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 4100/4200 系列

PSoC® 4 架构技术参考手册

文档编号 No. 001-86886 版本*A

2016 年 05 月 12 日

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

许可

© 赛普拉斯半导体公司，2013-2016，保留所有的权利。此软件，以及相关的文件或材料属于赛普拉斯半导体公司（Cypress），并可能受到保护，如全球专利保护（美国和外国的），美国版权法和国际条约的规定。除非另有在你和 Cypress 公司之间的单独的许可协议，规定达成一致，你同意将像任何其他受版权保护的材料看待这份材料。

你同意把资料保密，未经书面授权，不会透露或使用 Cypress 的材料。您同意遵守与你和 Cypress 公司之间的任何保密协议。

如果材料涉及到受第三方许可证的项目，您同意遵守该许可。

版权

版权所有©2013 赛普拉斯半导体公司。保留所有权利。

PSoC 和 CapSense 是注册商标，PSoC Designer 和 PSoC Creator 是赛普拉斯半导体公司的商标。所有其他商标或注册标本文提及的是其各自所有者的财产。

从赛普拉斯公司或它联营公司之一购买 I^2C 组件传达了一个经授权许可，即飞利浦提供的 I^2C 的专利权利，以将这些组件使用在一个 I^2C 系统中，该系统符合由飞利浦定义的 I^2C 的标准的产品规格。于 2006 年 10 月 1 起，飞利浦半导体有一个新的名字－恩智浦半导体（NXP Semiconductors）。

本文档中的信息如有更改，恕不另行通知，不应该被解释为一个承诺书。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，不得将本文档的任何部分以任何形式或以任何方式复制或转载。美国制造。

声明

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

闪存代码保护

赛普拉斯的产品符合规格包含在其特定的赛普拉斯数据表。赛普拉斯认为，不管他们是如何使用的，PSoC 产品是目前市场上的同类最安全的产品之一。但是，还是有可能的方法，可以破坏代码保护功能。据我们所知，这些方法中的任何一种，将是不诚实的，或者是非法的。无论是赛普拉斯，任何其他半导体厂商都不能可以保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”。

赛普拉斯是愿意与客户一起，关心代码的完整性。代码保护功能是处于不断不断变化的。赛普拉斯致力于不断改进我们的产品的代码保护功能。

目录概述

A 部分：概述 15

1 介绍 16
2 入门 21
3 文档结构 22

B 部分：中央处理器（CPU）子系统 25

4 CORTEX™-M0 处理器 26
5 中断 31

C 部分：存储器系统 41

6 存储器 42

D 部分：系统资源 44

7 输入/输出系统 45
8 时钟系统 54
9 电源和电源监测 60
10 运行模式 66
11 电源模式 67
12 看门狗定时器 73
13 复位 76
14 器件安全 79

E 部分：数字系统 80

15 串行通信模块 (SCB) 81
16 通用数字模块 117
17 （TIMER/COUNTER/PWM) 模块 154

F 部分：模拟系统 174

18 精确基准源 175
19 SAR ADC 模块 178
20 低功耗比较器 208
21 CTBM 212
22 LCD 段直接驱动 217
23 CAPSENSE 226
24 温度传感器 233

G 部分：编程和调试 237

25 编程和调试接口 238
26 非易失性存储器编程 244

目录

A 部分：概述 15

1 介绍 16

1.1 总体架构 17
1.2 特性 19
1.3 CPU 系统 19
1.3.1 中央处理器 19
1.3.2 中断控制器 19
1.4 内存 19
1.4.1 Flash 19
1.4.2 SRAM 19
1.5 系统资源 19
1.5.1 时钟系统 19
1.5.2 电源系统 19
1.5.3 GPIO 19
1.6 可编程数字资源 20
1.7 固定功能数字资源 20
1.7.1 Timer/Counter/PWM 模块 20
1.7.2 串行通信模块（SCB） 20
1.8 模拟系统 20
1.8.1 SAR ADC 20
1.8.2 CTBm 模块 20
1.8.3 低功耗比较器 20
1.9 特殊功能外设 20
1.9.1 LCD 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

3 文档结构 22

3.1 主要内容 22
3.2 文档约定 22
3.2.1 寄存器约定 22
3.2.2 数字命名 22
3.2.3 单位 23
3.2.4 缩略语 23

B 部分：中央处理器（CPU）子系统 25

4 CORTEX™-M0 处理器 26

4.1 特性 26
4.2 处理器子系统框图 27
4.3 CORTEX-M0 处理器的使用 27
4.3.1 Cortex-M0 处理器内核寄存器 27
4.3.2 操作模式 29
4.3.3 指令集 29
4.3.3.1 地址对齐 30
4.3.3.2 存储器格式 30
4.3.4 系统节拍定时器 30
4.3.5 调试 30

5 中断 31

5.1 特性 31
5.2 中断的实现 31
5.3 中断和异常的工作原理 32
5.3.1 PSoC 4 中的中断/异常处理 32
5.3.2 电平中断和脉冲中断 32
5.3.3 异常向量表 32
5.4 异常源 33
5.4.1 复位异常 33
5.4.2 不可屏蔽中断（NMI） 34
5.4.3 HardFault 异常 34
5.4.4 管理程序调用（SVCall）异常 34
5.4.5 PendSV 异常 34
5.4.6 SysTick 异常 34
5.5 中断源 35
5.6 使能/禁止异常 36
5.7 异常状态 37
5.7.1 挂起异常 37
5.7.2 异常优先级 37
5.8 堆栈使用 38
5.9 中断和低功耗模式 38
5.10 异常 - 初始化和配置 39
5.11 中断和异常寄存器 39
5.12 中断和异常相关文档 40

C 部分：存储器系统 41

6 存储器 42

6.1 特性 42
6.2 工作原理 42

D 部分：系统资源 44

7 输入/输出系统 45

7.1 特性 45

7.2 GPIO 模块框图 46
7.3 I/O 驱动模式 47
7.3.1 高阻模拟模式 48
7.3.2 高阻数字模式 48
7.3.3 电阻上拉或电阻下拉模式 48
7.3.4 漏极开路驱动高和漏极开路驱动低 48
7.3.5 增强驱动模式 48
7.3.6 电阻上拉和下拉模式 48
7.4 摆率控制 49
7.5 CMOS 和 LVTTL 电平控制 49
7.6 高速输入输出矩阵 49
7.7 模拟管脚 50
7.8 段式 LCD 驱动 50
7.9 CapSense 50
7.10 重新配置 GPIO 51
7.11 上电启动时 GPIO 状态 51
7.12 睡眠模式下的 GPIO 状态 51
7.13 低功耗模式下的 GPIO 状态 51
7.14 端口中断控制器模块 51
7.14.1 特性 51
7.14.2 端口中断控制器框图 51
7.14.3 功能和配置 52
7.15 数字输入和输出同步 52
7.16 活动和深度睡眠模式下专用管脚 52
7.17 端口 4 的使用限制 53
7.18 寄存器列表 53

8 时钟系统 54

8.1 模块框图 54
8.2 时钟源 55
8.2.1 内部主振荡器（IMO） 55
8.2.2 内部低速振荡器（ILO） 55
8.2.3 外部时钟（EXTCLK） 55
8.3 时钟分布 55
8.3.1 HFCLK 输入时钟源的选择 56
8.3.2 SYSCLK 分频器的配置 57
8.3.3 外设时钟分频器的设置 57
8.3.4 外设时钟分配 58
8.4 低功耗电源模式下的操作 59

9 电源和电源监测 60

9.1 系统框图 60
9.2 电源连接方式 61
9.2.1 外部电压较高时（1.8V-5.5V） 61
9.2.2 外部电压较低时（1.71V-1.89V） 62
9.3 使用 64
9.3.1 内部调压器 64
9.3.1.1 活动状态调压器 64
9.3.1.2 低噪声调压器 64
9.3.1.3 深度睡眠寄存器 64

9.3.1.4 休眠寄存器 64
9.3.2 电压检测 64
9.3.2.1 上电重启（POR） 64
9.3.2.2 欠压检测（BOD） 64
9.3.2.3 低压检测 64
9.4 寄存器列表 65
10 运行模式 66
10.1 引导模式 66
10.2 用户模式 66
10.3 特权模式 66
10.4 调试模式 66
11 电源模式 67
11.1 活动模式 69
11.2 睡眠模式 69
11.3 深度睡眠模式 69
11.4 休眠模式 70
11.5 停止模式 70
11.6 低功耗模式的使用 71
11.7 寄存器列表 72
12 看门狗定时器 73
12.1 特性 73
12.2 模块框图 73
12.3 工作原理 74
12.3.1 WDT 的使能和禁止 74
12.3.2 WDT 的操作模式 74
12.3.3 WDT 的中断和低功耗模式 74
12.3.4 WDT 的复位 74
12.4 寄存器列表 75
13 复位 76
13.1 复位源 76
13.1.1 上电复位 76
13.1.2 掉电检测复位 76
13.1.3 看门狗复位 77
13.1.4 软件复位 77
13.1.5 外部复位 77
13.1.6 特权保护复位： 77
13.1.7 休眠模式唤醒复位 77
13.1.8 停止模式唤醒复位 77
13.2 复位源识别 77
14 器件安全 79
14.1 特性 79
14.2 工作原理 79

E 部分：数字系统 80

15 串行通信模块 (SCB) 81

15.1 特性 81
15.2 串行外设接口（SPI）协议 81
15.2.1 特性 81
15.2.2 概述 82
15.2.3 SPI 协议介绍 82
15.2.3.1 摩托罗拉 SPI 协议 82
15.2.3.2 德州仪器 SPI 协议 84
15.2.4 国家半导体 SPI 协议 86
15.2.5 EZSPI (Easy SPI) — SPI 的 EZ 模式 87
15.2.5.1 写地址 88
15.2.5.2 写数据 88
15.2.5.3 读数据 88
15.2.5.4 配置 90
15.2.6 SPI 相关寄存器 90
15.2.7 SPI 的中断 90
15.2.8 SPI 的初始化 91
15.2.9 SPI 的时钟模式 91
15.2.9.1 非 EZSPI 的时钟模式 92
15.2.9.2 EZSPI 的时钟模式 93
15.3 通用异步收发（UART）协议 94
15.3.1 特性 94
15.3.2 概述 94
15.3.3 UART 协议介绍 95
15.3.3.1 标准 UART 协议 95
15.3.3.2 SmartCard (ISO7816) 协议 98
15.3.3.3 IrDA 协议 99
15.3.4 UART 相关寄存器 100
15.3.5 UART 的中断 100
15.3.6 UART 的初始化 100
15.4 I2C 协议 102
15.4.1 特性 102
15.4.2 概述 102
15.4.2.1 时钟延展 103
15.4.2.2 总线冲突检测与仲裁 103
15.4.3 I2C 协议介绍 103
15.4.3.1 写数据 103
15.4.3.2 读数据 104
15.4.4 EZI2C (Easy I2C) — I2C 的 EZ 模式 104
15.4.5 I2C 相关寄存器 105
15.4.6 I2C 的中断 106
15.4.7 I2C 的初始化 106
15.4.8 I2C 的时钟模式 106
15.4.8.1 非 EZI2C 的时钟模式 106
15.4.8.2 EZI2C 的时钟模式 107
15.4.9 唤醒操作 108
15.4.10 主设备模式数据传输示例 109
15.4.10.1 主设备发送数据 109
15.4.10.2 主设备接收数据 110

15.4.11 从设备模式数据传输示例 111
15.4.11.1 从设备发送数据 111
15.4.11.2 从设备接收数据 112
15.4.12 EZ 模式的数据传输示例 113
15.4.12.1 EZ 从设备发送数据 113
15.4.12.2 EZ 从设备接收数据 114
15.4.13 多主设备总线上的数据传输示例 115
15.4.13.1 仅作为主设备 115
15.4.13.2 主从设备 116

16 通用数字模块 117

16.1 特征 117
16.2 工作原理 118
16.2.1 PLD 118
16.2.1.1 PLD 的宏单元 118
16.2.1.2 PLD 的进位级联 119
16.2.1.3 PLD 的配置 119
16.2.2 数据通道处理器（DP：DataPath） 119
16.2.2.1 概览 120
16.2.2.2 数据通道处理器中的 FIFO 122
16.2.2.3 FIFO 的状态信号 127
16.2.2.4 DP 中的 ALU 127
16.2.2.5 DP 的输入多路器（MUX） 129
16.2.2.6 CRC/PRS 130
16.2.2.7 DP 的输出多路器 132
16.2.2.8 DP 的并行输入和并行输出 133
16.2.2.9 DP 的级联 134
16.2.2.10 动态配置寄存器 134
16.2.3 状态与控制模块 135
16.2.3.1 状态输入模式 136
16.2.3.2 控制输出模式 138
16.2.3.3 并行输入和并行输出模式 140
16.2.3.4 计数器模式 140
16.2.3.5 同步模式 141
16.2.3.6 状态与控制模块的时钟 141
16.2.3.7 辅助控制寄存器 141
16.2.3.8 状态与控制寄存器的总结 142
16.2.4 复位与时钟模块 142
16.2.4.1 时钟的控制 143
16.2.4.2 复位控制 144
16.2.4.3 UDB 的上电复位初始化 148
16.2.5 UDB 中寄存器的寻址 148
16.2.6 系统总线的访问 148
16.2.6.1 并发的系统总线访问 148
16.2.6.2 系统总线对累加器的访问 149
16.3 端口适配（PA：PORT ADAPTOR）模块 149
16.3.1 PA 模块的时钟多路器 149
16.3.2 PA 的复位多路器 150
16.3.3 PA 的数据输入单元 151
16.3.4 PA 的数据输出单元 151
16.3.5 PA 的输出使能单元 151
16.3.6 PA 的时钟输入多路器 152

17 （TIMER/COUNTER/PWM) 模块 154

17.1 特性 154
17.2 模块概述 155
17.2.1 TCPWM 模块中计数器的使能与禁止 155
17.2.2 时钟 155
17.2.3 事件的触发 156
17.2.4 输出信号 157
17.2.4.1 内部事件信号 157
17.2.4.2 中断 157
17.2.4.3 输出信号 157
17.2.5 电源模式 158
17.3 工作模式 158
17.3.1 定时器模式 159
17.3.1.1 模块框图 159
17.3.1.2 工作原理 159
17.3.1.3 定时器模式的配置流程 161
17.3.2 捕获模式 161
17.3.2.1 模块框图 161
17.3.2.2 工作原理 161
17.3.2.3 捕获模式的配置流程 162
17.3.3 正交编码模式 163
17.3.3.1 模块框图 163
17.3.3.2 工作原理 163
17.3.3.3 正交模式配置步骤 165
17.3.4 脉冲宽度调制（PWM）模式 166
17.3.4.1 模块框图 166
17.3.4.2 工作原理 166
17.3.4.3 其他配置 168
17.3.4.4 终止(Kill) 168
17.3.4.5 PWM 模式的计数器配置 169
17.3.5 带死区模式的 PWM 169
17.3.5.1 模块框图 169
17.3.5.2 工作原理 170
17.3.5.3 配置带死区模式的 PWM 170
17.3.6 PWM-伪随机模式 171
17.3.6.1 模块框图 171
17.3.6.2 工作原理 171
17.3.6.3 配置伪随机 PWM 模式 172
17.4 TCPWM 寄存器 173

F 部分：模拟系统 174

18 精确基准源 175

18.1 模块框图 175
18.2 工作原理 176
18.2.1 精确带隙电路 （Bandgap） 176
18.2.2 矫正缓冲器 176
18.2.3 低功耗缓冲 176
18.2.4 电流单元 177
18.2.5 V-CTAT 模块 177
18.2.6 IMO 参考源电路 177
18.3 配置 177

19 SAR ADC 模块 178

19.1 特性 178
19.2 SAR ADC 模块框图 179
19.3 SAR ADC 模块的使用 180
19.3.1 SARADC 核 180
19.3.1.1 单端模式和差分模式 180
19.3.1.2 输入范围 180
19.3.1.3 转换结果数据格式 180
19.3.1.4 负端输入选择 181
19.3.1.5 分辨率 182
19.3.1.6 采样时间 182
19.3.1.7 SAR ADC 时钟 182
19.3.1.8 SAR ADC 工作时序 182
19.3.2 SARMUX 183
19.3.2.1 模拟路由 183
19.3.2.2 模拟内部互连 184
19.3.2.3 外部管脚输入 184
19.3.2.4 来自 AMUXBU_A/AMUXBU_B 的输入 185
19.3.2.5 来自 CTBm 输出的输入 186
19.3.2.6 温度传感器输入 187
19.3.3 SARREF 188
19.3.3.1 参考电压选项 189
19.3.3.2 旁路电容 189
19.3.3.3 输入电压范围和参考电压 189
19.3.4 SARSEQ 189
19.3.4.1 求平均 190
19.3.4.2 阈值检测 191
19.3.4.3 双缓冲保存 191
19.3.4.4 插入通道 191
19.3.5 中断 193
19.3.5.1 扫描结束中断（EOS_INTR） 193
19.3.5.2 溢出中断 193
19.3.5.3 冲突中断 193
19.3.5.4 插入通道转换结束中断（INJ_EOC_INTR） 194
19.3.5.5 阈值检测中断 194
19.3.5.6 饱和检测中断 194
19.3.5.7 中断源识别 194
19.3.6 触发 194
19.3.6.1 DSI 触发配置 195
19.3.7 SAR ADC 状态 195
19.3.8 低功耗模式 196
19.3.9 系统操作 196
19.3.10 寄存器模式 197
19.3.10.1 SARMUX 模拟路由配置 197
19.3.10.2 SARSEQ 全局配置 198
19.3.10.3 通道配置 199
19.3.10.4 通道使能 199
19.3.10.5 中断屏蔽 199
19.3.10.6 触发 200
19.3.10.7 数据读取 200
19.3.10.8 插入通道转换（可选） 200

19.3.11 DSI 模式 ... 200
19.3.11.1 SARMUX 模拟路由配置 ... 202
19.3.11.2 SARSEQ 全局配置 ... 202
19.3.11.3 通道配置 ... 202
19.3.11.4 中断 ... 203
19.3.11.5 触发 ... 203
19.3.11.6 数据读取 ... 203
19.3.11.7 DSI 输出使能 ... 203
19.3.12 模拟路由配置示例 ... 204
19.3.13 温度传感器配置 ... 206
19.4 寄存器列表 ... 207

20 低功耗比较器 ... 208

20.1 特性 ... 208
20.2 低功耗比较器框图 ... 209
20.3 低功耗比较器的使用 ... 209
20.3.1 输入配置 ... 209
20.3.2 电源模式和速度配置 ... 209
20.3.3 输出和中断配置 ... 210
20.3.4 迟滞 ... 210
20.3.5 休眠唤醒 ... 210
20.3.6 比较器时钟 ... 210
20.3.7 失调电压校正 ... 210
20.4 寄存器列表 ... 211

21 CTBM ... 212

21.1 特性 ... 212
21.2 模块框图 ... 213
21.3 工作原理 ... 213
21.3.1 电源模式配置 ... 213
21.3.2 驱动能力配置 ... 214
21.3.3 开关配置 ... 214
21.3.3.1 输入开关控制 ... 214
21.3.3.2 输出开关控制 ... 215
21.3.4 比较器工作模式 ... 216
21.3.4.1 比较器的配置 ... 216
21.3.4.2 比较器的中断 ... 216
21.4 寄存器列表 ... 216

22 LCD 段直接驱动 ... 217

22.1 特性 ... 217
22.2 LCD 段驱动概述 ... 217
22.2.1 驱动模式 ... 218
22.2.1.1 PWM 驱动 ... 218
22.2.1.2 数字相关 ... 221
22.2.2 推荐的驱动模式 ... 223
22.2.3 数字对比度控制 ... 223
22.3 LCD 模块介绍 ... 224
22.3.1 工作原理 ... 224
22.3.2 高速和低速驱动信号发生器 ... 224

22.3.3 多路复用器和 LCD 引脚逻辑 225
22.3.4 显示数据寄存器 225
22.4 寄存器表 225

23 CAPSENSE 226

23.1 特性 226
23.2 模块框图 226
23.3 工作原理 227
23.3.1 CSD 感应原理 227
23.3.1.1 GPIO 单元的电容-电流转换器 228
23.3.1.2 开关时钟发生器 230
23.3.1.3 电流-数字转换器 230
23.3.1.4 模拟多路器 231
23.3.2 屏蔽电极 231
23.3.3 C_{MOD} 的预充电 231

24 温度传感器 233

24.1 特性 233
24.2 工作原理 233
24.3 温度传感器配置 234
24.4 算法 235

G 部分：编程和调试 237

25 编程和调试接口 238

25.1 特性 238
25.2 功能介绍 238
25.3 串行线调试（SWD）接口 239
25.4 CORTEX M0 调试和访问端口（DAP） 241
25.4.1 调试端口寄存器 241
25.4.2 访问接口寄存器 241
25.5 PSOC4 的编程 241
25.5.3 端口获取 241
25.5.3.1 主/次 SWD 端口 241
25.5.3.2 端口获取过程 242
25.5.4 进入 SWD 编程模式 242
25.5.5 执行 SWD 编程模式 242
25.6 PSOC4 SWD 调试接口 242
25.6.1 调试控制和配置寄存器 242
25.6.2 断点单元 243
25.6.3 数据观察点和跟踪（DWT） 243
25.6.4 调试 PSoC4 243

26 非易失性存储器编程 244

26.1 特性 244
26.2 功能介绍 244
26.3 SYSTEM CALL 的实现 244
26.4 阻塞和非阻塞 SYSTEM CALL 245
26.4.1 调用 System Call 的步骤 245

26.5 SYSTEM CALL 函数 246
26.5.1 芯片信息 246
26.5.2 加载闪存字节 247
26.5.3 写行 248
26.5.4 编程行 248
26.5.5 擦除所有 249
26.5.6 校验 250
26.5.7 写保护 250
26.5.8 非阻塞写行 251
26.5.9 非阻塞编程行 251
26.5.10 重新执行非阻塞 252
26.6 SYSTEM CALL 返回状态值 253
26.7 非阻塞 SYSTEM CALL 伪代码 253

A 部分：概述

这部分包括以下章节：

- 介绍　第 16 页
- 入门　第 21 页
- 文档结构　第 22 页

文档修改历史

修订版本	提交日期	变更人	变更说明
**	2013 年 4 月 19 日	JCHE	初次发布，翻译自 001-85634*A 版本
*A	2016 年 5 月 12 日	NIDH	Updated to new template. Completing Sunset Review.

1 介绍

PSoC® 4 是基于 ARM Cortex-M0 CPU（处理器）的可编程嵌入式系统控制器家族，为嵌入式应用提供了强大的可编程平台。它集合了可编程模拟资源、可编程内部互联、用户可编程数字逻辑、通用的固定功能外设计以及高性能的 ARM Cortex-M0 CPU 子系统。

PSoC 4100/4200 系列是 PSoC® 4 家族第一个入门级的成员，并将与 PSoC® 4 家族的其他成员完全兼容。

PSoC® 4 系列包括以下特性：

- 高性能 Cortex-M0 CPU 内核
- 固定功能以及可配置的数字模块
- 高度可编程的数字逻辑
- 高性能模拟系统
- 灵活可编程的内部互连

本文档详细描述了 PSoC 4100/4200 设备的每个功能模块来帮助设计人员创建系统级设计。

1.1 总体架构

图 1-1 显示了 PSoC 4100 的系统框图，图 1-2 显示了 PSoC 4200 的系统框图。

图 1-1 PSoC 4100 系统框图

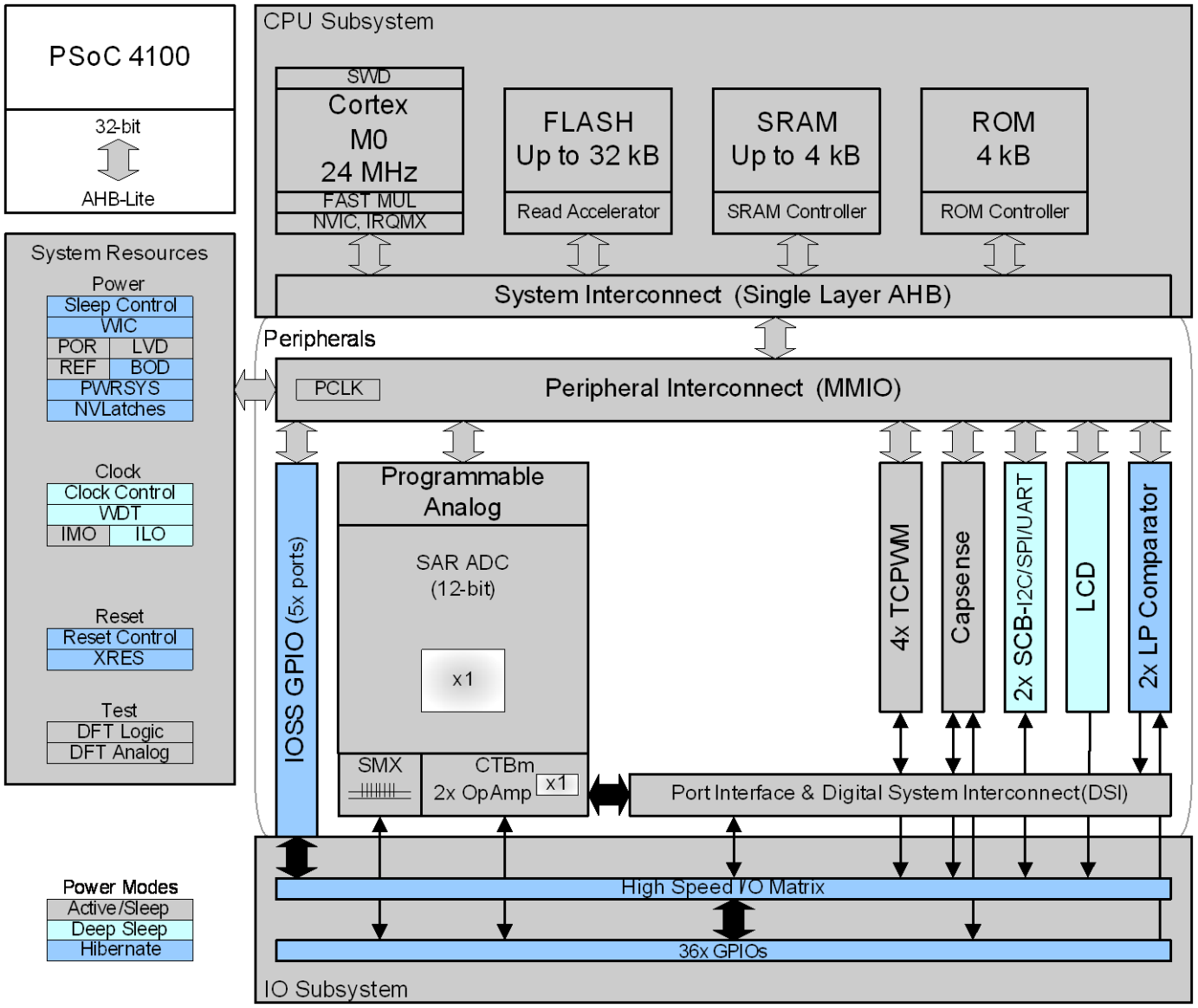
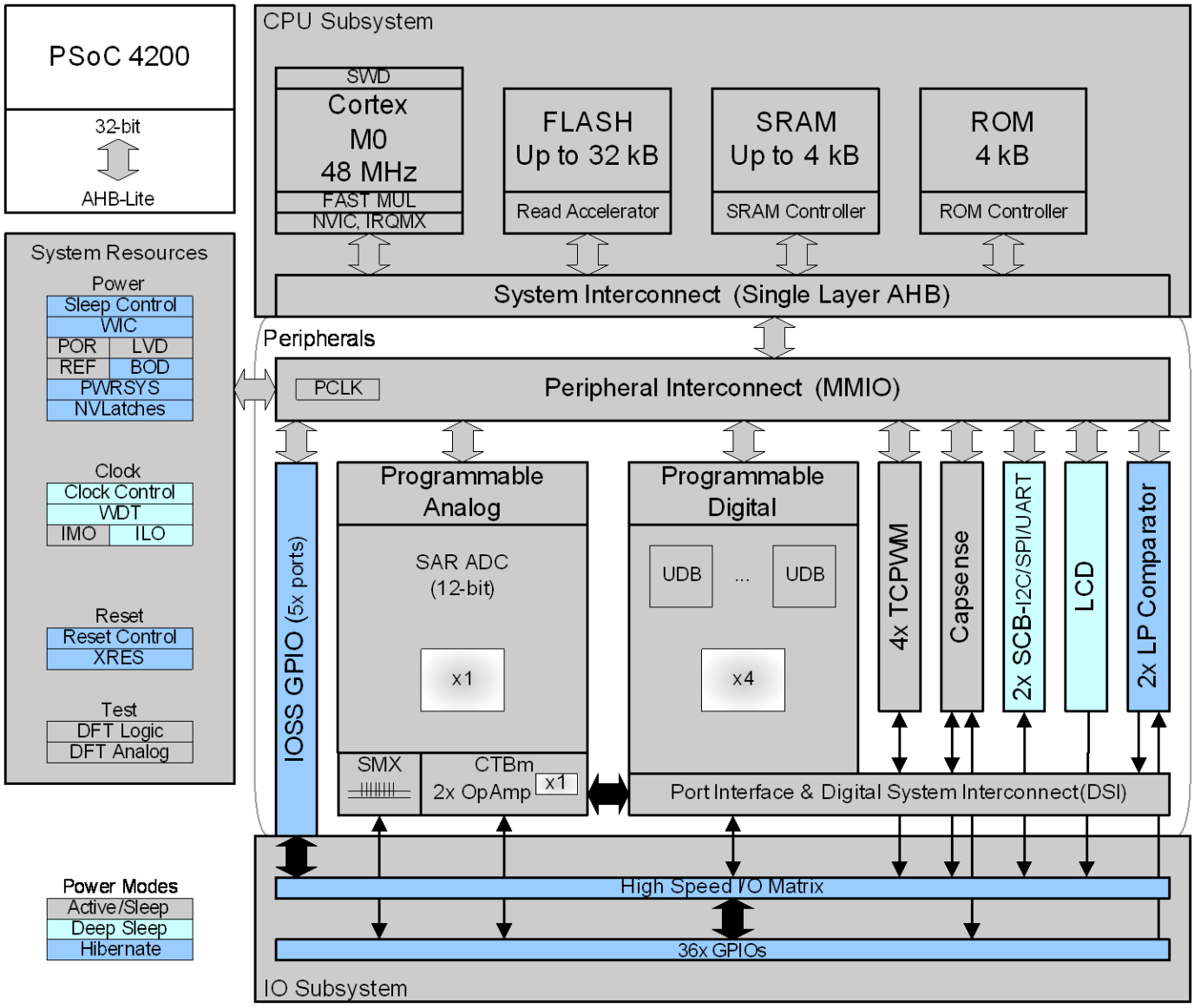


图 1-2 PSoC 4200 系统框图



1.2 特性

PSoC® 4100/4200 包含以下主要特性：

- 高达 48Mhz，43 DMIPS 的 32 位 Cortex-M0 CPU，支持单周期乘法
- 多达 32 KB Flash 及 4KB SRAM 内存
- 四个独立的可支持中央对齐的 PWM，支持互补的可编程死区及同步 ADC 操作
- 一个支持零开销通道切换功能的 12 位 1 Msps ADC
- 两个支持比较器模式及 SAR ADC 输入缓冲功能的运算放大器
- 两个低功耗比较器
- 两个可工作为 SPI/UART/I2C 串行通信接口的串行通信模块（SCB）
- 四个可编程数字逻辑模块(UDB)
- CapSense® 及 LCD 驱动
- 低功耗运行模式: Sleep、Deep Sleep、Hibernate 和 Stop
- SWD 编程及调试单元
- 全面支持 PSoC Creator IDE 工具

1.3 CPU 系统

1.3.1 中央处理器

PSoC 4100/4200 设备的核心是一个可运行至 48 MHz 的 32 位 Cortex-M0 CPU 内核，并为低功耗操作及广泛的时钟选通进行了优化。它通常使用 16 位指令并可以执行 Thumb-2 指令集的部分子集，因此完全兼容代码移植到类似于 Cortex M3 或 M4 的更高性能处理器。

PSoC 4100/4200 的 CPU 同时包含一个输出 32 位结果的单周期硬件乘法器。

1.3.2 中断控制器

PSoC 4100/4200 的 CPU 子系统包含一个支持 32 个中断输入的嵌套中断向量控制器 (NVIC) 和一个可在 Deep Sleep 模式中唤醒处理器的中断唤醒控制器(WIC)。PSoC 4100/4200 的 Cortex-M0 CPU 还支持一个可供用户使用的非屏蔽的中断输入。

1.4 内存

PSoC 4100/4200 内存子系统包括了 Flash 和 SRAM 以及包含引导及配置程序的管理型 ROM。

1.4.1 Flash

PSoC 4100/4200 的 Flash 模块包含一个与 CPU 紧密连接的 Flash 加速器来降低平均访问时间。PSoC 4100/4200 的 Flash 可以实现在 48 MHz 时访问时间为一个等待时序（WS）而在 24 MHz 时为零等待时序。平均下来，Flash 加速器可以实现 85%的单周期 SRAM 访问性能。部分 Flash 可以被用来模拟 EEPROM 操作。

1.4.2 SRAM

PSoC 4100/4200 中的 SRAM 内存可以在 Hibernate 模式中得到保持。

1.5 系统资源

1.5.1 时钟系统

PSoC 4100/4200 的时钟系统包含内部主振荡器（IMO）、内部低速振荡器（ILO）以及一个外部时钟接口。

精度为±2% 的 IMO 是 PSoC 4100/4200 内部时钟的主要时钟源，IMO 的默认频率为 24 MHz 并可以在 3 到 48 MHz 的范围内以 1 MHz 的级别调整。IMO 产生多个衍生时钟以供不同应用使用。

ILO 是一个非常低功耗、但是精度较低的振荡器，主要用于产生 Deep Sleep 模式下外设工作的时钟。它的时钟频率为 32 KHz，精度为 ±60% 。

也可以通过引入一个范围从 0 到 48 MHz 的外部时钟源来代替 IMO 产生用于 PSoC 4100/4200 功能模块的衍生时钟。

1.5.2 电源系统

PSoC 4100/4200 使用范围为 1.71 到 5.5 V 的单独外部供电。

除了默认的 Active 模式，PSoC 4100/4200 支持几种低功耗模式：Sleep、Deep Sleep、Hibernate 和 Stop 模式。Active 模式下， CPU 处于运行状态，所有功能模块供电。Sleep 模式下，CPU 主时钟停止，CPU 停止运行。在 Deep Sleep 模式下，CPU、SRAM 以及高速逻辑处于保持状态，主系统时钟关闭，低速时钟开启，低速外设运行。Hibernate 模式下，低速时钟也被关闭，低速外设停止运行。

PSoC 4100/4200 系统中有多个内部电压调节器用于支持不同功耗模式下的供电机制。

1.5.3 GPIO

PSoC 4100/4200 中的每个 GPIO 含有以下特征:

- 八个不同的驱动模式
- 输入输出关断的独立控制
- 可以来锁存先前状态的保存模式

- 可选择的摆率
- 中断产生：边沿触发
- Capsense 及 LCD 驱动支持

GPIO 管脚被组织为 8 位宽度的多个端口。使用一个高速的 I/O 矩阵可以在可能连接到同一个 I/O 管脚的多个信号间多路复用。固定功能外设的管脚位置也是固定的。

1.6 可编程数字资源

PSoC 4200 包含四个 UDB （通用数字模块）模块， 每个 UDB 包含结构化的数据路径逻辑、未约束的 PLD 逻辑以及灵活的内部互连。四个 UDB 通过称为数字系统互连（DSI）的可切换连线结构组成 UDB 阵列。通过 DSI 可以把信号从外设及端口连接到 UDB 模块中。

PSoC 4200 中的 UDB 阵列可以实现定制的逻辑、额外的 Timer/PWM 和通信接口，例如 I2C、SPI、I2S 及 UART 等。注意，PSoC 4100 没有 UDB。

1.7 固定功能数字资源

1.7.1 Timer/Counter/PWM 模块

Timer/Counter/PWM 模块包括四个16位的周期长度用户可编程的计数器，这些计数器之间可以进行功能同步。每个模块包含一个捕获寄存器、一个周期寄存器以及一些比较寄存器。每个模块都支持互补的可编程的死区，还支持一个关断输入信号来强迫输出信号进入预先设定的状态。Timer/Counter/PWM 模块还支持中央对齐PWM、时钟预分频、伪随机PWM 以及正交解码等功能。

1.7.2 串行通信模块（SCB）

PSoC 4100/4200 包含两个串行通信模块 SCB，每一个可配置的 SCB 可以实现一个串行通信接口，如 I2C、UART 或 SPI。

每个 SCB 支持以下功能:

- 标准 I2C （多）主和从接口
- 标准 SPI 主和从接口，支持 Motorola、TI 和 National （MicroWire） 模式
- 标准 UART 收发器，支持 SmartCard 读卡器（ISO7816）、IrDA 及 LIN 协议
- 带有 32 字节缓存的 EZ 模式 SPI 和 I2C

1.8 模拟系统

1.8.1 SAR ADC

PSoC 4200 包含一个可配置的支持 1 MSps 采样率的 12 位 SAR ADC，PSoC 4100 有一个支持 806Ksps 采样率的 12 位 SAR ADC，可适用于非常广泛的模拟应用。 此 ADC 的主要性能指标包括：增益误差±0.1%，积分非线性（INL）小于 1 LSB，差分非线性（DNL）小于 1 LSB， 信噪比（SNR）大于 68 dB。

此 ADC 提供了 3 种内部电压参考的选择（VDD， VDD/2 和 VREF）以及一个通过 GPIO 管脚输入的外部电压参考。ADC 通过一个 8 通道的序列器（Sequencer）连接到一些固定的管脚，每个通道的转换结果可以通过此序列器进行缓存从而减少了对 CPU 中断服务的请求。

1.8.2 CTBm 模块

CTBm模块包括两个高度可编程的高性能运算放大器以及一个开关阵列。这两个运算放大器也可以工作在比较器模式。不使用外部元器件，此CTBm模块可以实现开环运算放大器、缓冲器及比较器功能。使用一些外部元器件，还可以实现PGA、电压缓冲器、滤波器以及阻抗变换器（Trans-Impedance Amplifier）等功能。

1.8.3 低功耗比较器

PSoC 4100/4200 带有两个可工作于 Deep Sleep 和 Hibernate 模式下的低功耗比较器，这使得低功耗模式下即模拟系统时钟被关断的同时可以保持监控外部电压水平的能力。

两个输入可以来自于固定管脚，或通过 AMUXBUS 来自于内部信号。

1.9 特殊功能外设

1.9.1 LCD 驱动

PSoC 4100/4200 包含一个可以驱动多达 4 个 Common 和 32 个 Segment 的 LCD 控制器。此控制器可使用完全数字化的方式（数字相关或 PWM）来直接驱动 LCD 的段（Segment）而不需要产生内部 LCD 电压。

1.9.2 CapSense

PSoC 4100/4200 设备包含一个称为 CapSense® 的功能模块从而使得用户可以利用手指的电容特性来优雅的操作高级的按键、滑条和滚轮。PSoC 4100/4200 通过 CSD (CapSense Sigma-Delta) 模块在所有的 GPIO 上都实现了对 CapSense 功能的支持，并提供了超强的防水能力。当 CapSense 功能没有使用的时候， CSD 模块中的两个 IDAC 也可以作为一般用途使用。

1.10 编程和调试

PSoC 4100/4200 设备通过片上的 SWD（串行线调试器）接口支持编程和调试功能。PSoC Creator IDE 软件提供了对 PSoC 4100/4200 设备的完整的编程和调试支持。此 SWD 接口与业界标准的第三方工具完全兼容。

2 入门

2.1 支持

可在线访问http://www.cypress.com 来获得PSoC® 4产品的免费支持，包括培训研讨会、技术论坛、应用手册、PSoC 咨询、TightLink 技术支持邮件/知识库和应用支持专家。

可访问http://www.cypress.com/support/ 或拨打电话 1-800-541-4736获得应用方面的支持。

2.2 产品升级

Cypress 会免费提供PSoC Creator的定期升级及版本更新。可通过分销商提供的光盘或在 http://www.cypress.com 的软件目录下直接下载来获得升级版本，文档目录下同时提供系统文档的重要更新。

2.3 开发套件

可以通过 Digi-Key、 Avnet、Arrow 及 Future 获得开发套件。Cypress 在线商店包含所有成功开发 PSoC 项目所需的开发套件、 ***C*** 编译器和附件。请访问 Cypress 在线商店网址 http://www.cypress.com/shop/。在产品类别下点击 *PSoC (Programmable System-on-Chip)* 可以浏览现有的项目列表。

3 文档结构

本文档主要包括以下几部分内容：

- 篇章 A：概述　第 15 页
- 篇章 B：中央处理器系统　第 25 页
- 篇章 C：存储器系统　第 41 页
- 篇章 D：系统资源　第 44 页
- 篇章 E：数字系统　第 80 页
- 篇章 F：模拟系统　第 174 页
- 篇章 G：编程和调试系统　第 237 页

3.1　主要内容

为了阅读方便，本文档的内容主要以篇章和章节的形式组成。

- 篇章 — 介绍的顶层架构，如何开始使用和约定概念，并概述相应的领域，告知读者产品的架构。
- 章节 — 介绍具体到每个主题的章节。这些集成电路的某些方面的详细的实现和使用的信息。

3.2　文档约定

除了标题，本文用到了四种与正文不同的文字格式：

- 第一种是斜体字 *italics*，用来表示参考的文档标题或文件名字。
- 第二种是粗斜体字，***bold italics***，用来表示术语表中提到的术语。
- 第三种是 Times New Roman，用来表示等式。
- 第四种是 Courier New，用来表示示例代码。

3.2.1　寄存器约定

寄存器约定详见：PSoC® 4 Registers Technical Reference Manual

3.2.2　数字命名

表示十六进制数字的所有字母大写，附加小写字母“h”（例如，'14h'或‘3Ah’）和十六进制数，也可以用一个'0 x'前缀表示，C 编码约定。二进制数字附小写“b”（例如，01010100b’或'01000011b'）。不是一个'h'或'b'表示的数字是十进制的。

3.2.3 单位

以下表格表示了文档里用到的单位。

表 3-1 单位

符号	单位
°C	摄氏度
dB	分贝
fF	飞法
Hz	赫兹
k	千，1000
K	千，2^10
KB	1024 字节，大约为一千字节
Kbit	1024 比特
kHz	千赫兹
kΩ	千欧姆
MHz	兆赫兹
MΩ	兆欧姆
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	百万分之一
SPS	每秒一次采样
σ	一个标准偏差
V	伏特

3.2.4 缩略语

缩略语	注释
ABUS	模拟输出总线
AC	交流电
ADC	模数转换器
AHB	高性能总线，一种 ARM 的数据传输总线
API	应用编程接口
APOR	模拟上电复位
BC	广播时钟
BIFC	比特位实现的功能连接
BINC	比特位实现的无链接
BOM	物料表
BR	比特率
BRA	总线请求确认
BRQ	总线请求确认
CAN	控制器局域网络
CBUS	比较器总线
CI	近位输入
CMP	比较器
CMRR	共模抑制比
CO	近位输出
CPU	中央处理器
CRC	循环冗余校验
CT	连续时间
DAC	数模转换
DC	直流
DFB	数字滤波模块
DI	数字或数据输入
DMA	直接内存访问
DMAC	直接内存访问寄存器
DNL	差分非线性
DO	数字或数据输出
DSI	数字信号互联
ECO	外部晶振
EEPROM	电可擦除可编程只读存储器
EMIF	外部内存接口
FB	反馈
FIFO	先进先出
FSR	满幅值
GIE	全局接口使能
GPIO	通用输入输出口
I^2C	内部集成电路通讯协议
ICE	在线电路仿真
IDE	集成开发环境
ILO	内部低速振荡器
IMO	内部主振荡器
INL	积分非线性
I/O	输入/输出
IOR	I/O 读
IOW	I/O 写
IRES	初始上电复位

缩略语	注释
IRA	中断请求确认
IRQ	中断服务程序
ISSP	在线串行编程
IVR	中断向量读
LFSR	线性反馈移位寄存器
LRb	最后收到的比特位
LRB	最后收到的字节
LSb	最低比特位
LSB	最低位字节
LUT	查找表
MISO	主入从出
MMIO	存储器映射的输入/输出
MOSI	主出从入
MSb	最高比特位
MSB	最高位字节
PC	程序计数器
PCH	程序计数器高
PCL	程序计数器低
PCLK	可编程时钟
PD	断电
PGA	可编程增益放大器
PHUB	外设集线器
PICU	端口中断控制单元
PM	电源管理
PMA	PSoC 存储器仲裁
POR	上电复位
PPOR	精确上电复位
PRS	伪随机代码序列
PSoC®	可编程片上系统
PSRAM	伪静态存随机存取储器
PSRR	电源抑制比
PSSDC	电源睡眠占空比
PVT	过程电压温度
PWM	脉冲宽度调制
RAM	随机存取存储器
RAS	行地址选通信
RETI	中断返回
RO	弛张振荡器
ROM	只读存储器
RW	读写
SAR	逐次逼近型寄存器
SC	开关电容
SIE	串行接口引擎
SIO	特殊 I/O 口

缩略语	注释
SE0	单端 0
SNR	信噪比
SOF	帧头
SOI	指令开始
SP	堆栈指针
SPD	连续的相位检测器
SPI	串行外围设备互连
SPIM	串行外围设备互连主机
SPIS	串行外围设备互连从机
SRAM	静态随机存储器
SROM	特权只读存储器
SSADC	单斜率模数转换器
SSC	监控系统调用
SWD	串行线调试
SWV	单线浏览器
TC	终止计数
TD	事件描述符
TIA	跨阻放大器
UART	通用异步接收/发送装置
UDB	通用数字模块
USB	通用串行总线
USBIO	USB I/O
VCO	电压控制振荡器
WDT	看门狗计时器
WDR	看门狗复位
XRES_N	外部复位，低有效

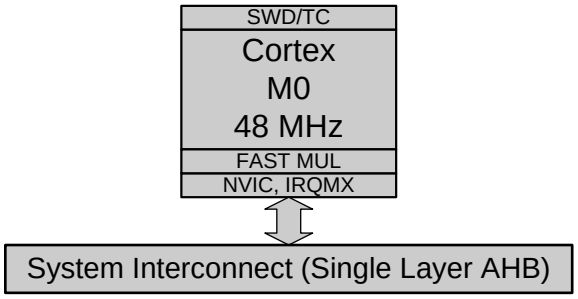
B 部分：中央处理器（CPU）子系统

这部分包括以下章节：

- Cortex-M0 中央处理器　第 26 页
- 中断　第 31 页

系统架构：

CPU 子系统结构框图



4 Cortex™-M0 处理器

PSoC 4 包含一个 ARM Cortex-M0 处理器。Cortex-M0 处理器基于一个低功耗的 32 位处理器内核，支持 3 级流水线、固定的存储器映射（高达 4GB 的可寻址存储空间）及 ARMv6-M Thumb 指令集；它还提供了一条单周期乘法指令和一个嵌套向量中断控制器（NVIC）。NVIC 可以实现较低的中断延迟。

Cortex-M0 处理器中除 CPU 内核外，还包括嵌套向量中断控制器（NVIC）、系统节拍定时器（SYSTICK Timer） 和 调试模块等。

本章节仅对 Cortex-M0 处理器做一个概述，如需对其详细了解，请参考《Cortex™-M0 Devices Generic User Guide》或《Cortex™-M0 Technical Reference Manual》，其可从网址 http://www.arm.com 上获得。

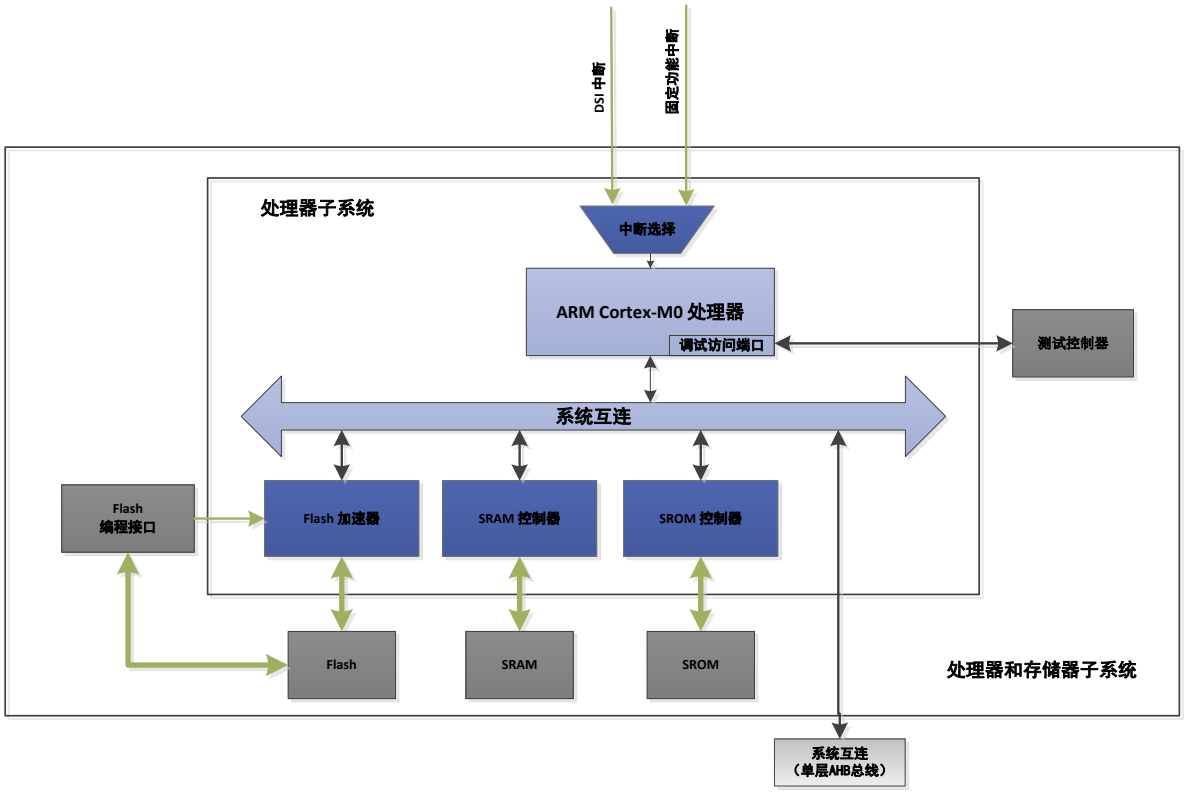
4.1 特性

PSoC 4 Cortex-M0处理器具有如下特性：

- 使用、编程、调试方便，便于8位/16位 MCU代码的移植
- 工作性能高达0.9 DMIPS/MHz。这有助于提高程序执行速度、降低功耗。
- 支持Thumb指令集。这可提高代码密度和对存储器的利用率。
- 集成嵌套向量中断控制器（NVIC）。NVIC可极大地缩短中断延时。
- 强大的调试能力：
  - 串行调试（SWD）
  - 断点（Break points）
  - 观察点（Watch points）

4.2 处理器子系统框图

图 4-1 PSoC 4 处理器子系统框图



4.3 Cortex-M0 处理器的使用

Cortex-M0 是一个 32 位的处理器，采用 32 位的数据通路、32 位的寄存器和 32 位的存储器接口，并支持大部分 16 位 Thumb 指令集和部分 32 位 Thumb-2 指令集。

Cortex-M0 支持两种操作模式：线程模式（Thread mode）和处理模式（Handler mode），并有一条单周期的 32 位乘法指令。

4.3.1 Cortex-M0 处理器内核寄存器

Cortex-M0 处理器内核有16个32位寄存器（详情见表4-1）：

- R0 to R12 – 通用寄存器。
- R13 – 堆栈指针（SP）。
- R14 – 链接寄存器（LR）。
- R15 – 程序计数器（PC）。

表 4-1. Cortex-M0 处理器内核寄存器

名称	类型 [a]	复位值	描述
R0-R12	可读写	未知	R0-R12 是供数据操作的 32 位通用寄存器。 R0~R7 可被所有指令访问；但 R8~R12 仅可被部分指令访问。
MSP	可读写	[0x00000000]	堆栈指针（SP）是寄存器 R13。 Cortex-M0 处理器支持两种堆栈指针：主堆栈指针（MSP）和进程堆栈指针（PSP）。在同一时刻，CPU 仅使用其中一种堆栈指针。 在线程模式中，寄存器 *CONTROL* 的 bit[1]指示了当前的有效堆栈指针： 0 = 主堆栈指针 (MSP)。这是复位默认值。 1 = 进程堆栈指针 (PSP)。 在处理模式中，仅使用主堆栈指针。 复位时，处理器将地址 0X00000000 的值加载到 MSP 中。
PSP	可读写	[0x00000000]	
LR	可读写	未知	链接寄存器（LR）是寄存器 R14。它保存子程序、函数调用和异常的返回信息。复位时，LR 的值是未知的。
PC	可读写	[0x00000004]	程序计数器（PC）是寄存器 R15。它包含当前的程序地址。复位时，处理器将复位向量（地址：0x00000004）的值加载到 PC。此值的 Bit[0]在复位时被加载到 EPSR 的 T 位，且必须为‘1’。
PSR	可读写	未知 [b]	程序状态寄存器（PSR）由下列 3 种寄存器组合而成： 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	可读写	未知	应用程序状态寄存器(APSR)包含执行完前面的指令后条件标志的当前状态。
EPSR	只读	未知 [b]	执行程序状态寄存器(EPSR)包含 Thumb 状态位。
IPSR	只读	0X00000000	中断程序状态寄存器(IPSR)包含当前中断服务程序（ISR）的异常编号。
PRIMASK	可读写	0X00000000	优先级屏蔽寄存器（PRIMASK）可阻止所有优先级可配置的异常被激活。
CONTROL	可读写	0X00000000	CONTROL 寄存器控制处理在线程模式下所使用的堆栈。

a. 描述在线程模式和处理模式下程序执行过程中的访问类型。调试访问时可与此不同。
b. Bit[24]是 Thumb 状态位，其值是从复位向量的 bit[0]加载进来的。

在 32 位的 PSR 中，3 个寄存器的位域分配是互斥的，见表 4-2。

表 4-2. Cortex-M0 PSR 位域分配

位	PSR 寄存器	名称	功能
31	APSR	N	负值标志
30	APSR	Z	零值标志
29	APSR	C	进位或借位标志
28	APSE	V	溢出标志
27 – 25			保留
24	EPSR	T	Thumb 状态位。此值必须总是为 1。当 T 位为 0（非 Thumb 状态）时，执行指令会导致 HardFault 异常或锁定故障。
23 – 6			保留
5 – 0	IPSR	异常编号	指示当前异常编号： 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

应用程序可通过 MSR 或 CPS 指令改变 PRIMASK 的 bit[0]值来禁止或重新使能异常。当 Bit[0] = 0 时，使能异常；当 Bit[0] = 1 时，禁止所有优先级可配置的异常。关于中断的详细介绍，请参考中断章节。

4.3.2 操作模式

Cortex-M0 处理器支持两种操作模式：

- 线程模式（Thread Mode） –用于执行应用程序。在线程模式下，寄存器 *CONTROL* 的 bit[1]控制着 CPU 使用主堆栈指针还是进程指针：
  - □ 0 = MSP 是 CPU 当前使用的堆栈指针
  - □ 1 = PSP 是 CPU 当前使用的堆栈指针
- 处理模式（Handler Mode） – 用于处理异常。在处理模式下，CPU 仅使用主堆栈指针（MSP）。

默认情况下，线程模式使用 MSP。在线程模式下，如需将堆栈指针切换到 PSP，只需要使用 MSR 指令将寄存器 CONTROL 的 bit[1]设置为 1 即可。但需要注意的是，当更改堆栈指针时，必须在 MSR 指令后立即使用一个 ISB 指令。这样可确保 ISB 之后的指令执行时使用新的堆栈指针。

处理模式始终使用 MSP。因此，在处理模式下，CPU 忽略软件对寄存器 CONTROL 的写操作。异常进入和返回机制会自动更新寄存器 CONTROL。

4.3.3 指令集

Cortex-M0 执行 Thumb 指令集。如需详细了解此版本的 Thumb 指令集，请参考《Cortex™-M0 Devices Generic User Guide》。

指令操作数可以是一个 ARM 寄存器、立即数或特殊指令参数。指令对操作数进行操作后，通常 将结果保存在目标寄存器中。一些指令不能用 PC 或 SP 作为操作数或目标寄存器。

表 4-3 Cortex-M0 指令集

助记符	简述
ADCS	进位加法
ADD{S}	加法
ADR	将基于 PC 相对偏移的地址读到寄存器
ANDS	位与操作
ASRS	算术右移
B{cc}	跳转{有条件地}
BICS	位清除
BKPT	断点
BL	带链接的跳转
BLX	带链接的间接跳转
BX	间接跳转
CMN	比较负值
CMP	比较
CPSID	更改处理器状态，禁止中断
CPSIE	更改处理器状态，使能中断
DMB	数据内存屏障
DSB	数据同步屏障
EORS	异或
ISB	指令同步屏障
LDM	加载多个寄存器，访问之后递增地址
LDR	从基于 PC 相对偏移地址上加载寄存器 用字加载寄存器
LDRB	用字节加载寄存器
LDRH	用半字加载寄存器
LDRSB	用有符号的字节加载寄存器
LDRSH	用有符号的半字加载寄存器
LSLS	逻辑左移
LSRS	逻辑右移
MOV{S}	传送
MRS	从特殊寄存器传送到通用寄存器
MSR	从通用寄存器传送到特殊寄存器
MULS	乘法，32 位结果值
MVNS	位非
NOP	无操作
ORRS	逻辑或
POP	出栈，将堆栈中的内容放入寄存器
PUSH	压栈，将寄存器中的内容压入堆栈
REV	反转字里面的字节顺序
REV16	反转每半字里面的字节顺序
REVSH	反转有符号半字里面的字节顺序
RORS	循环减法
RSBS	反向减法
SBCS	进位减法
SEV	发送事件
STM	存储多个寄存器，访问后递增地址
STR	将寄存器作为字来存储
STRB	将寄存器作为字节来存储
STRH	将寄存器作为半字来存储
SUB{S}	减法
SVC	特权用户调用
SXTB	符号扩展字节
SXTH	符号扩展半字
TST	基于测试的逻辑与
UXTB	0 扩展字节
UXTH	0 扩展半字
WFE	等待事件
WFI	等待中断

4.3.3.1 地址对齐

在对齐的存储访问中，字对齐的地址（地址的低两位为 00）用于对一个字或多个字的访问；半字对齐的地址（地址的最低位为 0）用于对半字的访问；对字节访问可是任何地址。

Cortex-M0 处理器不支持非对齐的存储访问。任何非对齐的存储访问，将产生一个 HardFault 异常。

4.3.3.2 存储器格式

PSoC 4 Cortex-M0 处理器采用小端（little-endian）格式，即在字单元中最低有效字节存放在最低地址，最高有效字节存放在最高地址处。

4.3.4 系统节拍定时器

系统节拍定时器（Systick Timer）集成于 NVIC 中，可产生 SYSTICK 异常。这个异常可被用作操作系统的系统节拍。此定时器包含一个可重载的 24 位寄存器（递减计数）。系统节拍定时器使用 Cortex-M0 的内部时钟作为其时钟源。

4.3.5 调试

PSoC 4 支持一个 SWD 调试接口，4 个硬件断点和 2 个观察点设置，以方便用户调试。详情请见编程和调试接口章节

5 中断

中断是嵌入式产品应用中的重要组成部分。在 PSoC 4 等片上系统（SoC）架构中，片上外设与 CPU 间的通讯一般使用中断来完成。PSoC 4 为中断处理和异常处理提供一个统一的向量表。

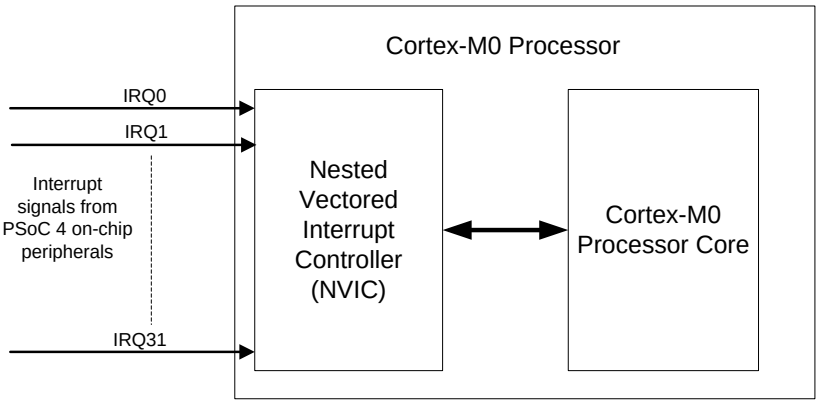
5.1 特性

PSoC 4 中断具有如下特性：

- 多达 32 个中断
- 嵌套向量中断控制器(NVIC)集成于 ARM Cortex-M0 处理器中，可实现较低的中断延迟
- 可选择保存在 Flash 或 SRAM 中的向量表
- 每个中断可配置为四种优先级（0~3）
- 支持两种中断源：固定功能中断源或 DSI 中断源
- 支持电平触发和边沿触发

5.2 中断的实现

图 5-1. 中断控制框图



如图 5-1 所示，PSoC 4 支持 32 个中断（由 Cortex-M0 处理器产生的异常，没有显示在图中）。嵌套向量中断控制器(NVIC)集成于 ARM Cortex-M0 处理器中，可以实现较低的中断延迟。NVIC 具有如下功能：

- 使能/禁止每个中断
- 连接 Cortex-M0 处理器内核
- 不同中断间的优先级处理
- 中断的嵌套
- 软件设置每个中断的挂起、清除

5.3 中断和异常的工作原理

5.3.1 PSoC 4 中的中断/异常处理

本章节将介绍中断或异常的处理过程。

当一个中断产生时，其首先被发送至 NVIC，并进入挂起状态，等待 CPU 处理。NVIC 再向 CPU 发送中断请求信号和异常编号。一旦 CPU 接收到来自 NVIC 的中断请求信号和异常编号，就将当前上下文（通用寄存器、程序计数器 PC 和其他寄存器）压入堆栈，并从向量表（见表 5-1）中取出与向量编号相对应的异常处理程序地址。程序计数器将跳转到异常处理程序地址处执行异常处理程序。一旦完成异常处理程序的操作，CPU 将对储存在堆栈内的中断上下文进行出栈操作，接着从中断处继续执行程序。

如果一个中断正在被 CPU 响应时 NVIC 又接收到一个中断请求，或者 NVIC 同时接收到多个中断请求时，其首先会评估不同中断间的优先级，然后将最高优先级中断的向量编号发送给 CPU。

CPU 对异常的处理与中断处理相同。每个异常事件均对应唯一的异常编号，CPU 通过异常编号读取异常程序地址并执行。

以上是 PSoC 4 中断和异常的简单介绍，如需深入了解 NVIC，请参考《Cortex™-M0 Devices Generic User Guide》中的有关 NVIC、异常处理相关章节。

5.3.2 电平中断和脉冲中断

图 5-2. 电平中断

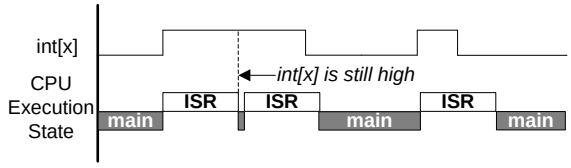
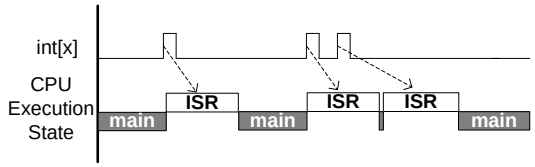


图 5-3. 脉冲中断



在 PSoC4 中，NVIC 可支持电平中断和脉冲中断。图 5-2 和图 5-3 分别是电平中断和脉冲中断的工作原理图。假设中断信号初始值为非活动状态，以下事件序列描述了电平中断和脉冲中断的处理过程：

- 当中断事件发生时，NVIC 将记录此中断请求。因该中断还未被 CPU 处理，故其处于挂起状态。
- NVIC 将中断向量编号及中断请求信号一起发送给 CPU。当 CPU 开始执行此中断的中断服务程序时，中断的挂起状态切换为活动状态。
- 在电平中断中，当 CPU 执行完中断服务程序，如中断信号仍然保持，则挂起中断，并再次执行中断服务程序（见图 5-2）。否则中断状态切换为非活动状态。
- 在脉冲中断中，当 CPU 执行中断服务程序的过程中，如果有脉冲中断再次发生，该中断被挂起（即使有多个中断发生也仅被记录为一个挂起请求），此时中断状态为挂起并活动状态，当前中断处理程序结束后则继续处理挂起的中断（见图 5-3）。如果没有新的中断发生则中断进入非活动状态。

5.3.3 异常向量表

PSoC 4 的异常向量表（见表 5-1）保存了所有异常的入口地址。当有异常发生时，CPU 会从异常向量表中取相应异常的入口地址，并执行相应的服务程序。

在表 5-1 中，从基地址开始的 4 个字节在器件复位时用来初始化主堆栈指针（MSP），而非异常向量。在 PSoC 4 中，用户可在程序的启动阶段，通过配置 VECS_IN_RAM 位(CPUSS_CONFIG[0])，来选择使用保存在 Flash 还是 SRAM 中的向量表。

- VECS_IN_RAM = 0: 选择保存在 Flash 中的向量表，起始地址为 0x00000000。如需修改异常向量入口地址，需要修改 Flash 代码。
- VECS_IN_RAM = 1: 选择保存在 SRAM 中的向量表，起始地址为 0x20000000。程序可通过修改 SRAM 的向量表，动态地修改向量入口地址。

如需异常源（异常编号：1 ~ 15）的详细信息，请参考异常源章节。

如需中断源（异常编号：16 ~ 47）的详细信息，请参考中断源章节。

表 5-1. PSoC 4 异常向量表

异常编号	异常类型	优先级	向量地址	功能
	初始化堆栈指针值	-	基地址： 0x00000000 – 向量表保存在 Flash 中 0x20000000 – 向量	初始化堆栈指针值（非异常，放在这里为了向量表的完整性）

			表保存在 SRAM 中	
1	复位（Reset）	-3，最高	基地址 + 0x04	异常模块将 Reset 看作一种特殊形式的异常来处理。当 Reset 发生时，处理器将停止操作，且停止位置不确定。当 Reset 解除时，应用程序重新从向量表中 Reset 入口地址处启动。
2	不可屏蔽中断（NMI）	-2	基地址 + 0x08	不可屏蔽中断（NMI）可以由硬件信号、软件或 System Call 函数调用触发。其是除 Reset 之外的最高优先级异常。NMI 永远使能，固定优先级为-2。NMI 不可被屏蔽，它的执行不能被其他任何异常中止， 也不能被除 Reset 之外的任何异常抢占。 在 PSoC 4 中，用户可将一个硬件数字信号配置为 NMI 功能（见表 5-2）。
3	HardFault	-1	基地址 + 0x0C	HardFault 是由于在正常操作过程中或异常处理过程中出错而产生的一个异常。HardFault 的优先级为-1。它的优先级高于任何可配置优先级的异常。
4-10	保留	-	基地址 +0x10 ~ 基地址 + 0x28	--
11	管理程序调用（SVCall）	可配置（0~3）	基地址 + 0x2C	SVCall 是一个由 SVC 指令触发的异常。在 OS 环境下，应用程序可以使用 SVC 指令来访问 OS 内核函数和设备驱动。
12-13	保留	-	基地址 + 0x30 ~ 基地址 + 0x34	--
14	PendSV	可配置（0~3）	基地址 + 0x38	PendSV 是一个中断驱动的系统级服务请求。在 OS 环境下，当没有其他异常有效时，使用 PendSV 来进行任务切换。
15	SysTick	可配置（0~3）	基地址 + 0x3C	SysTick 是系统定时器到达零时产生的异常。系统也可以通过软件产生一个 SysTick 异常。在 OS 环境下，处理器可以将这个异常用作系统节拍。
16	外部中断（IRQ0）	可配置（0~3）	基地址 + 0x40	中断是由外设产生或软件请求触发的异常。在系统中，外设使用中断与处理器进行通讯。中断源列表请见表 5-2。
…	…	可配置（0~3）	…	
47	外部中断（IRQ31）	可配置（0~3）	基地址 + 0xBC	

5.4 异常源

接下来将介绍表 5-1 中的不同异常源（异常编号：1 ~ 15）。

5.4.1 复位异常

器件的复位会引起复位异常，其拥有最高优先级（-3），永远使能。在 PSoC 4 中，上电复位(POR)、来自管脚 XRES 的外部信号或看门狗复位等均可以引起器件复位。器件复位后，保存在 SROM 中的启动程序首先被执行，来配置器件参数。SROM 中保存了复位异常的向量地址，其保存了执行 SROM 中启动程序的起始地址。CPU 执行完 SROM 中的启动程序后，会跳转到 Flash，从地址 0x00000004（见表 5-1）读取保存在 Flash 中的启动代码的入口地址，并跳转执行。

注意：在器件出厂时，SROM 中的启动程序和数据已经被固化，用户不可读取或修改。保存在 SRAM 向量表中的复位异常地址不可用，因为器件复位后，总是选择保存在 Flash 向量表中的复位异常地址；并且选择 SRAM 中向量表的寄存器配置是在启动代码（保存在 Flash 中）阶段完成的。

5.4.2 不可屏蔽中断（NMI）

不可屏蔽中断（NMI）的优先级是-2，具有除复位外的最高优先级，并且永远使能。在 PSoC 4 中，用户可以通过如下 3 种方式来触发 NMI：

- **硬件信号：**PSoC 4 支持通过数字信号来触发 NMI 的方式，对应的中断源是 irq_out[0]（见表 5-2）；CPU 执行 Flash 或 SRAM 向量表中指向的 NMI 处理程序。PSoC 4 具有灵活的数字信号互连功能，因此可将片上外设的数字输出、外部管脚信号连接到 irq_out[0]中断线上，来触发 NMI 异常。
- **设置 NMIPENDSET 位：**向中断控制状态寄存器 CM0_ICSR 的 NMIPENDSET 位写 1，可触发 NMI 异常；CPU 执行 Flash 或 SRAM 向量表中指向的 NMI 处理程序。
- **系统调用（System Call）：**此功能主要应用于非易失性存储器编程，比如对 Flash 进行写、校验等操作（见非易失性存储器编程章节）。当用户向寄存器 CPUSS_SYSREG 的 SYSCALL_REQ 位写 1 时，将触发 NMI 异常；CPU 执行保存在 SROM 中的 NMI 异常处理程序，但用户不可读取或修改 SROM 中的程序。

5.4.3 HardFault 异常

HardFault 是由于正常操作过程中或异常处理过程中出错而产生的一个异常，永远使能。HardFault 具有固定优先级-1，其优先级比任何可配置优先级的异常均高。HardFault 用于处理不同类型的故障；例如，执行未定义的指令或访问无效的内存地址均会引起 HardFault 异常 。Cortex-M0 处理器不向 HardFault 异常处理程序提供故障状态信息，但是允许执行完异常处理程序后返回并继续执行程序，这样程序就具有从故障情况下恢复过来的能力。

5.4.4 管理程序调用（SVCall）异常

SVCall 是一个由 SVC 指令触发的异常，永远使能。SVC 指令可以使系统进入特权模式。在 OS 环境下，应用程序可以使用 SVC 指令来访问 OS 内核函数和设备驱动。

注意：在 PSoC 4 中，系统进入特权模式是通过 System Call 触发 NMI 异常实现的，而不是使用 SVCall 异常；也就是说，在 PSoC 4 中，使用 SVC 指令，系统不进入特权模式。用户可以按照需求，定义 SVCall 异常的处理程序。

用户可通过 SHPR2[31:30]来配置 SVCall 异常的优先级（0~3）。当 SVC 指令被执行时，SVCall 异常会进入挂起状态并等待 CPU 处理。系统处理控制和状态寄存器 SHCSR 的 SVCALLPENDED 位可用来检查或修改 SVCall 异常的挂起状态。

5.4.5 PendSV 异常

PendSV 是又一个管理程序调用异常，与 SVCall 类似。PendSV 永远使能，且优先级可配置。当向中断控制状态寄存器 CM0_ICSR 的 PENDSVSET 位写 1 时，可触发 PendSV 异常。向寄存器 CM0_ICSR 的 PENDSVCLR 位写 1，可清除 PendSV 异常。其优先级（0~3）是通过配置 SHPR3[23:22]实现的。

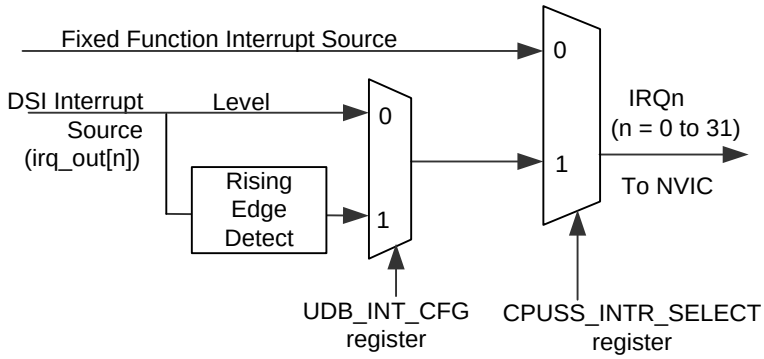
5.4.6 SysTick 异常

Cortex-M0 处理器支持系统时钟 SysTick。SysTick 是一个 24 位、递减计数的定时器，可用作 RTOS 的系统节拍、高速报警定时器或者简单的计数器。用户可配置 SysTick 控制和状态寄存器 CM0_SYST_CSR 的 TICKINT 位来使能异常，当计数值到 0 时，使其产生中断，即 SysTick 异常。SysTick 异常的优先级（0~3）是通过配置 SHPR3[31：30]来实现的。当用户向中断控制状态寄存器 CM0_ICSR 的 PENDSTSET 位写 1 时，可产生一个 SysTick 异常；向中断控制寄存器 CM0_ICSR 的 PENDSTCLR 位写 1 时，清除 SysTick 异常。

5.5 中断源

PSoC 4 支持 32 个外设中断（IRQ0 ~ IRQ31 或异常编号 16~17），见表 5-1。表 5-2 列出了每个中断的中断源。PSoC 4 支持灵活的中断源选择。如图 5-4 所示，每个中断有两个中断源：固定功能中断源和数字系统互连（DSI）中断源。应用程序可通过配置寄存器 CPUSS_INTR_SELECT 为每个中断选择使用固定功能中断源还是 DSI 中断源（见表 5-2）。

图 5-4. 中断源多路选择



注意：DSI 中断信号（irq_out[n]）是通过 PSoC Creator 软件对数字信号进行路由实现的。用户不需要手动配置。

固定功能中断源是来自片上外设的中断信号，比如 PWM、串行通信模块（SCB）、SAR ADC 等产生的中断信号。固定功能中断由不同外设状态的逻辑或产生，在执行中断服务程序时，CPU 可以通过读取外设状态寄存器来判断产生中断的中断源。固定功能中断通常是电平中断。在中断服务程序中，CPU 通过读取外设状态寄存器来清中断；如不读取，则中断将一直存在，并且 CPU 持续执行中断服务程序。

另一种中断源是 DSI 中断信号。任何数字信号均可通过路由作为 DSI 中断源，比如来自通用数字模块（UDB）的数字输出信号或引脚上的数字输入信号。通过对路由来的 DSI 信号进行上升沿检测后的信号也可作为中断源（见图 5-4）。图 5-4 中的边沿检测电路可以将 DSI 线上的上升沿信号转换为两个系统时钟周期宽度的脉冲信号，这样可确保 DSI 线上每个上升沿信号可触发一次中断。边沿检查电路也可为不能向 NVIC 产生合适电平中断信号的中断源提供一种使用方法。图 5-4 中的 UDB_INT_CFG 寄存器用于选择 DSI 信号是直接至 NVIC 还是经边沿检测电路至 NVIC。

表 5-2. PSoC 4 中断源列表

中断编号#	Cortex-M0 异常编号#	固定功能中断源	DSI 中断源
NMI (见异常源)	2		irq_out[0]
IRQ0	16	GPIO P0 (端口中断)	irq_out[1]
IRQ1	17	GPIO P1 (端口中断)	irq_out[1]
IRQ2	18	GPIO P2 端口中断)	irq_out[2]
IRQ3	19	GPIO P3 (端口中断)	irq_out[3]
IRQ4	20	GPIO P4 (端口中断)	irq_out[4]
IRQ5	21	<DSI-only>	irq_out[5]
IRQ6	22	<DSI-only>	irq_out[6]
IRQ7	23	<DSI-only>	irq_out[7]
IRQ8	24	LPCOMP（低功耗比较器）	irq_out[8]
IRQ9	25	WDT（看门狗定时器）	irq_out[9]
IRQ10	26	SCB1（串行通信模块 1）	irq_out[10]
IRQ11	27	SCB2（串行通信模块 2）	irq_out[11]
IRQ12	28	SPC（系统性能控制器）	irq_out[12]
IRQ13	29	PWR（电源管理）	irq_out[13]
IRQ14	30	SAR ADC (逐次逼近型 ADC)	irq_out[14]
IRQ15	31	CSD（电容感应模块计数器溢出中断）	irq_out[15]
IRQ16	32	TCPWM0（定时器/计数器/PWM0）	irq_out[16]
IRQ17	33	TCPWM1（定时器/计数器/PWM1）	irq_out[17]

中断编号#	Cortex-M0 异常编号#	固定功能中断源	DSI 中断源
IRQ18	34	TCPWM2（定时器/计数器/PWM2）	irq_out[18]
IRQ19	35	TCPWM3（定时器/计数器/PWM3）	irq_out[19]
IRQ20	36	<DSI-only>	irq_out[20]
IRQ21	37	<DSI-only>	irq_out[21]
IRQ22	38	<DSI-only>	irq_out[22]
IRQ23	39	<DSI-only>	irq_out[23]
IRQ24	40	<DSI-only>	irq_out[24]
IRQ25	41	<DSI-only>	irq_out[25]
IRQ26	42	<DSI-only>	irq_out[26]
IRQ27	43	<DSI-only>	irq_out[27]
IRQ28	44	<DSI-only>	irq_out[28]
IRQ29	45	<DSI-only>	irq_out[29]
IRQ30	46	<DSI-only>	irq_out[30]
IRQ31	47	<DSI-only>	irq_out[31]

5.6 使能/禁止异常

PSoC 4 支持对 32 个中断分别使能或禁止的功能。如果中断被禁止，则 NVIC 不处理相应的中断请求。用户可以通过 NVIC 提供的中断设置-使能寄存器（CM0_ISER）和中断清除-使能寄存器（CM0_ICER）来分别使能/禁止每个中断。这些寄存器均是 32 位宽度，每个位对应相同编号的中断。在软件中可以通过读取这些寄存器来获得中断的使能状态。表 5-3 描述了这两个寄存器的读写权限。

注意：向寄存器 CM0_ISER 和 CM0_ICER 的相应位写 0，不起作用。

表 5-3. 中断使能/禁止寄存器

寄存器	操作	位值	备注
中断设置-使能寄存器（CM0_ISER）	写	1	使能中断
		0	无影响
	读	1	中断使能
		0	中断禁止
中断清除-使能寄存器（CM0_ICER）	写	1	禁止中断
		0	无影响
	读	1	中断使能
		0	中断禁止

寄存器 CM0_ISER 和 CM0_ICER 仅对 32 个中断（IRQ0 ~ IRQ31）有效。15 个异常的使能/禁止可通过相应的寄存器来实现，见异常源章节。

Cortex-M0 处理器中的寄存器 PRIMASK 可以屏蔽所有可配置优先级的异常，不论是否被使能。除 Reset、NMI 和 HardFault 外，其他异常均可配置优先级，见表 5-1。当设置 PRIMASK[0]位为 1 时，CPU 不执行可配置优先级异常的服务程序；但是可配置优先级的异常可以保持在挂起状态，PRIMASK[0]位被清 0 后，CPU 会执行相应的服务程序。

注意：0：指最高优先级；3：指最低优先级。

5.7 异常状态

每个异常均处于下面其中一种状态：

表 5-4. 异常状态

异常状态	描述
非活动	异常处于非活动、非挂起 状态。异常被禁止或者没有被触发。
挂起	异常等待处理器处理。当一个来自外设或软件的异常请求发生时，NVIC 将相应的异常置为挂起状态，等待 CPU 处理。
活动	一个异常正被处理器处理，但处理尚未结束。一个高优先级的异常可以打断一个低优先级异常的处理。在这种情况下，两个异常均处于活动状态。
活动和挂起	一个异常正被处理器处理，而且有一个来自同一个异常源的异常正在等待处理。

中断控制状态寄存器（CM0_ICSR）包含了异常的各种状态信息。

- VECTACTIVE 位 (CM0_ICSR [8:0]) : 保存了当前正在执行的异常编号，其值与 IPSR[8:0]相同。0 表示 CPU 没有执行异常，运行在线程模式。
- VECTPENDING 位 (CM0_ICSR[20:12]):保存处于挂起状态的最高优先级的异常编号。0 表示没有被挂起的异常。
- ISRPENDING 位 (CM0_ICSR[22]):指示是否有中断(IRQ0 – IRQ-31)处于挂起状态。

5.7.1 挂起异常

当一个外设向 NVIC 产生一个中断请求信号或者一个异常事件发生时，相应的异常会进入挂起状态。一旦 CPU 开始执行相应的异常处理程序，异常将由挂起状态转为活动状态。异常处理程序结束后，中断可能进入非活动状态或者重新挂起状态。

NVIC 允许程序通过中断设置-挂起寄存器（CM0_ISPR）和中断清除-挂起寄存器（CM0_ICPR）来修改中断的挂起状态。这两个寄存器均为 32 位宽度，并且每个位对应相同编号的中断。表 5-5 描述了这两个寄存器的读写权限。

表 5-5 中断设置-挂起/清除-挂起寄存器

寄存器	操作	位值	注释
中断设置-挂起寄存器（CM0_ISPR）	写	1	设置中断挂起
		0	无影响
	读	1	中断挂起中
		0	中断非挂起中
中断清除-挂起寄存器（CM0_ICPR）	写	1	清除中断挂起
		0	无影响
	读	1	中断挂起中
		0	中断非挂起中

如果某个中断已经是挂起状态，则对其 CM0_ISPR 相应位写‘1’是不起作用的。不论中断是否被使能，均可更新中断的挂起位。如果中断被禁止，则中断线一直保持挂起状态直到中断被使能。

注意：寄存器 CM0_ISPR 和 CM0_ICPR 仅 32 个外设中断（异常编号：16~47）有效。关于编号 1~15 的异常的挂起，请参考异常源章节。

5.7.2 异常优先级

不同的优先级配置，可以使 CPU 对某时刻的多个异常进行仲裁。PSoC 4 支持灵活的异常优先级选择，见表 5-1。Reset、NMI 和 HardFault 的优先级分别是-3、-2 和-1；除此之外的其他异常均可配置为 0~3 的优先级。

注意：优先级的数值越小，对应异常的优先级越高。即复位异常（优先级：-3）具有最高优先级。

PSoC 4 支持异常嵌套，即高优先级异常可以抢占/中断低优先级异常处理程序的执行。PSoC 4 支持最多 4 级异常的嵌套。但当 CPU 接收到两个或多个相同优先级的异常时，低异常编号的异常会被优先处理。

配置异常（异常编号：1~15）优先级的寄存器配置见异常源章节。

PSoC 4 提供了 8 个中断优先级寄存器（CM0_IPR），用户可通过其为每个中断（IRQ0~IRQ31）配置四种优先级（0~3），其中 0 是最高优先级，3 是最低优先级。寄存器 CM0_IPR 的位定义见表 5-6。

表 5-6 中断优先级寄存器的位定义

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

5.8 堆栈使用

Cortex-M0 有两种堆栈指针：主堆栈指针（MSP）和进程堆栈指针（PSP）。在同一时刻，CPU 仅使用其中一种堆栈指针。当 CPU 正在线程模式（Thread mode）下执行主代码时，一个中断请求发生，CPU 会将当前的通用寄存器状态压入堆栈中，然后进入处理模式（Handler mode）执行中断服务程序。在线程模式中，*Control* 寄存器的 *Active Stack Pointer* 位指示了当前的有效堆栈指针。在处理模式中，仅使用主堆栈指针。

Cortex-M0 使用递减堆栈。堆栈指针始终指向最后一个入栈项。当处理器将一个新的项压入堆栈时，堆栈指针递减，然后将该项写入新的存储器单元中。

在线程模式中，当一个异常请求发生时，CPU 使用 *Control* 寄存器 *Active Stack Pointer* 位指示的堆栈指针来存储当前的通用寄存器值。入栈操作完成后，CPU 进入处理模式并执行异常处理程序。如果执行异常处理程序过程中有更高优先级异常发生，则 CPU 使用主堆栈指针进行入栈/出栈操作，因为此时 CPU 已经运行于处理模式。

Cortex-M0 使用了 Tail-chaining 和 Late-arriving 两种机制来减少中断延迟。下面是关于这两种机制的简单介绍。如需详细信息，请参考《Cortex™-M0 Devices Generic User Guide》的异常模式章节。

- Tail-Chaining

Tail-Chaining(末尾连锁)机制可加速对异常的处理。当一个异常处理程序结束时，如果一个挂起的异常满足异常进入的要求，处理器就跳过堆栈弹出，并执行新的异常处理程序。

- Late-arriving

Late-arriving（迟来） 机制可加速对抢占的处理。如果一个更高优先级的异常在前一个异常正在保存状态的过程中出现，处理器就转去处理这个更高优先级的异常，并开始提取这个异常的向量。状态保存不受迟来异常的影响，因为两个异常保存的状态相同。从迟来异常的异常处理程序返回时，处理器将遵守正常的末尾连锁规则。

5.9 中断和低功耗模式

PSoC 4 支持通过外设中断来唤醒处于睡眠、深度睡眠或休眠模式的器件。当唤醒源产生中断时，中断控制器（WIC）产生一个唤醒信号，使设备进入活动模式或复位状态。从睡眠或深度睡眠下唤醒，系统进入活动模式并处理中断服务程序；从休眠模式下唤醒，系统会进入复位状态。

在 Cortex-M0 中，中断等待指令（WFI）可将程序挂起并进入睡眠、深度睡眠或休眠等电源模式。关于进入/退出低功耗模式的方法，请参考电源模式章节。根据电源管理模式，固定功能中断源可分为 3 类：

- 在活动、睡眠、深度睡眠和休眠模式下都可工作的固定功能中断源（端口中断和低功耗比较器输出中断）。
- 仅在活动、睡眠和深度睡眠模式下可工作的固定功能中断源（来自 WDT 和 SCB1/SCB2 的中断信号）。
- 仅在活动模式下可工作的固定功能中断源（来自其他所有固定中断源的中断信号）。

在深度睡眠或休眠模式下，DSI 中断源（irq_out[n],见图 5-4）不能唤醒器件；如果通过寄存器 CPUSS_INTR_SELECT 选择了 DSI 中断源，则相应的固定中断源即使被触发，仍然无法唤醒器件。

5.10 异常 - 初始化和配置

本章节主要介绍初始化和配置异常的步骤。

1. 选择异常向量表：使用异常向量表的第一步是选择存放在 Flash 中还是 SRAM 中的异常向量表。用户可在代码启动阶段通过配置 VECS_IN_RAM 位（CPUSS_CONFIG[0]）来实现。

- 0：选择存放在 Flash 中的向量表。
- 1：选择存放在 SRAM 中的向量表。

建议用户选择存放在SRAM中的向量表，这样可以动态地修改异常入口地址。如果选择存放在Flash中的向量表，如需修改向量表内容，则需要修改Flash中的程序，这样会破坏原来的程序。但是保存在SRAM向量表中的复位异常地址不可用，因为器件复位后，总是选择保存在Flash向量表中的复位异常地址。

2.配置要使用的异常：用户可按照如下步骤来分别配置需要使用的异常。

a.异常源或中断源配置：设置中断产生条件，配置中断源（见图 5-4）。

b.编写异常处理函数，并将函数地址写入异常向量表（见表 5-1）。

c.设置异常异常优先级，见异常优先级章节。

d.使能异常，见使能/禁止异常章节。

5.11 中断和异常寄存器

寄存器简称	寄存器名称
CM0_ISER	中断设置-使能寄存器
CM0_ICER	中断清除-使能寄存器
CM0_ISPR	中断设置-挂起寄存器
CM0_ICPR	中断清除-挂起寄存器
CM0_IPR	中断优先级寄存器
CM0_ICSR	中断控制状态寄存器
CM0_AIRCR	应用中断和复位控制寄存器
CM0_SCR	系统控制寄存器
CM0_CCR	配置和控制寄存器
CM0_SHPR2	系统处理优先级寄存器 2
CM0_SHPR3	系统处理优先级寄存器 3
CM0_SHCSR	系统处理控制盒状态寄存器
CM0_SYST_CSR	SysTick 控制和状态寄存器
CPUSS_CONFIG	CPU 子系统配置寄存器
CPUSS_SYSREQ	系统请求寄存器
CPUSS_INTR_SELECT	中断多路选择寄存器
UDB_INT_CFG	UDB 子系统中断配置寄存器

5.12 中断和异常相关文档

- ARMv6-M Architecture Reference Manual – 介绍 ARM Cortex-M0 的架构，其包括指令集、NVIC 架构和 CPU 寄存器描述。
- Cortex™-M0 Devices Generic User Guide – Cortex-M0 的用户使用手册

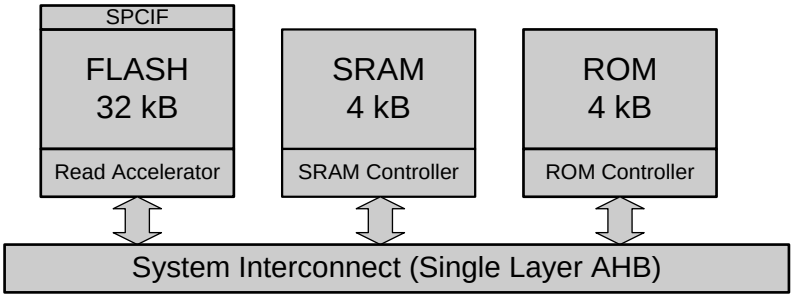
C 部分：存储器系统

这部分包括以下章节：

■ 存储器 第 42 页

系统架构：

存储器系统结构框图



6 存储器

所有 PSoC4 的存储器，包括闪存（Flash）、静态存储器(SRAM) 、只读管理存储器（SROM）以及所有的寄存器都可以被 CPU 和调试系统寻址（大多数情况下）。本章将详述所有存储器和寄存器的地址分布。

6.1 特性

PSoC4 的存储系统具备以下特性：

- 32KB闪存， 4KB 静态存储器（SRAM）
- 4KB 只读管理存储器 （SROM）包含启动和配置信息
- ARM Cortex-M0 线性寻址空间，包括代码区、静态存储器区、外设区以及中央处理器（CPU）内部寄存器区
- 闪存映射到Cortex-M0 寻址空间的代码区
- 静态存储器映射到Cortex-M0 寻址空间的SRAM 区
- 外设寄存器映射到Cortex-M0的寻址空间的外设区
- Cortex-M0 私有外设总线区(PPB)包括了在CPU中实现的寄存器，包括嵌套向量中断寄存器、SysTick寄存器以及系统控制寄存器。更多内容请见第四章：Cortex-M0 CPU。

6.2 工作原理

PSoC4 的内存映射如下表。如需更详细的信息，请参见： PSoC4 寄存器技术参考手册。

32-bit (4GB)的地址空间被分为几个区域，如下表 6-1。注意可以从内存映射的代码区或者静态存储区（SRAM）执行指令。

表 6-1. Cortex-M0 地址分布

地址范围	名称	使用
0x00000000 – 0x1FFFFFFF	代码 (Code)	执行代码区；也可以存放数据；包括从地址 0 开始的异常向量表
0x20000000 – 0x3FFFFFFF	静态寄存区（SRAM）	数据区，也可以存放代码。
0x40000000 – 0x5FFFFFFF	外设 (Peripheral)	所有外设寄存器区；代码不能再此区域执行
0x60000000 – 0xDFFFFFFF		不用
0xE0000000 – 0xE00FFFFF	私有外设总线区 (PPB)	CPU 内核内的外设寄存器
0xE0100000 – 0xFFFFFFFF	芯片（Device）	PSoC4 专属空间

PSoC4 的地址映射如下表 6-2。

表 6-2. PSoC4 地址映射

地址范围	使用
0x00000000 – 0x00007FFF	32KB 闪存
0x10000000 – 0x10000FFF	4KB 特权闪存 4KB
0x20000000 – 0x20000FFF	4KB 静态存储器（SRAM）
0x40000000 – 0x4000FFFF	CPU 子系统寄存器
0x40010000 – 0x40010FFF	IO 端口控制（高速 IO 矩阵）寄存器
0x40020000 – 0x4002FFFF	可编程时钟寄存器
0x40040000- -0x4004FFFF	端口寄存器
0x40050000- -0x40050FFF	计时器/计数器/脉宽调制器（TCPWM）寄存器
0x40060000- -0x4006FFFF	串行通讯模块(SCB)寄存器
0x40080000- -0x4008FFFF	电容感应寄存器
0x40090000- -0x4009FFFF	LCD 寄存器
0x400A0000- -0x400AFFFF	低功耗比较器寄存器
0x400B0000- -0x400BFFFF	电源、时钟、重启控制寄存器
0x400F0000- -0x400FFFFF	通用数字模块控制寄存器
0xE0000000 – 0xE00FFFFF	Cortex-M0 私有外设总线寄存器
0xF0000000 – 0xF0000FFF	Coresight 只读寄存器

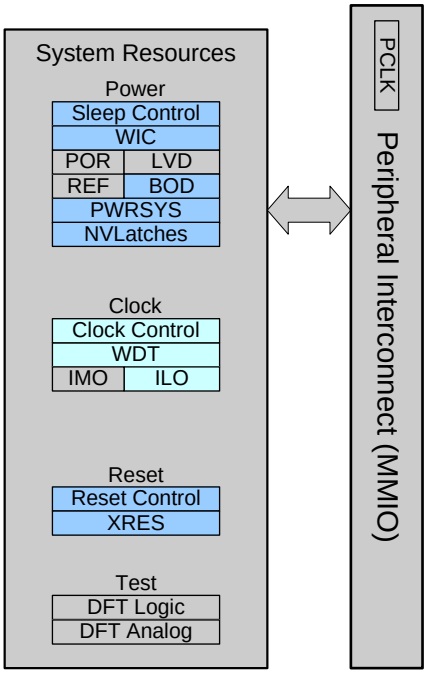
D 部分：系统资源

这部分包括以下章节：

- 输入/输出系统　第 45 页
- 时钟系统　第 54 页
- 电源和电源监测　第 60 页
- 运行模式　第 66 页
- 看门狗定时器　第 73 页
- 复位　第 76 页
- 器件安全　第 79 页

系统架构：

系统资源结构框图



7 输入/输出系统

输入/输出（I/O）系统为 CPU 内核与外部设备通讯提供了接口。PSoC®4 器件具有很高的灵活性，其输入/输出系统可以将一些信号路由至任何管脚，从而简化了电路设计和 PCB 布线。一些关键模块具有专用管脚和有限的路由能力，但建议用户使用专用管脚以提高性能。在 PSoC 器件中，I/O 管脚分为两类：通用输入输出（GPIO）和特殊输入输出（SIO）；但在 PSoC 4 中，没有 SIO，所以 I/O 仅指 GPIO。

在 PSoC4 中，GPIO 被分配到不同的端口，每个端口最多包含 8 个管脚。一些具有特殊功能的 GPIO 是复用管脚，用户可通过寄存器来配置管脚的功能。所有 I/O 管脚不但可作为 CPU 和数字外设的数字输入和输出端口，还可以产生中断。并且每个 GPIO 管脚均可被用于模拟输入、CapSense 和段式 LCD 直接驱动。

7.1 特性

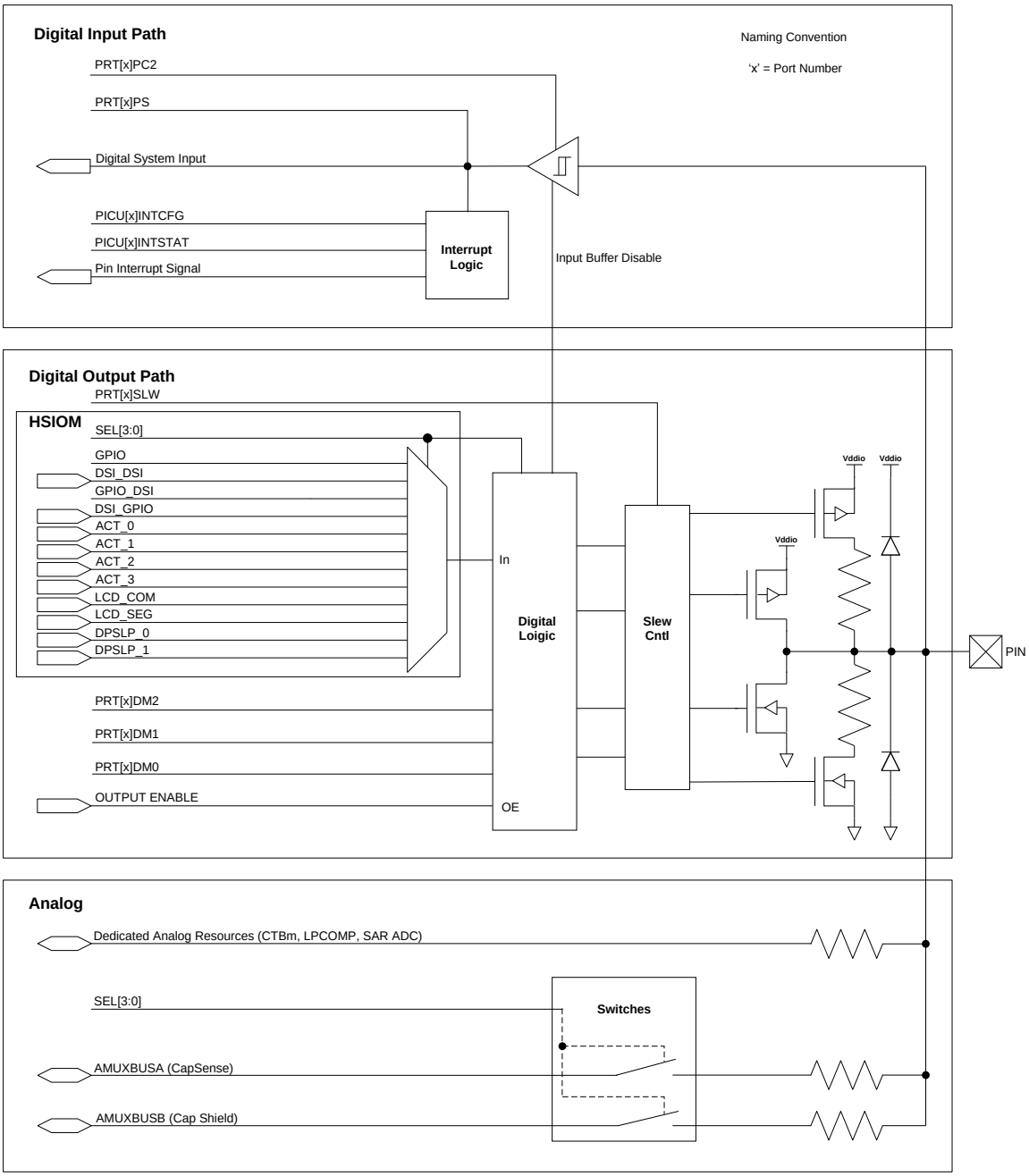
PSoC 4 的 I/O 系统具有如下特性：

- 支持模拟/数字输入输出
- 支持段式 LCD 直接驱动
- 支持 CapSense
- 灌电流可达 8 mA，拉电流可达 4 mA
- 独立的端口读寄存器（PRTx_PS）和写寄存器（PRTx_DR）
- 支持边沿触发中断，包括上升沿、下降沿和双边沿等触发方式。
- 摆率可控
- 可选择输入阈值：CMOS 输入或 LVTTL 输入

7.2 GPIO 模块框图

图 7-1 描述了驱动 GPIO 的各种模块和信号。

图 7-1. GPIO 模块框图



7.3 I/O 驱动模式

PSoC 4 的每个 GPIO 可被配置成表 7-1 中 8 种驱动模式的任何一种。图 7-2 是对每种模式管脚视图的简单描述。

用户可以通过寄存器 PRTx_PC 和寄存器 PRTx_PC2 来配置 PSoC 4 的管脚。每个端口均有对应的寄存器 PRTx_PC（见表 7-2）和 PRTx_PC2（见表 7-3）。

图 7-2. GPIO 驱动模式图

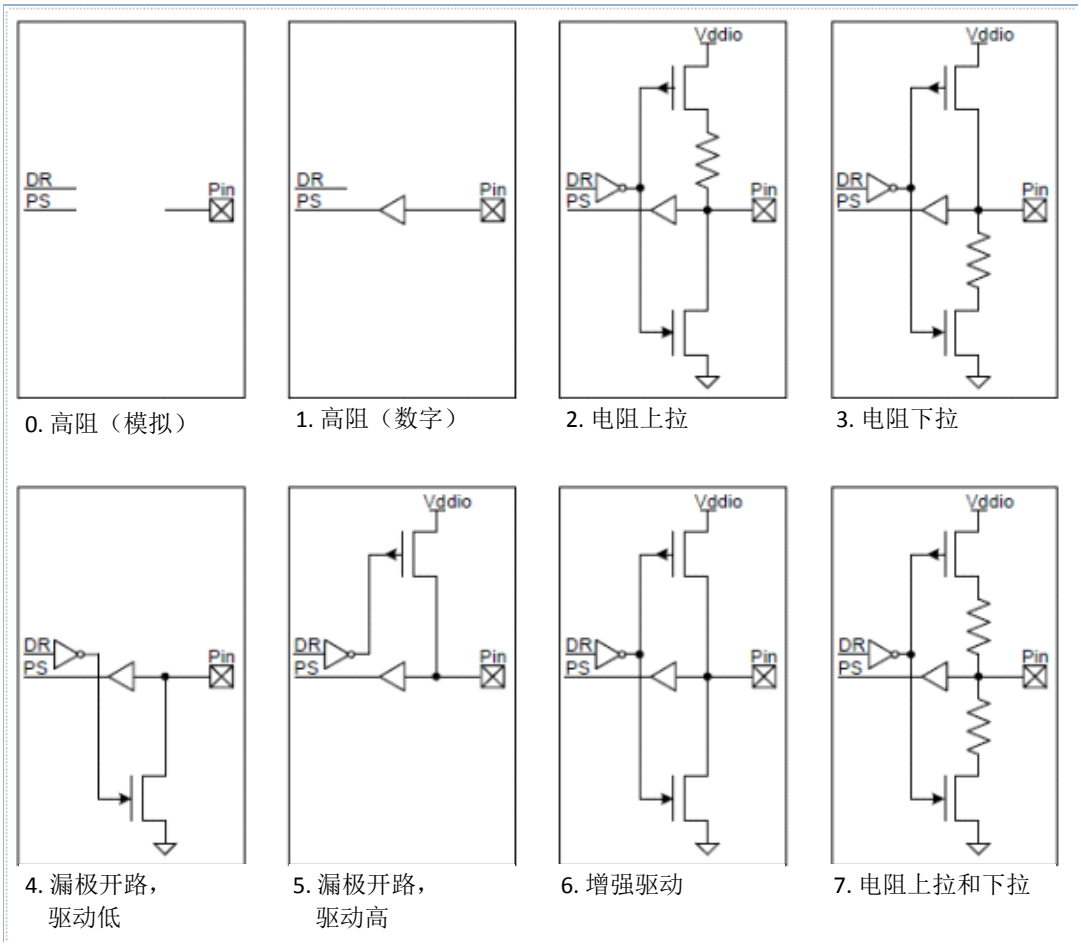


表 7-1 GPIO 的 8 种驱动模式（x 表示端口编号）

编号	驱动模式	DM[3y+2 : 3y]（PRTx_PC[3y+2 : 3y]）y: 管脚编号 （0 ≤ y ≤ 7）	Data = 1（向寄存器 PRTx_DR 中相应位写 1）	Data = 0（向寄存器 PRTx_DR 中相应位写 0）
0	高阻（模拟）	0	高阻（High Z）	高阻（High Z）
1	高阻（数字）	1	高阻（High Z）	高阻（High Z）
2	电阻上拉	2	1（5k 欧姆上拉电阻）	0（增强驱动型）
3	电阻下拉	3	1（增强驱动型）	0（5k 欧姆下拉电阻）
4	漏极开路，驱动低	4	高阻（High Z）	0（增强驱动型）
5	漏极开路，驱动高	5	1（增强驱动型）	高阻（High Z）
6	增强驱动	6	1（增强驱动型）	0（增强驱动型）
7	电阻上拉和下拉	7	1（5k 欧姆上拉电阻）	0（5k 欧姆下拉电阻）

表 7-2 寄存器 PRTx_PC

PRTx_PC 的位	描述
DM （PRTx_PC[23:0]）	配置每个 GPIO 管脚的驱动模式。每 3 个位对应一个管脚：DM[2..0]对应管脚 0；DM[5..3]对应管脚 1，…，DM[23..21]对用管脚 7
VTRIP_SEL （PRTx_PC[24]）	设置管脚的输入缓冲器电压是 CMOS 电平还是 LVTTL 电平（详情请见 7.5 CMOS 和 LVTTL 电平控制）。
SLOW （PRTx_PC[25]）	控制每个端口的输出摆率（详情请见 7.4 摆率控制）： 0: 快速。 1: 慢速。

表 7-3 寄存器 PRTx_PC2

PRTx_PC2 的位	描述
INP_DIS （PRTx_PC2[7:0]）	每个位对应端口的一个管脚，可用来禁用管脚的输入缓冲器，其与 DM（PRTx_PC[23:0]）相对独立。 当在某管脚上传输模拟信号，并且管脚被配置为非高阻模拟模式做为数字输出时，此位必须设置为 1，禁用输入缓冲器。

7.3.1 高阻模拟模式

高阻模拟模式（High-impedance Analog mode）是 GPIO 管脚的默认复位状态；在此模式下，数字输出驱动器和数字输入缓冲器均被关闭。从而可防止管脚上的浮动电压的电流流入到 I/O 数字输入缓冲器中。此驱动模式适用于有浮动电压或支持模拟电压的 GPIO 管脚，但不能用于数字输入管脚。如果读取高阻模拟管脚的状态值，则返回值总是 0x00。

如需使器件的电流最低，配置为驱动模式为 0、2~7（见表 7-1）的 GPIO 不能由外部驱动，而配置为驱动模式 1（见表 7-1）的 GPIO 应该由外部的 V_{ddd} 或 V_{ssd} 驱动。

7.3.2 高阻数字模式

高阻数字模式（High-impedance Digital mode）是常用的的高阻（High Z）状态,常被用于数字输入管脚。在这种模式下，被使能的输入缓冲器用于数字信号输入。

7.3.3 电阻上拉或电阻下拉模式

配置为电阻上拉模式（Resistive Pull-Up mode）的管脚在输出高电平时串联一个上拉电阻，在输出低电平时是增强驱动型；配置为电阻下拉模式（Resistive Pull-Down mode）的管脚与之相反（见图 7-2）。配置为此种模式的管脚可用于数字输入和输出，比如对机械开关控制的应用。如需对配置为电阻上拉模式的管脚进行上拉，则必须向数据寄存器（PRTx_DR）的相应位写 1。如需对配置为电阻下拉模式的管脚进行下拉，则必须向数据寄存器（PRTx_DR）的相应位写 0。

7.3.4 漏极开路驱动高和漏极开路驱动低

配置为漏极开路驱动高模式（Open Drain，Drives High mode）的管脚在输出为高电平时是增强驱动型，在输出低电平时是高阻态；配置为漏极开路驱动低模式（Open Drain，Drives Low mode）的管脚与之相反（见图 7-2）。配置为此种模式的管脚可用于数字输入和输出，比如用于驱动 I^2C 总线信号线。

7.3.5 增强驱动模式

增强驱动模式（Strong Drive mode）是管脚通用的数字输出模式。配置为此模式的管脚在输出高电平和低电平时均提供了一个增强性 CMOS 的输出驱动。配置为增强驱动模式的管脚常被用于驱动数字输出信号或外部的场效应晶体管，但不可被用于常用电路的数字输入。

7.3.6 电阻上拉和下拉模式

电阻上拉和下拉模式（Resistive Pull-Up and Pull-Down mode）是一个特殊模式，其与电阻上拉模式和电阻下拉模式类似，但是配置为此模式的管脚无论输出高还是低总是串联一个电阻。当管脚输出高电平时，使用上拉电阻；当输出低电平时，使用下拉电阻。当挂载的总线在其他设备驱动下可能导致短路时，可将管脚配置为此模式。

7.4 摆率控制

管脚的摆率控制是针对增强驱动模式，而不是电阻驱动模式，其分为快速和慢速摆率。其中快速摆率应用于 1 MHz 到 33 MHz 间的信号，慢速摆率应用于小于 1 MHz 的信号。为了降低电磁干扰（EMI），对于翻转速度要求不高的信号（摆率小于 1 MHz），可以选择使用慢速摆率。

端口的默认摆率为快速摆率，用户可通过 SLOW 位（PRTx_PC[25]）重新配置每个端口的摆率。

7.5 CMOS 和 LVTTL 电平控制

每个 GPIO 管脚的输入缓冲器电压可工作于两种状态：CMOS 电平和 LVTTL 电平。其可通过 VTRIP_SEL 位（PRTx_PC[24]）来配置。

用户也可通过寄存器 PRTx_PC2 来禁用输入缓冲器，其独立于管脚驱动模式（见表 7-3）。

7.6 高速输入输出矩阵

高速输入输出矩阵（High Speed IO Matrix，HSIOM）是一个高速矩阵，其中有一个可用于不同的功耗模式的多路复用器。HSIOM 不但可以连接各种模拟信号（如来自 AMUXBUS 和 CapSense 的信号），还可以连接数字信号（如 TCPWM、SCB 或 LCD 控制器）；此外，它还支持活动/深度睡眠模式下的管脚（见表 7-6）。

注意：端口 4 对 DSI 路由有限制，所以 DSI 路由配置对端口 4 无效，详情请见 7.17 端口 4 的使用限制。

32 位寄存器 HSIOM_PORT_SELx 中的每 4 个位对应一个管脚，可为每个管脚提供 16 种不同的连接功能选择（见表 7-4）。其连接的模块包括：

- GPIO
- 固定功能外设
- LCD 控制器
- 串行调试（SWD）
- 数字系统互连接口（DSI）
- CapSense 控制器
- 外部时钟输入
- WAKEUP 管脚

部分模块，如 SCB、SWD 等，有专用管脚，见表 7-5。在应用设计过程中，用户可使用专用管脚以提高性能。具体管脚定义请见器件数据手册。

表 7-4 HSIOM 端口配置

HSIOM_PORT_SELx（x:端口编号； y：管脚编号）			
位	名称	数值	描述
4y+3 : 4y	SEL'y'		选择管脚 y。
	GPIO	0	管脚由软件控制或连接到专用硬件模块
	GPIO_DSI	1	管脚输出由软件控制，但输出使能（OE）是来自 DSI 的信号控制
	DSI_DSI	2	管脚输出和输出使能（OE）均是来自 DSI 的信号控制
	DSI_GPIO	3	管脚输出是由来自 DSI 的信号控制，但是输出使能（OE）由软件控制
	CSD_SENSE	4	管脚作为 CapSense 的 sense 管脚（高阻模拟模式）
	CSD_SHIELD	5	管脚作为 CapSense 的 shield 管脚（高阻模拟模式）
	AMUXA	6	管脚被连接到 AMUXBUS_A
	AMUXB	7	管脚被连接到 AMUXBUS_B。这种模式也可用于为 CapSense 管脚充电。当用户通过寄存器 CSD_CONTROL 使能了对 CapSense 管脚的充电功能时，数字 IO 驱动器被连接到信号 csd_charge（同时该管脚仍然被连接到 AMUXBUS_B）
	ACT_0	8	活动模式源的专用管脚 #0（见表 7-6）
	ACT_1	9	活动模式源的专用管脚 #1（见表 7-6）
	ACT_2	10	活动模式源的专用管脚 #2（保留）
	ACT_3	11	活动模式源的专用管脚 #3（保留）
	LCD_COM	12	该管脚是段式 LCD 的公共端（COM）管脚。当 LCD 模块被使能并正确地配置后，此管脚在深度睡眠模式下仍可用。
	LCD_SEG	13	该管脚是段式 LCD 的段（SEG）管脚。当 LCD 模块被使能并正确地配置后，此管脚在深度睡眠模式下仍可用。
	DPSLP_0	14	睡眠模式源的专用管脚 #0
	DPSLP_1	15	睡眠模式源的专用管脚 #1

表 7-5. 专用管脚

功能模块	信号名称	信号类型	管脚编号	备注
外部参考电压	ext_vref	模拟	P1[7]	为 SAR ADC 提供外部参考电压
运算放大器 0	ctb.oa0.out	模拟	P1[2]	CTBm 模块运算放大器 0 的输出
运算放大器 1	ctb.oa1.out	模拟	P1[3]	CTBm 模块运算放大器 1 的输出
外部时钟	exe_clk	数字	P0[6]	提供可选择的外部时钟，用来代替 IMO。
I2C	SCL_0	数字	P4[0]	
	SDA_0	数字	P4[1]	
	SCL_1	数字	P0[4]/P3[0]	任选其一
	SDA_1	数字	P0[5]/P3[1]	任选其一
SPI	MOSI_0	数字	P4[0]	
	MISO_0	数字	P4[1]	
	CLK_0	数字	P4[2]	
	SSEL0_0	数字	P4[3]	
	SSEL1_0	数字	P0[0]	
	SSEL2_0	数字	P0[1]	
	SSEL3_0	数字	P0[2]	
	MOSI_1	数字	P0[4] / P3[0]	任选其一
	MISO_1	数字	P0[5] / P3[1]	任选其一
	CLK_1	数字	P0[6] / P3[2]	任选其一
	SSEL0_0	数字	P0[7] / P3[3]	任选其一
	SSEL1_1	数字	P3[4]	
	SSEL2_2	数字	P3[5]	
	SSEL3_3	数字	P3[6]	
SWD	IO	数字	P3[2]	
	CLK	数字	P3[3]	
WAKEUP	WAKEUP	数字	P0[7]	

7.7 模拟管脚

外部模拟信号仅可通过 GPIO 管脚与 PSoC 内核进行通讯。在通常的应用中，用户应该将模拟 I/O 管脚配置为高阻模拟驱动模式，在此模式下，输入缓冲器被禁用。此外，用户也可通过配置寄存器 PRTx_PC2 来禁用管脚的输入缓冲器。如果某 GPIO 管脚被同时用作数字输出和模拟输入/输出时，输入缓冲器应该被禁用。管脚的功能选择和驱动模式配置见表 7-4 和表 7-1。

模拟 GPIO 管脚可通过 AMUXBUS_A/AMUXBUS_B 连接至相应模拟模块。但部分模拟模块（SAR ADC、CTBm 和低功耗比较器）具有专用管脚（见表 7-5），其经过内部电阻，直接连接至专用管脚（见图 7-1）。建议用户在设计时优先使用专用管脚来提高性能。

7.8 段式 LCD 驱动

每个 GPIO 管脚均具有段式 LCD 驱动能力。用户可通过寄存器 HSIOM_PORT_SELx 来配置该管脚为 LCD 的公共端管脚还是段驱动管脚（见表 7-4），从而驱动 LCD 显示。如需 LCD 的详细信息，请参考 LCD 章节

7.9 CapSense

CapSense 为模拟模块，用户可以使用任意一个 GPIO 管脚作为 CapSense 的管脚，来创建按钮或滑条（见表 7-4）。如需 CapSense 的详细信息，请参考 CapSense 章节。

7.10 重新配置 GPIO

用户可通过寄存器 HSIOM_PORT_SELx 和 PRTx_PC 来重新配置 GPIO 功能（x：端口编号）。但必须注意的是，当器件管脚被直接连接到外部数字外设时，在管脚的重新配置期间， 管脚必须保持当前配置状态。如果管脚由寄存器控制、驱动，则在配置过程中，管脚状态会自动被保持。但是，如果 GPIO 管脚由 DSI 控制、驱动；则在重新配置前，管脚的当前配置值必须被读、并写入寄存器 PRTx_DR 中。用户可通过如下步骤来保持管脚的当前配置状态：

1. 软件读 GPIO 管脚状态寄存器 PRTx_PS。
2. 将此值写入寄存器 PRTx_DR。
3. 配置寄存器 HSIOM_PORT_SELx 的相应位，使该管脚由寄存器控制、驱动。
4. 修改相应寄存器，重新配置管脚状态。

通过如上步骤，用户可安全地重新配置器件管脚状态。当重现配置完成时，通过修改寄存器 HSIOM_PORT_SELx，使该管脚再由 DSI 控制、驱动。

7.11 上电启动时 GPIO 状态

所有 I/O 在上电启动时被设置为高阻模拟驱动模式，输入缓冲器被禁用。当上电启动结束后，GPIO 的驱动模式可以通过相应寄存器来配置。

7.12 睡眠模式下的 GPIO 状态

在 CPU 进入睡眠模式后，GPIO 仍保持当前的状态；并且所有 GPIO 均处于活动模式，可以被活动的外设驱动。

7.13 低功耗模式下的 GPIO 状态

在深度睡眠模式下（见表 7-4），仅有被连接到仍然处于活动状态模块的专用管脚可以正常工作。

在进入休眠和停止模式前，建议用户通过寄存器将 GPIO 输出设置为冻结（Frozen）状态；则在休眠或停止模式下，GPIO 的配置、模式及状态处于锁存状态。在器件进入活动模式前，无法改变 GPIO 的状态

如需电源模式的详细信息，请参考电源模式章节。

7.14 端口中断控制器模块

本节将介绍 PSoC 4 端口中断控制器模块（PICU）。用户可以通过寄存器 PRTx_INTCFG 和 PRTx_INTSTAT 来分别配置中断和读取中断状态。

7.14.1 特性

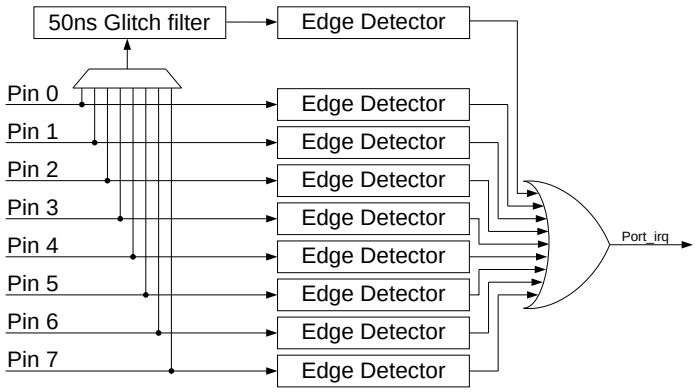
端口中断控制器模块具有如下特性：

- 每个端口的 8 个管脚分别具有独立的端口中断控制器，并与同一个中断向量关联。
- 支持上升沿/下降沿/双边沿中断
- 每个管脚中断可被独立的使能或禁止
- 可向中断控制器发送中断请求（PIRQ）信号。

7.14.2 端口中断控制器框图

每个端口都有独立的中断请求和相应的中断向量。此外，每个管脚均还可被选择作为通过 50ns 干扰过滤器的第 9 个中断，见图 7-3。

图 7-3. 管脚中断框图



7.14.3 功能和配置

通过寄存器 PRTx_INTCFG，用户可以将端口的每个管脚配置为以下 3 种不同的边沿中断触发方式：

- 禁用
- 上升沿
- 下降沿
- 双边沿 （上升沿或下降）

注意：端口管脚不支持电平中断。

同时也可以选择其中一个管脚作为第 9 个中断（信号会经过 50ns 干扰滤波器，见图 7-3）。

当一个端口中断被某个使能管脚上的信号改变触发时，软件可通过读取寄存器 PRTx_INSTAT 来判断具体由那个管脚触发的中断。建议用户使用如下步骤来编写中断服务程序：

1. 当所配置的管脚边沿触发模式发生时，端口中断状态寄存器的相应位会被置 1，一个中断请求会被发送至中断控制器。
2. 包含 1 的状态位在读取后会被清除。状态寄存器的其它位仍然可以响应进来的中断源。
3. 如果一个中断处于挂起状态，当软件读取中断状态寄存器时，即将进入同一个管脚上的中断将被阻塞，直到读取完成后才有效。但是其他没有挂起中断的管脚，其上的中断不会被阻塞。

此外，当读寄存器 PRTx_INSTAT 和此端口的中断同时发生时，可能导致无法检测到中断。所以，在使用端口中断时，建议用户仅在相应的中断服务程序中读取端口中断状态寄存器，以防止丢失端口中断。

7.15 数字输入和输出同步

对于数字输入和输出信号，GPIO 支持使用内部时钟或数字信号对其进行时钟同步。默认情况下，使用 HFCLK 来同步数字输入和输出信号。

此功能是通过 UDB 端口调试器和 GPIO 的相互结合来完成的，详情请 参考 16.3 端口适配模块。

7.16 活动和深度睡眠模式下专用管脚

正如在 7.6 高速输入输出矩阵所介绍的，每个 IO 可配置为 16 种不同功能，包括 4 个活动模式源（活动模式源的专用管脚 #2/#3 在 PSoC 4 中被保留）和 2 个深度睡眠模式源专用管脚，见表 7-6。关于活动模式和深度睡眠模式的详细介绍，请参考电源模式章节。

表 7-6 活动模式和深度睡眠模式下的专用管脚

管脚	活动模式源的专用管脚 #0*	活动模式源的专用管脚 #1	睡眠模式源的专用管脚 #0	睡眠模式源的专用管脚 #1
P0[0]				scb0_spi_ssel_1
P0[1]				scb0_spi_ssel_2
P0[2]				scb0_spi_ssel_3
P0[4]		scb1_uart_rx	scb1_i2c_scl	scb1_spi_mosi
P0[5]		scb1_uart_tx	scb1_i2c_sda	scb1_spi_miso
P0[6]	ext_clk			scb1_spi_clk
P0[7]			wakeup	scb1_spi_ssel_0
P1[0]	tcpwm2_p			
P1[1]	tcpwm2_n			
P1[2]	tcpwm3_p			
P1[3]	tcpwm3_n			
P2[4]	tcpwm0_p			
P2[5]	tcpwm0_n			
P2[6]	tcpwm1_p			
P2[7]	tcpwm1_n			
P3[0]	tcpwm0_p	scb1_uart_rx	scb1_i2c_scl	scb1_spi_mosi
P3[1]	tcpwm0_n	scb1_uart_tx	scb1_i2c_sda	scb1_spi_miso
P3[2]	tcpwm1_p		swd_io	scb1_spi_clk
P3[3]	tcpwm1_n		swd_clk	scb1_spi_ssel_0
P3[4]	tcpwm2_p			scb1_spi_ssel_1
P3[5]	tcpwm2_n			scb1_spi_ssel_2
P3[6]	tcpwm3_p			scb1_spi_ssel_3

管脚	活动模式源的专用管脚 #0*	活动模式源的专用管脚 #1	睡眠模式源的专用管脚 #0	睡眠模式源的专用管脚 #1
P3[7]	tcpwm3_n			
P4[0]		scb0_uart_rx	scb0_i2c_scl	scb0_spi_mosi
P4[1]		scb0_uart_tx	scb0_i2c_sda	scb0_spi_miso
P4[2]				scb0_spi_clk
P4[3]				scb0_spi_ssel_0

*通过 DSI，TCPWM 可以路由至除端口 4 外的任何管脚，但是仍建议用户使用上表中的管脚，以提高性能。

7.17 端口 4 的使用限制

因端口 4 没有专用的端口适配器（见 16.3 端口适配模块），所以其不能通过 DSI 被路由。但是通过 HSIOM，端口 4 仍然可以用作软件可控的 GPIO、LCD_COM、LCD_SEG 或连接到 SCB 模块。

7.18 寄存器列表

名称	数量	描述
PRTx_DR	4	端口输出数据寄存器
PRTx_PS	4	端口管脚状态寄存器，用来读取 GPIO 的当前状态
PRTx_PC	4	端口配置寄存器，用来配置驱动模式、输入阈值和摆率
PRTx_PC2	4	配置 GPIO 输入缓冲器的端口配置寄存器
PRTx_INTCFG	4	端口中断配置寄存器
PRTx_INTSTAT	4	端口中断状态寄存器
HSIOM_PORT_SELx	4	HSIOM 端口选择寄存器

8 时钟系统

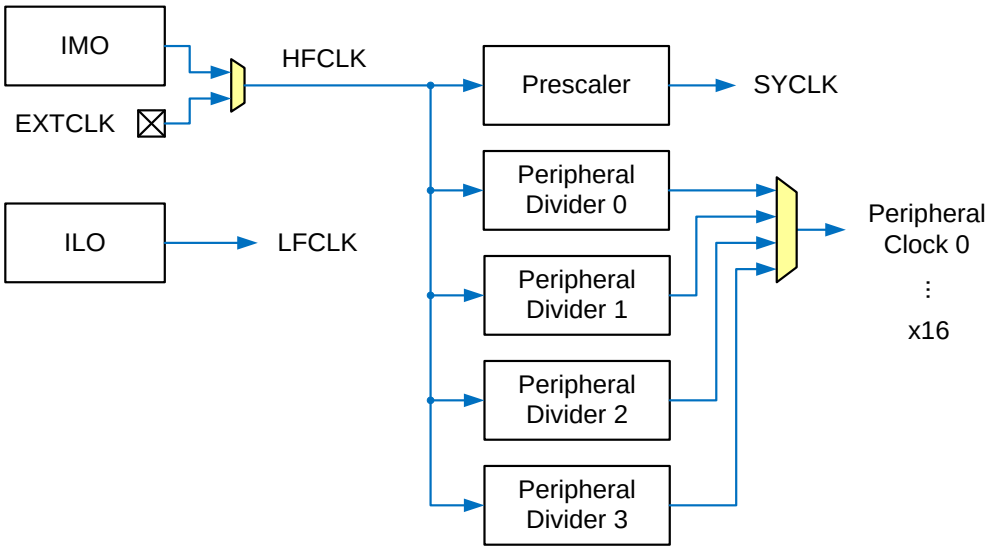
PSoC 4 的时钟系统主要特性如下：

- 两个内部时钟源：
  - 内部主振荡器（IMO）：3-48MHz 可配置，配置步长为 1MHz，经过校准调整，精度为+/-2%；
  - 内部低速振荡器（ILO）：32KHz，进过校准调整，精度为+/-60%。
- 可选择外供时钟源（EXTCLK），通过一个 GPIO 的管脚输入，具体管脚信息请查询相关器件手册。外供时钟的频率范围是 1-24MHz。
- 高频时钟（HFCLK）由 IMO 或者外供时钟源分频得到；
- 低频时钟（LFCLK）由 ILO 获得；
- 系统时钟 SYSCLK 由专用的预分频器对 HFCLK 分频后获得；
- 四组时钟分频器，每组分频器含三个可级联的 16-bit 分频器，所有的外设时钟由这些分频器提供；
- 16 个数字或者模拟的外设时钟；

8.1 模块框图

图 8-1 给出了 PSoC 4 的时钟系统框图。

图 8-1 时钟系统框图



框图中的左半部分给出了三个时钟源，包括两个内部的时钟源 IMO 和 ILO，以及一个通过 GPIO 管脚接入的外供时钟源 EXTCLK，一个多路器被用于选择 IMO 或者 EXTCLK 作为内部的高频时钟 HFCLK。右半部分给出了时钟输出的情况，SYSCLK 和各个外设时钟通过分频器输出。

8.2 时钟源

8.2.1 内部主振荡器（IMO）

IMO 可以输出一个稳定的时钟，不需要外接任何外部器件。用户可以配置和调整该时钟的频率，范围是 3-48MHz，步长为 1MHz。在制造过程中，每个器件都会进行 IMO 调整，以达到器件手册的要求，相应的调整参数被存放在 SFLASH 中。

8.2.1.1 启动过程中的 IMO 调整

启动开始阶段，IMO 初始频率为 24MHz。在启动过程中的后续阶段，引导程序从 SFLASH 中获取调整参数，将 IMO 校准到器件手册中的精度。

用户对 IMO 的调整

如果用户需要对 IMO 进行调整，需要进行以下的操作：

- 选择 IMO 的频率：配置 FREQ（CLK_IMO_TRIM2[5:0]）；
- 对 IMO 进行绝对频率调整：根据选择的 IMO 频率，将 SFLASH 中相应的绝对频率调整参数 IMO_TRIM[x]（x = 3,4,5,…,48）读入寄存器 OFFSET（CLK_IMO_TRIM1[7:0]）；
- 调整 IMO 的基准源（工作原理请参见“精确基准源”章节）：根据选择的 IMO 频率，相应的将 SFLASH 中电流基准源调整参数 IMO_ABSx（x = 0,1,2,3）读入寄存器 ABS_TRIM_IMO（PWR_BG_TRIM4[5:0]）；温度补偿系数 IMO_TMPCOx（x = 0,1,2,3）读入寄存器 TMPCO_TRIM_IMO（PWR_BG_TRIM4[5:0]）；

注意：如果 IMO 的频率在调整过程中超过工作频率的上限 48MHz，器件存在系统崩溃的风险。当 IMO 从 48MHz 调整到较低的频率时，FREQ 配置应放在第一个步骤；当 IMO 从较低的频率调整到 48MHz 时，FREQ 配置应放在最后一个步骤。这样可以规避上述的风险。

8.2.1.2 IMO 的频谱扩展

通过频谱扩展，能够降低 IMO 的输出在中心频点的能量，从而降低器件的 EMI 水平。在频谱扩展时，IMO 的频率在一定范围内按照一定的模式变化。变化模式由寄存器 SS_MODE（CLK_IMO_SPREAD[31:30]）决定，具体见表 8-1；频谱扩展范围由寄存器 SS_RANGE（CLK_IMO_SPREAD[29:28]）决定，具体见表 8-2。

表 8-1 IMO 频谱扩展的模式配置

寄存器名	描述
SS_MODE(CLK_IMO_SPREAD[31:30])	IMO扩频模式： 0：禁止IMO扩频； 1：IMO的频率在扩频范围内以三角曲线变化； 2：IMO的频率在扩频范围内以伪随机的方式变化； 3：IMO的频率在扩频范围内的变化由DSI信号控制；

表 8-2 IMO 频谱扩展的范围配置

寄存器名	描述
SS_RANGE (CLK_IMO_SPREAD[29:28])	IMO扩频范围： 0：扩频范围为0…-1% 1：扩频范围为0…-2% 2：扩频范围为0…-4% 3：预留，未使用

8.2.2 内部低速振荡器（ILO）

PSoC 4 内部提供的低速振荡器可以输出固定的 32KHz 的时钟，不需要外接任何的器件。相对于 IMO，ILO 具有较低的功耗，可以在除休眠和停止以外的电源模式中工作，但是精度也较低。ILO 输出的时钟在 PSoC 4 内部被称为 LFCLK。通过配置寄存器 ENABLE（CLK_ILO_CONFIG[31]），可以使能或者禁止 ILO。

8.2.3 外部时钟（EXTCLK）

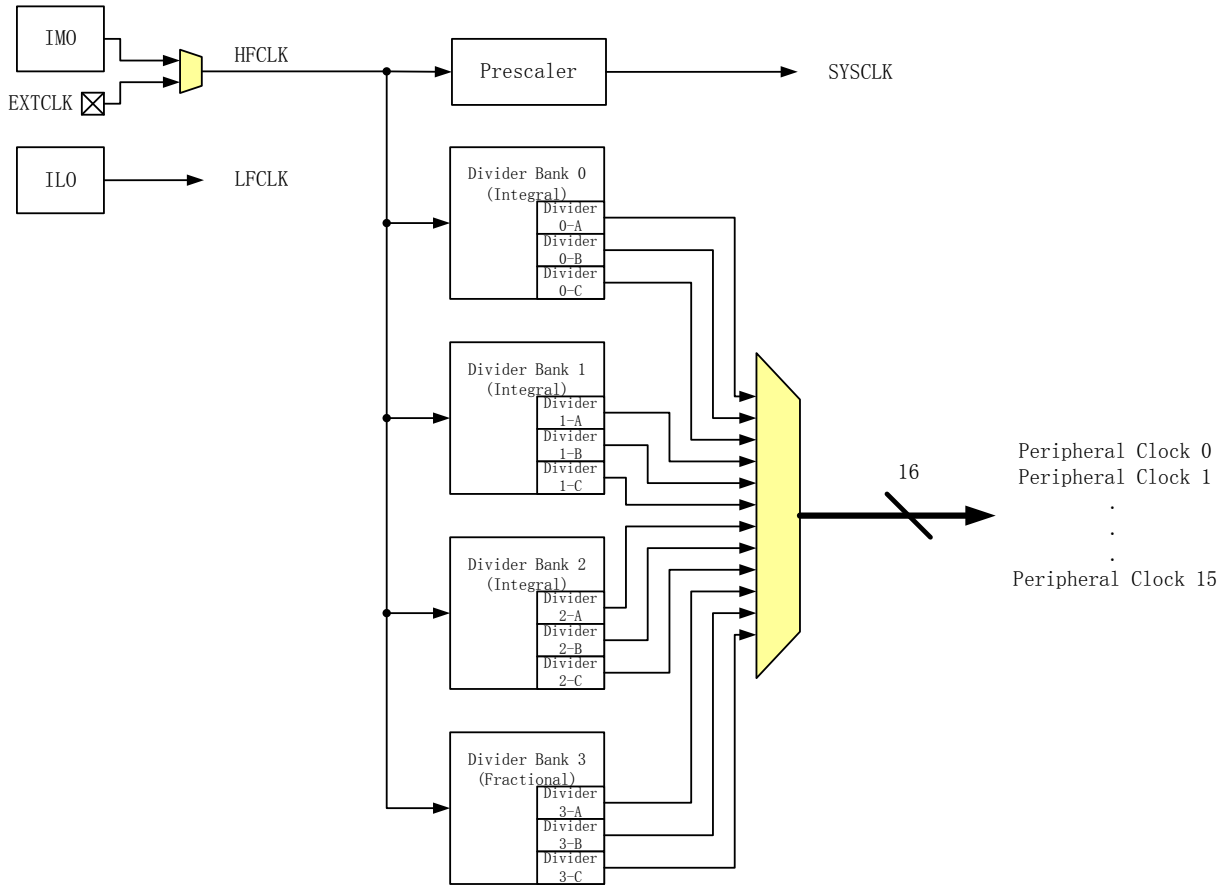
通过 PSoC 4 的一个 GPIO 管脚，可以输入一个外部时钟（EXTCLK），它的频率范围是 0MHz 到 48MHz。EXTCLK 能够替代 IMO 作为内部高频时钟（HFCLK）的时钟源。PSoC 4 在启动过程中，总是使用内部时钟源 IMO，用户可以在启动之后将时钟源切换为 EXTCLK。

8.3 时钟分布

PSoC 4 内部的时钟分布情况可以从图 8-2 中描述的模块框图中得到。用户可以配置的时钟分布选项如下：

- HFCLK 时钟输入源的选择；
- SYSCLK 分频器的配置；
- 外设时钟分频器的配置；

图 8-2 Clock Distribution System



8.3.1 HFCLK 输入时钟源的选择

在 PSoC 4 中，内部高频时钟由两个源可供选择，一个是 IMO，另一个是 EXTCLK。通过配置寄存器 DIRECT_SEL（CLK_SELECT[2:0]）可以进行选择，具体如表 8-3 所示。

表 8-3 HFCLK 输入时钟源的选择

寄存器名	描述
DIRECT_SEL CLK_SELECT[2:0]	HFCLK输入时钟源的选择 0：IMO作为HFCLK的时钟源 1：EXTCLK作为HFCLK的时钟源 2-7：预留，未被使用。

8.3.2 SYSCLK 分频器的配置

时钟源输入的 HFCLK 经过分频器可以得到系统时钟 SYSCLK。SYSCLK 的时钟频率需要大于或者等于各个外设的时钟频率。通过配置寄存器 SYSCLK_DIV（CLK_SELECT[21:19]），选择该分频器的分频比，具体如表 8-4。

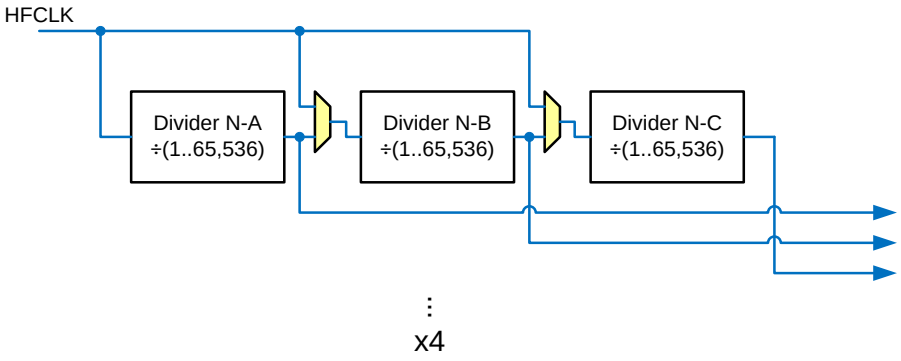
表 8-4 SYSCLK Prescaler Divide Value Bits SYSCLK_DIV

寄存器名	描述
SYSCLK_DIV（CLK_SELECT[21:19]）	0：SYSCLK = HFCLK 1：SYSCLK = HFCLK / 2 2：SYSCLK = HFCLK / 4 3：SYSCLK = HFCLK / 8 4：SYSCLK = HFCLK / 16 5：SYSCLK = HFCLK / 32 6：SYSCLK = HFCLK / 64 7：SYSCLK = HFCLK / 128

8.3.3 外设时钟分频器的设置

PSoC 4 共有四组分频器，每组包含 3 个 16-bit 的分频器（A，B 和 C），可级联使用，它的模块框图如图 8-3 所示。四组分频器中，三组是整数分频器，另一组是分数分频器。PSoC 4 所有的外设时钟，包括模拟时钟与数字时钟，均由这四组分频器输出。

图 8-3 外设时钟分频器的模块框图



对于整数分频器组，通过设置寄存器 DIVIDER_A，DIVIDER_B 和 DIVIDER_C ，可以选择分频比。对于分数分频器组，还需要配置额外的寄存器 FRAC_A，FRAC_B 和 FRAC_C。通过配置寄存器 CASCADE_x-1_x，可以控制相邻的分频器是否进行级联。配置寄存器 ENABLE_x 可以使能或者禁止相应的寄存器。（x = A，B 或 C）。具体寄存器如表 8-5 和表 8-6。

表 8-5 整数分频器组中分频器的配置

比特位	寄存器名	描述
15:0	DIVIDER_x	分频器组中分频器x的整数分频系数 输出频率=输入频率/ (DIVIDER_x +1)
30	CASCADE_x-1_x	级联控制 0：分频器x的输入为HFCLK 1：分频器x的输入为分频器x-1的输出 **注意：**对于分频器A，该寄存器位无效。分频器A的输入时钟为HFCLK
31	ENABLE_x	0：禁止分频器x； 1：使能分频器x。

表 8-6 分数分频器组中分频器的配置

比特位	寄存器名	描述
15:0	DIVIDER_x	分频器组中分频器x的整数分频系数 输出频率=输入频率/ (DIVIDER_x +1 + FRAC_x/32)
20:16	FRAC_x	分频器组中分频器x的分数分频系数 输出频率=输入频率/ (DIVIDER_x +1 + FRAC_x/32)
30	CASCADE_x-1_x	级联控制 0：分频器x的输入为HFCLK 1：分频器x的输入为分频器x-1的输出 **注意：**对于分频器A，该寄存器位无效。分频器A的输入时钟为HFCLK
31	ENABLE_x	0：禁止分频器x； 1：使能分频器x。

8.3.4 外设时钟分配

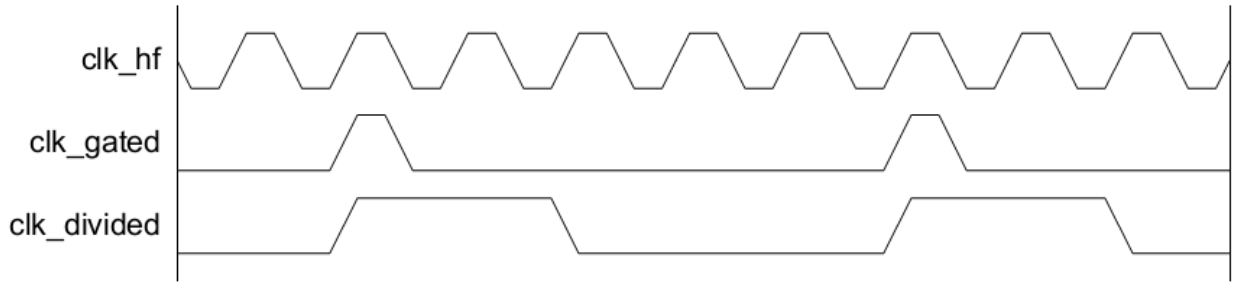
如图 8-1 右半部分所示，分频器组通过多路器分配，为 PSoC 4 提供 16 个外设时钟，包括 UDB、SAR ADC、CSD、TCPWM、LCD 等模块的时钟，如表 8-7 所列。

表 8-7 外设时钟列表

外设时钟号	外设名
0	IMO (SS)
1	SARPUMP
2	SCB0
3	SCB1
4	LCD
5	CSD (1)
6	CSD (2)
7	SAR
8	TCPWM0
9	TCPWM1
10	TCPWM2
11	TCPWM3
12	UDB0
13	UDB1
14	UDB2
15	UDB3

分频器可以提供两种形式的时钟：门控时钟（clk_gated）和分频时钟（clk_divided）。门控时钟的功耗较低，但是占空比无法保证，在一个门控周期中，高脉冲仅持续一个源时钟（clk_hf）周期；分频时钟相对前者功耗较高，但占空比能被保持在最接近 50%的一个值（偶数分频时，占空比等于 50%；奇数分频时，一个周期中的高脉冲持续时间比低脉冲持续时间少一个源时钟周期），如图 8-4 所示。在某些应用中，需要同时用到时钟的上升沿与下降沿，这时推荐使用分频时钟，因为门控时钟的上升沿和下降沿的间隔太短，容易引起时序问题。

图 8-4 门控时钟与分频时钟



PSoC 4 提供 16 组寄存器，如表 8-8。通过配置它，控制 16 个外设时钟分别由哪个分频器组的哪个分频器输出。

表 8-8 外设时钟选择（N = 0,1,2 或 3）

比特位	寄存器名	描述
3:0	DIVIDER_N	选择分频器组 0：分频器组0； 1：分频器组1； 2：分频器组2； 3：分频器组3；
5:4	DIVIDER_ABC	选择分频器组中的分频器 0：禁止该外设时钟； 1：分频器A 2：分频器B 3：分频器C

SAR ADC 的时钟与其他外设的时钟一样来自于外设分频器，但不同的是，它会在时钟沿上做一些处理。SAR ADC 是一个数模混合模块，它会产生两个时钟分别供给模拟采样电路和数字逻辑电路。模拟时钟和数字时钟的上升沿存在一定的偏移，这能够保证模拟电路的抗干扰能力，提高器件的性能。

8.4 低功耗电源模式下的操作

PSoC 4 中，IMO、EXTCLK，HFCLK、SYSCLK 和所有的外设时钟仅在活动和睡眠电源模式下工作；ILO 和 LFCLK 除活动和睡眠外，还能够在深度睡眠的电源模式下工作。

9 电源和电源监测

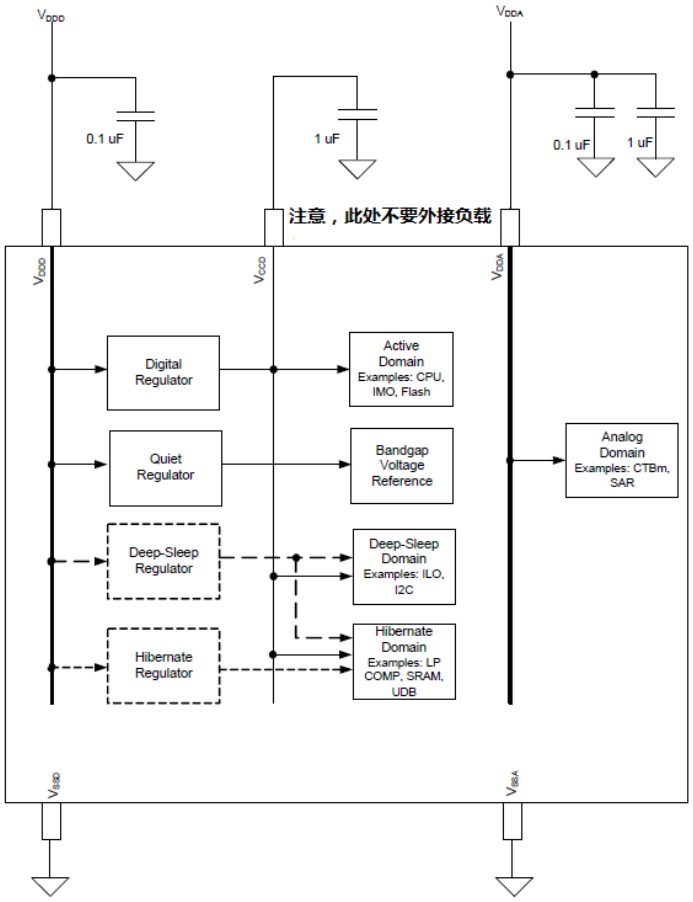
PSoC 4 可以在 1.71-5.5V 的外部电源供电时工作。支持一下两种工作电压范围：

- 1.80V-5.50V 到内部电压调节器
- 1.71V-1.89V 直接输入

芯片内部有四个电压调节器（降压），可以在不同的电源模式下载给内部的电路供电，包括：活动状态电压调节器、低噪声电压调节器、深度睡眠状态电压调节器和休眠状态电压调节器。

9.1 系统框图

图 9-1 电源系统框图



由图 9-1 可见，电源系统有独立的数字电源(Vddd)和模拟电源(Vdda) 输入引脚，同样也有独立的 Vssd 和 Vssa 引脚。在一些较小的封装型号中，数字电源和模拟电源在内部短路到一起，共用一个引脚（Vdd）。**但是注意，在 PSoC4 的系统中，VDDD 和 VDDA 必须相等**（推荐用同一电源供电）。虽然 VDDA 和 VDDD 由同一电源供电，但是在一些对精度要求较高的模拟电路中，推荐将两路分开走线已隔离模拟电路和数字电路的电流。同样的，推荐将 VDDA 和 VSSD 分开走线。如果 VDDA 和 VDDD 由不同的电源供电，VDDA 必须提前于 VDDD。芯片内部有四个个电压调节器（降压），都由 Vddd 供电。

活动状态电压调节器是主要的电压调节器，负责在活动状态或睡眠状态时将外部电源降压到 1.8V，以供所有的内部数字电路使用。其输出接到引脚 Vccd，必须外接一个电容（如图 9-1 所示）。活动状态电压调节器是给内部使用的，输出引脚 Vccd 除了接电容外，不能接任何外部负载。

Vccd 可以是内部调压器的输出，也可以直接接输入电压。当作为内部调压器输出时，Vddd 的输入范围是 1.8V-5.5V，通过内部调压器产生芯片内部电路工作的电压。当直接接入输入电源时，其输入范围是 1.71-1.89V。内部活动状态电压调节器仍然是上电的，因为默认是使能的。在这种情况下必须在固件程序里禁用(参见 9.3.1.1)，以节约能耗。

另外，还有两个电压调节器是为深度睡眠状态和休眠状态供电。还有一个低噪声调节器可以给一些敏感的模拟电路提供电源，例如 bandgap 参考源电路、电容感应电路等等。

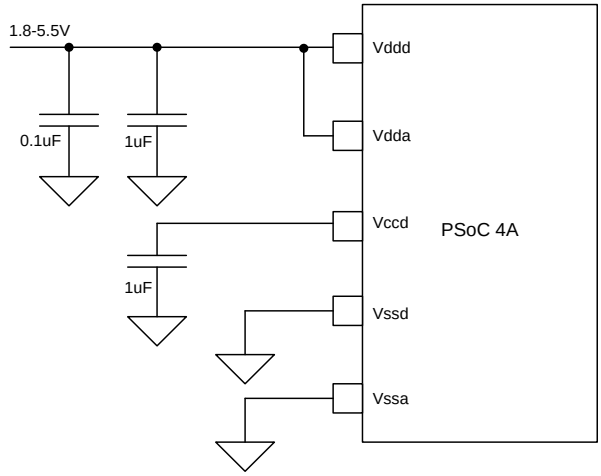
当外部电压最小值大于 1.8V 时，必须经过内部电压调节器供电。当外部电压最小值小于 1.8V 并且电压范围在 1.71V-1.89V (1.8V±5%)时，应直接由外部电源供电，旁路掉内部电压调节器。

经过内部电压调节器供电时，其输出引脚 Vccd 必须按照图 9-1 连接，即外接一个去耦电容到地（1uF, X5R 或者更好）。这个电压调节器仅供内部使用，其输出不能外接其他负载。外部连接如图 9-2 或 图 9-3 所示。

以下例举了 PSoC4A 常用电源连接方式。

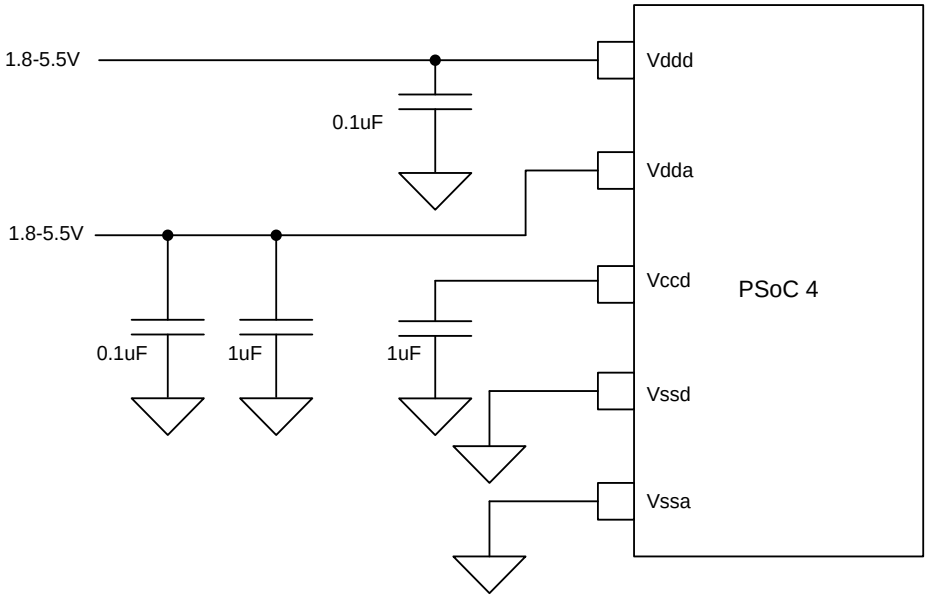
9.2.1 外部电压较高时（1.8V-5.5V）

图 9-2. 1.8V-5.5V，数字电源和模拟电源同路径供电



9.2 电源连接方式

图 9-3. 1.8V-5.5V，数字和模拟电源分开两条不同的路径供电



9.2.2 外部电压较低时（1.71V-1.89V）

在一些较小的封装中，只有 Vdd 和 Vss, 外部电压为 1.8V-5.5V 时，连接如图 9-4 所示：

图 9-4. 1.8V-5.5V, 单独 Vdd 供电

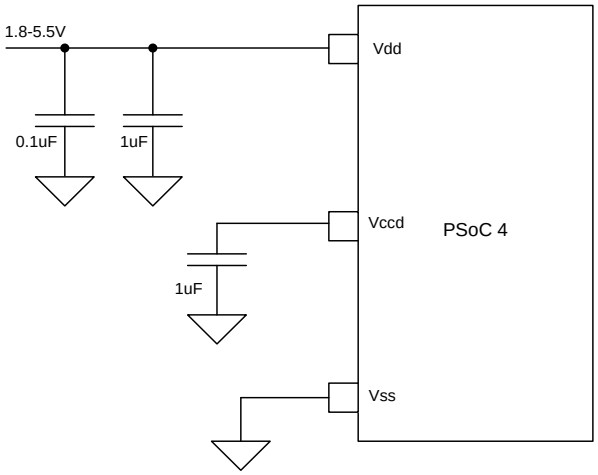


图 9-5. 1.71-1.89V, 外部电源直接供电，数字电源和模拟电源同路径供电

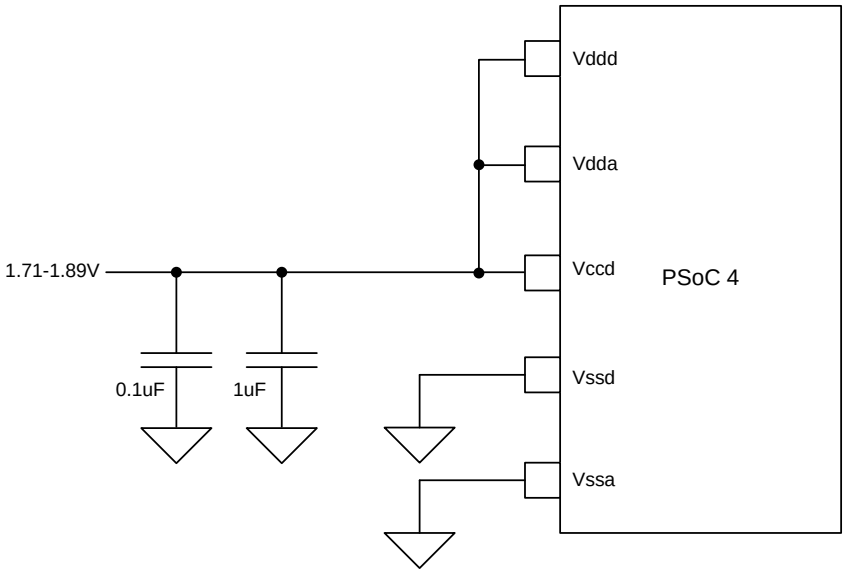


图 9-6. 1.71-1.89V, 外部电源直接供电，数字和模拟电源分开两条不同的路径供电

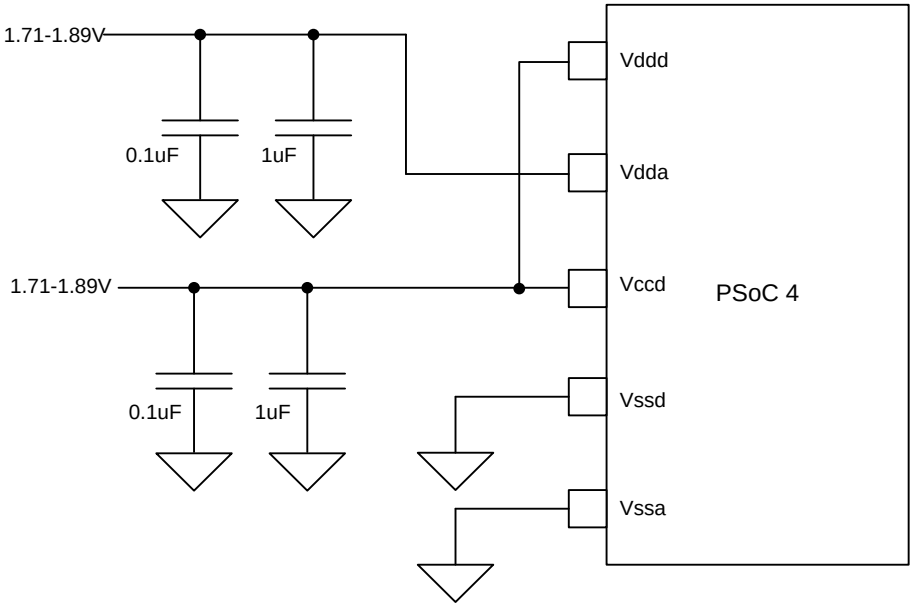
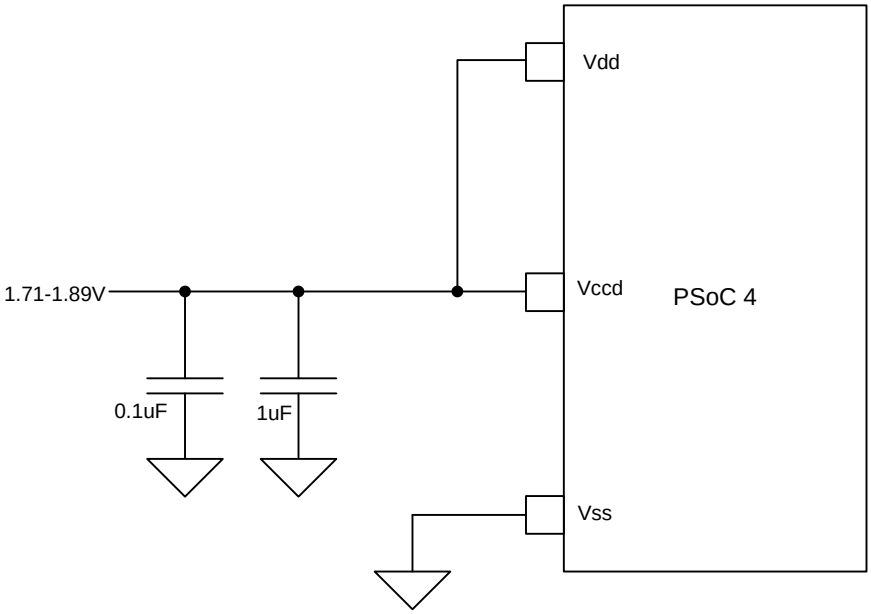


图 9-7. 1.71-1.89V, 外部电源直接供电，单独 Vdd



9.3 使用

如图 9-1 所示，芯片内部的数字电路由 Vddd 供电，模拟电路由 Vdda 供电。

9.3.1 内部调压器

活动和低噪声电压调节器可以在活动和睡眠状态工作。深度睡眠调压器和休眠调压器在相应的状态下工作。

9.3.1.1 活动状态调压器

当外部电压在 1.8V 到 5.5V 时，这个调节器在活动和睡眠状态给芯片内部的数字逻辑电路工作。这个调节器的输出街道引脚（Vccd），并要求改引脚外接一个去耦电容（1uF，X5R）。当电源电压低于 1.8V 时，Vccd 直接接电源，Vccd 和 Vddd 必须直接短接到一起。此时，活动状态寄存器可以通过设 PWR_CONTROL 寄存器中的 EXT_VCCD 位来禁用。

9.3.1.2 低噪声调压器

低噪声调节器可以给一些敏感的模拟电路提供电源，例如 bandgap 参考源电路、电容感应电路等等。这个调压器有特别高的电源抑制比，可以滤除数字电路高速开关切换在电源系统上引入的噪声。

9.3.1.3 深度睡眠寄存器

深度睡眠状态的电压调节器在深度睡眠状态下为数字电路提供电源（如内部低速时钟、串行通信模块等等）。在活动和睡眠模式中，这个调压器的输出连到活动状态调压器输出（VCCD）。这个调压器独立的一个输出，给 ILO 提供稳定的保持电压。

9.3.1.4 休眠寄存器

休眠状态的电压调节器在休眠状态下为数字电路提供电源，如睡眠控制器和低功耗比较器，以及 SRAM 和 UDB 的保持电压。在活动和睡眠模式中，这个调压器的输出连到活动状态调压器。在深度睡眠模式，这个调压器的输出连到深度睡眠调压器输出。

9.3.2 电压检测

电压检测包括上电重启（POR），欠压检测(BOD)和低压检测(LVD)。

9.3.2.1 上电重启（POR）

上电重启电路在芯片开始上电时提供重启脉冲，通常触发电平并不是十分精确。上电重启电路只在初始上电时工作，过后就不工作。

9.3.2.2 欠压检测（BOD）

当检测到 Vccd 电压低于最低安全运行电压（SOV，请见芯片数据手册）时，欠压检测电路会产生一个重启脉冲，以防止芯片在运行/保持状态时的电压逻辑不正常。欠压保护电路在除了停止模式以外的任何运行模式(活动/睡眠/深度睡眠/休眠) 都可以工作。

9.3.2.3 低压检测

低压检测用于精确的检测外部电压的状态。当外部电源出现异常或者即将耗尽时，可以产生一个中断，提醒系统做预防性措施。LVD 只能在活动或睡眠状态时使用。LVD 可以通过设置 LVD_EN 位使能（PWR_VMON_CONFIG [0]）。

通过配置 LVD_SEL 位（PWR_VMON_CONFIG [4:1]）， 触发电平可以在 1.75V-4.5V 范围内自定义。

在 LVD 电路稳定前，使能低压检测有可能会触发错误的中断。可以在使能低压检测位 LVD_EN 的 1uS 内屏蔽 LVD 中断来避免这个问题。推荐的 LVD 使用步骤如下：

1. 设置 PWR_INTR_MASK 寄存器中的 LVD 位为 0, 确保开始时中断被屏蔽。
2. 通过配置 LVD_SEL（PWR_VMON_CONFIG [4:1]），设置触发电平。
3. 通过设置 LVD_EN 位使能低压检测（PWR_VMON_CONFIG [0]）。这时可能会检测到有低压时间，但是由于设置了中断了屏蔽，不会有中断产生。
4. 等待 1uS 的电路稳定时间。
5. 通过写“1”到 PWR_INTR 寄存器的 LVD 位，清除中断标志位。
6. 清除 PWR_INTR_MASK 的中断屏蔽位（写“1”到 LVD 位）

9.4 寄存器列表

表 9-1. 电源和电源监测寄存器表

寄存器名称	描述
PWR_INTR	电源中断标志寄存器
PWR_INTR_MASK	电源中断屏蔽寄存器
PWR_VMON_CONFIG	电源检测校正和配置寄存器
PWR_CONTROL	电源模式控制，当前状态观察寄存器
PWR_BOD_KEY	BOD 重启标记寄存器

10 运行模式

PSoC 4 可以在 4 种不同运行模式下执行软件程序。在不同模式下，系统会执行存储在闪存（FLASH）和静态只读存储器（SROM）中的不同代码，并要求有相应的硬件权限。4 种模式中仅有 3 种模式会被用户使用；调试模式主要用于用户软件开发过程中的调试。

PSoC 4 的 4 种运行模式如下：

- 引导模式（Boot）
- 用户模式（User）
- 特权模式（Privileged）
- 调试模式（Debug）

10.1 引导模式

器件复位完成后，在没有调试请求的情况下系统会进入引导模式运行。运行在引导模式下的引导程序保存在静态只读存储器中。引导模式运行于特权模式下，所有中断会被禁止，从而使引导程序在无中断影响的情况下配置器件运行参数。在上电复位阶段，硬件的校正参数会被从非易失锁存器加载到相应寄存器中，从而保证后续正常操作所需的电压和时钟。引导程序执行结束后，程序会从闪存开始执行，并进入用户模式。闪存中的部分代码可能由 PSoC Creator 集成开发环境自动生成。

10.2 用户模式

用户的程序运行于用户模式下，在闪存中执行（不可在静态只读存储器中执行）。运行在用户模式下的程序包括 PSoC Creator 集成开发环境自动生成的代码和用户编写的代码；其中自动生成的代码包括软件启动代码和部分正常运行所需代码。一旦引导程序执行完成，系统会进入此模式来执行用户程序。

10.3 特权模式

用户程序通过设置寄存器 CPUSS_SYSARG 和 CPUSS_SYSREG，可使系统进入特权模式。运行在特权模式下的程序由 Cypress 编写并固化在静态只读存储器中（用户不可修改），用来执行专有功能，用户不可中断或观察其运行。在此模式下，调试功能被禁止。一旦运行在此模式下的程序执行完成，系统会自动返回到用户模式。

10.4 调试模式

调试模式主要用于用户软件开发阶段。在调试模式下，用户可借助调试器和集成开发环境（如 PSoC Creator 或 ARM MDK）对用户程序进行调试、查看运行参数。器件复位后，如果串行调试（SWD）端口有调试请求，则系统会进入调试模式。如需调试接口的详细信息，请参考编程和调试接口章节。

在不同的器件安全模式下，调试的权限不同。详情请参考器件安全章节。

11 电源模式

PSoC 4 提供了多种电源模式，可以最大程度地降低器件的功耗。

PSoC 4 的电源模式可分如下 5 种：

- 活动（Active）
- 睡眠（Sleep）
- 深度睡眠（Deep-Sleep）
- 休眠（Hibernate）
- 停止（Stop）

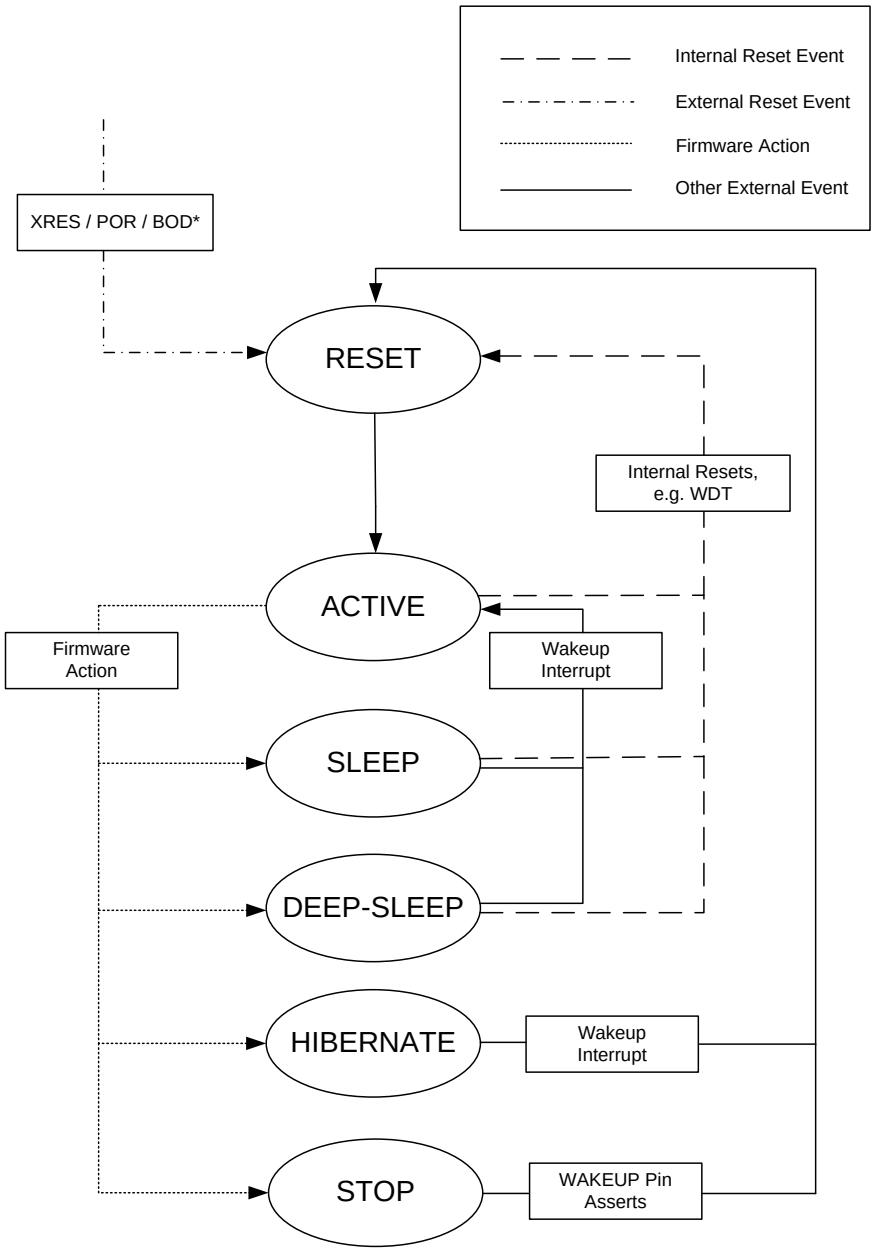
活动、睡眠和深度睡眠等 3 种电源模式是 ARM 的 3 种标准电源模式；ARM 处理器和指令集结构（ISA）均支持此 3 种电源模式。休眠和停止是 PSoC4 提供的功耗更低的电源模式。用户可通过软件设置进入休眠或停止模式。但是 CPU 及所有外设一旦被从休眠或停止模式唤醒，将进入复位（Reset）。

不同的电源模式可通过如下方式来实现：

- 使能/禁止外设时钟
- 开启/关闭内部电压调节器
- 开启/关闭时钟发生器
- 开启/关闭 PSoC 的部分组件

各种电源模式及其转换见图 11-1。

图 11-1 电源模式转换框图



*BOD 在停止模式下不可用。

表 11-1 描述了 PSoC 4 提供的各种电源模式。

表 11-1. PSoC 4 电源模式

电源模式	描述	进入条件	唤醒源	可用时钟	醒来动作	可用的电压调节器
活动	PSoC 4 器件的基本工作模式，所有外设均可用（也可选择性的使用外设）	从睡眠、深度睡眠模式唤醒； 内部或外部的复位； 掉电检测； 上电复位	N/A	主振荡器 IMO 和低速振荡器 ILO 均可用（可配置）	处理中断	所有电压调节器均可用。当使用外部电压调节器时，可通过软件禁用 Active Digital 电压调节器。
睡眠	CPU 处于睡眠状态，SRAM 处于保持状态，所有外设均可用（也可选择性的使用外设）	配置寄存器	任一中断	主振荡器 IMO 和低速振荡器 ILO 均可用（可配置）	处理中断	所有电压调节器均可用。当使用外部电压调节器时，可通过软件禁用 Active Digital 电压调节器。
深度睡眠	所有内部供电来自于 Deep-Sleep 电压调节器； 内部主振荡器 IMO 和高速外设均被关闭，内部低速振荡器 ILO（32 kHz）和低速外设是否继续工作可选。 来自低速异步外设或低功耗比较器的中断可将其唤醒	配置寄存器	来自如下模块的中断： 端口中断， 低功耗比较器， 串口通信模块（SCB1/SCB2）， 看门狗定时器	低速振荡器 ILO（32KHz）	处理中断	Deep-Sleep 电压调节器和 Hibernate 电压调节器
休眠	仅 SRAM 处于保持状态，大多数模块的内部供电被关闭。 来自端口或低功耗比较器的中断可将其唤醒。	配置寄存器	来自如下模块的中断： 端口中断源， 低功耗比较器	无	复位	Hibernate 电压调节器
停止	所有内部供电被关闭，仅 GPIO 输出处于冻结状态。 只有 XRES 或 WAKEUP 管脚上的有效信号才可将其唤醒。	配置寄存器	XRES， WAKEUP 管脚	无	复位	无

11.1 活动模式

活动模式是 PSoC 4 器件的基本模式，用户可以选择性的使用器件的子系统和外设。在此模式中，CPU 将保持运行，并且所有外设均开启。当然，用户也可以通过软件动态地禁止不用的专用外设，以降低功耗。

11.2 睡眠模式

在睡眠模式中，CPU 处于睡眠状态，其主时钟被关闭。当 CPU 不需要处理任何任务时，就可进入睡眠模式。在程序的运行中，PSoC 4 可经常进入此模式，以实现器件的低功耗要求。任何使能的中断均可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、TCPWM 和 PCLK 等处于保持状态，主系统时钟关闭，内部低速振荡器 ILO（32 kHz）是否继续工作是可选的（如果 ILO 被关闭，则看门狗和 LCD 模块也不可用）。但不需要时钟或由外部接口（比如从模式的 I2C/SPI）提供时钟的外设仍可继续工作。来自 GPIO 管脚、低功耗比较器、从模式的 I2C 和看门狗的中断信号可将器件从深度睡眠模式唤醒。深度睡眠模式下可用的唤醒信号源见表 11-3。

在深度睡眠模式下，当低速时钟开启时，工作电流值约为 1.3μA；当低速时钟关闭时，工作电流值约为 0.5μA。

11.4 休眠模式

在此模式下，SRAM 处于保持状态。通过关闭所有时钟、断开 CPU 和所有外设的电源（用于唤醒系统的外设除外），系统可进入休眠模式。在此模式下，CPU 和所有外设将失去其状态值。

在休眠模式下，系统使用休眠电压调节器供电，以达到超低功耗的目的。任何管脚上信号的最大频率将受到限制。所有管脚上信号的翻转频率总和不能超过 10kHz。如果信号的翻转频率很高，在功耗将与使用深度睡眠模式没有什么显著区别。

在休眠模式下，只有来自管脚或低功耗比较器的中断信号才可将器件唤醒。一旦被唤醒，CPU 和大部分寄存器将进入复位状态，软件将从复位向量地址处开始运行。

器件复位将使 GPIO 进入高阻模拟模式（被程序设置为冻结状态的管脚除外）。复位后，中断状态仍然有效，以使系统能够识别唤醒器件的中断源。

如果外部复位信号（XRES）被触发，将使整个系统重启。在此情况下，器件重启后，系统将无法判别重启的原因。在休眠模式下，工作电流值约为 150nA.

11.5 停止模式

在停止模式下，CPU、所有内部电压调节器和外设均被关闭；GPIO 输出处于冻结状态（Frozen：所有 GPIO 的配置、模式及状态处于锁存状态；在器件进入活动模式前，无法改变 GPIO 的状态）；仅有 GPIO 的状态被保持。在此模式下，只有通过 XRES 或 WAKEUP 管脚上的信号才可将系统唤醒，系统被唤醒后进入系统复位。

注意：如果使用 XRES 将器件唤醒，其将失去冻结 GPIO 的状态。

在停止模式下，工作电流值约为 20nA.。

表 11-2 描述了各种电源模式下的可用外设；表 11-3 描述了各种电源模式下有效的唤醒源。

表 11-2 各种电源模式下的可用外设

		活动	睡眠	深度睡眠	休眠	停止
CPU		on	Retention[b]	Retention	off	off
SRAM		on	Retention	Retention	Retention	off
时钟	内部主振荡器（IMO）	on	on	off	off	off
	内部低速振荡器（ILO）	on	on	on（可选）	off	off
	看门狗定时器 [a]	on	on	on	off	off
数字外设	通用数字模块（UDB）	on	on	Retention	off	off
	LCD 控制模块	on	on	on	off	off
	串口通信模块（SCB：UART）	on	on	off	off	off
	串口通信模块（SCB：SPI）	on	on	on（仅从模式）	off	off
	串口通信模块（SCB：I2C）	on	on	on（仅从模式）	off	off
	TCPWM	on	on	Retention	off	off
	PCLK	on	on	Retention	off	off
模拟外设	CTBm	on	on	off	off	off
	SAR ADC 模块	on	on	off	off	off
	低功耗比较器	on	on	on	on	off
	Capsense 模块	on	on	off	off	off
电源和复位	上电复位（POR）[c]，掉电检测（BOD）	on	on	on	on	off
	外部复位管脚 XRES	on	on	on	on	on
高速输入输出矩阵（HSIOM）		on	on	on	on	off
GPIO 输出状态		on	on	on/frozen	frozen[d]	frozen

a. 看门狗定时器 — 在深度睡眠模式下，如果 ILO 被关闭，看门狗定时器将不可用。

b. 保持（Retention）— 外设的配置和状态处于保留状态（相应电源模式的电压调节器提供电压，但是外设时钟被关闭）。器件一旦醒来进入活动模式，外设可继续工作。

c. 上电复位（POR）— 上电复位电路在器件上电启动完成后被禁用，但是其可被掉电检测（BOD）电路使能。所以说上电复位在深度睡眠和休眠模式下也是可用的。

d. 冻结（Frozen）— GPIO 的配置、模式及状态处于锁存状态。在器件进入活动模式前，无法改变 GPIO 的状态。

注意：on — 外设模块处于正常工作状态；off — 外设模块被关闭，不可用。

表 11-3 各种电源模式下的唤醒源

电源模式	唤醒源	醒来动作
睡眠	任何中断源	处理中断
	任何复位源	复位
深度睡眠	端口中断	处理中断
	低功耗比较器	处理中断
	串口通信模块 SCB	处理中断
	看门狗定时器中断/复位	处理中断 / 复位
	外部复位信号（XRES）[a]	复位
休眠	端口中断	复位
	低功耗比较器	复位
	外部复位信号（XRES）[a]	复位
停止	WAKEUP 管脚	复位
	外部复位信号（XRES）[a]	复位

a. 外部复位信号（XRES）会引起整个系统重启；所有外设（包括被冻结的 GPIO）的状态会被丢失。在这种情况下，PSoC 4 重启后，无法识别唤醒原因。

11.6 低功耗模式的使用

当 Cortex-M0 执行一条中断等待（WFI）指令时，系统将从活动模式进入睡眠、深度睡眠或休眠模式（见表 11-4）。如果用户需要 CPU 执行完所有中断服务程序（ISR）立即再次进入低功耗模式，可通过设置 Cortex-M0 系统控制寄存器（SCR）的 SLEEPONEXIT 位来实现。

寄存器 SCR 的 SLEEPDEEP 位和寄存器 PWR_CONTROL 的 HIBERNATE 位决定了系统将从活动模式转换至睡眠、深度睡眠还是休眠模式。表 11-4 描述了当 Cortex-M0 执行 WFI 指令时，因 SLEEPDEEP 位和 HIBERNATE 位配置不同而进入不同的低功耗模式。

表 11-4 从活动模式转换至低功耗模式的寄存器配置

从活动模式转换至的低功耗模式	SCR.SLEEPDEEP	PWR_CONTROL.HIBERNATE
睡眠	0	X（任意值）
深度睡眠	1	0
休眠	1	1

用户可通过寄存器 PWR_STOP 来设置 GPIO 在低功耗模式下是否处于冻结状态。在休眠模式下，建议用户将 GPIO 设置为冻结状态，因为从休眠模式醒来，器件将进入系统复位。

使能的中断可以将系统从睡眠、深度睡眠和休眠等低功耗模式唤醒，详情请参考表 11-3。此外，任何 RESET 后，系统将转换至活动模式。

通过设置寄存器 PWR_STOP，系统可直接从活动模式进入停止模式。在此模式下，系统中所有的低压逻辑的电源会被关闭，仅有冻结的 I/O 状态和寄存器 PWR_STOP 中的值会被保持。来自 XRES 或 WAKEUP 管脚的有效信号可将其唤醒，并进入系统复位。

寄存器 PWR_STOP 中的字段定义如下：

- TOKEN — 这个字段包括 8 个位，其数值在从停止模式到被唤醒的过程中保持不变。软件可以通过读取其值来判断系统的唤醒是由 WAKEUP 管脚信号还是由一个普通的 RESET 事件引起的。但是由 XRES 引起的唤醒将使此字段复位。
- UNLOCK — 通过向这个字段写 0x3A 可以解锁停止模式；其他值将导致 STOP 位无效。
- POLARITY — 此位用来设置 WAKEUP 管脚输入的极性。当 WAKEUP 管脚输入信号的极性与设置的值匹配时，处于停止模式的器件将被唤醒。
- FREEZE — 此位置 1 可冻结所有 GPIO 的配置、模式和状态。
- STOP — 设置此位使系统进入停止模式。

要使系统进入停止模式，建议对寄存器 PWR_STOP 按如下配置连续赋值 3 次：

- TOKEN = <应用程序指定的值>
- UNLOCK = 0x3A
- POLARITY = <应用程序指定的极性>
- FREEZE = 1
- STOP = 1

第 3 次赋值之后，建议用户放置两条空指令 NOP。

当 XRES 或 WAKEUP 管脚信号有效时，器件将退出停止模式。这两个事件均可将寄存器 PWR_STOP 中的 STOP 位清除，并触发上电复位（POR）。WAKEUP 事件不清除寄存器 PWR_STOP 的其他位，而 XRES 事件将清除寄存器 PWR_STOP 中的所有位。

系统重启后，建议用户使用如下软件处理流程：

- <可选>读取 TOKEN 值，处理不同的分支程序
- <可选>设置 IO 驱动模式及输出值。对于数字输出管脚，程序可将其设置为输出、读取其冻结值，并将此值设置到输出数据寄存器中。
- 解冻 IO

11.7 寄存器列表

表 11-5 电源模式寄存器列表

寄存器	描述
系统控制寄存器（SCR）	设置 或者返回系统控制数据
电源模式控制寄存器（PWR_CONTROL）	控制电源模式选择，查看当前的电源模式状态。
停止模式寄存器（PWR_STOP）	控制进入 / 退出停止模式

12 看门狗定时器

看门狗定时器（WDT：WatchDog Timer）是保证嵌入式系统鲁棒性的一个重要单元。当WDT被使能时，微控制器必须通过程序定期的去刷新看门狗定时器中的计数值，否则会被重启，以避免程序陷入异常状态。在PSoC 4中，WDT的工作时钟是ILO（Internal Low-power Oscillator）,它的频率是32KHz。

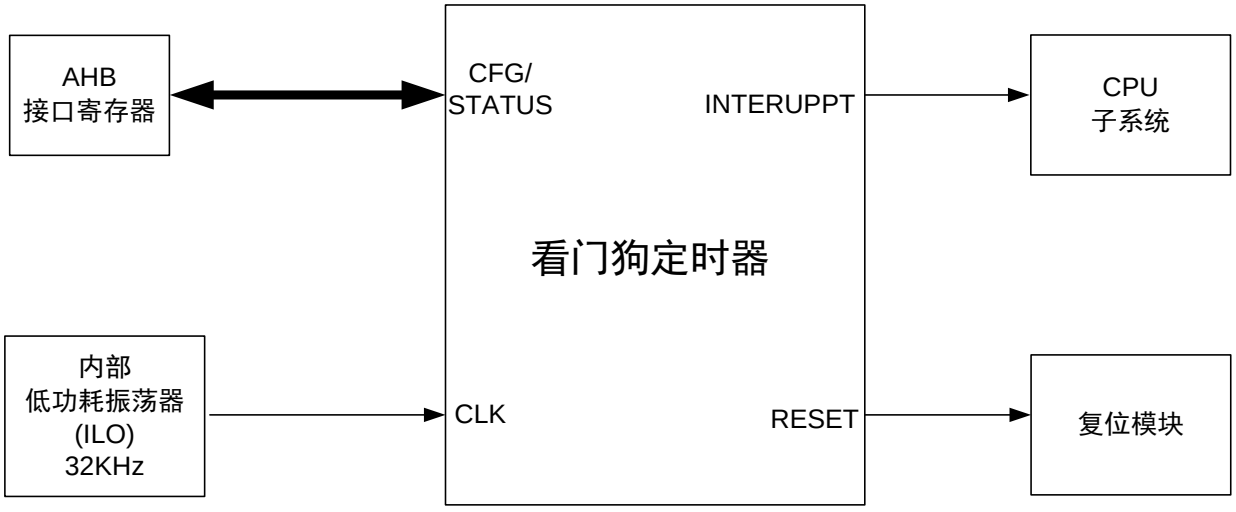
12.1 特性

PSoC 4中的WDT主要有以下几个特性：

- 三个可配置周期的计数器；
- 三个计数器可以独立应用或者级联应用；
- 在睡眠和深度睡眠的电源模式下，WDT能够产生唤醒中断；
- 在活动的电源模式下，可以在指定的时间产生中断；
- 防止寄存器数据被意外破坏的保护设置。

12.2 模块框图

图12-1 看门狗定时器的模块框图



12.3 工作原理

如果WDT的计数值在指定的时间内没有被刷新，它将输出中断或者硬件复位，时间可以通过编程控制。在PSoC 4中，WDT一共包含了两个16-bit的计数器（计数器0和计数器1）和一个32-bit计数器（计数器2）。这三个计数器可以被独立操作，或者级联操作。计数器通过级联，可以拉长两次中断或者复位之间的间隔。
计数器0和计数器1在达到指定时间后可以输出中断或者硬件复位；而计数器2则只能够输出中断。

12.3.1 WDT 的使能和禁止

WDT的使能和禁止分别通过寄存器WDT_ENABLEx（x = 0或者1）的置位与清零来实现。对WDT的使能或者禁止需要通过三个低频时钟（LFCLK）周期才能够生效。WDT_ENABLEx的有效值必须保持至少一个LFCLK的周期。
在WDT被使能的情况下，对配置寄存器WDT_CONFIG和控制寄存器WDT_CONTROL的操作是非法的。将CLK_SELECT寄存器中的WDT_LOCK置位，可以禁止所有WDT寄存器的写操作。如果用户需要更新WDT的配置，需要首先将WDT_LOCK清零。

12.3.2 WDT 的操作模式

计数器0和计数器1有多种操作模式，它可以产生中断唤醒低功耗电源模式中的PSoC 4，可以产生复位以避免PSoC 4由于不当的程序陷入异常的不响应状态。寄存器WDT_MODE[1:0]被用来配置具体的操作模式，当计数器的计数值WDT_CTRx等于目标计数值WDT_MATCHx时（x = 0或者1），中断或者复位就会发生。

表12-1 计数器0和计数器1的操作模式

寄存器	描述
WDT_MODEx[1:0]	当计数器计数值与目标计数值匹配（WDT_CTRx = WDT_MATCHx）时，发生以下操作： 00：无操作 01：产生中断 10：产生复位 11：产生中断，若连续三次中断未被处理，则产生复位

注意：x = 0 或 1。

计数器2只能产生中断，如表2。

表12-2 计数器2的操作模式

寄存器	描述
WDT_MODE2	当计数器寄存器中的某一位翻转时（具体哪一位由寄存器WDT_BITS2[4:0]的值决定，详见表12-4），发生以下操作： 0：无操作； 1：产生中断

12.3.3 WDT 的中断和低功耗模式

在活动电源模式下，WDT声明的中断请求被发送到CPU；在睡眠和深度睡眠电源模式下，WDT声明的中断被发送到唤醒中断控制器（WIC：Wake-up Interrupt Controller），具体如下：

- 活动电源模式：CPU子系统和WIC都工作在活动状态。CPU负责响应中断请求并执行中断服务程序（ISR：Interrupt Service Routine）。进入ISR后，软件需要向中断请求寄存器WDT_INTx（x = 0,1或2）写1，将它清零;
- 睡眠/深度睡眠电源模式：CPU子系统被关闭，中断被送到WIC来唤醒PSoC 4。CPU被唤醒后，对中断请求进行响应并执行中断服务程序。中断请求寄存器需要在ISR中被清零。

12.3.4 WDT 的复位

在 WDT 触发复位之后，寄存器 RESET_CAUSE 中的 RESET_WDT 位会被置位，以记录发生过 WDT 的复位。RESET_WDT 的置位状态会被一直保持到软件对它执行清零，发生上电复位（POR）或者掉电复位（BOD）。更多细节请参见复位章节。

12.4 寄存器列表

表12-3 计数器0和计数器1的状态与控制寄存器

寄存器	描述
WDT_CLEARx	WDT计数器的计数范围寄存器 0：计数器的计数范围为全0x0000到0xFFFF； 1：计数器的计数范围为全0到目标计数值WDT_MATCHx.
WDT_ENABLEx	计数器使能寄存器 0：禁止； 1：使能。
WDT_INTx	WDT的中断请求寄存器 硬件置位，软件清零。 在操作模式3中，如果连续三次中断寄存器未被清零，则WDT会声明复位
WDT_RESETx	WDT的计数器复位清零寄存器软件向该位写“1”，将WDT计数器复位，其计数值归零，之后硬件会自动将该寄存器位重新清零。
WDT_CASCADE0_1	级联配置寄存器 0：计数器0和计数器1独立操作 1：计数器0和计数器1级联操作，当计数器0的计数值达到目标计数值时，计数器1的计数值累加1。
WDT_MATCH0	计数器0的目标计数值
WDT_MATCH1	计数器1的目标计数值

注意：x = 0 或 1。

表12-4 计数器2的状态与控制寄存器

寄存器	描述
WDT_CASCADE1_2	级联配置寄存器 0：计数器1和计数器2独立操作 1：计数器1和计数器2级联操作，当计数器1的计数值达到目标计数值时，计数器2的计数值累加1。
WDT_ENABLE2	计数器使能寄存器 0：禁止； 1：使能。
WDT_INT2	WDT的中断请求寄存器 硬件置位，软件清零。
WDT_RESET2	WDT的计数器复位清零寄存器软件向该位写“1”，将WDT计数器复位，其计数值归零，之后硬件会自动将该寄存器位重新清零。
WDT_BITS2[4:0]	翻转触发中断的比特位设置寄存器 该寄存器决定：当计数器2的计数寄存器哪一位发生翻转会触发中断。 0：第0位翻转时触发中断； 1：第1位翻转时触发中断； …… 31：第31位翻转时触发中断。

更多细节请参见PSoC 4 寄存器TRM中xxx页的WDT寄存器。

13 复位

PSoC 4 支持多种类型的复位(Reset)，其包括器件的上电复位、用户提供的外部复位和软件复位等等。

PSoC 4 的复位系统具有如下复位源：

- 上电复位（Power-on Reset, POR）：当为器件连接电源使其启动时，供电电压有个斜坡上升的过程，当电压高于阈值时，会引起上电复位，并将器件锁定在复位状态。
- 掉电检测复位（Brown-out Reset，BOD）：在程序运行的过程中，如果器件的供电电压低于规定值时，会引起掉电检测复位。
- 看门狗复位（Watchdog Reset, WRES）：当软件在规定的时间内没有复位看门狗中的定时器，即会引起看门狗复位。
- 软件复位（Software Initiated Reset, SRES）：用户可通过软件写相应寄存器来产生软件复位。
- 外部复位（XRES）：此复位是由来自复位管脚(XRES)上的外部信号引起的。
- 特权保护复位（Protection Fault Reset, PROT_FAULT）:当用户操作需具有特权的代码时，会产生一个特权保护复位。
- 休眠模式唤醒复位：当处于休眠模式的器件被唤醒时，会产生一个休眠模式唤醒复位。
- 停止模式唤醒复位：当处于停止模式的器件被唤醒时，会产生一个停止模式唤醒复位。

13.1 复位源

下面将详细介绍各种复位源。

13.1.1 上电复位

PSoC 4 在上电启动时，会发生上电复位，产生一个系统复位。 上电复位会将器件锁定在复位状态，直到 Vdda、Vddd 和 Vccd 三个电压全部达到数据手册要求的值。

在 PSoC 4 中，没有记录上电复位的寄存器，只可通过推测来判断系统复位可能是由上电复位引起的，详细信息请参考复位源识别章节。

13.1.2 掉电检测复位

PSoC 4 中包含一个掉电检测模块，其用于监测电压 Vccd 是否正常；如果 Vccd 低于数据手册中规定的最小逻辑工作电压，则掉电检测模块会产生一个复位信号。掉电检测模块可在活动、睡眠、深度睡眠和休眠模式下正常工作，但不能在停止模式下正常工作。

掉电检测复位可分为两种类型：可识别掉电检测复位和不可识别掉电检测复位。

- 可识别掉电检测复位：当 Vccd 低于最小逻辑工作电压、但仍高于最小逻辑保持电压时，一个掉电检测复位将发生，但是具有保持能力的寄存器的状态会被保持。系统复位后，程序可以通过查询寄存器 PWR_BOD_KEY 来判别是否发生了可识别掉电检测复位。
- 不可识别掉电检测复位：当 Vccd 同时低于最小逻辑工作和最小逻辑保持电压时，一个掉电检测复位将发生，但是寄存器的状态不能被保持。这种情况下的复位无法与上电复位和外部复位进行区分。

  关于掉电检测复位的识别，请参考复位源识别章节。

13.1.3 看门狗复位

如果在设定的时间内，用户程序没有复位看门狗中的计数器，则看门狗会产生一个复位信号，表示用户程序运行发生了错误。

当一个看门狗复位事件发生时，状态位 RESET_WDT（RES_CAUSE[0]）会被置 1。

关于看门狗复位的识别请参考复位源识别章节。关于看门狗的详细信息，请参考看门狗定时器章节。

13.1.4 软件复位

软件复位（SRES）是指用户通过软件写寄存器引起的系统复位；即程序运行的过程中，当用户将位 SYSRESETREQ（AIRCR[2]）置 1 时，会引起整个系统复位。

当软件复位发生时，状态位 RESET_SOFT（RES_CAUSE[4]）会被置 1。关于软件复位的识别请参考复位源识别章节。

13.1.5 外部复位

外部复位（XRES）是指当专用管脚 XRES 上的电平满足复位条件时引起的复位（XRES 管脚低电平时复位有效）。当管脚 XRES 电平一直处于复位电平状态时，器件将一直处于复位状态，直至电平处于非复位电平状态，然后器件会进入正常的启动过程。关于管脚 XRES 的电气特性，请参考数据手册。

在 PSoC 4 中，没有记录外部复位的寄存器，只可通过推测来判断系统复位可能是由外部复位引起的。详细信息，请参考复位源识别章节。

13.1.6 特权保护复位：

当用户尝试操作需具有特权的代码时，会产生一个特权保护复位，比如 CPU 在执行特权代码时遇到一个调试断点。

当特权保护复位发生时，状态位 RESET_PROT_FAULT（RES_CAUSE[3]）会被置 1。关于特权保护复位的识别请参考复位源识别章节。

13.1.7 休眠模式唤醒复位

当器件处于休眠模式时，来自 GPIO 管脚或低功耗比较器的中断信号可以将器件唤醒，并使器件进入复位状态。复位后，SRAM 中的内容会被保持，但是代码的启动过程与其他复位源一样。

复位后，中断状态位也会被保持。因此，程序可以通过读取中断状态位来识别复位源。关于休眠模式的详细信息，请参考电源模式章节。

13.1.8 停止模式唤醒复位

当器件处于停止模式时，来自管脚 WAKEUP 的有效信号可以将器件唤醒，并使器件进入复位状态。复位后，TOKEN（PWR_STOP[7:0]）的值会被保持（外部复位会复位 TOKEN）。关于休眠模式的详细信息，请参考电源模式章节。

13.2 复位源识别

当器件复位完成时，如果软件能够查询到引起复位的复位源，将有益于程序处理和应用。PSoC 4 提供了寄存器 RES_CAUSE 和寄存器 PWR_BOD_KEY 来帮助用户判断复位源 。因此，应用程序复位后，用户可以通过查询寄存器来判断最近一次的复位源。

在寄存器 RES_CAUSE 中，仅包含看门狗复位、特权保护复位和软件复位等状态位（见表 13-1）。一旦看门狗复位、特权保护复位或软件复位发生时，相应的标志位会立即被置 1；此寄存器的值将一直被保持，直至被软件、上电复位或不可识别掉电检测复位清除。

对于唤醒复位，用户可通过查询中断状态寄存器和寄存器 PWR_STOP 来判断是否发生了休眠模式唤醒复位和停止模式唤醒复位。详情请见电源模式章节。

表 13-1 寄存器 RES_CAUSE

寄存器 RES_CAUSE的位	**描述**
RESET_WDT （RES_CAUSE[0]）	记录看门狗复位的状态位
RES_CAUSE[2:1]	保留
RESET_PROT_FAULT (RES_CAUSE[3])	当用户尝试操作特权代码时，会产生一个特权保护复位。
RESET_SOFT (RES_CAUSE[4])	记录软件复位状态。此复位是通过软件向SYSRESETREQ（AIRCR[2]）写1，使Cortex-M0 CPU产生的复位。
RES_CAUSE[31:5]	保留

寄存器 PWR_BOD_KEY 的值仅可被软件、不可识别的掉电检测复位、上电复位或外部复位修改。用户可用其来辨别可识别掉电检测复位。在应用程序初始化时给此寄存器赋值，并在系统复位后读取此寄存器来判断是否发生了可识别掉电检测复位。推荐的软件设计见表 13-2。

表 13-2 寄存器 PWR_BOD_KEY

寄存器 PWR_BOD_KEY的位	**描述**
KEY16 (PWR_BOD_KEY[15：0])	用来识别上次复位是否由可识别的掉电检测复位引起的。 在系统启动时，用户软件可以通过如下步骤来查询上次复位是否由可识别掉电检测复位引起的： 1. 声明一个变量并赋值: key = KEY16 2. 设置 KEY16 = 0x3A71 3. 判断key 是否等于0x3A71；如果相等并且排除其他可识别的复位源，则表示上次系统复位是由可识别掉电检测复位引起的。

如果上述方法还不能判断复位的原因，那么此复位一定是一个不可记录的复位：不可识别掉电检测复位、上电复位或外部复位。基于器件的资源，无法区分这三种复位源。

14 器件安全

PSoC4 给用户提供了多种防止未授权访问和复制的选项。通过强大的闪存保护、禁用调试功能，以及在 UDB 中（而非 CPU 固件程序）实现的用户编写的逻辑，能保证更高的安全级别。

调试电路在默认情况下是使能的，只能在固件程序中被禁用。一旦被禁用，重新使能的唯一方法是擦除整个器件，清除闪存保护，然后用新的程序重新烧写器件。另外，对于通过恶意重新编程器件来进行欺诈性攻击或者企图通过启动/ 中断闪存编程时序来进行安全性攻击的顾虑，可以永久禁用所有的编程、调试、测试接口。在大多数应用中不建议这么做，因为禁用所有接口后，设计人员将无法对器件进行访问，也就不能退回进行故障分析，请根据实际情况权衡利弊。

14.1 特性

PSoC4 的器件安全系统有如下的特性：

- 用户可选择的保护级别
- 在最高安全级别的模式中，芯片被“锁住”，无法进入测试/调试模式，也无法进入擦除操作。中断擦除操作是一种已知的利用芯片在擦除操作中的未知状态进行观察的黑客方法。
- CPU通过不可屏蔽中断（NMI）进入特权模式（SVC指令此时不被使用）；在特权模式中，NMI会保持有效，以防止任何无意的中断返回导致的安全泄露。

14.2 工作原理

CPU 工作在用户模式或特权模式，而器件工作在四种保护模式：启动（BOOT）、开放(OPEN)、保护(PROTECTED)、终极保护(KILL)。所有的模式都有相应的 CPU 和调试接口的访问权利。

- 启动模式： 器件在上电复位以后进入到启动模式，在此模式下调试接口禁用。这个模式会一直持续到启动代码将保护状态从特权闪存（sFlash）中拷贝到保护控制寄存器中。通过这个临时的启动模式，系统的各个接口可以被配置到所需的保护状态。
- 开放模式：这是默认的出厂模式。CPU 可以工作在用户模式或特权模式。在用户模式下，支持闪存编程和调试接口访问。
- 保护模式：PSoC4 可以从开放模式变化到保护模式。保护模式禁止对所有用户代码和存储空间的调试访问（访问用户寄存器是允许的）。这样可以防止通过调试接口来对闪存进行重新编程。保护模式只有在完全擦除闪存后才能回到开放模式。
- 终极保护模式：用户可以在开放模式下，将芯片配置成终极保护模式。这种模式彻底禁止调试接口，只允许访问用户寄存器。这种模式是不可逆的；进入终极保护模式的芯片可能不能退回进行故障分析。

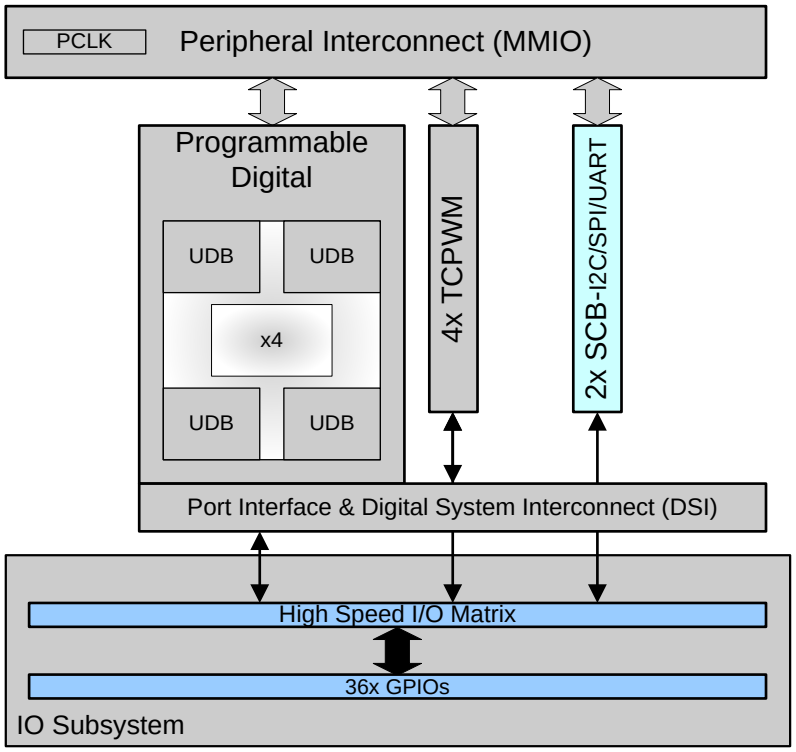
E 部分：数字系统

这部分包括以下章节：

- 串行通信模块　第 81 页
- 通用数字模块　第 117 页
- 计时器、计数器、脉冲宽度调制（PWM）模块　第 154 页

系统架构：

数字系统结构框图



15 串行通信模块 (SCB)

串行通信模块（SCB: Serial Communication Block）支持三种通信协议：SPI，UART 和 I2C。PSoC 4 提供了两个 SCB 模块，每个 SCB 模块在同一时刻只能用于实现一种串行通信协议。如果在设计中需要同时实现三个及以上串行通信协议，用户可以借助 UDB 来设计。

15.1 特性

SCB 模块的特性主要如下：

- 支持摩托罗拉 SPI 协议、德州仪器 SPI 协议和国家半导体 SPI 协议
- 支持 SPI 主设备模式和从设备模式；
- 支持数据卡协议（ISO7816）和红外数据传输协议（IrDA）；
- 支持 I2C 主设备模式和从设备模式；
- 在 SPI 协议和 I2C 协议中，支持 EZ 的工作模式，该模式下的通信无需 CPU 的介入；
- 在深度睡眠的电源模式下，仍能够支持 SPI 和 I2C 协议中的部分操作。

下面分别介绍如何应用 SCB 实现三种协议。

15.2 串行外设接口（SPI）协议

SPI 是一种同步的串行通信接口协议。此协议中有主设备和从设备，主设备负责发起数据传输。SCB 支持 SPI 协议中“一主多从”拓扑的应用，此时主设备利用从设备选择信号线指定某一从设备与之通信。

用户设计中，如果 PSoC 4 需要和多个 SPI 从设备通信，则应将 PSoC 4 设置为主设备模式；如果系统中已经存在 SPI 主设备，且 PSoC 4 需要与之通信，则应将 PSoC 4 设置为从设备模式。

15.2.1 特性

SCB 实现 SPI 协议时，有如下特性：

- 支持 SPI 主设备模式和从设备模式；
- 支持多家厂商定义的 SPI 协议：
  - 摩托罗拉 SPI 协议，包括模式 0, 1, 2 和 3；
  - 德州仪器 SPI 协议，加入数据帧头指示机制，仅适用于模式 1；
  - 国家半导体 SPI 协议，半双工的传输协议，仅适用于模式 0。
- 可配置数据帧长度，从 4 比特到 16 比特；
- CPU 通过中断或轮询与 SCB 通信；
- 支持数据接收的过采样；
- 支持 EZ（EASY）的工作模式；
- 当 SCB 实现的 SPI 设备作为从设备时，支持外部时钟模式
  - 这时 SCB 可以工作在活动、睡眠和深度睡眠电源模式下；
  - 如果 SCB 同时工作在 EZ 模式下，则此时的通信无需 CPU 介入。

15.2.2 概述

图 15-1 一主三从的 SPI 通信系统框图

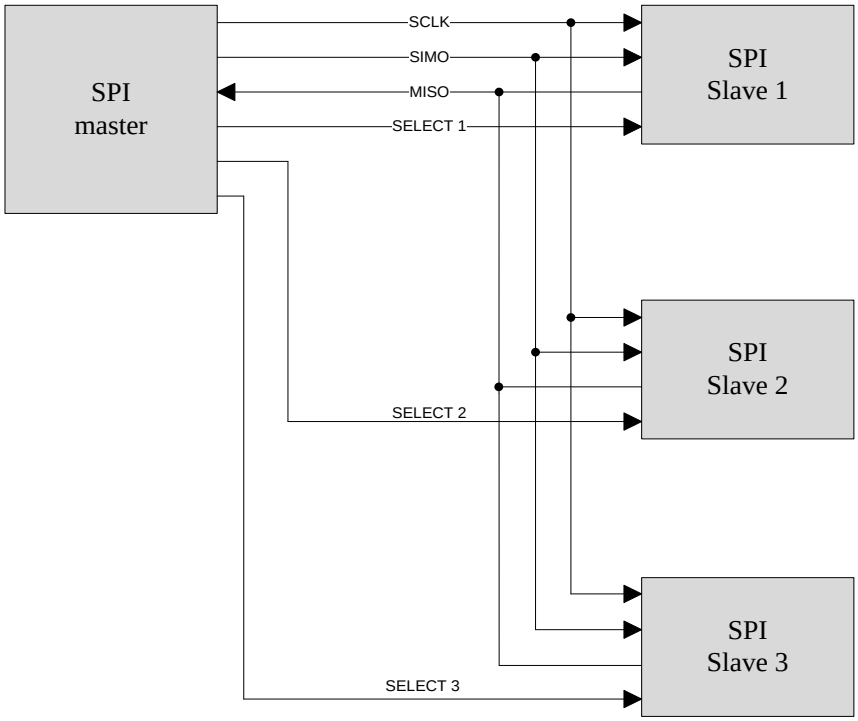


图 15-1 给出了一主三从拓扑结构的 SPI 通信系统框图，主要有四个信号：

- SCLK：串行时钟信号，由主设备产生，并传输到从设备作为同步时钟；
- MOSI/SIMO：主出从入数据信号，由主设备输出，输入到从设备；
- MISO/SOMI：主入从出数据信号，由从设备输出，输入到主设备；
- SELECTx（x = 1,2 和 3）：从设备选择信号，由主设备输出，输入到从设备，一般为低电平有效。

某些应用中，MOSI/SIMO 和 MISO/SOMI 这两个单向数据线被合并为一根双向数据线，如 SDAT，这样可以节省一根数据线。

SPI 协议的数据传输过程可以简述如下：

1. 主设备控制某一根 SELECTx 信号线，选择从设备；
2. 主设备向从设备同时传输时钟（通过 SCLK）与数据（通过 MOSI/SIMO）；
3. 从设备使用该时钟（或者与其同步的内部时钟）对数据采样；
4. 如果从设备需要向主设备回复数据，则使用传输时钟（SCLK）将回复的数据打到数据线 MISO/SOMI 上；
5. 主设备使用传输时钟对 MISO/SOMI 上的数据进行采样；

缺省情况下，SPI 协议中一帧数据的长度为 8（1 个字节）；但通过配置，可以在 4 到 16 之间变化。在 SCB 实现 SPI 协议时，最高有效位（MSB）或者最低有效位（LSB）的传输顺序可以配置的。

SCB 支持三家厂商定义的 SPI 协议：

- 摩托罗拉（Motorola）SPI 协议：这是 SPI 协议的初始版本；
- 德州仪器（Texas Instrument）SPI 协议：它在摩托罗拉 SPI 协议的基础上加入了帧指示机制；
- 国家半导体（National Semiconductor）SPI 协议：这是摩托罗拉 SPI 协议的一种半双工的变化。

下面分别介绍这三种厂商协议。

15.2.3 SPI 协议介绍

15.2.3.1 摩托罗拉 SPI 协议

15.2.3.1.1 简介

SPI 协议最早由摩托罗拉公司提出，是一种全双工的串行通信协议。

在连续传输多帧数据时，SELECT 信号一直被拉低。从设备就无法利用 SELECT 的状态来区分每一帧，从而需要一直监测数据，根据主从设备之间约定的协议找出帧头和帧尾。当总线结束数据传输时，SELECT 被重新拉高，SCLK 无时钟输出。

摩托罗拉定义了 SPI 协议中的四种信号传输模式，规定数据在哪个时钟沿被发送和采样。通过配置时钟极性（CPOL: Clock POLarity）和时钟相位（CPHA: Clock PHAse）来选择四种模式中的一种。

CPOL 决定了 SCLK 空闲态中的电平：

- CPOL = 0：空闲态中，SCLK 为低电平；
- CPOL = 1：空闲态中，SCLK 为高电平；

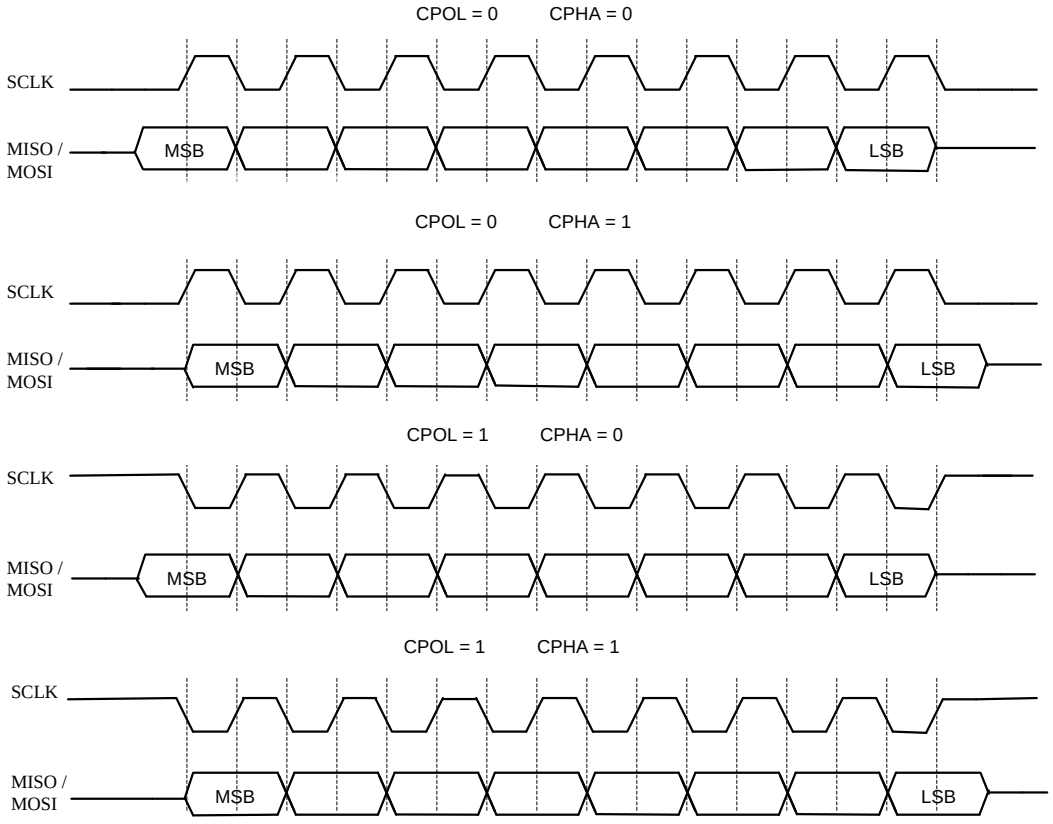
CPHA 决定了主设备和从设备在什么时候采样数据和发送数据

- CPHA = 0：主从设备在同步传输时钟的第一个沿采样数据，第二个沿发送数据；
- CPHA = 1：主从设备在同步传输时钟的第一个沿发送数据，第二个沿采样数据。

经过组合，可以得到四种传输模式，波形如图 15-2 所示，具体如下所列：

- 模式 0（CPOL = 0，CPHA = 0）：SCLK 空闲态为低，时钟上升沿采样数据，时钟下降沿发送数据；
- 模式 1（CPOL = 0，CPHA =1）：SCLK 空闲态为低，时钟上升沿发送数据，时钟下降沿采样数据；
- 模式 2（CPOL =1，CPHA = 0）：SCLK 空闲态为高，时钟上升沿发送数据，时钟下降沿采样数据；
- 模式 3（CPOL =1，CPHA =1）：SCLK 空闲态为高，时钟上升沿采样数据，时钟下降沿发送数据。

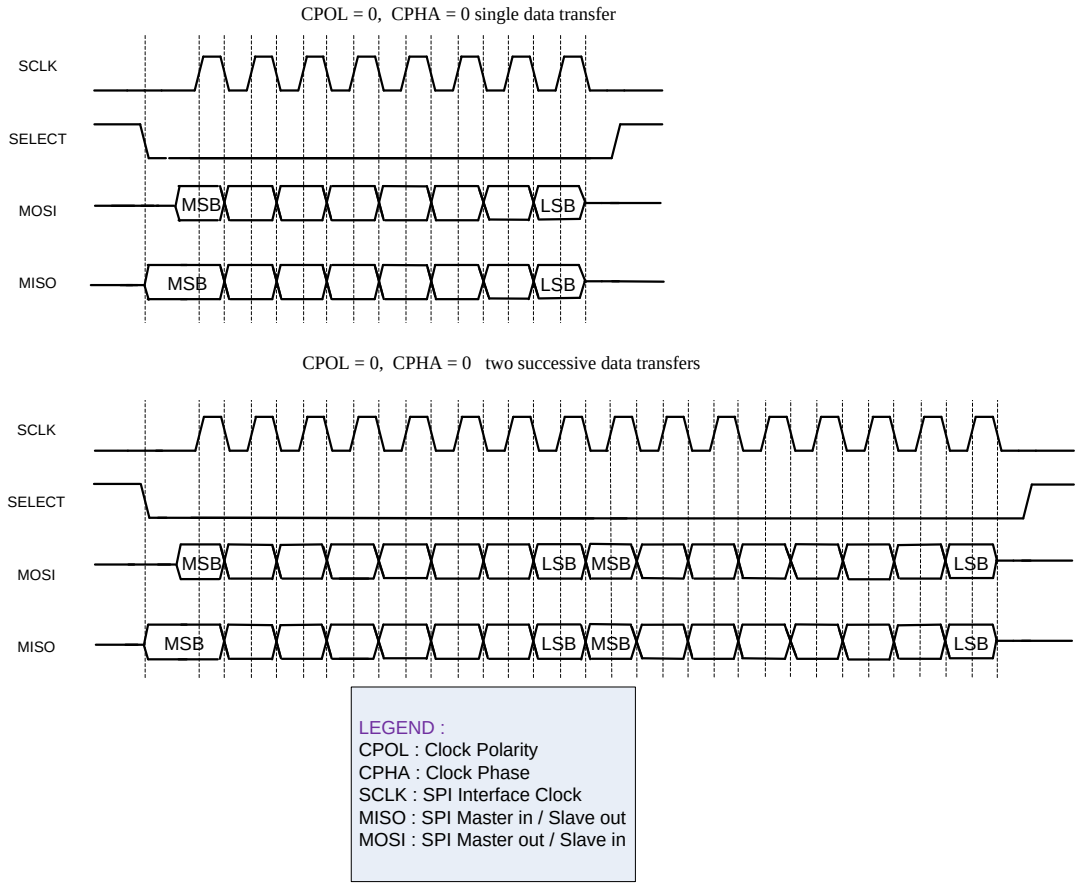
图 15-2 摩托罗拉 SPI 协议中四种传输模式的波形



LEGEND:
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI interface clock
MOSI : SPI Master out / Slave in
MISO : SPI master in / Slave out

图 15-3 给出了传输模式 0 下，单帧（8bit）数据传输和双帧（16bit）数据传输的波形图。

图 15-3 摩托罗拉 SPI 协议的传输波形



15.2.3.1.2 配置

通过下列的寄存器配置，可使 SCB 工作在摩托罗拉 SPI 协议下。

1. 将寄存器 MODE（SCB_CTRL[25:24]）设置为 01，选择 SPI 协议；
2. 将寄存器 MODE（SCB_SPI_CTRL[25:24]）设置为 00，进一步选择摩托罗拉 SPI 协议；
3. 配置寄存器 CPHA（SCB_SPI_CTRL[2]）和 CPOL（SCB_SPI_CTRL[3]），选择数据传输模式；
4. 根据章节 15.2.6（SPI 的初始化）中的步骤 2-4，对 SCB 实现的 SPI 接口控制器进行初始化和使能。

注意：上述的这些配置都可以在 PSoC Creator 中通过图形用户界面完成，我们更推荐这种设计方式。如果需要更多的了解寄存器的信息，请参考 PSoC 4 的寄存器用户手册。

15.2.3.2 德州仪器 SPI 协议

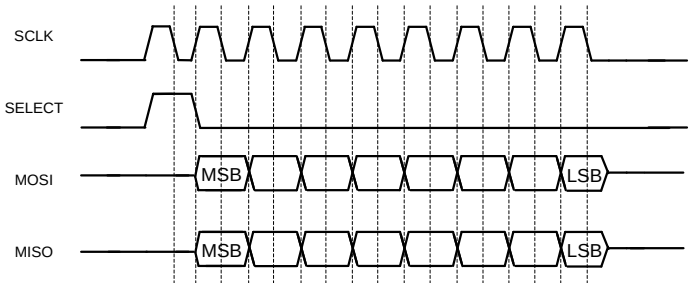
15.2.3.2.1 简介

德州仪器 SPI 协议重新定义了 SELECT 信号的用途。SELECT 信号被用于指示一帧数据的帧头，而不仅作为一个低有效的从设备选择信号。主设备在 SELECT 信号上驱动一个持续时间为一个周期的高脉冲，来表明一个数据帧传输的开始。根据配置，这个高脉冲可以与帧头重合，也可以提前一个周期。这时，从设备不必消耗内部资源找出帧头和帧尾，这是德州仪器版本相对于摩托罗拉版本的优势。根据配置，SELECT 上的帧指示脉冲可以指示帧头，也可以指示帧尾。德州仪器 SPI 协议仅支持模式 1（CPOL = 0, CPHA =1），即 SCLK 空闲态为低，时钟上升沿发送数据，时钟下降沿采样数据

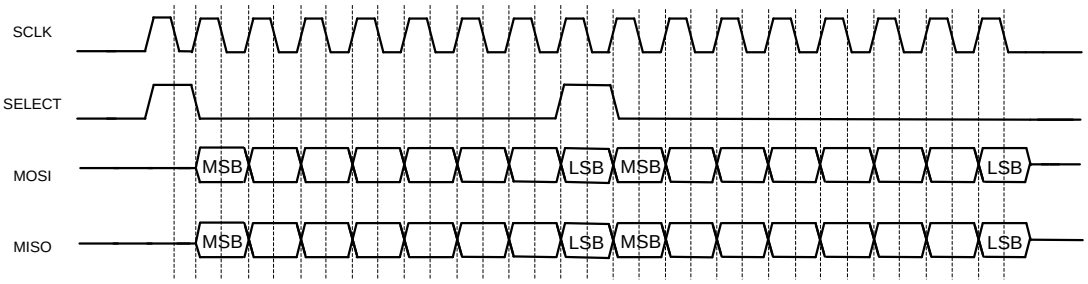
图 15-4 给出了单帧（8bit）数据传输和双帧（16bit）数据传输的时序图，此时 SELECT 上的帧指示脉冲相对于帧头提前了一个周期。

图 15-4 德州仪器 SPI 协议传输时序图 — 帧头指示脉冲比帧头提前一个周期

single data transfer



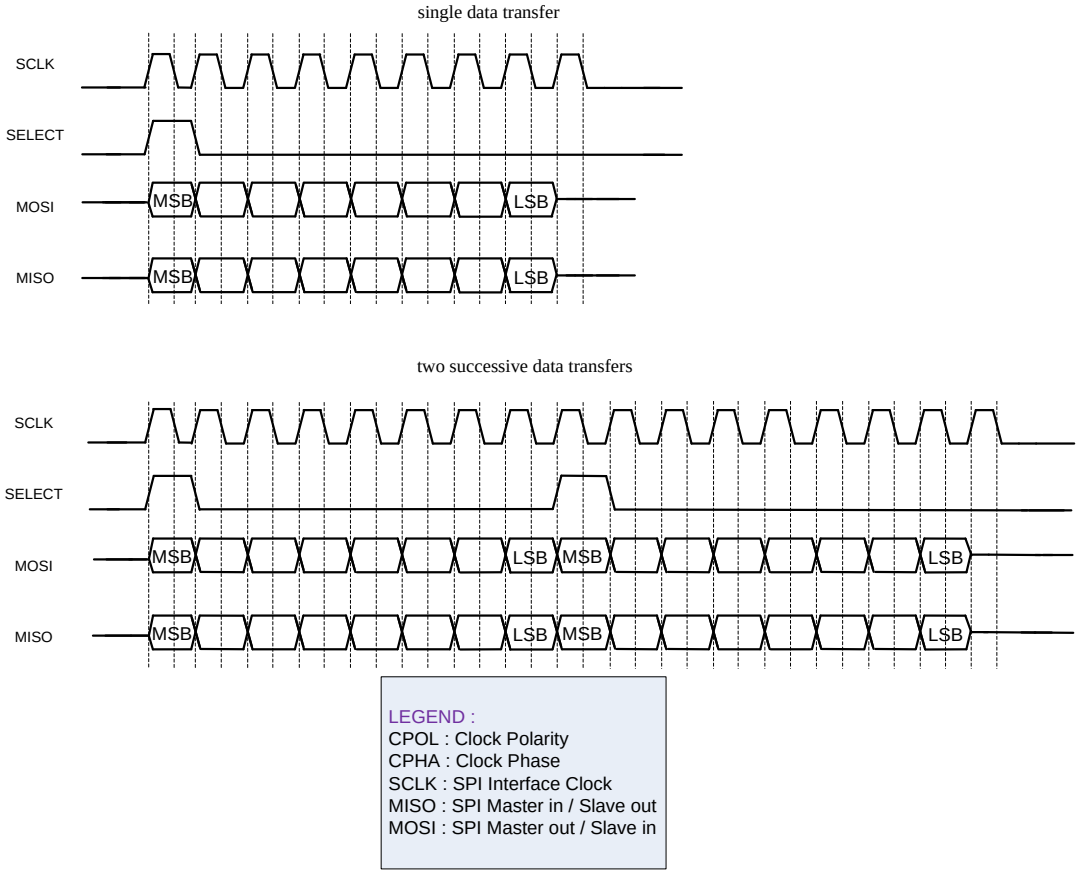
two successive data transfers



LEGEND :
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI Interface Clock
MISO : SPI Master in / Slave out
MOSI : SPI Master out / Slave in

图 15-5 给出了单帧（8bit）数据传输和连续两帧（16bit）数据传输的时序图，此时 SELECT 上的帧指示脉冲与帧头重合。

图 15-5 德州仪器 SPI 协议传输时序图 — 帧头指示脉冲与帧头重合



15.2.3.2.2 配置

通过下列的寄存器配置，可使 SCB 工作在德州仪器 SPI 协议下。

1. 将寄存器 MODE（SCB_CTRL[25:24]）设置为 01，选择 SPI 协议；
2. 将寄存器 MODE（SCB_SPI_CTRL[25:24]）设置为 01，进一步选择德州仪器 SPI 协议；
3. 配置寄存器 SELECT_PRECEDE（SCB_SPI_CTRL[1]），选择帧头指示脉冲出现的位置；
4. 根据章节 15.2.6（SPI 的初始化）中的步骤 2-4，对 SCB 实现的 SPI 接口控制器进行初始化和使能。

注意：上述的这些配置都可以在 PSoC Creator 中通过图形用户界面完成，我们更推荐这种设计方式。如果需要更多的了解寄存器的信息，请参考 PSoC 4 的寄存器用户手册。

15.2.4 国家半导体 SPI 协议

15.2.4.1 简介

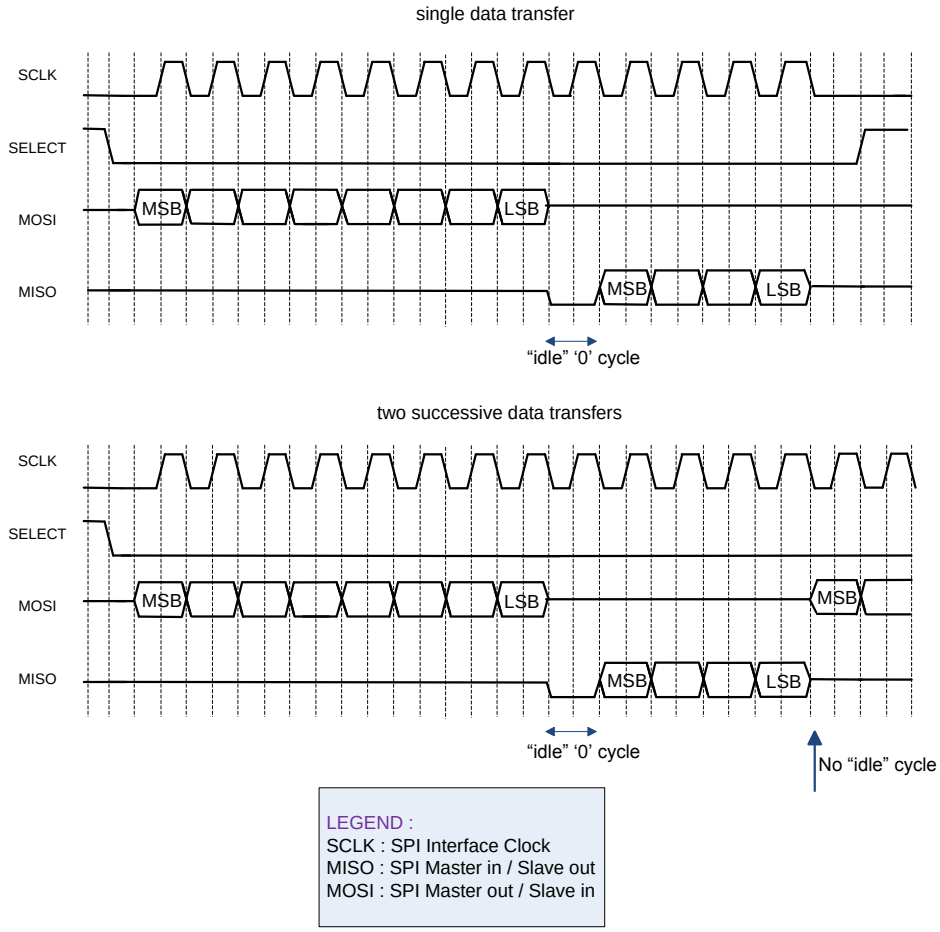
国家半导体 SPI 协议是一种半双工的协议，在同一时刻，只能发送或者接收数据。该版本的 SPI 协议仅支持模式 0，即 SCLK 空闲态为低，时钟上升沿采样数据，时钟下降沿发送数据。

图 15-7 给出了单帧数据传输和双帧数据传输的时序图，由时序图可知：

- 主设备发送的数据帧和接收的数据帧长度可以不相等。此处，发送的数据帧长度为 8 比特，接收的数据帧长度为 4 比特；
- 主设备通过 MOSI 向从设备传输数据完毕后，从设备需要等待一个空闲周期才可以通过 MISO 向主设备发送数据；
- 从设备通过 MISO 向主设备传输数据完毕后，主设备可以立刻通过 MOSI 向从设备发送数据，无需空闲周期。

空闲周期中，MISO 和 MOSI 上的数据无效。实际设计中，主从设备一般在空闲周期中将各自的数据输出信号线拉低。

图 15-6 国家半导体 SPI 协议中的数据传输时序图



15.2.4.2 配置

通过下列的寄存器配置，可使 SCB 工作在国家半导体 SPI 协议下。

1. 将寄存器 MODE（SCB_CTRL[25:24]）设置为 01，选择 SPI 协议；
2. 将寄存器 MODE（SCB_SPI_CTRL[25:24]）设置为 10，进一步选择国家半导体 SPI 协议；
3. 根据章节 15.2.6（SPI 的初始化）中的步骤 2-4，对 SCB 实现的 SPI 接口控制器进行初始化和使能。

注意：上述的这些配置都可以在 PSoC Creator 中通过图形用户界面完成，我们更推荐这种设计方式。如果需要更多的了解寄存器的信息，请参考 PSoC 4 的寄存器用户手册。

15.2.5 EZSPI (Easy SPI) — SPI 的 EZ 模式

EZSPI 是摩托罗拉 SPI 协议的一种演化，仅支持传输模式 0，即 SCLK 空闲态为低，时钟上升沿采样数据，时钟下降沿发送数据。在该协议下，主设备与从设备之间单帧数据的通信无需 CPU 的介入。从设备有一个 32 x 8bit 的存储器阵列，主设备可以通过地址寄存器 EZ_ADDR（SPI_STATUS[15:8]）中的 EZ 存储器地址对其寻址。

注意：SCB 中有一个基于 SRAM 的 16 x 16bit 的存储器。在非 EZ 模式中，该存储器作为 2 个 8 x 16bit 的 FIFO 队列使用：一个发送 FIFO 队列（TXFIFO）和一个接收 FIFO 队列（RXFIFO）。在 EZ 模式中，它作为一个 32 x 8bit 的 EZ 存储器使用。

EZSPI 中，数据传输的帧长度为 8bit，共有三种传输类型：写地址（主设备向从设备传输 EZ 存储器的地址），写数据（主设备向从设备传输数据），读数据（主设备由从设备读取数据）。

15.2.5.1 写地址

主设备首先通过 MOSI 向从设备传输命令字节 0x00，表明写地址的操作意图。从设备收到该信息后，在下一帧通过 MISO 向主设备传输响应字节，0xFE 表示响应，0xFF 表示不响应。在接收到 0xFE 之后，主设备向从设备传输 EZ 存储器的地址，该地址存放在地址寄存器 EZ_ADDR，指向 EZ 存储器 32 个字节中的某一个。

15.2.5.2 写数据

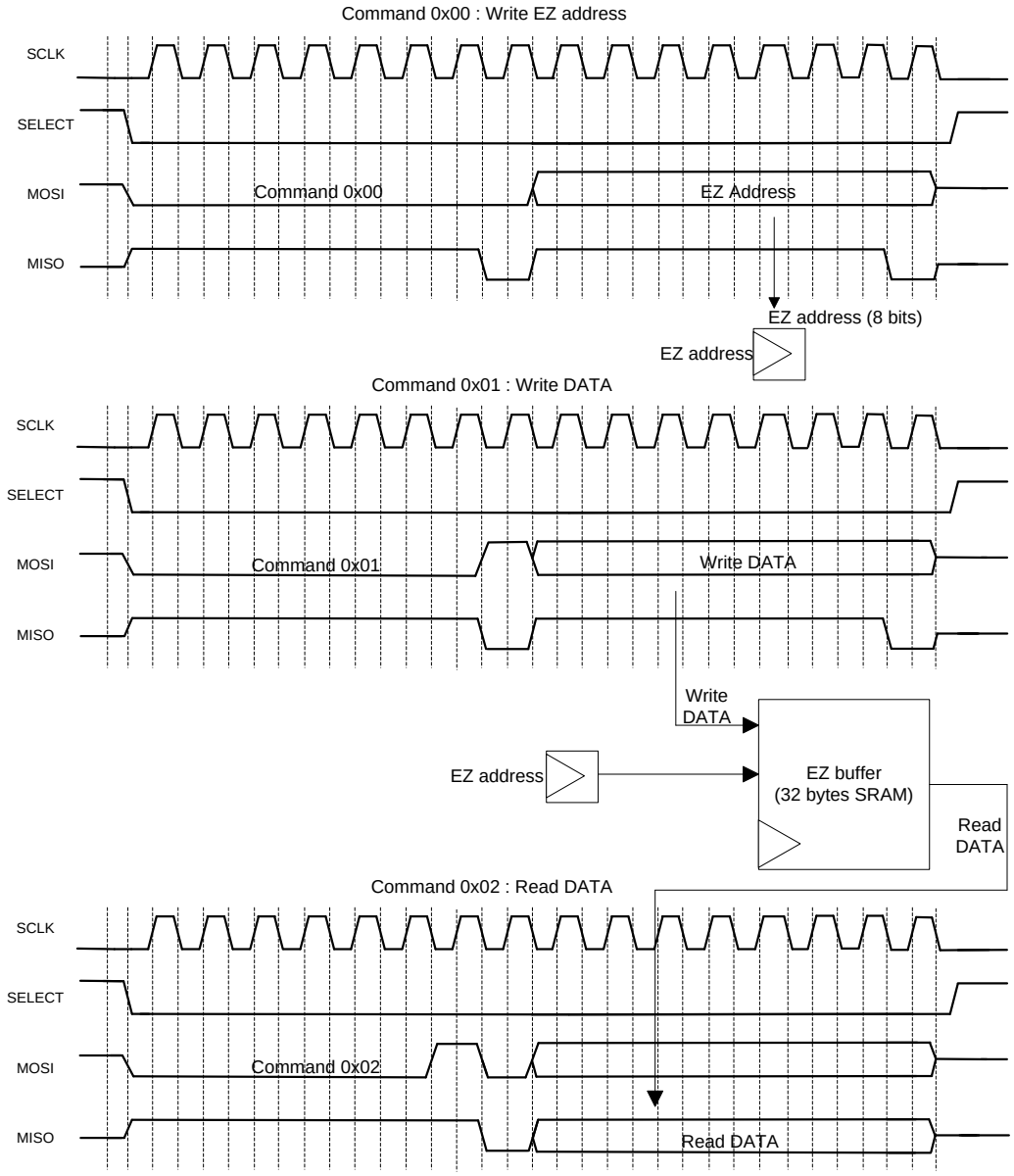
写数据的操作由主设备向从设备传输 0x01 的命令字节开始，从设备的响应机制同上。主设备得到相应后传输数据，从设备接收数据，并存放到 EZ_ADDR 指向的 EZ 存储器的字节空间。完成一个字节的传输后，EZ_ADDR 累加 1，如果累加结果超出 31，则返回 0。

15.2.5.3 读数据

读数据对应的命令字节为 0x02，它的传输标志着读数据操作的开始。从设备的响应机制同上。在相应主设备的读数据请求后，从设备将 EZ 存储器中 EZ_ADDR 指向的字节数据通过 MISO 传输到主设备，同时指针累加 1，指针的返回机制同上。

图 15-7 描述了 EZSPI 的三种传输类型。

图 15-7 EZSPI 的三种传输类型



LEGEND :

CPOL : Clock Polarity	0x00 : Write EZ address
CPHA : Clock Phase	0x01 : Write DATA
SCLK : SPI Interface Clock	0x02 : Read DATA
MISO : SPI Master in / Slave out	0xfe : “slave ready”
MOSI : SPI Master out / Slave in	0xff : “slave busy”

15.2.5.4 配置

通过下列的寄存器配置，可使 SCB 工作在 SPI 的 EZ 模式下。

1. 将寄存器 EZ_MODE（SCB_CTRL[10]）设置为 1，选择 EZ 模式；
2. 根据章节 15.2.6（SPI 的初始化）中的步骤 2-4，对 SCB 实现的 SPI 接口控制器进行初始化和使能。
3. 将寄存器 SCB_SPI_CTRL 中的 SCB_CONTINUOUS 置位，令发送器工作在连续发送模式；
4. 只有从设备能够工作在 EZSPI 模式，将寄存器 SCB_SPI_CTRL 中的 SCB_MASTER_MODE 清零，令 SCB 作为从设备；
5. 将寄存器 SCB_TX_CTRL 和 SCB_RX_CTRL 中的 SCB_DATA_WIDTH 值配置为‘0111’，将数据帧的长度设置为 8 比特；

注意：上述的这些配置都可以在 PSoC Creator 中通过图形用户界面完成，我们更推荐这种设计方式。如果需要更多的了解寄存器的信息，请参考 PSoC 4 的寄存器用户手册。

15.2.6 SPI 相关寄存器

SPI 协议相关的寄存器如表 15-1 所列，更多寄存器的信息请参考 PSoC 4 的寄存器用户手册。

表 15-1 SPI 相关寄存器

寄存器名称	描述
SCB_CTRL	使能或者禁止 SCB； 选择 SCB 工作的协议（SPI，UART 或者 I2C）； 选择内部时钟模式或外部时钟模式； 选择 EZ 模式或者非 EZ 模式；
SCB_STATUS	在 EZ 模式中，指示 32 x 8bit 的 EZ 存储器是否正在被外部时钟逻辑访问。
SCB_SPI_CTRL	选择主设备模式或从设备模式； 选择 SPI 协议（摩托罗拉，德州仪器或者国家半导体）； 工作在摩托罗拉 SPI 协议时，选择具体的数据传输模式（0，1，2 或 3）； 在德州仪器协议下，配置 SELECT 信号线上帧头指示脉冲出现的时间；
SCB_SPI_STATUS	总线状态标志位； EZ 存储器地址寄存器；
SCB_TX_CTRL	使能或者禁止 SPI 的发送； 定义数据传输帧的长度； 配置 MSB 在前或者 LSB 在前；
SCB_RX_CTRL	使能或者禁止 SPI 的接收； 定义数据传输帧的长度； 配置 MSB 在前或者 LSB 在前； 使能或者禁止接收前端的中值滤波器（使用该滤波器的前提是，采样率大于数据率）
SCB_TX_FIFO_WR	保存前一次压入发送 FIFO 队列的数据；
SCB_RX_FIFO_RD	保存前一次接收 FIFO 队列弹出的数据，数据弹出后将从 FIFO 队列中删除，相当于 POP 操作；
SCB_RX_FIFO_RD_SILENT	保存前一次接收 FIFO 队列弹出的数据，数据弹出后不从 FIFO 队列中删除，相当于 PEEK 操作；
SCB_EZ_DATA	EZ 存储器

15.2.7 SPI 的中断

SCB 在某些事件发生时，可以触发中断，用户可以在 PSoC Creator 中通过图形用户界面选择中断触发事件。设计中，PSoC Creator 可以自动生成中断服务程序，用户也可以自定义中断服务程序。与 SPI 相关的中断包括：

- SPI 数据传输完毕；
- SPI 收发器空闲；
- 外部时钟模式的中断（详见 SCB_INTR_SPI_EC）；
- 主设备模式的中断（详见 SCB_INTR_M）；
- 从设备模式的中断（详见 SCB_INTR_M）；
- TXFIFO 的中断（详见 SCB_INTR_TX）；
- RXFIFO 的中断（详见 SCB_INTR_RX）。

上述中断指向同一个中断向量，一旦其中一个未被屏蔽的中断被触发，则该中断向量被读取，CPU 转入中断服务程序处理该中断，查询中断寄存器进一步确认具体的中断触发事件。

15.2.8 SPI 的初始化

在 SCB 中，实现 SPI 的初始化步骤如下：

1. 配置寄存器 SCB_SPI_CTRL（如表 15-2），选择具体的 SPI 协议种类和主/从设备模式；
2. 通过寄存器 SCB_TX_CTRL 和 SCB_RX_CTRL（如表 15-3），配置通用发送器与接收器，具体包括：
   - 数据帧的长度；
   - 高有效位（MSB）在前或者低有效位（LSB）在前；
   - 使能发送器或者接收器；
3. 通过寄存器 SCB_TX_FIFO_CTRL 和 SCB_RX_FIFO_CTRL（如表 15-4），配置发送和接收的 FIFO 队列，具体包括：
   - 选择触发门限；
   - 清除 FIFO 队列和移位寄存器；
   - 暂时冻结发送器和接收器；
4. 通过寄存器 SCB_CTRL（如表 15-5），选择工作协议（SPI、UART 或者 I2C），使能 SCB 模块。

表 15-2 寄存器 SCB_SPI_CTRL

SCB_SPI_CTRL			
比特位	**寄存器名**	**值**	**描述**
[25:24]	MODE	00	摩托罗拉 SPI 协议
		01	德州仪器 SPI 协议
		10	国家半导体 SPI 协议
		11	预留
31	MASTER_MODE	0	主设备模式
		1	从设备模式

表 15-3 寄存器 SCB_TX_CTRL / SCB_RX_CTRL

SCB_TX_CTRL / SCB_RX_CTRL		
比特位	**寄存器名**	**描述**
[3:0]	DATA_ WIDTH	数据传输中帧的长度等于“DATA_WIDTH + 1”（如果应用于 UART 协议，不包括开始位，停止位和奇偶校验位）；该寄存器有效值的范围为 [3, 15]。
8	MSB_FIRST	1：MSB 在前； 0：LSB 在前；
31	ENABLED	使能或禁止发送器与接收器；在任何协议的数据传输中，都需要使能它们。

表 15-4 寄存器 SCB_TX_FIFO_CTRL / SCB_RX_FIFO_CTRL

SCB_TX_FIFO_CTRL / SCB_RX_FIFO_CTRL		
比特位	**寄存器名**	**描述**
[2:0]	TRIGGER _LEVEL	配置触发门限，如果 TXFIFO 中的数据个数大于该值或者 RXFIFO 中的数据个数小于该值，则会发生发送器触发事件或者接收器触发事件，该触发事件可以作为中断源。
16	CLEAR	置位时，发送器的 FIFO 或者接收器的 FIFO 及其移位寄存器将被清零，并保持无效状态。
17	FREEZE	置位时，发送器或者接收器被暂时冻结，对它们的读或写无效，此时 FIFO 队列的读写指针不会移动。

表 15-5 寄存器 SCB_CTRL

SCB_CTRL			
比特位	**寄存器名**	**值**	**描述**
[25:24]	MODE	00	I2C 协议
		01	SPI 协议
		10	UART 协议
		11	预留
31	ENABLED	0	SCB 禁止
		1	SCB 使能

如果需要对 SCB 进行配置，必须首先禁止 SCB，在配置完成后，重新使能 SCB，所有的配置将在使能后生效。注意重新使能 SCB 会带来重新的初始化，相关的状态也会丢失，如 FIFO 队列的数据。在配置中，使能 SCB 模块必须放在最后一步。

15.2.9 SPI 的时钟模式

对 SPI 和 I2C 协议，SCB 的操作可以支持内部时钟模式和外部时钟模式。内部时钟模式中，SCB 使用芯片本身提供的高频时钟（来自于 IMO）；外部时钟模式中，SCB 使用串行接口的同步时钟。在深度睡眠模式下，芯片内部不提供高频时钟，此时 SCB 的操作只能够工作在外部时钟模式。

当 SCB 为主设备时，所有的操作只能够工作在内部时钟模式；当 SCB 为从设备时，可以选择工作在内部时钟模式或者外部时钟模式。如果 SCB 的操作（对 SPI，从地址选择操作除外；对 I2C，从地址匹配操作除外）工作在外部时钟模式，其内部的 16 x 16bitSRAM 不能够支持 TXFIFO 或 RXFIFO 的功能，只能够作为 EZ 存储器，所以此时 SCB 实现的 SPI 或 I2C 只能工作在 EZ 模式。

在内部时钟模式中，SCB 的时钟来自于芯片内部的高频时钟，详情请参考时钟章节。此时 SCB 支持对接收的数据进行过采样，寄存器 OVS（SCB_CTRL[3:0]）定义了过采样率，过采样率（SCB 工作时钟除以接口速率）等于 OVS+1。SCB 在不同寄存器设置下对过采样率的限制如表 15-6 所列，更多详情请参考寄存器用户手册。

表 15-6 最小过采样率

寄存器设置	最小过采样率	最大比特率（SCB 工作时钟为 48MHz）
MEDIAN = 0；LATE_MISO_SAMPLE = 0	6	8 Mbps
MEDIAN = 0；LATE_MISO_SAMPLE = 1	3	16 Mbps
MEDIAN = 1；LATE_MISO_SAMPLE = 0	8	6 Mbps
MEDIAN = 1；LATE_MISO_SAMPLE = 1	4	12 Mbps

通过寄存器 SCB_CTRL 中的如下两位，可以配置 SCB 中操作的外部时钟模式或内部时钟模式：

- **EC_AM_MODE**：该位为 0 时，SPI 的从设备选择工作在内部时钟模式；该位为 1 时，SPI 的从设备选择工作在外部时钟模式。
- **EC_OP_MODE**：该位为 0 时，SPI 的其它协议操作工作（包括所有的数据传输）在内部时钟模式；该位为 1 时，SPI 的其它协议操作工作在外部时钟模式。

这两个寄存器位共同决定了 SCB 所实现 SPI 控制器的行为特性，在 PSoC 4 不同的电源模式下，它们必须被正确的配置，否则会导致错误的操作。

15.2.9.1 非 EZSPI 的时钟模式

在非 EZ 模式的 SPI 协议中，EC_OP_MODE 必须设置为 0；而 EC_AM_MODE 可以设置为 0 或者 1。其它的设置是不被支持或者无效的。如表 15-7 所示。

表 15-7 非 EZ 模式 SPI 协议的时钟模式设置

非 EZ 模式的 SPI 协议				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
电源模式	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
活动和睡眠	从设备选择和其它协议操作均工作在内部时钟模式（适用于主设备和从设备）	从设备选择工作在外部时钟模式，其它协议操作工作在内部时钟模式，唤醒中断被屏蔽（仅适用于从设备）	不支持	不支持
深度睡眠	不支持	从设备选择工作在外部时钟模式，一旦从设备选择信号有效，产生中断唤醒 CPU（仅适用于从设备）	不支持	
休眠	SCB 在这两种电源模式下无法工作（请参考电源模式章节）			
停止				

EC_AM_MODE = 0，EC_OP_MODE = 0：该配置仅在活动和睡眠电源模式中有效。此时，SCB 实现的 SPI 控制器完全工作在内部时钟模式，能够提供所有的功能。

EC_AM_MODE = 1，EC_OP_MODE = 0：该设置在活动、睡眠和深度睡眠的电源模式中均有效。当外部时钟逻辑检测到从设备选择信号有效时，SCB 的唤醒中断请求位 WAKE_UP（SCB_INTR_SPI_EC[0]）将被置位，它可以用于唤醒 CPU。

- 在活动和睡眠电源模式中，从设备选择工作在外部时钟模式（由于此时内部时钟也存在，从设备选择信号也会被内部时钟逻辑采样）；其它协议操作工作在内部时钟模式。在活动模式中，唤醒中断请求被屏蔽（SCB_INTR_SPI_EC_MASK[0] = 0）；在睡眠模式中，唤醒中断是否被屏蔽，用户可以自己设置。
- 在深度睡眠电源模式中，唤醒中断请求未被屏蔽（SCB_INTR_SPI_EC_MASK[0] = 1）。当从设备选择信号有效时，SCB 会触发中断唤醒 CPU。唤醒过程需要一定的时间，所以当前数据帧的传输不被从设备响应（从设备由 SOMI 向主设备传输 0xff，主设备会重发此帧）。唤醒以后，从设备选择工作在外部时钟模式；其它协议操作工作在内部时钟模式。

15.2.9.2 EZSPI 的时钟模式

在 EZ 模式的 SPI 协议中，当 EC_OP_MODE 为 0 时，EC_AM_MODE 可以是 0 或者 1；当 EC_OP_MODE 为 1 时，EC_AM_MODE 必须为 1。如表 15-8 所示，表中的灰色单元格表示该配置（EC_AM_MODE = 1，EC_OP_MODE = 0）是有效的，但是不推荐使用，因为这涉及到从外部时钟逻辑向内部时钟逻辑切换的问题。

表 15-8 EZ 模式 SPI 协议的时钟模式设置

SPI, EZ mode				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
电源模式	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
活动和睡眠	从设备选择和其它协议操作均工作在内部时钟模式（仅适用于从设备）	从设备选择工作在外部时钟模式，其它协议操作工作在内部时钟模式，唤醒中断被屏蔽（仅适用于从设备）	从设备选择和其它协议操作均工作在外部时钟模式（仅适用于从设备）	不支持
深度睡眠	不支持	从设备选择工作在外部时钟模式，一旦从设备选择信号有效，产生中断唤醒 CPU（仅适用于从设备）	从设备选择和其它协议操作均工作在外部时钟模式（仅适用于从设备）	
休眠	SCB 在这两种电源模式下无法工作（请参考电源模式章节）			
停止				

EC_AM_MODE = 0，EC_OP_MODE = 0：该配置仅在活动和睡眠电源模式中有效。此时，SCB 实现的 SPI 控制器完全工作在内部时钟模式，能够提供所有的功能。

EC_AM_MODE = 1，EC_OP_MODE = 0：EZSPI 的该时钟模式与非 EZSPI 中对应的时钟模式相同。

EC_AM_MODE = 1，EC_OP_MODE = 1：该设置在活动、睡眠和深度睡眠的电源模式中均有效。

- 在活动和睡眠电源模式中，从设备选择和其它协议操作均工作在外部时钟模式。由于此时内部时钟也存在，如果外部时钟逻辑和内部时钟逻辑同时访问 EZ 存储器，就会引起冲突。通过将寄存器位 BLOCK（SCB_CTRL[17]）置位，可以阻止内部时钟逻辑对 EZ 存储器的访问，避免冲突。如果未被阻止，并且与外部时钟逻辑的访问发生了冲突，那么内部时钟逻辑的访问会发生异常（读操作返回 0xFFFFFFFF，写操作被忽略），并且触发一个访问冲突中断请求，寄存器位 BLOCKED（SCB_INTR_TX[7] 和 SCB_INTR_RX[7]），该中断请求时可以被屏蔽的。
- 在深度睡眠模式中，从设备选择和其它协议操作均工作在外部时钟模式。从设备中的 EZ 存储器完全由外部主设备操作。

15.3 通用异步收发（UART）协议

通用异步收发协议（UART）定义了一种串行异步接口，其拓扑一般是点到点的结构。UART 是一种全双工的协议，主要有两个信号：

- TX：发送器输出；
- RX：接收器输入；

15.3.1 特性

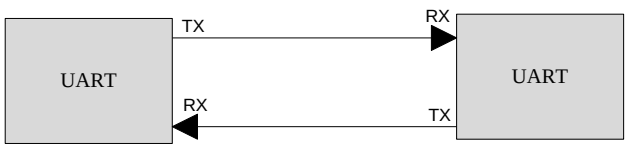
SCB 实现 UART 协议时，有如下特性：

- 串行数据收发；
- 数据传输率最高为 1Mbps；
- 支持多种 UART 协议；
  - 标准 UART 协议；
  - SmartCard（ISO7816）协议，较前者加入了错误响应机制；
  - 红外串行数据（IrDA）协议，对普通 UART 信号进行红外调制和解调；
- 支持 LIN（Local Interconnect Network）的主从模式
  - Break 检测；
  - 波特率检测；
  - 冲突检测；
- 支持多处理器模式；
- 数据帧长度可配置，范围从 4 比特到 16 比特（不包括开始位，停止位和奇偶校验位）；
- 停止位个数可配置；
- 支持奇偶校验；
- 支持中断和轮询两种方式与 CPU 通信；
- 支持对接收的数据进行过采样。

15.3.2 概述

图 15-8 给出了一个 UART 通信的示例。

图 15-8 UART 通信示例



UART 协议中一个典型的数据帧由开始位，数据位，奇偶校验位和停止位组成。开始位的逻辑值为 0，它标志一个数据帧的开始；数据位是需要传输的信息；奇偶校验位是可选的，它的值取决于所有数据位逻辑值求和结果的奇偶性；停止位的逻辑值为 1，它的长度可配置。当 UART 总线处于空闲态时，其逻辑值为 1（电路中通过电阻上拉实现），如同停止位。

UART 是一个异步的通信协议，收发两端的设备都使用各自内部的时钟接收和发送数据，如图 15-8，它们必须在数据传输的波特率上保持一致。

SCB 支持多种 UART 协议，如下：

- 标准 UART 协议；
- SmartCard 协议（ISO7816），与前者类似，但是加入了错误响应机制；
- 红外串行数据（IrDA）协议，对普通 UART 信号进行了红外调制和解调；

UART 数据传输帧长度的配置范围是 4 比特到 9 比特，缺省配置下为 8 比特，开始位、停止位和校验位不计入数据帧的长度。停止位的长度可配置为 1 比特、2 比特或 3 比特。在标准 UART 协议的非多处理器模式和 SmartCard 协议中，校验位是可选的，如果使用，可以是偶检验或者奇校验；在标准 UART 协议的多处理器模式和 IrDA 协议中，检验位是被禁止的。

注意：由于 UART 设备只能工作在内部时钟模式，所以只能工作在活动和睡眠工作模式。

15.3.3 UART 协议介绍

15.3.3.1 标准 UART 协议

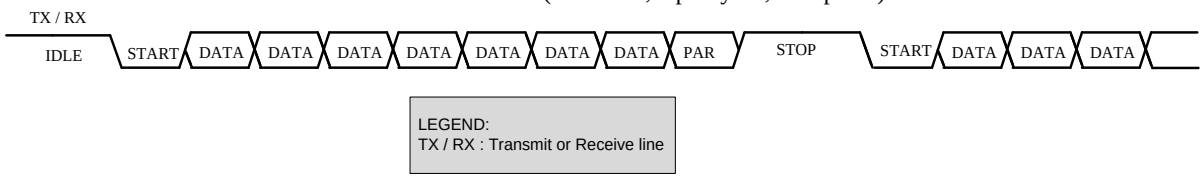
UART 协议中一个典型的数据帧由开始位，数据位，奇偶校验位和停止位组成。开始位的逻辑值为 0，它标志一个数据帧的开始；数据位是需要传输的信息；奇偶校验位是可选的，它的值取决于所有数据位逻辑值求和结果的奇偶性；停止位的逻辑值为 1，它的长度可配置。当 UART 总线处于空闲态时，其逻辑值为 1，如同停止位。

由于 UART 协议是异步的，所以发送器和接收器必须约定一个比特位的时间长度，它的倒数即为波特率。发送器和接收器拥有各自的工作时钟，如果接收器的工作时钟频率高于波特率，那么可以对接收到的数据进行过采样，并应用数字滤波的手段来提高信噪比。

从停止位（1）到开始位（0）有一个下降沿。通过每个数据帧传输开始的下降沿，接收器的时钟可以与发送器的时钟进行同步。这个同步机制可以弥补收发时钟之间频偏带来的影响，对频偏的容忍度取决于数据帧的长度以及接收器对数据建立保持时间的要求。

数据帧中停止位的长度是可编程的（范围 1bit – 3bit），但需要发送器与接收器在处理上达成一致。图 15-9 给出了一个数据帧的结构，它包含 1 个开始位，7 个数据位，1 个奇偶检验位和 2 个停止位。

图 15-9 标准 UART 协议中的一个数据帧



若接收器的数据采样时钟高于波特率，则接收器可以对一个比特位做一次采样，如图 15-10 所示；也可以对一个比特位做多次采样，如图 15-11 所示，即过采样。后者可以提高信噪比，降低误码率。

图 15-10 单次采样

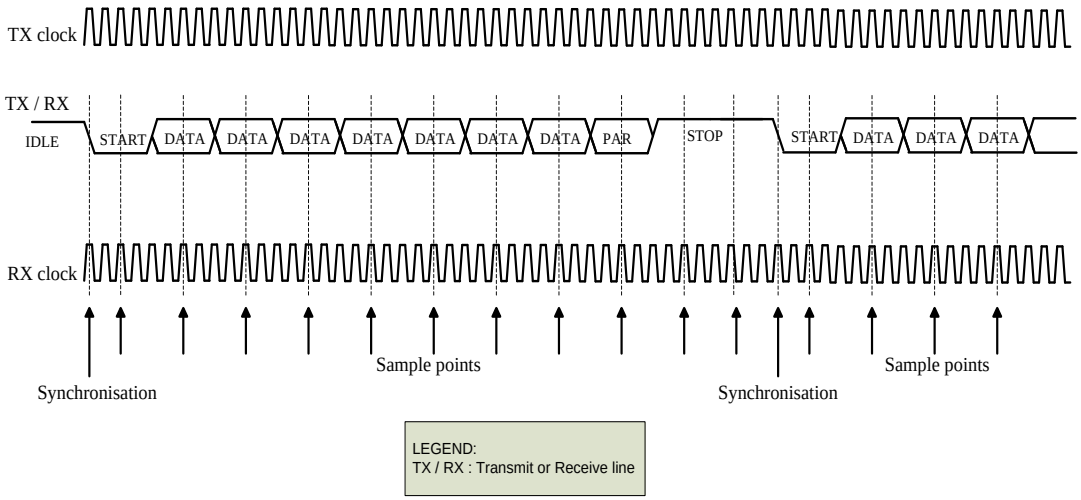
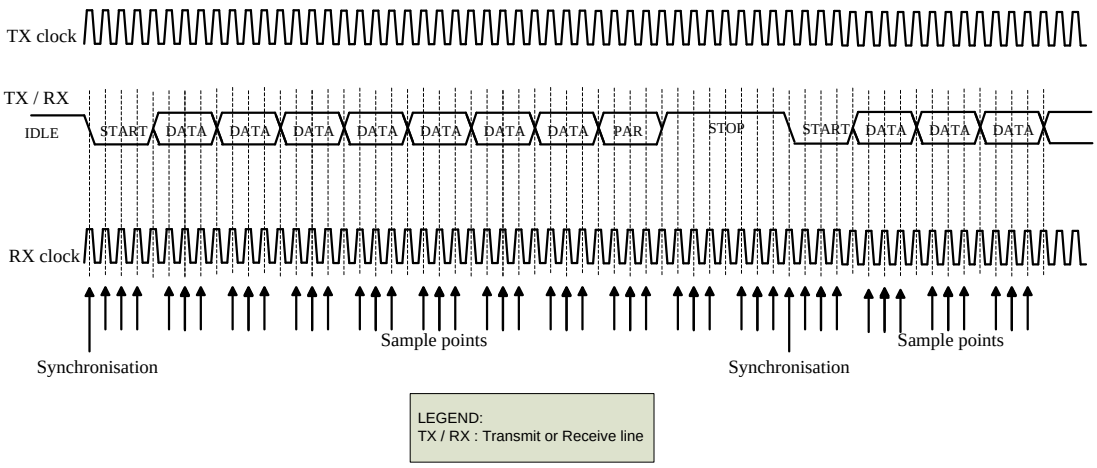


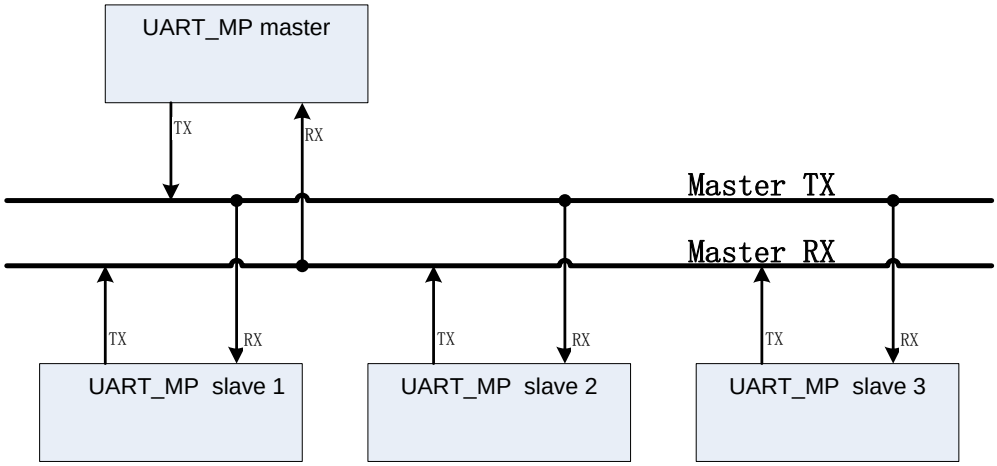
图 15-11 多次采样（过采样）



多处理器模式（UART_MP）

UART_MP（Multi-Processor）模式针对一主多从的应用，图 15-12 给出了拓扑。UART_MP 模式是标准 UART 协议的一部分，也被称为 9 比特 UART，该模式下的数据帧的长度始终为 9 比特。

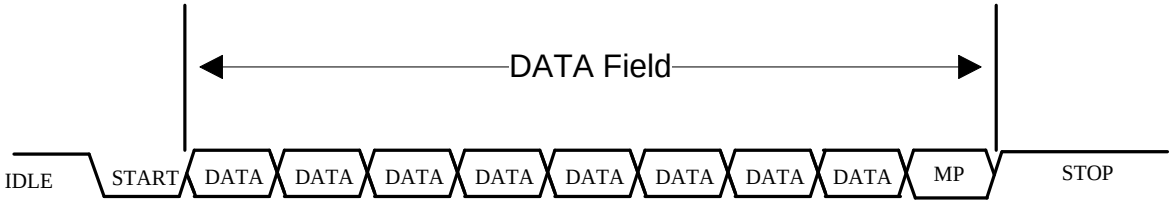
图 15-12 UART 多处理器模式的拓扑



UART_MP 主要特性如下：

- 一主多从的多节点网络拓扑结构；
- 每个从设备有唯一的地址；
- 数据帧的长度为 9 比特，由 8 比特数据和 1 比特地址/数据标志位（MP）组成，具体结构如图 15-13 所示。MP = 1 时，对应数据帧中的数据表示一个从设备的地址；MP = 0 时，对应数据帧中的数据表示一个字节的数据。
- 无奇偶校验位。

图 15-13 UART_MP 中数据帧的结构



在 UART_MP 中，SCB 可以作为主设备或者从设备。此时，将 MP_MODE（SCB_UART_RX_CTRL[10]）置位，并配置寄存器 DATA_WIDTH（SCB_TX_CTRL[3:0]和 SCB_RX_CTRL[3:0]），将数据帧的长度设置为 9bit。当 SCB 作为主设备时，数据帧中的 MP 需要根据当前帧中数据的意义（数据或地址）而改变。当 SCB 作为从设备时，需要通过寄存器 SCB_RX_MATCH 来配置从设备的地址（包括一个地址寄存器和一个地址屏蔽寄存器）。

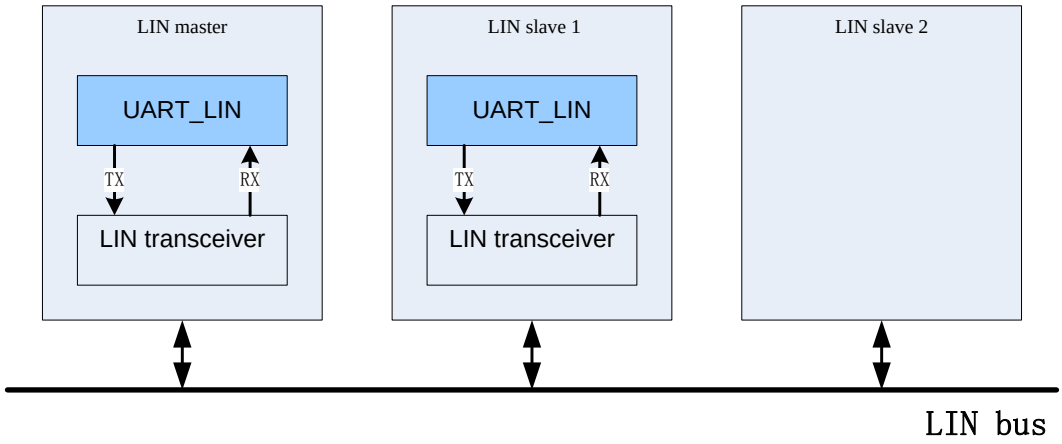
数据传输中，若从设备检测到接收到的地址与自身的不匹配，则忽略后面的数据。当检测到地址匹配，从设备开始接收数据。如果寄存器 ADDRESS_ACCEPT（SCB_CTRL[16]）为 1，则匹配的地址会先于数据被写入 RXFIFO。

本地互连网络模式（LIN）

LIN 模式同样针对一主多从的应用，与前者不同的是，LIN 总线的宽度为 1 比特，TX 和 RX 通过 LIN 收发器共用一根数据总线，如图 15-14 所示。在 LIN 中，主节点包含主任务和从任务；而从节点只有从任务。更多相关信息请参考“LIN 规范”。

SCB 能够作为 LIN 总线中的主节点或者从节点，总线拓扑如图 15-14 所示。

图 15-14 LIN 总线的拓扑结构



在 LIN 总线上，数据传输是以帧的形式进行的，具体如下。

- 帧头：由主任务发送到从任务，在主任务的 TX 信号线和从节点的 RX 信号线上传输。帧头具体由以下几个场组成：
  - 间隔场：持续不少于 13 个比特周期的逻辑 0；
  - 同步场：数据值为 0x55，用于同步主任务和从任务的时钟；
  - 标识符场：包含于指定从任务通信的内容。
- 帧相应：有从任务发送到主任务，在主任务的 RX 信号线和从节点的 TX 信号线上传输。

LIN 总线中的数据传输有主任务负责发起，从任务负责响应。当 SCB 作为从节点时，提供波特率检测（在同步场中进行）和间隔场长度检测。除间隔场，其它传输都是以 8 比特长度（不包含开始位和停止位）的帧格式进行的。

为了兼容 LIN 总线的物理层要求（总线宽度和电平标准），SCB 必须通过 LIN 收发器才可以被应用于 LIN 总线的通信。在通信中，SCB 可以通过比较 RX 和 TX 信号线上的数据来判断是否有总线冲突发生，如果 RX 和 TX 信号线上的数据不同，则寄存器 UART_ARB_LOST（SCB_INTR_TX[10]）被置位，触发一个总线冲突中断。

配置

通过下列的寄存器配置，可使 SCB 工作在标准 UART 协议下。

1. 将 SCB_MODE（SCB_CTRL[25:24]）设置为 10，选择 UART 协议；
2. 将 MODE（SCB_UART_CTRL[25:24]）设置为 00，进一步选择标准 UART 协议；
3. 如果要使能 UART_MP 模式，将寄存器 UART_MP_MODE（SCB_UART_RX_CTRL[10]）置位；如果要使能 LIN 模式，将寄存器 UART_LIN_MODE（SCB_UART_RX_CTRL[11]）置位；
4. 根据章节 15.3.6（UART 的初始化）中的步骤 2-4，对 SCB 实现的 UART 控制器进行初始化和使能。

注意：上述的这些配置都可以在 PSoC Creator 中通过图形用户界面完成，我们更推荐这种设计方式。如果需要更多的了解寄存器的信息，请参考 PSoC 4 的寄存器用户手册。

15.3.3.2 SmartCard (ISO7816) 协议

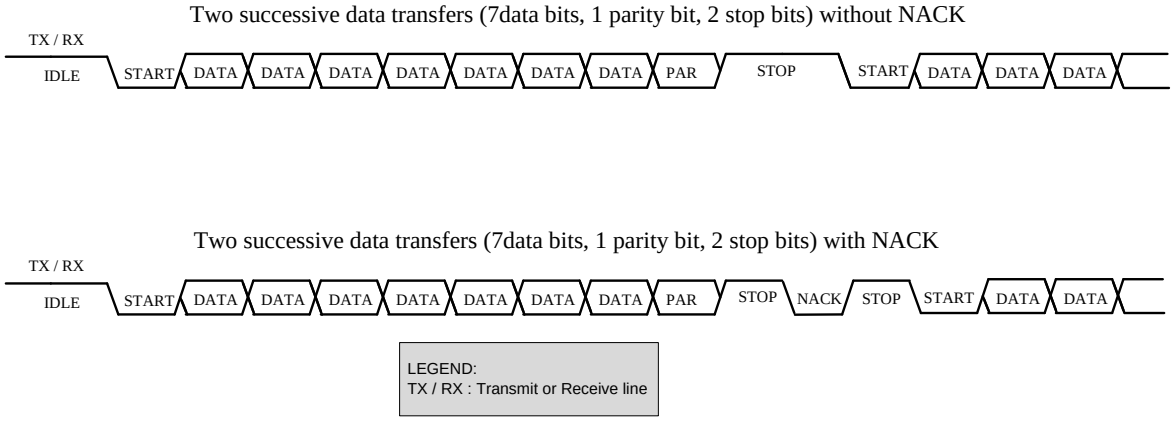
15.3.3.2.1 简介

SmartCard (ISO7816) 是一个异步串行通信协议，针对一主多从的应用。SCB 只能作为 SmartCard 协议中的主设备（读卡器）。SCB 可以进行异步的字符通信，为 SmartCard 协议提供基本的物理层支持。SCB 的 TX 信号线在 SmartCard 协议中作为 IO 信号线，此时 SCB 内部的发送与接收模块通过多路器分时连接到 TX 信号线上。

SmartCard 协议的数据帧与标准 UART 数据帧类似，额外加入了否定应答机制 (NACK)，它的逻辑值为 0。

图 15-15 给出了 SmartCard 协议中的两种数据帧格式，一种不带 NACK，另一种带 NACK。如图所示，发送器首先驱动 IO 信号线，依次传输开始位，数据位，奇偶校验位，然后释放对信号线的控制（由于上拉电阻的关系，信号线的逻辑值为 1，相当发送器传输停止位），此时会有两种情况发生。第一种情况，接收器不传输 NACK，停止位持续两个比特位长度后，发送器开始下个数据帧的传输；第二种情况，接收器在停止位持续一个比特位长度后，拉低信号线，传输 NACK 到发送器，表示否定应答。发送器接收到 NACK，再传输一个比特位长度的停止位，开始下个数据帧的传输。在 SmartCard 协议中，停止位的长度需要大于一个比特。

图 15-15 SmartCard 协议中数据帧的结构



在 SmartCard 协议中，串行通信的波特率计算方法如下：

Baud rate= f_{7816} * (D / F)

其中

f_{7816} 是工作时钟的频率；

F 是时钟速率转换系数；

D 是波特率调整系数；

举例说明：当 F=372，D=1，工作时钟选择为 3.57MHz，则波特率为 9600。

15.3.3.2.2 配置

通过下列的寄存器配置，可使 SCB 工作在 SmartCard 协议下。

1. 将寄存器 SCB_MODE（SCB_CTRL[25:24]）设置为 10，选择 UART 协议；
2. 将寄存器 SCB_UART_MODE（SCB_UART_CTRL [25:24]）设置为 01，进一步选择 SmartCard 协议；
3. 根据章节 15.3.6 中的步骤 2-4，对 SCB 实现的 UART 控制器进行初始化和使能。

注意：上述的这些配置都可以在 PSoC Creator 中通过图形用户界面完成，我们更推荐这种设计方式。如果需要更多的了解寄存器的信息，请参考 PSoC 4 的寄存器用户手册。

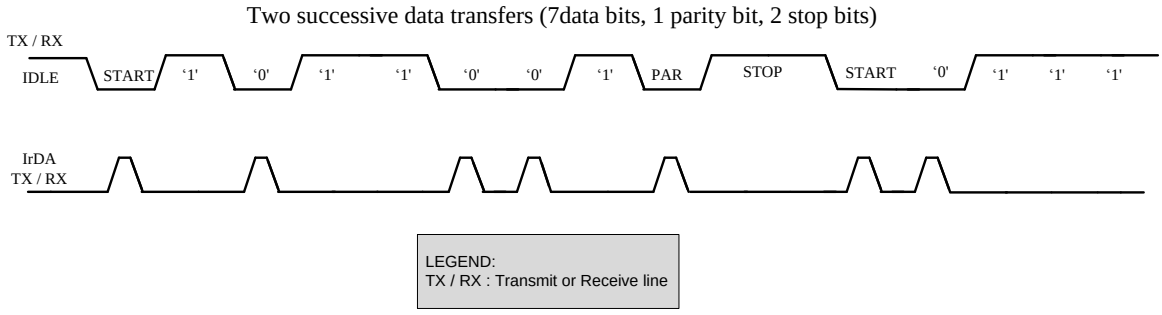
15.3.3.3 IrDA 协议

15.3.3.3.1 简介

IrDA 协议与标准 UART 协议在数据链路层上相同，只是在物理层上对信号做了红外调制和解调。这里的调制采用了 RZI（Return-to-Zero-Inverted）格式，即数据线上的逻辑 0 被调制为一个高脉冲，脉冲宽度取决于波特率；逻辑 1 被调制为零电平。SCB 支持红外串行数据协议物理层规格的 1.4 版本，最高波特率可达 115200。

SCB 实现的 IrDA 协议支持的波特率包括 2400、9600、19200、38400、57600 和 115200，高脉冲的宽度为 3/16 的比特位时间长度。通过寄存器 OVS（SCB_CTRL[3:0]），数据采样时钟应当被设置为波特率的 16 倍。

图 15-16 IrDA 协议中数据帧的结构



15.3.3.3.2 简介

通过下列的寄存器配置，可使 SCB 工作在 IrDA 协议下。

1. 将寄存器 SCB_MODE（SCB_CTRL[25:24]）设置为 10，选择 UART 协议；
2. 将寄存器 SCB_UART_MODE（SCB_UART_CTRL [25:24]）设置为 10，进一步选择 IrDA 协议；
3. 根据章节 15.3.6 中的步骤 2-4，对 SCB 实现的 UART 控制器进行初始化和使能。

注意：上述的这些配置都可以在 PSoC Creator 中通过图形用户界面完成，我们更推荐这种设计方式。如果需要更多的了解寄存器的信息，请参考 PSoC 4 的寄存器用户手册。

15.3.4 UART 相关寄存器

UART 协议相关的寄存器如表 15-9 所列，更多寄存器的信息请参考 PSoC 4 的寄存器用户手册

表 15-9 UART 相关寄存器

寄存器名称	描述
SCB_UART_CTRL	选择标准 UART 协议、SmartCard 协议或者 IrDA 协议； 本地自环控制；
SCB_UART_RX_STATUS	指示一个比特周期包含外设时钟周期的个数（检测波特率时有效）
SCB_UART_TX_CTRL	配置停止位的个数； 使能或者禁止奇偶校验位； 选择奇校验或者偶校验； 使能或者禁止数据帧的重发（仅适用于 SmartCard 协议中 RX 端发送 NACK 时）
SCB_UART_RX_CTRL	配置停止位的个数； 使能或者禁止奇偶校验位； 选择奇校验或者偶校验； 选择 UART_MP 模式； 选择 UART_LIN 模式； 配置校验错误时是否丢弃当前数据帧
SCB_TX_CTRL	使能或者禁止发送器； 选择数据帧的长度； 选择 MSB 在前或者 LSB 在前；
SCB_RX_CTRL	使能或者禁止接收器； 选择数据帧的长度； 选择 MSB 在前或者 LSB 在前； 使能或者禁止中值滤波器（对 IrDA 协议，必须始终使能）

15.3.5 UART 的中断

SCB 在某些事件发生时，可以触发中断，用户可以在 PSoC Creator 中通过图形用户界面选择中断触发事件。设计中，PSoC Creator 可以自动生成中断服务程序，用户也可以自定义中断服务程序。与 UART 相关的中断包括：

- UART 发送完毕；
- UART 的发送器接收到 NACK（SmartCard 协议）；
- UART 仲裁丢失（LIN 和 SmartCard 协议）；
- 帧传输错误
- 奇偶校验失败
- 波特率检测完成（LIN 协议）
- 间隔场检测成功（LIN 协议）

上述中断指向同一个中断向量，一旦其中一个未被屏蔽的中断被触发，则该中断向量被读取，CPU 转入中断服务程序处理该中断，查询中断寄存器进一步确认具体的中断触发事件。

15.3.6 UART 的初始化

在 SCB 中，实现 UART 的初始化步骤如下：

1. 配置寄存器 SCB_UART_CTRL（如表 15-10），选择具体的 UART 协议种类，进行环回配置；
2. 通过寄存器 SCB_TX_CTRL 和 SCB_RX_CTRL（如表 15-11），配置通用发送器与接收器，具体包括：
   - 数据帧的长度；
   - 高有效位（MSB）在前或者低有效位（LSB）在前；
   - 使能发送器或者接收器；
3. 通过寄存器 SCB_TX_FIFO_CTRL 和 SCB_RX_FIFO_CTRL（如表 15-12），配置发送和接收的 FIFO 队列，具体包括：
   - 选择触发门限；
   - 清除 FIFO 队列和移位寄存器；
   - 暂时冻结发送器和接收器；
4. 通过寄存器 SCB_CTRL（如表 15-13），选择工作协议（SPI、UART 或者 I2C），使能 SCB 模块。

表 15-10 寄存器 SCB_UART_CTRL

SCB_UART_CTRL			
比特位	**寄存器名**	**值**	**描述**
[25:24]	MODE	00	标准 UART 协议
		01	SmartCard 协议
		10	IrDA 协议
		11	预留
16	LOOP_BACK	自环回配置，置位时有效，发送器的输出直接由内部连接到接收器的输入。	

表 15-11 寄存器 SCB_TX_CTRL / SCB_RX_CTRL

SCB_TX_CTRL / SCB_RX_CTRL		
比特位	**寄存器名**	**描述**
[3:0]	DATA_ WIDTH	数据传输中帧的长度等于“DATA_WIDTH + 1”（如果应用于 UART 协议，不包括开始位，停止位和奇偶校验位）； 该寄存器有效值的范围为 [3, 15]。
8	MSB_FIRST	1：MSB 在前； 0：LSB 在前；
31	ENABLED	使能或禁止发送器与接收器；在任何协议的数据传输中，都需要使能它们。

表 15-12 寄存器 SCB_TX_FIFO_CTRL / SCB_RX_FIFO_CTRL

SCB_TX_FIFO_CTRL / SCB_RX_FIFO_CTRL		
比特位	**寄存器名**	**描述**
[2:0]	TRIGGER_LEVEL	配置触发门限，如果 TXFIFO 中的数据个数大于该值或者 RXFIFO 中的数据个数小于该值，则会发生发送器触发事件或者接收器触发事件。
16	CLEAR	置位时，发送器的 FIFO 或者接收器的 FIFO 及其移位寄存器将被清零，并保持无效状态。
17	FREEZE	置位时，发送器或者接收器被暂时冻结，对它们的读或写无效，此时 FIFO 队列的读写指针不会移动。

表 15-13 寄存器 SCB_CTRL

SCB_CTRL			
比特位	**寄存器名**	**值**	**描述**
[25:24]	MODE	00	I2C 协议
		01	SPI 协议
		10	UART 协议
		11	预留
31	ENABLED	0	SCB 禁止
		1	SCB 使能

如果需要对 SCB 进行配置，必须首先禁止 SCB，在配置完成后，重新使能 SCB，所有的配置将在使能后生效。注意重新使能 SCB 会带来重新的初始化，相关的状态也会丢失，如 FIFO 队列的数据。在配置中，使能 SCB 模块必须放在最后一步。

15.4 I2C 协议

I2C（Inter Integrated Circuit）是一种两线制的串行通信协议，被广泛应用于嵌入式设备中芯片之间的低速互连通路，如单片机，ADC，DAC，EEPROM，温度传感器，电源检测器等。

15.4.1 特性

SCB 实现 I2C 协议时，有如下特性：

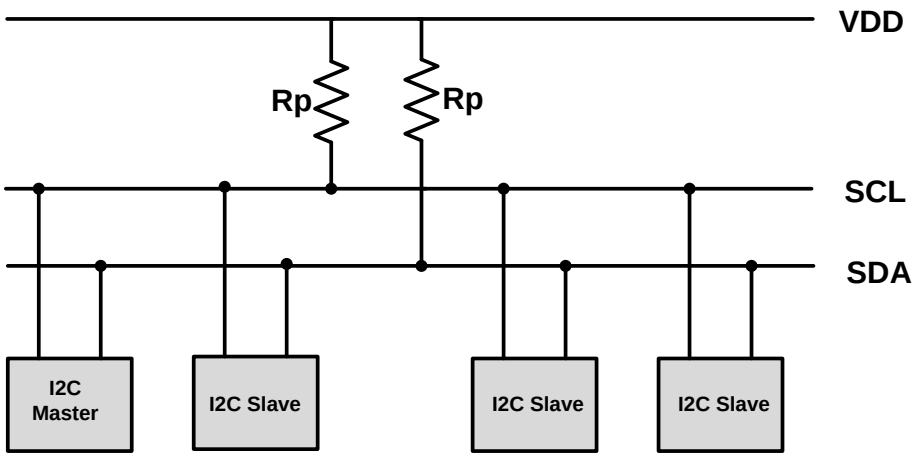
- 支持主设备，从设备和主从设备模式；
- 支持的速度等级包括：慢速模式（50 kbps），标准速度模式（100 kbps），快速模式（400 kbps）和超快速模式（1000 kbps）；
- 支持 7 比特或者 10 比特的从地址空间；
- 支持时钟延展和冲突检测；
- 支持对接收数据进行过采样；
- 通过中值滤波器提高通信的可靠性；
- 10ns 或 50ns 的毛刺滤波器；
- 支持 EZ 模式；
- 从设备支持外部时钟模式；
- CPU 通过中断或轮询与 SCB 通信；

如果使用 PSoC 4 的专用 SCB 模块来实现一个 I2C 接口，那么 SCL 和 SDA 可以连接到哪些 IO 口上是有限制，具体信息请查询器件手册。如果利用 PSoC 4 的 UDB 资源来实现一个 I2C 接口，那么 SCL 和 SDA 可以通过 DSI 互连到任何一个 IO 口上。被用于 I2C 接口的 IO 口不能够同时用于其他的用途。

15.4.2 概述

图 15-17 给出了 I2C 总线的示意图。

图 15-17 I2C 总线



- I2C 总线一共有两根信号线：
  - 串行数据线（SDA）
  - 串行时钟线（SCL）
- 同步串行数据传输，半双工，速率可配置；
- I2C 设备以集电极开路（漏极开路）的方式连接到总线上，总线上需要接上拉电阻；
- 一条总线上的每个 I2C 从设备拥有独一无二的 7bit 地址，软件可寻址，通过软件的支持，PSoC 4 也支持 10 比特地址的寻址；
- 主设备和从设备均可作为发送器和接收器；
- 支持多主设备，当多个主设备同时发起数据传输时，启动冲突检测和总线仲裁机制。

表 15-14 I2C 总线相关的术语

术语	描述
发送器	向总线发送数据的设备
接收器	从总线接收数据的设备
主设备	负责发起数据传输的设备，提供同步时钟，并主导数据传输的开始与结束
从设备	被主设备选址的设备
多主设备	一条总线上有多个主设备的应用，需要启动冲突检测和总线仲裁机制
总线仲裁	当总线上多个主设备同时发起数据传输时，总线仲裁机制保证只有一个主设备能够驱动总线的控制权
同步	同步机制确保了多个设备之间的时钟同步。

I2C 是一个同步的串行通信接口。总线上的设备可以是主设备，从设备或者主从设备。当一个主从设备被寻址时，它的角色会切换为从设备。同一个时刻，总线上只允许一个主设备工作，该主设备负责驱动总线的同步时钟，并主导数据传输的开始与结束。首先介绍一些 I2C 协议中的术语，如表 15-14 所示。

15.4.2.1 时钟延展

当总线上的数据率超出一个从设备的处理能力时，该从设备可以选择将 SCL 拉低。此时无论主设备如何驱动，都会检测到 SCL 上的逻辑值为 0，则暂停当前的数据传输，这被称为时钟延展。这是由从设备驱动 SCL 的唯一情况。当从设备完成数据处理后，会释放 SCL。主设备在 SCL 上驱动上升沿，接管总线，恢复数据传输。

15.4.2.2 总线冲突检测与仲裁

当总线上多个主设备在同一时刻发起数据传输时，冲突检测和仲裁机制能够保证数据不被丢失。当主设备侦测到 SDA 上的逻辑值与自身驱动的值不符时，则检测到总线冲突，并放弃对总线的控制。举例说明，主设备 1 向 SDA 驱动 1，同时主设备 2 同时驱动 0，此时它们检测到 SDA 上实际的逻辑值为 0，那么主设备 1 检测到总线冲突，放弃总线控制；主设备 2 未检测到冲突，继续控制总线。

更多 I2C 协议的知识请参考官方的 I2C 协议标准 http://www.nxp.com/documents/user_manual/UM10204.pdf

15.4.3 I2C 协议介绍

PSoC 4 的 SCB 能够对 I2C 的主从设备模式做最大化的支持，具体如表 15-15。

表 15-15 I2C 设备的操作模式

模式	描述
从设备	仅作为从设备（缺省设置）
主设备	仅作为主设备
主从设备	支持同时作为主设备和从设备

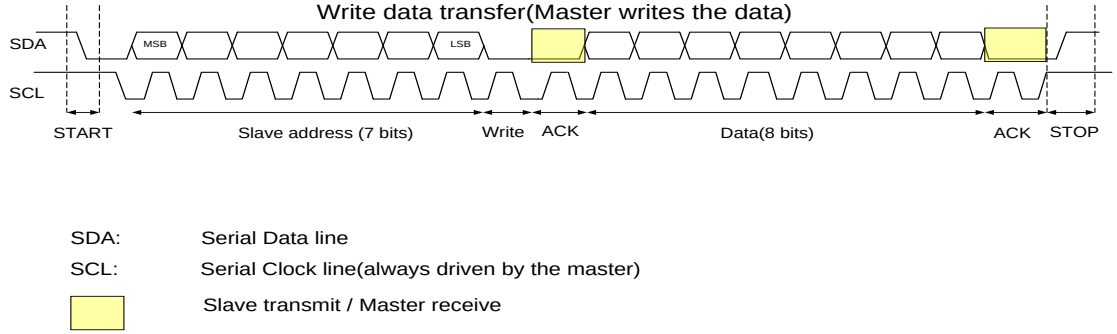
当工作在多主设备模式时，一个主设备（主设备 A）在发起数据传输之前必须检测当前总线是否在另一个主设备的控制下。如果总线被某一个主设备控制，那么必须等待该主设备结束当前传输（输出 STOP），主设备 A 才可以开始发起数据传输（输出 START）。

当工作在主从设备模式时，如果失去对总线的控制，则该设备转换为一个从设备的角色。

下面介绍 I2C 协议中两个数据传输：写数据和读数据。

15.4.3.1 写数据

图 15-18 写数据传输

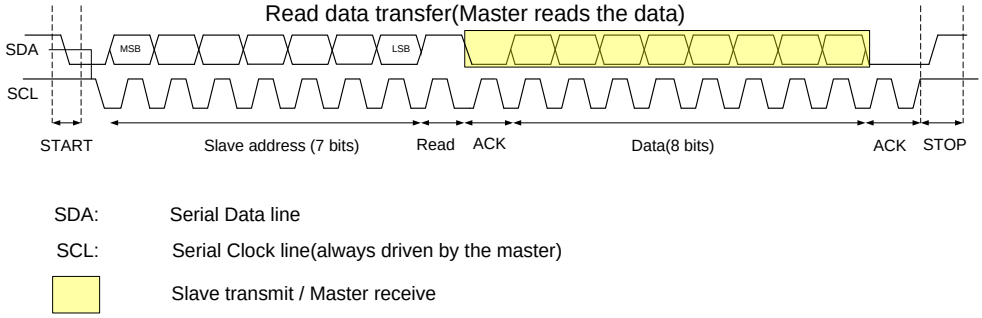


写数据传输可以简述如下：

- 主设备发起数据传输，输出 START 事件；
- 主设备输出 7 比特的 I2C 从设备地址和写标志‘0’；
- 主设备释放 SDA，被寻址的从设备拉低 SDA（发送 ACK），以表示对主设备寻址的响应；
- 如果 SDA 没有被拉低（所有的从设备发送 NACK），则说明当前总线上没有从设备的地址与该地址匹配，此时主设备将直接输出 STOP 事件结束本次数据传输；
- 如果寻址被响应，主设备将通过 SDA 向从设备写数据，从设备接收一帧数据后发送应答信号（ACK 或者 NACK）；
- 如果主设备接收到 ACK，则开始发送下一帧；如主设备接收到 NACK，则重发上一帧；
- 当主设备写完所有的数据，输出 STOP 事件，结束本次传输。

15.4.3.2 读数据

图 15-19 读数据传输



读数据传输可以简述如下：

- 主设备发起数据传输，输出 START 事件；
- 主设备输出 7 比特的 I2C 从设备地址和读标志‘1’；
- 主设备释放 SDA，被寻址的从设备拉低 SDA（发送 ACK），以表示对主设备寻址的响应；
- 如果 SDA 没有被拉低（所有的从设备发送 NACK），则说明当前总线上没有从设备的地址与该地址匹配，此时主设备将直接输出 STOP 事件结束本次数据传输；
- 如果寻址被响应，从设备将通过 SDA 向主设备发送数据，主设备接收一帧数据后发送应答信号（ACK 或者 NACK）；
- 如果从设备接收到 ACK，则开始发送下一帧；如从设备接收到 NACK，则重发上一帧；
- 当主设备读完所有的数据，输出 STOP 事件，结束本次传输。

15.4.4 EZI2C (Easy I2C) — I2C 的 EZ 模式

EZI2C（Easy I2C）协议允许从设备与主设备进行数据通信而无需 CPU 的介入，协议共提供了两种传输方式：写数据和读数据。在协议中，数据帧的长度只能为 8 比特。

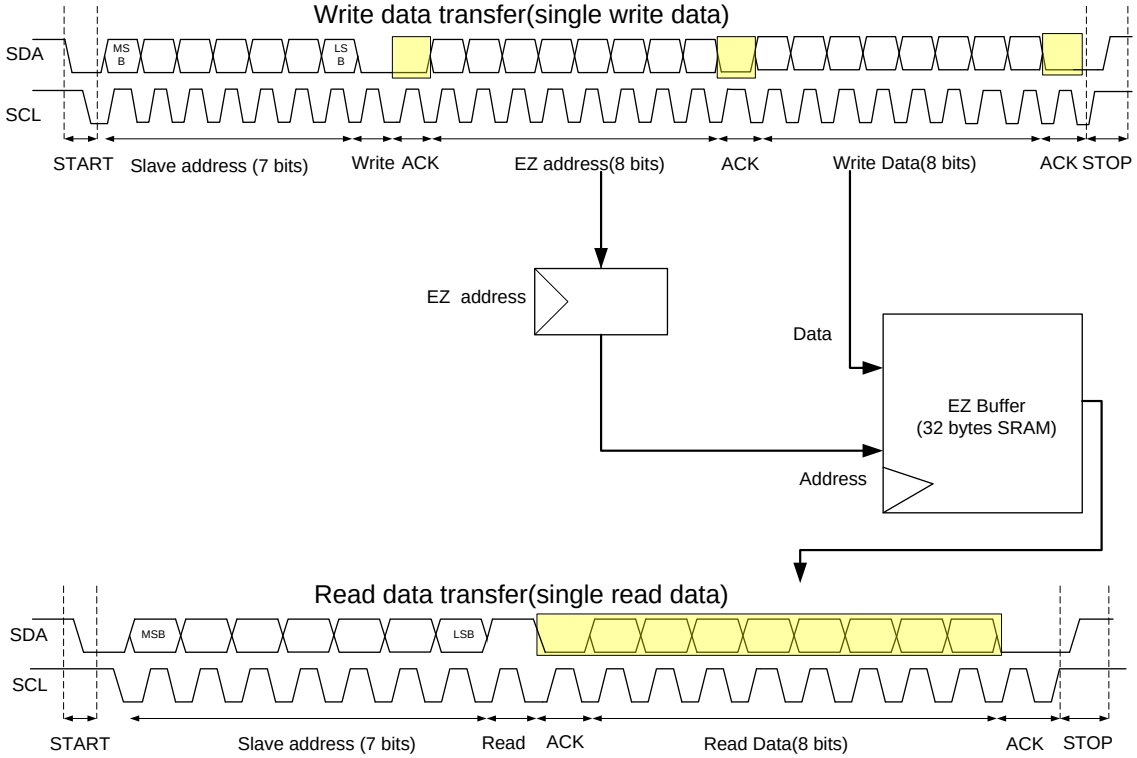
和在 EZSPI 中的使用情况相同，在 EZI2C 中，SCB 中的 SRAM 存储器作为 32 x 8bit 的 EZ 存储器使用。从设备中 EZ 存储器的地址存放在寄存器 EZ_ADDR（I2C_STATUS[15:8]）中，主设备可以对该地址指向的 EZ 存储器的某个字节空间进行读写操作。由于 EZ 存储器只有 32 个字节空间，所以 EZ 地址寄存器 EZ_ADDR 中只有低 5 比特是有效的。

如图 15-20 给出了 EZI2C 协议中读写操作的时序图。

在写操作中，首先主设备发起 START 事件并输出从设备地址和写标志；被从设备响应后，主设备会发送 EZ 地址（仅低 5 比特有效），更新从设备中的 EZ 地址寄存器 EZ_ADDR；被响应后，主设备对 EZ 地址寄存器指向的字节进行写操作。当完成一次写操作后， EZ_ADDR 自动累加 1，如果累加结果超出 31，则返回 0。

在读操作中，首先主设备发起 START 事件并输出从设备地址和读标志。与写操作不同，读操作中，主设备跳过了传输 EZ 地址的帧，后续读操作的对象所在的字节空间位置取决于上一次写数据操作结束时 EZ_ADDR 中的值。当完成一次读操作后，EZ_ADDR 累加 1，如果累加结果超出 31，则返回 0。

图 15-20 EZI2C 中的读写操作



SDA: Serial Data line

SCL: Serial Clock line(always driven by the master)

Slave transmit / Master receive

15.4.5 I2C 相关寄存器

I2C 协议相关的寄存器如表 15-16 所列，更多寄存器的信息请参考 PSoC 4 的寄存器用户手册

表 15-16 I2C 相关寄存器

寄存器名称	描述
SCB_CTRL	使能或者禁止 SCB； 选择 SCB 工作的协议（SPI，UART 或者 I2C）； 选择内部时钟模式或外部时钟模式； 选择 EZ 模式或者非 EZ 模式；
SCB_I2C_CTRL	选择主设备或者从设备模式； 配置从设备基于 FIFO 的何种状态发送 ACK 或者 NACK；
SCB_I2C_STATUS	总线状态标志位； 主设备或从设备的读写标志位； EZ 存储器地址寄存器；
SCB_I2C_M_CMD	主设备发送 START，STOP，ACK 和 NACK 的命令寄存器
SCB_I2C_S_CMD	从设备发送 ACK 和 NACK 的命令寄存器
SCB_STATUS	在 EZ 模式中，指示 32 x 8bit 的 EZ 存储器是否正在被外部时钟逻辑访问。
SCB_TX_CTRL	使能或者禁止 I2C 的发送； 定义数据传输帧的长度（I2C 中数据帧长度只能为 8bit）； 配置 MSB 在前或者 LSB 在前；
SCB_TX_FIFO_CTRL	配置 TXFIFO 的事件触发门限，该触发事件可以作为中断源；

寄存器名称	描述
	清除 TXFIFO； 冻结 TXFIFO；
SCB_TX_FIFO_STATUS	TXFIFO 中的数据个数； 指示当前 TX 移位寄存器中数据的有效性； TXFIFO 的读指针与写指针寄存器；
SCB_TX_FIFO_WR	保存前一次压入发送 FIFO 队列的数据；
SCB_RX_CTRL	使能或者禁止 I2C 的接收； 定义数据传输帧的长度（I2C 中数据帧长度只能为 8bit）； 配置 MSB 在前或者 LSB 在前；
SCB_RX_FIFO_CTRL	配置 RXFIFO 的事件触发门限，该触发事件可以作为中断源； 清除 RXFIFO； 冻结 RXFIFO；
SCB_RX_FIFO_STATUS	RXFIFO 中的数据个数； 指示当前 RX 移位寄存器中数据的有效性； RXFIFO 的读指针与写指针寄存器；
SCB_RX_FIFO_RD	保存前一次接收 FIFO 队列弹出的数据，数据弹出后将从 FIFO 队列中删除，相当于 POP 操作；
SCB_RX_FIFO_RD_SILENT	保存前一次接收 FIFO 队列弹出的数据，数据弹出后不从 FIFO 队列中删除，相当于 PEEK 操作；
SCB_RX_MATCH	从设备地址寄存器和地址屏蔽寄存器
SCB_EZ_DATA	EZ 存储器

15.4.6 I2C 的中断

与 I2C 相关的中断包括：

- 仲裁失败；
- 从地址匹配；
- 检测到开始/停止条件；
- 检测到总线错误；
- 数据传输完成；
- 外部时钟模式的中断（详见 SCB_INTR_I2C_EC）；
- 主设备模式的中断（详见 SCB_INTR_M）；
- 从设备模式的中断（详见 SCB_INTR_M）；
- TXFIFO 的中断（详见 SCB_INTR_TX）；
- RXFIFO 的中断（详见 SCB_INTR_RX）。

上述中断指向同一个中断向量，一旦其中一个未被屏蔽的中断被触发，则该中断向量被读取，CPU 转入中断服务程序处理该中断，查询中断寄存器进一步确认具体的中断触发事件。

15.4.7 I2C 的初始化

在 SCB 中，实现 I2C 的初始化步骤如下：

1. 配置寄存器 SCB_I2C_CTRL，选择主设备模式或从设备模式；
2. 通过寄存器 SCB_TX_CTRL 和 SCB_RX_CTRL，配置通用发送器与接收器，具体包括：
   - 数据帧的长度，对 I2C，数据帧的长度始终为 8；
   - 高有效位（MSB）在前或者低有效位（LSB）在前；
   - 使能发送器或者接收器；
3. 通过寄存器 SCB_TX_FIFO_CTRL 和 SCB_RX_FIFO_CTRL（如表 15-12），配置发送和接收的 FIFO 队列，具体包括：
   - 选择触发门限；
   - 清除 FIFO 队列和移位寄存器；
   - 暂时冻结发送器和接收器；
4. 通过寄存器 SCB_CTRL，选择工作协议（SPI、UART 或者 I2C），使能 SCB 模块。

如果需要对 SCB 进行配置，必须首先禁止 SCB，在配置完成后，重新使能 SCB，所有的配置将在使能后生效。注意重新使能 SCB 会带来重新的初始化，相关的状态也会丢失，如 FIFO 队列的数据。在配置中，使能 SCB 模块必须放在最后一步。

15.4.8 I2C 的时钟模式

- **EC_AM_MODE**：该位为 0 时，I2C 的地址匹配工作在内部时钟模式；该位为 1 时，I2C 的地址匹配工作在外部时钟模式。
- **EC_OP_MODE**：该位为 0 时，I2C 的其它协议操作工作（包括所有的数据传输）在内部时钟模式；该位为 1 时，I2C 的其它协议操作工作在外部时钟模式。

15.4.8.1 非 EZI2C 的时钟模式

在非 EZ 模式的 I2C 协议中，EC_OP_MODE 必须设置为 0；而 EC_AM_MODE 可以设置为 0 或者 1。其它的设置是不被支持或者无效的。如表 15-17 所示。

表 15-17 非 EZ 模式 I2C 协议的时钟模式设置

非 EZ 模式的 I2C 协议				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
电源模式	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
活动和睡眠	从设备地址匹配和其它协议操作均工作在内部时钟模式（适用于主设备和从设备）	从设备地址匹配工作在外部时钟模式，其它协议操作工作在内部时钟模式，唤醒中断被屏蔽（仅适用于从设备）	不支持	不支持
深度睡眠	不支持	从设备地址匹配工作在外部时钟模式，一旦从设备地址匹配成功，产生中断唤醒 CPU（仅适用于从设备）	不支持	
休眠	SCB 在这两种电源模式下无法工作（请参考电源模式章节）			
停止				

EC_AM_MODE = 0，EC_OP_MODE = 0：该配置仅在活动和睡眠电源模式中有效。此时，SCB 实现的 I2C 控制器完全工作在内部时钟模式，能够提供所有的功能。

EC_AM_MODE = 1，EC_OP_MODE = 0：该设置在活动、睡眠和深度睡眠的电源模式中均有效。当外部时钟逻辑检测到从设备地址匹配时，SCB 的唤醒中断请求位 WAKE_UP（SCB_INTR_I2C_EC[0]）将被置位，它可以用于唤醒 CPU。

- 在活动和睡眠电源模式中，从设备地址匹配工作在外部时钟模式；其它协议操作工作在内部时钟模式。在活动模式中，唤醒中断请求被屏蔽（SCB_INTR_SPI_EC_MASK[0] = 0）；在睡眠模式中，唤醒中断是否被屏蔽，用户可以自己设置。当从设备地址匹配成功后，SCB 需要将 SDA 的操作由外部时钟逻辑切换到内部时钟逻辑。这时有两种处理方式：向主设备发送 NACK（外部时钟逻辑不做任何操作），内部时钟逻辑向主设备发送 ACK（必须在主设备的采样沿前发生），并接管后面的数据传输；做时钟延展，由内部时钟逻辑结束时钟延展后，向主设备发送 ACK，并接管数据传输。通过配置 S_NOT_READY_ADDR_NACK（SCB_I2C_CTRL[14]），可以选择具体的处理方式。
- 在深度睡眠电源模式中，唤醒中断请求被使能（SCB_INTR_I2C_EC_MASK[0] = 1）。当从设备地址匹配成功时，SCB 会触发中断唤醒 CPU。唤醒过程需要一定的时间，所以当前数据帧的传输不被从设备响应。这时有两种处理方式：向主设备发送 NACK，等待设备被唤醒后，由内部时钟逻辑接管数据传输（此时如果主设备重新发起数据传输，并且地址匹配成功，则向主设备发送 ACK）；做时钟延展，等待被唤醒后，结束时钟延展，由内部时钟逻辑向主设备发送 ACK，并接管数据传输。通过配置 S_NOT_READY_ADDR_NACK（SCB_I2C_CTRL[14]），可以选择具体的处理方式。

15.4.8.2 EZI2C 的时钟模式

在 EZ 模式的 I2C 协议中，当 EC_OP_MODE 为 0 时，EC_AM_MODE 可以是 0 或者 1；当 EC_OP_MODE 为 1 时，EC_AM_MODE 必须为 1。如表 15-18 所示，表中的灰色单元格表示该配置（EC_AM_MODE = 1，EC_OP_MODE = 0）是有效的，但是不推荐使用，因为这涉及到从外部时钟逻辑向内部时钟逻辑切换的问题。

表 15-18 EZ 模式 I2C 协议的时钟模式设置

I2C, EZ mode				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
电源模式	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
活动和睡眠	从设备地址匹配和其它协议操作均工作在内部时钟模式 （仅适用于从设备）	从设备地址匹配同时工作在外部时钟模式，其它协议操作工作在内部时钟模式，唤醒中断被屏蔽 （仅适用于从设备）	从设备地址匹配和其它协议操作均工作在外部时钟模式 （仅适用于从设备）	不支持
深度睡眠	不支持	从设备地址匹配工作在外部时钟模式，一旦从设备地址匹配成功，产生中断唤醒 CPU （仅适用于从设备）	从设备地址匹配和其它协议操作均工作在外部时钟模式 （仅适用于从设备）	
休眠	SCB 在这两种电源模式下无法工作（请参考电源模式章节）			
停止				

EC_AM_MODE = 0，EC_OP_MODE = 0：该配置仅在活动和睡眠电源模式中有效。此时，SCB 实现的 I2C 控制器完全工作在内部时钟模式，能够提供所有的功能。

EC_AM_MODE = 1，EC_OP_MODE = 0：EZI2C 的该时钟模式与非 EZI2C 中对应的时钟模式相同。

EC_AM_MODE = 1，EC_OP_MODE = 1：该设置在活动、睡眠和深度睡眠的电源模式中均有效。

- 在活动和睡眠电源模式中，从设备地址匹配和其它协议操作均工作在外部时钟模式。由于此时内部时钟也存在，如果外部时钟逻辑和内部时钟逻辑同时访问 EZ 存储器，就会引起冲突。通过将寄存器位 BLOCK（SCB_CTRL[17]）置位，可以阻止内部时钟逻辑对 EZ 存储器的访问，避免冲突。如果未被阻止，并且与外部时钟逻辑的访问发生了冲突，那么内部时钟逻辑的访问会发生异常（读操作返回 0xFFFFFFFF，写操作被忽略），并且触发一个访问冲突中断请求，寄存器位 BLOCKED（SCB_INTR_TX[7]和 SCB_INTR_RX[7]），该中断请求时可以被屏蔽。
- 在深度睡眠模式中，从设备地址匹配和其它协议操作均工作在外部时钟模式。从设备中的 EZ 存储器完全由外部主设备操作。

15.4.9 唤醒操作

作为从设备时，当 I2C 接口检测到从地址匹配，可以触发中断。该中断能够将系统从睡眠/深度睡眠模式中唤醒。唤醒需要一定的事件，此时 I2C 可以有两种操作：

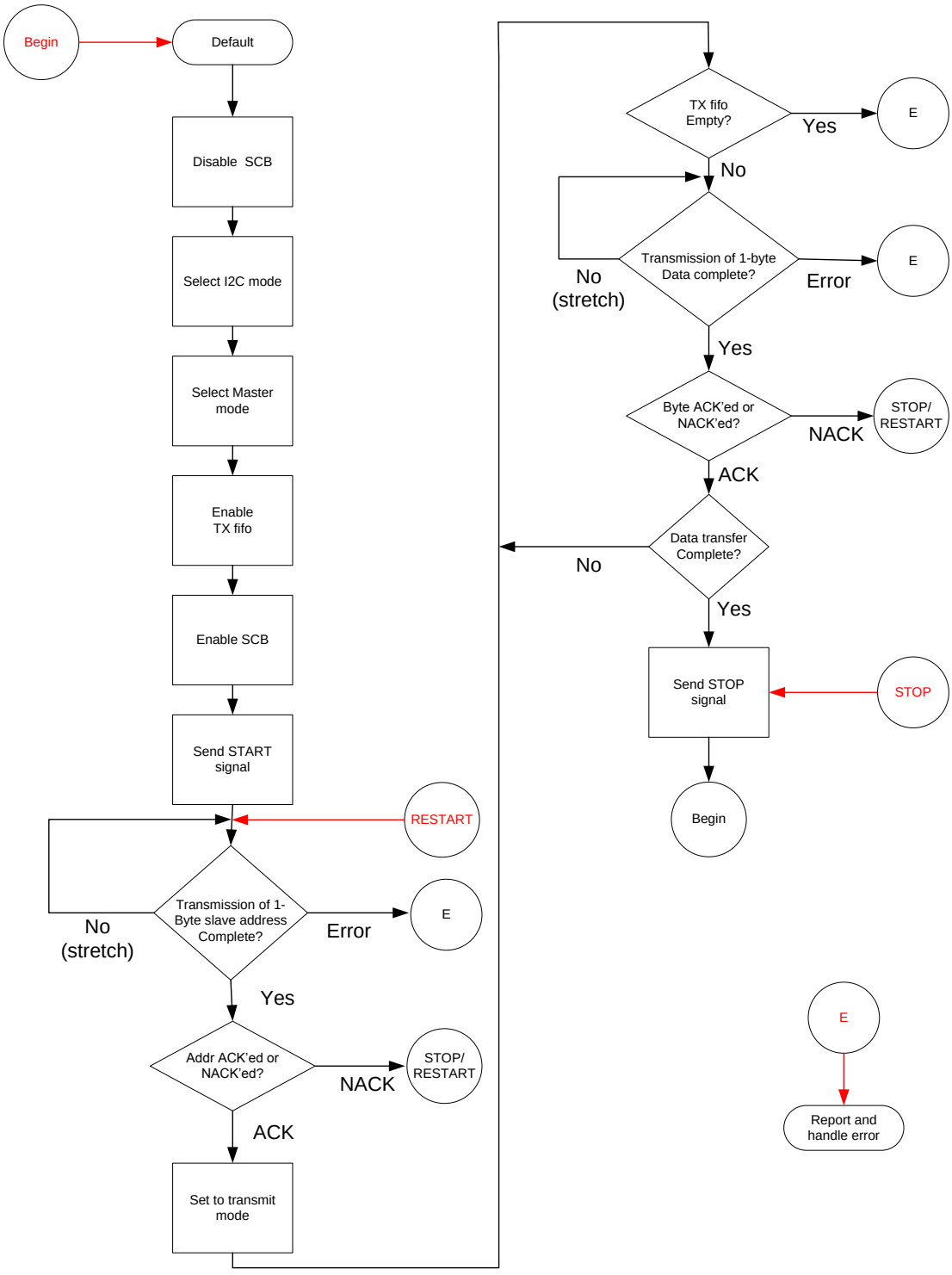
- 发送 ACK：检测到地址匹配后，执行时钟延展操作，等待系统被唤醒后，发送 ACK；
- 发送 NACK：检测到地址匹配后，立即发送 NACK，主机在系统被唤醒后，重新发送从地址；

注意：必须将中断比特位 SCB_WAKE_UP （SCB_INTR_I2C_EC[0]）使能，否则从地址匹配无法唤醒系统。

15.4.10 主设备模式数据传输示例

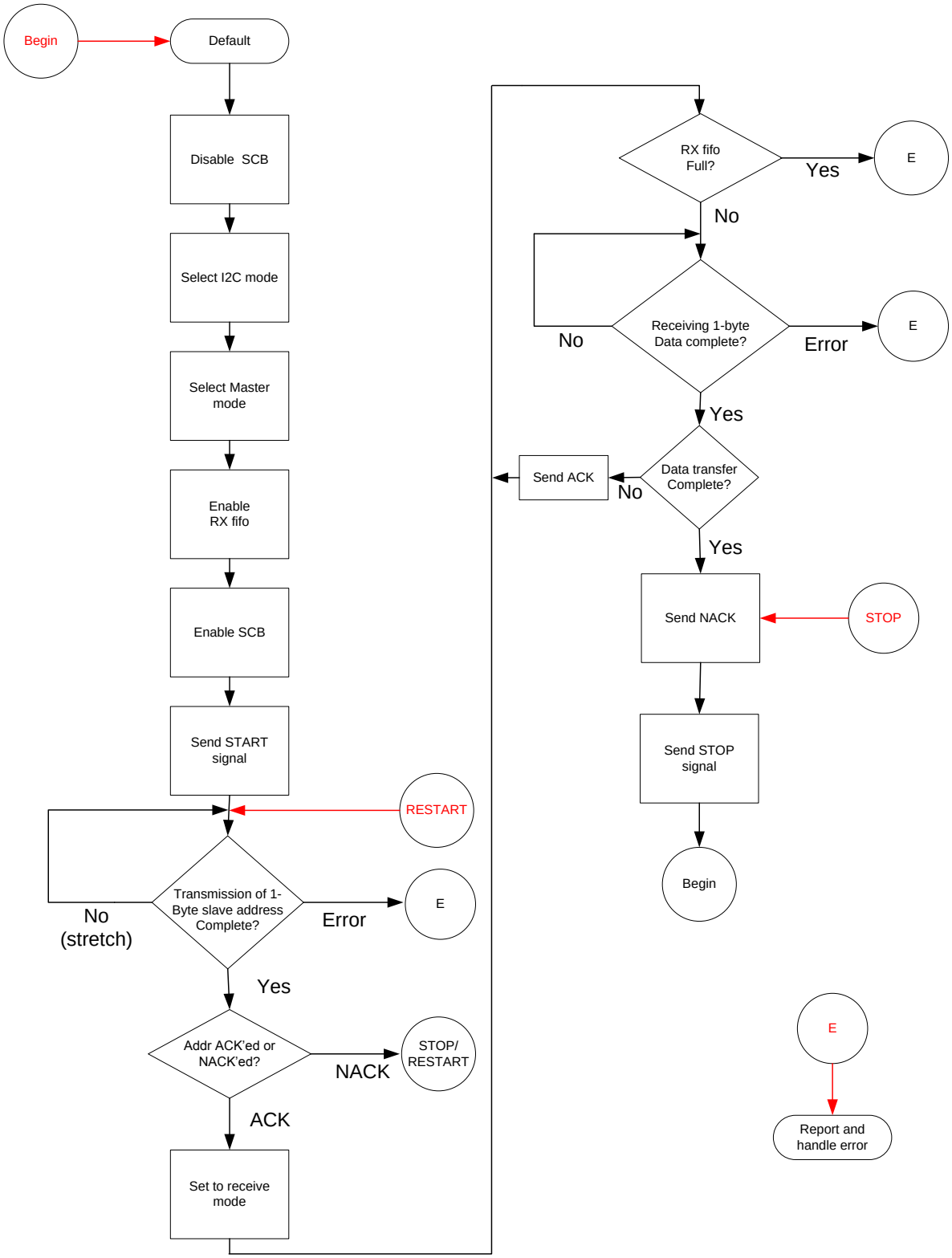
15.4.10.1 主设备发送数据

图 15-21 主设备发送数据的流程图



15.4.10.2 主设备接收数据

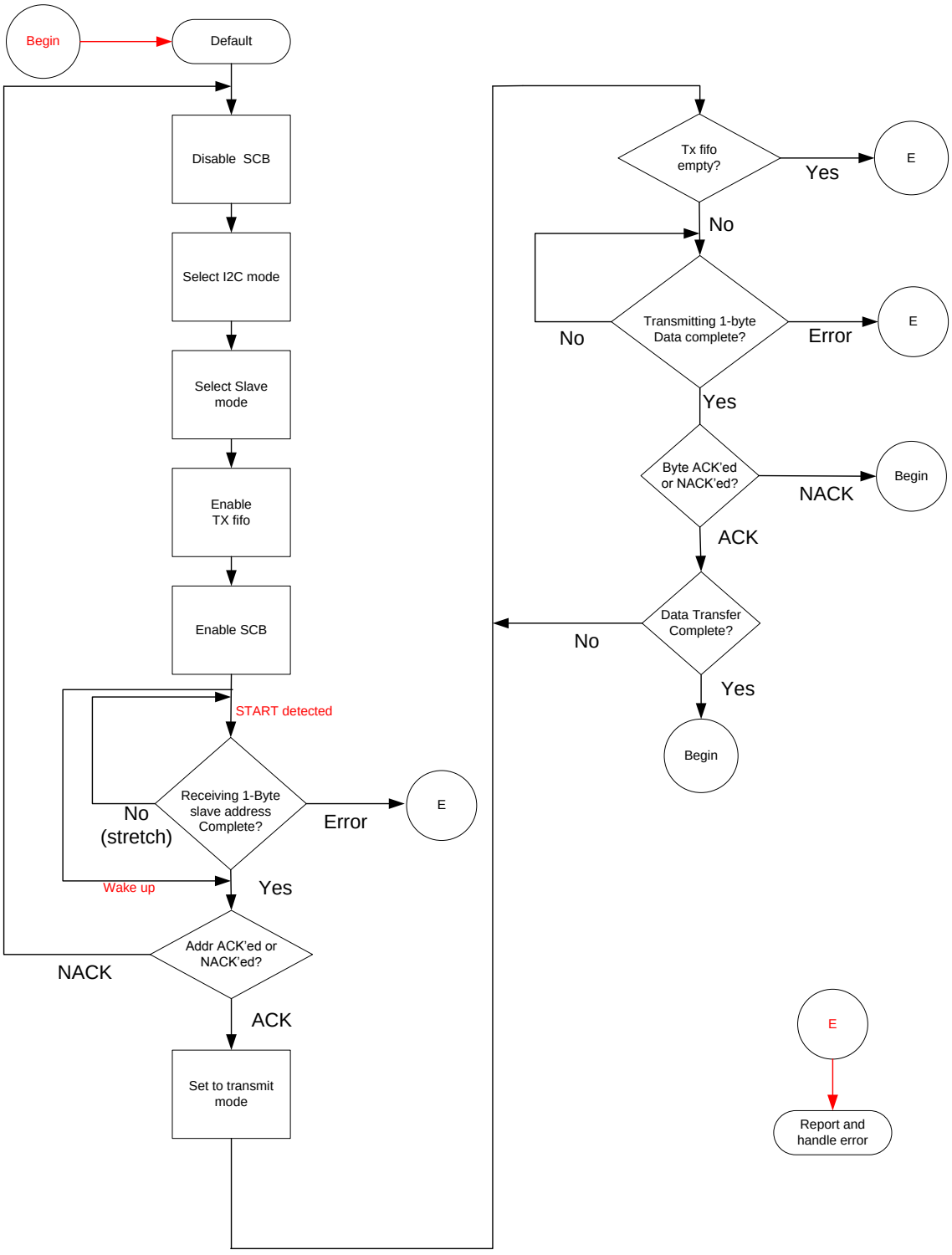
图 15-22 主设备接收数据的流程图



15.4.11 从设备模式数据传输示例

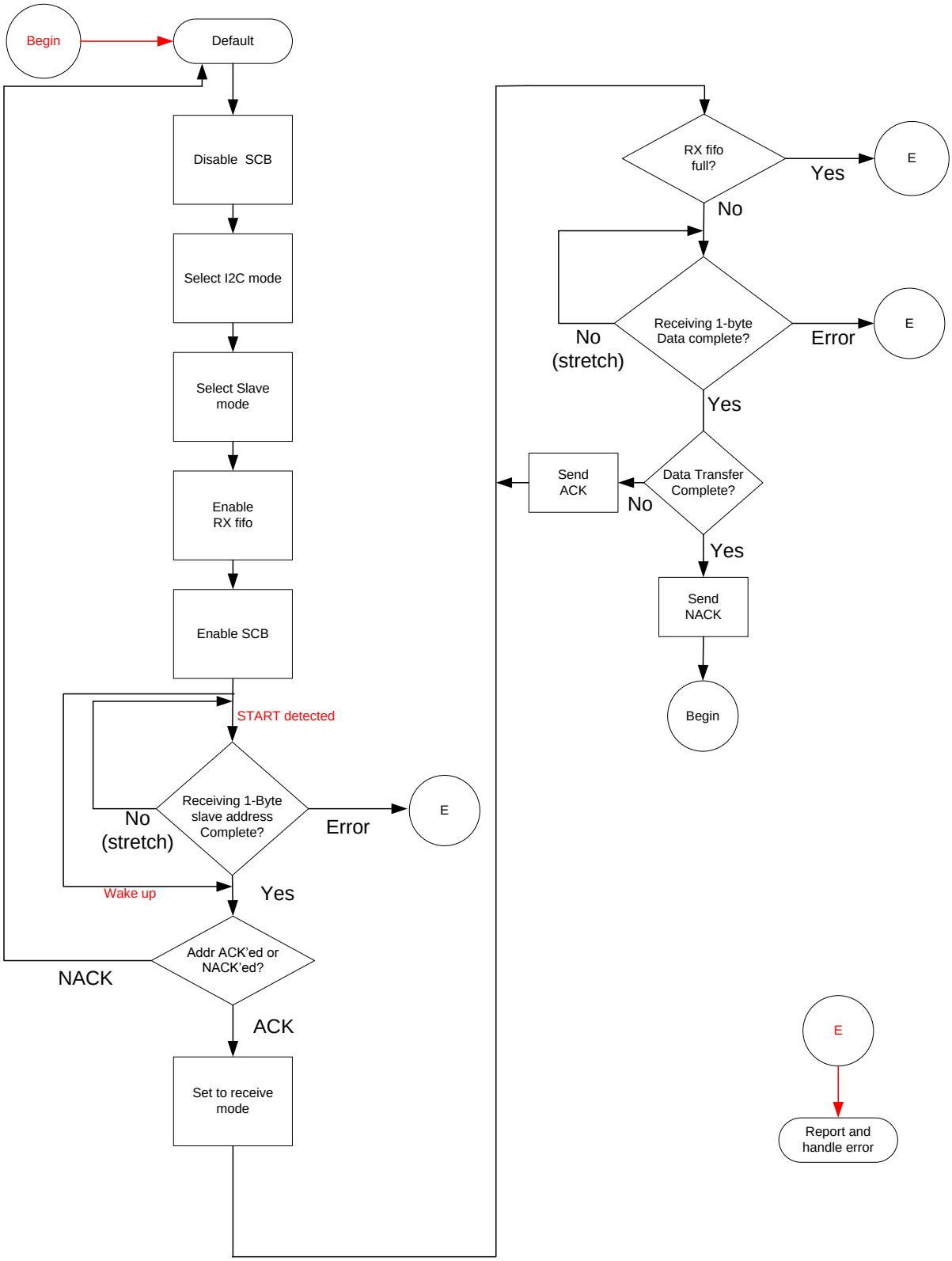
15.4.11.1 从设备发送数据

图 15-23 从设备发送数据的流程图



15.4.11.2 从设备接收数据

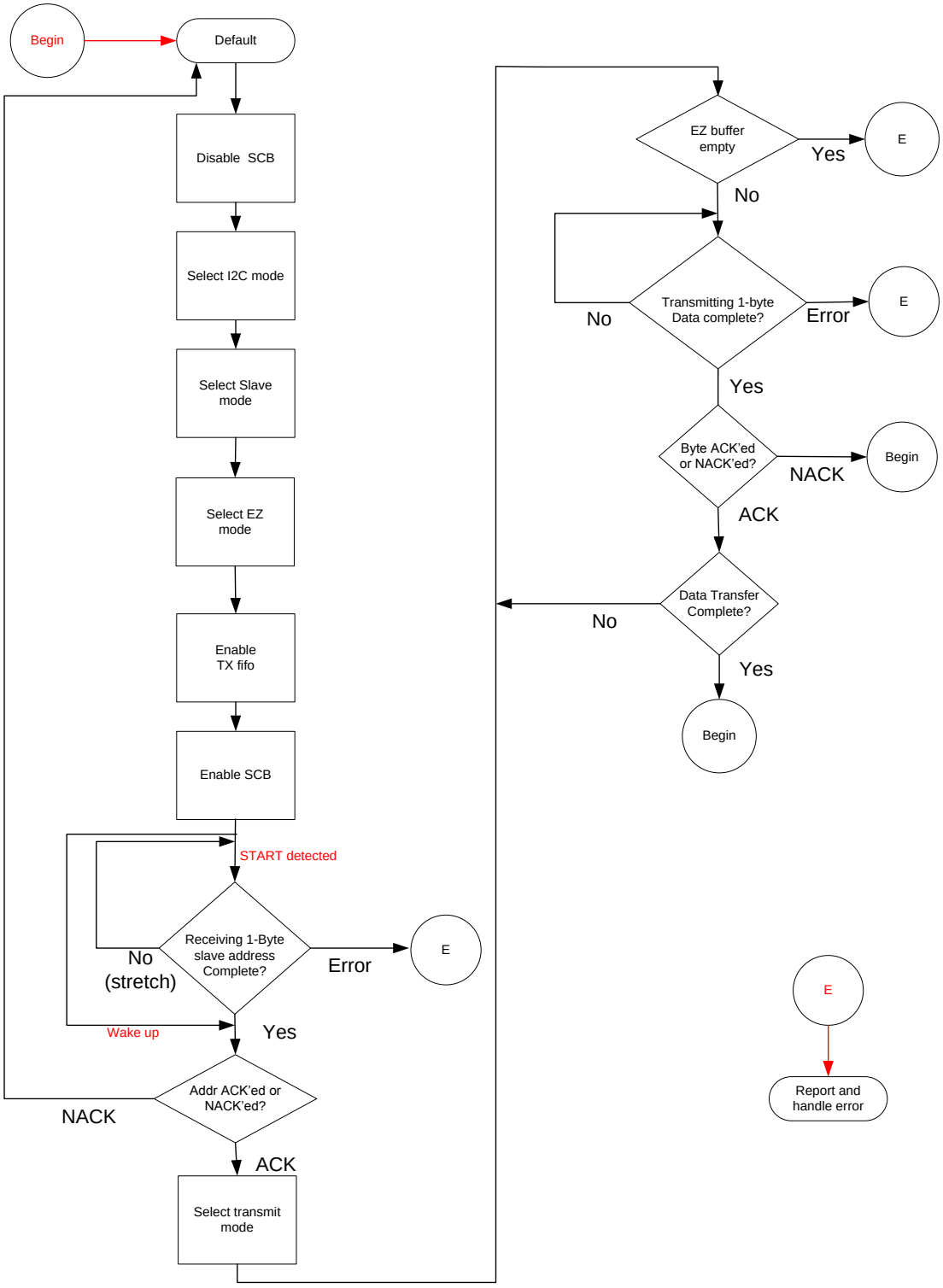
图 15-24 从设备接收数据的流程图



15.4.12 EZ 模式的数据传输示例

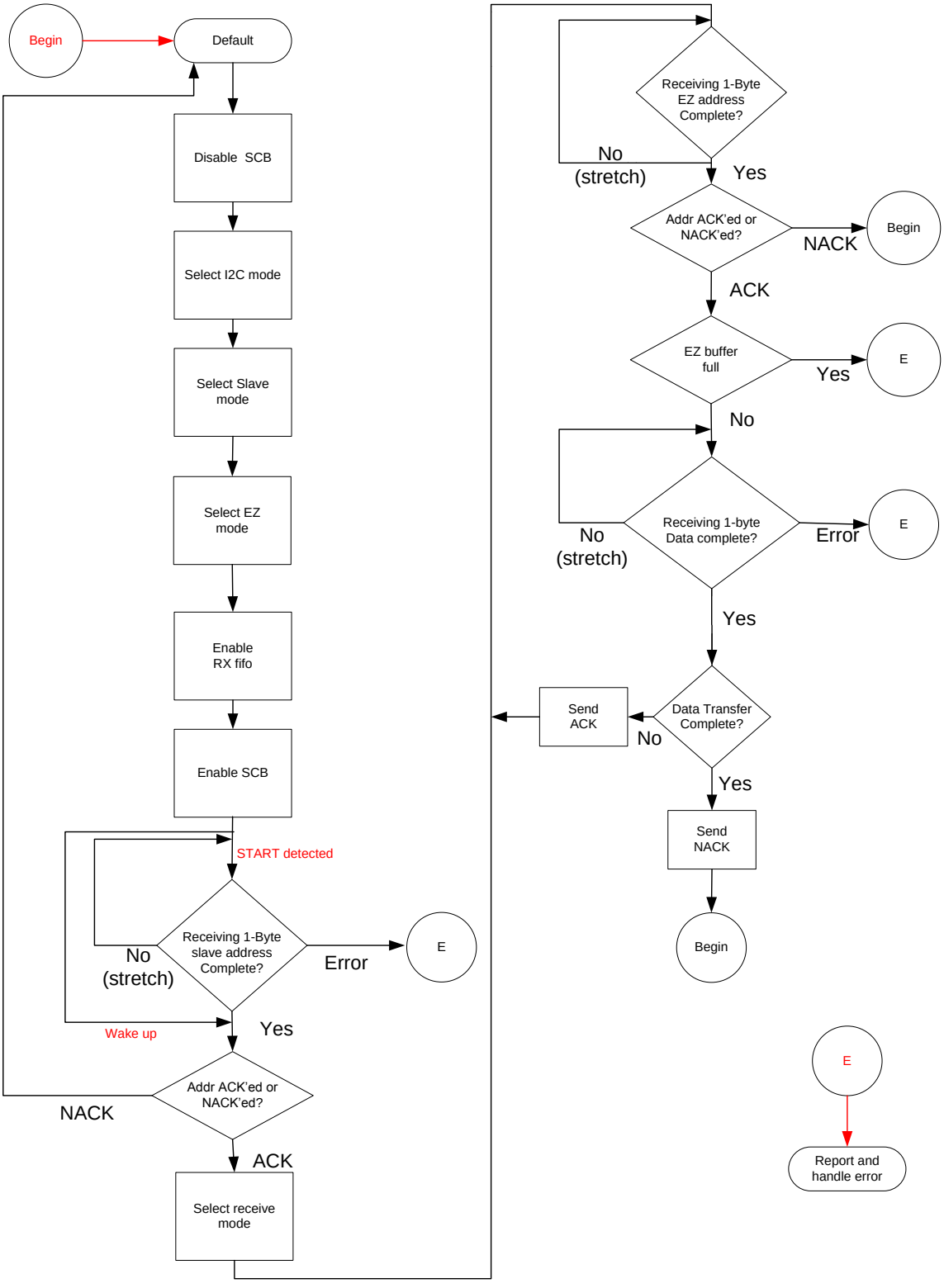
15.4.12.1 EZ 从设备发送数据

图 15-25 EZ 模式从设备发送数据的流程图



15.4.12.2 EZ 从设备接收数据

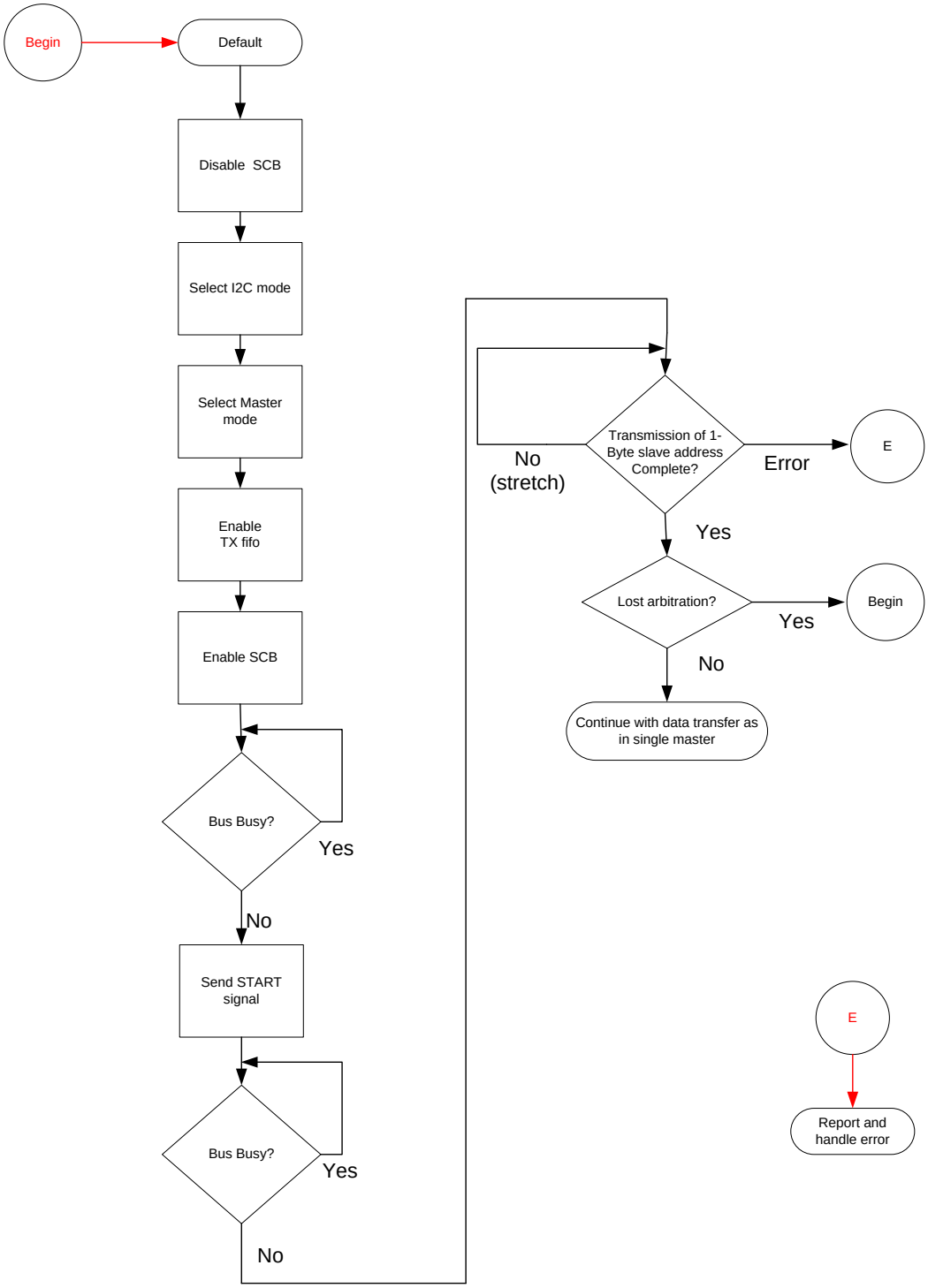
图 15-26 EZ 模式从设备接收数据的流程图



15.4.13 多主设备总线上的数据传输示例

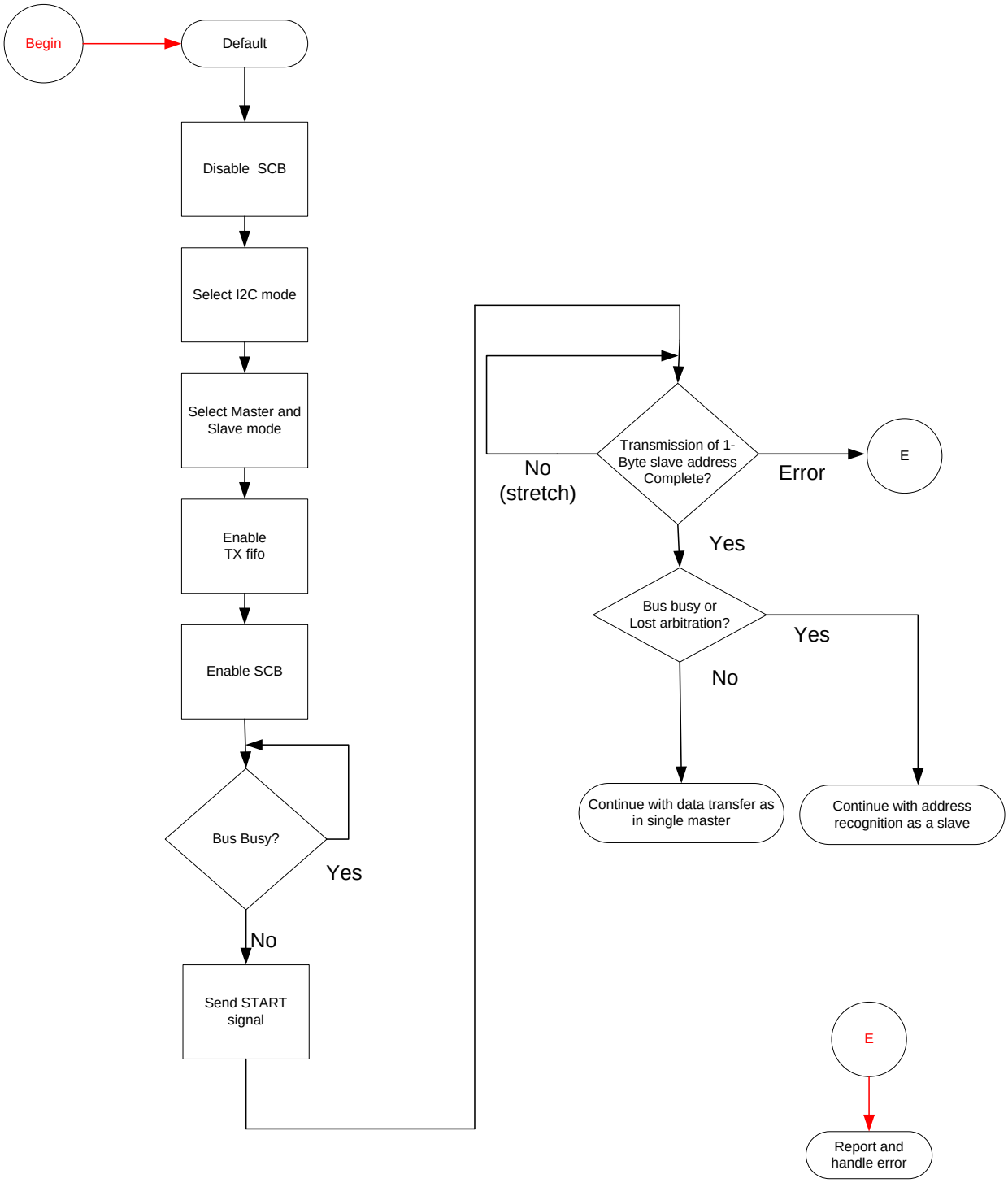
15.4.13.1 仅作为主设备

图 15-27 仅作为主设备的数据传输流程图



15.4.13.2 主从设备

图 15-28 主从设备的数据传输流程图



16通用数字模块

本章介绍 PSoC 4 中通用数字模块（UDB）的设计。UDB 实现了配置灵活性和高效性的平衡。UDB 包含了通用可编程逻辑器件（PLD：Programmable Logic Device），数据通道处理器（DP: DataPath），状态与控制模块（SC: Status and Control），复位与时钟模块（RC: Reset and Clock）及它们之间的互连信号通道。

16.1 特征

b. 为达到最佳的灵活性，每个UDB包含如下几个组件：
   - ❒ 一个基于算术逻辑单元的数据通道处理器（DP），包括指令存储器，寄存器和FIFO队列；
   - ❒ 2个PLD，每个PLD有12个输入，8个乘积项和4个宏单元输出；
   - ❒ 状态与控制（SC）模块；
   - ❒ 复位与时钟（RC）模块；

c. 为多个UDB组成的阵列提供了高度灵活的互连方案；

d. 同一时刻，UDB中的不同模块可用于实现不同的逻辑功能模块；

e. 多个UDB可以被级联，来设计更复杂的逻辑功能模块；

f. UDB能够实现的逻辑功能包括定时器，计数器，脉冲宽度调试器（PWM，带死区），UART收发器，I2C收发器，SPI收发器，循环冗余校验（CRC：Cyclic Redundancy Check）和伪随机序列（PRS：Pseudo Random Sequence）发生器；

g. CPU可以通过寄存器与UDB通信。

图 16-1 给出了 UDB 的内部框图，其中包括 2 个 PLD，1 个 DP，时钟与复位模块，状态与控制模块以及它们之间的互连。

图 16-2 给出了由 4 个 UDB 组成的一个 UDB 阵列及其接口。

图 16-1. UDB 框图

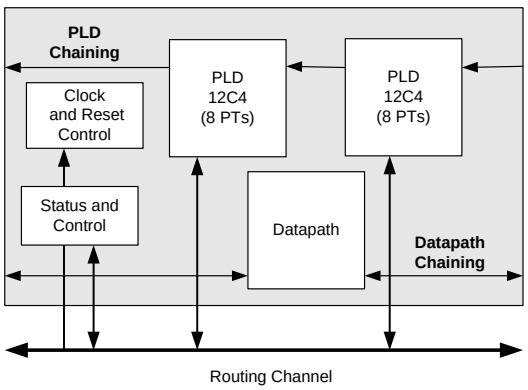
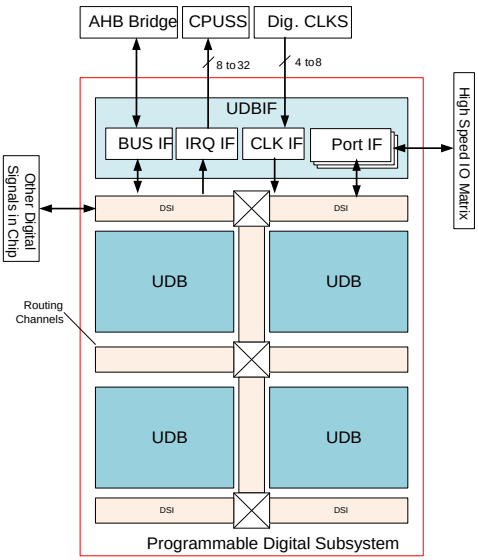


图 16-2. UDB 阵列



16.2 工作原理

The major components of a UDB are:

一个 UDB 主要由以下几个部分组成：

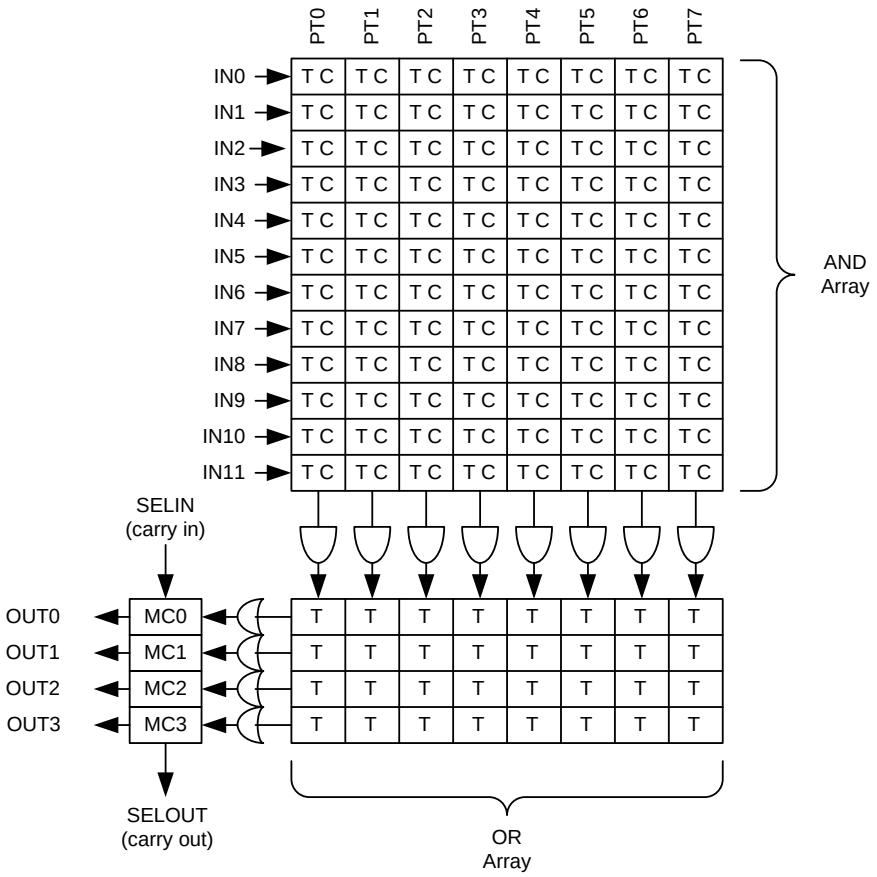
- **PLD(2 个)** — PLD 可以通过逻辑运算实现状态机，以及对 DP 操作，条件判决输入和驱动输出的控制；
- **DP** — 这个模块包含一个动态可编程的算术逻辑单元，4 个寄存器，2 个 FIFO 队列，比较器和条件判决器；
- **SC 模块** — 这个模块为 CPU 的软件与 UDB 之间的通信提供接口；
- **RC 模块** — 这个模块为 UDB 中其它模块提供时钟源的选择与使能，以及复位源的选择；
- **级联信号** — 通过级联信号，相邻的 UDB 可以级联，实现更加复杂的逻辑功能；
- **互连通道** — 通过一个可编程的开关阵列，UDB 的内部模块被连接到互连通道；
- **系统总线接口** — UDB 中所有的寄存器与随机存取存储器（RAM）都被映射到系统的地址空间，CPU 可以通过 8-bit 和 16-bit 的位宽访问它们。

16.2.1 PLD

每个 UDB 包含 2 个“12C4”结构的 PLD 模块，“12C4”结构主要由与门阵列（AND Array）、或门阵列（OR Array）和宏单元（MC：MacroCell）组成，如图 16-3。PLD 可以用来实现状态机，执行输入输出数据的条件判决，以及创建查找表（LUTs: Look-Up Tables）。通过配置，PLD 也可以执行算术运算，或者控制 DP 按照一定的配置字序列工作并产生不同的状态。PLD 可以用通用的 RTL 代码来进行设计。

1 个 PLD 中，12 个数字信号被输入到与门阵列分别与 8 个乘积项进行与运算，在输入之前，数字信号可以被选择是否先进行非运算。与门阵列的输出被输入到或门阵列，“12C4”中的 C 表示对所有的输入，或项是一个常数，通过编程，或门的输入可以是任意一个或者所有的乘积项。“12C4”结构赋予了逻辑运算最大的灵活性，并且保证了所有的输入与输出都是可置换的。

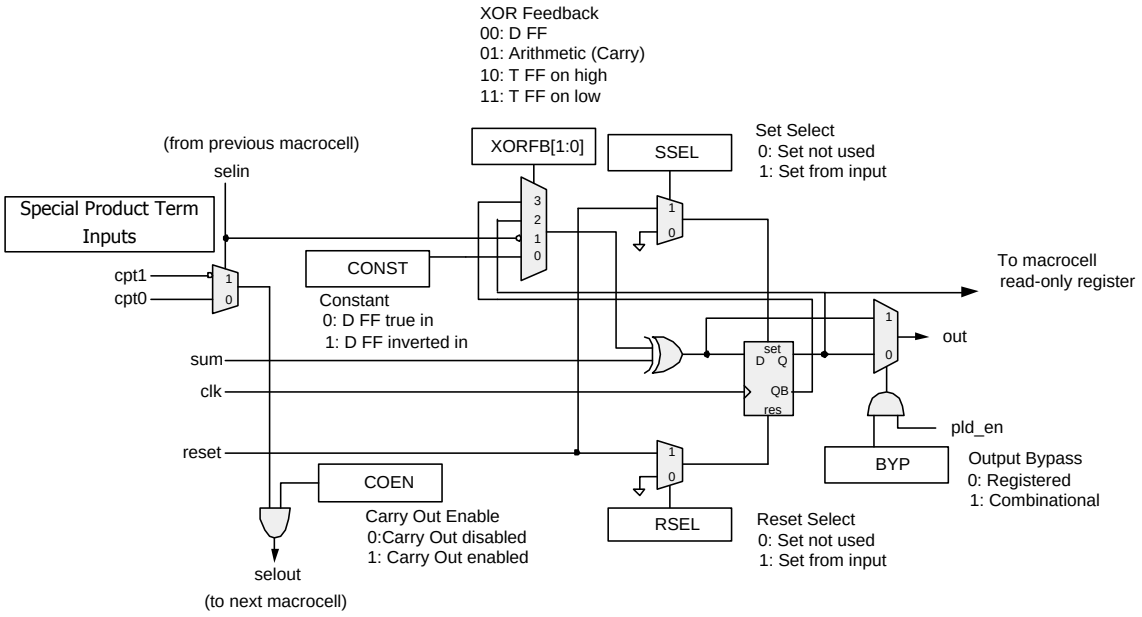
图 16-3. PLD 中的 12C4 结构



16.2.1.1 PLD 的宏单元

宏单元的架构如图 16-4 所示。在宏单元中，逻辑运算的结果作为输出信号驱动互连阵列，它有组合逻辑或者时序逻辑两种模式，其中的差别在于是否与 PLD 的时钟进行同步。在初始化中，或者有互连信号控制的异步操作中，输出寄存器可以被置位或者复位。每个宏单元有 7bit 的配置寄存器，它们分别位于 PLD RAM 的四个字节中。

图 16-4 PLD 中宏单元的架构

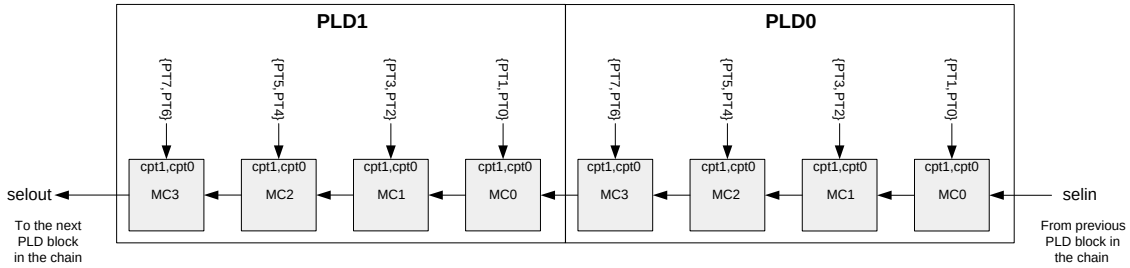


1 个 UDB 中的 8 个宏单元能够作为一个 8 比特的只读寄存器被 CPU/DMA 访问。多个 UDB 中的宏单元能够作为 16 比特或者 32 比特寄存器被 CPU/DMA 访问。请参见章节“UDB 中寄存器的寻址”。

16.2.1.2 PLD 的进位级联

在 UDB 中，2 个 PLD 被级联，如图 16-6 所示。进位链的输入“selin”从上一个 UDB 连接过来，通过 PLD 中的每个宏单元，最后作为进位链的输出“selout”连接到下一个 UDB。为了实现高效的算术功能，宏单元中会产生特殊的乘积项来与进位链进行协同工作。

图 16-6. PLD 进位级联示意图



16.2.1.3 PLD 的配置

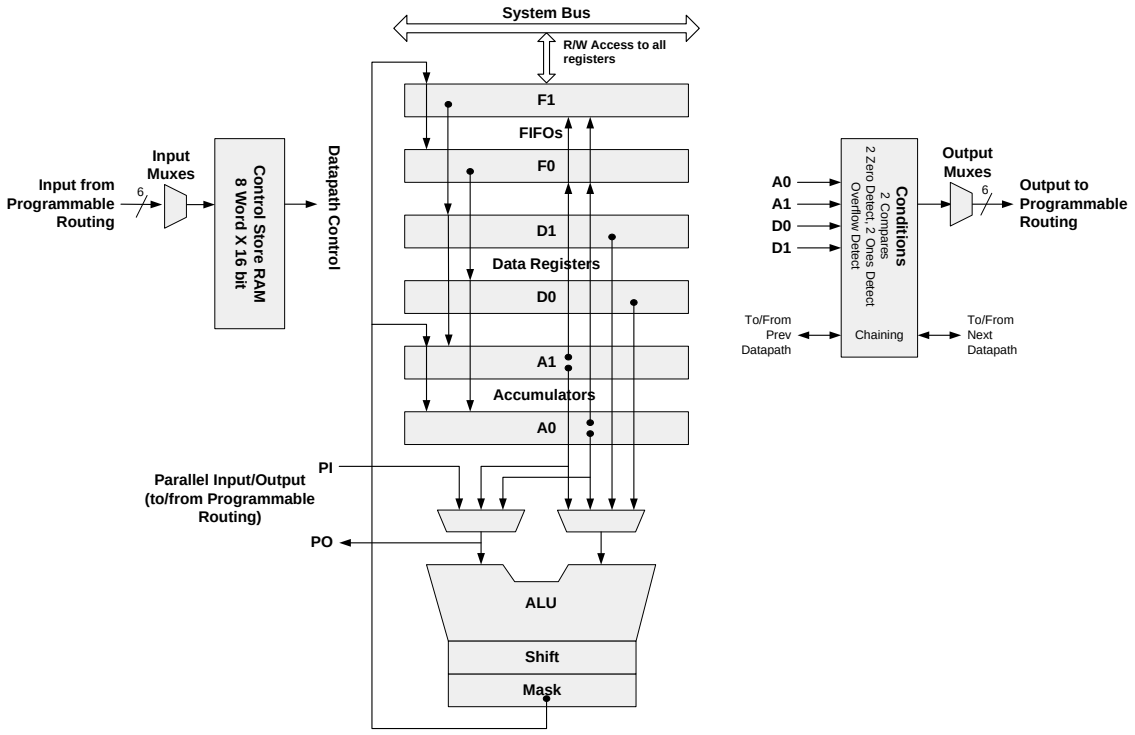
对于系统来说，PLD 表现为总线上一个位宽为 16bit 的 RAM。与门阵列对应于 12 x 8 x 2 bits，即 24 bytes；或门阵列对应于 4 x 8 bits，即 4 bytes；4 个宏单元对应于 4 x 8 bits，即 4 bytes。综上，1 个“12C4”结构的 PLD 对应的可配置寄存器的大小为 32 bytes。由于 1 个 UDB 包括 2 个 PLD，所以 PLD 对应的可配置寄存器的大小为 64 bytes，即 32 words。

16.2.2 数据通道处理器（DP：DataPath）

如图 16-7 所示，DP 包含一个 8bit 的单周期算术逻辑处理器（ALU：Arithmetic Logic Unit ）以及相应的移位寄存器（Shift）和屏蔽寄存器（Mask）。DP 还包含一个基于 8 x 16bit RAM 的动态控制寄存器（Control Store RAM），通过它可以动态控制 DP 的操作。相邻 UDB 中的 DP 可以被级联以实现更复杂的应用。

DP 能够实现的功能包括定时器，计数器，脉冲宽度调试器（PWM，带死区）和移位器等。DP 的加减功能使它可以支持数字Δ – Σ运算。

Figure 16-7. Datapath Top Level



16.2.2.1 概览

数据通道处理器的关键特性如下：

动态配置

动态配置指：在不同的周期里，寻址到动态配置寄存器中的不同配置字，设置 DP 的功能和内部信号互连。动态配置寄存器中，最多可写入 8 套不同的配置字，组成一个配置字序列。动态配置寄存器的地址线通过互连通道连接到 PLD，I/O 或者 DP。CPU 可以对动态配置配置寄存器进行读写。

算术逻辑单元

DP 中的算术逻辑单元（ALU）可以执行 8 种通用的运算：递增，递减，加，减，与，或，异或以及直通。通过动态配置寄存器，ALU 可以逐周期执行不同的运算。ALU 后面的移位寄存器和屏蔽寄存器可以对运算结果进行相应的操作。

条件判决

DP 中包含 2 个带位屏蔽功能的比较器，可以用来比较 DP 中的两个寄存器。通过配置，也可以对 DP 的寄存器进行条件判决，如全“0”，全“1”和溢出等条件判决，其结果可以通过互连通道输出到其它模块。

内置 CRC/PRS

DP 可以用于设计单周期循环冗余（CRC）校验器和伪随机序列（PRS）发生器。如果在设计中，CRC 或者 PRS 的位宽超过了 8bit，则需要级联相邻 UDB 的 DP。由于动态配置的特性，该运算可以与其它运算在同一个 DP 中交织进行的。

最高有效位可配置

DP 中寄存器的 MSB（最高有效位）可以通过配置指定，这使 DP 可以实现不同位宽的 CRC/PRS 运算。通过与 ALU 输出的屏蔽功能配合，DP 可以实现不同位宽的定时器，计数器和移位器。

输入/输出 FIFO 队列

DP 中包含 2 个深度为 4byte 的 FIFO 队列，可被独立配置为输入缓冲器（CPU 写 FIFO，DP 读 FIFO）或者输出缓冲器（DP 写 FIFO，CPU 读 FIFO）。FIFO 产生满/空状态信号，可以触发中断。通过设计，FIFO 的状态信号能够决定动态配置寄存器中不同的配置字被使能，使 DP 后续执行不同的操作和运算。

级联

通过配置，将相邻 UDB 中的 DP 的信号级联，能够实现更复杂的算术，移位和 CRC/PRS 运算。可以级联的信号包括条件判决信号，移位信号，进位信号，捕获信号等。

复用

某些设计中，运算速度要求不高，一个 ALU 可以被分时复用。通过动态配置寄存器，可以令 ALU 逐周期执行不同的运算，ALU 在一个周期的运算结果可以作为下一个周期运算的操作数。这样可以在一个 8bit 的 DP 中实现一个 16bit 的函数；也可以在 DP 中交叉进行数据移位操作与 CRC 运算。

数据通道处理器的输入

DP 有三种类型的输入信号：配置信号，控制信号和串/并行数据信号。配置信号选择控制存储器的地址；控制信号将 FIFO 中的数据载入数据寄存器，将累加器的数据载入 FIFO；串行数据输入包括移位输入和进位输入，由互连通道输入的并行数据位宽为 8bit。

数据通道处理器的输出

DP 一共生成 16 个信号，包括条件判断信号，状态信号以及其他的数据信号。通过一个 16-6MUX，其中的 6 个信号可以通过互连通道输出。默认的情况下，信号在输出之前会与 DP 的时钟进行同步；通过配置，也可以跳过这个同步。

数据通道处理器的工作寄存器

每个 DP 有 6 个 8bit 工作寄存器，如表 16-1。它们对 CPU 都是可读写的。

表 16-1 DP 的工作寄存器列表

类型	名称	描述
累加器	A0, A1	累加器的数据可以是 ALU 的运算操作数，也可以是结果。数据存储器和 FIFO 的数据可以被载入累加器。累加器中一般包含函数的当前运算值，如计数器的计数变量，CRC 中的多项式变量或者移位操作数变量。该寄存器无数据保持功能，进入睡眠会丢失原状态，复位后被复位到 0x00。
数据寄存器	D0, D1	数据寄存器一般包括函数中固定的操作数，如 PWM 发生器的比较值，定时器/计数器的周期或者 CRC 中的多项式系数。该寄存器有数据保持功能。
FIFO	F0, F1	FIFO 的深度为 4 个字节，它们可以用来缓存运算的操作数或者结果。2 个 FIFO 可以被独立的配置成输入缓存器或者输出缓存器。FIFO 可以产生信号，来表示缓存器处于满/空状态，这在很多应用中十分重要，如缓存 SPI/UART 的收发数据，缓存 PWM 的比较数，缓存定时器的周期等。这些寄存器没有数据保持功能。

16.2.2.2 数据通道处理器中的 FIFO

FIFO 的工作模式与配置

每个 FIFO 有多种配置与操作模式，如表 16-2 所示。

表 16-2 FIFO 的配置与操作模式

模式	描述
输入与输出	输入模式中，CPU 向 FIFO 写数据，DP 从 FIFO 读数据；输出模式中，DP 向 FIFO 写数据，CPU 从 FIFO 读数据；
单字节缓冲器	FIFO 可以工作为一个单字节缓冲器。这时写入 FIFO 的数据被透传到接收端，FIFO 中的数据随时可以被覆盖。
电平敏感与边沿敏感	装载 FIFO 的控制信号（F0_LD 和 F1_LD）可以是电平敏感的或者边沿敏感的。
正常与快速	DP 向 FIFO 写数据的速率分为正常与快速两种。正常模式下，数据传输时钟为 DP 的内部时钟；快速模式下，数据传输时钟为总线时钟。
软件捕获	当 FIFO 为输出状态时，可以工作在软件捕获的模式下。这时，CPU 读累加器的数据是由 FIFO 中转的，而非直接访问累加器。CPU 读累加器的动作会产生一个捕获触发信号，使累加器的数据被载入 FIFO，以便 CPU 读取。这个捕获触发信号可以被级联到相邻的 UDB 的 DP。
同步与异步	当 DP 的时钟与总线时钟同步时，FIFO 的状态信号可以直接输出或者与 DP 时钟做一次同步后输出； 当 DP 的时钟与总线时钟异步时，FIFO 的状态信号可以与 DP 时钟做一次同步或两次同步后输出；
时钟反向	通过配置，FIFO 的时钟可以与 DP 的时钟反向。

图 16-8 给出了两个 FIFO 可能的几种操作模式。TX/RX 模式中，F0 为输入状态，F1 为输出模式，典型的设计用例是 SPI 收发器。双捕获器模式中，两个 FIFO 均为输出状态。双缓冲器模式中，两个 FIFO 均为输入状态。

图 16-8 FIFO 的操作模式

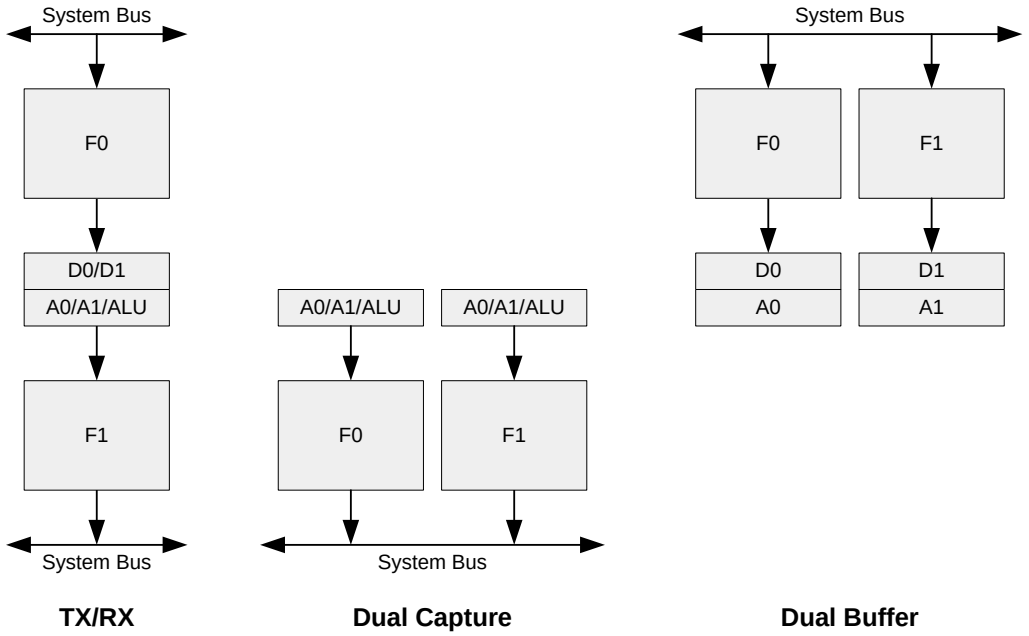


图 16-9 FIFO 的输入与输出

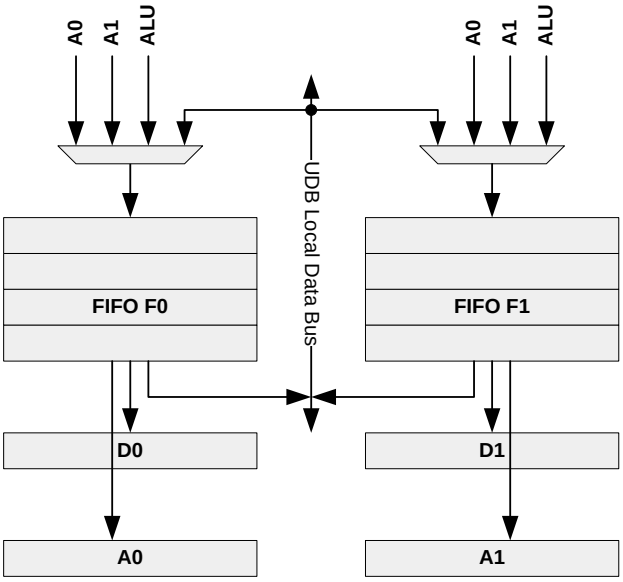


图 16-9 给出了 FIFO 的输入输出情况。当 FIFO 为输入模式时，其数据源是系统总线，其数据流包括累加器和数据寄存器；当 FIFO 为输出模式时，其数据源包括累加器和算术逻辑单元，其数据流为系统总线。通过寄存器 F0_INSEL[1:0]和 F1_INSEL[1:0]，可以配置 FIFO 的数据源或数据流，如表 16-3，其中 x = 0 或 1。

表 16-3 FIFO 的输入输出模式配置

Fx_INSEL[1:0]	描述
00	**FIFO 为输入状态，**数据源为系统总线，数据流为累加器或数据寄存器。
01	**FIFO 为输出状态，**数据源为累加器 A0，数据流为系统总线。
10	**FIFO 为输出状态，**数据源为累加器 A1，数据流为系统总线。
11	**FIFO 为输出状态，**数据源为算术逻辑单元，数据流为系统总线。

FIFO 的状态

FIFO 有两个状态信号：bus 和 block，它们可以通过 DP 的输出 MUX 传输到互连通道。状态信号“bus”通常可以用来触发一个 FIFO 的读/写中断；状态信号“block”主要是记录 FIFO 的状态，供 UDB 内部调用。这两个状态信号的具体含义由寄存器 Fx_INSEL[1:0]和 Fx_LVL 界定

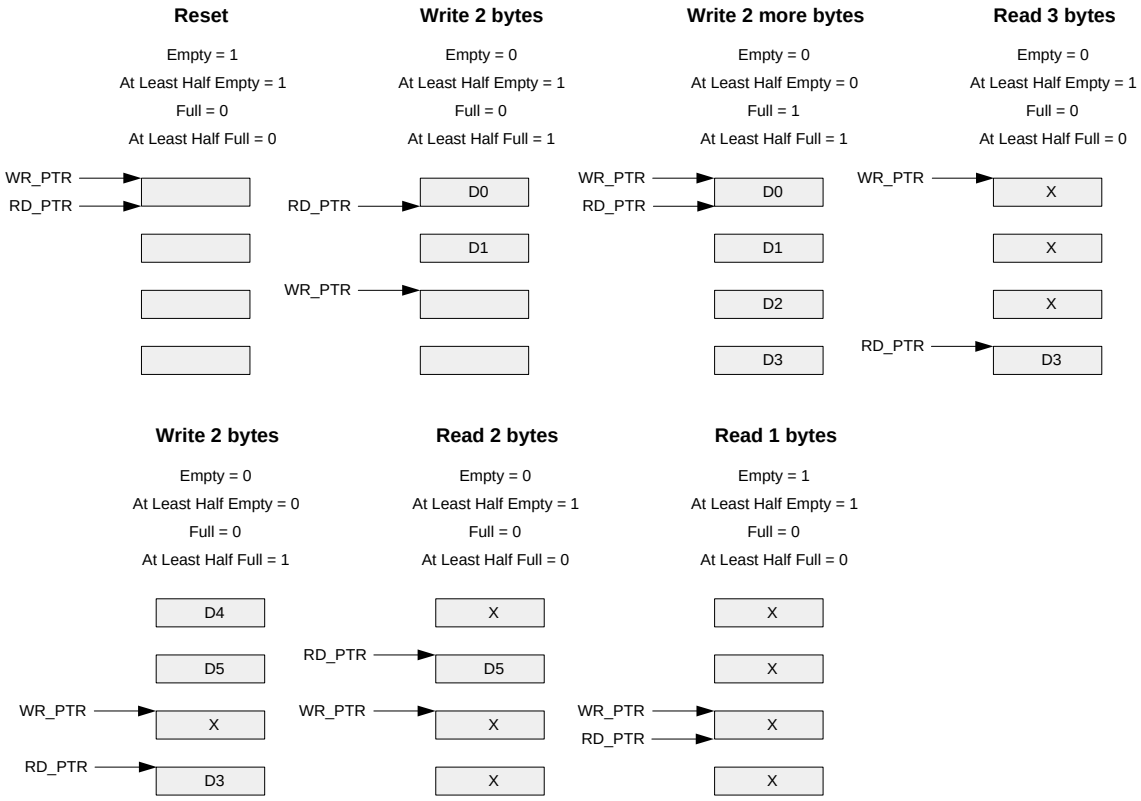
表 16-4 FIFO 的状态

Fx_INSEL[1:0]	Fx_LVL	状态	状态信号	描述
输入模式	0	非满	Bus Status	当 FIFO 至少还有 1 个字节为空时，输出为 1，此时数据总线可以向 FIFO 写数据。
输入模式	1	至少一半为空	Bus Status	当 FIFO 至少还有 2 个字节为空时，输出为 1。此时数据总线可以向 FIFO 写数据。
输入模式	NA	空	Block Status	当 FIFO 为空时，输出为 1。当 FIFO 不为空时，DP 可以从 FIFO 中读取数据。当 FIFO 为空时，DP 可以保持空闲状态，或者产生一个空载条件信号。
输出模式	0	非空	Bus Status	当 FIFO 至少还有 1 个字节数据时，输出为 1。此时数据总线从 FIFO 读数据。
输出模式	1	至少一半为满	Bus Status	当 FIFO 至少还有 2 个字节数据时，输出为 1。此时数据总线从 FIFO 读数据。
输出模式	NA	满	Block Status	当 FIFO 被写满数据时，输出为 1。当 FIFO 未满时，DP 内部可以继续向 FIFO 写入数据。当 FIFO 被写满时，DP 可以保持空闲状态，或者产生一个过载条件信号。

FIFO 的读写

图 16-10 描述了一个典型的 FIFO 读写过程，以及各状态信号的对应值。虽然从图中看来，读写是在不同时间发生的，但是它们确实是可以同时发生的。

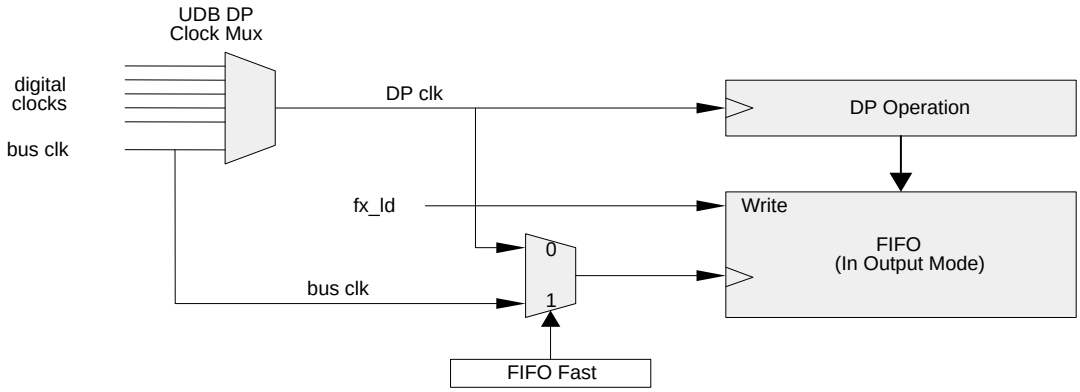
图 16-10 FIFO 的读写过程



FIFO 的快速模式

当 FIFO 工作在输出模式时，DP 的数据发送时钟和 FIFO 的数据接收采样时钟一般是 DP 自身的工作时钟。然而，如图 16-11 所示，通过设置，可以选择总线时钟（bus clk）作为 FIFO 的数据接收采样时钟。配合 FIFO 的边沿敏感写模式，这样做可以将从 DP 到 FIFO 的数据传输延迟从 DP 时钟周期的量级降低到总线时钟的量级，使 CPU 能够以最小的延迟来读 FIFO 的数据。当然，采用更高的时钟来工作会稍微增加系统的功耗。fx_ld 是外部输入的 FIFO 写触发信号，它通常由状态机或者条件判断器产生。如果 FIFO 工作在快速模式，fx_ld 必须满足总线时钟的时序要求。

图 16-11 FIFO 快速模式的配置



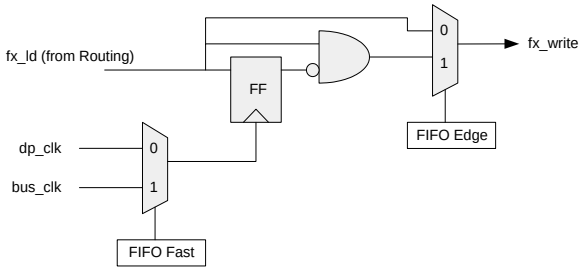
FIFO 的写模式

DP 向 FIFO 写数据有两种模式。在电平敏感写模式中，数据从累加器同步传输到 FIFO。fx_ld = 1 时，DP 持续向 FIFO 写数据。

在边沿敏感写模式中，fx_ld 有上升沿输入时，累加器中的数据被传输到 FIFO。在这种模式下，可以利用一个外部信号作为边沿写触发信号，捕获累加器中的值。

图 16-12 给出了两种模式的控制电路，通过配置寄存器位 FIFO_Edge，可以选择 FIFO 的写模式。

图 16-12. FIFO 写模式的配置



FIFO 的软件捕获模式

通过将寄存器位 FIFO_Cap 置位，使能 FIFO 的软件捕获模式。当 DP 在进行运算时，这种模式为 CPU 提供一种可靠读取累加器的方式。此时的 FIFO 必须工作在输出状态，CPU 以 FIFO 为中转，从累加器中读取数据。这种模式下，F0 从 A0 载入数据，F1 从 A1 载入数据。

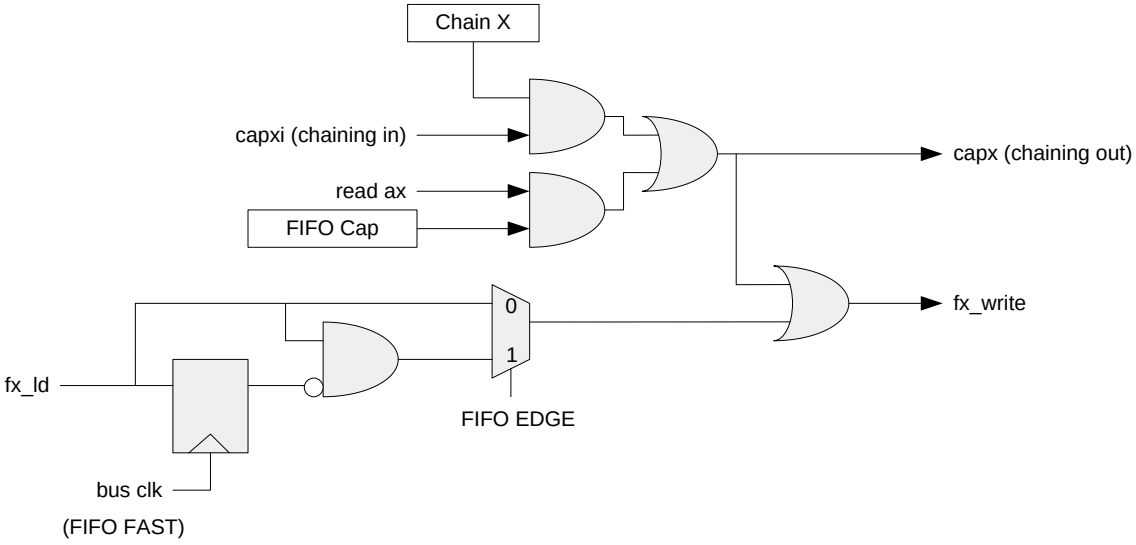
如图 16-13，读累加器的操作（read_ax）可以触发软件捕获信号（capx），使累加器的数据被传输到 FIFO，这时 CPU 可以立即从 FIFO 中读走这些数据。软件捕获信号可以被级联，这允许 CPU 从被级联的 UDB 的 FIFO 同时读取 16bit 或者更多的数据。

之前我们介绍的外部写触发信号 fx_ld 也可以触发 FIFO 的写操作。fx_ld 和 capx 逻辑或运算的结果共同决定了 FIFO 是否进行写操作。若这两个 FIFO 写触发信号同时有效，将出现不可预知的结果，因此通常情况下，这两个写触发信号应该是互斥的，但在以下情况下可以例外：

- FIFO 工作在快速写模式下；
- FIFO 工作在边沿敏感写模式下。

在使能软件捕获模式之前，可以通过 ACTL 寄存器设置将 FIFO 清零，使 FIFO 的读指针和写指针工作在一个可知的状态。

图 16-13 FIFO 软件捕获模式的配置



FIFO 的控制寄存器

在辅助控制寄存器 ACTL 中，共有四个寄存器位用于控制 FIFO 的操作，分别是 FIFO0_CLR，FIFO1_CLR，FIFO0_LVL 和 FIFO1_LVL。

FIFOx_CLR 用于复位 FIFOx。当它为 1 时，对应的 FIFO 将被复位。当它为 0 时，对应的 FIFO 能够被正常操作。如果 FIFOx_CLR 持续被置位，对应的 FIFO 表现为一个单字节的缓冲器，被写入 FIFO 的数据可以立即被读出，也可以在任意时候被覆盖。

配合 Fx_INSEL[1:0]的设置，FIFOx_LVL 可以选择对应的状态信号 fx_bus_stat 的意义，如表 16-4 所示。

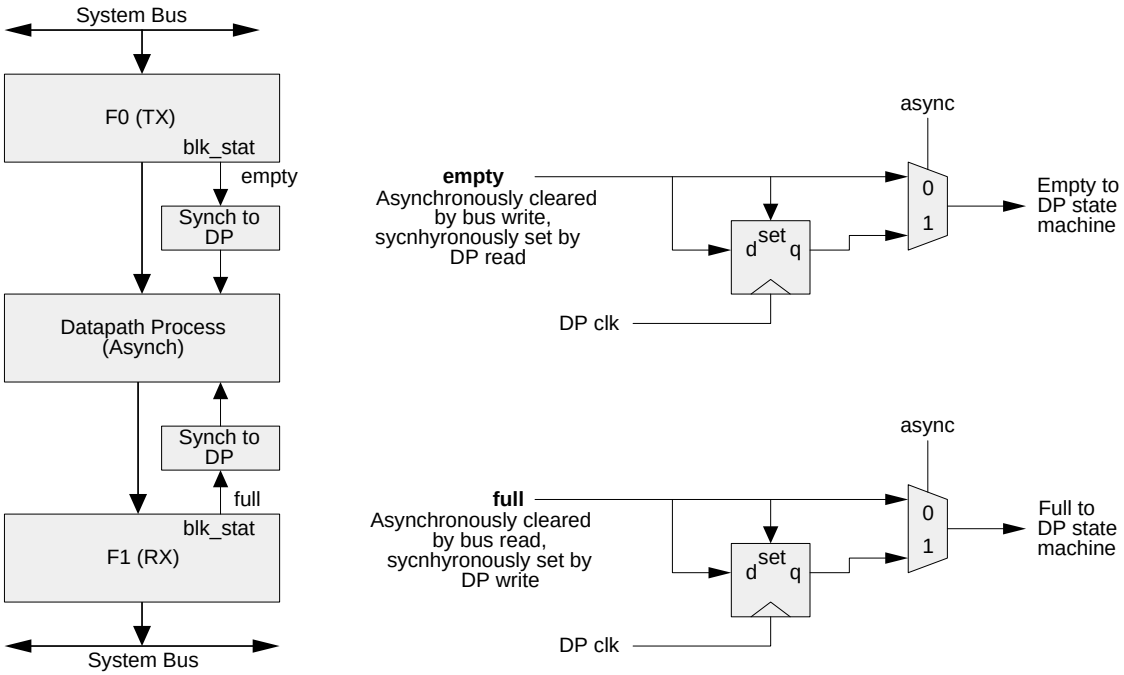
图 16-14 给出了 FIFO 中状态信号的同步电路，其中 F0 工作在输入状态，F1 工作在输出状态，分别作为 TX 和 RX 寄存器。

在 TX 端，DP 的状态机通过“empty”来判断 FIFO 中是否还有数据可读。“empty”由 DP 时钟同步置位，但是被总线时钟同步清零。当被清零时，“empty”需要被同步到 DP 时钟。

在 RX 端，DP 的状态机通过“full”来判断 FIFO 中是否还有数据可读。“full”由 DP 时钟同步置位，但是被总线时钟同步清零。当被清零时，“full”需要被同步到 DP 时钟。

当 DP 的时钟与总线时钟同步时，FIFO 的状态信号 blk_stat 可以直接输出或者与 DP 时钟做一次同步后输出；当 DP 的时钟与总线时钟异步时，FIFO 的状态信号可以与 DP 时钟做一次同步或两次同步后输出。这些选择是通过寄存器位 FIFO_ASYNC 和 FIFO_ADD_SYNC 进行配置的。

图 16-14 FIFO 状态信号的同步



FIFO 的溢出操作

由于 FIFO 不提供内置上溢保护或下溢保护，所以需要根据 FIFO 的状态信号来正确的进行读写操作。如果 FIFO 处于满状态，则下一次的写操作令 FIFO 发生上溢，写入的新数据将覆盖 FIFO 顶部的数据（下一个即将被读出的数据）。如果 FIFO 处于空状态，则下一次的读操作令 FIFO 发生下溢，读出的数据是不可预测的。虽然发生下溢或者上溢，但是 FIFO 的读写指针正常移动。

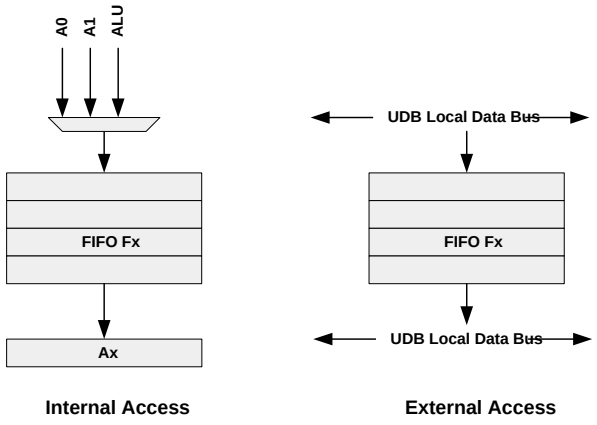
FIFO 的时钟反向

通过寄存器位 Fx_CK_INV 的配置，可以将对应的 FIFO 的时钟做反向。默认状态下，Fx_CK_INV = 0，FIFO 的时钟与 DP 的时钟通向；当 F0_CK_INV = 1 时，FIFO 的时钟与 DP 的时钟反向。这支持了双时钟沿的通信模块的设计，如 SPI。

FIFO 的动态控制

将寄存器位 Fx_DYN（CFG17[0]和 CFG17[1]）置位后，可以通过互连信号 dx_ld 动态配置对应 FIFO 的访问模式（0：内部访问模式；1：外部访问模式）。这时 dx_ld 不再是 FIFO 写数据寄存器的触发信号。

图 16-15 FIFO 读写的动态控制



如图 16-15，在内部访问模式中，只有 DP 能够读写 FIFO。配置寄存器位 Fx_INSEL 可以选择 FIFO 的数据源，在该模式中，Fx_INSEL = 0（选择系统总线为数据源）是非法的。另外注意，FIFO 的数据只能传输到累加器中，而不能够传输到数据寄存器中。在外部访问模式中，只有 CPU 能够读写 FIFO。

举例说明该应用。首先，FIFO 工作于外部访问模式（dx_ld = 1），CPU 可以向 FIFO 写一个或多个字节的数据；然后，FIFO 切换到内部访问模式（dx_ld = 0），DP 从 FIFO 中读取数据并做运算，并将结果写入 FIFO；最后，FIFO 重新切换到外部访问模式，CPU 从 FIFO 中读出运算结果。

16.2.2.3 FIFO 的状态信号

DP 中有 2 个 FIFO，每个 FIFO 有两个状态信号，分别是 fifo0_bus_stat，fifo0_blk_stat 和 fifo1_bus_stat，fifo1_blk_stat。它们的含义与寄存器位 Fx_INSEL[1:0]和 Fx_LVL 相关，具体参考表 16-4。

16.2.2.4 DP 中的 ALU

DP 中的算术逻辑单元可以完成 8bit 的运算，在算术逻辑单元之后还有一个 8bit 的移位寄存器和一个 8bit 的屏蔽单元。

算术与逻辑运算

通过位于寄存器位 Function[2:0]，可以配置算术逻辑单元进行何种运算。

表 16-7 ALU 的功能配置

Func[2:0]	算术运算	运算表达
000	直通	srca
001	递加	++srca
010	递减	--srca
011	加	srca +srcb
100	减	srca – srcb
101	异或	srca ^ srcb
110	与	srca & srcb
111	或	srca | srcb

进位输入

进位输入可以参与某些算术运算，它在各运算中的缺省值如表 16-8 所示。

表 16-8 进位输入在各运算中的应用

算术运算	运算表达式	运算式解析和进位默认值
递加	++srca	srca + 00h + ci， ci 的缺省值为 1
递减	--srca	srca + ffh + ci， ci 的缺省值为 0
加	srca + srcb	srca + srcb + ci， ci 的缺省值为 0
减	srca – srcb	srca + ~srcb + ci， ci 的缺省值为 1

通过寄存器位 CI_SELA（CFG13[1:0]）或 CI_SELB（CFG13[3:2]）可以配置进位信号在不同的操作模式工作，具体如表 16-9。通过动态配置寄存器中的 CI_SEL（CDFG[2]）位，可以选择 CI_SELA 或 CI_SELB 的配置有效。

表 16-9 进位操作的模式

CI SEL A CI SEL B	进位操作模式	描述
00	Default	默认算术模式：进位为缺省值，如表 16-8 所示。
01	Registered	寄存器缓存模式：上一周期运算中的进位结果存入进位寄存器，作为下一周期的进位输入。这种模式下，一个 8bit 的 ALU 工作两个周期可以模拟一个 16bit 的 ALU 工作一个周期的情况。
10	Routed	互连模式：进位输入来自于互连通道传输过来的信号。
11	Chained	级联模式：进位输入来自于级联情况下高一级的 DP。

在互连模式中，进位信号有效和无效时 ALU 的运算结果如表 16-10 所示。

表 16-10 互连进位输入的操作

算术运算	进位输入信号极性	进位输入有效	进位输入无效
递加	正	++srca	srca
递减	负	--srca	srca
加	正	(srca + srcb) + 1	srca + srcb
减	负	(srca – srcb) – 1	(srca – srcb)

进位输出

进位输出可以被选择为 DP 的输出信号，进位输出的值不仅与运算中的操作数有关，也与设置的 MSB 位置有关。通过级联，可以将进位输出的值输出，以扩展运算规模。表 16-11 给出了在不同运算情况下，进位输出是否有效。

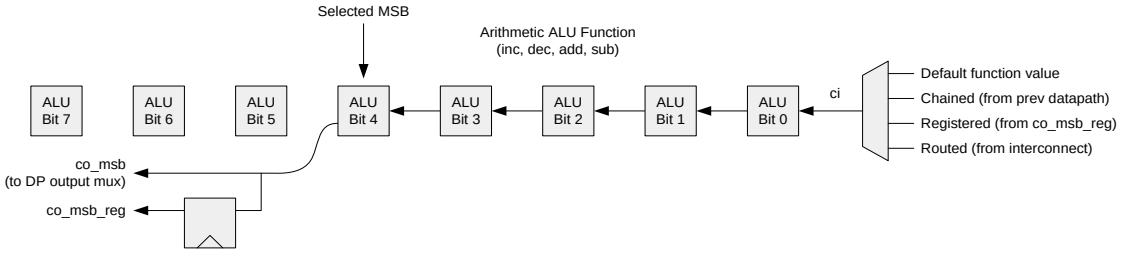
表 16-11 互连进位输出的操作

算术运算	进位输出信号极性	进位输出有效	进位输出无效
递加	正	++srca == 0	srca
递减	负	--srca == -1	srca
加	正	srca + srcb > 255	srca + srcb
减	负	srca – srcb < 0	(srca – srcb)

进位操作

具体的进位操作如图 16-16 所示。ALU 运算中，MSB 是可以通过编程来配置的；进位输入可以来自于四个不同的源；进位输出可以直接被互连通道输出，也可以被锁存到进位寄存器 co_msb_reg 中。

图 16-16 进位操作



移位操作

通过位于寄存器位 Shift[2:0]，可以配置 DP 移位操作的具体功能，具体如表 16-12 所示

表 16-12 移位操作中的功能配置

Shift[1:0]	功能
00	直通
01	左移
10	右移
11	高低半字节互换

配置寄存器 CFG15 的 SHIFT_SEL，DP 的输出 MUX 选择右移出（sor）和左输出（sol_msb）中的一个输出到互连通道，详见图 16-23。未发生移位操作时，sor 和 sol_msb 相当于 ALU 运算中的 LSB 和 MSB。

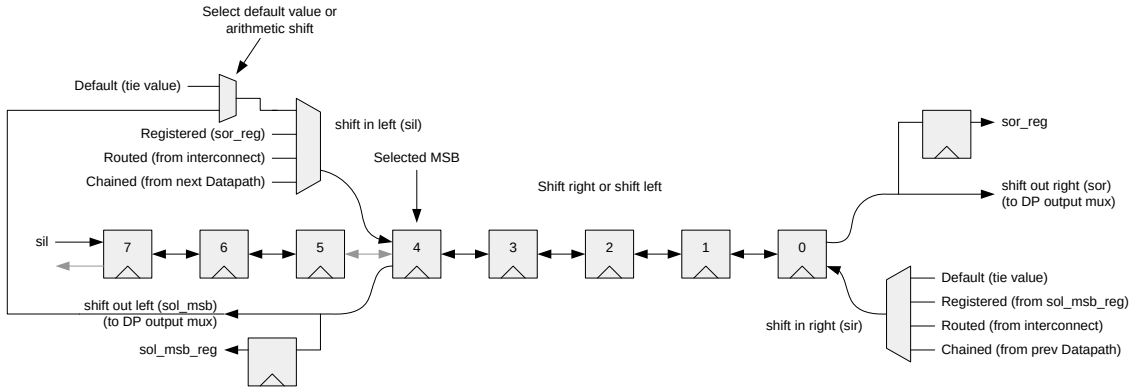
SI_SELA（CFG[1:0]）或 SI_SELB（CFG[3:2]）决定了移入操作数的来源，SI_SEL（DCFG[1]）可以逐周期的选择采用 SI_SELA 还是 SI_SELB 的配置。移入操作数只有在左移或右移操作时才有效（Shift[1:0] = 01 或 10）。SI_SELA 和 SI_SELB 的具体配置信息如表 16-13 所示。

表 16-13 移位操作的模式

SI SEL A SI SEL B	进位操作模式	描述
00	Default/ Arithmetic	默认模式：移位输入值为 DEF_SI（CFG12[4]）配置的缺省值（固定为 1 或 0）。但是，如果 MSB_SI（CFG15[4]）被置位，则移位输入值为当前周期的 MSB（此情况值发生在右移操作中）。
01	Registered	寄存器缓存模式：上一次移位操作的输出值存入移位寄存器中，作为下一次移位操作的输入值。左移操作使用上一次左移操作的移出值；右移操作使用上一次右移操作的移出出值。
10	Routed	互连模式：移位输入的值来自于 DP 输入 MUX 的输入信号 si。
11	Chained	级联模式：左移操作的级联输入来自于右边相邻 UDB 中的 DP；右移操作的级联输入来自于左边相邻 UDB 中的 DP。

如图 16-17 所示，左移操作的移出位从 MSB 的位置被输出；右移操作的移入位从 MSB 的位置被输入。在寄存器缓存模式中，左移操作和右移操作的移出位分别被锁存到寄存器 sol_msb_reg 和 sor_reg 中，可以在紧接的下一个周期中被使用，这使同一个移位寄存器在多周期中实现更高位宽的移位运算。

图 16-17 移位操作



MSB 以左的位称为隔离位，如图 16-17 中的 bit7，bit6 和 bit5，它们也参与移位运算。例如，左移操作中，bit4（MSB）的值被移入 sol_msb 和 sol_msb_reg 的同时也会被移入 bit5；右移操作中，sil 的值除被送到 bit4（MSB）外，也会被送到 bit7。但是一旦隔离位的移位值被移出隔离位的范围，就会被丢失，不会对正常的移位操作产生影响。

ALU 的屏蔽选项

ALU 的运算结果还可以与屏蔽寄存器做位与运算，从而可以屏蔽一些位的输出。屏蔽功能一个典型的应用场合是，通过设置输出屏蔽位，使同一个计数器或定时器表现出不同的工作周期。

16.2.2.5 DP 的输入多路器（MUX）

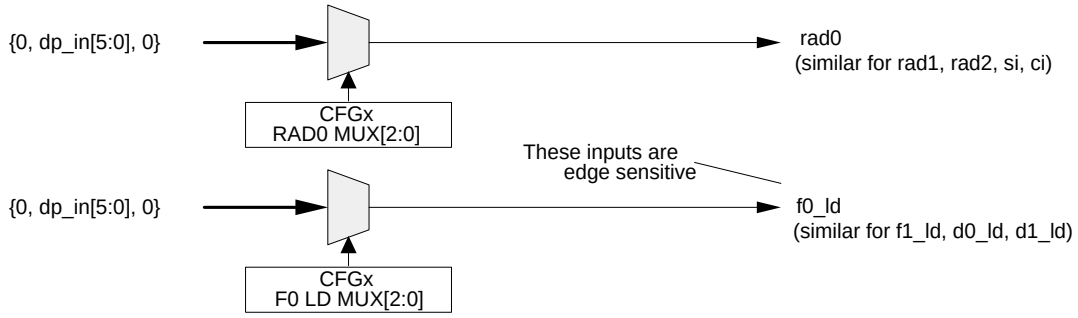
DP 共有 6 个输入（dp_in[5:0]），通过配置 9 个 6-1 输入 MUX（每个 MUX 有 6 个输入和 1 个输出），可以将输入配置为各种 DP 的内部输入信号，如表 16-14 所列。

表 16-14 DP 的输入信号

信号	描述
RAD2 RAD1 RAD0	异步动态配置寄存器的地址输入。动态配置寄存器中共有 8 个 16bit 的配置字可供寻址选择，通过输入不同的地址，动态配置寄存器就可以生成一个配置字序列，逐周期配置 DP 的工作。
F0 LD F1 LD	当这两个输入信号有效时，对应的 FIFO 会根据 Fx_INSEL[1:0]的配置从恰当的数据源载入数据。这两个信号可以是电平有效或者边沿有效的，具体工作情况参考 16.3.2.2。
D0 LD D1 LD	当这两个输入信号有效时，数据寄存器会从对应的 FIFO 中载入数据。这两个信号时上升沿有效的。
SI	移位操作的互连移入信号，可以参与左移位操作或者右移位操作。
CI	进位互连输入信号。

如图 16-18 举例所示，9 个 6-1MUX 分成两组，一组需要支持边沿有效的输入，另一组不需要。

图 16-18 DP 的输入选择



16.2.2.6 CRC/PRS

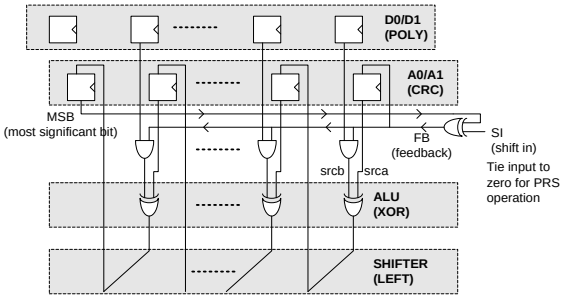
DP 能够生成循环冗余校验码（CRC）和 PRS（伪随机序列）。通过级联多个 DP，支持的位宽可以大于 8 位。

在 CRC/PRS 的运算中，最高有效模块的 MSB 被传输到最低有效模块与移位输入信号（SI）进行异或运算，生成反馈信号（FB）。FB 的作用是对数据寄存器的多项式系数进行门控，门控结果与累加器的当前值进行异或。异或的输出左移，存入累积器，作为下一个周期异或操作数。

图 16-19 给出了 CRC 运算的功能结构。数据寄存器存放多项式的系数，累加器存放初值，以及计算过程中的值。PRS 运算的功能结构与前者类似，此时 SI 为固定值“0”。

为了使能 CRC/PRS 运算，CFB_EN（CDFG[3]）必须被置位，使能反馈信号与 ALU 操作数 B（SRCB）的与运算。如果该位清零，那么反馈信号被禁用，始终为 1，DP 无法进行 CRC/PRS 运算，但可以进行其它算术逻辑运算。通过动态配置寄存器，可以使 CRC/PRS 运算与其他算术逻辑运算在 DP 中逐周期的交织执行。

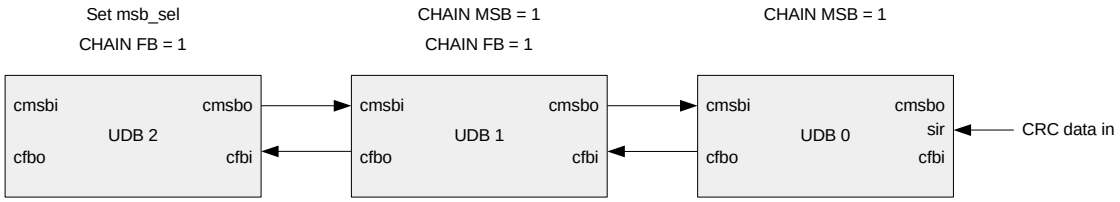
图 16-19 CRC 运算的逻辑电路



CRC/PRS 的级联

图 16-20 给出了一个 3 个 UDB 级联进行 CRC/PRS 运算的应用，可支持 17 位到 24 位的操作。如图所示，最低有效模块 UDB0 和 UDB1 的寄存器位 CHAIN_MSB（CFG14[3]）需要被置位；最高有效模块 UDB2 和 UDB1 的寄存器位 CHAIN_FB（CFG14[3]）需要被置位。通过 UDB2 的寄存器位 MSB_SEL[2:0]（CFG14[6:4]）可以配置运算的位宽（17bit - 24bit）。

图 16-20 CRC/PRS 级联的配置



反馈信号的级联：

- 最低有效模块 UDB0 负责产生反馈并通过 cfbo 信号输出到上级有效模块，它是 cmsbi 与 sir 的异或运算结果，其中，cmsbi 是上级有效模块（UDB1）输入的 MSB，sir 是 UDB0 的右移位输入信号（注意：当执行 PRS 运算时，sir 始终为 0）；
- 对于 UDB2 和 UDB1，它们的反馈输入信号由各自的下级有效模块级联过来。

最高有效位（MSB）信号的级联：

- 最高有效模块 UDB2 负责将 MSB 通过 cmsbo 信号输出到下级有效模块，UDB2 中 MSB 的具体位置是通过寄存器 MSB_SEL[2:0]（CFG14[6:4]）和 MSB_EN(CFG14[7])来配置的。
- 对于 UDB1 和 UDB0，它们的 MSB 信号由各自的上级有效模块级联过来。

CRC/PRS 中的多项式配置

举一个例子来说明 CRC/PRS 运算中的具体配置。定义一个 CCITT 的 CRC-16 算法的多项式 x16 + x12 +x5 + 1，它在数据寄存器中对应的系数值如图 16-21 所示。

图 16-21 CCITT 的 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 配置范例

下面给出了具体的配置，假定数据寄存器 D0 存放多项式系数，CRC/PRS 在累加器 A0 中进行运算：

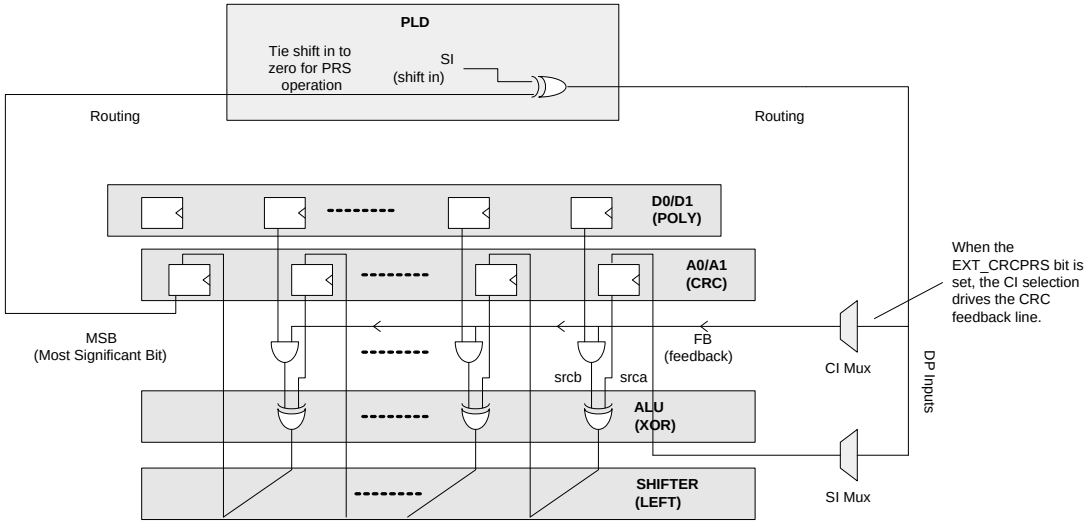
1. 选择一组合适的多项式系数写入 D0；
2. 将运算的初始值写入 A0；
3. 如果需要级联 DP 以扩展运算，则按照前文进行相关配置；
4. 通过寄存器 MSB_SEL[2:0]（CFG14[6:4]）和 MSB_EN(CFG14[7])配置最高有效位 MSB 的位置；
5. 操作动态配置寄存器：
   a) 将 D0 配置为操作数 B；
   b) 将 A0 配置为操作数 A；
   c) 将 ALU 的运算函数配置为异或；
   d) 将移位寄存器配置为左移；
   e) 如果是 PRS 运算，将移位信号设为固定值“0”；
   f) 将 CFB_EN 置位；
   g) 将 A0 的数据源配置为 ALU。

CRC/PRS 的外部模式

将寄存器位 EXT_CRCPRS（CFG16[1]）置位，可以在 DP 之外的模块中进行 CRC/PRS 的一部分运算。如图 16-22，CRC 的 FB 计算在 PLD 中进行这时，CRC 的 FB 通过输入 MUX 进入 DP。原先内部的 FB 运算逻辑被旁路。通过对一个 UDB 的时分复用，可以实现 CRC/PRS 运算的位宽可达到 16bit。

在这个模式中，通过动态配置寄存器中的 CFB_EN 位（CDFG[3]）同样可以使能或者禁止反馈信号与 ALU 操作数 B（SRCB）的与运算。这样 CRC/PRS 运算与其他算术逻辑运算在 DP 中逐周期的交织执行。

图 16-22 CRC/PRS 外部模式的逻辑电路



16.2.2.7 DP 的输出多路器

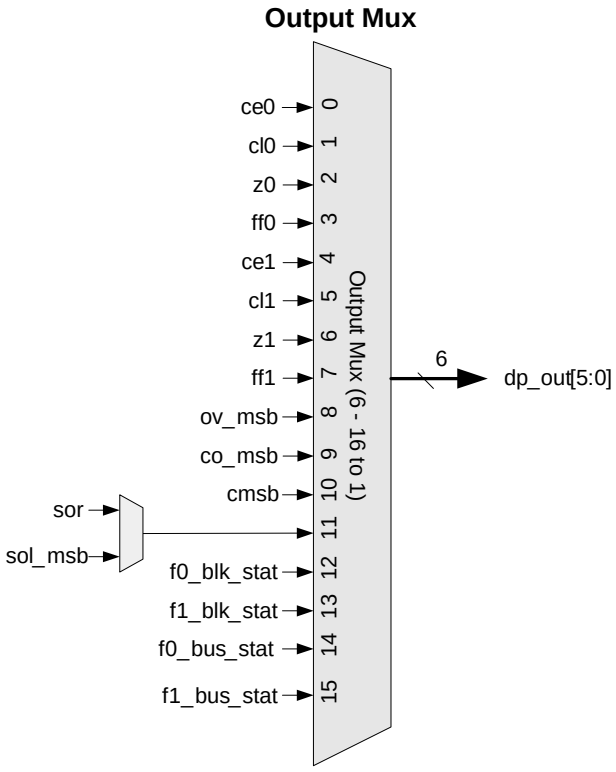
DP 能够对累加器，ALU 输出和 FIFO 状态进行条件判决。判决结果可以通过数字互连通道输出，供 UDB 中的其它模块使用，或者产生中断，或者直接输出到芯片的 I/O 管脚。表 16-15 给出了所有 16 个可被 DP 判决的条件。

表 16-15 DP 的判决条件

名称	判决条件	是否可级联	判决表达式和描述
ce0	A0 和 D0 相等	是	A0 == D0
cl0	A0 小于 D0	是	A0 < D0
z0	A0 为全 0	是	A0 == 00h
ff0	A0 为全 1	是	A0 == FFh
ce1	相等	是	A1 or A0 == D1 or A0 (dynamic selection)
cl1	小于	是	A1 or A0 < D1 or A0 (dynamic selection)
z1	A1 为全 0	是	A1 == 00h
ff1	A1 为全 1	是	A1 == FFh
ov_msb	溢出	否	Carry(msb) ^ Carry(msb-1)
co_msb	进位输出	是	进位输出
cmsb	CRC/PRS 的最高有效位	是	CRC/PRS 的最高有效位
so	移位输出	是	移位输出
f0_blk_stat	FIFO0 的 Block Status 状态信号	否	具体意义取决于 FIFO 的配置
f1_blk_stat	FIFO1 的 Block Status 状态信号	否	具体意义取决于 FIFO 的配置
f0_bus_stat	FIFO0 的 Bus Status 状态信号	否	具体意义取决于 FIFO 的配置
f1_bus_stat	FIFO1 的 Bus Status 状态信号	否	具体意义取决于 FIFO 的配置

DP 共有 6 个输出，通过 6 个 16-1 输出 MUX，上表中的 16 个条件判断的结果可以输出到互连通道上。

图 16-23 输出多路器的连接



比较器

DP 中共有两个比较器，第一个比较器（比较器 0）的操作数是固定的（A0 和 D0），另一个比较器（比较器 1）的操作数能够被动态配置。操作数在输入比较器前经过一个 8bit 的可配置屏蔽寄存器，这使比较运算可以在指定的比特位上进行。默认情况下，屏蔽寄存器是禁止的。比较器可以进行两种比较运算：第一个操作数是否“小于”第一个操作数（1 表示小于，0 表示不小于）；第一个操作数是否“等于”第二个操作数（1 表示等于，0 表示不等于）。

如表 16-16 所示，通过寄存器 CMP_SELA（CFG[5:4]）或 CMP_SELB（CFG[7:6]），可以选择比较器 1 的两个操作数。通过动态配置寄存器中的 CMP_SEL（DCFG[0]），可以逐周期的选择比较器 1 根据 CMP_SELA 还是 CMP_SELB 来选择操作数。

表 16-16 比较器配置

CMP SEL A CMP SEL B	描述
00	A1 与 D1 比较
01	A1 与 A0 比较
10	A0 与 D1 比较
11	A0 与 A0 比较

通过配置寄存器，可以选择比较器是否与相邻 DP 的比较器进行级联，图 16-24 和图 16-25 分别给出了“等于”和“小于”比较运算级联的逻辑电路。

图 16-24 “等于”比较运算级联逻辑电路

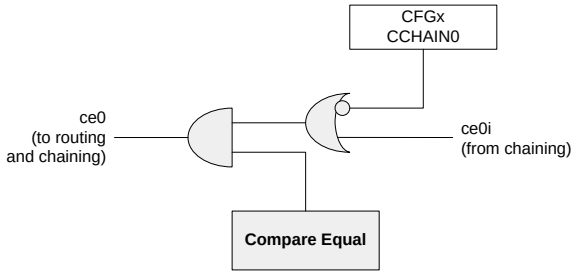
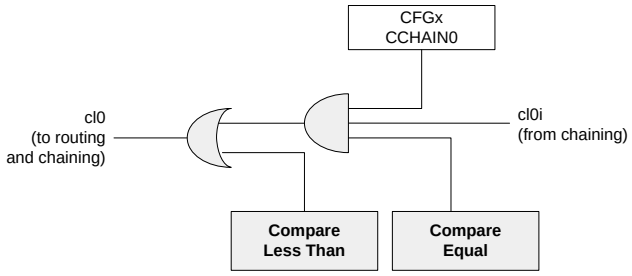


图 16-25“小于”比较运算级联逻辑电路



全 0 和全 1 的条件判决

累加器能够对自身的值进行全 0 和全 1 的条件判决。一个 DP 中的判断结果可以分别与相邻 DP 中的全 0 和全 1 条件判断器级联，级联的逻辑电路与“等于”比较级联的结构类似。

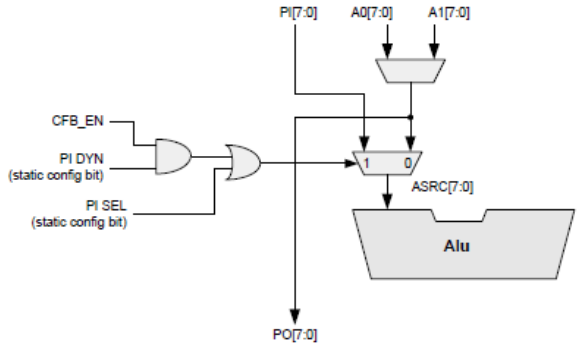
溢出操作

将 DP 的进位输入与进位输出做异或运算来做 DP 是否溢出的条件判断。这个条件判断是不可以级联的，在多个 DP 级联的应用中，各个 DP 独立进行各自的溢出判断。

16.2.2.8 DP 的并行输入和并行输出

如图 16-26 所示，PI[7:0]和 PO[7:0]分别是 DP 与互连通道通信的并行输入信号和并行输出信号。通过配置 SRC_A（DCFG[12]），可以将 DP 内部 A0 或 A1 的值并行输出到互连通道。

图 16-26 DP 并行输入和并行输出的连接



ALU 的操作数 A 的选择有两种操作方式。在静态操作方式中，通过将 PI_SEL（CFG15[7]）置位，可以将操作数 A 固定选择为 PI[7:0]；在动态操作方式中，将 PI_SEL 清零，将 PI_DYN（CFG15[5]）置位，通过 CFB_EN 来动态的选择 PI[7:0]或者 Ax[7:0]作为操作数 A。

说明：

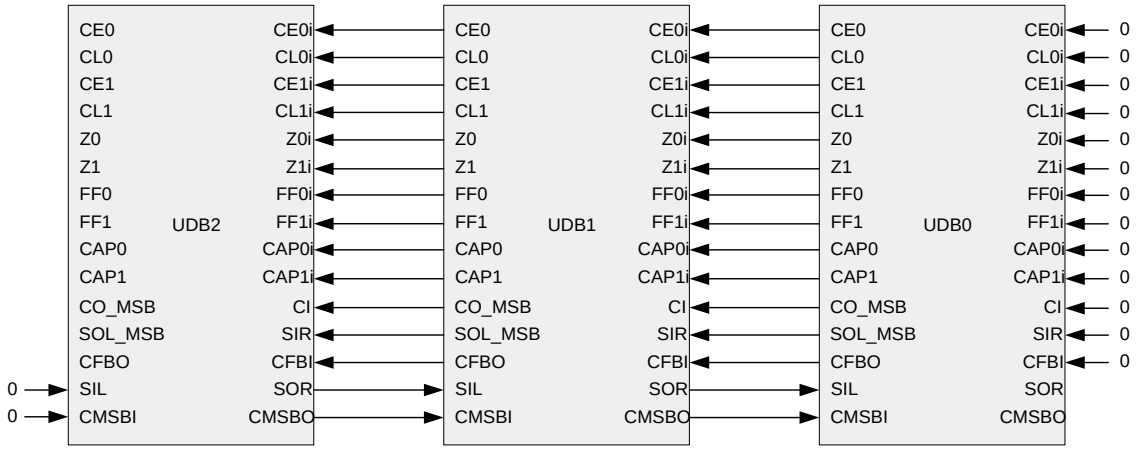
- 动态操作模式中，具体选择 A0[7:0]或者 A1[7:0]作为操作数 A，这取决于 SRC_A 的配置；
- CFB_EN 的主要功能是使能 CRC/PRS 的运算，当在动态操作方式被用来选择操作操作数 A 的数据源时，它不再具有使能 CRC/PRS 运算的功能。

16.2.2.9 DP 的级联

DP 包含一个 8bit 的 ALU，相邻 UDB 中的 DP 通过级联进位，捕捉触发信号，移位数据，条件判断信号能够支持更高位宽和更复杂的运算。这些级联信号通过专属互连通道进行传输，所以不用担心设计中的级联应用带来任何的时序问题。另外，捕捉触发信号的级联为 CPU 同时读多个 UDB 中 DP 的累加器提供了支持（详见 16.3.2.2 中的软件捕获模式）。

如图 16-27 所示，几乎所有级联信号的传输方向都是从低有效模块到高有效模块，右移位操作信号和 MSB 信号的级联是从高有效模块到低有效模块。

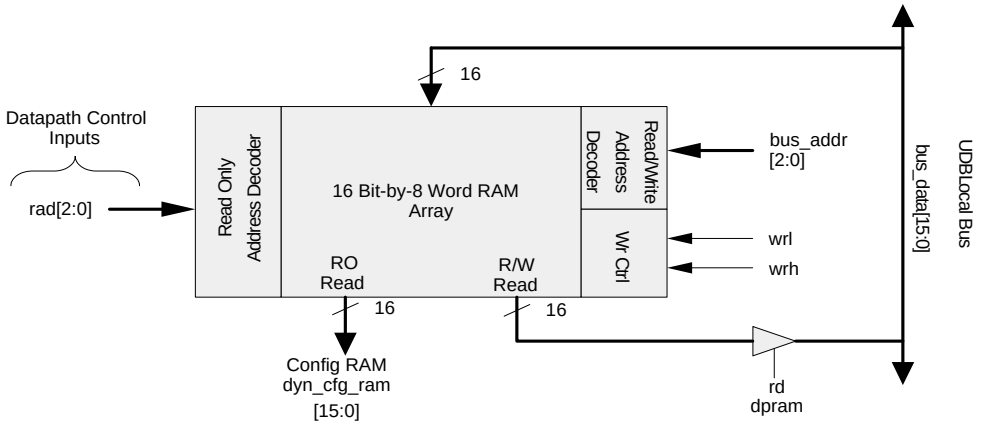
图 16-27 DP 级联时的信号流



16.2.2.10 动态配置寄存器

每个 DP 中包含一个动态配置寄存器，它的大小为 16bit x 8，如图 16-28 所示。通过 RAD[2:0]进行异步寻址，选择不同的配置字。动态寄存器提供了一个异步只读端口，将被选择的 16bit 配置字输出到 DP，对 DP 的工作进行逐周期动态配置。寻址信号通过 DP 的 MUX 输入，它的来源可以使互连信号，包括 I/O 管脚，PLD 的输出，控制模块的输出以及其他的 DP 输出。动态配置寄存器也为总线提供了读写端口，这些端口是与系统时钟同步的。

图 16-28 动态配置寄存器的输入输出



下面给出了动态配置寄存器的具体地址，以及其中各个比特位的具体说明。

Register	Address	15	14	13	12	11	10	9	8
CFGRAM	61h - 6Fh (Odd)	FUNC[2:0]			SRCA	SRCB[1:0]		SHIFT[1:0]	

Register	Address	7	6	5	4	3	2	1	0
CFGRAM	60h - 6Eh (Even)	A0 WRSRC[1:0]		A1 WRSRC[1:0]		CFB EN	CI SEL	SI SEL	CMPSEL

表 16-17 动态配置寄存器列表

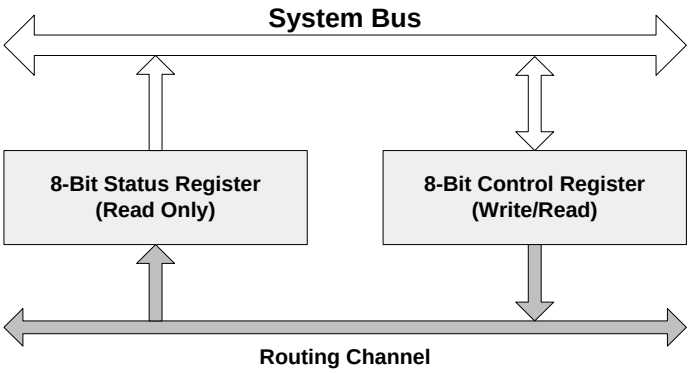
寄存器名	位宽	功能	详述
FUNC[2:0]	3	ALU 的运算种类的选择	000 直通 001 操作数 SRCA 递加 010 操作数 SRCA 递减 011 加 100 减 101 异或 110 与 111 或
SRCA	1	ALU 中操作数 SRCA 的选择	0 A0 1 A1
SRCB	2	ALU 中操作数 SRCB 的选择	00 D0 01 D1 10 A0 11 A1
SHIFT[1:0]	2	移位操作的选择	00 直通 01 左移 10 右移 11 高半字节与低半字节互换
A0 WR SRC[1:0]	2	选择由谁写 A0	00 None 01 ALU 10 D0 11 F0
A1 WR SRC[1:0]	2	选择由谁写 A1	00 None 01 ALU 10 D1 11 F1
CFB EN	1	使能或禁止	0 使能
		CRC 运算中的反馈信号（FB）	1 禁止
CI SEL	1	进位输入操作数的配置字选择	0 ConfigA 1 ConfigB[a]
SI SEL	1	移位输入操作数的配置字选择	0 ConfigA 1 ConfigB[a]
CMP SEL	1	比较器的配置字选择	0 ConfigA 1 ConfigB[a]

a. 对于 CI、SI 和 CMP 的操作，通过动态配置寄存器可以选择两套不同的配置字：ConfigA 和 ConfigB，两套配置字具体由另外的寄存器决定。

16.2.3 状态与控制模块

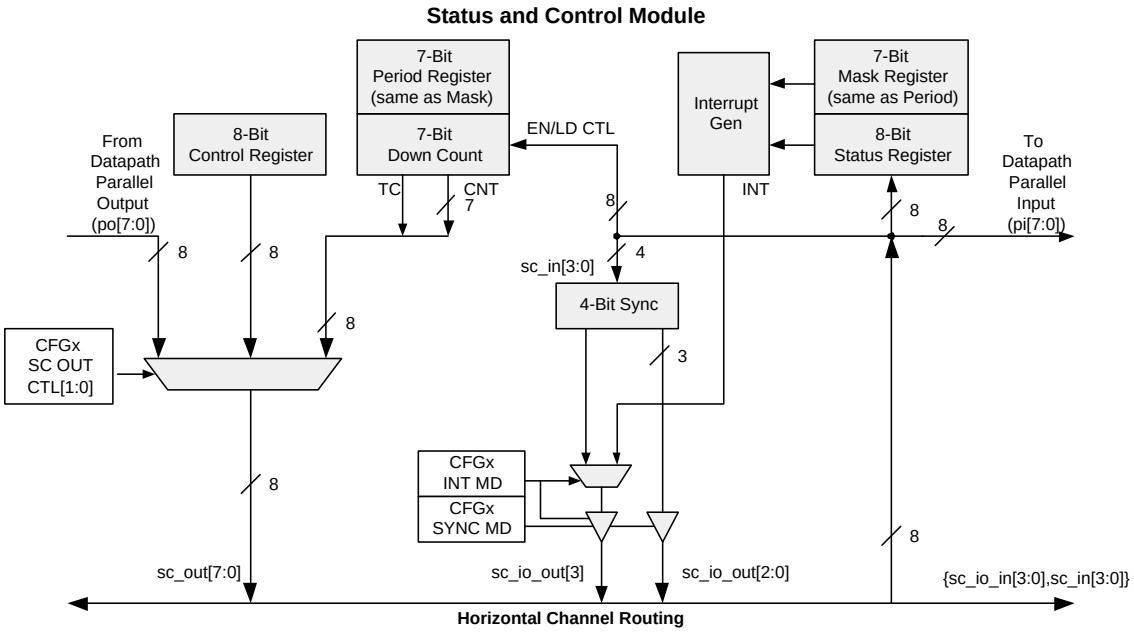
状态与控制模块两个主要的寄存器如图 16-29 所示。状态寄存器为固件监控 UDB 状态提供了途径，对固件来说，它是只读的；控制寄存器为固件控制 UDB 操作提供了途径，对固件来说，它是可读可写的。

图 16-29 状态寄存器与控制寄存器



该模块的内部细化图如图 16-30 所示。这个模块的主要作用是协调 CPU 固件与 UDB 的工作，但是基于它丰富的互连矩阵资源，也可以通过配置来执行其他功能。

图 16-30 整体框图



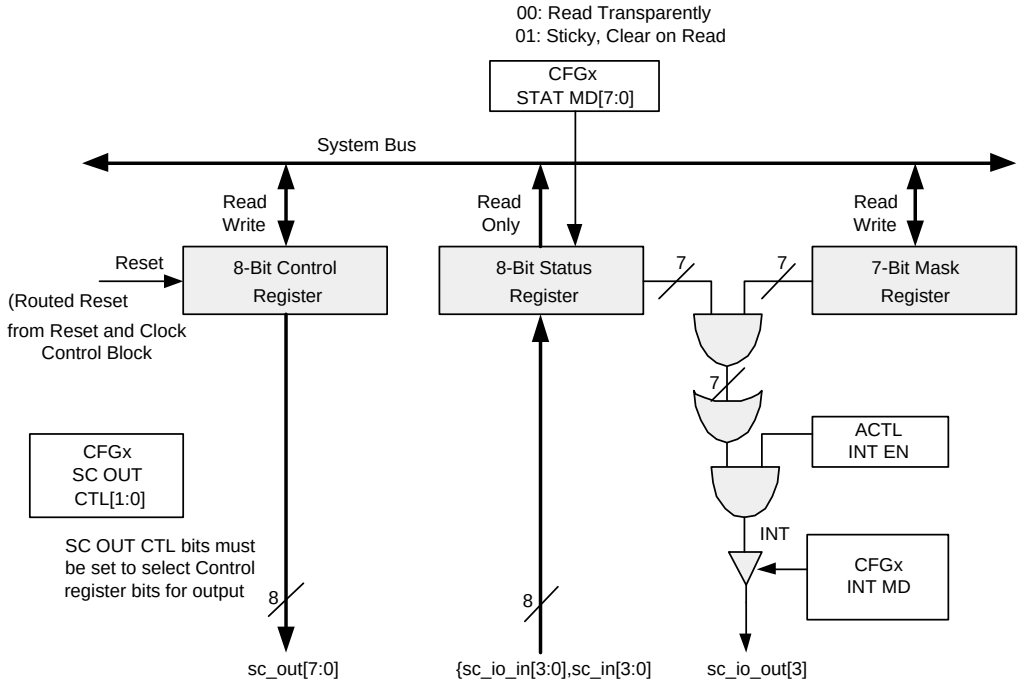
该模块的操作模式如下：

- **状态输入** — 通过读状态寄存器，CPU 读取和捕捉互连信号 sc_io_in[3:0]和 sc_in[3:0]的数据；
- **控制输出** — 通过写控制寄存器，CPU 驱动互连信号 sc_out[7:0]；
- **并行输入** — SC 模块将互连信号 sc_io_in[3:0]和 sc_in[3:0]连接到 DP 的并行输入信号 PI[7:0]；
- **并行输出** — SC 模块将 DP 的并行输出信号 PO[7:0]连接到互连信号 sc_out[7:0]；
- **计数器模式** — 控制寄存器工作为一个 7bit 的周期可配置的递减计数器，支持周期自动重载。通过互连信号能够使能计数或者主动重载周期。在此模式下，其它对控制寄存器的操作是非法的；
- **同步模式** — 状态寄存器工作为 4bit 的双同步器。在此模式下，其它对状态寄存器的操作是非法的。

16.2.3.1 状态输入模式

当模块工作在状态输入和控制输出模式下时，简化的模块内部结构如图 16-31，主要的组成部分有一个 8bit 的控制寄存器，一个 8bit 的状态寄存器和一个 7bit 的中断屏蔽寄存器。

图 16-31. 状态寄存器和控制寄存器的操作



状态寄存器的操作

一个 UDB 中有一个 8bit 的状态寄存器，对 CPU，该寄存器是只读的，它的状态由对应的互连信号输入。状态寄存器无数据保持功能，PSoC 4 进入睡眠状态将丢失原状态，唤醒后，被复位到 0x00。状态寄存器的位读取有两种模式，由寄存器 STAT_MD[7:0]分别配置，具体如下表：

表 16-18. 状态寄存器位读取模式的配置

STAT MD	描述
0	透传读取模式：CPU 读状态寄存器某一位时，状态寄存器将该位对应互连信号的当前值透传给 CPU。
1	粘滞读取模式：互连信号上一旦出现逻辑值“1”，状态寄存器对应位的值被锁定为“1”，只有 CPU 读取该位后，才会被清零。

状态寄存器按位独立操作，位与位之间互不影响，如：其中一位捕捉成功，进入锁定状态之后，并不影响其它位继续捕捉。

透传读取模式

CPU 通过系统总线进行读操作，状态寄存器返回对应互连信号的当前值。这种模式可以用来传输 UDB 中计算和操作的瞬间状态。

粘滞读取模式

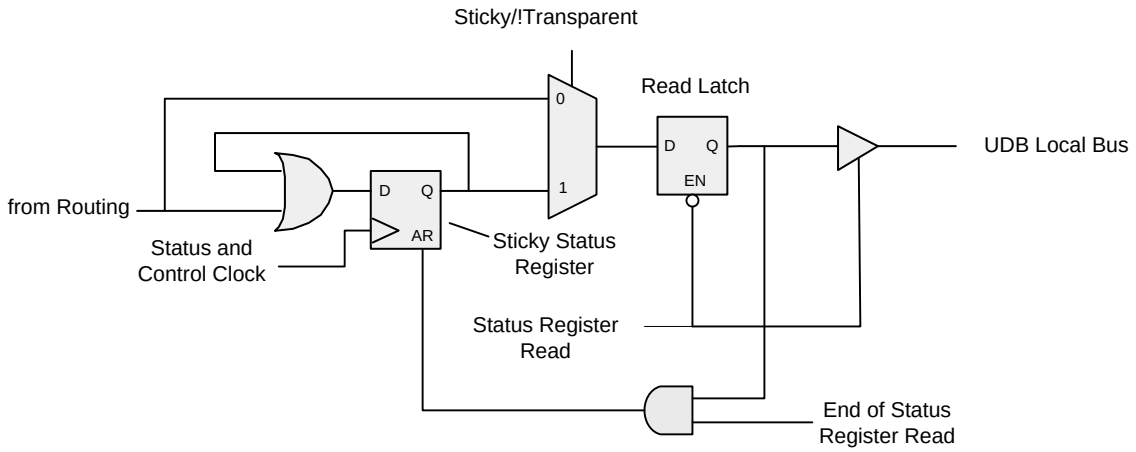
该模式下，在每个状态和控制时钟的周期中，状态寄存器捕捉对应互连信号的“1”值，一旦捕捉成功，状态寄存器一直保持“1”值，直到被 CPU 通过系统总线读取。被读取之后，状态寄存器自动清零，进入下一次捕捉。

这里涉及到 2 个时钟域，互连信号写状态寄存器是在模块时钟域；CPU 读取状态寄存器是在总线时钟域，二者相互独立。

读操作中的状态锁存

图 16-32 为状态寄存器的简化图。或门与 D 触发器组成了状态“1”保持电路；读锁存器保证读取过程中状态的稳定；通过 MUX 来切换状态寄存器工作在即时读取模式或者粘滞读取模式；与门确保状态寄存器被读取后自动清零。

图 16-32. 状态寄存器逻辑电路



中断的产生

在大多数应用中，中断的产生与状态位的设置相关联。如图 16-31 所示，通过配置寄存器位 INT_EN（ACTL[4]）和 INT_MD（CFG22[2]），寄存器的低 7 比特与屏蔽寄存器的 7 比特的逻辑运算结果作为中断由互连信号 sc_io_out[3]输出到中断控制器。这时，状态寄存器的最高有效位不参与逻辑运算，它存放上面逻辑运算的结果，即作为一个中断标志位，可供 CPU 查询。

16.2.3.2 控制输出模式

UDB 中有一个 8bit 的控制寄存器，对于 CPU，它是可读写的。状态寄存器各比特位的值可以通过互连信号（sc_out[7:0]）输出。控制寄存器无数据保持能力，进入睡眠状态后会丢失原逻辑值，唤醒后，被复位到 0x00。

控制寄存器的操作模式

控制寄存器一共有四种工作模式，对各比特位可以独立配置，具体取决于寄存器 CTL_MD1[7:0] 和 CTL_MD0[7:0] 的位组合。举例说明，{ CTL_MD1[0],CTL_MD0[0]}决定了控制寄存器中 bit0 位的工作模式，如表 16-19 所示，其它位的配置以此类推。

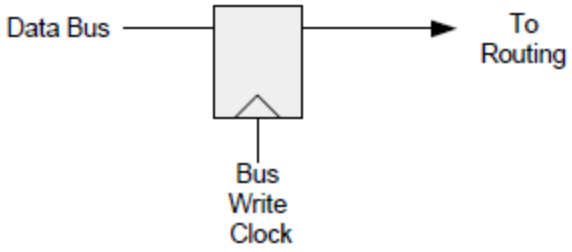
表 16-19. 控制寄存器 bit0 的操作模式

CTL MD	描述
00	Direct mode
01	Sync mode
10	Double sync mode
11	Pulse mdoe

直接模式

直接模式是控制寄存器的默认工作模式，如图 16-33 所示。CPU 将逻辑值写入控制寄存器后，控制寄存器直接利用该逻辑值驱动互连通道，该工作模式不需要 SC 时钟的参与。

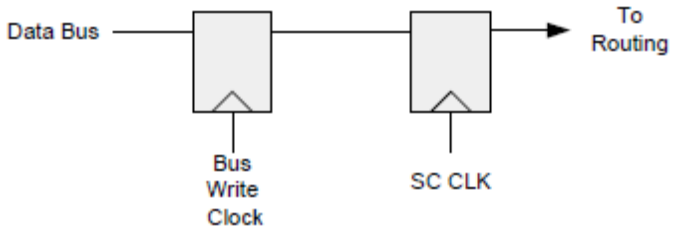
图 16-33. 控制寄存器的直接控制模式



同步模式

如图 16-34 所示，同步模式中，控制寄存器的逻辑值在驱动互连通道之前，会通过一个以 SC 时钟为工作时钟的 D 触发器，进行同步。互连通道的逻辑值是与 SC 时钟同步的，而在前一个工作模式中，它是与总线写时钟同步的。

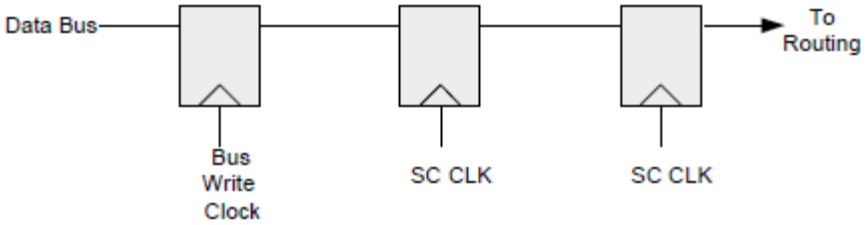
图 16-34. 控制寄存器的同步操作模式



双同步模式

在双同步模式中，我们在同步模式的基础上，又多加了一次同步。在总线时钟与 SC 时钟异步的情况下，这种工作模式能够增加信号链路的可靠性。

图 16-35. 控制寄存器的双同步操作模式



脉冲模式

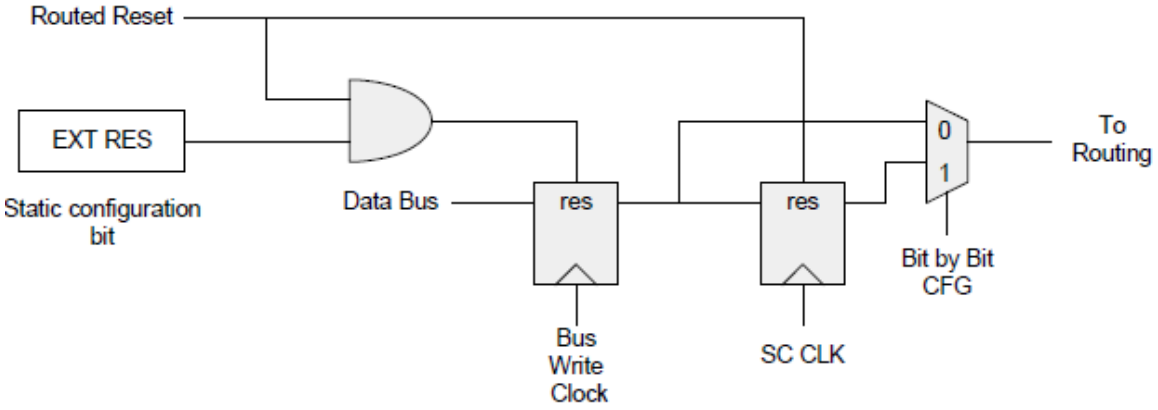
脉冲模式与同步模式有类似之处，控制寄存器的逻辑值在驱动互连通道之前，都会与 SC 时钟进行同步；不同之处在于，脉冲模式中，控制寄存器的逻辑值“1”只能持续一个脉冲，此后会被自动清零，该脉冲的宽度为一个 SC 周期。因此，在由控制寄存器驱动的互连通道上，该逻辑值也只能持续一个脉冲。

在这种工作模式下，固件向控制寄存器的某一位写“1”来产生脉冲。只有在控制寄存器该位的回读结果为“0”（驱动互连通道的脉冲信号已经结束）的情况下，才可以再次往控制寄存器的该位写“1”。可以推知，脉冲的最大频率为 SC 时钟频率的一半。

控制寄存器的复位

控制寄存器共有 2 种复位模式，如图 16-36。当 EXT_RES = 0（默认状态），互连复位（1 有效）只复位 D 触发器，而不复位控制寄存器的比特位；当 EXT_RES = 1，D 触发器和控制寄存器的比特位均被复位。

图 16-36. 控制寄存器复位的逻辑电路

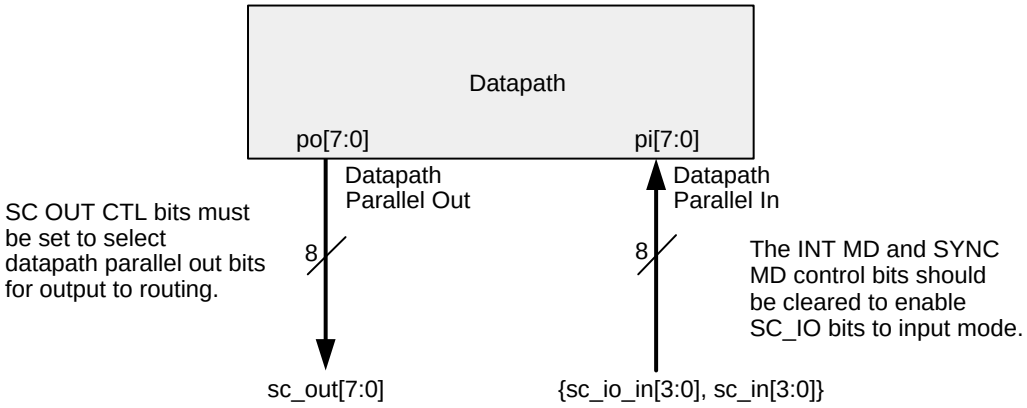


16.2.3.3 并行输入和并行输出模式

如图 16-37，DP 的并行输入端口 pi[7:0]和并行输出端口 po[7:0] 连接的互连通道分别是 sc_out[7:0] 和 {sc_io_in[3:0], sc_in[3:0]}，SC 模块与 DP 共享这些互连通道。

通过配置寄存器位 SC_OUT_CTL（CFG22[1:0]）可以选择 sc_out[7:0]被 SC 模块还是 DP 使用。当 sc_out[7:0]被 SC 模块使用时，还分为 2 种情况，或者输出状态寄存器的逻辑值，或者输出计数器的值，这都可以通过 SC_OUT_CTL（CFG22[1:0]）配置。

图 16-37. 并行输入与输出模式框图



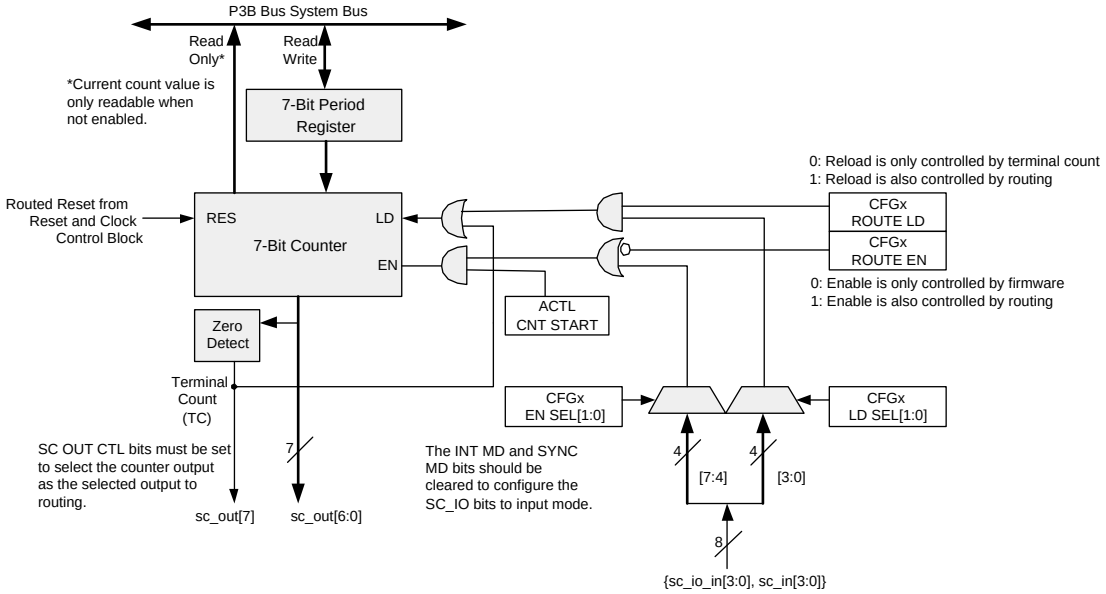
16.2.3.4 计数器模式

SC 模块可以被配置为向下计数器，如图 16-38。计数器的具体特性如下：

- 一个可读写的 7bit 周期寄存器；
- 一个可读写的 7bit 计数寄存器，只有在计数器被禁止时才可以访问该寄存器；
- 当终端计数（TC: Terminal Count）为 0 时，自动将周期重载到计数寄存器；
- 通过固件配置 CNT_START（ACTL[5]），可以启动或停止计数器（另一种方法是通过互连信号直接控制计数器的启动或停止）；
- 通过配置 ROUTE_LD（CFG23[4]）和 ROUTE_EN（CFG23[5]），选择互连信号操作计数器：
  - 启动或者停止计数器；
  - 强制重载计数器，此时不关心 TC 的逻辑值；
- 计数器的计数值可以通过互连信号 sc_out[6:0]输出；
- TC 的逻辑值可以通过互连信号 sc_out[7]输出；
- 通过配置寄存器位 ALT_CNT（CFG23[6]）可以选择计数器工作在默认模式或 ALT（Alternate）模式：
  - 默认模式中，TC 是时序逻辑值，只有通过互连信号启动或停止寄存器，才可通过互连信号重载计数器；
  - ALT 模式中，TC 是组合逻辑值，通过互连信号启动或停止寄存器和通过互连信号重载计数器是相互独立的。

通过配置寄存器 SC_OUT_CTL[1:0]，可以将计数器的值通过 sc_out[7:0]输出，其中 sc_out[6:0]输出的是计数值，sc_out[7]输出的是 TC。当 SC 模块工作在计数器模式下，控制寄存器的其它操作是非法的；状态寄存器仍然是可读的，但是无法产生中断，因为此时的屏蔽寄存器被用作计数器的周期寄存器，周期寄存器具有数据保持功能。如果要将计数器的周期设置为 N，那么需要在周期寄存器中写入 N-1。向周期寄存器写 0（N = 1）是非法的，导致的结果是计数值维持 1 不变。

图 16-38. 计数器框图

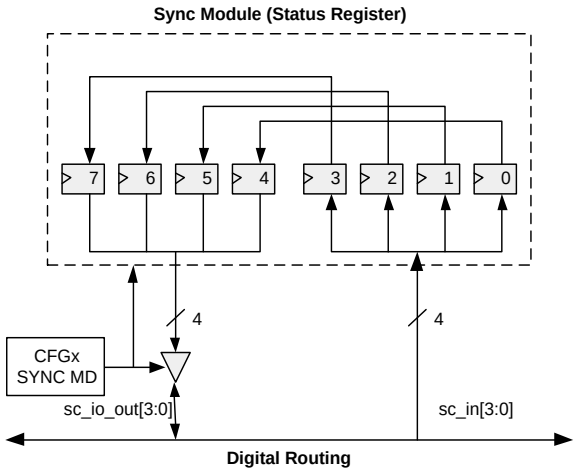


16.2.3.5 同步模式

同步模式下，状态寄存器可以作为 2 个串联的 4bit 的双同步器，将输入信号同步到 SC 模块的工作时钟上，如图 16-39 所示。当同步模式使能后，互连信号 SC_IN[3:0]作为同步器的输入；SC_IO_OUT[3:0]作为输出。无论 STAT_MD[7:0]如何设置，状态寄存器的粘滞读取模式被禁止。此时，状态寄存器的其它操作是非法的；控制寄存器的操作不受影响。控制寄存器依然可以工作计数器模式下，但是此时不能够使能 LD 和 EN 互连信号。

如果计数器模式中使用了互连信号来操作计数器，则同步模式无法启动，因为二者使用的互连信号有交叠。如果计数器模式中没有使用互连信号操作计数器，则一个 SC 模块可以同时工作在计数器模式和同步模式。

图 16-39. 双同步器的框图



16.2.3.6 状态与控制模块的时钟

当工作以下模式中，状态寄存器或者控制寄存器需要工作时钟：

- 状态寄存器中的任意一位工作在粘滞读取模式；
- 控制寄存器工作在计数器模式；
- 状态寄存器工作同步模式。

关于时钟的更多细节，请参考 16.3.4 章节。

16.2.3.7 辅助控制寄存器

辅助控制寄存器是 SC 模块中的一个具有固定硬件配置功能的特殊寄存器，通过它 CPU 可以动态的控制中断，FIFO 和计数器的操作，具体描述如下：

Auxiliary Control Register							
7	6	5	4	3	2	1	0
		CNT START	INT EN	FIFO1 LVL	FIFO0 LVL	FIFO1 CLR	FIFO0 CLR

FIFO0 Clear 与 FIFO1 Clear

FIFO0_CLR 与 FIFO1_CLR 分别控制 FIFO0 和 FIFO1 的复位。当被置位时，对应的 FIFO 被同步重启并清零，此时 FIFO 不能够保存数据，仅相当于一个缓冲器。当被清零时，对应的 FIFO 才可以正常操作。

FIFO0 Level 与 FIFO1 Level

FIFO0_LVL 与 FIFO1_LVL 分别设定对应 FIFO 状态判别的阈值，如表 16-20 所示。更多细节，请参考 16.3.2.2 章节。

中断使能

当 SC 模块内置的中断产生逻辑被使能后，产生的中断可以通过互连信号 sc_io_out[3]输出。

计数器的启动

配置 SC_OUT_CTL[1:0]，令互连信号 sc_out[7:0]输出计数器的值，此时通过 CNT_START 位可以启动或者禁止计数器。

16.2.3.8 状态与控制寄存器的总结

表 16-21 对状态与控制寄存器做了一个总结。在不同的模式中，控制寄存器与屏蔽寄存器分别是和计数寄存器与计数周期寄存器复用的。

表 16-21. 状态寄存器与控制寄存器总结

模式	控制寄存器 /计数值寄存器	状态寄存器 /同步寄存器	屏蔽寄存器 /计数周期寄存器
控制输出模式	向互连通道输出控制信息	由互连通道输入状态信息或者作为同步寄存器	作为中断屏蔽寄存器
计数器模式	向互连通道输出计数值		作为计数周期周期寄存器 [a]
状态输入模式	向互连通道输出控制信息或者计数值	由互连通道输入状态信息	作为中断屏蔽寄存器
同步模式		作为同步寄存器	作为计数周期寄存器 [b]

a. 在计数器模式下，中断屏蔽寄存器被复用为计数周期寄存器，所有无法产生中断；

b. 在同步模式下，对状态寄存器的操作是非法的，屏蔽寄存器也无法实现屏蔽中断功能；如果控制寄存器工作在计数器模式下，屏蔽寄存器可以用来作为计数周期寄存器。

16.2.4 复位与时钟模块

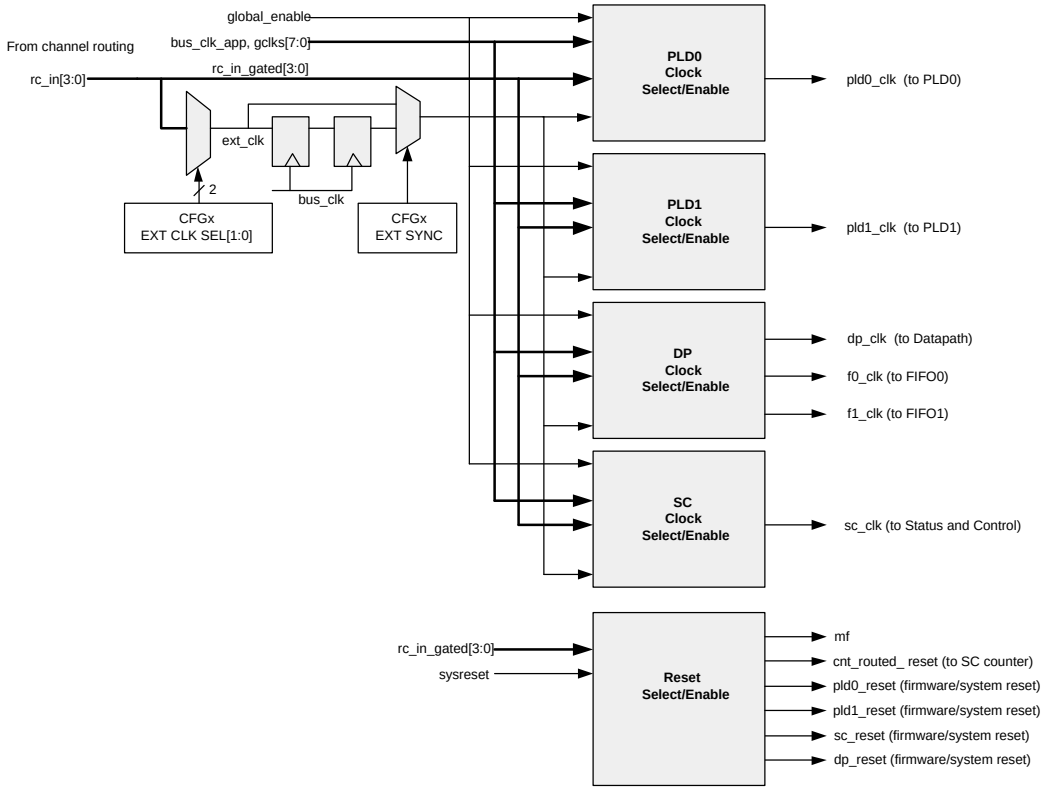
复位与时钟模块的主要作用是为 2 个 PLD，DP 和状态与控制模块选择复位源和时钟源。其中时钟源可以来自于全局系统时钟，总线时钟和外部时钟；复位源可以来自于互连信号和固件。如图 16-40 所示，UDB 共有 4 个时钟选择/使能模块和 1 个复位选择/使能模块。

对 UDB 来说，一共有 10 个时钟源可供选择，分别是 8 个全局数字时钟，总线时钟以及寄存器选定的外部时钟。需要声明，这里的总线时钟并不等同于系统总线时钟，它被称为“bus_clk_app”（如图所示），因为它是门控的，仅应用于 UDB 之中，其它的全局数字时钟也是同样。DP 中的时钟发生器一共产生三个时钟，一个是 DP 的整体工作时钟，还有两个分别供给它的 2 个 FIFO。

UDB 有四个互连信号 – RC_IN[3:0]。它们被输入到时钟选择/使能模块，每个模块可以根据配置选择其中的一个作为时钟使能信号，它可以被配置为电平敏感模式或边沿敏感模式；另一方面，通过 MUX 选择，其中的一个信号输入到各个时钟选择/使能模块中作为备选的外部时钟源。当然，在这同一个互连输入信号是不可能同时作为时钟使能信号和外部时钟源的。如图所示，外部时钟源在输入之前可以选择是否与总线时钟同步。

复位与时钟模块为 PLD 和状态控制模块提供了一个互连复位信号，同时也为各个模块提供了固件复位信号以支持重复配置。

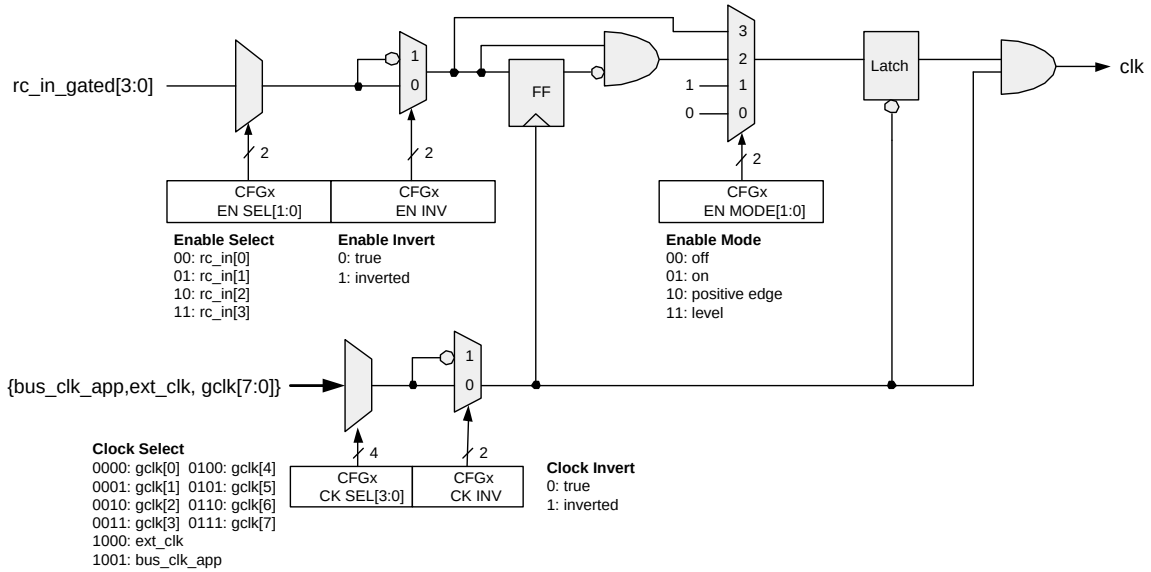
图 16-40. 复位与时钟的控制



16.2.4.1 时钟的控制

图 16-41 给出了一个时钟选择/使能模块内部电路的例子。在 UDB 中共有 4 个这样的电路，2 个 PLD 中各有 1 个，DP 和状态与控制模块中各有 1 个。该电路的主要组件有：1 个全局时钟 MUX，时钟反相器，时钟使能信号 MUX，时钟使能反相器和边沿检测器。

图 16-41. 时钟信号的选择与使能控制



时钟选择

时钟源的选择包括 8 个全局数字时钟，它们连接到每个 UDB，其中任意一个均可配置为 UDB 中任意一个模块的工作时钟。另外一个选择是总线时钟，它是系统中最高频率的时钟。最后一个选择是外部时钟，它通过互连信号 RC_IN[3:0]被送入 UDB 的各个模块。当设计一些需要外部提供时钟的逻辑功能模块，如 SPI，外部时钟会被选择为 UDB 中相关模块的时钟源。在实际设计中，存在一个 UDB 中不同模块被用来实现不同逻辑功能的需求，为 UDB 中各个模块提供独立的时钟源选择方案支持了这种需求的实现，提供了最大的设计灵活性。

时钟反向

通过配置，选定的时钟源可以被反向。但是此时有 2 个限制：

1. 因为时钟源被反向后，系统中会出现半周期数据通道，所以反向时钟源的频率应当比系统的最高频率要小，从而保证建立保持时间的时序要求；
2. 当反向时钟源的频率等于总线时钟时，同步的总线写与 UDB 内部写（如向一个正在计数的计数器中写一个新的值）是不支持的。这条限制影响的寄存器有 A0，A1，D0，D1 和 CTL（当作为 7bit 的计数器时）。

时钟使能选择

通过 MUX 选择互连信号 RC_IN[3:0]中的任意一个作为时钟使能信号输入。

时钟使能反向

通过配置，时钟使能信号也可以被反向。这个特点允许时钟使能的原始输入信号为任意极性。

时钟使能的模式

默认状态下，时钟是不被使能的（OFF 状态）。在配置完目标模块的操作之后，软件可以操作寄存器 CFGxEN MODE[1:0]，来设置时钟的工作状态，具体如表 16-22 所示。

表 16-22. 时钟使能的四种模式

时钟使能模式	描述
关闭	无时钟输入
连通	选择的信号源作为时钟输入
上升沿	一个时钟使能的上升沿触发一个时钟脉冲。显然时钟使能信号的最大频率必须小于时钟频率的 1/2
电平	时钟使能信号为高时，选择的信号源作为时钟输入

时钟使能的应用

时钟使能主要由 2 个使用场景：

固件使能 — 在实际应用中，几乎所有的逻辑功能模块都需要时钟使能功能。PSoC 4 中一个 UDB 可以被分割以实现多个简单的逻辑功能模块；多个 UDB 也可以被级联以实现一个复杂的逻辑功能模块，所以需要更加灵活的时钟使能控制。可以将控制寄存器的一位连接到一个或者多个时钟使能输入，并通过配置各个 UDB 中模块的时钟使能输入 MUX 来实现这样灵活的控制。

模拟本地时钟发生 — 此场景中，将时钟使能模式配置为上升沿，这样允许 UDB 产生本地时钟，如果其它 UDB 使用与此 UDB 同步的一个时钟使能信号，则它们也可以产生同样的本地时钟。在这个场景中，“时钟使能信号的最大频率必须小于时钟频率的 1/2”的限制将不再存在。

FIFO 的时钟方案

DP 中 FIFO 的时钟方案比较特别。默认情况下，FIFO 与 DP 的时钟配置是相同的，当时有以下的例外：

- DP 的时钟在进入 FIFO 之前，可以被反向；
- 当 FIFO 设置为 FAST 模式是，总线时钟会取代取代 DP 时钟作为 FIFO 的数据接收采样时钟。

16.2.4.2 复位控制

集中复位模式

复位控制有 2 种模式：集中模式与分散模式，通过设置 UDB 配置寄存器 CFG 的 ALT_RES 位，可以选择不同的控制模式。当 ALT_RES = 0 时，集中模式被选择，UDB 中所有模块共享一个互连复位信号；当 ALT_RES = 1 时，分散模式被选择，UDB 中不同的模块可以有不同互连复位信号。

表 16-23 给出了 2 种复位控制模式在不同方面的区别。

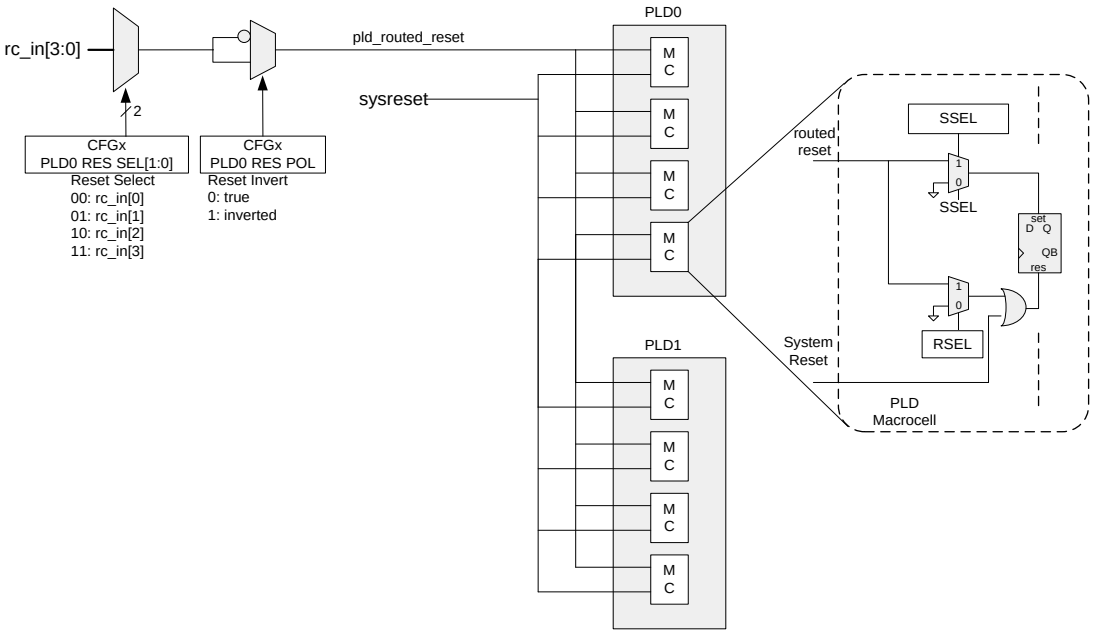
表 16-23. 复位模式

特征	集中复位模式	分散复位模式
灵活性	UDB 中的所有模块共享一个互连复位信号	每个 UDB 的组件可以有独立的互联复位信号
互连信号能否复位 SC 模块	不能复位状态寄存器	能够复位其中的一部分电路
互连信号能否复位 DP	否	能够复位其中的一部分电路

PLD 的集中复位模式

图 16-42 给出了工作在集中模式的 PLD 复位系统。这里选择了互连信号作为复位源，此外通过配置也可以选择固件控制的寄存器位作为复位源。

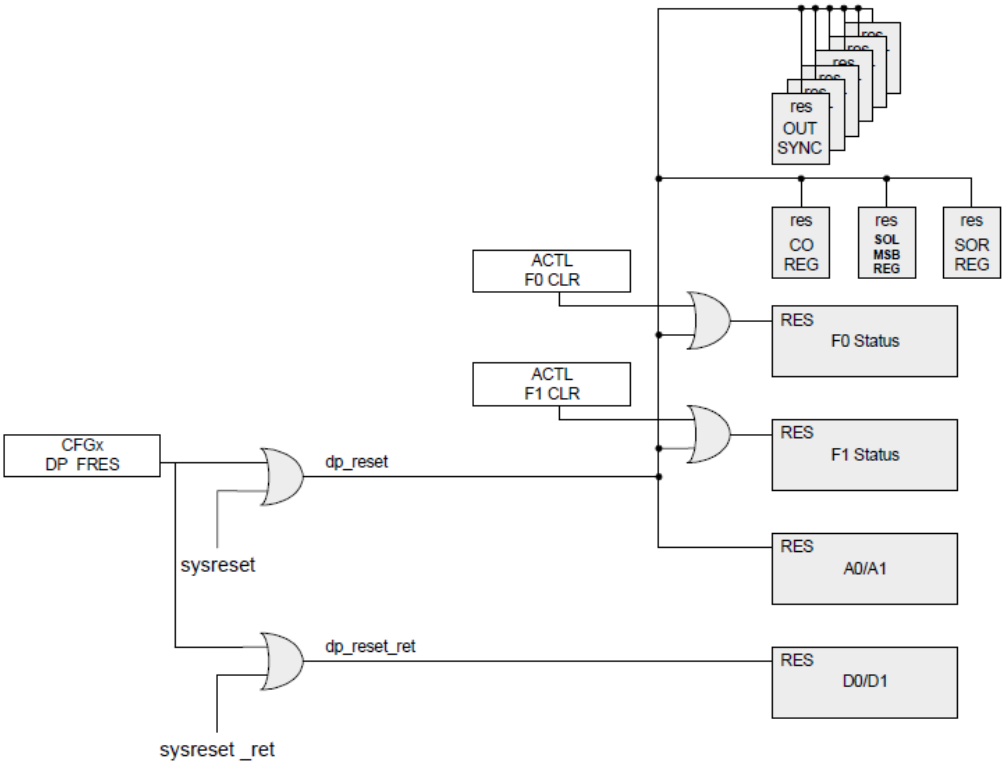
图 16-42. PLD 的集中复位模式



DP 的集中复位模式

图 16-43 给出了工作在集中模式的 DP 复位系统。由表 16-32 可知，在此模式下，互连信号不能够成为 DP 的复位源，只有固件控制的寄存器位作为 DP 的复位源。通过固件，可以异步清除 DP 的输出寄存器，进位和移位输出标志寄存器，FIFO 状态寄存器，累加器和数据寄存器。注意数据寄存器 D0 和 D1 具有数据保持功能，在睡眠状态中能够维持它们各自的状态。FIFO 中的数据是未知的，因为它们是基于随机存取存储器的。

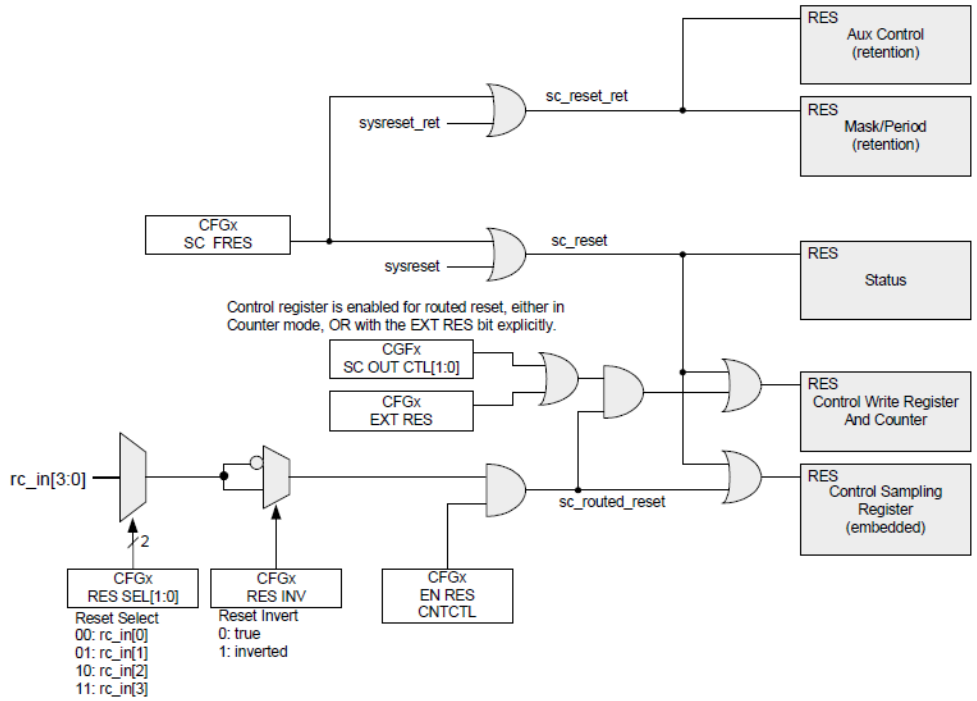
图 16-43. DP 的集中复位模式



SC 模块的集中复位模式

图 16-44 给出了工作在集中模式的状态与控制模块的复位系统，其中屏蔽/周期寄存器和辅助控制寄存器具有数据保持功能。

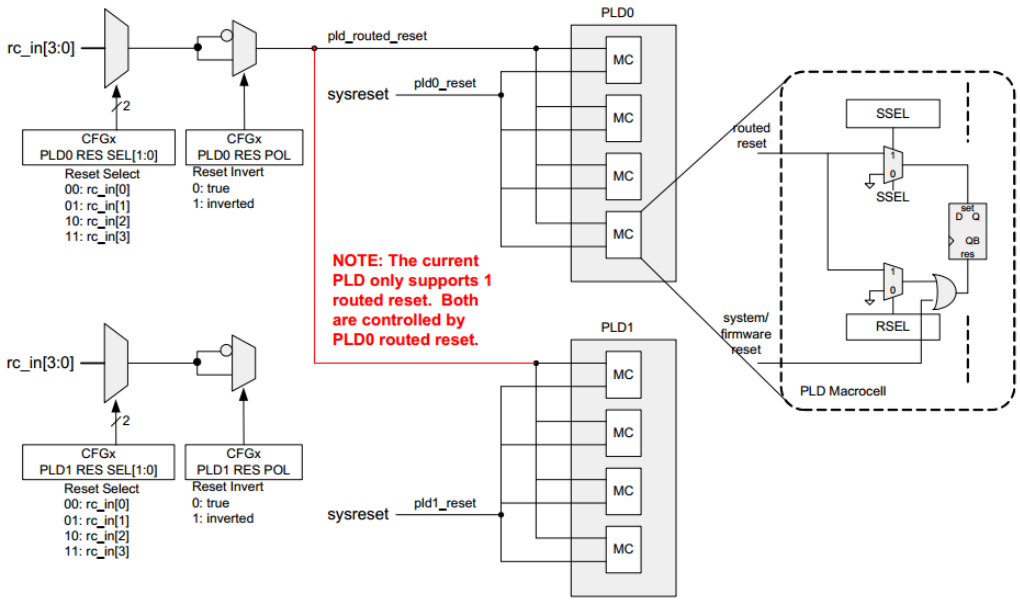
图 16-44. SC 模块的集中复位模式



PLD 的分散复位模式

图 16-45 给出了工作在分散模式的 PLD 复位系统。虽然在此模式下，UDB 中各个模块可以应用不同的互连信号作为复位源，但是其中的 2 个 PLD 必须应用同一个互连信号作为它们共同的复位源，具体选择哪个互连信号，是否做反向处理，由 PLD0 复位系统的配置决定。

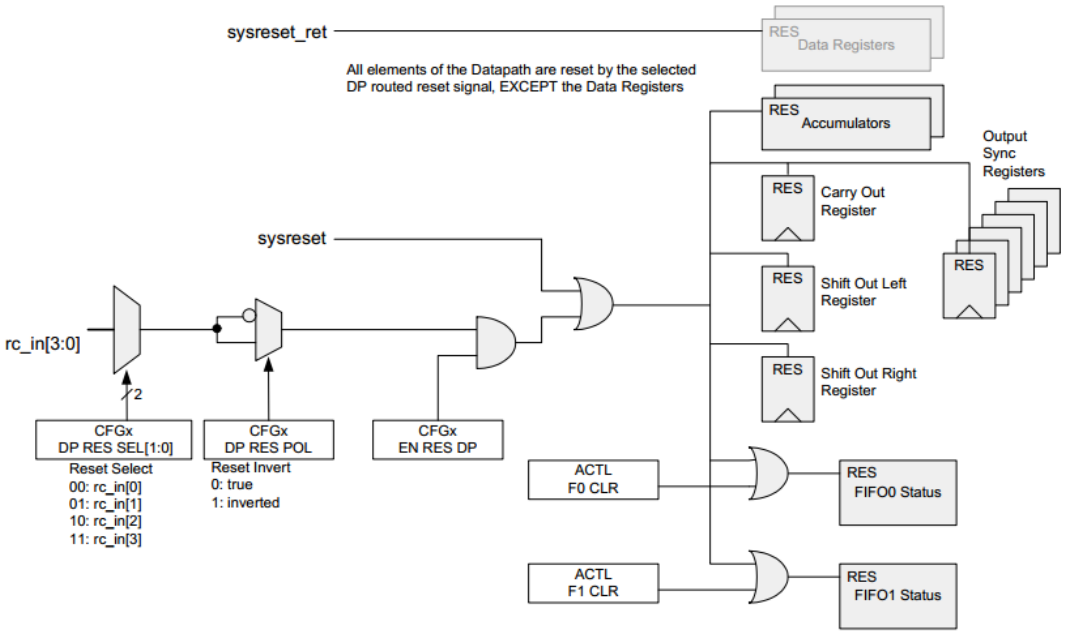
图 16-45. PLD 的分散复位模式



DP 的分散复位模式

图 16-46 给出了工作在分散模式下的 DP 复位系统。其中数据寄存器具有数据保持功能，它们不可以被互连信号复位。

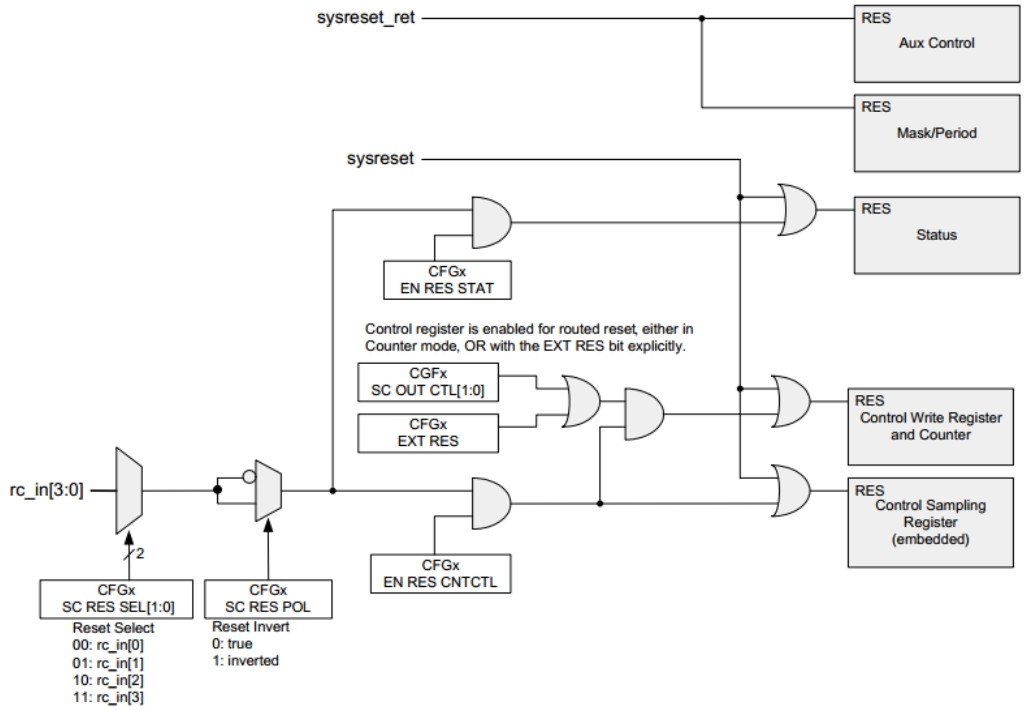
图 16-46. DP 的分散复位模式



SC 模块的分散复位模式

图 16-47 给出了工作在分散模式下的状态与控制模块的复位系统。其中其中屏蔽/周期寄存器和辅助控制寄存器具有数据保持功能具有数据保持功能，它们不可以被互连信号复位。

图 16-47. SC 模块的分散复位模式



16.2.4.3 UDB 的上电复位初始化

寄存器的初始化状态，如表 16-24

表 16-24. UDB 中寄存器的上电复位初始态

寄存器	详述	上电复位初始态
CFG 0 – 31	配置寄存器	0
Ax, Dx, CTL, ACTL, MASK	累加器，数据寄存器，控制寄存器，辅助控制寄存器和屏蔽寄存器	0
ST, MC	状态寄存器和宏单元对应的寄存器	0
DP CFG RAM & Fx (FIFOs)	动态配置寄存器和 FIFO 队列	未知
PLD RAM	PLD 的配置寄存器	未知

在上电复位中，UDB 的输入输出互连信号状态如下：

- UDB 所有的输出互连信号为“0”；
- UDB 所有的输入互连信号为“0”。

所有的初始配置的发生无先后顺序之分，可以看出，在初始化后，不会出现互连信号驱动冲突的情况。

16.2.5 UDB 中寄存器的寻址

UDB 中的寄存器 A0，A1，D0，D1，F0，F1 等能够以不同的位宽进行寻址，在系统总线上表现为不同位宽的工作寄存器，具体如下：

- 8bit 工作寄存器 – 允许访问一个 UDB 中的单个寄存器；
- 16bit 工作寄存器 –允许访问相邻的两个 UDB 中的两个同样的寄存器，如 UDBn 中的 D0 和 UDBn+1 的 D0；
- 16bit 工作寄存器 –允许访问一个 UDB 中的两个不同的寄存器，如同一个 UDB 中 A0 和 A1；
- 32bit 工作寄存器 – 允许访问相邻的四个 UDB 中的四个同样的寄存器；
- 以上三种情况都允许访问 UDB 内部的配置寄存器。

16.2.6 系统总线的访问

UDB 中的工作寄存器有两种访问模式

- 系统总线访问模式，CPU 可以读写 UDB 的工作寄存器
- UDB 内部访问模式，UDB 执行相关操作时需要读写内部的工作寄存器。

16.2.6.1 并发的系统总线访问

表 16-25 中详细列出了工作寄存器支持系统总线和 UDB 并发读/写的情况：

表 16-25. 工作寄存器的并发读写情况统计

<table>
<tr><th>寄存器</th><th>UDB 写
系统总线写</th><th>UDB 写
系统总线读</th><th>UDB 读
系统总线写</th><th>UDB 读
系统总线读</th></tr>
<tr><td>累加器 Ax</td><td rowspan="2">结果未知</td><td rowspan="2">不允许系统总线直接进行读操作 [a, b]</td><td rowspan="2">UDB 读取的是上一个操作周期的值</td><td rowspan="2">二者读取的都是当前值</td></tr>
<tr><td>数据寄存器 Dx</td></tr>
<tr><td>先入先出队列 Fx</td><td>不支持</td><td colspan="2">如果 FIFO 的状态标志位被使用，不允许 UDB 和系统总线同时读/写 FIFO</td><td>不支持</td></tr>
<tr><td>状态寄存器 ST</td><td>不支持系统总线的写操作</td><td>系统总线读取的是上一个操作周期的值</td><td colspan="2">不支持 UDB 的读操作</td></tr>
<tr><td>控制寄存器 CTL</td><td colspan="2">不支持 UDB 的写操作</td><td rowspan="5">UDB 读取的是上一个操作周期的值</td><td rowspan="6">二者读取的都是当前值</td></tr>
<tr><td>计数寄存器 CNT</td><td>结果未知</td><td>不支持系统总直接进行读操作 [c]</td></tr>
<tr><td>辅助控制寄存器 ACTL</td><td colspan="2" rowspan="3">不支持 UDB 的写操作</td></tr>
<tr><td>屏蔽寄存器 MASK</td></tr>
<tr><td>计数周期寄存器 PER</td></tr>
<tr><td>宏单元寄存器 MC (RO)</td><td>不支持系统总线的写操作</td><td>不允许系统总线直接进行读操作 [d]</td><td>不支持系统总线的写操作</td></tr>
</table>

a) 利用 FIFO 的软件捕获特性，CPU 通过系统总线可以对 Ax 进行可靠的读操作；
b) 只有通过 FIFO 才可以向数据寄存器 Dx 中写值，此时不能直接读 Dx 的值；
c) 只有计数或定时功能被禁止，计数寄存器才能被系统总线可靠的读取。如果想要动态的读计数寄存器，可以将它连接到互连信号 SC_OUT[6:0]上，详见 16.3.3.4 章节；
d) 通过互连到状态寄存器的输入，宏单元寄存器可以被系统总线可靠的读取。

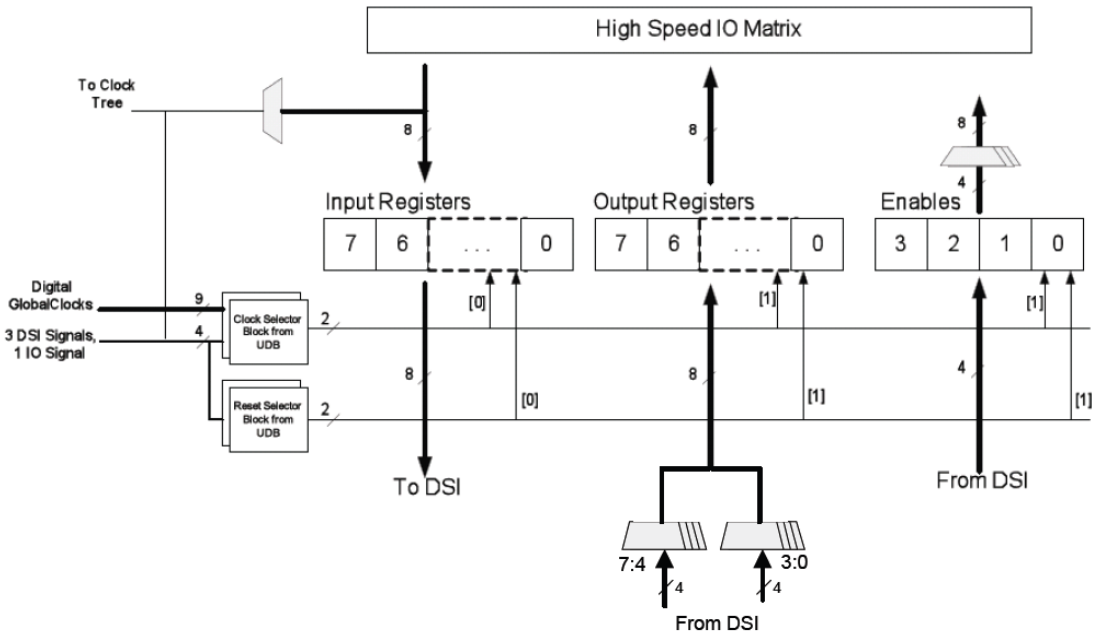
16.2.6.2 系统总线对累加器的访问

在 UDB 中，累加器是一个主要的操作数，所以系统总线直接访问累加器作会返回不可预知的结果。但是通过 FIFO 的软件捕获模式，可以对累加器进行可靠的读操作。在此模式中，对最低有效 UDB 中的累加器进行读操作会触发所有级联 UDB 将累加器中的值转移到对应的 FIFO 中，详见 16.3.2.2 章节中软件捕获模式。通过寄存器配置，可以将 FIFO 中的数据写入累加器 Ax 或者数据寄存器 Dx。

16.3 端口适配（PA：Port Adaptor）模块

数字互连通道（DSI）与高速 IO 矩阵通过端口适配（PA）模块进行通信。端口适配模块的内部结构如图 16-54 所示。端口适配模块的输入输出通过高速 I/O 矩阵中转连接到 I/O 端口管脚。高速 I/O 矩阵使得 I/O 端口可以被不同的功能模块复用。

图 16-54. 端口适配模块的框图



8bit 的 I/O 端口通过高速 I/O 矩阵和端口适配器与 DSI 通信。高速 I/O 矩阵和端口适配器之间的信号连接有 8 个 GPIO 的数据输入，8 个 GPIO 的数据输出和 8 个输出使能。

DSI 的输出信号传输到端口适配器的输出寄存器中，其中高低半字节的四个比特分别通过 4 个 4-1MUX 输出到输出寄存器的高低 4 位中。

注意 DSI 与端口适配器的 OE 信号只有四个，需要通过 8 个 4-1MUX 将它们连接到高速 I/O 矩阵。

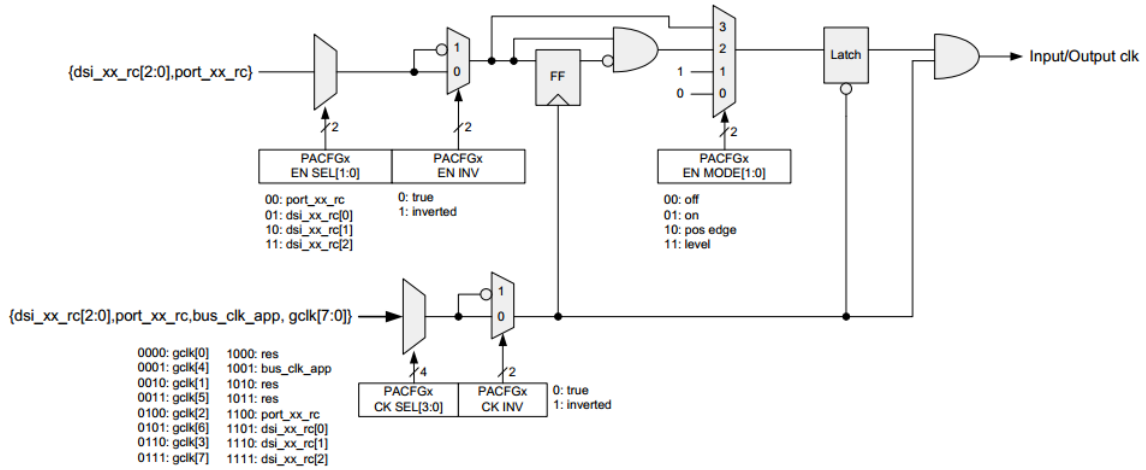
端口适配器共有两个可配置的时钟选择器和复位选择器，给出的两个时钟或复位一个提供给输入端口 ，另一个提供给输出端口和输出使能端口。

端口适配器中还有一个端口时钟多路器，通过配置它可以选择输入端口中的一个比特作为时钟源和复位源输入时钟选择器和复位选择器。

16.3.1 PA 模块的时钟多路器

端口适配器中共有两个时钟多路器，一个给输入端口提供工作时钟，一个为输出端口和输出使能端口提供工作时钟。时钟多路器的结构如图 16-55 所示。

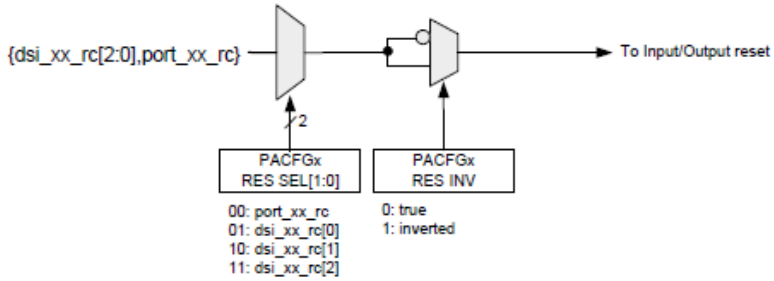
图 16-55. PA 时钟多路器的框图



16.3.2 PA 的复位多路器

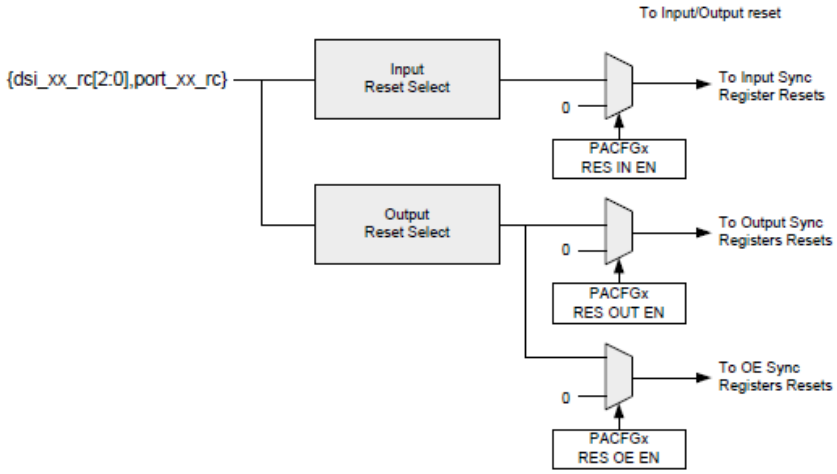
复位多路器的结构如图 16-56 所示，通过配置可以将选择的复位信号同相或反相输出。

图 16-56. PA 复位多路器的框图



如图 16-57，端口适配器中共有两个复位选择器，一个给输入端口提供复位信号，一个为输出端口和输出使能端口提供复位信号。通过配置，可以分别使能或禁止这些复位信号。

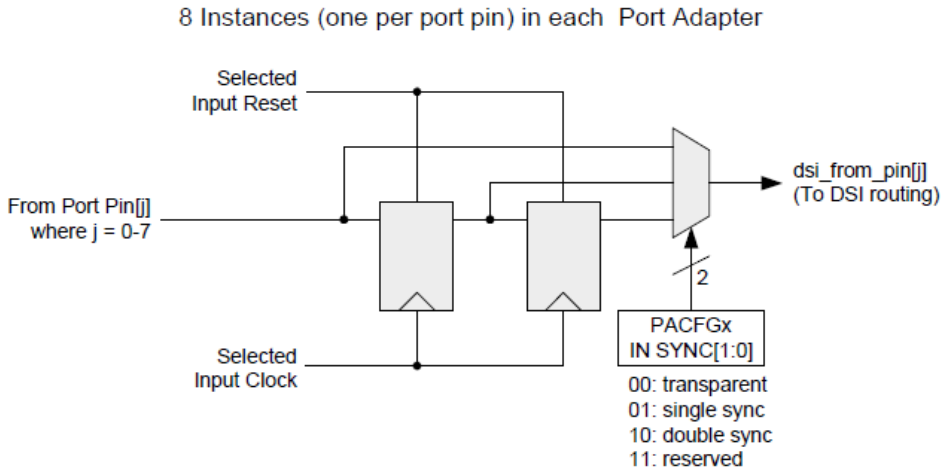
图 16-57. PA 的复位系统



16.3.3 PA 的数据输入单元

图 16-58 给出了数据输入单元的框图。输入来自高速 I/O 矩阵的各个端口。通过配置，输入信号可以选择被透传到 DSI，或者同步一次后输出到 DSI，或者同步两次后输出到 DSI。这里的同步时钟是输入端口的工作时钟。

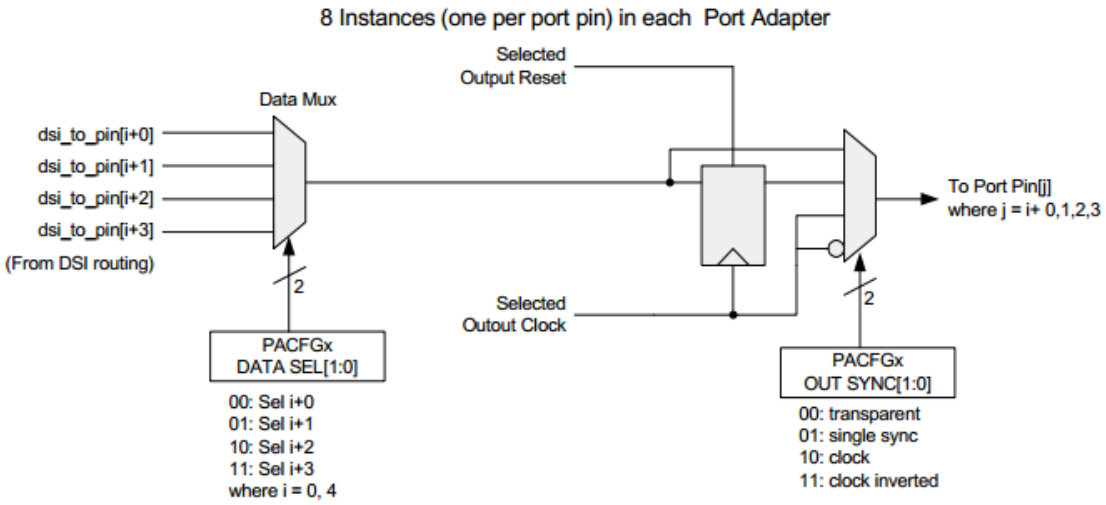
图 16-58. 数据输入单元的框图



16.3.4 PA 的数据输出单元

图 16-59 给出了数据输出单元的框图。端口适配模块的输出通过高速 I/O 矩阵传输到 I/O 端口。通过配置数据输入 MUX，可以将 DSI 输出高/低半字节中的任意一位传输到 I/O 端的高/低半字节的任意一位。通过配置数据输出 MUX，可以为输出信号提供四种选择：透传输入信号，同步后的输入信号，工作时钟和反向的工作时钟。

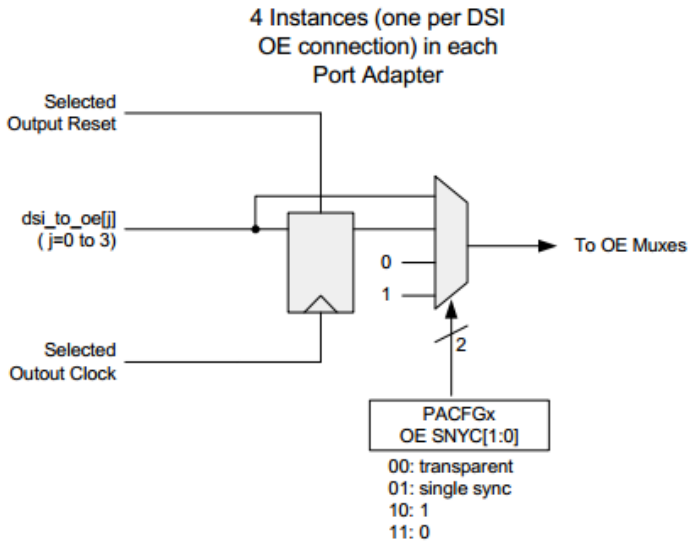
图 16-59. 数据输出单元的框图



16.3.5 PA 的输出使能单元

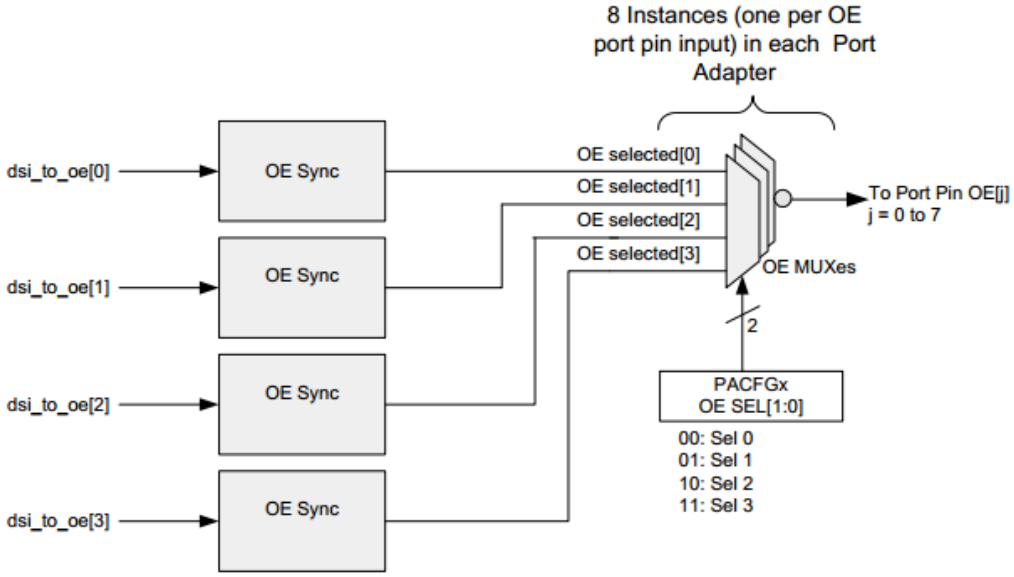
图 16-60 给出了输出使能单元的框图。这个单元与输出单元共享一个时钟和一个复位。

图 16-60. 输出使能单元的框图



与前面两个单元不同，DSI 与端口适配器的 OE 信号只有四个，所以需要通过 8 个 4-1MUX 将它们输出到与高速 I/O 矩阵的 8 个信号连接上，如图 16-61 所示。

图 16-61. 输出使能多路器



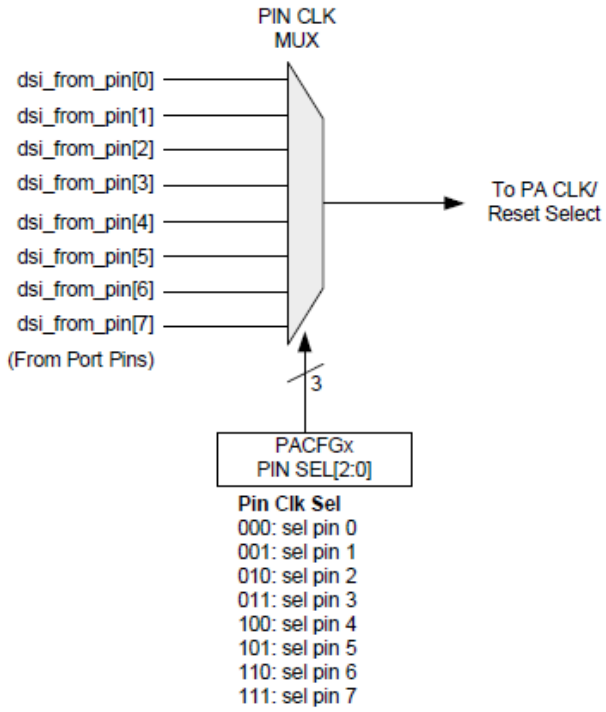
16.3.6 PA 的时钟输入多路器

图 16-62 给出了输入时钟多路器的逻辑电路结构。来自 8 个 IO 管脚的互连信号被输入一个多路器，通过多路器的选择，某一个信号可以别选择为以下的用途：

- PA 中的可编程时钟；
- UDB 的时钟；
- PA 中的可编程复位；
- PA 中的时钟使能；

注意这个被选择的信号没有经过同步，在使用时需要特别小心。

图 16-62. 输入时钟多路器



17 （Timer/Counter/PWM) 模块

通过四组专用计数器和相应的逻辑电路，TCPWM 模块能够实现 16-bit 定时器，计数器和脉冲宽度调制（PWM）等功能。该模块可以被用于测量输入信号的周期或者脉冲宽度，确定一个事件的发生时间，对电机输入的相位信号进行正交解调等。本章会介绍这个模块所有的工作模式。

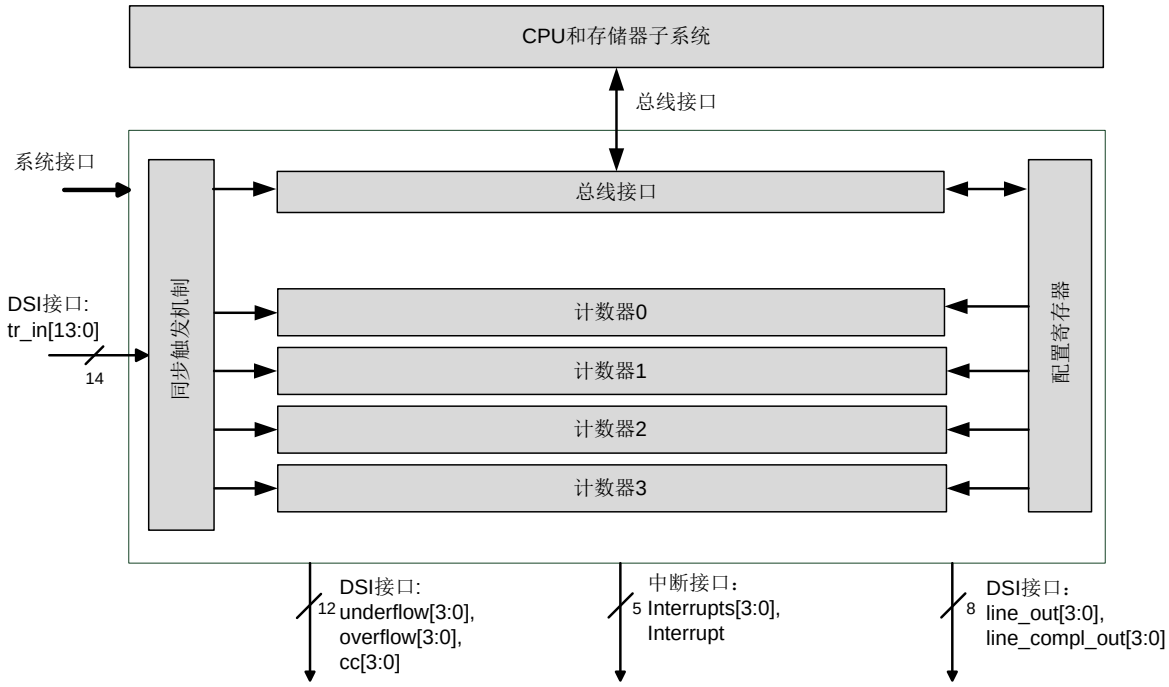
17.1 特性

- 四个 16-bit 定时器/计数器/PWM
- TCPWM 模块中的每个计数器均能够独立支持以下六种操作模式
  - 定时器/计数器
  - 捕获
  - 正交解调
  - PWM
  - 带死区的 PWM（PWM-DT）
  - 伪随机 PWM
- 四种计数模式 — 向上计数，向下计数，向上/向下计数模式 0 和向上/向下计数模式 1
- 时钟预分频选项，支持的分频系数有 2，4，…，64，128
- 比较/捕获（CC）寄存器和周期寄存器（period）有额外的缓冲寄存器
- 触发中断的方式：
  - 计数终值（TC） — 当计数器的值达到计数终值（包括 period 寄存器中的值或 0），能够触发中断
  - 比较/捕获(CC) — 当计数器的值等于 CC 寄存器中的值（定时器模式中），或者计数器的值被捕获到 CC 寄存器中（捕获模式中）时，能够触发中断
- 计数器的同步 — 四个计数器能够在同一时间发生重载，开始，停止和计数等事件
- 每组计数器的上溢,下溢,捕获/比较都可以产生 DSI 输出
- 每组计数器都有一对互补的 PWM 输出
- 14 个 DSI 输入信号，支持启动，重载，停止，计数和捕获事件，这些事件的触发条件可以使上升沿，下降沿，和电平触发

17.2 模块概述

模块框图

图 17-1. TCPWM 的模块框图



该模块有以下接口：

- 总线接口：负责模块与 CPU 和存储器子系统的通信；
- DSI 接口：负责模块与 UDB（请详见 UDB 章节）之间的通信，具体包括 14 个输入的 DSI（Digital system interconnect，数字互连信号，请详见 UDB 章节）触发信号，如重载，启动，停止，计数和捕获事件，输出的上溢信号，下溢信号和比较/捕获（CC）信号等
- 中断接口：负责传送四个计数器各自中断请求信号和这四个中断请求的逻辑或运算的结果信号
- 系统接口：负责传送系统的复位与时钟相关的信号

通过写寄存器，TCPWM 模块中的四个计数器能够被独立配置。更多寄存器信息请参考寄存器列表。

17.2.1 TCPWM 模块中计数器的使能与禁止

通过设置寄存器 TCPWM_CTRL 中的 COUNTER_ENABLED[3:0]，可以使能或禁止计数器，如表 17-1 所示。在配置计数器之前需要保证计数器被禁止，在完成配置后再使能计数器。禁止计数器并不会使计数器的值丢失，这个值会一直保持到下次计数器被使能之前。

举个例子，更新计数器 X 的计数值的典型流程：

1 禁止计数器 x
2 配置计数器 x 的计数寄存器（TCPWM_CNT）COUNTER）
3 使能计数器 x

表 17-1. 计数器的使能/禁止寄存器

COUNTER_ENABLED[x]	描述
0	禁止计数器 x
1	使能计数器 x

x = 0,1,2 或者 3

17.2.2 时钟

通过系统时钟，TCPWM 能够对事件进行同步。四个计数器分别通过各自的 counter_en（TCPWM_CTRL [3:0] ）信号对系统时钟进行门控，得到各自的 counter_clock。counter_clock 可以被预分频，分频系数包括 1, 2, 4… 64, 128，这可以通过寄存器 TCPWM_CNTx_CTRL 中的 GENERIC [10:8]来设置，具体如表 17-2 所示。

表 17-2. 计数器时钟的预分频寄存器

GENERIC[10:8]	Description 描述
0	不分频
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

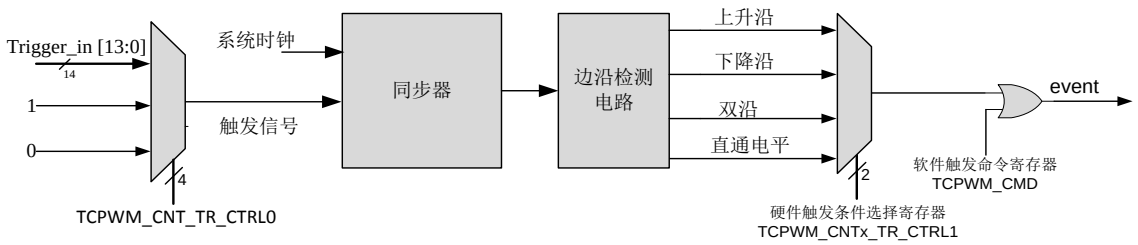
注意：在 PWM-DT 和正交解调模式中，不能够进行时钟的预分频。

17.2.3 事件的触发

- 在 TCPWM 模块中，一些事件可以通过硬件和软件的方式来触发。硬件触发的方式包括电平触发，上升沿触发，下降沿触发和双沿触发；软件触发方式则指的是操作寄存器。这些事件的具体描述如下：
  - **重载：**初始化并启动计数器
    - 在向上计数模式中，计数寄存器（TCPWM_CNTx_COUNTER）被初始化为 0；
    - 在向下计数模式中，计数寄存器被初始化为周期寄存器（TCPWM_CNTx_PERIOD）中的值；
    - 在向上向下计数模式 0/1 中，计数寄存器被初始化为 0；
    - 在正交解调模式中，重载事件被用作索引事件指示了一次旋转的完成，它被用于同步正交解调。
  - **启动：**启动计数过程，常发生在一次停止计数或者计数寄存器被重新初始化后。该事件不会对计数寄存器做初始化。
    - 在正交解调模式中，该事件复用为一个正交相位（phiB）输入的事件，正交工作模式部分会详细介绍。
  - **计数：**一次计数时间会引起计数寄存器加 1 或者减 1，这取决于选择的计数方式。
    - 在正交解调模式中，该事件复用为另一个正交相位（phiA）输入的事件。
  - **停止：**停止计数过程，计数寄存器不再随计数事件的发生而改变。
    - 在 PWM 相关模式中，停止事件被复用为一个终止事件，终止事件用于禁止 PWM 波形的输出。
  - **捕获：**当一次捕获事件被触发时，捕获寄存器（TCPWM_CNT_CC）中的值被送到缓冲捕获寄存器（TCPW_CNT_CC_BUFF）中，同时计数寄存器中的值被送到捕获寄存器中。
    - 在 PWM 相关模式中，捕获事件被复用为一个切换事件，该事件令 CC 寄存器和周期寄存器分别与它们的缓冲寄存器进行数值切换。这个特性可用于调制脉冲的宽度和频率。

PSoC4 的任何一个 GPIO 都可以作为 DSI 的输入或输出(数字互连信号，具体请见 UDB 章节)。外部信号通过 DSI 可以连接到 TCPWM 模块。其中的事件的硬件触发信号可以来自于 14 个 DSI 信号，以及一个常 1，一个常 0 信号。通过 TCPWM_CNT_TR_CTRL0 可以选择将这 16 个信号中的一个作为某一事件（重载/启动/计数/捕获/停止）的触发信号，如下图 2 所示。Remove this figure 通过配置寄存器 TCPWM_CNTx_TR_CTRL1，任何一个信号沿或者电平都可以作为某一事件的硬件触发方式。每个事件的硬件触发条件是可以单独配置的。除了硬件触发，这些事件也能够被计数器的命令寄存器（TCPWM_CMD）触发，如图 17-2。

图 17-2. TCPWM 事件触发条件的检测



注意：

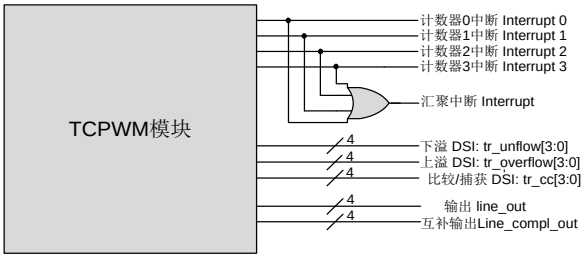
a. 所有的触发信号需要经过同步器，与系统时钟进行同步；

b. 如果在同一个计数器时钟周期中多个事件被触发，那么一个或者多个事件将会被忽视。这常发生在触发信号的频率高于计数器的时钟频率的情况下。

17.2.4 输出信号

TCPWM 模块的输出信号如图 17-3 所示

图 17-3. TCPWM 的输出信号



17.2.4.1 内部事件信号

每个计数器模块都有三个 DSI 内部事件信号输出：

- 当计数器向上计数达到周期寄存器中的值时，输出上溢（OV）信号；
- 当计数器向下计数达到“0”时，输出下溢（UN）信号；
- 当计数器发生下述两种情况时，输出 CC 信号
  - 计数寄存器的值等于比较寄存器中的值：
  - 捕获事件被触发此时 TCPWM_CNTx_COUNTER 的值被拷贝到捕获寄存器中，捕获寄存器的值被拷贝到缓冲捕获寄存器中。

17.2.4.2 中断

TCPWM 模块中的四个计数器分别有中断请求信号。在 TC 状态或者 CC 状态出现时，计数器产生中断请求，具体参考表 17-3。此外模块还有一个汇聚中断，它是上述四个中断请求信号的逻辑或运算结果。中断相关设置寄存器如表 17-3 所示。

表 17-3. 中断相关的寄存器

中断请求寄存器	比特位	名称	描述
TCPWM_CNTx _INTR	0	TC	当计数值达到计数终值时，置位，写“1”清零
	1	CC_MATCH	当计数值达到 CC 寄存器中的值，置位，写“1”清零
TCPWM_CNTX _INTR_SET	0	TC	写“1”，人为的生成一个 TC 中断请求，读取的状态反映当前中断请求寄存器的状态
	1	CC_MATCH	写“1”，人为的生成一个 CC_MATCH 中断请求，读取的状态反映当前中断请求寄存器的状态
TCPWM_CNTX _INTR_MASK	0	TC	TC 中断请求的屏蔽位
	1	CC_MATCH	CC_MATCH 中断请求的屏蔽位
TCPWM_CNTx _INTR_MASKED	0	TC	TC 中断请求寄存器位与其相应屏蔽位的逻辑与运算结果
	1	CC_MATCH	CC_MATCH 中断请求寄存器位与其相应屏蔽位的逻辑与运算结果

17.2.4.3 输出信号

TCPWM 模块中，四个计数器分别提供 2 个输出信号：line_out[3:0] and line_compl_out[3:0]。通过配置寄存器 TCPWM_CNTx_TR_CTRL2，每个计数器的 OV，UN 和 CC 状态可以分别以不同的形式来驱动各自的输出信号 line_out[x]和 line_compl_out[x]，如表 17-4 所示。

表 17-4. 在 OV, UN, CC 条件下的输出

Field	Bit	Value	Event	Description
比较/捕获(CC) 默认值 3	1:0	0	设 line_out 为‘1’	在比较/捕获（CC）模式下配置输出线
		1	清 line_out 为 ‘0’	
		2	电平翻转	
		3	不变化	
上溢(OV) 默认值 3	3:2	0	设 line_out 为‘1’	在上溢（OV）模式下配置输出线
		1	清 line_out 为 ‘0’	
		2	电平翻转	
		3	不变化	
下溢(UN) 默认值 3	5:4	0	设 line_out 为‘1’	在下溢（UN）模式下配置输出线
		1	清 line_out 为 ‘0’	
		2	电平翻转	
		3	不变化	

17.2.5 电源模式

TCPWM 模块可以在 PSoC 4 的活动和睡眠模式下工作。该模块的供电电源为 VCCD。在深度睡眠模式中，所有配置寄存器的值会被保留。TCPWM 模块在各电源模式下的工作情况见表 17-5。

表 17-5.各电源模式下 TCPWM 模块的状态

电源模式	TCPWM 模块状态
活动	支持所有的操作
睡眠	计数器时钟存在，但是总线接口不能被使用
深度睡眠	TCPWM 模块仍然被供电，但是计数器时钟被禁止，无法工作；但所有配置寄存器的状态被保留。
休眠	TCPWM 模块被彻底关闭，所有的状态均被丢失。

17.3 工作模式

TCPWM 模块中的计数器可以工作在 6 种工作模式，通过配置寄存器 MODE（TCPWM_CNTx_CTRL[26:24]）进行选择，具体如表 17-6 所示。

表 17-6. 工作模式的配置

模式	MODE Field[26:24]	描述
定时器	000	在每个发生计数事件的计数器时钟周期中，计数寄存器中的值加1或者减1
捕获	010	在每个发生计数事件的计数器时钟周期中，计数寄存器中的值加1或者减1，当捕获事件发生时，计数寄存器中的值会被拷贝到CC寄存器中，原CC寄存器中的值会送到缓冲CC寄存器中
正交解调	011	根据正交编码方法，计数器基于两个输入相位增加或减少（X1，X2或者X4）
脉冲宽度调制 PWM	100	调制输出波形的占空比与周期
带死区的脉宽调制 PWM-DT	101	与上者不同是，输出波形带死区，即在某些时刻，两个互补的 PWM 都没有输出
伪随机脉宽调制 PWM-PR	110	利用线性反馈移位寄存器产生伪随机PWM序列

TCPWM 模块中的计数器有四种计数方式，通过配置寄存器 UP_DOWN_MODE（TCPWM_CNTx_CTRL[17:16]）可以选择计数方式，具体如表 17-7。

表 17-7 计数模式的配置

计数模式	UP_DOWN_MODE[17:16]	描述
向上计数模式	00	计数寄存器的值累加，当达到周期寄存器中的值时，输出 TC 状态
向下计数模式	01	计数寄存器的值递减，当达到“0”时，输出 TC 状态
向上/向下计数模式 0	10	计数寄存器的值累加，当达到周期寄存器中的值时，计数寄存器的值递减，直到达到“0” 当计数寄存器的值达到 0 时，输出 TC 状态
UP/DOWN Counting Mode 1 向上/向下计数模式 1	11	计数寄存器的值累加，当达到周期寄存器中的值时，计数寄存器的值递减，直到达到“0” 当计数寄存器的值达到周期寄存器中的值和达到 0 时，输出 TC 状态

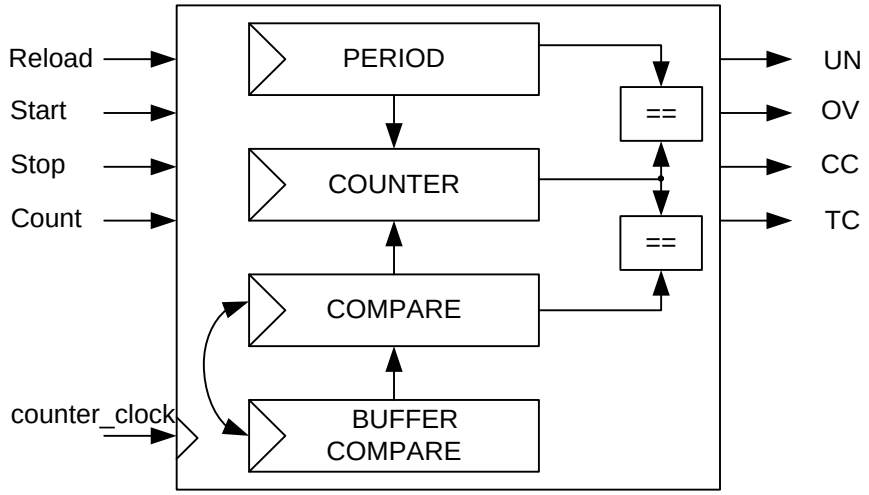
下面依次介绍各种工作模式

17.3.1 定时器模式

定时器模式通常用于测定事件发生的时间，或者通过中断请求来计算两个事件之间的时间跨度。

17.3.1.1 模块框图

图 17-4. 定时器模式的模块框图



17.3.1.2 工作原理

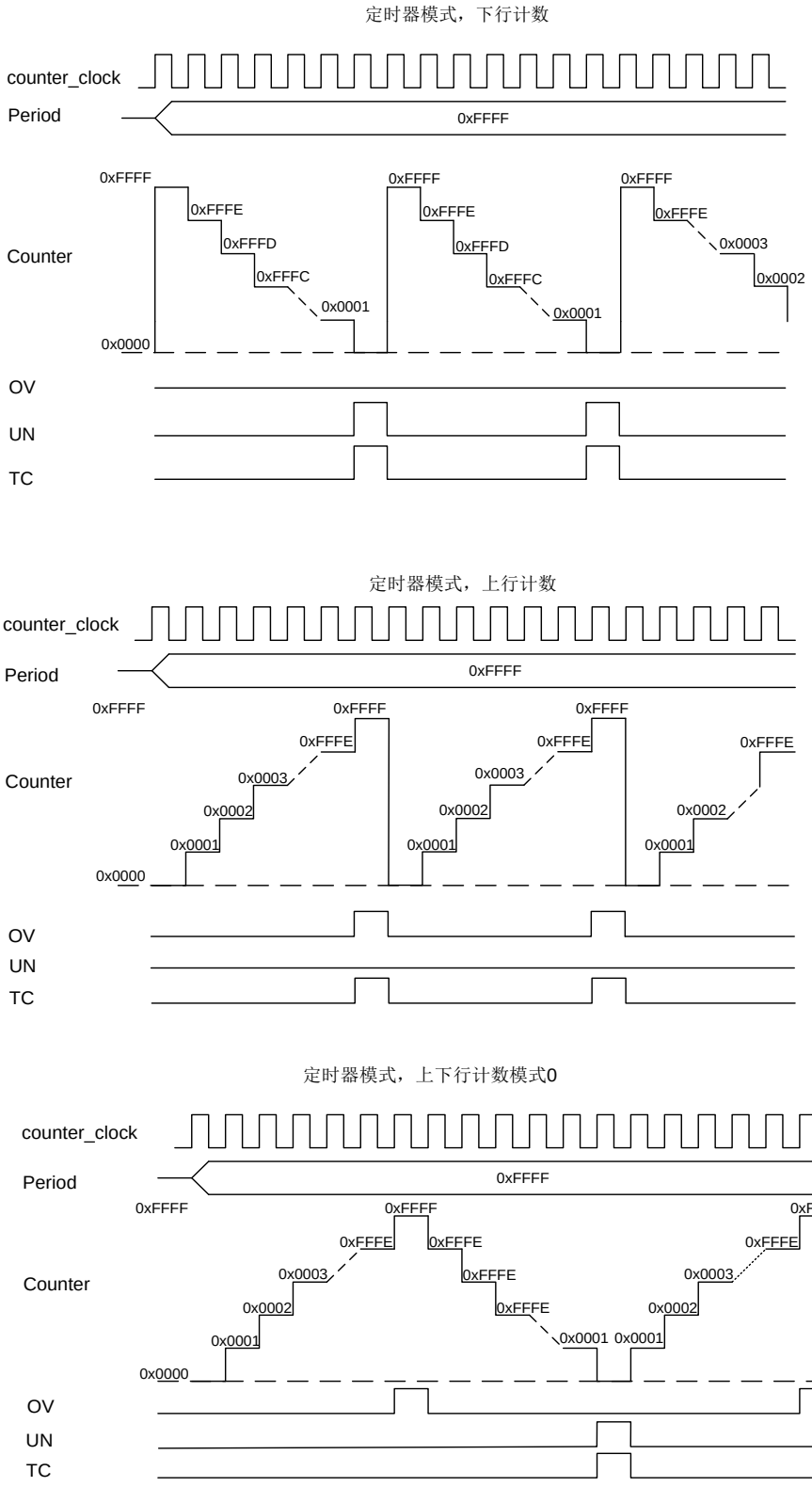
在该模式中，计数器可以工作在任何一种计数模式，它的运行方式可以是连续模式或者单次模式。

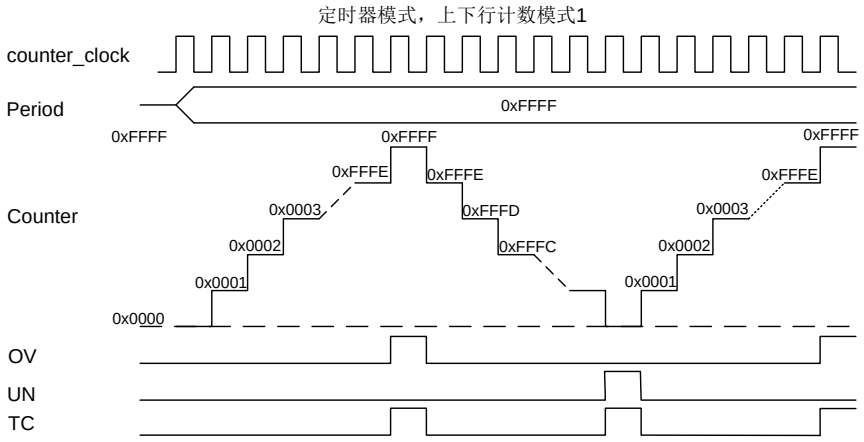
- 计数器可以向上计数，向下计数或者向上/向下计数
  - 计数器当前的计数值存放在计数寄存器（TCPWM_CNTx_COUNTER）中。**注意：**在计数器运行过程中，不要向计数寄存器中写值
  - 计数器的周期存放在周期寄存器中
- 计数器的重载
  - 在向上计数模式中，当计数值达到周期时，计数寄存器的值自动被重载为 0
  - 在向下计数模式中，当计数值达到 0 时，计数寄存器的值自动被重载为周期
  - 在向上/向下计数模式中，当计数值达到 0 或者周期时，计数寄存器的值不会被重载，但是计数器的计数方向将会被切换
- 当计数寄存器中的值与 CC 寄存器中的值相吻合时，计数器将输出 CC 状态，该状态可以用于生成中断请求。如果 AUTO_RELOAD_CC（TCPWM_CNTx_CTRL[0]）= 1，CC 寄存器的值将会在下一个 TC 事件发生时与 CC 缓冲寄存器的值交换。

图 17-5 给出了定时器模式中的计数器在四种不同的计数模式下的工作时序。假设周期为 A，那么

- 向上计数模式中，一轮计数占用 A+1 个计数时钟周期（0 到 A）；
- 向下计数模式中，一轮计数占用 A+1 个计数时钟周期（A 到 0）；
- 在两种向上/向下计数模式中，一轮计数占用 2*A 个计数时钟周期（0 到 A；A-1 到 1）

图 17-5. 定时器模式中个计数模式下的工作时序





17.3.1.3 定时器模式的配置流程

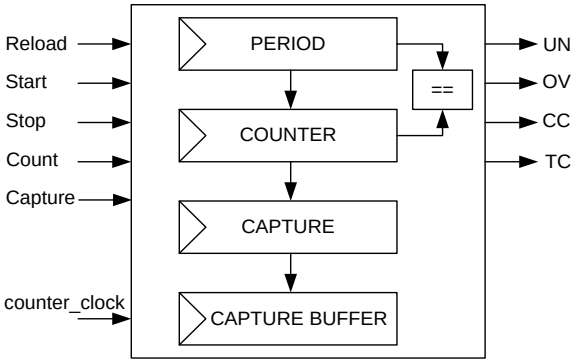
1. 将寄存器 COUNTER_ENABLED[x]（TCPWM_CTRL）清零，禁止计数器
2. 将寄存器 MODE（TCPWM_CNTx_CTRL[26:24]）清零，选择定时器模式
3. 在 TCPWM_CNTx_PERIOD 中配置计数器的计数周期
4. 配置 TCPWM_CNTx_CC 和 TCPWM_CNTx_CC_BUFF 寄存器。如果需要使能 CC 寄存器和 CC 缓冲寄存器的自动切换功能，将寄存器 AUTO_RELOAD_CC（TCPWM_CNTx_CTRL[0]）置位
5. 配置寄存器 GENRIC（TCPWM_CNTx_CTRL[10:8]），选择合适的预分频比值，详见表 17-2
6. 配置寄存器 UP_DOWN_MODE（TCPWM_CNTx_CTRL[17:16]），选择计数模式，详见表 17-7
7. 配置寄存器 ONE_SHOT（TCPWM_CNTx_CTRL[18]），选择计数器的运行方式
8. 通过 TCPWM_CNTx_TR_CTRL0 来设置触发事件信号源（如果是通过 TCPWM_CMD 进行软件触发则不需要设置）。
9. 配置寄存器 TCPWM_CNTx_TR_CTRL1，选择各个事件（包括重载，启动，停止和计数）的触发条件
10. 配置中断相关的寄存器，详见中断章节
11. 将寄存器 COUNTER_ENABLED[x]（TCPWM_CTRL）置位，使能计数器

17.3.2 捕获模式

在捕获模式中，计数寄存器的值可以被捕获到 CC 寄存器中。触发捕获事件的方式包括通过软件将寄存器 TCPWM_CMD 合适的比特位置位，和输入触发信号。该模式通常用于测量周期或者脉冲的宽度。

17.3.2.1 模块框图

图 17-6. 捕获方式的模块框图



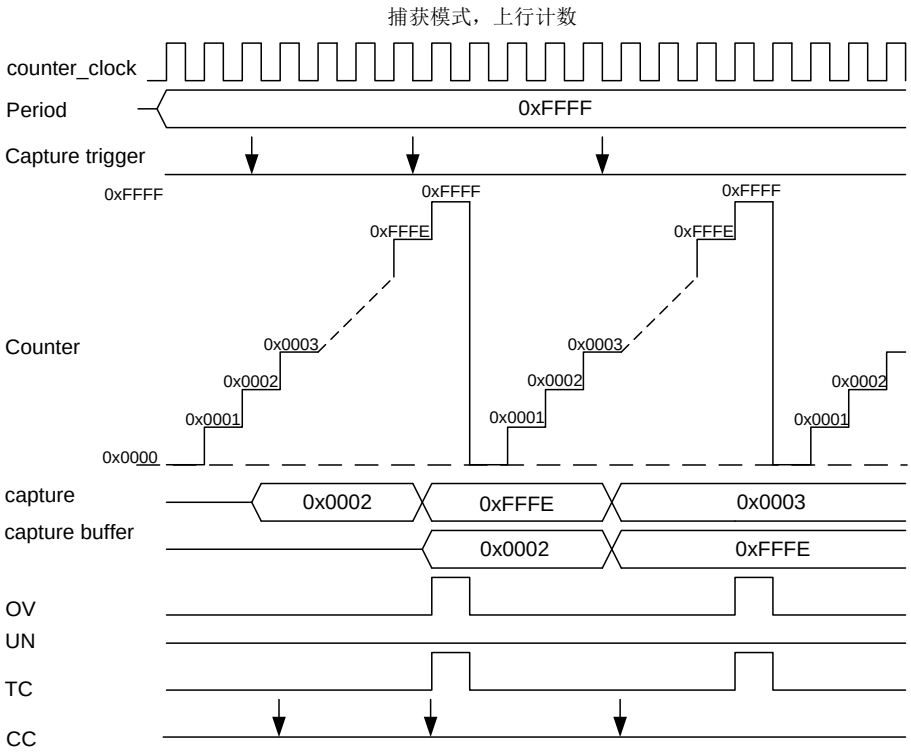
17.3.2.2 工作原理

捕获模式中的计数器可以工作在任何一种计数模式下。

- 当捕获事件发生时，计数寄存器的值被拷贝到寄存器 TCPWM_CNTx_CC，该寄存器原来的值被送到 TCPWM_CNTx_CC_BUFF。
- 当捕获事件发生时，计数器输出 CC 状态，该状态可以用于生成中断请求。

捕获模式中的计数器在向上计数模式下的工作时序如图 17-7 所示。

图 17-7. 捕获模式中向上计数模式下的工作时序



由图 7 我们可以观察到：

- 周期寄存器的值是计数寄存器能够达到的最大值
- 当计数寄存器中的值达到周期时，计数器会产生相应的（OV/ TC）事件，具体取决于计数方式
- 捕获事件只能够被信号的边沿触发
- 当捕获触发信号的频率大于计数时钟频率时，同一计数时钟周期中可能触发多次捕获事件，这时：
  - 总次数为偶数–等效于无触发事件发生
  - 总次数为奇数–等效于一次触发事件

17.3.2.3 捕获模式的配置流程

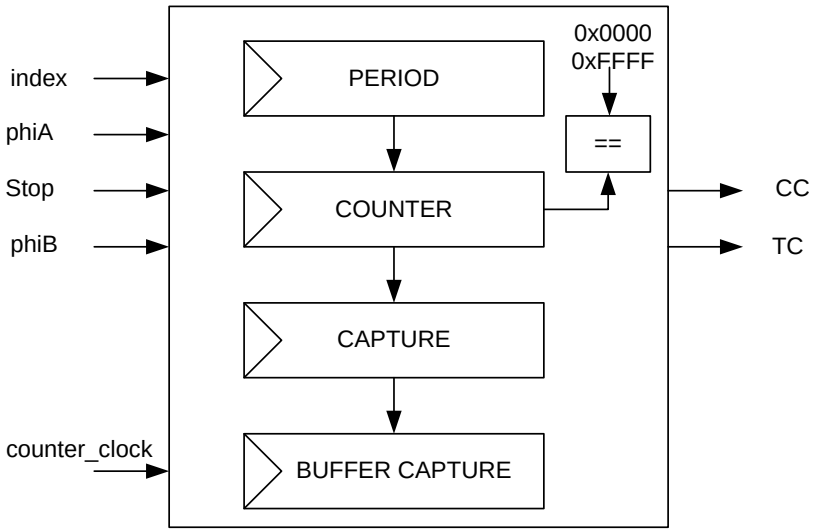
1. 将寄存器 COUNTER_ENABLED[x]（TCPWM_CTRL）清零，禁止计数器
2. 将寄存器 MODE（TCPWM_CNTx_CTRL[26:24]）设置为“010”，选择捕获模式
3. 在 TCPWM_CNTx_PERIOD 中配置计数器的计数周期
4. 配置寄存器 GENRIC（TCPWM_CNTx_CTRL[10:8]），选择合适的预分频比值，详见表 17-2
5. 配置寄存器 UP_DOWN_MODE（TCPWM_CNTx_CTRL[17:16]），选择计数模式，详见表 17-7
6. 配置寄存器 ONE_SHOT（TCPWM_CNTx_CTRL[18]），选择计数器的运行方式
7. 通过 TCPWM_CNTx_TR_CTRL0 来设置触发事件信号源（如果是通过 TCPWM_CMD 进行软件触发则不需要设置）。
8. 配置寄存器 TCPWM_CNTx_TR_CTRL1，选择各个事件（包括重载，启动，停止，捕获和计数）的触发条件
9. 配置中断相关的寄存器，详见中断章节
10. 将寄存器 COUNTER_ENABLED[x]（TCPWM_CTRL）置位，使能计数器

17.3.3 正交编码模式

正交解码器被用来确定一个旋转装置（如伺服电机，音量控制的轮子，PC 鼠标等）的速度和位置。正交编码器的信号作为输入 phiA，phiB 的给解码器。

17.3.3.1 模块框图

图 17-8. 正交解码模式模块框图



17.3.3.2 工作原理

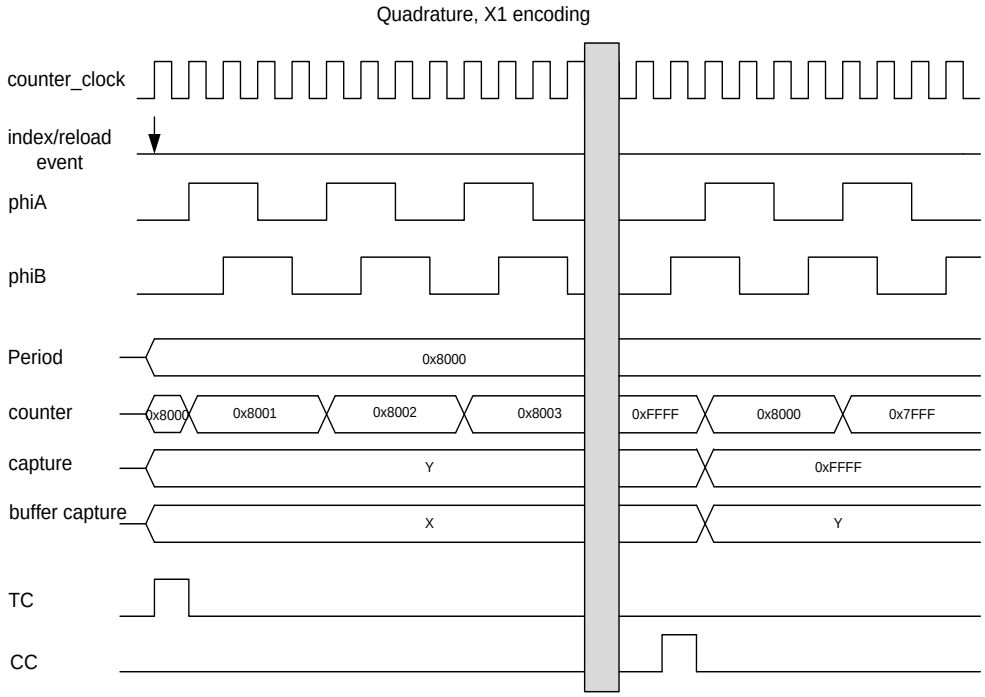
正交解码只能工作在时钟 counter_clock。可以工作在 3 种正交编码模式，即：X1，X2，X4。通过计数器控制寄存器（TCPWM_CNTx_CTRL）的 QUADRATURE_MODE 的[21:20]字段可以加以配置。

- 正交相位信号 phiA 和 phiB: 计数方向取决于 phiA 和 phiB 的相位情况，phiA 和 piB 可以分别通过计数和启动事件来输入。其中的事件的硬件触发信号可以来自于 14 个 DSI 信号，以及一个常 1，一个常 0 信号。通过 TCPWM_CNT_TR_CTRL0 可以选择将这 16 个信号中的一个做 phiA/phiB（复用计数/启动信号）的输入信号。
- DSI 输入的重载信号可以作为正交索引事件。
  如图 17-9 所示，这个事件会产生一个一个终止计数 TC 条件。

  在终止计数（TC）时，计数器被设为 0x0000（向上计数时）或者周期值（向下计数时）。

  注意：向下计数时的推荐周期值为 0x8000。
- 当计数值达到 0x0000 或者 0xFFFF 时。计数寄存器的被设置位周期值（如图 9 的例子中是 0x8000）。
- 在终止计数（TC）或者计数匹配（CC）的情况下：
  - 计数值会被拷贝到捕获寄存器中
  - 捕获寄存器被拷贝到缓存捕获寄存器中
  - 可以产生中断请求
- 从捕获寄存器中的值可以看出具体事件：
  - 计数器下溢（值为 0）
  - 计数器上溢（值为 0xFFFF）
  - 索引/终止计数事件发生 (值不为 0 或 0xFFFF)
- DOWN 比特位（TCPWM_CNTx_STATUS [0]）显示上一次计数的方向。“0”为向上计数，“1”为向下计数。图 17-9 是 X1 编码模式
  - 当 phiB 从“0”开始时，在 phiA 的上升沿计数器加 1；在 phiB 从“1”开始时，在 piA 的上升沿计数器减 1。
  - 在索引/重载事件时计数值会用周期值初始化。
  - 当计数器被索引事件初始化时，计数器终止计数（TC）。这个事件同时可以产生一个中断。
  - 当计数器值达到 0xFFFF 时（最大计数寄存器值），这个计数值可以被拷贝到捕获寄存器中（CC），同时计数寄存器值被周期值初始化 (0x8000)。

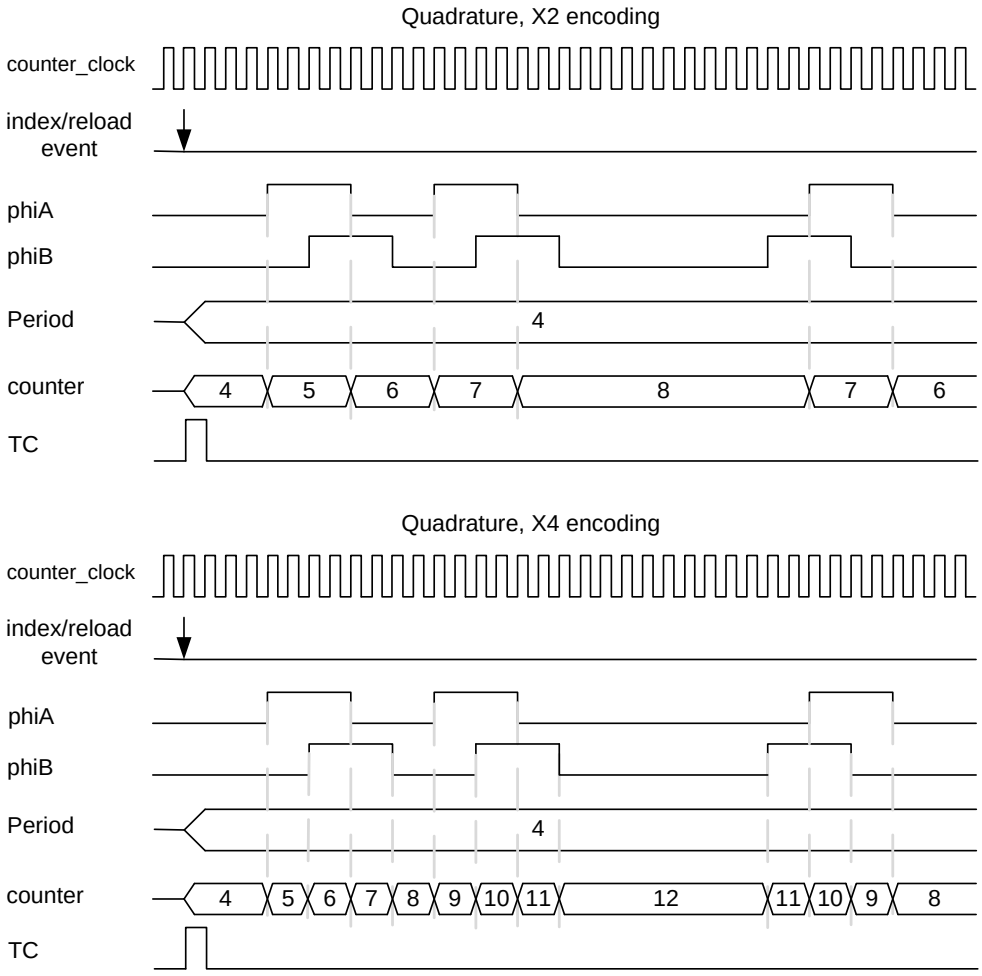
图 17-9. 正交编码 X1 模式时序图



正交相位信号时用 counter_clock 来检测的，所以在单个 counter_clock 周期正交相位信号只能改变一次。

正交编码模式 X2 和 X4 的计数速度分别是 X1 模式的两倍和四倍，具体见图 17-10。

图 17-10. 正交编码 X2 和 X4 模式时序图



17.3.3.3 正交模式配置步骤

正交编码模式的寄存器配置方法如下：

1. 将 TCPWM_CTRL 寄存器的 COUNTER_ENABLED [x]位清零，禁止相应的计数器。
2. 写“011”到 TCPWM_CNTx_CTRL 寄存器 MODE[26: 24]字段来选择正交模式。
3. TCPWM_CNTx_PERIOD 寄存器中设定 16 位的周期值。
4. 通过 TCPWM_CNTx_CTRL 的 QUADRATURE_ MODE[21:20]字段来选择正交编码模式。
5. 通过 TCPWM_CNTx_TR_CTRL0 选择具体事件(索引/piA/piB/停止)的触发信号。
6. 通过 TCPWM_CNTx_TR_CTRL1 寄存器来选择输入信号触发索引/停止事件的条件（上升沿/下降沿/双边沿）。
7. 根据实际需求，配置在 TC 或者 CC 是使能中断。
8. 通过写“1”到 TCPWM_CTRL 寄存器的 COUNTER_ENABLED[x]位使能相应的计数器。

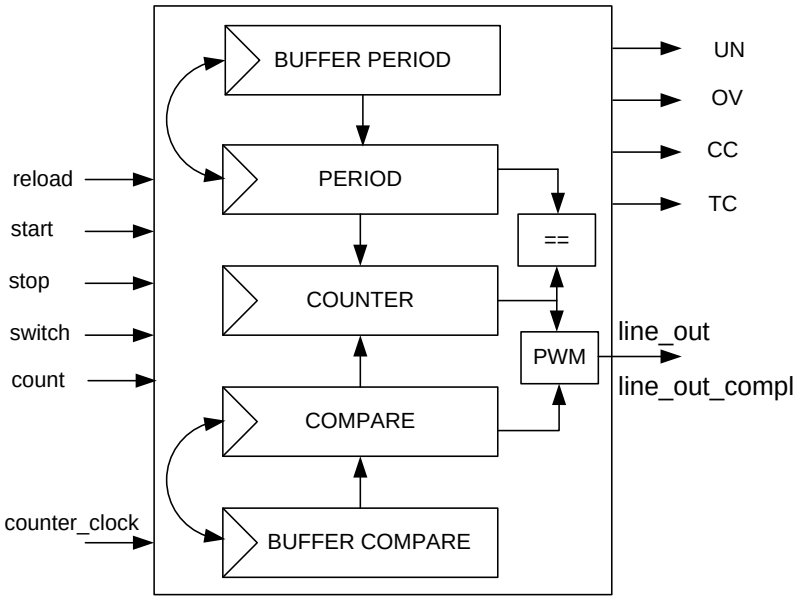
17.3.4 脉冲宽度调制（PWM）模式

PWM 模式用计数器产生的三角波和比较寄存器中的值比较，产生占空比和周期都可以变化的信号。周期依赖于周期寄存器的值；占空比依赖于比较寄存器和周期寄存器的值。

PWM Period = (Period Value *1Clock frequency)

17.3.4.1 模块框图

图 17-11. PWM 模式的模块框图



17.3.4.2 工作原理

PWM 模式可以使用计数器的三种计数模式，即：向上、向下、向上/向下。可以分别用来产生左对齐、右对齐、中间对齐的 PWM 波形。

在重载事件时，计数寄存器会被初始化，然后开始按照设置好的模式计数。每个计数值都会与比较计数器的值比较，如果匹配，PWM 输出信号就会翻转或者被置为 0/1。

PWM 输出线由 OV,UN 或者 CC 事件控制。这些事件可以将 PWM 输出电平翻转或者置为 0/1。可由 TCPWM_CNTx_TR_CTRL2 配置。

修改占空比：

- 更新缓存周期寄存器和缓存比较寄存器的值。
- 在终止计数时(TC)，如果有一个有效的交换(switch)事件并且 TCPWM_CNTx_CTRL 中的 AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD 位置 1 的话，周期和比较寄存器上的值会自动与缓存周期和比较器寄存器的值交换。如果只需要交换其中一个，则只需设置相应的 AUTO_RELOAD_CC 或 AUTO_RELOAD_PERIOD 位。
- 一个有效的交换（switch）事件，可以由硬件（DSI）或者软件触发。事件如果由硬件（DSI）作为触发信号，那么需要通过 TCPWM_CNT_TR_CTRL0[3:0] 设置 switch 事件触发信号源。如果由软件触发，可以通过写 1 到 TCPWM_CMD[7:0]实现。上升沿/下降沿/双沿/直通都可以触发交换事件。只要是边沿触发模式，无论是硬件还是软件触发，在下一次的 TC 时，触发信号都会被清零。不同的是，在直通模式且电平为高时不会清零，只要信号存在，就会一直发生交换事件。
- 有效的交换事件触发信号和更新缓存周期/比较寄存器的操作必须在下一个 TC 前完成，否则下一次将不会进行交换寄存器值，如图 17-13 所示。

图 17-12 例举了中心对齐 PWM 的波形以及各个寄存器值的变化

图 17-12. 中心对齐 PWM 时序图

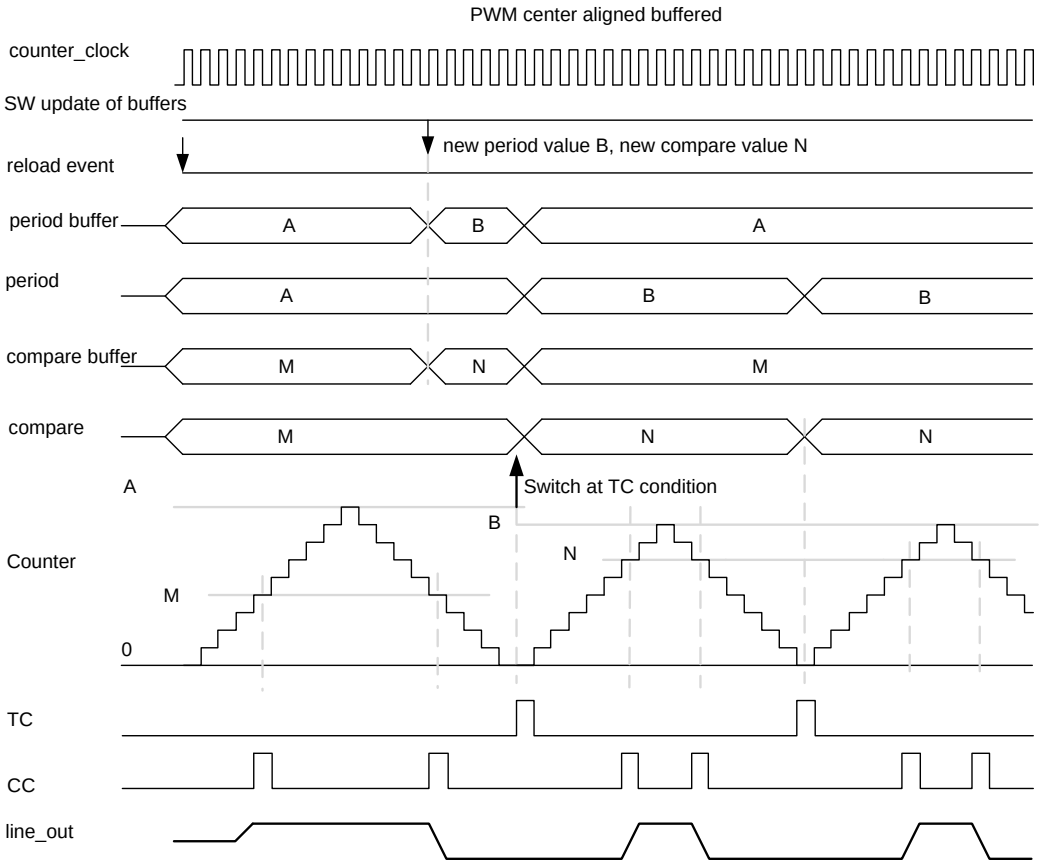
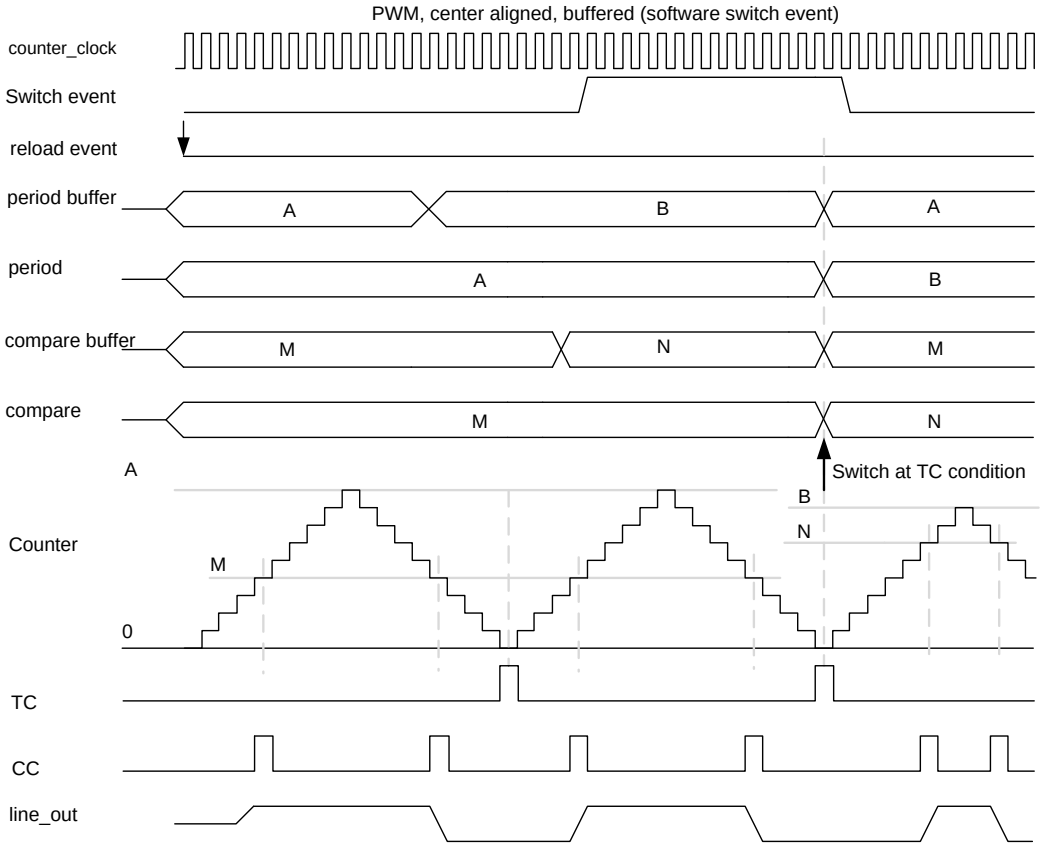


图 17-13 例举了中心对齐 PWM 以及软件产生的交换事件

- 因为交换事件在 TC 以后才发生，所以第一个 PWM 波形的占空比没有变化。
- 注意，在硬件终止计数时会自动将交换事件清零，除非是用直通模式且电平一直为高。

图 17-13. 中心对齐 PWM 的时序图(软件交换事件)



17.3.4.3 *其他配置*

- 产生非对称的 PWM 应采用向上/向下计数模式 1，TC 会发生在计数值达到 0 以及周期值时，需要将比较寄存器的值在每个 TC 时更新，周期寄存器的值会每隔一个 TC 时更新。
- 左对齐的 PWM 应用向上计数模式，在上溢（OV）的时候将输出设为高，当计数器与比较寄存器匹配时（CC）将输出设为低, 如表 17-4 所示。
- 右对齐的 PWM，应用向下计数模式，在下溢的时候将输出设为低，当计数器与比较寄存器匹配时（CC）将输出设为高，如表 17-4 所示。

17.3.4.4 *终止(Kill)*

在 PWM 模式中，停止(stop)事件被复用做终止(kill) 事件，可以将一对输出线同时禁止。通过配置 TCPWM_CNTx_CTRL 寄存器中的 PWM_STOP_ON_KILL 位，可以选择当 Kill 事件发生时，是否停止计数器计数，如表 17-8 所示。

表 17-8. 在 PWM 模式中，终止事件时配置计数器是否停止

PWM_STOP_ON_KILL (TCPWM_CNT_CTRL[3])	备注
0	Kill 停止 PWM 输出，但是计数器仍然计数
1	Kill 停止 PWM 输出，计数器停止计数

停止事件可以是同步或异步的，详细解释见下表 17-9。

表 17-9. 同步或异步的停止事件

PWM_SYNC_KILL (TCPWM_CNT_CTRL[2])	备注
0	异步停止模式会始终禁止 PWM 输出。这个事件的触发信号需要配置成直通（NO_EDGE_DET）模式。只要 Kill 电平为高，就不会输出 PWM。
1	同步停止模式会一直禁止 PWM 输出直到下一个 TC 事件。这个事件的触发信号必须配置成上升沿（RISING_EDGE）触发模式。 例外：在向上/向下计数模式 1 时，会一直禁止 PWM 输出直到计数器下一次到零时；到周期值时，虽然发生了 TC, 但是仍然会禁止 PWM 输出

17.3.4.5 PWM 模式的计数器配置

PWM 模式的寄存器配置步骤如下：

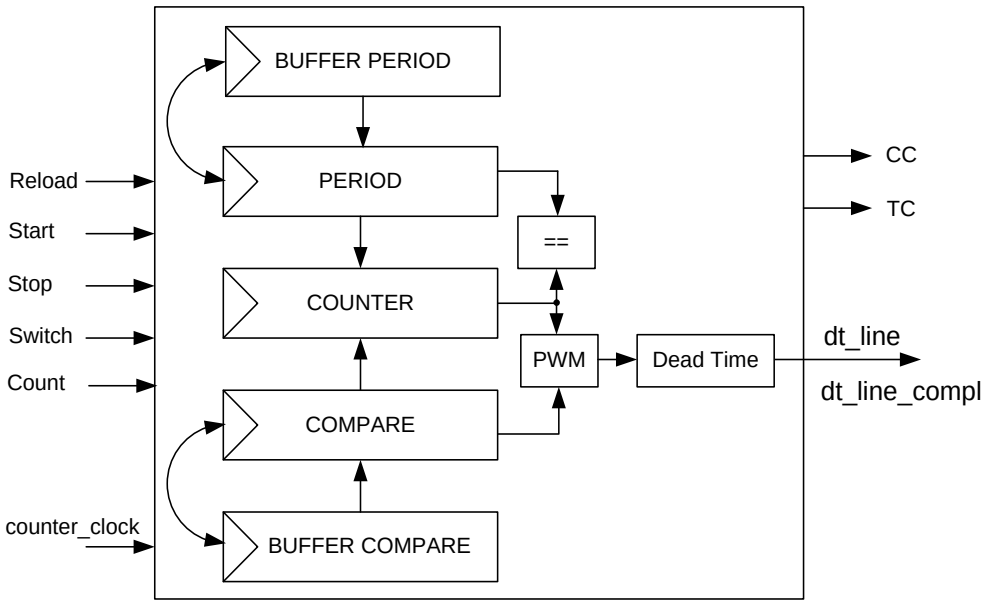
1. 通过写“0”到 TCPWM_CTRL 寄存器的 COUNTER_ENABLED[x]字段来禁止相应计数器。
2. 通过写“100”到 TCPWM_CNTx_CTRL 寄存器的 MODE[26:24]字段来选择 PWM 模式。
3. 通过 TCPWM_CNTx_CTRL 寄存器的 GENERIC 字段来选择时钟分频，如表 17-3 所示。
4. 设置 16 位的 TCPWM_CNTx_PERIOD 周期寄存器，如果需要交换值的时候，设置 TCPWM_CNTx_PERIOD_BUFF 缓存周期寄存器。
5. 设置 16 位的 TCPWM_CNTx_CC 比较值寄存器，如果需要交换值的时候，TCPWM_CNTx_CC_BUFF 缓存比较值寄存器。
6. 通过写 TCPWM_CNTx_CTRL 寄存器的 UP_DOWN_MODE[17:16]来设置计数器的左对齐、右对齐、中心对齐 PWM 模式，如表 17-7 所示。
7. 根据需求，配置 TCPWM_CNTx_CTRL 寄存器中的 PWM_STOP_ON_KILL 位和 PWM_SYNC_KILL 位。详见表 17-7 和表 17-8。
8. 通过 TCPWM_CNTx_TR_CTRL0 来设置触发事件信号源（如果是通过 TCPWM_CMD 进行软件触发则不需要设置）。
9. 通过 TCPWM_CNTx_TR_CTRL1 来设置发生事件的触发边沿条件。
10. 通过 TCPWM_CNTx_TR_CTRL2 寄存器来配置 PWM 输出信号（line_out 和 line _out_compl）在发生 CC,OV, UN 时的变化情况。
11. 根据需求，设置 TC 或 CC 情况下产生中断。
12. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED[x] 字段写“1”来使能相应的寄存器。

17.3.5 带死区模式的 PWM

在死区的时候，PWM 的两个互补输出“line_out”和“line_out_compl”都为低。死区可以防止两个 PWM 都为高的情况。死区时间最大为 255 个 TCPWM 模块时钟周期。

17.3.5.1 模块框图

图 17-14. PW-DT 模式模块框图



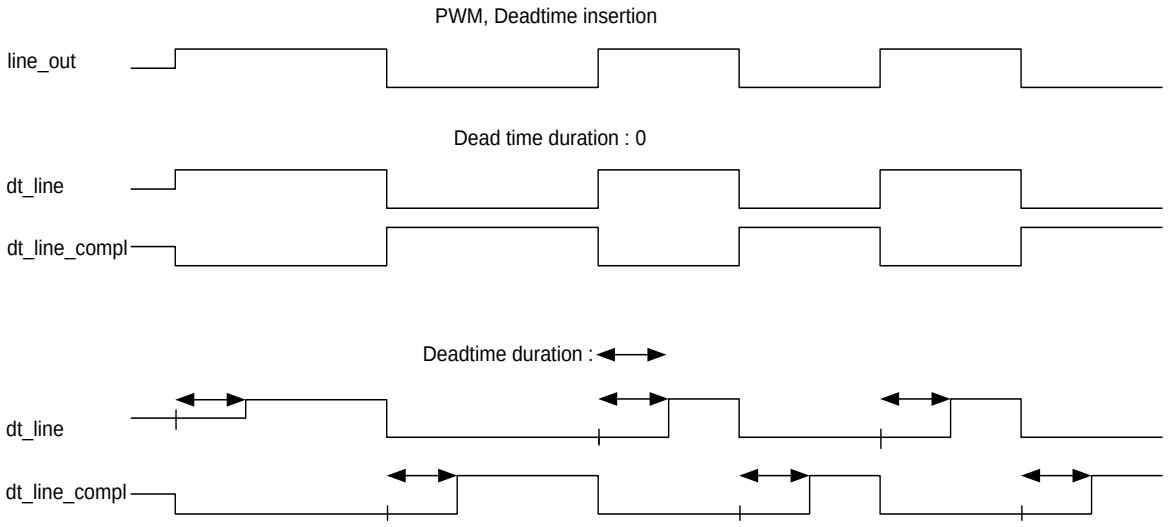
17.3.5.2 工作原理

带死区的 PWM 模式工作原理：

- 在死区时间将两路互补的输出都设为低。
- 死区时间由死区周期寄存器决定。
- 死区时间为零时，对原来的 PWM 输出没有影响。
- 当死区时间大于 PWM 脉冲的宽度时，这个脉冲就没有了。
- 除了 TCPWM 模块内的时钟分频，带死区的 PWM 支持所有 PWM 模式的功能，停止模式的设置方法也与 PWM 模式相同。在这个模式中 TCPWM_CNTx_CTRL 寄存器的 GENERIC 字段来设置死区时间。注意，TCPWM 模块中的四个计数器的主时钟（COUNTERx_CLOCK）都可以通过内部主时钟（IMO）分频后得到（详见时钟章节），所以模块内无法分频并不会影响带死区的 PWM 的实用性。

图 17-15 列举了带死区和不带死区的两个 PWM 互补输出

图 17-15. 带死区和不带死区的 PWM 时序图



17.3.5.3 配置带死区模式的 PWM

将计数器 X 配置为带死区的 PWM 模式的步骤如下：

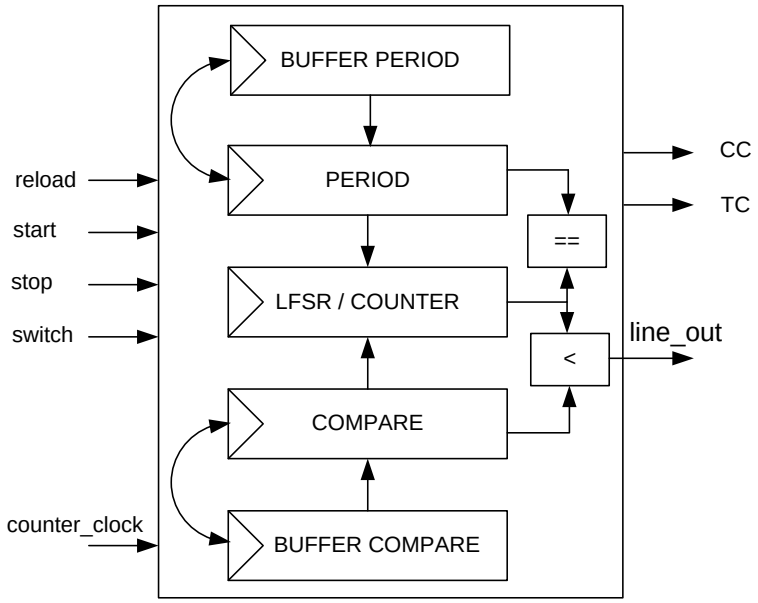
1. 通过写“0”到 TCPWM_CTRL 寄存器的 COUNTER_ENABLED[x]字段来禁止相应计数器。
2. 通过写“101”到 TCPWM_CNTx_CTRL 寄存器的 MODE[26:24]字段来选择带死区的 PWM 模式。
3. 通过 TCPWM_CNTx_CTRL 寄存器的 GENERIC[15:8]来设置死区时间，如表 17-2 所示。
4. 设置 16 位的 TCPWM_CNTx_PERIOD 周期寄存器，如果需要交换值的话，设置 TCPWM_CNTx_PERIOD_BUFF 缓存周期寄存器。
5. 设置 16 位的 TCPWM_CNTx_CC 比较寄存器，如果需要交换值的话，设置 TCPWM_CNTx_CC_BUFF 缓存周期寄存器。
6. 通过写 TCPWM_CNTx_CTRL 寄存器的 UP_DOWN_MODE[17:16]来设置计数器的左对齐、右对齐、中心对齐 PWM 模式，如表 17-7 所示。
7. 根据需求，配置 TCPWM_CNTx_CTRL 寄存器中的 PWM_STOP_ON_KILL 位和 PWM_SYNC_KILL 位。
8. 通过通过 TCPWM_CNTx_TR_CTRL0 来设置触发事件信号源（如果是通过 TCPWM_CMD 进行软件触发则不需要设置）。
9. 通过 TCPWM_CNTx_TR_CTRL1 来设置发生事件的触发边沿条件。
10. 通过 TCPWM_CNTx_TR_CTRL2 寄存器来配置 PWM 输出信号（dt_line 和 dt_line_compl）在发生 CC,OV, UN 时的变化情况。
11. 根据需求，设置 TC 或 CC 情况下产生中断。
12. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED[x] 字段写“1”来使能相应的寄存器。

17.3.6 PWM-伪随机模式

该模式用线性反馈移位寄存器（LFSR）来产生伪随机 PWM，LFSR 是一个输入为之前状态的线性方程的移位寄存器。

17.3.6.1 模块框图

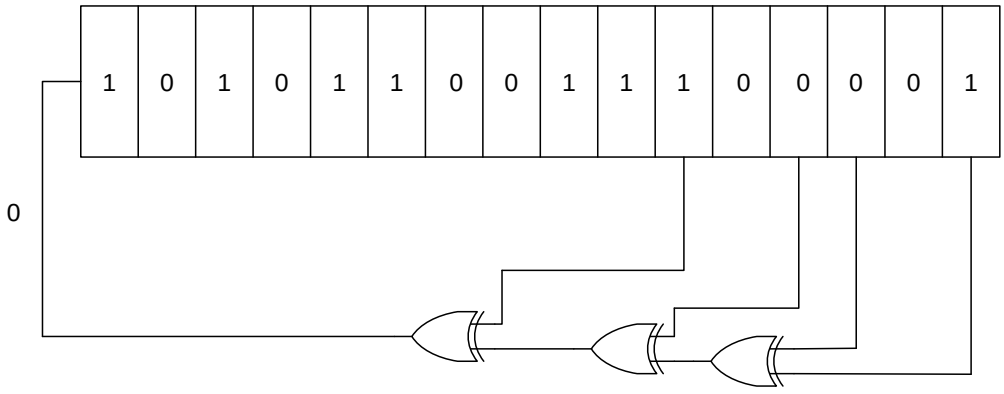
图 17-16. PWM-PR 模式模块框图



17.3.6.2 工作原理

LFSR 采用的线性方程的多项式为：$x^{16}+x^{14}+x^{13}+x^{11}+1$，可以用一个计数寄存器实现，具体方法如图 17-17 所示。它可以产生在[1, oxffff] 范围内的伪随机序列。注意，计数寄存器初始值必须是一个非零的数。

图 17-17. 用计数寄存器实现的伪随机序列

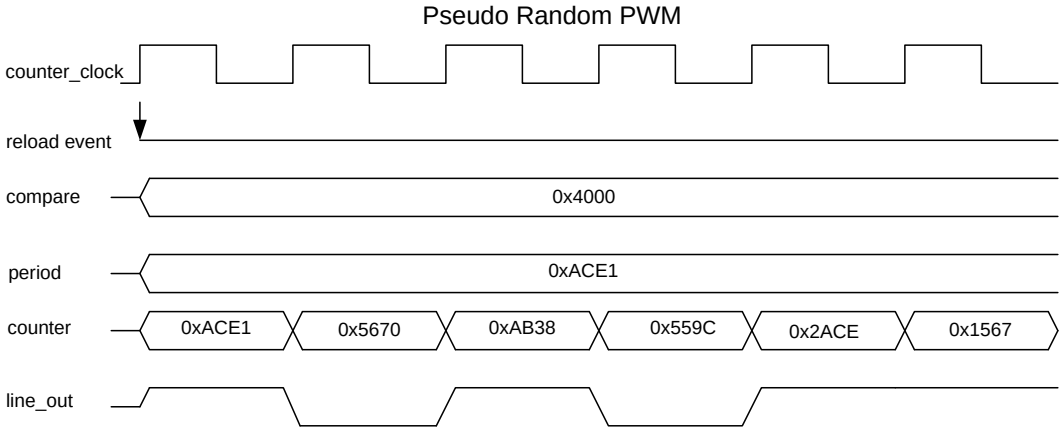


当计数寄存器的低15位小于比较寄存器的值时（counter[14:0] < compare[15:0]），PWM 输出线 line_out输出为高。当比较寄存器的值大于或等于0x8000时，PWM输出为高；当比较寄存器的值为零时，PWM 输出为低。

- 重载事件被当做开始事件，它会令计数器开始工作。但是并不会初始化计数器。
- 当计数器的值等于周期计数器的值时，会发生停止计数（TC）事件。当初始值已知时，LFSR 产生的计数器的值时可以预测的，也就是说，N 次移位后 LFSR 的计数器值时可以通过计算得到的。如果用这个值来做周期值，那么就可以精确的控制 N 次以后产生 TC 时间了。

- 在发生 TC 的时候，一个有条件的 switch/capture 事件可以交换周期寄存器和比较寄存器以及两者相对应的缓存寄存器中的值，这个条件取决于控制寄存器中 AUTO_RELOAD_CC 和 AUTO_RELOAD_PERIOD 位的值。
  - 在终止计数时(TC)，如果有一个有效的交换(switch) 事件并且 TCPWM_CNTx_CTRL 中的 AUTO_RELOAD_CC 和 AUTO_RELOAD_PERIOD 位置 1 的话，周期和比较寄存器上的值会自动与缓存周期和比较器寄存器的值更换。如果只需要交换其中一个，则只需设置相应的 AUTO_RELOAD_CC 或 AUTO_RELOAD_PERIOD 位。
  - 一个有效的交换（switch）事件，可以由硬件（DSI）或者软件触发。事件如果有由硬件（DSI）作为触发信号，那么需要通过 TCPWM_CNT_TR_CTRL0[3:0] 设置 switch 事件触发信号源。如果由软件触发，可以通过写 1 到 TCPWM_CMD[7:0]实现。
  - 上升沿/下降沿/双沿/直通都可以触发交换事件。只要是边沿触发模式，无论是硬件还是软件触发，在下一次的 TC 时，触发信号都会被清零。不同的是，在直通模式时不会清零，只要信号存在，就会一直发生交换事件。
- 支持终止（Kill）事件, 但是与 PWM 和 PWM_DT 模式不同的是，PWM_PR 模式只支持异步终止（即只要触发电平为高，就不会输出 PWM）。
- 支持单次模式（由控制寄存器中的 ONE_SHOT 位配置）：在发生 TC 时，计数器由硬件停止。
- 此模式中，没有下溢、上溢这些内部事件
- 当计数器值等于比较寄存器的值时，发生 CC 事件。图 17-18 给出了伪随机产生的噪声图：
- 因为只有第 15 位参与比较，所以用中值 0x4000，可以产生大致 50%占空比的伪随机 PWM

图 17-18. 伪随机 PWM 的时序图



17.3.6.3 *配置伪随机 PWM 模式*

将计数器 X 配置为伪随机的 PWM 模式的步骤如下：

1. 通过写”0”到 TCPWM_CTRL 寄存器的 COUNTER_ENABLED[x]字段来禁止相应计数器。
2. 通过写“110”到 TCPWM_CNTx_CTRL 寄存器的 MODE[26:24]字段来选择伪随机 PWM 模式。
3. 设置 16 位的 TCPWM_CNTx_PERIOD 周期寄存器，如果需要交换值的话，设置 TCPWM_CNTx_PERIOD_BUFF 缓存周期寄存器。
4. 设置 16 位的 TCPWM_CNTx_CC 周期寄存器，如果需要交换值的话，设置 TCPWM_CNTx_CC_BUFF 缓存周期寄存器。
5. 根据需求，配置 TCPWM_CNTx_CTRL 寄存器中的 PWM_STOP_ON_KILL 位和 PWM_SYNC_KILL 位。
6. 通过通过 TCPWM_CNTx_TR_CTRL0 来设置触发事件信号源（如果是通过 TCPWM_CMD 进行软件触发则不需要设置）。
7. 通过 TCPWM_CNTx_TR_CTRL1 来设置发生事件的触发边沿条件。
8. 通过 TCPWM_CNTx_TR_CTRL2 寄存器来配置 PWM 输出信号（dt_line 和 dt_line_compl）在发生 CC,OV, UN 时的变化情况。
9. 根据需求，设置 TC 或 CC 情况下产生中断。
10. 通过向 TCPWM_CTRL 寄存器的 COUNTER_ENABLED[x] 字段写“1”来使能相应的寄存器。

17.4 TCPWM 寄存器

表 17-10. TCPWM 模块寄存器

寄存器	备注	特性
TCPWM_CTRL	TCPWM 控制寄存器	用来使能计数器模块
TCPWM_CMD	TCPWM 命令寄存器	用来产生软件事件
TCPWM_INTR_CAUSE	TCPWM 计数器中断原因寄存器	查看中断的来源
TCPWM_CNTx_CTRL	计数器控制寄存器	配置计数器，编码方式，单次模式，交换，停止，死区，时钟分频
TCPWM_CNTx_STATUS	计数器状态寄存器	读取计数方向，死区长度，时钟分频，已经计数器的运行状态
TCPWM_CNTx_COUNTER	计数寄存器	包括 16 位的计数器值
TCPWM_CNTx_CC	比较/捕获寄存器	捕获计数器的值或者比较值
TCPWM_CNTx_CC_BUFF	比较缓存寄存器/捕获缓存寄存器	比较缓存寄存器
TCPWM_CNTx_PERIOD	周期寄存器	周期寄存器
TCPWM_CNTx_PERIOD_BUFF	周期缓存寄存器	周期缓存寄存器，其实可以由条件的和周期寄存器交换
TCPWM_CNTx_TR_CTRL0	计数器触发寄存器 0	选择触发某种事件的触发信号源
TCPWM_CNTx_TR_CTRL1	计数器触发寄存器 1	选择触发边沿条件
TCPWM_CNTx_TR_CTRL2	计数器触发寄存器 2	用来控制当发生 CC, LV, UN 事件时，输出的变化情况
TCPWM_CNTx_INTR	中断请求寄存器	TC, CC 事件发生时，产生中断请求
TCPWM_CNTx_INTR_SET	中断设置寄存器	软件设置中断
TCPWM_CNTx_INTR_MASK	中断屏蔽寄存器	中断屏蔽位，写 0 屏蔽
TCPWM_CNTx_INTR_MASKED	中断屏蔽结果寄存器	中断屏蔽结果寄存器，中断屏蔽位和中断请求位的逻辑与，若为 1，相应中断即会发生

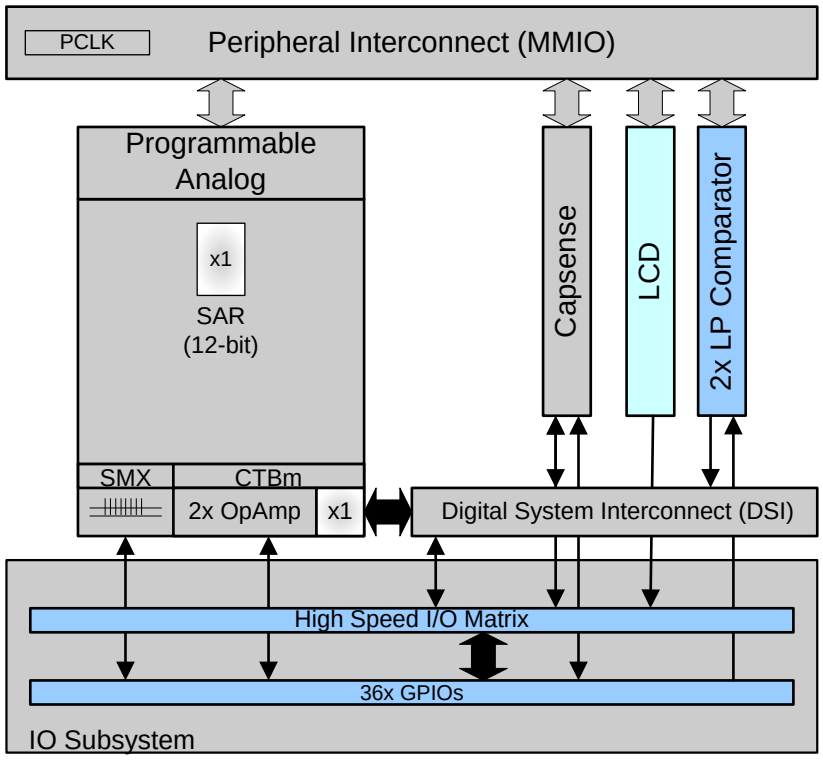
F 部分：模拟系统

这部分包括以下章节：

- 精确基准源　第 175 页
- SAR ADC 模块　第 178 页
- 低功耗比较器　第 208 页
- CTBm 模块　第 212 页
- LCD 段直接驱动　第 217 页
- CapSense 模块　第 226 页
- 温度传感器　第 233 页

系统架构：

模拟子系统结构框图



18精确基准源

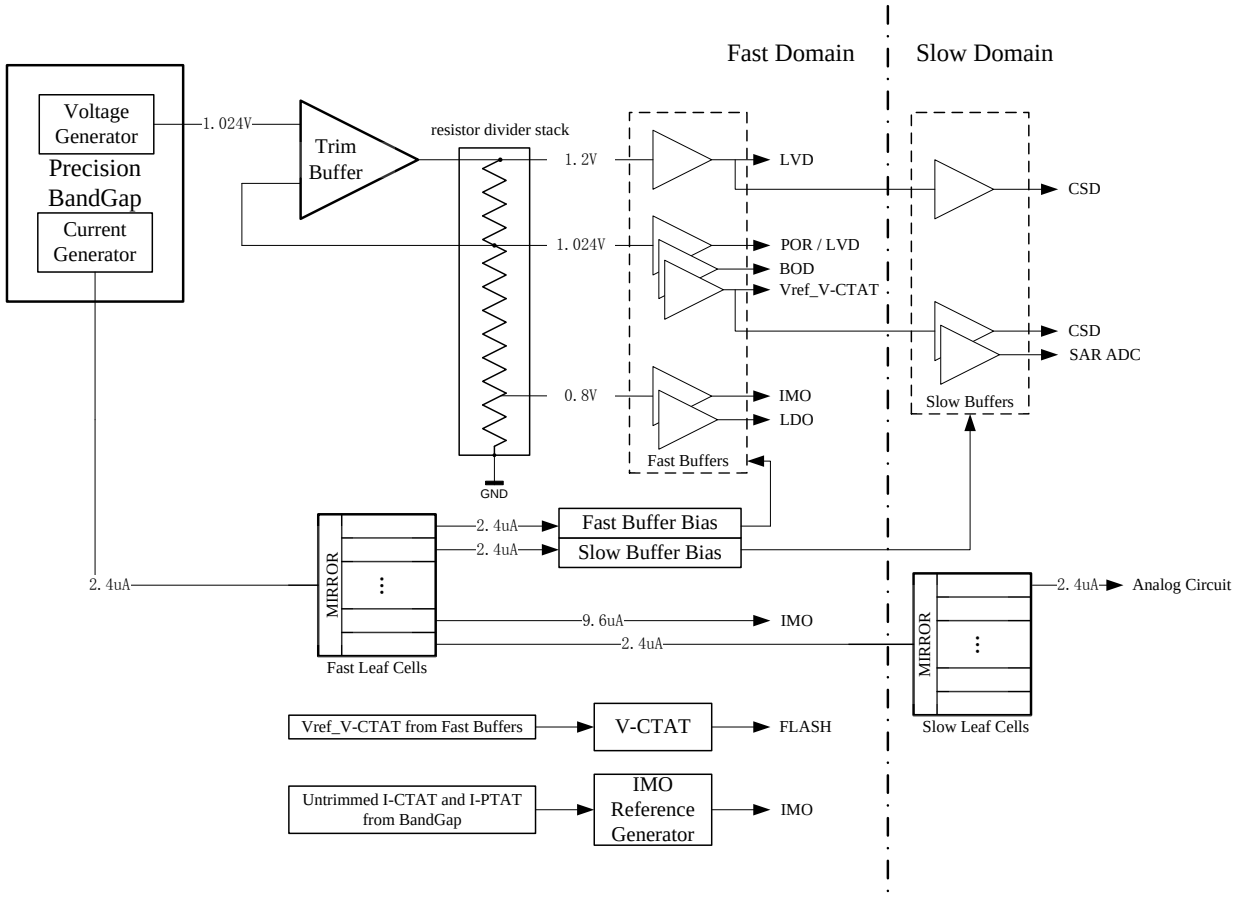
对于PSoC4内部的模拟电路， 一个不依赖于输入电源电压和温度的电压/电流基准至关重要。举例来说，一个精确的偏置电压对于很多电路来说都非常关键；在ADC中，基准电压用来量化输入电压；在一个VIDAC中，电压/电流基准定义了满幅时电压的输出范围。

18.1 模块框图

PSoC 4 有一个精确基准源模块，为整个芯片提供多个精确的电压/电流基准，模块框图如图18-1。
这个模块由以下几个部分组成

- 一个精确的能带（Bandgap）模块，可以产生精确的电压和电流基准。
- 一个校正缓冲器，用于为不同的应用产生不同的输出电压基准，并将电压校正到1.024V。
- 一组快速低功耗缓冲器，一组慢速低功耗缓冲器，分组不仅可以增强基准源的驱动能力，也可以隔离相互间的噪声。
- 一组快速电流单元和一组慢速电流单元，可以分别在快速域和慢速域复制出多个电流基准源。
- 一个V-CTAT模块，为闪存提供一个温度相关的基准电压。
- 一个根据温度可调的电流源，可以为 IMO 产生不受温度影响的电流基准。

图 18-1. 参考源模块框图



18.2 工作原理

上面介绍了模块框图，下面将详细介绍各个模块的工作原理。

18.2.1 精确带隙电路 （Bandgap）

带隙（Bandgap）电路的主要结构是两组运行在不同的发射极电流密度的以二极管形式连接的双极结型晶体管（BJT）。通过一个与温度成正相关（PTAT）的电路来抵消PN接的温度负相关性(CTAT)，来产生一个直流电压基本不随温度变化的基准电压源。基准电流源也用同样的方法产生。

18.2.2 矫正缓冲器

校正缓冲器用来产生1.024V的基准电压和不同的输出电压值的基准。

18.2.3 低功耗缓冲

作为基准源，调整缓冲器输出阻抗过高，需要通过低功耗的缓冲器将每个输出进行阻抗变换后输出后每个模块，并且可以隔离各个基准源。

这些低功耗的缓冲器分为快速缓冲区和慢速缓冲区，分别在芯片的快速域和慢速域。分成两个域的原因是，如果所有的基准源一起启动，由于负载电容过大，建立时间就不可能很快。快速域的模块需要基准源建立的事件比较快（如Flash, 电压监测器和电源系统），以确保系统成功启动， 这部分的基准源由快速缓冲区提供。

相对来说，慢速域模块的基准源建立可以比较慢，比如SAR, CSD等，这部分的基准源由慢速缓冲区提供。

18.2.4 电流单元

除了能带（Bandgap）基准电压源，芯片还提供了二阶曲率校正2.4uA电流源。电流基准经过电流单元产生多个电流基准。

电流基准同样分为快速域和慢速域，如模块框图所示，2.4uA的电流经过快速电流镜后分为四路，两路2.4uA为缓冲偏置模块输入，通过快速与慢速的缓冲偏置模块，分别为快速缓冲器和慢速缓冲器提供电压偏置；一路9.6uA为IMO的电流基准，一路2.4uA再经过慢速电流镜后为模拟电路提供电流基准。

快速域的电压和电流基准的建立时间为9uS，慢速域的电压/电流基准建立时间为90uS（即达到与终值误差在1%内所需的时间）。所有的电压和电流基准如表18-1所示。

表 18-1. 电压和电流基准

电压或电流基准	精度	使用该参考的模块
1.2 V	±2%	低压检测模块
1.2 V	±2%	CapSense 基准电压
1.024 V	±1%	SAR ADC 基准电压
1.024 V	±2%	闪存 – 设置VLIM电压
1.024 V	±2%	BOD – 检测内部欠压
1.024 V	±2%	CapSense 基准电压
0.8 V	±2%	IMO - 弛缓振荡器的比较门限
0.8 V	±2%	LDO- VCCD和VCCA调压器基准
2.4 uA	±2.5%	模拟电路偏置电流
3 uA	±2.5%	闪存宏单元中的电流基准IREF
9.6 uA	±5%	可编程温度系数的IMO的电流基准（IREF）

18.2.5 V-CTAT 模块

擦除的闪存的操作发生在较高的温度下更快。为了保持闪存的可靠性，这种过度的擦除条件必须加以避免。 V-CTAT模块生成一个与温度负相关 (CTAT) 电压基准，避免过擦除。
相对于55ºC时输出电压，-40ºC至150ºC的基准输出电压的线性变化范围是+ / -0%+/ -15%。

18.2.6 IMO 参考源电路

这个模块用来为内部主振荡器的块（IMO）提供一个独立的可调整的基准电流。利用Bandgap电路中互相抵消的温度正相关 (PTAT) 和温度负相关 (CTAT) 电流来实现低温漂以及在整个工作温度范围内±2%的精度， 如图18-1 所示。

18.3 配置

在上电期间，精密基准模块会用默认校正设置初始化， 这些设置保存在非易失性锁存器（NVL）和SFLASH中。这些设置会在芯片的生产过程中写入，客户不需要调整。

19 SAR ADC 模块

PSoC® 4 包含了一个逐次逼近型模数转换器（SAR ADC）模块，其可满足中等分辨率、快速转换的应用需求。SAR ADC 模块由 4 部分组成（见图 19-1）：专用多路输入选择（SARMUX）模块、SARADC 模块、参考电压（SARREF）模块和扫描序列控制（SARSEQ）模块。

注意：在本章节中，SAR ADC 指 PSoC 4 的 SAR ADC 模块；SARADC 指 SAR ADC 模块中的 ADC（见图 19-1）。

SARADC 是一个 12 位、采样速率可达 1 Msps 的 ADC，能实现快速、逐次逼近转换的功能；SARMUX 位于 SARADC 的前面，可以将外部管脚信号或来自 AMUXBUS_A/ AMUXBUS_B、CTBm 及温度传感器输出的内部信号传送给 SARADC 的 8 个内部通道；SARREF 可以为 SARADC 提供多种参考电压选择；序列控制器 SARSEQ 控制 SARMUX 和 SARADC，可在无 CPU 参与的情况下实现对所有使能通道的自动扫描，并可对输出信号进行预处理（比如求平均等）。

SARADC 的第 9 个通道是一个由软件控制的插入通道，可以对输入信号进行间隙采样，例如对温度传感器输出信号的采样。

SARSEQ 为每个通道的转换结果提供了双缓冲保存；通过配置，SAR ADC 在每次完成对所有通道的扫描后，会产生一个中断信号。转换后的数据既可被 CPU 处理，也可在无 CPU 参与的情况下被发送到 UDB 做进一步的处理。序列发生器（Sequencer）（见图 19-1）可记录溢出、冲突和饱和等错误标志；用户配置相应寄存器后，当错误标志产生时可产生相应的中断信号。

除了通过软件配置寄存器外，用户也可通过 UDB 来控制大部分模拟开关（包括 SARMUX 中的模拟开关），来实现对信号的采样和转换。因此，序列发生器（Seguencer）也可通过 UDB 来控制。

19.1 特性

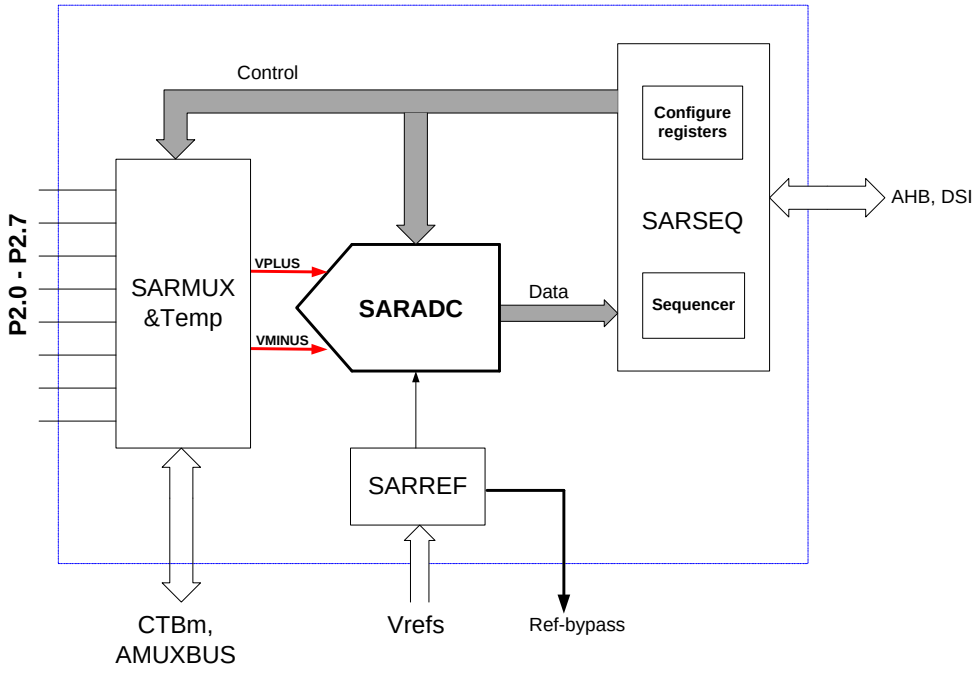
- 宽电压范围：1.71V-5.5V
- 高达 1 Msps 的采样速率
- 8 个可配置通道和 1 个插入通道
- 每个通道具有如下功能：
  - 输入信号来自外部管脚或 AMUXBUS/CTBm/温度传感器的内部信号
  - 4 个不同采样时间
  - 主分辨率：12 位 分辨率（默认）；次分辨率：8 位/10 位
  - 单端模式或差分模式
  - 求平均
  - 转换结果双缓冲保存
  - 转换结果左对齐或右对齐
- 通道扫描可被软件、定时器、管脚或来自 UDB 的信号触发
  - 单次、周期 或 连续模式
- 支持硬件求平均
  - 转换结果累加
  - 支持对 2^n（1≤n≤8）个采样信号求平均
  - 将转换结果进行符号扩展至 16 位后保存
- 可选的参考电压
  - 内部参考电压 Vdda 或 Vdda/2

- 带有缓冲的内部参考电压 1.024V
- 外部参考电压

- 中断
  - 扫描转换完成中断
  - 通道饱和检测中断
  - 通道阈值检测中断
  - 扫描结果溢出中断
  - 冲突检测中断
- 可配置的插入通道
  - 软件触发
  - 支持扫描追尾
  - 可配置采样时间、分辨率、单端/差分输入模式和求平均
- 转换结果可送至 UDB 作进一步处理，而不需要 CPU 参与
- 来自 UDB 的信号可控制 SARMUX
- 来自 UDB 的信号可控制 SARADC
  - 实现替代的序列发生器（sequencer）
- 低功耗模式
  - 可分别设置 SARADC 和参考电压的低功耗模式

19.2 SAR ADC 模块框图

图 19-1. SAR ADC 模块框图



19.3 SAR ADC 模块的使用

本章节主要内容如下：

- 介绍各个模块：SARADC、SARMUX、SARREF 和 SARSEQ
- SAR ADC 系统资源：中断、低功耗模式和 SAR ADC 状态
- 系统操作模式：
  - 寄存器模式
  - DSI 模式
- 配置示例

19.3.1 SARADC 核

PSoC 4 SARADC 是一个 12 位、逐次逼近型的 ADC。PSoC 4200 器件在 SAR ADC 时钟为 18MHz 时，其采样速率可达 1Msps；PSoC 4100 器件在 SAR ADC 时钟为 14.5MHz 时，其采样速率可达 806ksps。

主要特性：

- 全差分架构，也支持单端模式
- 主分辨率：12 位分辨率；次分辨率：8 位/10 位
- 可配置的采样时间
- 可配置的低功耗模式（正常，1/2，1/4）
- 支持单次和连续转换模式

19.3.1.1 单端模式和差分模式

PSoC 4 SAR ADC 的工作模式包括单端模式和差分模式，用户可通过寄存器 SAR_CHANx_CONFIG 来配置。SAR ADC 采用了优化的全差分结构，在差分模式下可提供完整的 12 位精度。即当差分输入范围为 –Vref ~ +Vref 时，SAR ADC 可以满量程输出（0 ~ 4095）。用户可通过固定负端输入信号并配置寄存器 SAR_CHANx_CONFIG，将 SAR ADC 作为单端模式来使用。

当 SAR ADC 被配置为单端模式时，用户可通过配置全局配置寄存器 SAR_CTRL 来选择 6 种不同的负端输入：VSSA、VREF、P2.1、P2.3、P2.5 或 P2.7。当负端连接到 P2.1/P2.3/P2.5/P2.7 时，此时的单端模式相当于差分模式。需要注意的是当对温度传感器信号采样时，只能使用单端模式，即 SAR_CTRL[11:9]会被设置为 0；如使用差分模式，转换结果不正确

19.3.1.2 输入范围

所有的输入信号的电压均应在 VSSA～VDDA 范围内。同时，输入电压范围也受限于参考电压 Vref。如果负端输入电压为 Vn，ADC 的参考电压为 Vref 时，则正端输入电压范围为 Vn+ / - Vref。单端模式和差分模式均遵循此标准。

需要注意的是，Vn+ / - Vref 应在的 VSSA~ VDDA 范围之内。例如，如果负端连接到 VSSA，则正端电压范围是 0 ~ Vref，而不是-Vref ~ Vref。因为该信号不能低于 VSSA。在这种情况下，ADC 只有一半的范围是可用的，因为正端输入电压不能低于 Vss，所以转换结果只有 11 位有效值。这是一个典型的例子，在单端模式下的负端输入为 VSSA。

19.3.1.3 转换结果数据格式

用户可从如下两个方面配置转换结果数据格式：

- 有符号/无符号
- 左对齐/右对齐

当转换结果是有符号的数据时，最高有效位为符号位，并将其扩展使结果为 16 位；当转换结果是无符号的数据时，将用 0 扩展转换结果至 16 位。用户可通过寄存器 SAR_SAMPLE_CTRL[3:2]分别为差分模式和单端模式转换进行配置。

采样值保存在 16 位寄存器中。在默认情况下，数据是右对齐的，即保存在低 12 位中，其余用符号位或 0 填充。但采用低分辨率、左对齐时，转换结果的低有效位将采用 0 填充。

12 位 /10 位 /8 位分辨率的转换结果在寄存器中的存储格式见下表 19-1。

表 19-1. 转换结果数据在寄存器中的存储格式

Alignment	Signed / Unsigned	Resolution	Result register															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Right	Unsigned	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
Right	Signed	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
Left	-	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	-	-	-	-	-	-	-	-

19.3.1.4 负端输入选择

不同的 SARADC 负端连接选择，会影响到电压范围、信噪比（SNR）、有效分辨率，详情见表 19-2。在单端模式下，SARADC 的负端输入可连接至 VSSA、VREF、P2.1、P2.3、P2.5 或 P2.7

表 19-2. SARADC 的负端连接的不同选择

单端/差分模式	有符号/无符号	负端连接	正端输入电压范围	结果寄存器	最大 SNR
单端	N/A*	VSSA	+Vref VSSA=0	0x7FF 0x000	较好
单端	无符号	Vref	+2*Vref Vref VSSA=0	0xFFF 0x800 0	好
单端	有符号	Vref	+2*Vref Vref VSSA=0	0x7FF 0x000 0x800	好
单端	无符号	Vx	Vx+Vref Vx Vx-Vref	0xFFF 0x800 0	最好
单端	有符号	Vx	Vx+Vref Vx Vx-Vref	0x7FF 0x000 0x800	最好
差分	无符号	Vx	Vx+Vref Vx Vx-Vref	0xFFF 0x800 0	最好
差分	有符号	Vx	Vx+Vref Vx Vx-Vref	0x7FF 0x000 0x800	最好

*在单端模式下，当 SARADC 负端连接到 VSSA 时，因 PSoC4 管脚上信号低于 VSSA 时无效，所以转换结果仅 11 位有效。在此情况下，SINGLE_ENDED_SIGNED 位（SAR_SAMPLE_CTRL[2]）不起作用，也就是无论选择有符号还是无符号，结果相同（范围：0x000~0x7FF）。

若在单端转换下需要 12 位的数据，用户需要将参考电压 VREF 连接到 SARADC 的负端；此时转换结果范围是 0~2*Vref。

注意：在单端模式下，当 SARADC 负端连接到 P2.1/P2.3/P2.5/P2.7 时等效于差分模式。但是，当差分对的奇数管脚连接到共用地时，转换结果仍然仅有 11 位有效位，因为被测信号不能低于 VSSA。

19.3.1.5 分辨率

PSoC 4 SARADC 的每个通道均支持 12 位主分辨率（默认）和可选择的 8 位/10 位次分辨率。

分辨率不同，转换时间也不同：

转换时间（sar_clk）= 分辨率（bit）+ 2

采样和转换总时间（sar_clk）= 采样时间+ 分辨率（bit）+ 2

当配置 SARADC 为 12 位分辨率，采样时间=4（默认值）时，总的采样转换时间需要 18 个 sar_clk。因此，当 sar_clk=18MHz 时，SAR ADC 的转换速率可达 1Msps。如果用户选择较低的分辨率，比如 10 位/8 位分辨率，将可以获得更高的转换率。

19.3.1.6 采样时间

采样时间指 SARADC 在采样和保持阶段所用时间。通过配置寄存器 SAR_SAMPLE_TIME01 和 SAR_SAMPLE_TIME23，用户可以为每个通道选择 4 种不同的采样时间（范围：4~1024 个 SAR ADC 时钟）。

19.3.1.7 SAR ADC 时钟

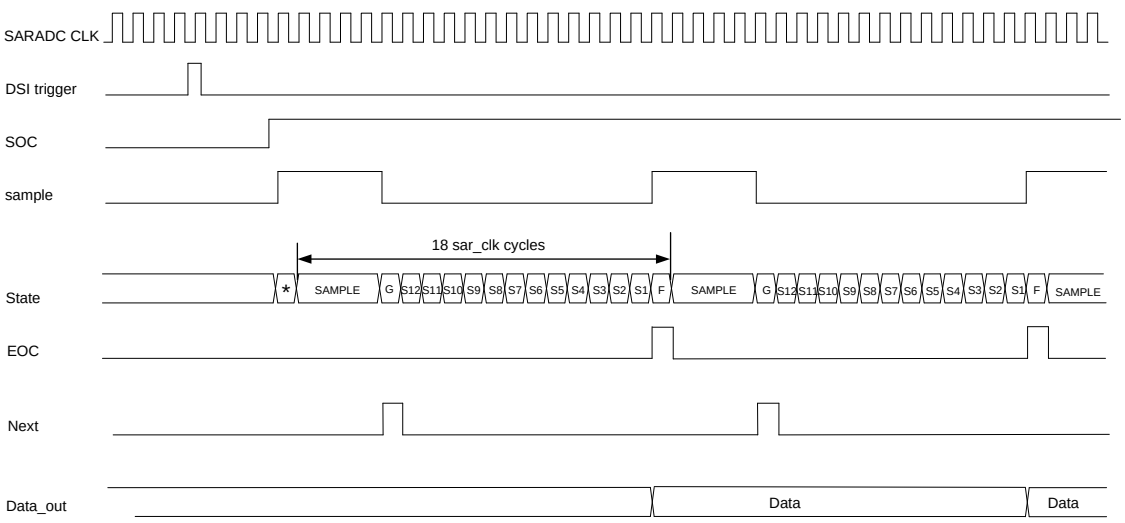
PSoC 4200 器件 SAR ADC 的时钟频率范围为 1MHz~18MHz, 而 PSoC 4100 器件 SAR ADC 的时钟频率范围为 1MHz~14.5MHz，其是通过对 IMO 分频得到的（SAR ADC 不支持小数分频）。如需 1Msps 的采样速率，用户需要将系统时钟（IMO）配置为 36MHz，SAR ADC 时钟配置为 18MHz；PSoC 4100 器件如需 806ksps 的采样速率，用户需要将系统时钟（IMO）配置为 29MHz。

在默认采样时间下（4 个 SAR ADC 时钟）：

- 分辨率为 12 位时，每完成一次采样需要 18 个 SAR ADC 时钟。
- 分辨率为 10 位时，每完成一次采样需要 16 个 SAR ADC 时钟。
- 分辨率为 8 位时，每完成一次采样需要 14 个 SAR ADC 时钟。

19.3.1.8 SAR ADC 工作时序

图 19-2. SAR ADC 工作时序



如图 19-2 所示，DSI 触发信号（DSI Trigger）被触发后，在转换开始信号（SOC）被置高之前，会延时几个 sar_clk。

在 12 位分辨率下，SARADC 转换过程需要 14 个 sar_clk，即

sar_clk *12bits + sar_clk（状态 G） + sar_clk（状态 F ）= 14 sar_clk

SARADC 采样需要 4 个 sar_clk（默认）；所以 SARADC 的整个采样和转换共需 18 个 sar_clk 时钟周期。

当采样完成时，SARADC 会发出一个脉冲信号 Next（或信号 dsi_sample_done），SARMUX 会将其他需要采样的信号切换到 SARADC 的输入端进行采样和转换（在序列控制器 sequencer 的控制下，SARMUX 可以自动切换，详情请参考 19.3.4 SARSEQ）。

19.3.2 SARMUX

SARMUX 是一个专用的多路模拟选择开关。

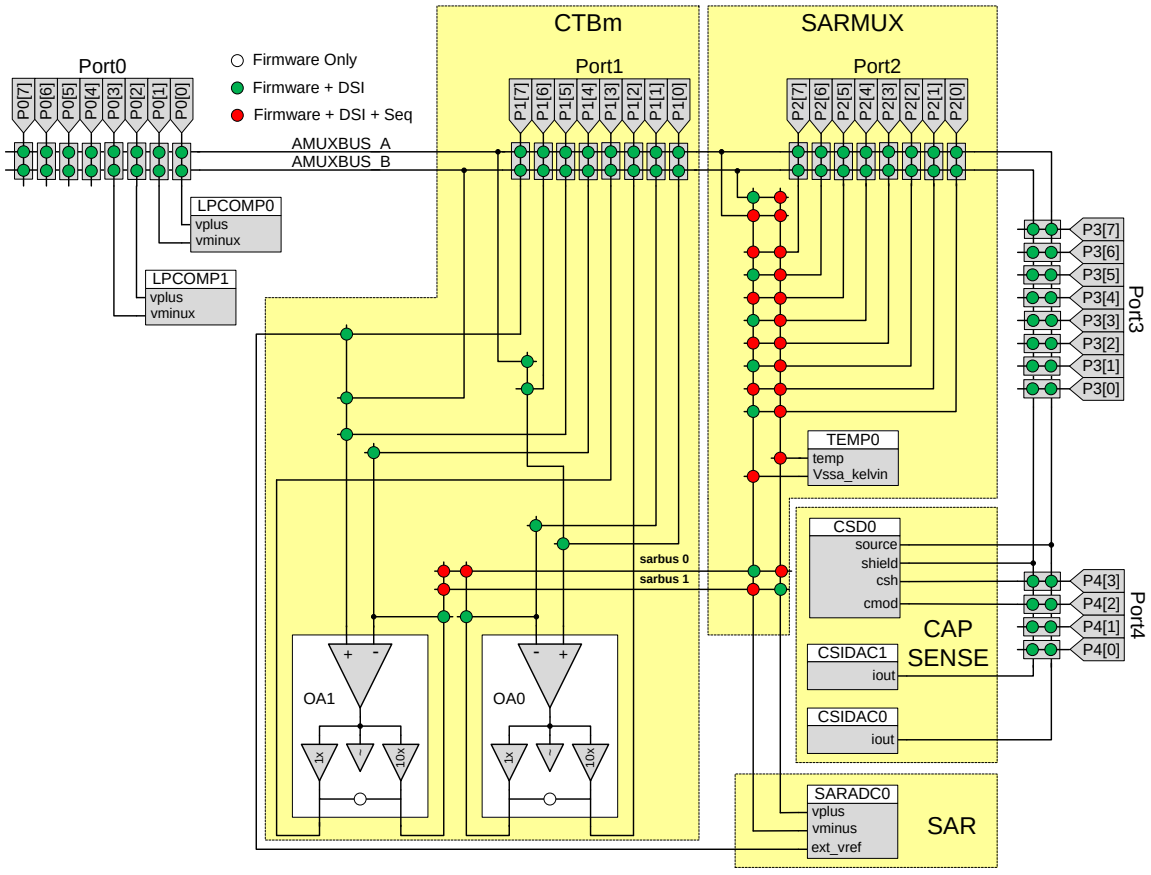
SARMUX 的主要特性如下：

- 开关导通电阻：600Ω（最大值）
- 连接内部温度传感器
- 可由 SARSEQ、UDB 或软件控制
- 内部电荷泵：
  - VDDA <4.0V 时，电荷泵被打开，以减少开关导通电阻
  - VDDA>= 4.0V，可关闭电荷泵，在其输出端提供 VDDA
- 多路输入：
  - 来自管脚（端口 2）的模拟信号
  - 温度传感器的输出信号
  - 经过 sarbus 0/ sarbus 1 的 CTBm 输出信号（速度不够快，无法达到 1Msps）
  - AMUXBUS_A/AMUXBUS_B（速度不够快，无法达到 1Msps）

19.3.2.1 模拟路由

SARMUX 有许多开关，可被 SARSEQ 模块中的序列控制器 Sequencer 控制（下文简称序列控制）、软件控制或 DSI 控制。序列控制和 DSI 控制可被看作硬件控制方法，用户可通过配置寄存器 SAR_MUX_SWITCH_HW_CTRL 的硬件控制位来屏蔽该方法。在不同的控制方法下，开关具有不同的控制能力，见图 19-3。

图 19-3. SARMUX 开关及不同的控制能力



序列控制：在此控制方式下，SARMUX 开关由 SARSEQ 模块中的序列控制器（sequencer）控制。每个通道的模拟路由被配置完成后，在一个循环内，SAR ADC 可以自动扫描多个通道，而无需 CPU 的参与。需要注意的是，并不是每个开关均可被序列控制，见图 19-3。与此控制方式相关的寄存器：SAR_CHANx_CONFIG，SAR_CTRL，SAR_MUX_SWITCH0，SAR_MUX_SWITCH_HW_CTRL。详细配置请参考寄存器模式章节：19.3.8.1 SARMUX 模拟路由配置。

软件控制：在此控制方式下，用户可通过配置寄存器来直接控制 SARADC 的正端（VPLUS）/负端（VMINUS）连接；图 19-3 中 SARMUX 的每个开关均可以被控制。例如，在差分模式下，SARADC 的正负端输入可以来自任何两个管脚/信号，而在序列控制方式下仅可以是相邻的管脚对。但是，多通道的扫描切换，需要 CPU 的参与控制。与此控制方式相关的寄存器：SAR_CTRL，SAR_MUX_SWITCH0，SAR_MUX_SWITCH_HW_CTRL。详细配置请参考寄存器模式章节：19.3.8.1 SARMUX 模拟路由配置。

DSI 控制：在此控制方式下，来自 UDB 的 DSI 信号按着设定的逻辑来控制 SARMUX 开关。除了一些用于芯片设计和测试的开关外，DSI 可以控制绝大多数 SARMUX 开关。因此，DSI 控制方式与软件控制一样，可以为任何两个输入管脚/信号进行差分测量。详细配置请参考 DSI 模式章节：19.3.9.1 SARMUX 模拟路由配置。

19.3.2.2 模拟内部互连

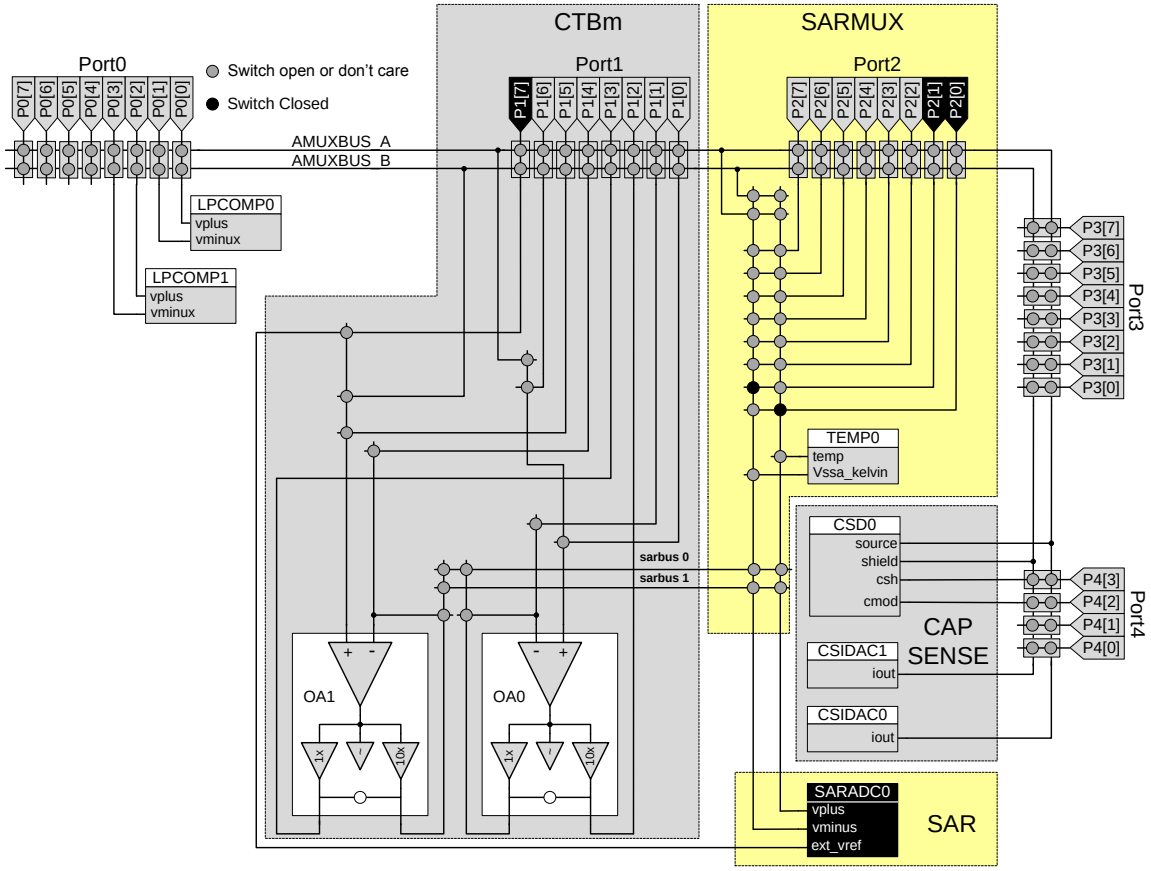
PSoC 4 的内部模拟互连是非常灵活的。通过 SARMUX，可以将来自外部管脚（端口 2）和内部信号的多个输入连接至 SARADC。例如，CTBm 输出信号经过 sarbus0 和 sarbus1 可以连接至 SARMUX；除端口 2 外其他管脚信号经过 AMUXBUS_A/ AMUXBUS_B 也可以连接至 SARMUX；但是这样会影响到扫描性能（因为会产生更大的寄生电容，需要更长的充放电时间）。

接下来将介绍一些使用示例，来帮助用户更好地理解模拟互连。

19.3.2.3 外部管脚输入

如图 19-4 所示，P2.0 和 P2.1 作为差分对分别连接到 SARADC 的正负端，两个开关闭合，这两个开关可以被序列、软件或 DSI 控制（见图 19-3）。但是，如果需要 P2.1 和 P2.2 作为差分对时，仅可使用软件或 DSI 控制。

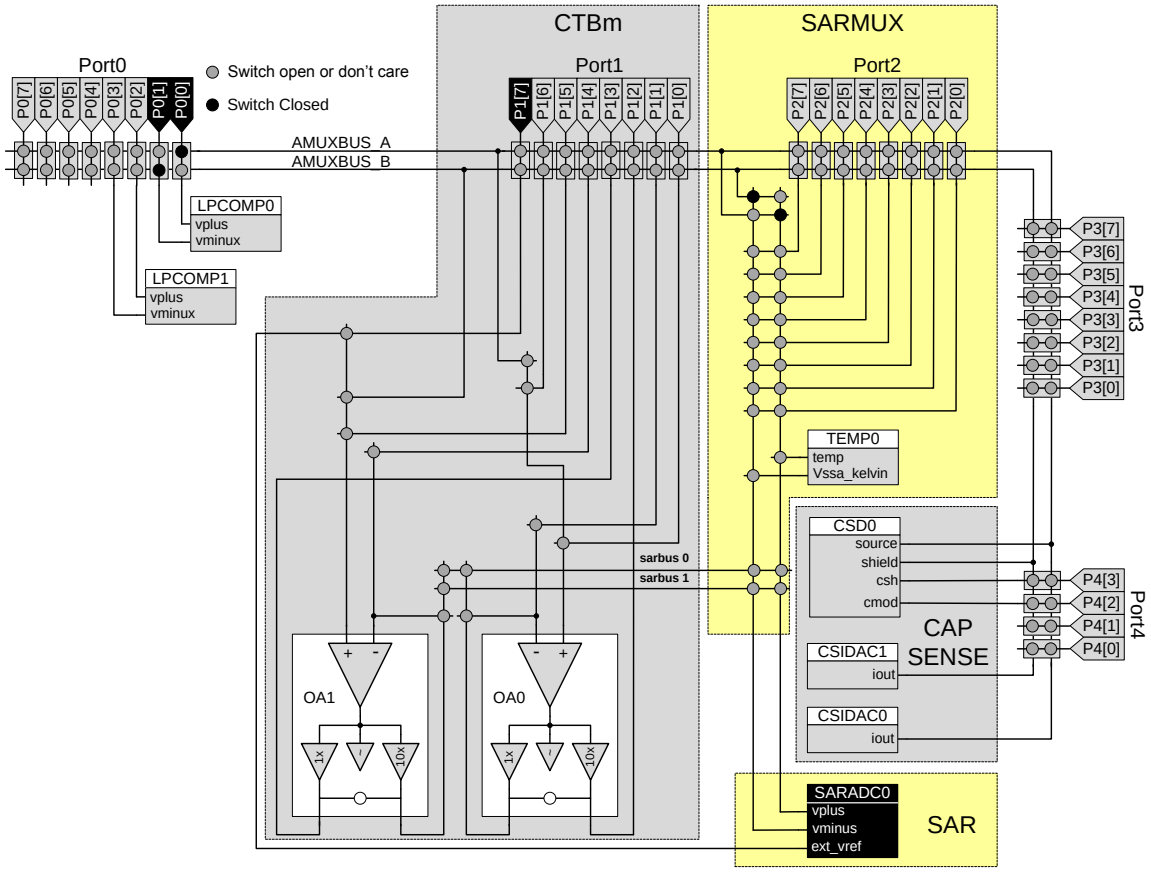
图 19-4. 外部管脚输入



19.3.2.4 来自 AMUXBU_A/AMUXBU_B 的输入

如图 19-5 所示，P0.0 和 P0.1 作为差分对通过 AMUXBUS_A 和 AMUXBUS_B 连接到 SARADC 的正负端，并使用了 SARMUX 之外的开关。这些开关的使用会产生更大的寄生电容，需要更长的充放电时间，因此采样速率无法达到 1Msps。如果端口 2 可用，不推荐用户通过 AMUXBUS_A 和 AMUXBUS_B 来使用外部管脚。

图 19-5. 来自模拟总线 AMUXBUS_A 和 AMUXBUS_B 的输入



19.3.2.5 来自 CTBm 输出的输入

如图 19-3 所示，CTBm 输出通过 sarbus0/ sarbus1 可以连接至 SAR ADC。图 19-6 展示了一个运算放大器（opamp）的输出如何通过一根 sarbus 连接到单端模式的 SAR ADC；其负端连接至 Vref。图 19-7 展示了两个 opamp 的输出如何通过 sarbus0 和 sarbus1 连接到差分模式的 SARADC。这些连接也使用了 SARMUX 之外的开关，从而导致采样速率无法达到 1Msps。但是片上的两个运算放大器可为许多应用提供更多便利。

图 19-6. 来自 CTBm 输出的输入（仅经过一根 sarbus）

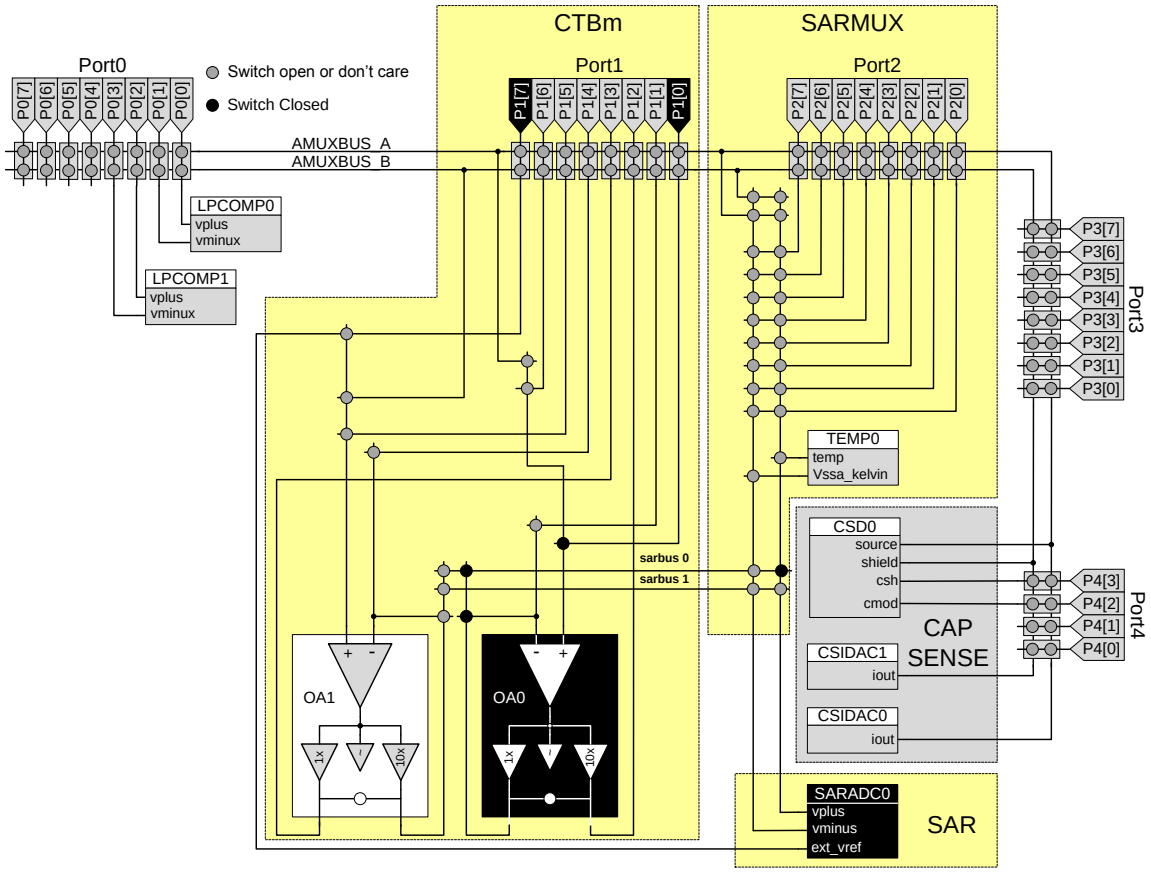
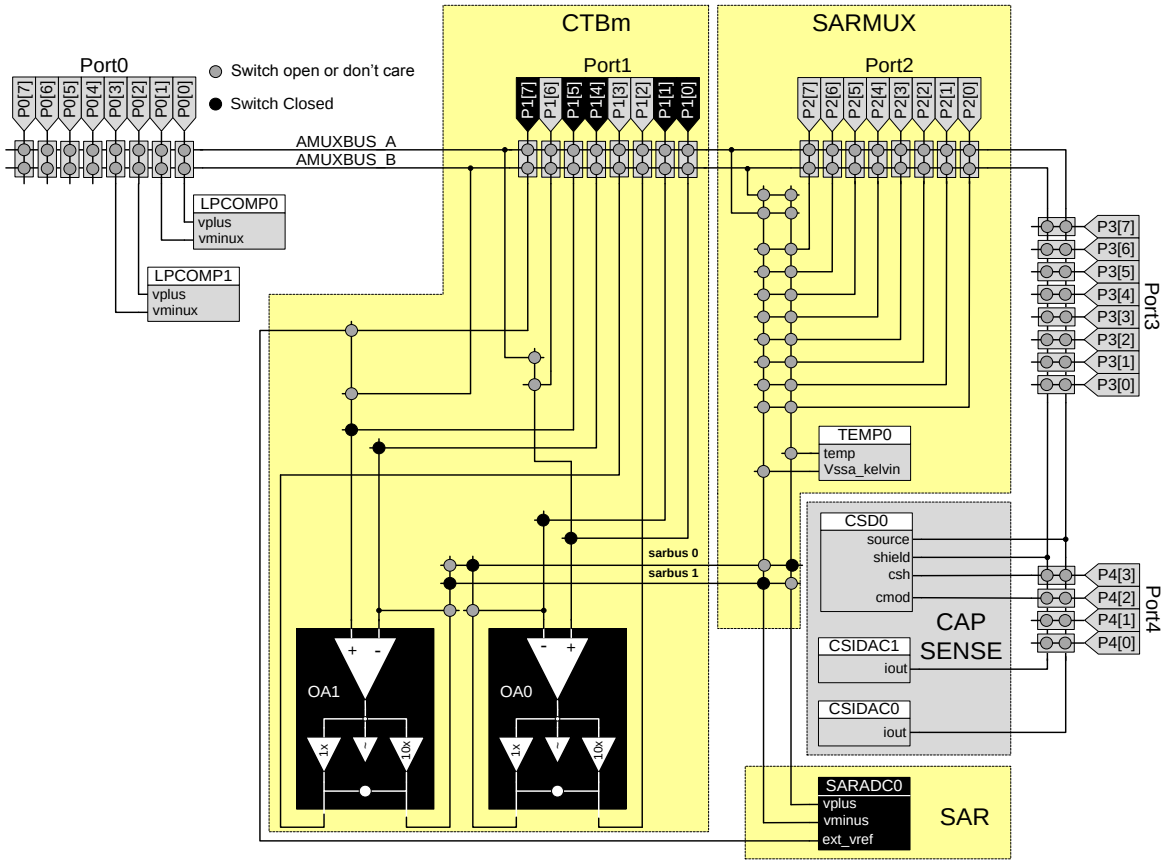


图 19-7. 来自 CTBm 输出的输入（经过 sarbus0 和 sarbus1）



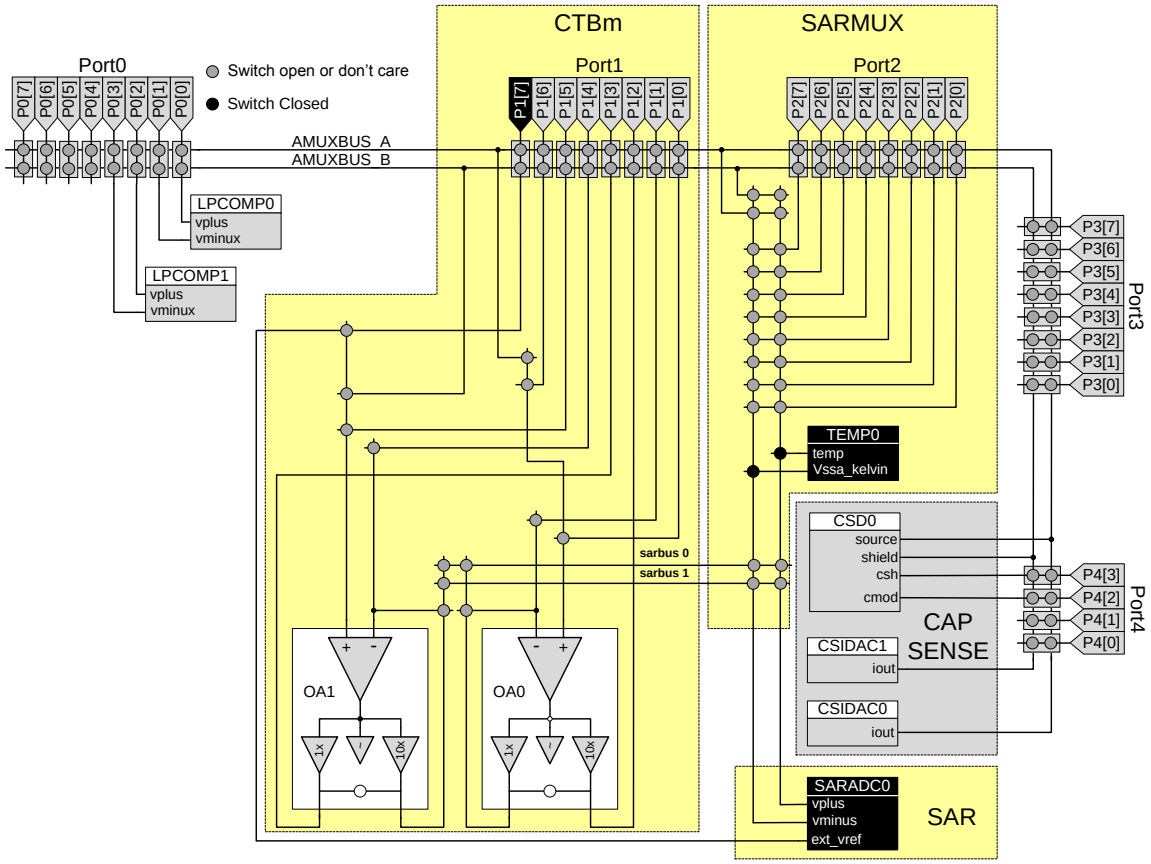
19.3.2.6 温度传感器输入

PSoC 4 包含一个片上温度传感器，其可用于温度传感和基于温度的校准。注意：当对温度传感器信号进行采样时，SARADC 仅可使用单端模式（在差分模式下，转换结果不正确），内部 1.024V 提供参考电压。

如图 19-8 所示，温度传感器信号可被路由至 SARADC 的正端输入，路由开关可以通过 sequencer、软件或 DSI 控制。

注意：设置 MUX_FW_TEMP_VPLUS 位（SAR_MUX_SWITCH0[17]），可以使能温度传感器并将其输出连接到 SARADC 的正端；清除该位将通过切断偏置电流来禁止温度传感器。

图 19-8. 温度传感器输入

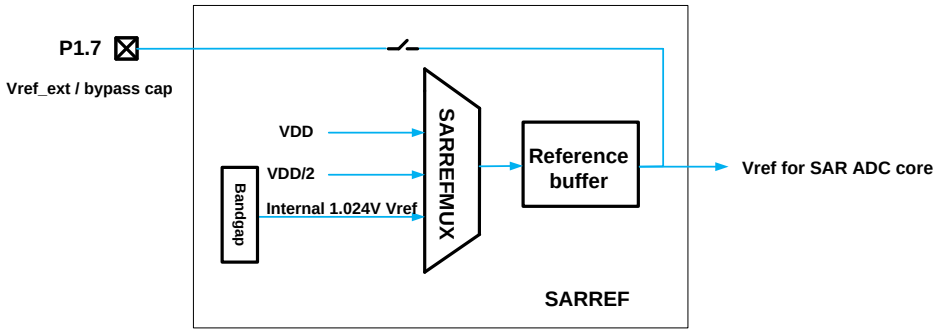


19.3.3 SARREF

SARREF 模块的主要特性如下：

- 参考电压选项：Vdda，Vdda /2，1.024V 带隙（Bandgap）（±1%），外部参考电压
- 使用参考电压缓冲器和旁路电容，可增强内部参考电压的驱动能力

图 19-9. SARMUX 模块框图



19.3.3.1 参考电压选项

SARADC 的参考电压选择电路由 SARREF 中的多路选择器和开关组成。通过配置寄存器 SAR_CTRL[6:4]，使 SARREFMUX 选择如下有效电压：

- Vdda
- Vdda/2
- 来自带隙的内部参考电压 1.024V
- 连接到 P1.7 的外部参考电压

19.3.3.2 旁路电容

电压 1.024V、VDDA/2 和 VDDA 在作为 SARADC 的参考电压之前，会经过缓冲器 Reference Buffer。当选择管脚 P1.7 上的电压作为参考电压时，用户可在此管脚上连接外部电容来过滤参考信号上可能存在的噪声。

如果没有连接外部参考旁路电容，SARADC 的采样速率最大只能达到 166Ksps。例如，当不连接旁路电容，并使用内部参考电容 1.024V 时，SAR ADC 的最大时钟为 3MHz。当使用外部参考电压时，推荐用户连接一个外部电容。可以通过设置 SAR_CTRL[7]来使能旁路电容。

下表列出了使用 12 位分辨率、不同的参考电压模式下，SARADC 的最高工作频率/采样率。

表 19-3. 参考电压对工作频率和采样速率的影响

参考电压	选择参考电压：SAR_CTRL[6:4]	选择旁路电容：SAR_CTRL[7]	选择 Reference buffer	最高工作频率	最大采样速率
内部参考电压 1.024V（无旁路电容）	4	0	Yes	3MHz	166Ksps
内部参考电压 1.024V（有旁路电容）	4	1	Yes	18MHz	1Msps
外部参考电压	5	0/1	NO	18MHz	1Msps
VDDA/2（无旁路电容）	6	0	Yes	3MHz	166Ksps
VDDA/2（有旁路电容）	6	1	Yes	18MHz	1Msps
VDDA	7	0/1	Yes	18MHz	1Msps

内部参考电压 1.024V 的启动时间因不同的旁路电容而不同，下表描述了在两种通用旁路电容下，内部参考电压 1.024V 的启动时间值。如果在两个扫描间改变参考电压至内部参考电压 1.024V，SARADC 开始采样前必须确保内部参考电压 1.024V 达到稳态。

表 19-4. 内部参考电压启动时间

内部参考电压的旁路电容	内部参考电压启动时间最大值
使用外部电容（1uF）	2ms
使用外部电容（100nF）	200us

19.3.3.3 输入电压范围和参考电压

所有的输入电压应该在 VSSA ~VDDA 的范围内。输入电压范围还受 Vref 选择限制。如果负端输入为 Vn，SARADC 的参考电压为 Vref 时，则正端输入范围为 Vn+ / - Vref。这个标准适用于单端和差分模式。

19.3.4 SARSEQ

SARSEQ 是一个专用序列控制器，可以自动地控制 SARMUX 依次切换输入通道，同时将转换结果保存在结果寄存器组中。

SARSEQ 可以对 每个通道：

- 自动地控制 SARMUX 的模拟路由，而无需 CPU 干预。
- 控制 SARADC（分辨率/采样时间/参考电压等）
- 接收 SARADC 转换数据，并做预前处理（求平均、范围检测等）
- 转换结果数据被双缓冲保存，可使 CPU 安全地读取最近一次扫描完成的结果

重要功能：

- 8 个通道可单独使能；在无需 CPU 参与情况下，自动扫描使能通道
- 在自动扫描过程中可对插入通道（第 9 个通道）进行间隙采样转换。
- 每通道可选择：
  - 输入信号来自外部管脚或内部信号（AMUXBUS/CTBm/温度传感器）
  - 4 个可配置的采样时间

- 主分辨率：12 位 分辨率（默认）；次分辨率：8 位/10 位
- 单端模式或差分模式
- 转换结果求平均

■ 扫描触发
- 单次模式、周期模式或连续模式
- 触发信号来自任何数字信号或 GPIO 管脚输入
- 支持由 UDB 构成的固定功能模块触发
- 软件触发

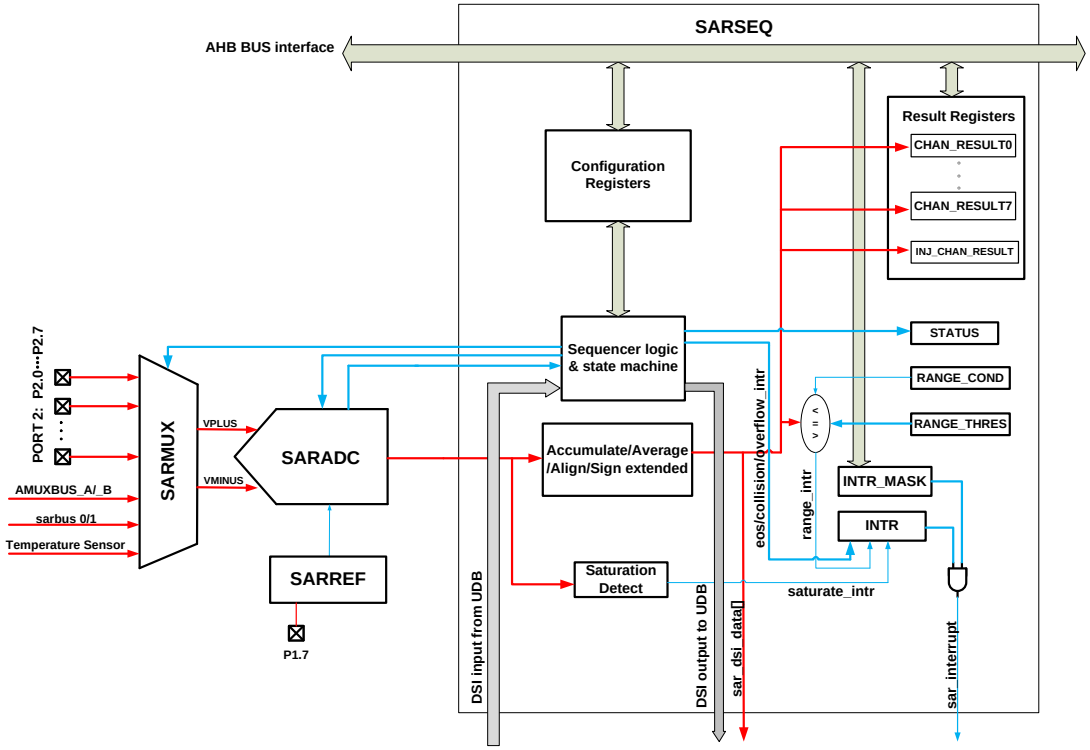
■ 硬件求平均
- 转换结果累加
- 支持对 2^n（1≤n≤8）个采样信号求平均
- 将转换结果进行符号扩展至 16 位后保存

■ 对输出数据进行双缓冲保存
- 左对齐或右对齐
- 转换结果保存在工作寄存器（working register）和结果寄存器（result register）中

■ 中断
- 扫描转换完成中断
- 通道饱和检测中断
- 通道阈值检测中断（阈值可配置）
- 扫描结果溢出中断
- 冲突检测中断

■ 可配置的注入通道
- 软件触发
- 支持扫描追尾功能
- 可配置采样时间、分辨率、单端/差分输入模式和求平均

图 19-10. SARSEQ 模块框图



19.3.4.1 *求平均*

SARSEQ 模块中有一个 20 位的累加、移位寄存器，可以执行求平均运算。求平均运算是在符号扩展之后进行的。用户可通过配置寄存器 SAR_SAMPLE_CTR 求平均。

在寄存器控制模式下，用户可通过配置 AVG_EN 位（SAR_CHAN_CONFIG[10]）来使能某个通道是否需要求平均运算。在 DSI 控制模式下，信号 dsi_cfg_average 决定是否使能求平均运算。

AVG_CNT 字段（SAR_SMAPLE_CTRL [6:4]）决定了求平均运算需要的采样次数（N）。

$N = 2^{AVG_CNT+1}$　N = [2..256]

例如，当 AVG_CNT=3 时，则 N=16。

当求平均运算被使能时，用户还需要设置 AVG_SHIFT 位（SAR_SAMPLE_CTRL[7]），来对结果进行移位求平均。

如需对某个通道进行求平均运算，在每个扫描周期中，SARSEQ 将对该通道连续执行 N 次采样，每次采样的结果经过符号扩展后进行累加。N 个数据累加之后，通过对结果进行移位求平均。因转换结果分辨率最大为 12 位，N 最大值为 256，所以 20 位的累加器不会溢出。

如果 AVG_SHIFT（SAR_SAMPLE_CTRL[7]）位设置为 1，硬件会对累加值右移 AVG_CNT+1 位得到平均值。如果 AVG_SHIFT（SAR_SAMPLE_CTRL[7]）=0，硬件会对结果强制向右移位，以使结果存放到 16 位寄存器中（右移位数为 0 和 AVG_CNT-3 中较大的值）。也就是说，如果采样次数大于 16（AVG_CNT >3），那么累加结果将被右移 AVG_CNT-3 位；如果 AVG_CNT<3，则转换结果不右移。但需要注意的是，这种情况下求平均的结果可能会比期望值大，所以建议用户将 AVG_SHIFT 位置 1。

移位后，转换结果保存在 16 位记过寄存器中。注意：求平均时总是使用 12 位分辨率和右对齐，也就是说 RESOLUTION 位（SAR_CHANx_CONFIG [9]）和 LEFT_ALIGN 位（SAR_SAMPLE_CTRL[1]）不起作用。

19.3.4.2 阈值检测

SARSEQ 支持在无 CPU 参与的情况下自动对采样值与上下限阈值比较，即阈值检测。上下限阈值分别保存在寄存器 SAR_RANGE_THRES 的 RANGE_HIGH 字段（SAR_RANGE_THRES[31:16]）和 RANGE_LOW 字段（SAR_RANGE_THRES[15:0]）中。

当某个通道的转换结果满足一定条件时，将触发可屏蔽的阈值检测中断（RANGE_INTR）。用户可通过对 RANGE_COND（SAR_RANGE_COND[31:30]）位进行如下配置来设置触发条件。

0: 结果值 < RANGE_LOW（小于下限阈值）

1: RANGE_LOW <=结果值< RANGE_HIGH（上下限阈值间）

2: RANGE_HIGH <=结果值（高于上限阈值）

3: 结果值<RANGE_LOW || RANGE_HIGH <=结果值（上下限阈值之外）

关于阈值检测中断的详细描述，请参考 19.3.5.5 阈值检测中断章节。

19.3.4.3 双缓冲保存

SAR ADC 使用双缓冲保存转换结果，这样可以在执行扫描的同时读取上次的转换结果。在扫描进行时，SAR ADC 结果会被写入相应通道的工作寄存器（working register）中；一旦这次扫描完成，转换结果会被拷贝到相应通道的结果寄存器（result register）中，以备应用程序读取。在当前扫描完成前，软件应有足够的时间来读取上次的转换结果，否则，上次的转换结果会被破坏。除插入通道外，每个通道均有两个寄存器来执行双缓冲保存；插入通道因不包含在常用的扫描序列中，不需要对结果进行双缓冲保存。

19.3.4.4 插入通道

插入通道与其他通道类似，但不属于自动扫描的一部分。其可对输入信号进行间隙采样、转换；例如，每 2 秒对温度传感器进行一次采集。但是，如果 SARADC 工作于连续模式下，使能插入通道将改变其他通道的采样速率。

插入通道仅可通过软件设置 INJ_START_EN 位（SAR_INJ_CHAN_CONFIG[31]）来触发，并且其数据和中断均不能发送至 DSI 总线。由于软件触发是单次触发，所以其不支持双缓冲和溢出中断。

用户可以通过寄存器 SAR_INJ_CHAN_CONFIG 来配置插入通道，其配置方式与其他通道类似。插入通道支持如下特性：

- 管脚/信号 选择
- 单端/差分模式选择
- 分辨率选择：12 位 — 主分辨率; 8 位/10 位 — 次分辨率
- 4 种采样时间选择
- 求平均

插入通道支持如下中断：

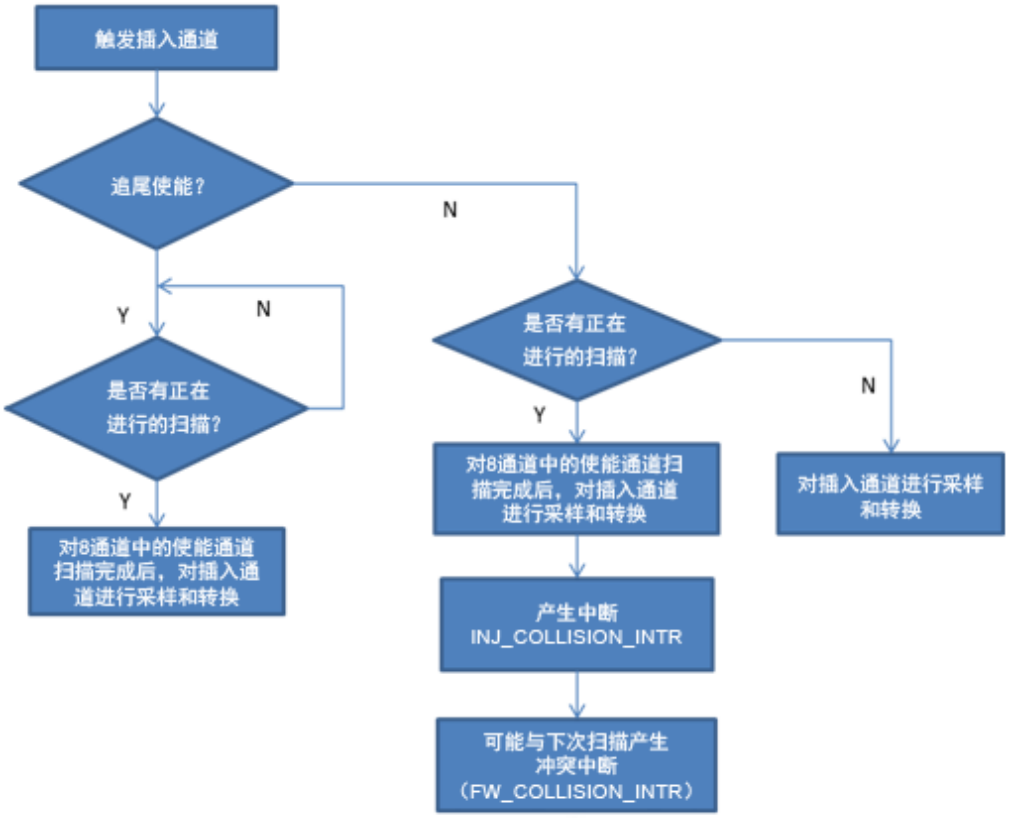
- 插入通道转换结束中断 INJ_EOC_INTR
- 插入通道阈值检测中断 INJ_RANGE_INTR
- 插入通道饱和检测中断 INJ_SATURATE_INTR
- 插入通道冲突检测中断 INJ_COLLISION_INTR

插入通道所对应的中断配置寄存器分别是：SAR_INJ_CHAN_CONFIG，SAR_INJ_RESULT，SAR_INTR，SAR_INTR_MASK，SAR_INTR_MASKED 和 SAR_INTR_SET。

追尾功能：软件可以通过对 INJ_START_EN 位（SAR_INJ_CHAN_CONFIG [31]）置 1 来触发对插入通道的采样、转换。如果有一个扫描正在进行，推荐通过设置 INJ_TAILGATING（SAR_INJ_CHAN_CONFIG [30]）=1 使能追尾功能，则 INJ_START_EN 仅在当前扫描结束后才使能插入通道来进行采样转换，而不产生任何冲突中断。如果当前没有扫描进行（SARADC 处于空闲状态），并使能了追尾功能，则 INJ_START_EN 将等待至下一个扫描结束后才使能插入通道；在这种情况下，建议禁止追尾功能。

当追尾功能被禁止，SARADC 处于空闲状态时，插入通道的转换会被立即执行。当追尾功能被禁止，SARADC 正执行其他通道的扫描时，触发插入通道转换将与当前的扫描产生冲突，并产生一个插入通道冲突检测中断 INJ_COLLISION_INTR；在这种情况下，插入通道的转换将被推迟到扫描结束后执行。还有一种情况（追尾被禁止），其他通道的下一个扫描可能与当前进行的插入通道转换产生冲突，并产生一个冲突中断 FW_COLLISION_INTR 或 DSI_COLLISION_INTR；在这种情况下，其他通道的扫描将被推迟到插入通道完成后执行，这会对定期扫描产生一个抖动。但需要注意的是，由连续触发或 DSI 触发（电平触发模式）激活的扫描在任何情况下均不产生冲突中断。

图 19-11. 插入通道的追尾功能流程图

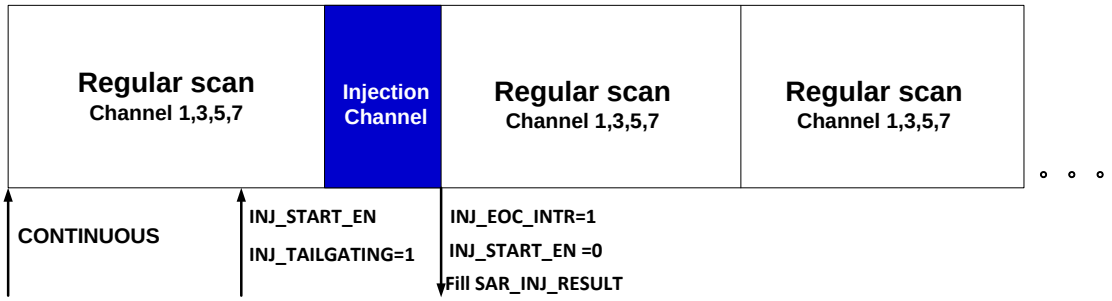


使用追尾功能时，存在一个缺点：在下一个触发前可能需要等待较长时间。如果插入通道对其他通道扫描不会产生冲突或引起抖动，可以禁止追尾功能。

插入通道完成转换后，插入通道转换结束中断 INJ_EOC_INTR 位（SAR_INTR[4]）会被置 1， INJ_START_EN 位（SAR_INJ_CHAN_CONFIG [31]）被清除；转换结果会被保存在寄存器 SAR_INJ_RESULT 的位[15:0]中。32 位寄存器 SAR_INJ_RESULT 的位[31:28]分别是 INJ_EOC_INTR（SAR_INTR[4]）、INJ_RANGE_INTR（SAR_INTR[6]）、INJ_SATURATE_INTR（SAR_INTR[5]）和 INJ_COLLISION_INTR（SAR_INTR[7]）等位的拷贝。

下图描述了一个插入通道在连续模式扫描间被使能的例子，其中通道 1、通道 3、通道 5 和通道 7 被使能，并且插入通道的追尾功能也被使能。

图 19-12. 连续模式扫描和插入通道



但需要注意的是，如果已使能的 SARADC 被禁止，则使能的插入通道的 INJ_START_EN 位（SAR_INJ_CHAN_CONFIG [31]）会被立即清除。

19.3.5 中断

接下来将介绍 SAR ADC 产生的中断信号：扫描结束中断（EOS_INTR）、溢出中断、冲突中断、插入通道转换结束中断、阈值检测中断及饱和检测中断。每个中断在寄存器 SAR_INTR_MASK 中均对应一个中断屏蔽位；如果屏蔽位设为 0，则对应的中断信号会被忽略。当屏蔽位为 1，并且相应的中断信号处于挂起状态时，SAR ADC 中断将会产生。

在 SAR ADC 的中断服务程序中，应用程序读取数据后，应该向中断位写 1 来清中断。

寄存器 SAR_INTR_MASKED 中保存了中断源与中断屏蔽位逻辑与的结果，其可用于软件识别中断源。

寄存器 SAR_INTR_SET 支持通过软件向相应位写 1 来触发相应中断，方便软件验证和调试。

19.3.5.1 扫描结束中断（EOS_INTR）

每次扫描结束时，SARADC 将产生一个扫描结束中断 EOS_INTR。建议用户在读取数据后及时清除此中断。

用户可通过设置 EOS_DSI_OUT_EN 位（SAR_SAMPLE_CTRL [31]）来控制 EOS_INTR 是否需要发送到 DSI 总线；如果发送，则 EOS_INTR 将在 DSI 总线上保持两个系统时钟周期。这些时钟周期与刚扫描通道的信号 sar_dsi_data_valid 保持一致

EOS_MASK 位（SAR_INTR_MASK[0]）是中断 EOS_INTR 的屏蔽位。EOS_MASKED 位（SAR_INTR_MASKED[0]）保存了 EOS_INTR 与 EOS_MASK 逻辑与的结果。用户也可以通过设置 EOS_SET 位（SAR_INTR_SET[0]）位来产生中断，而不需要 SARSEQ 和 SARADC 的参与，方便软件验证和调试。

19.3.5.2 溢出中断

每完成一次扫描，SARADC 将产生一个扫描结束中断 EOS_INTR，如果此时 EOS_INTR 仍然为 1，即没有被软件及时清除，那么溢出中断 OVERFLOW_INTR 将被产生，并且原来的数据会被覆盖。

溢出中断 OVERFLOW_INTR 可被 OVERFLOW_MASK 位（SAR_INTR_MASK[1]）位屏蔽。中断 OVERFLOW_INTR 与 OVERFLOW_MASK 位逻辑与的结果被保存在 OVERFLOW_MASKED 位（SAR_INTR_MASKED[1]）中。通过对 OVERFLOW_SET 位（SAR_INTR_SET[1]）写 1，可软件设置中断 OVERFLOW_INTR，其可方便软件验证和调试。

19.3.5.3 冲突中断

当 SARSEQ 正忙于扫描控制的过程中，一个新的触发（见触发章节）发生了。那么这个新触发的扫描将在本次扫描完成后才会有效。此时，SARADC 将产生一个冲突中断来通知软件此触发的扫描将会被延迟执行。在非连续触发模式下，这个新的触发可在任何时候发生。

冲突中断包括 3 种类型：软件触发冲突中断（FW_COLLISION_INTR），DSI 触发冲突中断（DSI_COLLISION_INTR）和插入通道冲突中断（INJ_COLLISION_INTR）。软件可通过读取寄存器 SAR_INTR 来辨别冲突中断的类型。

注意：当 DSI 触发是电平方式触发时，将不会产生 DSI_COLLISION_INTR（见 DSI 触发配置）。

这三种冲突中断可被寄存器 SAR_INTR_MASK 中的相应位屏蔽；其与寄存器 SAR_INTR_MASK 相应屏蔽位的逻辑与的结果被保存在寄存器 SAR_INTR_MASKED 的相应位中，以方便软件处理。对寄存器 SAR_INTR_SET 的相应位写 1 可产生相应的冲突中断，方便软件调试和验证。

19.3.5.4 插入通道转换结束中断（INJ_EOC_INTR）

当插入通道的转换结束时，硬件将产生一个插入通道转换结束中断 INJ_EOC_INTR。在应用程序从寄存器 SAR_INJ_RESULT 中读取数据后，请及时清除此中断。

注意：如果插入通道紧随一个扫描进行，则 SARADC 在产生中断 EOS_INTR 的同时开始对插入通道进行转换，也就是说插入通道不是扫描的一部分。

中断 INJ_EOC_INTR 可被 INJ_EOC_MASK 位（SAR_INTR_MASK[4]）屏蔽。中断 INJ_EOC_INTR 与 INJ_EOC_MASK 位逻辑与的结果存放在 INJ_EOC_MASKED 位（SAR_INTR_MASKED[4]）中，以方便软件处理。向 INJ_EOC_SET 位（SAR_INTR_SET[4]）写 1，也可触发中断 INJ_EOC_INTR，其可方便软件调试和验证。

19.3.5.5 阈值检测中断

当对 SARADC 使能了求平均、左右对齐或符号扩展，则阈值检测中断将在这些处理后产生，也就是说不需要等到整个扫描结束。阈值需要与结果数据具有相同的数据格式（见转换结果数据格式）。

阈值检测中断可以被寄存器 SAR_RANGE_INTR_MASK 相应 RANGE_MASK 位（此位设置为 0）屏蔽，其与寄存器 SAR_RANGE_INTR_MASK 的相应位逻辑与的结果保存在寄存器 SAR_RANGE_INTR_MASKED 的相应位中。因此，如果对应的 RANGE_MASK 位不为 0，当阈值检测中断发生时，发送至 NVIC 的 SARADC 中断信号为高。

当程序向寄存器 SAR_RANGE_INTR_SET 的相应位写 1 时，可产生相应的阈值检测中断，其可方便软件调试和验证。

每个通道均有一个阈值检测中断（RANGE_INTR 和 INJ_RANGE_INTR）。

19.3.5.6 饱和检测中断

SARADC 每完成一次转换后，会进行一次饱和检测。当饱和检测中断发生时，表示该通道的采样值等于相应分辨率下的下限值或上限值。软件接收到该中断后可以选择放弃此次转换结果。

当通道选择 10 位或 8 位分辨率时，其高位将被忽略。

饱和检测在每次转换之后、求平均之前进行。所以饱和中断发生时，求平均后的结果并不一定等于上限值或下限值。

一旦采样数据饱和，SARADC 就会立即产生饱和检测中断。每个通道的饱和检测中断均可被寄存器 SAR_SATURATE_INTR_MASK 相应的 SATURATE_MASK 位（此位设置为 0）屏蔽。饱和检测中断与屏蔽位 SATURATE_MASK 逻辑与的结果存放在寄存器 SAR_SATURATE_INTR_MASKED 的相应位中。因此，如果对应的 SATURATE_MASK 位不为 0，当饱和检测中断发生时，发送至 NVIC 的 SARADC 中断信号为高。

当应用程序向寄存器 SAR_SARTURATE_INTR_SET 的相应位写 1 时，可产生相应通道的饱和检测中断，其可方便软件调试和验证。

19.3.5.7 中断源识别

寄存器 SAR_INTR_CAUSE 中包含了 SARADC 当前挂起的中断。中断服务程序可以通过读此寄存器来辨别产生中断的来源。

寄存器 SAR_INTR_CAUSE 的位[7：0]的值与寄存器 SAR_INTR_MASKED 的位[7：0]相同，INTR_CAUSE 的位 [31：30] 分别是 8 个通道的阈值检测和饱和检测中断标志位，其分别是寄存器 SAR_RANGE_INTR_MASKED 和 SAR_SATURATE_INTR_MASKED 中的 8 个通道相应位逻辑或的值（插入通道的 INJ_RANGE_INTR 和 INJ_SATURATE_INTR 除外）。

19.3.6 触发

只有当触发被激活时，相应的扫描才开始工作；SAR ADC 支持 3 种触发方式：软件触发, DSI 触发和 连续触发，详情如下：

- 软件触发：当应用程序向 FW_TRIGGER 位（SAR_START_CTRL[0]）写 1 时，将激活一个软件触发。软件触发是单次触发，可触发对使能通道的扫描。当扫描完成时，SARSEQ 会清除 FW_TRIGGER 位并返回空闲模式，等待下一个触发。如果 SARADC 被禁用，则 FW_TRIGGER 位会被立刻清除。
- DSI 触发：DSI 触发是由来自 DSI 的信号 dsi_trigger 产生的触发。dsi_trigger 可以被连接到 TCPWM 的输出，也可被连接到任意 GPIO 或 UDB。例如，UDB 执行一个状态机以响应特定事件序列，进而产生触发。
- 连续触发：通过软件设置 CONTINUOUS 位（SAR_SAMPLE_CTRL[16]），可以激活连续触发。在这种触发模式下，SARSEQ 将不间断、连续地执行扫描，一直处于忙状态。在扫描的过程中，软件触发和 DSI 触发将不起作用，但是被置 1 的 FW_TRIGGER 位将在下一个扫描结束时被清除。

建议用户在同一时刻，只使用一个触发方式。如果 DSI 触发和软件触发同时发生了，则将优先处理 DSI 触发而执行一次扫描，然后再对软件触发执行一次扫描（同时产生一个软件触发冲突中断，见冲突中断章节）。当 DSI 触发和连续触发同时发生时，两个触发将被有效地同时处理（同时产生一个 DSI 触发的冲突中断，见冲突中断章节）。

当软件触发或连续触发被激活时，处于空闲状态的序列发生器将在 1 个 SARADC 时钟周期后通知 SARADC 开始采样。当 DSI 触发被激活时，SARADC 开始采样的时间将依赖于触发配置（见 DSI 触发配置章节）。

19.3.6.1 DSI 触发配置

DSI 同步：SARSEQ 的 DSI 接口应与 AHB 时钟同步。如果 DSI 触发信号与 AHB 时钟不同步，则需要采用同步触发器来实现同步（默认）；如果已同步，则同步触发器可被旁路。对于高频信号，同步是必不可少的，以避免亚稳定性。DSI_SYNC_TRIGGER 位（SAR_SAMPLE_CTRL[19]）可控制同步触发器是否被旁路。但是 DSI_SYNC_TRIGGER 会影响 DSI 脉冲触发信号的触发宽度（TW）和触发间隔（TI）。

DSI 触发电平：用户可以通过 DSI_TRIGGER_LEVEL 位（SAR_SAMPLE_CTRL[18]）来选择 DSI 触发为脉冲触发还是电平触发。如果是电平触发，当 SARADC 完成一次扫描，如果 DSI 触发信号仍为高电平，则再次开始一个新的扫描；如果是脉冲触发，DSI 触发信号应是一个脉冲输入，经过上升沿检测电路进而触发一次扫描。

传送时间：从 dsi_trigger 有效至 SARADC 被通知开始采样需要一定时间，并因 DSI_SYNC_TRIGGER 位（SAR_SAMPLE_CTRL[19]）和 DSI_TRIGGER_LEVEL 位（SAR_SAMPLE_CTRL[18]）的配置不同而不同，其最大时间间隔见下表。两次触发脉冲间的时间间隔必须大于传送时间，否则后一个将无效。

当 SARADC 被禁用（ENABLED（SAR_CTRL[31]）=0）时，DSI 触发将不起作用。

表 19-5. 从 dsi_trigger 有效至 SARADC 被通知开始采样的传送时间

DSI_TRIGGER 的最大传送时间	旁路同步 DSI_SYNC_TRIGGER （SAR_SAMPLE_CTRL[19]）=0	使能同步 DSI_SYNC_TRIGGER （SAR_SAMPLE_CTRL[19]）=1（默认）
脉冲触发 DSI_TRIGGER_LEVEL （SAR_SAMPLE_CTRL[18]）=0（默认）	1 clk_sys*+2 clk_sar**	3 clk_sys+2 clk_sar
电平触发 DSI_TRIGGER_LEVEL （SAR_SAMPLE_CTRL[18]）=1	2 clk_sar	2 clk_sys+2 clk_sar

*clk_sys 指系统时钟周期
**clk_sar 指 SARADC 时钟周期

表 19-6. 对触发信号的要求:

触发规范	要求
触发宽度（TW）	触发宽度需要足够大，才能被检测到。 当 DSI_SYNC_TRIGGER=1 时, TW >= 2 clk_sys cycle. 当 DSI_SYNC_TRIGGER=0 时, TW >= 1 SAR clock cycle.
触发间隔（TI）	两次 DSI 脉冲触发信号的时间间隔必须大于传送时间（见上表），否则后一个将无效

19.3.7 SAR ADC 状态

应用程序可通过读取 BUSY 位（SAR_STATUS[31]）和 CUR_CHAN 字段（SAR_STATUS[4:0]）获得 SARADC 的当前状态。当 SARADC 对某个通道进行采样或转换时，BUSY 位被置 1；CUR_CHAN 指示当前的转换通道。寄存器 SAR_STATUS 的 SW_VREF_NEG 位（SAR_STATUS[31]）表示负端与参考电压 VREF 的连接状态。

在扫描进行中，当某通道的采样完成时，寄存器 SAR_CHAN_WORK_VALID 的相应位会被置 1；当扫描结束时，寄存器 SAR_CHAN_RESULT_VALID 的相应位才会被置 1。CUR_AVG_ACCU 字段（SAR_AVG_STAT[19:0]）和 CUR_AVG_CNT 字段（SAR_AVG_STAT[31:24]）分别存放当前求平均的累加器中的值及采样次数（递减计数）。寄存器 SAR_MUX_SWITCH_STATUS 中存放寄存器 MUX_SWITCH0 相应位的状态。这些寄存器可方便用户调试 SAR ADC。

19.3.8 低功耗模式

SAR ADC 模块的电流消耗主要包括两部分：SARADC 和 SARREF。

用户可通过多种方式来降低功耗，比如关闭 SARADC 的电源和切换时钟，减少每秒的转换次数；最简单的方式是减少触发频率，即减少每秒钟内的转换次数。

PSoC 4 也支持通过 ICONT_LV 字段（SAR_ CTRL[25:24]）来设置 SARADC 的低功耗模式，见下表。

表 19-7. SARADC 低功耗模式设置

ICONT_LV（SAR_CTRL[25:24]）	SARADC 的相对功耗	最大工作频率[MHz]	最小采样时间 [时钟周期 cycles]	最大采样速率（@ 12 位）[ksps]
0	100%	18	4	1000
1	50%	9	3	529
2	133%	18	4	1000
3	25%	4.5	2	281

此外，应用程序也可设置 SARREF 的功耗为不同的低功耗模式。当使用内部参考电压，并且没有旁路电容时，需要降低 SARADC 时钟频率；如果使用旁路电容或使用外部参考电压，则 SARADC 最大频率仍为 18MHz。

表 19-8. SARREF 的低功耗模式设置

PWR_CTRL_VREF（SAR_CTRL[15:14]）	SARREF 的相对功耗	无旁路电容时 SARADC 的最大工作频率	最小采样时间	最大采样速率（@ 12 位）[ksps]
0：NORMAL_PWR	100%	3	1	230
1：HALF_PWR	50%	1.5	1	115
2：THIRD_PWR	33%	1	1	76
3：QUARTER_PWR	25%	18*	4	1000

*在使用最低功耗模式（25%）时，必须加旁路电容。

最后，也可通过使用低分辨率来降低 SARADC 的功耗。比如使用 8 位分辨率，其转换时钟与 12 位分辨率相比减少了 4 个 SARADC 的时钟周期，其采样时间也最小。

19.3.9 系统操作

本章节主要介绍如何使用 SAR ADC。关于 SARADC、SARMUX、SARREF、SARSEQ、中断和触发的理论，请参考上面相关章节。

用户通过 ENABLED 位（SAR_CTRL [31]）使能 SAR ADC 模块后，可通过如下步骤来配置 SAR ADC 模块，并通过 SARSEQ 控制 SARADC 完成数据的采样和转换。

1. 配置 SARADC 控制模式：寄存器模式或 DSI 模式
2. 配置 SARMUX（管脚/信号选择）
3. 配置 SARSEQ
4. 配置通道（比如管脚地址等）
5. 使能通道
6. 设置触发方式
7. 配置中断屏蔽
8. 开启触发源
9. 每个通道的转换结束中断发生后，取回数据
10. 插入通道转换（可选）

在寄存器模式下，用户可通过配置寄存器来使用 SAR ADC 模块；在 DSI 模式下，来自 UDB 的 DSI 信号来操作 SAR ADC 模块。两者间的不同见下表。使用寄存器模式还是 DSI 模式，是由 DSI_MODE 位（SAR_CTRL[29]）决定的。

表 19-9. 寄存器模式和 DSI 模式对比

配置项	寄存器模式	DSI 模式
DSI_MODE（SAR_CTRL[29]）	0	1
SARMUX 控制	序列（Sequencer）控制寄存器： SAR_CHANx_CONFIG SAR_MUX_SWITCH0 SAR_MUX_HW_SWITCH_CTRL SAR_CTRL 软件控制寄存器： SAR_MUX_SWITCH0 SAR_MUX_HW_SWITCH_CTRL SAR_CTRL	DSI 控制信号： dsi_out, dsi_oe, dsi_swctrl, dsi_sw_negvref 软件控制寄存器: 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： SAR_CHAN_CONFIG, SAR_CHAN_EN, SAR_INJ_CHAN_CONFIG	DSI 信号： dsi_cfg_st_sel dsi_cfg_average dsi_cfg_resolution dsi_cfg_differential （SAR_CHAN_CONFIG，SAR_CHAN_EN 和 SAR_INJ_CHAN_CONFIG 不可用）
触发方式	软件 触发 DSI 触发 连续触发	
中断	所有中断均可被输出	仅可将中断 EOS_INTR, RANGE_INTR 和 SATURATE_INTR 通过 DSI 输出
DSI 输出	支持	
结果数据	支持 8 个 通道结果寄存器 1 个插入通道结果寄存器	仅支持通道 0 的结果寄存器
插入通道	支持	不支持
求平均	支持对单个管脚/信号输入求平均	可支持对不同管脚/信号输入求平均

19.3.10 寄存器模式

在寄存器模式下，用户可以通过配置寄存器来控制 SAR ADC。关于寄存器位的定义，请参考寄存器 TRM。

19.3.10.1 SARMUX 模拟路由配置

在寄存器模式中，用户可通过两种方式来控制 SARMUX 的模拟路由：序列控制和软件控制。

序列控制：用户在使用序列控制方法时，需将寄存器 MUX_SWITCH_HW_CTRL 和 MUX_SWITCH0 中的相应位置 1，同时必须设置 SWITCH_DISABLE（SAR_CTRL[30]）= 0 来选择序列控制。

用户可通过配置 PORT_ADDR 字段（SAR_CHANx_CONFIG[6:4]）和 PIN_ADDR 字段（SAR_CHANx_CONFIG[2:0]）来配置 SARMUX 的选择，进而配置每个通道需要转换的信号，见下表。其他未使用的端口管脚是为 PSoC 4 系列其他产品保留的。

表 19-10. PORT_ADDR/PIN_ADDR 配置

PORT_ADDR	PIN_ADDR	SARADC 输入信号源
0	0..7	连接到 SARMUX 的 8 个专用管脚（P2.0-P2.7）
1	X	sarbus0*（连接到 SARADC 的正端）
1	X	sarbus1*（连接到 SARADC 的负端，仅在差分模式下有效）
7	0	温度传感器
7	2	AMUXBUS_A
7	3	AMUXBUS_B

*sarbus0 和 sarbus1 分别连接到 CTBm 模块中的运算放大器（opamp）0 和运算放大器 1 的输出，详情见 CTBm 章节。

NEG_SEL（SAR_CTRL [11:9]）控制 SARADC 负端输入信号在单端输入模式下的选择；在差分输入模式下，其不起作用。大多数情况下忽略 NEG_SEL 字段的配置，当选择单端模式和软件控制方法时，如果需要将负端连接到参考电压 Vref，请设置 NEG_SEL（SAR_CTRL [11:9]）= 7。

在差分模式下，SARADC 的负端连接依赖于正端连接，由 PORT_ADDR 和 PIN_ADDR 的值决定。当 DIFFERENTIAL_EN（SAR_CHANx_CONFIG[8]）= 1 时，差分转换模式被使能，通道将对奇/偶管脚对进行差分转换，此时 PIN_ADDR[0]值可被忽略。在序列控制下，P2.0/P2.1, P2.2/P2.3, P2.4/P2.5 和 P2.6/P2.7 分别是有效的差分对。当由软件或 DSI 控制，正负端的选择会更加灵活。

在单端模式下，NEG_SEL（SAR_CTRL [11:9]）控制 SARADC 负端输入信号在单端输入模式下的选择；在差分输入模式下，其不起作用。不同的 SARADC 负端连接选择，会影响到电压范围、信噪比（SNR）、有效分辨率。详情请见 19.3.1.4 负端输入选择章节。负端输入包括：VSSA, Vref, or P2.1, P2.3, P2.5 和 P2.7。需要注意的是，如果需要连接负端至 Vref，还需要设置 SAR_HW_CTRL_NEGVREF（SAR_CTRL[13]）= 1。因为寄存器 MUX_SWITCH_HW_CTRL 没有硬件控制位。

软件控制：默认情况下，对 SARMUX 的操作使用软件控制方法。用户可以通过设置 SAR_MUX_SWITCH0 [29:0] 中的相应位来控制 SARADC 的正负端输入信号。同时设置硬件开关控制寄存器的相应位为 0，即 SAR_MUX_SWITCH_HW_CTR[n] =0；否则，SARMUX 模拟路由仍由序列控制或 DSI 控制。

SWITCH_DISABLE 位（SAR_CTRL[30]）用来禁用 SARSEQ 中的 Sequencer, 从而禁用 SARMUX 中的路由开关；但是在软件控制方法中，SWITCH_DISABLE（SAR_CTRL[30]）位对路由开关不起作用，但建议将其设置为 1

NEG_SEL（SAR_CTRL [11:9]）控制 SARADC 负端输入信号在单端输入模式下的选择；在差分输入模式下，其不起作用。大多数情况下忽略 NEG_SEL 字段的配置，当选择单端模式和软件控制方法时，如果需要将负端连接到参考电压 Vref，请设置 NEG_SEL（SAR_CTRL [11:9]）= 7。

不同的 SARADC 负端连接选择，会影响到电压范围、信噪比（SNR）、有效分辨率。详情请见 19.3.1.4 负端输入选择章节。

19.3.10.2 SARSEQ 全局配置

全局配置是指，其配置参数在寄存器模式和 DSI 模式下均有效。

全局配置寄存器包括：SAR_CTRL、SAR_SAMPLE_CTRL、SAR_SAMPLE01、SAR_SAMPLE23、SAR_RANGE_THES 和 SAR_RANGE_COND。

在进行全局配置时，应遵循如下规则：全局配置仅可在两个扫描之间进行，配置的修改不应影响正在进行的扫描。否则，将导致正在进行的扫描转换结果不确定。

表 19-11. 全局配置寄存器

配置	配置寄存器	参考信息
参考电压选择	SAR_CTRL[6:4]	19.3.3.1 参考电压选项
转换结果的符号配置	SAR_SAMPLE_CTRL [3:2]	19.3.1.3 转换结果数据格式
数据左右对齐	SAR_SAMPLE_CTRL [1]	19.3.1.3 转换结果数据格式
单端模式下的负端输入选择	SAR_CTRL[11:9]	19.3.1.4 负端输入选择
分辨率	SAR_SAMPLE_CTRL[0]	19.3.1.5 分辨率
采样时间	SAR_SAMPLE_TIME01 [25:0] SAR_SAMPLE_TIME32 [25:0]	19.3.1.6 采样时间
求平均次数	SAR_SAMPLE_CTRL[7:4]	19.3.4.1 求平均
阈值检测	SAR_RANGE_THRES [31:0] SAR_RANGE_COND [31:30]	19.3.4.2 阈值检测

19.3.10.3 通道配置

通道配置包括：

- 单端 /差分模式选择
- 全局配置选择：采样时间、分辨率、求平均使能
- DSI 输出使能

通道配置应遵循如下规则：已加入扫描的使能通道配置的修改仅可在两个扫描周期间进行（与全局配置相同），否则转换结果将未知。如果使能通道的配置在扫描周期内被修改，则将影响采样速率。禁止通道配置的修改可在任意时刻进行，不影响正在进行的扫描；如果在扫描周期间禁止通道被使能，其配置仍可在下一个扫描周期开始前被修改，并在下一个扫描周期内有效。在一个扫描中，改变使能通道，将影响其采样速率。

有关通道配置的寄存器见下表

表 19-12. 通道配置寄存器

配置	寄存器	参考信息
单端模式/差分模式	SAR_CHANx_CONFIG [8]	19.3.1.1 单端模式和差分模式
采样时间选择	SAR_CHANx_CONFIG [13:12]	19.3.1.6 采样时间
分辨率选择	SAR_CHANx_CONFIG [9]	19.3.1.5 分辨率
求平均使能	SAR_CHANx_CONFIG [10]	19.3.4.1 求平均
DSI 输出使能	SAR_CHANx_CONFIG [30]	19.3.11.7 DSI 输出使能

SAR ADC 模块可通过配置 RESOLUTION 位（SAR_CHANx_CONFIG [9]）来选择使用主分辨率 12 位（默认）还是次分辨率 10 位/8 位。次分辨率又由 SUB_RESOLUTION 位（SAR_SAMPLE_CTRL[0]）控制。详情见下表

表 19-13. 分辨率配置

求平均	SUB_RESOLUTION（SAR_SAMPLE_CTRL[0]）	RESOLUTION（SAR_CHANx_CONFIG [9]）（寄存器模式）	通道分辨率
OFF	0	1	8 位
OFF	1	1	10 位
OFF	0	0	12 位
OFF	1	0	12 位
ON	X	X	12 位

19.3.10.4 通道使能

用户可通过配置寄存器 SAR_CHAN_EN 来使能每个通道。所有使能的通道将在下一个触发条件发生时被扫描。当扫描正在进行时，通道的使能状态可被更新，并在下一个扫描周期扫描有效，也就是说对通道使能的修改，不影响正在进行的扫描。但是当扫描正在进行时，已经在扫描周期内通道的其他配置不应被修改，否则转换结果将不确定。

19.3.10.5 中断屏蔽

SAR ADC 支持 6 种中断，每个中断均对应一个中断屏蔽位：

- 扫描转换完成中断
- 扫描结果溢出中断
- 冲突检测中断
- 插入通道转换结束中断
- 通道阈值检测中断
- 通道饱和检测中断

在中断请求寄存器（SAR_INTR，SAR_SATURATE_INTR 和 SAR_RANGE_INTR）、软件设置中断寄存器（SAR_INTR_SET，SAR_SATURATE_INTR_SET，SAR_RANGE_INTR_SET）、中断屏蔽寄存器（SAR_INTR_MASK，SAR_SATURATE_INTR_MASK，SAR_RANGE_INTR_MASK）和中断屏蔽结果寄存器（SAR_INTR_MASKED，SAR_SATURATE_INTR_MASKED，SAR_RANGE_INTR_MASKED）中，每个中断均有对应的位。此外，寄存器 SAR_INTR_CAUSE 中包含了 SARADC 当前挂起的中断。中断服务程序可以通过读此寄存器来辨别产生中断的来源。

关于中断的详细信息，请参考 19.3.5 中断章节。

19.3.10.6 触发

SARADC 支持 3 种触发方式，来启动模数转换：

- 软件触发：SAR_START_CTRL [0]
- DSI 触发：dsi_trigger
- 连续触发：SAR_SAMPLE_CTRL [16]

关于触发的详细信息，请参考 19.3.6 触发章节。

19.3.10.7 数据读取

每次扫描结束后，软件需及时从结果寄存器中读取转换结果；否则，转换结果可能被下次扫描的配置所改变。

8 个通道均使用双缓冲的 16 位寄存器来保持数据（插入通道不支持双缓冲）；双缓冲是通过工作寄存器 和结果寄存器来实现的。每个通道完成采样后，数据被存放在工作寄存器中；当完成对所有使能通道的扫描后，工作寄存器中的数据会被拷贝到结果寄存器中。

在扫描过程中，当某通道的工作寄存器中的数据有效时（此通道的采样已完成），寄存器 CHAN_WORK_VALID 中的相应位会被置 1。扫描完成后，寄存器 CHAN_RESULT_VALD 中的相应位会被置 1；同时，寄存器 CHAN_WORK_VALID 中相应位会被清除。

每个通道的寄存器 SAR_CHAN_WORK 的位 [31]与寄存器 SAR_CHAN_WORK_VALID 的相应位值相同；寄存器 SAR_CHAN_RESULT 的位[29]、位 [30]、位[31]位分别对应寄存器 SAR_SATURATE_INTR、SAR_RANGE_INTR 和 SAR_CHAN_RESULT_VALID 的相应位。需要注意的是此处仅与未屏蔽的中断相同。这样寄存器不但包含转换数据，还包括数据有效状态。

当 DSI 输出被使能（DSI_OUT_EN（SAR_CHANx_CONFIG[31]）= 1）时，处理后的结果数据会和通道序号一起通过 DSI 总线（sar_dsi_data, sar_dsi_chan_id）被发送到 UDB 做进一步的处理，紧接着也会被存放到结果寄存器；
这样 UDB 可对不同通道的数据做不同的处理。详情请见 19.3.11.7 DSI 输出使能章节。

19.3.10.8 插入通道转换（可选）

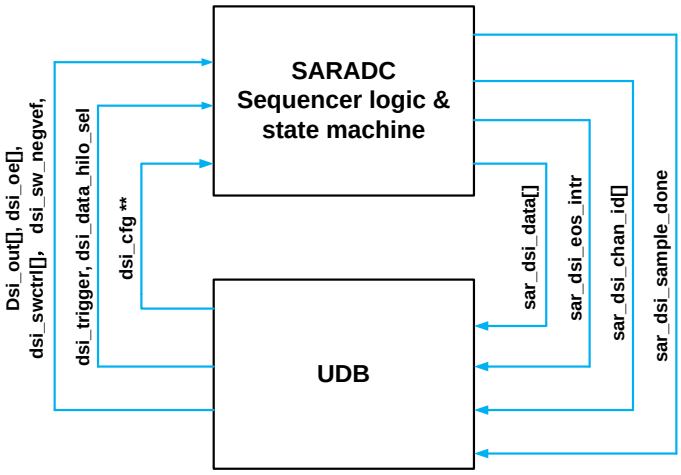
用户可以通过对 INJ_START_EN（SAR_INJ_CHAN_CONFIG [31]）位置 1 来触发对插入通道的采样、转换。如果有一个扫描正在进行，推荐通过设置 INJ_TAILGATING（SAR_INJ_CHAN_CONFIG [30]）使能追尾功能，则 INJ_START_EN 仅在当前扫描结束后才使能插入通道来进行采样转换，而不产生任何冲突中断。如果当前没有扫描进行（SARADC 处于空闲状态），并使能了追尾功能，则 INJ_START_EN 将等待至下一个扫描结束后才使能插入通道；在这种情况下，建议禁止追尾功能。

关于插入通道的详细信息，请参考 19.3.4.4 插入通道章节。

19.3.11 DSI 模式

在 DSI 模式下，对 SARADC 的配置可通过来自 UDB 的 DSI 信号完成（全局配置除外，例如，中断屏蔽、阈值检测设置和触发配置等）。DSI 模式和寄存器模式的主要不同是：DSI 模式允许硬件动态地控制 SARADC 的配置。下图描述了 SAR ADC 模块与 UDB 间的输入、输出信号。

图 19-13. SAR ADC 模块与 UDB 间的信号



用户可以通过设置 DSI_MODE 位（SAR_CTRL[29]），来选择使用 DSI 模式还是寄存器模式。在 DSI 控制模式下，SARSEQ 将忽略通道的所有寄存器配置，而由来自 DSI 的信号控制。来自 DSI 的信号可以配置 SAR ADC 模块，见下表。

表 19-14. DSI 信号

信号	信号宽度	描述
sar_dsi_sample_done	1	此脉冲信号表示 SARADC 已完成对当前通道采样；SARMUX 可以切换到下一个需要采样的信号。
sar_dsi_chan_id_valid	1	表示下面的通道序号信号有效
sar_dsi_chan_id	4	寄存器控制模式：当前正在转换的通道序号 DSI 控制模式：[0]=饱和检测中断，[1]=阈值检测中断（与输出数据同时有效）
sar_dsi_data_valid	1	表示下面的数据值信号有效
sar_dsi_data	12	对某个通道转换、求平均后的结果。内部求平均的结果是 16 位宽度。 当 dsi_data_hilo_sel=0 时，sar_dsi_data[11:0]= sar_data[11:0]。 当 dsi_data_hilo_sel=1 时，sar_dsi_data[7:0]= sar_data[15:8]，sar_dsi_data[11:8]=<未知>。
sar_dsi_eos_intr	1	扫描结束（EOS）中断信号，表示 SARSEQ 刚完成对所有使能通道的扫描
dsi_out	8	dsi_out[0]=1，连接 P2.0 至 SARADC dsi_out[1]=1，连接 P2.1 至 SARADC … dsi_out[7]=1，连接 P2.7 至 SARADC 注意：寄存器 MUX_SWITCH0 控制管脚连接到 SARADC 的正端还是负端
dsi_oe	4	dsi_oe[0]=1，连接 AMUXBUS_A 至 SARADC dsi_oe[1]=1，连接 AMUXBUS_B 至 SARADC dsi_oe[2]=1，连接 opamp0 输出至 SARADC dsi_oe[3]=1，连接 opamp1 输出至 SARADC 注意：寄存器 MUX_SWITCH0 控制这些信号连接到 SARADC 的正端还是负端
dsi_swctrl[0]	1	SARMUX 模拟开关控制信号，控制是否连接 vssa_kelvin 至 SARADC 的负端
dsi_swctrl[1]	1	SARMUX 模拟开关控制信号，控制是否连接温度传感器输出信号至 SARADC 的正端
dsi_sw_negvref	1	SARADC 内部切换控制信号，控制是否连接参考电压 VREF 至负端。
dsi_cfg_st_sel	2	DSI 控制模式下的配置控制信号：选择采样时间
dsi_cfg_average	1	DSI 控制模式下的配置控制信号：使能求平均
dsi_cfg_resolution	1	DSI 控制模式下的配置控制信号：0—主分辨率 12 位；1—次分辨率 8 位/10 位
dsi_cfg_differential	1	DSI 控制模式下的配置控制信号：0—单端模式；1—差分模式
dsi_trigger	1	启动 SARSEQ 对所有使能通道扫描的触发信号
dsi_data_hilo_sel	1	选择 sar_dsi_data[7:0]上是高 8 位还是低 8 位信号。此信号是异步信号。

19.3.11.1 SARMUX 模拟路由配置

在 DSI 模式下，可以通过 DSI 信号或软件来控制模拟路由。软件控制效果与寄存器模式下相同。关于软件控制方式的详细信息，请参考 19.3.10.1 SARMUX 模拟路由配置。

DSI 模式：在 DSI 模式下，使用来自 UDB 的 DSI 信号被用来控制 SARMUX 的开关切换。

但需要注意的是，在 DSI 控制模式下，SARSEQ 中的序列控制器对开关切换不起作用。如图 19-3 所示，除用于芯片设计和测试的开关外，DSI 可以控制所有开关。因此，在 DSI 模式下，SARADC 的正负端可以连接到任何开关。

下表罗列了 DSI 控制信号；同时还需设置 SAR_MUX_SWITCH_HW_CTR[n] = 1 & SAR_MUX_SWITCH0[n] = 1；当 SARADC 的负端连接到 Vref 时，还需要设置 SAR_CTRL [11:9] = 7 & SAR_CTRL [13] =1。

在 DSI 控制控制模式下，DSI 信号控制方式可通过信号 dsi_swctrl[0] 和 dsi_sw_negvref 控制 SARADC 的负端输入（在单端模式下）；如需设置 NEG_SEL（SAR_CTRL[11:9]），则仅 NEG_SEL（SAR_CTRL[11:9]）=7 有效。

表 19-15. DSI 信号控制

信号	信号宽度	描述
dsi_out	8	dsi_out[0]=1，连接 P2.0 至 SARADC dsi_out[1]=1，连接 P2.1 至 SARADC … dsi_out[7]=1，连接 P2.7 至 SARADC 注意：寄存器 MUX_SWITCH0 控制管脚连接到 SARADC 的正端还是负端
dsi_oe	4	dsi_oe[0]=1，连接 AMUXBUSA 至 SARADC dsi_oe[1]=1，连接 AMUXBUSB 至 SARADC dsi_oe[2]=1，连接 sarbus0 输出至 SARADC dsi_oe[3]=1，连接 sarbus1 输出至 SARADC 注意：寄存器 MUX_SWITCH0 控制这些信号连接到 SARADC 的正端还是负端
dsi_swctrl[0]	1	SARMUX 模拟开关控制信号，控制是否连接 vssa_kelvin 至 SARADC 的负端
dsi_swctrl[1]	1	SARMUX 模拟开关控制信号，控制是否连接温度传感器输出信号至 SARADC 的正端。
dsi_sw_negvref	1	SAR ADC 内部切换控制信号，控制是否连接参考电压 VREF 至负端。

19.3.11.2 SARSEQ 全局配置

SARSEQ 的全局配置适用于寄存器模式和 DSI 模式，请参考 19.3.10.2 SARSEQ 全局配置。

19.3.11.3 通道配置

在 DSI 控制模式下，只有通道 0 是可用的。通道 0 的配置是通过 DSI 信号来完成的，见下表。寄存器 SAR_CHAN_EN、SAR_CHAN_CONFIG 和 SAR_INJ_CHAN_CONFIG 中有关通道的配置在此模式下不起作用。

用户可通过配置 DSI_SYNC_CONFIG（SAR_CTRL[28]）位来选择是否需要 dsi_cfg_*信号来同步 SAR ADC 时钟域（主要指 clk_hf）。当 SARADC 运行于低频时，可以不用考虑同步 SARADC 时钟。

表 19-16. 配置通道 0 的 DSI 信号

信号	信号宽度	配置项	描述
dsi_cfg_st_sel	2	采样时间	DSI 控制模式下的配置控制信号：选择采样时间
dsi_cfg_average	1	使能求平均	DSI 控制模式下的配置控制信号：使能求平均
dsi_cfg_resolution	1	分辨率	DSI 控制模式下的配置控制信号：分辨率选择 0：主分辨率 12 位 1：次分辨率 8 位/10 位
dsi_cfg_differential	1	差分/单端模式	DSI 控制模式下的配置控制信号： 0：单端模式 1：差分模式

19.3.11.4 中断

中断屏蔽与在寄存器控制模式下相同。但需要注意的是，仅有中断 SATURATE_INTR, RANGE_INTR 和 EOS_INTR 会被通过 DSI 信号发出。如需 SARADC 中断的详细介绍，请参考 19.3.5 中断章节。

- 饱和检测中断 SATURATE_INTR 会通过信号 dsi_chan_id[0]被发出，并将寄存器 SATURATE_INTR 的位[0]置 1。
- 阈值检测中断 RANGE_INTR 会通过信号 dsi_chan_id[1]被发出，并将寄存器 RANGE _INTR 的位[0]置 1。
- 忽略通道使能，因为每次触发，仅对通道 0 进行转换。
- 扫描结束中断 EOS_INTR 总是通过 DSI 的信号 sar_dsi_eos_intr（dsi_data_valid 的拷贝）被发出。

通过 DSI 信号发出的中断见下表。

表 19-17. 通过 DSI 信号发出的中断

信号	信号宽度	描述
sar_dsi_chan_id	4	寄存器控制模式：当前正被转换的通道序号 DSI 控制模式： sar_dsi_chan_id [0] = 饱和检测中断， sar_dsi_chan_id [1]= 阈值检测中断（两个中断信号与有效数据会被一起发出）
sar_dsi_eos_intr	1	扫描结束中断，表示 SARSEQ 刚完成对所有使能通道的扫描

19.3.11.5 触发

在 DSI 控制模式下，通常使用 DSI 触发模式，但也支持软件触发和连续触发。触发配置与寄存器控制模式下相同，详情请见 19.3.6 触发章节。

如果使用 DSI 触发方式，每个通道的配置信号（dsi_cfg_*）和 SARMUX 的配置信号应该在信号 dsi_trigger 被发出前达到稳定状态，并一直保持到信号 sar_dsi_sample_done 的上升沿出现。

19.3.11.6 数据读取

SARADC 的转换结果和通道编号会通过 sar_dsi_data 被发送出去，等效于寄存器模式下 dsi_out_en 为高的情况。详情请参考 19.3.11.7 DSI 输出使能章节。每个通道的转换结果均会被写入寄存器 SAR_CHAN_WORK0 和 SAR_CHAN_RESULT0。

19.3.11.7 DSI 输出使能

当 DSI_OUT_EN（SAR_CHANx_CONFIG[31]）= 1 时，处理后的结果数据会和通道序号一起通过 DSI 总线（sar_dsi_data, sar_dsi_chan_id）被发送到 UDB 做进一步的处理，紧接着也会被存放到结果寄存器；这样 UDB 可对不同通道的数据做不同的处理。

发送到 DSI 总线上的数据与存放在结果寄存器中的数据格式完全相同。默认情况下，仅有低 12 位数据被发送，即 sar_dsi_data[11:0]= sar_data[11:0]，因此不建议用户使用数据左对齐。如需要结果寄存器中的 16 位数据，首先设置 dsi_data_hilo_sel=0，获得低 12 位数据，即 sar_dsi_data[11:0]= sar_data[11:0]；然后设置 dsi_data_hilo_sel=1，获得高 8 位数据，即 sar_dsi_data[7:0]= sar_data[15:8]；同时需要对获得的数据进行处理才可以组成 16 位的数据。

一旦 SARADC 对某通道完成采样，通道序号就会通过信号 sar_dsi_chan_id 被发送出去（也就是说，在转换开始时即被发送）。这样做有利于某些特殊的应用，例如，通道序号可以触发 UDB 来驱动一些 GPIO 管脚，这些管脚可以依次对一些外部设备进行上电（或断电）操作；而这些外设又依次驱动模拟输入管脚并被同一个扫描周期内的其他通道采样、转换（此处的采样时间应较长）。

注意：数据将在转换结束一个时钟周期后被发送到 DSI 总线上。通道序号、数据及相应的有效信号将在 DSI 总线上保持两个系统时钟周期。

DSI 输出信号见下表。

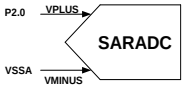
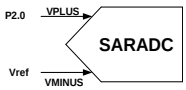
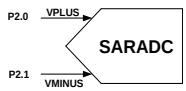
表 19-18. DSI 输出信号

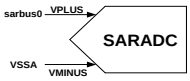
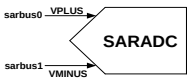
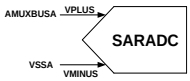
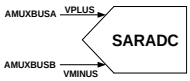
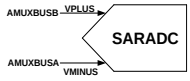
信号	信号宽度	描述
sar_dsi_sample_done	1	此脉冲信号表示 SARADC 采样结束。SARMUX 可以切换到下一个需要转换的信号。
sar_dsi_chan_id_valid	1	通道序号是有效信号。
sar_dsi_chan_id	4	寄存器控制模式： 指示当前正被转换的通道序号。 DSI 控制模式： [0]：饱和检测中断 [1]：阈值检测中断 （两个中断信号与数据输出同时有效）
sar_dsi_data_valid	1	数据值信号是有效信号
sar_dsi_data	12	单个通道的转换（和求平均）结果。内部求平均的结果是 16 位宽度。 当 dsi_data_hilo_sel=0，sar_dsi_data[11:0]= sar_data[11:0]。 当 dsi_data_hilo_sel=1，sar_dsi_data[7:0]= sar_data[15:8] & sar_dsi_data[11:8]=<未知>。
sar_dsi_eos_intr	1	扫描结束（EOS）中断信号表示 SARSEQ 刚对所有使能通道完成扫描。
dsi_data_hilo_sel	1	表示输出到 sar_dsi_data[7:0]的数据是高 8 位还是低 12 位信号。此信号是异步的（其对 sar_dsi_data[7:0]的控制不需要时钟参与）。

19.3.12 模拟路由配置示例

下表描述了在序列控制、软件控制和 DSI 控制下配置管脚/信号的示例。

表 19-19. 管脚/信号配置示例

	序列控制	软件控制	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 **NEG_SEL = 7**（CTRL [11:9]） **HW_CTRL_NEGVREF =1** （CTRL[13]） dsi_out [0] =1 dsi_sw_negvref =1
	**DIFFERENTIAL_EN = 1** （CHANx_CONFIG[8]） **SWITCH_DISABLE = 0**（CTRL[30]） **PORT_ADDR = 0** （CHANx_CONFIG[6:4]） **PIN_ADDR = 0 or PIN_ADDR = 1**	**DIFFERENTIAL_EN = 1** （CHANx_CONFIG[8]） **SWITCH_DISABLE = 1** （CTRL[30]） MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1	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

	序列控制	软件控制	DSI 控制
	（CHANx_CONFIG[2:0]） MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[1] = 1	MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[1] = 0	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：序列控制不支持 sarbus1 与 SARADC 正端的连接**	**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
	**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，在序列控制方式下，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

19.3.13 温度传感器配置

片上温度传感器可用于温度检测和基于温度的校准应用。当选择温度传感器信号作为 SAR ADC 模块的输入信号时，仅可使用单端模式，且参考电压应来自内部的 1.024V。

PSoC 4 支持通过 3 种控制方式将温度传感器信号切换至 SARADC 的输入端，见下表。

注意：设置 MUX_FW_TEMP_VPLUS 位（SAR_MUX_SWITCH0[17]），可以使能温度传感器并将其输出连接到 SARADC 的正端；清除该位将通过切断偏置电流来禁止温度传感器

表 19-20. 温度传感器信号输入至 SARADC 的配置

控制方式	寄存器配置
序列控制	VREF_SEL = 0（SAR_CTRL[6:4]） NEG_SEL = 0（SAR_CTRL [11:9]）* SWITCH_DISABLE = 0（SAR_CTRL[30]） PIN_ADDR = 0（SAR_CHANx_CONFIG[2:0]） PORT_ADDR = 7（SAR_CHANx_CONFIG[6:4]） DIFFERENTIAL_EN = 0（SAR_CHANx_CONFIG[8]） SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 1 SAR_MUX_SWITCH_HW_CTRL[17]= 1
软件控制	VREF_SEL = 0（SAR_CTRL[6:4]） NEG_SEL = 0（SAR_CTRL [11:9]）* SWITCH_DISABLE = 1（SAR_CTRL[30]） DIFFERENTIAL_EN = 0（SAR_CHANx_CONFIG[8]） SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 0 SAR_MUX_SWITCH_HW_CTRL[17]= 0
DSI 控制	VREF_SEL = 0（SAR_CTRL[6:4]） NEG_SEL = 0（SAR_CTRL [11:9]）* SWITCH_DISABLE = 1（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 **设置 DSI 信号：** dsi_cfg_differential=1 dsi_swctrl[1]=1 dsi_swctrl[0]=1

*对于温度传感器输入信号：NEG_SEL（SAR_CTRL[11:9]总为 0。

19.4 寄存器列表

寄存器名称	偏移地址	数量	数据位宽度	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，可配置采样时间 ST0 和 ST1
SAR_SAMPLE_TIME23	0x0014	1	32	全局配置寄存器，可配置采样时间 ST2 和 ST3
SAR_RANGE_THRES	0x0018	1	32	全局配置寄存器，阈值检测寄存器
SAR_RANGE_COND	0x001C	1	32	全局配置寄存器，阈值检测模式寄存器
SAR_CHAN_EN	0x0020	1	32	通道使能寄存器
SAR_START_CTRL	0x0024	1	32	开启对使能通道的扫描（由软件触发），扫描结束后硬件将对其清除。
SAR_CHAN_CONFIG	0x0080	8	32	通道配置寄存器
SAR_CHAN_WORK	0x0100	8	32	保存相应通道采样后的数据
SAR_CHAN_RESULT	0x0180	8	32	保存相应通道转换后的数据
SAR_CHAN_WORK_VALID	0x0200	1	32	指示某通道的工作寄存器数据是否有效，也就是某通道是否在此次扫描周期中已经完成采样
SAR_CHAN_RESULT _VALID	0x0204	1	32	指示某通道的结果寄存器数据是否有效，也就是某通道是否已在上个扫描周期完成处理。
SAR_STATUS	0x0208	1	32	指示 SARADC 当前的状态（用于调试）
SAR_AVG_STAT	0x020C	1	32	当前求平均的状态（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	屏蔽结果寄存器：如果此寄存器的值非 0，则发送到 NVIC 的 SARADC 的中断信号为高。其中存放的值为中断请求与中断屏蔽寄存器的位与逻辑值。
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	饱和中断屏蔽结果寄存器
SAR_RANGE_INTR	0x0230	1	32	阈值检测中断请求寄存器
SAR_RANGE_INTR_SET	0x0234	1	32	阈值检测中断请求设置寄存器
SAR_RANGE_INTR_MASK	0x0238	1	32	阈值检测中断屏蔽寄存器
SAR_RANGE_INTR_MASKED	0x023C	1	32	阈值检测中断屏蔽结果寄存器
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 软件控制开关寄存器
SAR_MUX_SWITCH_CLEAR0	0x0304	1	32	清除 SARMUX 软件控制开关寄存器
SAR_MUX_SWITCH_HW _CTRL	0x0340	1	32	SARMUX 硬件控制开关寄存器
SAR_MUX_SWITCH_STATUS	0x0348	1	32	SARMUX 开关状态寄存器
SAR_PUMP_CTRL	0x0380	1	32	电荷泵控制寄存器

20 低功耗比较器

PSoC® 4 的数字/休眠电源域（Digital/Hibernate power domain）中有两个低功耗比较器（Low Power Comparator）。在低功耗模式下，当模拟系统被关闭时，比较器仍可提供快速检测输入信号的能力。比较器的正负输入端可以分别连接到专用 GPIO 或 AMUXBUS_A/AMUXBUS_B；对低功耗比较器的输出信号可做如下处理：

- CPU 读取并做进一步处理
- 作为中断/唤醒信号（在活动/睡眠/深度睡眠/休眠模式下）
- 送至 DSI 做进一步处理

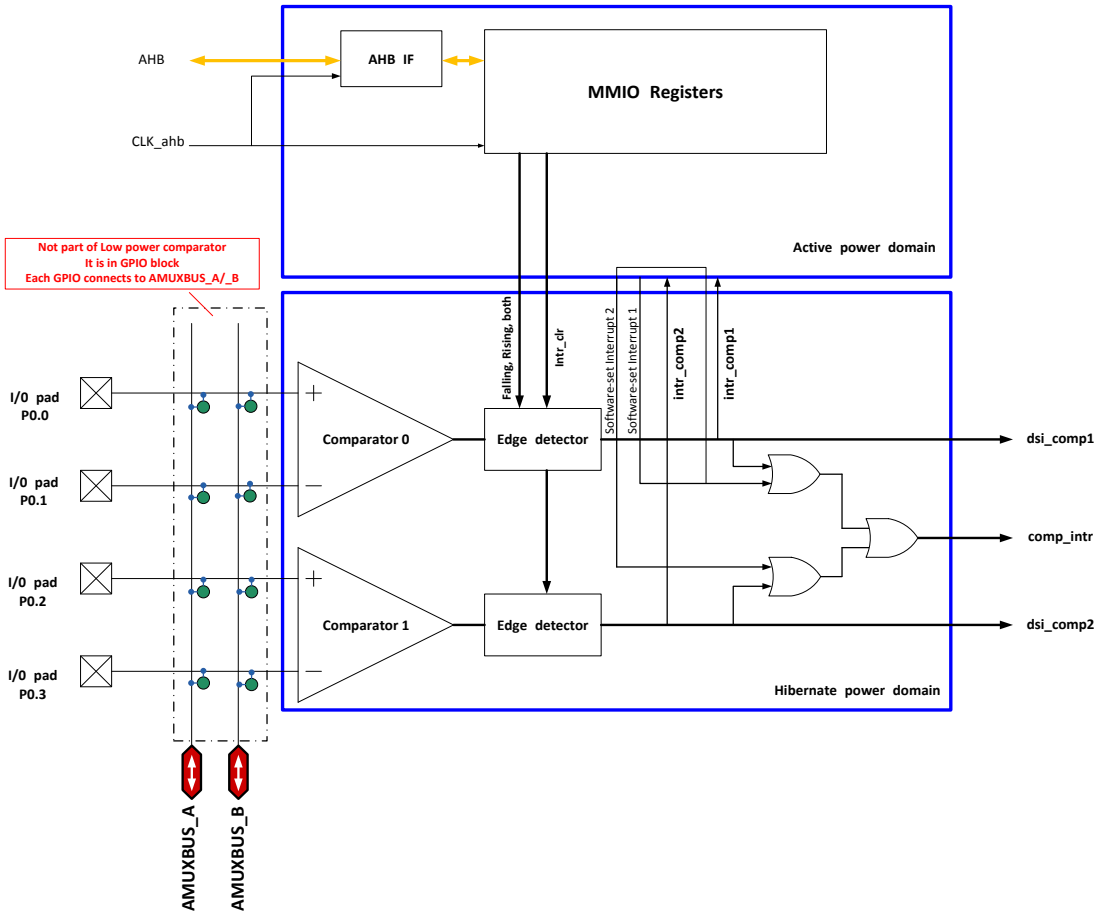
20.1 特性

PSoC® 4 低功耗比较器具有如下特性：

- 输入连接可配置
- 电源和速度可配置
- 支持 3 种低功耗模式（快速模式、低速模式和超低功耗模式）
- 可选的 10 mV 输入迟滞
- 较低的输入失调电压（校正后输入失调电压<4 mV）
- 比较器输出可作为唤醒信号（在睡眠/深度睡眠/休眠模式下）

20.2 低功耗比较器框图

图 20-1. PSoC 4 低功耗比较器框图



20.3 低功耗比较器的使用

下面将从输入配置、电源和速度模式、输出和中断配置、迟滞、休眠唤醒、比较器时钟和失调电压校正等方面介绍如何使用 PSoC 4 低功耗比较器。

20.3.1 输入配置

如下信号可以分别连接到低功耗比较器的正负输入端：

- 两个外部管脚的输入电压
- 一个外部管脚的输入电压和一个来自 AMUXBUS_A/AMUXBUS_B 的内部信号。这两个输入信号均可以连接到低功耗比较器的正端或负端。
- 两个来自 AMUXBUS-A/AMUXBUS-B 的内部信号

如图 20-1 所示，管脚 P0.0/P0.1/P0.2/P0.3 均可以直接连接到低功耗比较器的输入端。如果低功耗比较器的输入端通过 AMUXBUS 连接到其他管脚，则相应的管脚 P0.0/P0.1/P0.2/P0.3 仍然会被占用，不可用作其他功能。如何配置 GPIO 连接到 AMUXBUS_A/AMUXBUS_B，请参考输入/输出系统章节。

20.3.2 电源模式和速度配置

2 个低功耗比较器均可以在 3 种电源模式下工作：快速模式、低速模式和超低功耗模式。用户可以通过 MODE1 位（LPCOMP_CONFIG[1:0]）和 MODE2 位（LPCOMP_CONFIG[9:8]）来分别配置低功耗比较器 0 和低功耗比较器 1 的电源模式。但是，在电源模式改变的瞬间，比较器的输出可能会有毛刺。

不同电源模式，具有不同的反应时间和功耗。其中，快速模式的功耗最大，超低功耗模式最小。如需详细的功耗及反应时间值，请参考数据手册。

20.3.3 输出和中断配置

低功耗比较器 0 和低功耗比较器 1 的输出值保存在 OUT1 位（LPCOMP_CONFIG[6]）和 OUT2（LPCOMP_CONFIG[14]）。如图 20-1 所示，比较器的输出经过边沿检测电路和逻辑处理，形成了 3 个信号：dsi_comp1、dsi_comp2 和 comp_intr。用户可通过寄存器 LPCOMP_CONFIG 的 INTTYPE 位来配置中断触发方式（禁止/上升沿/下降沿/上升沿和下降沿触发）。

当 PSoC4 处于活动模式或睡眠模式时，信号 dsi_comp1/dsi_comp2 可通过 UDB 中的 DSI 被路由至 GPIO 或其他模块，DSI 输出可以选择是否使用同步器。例如在 PWM 的控制应用中，用户可以使用 dsi_comp1/dsi_comp2 来关断 PWM 模块的输出信号，是否同步可在 UDB 中选择。如果信号 dsi_comp1/dsi_comp2 被路由至 UDB 作进一步处理，则所需时间将因用户的算法及同步器的选择不同而不同。

当 PSoC4 处于在深度睡眠或休眠模式时，因 UDB 电源被关闭，信号 dsi_comp1/dsi_comp2 不能被路由。

两个低功耗比较器输出的信号（intr_comp1 / intr_comp2）相或后成为中断信号 comp_intr，该信号与 CLK_ahb 同步；比较器的清中断信号（Intr_clr）是异步的（见图 20-1）。寄存器 LPCOMP_INTR 的 COMP1 和 COMP2 位分别 是低功耗比较器 1 和低功耗比较器 2 的中断请求标志位；用户可通过寄存器 LPCOMP_INTR_SET 的 COMP1 和 COMP2 位对中断进行软件调试。

在低功耗模式下，比较器的输出翻转事件可激活中断唤醒控制器（WIC），进而唤醒 CPU。因此，即使在低功耗模式下，用户仍可用低功耗比较器来监控一些特殊信号。

20.3.4 迟滞

在比较器的应用中，两个输入信号大小可能比较接近，此时如果有噪声，易引起比较器的输出翻转。采用迟滞可以很好的避免此现象。

用户可以通过设置迟滞使能位 HYST1 位（LPCOMP_CONFIG[2]）和 HYST2 位（LPCOMP_CONFIG[10]）来分别使能低功耗比较器 0 和低功耗比较器 1 的迟滞（10mV）。

20.3.5 休眠唤醒

低功耗比较器可以在睡眠、深度睡眠和休眠等低功耗模式下运行，其输出的中断信号可将器件从睡眠、深度睡眠或休眠等低功耗模式下唤醒，不需要专门的设置。在深度睡眠或休眠模式下，比较器 0 或比较器 1 的输出经边沿检测电路可产生中断；但触发方式与寄存器 LPCOMP_CONFIG 的 INTTYPE 位无关。

20.3.6 比较器时钟

如图 20-1 所示，比较器使用系统主时钟 CLK_ahb 作为中断的同步信号。

20.3.7 失调电压校正

在出厂时，低功耗比较器的输入失调电压已经被校正到小于 4 mV。比较器的输入失调电压依赖于共模输入电压。对失调电压的校正分为两步：首先在共模输入电压为 0.1V 时校正；其次在共模输入电压位 Vdd - 0.1 V 时校正。小于 10.0mV 的输入失调电压远远小于 0.1V~Vdd-0.1V 的输入电压工作范围。因此在正常的应用中，用户一般无需考虑校正。

PSoC 4 支持用户在一个特定的共模输入电压下对失调电压进行校正。用户可通过配置如下寄存器来校正失调电压：

- 寄存器 LPCOMP_TRIM1 和 LPCOMP_TRIM2 用来校正低功耗比较器 0 的失调电压
- 寄存器 LPCOMP_TRIM3 和 LPCOMP_TRIM4 用来校正低功耗比较器 1 的失调电压。

如果期望较小的输入失调电压，可借助 CAL_EN 位（LPCOMP_DFT[0]）来完成。

用户校正比较器输入失调电压的方法如下（以低功耗比较器 0 为例，将失调电压调制为小于 1mV）：

1. 配置 CAL_EN 位（LPCOMP_DFT[0]）来短接比较器的正负输入端
2. 分别设置与低功耗比较器相连接的两个输入管脚至期望值。
3. 更改寄存器 LPCOMP_TRIM1 的设置：
   a. 根据测量的失调电压极性，设置或清除 LPCOMP_TRIM1[4]
   b. 增加 LPCOMP_TRIM1[3:0]的值，直至测量的偏置小于 1 mV。
4. 如果测量的失调电压极性已经改变，但是失调电压仍然大于 1 mV，请使用 LPCOMP_TRIM2[3:0]来微调失调电压值。这种调试方式仅对比较器在低速模式下有效。
5. 如果 LPCOMP_TRIM1[3:0]的值是 0Fh,测量的失调电压值仍然大于 1 mV，请根据测量的失调电压极性设置或清除 LPCOMP_TRIM2[3]，并增加 LPCOMP_TRIM2[2:0]的值，直至测量的失调电压小于 1 mV。

20.4 寄存器列表

寄存器	描述
LPCOMP_ID	LPCOMP_ID 寄存器包括了低功耗比较器的 ID 及修订编号等信息
LPCOMP_CONFIG	低功耗比较器的配置寄存器
LPCOMP_INTR	低功耗比较器的中断配置寄存器
LPCOMP_INTR_SET	低功耗比较器的中断设置寄存器（可用于软件调试）
LPCOMP_DFT	低功耗比较器的 DFT 寄存器
LPCOMP_TRIM1	比较器 1 校正字段
LPCOMP_TRIM2	比较器 1 校正字段
LPCOMP_TRIM3	比较器 2 校正字段
LPCOMP_TRIM4	比较器 2 校正字段

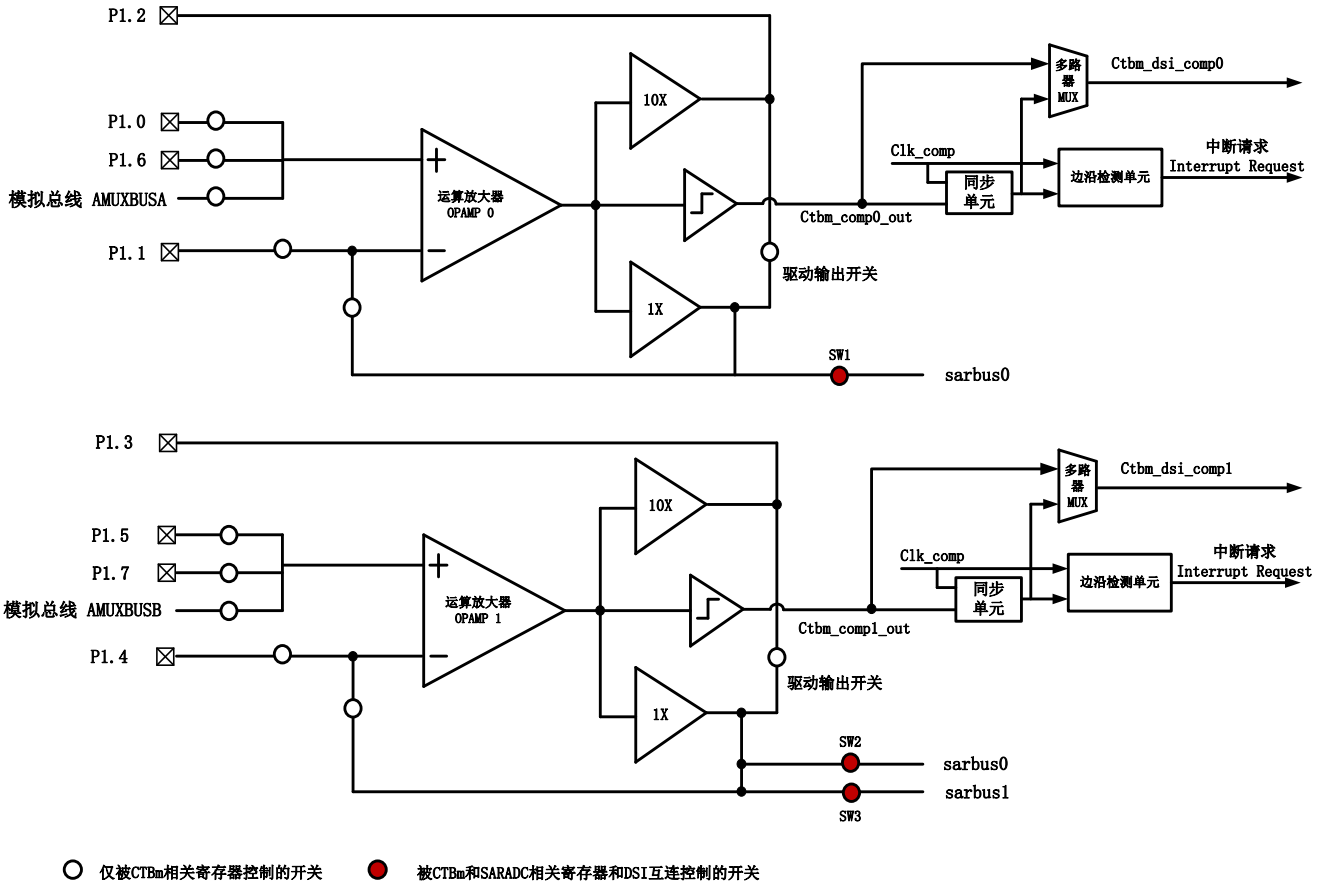
21 CTBm

CTBm（mini Continuous-Time Block）是 PSoC 4 提供的一个模拟功能模块，它包含了一个开关矩阵、两个相同的运算放大器（可配置为比较器）和一个数字接口，其中每个运算放大器中都集成了一个电荷泵。与 CTB（Continuous-Time Block）相比，CTBm 简化了一些内置电阻与开关。

21.1 特性

- 高度可配置的运算放大器：可配置速度，功耗，驱动能力和补偿；
- 通过内部开关，每个运算放大器可被配置为一个电压跟随器；
- 每个运算放大器可配置为比较器，带 10mV 的迟滞
- 最大驱动能力为 10 mA
- 容性负载为 20pF 时，增益带宽为 4MHz
- 失调电压可被调整到 1mV 以下
- 轨到轨的输入输出动态范围
  - 驱动 1mA 的负载时，动态范围为 Vss+0.2V 到 Vdda-0.2V；
  - 驱动 10mA 的负载时，动态范围为 Vss+0.5V 到 Vdda-0.5V；
- 容性负载为 50pF 时，输出摆率为 4V/us

21.2 模块框图



注意：同一时间，一个运放后两个驱动器（一个驱动能力为10mA；另一个为1mA）仅有一个可被使能。

21.3 工作原理

如模块框图所示，CTBm 主要有两个相同的运算放大器和一个开关矩阵组成。每个运算放大器有 1 个输入级和 3 个输出级，同一时刻，只有一个输出级能够工作。通过选择不同输出级，运算放大器可以工作为一个 A 类运算放大器，AB 类运算放大器或者比较器。CTBm 其它可配置的属性还包括功耗，速度，补偿和开关矩阵控制。

在使用 CTBm 模块时，首先需要安装好必需的外围器件，如电阻。通过将寄存器 CTBm_CTRL[31]置位，可以使能 CTBm 模块。为了获得轨到轨的输入输出动态范围，并保证运算放大器的线性工作特性，需要将运算放大器内部的电荷泵使能。具体的寄存器操作是将 CTBm_OA_RES0_CTRL[11]和 CTBm_OA_RES1_CTRL[11]置位，分别将运算放大器 0 和运算放大器 1 中的电荷泵使能。

完成上述的使能后，遵循下列步骤完成所有的配置：

1. 配置电源模式；
2. 配置输出的驱动能力；
3. 配置输入开关；
4. 配置输出开关，尤其当运放的输出需要被连接到 SARADC 时；
5. 如果需要，将运放配置为比较器的工作模式。

21.3.1 电源模式配置

通过减小进入运算放大器的参考电流，CTBm 可以降低功耗。通过配置寄存器 CTBM_OA_RESx_CTRL[1:0]可以选择运放的电源模式，共低、中、高三种电源模式可供选择。在低电源模式中，运放有最低的功耗，但输出摆率和工作带宽等特性也较差；在高电源模式中，则相反。电源模式的选择会对运算放大器的输出驱动能力有影响，详见表 21-1。器件手册具体给出了不同电源模式下，器件的功耗、输出摆率、工作带宽等特性。

21.3.2 驱动能力配置

每个运算放大器的输出级均可配置为内部驱动输出级（A 类/1X）或者外部驱动输出级（AB 类/10X）。同一个时刻，一个运算放大器只有一个输出级能够被使能。1X 输出级适合用于驱动较小的片上负载；10X 输出级适合用于驱动较大的片外负载。如模块框图所示，1X 输出级连接到 sarbus0 和 sarbus1，用于驱动 SARADC 的输入级；10X 输出级连接到了芯片的管脚上，用于驱动片外负载。每个输出级在不同电源模式下的驱动能力有所变化，具体请参见表 21-1。

表 21-1. 电源模式与驱动能力的关系

	电源模式 CTBm_OA_RESx_CTRL[1:0]			
	00 (禁止)	01 (低)	10 (中)	11 (高)
外部驱动输出级 (10X)	关闭	10mA	10mA	10mA
内部驱动输出级 (1X)	关闭	100uA	400uA	1mA

通过配置寄存器 OAx_DRIVE_STR_SEL（CTB_OA_RESx_CTRL[2]），可以选择哪一个驱动器被使能。如果在一些特殊的应用场合，需要 1mA 的驱动器驱动片外负载，或者 10mA 驱动器驱动片内负载，则需要额外将寄存器 CTBM_OAx_SW [21]置位，使驱动输出开关闭合。此时，整个模块的性能会有所下降。

每个运算放大器有一个可编程控制的补偿电容。当负载发生改变时，可以通过调整补偿电容来优化运算放大器的性能。补偿电容的配置寄存器为 CTBm_OAx_COMP_TRIM（x= 0 或 1），具体如表 21-2 所示。注意，datasheet 中增益带宽和输出摆率在任何一种补偿电容调整的情况下均能够达到。

表 21-2. 补偿电容的配置寄存器

CTBm_OAx_COMP_TRIM[1:0]	描述
00	最小补偿，高速，低稳定性
01	中度补偿，平衡的速度和稳定性
11	最大补偿，低速，高稳定性

21.3.3 开关配置

如模块框图所示，CTBm 中集成了很多开关，其中大部分的开关通过 CTBm 的相关寄存器（CTBM_OA0_SW，CTBM_OA1_SW）配置。但是有三个开关（输出开关 SW1、SW2 和 SW3）的配置不仅取决于 CTBm 的相关寄存器，还取决于 SARADC 的相关寄存器和 DSI 的互连。

将寄存器 CTBM_OAx_SW 置位，能够将相应的开关配置为闭合状态。将寄存器 CTBM_OAx_SW_CLEAR 置位令寄存器 CTBM_OAx_SW 被清零，从而将相应的开关配置为断开状态。

21.3.3.1 输入开关控制

通过控制开关，运放的正负输入端能够被连接到芯片的管脚。另外每个运放的正输入端还能够通过开关直接连接到芯片内部的模拟总线，其中运放 Opamp0 的正输入端可直接连到 AMUXBUS A；运放 Opamp1 的正输入端可直接连到 AMUXBUS B。

注意：在使用中应当确保同一时刻下，一个运放输入端的多个开关中只有一个被闭合，否则同时被闭合的开关连接的输入源将被短路。

- 正输入端

运放Opamp0和Opamp1的正输入端通过开关可以分别连接到三个输入源，如表 21-3所示。

表 21-3

	正向输入	开关控制位	开关状态
运放 Opamp0	AMUXBUSA	CTBM_OA0_SW [0]	0: 断开 1: 闭合
	P1.0	CTBM_OA0_SW [2]	0: 断开 1: 闭合
	P1.6	CTBM_OA0_SW [3]	0: 断开 1: 闭合
运放 Opamp1	AMUXBUSB	CTBM_OA1_SW [0]	0: 断开 1: 闭合
	P1. 5	CTBM_OA1_SW [1]	0: 断开 1: 闭合
	P1.7	CTBM_OA1_SW [4]	0: 断开 1: 闭合

■ 负输入端

运放 Opamp0 和 Opamp1 的负输入端通过开关可以分别连接到两个输入源，当连接到运放的输出时，为运放提供了一个反馈回路，如表 21-4 所示。

表 21-4

	负向输入	开关控制位	开关状态
运放 Opamp0	P1.1	CTBM_OA0_SW [8]	0: 断开 1: 闭合
	1mA 驱动器的输出	CTBM_OA0_SW [14]	0: 断开 1: 闭合
运放 Opamp1	P1.4	CTBM_OA1_SW [8]	0: 断开 1: 闭合
	1mA 驱动器的输出	CTBM_OA1_SW [14]	0: 断开 1: 闭合

21.3.3.2 输出开关控制

运放的输出通过 10mA 驱动器直接被连接到固定的管脚（Opamp0：P1.2；Opamp1：P1.3），这不需要任何配置。通过配置开关 SW1、SW2 和 SW3，运放的输出还可以被连接到 SARADC 的模拟输入总线 sarbus0 或者 sarbus1，如模块框图所示。这三个开关的配置取决于 CTBm 相关寄存器、SARADC 相关寄存器和 DSI 互连，具体控制逻辑如下表所示，其中 PORT_ADDR，PIN_ADDR 和 DIFFERENTIAL_EN 分别来自于 SAR_CHANx_CONFIG [6:4]，SAR_CHANx_CONFIG [2:0]和 SAR_CHANx_CONFIG [8]，它们中任何一个被设置为 0 都会使相应的开关被断开；当使用 SARADC 相关寄存器或 DSI 互连来配置开关时，CTB_SW_HW_CTRL bit [2]或 CTB_SW_HW_CTRL bit [3]必须被置位；CTB_OAx_SW[18]或 CTB_OAx_SW[19]可以屏蔽其他其他寄存器位的控制；SW1、SW2 和 SW3 分别是 CTBM_SW_STATUS [30:28]中的三位，它们记录了对应开关的当前状态，0 表示断开，1 表示闭合。

表 21-5 开关 SW1 控制逻辑的真值表

PORT_ADDR	PIN_ADDR	CTB_SW_HW_CTRL[2]	dsi_out[2]	CTB_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 21-6 开关 SW2 控制逻辑的真值表

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTB_SW_HW_CTRL[3]	dsi_out[3]	CTB_OA1_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 21-7 开关 SW3 控制逻辑的真值表

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTB_SW_HW_CTRL[3]	dsi_out[3]	CTB_OA1_SW[19]	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

21.3.4 比较器工作模式

通过将寄存器 CTBM_OA_RESx_CTRL[4]置位，可以配置相应的运放工作在比较器模式。在这种模式下，有如下特性：

- 10mV 的迟滞范围，可选
- 三种电源模式可供选择
- 比较器的输出连接到 DSI，可被同步
- 失调电压可被调整到 1mV 以下
- 多种中断触发方式

21.3.4.1 比较器的配置

将寄存器 CTBM_OA_RESx_CTRL[5]置位能够使能比较器的 10mV 迟滞特性。比较器共有三种电源模式，包括低、中和高。通过配置寄存器 CTBM_OA_RESx_CTRL[1:0]，可以选择具体工作在何种电源模式。在不同的电源模式中，比较器的动态响应、功耗等互不相同，具体请参考相应的器件手册。

如模块框图所示，比较器的输出连接到 DSI，可以被系统时钟同步，是否进行同步可以通过寄存器 CTBM_OA_RESx_CTRL[6]设置。

比较器的当前状态被存储在寄存器 CTBM_COMP_STAT 中。

21.3.4.2 比较器的中断

比较器有四种中断触发模式：禁止、上升沿、下降沿和双沿，通过配置寄存器 CTBM_OA_RESx_CTRL[9:8]，可以选择具体的中断触发方式。一旦某个比较器触发了中断，寄存器 CTBM_INTR 中相应的位会被置位，记录此中断请求。

将寄存器 CTBM_INTR_MASK 的某一位清零，相应的中断请求被屏蔽，则该请求不会被发送至嵌套向量中断控制器（NVIC: Nested Vectored Interrupt Controller）。为方便软件设计，CTBM_INTR 和 CTBM_INTR_MASK 的位与结果被记录在寄存器 CTBM_INTR_MASKED 中，如果其中某位为 1，说明相应的比较器触发了中断，并且该中断未被屏蔽，会被发送到 NVIC 进行处理。

通过软件向寄存器 CTBM_INTR 的某一位写 1，可以清除相应的中断请求。

另外为方便调试，CTBm 还提供了一个寄存器 CTBM_INTR_SET，通过软件人为的将该寄存器中的某位置位，可以人为的使相应的比较器触发中断。

21.4 寄存器列表

表 21-8

偏移地址	比特数	名称	描述
0x0000	32	CTBm_CTRL	CTBm 模块的全局控制寄存器
0x0004	32	CTBm_OA_RES0_CTRL	运放 Opamp0 的控制寄存器
0x0008	32	CTBm_OA_RES1_CTRL	运放 Opamp1 的控制寄存器
0x000C	32	CTBm_COMP_STAT	比较器的状态寄存器
0x0020	32	CTBm_INTR	中断请求寄存器
0x0024	32	CTBm_INTR_SET	中断请求置位寄存器
0x0028	32	CTBm_INTR_MASK	中断请求屏蔽寄存器
0x002C	32	CTBm_INTR_MASKED	中断请求屏蔽结果寄存器
0x0030	32	CTBm_DFT_CTRL	模拟可测试性设计（DFT: Design For Test）控制寄存器
0x0080	32	CTBm_OA0_SW	运放 Opamp0 的开关控制寄存器
0x0084	32	CTBm_OA0_SW_CLEAR	运放 Opamp0 的开关控制清零寄存器
0x0088	32	CTBm_OA1_SW	运放 Opamp1 的开关控制寄存器
0x008C	32	CTBm_OA1_SW_CLEAR	运放 Opamp1 的开关控制清零寄存器
0x00C0	32	CTBm_SW_HW_CTRL	CTBm 模块硬件控制使能寄存器
0x00C4	32	CTBm_SW_STATUS	CTBm 模块总线开关控制寄存器
0x0F00	32	CTBm_OA0_OFFSET_TRIM	运放 Opamp0 的失调电压调整寄存器
0x0F04	32	CTBm_OA0_SLOPE_OFFSET_TRIM	运放 Opamp0 的调整控制寄存器
0x0F08	32	CTBm_OA0_COMP_TRIM	运放 Opamp0 的调整控制寄存器
0x0F0C	32	CTBm_OA1_OFFSET_TRIM	运放 Opamp1 的失调电压调整寄存器
0x0F10	32	CTBm_OA1_SLOPE_OFFSET_TRIM	运放 Opamp1 的调整控制寄存器
0x0F14	32	CTBm_OA1_COMP_TRIM	运放 Opamp1 的调整控制寄存器

22 LCD 段直接驱动

PSoC4 的液晶显示 (LCD) 模块可以直接支持 STN, TN 类型的段 LCD。

22.1 特性

LCD 段直接驱动模块具有以下特性：

- 可支持 4 个公共端
- 支持 Type A (标准型) 和 Type B（低功耗型）驱动波形
- 任何一个 GPIO 都可以作公共端或者段
- 支持三种驱动模式
  - 数字相关型
  - PWM 1/3 偏置
  - PWM1/2 偏置
- 在数字相关模式时，可以用 1.8V 的 VDD 来驱动 3V 的显示屏
- 可以工作在活动、睡眠和深度睡眠模式
- 数字对比度控制

22.2 LCD 段驱动概述

分段的液晶显示面板夹在两个电极（端和段）和各种偏振层和反射层之间。上下两个电极构成一个电极对，每对最多只能连到液晶屏的一个段。两个电极，其中一个叫端电极（COM）或者背板，另一个叫段电极（SEG）。从电路的角度来看，LCD 段可以看做一个非线性的电容负载，而公共端/段电极可以看做组成矩阵的行和列。LCD 段的不透明度取决于加在公共端和段之间的有效电压值（RMS）。

本章介绍 LCD 时会使用如下术语：

- **V_{LO}**：驱动电路产生的将段关闭的电压。
- **V_{HI}**：驱动电路产生的将段选通的电压。
- **区分度**：V_{HI} and V_{LO} 的比值。这取决于驱动电路的能力和驱动波形。比值越高，对比度越高。

液晶不能忍受长时间的直流电压。因此，任何驱动波形的直流分量必须为零（开和关时）。通常情况下，驱动到公共端和段的波形都会有多个电压变化。定义驱动波形会用到如下的术语：

- **占空比**：如果有 M 个公共端（COM），那这个驱动就被认为是 $1/M^{th}$ 的占空比。每个公共端在 $1/M^{th}$ 的时间内有效。
- **偏置**：如果驱动电压波形变化步长为用到$(1/B)*V_{DRV}$ 的，那就认为这个驱动采用 $1/B^{th}$ 偏置。其中，V_{DRV} 是系统中最高的驱动电压（在 PSoC4A 中是 Vddd）。
- **帧（Frame）**：每个帧的长度是选中所有段所需的时间。每个帧中，按段的顺序扫描所有公共端。在整个帧的尺度上看，所有段上的电压时直流分量为 0V 。

PSoC 4 支持两种驱动波形，如下：

- **Type-A 波形**：在这种波形中，如果有 M 是公共端，那么就将帧分成 M 个子帧。一个公共端，只会被选中一次。比如 COM[i] 会在子帧 i 中被选中。
- **Type-B 波形**：一个帧被分成 2M 个子帧。每个公共端在一个帧中会被选中两次。比如，COM [i] 会在 i 和 M+i 中被选中。Type-B 波形的能耗通常会比较低，因为它的电压变化比较小。

22.2.1 驱动模式

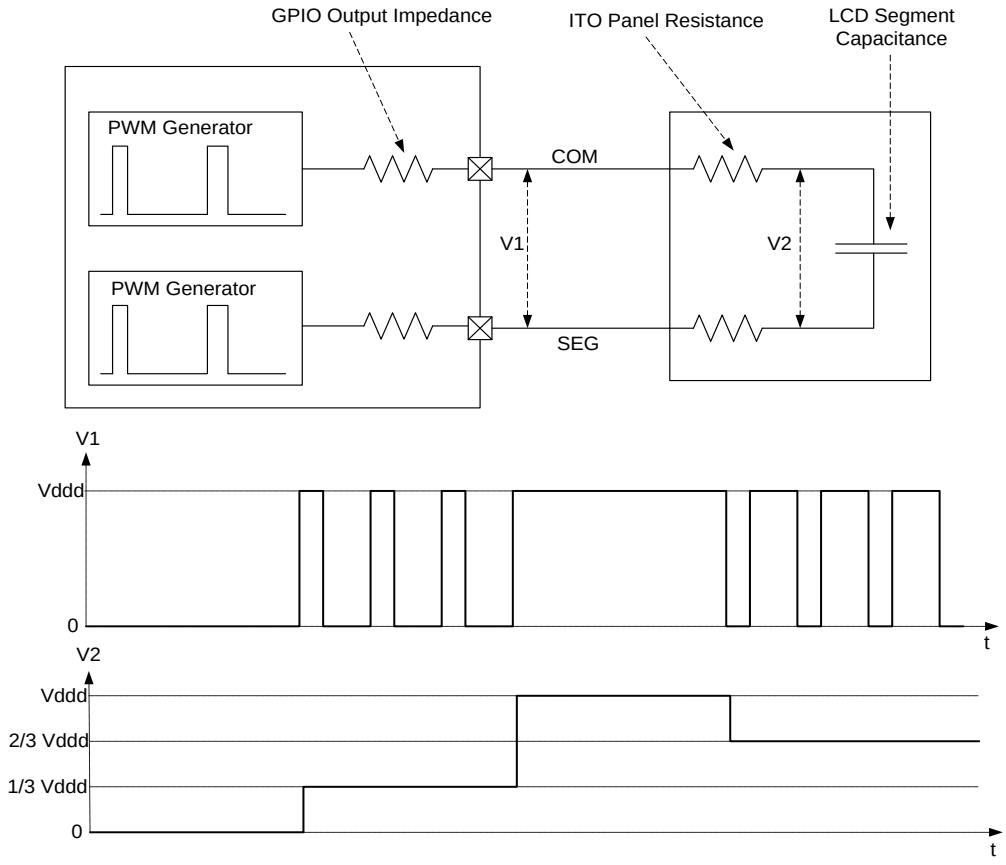
PSoC4A 支持一下的驱动模式：

- PWM 驱动
  - 1/3rdbias1/3rd 偏置
  - 1/2ndbias1/2nd 偏置
  - 数字校正

22.2.1.1 PWM 驱动

在 PWM 驱动模式中，多种驱动电压是有 PWMLCD 屏内在的电阻和电容产生的，如图 22-1 所示。

图 22-1：PWM 驱动（1/3 偏置）



驱动电路输出的是 PWM 波形。PWM 经过用 ITO（铟锡氧化物）的面板间电阻值和段电容的滤波，LCD 段两端的电压是一个平滑的模拟电压，如图 22-1 所示。

PWM 的波形可以由内部低速时钟 ILO 或者内部高速时钟 IMO 产生。驱动段码式 LCD 的模拟电压频率通常很低（~50Hz）。

图 22-2 和图 22-3 显示了 1/2rd 的偏置和 1/4th 占空比的 COM 和 SEG 波形(Type-A/Type-B)。图中只显示了 COM0/COM1 和 SEG0/SEG。

图 22-2. PWM1/2nd Type-A 波形

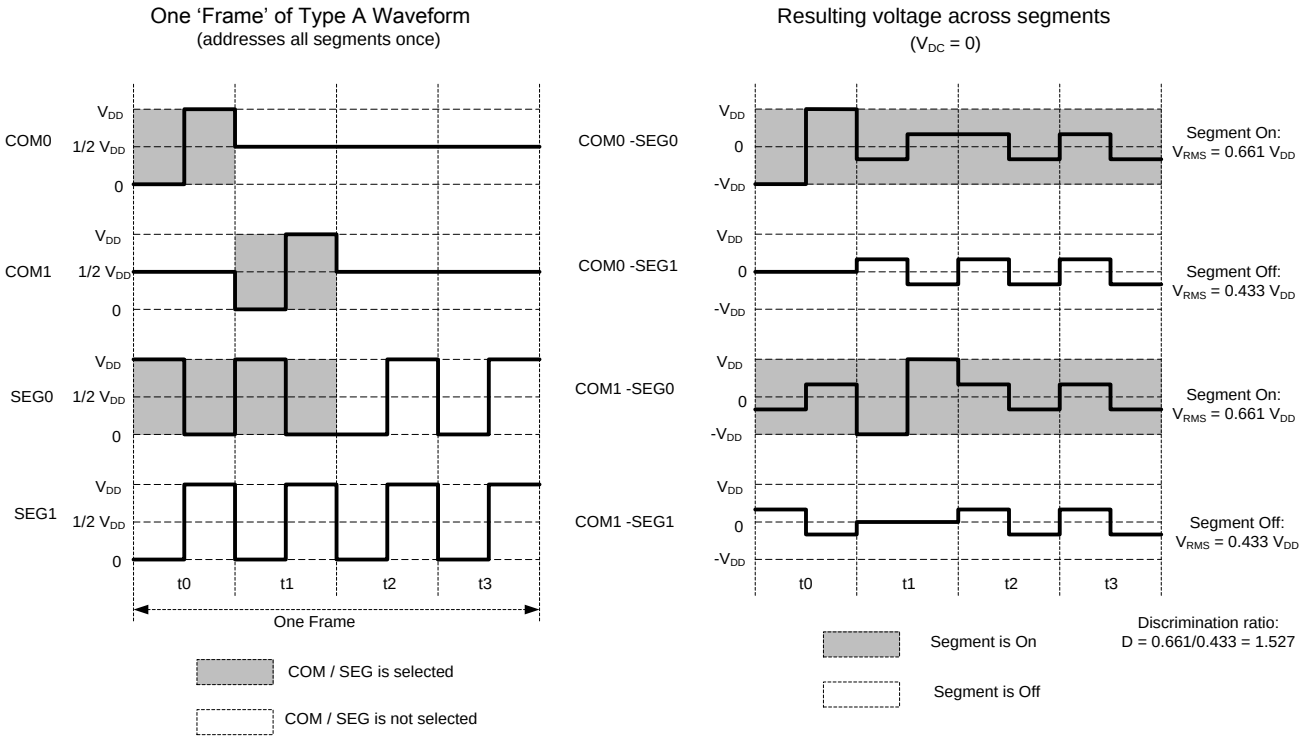


图 22-3. PWM1/2nd Type-B 波形实例

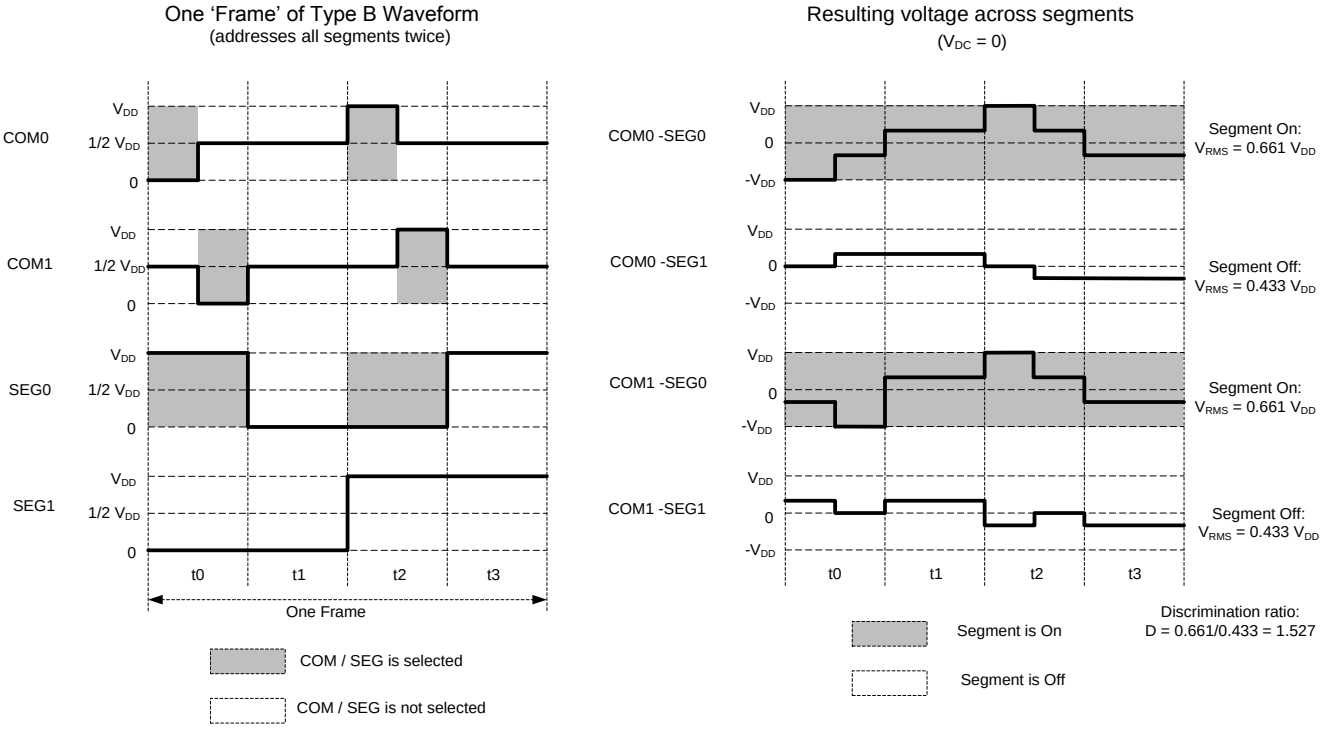


图 22-4. PWM1/3rdType-A 波形实例

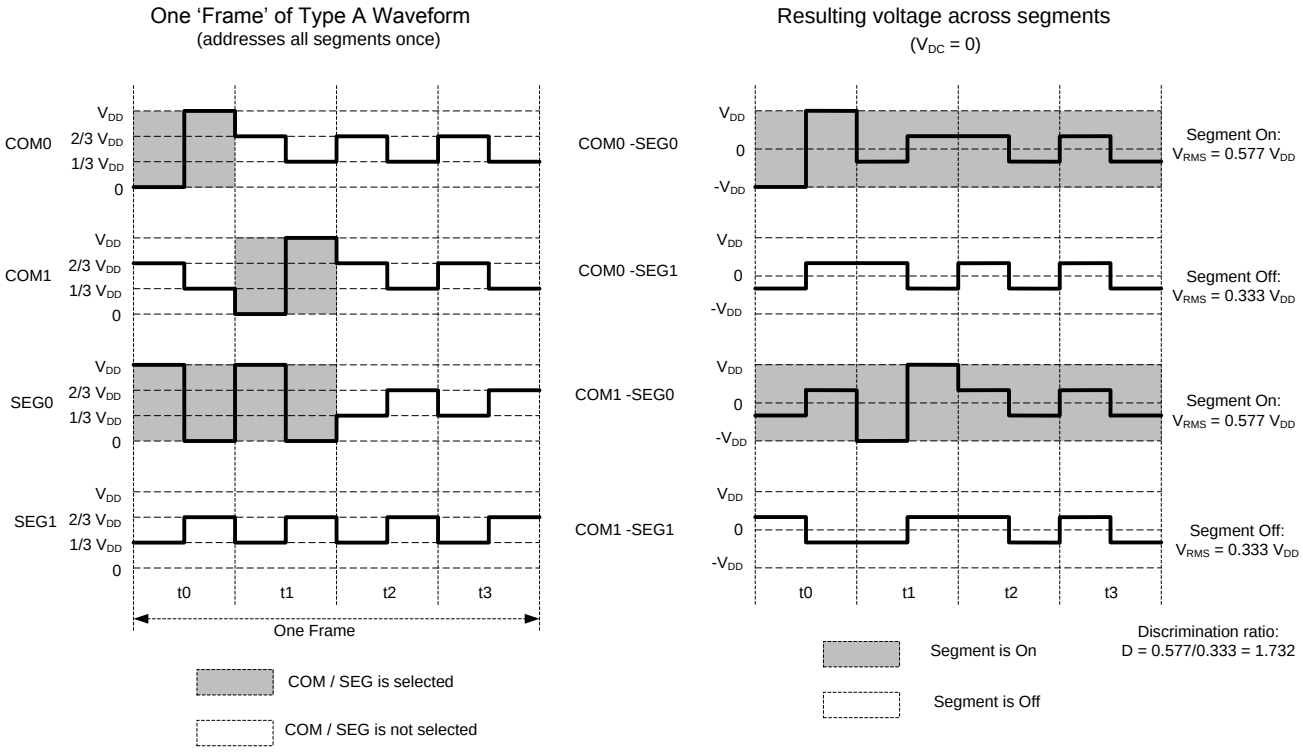
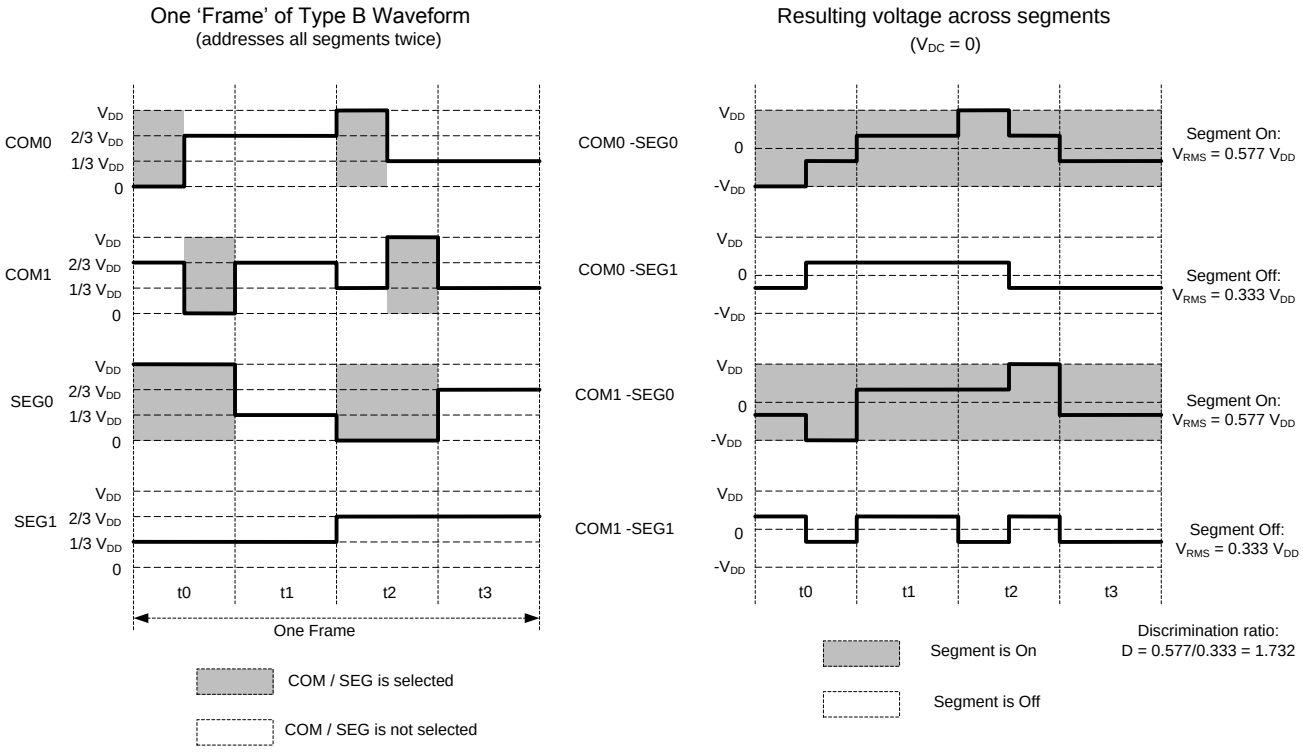


图 22-5. PWM1/3rdType-B 波形实例



使用如下这些方程式可以计算开通和关闭时电压的有效值（RMS）：

$$V_{RMS(OFF)} = \sqrt{\frac{2(B-2)^2 + 2(M-1)}{2M}} x (V_{DRV} / B)$$

$$V_{RMS(ON)} = \sqrt{\frac{2B^2 + 2(M-1)}{2M}} x (V_{DRV} / B)$$

B 是偏置，M 是占空比。对于大多数应用，建议采用 1/3rd 的偏置。

当 LCD 工作在低速模式时，PWM 信号来自于 32KHz 的 ILO（内部低速振荡器）。因为显示屏电容值很小，为了使纹波和上升/下降时间在可接受的范围，使用 32 kHz 的 PWM 时，有必要一个额外地串联 100K-1M 欧姆外部电阻。当 PWM 频率大于 1M 时，不需要外加电阻。最佳的 PWM 频率取决于屏的电容和 ITO 的电阻。

如图 22-2 和 22-3 所示，1/2ndPWM 的优势是 PWM 只需要加在 COM 端，SEG 端的信号只需要是或高或低的逻辑电平即可。因此如图的例子，1/2nd 偏置的 PWM 只需要四个外部电阻即可。

22.2.1.2 数字相关

数字相关模式不需要在两个电极上产生偏置电压，而是利用 LCD 的对比度取决于段上的电压有效值的特性，直接用逻辑电平来驱动 COM 和 SEG。利用 COM 和 SEG 的不同相位关系对应的两种差的不同有效值来决定 LCD 开通或关闭。

因此通过将 COM 信号在无效子帧时基础频率加倍，COM 和 SEG 驱动信号的相位关系可以被改变，两者的差的有效值也随之改变，从而开通或关闭 LCD 段。

工作的基本原理如图 22-6 和 22-7 所示。

图 22-6. 数字关联 type-A 波形实例

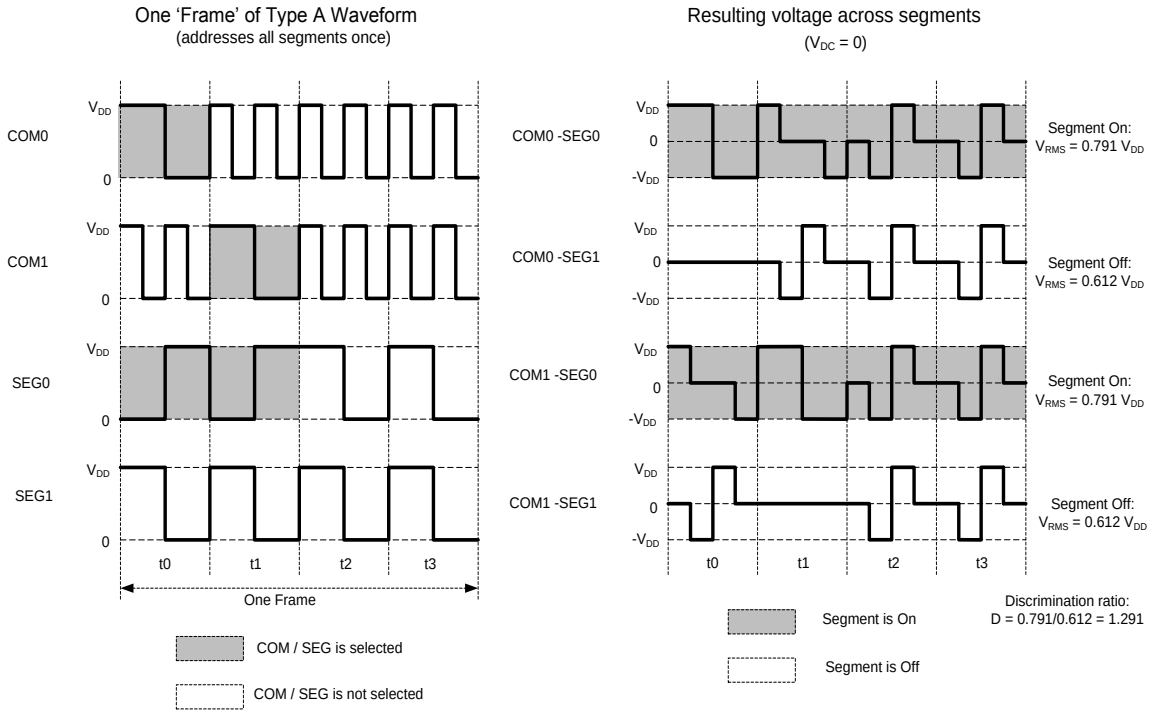
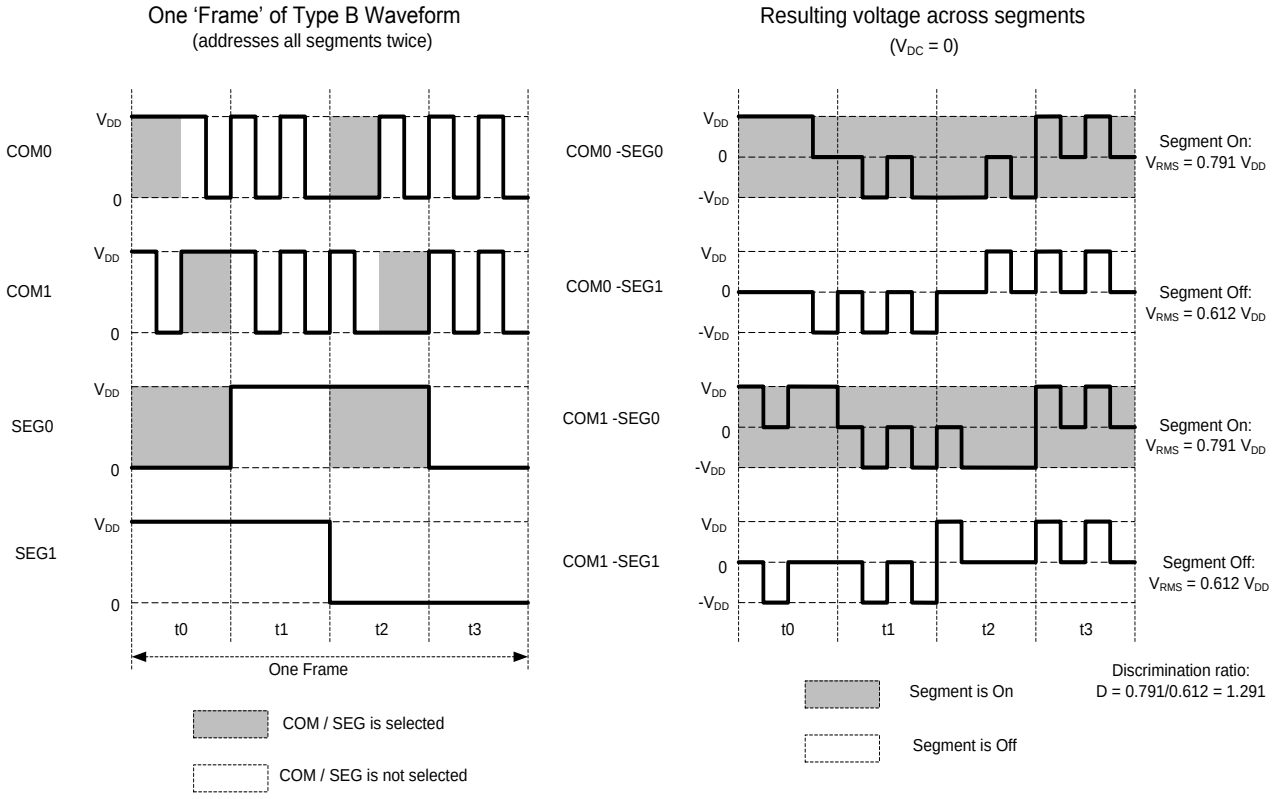


图 22-7. 数字相关模式 Type-B 波形实例



开通和关闭时段上的电压有效值如下公式：

$$V_{RMS(OFF)} = \sqrt{\frac{(M-1)}{2M}} x (V_{DD})$$

$$V_{RMS(ON)} = \sqrt{\frac{2+(M-1)}{2M}} x (V_{DD})$$

可以计算出，4 个 COM 时的区分度是 1.291。数字相关模式的对比度比 PWM 模式的要低。但是数字相关模式的功耗比较低，因为其波形变化的频率很低。数字相关模式用在 TN 显示器上可以看到降低但可以接受的对比度，但是在高对比度的 STN 屏上看不到明显的差异。

数字相关模式可以用 1.8V 的 Vdd 来驱动 3V 的显示屏。

22.2.2 推荐的驱动模式

PWM 驱动模式的对比度高于数字相关模式，详情请见 22.2.1.1PWM 驱动和 22.2.1.2 数字相关。在活动模式或者睡眠模式时，采用 PWM 模式以获得更好的对比度。在深度睡眠模式时，可采用数字相关模式来达到低功耗。模式的切换，需要重新配置寄存器。

数字相关模式用在 TN 屏上，虽然对比度降低，但是可以接受。用在自身对比度较高的 STN 屏上，对比度和可视角无明显差异。

每种驱动模式都有其优缺点，在不同情况下的推荐模式如表 22-1。

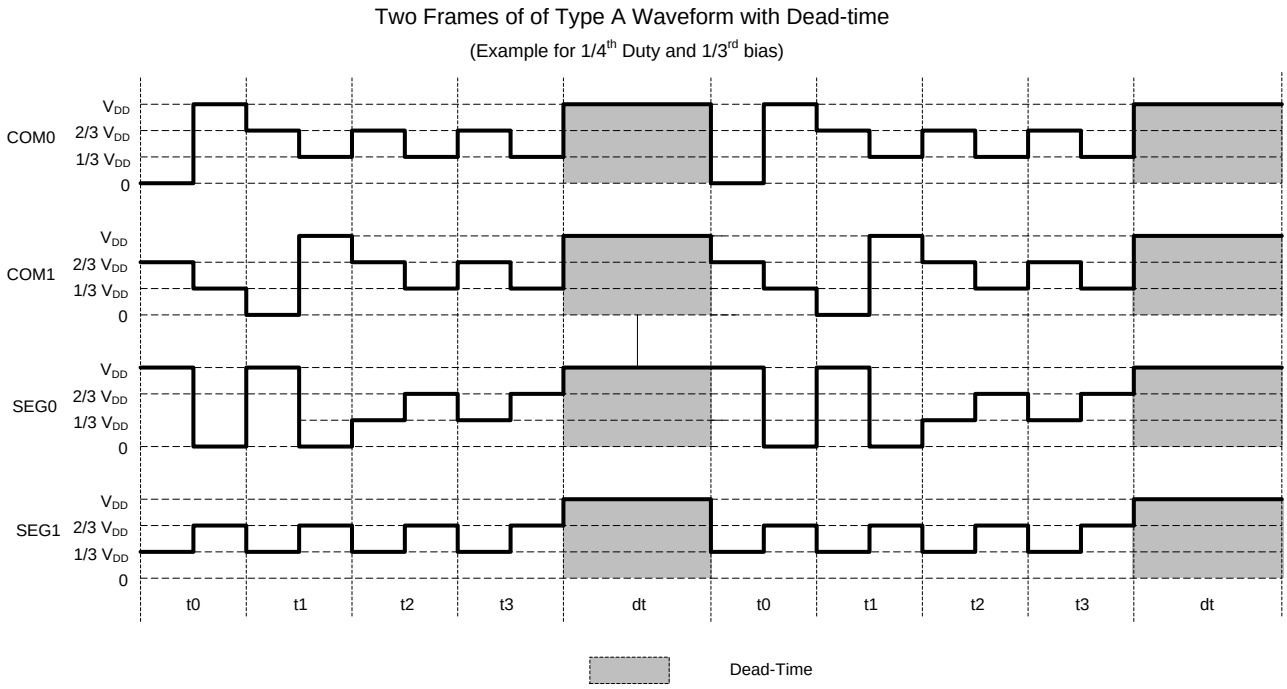
表 22-1. 推荐使用模式

显示器类型	深度睡眠模式	活动/睡眠模式	备注
TN Glass	数字相关	PWM 1/3 偏置	切换到深度睡眠或中唤醒，需要软件切换 LCD 驱动模式
STN Glass	数字相关		在 STN 显示屏上，PWM 驱动没有对比度优势

22.2.3 数字对比度控制

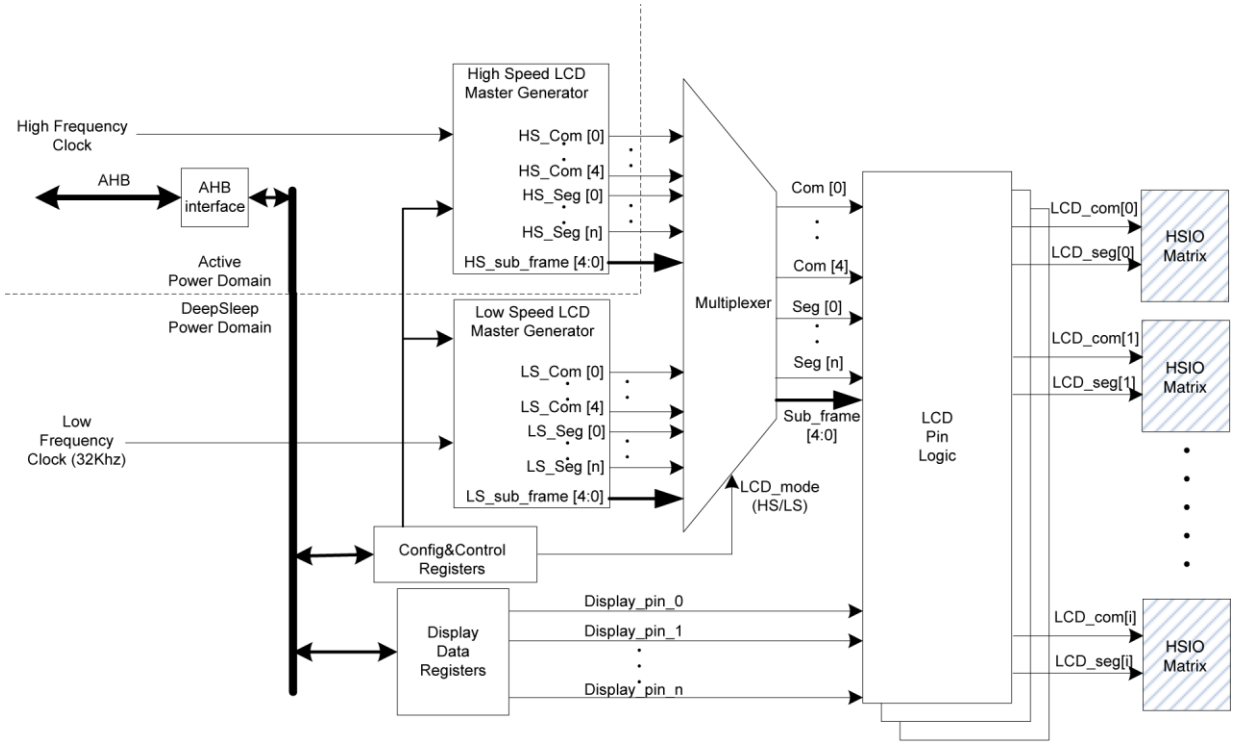
在所有驱动模式中，数字对比度控制可以用来改变段的对比度。这个方法通过在每帧后插入“死区时间”来减少驱动时间，以来减小对比度。在死区阶段，所有的 COM 和 SEG 都被驱动到高电平。图 22-8 显示了 1/3 偏置和 1/4 占空比 PWM 驱动的死区对比度控制方法。

图 22-8. “死区”对比度控制



22.3 LCD 模块介绍

图 22-9. LCD 直接驱动模块



22.3.1 工作原理

LCD 控制器有两个驱动信号源，一个来自有于内部主要时钟源(IMO)的高速时钟，另一个来自于内部低速时钟源（ILO，32KHz）的低速时钟。如图 22-11，它们分别被称作高速 LCD 驱动信号发生器和低速 LCD 驱动信号发生器，它们共用相同的配置寄存器。两个驱动信号发生器都支持 PWM 和数字关联驱动方式。如 22.2.1.1PWM 驱动模式所描述，低速 PWM 驱动模式需要外部电阻。

根据软件的配置，多路转换器选择这两个发生器输出的其中一个来驱动 LCD。LCD 管脚逻辑模块再将这个输出路由到相应的 I/O 管脚上。任何一个 GPIO 都可以用作 COM 或者 SEG，这可以由 GPIO 矩阵来实现，可以通过 GPIO 配置寄存器来选择，详情请见 GPIO 章节。

LCD 控制器可以工作在三种电源模式：活动、睡眠和深度睡眠。高速工作只支持活动或睡眠模式，低速工作则可以三种模式都支持。在休眠状态下，LCD 控制器不可用。

22.3.2 高速和低速驱动信号发生器

除了高速驱动信号发生器有较大的分频器以外，其余部分和低速驱动信号发生器比较相似。因为产生高速信号的的 IMO 的时钟频率一般是产生低速信号的 ILO (32 KHz)的 30-100 倍。高速驱动信号发生器工作在活动/睡眠电源域，低速驱动信号发生器工作在深度睡眠电源域。两者共用一组配置寄存器。

寄存器 LCD_CONTROL 可以配置如下参数：

- Type A 或 Type B 驱动波形选择，LCD_MODE 位(寄存器 LCD_CONTROL [3])。
- 公共端个数选择，可选值为 2，3，4，COM_NUM 字段（寄存器 LCD_CONTROL[11:8]）。
- 工作模式选择：
  - 数字关联模式，OP_MODE 位（寄存器 LCD_CONTROL[4]）
  - PWM 模式（两种模式二选一），BIAS 位（寄存器 LCD_CONTROL[6:5]）
  - PWM 1/2 偏置
  - PWM 1/3 偏置
- 停止：高速驱动信号发生器 HS_EN 位（寄存器 LCD_CONTROL [1]）；低速 LS_EN 位（寄存器 LCD_CONTROL[0]）。通常其中一个停止。

- 一个计数器用来产生子帧的时间：SUBFR_DIV 字段（寄存器 LCD_DIVIDER[15:0]）。如果写入的值位 C，那么子帧周期即为 4*(C+1)个时钟周期。低速信号发生器有 8 位（16 位的低 8 位）的计数器，对于 32KHz 的 ILO 时钟，可以产生最大 8ms 的 1/4 子帧时间。高速驱动信号发生器有 16 位的计数器。
- 死区时间计数器：DEAD_DIV 字段（寄存器 LCD_DIVIDER[31:16]）用来配置死区时间。和子帧时间计数器的位数一样。

22.3.3 多路复用器和 LCD 引脚逻辑

LCD_CONTROL [1:0] 寄存器中的可以分别用来使能或禁止低速和高速驱动信号发生器。高速或者低速信号发生器产生的信号经过 LCD 引脚逻辑模块，连接到高速 IO 矩阵（HSIO Matrix）。

22.3.4 显示数据寄存器

任何一个 GPIO 都可以通过寄存器 HSIOM4A_PORT_SEL0...4 配置成 COM 或者 SEG。每个管脚在寄存器 LCD_DATA0 中有相应的四个比特位（即比特位 [4i+3:4i] 对应管脚[i]），对应这个管脚在在子帧（0-3）中的选中情况。一共有 8 个 LCD_DATA0 寄存器，对应不同的端口的管脚。但是同一子帧中只能有一个 COM 被选中，SEG 则可以任意选择。如果管脚的 COM 和 SEG 在子帧中被同时选中，则这个 LCD 段就会选通。所有子帧中选通的 LCD 段就是这一帧的选通段，在 LCD 上可以显示出来。

如果有四个 COM，其在每个子帧中的数据如下：

COM 0 – 1, 0, 0, 0

COM 1 – 0, 1, 0, 0

COM 2 – 0, 0, 1, 0

COM 3 – 0, 0, 0, 1

22.4 寄存器表

表 22-2. 直接驱动 LCD 寄存器列表

寄存器名	Description
LCD_DIVIDER	控制子帧和死区时间寄存器
LCD_CONTROL	配置驱动信号寄存器
LCD_DATA0	LCD 显示数据寄存器，即 COMx 和 SEGx
LCD_DATA1	LCD 引脚数据寄存器
LCD_DATA2	LCD 引脚数据寄存器
LCD_DATA3	LCD 引脚数据寄存器

23 CapSense

CapSense 电容感应技术允许用户应用手指的电容效应来控制按键、滑条和轮盘，此外触摸板和触摸屏也是常见的应用。PSoC 4 使用的电容感应技术是 CSD（CapSense Sigma Delta），它能够提供业界一流的信号噪声比。CSD 电容感应包括了硬件和软件技术。

23.1 特性

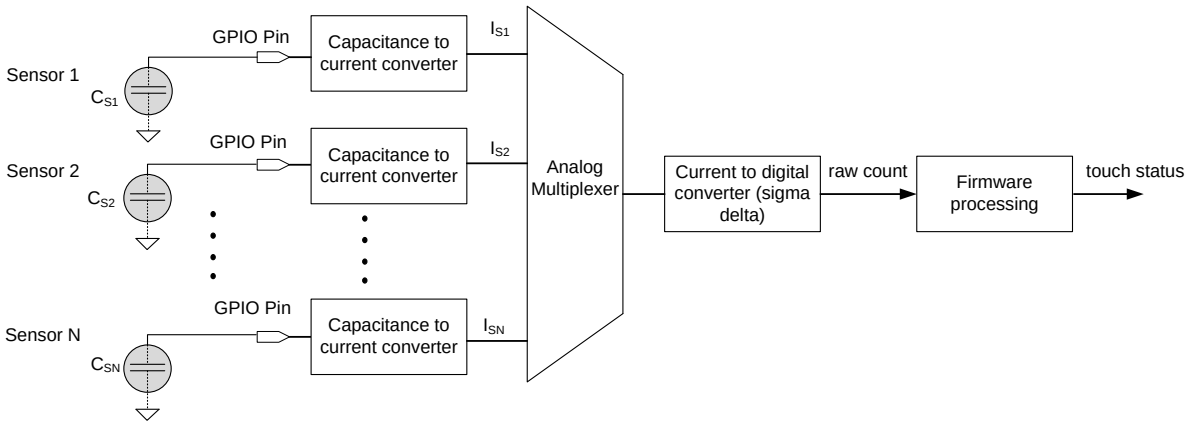
PSoC 4 CSD 模块有如下特性：

- □鲁棒的感应技术；
- □业界一流的信号噪声比；
- □支持多种覆盖物；
- □SmartSense™ 智能感应技术；
- □最多支持 35 个电容感应按键；
- □灵敏的接近式感应；
- □防水性能；
- □低功耗；
- □双 IDAC 的结构能够有效的提高扫描速度和信噪比；
- □任何一个管脚均可用于连接感应传感器（调制电容专用管脚除外）；
- □伪随机序列的时钟源，有效的降低电磁干扰；
- □GPIO 单元预充电模式，提高模块的初始化速度；

23.2 模块框图

图 23-1 显示了 CSD 感应方法的示意图。

图 23-1. CSD 感应方法的示意图



23.3 工作原理

当感应传感器连接到管脚上，PSoC 4 内部 GPIO 单元的开关电容电路将感应传感器的电容量转换为一个等效的电流值。通过一个模拟的多路器，某一路的电流值被送到电流-数字转换器，该电流-数字转换器类似于一个 Delta-Sigma 架构的模数转换器。

电流-数字转换器的输出被称为初始计数值（raw count），该数值与传感器的电容值成正比，如式 23-1 所示。

$$\text{raw count} = G_C C_S \qquad (23-1)$$

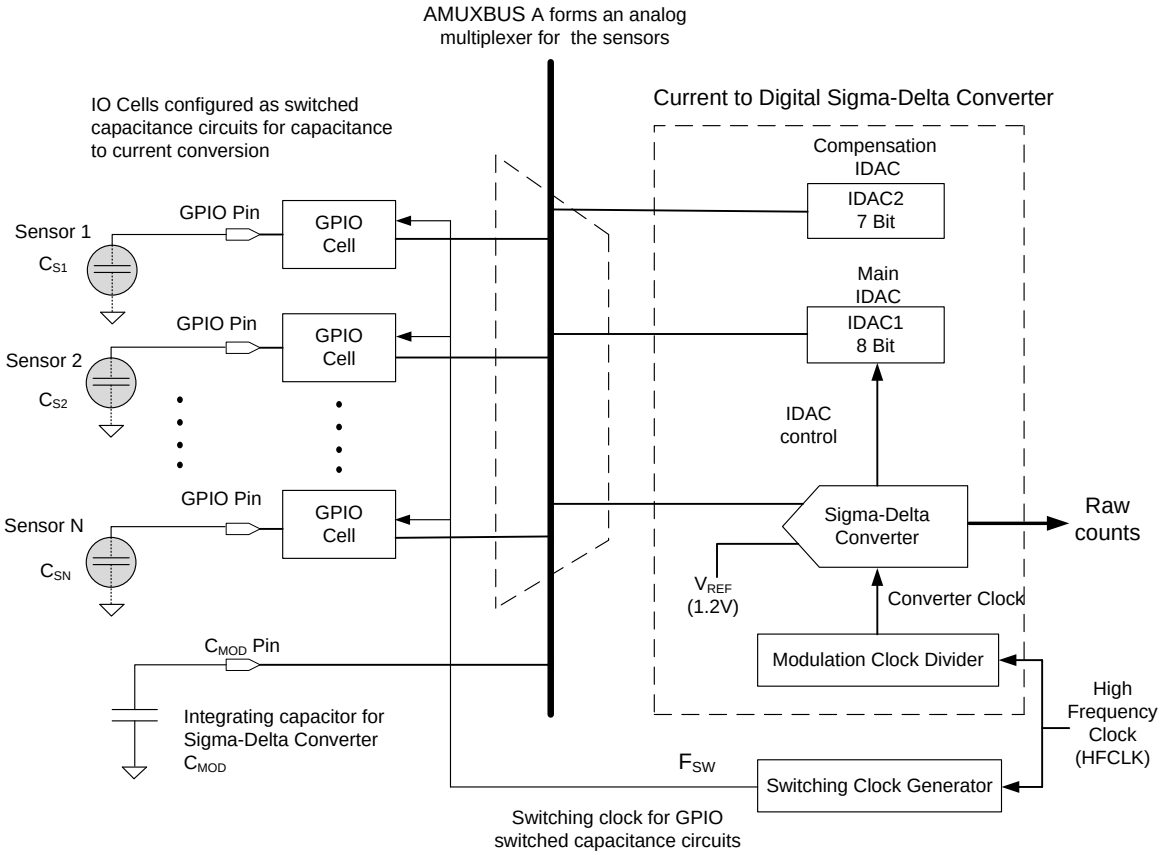
G_C 是 CSD 模块的电容-数字的转换增益。

当有手指靠近或者接触传感器时，电容值会增大，初始计数值也会相应的增大。当计数值的变化超过一个阈值时，软件就可以判定手指的存在。

23.3.1 CSD 感应原理

图 23-2 给出 PSoC 4 的 CSD 模块框图。

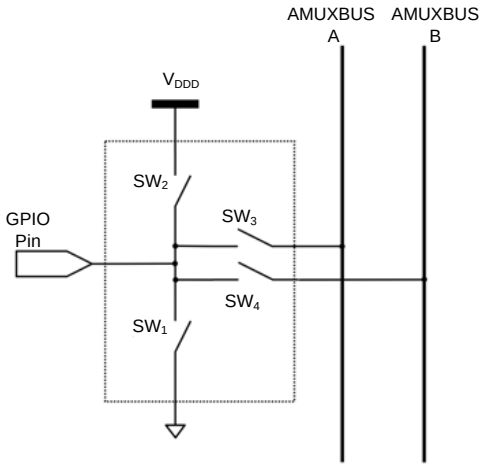
图 23-2. CSD 模块框图



23.3.1.1 GPIO 单元的电容-电流转换器

当一个管脚被配置为 CSD 的用途时，GPIO 单元的开关电容电路可以将传感器的电容量转换为等效的电流值。图 23-3 给出了 GPIO 单元中开关电容电路的简化框图。

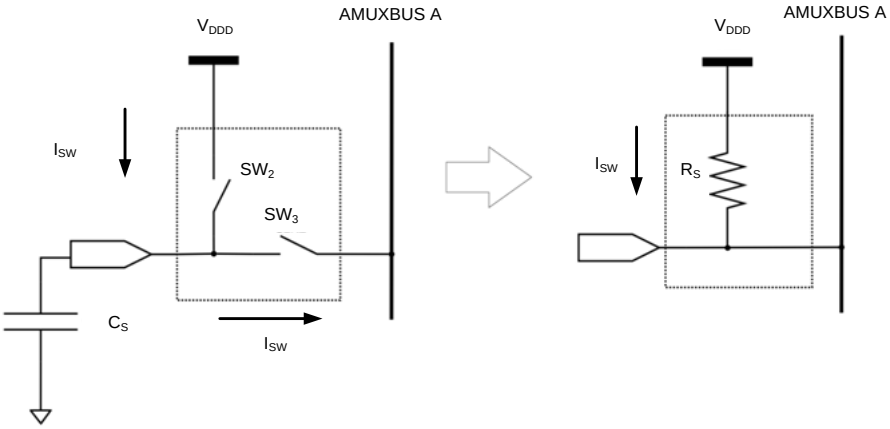
图 23-3. 开关电容简化框图



PSoC 4 有两条模拟总线：AMUXBUS A 被用于连接 CSD 应用中的传感器；AMUXBUS B 被用于连接 CSD 应用中的屏蔽电极（详见章节 23.3.2）。GPIO 单元的开关电容电路有两种工作模式：向 AMUXBUS A 输出拉电流（简称拉电流模式）；从 AMUXBUS A 接受灌电流（简称灌电流模式）。

图 23-4 给出了开关电容拉电流工作模式下的电路图。

图 23-4. 开关电容拉电流模式



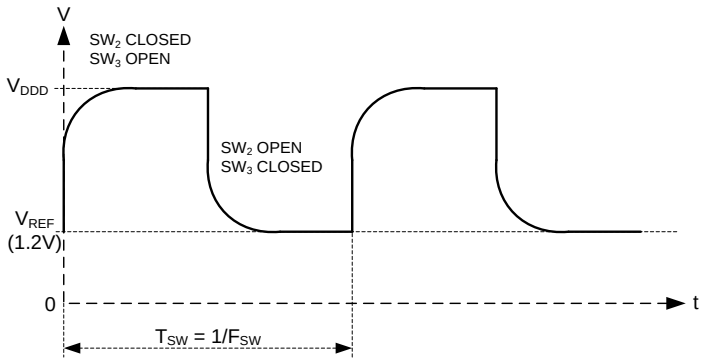
开关电容电路有两个开关：SW_2 和 SW_3，两个同频率的互补的方波分别用于控制这两个开关，被称为开关时钟，它的频率为 F_{SW}。通过开关电容电路，传感器的电容可以被等效为一个电阻 R_S,如式 23-3 所示。

$$R_S = \frac{1}{C_S F_{SW}} \qquad (23-2)$$

这里 C_S 是传感器的电容，F_{SW} 是方波的频率。

Sigma-Delta 转换器将 AMUXBUS A 的电压维持在 V_{REF} 附近（详见章节 23.3.1.3）。图 23-5 给出了开关电容拉电流模式下，传感器（电容 C_S）上的波形。

图 23-5. 开关电容拉电流模式中 C_S 上的波形



式 23-3 给出了开关电容向 AMUXBUS A 输出的拉电流的平均值

$$I_{CS} = C_S F_{SW} (V_{DDD} - V_{REF}) \qquad (23-3)$$

图 23-6 给出了开关电容电路灌电流工作模式下的电路图。图 23-7 给出了开关电容灌电流模式下，传感器（电容 C_S）上的波形。

图 23-6. 开关电容灌电流模式

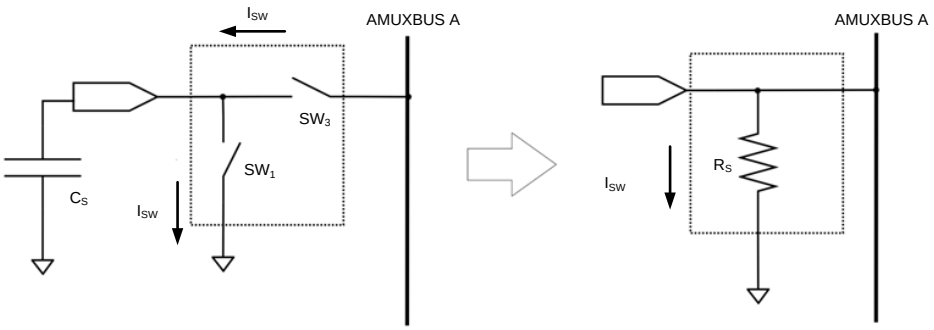
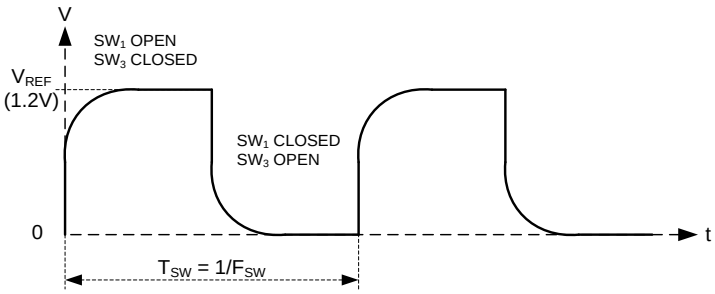


图 23-7. 开关电容灌电流模式中 C_S 上的波形



式 23-4 给出了开关电容从 AMUXBUS A 接收的灌电流的平均值

$$I_{CS} = C_S F_{SW} V_{REF} \qquad (23-4)$$

23.3.1.2 开关时钟发生器

如图 23-2 所示，CSD 模块的开关时钟发生器将 PSoC 4 高频时钟（HFCLK）分频，得到 CSD 模块的开关时钟，该开关时钟用于控制开关电容电路的开关。开关时钟发生器一共有三个工作模式：固定频率，8 比特伪随机序列，12 比特伪随机序列。通过配置发生器中分频器的分频比（Analog Switch Divider），能够得到不同频率（F_{SW}）的开关时钟。具体的配置公式见式 23-5。

$$F_{SW} = \frac{\text{HFCLK}}{2\ \text{Analog Switch Divider}} \qquad (23-5)$$

当时钟发生器工作在固定频率模式时，输出的开关时钟频率为固定的 F_{SW}。当时钟发生器工作在伪随机序列的模式下，输出的开关时钟频率会在一个范围内变化。式 23-6 和式 23-7 给出了频率变化范围的上限与下限。

$$F_{SW}\ (\text{maximum}) = \frac{\text{HFCLK}}{\text{Analog Switch Divider}} \qquad (23-6)$$

$$F_{SW}\ (\text{minimum}) = \frac{\text{HFCLK}}{\text{m Analog Switch Divider}} \qquad (23-7)$$

m 是伪随机序列的分辨率。

23.3.1.3 电流-数字转换器

电流-数字转换器 Sigma Delta 转换器将输入电流转化为数字值。它由一个 Sigma Delta 转换器，一个时钟分频器和两个电流型数模转换器（IDAC）组成，如图 23-2 所示。其中， IDAC1 的分辨率为 8 位，称为主 IDAC；IDAC2 的分辨率为 7 位，称为补偿 IDAC。IDAC2 能够提高 CSD 模块的性能，但在基础的操作中，它不是必须的。CSD 模块的正常工作还需要外接一个积分电容，称为 C_{MOD}，推荐值为 2.2nF。

在工作过程中，Sigma Delta 转换器将连接 C_{MOD} 的模拟总线 AMUXBUS A 上的电压保持在 V_{REF} 左右，具体如下：

- IDAC 拉电流模式：当开关电容从 AMUXBUS A 接受灌电流时，IDAC 向 AMUXBUS A 输出拉电流。当二者达到动态平衡时，总线上的电压被维持在 V_{REF} 附近。
- IDAC 灌电流模式：当开关电容向 AMUXBUS A 输出拉电流时，IDAC 从 AMUXBUS A 接受灌电流。当二者达到动态平衡时，总线上的电压被维持在 V_{REF} 附近。

Sigma Delta 转换器的分辨率可配置，配置范围从 8 比特到 16 比特。当 IDAC2 未被使用时，初始计数值与传感器电容成正比。在 IDAC 拉电流模式下，初始计数值的计算公式如式 23-8 所示。

$$\text{raw count} = 2^N \frac{V_{REF}\ F_{SW}}{I_{DAC1}}\ C_S \qquad (23-8)$$

类似，IDAC 灌电流模式下，初始计算值的计算公式如实 23-9 所示。

$$\text{raw count} = 2^N \frac{(V_{DD} - V_{REF})\ F_{SW}}{I_{DAC1}}\ C_S \qquad (23-9)$$

在使用 IDAC2 后，IDAC 拉电流模式下的初始计数值计算公式如式 23-10 所示。

$$\text{raw count} = 2^N \frac{V_{REF}\ F_{SW}}{I_{DAC1}}\ C_S - 2^N \frac{I_{DAC2}}{I_{DAC1}} \qquad (23-10)$$

在使用 IDAC2 后，IDAC 灌电流模式下的初始计数值计算公式如式 23-11 所示。

$$\text{raw count} = 2^N \frac{(V_{DD} - V_{REF})\ F_{SW}}{I_{DAC1}}\ C_S - 2^N \frac{I_{DAC2}}{I_{DAC1}} \qquad (3-10)$$

这里，N 是 Sigma Delta 转换器的分辨率，I_{DAC1} 是 IDAC1 的电流值，I_{DAC2} 是 IDAC2 的电流值。

23.3.1.4 模拟多路器

在同一时刻，Sigma Delta 转换器只能够扫描一个传感器。通过一个模拟的多路器，某个 GPIO 单元上的传感器被连接到 Sigma Delta 转换器的输入，如图 23-1 所示。模拟总线 AMUXBUS A 和 GPIO 单元的开关（如图 23-3 中的 SW_3）共同组成了这个模拟多路器。

23.3.2 屏蔽电极

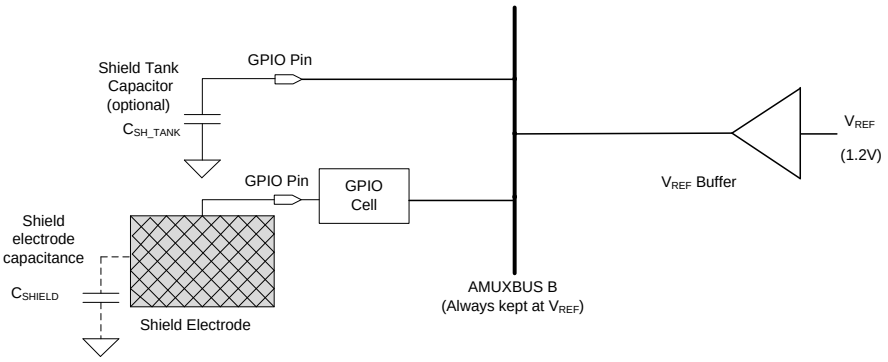
PSoC4 的 CapSense 支持屏蔽电极，以完成防水和接近式感应的设计。在防水设计中，屏蔽电极与传感器保持相同的电势。PSoC 4 的 CapSense 通过向屏蔽电极和传感器驱动相同的开关信号，以消除二者之间的电势差。

在感应传感器电路中，Sigma Delta 转换器将模拟总线 AMUXBUS A 上的电压维持在 V_{REF} 附近。GPIO 单元通过切换开关将传感器周期性的连接到 AMUXBUS A 和电源轨上（VDD 或者地，取决于开关电容工作在拉电流模式或者灌电流模式）。屏蔽电路工作情况类似，此时用到的模拟总线为 AMUXBUS B。

针对不同电容值的屏蔽电极应用，为了将 AMUXBUS B 的电压保持在 V_{REF} 附近，有两种不同的配置方法。

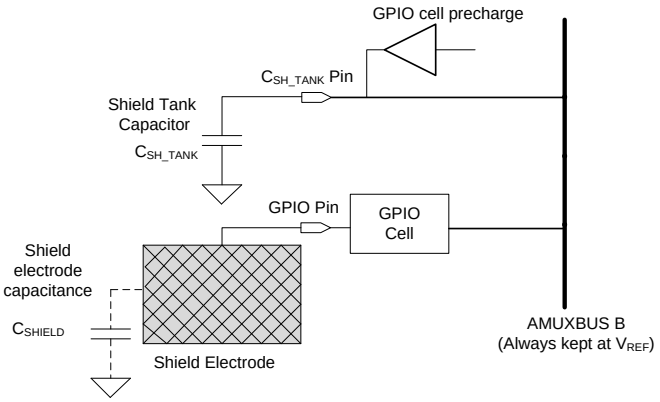
当屏蔽电极的电容值小于 200pF 时，一个 V_{REF} 电压缓冲器用于驱动 AMUXBUS B，如图 23-8 所示。当屏蔽电极的电容 C_{SHIELD} 为 60pF 左右时，外接积分电容 C_{SH_TANK} 的推荐值为 2.2nF。

图 23-8. V_{REF} 电压缓冲器驱动 AMUXBUS B



当 C_{SHIELD} 的电容值超过 200pF 时，连接 C_{SH_TANK} 的 GPIO 单元被用于驱动 AMUXBUS B，如图 23-9 所示。PSoC 4 中只有某些固定 GPIO 单元有这样的驱动能力，详细信息请参见 PSoC 4 数据手册中的管脚分布。

图 23-9. GPIO 单元驱动 AMUXBUS B



23.3.3 C_{MOD} 的预充电

当 CSD 模块第一次被使能时，C_{MOD} 上的电压初始值为 0。在工作过程中，C_{MOD} 的电压会被缓慢的充到 V_{REF}，充电电流来自于 IDAC 或者开关电容电路。该充电过程很缓慢，因为 C_{MOD} 是一个较大的电容。

为了加快 CSD 模块的初始化，可以对 C_{MOD} 进行预充电，令它的电压迅速达到 V_{REF}。预充电的方法共有两种：

- 利用 V_{REF} 电压缓冲器预充电：C_{MOD} 首先被连接到 V_{REF} 电压缓冲器上进行预充电，预充电结束后，C_{MOD} 被断开。当屏蔽电极被使能时，C_{MOD} 通过 AMUXBUS B 连接到 V_{REF} 电压缓冲器上进行预充电；当屏蔽电极被禁止时，C_{MOD} 通过 AMUXBUS A 连接到 V_{REF} 电压缓冲器上进行预充电。
- 利用 GPIO 单元预充电：此时连接 C_{MOD} 的 GPIO 单元被用于为 C_{MOD} 充电，这种方法的充电速度更快。PSoC 4 中只有某些固定 GPIO 单元有这样的驱动能力，详细信息请参见 PSoC 4 数据手册中的管脚分布。

24 温度传感器

PSoC4有一个片上的温度传感器，用于测量芯片内部温度。这个温度传感器是一个三级管，配置成二极管模式。其基本原理是利用基极结电压（Vbe）的温度依赖性来测量温度。

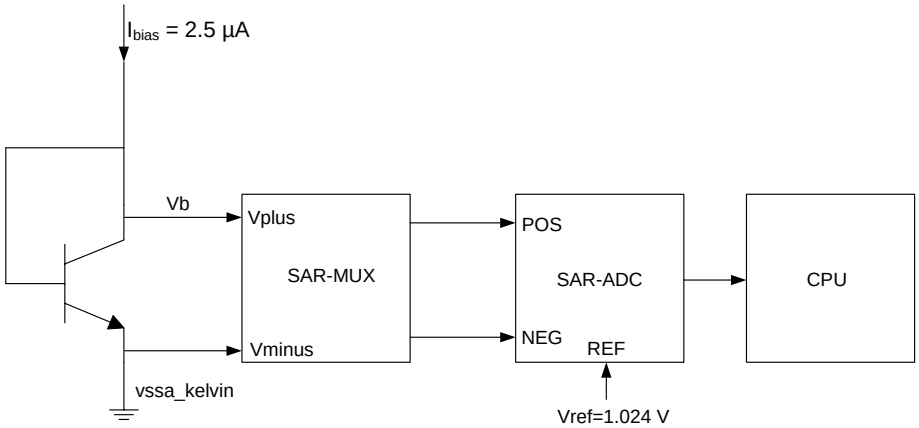
24.1 特性

这个温度传感器具有以下特性：

- -40ºC to +100ºC 范围内有± 5 ºC 精度
- 0.5 ºC/LSB 分辨率
- 10 µS 的稳定时间

24.2 工作原理

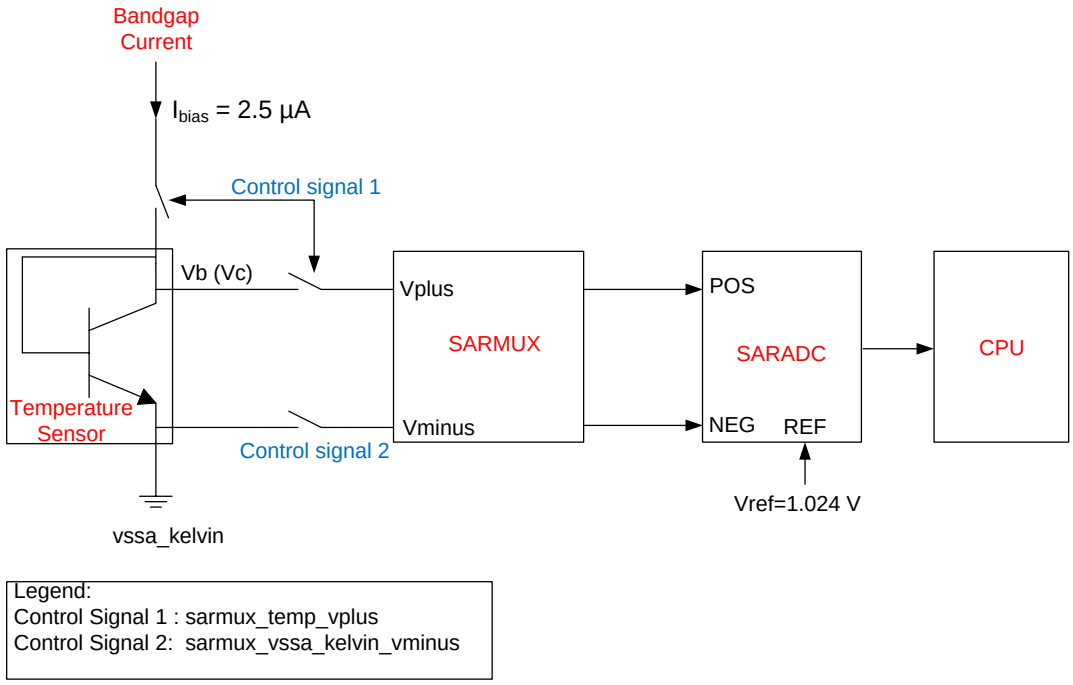
图 24-1. 温度传感器工作机制



温度传感器

在一个集电极电流恒定和集电极 — 基极电压为零时，双极结型晶体管（BJT）的基极 — 发射极电压（Vbe）具有很强的温度依赖性，且温度和 Vbe 的关系基本呈线性。这样用 SAR ADC 的单端，无符号模式读出 Vbe 后，用如下的一个简单的线性公式就可以计算出温度，如图 24-2 所示。

图 24-2 温度传感器工作机制



读到 SAR ADC 的输出并在软件中进行校正后，可以通过以下线性公式得到温度。

$$Temp = A \times V_{be} + B$$

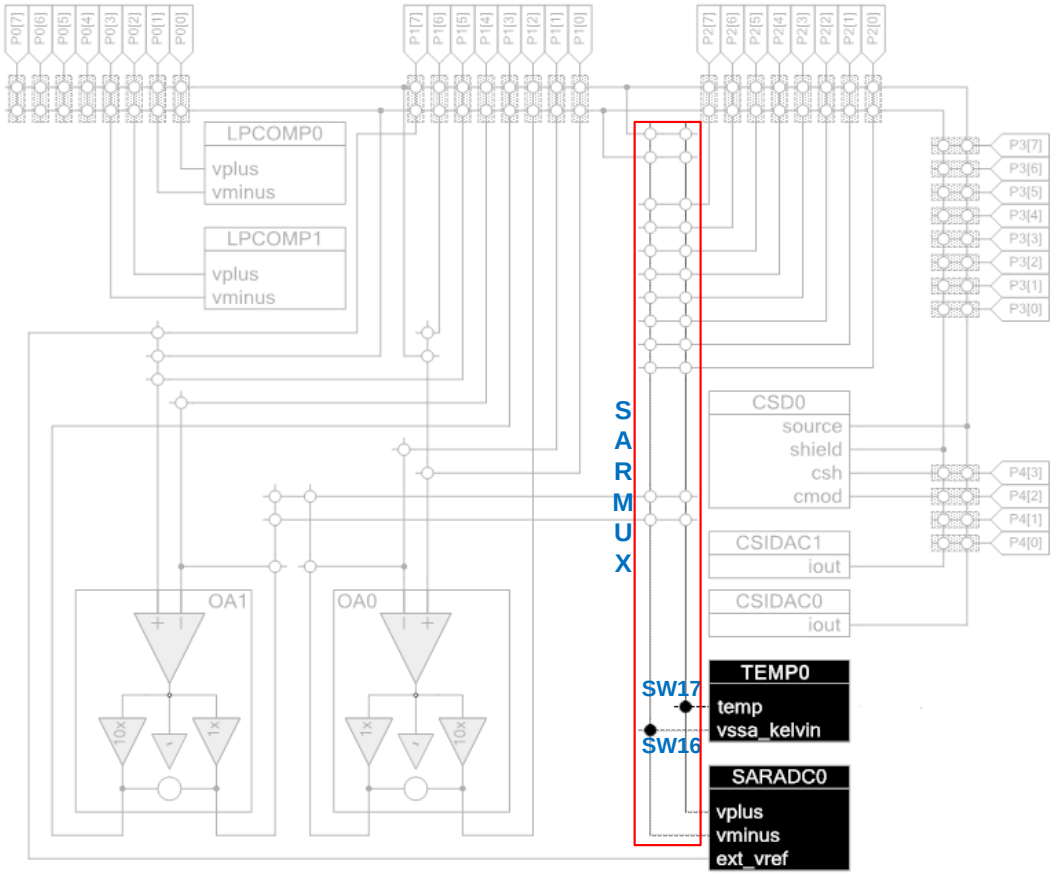
注意：A 和 B 是在出厂的时候存在闪存中的 16 位常数。用户不能改变它们的值。

- “A”的值取决于 PSoC 4 系列特性，对于所有该系列的芯片都是都一个恒定的常数。储存在 PSoC creator 定义的寄存器 CYREG_SFLASH_SAR_TEMP_MULTIPLIER 中(内存地址：0x0FFFF164)。两个十六位的定点数（A 与 Vbe 数模转换结果）相乘，乘积是一个 32 位的定点数（16.16）。
- “B”的值取决于每块芯片的生产流程中各种变量和实际的偏置电流，所以每块芯片都会不同。储存在 PSoC creator 定义的寄存器 CYREG_SFLASH_SAR_TEMP_OFFSET 中(内存地址：0x0FFFF166)。B 将被乘以 1024 得到一个 32 位定点数的乘积（16.16）。
- “Temp”（温度）以 32 位的定点数（16.16）表示，高 16 位是温度的整数部分，低 16 位是小数部分。32 位的温度值右移后可得到温度的整数值。例如，0x1E5DFC 表示 30.36713ºC；0xFFFD09D2 表示-2.96164ºC。

24.3 温度传感器配置

SAR MUX 中的开关不仅可以将 SAR ADC 的输入选择为温度传感器的输入，而且选中的同时将会使能温度传感器，这是通过偏置电流来实现的。如图 24-3 所示，通过 SAR ADC 定序器（sequencer），软件（firmware），数字互联信号(DSI)都可以将温度传感器的输出连到 SAR ADC 的正端输入。详情请见：*19.3.13 温度传感器配置*。

图 24-3. 温度传感器输出连接到 SARADC



注意：温度传感器不适用于 SAR ADC 的差分模式（转换结果不确定），只能用单端模式。SAR ADC 的参考源只能从内部的 1.024V 来。

24.4 算法

1. 使能 SARMUX 和 SARADC。
2. 将 SARADC 配置成单端模式，SARADC 的负端接地（强制）。配置 SARADC 电压参考为内部精确参考电压 1.024V。SARADC 的输出范围为 0 到 2048（12-bit 分辨率）。
3. 配置 SARMUX 使 SARADC 的输入来自温度传感器，正确配置的同时，温度传感器会自动使能。
4. 读取 SARADC 的输出值。
5. 从 CYREG_SFLASH_SAR_TEMP_MULTIPLIER 中获取 A 的值，从 CYREG_SFLASH_SAR_TEMP_OFFSET 中获取 B 的值。
6. 利用公式：Temp = A × Vbe + B 计算温度
   - 例如：A = 0xBC4B，B = 0x65B4
   - 假设 SARADC (Vbe) 输出是 0x595
   - 用软件做如下计算：

1. A 乘以 Vbe
   - 0xBC4B × 0x595 = $(-17333)_{10} \times (1429)_{10} = (-24768857)_{10}$
2. B 乘以 1024
   - 0x65B4× 0x400 = $(26036)_{10} \times (1024)_{10} = (26660864)_{10}$
3. 将第一步和第二部的结果相加
4. $(-24768857)_{10} + (26660864)_{10} = (1892007)_{10}$ = 0x1CDEA7
5. 温度整数值（高 16 位）= 0x001C = $(28)_{10}$
6. 温度小数值（低 16 位）= 0xDEA7 = $(0.86974)_{10}$
7. 结合第 4，第 5 步，Temp = 28.86974°C

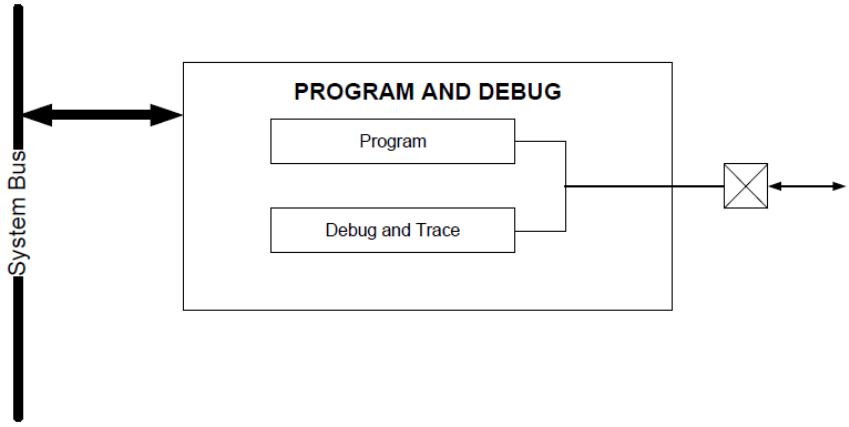
G 部分：编程和调试

这部分包括以下章节：

- 编程和调试接口　第 238 页
- 非易失性存储器编程　第 244 页

系统架构：

编程和调试系统框图



25 编程和调试接口

PSoC4 可以通过外部设备进行编程/调试，这个设备可以是 Cypress 提供的编程/调试器，或者是第三方支持 PSoC4 编程/调试的工具。PSoC4 的编程/调试采用标准的串行线调试（SWD）接口。

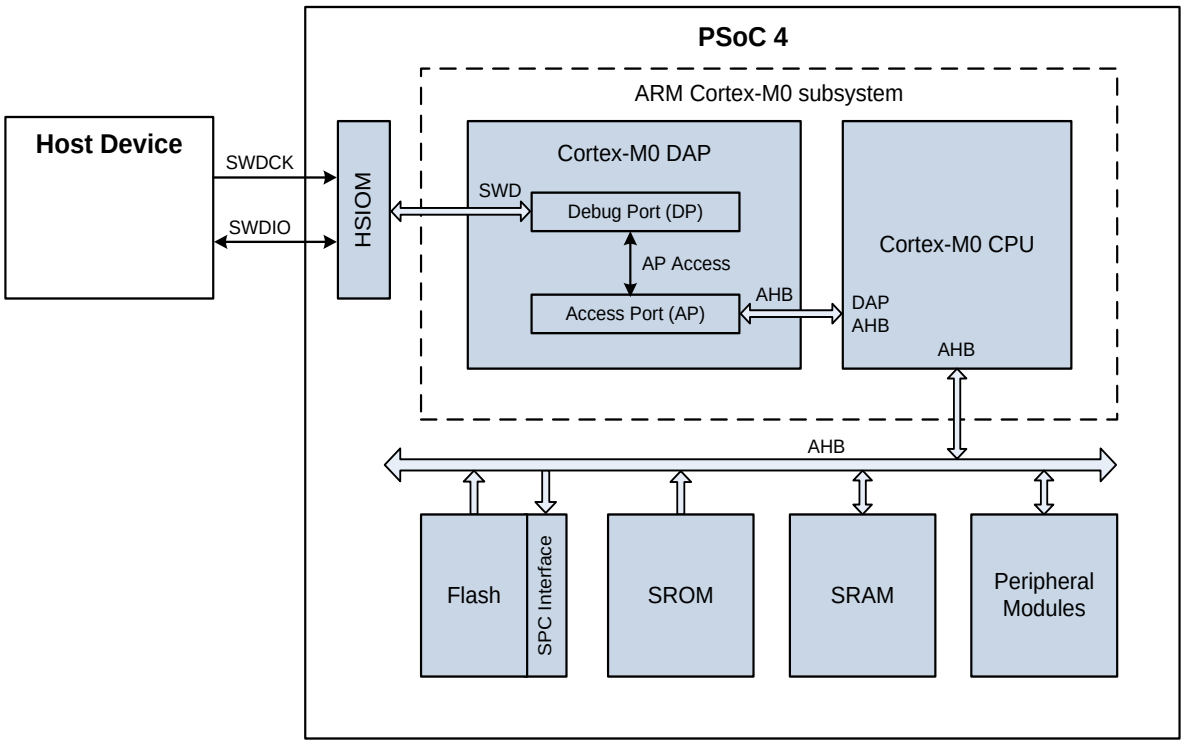
25.1 特性

- 通过 SWD 支持编程、调试
- 调试支持四个硬件断点（breakpoint）和两个数据观察点（watchpoint）
- 调试中可以访问到系统中所有的存储器、外设寄存器和 CPU 内部寄存器，包括 CPU 内核

25.2 功能介绍

图 25-1 显示了 PSoC4 编程/调试接口的系统框图。Cortex M0 的调试和访问端口（DAP）可以认为是 PSOC 4 的编程和调试接口。外部的编程器或调试器（以下引用为主机）通过两线的 SWD 接口与 DAP 通讯 — 包括一个双向数据线（SWDIO）和一个时钟线（SWDCK）。SWD 接口通过高速 IO 矩阵开关（HSIOM）与 PSoC4（以下引用为目标器件）内部的 DAP 通讯。HSIOM 是一些多路复用开关，通过 HSIOM，任何 GPIO 引脚都可以复用多种功能，即将 GPIO 与不同的模块相连，比如 LCD，CapSense 等等。

图 25-1. 编程/调试接口



DAP 通过 AHB 与 Cortex M0 CPU 通讯。AHB 是 PSoC4 内部的系统互连协议。通过它，AHB 主机可以访问存储器、外设寄存器。PSoC4 内部有两个 AHB 主机 — 一个是 ARM cortex M0 CPU 内核，另一个就是 DAP。所以，通过 DAP 外部设备可以很容易的控制整个芯片来进行编程和调试操作。

25.3 串行线调试（SWD）接口

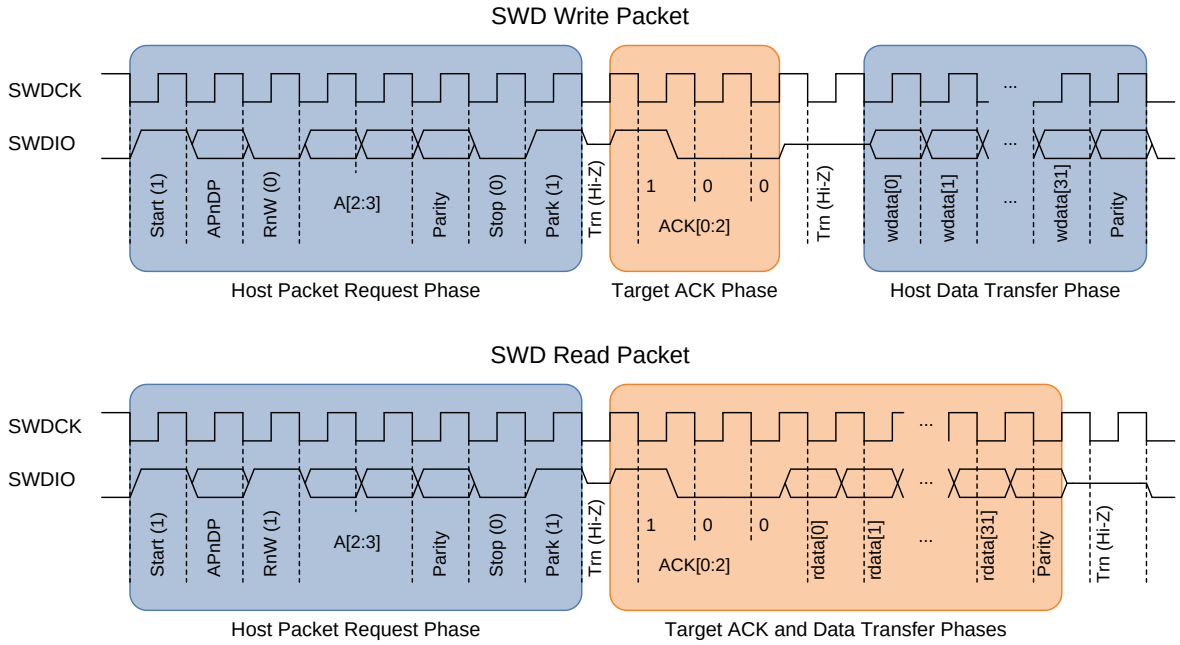
Cortex M0 支持通过 SWD 接口进行编程和调试。SWD 协议是一个基于数据包的串行协议。从引脚上看，包括一个双向数据引脚（SWDIO）和一个单相时钟引脚（SWDCK）。时钟总是由编程器驱动，而数据可以由编程器或者 PSoC4 驱动。一个完整的数据传输（一个 SWD 包）包括 46 个时钟，有三个阶段：

- **主机数据包请求** — 编程器（主机）发出一个请求到器件：PSoC 4 芯片
- **器件应答响应（ACK）** — PSoC4 发出一个应答到主机
- **数据传输阶段** — 主机和从机的双向数据传输

当 SWDIO 数据线的主机变化时，有一个过渡期（Trn），在此器件数据线浮空呈现高阻态。这个周期大概是 1/2 或者 1 ½个时钟周期，取决于不同的主机过渡情况。

SWD 读写数据包的时序图如图 25-2 所示。

图 25-2. SWD 读写数据包的时序图



SWD 数据包传输过程如下：

1. 主机数据包请求：主机驱动 SWDIO

   A. 开始位是一次传输的开始，始终为 1。

   B. APnDP 位决定了这此的传输是属于 AP 的访问（1b1）还是 DP 的访问（1b0）。

   C. 读写 RnW 位，1b1 为读 PSoC4，1b0 为写 PSoC4。

   D. ADDR 位（[3:2]）是访问端口（AP）或者调试端口（DP）的寄存器选择位。具体寄存器定义见表 25-4、表 25-5。

   注：地址位从低位开始传。

   E. 奇偶校验位用来校验 APnDP 位、RnW 位以及 ADDR 位的奇偶。如果这几位中有奇数位为 1，那么奇偶校验位为 1，否则为 0。

   如果奇偶校验失败，那么 PSoC4 就会忽略头文件；也就不会应答（ACK = 3b111）。主机将会取消编程操作，重启一次再试。

   F. 停止位永远是 0。

   G. Park 位一直 1 是。

2. ACK 位是芯片到主机的响应：目标器件（PSoC4）驱动 SWDIO

ACK[2:0] 位代表目标器件给主机的相应，表示了上一次传输的状态。详细的定义见表 25-1。

注：ACK 的传输从最低位开始。

表 25-1. SWD 传输的 ACK 响应

ACK[2:0]	SWD
OK	001
WAIT	010
FAULT	100
NO ACK	111

注意，当前的 ACK 位表示上一次传输的状态。正确（OK）说明上一个数据包是成功的。等待（WAIT）说明还需要一个数据阶段（详细说明如下）。错误（FAULT）状态说明编程操作应该停止。No ACK 即无应答。

- 对于一个等待响应，如果是个读操作，那么主机会忽略在这个数据阶段读到的数据。芯片不会驱动数据线，主机也不必检查奇偶校验位。
- 对于一个等待响应，如果是个写操作，那么这个数据阶段会被 PSoC4 忽略。但是主句还是会发出写的数据以及数据的奇偶校验位以完成这个数据包。

等待响应意味着 PSoC4 正在处理上一次的传输。主机最多可以尝试 4 次。如果 4 次以后仍然不能收到正确（OK） 响应，那么这次编程操作应停止，重新再试。

- 错误（FAULT）状态说明编程操作应该停止，通过重启再次尝试通讯。

3. 数据传输阶段：SWDIO 由主机或这目标器件驱动

A. 读写数据，低位优先

B. 数据阶段包括奇偶校验，如果有奇数个 1，结果为 1，否则结果为零。

- 对于一个读数据包，如果主机检查到有奇偶检验错误，那么编程终止，重新通讯。
- 对于一个写数据包，如果 PSoC 4 检查到主机发出的数据有奇偶检验错误，那么它会在下一个数据包产生一个错误响应（FAULT）。

根据 SWD 协议，主机可以在两个数据包中间间隔任意数量的 SWD 时钟周期（SWDIO 为低）。如果时钟不是在闲置状态的话，推荐间隔三个或更多的时钟周期。如果 SWDIO 在连续的五十个 SWDLK 时钟周期为高，SWD 接口会被重置。如果要回到闲置状态，SWDIO 需要置低。

SWD 时序

当主机驱动 SWDIO 线时，新的一位在时钟 SWDCK 下降沿的时候被写入，在时钟上升沿的时候数据被 PSoC4（目标器件）读取。当 PSoC4 驱动 SWDIO 线的时候，新的一位数据在时钟上升沿的时写入，主机在时钟下降沿的时候读取数据。

SWDIO 数据的读写可见表 25-2 和图 25-2。

表 25-2. SWDIO 位读写时序

SWD 数据包阶段	SWDCK 边沿	
	上升沿	下降沿
主机数据包请求	主机写	目标器件读
主机传输数据		
目标器件（PSoC4）应答	主机读	目标器件写
目标器件（PSoC4）数据传输		

ACK 细节：

当前的 ACK 位表示上一次传输的状态。正确（OK）说明上一个数据包是成功的。等待（WAIT）说明还需要一个数据阶段（详细说明如下）。错误（FAULT）状态说明编程操作应该停止。No ACK 即无应答。具体见表 25-3。

表 25-3. SWD 传输的 ACK 响应

ACK[2:0]	SWD
OK	3b001
WAIT	3b010
FAULT	3b100
NO ACK	3b111

- 对于一个等待响应，如果是个读操作，那么主机会忽略在这个数据阶段读到的数据。芯片不会驱动数据线，主机也不必检查奇偶校验位。
- 对于一个等待响应，如果是个写操作，那么这个数据阶段会被 PSoC4 忽略。但是主机还是会发出写的数据以及数据的奇偶校验位以完成这个数据包。
- 等待响应意味着 PSoC4 正在处理上一次的传输。主机最多可以尝试 4 次。如果 4 次以后仍然不能收到正确（OK） 响应，那么这次编程操作应停止，重新再试。
- 错误（FAULT）状态说明编程操作应该停止，通过重启再次尝试通讯。

过渡（TrN）阶段：

在数据包请求阶段和接收应答响应阶段之间，有一个过渡阶段（即第一个 TrN 周期值），它同样存在于写操作的接收应答阶段和数据阶段之间（即第二个 TrN 周期），如图 25-2 所示。在 SWD 协议中，TrN 阶段用于主机和

芯片在 SWDIO 线上改变驱动模式。在数据包请求后的第一个 TrN 阶段时，芯片开始在 SWDIO 线上输出应答信号（ACK）。这样保证了主机可以在下一个下降沿收到应答信号（ACK）。第一个 TrN 阶段值只有½ 时钟周期。第二个 TrN 阶段周期长度为 1½ 时钟周期，在 TrN 这阶段主机和 PSoC4 都不能驱动 SWDIO 线（为高阻态，如图 25-2 所示，标记为 z）。

25.4 Cortex M0 调试和访问端口（DAP）

Cortex M0 编程和调试接口 DAP 包括一个调试端口（DP）和一个访问端口（AP）。调试端口实现了与外部主机通讯的 SWD 接口协议的状态机。它也包括了访问端口的配置寄存器、DAP 标识代码等等。访问端口包括了让外部工具访问 Cortex M0DAP-AHB 接口的寄存器。一般来说，DP 中的寄存器用来做一次性的配置或者错误识别（详细说明如下），而 AP 的寄存器会在编程、调试操作中使用。完整的调试接口架构介绍请见 *ARM® Debug Interface v5 Architecture Specification*。

25.4.1 调试端口寄存器

用来编程/调试的 Cortex M0 DP 寄存器以及相应的 SWD 地址位如表 25-4 所示。在 DP 的访问时，APnDP 位始终为 0。两个地址位用来选择不同的 DP 寄存器。注意，对于同一个地址位，读和写访问不同的寄存器。DP 寄存器的更加详细信息请参见 *ARM® Debug Interface v5 Architecture Specification*。

表 25-4. 主要调试端口（DP）寄存器

寄存器	APnDP（1-bit）	地址 A[3:2]	访问（R/W）	全称	寄存器功能
ABORT	0（DP）	2'b00	0（W）	AP 停止寄存器	这个寄存器用来强制 DAP 停止，同样也用来清除错误和一些标志位。
IDCODE	0（DP）	2'b00	1（R）	识别代码寄存器	这个寄存器存放 Cortex M0 CPU 的 SWD 识别号：0x0BB11477。
CTRL/STAT	0（DP）	2'b01	X（R/W）	控制和状态寄存器	这个寄存器可以控制 DP 也可以查看 DP 的状态
SELECT	0（DP）	2'b10	0（W）	AP 选择寄存器	这个寄存器可以用来选择现在的访问端口（AP）。在 PSoC4 中，只有一个访问端口可以和 DAP AHB 通讯。
RDBUFF	0（DP）	2'b11	1（R）	读缓冲寄存器	这个寄存器存放了最后一个 AP 读操作的结果。

25.4.2 访问接口寄存器

用于编程/调试的几个主要 Cortex M0 AP 寄存器以及相应的 SWD 地址位如表 25-5 所示。在 AP 访问时 APnDP 位在此时始终为 1。两个地址位用来选择不同的 AP 寄存器。

表 25-5. 主要访问端口寄存器

寄存器	APnDP（1-bit）	地址 A[3:2]	访问（R/W）	全称	寄存器功能
CSW	1（AP）	2'b00	X（R/W）	控制和状态/字寄存器	这个寄存器配置和控制 PSoC4 内存系统的访问
TAR	1（AP）	2'b01	X（R/W）	传输地址寄存器	这个寄存器存放了需要访问的 32 位内存地址
DRW	1（AP）	2'b11	X（R/W）	数据读/写寄存器	这个寄存器保留了在 TAR 寄存器中指定的地址的读写数据。

25.5 PSoC4 的编程

对 PSoC4 编程步骤如下：

1. 外部工具获取 SWD 接口
2. 进入编程模式
3. 执行编程流程：比如说芯片 ID 检测、闪存编程、闪存验证和数据校验

下文会有简单的介绍。详细的内容，比如编程算法、时序规范以及编程的硬件连接，请参看 *PSoC 4 Device Programming Specifications*

25.5.3 端口获取

25.5.3.1 主/次 SWD 端口

器件编程的第一步是获取 PSoC4 的 SWD 端口。PSoC4 有两组引脚都可以做 SWD 接口-P3[2]（SWDIO），P3[3]（SWDCK）和 P3[6]（SWDIO），P3[7]（SWDCK）。P3[2]，P3[3] 为主 SWD 接口，P3[6]，P3[7]为次 SWD 接口。主 SWD 接口在所有的 PSoC4 封装中都有，而次 SWD 接口只有在较大的封装中有。具体引脚信息请参见 PSoC 4 芯片手册。

特权寄存器的 SWD_CONFIG 寄存器用来选择哪个接口做实际的编程/调试接口。注意，任意编程/调试过程中，只能有一个 SWD 接口。默认出厂配置编程接口是选择主 SWD 接口。所以，如果要选择次 SWD 接口，那么必须用主 SWD 接口先做一个编程，使能次 SWD 接口，此后就可以用次 SWD 接口进行编程和调试。

25.5.3.2 端口获取过程

每次编程时，编程工具会先通过外部重置引脚（XRES）做一次系统重置。然后，编程器会在这个端口获取阶段发出一个 SWD 连接序列，来连接 DAP 中的 SWD 接口。该序列的伪代码如下：

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)
```

在上面的伪代码中，SWD_LineReset（）是一个标准的重置调试端口的 ARM 命令，需要至少 50 个 SWDCK 周期（SWDIO 为高）。结束重置过程必须通过维持至少一个 SWDCK 周期的 SWDIO 为低。这个过程同步了编程器和芯片。Read_DAP（）是指读 IDCODE 寄存器。重置和读 IDCODE 这个过程会一直持续，直到收到 OK ACK 或者发生超时（1.5ms）。一旦在规定时间内收到 OK ACK 并且读取到的 IDCODE 与 Cortex M0 DAP 中的值相匹配，即认为 SWD 端口获取成功。

25.5.4 进入 SWD 编程模式

一旦获取了 SWD 端口，编程器就必须在一定时间内进入器件编程模式，这可以通过在 TEST_MODE 寄存器中设置 TEST_MODE 位来完成。在器件进入编程模式时，需要配置好调试端口。进入编程模式的伪代码和时序要求请参见：*PSoC4 device programming specification*。

25.5.5 执行 SWD 编程模式

芯片一旦进入编程模式，外部编程器就可以开始发送 SWD 数据包序列，进行编程操作，如闪存擦除、闪存编程、数据校验等等。这个编程例行过程会在另一章节 *Non Volatile Memory Programming* 中阐述。*PSoC4 device programming specifications* 中会阐述如何产生这些例行过程。

25.6 PSoC4 SWD 调试接口

PSoC4 调试接口的特性可以分为两类：侵入型调试和非侵入型调试。侵入性调试包括停止、单步、断点和数据观察点。非侵入型的调试包括指令地址分析、器件存储器（闪存、SRAM、其他外设存储器）访问。

PSoC4 中有三种主要的调试子系统，如下：

1. 调试控制和配置寄存器
2. 断点单元（BPU）— 提供了断点支持
3. 调试观察点和跟踪（DWT）— Cortex M0 仅支持数据观察点，不支持跟踪。

外部调试器通过 DAP 与 PSoC4 通讯，而 DAP 则通过 AHB 访问以上几个调试子系统中的寄存器。

不同的调试子系统下面会解释，如需完整的调试架构介绍，请参见 *ARMv6-M Architecture Reference Manual*。

25.6.1 调试控制和配置寄存器

调试控制和配置寄存器用来执行固件的调试。几个重要的控制、配置、状态寄存器和它们的关键功能如下，完整的寄存器位定义请参见：*ARMv6-M Architecture Reference Manual*。

- 调试停止寄存器和状态寄存器（DHCSR）— 这个寄存器包括了很多操作的控制位，比如使能调试，停止 CPU，进行单步操作等等，以及处理器的调试状态位。

- 错误状态寄存器（DFSR）— 这个寄存器提供了顶层的调试事件发生的原因。这包括由于 CPU 停止、断点事件、观察点事件而引起的调试事件。
- 内核寄存器选择寄存器（DCRSR）— 这个寄存器选择了编程器选择 Cortex M0 CPU 里的哪个通用寄存器做读写操作。
- 内核寄存器数据寄存器（DCRDR）— 这个寄存器存储了在 DCRSR 中选择的寄存器读出或者写入的数据。
- 调试异常和监督控制寄存器（DEMCR）— 这个寄存器包括全局调试观察点（DWT）模块的使能位、重置向量捕捉和错误异常硬捕捉使能位。

25.6.2 断点单元

断点单元（BPU）提供了断点功能。除了 4 个硬件断点，还可以用 Cortex M0 的 BKPT 指令生成任意个软件断点。BPU 中有两种寄存器。

- 断点控制寄存器 BP_CTRL：用来使能 BPU，同时存储着调试系统支持的硬件断点的个数（对于 CM0 CPU 来说是 4 个）
- 断点比较寄存器（BP_COMPx）：每个硬件断点都有一个断点比较寄存器，包括了这个断点的使能位、比较地址值、以及触发调试断点发生的匹配条件。典型应用是，当指令运行地址与比较地址值匹配时，处理器停止，产生调试断点。

25.6.3 数据观察点和跟踪（DWT）

观察点（Watchpoint）支持访问数据地址和 PC 指令地址。PSoC4 CM0 核心不支持跟踪（Trace）。DWT 支持两个观察点；PC 采样寄存器可采样当前程序指针的值，提供了一种粗略地了解代码执行情况的方法。DWT 中重要的寄存器如下所示。

- 数据观察点比较寄存器 DWT_COMPx 用于储存产生观察点的比较器所需要的比较值。每个观察点都有相对应的 DWT_COMPx 寄存器。
- 数据观察点屏蔽寄存器 DWT_MASKx 用于储存被屏蔽的地址范围的大小，这些地址不会产生观察点。
- 数据观察点功能寄存器 DWT_FUNCTIONx 用于存储实际触发观察点产生的原因。一个观察点可能是一个程序指针 PC）观察点或者数据地址读写访问观察点。这个寄存器还有一个位会标记是否有观察点产生，如产生时就会被置位。
- 数据观察点比较器 PC 采样寄存器 DWT_PCSR 用来采样当前程序指针的值，通过这个寄存器可以大致了解代码的执行情况。

25.6.4 调试 PSoC4

外部的调试器通过访问调试控制和配置寄存器、BPU 中的寄存器以及 DWT 中的寄存器来调试 PSoC4。所有的访问都是通过 SWD 接口来进行的，CM0 DAP 中的 SWD 的调试端口（SW-DP）通过 DAP-AHB 接口将 SWD 数据包转化成合适的寄存器访问。

调试的第一步是获取 SWD 端口，获取的过程包括一次 SWD 的重启和通过 SWD 接口进行的 DAP SWDID 的读取。当读到正确的 DAP SWDID 信号时，就可认为成功获取了 SWD 端口。为了顺利地调试，相应的引脚不能被其他任务占用。PORT_SEL3 寄存器可以将这些引脚设为 SWD 接口专用或者其他用途如 LCD，GPIO。如果需要调试，这些引脚必须设置成 SWD 专用；如果仅仅需要编程，那么他们可以被重新设置成其他用途，因为编程和代码执行不会同时进行。

一旦获取到 SWD 端口，外部的调试器会置位 DHCSR 寄存器中的 C_DEBUGEN 位来使能调试。然后，通过写入调试系统中的相应位来进行不同调试操作，如单步调试、停止、断点配置、观察点配置。

调试操作同时取决于整体的器件保护设置，只有开放模式支持调试，在器件保护这个章节中会详述。

在休眠或者停止模式中，外部调试器将失去与器件的连接。一旦器件重新进入到活动模式，就会重新建立连接。器件从活动模式进入到睡眠或者深度睡眠时，调试器不会断开连接。

26非易失性存储器编程

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

26.1 特性

- 支持通过调试访问端口（DAP）和 Cortex-M0 CPU 进行编程
- 支持通过 Cortex-M0 CPU 对闪存进行阻塞和非阻塞的擦除和编程操作

26.2 功能介绍

编程相关的所有操作均是通过调用“System Call”函数来实现的；存储在 SROM 中的 System Call 函数仅可在特权模式下才可被执行。用户没有权限对 SROM 代码进行读取或修改。System Call 的一个重要功能是对闪存进行编程，即 SROM 中的部分 System Call 函数与系统性能控制器（SPC）接口相互通讯来完成对闪存的编程操作。调试访问端口或 Cortex-M0 CPU 调用 System Call，是通过向相应寄存器写函数操作码和参数并请求 SROM 执行相应函数来实现的。SROM 中的代码会根据不同的函数操作码来执行相应的 System Call，并将函数执行结果写入一个寄存器。调试访问端口或 CPU 必须读取这个寄存器来判断 System Call 函数执行的结果是成功的还是失败的。

PSoC 4 支持通过调试访问端口（DAP）和 Cortex-M0 CPU 两种方式对闪存进行编程。外部编程器可通过向调试访问端口（DAP）发送命令来对 PSoC 4 中的闪存进行编程，其间的通讯是基于串行调试（SWD）协议。用户使用外部编程器对 PSoC4 进行编程的步骤请见 PSoC 4 Device Programming Specifications 文档。Cortex-M0 CPU 通过 AHB 接口访问相关寄存器也可对闪存进行编程；这种编程方式经常用于通过引导加载程序（bootloader）来更新部分闪存或在应用程序中完成其他应用需求，如更新存储在闪存中的查找表。对闪存的所有写操作，无论是来自调试访问端口还是 CPU，均是通过系统性能控制器（SPC）接口来完成的。

26.3 System Call 的实现

每个 System Call 函数均包含如下部分：

- 操作码：唯一的 8 位函数操作码
- 参数：每个 System Call 函数均包含 key1 和 key2 两个字节参数：

  key1 = 0xB6

  key2 = 0xD3 + 函数操作码

  传递这两个参数是为了确保用户代码准确调用相应的 System Call 函数。如果参数 key1 和 key2 不正确，则 SROM 不执行任何函数，并返回一个错误码。除了上面的两个参数，部分 System Call 函数还需要其他参数。
- 返回值：部分 System Call 函数在被执行完成时会返回一个值，比如执行芯片信息（Silicon ID）或校验等 System Call 函数。
- 完成状态：每个 System Call 被调用完成后，其执行状态会被存放在一个寄存器中。CPU 或调试访问端口必须读取该状态值来判断该函数是否被成功执行，如果失败，可用于分析失败原因。

26.4 阻塞和非阻塞 System Call

当通过调试访问端口调用 System Call 时，器件处于编程模式，CPU 处于停止状态。当通过 CPU 调用 System Call 函数时，基于函数执行的特点，可将其分为阻塞（Blocking System Call）和非阻塞（Non-blocking System Call）两种。

- 当调用阻塞的 System Call 时，CPU 除了执行此 System Call 外，不能并行地执行其他任何任务。当一个 System Call 函数被一个进程或线程调用时，CPU 的执行会跳转到 SROM 中 System Call 相应代码处执行；一旦执行结束，原来的线程将恢复并继续执行。
- 非阻塞 System Call 是指那些允许 CPU 并行执行一些其他的代码；System Call 执行完成后将通过一个中断告知 CPU。

这里仅有三个非阻塞的 System Call 函数：非阻塞写行 (Non-Blocking Write Row)、非阻塞编程行（Non-Blocking Program Row）和重新执行非阻塞（Resume Non-Blocking）。其他 System Call 函数均为阻塞函数。

因此，在写和编程闪存时，如还需要执行其他代码，可以使用这些非阻塞 System Call 函数。当对闪存进行擦除或编程操作时，CPU 不能运行闪存中的代码，因此，只有运行在 SRAM 中的代码才可以调用非阻塞函数。如果非阻塞函数被运行于闪存中的代码调用，运行结果不确定；当执行闪存取代码操作时，可能会返回一个总线错误，并触发一个 HardFault 异常。

系统性能控制器（SPC）是 PSoC 4 的一个重要模块，其可产生为闪存进行擦除和编程操作所需的有序高电压脉冲。当 SRAM 中的代码调用非阻塞写和非阻塞编程函数时，一旦写或编程的某个子操作完成后，系统性能控制器的定时器将产生一个中断。系统性能控制器中断服务程序调用重新执行非阻塞函数来完成 System Call 中的后续步骤。由于在执行非阻塞写或非阻塞编程函数时，CPU 仅可在 SRAM 中执行代码，所以系统性能控制器的中断服务程序必须放在 SRAM 中运行。在调用非阻塞编程函数时，系统性能控制器中断会被触发 1 次，从而调用 1 次重新执行非阻塞函数；而在调用非阻塞写函数时，系统性能控制器中断会被触发 3 次，从而调用 3 次重新执行非阻塞函数。

使用非阻塞写函数的伪示例代码，请见非阻塞 System Call 伪代码章节。

26.4.1 调用 System Call 的步骤

调用 System Call 的步骤如下：

1. 写函数参数：下面给出了两种写函数参数（KEY1，KEY2，其它参数）的方法：

   a.) 向寄存器 CPUSS_SYSARG 写入函数参数：

   此方法被用于将参数存放在寄存器 CPUSS_SYSARG 的 System Call 函数。写入 32 位寄存器 CPUSS_SYSARG 中参数的字节顺序必须符合相应函数的要求（见相关 System Call 函数表）。

   b.) 向静态随机存储器（SRAM）中写入函数参数：

   此方法被用于将参数存放在 SRAM 中的 System Call 函数。首先，用户将参数按函数要求依次写入 SRAM 中指定的连续存储单元；然后，将 SRAM 中参数的起始地址，即函数的第一个参数地址，写入寄存器 CPUSS_SYSARG。这个起始地址必须是字对齐的（32 位）；SROM 使用这个地址来读取函数的参数。

2. 写函数操作码并启动 System Call：

   用户将 8 位函数操作码写入 SYSCALL_COMMAND（CPUSS_SYSREQ[15:0]）的低 8 位；其高 8 位为 0。同时将 SYSCALL_REQ 位（CPUSS_SYSREG[31]）置 1，触发 NMI 中断，来启动 SROM 中的 System Call 函数。

3. 等待 System Call 执行完成

   当 SROM 开始执行 System Call 时，它会将 PRIVILEGED 位（CPUSS_SYSREQ[28]）置 1；此位仅可由 SROM 中的代码设置。当 System Call 被执行完成时，SYSCALL_REQ 位（CPUSS_SYSREG[31]）和 PRIVILEGED 位（CPUSS_SYSREQ[28]）会被清除。因此，如果用户使用调试访问端口编程，其会不停地轮询这两个位的状态，来判断 System Call 是否已经执行完成。最大轮询时间为 1 秒；在这个时间内，如果这两个位没有被清零，则认为操作失败，并中止其他步骤的执行。

   注意：在 system call 被执行的过程中，CPU 的应用程序不能查询这两个位，因为此时 CPU 正在执行 SROM 中代码；应用程序仅可检查 system call 执行完成的结果，即从 SROM 中返回的 system call 执行的结果。

4. 判断完成状态

   一旦 System Call 被执行完成，SYSCALL_REQ 位（CPUSS_SYSREG[31]）和 PRIVILEGED 位（CPUSS_SYSREQ[28]）会被清除；寄存器 CPUSS_SYSARG 必须被读取，以判断 System Call 被执行的结果是成功还是失败。从寄存器 CPUSS_SYSARG 读取的数值有如下两种：

   - 0xAXXXXXXX：表示 System Call 被成功执行，其中 X 表示无关值
   - 0xF00000YY：表示执行 System Call 失败，其中 YY 表示失败的原因，详情见表 2。

5. 读取返回信息值

   部分 System Call 函数会返回一些值，如芯片信息，校验等；CPU 或调试访问端口可从寄存器 CPUSS_SYSREG 和 CPUSS_SYSARG 中读取这些值。

26.5 System Call 函数

表 26-1 列出了 PSoC4 支持的 System Call 及其在不同器件保护模式下的访问权限。如需详细了解器件保护模式，请参考器件安全章节。每个 System Call 的信息将在表后被详细讲解，调用这些函数的步骤请参考“调用 System Call 的步骤”。

注意：一些 System Call 函数不能通过 CPU 调用（见表 26-1，无“✓”表示不支持）。

表 26-1. System Call 函数

System Call 函数	描述	调试访问端口的访问权限			CPU 访问权限
		开放 (Open)	保护 (Protected)	终极保护 (Kill)	
芯片信息(Silicon ID)	返回器件的芯片编号、系列编号和修订编号	✓	✓		✓
加载闪存字节(Load Flash Bytes)	将数据加载到页锁存缓冲器中，以备后续编程进入闪存行，最小单位为 1 个字节，最大为闪存行的大小（128 个字节）。	✓			✓
写行(Write Row)	先擦除，然后用页锁存缓冲器中的数据对闪存的某行进行编程	✓			✓
编程行(Program Row)	用页锁存缓冲器中的数据对闪存的某行进行编程	✓			✓
擦除所有(Erase All)	擦除闪存阵列中所有用户代码和特权闪存（sFlash）中闪存行级保护数据	✓			
校验(Checksum)	校验整个闪存（用户区和 sFlash）或校验闪存行	✓	✓		✓
写保护(Write Protection)	此功能可将闪存行级保护设置和芯片级保护设置编程到 sFlash 的第 0 行中。	✓	✓		
非阻塞写行(Non-Blocking Write Row)	擦除，然后用页锁存缓冲器中的数据对闪存的某行进行编程。在编程和擦除脉冲进行时，用户可从 SRAM 执行代码。此功能只可通过 CPU 访问。				✓
非阻塞编程行(Non-Blocking Program Row)	用页锁存缓冲器中的数据对闪存的某行进行编程。在编程和擦除脉冲进行时，用户可从 SRAM 执行代码。此功能只可通过 CPU 访问。				✓
重新执行非阻塞 (Resume Non-Blocking)	重新执行一个非阻塞写行或非阻塞编程行。此功能只可通过 CPU 访问。				✓

26.5.1 芯片信息

在调用芯片信息（Silicon ID）函数时，要将其参数写入寄存器 CPUSS_SYSARG，而非 SRAM。这个函数被成功调用后将返回 12 位的系列编号、16 位芯片编号、8 位修订编号及当前器件的保护模式；这些值分别会被写入到寄存器 CPUSS_SYSARG 和 CPUSS_SYSREQ 中。

参数：

地址	写入值	描述
寄存器 CPUSS_SYSARG		
位 [7:0]	0xB6	Key1
位[15:8]	0xD3	Key2
位[31:16]	0x0000	保留
寄存器 CPUSS_SYSREQ		
位 [15:0]	0x0000	“Silicon ID” 操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回：

地址	返回值	描述
寄存器 CPUSS_SYSARG		
位 [7:0]	Silicon ID 低 8 位	关于 Silicon ID 值，请参考芯片手册
位 [15:8]	Silicon ID 高 8 位	
位 [19:16]	次修订编号	修改编号值请参考器件编程规范
位 [23:20]	主修订编号	
位 [27:24]	0xXX	保留
位 [31:28]	0xA	成功状态码
寄存器 CPUSS_SYSREQ		
位 [11:0]	系列编号	PSoC 4 的系列编号（Family ID）是 0x093
位 [15:12]	芯片保护	请参考器件安全章节
位 [31:16]	0xXXXX	保留

26.5.2 加载闪存字节

通过调用加载闪存字节（Load Flash Bytes）函数，可将数据加载到页锁存缓冲区；然后，再通过调用其他函数将该数据编程至闪存行。加载的最小值为 1 个字节，最大值为闪存行的最大字节数（128 个字节）。加载到页锁存缓冲器的数据从输入参数“字节地址(Byte Addr)”所指定的位置处开始存放。该数据在页锁存缓冲器中会被保存，直到一个编程操作被执行（编程操作会清除页锁存器中的内容）。加载闪存字节函数的参数及将被加载到页锁存缓冲器的数据，均会被写到 SRAM 中；而在 SRAM 中的起始地址数据会被写入寄存器 CPUSS_SYSARG。

注意：参数起始地址应该是一个字对齐的地址。

参数：

地址	写入值	描述
SRAM 地址- 32’hYY (32 位、字对齐的 SRAM 地址)		
位 [7:0]	0xB6	Key1
位 [15:8]	0xD7	Key2
位 [23:16]	字节地址(Byte Addr)	页锁存缓冲器中写入数据的起始地址 0x00 – 锁存器的第 0 字节 0x7F – 锁存器的第 127 字节
位 [31:24]	闪存宏（Flash Macro）选择	0x00 – 闪存宏 0 0x01 – 闪存宏 1 **注意：**PSoC 4 仅包含 1 个闪存宏，即闪存宏 0；每个闪存行大小为 128 个字节。
SRAM 地址- 32’hYY + 0x04		
位 [7:0]	加载大小	写到页锁存缓冲器中的字节大小 0x00 – 1 个字节 0x7F – 128 个字节
位 [15:8]	0xXX	保留
位 [23:16]	0xXX	保留
位 [31:24]	0xXX	保留
SRAM 地址- 从 (32’hYY + 0x08) 至 (32’hYY + 0x08 + 加载大小)		
字节 0	数据字节[0]	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]	加载的最后一个字节
寄存器 CPUSS_SYSARG		
位 [31:0]	32’hYY	SRAM 的 32 位字对齐地址，其保存了函数的第一个参数 (Key 1)

地址	写入值	描述
寄存器 CPUSS_SYSREQ		
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	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.5.3 写行

写行（Write Row）函数会擦除、并用页锁存缓冲器中的数据对指定的闪存行进行编程。如果页锁存缓冲器中的所有数据都是 0，则跳过编程。此函数的参数保存在 SRAM 中，其起始地址会被写入寄存器 CPUSS_SYSARG 中。闪存行被编程后，此函数将清除页锁存缓冲区中的内容。

使用要求：在使用这个函数之前，用户必须先调用加载闪存字节函数。此函数仅可对非保护状态的指定闪存行执行写操作。

参数：

地址	写入值	描述
SRAM 地址: 32’hYY（32 位、字对齐的 SRAM 地址）		
位 [7:0]	0xB6	Key1
位 [15:8]	0xD8	Key2
位 [31:16]	行编号	被写入的行编号 0x0000 – 第 0 行
寄存器 CPUSS_SYSARG		
位 [31:0]	32’hYY	SRAM 的 32 位字对齐地址，其保存了第一个函数参数(Key 1)
寄存器 CPUSS_SYSREQ		
位 [15:0]	0x0005	“写行”操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回：

地址	返回值	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.5.4 编程行

编程行（Program Row）函数用页锁存缓冲器中的数据对指定的闪存行进行编程。如果页锁存缓冲器中的所有数据都是 0，则跳过编程。闪存行被编程后，此函数将清除页锁存缓冲区中的内容。在调用这个函数之前，该闪存行必须处于擦除状态。

使用要求：在使用这个函数前，用户必须先调用加载闪存字节函数，并且指定的闪存行必须处于擦除状态。此函数仅可对非保护状态的指定闪存行执行编程操作。

参数：

地址	写入值	描述
SRAM 地址: 32'hYY (32 位、字对齐的 SRAM 地址)		
位 [7:0]	0xB6	Key1
位 [15:8]	0xD9	Key2
位 [31:16]	Row ID	编程闪存行的编号. 0x0000 – 第 0 行
寄存器 CPUSS_SYSARG		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，其保存了第一个函数参数(Key 1)
寄存器 CPUSS_SYSREQ		
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	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.5.5 擦除所有

擦除所有（Erase All）函数会擦除闪存主阵列中的所有用户代码，以及每个闪存宏的特权闪存区第 0 行中的行级保护数据。

使用要求：擦除所有函数仅可通过处于编程模式的调试访问端口被调用，并且芯片处于开放(Open)模式。如果芯片处于保护（Protected）模式，用户只有通过调试访问端口调用写保护函数，将芯片的保护模式改为开放模式，才可完成擦除所有的功能（在将芯片的保护设置从保护改为开放时，会自动执行擦除所有操作）。

参数：

地址	写入值	描述
SRAM 地址: 32'hYY (32 位、字对齐的 SRAM 地址)		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDD	Key2
位 [31:16]	0xXXXX	保留
寄存器 CPUSS_SYSARG		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，其保存了函数的第一个参数(Key 1)
寄存器 CPUSS_SYSREQ		
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	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.5.6 校验

校验（Checksum）函数读取整个闪存或某个闪存行，并返回读取的所有字节的算术总和（24 位）。当对整个闪存进行校验时，校验区域其包括用户代码和 sFlash；当仅对某个闪存行进行校验时，闪存行编号将作为参数被传递。

参数：

地址	写入值	描述
寄存器 CPUSS_SYSARG		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDE	Key2
位 [31:16]	Row ID	选择需要校验的闪存行编号： 行编号 – 16 位闪存行编号 0x8000 – 对整个闪存进行校验
寄存器 CPUSS_SYSREQ		
位 [15:0]	0x000B	“校验”操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回：

地址	返回值	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:24]	0xX	保留
位 [23:0]	校验值	对选择的闪存区校验的 24 位校验值

26.5.7 写保护

写保护（Write Protection）函数会对 sFlash 中的闪存行级保护和芯片级保护字节进行编程。

如果闪存包含多个闪存宏，则需要对每个闪存宏中的闪存行级保护字节分别编程。每个保护位对应一个闪存行，每个闪存宏中行级保护字节的大小等于此闪存宏中用户代码闪存行的数量除以 8。

芯片级保护字节（1 个字节）存储在闪存宏 0 的 sFlash 的第 0 行的最后一个字节中。sFlash 的行与用户代码闪存区行大小相同。

在调用写保护函数前，用户必须先调用“加载闪存字节”函数将闪存宏的闪存行级保护字节加载到此闪存宏对应的页锁存缓冲器中。加载闪存字节函数的参数中的起始地址必须是 0，闪存宏编号对应需要编程的闪存宏，加载字节的大小为此闪存宏中闪存行级保护字节的数量。然后，调用“写保护”函数将页锁存缓冲器中的闪存行级保护字节写入相应闪存宏的 sFlash 中。对于闪存宏 0，芯片级保护设置将作为寄存器 CPUSS_SYSARG 的参数被编程。

参数：

地址	写入值	描述
寄存器 CPUSS_SYSARG		
位 [7:0]	0xB6	Key1
位 [15:8]	0xE0	Key2
位 [23:16]	芯片保护设置值	此参数仅对闪存宏 0 有效 0x01 – 开放模式 0x02 – 保护模式 0x04 – 终极保护模式
位 [31:24]	闪存宏选择	0x00 – 闪存宏 0 0x01 – 闪存宏 1
寄存器 CPUSS_SYSREQ		
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	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.5.8 非阻塞写行

非阻塞写行（Non-Blocking Write Row）函数可被 Cortex-M0 CPU 调用以非阻塞方式写某个闪存行；同时 CPU 还可在 SRAM 中执行其他代码。非阻塞 System Call 的详细介绍请见阻塞和非阻塞 System Call 章节。

非阻塞写行函数的执行 包括 3 个阶段：预编程操作、擦除和编程。预编程操会将闪存行的所有位写成‘1’，为擦除操作做准备。擦除阶段将闪存行的每位写成‘0’。在编程阶段将写的数据写到相应闪存行。在这些操作被执行的过程中，CPU 还可在 SRAM 中执行其他代码。

非阻塞写行函数被调用后，用户不能调用除重新执行非阻塞（*Resume Non-Blocking*）函数之外的任何 System Call 函数；调用重新执行非阻塞函数是完成非阻塞写操作的必要条件。每完成一个阶段，系统性能控制器会触发一个中断来调用重新执行非阻塞函数。因此，在非阻塞写行函数执行的过程中，重新执行非阻塞函数会被调用 3 次。

使用要求：在调用此函数之前，用户必须先调用加载闪存字节函数，将要编程的数据加载到页锁存缓冲器中。此外，非阻塞写行函数仅可在 SRAM 中被调用。因为，Cortex-M0 CPU 在执行擦除、编程操作时不能在闪存中执行代码。如果这个函数在闪存中被调用，则运行结果不确定；当从闪存中取指令时，可能会返回一个总线错误，并触发一个 hardfault 异常。

参数：

地址	写入值	描述
SRAM 地址：32’hYY (32 位、字对齐的 SRAM 地址)		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDA	Key2
位 [31:16]	Row ID	被写入的行编号. 0x0000 – 第 0 行
寄存器 CPUSS_SYSARG		
位 [31:0]	32’hYY	SRAM 的 32 位字对齐地址，其保存了函数的第一个参数(Key 1)
寄存器 CPUSS_SYSREQ		
位 [15:0]	0x0007	“非阻塞写行”操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回:

地址	返回值	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.5.9 非阻塞编程行

非阻塞编程行（Non-Blocking Program Row）函数可被 Cortex-M0 CPU 调用以非阻塞方式对某个闪存行进行编程；同时 CPU 还可在 SRAM 中执行其他代码。非阻塞 System Call 的详细介绍请见阻塞和非阻塞 System Call 章节。

在这个编程操作被执行的过程中，CPU 仍可在 SRAM 中执行其他代码。非阻塞编程行函数被调用后，用户不能调用除重新执行非阻塞函数 函数之外的其他任何 System Call 函数；调用重新执行非阻塞函数是完成非阻塞编程操作的必要条件。与非阻塞写行不同，非阻塞编程行仅有 1 个阶段：编程。执行完编程阶段，系统性能控制器会触发一个中断来调用重新执行非阻塞函数。因此，在非阻塞写行函数执行的过程中，重新执行非阻塞函数仅会被调用 1 次。

使用要求：在调用此函数之前，用户必须先调用加载闪存字节函数，将要编程的数据加载到页锁存缓冲器中。此外，非阻塞编程行函数仅可在 SRAM 中被调用。这是由于 Cortex-M0 CPU 在执行编程操作时不能在闪存中执行代码。如果这个函数是在闪存中被调用，则运行结果不确定；当从闪存中取指令时，可能会返回一个总线错误，并触发一个 hardfault 异常。

参数：

地址	写入值	描述
SRAM 地址: 32'hYY (32 位、字对齐的 SRAM 地址)		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDB	Key2
位 [31:16]	Row ID	写入行编号 0x0000 – 第 0 行
寄存器 CPUSS_SYSARG		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，其保存了第一个函数参数(Key 1)
寄存器 CPUSS_SYSREQ		
位 [15:0]	0x0008	"非阻塞编程行"操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回值:

地址	返回值	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.5.10 重新执行非阻塞

重新执行非阻塞（Resume Non-Blocking）函数可分别完成非阻塞写行和非阻塞编程行函数的擦除和编程阶段。如果调用非阻塞写行函数，则随后会执行 3 次重新执行非阻塞函数；如果调用非阻塞编程行，则随后会执行 1 次重新执行非阻塞函数。重新执行非阻塞函数是在系统性能控制器中断服务程序中被调用的。在编程或擦除操作的所有阶段被完成前，CPU 无法执行其他任何 System Call。关于非阻塞函数的详细使用步骤见阻塞和非阻塞 System Call 章节。

参数：

地址	写入值	描述
SRAM 地址: 32'hYY (32 位、字对齐的 SRAM 地址)		
位 [7:0]	0xB6	Key1
位 [15:8]	0xDC	Key2
位 [31:16]	0xXXXX	保留
寄存器 CPUSS_SYSARG		
位 [31:0]	32'hYY	SRAM 的 32 位字对齐地址，其保存了第一个函数参数(Key 1)
寄存器 CPUSS_SYSREQ		
位 [15:0]	0x0009	"重新执行非阻塞"操作码
位 [31:16]	0x8000	设置 SYSCALL_REQ 位

返回：

地址	返回值	描述
寄存器 CPUSS_SYSARG		
位 [31:28]	0xA	成功状态码
位 [27:0]	0xXXXXXXX	保留

26.6 System Call 返回状态值

每个 System Call 被调用结束后，一个状态码会被写入到寄存器 CPUSS_SYSARG 中：

- 0xAXXXXXXX 表示成功状态（X 表示无关值；但在部分 System Call 中，其表示有意义返回值）
- 0xF00000XX 表示失败状态（XX 表示故障码）

表 2. System Call 返回状态码

状态码（保存在寄存器 CPUSS_SYSARG 中的 32 位值）	描述
AXXXXXXXh	表示成功。“X”有两种意义： ■ 无关值。 ■ 如果该 System Call 函数需要返回值，则表示有意义的返回值
F0000001h	该 System Call 函数在当前芯片保护模式下不可用
F0000003h	页锁存缓冲区中的地址出界，页锁存地址无效
F0000004h	无效地址，提供的行编号或字节地址在可用内存之外
F0000005h	指定的行处于保护状态
F0000007h	所有非阻塞 System Call 函数都已经完成。 **注意：**在下一个非阻塞 System Call 函数被调用之前，不能调用重新执行非阻塞函数
F0000008h	一个非阻塞 System Call 函数已经被调用，并且必须通过调用重新执行非阻塞函数来完成；在完成之前，不可调用其他 System Call 函数
F0000009h	一个重新执行非阻塞函数或非阻塞写/编程行函数仍在进行中。SPC 中断服务程序会被再次执行，来调用下一个重新执行非阻塞函数
F000000Ah	校验零失败，校验和不为零
F000000Bh	操作码无效，操作码与 System Call 函数的操作码不匹配
F000000Ch	Key1 或 Key2 操作码不匹配
F000000Eh	起始地址无效

26.7 非阻塞 System Call 伪代码

下面使用非阻塞写行函数的伪代码来说明如何使用非阻塞 System Call，并在对闪存编程的同时在 SRAM 中执行代码。

```
#define REG(addr)          (*((volatile uint32 *) (addr)))
#define CM0_ISER_REG       REG( 0xE000E100 )
#define CPUSS_CONFIG_REG   REG( 0x40000000 )
#define CPUSS_SYSREQ_REG   REG( 0x40000004 )
#define CPUSS_SYSARG_REG   REG( 0x40000008 )
#define ROW_SIZE              128
//Variable to keep track of how many times SPC ISR is triggered
__ram int iStatusInt = 0x00;

__flash int main(void)
{
    DoUserStuff();
    //CM0 interrupt enable bit for spc interrupt enable
    CM0_ISER_REG |= 0x00000040;
    //Set CPUSS_CONFIG.VECS_IN_RAM since SPC ISR should be in SRAM
    CPUSS_CONFIG_REG |= 0x00000001;
```

```
    //Call non-blocking write row API
    NonBlockingWriteRow();
    //End Program
    while(1);
}
__sram void SpcIntHandler(void)
{
    /* Call Resume API */
    // Write key1, key2 parameters to SRAM
    REG( 0x20000000 ) = 0x0000DCB6;
    //Write the address of key1 to the CPUSS_SYSARG reg
    CPUSS_SYSARG_REG = 0x20000000;
    //Write the API opcode = 0x09 to the CPUSS_SYSREQ.COMMAND
    //register and assert the sysreq bit
    CPUSS_SYSREQ_REG = 0x80000009;
    iStatusInt ++; // Number of times the ISR has triggered
}
__sram void NonBlockingWriteRow(void)
{
   int iter;

   /*Load the Flash page latch with data to write*/
   //Write key1, key2, byte address,
   //and macro sel parameters to SRAM
   REG( 0x20000000 ) = 0x0000D7B6;
   //Write load size param (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 till 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;
}
```

在上面的代码中，通过向寄存器 CPUSS_CONFIG 写 0x01，设置 Cortex-M0 从 SRAM 中读取异常向量表。SRAM 的异常向量表中应有系统性能控制器中断的向量地址，其保存着在 SRAM 中定义的 *SpcIntHandler()* 函数地址。执行非阻塞编程行函数的伪代码与上面的类似，仅函数操作码、参数及应查询变量 iStatusInt 值（iStatusInt 是否等于 0x01）不同。因为在执行非阻塞编程 System Call 时，系统性能控制器中断仅被触发 1 次。