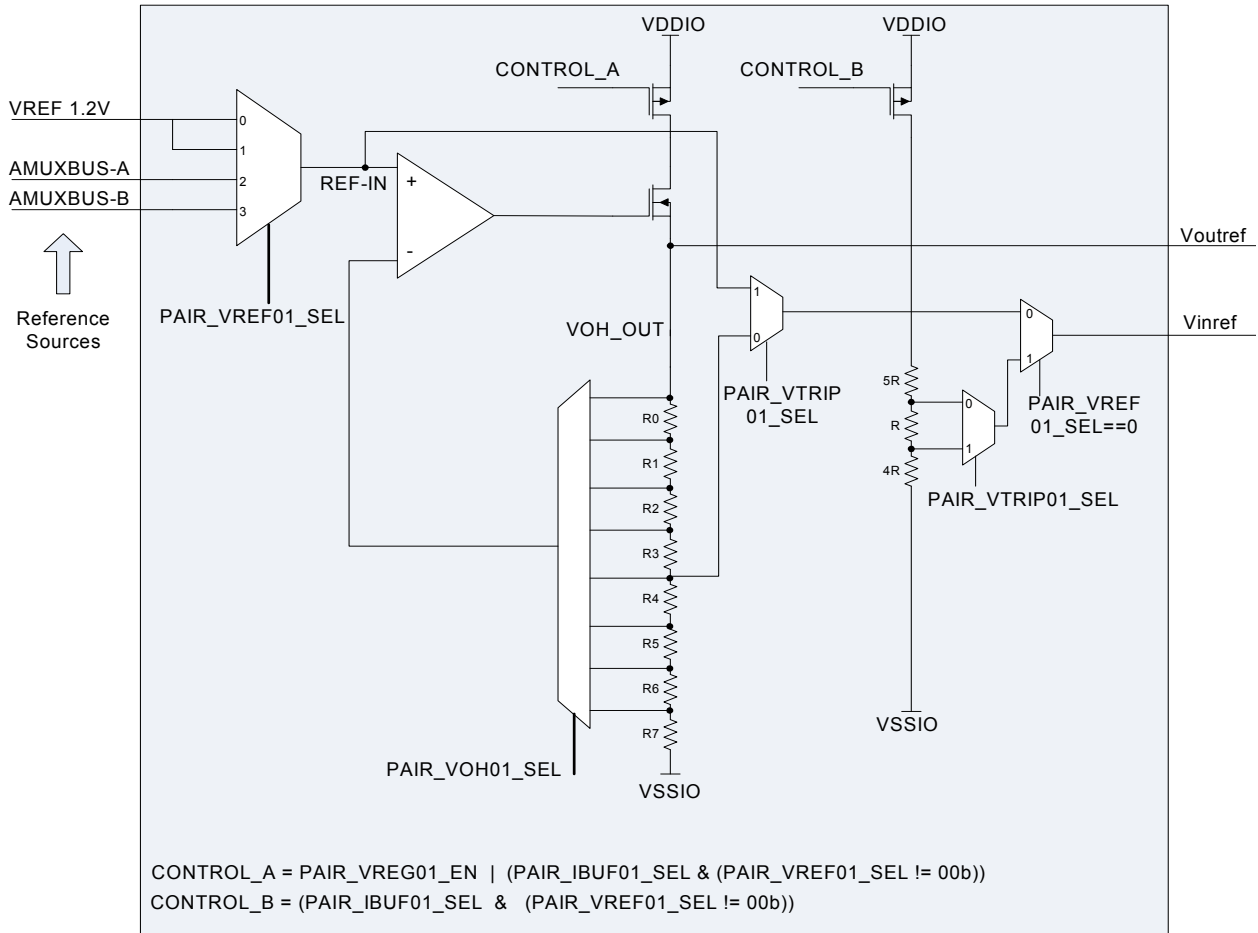
但是 SIO 引脚不支持 1.8 V CMOS 模式，并且不能与模拟外设建立任何连接。

在 PSoC 4200L 中，SIO 引脚（P12[0] 和 P12[1]）构成一个引脚对。这个 SIO 引脚对包含每个引脚的单元以及两个引脚共用的参考电压发生器。

7.5.1 参考电压发生器

参考电压发生器模块是一个被连接到负向反馈环的运算放大器，用于分别为稳压输出和跳变点设置生成输出参考电压 V_{outref} 以及输入缓冲器参考电压 V_{inref} ，如图 7-6 所示。

图 7-6. 参考电压发生器



该运算放大器的输入可以通过 SIO 寄存器中的 PAIR_VREF01_SEL 位选择，如表 7-5 所示。

表 7-5. 参考电压发生器的输入选择

PAIR_VREF01_SEL	选中的电源参考输入
00	VREF (1.2 V)
01	VREF (1.2 V)
10	AMUXBUS-A
11	AMUXBUS-B

通过使用 SIO 寄存器中的 PAIR_VOH01_SEL 位设置电阻抽头 (Rtap)，可以控制电路中的输出参考电压 Voutref/VOH_OUT。表 7-6 提供了相对于不同 PAIR_VOH01_SEL 位值的增益 (该增益是针对 PAIR_VREF01_SEL 位选择的输入)。

表 7-6. 参考电压发生器的增益选择

PAIR_VOH01_SEL	Rtap/Rtotal	增益 (Rtotal/Rtap)
000	1	1
001	0.8	1.25
010	0.67	1.49
011	0.6	1.66
100	0.48	2.08
101	0.4	2.5
110	0.36	2.77
111	0.25	4

修改反馈分频器网络中的电阻值，从而生成标准的 Voutref 电压；即：当 VREF (1.2 V) 被选为参考电压发生器的输入时，Voutref 电压分别为 1.2 V、1.5 V、1.8 V、2 V、2.5 V、3.3 V 和 4.8 V。此外，Voutref 的上限值被限制为 ($V_{DDIO} - 400 \text{ mV}$)。

使用相同的电路生成输入缓冲器的参考电压 Vinref。可使用 SIO 寄存器中的 PAIR_VREF01_SEL 和 PAIR_VTRIP01_SEL 位来设置输入缓冲器的参考电压 Vinref。表 7-7 中的 VOH_OUT 取决于由 PAIR_VREF01_SEL 位选定的源 (如表 7-5 所示) 以及由 PAIR_VOH01_SEL 位设置的增益 (如表 7-6 所示)。例如，如果 PAIR_VREF01_SEL 被设置为 10b，且 PAIR_VOH01_SEL 被设为 001b，则 VOH_OUT 等于 AMUXBUS-A 电压 x 1.25。如果 PAIR_VTRIP_SEL 被设置为 0，则 Vinref 等于 AMUXBUS-A 电压 x 1.25 x 0.5。REF-IN 是由 PAIR_VREF01_SEL 位选定的参考源 (如表 7-5 所示)。

表 7-7. Vinref 的生成

PAIR_VREF01_SEL	PAIR_VTRIP_SEL	Vinref
00	0	V_{DDIO} 的 50%
00	1	V_{DDIO} 的 40%
01/10/11	0	$0.5 \times \text{VOH_OUT}$
01/10/11	1	REF-IN

7.5.2 差分输入缓冲器

通用 GPIO 具有一个单端缓冲器，它的跳变点是通过 PORT_VTRIP_SEL 选定的，如 7.3.1 数字输入缓冲器一节所述。SIO 具有差分输入缓冲器，它的输入参考电压 Vinref 是通过参考电压发生器生成的，如 7.5.1 参考电压发生器一节所述。Vinref 用于设置数字输入缓冲器的跳变点。请参考器件数据手册，了解输入高电平电压 (VIH) 和输入低电平电压 (VIL) 的规范。

通过写入到 SIO 寄存器中的 PAIR_IBUF01_SEL 位，可以将缓冲器切换为单端模式，如表 7-8 所示。

表 7-8. 输入缓冲器模式配置

PAIR_IBUF01_SEL	输入缓冲器模式
0	单端模式
1	差分模式

请注意，在深度睡眠、休眠和停止模式下不能使用输入缓冲器的差分模式。

7.5.3 稳压输出驱动器

与标准 GPIO 相似，SIO 单元的输出驱动器也包含一个 CMOS 输出驱动器，可支持 8 种驱动模式。此外，它还具有一个稳压输出驱动器，以便在引脚上可提供各个可配置的输出电压。通过写入 SIO 寄存器中的 PAIR_VREG01_EN 位，可以将 SIO 作为 GPIO 使用，如表 7-9 所示。

表 7-9. SIO 输出驱动模式选择

PAIR_VREG01_EN	输出驱动模式
0	CMOS 输出
1	稳压输出

要想将输出驱动器设置为稳压输出模式，请将端口配置寄存器 (GPIO_PRTx_PC) 的驱动模式位配置为强驱动模式。如果选中了其他模式，则输出驱动器被配置为标准 CMOS 模式，并且稳压输出驱动器被禁用。

在稳压输出模式下，通过参考电压发生器所提供的 Voutref 来设置输出电压，如 7.5.1 参考电压发生器一节所述。表 7-10 总结了 SIO 输出配置。

表 7-10. SIO 输出驱动器配置

PAIR_VREG01_EN	PAIR_VREF01_SEL	PAIR_VOH01_SEL	SIO 配置
0	X	X	CMOS 标准模式
1	00/01	[0-7]	稳压输出模式（Voutref 由 1.2 V 的 VREF 线供电）
1	10	[0-7]	稳压输出模式（Voutref 由 AMUXBUS-A 电压线供电）
1	11	[0-7]	稳压输出模式（Voutref 由 AMUXBUS-B 电压线供电）

与通用 GPIO 相似，同样可以通过使用端口配置寄存器（GPIO_PRTx_PC[25]）中的 PORT_SLOW 位来配置 SIO 输出的转换率。请注意，在深度睡眠、休眠和停止模式下，不能在稳压输出模式中使用 SIO。

7.5.4 过压容差

SIO 的过压容差特性与 GPIO-OVT 引脚的相同。唯一区别既为它不支持 I²C 在增强型快速模式的下降时间规格。

7.6 高速输入 / 输出矩阵

高速输入 / 输出矩阵（HSIOM）是一组将 GPIO 路由到器件中外设的高速开关。当多项功能共享 GPIO，HSIOM 将复用该引脚，并将其连接到用户选定某个具体外设。HSIOM_PORT_SELx 寄存器用于选择外设。每个端口使用一个 32 位宽的寄存器，其中每个引脚占用 4 位。这样最多可为每个引脚提供 16 个不同的选项，如表 7-11 所示。

表 7-11. PSoC 4200L 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	引脚是固件控制的通用 I/O，或者被连接到专用的硬件模块。
	DR_DSI	1	输出是由固件控制的，但 OE 通过 DSI 得到控制。
	DSI_DSI	2	输出和 OE 都是通过 DSI 控制的。
	DSI_DR	3	输出是通过 DSI 控制的，OE 则是由固件控制的。
	CSD_SENSE	4	引脚是 CSD 检测引脚（模拟模式）。
	CSD_SHIELD	5	引脚是 CSD 屏蔽引脚（模拟模式）。
	AMUXA	6	引脚被连接到 AMUXBUS-A。
	AMUXB	7	引脚被连接到 AMUXBUS-B。该模式还用于 CSD I/O 充电。如果 CSD I/O 充电是通过 CSD_CONTROL 使能，那么数字 I/O 驱动器会连接到 csd_charge 信号（引脚仍连接到 AMUXBUS-B）。
	ACTIVE_0	8	特定于引脚的活动源 #0（TCPWM、EXT_CLK）
	ACTIVE_1	9	特定于引脚的活动源 #1（SCB-UART）
	ACTIVE_2	10	特定于引脚的活动源 #2（CAN）
	ACTIVE_3	11	保留
	DEEP_SLEEP_0	12	特定于引脚的深度睡眠源 #0（LCD - COM）
	DEEP_SLEEP_1	13	特定于引脚的深度睡眠源 #1（LCD - SEG）
	DEEP_SLEEP_2	14	特定于引脚的深度睡眠源 #2（SCB-I ² C、SWD、LPCOMP、WAKEUP）
	DEEP_SLEEP_3	15	特定于引脚的深度睡眠源 #3（SCB-SPI）

注意：活动和深度睡眠源由引脚决定。有关每个引脚所支持的特性信息，请参考器件数据手册的“引脚分布”一节的内容。

7.7 上电时的 I/O 状态

在上电过程中，所有 GPIO 均处于模拟高阻状态，并且输入缓冲器均被禁用。器件运行期间，可以通过写入相应的寄存器来配置 GPIO。注意：如果引脚支持调试访问端口（DAP）的接口（SWD 线），则在上电时，这些引脚始终被配置为 SWD 线。但是，可以通过 HSIOM 禁用 DAP 接口，或重新配置它，以用于通用目的。然而，只在器件启动并开始执行代码后，才能重新配置该接口。

7.8 低功耗模式下的行为

表 7-12 显示的是低功耗模式下 GPIO 的状态。

表 7-12. 在低功耗模式下的 PSoc 4200L I/O

低功耗模式	状态
睡眠	<ul style="list-style-type: none"> 标准 GPIO、GPIO-OVT 和 SIO 引脚均处于活动状态，并能够通过 CapSense、TCPWM 和 SCB 等外设驱动，这些外设可在睡眠模式下工作。 输入缓冲器均处于活动状态，因此任意 I/O 引脚上的中断都能唤醒 CPU。
深度睡眠	<ul style="list-style-type: none"> 与深度睡眠模外设连接的 GPIO、GPIO-OVT 和 SIO 引脚均可用。具有使能输出功能的其他引脚都处于冻结状态。 所有 I/O 引脚上的中断均可用。
休眠	<ul style="list-style-type: none"> 引脚输出状态被锁存，并保持为冻结状态。 所有 I/O 引脚上的中断均可用。请注意，不能将 GPIO、GPIO-OVT 和 SIO 引脚的缓冲输入配置为 1.8 V CMOS 模式。在休眠模式下，该模式不可用。
停止	<ul style="list-style-type: none"> GPIO 和 GPIO-OVT 引脚输出状态被锁存，并保持为冻结状态。 只有端口 P2[2] 上的中断可唤醒器件，因此不能将缓冲器输入配置为 1.8 V CMOS 模式。其他引脚上的输入缓冲器均无效。

7.9 输入和输出的同步

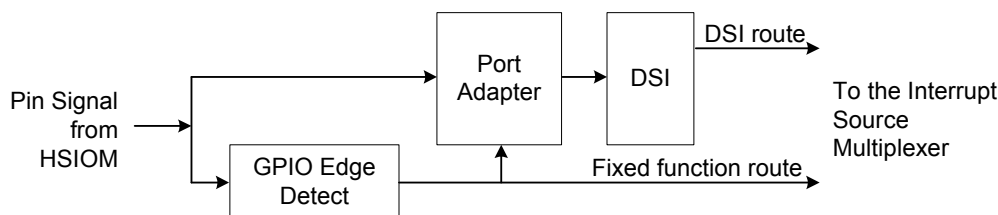
对于数字输入及输出信号，I/O 支持与内部时钟或作为时钟的数字信号进行同步。默认情况下使用 HFCLK 时钟，但仍可以使用其他时钟进行同步。

该特性可以通过使用 UDB 端口适配器实现。有关端口适配器的详细信息，请参见第 113 页上的通用数字模块（UDB）章节。

7.10 中断

在 PSoc 4 器件中，所有端口引脚都能够生成中断。通过以下三种引脚信号的路由方法，可以生成中断，如图 7-7 所示。

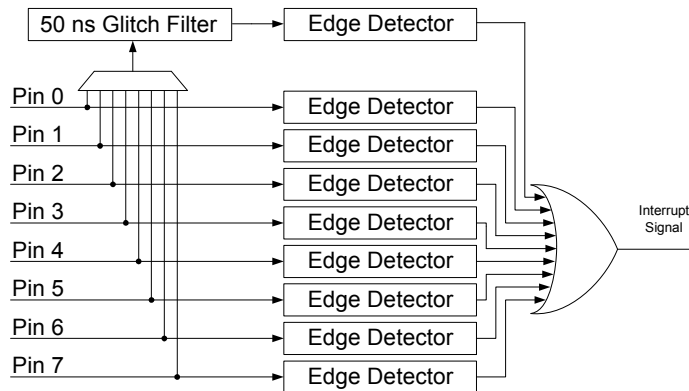
图 7-7. 中断信号路由



- 经过“GPIO 边沿检测”模块，然后直接连接至中断源复用器
- 经过“GPIO 边沿检测”模块，然后通过 DSI 连接至中断源复用器
- 引脚信号通过 DSI 连接至中断源复用器（它没有经过“GPIO 边沿检测”模块）

图 7-8 显示的是 GPIO 边沿检测模块的架构。

图 7-8. GPIO 边沿检测模块的架构



每个引脚上都有一个边沿检测器。它无需进行任何重新配置操作即可检测上升沿、下降沿和双边沿。通过写入端口中断配置寄存器 `GPIO_PRTx_INTR_CFG` 中的 `EDGE_SEL` 位字段，可以配置边沿检测器，如表 7-13 中所示。

表 7-13. 边沿检测器配置

EDGE_SEL	配置
00	中断被禁用
01	在上升沿上触发中断
10	在下降沿上触发中断
11	在双边沿上触发中断

不仅在每个引脚上使用了边沿检测器，窄脉冲滤波器输出上也使用了该检测器。端口上的每一引脚都可以使用滤波器。通过写入端口中断配置寄存器 `GPIO_PRTx_INTR_CFG` 中的 `FLT_SEL` 字段，可以选择引脚，如表 7-14 所示。

表 7-14. 窄脉冲滤波器输入选择

FLT_SEL	所选引脚
000	引脚 0 被选中
001	引脚 1 被选中
010	引脚 2 被选中
011	引脚 3 被选中
100	引脚 4 被选中
101	引脚 5 被选中
110	引脚 6 被选中
111	引脚 7 被选中

对端口的边沿检测器的输出进行 OR（或）运算，然后将结果路由到中断控制器上（即 CPU 系统中的 NVIC）。因此，每个端口只有一个中断向量。对于引脚中断，需要通过读取端口中断状态寄存器 `GPIO_PRTx_INTR` 来确定生成中断的引脚。该寄存器不仅包含了触发中断的引脚信息，它还包含了引脚的状态；因此 CPU 只要通过单一的读取操作便能够读取这两种信息。更重要的是该寄存器可用于清除中断。将 ‘1’ 写

入到相应中断状态位，即可清除引脚中断必须清除中断状态位。否则，单个触发器会重复触发中断，或多个触发器只触发一次中断。稍后会在本节介绍该内容。另请注意，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，则不能正确检测该中断。因此，当使用 GPIO 中断时，建议仅在相应的中断服务子程序中（而不是在代码的任何其他部分）读取状态寄存器。表 7-15 显示的是端口中断状态寄存器的位字段。

表 7-15. 端口中断状态寄存器

GPIO_PRTx_INTR	说明
0000b 到 0111b	引脚 0 到引脚 7 的中断状态。将 ‘1’ 写入到相应位可以清除中断
1000b	窄脉冲滤波器输出信号的中断状态
10000b 到 10111	引脚 0 到引脚 7 的中断状态
11000b	窄脉冲滤波器的输出状态

边沿检测模块的输出信号被路由到中断源复用器上（如第 55 页上的图 6-5 所示），这样便提供了电平和上升沿检测选项。选中电平选项时，只要设置了端口中断状态寄存器位，便可重复触发中断。选中上升沿检测选项时，如果端口中断状态寄存器未被清除，则只会触发一次中断。因此，使用边沿检测模块时，必须清除中断状态位。

通过固定功能模块路由中断信号时，每个端口上都有一个专用的中断向量。但是，当该信号通过 DSI 进行路由时，则中断向量非常灵活，它能够占用 NVIC 中任意 32 条中断线。有关详细信息，请参考第 51 页上的中断章节中介绍的内容。

当该信号被路由到 DSI（而旁路边沿检测模块）时，可以使用中断源复用器重新配置边沿检测选项。请注意，如果复用器将边沿触发选项配置为电平触发选项，只要引脚信号为高电平，便能重复触发中断。选择这种路由方法时，推荐使用上升沿检测选项。

7.11 外设连接

7.11.1 固件控制的 GPIO

请查看表 7-11，以了解由固件控制的 GPIO 的 HSIOM 设置。GPIO_PRTx_DR 是数据寄存器，用于读写 GPIO 的输出数据。对该寄存器进行写操作可将 GPIO 输出修改为所写入的值。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 的端口上，可以安全执行读 - 修改 - 写序列。

除了数据寄存器外，其他三个寄存器（GPIO_PRTx_DR_SET、GPIO_PRTx_DR_CLR 和 GPIO_PRTx_INV）分别用于设置、清除和反转端口上特定引脚的输出数据，并不会对其他引脚产生影响。将 ‘1’ 写入到这些寄存器可设置、清除或反转数据；写入 ‘0’ 时不会影响引脚的状态。

GPIO_PRTx_PS 是端口 I/O 焊盘寄存器；读取该寄存器时，可获取 GPIO 的状态。无法对该寄存器进行写操作。

7.11.2 模拟 I/O

需要低阻抗路由路径的模拟资源（如 LPCOMP、SARMUX 和 CTBm）都有专用引脚。专用的模拟引脚可以直接连接到特定的模拟模块。它们有助于提高性能；当使用这些模拟资源时，应优先使用这些专用引脚。有关 PSoC 4 的专用引脚的详细信息，请参阅 [器件数据手册](#)。

为了将一个 GPIO 配置为专用模拟 I/O，应将它配置为模拟高阻态模式（请参考表 7-3），并且在特定的模拟资源中使能相应连接。通过使用与相应模拟资源相关联的寄存器，可以实现该操作。

为了将一个 GPIO 配置为连接至 AMUXBUS 的模拟引脚，需要将它配置为模拟高阻态模式，然后通过 HSIOM_PORT_SELx 寄存器路由到 AMUXBUS。

7.11.2.1 AMUXBUS 连接和 DSI

在支持 DSI 连接的端口中，将某个引脚连接到 AMUXBUS A/B 时，除了需要配置 HSIOM_PORT_SELx 寄存器外，用户还要配置一对 DSI 信号。应将引脚的 DSI 输出和 DSI 输出使能信号设置为高电平，以使能 AMUXBUS 连接。这样能够使用 DSI 信号来控制引脚的 AMUXBUS 连接，因此用户可以通过 DSI 实现 AMUXBUS 的硬件功能（即切换信号）。要正确将某个引脚配置为 AMUXBUS 输入，需要执行以下步骤：

1. 在设计中，将引脚的 DSI 输出和 DSI 输出使能信号连接至逻辑 ‘1’，以进行静态 AMUXBUS 连接。如果设计中要求动态 AMUXBUS 连接，可以将这些信号连接至 DSI 选定的信号上。可以在器件中进行该操作。
2. 配置 GPIO_PRTx_PC 寄存器，以便将引脚设置为模拟高阻态模式，从而可以通过禁用输入缓冲器来使能该模拟连接。
3. 配置 HSIOM_PRT_SELx 寄存器，以将该引脚连接到 AMUXBUS A/B。

7.11.3 LCD 驱动

所有 GPIO 都具有驱动某个 LCD 的共模信号和段信号的能力。HSIOM_PORT_SELx 寄存器用于选择 LCD 驱动使用的引脚。有关详细信息，请参考第 263 页上的 [LCD 直接驱动章节](#) 内容。

7.11.4 CapSense

可将支持 CSD 的引脚配置为 CapSense Widgets，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 7-16 显示了 CapSense 所需的 GPIO 和 HSIOM 设置。相关详细信息，请参见第 275 页上的 [CapSense 章节](#)。

表 7-16. 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	模拟高阻态	禁用缓冲器	CSD_SENSE
屏蔽	模拟高阻态	禁用缓冲器	CSD_SHIELD
CMOD（正常操作）	模拟高阻态	禁用缓冲器	AMUXBUS A 或 CSD_COMP
CMOD（GPIO 预充电，仅适用于选择 GPIO）	模拟高阻态	禁用缓冲器	AMUXBUS B 或 CSD_COMP
CSH TANK（GPIO 预充电，仅适用于选择 GPIO）	模拟高阻态	禁用缓冲器	AMUXBUS B 或 CSD_COMP

7.11.5 串行通信模块（SCB）

可将 SCB 配置为 UART、I²C 和 SPI。可单独将 SCB 连接到该引脚。有关 PSoC 4 的专用引脚的详细信息，请参阅[器件数据手册](#)。使用 UART 和 SPI 模式时，SCB 将控制该输入引脚的数字输出缓冲器驱动模式，从而使该引脚保持为高阻态。SCB 模块禁用 UART Rx 引脚上的输出缓冲器。如果该模块被配置为 SPI 主设备，它将禁用 MISO 引脚上的输出缓冲器。如果被配置为 SPI 从设备，则禁用 MOSI 引脚和选择线引脚上的输出缓冲器。这样会更改驱动模式的设置，该操作是通过使用 GPIO_PRTx_PC 寄存器来完成的。

7.12 端口限制

端口 7、端口 8、端口 9 和端口 10 的引脚都没有端口适配器，因此存在以下限制：

- 不能通过 DSI 路由。因此，基于 UDB 的数字信号不能路由到这些端口的引脚上。
- 无输入 / 输出同步

但是，这些端口可以用于以下情况：

- 作为由固件控制的 GPIO 引脚使用
- 直接连接到 TCPWM、SCB 或 CAN
- 作为 LCD 和 CapSense 引脚使用
- 中断生成

7.13 寄存器

表 7-17. 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：GPIO 寄存器名称中的 ‘x’ 表示端口编号。例如，GPIO_PTR1_DR 是端口 1 的输出数据寄存器。

8. 时钟系统



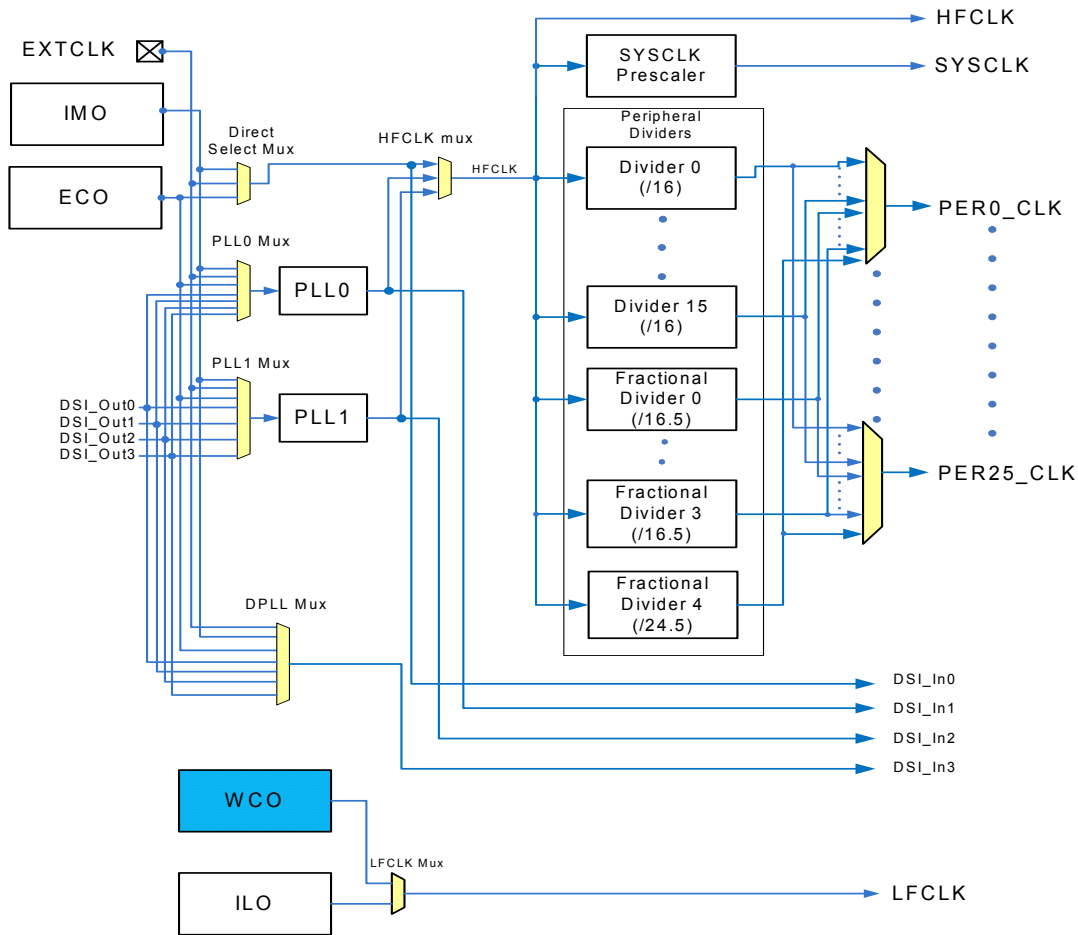
PSoC® 4 时钟系统包括以下时钟源：

- 两个内部时钟源：
 - 频率为 3 ~ 48 MHz 的内部主振荡器（IMO），所有频率的调整范围为 $\pm 2\%$
 - 32 kHz 内部低速振荡器（ILO），调整范围为 $\pm 60\%$ （可以使用 IMO 进行校准）
- 三个外部时钟源：
 - 通过使用 I/O 引脚的信号生成的外部时钟（EXTCLK）
 - 外部晶振（ECO）的频率范围为 4 MHz 到 33 MHz
 - 外部 32 kHz 监视晶振（WCO）
- 从 IMO、ECO、外部时钟或一个 PLL 中选择频率高达 49.152 MHz 的高频时钟（HFCLK）
- 来自 ILO 或 WCO 的低频时钟（LFCLK）
- 专用预分频器使用于来自 HFCLK 的频率高达 49.152 MHz 的系统时钟（SYSCLK）
- IMO、ECO 或外部时钟通过两个 PLL 生成高达 49.152 MHz 输出
- 16 个 16 位外设时钟分频器
- 5 个小数分频器（4 个 16.5 小数分频器和一个 24.5 小数分频器）用于生成准确的时钟
- 25 个 数字和模拟外设时钟

8.1 框图

图 8-1 提供 PSoC 4 器件中时钟系统的通用视图。

图 8-1. 时钟系统框图



该器件的 5 个时钟源包括 IMO、ECO、EXTCLK、WCO、和 ILO，如图 8-1 所示。通过配置两个 PLL，可以从 IMO、ECO、EXTCLK 或一个 DSI_out 时钟中提取时钟源。使用 DSI_out 时钟来将时钟信号通过 DSI 路由到可编程数字逻辑。PLL 输出频率可以高达 108 MHz。虽然 PLL 输出频率可以高达 108 MHz，但不是所有模块都接受高于 48 MHz 的频率。更多详细信息，请参见第 81 页上的 8.2.5 PLL0 和 PLL1。HFCLK 复用器从 EXTCLK、ECO、PLL0、PLL1 或 IMO 中选出一个作为 HFCLK 源。HFCLK 的最大频率可达 49.152 MHz。SYSCLK 预分频器生成 SYSCLK，而外设分频器则生成单独的外设时钟。该器件具有 16 个积分分频器和 4 个小数分频器。时钟树具有 25 个时钟输出，这些输出被路由到器件的特定外设。可从 16 个积分分频器输出或 5 个小数分频器输出中选出任意一个作为时钟输出。更多详细信息，请参见第 83 页上的时钟分布。时钟系统具有 4 个路由到 DSI 的时钟输出。这些时钟输出被称为 DSI_In 时钟信号，它们被传送到 DSI，为数字电路提供时钟脉冲。LFCLK 复用器从 WCO 或 ILO 中选择 LFCLK。LFCLK 作为看门狗定时器中一个定时器的时钟源。有关详细信息，请参考第 101 页上的看门狗定时器章节中介绍的内容。

8.2 时钟源

8.2.1 内部主振荡器

内部主振荡器在运行时不需要外部组件，并可输出一个稳定的时钟，其频率范围为 3 ~ 48 MHz，步长为 1 MHz。通过在寄存器 CLK_IMO_TRIM2 中设置频率、在寄存器 CLK_IMO_TRIM1 中设置 IMO 调整，最后在寄存器 PWR_BG_TRIM4 和 PWR_BG_TRIM5 中设置带隙调整，可实现频率选择。在 CLK_IMO_TRIM2 中设置的频率决定了 IMO 频率输出。表 8-1 提供了与 IMO 频率输出相对应的设置。除了在 CLK_IMO_TRIM2 中设置频率外，用户还需要加载 CLK_IMO_TRIM1、PWR_BG_TRIM4 和 PWR_BG_TRIM5 中各相应的调整值。选择频率时需要执行一个算法，从而确保没有任何中间状态被编程为高于 48 MHz 的频率。每个 PSoC 器件都有在制造过程中测量的 IMO 调整设置，以满足数据手册的要求；此调整被存储在 SFLASH 中的制造配置数据中。这些器件具有的 TRIM（调整）值与用户选定的频率相对应。SFLASH 中的 TRIM 值被加载到相应的调整寄存器（CLK_IMO_TRIM1、PWR_BG_TRIM4 和 PWR_BG_TRIM5）中。可以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。

想要配置 IMO 频率，请遵循下面的算法：

- 如果（（新频率 ≥ 43 MHz），并且（旧频率 ≥ 43 MHz）），那么要将 CLK_IMO_TRIM2 的频率更改为一个更低的值，如 24 MHz
将 CLK_IMO_TRIM1、PWR_BG_TRIM4 以及 PWR_BG_TRIM5 应用于新频率
等待 ≥ 5 us
将 CLK_IMO_TRIM2 更改成新频率
- 另外，如果（新频率 $>$ 旧频率），
将 CLK_IMO_TRIM1、PWR_BG_TRIM4 以及 PWR_BG_TRIM5 应用于新频率
等待 ≥ 5 us
将 CLK_IMO_TRIM2 更改成新频率
- 另外，
将 CLK_IMO_TRIM2 更改成新频率
等待 ≥ 5 个周期
将 CLK_IMO_TRIM1、PWR_BG_TRIM4 以及 PWR_BG_TRIM5 应用于新频率

表 8-1. IMO 频率配置

CLK_IMO_TRIM2						频率（MHz）
位 5	位 4	位 3	位 2	位 1	位 0	
0	0	0	0	1	1	3
0	0	0	1	0	0	4
0	0	0	1	0	1	5
0	0	0	1	1	0	6
0	0	0	1	1	1	7
0	0	1	0	0	0	8
0	0	1	0	0	1	9
0	0	1	0	1	0	10
0	0	1	0	1	1	11
0	0	1	1	0	0	12
0	0	1	1	1	0	13
0	0	1	1	1	1	14
0	1	0	0	0	0	15
0	1	0	0	0	1	16
0	1	0	0	1	0	17
0	1	0	0	1	1	18
0	1	0	1	0	0	19
0	1	0	1	0	1	20
0	1	0	1	1	0	21
0	1	0	1	1	1	22
0	1	1	0	0	0	23
0	1	1	0	0	1	24
0	1	1	0	1	1	25
0	1	1	1	0	0	26
0	1	1	1	0	1	27
0	1	1	1	1	0	28
0	1	1	1	1	1	29

表 8-1. IMO 频率配置（续）

CLK_IMO_TRIM2						频率（MHz）
位 5	位 4	位 3	位 2	位 1	位 0	
1	0	0	0	0	0	30
1	0	0	0	0	1	31
1	0	0	0	1	0	32
1	0	0	0	1	1	33
1	0	0	1	0	1	34
1	0	0	1	1	0	35
1	0	0	1	1	1	36
1	0	1	0	0	0	37
1	0	1	0	0	1	38
1	0	1	0	1	0	39
1	0	1	0	1	1	40
1	0	1	1	1	0	41
1	0	1	1	1	1	42
1	1	0	0	0	0	43
1	1	0	0	0	1	44
1	1	0	0	1	0	45
1	1	0	0	1	1	46
1	1	0	1	0	0	47
1	1	0	1	0	1	48

8.2.1.1 启动行为

复位后，IMO 被配置为 24 MHz。在启动的“引导”过程中，可从闪存读取调整值，并配置 IMO，以满足数据手册指定的准确度。

8.2.1.2 IMO 频率扩展

IMO 能在扩频模式下运行，以减少在 IMO 中心工作频率中生成的噪声振幅。该模式使 IMO 频率在寄存器选择的四种分布之一不相同。四种分布包括：固定频率、三角波、伪随机及 DSI 输入。DSI 输入模式允许您使用数字信号来指定 IMO 频率分布的形状。分布形状由寄存器 CLK_IMO_SPREAD 位 SS_MODE 选择（如表 8-2 所示）。分布限制由寄存器 CLK_IMO_SPREAD 位 SS_RANGE 指定（如表 8-3 所示）。所有扩展选项均为向下扩展，表示瞬间时钟频率值总是等于或低于配置频率。

表 8-2. IMO 扩频分布模式位 SS_MODE

名称	说明
SS_MODE[1:0]	IMO 扩频模式。定义扩频频率分布的形状。 0: 关闭。IMO 频率不变。 1: 三角形。IMO 频率根据中心频率形成一个三角形的分布。计数限制由位 SS_MAX 定义。 2: 使用 LFSR 的伪随机序列。IMO 频率根据中心频率形成一个伪随机分布。 3: DSI。通过使用 DSI 输入信号可确定 IMO 频率分布。

表 8-3. IMO 扩频分布范围位 SS_RANGE

名称	说明
SS_RANGE[1:0]	IMO 扩频的最大范围。定义了扩频计数器到达极限计数值时标称频率的扩展。 0: 1%。扩频变化在极限计数值上的频率范围为 0 至 -1%。 1: 2%。扩频变化在极限计数值上的频率范围为 0 至 -2%。 2: 4%。扩频变化在极限计数值上的频率范围为 0 至 -4%。 3: 保留。请勿使用。

CLK_IMO_SPREAD 寄存器中的 SS_MAX 字段设置了扩频计数器的最大计数值。当将 SS_MODE 设置为一个三角形扩频时，增大该值会延长三角形扩频整个周期时间。

IMO 扩频逻辑需要一个与它连接的时钟，以便获取 IMO 扩频性能。该逻辑将外设时钟 0 作为它的时钟。IMO 扩频需要外设时钟 0 被路由到外设时钟分频器的相应时钟。该时钟的频率决定了扩频逻辑的速率以及频率的变化率。

8.2.1.3 编程时钟 (36 MHz)

IMO 模块具有一个 36 MHz 的输出，该输出作为闪存编程模块的时钟使用。该时钟仅适用于闪存编程模块，并不能作为任意时钟分频器或时钟树的时钟源。

8.2.2 内部低速振荡器

内部低速振荡器运行时不需要外部组件，并会输出一个 32 kHz 的稳定时钟。ILO 是功耗相对低且准确度较低的振荡器。通过使用准确度更高并且频率更高的时钟，可以通过校准来提高该振荡器的准确度。该 ILO 在所有功耗模式（休眠模式及停止模式除外）下均可用。在 PSoC 4 中，ILO 作为系统低频率时钟使用。该 ILO 是一个相对不够准确（过压和过温时 ±60%）的振荡器，可用于生成低频率时钟。如果使用 IMO 进行了校准，则在稳定的温度和电压条件下，ILO 的准确度可达 ±10%。通过使用寄存器 CLK_ILO_CONFIG 中的位 ENABLE 使能及禁用 ILO。

8.2.3 外部时钟 (EXTCLK)

外部时钟 (EXTCLK) 是 MHz 范围的时钟，其可由设计 PSoC 4 引脚上的信号生成。该时钟可取代 IMO，作为系统高频率时钟 HFCLK 源使用。外部时钟频率的可用范围为 0 ~ 48 MHz。PSoC 4 始终使用 IMO 启动，并且必须在用户模式下使能外部时钟。因此，不可通过使用外部时钟提供时钟脉冲的复位信号来启动器件。

当手动将引脚配置为 EXTCLK 的输入时，必须将引脚的驱动模式设置为数字高阻态，以使能数字输入缓冲区。有关更多信息，请参考第 63 页上的 I/O 系统章节。

8.2.4 外部晶体振荡器 (ECO)

PSoC 4 器件具有一个振荡器，用于驱动频率在 4 到 33.33 MHz 范围内的外部晶振。通过使用 PSoC 中的振荡器电路来创建该时钟源。该电路还要使用另一个外部振荡器，该振荡器被组装在 PSoC 的外部晶体引脚上。

通过 ECO_CONFIG.CLK_EN（位 0）和 ECO_CONFIG.ENABLE（位 31）寄存器位字段，可以使能 ECO。

8.2.4.1 ECO 调整

ECO 支持各种晶体和陶瓷谐振器，其额定频率范围为 $f = 4$ MHz 至 33.333 MHz。晶体制造商通常提供了参数值（即最大驱动电平 (DL)、等效串联电阻 (ESR) 以及并行负载电容 (C_L)）。使用这些参数可计算跨导 (g_m) 和最大峰峰值 (V_{PP})。

最大峰峰值：

$$V_{PP} = 2 \times \sqrt{\frac{2 \times DL}{ESR}} \times \frac{1}{4\pi \times f \times C_L}$$

跨导：

$$g_m = 4 \times 5 \times (2\pi \times f \times C_L)^2 \times ESR$$

ECO 不支持 $V_{PP} < 0.4$ V。同样， g_m 必须小于 18 mA/V。

ATRIM 和 WDTRIM 设置控制着振荡器输出幅度的调整。当使能 AGC (ECO_CONFIG.AGC_EN = 1) 时, 幅度调整 (ATRIM) 将设置晶体驱动电平。V_{PP} < 2 V 时, 必须使能 AGC, 并且在其他所有情况中都禁用了 AGC。根据 V_{PP} 值, 将 ATRIM 和 WDTRIM 值设置为表 8-4 所示的值。

表 8-4. ATRIM 和 WDTRIM 设置

VPP	ATRIM	WDTRIM
0.4 V ≤ V _{PP} < 0.5 V	0x00	0x00
V _{PP} < 0.6 V	0x01	0x00
V _{PP} < 0.7 V	0x02	0x01
V _{PP} < 0.8 V	0x03	0x01
V _{PP} < 0.9 V	0x04	0x02
V _{PP} < 1.025 V	0x05	0x02
V _{PP} < 1.15 V	0x06	0x03
V _{PP} < 1.275 V	0x07	0x03

GTRIM 用于设置放大器增益的调整, 并根据表 8-5 中计算的 g_m 来设置 GTRIM。

表 8-5. GTRIM 设置

G _m	GTRIM
0 mA/V < g _m ≤ 4.5 mA/V	0x00
4.5 mA/V < g _m ≤ 9 mA/V	0x01
9 mA/V < g _m ≤ 13.5 mA/V	0x02
13.5 mA/V < g _m ≤ 18 mA/V	0x03

FTRIM 和 RTRIM 用于设置滤波器特性的调整, 并根据表 8-6 中计算出的 g_m 设置它们。

表 8-6. FTRIM 和 RTRIM 设置

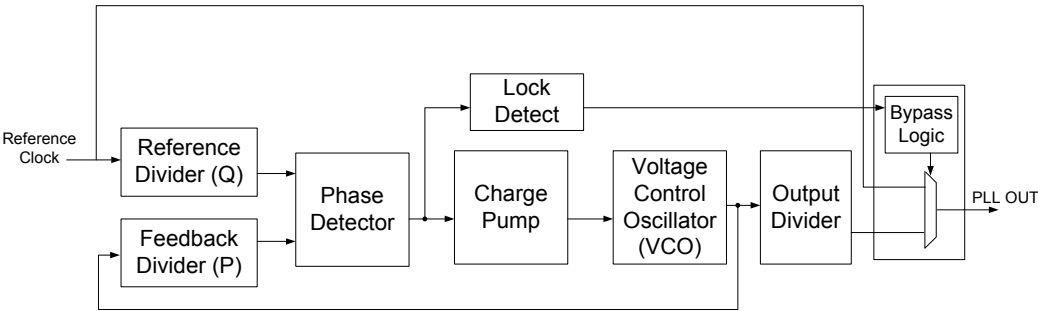
额定频率 f (MHz)	RTRIM	FTRIM
f > 30 MHz	0x00	0x00
30 MHz ≥ f > 24 MHz	0x01	0x01
24 MHz ≥ f > 17 MHz	0x02	0x02
17 MHz ≥ f	0x03	0x03

8.2.5 PLL0 和 PLL1

PSoC 器件使用两个 PLL。PLL 支持可灵活配置的频率合成。对 PLL 的配置, 主要通过 P 和 Q 这两项的设置完成。其中, P 是设置 PLL 的反馈分频器, 而 Q 是参考时钟分频器。因此, PLL 的输出频率为参考时钟的 P/Q 倍。

PLL0_CONFIG 和 PLL1_CONFIG 分别用于配置 PLL0 和 PLL1 的 P 和 Q。在该寄存器中, PLLx_CONFIG[7:0] 是 P 值, PLLx_CONFIG[13:8] 是 Q 值。通过改变反馈计数器 (P 具有 8 位分辨率) 和参考计数器 (Q 有 6 位分辨率) 中这个整数值, PLL 可以从输入参考时钟频率 (clk_ref) 合成大量输出频率。

图 8-2. PLL 框图



PLL 具有一个集成输出分频器，该分频器允许在更高的频率下运行 VCO。这样在同一个频率下，与未分频的输出相比，抖动和占空比更得到改善。输出分频器具有四个由 PLLx_CONFIG[15:14] 配置的后分频选项，如表 8-7 中所示。更改分频器的分频值时，PLL 输出会出现短时脉冲。更改分频器的分频值前，用户需要更改 PLL 路由的目标。

表 8-7. 输出分频器选项

PLL_CONFIG[15:14]	PLL 频率
00	FVCO （分频器被旁路）
01	FVCO/2
10	FVCO/4
11	FVCO/8

PLL 的电荷泵具有三个配置位。使用这些位设置 PLL 的电荷泵电流。该配置根据频率预设置，并不建议用于用户操作。

锁定检测块用于指出 PLL 处于“锁定”状态的时间。通过 PLL_STATUS 寄存器可以观察锁定状态。

PLL 还具有旁路逻辑（用于旁路 PLL 模块），并且具有参考时钟流（被连接到 PLL 输出）。通过 PLLx_CONFIG[21:20] 配置该性能。旁路逻辑具有三种工作模式：

- **AUTO/AUTO1:** 当检测到锁定状态时，PLL 输出被路由；否则，PLL 被旁路。
- **PLL_REF:** 始终旁路 PLL，并忽略锁定检测。
- **PLL_OUT:** 正常工作模式。使用 PLL 来锁定输出。

8.2.6 监视晶体振荡器（WCO）

PSoC 器件包含一个振荡器，用于驱动 32.768 KHz 的监视晶振。它还作为 LFCLK 的源使用。与 ILO 相同，WCO 在所有模式（休眠和停止模式除外）下均可用。该时钟的功耗较低，因此它非常适合低功耗模式（如深度睡眠模式）。通过 WCO_CONFIG 寄存器的 ENABLE 位使能及禁用 WCO。

通过设置 WCO_CONFIG[0] 位，可以迫使 WCO 进入低功耗模式。另外，可以将该模块设置为自动模式，这样只有器件进入深度睡眠模式时，才发生低功耗模式转换。通过设置 WCO_CONFIG[1] 使能该模式。请注意，如果通过设置 WCO_CONFIG[0] 迫使该模块进入低功耗模式，则自动模式将被覆盖。自动模式将根据器件的功耗模式在 WCO 的正常模式和低功耗模式间切换。在切换过程中，WCO 输出会遇到一些频率干扰。因此，不建议在高精度应用（如 RTC）中使用自动模式。

在放大器增益上，正常模式和低功耗模式间存在差别。低功耗模式需要较低的放大器增益，从而有效地降低功耗。通过 WCO_TRIM 寄存器，可以设置这两种模式的放大器增益。

IMO 支持锁定 WCO。WCO 包含了用于测量和比较 IMO 时钟以及调整 IMO 的逻辑。WCO 实现了数字锁相环方案，以支持误差为 $\pm 1\%$ 的时钟。通过使用 WCO_CONFIG 的 DPPLL_ENABLE 位可以使能 WCO 的 IMO 调整逻辑。如果使用该性能，用户固件必须确保从使能 WCO 到 DPPLL_ENABLE 事件时间最小为 500 ms。

8.3 时钟分布

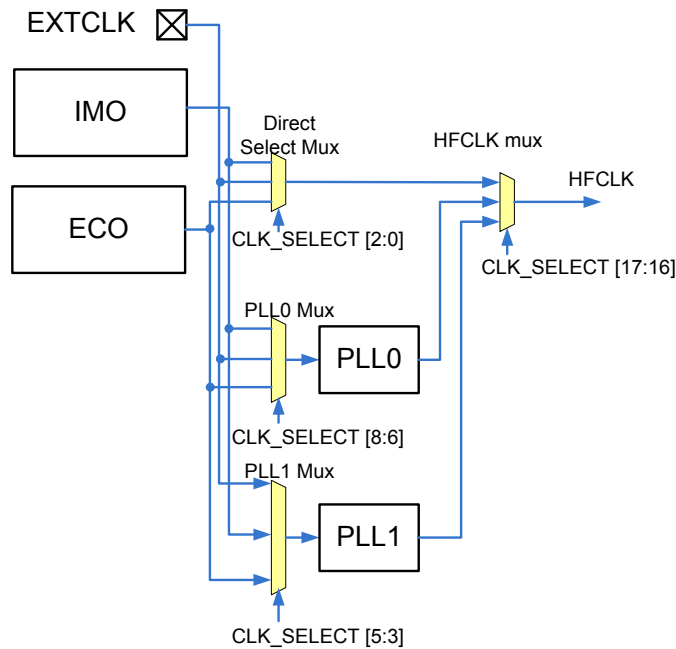
PSoC 4 时钟在整个器件中开发和分布，如图 8-1 所示。分布配置有以下的选项：

- HFCLK 输入选择
- PLL0 输入选择
- PLL1 输入选择
- LFCLK 输入选择
- SYSCLK 预分频器配置
- 外设分频器配置

8.3.1 HFCLK 和 PLL 输入选择

图 8-3 显示了 HFCLK 和 PLL 的选择项。

图 8-3. HFCLK 和 PLL 的选择项



PLL0 和 PLL1 具有多个复用器，通过这些复用器选择 EXTCLK、ECO、IMO 以及四个 DSI 输出信号。PLL0 的输入选择位于 CLK_SELECT[8:6] 寄存器中。PLL1 的输入选择位于 CLK_SELECT[5:3] 寄存器中。虽然图 8-1 中的拓扑结构会使 DSI_Out 信号（通过 PLL#0/1）生成 HFCLK，但在实际使用中不允许执行该操作。将 DSI_Out 信号传入 PLL#0/1 模块，以创建用于 UDB 用户模块的同步时钟。

对于 HFCLK，它是一个两级选项。通过使用直接选择复用器（其选择被配置在 CLK_SELECT[2:0] 中）首次复用 ECO、EXTCLK 以及 IMO 三个源。称为 HFCLK 复用器的第二个复用器用于选择直接选择输出和 PLL。通过 CLK_SELECT[17:16] 寄存器配置该选择。

8.3.2 HFCLK 输入选择

PSoc 4 中的 HFCLK 共有 5 个输入选项：IMO、EXTCLK、PLL0、PLL1 及 ECO。在一个分级中，首次选择 IMO、EXTCLK 及 ECO。通过使用 CLK_SELECT 寄存器的 DIRECT_SEL 位选择 HFCLK 输入，如表 8-8 所述。

表 8-8. 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: ECO。ECO 作为 HFCLK 源使用 3-7: 保留。请勿使用

HFCLK 会在另一个分级中选择，它是从 DIRECT_SEL、PLL0 以及 PLL1 的混合输出选出的。通过 CLK_SELECT 寄存器的 HFCLK_SEL 位使能该选择，如表 8-9 所示。

表 8-9. CLK_SELECT 的选择

名称	说明
CLK_SELECT [17:16]	HFCLK 输入时钟选择 0: 根据 CLK_SELECT[2:0] 的设置情况选择 HFCLK 源 1: PLL1 作为 HFCLK 源 2: PLL0 作为 HFCLK 源

8.3.3 PLL 输入选择

PSoC 4 中的 PLL 共有 7 个输入选项：IMO、ECO、EXTCLK、以及四个 DSI 源。当 PLL 作为 HFCLK 的源时，DSI_Out 信号不是有效的选择。通过使用 CLK_SELECT 寄存器选择 PLL 源，如表 8-10 所示。

表 8-10. PLL 输入选择位

名称	说明
CLL_SELECT[5:3]	PLL1 输入时钟选择 0: IMO。IMO 作为 PLL1 源使用 1: EXTCLK。外部时钟作为 PLL1 源使用 2: ECO。外部晶体振荡器作为 PLL1 源使用 4: DSI_out[0] 作为 PLL1 源使用（当 PLL 为 HFCLK 提供时钟脉冲时，不再使用 DSI_out[0]） 5: DSI_out[1] 作为 PLL1 源使用（当 PLL 为 HFCLK 提供时钟时，不再使用 DSI_out[1]） 6: DSI_out[2] 作为 PLL1 源使用（当 PLL 为 HFCLK 提供时钟时，不再使用 DSI_out[2]） 7: DSI_out[3] 作为 PLL1 源使用（当 PLL 为 HFCLK 提供时钟时，不再使用 DSI_out[3]）
CLL_SELECT[8:6]	PLL0 输入时钟选择 0: IMO。IMO 作为 PLL0 源使用 1: EXTCLK。外部时钟作为 PLL0 源使用 2: ECO。外部晶体振荡器作为 PLL0 源使用 4: DSI_out[0] 作为 PLL0 源使用（当 PLL 为 HFCLK 提供时钟时，不再使用 DSI_out[0]） 5: DSI_out[1] 作为 PLL0 源使用（当 PLL 为 HFCLK 提供时钟时，不再使用 DSI_out[1]） 6: DSI_out[2] 作为 PLL0 源使用（当 PLL 为 HFCLK 提供时钟时，不再使用 DSI_out[2]） 7: DSI_out[3] 作为 PLL0 源使用（当 PLL 为 HFCLK 提供时钟时，不再使用 DSI_out[3]）

8.3.4 LFCLK 输入选择

PSoC 4 的 LFCLK 具有两个输入选项：ILO 和 WCO。通过使用 WDT_CONFIG 寄存器的 LFCLK_SEL 位选择 LFCLK，如表 8-11 所示。如果在 LFCLK 边沿附近更改了该选择，那么 LFCLK 选择可能出现故障。LFCLK 为看门狗定时器（WDT）提供时

钟脉冲；因此，要确保 LFCLK 源的切换不会影响到 WDT。为了安全更改 LFCLK_SEL，需要等待 WDT_CTRL0W/ WDT_CTRLHIGH 更改，然后立即更改该设置。

表 8-11. LFCLK 输入选择位 LFCLK_SEL

名称	说明
LFCLK_SEL[1:0]	<p>LFCLK 输入时钟选择</p> <p>0: ILO。内部本机振荡器作为 LFCLK 源使用</p> <p>1: WCO。监视晶振作为 LFCLK 源使用</p> <p>2-3: 保留。请勿使用</p>

8.3.5 SYSCLK 预分频器配置

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 时钟频率需要大于或等于器件中由 HFCLK 派生的所有其他时钟频率。SYSCLK 预分频器可对 HFCLK 进行 2 的 0 ~ 7 次方分频。通过使用寄存器 CLK_SELECT 的位 SYSCLK_DIV 设置预分频器分频值，如表 8-12 所示。预分频器最初配置的分频值是 1。

表 8-12. SYSCLK 预分频器的分频值位 SYSCLK_DIV

名称	说明
SYSCLK_DIV[3:0]	<p>SYSCLK 预分频器的分频值</p> <p>0: SYSCLK = HFCLK</p> <p>1: SYSCLK = HFCLK/2</p> <p>2: SYSCLK = HFCLK/4</p> <p>3: SYSCLK = HFCLK/8</p> <p>4: SYSCLK = HFCLK/16</p> <p>5: SYSCLK = HFCLK/32</p> <p>6: SYSCLK = HFCLK/64</p> <p>7: SYSCLK = HFCLK/128</p>

8.3.6 外设时钟分频器的配置

PSoC 4 共有 21 个时钟分频器，具体包括 16 个 16 位时钟分频器、4 个 16.5 位的小数时钟分频器，以及一个 24.5 位的小数时钟分频器。小数时钟分频器可为时钟分频器提供 0.31/32 的小数分频。小数分频器的输出频率的公式为： $F_{out} = F_{in} / (INT16_DIV + (FRAC5_DIV/32))$ 。例如，通过一个整数分频值为 3（INT16_DIV=3，FRAC5_DIV=0）的 16.5 位分频器可以使用将 48 MHz HFCLK 时钟生成一个 16 MHz 时钟信号。通过一个整数分频值为 4（INT16_DIV=4，FRAC5_DIV=0）的 16.5 位分频器可以使用 48 MHz HFCLK 时钟生成一个 12 MHz 时钟信号。通过一个整数分频值为 3（INT16_DIV=3）和小数分频值为 16（FRAC5_DIV=16）的 16.5 位分频器，可以使用 48 MHz HFCLK 时钟信号生成一个 13.7 MHz 的时钟信号。并非所有 13.7 MHz 时钟都有相同的周期。一半为 3 个 HFCLK 周期，另一半则为 2 个 HFCLK 周期。

如果需要使用一个高精度时钟（如供给 UART/SPI 串行接口），则推荐使用小数分频器。因为时钟周期带有 1 个 HFCLK 周期的抖动，所以需要有一个低抖动时钟时，不要使用小数分频器。

分别使用 PERI_DIV_16_CTLx 和 PERI_DIV_16_5_CTLx 寄存器来配置每个 16 位整数时钟分频器的分频值和 4 个 16.5 位小数时钟分频器的分频值。表 8-13 和表 8-14 显示的是这些寄存器的配置情况。

通过使用 PERI_DIV_24_5_CTLx 寄存器来配置 24.5 位小数分频器。表 8-15 显示了这些寄存器的配置情况。

表 8-13. 非小数外设时钟分频器配置寄存器 PERI_DIV_16_CTLx

位	名称	说明
0	ENABLE_x	分频器使能。硬件会通过 ENABLE 指令将该字段设置为‘1’。硬件会通过 DISABLE 指令将该字段设置为‘0’。
23:8	INT16_DIV_x	分频值为（1+INT16_DIV）的整数分频。整数分频范围为 [2, 65,536]。

表 8-14. 小数外设时钟分频器配置寄存器 PERI_DIV_16_5_CTLx

位	名称	说明
0	ENABLE_x	分频器使能。硬件会通过 ENABLE 指令将该字段设置为 ‘1’。硬件会通过 DISABLE 指令将该字段设置为 ‘0’。
7:3	FRAC5_DIV_x	分频值为 (FRAC5_DIV/32) 的小数分频。小数分频的范围为 [0, 31/32]。请注意，因为某些时钟周期比其他时钟周期多 1 个 “clk_hf” 周期，所以小数分频会产生时钟抖动。
23:8	INT16_DIV_x	分频值为 (1+INT16_DIV) 的整数分频。整数分频的范围为 [1, 65,536]。

表 8-15. 小数外设时钟分频器配置寄存器 PERI_DIV_24_5_CTLx

位字段	名称	说明
0	EN	分频器使能。硬件会通过执行 ENABLE 指令将该字段设置为 ‘1’，另外通过执行 DISABLE 指令可将该字段设置为 ‘0’。
7:3	FRAC5_DIV	分频值为 (FRAC5_DIV/32) 的小数分频。小数分频的范围为 [0, 31/32]。请注意，因为某些时钟周期比其他时钟周期多 1 个 “clk_hf” 周期，所以小数分频会产生时钟抖动。
31:8	INT24_DIV	分频值为 (1+INT24_DIV) 的整数分频。整数分频的范围为 [1, 16,777,216]。

使用 PERI_DIV_CMD 寄存器可以使能所有分频器。该寄存器可作为所有 16 位整数分频器和四个小数分频器的指令寄存器。PERI_DIV_CMD 寄存器的结构如下所示。

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Description	Enable	Disable															PA_SEL_TYPE			PA_SEL_DIV					SEL_TYPE							

SEL_TYPE 字段指定了所配置的分频器类型。对于 16 位整数分频器，该字段为 ‘1’；对于 16.5 位小数分频器，该字段为 ‘2’；对于 24.5 位小数分频器，该字段为 ‘3’。

SEL_DIV 字段指定了被配置的指定分频器的数量 ... 当 SEL_TYPE = 63 和 SEL_TYPE = 3 时，没有指定任何分频器。

(PA_SEL_TYPE, PA_SEL_DIV) 字段对可使分频器与其他分频器之间相位对齐。PA_SEL_DIV 指定了被相位对齐的分频器。任意已使能的分频器都可作为一个参考频率。PA_SEL_TYPE 指定了被相位对齐的分频器类型。当 PA_SEL_DIV = 63、PA_SEL_TYPE = 3 时，HFCLK 可作为参考时钟。

例如，一个 48 MHz 的 HFCLK 需要一个 12 MHz 的分频时钟 A 和一个 8 MHz 的分频时钟 B。时钟 A 使用一个 16 位整数分频器 0，通过将该分频器与 HF_CLK ((PA_SEL_TYPE, PA_SEL_DIV) 为 (3, 63)) 和 DIV_16_CTL0.INT16_DIV 为 3 对齐可以创建该时钟。时钟 B 使用一个整数分频器 1，通过将该分频器与时钟 A ((PA_SEL_TYPE, PA_SEL_DIV) 为 (1, 0)) 和 DIV_16_CTL1.INT16_DIV 为 5 对齐可以创建该时钟。这样可保证时钟 B 与时钟 A 相位对齐，因为两个时钟周期的最小公倍数为 12 HFCLK。因此每经过 12 个 HFCLK 周期，时钟 A 和时钟 B 都会对齐一次。请注意，时钟 B 与时钟 A 之间是相位对齐，但它仍将 HFCLK 作为其分频器分频值的参考时钟使用。

PSoC 中的每个外设模块都具有一个与它相关联的单独外设时钟 (PER#_CLK)。每个外设时钟都有一个复用输入，可从将任意一个现存的时钟分频器提取出输入时钟。

表 8-16 显示 复用器 输出与相应外设模块的映射情况 (如图 8-1 所示)。通过使用相应的 PERI_PCLK_CTLx 寄存器，可以将任意一个外设时钟分频器映射到一个特定的外设内，如表 8-16 所示。

表 8-16. 外设时钟复用器输出映射

外设时钟编号	外设
0	IMO (扩频)
1	CLOCK_PUMP
2	SCB0
3	SCB1
4	SCB2
5	SCB3
6	CSD0_CLK0
7	CSD0_CLK1
8	CSD1_CLK0

表 8-16. 外设时钟复用器输出映射（续）

外设时钟编号	外设
9	CSD 1_CLK1
10	SAR
11	TCPWM0
12	TCPWM1
13	TCPWM2
14	TCPWM3
15	TCPWM4
16	TCPWM5
17	TCPWM6
18	TCPWM7
19	UDB0
20	UDB1
21	UDB2
22	UDB3
23	LCD
24	PRGIO

表 8-17. 可编程时钟控制寄存器 — PERI_PCLK_CTLx

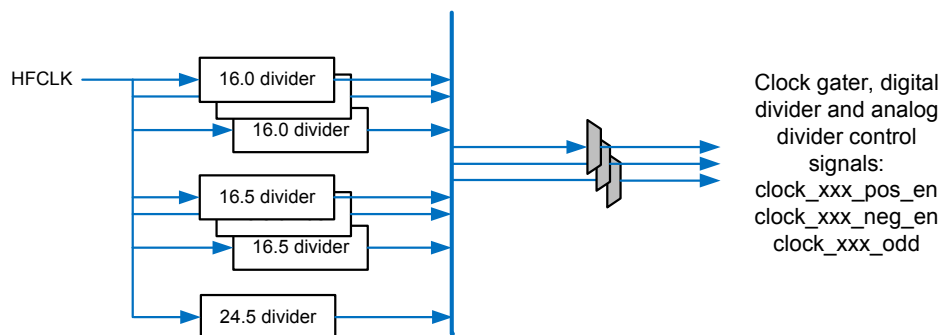
位	名称	说明
5:0	SEL_DIV	确定一个特定分频器（此分频器的类型由 SEL_TYPE 指定）。如果 SEL_DIV 为 “4”、SEL_TYPE 为 “1”，那么第五个（将 “0” 为第一个）16 位时钟分频器将被路由到复用器输出端，以供外设 clock_x 使用。同样，如果 SEL_DIV 为 “0”、SEL_TYPE 为 “2”，那么第一个 16.5 位时钟分频器将被路由到复用器输出端。
7:6	SEL_TYPE	0: 请勿使用 1: 16.0（整数）时钟分频器 2: 16.5（小数）时钟分频器 3: 24.5（小数）时钟分频器

8.3.7 外设时钟配置

外设时钟分频器、时钟门控器、数字时钟分频器以及模拟时钟分频器生成最终外设使用的时钟。

外设时钟分频器生成以下 3 种控制信号：clock_XXX_pos_en、clock_XXX_neg_en 及 clock_XXX_odd。时钟门控器、数字时钟分频器以及模拟时钟分频器都使用这些信号和 HFCLK，以便生成所需的时钟信号。图 8-4 提供外设时钟分频器的概述。

图 8-4. 外设时钟分频器



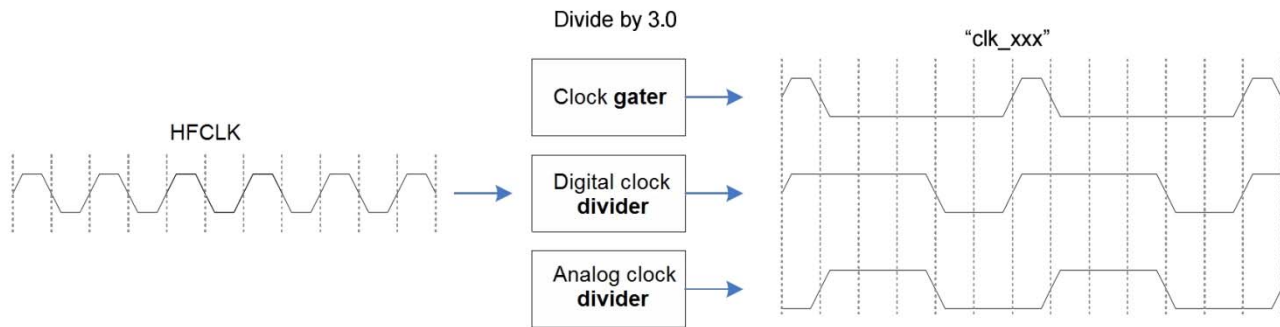
外设时钟分频器包括两层：

- 时钟分频器会根据一个整数分频器和一个可选小数时钟分频器值生成时钟信号。
- 时钟复用器从一个特定时钟分频器中选择信号，然后将三种时钟信号提供给特定的最终外设。

8.3.7.1 时钟生成

图 8-5 提供关于通过使用时钟门控器、数字时钟分频器及模拟时钟分频器生成时钟的概述。

图 8-5. 时钟的生成（通过使用时钟门控器、数字时钟分频器及模拟时钟分频器）



- 时钟门控器：该组件用于门控输入时钟源。“Clock_XXX_pos_en”信号用于使能时钟门控。该组件会生成单一的高脉冲，该脉冲的时间等于 HFCLK 高脉冲的时间。该逻辑通常使用于只有上升沿触发器的数字逻辑。
- 数字时钟分频器：该组件通过使用“clock_XXX_pos_en”及“clock_XXX_neg_en”生成时钟源。偶整数分频器生成的时钟均有 50% 占空比。奇整数分频器生成的时钟不具有 50% 的占空比；低电平时间将比高电平时间多 1 个周期。该组件使用于具有上升沿和下降沿触发器的数字逻辑（比如某个 UDB 阵列）。
- 模拟时钟分频器：该组件通过使用“clock_XXX_pos_en”、“clock_XXX_neg_en”及“clock_XXX_odd”信号生成时钟源。整数分频器（偶数及奇数分频器）生成的时钟均有 50% 占空比。

8.4 低功耗模式操作

高频率时钟（包括 IMO、EXTCLK、ECO、HFCLK、SYSCLK 及 PLL）和外设时钟仅在活动模式和睡眠模式下运行。ILO、WCO、及 LFCLK 可在所有功耗模式（休眠模式及停止模式除外）下运行。

8.5 寄存器列表

表 8-18. 时钟系统寄存器列表

寄存器名称	说明
CLK_IMO_TRIM1	IMO 调整寄存器 — 该寄存器包含 IMO 调整，允许对其频率进行细调。
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。
PWR_BG_TRIM4	带隙调整寄存器 — 这些寄存器控制带隙参考的调整，允许进行器件中相关参考电压的操作。
PWR_BG_TRIM5	
CLK_IMO_SPREAD	IMO 扩频控制寄存器 — 该寄存器控制 IMO 的扩频功能。
CLK_ILO_CONFIG	ILO 配置寄存器 — 该寄存器控制 ILO 的配置。
CLK_IMO_CONFIG	IMO 配置寄存器 — 该寄存器控制着 IMO 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。
WCO_CONFIG	WCO 使能。该寄存器用于禁用和使能外部监视晶振。
CLK_IMO_SPREAD	这个寄存器控制着 IMO 的扩频配置。
ECO_CONFIG	该寄存器用于配置 ECO。
PLL0_CONFIG	该寄存器用于配置 PLL0。
PLL1_CONFIG	该寄存器用于配置 PLL1。

9. 电源与电源监测



PSoC[®] 4 可运行在 1.71 V 到 5.5 V 的外部供应电压范围内。支持的工作电压范围如下：

- 内部电压调节器的输入电压范围为 1.80 V 到 5.50 V
- 直接供电电压范围为 1.71 V 到 1.89 V¹

请注意，根据器件封装中的可用性，PSoC 4 器件支持多种电源轨 — V_{DDA} 、 V_{DDD} 和 V_{DDIO} 。这些电源轨相互独立，并且可以单独将它们连接到所需的独立电源。即使在直接供电电压模式下，只需要将 V_{DDD} 限制在 1.71 V 到 1.89 V 的范围内，而其他电源轨（如 V_{DDA}/V_{DDIO} ）可以在整个 1.71 V 到 5.50 V 的范围内。更多有关各种电源轨和它们的使用情况的信息，请参考器件数据手册。

PSoC 4 器件具有多种内部电压调节器，以支持不同的功耗模式。它们包括：活动状态数字电压调节器、低噪声电压调节器、深度睡眠状态电压调节器以及休眠状态电压调节器。

1. 当系统电源的电压范围为 1.80 V 到 1.89 V 时，可以同时使用直接供电电压或内部电压调节器，具体情况取决于用户的系统功能。请注意，使用直接供电选项时，供电电压不能超过 1.89 V，否则会损坏器件。使用内部电压调节器时，供电电压也不能低于 1.80 V，否则电压调节器将被关闭。

9.1 框图

图 9-1. 电源系统框图

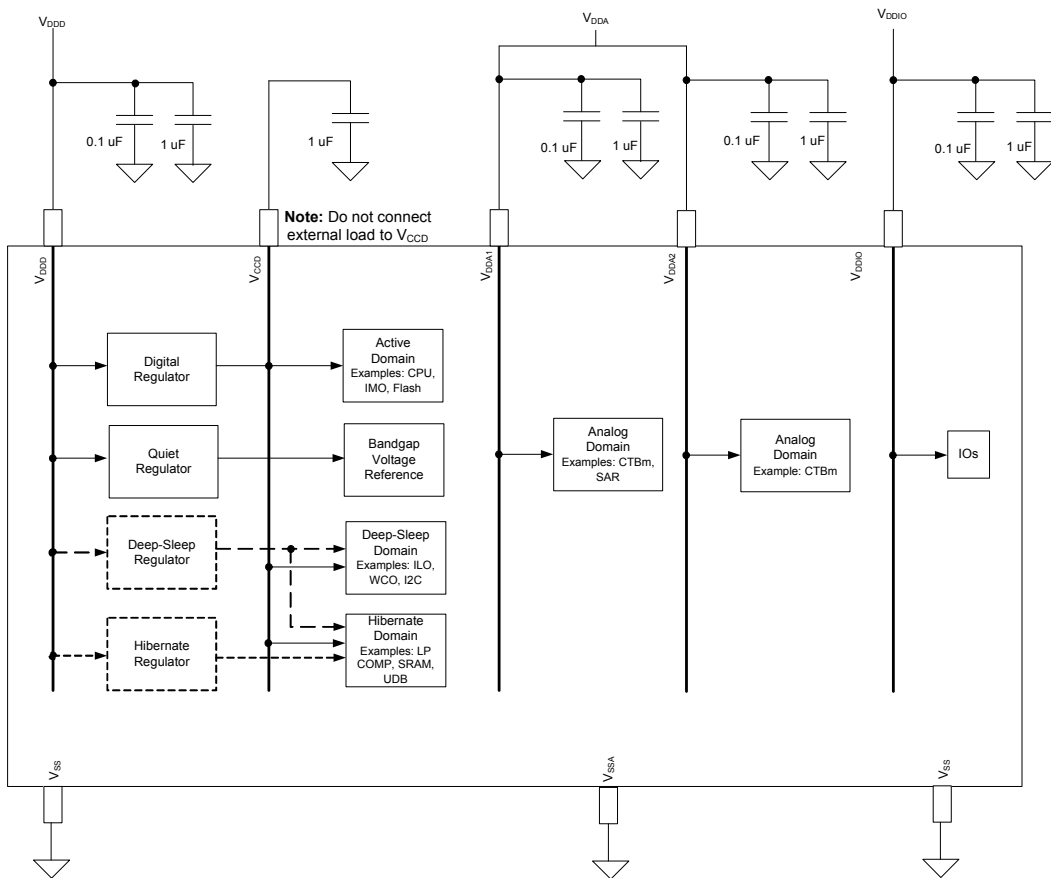


图 9-1 显示了 PSoC 4 的电源系统框图以及所有供电引脚。系统中的数字电路使用一个活动状态电压调节器。由于没有模拟电压调节器，因此模拟电路直接使用 V_{DDA} 输入所提供的电压来运行。深度睡眠模式和休眠（供电电压降低和数据保持）模式各有独自的电压调节器。带隙参考电压模块有一个独立的低噪声电压调节器。所有功能和电路都使用 1.71 到 5.5 V 的工作电压范围。PSoC 4 系列提供两种不同的电源操作模式：未调节外部供电和调节外部供电。有关电源连接的详细信息，请参考器件数据手册中的“电源”部分。

9.2 工作原理

图 9-1 中的电压调节器为器件的不同区域供电。所有内核电压调节器的输入都来自 V_{DD} 引脚的电源。数字 I/O 由 V_{DDIO} 供电。模拟电路直接使用了 V_{DDA} 输入所提供的电压运行。

9.2.1 电压调节器汇总

在活动或睡眠功耗模式下，活动状态数字电压调节器和低噪声电压调节器都被使能。在深度睡眠和休眠模式下，它们均被关闭（请参见表 11-1 和图 9-1）。深度睡眠和休眠状态电压调节器旨在满足器件在各低功耗模式下的功耗要求。

表 9-1. 电压调节器在各种功耗模式下的状态

模式	活动电压调节器	低噪声电压调节器	深度睡眠电压调节器	休眠电压调节器
停止	关闭	关闭	关闭	关闭
休眠	关闭	关闭	关闭	启用
深度睡眠	关闭	关闭	启用	启用
睡眠	启用	启用	启用	启用
活动	启用	启用	启用	启用

9.2.1.1 活动数字电压调节器

使用电压范围为 1.8 V 到 5.5 V 的外部电源时，在活动模式和睡眠模式下，活动状态数字电压调节器为主数字逻辑模块供电。该电压调节器的输出连接到 V_{CCD} 引脚，并且需要使用一个外部去耦电容（1 μ F 的 X5R 电容）。

通过置位 PWR_CONTROL 寄存器中的 EXT_VCCD 位，可以禁用活动状态数字电压调节器。这样可以降低直接供电模式中的功耗。活动状态数字电压调节器仅在活动和睡眠功耗模式下可用。

9.2.1.2 低噪声电压调节器

在活动和睡眠模式中，该电压调节器为模拟电路（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。该电压调节器具有高电源抑制比。低噪声电压调节器仅在活动和睡眠模式下可用。

9.2.1.3 深度睡眠状态电压调节器

该电压调节器为处于深度睡眠模式下保持上电状态的电路供电，如 ILO、WCO 和 SCB。除了休眠模式外，可以在所有功耗模式下使用深度睡眠状态电压调节器。在活动和睡眠功耗模式下，该电压调节器的主输出连接着活动状态数字电压调节器的输出（ V_{CCD} ）。该电压调节器也有单独的副本输出，它为低速时钟源提供了稳定的电压。在活动和睡眠模式下，该输出要断开与 V_{CCD} 的连接。

9.2.1.4 休眠状态电压调节器

该电压调节器将供电给在休眠睡眠模式下保持上电的电路，如睡眠控制器、低功耗比较器以及 SRAM。休眠状态电压调节器在所有功耗模式中均可用。在活动和睡眠模式中，该调节器的输出连接到数字电压调节器的输出。在深度睡眠模式中，该调节器的输出连接到深度睡眠电压调节器的输出端。

9.3 电压监控

电压监控系统包括上电复位（POR）、欠压检测（BOD）和低压检测（LVD）。

9.3.1 上电复位（POR）

在初次电源上升期间，上电复位电路提供了一个复位脉冲。POR 电路监控了 V_{CCD} 电压。通常，在触发点上，POR 电路对 V_{CCD} 的监控不是很准确。

在芯片的初次上电期间中使用 POR 电路，然后将其禁用。

9.3.1.1 欠压检测（BOD）

通过对器件进行复位，BOD 电路可使处于工作 / 保持状态的逻辑免发生无安全的供电状态。BOD 电路将监控 V_{CCD} 电压。如果电压偏移低于确保安全操作的最低 V_{CCD} 电压，那么，BOD 电路将生成一个复位（有关详细信息，请参考器件数据

手册）。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。

为了让固件能够区分欠压事件和正常的电源周期，请使用 BOD 生成一个复位后也不会被清除的特别寄存器（PWR_BOD_KEY）。然而，如果器件发生 POR 或 XRES，该寄存器将被清除。BOD 在所有功耗模式下均可用（停止模式除外）。

9.3.1.2 低压检测（LVD）

LVD 电路监控外部电源电压，并准确地检测能量源的消耗。LVD 检测器将生成一个中断，以使系统采取预防措施。

LVD 仅在活动和睡眠功耗模式下可用。如果在深度睡眠模式下需要使用 LVD，那么要配置芯片，以便使用 WDT 作为唤醒源来定期将其从深度睡眠模式唤醒。另外 LVD 监控最好在活动模式下实现。LVD 电路在安全工作电压范围内将以各个可编程的电平生成中断。通过 PWR_VMON_CONFIG 寄存器中的 LVD_SEL 字段，可以将 LVD 的触发点配置为介于 1.75 V 至 4.5 V 之间的范围。

使能 LVD 电路时，在初始建立时间内，可能发生错误中断。置位 PWR_VMON_CONFIG 寄存器中的 LVD_EN 位后，通过等待 1 μ s 时间，固件可以屏蔽错误中断。使能 LVD 功能的推荐固件程序如下：

1. 要确保 PWR_INTR_MASK 寄存器中的 LVD 位等于 0，以防传输错误中断信号。
2. 设置 PWR_VMON_CFG 寄存器中 LVD_SEL 字段的所需触发点。
3. 通过设置 PWR_VMON_CFG 寄存器中的 LVD_EN 位来使能 LVD。这时，可能导致错误 LVD 事件。
4. 等待至少 1 μ s 以让电路稳定。
5. 将 ‘1’ 写入到 PWR_INTR 寄存器中的 LVD 位，以清除错误事件。如果发生 LVD 条件，该位将不被清除。
6. 通过 PWR_INTR_MASK 中的 LVD 位，取消屏蔽中断。

9.4 寄存器列表

表 9-2. 电源与监控寄存器列表

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以及电压调节器的工作。
PWR_INTR	电源系统中断寄存器 — 该寄存器用于表示电源系统中断状态。
PWR_INTR_MASK	电源系统中断屏蔽寄存器 — 该寄存器控制哪个中断会被传输到 CPU 的中断控制器内。
PWR_VMON_CONFIG	电源系统电压监控调整与配置 — 该寄存器包含了电压监控系统的调整与配置位。

10. 芯片运行模式



PSoC® 4 可以在四种不同模式下运行固件。在这些模式下，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，调试模式专门用来对设计进行调试。

PSoC 4 的工作模式包括：

- 启动模式
- 用户模式
- 特权模式
- 调试模式

10.1 启动模式

启动模式是一种工作模式，在这种模式下通过器件 SROM 中的硬编码指令对器件进行配置。复位结束后，如果没有接收到调试序列，芯片将进入这种模式。启动模式是特权模式。在这种模式下，中断被禁止，从而免引导固件进行器件设置时被中断。在启动模式期间，器件从非易失性（NV）锁存加载硬件调整设置，从而确保在上电过程中能正常运行。启动结束时，器件将进入用户模式，并开始执行闪存的代码。闪存内的代码可能包括从 PSoC Creator IDE 自动生成的指令，该指令将进一步配置器件。

10.2 用户模式

用户模式是一种工作模式，在该模式下，可以执行闪存中存储的普通用户固件。在用户模式下，不能执行 SROM 中存储的代码。用户模式下的固件执行操作包括：PSoC Creator IDE 自动生成的固件和用户编写的固件。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，启动过程会将控制权交给该模式。

10.3 特权模式

特权模式是一种工作模式，在该模式下，可以执行器件 ROM 中所存储的特殊子程序。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。

通过执行一个系统调用，CPU 可以转换到特权模式。更多有关如何执行一个系统调用的信息，请参考第 300 页上的[执行系统调用](#)。退出该模式后，器件将返回用户模式。

10.4 调试模式

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，该模式用于调试固件。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。调试模式只适用于处于开放模式（四种保护模式之一）的器件。更多有关调试接口的信息，请参考第 291 页上的[编程与调试接口章节](#)。

更多有关保护模式的信息，请参考第 109 页上的[器件安全性章节](#)。

11. 功耗模式



PSoC[®] 4 支持五种功耗模式，从而最小化某个具体应用的平均功耗。按功耗从大到小的顺序，功耗模式依次为：

- 活动模式
- 睡眠模式
- 深度睡眠模式
- 休眠模式
- 停止模式

活动、睡眠及深度睡眠模式是 ARM 所定义的标准功耗模式，这些模式由 ARM CPU 和指令集架构（ISA）支持。休眠和停止模式是 PSoC 4 所支持的附加低功耗模式。与深度睡眠模式相同，可以通过固件进入这些模式。但被唤醒时，CPU 和所有外设都被复位。

可通过下列方法控制不同功耗模式下的功耗：

- 使能 / 禁用外设
- 开启 / 关闭内部电压调节器
- 开启 / 关闭时钟源
- 开启 / 关闭 PSoC 4 部分组件

图 11-1 描述了各种功耗模式以及它们之间的可能转换。

图 11-1. 功耗模式转换状态框图

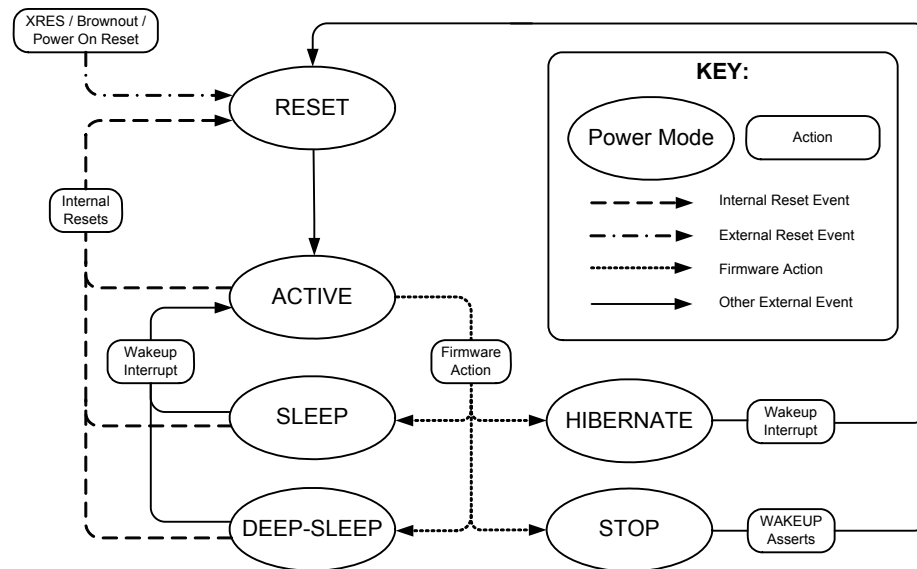


表 11-1 描述了 PSoC 4 所提供的各种功耗模式

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	可用电压调节器
活动模式	主要工作模式；所有外设都可用（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	所有时钟（可编程）	中断	所有电压调节器均可用。如果使用外部稳压调节器，则可以禁用活动状态数字电压调节器。
睡眠模式	CPU 进入睡眠模式，SRAM 被保持；所有外设均可用（可编程）。	通过寄存器手动写入	任何中断	所有时钟（可编程）	中断	所有电压调节器均可用。如果使用外部稳压调节器，则可以禁用活动状态数字电压调节器。
深度睡眠模式	所有内部电源来自深度睡眠电压调节器。IMO 和高速外设均关闭。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、低功耗比较器、SCB、看门狗定时器	ILO（32 kHz）、WCO（32 kHz）	中断	深度睡眠状态下的电压调节器和休眠状态下的电压调节器
休眠模式	仅保持 SRAM 和 UDB，所有内部供电电源（休眠供电电源除外）均断开。通过引脚中断或低功耗比较器可以实现唤醒操作。	通过寄存器手动写入	GPIO 中断，低功耗比较器	无	复位（保持中断状态）	休眠状态电压调节器
停止模式	所有内部供电电源均断开。仅保持 GPIO 状态。只能通过 XRES 或 WAKEUP 引脚实现唤醒操作。	通过寄存器手动写入	WAKEUP 引脚	无	复位	无

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意功耗模式转换为活动模式。

11.1 活动模式

活动模式是 PSoC 器件的主要功耗模式。在该模式下，器件中所有子系统 / 外设均可用，CPU 正在运行且所有外设都被供电。通过配置固件可以禁用未使用的特定外设，从而降低功耗。

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，且 CPU 时钟被禁用。为了降低功耗，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能将器件从睡眠模式唤醒。

11.3 深度睡眠模式

在深度睡眠模式下，CPU、SRAM、UDB 以及高速逻辑均保持活动状态。高频率时钟（包括 HFCLK 和 SYSCLK）被禁用。可选用内部低频（32 kHz）振荡器和监视晶体振荡器（WCO），使低频外设继续正常工作。无需时钟或从其外部接口接收时钟（例如，I2C 从设备）的数字外设将持续执行操作。来自低速、异步或低功耗模拟外设的中断可以使器件从深度睡眠模式唤醒。CTBm 也可以在该模式下运行（其功耗和带宽在这种情况下被降低）。有关功耗和 CTBm 带宽的更多信息，请参见器件数据手册。

表 11-2 中列出了可用的唤醒源。

11.4 休眠模式

这是 PSoc 4 中保持 SRAM 数据的最低功耗模式。通过关闭所有时钟和断开 CPU 及所有外设的电源（能够通过外部事件来唤醒系统的一些（异步）外设除外），可以达到最低功耗。请注意，在该模式下，CPU 和所有外设的状态不被保留。

该模式使用了容量有限的休眠状态电压调节器，从而能够实现极低功耗。因此，在休眠模式下，输入引脚上所有信号的最大频率将受到限制。所有 I/O 引脚的总切换率（所有输入和输出引脚信号的总频率）不能超过 10 kHz。

对于信号高速切换的系统，可以使用深度睡眠模式，而总功耗几乎不变。

只能通过引脚中断或低功耗比较器从休眠模式中唤醒。从休眠模式中唤醒后将引起一个复位事件，而不是一个中断唤醒事件。从休眠模式唤醒时，CPU 和大多数外设都处于复位状态，而固件将从复位向量处开始执行。除非在进入休眠模式前固件已经明确冻结了这些引脚，否则该复位会使各个 I/O 引脚处于三态。要想了解中断原因，请使用 PWR_STOP 寄存器中的 TOKEN 位，如第 100 页上的[进入和退出低功耗模式](#)中所述。

外部复位（XRES）将触发整个系统的重启事件。在这种情况下，

器件重启后，无法读取唤醒原因，且 I/O 引脚不会保留其“冻结”状态。

11.5 停止模式

在停止模式下，CPU、所有内部电压调节器以及所有外设均被关闭。从停止模式中唤醒是一个系统复位事件，并且只能通过 XRES 或 WAKEUP 引脚实现。进行复位后，I/O 引脚将处于三态（除非在进入停止模式之前，固件已经明确冻结了这些引脚）。要想了解中断原因，请使用 PWR_STOP 寄存器中的 TOKEN 位，如第 100 页上的[进入和退出低功耗模式](#)中所述。

外部复位（XRES）将触发整个系统的重启事件。在这种情况下，器件重启后，无法读取唤醒原因，且 I/O 引脚不会保留其“冻结”状态。

11.6 功耗模式总结

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

表 11-2. 唤醒源

功耗模式	唤醒源	唤醒事件
睡眠	任意中断源	中断
	所有复位源	复位
深度睡眠	GPIO 中断	中断
	低功耗比较器	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
	CTBm	中断
	BLESS	中断
休眠模式	GPIO 中断	复位
	低功耗比较器	复位
	XRES（外部复位引脚） ^a ，欠压	复位
停止模式	WAKEUP 引脚	复位
	XRES（外部复位引脚） ^a ，欠压	复位

a. XRES 将触发整个系统的重启事件。所有状态（包括冻结 GPIO）均被丢失。在这种情况下，器件重启后，无法读取唤醒原因。

11.7 进入和退出低功耗模式

Cortex-M0 (CM0) 的 “Wait For Interrupt” (等待中断, WFI) 指令能够触发切换到睡眠、深度睡眠 和休眠 模式的事件。Cortex-M0 可以将进入低功耗模式的事件拖延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)。

切换到睡眠、深度睡眠和休眠模式的事件是由 CM0 系统控制寄存器 (CM0_SCR) 中的 SLEEPDEEP 标志和系统资源电源子系统 (PWR_CONTROL) 中的 HIBERNATE 标志控制的。

- WFI 指令被执行, 并且 SLEEPDEEP = 0 和 HIBERNATE = x 时, 将进入睡眠模式。
- WFI 指令被执行, SLEEPDEEP = 1 和 HIBERNATE = 0 时, 器件将进入深度睡眠模式。
- WFI 指令被执行, 且 SLEEPDEEP = 1 和 HIBERNATE = 1 时, 器件将进入休眠模式。

PWR_CONTROL 寄存器中的 LPM READY 位显示的是深度睡眠和休眠状态电压调节器的状态。如果在电压调节器准备好前, 固件尝试进入深度睡眠或休眠模式, 那么 PSoC 4 会先进入睡眠模式, 然后等待电压调节器准备好后, 器件才会进入深度睡眠或休眠模式。通过使用硬件自动完成该操作。

在睡眠和深度睡眠模式下, 允许选择外设 (请参见表 11-2), 且固件可以使能或禁用与其相应的中断。已使能的中断可以使系统从低功耗模式唤醒, 然后再进入活动模式。另外, 所有 RESET 都会使系统返回到活动模式。更多有关信息, 请参见第 51 页上的中断章节和第 105 页上的复位系统章节。

通过 PWR_STOP 寄存器, 可以将 GPIO 状态冻结在低功耗模式中。建议将该方法应用于休眠和停止模式, 因为从这些模式中唤醒会使系统复位。

通过使用系统资源电源系统中的 PWR_STOP 寄存器, 可以直接进入停止模式。这时, 系统中所有低电压逻辑电源都被断开。只会保持 I/O 状态和 PWR_STOP 寄存器中的内容, 并且 XRES 引脚或固定 WAKEUP 引脚的切换会导致唤醒 (复位)。

PWR_STOP 寄存器中的字段包括:

- **TOKEN** — 该字段包含一个能够在 STOP/WAKEUP 序列中保持的 8 位令牌。该令牌可供固件使用, 这样可以区分 WAKEUP 事件和通用的 RESET 事件。请注意, 使用 XRES 将器件从停止模式唤醒会复位该寄存器。
- **UNLOCK** — 必须向该字段写入 ‘0x3A’, 以解锁停止模式。如果对该字段写入其他值, 硬件将忽略 STOP 位。
- **POLARITY** — 该位用于设置 WAKEUP 引脚输入的极性。WAKEUP 引脚输入同 POLARITY 位值匹配时, 器件将被唤醒。
- **FREEZE** — 置位该位后将冻结系统中所有 GPIO 的配置、模式和状态。
- **STOP** — 必须置位该位才能进入停止模式。

进入停止模式的推荐程序如下:

1. 写 TOKEN = < 任何特定于应用的值 >
2. 写 UNLOCK = 0x3A
3. 写 POLARITY = < 特定于应用的极性 >
4. 写 FREEZE = 1
5. 写 STOP = 1

建议在第三个写步骤后面添加两个 NOP 周期。切换 XRES 或 WAKEUP 引脚时, 将退出停止模式。这两个事件将清除 PWR_STOP 寄存器中的 STOP 位, 并触发 POR。唤醒事件不会清除 PWR_STOP 寄存器中的其他位, 但 XRES 事件将清除所有位。

从停止或休眠模式中唤醒的推荐固件程序如下:

1. 为特定应用的分支可选读取 TOKEN。
2. 可以按照所需设置对各 I/O 驱动模式和输出数据寄存器进行写操作。数字输出端口的典型程序为: 将引脚设置为输出, 读取它的冻结值, 然后将该值设置给输出数据寄存器。
3. 解冻 I/O。

11.8 寄存器列表

表 11-3. 功耗模式寄存器列表

寄存器名称	说明
CM0_SCR	系统控制 — 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。
PWR_STOP	停止功耗模式 — 控制进入 / 退出停止功耗模式。

12. 看门狗定时器



当固件执行发生异常时，可以使用看门狗定时器（WDT）进行自动复位器件。WDT 由 ILO 或 WCO 生成的 LFCLK 提供时钟。如果定时器被使能，则固件必须定期刷新它，以避免发生复位。否则，定时器将停止并生成器件复位。在低功耗模式下，WDT 还可作为中断源或唤醒源使用。

12.1 特性

WDT 具有以下特性：

- 可以在可配置时间间隔后生成系统复位
- 在活动、睡眠和深度睡眠模式下，能够定期中断（或唤醒）器件
- 提供两个 16 位和一个 32 位的独立计数器，可以将它们级联起来延长间隔

12.2 框图

图 12-1. 看门狗定时器框图

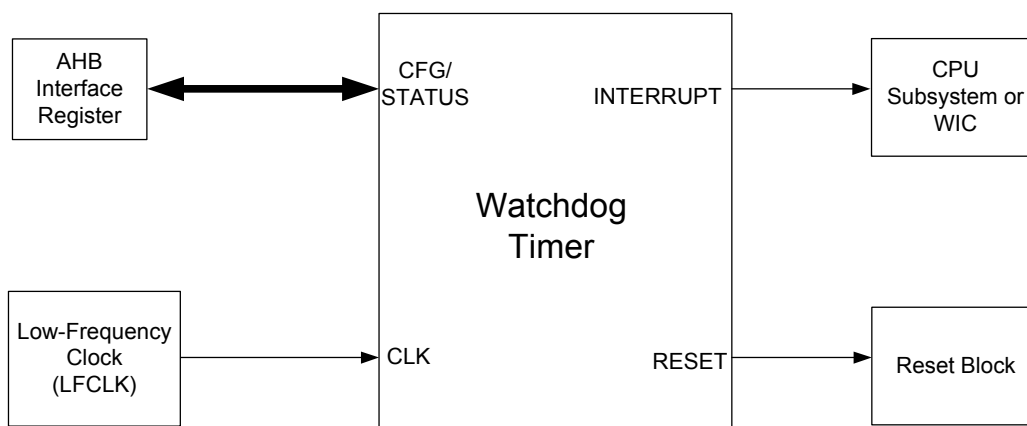
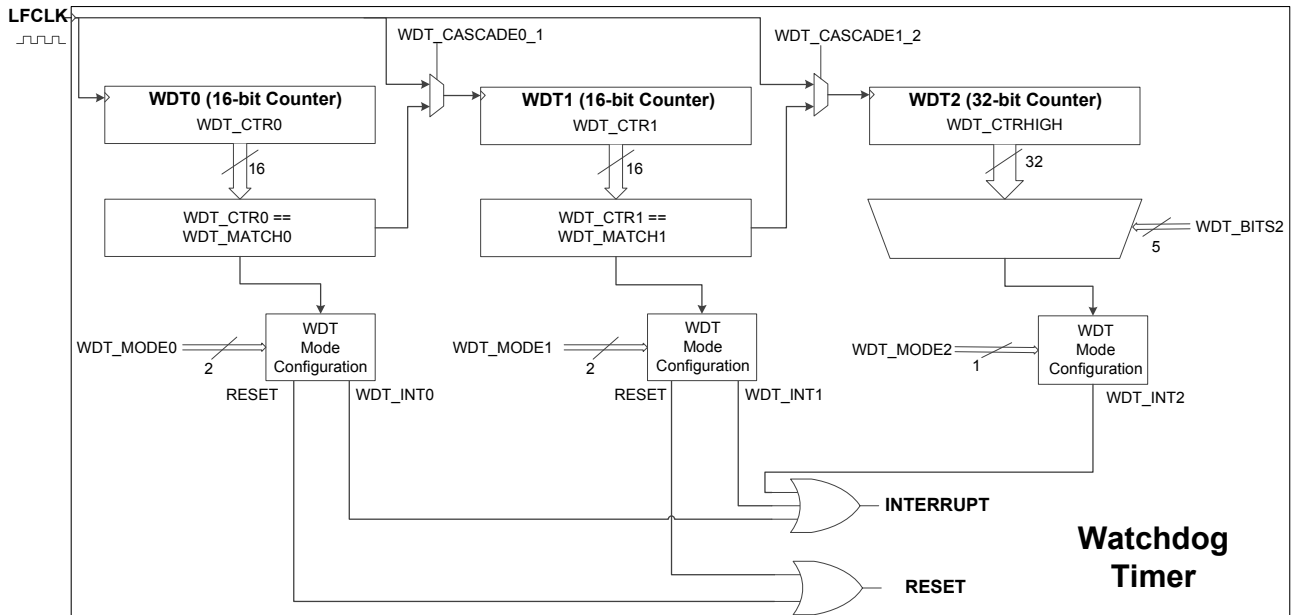


图 12-2. 看门狗定时器内部框图



12.3 工作原理

除非固件对 WDT 进行定期处理，否则经过可编程间隔后，它会激活器件的中断或硬件复位。WDT 拥有两个 16 位计数器（WDT0 和 WDT1）和一个 32 位计数器（WDT2）。可以将这些计数器配置为独立或级联计数器。可以配置 WDT0 和 WDT1，从而能在发生匹配事件时（即计数器的值等于匹配值）生成中断。也可以对 WDT0 和 WDT1 计数器进行配置，以便在发生匹配事件时或在 3 个连续事件未得到处理（不清除匹配事件中断）后生成复位。

根据 WDT_CONFIG 寄存器中 WDT_BITS2[4:0] 位的值，可以配置 WDT2 生成中断。当任意一个匹配值等于 WDT1 或 WDT0 值时，WDT2 不能生成系统复位或中断。WDT2 只能在计数器内 32 位中任意一位的上升沿上生成中断。WDT_CONFIG 寄存器中的 WDT_BITS2[4:0] 位控制着用于生成中断的位。有关更多信息，请参见 PSoC 4200L 寄存器技术参考手册中 WDT_CONFIG 寄存器的内容。

WDT_MODEx 位可用于配置看门狗计数器，如上所述。

通过图 12-2 中所示的级联配置，可以延长复位或中断间隔。请注意，级联两个 16 位计数器并不能提供一个 32 位计数器；但能够得到一个 16 位定期计数器和一个 16 位预分频器。例如，级联 WDT0 和 WDT1 时，WDT0 将作为 WDT1 的预分频器，并且预分频值将由 WDT_MATCH 寄存器中的 WDT_MATCH0[15:0] 位定义。WDT1 的周期大小将由 WDT_MATCH 寄存器中的 WDT_MATCH1[31:16] 位定义。相似的逻辑也适用于 WDT1 和 WDT2 的级联。

当使用 WDT 来防止系统崩溃时，必须使用与 WDT 中断没有直接关联的代码来清除 WDT 中断位，从而实现复位 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 来防止系统崩溃的最安全方法是：

- 配置看门狗复位周期，以便在该周期甚至固件的最大延迟路径中，固件能够至少复位一次看门狗。
- 通过在固件代码的主体中定期清除中断位，从而复位看门狗。如果将 WDT 配置为在匹配事件发生时生成复位，那么通过清除 WDTx 计数器来复位看门狗。通过设置 WDT_CONFIG 寄存器中的 WDT_RESETx 位，可以清除 WDTx 计数器。更多有关信息，请参见 PSoC 4200L 寄存器技术参考手册中 WDT_CONFIG 寄存器的内容。
- 如果将 WDT 作为复位源来防止系统崩溃，那么不建议在 WDT 中断服务子程序（ISR）中复位看门狗。因此，请不要使用相同的看门狗计数器来生成系统复位和中断。例如，如果使用 WDT0 来生成系统复位，从而防止系统崩溃，那么需要使用 WDT1 或 WDT2 来定期生成中断。

请按照以下各步骤将 WDT 作为一个周期性中断发生器使用：

1. 为 WDT0（或 WDT1）设置 WDT_CONFIG 寄存器中的 WDT_CLEAR0（或 WDT_CLEAR1）位，以便在发生匹配事件时对相应的看门狗计数器进行复位。
2. 将所需的匹配值写入到 WDT0/WDT1 的 WDT_MATCH 寄存器内，并将 WDT_BITS2 的值写入到 WDT2 的 WDT_CONFIG 寄存器内。
3. 通过清除 WDT_CONTROL 寄存器中的 WDT_INTx 位，可以清除所有待处理中断。

- 通过配置 WDT_CONFIG 中的 WDT_MODEx 位来使能 WDT 中断。对于 WDT0 或 WDT1，将 WDT_CONFIG 中的 WDT_MODE0 或 WDT_MODE1 位配置为 ‘1’（匹配时生成中断）或 ‘3’（匹配时生成中断，并在尚未处理第三个匹配时生成复位）。对于 WDT2，要设置 WDT_CONFIG 寄存器中的 WDT_MODE2 位。
- 使能 CM0_ISER 寄存器中的全局 WDT 中断（有关详细信息，请参考第 51 页上的中断章节）。
- 在 ISR 中，清除 WDT 中断。

更多有关中断的信息，请参见第 51 页上的中断章节。

对 WDT_MATCH 的更改需要经过三个 LFCLK 周期后才会生效。更改 WDT_MATCH 后，请经过一个 LFCLK 周期后才进入深度睡眠模式，以确保 WDT 更新了设置内容。

12.3.1 使能和禁用 WDT

通过设置 WDT_CONTROL 寄存器中的 WDT_ENABLEx 位，可以使能 WDT 计数器；如果想禁用它，则需要清除该位。使能或禁用 WDT 需要经过三个 LFCLK 周期后才能生效。因此，在这段时间内，不能多次更改 WDT_ENABLEx 位。

表 12-1. WDT0 和 WDT1 模式

位字段名称	说明
WDT_MODE0[1:0] 或 WDT_MODE1[1:0]	发生匹配情况，即（WDT_CTR0=WDT_MATCH0）或（WDT_CTR1=WDT_MATCH1）时，看门狗计数器的操作如下： 00：无操作 01：激活 WDT_INT0 或 WDT_INT1 10：激活 WDT 复位 11：激活 WDT_INT0 或 WDT_INT1，并在尚未处理第三个中断后激活 WDT 复位

根据 WDT_BITS2[4:0] 寄存器位的状态，可以使用 WDT2 生成中断。

表 12-2. WDT2 模式

位字段名称	说明
WDT_MODE2	0：无中断请求的自由运行计数器 1：自动运行计数器，将在由 WDT_CONFIG 寄存器中 WDT_BITS2 位所指定的位上升沿上生成中断请求

注意：如果对看门狗进行配置，以便在每个 LFCLK 周期内都会生成中断，那么请确保清除看门狗中断（设置 WDT_CONTROL 寄存器内的 WDT_INTx 位）后读取 WDT_CONTROL 寄存器的值。否则，会丢失下一个中断，也会在 LFCLK/2 时间内生成中断。

使能 WDT 后，建议不要对 WDT 配置寄存器（WDT_CONFIG）和控制（WDT_CONTROL）寄存器进行写操作。通过设置 CLK_SELECT 寄存器中的 WDT_LOCK[15:14] 位，可以防止 WDT 寄存器意外损坏。如果在运行 WDT 时，应用需要更新匹配值（WDT_MATCH），则必须清除 WDT_LOCK 位。需要进行两次写操作才能清除这两个 WDT_LOCK 位。写入 ‘1’ 会清除位 0，写入 ‘2’ 会清除位 1，写入 ‘3’ 会设置这两位，而写入 ‘0’ 则会使其无效。更多有关信息，请参见 PSoC 4200L 寄存器技术参考手册中 WDT_SELECT 寄存器的内容。

12.3.2 WDT 操作模式

通过使用 WDT0 和 WDT1 来生成复位，可以阻止系统进入无响应状态，或者通过使用这些定时器生成中断，这样可使系统从睡眠或深度睡眠模式唤醒。可以对 WDT_CONFIG 寄存器中的 WDT_MODEx[1:0] 位字段进行配置，以便在 WDT_CTRx 寄存器中存储的计数值等于 WDT_MATCH 寄存器中存储的匹配值时，能够选择所需要的操作。更多有关信息，请参见 PSoC 4200L 寄存器技术参考手册中介绍 WDT_CTRHIGH、WDT_CTRHIGH 和 WDT_MATCH 寄存器的内容。

12.3.3 WDT 中断和低功耗模式

在活动模式下，将看门狗计数器的中断请求发送给 CPU；在睡眠和深度睡眠模式下，则将该中断请求发送给唤醒中断控制器（WIC）。其工作原理如下：

- **活动模式：**在活动模式下，WDT 可以将中断发送给 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 的中断请求将被直接发送到 WIC，以唤醒 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。

更多有关 BLE-SS 状态功耗模式的信息，请参考第 97 页上的 [功耗模式章节](#)。

12.3.4 WDT 复位模式

RES_CAUSE 寄存器中的 RESET_WDT 位表示 WDT 生成了复位。该位将保持置位状态，直到对其进行清除操作或发生上电复位（POR）、欠压复位（BOD）或外部复位（XRES）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 105 页上的 [复位系统章节](#)。

12.4 寄存器列表

表 12-3. WDT 寄存器

寄存器名称	说明
WDT_CTRLOW	看门狗计数器 0 和 1
WDT_CTRHIGH	看门狗计数器 2
WDT_MATCH	看门狗计数器 0 和 1 的匹配值
WDT_CONFIG	包含 WDT 的配置位
WDT_CONTROL	控制 WDT 计数器的操作

13. 复位系统



PSoC[®] 4 支持多种复位类型，从而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 还包括用于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，如果供电电压低于标准范围，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，可以使用固件进行复位
- 外部复位（XRES）— 为 PSoC 4 使用外部电子信号来复位器件
- 保护错误复位（PROT_FAULT）— 如果用户尝试进行特权操作，将产生复位
- 休眠唤醒复位 — 使器件退出休眠低功耗模式
- 停止唤醒复位 — 使器件退出停止低功耗模式

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

上电复位用于上电时复位系统。POR 会将器件锁定于复位状态，直到供电电压（即 V_{DD} ）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，可以在一定程度上推断出原因。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

欠压复位将监控芯片数字电源电压 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，将生成复位。BOD 在所有功耗模式下均可用（停止模式除外）。

BOD 事件将不设置复位源状态位，但是在某些情况下，可以检测到它们。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。详细信息，请查阅第 106 页上的识别复位源。

13.1.3 看门狗复位

如果看门狗定时器未在用户指定的时间内被清除掉，那么看门狗复位（WRES）将通过生成复位发现代码的错误。通过置位 WDT_CONTROL 寄存器内的 WDT_ENABLEx 位来使能该功能。

发生看门狗复位时，RES_CAUSE 寄存器内的 RESET_WDT 状态位将被置位。该位保持为置位状态，直到被清除或发生 POR、XRES 或不可检测的 BOD 复位为止，例如：在一个器件的供电周期中。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 101 页上的看门狗定时器章节。

13.1.4 软件复位

软件复位 (SRES) 是一个允许软件驱动复位的机制。将 ‘1’ 写入 SYSRESETREQ 位, 这时 Cortex-M0 应用中断与复位控制寄存器 (CM0_AIRCR) 将强制器件进行复位。为了使能写操作, 需要将数值 A05F 写入到 CM0_AIRCR 寄存器的前两个字节内。因此, 复位操作需要写入 A05F0004。

发生软件复位时, RES_CAUSE 寄存器内的 RESET_SOFT 状态位将被置位。该位保持为置位状态, 直到被清除或发生 POR、XRES 或不可检测的 BOD 复位为止, 例如: 在一个器件的供电周期内。所有其它复位操作都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 引脚为 **低电平有效** — 该引脚上的高电平电压不起任何作用, 而低电平电压则会导致复位。在器件中, 引脚电平被拉高。在大多数器件中, XRES 均作为专用引脚使用。更多有关引脚分布的详细信息, 请参见 器件数据手册中的 “引脚分布” 一节。

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 的逻辑阈值和其它电气特性。

XRES 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缺失情况, 可以在一定程度上推断出原因。如果未检测到其它复位事件, 那么, 复位就是由 POR、不可检测的 BOD 或 XRES 导致的。

13.1.6 保护错误复位

保护错误复位 (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。更多有关特权代码的信息, 请参见 [第 95 页上的特权模式](#)。

发生保护错误时, RES_CAUSE 寄存器内的 RESET_PROT_FAULT 位被置位。该位保持为置位状态, 直到被清除或发生 POR、XRES 或不可检测的 BOD 复位为止, 例如: 在一个器件的供电周期中。所有其它复位操作都不会对该位产生任何影响。

13.1.7 休眠模式唤醒复位

休眠模式唤醒复位将检测休眠唤醒源, 并执行器件复位, 以返回活动模式。休眠模式唤醒复位是由中断导致的。在休眠低功耗模式下, 引脚和比较器中断均可用。执行休眠唤醒复位后, SRAM 和 UDB 寄存器的内容均被保留, 但是此复位后的代码执行与其它复位源发生后的执行相同。

通过检查比较器和引脚的中断寄存器, 可以检测休眠复位。休眠状态唤醒复位后, 仍保留这些中断寄存器的状态。

更多信息, 请查阅 [第 99 页上的休眠模式](#)。

13.1.8 停止模式唤醒复位

停止模式唤醒复位将检测各个停止模式唤醒源, 并执行一次器件复位, 以返回到活动模式。停止模式唤醒复位是由 XRES 引脚或 WAKEUP 引脚导致的。执行停止模式唤醒复位后, 将不会保留存储器中的任何内容; 复位后的代码执行与其它复位源发生后的相同。

通过检查 PWR_STOP 寄存器中的 TOKEN 位字段 (位 0:7), 可以检测某些停止模式的唤醒源。进入停止模式时, 会向该字段内填充一个关键字。如果使用 WAKEUP 引脚唤醒器件, 该字段的内容将被保留。如果使用 XRES 引脚唤醒器件, 则无法检测唤醒源。更多信息, 请查阅 [第 99 页上的停止模式](#)。

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对应的特定状态位。RES_CAUSE 寄存器支持看门狗复位、软件复位以及保护错误复位等复位源检测。它并没记录发生 POR、BOD、XRES 或休眠和停止模式唤醒复位。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置位状态, 直到它被清除或丢失保留为止, 如 POR 复位、外部复位或低于逻辑保持电压的欠压复位。

通过检查被配置为将器件从休眠模式中唤醒的比较器和引脚中断寄存器, 可以检测休眠唤醒复位。通过检查 PWR_STOP 寄存器, 可以检测由 WAKEUP 引脚事件导致的停止唤醒复位, 如上面所述。无法检测到由 XRES 导致的停止唤醒复位。通过 [表 13-1](#) 所示的 RES_CAUSE 状态, 可以在一定程度上推断其他复位源。

表 13-1. 用于检测复位源的复位原因位

位	名称	说明
0	RESET_WDT	芯片发生了一次看门狗定时器复位。
3	RESET_PROT_FAULT	发生保护违法行为时就需要一次 RESET。
4	RESET_SOFT	Cortex-M0 通过它的 SYSRESETREQ 请求一次系统复位。

欠压事件可分为两个子类型：保持复位和无保持复位。如果 V_{CCD} 低于最小逻辑工作电压，但不小于最小逻辑保持电压，这时将发生 BOD 复位；寄存器中的内容会被保持。如果 V_{CCD} 低于最小工作电压和最小保持电压，将发生 BOD 复位，但不会保持寄存器中的内容。可以使用特别的寄存器（PWR_BOD_KEY）来检测寄存器所持有的内容。仅在通过固件进行写操作或发生无保持复位（如无保持 BOD、XRES

或 POR 事件）时，PWR_BOD_KEY 寄存器的值才被修改。通过固件初始化该寄存器，然后在启动代码的后续执行中进行检查该寄存器，从而确认是否发生了保持 BOD 复位。

如果这些方法都不能检测到复位源，那么复位源可能是下面无记录或无保持复位中的一种：没保持 BOD、POR、XRES 或停止唤醒复位。通过片上资源无法区分这些复位。

13.3 寄存器列表

表 13-2. 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。
CM0_AIRCR	Cortex-M0 应用中断和复位控制寄存器 — 除了初始化软件复位功能外，寄存器还提供了 Cortex-M0 的其他功能。
RES_CAUSE	复位原因寄存器 — 该寄存器捕捉最近发生的复位的原因。
PWR_STOP	该寄存器用于控制停止功耗模式的进入和退出情况。

14. 器件安全性



PSoC® 4 为用户提供了多种方式来防止未授权的访问和复制。禁止调试性能以及使能闪存保护提供了高级别的安全性。在 PSoC 4200L 器件中，通过执行通用数字模块（即 UDB）（不是在固件中的自定义功能），可以提高安全性。与进行目标代码的逆向工程相比，对在 UDB 中实现的硬件设计进行逆向工程更加困难。

默认情况下，调试电路处于使能状态，并且只能通过固件禁用它们。如果调试电路被禁用，则重新使能该电路的唯一方法是擦除整个器件，清除闪存保护，然后使用允许调试的新固件重新编程该器件。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。对于大多数应用，不建议永久性禁用接口，因为这样会使设计人员无法访问器件。更多有关信息以及关于闪存行和芯片保护的讨论，请参阅 CY8C4xxxL 的编程规范中介绍的内容。

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，因此已使能全器件安全性的 PSoC 4 器件将不能再退回进行故障分析。

14.1 特性

PSoC 4 器件安全系统具有以下特性：

- 支持用户可选的保护级别。
- 在最高安全级别模式下，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。在特权模式下，NMI 保持确认状态，以防止中断指令的意外返回而造成安全漏洞。

另外，PSoC 4 还为单独闪存行数据提供保护。

14.2 工作原理

14.2.1 器件安全性

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。每种模式均为 CPU 软件和调试提供了特定的功能。通过对 CPUSS_PROTECTION 寄存器进行写操作，可以更改该模式。

- **BOOT 模式：**在此模式下，器件将退出复位状态。器件一直处于该模式，直到将其保护状态从监控闪存复制到保护控制寄存器（CPUSS_PROTECTION）内为止。该操作完成前，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 始终在特权模式下运行。
- **OPEN 模式：**这是出厂默认模式。CPU 可以在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，并且支持调试器功能。在特权模式下，访问受限。
- **PROTECTED 模式：**用户可以将 OPEN 模式转换为 PROTECTED 模式。在此模式下，禁止对用户代码或存储器进行的所有调试访问。尽管仍可以对大部分寄存器进行访问，但不能调试寄存器，以重新对闪存进行编程。只有完成对闪存的擦除后才能将此模式转换回到 OPEN 模式。
- **KILL 模式：**用户可以将 OPEN 模式转换为 KILL 模式。这样可清除对用户代码或存储器的所有调试访问，但不能擦除闪存。尽管仍可以对大部分寄存器进行访问；但不能对寄存器进行调试来实现对闪存进行重新编程。器件无法退出 KILL 模式。在该模式下，器件不能退回进行故障分析。

14.2.2 闪存安全性

PSoC 4 器件包含了一个灵活的闪存保护系统，该系统控制着对闪存存储器进行的访问。该性能可以保护专有代码，另外它还可以防止对闪存内的 **Bootloader** 部分进行的意外写操作。

闪存存储器按行组织。您可以将两个保护级别的其中一个分配给每一行；请参考表 14-1。要想更改闪存保护级别，必须擦除整个闪存。

更多信息，请参考第 299 页上的非易失性存储器编程章节。

表 14-1. 闪存保护级别

保护设置	支持	不支持
无保护	外部读写操作， 内部读写操作	—
全面保护	外部读操作 ^a 内部读操作	外部写操作， 内部写操作

a. 为了防止对 PSoC 4 器件进行的外部读操作，您需要将器件保护设置更改为 PROTECTED（受保护）。

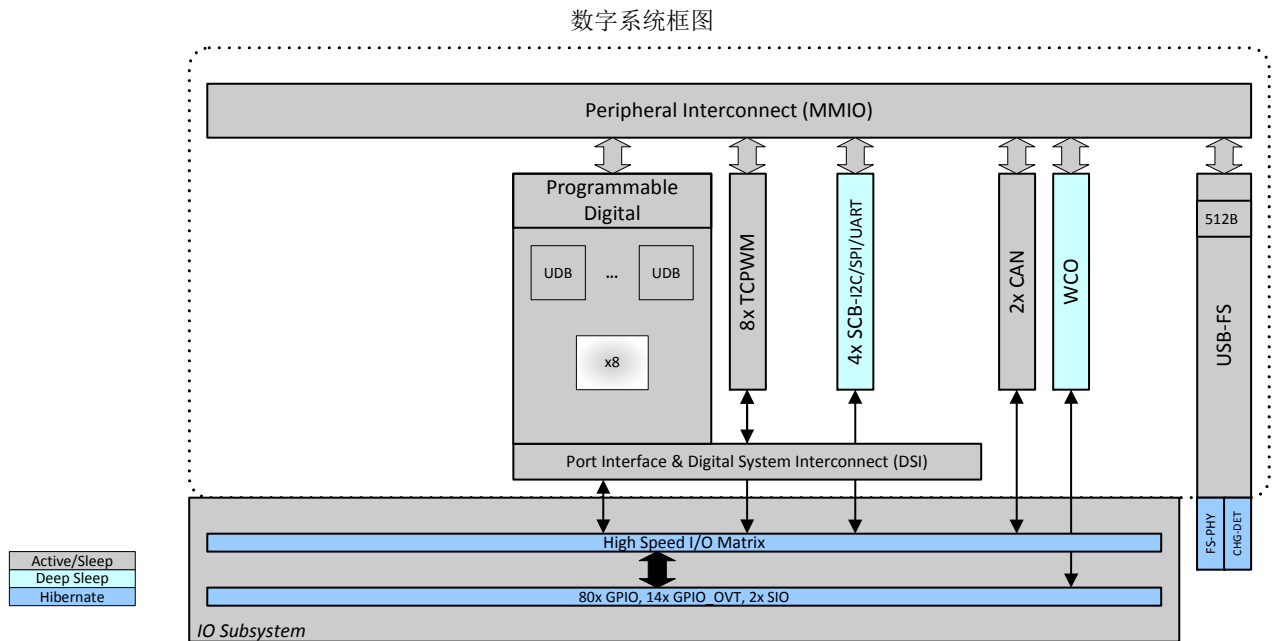
节 D: 数字系统



本章节包括下面小节：

- 第 113 页上的通用数字模块（**UDB**）章节
- 第 153 页上的控制器区域网络（**CAN**）章节
- 第 171 页上的通用串行总线（**USB**）章节
- 第 189 页上的定时器、计数器和 **PWM** 章节

系统架构



15. 通用数字模块（UDB）



本章节介绍了 PSoC[®] 4 通用数字模块（UDB）的详细设计信息。UDB 架构在配置精细程度和高效实现两者之间取得了平衡；UDB 包含了可编程逻辑器件（PLD）、结构逻辑（数据路径）和灵活布线方案。

15.1 特性

- PSoC 4 包含一个带有八个 UDB 的阵列
- 为了实现最佳灵活性，每个 UDB 包含以下各组件：
 - 一个基于 ALU 的 8 位数据路径（DP），包含许多寄存器、FIFO 和一个 8 字指令存储区
 - 两个 PLD，每个都具有 12 个输入、8 个乘积项和 4 个宏单元输出
 - 控制和状态模块
 - 时钟和复位模块
- 通过 UDB 阵列可灵活进行路由
- 各个 UDB 的部分可以共用或进行级联，以扩展使用更多功能。
- 可灵活实现多项数字功能，包括：定时器、计数器、PWM（带死区发生器）、UART、SPI 和 CRC 生成 / 检查功能
- 用于与 CPU 通信的接口寄存器

图 15-1 显示的是构成一个 UDB 的各个组件：两个 PLD 模块、一个数据路径以及控制、状态、时钟和复位模块。

图 15-1. 单个 UDB 的框图

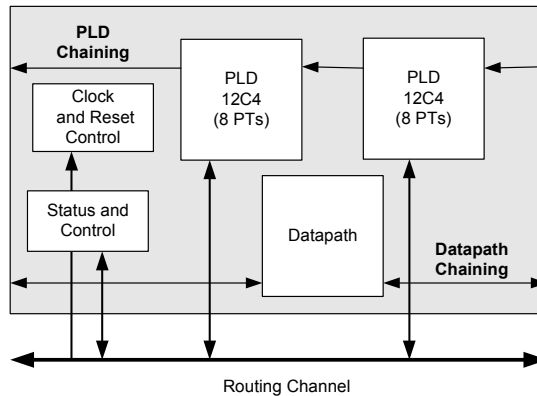
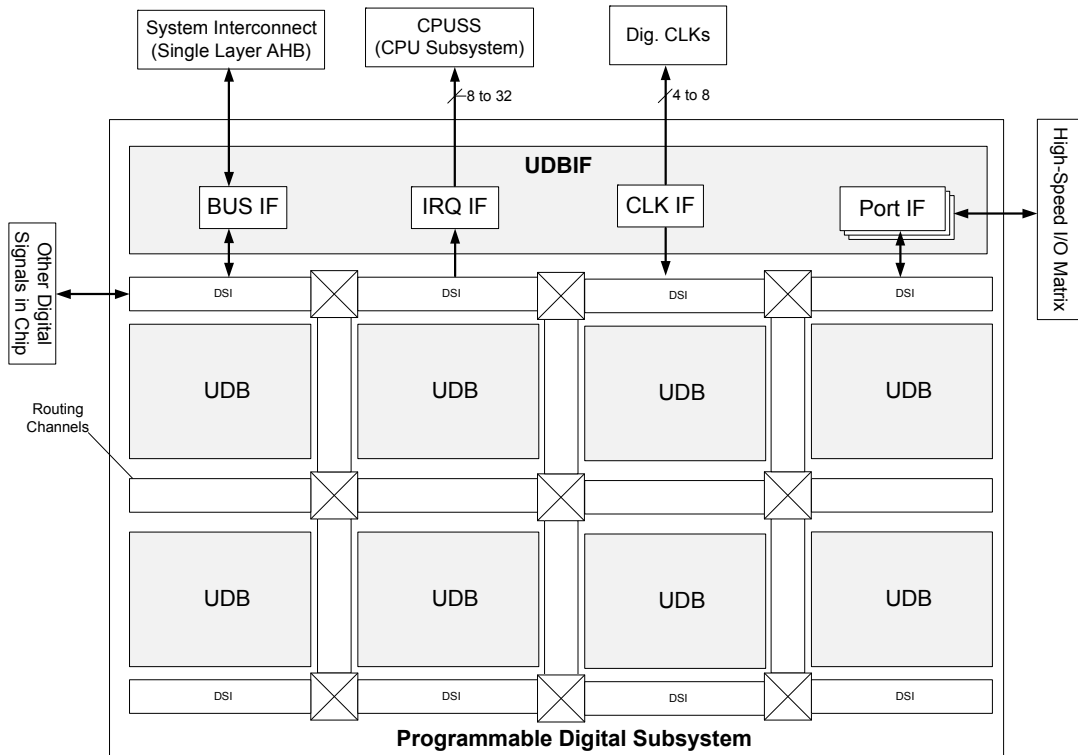


图 15-2 显示了如何将四个 UDB 连接到 PSoC 4 的其余部分。

图 15-2. PSoC 4 中的 UDB 阵列



15.2 工作原理

UDB 由以下各个主要组件组成：

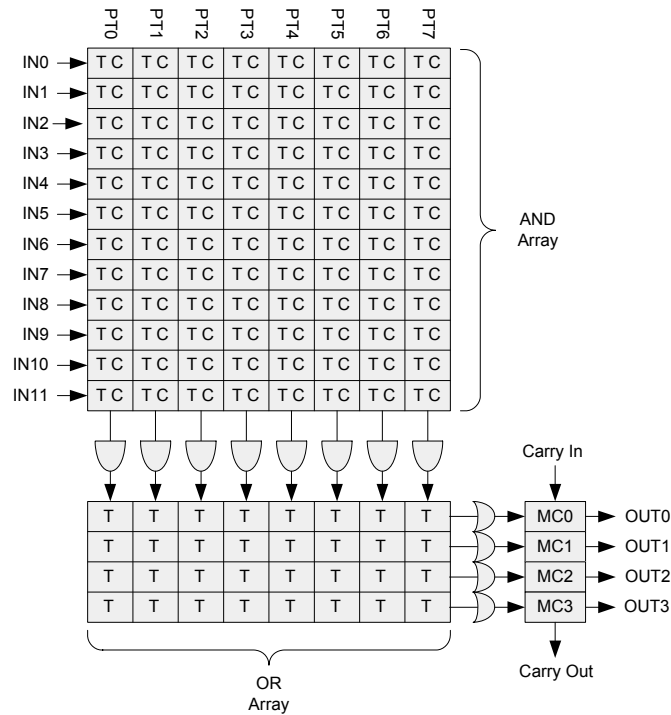
- **PLD (2)** — 这些模块从路由通道获取输入，并构成寄存或组合“乘积和”逻辑来实现状态机、数据路径操作的控制、条件输入和驱动输出。
- **数据路径** — 该模块包含一个动态可编程 ALU、四个寄存器、两个 FIFO、比较器和条件生成。
- **控制和状态模块** — 这些模块为 CPU 固件提供与 UDB 操作进行交互和同步的方式。
- **复位和时钟控制** — 这些模块为 UDB 中的其他模块提供了时钟选择和使能，以及复位选择。
- **级联信号** — PLD 和数据路径的级联信号允许链接相邻 UDB，以创建更高精度的功能。
- **路由通道** — 通过可编程开关阵列，将 UDB 连接到路由通道。该通道可以连接 UDB 中的所有模块，还可以连接阵列中的所有其他 UDB。
- **系统总线接口** — 每个 UDB 中的所有寄存器和 RAM 都被映射到系统地址空间内，并且 CPU 通过 8 位、16 位和 32 位访问可以对这些寄存器和 RAM 进行访问。

15.2.1 PLD

每个 UDB 具有两个“12C4” PLD。显示在图 15-3 内的 PLD 模块可用于实现状态机、执行输入或输出数据调整以及创建查找表 (LUT)。通过配置 PLD，还可以执行算术功能、定序数据路径和生成状态。通用 RTL 可以合成并映射到 PLD 模块。本节对 PLD 设计的概述进行了介绍。

该 PLD 具有 12 个输入，可用于驱动 **AND** 阵列中的 8 个乘积项 (PT)。在给定的乘积项中，可以选择输入的“真值” (T) 或“补码” (C)。PT 的输出就是 OR 阵列的输入。12C4 中的‘C’表示 OR 项在所有输入中均保持不变，并且每个 OR 输入可以通过编程方式访问任何 PT 或所有 PT。这种结构能够提供最大的灵活性，并确保所有输入和输出都是可交换的。

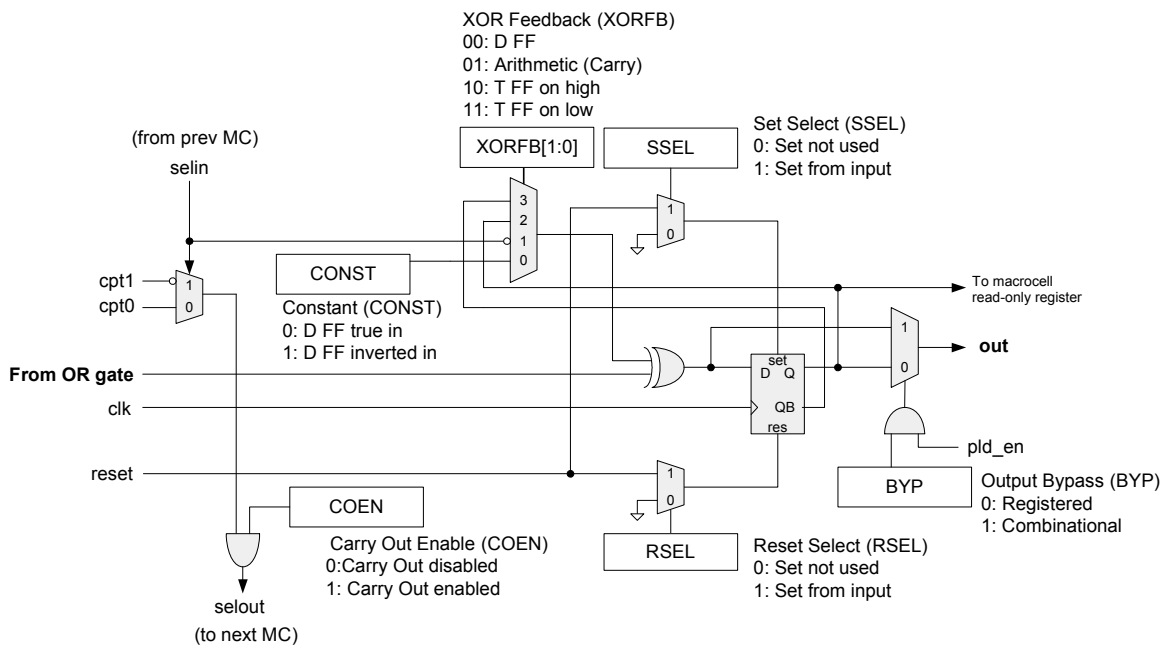
图 15-3. PLD 12C4 结构



15.2.1.1 PLD 宏单元

图 15-4 显示的是宏单元的架构。该输出会驱动路由阵列，并且能够被寄存或组合。寄存模式拥有包含“真”或反相输入的 D 触发器 (DFF) 和处理输入高或低的反转触发器 (TFF)。可以设置或复位输出寄存器，以用于初始化，或在运行过程中通过路由信号对该寄存器进行异步设置或复位。

图 15-4. PLD 宏单元架构



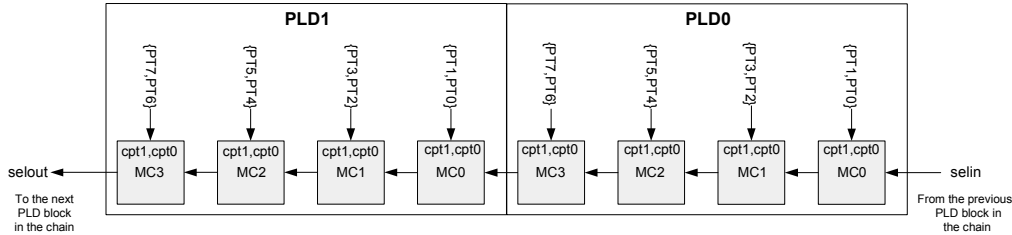
只读的 PLD 宏单元寄存器

CPU 可以将两个 PLD 中 8 个宏单元的输出作为 8 位只读寄存器进行访问。另外，它还可以将许多 UDB 中的宏单元作为 16 位或 32 位只读寄存器进行访问。请参见第 146 页上的 UDB 寻址。

15.2.1.2 PLD 进位链路

PLD 按照 UDB 地址顺序被级联在一起。如图 15-5 中所示，进位链路输入 “selin” 从链路中前一个 UDB 路由到两个 PLD 中各个宏单元，然后路由到下一个 UDB，成为进位链路输出 “selout”。为了支持算术功能的高效映射，特殊的乘积项被生成。这些乘积项和进位链路可以同时被使用在宏单元中。

图 15-5. PLD 进位链路和特殊乘积项输入



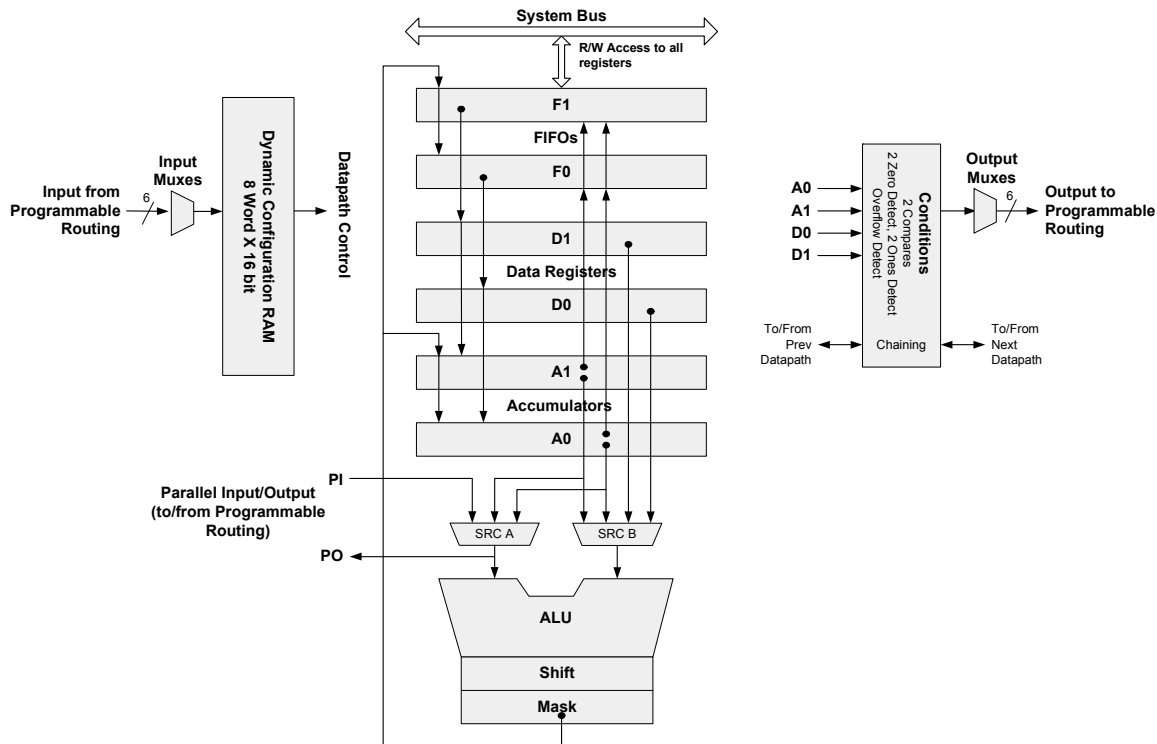
15.2.1.3 PLD 配置

通过访问一组 16 位或 32 位的寄存器，可以对 PLD 进行配置；请参考第 146 页上的 UDB 寻址中的内容。

15.2.2 数据路径

如图 15-6 所示，数据路径模块包含一个 8 位的单周期 ALU 和相关的比较及条件生成电路。数据路径可被级联到相邻 UDB 中的数据路径，从而得到更高精度的功能。数据路径包含一个小型动态配置 RAM，用于动态选择将在给定周期内执行的操作。对数据路径进行优化，以实现典型的嵌入式功能，如定时器、计数器、PWM、PRS、CRC、移位器和死区生成器。加法和减法功能允许支持数字 delta-sigma 操作。

图 15-6. 数据路径架构



15.2.2.1 概述

下面显示的是主要的数据路径特性:

动态配置

动态配置是指在序列发生器的控制下,按周期更改数据路径功能和内部连接的能力。该操作是使用配置 RAM 实现的,该 RAM 存储了 8 个独立的配置。该 RAM 的地址输入可以来自与路由结构相连的任意模块 (通常是 PLD 逻辑、I/O 引脚),也可以来自其他数据路径。

ALU

ALU 可以执行 8 个通用功能:递增、递减、加法、减法、AND、OR、XOR 和 PASS。功能选择操作是由所配置的 RAM 按周期控制的。可以在 ALU 的输出上使用独立移位 (左移、右移、半字节交换) 和掩码操作。

条件

每个数据路径都有两个带位掩码选项的比较器。通过配置这些比较器可以选择用于比较的多个数据路径寄存器输入。其他可检测的条件包括全零、全一和溢出检测。这些条件允许将主要的数据路径输出选择作为其他功能的输入路由到数字布线结构。

内置 CRC/PRS

数据路径包含对以下内容的内置支持:任意宽度和任意多项式的单周期循环冗余校验 (CRC) 计算和伪随机序列 (PRS) 生成。为得到超过 8 位的 CRC/PRS 宽度,可以在各数据路径之间链接信号。该特性可以通过动态方式控制,因此可以与其他功能交替进行。

可变 MSB

算术和移位功能的最高有效位可通过编程方式指定。可变 MSB 支持可变宽度 CRC/PRS 功能,而且通过与 ALU 输出掩码相结合,可实现任意宽度的定时器、计数器和移位模块。

输入 / 输出 FIFO

每个数据路径都包含两个 4 字节的 FIFO,这些 FIFO 可单独被配置为输入缓冲区 (CPU 写入到 FIFO,数据路径内部读取 FIFO) 或输出缓冲区 (数据路径内部写入到 FIFO,CPU 读取 FIFO)。这些 FIFO 能够生成已满或空白的状态信号,可以路由由这些信号,以便与序列发生器或中断进行交互。

链路

可配置数据路径,以便将条件和信号同相邻的数据路径连接起来。可以链接移位、进位、捕获和其他条件信号,以实现精度更高的算术、移位和 CRC/PRS 功能。

时间复用

在过采样或不需要最高时钟频率的应用中,数据路径中的单个 ALU 模块可以有效地被两组寄存器和条件生成器共用。ALU 和移位输出会被寄存起来,并可在后续周期中作为输入使用。使用示例支持在一个 (8 位) 数据路径中实现 16 位功能,并支持在 CRC 生成操作和数据移位操作之间进行切换。

数据路径输入

该数据路径具有三个输入类型:配置、控制以及串行和并行数据。通过配置输入可以选择动态配置 RAM 地址。控制输入从 FIFO 加载数据寄存器,累加器并将输出捕获到 FIFO 内。串行数据输入包括 shift in 和 carry in。并行数据输入端口能够在路由过程中引入多达 8 位的数据。

数据路径输出

该数据路径可以生成多达 16 个信号。其中,一些信号为条件信号 (例如,比较信号)、一些为状态信号 (例如,FIFO 状态),剩下的信号为数据信号 (例如,shift out 信号)。这些信号被复用成 6 个数据路径输出,然后被驱动到路由矩阵内。默认情况下,各输出是单同步 (即管道式) 的。这些输出还有一个组合输出选项。

数据路径的工作寄存器

每个数据路径模块有 6 个 8 位工作寄存器。CPU 可以对这些寄存器进行读 / 写操作:

表 15-1. 数据路径工作寄存器

类型	名称	说明
累加器	A0, A1	累加器可以作为 ALU 的操作数, 也可以作为结果。数据寄存器或 FIFO 的数据可被载入这些累加器内。累加器通常包含了某个功能 (如: 计数、CRC 或移位) 的当前值。这些寄存器都是非保留寄存器; 在睡眠模式下, 这些寄存器的值将被丢失, 在唤醒模式下则被复位为 0x00。
数据	D0, D1	数据寄存器通常包含某个功能的常数数据, 如: PWM 比较值、定时器或 CRC 多项式。这些寄存器在睡眠间隔中会保留它们的值。
FIFO	F0, F1	两个 4 字节 FIFO 为缓冲的数据提供一个数据源和一个结果。可以将这两个 FIFO 都配置为输入缓冲区、输出缓冲区, 或配置为一个输入缓冲区和一个输出缓冲区。状态信号显示这些寄存器的满 / 空状态。使用示例包括 SPI 或 UART 中的缓冲 TX 和 RX 数据, 以及缓冲 PWM 比较和缓冲定时器周期数据。这些寄存器都是非保留寄存器; 在睡眠模式下, 这些寄存器的值将被丢失, 在唤醒模式下则被复位为 0x00。

15.2.2.2 数据路径 FIFO

FIFO 模式和配置

每个 FIFO 具有多种操作模式和配置。

表 15-2. FIFO 模式和配置

模式	说明
输入 / 输出	在输入模式下, CPU 将数据写入到 FIFO 中, 数据路径内部读取并使用该数据。在输出模式下, 数据路径内部会将数据写入到 FIFO 内, 然后 CPU 将读取并使用该数据。
单字节缓冲区	FIFO 作为无状态的单字节缓冲区运行。写入 FIFO 内的数据可以立即被读取, 并且可以随时被覆盖。
电平 / 边沿	从数据路径内部加载 FIFO 可以通过电平或边沿触发实现。
正常 / 快速	从数据路径源加载 FIFO 的操作是在当前选定的数据路径时钟 (正常) 或 HFCLK (快速) 上进行采样的。这样能按系统的最高频率 (HFCLK) 进行捕获, 独立于数据路径时钟。
软件捕获	当使能该模式而且 FIFO 处于输出模式时, CPU 读取相应累加器 (F0 的 A0、F1 的 A1), 累加器值将被同步传输到 FIFO 内。然后可以立即从 FIFO 中读取该捕获值。如果链接被使能, 该操作会跟着链路去到 MS 模块以对多字节值的数据路径进行原子读操作。
异步	当数据路径时钟与 HFCLK 异步时, FIFO 状态信号可以直接被路由到数据路径中其余部分, 或与数据路径时钟做单次采样, 或与异步数据路径时钟做两次采样后输出。
独立时钟极性	每个 FIFO 都有一个控制位, 用于反转 FIFO 时钟信号的极性以将其跟随数据路径时钟信号。

图 15-7 显示的是输入 / 输出模式控制的各种可能的 FIFO 配置。TX/RX 模式具有两个 FIFO, 一个处于输入模式, 另一个处于输出模式。SPI 是该配置的主要示例。双捕获配置提供了 A0 和 A1 的独立捕获, 或 A0/A1 的两个单独控制捕获。最后, 双缓冲模式可以提供被缓冲的周期和比较, 或两个独立的周期 / 比较。

图 15-7. FIFO 配置

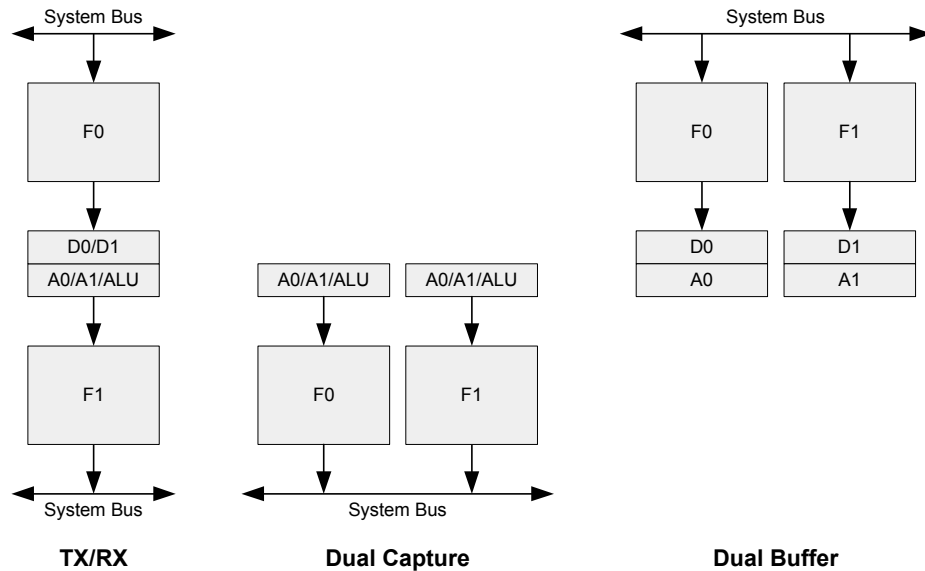
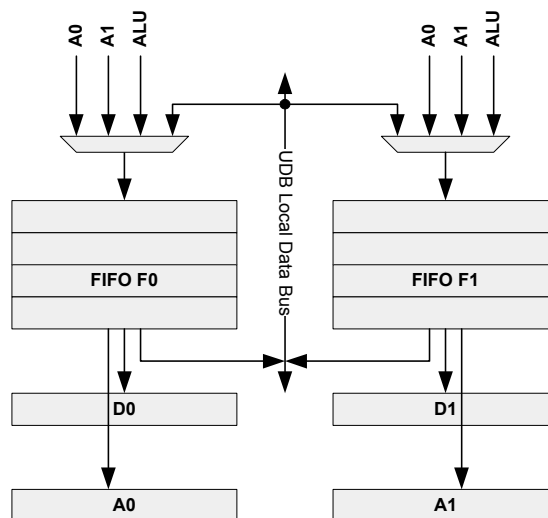


图 15-8 显示的是 FIFO 数据源和数据流的详细概况。

图 15-8. FIFO 数据源和数据流



FIFO 处于输入模式时，数据源为系统总线，数据流为 Dx 和 Ax 寄存器。而 FIFO 处于输出模式时，数据源为 Ax 寄存器和 ALU，数据流为系统总线。该复用器的选择是在 UDB 配置寄存器 CFG15 中静态设置的，用于 F0_INSEL[1:0] 或 F1_INSEL[1:0]，如表 15-3 中所示。

表 15-3. UDB CFG15 寄存器中的 FIFO 多路复用器设置

Fx_INSEL[1:0]	说明
00	输入模式 — 系统总线对 FIFO 进行写操作，FIFO 输出目标为 Ax 或 Dx。
01	输出 A0 模式 — FIFO 输入源为 A0，FIFO 输出目标为系统总线。
10	输出 A1 模式 — FIFO 输入源为 A1，FIFO 输出目标为系统总线。
11	输出 ALU 模式 — FIFO 输入源为 ALU 输出，FIFO 输出目标为系统总线。

FIFO 状态

每个 FIFO 会生成两个状态信号：“总线”和“模块”。这两个信号通过数据路径输出复用器被传送到 UDB 路由。可以使用“总线状态来激活某个中断请求，以对 FIFO 进行读/写操作。”“模块”状态主要用于为 UDB 内部提供 FIFO 状态。状态位的含义取决于配置方向（UDB CFG15 寄存器中 Fx_INSEL[1:0]）和 FIFO 电平位。FIFO 电平位（Fx_LVL）在工作寄存器空间中的辅助控制工作控制寄存器（ACTL）内设置。表 15-4 显示了各个选项。

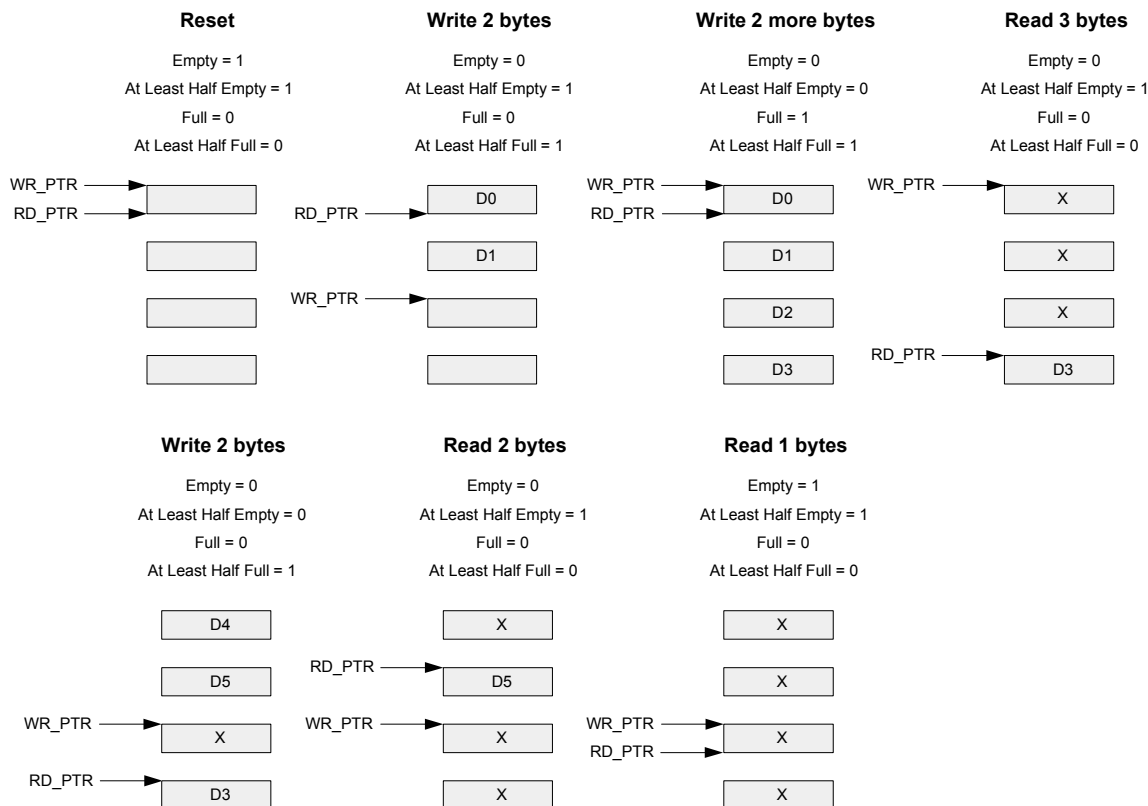
表 15-4. FIFO 状态选项

Fx_INSEL[1:0]	Fx_LVL	FIFO 状态	FIFO 状态信号	说明
输入	0	未滿	总线状态	只有 FIFO 中至少有一个字节的空间时，输入才被激活。
输入	1	至少一半为空	总线状态	FIFO 中至少存在两个字节的空间时，输入才被激活。
输入	NA	空白	模块状态	FIFO 中没有字节（FIFO 为空）时，输入才被激活。FIFO 非空时，数据路径内部可能使用这些字节。FIFO 为空时，数据路径会处于闲置状态或生成一个欠载条件。
输出	0	非空	总线状态	FIFO 中至少存在一个字节可读时，输出才被激活。
输出	1	至少一半为空	总线状态	FIFO 中至少存在两个字节用于读取时，输出才被激活。
输出	NA	已满	模块状态	FIFO 已满时，输出才被激活。FIFO 未滿时，数据路径内部会将字节写到 FIFO 内。FIFO 已满时，数据路径会处于闲置状态或生成一个过载条件信号。

FIFO 操作

图 15-9 显示的是典型的读写序列和相关的状态生成。虽然该图指出读和写操作在不同的时间内发生，但读写操作还可以同时发生。

图 15-9. 详细的 FIFO 操作数据流

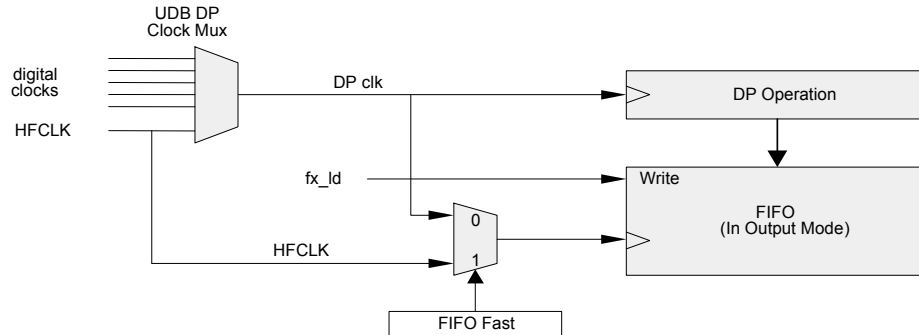


FIFO 快速模式 (FIFO FAST)

当将 FIFO 配置为输出时，FIFO 加载操作通常使用当前选择的数据路径时钟来采样写信号。如图 15-10 中所示，设置 FIFO 快速模式后，可以选择 HFCLK 用于该操作。与边沿敏感模式一起使用时，可以使累加器到 FIFO 的传输延迟从数据路径时钟分辨率下降为 HFCLK 分辨率（该延迟可以为更高的值）。这样可使 CPU 读取 FIFO 中最小延迟的捕获结果。

图 15-10 指出了快速加载操作独立于当前所选择的数据路径时钟；但使用 HFCLK，功耗会更大。请注意，fx_id 输入信号必须满足 HFCLK 时序要求，这可能需要对局部进行重新同步。

图 15-10. FIFO 快速配置数据流



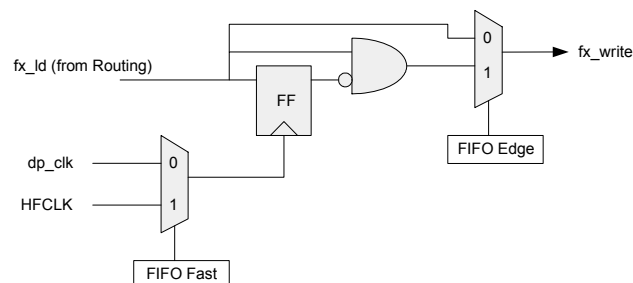
FIFO 电平 / 边沿写模式

在这两种模式下可以通过数据路径进行写 FIFO。在第一种模式下，各个累加器的数据被同步传输给 FIFO。该写操作的控制信号（fx_id）通常由状态机或与数据路径时钟同步的条件生成。可以在输入加载控制为‘1’的任何周期内对 FIFO 执行写操作。

在第二种模式下，fx_id 信号的上升沿出现时，通过 FIFO 可以捕获累加器的值。在该模式下，波形的占空比是任意的（然而，它的宽度至少必须为一个数据路径时钟周期）。例如，通过将外部引脚输入作为触发器使用来捕获累加器的值。该模式的限制是在检测到另一个上升沿前，输入控制必须至少在一个周期内恢复为‘0’。

图 15-11 显示了 fx_id 控制输入上的边沿检测选项。该选项的 1 位在 UDB 中为两个 FIFO 设置模式。请注意，边沿检测在已选定的 FIFO 时钟频率上被采样。

图 15-11. 内部 FIFO 写操作的边沿检测选项



FIFO 软件捕获模式

一个常见且重要的要求是：在普通操作期间允许 CPU 读取累加器中的内容。该操作是通过软件捕获实现的，并且通过设置 FIFO 捕获配置位（UDB CFG16 寄存器中的 FIFO_CAP 位）被触发。该位适用于 UDB 中的两个 FIFO，但仅在 FIFO 处于输出模式时，才能进行操作。使用软件捕获时，需要将 F0 设置为从 A0 加载，并将 F1 设置为从 A1 加载。

如图 15-12 中所示，读取累加器将触发该累加器对 FIFO 进行的写操作。该信号被链接，以便读取已给字节时同时会捕获所有链接 UDB 中的累加器。这样允许 CPU 可以同时读取 16 位或更多位。读取累加器后返回的数据将被忽略；FIFO 中的捕获值可以立即被读取。

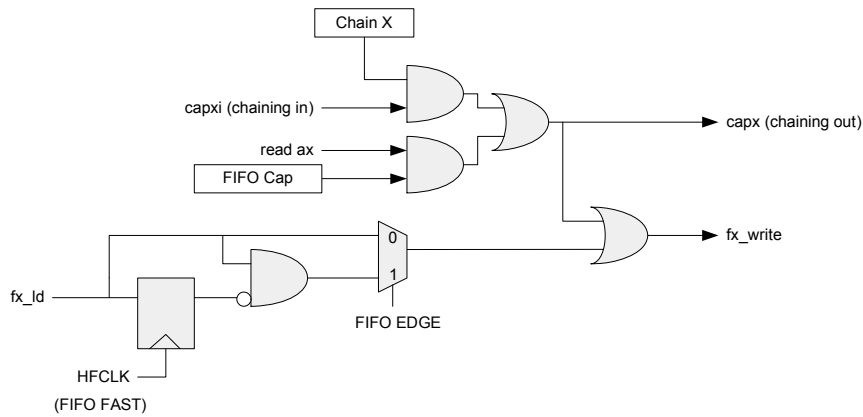
对生成 FIFO 加载的 fx_id 信号与软件捕获信号进行“OR”（或）运算；同时使用硬件和软件捕获时，则不可预测这些结果。据一般规则，这些功能是相互排斥的；然而，通过进行下列设置，可以同时使用硬件和软件捕获：

- 将 FIFO 捕获时钟模式设置为 FIFO FAST
- 将 FIFO 写入模式设置为 FIFO EDGE

进行这些设置后，硬件和软件捕获以相同的方式运行，在给定的 HFCLK 周期内，确认某个信号也会启动一个捕获。

在启动一个软件捕获前，建议先清除固件中的目标 FIFO（UDB ACTL 寄存器）。这样会将 FIFO 读和写指针初始化为已知状态。

图 15-12. 软件捕获配置



FIFO 控制位

在进行正常操作时，CPU 固件通过使用辅助控制寄存器（ACTL）中的 4 位来控制 FIFO。

FIFO0 CLR 和 FIFO1 CLR 位用于复位或清理 FIFO。向这些位中的某一位写入 ‘1’ 后，可复位相应的 FIFO。为了继续进行 FIFO 操作，必须将该位回写成 ‘0’。如果该位保持激活状态，给定的 FIFO 将被禁用并作为无状态的一字节缓冲区运行。可以将数据写入到 FIFO 内；可以立即读取该数据并且可以随时覆盖掉该数据。使用 Fx INSEL[1:0]（UDB CFG15 寄存器）配置位的数据方向仍处于有效状态。

FIFO0 LVL 和 FIFO1 LVL 位控制了 4 字节 FIFO 确认总线状态处于被确认的等级（总线可以对 FIFO 进行读或写操作）。FIFO 总线状态的含义取决于所配置的方向，如表 15-5 中所示。

表 15-5. UDB ACTL 寄存器中 FIFO 等级的控制位

FIFOxLVL	输入模式（总线正在写入 FIFO）	输出模式（总线正在读取 FIFO）
0	未满 至少可以写入一个字节	非空 至少可以读取一个字节
1	至少一半为空 至少可以写入两个字节	至少一半为满 至少可以读取两个字节

FIFO 异步操作

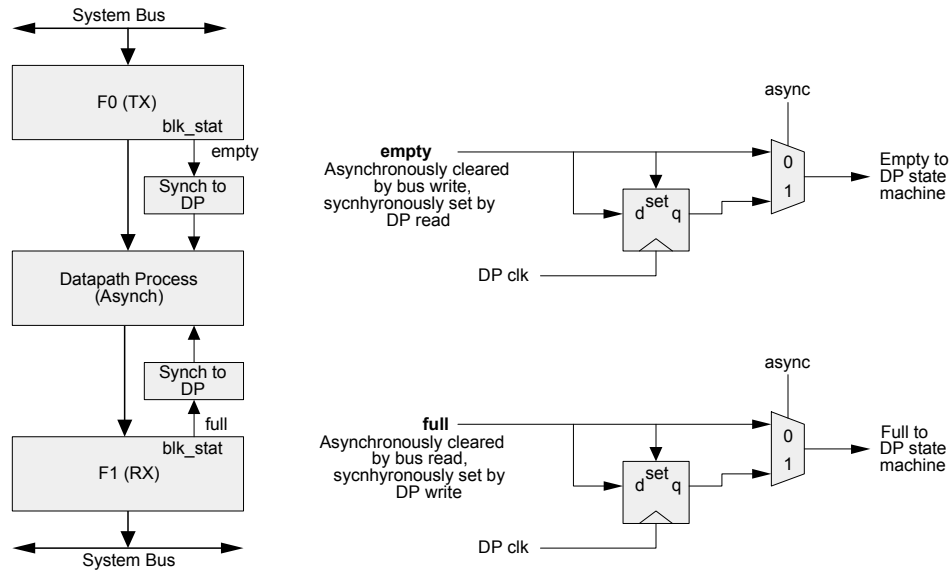
图 15-13 显示的是 FIFO 异步操作的概念。例如，假设 F0 被设置为输入模式和 F1 被设置为输出模式。这是 TX 和 RX 寄存器的典型配置。

在 TX 侧，数据路径状态机使用 “empty”（空）来确定是否有字节可用。“Empty”（空）与 DP 状态机是同步设置的，但被异步清除的（由于进行了一个总线写操作）。被清除时，该状态与 DP 状态机同步。

在 RX 侧，数据路径状态机使用 “full”（满）来确定 FIFO 中是否还存在空间，以执行写操作。Full（满）与 DP 状态机是同步设置的，但被异步清除的（由于进行了一个总线读操作）。被清除时，该状态与 DP 状态机同步。

通过使用 UDB CFG16 寄存器的一个单 FIFO ASYNCH 位，可以使能该同步方式；被设置时，它适用于两个 FIFO。它仅适用于模块状态，因为它假设总线状态是通过中断程序自然同步的。

图 15-13. FIFO 异步操作



FIFO 上溢操作

通过使用 FIFO 状态信号可以安全实现内部（数据路径）和外部（CPU）读/写操作。而 FIFO 不具备下溢和上溢条件的内置保护。如果 FIFO 已满，并且发生后续写入操作（上溢），则新的数据将覆盖掉 FIFO 中前部分的数据（当前数据被输出，将读取下一个数据）。如果 FIFO 为空，并且发生后续读取操作（下溢），则读取的值是未定义值。无论下溢和上溢情况如何，FIFO 指针都将保持正确。

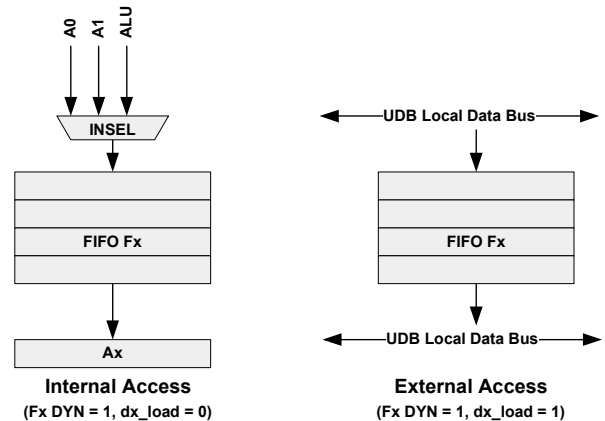
FIFO 时钟反转选项

每个 FIFO 的 UDB CFG16 寄存器都有一个名称为 Fx CK INV 的控制位，用于控制 FIFO 时钟的极性，使之与 DP 时钟的极性相对。默认情况下，在运行过程中，FIFO 的极性和 DP 时钟极性相同。置位该位时，FIFO 的运行极性与 DP 时钟的极性相对。这样便支持“两个时钟沿”的通信协议（如 SPI）。

FIFO 动态控制

一般情况下，FIFO 通过静态方式被配置为输入或输出模式。另外，可以将每一个 FIFO 配置为一种模式，其中方向是由路由信号动态控制的。通过使用每个 FIFO 中的一个配置位（UDB CFG17 寄存器的 Fx DYN 位）可以使能该模式。图 15-14 显示了动态 FIFO 模式的可用配置。

图 15-14. FIFO 动态模式



在内部访问模式下，数据路径可以对 FIFO 进行读/写操作。在该配置中，必须配置 Fx INSEL 位以选择 FIFO 写操作源。在该模式下，Fx INSEL = 00（CPU 总线源）无效，因此该值必须为 01、10 或 11（A0、A1 或 ALU）。请注意，只能对相应的累加器进行读访问；在该模式下，数据寄存器目标不可用。

在外部访问模式下，CPU 可以对 FIFO 进行读/写操作。通过使用数据路径路由信号可以在内部和外部访问之间进行动态切换。数据路径输入信号 d0_load 和 d1_load 都用于该控制操作。请注意，在动态控制模式下，不能使用 d0_load 和 d1_load 进行从 F0/F1 加载 D0/D1 寄存器。dx_load 信号可以由任何路由信号（包括各常数）驱动。

在一个使用示例中，开始进行外部访问（dx_load = 1）后，CPU 可以将一个或多个数据字节写入到 FIFO 内。然后切换到内部访问（dx_load = 0），数据路径可以对数据进行各种操作。切换回外部访问时，CPU 可以读取计算的结果。

由于始终要将 Fx INSEL 设置为 01、10 或 11（A0、A1 或 ALU）（表示处于正常操作的“输出模式”），因此 FIFO 状

态信号具有以下的含义（独立于 Fx LVL 控制）。

表 15-6. FIFO 状态

状态信号	含义	Fx LVL = 0	Fx LVL = 1
fx_blk_stat	写状态	FIFO 已满	FIFO 已满
fx_bus_stat	读状态	FIFO 非空	至少一半已满

由于数据路径和 CPU 可以对 FIFO 进行读和写操作，因此不再将这些信号视为“模块”和“总线”状态。blk_stat 信号用于写状态，bus_stat 信号用于读状态。

15.2.2.3 FIFO 状态

有 4 个 FIFO 状态信号，每个 FIFO 使用其中两个：fifo0_bus_stat、fifo0_blk_stat、fifo1_bus_stat 和 fifo1_blk_stat。这些信号的含义取决于已给 FIFO 的方向，该方向由静态配置确定。

15.2.2.4 数据路径 ALU

ALU 内核包含 3 个独立的 8 位可编程功能，分别为：一个算术 / 逻辑单元、一个移位器单元和一个掩码单元。有关详细信息，请参考 UDB 数据路径架构框图（图 15-6）。

算法和逻辑操作

表 15-7 中显示的是 ALU 功能，这些功能由配置 RAM 动态配置。

表 15-7. UDB DCFG 寄存器中的 ALU 功能

Func[2:0]	函数	操作
000	PASS	srca
001	INC	++srca
010	DEC	--srca
011	ADD	srca + srcb
100	SUB	srca - srcb
101	XOR	srca ^ srcb
110	AND	srca and srcb
111	OR	srca srcb

srca = ‘A’ 输入源连接到 ALU，srcb = ‘B’ 输入源连接到 ALU。请参见图 15-6。

Carry In（进位输入）

carry in 作为算术操作。表 15-8 显示的是某些函数的默认 carry in 值。

表 15-8. Carry In（移入）函数

函数	操作	默认的 Carry In（移入）实现
INC	++srca	srca + 00h + ci，其中 ci 被强制设置为 1
DEC	--srca	srca + ffh + ci，其中 ci 被强制设置为 0
ADD	srca + srcb	srca + srcb + ci，其中 ci 被强制设置为 0
SUB	srca - srcb	srca + ~srcb + ci，其中 ci 被强制设置为 1

除了可用于进位操作的该默认算术模式外，还有其他三个进位选项。使用 CFG13 寄存器中的 CI SELA 和 CI SELB 配置位可以确定一个给定周期的 carry in。通过动态配置 RAM 可以按周期性选择 A 或 B 配置。在表 15-9 中定义了各选项。

表 15-9. UDB CFG13 中附加的移入函数

CI SEL A CI SEL B	Carry 模式	说明
00	默认	默认算术模式在表 15-8 中介绍。
01	寄存	Carry 标志 — 上一周期中的 carry 结果。该模式用于实现 add with carry（进位加）和 subtract with borrow（借位减）运算。可以在连续周期内使用该模式，以仿真一个双精度操作。
10	互连	在其他位置生成 Carry 信号，并将它路由到该输入端。该模式可用于实现可控制的计数器。
11	级联	Carry 信号从前一数据路径级联。该模式可用于实现两个或更多数据路径上精度更高的单周期操作。

使用路由进位时，它在每个算术函数的相应意义在表 15-10 中显示。请注意，对于递减和减法函数，该进位为低电平有效（反转）。

表 15-10. 互连进位输入函数

函数	进位输入 信号极性	进位输入有效	进位输入无效
INC	真	++srca	srca
DEC	反转	--srca	srca
ADD	真	(srca + srcb) + 1	srca + srcb
SUB	反转	(srca - srcb) - 1	(srca - srcb)

Carry Out（进位输出）

Carry out 是一个可选的数据路径输出，由静态可编程的当前定义 MSB 位置派生。该值还可以作为可选的 carry in 链接到下一个最高有效模块。请注意，对于递减和减法函数，carry out 被反转。

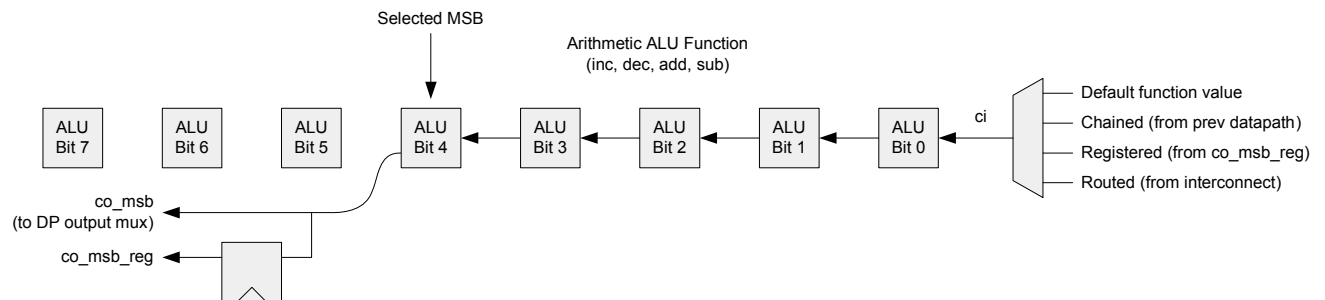
表 15-11. 进位输出函数

函数	进位输出信号极性	进位输出有效	进位输出无效
INC	真	$++src_a == 0$	src_a
DEC	反转	$--src_a == -1$	src_a
ADD	真	$src_a + src_b > 255$	$src_a + src_b$
SUB	反转	$src_a - src_b < 0$	$(src_a - src_b)$

进位结构

图 15-15 显示的是 carry in 选项和用于生成 carry out 的 MSB 选项。寄存的 carry out 值可以作为后续算术操作的 carry in 使用。在多个周期中，可以通过该特性来实现更高精度功能。

图 15-15. 进位操作



移位操作

根据表 15-12，移位操作独立于 ALU 操作。

表 15-12. UDB DCFG 寄存器中的移位操作函数

Shift[1:0]	函数
00	通过
01	向左移位
10	向右移位
11	半字节交换

Shift out 值可作为一个数据路径输出使用。Shift out right（右移输出 — sor）和 shift out left（左移输出 — sol_msb）共享该输出选择。静态配置位（UDB CFG15 寄存器中的 SHIFT SEL）用于确定作为数据路径输出使用的移位输出。没有发生移位时，sor 和 sol_msb 信号分别被定义为 ALU 功能的最低有效位和最高有效位。

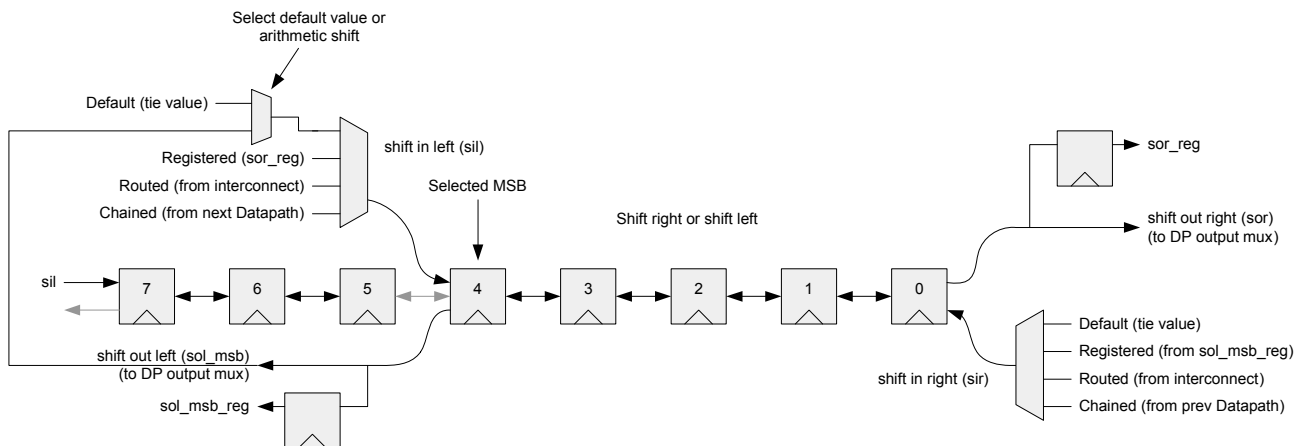
通过 SI SELA 和 SI SELB 配置位可以确定某个已给操作的 shift in 数据。通过动态配置 RAM 可以根据周期来选择 A 或 B 配置。Shift in 数据仅适用于左移和右移；它不适用于传送和半字节交换。表 15-13 显示的是左移和右移的选项以及使用情况。

表 15-13. UDB CFG15 寄存器中的移入函数

SI SEL A/ SI SEL B	移入源	说明
00	默认 / 算术	默认输入是 DEF SI 配置位的值（固定为 1 或 0）。然而，如果设置 MSB SI 位，默认输入将为当前定义的最高有效位（仅用于右移）。
01	寄存	移入值是由（前一周期）寄存的当前移出值驱动。“shift left”（左移）操作使用最后“shift out left”（左移输出）的值。“shift right”（右移）操作使用最后“shift out right”（右移输出）的值。
10	互连	“Shift in”是从路由通道中选择的（SI 输入）。
11	级联	“Shift in left”（左移输入）来自右侧相邻的数据路径，“shift in right”（右移输出）来自左侧相邻的数据路径。

Shift out left 数据来自当前定义的 MSB 位置（CFG14 寄存器中的 MSB_EN 和 MSB_SEL 位），并且从左边移入的数据（在右移操作中）进入当前定义的 MSB 位置。可以寄存 shift out 数据（左或右）并在后续周期内使用它。在多个周期中，可以使用该特性来实现更高精度的 shift in 操作。

图 15-16. 移位操作



请注意，由 MSB 选项隔离的各位仍然被移位。如该实例所示，位 7 通过右移移到 sil 值，位 5 通过左移移到位 4。从隔离位进行的右 shift out 或左 shift out 操作被抛弃。

ALU 掩码操作

通过 UDB 静态配置寄存器（CFG9）中的 8 位掩码寄存器（AMASK），可以定义掩码操作。在该操作中，ALU 的输出由掩码寄存器中的值掩码（进行 AND 运算）。ALU 掩码功能通常用于实现分辨率为 2 幂次的自由运行定时器和计数器。

15.2.2.5 数据路径输入和复用

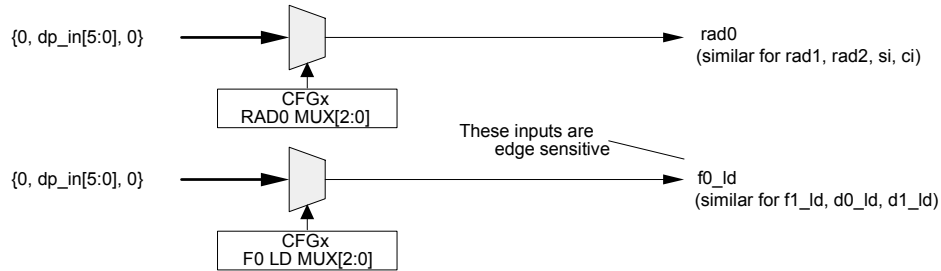
如表 15-14 中所示，数据路径共有 9 个输入，其中 6 个输入来自通道路由。这些输入包括配置 RAM 地址、FIFO 和数据寄存器加载控制信号，以及数据输入的 shift in 和 carry in 信号。

表 15-14. 数据路径输入

输入	说明
RAD2 RAD1 RAD0	异步动态配置的 RAM 地址。共有八个用户可编程的 16 位字。每个字都包含了当前周期的数据路径控制位。各指令的序列均由这些地址输入控制。
F0 LD F1 LD	在给定的周期中激活它时，将使用一个 A0 或 A1 累加器数据或 ALU 输出的数据载入选定 FIFO。Fx INSEL[1:0] 配置位用于选择输入源。该输入是边沿有效输入。它在数据路径时钟上被采样；当检测到 ‘0’ 至 ‘1’ 的转换时，将在随后的时钟边缘上执行加载操作。
D0 LD D1 LD	在给定的周期中激活它时，将从相应的 FIFO Fx 加载 Dx 寄存器。该输入为边沿敏感输入。它在数据路径时钟上被采样；当检测到 ‘0’ 至 ‘1’ 的转换时，将在随后的时钟边缘上执行加载操作。
SI	这是可用于左移输入或右移输入的数据输入值。
CI	它是 carry in select control（进位输入选择控制）被设置为 “routed carry”（互连进位）时使用的移入值。

如图 15-17 中所示，每个输入都有一个 6 输入 1 输出的复用器，因此所有输入都是可交换的。可以通过电平敏感或边沿敏感的方式来处理这些输入。RAM 地址、shift in 和 data in 值都是电平敏感的；FIFO 和数据寄存器加载信号则是边沿敏感的。

图 15-17. 数据路径输入选择



15.2.2.6 CRC/PRS 支持

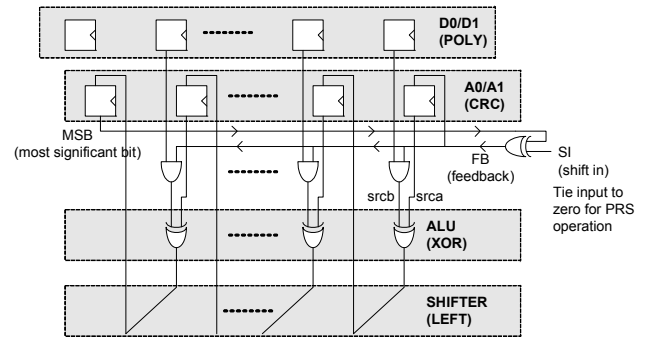
数据路径支持循环冗余校验（CRC）和伪随机序列（PRS）的生成。链接信号在各数据路径模块之间被路由，以支持超过 8 位的 CRC/PRS 位长度。

CRC/PRS 计算中最高有效模块的最高有效位（MSB）被选中，（在各个模块上链接）并被路由到最低有效模块。然后最高有效位与数据输入（SI 数据）进行 XOR 运算，以提供反馈（FB）信号。这时反馈信号被路由（并在各模块上链接）到最高有效模块。该反馈值被使用于所有模块，以便传输使用当前累加器值将多项式（Data0 或 Data1 寄存器中）进行 XOR 运算后得到的值。

图 15-18 显示了 CRC 操作的结构配置。PRS 配置是相同的，但 shift in（SI）被连接到 ‘0’。在 PRS 配置中，D0 或 D1 包含了多项式的值，而 A0 或 A1 则包含了计算结果的初始（种子）值和 CRC 余数。

为了使能 CRC 操作，必须将动态配置 RAM 中的 CFB_EN 位设置为 ‘1’。这样可使能对 SRCB ALU 输入和 CRC 反馈信号进行的 AND 运算。将该位置位为 ‘0’ 时，反馈信号被驱动为 ‘1’。这样可以进行普通的算术操作。按周期动态控制该位时，CRC/PRS 操作可以与其他算术操作交叉进行。

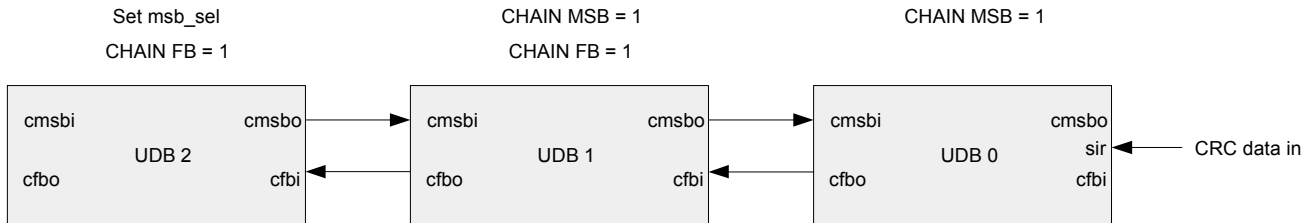
图 15-18. CRC 功能结构



CRC/PRS 链接

图 15-19 显示的是 3 个 UDB 上的 CRC/PRS 链接的实例。这种配置可以支持 17 到 24 位操作。根据链接中数据路径的位置来设置链接控制位，如该图所示。

图 15-19. CRC/PRS 链接配置



CRC/PRS 反馈信号 (cfbo、cfbi) 如下链接:

- 如果已给模块为最低有效模块，那么反馈信号将从该模块的内置逻辑中生成。该逻辑从右边获取 shift in (sir)，并将该值与最高有效位信号进行 XOR 运算。(对于 PRS，“sir”信号被连接到 ‘0’。)
- 如果已给模块不是最低有效模块，那么必须要设置 CHAIN FB 配置位，并且反馈信号被传输给链接中的前一个模块。

CRC/PRS MSB 信号 (cmsbo、cmsbi) 如下链接:

- 如果已给模块为最高有效模块，那么 (根据多项式选择的) 最高有效位将通过使用 UDB CFG14 寄存器中的 MSB_SEL 配置位进行配置。
- 如果已给模块不是最高有效模块，则必须设置 UDB CFG14 寄存器的 CHAIN CMSB 配置位，并且最高有效位信号被传输给链接中的下一个模块。

CRC/PRS 多项式规范

例如，要想配置多项式，以将其编程到相应的 D0/D1 寄存器内，则需要考虑定义为 $x^{16} + x^{12} + x^5 + 1$ 的 CCITT CRC-16 多项式。图 15-20 中显示了从多项式获取数据格式的方式。

x^0 始终为 ‘1’，因此不需要进行编程。而对于多项式中剩余的每一项，一个 ‘1’ 被设置在显示对齐中合适的位置上。

注意：该多项式格式与通常指定的 Hex 格式稍微不同。例如，CCITT CRC16 多项式通常表示 1021H。要想转换到数据路径操作需要的格式，需要右移一个值，并将在最高有效位内添加一个 ‘1’。这时，加载到 D0 或 D1 寄存器内正确的多项式值将为 8810H。

图 15-20. CCITT CRC16 多项式格式

x^{16}	x^{15}	x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
x^{16}	+			x^{12}	+						x^5	+				1
1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	

CCITT 16-Bit Polynomial is 0x8810

CRC/PRS 的示例配置

下面显示的是 CRC/PRS 配置要求的总结（假设 D0 是多项式和 CRC/PRS 在 A0 中计算）：

1. 选择一个合适的多项式并将其写入到 D0 内。
2. 选择合适的种子值（例如，CRC 全为 ‘0’、PRS 全为 ‘1’），并将其写入到 A0 内。
3. 配置链接（若需要）。
4. 根据该多项式内所定义的 MSB_SEL 静态配置寄存器位来选择 MSB 位置，并设置 UDB CFG14 寄存器的 MSB_EN 位。
5. 配置动态配置 RAM 字域：
 - a. 选择 D0 作为 ALU “SRCB”（ALU B 输入源）
 - b. 选择 A0 作为 ALU “SRCA”（ALU A 输入源）
 - c. 为 ALU 功能选择 “XOR”
 - d. 为 SHIFT 功能选择 “SHIFT LEFT”
 - e. 选择 “CFB_EN”，以支持 CRC/PRS
 - f. 选择 ALU 作为 A0 写入源

如果是一个 CRC 操作，则为路由中的输入数据配置 “shift in right”，并在每个时钟上提供输入。如果是一个 PRS 操作，请将 “shift in right” 连接到 ‘0’。

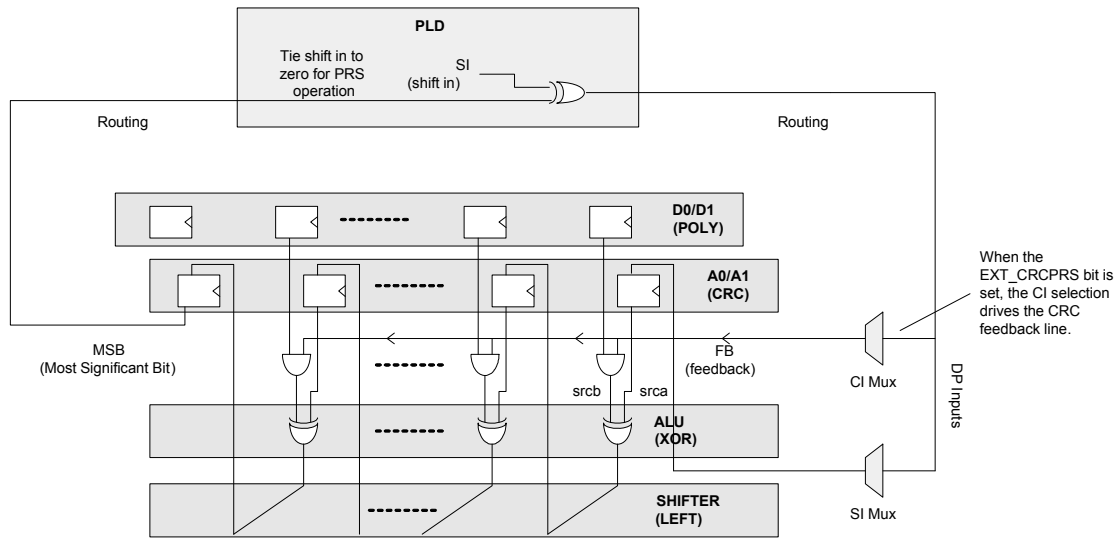
通过该配置给 UDB 提供时钟，会生成所需要的 CRC 或输出最高有效位。该位将被输出到 PRS 序列的布线。

外部 CRC/PRS 模式

通过设置静态配置位（UDB CFG16 寄存器的 EXT_CRCPRS），可以支持 CRC 或 PRS 的外部计算。如图 15-21 中所示，CRC 反馈的计算是在 PLD 模块中进行的。设置该位时，可以从 CI（Carry In）数据路径输入选择复用器直接驱动 CRC 反馈信号，不用进行内部计算。该图显示的是支持 8 位 CRC 或 PRS 的简单配置。一般情况下使用内置电路，但该特性提供了更复杂的性能，如在某个 UDB 中通过时分复用实现了 16 位 CRC/PRS 功能。

在该模式下，UDB DCFG0 寄存器的动态配置 RAM 位 CFB_EN 仍然控制着是否将 CRC 反馈信号与 SRCB ALU 输入进行 AND 运算。因此，如内置 CRC/PRS 操作，该功能可以与其他功能交叉使用（若需要）。

图 15-21. 外部 CRC/PRS 模式



15.2.2.7 数据路径输出和复用

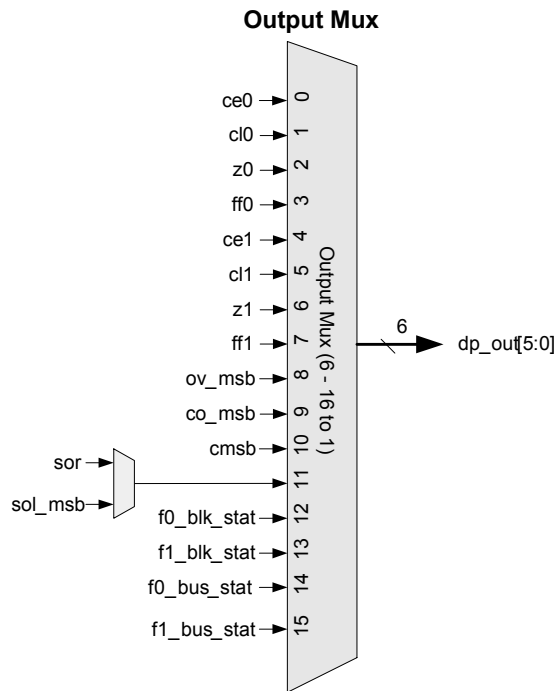
各个条件均由各寄存累加器值、ALU 输出和 FIFO 状态生成。可以将这些条件驱动给数字路由，以使用于其他 UDB 模块，并作为各个中断或者将它们驱动到 I/O 引脚上。表 15-15 显示的是 16 种可能条件。

表 15-15. 数据路径条件生成

名称	条件	链接	说明
ce0	“等于”比较	是	A0 == D0
cl0	“小于”比较	是	A0 < D0
z0	“0”值检测	是	A0 == 00h
ff0	“1”值检测	是	A0 == FFh
ce1	“等于”比较	是	A1 或 A0 == D1 或 A0（动态选择）
cl1	“小于”比较	是	A1 或 A0 < D1 或 A0（动态选择）
z1	“0”值检测	是	A1 == 00h
ff1	“1”值检测	是	A1 == FFh
ov_msb	溢出	否	Carry(msb) ^ Carry(msb-1)
co_msb	进位	是	对 MSB 定义位进行进位
cmsb	CRC MSB	是	CRC/PRS 功能的最高有效位
So	移出	是	移位输出选择
f0_blk_stat	FIFO0 模块状态	否	定义取决于 FIFO 的配置情况
f1_blk_stat	FIFO1 模块状态	否	定义取决于 FIFO 的配置情况
f0_bus_stat	FIFO0 总线状态	否	定义取决于 FIFO 的配置情况
f1_bus_stat	FIFO1 总线状态	否	定义取决于 FIFO 的配置情况

共有 6 个数据路径输出端。如图 15-22 中所示，每个输出端都有一个 16-1（16 输入 1 输出）的复用器，通过该复用器可以将 16 个信号中的任意一个路由到任意数据路径输出端。

图 15-22. 输出复用连接



比较器

共有两个比较器，其中一个具有固定源（比较器 0），而另一个则具有动态的可选源（比较器 1）。每个比较器均有一个 8 位静态编程的掩码寄存器，通过它可在一个指定的位字段中进行比较操作。默认情况下，该掩码无效（比较所有位），因此必须使它。

可动态配置比较器 1 输入端。如表 15-16 所示，比较器 1 共有四种选项，均适用于“小于”和“等于”条件。UDB CFG12 寄存器中的 CMP SELA 和 CMP SELB 配置位决定了可能的比较配置。通过 UDB DCFG0 寄存器中的动态配置 RAM 位 CMP SEL，可以按周期选择 A 或 B 配置。

表 15-16. 比较配置

CMP SEL A CMP SEL B	比较器 1 的比较配置
00	将 A1 与 D1 进行比较
01	将 A1 与 A0 进行比较
10	将 A0 与 D1 进行比较
11	将 A0 与 A0 进行比较

比较 0 和比较 1 可被独立链接到前一个数据路径生成的条件（按照地址顺序）。通过 UDB CFG14 寄存器中的 CHAIN0 和 CHAIN1 位可以确定链接比较器。图 15-23 显示的是比较相等链接，这便是该模块中的比较相等和前一模块中的链接输入进行“AND”（和）运算的结果。

图 15-23. 比较“相等”链接

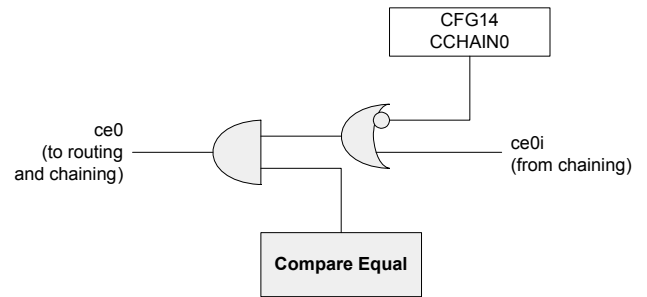
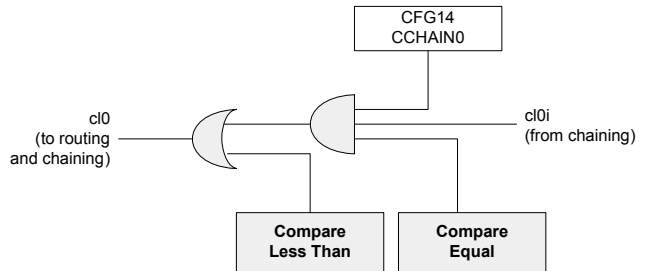


图 15-24 显示的是比较“小于”链接。在这种情况下，“小于”由该模块中无条件的比较小于输出形成。该模块的比较“小于”与“相等”进行“OR”运算，并且前一模块中的链接输入被确认为比较“小于”。

图 15-24. 比较“小于”链接



全零和全一检测

每个累加器均有专用的全零检测和全一检测条件。可以静态链接这些条件，如 UDB 配置寄存器中所指定的情况。通过 UDB 配置寄存器中所指定的信息，可确定是否链接这些条件。零检测的链接和相等比较是相同的概念。如果链接被使能，那么将对连续链接数据进行“AND”运算。

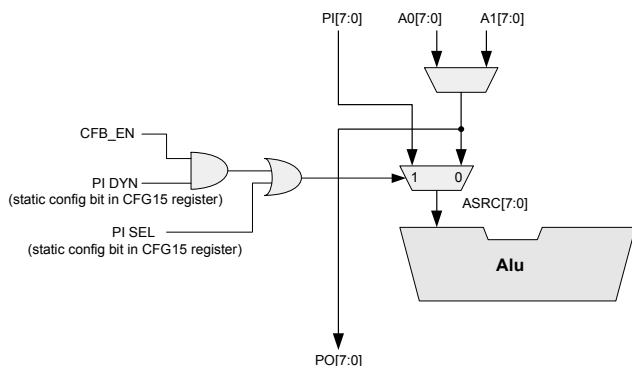
上溢

上溢是指对 carry（进位）移入 MSB（最高有效位）和 carry 移出 MSB 进行“XOR”运算。计算操作在当前定义的 MSB 实现，如 MSB_SEL 位所指定。不可链接该条件。然而，如果进位在各模块之间链接，而且在多精度功能的最高有效数据路径中执行计算操作，该操作将为有效的。

15.2.2.8 数据路径的并行输入和输出

如图 15-25 中所示，数据路径并行输入（PI）和并行输出（PO）信号对于直接将路由数据传输给数据路径和从数据路径输出提供的能力有限。并行输出信号始终可以路由，并作为 A0 和 A1 之间的 ALU asrc 选择。

图 15-25. 数据路径的并行输入 / 输出



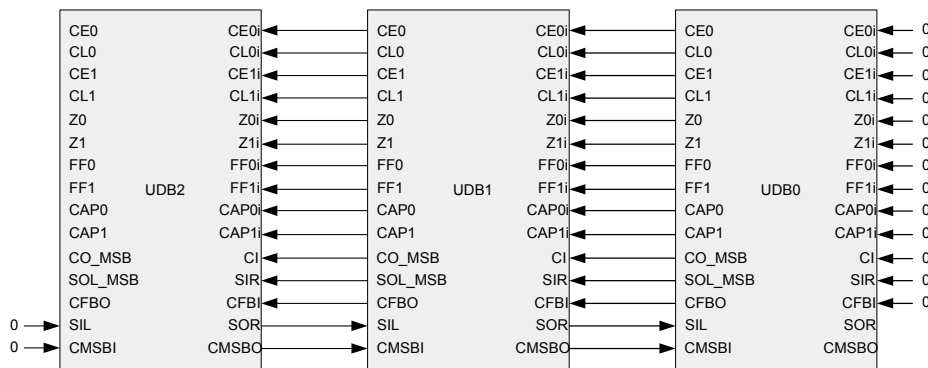
需要为 ALU 的输入选择并行输入。两个可用的选项为静态操作或动态操作。对于静态操作，UDB CFG15 寄存器中的 PI SEL 位会强制将 ALU asrc 作为 PI。UDB CFG15 寄存器中的

PI DYN 位用于使能 PI 动态操作。当该操作被使能，并假设 PI SEL 为 ‘0’ 时，CFB_EN (UDB DCFG0 寄存器) 动态控制位可控制 PI 复用器。CFB_EN 主要用于使能 PRS/CRC 功能。

15.2.2.9 数据路径链接

每个数据路径模块都包含了一个 8 位的 ALU，用于将各进位、移位数据、捕获触发器和条件信号链接到最近的相邻数据路径，从而提供精度更高的算术功能和移位器。通过这些专用的链接信号，可以有效地实现单周期 16、24 和 32 位功能，而不会发生通路延迟由资源的时序误差。另外，捕获链接还支持在链接模块中对累加器进行原子读操作。如图 15-26 所示，所有生成的条件和捕获信号链接的方向都是从最低有效模块到最高有效模块。Shift left（向左移位）也是从最低有效模块链接到最高有效模块。Shift right（向右移位）是从最高有效模块链接到最低有效模块。反馈的 CRC/PRS 链接信号则从最低有效模块链接到最高有效模块。MSB 输出是从最高有效模块链接到最低有效模块。

图 15-26. 数据路径链接流

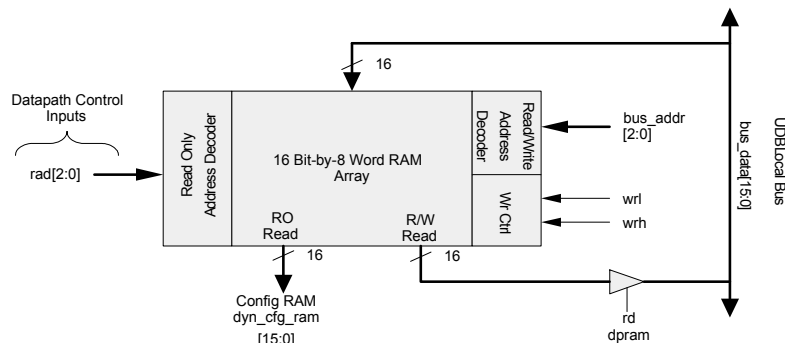


15.2.2.10 动态配置 RAM

每个数据路径包含一个大小为 **8x16** 位字的动态配置 **RAM**，如图 15-27 中所示。该 **RAM** 的功能是根据数据路径所选的时钟按周期控制数据路径配置位。该 **RAM** 具有同步的读和写端口，以便通过系统总线加载配置。

此外，还提供了另一个异步读端口作为快速路径使用，用以将这些 **16** 位字转为控制位，然后传送到数据路径。异步地址输入是从数据路径输入中选择的，它们可以从通路由上的任何一个可能信号（如 I/O 引脚、PLD 输出、控制模块输出或其他数据路径输出）生成。异步读取路径的主要功能是为数据路径位提供快速单周期解码。

图 15-27. 配置 RAM I/O



下面显示的是动态配置 **RAM** 字的字段信息。后面是各个字段的使用情况。

寄存器	地址	15	14	13	12	11	10	9	8
CFGRAM	61h - 6Fh (奇数)	FUNC[2:0]			SRCA	SRCB[1:0]		SHIFT[1:0]	

寄存器	地址	7	6	5	4	3	2	1	0
CFGRAM	60h - 6Eh (偶数)	A0 WR SRC[1:0]		A1 WR SRC[1:0]		CFB EN	CI SEL	SI SEL	CMP SEL

表 15-17. 动态配置的快速参考

字段	位	参数	值
FUNC[2:0]	3	ALU 功能	000: PASS
			001: INC SRCA
			010: DEC SRCA
			011: ADD
			100: SUB
			101: XOR
			110: AND
			111: OR
SRCA	1	ALU A 输入源	0: A0 1: A1
SRCB	2	ALU B 输入源	00: D0
			01: D1
			10: A0
			11: A1
SHIFT[1:0]	2	SHIFT 功能	00: PASS
			01: 左移
			10: 右移
			11: 半字节交换

表 15-17. 动态配置的快速参考

字段	位	参数	值
A0 WR SRC[1:0]	2	A0 写入源	00: 无
			01: ALU
			10: D0
			11: F0
A1 WR SRC[1:0]	2	A1 写入源	00: 无
			01: ALU
			10: D1
			11: F1
CFB EN	1	CRC 反馈使能	0: 使能 1: 禁用
CI SEL	1	Carry In 配置选择	0: ConfigA 1: ConfigB ^a
SI SEL	1	Shift In 配置选择	0: ConfigA 1: ConfigB ^a
CMP SEL	1	比较配置选择	0: ConfigA 1: ConfigB ^a

a. RAM 域为 CI、SI 和 CMP 选择两个预定义状态设置中的一个。请参考表 15-9、表 15-13 和表 15-16 中的内容。

15.2.3 状态和控制模块

图 15-28 显示的是状态和控制模块的顶层视图。控制寄存器驱动到路由，从而为 UDB 操作提供固件控制输入。通过在路由过程中读取状态寄存器的数值，固件可以监控 UDB 操作的状况。

图 15-29 显示的是状态和控制模块的详细视图。该模块的主要用途是协调 CPU 固件与内部 UDB 操作之间的交互。然而，由于该模块能够灵活地连接到路由矩阵，因此将其配置为执行其他功能。

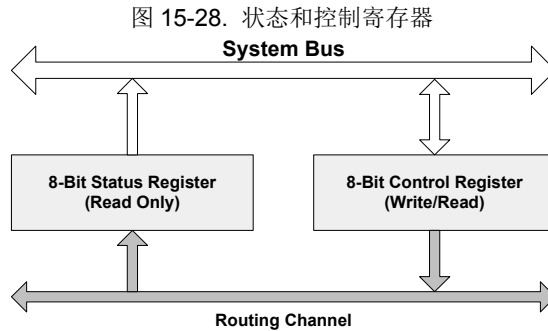
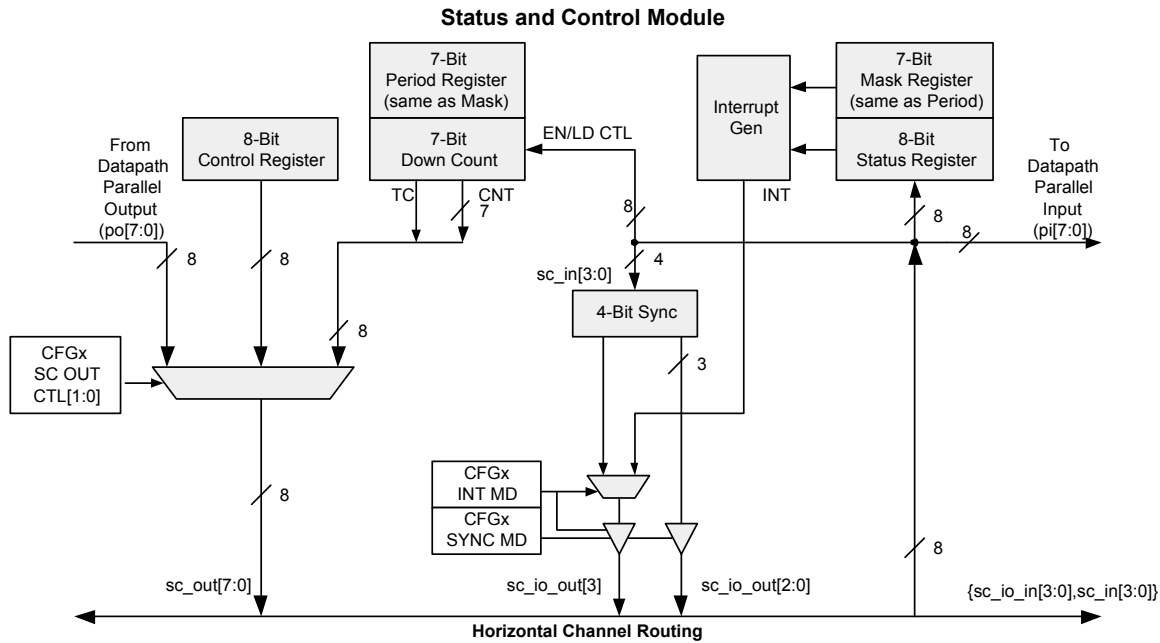


图 15-29. 状态和控制模块



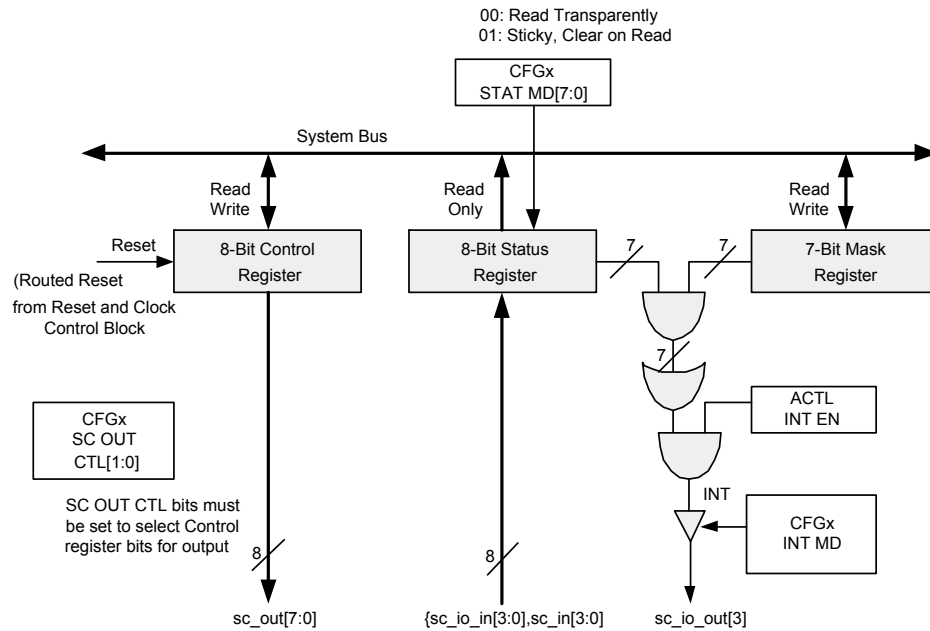
有以下各种操作模式：

- **状态输入** — 路由信号的状态可以为输入，可以捕获它作为状态并由 CPU 读取。
- **控制输出** — CPU 可以通过写入到控制寄存器内驱动路由状态。
- **并行输入** — 到数据路径并行输入。
- **并行输出** — 从数据路径并行输出。
- **寄存器模式** — 在该模式下，控制寄存器作为 7 位递减计数器运行，并且具有可编程周期和自动重新加载的特性。通过配置路由输入，可以控制计数器的使能和重新加载操作。当该模式被使能时，控制寄存器操作不可用。
- **同步模式** — 在该模式下，状态寄存器作为一个 4 位的双同步器运行。使能该模式时，状态寄存器操作不可用。

15.2.3.1 状态和控制模式

该模块运行于状态和控制模式时，它可作为状态寄存器、中断掩码寄存器和控制寄存器使用，如图 15-30 中所示的配置。

图 15-30. 状态和控制操作



状态寄存器操作

每个 UDB 都有一个 8 位的只读状态寄存器。该寄存器的输入来自数字路由结构中的所有信号。状态寄存器是非保存的寄存器，它在睡眠间隔内丢失状态，并在唤醒时复位为 0x00。可以独立编程每一位，以便让它能以两个方式中的一个运行，如表 15-18 中所示。

表 15-18. UDB CFG20 寄存器中状态寄存器模式选择

STAT MD	说明
0	透明读取。读操作会返回路由信号的当前值
1	粘滞位，在读取时被清除。在输入中产生的高电平信号被采样和捕捉。读取寄存器时，它将被清除。

状态寄存器清除操作的重要功能是清除正被置位的位。这样，尚未置位的其他位将继续捕获状态，从而过程可保持一致。

透明状态读取

默认情况下，CPU 读取寄存器时也会透明地读取相关路由的状态。该模式可用于计算和寄存 UDB 内部的瞬变状态。

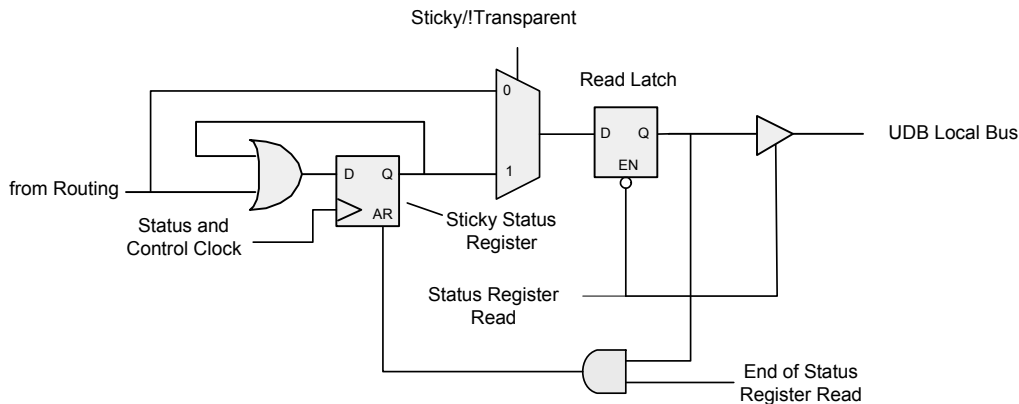
粘滞状态，在读取后被清除

在该模式下，状态和控制时钟完成每个周期后，会对状态寄存器输入进行采样。如果信号在指定采样中为高电平，则在状态位上被捕获，并且不管输入的后续状态如何它都会保持为高电平。当 CPU 读取状态寄存器时，将清除该位。状态寄存器的清除不依赖于任何模式，即使在 UDB 时钟被禁用时也同样发生；它根据 HFCLK 作为读取操作的一部分而发生。

读取期间锁存状态

图 15-31 显示的是状态读取逻辑的结构。粘滞状态寄存器后紧跟一个锁存。在读周期期间，无论指定读操作中的等待状态数量是多少，该锁存都用于锁存状态寄存器的数据并保持其稳定状态。

图 15-31. 状态读取逻辑



中断生成

几乎在所有功能中，中断的生成都取决于状态位的设置。如图 15-31 中所示，该功能作为掩码状态和 OR 减少状态被内置到状态寄存器逻辑内。只有状态输入的低 7 位可以与内置中断生成电路一起使用。最高有效位通常作为中断输出使用，并可以通过数字路由被路由给中断控制器。在该配置中，状态寄存器的最高有效位作为中断位的状态被读取。

15.2.3.2 控制寄存器操作

每个 UDB 都有一个 8 位控制寄存器。它作为系统总线上的标准读 / 写寄存器运行，其中这些寄存器位的输出可以用于驱动数字路由结构。

控制寄存器是非保持寄存器；它在睡眠间隔内丢失内容，并在唤醒时复位到 0x00。

控制寄存器的工作模式

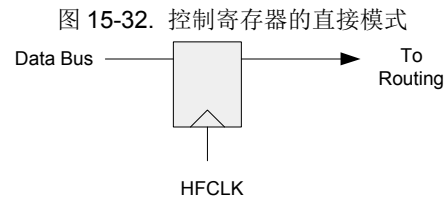
可以按位配置三种模式。该配置由 UDB CFG18 和 CFG198 寄存器内的两个 8 位寄存器 CTL_MD1[7:0] 和 CTL_MD0[7:0] 的位串联控制。例如，{CTL_MD1[0], CTL_MD0[0]} 控制着控制寄存器位 0 的模式，如表 15-19 所示。

表 15-19. UDB CFG18 和 CFG19 寄存器中控制寄存器位 0 的模式

CTL MD	说明
00	直接模式
01	同步模式
10	双同步模式
11	脉冲模式

控制寄存器的直接模式

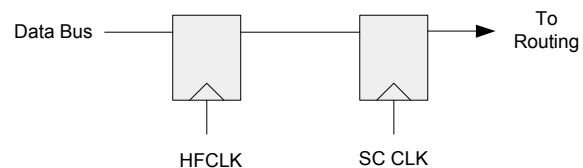
默认情况下为直接模式。如图 15-32 所示，当 CPU 写入到控制寄存器内时，该控制寄存器的输出将在该写周期内被直接驱动给路由。



控制寄存器的同步模式

如图 15-33 所示，在同步模式下控制寄存器输出由一个重新采样寄存器驱动。该寄存器由当前选定的状态和控制（SC）时钟提供时钟脉冲。这样允许已选的 SC 时钟（而不是 HFCLK）控制输出的时序。

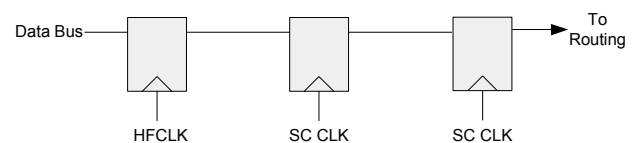
图 15-33. 控制寄存器的同步模式



控制寄存器的双同步模式

在双同步模式下，重新采样寄存器后，将添加由已选 SC 时钟提供脉冲的第二个寄存器，如图 15-34 中所示。这样，当 HFCLK 和 SC 时钟均异步时，可以强行执行电路板。

图 15-34. 控制寄存器的双同步模式



控制寄存器的脉冲模式

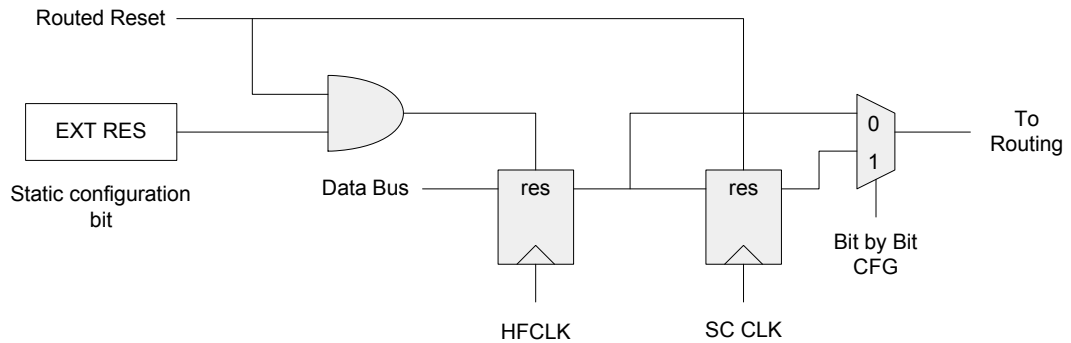
与同步模式相似，在脉冲模式下 SC 时钟也能对控制位进行重新采样；在总线写周期后，脉冲会在第一个 SC 时钟周期内开始。控制位的输出在一个完整的 SC 时钟周期内有效。在该时钟周期结束时，控制位被自动复位。

在该操作模式下，固件可以通过将 ‘1’ 写入到控制寄存器内来生成一个脉冲。写入 ‘1’ 后，它会被固件回读为 ‘1’，直到该脉冲结束为止。然后，该控制寄存器的值将被回读为 ‘0’。固件可以写入另一个 ‘1’，以启动另一个脉冲。只有完成前一个脉冲，才能生成新的脉冲。因此，脉冲生成的最大频率是所有其他 SC 时钟周期。

控制寄存器的复位

控制寄存器具有由 EXT RES 配置位控制的两种复位模式，如图 15-35 所示。当 EXT RES 位为 ‘0’（默认）时，在同步模式或脉冲模式下，路由复位输入将复位同步输出，但不会复位实际的控制位。当 EXT RES 位为 ‘1’ 时，路由复位输入将复位控制位和同步输出。

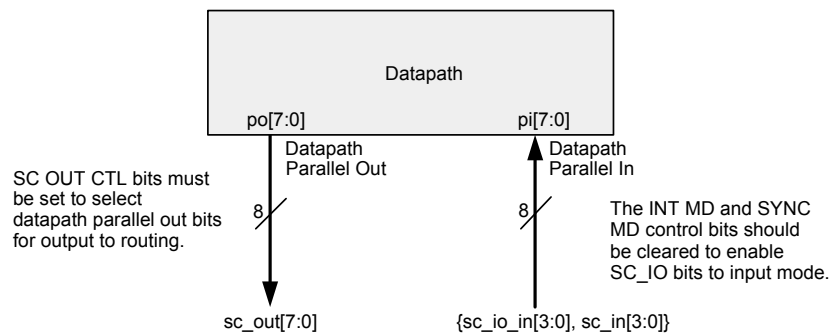
图 15-35. 控制寄存器的复位



15.2.3.3 并行输入 / 输出模式

在该模式下，状态和控制路由被连接到数据路径并行输入和并行输出信号，如图 15-36 所示。要想使能该模式，请通过设置 UDB CFG22 寄存器中的 SC OUT 配置位选择数据路径输出。该并行输入连接始终有效，但状态寄存器输入、计数器控制输入和中断输出共享这些路由连接。

图 15-36. 并行输入 / 输出模式



15.2.3.4 计数器模式

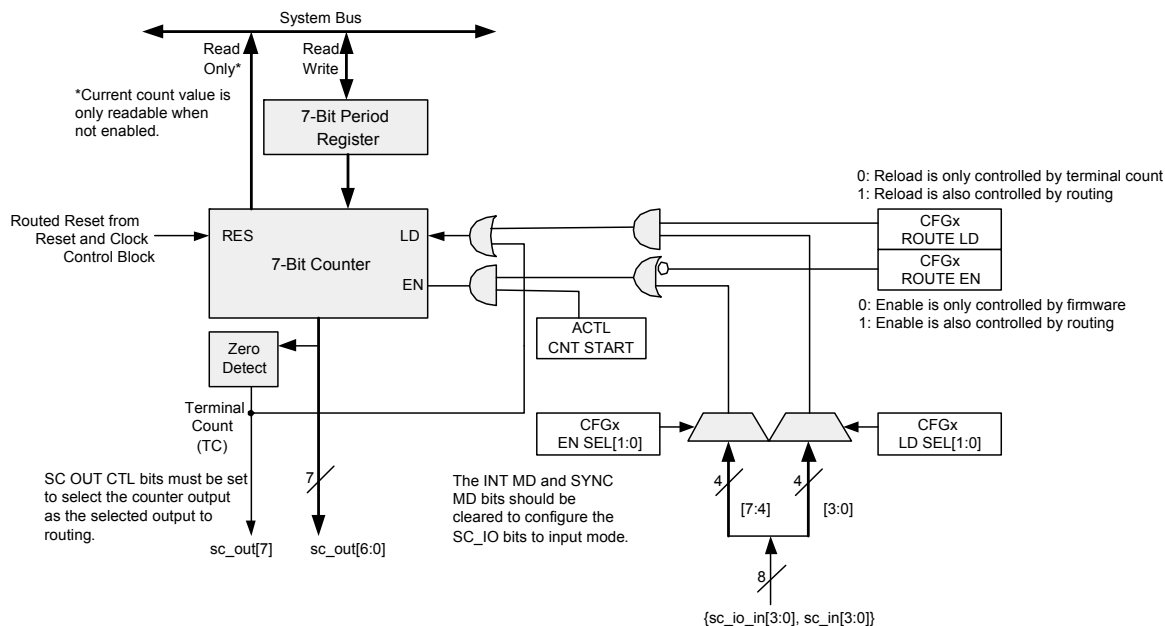
如图 15-37 所示，当模块处于计数器模式时，UDB 内部操作或固件应用程序可以使用 7 位递减计数器。该计数器具有下列特性：

- 一个 7 位读 / 写周期寄存器。
- 一个 7 位读 / 写计数寄存器。只有该计数器被禁用时，才能访问它。
- 在终端计数上 (0) 自动将周期重新加载到计数寄存器内。
- 辅助工作控制寄存器内的固件控制位（又称为 CNT START），用于启动和停止计数器。（这是一个覆盖的使能操作，必须进行设置才能执行可选路由的使能）。
- 路由中用于计数器使能和加载功能的可选动态控制的可选位：
 - EN，它是用于启动或停止计数的路由使能。
 - LD，它是用于强制重新加载周期的路由加载信号。当该信号被激活时，它将覆盖一个挂起的终端计数。激活期间，该信号是电平敏感型的，并会继续加载周期。

- 将 7 位计数作为 `sc_out[6:0]` 驱动到路由结构上。
- 将终端计数作为 `sc_out[7]` 驱动到路由结构上。
- 在默认模式下，终端计数被寄存。在备用模式下，终端计数被组合。
- 在默认模式下，必须激活路由使能信号（若使用），使路由由加载信号运行。在备用模式下，路由使能和路由加载信号独立运行。

要想使能计数器模式，必须将 UDB CFG22 寄存器中的 `SC_OUT_CTL[1:0]` 位设置为计数器输出。在该模式下，控制寄存器的普通操作无效。状态寄存器仍然可以用于读操作，但不应该用于生成中断，因为掩码寄存器作为计数器周期寄存器重用。周期寄存器作为保持寄存器实现，并能在睡眠间隔内保持其状态。在 N 个时钟的一个周期内，应该加载 $N-1$ 周期值。 $N=1$ (0 周期) 不能作为时钟分频值，并会使终端计数输出的结果始终为 1。`SYNC` (同步) 模式的使用情况取决于是否使用了动态控制输入 (`LD/EN`)。如果不使用这些输入，则同步模式不受到任何影响。如果使用这些输入，`SYNC` 模式无效。

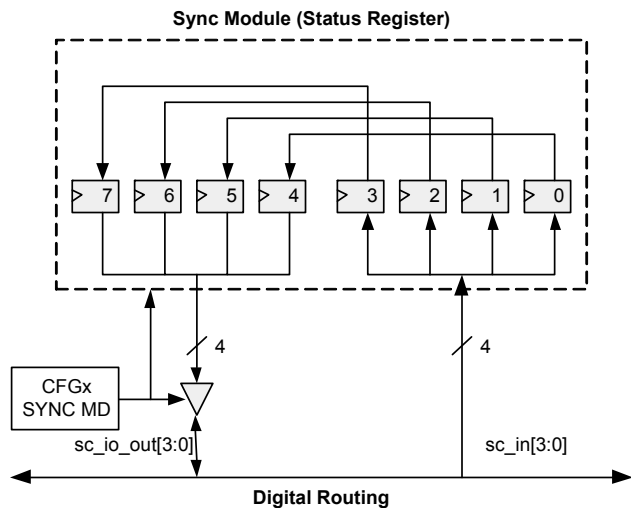
图 15-37. 计数器模式



15.2.3.5 同步模式

如图 15-38 所示，当 UDB CFG22 寄存器中 SYNC MD 位被置位时，状态寄存器可以作为 4 位双同步器运行，并由当前的 SC_CLK 时钟提供脉冲。该模式会用于实现异步信号（如 GPIO）的局部同步。被使能时，要同步的信号是从 SC_IN[3:0] 中选出的，输出被驱动到 SC_IO_OUT[3:0] 引脚，并且 SYNC MD 自动使 SC_IO 引脚进入输出模式。在该模式下，无论该模式的控制设置如何，状态寄存器的正常操作都不可用，并且状态粘滞位模式被强制关闭。在该模式下，该控制寄存器不受任何影响。在具体的限制下还可以使用该计数器。在该模式下，不能使能计数器的动态输入（LD/EN）。

图 15-38. 同步模式



15.2.3.6 状态和控制时序

状态和控制寄存器需要选择某个时钟，用于执行以下的工作模式：

- 状态寄存器，其中将任意位设置为粘滞位，在读取模式下被清除。
- 计数器模式中的控制寄存器。
- 同步模式。

该模式的时钟在复位和时钟控制模块中分配。请参见 第 140 页上的复位和时钟控制模块。

15.2.3.7 辅助控制寄存器

读 / 写辅助控制寄存器是一个特殊的寄存器，使用它来控制 UDB 中的固定功能硬件。通过该寄存器，CPU 可以动态控制中断、FIFO 和计数器操作。寄存器位和其说明如下所示。

辅助控制寄存器							
7	6	5	4	3	2	1	0
		CNT START	INT EN	FIFO1 LVL	FIFO0 LVL	FIFO1 CLR	FIFO0 CLR

FIFO0 清除，FIFO1 清除

FIFO0 CLR 和 FIFO1 CLR 位用于复位相应 FIFO 的状态。将 ‘1’ 写入到这些位后，会清除相应 FIFO 的状态。必须将这些位回写为 ‘0’，以便能够持续执行 FIFO 操作。当这些位被激活时，FIFO 会作为无状态的简单单字节缓冲区运行。

FIFO0 电平，FIFO1 电平

FIFO0 LVL 和 FIFO1 LVL 位控制了 4 字节 FIFO 确认总线状态处于被确认的等级（总线可以对 FIFO 进行读或写操作）。FIFO 总线状态的意义取决于所配置的方向，如表 15-20 中所示。

表 15-20. FIFO 电平控制位

FIFOx LVL	输入模式 (总线正在写入 FIFO)	输出模式 (总线正在读取 FIFO)
0	未满 至少可以写入一个字节	非空 至少可以读取一个字节
1	至少一半为空 至少可以写入两个字节	至少一半为满 至少可以读取两个字节

中断使能

当状态寄存器的中断生成逻辑被使能时，INT EN 位将关断所引起的中断信号。

计数启动

CNT START 位可用于使能和禁用计数器（仅在 SC_OUT_CTL[1:0] 位被配置为计数器输出模式时，该位才有效）。

15.2.3.8 状态和控制寄存器总结

表 15-21 总结了状态和控制寄存器的功能。请注意，寄存器可以是控制和掩码寄存器，也可以是计数和周期寄存器，具体情况取决于 UDB 的工作模式。

表 15-21. 状态、控制寄存器的功能汇总

模式	控制 / 计数	状态 / 同步	掩码 / 周期
控制	控制输出	状态输入或同步	状态掩码
计数	计数输出		计数周期 ^a
状态	控制输出或计数输出	状态输入	状态掩码
同步		同步	NA ^b

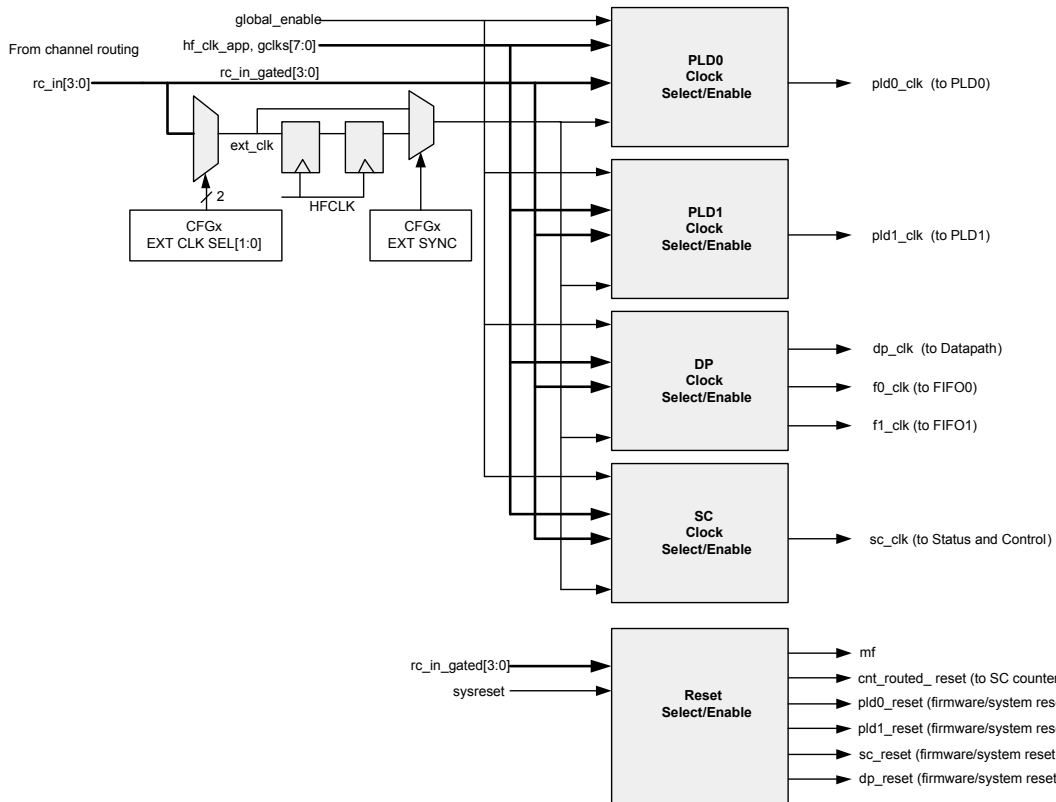
- a. 请注意，在计数器模式下，掩码寄存器作为周期寄存器运行，并不能执行掩码寄存器的功能。因此，计数器模式被使能时，中断输出不可用。
- b. 请注意，在同步模式下，不能使用状态寄存器，因此掩码寄存器也可用。然而，在计数模式下，它可以作为周期寄存器使用。

15.2.4 复位和时钟控制模块

复位和时钟模块主要用于从可用的全局系统时钟或 HFCLK 为每个 PLD、数据路径及状态和控制模块中选择一个时钟。它还为 UDB 模块提供动态以及基于固件的复位。如图 15-39 所示，共有四个时钟控制模块和一个复位模块。通过路由矩阵（RC_IN[3:0]）可以使用四个输入。每个时钟控制模块可以从这些路由输入中选择一个时钟使能源。另外，通过一个复用器可以选择路由输入中的一个作为外部时钟源使用。如该图所示，可以选择性地同步外部时钟源选择。共有六个供给每个 UDB 组件使用的时钟：四个 UDB 外部时钟、一个 HFCLK 和一个所选的外部时钟（ext clk）。可以将任意路由输入信号（rc_in）作为电平敏感型或边沿敏感型使能使用。该模块的复位功能为 PLD 模块和 SC 计数器提供了路由复位，并且为每个模块提供了固件复位能力，以此支持重新配置。

复位和时钟控制的 HFCLK 输入不同于系统 HFCLK。之所以将该时钟称为“hf_clk_app”，是因为关闭它的方式与其他 UDB 外设时钟相类似，并且它还用于 UDB 操作。系统 HFCLK 仅适用于 I/O 访问操作，并且进行访问操作时它被自动关断。数据路径时钟生成器生成了 3 个时钟：一个用于数据路径，另外两个则用于 FIFO（每个 FIFO 使用一个时钟）。

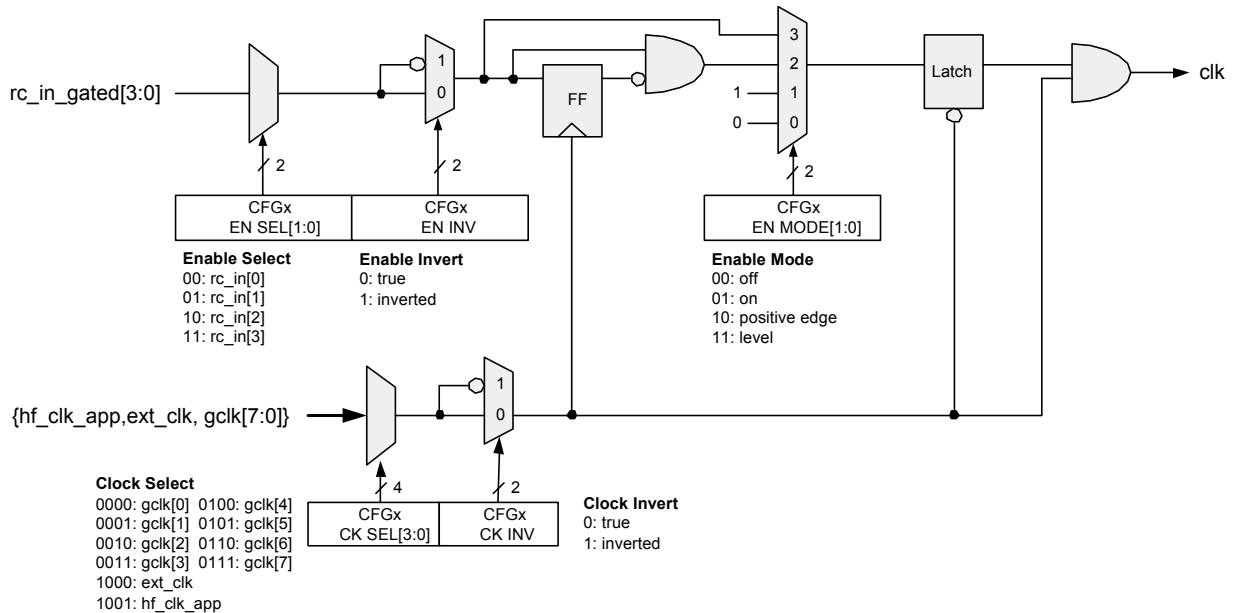
图 15-39. 复位和时钟控制



15.2.4.1 时钟控制

图 15-40 显示的是时钟选择和使能电路的一个实例。每个 UDB 包含了四个电路，其中两个用于 PLD 模块（每个 PLD 使用一个）、一个用于数据路径，另外一个用于状态和控制模块。该电路的主要组件包括：全局时钟选择复用器、时钟反转、时钟使能选择复用器、时钟使能反转和边沿检测逻辑。

图 15-40. 时钟选择 / 使能控制



时钟选择

四个 UDB 外设时钟（请参见第 77 页上的时钟系统章节）（gclk[0] 至 gclk[3]）被路由给所有 UDB；剩余的四个时钟（gclk[4] 至 gclk[7]）在 PSoC 4 器件系列中不受支持。可以从这些时钟选择一个。UDB 外设时钟是用户选择时钟分频器的输出。另外，还可以选择 HFCLK，在系统中该时钟的频率最高。被称为“hf_clk_app”的信号可从系统 HFCLK 单独路由得到。另外，可以选择外部路由信号作为时钟输入使用，以支持直接提供时钟脉冲的功能（如 SPI）。由于应用功能被映射到各 UDB 上任意边缘，所以每个 UDB 子组件模块的单独时钟选择支持高精度编程。

时钟反转

可以选择性地反转所选时钟。由于存在半周期时序路径，因此最大工作频率受限。当内部时钟被反转，并且其频率与 HFCLK 相同时，同时总线写操作和内部写操作（例如计数器正在计数时写入一个新的计数值）都不受支持。该限制会影响 A0、A1、D0、D1 和处于计数器模式的控制寄存器。

时钟使能选择

可以将该时钟使能信号路由到任意同步信号，并且可以从路由矩阵上四个输入中选择一个（这些输入均用于此模块）。

时钟使能反转

可以选择性地反转时钟使能信号。通过该特性，可以通过任何极性来使能生成时钟。

时钟使能模式

默认情况下，该时钟使能处于关闭状态。配置目标模块操作后，通过使用 UDB CFG24 寄存器中的 EN MODE[1:0] 位，软件可以将该模式设置为以下模式，如图 15-40 所示。

表 15-22. UDB CFG24 寄存器中的时钟使能模式

时钟使能模式	说明
OFF	时钟被关闭。
ON	时钟被打开。选定的全局时钟可以自由运行。
上升沿	检测到时钟使能输入的上升沿时，将生成门控时钟。使能输入的最大频率是选定的全局时钟的二分频。
电平	当时钟使能输入为高电平（‘1’）时，将生成时钟。

时钟使能的使用情况

时钟使能信号通常用于以下两种情况：

固件使能 — 假设几乎所有功能都需要一个固件时钟使能，用于启动和停止功能。由于映射到 UDB 矩阵的功能边界是任意的，因此它可以跨多个 UDB 和 / 或 UDB 部分；必须有自动使能给定功能的方式。该情况通常通过控制寄存器中的某一位实现的（该寄存器被路由至一个或多个时钟使能输入）。该情况还支持应用要求同时使能多个无关模块的场合。

仿真局部时钟生成 — 通过该特性，局部时钟会由 UDB 生成，并通过使用同步时钟使能方案（而不是由一个 UDB 直接给另一个提供时钟）分配到阵列中的其他 UDB。使用时钟使能模式下的上升沿特性可以消除时钟使能波形占空比的限制。

特殊的 FIFO 时序

数据路径 FIFO 具有特殊的时序注意事项。默认情况下，FIFO 时钟和数据路径时钟的配置相同。然而，FIFO 有特殊的控制位，用来修改时钟配置：

- 根据选定的数据路径时钟极性，可以反转 FIFO 时钟。
- 当 FIFO FAST 模式在 UDB CFG16 寄存器中被设置时，HFCLK 将覆盖 FIFO 通常使用的数据路径时钟选择。

15.2.4.2 复位控制

复位控制具有两种模式，分别为：兼容模式和备用模式。这两个模式由每个 UDB CFG31 寄存器中的 ALT RES 位控制。当该位为 ‘0’ 时，可实现兼容模式。该位为 ‘1’ 时，则实现备用模式。

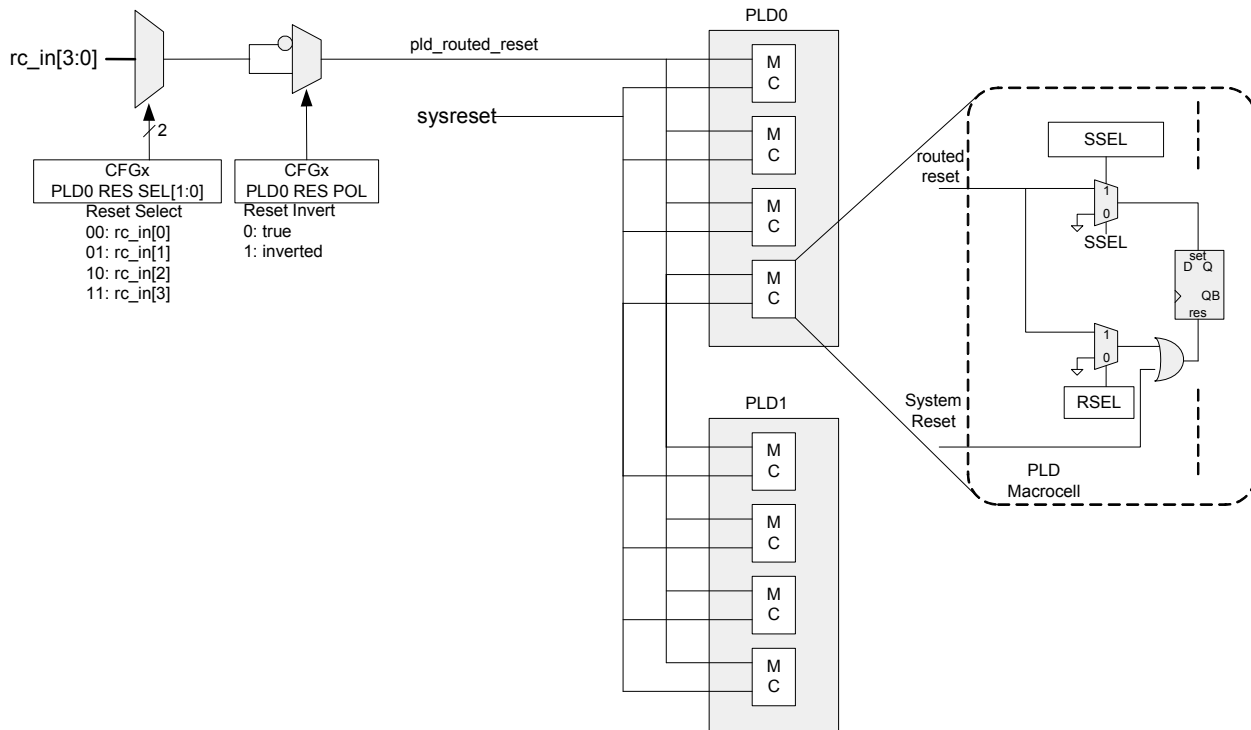
兼容复位方案

该方案包括一个路由复位，用于动态复位模块的嵌入式状态（该状态适用于各 PLD 宏单元和 SC 计数器）。

兼容 PLD 复位控制

图 15-41 显示的是使用路由动态复位的兼容 PLD 复位系统。

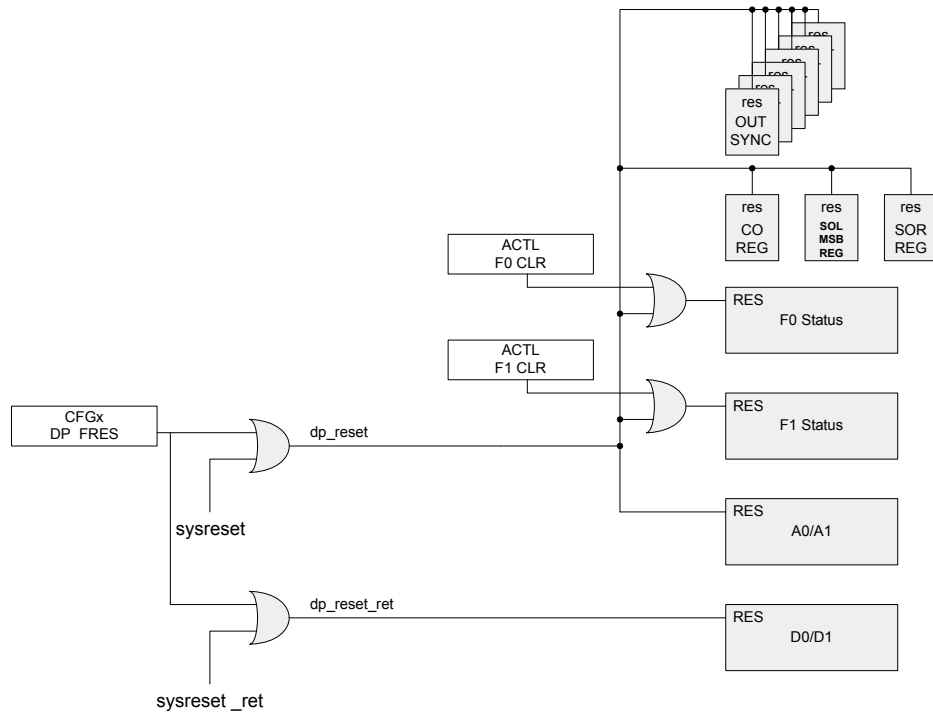
图 15-41. 兼容 PLD 复位结构



兼容的数据路径复位控制

图 15-42 显示的是使用固件复位的兼容数据路径复位系统。固件复位异步地清除 DP 输出寄存器、进位和移出标志、FIFO 状态、累加器以及数据寄存器。请注意，D0 和 D1 寄存器作为保持寄存器使用，它们在睡眠间隔中会保持其状态。FIFO 数据是未知的，因为它是基于 RAM 的数据。

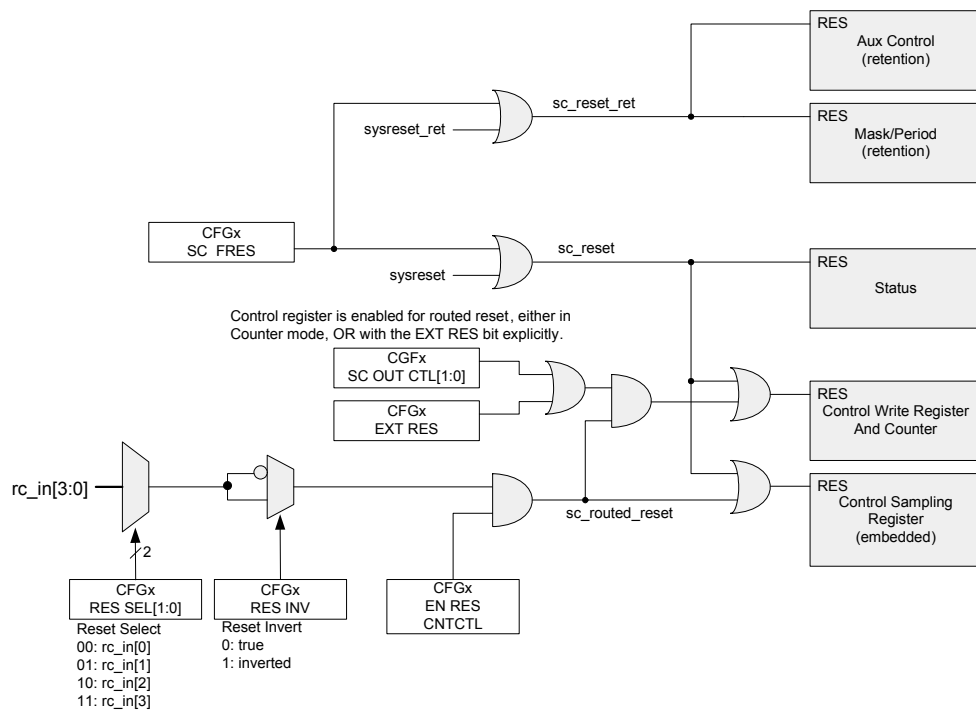
图 15-42. 兼容的数据路径复位结构



兼容的状态和控制复位控制

图 15-43 显示了兼容的状态和控制模块复位。掩码 / 周期和辅助控制寄存器都是保持寄存器。

图 15-43. 兼容的状态和控制复位控制



备用复位方案

表 15-23 显示了兼容复位方案和备用复位方案间的差异。

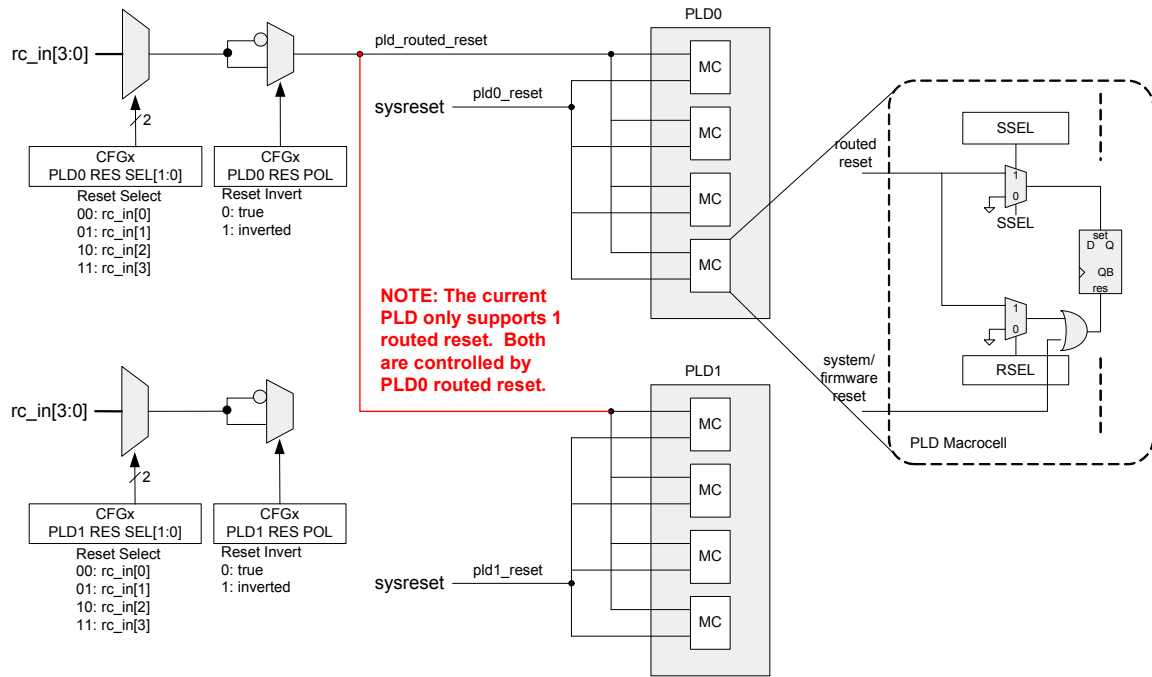
表 15-23. 复位方案

特性	兼容	备用
精度	UDB 中的所有模块共用了一个路由复位	每个 UDB 组件模块都可以选择一个单独的复位
状态寄存器	没有路由复位功能	可以使用已选定的 SC 路由复位
数据路径	没有路由复位功能	可选的，可以使用已选定的 DP 路由复位

备用的 PLD 复位控制

图 15-44 显示了备用的 PLD 复位系统。虽然每个 PLD 都有自己的用于单独复位的规定，但 PLD 模块不支持该规定。因此，在备用复位方案中，PLD0 复位控制设置适用于两个 PLD。

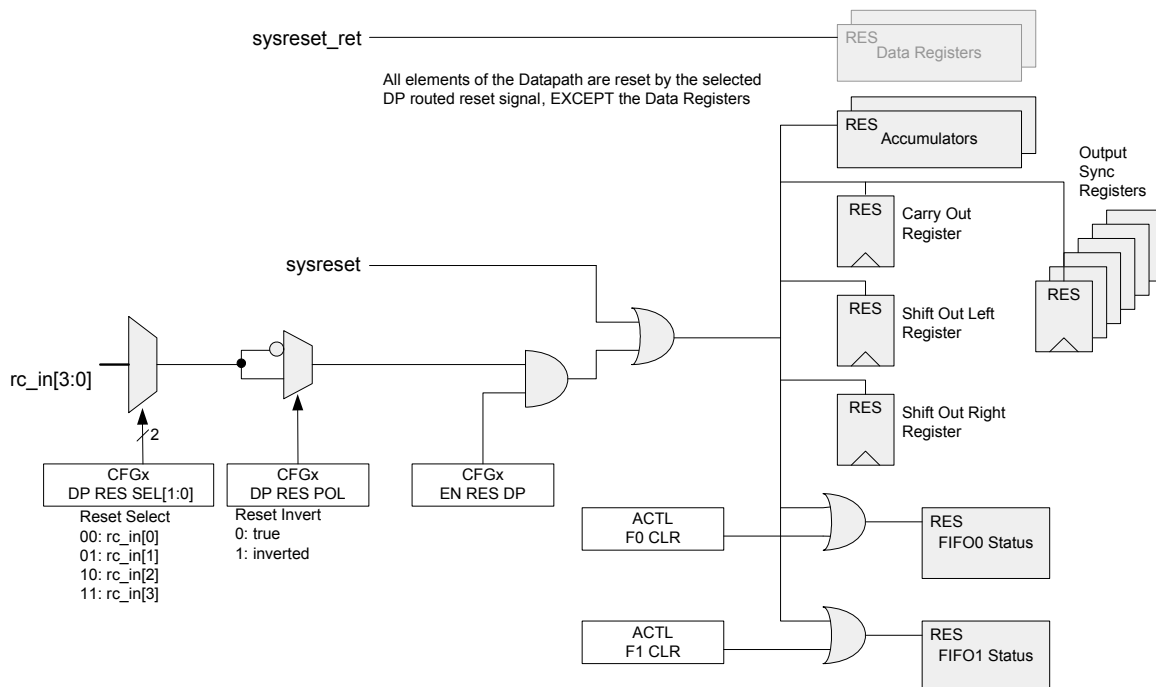
图 15-44. 备用的 PLD 复位结构



备用的数据路径复位控制

图 15-45 显示的是备用的数据路径复位系统。数据路径的路由复位适用于所有数据路径状态，作为保持寄存器实现的数据寄存器除外。

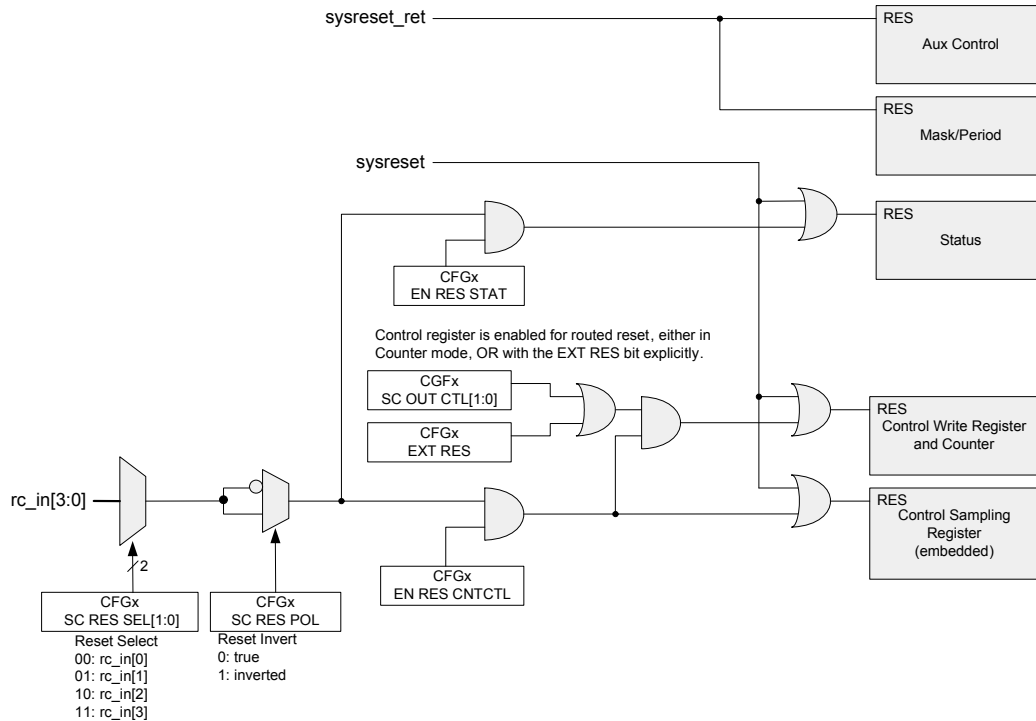
图 15-45. 备用的数据路径复位结构



备用的状态和控制复位控制

图 15-46 显示的是备用的状态和控制模块复位。掩码 / 周期和辅助控制寄存器都是保持寄存器。

图 15-46. 备用状态和控制复位控制



15.2.4.3 UDB POR 初始化

寄存器和状态初始化

表 15-24. UDB POR 状态初始化

状态元素	状态元素	POR 状态
配置锁存	CFG 0 – 31	0
Ax、Dx、CTL、 ACTL、MASK	累加器、数据寄存器、辅助 控制寄存器、掩码寄存器	0
ST、宏单元	只读状态和宏单元寄存器	0
DP CFG RAM 和 Fx (FIFO)	数据路径配置 RAM 和 FIFO RAM	未知
PLD RAM	PLD 配置 RAM	未知

路由初始化

上电复位（POR）时，输入和输出的路由状态如下：

- 从 UDB 驱动到路由矩阵中的所有输出均保持为 ‘0’。
- 从路由驱动到 UDB 输入内的所有输出都为 ‘0’。

这样可以避免路由中驱动状态的冲突，而且初始配置会按独立序列依次发生。

15.2.5 UDB 寻址

可以通过多个地址空间对 UDB 进行访问，例如，工作寄存器（A0、A1、D0、D1、FIFO 等）和配置寄存器的 8 位、16 位和 32 位访问。

- 8 位工作寄存器 — 通过该地址空间可以访问 UDB 中的单个工作寄存器。
- 16 位连续工作寄存器 — 通过该地址空间可以访问两个连续 UDB 中相同的工作寄存器，例如 UDB n 的 D0 和 UDB n+1 的 D0。
- 已配对的 16 位工作寄存器 — 通过该地址空间可以访问同一个 UDB 中的两个工作寄存器，如 A0 和 A1。
- 32 位工作寄存器 — 通过该地址空间可以访问四个 UDB 中相同的工作寄存器，如 A1。
- 8 位、16 位或 32 位配置寄存器 — 通过该地址空间可以访问单个 UDB 中的配置寄存器。

15.2.6 系统总线访问一致性

UDB 寄存器具有两种访问模式：

- 系统总线访问模式，在该模式下 CPU 对 UDB 寄存器进行读或写操作。
- UDB 内部访问模式，在该模式下 UDB 函数会更新或使用寄存器的内容。

15.2.6.1 同时进行系统总线访问

表 15-25 列出了可能同时访问的事件和所需的行为：

表 15-25. 同时进行的系统总线访问

寄存器	UDB 写入 总线写入	UDB 写入 总线读取	UDB 读取 总线写入	UDB 读取 总线读取
Ax	未定义结果	不允许直接进行 ^{a、b}	UDB 读取先前的数值	当前值通过这两个操作读取
Dx				
Fx	不受支持（UDB 和总线必须为相反访问）	如果使用 FIFO 状态标志，在同一个位置上不能同时进行读 / 写操作		不受支持（UDB 和总线必须为相反访问）
ST	NA，总线不进行写操作	总线读取先前的数值	NA，UDB 不进行读操作	
CTL	NA，UDB 不进行写操作		UDB 读取先前的数值	当前值通过这两个操作读取
CNT	未定义结果	不允许直接进行 ^c		
ACTL	NA，UDB 不进行写操作			
MASK				
PER				
宏单元（RO）	NA，总线不进行写操作	不允许直接进行 ^d	NA，总线不进行写操作	

a. 通过使用 FIFO 的软件捕获特性，可以安全读取 Ax 寄存器。

b. 只有 FIFO 才能对 Dx 寄存器进行动态写入。编程该模式时，不允许直接读取 Dx 寄存器。

c. CNT 寄存器只在被禁用时，才能被安全读取。另外，通过将输出路由到 SC 寄存器（透传模式），也可以动态读取 CNT 值。

d. 通过将宏单元寄存器位路由到状态寄存器（透传模式），也可以进行安全读取操作。

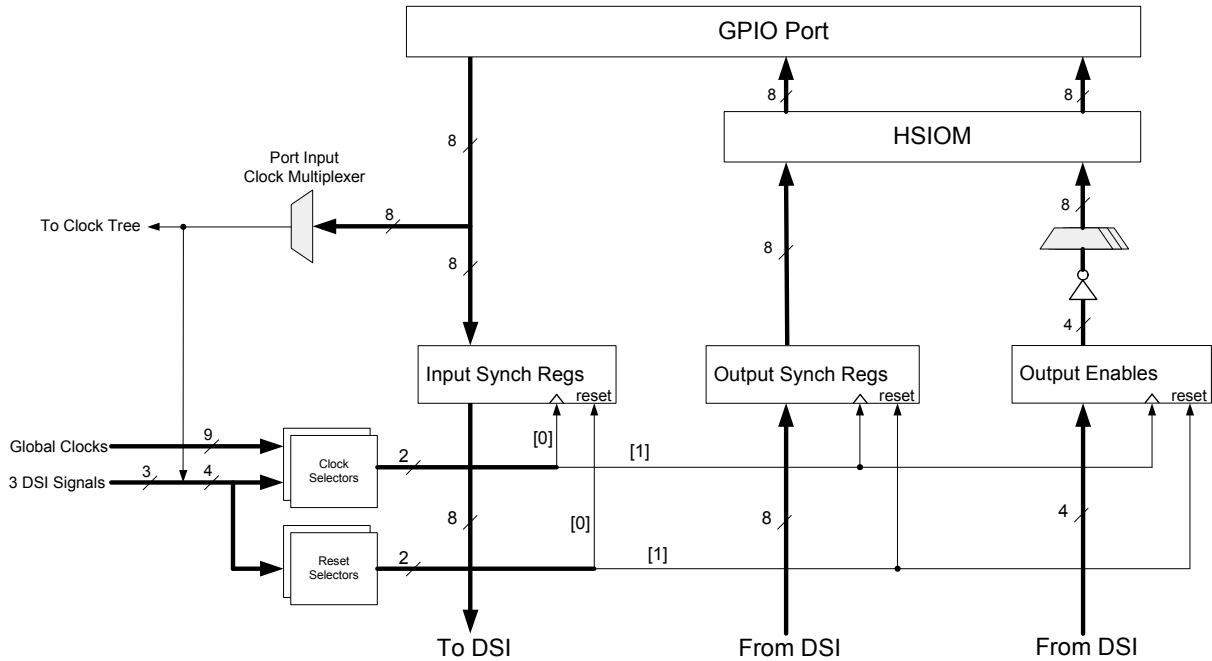
15.2.6.2 一致性累加器访问（原子读和写操作）

UDB 累加器是数据计算的主要目标。因此，在正常操作期间直接读取这些寄存器会得到未定义的结果，如表 15-25 所示。然而，原子读操作的内置支持（其格式为软件捕获）在各链接模块上实现。在该使用模型中，对最低有效累加器的读取操作会将所有链接模块中的数据传输到相应的 FIFO 内。通过编程方式可以对累加器进行原子写操作。可以对输入 FIFO 单独执行写操作，然后将最后一次写入 FIFO 内的状态信号路由到所有相关模块，同时将 FIFO 数据传输到 Dx 或 Ax 寄存器内。

15.3 端口适配器模块

端口适配器模块扩展了 UDB，以便通过高速 I/O 矩阵（HSIOM）连接到 GPIO，如第 72 页上的高速输入 / 输出矩阵中所介绍。HSIOM 对各寄存器进行放置，以便将 DSI 信号快速路由到 GPIO 输出和输出使能。HSIOM 还允许多个模块共用 GPIO，例如，端口数据寄存器和外设（如 I2C）。图 15-47 显示的是一个顶层视图。

图 15-47. 端口适配器框图



每个 8 位 GPIO 端口都有一个端口适配器（PA）。GPIO 输入数据具有 8 个输入，GPIO 输出数据具有 8 个输出，以及 8 个输出使能（OE）连接。PA 中的寄存器用于同步输入、输出和输出使能。

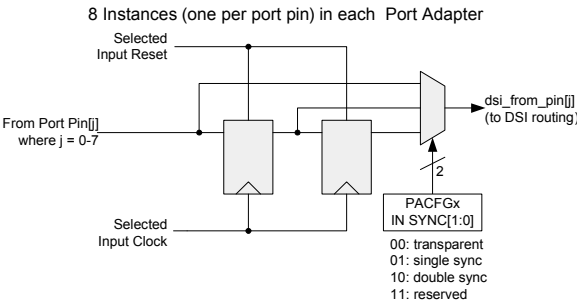
另一个特性是端口输入时钟复用器。通过该复用器可以选择作为时钟使用的一个端口输入。该时钟可以在 PA 中局部使用并路由至全局时钟（见第 77 页上的时钟系统章节）。

两个可编程的时钟选择器均可用，用以为输入和输出同步寄存器提供单独的时钟。OE 寄存器和输出寄存器使用同一个时钟。另外，与可编程的时钟选择器相同，两个可编程的复位选择器均可用。

15.3.1 PA 数据输入逻辑

图 15-48 显示的是数据输入逻辑的结构。各输入来自输入 / 输出端口上的每一个引脚。可以对该信号进行单同步或双同步，或旁路同步过程，以得到异步输入。对已选的端口输入时钟进行同步。该电路的输出连接到 DSI 路由。

图 15-48. GPIO 输入逻辑的详细内容



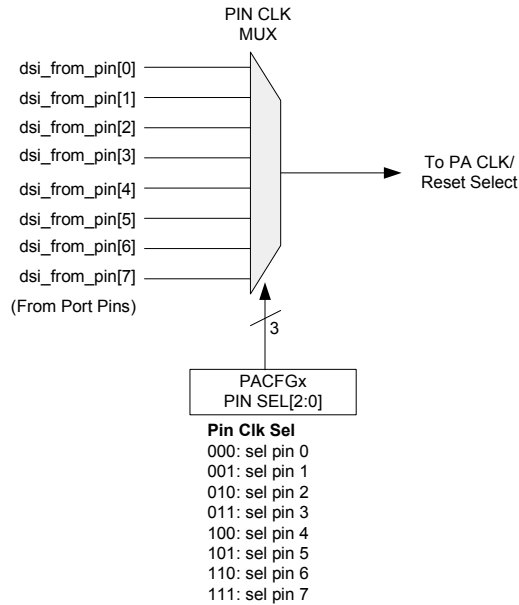
15.3.2 PA 端口引脚的时钟复用器逻辑

图 15-49 显示的是端口引脚复用器。每个端口具有八个数据输入信号，其中一个被作为时钟使用。该时钟被路由，以作为：

- 端口适配器中的可编程时钟
- UDB 时钟树源
- 端口适配器中的可编程复位
- 作为端口适配器中的时钟使能使用。

请注意，已选信号不能通过同步器传送，并与模块中的其他时钟域异步。将该信号用于已选功能时，必须十分慎重。

图 15-49. GPIO 引脚选择的详细内容

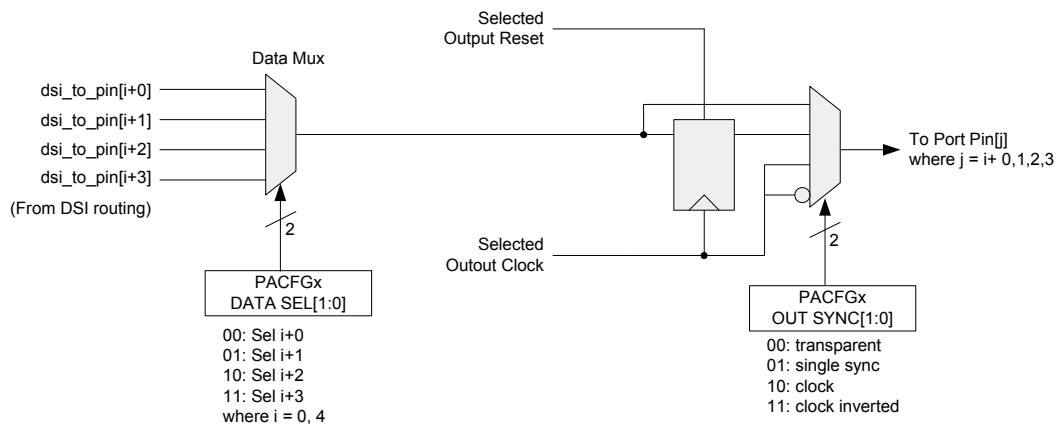


15.3.3 PA 数据输出逻辑

图 15-50 显示的是数据输出逻辑的结构。输出（通过 HSIOM）被发送给 I/O 端口上的各个引脚。可以对该信号进行单同步，或旁路同步过程，以得到异步输出。通过其他选项可以输出所选的时钟或输出时钟的反转版本。

图 15-50. GPIO 输出数据逻辑的详细内容

8 Instances (one per port pin) in each Port Adapter



15.3.4 PA 输出使能逻辑

图 15-51 显示的是输出使能（OE）逻辑。该电路和数据输出共用了相关的时钟和复位。该连接是唯一的，就是说四个 DSI 输出与 OE 相连，但这些输出被复用到 I/O 端口引脚上的四个 OE 连接，如图 15-52 所示。

图 15-51. GPIO 输出使能（OE）同步逻辑

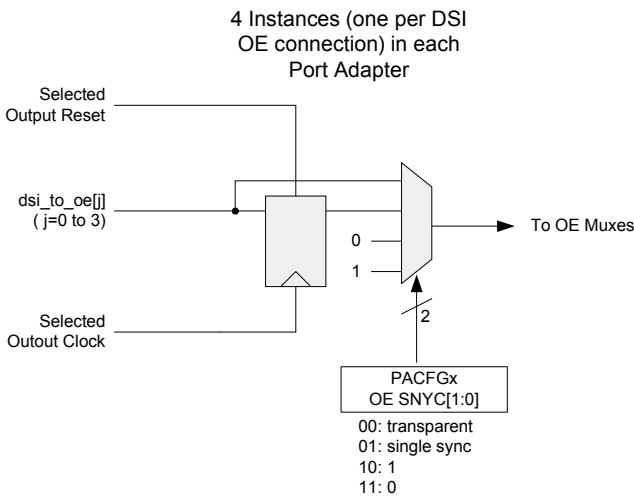
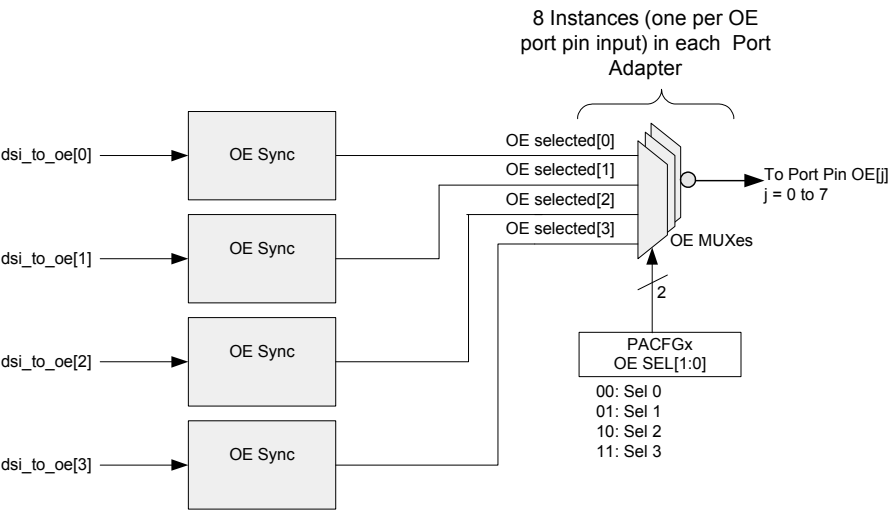


图 15-52. GPIO 输出使能（OE）复用器

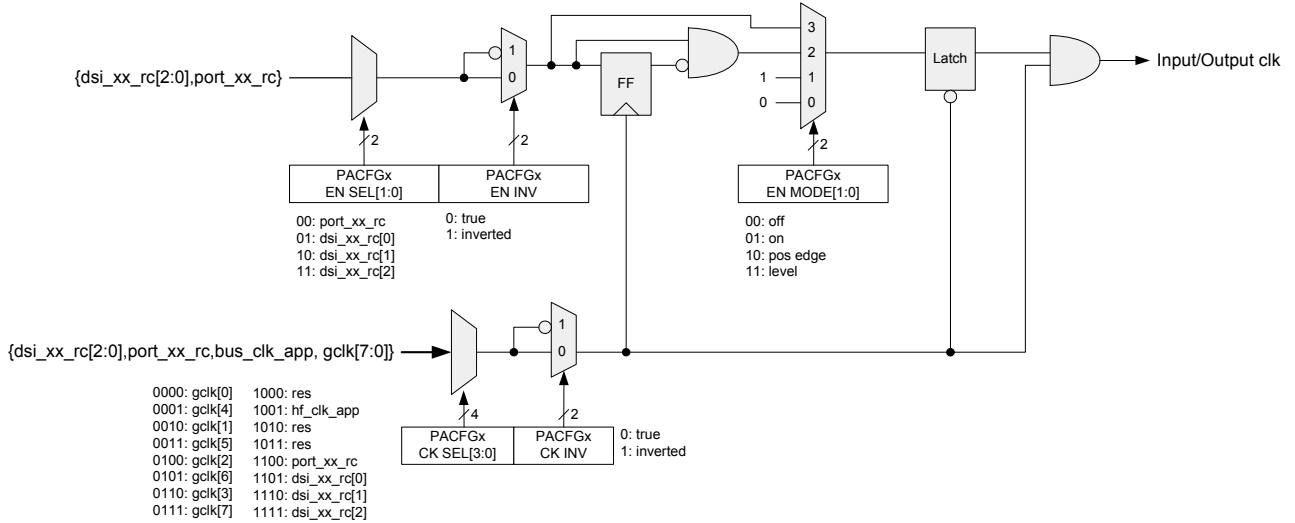


请注意，由于 OE 信号在各端口上为低电平有效，因此在 OE 同步逻辑和 OE 复用器之间的路径中存在一个附加的反转。

15.3.5 PA 时钟复用器

图 15-53 显示的是 PA 时钟复用器的结构。如上面所述，每个 PA 都有两个可编程的时钟选择器，用于为端口输入、输出和输出使能 (OE) 提供单独的时钟。

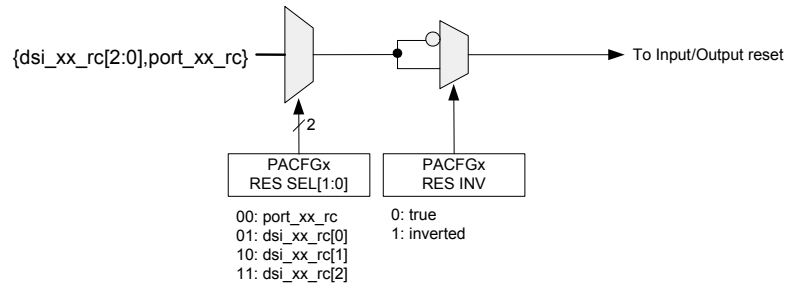
图 15-53. PA 时钟复用器的详细内容



15.3.6 PA 复位复用器

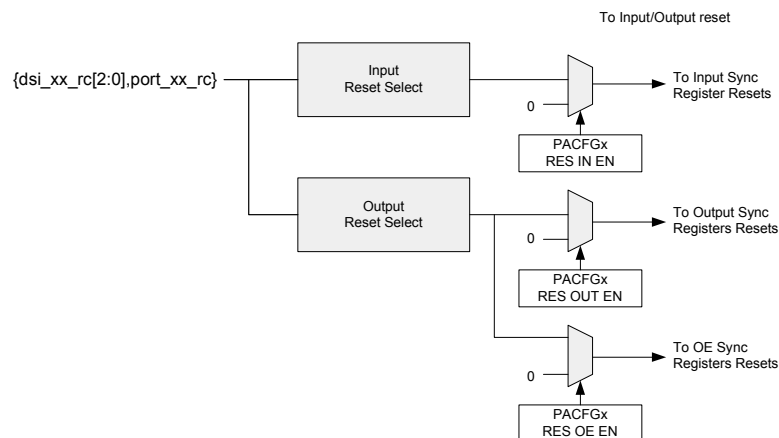
PA 复位复用器的结构如图 15-54 中所示。

图 15-54. PA 复位复用器的详细内容



如图 15-55 所示，复位选择逻辑被重复，一个用于输入，一个用于控制输出和输出使能。每个复位都有一个单独使能，适用于相应类别中的所有 8 位。

图 15-55. PA 复位系统

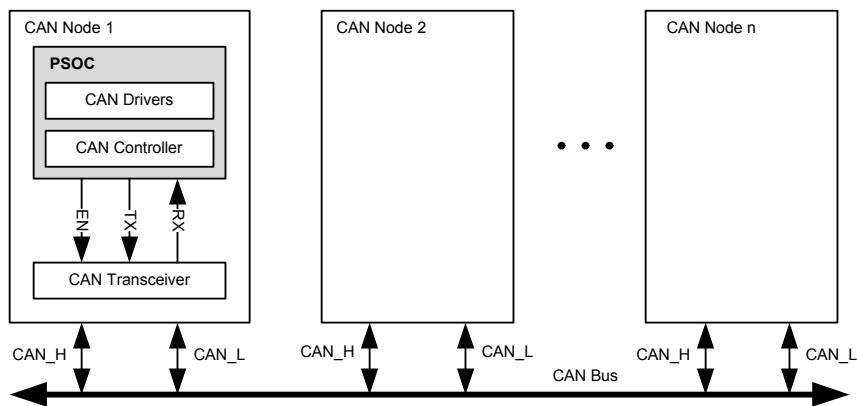


16. 控制器区域网络（CAN）



CAN 外设是功能齐全的控制区域网络（CAN），能够支持高达 1 Mbps 的通信波特率。PSoC 4200L 产品系列具有两个特定的 CAN 控制器模块，可以将它们路由到不同的引脚组上。这些 CAN 控制器包括两种与 ISO-11898 规范相兼容的类型，分别为：CAN2.0A 和 CAN2.0B。CAN 协议最初专门设计用于汽车级应用，侧重于高水平的故障检测和恢复。这样能够确保以较低的成本实现高度可靠的通信。由于在汽车级应用中取得了巨大成功，因此 CAN 被用作运动嵌入式控制应用（CANOpen）和工厂自动化应用（DeviceNet）的标准通信协议。通过 CAN 的特性，可以有效地实现更高级的协议，而不会影响微控制器 CPU 的性能。

图 16-1. CAN 总线系统实现



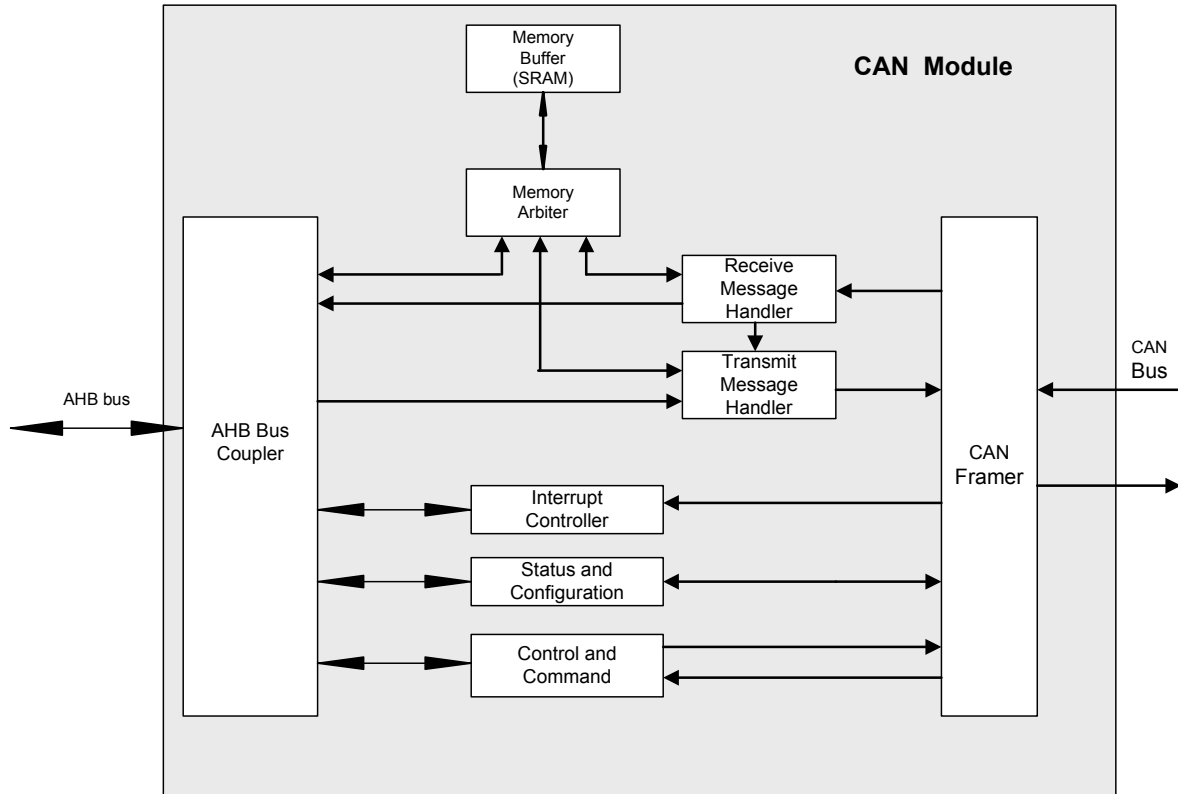
16.1 特性

- 与 CAN2.0A/B 协议规范相兼容：
 - 标准帧和扩展帧
 - 支持远程发送请求（RTR）
 - 支持高达 1 Mbps 的可编程比特率
- 接收路径：
 - 16 个接收信息缓冲区
 - 具有 16 个验收滤波器和 16 个验收屏蔽寄存器
 - 支持 DeviceNet 寻址功能
 - 可以选择链接多个接收缓冲区，从而组成一个硬件 FIFO
- 发送路径：
 - 八个发送信息缓冲区
 - 每个发送信息缓冲区的优先级都是可编程的
 - 支持一次性发送信息
- 可通过只听模式进行自动的波特率检测
- 可通过内部和外部环回模式进行模块级检测
- 总线上发生数据传输时，可从睡眠模式中唤醒器件
- 具有一个计数器，用于实现时间触发 CAN

16.2 框图

需要发送一个信息时，主机控制器需要将信息存储在发送信息的缓冲区中，并通知给发送信息的处理器，让它发送该信息。接收信息后，该信息将被存储在存储器缓冲区内，以便主机控制器在需要时处理它。执行发送和接收操作，主要是由状态寄存器和配置寄存器控制的。中断控制器处理来自 CAN 模块的各种中断。图 16-2 展示了该过程。

图 16-2. CAN 框图



16.3 CAN 中的信息帧

CAN 具有以下四种帧类型，用于控制信息发送和接收操作：

- 数据帧
- 远程帧
- 错误帧
- 过载帧

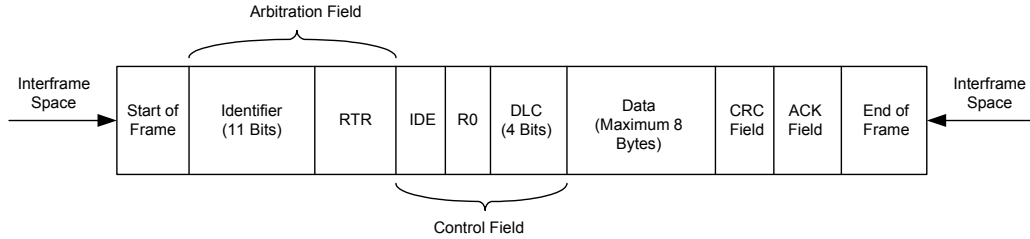
16.3.1 数据帧

数据帧主要用于在发送器和接收器间进行数据传输。CAN 主要支持两种数据帧，分别为：标准的数据帧和扩展的数据帧。对于一个 CAN 帧，数值 ‘0’ 表示显性位，数值 ‘1’ 表示隐性位。

16.3.1.1 标准的数据帧

图 16-3 显示的是 CAN 的标准数据帧。

图 16-3. 标准的数据帧



帧开始：数据帧的开头部分是由帧位的开头表示的。它是一个单显性位。

标识符：对于一个标准的 CAN 数据帧，标识符长度为 11 位。它主要用于筛选接收器端的数据。

远程发送请求位 (RTR)：将 RTR 位的值设置为 ‘0’（显性）时可采用数据帧；设置为 ‘1’（隐性）时可采用远程帧。其中，标识符和 RTR 位被称为仲裁字段。

扩展的标识位 (IDE)：采用标准的数据帧时，该位的数值必须为 ‘0’（显性）；采用扩展的 CAN 数据帧时，该值必须为 ‘1’（隐性）。

R0：保留位。

数据长度代码 (DLC)：这四个位表示数据字段中的数据字节数。IDE、R0 和 DLC 位一起组成控制字段。

数据字段：该字段包含了信息数据。它可以具有多种长度，最长为 8 位。

循环冗余校验 (CRC)：帧校验是通过循环冗余校验 (CRC) 的方式进行的。该字段包含了一个 15 位的 CRC 代码，后面是一个 CRC 分隔符。

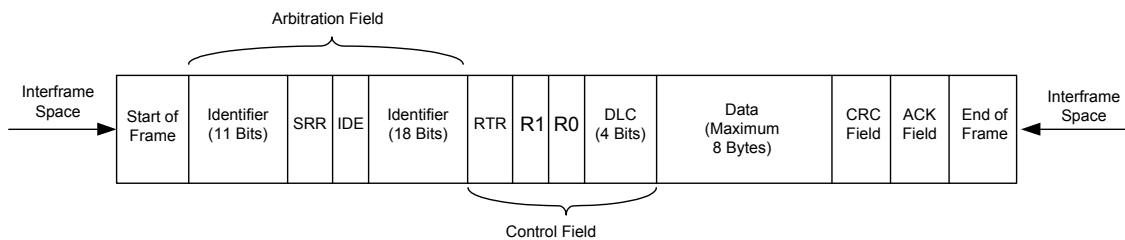
确认位字段 (ACK)：该 ACK 字段长度为两位，并默认为隐性。当接收器正确接收到一条信息时，它会使用一个显性位覆盖 ACK 字段。

帧结束：每一帧的结尾部分是帧结束字段，它包含七个隐性位。

16.3.1.2 扩展的数据帧

图 16-4 显示的是扩展 CAN 帧的格式。扩展 CAN 具有一个 29 位的标识符。它被分成一个 11 位的标识字段和一个 18 位的标识字段，在这两个字段之间存在一个替代远程请求 (SRR) 位和一个 IDE 位。采用标准的数据帧时，SRR 位的位置与 RTR 位的位置一样，并且为隐性的。IDE 位用于设置扩展帧。与标准的数据帧相比，扩展的数据帧中的控制字段具有额外的保留位 ‘R1’。

图 16-4. 扩展的数据帧



16.3.2 远程帧

CAN 总线允许一个目的节点通过向所使用的数据源发送一个远程帧来提取数据。数据帧与远程帧间的区别是：可以将远程帧中的 RTR 位作为隐性位发送，并且远程帧中没有数据字段。

对于扩展的远程帧，同样可以将 SRR 位作为隐性位进行发送。

帧间距：帧间距使数据帧和远程帧与前面的帧种类相互分开。

16.3.3 错误帧

当任意节点检测到总线错误时，它会生成一个错误帧。该错误帧包含一个错误标志和一个错误分隔符。错误标志分两种：主动错误标志和被动错误标志。

主动错误标志：当主动错误节点检测到某个错误时，它会发出六个显性位，作为主动错误标志。因此，错误标志的格式违反了位填充规格。这样会强制所有其他节点发出错误标志，从而引起一系列 6~12 个显性位被传送到总线上。

被动错误标志：一个被动错误标志包含六个隐性位。当被动错误节点检测到某个错误时，它会发出被动错误标志。被动错误标志不会影响任何其他节点，并且只有传输的节点检测到某个总线错误时才会检测到被动错误。

错误分隔符：错误分隔符包含八个隐性位。发送完某个错误标志后，各端会发出‘隐性’位，并监控总线状态，直到检测到一个‘隐性’位为止。然后，它会发送其余七个‘隐性’位。

16.3.4 过载帧

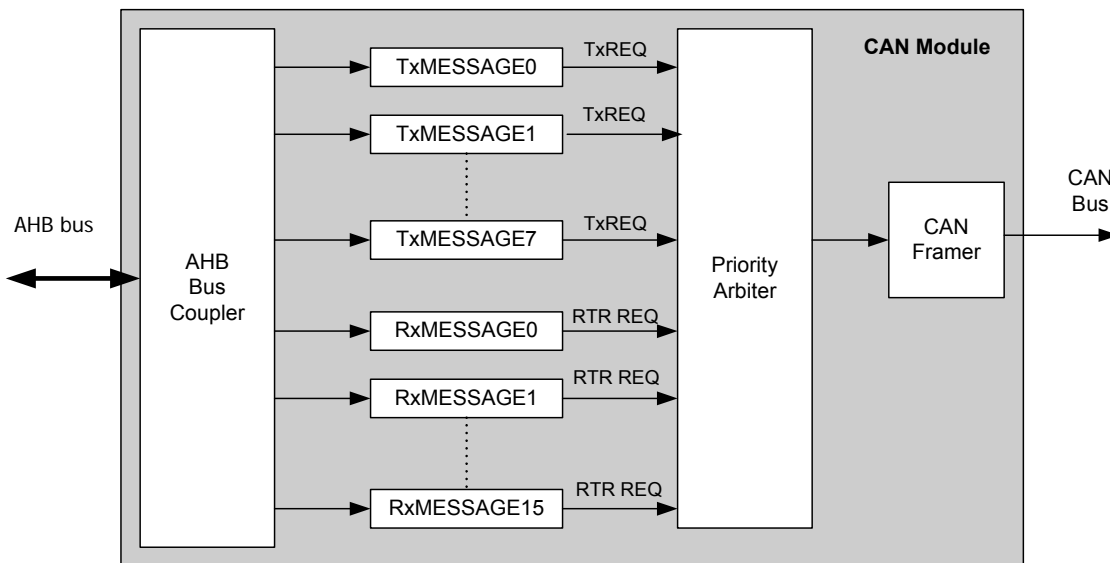
过载帧 (OF) 包括一个过载标志和一个过载分隔符。PSoC 4200L 器件中的 CAN 控制器支持可反应的 (对错误条件做出反应) 过载帧，在发生下述情况时，它被激活：

- 在停顿期间的前两位中检测到一个显性位。
- 在 OF 最后一位时间内使用接收器检测到一个显性位
- 在错误分隔符或过载分隔符的最后一位时间内使用任意节点检测到一个显性位

16.4 在 CAN 中发送信息

CAN 模块支持八个缓冲区，用于存储要发送的信息。内部优先级的仲裁器会根据已选的仲裁方案来选择相应信息。仲裁方案可以是一个轮循方案，也可以是固定优先级方案。在发送一条信息后或发生信息仲裁失败时，优先级仲裁器将重新评估下一条信息的优先级。接收信息的缓冲区还可以发送远程发送请求，该内容会在本章节后面部分进行说明。

图 16-5. 发送 (Tx) 框图



16.4.1 信息仲裁

优先级仲裁器支持轮循仲裁方案和固定优先级的仲裁方案。可以使用配置寄存器选择仲裁模式。

轮循：在一个轮循方案中，会先选中缓冲区 0，然后选中缓冲区 1，并逐渐选中缓冲区 7；再重新从缓冲区 0 开始，这样会构成一个循环。只有设置了特定缓冲区的 TX_REQ 位时，才会选中该缓冲区。该方案保证了所有缓冲区具有同等的信息发送概率。

固定的优先级：缓冲区 0 具有最高的优先级。将缓冲区 0 指定为存储关键信息的缓冲区，从而保证这些信息是最先被发送的。通过使用配置寄存器 (CAN_CONFIG[12]) 中的 CFG_ARBITER 位，可选中优先级仲裁功能。

注意：处理 TxMessage 缓冲区之前，先接收 RTR 信息请求。

16.4.2 信息发送过程

图 16-6 显示的是与发送信息相关的寄存器。

图 16-6. 发送 (Tx) 信息寄存器

REGISTERS											
COMMAND REGISTER (CAN_Txn_CMD)	Reserved [31:24]	WPN2 [23]	Reserved 1 [22]	RTR [21]	IDE [20]	DLC [19:16]	Reserved [15:4]	WPN1 [3]	Tx INT ENBL [2]	Tx ABORT [1]	Tx REQ [0]
IDENTIFIER (CAN_Txn_ID)	ID [31:3]									Reserved [2:0]	
DATA REGISTER High (CAN_Txn_DH)	D0 [63:56]			D1 [55:48]			D2 [47:40]			D3 [39:32]	
DATA REGISTER Low (CAN_Txn_DL)	D4 [31:24]			D5 [23:16]			D6 [15:8]			D7 [7:0]	

$n = 0, 1, \dots, 7$

发送标准数据帧的过程包括以下主要步骤：

1. 将信息写入到一个空白的发送信息存储缓冲区内。该空白的缓冲区是由 **TX_REQ** 位指定的，该位的值为 0。
 - a. 对于标准的数据帧，需要向 **RTR** 和 **IDE** 位写入数值 ‘0’（显性）。
 - b. 准确编写各个 **DLC** 位，以指定将发送的数据字节数量。数据字节数量最大限定为 8。每个字节中带有先被发送的 **MSB**（最高有效位）的各个数据字节分别被写入到 **D0**、**D1**...**D7** 等寄存器内。
 - c. 11 位长的信息标识符被写入到 **ID[31:21]** 位字段中。
2. 选择一个合适的优先级仲裁方案。内部信息优先级的仲裁器会根据已选的仲裁方案来选择相应的信息。
3. 通过将相应的 **TX_REQ** 位设置为数字 ‘1’ 来实现请求发送。
4. 信息发送请求处于待处理状态时，**TX_REQ** 位仍保持为设置状态。在设置 **TX_REQ** 位期间，不需要修改信息缓冲区中的内容。

信息发送完成后，将清除 **TX_REQ** 位，并激活中断状态寄存器 (**CAN_INT_STATUS**) 中的 **TX_MSG** 中断状态位 (**[CAN_INT_STATUS[11]]**)。只有将 **TxINT ENBL** (**[CAN_TX[n]_CONTROL[2]]**) 位设置为 ‘1’ 时，中断状态位才被激活。

16.4.3 信息中止

通过设置 **CAN_TX[n]_CONTROL** 寄存器中的 **TX_ABORT** 位 (**[CAN_TX[n]_CONTROL[1]]**)，可以中止发送信息。信息被中止时，硬件将自动清除该位。

注意：

- **CAN** 缓冲区寄存器 (**CAN_BUFFER_STATUS**) 用于读取是否存在待处理的发送请求。
- 如果写保护位 **wpn2** (**[CAN_TX[n]_CONTROL[23]]**) 为 ‘0’，则不能修改指令寄存器中的位 **[21:16]**，因为它们已经受保护，并且在回读时会提供未定义的数值。
- 如果写保护位 **wpn1** (**[CAN_TX[n]_CONTROL[3]]**) 为 ‘0’，则不能修改命令寄存器中的位 **[2]**。在回读时，该位会提供一个数值 ‘0’。
- 使用 **WPN** 标志 (**wpn1** 和 **wpn2**) 时，只要设置 **TX_REQ** 位（而无需关注 **RTR**、**IDE**、**DLC** 和 **TxINTENBL** 等特殊标志）便能重新发送同一条信息。

16.4.4 一次性发送

在使用一次性发送模式的系统中，存在由仲裁失败或总线错误引起的 **CAN** 信息重新发送必须被阻止。这对时间触发（即所有信息均在一个固定的时间内被发送）的 **CAN** 系统非常有用。

同时激活 **CAN_TX.CONTROL.TX_REQ** 位和 **TX_ABORT** 位，可设置一次性发送请求。信息发送成功后，这两位将被清除。

如果在发送过程中发生仲裁失败或总线错误，那么将清除 **TX_REQ** bit 位，但会保持 **TX_ABORT** 位为激活状态。同时，也会激活一次性发送故障 (**sst_failure**) 中断。

16.4.5 发送扩展的数据帧

与标准的数据帧相比，必须修改某些寄存器设置才能发送一个扩展的数据帧。修改内容具体如下：

- 对于扩展的数据帧，向 **IDE** 位内写入数值 ‘1’（隐性）。
- 将信息标识符写入到 **ID[31:3]** 位字段中。

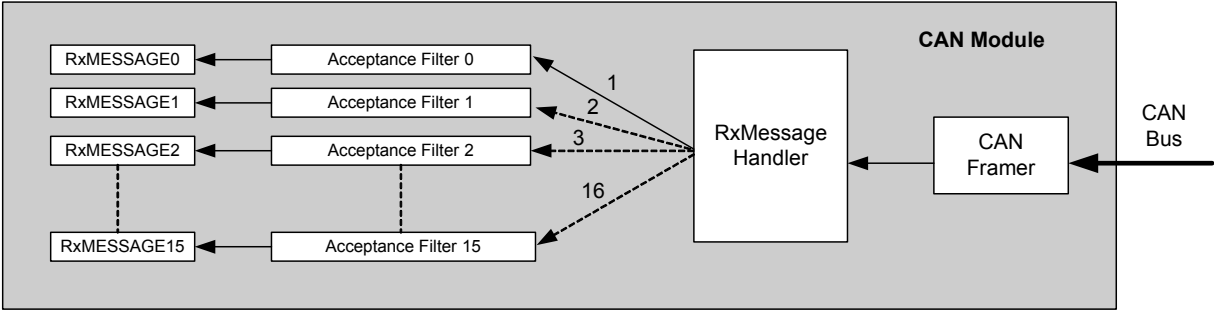
16.5 CAN 内接收信息

CAN 模块有 16 个接收信息缓冲区，如图 16-7 所示。每个信息缓冲区都有一个专用的验收滤波器。CAN 信息通过 CAN 帧接收。然后，将接收的信息同时和所有验收滤波器进行比较；被接受的信息将被存储在相应的接收信息缓冲区内。设置信

息缓冲区内的信息有效（MSG_AV）位，以表示可以接收新的信息。只有清除 MSG_AV 位，从而允许接收另一个信息时，信息接收才得到确认。

验收滤波器是通过验收屏蔽寄存器（AMR）和验收代码寄存器进行配置。

图 16-7. 接收（Rx）框图



16.5.1 信息接收过程

图 16-8 显示的是与所接收的信息相关的寄存器。

图 16-8. 接收（Rx）信息寄存器

REGISTERS															
COMMAND REGISTER (CAN_RXn_CMD)	Reserved [31:24]	WPNH [23]	Reserved ¹ [22]	RTR [21]	IDE [20]	DLC [19:16]	Reserved [15:8]	WPNL [7]	LINK FLAG [6]	RX INT ENBL [5]	RTR REPLY [4]	BUFFER EN [3]	RTR ABORT [2]	RTR REPLY PNDG [1]	MSG AV [0]
IDENTIFIER (CAN_RXn_ID)	ID [31:3]														Reserved [2:0]
DATA REGISTER High (CAN_RXn_DH)	D0 [63:56]				D1 [55:48]				D2 [47:40]				D3 [39:32]		
DATA REGISTER Low (CAN_RXn_DL)	D4 [31:24]				D5 [23:16]				D6 [15:8]				D7 [7:0]		

n = 0,1,...,15

接收信息过程包括下面主要步骤：

1. 接收新的信息后，RxMessageHandler 硬件（如图 16-7 所示）将查找所有接收缓冲区（从 RxMessage0 缓冲区起，直到它寻找到一个有效的缓冲区为止）。有效的缓冲区定义如下：
 - a. BUFFER EN = ‘1’（CAN_RX[n]_CONTROL[3]）表示接收缓冲区被使能。
 - b. 接收缓冲区内的验收滤波器与输入的信息相互匹配。
2. 如果 RxMessageHandler 发现某个有效缓冲区为空，则信息被存储，并且该存储器的 MSG_AV 位被设置为 ‘1’。
3. 如果 RX INT ENBL 位被设置，则中断控制器的 RX_MSG 标志（CAN_INT_STATUS[12]）将被激活。
4. 如果接收缓冲区中包含了一个信息（MSG_AV = ‘1’），并且 LINK_FLAG 位未被设置，那么 RX_MSG_LOSS 中断标志（CAN_INT_STATUS[10]）将被激活。此时，新收到的信息将被丢弃。

注意：CAN 缓冲区寄存器（CAN_BUFFER_STATUS）确定是否有接收信息缓冲区可用。

16.5.2 验收滤波器

每个接收缓冲区都有自己的验收滤波器，用于筛选输入的信息。验收滤波器是通过验收屏蔽寄存器（AMR）和验收代码寄存器（ACR）进行配置。AMR 规定输入信息中的哪些位必须与相应 ACR 位相匹配，从而进行验收信息。

AMR 的值为 ‘0’：针对 ACR 位检查输入位。当输入位与相应的 ACR 位不相互匹配时，便不接收信息。

AMR 的值为 ‘1’：输入位无需关注。

包含下面信息字段：

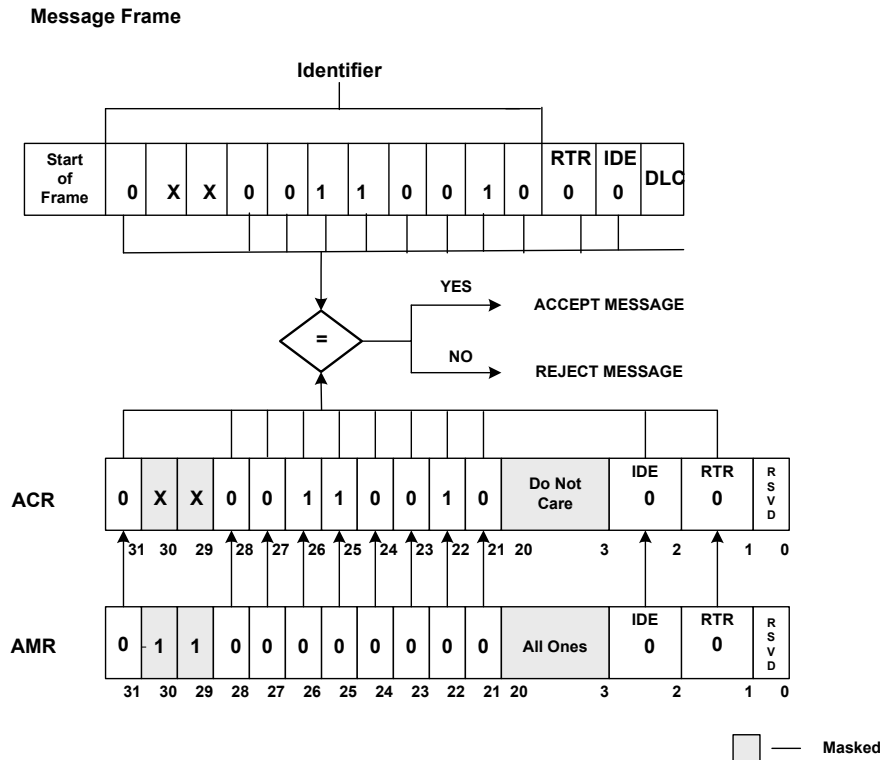
- 标识符
 - 扩展的标识位（IDE）
 - 远程发送请求（RTR）
 - 数据字节 1（D0）和数据字节 2（D1）（DATA[63:48]）¹
- 对于标准的 CAN 信息，当 IDE = 0 时，11 位的标识符便为 AMR 和 ACR 的位 [31:21]。

1. 有助于 第 160 页上的 DeviceNet 筛选中所介绍的 DeviceNet 筛选。

16.5.2.1 示例

图 16-9 显示的是一个示例信息和接收该信息所需的验收滤波器设置。

图 16-9. 验收滤波器



如图 16-9 所示，阴影区为屏蔽位。AMR 寄存器中的某一位被设置为 ‘1’ 时，将不会针对所接收到的信息帧检查 ACR 寄存器中相应的位。正如示例中所示，位 30、位 29 和位 [3:20] 均被设置为 ‘1’，并被屏蔽。由于 AMR 寄存器中的其他位被编写为数值 ‘0’，因此 ACR 寄存器中的相应位将与各个信息位进行比较，如图 16-9 所示。如果 ACR 寄存器中某相应位同信息位相匹配，那么信息将被存储在接收信息缓冲区内。否则，输入信息将被拒绝。

AMR 设置:

ID[31]、ID[28:21] = 0

ID[30]、ID[29] = 1

ID[20:3] = 1 (其所有位均为 1)

IDE = 0

RTR = 0

ACR 设置:

ID[31:21] = 182h

ID[20:3] = 无需关注

IDE = 0

RTR = 0

16.5.3 DeviceNet 筛选

对于 CAN 的某些高级协议（如 DeviceNet），数据的前两个字节将涵盖了与协议相关的额外信息。验收滤波器为这两个字节进行其他的筛选，从而更加有效地实现该协议。将输入信息中前两个字节的的数据位与 ACR_DATA 寄存器的位（CAN_RX[n]_ACR_DATA）进行比较，这些位是由 AMR_DATA 寄存器中的位（CAN_RX[n]_AMR_DATA）指定的。图 16-10 对 DeviceNet 筛选进行说明。

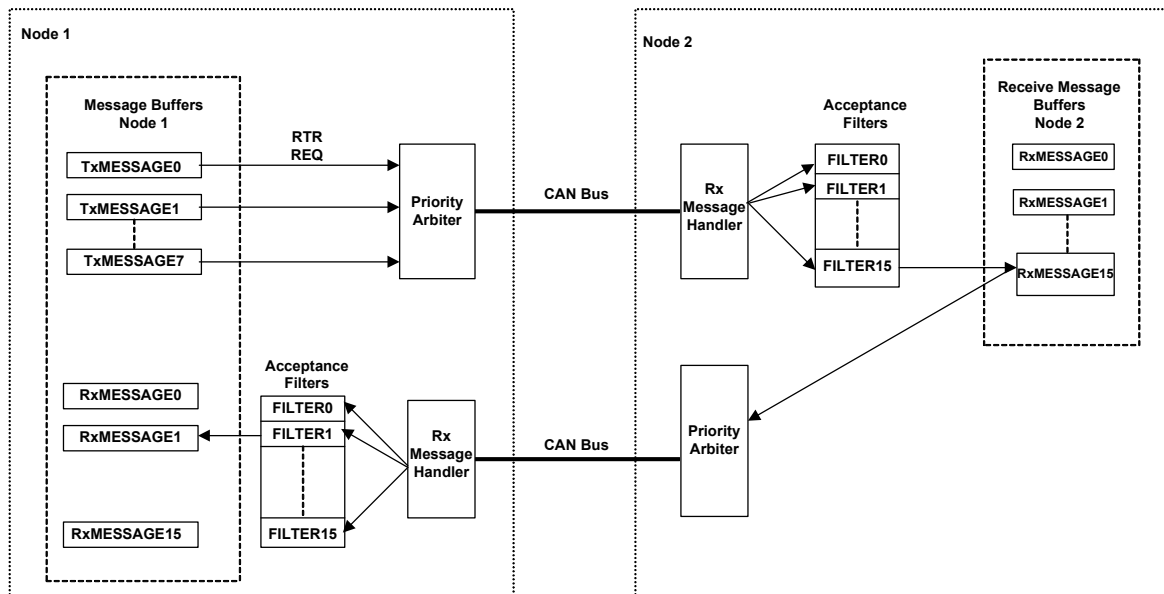
16.6 远程帧

远程帧用于初始化两个节点间的发送，其中作为接收器的节点将发送远程帧。远程帧可以采用标准格式或者扩展格式。远程帧与数据帧的区别在于：它的 RTR 位始终为 ‘1’，其数据字段不存在，并且独立于 DLC 字段的值。图 16-11 显示的是远程发送请求的序列。

如图 16-11 所示：

- 节点 1 的信息缓冲区 0 将一个远程帧发送到 CAN 总线上。
- 节点 2 的 RxMessageHandler 会接收 RTR 请求，并将其发送给验收滤波器。
- 如果接收信息缓冲区 15 中的验收滤波器设置与该信息的设置相互匹配，则该信息将被移植到接收信息缓冲区 15 中。
- 如果使能了 RTR 自动回复功能，那么接收信息缓冲区 15 将按照它所接收到的标识符发送该信息（无需 CPU 干预）。
- 成功发送 RTR 自动回复指令时，将置位 CAN_RX.CONTROL 寄存器中的 MSG_AV_RTR_SENT 位和中断状态寄存器中的 RTR_MSG 位。
- 节点 1 内接收信息缓冲区 1 的验收滤波器与发送信息的节点 1 具有相同的标识符。因此，RTR 信息将被存储在节点 1 的接收信息缓冲区 1 内。

图 16-11. 远程帧发送请求



16.6.1 通过请求节点发送远程帧

下面显示的是请求节点发送远程帧的过程（节点 1 如图 16-11 所示）。

1. 将信息写入到一个空白的发送缓冲区内。一个空白的缓冲区是由 TX_REQ = ‘0’ 表示（CAN_TX[n].CONTROL[0]）的。
2. 将 RTR 位（CAN_TX[n].CONTROL[21]）设置为 ‘1’。
3. 选择一个合适的优先级仲裁方案。
4. 将发送请求标志设置为初始发送。
5. 发送信息中所包含的标识符必须与接收信息的标识符一样。

16.6.2 接收一个远程帧

接收一个远程帧的过程如下所述：

1. 必须配置验收滤波器，以接收所需信息 ID。
2. 将 ‘RTR REPLY’ 位设置为 ‘1’，以使用 RTR 信息自动处理功能。
 - a. 如果使能了该功能，将自动以相同的标识符发送远程帧。
 - b. 否则，远程帧必须按照标准子程序（正如数据帧的情况）发送。
3. 设置请求节点（其接收 RTR 回复信息），以接收普通信息。请勿设置 RTR REPLY 位。

16.6.3 RTR 自动回复

CAN 模块支持 RTR 信息请求的自动回复功能。所有 16 个接收缓冲区都支持该特性。如果接收缓冲区接收了某个 RTR 信息，并且设置了 RTR REPLY FLAG 位，那么该缓冲区将使用其内容自动回复该信息。在接收到 RTR 信息请求时，会设置 RTR REPLY PNDG FLAG 位。发送该信息或禁用信息缓冲区时，将复位该位。想要中止一个待处理的 RTR 回复信息时，请使用 RTR ABORT 命令。

16.6.4 扩展格式的远程帧

扩展格式远程帧的发送和接收过程基本上与标准格式的相同，但在下面各特点上存在区别：

- IDE 位 (CAN_TX[n]_CONTROL [20]) 被设置为 ‘1’，从而使远程帧作为一个扩展的数据帧使用。
- 其标识符的长度为 29 位（标准的数据帧为 11 位长）。

16.7 时间触发的 CAN

时间触发的 CAN (TTCAN) 是 CAN 协议本身更高层次的协议层。TTCAN 对周期使用 CAN 信息的各种应用很有帮助。在 TTCAN 系统中，信息与网络中时间主设备的参考信息同步进行发送 / 接收。CAN 控制器具有的下述特性，使它符合扩展标准第一级 TTCAN 系统的实现：

- 能够进行一次性发送，如 16.4.4 一次性发送中所述。
- 具有一个内部定时器，用于对 TTCAN 信息进行定时。

注意：配置 TTCAN 接收缓冲区来接收远程帧时，应通过清除 CAN_RX.CONTROL.RTR_REPLY 位来禁用信息自动回复功能。

16.7.1 TTCAN 定时器

指的是 CAN 模块中的一个 16 位定时器子模块，可将其作为 TTCAN 系统的定时器使用。通过 CAN 控制寄存器的设置 (CNTL.TT_ENABLE)，可以使能该位。该定时器计数 CAN 位的额定时间，并在每次检测到 CAN 总线上的帧开始 (SOF) 时都捕获该定时器的计数值。与 TTCAN 定时器相关的寄存器分别为：

■ TTCAN_COUNTER

它是一个 16 位的局部定时器计数器寄存器 (TTCAN_COUNTER.LOCAL_TIME)。它会根据 TTCAN_TIMING 寄存器内的位时序设置进行计数额定位的时间。

■ TTCAN_COMPARE

它是局部计数器的比较值，用于生成时间事件。当 TTCAN_COUNTER.LOCAL_TIME 计数到 TTCAN_COMPARE 寄存器的值时，将触发一个 tt_compare 硬件事件。如果使能了中断寄存器 (INTR_CAN_SET) 内的 TT_COMPARE 位，那么中断状态寄存器 (INTR_CAN) 内的相应位也被设置。更多信息，请参见 16.9.4.2 TT_ENABLE = 1 时的

中断路由。

■ TTCAN_CAPTURE

当检测到 CAN 总线 (CAN Rx 输入) 上的一个 SOF 时，该寄存器将捕获 TTCAN_COUNTER 的值。这样还会触发 tt_capture 硬件事件；如果使能了中断寄存器 (INTR_CAN_SET) 内的 TT_CAPTURE 位，则中断状态寄存器 (INTR_CAN) 内的相应位也被设置。更多信息，请参见 16.9.4.2 TT_ENABLE = 1 时的中断路由。

参考帧的 SOF 可作为 TTCAN 系统的同步信号使用。

可通过使用 AMR/ACR 寄存器进行筛选参考信息的信息 ID。检测到参考信息时，将读取 TTCAN_CAPTURE 寄存器的值，用于同步参考信息的时间。

■ TTCAN_TIMING

该寄存器配置额定位时序，用于生成 TTCAN 计数器的时钟。该寄存器的配置情况必须与 CAN 配置寄存器的位时间设置相同。

注意：必须注意 TTCAN 系统在具体的应用情况中的设计。系统设计员需要决定如何使用现有的硬件资源来构成一个完整的 TTCAN 系统。

16.8 位时间配置

CAN 模块由单个时钟输入 SYSCLK 提供脉冲。本章节介绍了如何配置可编程的比特率分频器，从而获得所需比特率；此外还说明了比特率与 SYSCLK 的关联。

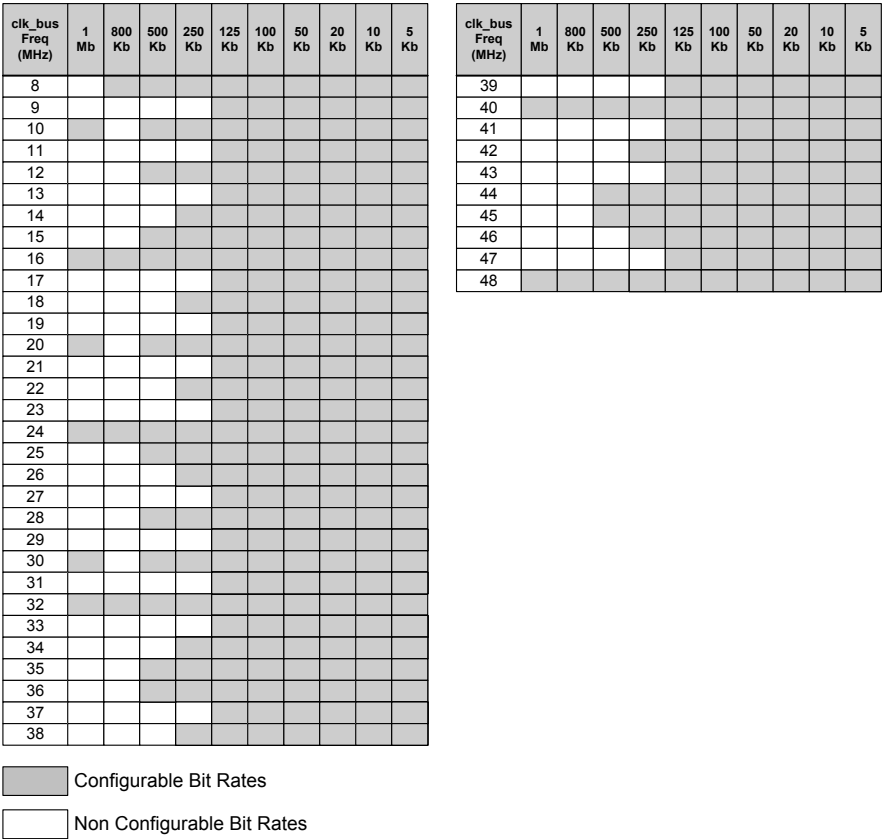
16.8.1 所允许的比特率和系统时钟 (SYSCLK)

在该行业中，几乎所有 CAN 总线的实现都使用下面 10 种比特率中的一种：

- 1 Mbps
- 800 Kbps
- 500 Kbps
- 250 Kbps
- 125 Kbps
- 100 Kbps
- 50 Kbps
- 20 Kbps
- 10 Kbps
- 5 Kbps

如果 SYSCLK 的频率为 8 MHz 的整数倍，则可配置所有这些比特率。为了支持 1 MHz 最大比特率，SYSCLK 的最小频率必须为 10 MHz。如果 SYSCLK 的频率为 10 MHz 或 10 MHz 的整数倍，则可配置所有比特率（800 Kbps 除外）。由于存在一些例外，因此，如果 SYSCLK 的频率不是 1000000 Hz 的整数倍，那么不能将比特率配置为所有 10 的倍数。为了确保比特率的有效性，SYSCLK 的精确度最少为 1.58%（对于 125 Kbps 和更慢的比特率），0.5% 或更高（对于速度超过 125 Kbps 的比特率）。要想满足 CAN 模块对时钟精确度的要求，需要将外部晶振（ECO）作为 SYSCLK 的时钟源使用，或将一个精确的外部时钟连接到器件上，并将它作为 SYSCLK 时钟源使用。图 16-12 中的列表提供了 10 倍数的比特率，其支持 48 MHz ~ 100 MHz 范围内的任意组件时钟频率。PSoC 4200L 器件的最大有效频率为 48 MHz。

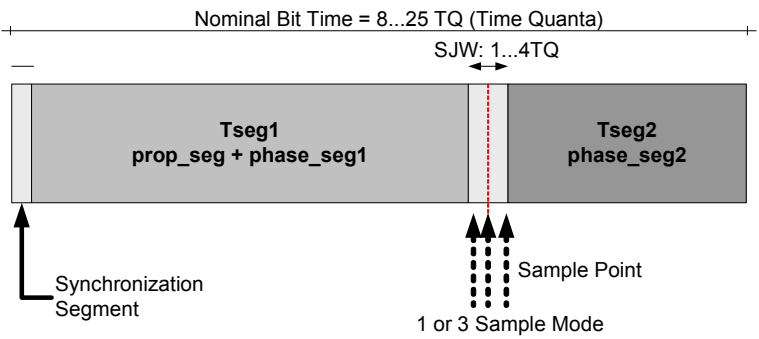
图 16-12. 比特率与 SYSCLK 的比较



16.8.2 设置比特率 TSEG1 和 TSEG2

比特率被定义为 CAN 总线上每秒所发送的位数。位时间便是比特率的倒数。位时间被分为三段，如图 16-13 所示。每段使用了时间的固定单元术语（起源于系统时钟（SYSCLK）的时间量子 — TQ）来表示。

图 16-13. 位时间



$$\text{BitTime} = (1 + \text{tseg1} + 1 + \text{tseg2} + 1)\text{TQ}$$

公式 1

$$\text{TQ} = \frac{\text{BRP} + 1}{\text{SYSCLK}}$$

公式 2

注意：比特率预分频器是一种寄存器，它对 SYSCLK 执行预分频，从而生成 CAN 模块的时钟。请参见图 16-14。

同步段。它是位时间中具有 1 TQ 长的第一段，并且主要用于实现同步。预计在该段中有一个信号下降沿。

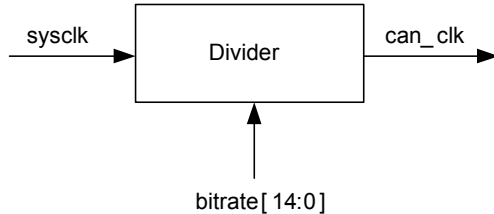
Tseg1、Tseg2。这两段用于补偿边沿相位移动的错误。其中，tseg1 还用于传播时间（包含该网络中的任意延迟）。可以增大或缩小这些段的长度，以补偿由边沿相位移动引起的错误，该操作称为重新同步处理。

采样点。它便是总线状态被读取并且该位被体现的位置。它位于 tseg1 的末端。

同步跳转宽度。进行重新同步处理后，可能 tseg1 被拉长，也可能 tseg2 被缩短。同步跳转宽度为该重新同步处理操作提供了一个限制条件。tseg2 的长度必须大于同步跳转宽度。

配置寄存器 CAN_CONFIG 用于设置比特率预分频器 (BRP)、tseg1、tseg2 以及同步跳转宽度。通过将系统时钟 (SYSCLK) 进行 (BRP+1) 分频来生成 CAN 外设时钟 (CAN_CLK)。有关“用于生成系统时钟的有效选项”的详细信息，请参考“时钟”一节。对于位时间内的 N 时间量子，必须将 CAN 外设时钟频率配置为 CAN 总线比特率 N 的整数倍。

图 16-14. 位时序框图



16.8.2.1 示例

下面显示的是 40 MHz 频率下获得 1 Mbps 速度的示例：速度为 1 MHz，并且位时间为 1 μs。

在位时间内选择一个最小值 8TQ (1TQ = 0.125 μs)。

$$\text{BRP} = ((\text{时间量子} * \text{SYSCLK}) - 1) = 4。$$

因此，将数值 ‘4’ 写入到配置寄存器中的 CFG_BITRATE 位内。

选择采样点为位时间的 60%，该值大约为 5TQ。因为采样点位于 tseg1 的末尾，因此 (tseg2+1) = 3TQ 或 tseg2 = 2TQ。

为了固定采样点的同步跳转宽度，请将 ‘1’ 写入 CFG_SJW 位内。

将数值 ‘2’ 写入到 cfg_tseg2 位内，以便将 tseg2 的值设置为 2TQ。

现在，使用下面公式计算 tseg1: tseg1 = ((位时间 - (1TQ + tseg2 + 1TQ)) - 1TQ，

结果是: tseg1 = 3TQ。

因此，将数值 ‘3’ 写入到配置寄存器中的 cfg_tseg1 位。

通过该程序，可以在使用图 16-12 中指定的时钟频率时达到标准的比特率。

设置 tseg1 和 tseg2 时，请参考下面条件：

- 不允许设置: tseg1 = 0 或 tseg1 = 1。
- 不允许设置 tseg2 = 0；只有在直接采样模式中才允许设置 tseg2 = 1。

注意 1：配置寄存器 (CAN_CONFIG) 中的 Sampling_mode 位指定在接收路径上使用的是一个采样点，还是使用三个具有多数决定功能的采样点。

注意 2：配置寄存器 (CAN_CONFIG) 中的 Edge_mode 位指定下降沿同步还是双边沿同步。

16.9 CAN 中的错误和中断

根据 CAN 协议规范，共有五种不同的错误类型。总线上的每个 CAN 节点尝试检测错误，当检测到错误时，它会发出一个错误帧。下面各节介绍不同的错误类型和错误处理过程。

16.9.1 错误类型

16.9.1.1 BIT 错误

CAN 单元在总线上发送一位也可以监控总线。当监控的位与发送的位不同时，将检测到一个 BIT 错误。一个例外的是，在传输仲裁字段的填充位期间或在 ACK Slot 时间内发送一个隐性位将不会检测到错误。系统不将发送器发送一个被动错误标志和检测一个显性位的事件视为一个 BIT 错误。

16.9.1.2 FORM 错误

CAN 信息格式存在错误时，将检测到 FORM 错误。信息帧中的固定格式字段（如：帧结束和帧间距）包含非法位。

16.9.1.3 ACKNOWLEDGE 错误

发送器在 ACK slot 时间内发送一个隐性位，会监控在这段时间内是否存在一个显性位。如果接收器正确收到信息，在 ACK slot 过程中写入一个显性位。因此，如果传输后发送器不找到 ACK slot 过程中的显性位，将检测到 ACKNOWLEDGE 错误。

16.9.1.4 CRC 错误

一个发送节点执行某些计算用以生成一个 CRC 代码，并将 CRC 代码传送到 CRC 字段内。另外，一个接收节点也会执行相同的计算来生成 CRC 代码。如果接收器生成的代码与发送的代码不匹配，那么将检测到一个 CRC 错误。

16.9.1.5 STUFF 错误

在一个信息字段（由位填充的信息编码）中共有六个连续相等的位值时，在第六个连续位的位时间内会检测到一个 STUFF 错误。

16.9.2 错误捕获寄存器

PSoC 4200L 器件中的 CAN 控制器拥有一个用于诊断 CAN 总线的专用错误捕获寄存器（ECR）。图 16-15 中显示的是错误捕获寄存器。

图 16-15. 错误捕获寄存器

Reserved [31:17]	FIELD [16:12]	BIT [11:6]	TX MODE [5]	RX MODE [4]	ERROR TYPE [3:1]	ECR STATUS [0]
------------------	---------------	------------	-------------	-------------	------------------	----------------

错误捕获寄存器具有两种模式：

自由运行模式：在该模式中，ECR 会捕获当前 CAN 帧内字段和位的位置。

错误捕获模式：在该模式中，当检测到一个 CAN 错误时，ECR 将对字段和位的位置进行采样。为了采样一个这样的事件，需要对 ECR 进行写访问来重新配置该位。重新配置后，ECR 只能捕获一个错误事件。为了继续捕获错误，要通过将数值 ‘1’ 写入到错误捕获寄存器内来重新设置 ECR。

注意：IDE 位被作为 ECR.FIELD 中的仲裁字段。

16.9.3 CAN 中的错误状态

CAN 具有三种错误状态：

- **主动错误。**一个主动错误节点可以参与正常的总线通信。当主动错误节点检测到一个错误时，它会发出一个主动错误标志。
- **被动错误。**一个被动错误节点参与总线通信。当被动错误节点检测到一个错误时，它会发出一个被动错误标志。发出被动错误标志后，要想继续进行传输，则被动错误节点要等待一段时间。在帧间距字段中，一个被动错误节点将发送其他八个隐性位。这一时期也被称为暂停数据传输，因为没有发生任何数据传输。
- **总线关闭。**处于该模式下的节点不参与任何总线通信，并不会对总线产生影响。

CAN 中的错误状态是通过错误状态寄存器 (CAN_ERROR_STATUS) 指示的。ERROR_STATE 位 (CAN_ERROR_STATUS[17:16]) 指示 CAN 节点处于哪种错误状态。CAN 中的错误状态是根据下面两个计数器的值决定的：

- 发送错误计数器 (CAN_ERROR_STATUS[7:0])
- 接收错误计数器 (CAN_ERROR_STATUS[15:8])

根据 CAN 2.0B 规范可以修改错误计数器。

如果发送错误计数器的值和接收错误计数器的值不大于 127 (十进制)，则某个节点将处于“主动错误”状态。如果发送错误计数器的值或接收错误计数器的值超过或等于 128 (十进制)，则一个节点将处于“被动错误”状态。如果发送错误计数器的值超过或等于 256 (十进制)，则一个节点处于“总线关闭”状态。

当发送错误计数器值和接收错误计数器值都不大于 127 时，则“被动错误”节点再次变为“主动错误”节点。

11 个连续隐性位在总线上被监控 128 次后，处于“总线关闭”状态的节点会变为“主动错误”的节点 (在主动错误状态下，将所有错误计数器值都设置为‘0’)。

错误状态寄存器具有两位：‘txgte96’ (CAN_ERROR_STATUS[18]) 和 ‘rxgte96’ (CAN_ERROR_STATUS[19])。这两位表示发送错误计数器的值和接收错误计数器的值是否大于或等于 96 (十进制)。该功能作为一个错误警告，因为如果一个错误计数值大于 96 和大约 96 则表示总线受严重影响。

16.9.4 CAN 中的中断源

可以将 CAN 中断源分为两类：CAN 内核中断和 TTCAN 中断。控制寄存器 (CAN.CNTL) 中的 TT_ENABLE 位控制着中断的生成：

- 如果 TT_ENABLE = 0，则通过 CAN 内核中断 (INT_STATUS 和 INT_ENBL) 可直接生成中断
- 如果 TT_ENABLE = 1，则通过内核中断和 TTCAN 定时器中断来生成中断

16.9.4.1 CAN 的内核中断

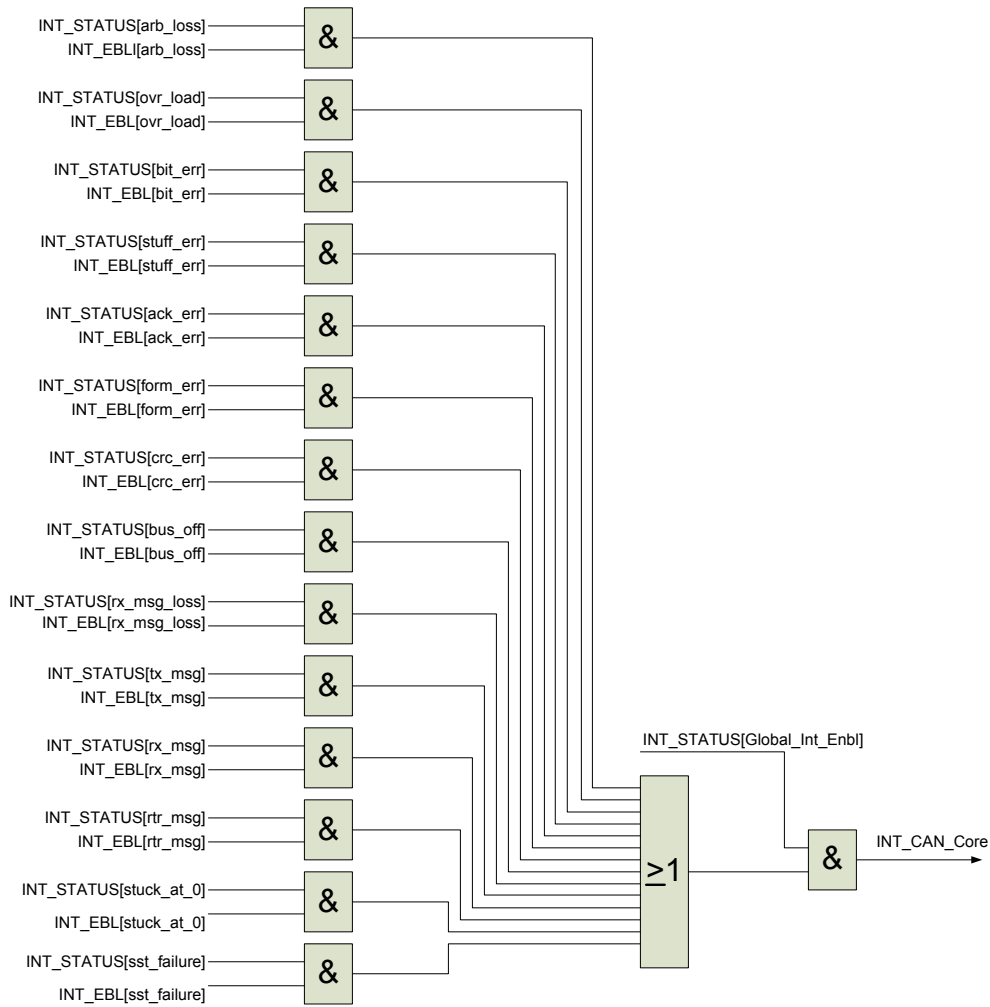
内核中断由一个中断状态寄存器 (CAN_INT_STATUS) 和一个中断使能寄存器 (CAN_INT_EBL) 控制，如图 16-16 所示。中断状态寄存器存储着内核中断事件。当某位被置位时，它会保持设置状态直到写入数值‘1’来被清除为止。中断使能寄存器不会对中断状态寄存器产生任何影响。

中断使能寄存器 (INT_EBL) 控制中断状态寄存器中用于激活中断输出 (INT_CAN_Core) 的特殊位。如果一个特殊的中断状态位和相应的使能位被置位，则 INT_CAN_Core 被激活。INT_CAN_Core 路由是由 CAN_CNTL.TT_ENABLE 位控制的。如果 TT_ENABLE 位为零，会将 INT_CAN_Core 信号直接路由到 CAN 模块中断输出。如果 TT_ENABLE 位被置位 (1)，则根据 INTR_CAN_SET 寄存器的设置来路由中断，如图 16-11 所示。

CAN 中的各个内核中断源如下：

- rx_msg: 表示已接收了一个信息。
- tx_msg: 表示已发送了一个信息。
- rx_msg_loss: 当收到一个新的信息，但 RxMessage 标志位 MSG_AV 已经被设置，且 LINK_FLAG 未被设置时，会触发该中断。此时，新的信息将被丢弃，因为没有任何缓冲区可存储它。
- bus_off: CAN 已达到总线关闭状态。
- crc_err: 检测到 CRC 错误。
- form_err: 检测到信息格式错误。
- ack_err: 检测到信息确认错误。
- stuff_err: 检测到位填充错误。
- bit_err: 检测到位错误。
- ovr_load: 接收到过载帧。
- arb_loss: 表示发送信息时丢失仲裁。
- stuck_at_0: 检测到 dominant(0) 阻塞错误。
- rtr_msg: 发送的 RTR 自动回复信息。
- sst_failure: 一次性发送。

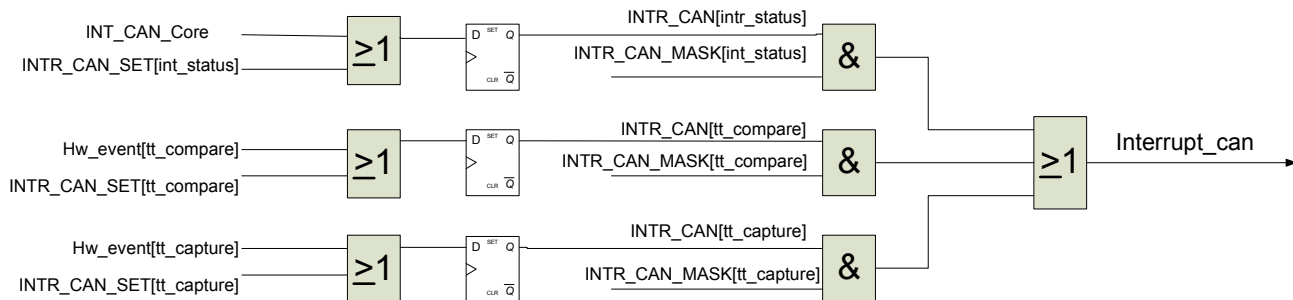
图 16-16. CAN 内核中的中断



16.9.4.2 TT_ENABLE = 1 时的中断路由

如果 CNTL.TT_ENABLE 位被设置为数值 ‘1’，则根据 INTR_CAN_SET 和 INTR_CAN_MASK 寄存器的设置（如图 16-17 所示）并通过 CAN 内核中断或额外的 TTCAN 定时器硬件中断来生成 CAN 中断信号。

图 16-17. CAN_CNTL.TT_ENABLE = 1 时的 CAN 中断



对 INTR_CAN_SET 和 INTR_CAN_MASK 进行逻辑 AND 运算可以在 INTR_CAN_MASKED 寄存器中进行。如果通过使用 INTR_CAN_SET 寄存器使能 CAN 内核中断、TTCAN 定时器比较和捕获中断，则 INTR_CAN 寄存器会保持它们的状态。

16.10 CAN 的工作模式

CAN 模块运行于三种不同的模式。指令寄存器 CAN_COMMAND 用于通过为每种模式设置相应位来选择工作模式。三种工作模式分别为：

- 运行 / 停止模式：CAN_COMMAND[0]
- ‘仅倾听’模式：CAN_COMMAND[1]
- 环回测试模式：CAN_COMMAND[2]

16.10.1 运行 / 停止模式

当CAN控制器正常运行时，它处于运行模式。分别通过设置、清除 CAN_COMMAND[0] 位，可分别使 CAN 控制器处于运行模式、停止模式。

16.10.2 “仅倾听”模式

在“仅倾听”模式中，CAN 控制器只会倾听 CAN 接收线，而不会确认 (ACK) 总线上接收的信息。在这种模式下，它不会发送任何信息。但要更新错误标志，以便进行调整位时序直到不发生错误为止。

通过以下步骤实现自动检测波特率：

1. 初始化 CAN 控制器，用以接收所有信息（全局 / 局部掩码位被设置为‘0’）。
2. 将第一个可用 CANOpen 比特率（10 Kbps）的位时序值加载到 CAN 控制器内，然后 CAN 控制器会切换为‘仅倾听’模式。
3. 假设在网络上进行通信并且波特率正确，那么将接收信息。
4. 错误寄存器不发生改变，并且接收信息的标志被设置在 CAN 控制器内。这便表示检测到了正确的波特率。
5. 假设波特率不正确，将更新错误标志（stuff 错误、CRC 错误或 form 错误）。
6. CAN 控制器被关闭，并且下一个可能的位时序值会从比特率表中下载。

16.10.3 环回测试模式

环回模式用于测试目的。根据指令寄存器的 LOOPBACK 位 (CAN_COMMAND[2]) 和 LISTEN 位 (CAN_COMMAND[1]) 的设置情况，PSoC 4200L 器件中的 CAN 控制器模块支持两种环回模式。

16.10.3.1 外部环回模式

当 COMMAND_LOOPBACK 被设置为 1，并且 COMMAND_LISTEN 被设置为 0 时，可使能外部环回模式。在这种模式下，可通过外部将 CAN_TX 输出引脚连接到 CAN_RX 输入引脚；该 IP 可以接收其自身传送的数据传输。

16.10.3.2 内部环回模式

当设置 COMMAND_LOOPBACK、COMMAND_LISTEN 为 1 时，使能内部环回模式。在这种模式下，所发送的数据传输被内部路由由回接收器逻辑。

17. 通用串行总线（USB）



PSoC[®] USB 模块作为 USB 从设备使用并与 USB 主设备进行通信。该 USB 模块作为 PSoC 设备中的固定功能数字模块使用。它支持全速通信（12 Mbps），并符合 USB 规范版本 2.0 标准。USB 从设备支持主机的即插即用应用和热插拔功能。本节详细介绍了 PSoC USB 模块以及各种传输模式。更多有关 USB 规范的信息，请访问 USB 实施者论坛网站。

17.1 特性

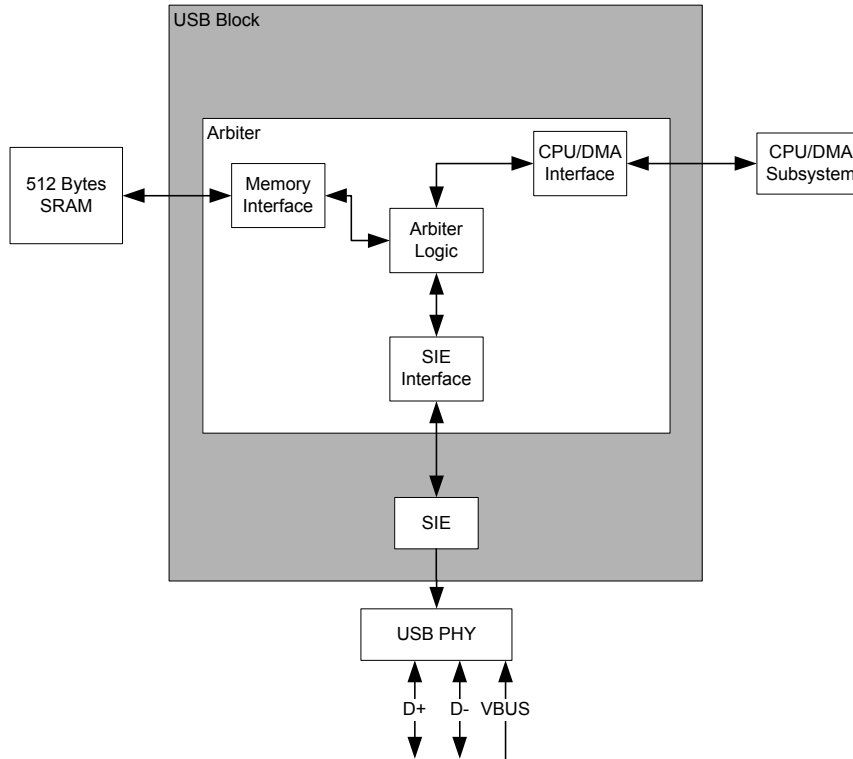
PSoC USB 具有以下特性：

- 符合 USB 2.0 规范
- 支持全速外设操作（信号比特率为 12 Mbps）
- 支持 8 个数据端点和一个控制端点
- 支持批量、中断、同步和控制 4 种传输类型
- 支持总线供电和自供电配置
- 支持 USB 暂停模式，以降低功耗
- 支持两种逻辑传输模式：
 - 存储和转发模式
 - 直通模式
- 差分信号（D+ 和 D-）输出
- 在 D+ 和 D- 每一个信号线上集成了一个 22 Ω 大小的 USB 终端电阻，且在 D+ 信号线上还集成了一个 1.5 k Ω 的上拉电阻
- 电池充电器检测（BCD）符合 USB 电池充电规范版本 1.2（只检测外设）
- 在存储和转发模式下最大能够传输 64 字节的数据包，而在直通模式下最大能同步传输 1023 字节的数据包。

17.2 框图

图 17-1 显示的是 USB 模块的架构。它包含 USB 物理层（USB PHY）、串行接口引擎（SIE）、仲裁器以及 512 字节的局部存储器缓冲区。

图 17-1. USB 框图



17.2.1 USB 物理层（USB PHY）

USB 物理层通过 D+、D- 和 VBUS 引脚与 USB 主设备进行物理层通信。它处理与主机进行的差分模式通信、VBUS 检测以及监控事件（如 USB 总线上的 SE0）。

17.2.2 串行接口引擎（SIE）

串行接口引擎（SIE）在发送和接收过程中负责解码和创建数据，并控制数据包。在接收过程中，它将 USB 位流解码成 USB 数据包；在发送期间，它创建 USB 位流。下面是 SIE 模块的各种特性：

- 符合 USB 2.0 规范
- 支持一个设备地址
- 支持八个数据端点和一个控制端点
- 为各个端点提供中断触发事件支持
- 在控制端点内集成了一个 8 字节的缓冲区

串行接口引擎的寄存器主要用于配置数据端点操作和控制端点数据缓冲区。另外，该模块还控制着每个端点的可用中断事件。

17.2.3 仲裁器

仲裁器用于处理端点对 SRAM 存储器的访问。CPU、DMA 或 SIE 都可以对 SRAM 存储器进行访问。仲裁器将处理 CPU、DMA 和 SIE 间的仲裁操作。仲裁器包含以下各项：

- SIE 接口模块
- CPU/DMA 接口模块
- 存储器接口
- 仲裁器逻辑

仲裁器寄存器用于处理端点配置以及端点的读地址和写地址。另外，它还配置了每个端点所需要的逻辑传输类型。

17.2.3.1 SIE 接口模块

该模块处理与 SIE 进行的所有数据传输。SIE 读取 SRAM 存储器中的数据并将其发送给主机。同样，它将从主机接收到的数据写入到 SRAM 存储器内。这些请求被注册在 SIE 接口模块内，并由该模块处理。

17.2.3.2 CPU/DMA 接口模块

该模块处理与 CPU 和 DMA 进行的所有数据传输。CPU 为每个端点发出读取 / 写入 SRAM 存储器的请求。这些请求被注册在该接口内，并由该模块处理。当配置 DMA 时，该接口负责 DMA 和 USB 间的所有数据传输。该模块为每个数据端点支持 DMA 请求线。DMA 的性能由配置寄存器中所配置的逻辑传输模式决定。

17.2.3.3 存储器接口

存储器接口用于控制 USB 模块和 SRAM 存储器单元间的连接。最大支持 512 字节的存储器，存储器被分成 256 个存储器单元，每个单元 16 位。这是 USB 的专用存储器。SIE 或 CPU/DMA 可以发出存储器访问的请求。SIE 接口模块和 CPU/DMA 接口模块将处理这些请求。

17.2.3.4 仲裁器逻辑

这是仲裁器的主模块。它对在仲裁器中发生的所有数据传输进行仲裁。它对存储器单元和寄存器的 CPU、DMA 和 SIE 访

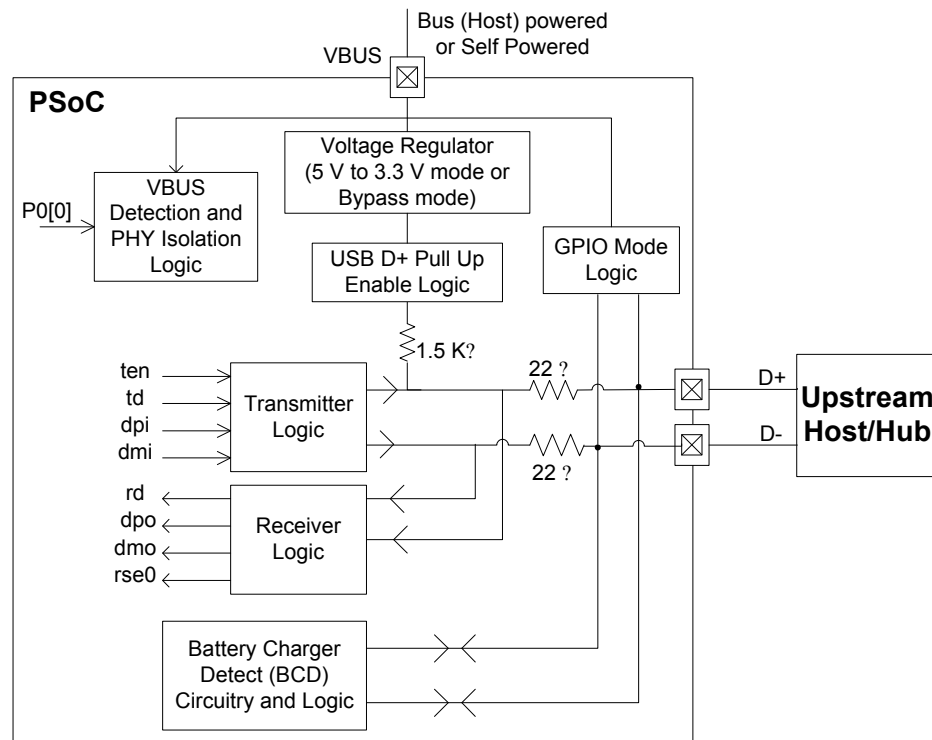
问进行仲裁。另外，它还处理存储器的管理模式。可以“手动”或“自动”进行存储器管理。在手动模式下，读和写地址操作由固件完成。在自动模式下，所有存储器处理操作均由该模块本身完成。该模块负责分配缓冲区的大小。它也处理公用存储器区域。另外，它还处理每个端点的中断请求。

17.3 工作原理

17.3.1 USB 物理层 (USB PHY)

USB 模块含有发送器和接收器（收发器），即 USB PHY。图 17-2 显示的是 PHY 架构。PSoC 中的 USB PHY 还包含 D+ 线上的上拉电阻，以通知主机该设备是全速设备。PHY 在 USB 线上集成了 22 Ω 的串联终端电阻。USB 从设备和主设备间的信号是差分信号。接收器接收来自主机的差分信号，并将其转换成单端信号，以通过 SIE 进行处理。发送器将 SIE 的单端信号转换成差分信号，然后将其发送给主机。在 0 到 3.3 V 的额定电压范围内，差分信号被提供给上行设备。

图 17-2. USB PHY 架构



17.3.1.1 电源方案

USB PHY 由 PSoC 设备的 VBUS 电源焊盘供电。VBUS 焊盘可以由主机的 VBUS (总线供电) 或外部电源 (自供电) 驱动。

USB PHY 需要 3.3 V 的额定电压, 用于与主机通信。USB_CR1 寄存器中的 REG_ENABLE 位控制内部电压调节器的操作。对于 VBUS 电压处于 5 V 范围 (4.35 V 到 5.25 V) 的应用, 必须将 REG_ENABLE 位设置为高电平以使能内部电压调节器。如果 VBUS 电压处于 3.3 V 范围 (3.15 V 到 3.45 V) 内, 则必须清除该寄存器位以使接收器直接连接到电源。

17.3.1.2 VBUS 检测

USB 从设备可以使用主机 VBUS 电源信号提供的电源 (总线供电配置), 也可以使用独立于主机 VBUS 的外部电源 (自供电配置)。PSoC USB 系统使用两个信号来检测是否存在 VBUS。

VBUS 电源焊盘:

PSoC 中的 VBUS 电源焊盘引脚给 USB PHY 和 USB I/O (D+ 和 D- 引脚) 供电。在 PSoC 引脚映射中, VBUS 电源焊盘作为 P13[2] 引脚使用。请注意, 不能将 P13[2] (VBUS 电源焊盘) 作为通用 I/O 使用, 只能将其作为 USB PHY 的电源信号。该引脚可以由主机 VBUS 或独立于 VBUS 的电源供电。USB PHY 中的内部比较器使用 P13[2] I/O 状态信号来表示 VBUS 是否存在。该信号被反映为 GPIO_PRT13_PS 端口状态寄存器中的位 2。另外, 可以配置 GPIO_PRT13_INTR_CFG 寄存器, 以便在 P13[2] 引脚的上升沿 / 下降沿上生成 PICU 中断。可以在 “GPIO 中断 - 所有端口” 中断内读取 GPIO_PRT13_INTR 中断状态寄存器, 以确定 P13[2] 引脚上的某个事件是否触发了该中断。

P0[0] GPIO 引脚:

当满足下列任何条件时, 应该使用 P0[0] GPIO 引脚 (而不是

VBUS 电源焊盘) 来检测 VBUS:

- 当 VBUS 电源焊盘由独立于主机 VBUS 的外部电源供电时, 应该使用 P0[0] 引脚来检测主机 VBUS。
- 当主机 VBUS 电压不在 4.35 V 到 5.25 V 的范围内时, 应该使用 P0[0] 引脚来检测 VBUS。P0[0] 是由 V_{DDD} 供电的 GPIO。请参考 PSoC 设备数据手册, 了解 GPIO 引脚的输入电压阈值以及最大输入电压的 GPIO 规范。如果使用 P0[0] 引脚检测 VBUS, 当将主机 VBUS 连接到 P0[0] 引脚时, 应该安装一个合适的过压保护电路 (如电阻分压器)。

与 P13[2] (VBUS 焊盘) 相似, 固件可以读取 P0[0] 引脚的状态, 并根据端口 0 寄存器配置 PICU 中断。

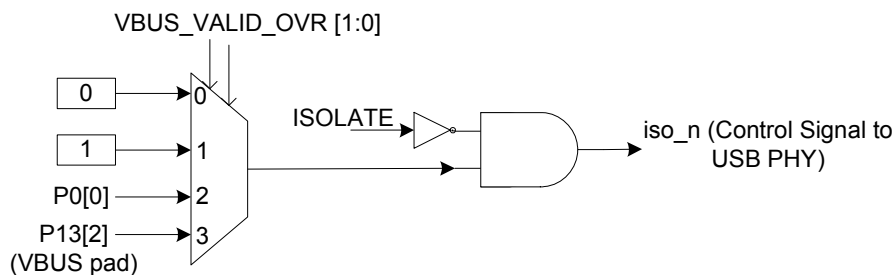
17.3.1.3 USB PHY 隔离逻辑

不存在主机 VBUS 时, USB PHY 使用隔离逻辑来禁用 PHY 输出信号。

图 17-3 显示的是隔离逻辑信号 (iso_n) 的生成情况。iso_n 是低电平有效信号。无论 VBUS 的状态如何, 设备固件都能够通过设置 USB_POWER_CTRL 寄存器中的 ISOLATE 位字段来隔离各个 PHY 输出。对于取决于 VBUS 状态的动态隔离, 设备固件将配置 USB_POWER_CTRL 寄存器中的 VBUS_VALID_OVR[1:0] 位字段, 旨在根据 VBUS 检测所使用的 I/O 来选择 VBUS 焊盘 (P13[2]) 或 P0[0] GPIO 信号。

- 只在通过读取 P13[2] 或 P0[0] 的引脚状态寄存器检测到主机 VBUS 存在后, 设备固件才使能 USB 模块的操作。
- VBUS 存在时, 经过至少 2 μ s 的延迟后, 设备固件将清除 USB_POWER_CTRL 寄存器内的 ISOLATE 位字段。因此, 由 VBUS_VALID_OVR[1:0] 选择的信号将负责处理动态 PHY 输出隔离。当设备固件停止 USB 模块的操作时, 将设置 ISOLATE 位字段。

图 17-3. USB PHY 输出隔离逻辑



17.3.1.4 USB D+ 引脚上拉电阻使能逻辑

如果 USB 从设备为自供电, 则 USB 规范确保设备使能了 D+ 引脚上的上拉电阻, 从而向主机通知它是一个全速设备。清除主机 VBUS 时, 设备将禁用 D+ 线上的上拉电阻, 以防止向主机反向馈电。USB PHY 模块在 D+ 线上集成了 1.5 kΩ 内部上拉电阻, 以通知主机 PSoc 是一个全速设备。通过配置 USBIO_CR1 寄存器中的 USBPUEN 位, 可以使能或禁用上拉电阻。根据 VBUS 检测所使用的是 VBUS 焊盘 (P13[2]) 还是 P0[0] 引脚, 固件将读取该引脚的状态, 以使能 / 禁用 D+ 引脚上的上拉电阻。

17.3.1.5 发送器和接收器的逻辑

发送器逻辑用于将差分信号发送给 USB 主设备。

可以手动强制发送器发送信号。通过使用 USB_USBIO_CR0 寄存器, 可以手动发送这些信号。各种实例如下:

- 使能手动发送时, 可以配置该寄存器来发送单端 0 信号 (即 D+ 和 D- 均为低电平)。
- 可以配置该寄存器, 以发送 USB 信号。USB 信号有两种类型:
 - D+ 为低电平和 D- 为高电平 = J
 - D+ 为高电平和 D- 为低电平 = K
- 该寄存器也有一个用于读取所接收信号电平的位。通过该位可以了解 D+ < D- 还是 D+ > D-。

17.3.1.6 电池充电器检测 (BCD)

USB PHY 支持检测主机下行端口的类型。更多有关检测算法和下行端口类型的信息, 请从 www.usb.org/home 网站上下载电池充电规范版本 1.2。USB PHY 可以检测的下行端口类型包括标准下行端口 (SDP)、充电下行端口 (CDP) 和专用充电端口 (DCP)。检测端口类型的步骤如下:

1. VBUS 检测: 设备检测 VBUS 的存在是进行 BCD 算法的第一步。使用 VBUS 监控引脚上的信号上升沿 / 下降沿向 CPU 触发中断, 从而实现该操作。VBUS 监控引脚可以是 VBUS 电源焊盘, 也可以是 P0[0] 引脚, 如第 174 页上的 17.3.1.2 VBUS 检测所述。
2. BCD 的 USB IP 配置: 下面需要保证正确配置 USB IP, 以执行 BCD。具体流程如下:
 - a. 禁用与 USB 端点相关联的所有 DMA 通道和三个 USB 中断向量。
 - b. 清除 USBDEVv2_CR0 寄存器中的 USB_ENABLE 位。通过清除 USBDEVv2_USBIO_CR1 寄存器中的 USBPUEN 位来禁用 D+ 线上的上拉电阻。

- c. 设置 USBDEVv2_USB_CLK_EN 中的 CSR_CLK_EN 位, 以使能对模块的定时。通过清除 USBDEVv2_USBIO_CR1 寄存器中的 IOMODE 位, 将 D+ 和 D- 引脚配置为 Bit-Bang 模式。
- d. 按照以下内容配置 USBDEVv2_USB_POWER_CTRL 寄存器: 禁用 VBUS 电源焊盘和 D- 引脚上的上拉电阻。配置 VBUS_VALID_OVR[1:0] 位以便从 VBUS 电源焊盘 (PHY 检测器) 或 P0[0] GPIO 引脚 (具体取决于 VBUS 监控所使用的引脚) 生成 vbus_valid 信号。如果设计中并未使用 VBUS 监控引脚, 则配置 VBUS_VALID_OVR[1:0] 位以强制使 vbus_valid 信号始终被设置为 1。设置 SUSPEND、SUSPEND_DEL、ENABLE_RCVR、ENABLE_DPO、ENABLE_DMO、ENABLE_CHGDET 和 ENABLE 位。
- e. 清除 USBDEVv2_USB_POWER_CTRL 寄存器中的 SUSPEND 位; 经过 2 μs 延迟后, 请清除 SUSPEND_DEL 位。
3. 数据线连接检测 (DCD): 如果 VBUS 连接和数据线连接之间存在时间延迟, 则在执行充电器检测前, USB 从设备需要检测数据线的连接情况。该时间延迟是由处理 BCD 算法的下一步骤前所引入的 400 ms 硬编码延迟导致的。
4. 主要检测: 主要检测用于区分 SDP 和 DCP/CDP 端口。要想执行主要检测, 需要设置 USBDEVv2_USB_CHGDET_CTRL 寄存器中的下列各位: REF_EN、REF_DP、COMP_EN 和 COMP_DM。应该清除寄存器的其他位。经过 40 ms 的延迟后, 请读取 USBDEVv2_USBIO_CR1 寄存器中的 DMO 位, 以监控 D- 线的状态。比较器输出由 USBDEVv2_USB_CHGDET_CTRL 寄存器中的 COMP_OUT 位表示。下表指出了这两个信号不同组合的端口类型。

比较器输出	D- 的状态	端口类型
0	0	SDP
0	1	不合适的组合。中止充电器检测。
1	0	DCP 或 CDP。需要执行次要检测。
1	1	可能连接到专用充电器 (D+ 和 D- 被电阻上拉)。CPU 应该读取 D+ 的状态, 从而进行确认。

5. 次要检测：次要检测用于区分 CDP 和 DCP 端口。要想执行次要检测，需要设置 USBDEVV2_USB_CHGDET_CTRL 寄存器中的下列各位：REF_EN、REF_DM、COMP_EN 和 COMP_DP。应该清除寄存器的其他位。经过 40 ms 的延迟后，请通过读取 USBDEVV2_USBIO_CR1 寄存器中的 DPO 位来监控 D+ 线的状态。比较器输出由 USBDEVV2_USB_CHGDET_CTRL 寄存器中的 COMP_OUT 位表示。下表指出了这两个信号不同组合的端口类型。

比较器输出	D+ 的状态	端口类型
0	0	CDP
0	1	不合适的组合。中止充电器检测。
1	0	DCP
1	1	可能连接到专用充电器（D+ 和 D- 被电阻上拉）。CPU 应该读取 D- 信号的状态来确认。

6. 恢复寄存器值：检测端口类型后，需要清除 USBDEVV2_USB_CHGDET_CTRL 寄存器中的所有位。另外，还要清除 USBDEVV2_USB_POWER_CTRL 寄存器中的 ENABLE_CHGDET 位。将 USBDEVV2_USB_POWER_CTRL 寄存器恢复到开始执行 BCD 算法前的值，并清除 USBDEVV2_USB_CLK_EN 寄存器中的 CSR_CLK_EN 位。然后，设置 USBDEVV2_USBIO_CR1 寄存器中的 IOMODE 位。如果检测到的端口类型为 SDP 或 CDP，则可以对 USB 从设备进行枚举。

17.3.1.7 链路层电源管理 (LPM)

类似于暂停模式，USB PHY 也支持链路层电源管理 (LPM)，但各电源状态间存在几十微秒的转换延迟。在暂停 / 恢复模式下，该值延迟长于 20 ms。更多有关 LPM 的信息，请参考 USB 2.0 规范 (LPM ECN 文档和 LPM 勘误表文档) 中介绍的内容。LPM 支持以下功能：

- 需要配置 USBDEVV2_USB_LPM_CTRL 寄存器，以实现以下操作：使能 / 禁用 LPM，使能 LPM 时选择响应类型，收到来自主机的 PID（而不是 LPM 标记）时做出响应。
- USBDEVV2_USB_LPM_STAT 寄存器存储了由主机发送的 Best Effort Service Latency (BESL) 以及远程唤醒特性的值。固件应该在发生 LPM 中断事件时读取该寄存器，并根据主机所发送的 BESL 值进入准确的低功耗模式（深度睡眠或睡眠）。

17.3.2 端点

SIE 和仲裁器支持 8 个数据端点 (EP1 到 EP8) 和一个控制端点 (EP0)。这些数据端点共享了 512 字节的 SRAM 存储器。端点存储器管理模式是“手动”或者“自动”的。通过使用 SIE 和仲裁器寄存器，可以配置这些端点的传输方向或对其进行其他配置。通过仲裁器可以访问端点的读地址和写地址寄存器。

可以单独使能这些端点。在自动管理模式下，通过写入 USB_EP_ACTIVE 寄存器可以控制端点的有效状态。在运行期间，不能动态更改该端点的有效状态。在存储器的手动管理模式下，存储器分配由固件决定，因此无需指定有效端点。在存储器的手动管理模式下，USB_EP_ACTIVE 寄存器被忽略。USB_EP_TYPE 寄存器用于控制端点的传输方向 (IN、OUT)。控制端点使用 8 个单独字节（即 USB_EP0_DR 寄存器）存储它的数据。

17.3.3 传输类型

PSoC USB 支持全速传输，并且符合 USB 2.0 规范。它支持 4 种传输类型：

- 中断传输
- 批量传输
- 同步传输
- 控制传输

更多有关这些传输类型的信息，请参考 USB 规范 2.0。

17.3.4 中断

USB 模块有三条通用中断线：INTR_LO、INTR_MED 和 INTR_HI。更多有关这些中断线的向量编号的信息，请参考第 51 页上的中断章节。USB 模块拥有一个包含了 13 个中断触发事件的预定义集，它可以映射到三个中断中的某一个。每个触发事件映射到的中断线通过 USB_INTR_LVL_SEL 寄存器进行配置。每条中断线都有一个状态寄存器，用于确定导致中断的事件。这些寄存器为 USB_INTR_CAUSE_LO、USB_INTR_CAUSE_MED 和 USB_INTR_CAUSE_HI。

下列 12 种事件可以在中断线上生成中断。

- USB 帧起始 (SOF) 事件
- USB 总线复位事件
- 8 个数据端点 (EP1 – EP8) 中断事件
- 控制端点 (EP0) 中断事件
- 仲裁器中断事件
- 链路层电源管理 (LPM) 事件

下述内容对这些中断事件进行了详细介绍。

17.3.4.1 数据端点中断事件

这些是与数据端点 (EP1-EP8) 相对应的 8 个中断事件。通过使用 USB_SIE_EP_INT_EN 寄存器中的相应位，可以使能 / 禁用各个端点中断事件。通过读取 USB_SIE_EP_INT_SR 状态寄存器，可以了解每个端点的中断状态。发生以下事件时，中断被使能的端点可以触发中断：

- IN/OUT 传输成功完成
- 取消应答 IN/OUT 数据传输（如果设置了 SIE_EPx_CR0 寄存器中相应的 NAK_INT_EN 位）
- 当数据传输过程中存在错误时，将设置 SIE_EPx_CR0 寄存器中的 ERR_IN_TXN 位，并生成中断。更多有关可设置该位的条件的信息，请参考该寄存器的说明内容。

- 如果设置了 SIE_EPx_CR0 中的 STALL 位, 则停止事件会触发中断。如果接收到端点的 OUT 数据包 (SIE_EPx_CR0 中的模式位被设置为 ACK_OUT) 或者接收到 IN 数据包 (模式位被设置为 ACK_IN), 仍然可以发生该停止事件。

17.3.4.2 控制端点 (EP0) 中断事件

在发生以下事件时, 将生成与控制端点 (EP0) 相对应的中断事件:

- IN/OUT 传输成功完成
- 在控制端点上接收到 SETUP 数据包

17.3.4.3 仲裁器中断事件

发生以下事件时, 仲裁器将生成端点的中断事件: 最后的仲裁器中断事件是这些事件经过逻辑 OR 运算后的结果。

DMA 授予:

该事件适用于所有 DMA 模式 — DMA 存储和转发模式 (模式 2) 或直通模式 (模式 3)。当 DMA 控制器发出与该端点相对应的 “Burstend” 信号时, 会触发该事件 (该端点的 DMA 请求先前已被发送给 DMA 控制器)。该请求可以为手动 DMA 请求, 也可以是自动仲裁器 DMA 请求。请求的两种模式拥有相同的授予状态。该状态表示 DMA 数据传输已经完成。固件可以使用该状态指示来确定发送下一个手动 DMA 请求的时间。在设置 DMA 授予状态前, 如果向同一个端点发送了多个请求, 则该模块会清除这些请求。只有第一个请求被发送给 DMA 控制器。

IN 端点的局部缓冲区已满:

在存储和转发模式 (模式 1 和 2) 以及直通模式 (模式 3) 下, 该事件的状态会是在不同的条件下设置的。

- 存储和转发模式: 当整个数据包已被传输到局部存储器内时, 将设置该事件状态。但需要确保为特殊端点编写的的数据等于 USB_SIE_EPx.CNT0 和 USB_SIE_EPx.CNT1 寄存器中为该端点编程的字节数据。
- 直通模式: 在该模式下, 当 IN 端点的专用缓冲区中填满了数据包时, 将设置 “IN 缓冲区的充满状态”。该缓冲区的大小由 USB_BUF_SIZE 寄存器的位 [3:0] 确定。通过使用该状态指示, 可以确定能够编程 USB_SIE_EPx_CR0 寄存器中模式位数值的时间, 以应答该端点的 IN 标记。

缓冲区上溢:

该事件仅在直通模式下发生。如果在专用缓冲区或公用缓冲区中发生缓冲上溢, 将发生该事件。缓冲区的上溢条件包括:

- 在 IN 端点的情况下, 如果 DMA 传输编写的字节数量超过了专用缓冲区内的可用空间, 则专用缓冲区将发生上溢。在接收到 IN 标记前, 该端点不能使用公用的缓冲区。因此, 会发生数据上溢。发生这种上溢的原因可能是错误地编程了 DMA 传输描述符的传输大小, 或错误地编程了 USB_BUF_SIZE 寄存器。
- 在某个 OUT 端点, 如果连续发生了两个 OUT 数据传输, 则专用缓冲区将上溢。之前传输的数据仍被存储在公用区域

内, 并且正在传输的数据将填充到 OUT 端点专用缓冲区内, 因此导致上溢。该上溢可能是由其他 DMA 数据传输导致的总 DMA 带宽限制或专用 OUT 缓冲区空间减少而引起的。

- 在 IN 端点, 如果 DMA 传输编写的字节数量超过了公用区域内的可用空间, 则公用缓冲区将发生上溢。其原因可能是错误地编程了 DMA 传输描述符的传输大小, 或错误地编程了 USB_DMA_THRES 和 USB_DMA_THRES_MSB 寄存器。
- 在 OUT 端点, 如果尚未读取写入到公用区域中的数据, 而新的数据开始覆盖现有数据, 那么公用缓冲区将发生上溢。该上溢是由其他 DMA 数据传输引起的总 DMA 带宽限制或 OUT 端点 DMA 的优先级过低而引起的。

缓冲区下溢:

该事件仅在直通模式下发生。该下溢只针对 IN 端点发生。下溢可能在专用缓冲区或公用缓冲区内发生。专用缓冲区的下溢状态可能由专用 IN 缓冲区大小减少或 DMA 带宽受限引起。由于 DMA 带宽的限制和 / 或 DMA 通道的优先级较低, 下溢可能发生在公用缓冲区内。

设置和处理仲裁器中断事件: 所有端点共享了一个仲裁器中断事件, 它是对各端点的所有事件进行逻辑 OR 运算得到的结果。通过使用 USB_ARB_INT_EN 寄存器, 可以为每个端点配置仲裁器中断事件的主设备使能。可以使用 ARB_INT_SR 状态寄存器来确定触发仲裁器中断事件的端点。

寄存器 USB_ARB_EPx_INT_EN (x = 1 到 8) 用于使能或禁用每个端点的仲裁器中断事件。通过使用 USB_ARB_EPx_INT_SR 寄存器可以读取这些子事件的状态。

17.3.4.4 USB 帧起始 (SOF) 事件

- 每次收到 SOF 时都会生成该事件。通过设置 USB_INTR_SIE_MASK 寄存器中的 SOF_INTR_MASK 位, 可以使能 SOF 中断。SOF 中断的状态由 USB_INTR_SIE 状态寄存器中的 SOF_INTR 状态位表示。
- SOF 中断状态也可以由 USB_INTR_SIE_MASKED 寄存器的 SOF_INTR_MASKED 位表示, 该位是对 USB_INTR_SIE_MASK 寄存器和 USB_INTR_SIE 寄存器中相应 SOF 位进行逻辑 AND 运算后得到的结果。
- SIE 中的 USB 数据包解码器对 SOF PID 进行解码并使能了宽度为一个 12 MHz 时钟周期的脉冲。该脉冲与系统时钟 (SYSCLK) 同步, 并且可以通过数字系统互联 (DSI) 接口将其作为数字信号进行路由。除了中断功能外, 该接口还有路由功能。

17.3.4.5 USB 总线复位事件

- 每次发生 USB 总线复位时, 都会生成该事件。通过设置 USB_INTR_SIE_MASK 寄存器中的 BUS_RESET_INTR_MASK 位, 可以使能总线复位中断。总线复位中断的状态由 USB_INTR_SIE 状态寄存器中的 BUS_RESET_INTR 状态位表示。

- 总线复位中断状态也可以由 USB_INTR_SIE_MASKED 寄存器的 BUS_RESET_INTR_MASKED 位表示，该位是对 USB_INTR_SIE_MASK 寄存器和 USB_INTR_SIE 寄存器中相应总线复位位进行逻辑 AND 运算得到的结果。
- 可以使用 32 kHz 的低频时钟 (LFCLK) 来检测 USB 总线的复位状态。需要通过应用固件来使能 LFCLK。在 USB 总线上检测到 SE0 时，SIE 逻辑将通过由 LFCLK 提供脉冲的计数器触发。当该计数器达到 USB_BUS_RST_CNT 寄存器中所配置的计数值时，将触发总线复位中断。建议将 USB_BUS_RST_CNT 寄存器中的计数值设置为 3，从而可以检测总线的复位状态。

17.3.4.6 链路层电源管理 (LPM) 事件

每次收到 LPM 标记数据包时都会生成该事件。通过设置 USB_INTR_SIE_MASK 寄存器中的 LPM_INTR_MASK 位，可以使能 LPM 中断。LPM 中断的状态由 USB_INTR_SIE 状态寄存器中的 LPM_INTR 状态位表示。

LPM 中断状态也可以由 USB_INTR_SIE_MASKED 寄存器的 LPM_INTR_MASKED 位表示，该位是对 USB_INTR_SIE_MASK 寄存器和 USB_INTR_SIE 寄存器中相应 LPM 位进行逻辑 AND 运算后得到的结果。

固件需要读取 USBDEVV2_USB_LPM_STAT 寄存器，从而读取 BESL 和远程唤醒值，并使器件进入所需要的低功耗模式。推荐使用主代码使器件进入低功耗模式。在暂停模式下，退出 LPM 模式与恢复事件唤醒相同。

17.3.5 DMA 支持

每个数据端点都有一个 DMA 通道，用于在端点缓冲区和 SRAM 存储区间进行数据传输。USB IP 为相应的 DMA 通道生成 DMA 请求信号 (usb.dma_req[7:0])，从而启动端点的数据传输。该信号可作为触发组 1 复用器的触发输入使用，并且可以连接到第 8 到第 15 个 DMA 通道其中一个触发输入上。更多有关触发组 1 的信息，请参考第 35 页上的 [DMA 控制模式](#) 章节。来自端点 DMA 通道的传输完成信号 (cpuss.dmac_tr_out[8:15]) 可作为输入信号 usb.dma_burs-

tend[0:7] 路由到 USB IP。触发组 3 复用器用于路由这些信号。这些触发复用器的 8 个输出信号为 usb.dma_burstend[0:7]。可以从下表所示的 9 个输入信号路由这些输出。通过配置 PERI_TR_GROUP3_TR_OUT_CTLx 寄存器 (x 可以为 0 到 7)，可以选择其中一个输出。

表 17-1. 触发组 3 复用器输出的输入源
(usb.dma_burstend[0:7])

输入源	说明
0	软件触发 — 固定连接到逻辑 0
1	cpuss.dmac_tr_out[8]
2	cpuss.dmac_tr_out[9]
3	cpuss.dmac_tr_out[10]
4	cpuss.dmac_tr_out[11]
5	cpuss.dmac_tr_out[12]
6	cpuss.dmac_tr_out[13]
7	cpuss.dmac_tr_out[14]
8	cpuss.dmac_tr_out[15]

17.4 逻辑传输模式

PSoC 设备中的 USB 模块支持两种逻辑传输。通过使用每个端点的寄存器设置，可以配置这些逻辑传输。任何逻辑传输方式都支持 USB 2.0 规范中所述的三种数据传输类型（中断、批量和同步）。任何 USB 从设备都需要具有控制传输功能。

逻辑传输模式是存储器管理和 DMA 配置的组合。逻辑传输模式与 USB 模块中的数据传输（每个端点的 SRAM 存储器单元的输入和输出）相关联。它并不代表设备和主机间的传输方式（USB 2.0 规范中所指定的传输类型）。

USB 模块支持两种基本的传输模式，表 17-2 详细介绍了这两种模式。

- 存储和转发模式
- 直通模式

表 17-2. USB 传输模式

性能	存储和转发模式	直通模式
SRAM 存储器的使用情况	需要更大的存储器容量	需要较小的存储器容量
SRAM 存储器管理方法	手动	自动
SRAM 存储器的分配	各个端点共享 512 字节大小的 SRAM。分配 SRAM 的操作由固件实现。	该模块自动为每个端点分配更小的存储器容量。存储器剩余部分作为“公用区域”。在传输期间可以使用该区域。
IN 指令	接收 IN 指令前，整个数据包被存储在 SRAM 存储器内。	只在接收到 SRAM IN 指令时，才会向存储器内填充数据。有足够的可用数据（根据 DMA 配置）时，数据将被发送给主机。它不会等待到填充完整个数据。

表 17-2. USB 传输模式

性能	存储和转发模式	直通模式
OUT 指令	发送 OUT 指令时，整个数据包将被写入到 SRAM 存储器内。当整个数据包可用后，它将被复制到 USB 从设备内。	等待有足够的字节数量（根据 DMA 配置）后，才能写入到 SRAM 存储器内。当有足够的字节数量时，它将立即被复制到 USB 从设备内。
数据传输	将所有字节写入到存储器内时，数据将被传输。	当有足够的字节可用时，将传输数据。它不会等待到填充完整数据。
基于 DMA 的类型	无 DMA 模式 手动 DMA 模式	仅支持自动 DMA 模式
受支持的传输类型	最适用于中断和批量传输	最适用于同步传输

每个端点都有一组寄存器，需要在各种工作模式期间处理这些寄存器，如表 17-3 所述。

表 17-3. 端点寄存器

寄存器	说明	内容	使用说明
ARB_RWx_WA	端点写地址寄存器	SRAM 的地址	该寄存器指出了存储数据寄存器中的数据的 SRAM 地址。
ARB_RWx_RA	端点读地址寄存器	SRAM 的地址	该寄存器指出了 SRAM 地址，必须读取该地址的数据，并将该数据存储在数据寄存器内。
ARB_RWx_DR	端点数据寄存器	8 位数据	通过读取 / 写入数据寄存器可以执行所有数据传输。 IN 指令: 写入到数据寄存器中的数据被复制到 WA 寄存器所指定的 SRAM 地址。进行写操作后，WA 值将自动递增，以指向下一个存储器地址。 OUT 指令: 读取由 RA 寄存器指向的 SRAM 地址中的数据，并将该数据存储在 DR 内。读取 DR 时，RA 的值将自动递增，以指向下一个必须读取的 SRAM 存储器地址。
SIE_EPx_CNT0 和 SIE_EPx_CNT1	端点字节计数寄存器	字节数量	保存可以传输的字节数量。 IN 指令: 保存将被传输给主机的字节数量。 OUT 指令: 保存可以接收的最大字节数量。固件对该端点可接收的最大字节数量进行编程。SIE 使用端点接收到的字节数量更新该寄存器。
SIE_EPx_CR0 中的“模式”位	模式值	对主机作出响应	控制 USB 从设备对 USB 传输和 USB 主设备作出响应的方式。某些模式实例为 ACK、NAK、STALL，等等。更多有关信息，请参见第 178 页上的表 17-2。

在存储器手动管理模式下，端点读 / 写地址寄存器通过固件更新。因此，可以根据用户要求分配寄存器，并且存储器分配决定有效端点。因此用户可以将 512 个字节分配给所有 8 个或更少数量的端点。

在存储器自动管理模式下，端点读 / 写地址寄存器通过 USB 模块更新。该模块将存储器分配给使用 USB_EP_ACTIVE 寄存器激活的端点。所分配的存储器大小由 USB_BUF_SIZE 寄存器的值决定。分配后剩下的存储器容量被称为“公用区域”存储器，用于传输数据。

在所有模式下，可以使用 8 位端点数据寄存器或 16 位端点数据寄存器来读取 / 写入端点缓冲区。将数据传输到 16 位数据寄存器时，必须确保相应的 SRAM 存储器地址位置是 16 位对齐的。

以下内容介绍了每种模式的 IN 和 OUT 数据传输的算法。USB 主设备（例如，PC）读取数据时，将发生 IN 数据传输。USB 主设备将数据写入到 USB 从设备内时，将发生 OUT 数据传输。通过 USB_ARB_CFG 寄存器可以配置使用 DMA 还是存储器管理，而且该模式适用于所有端点。

17.4.1 存储和转发模式

17.4.1.1 无 DMA 访问

这是无 DMA 访问的存储器手动管理模式。

IN 数据传输 (CPU 写、SIE 读)：在 IN 端点上执行 IN 数据传输的步骤如图 17-4 所示。OUT 数据传输 (CPU 读、SIE 写)：在 OUT 端点上执行 OUT 数据传输的步骤如图 17-5 所示。

图 17-4. 无 DMA 访问的 IN 数据传输

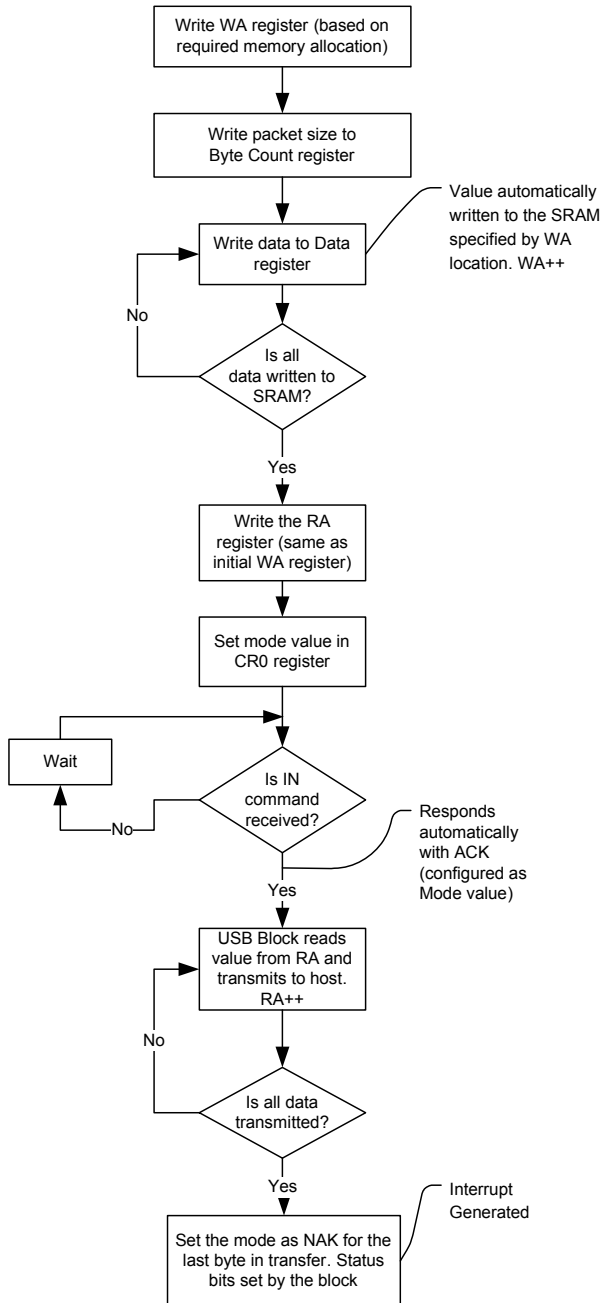
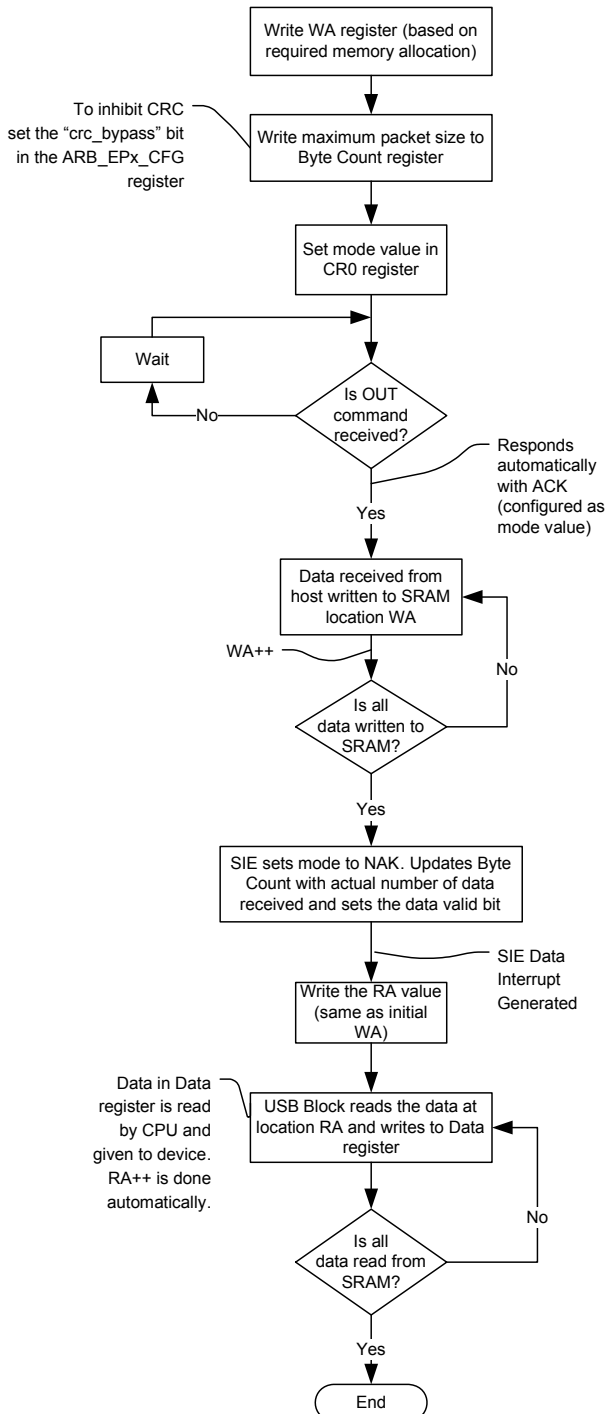


图 17-5. 无 DMA 访问的 OUT 数据传输

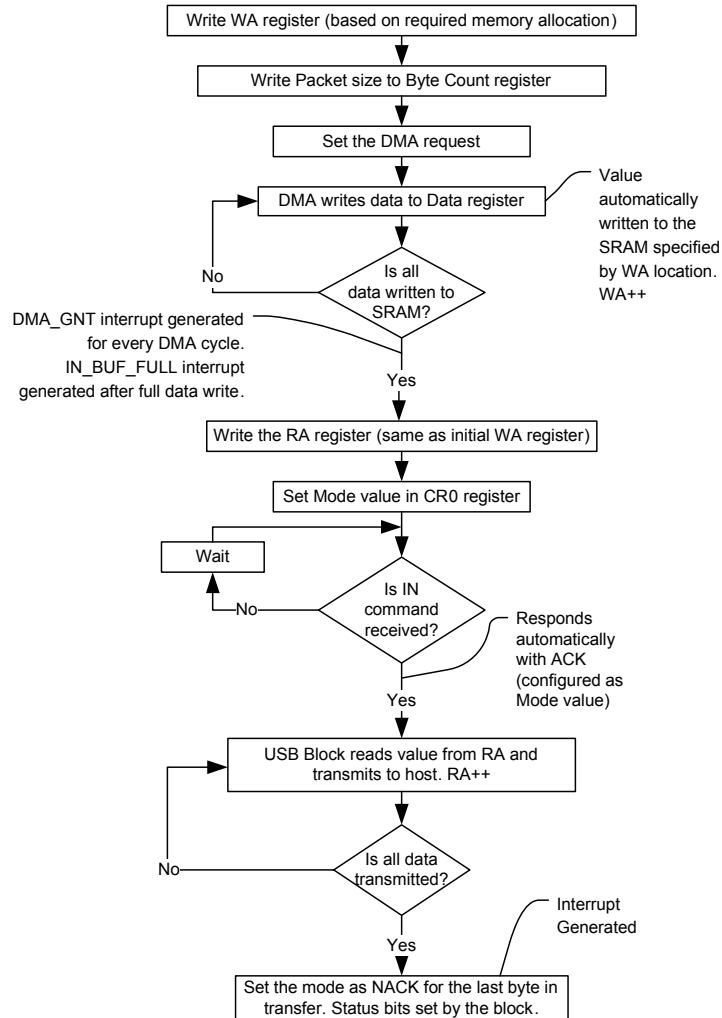


17.4.1.2 手动 DMA 访问

这是带手动 DMA 访问的存储器手动管理模式。该模式要求配置 DMA 控制器。更多有关 DMA 配置的信息，请参考第 35 页上的 [DMA 控制模式章节](#)。该模式与无 DMA 访问的模式相同，但读/写数据包的操作由 DMA 执行。通过设置 USB_ARB_EPx_CFG 寄存器中的 DMA_CFG 位，可以生成端点的 DMA 请求。当 DMA 服务得到确认并且被执行时 (DMA_GNT)，可以编程仲裁器，以生成中断。使用单个 DMA 周期或多个 DMA 周期执行该传输。每次完成一个 DMA 周期后，将生成仲裁器中断 (DMA_GNT)。同样，当所有数据字节 (由所编程的字节计数决定) 被写入到存储器内时，将发生仲裁器中断，并设置 IN_BUF_FULL 位。

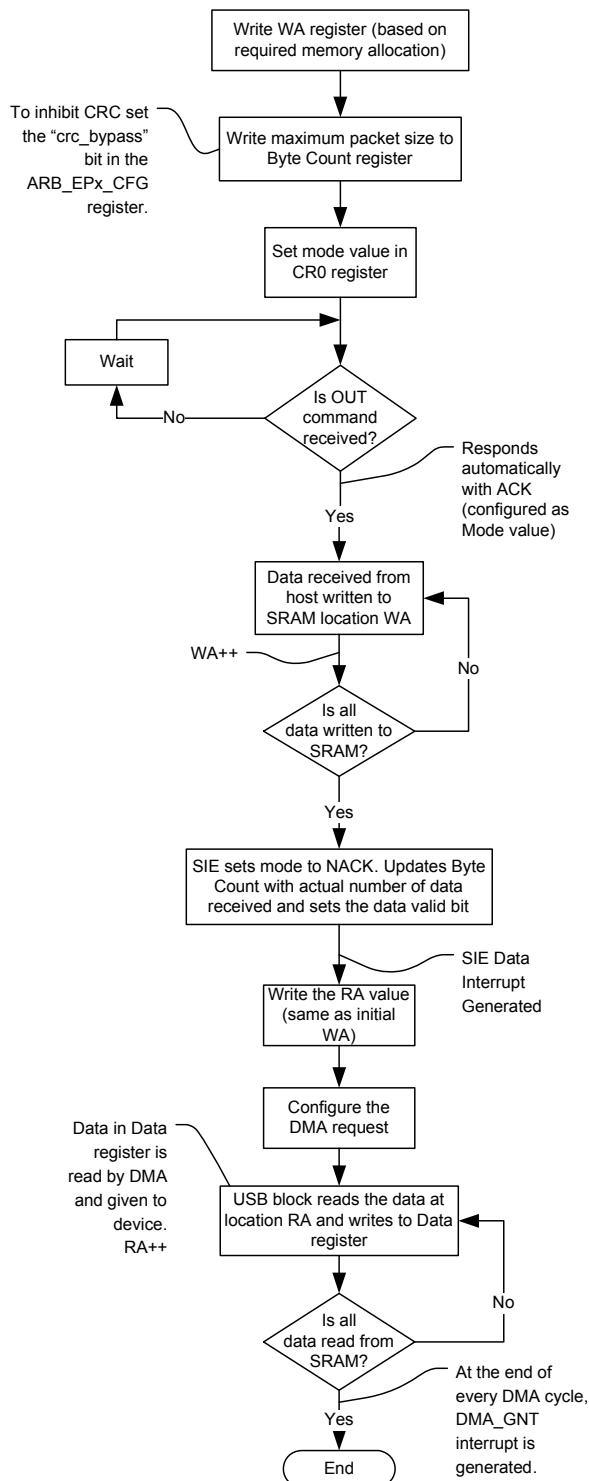
IN 数据传输 (CPU 写、SIE 读)。在 IN 端点上执行 IN 数据传输的步骤如图 17-6 所示。

图 17-6. 手动 DMA 的 IN 数据传输



OUT 数据传输 (CPU 读、SIE 写)：在 OUT 端点上执行 OUT 数据传输的步骤如图 17-7 所示。

图 17-7. 手动 DMA 的 OUT 数据传输



17.4.2 直通模式

这是带自动 DMA 访问的存储器手动管理模式。CPU 为 IN/OUT 数据包编程初始缓冲区大小，并通知仲裁器模块准备使用的特殊应用的端点配置内容。然后，该模块将控制存储器分配并处理所有存储器指针。在存储器分配期间，使用 USB_BUF_SIZE 寄存器（每个端点 32 个字节）给每个有效的 IN 端点（由 USB_EP_ACTIVE 和 USB_EP_TYPE 寄存器设置）分配一个较小的存储空间。剩下的存储空间（256 个字节）被称为“公用区域”，并由所有端点公用。

在该模式下，需要较小的存储空间，因此适用于全速同步的传输（数据包大小可达 1023 个字节）。

当主设备发送 IN 指令时，从设备将对存储在端点专用存储器区域内的数据作出响应。同时，它还会发送一个 DMA 请求，以获取该端点更多的数据。该数据将被存储在公用区域内。从设备不会等待整个数据包可用。它只等到 SRAM 存储器内的（USB_DMA_THRES_MSB、USB_DMA_THRES）数据数量可用时便从公用区域开始传输。

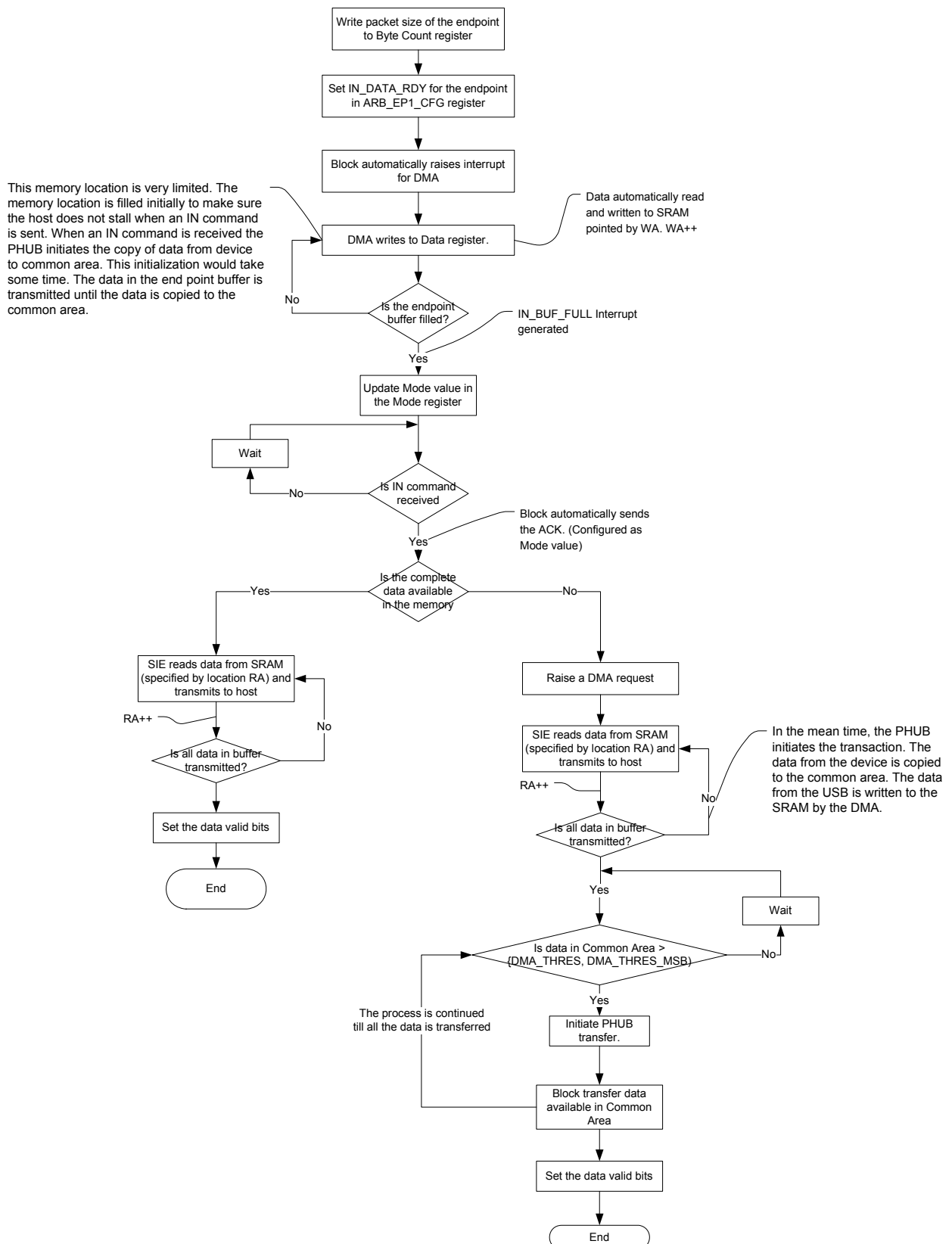
同样当收到 OUT 指令时，OUT 端点的数据将被写入到公用区域内。当某些数据（大于 USB_DMA_THRES_MSB、USB_DMA_THRES）在公用区域中可用时，仲裁器模块将初始化一个 DMA 请求，并且这些数据立即被写入到从设备内。从设备不用等到填充完公用区域。

该模式要求对 USB_DMA_THRES 和 USB_DMA_THRES_MSB 寄存器进行配置，以保留 DMA 在次传输中能传输的字节数量（32 个字节）。同样，DMA 的突发计数（burst count）始终要等于 USB_DMA_THRES 寄存器中的值。除了 DMA 配置外，该模式还需要对 IN 和 OUT 缓冲区的 USB_BUF_SIZE 以及 USB_EP_ACTIVE 和 USB_EP_TYPE 寄存器进行配置。

每个 DMA 通道都有适用于该模式的两个描述符。每个描述符被视为 32 字节的数据块并根据触发机制执行。这些描述符被链接在一起，因此不需要固件的干预仍能传输 64 个字节。当两个描述符传输完成时，将触发端点 DMA 完成中断和 DMA 错误中断（由于缺少传输数据）。更新这些描述符，以升级源 SRAM（IN 端点）或目标 SRAM（OUT 端点）指针位置，然后再次使能这些描述符。另外，固件也触发了 DMA 传输。该序列持续执行，直到传输完所有数据为止。更多有关 DMA 配置的信息，请参考第 35 页上的 DMA 控制模式章节。

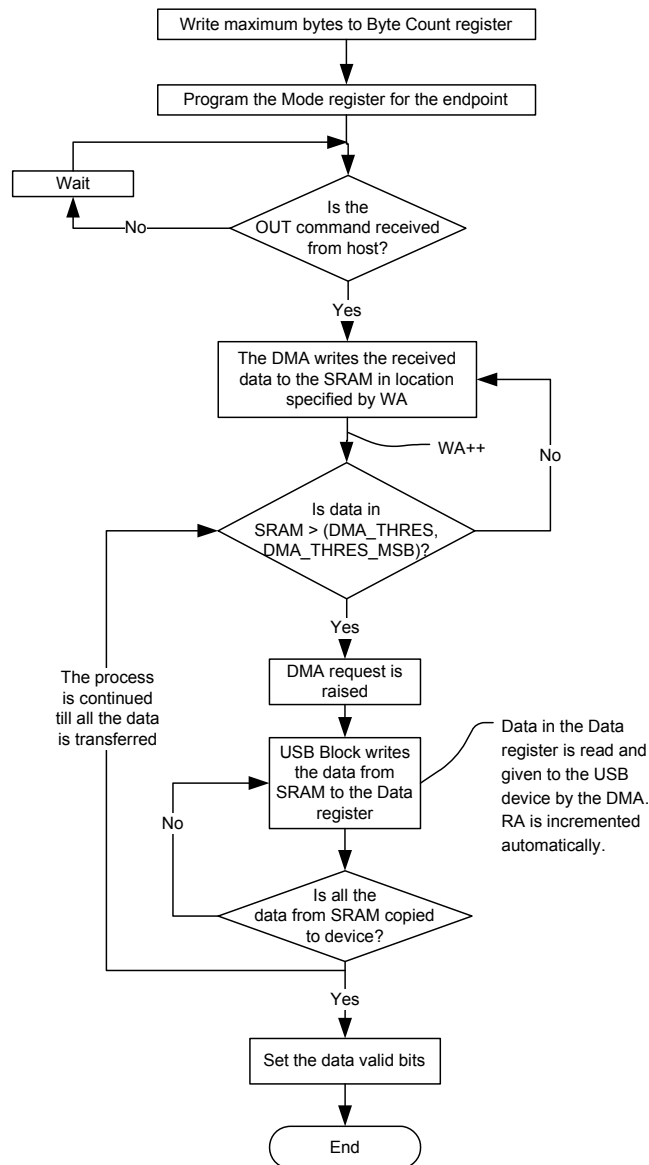
IN 数据传输（CPU 写、SIE 读）：在 IN 端点上执行 IN 数据传输的步骤如图 17-8 所示。

图 17-8. 直通模式的 IN 数据传输



OUT 数据传输（CPU 读、SIE 写）。在 OUT 端点上执行 OUT 数据传输的步骤如图 17-9 所示。

图 17-9. 直通模式的 OUT 数据传输



17.4.3 控制端点的逻辑传输

控制端点具有一个特殊的逻辑传输模式。该端点并不使用 512 字节存储器，而使用了专用的 8 字节寄存器缓冲区（USB_EP0_DRx 寄存器）。下列各图详细介绍了控制端点的 IN 和 OUT 数据传输。

图 17-10. 控制端点的 IN 数据传输

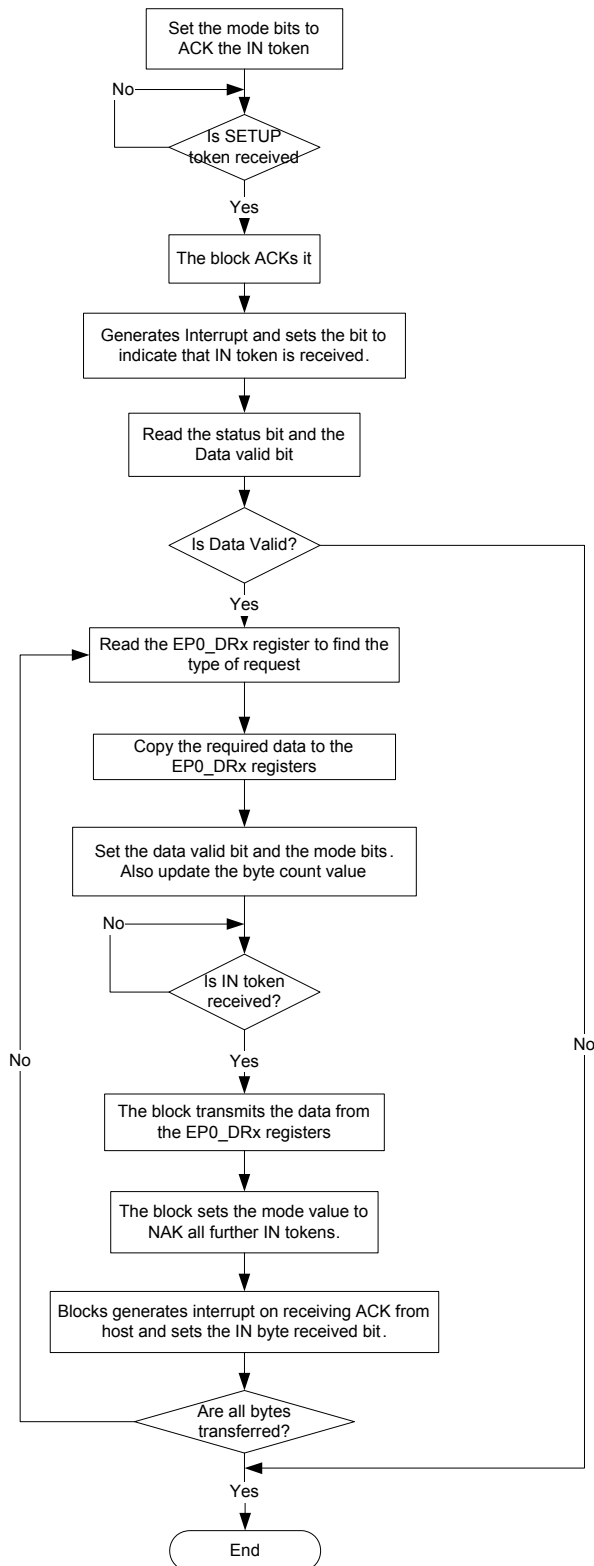
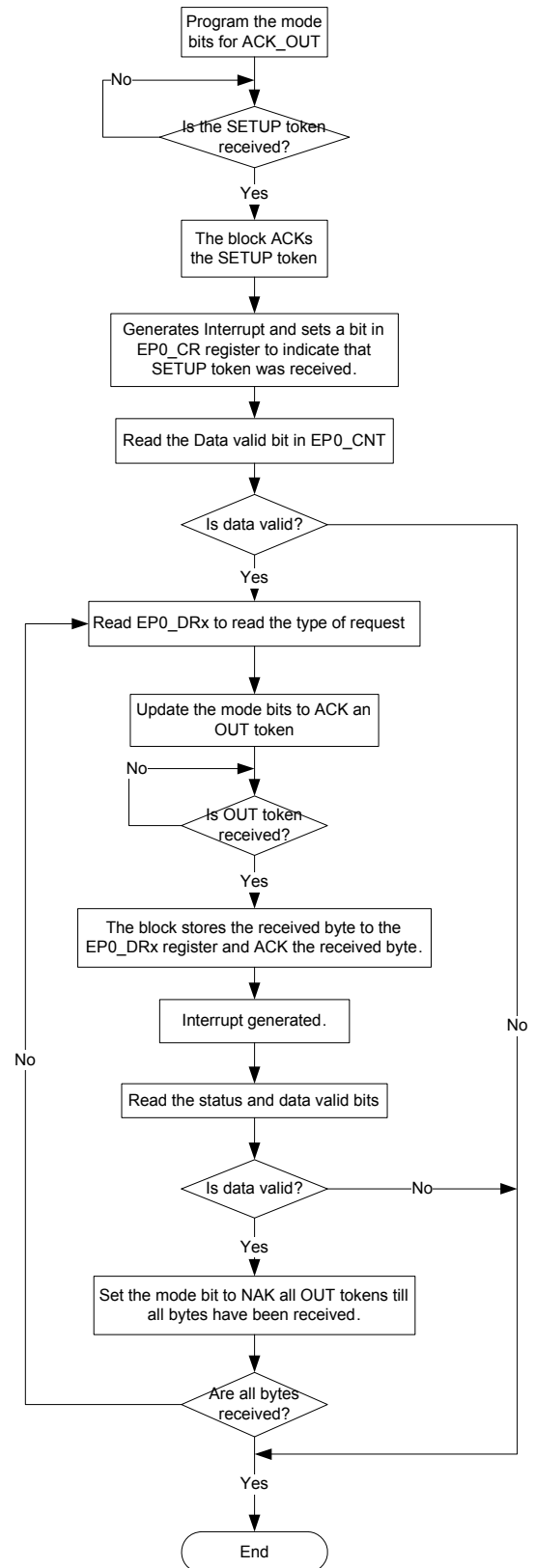


图 17-11. 控制端点的 OUT 数据传输



17.5 寄存器汇总

寄存器名称	说明
USBDEVV2_EP0_DRx	控制端点 EP0 数据寄存器。对于 8 字节 EP0 数据缓冲区，‘x’ 为 0 到 7 中的一个数字。
USBDEVV2_CR0	USB 控制 0 寄存器，用于使能 USB 从设备和指定 7 位设备地址。
USBDEVV2_CR1	USB 控制 1 寄存器。该寄存器用于配置 USB 模块的电压调节器、读取总线的活动状态以及使能 USB 传输的内部振荡器。
USBDEVV2_SIE_EP_INT_EN	USB SIE 数据端点中断使能寄存器
USBDEVV2_SIE_EP_INT_SR	USB SIE 数据端点中断状态
USBDEVV2_SIE_EPx_CNT0	非控制端点字节计数器寄存器。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。该寄存器存储 11 位字节计数器的三个最高有效位、数据包的切换状态以及数据有效状态。
USBDEVV2_SIE_EPx_CNT1	非控制端点字节计数器寄存器。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。该寄存器存储 11 位字节计数值的低 8 位。
USBDEVV2_SIE_EPx_CR0	非控制端点控制寄存器。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。该寄存器包含端点的工作模式、错误状态、停止控制和 NAK 中断的生成。
USBDEVV2_USBIO_CR0	USBIO 控制 0 寄存器。该寄存器包含 USB I/O（分别为 D+ 和 D- 引脚）的控制和配置位，用于单端模式和差分模式。
USBDEVV2_USBIO_CR1	USBIO 控制 1 寄存器。该寄存器包含 USB I/O（分别为 D+ 和 D- 引脚）的控制和配置位，用于选择 USB 工作模式、读取单端 USBIO 接收器输出以及使能 D+ 线上的上拉电阻。
USBDEVV2_DYN_RECONFIG	USB 动态重新配置寄存器。该寄存器用于控制一个端点的动态重新配置的状态。
USBDEVV2_SOF0	帧起始寄存器。该寄存器包含 SOF 帧编号的低 8 位 [7:0]。
USBDEVV2_SOF1	帧起始寄存器。该寄存器包含 SOF 帧编号的高 3 位 [10:8]。
USBDEVV2_OSCCLK_DR0	振荡器锁定数据寄存器 0。该寄存器包含振荡器锁定电路的加法器输出的低 8 位。
USBDEVV2_OSCCLK_DR1	振荡器锁定数据寄存器 1。该寄存器包含振荡器锁定电路的加法器输出的高 7 位。
USBDEVV2_EP0_CR	端点 0 控制寄存器。该寄存器包含控制端点的工作模式以及控制端点上不同数据包条件的状态位。
USBDEVV2_EP0_CNT	端点 0 计数寄存器。该寄存器存储 4 位字节计数器、数据包的切换状态以及数据有效状态。
USBDEVV2_ARB_EPx_CFG	端点配置寄存器。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。该寄存器包含了用于复位端点缓冲区指针、CRC 旁路功能、手动 DMA 请求和数据就绪状态的各种设置。
USBDEVV2_ARB_EPx_INT_EN	端点中断使能寄存器。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。该寄存器用于配置生成端点中断的条件。
USBDEVV2_ARB_EPx_SR	端点中断状态寄存器。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。该寄存器用于读取端点的中断状态。
USBDEVV2_ARB_RWx_WA	端点写地址值。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。9 位写地址指针的低 8 位。
USBDEVV2_ARB_RWx_WA_MSB	端点写地址值。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。9 位写地址指针的最高有效位。
USBDEVV2_ARB_RWx_RA	端点读地址值。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。9 位读地址指针的低 8 位。
USBDEVV2_ARB_RWx_RA_MSB	端点读地址值。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。9 位读地址指针的最高有效位。
USBDEVV2_ARB_RWx_DR	端点数据寄存器。‘x’ 可以为 1 到 8，同 8 个数据端点中的一个相对应。
USBDEVV2_BUF_SIZE	专用的端点缓冲区大小寄存器。
USBDEVV2_EP_ACTIVE	端点有效指示寄存器。
USBDEVV2_EP_TYPE	端点类型（IN/OUT）指示寄存器。
USBDEVV2_ARB_CFG	仲裁器配置寄存器。该寄存器用于配置 USB 模块的缓冲区管理模式。
USBDEVV2_USB_CLK_EN	USB 模块的时钟使能寄存器。
USBDEVV2_ARB_INT_EN	仲裁器中断使能寄存器。该寄存器包含用于使能每个端点的仲裁器中断生成的配置。
USBDEVV2_ARB_INT_SR	仲裁器中断状态寄存器。该寄存器包含每个端点的仲裁器中断生成状态。
USBDEVV2_CWA	公用区域的写地址寄存器。该寄存器包含公用区域的 9 位写地址指针的低 8 位。
USBDEVV2_CWA_MSB	公用区域的写地址寄存器。该寄存器包含公用区域 9 位写地址指针的最高有效位。
USBDEVV2_DMA_THRES	DMA 突发 / 阈值配置寄存器。该寄存器包含 9 位阈值字节计数值的低 8 位。
USBDEVV2_DMA_THRES_MSB	DMA 突发 / 阈值配置寄存器。该寄存器包含 9 位阈值字节计数值的最高有效位。

寄存器名称	说明
USBDEVV2_BUS_RST_CNT	总线复位计数寄存器。该寄存器指出检测总线复位条件时所需要的 LFCLK 时钟周期数量。
USBDEVV2_MEM_DATAx	这是存储非控制端点数据的 512 字节数据缓冲区。‘x’ 可以为 0 到 511 中的某个值。
USBDEVV2_SOF16	帧起始寄存器的 16 位版本。
USBDEVV2_OSCLK_DR16	振荡器锁定数据寄存器的 16 位版本。
USBDEVV2_ARB_RWx_WA16	端点写地址值寄存器的 16 位版本。‘x’ 可以为 1 到 8 中的某个值。
USBDEVV2_ARB_RWx_RA16	端点读地址值寄存器的 16 位版本。‘x’ 可以为 1 到 8 中的某个值。
USBDEVV2_CWA16	公用区域的写地址寄存器的 16 位版本。
USBDEVV2_ARB_RWx_DR16	端点数据寄存器的 16 位版本。‘x’ 可以为 1 到 8 中的某个值。
USBDEVV2_DMA_THRES16	DMA 突发 / 阈值配置寄存器的 16 位版本。
USBDEVV2_USB_POWER_CTRL	电源控制寄存器
USBDEVV2_USB_CHGDET_CTRL	充电器检测控制寄存器
USBDEVV2_USB_USBIO_CTRL	USB I/O 控制寄存器
USBDEVV2_USB_FLOW_CTRL	流量控制寄存器
USBDEVV2_USB_LPM_CTRL	LPM 控制寄存器
USBDEVV2_USB_LPM_STAT	LPM 状态寄存器
USBDEVV2_USB_INTR_SIE	USB SOF、总线复位和 EP0 中断状态寄存器
USBDEVV2_USB_INTR_SIE_SET	USB SOF、总线复位和 EP0 中断设置寄存器
USBDEVV2_USB_INTR_SIE_MASK	USB SOF、总线复位和 EP0 中断屏蔽寄存器
USBDEVV2_USB_INTR_SIE_MASKED	USB SOF、总线复位和 EP0 中断屏蔽寄存器
USBDEVV2_USB_INTR_LVL_SEL	USB 中断级选择寄存器
USBDEVV2_USB_INTR_CAUSE_HI	高优先级中断原因寄存器
USBDEVV2_USB_INTR_CAUSE_MED	中优先级中断原因寄存器
USBDEVV2_USB_INTR_CAUSE_LO	低优先级中断原因寄存器
USBDEVV2_USB_PHY_TRIM0	PHY 调整控制寄存器
USBDEVV2_USB_PHY_TRIM1	PHY 调整控制寄存器
USBDEVV2_USB_PHY_TRIM2	PHY 调整控制寄存器
USBDEVV2_USB_PHY_TRIM3	PHY 调整控制寄存器
USBDEVV2_USB_CHGDET_TRIM	充电器检测调整值寄存器
USBDEVV2_USB_TRIM	调整值寄存器
USBDEVV2_USB_USBIO_TRIM	I/O 的调整值寄存器

18. 定时器、计数器和 PWM



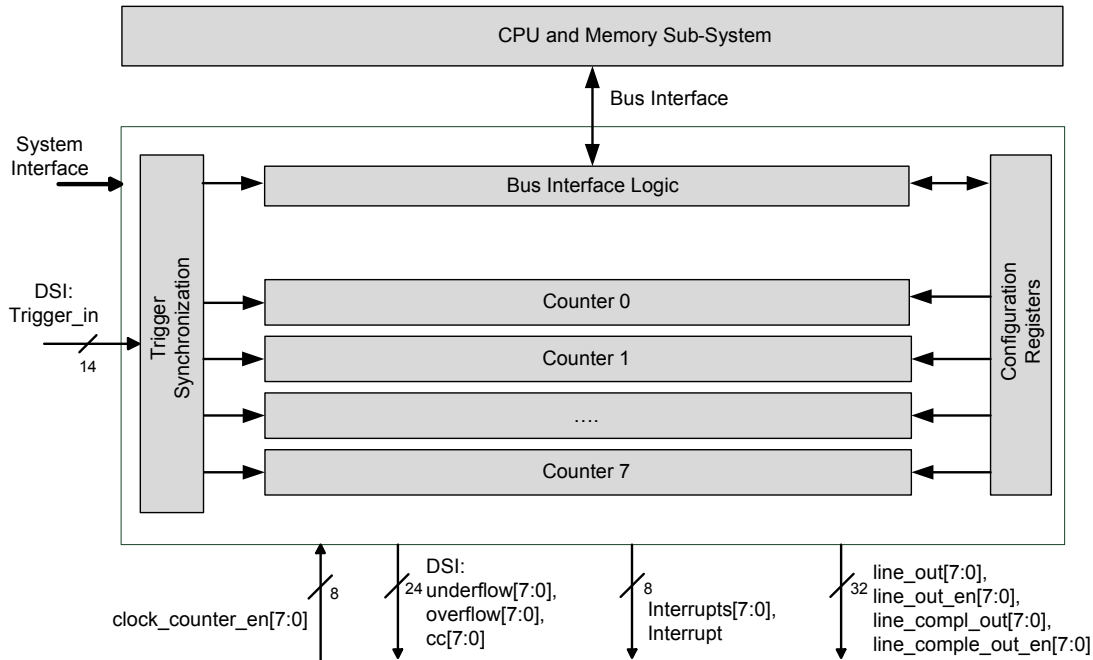
PSoC® 4 中的定时器、计数器和脉宽调制器（TCPWM）模块能够实现 16 位定时器、计数器、脉宽调制器（PWM）和正交解码器等功能。该模块用于测量输入信号的周期和脉宽（定时器），捕获特定事件发生的次数（计数器），生成 PWM 信号或解码正交信号。本节介绍的是 TCPWM 模块的特性、工作原理以及操作模式。

18.1 特性

- 包含八个 16 位定时器、计数器或脉宽调制器（PWM）
- TCPWM 模块支持下面各种操作模式：
 - 定时器
 - 捕获
 - 正交解码
 - 脉宽调制
 - 伪随机 PWM
 - 带死区时间的 PWM
- 多种计数模式 — 递增、递减和递增 / 递减
- 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：
 - 终止计数 — 达到计数器寄存器中的最终值
 - 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
- 同步计数器 — 计数器在相同的时间内可重新加载、启动、停止和计数
- PWM 的互补线路输出

18.2 框图

图 18-1. TCPWM 框图



该模块具有以下接口：

- 总线接口：将模块连接到 CPU 子系统。
- 带有 DSI 的 I/O 信号接口：路由通用数字模（UDB）和 TCPWM 模块的信号。它包括输入触发信号（如：重新加载、启动、停止、计数和捕获）和输出信号（如：上溢（OV），下溢（UN）和捕获 / 比较（CC））。任意 GPIO 都可作为输入触发信号使用。
- 中断：根据终止计数（TC）或 CC 条件提供每个计数器的中断请求信号，以及由八个中断请求信号经过逻辑 OR 后生成的组合中断信号。
- 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。有关该模块所需的所有寄存器的详细信息，请参见第 209 页上的 TCPWM 寄存器。

18.2.1 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，先对它进行配置。如果该计数器在配置后被使能，则寄存器将被更新为新的数值。禁用计数器后，寄存器中的值仍被保留，直到再次使能（或重新配置）该计数器为止。状态寄存器在计数器被禁用后均被清零。

18.2.2 时钟

TCPWM 通过系统接口接收 HFCLK，用于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，输出触发信号（本节中后面介绍）也与 HFCLK 同步。

时钟预分频：可对 counter_clock 进行预分频，分频值可为 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 18-1 所示。

表 18-1. 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：在正交模式和带死区时间的脉宽调制模式（PWM-DT）下，不能对时钟进行预分频。

18.2.3 基于触发输入的事件

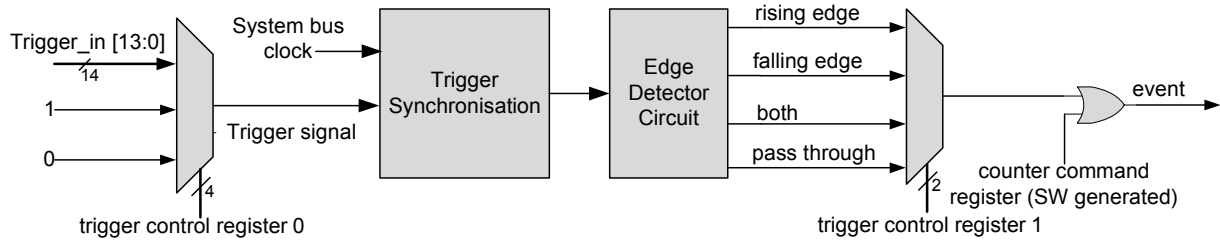
这些事件可由硬件或软件触发。

- 重载
- 启动
- 停止
- 计数
- 捕获 / 切换

硬件触发可以是电平信号、上升沿、下降沿或双边沿。图 18-2 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) 选择十四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’ 信号作为触发事件信号使用。

通过配置触发控制寄存器 1 (TCPWM_CNT_TR_CTRL1)，可以选择边沿 (上升沿、下降沿或双边沿) 或电平 (高电平或低电平) 来触发事件。可分别为每个触发事件选择边沿 / 电平配置。另外，通过写入计数器指令寄存器 (TCPWM_CMD)，固件可以生成一个事件，如图 18-2 所示。

图 18-2. TCPWM 触发选项和事件检测



在 TCPWM 模块的各种模式下，由这些触发源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。
 - 在递增计数模式下，计数寄存器 (TCPWM_CNT_COUNTER) 被初始化为 ‘0’。
 - 在递减计数模式下，计数器使用存储在 TCPWM_CNT_PERIOD 寄存器中的周期值进行初始化。
 - 在递增 / 递减计数模式下，计数寄存器均被初始化为 ‘1’。
 - 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操作；在发生停止事件后，或在软件将计数寄存器重新初始化为某个值后，可使用该事件。请注意，发生该事件时，计数寄存器不会被初始化。
 - 在正交模式下，启动事件作为正交相位输入 phiB，这在第 198 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，计数事件使它进行递增或递减。
 - 在正交模式下，计数事件作为正交相位输入 phiA。
- **停止：** 停止事件用于停止计数器的递增或递减。启动事件会再次启动计数操作。
 - 在脉宽调制模式中，停止事件作为禁止输出事件。禁止事件会禁用所有 PWM 输出信号线。
- **捕获：** 捕获事件会将计数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冲捕获寄存器内。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的缓冲寄存器内的相应数值。通过该特性，可调制脉冲的宽度和频率。

注意：

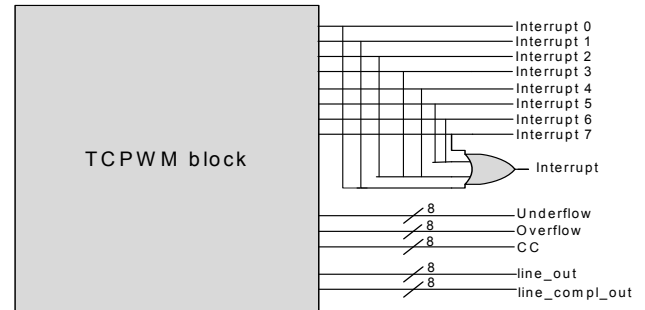
- 所有触发输入会与 HFCLK 同步。

- 在正交模式下，边沿检测由计数器时钟执行的。在其他 5 种模式下，可以使用 HFCLK 门控进行边沿检测。

18.2.4 输出信号

TCPWM 模块生成输出信号，如图 18-3 所示。

图 18-3. TCPWM 输出信号



18.2.4.1 发生触发事件时的信号

- 当计数器进行递增，并且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。
- 当计数器进行递减，并且计数寄存器达到零值时，计数器会生成内部下溢出 (UN) 状态。
- 当计数器正在运行并发生下面任何一个条件时，TCPWM 都会生成捕获 / 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冲捕获寄存器内。

注意： 这些信号在发生时，将在两个系统时钟周期内保持为逻辑高电平。为了保证运行过程可靠，引起该触发事件的信号频率应该小于 HFCLK 频率的四分之一。例如，如果 HFCLK 的工作频率为 24 MHz，则触发事件的信号频率需要小于 6 MHz。

18.2.4.2 中断

TCPWM 模块提供一个计数器的专用中断输出信号。在发生 TC 条件或 CC 条件时会生成中断。这些条件的准确定义因特定模式不同而异。对来自八个 TCPWMs 的所有八个中断输出信号进行“OR”运算结合，以生产单一中断输出信号。

四个寄存器用于该模块中的中断处理，如表 18-2 所示。

表 18-2. 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’。写入‘1’可清零该位。
	1	CC_MATCH	当计数值与捕获/比较寄存器中的值相匹配时，该位将被置为‘1’。写入‘1’可清零该位。
TCPWM_CNT_INTR_SET (中断设置请求寄存器)	0	TC	写‘1’可以设置中断请求寄存器中的相对位。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相对位。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	相对应的 TC 请求和屏蔽位的逻辑 AND 运算。
	1	CC_MATCH	相对应的 CC_MATCH 请求和屏蔽位的逻辑 AND 运算。

18.2.4.3 输出

TCPWM 有两个输出信号，line_out 和 line_compl_out (line_out 的互补)。请注意，如需要，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 18-3)。line_out 和 line_compl_out 分别由 line_out_en 和 line_compl_out_en 使能，每个计数器有一个。

表 18-3. OV、UN 和 CC 条件的输出信号线配置

字段	位	数值	事件	说明
CC_MATCH_MODE 默认值为 3	1:0	0	将 line_out 设置为‘1’	在发生比较匹配 (CC) 事件时配置输出信号线
		1	将 line_out 清零	
		2	反转 line_out	
		3	无变化	
OVERFLOW_MODE 默认值为 3	3:2	0	将 line_out 设置为‘1’	在发生上溢 (OV) 事件时配置输出信号线
		1	将 line_out 清零	
		2	反转 line_out	
		3	无变化	
UNDERFLOW_MODE 默认值为 3	5:4	0	将 line_out 设置为‘1’	在发生下溢 (UN) 事件时配置输出信号线
		1	将 line_out 清零	
		2	反转 line_out	
		3	无变化	

18.2.5 功耗模式

TCPWM 模块可在活动模式和睡眠模式下工作。TCPWM 模块由 V_{CCD} 供电。在深度睡眠模式下，配置寄存器和其他逻辑仍被供电，以保持配置寄存器的状态。更多详细信息，请参考表 18-4。

表 18-4. TCPWM 模块的功耗模式

功耗模式	模块状态
活动	在活动模式下，正常给该模块供电，时钟正常运行，支持模块的所有操作。
睡眠	所有计数器时钟都存在，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，逻辑因此无法工作。所有配置寄存器将保持其状态。
休眠	在这种模式下，该模块的电源被关闭。不能保持配置寄存器的状态。
停止模式	在这种模式下，该模块的电源被关闭。不能保持配置寄存器的状态。

18.3 各种操作模式

计数器模块可在六种操作模式下工作，如表 18-5 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 18-5. 工作模式配置

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，计数器就加 ‘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，计数器就加 ‘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。执行正交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冲比较 / 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）和缓冲比较 / 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产生一个伪随机 PWM。

通过设置 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_MODE[17:16] 字段，可将计数器配置为递增、递减或递增 / 递减计数模式，如表 18-6 所示。

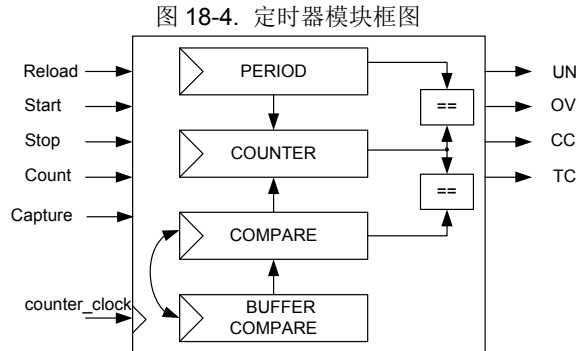
表 18-6. 计数模式配置

计数模式	UP_DOWN_MODE[17:16]	说明
递增计数模式	00	计数器递增，直到周期值为止。当计数器达到周期值时，将生成终值计数（TC）状态。
DOWN 计数模式	01	计数器从周期值开始递减，直到数值等于 0 为止。当计数器的值为 0 时，将生成 TC 状态。
递增 / 递减计数模式 0	10	计数器递增，达到周期值时开始递减，直到达到 ‘0’ 为止。只有达到 ‘0’，才生成 TC 状态。
递增 / 递减计数模式 1	11	与递增 / 递减计数模式 0 相似，但在计数器达到 ‘0’ 和计数器达到周期值时，都会生成 TC 状态。

18.3.1 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。

18.3.1.1 框图



18.3.1.2 工作原理

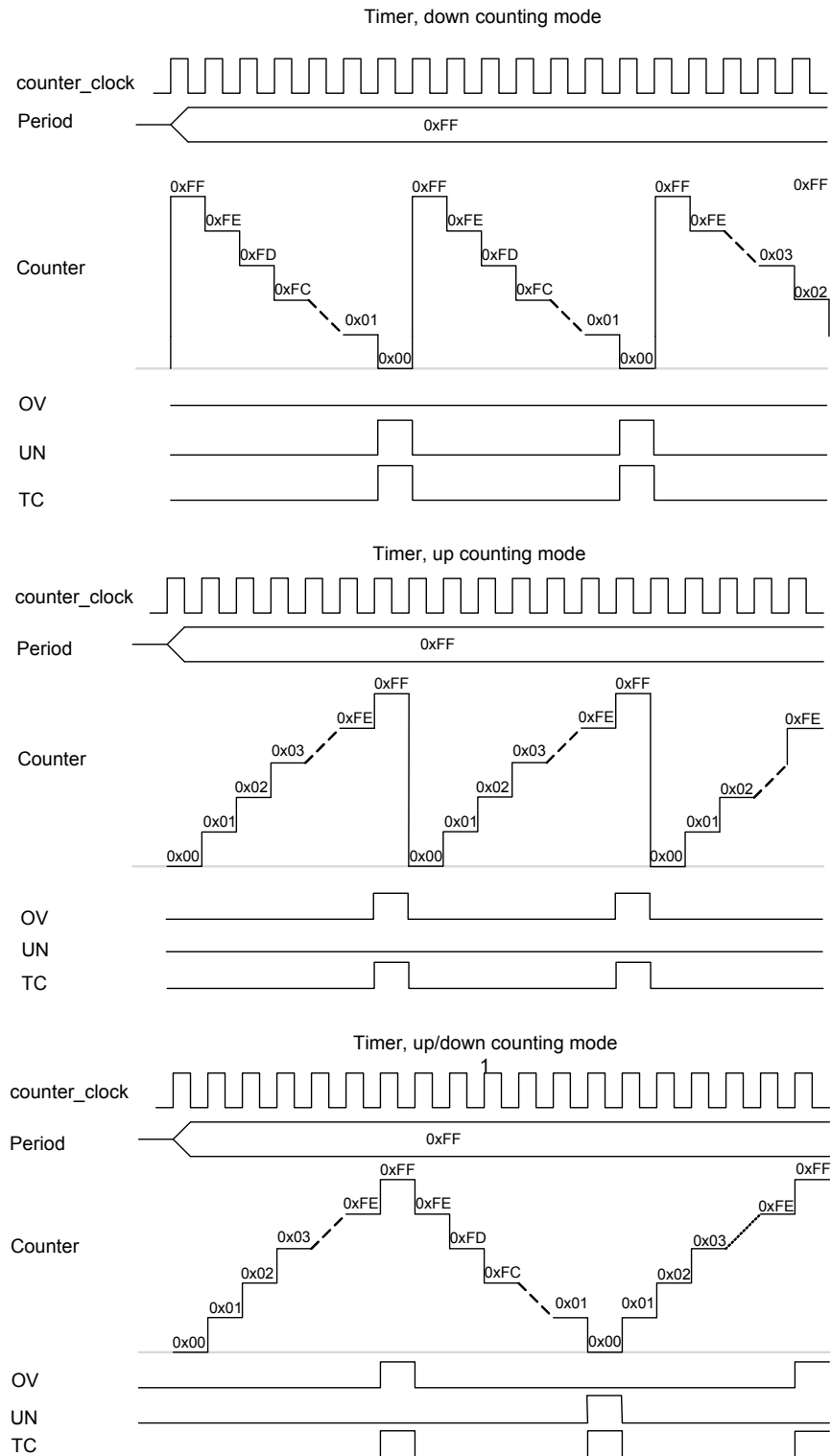
可配置定时器计数，以能够在递增、递减或递增 / 递减计数模式下进行计数。也可以将该模块配置为连续模式或单触发模式。下面介绍的是定时器的工作原理：

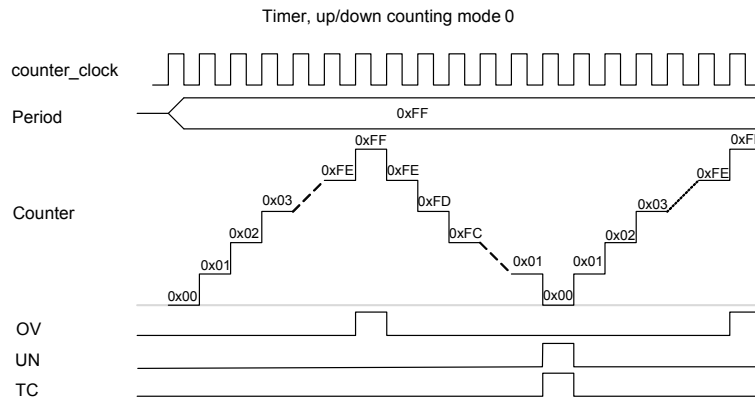
- 定时器是一个递增、递减和递增 / 递减计数器。
 - 当前的计数值被存储在计数寄存器（TCPWM_CNTx_COUNTER）内。
注意： 在计数器运行过程中不能对该寄存器进行写操作。
 - 定时器的周期值被存储在周期寄存器内。
- 可在不同的计数模式下重新初始化计数器：
 - 在递增计数模式下，计数达到周期值之后，计数寄存器会自动重新加载 ‘0’。
 - 在递减计数模式下，计数寄存器达到零之后，计数寄存器会重新加载周期寄存器中的值。
 - 在递增 / 递减计数模式下，计数寄存器达到终值时，它的值不被更新。当计数值等于零或周期值时，计数方向才发生改变。
- 当计数寄存器中的值等于比较寄存器中的值时，会生成 CC 条件。发生该条件时，如果通过计数器控制（TCPWM_CNT_CTRL）寄存器中的 AUTO_RELOAD_CC 位字段使能了比较寄存器和缓冲比较寄存器，则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 18-5 显示的是计数器在四种不同的计数模式下的定时器操作模式。周期寄存器包含最大的计数器值。

- 在递增计数模式下，A 的周期值会导致 A+1 个计数器周期（0 到 A）。
- 在递减计数模式中，A 的周期值会导致 A+1 个计数器周期（A 到 0）。
- 在两种递增 / 递减计数模式（模式 0 和 1）中，A 的周期值需要 2*A 个计数器周期（1 到 A 并 A 回至 0）。

图 18-5. 各种计数模式下的定时器时序图





注意：OV 和 UN 信号在 HFCLK 的一个周期内保持为逻辑高电平状态，如第 191 页上的发生触发事件时的信号所示。本节中的所有图表都假设 HFCLK 与计数器时钟相同。

18.3.1.3 配置定时器模式的计数器

下面各步骤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。

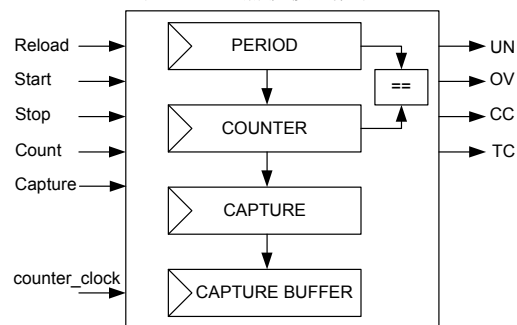
1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段的相对位写入零，可以禁用该计数器。
2. 通过向 TCPWM_CNT_CTRL 寄存器中的 MODE[26:24] 字段写入 '000' 选择 PWM 模式。
3. 设置 TCPWM_CNT_PERIOD 寄存器中所需要的 16 位周期。
4. 在 TCPWM_CNT_CC 寄存器中设置 16 位比较值，并在 TCPWM_CNT_CC_BUFF 寄存器中设置缓冲器比较值。
5. 如果需要每次发生 CC 条件时切换寄存器的值，请设置 TCPWM_CNT_CTRL 寄存器中的 AUTO_RELOAD_CC 字段。
6. 通过对 TCPWM_CNT_CTRL 寄存器中的 GENERIC[15:8] 字段进行写操作，设置时钟预分频，如表 18-1 所示。
7. 通过对 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_MODE[17:16] 字段进行写操作，设置计数的方向，如表 18-6 所示。
8. 分别将 0 和 1 写入到 TCPWM_CNT_CTRL 寄存器的 ONE_SHOT[18] 字段，以分别将定时器配置为连续模式或单触发模式。
9. 设置 TCPWM_CNT_TR_CTRL0 寄存器以选择引起发生事件（重载、启动、停止、捕获和计数）的触发器。
10. 设置 TCPWM_CNT_TR_CTRL1 寄存器以选择引起发生事件（重载、启动、停止、捕获和计数）的边沿触发。
11. 根据需求，根据 TC 或 CC 条件设置中断，如第 192 页上的中断所示。
12. 通过向 TCPWM_CTRL 寄存器 COUNTER_ENABLED 字段的相对位写入 '1'，可以使能该计数器。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。

18.3.2 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信号，都可随时捕获计数器的值。该模式用于测量周期和脉宽。

18.3.2.1 框图

图 18-6. 捕获模式框图



18.3.2.2 工作原理

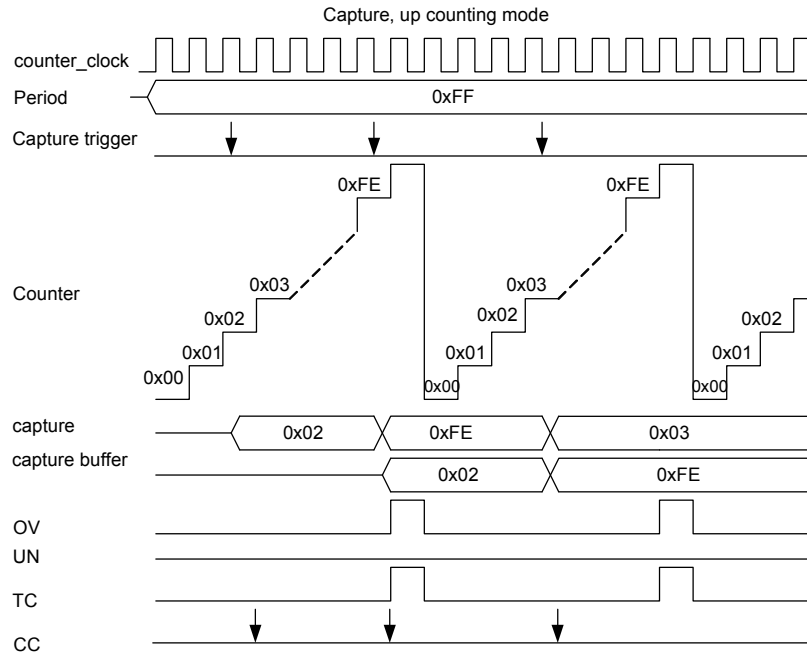
通过配置计数器控制（TCPWM_CNT_CTRL）寄存器中的 UP_DOWN_MODE[17:16] 位字段，可将计数器设置为递增、递减或递增 / 递减计数模式。

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捕获寄存器（TCPWM_CNT_CC_BUFF）内。
- 当计数器的值被复制到捕获寄存器内时，CC 输出信号上将生成一个脉冲。此条件还可用于生成中断请求。

图 18-7 显示了在递增计数模式下的捕获行为。

图 18-7. 在递增计数捕获模式下的计数器时序图



在该图中可以观察到：

- 周期寄存器包含最大的计数值。
- 当计数器达到周期值时，会生成内部上溢（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。

18.3.2.3 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下的计数器以及受影响的寄存器位。

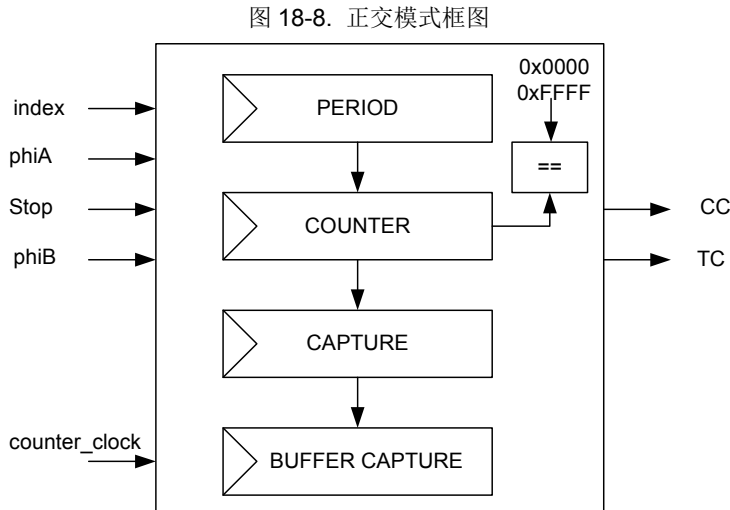
1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段的相对位写入零，可以禁用该计数器。
2. 通过将 ‘010’ 写入到 TCPWM_CNT_CTRL 寄存器的 MODE[26:24] 字段内选择捕获模式。
3. 设置 TCPWM_CNT_PERIOD 寄存器中所需要的 16 位周期值。
4. 通过对 TCPWM_CNT_CTRL 寄存器中的 GENERIC[15:8] 字段进行写操作，设置时钟预分频，如表 18-1 所示。
5. 通过对 TCPWM_CNT_CTRL 寄存器中的 UP_DOWN_MODE[17:16] 字段进行写操作，设置计数的方向，如表 18-6 所示。

6. 分别将 0 和 1 写入到 TCPWM_CNT_CTRL 寄存器中的 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 寄存器以选择引起发生事件（重载、启动、停止、捕获和计数）的触发器。
8. 设置 TCPWM_CNT_TR_CTRL1 寄存器以选择引起发生事件（重载、启动、停止、捕获和计数）的边沿。
9. 如果需要，根据 TC 或 CC 条件设置中断，如第 192 页上的中断所示。
10. 通过向 TCPWM_CTRL 寄存器 COUNTER_ENABLED 字段的相对位写入 ‘1’，可以使能该计数器。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。

18.3.3 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。

18.3.3.1 框图



18.3.3.2 工作原理

只能在 counter_clock 中执行正交解码。该操作可在 X1、X2 和 X4 等三种子模式下执行。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 计数方向可由 phiA 和 phiB 间的相位关系确定。这些相位分别连接至计数和启动触发输入，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，作为硬件输入使用。该事件会生成 TC 状态，如 图 18-9 所示。

发生 TC 条件时，计数器将被设置为 0x0000（在递增计数模式下）或被设置为周期值（在递减计数模式下）。

注意：在递减计数模式下，建议使用周期值（中点值）为 0x8000。

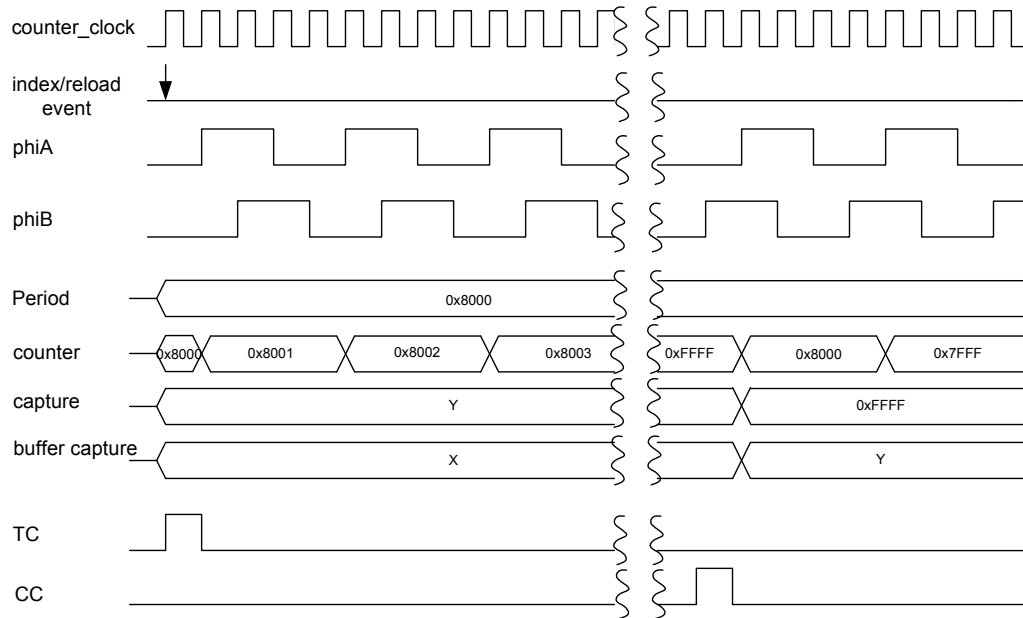
- 当计数寄存器值达到 0x0000 或 0xFFFF 时，会在 CC 输出信号上生成一个脉冲。在 CC 的情况下，计数寄存器的值被设置为 0x8000。
- 在 TC 或 CC 的情况下：
 - 计数寄存器的值被复制到捕获寄存器中
 - 捕获寄存器的值被复制到缓冲捕获寄存器中
 - 可以使用该条件生成中断请求
- 使用捕获寄存器中的值可确定引起发生事件的条件，并确定是否：
 - 发生计数器下溢（该值为 0）
 - 发生计数器上溢（该值为 0xFFFF）
 - 发生索引 / TC 事件（该值为非 0 或非 0xFFFF）

- 通过读取计数器状态 (TCPWM_CNTx_STATUS) 寄存器中的 DOWN 位字段，可以确定计数的当前方向。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 18-9 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 值（最大计数寄存器值）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 18-9. X1 正交解码模式的时序图

Quadrature, X1 encoding

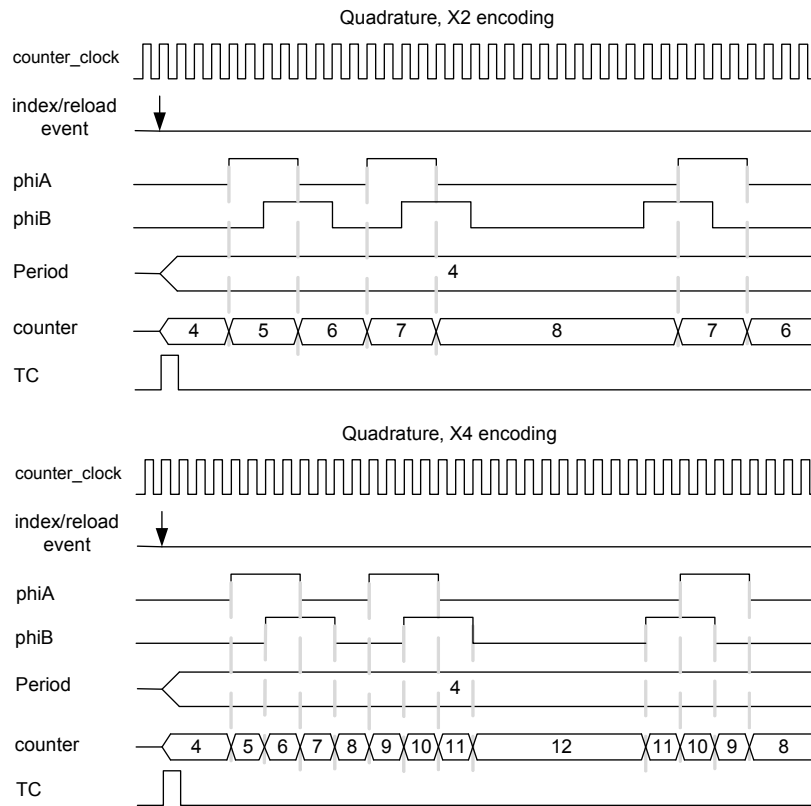


可在 **counter_clock** 上检测正交相位。在单一的 **counter_clock** 周期内，只应该对相位的值进行一次修改。

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 18-10 说明了 **X2** 和 **X4** 解码模式下的正交模式状况。

图 18-10. X2 和 X4 正交解码模式的时序图



18.3.3.3 配置正交模式的计数器

下面各步骤介绍的是配置正交模式下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 字段的相对应位写入零，可以禁用该计数器。
2. 通过向 TCPWM_CNT_CTRL 寄存器中的 MODE[26:24] 字段写入 ‘011’，可以选择正交模式。
3. 设置 TCPWM_CNT_PERIOD 寄存器中所需要的 16 位周期值。
4. 对 TCPWM_CNT_CTRL 寄存器中的 QUADRATURE_MODE[21:20] 字段进行写操作，以便设置所需要的编码模式。
5. 设置 TCPWM_CNT_TR_CTRL0 寄存器可以选择引起发生事件（索引和停止）的触发器。
6. 设置 TCPWM_CNT_TR_CTRL1 寄存器可以选择引起发生事件（索引和停止）的边沿。
7. 如果需要，根据 TC 或 CC 条件设置中断，如第 192 页上的中断 所示。
8. 通过向 TCPWM_CTRL 寄存器 COUNTER_ENABLED 字段的相对应位写入 ‘1’，可以使能该计数器。

18.3.4 脉宽调制模式

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，而其占空比则取决于比较和周期寄存器的值。

在左对齐和右对齐模式下：PWM 周期 = (周期值 / 计数器时钟频率)

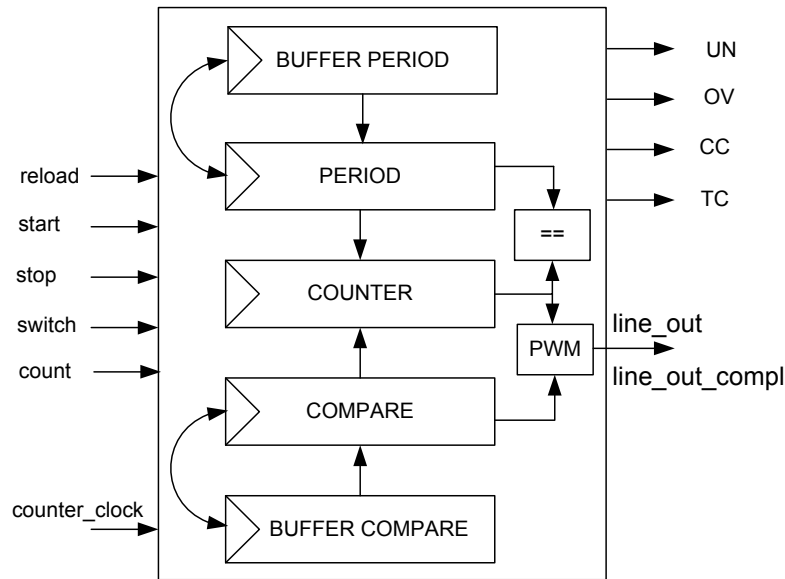
在中心对齐模式下：PWM 周期 = (2 × (周期值 / 计数器时钟频率))

在左对齐和右对齐模式下：占空比 = (比较值 / 周期值)

在中心对齐模式下：占空比 = ((周期值 - 比较值) / 周期值)

18.3.4.1 框图

图 18-11. PWM 模式框图



18.3.4.2 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 18-6 所示。

CC、OV 和 UN 信号均能够控制 PWM 输出信号线。这些信号可以翻转输出信号线，或可以通过配置 TCPWM_CNT_TR_CTRL2 寄存器将该信号线设置为逻辑 ‘0’ 或逻辑 ‘1’。通过配置信号影响输出信号线的方式，可以得到所需要的 PWM 输出对齐。

修改占空比的推荐方法包括：

- 使用新值更新缓冲周期寄存器和缓冲比较寄存器。
- 在 TC 条件下，如果发生了某个有效的切换事件，周期寄存器和比较寄存器将自动被缓冲周期寄存器和缓冲比较寄存器更新。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 事件前，需要通过有效的切换事件完成对缓冲周期寄存器和缓冲比较寄存器的更新；否则，切换操作不会影响寄存器的更新，如 图 18-13 所示。

在中心对齐模式下，需要进行的操作为：下溢 = 清除，上溢 = 设置，CC = 反转。

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 18-12 显示了有缓冲周期寄存器和比较寄存器的中心对齐脉宽调制（递增 / 递减计数模式 0）。

图 18-12. 中心对齐 PWM 的时序图

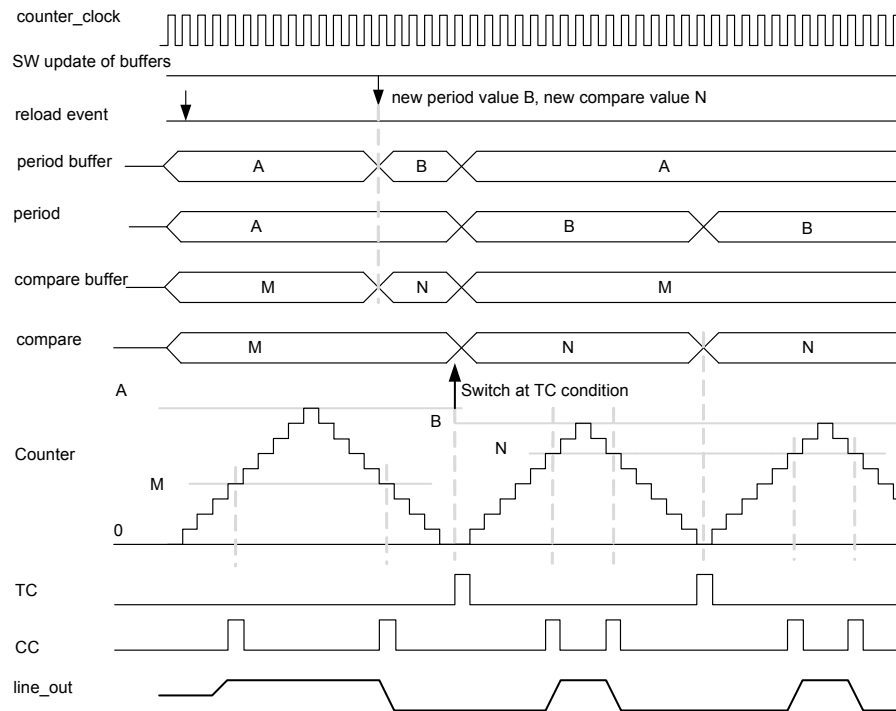
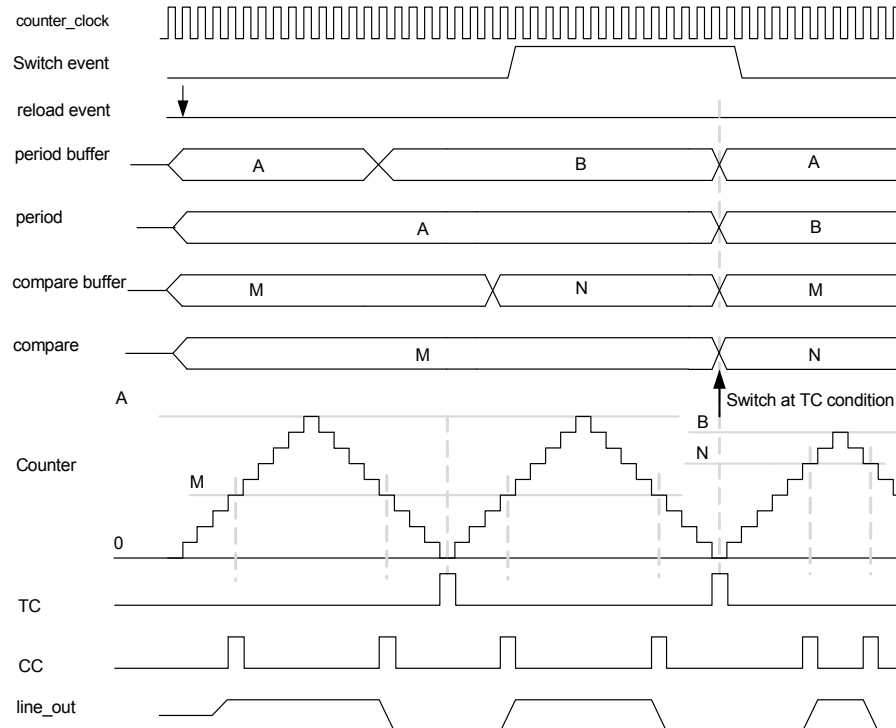


图 18-12 显示的是中心对齐 PWM，其中切换事件是由软件生成的：

- 仅在更新了周期缓冲寄存器 and 比较缓冲寄存器后，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 脉冲将被重复。
- 请注意，切换事件完成后，硬件将在 TC 条件下自动取消该事件。

图 18-13. 中心对齐脉宽调制的时序图（软件切换事件）



18.3.4.3 其他配置

- 对于不对称PWM，应使用递增/递减计数模式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。确保周期值在一个PWM周期内保持不变。
- 对于左对齐PWM，使用递增计数模式；配置OV条件以将输出信号线设置为‘1’并配置CC条件以将输出信号线复位为‘0’。请参见表18-3。
- 对于右对齐PWM，使用递减计数模式；配置UN条件以将输出信号线复位为‘0’并配置CC条件以将输出信号线设置为‘1’。请参见表18-3。

18.3.4.4 终止（kill）特性

使用终止（Kill）特性，可以立即禁用两个输出信号线。可以编程该事件，从而，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18-7所示。

表 18-7. 终止（Kill）功能的停止字段设置

PWM_STOP_ON_KILL 字段	注释
0	终止（kill）事件暂时停止 PWM 输出信号线，但计数器仍然运行。
1	终止（kill）事件暂时停止 PWM 输出信号线，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18-8所示。

表 18-8. 同步 / 异步终止（Kill）的字段设置

PWM_SYNC_KILL 字段	注释
0	只要存在异步终止（kill）事件，它将始终有效。该事件需要“通过”模式。
1	同步终止（kill）事件禁用 PWM 输出信号线，直到发生下一个 TC 事件为止。该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18-8）。通过所生成的停止（stop）事件可以禁用两个输出信号线。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。

18.3.4.5 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。

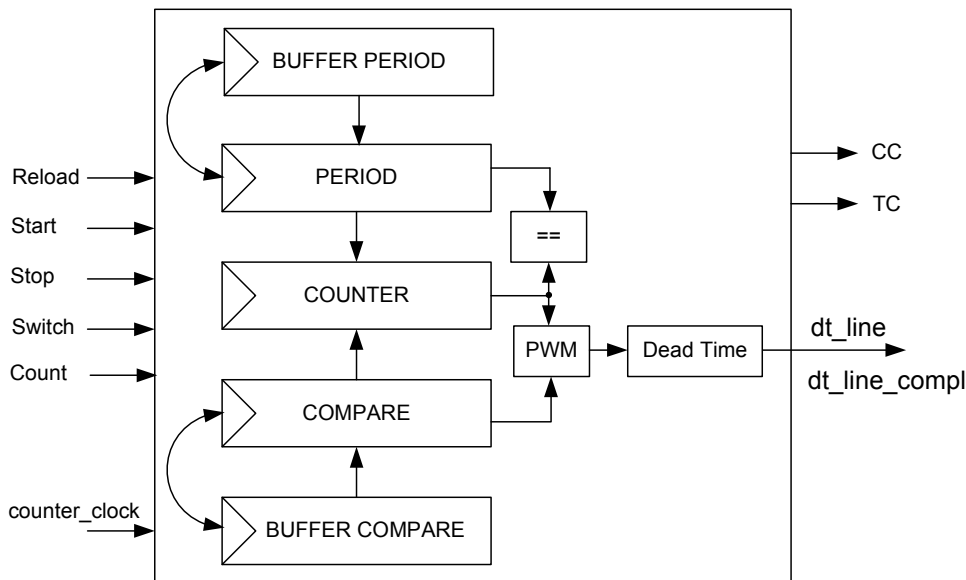
1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段的相对位写入零，可以禁用该计数器。
2. 通过向 TCPWM_CNT_CTRL 寄存器中的 MODE[26:24] 字段内写入‘100’选择PWM模式。
3. 通过对 TCPWM_CNT_CTRL 寄存器中的 GENERIC[15:8] 字段进行写操作，设置时钟预分频，如表18-1所示。
4. 根据需求，分别将 TCPWM_CNT_PERIOD 寄存器中的16位周期值和 TCPWM_CNT_PERIOD_BUFF 寄存器中的缓冲周期值设置为切换值。
5. 根据需求，分别将 TCPWM_CNT_CC 寄存器中的16位比较值和 TCPWM_CNT_CC_BUFF 寄存器中的缓冲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，可将其配置为左对齐、右对齐或中心对齐的PWM，如表18-6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）、切换和计数）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. 根据需求，根据 TC 或 CC 条件设置中断，如第192页上的中断所示。
12. 通过向 TCPWM_CTRL 寄存器 COUNTER_ENABLED 字段的相对位写入‘1’，可以使能该计数器。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。

18.3.5 带死区时间模式的脉宽调制

死区时间用于延迟 ‘line_out’ 和 ‘line_out_compl’ 信号的转换。它通过特定的时间间隔分离这两个信号的转换边沿。两条互补输出信号线 ‘dt_line’ 和 ‘dt_line_compl’ 是由这两条信号线生成的。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。

18.3.5.1 框图

图 18-14. PWM-DT 模式框图



18.3.5.2 工作原理

带有死区时间模式的 PWM 操作如下介绍：

- 在 PWM line_out 的上升沿上，根据 UN、OV 和 CC 事件，死区时间模块将 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 的下降沿上，根据 UN、OV 和 CC 事件，死区时间模块将 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_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。

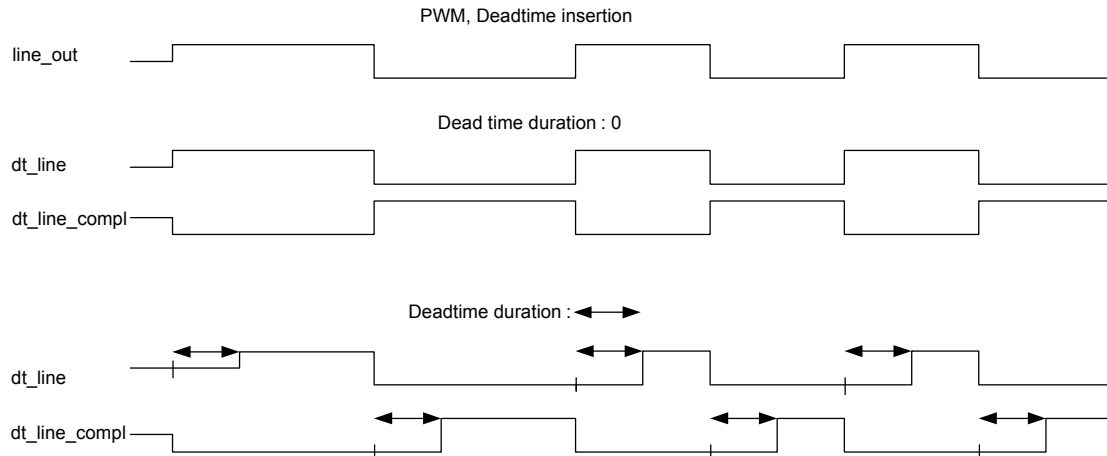
该模式遵循 PWM 模式并支持 PWM 模式的下述可用特性：

- 多种输出对齐模式
- 两个互补输出信号线（dt_line 和 dt_line_compl）分别由 PWM “line_out” 和 “line_out_compl” 生成
 - 同步模式和异步模式下的停止（Stop）/ 终止（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 18-15 下图显示了如何根据 PWM 输出信号线 “line_out” 生成互补输出信号线 “dt_line” 和 “dt_line_compl”。

图 18-15. 带和不带死区时间的 PWM 的时序图



18.3.5.3 配置 PWM（带死区时间）模式下的计数器

下面各步骤介绍的是配置 PWM（带死区时间）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 字段的相对应位写入零，可以禁用该计数器。
2. 通过将 ‘101’ 写入到 TCPWM_CNT_CTRL 寄存器中的 MODE[26:24] 字段选择带死区时间模式的 PWM。
3. 通过对 TCPWM_CNT_CTRL 寄存器中的 GENERIC[15:8] 字段进行写操作设置所需要的死区时间，如表 18-1 所示。
4. 根据需求，分别将 TCPWM_CNT_PERIOD 寄存器中的 16 位周期值和 TCPWM_CNT_PERIOD_BUFF 寄存器中的缓冲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 寄存器中的缓冲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，可将其配置为左对齐、右对齐或中心对齐的 PWM，如表 18-6 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，如第 201 页上的脉宽调制模式所示。
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）、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）、切换和计数）的边沿。
10. 通过 TCPWM_CNT_TR_CTRL2 寄存器控制 dt_line 和 dt_line_compl 在发生 CC、OV 和 UN 时进行置位、复位或翻转。
11. 根据需求，根据 TC 或 CC 条件设置中断，如第 192 页上的中断所示。

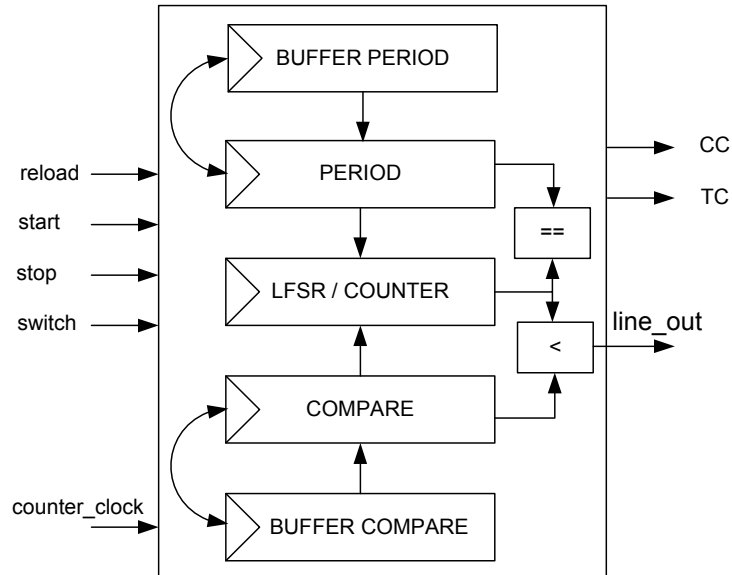
12. 通过向 TCPWM_CTRL 寄存器 COUNTER_ENABLED 字段的相对应位写入 ‘1’，可以使能该计数器。如果硬件启动信号未被使能，则固件（TCPWM_CMD 寄存器）必须提供一个启动触发信号启动计数器。

18.3.6 脉宽调制伪随机模式

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。

18.3.6.1 框图

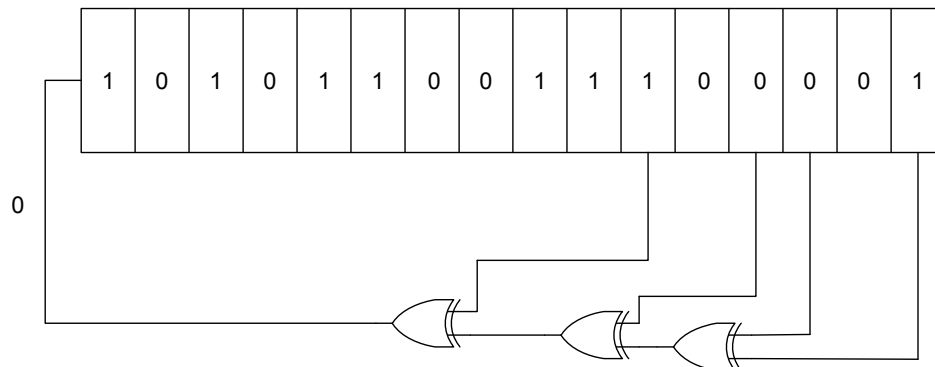
图 18-16. PWM-PR 模式的框图



18.3.6.2 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 18-17 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。

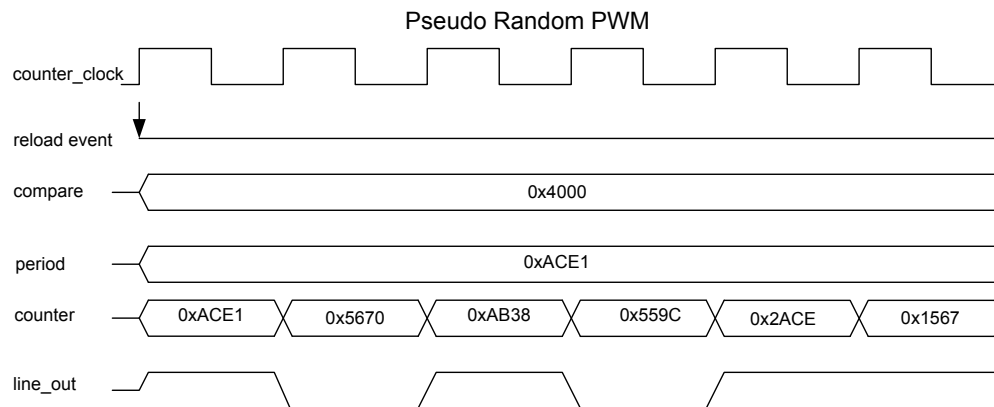
图 18-17. 使用计数器寄存器生成伪随机序列



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（如果计数器 [14:0] < 比较器 [15:0]）时，PWM 输出信号线（‘line_out’）将被驱动为 ‘1’。等于或大于 ‘0x8000’ 的比较值，都会使 PWM 输出信号线等于 ‘1’。如果比较值等于 ‘0’，则会使 PWM 输出信号线等于 ‘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 事件。
- 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、上溢和触发条件等事件。
- 当计数器运行，并且它的值等于比较值时，将发生 CC 事件。图 18-18 描述了伪随机噪声的状况。
- 如果比较值等于 0x4000，则能够生成 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 18-18. 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冲寄存器间的值，也可以切换周期寄存器和周期缓冲寄存器间的值。通过使用触发输入信号来控制调制操作，该功能可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。该输入信号被记录，直到到达下一个计数终值为止。

18.3.6.3 配置伪随机 PWM 模式的计数器

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

1. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED 字段的相对位写入零，可以禁用该计数器。
2. 通过向 TCPWM_CNT_CTRL 寄存器的 MODE[26:24] 字段写入 ‘110’ 可以选择伪随机 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 寄存器的缓冲周期值。
4. 为了切换各个值，分别设置 TCPWM_CNT_CC 寄存器的 16 位比较值和 TCPWM_CNT_CC_BUFF 寄存器的缓冲比较值。
5. 根据需求，设置 TCPWM_CNT_CTRL 寄存器的 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）和切换）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）和切换）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. 根据需求，根据 TC 或 CC 条件设置中断，如[第 192 页上的中断](#)所示。
10. 通过向 TCPWM_CTRL 寄存器 COUNTER_ENABLED 字段的相对位写入 ‘1’，可以使能该计数器。

18.4 TCPWM 寄存器

表 18-9. 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_CNTx_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、终止 (kill) 功能、死区时间、时钟预分频和计数方向
TCPWM_CNTx_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；检查计数器是否运行
TCPWM_CNTx_COUNTER	计数寄存器	包含 16 位计数器的值
TCPWM_CNTx_CC	计数器比较 / 捕获寄存器	捕获计数器的值或将需要比较的值与计数器的值进行比较
TCPWM_CNTx_CC_BUFF	计数器缓冲比较 / 捕获寄存器	计数器 CC 寄存器的缓冲寄存器；切换比较值
TCPWM_CNTx_PERIOD	计数器周期寄存器	包含计数器的上限值
TCPWM_CNTx_PERIOD_BUFF	计数器缓冲周期寄存器	计数器周期寄存器的缓冲寄存器；切换周期值
TCPWM_CNTx_TR_CTRL0	计数器触发控制寄存器 0	为特定计数器事件选择触发器
TCPWM_CNTx_TR_CTRL1	计数器触发控制寄存器 1	为特定计数器输入信号确定边沿检测
TCPWM_CNTx_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信号线
TCPWM_CNTx_INTR	中断请求寄存器	检测到 TC 或 CC 条件时设置该寄存器中的位
TCPWM_CNTx_INTR_SET	中断设置请求寄存器	设置中断请求寄存器中的相对位
TCPWM_CNTx_INTR_MASK	中断屏蔽寄存器	中断请求寄存器的屏蔽
TCPWM_CNTx_INTR_MASKED	中断屏蔽请求寄存器	中断请求和屏蔽寄存器的按位 AND 运算

节 E: 模拟系统

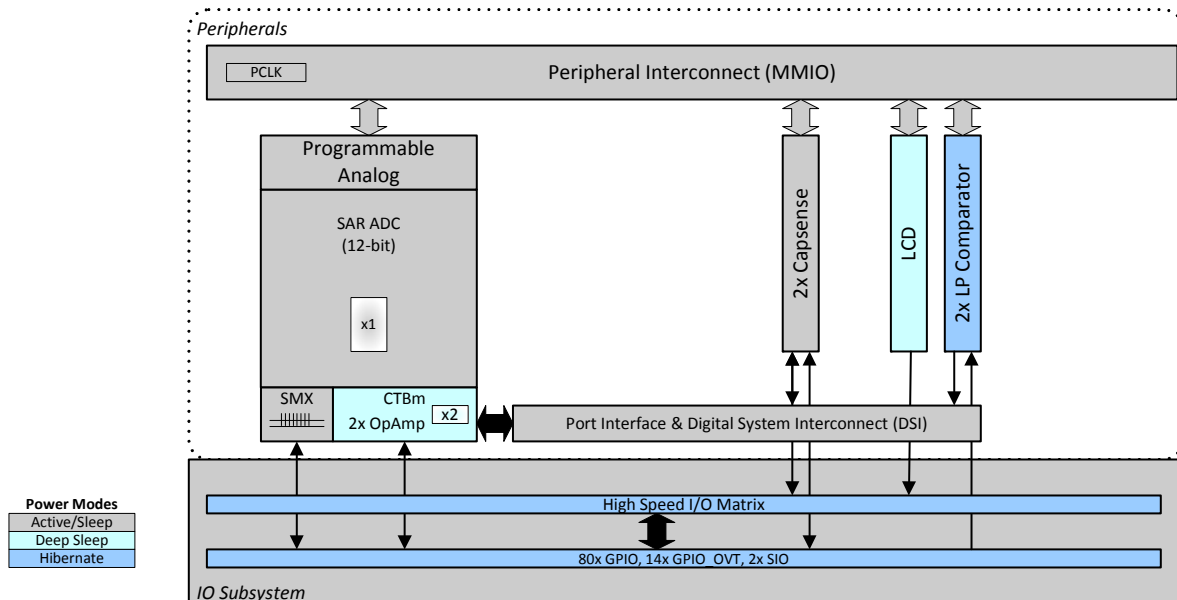


本章节包括以下小节：

- 第 213 页上的高精度参考章节
- 第 217 页上的 SAR ADC 章节
- 第 247 页上的低功耗比较器章节
- 第 253 页上的微型连续时间模块（CTBm）章节
- 第 263 页上的 LCD 直接驱动章节
- 第 275 页上的 CapSense 章节
- 第 285 页上的温度传感器章节

系统架构

模拟系统框图



20. 高精度参考



PSoC[®] 4 具有一个高精度参考电压模块，它可以为整个芯片创建多个参考用的偏置电压和电流。在器件温度范围内，该模块为内部主振荡器（IMO）电路和闪存模块提供了随温度变化的参考电压，以分别为它们保证准确的 IMO 输出频率和无错误的闪存读 / 写操作。

20.1 特性

高精度参考模块具有以下特性：

- 带隙电路，可生成 1.024 V 和 2.4 μ A 的参考电源
- 调整缓冲区，可从带隙电路的输入生成 1.2 V、1.024 V 和 0.8V 等多种输出电压
- 多个快速和慢速的低功耗缓冲区，它们不仅提高了各种参考输出的驱动能力，并且还可以将噪声彼此隔离开
- 多个快速和慢速的电流镜像电路
- 随温度变化的参考电压，用于闪存存储器
- 随温度变化的参考电流，用于 IMO

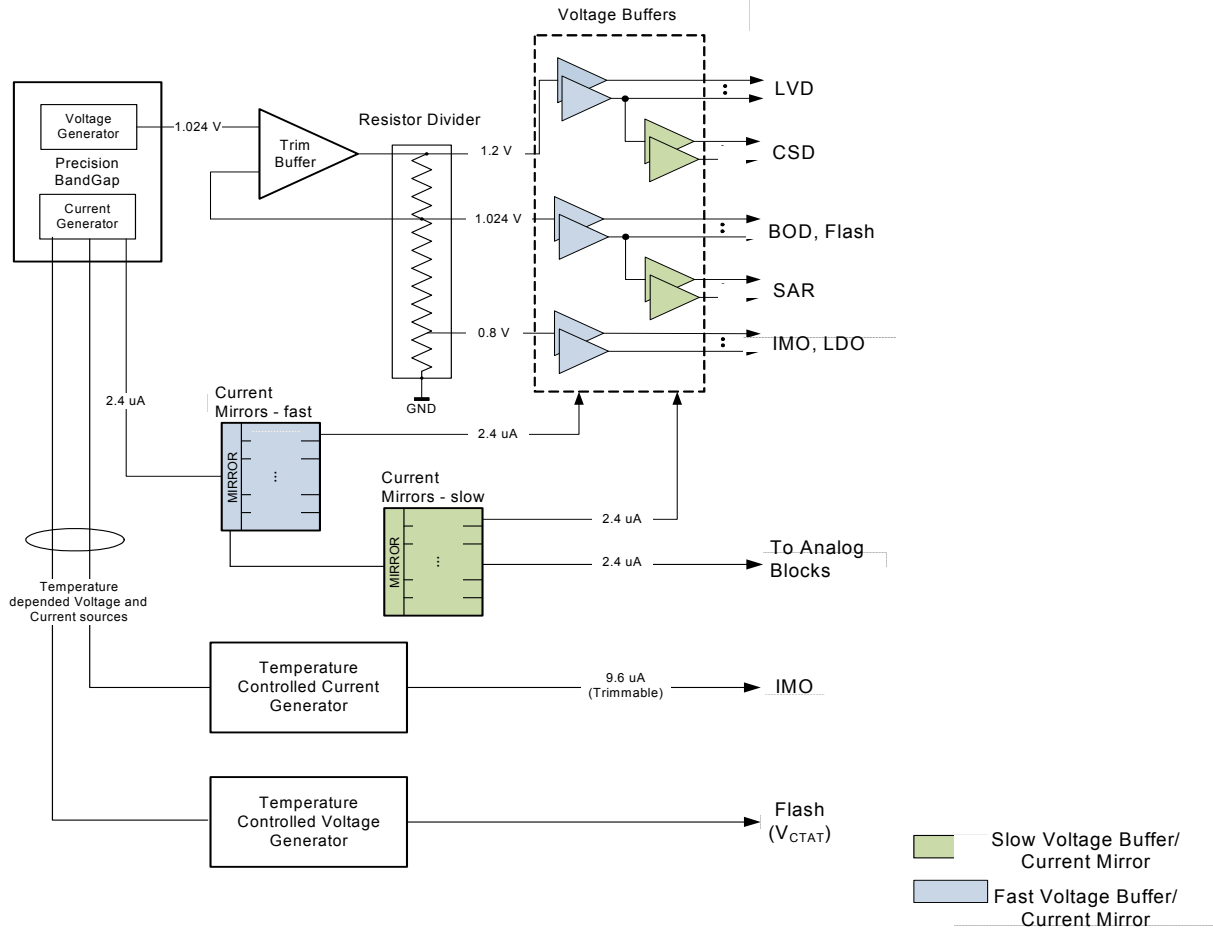
20.2 框图

图 20-1 显示了参考电压模块框图。

高精度参考电压主要包括以下模块：

- 一个高精度带隙模块，用于生成精确的参考电压和参考电流
- 一个调整缓冲区，它为各种应用程序生成不同的输出参考电压并调整 1.024 V 的输出电压
- 它是一组快速低功耗缓冲区和慢速低功耗缓冲区。它不仅提高了各种参考输出的驱动能力，而且还可以将噪声彼此隔离开
- 一组快速元件和慢速元件，可分别在快速域和慢速域中创建多个参考电流的副本
- 一个由温度控制的电压发生器模块，用于闪存系统
- 一个由温度控制的电流源，用于 IMO

图 20-1. 参考电压模块框图



20.3 工作原理

本节详细介绍了主要组件的工作原理。

20.3.1 高精度带隙

该电路是高精度参考模块所生成的所有参考电压的源。它提供了经过第二阶曲率纠正后的 1.024 V 参考电压和 $2.4\text{ }\mu\text{A}$ 的参考电流。参考电压被路由到调整缓冲区，而参考电流则被输送给电流镜像电路。

20.3.2 调整缓冲区

调整缓冲区是一个运算放大器网络，它从带隙电路中获取输入信号，然后生成三种不同的参考电压（分别为 1.2 V 、 1.024 V 和 0.8 V ）。从反馈网络的电阻阵列中输出这些参考电压，从而形成高输出阻抗。因此，必须使用各个缓冲区来驱动参考电压。

20.3.3 低功耗缓冲区

PSoC 4 具有多个低功耗缓冲区，并分为两组：快速和慢速。这些缓冲区从调整缓冲区电路获取输入，并驱动目标模块。经过 $9\text{ }\mu\text{s}$ 后，快速缓冲区可以达到最终值，差值为 1% 。而慢速缓冲区则需要经过 $40\text{ }\mu\text{s}$ 才能达到最终值。多缓冲区可以确保低参考线电容，从而降低建立时间。快速电压缓冲区用于输出给对系统启动过程起着重要作用的模块的参考电压。它们包括 IMO、闪存、低压差电压调节器（LDO）、低压检测（LVD）以及欠压检测（BOD）电路。

快速缓冲区的输出被驱动给慢速缓冲区。这样可以确保将相关未启动模块引起的额外负载与快速缓冲区驱动的模块隔离开。慢速缓冲区驱动各功能模块，如 SAR ADC 和 CapSense CSD。

快速缓冲区和带隙模块始终被使能，而用户通过使用 PWR_BG_CONFIG 寄存器中的 VREF_EN 位可以单独使能或禁用慢速缓冲区。

20.3.4 电流镜像

通过使用电流镜像电路可以从带隙电路中生成 **2.4 μ A** 参考电流的多个副本。与电压缓冲器相似，电流镜像也分为两种类型：快速和慢速。快速电流镜像电路经过 **9 μ s** 的建立时间可以达到最终电压值，差值为 **1%**；而慢速电流镜像电路则需要 **40 μ s** 才能达到。通过使用快速电流镜像可以为快速电压缓冲器提供偏置电压。慢速电流镜像的各个输出用于驱动各种模拟模块，如：SAR、CTBm、CSD 和 LPCOMP。

20.3.5 温度控制的电压发生器

由该模块生成的偏置信号会根据温度来控制闪存存储器的参考电压。它从高精度带隙模块中接收输入。在器件温度范围内，随温度变化的参考电压（ V_{CTAT} ）将补偿在闪存存储器模块中所生成的所需泵电压，从而正常执行读写操作。

20.3.6 温度控制的电流发生器

该模块为 IMO 生成随温度变化的参考电流，以便在器件的工作温度范围内保持它的时钟频率差值为 $\pm 2\%$ 。

表 20-1. 参考电压

参考电压	缓冲速度	说明
1.2 V	快速	LVD 模块的参考电压
1.2 V	慢速	USB 电压调节器在 5 V 模式下的参考电压
1.2 V	慢速	CapSense 模块的参考电压
1.024 V	快速	BOD 模块的参考电压
1.024 V	快速	闪存模块的偏置参考电压用于读取闪存内的数据
1.024 V	慢速	ECO 模块的参考电压，用于控制振幅
1.024 V	慢速	SAR ADC 模块的参考电压
0.8 V	快速	比较器阈值，适用于 IMO 中的张弛振荡器
0.8 V	快速	LDO 模块中 VCCD 和 VCCA 电压调节器的参考电压
VCTAT	快速	闪存正电压（VPOS）泵的参考电压，该参考电压会随温度不同而变化

表 20-2. 参考电流

参考电流	缓冲速度	说明
2.4 μ A	快速	USB、LCD 驱动、BOD 以及闪存模块的参考电流，并为快速电压缓冲器提供偏置电压
2.4 μ A	慢速	模拟模块（SAR、CapSense、IDAC、LPCOMP 和 CTBm）的参考电流，并为慢速电压缓冲器提供偏置电压
9.6 μ A	快速	IMO 模块的参考电流，支持基于温度的可编程补偿

20.4 配置

在上电过程中，通过使用保存在非易失性锁存器（NVL）和 SFLASH 中的默认调整设置初始化高精度参考模块。这些设置在制造过程中已经过编程，并不需要进行字段调整。

21. SAR ADC



PSoC® 4 具有一个逐次逼近寄存器模数转换器（SAR ADC）。SAR ADC 的设计适用于各种需要中等分辨率以及高数据速率的应用。它包含以下模块（请参见图 21-1）：

- SARMUX
- SAR ADC 内核
- SARREF
- SARSEQ

SAR ADC 内核是 PSoC 4200L 中 12 位分辨率和 1 Msps 采样率的快速 ADC。SAR ADC 模块的前面是 SARMUX，它可将外部引脚和内部信号（如 AMUXBUS-A/-B、CTBm、温度传感器输出）连接到 SAR ADC 的八个内部通道。SARREF 用于多个参考电压选择。定序器控制器 SARSEQ 用于控制 SARMUX 和 SAR ADC，从而自动扫描所有启用的通道（无需 CPU 的干预）并进行预处理（如求输出数据的平均值）。

第九个通道为插入通道，固件将其用于偶然和附带的引脚以及信号（如内部温度传感器）采样。

每个通道的结果都是双缓冲的（其结果被写入到两组寄存器上），并且会配置一个完整的扫描操作在扫描结束时生成中断。此外，数据还可传输到可编程数字模块（UDB）中以进一步处理而无需 CPU 的干预。也可以配置定序器来标志溢流、冲突和饱和等可触发中断的错误。

为了提供更大的灵活性，还可以通过使用 UDB 或固件来控制几乎所有模拟开关，其中包括 SARMUX 中的开关。这样使它能够通过 UDB 或固件来实现另一个定序器。

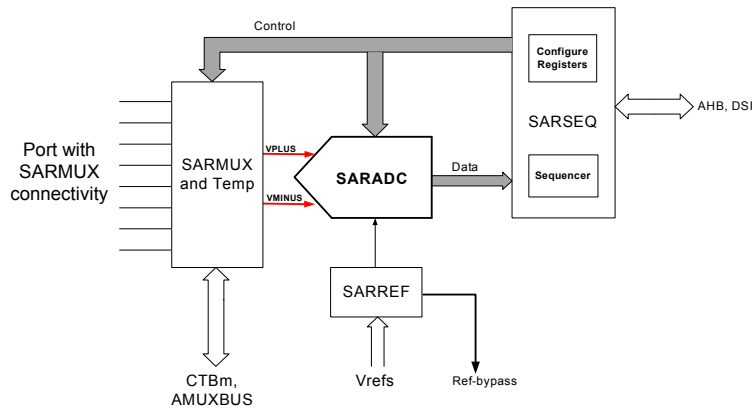
21.1 特性

- 可以在整个器件的供电范围内进行操作
- PSoC 4200L 中的最大采样率为 1 Msps
- 具有八个可独立配置通道及一个插入通道
- 每个通道具有以下特性：
 - 从外部引脚或内部信号输入（如 AMUXBUS/CTBm/ 温度传感器）
 - 可编程采集时间
 - 可选的 8 位、10 位和 12 位分辨率
 - 单端或差分测量
 - 求平均
 - 结果数据是双缓冲的
 - 可对结果数据进行左对齐或右对齐
- 由固件、定时器、引脚或 UDB 触发扫描
 - 单触发周期性或连续模式
- 支持硬件求平均
 - 第一阶累加
 - 对从 2 至 256 样本（2 的幂）进行求平均
- 结果表示为 16 位有符号的扩展值
- 可选的参考电压

- 内部参考电压 V_{DDA} 和 $V_{DDA}/2$
- 使用缓冲区的内部 1.024 V 参考电压
- 外部参考电压
- 中断生成
 - 完成了扫描转换操作
 - 每个通道的饱和检测和超出范围（可配置）检测
 - 扫描结果溢出
 - 冲突检测
- 可配置插入通道
 - 由固件触发
 - 可以在两个扫描序列之间交错进行（紧接）
 - 可选的采样时间、分辨率、单端或差分、平均值
- 用于处理可编程数字模块中的数据以减轻 CPU 负载的选项
- 用于控制可编程数字模块开关的选项
- 用于控制 SAR ADC 和可编程数字模块开关的选项
 - 实现一个备用 SAR 定序器
 - 能够实现 1 Msps 的采样率
- 低功耗模式
 - ADC 内核和参考电压具有专用的低功耗模式

21.2 框图

图 21-1. 框图



21.3 工作原理

本节将介绍了以下内容：

- SAR ADC 内核、SARMUX、SARREF 和 SARSEQ 等模块的简介
- SAR ADC 系统资源：中断、低功耗模式以及 SAR ADC 状态
- 系统操作模式
 - 寄存器模式
 - DSI 模式
- 配置示例

21.3.1 SAR ADC 内核

PSoC 4 SAR ADC 内核是一个 12 位 SAR ADC。该 ADC 的最大采样率为 1 Msps。SAR ADC 内核具有以下特性：

- 基于全差分架构模式，另外也支持单端模式
- 12 位分辨率和可选的备用分辨率（8 位或 10 位）
- 可编程采集时间
- 可编程功耗模式（100%、50%、25%）
- 支持单一和连续转换模式

21.3.1.1 单端模式和差分模式

PSoC 4 SAR ADC 可在单端模式和差分模式下运行。它的设计基于一个全差分架构，并得到优化，从而在差分操作模式下可提供 12 位的精度。它为处于 $-V_{REF}$ 到 $+V_{REF}$ 范围内的差分输入提供了全范围输出（0 至 4095）。通过确定负输入，可在单端模式下配置 SAR ADC。可以使用通道配置寄存器（SAR_CHANx_CONFIG）配置为差分模式或单端模式。

负输入的单端模式选项包括： V_{SSA} 、 V_{REF} 或已连接至 SARMUX 的八个引脚中任意引脚的外部输入。更多有关引脚的信息，请参见该器件的数据手册。该模式是由全局配置寄存器 SAR_CTRL 配置的。当 V_{minus} 被连接到这些 SARMUX 引脚时，该单端模式便相当于差分模式。然而，当每个差分对中的奇数引脚被连接到通用的备用接地引脚时，这些转换操作便为 11 位，因为测量的信号值（SARMUX.vplus）不能低于接地值。

为了实现分辨率为 12 位的单端转换，需要将 V_{REF} 连接至 SAR ADC 的负输入；这样输入的电电压范围为 0 到 $2 \times V_{ref}$ 。

请注意，仅在单端模式下才能使用温度传感器；它会覆盖 SAR_CTRL [11:9]，使之变为 0。差分转换不适用于温度传感器，因此它的结果是“未定义”的。

21.3.1.2 输入电压范围

所有输入都要处于从 V_{SSA} 至 V_{DDA} 的范围内。输入电压范围也受 V_{REF} 的限制。如果负输入上的电压为 V_n ，并且 ADC 参

考电压为 V_{REF} ，那么正输入的电压范围将为 $V_n \pm V_{REF}$ 。该标准适用于单端模式和差分模式。在单端模式下， V_n 被连接至 V_{SSA} 、 V_{REF} 或某个外部输入。

请注意： $V_n \pm V_{REF}$ 需要处于从 V_{SSA} 到 V_{DDA} 的范围内。例如，若负输入被连接到 V_{SSA} ，则正输入的电压范围将为 0 至 V_{REF} ，而不是 $-V_{REF}$ 至 V_{REF} 。这是因为该信号不可低于 V_{SSA} 。由于正输入信号的摆幅不能低于 V_{SS} （这样会引起只能生成一个 11 位的结果），因此只有 ADC 范围的一半是可用的。

21.3.1.3 结果数据格式

可从以下两个方面对结果数据格式进行配置：

- 有符号 / 无符号
- 左 / 右对齐

当结果为有符号时，转换操作的最高有效位将以符号扩展到 16 位，且扩展结果为 MSB。对于一个无符号的转换，使用零将结果扩展为 16 位。通过 SAR_SAMPLE_CTRL [3:2]，可以将结果数据格式分别配置为差分转换和单端转换。

采样值在结果寄存器的 16 位中可以是右对齐或左对齐。在默认情况下，data[11:0] 中的数据是右对齐的，并且被扩展为 16 位（若需要）。分辨率低与左对齐的结合会使较低有效位变为 0。

针对 12、10、8 位转换将有符号和无符号、左对齐和右对齐结合起来，结果数据格式可如下显示：

表 21-1. 结果数据格式

对齐	有符号 / 无符号	分辨率	结果寄存器															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
右对齐	无符号	12	–	–	–	–	11	10	9	8	7	6	5	4	3	2	1	0
		10	–	–	–	–	–	–	9	8	7	6	5	4	3	2	1	0
		8	–	–	–	–	–	–	–	–	7	6	5	4	3	2	1	0
右对齐	有符号	12	11	11	11	11	11	10	9	8	7	6	5	4	3	2	1	0
		10	9	9	9	9	9	9	9	8	7	6	5	4	3	2	1	0
		8	7	7	7	7	7	7	7	7	7	6	5	4	3	2	1	0
左对齐	–	12	11	10	9	8	7	6	5	4	3	2	1	0	–	–	–	–
		10	9	8	7	6	5	4	3	2	1	0	–	–	–	–	–	–
		8	7	6	5	4	3	2	1	0	–	–	–	–	–	–	–	–

21.3.1.4 负输入选择

负输入连接选择会影响电压范围、信噪比以及有效分辨率（请参考表 21-2）。在单端模式下，SAR ADC 的负输入可连接至 V_{SSA} 、 V_{REF} ，或连接至与 SARMUX 互连的八个引脚中任意一个引脚。

表 21-2. 负输入选择的比较

单端 / 差分	有符号 / 无符号	SARMUX V_{minus}	SARMUX V_{plus} 范围	结果寄存器	最大信噪比
单端	N/A ^a	V_{SSA}	$+V_{REF}$ $V_{SSA} = 0$	0x7FF 0x000	更好
单端	无符号	V_{REF}	$+2 \times V_{REF}$ V_{REF} $V_{SSA} = 0$	0xFFFF 0x800 0	良好
单端	有符号	V_{REF}	$+2 \times V_{REF}$ V_{REF} $V_{SSA} = 0$	0x7FF 0x000 0x800	良好
单端	无符号	V_x	$V_x + V_{REF}$ V_x $V_x - V_{REF}$	0xFFFF 0x800 0	最佳
单端	有符号	V_x	$V_x + V_{REF}$ V_x $V_x - V_{REF}$	0x7FF 0x000 0x800	最佳
差分	无符号	V_x	$V_x + V_{REF}$ V_x $V_x - V_{REF}$	0xFFFF 0x800 0	最佳
差分	有符号	V_x	$V_x + V_{REF}$ V_x $V_x - V_{REF}$	0x7FF 0x000 0x800	最佳

a. 在 V_{minus} 连接至 V_{SSA} 的单端模式下，因为在任何 PSoc 4 引脚上电压的摆幅不可低于 V_{SSA} ，转换的有效位数为 11 位。因此，全局配置位 SINGLE_END-ED_SIGNED (SAR_SAMPLE_CTRL[2]) 将被忽略，并且结果总是 (0x000-0x7FF)。

为了能够实现 12 位分辨率的单端转换，需要将 V_{REF} 连接至 SAR ADC 的负输入端，这样输入的电压范围便为 0 到 $2 \times V_{REF}$ 。

请注意， V_{minus} 连接到与 SARMUX 相连的引脚时，单端转换操作便为差分模式。然而，当每个差分对中的奇数引脚被连接到通用的备用接地引脚时，这些转换操作便为 11 位的，因为测量的信号值 (SARMUX.vplus) 不能低于接地值。

21.3.1.5 分辨率

PSoc 4 支持 12 位的分辨率（默认）以及一个可选的备用分辨率：每个通道可选 8 位或 10 位。

分辨率影响到转换时间：

$$\text{转换时间 (sar_clk)} = \text{分辨率 (位)} + 2$$

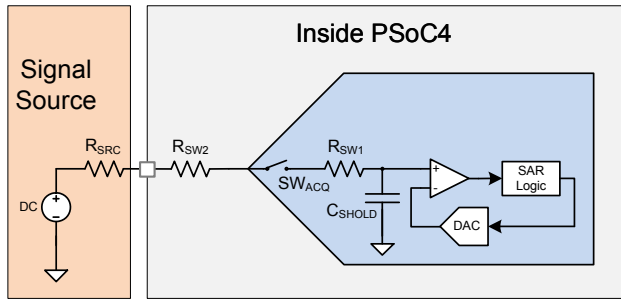
$$\text{采集和转换的总时间 (sar_clk)} = \text{采集时间} + \text{分辨率 (位)} + 2$$

12 位转换和采集时间为 4 时，sar_clk 需要等于 18。例如，如果 sar_clk 为 18 MHz，则需要使用 18 sar_clk 进行转换，此时转换速率将为 1 Msps。分辨率越低，转换速率越高。

21.3.1.6 采集时间

采集时间是指 SAR ADC 中采样以及保持 (S/H) 电路用于达到稳定状态的时间。采集时间后，输入信号源会断开与 SAR ADC 内核的连接，并使用 S/H 电路的输出进行转换。每个通道都可以选择四个采集时间选项中的一个，即在全局配置寄存器 SAR_SAMPLE_TIME01 和 SAR_SAMPLE_TIME23 中所定义的从 4 到 1023 的 SAR 时钟周期。

图 21-2. 采集时间



采集时间要足够长，以通过路由路径的电阻给 ADC 的内部保持电容充电，如图 21-2 所示。采集时间的推荐值为：

$$t_{ACQ} \geq 9 \times (R_{SRC} + R_{SW2} + R_{SW1}) \times C_{SHOLD}$$

其中：

$$C_{SHOLD} \approx 10 \text{ pF}$$

$R_{SW2} + R_{SW1} = \sim 500$ 至 1000Ω ，具体情况取决于路由路径（具体请参见第 222 页上的模拟路由）。

R_{SRC} = 信号源的串联电阻

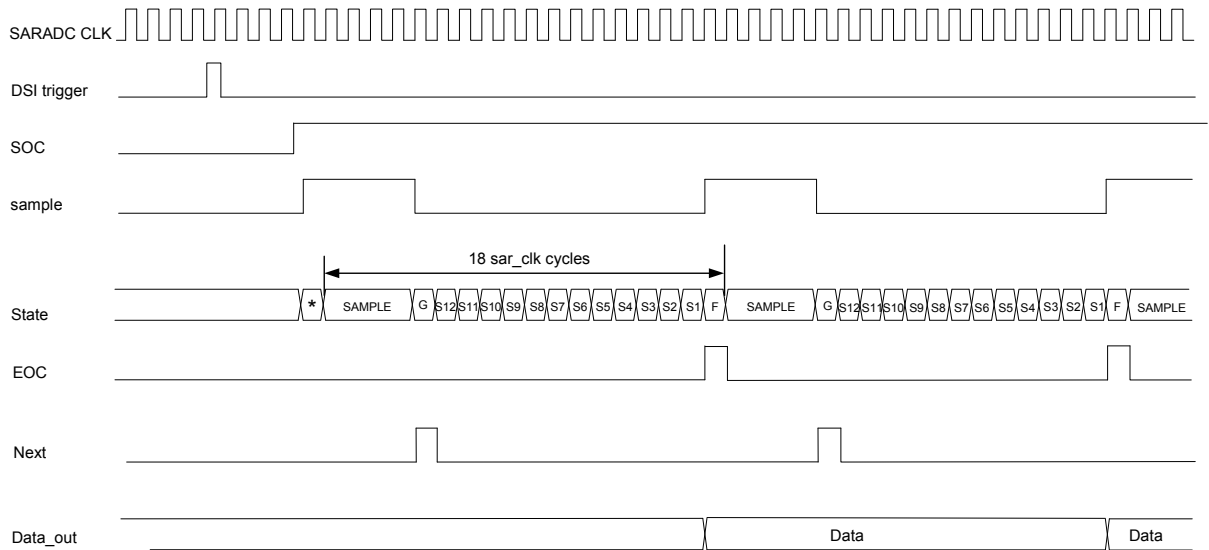
21.3.1.7 SAR ADC 时钟

PSoC 4200L 的 SAR ADC 时钟频率必须在 1 MHz 到 18 MHz 之间，需要对 HFCLK 时钟源进行时钟分频以获得所需频率范围。请注意，SAR ADC 不支持小数分频。要想在 PSoC 4200L 系列中获取 1 Msps 的采样率，需要使用一个 18 MHz 的 SAR ADC 时钟。此时，必须将系统时钟（HFCLK）设置为 36 MHz 而不是 48 MHz。当最小采集时间为 4 个时钟周期时（频率为 18 MHz），需要 18 个时钟周期来完成 12 位 ADC 转换操作。10 位和 8 位转换则分别需要 16 和 14 个时钟周期。请注意，频率为 18 MHz 时，最小采集时间为 4 个时钟周期，该数据是根据 SAR 模块（第 221 页上的图 21-2 中的 R_{SW1} 和 C_{SHOLD} ）所支持的最小采集时间（194 ns）得到的。

21.3.1.8 SAR ADC 时序

如图 21-3 所示，在生成 ‘开始转换’（SOC）前，会出现一个 `sar_clk` 延迟。12 位分辨率的转换需要 14 个时钟周期来完成（每一位需要一个 `sar_clk`，再加上处于 G 和 F 状态的两个超额 `sar_clk`）。当采集时间默认为 4 个 `sar_clk` 周期时，ADC 的采集及转换的总时间需要 18 个 `sar_clk` 时钟周期。进行采样（采集）后，它会输出下一个脉冲（或 `ds_i_sample_done`），并且 SARMUX 可连接至其它引脚和信号，通过定序器控制会自动进行该操作（具体信息，请参见第 229 页上的 SARSEQ）。

图 21-3. SAR ADC 时序



21.3.2 SARMUX

SARMUX 是一个专用的模拟可编程复用器。SARMUX 的主要特性如下：

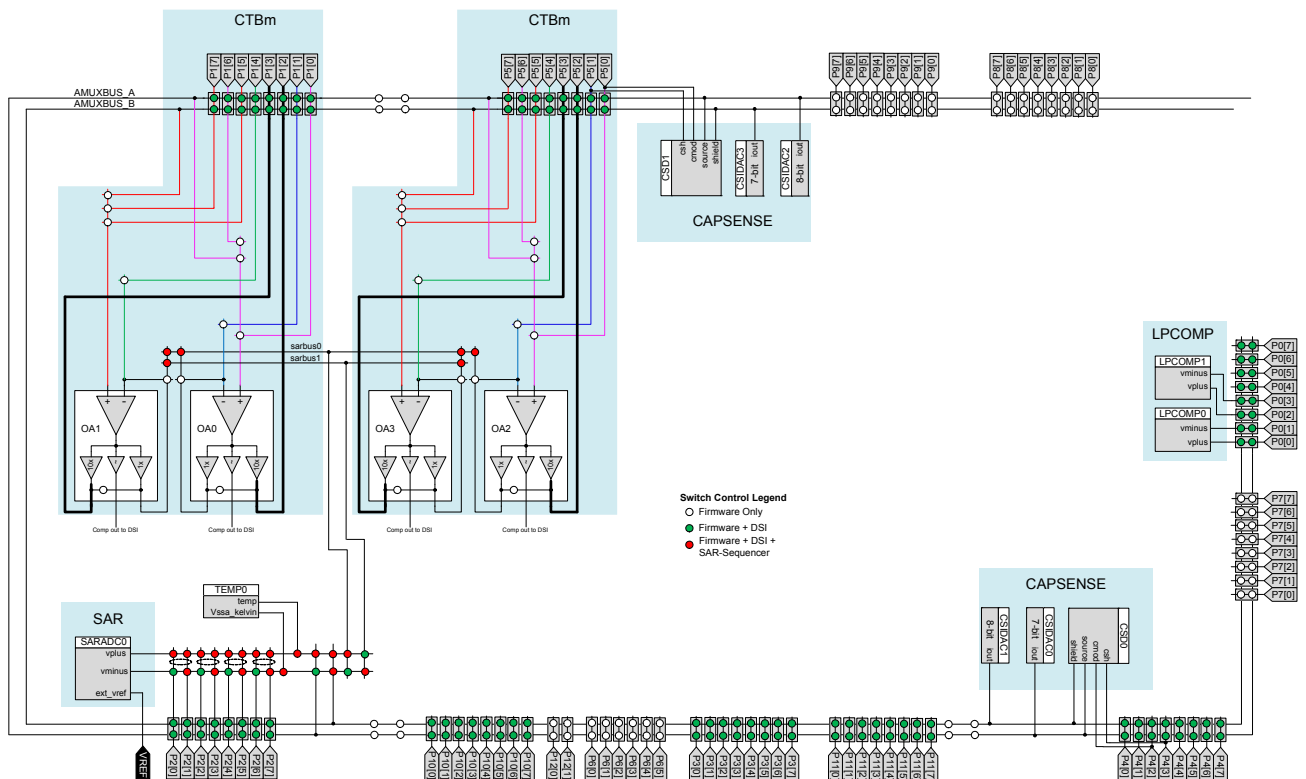
- 导通电阻：600 Ω （最大值）
- 内部温度传感器
- 由定序器控制模块（SARSEQ）、固件或 UDB 控制。
- 电荷泵内部：
 - 如果 $V_{DDA} < 4.0\text{ V}$ ，应打开电荷泵，以降低开关电阻
 - 如果 $V_{DDA} \geq 4.0\text{ V}$ ，将关闭电荷泵，并输出 V_{DDA} 电压。
- 多个输入：

- 从各引脚的模拟信号（端口 2）
- 温度传感器输出
- 通过 sarbus0/1 的 CTBm 输出（不足以按 1 Msps 的采样频率进行采样）
- AMUXBUS_A/_B（不足以按 1 Msps 的采样率下进行采样）

21.3.2.1 模拟路由

SARMUX 有多个可由 SARSEQ 模块（定序器控制器）、固件或 DSI 控制的开关。定序器和 DSI 都是硬件控制方法，它们可被 SAR_MUX_SWITCH_HW_CTRL 寄存器中的硬件控制位屏蔽。不同的控制方法会有不同的开关控制能力。请参见图 21-4。

图 21-4. SARMUX 开关与控制能力



定序器控制：各个开关由 SARSEQ 模块中的定序器控制。对每个通道的模拟路由进行配置后，将会使能以循环方式自动扫描多个通道，而无需 CPU 的干预。并不是所有开关都可由定序器控制；请参考图 21-4。相应的寄存器包括：

SAR_CHANx_CONFIG、SAR_MUX_SWITCH0、SAR_CTRL 以及 SAR_MUX_SWITCH_HW_CTRL。可用在寄存器模式中的具体配置，请参考第 239 页上的固件模拟路由。

固件控制：可编程寄存器直接定义 VPLUS/VMINUS 接口。它可控制 SARMUX 中的所有开关；请参考图 21-4。例如，在固件控制模式中，可以对任何两个引脚或信号进行差分测量，而不仅仅是测量两个相邻的引脚（如定序器控制方法中介绍的内容）。然而，采集多个通道时则需要 CPU 的干预。相应的寄存器包括：SAR_MUX_SWITCH0、SAR_MUX_SWITCH_HW_CTRL 以及 SAR_CTRL。可用在寄存器模式中的具体配置，请参考第 239 页上的固件模拟路由。

DSI 控制：各个开关由 UDB 中的 DSI 信号控制。在自定义逻辑设计中，它可作为第二个定序器工作。DSI 能控制大多数开关。因此，它可对任何两个引脚和信号间进行差分测量，如固件控制一样。可用于 DSI 模式的具体配置，请参考第 236 页上的 SARMUX 模拟路由。

21.3.2.2 模拟互连

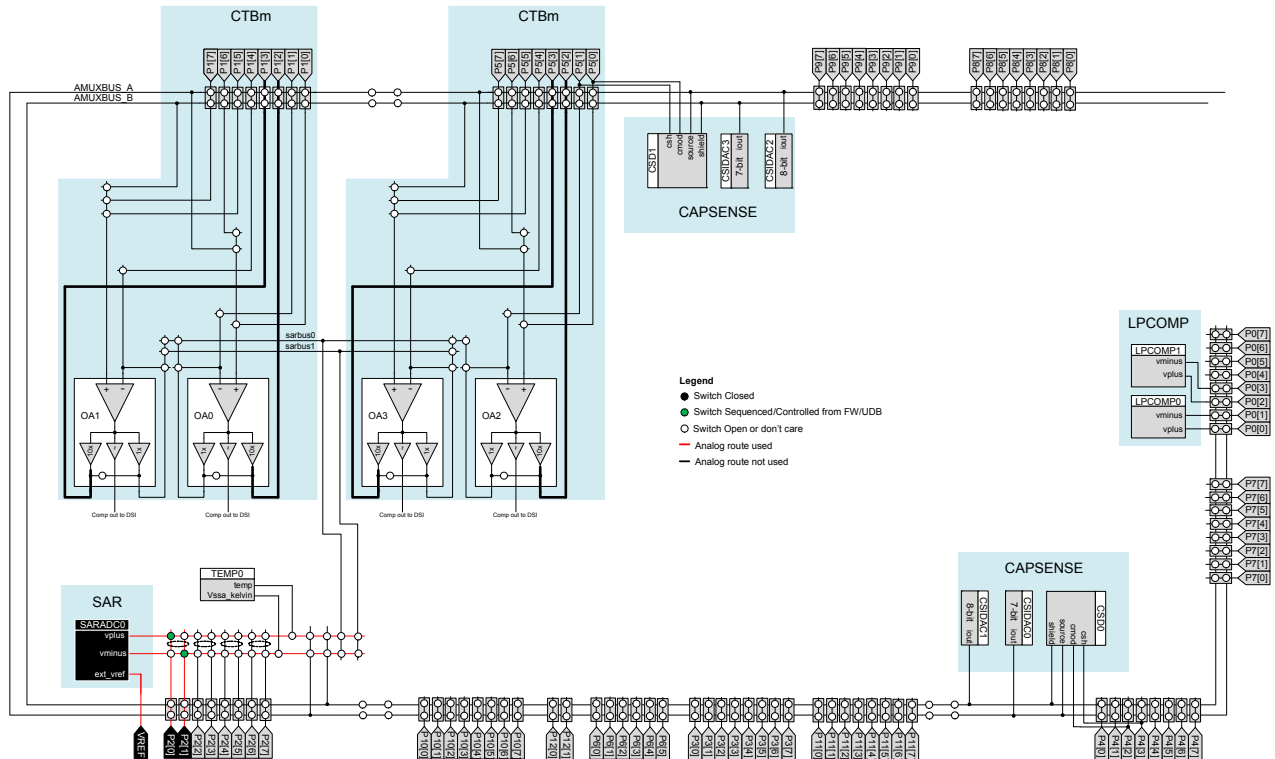
PSoC 4 模拟互连的灵活性很强。通过 SARMUX，可将 SAR ADC 连接至多个输入端，包括外部引脚以及内部信号。例如，它可以连接到一个相邻模块，如 CTBm。它也可以通过 AMUXBUS_A/_B 连接到其它引脚（端口 2 除外），但扫描性能却被降低（需要较多的寄生耦合，较长的 RC 时间来进行扫描）。

下面将介绍几种情况，加深用户对模拟互连的了解。

来自外部引脚的输入

图 21-5 显示的是如何通过各开关，将支持 SARMUX 的两个 GPIO 作为差分对（Vpuls/Vminus）连接至 SAR ADC。这两个开关可由定序器、固件或 DSI 控制。这些引脚被安排在各相邻对中；例如，在 SARMUX 接口 P2[0] 和 P2[1]、P2[2] 和 P2[3]，等等。如果您需要使用未被配对成差分对的各个引脚，（如 P2[1] 和 P2[2]），那么定序器便不适用；请使用固件或 DSI。

图 21-5. 来自外部引脚的输入

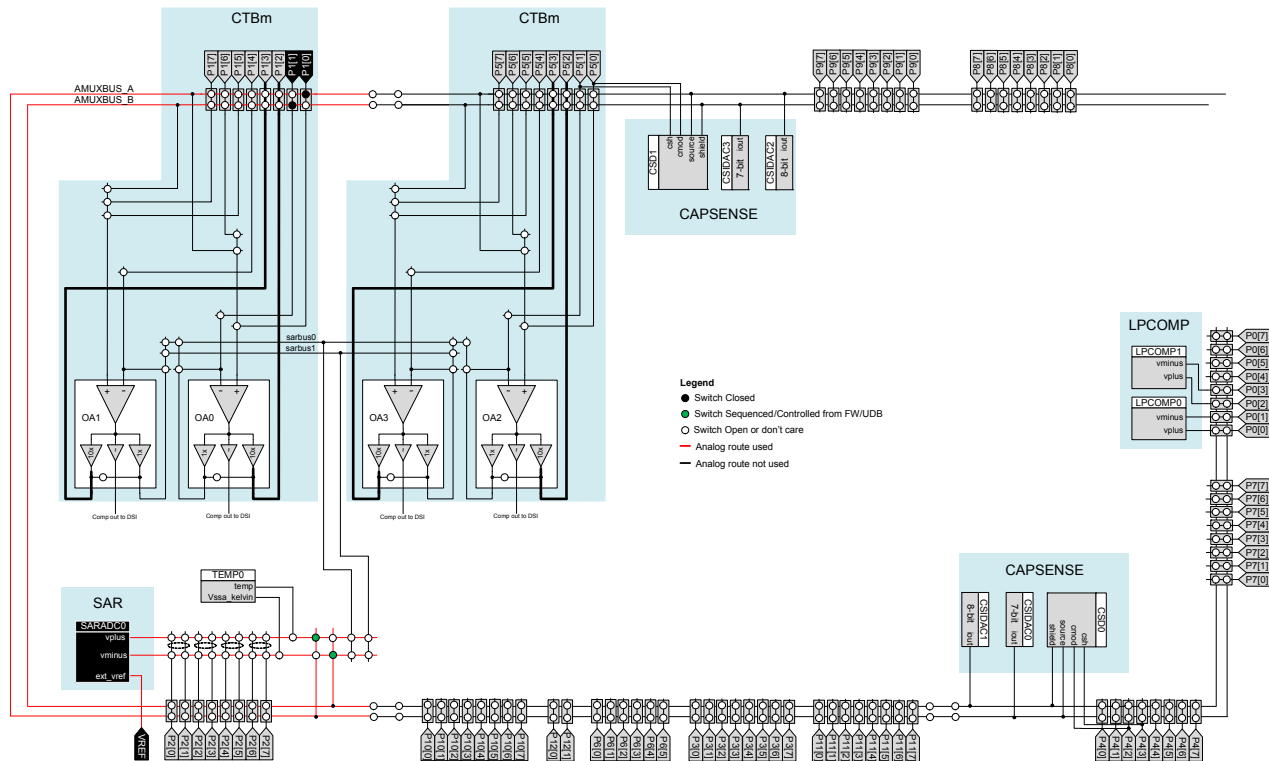


来自模拟总线的输入 (AMUXBUS_A/B)

图 21-6 显示的是不支持 SARMUX 连接的两个引脚如何作为差分对连接至 ADC。需要使用额外的开关将这些差分对连接到 AMUXBUS_A 和 AMUXBUS_B 两个引脚上，然后将 AMUXBUS_A 和 AMUXBUS_B 连接到 ADC 上。

额外的开关将降低扫描性能（需要较多的寄生耦合，较长的 RC 时间来进行扫描）—— 这样不足以按 1 Msps 的采样率下进行采样。不建议使用外部信号。请使用专用的 SARMUX 端口（如果可能）

图 21-6. 来自模拟总线的输入



通过 sarbus 与 CTBm 输出连接的输入

通过 sarbus 0/1, 可以将 SAR ADC 连接至 CTBm 输出。图 21-7 显示了如何将一个运算放大器（被配置为一个跟随器）输出连接到单端 SAR ADC。负端连接至 V_{REF} 。图 21-8 显示的是如何将两个运算放大器输出作为一个差分对连接至 SAR ADC。必须先将运算放大器输出连接至 sarbus 0/1, 然后再将 SAR ADC 输入与 sarbus 0/1 相连接。因为还有额外的开关, 所以不足以按 1 Msps 的采样率进行采样。然而, 该片上运算放大器在很多应用中非常有用。

图 21-7. 通过 sarbus 与 CTBm 输出连接的输入

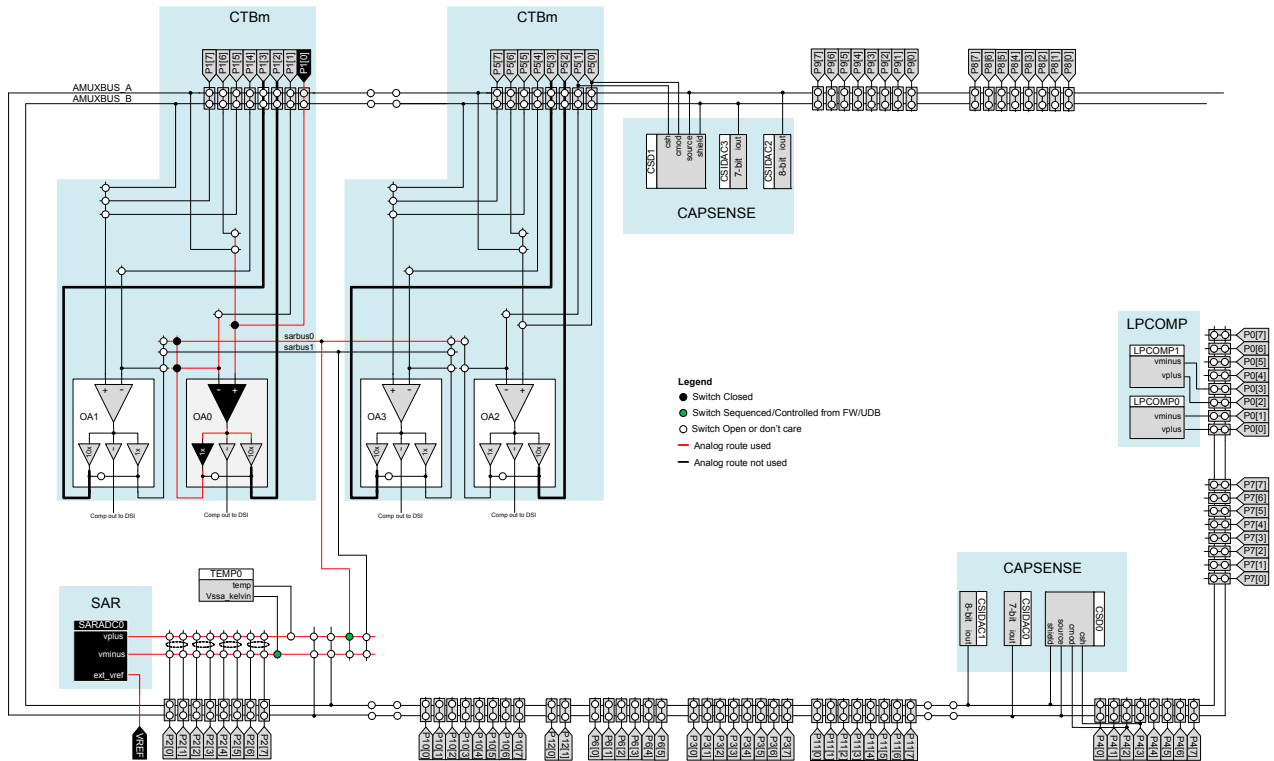
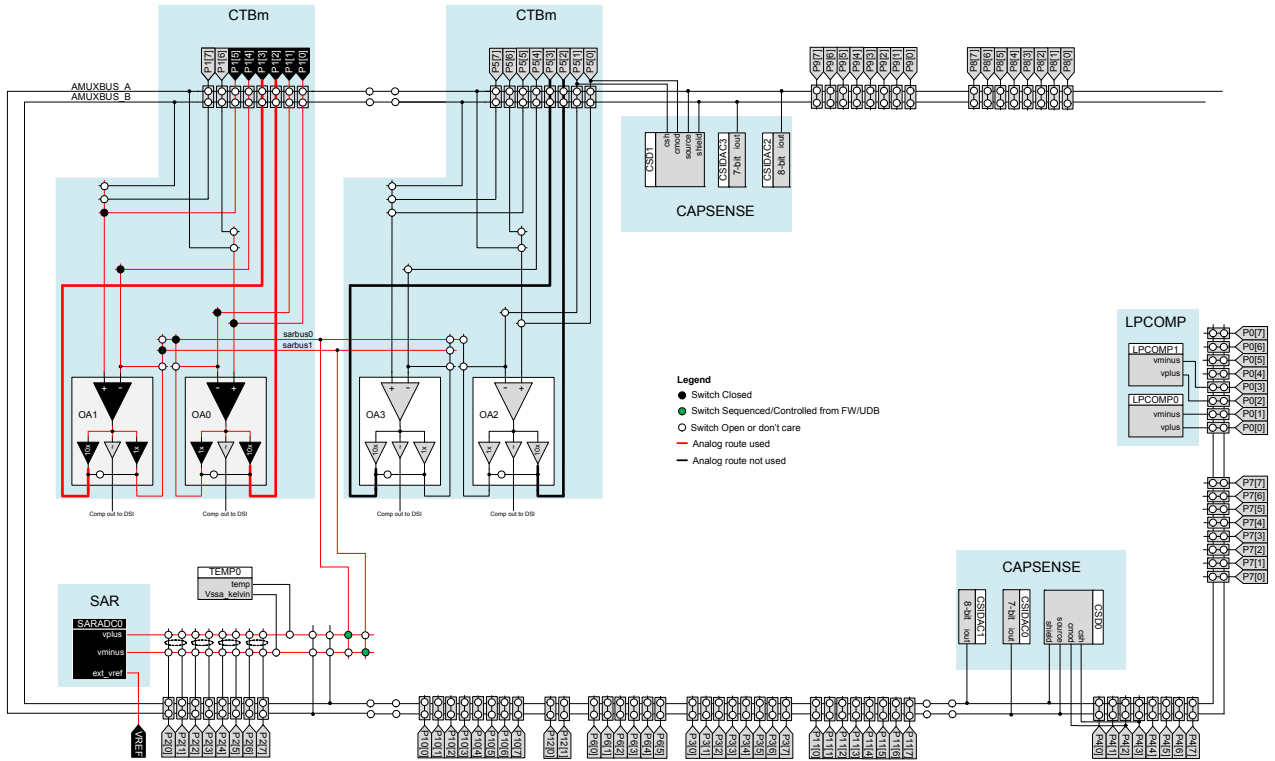


图 21-8. 通过 sarbus0 和 sarbus1 与 CTBm 输出连接的输入

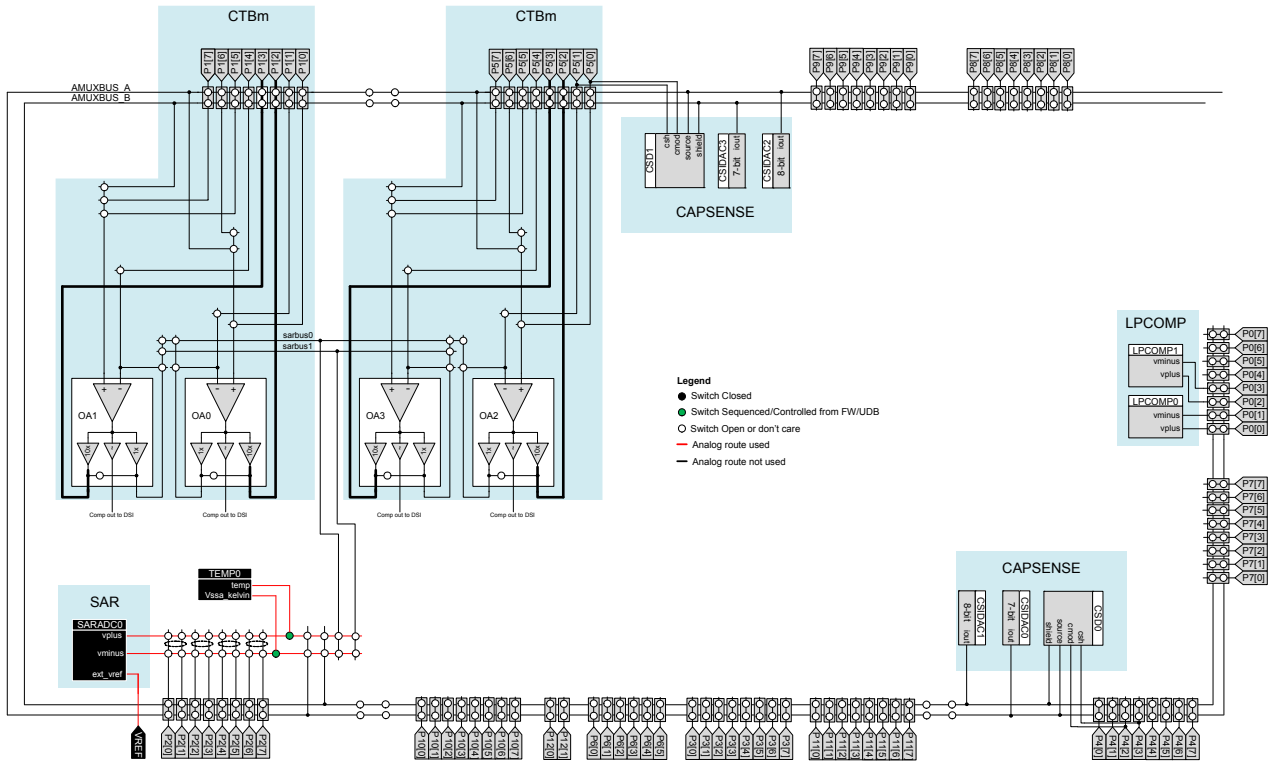


来自温度传感器的输入

一个片上温度传感器可适用于温度感应以及基于温度的校准。请注意，对于温度传感器，差分转换不可用（转换结果未定义），因此总是在单端模式下使用它。内部参考电压为 1.024 V。

如图 21-9 所示，通过开关可将温度传感器路由到 SAR ADC 的正输入上，该操作可由定序器、固件或 DSI 控制。置位 MUX_FW_TEMP_VPLUS 位（即 SAR_MUX_SWITCH0[17]）可以使能温度传感器，并将其输出连接到 SAR ADC 的 VPLUS；但清除该位时，可通过切断其偏置电流来禁用温度传感器。

图 21-9. 来自温度传感器的输入

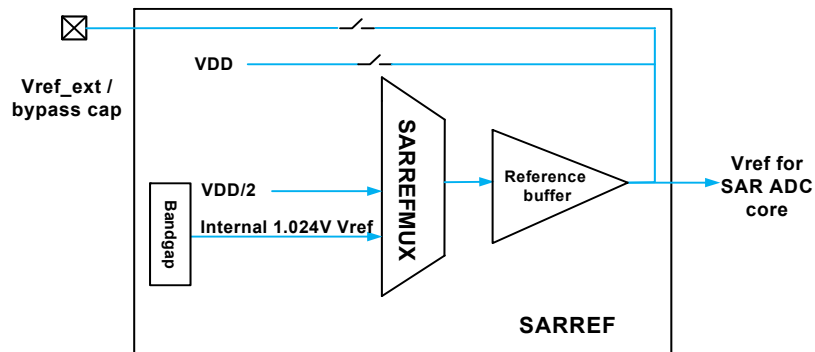


21.3.3 SARREF

SARREF 的主要特性如下：

- 参考电压选项：V_{DDA}、V_{DDA}/2、1.024 V 带隙（±1%）以及外部参考电压
- 参考缓冲器以及旁路电容，用以加强内部参考驱动能力

图 21-10. SARREF 框图



21.3.3.1 参考电压选项

SAR ADC 的参考电压选项包括 SARREF 中的参考复用器和开关。这样的选项允许将 V_{DDA} 、 $V_{DDA}/2$ 和从一个带隙的 1.024 V 内部参考电压或外部 V_{REF} 连接至一个外部 Vref/SAR 旁路引脚（具体请参见器件数据手册）。SARREF 内的参考复用器的控制是在全局配置寄存器 SAR_CTRL [6:4] 内进行的。

21.3.3.2 旁路电容

从带隙的 1.024 V 或 $V_{DDA}/2$ 等内部参考电压都是通过参考缓冲器得到缓冲的。该参考可能会路由至外部 Vref/SAR 旁路引脚，该引脚上的外部电容可用于过滤掉参考信号中存在的内部噪声。

如果不使用外部参考旁路电容，则 SAR ADC 采样率不可超过 100 ksps（在 12 位分辨率时）。例如，当不使用旁路电容并且内部 V_{REF} 为 1.024 V 时，SAR ADC 时钟的最大频率为 1.6 MHz。使用外部参考电压时，推荐使用一个外部电容。通过设置 SAR_CTRL [7] 可以启用旁路电容。表 21-3 列出了不同的参考模式以及在 12 位连续模式操作条件下的最大频率 / 采样率。

表 21-3. 参考模式

参考模式	参考 SAR_CTRL [6:4]	旁路电容 SAR_CTRL[7]	缓冲器	最大频率	最大采样率 (在 12 位分辨率时)
不使用旁路电容时 1.024 V 内部参考电压 V_{REF}	4	0	支持	1.6 MHz	100 ksps
使用旁路电容时的 1.024 V 内部参考电压 V_{REF}	4	1	支持	18 MHz	1 Msps
外部 V_{REF} （低阻抗路径）	5	X	不支持	18 MHz	1 Msps
不使用旁路电容时的 $V_{DDA}/2$	6	0	支持	1.6 MHz	100 ksps
使用旁路电容时的 $V_{DDA}/2$	6	1	支持	18 MHz	1 Msps
V_{DDA}	7	X	不支持	9 MHz	500 ksps

1.024 V 内部 V_{REF} 的启动时间会因旁路电容大小不同而不同。表 21-4 列出了两个最常见的旁路电容值以及对应的启动时间。如果在各扫描操作间或退出睡眠 / 深度睡眠状态后进行扫描时更改了参考选项，需要确保 SAR ADC 开始采样时，1.024 V 的内部 V_{REF} 处于稳定状态。在最坏的情况下，稳定时间（ V_{REF} 完全放电时）与启动时间一样长。

表 21-4. 旁路电容值

内部 V_{REF} 的启动时间	最大时间规范
使用 1 μ F 大小的外部电容时的参考电压启动时间	2 ms
使用 100 nF 大小的外部电容时参考电压的启动时间	200 μ s

21.3.3.3 输入范围和参考

所有输入的电压要处于 V_{SSA} 和 V_{DDA} 之间的范围内。该 ADC 输入电压的范围受 V_{REF} 选择的限制。如果负输入上的电压为 V_n ，并且 ADC 参考电压为 V_{REF} ，那么正输入的电压范围将为 $V_n \pm V_{REF}$ 。只要负和正输入上的电压都处于 V_{SSA} 到 V_{DDA} 的范围内，该标准就适用于单端模式和差分模式。

21.3.4 SARSEQ

SARSEQ 是一个专用的定时器控制器，它会自动控制输入复用器逐个对通道进行扫描，并将其结果放置在寄存器的一个阵列中，这样每个寄存器便存储每一个通道的扫描结果。

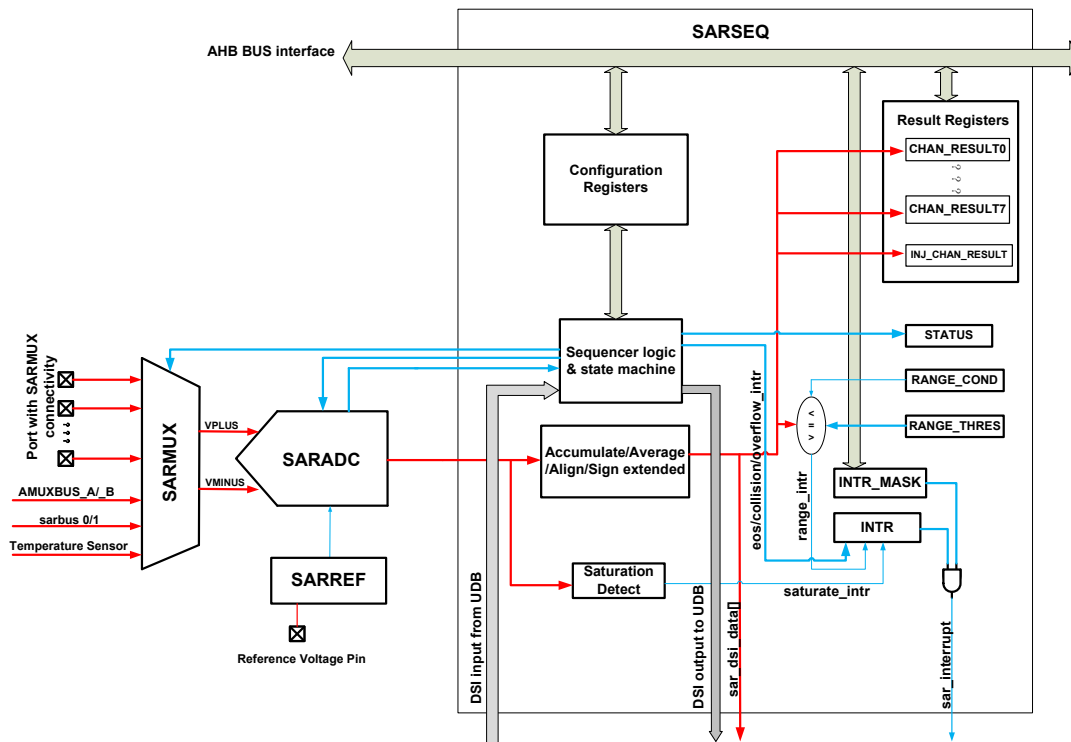
- 自动控制 SARMUX 模拟路由而无需 CPU 的干预
- 控制 SAR ADC 内核（如分辨率、采集时间和参考电压）
- 接收 SAR ADC 输出的数据，并对该数据进行预处理（求平均值、范围检测）
- 结果被双缓冲，因此处理下一个扫描时，CPU 能够安全读取最终扫描操作的结果。

SARSEQ 的特性如下：

- 可单独使能 8 个通道，使之自动进行扫描而无需 CPU 的干预
- 第九个通道（插入通道）用于在一个自动扫描中插入偶发信号
- 每个通道具有以下特性：
 - 从外部引脚或内部信号输入（如 AMUXBUS/CTBm/温度传感器）
 - 最多支持四个可编程采集时间选项
 - 分辨率默认为 12 位，另外还有可选的备用分辨率（8 位或 10 位）
 - 单端模式或差分模式
 - 对结果进行求平均值
- 扫描触发

- 单触发周期模式或连续模式
- 由 GPIO 引脚上的任何数字信号或输入触发
- 由固定功能模块的内部 UDB 触发
- 由软件触发
- 支持硬件求平均
 - 第一阶累加
 - 对从 2 至 256 样本（2 的幂）进行求平均运算
 - 结果使用 16 位表达式
- 输出数据的双缓冲
 - 对结果进行左对齐或右对齐
 - 结果被保存在工作寄存器和结果寄存器内
- 中断生成
 - 完成了扫描转换操作
 - 在所有控制模式下的通道饱和检测
 - 每个通道的超范围（可配置）检测
 - 扫描结果溢出
 - 冲突检测
- 可配置插入通道
 - 由固件触发
 - 可以在两种扫描序列之间交错（紧接）
 - 可选的采样时间、分辨率、单端或差分模式、求平均值

图 21-11. SARSEQ 框图



21.3.4.1 求平均

SARSEQ 模块具有一个 20 位的累加器和移位寄存器，用于进行求平均值运算。有符号扩展后进行求平均值。全局配置寄存器 SAR_SAMPLE_CTRL 指定求平均操作的详细信息。

在寄存器控制模式下，通道配置寄存器 SAR_CHAN_CONFIG 具有一个用于使能求平均的使能位 (AVG_EN)。在 DSI 控制模式下，dsi_cfg_average 信号会使能求平均操作。

在全局配置寄存器中，AVG_CNT (SAR_SAMPLE_CTRL [6:4]) 将根据以下公式指定采样数 (N)：

$$N = 2^{(AVG_CNT+1)}, \quad N \text{ 范围} = [2..256]$$

例如，如果 AVG_CNT (SAR_SMAPLE_CTRL [6:4]) = 3，则 N = 16。

AVG_SHIFT 位 (即 SAR_SAMPLE_CTRL[7]) 用于移位结果，从而提取平均值；如果使能了求平均操作，应设置该位。

如果配置了某个通道用以进行求平均，那么每次扫描时，SARSEQ 将会使用指定通道的 N 个连续样本。由于转换结果是 12 位，并且 N 的最大值是 256 (左移 8 位)，因此 20 位的累加器将不会发生上溢现象。

如果设置了 SAR_SAMPLE_CTRL 寄存器中的 AVG_SHIFT，那么 SAR 定序器将实现有符号扩展，然后进行累加。累加结果将右移 AVG_CNT + 1 位，以进行求平均运算。如果不置位该位，会强制该结果向右转移，以确保该位的宽度不超过 16 位。通过 (0, AVG_CNT - 3) 的最大值实现右移 — 如果采样数超过 16 个 (AVG_CNT > 3)，则累加结果被右移 (AVG_CNT - 3) 位；如果 AVG_CNT < 3，则该结果不被移位。请注意，在这种情况下，平均结果大于所需结果；推荐置位 AVG_SHIFT。该模式总是使用 ADC 所选的分辨率 (12、10 或 8 位)。

21.3.4.2 范围检测

通过 SARSEQ 支持的范围检测，可以在无 CPU 干涉的情况下，将结果值与两个可编程阈值自动进行比较。范围检测是由 SAR_RANGE_THRES 寄存器定义。RANGE_LOW 字段 (即 SAR_RANGE_THRES [15:0]) 的值定义了下限阈值，而 RANGE_HIGH 字段 (即 SAR_RANGE_THRES [31:16]) 则定义了它的上限阈值。

SAR_RANGE_COND 位用于定义触发某个通道的可屏蔽范围检测中断 (RANGE_INTR) 的条件。可以选择以下条件：

- 0: 结果 < RANGE_LOW (低于阈值)
- 1: RANGE_LOW ≤ 结果 < RANGE_HIGH (在范围内)
- 2: RANGE_HIGH ≤ 结果 (超出范围)
- 3: 结果 < RANGE_LOW || RANGE_HIGH ≤ 结果 (在范围外)

更多有关信息，请参考第 232 页上的范围检测中断。

21.3.4.3 双缓冲

通过使用双缓冲，固件可以在处理下一个扫描操作时读取已完成的扫描结果。将 SAR ADC 的结果写入到一组工作寄存器内，直到扫描结束为止。此时数据将被复制到第二组寄存器内，这样数据可被用户应用读取。这样会为固件提供足够的时间，以在当前扫描结束前能够读取前一个扫描结果。使用 16 个寄存器对插入通道除外的所有输入通道进行双缓冲。由于插入通道一般不是正常通道扫描的一部分，因此不需要对它进行双缓冲。

21.3.4.4 插入通道

插入通道同其它通道相似，但它不是正常扫描操作的一部分。插入通道用于偶然或罕见的转换；例如，每隔两秒都会对温度传感器进行采样。请注意，如果 SAR 正运行于连续模式，则使能插入通道会改变采样率。

只有带有固件触发器 (单触发) 的固件才能控制插入通道。因此插入通道不支持连续或 DSI 触发器。它也不支持将它的数据或中断输出到 DSI 总线上。由于只触发一次，所以不需要双缓冲或溢出中断。

通过设置 SAR_INJ_CHAN_CONFIG 寄存器，可以使用与正常通道转换相同的方法来配置插入通道的转换。该寄存器支持：

- 引脚或信号选择
- 单端或差分选择
- 选择 12 位分辨率或全局指定 SUB_RESOLUTION 分辨率
- 从 4 个全局指定采样时间选项中选择一个
- 平均值选择

它支持将相同的中断使用于正常通道 (溢出中断除外)。

- 可屏蔽的转换结束中断 INJ_EOC_INTR
- 可屏蔽的范围检测中断 INJ_RANGE_INTR
- 可屏蔽的饱和检测中断 INJ_SATURATE_INTR
- 可屏蔽的冲突检测中断 INJ_COLLISION_INTR

SAR_INTR、SAR_INTR_MASK、SAR_INTR_MASKED 以及 SAR_INTR_SET 是相应的寄存器。

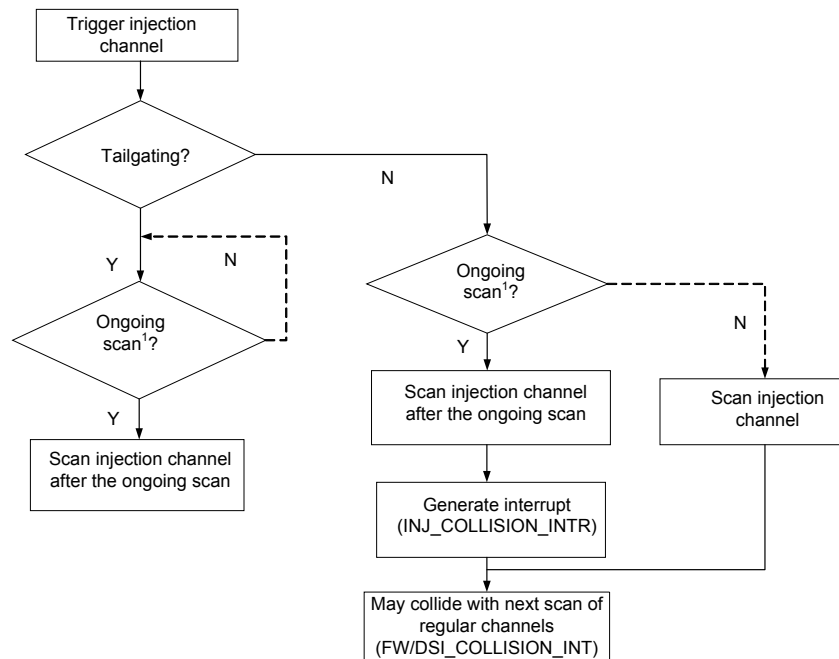
第 236 页上的全局 SARSEQ 配置、第 237 页上的通道配置和第 232 页上的中断中详细地介绍了这些特性。

紧接

通过置位起始或使能位 **INJ_START_EN** (**SAR_INJ_CHAN_CONFIG** [31])，可以触发插入通道的转换。推荐通过设置 **INJ_TAILGATING = 1** (**SAR_INJ_CHAN_CONFIG** [30]) 来选择紧接方式。可在正常通道正在进行的扫描操作结束时扫描插入通道，而不会发生任何冲突。然而，如果当前未进行任何扫描或者 **SAR ADC** 处于闲置状态，并选择了紧接方式，那么 **INJ_START_EN** 位会允许在正常通道的下一次扫描结束时扫描插入通道。

如果未选择紧接方式，并且当前没有进行任何扫描或者 **SAR ADC** 处于闲置状态，那么设置 **INJ_START_EN** 位会立即启动插入通道的转换。如果正在扫描正常通道，那么将在当前扫描结束时开始扫描插入通道，但是这样会导致冲突，并生成一个冲突中断 (**INJ_COLLISION_INTR**)。另一个潜在问题就是正常通道的下次扫描与插入通道的转换发生冲突 (生成 **FW/DSI_COLLISION_INTR** 中断)。因此，插入扫描结束前，正常扫描一直被延缓，从而导致正常扫描过程发生抖动。请注意，连续触发和 **DSI** 触发电平模式绝不会触发冲突中断。

图 21-12. 插入通道流程图



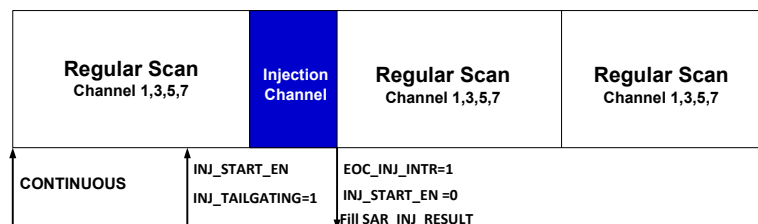
¹ scan here means scan of ALL the regular channels

紧接方式的缺点是，它可能需要很长时间才会发生下一次触发。如果不存在导致正常通道的冲突或抖动的风险，则没有选择紧接时能够安全使用插入通道。

插入通道的转换完成后，将设置“转换结束”中断 (**INJ_EOC_INTR**) 并清除 **INJ_START_EN** 位。插入通道的转换数据被放置在 **SAR_INJ_RESULT** 寄存器中。与 **SAR_CHAN_RESULT** 相似，各寄存器包含“valid” (= **INJ_EOC_INTR**)、范围检测、饱和检测中断等的镜像位以及冲突中断 (**INJ_COLLISION_INTR**) 的镜像位。

图 21-13 显示的实例是在持续扫描 (使能通道 1、3、5 和 7) 时使能插入通道，并且选中了紧接的情况下发生的。请注意，**SAR** 被禁止 (但仅在以前已使能的条件下) 时，将立即清除 **INJ_START_EN** 位。

图 21-13. 选中紧接时使能插入通道



21.3.5 中断

本节所描述的每个中断都在 SAR_INTR_MASK 寄存器中有一个中断屏蔽。将中断屏蔽置为低电平时，相应的中断源将被忽略。如果中断屏蔽位为高电平，并且相应的中断源处于挂起状态，将生成 SAR 中断。

执行中断时，中断服务子程序（ISR）在读取数据后，通过将“1”写入中断位来清除中断源。

SAR_INTR_MASKED 寄存器是中断源和中断屏蔽间的逻辑“与”（AND）的结果。这样会为固件提供一个便利的方法来确定中断源。

要想进行验证和调试，需要通过一个设置位（如 SAR_INTR_SET 寄存器中的 EOS_SET）来触发每个中断。这样会允许固件生成中断而不发生实际事件。

21.3.5.1 “扫描结束”中断（EOS_INTR）

扫描完成后，将生成“扫描结束”中断（EOS_INTR）。从 RESULT（结果）寄存器采集数据后，固件会清除该中断。

还可通过置位 SAR_SAMPLE_CTRL [31] 中的 EOS_DSI_OUT_EN 位来向 DSI 总线发出 EOS_INTR 中断。在 DSI 总线上，EOS_INTR 信号会保持两个系统时钟周期。针对扫描进行的最后一个通道（若选中），这些周期与 data_valid 信号一致。

通过将 SAR_INTR_MASK 寄存器中的 EOS_MASK 位设置为 0，可以屏蔽 EOS_INTR。SAR_INTR_MASKED 寄存器的 EOS_MASKED 位是中断标志和中断屏蔽间的逻辑“与”（AND）的结果。将“1”写入到 SAR_INTR_SET 寄存器中的 EOS_SET 位可以设置用于调试和验证的 EOS_INTR。

21.3.5.2 溢出中断

如果新的扫描操作已结束，并且硬件尝试将 EOS_INTR 和 EOS_INTR 设置为高电平（固件不够快地清除它），那么硬件会生成一个溢出中断（OVERFLOW_INTR）。这通常意味着当前的扫描完成前，固件不可读取先前的扫描结果。在这种情况下，旧数据将被覆盖。

通过将 SAR_INTR_MASK 寄存器中的 OVERFLOW_MASK 位置 0，可以屏蔽 OVERFLOW_INTR。SAR_INTR_MASKED 寄存器的 OVERFLOW_MASKED 位是中断标志和中断屏蔽经过逻辑“与”（AND）运算的结果，这样便于固件使用。将“1”写入到 SAR_INTR_SET 寄存器中的 OVERFLOW_SET 位可以设置用于调试和验证的 OVERFLOW_INTR。

21.3.5.3 冲突中断

SARSEQ 忙碌于以前触发器所触发的扫描操作时，有可能生成了一个新的触发。因此，当前扫描完成前，新触发器的扫描将一直被延迟。此时，要注意通知固件新采样无效。可通过冲突中断来实现。当接收到一个新的触发（而不是连续触发）时，便会生成该中断。

有三种冲突中断：用于固件触发器（FW_COLLISION_INTR），用于 DSI 触发器（DSI_COLLISION_INTR）和用于插入通道（INJ_COLLISION_INTR）。这允许固件确定哪个触发器与正在执行的扫描相冲突。

在电平模式下使用 DSI 触发器时，不会设置 DSI_COLLISION_INTR。

通过将 SAR_INTR_MASK 寄存器中的相应位设置为“0”，可以屏蔽三个冲突中断。SAR_INTR_MASKED 寄存器的相应位是中断标志和中断屏蔽间的逻辑“与”（AND）的结果。将数值“1”写入到 SAR_INTR_SET 寄存器中的相应位，可以设置用于调试和验证的冲突中断。

21.3.5.4 “插入转换结束”中断（INJ_EOC_INTR）

完成插入通道的转换后，将生成“插入转换结束”中断（INJ_EOC_INTR）。从 INJ_RESULT 寄存器采集数据后，固件会清除该中断。

请注意，如果插入通道扫描紧接着一个扫描，将生成 EOS_INTR，同时也开始进行插入通道转换。该插入通道不被视为扫描的一部分。

通过将 SAR_INTR_MASK 寄存器中的 INJ_EOC_MASK 位设置为“0”，可以屏蔽 INJ_EOC_INTR。SAR_INTR_MASKED 寄存器的 INJ_EOC_MASKED 位是中断标志和中断屏蔽间的逻辑“与”（AND）的结果。将“1”写入到 SAR_INTR_SET 寄存器中的 INJ_EOC_SET 位，可以设置用于调试和验证的 INJ_EOC_INTR。

21.3.5.5 范围检测中断

实现求平均值、调整以及符号扩展（如适用）后即可设置范围检测中断标志。这意味着不用等待整个扫描过程完成，即可确定通道转换是否超出范围。该阈值的数据格式需要与结果数据格式相同。

通过将 SAR_RANGE_INTR_MASK 寄存器中的指定位设置为“0”，可以屏蔽指定通道的范围检测中断。SAR_RANGE_INTR_MASKED 寄存器反映的是中断请求和屏蔽寄存器间的按位进行逻辑与运算（AND）后得到的结果。如果该值为非零值，则输入到 NVIC 的 SAR 中断信号为高电平。

SAR_RANGE_INTR_SET 可用于调试/验证。写入“1”值，从而设置中断请求寄存器中的相应位；读取它时，该寄存器反映的是中断请求寄存器。

每个通道都有一个范围检测中断（RANGE_INTR 和 INJ_RANGE_INTR）。

21.3.5.6 饱和检测中断

饱和检测适用于所有转换操作。该特性将检测采样值是否等于指定分辨率的最小值或最大值。如果它等于最小值 / 最大值，那么将为相应通道设置可屏蔽中断标志。这样会允许固件在 SAR ADC 饱和时采取行动，如清除结果。转换结束后并且实现求平均前会立即测试采样值。因此当数据寄存器中的平均值不等于最小值或最大值时，将设置该中断。

当为通道选择 10 位或 8 位的分辨率时，将在 10 位或 8 位的数据上进行饱和检测。

扫描结束和实现求平均值前，应立即设置饱和中断标志以能够快速响应饱和。通过将 SAR_SATURATE_INTR_MASK 寄存器中的指定位设置为 ‘0’，可以屏蔽指定通道的饱和检测中断。SAR_SATURATE_INTR_MASKED 寄存器反映的是中断请求和屏蔽寄存器之间按位进行逻辑与运算（AND）后的结果。如果该值为非零值，则输入到 NVIC 的 SAR 中断信号为高电平。

SAR_SATURATE_INTR_SET 可用于调试 / 验证。写入 ‘1’ 值，从而设置中断请求寄存器中的相应位；读取它时，该寄存器反映的是中断请求寄存器。

21.3.5.7 中断源的概述

INTR_CAUSE 寄存器包含所有挂起 SAR 中断的概述。它允许 ISR 确定导致中断的原因。该寄存器包含 SAR_INTR_MASKED 的镜像副本。此外，它还有两位，它们汇总了所有通道的范围和饱和检测中断。它对 RANGE_INTR_MASKED 和 SATURATE_INTR_MASKED 寄存器中所有位进行逻辑 “或” 运算（OR）（INJ_RANGE_INTR 和 INJ_SATURATE_INTR 除外）。

21.3.6 触发器

可通过以下的三种方法触发扫描：

- 当固件对 SAR_START_CTRL 寄存器中的 FW_TRIGGER 位进行写操作时，会生成一个固件或单触发的触发器。扫描结束后，SARSEQ 将清除 FW_TRIGGER 位，并进入闲置模式等待下次触发。SAR 被禁用后会立即清除 FW_TRIGGER 位。
- 通过 DSI 接口（dsi_trigger）传输周期性触发器。该触发器被连接到 TCPWM 的输出端，还可将它连接到 UDB 或任何一个 GPIO 引脚。UDB 可以作为一个状态机使用，以查找一定的事件序列。
- 通过置位 SAR_SAMPLE_CTRL 寄存器中的 CONTINUOUS（连续）位，可以激活连续触发器。在该模式下，扫描完成后，SARSEQ 会立即启动下一个扫描，所以 SARSEQ 总是处于 BUSY（忙碌）状态。因此，必须忽略其它所有触发器。请注意，下一次扫描结束时，硬件仍然会清除 FW_TRIGGER。

尽管不需要硬件，但这三个触发器仍相互独立。如果 DSI 触发器与固件触发器一致，则先会处理 DSI 触发器，然后单独扫描固件触发器（并设置一个冲突中断）。当 DSI 触发器与连续触发器一致时，将同时有效处理这两个触发器（可能为 DSI 触发器设置冲突中断）。

对于固件连续触发器，定序器通知 SAR ADC 开始进行采样（已给的定序器处于闲置状态）前，只需要一个 SAR ADC 时钟周期。对于 DSI 触发器，这取决于触发配置的设置。

21.3.6.1 DSI 触发配置

■ DSI 同步

SARSEQ 的 DSI 接口在系统时钟频率（clk_sys）下运行；请参考第 77 页上的时钟系统章节了解详细的信息。如果正在输入的 DSI 触发器信号与 AHB 时钟不同步，则需要通过双反转该信号（默认）使其同步。但是如果 DSI 触发信号已经与 AHB 时钟同步，则可旁路这双反转操作。SAR_SAMPLE_CTRL 寄存器中的配置位（DSI_SYNC_TRIGGER）控制双反转的旁路。DSI_SYNC_TRIGGER 影响到 DSI 脉冲触发信号所需要的触发宽度（TW）以及触发的时间间隔（TI）。

■ DSI 触发电平

DSI 触发可以是脉冲触发或电平触发；这取决于 SAR_SAMPLE_CTRL 寄存器中的配置位 DSI_TRIGGER_LEVEL。如果是电平触发，只要 DSI 触发信号保持为高电平，SAR 将启动新的扫描操作。DSI 触发信号是一个脉冲输入时，则在 DSI 触发信号上检测到的上升沿将触发一个新的扫描。

■ 传输时间

如果生成了 ‘dsi_trigger’，则通知 SAR ADC 开始进行采样前，需要一段传输时间。根据 DSI_SYNC_TRIGGER 和 DSI_TRIGGER_LEVEL 配置的不同，传输时间也不一样；表 21-5 显示的是传输的最大时间。两个触发脉冲时间间隔应该长于传输时间，否则第二个触发脉冲将被忽略。

SAR 被禁用（ENABLED=0）时，会忽略 DSI 触发器。

表 21-5. DSI 触发的最长时间

DSI_TRIGGER 的最长转换时间	旁路同步 DSI_SYNC_TRIGGER = 0	使能同步 DSI_SYNC_TRIGGER = 1 (默认)
脉冲触发: DSI_TRIGGER_LEVEL = 0 (默认)	1 clk_sys+2 clk_sar	3 clk_sys+2 clk_sar
电平触发: DSI_TRIGGER_LEVEL=1	2 clk_sar	2 clk_sys+2 clk_sar

表 21-6. 触发信号要求

触发规范	要求
触发宽度 (TW)	TW 应足够大, 以锁定触发器。如果 DSI_SYNC_TRIGGER = 1, 则 TW >= 2 clk_sys 周期。如果 DSI_SYNC_TRIGGER = 0, 则 TW >= 1 SAR 时钟周期。
触发的时间间隔 (TI)	与传输时间相比, DSI 脉冲触发信号的触发间隔应更长 (如表 21-5 中所指定), 否则, 第二个触发脉冲将被忽略。

21.3.7 SAR ADC 状态

通过 SAR_STATUS 寄存器中的 BUSY 和 CUR_CHAN 字段, 可以观察到 SAR 的当前状态。无论 SAR 正在对某个通道进行采样还是转换, BUSY (忙碌) 位都会保持高电平; CUR_CHAN [4:0] 位表示正在采样的当前通道的编号 (通道 16 指示插入通道)。SW_VREF_NEG 位表示 SAR ADC 中当前开关的状态, 包括 DSI 控制和寄存器控制 (该开关将 V_{REF} 输入短路连接到 NEG 引脚)。

如果在终端扫描过程中被采样的 WORK (工作) 数据可用, 那么会置位 CHAN_WORK_VALID 寄存器中的 CHAN_WORK_VALID 位。当 CHAN_RESULT_VALID 寄存器中的 CHAN_RESULT_VALID 被置位时, 便意味着 RESULT (结果) 数据是可用的, 并且相应的 CHAN_WORK_VALID 位被清除。SAR_AVG_STAT 寄存器中的 CUR_AVG_ACCU 和 CUR_AVG_CNT 字段指示了当前平均累加器的内容以及用于实现求平均的当前采样计数器值 (递减计数)。

SAR_MUX_SWITCH_STATUS 寄存器提供了 MUX_SWITCH0 寄存器当前的开关状态。

这些状态寄存器有助于进行调试 SAR。

21.3.8 低功耗模式

SAR ADC 的电流消耗可分为两个部分: SAR ADC 内核和 SARREF。

可以通过某些方法来降低 SAR ADC 内核的功耗。最简单的方法是降低触发器频率, 即降低每秒中进行的转换次数。另一个方法是对不要求高精度的通道采用较低的分辨率。这样, 最多可减少转换时间共 18 个周期中的四个 (对于 8 位分辨率和最小的采样时间)。

另外, SAR ADC 还提供了用于控制其总功耗的 ICNT_LV[1:0] 配置位。需要观察每个功耗设置的最大时钟速率。

表 21-7. 低功耗的 ICNT_LV

ICNT_LV[1:0]	SAR ADC 内核的相对功耗 [%]	最大频率 [MHz]	最短的采样时间 [周期]	最大采样率 (在 12 位分辨率时) [ksps]
0	100	18	4	1000
1	50	9	3	529
2	133	18	4	1000
3	25	4.5	2	281

除了控制 SAR ADC 内核的功耗外, 还可以配置 VREF 缓冲器 (若使用) 的功耗。请注意, 为了在不使用外部旁路电容时能够获得全 VDDA 范围 (1.7 V 到 5.5 V), VREF 缓冲器必须运行于 2x 功耗模式。然而, 不使用外部旁路电容时所支持的最大采样率仍是 100 ksps。为了达到 1 Msps 的采样率, 需要使用外部旁路电容和一个 18 MHz 的时钟。更多详细信息, 请参见表 21-8。

表 21-8. SAR VREF 功耗选项

PWR_CTRL_VREF [1:0]	要求外部旁路电容	相对 VREF 功耗 [%]	最大频率 [MHz]	最短的采样时间 [周期]	最大采样率 (在 12 位分辨率时) [ksps]	VDDA 范围
0	是	100	18	4	1000	1.7 V - 5.5 V
0	否	100	1.6	2	100	2.7 V - 5.5 V
2	否	200	1.6	2	100	1.7 V - 5.5 V
1 或 3	无效设置 — 不使用					

有了外部 VREF 便不需要使用 VREF 缓冲器和旁路电容，因此能够降低 SAR ADC 模块的总功耗。

21.3.9 系统操作

通过置位 ENABLED 位 (SAR_CTRL [31]) 来使能 SAR 模拟模块后，请使用 SARSEQ 根据下列步骤开始进行 ADC 转换：

1. 设置 SAR ADC 控制模式：21.3.10 寄存器模式或 21.3.11 DSI 模式
2. 通过定时器 / 固件 / DSI 设置 SARMUX 模拟路由 (引脚 / 信号选择)
3. 设置全局 SARSEQ 转换配置
4. 对每个通道源 (如引脚地址) 进行配置
5. 使能各通道
6. 设置触发器类型
7. 设置中断屏蔽
8. 启动触发源
9. 每个转换中断结束后，都要检索数据
10. 进行插入转换 (如果需要)

寄存器模式是指使用寄存器控制 SARMUX 和 SAR ADC 转换；DSI 模式是指使用来自 UDB 的 DSI 进行控制转换。表 21-9 显示了这两种控制模式间的主要区别。可通过设置 DSI_MODE 位 (SAR_CTRL [29]) 使能 DSI 模式。

表 21-9. 各种控制模式间的差别

控制模式	寄存器	DSI
DSI_MODE	0	1
SARMUX 控制	定时器控制寄存器： SAR_CHANx_CONFIG、SAR_MUX_SWITCH0、SAR_MUX_HW_SWITCH_CTRL SAR_CTRL 固件控制寄存器： SAR_MUX_SWITCH0、SAR_MUX_HW_SWITCH_CTRL、SAR_CTRL	DSI 控制信号：dsi_out、dsi_oe、dsi_swctrl、dsi_sw_negvref 固件控制寄存器：SAR_MUX_SWITCH0、SAR_MUX_HW_SWITCH_CTRL、SAR_CTRL
全局配置	全局配置寄存器： SAR_CTRL、SAR_SAMPLE_CTRL、SAR_SAMPLE01、SAR_SAMPLE23、SAR_RANGE_THES、SAR_RANGE_COND	全局配置寄存器： SAR_CTRL、SAR_SAMPLE_CTRL、SAR_SAMPLE01、SAR_SAMPLE23、SAR_RANGE_THES、SAR_RANGE_COND
通道配置	通道配置寄存器： CHAN_CONFIG、CHAN_EN、INJ_CHAN_CONFIG	使用以下 DSI 信号： dsi_cfg_st_sel、dsi_cfg_average、dsi_cfg_resolution、dsi_cfg_differential (CHAN_CONFIG、CHAN_EN、INJ_CHAN_CONFIG 均被忽略)
触发器	全适用 固件触发 (SAR_START_CTRL[0]) DSI 触发 (dsi_trigger) 连续触发 (SAR_SAMPLE_CTRL [0])	全适用 固件触发 (SAR_START_CTRL[0]) DSI 触发 (dsi_trigger) 连续触发 (SAR_SAMPLE_CTRL [0])

表 21-9. 各种控制模式间的差别

控制模式	寄存器	DSI
中断	全适用	全适用（DIS 信号上只有 EOS_INTR、RANGE_INTR、SATU-RATE_INTR 输出）
DSI 输出	支持	支持
结果数据	8 个通道结果寄存器以及一个插入通道结果寄存器	只有通道 0 结果寄存器可用
插入	支持	不支持
求平均	支持在一个引脚 / 信号上进行求平均	支持不同引脚 / 信号上实现求平均

21.3.10 寄存器模式

使用寄存器配置 SAR ADC；这是最通用的方法。有关寄存器位定义的详细信息，请参考 [PSoC 4200L 系列：PSoC 4 寄存器技术参考手册](#)。

21.3.10.1 SARMUX 模拟路由

在寄存器模式下，可以通过两种方法来控制 SARMUX 模拟路由，即：定序器和固件。

定序器控制

请注意，需要将 MUX_SWITCH_HW_CTRL 寄存器中的相应硬件控制位以及 MUX_SWITCH0 寄存器中的固件控制位全部设置为 ‘1’。请确保 SWITCH_DISABLE = 0；设置 SWITCH_DISABLE 以禁止定序器控制。

对于定序器控制，通过端口和引脚地址的组合可以指定通道转换的引脚或内部信号。PORT_ADDR 位是 SAR_CHANx_CONFIG [6:4]，并且 PIN_ADDR 位为 SAR_CHANx_CONFIG [2:0]。表 21-10 显示的是 PORT_ADDR 和 PIN_ADDR 设置情况，以及相应的 SARMUX 选择。未使用的端口 / 引脚被保留给 PSoC 4 系列中其它产品。

表 21-10. PORT_ADDR 和 PIN_ADDR

PORT_ADDR	PIN_ADDR	描述
0	0...7	SARMUX 的 8 个专用引脚
1	X	sarbus0 ^a
1	X	sarbus1 ^a
7	0	温度传感器
7	2	AMUXBUS-A
7	3	AMUXBUS-B

a. sarbus0 和 sarbus1 连接至 CTBm 模块的输出端（该 CTBm 模块包含 opamp0/1）。请参考第 253 页上的微型连续时间模块（CTBm）章节，了解更多信息。PORT_ADDR = 1 时，无论 PIN_ADDR 的值如何，sarbus0 都会连接至 SAR ADC 的正端；当差分模式被使能且 PORT_ADDR=1 时，sarbus1 只能连接至 SAR ADC 的负端。

对于差分转换，负端的连接取决于正端的连接，它由 PORT_ADDR 和 PIN_ADDR 定义。通过设置 DIFFERENTIAL_EN，通道会在引脚地址所指定的偶数 / 奇数

引脚对上上进行差分转换，PIN_ADDR [0] 被忽略。P2.0/P2.1、P2.2/P2.3、P2.4/P2.5、P2.6/P2.7 均为可用的差分对，用于定序器控制。固件或 DSI 可实现更加灵活的模拟。

对于单端转换，NEG_SEL 字段（即 SAR_CTRL [11:9]）用于决定连接至负输入的信号。在差分模式下，这些位都被忽略。负输入的选择影响到输入电压范围和有效分辨率。更多详细信息，请参考第 220 页上的负输入选择。这些选项包括：V_{SSA}、V_{REF} 或来自带有 SARMUX 连接的八个引脚中任意一个引脚的外部输入。要想将负输入连接至 V_{REF}，必须置位 SAR_HW_CTRL_NEGVREF（SAR_CTRL[13]），因为 MUX_SWITCH_HW_CTRL 寄存器没有该硬件控制位。

固件控制

在默认情况下，SARMUX 会运行于固件控制模式。通过置位 SAR_MUX_SWITCH0 [29:0] 中的合适位，可单独控制 SAR ADC 的 V_{PLUS}（正）和 V_{MINUS}（负）输入。清除硬件开关控制寄存器中相应的位（SAR_MUX_SWITCH_HW_CTRL[n] = 0）。否则，硬件控制方法（定序器 / DSI）将控制 SARMUX 模拟路由。

SAR_CTRL 寄存器的 SWITCH_DISABLE 位用于禁止 SAR 定序器使能路由开关。请注意，固件控制模式总能够关闭开关而不受该位的影响；但推荐将该位设置为 ‘1’。

NEG_SEL 字段（SAR_CTRL [11:9]）决定了在单端模式下连接至 SAR ADC 负端（V_{minus}）的信号。在差分模式下，这些位都被忽略。在单端模式下，使用定序器控制时，必须置位这些位。使用硬件控制时，将忽略 NEG_SEL 并设置 SAR_MUX_SWITCH0，以控制负输入。在特殊情况下，当 SAR_MUX_SWITCH0 未将内部 V_{REF} 连接至 V_{minus} 时，则需要将 NEG_SEL 设置为 ‘7’。负输入的选择影响到输入电压范围、信噪比和有效分辨率。更多详细信息，请参考第 220 页上的负输入选择。

21.3.10.2 全局 SARSEQ 配置

一些适用于所有通道的选项是全局配置。在某些情况下，通道配置具有用于选用哪个全局配置部分的位。全局配置适用于寄存器控制模式和 DSI 控制模式。

SAR_CTRL、SAR_SAMPLE_CTRL、SAR_SAMPLE01、SAR_SAMPLE23、SAR_RANGE_THES 以及 SAR_RANGE_COND 都是全局配置寄存器。

正在进行扫描时，通常不会更改这些配置。如果更改了使用过程中的配置设置，将产生“未定义”的结果。可以更改未使用的配置设置，而不会影响正在进行的扫描操作。

表 21-11. 全局配置寄存器

配置	控制寄存器	详细的参考信息
参考电压选择	SAR_CTRL[6:4]	21.3.3.1 参考电压选项
有符号 / 无符号选择	SAR_SAMPLE_CTRL [3:2]	21.3.1.3 结果数据格式
数据的左 / 右对齐	SAR_SAMPLE_CTRL [1]	21.3.1.3 结果数据格式
单端模式下的负输入选择	SAR_CTRL[11:9]	21.3.1.4 负输入选择
分辨率	SAR_SAMPLE_CTRL[0] ^a	21.3.1.5 分辨率
数据采集时间	SAR_SAMPLE_TIME01 [25:0] SAR_SAMPLE_TIME32 [25:0]	21.3.1.6 采集时间
平均计数	SAR_SAMPLE_CTRL[7:4]	21.3.4.1 求平均
范围检测	SAR_RANGE_THRES [31:0] SAR_RANGE_COND [31:30]	21.3.4.2 范围检测

a. 备用分辨率应通过 SAR_CHAN_CONFIG 寄存器中的 SAR_RESOLUTION 位进行使能。如果备用分辨率未被使能，则 ADC 不再考虑由 SAR_SAMPLE_CTRL 寄存器设置的分辨率，在 12 位分辨率下运行。

21.3.10.3 通道配置

通道配置包括：

- 差分或单端模式的选项
- 全局配置选项包括：采样时间、分辨率、求平均使能
- DSI 输出使能

作为一般规则，只能在各扫描之间更改通道配置（和全局配置相同）。但如果在扫描过程中未使能通道，则可随意更改该通道的配置而不会影响正在进行的扫描。如果违反此规则，将产生不确定的结果。被自使能的通道是唯一例外；可在扫描过程中更改被使能的通道，则更改的内容将在下一次扫描时生效。更改使能的通道，可能会改变采样率。

表 21-12. 通道配置寄存器

配置	寄存器	详细的参考信息
单端 / 差分	SAR_CHANx_CONFIG [8]	21.3.1.1 单端模式和差分模式
采集时间选择	SAR_CHANx_CONFIG [13:12]	21.3.1.6 采集时间
分辨率选择	SAR_CHANx_CONFIG [9]	21.3.1.5 分辨率
求平均使能	SAR_CHANx_CONFIG [10]	21.3.4.1 求平均
DSI 输出使能	SAR_CHANx_CONFIG [30]	21.3.11.8 DSI 输出使能

可通过 SUB_RESOLUTION 位（SAR_SAMPLE_CTRL[0]）来选择所使用的备用分辨率（8 位或 10 位）。分辨率（SAR_CHANx_CONFIG [9]）用于决定使用默认的 12 位分辨率还是备用分辨率。当使能了求平均功能时，会忽略 SUB_RESOLUTION；这时分辨率将被调整为最大值（12 位）。

表 21-13. 分辨率

求平均功能	SUB_RESOLUTION SAR_SAMPLE_CTRL[0]	寄存器模式的分辨率 SAR_CHANx_CONFIG [9]	通道分辨率
禁用	0	1	8 位
禁用	1	1	10 位
禁用	0	0	12 位
禁用	1	0	12 位
启用	X	X	12 位

21.3.10.4 通道使能

CHAN_EN寄存器可以单独使能各个通道。发生下一个触发器时，所有使能的通道均被扫描。触发后，可立即更新通道使能，以为下一次扫描做准备。这不会影响到正在进行的扫描。请注意，这是一个例外；扫描过程中不应更改其它所有配置（全局或通道）。

21.3.10.5 中断屏蔽

共有六个中断源：每个都有一个中断屏蔽：

- “扫描结束”中断
- 溢流中断
- 冲突中断
- “插入转换结束”中断
- 范围检测中断
- 饱和检测中断

每个中断均有一个中断请求寄存器（INTR、SATURATE_INTR、RANGE_INTR）、一个软件中断设置寄存器（INTR_SET、SATURATE_INTR_SET、RANGE_INTR_SET）、一个中断屏蔽寄存器（INTR_MASK、SATURATE_INTR_MASK、RANGE_INTR_MASK）以及一个中断请求屏蔽结果寄存器（INTR_MASKED、SATURATE_INTR_MASKED、RANGE_INTR_MASKED）。也可以通过添加一个中断源寄存器来取得所有正在挂起的SAR中断概述，并允许ISR只要通过读取该寄存器即可确定中断源。

更多有关信息，请参考21.3.5 中断。

21.3.10.6 触发器

可使用以下三种方法来启动A/D转换：

- 固件触发：SAR_START_CTRL[0]
- DSI触发：dsi_trigger
- 连续触发：SAR_SAMPLE_CTRL[16]

更多有关信息，请参考21.3.6 触发器。

21.3.10.7 每当转换中断结束后，都将检索数据

请确保每次扫描结束后都要读取结果寄存器中的数据；否则数据会因下次扫描的配置而被改变。

16位数据寄存器用于为多达8个通道实现双缓冲（插入通道没有双缓冲功能）。双缓冲是指每个通道都有个工作寄存器以及一个结果寄存器。采样该通道后，数据立即被写入到工作寄存器中。然后，该扫描中的全部使能通道完成采样后，数据将从工作寄存器复制到结果寄存器内。

当相应的WORK数据有效时，将置位CHAN_WORK_VALID位，即它在当前扫描过程中已被采样。扫描完成后，相应的CHAN_RESULT_VALID将被置位。置位CHAN_RESULT_VALID位时，相应的CHAN_WORK_VALID位将被清除。

为了便于固件，SAR_CHAN_WORK寄存器中的位[31]是SAR_CHAN_WORK_VALID寄存器中相应位的镜像位。SAR_CHAN_RESULT寄存器中的位[29]、位[30]和位[31]都是SAR_SATURATE_INTR、SAR_RANGE_INTR和SAR_CHAN_RESULT_VALID寄存器中相应位的镜像位。请注意，这里所映射的中断位都是原始（未屏蔽）中断位。它有助于固件只要读取数据寄存器，即可检查数据的有效性。

若使能DSI输出，它会允许UDB处理SARSEQ结果数据，并且通道编号允许对不同通道内的数据采用不同的处理方法。有关详细信息，请参见21.3.11.8 DSI输出使能。

21.3.10.8 插入转换

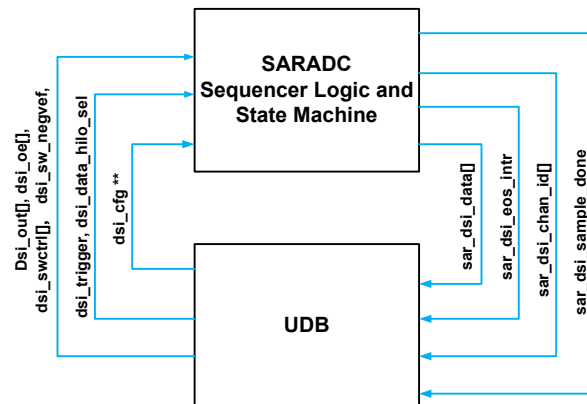
通过置位起始位INJ_START_EN（INJ_CHAN_CONFIG[31]），可以触发插入通道。要想避免正常自动扫描的冲突，推荐通过置位INJ_CHAN_CONFIG[30]使能紧接。被使能时，INJ_START_EN会允许在下一次正常通道扫描结束时进行插入通道扫描。

更多有关信息，请参考21.3.4.4 插入通道。

21.3.11 DSI 模式

在DSI控制模式下，除全局配置（例如，中断屏蔽、范围检测设置以及触发器）外，几乎所有SAR ADC配置均可由DSI信号实现。DSI模式和寄存器模式间的主要区别是：DSI模式允许硬件动态控制ADC配置。图21-14是SAR ADC框图（图21-1）的子集，它指定了DSI输入和输出信号。

图 21-14. DSI 控制模式框图



通过置位SAR_CTRL寄存器中的DSI_MODE位，可以选择DSI控制模式。在该模式下，SARSEQ会忽略CHAN_EN、CHAN_CONFIG、和INJ_CHAN_CONFIG中的所有通道配置。反而，它会使用由DSI信号传输的配置。

以下各 DSI 信号将被使用。

表 21-14. DSI 信号

信号	宽度	描述
sar_dsi_sample_done	1	用于指示 SAR ADC 采样操作已完成的脉冲。可以更改开关，以选择需要转换的下一个信号（与 SAR ADC 下个输出相同）
sar_dsi_chan_id_valid	1	通道 ID 的有效信号
sar_dsi_chan_id	4	常规模式：通道 ID（正在转换的通道 ID）（早期） DSI 控制模式： [0] = 饱和检测中断 [1] = 范围检测中断（发生数据输出时，该信号为有效）
sar_dsi_data_valid	1	数据值的有效信号
sar_dsi_data	12	某个通道转换（并求平均，若可用）的结果；内部平均结果为 16 位宽。 如果 dsi_data_hilo_sel = 0，则 sar_dsi_data[11:0] = sar_data[11:0]。 如果 dsi_data_hilo_sel = 1，则 sar_dsi_data[7:0] = sar_data[15:8]，且 sar_dsi_data[11:8] = < 未定义 >。
sar_dsi_eos_intr	1	用于表示 SARSEQ 已完成所有启用通道的扫描操作的“结束扫描”中断。
dsi_out	8	dsi_out[0]=1, P2.0 连接至 ADC dsi_out[1]=1, P2.1 连接至 ADC ... dsi_out[7] = 1, P2.7 连接至 ADC 注意：MUX_SWITCH0 的配置决定了该引脚被连接到 Vplus 还是 Vminus。
dsi_oe	4	dsi_oe[0] = 1, AMUXBUSA 连接至 ADC dsi_oe[1] = 1, AMUXBUSB 连接至 ADC dsi_oe[2] = 1, opamp0 输出连接至 ADC dsi_oe[3] = 1, opamp1 输出连接至 ADC 注意：MUX_SWITCH0 的配置决定了该信号被连接到 Vplus 还是 Vminus。
dsi_swctrl[0]	1	该信号控制着 SARMUX 模拟信号的转换情况，将 Vssa_kelvin 连接至 Vminus
dsi_swctrl[1]	1	该信号控制着 SARMUX 模拟信号的转换情况，将 temp_sens 连接至 Vplus
dsi_sw_negvref	1	该信号控制着 SAR ADC 内部信号的转换情况，将 VREF 输入连接至 NEG 输入
dsi_cfg_st_sel	2	DSI 控制模式的配置控制：选择 4 个全局采样时间中的一个
dsi_cfg_average	1	DSI 控制模式的配置控制：使能求平均功能
dsi_cfg_resolution	1	DSI 控制模式的配置控制：0 = 12 位的分辨率 1 = 使用全局配置的其它分辨率（8 位或 10 位）
dsi_cfg_differential	1	DSI 控制模式的配置控制：0 = 单端模式，1 = 差分模式
dsi_trigger	1	触发 SARSEQ 开始扫描所有启用的通道
dsi_data_hilo_sel	1	为 sar_dsi_data[7:0] 选择高或低字节输出。该信号是完全异步的（它影响着 sar_dsi_data 信号而不需要使用任何有关的时钟）。

21.3.11.1 固件模拟路由

在 DSI 模式下，通过 DSI 信号和固件，可以实现模拟路由。无论寄存器的配置如何，固件控制总是有效的，并且它在寄存器模式下也保持有效状态。请参考 21.3.2.1 模拟路由，了解有关固件控制的详细信息。

21.3.11.2 DSI 模拟路由

来自 UDB 模块的 DSI 信号用于控制 SARMUX 开关。在 DSI 控制模式下，SARSEQ 不会输出任何来自定序器使能的开关。图 21-4 显示 DSI 可以控制所有开关（除用于测试设计的

DTP 开关外）。因此，在该模式下，SAR ADC 的负输入和正输入可连接至任何开关。

除了 DSI 信号，寄存器中的相应硬件和固件控制位都需要置位。这些寄存器和信号包括 SAR_MUX_SWITCH0[n] = 1 以及 SAR_MUX_SWITCH_HW_CTRL[n] = 1。当 VREF 连接至负输入时，除了 DSI 信号以外，应设置 SAR_CTRL[11:9] = 7（固件控制字段）以及 SAR_CTRL[13] = 1（硬件控制位）。

在单端模式下，通过 dsi_swctrl[0] 和 dsi_sw_neg vref，DSI 信号能够控制 SAR ADC 的负端。如果置位了 NEG_SEL（SAR_CTRL[11:9]），则仅有 NEG_SEL=7 可用；其它数值都被忽略。

表 21-15 显示了 DSI 信号。

表 21-15. DSI 模拟路由

信号	宽度	描述
dsi_out	8	dsi_out[0]=1, P2.0 连接至 ADC dsi_out[1]=1, P2.1 连接至 ADC ... dsi_out[7] = 1, P2.7 连接至 ADC 注意: 该引脚连接至 Vplus 还是 Vminus 是由 MUX_SWITCH0 配置决定的。
dsi_oe	4	dsi_oe[0] = 1, AMUXBUSA 连接至 ADC dsi_oe[1] = 1, AMUXBUSB 连接至 ADC dsi_oe[2] = 1, sarbus0 输出连接至 ADC dsi_oe[3] = 1, sarbus1 输出连接至 ADC 注意: 该信号连接至 Vplus 还是 Vminus 是由 MUX_SWITCH0 配置决定的。
dsi_swctrl[0]	1	该信号控制着 SARMUX 模拟信号的转换情况, 将 V _{SSA} 连接至 Vminus
dsi_swctrl[1]	1	该信号控制着 SARMUX 模拟信号的转换情况, 将温度传感器连接至 Vplus
dsi_sw_negvref	1	该信号控制着 SAR ADC 内部信号的转换情况, 将 V _{REF} 输入连接至 NEG 输入端

21.3.11.3 全局 SARSEQ 配置

全局配置适用于寄存器模式以及 DSI 控制模式。更多详细信息, 请参考 21.3.10.2 全局 SARSEQ 配置。

21.3.11.4 DSI 通道配置

对于 DSI 控制模式, 只有通道 0 可用。通过 DSI 信号, 可执行通道 0 配置, 如表 21-16 所示。CHAN_EN 和 CHAN_CONFIG 与 INJ_CHAN_CONFIG 中的通道配置均被忽略。

通过设置 DSI_SYNC_CONFIG, 可以使 dsi_cfg_* 信号与 SAR 时钟域 (实际的 clk_hf) 同步。SAR 在低频率下运行时, 可能需要旁路同步。

表 21-16. DSI 通道配置

信号	宽度	配置	描述
dsi_cfg_st_sel	2	数据采集时间	DSI 控制模式的配置控制: 选择 4 个全局采样时间中的一个
dsi_cfg_average	1	求平均使能	DSI 控制模式的配置控制: 使能求平均功能
dsi_cfg_resolution	1	分辨率	DSI 控制模式的配置控制: 0: 12 位的分辨率 1: 使用全局配置分辨率位 SUB_RESOLUTION (8 位或 10 位)
dsi_cfg_differential	1	差分 / 单端	DSI 控制模式的配置控制: 0: 单端模式 1: 差分模式

21.3.11.5 中断

有关 SAR ADC 中断的介绍, 请查阅第 238 页上的中断屏蔽。所有中断屏蔽均正常工作于寄存器控制模式。并非所有中断都是在 DSI 模式中生成的; SATURATE_INTR、RANGE_INTR 以及 EOS_INTR 是通过 DSI 信号发送的。

- 数据以及 SATURATE_INTR 都是在 dsi_chan_id[0] 上输出的; 另外 SATURATE_INTR[0] 被设置为 DSI 控制模式, 因为在 DSI 模式中只有通道 0 可用。

- 数据以及 RANGE_INTR 都是在 dsi_chan_id[1] 上输出; RANGE_INTR[0] 在 DSI 控制模式下设置, 因为在 DSI 模式中只有通道 0 可用。
- 通道使能都被忽略, 即每次触发只能执行一次转换。为每个转换生成 EOS_INTR 中断。
- 始终通过 DSI 信号 sar_dsi_eos_intr (dsi_data_valid 的副本) 发送 EOS_INTR 中断。

表 21-17 列出了由 DSI 信号发送的中断。

表 21-17. DSI 信号中断

信号	宽度	描述
sar_dsi_chan_id	4	寄存器模式：通道 ID（正在转换的通道 ID） DSI 控制模式： [0] = 饱和检测中断； [1] = 范围检测中断（发生数据输出时，该信号为有效）
sar_dsi_eos_intr	1	用于表示 SARSEQ 已完成对所有启用通道的扫描操作的“扫描结束”中断。

21.3.11.6 触发器

DSI 控制模式通常与 DSI 触发器一起使用。但其它触发源（如固件触发和连续触发）也受支持。触发配置与寄存器控制模式中的配置相同。更多有关信息，请参考第 233 页上的触发器。

针对 DSI 触发，配置设置（dsi_cfg_*）以及开关设置应稳定，该设置不能迟于 dsi_trigger 被传送的周期。它们应保持稳定状态，直到 sar_dsi_sample_done 的上升沿到来为止。

21.3.11.7 检索数据

结果数据和通道编号都在 sar_dsi_data 上输出。它相当于在寄存器控制模式下将 dsi_out_en 置高。更多有关信息，请参考 21.3.11.8 DSI 输出使能。每当转换完成后，会将数据写入到 CHAN_WORK0 和 CHAN_RESULT0 寄存器中。

表 21-18. DSI 输出信号

信号	宽度	描述
sar_dsi_sample_done	1	用于指示 SAR ADC 采样操作已完成的脉冲。可以更改开关，以选择需要转换的下一个信号（与 SAR ADC 下个输出相同）
sar_dsi_chan_id_valid	1	通道 ID 的有效信号
sar_dsi_chan_id	4	常规模式：通道 ID（正在转换的通道 ID）（早期） DSI 控制模式： [0] = 饱和检测中断 [1] = 范围检测中断（发生数据输出时，该信号为有效）
sar_dsi_data_valid	1	数据值的有效信号
sar_dsi_data	12	一个通道的转换（和求平均，若可用）结果。内部平均值为 16 位宽。 如果 dsi_data_hilo_sel = 0，则 sar_dsi_data[11:0] = sar_data[11:0] 如果 dsi_data_hilo_sel = 1，则 sar_dsi_data[7:0] = sar_data[15:8]，且 sar_dsi_data[11:8] = < 未定义 >
sar_dsi_eos_intr	1	用于表示 SARSEQ 已完成所有使能通道的扫描操作的扫描结束中断。
dsi_data_hilo_sel	1	为 sar_dsi_data[7:0] 选择高或低字节输出。该信号是完全异步的（它对 sar_dsi_data 信号产生影响而不需要任何相关时钟）

21.3.12 模拟路由配置示例

表 21-19 显示的是定时器控制、固件控制和 DSI 控制的引脚和信号选项。

21.3.11.8 DSI 输出使能

如果置位了 DSI_OUT_EN 位（即 SAR_CHANx_CONFIG[31]），结果数据和通道编号也在 DSI 总线（sar_dsi_data、sar_dsi_chan_id）上输出，然后被存储在正常结果寄存器中。它允许 UDB 处理 SARSEQ 结果数据，并且通道数量允许对不同通道采用不同的处理方法。

在 DSI 总线上发出的数据格式与在结果寄存器中存储的数据格式相同。但在默认情况下，只输出 12 个最低有效位（LSB）；除非需要多于 12 位，否则不推荐使用左对齐。要想获取 8 个高最低有效位（LSB），需要将 dsi_data_hilo_sel 输入设置为‘1’。如果想从结果寄存器获取全 16 位数据，首先要将 dsi_data_hilo_sel 设置为‘0’，从而获得低 12 位数据，然后设置 dsi_data_hilo_sel = 1 以获得高 8 位数据。需要处理额外数据，以便处理数据重叠现象。

SAR ADC 完成通道采样后，会发出通道编号（sar_dsi_chan_id）。该通道的编号能够自身触发 UDB，以驱动某些 GPIO 引脚（这些引脚可以为片外器件供电或断电）。这样会允许随后的一个通道在同一扫描操作中对模拟输入引脚进行扫描（该操作需要一个较长的采样时间）。

请注意，数据在转换完成的一个周期后被输出。在 DSI 总线上两个系统时钟周期内维持通道编号、数据以及它们各自的有效位。

表 21-19. 模拟路由配置示例

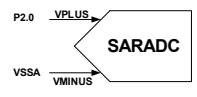
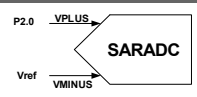
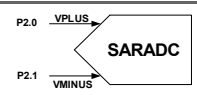
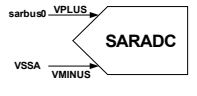
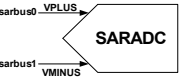
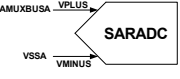
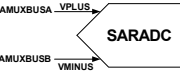
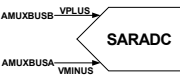
	定序器控制	固件控制	DSI 控制
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_out [0] =1 dsi_swctrl[0]=1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 7 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0]=1 HW_CTRL_NEGVREF =1 (CTRL[13])	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] =0 NEG_SEL = 7 (CTRL [11:9]) HW_CTRL_NEGVREF =0 (CTRL[13])	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 dsi_out [0] =1 dsi_sw_negvref =1 HW_CTRL_NEGVREF =1 (CTRL[13])
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 或 PIN_ADDR = 1 (CHANx_CONFIG[2:0]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[1] = 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[1] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_out [0] =1 dsi_out [1] =1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH0 [9] = 1 MUX_SWITCH_HW_CTRL[1]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 1 (CHANx_CONFIG[6:4]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] =1 MUX_SWITCH_HW_CTRL[16] =1 注意: 为了能够控制端口 / 引脚, 不支持将 sarbus1 连接至 VPLUS	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [2] =1 dsi_swctrl[0]=1 MUX_SWITCH0 [16] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH_HW_CTRL[22] =1

表 21-19. 模拟路由配置示例（续）

	定序器控制	固件控制	DSI 控制
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 1 (CHANx_CONFIG[6:4]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[23] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [2] = 1 dsi_oe [3] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 7 (CHANx_CONFIG[6:4]) PIN_ADDR = 2 (CHANx_CONFIG[2:0]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[16]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [0] = 1 dsi_swctrl[0]=1 MUX_SWITCH0[18] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 7 (CHANx_CONFIG[6:4]) PIN_ADDR = 2 (CHANx_CONFIG[2:0]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[19]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1
	不支持。 端口 / 引脚控制的差分对是固定的	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18]=0 MUX_SWITCH_HW_CTRL[19]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18]=1 MUX_SWITCH_HW_CTRL[19]= 1

21.3.13 温度传感器配置

一个片上温度传感器可适用于温度感应以及基于温度的校准。对于温度传感器，差分转换不可用（转换结果为未定义）。因此，该转换始终运行于单端模式。内部参考电压为 1.024 V。

可通过三种方法将某个引脚或信号连接到 SAR ADC。表 21-20 列出了将温度传感器连接至 SAR ADC 的方法。置位 MUX_FW_TEMP_VPLUS 位（即 SAR_MUX_SWITCH0[17]）可使能温度传感器，并且可将它的输出连接到 SAR ADC 的 VPLUS 上；如果清除该位，将通过切断温度传感器的偏置电流来禁用它。

表 21-20. 将温度传感器连接至 SAR ADC

控制方法	设置
定序器	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) PORT_ADDR = 7 (SAR_CHANx_CONFIG[6:4]) PIN_ADDR = 0 (SAR_CHANx_CONFIG[2:0]) SWITCH_DISABLE = 0 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16] = 1 SAR_MUX_SWITCH_HW_CTRL[17] = 1 NEG_SEL = 0 (SAR_CTRL [11:9]) 被覆盖为 0 ^a
固件	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) SWITCH_DISABLE = 1 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16] = 0 SAR_MUX_SWITCH_HW_CTRL[17] = 0 NEG_SEL = 0 (SAR_CTRL [11:9]) 被覆盖为 0 ^a
DSI	SWITCH_DISABLE = 1 (SAR_CTRL[30]) VREF_SEL = 0 (SAR_CTRL[6:4]) 设置 DSI 信号： dsi_cfg_differential=1 dsi_swctrl[1]=1 dsi_swctrl[0]=1 SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16] = 1 SAR_MUX_SWITCH_HW_CTRL[17] = 1 NEG_SEL = 0 (SAR_CTRL [11:9]) 被覆盖为 0 ^a

a. 对于温度传感器，需要覆盖 NEL_SEG (SAR_CTRL [11:9])，使之变为 ‘0’。

21.4 寄存器

名称	偏移地址	数量	宽度	描述
SAR_CTRL	0x0000	1	32	全局配置寄存器 模拟控制寄存器
SAR_SAMPLE_CTRL	0x0004	1	32	全局配置寄存器 采样控制寄存器
SAR_SAMPLE_TIME01	0x0010	1	32	全局配置寄存器 采样时间规范 ST0 和 ST1
SAR_SAMPLE_TIME23	0x0014	1	32	全局配置寄存器 采样时间规范 ST2 和 ST3
SAR_RANGE_THRES	0x0018	1	32	全局范围检测阈值寄存器
SAR_RANGE_COND	0x001C	1	32	全局范围检测模式寄存器
SAR_CHAN_EN	0x0020	1	32	通道使能位
SAR_START_CTRL	0x0024	1	32	启动控制寄存器（固件触发）
SAR_CHAN_CONFIG	0x0080	8	32	通道配置寄存器
SAR_CHAN_WORK	0x0100	8	32	通道工作数据寄存器
SAR_CHAN_RESULT	0x0180	8	32	通道结果数据寄存器
SAR_CHAN_WORK_VALID	0x0200	1	32	通道工作数据寄存器的有效位
SAR_CHAN_RESULT_VALID	0x0204	1	32	通道结果数据寄存器的有效位
SAR_STATUS	0x0208	1	32	内部 SAR 寄存器的当前状态（用于调试）
SAR_AVG_STAT	0x020C	1	32	当前的平均值状态（用于调试）
SAR_INTR	0x0210	1	32	中断请求寄存器
SAR_INTR_SET	0x0214	1	32	中断设置请求寄存器
SAR_INTR_MASK	0x0218	1	32	中断屏蔽寄存器
SAR_INTR_MASKED	0x021C	1	32	中断屏蔽请求寄存器：如果该值为非零值，则输入到 NVIC 的 SAR 中断信号为高电平。读取时，该寄存器反映中断请求和屏蔽寄存器之间的按位逻辑与（AND）运算
SAR_SATURATE_INTR	0x0220	1	32	饱和中断请求寄存器
SAR_SATURATE_INTR_SET	0x0224	1	32	饱和中断设置请求寄存器
SAR_SATURATE_INTR_MASK	0x0228	1	32	饱和中断屏蔽寄存器
SAR_SATURATE_INTR_MASKED	0x022C	1	32	饱和中断屏蔽请求寄存器
SAR_RANGE_INTR	0x0230	1	32	范围检测中断请求寄存器
SAR_RANGE_INTR_SET	0x0234	1	32	范围检测中断设置请求寄存器
SAR_RANGE_INTR_MASK	0x0238	1	32	范围检测中断屏蔽寄存器
SAR_RANGE_INTR_MASKED	0x023C	1	32	范围中断屏蔽请求寄存器
SASR_INTR_CAUSE	0x0240	1	32	中断源寄存器
SAR_INJ_CHAN_CONFIG	0x0280	1	32	插入通道配置寄存器
SAR_INJ_RESULT	0x0290	1	32	插入通道结果寄存器
SAR_MUX_SWITCH0	0x0300	1	32	SARMUX 固件开关控制
SAR_MUX_SWITCH_CLEAR0	0x0304	1	32	SARMUX 固件开关控制清除
SAR_MUX_SWITCH_HW_CTRL	0x0340	1	32	SARMUX 硬件开关控制
SAR_MUX_SWITCH_STATUS	0x0348	1	32	SARMUX 开关状态
SAR_PUMP_CTRL	0x0380	1	32	开关泵控制

22. 低功耗比较器



PSoC[®] 4 具有两个低功耗比较器。这些比较器可以在所有的系统功耗模式（停止模式除外）下快速进行模拟信号比较。有关器件的各种功耗模式的详细信息，请参见第 97 页上的功耗模式章节。可将正向和负向输入连接到专用的 GPIO 引脚或 AMUXBUS-A/AMUXBUS-B。CPU 通过状态寄存器读取该比较器输出。该输出能作为中断源或唤醒源使用，此外，它还可以反馈到 DSI 进行处理或路由到 GPIO。

22.1 特性

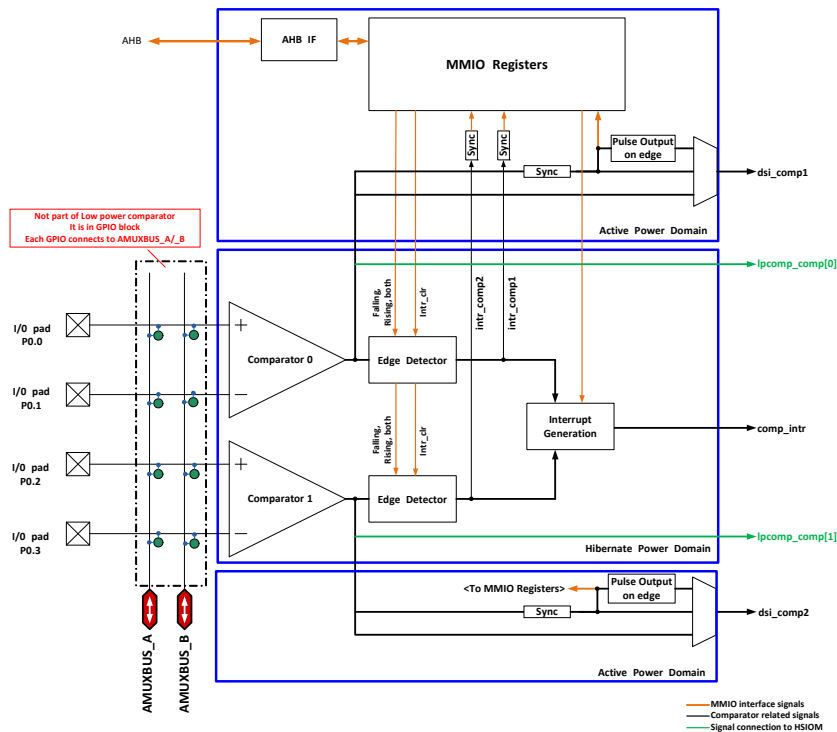
PSoC 4 比较器具有以下特性：

- 可配置的正向和负向输入
- 可编程的功耗和速度
- 支持超低功耗模式（< 4 μ A）
- 可选的 10 mV 输入迟滞
- 低输入偏移电压（调整后该电压低于 4 mV）
- 支持在深度睡眠 / 休眠模式下的唤醒源

22.2 框图

图 22-1 显示了低功耗比较器的框图。

图 22-1. 低功耗比较器框图



22.3 工作原理

下面各部分介绍了 PSoc 4 低功耗比较器的操作情况，包括输入配置、功耗和速度模式、输出和中断配置、迟滞、从低功耗模式唤醒、比较器时钟以及偏移调整。

22.3.1 输入配置

可用的比较器输入包括：

- 专用输入引脚的正向和负向输入。
- 通过 AMUXBUS 的任何引脚的正向和负向输入（在深度睡眠模式下不可用）。
- 一个来自外部引脚的输入和一个来自内部生成的信号的输入。可以将两个输入连接到比较器的正向或负向输入。通过模拟 AMUXBUS，将内部生成的信号连接到比较器输入端。
- 内部生成的信号的正向和负向输入。通过 AMUXBUS-A/AMUXBUS-B，将内部产生的信号连接到比较器输入。

通过图 22-1 可知，P0.0 和 P0.1 连接到比较器 0 的正向和负向输入；P0.2 和 P0.3 连接着比较器 1 的输入。另外还要注意：AMUXBUS 网络并不直接连接比较器输入。因此，比较器连接是通过相应的输入引脚被路由到 AMUXBUS 网络的。将 AMUXBUS 用于比较器连接时，这些输入引脚不可用于其他目的。在 AMUXBUS 被用于比较器输入连接的设计中，它们应该保持为开路状态。请注意，在深度睡眠和休眠模式下 AMUXBUS 连接不可用。如果需要在深度睡眠或休眠模式下进行操作，则必须将低功耗比较器连接到专用引脚上。该限制还包括任何内部生成信号的路由，因为它们也通过 AMUXBUS 进行连接。请参见第 63 页上的 I/O 系统章节，以了解 GPIO 到 AMUXBUS A/B 的连接或比较器输入的 GPIO 设置。

22.3.2 输出和中断配置

在 LPCOMP_CONFIG 寄存器中（表 22-1），OUT1 位 [6] 和 OUT2 位 [14] 分别提供了比较器 0 和比较器 1 的输出。在将比较器输出锁存到 LPCOMP_CONFIG 寄存器中的 OUTx 位前，必须将这些输出与 SYSCLK 进行同步。每个比较器的输出被连接到一个对应的边沿检测器模块。该模块会确定触发中断的边沿。通过 LPCOMP_CONFIG 寄存器中的 INTTYPE1 位 [5:4] 和 INTTYPE2 位 [13:12]，可以配置边沿选择和中断使能。通过使用 INTTYPEx 位，可以将中断类型设置为禁用、上升沿、下降沿，或双边沿，如表 22-1 所示。

通过 HSIOM，可以将每个比较器的输出直接路由到 GPIO 引脚上。在 HSIOM 中，比较器输出作为深度睡眠模式的源 2 连接。有关 HSIOM 的详细信息，请参考第 72 页上的高速输入/输出矩阵。有关支持低功耗比较器输出的引脚的详细信息，请参考器件数据手册。这些引脚上的输出是直接从比较器输出的，并且它们不同步。因为这些输出可作为深度睡眠资源使用，所以比较器输出也可用于深度睡眠模式。请注意，HSIOM 在休眠模式下不可用。因此，受支持引脚上的比较器输出在休眠模式下亦不可用。

另外，通过 DSI（即图 22-1 中的 dsi_comp1/dsi_comp2 信号），可以将比较器输出单独路由到引脚或其他模块。可将连接至 DSI 的比较器输出配置为异步输出，与 SYSCLK 同步的输出，或比较器输出上升沿上的同步脉冲输出。LPCOMP_CONFIG 寄存器的 DSI_BYPASS1 位 [16] 和 DSI_LEVEL1 位 [17] 用于将比较器 0 的输出配置为 DSI。同样，LPCOMP_CONFIG 寄存器中的 DSI_BYPASS2 位 [20] 和 DSI_LEVEL2 位 [21] 用于将比较器 1 的输出配置为 DSI。更多详细信息，请参见表 22-1 中的内容。

在一个边沿事件中，比较器会触发一个中断（图 22-1 中的 intr_comp1/intr_comp2 信号）。比较器 0 和比较器 1 的中断请求分别被寄存在 LPCOMP_INTR 寄存器的 COMP1 位 [0] 和 COMP2 位 [1] 上。比较器 0 和比较器 1 共享了一个常用中断（即图 22-1 中的 comp_intr 信号）。该中断是由两个中断经过逻辑 OR 运算得到的，并被映射为 CPU NVIC 中低功耗比较器模块的中断。更多详细信息，请参考第 51 页上的中断章节。如果在设计中使用了这两个比较器，则必须在中断服务子程序中读取 LPCOMP_INTR 寄存器的 COMP1 和/或 COMP2 位，以确定触发中断的比较器。另外，通过使用 LPCOMP_INTR_MASK 寄存器的 COMP1_MASK 位 [0] 和 COMP2_MASK 位 [1]，可以屏蔽 CPU 的比较器 0 和比较器 1 中断。其中，CPU 只会处理屏蔽中断。处理中断后，需要将‘1’写入到固件中 LPCOMP_INTR 寄存器的 COMP1 和 COMP2 位，以清除该中断。如果未清除该中断，那么下一个比较事件将不会触发中断，并且 CPU 将无法处理该事件。在活动和睡眠模式下，dsi_comp1/dsi_comp2 输出会被路由到 UDB 映射中断，以单独处理每个比较器的触发。但在深度睡眠和休眠模式下，不能实现 UDB/DSI 路由。

LPCOMP 中断（comp1_intr/comp2_intr）与 SYSCLK 同步。清除 dsi_comp1/dsi_comp2 和 comp1_intr/comp2_intr 的操作是同步进行的。

在活动和睡眠模式下，通过 UDB 中的 DSI 路由可以将 dsi_comp1/dsi_comp2 路由到 GPIO 或其他模块（需要 / 无需实现同步化）。UDB DSI 输出有一个附加的同步器。有关 DSI 信号同步的详细信息，请参见第 113 页上的通用数字模块（UDB）章节。在深度睡眠和休眠模式下，由于 UDB 被断电，所以不能进行上述路由操作。此外，如果将 dsi_comp1/dsi_comp2 路由到 UDB 以进行后续的处理操作，那么，时序则由用户的算法和同步器的选择决定。

通过使用 LPCOMP_INTR_SET 寄存器中的 [1:0] 位，可以激活一个软件调试的中断。

在深度睡眠和休眠模式下，可以通过比较器边沿事件来激活唤醒中断控制器（WIC），进而唤醒 CPU。LPCOMP 因此在低功耗模式下仍然可以监控一个指定的信号。

表 22-1. LPCOMP_CONFIG 寄存器中的输出和中断配置

寄存器 [位_位置]	位_名称	说明
LPCOMP_CONFIG[6]	OUT1	比较器 0 的当前 / 瞬间输出值
LPCOMP_CONFIG[14]	OUT2	比较器 1 的当前 / 瞬间输出值
LPCOMP_CONFIG[5:4]	INTTYPE1	设置比较器 0 触发 IRQ 的边沿 00: 禁用 01: 上升沿 10: 下降沿 11: 上升沿和下降沿
LPCOMP_CONFIG[13:12]	INTTYPE2	设置比较器 1 触发 IRQ 的边沿 00: 禁用 01: 上升沿 10: 下降沿 11: 上升沿和下降沿
LPCOMP_CONFIG[16]	DSI_BYPASS1	比较器 0 旁路 DSI 输出的输出同步 0: 输出与 SYSCLK 同步 1: 旁路 / 输出异步
LPCOMP_CONFIG[17]	DSI_LEVEL1	比较器 0 的 DSI 输出电平 0: 脉冲输出 — 在上升沿上生成宽度为两个 SYSCLK 周期的脉冲 1: 电平
LPCOMP_CONFIG[20]	DSI_BYPASS2	比较器 1 旁路 DSI 输出的输出同步 0: 输出与 SYSCLK 同步 1: 旁路 / 输出异步
LPCOMP_CONFIG[21]	DSI_LEVEL2	比较器 1 的 DSI 输出电平 0: 脉冲输出 — 在上升沿上生成宽度为两个 SYSCLK 周期的脉冲 1: 电平
LPCOMP_INTR[0]	COMP1	比较器 0 中断: 硬件在比较器 0 触发时设置该中断。通过写入 ‘1’ 来清除中断。
LPCOMP_INTR[1]	COMP2	比较器 2 中断: 硬件在比较器 1 触发时设置该中断。通过写入 ‘1’ 来清除中断。
LPCOMP_INTR_SET[0]	COMP1	通过写入 ‘1’ 来触发比较器 0 的软件中断。
LPCOMP_INTR_SET[1]	COMP2	通过写入 ‘1’ 来触发比较器 1 的软件中断。

22.3.3 功耗模式和速度配置

低功耗比较器能够在以下三种模式下运行：

- 快速
- 慢速
- 超低功耗

通过 LPCOMP_CONFIG 寄存器中的 MODE1 位 [1:0]，配置了比较器 0 的速度设置或功耗设置。通过同一个寄存器中的 MODE2 位 [9:8]，配置了比较器 1 的速度设置或功耗设置。功耗和响应时间会根据所选功耗模式而不同：在快速模式下，功耗最高和响应时间最快；在超低功耗模式下，功耗最低和响应时间最慢。请参见 器件数据手册，了解在各种功耗设置下响应时间和功耗的规范。

通过 LPCOMP_CONFIG 寄存器中的 ENABLE1 位 [7] 和 ENABLE2 位 [15] 可以使能 / 禁用比较器，如表 22-2 所示。

请注意：当比较器在使能时，如果更改功耗模式，比较器输出可能会发生短时脉冲。为了避免发生这种情况，需要在更改功耗模式前禁用比较器。

表 22-2. 比较器功耗模式选择位 MODE1 和 MODE2

寄存器 [位_位置]	位_名称	说明
LPCOMP_CONFIG[1:0]	MODE1	比较器 0 功耗模式选择 00: 慢速工作模式 (功耗较低) 01: 快速工作模式 (功耗较高) 10: 超低功耗工作模式 (功耗最低)
LPCOMP_CONFIG[9:8]	MODE2	比较器 1 功耗模式选择 00: 慢速工作模式 (功耗较低) 01: 快速工作模式 (功耗较高) 10: 超低功耗工作模式 (功耗最低)
LPCOMP_CONFIG[7]	ENABLE1	比较器 0 使能位 0: 禁用比较器 0 1: 使能比较器 0
LPCOMP_CONFIG[15]	ENABLE2	比较器 1 使能位 0: 禁用比较器 1 1: 使能比较器 1

22.3.4 迟滞

对于比较相邻信号或缓慢更改信号的应用，迟滞有助于避免信号嘈杂时在比较器输出上产生的振荡。对于这种应用，在比较器模块中将使能固定的 10 mV 迟滞电平。

通过使用 LPCOMP_CONFIG 寄存器中的 HYST1 位 [2] 和 HYST2 位 [10] 可以使能 / 禁用 10 mV 迟滞电平，如表 22-3 所示。

表 22-3. 迟滞控制位 HYST1 和 HYST2

寄存器 [位_位置]	位_名称	说明
LPCOMP_CONFIG[2]	HYST1	使能 / 禁用比较器 0 的 10 mV 迟滞 - 0: 使能迟滞 - 1: 禁用迟滞
LPCOMP_CONFIG[10]	HYST2	使能 / 禁用比较器 1 的 10 mV 迟滞 - 0: 使能迟滞 - 1: 禁用迟滞

22.3.5 从低功耗模式唤醒

比较器可以在器件的低功耗模式（睡眠、深度睡眠和休眠模式）下运行。比较器输出中断可将器件从睡眠、深度睡眠和休眠模式唤醒。为了从低功耗模式中唤醒器件，必须通过 LPCOMP_CONFIG 寄存器使能比较器，而且不能将 LPCOMP_CONFIG 寄存器中的 INTTYPE_x 位设置为禁用，另外需要置位 LPCOMP_INTR_MASK 寄存器中相对应的比较器的 INTR_MASK_x 位。在深度睡眠和休眠模式下，不可用的性能包括：

- 使用 AMUXBUS 连接进行的比较
- 通过 DSI 路由比较器输出

在深度睡眠和休眠模式下，发生在比较器 0 或比较器 1 输出的比较事件均可生成唤醒中断。应该根据要求配置 LPCOMP_CONFIG 寄存器中的 INTTYPE_x 位，以使使相应的比较器将器件从低功耗模式唤醒。通过使用 LPCOMP_INTR_MASK 寄存器中的屏蔽位，可以选择 CPU 处理一个还是两个比较器中断。

22.3.6 比较器时钟

比较器将系统主时钟 SYSCLK 作为中断同步时钟使用。

22.3.7 偏移调整

比较器的偏移在工厂内被调整为 4.0 mV 以下。调整过程分两步：依次在通用模式下的电压为 0.1 V 和 $V_{DD}-0.1$ V 时进行调整。对于输入电压范围为 0.1 V 到 $V_{DD}-0.1$ V 的输入，偏移电压必须低于 10.0 mV。对于正常操作模式，不建议使用超过 10.0 mV 的偏移电压。

如果需要准确调整为特定的输入共模电压，则请执行所需操作。通过使用 LPCOMP_TRIM1/2/3/4 寄存器，可以进行比较器偏移调整。LPCOMP_TRIM1 和 LPCOMP_TRIM2 用于调整比较器 0。LPCOMP_TRIM3 和 LPCOMP_TRIM4 用于调整比较器 1。用于更改调整值的位字段是 LPCOMP_TRIM1 和 LPCOMP_TRIM3 的 TRIMA 位 [4:0] 以及 LPCOMP_TRIM2 和 LPCOMP_TRIM4 的 TRIMB 位 [3:0]。TRIMA 位用于粗调偏移，而 TRIMB 位用于微调偏移。只能在比较器的慢速模式下使用 TRIMB 位来纠正偏移。

可以使用任何标准比较器偏移调整程序进行调整。通过使用以下方法可以提高参考 / 共模电压输入中的偏移电压。

1. 对比较器输入进行外部短接，并将电压参考 V_{ref} 连接到输入。
2. 设置比较器（以便进行比较），关闭迟滞并检查输出。
3. 如果该输出为高电平，则偏移将为正向。否则，该偏移为负向。请按照以下步骤调整偏移：
 - a. 调校 TRIMA 位 [4:0]，直到输出方向发生偏转为正。TRIMA 位 [3:0] 控制偏移量，TRIMA 位 [4] 则控制偏移的极性（其中，‘1’表示正偏移，‘0’表示负偏移）。
 - b. 当完成调校 TRIMA 位时，应一直调校 TRIMB 位 [3:0]，直到输出再次转换方向为止。TRIMB 位的调校只能在比较器的慢速模式下进行。TRIMB 位 [3] 控制着偏移的极性。增加 TRIMB 位 [2:0] 会减少偏移。
 - c. 完成 3-b 步骤之后，TRIMA 和 TRIMB 位的值将为该特定 V_{ref} 最接近的调整值。

22.4 寄存器汇总

表 22-4. 低功耗比较器寄存器汇总

寄存器	功能
LPCOMP_ID	包含 LPCOMP 控制器 ID 和版本编号的信息
LPCOMP_CONFIG	LPCOMP 配置寄存器
LPCOMP_INTR	LPCOMP 中断寄存器
LPCOMP_INTR_SET	LPCOMP 中断设置寄存器
LPCOMP_INTR_MASK	LPCOMP 中断请求屏蔽寄存器
LPCOMP_INTR_MASKED	LPCOMP 屏蔽中断输出寄存器
LPCOMP_TRIM1	比较器 0 的调整字段
LPCOMP_TRIM2	比较器 0 的调整字段
LPCOMP_TRIM3	比较器 1 的调整字段
LPCOMP_TRIM4	比较器 1 的调整字段

23. 微型连续时间模块（CTBm）



微型连续时间模块（CTBm）在芯片内提供了离散运算放大器（opamps），以用于连续时间信号链路。每个 CTBm 模块包括一个用于输入 / 输出配置的开关矩阵、两个相同的运算放大器（也可以将它们配置为两个比较器）、每个运算放大器中的一个电荷泵，另外还包括用于比较器输出路由、开关控制和中断的数字接口。PSoC 4200L 系列具有两个 CTBm 模块，即四个离散运算放大器。另外，该 CTBm 模块还可以在深度睡眠的功耗模式下工作。

23.1 特性

PSoC 4 CTBm 模块中的运算放大器具有以下特性：

- 离散、高性能并且高度可配置的片上放大器
- 可编程功耗、带宽、补偿和输出驱动强度
- 可选输出电流的驱动能力为 1 mA 或 10 mA
- 对于 20 pF 负载，增益带宽为 6 MHz
- 调整偏移低于 1 mV
- 支持运算放大器的 Follower 模式
- 比较器模式带有可选的 10 mV 迟滞
- SAR 输入的缓冲器 / 前置放大器
- 对于 1 mA 负载，轨至轨电压范围为 V_{SS} 至 V_{DDA} ，误差为 ± 0.2 V
- 对于 10 mA 负载，轨至轨范围为 V_{SS} 至 V_{DDA} ，误差为 ± 0.5 V
- 对于 50 pF 负载，转换速率为 4 V/ μ s
- 支持器件的深度睡眠模式

23.2 框图

图 23-1 显示的是 PSoC 4 器件中 CTBmx — CTBm0 和 CTBm1 模块的框图。

Legend:

- Switch: CTBm Register control
- Switch: CTBm Register + SARADC register+ DSI control

Note: 'Pn' – 'P1' for CTBm0 and 'P5' for CTBm1
 'x' (in CTBMx) – '0' for CTBm0 and '1' for CTBm1

Note: 10X or 1X output driver cannot be on at the same time.

如框图所示，每个 CTBm 由两个相同的运算放大器和一个开关连接阵列组成。每个运算放大器包括一个输入级和三个输出级，它们共享了一个公用输入级（如图 23-1 中所示）；每次只能选中其中一个输出级。输出级可以作为 A 级（1X）、AB 级（10X）或比较器运行。另外，还可以配置其他特性，如：功耗与速度、补偿和开关连接控制。

使能运算放大器和电荷泵后，请按照下面的步骤设置放大器：

- 请按照下面步骤设置比较器:

1. 配置功耗模式
2. 配置输入开关
3. 根据要求配置比较器输出电路、中断生成以及 DSI 输出，等等
4. 配置迟滞并使能比较器

运算放大器可以在下面三个功耗模式下运行，即：低功耗、中等功耗以及高功耗。CTBm 可以通过调整运算放大器的参考电流来调整功耗。通过 CTBMx_OA_RESy_CTRL 内的 PWR_MODE 位 [1:0]，可以配置功耗模式。转换速率和增益带宽在高功耗模式下为最大值，而在低功耗模式下为最小值。请注意，功耗模式的配置也会影响到 1X 模式下的最大输出驱动能力 (I_{OUT})。更多详细信息，请参考表 23-1。欲了解各种功耗模式下的增益带宽、转换速率以及 I_{OUT} 的规格，请参阅器件的数据手册内容。

注意：‘y’ 表示 CTBMx 中的 Opamp1/0（y = 1 表示 Opamp1；y = 0 表示 Opamp0）。

23.3.2 输出驱动配置

可将每个运算放大器的输出驱动器配置为内部驱动器（A 级 /1X 驱动器）或外部驱动器（AB 级 /10X 驱动器）。1X 和 10X 驱动器是相互排斥的，即不能同时启用。1X 输出驱动器适合于以较高速度驱动较小的片上电容性和电阻性负载。10X 输出驱动器则适用于驱动较大的片外电容性和电阻性负载。分别将 1X 和 10X 驱动器输出连接至 **sarbus 0/1** 和外部引脚。每个驱动器都有低等、中等和高等三种功耗模式，如表 23-1 所示。

表 23-1. 输出驱动器与功耗模式

功耗模式 I _{OUT} 驱动能力	CTBMx_OA_RESy_CTRL[1:0]			
	00（禁用）	01（低功耗）	10（中等功耗）	11（高功耗）
外部驱动器（10X）	关闭	10 mA	10 mA	10 mA
内部驱动器（1X）	关闭	100 μ A	400 μ A	1 mA

通过 CTBMx_OA_RESy_CTRL[2] 位，可以选择 10X 或 1X 的输出能力（0: 1X，1: 10X）。如果运算放大器的输出被连接到 SAR ADC，则建议选择 1X 输出驱动器。如果运算放大器的输出被连接到某个外部引脚，则建议选择 10X 输出驱动器。在特殊情况下，如果需要通过 1X 输出驱动器将输出连接至外部引脚，或通过 10X 输出驱动器将输出连接至内部负载（例如，SAR ADC），可以将 CTBMx_OAy_SW [21] 置为 ‘1’。然而，赛普拉斯不能保证此情况下的性能。

表 23-2 总结了用于配置运算放大器输出驱动强度和功耗模式的各个位。

表 23-2. CTBM 寄存器中的输出强度和功耗模式配置

寄存器 [位 _ 位置]	位 _ 名称	描述
CTBMx_CTB_CTRL[31]	ENABLE	CTBMx 功耗模式选择 0: CTBM 被禁用 1: CTBM 被使能
CTBMx_OA_RES0_CTRL [11]	OA0_PUMP_EN	Opamp0 泵使能位 0: Opamp0 泵在 CTBMx 中被禁用 1: Opamp0 泵在 CTBMx 中被使能
CTBMx_OA_RES1_CTRL [11]	OA1_PUMP_EN	Opamp1 泵使能位 0: Opamp1 泵在 CTBMx 中被禁用 1: Opamp1 泵在 CTBMx 中被使能
CTBMx_OA_RES0_CTRL [1:0]	OA0_PWR_MODE	Opamp0 功耗模式选择位 00: Opamp0 在 CTBMx 中被关闭 01: Opamp0 在 CTBMx 中为低功耗模式 10: Opamp0 在 CTBMx 中为中等功耗模式 11: Opamp0 在 CTBMx 中为高功耗模式
CTBMx_OA_RES1_CTRL [1:0]	OA1_PWR_MODE	Opamp1 功耗模式选择位 00: Opamp1 在 CTBMx 中被关闭 01: Opamp1 在 CTBMx 中为低功耗模式 10: Opamp1 在 CTBMx 中为中等功耗模式 11: Opamp1 在 CTBMx 中为高功耗模式
CTBMx_OA_RES0_CTRL [2]	OA0_DRIVE_STR_SEL	Opamp0 输出驱动强度选择位 0: Opamp0 在 CTBMx 中的输出驱动强度为 1X 1: Opamp0 在 CTBMx 中的输出驱动强度为 10X
CTBMx_OA_RES1_CTRL [2]	OA1_DRIVE_STR_SEL	Opamp1 输出驱动强度选择位 0: Opamp1 在 CTBMx 中的输出驱动强度为 1X 1: Opamp1 在 CTBMx 中的输出驱动强度为 10X

23.3.3 补偿

每个运算放大器还有一个可编程补偿的电容模块，允许根据输出负载来优化运算放大器性能的稳定性。通过相应的 CTBMx_OAy_COMP_TRIM 寄存器，可以控制每个运算放大器的补偿，如在表 23-3 中所示。请注意，器件数据手册中的所有 GBW 和转换速率规格可用于所有补偿调整。

表 23-3. CTBM 中的 Opampy（Opamp0 或 Opamp1）补偿位

寄存器 [位_位置]	位_名称	描述
CTBMx_OAy_COMP_TRIM[1:0]	OAy_COMP_TRIM	Opampy 补偿调整位
		00: 无补偿
		01: CTBMx 中有最小补偿，高速度和低稳定性
		10: CTBMx 中有中度补偿，平衡的速度和稳定性
		11: CTBMx 中有最大补偿，低速度和高稳定性

23.3.4 开关控制

每个 CTBm 均有许多用于配置运算放大器的输入和输出的开关。通过配置 CTBm 寄存器（CTBMx_OA0_SW、CTBMx_OA1_SW），可以控制几乎所有开关，除了三个用于将运算放大器输出通过 sarbus0 和 sarbus1 连接至 SAR ADC 的开关以外。需要通过 SAR ADC 寄存器、CTBm 寄存器和 DSI 信号来控制这三个开关。

通过设置 CTBMx_OAy_SW 寄存器中的相应位，可以关闭开关；而清除这些位时会使相应开关被打开。向 CTBMx_OAy_SW_CLEAR 写入 ‘1’ 时，可以清除 CTBMx_OAy_SW 中的相应位。更多有关这些开关和它们所使能的连接的信息，请参考 PSoc 4200L 系列：PSoc 4 寄存器技术参考手册。

23.3.4.1 输入配置

模拟开关为运算放大器的正和负输入选择提供了一些选项。这些开关会连接至运算放大器输入（来自外部引脚或 AMUX 总线），或构成一个局部反馈回路（用于缓冲功能）。每个运算放大器上的开关都将连接两个 AMUXBUS 线中的一个：Opamp0 和 Opamp1 分别连接至每个 CTBm 模块中的 AMUXBUS-A 和 AMUXBUS-B。

注意：要确保正输入和负输入路径中只有一个开关被关闭；否则，不同的输入源将被互相短接。

- 正输入：模拟开关为每个 CTBm 的 opamp0 和 opamp1 提供了三个正输入选项，分别为两个外部引脚和一个 AMUXBUS 线。更多有关信息，请参考表 23-4。

表 23-4. 正输入选择

	正输入	开关控制位	描述
Opamp0	AMUXBUS A	CTBMx_OA0_SW [0]	0: 打开开关； 1: 关闭开关
	Pn.0 ^a	CTBMx_OA0_SW [2]	0: 打开开关； 1: 关闭开关
	Pn.6	CTBMx_OA0_SW [3]	0: 打开开关； 1: 关闭开关
Opamp1	AMUXBUS B	CTBMx_OA1_SW [0]	0: 打开开关； 1: 关闭开关
	Pn. 5	CTBMx_OA1_SW [1]	0: 打开开关； 1: 关闭开关
	Pn.7	CTBMx_OA1_SW [4]	0: 打开开关； 1: 关闭开关

a. Pn = P1（对于 CTBm0）以及 P5（对于 CTBm1）

- 负输入：模拟开关为每个 CTBm 的 opamp0 和 opamp1 提供了两个负输入选项，即为一个外部引脚或输出反馈（具体由 CTBMx_OAy_SW 寄存器控制）。表 23-5 显示的是控制位的详细信息。

表 23-5. 负输入选择

	负输入	开关控制位	描述
Opamp0	Pn.1 ^a	CTBMx_OA0_SW [8]	0: 打开开关； 1: 关闭开关
	Opamp0 输出反馈经过 1X 输出驱动器	CTBMx_OA0_SW [14]	0: 打开开关； 1: 关闭开关
Opamp1	Pn.4	CTBMx_OA1_SW [8]	0: 打开开关； 1: 关闭开关
	Opamp1 输出反馈经过 1X 输出驱动器	CTBMx_OA1_SW [14]	0: 打开开关； 1: 关闭开关

a. Pn = P1（对于 CTBm0）以及 P5（对于 CTBm1）

23.3.4.2 输出配置

每个运算放大器的输出被直接连接到一个固定引脚上；不求其它设置。通过三个开关 (SW1/2/3)，可以选择将它连接至 sarbus0 或 sarbus1。每个 CTBm 的 Opamp0 输出可以连接至 sarbus0，而 opamp1 输出可以连接至 sarbus0 或 sarbus1。通过 sarbus0 和 sarbus1 可以将运算放大器输出连接至 SAR ADC 输入复用器。连接至 sarbus 的这三个输出路由由开关由 CTBm 寄存器、SAR ADC 寄存器和 DSI 信号一起控制；其他开关则仅由 CTBm 寄存器控制。

下面的真值表 (表 23-6, 表 23-7 和表 23-8) 显示了三个开关的控制逻辑。PORT_ADDR、PIN_ADDR 和 DIFFERENTIAL_EN 分别源于 SAR_CHANx_CONFIG [6:4]、SAR_CHANx_CONFIG [2:0] 和 SAR_CHANx_CONFIG [2:0]。PORT_ADDR = 0 或 PIN_ADDR = 0 都会导致 SW[n] = 0。使用 SAR 寄存器或 DSI 信号来控制开关时，需要设置 CTBMx_SW_HW_CTRL 位 [2] 或 [3]。如果 CTBMx_OAy_SW[18]/[19] = 0 并且 SW[n] = 0，CTBMx_OAy_SW[18]/[19] 可以屏蔽其他控制位。

寄存器 CTBMx_SW_STATUS [30:28] 显示了 SW1/2/3 的当前开关状态。

表 23-6. SW1 控制逻辑的真值表

PORT_ADDR	PIN_ADDR	CTBMx_SW_HW_CTRL[2]	dsi_out[2]	CTBMx_OA0_SW[18]	SW1
X	X	X	X	0	0
X	0	1	0	1	0
0	X	1	0	1	0
X	X	X	1	1	1
X	X	0	X	1	1
1	2	X	X	1	1

表 23-7. SW2 控制逻辑的真值表

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTBMx_SW_HW_CTRL[3]	dsi_out[3]	CTBMx_OA0_SW[18]	SW2
X	X	X	X	X	0	0
X	X	0	1	0	1	0
X	0	X	1	0	1	0
1	X	X	X	0	1	0
X	X	X	0	X	1	1
X	X	X	X	X	1	1
0	1	3	X	X	1	1

表 23-8. SW3 控制逻辑的真值表

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTBMx_SW_HW_CTRL[3]	dsi_out[3]	CTBMx_OA0_SW[18]	SW3
X	X	X	X	X	0	0
X	X	0	1	0	1	0
X	0	X	1	0	1	0
0	X	X	X	0	1	0
X	X	X	0	X	1	1
X	X	X	X	X	1	1
1	1	2	X	X	1	1

23.3.4.3 比较器模式

通过置位相应的 CTBMx_OA_RESy_CTRL[4] 位，可以将每个运算放大器配置为比较器。请注意，使能比较器后会完全禁用补偿电容，并会关闭 A 级（1X）和 AB 级（10X）输出驱动器。该比较器具有以下特性：

- 可选的 10 mV 输入迟滞
- 可配置的功耗 / 速度
- 可选的 DSI 输出同步
- 偏移被调整到低于 1 mV 的值
- 可配置的边沿检测（上升沿 / 下降沿 / 两者 / 禁用）

23.3.4.4 比较器配置

可以单向（从低到高）使能 ‘10 mV \pm 5%’ 迟滞。通过设置 CTBMx_OA_RESy_CTRL[5]，可以使能输入迟滞。通过设置 CTBMx_OA_RESy_CTRL [1:0]，可以将每个 CTBm 模块中的两个比较器配置为低、中等和高等三种功耗模式。根据不同的功耗模式，响应时间和功耗也不一样；快速模式下的功耗最高，而超低功耗模式下的功耗最低。本数据手册中提供了有关功耗和响应时间的详细规范。

比较器输出可选同步连接至 DSI。通过 CTBMx_OA_RESy_CTRL[6]，可以配置比较器时钟（系统 AHB 时钟）的同步。

comparator0 和 comparator1 的输出状态分别被存储于 CTBMx_COMP_STAT[0] 和 CTBMx_COMP_STAT[16] 中。

表 23-9 总结了用于配置 CTBm 模块中的比较器模式的各个位。

表 23-9. 比较器模式和配置寄存器的设置

寄存器 [位_位置]	位_名称	描述
CTBMx_OA_RESy_CTRL[4]	OAY_COMP_EN	Opampy 比较器使能位 0: CTBMx 中 opampy 内的比较器模式被禁用 1: CTBMx 中 opampy 内的比较器模式被使能
CTBMx_OA_RESy_CTRL[5]	OAY_HYST_EN	Opampy 比较器迟滞使能位 0: CTBMx 中 opampy 内的迟滞被禁用 1: CTBMx 中 opampy 内的迟滞被使能
CTBMx_OA_RESy_CTRL[6]	OAY_BY-PASS_DSI_SYNC	Opampy 旁路比较器输出与 DSI（触发器）输出的同步 0: 同步（电平或脉冲） 1: 旁路
CTBMx_OA_RESy_CTRL[7]	OAY_DSI_LEVEL	Opampy 比较器 DSI（触发器）输出同步电平 0: 脉冲 1: 电平

23.3.4.5 比较器中断

将比较器输出连接至边沿检测器模块，通过该模块可以检测生成中断的边沿（禁用 / 上升沿 / 下降沿 / 双边沿）。通过使用 CTBMx_OA_RESy_CTRL[9:8] 位，可以配置该中断。

每个比较器均有独立的 IRQ。CTBMx_INTR [0] 用于 comparator0 IRQ，CTBMx_INTR [1] 用于 comparator1 IRQ。虽然各个 CTBM 中的每个比较器具有不同的 IRQ 位，但它们都共享了被映射到 CPU NVIC 中的单个 CTBM ISR。有关详细信息，请参考第 51 页上的中断章节中介绍的内容。通过轮询 CTBMx_INTR 位，您可以检查到底是 CTBM 中的哪个比较器触发了该 ISR。

每个中断具有一个 CTBMx_INTR_MASK 寄存器中的中断屏蔽位。通过将中断屏蔽位设置为低电平，可以忽略相应的中断源。如果 CTBMx_INTR 寄存器中的中断标志和 CTBMx_INTR_MASK 寄存器中的相应中断屏蔽进行逻辑 AND 得到的结果为 ‘1’，将生成对 NVIC 的 CTBm 比较器中断。

向 CTBMx_INTR 位 [1:0] 写入 ‘1’，可以清除相应中断。

为方便固件，还可以在 CTBMx_INTR_MASKED 寄存器中对中断标志和中断屏蔽进行交叉（逻辑 AND）。

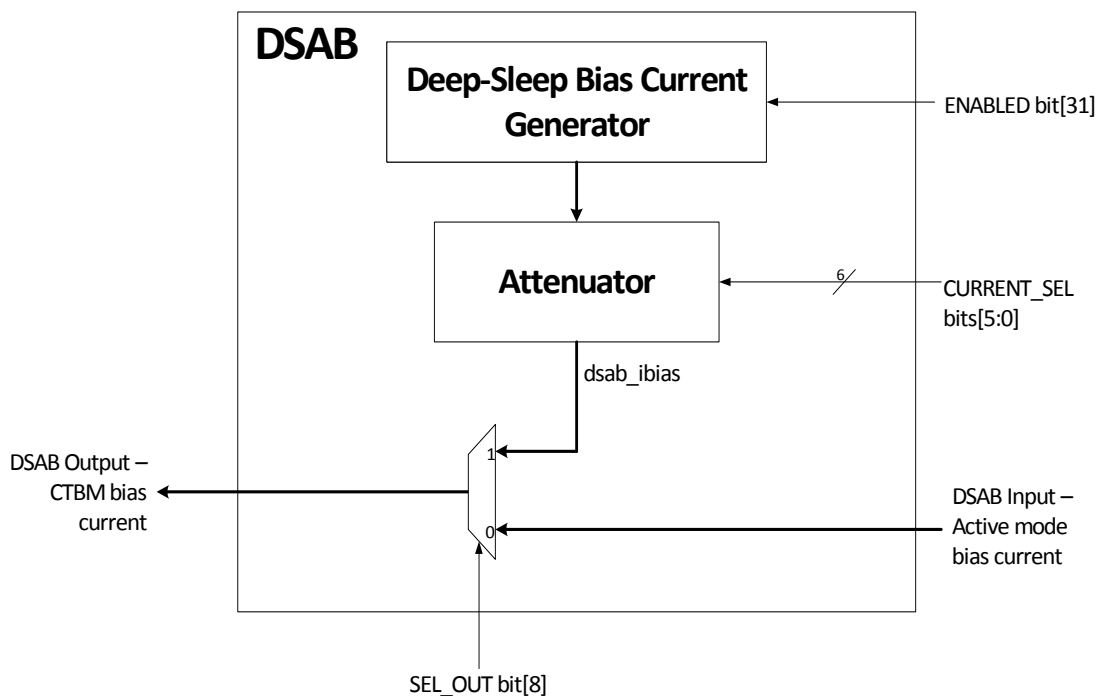
为了便于进行验证和调试，CTBMx_INTR_SET 寄存器为每个中断提供了一个设置位。这样，固件不需要真正的比较器开关事件也可以生成中断。

23.3.4.6 深度睡眠模式操作

在深度睡眠模式下，提供偏置电流、参考电压和 IMO 时钟的模块被关闭。因此，CTBm 功能（它的操作依赖于偏置电流和 IMO 时钟）也随之无效。更多有关各种功耗模式和它们可用模块的信息，请参考第 97 页上的功耗模式章节。为了在深度睡眠模式下支持 CTBm 功能，需要通过一个被称为深度睡眠放大器偏置 (DSAB) 的特殊模块来生成交流偏置电流。这样，CTBm 中的运算放大器能够在深度睡眠模式下工作。

图 23-2 显示的是 DSAB 模块的架构。活动模式偏置电流作为该模块的输入电流。它会输出偏置电流，并提供给运算放大器偏置电路。在活动模式下，DSAB 模块作为一个直通模块运行，并将偏置电流从输入路由至输出。在深度睡眠模式下，如果被使能，DSAB 将生成交流偏置电流，并且输出被衰减为一个用户指定的值，然后为该输出上的 CTBm 模块提供偏置电流。如果禁用了 DSAB 模块，输出会始终连接至输入偏置电流，而且在深度睡眠模式下不会生成交流偏置电流。DSAB 模块被禁用时，运算放大器不会在深度睡眠模式下工作。PASS_DSAB_DSAB_CTRL 寄存器中的 ENABLED 位 [31] 用于使能 / 禁用该模块；而 CURRENT_SEL 位 [5:0] 用于选择输出偏置电流的值。该值被选为 $CURRENT_SEL \times 0.075 \mu A (\pm 5\%)$ 。可以通过 SEL_OUT 位 [8] 选择可路由至 CTBm 偏置的两个偏置电流的其中一个。表 23-10 总结了 PASS_DSAB_DSAB_CTRL 寄存器的位配置设置情况。

图 23-2. 深度睡眠模式下的放大器偏置框图



该特性对要求基于运算放大器电路的设计非常有用，因为可以在低功耗模式下（如深度睡眠模式）保持活动状态，从而节省电源。例如，对于要求运算放大器始终处于使能状态的电池供电系统（如心率监视器），如果芯片其余部分可以进入深度睡眠模式并且只在需要时才会唤醒，则可以节省大量的功耗。请注意，DSAB 模块提供的偏置电流不能满足活动模式偏置电流的准确度和稳定性。另外，DSAB 模块不会生成替代时钟。因此，开关以及与运算放大器相关的电荷泵都不可用。因此，运算放大器的最高输入共模电压被限于 $V_{DDA} - 1.3\text{ V}$ 。另外，由于开关泵不可用（在电压低于 3.3 V 条件下运行时需要使用它们来实现模拟开关），当电源电压下降至低于 3.3 V 水平时，模拟开关的导通电阻会超过正常规范。信号速度低

时，模拟开关需要使用更高的导通电阻值。所以，模拟开关的阻力变得过高前， V_{DDA} 可以降低至 $\sim 2.8\text{ V}$ 。这是电源电压的最低值。但是，在深度睡眠模式下使用运算放大器时，建议将 V_{DDA} 设置为 3.3 V 或更高的值。有关深度睡眠模式下的运算放大器规范的详细信息，请参考 器件数据手册。

要想在深度睡眠模式下使能运算放大器，需要设置 CTBMx-CTBM_CTLB_CTRL 寄存器的 DEEPSLEEP_ON 位 [30]。这样便可以在深度睡眠模式下使能 CTBMx 的两个运算放大器。DSAB 模块被使能后，CTBm 才能在深度睡眠模式下运行。

表 23-10. DSAB 和 CTBM 深度睡眠配置寄存器的设置

寄存器 [位_位置]	位_名称	描述
PASS_DSAB_DSAB_CTRL [5:0]	CURRENT_SEL	dsab_ibias 的电流选择： $\text{dsab_ibias} = \text{CURRENT_SEL} \times 0.075\text{ }\mu\text{A}$ ($\pm 5\%$)
PASS_DSAB_DSAB_CTRL [8]	SEL_OUT	CTBm 的偏置电流选择 0: 旁路 DSAB，并使用活动模式中的偏置电流 1: 将 dsab_ibias 作为 CTBm 偏置电流使用
PASS_DSAB_DSAB_CTRL [31]	ENABLED	使能 / 禁用 DSAB 偏置发生器 0: DSAB 模块被禁用，并且 CTBm 偏置电流被连接到活动模式的偏置电流 1: DSAB 模块被使能，并且 CTBm 偏置电流是由 SEL_OUT 信号控制的
CTBMx_CTBM_CTLB_CTRL [30]	DEEPSLEEP_ON	使能 / 禁用深度睡眠模式中的 CTBMx 功能 0: 使能 1: 禁用

23.4 寄存器汇总

表 23-11. 寄存器总结

名称	描述
CTBMx_CTRL	全局 CTBm 模块使能
CTBMx_OA_RES0_CTRL	Opamp0 控制寄存器
CTBMx_OA_RES1_CTRL	Opamp1 控制寄存器
CTBMx_COMP_STAT	比较器状态
CTBMx_INTR	中断请求寄存器
CTBMx_INTR_SET	中断请求设置寄存器
CTBMx_INTR_MASK	中断请求屏蔽
CTBMx_INTR_MASKED	中断请求被屏蔽
CTBMx_OA0_SW	Opamp0 开关控制
CTBMx_OA0_SW_CLEAR	Opamp0 开关控制清除
CTBMx_OA1_SW	Opamp1 开关控制
CTBMx_OA1_SW_CLEAR	Opamp1 开关控制清除
CTBMx_SW_HW_CTRL	CTBm 硬件控制使能
CTBMx_SW_STATUS	CTBm 总线开关控制状态
CTBMx_OA0_OFFSET_TRIM	Opamp0 调整控制
CTBMx_OA0_SLOPE_OFFSET_TRIM	Opamp0 调整控制
CTBMx_OA0_COMP_TRIM	Opamp0 调整控制
CTBMx_OA1_OFFSET_TRIM	Opamp1 调整控制
CTBMx_OA1_SLOPE_OFFSET_TRIM	Opamp1 调整控制
CTBMx_OA1_COMP_TRIM	Opamp1 调整控制
PASS_DSAB_DSAB_CTRL	DSAB 控制寄存器
PASS_DSAB_TRIM	IBIAS 调整寄存器

24. LCD 直接驱动



PSoC[®] 4 液晶显示屏（LCD）驱动器系统是一种可灵活配置的外设，通过它可以使 PSoC 器件直接驱动 STN 及 TN segment LCD。

24.1 特性

PSoC 4 LCD segment 驱动模块具有以下特性：

- 支持多达 56 个 segment 和 8 个 common
- 支持 A 型（标准）和 B 型（低功耗）驱动波形
- 可将任意一个 GPIO 配置为 common 或 segment
- 支持以下 5 种驱动模式：
 - 数字相关
 - 1/2 偏压下的 PWM 驱动模式
 - 1/3 偏压下的 PWM 驱动模式
 - 1/4 偏压下的 PWM 驱动模式
 - 1/5 偏压下的 PWM 驱动模式
- 在数字相关模式下，能够使用 1.8 V 的 V_{DD} 驱动 3 V 的显示屏
- 能够在活动、睡眠及深度睡眠模式下运行
- 支持数字对比度控制

24.2 LCD segment 驱动概述

一个分段的 LCD 显示屏由以下部分构成：悬浮于两组电极之间的液晶材料、多个偏振和反射层。这两组电极分别称为共模信号（COM）或背板和段式电极（SEG）。从电气的角度来看，可将一个 LCD segment 视为电容负载；将 COM/SEG 电极视为 segment 矩阵中的行和列。通过改变对应的 COM/SEG 对的均方根（RMS）电压，可以控制 LCD segment 的不透明度。

本章节中使用了以下各术语 / 电压来介绍 LCD 驱动器：

- **V_{RMSOFF}** ：表示 LCD 驱动器对准备关闭的各 segment 可执行的电压值。
- **V_{RMSON}** ：表示 LCD 驱动器对打算打开的各 segment 可执行的电压值。
- **识别率（D）**：表示 LCD 驱动器可执行的 V_{RMSON} 及 V_{RMSOFF} 的比率。该比率取决于应用于 LCD 屏幕的波形类型。识别率越高，则对比度越高。

液晶材料不能长期在直流电压下工作。因此，应用于屏幕的任何波形必须在各 segment（打开或关闭）上产生 0 V 的直接电流。通常，LCD 驱动器将波形应用于（通过多种电压之间的切换生成的）COM 和 SEG 电极。文档使用下列术语定义了这些波形：

- **占空比**：当一个驱动器驱动了 ‘M’ 个 COM 电极时，便可以确定它运行时的占空比为 1/M。在 1/M 的时间内，各个 COM 电极被有效驱动。
- **偏压**：当驱动器的波形使用 $(1/B) \times V_{DRV}$ 的电压阶跃时，则可以确定它的偏压为 1/B。VDRV 是系统中最高的驱动电压（等于 PSoC 4 中的 V_{DD} 电压）。在 PWM 驱动模式下，PSoC 4 支持 1/2、1/3、1/4 以及 1/5 的偏压。
- **帧**：一帧表示驱动所有 segment 所需的时间。执行一个帧期间，驱动器将经过 common 顺序循环。测量整个帧时，所有 segment 将接收 0 V 直流（非零的 RMS 电压）。

在所有驱动模式下，PSoC 4 均支持两种不同的驱动波形类型，包括：

- **A 类型波**：在这种波形中，驱动器将一帧分成 M 个子帧。‘ M ’ 为 COM 电极的数量。在执行一个帧期间，仅对各个 COM 标记一次。例如，COM[i] 会在子帧 i 中被标记。
- **B 类型波**：驱动器将一帧分成 $2M$ 个子帧。这两个子帧是互为相反的。在执行一个帧期间，对各个 COM 标记两次。例如，COM[i] 会在子帧 i 及 $M+i$ 中被标记。由于 B 类型波的每个帧中都包含了较少的转换，所以它的节能特性更显著。

24.2.1 驱动模式

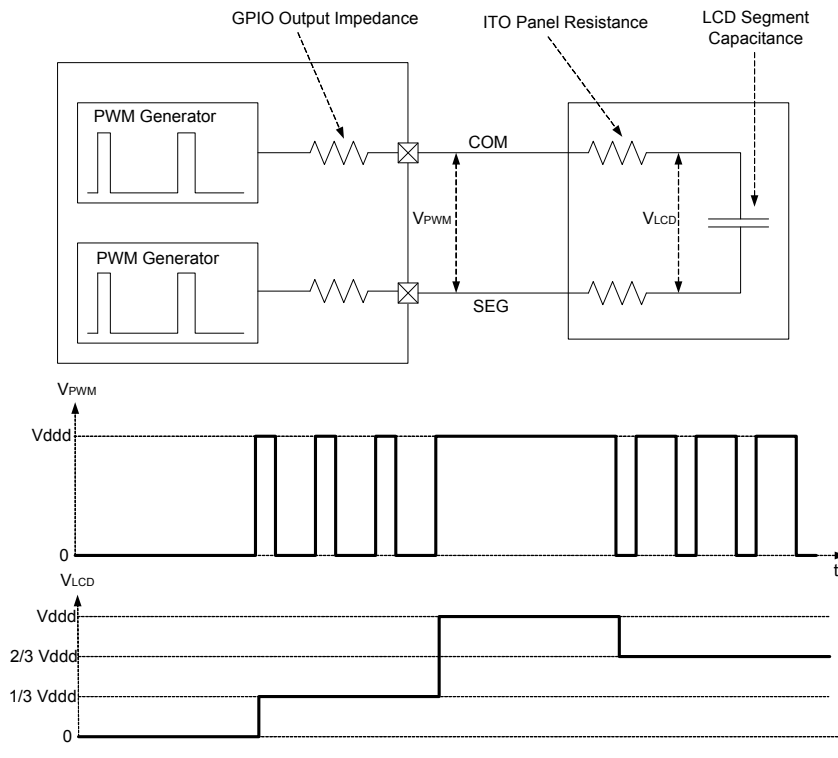
PSoC 4 支持以下驱动模式。

- 1/2 偏压下的 PWM 驱动
- 1/3 偏压下的 PWM 驱动
- 当时钟输入信号为高频时，PWM 会在 1/4 偏压模式下驱动
- 当时钟输入信号为高频时，PWM 会在 1/5 偏压模式下驱动
- 数字相关

24.2.1.1 PWM 驱动

在 PWM 驱动模式下，通过使用 PWM 输出信号与 LCD 显示屏的固有电阻和电容来生成多个电压驱动信号，如图 24-1 所述。

图 24-1. 1/3 偏压下的 PWM 驱动



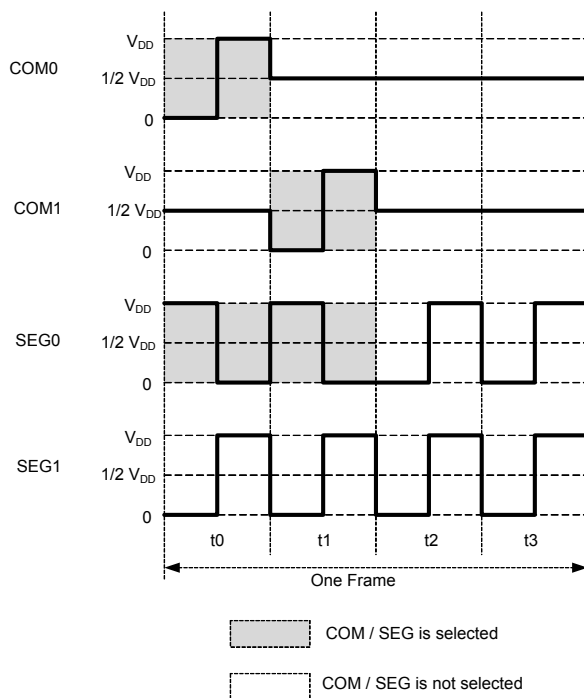
驱动电气的输出波形是一个 PWM 波形。通过使用铟锡氧化物（ITO）屏幕电阻和 segment 电容来过滤 PWM，LCD segment 的是一个模拟电压，如图 24-1 所示。该图显示了在 1/3 偏压下波形的生成过程（四个 common，电压阶跃为 $V_{DD}/3$ ）。

PWM 是由 ILO（频率为 32 kHz、低速操作）或 IMO（高速操作）派生的。通常，所生成的模拟电压在超低的频率下（~ 50 Hz）运行，以驱动 segment LCD。

图 24-2 及图 24-3 分别说明了在 1/2 偏压及 1/4 占空比下 COM 和 SEG 电极的 A 类型和 B 类型波。其中只绘制了 COM0/COM1 及 SEG0/SEG1，用于演示。同样，图 24-4 及图 24-5 分别说明了在 1/3 偏压及 1/4 占空比下的 COM 和 SEG 电极的 A 类型和 B 类型波。

图 24-2. PWM1/2 的 A 类型波示例

One 'Frame' of Type A Waveform
(addresses all segments once)



Resulting voltage across segments
($V_{DC} = 0$)

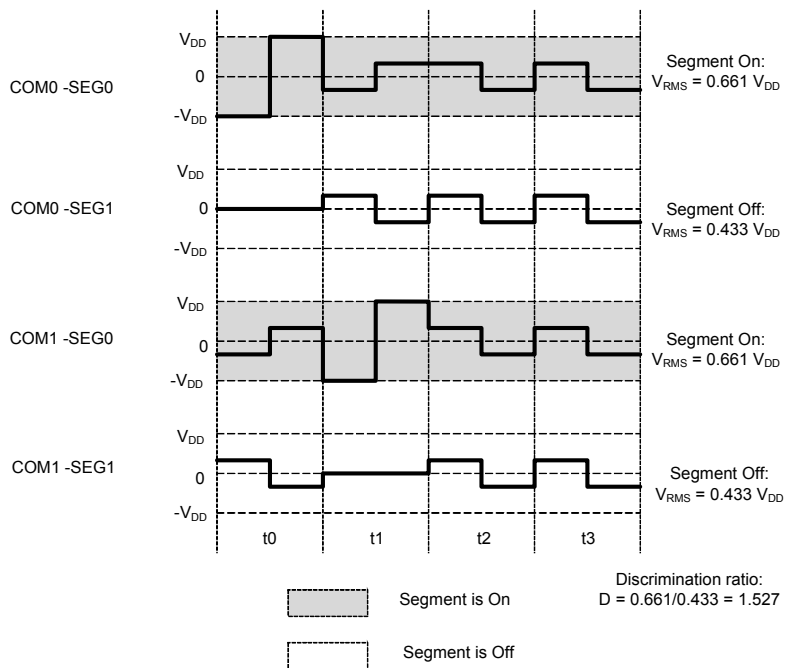
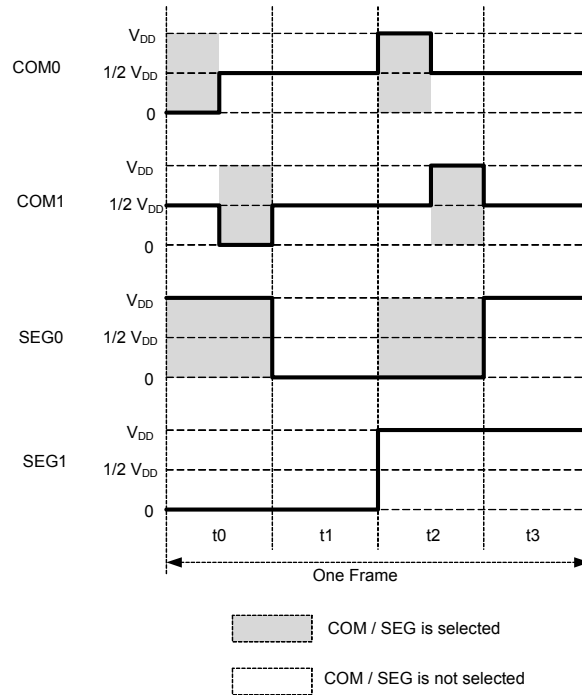


图 24-3. PWM1/2 的 B 类型波示例

One 'Frame' of Type B Waveform
(addresses all segments twice)



Resulting voltage across segments
($V_{DC} = 0$)

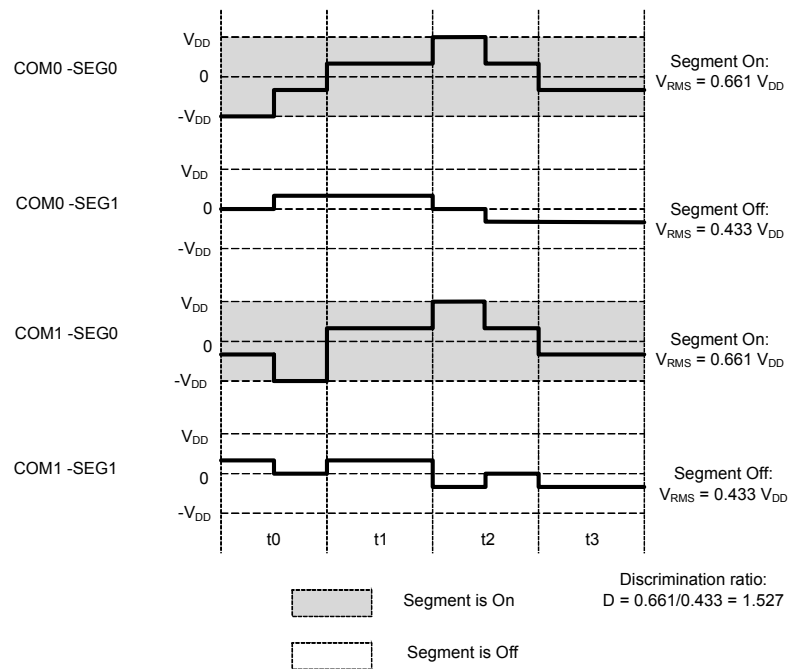


图 24-4. PWM1/3 的 A 类型波示例

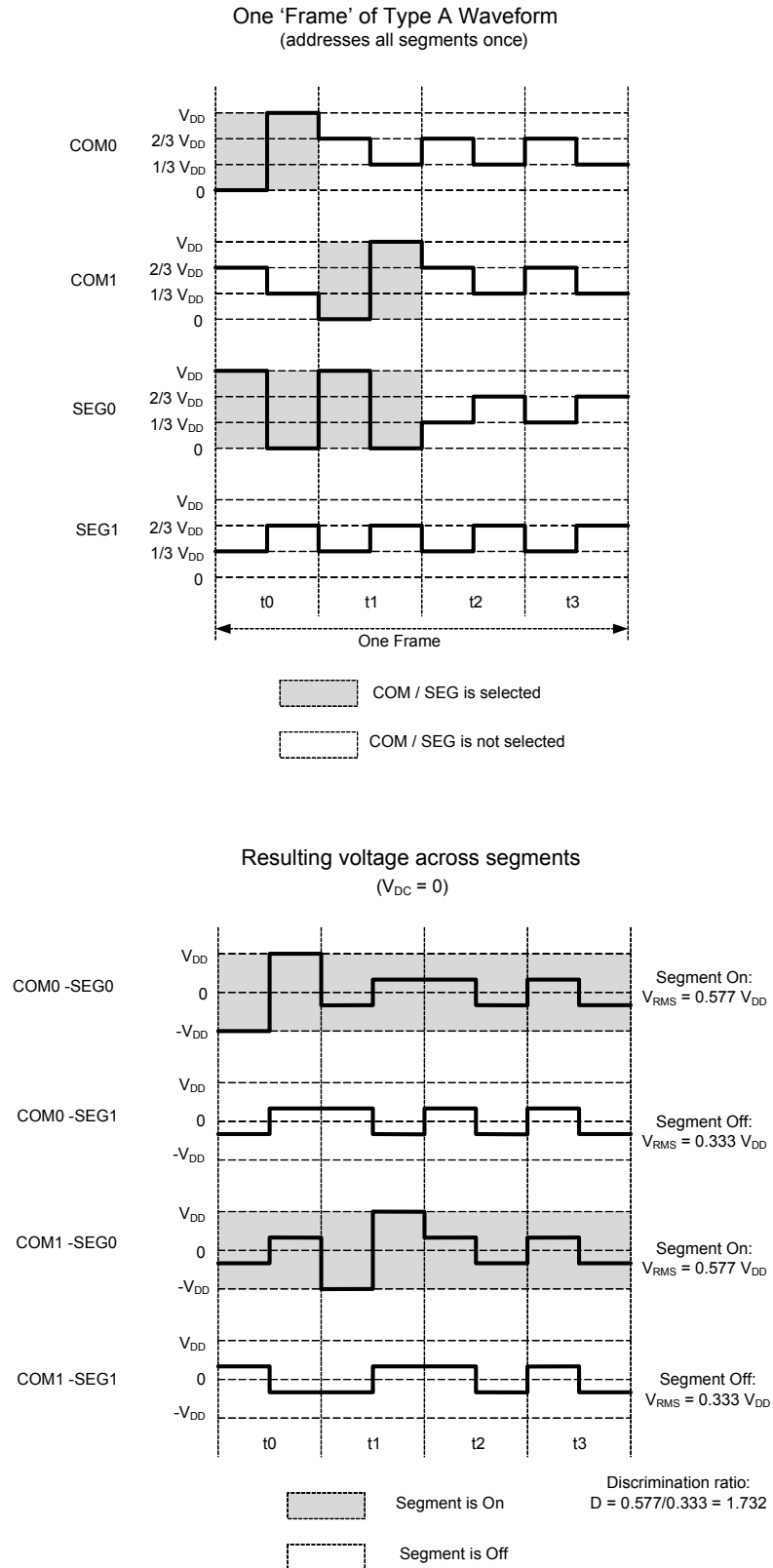
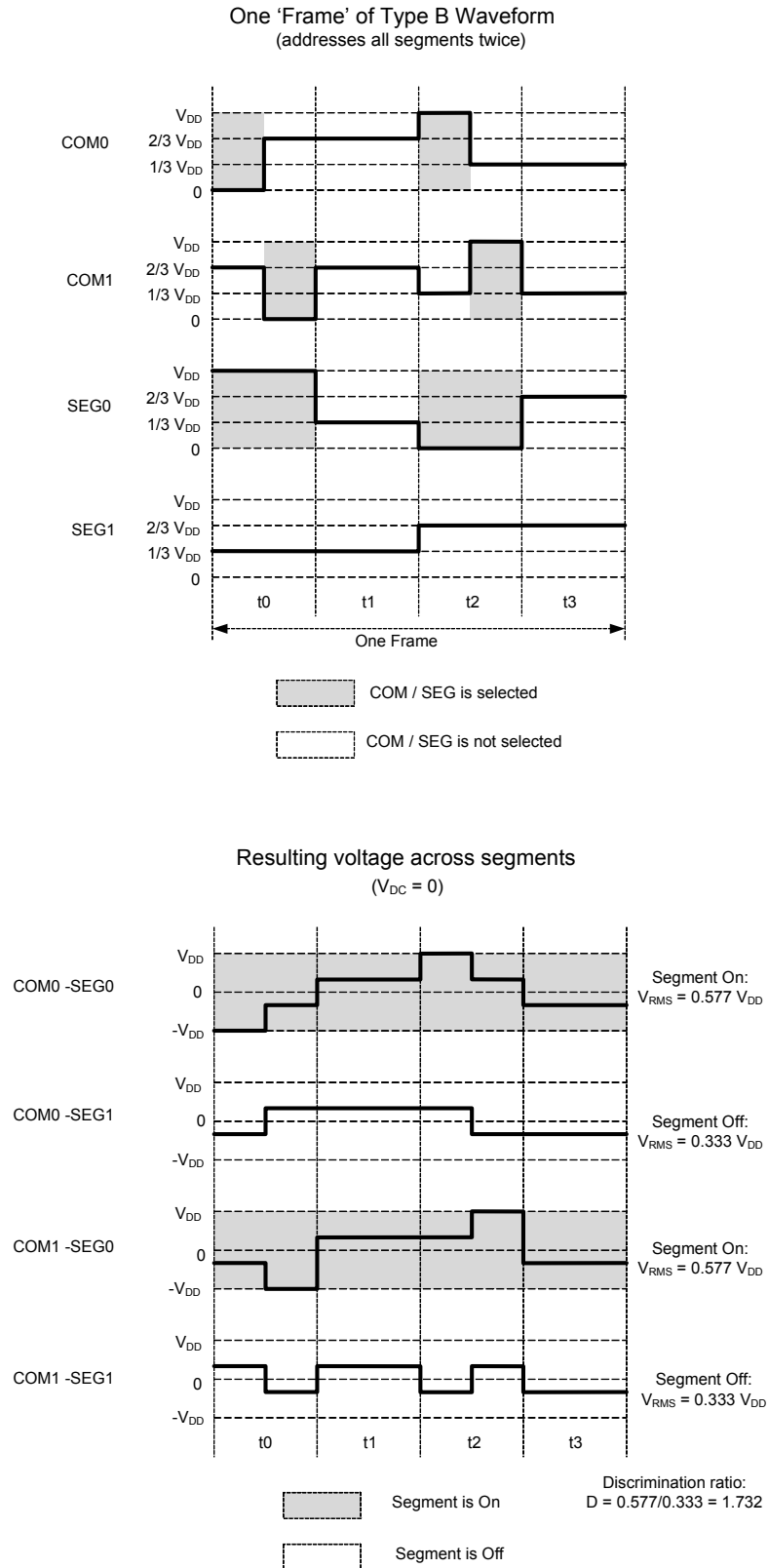


图 24-5. PWM1/3 的 B 类型波示例



通过下列公式，可容易地计算出打开（ON）或关闭（OFF）段式的有效 RMS 电压：

$$V_{\text{RMS(OFF)}} = \sqrt{\frac{2(B-2)^2 + 2(M-1)}{2M}} \times \left(\frac{V_{\text{DRV}}}{B}\right) \quad \text{公式 24-1}$$

$$V_{\text{RMS(ON)}} = \sqrt{\frac{2B^2 + 2(M-1)}{2M}} \times \left(\frac{V_{\text{DRV}}}{B}\right) \quad \text{公式 24-2}$$

其中，B 是偏压，M 是占空比（也是 COM 的数值）。

例如，如果 COM 的数量是 4，那么在 1/2 及 1/3 偏压下的识别率分别为 1.528 及 1.732。如果 COM 的数量是 2 和 3，则 1/3 偏压所提供的识别率也较好。因此，与 1/2 偏压相比，1/3 偏压提供了更高的对比度，并建议将其使用于大多数应用中。1/4 和 1/5 偏压仅适用于 LCD 的高速操作。特别是使用率高的 COM 设计（多于 4 个 COM）时，它们会提供更好的识别率。

当使用 LCD 的低速操作时，PWM 信号会由 32 kHz ILO 派生得到。为了使用 32 kHz 脉冲宽度调制驱动具有可接受波纹及上升 / 下降时间的低电容显示屏，需要使用 100 k-1 MΩ 的附加外部串联电阻。频率大于 ~1 MHz 的 PWM 无需外部电阻。理想的 PWM 频率取决于显示屏的电容及 ITO 路由线的内部 ITO 电阻。

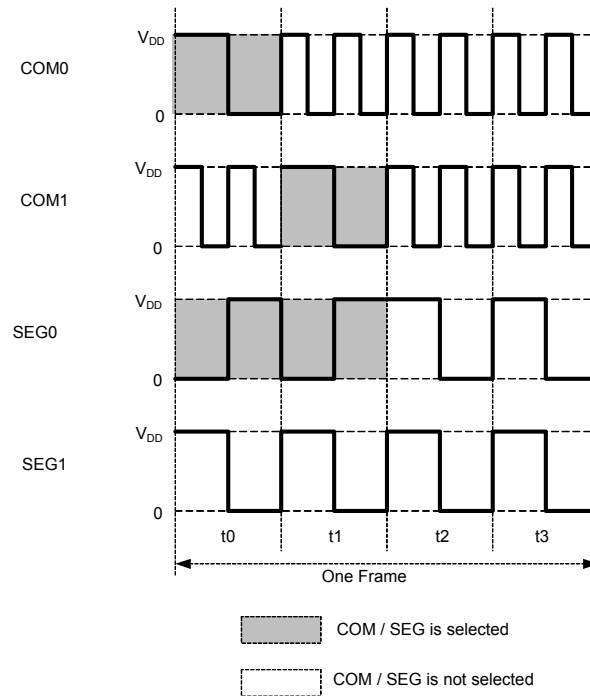
1/2 偏压模式的优势是，脉冲宽度调制只需要用于 COM 信号，而 SEG 信号则仅使用逻辑电平，如图 24-2 和图 24-3 所示。

24.2.1.2 数字相关

数字相关方法并非在轨与轨之间生成偏压，而是利用 LCD 显示屏的特性：LCD segment 的对比度由各 segment 的 RMS（均方根）电压决定。使用这种方法，任意给定 COM 和 SEG 信号对之间的关联系数确定了对应的 LCD segment 是打开还是关闭。因此，通过将 COM 信号在其非活动子帧间隔中的基准驱动频率翻倍，COM 和 SEG 驱动信号的相位关系可发生变化，以打开和关闭各 segment。这种情况与改变信号的直流电平的 PWM 驱动方法不同。图 24-8 及图 24-9 显示的是描述操作原则的波形示例。

图 24-6. 数字相关 A 类型波

One 'Frame' of Type A Waveform
(addresses all segments once)



Resulting voltage across segments
($V_{DC} = 0$)

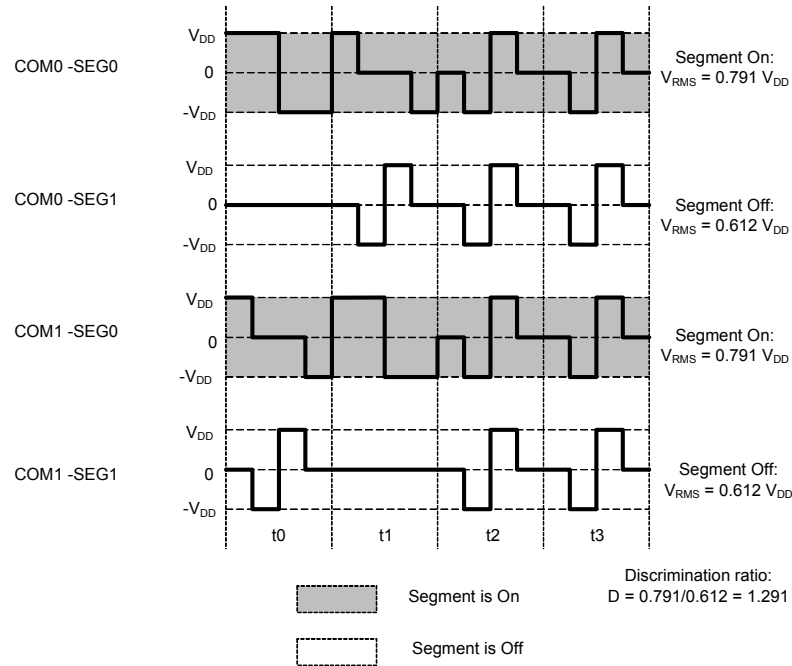
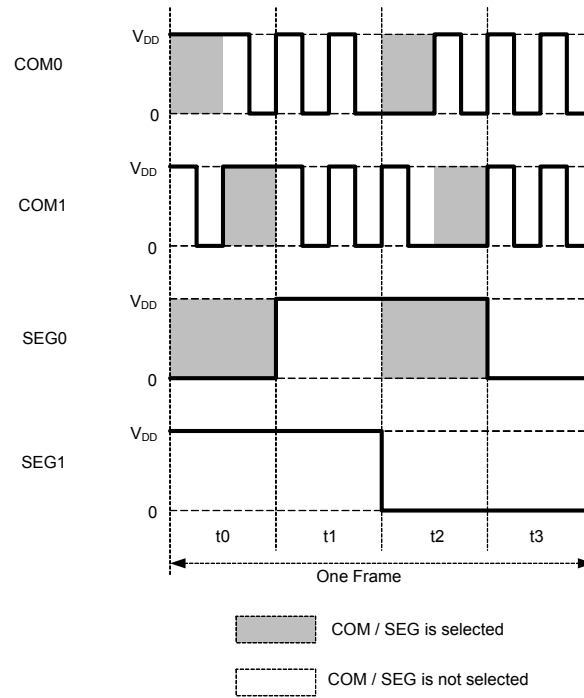
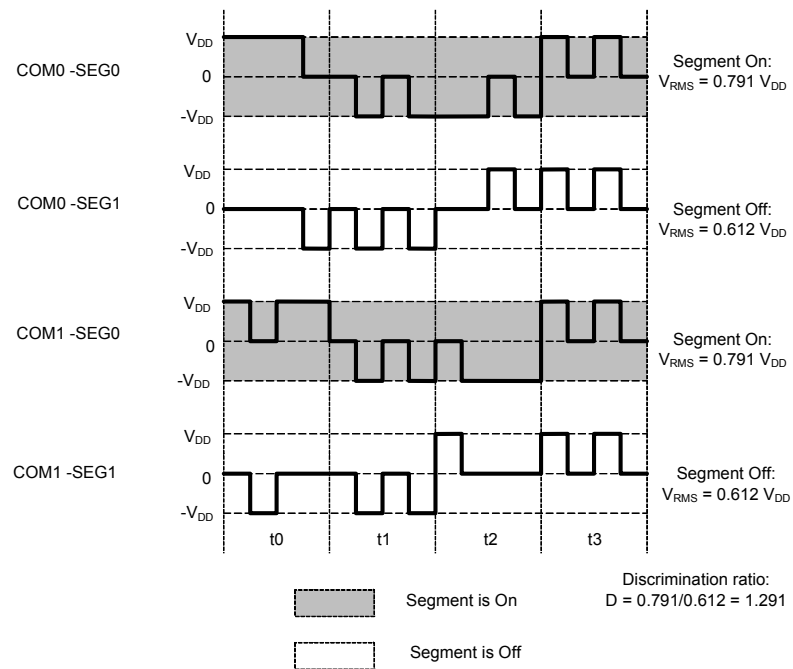


图 24-7. 数字相关 B 类型波

One 'Frame' of Type B Waveform
(addresses all segments twice)



Resulting voltage across segments
($V_{DC} = 0$)



可使用以下公式计算出用于打开或关闭 **segment** 的 RMS 电压：

$$V_{\text{RMS(ON)}} = \sqrt{\frac{(M-1)}{2M}} \times (V_{\text{DD}})$$

$$V_{\text{RMS(OFF)}} = \sqrt{\frac{2 + (M-1)}{2M}} \times (V_{\text{DD}})$$

其中，**B** 是偏压，**M** 是占空比（也是 **COM** 的数值）。这样可使四个 **COM** 的识别率（**D**）等于 1.291。

数字相关模式还能够使用 1.8 V 的 V_{DD} 来驱动 3 V 的显示屏。

24.2.2 建议使用的驱动模式

与数字相关模式相比，PWM 驱动模式具有较高的识别率，如 24.2.1.1 PWM 驱动及 24.2.1.2 数字相关所示。因此，数字相关方法的对比度小于 PWM 方法的对比度，但由于数字相关方法的波形切换频率较低，所以它的功耗也较低。

数字相关模式对 TN 显示屏创建了较低但可接受的对比度，但在对比度较高的 STN 显示屏上，对于对比度和视角，将没有明显的区别。

因为每种模式都各有优劣，所以对使用情况提出如下建议：

表 24-1. 推荐使用的驱动模式

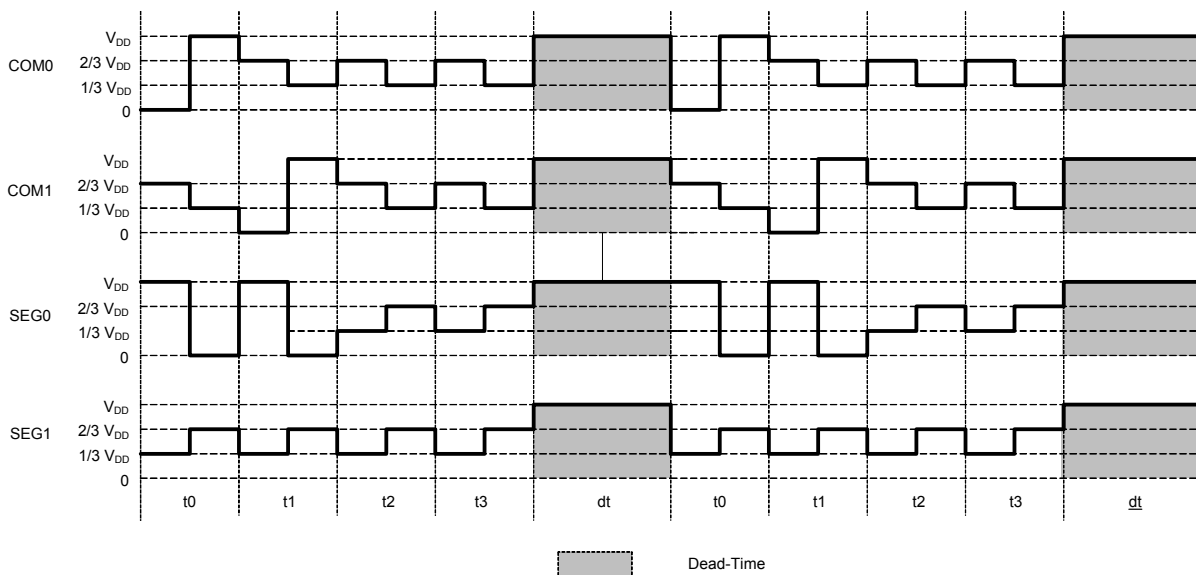
显示屏类型	深度睡眠模式	睡眠 / 活动模式	注意
四个 COM 的 TN 显示屏	数字相关	PWM 1/3 偏压	LCD 进入深度睡眠状态或唤醒状态前，固件必须对各种 LCD 驱动模式进行切换。
四个 COM 的 TN 显示屏	数字相关		STN 液晶屏使用 PWM 驱动时不会带来对比度的优势。
8 或 16 个 COM 的 STN 显示屏	不支持	PWM 模式下的 1/4 偏压和 1/5 偏压	仅在高速 LCD 模式下得到支持。低速时钟会使 PWM 不能在高复用率的条件下运行。

24.2.3 数字对比度控制

在所有驱动模式下，均可使用数字对比度控制来更改各 **segment** 的对比度。该方法通过缩短各 **segment** 的驱动时间来降低对比度。通过在各帧后插入一个死区时间间隔来实现这一操作。在死区时间期间，所有 **COM** 和 **SEG** 信号都被驱动为逻辑“1”。在良好的分辨率条件下，可以控制死区时间。图 24-8 说明在 1/3 偏压和 1/4 占空比下实现的死区时间对比度控制方法。

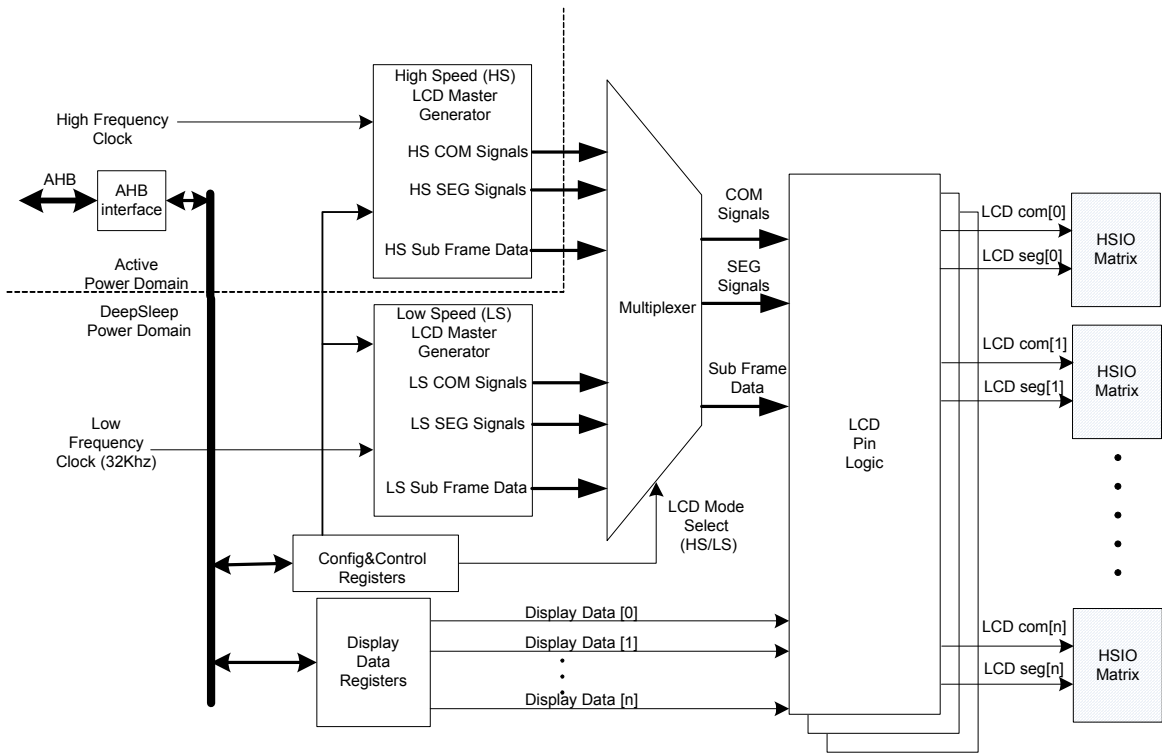
图 24-8. 死区时间对比度控制

Two Frames of of Type A Waveform with Dead-time
(Example for 1/4th Duty and 1/3rd bias)



24.3 框图

图 24-9. LCD 直接驱动系统框图



24.3.1 工作原理

LCD 控制器模块包括两个发生器；其中一个使用高速时钟源 HFCLK，另一个使用由 ILO 生成的低速时钟源（32 kHz）。这两个发生器分别被称为高速 LCD 主发生器和低速 LCD 主发生器。它们都支持 PWM 驱动模式及数字相关驱动模式。若 PWM 驱动模式下使用了低速发生器，则需要使用外部电阻，如第 264 页上的 PWM 驱动所示。

根据固件的配置，复用器将选择一个发生器输出，用以驱动 LCD。LCD 引脚逻辑模块将 COM 及 SEG 输出从发生器路由到相应的 I/O 矩阵。任何 GPIO 均可作为 COM 或 SEG 使用。该 COM 或 SEG 的可配置引脚分配会在 GPIO 和 I/O 矩阵上进行，请参考第 72 页上的高速输入 / 输出矩阵。这两个发生器共用了相同的配置寄存器。这些存储器映射 I/O 寄存器通过使用 AHB 接口连接到系统总线（AHB）。

LCD 控制器可在三种器件功耗模式下运行：活动、睡眠及深度睡眠模式。在活动 and 睡眠模式下能够进行高速操作。在活动、睡眠及深度睡眠模式下能够进行低速操作。在休眠及停止模式下，LCD 控制器被断电。

24.3.2 高速和低速主发生器

高速和低速主发生器是一样的。唯一的差别是高速主发生器具有较大频率的分频器，用于生成帧周期及子帧周期。这是因为高速模块（HFCLK）的时钟从 IMO 派生的，而 IMO 的频率通常是供给低速模块的 ILO（32 kHz）频率的 30 到 100 倍。高速发生器运行于活动电压域，而低速发生器则运行于深度睡眠电压域。另外，为了控制高速及低速模块提供单一的一组配置寄存器。每个主发生器都具有以下特性及特点：

- 用于配置 A 类型或 B 类型驱动波形的模块的寄存器位（LCD_CONTROL 寄存器中的 LCD_MODE 位）。
- 用于选择 COM 数量的寄存器位（LCD_CONTROL 寄存器中的 COM_NUM 字段）。可用值分别是 2、3 和 4。
- 通过使能操作模式配置位，可以选择以下模式：
 - 数字相关
 - PWM 1/2 偏压
 - PWM 1/3 偏压
 - PWM 1/4 偏压（在低速发生器中不受支持）
 - PWM 1/5 偏压（在低速发生器中不受支持）
 - 关闭 / 禁用。通常，两个发生器中的一个被配置为 Off（关闭）

LCD_CONTROL 位中的 OP_MODE 和 BIAS 字段用于选择驱动模式。

- 用于生成子帧时序的计数器。LCD_DIVIDER 寄存器中的 SUBFR_DIV 字段可确定各子帧的持续时间。如果写入该计数器的分频值为 C，那么子帧持续时间为 $4 \times (C+1)$ 。低速发生器具有一个 8 位的计数器。该计数器从 32 kHz ILO 时钟生成一个最大为 8 ms 的半子帧周期。高速发生器具有一个 16 位的计数器。
- 用于生成死区时间周期的计数器。这些计数器与子帧周期计数器具有相同的位数量，并且还使用了相同的时钟。LCD_DIVIDER 寄存器中的 DEAD_DIV 字段控制死区时间周期。

24.3.3 复用器及 LCD 引脚逻辑

复用器选择高速或低速主发生器模块的输出信号，并将该信号传输给 LCD 引脚逻辑。该选择由配置及控制寄存器控制。LCD 引脚逻辑通过使用复用器的子帧信号来选择显示数据。该引脚逻辑将被复制，以供给各个 LCD 引脚。

24.3.4 显示数据寄存器

每个 LCD segment 引脚是一个 LCD 端口的一部分，各自都有独立的 LCD_DATA_nx 显示数据寄存器。这样的 LCD 端口在器件中共有 8 个。请注意，这些端口不是真正的引脚端口，而是 LCD 硬件中有效的端口 / 连接，用于将各 segment 映射到各 common。所配置的每一个 LCD segment 都被作为这些 LCD 端口中的一个引脚使用。LCD_DATA_nx 寄存器是 32 位宽寄存器，用于存储设计中所有被使能的 SEG-COM 组合的 ON/OFF 数据。LCD_DATA0x 用于保持 COM0 到 COM3 的 SEG-COM 数据，而 LCD_DATA1x 则用于保持 COM4 到 COM7 的 SEG-COM 数据。每个 LCD_DATA0x 寄存器的 [4i+3:4i] 位（‘i’ 表示引脚编号）显示了 Port[x] 及 COM[3,2,1,0] 组合中的 Pin[i] ON/OFF 数据，如表 24-2 所示。应根据每一帧显示的数据对 LCD_DATA_nx 寄存器进行编程。显示数据寄存器是存储器映射 I/O（MMIO），并且可以通过 AHB 从设备接口访问它。

表 24-2. LCD_DATA0x 寄存器的 SEG-COM 映射情况（每一 SEG 是 LCD 端口的一个引脚）

BITS[31:28] = PIN_7[3:0]				BITS[27:24] = PIN_6[3:0]			
PIN_7-COM3	PIN_7-COM2	PIN_7-COM1	PIN_7-COM0	PIN_6-COM3	PIN_6-COM2	PIN_6-COM1	PIN_6-COM0
BITS[23:20] = PIN_5[3:0]				BITS[19:16] = PIN_4[3:0]			
PIN_5-COM3	PIN_5-COM2	PIN_5-COM1	PIN_5-COM0	PIN_4-COM3	PIN_4-COM2	PIN_4-COM1	PIN_4-COM0
BITS[15:12] = PIN_3[3:0]				BITS[11:8] = PIN_2[3:0]			
PIN_3-COM3	PIN_3-COM2	PIN_3-COM1	PIN_3-COM0	PIN_2-COM3	PIN_2-COM2	PIN_2-COM1	PIN_2-COM0
BITS[7:3] = PIN_1[3:0]				BITS[3:0] = PIN_0[3:0]			
PIN_1-COM3	PIN_1-COM2	PIN_1-COM1	PIN_1-COM0	PIN_0-COM3	PIN_0-COM2	PIN_0-COM1	PIN_0-COM0

24.4 寄存器列表

表 24-3. LCD 直接驱动寄存器列表

寄存器名称	说明
LCD_ID	该寄存器包含了 LCD 控制器 ID 及版本号信息
LCD_DIVIDER	该寄存器控制子帧及死区时间周期
LCD_CONTROL	该寄存器用于配置高速和低速发生器
LCD_DATA0x	COM0 到 COM3 的 LCD 端口引脚数据寄存器；x 表示端口编号，共有 8 个可用端口
LCD_DATA1x	COM4 到 COM7 的 LCD 端口引脚数据寄存器；x 表示端口编号，共有 8 个可用端口

25. 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优的信噪比 (SNR)。CSD 是硬件技术和固件技术的结合。本章节介绍的是在 PSoC 4 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、可用的 CapSense 设计工具、容易使用的 PSoC Creator[™] 组件、使用调谐器 GUI 来调校性能以及 PCB 布局设计注意事项等详细信息。

25.1 特性

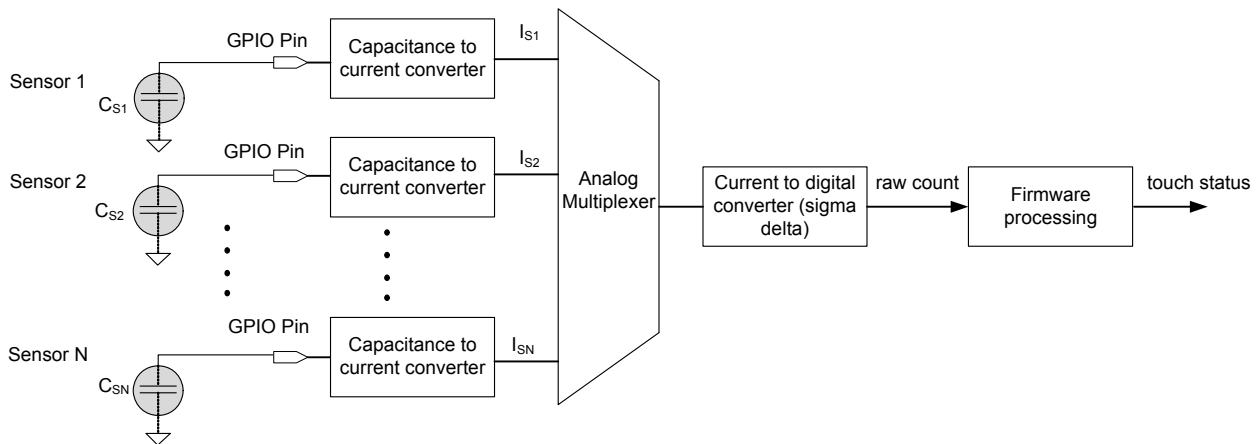
PSoC 4 CapSense 具有以下特性：

- 强大的感应技术
- CSD 操作提供了最佳的 SNR
- 在不同覆盖材料和厚度的条件下仍能提供高性能感应
- SmartSense[™] 自动调校技术
- 支持多达 90 个传感器
- 具有大范围的接近感应
- 所有 GPIO 都支持基于屏蔽信号的防水功能
- 低功耗
- 同时使用两个 IDAC，可提高扫描速度和 SNR
- 任何 GPIO 引脚都可用于感应或屏蔽
- 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
- 通过两个单独的 CSD 模块，可以将电容感应与模拟电路的供应 IDAC 结合（例如：通电链路和传感器的结合）。

25.2 框图

图 25-1 显示的是 CSD 系统框图。

图 25-1. CapSense 模块 框图



25.3 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 - 数字转换器的输出计数（被称为原始计数值）是一个与传感器电容大小成比例的数字值。

图 25-2 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，原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，固件中的逻辑可判定传感器是否处于激活状态（即存在手指触摸）。

图 25-2. 原始计数值与时间

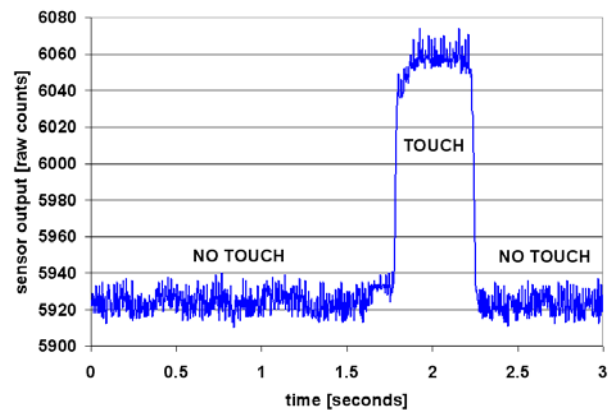


图 25-3 显示的是 PSoC 4 CapSense 硬件的框图。

图 25-3. PSoC 4 CapSense CSD 感应

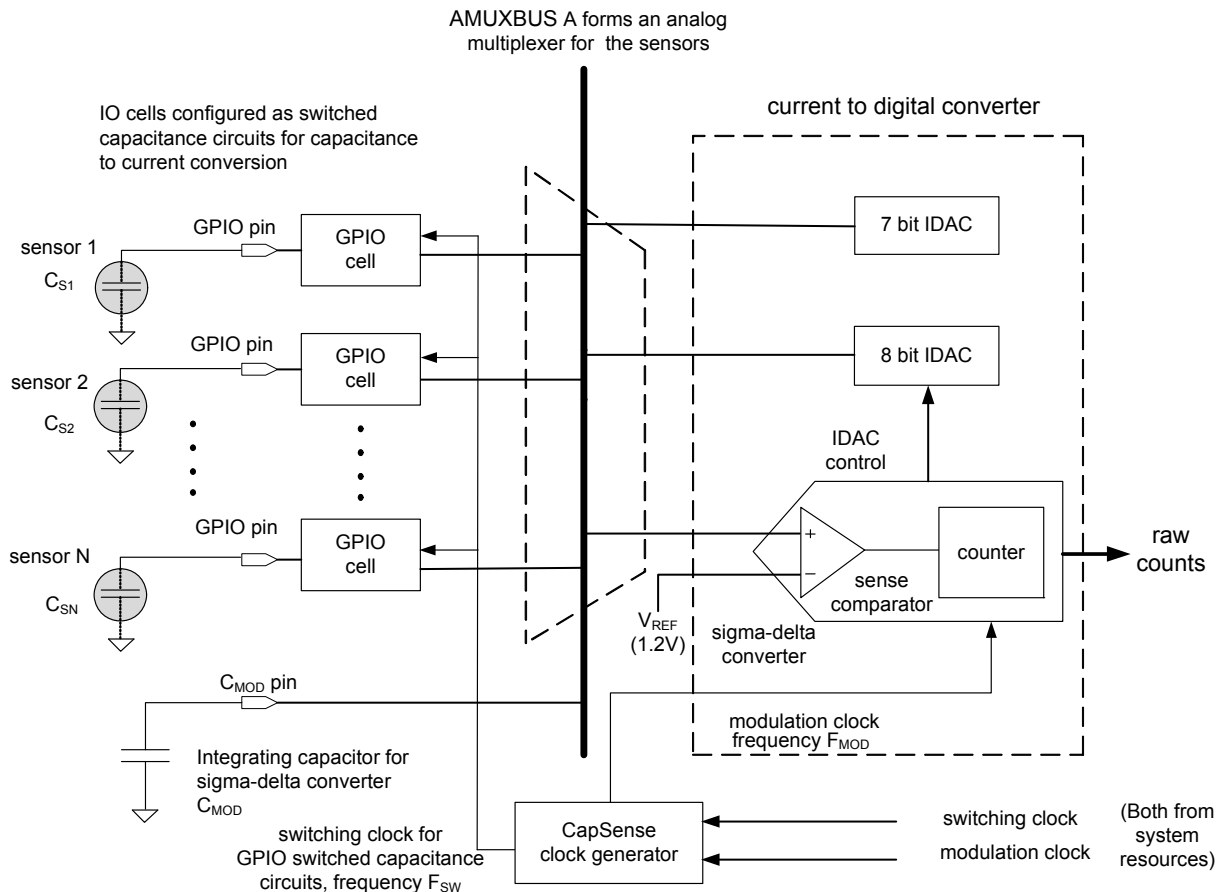
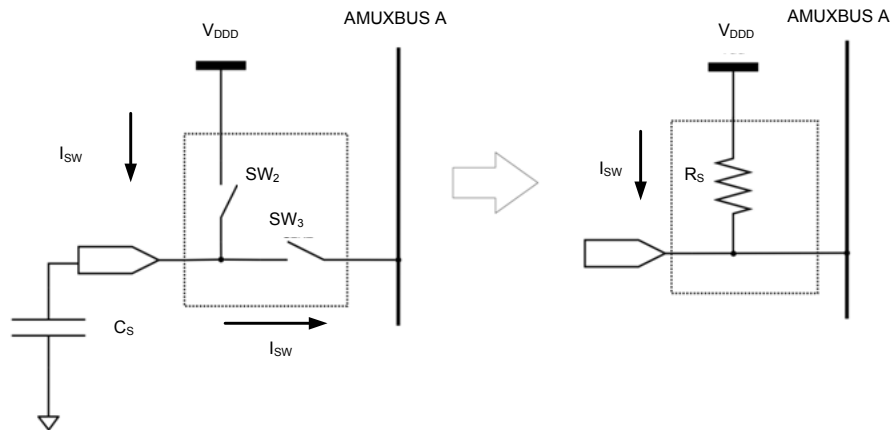


图 25-4. PSoC 4 中的 GPIO 单元

The diagram shows a 4051 IC (multiplexer) with its VDD pin connected to VDD and its GND pin connected to ground. The select pin is connected to a GPIO Pin. The two data inputs are labeled SW1 and SW2. The output is labeled SW3. The output SW3 is connected to the A input of the AMUXBUS. The output SW4 is connected to the B input of the AMUXBUS.

PSoC 4200L 系列: PSoC 4 架构技术参考手册, 文档编号: 002-11591 版本**

图 25-5. 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 25-3）控制着开关 SW_2 和 SW_3 。连续切换 SW_2 和 SW_3 会构成一个等效的电阻 R_S ，如图 25-5 所示。等效的 R_S 电阻值为：

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

公式 25-1

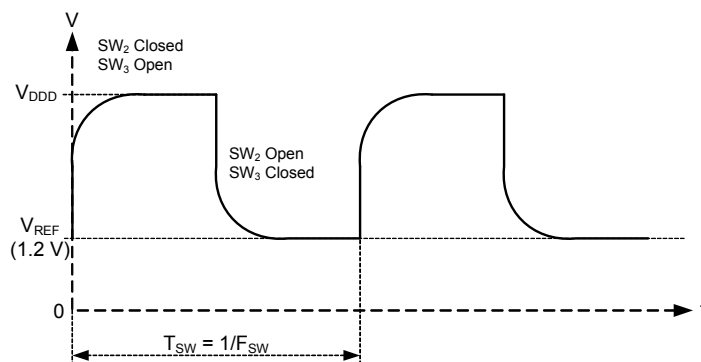
其中：

C_S = 传感器电容

F_{SW} = 开关时钟频率

Sigma Delta 转换器使 AMUXBUS A 的电压保持为一个常量 V_{REF} （在第 279 页上的 Sigma Delta 转换器中详细介绍了该过程）。图 25-6 显示的是传感器电容的电压波形。

图 25-6. 传感器电容上的电压



通过公式 23-3，可以计算得出提供给 AMUXBUS A 的平均电流。

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

公式 25-2

图 25-7 显示了从 AMUXBUS A 输出的灌电流的开关电容配置。图 25-8 则显示的是 C_S 上的电压波形结果。

图 25-7. AMUXBUS A 输出的灌电流

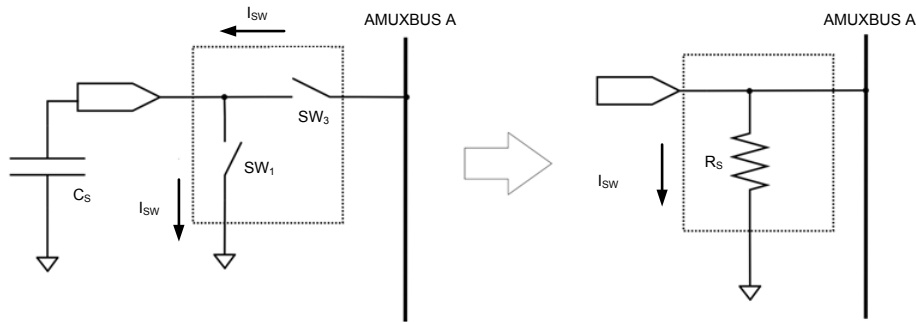
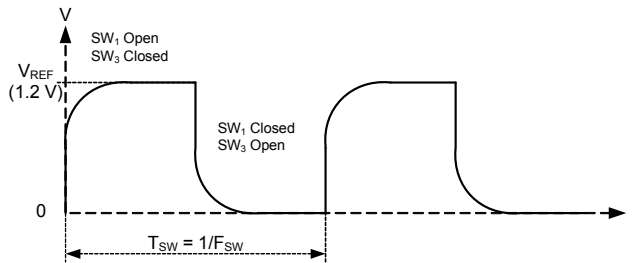


图 25-8. 传感器电容上的电压



通过公式 23-4，可以计算出从 AMUXBUS A 输出的平均电流。

$$I_S = C_S F_{SW} V_{REF} \quad \text{公式 25-3}$$

Sigma Delta 转换器每次扫描一个传感器。通过 AMUXBUS A，可以选择一个 GPIO 单元，并将该单元连接至 Sigma Delta 转换器的引脚上，如图 25-3 所示。AMUXBUS A 和 GPIO 单元开关共同组成该模拟复用器（请参见图 25-4 中的 SW₃）。AMUXBUS A 可以连接到支持 CSD 的所有 PSoC 4 引脚上。欲了解 CSD 引脚的信息，请参考器件数据手册中的内容。

请参见第 63 页上的 I/O 系统章节，了解如何为感应、屏蔽和连接 C_{MOD} 配置 GPIO 单元。

25.4.2 CapSense 时钟发生器

该模块和系统资源中的可编程时钟分频器一起生成开关时钟 F_{SW} 和调制时钟 F_{MOD}，如图 25-3 所示。有关详细信息，请参见第 77 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。

可以使用系统资源中的任意两个可编程时钟分频器来对 HFCLK 进行分频，从而生成所需的频率。有关详细信息，请参见第 77 页上的时钟系统章节。通常使用两个级联分频器。其中，第一个时钟分频器生成调制时钟，第二个生成开关时钟。

然而，最终的开关时钟频率取决于 CapSense 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。想要选择该选项，请在 CSD_CONFIG 寄存器 ‘1’ 中设置 BYPASS_SEL 位。
- 二分频：对时钟频率进行二分频。想要选择该选项，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，可以降低 CapSense 系统中的 EMI。想要选择该选项，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。具体操作为：通过设置 PRS_12_8 位，可以选择 12 位伪随机序列；通过清除该位选择 8 位 PRS。

如果选中了 PRS，则最大开关频率为：

$$F_{SW(maximum)} = \frac{F_{in}}{2} \quad \text{公式 25-4}$$

其中：F_{in} 是开关分频器的输出频率。最小频率为：

$$F_{SW(minimum)} = \frac{F_{in}}{PRS \text{ length}-1} \quad \text{公式 25-5}$$

其中：PRS 的长度为 12 位或 8 位。平均开关频率为：

$$F_{SW(average)} = \frac{F_{in}}{4} \quad \text{公式 25-6}$$

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。

25.4.3 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 25-3 所示。

Sigma delta 调制器以打开 / 关闭方式来控制 8 位 IDAC 的电流。该 IDAC 被称为调制 IDAC。被称为补偿 IDAC 的 7 位 IDAC 始终处于打开或关闭状态。

Sigma delta 转换器可在单 IDAC 模式或双 IDAC 模式下运行。在单 IDAC 模式下，补偿 IDAC 始终处于关闭状态。在双 IDAC 模式下，补偿 IDAC 始终处于打开状态。

Sigma Delta 转换器还需要一个外部积分电容 C_{MOD} ，如图 25-1 所示。 C_{MOD} 的推荐值为 2.2 nF。PSoC 4 具有一个专用的 C_{MOD} 引脚。欲了解更详细的信息，请参考器件数据手册中的引脚分布一节。

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，使其电压平衡。
- IDAC 灌电流模式：在该模式下，IDAC 从 C_{MOD} 消耗电流，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} 上电压发生的微小变化，将调制 IDAC 电流切换为打开或关闭状态，从而保持 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，原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原始计数值。

$$Rawcount = 2^N \frac{V_{REF} F_{SW}}{I_{MOD}} C_S \quad \text{公式 25-7}$$

同样，IDAC 灌入模式中原始计数的近似值为：

$$Rawcount = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{MOD}} C_S \quad \text{公式 25-8}$$

在这两种情况下，原始计数值均与传感器电容值 C_S 成正比。通过固件来处理原始计数值，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 始终处于打开状态。如果 I_{COMP} 是补偿 IDAC 电流，则通过下面的公式可计算出 IDAC 拉电流模式下的原始计数值：

$$Rawcount = 2^N \frac{V_{REF} F_{SW}}{I_{MOD}} C_S - 2^N \frac{I_{COMP}}{I_{MOD}} \quad \text{公式 25-9}$$

IDAC 灌电流模式下的原始计数值可通过公式 16-10 计算得出。

$$Rawcount = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{MOD}} C_S - 2^N \frac{I_{COMP}}{I_{MOD}} \quad \text{公式 25-10}$$

请注意，原始计数值始终为正值。

应将硬件参数（如： I_{COMP} 、 I_{MOD} 和 F_{SW} ）调试为最佳值，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](#) 中的内容。

CSD_CONFIG、CSD_COUNTER 和 CSD_IDAC 寄存器控制着 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：它是 CSD 模块的主设备使能位。必须将该位设置为 ‘1’，以使 CSD 操作有效。
- CSD_CONFIG 寄存器中的 POLARITY 位：用于选择 IDAC 灌电流模式或者 IDAC 拉电流模式。0 值表示 IDAC 拉电流模式，1 值表示 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，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，它还会启动各 IDAC。

必须准确配置 IDAC，以能进行 CSD 操作。有关详细信息，请参考 PSoC 4200L 系列：PSoC 4 寄存器技术参考手册中介绍的 CSD_IDAC 寄存器。

CSD_COUNTER 寄存器用于对当前选定的传感器进行采样，并读取结果。每当以调制时钟频率对比较器进行采样时，如果采样结果为 ‘1’，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’ 写入到该字段内。CSD_COUNTER 寄存器中的 16 位周期字段用于初始化将电容值转换为数字值的操作。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可以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，字段值等于传感器中电容值相应的原始计数值。

25.5 CapSense CSD 屏蔽

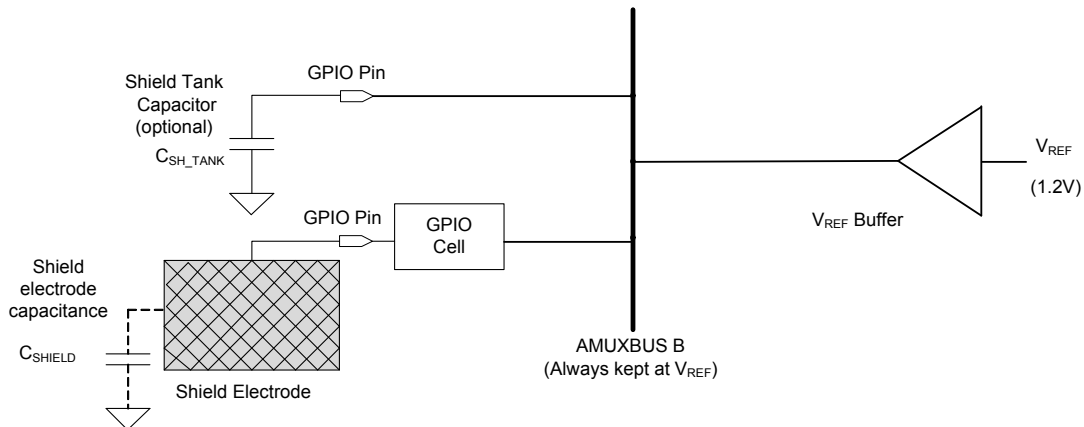
PSoC 4 CapSense 支持用于防水和接近感应性能的屏蔽电极。对于防水性能，屏蔽电极总是保持为与传感器相同的电位。PSoC 4 CapSense 具有一个屏蔽电路，该电路会使用传感器开关信号的副本来驱动屏蔽电极（请参考第 277 页上的 GPIO 单元中电容 - 电流转换器），这样可以避免传感器与屏蔽电极间潜在的差异。欲了解屏蔽的基本信息，请参考 PSoC 4 CapSense 设计指南一节的内容。

在感应电路中，Sigma Delta 转换器保持使 AMUXBUS A 的电压等于 V_{REF} （请参考第 279 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式工作；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 的电压为 V_{REF} 的方法，有以下两种不同配置。

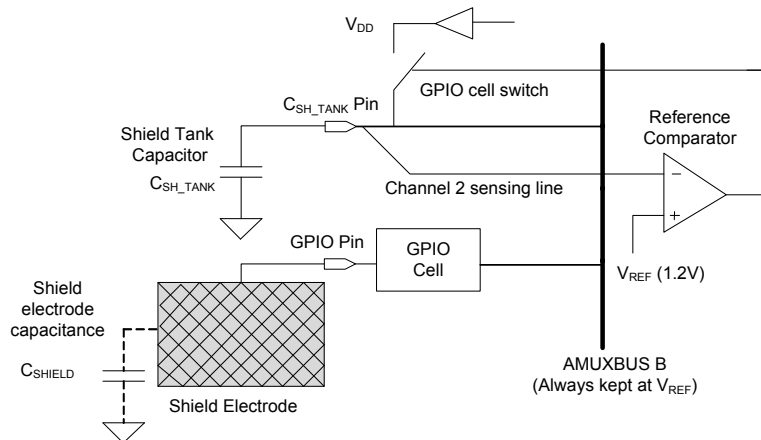
- 使用 V_{REF} 缓冲区驱动屏蔽：在此配置中，会使用一个电压缓冲区将 AMUXBUS B 驱动为 V_{REF} ，如图 25-9 所示。推荐使用外部电容 C_{SH_TANK} 来降低开关跃变。设置 CSD_CONFIG 寄存器中 REBUF_OUTSEL 位可以将缓冲区输出连接至 AMUXBUS B。同一寄存器中的 REBUF_DRV 位字段可用来设置缓冲区的驱动能力。将 ‘0’ 写入到该字段可禁用该缓冲区；将 1、2 和 3 写入到该字段可分别选择低电流、中电流和高电流驱动模式。

图 25-9. 使用 V_{REF} 缓冲区驱动屏蔽



- 使用 GPIO 单元预充电驱动屏蔽：该配置需要一个外部电容 C_{SH_TANK} ，如图 25-10 所示。通过特殊的 GPIO 单元和一个参考比较器，可以给电容 C_{SH_TANK} 充电，这样 AMUXBUS B 会保持其电压为 V_{REF} 。参考比较器始终监控 C_{SH_TANK} 电容上的电压，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），可将参考比较器连接至 C_{SH_TANK} 电容上，如图 25-10 所示。

图 25-10. 使用 GPIO 预充电驱动屏蔽



GPIO 单元预充电功能仅适用于 C_{SH_TANK} 固定引脚。欲了解更详细的信息，请参考器件数据手册中器件引脚分布一节。

CSD_CONFIG 寄存器中的 COMP_MODE 位可选择参考缓冲区预充电或 GPIO 预充电模式：0 值表示参考缓冲区预充电，1 值表示 GPIO 预充电。

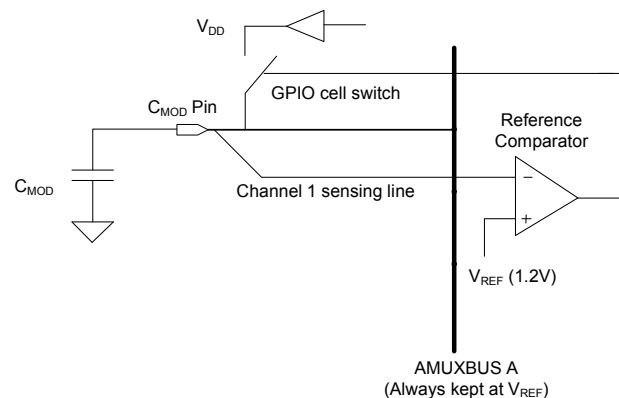
25.5.1 C_{MOD} 预充电

第一次使能 CapSense 硬件时， C_{MOD} 上的电压为 0 V。然后，Sigma Delta 转换器会缓慢地为 C_{MOD} 充电，使其电压达到 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} 预充电指的是快速初始化 C_{MOD} 上的电压，并使之达到 V_{REF} 的过程。使用预充电方式可缩短 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 25-9）。要想使用 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 能够正常操作。屏蔽被禁用时，在预充电期间 V_{REF} 缓冲区输出始终连接着 AMUXBUS A，然后再断开连接。
- 使用 GPIO 单元进行预充电：在该配置中，通过一个特殊的 GPIO 单元和一个参考比较器，为电容 C_{MOD} 充电，使之电压达到 V_{REF} 。GPIO 单元预充电功能仅适用于 C_{MOD} 固定引脚。欲了解更详细的信息，请参考器件数据手册中的引脚分布一节。用于该目的的比较器与预充电 CSH_TANK 的参考比较器相同。通过 CSD_CONFIG 寄存器中的 COMP_PIN 位，可以选择连接至参考比较器的电容。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 25-11 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 25-10 所示。请注意：必须正确配置 GPIO 单元，这样才能进行 GPIO 单元预充电。

图 25-11. 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 感应线路还用来将 C_{MOD} 连接至 Sigma delta 调制器中的感应比较器。通过将 CSD_CONFIG 寄存器中的 SENSE_INSEL 位设置为‘1’，可以使能该选项。清除该位可使 C_{MOD} 通过 AMUXBUS A 连接至感应比较器。

25.6 通用资源：IDAC 和比较器

如果 CapSense 模块不用于触摸感应，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 $1.2\text{ V } V_{\text{REF}}$ （请参见图 25-3）。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 279 页上的 Sigma Delta 转换器 中所介绍。

如果需要将 AMUXBUS 用于其他目的，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 282 页上的 CMOD 预充电 中的介绍。比较器的输出可以连接到多个 GPIO，请参见第 63 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 $0\sim 306\text{ }\mu\text{A}$ （ $1.2\text{ }\mu\text{A/位}$ ）或 $0\sim 612\text{ }\mu\text{A}$ （ $2.4\text{ }\mu\text{A/位}$ ）。7 位 IDAC 的工作电流范围为 0 到 $152.4\text{ }\mu\text{A}$ （ $1.2\text{ }\mu\text{A/位}$ ）或 0 到 $304.8\text{ }\mu\text{A}$ （ $2.4\text{ }\mu\text{A/位}$ ）。

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）模式以及 CSD 和通用模式。表 25-1 介绍了在每个模式下如何将 IDAC1 和 IDAC2 连接到 AMUXBUS A 和 AMUXBUS B。

表 25-1. 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

有关详细信息，请参考 PSoC 4200L 系列：PSoC 4 寄存器技术参考手册中介绍的 CSD_IDAC 寄存器。通过 CSD_CONFIG 寄存器可以使能 IDAC，并设置极性，如第 279 页上的 Sigma Delta 转换器 所介绍。有关如何将 GPIO 连接至 AMUXBUS A 和 AMUXBUS B 的详细信息，请参考第 63 页上的 I/O 系统章节。端口 5、端口 8 和端口 9 引脚被固定连接到 CSD1 模块，因此，CSD0 模块不能控制它。可将 CSD0 模块连接到所有引脚（端口 5、端口 8 和端口 9 的引脚除外）。

25.7 寄存器列表

表 25-2. 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	该寄存器用于初始化对选定电容传感器进行的采样，并读取转换结果。
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。

26. 温度传感器



PSoC[®] 4 使用片上温度传感器来测量内部 Die（裸片）的温度。传感器包含一个在二极管配置中连接的晶体管。

26.1 特性

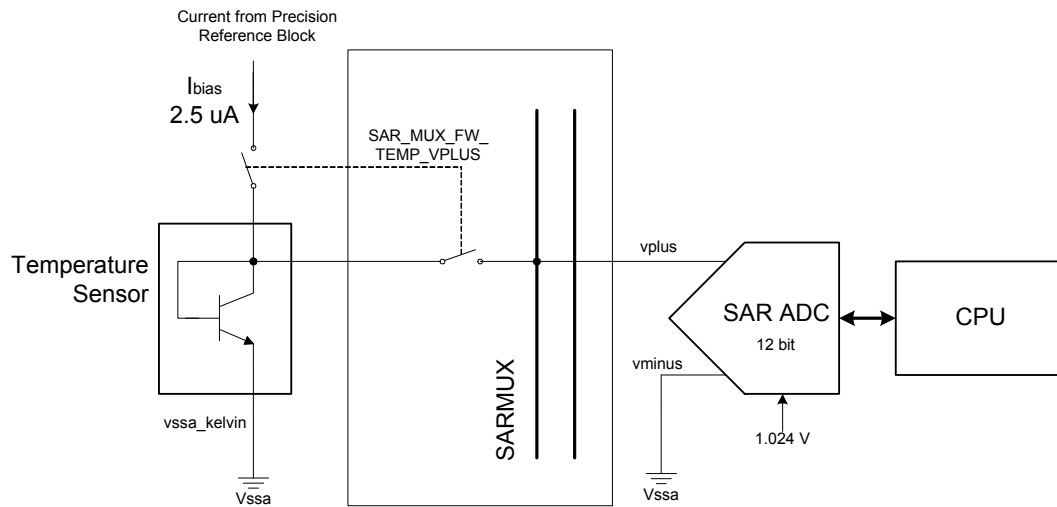
温度传感器的特性包括：

- 温度范围为 -40°C 到 $+85^{\circ}\text{C}$ ，精度为 $\pm 5^{\circ}\text{C}$
- 当使用一个 12 位 SAR ADC，并且参考电压为 1.024 V 时，分辨率为 $0.5^{\circ}\text{Celsius/LSB}$ （尚未放大）
- 10 μs 的建立时间

26.2 工作原理

温度传感器包括一个外形为二极管的单端双极结型晶体管（BJT）。在恒定集电极电流和零集电极 - 基极电压条件下，温度对该二极管的基极 - 发射电压（ V_{BE} ）产生的影响非常大。可利用该属性计算裸片（die）的温度，具体是通过使用 SAR ADC 测量晶体管的 V_{BE} ，如图 26-1 所示。

图 26-1. 温度感应机制



通过使用 SAR ADC 可以测量传感器的模拟输出（ V_{BE} ）。可以通过以下公式使用 ADC 结果计算 Die 温度（单位为 $^{\circ}\text{C}$ ）：

$$\text{Temp} = (A \times \text{SAR}_{\text{out}} + 2^{10} \times B) + T_{\text{adjust}} \quad \text{公式 26-1}$$

- 温度是指单位为 $^{\circ}\text{C}$ 的斜率补偿温度，它的表示形式为 32 位（16.16）定点数。
- ‘A’ 是一个 16 位的乘数常量。通过使用 PSoC 4 系列的数据计算两点间斜率，可以确定 A 值。可以按照以下公式计算该值：

$$A = (\text{signed int}) \left(2^{16} \left(\frac{100^{\circ}\text{C} - (-40^{\circ}\text{C})}{\text{SAR}_{100^{\circ}\text{C}} - \text{SAR}_{-40^{\circ}\text{C}}} \right) \right)$$

公式 26-2

其中：

$\text{SAR}_{100^{\circ}\text{C}}$ 表示温度为 100°C 时 ADC 的计数值

$\text{SAR}_{-40^{\circ}\text{C}}$ 表示温度为 -40°C 时 ADC 的计数值

常量 ‘A’ 被存储在 SFLASH_SAR_TEMP_MULTIPLIER 寄存器内。

- ‘B’ 是一个 16 位的偏移值。针对每一个裸片（die）计算出 B 值。计算操作所涉及的因素包括加工技术的差异和芯片上实际的偏置电流（ I_{bias} ）。可以按照以下公式计算出该值：

$$B = (\text{unsigned int}) \left(2^6 \times 100^{\circ}\text{C} - \left(\frac{A \times \text{SAR}_{100^{\circ}\text{C}}}{2^{10}} \right) \right)$$

公式 26-3

其中：

$\text{SAR}_{100^{\circ}\text{C}}$ 表示温度为 100°C 时的 ADC 计数值

常量 ‘B’ 被存储在 SFLASH_SAR_TEMP_OFFSET 寄存器内。

- T_{adjust} 是斜率修正因子（单位为 $^{\circ}\text{C}$ ）。通过使用斜率修正因子可以修正双斜率的温度传感器。根据得到的结果（未经过斜率修正）计算该值， $T_{\text{initial}} = (A \times \text{SAR}_{\text{out}} + 2^{10} \times B)$ 。如果该值大于中心值（ 15°C ），则可以按照下面公式计算 T_{adjust} 。

$$T_{\text{adjust}} = \left(\frac{0.5^{\circ}\text{C}}{100^{\circ}\text{C} - 15^{\circ}\text{C}} \times (100^{\circ}\text{C} \times 2^{16} - T_{\text{initial}}) \right)$$

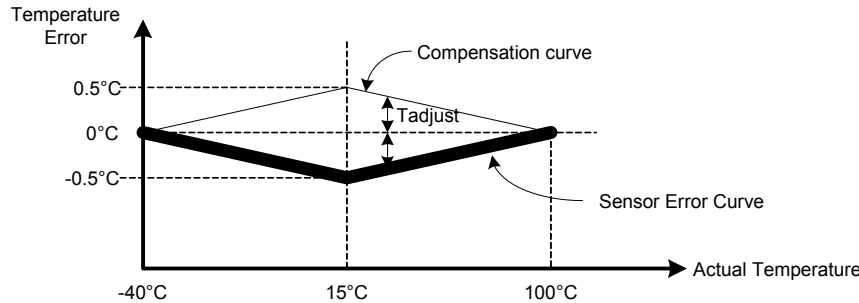
公式 26-4

如果该值小于中心值，则可以按照下面公式计算 T_{adjust} 。

$$T_{\text{adjust}} = \left(\frac{0.5^{\circ}\text{C}}{40^{\circ}\text{C} + 15^{\circ}\text{C}} \times (40^{\circ}\text{C} \times 2^{16} - T_{\text{initial}}) \right)$$

公式 26-5

图 26-2. 温度误差补偿



注意： A 和 B 是工厂校准过程中存储在闪存内的 16 位常量。请注意，只有 SAR ADC 运行于 12 位分辨率并且参考电压为 1.024 V 的情况时，这些常量才有效。

26.3 温度传感器配置

温度传感器输出通过专用开关连接到 SAR ADC 的正向输入端。这些开关可以由序列发生器、固件或数字系统互连（DSI）控制。请参考第 217 页上的 SAR ADC 章节，详细了解如何使用 ADC 读取温度传感器输出。

26.4 算法

1. 启用 SARMUX 和 SAR ADC。
2. 将 SAR ADC 配置为单端模式，其中 $V_{\text{NEG}} = V_{\text{SS}}$ ， $V_{\text{REF}} = 1.024 \text{ V}$ ，分辨率为 12 位，并且结果为右对齐。
3. 启用温度传感器。
4. 从 SAR ADC 中获取数字输出。
5. 分别从 SFLASH_SAR_TEMP_MULTIPLIER 和 SFLASH_SAR_TEMP_OFFSET 上提取 ‘A’ 值和 ‘B’ 值。
6. 使用线性公式（公式 26-1）计算裸片温度。

例如， $A = 0xBC4B$ 和 $B = 0x65B4$ 。假设在给定的温度条件下，SAR ADC 的输出 (V_{BE}) 为 $0x595$ 。

这时，固件会进行以下计算：

- a. 将 A 乘以 V_{BE} ： $0xBC4B \times 0x595 = (-17333)_{10} \times (1429)_{10} = (-24768857)_{10}$
- b. 将 B 乘以 1024： $0x65B4 \times 0x400 = (26036)_{10} \times (1024)_{10} = (26660864)_{10}$
- c. 将第一步和第二步得到的结果相加，从而得到 T_{initial} 值： $(-24768857)_{10} + (26660864)_{10} = (1892007)_{10} = 0x1CDEA7$
- d. 使用 T_{initial} 值来计算 T_{adjust} ： T_{initial} 是高 16 位乘以 2^{16} ，即为 $0x1C00 = (1835008)_{10}$ 。该值超过了 15°C ($0x1C$ - 高 16 位)。使用公式 4 计算 T_{adjust} 。得到的结果为 $0x6C6C = (27756)_{10}$
- e. 将 T_{adjust} 加上 T_{initial} ： $(1892007)_{10} + (27756)_{10} = (1919763)_{10} = 0x1D4B13$
- f. 温度的整数部分为高 16 位 = $0x001D = (29)_{10}$
- g. 温度的十进制部分为低 16 位 = $0x4B13 = (0.19219)_{10}$
- h. 将 f 和 g 步骤的结果结合起来，可得到： $\text{Temp} = 29.19219^{\circ}\text{C} \sim 29.2^{\circ}\text{C}$

26.5 寄存器

姓名	描述
SAR_MUX_SWITCH0	该寄存器具有 SAR_MUX_FW_TEMP_VPLUS 字段，用于将温度传感器连接到 SAR MUX 终端。
SAR_MUX_SWITCH_STATUS	该寄存器提供了连接到 SAR MUX 的温度传感器开关的状态。
SFLASH_SAR_TEMP_MULTIPLIER	如公式 26-1 所定义的乘数常量 ‘A’。
SFLASH_SAR_TEMP_OFFSET	如公式 26-1 所定义的常量 ‘B’。

温度传感器

节 F: 编程和调试

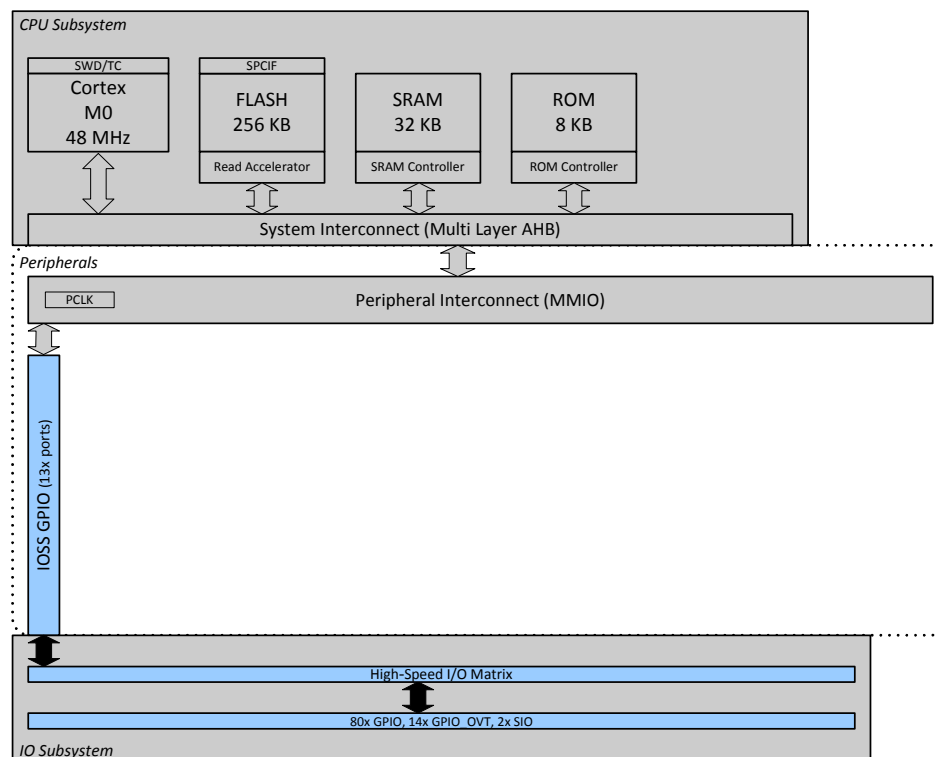


本部分包括以下章节:

- 第 291 页上的编程与调试接口章节
- 第 299 页上的非易失性存储器编程章节

系统架构

编程和调试框图



27. 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编程或调试的通信网关。外部器件可以是赛普拉斯提供的编程器和调试器，也可以是支持 PSoC 4 编程和调试功能的第三方器件。串行线调试 (SWD) 接口用作外部器件与 PSoC 4 间的通信协议。

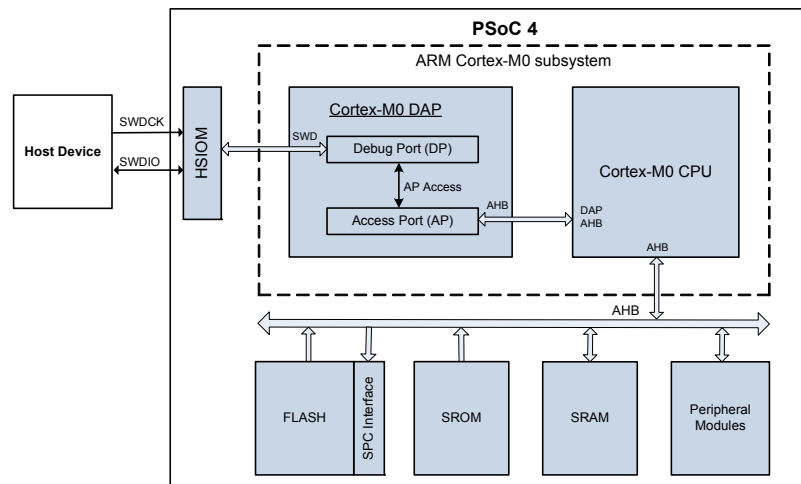
27.1 特性

- 通过 SWD 接口进行编程和调试
- 调试过程中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存储器和寄存器（包括内核处于运行或停止状态下的 Cortex-M0 寄存器组）进行读和写访问

27.2 功能说明

图 27-1 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) 用作编程和调试接口。外部编程器或调试器（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 “目标器件”的 DAP 进行通信。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 63 页上的 I/O 系统章节。

图 27-1. PSoC 4 编程和调试接口



DAP 使用 ARM 指定的高级和性能总线 (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 主设备 — 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。

27.3 串行线调试（SWD）接口

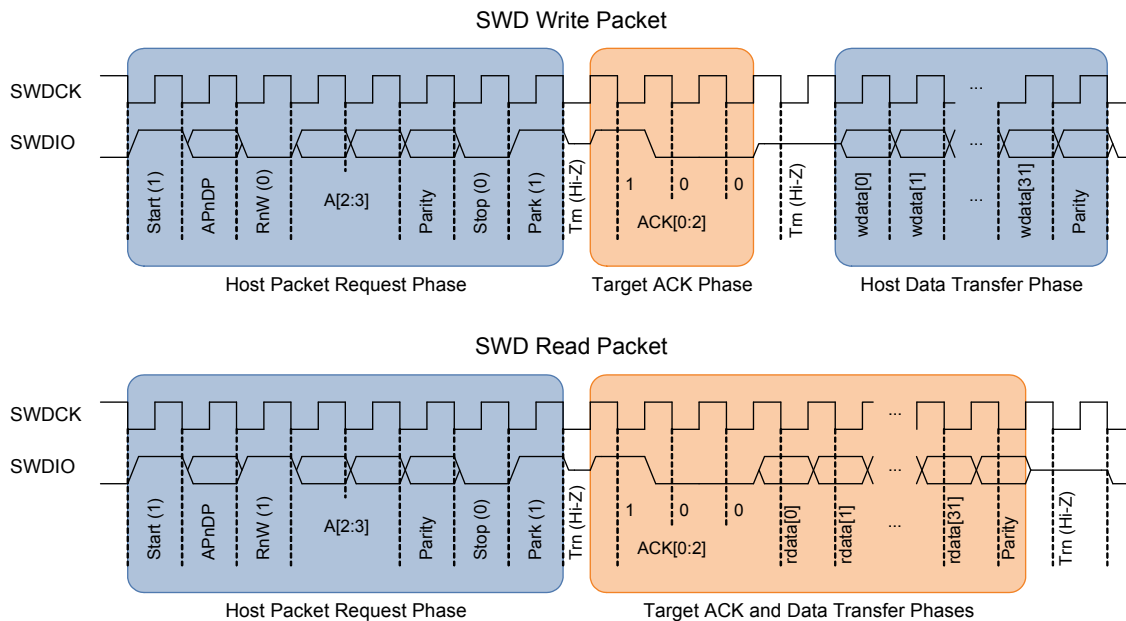
PSoC 4 的 Cortex-M0 支持通过 SWD 接口进行的编程和调试。SWD 协议是基于数据包的串行数据操作协议。在引脚等级上，它使用一个单一双向数据信号（SWDIO）和一个单向时钟信号（SWDCK）。主机编程器始终驱动时钟线，同时，主机或目标将驱动数据线。完整的数据传输（一个 SWD 数据包）需要 46 个时钟周期，并包括三个阶段：

- **主机数据包请求阶段** — 主机向 PSoC 4 目标器件发送一个请求。
- **目标器件应答响应阶段** — PSoC 4 目标器件给主机发送一个应答。
- **数据传输阶段** — 根据传输方向，主机或目标器件将数据写入到总线上。

当 SWDIO 线的控制权从主机转交给目标器件（或反过来转交）时，在一个反转周期（ T_{rn} ）内，没有任何器件驱动该信号线，它会浮动于高阻态（Hi-Z）。根据转交情况，反转周期可能等于半个或一个半时钟周期。

图 27-2 显示了读取和写入 SWD 数据包的时序框图。

图 27-2. 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 表示从目标器件‘读取’，1b0 表示‘写入’目标器件。
 - 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 27-3 和表 27-4，了解相关的定义。**注意：**地址位的最低有效位（LSB）先被传送。
 - e. 奇偶校验位包含 APnDP、RnW 和 ADDR 位的奇偶校验。它是一个偶数校验位，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’。
如果奇偶校验位不正确，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. 目标器件应答响应阶段：SWDIO 由目标器件驱动
 - a. ACK[2:0] 位表示从目标器件到主机的响应，并指出响应结果是成功还是失败。请参见表 27-1，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 由目标器件或主机驱动（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. 数据的奇偶校验位指示需要读取或写入的数据的奇偶情况。它是一个偶数校验位，表示将该位与数据位进行 XOR 运算时，得到的结果将为‘0’。
如果奇偶校验位指出一个数据错误，那么，需要对它进行纠正。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，如果目标器件检测到奇偶错误，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。如果时钟不能自由运行，或要求时钟在空闲模式下自由运行，则建议在某两个 **SWD** 数据包传输之间生成三个或更多的虚拟时钟周期。

SWDIO 的状态为高电平时，在 50 个或更多时钟周期内给 **SWDCK** 线提供时钟脉冲，可以复位 **SWD** 接口。要想返回空闲状态，只要在 **SWDIO** 为低电平时给 **SWDCK** 线提供一个周期的时钟脉冲。

27.3.1 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。在主机数据包请求阶段，主机会驱动 **SWDIO** 线。如果主机正向目标器件写入数据，它也会在数据传输阶段内驱动 **SWDIO** 线。当主机驱动 **SWDIO** 线时，它会在 **SWDCK** 的下降沿上写入新位，同时，目标器件会在 **SWDCK** 的上升沿上读取该位。在目标器件应答响应阶段期间，目标器件会驱动 **SWDIO** 线。如果目标器件正在读取数据，则它也会在数据传输阶段期间驱动 **SWDIO** 线。当目标器件驱动 **SWDIO** 线时，它会在 **SWDCK** 的上升沿上写入新位，同时，主机会在 **SWDCK** 的下降沿上读取该位。

表 27-1 和图 27-2 介绍了 **SWDIO** 位的写入和读取的时序情况。

表 27-1. **SWDIO** 位的写入和读取时序

SWD 数据包相位	SWDIO 边沿	
	下降沿	上升沿
主机数据包请求	主机写入	目标器件读取
主机数据传输		
目标器件响应	主机读取	目标器件写入
目标器件数据传输		

27.3.2 ACK 的详细信息

应答 (**ACK**) 位字段用于与上一次传输的状态通信。**OK ACK** 表示上一次数据包传输已成功。一个 **WAIT** 响应要求一个数据阶段。**FAULT** 状态表示需要立即中止编程过程。表 27-2 显示的是 **ACK** 位字段解码的详细信息。

表 27-2. **SWD** 传输 **ACK** 响应解码

响应	ACK[2:0]
OK	3b001
WAIT	3b010
FAULT	3b100
NO ACK	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。目标器件不驱动该线，所以主机也不需检查奇偶位。
- 对于 **WAIT** 响应，如果数据操作为写入，**PSoC 4** 会忽略数据阶段。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校验位。
- **WAIT** 响应表示 **PSoC 4** 正在处理上一次数据操作。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。

27.3.3 反转 (Trn) 周期的详细信息

如果主机进行写入操作，数据包请求阶段和 **ACK** 阶段间会有一个反转周期；**ACK** 阶段和数据阶段间也会有一个反转周期，如图 27-2 所示。根据 **SWD** 协议，主机和目标器件都使用 **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，目标器件在 **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。

27.4 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 标识代码等的寄存器。访问端口包含多个寄存器，允许外部器件能够通过 DAP-AHB 接口访问 Cortex-M0。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](#)。

27.4.1 调试端口（DP）寄存器

表 27-3 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 ‘0’。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 架构规范](#)。

表 27-3. 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，以及清除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)	读取缓冲区寄存器	该寄存器保存了最后一次 AP 读操作的结果。

27.4.2 访问端口（AP）寄存器

表 27-4 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 ‘1’。两个地址位（A[3:2]）用于选择不同的 AP 寄存器。

表 27-4. 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	该寄存器用于保存需要读取或写入 TAR 寄存器所指定的地址的 32 位数据

27.5 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 4200L 器件编程规范](#)，了解编程时所需的编程算法、时序规范，以及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。

27.5.1 获取 SWD 端口

27.5.1.1 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。请看考 器件数据手册，了解更多有关 SWD 引脚的信息。

如果器件中的两对 SWD 引脚都可用，则通过监控闪存区中的 SWD_CONFIG 寄存器，可以选择两对 SWD 引脚中的一对进行编程和调试操作。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。如要选用辅助的 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。

27.5.1.2 SWD 端口获取序列

编程器件的第一步是为主机获取目标器件的 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 SWD 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 命令。复位过程需要超过 49 个 SWDCK 时钟周期，并且还要求 SWDIO 上的信号处于高电平状态。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 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。

27.5.2 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。在测试模式控制寄存器 (MODE 寄存器) 下设置 TEST_MODE 位 (位 31) 便能完成该操作。进入器件编程模式前，还要配置调试端口。更多有关进入编程模式的时序规范和伪代码的信息，请参考 [PSoC 4200L 器件编程规范文档](#)。

27.5.3 SWD 编程子程序执行

当器件处于编程模式时，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。第 299 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 4200L 器件编程规范文档](#)。

27.6 PSoC 4 SWD 调试接口

Cortex-M0 DAP 拥有两类调试功能，包括：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。

27.6.1 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。它还包含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）— 通过使用该寄存器，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 CM0_DCRSR 寄存器选择）进行读取或写入操作的数据。
- 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。

27.6.2 断点单元（BPU）

BPU 提供了提取指令时的断点功能。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 Cortex-M0 中的 BKPT 指令，可以创建任意软件断点的数量。BPU 具有两类寄存器。

- 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）。
- 每个硬件断点具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：当某个指令提取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。

27.6.3 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 支持了两个观察点。它还通过 PC 采样寄存器提供外部编程计数器采样功能。该采样寄存器用于为编程计数器进行非侵入性粗调配置。下面将介绍 DWT 中的最重要寄存器。

- 观察点比较（CM0_DWT_COMPx）寄存器存储了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）寄存器存储了忽略掩码，这些忽略掩码适用于相关观察点中的地址范围匹配。
- 观察点功能（CM0_DWT_FUNCTIONx）寄存器存储了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。发生相应的观察点事件时，状态位将被置位。
- 观察点比较器 PC 采样寄存器（CM0_DWT_PCSR）存储了编程计数器的当前值。该寄存器用于对编程计数器寄存器进行的粗调、非侵入性配置。

27.6.4 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 接口对所有寄存器进行访问；Cortex-M0 DAP 中的 SWD 调试端口（SW-DP）将 SWD 数据包转换为通过 DAP-AHB 接口进行的适当寄存器访问。

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 时，可以获取 SWD 端口。如果要使调试操作发生在 SWD 接口上，请勿将相应的引脚用于其他任何目的。请参考第 63 页上的 [I/O 系统章节](#) 以了解如何配置 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 LCD 和 GPIO。如果需要调试，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 109 页上的 [器件安全性章节](#) 中所述。因此，只有 OPEN（打开）保护模式支持器件调试。此外，当器件进入休眠或停止模式时，外部调试器会与目标器件断连。器件进入活动模式后，必须恢复该连接。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开。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。

27.7 寄存器

表 27-5. 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 采样寄存器

28. 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。本章节介绍了属于器件编程的各功能，如擦除、写入、编程和校验和计算等。Cypress 公司提供的编程器或其他第三方的编程器可以通过使用这些功能，结合使用应用十六进制文件中的数据对 PSoC 4 器件进行编程。此外，在执行引导加载（Bootload）操作时，用户也可以使用这些功能来更新部分闪存存储器。

28.1 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 执行的阻塞和非阻塞的闪存编程和擦除操作

28.2 功能说明

闪存编程相关的所有操作均是通过调用系统函数来实现的。在特权操作模式下，在 SROM 外执行这些系统调用。用户无权读取或修改 SROM 代码。DAP 或 CM0 CPU 通过将函数操作码和函数参数写入系统性能控制器接口（SPCIF）输入寄存器内，然后请求 SROM 执行函数来调用系统函数。根据函数操作码，系统性能控制器（SPC）将执行 SROM 中的相应系统调用，并更新 SPCIF 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 SPCIF 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。这样可使存储电荷得到平衡，从而延长闪存的使用寿命。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器内，然后通过使用闪存写函数将该数据传输到闪存。

通过将各命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 4200L 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。这类编程通常在引导加载操作期间更新闪存存储器时使用，或用于满足其他应用要求，如更新存储在闪存存储器内的查找表。DAP 或 CPU 对闪存存储器进行的所有写操作都通过 SPCIF 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，否则一部分闪存会发生意外变化。复位源（请参考 [第 105 页上的复位系统章节](#)）包括 XRES 引脚、软件复位以及看门狗；需要确保不会无意激活这些复位源。另外，低电压检测电路应该配置为生成中断而不是复位。

注意：PSoC 4 实现了一个用户监控闪存（SFlash），该闪存可用于存储应用特定的信息。这些行不属于十六进制文件；可选择性对其进行编程操作。

注意：除了用户定义的各行外，PSoC 4 器件还具有监控闪存行。用户可以使用该监控闪存的一部分进行存储应用特定的信息。更多有关监控闪存的结构以及编程监控闪存区的方法的信息，请参考 [PSoC 4200L 编程规范](#)。

28.3 系统调用实现

一个系统函数包括以下部分：

- 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
这两个 key 均被传送，以确保不会无意中启动用户系统函数。如果参数 key1 和 key2 不正确，SRAM 将不会执行函数并返回错误代码。除了这两个参数以外，根据被调用的特定函数，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：每个系统调用将返回 CPU 或 DAP 可读取的一个 32 位状态，以验证成功或确定失败的原因。

28.4 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。从某个过程调用阻塞系统函数时，CPU 会跳转到 SRAM 中相应的代码。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经完成。

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 则在该过程中停止。

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 在对闪存进行擦除或编程时不能执行闪存中的代码，因此只能通过在 SRAM 外执行的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，执行闪存提取操作时，结果将为未定义并可能返回一个总线错误，而且将触发一个硬故障。

系统性能控制器（SPC）模块能够生成擦除和编程闪存存储器时所需的正确序列高电压脉冲。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；进行非阻塞写操作时，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。

将在本节后面的内容中介绍用于一个非阻塞写系统调用以及在 SRAM 外执行用户代码的伪代码。

28.4.1 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：
 - a. 将函数参数写入到 CPUSS_SYSARG 寄存器内：该方法用于从 CPUSS_SYSARG 寄存器提取参数的函数。必须根据相应的系统调用表所指定的序列将各个参数写入到 32 位 CPUSS_SYSARG 寄存器内。
 - b. 将函数参数写入到 SRAM 内：该方法用于从 SRAM 提取参数的函数。首先，应该按照所指定的序列将这些参数写入 SRAM 的连续地址内。然后，将 SRAM 的起始地址（即第一个参数的地址）写入 CPUSS_SYSARG 寄存器内。该起始地址应该为字对齐（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来指定系统调用并将其启用：将 8 位操作码写入 CPUSS_SYSREQ 寄存器中的 SYSCALL_COMMAND 位（[15:0]）内。这个操作码被放置在低八位 [7:0] 内，0x00 被写入到高八位 [15:8] 内。要启动该系统调用，需要设置 CPUSS_SYSREQ 寄存器中的 SYSCALL_REQ 位（31）。设置该位会触发一个不可屏蔽的中断，该中断使 CPU 跳转到操作码参数参照的 SRAM 代码。
3. 等待系统调用以完成执行操作：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_SYSREQ 寄存器中的 PRIVILEGED 位和 SYSCALL_REQ 位，以检查系统调用是否已经完成。系统调用完成时，这两位均被清除。执行时间最多占用 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，而不进行后面步骤。请注意，CPU 应用代码在系统调用执行期间不能轮询这些位，这一点与 DAP 不同。这是因为在系统调用期间，CPU 在 SRAM 外执行代码。该执行操作从 SRAM 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，则状态代码将为 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 28-1 中的内容。
5. 提取返回值：对于返回芯片 ID 和校验和等数值的系统调用，CPU 或 DAP 需要读取 CPUSS_SYSREQ 和 CPUSS_SYSARG 寄存器来提取所返回的值。

28.5 系统调用

表 28-1 列出了 PSoC 4 所支持的所有系统调用、其功能说明以及在各个器件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 109 页上的器件安全性章节中的内容。请注意，CPU 不能调用某些系统调用，如下表所示。随后是各个系统调用的详细内容。

表 28-1. 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 和版本 ID	✓	✓	–	✓
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 256 个字节	✓	–	–	✓
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行闪存行内	✓	–	–	✓
编程行	将页锁存缓冲区中的数据编程到闪存行内	✓	–	–	✓
全部擦除	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	✓	–	–	
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计算校验和	✓	✓	–	✓
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监控闪存（行 0）内	✓	✓	–	
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行闪存行内。在编程 / 擦除脉冲期间，用户可以执行 SRAM 中的代码。该函数仅适用于 CPU 访问	–	–	–	✓
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，用户可以执行 SRAM 中的代码。该函数仅适用于 CPU 访问	–	–	–	✓
恢复非阻塞	恢复非阻塞写入行或非阻塞编程行。该函数仅适用于 CPU 访问	–	–	–	✓

28.5.1 芯片 ID

该函数返回一个 12 位系列 ID、一个 16 位芯片 ID、一个 8 位版本 ID 和当前的器件保护模式。这些值被返回到 CPUSS_SYSARG 和 CPUSS_SYSREQ 寄存器。各参数被传送到 CPUSS_SYSARG 和 CPUSS_SYSREQ 寄存器内。

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	有关不同器件编号的芯片 ID 值，请查看 器件数据手册
位 [15:8]	芯片 ID 高	
位 [19:16]	次要版本 ID	请参见 PSoC 4200L 编程规范，了解这些值
位 [23:20]	主要版本 ID	

地址	返回值	说明
位 [27:24]	0xXX	未使用（无需关注）
位 [31:28]	0xA	成功的状态代码
CPUSS_SYSREQ 寄存器		
位 [11:0]	系列 ID	PSoC 4200L 的系列 ID 为 0x0A0
位 [15:12]	芯片保护	请参阅 第 109 页上的器件安全性章节
位 [31:16]	0XXXXX	未使用

28.5.2 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写入和闪存擦除 API 等函数前，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，则闪存写入和闪存擦除 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。

28.5.3 加载闪存字节

通过使用该函数 可以将要编程到一个闪存行中的数据加载到页锁存缓冲区内。加载范围为一个闪存行中的 1 字节到最大字节数量（即 256 个字节）。数据被加载到页锁存缓冲区内，开始于“Byte Addr”输入参数指定的地址。加载到页锁存缓冲区内数据被保持，直到执行清除页锁存内容的编程操作为止。该函数的参数（包括加载到页锁存内的数据）被写入 SRAM 内；SRAM 数据的起始地址被写入 CPUSS_SYSARG 寄存器内。请注意，参数的起始地址应该是字对齐的地址。

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	页锁存缓冲区内写入数据的的起始地址 0x00 — 锁存缓冲区的字节 0 0xFF — 锁存缓冲区的字节 255
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 29 页上的 Cortex-M0 CPU 章节)
SRAM 地址 — 32'hYY + 0x04		
位 [7:0]	加载大小	写入到页锁存缓冲区内字节数量。 0x00 — 1 个字节 0xFF — 256 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 + 加载大小）		
字节 0	数据字节 [0]	要加载的第一个数据字节
.	.	.
.	.	.
字节（加载大小 -1）	数据字节 [加载大小 -1]	要加载的最后数据字节
CPUSS_SYSARG 寄存器		

地址	要写入的值	说明
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	加载闪存字节操作码
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	成功的状态代码
位 [27:0]	0xFFFFFFFF	未使用 (无需关注)

28.5.4 写入行

该函数先擦除寻址闪存行的内容, 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。该函数的参数被存储在 SRAM 内。存储参数的起始地址被写入 CPUSS_SYSARG 寄存器内。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。调用该函数前, 先调用加载闪存字节函数。只在相应的闪存行不受写保护的情况下, 该函数才能执行写操作。

更多有关信息, 请参考 PSoC 4200L 系列: 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]	0xFFFFFFFF	未使用 (无需关注)

28.5.5 编程行

该函数将页锁存缓冲区中的数据编程到闪存的寻址行内。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。调用该函数前，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受写保护的情况下，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]	0xFFFFFFFF	未使用（无需关注）

28.5.6 全部擦除

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监控闪存第 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）均被设为频率为 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，会自动执行全部擦除操作。

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	成功的状态代码
位 [27:0]	0XXXXXXXX	未使用 (无需关注)

28.5.7 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, 并返回在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区进行校验和。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 位的闪存行号 或 0x8000 — 在整个闪存存储器上进行校验和操作
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	成功的状态代码
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

28.5.8 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是独立编程的。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应该为 '0'。闪存宏的数量是需要进行编程的数量；要加载的字节数量是该宏中闪存保护字节的数量。

然后调用写保护函数，从而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，器件级保护设置将被视为 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	成功的状态代码
位 [27:24]	0xX	未使用（无需关注）
位 [23:0]	0x000000	

28.5.9 非阻塞写入行

CM0 CPU 使用非阻塞方式写入闪存行时，该函数将被使用。这样可以确保在执行写操作时，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 300 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，然后通过编程操作将新数据写到该行内。

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。发生该中断时，将调用恢复非阻塞系统调用。

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

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，只能调用 SRAM 中的非阻塞写入行函数。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，执行闪存提取操作时，结果为未定义并会返回一个总线错误，而且会触发一个硬故障。

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	成功的状态代码
位 [27:0]	0xFFFFFFFF	未使用（无需关注）

28.5.10 非阻塞编程行

CM0 CPU 使用非阻塞方式编程闪存行时，该函数将被使用。这样可以确保在执行编程操作时，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 300 页上的阻塞和非阻塞的系统调用。执行编程操作时，CPU 可以执行 SRAM 中的代码。当调用非阻塞编程行的系统函数时，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，执行闪存提取操作时，结果将为未定义并会返回一个总线错误，而且将触发一个硬故障。

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)

28.5.11 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, 需要从 SPC ISR 调用该函数三次, 调用非阻塞编程行后, 则需要从 SPC ISR 调用该函数一次。编程或擦除操作的所有阶段完成前, 不能执行其他系统调用。有关使用非阻塞函数的流程的更多信息, 请参考第 300 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	成功的状态代码
位 [27:0]	0XXXXXXXX	未使用 (无需关注)

28.6 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 28-2. 系统调用的状态代码

状态代码 (CPUSS_SYSARG 寄存器内的 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，否则该值为 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，不能使用此 API。
F0000003h	无效的页锁存地址 — 表示页锁存缓冲区超出了边界，或者输入数据的大小大于页地址。
F0000004h	无效地址 — 所提供的行 ID 或字节地址不处于可用的存储器范围内。
F0000005h	受保护的行 — 所提供的行 ID 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	正在进行系统调用 — 正在进行调用一个恢复的或非阻塞的 API。在尝试进行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，并将其设为 HF 时钟源。

28.7 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进行闪存编程操作期间在 SRAM 外执行代码。

```
#define REG(addr)          (*((volatile uint32 *) (addr)))
#define CM0_IUSER_REG      REG( 0xE000E100 )
#define CPUSS_CONFIG_REG   REG( 0x40100000 )
#define CPUSS_SYSREQ_REG   REG( 0x40100004 )
#define CPUSS_SYSARG_REG   REG( 0x40100008 )

#define ROW_SIZE_          ( )
#define ROW_SIZE (ROW_SIZE_)

/*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_IUSER_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)
{
    /* 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;

    /* Number of times the ISR has triggered */
    iStatusInt ++;
}
__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 (128 bytes) to SRAM
    REG( 0x20000004 ) = 0x0000007F;

    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 寄存器内，可以将 CM0 异常表配置为处于 SRAM 内。SRAM 异常表将 SPC 中断的向量地址作为定义在 SRAM 内的 *SpclntHandler()* 函数使用。有关将 CM0 异常表配置在 SRAM 内的详细信息，请参考第 51 页上的中断章节。非阻塞编程系统调用的伪代码是相同的，但函数操作码和各个参数不一样，且 iStatusInt 变量被轮询 1 次，而不是 3 次。这是因为执行非阻塞编程系统调用时，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。术语表中的所有术语以**粗斜体字**显示。

A

累加器	位于 CPU 中并用于存储中间结果的寄存器。如果没有累加器，则需要将各项计算（加、减、移位等等）的结果写入到主存储器内，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	<ol style="list-style-type: none">1. 它是一种逻辑信号，它的激活状态为逻辑 1。2. 它是一种逻辑信号，逻辑 1 状态作为两种状态中较高的电压状态。
低电平有效	<ol style="list-style-type: none">1. 它是一种逻辑信号，它的激活状态为逻辑 0。2. 它是一种逻辑信号，逻辑 1 状态作为两种状态中较低的电压状态：反转逻辑。
地址	确定存储器位置（RAM、ROM 或寄存器）的标签或号码，在该位置上存储了一单位大小的信息。
算法	通过限定步骤解决数学问题的程序，这些步骤通常涉及到一个重复操作。
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	它们基本的可编程运算放大器电路，包括 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范围之间的任何电压，而不仅是逻辑 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）	是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值。数模（DAC）转换器可以用于执行逆向操作。

AND	请参见 <i>布尔代数</i> 。
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之间的接口（例如，用户模块和库）。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。数组存储数目固定、大小相等并且通常属于同一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元素进行访问。大多数高级编程语言都将数组作为内置数据类型使用。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进行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。通过汇编程序可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。汇编语言被视为低层语言，相反，C 被视为高层语言。
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。衰减的单位通常为分贝（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	<ol style="list-style-type: none"> 1. 指的是消息或信息处理系统的频率范围（单位为 Hz）。 2. 放大器（或吸收器）在其频谱区会有大量增益（或损失）；有时，它表示更为具体，例如，半峰全宽。
偏置	<ol style="list-style-type: none"> 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. 施加于器件的电力、机械力、磁场或其他力（场），用于建立运行该器件所需的参考电平。
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，以 10 为基数的数字系统是最常用的数字系统。数字系统的基数表示系统中一个数字里的特定位置可以存在的数字数量。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，一位仅能具有 ‘0’ 或 ‘1’ 值。每 8 个位组成一个字节。由于 PSoC M8CP 是一个 8 位的微控制器，因此 PSoC 器件的原始数据单元大小为一个字节。

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单元，例如振荡器。
2. 按目标功能进行配置的功能单元，例如，数字 PSoC 模块或模拟 PSoC 模块。

布尔代数

在数学和计算机科学中，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）和 NOT（非）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，通过这些定理可以简化布尔等式。这样可减少实现等式所需的逻辑元素数量。

布尔代数的运算符可以使用多种方式来表示。通常可以将它们简写为 AND、OR 和 NOT。在介绍电路时，也可以使用 NAND（NOT AND）、NOR（NOT OR）、XNOR（非 NOT OR）和 XOR（非 OR）等符号。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， $A \cdot B$ ）（在某些情况下，那些运算与代数运算中的加法和乘法类似）。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： $\sim A$ 、 A_{\sim} 、 $!A$ ）。

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，便于使用类似的路由模式来对网络进行路由。
2. 执行通用功能并携带类似数据的一组信号。通常使用向量符号来表示；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

是包含 8 个位的数据存储单位。

C

C	是一种高层编程语言。
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 位的功能。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	将某一位或某寄存器的值强制设置为逻辑 ‘0’。
时钟	是指生成具有固定频率和占空比的周期信号的器件。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 晶体管构建的逻辑门。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。它会影响系统的性能。
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。
crowbar	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）	用于检测数据通讯中的错误的计算方法，通常使用线性反馈移位寄存器执行。相似的计算方法可用于其他用途，如数据压缩。

D

数据总线	计算机使用以将信息从存储器位置传输到中央处理单元（CPU）或反向传输信息的双向信号组。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逐步执行固件操作，设置断点及分析存储器。
死区	两个或多个信号都没有处于活动状态或切换状态的一段时间。
十进制	是以 10 为基数的数字系统。它使用符号 0、1、2、3、4、5、6、7、8 和 9（名为数字）、小数点以及 ‘+’（加号）和 ‘-’（减号）两个符号表示数字。
默认值	指的是系统所假定、使用或在没有用户指令的情况下执行的预定义初始、原始或特定的设置、条件、数值或行动。
器件	除非另有说明，否则该手册所提到的器件均是 PSoC 器件。
裸片	一个非封装集成电路（IC），通常从晶圆切割。
数字	是一个信号或功能，它的幅值使用两个离散值 ‘0’ 或 ‘1’ 的其中一个来描述。
数字模块	可用作计数器、定时器、串行接收器、串行发送器、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。表达式包含了说明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相应量级的模拟信号的器件。 <i>模数（ADC）</i> 转换器可以用来执行逆向操作。
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 <i>高电平时间</i> 与其 <i>低电平时间</i> 的关系，表示为一个百分比。

E

外部复位 (XRES_N) 传入 PSoC 器件的高电平有效信号。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。非易失性表示在断电时，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范围的保护信息。

闪存模块 能够通过单个编程操作进行编程的最小闪存 ROM 空间以及受保护的最小闪存空间。闪存模块容量为 64 个字节。

触发器 它是一个具有两种稳定状态和两个输入端（或两种输入信号类型）的器件，一个输入端与一种状态相对应。电路处于任意一种状态，直到使用相应的信号使它切换为另一种状态。

频率 是指执行周期性功能时每个时间单位内的周期数或发生事件的次数。

G

增益 分别为输出电流、电压或功率与输入电流、电压或功率之间的比率。增益的单位通常使用分贝 (dB)。

门

1. 对于具有一个输出通道和一个或多个输入通道的器件，除开关跃变外，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最少具有两个输入（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接触地面的某些导体之间的导电路径。

H

硬件

它是一个使用于计算机或嵌入式系统所有物理部分的综合术语。硬件不同于其自身所包含的或操作的数据和软件（为硬件提供完成任务所需的指令）。

硬件复位

是由电路触发的复位，例如 **POR**、看门狗复位或外部复位。硬件复位将器件返回到它首次被上电的状态。因此，所有寄存器均被设为 **POR** 值（如本文档中的寄存器表所示）。

十六进制

它是一个十六进制数字系统（通常被简写并称为 **hex**），通常使用数字 **0** 到 **9** 以及字母 **A** 到 **F** 来表示。由于很容易将四个位元映射为十六进制数字，它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每一个字节。二进制、十六进制和十进制表示法对比如下：

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，十进制数字 **79** 的二进制表示法为 **0100 1111b**，它的十六进制表示法则为 **4Fh (0x4F)**。

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

I²C	由 Philips Semiconductors（现为 NXP Semiconductors）生产的两线串行计算机总线。I ² C 是互联集成电路。它用于连接嵌入式系统中的低速外设。原始系统创建于 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的简单的内部总线系统。I ² C 仅使用两个双向引脚（时钟和数据引脚），两者均在 +5 V 的电压条件下运行并采用电阻上拉。在标准模式下，总线速率为 100 Kbps，而在快速模式下，总线速率为 400 Kbps。
闲置状态	当用户消息不被传输时但可以立即使用服务的状态。
阻抗	<ol style="list-style-type: none">1. 是电流上的阻力，该阻力由电路中的电阻、电容或电感产生。2. 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、感抗和容抗组合决定的。
输入	在器件、过程或通道中用于接收数据的一点。
输入 / 输出 (I/O)	是用于将数据引入到系统或从系统中提取数据的器件。
指令	它是在编程语言中（例如 C 或汇编语言）指定一个操作并识别它的操作数（若有）的表达方式。
指令助记符	是表示各汇编语言指令的操作码的一组缩略语（例如，ADD、SUBB、MOV）。
集成电路 (IC)	它是将电阻、电容、二极管和晶体管等组件集成在半导体单一芯片表面的器件。
接口	使两个系统或器件连接或有相互作用的方式。
中断	流程暂停（例如，执行计算机程序），由流程外部事件导致的，且在暂停后可以恢复流程。
中断服务子程序 (ISR)	M8CP 收到硬件中断时常规代码执行转入的代码模块。可以存在多个中断源，每个中断源均有各自的优先级和单个 ISR 代码模块。每个 ISR 代码模块均以 RETI 指令结束，该指令将器件返回到离开常规程序执行的程序点。

J

抖动	<ol style="list-style-type: none">1. 是指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损坏。2. 一个或多个信号特性的突发和意外变化，例如连续脉冲之间的间隔、连续周期的振幅或连续周期的频率或相位。
-----------	---

L

延迟	将一个信号传输到一个给定的电路或网络所需的时间或延迟。
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。为了区分位和字节，在 LSb 中的位 (bit) 上使用了小写字母 “b”。
最低有效字节 (LSB)	它是一个多字节中的字节，用于表示最低有效值（通常为右边的字节）。为了区分位和字节，在 LSB 中的字节 (byte) 使用大写字母 “B”。
线性反馈移位寄存器 (LFSR)	它是一个移位寄存器，其数据输入是寄存器链中两个或两个以上元素经过 XOR 运算后得到的结果。
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 个选择线的两引脚 LUT 对两个输入引脚上执行 16 个逻辑函数中的任意一个，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。
低电平时间	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_{DD} 降低到选定阈值以下时，可检测 V_{DD} 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，解释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	<ol style="list-style-type: none">1. 用于掩饰、隐藏信息、或防止信息从信号派生。掩码通常是与其他信号（例如噪声、静态、拥塞或其他干扰形式）相交互的结果。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。受控制的器件被称为 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。这是为了能够使用最小器件数目实现一个控制器，从而能够实现最大程度的微型化。这会降低控制器的体积和成本。微控制器通常不用于通用计算功能，这些功能由微处理器进行处理。
助记符	用于协助存储器的工具。助记符不仅通过重复，而且还通过创建易记结构和数据列表之间的关联来记住实际情况。一个具有两到四个字符的字符串，表示微处理器指令。
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	用于对载波信号（通常为正弦波信号）上的信息进行编码的一系列技术。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。为了区分位和字节，在 MSb 字母中的位（bit）使用小写字母“b”。
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）。为了区分位和字节，在 MSB 字母中的字节（byte）使用大写字母“B”。
复用器 (mux)	<ol style="list-style-type: none"> 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。若用于线路和 I/O 端口，则复用器具有保存功能。

N

NAND	请参见 布尔代数。
负向沿	从逻辑 1 到逻辑 0 的跃变。它也被称为下降沿。
网络	是器件之间的连接。
半字节	由四个位，就是一个字节的一半构成的组。
噪声	<ol style="list-style-type: none"> 1. 会影响信号，且可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	是指存储器中执行 SSC 指令前用于存储参数的位置。
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	<ol style="list-style-type: none">1. 是由计算机执行的指令逻辑序列。2. 电路中的电子信号流。
挂起中断	一个被触发但未得到处理的中断，这可能是由于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。
引脚分配	引脚号分配：PSoC 器件的逻辑输入和输出与它们在印刷电路板（PCB）封装中相对物理位置之间的关系。引脚分配包括原理图与 PCB 设计（两者均是计算机生成的文件）之间链接的引脚号，也包括引脚名称。
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，但可通过屏蔽位使能或禁用该中断。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以下时强迫 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，用于指出在 CPU 正在处理指令的存储位置。根据特定机器的详细内容，它会存储将被执行的指令地址，或将被执行的下一个指令地址。
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	是指信号特性（例如，相位或频率）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（例如一位或一个字节）的存储器件。
复位	它是一种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，其单位为欧姆（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。将时钟信号提供到计数器的第一段内。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，由于信号的非零上升沿和下降沿，窄脉冲未达到一个有效高电平或低电平。例如，在异步时钟间进行切换，或者是信号经过两个独立的路径输入同一个电路造成竞争状态时，将发生短脉冲。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	指的是将模拟信号转换为一组数字值或反转的过程。
原理图	它是用于详细描述一个系统中各元件（例如电子电路的元件或计算机中逻辑图的元素）的示意图、绘制图或草图。
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	<ol style="list-style-type: none">1. 是指所有包含事件连续发生的流程。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 1，它将成为 0x48。如果十六进制值 0x24 向右边移位 1，则它便成为 0x12。
移位寄存器	按顺序向左或向右转移一个字以便输出串行数据流的存储器件。
符号位	是带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 芯片唯一的标识符。
时滞	是在并行发送中，同时传送的位到达目标的时间差别。
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 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，代码执行将从闪存地址 0x0000 开始。

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，并在闪存中运行。
堆栈	堆栈是按照后入先出（LIFO）的原理运作的数据结构。即：最后输入到堆栈的项目也是第一个被取出的项目。
堆栈指针	堆栈可能位于计算机中模块里的存储器单元内。它的底部被放在单元的一个固定位置上，而变量堆栈指针则被放在当前顶部单元中。
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，剩余的一组开关在 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	<ol style="list-style-type: none"> 1. 是指一个信号，其数据未被确认或做出响应，直到时钟信号的下一个有效边沿到来为止。 2. 是指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。
终端计数	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小值。

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 16 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。在数字电路中，可将晶体管做为快速电子开关使用；另外，合适的晶体管组合可作为逻辑门、**RAM** 型存储器和其他器件使用。在模拟电路，基本上将晶体管作为放大器使用。

三态

指的是一种功能，其输出有三种状态：**0**、**1** 和 **Z**（高阻态）。该功能不会驱动 **Z** 状态下的任何值，在许多方面，可将其视为与其余电路断开，允许另一个输出驱动相同的网络。

U**UART**

UART 或通用异步接收器 - 发送器在数据并行位和串行位之间进行转换。

用户

使用 **PSoC** 器件或阅读本手册的人。

用户模块

负责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，用户模块还针对外设功能提供高级 **API**（应用编程接口）。

用户空间

寄存器映射的组 **0** 空间。在执行常规程序和初始化期间，很可能对该组中的寄存器进行修改。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

如果不属于合适的范围，便不能保证保持相同的值或电平。

V_{SS}

电源网络名称，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

活动模式	
PSoC	98
模拟 I/O	75

B

框图	
编程和调试接口	291
看门狗定时器电路	101
欠压复位	105

C

时钟分布	83
时钟源	
分布	83
时钟系统	
介绍	77
Cortex-M0	
特性	29
指令集	32
寄存器	31

D

开发套件	21
文档	
术语表	313
修订记录	15

E

异常事件	
HardFault	54
NMI	54
PendSV	54
reset	53
SVCall	54
SysTick	54
外部复位	106

F

特性	
----	--

I/O 系统	63
看门狗定时器	101

G

术语表	313
按键和滑条创建的 GPIO 引脚	75

H

休眠模式	99
休眠唤醒复位	106
模拟高阻态驱动模式	66
数字高阻态驱动模式	66
工作原理	
看门狗定时器	102

I

I/O 驱动模式	
模拟高阻态	66
数字高阻态	66
开漏	66
电阻	66
强	66
I/O 系统	
模拟 I/O	75
CapSense	75
特性	63
介绍	63
LCD 驱动能力	75
开漏模式	66
寄存器汇总	76
电阻模式	66
转换速率控制	66
强驱动模式	66
识别复位源	106
内部低速振荡器	80
内部主振荡器	78
内部电压调节器	91
介绍	
时钟生成	77
I/O 系统	63
复位	105
逐次逼近寄存器模数转换器	217

L		W	
LCD 驱动		看门狗复位	105
I/O 系统能力	75	看门狗定时器	103
		禁用	103
O		使能	101
振荡器		特性	102
内部 PSoC	78	工作原理	104
概况, 文档		中断	103
修订记录	15	工作模式	
P			
上电复位	105		
编程和调试			
PSoC	20		
保护故障复位	106		
PSoC			
活动模式	98		
编程和调试	20		
R			
寄存器汇总			
I/O 系统	76		
寄存器			
Cortex-M0	31		
电压调节器			
内部	91		
复位			
识别源	106		
介绍	105		
复位源			
说明	105		
修订记录	15		
S			
SAR ADC			
介绍	217		
睡眠模式	98		
I/O 系统中的转换速率控制	66		
软件复位	106		
停止唤醒复位	106		
支持	21		
SWD 接口			
编程和调试接口	292		
系统调用			
概述	300		
U			
升级	21		