



Hi309a 通用 MPU 芯片

用户指南 04-接口（下）

文档版本 02

发布日期 2025-11-27

版权所有 © 海思技术有限公司2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

海思技术有限公司

地址：上海市青浦区虹桥港路2号101室 邮编：201721

网址：<https://www.hisilicon.com/cn/>

客户服务邮箱：support@hisilicon.com



前言

概述

本文档主要介绍Hi309a芯片接口的各项基本功能，常见应用模式的配置方法。

本文档提供Hi309a芯片接口的应用配置方法。

读者对象

本文档主要适用于以下工程师：

- 技术支持工程师
- 单板硬件开发工程师
- 单板软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。



修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2025-10-30	01	第1次正式版本发布。
2025-11-27	02	第2次正式版本发布。



手册使用约束

须知

应用芯片时，需按照芯片手册注明的功能、性能、配置步骤等信息进行软硬件开发，未按手册注明的信息进行操作，可能会出现不可预知风险。任何不按手册注明的信息进行的应用需得到芯片SE或者FAE书面认可才能进行。切勿依商用器件经验推测，否则后果自行承担，谢谢！



目录

前言.....	i
手册使用约束.....	i
1 USB.....	1
1.1 概述.....	1
1.2 关键术语.....	1
1.3 特性.....	1
1.3.1 公共特性.....	1
1.3.2 Device 特性.....	2
1.3.3 Host 特性.....	2
1.4 遵循协议.....	2
1.5 功能描述.....	2
1.5.1 编程接口.....	2
1.5.1.1 Host 模式编程接口.....	2
1.5.1.2 Device 模式编程接口.....	2
1.6 软件指南.....	16
1.6.1 AXI 突发长度的控制.....	16
1.6.2 Cache 属性配置约束.....	17
1.6.3 初始化配置.....	17
1.6.3.1 Host 初始化流程.....	17
1.6.3.2 初始化流程.....	17
1.7 DFX 应用指导.....	17
1.7.1 故障在线检测.....	18
1.7.2 眼图测试码流.....	18
2 UART.....	20
2.1 概述.....	20
2.2 特性.....	20
2.3 关键术语.....	21
2.4 遵循协议.....	21
2.5 功能描述.....	21
2.5.1 应用场景.....	22
2.5.2 接口信号.....	22
2.5.3 相关模块.....	23



2.5.4 数据结构.....	23
2.5.5 同步功能.....	24
2.5.6 主从串口通信（UART 9 bits 通信）.....	24
2.5.7 UART0 输入重定向.....	25
2.6 工作方式.....	26
2.6.1 管脚复用配置.....	26
2.6.2 时钟门控.....	26
2.6.2.1 打开时钟.....	26
2.6.2.2 关闭时钟.....	27
2.6.2.3 波特率配置.....	27
2.6.3 软复位.....	27
2.7 工作流程.....	27
2.7.1 中断或查询方式下的数据传输.....	27
2.7.1.1 初始化.....	27
2.7.1.2 数据发送.....	28
2.7.1.3 数据接收.....	28
2.7.2 DMA 方式下的数据传输.....	28
2.7.2.1 初始化.....	28
2.7.2.2 数据发送.....	29
2.7.2.3 数据接收.....	29
3 JTAG.....	30
3.1 概述.....	30
3.2 特性.....	30
3.3 关键术语.....	31
3.4 遵循协议.....	31
3.5 功能描述.....	31
3.5.1 应用框图.....	31
3.5.1.1 JTAG 应用框图.....	31
3.5.1.2 JTAG 拓扑配置.....	33
3.5.2 SOC JTAG.....	35
3.5.2.1 SOC JTAG 概述.....	35
3.5.2.2 SOC JTAG 电路结构及原理.....	36
3.5.3 ARM JTAG.....	39
3.5.4 DJTAG.....	39
3.5.4.1 DJTAG 概述.....	39
3.5.4.2 DJTAG 指令.....	40
3.5.4.3 Debug Bus.....	42
3.6 软件指南.....	44
3.6.1 初始化.....	44
3.7 注意事项.....	44
4 DJTAG Master.....	45
4.1 概述.....	45



4.2 特性.....	45
4.3 遵循协议.....	45
4.4 功能描述.....	45
4.4.1 应用框图.....	45
4.4.1.1 DJTAG Master 访问配置流程.....	46
4.4.1.1.1 DJTAG Master 访问 Debug bus 链配置流程.....	47
4.4.1.1.2 DJTAG Master 安全和非安全访问流程.....	47
4.5 注意事项.....	48
5 JTAG-AUTH.....	49
5.1 关键术语.....	49
5.2 概述.....	50
5.3 特性.....	51
5.4 功能描述.....	52
5.4.1 鉴权等级.....	52
5.4.1.1 鉴权等级真值表.....	52
5.4.1.2 鉴权结果行为描述.....	53
5.4.2 鉴权填充协议.....	54
5.4.2.1 EMSA-PKCS1-v1_5 协议.....	54
5.4.2.2 EMSA-PSS 协议.....	54
5.4.3 鉴权流程.....	56
5.4.3.1 功能模块.....	56
5.4.3.2 详细步骤.....	57
5.4.4 鉴权读指令.....	58
5.4.4.1 鉴权读指令编码.....	58
5.4.4.2 获取原始挑战信息.....	59
5.4.4.2.1 原始挑战信息工作机制.....	59
5.4.4.2.2 不鉴权情况下的处理方式.....	60
5.4.4.2.3 禁止 JTAG 使用下的处理方式.....	60
5.4.4.2.4 TRNG 或 EFUSE 未准备好的处理方式.....	60
5.4.4.3 查询状态信息.....	61
5.4.4.4 查询 Die ID.....	61
5.4.5 鉴权写指令.....	62
5.4.5.1 鉴权写指令编码.....	62
5.4.5.2 输入服务器挑战信息管理方法.....	63
5.4.5.3 输入服务器挑战信息异常情况.....	63
5.4.6 故障注入写指令.....	64
5.4.7 鉴权计算.....	64
5.4.7.1 鉴权计算流程.....	64
5.4.7.2 不鉴权情况下的处理方式.....	65
5.4.7.3 禁止 JTAG 使用下的处理方式.....	65
5.5 编程接口.....	65
5.5.1 Debug_bus_inf 编址.....	65



5.5.2 用户鉴权指令说明.....	67
5.5.3 原始挑战信息.....	68
5.5.3.1 原始挑战信息数据结构.....	68
5.5.3.2 原始挑战信息域段解析.....	69
5.5.3.3 初始化异常情况及反馈值.....	69
5.5.3.4 不鉴权情况下的反馈值.....	70
5.5.3.5 禁止 JTAG 使用下的反馈值.....	70
5.5.3.6 原始挑战信息 HASH 计算方式.....	70
5.5.4 状态信息查询.....	70
5.5.4.1 状态信息数据结构.....	71
5.5.4.2 状态信息数据结构域段解析.....	72
5.5.4.3 具体情况下的状态查询值说明.....	75
5.5.5 DIE ID 查询.....	76
5.5.6 服务器输入挑战信息.....	76
5.5.6.1 服务器输入挑战信息签名值计算约定.....	77
5.5.6.1.1 签名值的存储和表示法.....	77
5.5.6.1.2 公钥数据的存储和表示方法.....	78
5.5.6.1.3 公钥信息 HASH 计算方式.....	78
5.6 JTAG_AUTH 内部交互数据结构.....	78
5.6.1 eFuse 接口数据结构.....	78
5.6.2 TRNG 接口数据结构.....	78
5.7 应用配置.....	79
5.7.1 不使用 JTAG 鉴权.....	79
5.7.2 使用 JTAG 鉴权.....	79
5.7.3 禁止使用 JTAG.....	79
5.7.4 边界扫描使能.....	79
5.8 初始化.....	79
5.8.1 周边初始化.....	79
5.8.2 重置 JTAG 鉴权.....	79
5.8.3 DFT 测试注意事项.....	80
5.9 模块接口关系.....	80
5.10 芯片调试 DFX 功能特性描述.....	80
5.10.1 模块维测.....	81
5.10.2 软硬件接口故障.....	81
5.10.3 硬件内部数据完整性错误.....	81
5.11 性能指标.....	81
5.11.1 实际测试结果.....	81
5.11.2 吞吐率（Throughput）.....	81
5.11.2.1 延迟（Latency）.....	81
6 SPI.....	82
6.1 概述.....	82
6.2 特性.....	82



6.3 关键术语.....	83
6.4 遵循协议.....	83
6.5 功能描述.....	83
6.5.1 应用场景.....	83
6.5.2 相关模块.....	84
6.5.3 功能原理.....	84
6.5.3.1 传输协议.....	84
6.5.3.1.1 MotorolaSPI 帧格式（spo=0、sph=0）.....	85
6.5.3.1.2 MotorolaSPI 帧格式（spo=0、sph=1）.....	86
6.5.3.1.3 MotorolaSPI 帧格式（spo=1、sph=0）.....	87
6.5.3.1.4 MotorolaSPI 帧格式（spo=1、sph=1）.....	88
6.5.3.1.5 TI 同步串行帧格式.....	89
6.5.3.1.6 NationalSemiconductorMicrowire 帧格式.....	89
6.6 工作方式.....	90
6.6.1 管脚复用配置.....	90
6.7 工作流程.....	91
6.7.1 时钟门控.....	91
6.7.1.1 打开门控时钟.....	92
6.7.1.2 关闭门控时钟.....	92
6.7.2 时钟配置.....	92
6.7.3 软复位.....	92
6.7.4 初始化.....	93
6.7.5 中断或查询方式下的数据传输.....	93
6.7.6 数据发送.....	94
6.7.7 数据接收.....	95
6.7.8 三线 SPI 模式.....	96
7 ESPI.....	97
7.1 功能、特点概述.....	97
7.2 运用背景.....	97
7.3 关键术语.....	98
7.4 Master 模式流程.....	99
7.4.1 Master 模式配置访问流程.....	99
7.4.1.1 场景描述.....	99
7.4.1.2 配置流程.....	99
7.4.1.3 配置约束.....	100
7.5 Slave 模式流程.....	100
7.5.1 Slave 模式配置访问流程.....	100
7.5.1.1 场景描述.....	100
7.5.1.2 配置流程.....	100
7.5.1.3 配置约束.....	101
7.6 IP 可维可测.....	101
7.6.1 功能描述.....	101



7.6.2 操作流程.....	104
7.7 中断处理.....	104
7.7.1 中断描述.....	104
7.7.2 处理流程.....	104
7.7.2.1 直接复位芯片流程.....	105
7.7.2.2 可以恢复的错误类型处理流程.....	109
7.8 性能分析.....	116
7.8.1 性能分析.....	116
7.8.2 接口时序要求.....	117
7.8.2.1 ESPI_Slave 写接口时序.....	117
7.9 使用约束.....	118
8 HACSPI.....	119
8.1 概述.....	119
8.2 特性.....	119
8.3 关键术语.....	120
8.4 功能描述.....	120
8.4.1 应用场景.....	120
8.4.2 功能原理.....	120
8.5 工作流程.....	121
8.5.1 管脚复用配置.....	121
8.5.2 HACSPI 初始化配置.....	121
8.5.3 HACSPI 其他.....	123
8.5.3.1 回读.....	123
8.5.3.2 超时告警.....	123
8.5.3.3 时序操作完成指示.....	123
8.5.3.4 动态操作.....	123
8.5.3.5 CS 片选切换.....	123
9 I2C.....	125
9.1 概述.....	125
9.2 特性.....	125
9.3 关键术语.....	125
9.4 遵循协议.....	126
9.5 功能描述.....	126
9.5.1 应用场景.....	126
9.5.2 相关模块.....	127
9.5.3 数据结构.....	128
9.6 初始化流程.....	129
9.6.1 初始化.....	129
9.6.1.1 I2C 从模式复位后初始化.....	129
9.6.1.2 I2C 主模式复位后初始化.....	129
9.6.2 时钟门控.....	130
9.6.2.1 打开时钟.....	130



9.6.2.1.1 关闭时钟.....	130
9.6.3 软复位.....	131
9.6.4 I2C disable 的流程.....	131
9.7 工作方式.....	131
9.7.1 管脚复用配置.....	131
9.7.2 时钟配置.....	132
9.7.3 I2Cx_DELAY.....	133
9.8 工作流程.....	133
9.8.1 中断或查询方式下的数据传输.....	133
9.8.1.1 数据发送.....	133
9.8.1.2 数据接收.....	133
9.8.2 对端器件挂死的处理.....	134
9.8.2.1 异常挂死场景.....	134
9.8.2.2 解决方案一.....	134
9.8.2.3 解决方案二.....	135
9.8.3 滤波配置.....	135
9.9 注意事项.....	135
10 RTC.....	136
10.1 概述.....	136
10.2 特性.....	136
10.3 关键术语和缩略语.....	138
10.4 功能描述.....	139
10.4.1 应用框图.....	139
10.4.2 相关模块.....	140
10.4.3 数据结构.....	140
10.4.4 同步时间戳.....	142
10.4.4.1 1PPS+I2C TOD.....	143
10.4.4.2 1PPS+ TOD.....	144
10.4.5 RTC 内部时间更新.....	146
10.4.6 MAC 时间戳请求.....	147
10.4.7 48bit 时间戳产生.....	147
10.4.8 64bit 串行时间戳.....	147
10.4.9 T1~T4CF 的传送.....	148
10.4.9.1 T1~T4、CF 更新.....	150
10.4.10 DFX.....	151
10.5 工作方式.....	151
10.5.1 时钟和复位.....	151
10.6 初始化.....	151
10.6.1 软复位.....	152
10.6.2 本地时间同步模式.....	152
10.7 性能.....	153
10.7.1 精度.....	153



10.8 调试（模块级错误处理）	153
10.8.1 关键状态	153
10.8.2 诊断计数器	153
10.9 注意事项	154
10.10 附顶层接口信号描述	154
11 MDIO	156
11.1 MDIO	156
11.1.1 概述	156
11.1.2 特性	156
11.1.3 关键术语	156
11.1.4 遵循协议	157
11.1.5 功能描述	157
11.1.5.1 应用框图	157
11.1.5.2 相关模块	158
11.1.5.3 模块框图	158
11.1.5.4 数据结构	158
11.1.5.4.1 Clause22 的数据帧格式	158
11.1.5.4.2 Clause45 的数据帧格式	159
11.1.5.5 接口信号	160
11.1.5.6 接口读时序	160
11.1.5.7 接口写时序	160
11.1.5.8 PHY 状态自动查询	161
11.1.5.9 中断	161
11.1.5.10 Clause22 章和 Clause45 章模式配置	161
11.1.6 软件指南	161
11.1.6.1 Clause22 PHY 寄存器读操作	161
11.1.6.2 Clause22 PHY 寄存器写操作	162
11.1.6.3 Clause45 PHY 寄存器读数据操作	162
11.1.6.4 Clause45 PHY 寄存器写数据操作	164
11.1.6.5 Clause45 PHY 寄存器地址自增读数据操作	165
11.1.6.6 MDIO 使用约束	166
11.1.7 初始化	166
11.1.7.1 MDIO 管脚复用配置	166
11.1.7.2 模块初始化	166
11.1.7.3 时钟门控	167
11.1.7.3.1 打开时钟	167
11.1.7.3.2 关闭时钟	167
11.1.7.4 复位	167
12 PMC	168
12.1 关键术语	168
12.2 概述	168
12.2.1 309a 特性	169



12.3 接口信号.....	170
12.4 功能描述.....	170
12.4.1 应用框图.....	170
12.4.2 TDM 模块结构.....	172
12.4.3 相关模块.....	172
12.4.4 板级应用.....	172
12.4.4.1 互联模式.....	172
12.4.4.2 时分复用模式.....	173
12.4.5 存储结构.....	174
12.4.6 存储原理.....	176
12.4.6.1 寄存器配置.....	176
12.4.6.2 TSM 配置.....	179
12.4.6.3 TSA 配置.....	183
12.4.6.4 通道参数配置.....	185
12.4.6.5 帧统计信息.....	194
12.4.7 缓存描述符.....	195
12.4.7.1 组织框图.....	195
12.4.7.2 发送 BD.....	196
12.4.7.3 接收 BD.....	197
12.4.7.4 BD 应用.....	201
12.4.8 接口时序.....	202
12.4.9 中断处理.....	205
12.4.9.1 中断概述.....	205
12.4.9.2 异常中断处理.....	210
12.4.9.3 通道中断处理.....	211
12.4.9.4 中断处理流程图.....	211
12.4.10 大小端工作模式.....	212
12.4.10.1 DDR 访问的大小端控制.....	213
12.4.10.2 数据流框图.....	213
12.4.10.3 大端模式（Big-Endian）.....	214
12.4.10.4 小端模式（Little-Endian）.....	214
12.4.11 性能约束.....	215
12.4.12 32M TDM 模式.....	216
12.4.13 32M TDM 接口功能时序.....	217
12.5 工作方式.....	219
12.5.1 管脚复用配置.....	219
12.5.2 初始化.....	219
12.5.3 软复位.....	222
12.5.4 时钟门控.....	223
12.5.5 报文发送.....	223
12.5.6 报文接收.....	225
12.5.7 通道重新启动.....	226



12.5.8 通道使能.....	228
12.5.9 网表切换.....	229
12.5.10 多时隙应用.....	230
12.5.11 BD 溢出异常处理.....	231
12.5.12 总线错误与 RAM 失效处理.....	232
12.6 调试（Debug）.....	233
12.6.1 外环回测试模式.....	233
12.6.2 通道环回测试模式.....	234
12.6.3 心跳检测.....	234
12.7 注意事项.....	234
13 CodecTDM.....	235
13.1 概述.....	235
13.2 关键术语.....	235
13.3 功能特性.....	236
13.4 功能描述.....	237
13.4.1 应用框图.....	237
13.4.2 相关接口.....	238
13.4.3 内部存储功能.....	238
13.4.4 数据交叉功能.....	240
13.5 编程接口.....	242
13.5.1 软复位.....	242
13.5.2 接收数据写 DDR.....	243
13.5.3 发送数据读 DDR.....	244
13.6 应用配置.....	244
13.7 初始化.....	244
13.8 注意事项.....	245
13.9 DFX.....	245
13.10 性能指标.....	246
13.10.1 吞吐率（Throughput）.....	246
13.10.2 延迟（Latency）.....	246
14 HDLC.....	247
14.1 关键术语.....	247
14.2 概述.....	247
14.3 特性.....	247
14.4 信号描述.....	248
14.5 功能描述.....	248
14.5.1 结构框图.....	248
14.5.2 HDLC 通道协议处理.....	250
14.5.3 通道仲裁.....	251
14.5.4 中断结构.....	252
14.5.5 BD 数据结构.....	253
14.5.5.1 BD 管理结构.....	253



14.5.5.2 发送 BD 表.....	254
14.5.5.3 接收 BD 表.....	255
14.5.6 标准 HDLC 帧与简化 HDLC 帧.....	256
14.6 工作方式.....	257
14.6.1 管脚复用配置.....	257
14.6.2 软复位.....	257
14.6.3 操作流程.....	257
14.6.3.1 整体工作流程.....	257
14.6.3.2 标准复位.....	259
14.6.3.3 初始化配置.....	260
14.6.3.4 启动.....	261
14.6.3.5 交互机制.....	261
14.6.3.5.1 CPU 发送数据交互.....	261
14.6.3.5.2 HDLC 发送数据交互.....	262
14.6.3.5.3 CPU 接收数据交互.....	263
14.6.3.5.4 HDLC 接收数据交互.....	263
14.6.3.6 完成条件.....	264
14.6.4 发送通道使用说明.....	265
14.6.4.1 发送源缓冲区管理.....	265
14.6.4.2 发送中断管理.....	267
14.6.4.3 发送命令.....	268
14.6.5 接收通道使用说明.....	268
14.6.5.1 接收通道 BD 配置说明.....	268
14.6.5.2 接收中断管理.....	271
14.6.5.3 接收命令.....	271
14.6.6 测试功能使用说明.....	272
14.6.6.1 协议处理前环回.....	273
14.6.6.2 协议处理后环回.....	274
14.6.6.3 端口环回.....	274
14.6.6.4 PN18 测试.....	275
14.6.6.5 单通道数据采样.....	275
14.6.6.6 实时业务数据采样.....	275
14.6.7 中断操作维护.....	276
14.6.7.1 通道 8 中断处理.....	276
14.6.7.2 发送源缓冲完成中断/发送帧完成中断处理建议.....	277
14.6.7.3 发送 BD 未准备好中断处理建议.....	278
14.6.7.4 发送 Abort 中断/发送超长帧中断处理建议.....	278
14.6.7.5 接收目的缓冲完成中断/接收帧完成中断处理建议.....	278
14.6.7.6 接收 BD 未准备好中断.....	279
14.6.7.7 通道仲裁中断处理.....	279
14.6.7.8 通道超时中断.....	280
14.7 HDLCBUS 模式.....	280



14.7.1 概述.....	280
14.7.2 特性.....	280
14.7.2.1 整体特性.....	280
14.7.2.2 配置特性.....	281
14.7.2.3 功能特性.....	281
14.7.3 接口信号.....	282
14.7.4 功能描述.....	282
14.7.4.1 功能概述.....	282
14.7.4.2 工作规则.....	282
14.7.5 寄存器配置说明.....	283
14.7.5.1 配置寄存器.....	283
14.7.5.2 Debug 寄存器.....	293
15 GPIO.....	297
15.1 关键术语.....	297
15.2 概述.....	297
15.3 特性.....	298
15.4 遵循协议.....	298
15.5 功能描述.....	298
15.5.1 应用场景.....	298
15.5.2 相关模块.....	299
15.5.3 功能原理.....	299
15.6 工作流程.....	300
15.6.1 初始化.....	300
15.6.2 管脚复用配置.....	300
15.6.3 软复位.....	300
15.6.4 时钟门控.....	300
15.6.4.1 打开时钟.....	300
15.6.4.2 关闭时钟.....	301
15.6.5 模式选择.....	301
15.6.6 输入输出配置.....	301
15.6.7 中断处理.....	302
15.6.7.1 自研模式或 syn 模式（syn 寄存器）.....	302
15.6.7.2 syn 模式（多核并发寄存器）.....	303
15.6.8 GPIO 毛刺过滤.....	303
15.6.8.1 GPIO 消抖动.....	303
16 QDMA.....	305
16.1 IP 功能及特性描述.....	305
16.2 应用范围.....	309
16.3 协议或包格式说明.....	310
16.4 IP 配置流程.....	319
16.5 寄存器写保护设置.....	319
16.5.1 场景描述.....	319



16.5.2 配置流程.....	319
16.5.3 配置约束.....	320
16.6 寄存器多模隔离设置.....	321
16.6.1 场景描述.....	321
16.6.2 寄存器多模隔离使能配置流程.....	321
16.6.3 寄存器多模隔离关闭配置流程.....	324
16.6.4 配置约束.....	325
16.7 Queue 的初始化.....	325
16.7.1 场景描述.....	326
16.7.2 配置流程.....	326
16.7.3 配置约束.....	328
16.8 Queue 的任务配置.....	328
16.8.1 普通 BD 任务场景描述.....	328
16.8.2 普通 BD 任务配置流程.....	329
16.8.3 配置约束.....	330
16.8.4 配置约束.....	330
16.8.5 实例——普通 BD 一维 M2M 搬移.....	330
16.8.6 实例——普通 BD 一维 M2F 搬移.....	331
16.8.7 实例——普通 BD 一维 F2M 搬移.....	332
16.8.8 实例——普通 BD 一维 F2F 搬移.....	333
16.8.9 实例——普通 BD 三维 M2M A-sync 搬移.....	334
16.8.10 实例——普通 BD 三维 F2M AB-sync 搬移.....	335
16.8.11 实例——普通 BD 清固定值任务.....	337
16.8.12 实例——普通 BD 定浮点转换任务.....	338
16.8.13 实例——普通 BD 链接任务.....	339
16.8.14 实例——普通 BD 压缩解压缩任务.....	342
16.9 Group ID 设置.....	343
16.9.1 场景描述.....	343
16.9.2 配置流程.....	344
16.9.3 配置约束.....	344
16.10 QoS 设置.....	345
16.10.1 场景描述.....	345
16.10.2 配置流程.....	345
16.10.3 配置约束.....	346
16.11 USER 设置.....	346
16.11.1 场景描述.....	346
16.11.2 配置流程.....	346
16.11.3 配置约束.....	346
16.12 QDMA AXI 信号输出路径选择.....	347
16.12.1 场景描述.....	347
16.12.2 配置流程.....	347
16.12.3 配置约束.....	347



16.13 Outstanding 设置.....	347
16.13.1 场景描述.....	347
16.13.2 配置流程.....	347
16.13.3 配置约束.....	348
16.14 读命令间隔配置.....	349
16.14.1 场景描述.....	349
16.14.2 配置流程.....	349
16.14.3 配置约束.....	350
16.15 处理能力隔离配置.....	350
16.15.1 场景描述.....	350
16.15.2 配置流程.....	350
16.15.3 配置约束.....	353
16.16 Stream ID 设置.....	353
16.16.1 场景描述.....	353
16.16.2 配置流程.....	353
16.16.3 配置约束.....	354
16.17 IP 可维可测.....	354
16.18 Queue 的任务查询.....	354
16.18.1 场景描述.....	355
16.18.2 配置流程.....	355
16.18.3 配置约束.....	358
16.19 Queue 的任务撤销.....	358
16.19.1 场景描述.....	358
16.19.2 配置流程.....	358
16.19.3 配置约束.....	359
16.20 QDMAC 的状态查询.....	359
16.20.1 场景描述.....	359
16.20.2 配置流程.....	359
16.20.3 配置约束.....	366
16.21 QDMAC 的 HALT 操作.....	366
16.21.1 场景描述.....	366
16.21.2 配置流程.....	366
16.21.3 配置约束.....	368
16.22 IP 低功耗.....	368
16.23 Memory 低功耗.....	368
16.23.1 场景描述.....	368
16.23.2 配置流程.....	370
16.23.3 配置约束.....	372
16.24 QDMAC 低功耗接口.....	372
16.24.1 场景描述.....	372
16.24.2 配置流程.....	373
16.24.3 配置约束.....	374



16.25 时钟复位.....	374
16.26 时钟复位步骤.....	374
16.27 中断处理.....	375
16.28 QDMAC 完成中断.....	375
16.28.1 中断描述.....	375
16.28.2 处理流程.....	375
16.29 QDMAC 告警中断.....	378
16.29.1 中断描述.....	378
16.29.2 处理流程.....	379
16.30 性能分析.....	384
16.31 性能分析.....	384
16.31.1 性能分析依据.....	384
16.31.2 性能分析数据.....	384
16.32 接口时序要求.....	385
17 CAN.....	386
17.1 功能描述（feature）.....	386



插图目录

图 1-1 TRB 结构.....	2
图 1-2 PORTPMSC 的详细描述.....	19
图 2-1 UART 应用框图.....	22
图 2-2 UART 帧格式.....	23
图 2-3 UART0 重定向功能.....	25
图 3-1 Nimbus JTAG 拓扑结构图.....	32
图 3-2 Boundary Scan Chips on Board.....	35
图 3-3 JTAG 硬件结构.....	36
图 3-4 边界扫描寄存器.....	37
图 3-5 TAP 控制器的状态机跳转图.....	38
图 3-6 统一调试、测试结构图.....	40
图 3-7 DJTAG 寄存器格式.....	42
图 3-8 Debug Bus 接口图.....	44
图 4-1 DJTAG Master 数据流图.....	46
图 4-2 DJTAG 链.....	48
图 5-1 JTAG 鉴权流程.....	50
图 5-2 JTAG 鉴权框图.....	51
图 5-3 EMSA-PSS 协议.....	55
图 5-4 JTAG-AUTH 鉴权流程详细框图.....	56
图 5-5 获取原始挑战信息示意图.....	59
图 5-6 查询状态信息示意图.....	61
图 5-7 鉴权写指令示意图.....	62
图 5-8 鉴权计算示意图.....	64
图 5-9 状态查询数据结构.....	71
图 5-10 JTAG-AUTH 所需的 2048 比特签名值 WORD 对应图.....	78
图 5-11 JTAG 鉴权时钟关系.....	80
图 6-1 SPI 应用框图.....	84
图 6-2 MotorolaSPI 单帧帧格式（spo=0、sph=0）.....	85
图 6-3 MotorolaSPI 连续帧帧格式（spo=0、sph=0）.....	85
图 6-4 MotorolaSPI 单帧帧格式（spo=0、sph=1）.....	86
图 6-5 MotorolaSPI 连续帧帧格式（spo=0、sph=1）.....	86
图 6-6 MotorolaSPI 单帧帧格式（spo=1、sph=0）.....	87
图 6-7 MotorolaSPI 连续帧帧格式（spo=1、sph=0）.....	87



图 6-8 MotorolaSPI 单帧帧格式（spo=1、sph=1）	88
图 6-9 MotorolaSPI 连续帧帧格式（spo=1、sph=1）	88
图 6-10 TI 同步串行单帧帧格式	89
图 6-11 TI 同步串行连续帧帧格式	89
图 6-12 NationalSemiconductorMicrowire 单帧帧格式	89
图 6-13 NationalSemiconductorMicrowire 连续帧帧格式	90
图 6-14 主机模式工作流程图	94
图 6-15 SPI 发送数据流程图	95
图 6-16 SPI 接收数据流程图	96
图 7-1 ESPI 应用框图示例	98
图 7-2 ESPI 中断处理框图	104
图 7-3 ESPI 写接口时序 8 线模式	117
图 7-4 ESPI 读接口时序 8 线模式	117
图 7-5 ESPI 写接口时序 4 线模式	117
图 7-6 ESPI 读接口时序 4 线模式	118
图 7-7 U_ASYNC_2FF 参考图	118
图 8-1 HACSPI 应用框图	120
图 8-2 数据 RAM 结构	121
图 8-3 参数 RAM 结构	122
图 8-4 SP_REQ_CS_SEL 寄存器说明	123
图 9-1 I2C 应用框图	127
图 9-2 I2C 数据传输时序图	128
图 9-3 I2C 数据传输帧格式图	128
图 9-4 I2C START 传输时序	134
图 10-1 RTC 功能框图	139
图 10-2 RTC 功能框图	140
图 10-3 1588 80b Timestamp Format	140
图 10-4 Timestamp Conversion 80-bit → 32-bit	141
图 10-5 Timestamp Conversion 32-bit → 80-bit	141
图 10-6 1588 Timestamp Download Format	141
图 10-7 47b/48b Timestamp Format	142
图 10-8 串行 64bit 时间戳	142
图 10-9 RTC 功能框图	142
图 10-10 I2C 接口数据格式	143
图 10-11 时戳格式	143
图 10-12 I2C 数据下发帧格式	143
图 10-13 2kHz 调制图	144
图 10-14 标准 PPS 信号与 P300 内部 PPS 信号的时序对应关系	144
图 10-15 64b Timestamp Format	147
图 10-16 串行 64bit 时间戳传输示意图	147
图 10-17 上送时间戳传送格式	150
图 11-1 MDIO 应用图	157



图 11-2 MDIO 接口读时序.....	160
图 11-3 MDIO 接口写时序.....	160
图 12-1 PMC 应用图.....	171
图 12-2 PMC 结构框图.....	172
图 12-3 PMC 直连模式之 4 线连接图.....	173
图 12-4 PMC 同相共享模式.....	173
图 12-5 PMC 独立共享模式.....	174
图 12-6 PMC 存储结构图.....	175
图 12-7 TSM 静态路由模式.....	180
图 12-8 TSM 表动态模式图.....	180
图 12-9 TSM 动态模式切换控制示意图.....	181
图 12-10 TSM 应用示意图.....	182
图 12-11 TSA 表静态模式.....	183
图 12-12 TSA 表动态模式.....	183
图 12-13 TSM 和 TSA 表动态切换模式示意图.....	184
图 12-14 TSA 应用示意图.....	185
图 12-15 发送方向 TX_ADDR 配置示意图.....	187
图 12-16 帧间 7E 配置 (idle_mode=0, pad=2, num_of_flag=4)	189
图 12-17 帧间 7E 配置 (idle_mode=1, pad=2, num_of_flag=2)	190
图 12-18 接收方向 RX_ADDR 配置示意图.....	191
图 12-19 PMC 帧统计计数器存储结构.....	194
图 12-20 BD 应用结构示意图.....	196
图 12-21 接收帧错误处理流程图.....	200
图 12-22 字节非对齐数据存储图.....	201
图 12-23 BD 应用示意图.....	201
图 12-24 PMC 接口时序 (clock_edge 为 0, 数据延迟为 1)	204
图 12-25 PMC 接口时序 (clock_edge 为 1, 数据延迟为 1.....	204
图 12-26 PMC 零延迟接口时序 (frame_edge 为 0, clock_edge 为 0)	205
图 12-27 PMC 零延迟接口时序 (frame_edge 为 0, clock_edge 为 1)	205
图 12-28 PMC 零延迟接口时序 (frame_edge 为 1, clock_edge 为 0)	205
图 12-29 PMC 零延迟接口时序 (frame_edge 为 1, clock_edge 为 1)	205
图 12-30 PMC 中断存储结构图.....	208
图 12-31 异常中断处理流程图.....	211
图 12-32 通道中断处理流程图.....	212
图 12-33 TDM 数据处理图.....	214
图 12-34 PMC 连续时隙性能示意图.....	215
图 12-35 PMC 间插时隙性能示意图.....	215
图 12-36 TDM 性能约束示意图.....	216
图 12-37 TDM 32M 的工作原理.....	217
图 12-38 TDM 接口时序图 0.....	218
图 12-39 TDM 接口时序图 1.....	218
图 12-40 PMC 报文发送软件配置流程.....	224



图 12-41 PMC 接收报文流程图.....	225
图 12-42 PMC 发送通道重启流程图.....	227
图 12-43 PMC 接收 BD 溢出处理流程图.....	231
图 12-44 PMC 全局异常中断处理流程图.....	232
图 12-45 外环回测模式配置流程图.....	233
图 13-1 CodecTDM 应用图.....	237
图 13-2 CodecTDM 接口 IO 时序图.....	238
图 13-3 CodecTDM 接口内部 TDM 数据乒乓 MEM 存储格式.....	239
图 13-4 MEM 存储结构图.....	239
图 13-5 DDR 缓存块数据字节序.....	240
图 13-6 CodecTDM 接口内部数据流.....	242
图 14-1 HDLC 的应用框图.....	249
图 14-2 HDLC 分层中断结构.....	252
图 14-3 HDLCBD 管理数据缓冲区.....	253
图 14-4 HDLC 标准帧.....	256
图 14-5 HDLC 简化帧.....	256
图 14-6 HDLCOM 接口全双工工作时序.....	256
图 14-7 HDLC 整体工作流程.....	259
图 14-8 HDLC 测试功能位置说明图.....	273
图 14-9 HDLC 通道 8 中断树结构示意图.....	276
图 14-10 HDLC 发送通道指针关系.....	277
图 14-11 HDLC 通道仲裁中断树结构.....	279
图 14-12 HDLCBUS 模式应用框图.....	282
图 15-1 GPIO 应用框图.....	299
图 16-1 QDMAC 模块内部结构框图.....	307
图 16-2 QDMAC 工作数据流图.....	308
图 16-3 QDMAC 应用框图.....	309
图 16-4 QDMAC 的普通 BD 结构图.....	310
图 16-5 QDMAC 的消息结构图.....	310
图 16-6 Queue 任务配置流程图.....	328
图 16-7 一维 M2M 搬移示意图.....	330
图 16-8 一维 M2F 搬移示意图.....	331
图 16-9 一维 F2M 搬移示意图.....	332
图 16-10 一维 F2F 搬移示意图.....	333
图 16-11 三维 M2M A-sync 搬移示意图.....	334
图 16-12 三维 F2M AB-sync 搬移示意图.....	336
图 16-13 清固定值任务搬移示意图.....	337
图 16-14 定浮点转换任务搬移示意图.....	338
图 16-15 BD 链接任务搬移示意图.....	340
图 16-16 压缩解压缩任务搬移示意图.....	342
图 16-17 SC_DMACH_USER_CFG0 和 SC_DMACH_USER_CFG1 寄存器.....	346
图 16-18 剩余任务查询流程图.....	355



图 16-19 Memory 低功耗处理流程图.....	369
图 16-20 Memory 低功耗控制时序图.....	370
图 16-21 完成中断结构示意图.....	375
图 16-22 告警中断结构示意图.....	379



表格目录

表 1-1 USB 术语表.....	1
表 1-2 TRB 域段描述.....	3
表 1-3 Device 通用命令类型.....	5
表 1-4 DEPCFG.....	7
表 1-5 DEPXFERCFG.....	9
表 1-6 DEPSTRXFER.....	11
表 1-7 DEPENDXFER.....	12
表 1-8 Device Endpoint-n Events.....	13
表 1-9 Device-Specific Event.....	15
表 1-10 PORTSC 寄存器.....	18
表 2-1 波特率误差.....	21
表 2-2 UART 关键术语描述.....	21
表 2-3 UART 接口信号描述.....	22
表 2-4 UART 发送方向奇偶校验功能.....	24
表 2-5 UART 接收方向奇偶校验功能.....	25
表 3-1 JTAG 关键术语描述.....	31
表 3-2 JTAG Chain 链配置表.....	33
表 3-3 DJTAG 指令.....	40
表 3-4 仿真器访问 DJTAG 寄存器读写命令格式说明.....	42
表 3-5 仿真器访问 DJTAG 寄存器读返回数据格式说明.....	43
表 4-1 DJTAG Master 寄存器组.....	46
表 5-1 JTAG-AUTH 关键术语描述.....	49
表 5-2 鉴权等级真值表.....	52
表 5-3 JTAG-TAP 与鉴权行为关系.....	53
表 5-4 输入服务器挑战信息异常情况表.....	63
表 5-5 四组 Debug_Bus_Interface 接口.....	65
表 5-6 JTAG 指令.....	66
表 5-7 Debug Bus JTAG/TDRE 寄存器访问数据格式.....	67
表 5-8 鉴权指令编址说明.....	67
表 5-9 原始挑战信息数据结构.....	68
表 5-10 原始挑战信息域段解析.....	69
表 5-11 初始化异常情况及反馈值.....	69
表 5-12 状态查询数据对应地址.....	71



表 5-13 状态信息数据结构域段解析.....	72
表 5-14 状态信息 WORD0 在不同阶段访问的反馈值.....	75
表 5-15 DIE ID 信息查询.....	76
表 5-16 服务器输入鉴权挑战信息.....	77
表 5-17 算法核计算 延迟（Latency）.....	81
表 6-1 SPI 关键术语描述.....	83
表 7-1 ESPI 关键术语描述.....	98
表 7-2 ESPI_CTRL 寄存器.....	101
表 7-3 ESPI_STATUS.....	102
表 7-4 ESPI_FAULT_STATUS 寄存器.....	103
表 7-5 ESPI_FAULT_ADDR 寄存器.....	103
表 7-6 ESPI_SLV_CRC_CNT 寄存器.....	103
表 7-7 ESPI_SLV_WR_DISCARD_CNT 寄存器.....	103
表 7-8 ESPI_INT_MASK 中断屏蔽寄存器.....	105
表 7-9 ESPI_ABNOR_INTR_STA 寄存器.....	106
表 7-10 ESPI_CTRL 寄存器.....	107
表 7-11 ESPI_ABNOR_IMSK 中断屏蔽寄存器.....	109
表 7-12 ESPI_ABNOR_INTR_STAT 寄存器.....	110
表 7-13 ESPI_FAULT_ADDR 寄存器.....	112
表 7-14 ESPI_ABNOR_INTR_RAW 中断原始状态寄存器.....	112
表 8-1 SPI 关键术语描述.....	120
表 9-1 I2C 关键术语描述.....	126
表 9-2 I2C_SS_SCL_HCNT 典型配置值.....	132
表 9-3 I2C_SS_SCL_LCNT 典型配置值.....	132
表 9-4 I2C_FS_SCL_HCNT 典型配置值.....	133
表 9-5 I2C_FS_SCL_LCNT 典型配置值.....	133
表 10-1 RTC 关键术语和缩略语.....	138
表 10-2 1588 Timestamp Upload Format.....	148
表 10-3 RTC 模块接口信号.....	154
表 11-1 MDIO 关键术语描述.....	157
表 11-2 MDIO 接口的 Clause22 数据帧格式.....	158
表 11-3 MDIO 接口的 Clause45 数据帧格式.....	159
表 11-4 MDIO 接口信号描述.....	160
表 12-1 关键术语描述.....	168
表 12-2 PMC 接口信号描述.....	170
表 12-3 PMC_TSMCFG 寄存器描述.....	176
表 12-4 PMC_TPTRCFG 寄存器描述.....	177
表 12-5 PMC_RPTRCFG 寄存器描述.....	178
表 12-6 PMC_INTBASE 寄存器描述.....	178
表 12-7 PMC_BASE 寄存器描述.....	179
表 12-8 TSM 表项描述.....	181
表 12-9 TSA 表项描述.....	184



表 12-10 HDLC 通道参数表.....	185
表 12-11 TX_ADDR 寄存器描述.....	186
表 12-12 CHNL_MODE 寄存器描述.....	187
表 12-13 TXSTATE 寄存器描述.....	190
表 12-14 RX_ADDR 寄存器描述.....	191
表 12-15 RX_MODE 寄存器描述.....	192
表 12-16 PMC_RFRMCFG 寄存器描述.....	192
表 12-17 RXSTATE 寄存器描述.....	193
表 12-18 PMC 接收帧计数器描述.....	194
表 12-19 PMC 发送帧计数器描述.....	195
表 12-20 TxBD 描述.....	196
表 12-21 RxBD 描述.....	198
表 12-22 PMC_MRBL 寄存器描述.....	200
表 12-23 PMC_CONFIG 寄存器描述.....	202
表 12-24 PMC_INT 寄存器描述.....	206
表 12-25 PMC 通道中断（INT_ENTRY）描述.....	207
表 12-26 PMC_INTCTRL 寄存器描述.....	208
表 12-27 PMC_INTRD 寄存器描述.....	211
表 12-28 32 位数据小端存储.....	213
表 12-29 32 位数据大端存储.....	213
表 13-1 CodecTDM 关键术语描述.....	235
表 13-2 TDM 接口信号描述.....	238
表 13-3 时隙交叉表项 TSCT 格式.....	241
表 13-4 时隙交叉表项 TSCT 描述.....	241
表 13-5 CTDM_RX_DATA_RDY 寄存器描述（0x0500+dn*0x4）共 32 个寄存器（具体参见寄存器描述）.....	243
表 13-6 CTDM_TX_DATA_READY 寄存器描述（0x0580+dn*0x4）共 32 个寄存器（具体参见寄存器描述）.....	244
表 14-1 HDLCBUS 接口信号描述.....	248
表 14-2 HDLC 发送缓冲区 BD（Word0）.....	254
表 14-3 HDLC 发送缓冲区 BD（Word1）.....	255
表 14-4 HDLC 接收缓冲区 BD（Word0）.....	255
表 14-5 HDLC 接收缓冲区 BD（Word1）.....	256
表 14-6 发送缓冲区 BD 的关键控制比特说明.....	265
表 14-7 接收缓冲区 BD 的关键控制比特说明.....	269
表 14-8 冲突分析规则.....	283
表 14-9 模式寄存器（MODE）偏移地址：0x138.....	283
表 14-10 总线操作类型寄存器（BUSOPMD）偏移地址：0x13C.....	292
表 14-11 获取总线优先级寄存器（CTS_PRI）偏移地址：0x1100.....	293
表 14-12 冲突重传计数器（COL_RETRAN_CNT）偏移地址：0x1104.....	293
表 14-13 链路异常重传计数器（ABN_RETRAN_CNT）偏移地址：0x1108.....	293
表 14-14 地址不匹配帧计数器（ADDUN_CNT）偏移地址：0x110C.....	294
表 14-15 单帧发送时间计数器（ONEFR_CNT）偏移地址：0x1110.....	294



表 14-16 同步物理接口信息寄存器（SYN_PHY_INFO）偏移地址：0x1114.....	294
表 14-17 发送源 FIFO 信息寄存器（TX_SRC_FIFO_INFO）偏移地址：0x1118.....	295
表 15-1 GPIO 关键术语描述.....	297
表 16-1 普通 BD CONFIG 域说明.....	311
表 16-2 普通 BD DST_ADDR 域说明.....	314
表 16-3 普通 BD SRC_ADDR 域说明.....	314
表 16-4 普通 BD B_COUNT/A_COUNT 域说明.....	314
表 16-5 普通 BD INIT_DATA/C_COUNT 域说明.....	315
表 16-6 普通 BD BINDX 域说明.....	315
表 16-7 普通 BD CINDX 域说明.....	315
表 16-8 普通 BD LLI 域说明.....	315
表 16-9 普通 BD ALG 域说明.....	316
表 16-10 普通 BD MSG_ADDR_H/SRC_ADDR_H/DST_ADDR_H 域说明.....	317
表 16-11 普通 BD MSG_ADDR 域说明.....	318
表 16-12 普通 BD MSG_DATA0 域说明.....	318
表 16-13 普通 BD MSG_DATA1 域说明.....	318
表 16-14 普通 BD MSG_DATA2 域说明.....	318
表 16-15 普通 BD MSG_DATA3 域说明.....	319
表 16-16 QDMAC_LOCK 寄存器.....	319
表 16-17 QDMAC_LOCK 寄存器.....	320
表 16-18 QDMAC_LOCK 寄存器.....	321
表 16-19 QDMAC_COMM_VMID 寄存器.....	322
表 16-20 QDMAC_INT_VMID 寄存器.....	322
表 16-21 QDMAC_CHG_VMID 寄存器.....	323
表 16-22 QDMAC_GLB_EN 寄存器.....	323
表 16-23 QDMAC_LOCK 寄存器.....	324
表 16-24 QDMAC_GLB_EN 寄存器.....	325
表 16-25 QDMAC_STAT 寄存器.....	326
表 16-26 QDMAC_QUEUE_NUM 寄存器.....	326
表 16-27 QDMAC_QUEUE_CHG_IDX 寄存器.....	327
表 16-28 QDMAC_QUEUE_PRI 寄存器.....	327
表 16-29 QDMAC_QUEUE_STAT 寄存器.....	329
表 16-30 QDMAC_QUEUE_TASK 寄存器.....	329
表 16-31 一维 M2M BD 配置表.....	330
表 16-32 一维 M2F BD 配置表.....	332
表 16-33 一维 F2M BD 配置表.....	333
表 16-34 一维 F2F BD 配置表.....	334
表 16-35 三维 M2M A-sync 模式 BD 配置表.....	335
表 16-36 三维 F2M AB-sync 模式 BD 配置表.....	336
表 16-37 清固定值任务 BD 配置表.....	337
表 16-38 定浮点转换任务 BD 配置表.....	338
表 16-39 BD 链接任务 BD1 配置表.....	340



表 16-40 BD 链接任务 BD2 配置表.....	341
表 16-41 BD 链接任务 BD3 配置表.....	341
表 16-42 定浮点转换任务 BD 配置表.....	342
表 16-43 QDMAC_QUEUE_CHG_IDX 寄存器.....	344
表 16-44 QDMAC_CHG_GRPID 寄存器.....	344
表 16-45 QDMAC_STAT 寄存器.....	345
表 16-46 QDMAC_QOS 寄存器.....	345
表 16-47 QDMAC_OSD_CTRL 寄存器.....	348
表 16-48 QDMAC_GLB_EN 寄存器.....	349
表 16-49 QDMAC_QUEUE_CHG_IDX 寄存器.....	351
表 16-50 QDMAC_CHG_RATE 寄存器.....	351
表 16-51 QDMAC_CHG_MAXTIME 寄存器.....	351
表 16-52 QDMAC_EXEC_THLD 寄存器.....	352
表 16-53 QDMAC_GLB_EN 寄存器.....	352
表 16-54 QDMAC_QUEUE_CHG_IDX 寄存器.....	354
表 16-55 QDMAC_CHG_STREAMID 寄存器.....	354
表 16-56 QDMAC_QUEUE_PAUSE 寄存器.....	356
表 16-57 QDMAC_QUEUE_STAT 寄存器.....	356
表 16-58 QDMAC_QUEUE_PEND 寄存器.....	357
表 16-59 QDMAC_QUEUE_PAUSE 寄存器.....	357
表 16-60 QDMAC_QUEUE_CANCEL 寄存器.....	358
表 16-61 QDMAC_QUEUE_STAT 寄存器.....	359
表 16-62 QDMAC_STAT 寄存器.....	360
表 16-63 QDMAC_STAT 寄存器.....	361
表 16-64 QDMAC_MIF_BD_WORD0 寄存器.....	361
表 16-65 QDMAC_MIF_BD_WORD1 寄存器.....	362
表 16-66 QDMAC_MIF_BD_WORD2 寄存器.....	362
表 16-67 QDMAC_MIF_BD_WORD3 寄存器.....	362
表 16-68 QDMAC_MIF_BD_WORD4 寄存器.....	362
表 16-69 QDMAC_MIF_BD_WORD5 寄存器.....	363
表 16-70 QDMAC_MIF_BD_WORD6 寄存器.....	363
表 16-71 QDMAC_MIF_BD_WORD7 寄存器.....	363
表 16-72 QDMAC_MIF_BD_WORD8 寄存器.....	363
表 16-73 QDMAC_MIF_BD_WORD9 寄存器.....	364
表 16-74 QDMAC_MIF_BD_WORD10 寄存器.....	364
表 16-75 QDMAC_MIF_BD_WORD11 寄存器.....	364
表 16-76 QDMAC_MIF_BD_WORD12 寄存器.....	364
表 16-77 QDMAC_MIF_BD_WORD13 寄存器.....	365
表 16-78 QDMAC_MIF_BD_WORD14 寄存器.....	365
表 16-79 QDMAC_MIF_BD_WORD15 寄存器.....	365
表 16-80 QDMAC_HALT 寄存器.....	366
表 16-81 QDMAC_HALT 寄存器.....	367



表 16-82 QDMAC_HALT 寄存器..... 367

表 16-83 QDMAC_MEM_LP 寄存器.....370

表 16-84 QDMAC_MEM_LP 寄存器.....371

表 16-85 QDMAC_STAT 寄存器..... 373

表 16-86 QDMAC_INT_TC_MASK 寄存器..... 375

表 16-87 QDMAC_INT_TC_STAT 寄存器..... 376

表 16-88 QDMAC_QUEUE_TC 寄存器..... 376

表 16-89 QDMAC_QUEUE_INT_TC_RAW 寄存器..... 377

表 16-90 QDMAC_QUEUE_INT_ALARM_MSK 寄存器.....379

表 16-91 QDMAC_INT_ALARM_MSK 寄存器..... 380

表 16-92 QDMAC_INT_QUEUE_ALARM_STAT 寄存器..... 380

表 16-93 QDMAC_QUEUE_ALARM_BD 寄存器..... 381

表 16-94 QDMAC_QUEUE_ALARM_TYPE 寄存器..... 381

表 16-95 QDMAC_QUEUE_ALARM_CFGERR 寄存器..... 382

表 16-96 QDMAC_QUEUE_INT_ALARM_RAW 寄存器..... 383



1 USB

1.1 概述

USB是通用的串行总线，可以提供与片外逻辑的标准通信接口。

片内提供了2个USB控制器，USB0为单纯的Host功能；USB1位双模（DRD）功能，支持软件静态切换Host模式和Device模式。

1.2 关键术语

USB相关的关键术语表如表1-1所示。

表 1-1 USB 术语表

术语	全称
USB	Universal Serial Bus。
DRD	Dual Role Device。
TRB	Transfer Request Block。

1.3 特性

USB模块具有以下特点。

1.3.1 公共特性

- 支持USB2.0功能，向下兼容USB1.0。
- DRD模式的控制器，支持软件静态切换Host和Device功能。
- 内置DMA控制器。



1.3.2 Device 特性

- 支持的速率模式HS（480-Mbps），FS（12-Mbps）。
- 支持多达8个双向endpoint，包括控制endpoint。
- 支持Control、Bulk、Interrupt and Isochronous endpoint 类型。
- Endpoint类型软件可配。

1.3.3 Host 特性

- 支持的速率模式HS（480-Mbps），FS（12-Mbps）和LS（1.5-Mbps）。
- 支持多达64个外接Device。
- 兼容xHCI 1.1软件接口协议。

1.4 遵循协议

Universal Serial Bus Specification, Revision 2.0, USB-IF, April 27, 2000。
Errata for “USB Revision 2.0 April 27 2000” as of May 28, 2002, USB-IF。
USB 2.0 Transceiver Macrocell Interface（UTMI）Specification, Revision 1.05, Intel Corp., March 29, 2001。
AMBA Specification, Revision 2.0, ARM Ltd, ARM IHI 0001, 1999。
AMBA AXI（3）Protocol Specification, v2.0, ARM Ltd, ARM IHI 0022C, 2003-2010。
eXtensible Host Controller Interface for Universal Serial Bus（xHCI），Revision 1.1 with errata to 12/20/13, Intel Corp., December 20, 2013。

1.5 功能描述

USB的详细功能参见USB标准协议，此处不赘述。

1.5.1 编程接口

编程接口是软件的配置接口。此处分Host和Device两种模式进行描述。

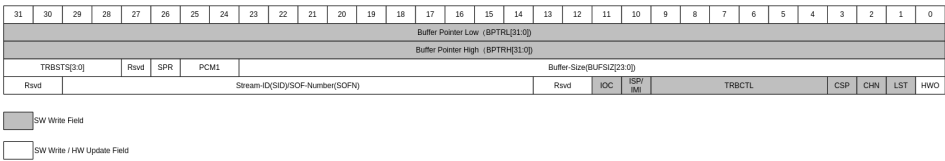
1.5.1.1 Host 模式编程接口

详细参见xHCI协议规范，此处不赘述。

1.5.1.2 Device 模式编程接口

- 数据结构

图 1-1 TRB 结构





上图1-1为TRB（Transfer Request Block）各个域段的细节指示。

表 1-2 TRB 域段描述

Field	Description	Hardware Access
DW 03-00		
31:0	Buffer Pointer Low（BPTRL）。 数据Buffer指针的低32bit。	R_W
DW 07-04		
31:0	Buffer Pointer High（BPTRH）。 数据Buffer指针的高32bit。	R_W
DW 0B-08		
23:0	Buffer size（BUFSIZ）。 当前TRB所指示Buffer剩余空间指示。	R_W
25:24	Packet Count M1（PCM1）。 在High-Speed，高带宽的isochronous IN endpoint通信过程中，指示的是Buffer中总的 packet数量减1的值。	R_W
26	SPR/Rsvd。 GUCTL1[DEV_TRB_OUT_SPR_IND]=1时，这个空间表明当前这个TRB的最后一个传输，接收到一个短包。在OUT传输期间，如果最后一个传输是一个短包，则该TRB的这个位将会置1。 GUCTL1[DEV_TRB_OUT_SPR_IND]=1时，该域段保留。	R_W
27	保留。	R_W
31:28	TRB status（TRBSTS）。 在释放该TRB之前，硬件更新该域段。 0x0：OK； 0x1：Missedloc:Isochronous 间隔丢失或未完成； 0x2：SetupPending-在control传输的data/status阶段，又接收到一个SETUP； 0x4：TransferInProgress-传输过程中，接收到一个end transfer命令； 0xF：ZLP_PENDING-只用于NORMAL-ZLP模式。表明IN传输的zero-length-packet仍然处于挂起状态。	R_W
DW 0F-0C		



Field	Description	Hardware Access
0	Hardware Owner of Descriptor (HWO) 。 指示硬件获取TRD的控制权。 当软件创建一个TRB的时候就会将这个bit设置为1，只有在硬件将这个bit修改为0以后，软件才能够回收这个TRB。	R_W
1	Last TRB (LST) 。 表明这个TRB是整个TRB列表中最后一个TRB。在完成传输操作以后，将会产生一个XferComplete Event。	R
2	Chain Buffer (CHN) 。 适用于IN和OUT endpoint。 软件通过将这个bit设置为1，用来指示这个TRB与下一个TRB之间的关系。	R
3	Continue on Short Packet (CSP) 。 适用于OUT endpoint，且尽在接收到短包的时候有用。 如果为1，在接收到短包的时候将会继续执行下一个传输。isochronous endpoint需要设置该bit。 如果为0，在接收到短包的时候会产生一个XferComplete Event并且移除这个stream。	R
9:4	TRB Control (TRBCTL) 。 表明的是TRB类型： 1： Normal (Control-Data-2+/Bulk/Interrupt) 。	R
10	Interrupt on Short Packet / Interrupt on Missed ISOC (ISP / IMI) 。 对于OUT endpoint： 接收到一个短包的时候，并且CSP = 1且LST = 0的时候，如果该bit为1，将会产生一个XferInProgress Event。 对于Isochronous endpoint： 当产生Missed Isoc状态的时候，将产生一个XferInProgress Event。	R
11	Interrupt on Complete (IOC) 。 当该bit为1的时候，这个TRB对应的传输Buffer的所有传输完成后，将会产生一个XferInProgress Event，并且Event status的IOC位为1。	R
13:12	保留。	R_W



Field	Description	Hardware Access
29:14	Stream ID / SOF Number。 如果是stream-based bulk endpoint：这个域段代表的是Stream ID。 如果是isochronous endpoint：这个域段代表的是最后一个packet的微帧编号。	R_W
31:30	保留。	R_W

- 命令结构
命令结构是软件通过配置DGCMD和DGCMDPAR寄存器，来管理链路包的收发。
Device的命令结构分为：
 - Device通用命令结构。
 - Device物理Endpoint命令结构。
 - Device通用命令结构
通用的Device命令结构包括：
Set Periodic Parameters
Set Scratchpad Buffer Array Address Lo
Set Scratchpad Buffer Array Address Hi
Transmit Device Notification
Selected FIFO Flush
All FIFO Flush
Set Endpoint NRDY
Run SoC Bus LoopBack Test
Restart After Disconnect

表 1-3 Device 通用命令类型

Command	Description
02h	Set Periodic Parameters。 Parameter[9:0]（SystemExitLatency）：软件应该将这个空间，配置成host通过Set SEL命令请求Device所配置的值，单位是ms。
04h	Set Scratchpad Buffer Array Address Lo。 这个命令配置的是scratchpad Buffer地址bit[31:0]，这个scratchpad用于save/restore。 在控制器处于stop模式下的时候（RunStop=0），如果发送了这个命令或05h命令，那么CmdIIOC要配置为0。
05h	Set Scratchpad Buffer Array Address Hi。 这个命令配置的是scratchpad Buffer地址bit[63:32]，这个scratchpad用于save/restore。 在控制器处于stop模式下的时候（RunStop=0），如果发送了这个命令或05h命令，那么CmdIIOC要配置为0。



Command	Description
07h	Transmit Device Notification。 这个命令可以发送任何Device类型的通知信息，通知类型和通知参数可以通过DGCMDPAR寄存器来指定。 DGCMDPAR[31:4]=通知参数； DGCMDPAR[3:0]=通知类型。
09h	Selected FIFO Flush。 Parameter[5]=0, RX FIFO；1, TX FIFO； Parameter[4:0]=FIFO number。
0Ah	All FIFO Flush。 无参数。
0Ch	Set Endpoint NRDY。 这个命令会令内核认为指定的endpoint处于NRDY状态。如果这个endpoint有可用的Buffer，那么控制器会立即发送一个ERDY。 Parameter[4:0] = Physical Endpoint Number。
10h	Run SoC Bus LoopBack Test。 使能的时候，可以让内核处于SoC Bus Loopback test模式，在没有任何设备与PHY连接的时候，发送端的数据会直接进入接收到端。 <ol style="list-style-type: none">1. 配置Physical EP0，为一个non-stream capable Bulk OUT endpoint，并且配置期望的MaxPacketSize。2. 配置Physical EP1，为一个non-stream capable Bulk IN endpoint，并且将TxFIFO 0分配给这个endpoint，并且配置期望的MaxPacketSize。3. 发送这个命令，Parameter[0]配置为1，使能Loopback模式。4. 发送Start Transfer给EP0。5. 发送Start Transfer给EP1。6. 内核从IN Buffer读取数据，并写会给OUT Buffer。 Parameter[0] = 0, Disable Loopback模式。 Parameter[0] = 1, Enable Loopback 模式。
11h	Restart After Disconnect。 无效命令。

■ Device物理Endpoint命令结构

软件通过DEPCMDn，DEPCMDPAR0n，DEPCMDPAR1n，DEPCMDPAR2n来实现物理endpoint命令的发送。



Device物理endpoint命令如下：

Set Endpoint Configuration:DEPCFG
Set Endpoint Transfer Resource Configuration:DEPXFERCFG
Get Endpoint State:DEPGETSTATE
Set Stall and Clear Stall:DEPSSTALL,DEPCSTALL
Start Transfer:DEPSTRTXFER
Update Transfer:DEPUPTXFER
End Transfer:DEPENDXFER
Start New Configuration:DEPSTARTCFG

Set Endpoint Configuration：DEPCFG

这个命令用来设置物理endpoint的配置信息。

表 1-4 DEPCFG

Field	Description
Parameter 2	
31:0	当Parameter 0的Config Action = 1的时候，这个空间设置的是 endpoint的状态。否则，该空间保留。
Parameter 1	
31	FIFO-based。 如果这个isochronous endpoint是一个FIFO-based 数据流结构，需要将该bit配置为1。
30	保留。
29:25	USB Endpoint Number。 bit[29:26]：Endpoint Number。 bit[25]：Endpoint direction。 0：OUT； 1：IN； 物理endpoint0（EP0）一定要是control endpoint 0 OUT。 物理endpoint1（EP1）一定要是control endpoint 1 IN。
24	Stream Capable（StrmCap）； 只是这个endpoint是stream-capable（MaxStreams !=0）。
23:16	bInterval_m1。 设置为bInterval - 1。 有效值是0~13。 bInterval在endpoint描述符中反馈。工作在Full-Speed模式的时候，该bit配置为0。
15	如果endpoint使用External Buffer Control（EBC）模式。可用来限制DMA的outstanding数量。 保留。



Field	Description
14	在bit[15] = 1的时候，才需要配置该bit，否则，无需配置该bit。 0：控制器回写TRB描述符（更新TRB的HWO）。 1：控制器不会回写TRB描述符（不更新TRB的HWO）。
13:8	Device Endpoint Specific Event Enable（DEPEVTEN）。 如果某个bit配置为0，那么相应的Event就会被关闭。 bit[13]：Stream Event Enable； bit[12]：Reserved； bit[11]：Reserved； bit[10]：XferNotReady Enable； bit[9]：XferInProgress Enable； bit[8]：XferComplete Enable。
7:5	保留。
4:0	Interrupt number（IntrNum）； 必须要配置为0。
Parameter 0	
31:30	Config Action； 00：初始化endpoint状态。 01：重载endpoint状态。 10：修改endpoint状态。
29:26	保留。
25:22	Burst Size（BrstSiz）； 0x0：Burst length = 1； 0x1：Burst length = 2，以此类推； 0xF：Burst length = 16。
21:17	FIFO Number（FIFONum）。 指示那个TxFIFO分配给这个endpoint。 对于Control endpoint，OUT endpoint的FIFONum要与IN endpoint的相同。对于其他OUT endpoint，这个空间需要配置为0。 虽然在DRD模式中，TxFIFO的数量大于等于16个，但是Device模式必须要使用低16个TxFIFO。
16:14	保留。
13:3	Maximum Packet Size（MPS）； 适用于IN和OUT endpoint。



Field	Description
2:1	Endpoint Type (EPTYPE) ; USB endpoint的传输类型; 00: Control; 01: Isochronous; 10: Bulk 11: Interrupt。
0	保留。

Set Endpoint Transfer Resource Configuration: DEPXFERCFG

每个endpoint只能有一个传输资源。Start Transfer会使用这个传输资源。End Transfer或XferComplete Event会释放这个传输资源。若软件尝试为一个endpoint分配多个传输资源，则会导致CmdStatus或EventStatus报告错误。

表 1-5 DEPXFERCFG

Field	Description
Parameter 2	
31:0	保留。
Parameter 1	
31:0	保留。
Parameter 0	
31:16	保留。
15:0	Number of Transfer Resources (NumXferRes) ; 为这个endpoint分配资源数量。必须要为1。

Get Endpoint State: DEPGETSTATE

在执行“Get Endpoint State”命令完成后，DEPCMDPAR2中将包含endpoint的状态信息，包括当前数据序列号、流控状态以及控制传输状态（仅适用于控制端点）。由于本控制器不支持休眠状态，因此不会使用此命令。

Set Stall and Clear Stall: DEPSSTALL, DEPCSTALL

仅适用于non-isochronous IN和OUT endpoint。

软件发送一个Set Stall命令来暂停USB host发送给这个endpoint的token。

如果endpint处于NRDY状态，那么STALL状态抢占NRDY优先级。



当软件发送Set Stall命令时，如果当前有业务需要处理，则需根据endpoint方向进行如下处理：

- 对于OUT endpoint，当前包将会完成。控制器会在下一个OUT DP或token的时候使用STALL来进行响应。
- 对于SuperSpeed模式下的IN endpoint，当前的突发业务会被完成，内核会在下一个ACK TP的时候使用STALL进行响应。对于其他速率，控制器会完成当前的包，并且在下一个IN token使用STALL进行响应。
- 对于non-control endpoint，软件的职责就是通过Set Stall/Clear Stall命令来设置和清除STALL。当软件清除了STALL的时候，endpoint的数据序列号就会被重置为0。
- 对于control endpoint，软件只需在OUT方向的端点上发送Set Stall命令。当控制器接收到针对该端点的SETUP token时，控制器会自动清除STALL状态。软件不能在控制端点上发送Clear Stall命令。

如果endpoint处于流控状态（NRDY），并且仍处在STALL状态，控制将使用STALL来响应所有包。唯一例外的情况是，控制器总是会使用ACK握手包来响应SETUP数据包，无论当前是不是处于STALL状态。

Start Transfer: DEPSTRTXFER

这个命令适用于IN和OUT endpoint。

在Device模式下，传输描述符（TD）被实现为一个TRB链表结构，负责多项传输操作。软件通过发出Start Transfer命令来提交这个TD的指针；而当XferComplete Event触发或者End Transfer命令执行时，该指针将自动从列表中移除。

软件发出此命令以告知控制器已经构建好描述符于内存之中。对于IN endpoint而言，此命令将触发控制器根据描述符进行操作，并将数据从对应的memory空间传送到内部TxFIFO；对于OUT endpoint，同样回触发控制器执行该描述符，同时将接收到的数据从RxFIFO搬移至相应的memory区间。

在Start Transfer命令执行时，硬件将会为此传输分配一个资源索引编号（XferRscIdx），并将这个索引值通知DEPCMDn寄存器以及Command Complete Event中。后续的Update和End Transfer命令必须使用同一index以保持数据一致性和流程连续性。

如果控制器中的缓存中已存在相同的TD，那么对这个TD再次执行Start Transfer命令仍被视为合法操作，不会引发冲突或错误。

non-stream capable endpoint规则：

- Stream ID必须要为0。
- HighPri空间保留。

Stream-capable endpoint规则：

- Stream ID空间一定要是一个非0值，并且与Start Transfer endpoint命令中的Stream ID相匹配。在这个传输的所有TRB中，Stream ID的值相同。
- HighPri空间保留。



表 1-6 DEPSTRXFER

Field	Description
Parameter 2	
31:0	保留。
Parameter 1	
31:0	Transfer Descriptor Address (TDAAddr Low) 。
Parameter 0	
31:0	Transfer Descriptor Address (TDAAddr High) 。

Update Transfer:DEPUPDXFER

如果软件使用的是一个环形的TRB Buffer，并且更新一个TRB的HWO=1，但该TRB的HWO=0，则软件必须执行Update Transfer命令，并在DEPCMD中提供该TRB的资源索引。硬件将利用此信息重新缓存该TRB。

软件可能会通过将CmdAct=0且CmdIOC=0的方式发送“ No Response Update Transfer”命令。在这种情况下，硬件不会生成Command Complete Event，也不会将CmdAct设置为0（因为该位已经为0）。软件可能会在此命令之后向相同的endpoint发送其他命令。当软件根据XferNotReady Event构建TRB时，可能不会使用Update Transfer命令。

End Transfer:DEPENDXFER

该命令适用于IN和OUT endpoint。软件通过发送此命令来请求DMA停止预取操作，并且DEPCMD中的ForceRM将会被设置为1。

发送一个End Transfer命令之后，软件一定要将CmdIOC设置为1，以便在传输完成后触发Endpoint Command Complete事件。

对于IN endpoint，此命令可用于停止描述符的执行操作。内核将停止预取针对该endpoint的新数据，并停止传输操作。内核可能使用DPPABORT序列来中断当前正在进行的数据包传输，而无需等待任何ACK，并且不更新TRB状态。

对于OUT endpoint，此命令可用于停止描述符的执行操作。如果有数据在FIFO中，这些数据将被移动到相应的内存空间。内核不会更新TRB状态。

在如下条件下使用这个命令：

- 当处理USBReset或SetConfiguration的时，可以使用此命令来关闭endpoint。
- 在接收到一个ClearFeature（STALL）控制传输的时，软件可以在Clear Stall命令后发送End Transfer命令，然后发送Start Transfer命令。
- 在XferInProgress Event后，如果某个TRB的HWO字段被设置为0，则可以使用此命令。
- 对于isochronous endpoint，在host在几个interval内都没有传输数据的情况下，软件可能会强制传输结束，并等待host重新发起传输。

在End Transfer的时候，硬件不会发送XferComplete Event，而是发送CommandComplete Event。



表 1-7 DEPENDXFER

Field	Description
Parameter 2	
31:0	保留。
Parameter 1	
31:0	保留。
Parameter 0	
31:0	保留。

Start New Configuration:DEPSTARTCFG

软件在如下情况下发送该命令：

- power-on-reset后，XferRsclDx=0，开始配置endpoint 0 和 1之前。
CmdIOc必须要配置为0，并且软件要轮询的方式查询CmdAct，以便确定命令完成的时刻。
- XferRsclDx = 2，当接收到SetConfiguration的时候，且在开始配置Endpoint > 1之前。
CmdIOc可能会配置为0，也可能会配置为1。
这个命令只能发送给Endpoint0。
在接收到这个命令的时候，硬件会将传输资源复位成XferRsclDx参数（一定要是0或2）。

Device Event Buffer结构

当有Event产生的时候，硬件会去检查相应中断的使能位。Event Buffer中存放的是如下类型的信息，具体要看Event的低bit的状态：

- Device Endpoint-Specific Event (DEPEVT) (Event[0] = 0) ;
DEPCFG endpoint命令可以使能这个中断。
- Device-Specific Event (DEVT) (Event[0] = 1, Event[7:1] = 0x00) ;
Generic Command Complete Event可以通过DGCMD.CmdIOc来使能。
Event Buffer Overflow Event不能被关闭。
其他的Device-Specific Event是通过DEVTEN寄存器来使能的。
除了Event Buffer Overflow Event，这些Event都会被写入到DCFG.IntrNum空间指定的Event Buffer中。



表 1-8 Device Endpoint-n Events

Field	Description
31:16	Event Parameters (EventParam) 。 <ul style="list-style-type: none">• Bulk Enpoint上的XferNotReady,XferComplete,和Stream Event:• bit[31:16]: Stream ID。 XferInProgress: <ul style="list-style-type: none">• bit[31:16]: Isochronous Microframe Number。 Isochronous Endpoint上的XferNotReady Event: <ul style="list-style-type: none">• bit[31:16]: Isochronous Microframe Number。 EPCmdCmplt Event: 对于所有的EPCmdCmplt Event。 <ul style="list-style-type: none">• bit[27:24]: Command Type。 对于Start Transfer命令的EPCmdCmplt Event。 <ul style="list-style-type: none">• bit[22:16]: Transfer Resource Index。



Field	Description
15:12	<p>Event Status (EventStatus)</p> <p>在XferNotReady Event中:</p> <p>bit[15]: 表明XferNotReady Event产生的原因。</p> <ul style="list-style-type: none">• 0: XferNotActive;• 1: XferActive; <p>bit[14]: Not Used;</p> <p>bit[13:12]: 对于控制endpoint, 指示请求阶段:</p> <ul style="list-style-type: none">• 01: Control Data Request;• 10: Control Status Request;• 其他: 保留。 <p>在XferComplete或XferInProgress Event中:</p> <p>bit[15]: 完成TRB的LST位;</p> <p>bit[15]: MissedIsoc;</p> <p>bit[14]: TRB的IOC位;</p> <p>bit[13]: 只是TRB的完成, 是因为接收到一个短包, 或者是 isochronous interval中最后一个包;</p> <p>bit[12]: 保留。</p> <p>在Stream Event中:</p> <p>bit[15:12]:</p> <ul style="list-style-type: none">• 0x1: StreamFound。• 0x2: StreamNotFound。• 其他: 保留。 <p>在响应Start Transfer命令的时候:</p> <p>bit[15:12]:</p> <ul style="list-style-type: none">• 0x1: 表明没有有效的传输资源适用于这个endpoint。• 0x2: 表明Start Transfer命令的总线时间已经溢出。• 其他: 保留。 <p>在响应Set Transfer Resource命令的时候:</p> <p>bit[15:12]:</p> <ul style="list-style-type: none">• 0x1: 表明有错误产生, 因为软件请求了多个传输资源。• 其他: 保留。 <p>在响应End Transfer命令的时候:</p> <p>bit[15:12]:</p> <ul style="list-style-type: none">• 0x1: 表明分配了无效的传输资源。• 其他: 保留。
11:10	保留。



Field	Description
9:6	0x0：保留。 0x1：XferComplete Event。 0x2：XferInProgress Event。 0x3：XferNotReady Event。 0x4：保留。 0x5：保留。 0x6：Stream Event。 0x7：Endpoint Command Complete。
5:1	Physical Endpoint Number（0~31）。
0	0：表明这个是一个endpoint-specific Event。

表 1-9 Device-Specific Event

Field	Description
31:25	保留。
24:16	Event Information bits（EvtInfo）。 对于一个USB/Link状态改变Event，这个域段表明的是链路状态： EvtInfo[8:5]：HIRD值，来自LPM token。 EvtInfo[4]：SuperSpeed Event。0：non-SS；1：SS；。 EvtInfo[3:0]：Link State。
15:13	保留。



Field	Description
12:8	0x0: Disconnect Detected Event; 0x1: USB Reset; 0x2: Connection Done; 0x3: USB/Link State Change; 0x4: Resume/Remote Wakeup Detected Event; 0x5: Hibernation Request Event; 0x6: USB Suspend Entry Event; 0x7: Start of Frame; 0x8: L1 Suspend Event; 0x9: Erratic Error Event; 0xA: Generic Command Complete Event; 0xB: Event Buffer Overflow Event; 0xC: Vendor Device Test LMP Received Event; 0xD: Stopped On Disconnect; 0xE: L1 Resume/RemoteWake Event; 0xF: Reserved; 0x10: ECC Error。
7:1	0x00: Device Specifc Event。
0	1: Non-Endpoint-Specific Event。

1.6 软件指南

1.6.1 AXI 突发长度的控制

USB与内存之间的数据交互，要求按照对齐方式进行访问，具体配置如下：

PERI_USB3_GSBUSCFG0.incrbrstena=0。

根据系统需求，内存访问期间不能跨越64Byte地址空间，具体配置如下：

PERI_USB3_GSBUSCFG0.incr4brstena=1；

PERI_USB3_GSBUSCFG0.incr8brstena=0；

PERI_USB3_GSBUSCFG0.incr16brstena=0；

PERI_USB3_GSBUSCFG0.incr32brstena=0；

PERI_USB3_GSBUSCFG0.incr64brstena=0；

PERI_USB3_GSBUSCFG0.incr128brstena=0；

PERI_USB3_GSBUSCFG0.incr256brstena=0。



1.6.2 Cache 属性配置约束

详细的cache属性请参阅AXI协议。USB的cache属性，需要按照如下配置方式进行配置：

PERI_USB3_GSBUSCFG0.datrdreqinfo=0xF（数据读操作cache属性）；

PERI_USB3_GSBUSCFG0.desrdreqinfo=0xF（描述符读操作cache属性）；

PERI_USB3_GSBUSCFG0.datwrreqinfo=0xF（数据写操作cache属性）；

PERI_USB3_GSBUSCFG0.deswrreqinfo=0xE（描述符写操作cache属性）。

1.6.3 初始化配置

系统上电以后，使用USB3.0控制器的时候，对USB控制器执行初始化。

步骤1 按要求配置PHY的相应参数，PHY的参数包括信号质量的参数控制。PLL的参数控制等（PHY的参数配置过程中，PHY需要处于复位状态）。

步骤2 解除PHY复位。

步骤3 解除USB控制器的时钟和复位。

步骤4 配置USB寄存器的GUCTL（0xC12C）的bit[10:9] = 0b11（适配某些设备的响应速度，此处是将其调整为最大的Timeout时间：6.5msec）（仅Host模式下需要配置该步骤）。

步骤5 Host模式下执行xHCI的正常初始化流程（请参见《eXtensible Host Controller Interface for Universal Serial Bus（xHCI）》）。

步骤6 Device模式下执行Device初始化流程。

----结束

1.6.3.1 Host 初始化流程

参见xHCI协议配置流程。此处不赘述。

1.6.3.2 初始化流程

参见开源驱动配置流程。此处不赘述。

1.7 DFX 应用指导



1.7.1 故障在线检测

表 1-10 PORTSC 寄存器

描述	寄存器名	域段名	偏移地址	比特位	说明
Link 状态	PORTSC	pls	0x0400+0x20+port_num*0x10	8:5	Host功能状态说明。 Port Link State; 详细参见xHC协议。
速率	PORTSC	port_speed	0x0400+0x20+port_num*0x10	13:10	Host功能状态说明。 Port Speed; 0: 无效值; 0x1~0x15: Protocol Speed ID, 参见xHCI协议。

1.7.2 眼图测试码流

USB2.0的协议中规定控制器要支持Test_J_State、Test_K_State、Test_Packet、Test_Force_Enable、和Test_SE0_NAK测试。同一时刻，只能对一个端口执行测试。如果想要将多个端口都配置成测试模式，控制器工作会出现异常。每个端口进入测试模式的具体步骤如下：

- 步骤1 Dsiable 所有的Device Slot（默认上电启动情况下所有Device Slot都是disable的）。
- 步骤2 所有的端口都应该处于Disabled状态（将对应PORTSC寄存器的PP 配置成0）。
- 步骤3 将USBCMD的Run/Stop配置成0，等待USBSTS寄存器的HCHalted（HCH）变为1。

须知

xHC控制器不允许端口在R/S位为1的时候进行测试。

- 步骤4 将PORTPMSC（**注意只配置USB2.0的对应寄存器**）寄存器的Port Test Control空间，配置成自己想要的测试模式。

对于USB2端口，如果备选测试为Test_Force_Enable，那么选定了测试模式以后，一定要将USBCMD的Run/Stop（R/S）位配置成1，目的是为了能够让控制器在这个测试模式下发出SOF。
- 步骤5 测试完成以后，如果xHC还处于运行状态，软件应该清除USBCMD的R/S位，并且等到控制器的USBSTS的HCHalted（HCH）位变为1。



步骤6 通过将HCRST配置为1，结束测试模式。

----结束

PORTPMSC的详细描述如（详细参见xHC协议中PORTPMSC寄存器的描述，5.4.9.2节）。

图 1-2 PORTPMSC 的详细描述

Bit	Description																		
31:28	<p>Port Test Control – RW. Default = '0'. When this field is '0', the port is NOT operating in a test mode. A non-zero value indicates that it is operating in test mode and the specific test mode is indicated by the specific value.</p> <p>A non-zero Port Test Control value is only valid to a port that is in the <i>Powered-Off</i> state (PLS = <i>Disabled</i>). If the port is not in this state, the xHC shall respond with the <i>Port Test Control</i> field set to <i>Port Test Control Error</i>. Refer to section 4.19.6 for the operational model for using these test modes.</p> <p>The encoding of the Test Mode bits for a USB2 protocol port are:</p> <table><tr><th>Value</th><th>Test Mode</th></tr><tr><td>0</td><td>Test mode not enabled</td></tr><tr><td>1</td><td>Test J_STATE</td></tr><tr><td>2</td><td>Test K_STATE</td></tr><tr><td>3</td><td>Test SE0_NAK</td></tr><tr><td>4</td><td>Test Packet</td></tr><tr><td>5</td><td>Test FORCE_ENABLE</td></tr><tr><td>6-14</td><td>Reserved.</td></tr><tr><td>15</td><td>Port Test Control Error.</td></tr></table> <p>Refer to the sections 7.1.20 and 11.24.2.13 of the USB2 spec for more information on Test Modes.</p>	Value	Test Mode	0	Test mode not enabled	1	Test J_STATE	2	Test K_STATE	3	Test SE0_NAK	4	Test Packet	5	Test FORCE_ENABLE	6-14	Reserved.	15	Port Test Control Error.
Value	Test Mode																		
0	Test mode not enabled																		
1	Test J_STATE																		
2	Test K_STATE																		
3	Test SE0_NAK																		
4	Test Packet																		
5	Test FORCE_ENABLE																		
6-14	Reserved.																		
15	Port Test Control Error.																		

USBCMD的详细描述参见xHC协议中[鉴权等级](#)。

USBSTS的详细描述参见xHC协议中[鉴权等级](#)。



2 UART

2.1 概述

UART（UniversalAsynchronousReceiver/Transmitter）是一个异步串行的通信接口，主要执行串并转换功能，将芯片发送到外围设备的数据进行并串转换，同时将来自外围设备的串行数据进行串并转换。该接口模块遵循RS232串行通信协议。

UART内部包括128Byte的接收FIFO和发送FIFO，CPU可通过访问UART状态寄存器查询数据传输的错误信息和器件的工作状态，可通过访问UART中断控制寄存器调整系统中断来满足用户的要求。

2.2 特性

UART具有以下特点：

- 支持8bit宽、深度为128的发送FIFO和10bit宽、深度为128的接收FIFO，支持FIFO禁止或使能。
- 支持5/6/7/8bit可编程数据位。
- 支持1/1.5/2bit可编程停止位。
- 支持奇、偶校验方式，支持无校验位，支持校验位为软件设定值。
- 支持一个组合中断连接到中断控制器，中断类型包括：接收FIFO中断、发送FIFO中断、接收超时中断、错误中断。
- 支持传送速率可编程，典型传送速率范围为200Mbps/（16*N）（0<N<65536，N为整数）。
- 支持关断UART时钟以降低功耗。
- UART0支持输入重定向传输功能。
- 只支持UART三线模式（包括地线）。
- UART支持主从模式，9bitUART通信。

由于工作频率固定，在支持某些标准的波特率时会有误差，表2-1给出以168.75MHz工作时的误差率，当速率越高时，误差将会越小。



表 2-1 波特率误差

波特率（ bps ）	分频比	误差率
9600	1099	-0.03%
57600	183	-0.49%
115200	92	0.45%

2.3 关键术语

表 2-2 UART 关键术语描述

术语	描述
UART	Universal Asynchronous Receiver/Transmitter，异步串行的通信接口，主要执行串并转换功能，将芯片发送到外围设备的数据进行并串转换，同时将来自外围设备的串行数据进行串并转换。该接口模块遵循RS232串行通信协议。

2.4 遵循协议

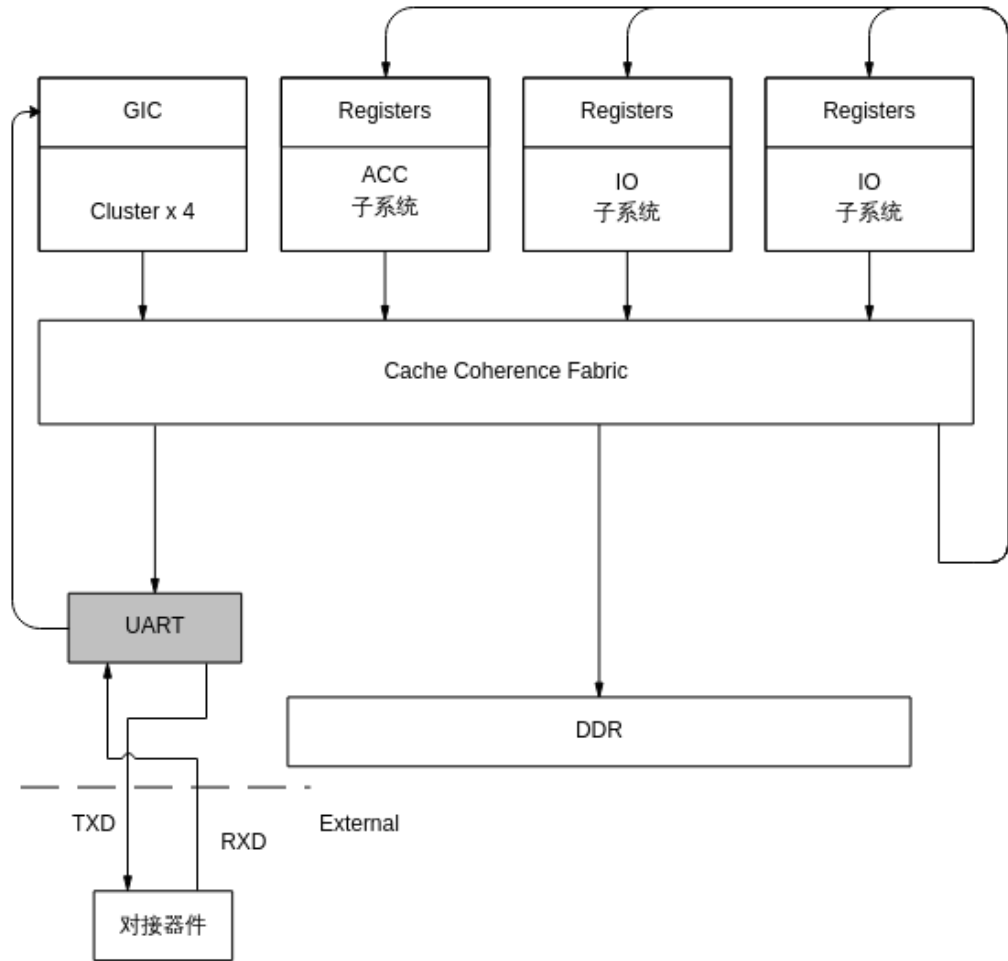
该接口模块遵循RS232串行通信协议。

2.5 功能描述



2.5.1 应用场景

图 2-1 UART 应用框图



UART是一种异步双向串行总线，提供了一种简单有效的数据传输方式，只需要两根数据线互相对接即可。

CPU可通过总线配置UART的寄存器配置数据传输到对端，或者读取寄存器获取接收到的对端数据；对于较大量的数据，CPU还可以通过总线配置DMA自动完成一段数据的发送和接收。

UART在传输完成或传输发生错误时，可以通过中断通知CPU进行处理。

2.5.2 接口信号

UART接口信号如表2-3所示。

表 2-3 UART 接口信号描述

名称	方向	描述	对应管脚
URXD0	I	UART0接收信号。	URXD0



名称	方向	描述	对应管脚
UTXD0	O	UART0发送信号。	UTXD0
URXD01	I	UART0重定向接送信号。	URXD01
UART_CTRL	I	UART0重定向控制信号。	UART_CTRL
URXD1	I	UART1接收信号。	URXD1
UTXD1	O	UART1发送信号。	UTXD1

2.5.3 相关模块

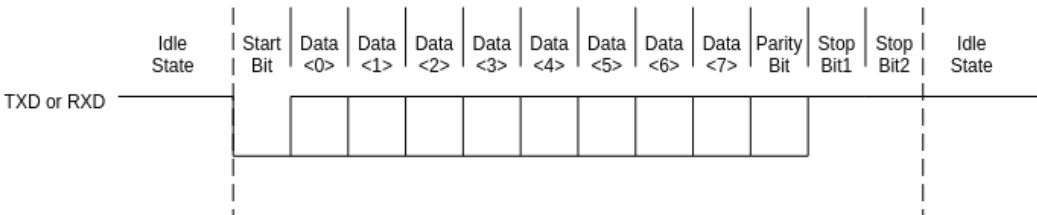
在系统中，UART模块会和下面多个模块进行交互，分别描述如下：

- CPU：CPU通过总线访问UART的寄存器接口，操作属性必须为Device或Strongly-Ordered。CPU首先需要完成UART的传输模式和速率配置，对于少量或即时性要求高的数据，CPU可以直接通过FIFO接口读写完成数据传输。
- DMAC：如果UART传输的数据量较大，且即时性并不强，可以通过DMAC完成数据的发送和接受，这需要CPU配置DMAC的一个独立通道，将为该通道与UART及传输的数据区域绑定，DMAC在总线空闲时，将自动完成数据的发送和接收。
- GIC：UART在传输时如果发生错误，例如奇偶校验错误、数据帧断裂等，以及在非DMAC搬移模式时，如果接收FIFO高于水线，或发送FIFO低于水线，均会产生中断信号给GIC模块，通过GIC将中断发送给对UART具有控制权的CPU。

2.5.4 数据结构

UART使用RS232串行传输协议的帧结构完成数据传输。一次帧传输主要包括起始信号、数据、校验位和结束信号，如图2-2所示。数据帧从UART的TXD端输出，从UART的RXD端输入。

图 2-2 UART 帧格式



说明

- 起始信号（Start bit）：一个数据帧开始的标志，UART协议规定TXD信号出现一个低电平就表示一个数据帧的开始。在UART不传输数据时，应该保持高电平。
- 数据信号（Data bit）：数据位宽可以根据不同的应用要求进行调整，可以配置成5bit、6bit、7bit或8bit数据位宽。
- 校验位（Parity bit）：校验位是1bit纠错信号，UART的校验位有奇校验、偶校验和固定校验位，同时支持校验位的使能和禁止，详细描述请见《Hi309a 寄存器手册》。
- 结束信号（Stop bit）：结束信号即数据帧的停止位，支持1bit和2bit停止位两种配置。数据帧的结束信号是把TXD拉成高电平。



2.5.5 同步功能

说明

UARTx表示UART0，UART1。

异步串行通信的关键是要保证通信双方收发同步。由于没有随路时钟线，所以双方必须约定一个固定的频率，用于同步收发数据，这样才能保证数据传送的正确性。

UART的同步就是寻找起始位来确定一帧数据的开始。UART的同步过程如下：

- 步骤1** UART产生一个16倍于波特率的总线时钟CLK_X（设时钟周期为T）。
- 步骤2** 当CLK_X检测到串行输入的下降沿后，启动周期计数。
- 步骤3** 在第8个CLK_X周期采样数据，若为低电平，表明检测到了起始位。
- 步骤4** 每隔16个CLK_X周期便对数据进行采样，直到一帧数据结束。

----结束

2.5.6 主从串口通信（UART 9 bits 通信）

说明

UARTx表示UART0、UART1。

网络应用中，可以通过PEN、SP、EPS实现UART 9 bit通信功能。UARTx_LCR寄存器PEN为奇偶校验使能，SP为Stick Parity使能，EPS为偶校验使能，具体功能如表2-4和表2-5所示。

表 2-4 UART 发送方向奇偶校验功能

PEN	SP	EPS	功能描述
0	0	0	不发送Parity bit。
0	0	1	不发送Parity bit。
0	1	0	不发送Parity bit。
0	1	1	不发送Parity bit。
1	0	0	奇偶校验：奇校验使能，发送带有奇校验位；接收检测奇校验。
1	0	1	奇偶校验：偶校验使能，发送带有偶校验位；接收检测偶校验。
1	1	0	发送帧带有奇偶校验位，发送的Parity bit = 1。
1	1	1	发送帧带有奇偶校验位，发送的Parity bit = 0。



表 2-5 UART 接收方向奇偶校验功能

PEN	SP	EPS	功能描述
0	0	0	没有Parity bit，不检测奇偶校验。
0	0	1	没有Parity bit，不检测奇偶校验。
0	1	0	没有Parity bit，不检测奇偶校验。
0	1	1	没有Parity bit，不检测奇偶校验。
1	0	0	奇偶校验：奇校验使能，接收检测奇校验。
1	0	1	奇偶校验：偶校验使能，接收检测偶校验。
1	1	0	预期接收到的Parity bit =1；如果为0则上报Parity error。
1	1	1	接收到的Parity bit = 0；如果为1则上报Parity error。

以采用UART主从串口方式进行多机通信为例，波特率为187500bps，1个起始位、8个数据位、1个控制位、1个停止位，一共11位，可以从节点通过第9位区分是地址还是数据。假设接收方向采用奇校验，那么接收方向识别地址和数据的步骤如下：

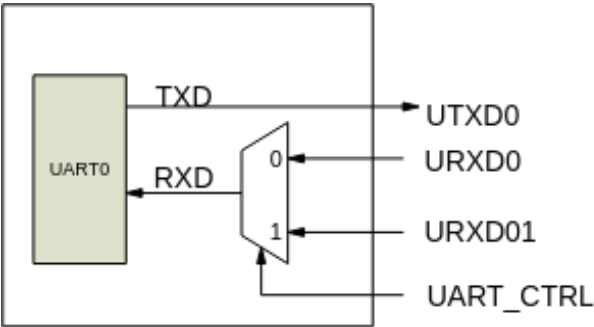
- 步骤1 主节点发送地址时，将发送数据设置为奇校验；
- 步骤2 接收端收到数据后，将上报校验正确，接收端则识别为地址；
- 步骤3 主节点发送数据时，将发送数据设置为偶校验；
- 步骤4 接收端收到数据后，将上报校验错误，接收端则识别为数据。

----结束

2.5.7 UART0 输入重定向

UART0输入支持重定向，可通过管脚UART_CTRL控制UART0的输入是来自URXD0还是URXD01，示意图如图2-3所示。

图 2-3 UART0 重定向功能





2.6 工作方式

说明

UARTx表示UART0、UART1。

须知

UART模块进行模式配置时，必须将数据收发工作停止，否则不能正确配置。

2.6.1 管脚复用配置

- UART0的URXD0和UTXD0管脚，使用UART0前，需要配置IOMG243[2:0]（物理地址为0x0FA8602B4）和IOMG244[2:0]（物理地址为0x0FA8602B8）为0，使能相应管脚的URXD0和UTXD0功能。
- UART0的URXD01和UTXD_CTRL管脚，使用UART0的重定向功能前，需要配置IOMG129[1:0]（物理地址为0x0FA8602C0）和IOMG248[1:0]（物理地址为0x0FA8602C4）为1，使能相应管脚的URXD01和UTXD_CTRL功能。
- UART1的URXD1和UTXD1管脚，使用UART1前，需要配置IOMG249（物理地址为0x0FA8602C8）和IOMG250（物理地址为0x0FA8602CC）为0，使能相应管脚的URXD1和UTXD1功能。

2.6.2 时钟门控

上电时三个UART时钟均默认为打开状态。UART工作完成后，可以关闭UART时钟，以降低芯片功耗。UART0、UART1使用不同的门控信号，它们的时钟可独立打开或关闭。

2.6.2.1 打开时钟

具体配置步骤如下：

- 步骤1** 读SUBCTRL系统控制器寄存器SC_UART_CLK_ST，获取当前时钟门控信息；
- 步骤2** 若SC_UART_CLK_ST [0]为0，则进入**步骤4**；SC_UART_CLK_ST [0]为1，则表示UART0时钟已经打开；
- 步骤3** 若SC_UART_CLK_ST [1]为0，则进入**步骤5**；SC_UART_CLK_ST [1]为1，则表示UART1时钟已经打开；
- 步骤4** 配置SC_UART_CLK_EN[0]为1，打开UART0模块的时钟门控；
- 步骤5** 配置SC_UART_CLK_EN[1]为1，打开UART1模块的时钟门控；
- 步骤6** 时钟打开后，需要等待10μs查看时钟门控状态，若SC_UART_CLK_ST对应bit为1，则进行下一步，否则认为存在异常。

----结束



2.6.2.2 关闭时钟

具体配置步骤如下：

- 步骤1** 读SUBCTRL系统控制器寄存器SC_UART_CLK_ST，获取当前时钟门控信息；
- 步骤2** 若SC_UART_CLK_ST [0]为1，则进入**步骤4**；SC_UART_CLK_ST [0]为0，则表示UART0时钟已经关闭；
- 步骤3** 若SC_UART_CLK_ST [1]为1，则进入**步骤5**；SC_UART_CLK_ST [1]为0，则表示UART1时钟已经关闭；
- 步骤4** 配置SC_UART_CLK_DIS [0]为1，关闭UART0模块的时钟门控；
- 步骤5** 配置SC_UART_CLK_DIS [1]为1，关闭UART1模块的时钟门控。

----结束

2.6.2.3 波特率配置

通过配置低位分频系数寄存器UART_DLL和高位分频系数寄存器UART_DLH，设置UART工作的波特率，波特率计算公式如下：

当前波特率=内部总线时钟频率（200MHz）/（16*分频系数）。

分频系数由高位和低位两部分组成，分别对应寄存器UART_DLH和UART_DLL。

UART波特率配置的典型值如下：

9,600bps、14,400bps、19,200bps、38,400bps、57,600bps、76,800bps、115,200bps、230,400bps、460,800bps。

2.6.3 软复位

对接器件容许的情况下，可以任意时间对UART工作逻辑进行软复位，不会对系统造成影响。

- 通过配置SUBCTRL系统控制器SC_UART_RESET_REQ相应位为0b1，可实现对UART工作逻辑的单独软复位。
- 通过配置SUBCTRL系统控制器SC_UART_RESET_DREQ的相应位为0b1，可实现对UART工作逻辑的单独软复位撤离，复位撤离配置后，需要等待10μs查看复位状态寄存器SC_UART_RESET_DREQ，若相应位为0，则进行后续操作，否则认为存在异常。

2.7 工作流程

2.7.1 中断或查询方式下的数据传输

2.7.1.1 初始化

配置步骤如下：

- 步骤1** 配置线性控制寄存器UARTx_LCR[7]为1，使UARTx_DLL与UARTx_DLH可访问。



- 步骤2** 写相应的配置值到低位分频系数寄存器UART_DLL和高位分频因子寄存器UARTx_DLH，配置传输速率。
- 步骤3** 配置寄存器UARTx_LCR[6:0]，设定UART的工作模式。
- 步骤4** 配置FIFO控制寄存器UART_FCR使能或者关闭FIFO，如果使能FIFO，设定相应的发送及接收FIFO水位值。
- 步骤5** 中断方式下配置中断使能寄存器UARTx_IEL相应位为1，使能相应中断；查询方式下配置中断使能寄存器UARTx_IEL相应位为0，禁止中断。

----结束

2.7.1.2 数据发送

具体配置步骤如下：

- 步骤1** 将待发送数据写入寄存器发送数据缓存寄存器UARTx_THR，启动数据发送。
- 步骤2** 查询方式下，进行连续数据发送时，读取发送FIFO水位寄存器UART_TFL和UART状态寄存器UARTx_USR检测发送FIFO状态，根据发送FIFO的状态决定是否向发送FIFO中发送数据；中断方式下，则检测相应中断状态，决定是否向发送FIFO中发送数据。
- 步骤3** 读取UART_USR[0]，直到该值为0。

----结束

2.7.1.3 数据接收

具体配置步骤如下：

- 步骤1** 查询方式下，读取接收FIFO水位寄存UARTx_RFL和UART状态寄存器UARTx_USR检测接收FIFO状态，根据接收FIFO的状态决定是否读取接收FIFO中的数据；中断方式下，根据相应中断状态位检测决定是否读取接收FIFO中的数据。
- 步骤2** 读取寄存器接收数据缓存寄存器UARTx_RBR，获取接收的数据。

----结束

2.7.2 DMA 方式下的数据传输

说明

UARTx表示UART0、UART1。

2.7.2.1 初始化

具体配置步骤如下：

- 步骤1** 配置线性控制寄存器UARTx_LCR bit[7]为1，使UARTx_DLL与UARTx_DLH可访问。
- 步骤2** 写相应的配置值到低位分频因子寄存UARTx_DLL和高位分频因子寄存器UARTx_DLH，配置传输速率。
- 步骤3** 配置寄存器UARTx_LCR bit[6:0]，设定UART的工作模式。
- 步骤4** 配置寄存器UARTx_FCR，使能或者关闭FIFO。如果使能FIFO，还需要设定相应的发送及接收FIFO水位值。



步骤5 配置配置寄存器UARTx_LCR bit[3]为1，使能DMA功能。

----结束

2.7.2.2 数据发送

具体配置步骤如下：

步骤1 配置DMA数据通道，包括数据传输源和目的地址、数据传输个数、传输类型等参数，其中数据传输的目的地址为发送数据缓存寄存器UARTx_THR。

步骤2 通过DMA中断上报，判断数据是否发送完成。

----结束

2.7.2.3 数据接收

具体配置步骤如下：

步骤1 配置DMA数据通道，包括数据传输源和目的地址、数据传输个数、传输类型等参数，其中数据传输的源地址为接收数据缓存寄存器UARTx_RBR。

步骤2 通过DMA中断上报，判断数据是否接收完成。

----结束



3 JTAG

在芯片中，按照功能的不同共包含有4类JTAG TAP控制器，第一类是ARM DAP控制器，主要功能是连接Ultrasoc调试系统，第二类是DJTAG TAP控制器，主要功能是连接可编程仿真器用于寄存器访问和部分寄存器初始化配置，第三类是DFT TAP控制器，主要功能用于边界扫描，第四类为AUTH TAP控制器，主要功能用于JTAG访问权限识别。本段落主要是对JTAG的功能和应用做详细描述。

3.1 概述

JTAG是芯片和关键子系统调试的主要控制接口，外部仿真器或者调试工具通过JTAG接口对芯片调试。本章节所指的JTAG包括DFT JTAG、ARM JTAG、DJTAG三种调试手段。

3.2 特性

JTAG具有以下特性：

- 板级提供标准的JTAG接口，每个DIE上封装一组标准JTAG接口，兼容IEEE1149.1JTAG标准、IEEE1149.6JTAG标准。
- JTAG接口支持外接调试工具，用于程序调试和单板制造测试。
- JTAG接口支持工作在2512MHz及以下的速率；挑战25M。
- 基于arm coresight调试系统。
- 支持独立的调试总线Debug Bus，提供总线异常情况下的问题定位手段。
- 支持多种JTAG配置通路，适应不同的调试场景和应用场景。
- 支持ATE测试、样片调试场景统一的JTAG接口。
- 支持管脚和内部JTAG寄存器不同配置，选择内部DFT JTAG、ARM JTAG和DJTAG在JTAG的通路上的有无。
- 支持安全调试。
- 提供JTAG鉴权。
- JTAG管脚选用带施密特触发的IO。



3.3 关键术语

表3-1中的术语将在本章节表示特殊的含义。

表 3-1 JTAG 关键术语描述

术语	描述
JTAG	Joint Test Action Group 联合测试行动小组。

3.4 遵循协议

- 支持IEEE1149.1 JTAG标准；
- 支持IEEE1149.6 JTAG标准。

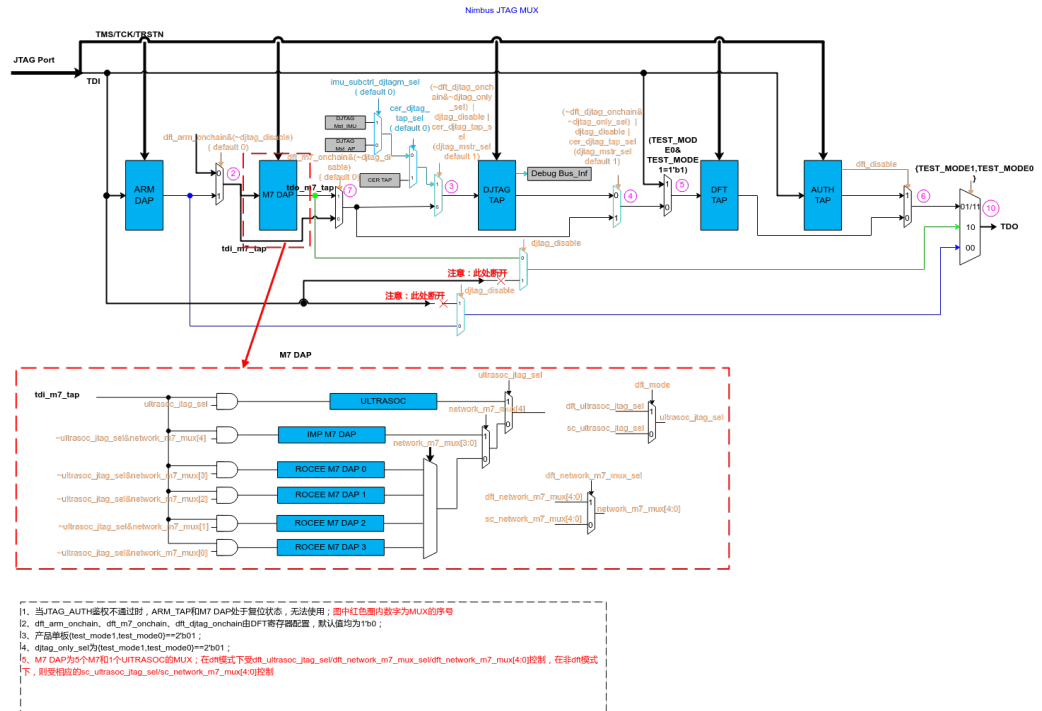
3.5 功能描述

3.5.1 应用框图

3.5.1.1 JTAG 应用框图

Hi309a JTAG拓扑结构，采用Nimbus JTAG拓扑结构扩展而来。与Nimbus JTAG拓扑结构不同的是，Hi309a的JTAG拓扑结构包含有6个JTAG TAP控制器：ARM TAP（coresight）、M7 TAP（UCE）、DFT TAP、DJTAG TAP、HISS TAP和AUTH TAP。其中AUTH TAP控制器用于JTAG访问权限识控制，和其他5个TAP控制器互斥，不可同时使用；另外5个TAP控制器在Hi309a系统内均是菊花链的方式首尾连接。

图 3-1 Nimbus JTAG 拓扑结构图



各个TAP控制器的描述如下：

- M7 TAP控制器：**
指用于DDR UCE调试系统的TAP控制器。
- ARM TAP控制器：**
指用于CORESIGHT调试系统的TAP控制器，支持系统ARM A55核侵入式和非侵入式调试。
- DJTAG TAP控制器：**
指连接可编程仿真器，用于寄存器访问的TAP控制器，注意，只有串入DJTAG TAP链的寄存器才能支持DJTAG访问。DJTAG的访问通路有两条：片外仿真器访问和片内CPU核访问。当DJTAG TAP控制器在菊花链上未被使能时，DJTAG可被CPU核访问，用于芯片内部Memory的MBIST测试；当DJTAG TAP控制器在菊花链上被使能时，片外仿真器可通过DJTAG TAP访问芯片内部寄存器。
- DFT TAP控制器：**
DFT TAP控制器主要用于ATE测试，具体功能有：Boundary scan边界扫描测试、Memory MBIST测试、All scan测试。功能十分强大，一般也作为JTAG 拓扑的主控TAP，用来配置其他TAP的功能选通。
- AUTH TAP控制器：**
指用于JTAG访问权限识别的TAP控制器，系统启动后，外部仿真器工具可通过AUTH TAP获取JTAG口的访问权限。若AUTH TAP获取JTAG口访问权限失败，外部仿真器则无法访问ARM DAP、DJTAG TAP、DFT TAP和HISS TAP，若AUTH TAP获取权限成功，AUTH TAP控制器失效，直到下次上电复位。
- HISS TAP控制器：**
高安子系统的TAP控制器，可以访问HISS内部CPU核以及HISS内部的寄存器空间。类似于ARM TAP和DJTAG的功能，属于HISS子系统私有的TAP控制器。



JTAG使用之前需要鉴权，鉴权TAP为AUTH TAP，AUTH TAP指令和DJTAG TAP指令一致，使用了DBG_CHAIN0 ~ DBG_CHAIN2，IR指令可参考[表3-3](#)。

3.5.1.2 JTAG 拓扑配置

Hi309a的ARM TAP、M7 TAP、DJTAG、DFT TAP和HISS TAP以首尾菊花链的形式连接，连接形式通过管脚TEST_MODE0、管脚TEST_MODE1、JTAG配置信号dft_arm_onchain、dft_m7_onchain、dft_djtag_onchain和dft_hiss_onchain的组合配置JTAG chain，当JTAG鉴权通过时，JTAG的chain链和对应TEST_MODE0/1管脚，以及dft配置信号的关系如[表3-2](#)。

表 3-2 JTAG Chain 链配置表

配置方式	TEST_MODE1 (管脚)	TEST_MODE0 (管脚)	dft_arm_onchain	dft_djtag_onchain	dft_m7_onchain (UCE)	Dft_hiss_onchain	JTAG 连接模式
0	1	1	*	*	*	1	DFT JTAG 和 HISS JTAG
1	1	1	*	*	*	0	DFT JTAG
2	0	0	*	*	*	0	ARM JTAG
3	1	0	*	*	*	0	M7 DAP (UCE)
4	0	1	0	0	0	0	菊花链连接方式：只有DFT JTAG，但SYSCTRL可通过DJTAG配置寄存器。
5	0	1	0	1	0	0	菊花链连接方式：DJTAG和DFT JTAG在菊花链上。



配置方式	TEST_MODE1 (管脚)	TEST_MODE0 (管脚)	dft_arm_onchain	dft_djtag_onchain	dft_m7_onchain (UCE)	Dft_hiss_onchain	JTAG 连接模式
6	0	1	1	0	0	0	菊花链连接方式： ARM JTAG 和 DFT JTAG在菊花链上， SYSCTRL 可通过 DJTAG配置寄存器。
7	0	1	0	0	1	0	菊花链连接方式： M7 DAP和 DFT JTAG 在菊花链上。
8	0	1	1	1	1	0	菊花链连接方式： ARM JTAG、DJTAG、 M7 DAP和 DFT JTAG 在菊花链上。

JTAG典型应用场景如下：

- ATE测试：
使用配置方式1，DFT JTAG通过JTAG接口配置，且ARM JTAG和DJTAG 的TAP控制器处于复位状态。
- 样片测试：
使用配置方式1或配置方式4，用于JTAG扫描；
使用配置方式2，用于ARM仿真器调试，DFT JTAG处于复位状态，CPU 可通过SYSCTRL 访问DJTAG配置和debug寄存器；
使用配置方式6，用于ARM仿真器调试，ARM JTAG和DFT JTAG菊花链连接，可通过指令将DFT JTAG bypass掉；
使用配置方式7，用于ARM仿真器调试，ARM JTAG、DJTAG和DFT JTAG通过菊花链连接，可通过指令将DJTAG和DFT JTAG bypass掉，也可通过ARM仿真器配置DJTAG访问各模块寄存器。

- 装备测试：
使用配置方式4，用于JTAG扫描。
- JTAG加载SPI Nor FLASH：
通过JTAG控制SYSCTRL的配置寄存器，启动SYSCTRL内AXI_MSTR0口的访问，AXI总线的访问地址配置为SFC地址，AXI总线的数据位宽为128，所以一次AXI操作会配置4个SFC寄存器。

3.5.2 SOC JTAG

3.5.2.1 SOC JTAG 概述

SOC JTAG即指DFT TAP控制器，SOC JTAG主要实现芯片的边界扫描功能，边界扫描电路（Boundary scan）又被称为JTAG（Joint Test Action Group）是一种应用于板级芯片互连测试的DFT技术。使用JTAG电路显著提高了板级测试的效率和质量，缩短了生产测试和诊断的过程。目前海思的JTAG电路是按照IEEE 1149.1标准实现的。

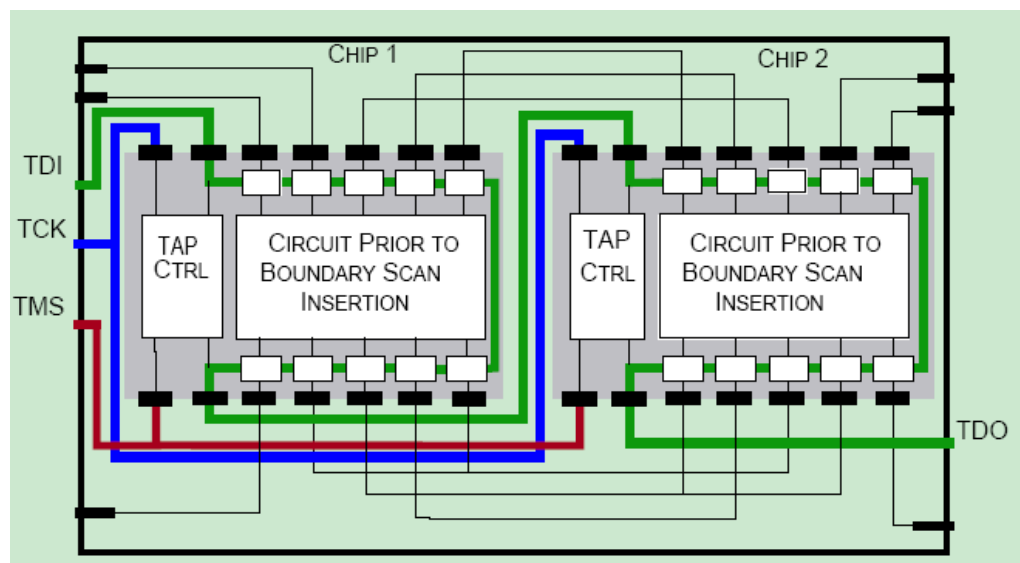
使用边界扫描电路可以有效检测出单板在生产过程中引入的缺陷，例如使用错误组件，遗漏组件，芯片摆放方向错误，芯片的stuck pin错误，短路、开路等缺陷。

由于边界扫描结构嵌入在IC内部，具有边界扫描功能的元件的成本有所增加。但使用边界扫描电路可以缩短产品开发时间，由此而实现的成本效益和节省的测试成本远远高于元件级增加的成本。

JTAG将芯片的输入和输出串成一条scan path，通过这个scan path来控制或访问芯片的输入和输出端口。数据沿着JTAG的scan path，从TDI（Test Data In）串行移入，从scan path末端连接的TDO（Test Data Out）移出。在TDI和TDO之间，串联了板上所有包含JTAG电路的芯片。上一级芯片的TDO会连接下一级芯片的TDI。其他的信号，包括TCK（Test Clock）和TMS（Test Mode Select），并行连接到各个包含JTAG电路的芯片。通过这种结构可以进行芯片板级的互连测试或者单独对某一芯片进行测试。TAP（Test Access Port）控制器通过状态机实现边界扫描电路的各种操作。

JTAG电路的主要应用是芯片的板级互连测试，另外它也可以控制芯片内部的测试结构，例如BIST。

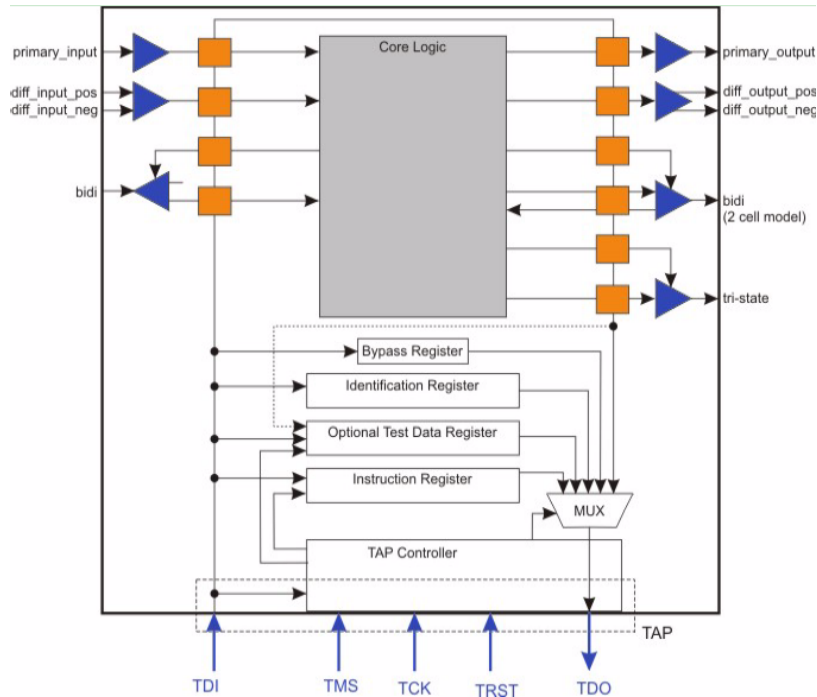
图 3-2 Boundary Scan Chips on Board



3.5.2.2 SOC JTAG 电路结构及原理

边界扫描硬件结构如图3-3所示。

图 3-3 JTAG 硬件结构



其基本结构分为四部分：测试存取通道（TAP）、指令寄存器（IR）、数据寄存器组（DR）、TAP 控制器。

- 测试存取通道（TAP）

TAP（Test Access Port）由5跟信号组成，这些信号包括：

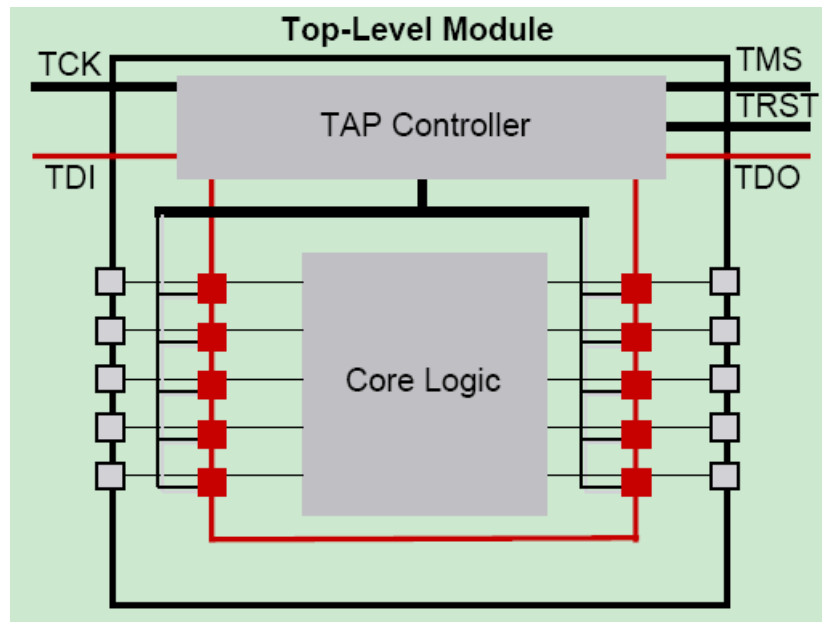
- 测试时钟（TCK：Test Clock）：TDI的数据和功能管脚上的数据由TCK的上升沿采样，TCK下降沿输出TDO的数据。该时钟的要求是：必须独立，不得与任何系统时钟相干扰。该信号采用下拉PAD。
- 测试模式选择（TMS：Test Mode Select）：控制TAP controller中的状态跳转。该信号采用上拉PAD。
- 测试数据输入（Test Data Input）：测试数据的输入信号。该信号采用上拉PAD。
- 测试数据输出（Test Data Output）：测试数据输出端口，无数据输出时为高阻态。该信号不需要上下拉。
- 测试复位（TRST：Test Reset）：0有效，可选信号；当TMS维持5个时钟周期的高电平时，JTAG也会被复位。该信号采用上拉PAD。

- 数据寄存器组

数据寄存器组必须存在的寄存器有两类：bypass寄存器和边界扫描寄存器。另外器件标识寄存器和用户自定义寄存器为可选的。

JTAG电路中在设计CORE层和PAD之间添加的BC Cell组成了边界扫描寄存器。图3-4中红色部分即为边界扫描寄存器。

图 3-4 边界扫描寄存器



- 指令寄存器

一个JTAG只有一组指令寄存器。当指令寄存器被选定后，可以从TDI进行指令移位，完成装载JTAG指令的工作。JTAG指令所产生的操作会影响到数据寄存器内的数据。

每个用户自定义数据寄存器都对应一个指令，当需要对某一数据寄存器进行移位操作时，指令寄存器内的值就需要被配置成该数据寄存器的指令值。

JTAG的强制指令包括：

- EXTEST：该指令用于测试外部电路到芯片的路径，例如板级互连。EXTEST是边界扫描测试的最主要的一个测试项。
- SAMPLE/PRELOAD：该指令从芯片的I/O pad上采样数据并锁存在边界扫描寄存器中（该过程可以在芯片板级正常使用的情况下完成）。IEEE 1149.1-2001将该指令拆分成两个独立的指令（SAMPLE和PRELOAD）。
- SAMPLE：该指令用于采样芯片I/O pad上的数据并锁存于扫描寄存器中（该过程可以在芯片板级正常使用的情况下完成）。
- PRELOAD：该指令允许在选择另外一个指令前将数据装载到边界扫描寄存器中（该过程可以在芯片板级正常使用的情况下完成）。
- BYPASS：该指令用于bypass所选芯片不予测试。例如，如果单板上有多颗芯片串连在边界扫描寄存器链上，而我们只想测试其中一颗芯片，那么除了需要测试的那颗芯片外，其他芯片都需要被配置成bypass指令。这样在串行移位数据时，每个bypass的芯片只需要多加一个shift就可以完成了。
- EXTEST_PULSE：该指令应用于存在AC管脚的芯片，用于测试外部电路到芯片的路径。

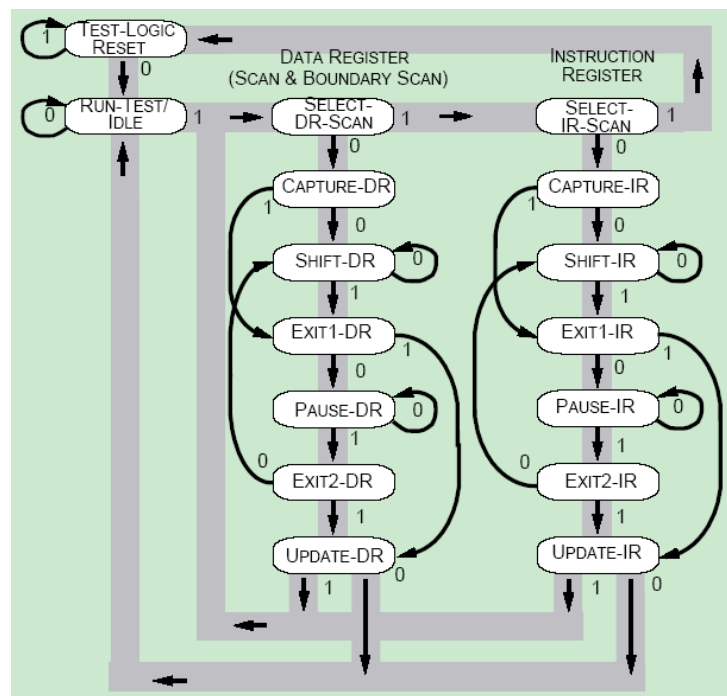
JTAG的可选指令包括：

- INTEST：该指令用于测试芯片内部逻辑，需要内部测试电路配合使用。指令会作用一个测试向量到内部电路上，然后捕获该电路的输出结果。
- IDCODE：该指令用于输出芯片的ID编码，常用来检测单板所用芯片是否正确。

- c. USERCODE: 该指令用于输出作为ID编码补充信息的内容，由用户自定义。
- d. CLAMP: 该指令通过将某些节点force成1或0，以阻塞接口信号。
- e. HIGHZ: 该指令将芯片的输出和双向管脚配置成高阻态。
- f. RUNBIST: 该指令执行电路的内部BIST程序，需要内部电路配合使用。
- g. EXTEST_TRAIN: 该指令用于含有AC cell的情况。
- TAP控制器

TAP控制器的基本功能是产生时钟信号和控制信号，这些信号是指令寄存器和数据寄存器正常工作所要求的，包括：提供信号允许将指令装入指令寄存器中、提供信号将数据通过TDI管脚移入移位寄存器中，并把数据通过TDO管脚从移位寄存器输出、执行相应功能，如捕获、移位和更新数据等、TAP控制器的状态跳转如图3-5所示，图中状态跳转发生在TCK的上升沿，跳转条件由TMS控制。系统上电后处于复位状态（TEST_LOGIC_RESET）。

图 3-5 TAP 控制器的状态机跳转图



下面介绍该状态机的几个重要状态：

- a. TEST_LOGIC_RESET，测试逻辑复位状态：TRST的一个低电平可以进入该状态，另外TMS维持5个TCK周期的高电平也可以进入该状态。TAP控制器处于复位状态时，mode = 0，测试逻辑失去作用，BC_Cell的PI和PO直通，芯片功能逻辑正常工作；
- b. RUN_TEST/IDLE，运行-测试/空闲状态：当TAP处于这个状态时，只要TMS信号保持为0，就可以停留在该状态。相应的测试逻辑取决于指令寄存器中存放的指令。TAP控制器保持在该状态是为了等待某一指令执行完成（例如RUNBIST），在指令执行完成之前不得进入其他状态；
- c. CAPTURE_DR，捕获数据寄存器状态：在TCK上升沿时，将数据从数据寄存器的并行输入端口load到数据寄存器中。



- d. SHIFT_DR, 移位数据寄存器状态: 在每个TCK上升沿, 前一数据移位寄存单元的数据移向下一数据移位寄存单元, 最后一个数据移位寄存单元的数据从TDO移出, 最前面一个数据移位寄存单元的数据从TDI移入。
- e. UPDATE_DR, 更新数据寄存器状态: TCK的下降沿将数据寄存器的新数据所存在它们的并行输出端。所选择的数据寄存器中移位寄存器将保持它以前的状态。
- f. CAPTURE_IR, 捕获指令寄存器状态: 该状态下, TCK的上升沿将指令数据并行load指令寄存器的移位寄存器中。
- g. SHIFT_IR, 移位指令寄存器状态: 该状态下, TCK的上升沿将前一指令移位寄存单元的数据移向下一指令移位寄存单元。
- h. UPDATE_IR, 更新指令寄存器状态: 该状态下, TCK的下降沿将已移入的指令数据从移位寄存器load并行指令寄存器中, 则新的指令变为有效。

3.5.3 ARM JTAG

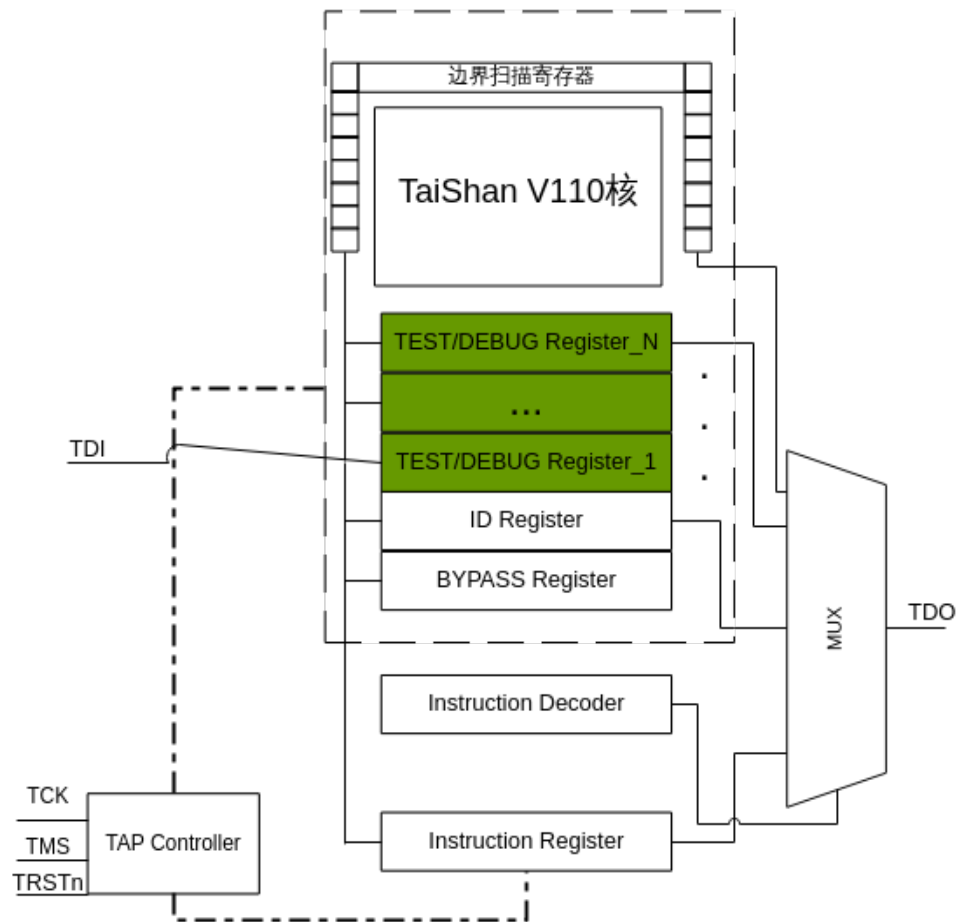
3.5.4 DJTAG

3.5.4.1 DJTAG 概述

DJTAG即DJTAG TAP, DJTAG TAP包括TAP控制器, 指令寄存器、指令译码器、旁路寄存器、ID寄存器、测试/调试寄存器, TDO多选逻辑。DJTAG控制器采用Mentor Graphics的BSDArchitect工具产生。JTAG的5个接口管脚最初专用于边界扫描测试需求。通过逐步扩展, 另外一个重要的用途就是测试、调试接口, 用于芯片内部的测试和调试。在实现阶段DFT添加其余的DFT逻辑, 如边界扫描寄存器、ID JTAG寄存器和测试模式寄存器等, Hi309a统一的测试、调试控制模块如图3-6所示。



图 3-6 统一调试、测试结构图



3.5.4.2 DJTAG 指令

表 3-3 DJTAG 指令

指令名	Code	目标寄存器	JTAG寄存器
EXTEST	00000000	-	标准BSD指令。
BYPASS	11111111	-	标准BSD指令。
SAMPLE	00000001	-	标准BSD指令。
PRELOAD	00000001	-	标准BSD指令。
CLAMP	00000010	-	标准BSD指令。
HIGHZ	00000011	-	标准BSD指令。
IDCODE	00000100	-	标准BSD指令。
DBG_CHAIN0	00100000	-	DJTAG Debug链0。



指令名	Code	目标寄存器	JTAG寄存器
DBG_CHAIN1	00100001	-	DJTAG Debug链1。
DBG_CHAIN2	00100010	-	DJTAG Debug链2。
DBG_CHAIN3	00100011	-	DJTAG Debug链3。
DBG_CHAIN4	00100100	-	DJTAG Debug链4。
DBG_CHAIN5	00100101	-	DJTAG Debug链5。
DBG_CHAIN6	00100110	-	DJTAG Debug链6。
DBG_CHAIN7	00100111	-	DJTAG Debug链7。
DBG_CHAIN8	00101000	-	DJTAG Debug链8。
DBG_CHAIN9	00101001	-	DJTAG Debug链9。
DBG_CHAIN10	00101010	-	DJTAG Debug链10。
DBG_CHAIN11	00101011	-	DJTAG Debug链11。
DBG_CHAIN12	00101100	-	DJTAG Debug链12。
DBG_CHAIN13	00101101	-	DJTAG Debug链13。
DBG_CHAIN14	00101110	-	DJTAG Debug链14。
DBG_CHAIN15	00101111	-	DJTAG Debug链15。
DBG_CHAIN16	00110000	-	DJTAG Debug链16。
DBG_CHAIN17	00110001	-	DJTAG Debug链17。
DBG_CHAIN18	00110010	-	DJTAG Debug链18。
DBG_CHAIN19	00110011	-	DJTAG Debug链19。
DBG_CHAIN20	00110100	-	DJTAG Debug链20。
DBG_CHAIN21	00110101	-	DJTAG Debug链21。
DBG_CHAIN22	00110110	-	DJTAG Debug链22。
DBG_CHAIN23	00110111	-	DJTAG Debug链23。
DBG_CHAIN24	00111000	-	DJTAG Debug链24。
DBG_CHAIN25	00111001	-	DJTAG Debug链25。
DBG_CHAIN26	00111010	-	DJTAG Debug链26。
DBG_CHAIN27	00111011	-	DJTAG Debug链27。
DBG_CHAIN28	00111100	-	DJTAG Debug链28。
DBG_CHAIN29	00111101	-	DJTAG Debug链29。
DBG_CHAIN30	00111110	-	DJTAG Debug链30。



指令名	Code	目标寄存器	JTAG寄存器
DBG_CHAIN31	00111111	-	DJTAG Debug链31。
DBG_CHAIN32	01000000	-	DJTAG Debug链32。
DBG_CHAIN33	01000001	-	DJTAG Debug链33。
.....		-	DJTAG Debug链N。
DBG_CHAIN63	01011111	-	DJTAG Debug链63。

3.5.4.3 Debug Bus

Debug Bus是独立于系统总线之外专用于调试的总线，通过JTAG接口访问系统内部的地址空间。首尾串行连接不同模块的JTAG寄存器构成了Debug Bus，即DJTAG TAP的TDI输出信号连接到Debug Bus上第一个模块的si端口，第一个模块的so连接到第二个模块的si端口，以此串行相连。Debug Bus上最后一个模块JTAG寄存器的so输出返回DJTAG，通过TDO_MUX输出到TDO管脚。其他JTAG控制信号连接到JTAG寄存器上的其他的控制端口上。给DJTAG控制器的指令寄存器加载相应的调试指令来使能不同子系统的debug bus，同一时刻只能一条Debug Bus执行调试操作，而可以同时访问这个子系统的不同模块的同一个地址。

DJTAG寄存器的数据格式为66bit，DJTAG寄存器能够捕捉并行数据、串行移位和并行更新。寄存器格式如图3-7所示。

图 3-7 DJTAG 寄存器格式



表 3-4 仿真器访问 DJTAG 寄存器读写命令格式说明

Bits	Access	Width	Name	Description
[65]	W	1	CS	bit[65]: 操作有效标志位，高有效。
[64]	W	1	R/W	bit[64]: 寄存器读写控制信号。 0: 读操作; 1: 写操作。
[63]	W	1	PROT_NS	bit[63]: 访问安全位。 0: 非安全访问; 1: 安全访问。
[62:32]	W	31	ADDR	bit[62:32]: 寄存器读写访问地址。
[31:0]	W	32	DATA	寄存器写数据。 读操作时该字段保留，默认全0; 写操作为数据字段。



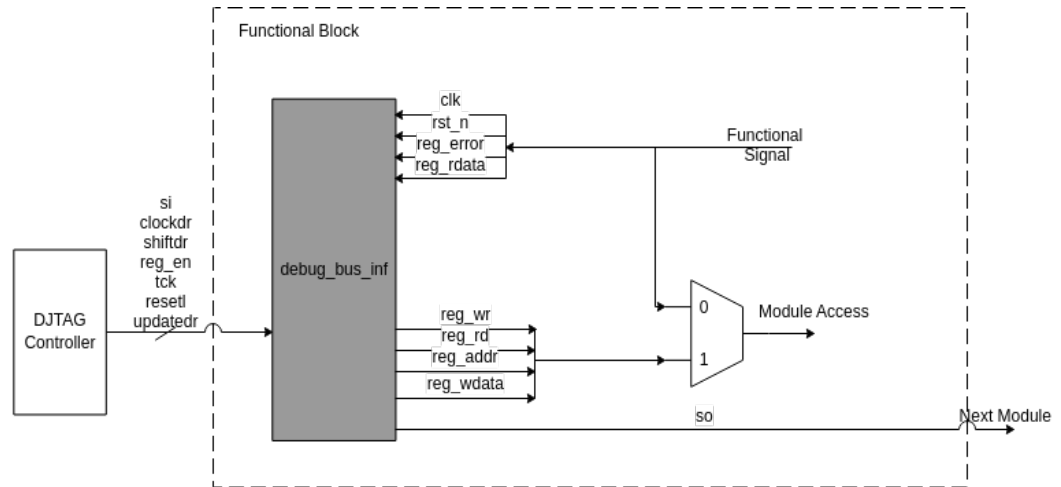
表 3-5 仿真器访问 DJTAG 寄存器读返回数据格式说明

Bits	Access	Width	Name	Description
[65]	R	1	CS	TDRE类寄存器。 bit[65]: 保持操作命令的值，即片选信号； 非TDRE类寄存器。 bit[65]: 为操作冲突错误标记位； 0: 操作无误，结果可用； 1: 操作出错，结果不可用。
[64]	R	1	R/W	TDRE类寄存器。 bit[64]: 保持操作命令的值，即读写使能； 非TDRE类寄存器。 bit[64]为读有效位； 0: 读无效； 1: 读有效。
[63]	R	1	PROT_NS	bit[63]: 访问安全位，保持操作命令的值。 0: 非安全访问； 1: 安全访问。
[62:32]	R	31	ADDR	bit[62:32]: 寄存器读访问地址，保持操作命令的值。
[31:0]	R	32	DATA	返回寄存器读数据。

Debug Bus的主要功能是把JTAG访问翻译成功能访问，从而实现从JTAG端口访问功能模块的地址空间。

在Debug Bus给出reg_rd读使能信号之后，外部模块如果不是访问error，需要锁存对应的数据，在debug_bus_inf发出下次操作之前，数据需稳定不变。debug_bus_inf模块在2个TCK时钟之后，才会使用读取的数据，所以对于需要多个工作时钟周期访问才能获取数据的模块，在一个TCK时钟内，能够把数据读出，均可正常工作。TCK时钟频率为25MHz，对应时间为40ns，外部模块只要在这个时间内能够提供数据均可正常工作。

图 3-8 Debug Bus 接口图



3.6 软件指南

3.6.1 初始化

Debug Bus使用参考JTAG STUDIO用户手册。

3.7 注意事项

仿真器进行了DJTAG调试，调试结束后并未拔出仿真器，或配置DFT TAP将DJTAG的mux端切回内部，由于DJTAG的mux端由JTAG复位或者DFT控制，并不跟随芯片热复位，此时，若芯片直接热复位，DJTAG仍然由仿真器控制，CPU无法获得DJTAG的访问权，而导致CPU无法配置总线，启动失败；该场景并非真实的使用场景，可以认为是操作不当导致，即使出现问题，也可以通过在调试结束后拔出仿真器，或通过仿真器发出复位指令，复位DJTAG的mux端，将DJTAG的控制权归还CPU。



4 DJTAG Master

本段落主要是对DJTAG Master的功能和应用做详细描述。

4.1 概述

DJTAG Master是将CPU配置的寄存器时序转换成标准的JTAG接口时序，如[图4-1](#)所示，CPU通过AMBA总线配置DJTAG对应的寄存器组，配置完成后DJTAG Master将寄存器信息转换成JTAG标准时序，在DJTAG TAP控制器中译码，转换成特定模块的寄存器访问时序。

4.2 特性

DJTAG Master支持以下特性：

DJTAG Master支持TDRE类寄存器访问：包括Cluster、TSV、HHA、LLC、BISR等模块的TDRE寄存器，此类寄存器的访问是通过DJTAG访问模块包含的TDRE接口实现，通过DJTAG配置TDRE寄存器，启动对应模块的Memory/TSV BIST。

4.3 遵循协议

支持AMBA APB2.0标准。

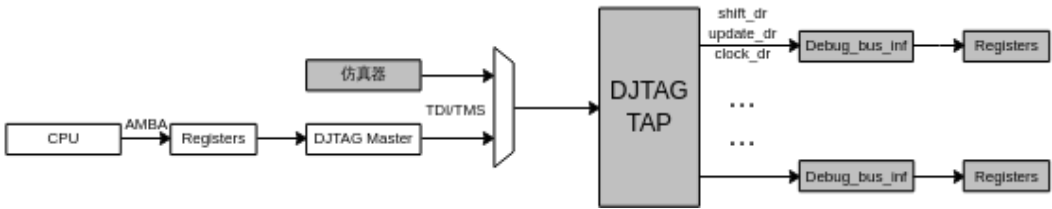
4.4 功能描述

4.4.1 应用框图

DJTAG Master的DBI Slave主要为TDRE以及BISR模块。



图 4-1 DJTAG Master 数据流图



DJTAG的寄存器组（参考表4-1）需要软件保证原子性操作，在一次DJTAG访问结束前，不允许再次发起DJTAG访问操作。

SCCL DJTAG的寄存器组放在Sysctrl的寄存器内，具体如表4-1所示。

表 4-1 DJTAG Master 寄存器组

寄存器名	偏移地址	类型
SC_DJTAG_SEC_ACC_EN	0xD800	RW
SC_DJTAG_MSTR_ADDR	0xD810	RW
SC_DJTAG_MSTR_DATA	0xD814	RW
SC_DJTAG_MSTR_CFG	0xD818	RW
SC_DJTAG_MSTR_START_EN	0xD81C	RW
SC_DJTAG_MSTR_PIPE_CFG	0xD820	RW
SC_DJTAG_TMOUT	0xD840	RW
SC_DJTAG_SEC_MODE	0xD844	RW
SC_DJTAG_RD_DATA0	0xE800	RO
SC_DJTAG_RD_DATA1	0xE804	RO
SC_DJTAG_RD_DATA2	0xE808	RO
SC_DJTAG_RD_DATA3	0xE80C	RO
SC_DJTAG_RD_DATA4	0xE810	RO
SC_DJTAG_RD_DATA5	0xE814	RO
SC_DJTAG_RD_DATA6	0xE818	RO
SC_DJTAG_OP_ST	0xE828	RO

4.4.1.1 DJTAG Master 访问配置流程



4.4.1.1.1 DJTAG Master 访问 Debug bus 链配置流程

DJTAG Master访问Debug bus链上模块的配置步骤如下：

- 步骤1** 若本次访问涉及到安全寄存器，则需要安全CPU配置寄存器SC_DJTAG_SEC_ACC.djtag_sec_acc_en（偏移地址0xD800）为1。若本次访问不涉及安全寄存器，则无需配置该寄存器。
- 步骤2** 如果本次访问涉及Cross Station、LLC、HHA、Access Agent、SLLC，CPU可通过配置寄存器SC_DJTAG_FAST_MODE.djtag_fast_mode（偏移地址0xD804）来加快访问速度，写入0为低速访问，写入1为高速访问，若不为上述模块，该寄存器必须写入0。
- 步骤3** CPU配置寄存器SC_DJTAG_MSTR_ADDR.djtag_mstr_addr（偏移地址0xD810），写入被访问模块的内部寄存器的偏移地址。
- 步骤4** CPU配置寄存器SC_DJTAG_MSTR_DATA.djtag_mstr_data（偏移地址0xD814），写入写数据（如果是读则省略该步骤）。
- 步骤5** CPU配置寄存器SC_DJTAG_MSTR_CFG（偏移地址0xD818），完成本次DJTAG的相关设置。
1. chain_unit_cfg_en：只针对一条链上都是相同的模块，如Cross Station、Access Agent，用于设置是否将当次配置应用于链上的所有模块，默认值为全1，即默认打开一次配置应用于所有模块。
 2. debug_module_sel：写入需要操作的模块对应的地址。
 3. djtag_mstr_wr：该次DJTAG操作的读写属性，0为读，1为写。
 4. djtag_nor_cfg_en：切到PLL前，写入0，切到PLL后，写入1。
 5. djtag_mstr_disable：禁止该次DJTAG操作，默认为0，允许本次DJTAG操作。
- 步骤6** CPU配置寄存器SC_DJTAG_MSTR_START_EN.djtag_mstr_start_en（偏移地址0xD81C）为1，使能DJTAG访问。
- 步骤7** CPU访问寄存器SC_DJTAG_OP_ST（偏移地址0xE828）中的djtag_op_done，为1，表示DJTAG访问结束。访问寄存器SC_DJTAG_OP_ST（偏移地址0xE828）中的unit_conflict，为0表示访问正常，不为0表示访问存在异常。若是Cross Station、Access Agent模块的读操作，读数据分别存放在寄存器SC_DJTAG_RD_DATA0~9（偏移地址0xE800~0xE824）寄存器内，否则读数据通过寄存器SC_DJTAG_RD_DATA0（偏移地址0xE800）读取。
- EMU、FPGA的DJTAG配置和使用请参考DJTAG Master访问Debug bus配置步骤，因为资源限制问题，导致版本中的DJTAG链和文档描述的DJTAG链不一致时需要注意：
1. Cross Station、Access的链上所有模块统一配置写不受链变化的影响。
 2. DJTAG Master访问Slave请参考《Hi309a AS架构图》和《Hi309a AS架构图》DJTAG页的DJTAG CHAIN。

----结束

4.4.1.1.2 DJTAG Master 安全和非安全访问流程

DJTAG Master默认只能安全访问，可通过安全模式下授权非安全模式下DJTAG Master的访问权限。

授权非安全模式流程如下：

- 步骤1** 当DJTAG Master处于访问Debug bus链配置流程，CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828）中的djtag_op_done，通过读到的值1确认DJTAG访问结束。或者确认当前DJTAG Master处于空闲时刻。



步骤2 CPU清除DJTAG Master上的敏感数据，确认切换到非安全访问流程是不会造成数据泄露。

步骤3 配置SC_DJTAG_SEC_MODE. dntag_sec_mode（偏移地址0xD844）为0，授权非安全模式下DJTAG Master的访问权限。

步骤4 非安全模式下遵照DJTAG Master访问Debug bus链配置流程进行访问。

----结束

收回非安全模式流程下的访问权限流程：

步骤1 配置SC_DJTAG_SEC_MODE. dntag_sec_mode（偏移地址0xD844）为1，取消非安全模式下DJTAG Master的访问权限授权。

步骤2 非安全模式下遵照DJTAG Master访问Debug bus链配置流程进行访问。

----结束

4.5 注意事项

SPPE掉电后，SPPE所在的DJTAG链整条链的访问无法进行。

图 4-2 DJTAG 链

Chain8	8'b0010_1000	SMMU (SPPE) (8'h70)	SCHE (SPPE) (8'h71)	DISP (SPPE) (8'h72)	FTE (SPPE) (8'h73)	SPPE (SPPE) (8'h74)	SA (SPPE) (8'h75)	CodeTDM (SPPE) (8'h76)
--------	--------------	---------------------------	---------------------------	---------------------------	--------------------------	---------------------------	-------------------------	------------------------------



5 JTAG-AUTH

5.1 关键术语

表 5-1 JTAG-AUTH 关键术语描述

术语	描述
JTAG	Joint Test Action Group 联合测试行动小组。
E-Fuse	Electrical-Fuse，一次性的电可编程的Fuse。
RSA	RSA算法的名字以发明者的名字命名：Ron Rivest，Adi Shamir 和Leonard Adleman。RSA属于非对称加密算法，自1977年提出至今，经历了各种攻击的考验，逐渐为人们接受，普遍认为是目前最优秀的非对称加密方案之一。
ROTAUK	JTAG认证所使用公钥的SHA256 HASH值，在JTAG认证打开JTAG调试接口时使用。JTAG认证服务器基于RSA算法产生JTAG认证需要的公私钥密钥对，所有芯片共用一堆。其中私钥存储在JTAG认证服务器中，公钥随认证证书发放，公钥HASH值存在芯片E-Fuse中。把公钥{N,E}看成0x{N,E}，然后进行计算HASH值（使用SHA256算法）。
OCM	Original Challenge Message，原始挑战信息，由随机数和芯片AUTH-KEY ID组成。鉴权服务器会对OCM进行签名，由芯片鉴权模块完成签名的校验。
CM（也称OAM）	Challenge Message，挑战信息。由鉴权服务器根据OCM产生的。具体是鉴权服务器对OCM进行签名，然后连同签名公钥，一并构成CM。
JTAGAUTH_KEYID	JTAG鉴权密钥ID；通常一个版本单板用一个KEY。

5.2 概述

为了解决JTAG接口强大调试特性与系统安全的矛盾，我们必须限制JTAG接口的访问权限。JTAG_AUTH（JTAG鉴权）是用于控制芯片JTAG接口访问权限的模块，芯片上的每个JTAG口都需要用一个JTAG-AUTH模块管理JTAG访问权限。

JTAG鉴权等级分为：

- JTAG接口不需要鉴权，允许所有JTAG调试，包括ARM JTAG、DFT JTAG、DJTAG、HISS JTAG。
- JTAG接口不需要鉴权，仅允许使用DFT JTAG调试，禁止ARM JTAG、DJTAG调试，HISS JTAG单独efuse域段控制。
- JTAG鉴权模式使能方式1，鉴权成功允许所有JTAG调试，包括ARM JTAG、DFT JTAG、DJTAG、HISS JTAG单独eFuse域段控制。
- JTAG鉴权模式使能方式2，鉴权成功允许DFT JTAG调试，禁止ARM JTAG、DJTAG调试、HISS JTAG单独efuse域段控制。
- JTAG接口永久关闭。

鉴权等级在生产阶段固化在芯片的eFuse中。

鉴权流程如图5-1所示。

图 5-1 JTAG 鉴权流程

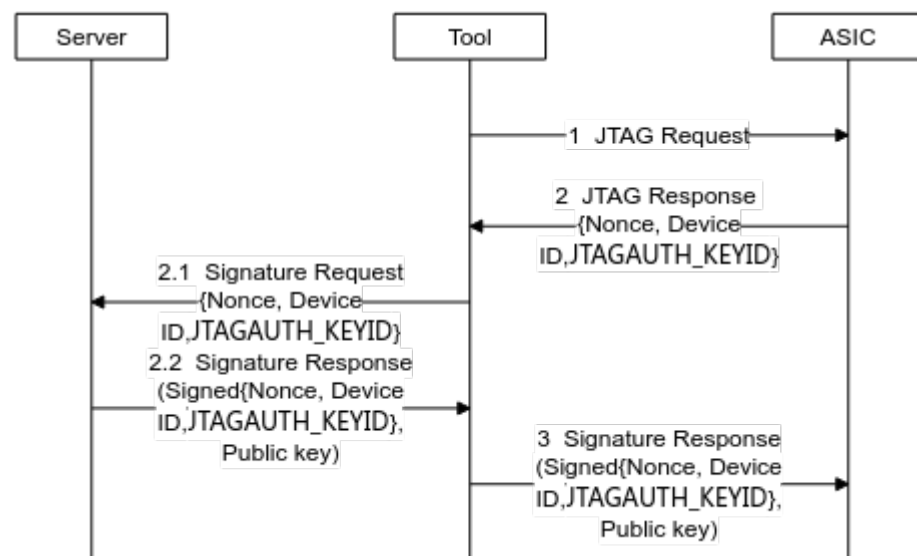


图5-1中，ASIC是指芯片，Tool是指鉴权过程中需要的JTAG工具及软件，Server是指完成JTAG鉴权签名计算的服务器。

1. 鉴权工具（Tool）连接上芯片的JTAG接口，首先查询ASIC（即内部JTAG_AUTH鉴权模块）是否可以JTAG鉴权操作，若可以鉴权，则Tool发送JTAG鉴权请求。
2. 芯片通过JTAG口向Tool反馈挑战原始信息{随机数Nonce，设备号Device ID，鉴权密钥号JTAGAUTH_KEYID}，并且自己存储下发送的挑战原始信息
 - a. Tool把原始挑战信息发送给远程的鉴权服务器，Tool和鉴权服务器之间的安全性通过网络安全协议保证。

- b. 鉴权服务器使用RSA私钥对原始挑战信息进行签名，然后连同RSA公钥一并发给Tool。
3. Tool把服务器返回的挑战信息，通过JTAG接口发送给芯片，芯片根据挑战信息和原始挑战信息进行鉴权测试。

JTAG-AUTH设计简图如下图5-2所示。

图 5-2 JTAG 鉴权框图

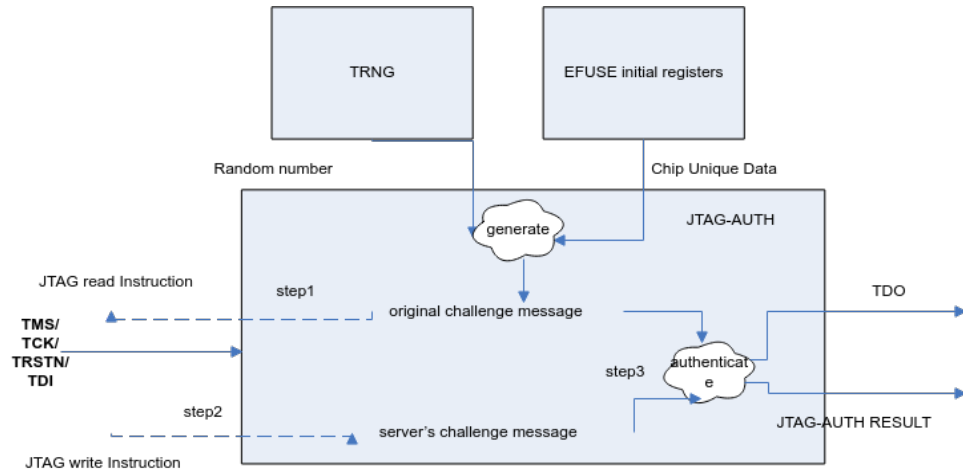


图5-2中step1对应图5-1的1、2；

图5-2中step2、step3对应图5-1的3。

须知

JTAG_AUTH鉴权模块和TRNG模块属于系统RSTN复位域。而非POR上电复位或芯片按键复位。

5.3 特性

JTAG-AUTH具有以下特性：

- JTAG鉴权功能受E-Fuse域段控制。
 - E-Fuse可配置鉴权等级，请参见“[鉴权等级](#)”。
 - E-Fuse可配置鉴权功能永久关闭。
 - HASH算法使用SHA256。
 - RSA算法使用4096bit和2048bit密钥长度。
 - 读指令功能如下：
 - 获取原始鉴权挑战信息（OCM）。
 - 获取JTAG-AUTH状态信息。



- 查询DIE_ID。
- 写指令功能如下：
 - 输入鉴权服务器反馈的JTAG鉴权挑战信息（CM）。
- 故障注入写指令功能如下：
 - 输入故障注入使能写指令，输入故障注入数据对SRAM进行ECC错误注入。
- 鉴权失败3次后，用户仍可以通过查询指令获取鉴权模块的状态信息。
- 鉴权失败3次后，JTAG-AUTH不接受鉴权测试指令。
- 用户可以通过芯片上电复位，重新发起鉴权流程。
- 在鉴权模式下，只要鉴权没通过，通过JTAG访问芯片相关能力就会被限制。
- 用户可以通过JTAG的bypass指令跳过芯片节点继续操作单板的下一个节点。（该芯片节点指的是JTAG_AUTH TAP控制器）

须知

该功能特性需JTAG调试接口保证。

- 鉴权等级。
- 鉴权模块当前状态。
- 支持SRAM ECC错误检查。
 - 支持1比特错误自动更正，但是不回写。
 - 支持1比特错误次数统计，统计达到最大值后保持。
 - 支持2比特错误记录最近一次出错地址。
 - 支持1比特和2比特故障注入（通过测试软件注入错误）。

5.4 功能描述

5.4.1 鉴权等级

5.4.1.1 鉴权等级真值表

表 5-2 鉴权等级真值表

JTAG_FORBID二进制值	功能说明	鉴权结果 {hiss,dft,djtag}disable
000	JTAG访问不需要鉴权，允许所有JTAG调试，包括ARM JTAG、DFT JTAG、DJTAG、HISS JTAG。	恒定为{0b0,0b0,0b0}



JTAG_FORBID二进制值	功能说明	鉴权结果 {hiss,dft,djtag}disable
001	JTAG访问不需要鉴权，仅允许使用DFT JTAG和HISS JTAG。	恒定为{0b0,0b0,0b1}
010	JTAG访问需要鉴权，鉴权成功允许所有JTAG调试，包括ARM JTAG、DFT JTAG、DJTAG、HISS JTAG。	初值为{0b1,0b1,0b1}，鉴权通过后变为{0b0,0b0,0b0}
011	JTAG访问需要鉴权，鉴权成功后仅允许DFT JTAG和HISS JTAG。	初值为{0b1,0b1,0b1}，鉴权通过后变为{0b0,0b0,0b1}
100/101/110/111	JTAG调试功能禁止，包括ARM JTAG、DFT JTAG、DJTAG、HISS JTAG。	恒定为{0b1,0b1,0b1}

5.4.1.2 鉴权结果行为描述

表 5-3 JTAG-TAP 与鉴权行为关系

鉴权等级	JTAG-AUTH
000/001	Die ID可查； 可获取OCM； 鉴权状态信息可查； 输入CM不会进行鉴权。
010/011鉴权失败小于3次	Die ID可查； 可获取OCM； 鉴权状态信息可查； 输入CM可进行鉴权。
010/011鉴权失败3次	Die ID可查； 可获取OCM； 鉴权状态信息可查； 输入CM不会进行鉴权。
010/011鉴权成功	Die ID可查； 可获取OCM； 鉴权状态信息可查； 输入CM不会进行鉴权。



鉴权等级	JTAG-AUTH
100/101/110/111	Die ID可查； 可获取OCM； 鉴权状态信息可查； 输入CM不会进行鉴权。

须知

1. 在鉴权的不同时刻Tool发起读原始挑战信息，获取的结果可能并非真实可用的OCM，且结果可能是不一样的。请参见“[原始挑战信息工作机制](#)、[不鉴权情况下的处理方式](#)”、“[禁止JTAG使用下的处理方式](#)”。

2. 每次芯片上电复位后，由于TRNG产生的随机数不一样，因此获取的真实可用的OCM也是不一样的。

说明鉴权失败 定义：芯片E-Fuse中的鉴权等级烧写为需要JTAG鉴权的情况下，TOOL输入CM，但是无法通过芯片的JTAG鉴权测试。

说明

鉴权失败定义：芯片E-Fuse中的鉴权等级烧写为需要JTAG鉴权的情况下，TOOL输入CM，但是无法通过芯片的JTAG鉴权测试。

5.4.2 鉴权填充协议

5.4.2.1 EMSA-PKCS1-v1_5 协议

按照PKCS#1v2.2，EMSA-PKCS-v1_5，签名值原文格式为：

0x00||0x01||PS||0x00||T；

其中，SHA256算法对应的T值为：0x30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 || H；（H：代表原始消息的HASH值）

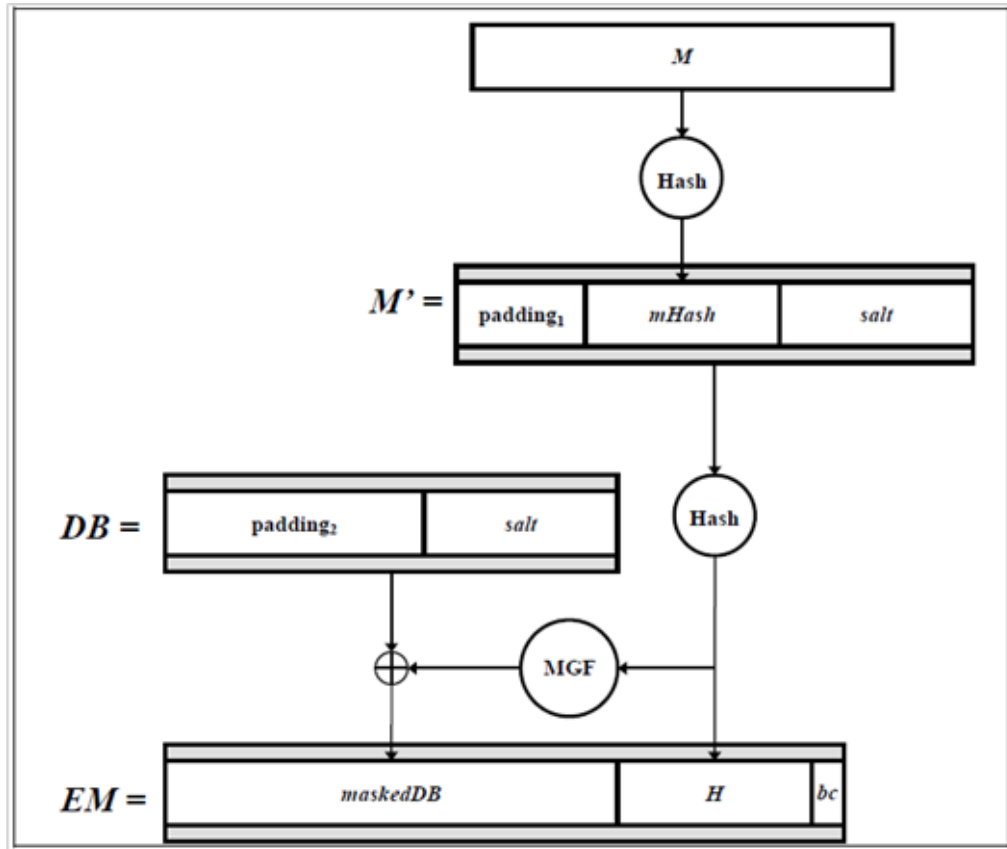
若签名值是256字节长，则PS值为202字节长的0xFF；

若签名值是512字节长，则PS值为458字节长的0xFF。

5.4.2.2 EMSA-PSS 协议

按照PKCS#1v2.2，EMSA-PSS，原始消息M经过编码协议处理后得到EM：

图 5-3 EMSA-PSS 协议



签名值原文格式即为图5-4中EM值：maskedDB||H||0xbc。

上图5-4中padding1代表0x 00 00 00 00 00 00 00 00，共计8字节长；DB = PS || 0x01 || salt；padding2代表：PS || 0x01；MGF为掩码函数，选择SHA-256算法；

PS由（emLen -sLen -hLen -2）字节的0值组成，其中emLen、sLen和hLen分别代表EM、salt和HASH值的字节长度；

salt为一串随机数，长度值可变，典型长度为hLen或0值；

若选择RSA-2048bit和SHA-256算法，则emLen = 256，hLen = 32，sLen范围：[0,222]；

若选择RSA-4096bit和SHA-256算法，则emLen = 512，hLen = 32，sLen范围：[0,478]；

EMSA-PSS协议签名原文验证流程：

- 步骤1** 计算原始消息M的SHA-256摘要值 $mHash = Hash(M)$ 。
- 步骤2** 检查 $emLen < hLen + sLen + 2$ 不满足，否则编码有错误，停止验证。
- 步骤3** 检查 EM中的最低字节为0xbc，否则编码有错误，停止验证。
- 步骤4** 检查EM中的最高（ $8emLen - emBits$ ）比特（对应为maskDB域段）为全0值，否则编码有错误，停止验证；（考虑emLen在鉴权应用场景取为2048bit， $8emLen - emBits$ 为0。）

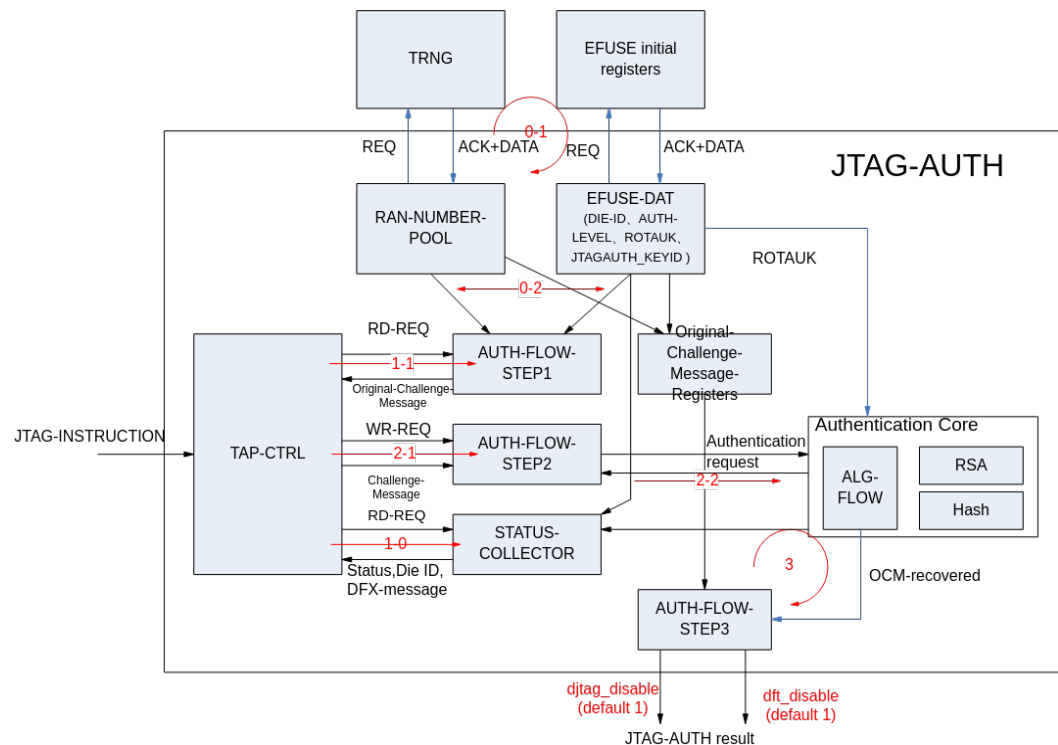


- 步骤5** 使用EM中的H值，计算 $dbMask = MGF(H, emLen - hLen - 1)$ ，其中emLen和hLen分别为256和32字节长。
- 步骤6** 使用EM中的maskedDB，计算得到 $DB = maskedDB \oplus dbMask$ 。
- 步骤7** 检查**步骤6**中的DB的PS是否为全0字节，紧邻的中间字节是否为0x01。
- 步骤8** 利用DB中的salt值，拼接得到 $M' = (0x) 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00 \parallel mHash \parallel salt$ 。
- 步骤9** 使用SHA-256算法，计算 $H' = Hash(M')$ 。
- 步骤10** 校验EM中的H值是否等于H'，若一致则校验通过，否则校验失败。

----结束

5.4.3 鉴权流程

图 5-4 JTAG-AUTH 鉴权流程详细框图



5.4.3.1 功能模块

JTAG-AUTH鉴权流程涉及各功能模块如下：

- TRNG：硬件随机数发生器。
- EFUSE initial registers：EFUSE存储体。
- RAN-NUMBER-POOL：用于存储JTAG-AUTH模块从TRNG中获取的随机数。
- EFUSE-DAT：用于存储从E-Fuse中获取的芯片静态参数，包含：Die ID、AUTH-LEVEL（鉴权等级）、ROTAUK（鉴权服务器公钥HASH值）、JTAGAUTH_KEYID（鉴权密钥号）。



- TAP-CTRL: JTAG-AUTH的TAP控制器，用于翻译JTAG接口指令。
- AUTH-FLOW-STEP1: JTAG-AUTH读指令数据返回控制，用于输入读挑战信息请求和输出读请求需要返回的数据。
- AUTH-FLOW-STEP2: JTAG-AUTH写指令数据处理，用于输入鉴权服务器返回的挑战信息；同时提供给Authentication-Core内部RAM进行故障注入功能。
- STATUS-COLLECTOR: JTAG-AUTH状态信息收集，查询JTAG-AUTH状态信息和Die ID信息。
- Original-Challenge-Message-Registers: 存储用于比对的原始挑战信息。
- Authentication-Core: 完成JTAG鉴权计算所需的算法模块。
- AUTH-FLOW-STEP3: JTAG-AUTH结果生成模块。

5.4.3.2 详细步骤

- 步骤1** 阶段0~1: TRNG、EFUSE执行上电初始化操作。
- 步骤2** 阶段0~2: 阶段0-1完成后，JTAG-AUTH从TRNG和EFUSE initial registers中获取所需随机数、芯片参数。
- 步骤3** 阶段1~0: 用户通过JTAG口输入查询JTAG状态信息的指令，判断是否需要进行鉴权操作，以及正常鉴权操作的相关条件：
1. TRNG随机数准备好。
 2. EFUSE数据准备好。
 3. 鉴权等级为需要鉴权。
 4. 鉴权尝试机会有剩余，并且鉴权失败或未启动鉴权。
- 是否全部满足。
- 步骤4** 阶段1~1: 用户通过JTAG口输入读取原始挑战信息的指令，JTAG-AUTH返回用户发起鉴权操作所需的原始挑战信息（即OCM）。与此同时，硬件内部保留一份OCM信息。OCMR每次在芯片上电后更新一次，且只能通过重新让芯片上电复位进行更新。
- 步骤5** 阶段2~1: 用户通过JTAG口输入从鉴权服务器获取的挑战信息（CM）。
- 步骤6** 阶段2~2: Authentication-Core对Tool输入挑战信息中包含的公钥做HASH运算得到的结果ROTAUK1, 将其与EFUSE中的公钥HASH值（ROTAUK）进行比对来校验，若ROTAUK通过校验，则对OCM做SHA256计算，得到OCM_HASH值；同时利用校验通过的公钥对Tool输入挑战信息中包含的签名值进行验证，还原出原始挑战信息，进行后续认证。若公钥校验失败，则认为本次鉴权失败。
- 步骤7** 阶段3: 鉴权结果生成模块根据OCM_HASH值和阶段2-2得到还原的原始挑战信息HASH值进行比较，若两者相等，则认为本次鉴权通过，否则，认为本次鉴权失败（鉴权尝试次数上限是3次，尝试次数耗尽，只能通过芯片上电复位重置）。鉴权成功后，修改鉴权结果。

----结束

须知

1. **步骤4**和阶段**步骤5**没有必然的关系。
2. 推荐：按上述流程顺序进行鉴权操作，若Tool不按照该流程顺序执行，即阶**步骤4**不成功或者没执行的前提下，直接执行阶段**步骤5**，则会判定为本次鉴权失败。



5.4.4 鉴权读指令

5.4.4.1 鉴权读指令编码

JTAG鉴权指令编码只针对JTAG中Debug Bus JTAG/TDRE寄存器访问数据格式中的32位ADDR地址进行。高8位的IR地址由JTAG统一编址。

须知

JTAG测试指令设计人员需确保DR串行移位进来后可转换为下述地址格式。

JTAG鉴权读指令分为三类：

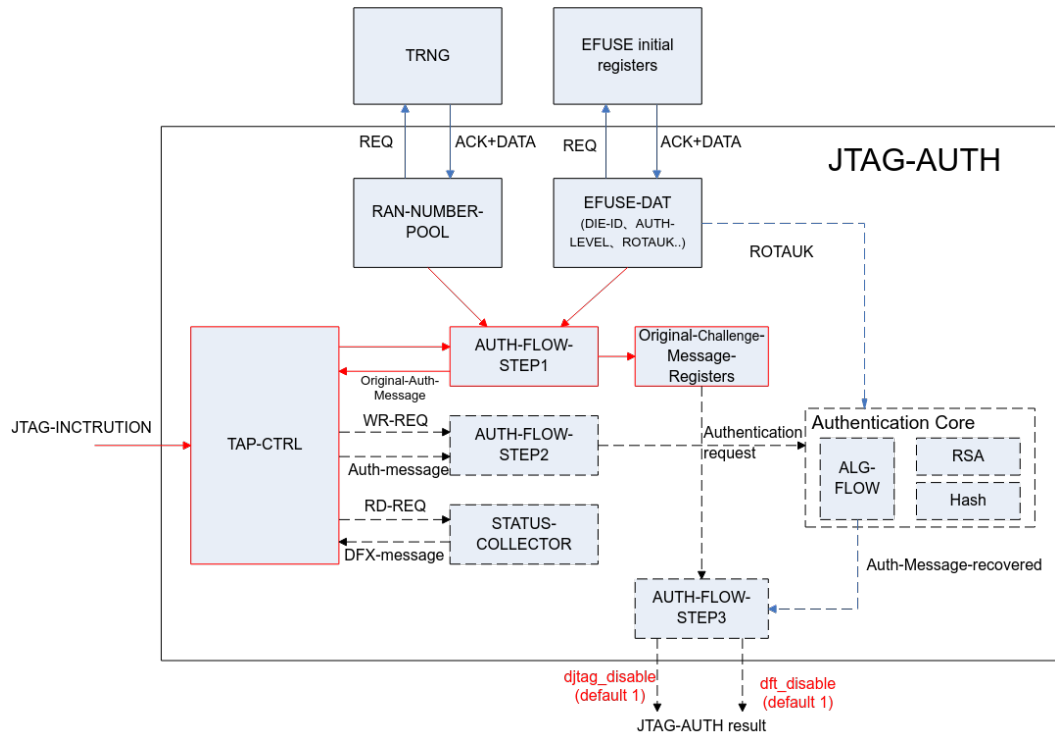
1. RDCMD_GET_OCM 获取原始挑战信息，地址为：0x0000_1000，地址按WORD对齐。例如：OCM长度为7个字，分别为WORD0~WORD6；其对应的地址为：0x0000_1000，0x0000_1004，0x0000_1008，0x0000_100C，0x0000_1010，0x0000_1014，0x0000_1018。
2. RDCMD_GET_DIE_ID 查询DIE ID信息，地址为：0x0000_2000，地址按WORD对齐。例如：DIE_ID长度为160比特即5个字，分别为WORD0~WORD4；其对应的地址为：0x0000_2000，0x0000_2004，0x0000_2008，0x0000_200C。
3. RDCMD_GET_STAT_INFO 查询状态信息，地址为0x0000_3000，地址按WORD对齐。例如：STAT_INFO长度为5个字，分别为WORD0~WORD4；其对应的地址为：0x0000_3000，0x0000_3004，0x0000_3008，0x0000_300C，0x0000_3010。

须知

读指令返回的数据格式请参见“[原始挑战信息](#)”和“[状态信息查询](#)”小节。

5.4.4.2 获取原始挑战信息

图 5-5 获取原始挑战信息示意图



上图5-5中红色实线部分代表用户发起读取原始挑战信息的指令。TRNG和eFuse数据准备阶段：芯片上电复位后，当TRNG和EFUSE初始化完成后，JTAG-AUTH读取随机数和芯片参数内部保存。该阶段内若用户发起读取OCM指令，得不到真实的OCM，返回的为异常情况（TRNG数据未准备好或EFUSE数据未准备好）的数据。

当数据准备好且鉴权等级为需要鉴权时，用户发起第一次申请原始挑战信息完成后 OCM生成完成，同时保留一份在JTAG-AUTH内部OCMR中。OCM信息输出完成标志 ocm_rdy被置位。（ocm_rdy标志请参见“[状态信息数据结构](#)”小节）

5.4.4.2.1 原始挑战信息工作机制

- 用户第一次申请原始挑战信息成功后，JTAG-AUTH返回OCM信息的同时会保留一份原始挑战信息在本地OCMR中，用于鉴权计算中进行比对。
- 用户第一次申请原始挑战信息成功后，OCM信息输出完成标志ocm_rdy被置位。
- JTAG-AUTH支持用户重复申请原始挑战信息，且用户再申请时获取的原始挑战信息与第一次成功申请的OCM信息相同，除非芯片重新上电复位。
- 原始挑战信息在芯片重新上电复位时清除，当芯片解复位后，用户发起申请原始挑战信息成功后得到OCM与复位之前的原始挑战信息不同。
- 若JTAG-AUTH无法获取足够的TRNG随机数，原始挑战信息无法生成，若此时用户申请原始挑战信息得不到真实OCM信息。
- 若JTAG-AUTH无法获取eFuse提供的参数，原始挑战信息无法生成，若此时用户申请原始挑战信息得不到真实OCM信息。



5.4.4.2.2 不鉴权情况下的处理方式

若鉴权等级是不鉴权，并且JTAG又发起获取原始挑战信息的指令，则处理方式如下：

- 数据返回值：OCM每个WORD都是0x0。
- 不产生原始挑战信息。
- ocm_rdy标志不会被置位。

5.4.4.2.3 禁止 JTAG 使用下的处理方式

若鉴权方式是禁止，并且JTAG又发起获取原始挑战信息的指令，则处理方式如下：

- 数据返回值：OCM每个WORD都是0xFFFF_FFFF。
- 不产生原始挑战信息。
- ocm_rdy标志不会被置位。

5.4.4.2.4 TRNG 或 EFUSE 未准备好的处理方式

- TRNG数据没准备好；
数据返回：
0x{55AA_55AA_55AA_55AA_55AA_55AA_55AA_55AA_FFFF_FFFF_FFFF_FFFF_F
FFF_FFFF}。
- E-Fuse数据没准备好；
数据返回：
0x{55AA_55AA_55AA_55AA_55AA_55AA_55AA_55AA_FFFF_FFFF_FFFF_FFFF_F
FFF_FFFF}。
- ocm_rdy标志不会被置位。

5.4.4.3 查询状态信息

图 5-6 查询状态信息示意图

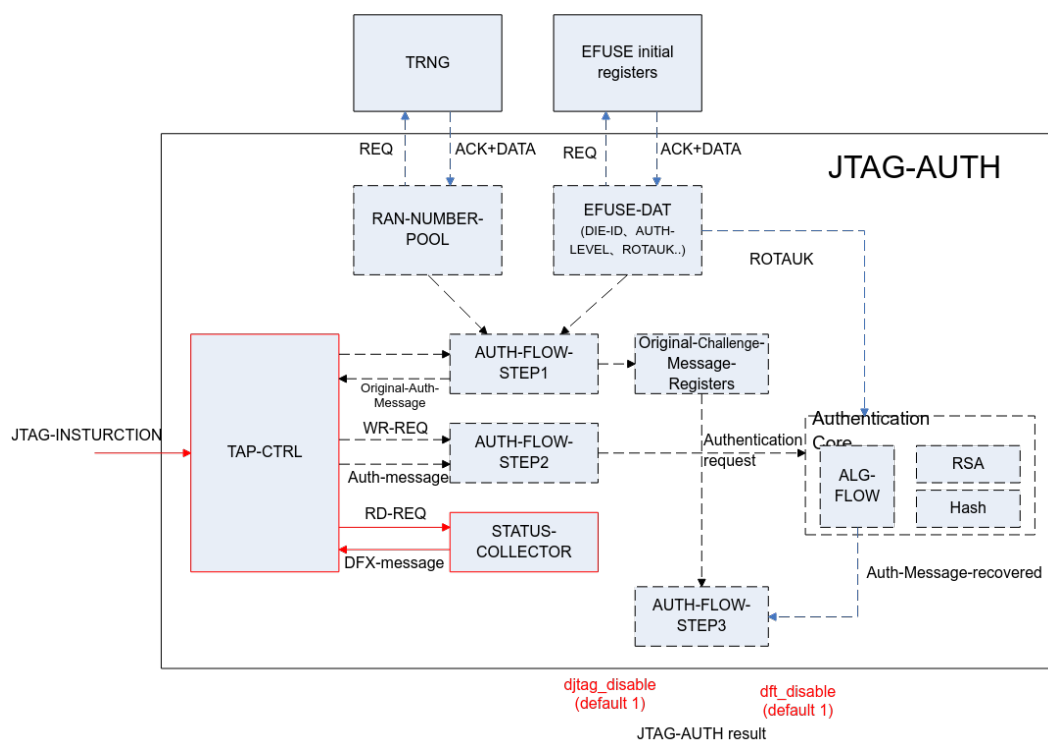


图5-6中，红色实线部分的描述。

状态信息包含如下内容:

- 鉴权等级（若eFuse没有初始化好，查询得到的值就是0x7）。
- TRNG随机数是否准备好。
- E-Fuse数据是否准备好。
- 鉴权结果及剩余鉴权尝试次数。
- 鉴权算法核异常信息记录。
- 是否已成功生成并存储原始挑战信息。

须知

1. 获取原始挑战信息和查询状态信息是两个不同的JTAG读指令。
2. 查询状态信息的指令不受JTAG鉴权等级约束。

5.4.4.4 查询 Die ID

Die ID指的是256比特BASIC DIE ID Info。

若eFuse数据没准备好，此时查询的Die ID的每个WORD都返回0x5A5A 0001。

须知

查询Die ID信息的指令不受JTAG鉴权等级约束。

5.4.5 鉴权写指令

图 5-7 鉴权写指令示意图

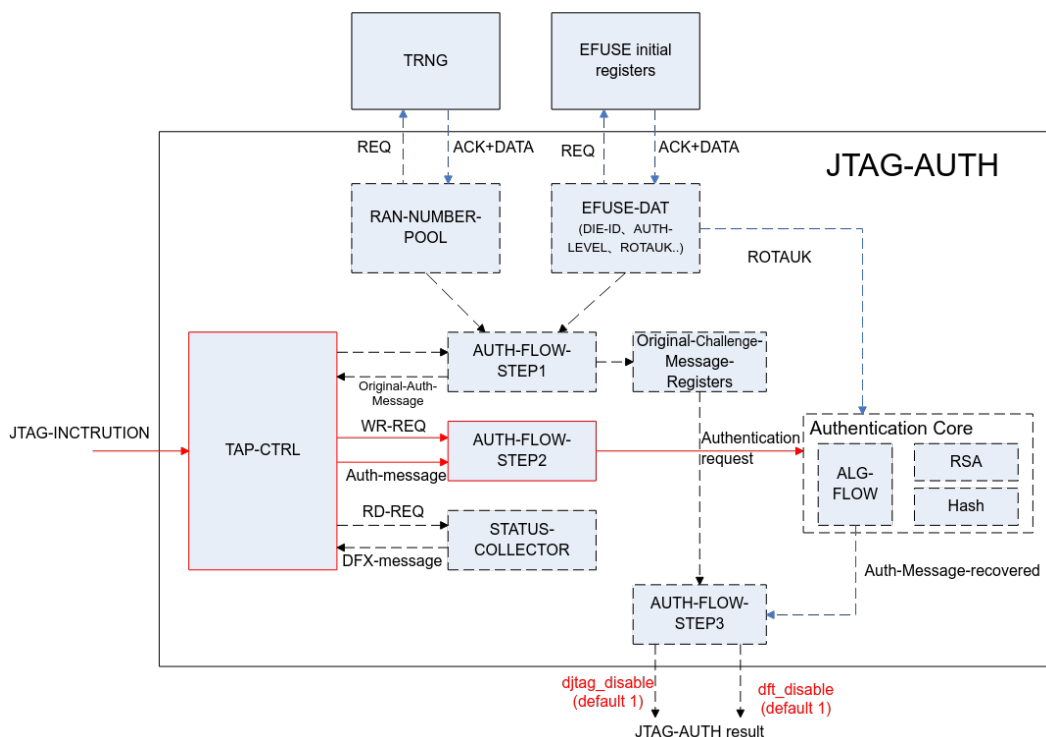


图5-7中，红色实线部分的描述。

- 用户通过鉴权写指令写入的数据是固定长度的鉴权服务器挑战信息（即CM）。数据格式请参见“[服务器输入挑战信息](#)”小节。
- 在鉴权等级为需要鉴权的前提下，JTAG-AUTH检测输入写指令的地址是否符合编址要求，并且当写入的数据达到挑战信息长度后，就会启动鉴权计算。
- 用户输入的CM挑战信息是按字编址的，按顺序每个地址对应一个字排下来。例如：CM挑战信息为WORD0~WORD191，则对应地址为0~191。用户需按地址顺序写入CM挑战信息的每个字。硬件按地址顺序记录写入的数据，对于同一个地址，硬件只记录用户第一次写入的数据，不支持对该地址重复写入数据，重复写入的数据会被硬件忽略。

5.4.5.1 鉴权写指令编码

JTAG鉴权写指令编址如下:

WRCMD_GET_OAM 输入服务器挑战信息（即CM），地址为：0x0000_0000，地址按WORD对齐。例如：CM长度为192个字，分别为WORD0~WORD191；其对应的地址为：0x0000 0000，0x0000 0001，…，0x0000 00BE，0x0000 00BF。



5.4.5.2 输入服务器挑战信息管理方法

CM更新方式必须同时满足如下四点，否则无法被更新：

- 鉴权等级：需要鉴权，对应值为0x2或者0x3。
- 鉴权尝试次数未耗尽。
- 鉴权计算未开始，且之前的鉴权计算未通过。
- 已经成功获取原始挑战信息OCM（即ocm_rdy标志被置位）。
- 启动鉴权计算后，JTAG-AUTH会锁定输入的服务器挑战信息。

5.4.5.3 输入服务器挑战信息异常情况

表 5-4 输入服务器挑战信息异常情况表

CM异常情况	处理流程
鉴权等级：不需要鉴权，但是输入的鉴权挑战信息；	不接收，不启动鉴权计算流程，维持鉴权结果和鉴权次数默认值。
鉴权等级：禁止使用JTAG，但是输入的鉴权挑战信息；	不接收，不启动鉴权计算流程，维持鉴权结果和鉴权次数默认值。
TRNG数据没准备好；	不接收，不启动鉴权计算流程，维持鉴权结果和鉴权次数。
E-Fuse数据没准备好；	不接收，不启动鉴权计算流程，维持鉴权结果和鉴权次数。
鉴权等级：需要鉴权，且鉴权通过情况下发起鉴权操作。	不接收，不启动鉴权计算流程，维持鉴权结果和鉴权次数。
鉴权等级：需要鉴权，且鉴权次数耗尽。	不接收，不启动鉴权计算流程，维持鉴权结果和鉴权次数。
鉴权等级：需要鉴权，用户第一次发起读OCM指令，但未完全读走OCM信息，发起鉴权操作。	不接收，不检查该类错误，tool操作者必须确保流程正确。
鉴权等级：需要鉴权，当前CM信息未完成输入，插入新的CM信息。	接收，不检查该类错误，tool操作者必须确保流程正确。
鉴权等级：需要鉴权，正在进行； 鉴权计算：算法核没有完成计算流程又输入新的CM信息。	不接收，不检查该类错误，tool操作者必须确保流程正确。

说明

接收：表示刷新CM缓存memory；不接收：表示不刷新CM缓存memory。

5.4.6 故障注入写指令

JTAG故障注入写指令不受JTAG鉴权等级限制，分为以下两类：

- 故障注入使能写指令：ERR_INJECT_EN_ADDR，地址编址为：0×00004000，用户写入32比特固定数据0×1ACCE551，使能ECC故障注入；用户再次写入其他值，禁止ECC故障注入。
- 故障注入数据写指令：ERR_INJECT_DAT_ADDR，地址编址为：0×00004004，用户写入32比特数据。使能无效时，写入故障注入数据也不会产生ECC错误。当故障注入使能打开，用户可通过JTAG接口对JTAG_AUTH模块内部RAM进行ECC错误注入。

5.4.7 鉴权计算

图 5-8 鉴权计算示意图

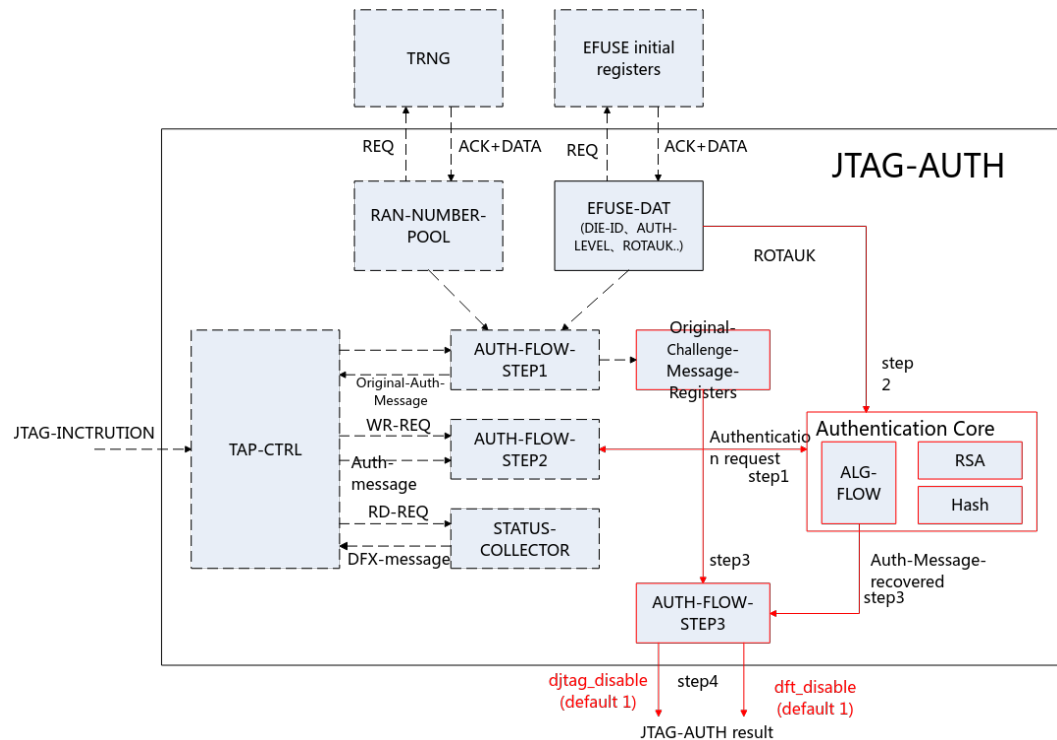


图5-8中，红色实线部分的描述。

5.4.7.1 鉴权计算流程

- 步骤1** 正确获取到输入鉴权挑战信息CM后，启动鉴权计算，进入**步骤2**。
- 步骤2** 对鉴权挑战信息中的公钥值进行HASH计算，检查该HASH结果是否跟E-Fuse中存储的公钥HASH值ROTAUK相同，若相同，则进入**步骤3**，否则判定为失败。
- 步骤3** 使用经过**步骤2**校验通过的公钥值对鉴权挑战信息中的签名值进行认证计算，还原出鉴权挑战信息中的签名值对应的签名明文；完成后进入**步骤4**。
- 步骤4** 校验签名明文的填充值，若填充值不满足EMSA-PKCS1-v1_5要求则判定为鉴权失败，并结束；否则进入**步骤5**。



步骤5 计算OCM的HASH值OCM_HASH1；完成后进入**步骤6**。

步骤6 使用签名明文中的HASH值与OCM_HASH1进行比较，若相同，则判定为鉴权通过；否则，判定为鉴权失败。

----结束

说明

所有步骤涉及的算法如下：

- HASH算法是SHA256。
- 非对称算法是计算。
- 鉴权签名的认证算法遵循PKCS1v2的EMSA-PKCS1-v1_5。

须知

步骤1的成功必须满足如下4个条件：

1. 鉴权等级：需要鉴权。
2. 鉴权尝试次数未耗尽。
3. 鉴权计算未开始，且之前的鉴权计算未通过。
4. 已经成功获取原始挑战信息（即ocm_rdy标志被置位）。

5.4.7.2 不鉴权情况下的处理方式

用户在此情况下输入鉴权挑战信息，JTAG-AUTH不会接收输入的鉴权挑战信息，不会启动鉴权计算流程，也不会对状态信息产生影响。

5.4.7.3 禁止 JTAG 使用下的处理方式

用户在此情况下输入鉴权挑战信息，JTAG-AUTH不会接收输入的鉴权挑战信息，不会启动鉴权计算流程，也不会对状态信息产生影响。

5.5 编程接口

5.5.1 Debug_bus_inf 编址

JTAG_AUTH鉴权模块使用了四组Debug_Bus_Interface接口，JTAG测试时通过给JTAG_AUTH内部TAP控制器写入不同的指令值使能哪个DBI接口。具体对应关系如**表5-5**所示。

表 5-5 四组 Debug_Bus_Interface 接口

序号	接口名	功能描述	对应的JTAG指令CODE	备注
0	U_DEBUG_BUS_INFO ；	用户写CM，写故障注入指令接口；	0x20	-



序号	接口名	功能描述	对应的JTAG指令CODE	备注
1	U_DEBUG_BUS_INF1 ；	用户读OCM指令接口；	0x21	-
2	U_DEBUG_BUS_INF2 ；	用户读鉴权状态信息和DIE ID指令接口；	0x22	-
3	U_DEBUG_BUS_INF3 ；	用户写边界扫描使能指令接口。	0x23	-

JTAG_AUTH模块内部TAP控制器支持的JTAG指令，对应到JTAG FS文档中如表5-6所示。

表 5-6 JTAG 指令

指令名	Code	目标寄存器	JTAG寄存器
EXTEST	00000000	-	标准BSD指令。
BYPASS	11111111	-	标准BSD指令。
SAMPLE	00000001	-	标准BSD指令。
PRELOAD	00000001	-	标准BSD指令。
CLAMP	00000010	-	标准BSD指令。
HIGHZ	00000011	-	标准BSD指令。
IDCODE	00000100	-	标准BSD指令。
DBG_CHAIN0	00100000	用户写CM、写故障注入指令，JTAG_AUTH模块内部寄存器。	DJTAG Debug链0。
DBG_CHAIN1	00100001	用户读OCM指令，JTAG_AUTH模块内部寄存器。	DJTAG Debug链1。
DBG_CHAIN2	00100010	用户读状态信息和DIE ID指令，JTAG_AUTH模块内部寄存器。	DJTAG Debug链2。
DBG_CHAIN3	00100011	单独边界扫描指令。	DJTAG Debug链3。
.....		不支持。	DJTAG Debug链N。
DBG_CHAIN63	01011111	不支持。	DJTAG Debug链63。



5.5.2 用户鉴权指令说明

JTAG_AUTH依据鉴权流程需要，对JTAG测试接口输入的TDRE寄存器的ADDR和DATA进行重新定义。

表 5-7 Debug Bus JTAG/TDRE 寄存器访问数据格式

Bits	Name	Description
[65]	CS	操作有效标识位，高有效。
[64]	R/W	寄存器读写控制信号： 0：读操作； 1：写操作。
[63]	PROT_S	安全访问标识。 0：非安全访问； 1：安全访问。
[62:32]	ADDR（Err）	DJTAG访问地址。 读操作返回数据地址最低位表征JTAG和CPU读冲突标识，高有效。
[31:0]	DATA	DJTAG访问数据。

鉴权模块与用户之间的数据交互访问是通过以下3类指令进行，分别为鉴权读指令、鉴权写指令和故障注入写指令。其中鉴权读指令又细分为获取OCM、查询状态信息和查询DIE ID。具体指令的地址编址如表5-8所示。

表 5-8 鉴权指令编址说明

类型	名称	功能描述	地址编址ADDR	数据位宽 DATA	数据总长度 （单位：字）	要求
读指令	鉴权获取OCM	用户读取原始挑战信息	基地址：0x0000_1000，偏移地址：按字递增对齐	32bit	7	-
	查询状态信息	用户查询鉴权信息	基地址：0x0000_3000，偏移地址：按字递增对齐	32bit	5	-
	查询DIE ID	用户读取DIE ID	基地址：0x0000_2000，偏移地址：按字递增对齐	32bit	8	-



类型	名称	功能描述	地址编址ADDR	数据位宽 DATA	数据总长度 (单位: 字)	要求
写指令	鉴权写指令	输入服务器挑战信息	基地址: 0x0000_0000, 偏移地址: 自增加1	32bit	192	保序写入
	故障注入写指令	用户使能ECC故障注入	地址: 0x0000_4000	32bit	1	写0x1ACC_E551 开启故障注入使能
		用户发起ECC故障注入	地址: 0x0000_4004	32bit	1	先使能, 后写数据

须知

故障注入首先写使能ECC故障注入，然后写入故障注入数据。输入服务器挑战信息严格要求按地址顺序写入。

5.5.3 原始挑战信息

5.5.3.1 原始挑战信息数据结构

表 5-9 原始挑战信息数据结构

序号	域段名称（OCM）	用户输入读取地址
WORD0	WORD0: Nonce_3;	0x00001000;
WORD1	WORD1: Nonce_2;	0x00001004;
WORD2	WORD2: Nonce_1;	0x00001008;
WORD3	WORD3: Nonce_0;	0x0000100C;
WORD4	WORD4: DeviceID_H;	0x00001010;
WORD5	WORD5: DeviceID_L;	0x00001014;
WORD6	WORD6: JTAGAUTH_KEYID;	0xh00001018。



0x{Nonce_3, Nonce_2, Nonce_1, Nonce_0, DeviceID_H, DeviceID_L, JTAGAUTH_KEYID}构成原始挑战信息（OCM）。鉴权服务器根据0x{JTAGAUTH_KEYID}匹配鉴权密钥对。服务器签名原文需要使用0x{Nonce_3, Nonce_2, Nonce_1, Nonce_0, DeviceID_H, DeviceID_L, JTAGAUTH_KEYID}。

须知

若用户按地址递增顺序读取，则鉴权模块按照WORD0~WORD6顺序返回数据给用户。推荐按照此方式读取OCM值。

5.5.3.2 原始挑战信息域段解析

表 5-10 原始挑战信息域段解析

域名称	复位值	功能描述
JTAGAUTH_KEYID	0x0	JTAG鉴权密钥ID；通常一个版本单板用一个KEY。
DeviceID	0x0	对应E-FUSE中的DeviceID域段，该域段经过如下填充方式构成了64bit输出： {0x0, Die ID}。 其中DeviceID_H对应{7'b0,DeviceID[56:32]}。 其中DeviceID_L对应{DeviceID[31:0]}。
DeviceID_H	0x0	DeviceID的高32bit。
DeviceID_L	0x0	DeviceID的低32bit。
Nonce_0、Nonce_1、Nonce_2、Nonce_3	0x0	{Nonce_3, Nonce_2, Nonce_1, Nonce_0}构成128bit随机数。

5.5.3.3 初始化异常情况及反馈值

满足如下两个条件之一，用户发起原始挑战信息读指令都返回原始挑战信息异常值。

- E-Fuse数据未准备好。
- 随机数未准备好。

表 5-11 初始化异常情况及反馈值

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
WORD0: 0x55AA_55AA																														



3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
WORD1: 0x55AA_55AA																														
WORD2: 0x55AA_55AA																														
WORD3: 0x55AA_55AA																														
WORD4: 0xFFFF_FFFF																														
WORD5: 0xFFFF_FFFF																														
WORD6: 0xFFFF_FFFF																														

须知

若用户按地址递增顺序读取，则鉴权模块按照W0~W6的数据顺序返回给用户。推荐按此方式读取状态信息。

5.5.3.4 不鉴权情况下的反馈值

返回7个WORD均为0x0。

5.5.3.5 禁止 JTAG 使用下的反馈值

返回7个WORD均为0xFFFFFFFF。

5.5.3.6 原始挑战信息 HASH 计算方式

OCM输入SHA256算法的明文数据遵循以下格式：
0x{ WORD0,WORD1,WORD2,WORD3,WORD4,WORD5,WORD6}；其中数据为字内小端模式。鉴权模块根据该格式的OCM值计算OCM的HASH值。

5.5.4 状态信息查询

- 鉴权等级。
- 随机数是否准备好。
- E-Fuse数据是否准备好。
- 鉴权结果及剩余鉴权尝试次数。
- 鉴权算法核异常信息记录。
- 是否已成功生成原始挑战信息（即ocm_rdy标志被置位）。



5.5.4.1 状态信息数据结构

图 5-9 状态查询数据结构

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
W0	rsa_init	alg_flow_shb	redc_shb	r2modn_shb	auth_flow_status							alg_abnormal_record									auth_result				apn_rdy	server_msg_rdy	efuse_rdy	ran_rdy	auth_level																							
W1	ecc_1bit_err_addr																								ecc_1bit_err_cnt																											
W2	ecc_2bit_err_addr																								ecc_2bit_err_cnt																											
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
W3	reserved					normal_coded	dfx_nc_flow_st					dfx_para_c_st										dfx_redc_st																														
W4	reserved			dfx_para_c_i_value														dfx_exp_cnt																																		

表 5-12 状态查询数据对应地址

序列	域段名称（STATE_INFO）	用户输入地址
W0	鉴权状态信息	0x0000_3000
W1	ECC 1比特错误信息	0x0000_3004
W2	ECC 2比特错误信息	0x0000_3008
W3	DFX信息	0x0000_300C
W4	DFX信息	0x0000_3010

须知

若用户按地址递增顺序读取，则鉴权模块按照W0~W4的数据顺序返回给用户。推荐按此方式读取状态信息。



5.5.4.2 状态信息数据结构域段解析

表 5-13 状态信息数据结构域段解析

域名称	复位值	功能描述
auth_level	0x7	鉴权等级，来自eFuse静态数据。
ran_rdy	0b0	是否已获取随机数。 0x0：尚未获取足够的TRNG随机数； 0x1：已获取足够的TRNG随机数。
efuse_rdy	0b0	是否已获取E-Fuse数据。 0x0：E-Fuse数据尚未完成获取； 0x1：E-Fuse数据已获取。
server_msg_rdy	0b0	需要进行鉴权计算，且服务器输入挑战信息准备好标志。 0x0：服务器输入挑战信息未准备好； 0x1：服务器输入挑战信息已准备好。
ocm_rdy	0b0	硬件是否已成功生成原始挑战信息。 0x0：未成功生成原始挑战信息； 0x1：已成功生成原始挑战信息。 注意：成功生成OCM的充要条件： TRNG随机数准备好； EFUSE随机数准备好。 用户发起第一次OCM读取指令，通过JTAG接口按地址递增顺序完整读取OCM信息。
auth_result	0x3	0x0：鉴权失败，无鉴权尝试机会； 0x1：鉴权失败，剩余1次鉴权尝试机会； 0x2：鉴权失败，剩余2次鉴权尝试机会； 0x3：未启动鉴权，剩余3次鉴权尝试机会；（初始值） 0x8：鉴权通过，无鉴权尝试机会； 0x9：鉴权通过，剩余1次鉴权尝试机会； 0xA：鉴权通过，剩余2次鉴权尝试机会； 其他：保留。



域名称	复位值	功能描述
alg_abnormal_record	0x0	导致鉴权失败的原因，各个bit为0x1，表征错误触发，0x0表征错误没有触发。 只记录鉴权过程中的失败原因，随芯片上电复位清除。 bit[9:5]：保留。 bit[4]：RSA数据检查错误；以下三种情况任意一项或多项发生则报错： N<M；E为0；N[LSB]不为0b1或N[MSB:MSB-1]同时为0b00。 bit[3]：鉴权签名值校验填充规则校验成功，OCM的HASH值匹配不通过。 bit[2]：ROTAUK匹配通过后，鉴权签名值填充规则校验失败； bit[1]：ROTAUK匹配不通过； bit[0]：鉴权计算出现ECC 2bit错误。
alg_flow_status	0x1	鉴权计算状态机状态，共计7bit。 0x1：空闲； 0x2：获取输入的服务器挑战信息OAM； 0x4：鉴权公钥HASH值与ROTAUK比对阶段； 0x8：OCM进行HASH计算阶段； 0x10：RSA解认证阶段； 0x20：鉴权签名值填充校验阶段（遵循ESMA_PKCS1-v1_5协议）； 0x40：鉴权计算结果生成阶段。
r2modn_shb	0b0	r2modn计算超时标志，当计数值达到智能心跳配置值，置0b1。
redc_shb	0b0	redc计算超时标志，当计数值达到智能心跳配置值，置0b1。
alg_flow_shb	0b0	鉴权计算超时标志，当计数值达到智能心跳配置值，置0b1。
rsa_ini_st	0b0	RSA算法内部Memory初始化完成标志，完成后置0b1。
-		
ecc_1bit_err_cnt	0x0	内部memory ECC 1bit错误统计，计满保持。当2个及以上的RAM块同时发生1bit错误时，计数器只自增加1；外部通过JTAG口读清零。
ecc_1bit_err_addr	0x0	最近一次发生ECC 1bit错误的内部SRAM地址，默认值为0x0。



域名称	复位值	功能描述
-		
ecc_2bit_err_cnt	0x0	内部memory ECC 2bit错误统计，计满保持。当2个及以上的RAM块同时发生1bit错误时，计数器只自增加1；外部通过JTAG口读清零。
ecc_2bit_err_addr	0x0	最近一次发生ECC 2bit错误的内部SRAM地址，默认值为0x0。
-		
dfx_redc_st	0x1	RSA算法REDC计算控制状态机状态位。 0x1: MODEX_IDLE; 0x2: MODEX_N_INV_MOD; 0x4: MODEX_GET_FIRST_E; 0x8: MODEX_WAIT_FOR_MODEX; 0x10: MODEX_GET_M1; 0x20: MODEX_GET_FIRST_R; 0x40: MODEX_GET_MIDDLE_R; 0x80: MODEX_GET_FINAL_R; 0x100: MODEX_WAIT_FOR_END; 0x200: MODEX_GEN_COMPLETE。
dfx_para_c_st	0x1	RSA算法r2modn计算状态机状态位。 0x1: R2MODN_IDLE; 0x2: R2MODN_INITIAL; 0x4: R2MODN_RR_MINUS_N_SHIFT; 0x8: R2MODN_GET_RR_RANK; 0x10: R2MODN_N_SHIFT; 0x20: R2MODN_HIGH_ZERO_FIND; 0x40: R2MODN_HIGH_ZERO_KILL; 0x80: R2MODN_MINUS; 0x100: R2MODN_JUDGE; 0x200: R2MODN_JUDGE_AGAIN; 0x400: R2MODN_OVER。
dfx_nc_flow_st	0x1	RSA算法normal_codec控制状态机状态位。 0x1: NC_IDLE; 0x2: NC_MNE_CHECK; 0x4: NC_GET_C; 0x8: NC_GET_REDC。



域名称	复位值	功能描述
normal_codec	0b0	RSA算法normal_codec计算超时标志，当计数值达到智能心跳配置值，置0b1。
-		
dfx_exp_cnt	0x0	RSA算法REDC计算内部计数器exp_cnt值。
dfx_para_c_i_value	0x0	RSA算法r2modn计算内部控制信号i_value值。

5.5.4.3 具体情况下的状态查询值说明

表 5-14 状态信息 WORD0 在不同阶段访问的反馈值

状态信息查询时刻	返回的状态信息
TRNG数据没准备好；	鉴权等级——0x7； 随机数是否准备好——否； E-Fuse数据是否准备好——反映实际值； 鉴权结果——未启动； 剩余鉴权尝试次数——3； 鉴权算法核异常信息记录——反映实际值； 是否已成功生成并存储原始挑战信息——反映实际值； 鉴权计算状态机 状态——反映实际值。
E-Fuse数据没准备好；	鉴权等级——0x7； 随机数是否准备好——反映实际值； E-Fuse数据是否准备好——否； 鉴权结果——未启动； 剩余鉴权尝试次数——3； 鉴权算法核异常信息记录——反映实际值； 是否已成功生成并存储原始挑战信息——反映实际值； 鉴权计算状态机 状态——反映实际值。
若鉴权等级是不鉴权，查询状态信息（不鉴权查询依赖系统JTAG拓扑设计，JTAG下链读出全0）。	鉴权等级——反映实际值； 随机数是否准备好——反映实际值； E-Fuse数据是否准备好——是； 鉴权结果——未启动； 剩余鉴权尝试次数——3； 鉴权算法核异常信息记录——反映实际值； 是否已成功生成并存储原始挑战信息——反映实际值； 鉴权计算状态机 状态——反映实际值。



状态信息查询时刻	返回的状态信息
若鉴权等级是禁止 JTAG，查询状态信息；	鉴权等级——反映实际值； 随机数是否准备好——反映实际值； E-Fuse数据是否准备好——是； 鉴权结果——未启动； 剩余鉴权尝试次数——3； 鉴权算法核异常信息记录——反映实际值； 是否已成功生成并存储原始挑战信息——反映实际值； 鉴权计算状态机 状态——反映实际值。

5.5.5 DIE ID 查询

Die ID共计8个字长。若E-Fuse数据没准备好，此时查询的Die ID的每个WORD都返回0x5a5a_0001。复位默认值{0x5a5a_0001, 0x5a5a_0001, 0x5a5a_0001, 0x5a5a_0001, 0x5a5a_0001, 0x5a5a_0001, 0x5a5a_0001, 0x5a5a_0001}。

表 5-15 DIE ID 信息查询

序号	域段名称（DIE ID）	用户输入地址
W0	DIE_ID[255:224]	0x0000_2000
W1	DIE_ID[223:192]	0x0000_2004
W2	DIE_ID[191:160]	0x0000_2008
W3	DIE_ID[159:128]	0x0000_200C
W4	DIE_ID[127:96]	0x0000_2010
W5	DIE_ID[95:64]	0x0000_2014
W6	DIE_ID[63:32]	0x0000_201C
W7	DIE_ID[31:0]	0x0000_2020

须知

若用户按地址递增顺序读取，则鉴权模块按照W0~W7的数据顺序返回给用户。推荐按此方式读取状态信息。

5.5.6 服务器输入挑战信息

服务器输入挑战信息数据结构：0x{挑战信息签名值，公钥模值，公钥指数}，共12288bit数据。具体划分如表5-16所示。



表 5-16 服务器输入鉴权挑战信息

序列	域段名称（CM）	用户输入地址
WORD0	WORD0：服务器输入挑战信息签名值WORD0；	0×0000_0000
WORD1~WORD126	WORD1~WORD126：服务器输入挑战信息签名值WORD1~WORD126；	0×0000_0001~0×0000_007E
WORD127	WORD127：服务器输入挑战信息签名值WORD127	0×0000_007F
WORD128	WORD128：服务器签名公钥模值WORD0；	0×0000_0080
WORD129~254	WORD129~254：服务器签名公钥模值WORD1~WORD126；	0×0000_0081~0×0000_00FE
WORD255	WORD255：服务器签名公钥模值WORD127；	30×0000_00FF
WORD256	WORD256：服务器签名公钥指数WORD0；	0×0000_0100
WORD257~382	WORD257~382：服务器签名公钥指数WORD1~WORD126；	0×0000_0101~0×0000_017E
WORD383	WORD383：服务器签名公钥指数WORD127；	0×0000_017F

须知

用户输入的CM挑战信息是按字编址的，按顺序每个地址对应一个字排下来。例如：CM挑战信息为WORD0~WORD383，则对应地址为0~383。用户需按地址顺序写入CM挑战信息的每个字。硬件按地址顺序记录写入的数据，对于同一个地址，硬件只记录用户第一次写入的数据，不支持对该地址重复写入数据，重复写入的数据会被硬件忽略。

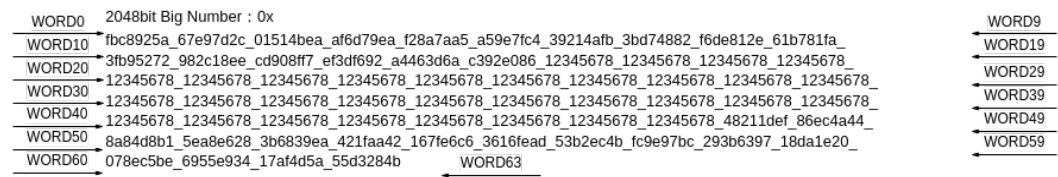
5.5.6.1 服务器输入挑战信息签名值计算约定

5.5.6.1.1 签名值的存储和表示法

以RSA2048算法进行鉴权（RSA4096可参考RSA2048方式）。JTAG-AUTH所需的2048比特签名值，模块内部按照数据大端方式进行存储，大小端针对每个WORD而言，而非整个2048比特大数。具体存储和表示法如下图5-10所示。



图 5-10 JTAG-AUTH 所需的 2048 比特签名值 WORD 对应图



2048bit大数按照32个WORD来存储；大小端针对每个WORD而言，而非整个4096bit大数。对于大数而言，WORD0是最高有效bit所在WORD，WORD63是最低有效bit所在WORD。

图5-10中WORD0~63对应表5-16中的WORD0~WORD63。

须知

公钥数据的存储和表示方式参考签名值的做法。

5.5.6.1.2 公钥数据的存储和表示方法

公钥数据的存储和表示方法与签名值数据格式一致。

5.5.6.1.3 公钥信息 HASH 计算方式

公钥模值和指数值 $0x\{N,E\}$ 输入SHA256算法的明文数据遵循格式为：
 $0x\{ \text{WORD64}, \text{WORD65}, \dots, \text{WORD127}, \text{WORD128}, \dots, \text{WORD190}, \text{WORD191} \}$ ；数据为字内小端模式。鉴权模块按照该格式的公钥信息计算公钥HASH值。

5.6 JTAG_AUTH 内部交互数据结构

5.6.1 eFuse 接口数据结构

eFuse接口传输过来的数据格式如下所示： $\{0x0, \text{JTAG_FORBID}, \text{JTAGKEY_ID}, \text{ROTAUK}, \text{DIE_ID}\}$ ，共计 $18 \times 32\text{bit}$ 等于576bits。实际E-Fuse数据547bit，其中高位补0。E-FUSE_SHIFTER组件按32比特为一组接收E-FUSE数据，然后按照串行方式输出数据给JTAG_AUTH。例如：EFUSE_SHIFTER组件要输出的数据为 $\{0x57D3\}$ ，则输出按串行方式各比特先后顺序为：LSB：1 -> 1 -> 0 -> 0 -> -> 1 -> 0 -> 1 -> 0：MSB。

鉴权模块涉及的eFUSE域段JTAG_FORBID，JTAGKEY_ID，ROTAUK，DIE_ID数据，在eFUSE初次烧写按小端模式写入。

5.6.2 TRNG 接口数据结构

TRNG接口传输的数据总共两个64bit，拼接得到128bitNONCE数据。数据格式按字为单位： $\{\text{NONCE3} \sim \text{NONCE0}\} = 0x00000000_11111111_22222222_33333333$ 。



5.7 应用配置

JTAG-AUTH根据eFuse的鉴权等级，选择相应的处理方式。

5.7.1 不使用 JTAG 鉴权

eFuse中的JTAG鉴权级别对应为0x0和0x1。

推荐应用场景：

芯片ATE测试阶段、实验室调试、产品装备阶段、暂时无法部署鉴权服务器的产品。

5.7.2 使用 JTAG 鉴权

根据eFuse中的JTAG鉴权级别选择需要的JTAG访问限制方式，请参见“[鉴权等级](#)”。

推荐应用场景：

芯片商用后，JTAG访问需要受限的场景，这种场景需要产品部署鉴权服务器。

5.7.3 禁止使用 JTAG

通过eFuse中的JTAG鉴权级别，完全禁止通过JTAG访问及调试芯片。

5.7.4 边界扫描使能

芯片发给客户后，需要鉴权才能使能DFT TAP的边界扫描功能。为了满足用户在不需要鉴权的也能使用DFT TAP的边界扫描功能，同时保证芯片的安全性，JTAG_AUTH新增一条鉴权扫描使能指令。边界扫描使能指令使用详细流程如下：

步骤1 输入鉴权边界扫描指令0x23，进行边界扫描使能认证。

步骤2 输入边界扫描的地址0x0000_4000。

步骤3 输入边界扫描数据0x5ACC_1080，指令正确即可打开鉴权边界扫描功能。

----结束

5.8 初始化

5.8.1 周边初始化

步骤1 芯片按照上电流程，完成eFuse初始化。

步骤2 复位值逻辑在检测到E-Fuse初始化完成后，解复位TRNG，TRNG工作在晶振时钟下，并根据默认初始化选项完成初始化。

----结束

5.8.2 重置 JTAG 鉴权

通过芯片上电复位流程完成鉴权等级重置。

通过重置JTAG鉴权达到以下目的：



- 重置JTAG鉴权次数。
- 排除因为TRNG初始化失败，导致JTAG鉴权因为无法获取随机数，导致流程无法完成的问题。

5.8.3 DFT 测试注意事项

JTAG_AUTH鉴权模块由于DFT测试时需要部分逻辑不插链，部分逻辑插链的特殊性要求，以及输出的鉴权结果（dft_disable和djtag_disable信号）若因DFT测试不当会影响整个芯片JTAG调试接口，因此要求芯片进入DFT模式操作严格遵守以下步骤进行：

- 步骤1** 芯片解复位后，若鉴权等级为需要鉴权，则首先进行鉴权流程，“[鉴权流程](#)”小节描述了详细步骤；若鉴权等级为不需要鉴权，则跳过该步骤。
- 步骤2** 鉴权通过后，由CRG控制将鉴权模块工作时钟停掉，并切换到DFT时钟；由CRG控制将鉴权模块模块复位切换到DFT复位。

----结束

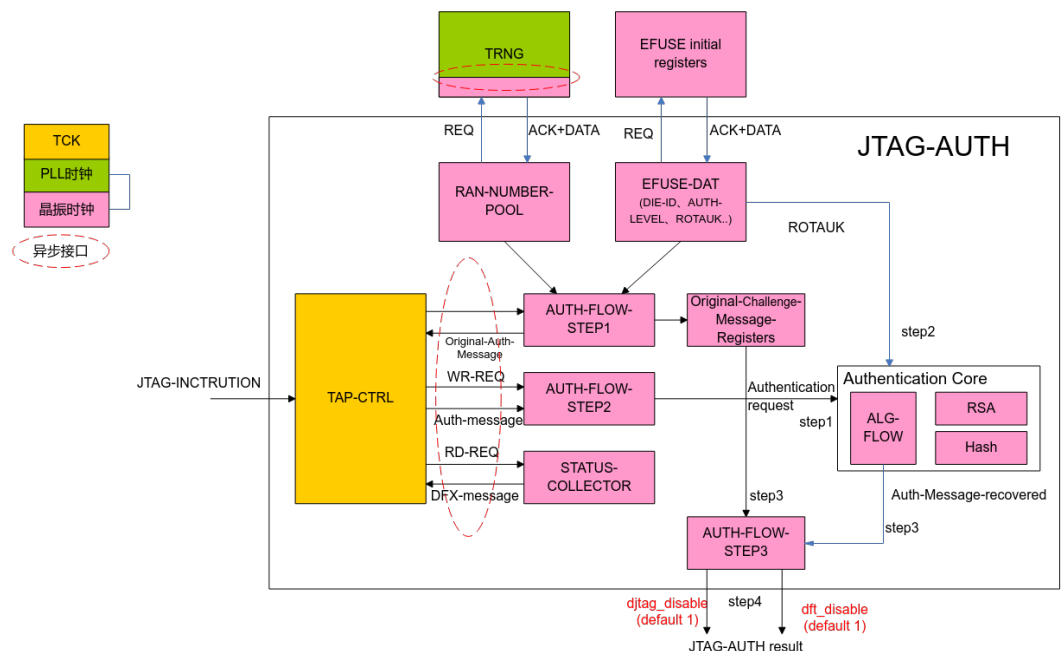
5.9 模块接口关系

如图5-11，展示了JTAG-AUTH的内部接口时钟关系。

JTAG-AUTH与TRNG的接口按同步设计，TRNG内部完成异步处理。

TAP控制器和JTAG-AUTH其他逻辑间的交互按照异步处理。

图 5-11 JTAG 鉴权时钟关系



5.10 芯片调试 DFX 功能特性描述



5.10.1 模块维测

请参见“[状态信息查询](#)”。

5.10.2 软硬件接口故障

- TRNG获取随机数失败，可参考“[状态信息查询](#)”获知具体情况。
- E-Fuse初始化失败，可参考获知具体情况。

5.10.3 硬件内部数据完整性错误

- RSA算法计算出现内部SRAM ECC1bit/2bit错误。
- 原始挑战信息的Parity校验错误。

5.11 性能指标

- 内部划分的各个部分性能参数。

5.11.1 实际测试结果

无

5.11.2 吞吐率（Throughput）

无

5.11.2.1 延迟（Latency）

鉴权使用的RSA和SHA256算法的延迟（Latency）如[表5-17](#)所示。

表 5-17 算法核计算 延迟（Latency）

行为	延迟（Latency）Value
RSA 4096bit公钥计算时间（公钥指数选择0x10001）；	2.5*10 ⁵ Cycles
224bit数据SHA256bit计算时间；	40 Cycles



6 SPI

6.1 概述

SPI（SerialPeripheralInterface）控制器，可以作为一个主设备与外部的设备来进行同步串行通信。支持以下接口协议：

- AMotorolaSPI-compatibleinterface。
- ATexasInstrumentssynchronousserialinterface。
- ANationalSemiconductorMicrowireinterface。

主要应用于外接触摸屏、SD卡、WIFI等。

6.2 特性

SPI具有以下特点：

- 支持串行主设备操作。
- 提供4个SPI控制器。
- 支持可编程数据传输速率，传输速率配置范围为200MHz/N（ $2 \leq N \leq 65534$ ，偶数），最高传输速率不低于25MHz。
- 支持主操作，每个SPI控制器支持最大2个Slave。
- 支持接口时钟频率可编程。
- 收/发为分开的16bit宽、深度为32的FIFO（发送FIFO和接收FIFO各一个），支持FIFO禁止或使能。
- 串行数据序列为先MSB，后LSB。
- 支持三种帧格式：Motorola SPI、National Microwire、TI SSP。
- 支持各种帧格式静态切换。
- 数据帧大小可编程：4bit ~ 16bit。
- 支持每个片选的有效极性可配置。
- 支持一个组合中断连接到中断控制器，中断类型包括：接收FIFO中断、发送FIFO中断、传输冲突中断。



- 支持初始中断状态查询和屏蔽后中断状态查询。
- 支持关断SPI接口时钟以节省功耗。
- 内部提供环回测试模式。
- Motorola SPI帧格式时，串行时钟相位可配置；SPI传输时数据跟时钟的关系支持软件可配。

6.3 关键术语

如表6-1所示的术语将在本章节表示特殊的含义。

表 6-1 SPI 关键术语描述

术语	描述
SPI	Serial Peripheral Interface，高速同步串行口，是Motorola公司推出的一种同步串行接口。

6.4 遵循协议

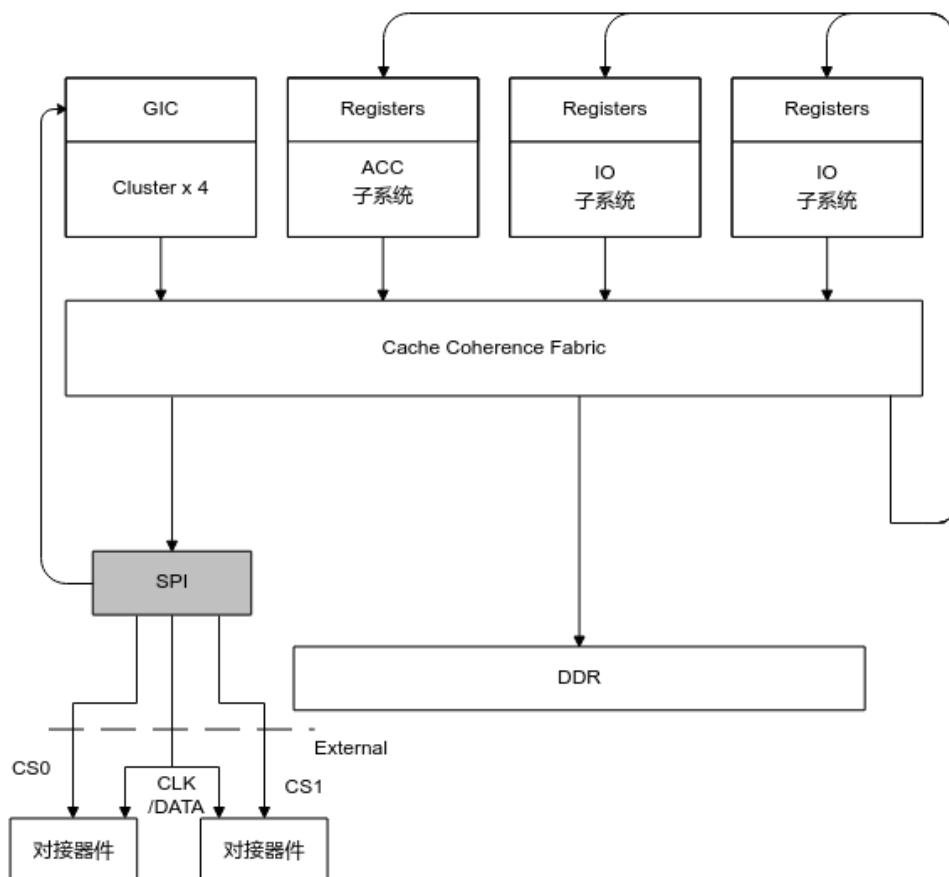
- AMotorolaSPI-compatibleinterface。
- ATexasInstrumentssynchronousserialinterface。
- ANationalSemiconductorMicrowireinterface。

6.5 功能描述

6.5.1 应用场景

SPI接口的应用框图如图6-1所示。

图 6-1 SPI 应用框图



SPI工作时，通过片选信号SPICSN0、SPICSN1、SPICSN2、SPICSN3来选择相应的从机，进行同步串行数据传输，传输时SPI按照数据高位在前，低位在后的方式传输。

CPU可通过总线配置SPI的寄存器配置数据传输到对端，或者读取寄存器获取接收到的对端数据。SPI在传输完成或传输发生错误时，可以通过中断通知CPU进行处理。

6.5.2 相关模块

在系统中，SPI模块会和下面多个模块进行交互，分别描述如下：

- CPU：CPU通过总线访问SPI的寄存器接口，操作属性必须为Device或Strongly-Ordered。CPU首先需要完成SPI的传输模式和速率配置，对于少量或即时性要求高的数据，CPU可以直接通过FIFO接口读写完成数据传输。
- GIC：SPI在传输时如果发生错误，例如传输冲突等，以及在非DMAC搬移模式时，如果接收FIFO高于水线，或发送FIFO低于水线，均会产生中断信号给GIC模块，通过GIC将中断发送给对SPI具有控制权的CPU。

6.5.3 功能原理

6.5.3.1 传输协议

支持3种传输帧格式，通过SPI_CTRLR0[frf]配置：

- 00: MotorolaSPI帧格式;
- 01: TISSP帧格式;
- 10: NationalMicrowire帧格式;
- 11: 保留。

说明

图6-2、图6-13中，以下缩略语或字母意义不变：

- MSB: MostSignificantBit
- LSB: LeastSignificantBit
- Q: undefinedsignal

图6-2、图6-9中，spo、sph不同取值由MotorolaSPI帧格式内部字段控制。

6.5.3.1.1 MotorolaSPI 帧格式（spo=0、sph=0）

图 6-2 MotorolaSPI 单帧帧格式（spo=0、sph=0）

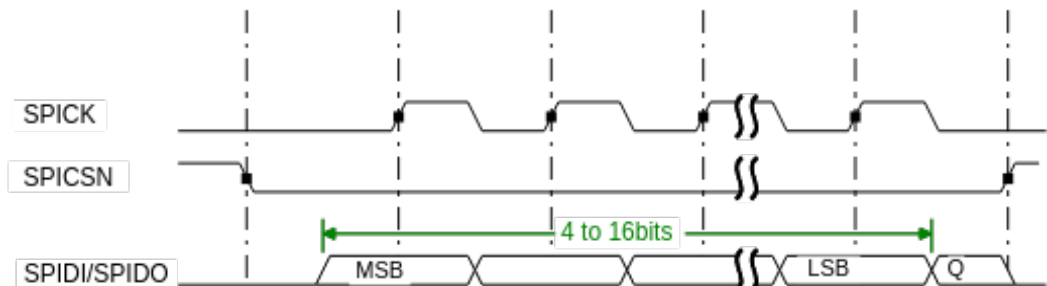
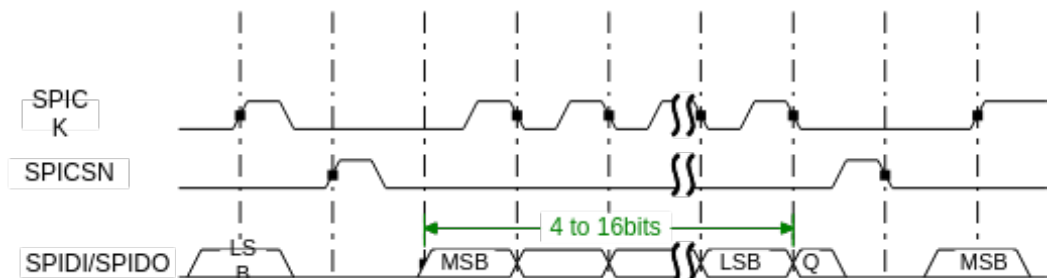


图 6-3 MotorolaSPI 连续帧帧格式（spo=0、sph=0）



在该模式下，当SPI处于空闲状态时：

- SPICK信号设置为低。
- SPICSNx设置为高。
- 发送数据线SPIDO强制为低。

当SPI处于使能状态，而且发送FIFO内不存在有效数据时，设置SPICSNx信号为低，表示开始传输数据。这样就会使能Slave的数据放在Master的输入SPIDI线上。半个SPICK时钟周期之后，有效的Master数据传输到SPIDO管脚。此时Master和Slave数据都已经有效，SPICK主时钟管脚会在接下来的半个SPICK时钟周期之后变化为高电平。数据就会在SPICK时钟的上升沿被捕获，在时钟的下降沿被传送。

如果传输单个Word，当捕捉到传输的最后1bit数据，SPICSN会在接下来的1个SPICK时钟之后恢复为高电平。

如果是连续的传输，SPICSN信号在2个Word传输之间必须将SPICK时钟拉高一个时钟周期。这是因为sph为0时，Salve选择管脚会固定其内补串行设准寄存器的数据，使它不会变化。因此在连续传输时，主设准必须在每2个Word传输之间将SPICSN信号拉高。连续传输结束时，SPICSN会在捕捉到最后1bit数据之后的1个SPICK时钟周期之后恢复为高电平。

6.5.3.1.2 MotorolaSPI 帧格式（spo=0、sph=1）

图 6-4 MotorolaSPI 单帧帧格式（spo=0、sph=1）

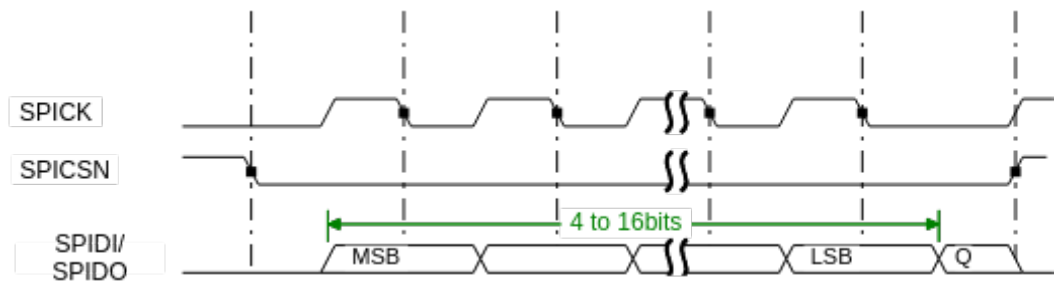
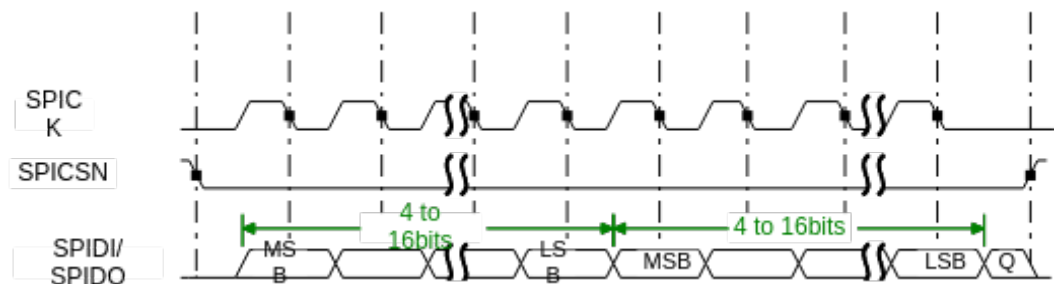


图 6-5 MotorolaSPI 连续帧帧格式（spo=0、sph=1）



在该模式下，当SPI处于空闲状态时：

- SPICK信号设置为低。
- SPICSNx设置为高。
- 发送数据线SPIDO强制为低。

当SPI为使能状态，而且在发送FIFO内补有有效数据时，设置SPICSNx信号为低表示开始传输数据。半个SPICK时钟周期之后，Master和Slave的有效数据就分别在各自的传输线上有效。同时，SPICK从一个上升沿开始有效。数据就会在SPICK时钟的上升沿被捕获，在时钟的下降沿被传送。

如果传输单个Word，当捕捉到传输的最后1bit数据，SPICSNx会在接下来的1个SPICK时钟之后恢复为高电平。

当连续传输时，在传输数据Word之间SPICSNx保持为低。连续传输结束时，SP_SPICSNx会在捕捉到最后1bit数据之后的1个SPICK时钟之后恢复为高电平。

6.5.3.1.3 MotorolaSPI 帧格式（spo=1、sph=0）

图 6-6 MotorolaSPI 单帧帧格式（spo=1、sph=0）

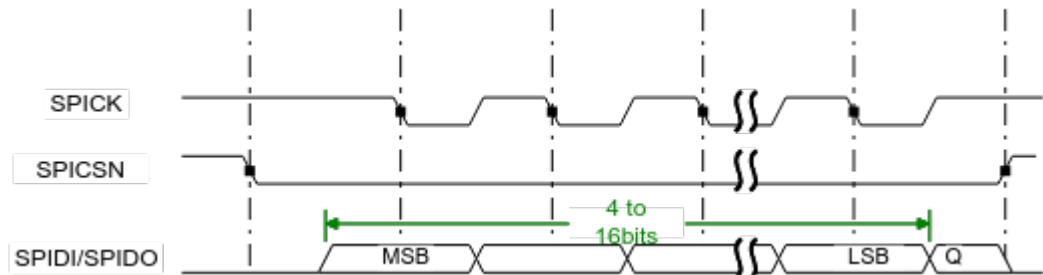
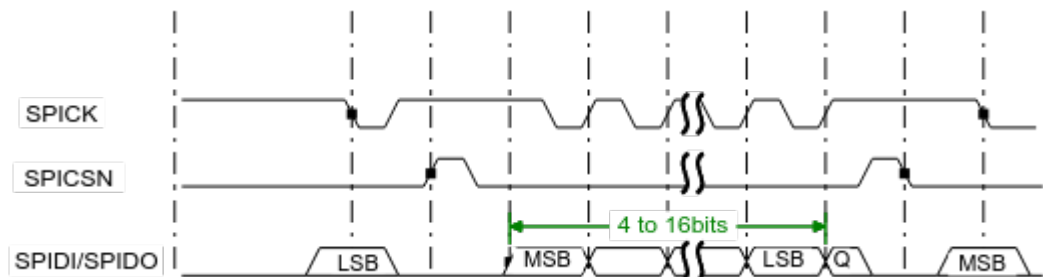


图 6-7 MotorolaSPI 连续帧帧格式（spo=1、sph=0）



在该配置下，当SPI处于空闲状态时：

- SPICK信号设置为高。
- SPICSNx设置为高。
- 发送数据线SPIDO强制为低。

当SPI为使能状态，而且在发送FIFO内补有有效数据时，设置SPICSNx信号为低表示开始传输数据，此时Slave的数据就会立刻发送到Master的接收数据线SPIDI上。半个SPICK周期之后，Master的有效数据就传送到SPIDO线上。再过半SPICK时钟周期之后，SPICKMasterpin设置为低。这意味着数据是在SPICK时钟的下降沿被捕获，在SPICK时钟的上升沿被传送。

如果传输单个Word，当捕捉到传输的最后1bit数据，SPICSN会在接下来的1个SPICK时钟之后恢复为高电平。

如果是连续的传输，SPICSN信号在2个Word传输之间必须拉高。这是因为，当sph为0时，salve选择管脚固定其内补串行设准寄存器的数据，使它不会变化。SPICSNx就会在捕捉到最后1bit数据之后的1个SPICK时钟周期之后恢复为高电平。

6.5.3.1.4 MotorolaSPI 帧格式（spo=1、sph=1）

图 6-8 MotorolaSPI 单帧帧格式（spo=1、sph=1）

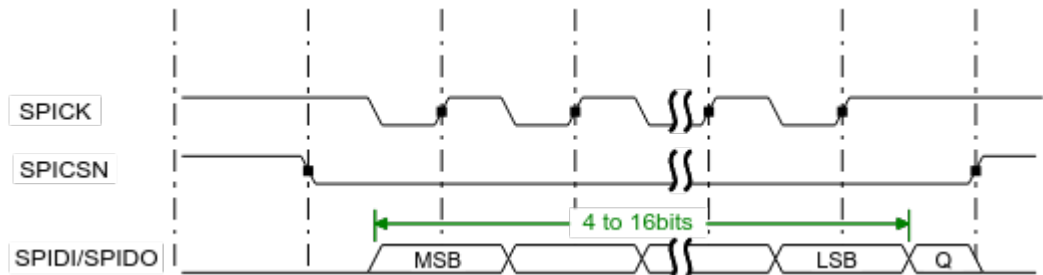
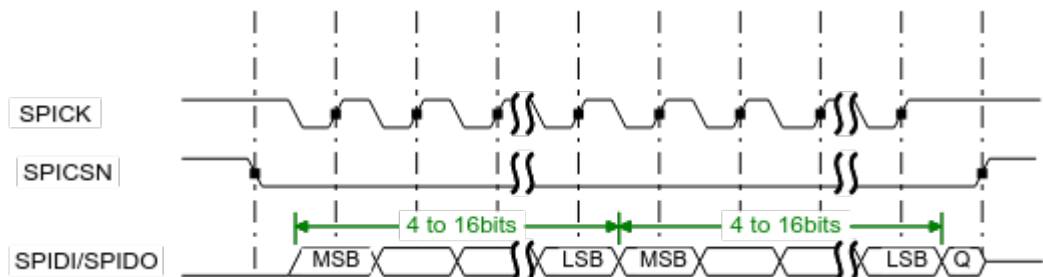


图 6-9 MotorolaSPI 连续帧帧格式（spo=1、sph=1）



在该模式下，当SPI处于空闲状态时：

- SPICK信号设置为高。
- SPICSNx设置为高。
- 发送数据线SPIDO强制为低。

当SPI为使能状态，而且在发送FIFO内补有有效数据时，设置SPICSNxMaster信号为低表示开始传输数据。半个SPICK时钟之后，Master和Slave数据都会在各自己的传输线上有效。同时，时钟SPICK从一个下降沿开始有效。数据是在SPICK时钟的下降沿被捕获，在时钟的上升沿被传送。

如果传输单个Word，SPICSNx会在捕获到传输的最后1bit数据之后的1个SPICK时钟周期之后恢复为高电平。

如果是连续传输，SPICSNx信号始终保持为低，直到传输结束时，SPICSNx就会在捕获到最后1bit数据之后的1个SPICK时钟周期之后恢复到空闲为高的状态。对于连续的传输来说，信号SPICSN在传输的过程中一直保持为低，结束的方式与单个传输的方式相同。

6.5.3.1.5 TI 同步串行帧格式

图 6-10 TI 同步串行单帧帧格式

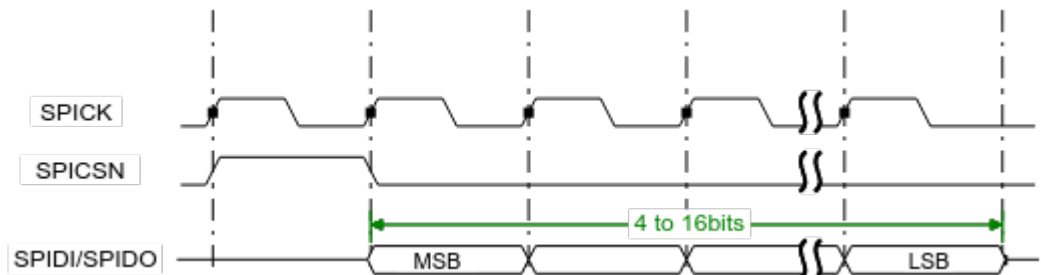
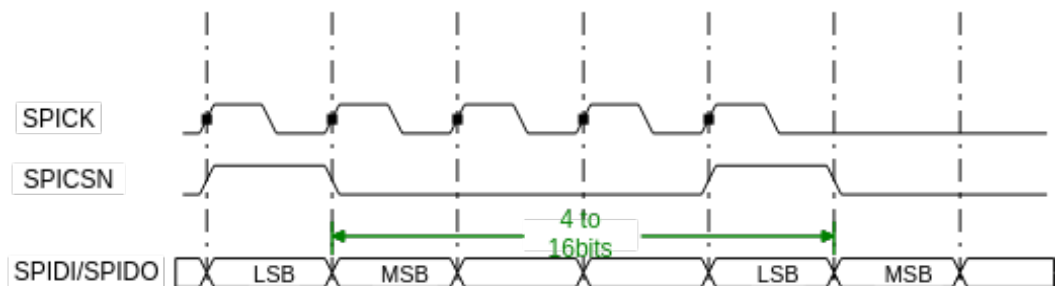


图 6-11 TI 同步串行连续帧帧格式



在该模式下，当SPI处于空闲状态时：

- SPICK为低。
- SPICSNx为低。
- 传输数据线SPIDO保持为高阻。

一旦发送FIFO有数据，SPICSNx就会产生一个SPICK时钟周期的高电平脉冲，将被发送的数据就会从发送FIFO传送到发送逻辑串行移位寄存器。在SPICK时钟的下一个上升沿，4bits~16bits数据帧的MSB就会从SPIDO移位输出。同样，从外补串行Slave设准接收数据的MSB会从SPIDI管脚移位输入。

SPI和片外串行设准在SPICK时钟的下降沿将数据存入串行移位寄存器。接收串行寄存器在接收到LSB之后的第1个SPICK时钟上升沿将数据送给接收FIFO。

6.5.3.1.6 NationalSemiconductorMicrowire 帧格式

图 6-12 NationalSemiconductorMicrowire 单帧帧格式

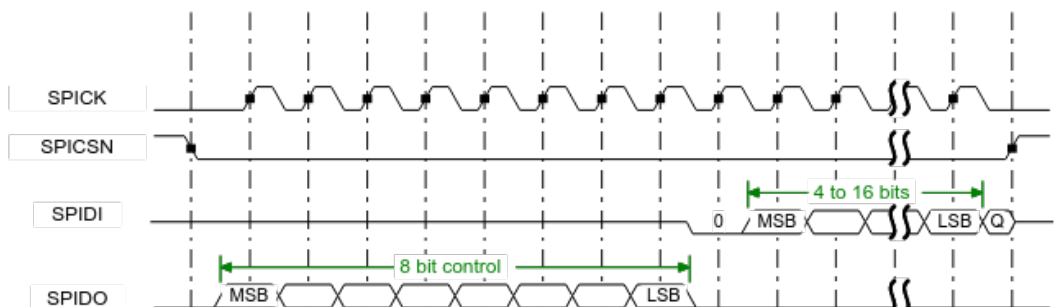
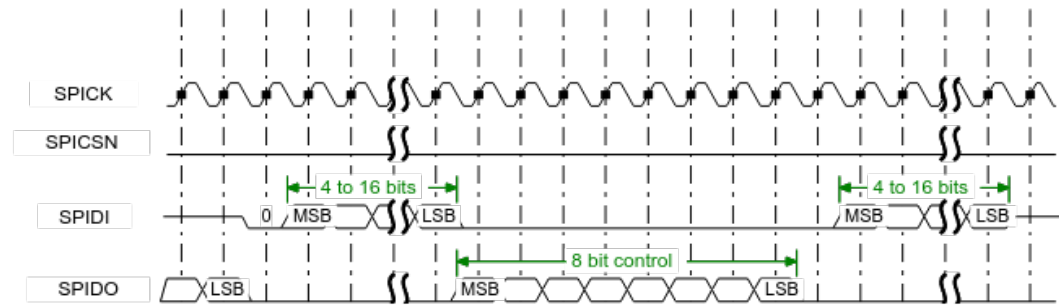


图 6-13 NationalSemiconductorMicrowire 连续帧格式



Microwire的格式与SPI的格式非常相近，使用Master-slave信息的传输根根，只不过SPI是全双工通信，而Microwire半双工通信。在SPI向外补芯片发送串行数据的时候，都要先加8bit控制字。在这个过程中，SPI没有接收到任何数据。传输完毕之后，片外芯片对接收到的数据进行解码，在与8bit控制信息间隔一个时钟周期之后，Slave开始响应所需求的数据。返回的数据长度为4bit ~ 16bit，使得整个帧的长度为13bit ~ 25bit。

在这个模式下，当SPI处于空闲状态时：

- SPICK信号设置为低。
- SPICSNx设置为高。
- 发送数据线SPIDO强制为低。

向发送FIFO内补写进一个控制字节开始一次传送。SPICSNx的下降沿引发数据的传输，发送FIFO的数据被发送到串行移位寄存器，8bit控制帧的MSB被发送到发送管脚SPIDO。在帧的传送过程中，SPICSNx保持为低。SPIDI在这个传送过程中保持为高阻。

片外的串行从设准在SPICK时钟的每一个上升沿将数据锁存到串行移位寄存器中。当从设准锁存完最后1bit数据之后，在接下来的1个时钟周期的等待时间里，对接收到的数据开始解码，然后从设准反馈给SPI所要求的数据。每bit都是在SPICK时钟的下降沿写到SPIDI的。对单个数据传送来说，在帧的结尾，SPICSNx在最后1个bit写到接收串行寄存器之后的1个时钟周期后拉高，这样就使接收到的数据传送到接收FIFO。

对于连续的传送来说，数据传送的开始和结束都和单个数据的传送方式相同。在这个传送过程中，信号SPICSNx时一直保持为低的，传送的数据也是连续的。下一帧的控制字直接和上一帧的LSB相邻。当帧的LSB锁存到SPI之后，接收到的每一个数值都是在SPICK时钟的下降沿取自接收移位寄存器。

6.6 工作方式

6.6.1 管脚复用配置

SPI的管脚与GPIO管脚、信号probe复用。使用SPI前，需要配置如下IOCONFIG寄存器。

- 配置IOMG101[2:0]（物理地址为0xFA86012C）为0b000，SPICK0引脚为SPI0模块的spick功能，为SPI0的时钟信号。



- 配置IOMG102[2:0]（物理地址为0xFA860130）为0b000；SPIDI0引脚为SPI0模块的spidi功能，为SPI0的输入数据信号。
- 配置IOMG103[2:0]（物理地址为0xFA860134）为0b000；SPIDQ0引脚为SPI0模块的spido功能，为SPI0的数据输出。
- 配置IOMG104[2:0]（物理地址为0xFA860138）为0b000；SPICSN0_0引脚为SPI0模块的spicsn_0功能，为SPI0的片选0。
- 配置IOMG105[2:0]（物理地址为0xFA86013C）为0b000；SPICSN0_1引脚为SPI0模块的spicsn_1功能，为SPI0的片选1。
- 配置IOMG086[2:0]（物理地址为0xFA8600F4）为0b001；SPICK1引脚为SPI1模块的spick功能，为SPI1的时钟信号。
- 配置IOMG087[2:0]（物理地址为0xFA8600F8）为0b001；SPIDI1引脚为SPI1模块的spidi功能，为SPI1的输入数据信号。
- 配置IOMG088[2:0]（物理地址为0xFA8600FC）为0b001；SPIDQ1引脚为SPI1模块的spido功能，为SPI1的输入数据信号。
- 配置IOMG089[2:0]（物理地址为0xFA860100）为0b001；SPICSN1_0引脚为SPI1模块的spicsn_0功能，为SPI1的片选0。
- 配置IOMG063[2:0]（物理地址为0xFA86009C）为0b010；SPICK2引脚为SPI2模块的spick功能，为SPI1的时钟信号。
- 配置IOMG067[2:0]（物理地址为0xFA8600AC）为0b010；SPIDI2引脚为SPI2模块的spidi功能，为SPI1的输入数据信号。
- 配置IOMG066[2:0]（物理地址为0xFA8600A8）为0b010；SPIDQ2引脚为SPI2模块的spido功能，为SPI1的输入数据信号。
- 配置IOMG064[2:0]（物理地址为0xFA8600A0）为0b010；SPICSN2_0引脚为SPI2模块的spicsn_0功能，为SPI1的片选0。
- 配置IOMG065[2:0]（物理地址为0xFA8600A4）为0b010；SPICSN2_1引脚为SPI2模块的spicsn_1功能，为SPI1的片选1。
- 配置IOMG206[2:0]（物理地址为0xFA860288）为0b011，SPICK3引脚为SPI3模块的spick功能，为SPI0的时钟信号。
- 配置IOMG205[2:0]（物理地址为0xFA860284）为0b011；SPIDI3引脚为SPI3模块的spidi功能，为SPI0的输入数据信号。
- 配置IOMG262[2:0]（物理地址为0xFA860368）为0b011；SPIDQ3引脚为SPI3模块的spido功能，为SPI0的数据输出。
- 配置IOMG215[2:0]（物理地址为0xFA8602AC）为0b011；SPICSN3_0引脚为SPI3模块的spicsn_0功能，为SPI0的片选0。
- 配置IOMG197[2:0]（物理地址为0xFA860264）为0b011；SPICSN3_1引脚为SPI0模块的spicsn_1功能，为SPI0的片选1。

6.7 工作流程

6.7.1 时钟门控

上电时SPI工作时钟默认为打开状态。SPI工作完成后，可以关闭SPI工作时钟，以降低芯片功耗；但需要确保硬件已处于空闲状态（SPI_SR[0]为0）。



6.7.1.1 打开门控时钟

具体配置步骤如下：

- 步骤1** 读SUBCTRL系统控制器寄存器SC_SPI_CLK_ST，获取当前时钟门控信息。
- 步骤2** 如果SC_SPI_CLK_ST为低，进入**步骤3**；SC_SPI_CLK_ST为高，表示SPI时钟已经打开，直接跳过下面的步骤结束操作。
- 步骤3** 配置SC_SPI_CLK_EN为高，打开SPI模块的时钟门控。在初始化期间，打开门控后，等到10μs查看门控状态寄存器SC_SPI_CLK_ST，若为1，则进行下一步，否则认为存在异常。

----结束

6.7.1.2 关闭门控时钟

具体配置步骤如下：

- 步骤1** 读SPI_SR，若SPI_SR[0]为0，则进入**步骤2**；若SPI_SR[0]为1，则延时等待。
- 步骤2** 读bottom系统控制器寄存器SC_SPI_CLK_ST，获取当前时钟门控信息。
- 步骤3** 如果SC_SPI_CLK_ST为高，进入**步骤4**；SC_SPI_CLK_ST为低，表示SPI时钟已经关闭，直接跳过下面的步骤结束操作。
- 步骤4** 配置SC_SPI_CLK_DIS为高，打开SPI模块的时钟门控。

----结束

6.7.2 时钟配置

通过配置波特率选择寄存器SPI_BAUDR，可以设置SPI工作的波特率，波特率计算公式如下：

当前波特率=内部总线时钟频率/分频系数。

分频系数必须为偶数（8~65534）；内部总线时钟频率为200MHz。

6.7.3 软复位

对接器件容许的情况下，可以任意时间对SPI工作逻辑进行软复位，不会对系统造成影响。

- 通过配置SUBCTRL系统控制器中SC_SPI_RESET_REQ对应位为0b1，可实现对SPI工作逻辑的单独软复位。
- 通过配置SUBCTRL系统控制器中SC_SPI_RESET_DREQ对应位为0b1，可实现对SPI工作逻辑的单独软复位撤离，在初始化期间，配置复位撤离后，等待10μs查看复位状态寄存器SC_SPI_RESET_ST，若为0，则进行后续初始化，否则认为存在异常。

须知

SPI软复位不会将SPI本身寄存器恢复成默认值，只有上电复位、自刷新复位或全局软复位可以将SPI本身寄存器恢复成默认值。



6.7.4 初始化

初始化配置步骤如下：

- 步骤1** 向SPI_SSIENR[0]写0，关闭SPI。
- 步骤2** 配置寄存器SPI_CTRLR0，配置PERI_SUB寄存器spi*_cs_polarity根据对接从设备的时序来配置时钟极性参数。

须知

请务必保证与对接器件一致，例如：帧格式（MOTO/Microwire）、传输模式、极性等。

- 步骤3** 如果主设备需要从从设备接收数据则需要配置寄存器SPI_CTRLR1，以确定接收数据的个数。
- 步骤4** 配置寄存器SPI_BAUDR，设置传输的波特率。
- 步骤5** 配置发送FIFO阈值寄存器SPI_TXFTLR和接收FIFO阈值寄存器SPI_RXFTLR，设置发送和接收FIFO的阈值水准。
- 步骤6** 配置寄存器SPI_IMR，屏蔽不需要的中断。
- 步骤7** 配置寄存器SPI_SER，来选择对接的从设备。
- 步骤8** 向SPI_SSIENR[0]写1，使能SPI。

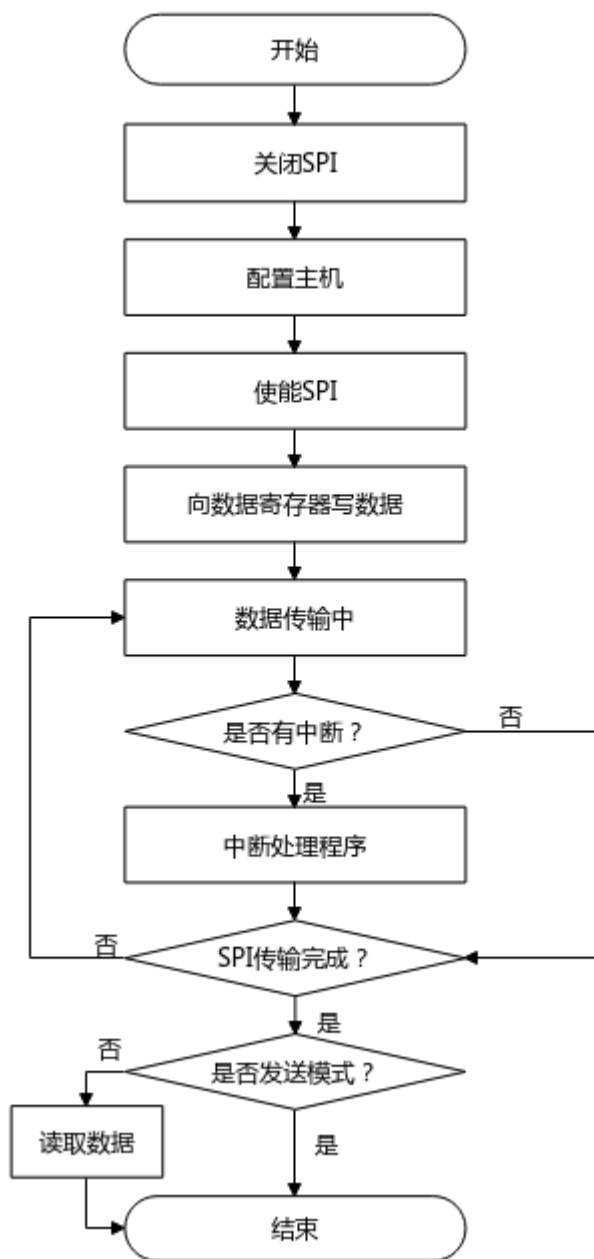
----结束

6.7.5 中断或查询方式下的数据传输

主设备模式下的工作流程图如图6-14所示。



图 6-14 主机模式工作流程图



6.7.6 数据发送

SPI使能后，会自动将发送FIFO中的数据发送出去，操作流程如下：

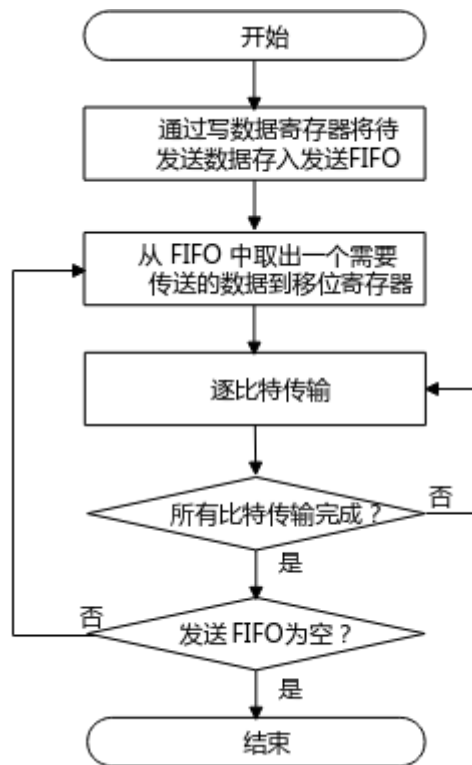
- 步骤1** 读取SPI_SER寄存器，确认已经选择了对接的从设备。
- 步骤2** 将待发送的数据按顺序写入数据寄存器SPI_DR，数据存放在发送FIFO中，启动SPI数据传输。
- 步骤3** 将待发送数据的第一个数据从发送FIFO中取出，放到移位寄存器中。
- 步骤4** 由串行时钟控制，通过发送数据线SPIDO逐比特发送FIFO中的第一个数据，高位在前，低位在后。

- 步骤5** 把第二个数据从发送FIFO中取出放到移位寄存器中发送，按照第3步操作，直到发送FIFO为空。
- 步骤6** 当产生发送FIFO空中断时，如果还有需要传输的数据，则需要向数据寄存器中写数据，如果没有，则传输完成。
- 步骤7** 向SPI_SSIENR[0]写0，关闭SPI。

----结束

SPI发送数据的工作流程如图6-15所示。

图 6-15 SPI 发送数据流程图



6.7.7 数据接收

数据接收步骤如下：

- 步骤1** 读取SPI_SER寄存器，确认已经选择了对接的从设备。
- 步骤2** 向数据寄存器SPI_DR中写需要获取数据的地址和读命令，启动SPI数据传输。
- 步骤3** 通过接收数据线SPIDI逐比特接收数据。
- 步骤4** 接收完一个数据之后存入到接收FIFO，继续接收。
- 步骤5** 如果接收FIFO满中断产生时，则将接收FIFO中的数据写入数据寄存器；如果接收FIFO未满，则继续接收数据。
- 步骤6** 将接收FIFO中的数据写入数据寄存器，直到接收FIFO为空。

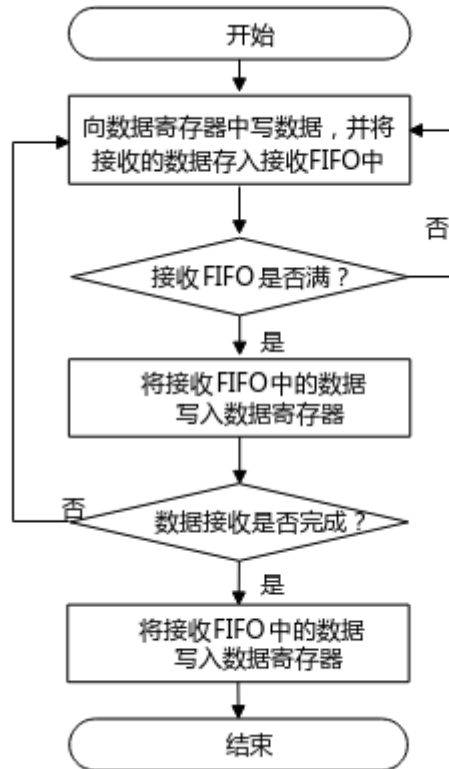


步骤7 向SSIENR[0]写0，关闭SPI。

----结束

SPI接收数据的工作流程如图6-16所示。

图 6-16 SPI 接收数据流程图



6.7.8 三线 SPI 模式

不支持三线SPI，如果实际应用中需要使用三线SPI，需要通过设置将对应管脚的复用关系设置为普通GPIO模式，然后通过软件来模拟三线SPI。



7 ESPI

7.1 功能、特点概述

ESPI（扩展串行总线）是基于SPI总线协议和AXI协议，面向SOC系统芯片应用开发的占用管脚少、可以满足功能、性能和可靠性要求的实现片间资源互访的总线。

ESPI分为master和slave两种工作模式。espi_master模式，通过AXI slave口接收AXI master口发送的AXI读写命令转换成ESPI包格式并通过ESPI接口实现对接器件的远端访问。espi_slave模式，解析ESPI主控发送的命令包，转换成相应的AXI命令，通过AXI master接口完成芯片寄存器的访问，该模式还可以达到启动芯片的目的。

- 支持ESPI master接口，支持1bit/2bit/4bit/8bit数据位宽。
- 支持ESPI slave接口，支持1bit/2bit/4bit/8bit数据位宽。
- 支持APB3.0 Slave接口用于寄存器配置。
- 支持1个AMBA 3 AXI master接口，数据位宽64bit。
- 支持1个AMBA 3 AXI slave接口，数据位宽64bit。
- Master模式支持ESPI命令包的产生。
- Slave模式支持ESPI命令包格式的解析。
- 支持CRC校验。
- 支持错误检测上报和超时检测上报。

7.2 运用背景

为了解决主控芯片local bus + HPI传输速率低，管脚多，不能支持原子操作等问题，决定开发一种占用管脚少，满足功能、性能、可靠性要求的片间访问总线。

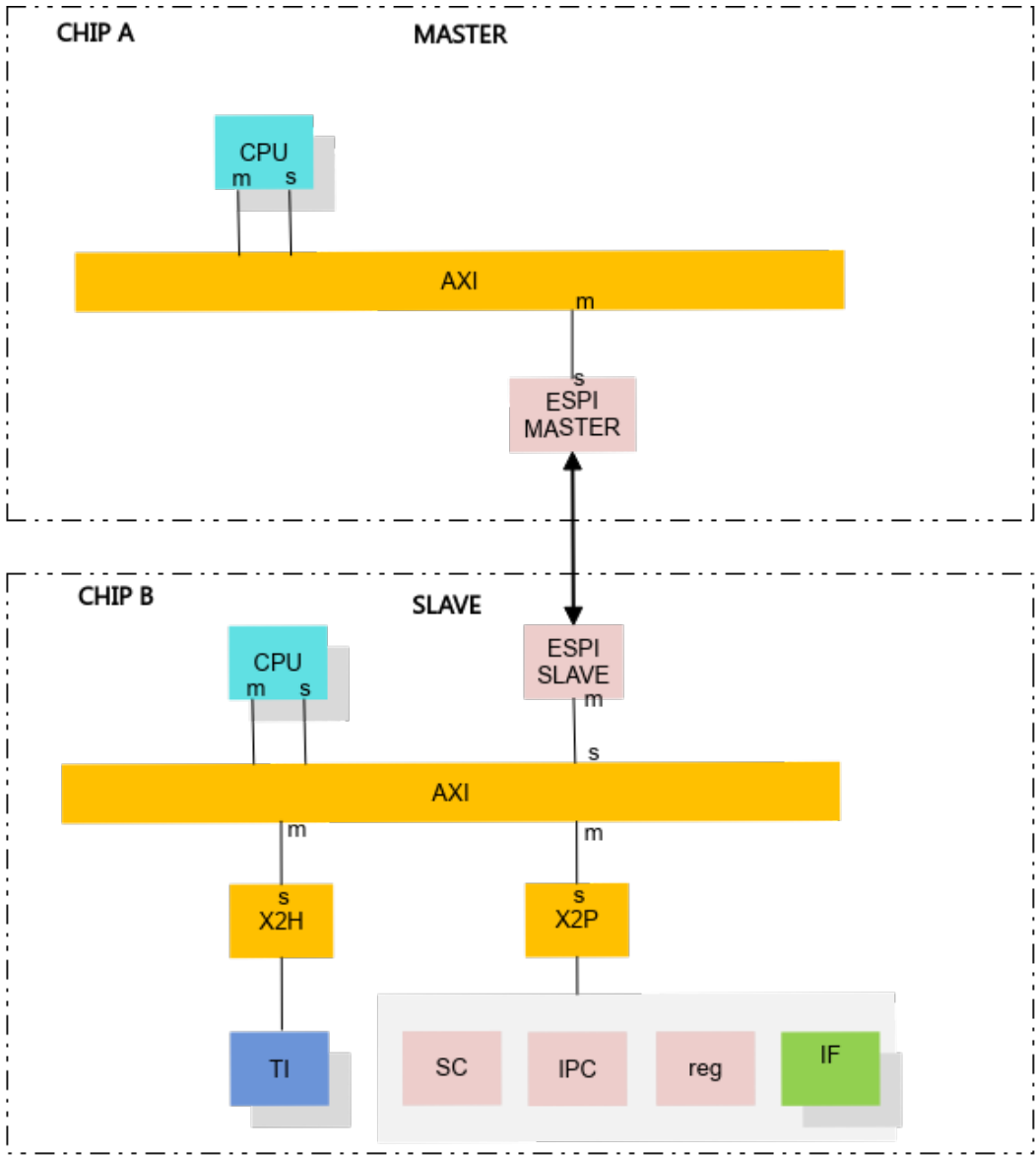
主片通过ESPI接口可以实现对从片寄存器空间的访问，或者实现对从片的加载功能。主模式最多支持4个片选。从片可以对接espi_master或者其他支持ESPI包格式的master器件，并对包格式进行解析，生成对应的AXI访问命令。

另外还可以通过ESPI接口实现对从片芯片加载配置，完成启动。如图7-1所示为ESPI应用框图。CHIPA、CHIPB通过ESPI来实现通信。master工作模式下，axi slave接收axi master发出的读写命令，对地址进行重映射，转换成ESPI包格式，然后espi master将命令发送给从片espi slave，实现对从片的访问。slave工作模式下，espi slave接收主



片espi master发出的ESPI命令，转换为AXI协议命令，然后通过axi master完成芯片寄存器的访问。

图 7-1 ESPI 应用框图示例



7.3 关键术语

如表7-1的术语将在本章节表示特殊的含义。

表 7-1 ESPI 关键术语描述

术语	描述
ESPI	Enhanced Serial Protocol Interface



术语	描述
SPI	Serial Protocol Interface

7.4 Master 模式流程

7.4.1 Master 模式配置访问流程

7.4.1.1 场景描述

软件可以发出axi命令经ESPI master模式转换成ESPI总线访问对端期间。

7.4.1.2 配置流程

- 步骤1** 配置ESPI_LOCK（0x038），写入0x36323136为ESPI配置寄存器解锁，回读则是锁定状态指示。
- 步骤2** 配置ESPI工作模式，工作模式由模块输入管脚espi_work_mode（1bit）决定，输入0为Slave模式，输入1为Master模式（该配置值由系统控制器决定）。
- 步骤3** 配置ESPI控制寄存器ESPI_CTRL（0x000），master下包括寄存器自动锁定使能bit[11]，时钟动态gating使能bit[10]，读写插拍采样时刻选择bit[9:8]，采样延时配置bit[7]，数据发送沿bit[6]、错误响应使能bit[5]，数据采样沿bit[3]，未指定的bit的值保持不变，详细配置参数如下：
1. 配置空闲时钟gating延时寄存器ESPI_MST_IDLE_CNT（0x4C）。
 2. 配置超时检测门限寄存器ESPI_OT_GATE（0x01C）。
 3. 配置ESPI SCLK（0x004）分频系数。
 4. 配置ESPI Master插拍寄存器ESPI_MST_PIPE_EN（0x304），具体配置值根据具体场景决定。
- 步骤4** 获取地址重映射空间资源，此处有两种方法：
1. 采用读取：（偏移地址0x14）或16M（ESPI_SCR_16M：偏移地址0x18）地址空间资源获取寄存器来获取空闲的资源。
 2. 采用软件查询方式，查询重映射空间配置寄存器bit[20]，看是否被占用。
- 步骤5** 按需求配置地址重映射空间寄存器（4M偏移地址0x100 + 0x04×n1，16M偏移地址0x200 + 0x04×n2）。
- 步骤6** 等待2μs，保证ESPI时钟稳定。
- 步骤7** 配置ESPI中断屏蔽寄存器ESPI_ABNOR_IMSK（0x010），具体配置值由应用场景决定。
- 步骤8** 有ESPI中断，读ESPI_ABNOR_INTR_RAW（0x008）、ESPI_ABNOR_INTR_STAT（0x00C）寄存器获取中断状态，中断采用写1清除中断（bit[12]：slv_fsm_err_raw和bit[11]：slv_rx_fifo_fill_raw不能写清，仅在复位时可清除）。如下为ESPI_ABNOR_INTR_STAT寄存器描述，ESPI_ABNOR_INTR_RAW和ESPI_ABNOR_INTR_STAT按bit对应（略）。

----结束



7.4.1.3 配置约束

对于ESPI MASTER，当ESPI接口外面接打拍逻辑时，同时当ESPI发送读命令访问slave时需要配置读数据采样的延迟点，如下：需要将espi_out_dly_sel根据外面打拍逻辑的情况配置，比如原来p2的ESPI外面的打拍逻辑有一拍的延迟，这里配置必须大于1拍，此时tx_edge，sample_edge可以任意配置。如果不配置，采用默认配置，同时tx_edge，sample_edge也采用默认配置，则读数据会出错，在使用时应该注意一下，看默认配置会不会影响应用。

ESPI寄存器CTRL[9:8]读写采样延时不能配置为0。

7.5 Slave 模式流程

7.5.1 Slave 模式配置访问流程

7.5.1.1 场景描述

ESPI Slave收到端口信号，可以转换成axi信号发往从片的总线通路。

7.5.1.2 配置流程

- 步骤1** 配置ESPI_LOCK（0x038），写入0x36323136为ESPI配置寄存器解锁，回读则是锁定状态指示。
- 步骤2** 配置ESPI工作模式及传输位宽，工作模式由模块输入管脚espi_work_mode（1bit）决定，输入0为Slave模式，输入1为Master模式（该配置值由系统控制器决定）；Slave下的传输位宽由输入管脚espi_slv_dw（2bit）决定，该配置值由系统控制器决定，如下：
 1. espi_slv_dw配置为0b00时，支持slave接口数据位宽为1bit，1Byte数据传输需要8个sclk完成。
 2. espi_slv_dw配置为0b01时，支持slave接口数据位宽为2bit，1Byte数据传输需要4个sclk完成。
 3. espi_slv_dw配置为0b10时，支持slave接口数据位宽为4bit，1Byte数据传输需要2个sclk完成。
 4. espi_slv_dw配置为0b11时，支持slave接口数据位宽为8bit，1Byte数据传输需要1个sclk完成。
- 步骤3** 配置控制寄存器ESPI_CTRL（0x000），其中bit[0]和bit[13:12]为RO寄存器，对应工作模式及slave下传输位宽，配置CRC使能（bit[4]）、寄存器自动锁定使能（bit[11]），未指定的bit的值保持不变。
- 步骤4** 配置ESPI_AXI_BASE_ADDR（0x050）来配置AXI Master高axi[47:32]bit地址（如果位宽为32bit，该寄存器无需配置）。
- 步骤5** 配置ESPI Slave直接访问内部寄存器使能控制ESPI_REG_CFG_EN（0x054）。
- 步骤6** 配置超时检测门限寄存器ESPI_OT_GATE（0x01C）。
- 步骤7** 配置ESPI Slave插拍寄存器ESPI_SLV_PIPE_EN（0x300），具体配置值根据具体场景决定。



- 步骤8** 等待2μs，保证ESPI时钟稳定。
- 步骤9** 配置ESPI中断屏蔽寄存器ESPI_ABNOR_IMSK（0x010），具体配置值由应用场景决定。
- 步骤10** 有ESPI中断，读ESPI_ABNOR_INTR_RAW（0x008）、ESPI_ABNOR_INTR_STAT（0x00C）寄存器获取中断状态，中断采用写1清除中断（bit[12]：slv_fsm_err_raw和bit[11]：slv_rx_fifo_fill_raw不能写清，仅在复位时可清除）。如下为ESPI_ABNOR_INTR_STAT寄存器描述，ESPI_ABNOR_INTR_RAW和ESPI_ABNOR_INTR_STAT按bit对应（略）。
- 结束

7.5.1.3 配置约束

ESPI作为从片时，从片解复位后，建议第一步操作即将从片ESPI寄存器锁定，避免误配置ESPI寄存器，导致从片无法访问。

ESPI Slave下配置的随路时钟SCLK不的高于ESPI的总线时钟aclk。

7.6 IP 可维可测

7.6.1 功能描述

- 1. ESPI提供当前工作模式查询ESPI_CTRL[1:0]，如表7-2所示。
- 2. ESPI提供CRC使能查询ESPI_CTRL[4]，如表7-2所示。
- 3. ESPI提供当前状态可查询ESPI_STAUS [0]，如表7-3所示。
- 4. ESPI提供当前总线工作状态可查询ESPI_STAUS [1]：4线/8线模式，如表7-3所示。
- 5. ESPI提供错误状态查询寄存器ESPI_FAULT_STATUS可查询：如表7-4所示。
- 6. ESPI提供错误地址查询寄存器ESPI_FAULT_ADDR，可以查询出错时的访问地址，如表7-5所示。
- 7. ESPI提供CRC校验错误计数器ESPI_SLV_CRC_CNT，可以查询产生CRC校验错误的次数，如表7-6所示。
- 8. ESPI提供SLAVE端写命令丢失计数器ESPI_SLV_WR_DISCARD_CNT，可以查询产生命令丢失的次数，如表7-7所示。

表 7-2 ESPI_CTRL 寄存器

bit	寄存器描述	配置值
[31:10]	保留	0x0
[9:8]	ESPI Master模式输入外部插拍情况下内部采样时刻选择。 00：外部不插拍情况下的采样时刻点； 01：外部插入1拍情况下的采样时刻点； 10：外部插入2拍情况下的采样时刻点； 11：外部插入3拍情况下的采样时刻点。	无需配置。



bit	寄存器描述	配置值
[7]	Master端数据采样沿延迟选择。 0：延迟2个aclk周期采样； 1：延迟1个aclk周期采样。	无需配置。
[6]	Master端数据发送沿选择。 0：上升沿发送； 1：下降沿发送。	无需配置。
[5]	AXI总线错误响应使能。 0：禁止； 1：使能。	由场景决定。
[4]	CRC校验使能。 0：禁止； 1：使能。	由场景决定。
[3]	Master端采样沿选择。 0：下降沿采样； 1：上升沿采样。	无需配置。
[2]	ESPI接口位选择 0：4bit； 1：8bit。	0x0
[1:0]	ESPI工作模式。 00：从模式； 01：主模式； 10：环回模式； 11：保留。	0x0

表 7-3 ESPI_STATUS

Bits	寄存器说明	配置值
ESPI_STATUS：偏移地址0x03C		
[0]	Master模式。 0：espi总线和AXI_SLAVE当前处于空闲状态； 1：espi总线和AXI_SLAVE当前处于忙状态。 Slave模式。 0：axi_master总线处于空闲状态； 1：axi_master总线处于忙状态。	-



表 7-4 ESPI_FAULT_STATUS 寄存器

Bits	寄存器说明	配置值
ESPI_FAULT_STATUS: 偏移地址0x024		
[17:8]	错误传输ID	-
[7:0]	错误类型。 0x00: 无错误; 0x01: Master端操作模式错误; 0x02: Master端无效读命令错误; 0x04: Master端无效写命令错误; 0x08: Master端读超时错误; 0x10: Master读数据CRC校验错误; 0x20: Slave端接收写命令丢弃; 0x40: Slave端接收读命令CRC校验错误; 0x80: Slave端接收写命令CRC校验错误; 0x81: slave模式axi总线错误反馈; 0x82: slave模式axi总线写超时; 0x83: slave模式axi总线读超时; 0x84: slave模式, 包长错误; 0x85: master模式, 读FIFO溢出错误。	-

表 7-5 ESPI_FAULT_ADDR 寄存器

Bits	寄存器说明	配置值
ESPI_FAULT_ADDR: 偏移地址0x020		
[31:0]	ESPI错误地址寄存器。	-

表 7-6 ESPI_SLV_CRC_CNT 寄存器

Bits	寄存器说明	配置值
ESPI_SLV_CRC_CNT: 偏移地址0x02C		
[15:0]	CRC Slave端校验错误计数器。	-

表 7-7 ESPI_SLV_WR_DISCARD_CNT 寄存器

Bits	寄存器说明	配置值
ESPI_SLV_WR_DISCARD_CNT: 偏移地址0x034		



Bits	寄存器说明	配置值
[15:0]	SLAVE端写命令丢失计数器。	-

7.6.2 操作流程

通过ESPI总线访问ESPI维测寄存器，可以读取到ESPI总线的工作模式，CRC使能如表7-2；还可以读取到ESPI工作状态：ESPI忙闲，4/8线模式如表7-3；

当检测到ESPI错误中断时，可以通过ESPI总线访问维测寄存器。可以读取到ESPI的错误类型如表7-4；还可以查询到出错的访问地址表7-5；如果是产生的CRC校验错误可以访问CRC slave端校验错误计数器查询产生的CRC错误次数，如表7-6；如果是产生的写命令丢失错误，则可以查询丢失命令次数寄存器，表7-7。

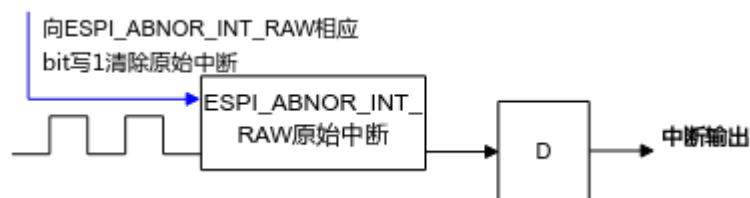
7.7 中断处理

7.7.1 中断描述

当ESPI检测到错误的（slave状态机异常、slave异步FIFO溢出、模式错误中断、包长错误、CRC错误、写命令丢失错误、AXI总线超时错误、AXI总线反馈错误、master读命令超时中断、master读操作crc错误中断、master读时rx_fifo溢出中断）时，上报中断。需要说明的是对于模式错误中断、包长错误中断、CRC错误中断、写命令丢失错误中断、AXI总线超时错误中断、AXI总线反馈错误中断、master读命令超时中断、master读操作crc错误中断、master读时rx_fifo溢出中断将合并成一个中断espi_intr上报，而slave状态机异常中断和slave异步FIFO溢出中断可以合并成一个中断espi_fsm_fifo_err_intr上报，也可以分开单独上报，分别对应管脚信号espi_fsm_err_intr和espi_fifo_err_intr，同时模块提供管脚信号espi_rxfifo_full_status，slv_fsm_err_st[2:0]可检测接收fifo满状态和espi slave内部状态机状态。

中断可以被查询和清除，如图7-2所示。

图 7-2 ESPI 中断处理框图



7.7.2 处理流程

ESPI检测到错误后会上报错误中断，软件检测到错误中断后可以通过直接复位芯片的方式进行恢复；也可以查询错误类型，对于可以恢复的错误（包长错误、CRC错误、AXI总线反馈错误）可以重新进行数据发送，其他错误类型需要进行整芯片复位。



7.7.2.1 直接复位芯片流程

步骤1 ESPI_INT_MASK寄存器配置需要屏蔽的中断源，例如将所有的错误中断屏蔽关闭，如表7-8所示。

表 7-8 ESPI_INT_MASK 中断屏蔽寄存器

bit	寄存器描述	配置值
[31:13]	保留	0x0
[12]	slave状态机异常中断。 0：不屏蔽； 1：屏蔽。 注意：当该屏蔽位生效时，ESPI不会发出异常复位请求信号req_out。	由场景决定。
[11]	slave异步FIFO溢出中断。 0：不屏蔽； 1：屏蔽。 注意：当该屏蔽位生效时，ESPI不会发出异常复位请求信号req_out。	由场景决定。
[10]	包长错误中断。 0：不屏蔽； 1：屏蔽	由场景决定。
[9]	slave axi总线超时中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[8]	slave axi错误反馈中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[7]	slave写crc错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[6]	slave读crc错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[5]	slave写命令丢失中断。 0：不屏蔽； 1：屏蔽	由场景决定。



bit	寄存器描述	配置值
[4]	master读crc错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[3]	master读超时错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[2]	Master模式下无效写命令原始中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[1]	Master模式下无效读命令原始中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[0]	模式错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。

步骤2 检测到ESPI上报错误中断，查询中断状态寄存器，识别不可恢复中断时对整芯片进行复位，中断状态寄存器如表7-9所示。

表 7-9 ESPI_ABNOR_INTR_STA 寄存器

bit	寄存器描述	配置值
[31:13]	保留。	0x0
[12]	slave状态机异常中断。 0：无中断； 1：有中断。	-
[11]	slave异步FIFO溢出中断。 0：无中断； 1：有中断。	-
[10]	包长错误中断。 0：无中断； 1：有中断。	-
[9]	slave axi总线超时中断。 0：无中断； 1：有中断。	-



bit	寄存器描述	配置值
[8]	slave axi错误反馈中断。 0：无中断； 1：有中断。	-
[7]	slave写crc错误中断。 0：无中断； 1：有中断。	-
[6]	slave读crc错误中断。 0：无中断； 1：有中断。	-
[5]	slave写命令丢失中断。 0：无中断； 1：有中断。	-
[4]	master读crc错误中断。 0：无中断； 1：有中断。	-
[3]	master读超时错误中断。 0：无中断； 1：有中断。	-
[2]	Master模式下无效写命令 原始中断。 0：无中断； 1：有中断。	-
[1]	Master模式下无效读命令 原始中断。 0：无中断； 1：有中断。	-
[0]	模式错误中断。 0：无中断； 1：有中断。	-

步骤3 配置ESPI_CTRL寄存器，选择相应的工作模式，如表7-10所示。

表 7-10 ESPI_CTRL 寄存器

bit	寄存器描述	配置值
[31:10]	保留。	0x0



bit	寄存器描述	配置值
[9:8]	ESPI Master模式输入外部插拍情况下内部采样时刻选择。 00: 外部不插拍情况下的采样时刻点; 01: 外部插入1拍情况下的采样时刻点; 10: 外部插入2拍情况下的采样时刻点; 11: 外部插入3拍情况下的采样时刻点。	无需配置。
[7]	Master端数据采样沿延迟选择。 0: 延迟2个ack周期采样; 1: 延迟1个ack周期采样。	无需配置。
[6]	Master端数据发送沿选择。 0: 上升沿发送; 1: 下降沿发送。	无需配置。
[5]	AXI总线错误响应使能。 0: 禁止; 1: 使能。	由场景决定。
[4]	CRC校验使能。 0: 禁止; 1: 使能。	由场景决定。
[3]	Master端采样沿选择。 0: 下降沿采样; 1: 上升沿采样。	无需配置。
[2]	ESPI接口位选择 0: 4bit; 1: 8bit。	0x0
[1:0]	ESPI工作模式。 00: 从模式; 01: 主模式; 10: 环回模式; 11: 保留。	0x0



步骤4 重复步骤1操作，配置相应的屏蔽位。

----结束

7.7.2.2 可以恢复的错误类型处理流程

步骤1 ESPI_ABNOR_IMSK寄存器配置需要屏蔽的中断源，例如将所有的错误中断屏蔽关闭，如表7-11所示。

表 7-11 ESPI_ABNOR_IMSK 中断屏蔽寄存器

bit	寄存器描述	配置值
[31:13]	保留。	0x0
[12]	slave状态机异常中断。 0：不屏蔽； 1：屏蔽。 注意：当该屏蔽位生效时，ESPI不会发出异常复位请求信号req_out。	由场景决定。
[11]	slave异步FIFO溢出中断。 0：不屏蔽； 1：屏蔽。 注意：当该屏蔽位生效时，ESPI不会发出异常复位请求信号req_out。	由场景决定。
[10]	包长错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[9]	slave axi总线超时中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[8]	slave axi错误反馈中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[7]	slave写crc错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[6]	slave读crc错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。



bit	寄存器描述	配置值
[5]	slave写命令丢失中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[4]	master读crc错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[3]	master读超时错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[2]	Master模式下无效写命令原始中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[1]	Master模式下无效读命令原始中断。 0：不屏蔽； 1：屏蔽。	由场景决定。
[0]	模式错误中断。 0：不屏蔽； 1：屏蔽。	由场景决定。

步骤2 检测到ESPI上报错误中断，查询中断状态寄存器表7-12所示。

表 7-12 ESPI_ABNOR_INTR_STAT 寄存器

bit	寄存器描述	配置值
[31:13]	保留。	0x0
[12]	slave状态机异常中断。 0：无中断； 1：有中断。	-
[11]	slave异步FIFO溢出中断。 0：无中断； 1：有中断。	-
[10]	包长错误中断。 0：无中断； 1：有中断。	-



bit	寄存器描述	配置值
[9]	slave axi总线超时中断。 0：无中断； 1：有中断。	-
[8]	slave axi错误反馈中断。 0：无中断； 1：有中断。	-
[7]	slave写crc错误中断。 0：无中断； 1：有中断。	-
[6]	slave读crc错误中断。 0：无中断； 1：有中断。	-
[5]	slave写命令丢失中断。 0：无中断； 1：有中断。	-
[4]	master读crc错误中断。 0：无中断； 1：有中断。	-
[3]	master读超时错误中断。 0：无中断； 1：有中断。	-
[2]	Master模式下无效写命令 原始中断。 0：无中断； 1：有中断。	-
[1]	Master模式下无效读命令 原始中断。 0：无中断； 1：有中断。	-
[0]	模式错误中断。 0：无中断； 1：有中断。	-

步骤3 对于可以恢复的错误（包长错误、CRC错误、AXI总线反馈错误）可以查询产生错误的地址，如表7-13所示。



表 7-13 ESPI_FAULT_ADDR 寄存器

Bits	寄存器说明	配置值
ESPI_FAULT_ADDR: 偏移地址0x024		
[31:0]	ESPI错误地址寄存器。	-

步骤4 清除错误中断，如表7-14所示。

表 7-14 ESPI_ABNOR_INTR_RAW 中断原始状态寄存器

Bits	寄存器说明	配置值
ESPI_ABNOR_INTR_RAW: 偏移地址0x008		
[13]	master 模式，读操作由于 AXI总线响应不及时，导致 ESPI读FIFO溢出，产生该异常中断。 读： 0：无中断； 1：有中断。 注意：该中断不能写清，仅在复位时可清除。	0b1
[12]	slave状态机异常中断。 当slave检测到状态机出现异常（一直处于某一状态，且超过配置的超时时间时）产生该异常中断。 读： 0：无中断； 1：有中断。 注意：该中断不能写清，仅在复位时可清除。	0b1
[11]	slave异步FIFO溢出中断。 读： 0：无中断； 1：有中断。 注意：该中断不能写清，仅在复位时可清除。	0b1



Bits	寄存器说明	配置值
[10]	<p>espi_slave模式时，当espi总线发送的数据长度和opcode中的length指示不一致（数据包长过长，数据包长过短），上报包长错误中断。</p> <p>0：无中断； 1：有中断。</p> <p>写： 0：无效。 1：清除此中断。</p> <p>注意：slave仅检测包长和length指示不匹配的包长错误。</p>	0b1
[9]	<p>espi_slave模式下产生axi总线超时错误中断。</p> <p>0：无中断； 1：有中断。</p> <p>写： 0：无效； 1：清除此中断。</p> <p>注意：仅在slave模式下产生写数据响应和读数据响应超时该指示信号有效。</p>	0b1
[8]	<p>espi_slave模式下检测到axi总线错误响应中断。</p> <p>0：无中断； 1：有中断。</p> <p>写： 0：无效； 1：清除此中断。</p> <p>注意：仅在slave模式下检测错误的写或者读响应该指示信号有效。</p>	0b1



Bits	寄存器说明	配置值
[7]	espi_slave接收写数据时产生crc校验错误中断。 读： 0：无中断； 1：有中断。 写： 0：无效。 1：清除此中断。 注意：仅在slave接收数据产生crc错误该指示有效。	0b1
[6]	espi_slave发送读数据时产生crc校验错误中断。 读： 0：无中断； 1：有中断。 写： 0：无效。 1：清除此中断。 注意：仅在slave发送读数据产生crc错误该指示有效。	0b1
[5]	Slave模式下接收写命令丢弃原始中断。 读： 0：无中断； 1：有中断。 写： 0：无效 1：清除此中断。 注意：仅在espi_slave模式下，当axi_master接收写命令溢出。	0b1



Bits	寄存器说明	配置值
[4]	espi_master接收读数据时产生crc校验错误中断。 读： 0：无中断； 1：有中断。 写： 0：无效。 1：清除此中断。 注意：仅在master接收读数据产生crc错误该指示有效。	0b1
[3]	espi_master模式下产生读超时错误中断。 0：无中断； 1：有中断。 写： 0：无效； 1：清除此中断。 注意：仅在master模式发送等待读数据返回超时该指示信号有效。	0b1
[2]	Master模式下无效写命令原始中断。 0：无中断； 1：有中断。 注意：仅在master模式有效，指示master模式下接收到axi总线传递的无效写传输。	0b1
[1]	Master模式下无效读命令原始中断。 0：无中断； 1：有中断。 注意：仅在master模式有效，指示master模式下接收到axi总线传递的无效读传输。	0b1



Bits	寄存器说明	配置值
[0]	<p>当宏定义同时支持 espi_master和espi_slave 模式时，模式寄存器配置为espi_slave模式，但是进行master模式的axi命令/数据操作。</p> <p>读： 0：无中断； 1：有中断。</p> <p>写： 0：无效。 1：清除此中断。</p> <p>注意：仅在master和slave模式宏定义均有效，在slave模式下进行master模式的命令收发该指示信号有效。</p>	0b1

步骤5 从错误地址开始重新进行MEM的写访问或者读访问。

----结束

7.8 性能分析

7.8.1 性能分析

ESPI_CLK最快可以达到工作时钟频率；介于硬件实现的条件，需要控制在50Mhz以下。以50MHz作为串行时钟为例：

4bit数据线配置下:典型传输为8bit opcode、32bit addr、32bit data，除去地址等开销有效速率可达到80Mbps。使用burst传输能再有效提高数据传输效率，理论最大值为185Mbps。

8bit数据线配置下：典型传输为8bit opcode、32bit addr、32bit data，除去地址等开销有效速率可达到160Mbps。使用最大的burst，能大大提高总线有效利用率 $16 \times 64 / (16 \times 64 + 32 + 8) = 98\%$ 。

须知

- ESPI MASTER端配置的ESPI_CLK随路时钟频率不应大于ESPI_SLAVE端的工作时钟频率，否则会导致异步FIFO溢出引起ESPI传输异常。
- 由于整个ESPI总线传输无反压，ESPI MASTER端ESPI_CLK的配置还需要考虑ESPI_SLAVE端反压的情况，避免导致ESPI传输异常。



7.8.2 接口时序要求

无总线反压情况下，ESPI写操作一共有2个ESPI_CLK延迟+3个工作始终延迟。

7.8.2.1 ESPI_Slave 写接口时序

图 7-3 ESPI 写接口时序 8 线模式

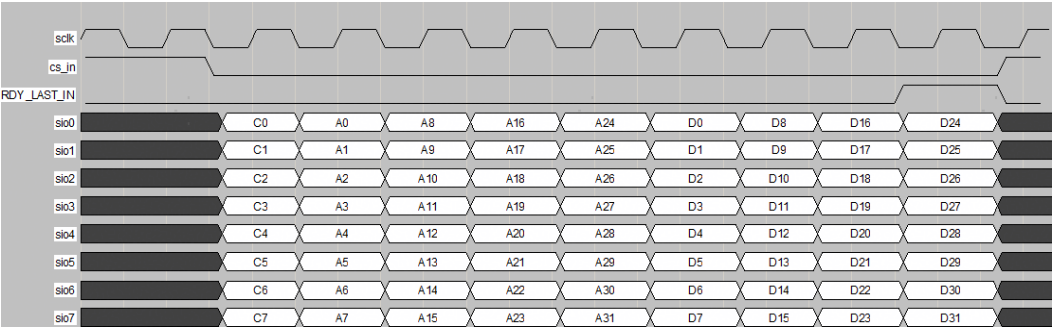


图 7-4 ESPI 读接口时序 8 线模式

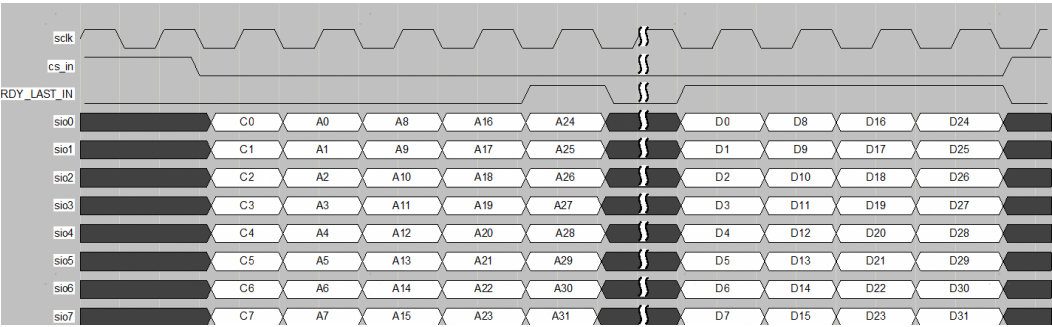


图 7-5 ESPI 写接口时序 4 线模式

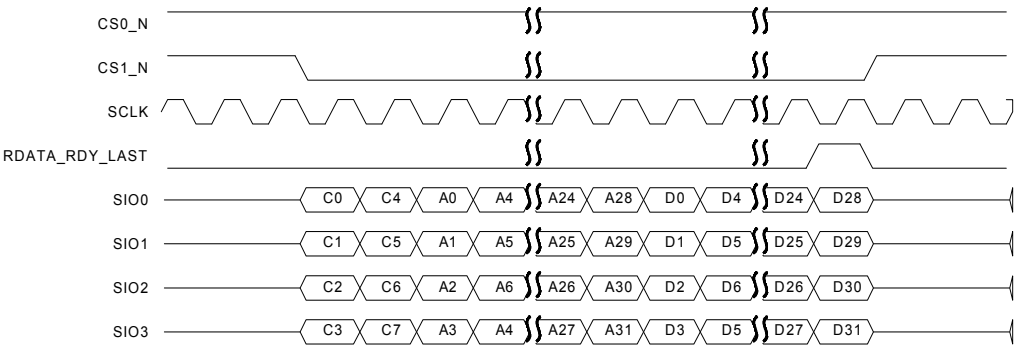
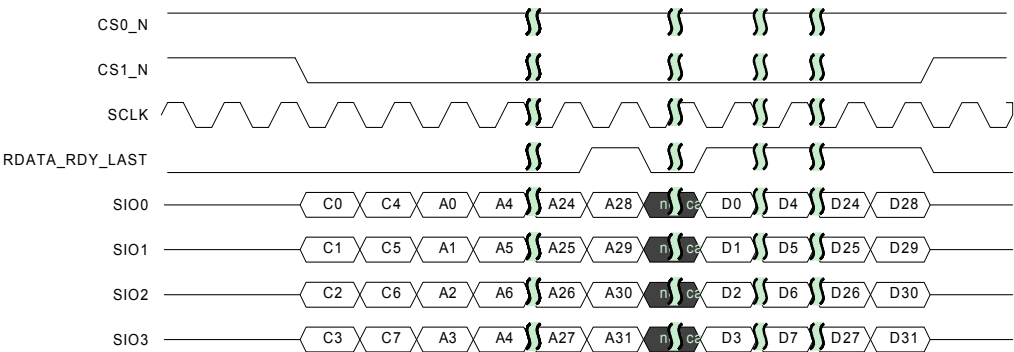




图 7-6 ESPI 读接口时序 4 线模式



7.9 使用约束

对于ESPI Slave，aclk时钟频率必须高于spi随路时钟的频率（espi_tx_sclk_in）。

ESPI Master和ESPI Slave在设计架构上，约定一旦开始一个beat数据的传输（即1个axsize的数据）后，在该beat数据传输完成之前，中途不会冒泡。

但是，在ESPI Slave中，发送数据时，数据从异步FIFO中弹出，异步FIFO中，CBB_CDC_L2L的仿真模型模拟亚稳态，数据输出会随机选择前一排或者后一拍数据（如图7-7），会导格雷码传递延迟（前一排采样async_in,下一拍采样U_ASYNC_1FF，格雷码多保持一拍，下一个格雷码传递延迟），进一步导致异步FIFO不完全流水，使ESPI Slave打破该约定，最终导致ESPI采样到冒泡数据（ESPI Master一旦开始采样一个Beat数据，中途不看rdy信号），CRC检测错误。

图 7-7 U_ASYNC_2FF 参考图

```
U_ASYNC_1FF[i]      <= {async_in[i]};
U_ASYNC_2FF[i]      <= cycle_sel ? U_ASYNC_1FF[i]:{async_in[i]};
U_ASYNC_3FF[i]      <= U_ASYNC_2FF[i];
```

总结一句话，当aclk和espi_tx_sclk_in同频时，异步FIFO不完全保证流水，导致传输异常。

所以，约束aclk频率必须高于espi_tx_sclk_in。

espi寄存器CTRL[9:8]读写采样延时espi_out_dly_sel不能配置为0。

由于309a espi外部新增输出方向打拍，导致数据指示rdy有时延，主片将输出的rdy误认为接收数据的rdy，从而导致主片未收完读数据提前结束，读数据错误/读超时。



8 HACSPI

8.1 概述

HACSPI模块实现将软件预先配置的SPI时序按照SPI规范输出的功能。同时完成识别外部时钟倒换信号触发SPI时序，完成外部时钟的快速倒换功能。

8.2 特性

HACSPI满足以下规格：

- OR.IO.0006：支持HAC SPI接口。
- 支持3组SPI总线，每组SPI各2片选。
- 支持3线制、4线制可配，默认4线制模式。
- 支持工作在主模式。
- SPI接口时钟速率需要支持可以灵活配置，最大50MHz，备选频点至少需要支持40MHz到50MHz、30MHz到40MHz、20MHz到30MHz（优先25MHz或略低于25MHz）、10MHz到20MHz及1MHz到10MHz这几档。考虑单组SPI 1驱动2时，SPI时钟速率可进一步降低。
- SPI接口各信号之间的相位可调整。
- 支持上下沿可配。
- SPI选用RAM方式实现的、可任意配置数据和时序的SPI IP。
- OR.IO.0007：外置PLL配置管理相关接口需求。
- 支持2组SPI口，每组SPI各2片选；支持3线制、4线制可配，默认4线制模式。
- SPI选用RAM方式实现的、可任意配置数据和时序的SPI IP。
- 支持逻辑配置时钟硬切换，控制过程中软件不介入。但切换控制的各个环节的各种参数保持足够的灵活度，需支持通过软件进行灵活配置（mask/切换信号映射可配）。
- 支持时钟切换管脚，内部通过HAC对SPI输出指定序列实现时钟切换（仅针对第0组端口）。



8.3 关键术语

表 8-1 SPI 关键术语描述

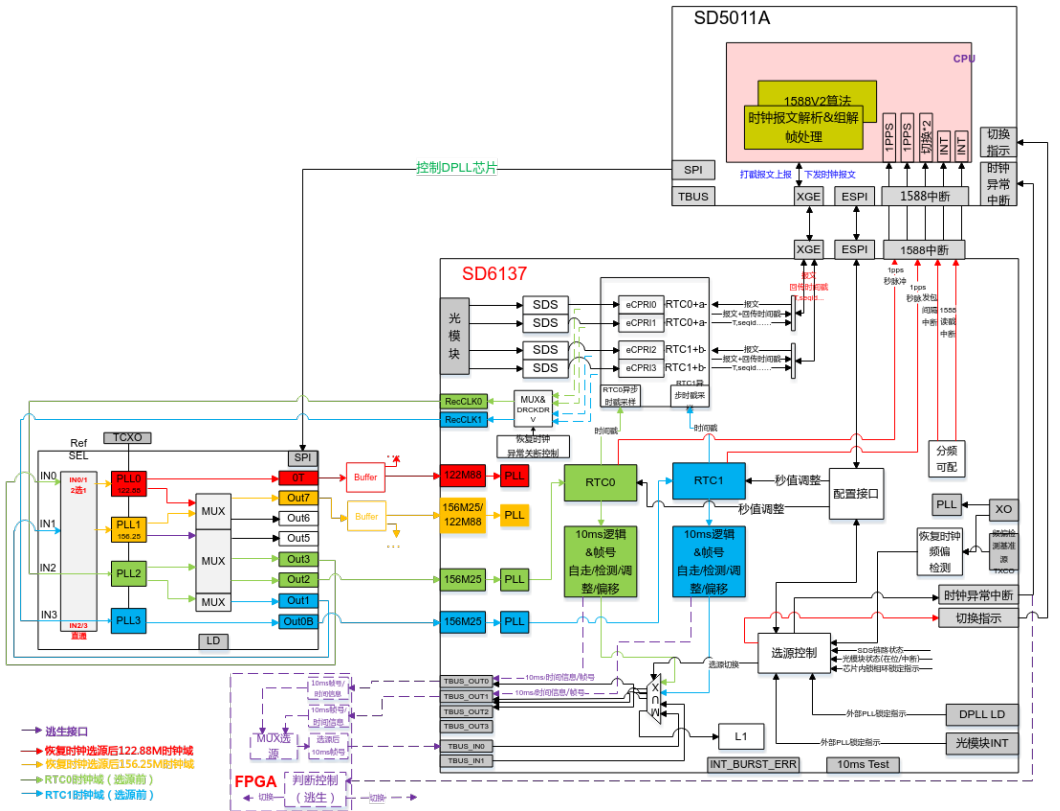
术语	描述
SPI	SPI (Serial Peripheral Interface) 控制器，可以作为一个主设备与外部的设备来进行同步串行通信。

8.4 功能描述

8.4.1 应用场景

HACSPI应用框如图8-1所示。

图 8-1 HACSPI 应用框图



8.4.2 功能原理

每个HACSPI口提供两组SPI接口的CS信号，一组SPICK&SPIDI/SPIDO信号。



用户根据对接SPI设备的SPI时序，将数据写入内置RAM中，芯片在接收到时序触发信号之后，完成CPU操作触发或是外部时钟切换管脚触发的判断，按照用户配置驱动RAM中存储的时序通过PIN发出去。

外部输入的时钟切换管脚，支持是否滤毛刺配置（10ns）。

8.5 工作流程

8.5.1 管脚复用配置

HACSPI管脚与其他功能管脚复用，需要完成IOMUX配置（寄存器 IOMG069~IOMG072、IOMG075、IOMG119~IOMG122、IOMG130、IOMG143~IOMG146、IOMG154、IOMG291~IOMG294、IOMG302、IOMG332~IOMG336、IOMG368~IOMG372），三线/四线模式也需要通过PERI SUBCTRL中sc_hac_spi_mod配置。在使用HACSPI功能之前，需要先完成IO相关配置。

8.5.2 HACSPI 初始化配置

具体配置步骤如下：

- 步骤1 配置“HACSPI功能解复位”（ETH_SUBCTRL中SRST_DREQ_HAC_SPI寄存器为0xF），等待10μs后查看解复位状态（ETH_SUBCTRL中SRST_ST_HACSPI寄存器值为x0），若处于解复位状态则继续下一步，否则继续读取，多次读取不为0则视为异常。
- 步骤2 配置“SPI_RAM初始化”（RAM_INIT寄存器）。
- 步骤3 配置“SPI-数据RAM和SPI-参数RAM”，完成对接设备SPI时序和发送数据的配置。数据RAM（DRAM）和参数RAM（PRAM）的结构如下：

图 8-2 数据 RAM 结构

dram偏移	byte3	byte2	byte1	byte0	
0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	CPU (ram_sel = 0)
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
19	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
1*20+0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	CPU (ram_sel = 1)
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
1*20+19	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
2*20+0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	Logic (逻辑自行处理)
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
2*20+47	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	



图 8-3 参数 RAM 结构

pram偏移	oen	clk	cs	
0	xxxxxxxx	xxxxxxxx	xxxxxxxx	CPU
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	
79	xxxxxxxx	xxxxxxxx	xxxxxxxx	Logic
80	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	
.	xxxxxxxx	xxxxxxxx	xxxxxxxx	
271	xxxxxxxx	xxxxxxxx	xxxxxxxx	

步骤4 配置“SPI-IP参数”，IP参数包括：

1. 无效电平配置（SPI_CLK_UNVLD）。
2. 初始电平配置（SPI_CLK_LEVEL）。
3. CS无效电平配置（SPI_CS_UNVLD）。
4. 时序频率配置（SPI_REQ_FREQ_DIV）。
5. 发送相位配置（SPI_REQ_CFG_PHA）。
6. 片选信号配置（SPI_REQ_CS_SEL）。
7. 回读指示配置（SPI_REQ_RDB_IND）。
8. RAM时序选择（SPI_REQ_RAM_SEL）。
9. RAM时序地址和数据长度配置（SPI_REQ_ADDR_x和SPI_REQ_RD_LEN_x，x代表有CPU触发和外部时钟管脚切换触发两套配置）。

步骤5 若有外部时钟切换需求，还需要完成如下配置：

1. 切换状态的灵活映射（SPI_IP_LOGIC_OP_x）。
2. 外部触发采样配置（SPI_CHG_IND_FILTER_EN&SPI_CHG_IND_FILTER_CNT&SPI_STABLE_SMP_CNT）。
3. 为满足区分外部切换完成中断识别，配置回读指示寄存器SPI_REQ_RDB_IND为1。

步骤6 外部时钟时序请求MASK配置：（SPI_IP_LOGIC_REQ_MASK&SPI_IP_LOGIC_MASK）。

步骤7 如果需要CPU触发，则对start寄存器写1即完成一次时序触发（SPI_IP_START）。

----结束

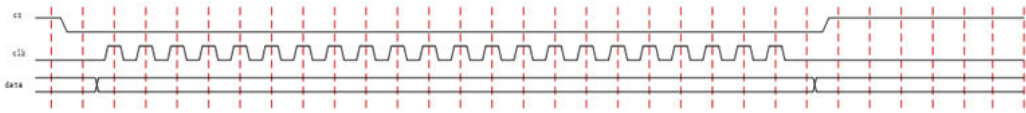


说明

SPI RAM 参数配置规则

对于 CS 为低电平有效的器件

RAM 中数据存放时序以器件手册上的 CS 和 clk 为主，进行数据划分，如下图所示。



上图红色虚线表示采样时刻点，要求：

Cs: 电平每次变化前后需要采样到

Clk: 有效电平都要采样到（只用于告诉 SPI 当前的 clk 是否要输出）

Data: 时钟无效时，数据均是无效值；时钟有效时，才将对应 bit 填充

所以，以上图为例。

Cs 的配置值为 8'b10000000_8'b00000000_8'b00000000_8'b01111111

Clk 的配置值为 8'b00111111_8'b11111111_8'b11111111_8'b00000000

(上述在RAM中的bit是7~0, 15~8...)

8.5.3 HACSPI 其他

8.5.3.1 回读

- 1. 配置回读触发相关配置：回读指示/回读相位/回读延时。
- 2. 同上一章节的CPU触发操作。

8.5.3.2 超时告警

- 1. 配置超时门限寄存器。
- 2. 若触发操作后，在门限寄存器配置的时间内SPI操作没有完成，则会触发超时告警（中断）。

8.5.3.3 时序操作完成指示

一次操作完成后，会产生时序操作完成中断：SPI_REQ_SEND_END。

8.5.3.4 动态操作

- 1. 不允许DRAM和PRAM的动态改配。
- 2. 如果开启硬件切换外部时钟功能，不允许对所有和硬件切换相关寄存器的动态配置。
- 3. 如果需要配置，可以将所有时序触发动作MASK掉，重新初始化HACSPI。

8.5.3.5 CS 片选切换

Hi309a每个HACSPI有2个片选，访问某个片选对接器件，需要配置SP_REQ_CS_SEL。如访问CS0，配置为0x1；访问CS1，配置为0x2。

图 8-4 SP_REQ_CS_SEL 寄存器说明

SPI_REQ_CS_SEL	SPI_REQ_CS_SEL为每请求CS信号有效使能配置寄存器。	0x009c + 0x3000*n	32	RW	0x00000000	reserved	31:2	-	0x00000000	保留。
						spi_req_cs_sel	1:0	RW	0x0	SPI每请求CS信号有效使能。bit[0]到bit[1]对应IP输出的CS0到CS1。 每bit含义： 0: 对应CS无效； 1: 对应CS有效。





9 I2C

9.1 概述

I2C（The Inter-Integrated Circuit）控制器实现标准I2C主设备功能，兼容Philips I2C总线协议，可完成对I2C总线上的从设备的数据发送和接收。主要用于音视频A/D、D/A等外部I²C器件的控制。

9.2 特性

I2C控制器具有以下特点：

- 支持标准I2C总线协议。
- 支持主设备或从设备操作。
- 支持7bit和10bit从设备地址。
- 支持可编程时钟，可实现不同通讯速率（100K/400K/1M）。
- 支持以中断方式和查询方式进行数据搬运。
- I2C_DELAY功能可将I2C_SDA信号输出延迟I2Cx_SDA_HOLD个I2C的工作周期。
- 支持CLK_STRETCHING功能（即用时钟同步机制作为握手）。
- 支持接口超时保护，防挂死处理。
- 每个I2C接口支持寻址8个device。
- 支持读写各256深的FIFO。
- TX FIFO发送为空时，控制器HOLD SCL等待新数据写入控制器。
- 支持SCL和SDA的滤波配置。

9.3 关键术语

如表9-1的术语将在本章节表示特殊的含义。



表 9-1 I2C 关键术语描述

术语	描述
I2C	Inter-Integrated Circuit, I2C总线是由PHILIPS公司开发的两线式串行总线，用于连接微控制器及其外围设备。是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。

9.4 遵循协议

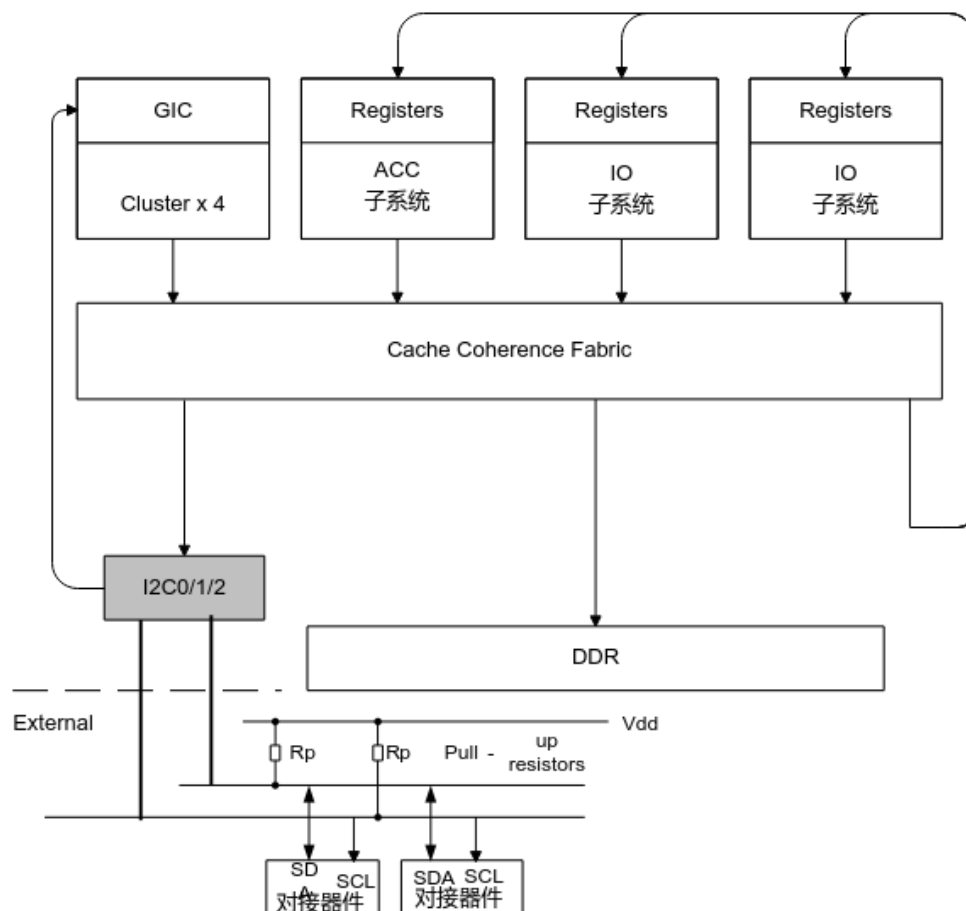
该接口模块遵循标准I2C总线协议，同时兼容Philips I2C总线协议。

9.5 功能描述

9.5.1 应用场景

I2C应用框图如图9-1所示。

图 9-1 I2C 应用框图



I2C 总线是一种双线、双向串行总线，它提供了一种简单有效的数据传输方式，可以简化设备间的连接，适合多设备间的短距离少量数据传输。I2C 总线的灵活性使得系统开发和设备扩展变得很容易。

CPU 可通过总线配置 I2C 的寄存器配置数据传输到对端，或者读取寄存器获取接收到的对端数据；I2C 在传输完成或传输发生错误时，可以通过中断通知 CPU 进行处理。

9.5.2 相关模块

在系统中，I2C 模块会和下面多个模块进行交互，分别描述如下：

- **CPU：** CPU 通过总线访问 I2C 的寄存器接口，操作属性必须为 Device 或 Strongly-Ordered。CPU 首先需要完成 I2C 的传输模式和速率配置，对于少量或即时性要求高的数据，CPU 可以直接通过 FIFO 接口读写完成数据传输。
- **GIC：** I2C 在传输时如果发生错误，例如传输冲突等，以及在非 DMAC 搬移模式时，如果接收 FIFO 高于水线，或发送 FIFO 低于水线，均会产生中断信号给 GIC 模块，通过 GIC 将中断发送给对 I2C 具有控制权的 CPU。



9.5.3 数据结构

一次典型的主模式I2C数据传输主要包括起始信号、从地址发送、数据传送、结束信号等部分。其时序如图9-2所示，数据帧格式如图9-3所示。

图 9-2 I2C 数据传输时序图

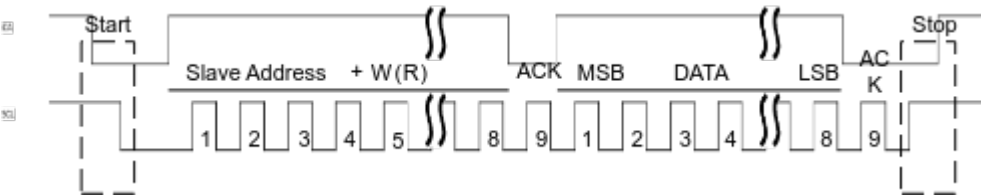
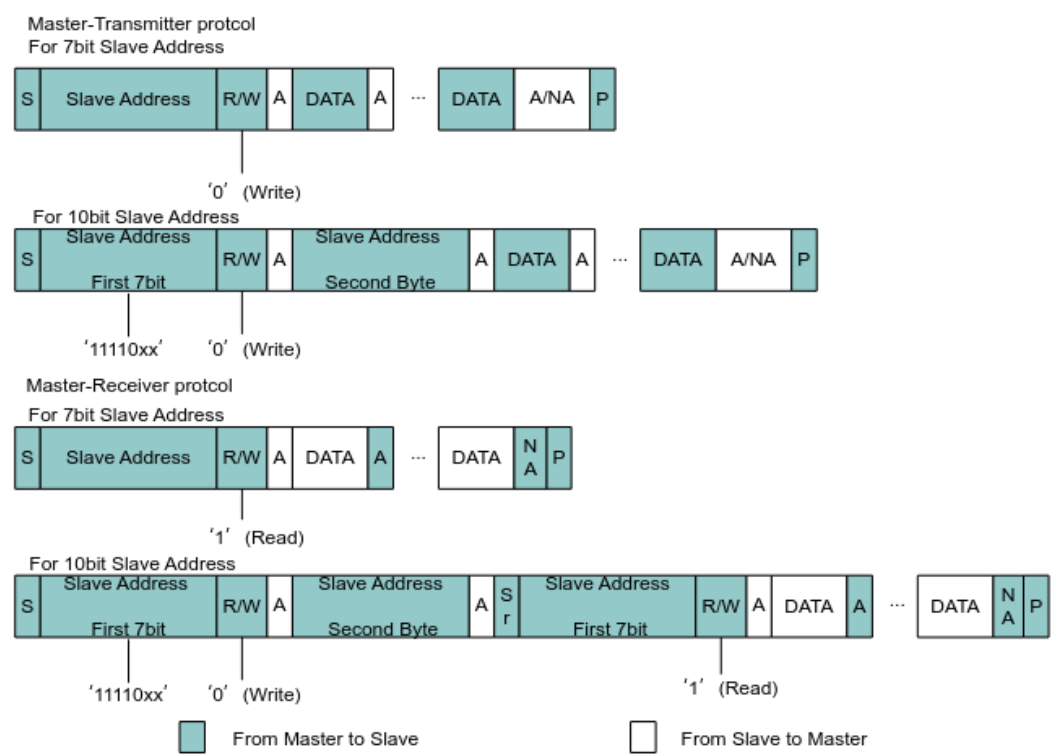


图 9-3 I2C 数据传输帧格式图



说明

- 图9-3中部分缩略语含义如下：
- A: Acknowledge（SDA Low）。
 - NA: No Acknowledge（SDA High）。
 - S: Start Condition。
 - Sr: Repeated Start Condition。
 - P: Stop Condition。

I2C标准协议规定的起始信号（Start），由总线上的主设备发出，用来唤醒所有从设备并指示数据传送开始的特殊信号。



从地址发送（Slave Address + W/R）是指主设备在发出起始信号后所发出的第一个数据包，包括7bit从地址位（如果是10bit从地址，则需发送两个数据包）和1bit读/写数据。各从设备依靠该地址信息来确认是否需要传送或接收数据。当从地址被成功接收后，相应从设备会在第9个时钟周期（SCL）将SDA拉低，从而反馈给主设备一个应答信号（ACK），然后就可以开始进行后续的数据传输。

数据传输（DATA）是指主设备在成功接收到从地址响应信号后按照读、写命令进行相应的数据接收或发送。在数据传输过程中SDA只能在SCL为低电平时变化，而在SCL高电平时保持；同时接收设备在每接收到1个字节后需要给发送设备发出应答信号（ACK）。如果发送设备没有成功接收到应答信号，就会终止数据传输或者重新开始发送。

结束信号（Stop）是主设备在当前数据传输完成并且没有新的数据传输需要发起时，所发出的I2C标准协议规定的表示数据传输结束的特殊信号。当主设备发出结束信号后从设备必须释放总线。

9.6 初始化流程

9.6.1 初始化

9.6.1.1 I2C 从模式复位后初始化

- 步骤1** 设定超时标志，检测IC_STATUS [activity]（判断I2C控制器是否空闲），如果是1，则延时等待，继续检测，直到IC_STATUS[activity]变为0或者超时，向IC_ENABLE bit[0]写0，使I2C处于关闭状态。
- 步骤2** 向IC_ENABLE[enable]写0，禁止I2C。
- 步骤3** 写相应的配置值到IC_CON，配置传输速率模式等参数。
- 步骤4** 将对接器件地址写入IC_SAR寄存器。
- 步骤5** 配置IC_SS_SCL_HCNT、IC_SS_SCL_LCNT等寄存器，设定相应的I2C总线时钟周期。
- 步骤6** 配置IC_RX_TL、IC_TX_TL，设定相应的发送及接收FIFO阈值。
- 步骤7** 如果驱动程序采用中断方式则需设定IC_INTR_MASK，使能相应中断信号；采用查询方式时应禁止产生相应中断信号。
- 步骤8** 向IC_ENABLE[enable]写1，使能I2C，完成初始化配置。

----结束

9.6.1.2 I2C 主模式复位后初始化

- 步骤1** 向IC_ENABLE[enable]写0，禁止I2C。
- 步骤2** 配置IC_CON寄存器，设置支持的最大速度模式。
- 步骤3** 配置IC_TAR寄存器。
- 步骤4** 向IC_ENABLE[enable]写1，使能I2C。



步骤5 将数据或命令写入IC_DATA_CMD寄存器；如果在IC_ENABLE使能之前写入数据或命令到IC_DATA_CMD寄存器，这些数据将会丢失。

----结束

9.6.2 时钟门控

上电时I2C控制器工作时钟默认为打开状态。在软件完成当前数据传输且未启动新的数据传输的情况下可关断I2C工作时钟，但需要确保硬件已处于空闲状态（即I2C_STATUS[Activity]为0时）。

9.6.2.1 打开时钟

具体配置步骤如下：

步骤1 读PERI SUBCTRL寄存器SC_I2C_CLK_ST，获取当前时钟门控信息。

步骤2 若SC_I2C_CLK_ST [0]为0，则进入**步骤5**；SC_I2C_CLK_ST [0]为1，则表示I2C0时钟已经打开。

步骤3 若SC_I2C_CLK_ST [1]为0，则进入**步骤6**；SC_I2C_CLK_ST [1]为1，则表示I2C1时钟已经打开。

步骤4 若SC_I2C_CLK_ST [2]为0，则进入**步骤7**；SC_I2C_CLK_ST [2]为1，则表示I2C2时钟已经打开。

步骤5 配置SC_I2C_CLK_EN[0]为1，打开I2C0模块的时钟门控。

步骤6 配置SC_I2C_CLK_EN[1]为1，打开I2C1模块的时钟门控。

步骤7 配置SC_I2C_CLK_EN[2]为1，打开I2C2模块的时钟门控。

----结束

9.6.2.1.1 关闭时钟

具体配置步骤如下：

步骤1 读I2C_STATUS，若I2C_STATUS[Activity]为0，则向I2C_ENABLE[enable]写0，进入**步骤2**；若I2C_STATUS[Activity]为1，则延时等待，返回**步骤1**。

步骤2 读PERI SUBCTRL寄存器SC_I2C_CLK_ST，获取当前时钟门控信息。

步骤3 若SC_I2C_CLK_ST [0]为1，则进入**步骤6**；SC_I2C_CLK_ST [0]为0，则表示I2C0时钟已经关闭。

步骤4 若SC_I2C_CLK_ST [1]为1，则进入**步骤7**；SC_I2C_CLK_ST [1]为0，则表示I2C1时钟已经关闭。

步骤5 若SC_I2C_CLK_ST [2]为1，则进入**步骤8**；SC_I2C_CLK_ST [2]为0，则表示I2C2时钟已经关闭。

步骤6 配置SC_I2C_CLK_DIS[0]为1，关闭I2C0模块的门控时钟。

步骤7 配置SC_I2C_CLK_DIS[1]为1，关闭I2C1模块的门控时钟。

步骤8 配置SC_I2C_CLK_DIS[2]为1，关闭I2C2模块的门控时钟。

----结束



9.6.3 软复位

对接器件容许的情况下，可以任意时间对I2C工作逻辑进行软复位，不会对系统造成影响。

- 通过配置PERI SUBCTRL寄存器SC_I2C_RESET_REQ相应位为0b1，可实现对I2C工作逻辑的单独软复位。
- 通过配置PERI SUBCTRL寄存器SC_I2C_RESET_DREQ的相应位为0b1，可实现对I2C工作逻辑的单独软复位撤离。

须知

I2C软复位不会将I2C本身寄存器恢复成默认值。

9.6.4 I2C disable 的流程

- 在软件完成当前数据传输且未启动新的数据传输的情况下可把I2C disable掉，但需要确保硬件已处于空闲状态（即I2C_STATUS[Activity]为0时）。读I2C_STATUS，若I2C_STATUS[Activity]为0，则向I2C_ENABLE[enable]写0 disable I2C。
- 按照I2C的初始化流程配置I2C，之后使能I2C进行工作。

9.7 工作方式

9.7.1 管脚复用配置

- I2C0的SDA0和SCL0管脚与GPIO复用，使用I2C0前，需要配置IOMG140（物理地址为0xFA8601BC）和IOMG141（物理地址为0xFA8601C0）为0，使能相应管脚的SDA0和SCL0功能。
- I2C1的SDA1和SCL1管脚与GPIO复用，使用I2C1前，需要配置IOMG142（物理地址为0xFA8601C4）和IOMG143（物理地址为0xFA8601C8）为0，使能相应管脚的SDA1和SCL1功能。
- I2C2的SDA2和SCL2管脚与GPIO复用，使用I2C2前，需要配置IOMG157（物理地址为0xFA8601FC）和IOMG156（物理地址为0xFA8601F8）为1，使能相应管脚的SDA2和SCL2功能。
- I2C3的SDA3和SCL3管脚与GPIO复用，使用I2C3前，需要配置IOMG159（物理地址为0xFA860204）和IOMG158（物理地址为0xFA860200）为1，使能相应管脚的SDA3和SCL3功能。
- I2C4的SDA4和SCL4管脚与GPIO复用，使用I2C4前，需要配置IOMG144（物理地址为0xFA8601CC）和IOMG145（物理地址为0xFA8601D0）为2，使能相应管脚的SDA4和SCL4功能。
- I2C5的SDA5和SCL5管脚与GPIO复用，使用I2C5前，需要配置IOMG146（物理地址为0xFA8601D4）和IOMG147（物理地址为0xFA8601D8）为1，使能相应管脚的SDA5和SCL5功能。
- I2C6的SDA6和SCL6管脚与GPIO复用，使用I2C6前，需要配置IOMG148（物理地址为0xFA8601DC）和IOMG149（物理地址为0xFA8601E0）为1，使能相应管脚的SDA6和SCL6功能。



- I2C7的SDA7和SCL7管脚与GPIO复用，使用I2C7前，需要配置IOMG150（物理地址为0xFA8601E4）和IOMG151（物理地址为0xFA8601E8）为1，使能相应管脚的SDA7和SCL7功能。

9.7.2 时钟配置

I2C Master模式下需要通过配置I2Cx_SS_SCL_HCNT、I2Cx_SS_SCL_LCNT、I2Cx_FS_SCL_HCNT、I2Cx_FS_SCL_LCNT等寄存器可以设置标准模式和快速模式下I2C总线SCL信号的高低电平宽度，即相对于I2C工作时钟的周期数，其配置值的计算公式

分别为：

$$*_HCNT = T_{scl_h} \times F_{DC} - 7 - SPKLEN$$
$$*_LCNT = T_{scl_l} \times F_{DC} - 1$$

和

说明

- T_{scl_h} 为I2C工作时钟SCL的高电平宽度，单位为 μs 。
- T_{scl_l} 为I2C工作时钟SCL的低电平宽度，单位 μs 。
- F_{I2C} 为I2C工作时钟，单位MHz。
- I2C slave模式下无需配置I2C_SS_SCL_HCNT、I2C_SS_SCL_LCNT、I2C_FS_SCL_HCNT和I2C_FS_SCL_LCNT寄存器。
- SPKLEN通过寄存器IC_FS_SPKLEN配置。

I2C_SS_SCL_HCNT、I2C_SS_SCL_LCNT、I2C_FS_SCL_HCNT和I2C_FS_SCL_LCNT典型配置值如表9-2~表9-5所示。

表 9-2 I2C_SS_SCL_HCNT 典型配置值

I2C总线速率 (kbps)	I2C工作时钟 (MHz)	SCL高电平要求 最小宽度 (工作 时钟Cycle)	SCL高电平 建议宽度 (μs)	I2C_SS_SCL_HCNT
100	200	13	5	987

表 9-3 I2C_SS_SCL_LCNT 典型配置值

I2C总线速率 (kbps)	I2C工作时钟 (MHz)	SCL低电平要求 最小宽度 (工作 时钟Cycle)	SCL低电平 建议宽度 (μs)	I2C_SS_SCL_LCNT
100	200	9	5	998



表 9-4 I2C_FS_SCL_HCNT 典型配置值

I2C总线速率 (kbps)	I2C工作时钟 (MHz)	SCL高电平要求 最小宽度 (工作 时钟Cycle)	SCL高电平 建议宽度 (μ s)	I2C_FS_SCL_HCNT
400	200	13	2.5	237

表 9-5 I2C_FS_SCL_LCNT 典型配置值

I2C总线速率 (kbps)	I2C工作时钟 (MHz)	SCL高电平要求 最小宽度 (工作 时钟Cycle)	SCL高电平 建议宽度 (μ s)	I2C_FS_SCL_LCNT
400	200	9	2.5	249

9.7.3 I2Cx_DELAY

I2C控制器额外具有I2Cx_SDA_HOLD功能，该功能用以将I2Cx_SDA信号输出延迟I2Cx_SDA_HOLD个I2C的工作周期，以满足某些特定的时序需要。

9.8 工作流程

9.8.1 中断或查询方式下的数据传输

说明

I2Cx表示I2C0、I2C1、I2C2。

9.8.1.1 数据发送

具体配置步骤如下：

- 步骤1** 将发送数据写入I2C_DATA_CMD，启动数据发送。在每发送一帧数据后，需要发送STOP或者restart指令。
- 步骤2** 查询方式下，进行连续数据发送时通过读取I2C_STATUS和I2C_TXFLR检测发送FIFO状态；中断方式下，根据相应中断状态位检测。在数据传送完成之前，既要保证发送FIFO中的数据没有溢出（否则会造成数据丢失）。
- 步骤3** 通过检测I2C_STATUS [tfe]是否为1，判断I2C是否完成全部数据发送。

----结束

9.8.1.2 数据接收

具体配置步骤如下：

- 步骤1** 将读命令写入I2C_DATA_CMD，启动数据接收。

步骤2 进行连续数据接收时，首先要发送次数与接收数据数目相同的读命令到 I2C_DATA_CMD（例如要接收3个数据，则需发送3次读命令到 I2C_DATA_CMD 寄存器）。在数据接收完成之前，既要保证发送 FIFO 中的数据没有溢出，又保证发送 FIFO 非空（否则 I2C 会认为没有新的数据需要接收而结束），另外在连续数据接收过程中还需要检测接收 FIFO 的状态（如果采用中断方式，则根据相应中断状态位检测），避免接收 FIFO 溢出。在每发送一帧数据后，需要发送 STOP 或者 restart 指令。

步骤3 通过检测 I2C_STATUS[rff] 是否为 1，判断 I2C 是否完成全部数据接收。

----结束

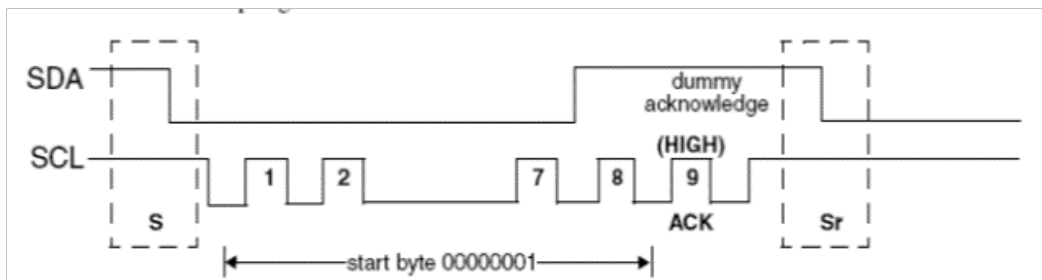
9.8.2 对端器件挂死的处理

9.8.2.1 异常挂死场景

异常挂死主要针对没有复位端的 I2C 对接器件，比如 I2C 接口的 EEPROM。

如果 I2C 接口正在访问的过程中，单板出现异常复位而正常重启，但是因为 EEPROM 没有复位端，没有进行复位，有可能导致 SDA 数据线一直拉低。如果此时需要继续访问 I2C 接口外接器件，由于 SDA 线拉低，将导致 I2C 总线发不出 START 操作（START 操作如图 9-4 所示），以至于 I2C 总线挂死。此时要想让 EEPROM 恢复正常，只能让器件重新上电。

图 9-4 I2C START 传输时序



说明

由于 SDA 线长拉低，导致发不出 START 操作，从而时钟线 SCL 也不会有跳变。如果能让 SCL 线主动进行翻转，将可以使 SDA 线进行 ACK 应答，恢复高，从而让挂死的 I2C 总线恢复正常。但是 I2C 模块本身不能直接控制 SCL 信号。所以采用一个 GPIO 或者系统控制器一个控制 bit 与 SCL 进行复用，来达到控制目的。但这需要软件主动参与这一过程，如果应用到启动场景，指令还未正确读回 I2C 总线就挂死，将导致整个系统需要重新上电，影响系统可靠性。

9.8.2.2 解决方案一

该解决方案需要逻辑和软件配合使用。I2C 模块不需做修改，利用管脚复用中 SCL 管脚与 GPIO 的复用即可解决该 I2C 防挂死功能。具体配置步骤如下：

步骤1 切换 SCL 和 SDA 管脚复用关系，设置 SCL 和 SDA 管脚电平由 GPIO 来控制配置。

步骤2 配置 SDA 管脚 GPIO 为输入模式，SCL 管脚 GPIO 为输出模式。

步骤3 通过配置 GPIO 发送高低电平来控制 SCL 时钟的翻转，同时程序插入等待，使得翻转频率为 10KHz，至少发送 9 个 SCL 时钟脉冲。



步骤4 切换SCL管脚复用关系，设置SCL管脚电平由I2C来控制配置。

步骤5 软复位I2C模块后初始化I2C，I2C即可正常工作。

----结束

9.8.2.3 解决方案二

该解决方案需要软件配置寄存器使用，I2C模块本身不需做修改，具体配置步骤如下：

步骤1 配置PERI SUBCTRL寄存器中的i2c*_clk_mux_sel_set和i2c*_dat_mux_sel_set为高，通过软件完全控制I2C SCL和SDA接口的管脚行为。

步骤2 配置PERI SUBCTRL寄存器中的i2c*_clk_oe_cfg_clr为高，配置SCL管脚为输出；配置PERI SUBCTRL寄存器中的i2c*_dat_oe_cfg_set为高，配置SDA管脚为输入。

步骤3 配置PERI SUBCTRL寄存器中的i2c*_scl_cfg_set为高，控制I2C SCL接口管脚输出高电平。

步骤4 配置PERI SUBCTRL寄存器中的i2c*_scl_cfg_clr为高，控制I2C SCL接口管脚输出低电平。

步骤5 重复**步骤3**和**步骤4**使得I2C管脚电平不停翻转，同时程序插入等待，使得翻转频率为10kHz，至少发送9个SCL时钟脉冲。

步骤6 配置PERI SUBCTRL寄存器中的i2c*_clk_mux_sel_clr和i2c*_dat_mux_sel_clr为高，设置I2C管脚电平由I2C控制器来控制配置。

步骤7 配置IC_ENABLE[abort]为高电平。

步骤8 等待tx_abrt中断。

步骤9 读IC_TX_ABRT_SOURCE[abrt_user_abrt]是否为高，如果为高I2C重新初始化后即可正常工作；否则软复位I2C模块后初始化I2C，I2C即可正常工作。

----结束

说明

I2Cx表示I2C0、I2C1、I2C2和I2C3。

9.8.3 滤波配置

I2C支持滤波配置，具体可以参考PEIR_SUBCTRL的0x2400和0x2404寄存器描述，滤波支持软件关闭开启。

9.9 注意事项

当I2C作为master时，如果slave要求每次读写操作之前必须要发送slave addr（因为有些slave需要根据固定的数据结构来解析，目前SD6602作为slave就有这个要求）的时候，在每次读写操作之前要判断I2C是不是处于空闲状态（即I2C_STATUS[Activity]为0时）。



10 RTC

本段落主要是对网络RTC的功能和应用做详细描述。

10.1 概述

RTC模块支持通过I2C接口和1PPS_TOD单线时间接口，解析得到秒脉冲和80bit时间戳，Hi309a芯片网络RTC和时钟板之间配合完成1588时间同步功能。在每个秒脉冲1pps（Pulse Per Second）到来时，时钟板下发标准的80bit时间戳信息给芯片。时间戳信息写入Hi309a芯片网络RTC寄存器，Hi309a芯片网络RTC内部将每秒的时间戳信息同步到所有时间戳应用模块。在两个秒脉冲之间，则通过输入到芯片的高精度时钟在RTC模块计时，时钟频率500MHz。这个时钟来自外部的和时钟板同步的时钟源，然后通过内部的PTP PLL倍频。上送时钟处理模块的T1/T2/T1CF/T3/T4/T3CF时间戳由NP下发给RTC，然后通过I2C接口被时钟处理模块Hi309a芯片外部的8006芯片定时读取。

Hi309a芯片网络RTC支持网络RTC的时间戳和无线主控传输用的时间戳静态二选一的功能，将同步标识和64bit时间戳串行输出给各个需要时间戳的模块。同时支持与MAC间通过握手完成并行80bit时间戳和48bit时间戳的同步更新。

10.2 特性

RTC具有以下特性：

- 支持一个I2C接口专用于1588，负责时间戳的上送下发。
 - 该接口支持的频率范围为100kHz~2MHz。
 - 每秒计时到来时候，由时钟板通过I2C接口下发时间（TOD）给RTC，解析获取80bit时间戳。
 - 上送时钟板的T1/T2/T1CF/T3/T4/T3CF时间戳由NP下发给RTC，然后封装成480bit包通过I2C接口被Hi309a芯片外部的8006芯片定时读取。
 - 支持时钟板通过I2C接口1s内多次下发时间戳，最大2000次/s，要求I2C接口的频率在500kHz以上。
- 支持1PPS_TOD单线时间接口，解析得到秒脉冲和48bit时间戳，完成1588时间同步功能。
 - 按《接入网P300内部1pps时间同步规范v1.2》要求完成串行传输。
 - 解析得到秒脉冲和48bit秒时间，然后生成30bit纳秒时间。



- 考虑到单板传输延时，RTC逻辑看到的1PPS_TOD下降沿对应的秒脉冲相比客观的、标准的秒脉冲有一点滞后，支持软件配置延时调整以对此进行补偿。
- 上述48bit秒时间，指的是本秒的时间，但为了提高应用的灵活性和兼容性，支持“48bit秒时间指示本秒（缺省）还是下一秒”软件可配置，硬件根据配置对下发的秒信息做减1秒操作。
- 如果串口信息段的最后一个字节为0x00，则表示没有CRC校验，自然也不存在校验错的问题（非强制校验）。
- 支持秒和秒的间隔时间内，由芯片内部的精准参考时钟提供计时，完成在每秒内提供芯片所有相关模块所需的精确时戳，该时戳在每个1pps时会和系统下发的时钟对齐，软件对时戳更新时延可配置。
- RTC支持两种时戳：网络RTC的时间戳和无线主控传输时间戳，软件可配置MAG1_TIMESTAMP_MODE_CFG选择，MAC打戳的时间戳由MAG1_TGM_TEST_CFG配置选择是网络时间戳还是无线时间戳。
- 秒脉冲更新计时同时支持偏移量时钟调节方式。
 - CPU通过配置寄存器MAG1_TGM_TIMESTAMP_OFFSET_MODE和MAG1_TGM_TIMESTAMP_OFFSET_VALUE，可实现1秒内多次更新时戳。在当前时间基础上，完成1秒以内的正/负偏移。
 - 支持在500ms处完成偏移量调节模式，由寄存器MAG1_TGM_TIMESTAMP_OFFSET_MODE配置；在该模式下，在任意时刻下发偏移量时钟调节模式使能寄存器，都只在RTC时间为500ms（误差1ns）时更新时戳，此时配置的偏移量值小于500ms。
 - CPU通过配置寄存器MAG1_TS_OFFSET_SECOND_LO/MAG1_TS_OFFSET_SECOND_HI/MAG1_TS_OFFSET_SECOND_MOD_EN/MAG1_TS_OFFSET_SECOND_WR_EN，可实现替换秒信息功能。
- 支持生成串行64bit时戳信号给其他相关模块（SPPE，PPE，BMU等），64bit时间戳低位先传，每一个bit传输64ns。
- 支持MAC主动申请时间戳，RTC收到时间戳请求后下发80bit时间戳和48bit纳秒时间戳给MAC。
- 支持SPPE直接写寄存器的上送时戳方式。
- 1pps下发秒脉冲同时兼容支持2kHz调制产生1pps模式。2kHz模式下，在整秒时间后间隔18个2K时钟周期，产生指示1pps信息的脉冲。正常的2K时钟占空比是50%，而包含1pps信息的占空比是3：1，在375us的下降沿处产生1pps。
- RTC模块支持独立复位功能。
- RTC模块支持JTAG。
- RTC模块具备通过配置将本地RTC秒帧头输出到管脚的能力，用于测试芯片内的RTC是否与时钟板同步，将RTC内部时戳更新时秒脉冲展宽给芯片管脚，通过配置MAG1_TGM_PPS_OUT_CFG[1]、MAG1_TGM_PPS_OUT_WIDTH进行开启和配置脉冲宽度。
- RTC支持时间刷新的防护功能。
 - 对于接收到时钟板下发的时间与本地时间的差值小于用户配置的门限值的情况，本次接收到的时间不刷新到RTC。
 - 在相邻两秒刷新的80bit时戳信息的秒值相同的情况，具备后一秒不刷新的防护机制；上述两个功能均可配使能，默认关闭。

以下是RTC模块芯片DFX特性：

- 支持秒脉冲统计和实际刷新时间戳统计。



- 支持软件捕捉RTC内部本地80bit/48bit时间戳，支持捕捉时钟板通过I2C或者1pps+TOD单线接口下发的80bit时间戳。
- 支持软件捕捉SPPE下发的T1/T2/T1CF/T3/T4T3CF时间戳。
- RTC模块支持逻辑容错功能，对于不符合逻辑处理格式的接口命令给予丢弃并上报中断：
 - a. 上送时戳无响应中断：I2C上送时戳时，对端没有返回ack，报中断。

须知

RTC的对接芯片（SA8006）在通过I2C读取上送的60Bytes时戳的最后一个字节返回了NACK，导致RTC上报NACK中断，因此该中断对应的中断使能寄存器MAG1_INT_ENABLE[8]需要配置为0，屏蔽由于对端芯片（SA8006）返回NACK造成的中断。

- b. 接收时戳数据序列错误中断：当正在接收数据时，I2C重新启动一次Start，产生错误中断。
 - c. 秒帧头位置错误中断：接收时戳数据时，秒帧头出现位置错误。秒帧头在接收时戳数据或等待状态出现，上报该错误。
 - d. 丢失秒帧头错误中断：接收完时戳数据等待秒帧头时，I2C重启发起一次Start，产生该错误。
 - e. I2C接收数据CRC校验错误中断：I2C接收时戳数据CRC16校验错误。
- 1PPS+TOD独有中断：
- a. 刷新时间偏差过大中断：PPS刷新时间（pps_tod模式时）和本地时间偏差过大，大于fresh_rpt_offset_th时，上报中断告警。
 - b. PPS_TOD crc 8bit校验错误告警。
 - c. pps帧头丢失告警：当相邻pps间距偏离1s的量超过寄存器配置的抖动阈值，上报中断告警。
- 支持统计连续两次I2C下发时戳的最大间隔时间和最小间隔时间。
 - 支持内部关键状态寄存器可查询，供芯片debug使用。
 - 支持统计连续两次1pps 秒脉冲下发最大间隔时间和最小间隔时间，以及实时间隔统计。

10.3 关键术语和缩略语

该表10-1中包含了本章中出现的所有英文缩略语的解释描述。

表 10-1 RTC 关键术语和缩略语

术语	描述
PPS	Pulse per Second，每秒脉冲。
ToD	Time of Day，日期时间（就是年月日时分秒）。
RTC	Real Time Counter，实时计时器。
CDR	Clock Data Recovery，时钟数据恢复。



术语	描述
PTP	Precision Time Protocol，精确时间协议（就是IEEE 1588）。
TS	TimeStamp，时戳。

10.4 功能描述

10.4.1 应用框图

图 10-1 RTC 功能框图

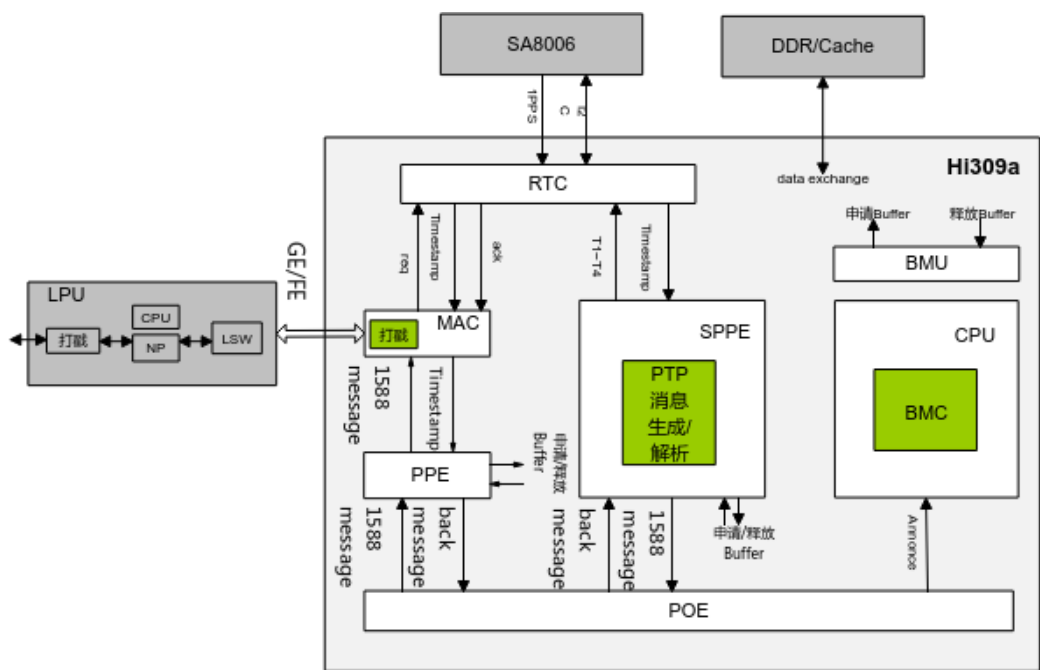
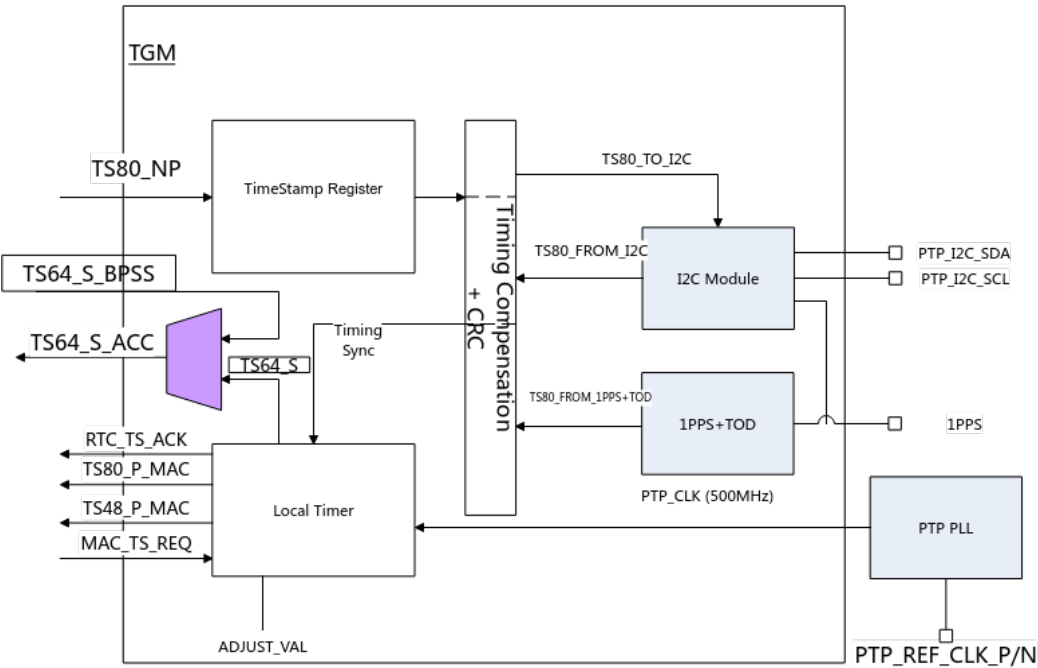


图 10-2 RTC 功能框图



10.4.2 相关模块

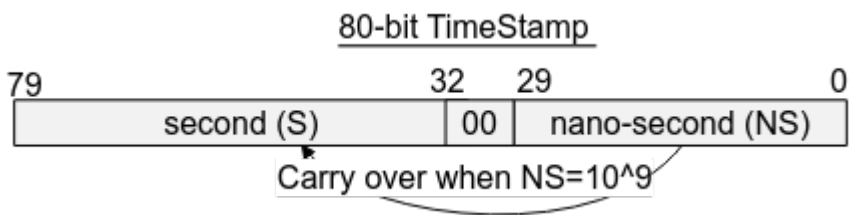
在Hi309a系统中, RTC模块会和下面多个模块进行交互, 分别描述如下:

- MAC: MAC向RTC请求时间戳, RTC将精确的80bit时间戳和48bit纳秒时间戳并行下发给MAC。
- SPPE: RTC将64bit时间戳以串行方式下发给SPPE, 同时SPPE上送T1、T2、T3、T4、T1CF、T3CF给RTC。
- BMU/PPE: RTC将64bit时间戳以串行方式下发给各加速器。
- 8006: 通过I2C接口或1PPS+TOD接口下发秒脉冲和时间戳给RTC, 读取RTC本地的T1、T2、T3、T4、T1CF、T3CF时间戳。
- CPU/SPPE: CPU和SPPE可以读取RTC的寄存器获取实时时间戳。

10.4.3 数据结构

Hi309a芯片内部有两种时戳格式, 一种是IEEE标准的80b, 其中低32b是ns信息, 高48b是s信息。格式如图10-3所示。

图 10-3 1588 80b Timestamp Format





须知

时钟板或软件下发的同步时间戳的ns部分不能超过1秒。

一种是内部Ingress用的32b时戳，其和标准的80b时戳转换关系如图10-4和图10-5所示。

图 10-4 Timestamp Conversion 80-bit → 32-bit

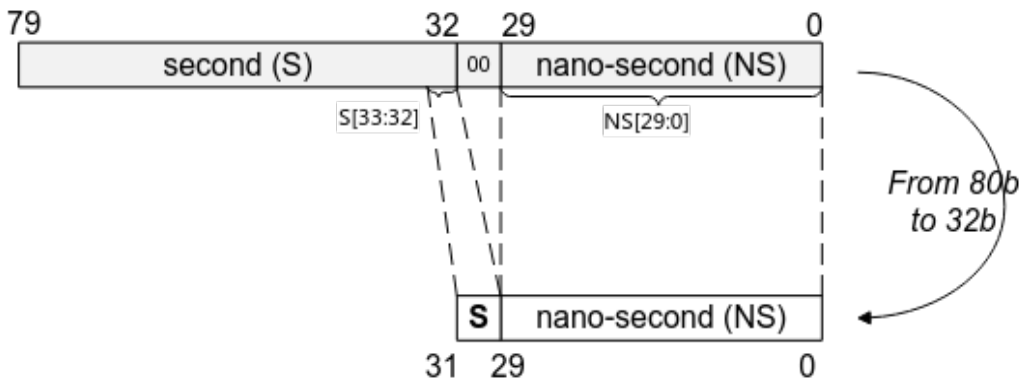
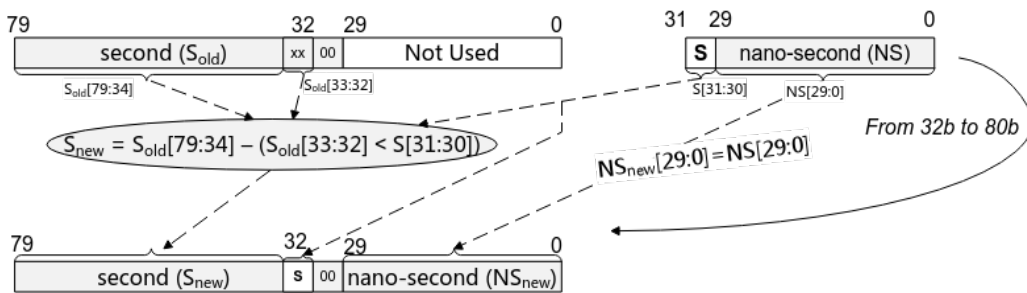


图 10-5 Timestamp Conversion 32-bit → 80-bit

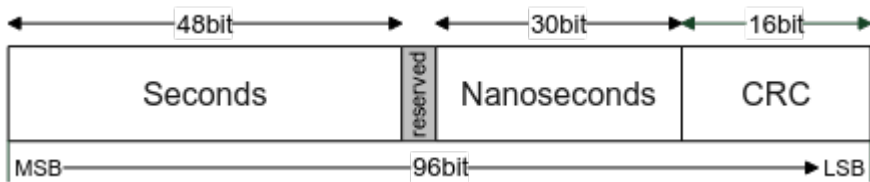


须知

RTC本身不输出32bit时间戳，由使用者自行转换。

时钟处理模块下发给RTC的时戳格式如图10-6所示（根据1pps和时戳的延时关系，软件可能会做时间补偿）。

图 10-6 1588 Timestamp Download Format

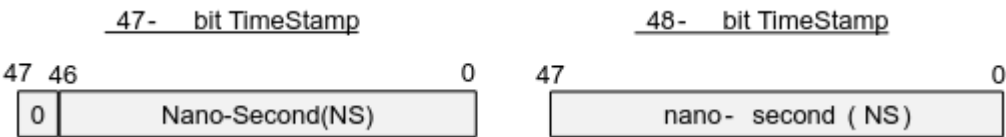




以上CRC采用CRC-16，多项式： $G(x) = X^{16} + X^{12} + X^5 + 1$ 。初始值为0xFFFF，对于计算产生的CRC，是取反之后下发的。

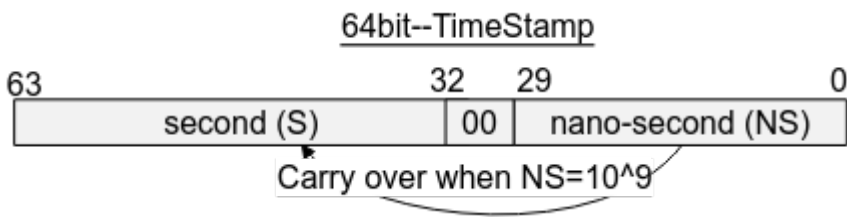
对于T1/T3的Correction Field，支持47b/48b两种模式可配置。47b为了和现网的1588逻辑方案兼容但有大概1.5天时戳溢出导致1588时间不准确风险，48b没有该问题。47b/48b和80b时戳转换关系如图10-7所示。

图 10-7 47b/48b Timestamp Format



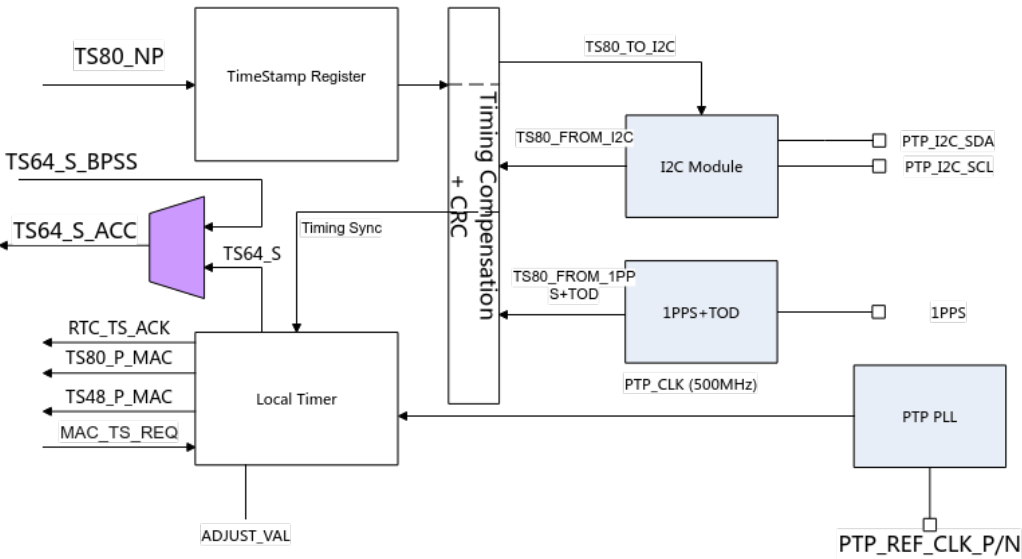
RTC串行输出给BMU/ PPE的64bit时间戳格式如图10-8所示。

图 10-8 串行 64bit 时间戳



10.4.4 同步时间戳

图 10-9 RTC 功能框图





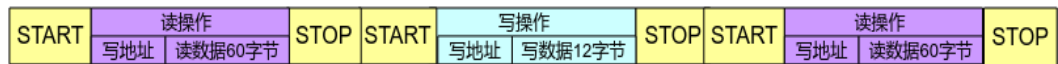
10.4.4.1 1PPS+I2C TOD

芯片外部时钟板通过I2C接口下发写命令，将80bit时间戳和16bit CRC校验位下发给RTC，由RTC完成I2C接口的协议处理，写命令的解析，接收80bit时间戳和16bit校验位，并将时间戳和校验位串行传输给RTC内部本地时间计数器，待时钟板通过1PPS管脚下发的秒脉冲到来时更新本地时间计数器。I2C接口是上送和下发时戳的通道，该接口时钟速率范围为100kHz~2MHz，最高速率高于常见I2C的400kHz。其他操作和普通I2C相同。

时钟单元下发秒帧头和I2C实时钟校正信息给RTC处理单元（先下发校正信息，然后下发秒帧头），然后RTC直接将校正信息写到实时计数器中校正时间。

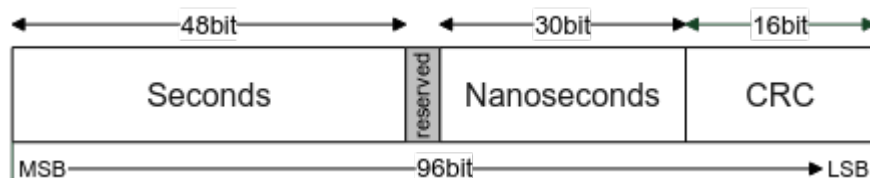
通过I2C接口进行时间信息下发和时间戳信息读取的操作过程为：每秒1次写操作，每次写12个字节，其余时间为读操作，每次读取80个字节，每次读写操作都由start开始，stop结束。所有读写操作都由时钟单元的SA8006发起，在时钟单元的I2C 永远作为master；业务板1588处理单元作为slave端。I2C协议中SCK由master控制，为output；SDA由master和slave共同控制，为inout。

图 10-10 I2C 接口数据格式



时钟处理模块下发给RTC的时戳格式如下（根据1pps和时戳的延时关系，软件可能会做时间补偿）：

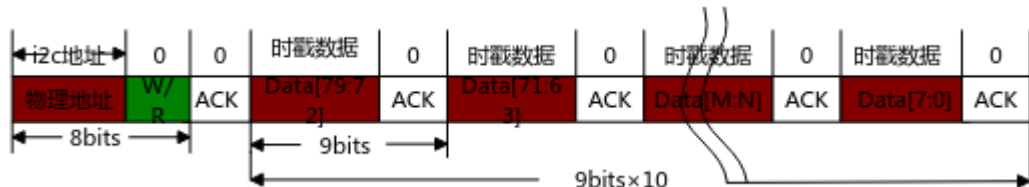
图 10-11 时戳格式



时钟板与接口I2C接口时戳通信：

I2C数据下发帧格式：

图 10-12 I2C 数据下发帧格式



传送格式为：

I2C地址位（7bits）+读写标志位（1bit,写标志），等待是否有应答（ACK）；如果有应答则下发80bits时戳信息（高48位为秒信息，低32位为纳秒信息）的高8bits

（DATA[79:72]），然后次高位（DATA[71:64]），以此类推，直到最低位的8bits（[7:0]），然后是16位的CRC校验，同样是先高后低传送；接口板接收完成后则接收到秒帧头后刷新时间，下发时间戳的入口补偿通过配置MAG1_TGM_DELAY_CFG寄存器完成。

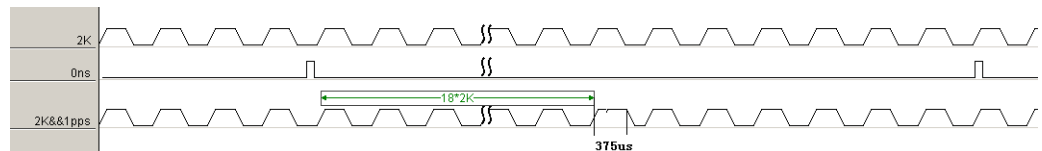
芯片同时兼容支持2kHz调制产生1pps模式。时间信息秒帧头支持2种格式：整秒秒帧头和2kHz调制的秒帧头：

整秒秒帧头：RTC单元检测该信号的上升沿，作为时间校准时刻。

2kHz调制的秒帧头把整秒指示调制到占空比为1:1的2kHz信号上，具体为把整秒帧头后面的第19个2K信号的占空比从1:1调制为3:1，当芯片内部检测到一个大于等于350μs的高脉宽时，便把此脉宽的下降沿作为时间校准时刻。对应下发时间为调制后2K下降沿的时间信息。

2kHz模式下，在整秒时间后间隔18个2K时钟周期，产生指示1pps信息的脉冲。正常的2K时钟占空比是50%，而包含1pps信息的占空比是3:1，在375μs的下降沿处产生1pps，如图10-13所示。

图 10-13 2kHz 调制图



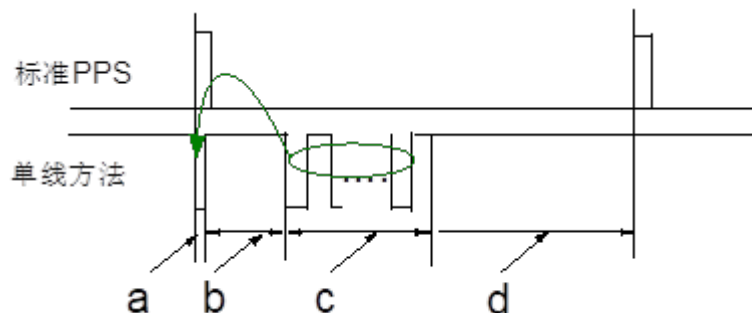
10.4.4.2 1PPS+ TOD

芯片外部时钟板通过复用1PPS，按照1PPS+TOD协议下发秒脉冲和时间戳，将秒脉冲和48bit秒信息及8bit CRC校验位下发给RTC，由RTC完成协议处理，接收48bit秒信息时间戳和8bit校验位，然后将时间戳扩展为80bit时间戳串行传输给RTC内部本地时间计数器，并且主动将秒脉冲延时200个周期后更新本地时间计数器，由于一次时间戳传输需要181.5μs，硬件需要根据软件配置值自动在30bit纳秒上补偿时间戳传输时间。

内部1pps时间同步具体格式。

此方法以串口为基础，在串口信号中添加PPS（每秒一个脉冲）来实现高精度PPS和年月日时分秒的串口数据的传送。

图 10-14 标准 PPS 信号与 P300 内部 PPS 信号的时序对应关系



1. 标准PPS信号与P300内部PPS信号的时序对应关系：



单线信号由4段组成：a（PPS段）+ b（等待段）+ c（串口信息段）+ d（空闲段）。

串口的波特率为1,000,000 bps，串口的起始停止各一位，无校验位。

PPS段设计：

图10-7中的a段，采用下降延的脉冲的方式，脉冲的下降延和系统PPS的上升延保持对齐。脉宽 = $(1/2) * (1/1M) = 0.5\mu s$ 。接收端检测a段宽度，偏差>10%，认为无效，处理流程参考附图。

主控板下发PPS相位与上级标准时间0延时（测量点在背板上）。PPS脉冲相位误差，跟RTC计数时钟频率相关，500M的时钟，最大相位误差2ns。各业务板需要的延时补偿在业务板完成，包含CITD的输出口的延时补偿（只支持开局测量时补偿）。

1PPS+TOD输入口延时补偿在时钟模块完成。CITD板接口输入1PPS+TOD信号转换为标准的1PPS复合信号，上背板送主控板时钟模块，考虑到CITD无与输入1PPS+TOD接口同源的高频时钟，延时补偿需要在时钟模块完成。

等待段：

图10-6中的b段，为了让下游设备能够很好的区分PPS段和串口信息段，需要在两者之间添加等待段。等待段保持高电平，等待段的持续的时长为2个串口帧周期 = $2 * (1/1M) * (1+8+1) = 20\mu s$ 。（注：1：起始位 + 8：数据位 + 1：停止位）。

串口信息段：

图10-4中的c段，串口的详细格式：1位起始位（低电平）+ 8位数据位 + 1位停止位（高电平）。串口信息段的持续的时长为16个串口帧周期 = $16 * (1/1M) * (1+8+1) = 160\mu s$ 。

须知

- PPS段之前的高电平要求至少40μs。
- PPS段的0.5μs帧头脉宽抖动阈值，当实际的脉宽与0.5μs的偏差超过这个阈值，认为pps帧头不正确。
- 等待段的20μs高电平脉宽抖动阈值寄存器可配。当实际的高电平宽度与20μs的偏差超过这个阈值，认为此pps格式不正确。
- 相邻pps间距抖动阈值寄存器可配。当间距偏离1s的量超过此阈值，认为1pps丢失。

串口信息段 CRC 实例

假设48bit 1588秒时间为0x0000_509B_9EC0（大约是公元2012年11月9日中午的某个时刻），6字节ToD暂未使用，连同随后的3字节保留均为0x00，则CRC8计算结果为0x39，bitswap后为0x9C，整个串口信息段的16字节按从前到后的顺序依次为：（均为16进制，每个字节均为最低位/bit[0]先传，最高位/bit[7]后传）。

C0 9E 9B 50 00 00 00 00 00 00 00 00 00 00 00 9C

空闲段

图10-4中的d段，由于发送数据的间隔较长（为1秒），同时为了很好的区分串口信息段和下一个PPS段，需要插入空闲段。空闲段的为持续高电平，持续的时长（取决于前三个段的长度）= $1s - 1/2M - 2 * (1/1M) * (1+8+1) - 16 * (1/1M) * (1+8+1) = 999.8195ms$ 。



10.4.5 RTC 内部时间更新

- 支持秒和秒的间隔时间内，由芯片内部的精准参考时钟500M提供计时，完成在每秒内提供芯片所有相关模块所需的精确时戳，每周期步进2ns，该时戳在每个1pps时会和系统下发的时钟对齐。

Local Timer完成在每秒内提供芯片所有1588相关模块所需的精确时戳，该时戳在每个1pps时会和系统下发的时钟对齐。当然因为时戳的计时时钟精度非常准可以达到50ppb（Parts Per Billion），可能连续的多次秒脉冲到来时芯片内部的Local Timer和下发的时戳本身就是一样的。Local Timer的计数周期通过以下参数配置：integer, half, quarter 和numerator/denominator，单位为ns，计算公式如下：

Timer Period (ns) = integer + half*0.5 + quarter*0.25 + numerator/denominator.

例如，对于156.25MHz的时钟周期是6.4ns，则：integer=6, half=0, quarter=1, numerator/denominator = 3/20。

Local Timer的计时时钟是PTP_CLK，500MHz。计时周期2ns，因此配置：

```
integer = 2
half = 0
quarter = 0;
denominator = 0;
numerator = 0;
NP upload interface
```

- 秒脉冲更新计时同时支持偏移量时钟调节方式。
 - CPU通过配置寄存器MAG1_TGM_TIMESTAMP_OFFSET_MODE和MAG1_TGM_TIMESTAMP_OFFSET_VALUE，可实现1秒内多次更新时戳。在当前时间基础上，完成1秒以内的正/负偏移。
 - 支持在500ms处完成偏移量调节模式，由寄存器MAG1_TGM_TIMESTAMP_OFFSET_MODE配置；在该模式下，在任意时刻下发偏移量时钟调节模式使能寄存器，都只在RTC时间为500ms（误差1ns）时更新时戳，此时配置的偏移量值小于500ms。

须知

80bit的ns部分，只支持500ms刷新模式，非500ms不需要支持。

- CPU通过配置寄存器MAG1_TS_OFFSET_SECOND_LO/MAG1_TS_OFFSET_SECOND_HI/ MAG1_TS_OFFSET_SECOND_MOD_EN/MAG1_TS_OFFSET_SECOND_WR_EN，可实现替换秒信息功能。
- 1PPS+TOD Debug模式支持软件下发80bit时间戳给本地时戳计数器。
 - 1PPS+TOD模式下CPU通过配置寄存器TS_MODE，选择debug模式。
 - CPU通过配置寄存器LOAD_S_H16、LOAD_S_L32、LOAD_NS来下发80bit时间戳给本地计数器。
 - CPU通过配置寄存器LOAD_EN来通知硬件将软件下发的时间更新本地时间计数器。

须知

在无线主控模式下，建议使用偏移量时钟调节方式给RTC本地计数器下发一次初始时间戳。

10.4.6 MAC 时间戳请求

MAC向RTC发起时间戳同步的请求后，RTC检测到该请求将最新的本地80bit时间戳和48bit时间戳发送给MAC，同时发出响应信号到MAC。

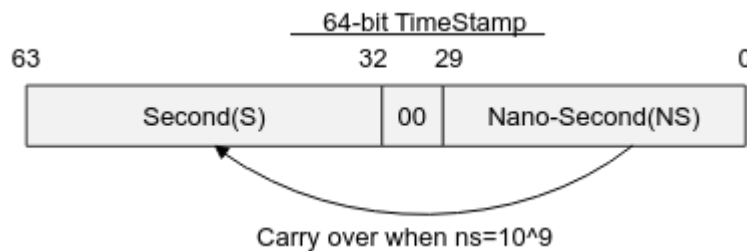
10.4.7 48bit 时间戳产生

- 48bit纳秒级时间和80bit时间戳的换算公式如下：
48比特纳秒时间 = 48bit秒级时间* 10^9 + 32bit纳秒级时间。

10.4.8 64bit 串行时间戳

RTC实时通过专用接口，将时戳传递给各加速器。该时戳是64b格式（取全部80b的低64b）。64b时戳格式如图10-15所示。

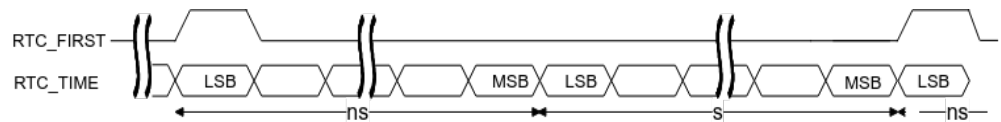
图 10-15 64b Timestamp Format



外送串行时间戳二选一：

- 网络RTC生成串行64bit时间戳信息给外部各个模块。
- 选择内部自己产生的时间戳输出。

图 10-16 串行 64bit 时间戳传输示意图



RTC_FIRST信号指示第一组64b的时间戳开始位置，RTC_TIME对应是64b时戳值，有32b的秒部分时间和32b的ns时间部分。32b的ns时间LSB首先发送，然后是秒时间的LSB。中间没有停顿。

软件通过配置MAG1_TIMESTAMP_MODE_CFG和MAG1_TGM_I2C_CFG选择，其中：

MAG1_TIMESTAMP_MODE_CFG bit[1]: 0表示无串行时间戳输出；1表示选择本地时间计数器。



若时钟板下发同步时间戳更新RTC内部时间戳，配置MAG1_TIMESTAMP_MODE_CFG选择不同模式，bit[0]：0表示I2C+1PPS，1表示1PPS+TOD

MAG1_TGM_I2C_CFG bit[9]：0表示不与时钟板下发的时间戳同步；1表示与时钟板下发的时间戳同步。

64bit时间戳传输格式：低bit先传，每一bit传输64ns。

10.4.9 T1~T4CF 的传送

SPPE从事件报文中提取时戳和校正域，传递给I2C模块。每次时钟单元的I2Cmaster发起读请求时，I2C模块把时戳信息按照固定格式放到IIC串行总线传递给时钟板。

上送时钟板时戳寄存器定义，设置多个寄存器，含以下信息：

表 10-2 1588 Timestamp Upload Format

80bit	80bit	64bit	16bit	80bit	80bit	64bit	16bit
T1	T2	T1_CF	T1_T2_CRC C	T3	T4	T3_CF	T3_T4_CRC

须知

上送顺序为T1->T2->T1_CF->T1T2_CRC->T3->T4->T3_CF->T3T4_CRC。

上述同步信息域按照从左到右顺序发送；每个域在IIC总线上都是坚持大端原则，即高比特先发送。

- 如果延时测量机制为delay require-response mechanism（E2E），同步信息域定义如下：
 - T1：master产生sync报文时的时间戳。
 - T2：slave接收到sync报文时的时间戳，必须是与T1配对的。
 - T1_CF：sync报文携带的修正域（+follow_up报文细带的修正域）。
 - T1T2_CRC：根据T1、T2和T2_CF计算的CRC值。
 - T3：slave产生delay_req报文时的时间戳。
 - T4：master接收到delay_req报文时的时间戳，必须是与T3配对的。
 - T3_CF：delay_resp报文携带的修正域。
 - T3T4_CRC：根据T3、T4和T4_CF计算的CRC值。
- 如果延时测量机制为peer delay mechanism，同步信息域定义如下：
 - T1：master产生sync报文时的时间戳。
 - T2：slave接收到sync报文时的时间戳，必须是与T1配对的。
 - T1_CF：sync报文携带的修正域（+follow_up报文携带的修正域）。
 - T1T2_CRC：根据T1、T2和T2_CF计算的CRC值。
 - T3：0。
 - T4：0。



- T3_CF: Pdelay值（经软件过滤后的Pdelay值），占T3_CF的位数[63:16]。
- T3T4_CRC: 根据T3、T4和T4_CF计算的CRC值。
- T1T2_CRC的值为T1/T2/T1_CF三个域做CRC计算后，按位取反的值。
- T3T4_CRC的值为T3/T4/T3_CF三个域做CRC计算后，按位取反的值。

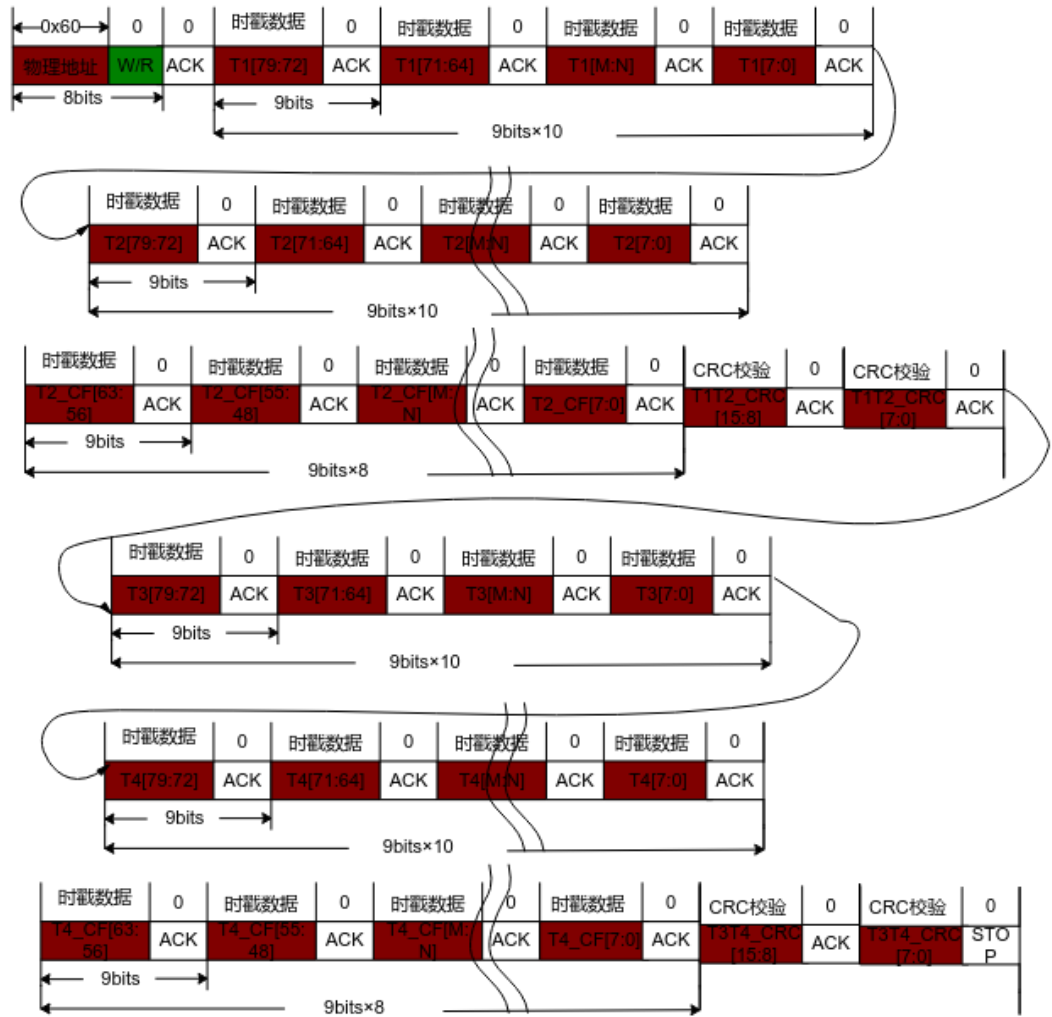
上述的CRC为CRC16，采用标准的CRC-CCITT，初始值为0xFFFF，生成多项式为 $G(x) = X^{16} + X^{12} + X^5 + 1$ 。对于计算产生的CRC，是取反之后下发的，这点需要特别注意。

传送格式为：

I2C地址位（7bit）+读写标志位（1bit，读标志），等待是否有接口板应答（ACK）；如果有应答则开始接收80bit的T1时戳信息的高8bit（T1[79:72]），同时产生一个应答信号，然后接收T1次高位（T1[71:64]），以此类推，时钟板每接收8bit数据，就会产生一个应答信号，直到T1的最低位的8bit（T1[7:0]）；然后接收80bit的T2时戳信息高8bit（T2[79:72]），接受完成后产生应答，接收T2次高位（T2[71:64]），以此类推，直到T2时戳的最低位的8bit（T2[7:0]）；再次接收80bit的T2_CF时戳信息高8bit（T2_CF[63:56]），接收完成后产生应答，接收T2_CF次高位（T2_CF[55:48]），以此类推，直到T2_CF时戳的最低位的8bit（T2_CF[7:0]）；然后是16位的CRC校验，同样是先高后低传送，如图10-17所示。同样的T3（80bit）、T4（80bit）、T4_CF（64bit）以及CRC（16bit）接收。



图 10-17 上送时间戳传送格式



10.4.9.1 T1~T4、CF 更新

芯片支持SPPE直接写寄存器的上送时戳方式。相关寄存器为MAG1_TS_T1_LO、MAG1_TS_T1_MI、MAG1_TS_T1_HI、MAG1_TS_T2_LO、MAG1_TS_T2_MI、MAG1_TS_T2_HI、MAG1_TS_T3_LO、MAG1_TS_T3_MI、MAG1_TS_T3_HI、MAG1_TS_T4_LO、MAG1_TS_T4_MI、MAG1_TS_T4_HI、MAG1_TS_T1CF_LO、MAG1_TS_T1CF_HI、MAG1_TS_T3CF_LO、MAG1_TS_T3CF_HI，然后通过I2C上送时钟板。时戳寄存器必须按以下顺序配置：

MAG1_TS_T1* -> MAG1_TS_T2* -> MAG1_TS_T1CF_LO-> MAG1_TS_T1CF_HI；

MAG3_TS_T3* -> MAG1_TS_T4* -> MAG1_TS_T3CF_LO-> MAG1_TS_T3CF_HI。

然后配置MAG1_TS_T1CF_HI_WEN、MAG1_TS_T3CF_HI_WEN通知RTC最新的上报时间戳已写入寄存器。

RTC完成T1~T4、T1CF、T2CF的封装，产生T1、T2、T1CF的CRC校验位，T3、T4、T3CF的CRC校验位，封装成480bit数据，串行送给I2C处理。

RTC响应I2C请求传送封装好的T1~T4、T1CF、T3CF给SA8006。



10.4.10 DFX

RTC模块芯片特性：

- 支持内部异常中断主动上报，详见MAG1_RINT，MAG1_RINT1。
- 支持秒脉冲统计和实际更新时间戳统计。
- 支持软件捕捉RTC内部本地80bit/48bit时间戳，支持捕捉时钟板通过I2C或者DLCS下发的80bit时间戳。
- 支持软件捕捉SPPE下发的T1/T2/T1CF/T3/T4T3CF时间戳。
- RTC模块支持逻辑容错功能，对于不符合逻辑处理格式的接口命令给予丢弃并上报中断。
- 支持统计连续两次I2C下发时戳的最大间隔时间和最小间隔时间。
- 支持内部关键状态寄存器可查询，供芯片debug使用。

须知

请参见《Hi309a 寄存器手册》。

10.5 工作方式

10.5.1 时钟和复位

三要素：模块工作时钟频率，复位方式和复位置起/撤离顺序，是否有跨时钟域处理。

- RTC的PTP工作时钟频率为500MHz。
- RTC的APB slave时钟频率为200MHz。
- RTC模块全部采用异步复位。

须知

RTC的PTP时钟域上电后处于复位状态，需要软件先配置ETH系统控制器SC_RTC_500M_RESET_REQ使RTC的PTP时钟域撤离复位后软件可访问RTC。

10.6 初始化

- 步骤1** 软件配置ETH_SUBCTRL的寄存器SC_PPE_ICG_DIS0.icg_dis_rtc为1，关闭RTC的工作时钟。
- 步骤2** 软件配置hilinkmux SUBCTRL的寄存器RTC_500M_CLK_SEL，选择RTC的工作时钟源。（如果软件选择使用hilink的mclk作为RTC的工作时钟，请在初始化RTC前完成hilink的初始化）
- 步骤3** 软件配置ETH_SUBCTRL的寄存器SC_PPE_ICG_EN0.icg_en_rtc为1，打开RTC的工作时钟，等待10μs后查看门控状态寄存器SC_PPE_ICG_ST0，若icg_st_rtc位为0，则执行下一步，否则认为存在异常。



- 步骤4** 软件配置RTC的寄存器MAG1_TIMESTAMP_MODE_CFG.timestamp_mode_cfg为1，选择RTC送给加速器的时间戳模式为本地时间（一定不能配置为0）。
- 步骤5** 软件配置RTC的寄存器MAG1_TIMESTAMP_MODE_CFG.pps_tod_mod_cfg，配置时钟板下发时间戳的模式。
- 步骤6** 如果**步骤5**中的时钟下发时间戳模式配置为I2C+1PPS，还需要配置IOMUX的寄存器iomg144、iomg145、iomg 197，选择将管脚复用为RTC的I2C，否则跳过此步骤。
- 步骤7** 软件配置RTC的寄存器MAG1_TGM_TEST_CFG.rtc2mac_ts_sel为1，选择使用网络RTC给MAC；（网络模式必须为1，否则会导致时间戳不符合业务预期）。
- 步骤8** 软件配置RTC的寄存器MAG1_TGM_TEST_CFG.ref_second_same_en。
- 步骤9** 软件根据业务需求配置RTC的寄存器PPS_TOD_CFG、PPS_CHK、FRESH_RPT_OFFSET_TH、TOD_CHK、TS_MODE。

----结束

须知

在时钟下发时间戳配置为1PPS+TOD时，如果调试不通，可以试着将PPS_TOD_CFG.auto_width_adj_en配置为0x1，使用相对时钟板的时间去采样时钟板的信号。

10.6.1 软复位

RTC模块的PTP时钟域可以单独软复位，配置系统软复位控制器，RTC被复位。

独立复位流程：

RTC模块独立复位应用解决内部逻辑挂死或其他不可恢复的问题，复位后RTC恢复到上电状态，为保证RTC复位期间不挂死总线，要严格按照步骤依次完成RTC的独立复位。

- 步骤1** 软件配置SC_PPE_RESET_REQ3.srst_req_rtc为1，启动RTC 500M时钟域的复位，然后软件查询SC_PPE_RESET_ST3.srst_st_rtc为1，表示500M时钟域复位有效。
- 步骤2** 软件延时等待，保证RTC复位可以正常完成。
- 步骤3** 软件配置SC_PPE_RESET_DREQ3.srst_dreq_rtc为1，撤离RTC 500M时钟域的复位，然后10 μ s后软件查询SC_PPE_RESET_ST3.srst_st_rtc为0，表示500M时钟域复位已撤离，否则认为存在异常。
- 步骤4** 独立复位完成后，软件需要重新配置RTC寄存器，启动RTC工作。

----结束

10.6.2 本地时间同步模式

- 1PPS + I2C模式
- 2K复用1PPS + I2C模式
- 偏移量调节模式
- 1PPS + TOD模式
- 1PPS + TOD debug模式



须知

- 以上五种同步时间的模式在更新本地计数器时都是互斥的。
- 同一时刻只允许一种模式，debug模式下，软件最快1s下发一个时间戳。
- 偏移量调节模式下，如果软件下发的时间导致时间戳下溢或上溢，80bit时间戳和48bit纳秒时间戳会不匹配。

10.7 性能

10.7.1 精度

芯片RTC引入的抖动小于等于10ns。

10.8 调试（模块级错误处理）

10.8.1 关键状态

MAG1_TS_P2S_STATUS、MAG1_TGM_I2C_SLAVE_STATUS、MAG1_TGM_TS_PRO_STATUS、I2C_STAT、PPS_TOD_STAT寄存器记录RTC的内部实时的状态。CPU对以上寄存器只读。

10.8.2 诊断计数器

- MAG1_TGM_PPS_REFRESH_CNTT：时戳处理模块实际刷新时戳次数统计寄存器，计满翻转，读清0。
- MAG1_TGM_PPS_ONESEC_CNT：时戳处理模块pps脉冲统计寄存器，计满翻转，读清0。
- MAG1_TGM_I2C_COD_CNT：I2C模块I2C处理时戳状态计数器，计满翻转，读清0。
- MAG1_TGM_I2C_OP_CNT：I2C模块读写读写时戳计数器，计满翻转，读清0。
- MAG1_TGM_TS_OP_CNT：TGM模块时戳处理计数器，计满翻转，读清0。
- RMAG1_TGM_TS_PROCESS_ERR_CNT：时戳处理中的错误计数寄存器，饱和计数，写清0。
- MAG1_TGM_I2C_CRC_ERR_CNT：接收时戳CRC16校验错误计数寄存器，饱和计数，写清0。
- 1PPS+TOD复用模式独有统计计数器如下：
- PPS_LOS_CNT：PPS帧头丢失统计计数器，饱和计数，写清0。
- PPS_HEAD_ERR_CNT：PPS_TOD帧头脉宽错误次数，饱和计数，写清0。
- PPS_CTRL_ERR_CNT：PPS帧头数据控制位错误，饱和计数，写清0。
- CRC_ERR_CNT：CRC校验错误次数，饱和计数，写清0。
- FRESH_TIME_SAME_CNT：PPS_TOD刷新时间和本地时间相近次数统计计满翻转，读清0。



- RTC_FRESH_CNT：RTC刷新次数计数器，计满翻转，读清0。
- 1PPS+TOD诊断模式下支持CPU下发刷新时戳：
 - 配置时钟板接口为1PPS+TOD格式（pps_tod_mod_cfg）。
 - 配置1PPS+TOD的debug功能（ts_mode）。
 - 软件配置时戳下发使能（load_en）。

10.9 注意事项

须知

I2C接口下发时间戳时，如果在某次时间戳的传输过程中因异常中断传输，导致CRC校验错误，在I2C接口继续正常下发两次时间戳后RTC逻辑功能恢复正常。

10.10 附顶层接口信号描述

表 10-3 RTC 模块接口信号

Signal	位宽	I/O	Descriptions
系统相关信号			
clk_rtc_500m	1	I	RTC模块功能主时钟。
rst_rtc_n	1	I	RTC模块功能主时钟对应复位信号，低有效。
clk_apb_200m	1	I	RTC APB Slave接口时钟。
rst_apb_n	1	I	RTC APB Slave接口复位低有效。
I2C接口信号			
i2c_rtc_scl	1	I	i2c_1588时钟输入信号。
i2c_rtc_sda	1	I	i2c_1588数据输入信号。
rtc_i2c_sda	1	O	i2c_1588数据输出信号。
rtc_i2c_sda_oe	1	O	i2c_1588数据输出使能信号，高有效。
串行64bit时间戳			
timestamp_clk_rtc	1	O	RTC的串行64bit时间戳输入随路时钟信号。
timestamp_start_rtc	1	O	RTC的串行64bit时间戳输入起始指示信号。
timestamp_time_rtc	1	O	RTC的串行64bit时间戳输入数据信号。



Signal	位宽	I/O	Descriptions
MAC 并行48bit/80bit时间戳接口			
o_mag0_timie_stamp	80	O	并行80bit时间戳。
o_mag0_timie_stamp_ns	48	O	并行48bit时间戳。
APB接口			
paddr_rtc	32	I	APB paddr地址信号。
psel_rtc	1	I	APB psel片选信号。
penable_rtc	1	I	APB penable允许信号。
pwrite_rtc	1	I	APB pwrite读/写控制信号，高：表示写操作；低：表示读操作。
pwdata_rtc	32	I	APB pwdata写数据总线。
Prdata_rtc	32	O	APB prdata读数据总线。
JTAG接口			
tap_si_rtc	1	I	JTAG测试数据串行输入信号。
tap_clockdr_rtc	1	I	JTAG串行寄存器时钟使能信号。
tap_shiftdr_rtc	1	I	JTAG串/并输入数据选择信号。
tap_tck_rtc	1	I	JTAG输入时钟信号。
tap_reset1_rtc	1	I	JTAG寄存器复位信号，低电平有效。
tap_updatedr_rtc	1	I	JTAG并行寄存器时钟使能信号。
tap_reg_en_rtc	1	I	JTAG寄存器使能信号。
tap_so_rtc	1	O	JTAG测试数据串行输出信号。
中断接口			
int_rtc	1	O	RTC送往CPU的中断上报信号。
其他			
pps_second_out	1	O	输出到IO的内部1pps时钟。



11 MDIO

11.1 MDIO

本段落主要是对MDIO的功能和应用做详细描述。

11.1.1 概述

MDIO是对外部以太网接口的PHY芯片进行配置和管理的接口控制模块，软件可以通过配置MDIO相应的寄存器实现对PHY芯片内部寄存器的读写，最终实现对PHY芯片的控制和管理。例如复位控制、自环控制、速度选择、双工选择、冲突测试、自协商使能、物理层状态获取等等。

11.1.2 特性

MDIO具有以下特性：

- 支持3个独立的MDIO接口。
- 每个MDIO接口可独立配置为支持IEEE802.3协议的Clause22章节或Clause45章节。
- 支持MDIO接口时序，MDC时钟频率为2/20/6.25MHz。
- 支持对PHY内部寄存器的读写访问、PHY地址、寄存器地址等可配置。
- 支持对GE PHY内部寄存器的自动查询，每个MDIO接口最大支持5个PHY器件的状态查询，PHY状态改变则上报中断（每个配置的PHY地址可以监控指定最多2个地址寄存器的任意指定位状态值是否发生改变）。

11.1.3 关键术语

下表中的术语将在本章节表示特殊的含义。



表 11-1 MDIO 关键术语描述

术语	描述
MDIO	Management Data Input/Output。 MDIO是以太网中PHY和STA之间的一个双向的信号，主要是用来传输PHY和STA之间的控制和状态信息。STA发出控制信息，和MDC同步，PHY返回状态信息。
MDC	是MDIO接口的时钟，一般为2.5MHz。
STA	Station Management，一般指MAC层所在的芯片，这里指AurumV200。
PHY	物理层器件/对接芯片（Physical Layer Device）。

11.1.4 遵循协议

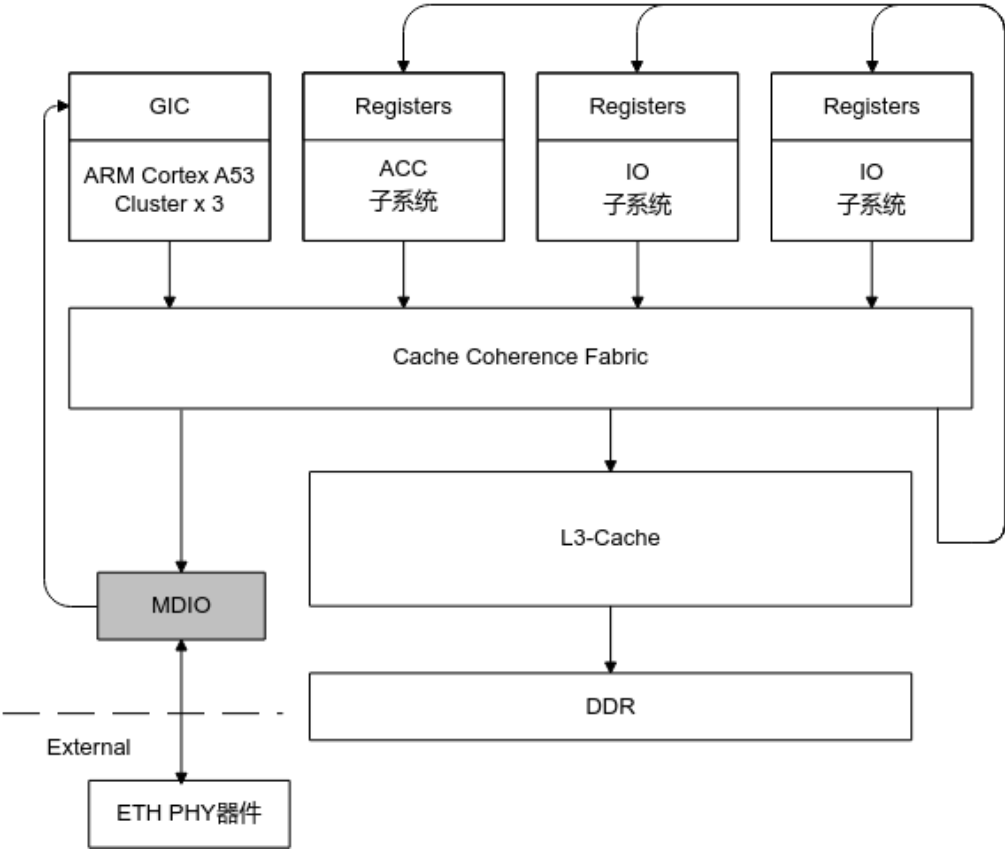
遵循IEEE802.3协议。

11.1.5 功能描述

11.1.5.1 应用框图

MDIO的应用如图11-1所示。

图 11-1 MDIO 应用图





11.1.5.2 相关模块

在系统中，MDIO模块会和下面多个模块进行交互，分别描述如下：

CPU：CPU可以通过配置MDIO相应寄存器实现对PHY芯片内部寄存器的读写，最终实现对PHY芯片的控制和管理。

11.1.5.3 模块框图

略。

11.1.5.4 数据结构

MDIO支持IEEE802.3 Clause22和Clause45，对应的有两种数据帧格式。

11.1.5.4.1 Clause22 的数据帧格式

表 11-2 MDIO 接口的 Clause22 数据帧格式

操作方式	数据帧格式							
	PRE	ST	OP	PHY AD	REG AD	TA	DATA (payload)	IDLE
Read	1……1	01	10	AAAAA	RRRRR	Z0	DDDDDDDDDDDDDDDDDD	Z
Write	1……1	01	01	AAAAA	RRRRR	10	DDDDDDDDDDDDDDDDDD	Z

数据帧各域的含义如下：

- PRE：帧同步码，32bit位宽。
PRE是32个连续的“1”，用于器件的同步通信。但该域不是必须的，为了提高效率，有些PHY器件的MDIO操作就没有此域。
- ST：帧开始标志。MAC输出“01”比特通知PHY表示帧操作开始。
- OP：帧操作码，2bit位宽。
 - 01：表示此帧为一次写操作请求。
 - 10：表示此帧为一次读操作请求。
- PHY AD：PHY器件地址，5bit位宽。
每个PHY器件都把自己的地址与这5bit进行比较，若匹配则响应后面的操作；若不匹配，则忽略后面的操作。
5bit决定了一个MDIO接口最多只能支持32个PHY器件的配置。
- REG AD：PHY寄存器的地址，5bit位宽。
用来选择PHY内部的32个寄存器中的某个寄存器，PHY为物理层接口，每个PHY接口含有32个16bit的寄存器。
一般情况下，不同PHY芯片的前16个寄存器内容应该一致，后面16个是可由用户指定功能的寄存器。



- TA：状态转换域，2bit位宽。
 - 读操作时，第1bit由MDIO送出高阻态，第2bit由PHY给MDIO信号置0。
 - 写操作时，MDIO信号由MDIO模块控制，连续输出“10”两个bit。
- DATA：帧的寄存器数据域，16bit位宽。
 - 读操作时，为PHY送到MDIO模块的串行数据。
 - 写操作时，为MDIO模块送到PHY的串行数据。
- IDLE：帧结束后的空闲状态。
此时MDIO无源驱动，处于高阻状态，但一般用上拉电阻使其处在高电平。

11.1.5.4.2 Clause45 的数据帧格式

表 11-3 MDIO 接口的 Clause45 数据帧格式

操作方式	数据帧格式							
	PRE	ST	OP	PRTAD	DEVA D	TA	ADDRESS/DATA (payload)	IDLE
Address	1……1	00	00	PPPP P	EEEE E	10	AAAAAAAAAAAAAAAA	Z
Write	1……1	00	01	PPPP P	EEEE E	10	DDDDDDDDDDDDDDDD	Z
Read	1……1	00	11	PPPP P	EEEE E	Z0	DDDDDDDDDDDDDDDD	Z
Post-read-increment-address	1……1	00	10	PPPP P	EEEE E	Z0	DDDDDDDDDDDDDDDD	Z

数据帧各域的含义如下：

- PRE：帧同步码，32bit位宽。
PRE是32个连续的“1”，用于器件的同步通信。但该域不是必须的，为了提高效率，有些PHY器件的MDIO操作就没有此域。
- ST：帧开始标志。MAC输出“00”比特通知PHY表示帧操作开始。
- OP：帧操作码，2bit位宽。
 - 00：表示此帧为一次写地址操作，payload中的内容为要访问的寄存器的地址。
 - 01：表示此帧为一次写数据操作，payload中的内容为要写入的数据。
 - 11：表示此帧为一次读数据操作，payload中的内容为读出的数据。
 - 10：表示此帧为地址自增读数据操作，payload中的内容为读出的数据。
- PRTAD：PHY器件地址，5bit位宽。
 - 每个PHY器件都把自己的地址与这5bit进行比较，若匹配则响应后面的操作；若不匹配，则忽略后面的操作。



- 5bit决定了一个MDIO接口最多只能支持32个PHY器件的配置。
- DEVAD：PHY器件中的设备地址，5bit位宽。
支持IEEE802.3 Clause45章的1个PHY器件内部可以支持最多32个设备，这个域就是表示PHY器件内部的设备地址。
- TA：状态转换域，2bit位宽。
 - 读数据或者地址自增读数据操作时，第1bit由MDIO送出高阻态，第2bit由PHY给MDIO信号置0。
 - 写地址或者写数据操作时，MDIO信号由MDIO模块控制，连续输出“10”两个比特。
- DATA：帧的寄存器数据域，16bit位宽。
 - 读数据或者地址自增读数据操作时，为PHY送到MDIO模块的串行读数据。
 - 写地址操作时，为MDIO模块发出的要访问的串行某个寄存器地址。
 - 写数据操作时，为MDIO模块发出的串行写数据。
- IDLE：帧结束后的空闲状态。
 - 此时MDIO无源驱动，处于高阻状态，但一般用上拉电阻使其处在高电平。

11.1.5.5 接口信号

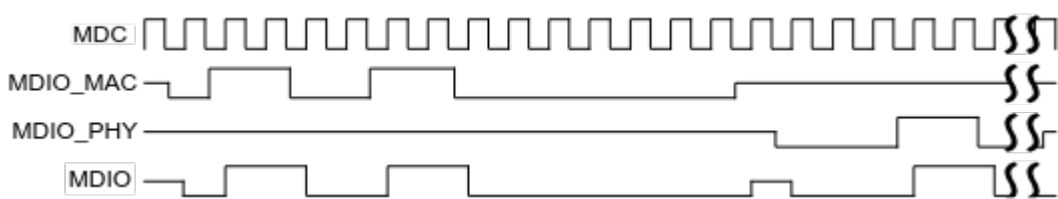
表 11-4 MDIO 接口信号描述

名称	方向	描述	管脚
MDIO0_CLK	O	时钟信号	MDIO0_CLK
MDIO0_DATA	I/O	数据信号	MDIO0_DATA

11.1.5.6 接口读时序

Clause22章的读操作和Clause45章的读数据和地址自增读操作的时序是一致的。

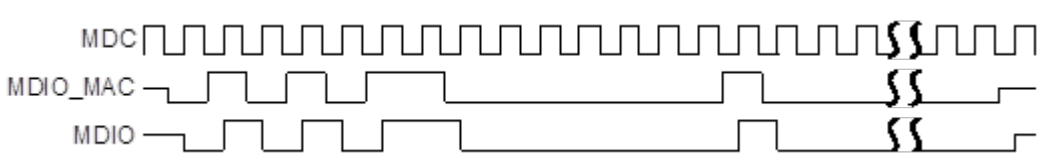
图 11-2 MDIO 接口读时序



11.1.5.7 接口写时序

Clause22章的写操作和Clause45章的写地址和写数据操作的时序是一致的。

图 11-3 MDIO 接口写时序





说明

- 图11-2和图11-3中，MDC为2.5MHz时钟，MDIO_MAC是由MAC发出的信号，MDIO_PHY是由PHY发出的信号，MDIO是两侧信号在三态总线上的状态。
- 无论是MAC还是PHY，都是在MDC的下降沿发出数据，对端在MDC的上升沿接收数据，确保数据的稳定接收窗口。

11.1.5.8 PHY 状态自动查询

IEEE802.3 Clause22章中的MDIO模式支持PHY状态自动查询，当配置为自动查询PHY状态时，MDIO模块可以自动发出对指定PHY器件指定寄存器地址的读操作，最大支持5个PHY器件，PHY地址可配置，PHY中的寄存器地址也可配置。

11.1.5.9 中断

当PHY状态查询功能开启，MDIO检测到指定PHY的指定寄存器的状态发生变化时，上报中断。

11.1.5.10 Clause22 章和 Clause45 章模式配置

PHY接口模式可通过配置寄存器MDIO_COMMAND_REG（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）选择是支持Clause22章模式还是Clause45章模式，Clause22章模式的PHY为GE PHY，Clause45章的PHY为XGE PHY。

11.1.6 软件指南

11.1.6.1 Clause22 PHY 寄存器读操作

通过配置相关寄存器可实现对PHY外部寄存器的读访问，具体配置说明如下：

- 步骤1** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤2** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的寄存器地址。
- 步骤3** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x01，选择接口模式为GE模式。
- 步骤4** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x10，表示读操作。
- 步骤5** 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动读操作。
- 步骤6** 查询寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）是否为0，如果为0表示读操作正常，读完成。



步骤7 查询寄存器MDIO_RDATA_REG[Mdio_Rdata]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x000C，bit[15:0]），MDIO读访问的读数据。

----结束

说明

上述步骤其实可以缩减为3个步骤，**步骤1**到**步骤5**都是对同一个寄存器进行配置的，可以一次配置完成。

11.1.6.2 Clause22 PHY 寄存器写操作

通过配置相关寄存器可实现对PHY内部寄存器的写访问，具体配置说明如下：

- 步骤1** 配置寄存器MDIO_WDATA[Mdio_Wdata]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0008，bit[15:0]），配置需要写入PHY寄存器的数值。
- 步骤2** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤3** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的寄存器地址。
- 步骤4** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x01，选择接口模式为GE模式。
- 步骤5** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x01，表示写操作。
- 步骤6** 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动写操作。
- 步骤7** 查询寄存器MDIO_STA_REG[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）是否为0，如果为0表示写操作正常，读完成。

----结束

说明

上述步骤其实可以缩减为3个步骤，**步骤2**到**步骤6**都是对同一个寄存器进行配置的，可以一次配置完成。

11.1.6.3 Clause45 PHY 寄存器读数据操作

Clause45的MDIO读数据操作分为两个MDIO接口操作，一个是写地址操作，一个是读数据操作。通过配置相关寄存器可实现对PHY内部寄存器的读数据操作，具体配置说明如下：



- 步骤1** 配置寄存器MDIO_ADDR[Mdio_Address]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0004，bit[15:0]），配置需要读取的PHY中DEV中寄存器的地址。
- 步骤2** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤3** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的设备地址。
- 步骤4** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x00，选择接口模式为XGE模式。
- 步骤5** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x00，表示写地址操作。
- 步骤6** 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动写地址操作。
- 步骤7** 查询寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）是否为0，如果为0表示写地址操作正常完成，进入下个步骤，不是0则在**步骤7**继续查询。
- 步骤8** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤9** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的设备地址。
- 步骤10** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x00，选择接口模式为XGE模式。
- 步骤11** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x11，表示读操作。
- 步骤12** 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动读操作。
- 步骤13** 查询寄存器MMDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）是否为0，如果为0表示读操作正常，读完成。
- 步骤14** 查询寄存器MDIO_RDATA_REG[Mdio_Rdata]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x000C，bit[15:0]），MDIO读访问的读数据。

----结束



说明

上述步骤其实可以缩减为6个步骤，[步骤2](#)到[步骤6](#)都是对同一个寄存器进行配置的，可以一次配置完成。[步骤8](#)到[步骤12](#)都是对同一个寄存器进行配置的，可以一次配置完成。

11.1.6.4 Clause45 PHY 寄存器写数据操作

Clause45的MDIO写数据操作分为两个MDIO接口操作，一个是写地址操作，一个是写数据操作。通过配置相关寄存器可实现对PHY内部寄存器的写数据操作，具体配置说明如下：

- 步骤1** 配置寄存器MDIO_ADDR[Mdio_Address]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0004，bit[15:0]），配置需要写的PHY中DEV中寄存器的地址。
- 步骤2** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤3** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的设备地址。
- 步骤4** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x00，选择接口模式为XGE模式。
- 步骤5** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x01，表示写地址操作。
- 步骤6** 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动写地址操作。
- 步骤7** 查询寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）是否为0，如果为0表示写地址操作正常完成，进入下个步骤，不是0则在[步骤7](#)继续查询。
- 步骤8** 配置寄存器MDIO_WDATA[Mdio_Wdata]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0008，bit[15:0]），配置需要写入PHY寄存器的数值。
- 步骤9** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤10** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的寄存器地址。
- 步骤11** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x00，选择接口模式为XGE模式。
- 步骤12** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x01，表示写操作。



步骤13 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动写操作。

步骤14 查询寄存器MDIO_STA_REG[Mdio_Sta]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0010，bit[0]）是否为0，如果为0表示写操作正常，读完成。

----结束

说明

上述步骤其实可以缩减为6个步骤，[步骤2](#)到[步骤6](#)以及[步骤9](#)到[步骤13](#)都是对同一个寄存器进行配置的，可以一次配置完成。

11.1.6.5 Clause45 PHY 寄存器地址自增读数据操作

Clause45中对MDIO的读操作有一种模式，地址自增读数据操作，如果采用这种模式，那么写地址操作只需要发送一次，然后可以发出多个自增读数据操作，PHY器件每次收到一个地址自增读操作之后，内部会把访问的寄存器地址加1。通过配置相关寄存器可实现对PHY内部寄存器的地址自增读数据操作，具体配置说明如下：

- 步骤1** 配置寄存器MDIO_ADDR[Mdio_Address]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0004，bit[15:0]），配置需要读取的PHY中DEV中寄存器的地址。
- 步骤2** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤3** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的设备地址。
- 步骤4** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x00，选择接口模式为XGE模式。
- 步骤5** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x00，表示写地址操作。
- 步骤6** 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动写地址操作。
- 步骤7** 查询寄存器MDIO_STA_REG[Mdio_Sta]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0010，bit[0]）是否为0，如果为0表示写地址操作正常完成，进入下个步骤，不是0则在[步骤7](#)继续查询。
- 步骤8** 配置寄存器MDIO_COMMAND[Mdio_Prtad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[9:5]），配置相应的外部PHY地址。
- 步骤9** 配置寄存器MDIO_COMMAND[Mdio_Devad]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[4:0]），配置相应的设备地址。



- 步骤10** 配置寄存器MDIO_COMMAND[Mdio_St]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）为0x00，选择接口模式为XGE模式。
- 步骤11** 配置寄存器MDIO_COMMAND[Mdio_Op]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[11:10]）为0x10，表示地址自增读操作。
- 步骤12** 配置寄存器MDIO_COMMAND[Mdio_Start]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[14]）为0x1，启动地址自增读操作。
- 步骤13** 查询寄存器MDIO_STA_REG[Mdio_Sta]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0010，bit[0]）是否为0，如果为0表示读操作正常，读完成。
- 步骤14** 查询寄存器MDIO_RDATA_REG[Mdio_Rdata]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x000C，bit[15:0]），MDIO读访问的读数据。
- 步骤15** 继续读取下一个寄存器地址（地址为之前访问的寄存器值+1）的数据，从**步骤8**开始循环到**步骤15**。

----结束

说明

上述非循环步骤（**步骤1**到**步骤14**）其实可以缩减为6个步骤，**步骤2**到**步骤6**都是对同一个寄存器进行配置的，可以一次配置完成。**步骤8**到**步骤12**都是对同一个寄存器进行配置的，可以一次配置完成。

11.1.6.6 MDIO 使用约束

MDIO在每次PHY器件的操作结束后会自动清除模块内部的状态寄存MDIO_STA_REG，因此在使用MDIO模块配置PHY器件是，建议使用核进行配置和状态读取，且在配置下发之后理解进行状态回读，不要等到上一次配置完毕再去查询状态，这时会存在前一次操作状态被清掉的可能。

11.1.7 初始化

11.1.7.1 MDIO 管脚复用配置

MDIO管脚与GPIO管脚复用，使用MDIO管脚时，需要配置复用关系：配置寄存器IOMUX（基地址为0xFA86_0000，偏移地址为0x1F0/0x1F4、0x1F8/0x1FC、0x200/0x204）为0，配置管脚功能为MDIO0_CLK/MDIO0_DATA、MDIO1_CLK/MDIO1_DATA、MDIO2_CLK/MDIO2_DATA。

11.1.7.2 模块初始化

MDIO初始化步骤如下：

对于MDIO自动查询功能，需要做初始化操作：

- 步骤1** 配置寄存器MDIO_COMMAND（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[17]，bit[15]）选择接口的时钟。



- 步骤2** 配置寄存器MDIO_COMMAND（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[13:12]）选择接口模式为GE模式。
- 步骤3** 配置寄存器MDIO_AUTO_CFG_0（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0020~0x003C），配置自动查询模式下PHY地址值。
- 步骤4** 配置寄存器MDIO_COMMAND [Mdio_AutoScan]（MDIO0基地址0xFA44_0000/MDIO1基地址0xFA45_0000/MDIO2基地址0xFA46_0000，偏移地址为0x0000，bit[16]），配置是否使能自动查询功能。
- 结束

11.1.7.3 时钟门控

上电时MDIO工作时钟默认为打开状态。在软件完成当前数据传输且未启动新的数据传输的情况下关断MDIO工作时钟。

11.1.7.3.1 打开时钟

具体配置步骤如下：

- 步骤1** 读PERI_SUBCTRL寄存器SC_MDIO_CLK_ST，获取当前时钟门控信息。
- 步骤2** 若SC_MDIO_CLK_ST[0]为0，则进入**步骤3**；SC_MDIO_CLK_ST[0]为1，则表示MDIO0时钟已经打开。
- 步骤3** 配置SC_MDIO_CLK_EN[0]为1，打开MDIO0模块的时钟门控。
- 结束

11.1.7.3.2 关闭时钟

具体配置步骤如下：

- 步骤1** 读PERI_SUBCTRL寄存器SC_MDIO_CLK_ST，获取当前时钟门控信息。
- 步骤2** 若SC_MDIO_CLK_ST[0]为1，则进入**步骤3**；SC_MDIO_CLK_ST[0]为0，则表示MDIO0时钟已经关闭。
- 步骤3** 配置SC_MDIO_CLK_EN[0]为0，关闭MDIO0模块的时钟门控。
- 结束

11.1.7.4 复位

软件在MDIO空闲时（所有访问完成），可以通过配置PERI_SUBCTRL寄存器对MDIO模块进行复位；具体流程如下：

- 步骤1** 配置寄存器SC_MDIO_RESET_REQ为1，复位MDIO0。
- 步骤2** 查询寄存器SC_MDIO_RESET_ST为1，表示MDIO0处于复位状态。
- 步骤3** 配置寄存器SC_MDIO_RESET_DREQ为1，解复位MDIO0。
- 步骤4** 查询寄存器SC_MDIO_RESET_ST为0，表示MDIO0处于解复位状态。
- 结束



12 PMC

12.1 关键术语

如表12-1的术语将在本章节表示特殊的含义。

表 12-1 关键术语描述

术语	描述
PM C	Phosphor Multi-Channel Controller，PMC和TDM是功能相同的同一个模块。
BD	Buffer Descriptor，缓存描述符，CPU用描述符和TDM交互接收和发送的数据。
TS M	Time-Slot Map，TSM对TDM总线的时隙映射，完成时隙的有效性以及时隙到TSA的映射关系。
TS A	Time-Slot Allocate，TSA将TSM映射之后的时隙与HDLC通道关联，完成时隙到通道的分配。

12.2 概述

PMC（PhosphorMulti-channelController）是基于TDM总线传输HDLC报文的多通道控制器，实现时分复用和HDLC协议处理。PMC最大可以支持128个接收通道和128个发送通道。每个通道可以独立的配置参数。

PMC使用描述符管理HDLC报文的存储。CPU通过中断和描述符获取当前报文的信息，同时通过配置描述符来完成HDLC报文的传输。

TDM总线可以灵活的配置各种时分复用方式，提供满足系统要求的复用方式。TDM时隙分配到HDLC通道映射都可以根据应用场景进行配置，构建业务传输通道。

Hi309a中集成4个PMC控制器，通过功能组合，可以支持2路16M的TDM通道，或者2路32M的TDM通道。



12.2.1 309a 特性

每个PMC具有以下特性：

- 芯片提供2个TDM总线接口：仅支持4线模式。内部集成了4个PMC模块。在总线数据传输速率为8.192Mbps、16.384Mbps时，有PMC0/2实现两个TDM总线接口功能。当传输速率在32.768Mbps时，由PMC0/1组合实现一个TDM接口的传输，PMC2/3组合实现另一个TDM接口的传输。
- 仅支持从芯片外部提供总线时钟和帧同步信号。
- 支持总线数据传输速率为8.192Mbps、16.384Mbps或32.768Mbps。其中32.768Mbps由两个PMC模块工作在16.384Mbps下拼接实现。
- 支持TDM总线单路接收/发送各自最大128个通道进行数据传输。
- 支持TDM总线每个时隙的有效数据位数可配置：0、1、2、4、6、8（单位：bit）。
- 支持外环回模式和基于通道的内环回模式。在环回模式下，TDM总线必须工作在4线模式，即发送和接收方向的时序相同，时隙分配也一致的模式。
- 支持帧同步信号（FSP）的异常检测，并通过寄存器记录错误信息，供给CPU查询。在总线的FSP信号恢复正常后，逻辑可以自动检测到，并重新完成帧同步，为数据传输做好准备。
- 支持HDLC协议处理。
 - 发送支持帧头/尾标志的生成和插入。
 - 接收支持帧头/尾标志的删除。
 - 发送支持FCS（0位/16位/32位可配）的生成插入，使用16/32位CRC-CCITT。
 - 接收支持FCS（0位/16位/32位可配）的校验删除。
 - 发送支持“0”插入。
 - 接收支持“0”删除。
 - 发送时支持空闲时插入“flag”或“idle”可配。
 - 接收支持帧间共享一个“flag”。
 - 接收支持“flag”共享一个0。
 - 支持帧长检测，包括超长帧和超短帧检测。
 - 支持Abort帧的发送和接收。
 - 支持上行时中止传输和当前帧结束后中止传输。
 - 支持自动过滤小于1个字节的HDLC碎帧（该帧长不包括CRC字节）。
- 支持单路128个HDLC接收通道和128个发送通道，每个通道可以单独使能和禁止。
- 支持中断方式和查询方式，便于软件完成报文的处理。
 - 接收超长帧中断。
 - 发送帧完成中断。
 - 接收帧完成中断。
 - 发送BD没有准备好中断。
 - 接收BD没有准备好中断。
 - FIFO溢出中断。
 - 总线返回中断。



- 偶校验错误中断。
- 支持RAM的偶校验。

12.3 接口信号

PMC接口信号描述如表12-2所示（Hi309a仅支持4线模式）。

表 12-2 PMC 接口信号描述

名称	方向	描述	对应管脚
TDM0_RXCLK	I	TDM0接收方向总线时钟信号，在4线模式下，发送方向也使用该时钟信号。	TDM0_RXCLK
TDM0_RXFSP	I	TDM0接收方向帧同步信号，在4线模式下，发送方向也使用该帧同步信号。	TDM0_RXFSP
TDM0_RX	I	TDM0接收数据线。	TDM0_RX
TDM0_TX	O	TDM0发送数据线。	TDM0_TX
TDM1_RXCLK	I	TDM1接收方向总线时钟信号，在4线模式下，发送方向也使用该时钟信号。	TDM1_RXCLK
TDM1_RXFSP	I	TDM1接收方向帧同步信号，在4线模式下，发送方向也使用该帧同步信号。	TDM1_RXFSP
TDM1_RX	I	TDM1接收数据线。	TDM1_RX
TDM1_TX	O	TDM1发送数据线。	TDM1_TX

12.4 功能描述

12.4.1 应用框图

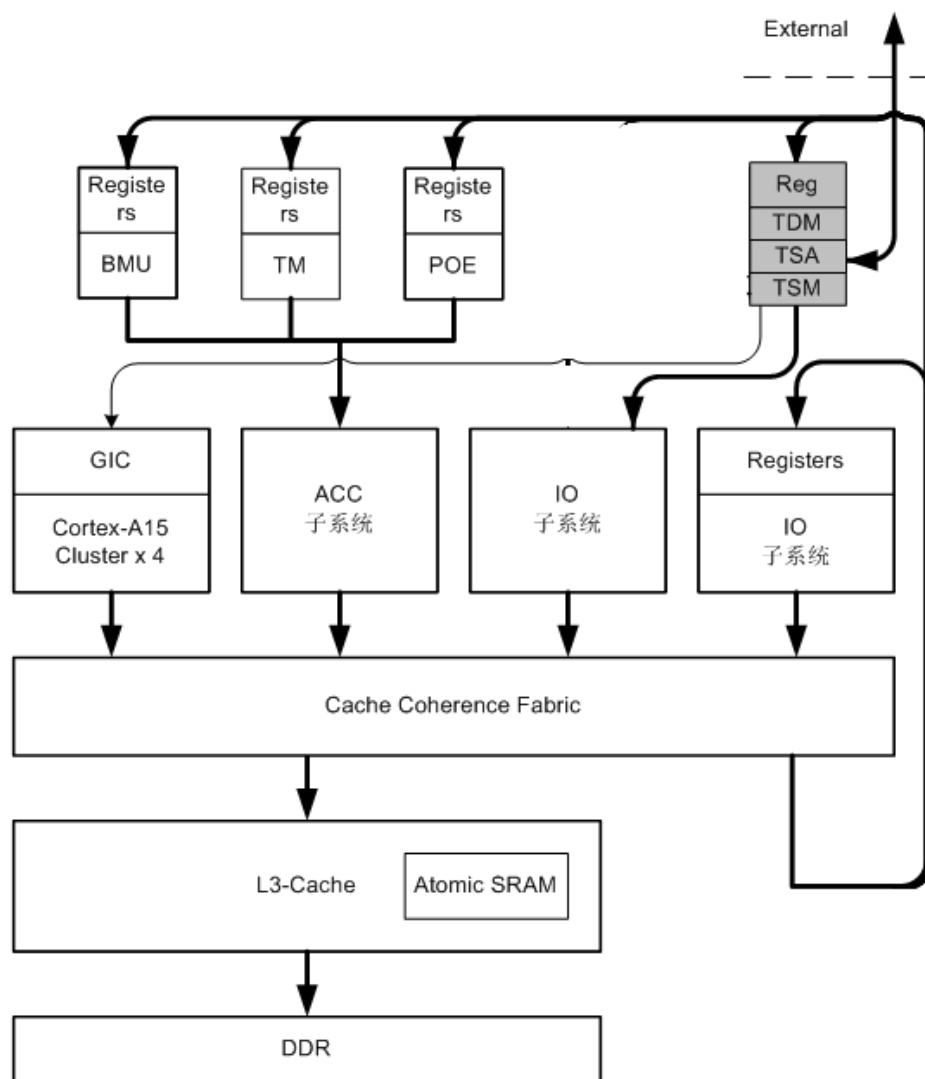
PMC的应用如图12-1所示。PMC模块包括TSM、TSA、TDM（TX/RX）三部分，各个模块的功能分别如下：

1. TSM对TDM总线的时隙映射，完成时隙的有效性以及时隙到TSA的映射关系。
2. TSA表可以将总线上的单个时隙或多个时隙映射到同一个TSM表项。

 说明

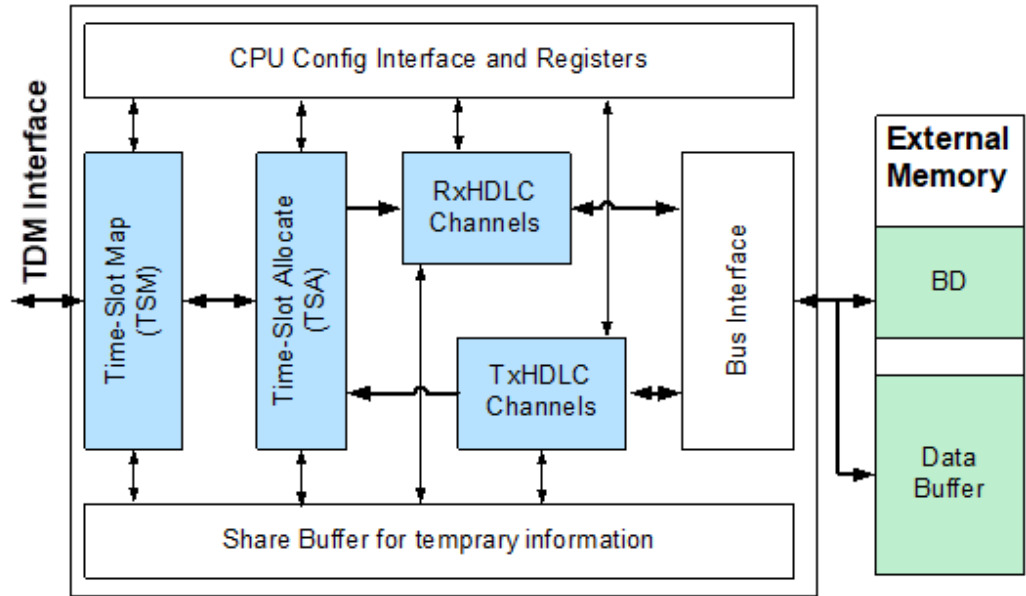
- 推荐在PMC的应用场景中，TDM总线的每个时隙都需要配置一个TSM表项与之对应。配置该时隙有效性，映射关系以及工作模式。
 - TSA将映射之后的时隙与HDLC通道关联，完成时隙到通道的分配。TSA表可以配置为单个TSA表项或多个TSA表项同时关联到一个HDLC通道。TSA表项还可以实现单个时隙中部分位有效的传输配置。
- 推荐在多个时隙分配到同一个HDLC通道的场景下，先通过多个TSM表项指向同一个TSA表项，再通过该TSA表项指向HDLC逻辑通道。
 - HDLC通道接收和发送控制逻辑（TX/RX），实现将TDM总线接收到的数据通过BD（Buffer Descriptor）存放到DDR中；或者将DDR中的数据从TDM总线发送出去。

图 12-1 PMC 应用图



12.4.2 TDM 模块结构

图 12-2 PMC 结构框图



12.4.3 相关模块

在系统中，TDM模块会和下面多个模块进行交互，分别描述如下：

- CPU：CPU用于配置和查询PMC的寄存器，PMC如果出现传输错误或超时，通过中断信号上报CPU处理；
- DDR：上行数据流方向，CPU将裸数写入DDR，并编写BD描述数据结构，PMC获取BD后从DDR读取数据组帧，发送到接口；下行数据流方向，PMC接收帧数据并解帧，然后获取BD，将数据恢复为裸数写入DDR。
- GIC：PMC中断处理。

12.4.4 板级应用

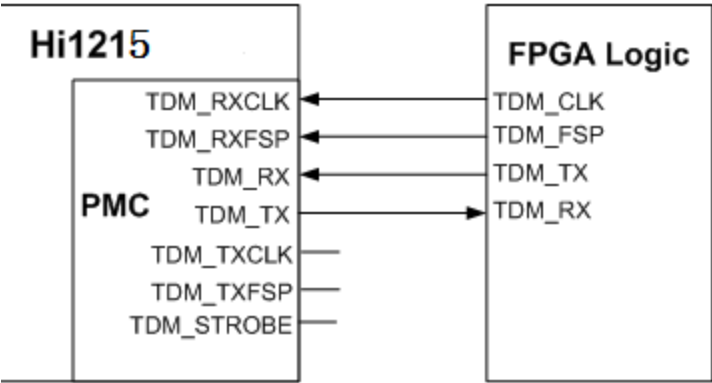
12.4.4.1 互联模式

通信处理器的PMC接口有2种主要的应用场景：

- 芯片和FPGA逻辑直接通信的互联方式。
- 多个控制芯片和FPGA逻辑通信的互联方式（需要用到TDM_STROBE信号）。

在大多数的应用场景下，都会使用第一种互联方式（Hi309a只支持第一种互联方式），其连接如图12-3所示。

图 12-3 PMC 直连模式之 4 线连接图



连接说明：

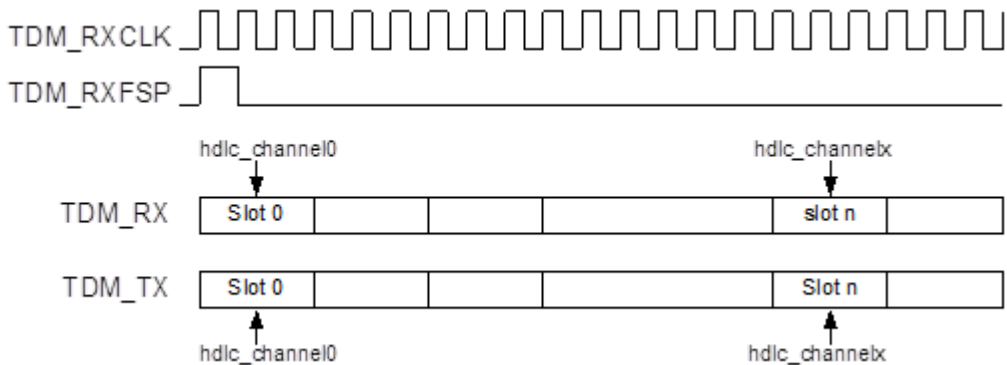
1. PMC的总线时钟和帧同步信号都由FPGA逻辑提供。在4线模式下，数据的发送和接收都共享总线时钟和帧同步信号。
2. TDM_TXCLK、TDM_TXFSP、TDM_STROBE信号在该模式下都不使用，可以悬空。

12.4.4.2 时分复用模式

PMC接口可以支持多种时分复用模式，以满足产品不同应用场景的需求。根据PMC的实现方式，可以将时分复用模式分为4种类型：同相共享模式、独立共享模式、异步共享模式、异步模式。

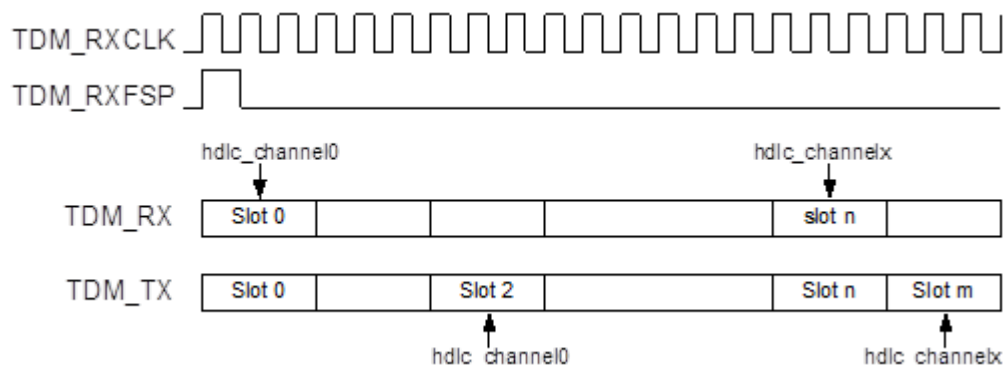
同相共享模式示意图如图12-4所示，该模式要求PMC互联方式为4线模式。其特点是PMC的发送和接收共享总线时钟和帧同步信号，并且相同的时隙分配给同一个HDLC通道。

图 12-4 PMC 同相共享模式



独立共享模式示意图如图12-5所示，该模式要求PMC互联方式为4线模式。其特点是PMC的发送和接收共享总线时钟和帧同步信号，发送方向时隙的分配和接收方向时隙的分配相互独立。

图 12-5 PMC 独立共享模式



PMC包含了TSM（Time-SlotMap）和TSA（Time-SlotAllocate）两个配置表项，软件可以通过配置TSM和TSA，实现上面4种传输模式。

PMC还支持外环回测试模式和基于通道的内环测试模式。在外环回测试模式下，PMC将从TDM_RX接收到的数据直接环回到TDM_TX数据线上。在内环回模式下，PMC按照TSM发送表项中配置，将配置时隙的数据从TDM_TX环回到TDM_RX通路上。

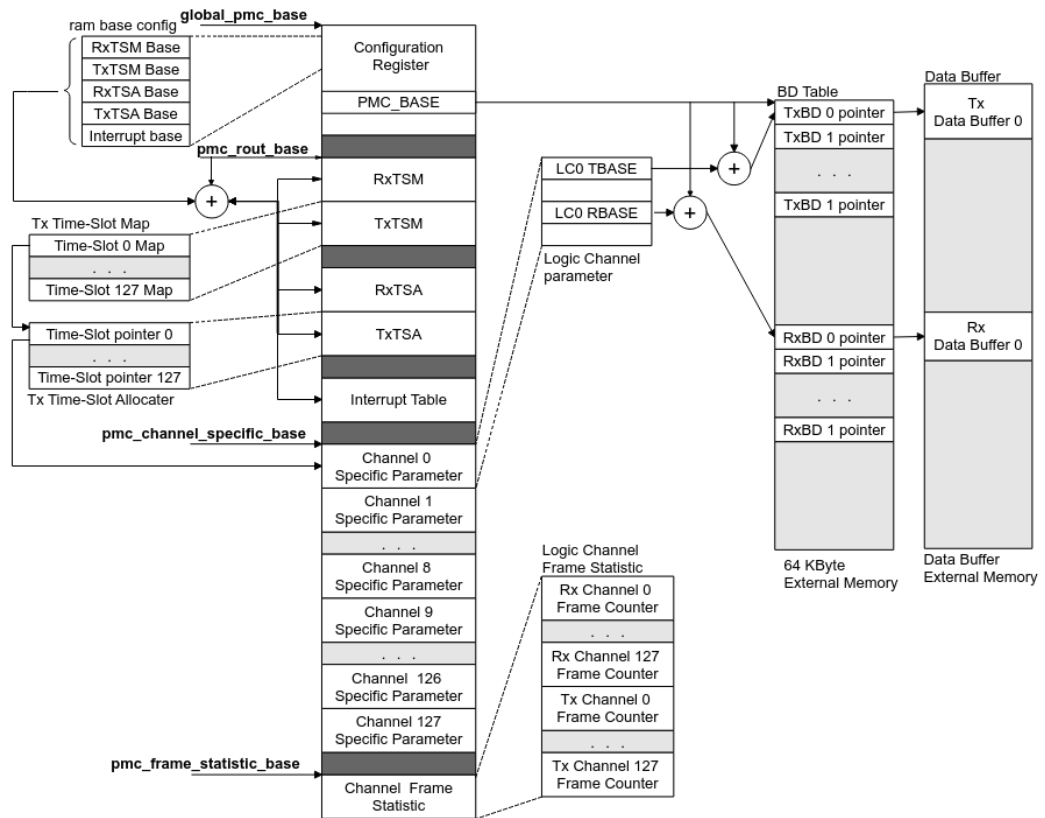
须知

环回测试模式要求PMC工作在同相共享模式。

12.4.5 存储结构

PMC存储结构图如图12-6所示，该图示意了PMC在数据处理流程中，如何进行数据的传输以及地址的寻址。

图 12-6 PMC 存储结构图



- **PMC配置寄存器（ ConfigurationRegister ）**

PMC配置寄存器完成对PMC工作模式的配置以及对数据流中，地址空间的配置。包括TSM表基地址配置，TSA表基地址配置，中断向量表基地址配置和BD表基地址配置。

- **时隙映射表（ TSM ）**

TSM表定义了TDM总线时隙是否有效，和指向TSA表项的地址。TDM总线上的时隙根据TSM表项就可以寻址到对应的TSA表项。所以软件在配置时需要关注TSM表项和TSA表项的对应关系。

- **时隙分配表（ TSA ）**

TSA表定义了每个时隙中有效的数据位数，以及指向HDLC逻辑通道的指针。接收方向每个时隙的数据经过TSM路由后，经过TSA处理，将有效的数据送入到对应的逻辑通道，接收通道控制器再完成报文数据的存储。发送方向亦然。

- **中断向量表（ InterruptTable ）**

中断向量用于存储PMC在HDLC报文接收和发送过程中，产生的中断信息。在每个HDLC帧收发完成时，PMC都会根据软件配置的中断模式，将中断送入到中断向量表中。CPU就可以根据中断信息完成和PMC的通信，进行业务的传输控制。

- **通道特性参数（ ChannelSpecificparameter ）**

通道特性参数空间，主要完成对每个通道独有的参数进行配置。配置参数的描述可以参考“[通道参数配置](#)”章节的描述。



须知

BD空间和数据缓存空间都需要分配在non-cacheable的外存空间中。

- BD空间（BufferDescriptor）
BD表是PMC所有HDLC通道BD的集合。PMC最多支持64kByte的BD空间，即8K个BD。每个BD中包含了32bit的物理地址，指向一段数据缓存区间。BD的描述请参考“[缓存描述符](#)”章节的描述。
- 数据缓存空间（DataBuffer）
数据缓存空间用于存放HDLC报文数据。数据缓存空间通过BD进行管理。

12.4.6 存储原理

12.4.6.1 寄存器配置

PMC配置寄存器完成对PMC工作模式的配置以及对各个参数空间的基地址配置TSM表地址配置。

PMC_TSMCFG寄存器定义如[表12-3](#)所示。该寄存器完成PMC的TSM表基地址配置。

表 12-3 PMC_TSMCFG 寄存器描述

位	名称	描述
[31:25]	-	保留。
[24]	tx_ramsel	发送方向TSM和TSA表选择。 0：主配置表； 1：备份配置表。
[23:20]	tsm_tx_shbase	TDM发送方向备份TSM的基地址。 0x0 ~ 0xF：起始偏移地址为n*128（Byte）。 例如，0x2表示：物理地址为PMC_REGBASE + 0x1000 + 2*128。
[19:16]	tsm_tx_base	TDM发送方向TSM的基地址。 0x0 ~ 0xF：起始偏移地址为n*128（Byte）。 例如，0x2表示：物理地址为PMC_REGBASE + 0x1000 + 2*128。
[15:9]	-	保留。
[8]	rx_ramsel	接收方向TSM和TSA表选择。 0：主配置表； 1：备份配置表。



位	名称	描述
[7:4]	tsm_rx_shbase	TDM接收方向备份TSM的基地址。 0x0 ~ 0xF：起始偏移地址为n*128（Byte）。 例如，0x2表示：物理地址为PMC_REGBASE+0x1000+2*128。
[3:0]	tsm_rx_base	TDM接收方向TSM的基地址。 0x0 ~ 0xF：起始偏移地址为n*128（Byte）。 例如，0x2表示：物理地址为PMC_REGBASE+0x1000+2*128。

PMC_TSMCFG[_base]配置为0x0，即物理地址为：PMC_REGBASE+0x1000。

PMC_TSMCFG[tsm_rx_shbase]的配置需要根据当前的总线工作频率进行计算得到。
TSM每个表项为2字节的参数，所以其计算公式如下：

$$TSM_RX_SHBASE=2\times TS_NUM/128+0=2\times 256/128=4$$

PMC_TSMCFG[tsm_rx_shbase]配置为4，即物理地址为PMC_REGBASE+0x1000+4*128

同理可知道PMC_TSMCFG[tsm_tx_base]的配置公式如下：

$$TSM_TX_BASE=2\times TS_NUM/128+4=8$$

PMC_TSMCFG[tsm_tx_base]配置为8，即物理地址为PMC_REGBASE+0x1000+8*128

PMC_TSMCFG[tsm_tx_shbase]的配置公式如下：

$$TSM_TX_BASE=2\times TS_NUM/128+8=12$$

PMC_TSMCFG[tsm_tx_shbase]配置为0xC，即物理地址为PMC_REGBASE+0x1000+12*128

- TSA表地址配置
PMC_TPTRCFG和PMC_RPTRCFG寄存器定义了TSA表基地址的配置，其寄存器描述如表12-4和表12-5所示。

表 12-4 PMC_TPTRCFG 寄存器描述

位	名称	描述
[31:16]	tsa_tx_shbase	发送方向备份TSA表的基地址。 0x0000 ~ 0xFFFFE：偏移地址。 注意：最低1bit必须配置为0。网表切换模式下，和TxTSM表一起切换。
[15:0]	tsa_tx_base	发送方向主TSA表的基地址。 0x0000 ~ 0xFFFFE：偏移地址。 注意：最低1bit必须配置为0。



表 12-5 PMC_RPTRCFG 寄存器描述

位	名称	描述
[31:16]	tsa_rx_shbase	接收方向备份TSA表的基地址。 0x0000 ~ 0xFFFFE：偏移地址。 注意：最低1bit必须配置为0。网表切换模式下，和RxTSM表一起切换。
[15:0]	tsa_rx_base	接收方向TSA表的基地址。 0x0000 ~ 0xFFFFE：偏移地址。 注意：最低1bit必须配置为0。

PMC在需要动态删除或增加通道的场景下，TSM表从主表切换到备份表时，TSA表同时也从主表切换到备份表。即在这种应用场景下，TSA表的主和备份的使用关系始终和TSM的使用关系一致。

以PMC工作在16.384MHz总线传输模式为例，对TSA地址空间配置进行说明。在该模式下每个TDM帧包含了256时隙。参考[存储结构](#)章节的描述，可以知道TSA和TSM放在连续的内部存储空间，所以TSA表的基地址需要放在TSM表之后。

须知

PMC最大支持128个发送通道和128个接收通道，所以这里的单个TSA表项最大支持128个。在配置PMC_TPTRCFG和PMC_RPTRCFG寄存器时，需要注意到这个约束条件。也就是说，在TDM总线速率为16.384MHz的场景下，不能实现每个时隙对应一个独立的HDLC逻辑通道。

PMC_RPTRCFG[tsa_rx_base]配置为0x1A00，PMC_RPTRCFG[tsa_rx_shbase]配置为0x1B00。

PMC_TPTRCFG[tsa_tx_base]配置为0x1C00，PMC_TPTRCFG[tsa_tx_shbase]配置为0x1B00

- 中断向量表地址配置

PMC_INTBASE定义了PMC中断向量表的地址信息。寄存器的定义如[表12-3](#)所示。

表 12-6 PMC_INTBASE 寄存器描述

位	名称	描述
[31:16]	int_base	中断向量表入口地址。
[15:0]	Reserved	保留。

以PMC工作在16.384MHz总线传输模式为例，对中断地址空间配置进行说明。在该模式下每个TDM帧包含了256时隙。参考[存储结构](#)章节的描述，可以知道中断向量表放在TSA表之后。



PMC_INTBASE[int_base]配置为0x1E00。

须知

PMC内部分配给TSM、TSA和中断向量表的总存储空间为4096字节。所以在对整个TSM、TSA和中断向量表进行空间分配时，需要考虑该约束条件。

- BD表地址配置
PMC_BASE寄存器定义了PMC的BD表在外存空间的基地址。寄存器定义如表12-7所示。

表 12-7 PMC_BASE 寄存器描述

位	名称	描述
[31:0]	pmc_base	BD表的基地址，单位字节。 注意：软件配置时，该值的最低3bit必须为0，即按照8字节对齐。

须知

PMC最大只能支持连续的64K字节空间存储BD表，所以在配置和使用BD表时，需要注意该约束条件。

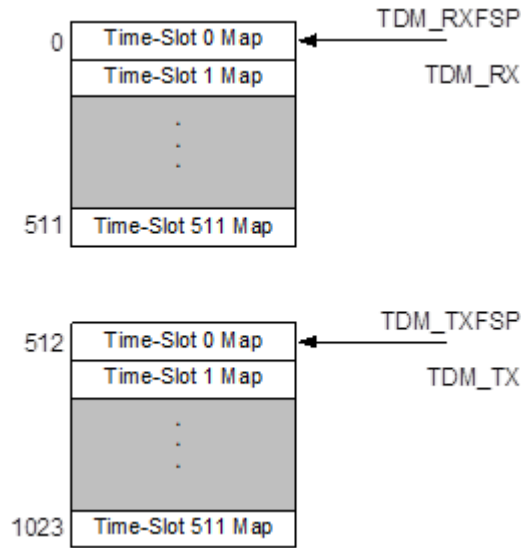
完成对上述寄存器的配置后，PMC的存储空间的划分就完成了。用户可以结合存储结构章节的示意图和软件配置描述，理解PMC的存储结构，最终可以根据实际的应用场景来配置PMC的存储结构。

12.4.6.2 TSM 配置

TSM表定义了TDM总线的时隙属性，完成数据通路的第一级时分复用控制。TSM表可以工作在两种模式：静态路由模式和动态路由模式。

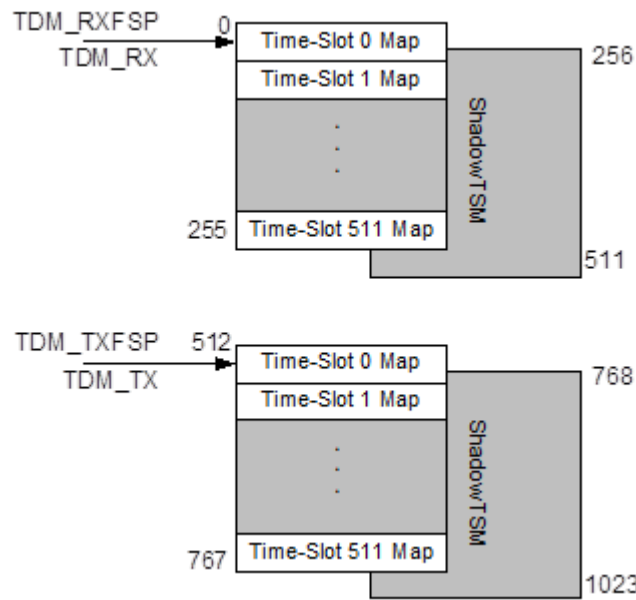
静态路由模式下，TSM表可最大包含512个发送表项和512个接收表项。这种模式只能应用在路由表不需要切换的场景。在实际的应用场景中，用户可以根据场景来配置接收和发送方向的表项。TSM表项静态模式示意图如图12-7所示。

图 12-7 TSM 静态路由模式



动态路由模式下，TSM表最大只能同时使用256个发送表项和256个接收表项。这种路由模式，可以支持多种场景的应用，特别是在业务需要动态增加HDLC逻辑通道和删除HDLC逻辑通道的场景。在这种场景下，TDM总线需要在不中断业务的情况下，新建一条物理通路来传输HDLC报文数据，使用动态TSM表是完成这项功能的必要条件。在实际的应用场景中，用户可以场景来配置接收和发送方向的表项个数。TSM表项动态路由模式示意图如图12-8所示。

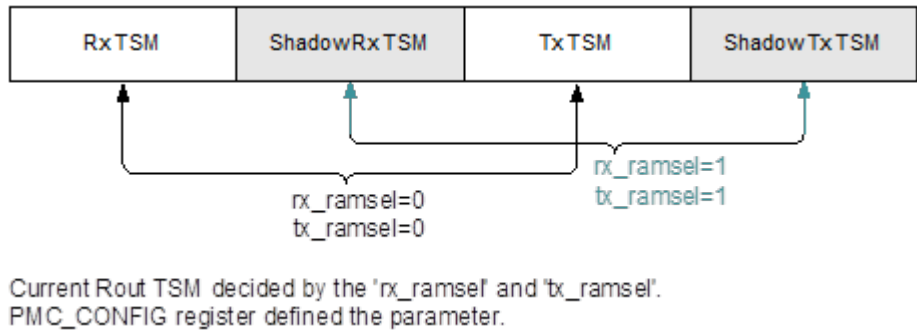
图 12-8 TSM 表动态模式图



在动态路由模式下，软件需要先将未使用的TSM表配置为新的应用表项，然后配置PMC_CONFIG[rx_ramsel]和PMC_CONFIG[tx_ramsel]选择新的TSM表作为当前的工作路由表。PMC_CONFIG配置值和TSM表的对应关系如图12-9所示，具体的配置流程可以参考网表切换章节的描述。



图 12-9 TSM 动态模式切换控制示意图



TSM表项定义一个TDM时隙的属性，包括该时隙是否进行数据传输，发送方向是否使用TDM_STROBE等等。表项的基址参考“寄存器配置”章节的描述，其定义如表12-8所示。

表 12-8 TSM 表项描述

位	名称	描述
[15]	wrap	最后一个时隙路由配置。 0：不是最后一个； 1：最后一个路由时隙配置。 内部逻辑在检测到该配置后，自动环回到表头开始读取表项。
[14]	valid	数据有效配置。 0：无效； 1：有效。
[13:12]	mode	时隙模式选择。 00：Normal模式； 10：LoopBack模式； 其他：保留。 注意： 仅当发送方向配置了该时隙的LoopBack属性后，该时隙将会被环回到接收方向。
[11]	strobe_en	STROB使能。 0：禁止； 1：使能。
[10:8]	-	保留。



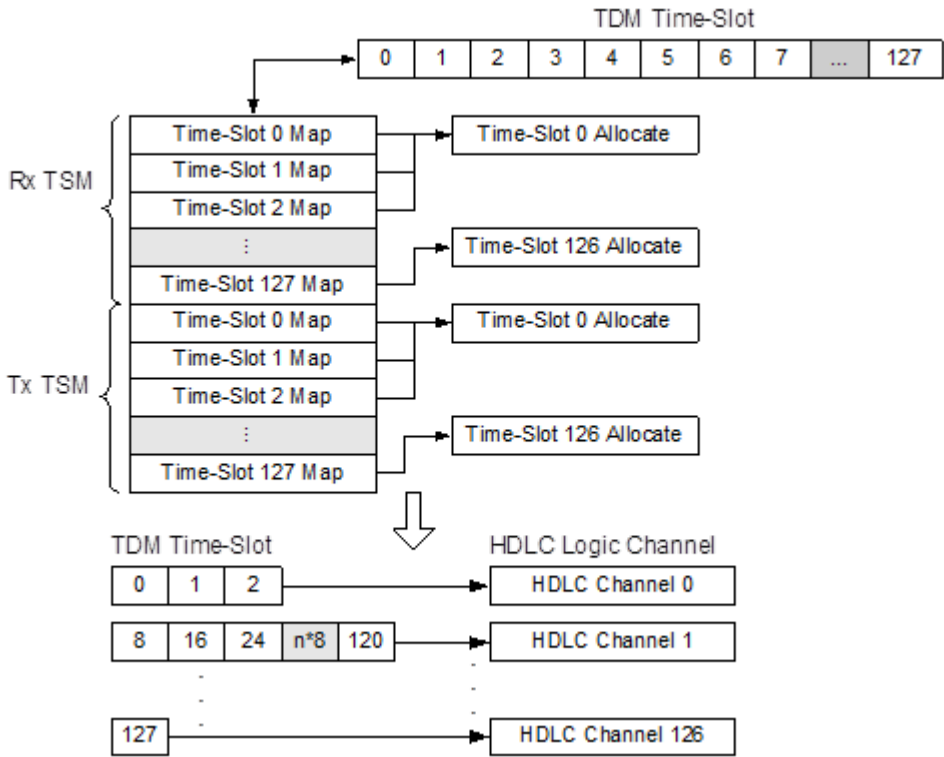
位	名称	描述
[7:0]	ts_map	TSA表偏移指针。 0x00~0x7F：第n个TSA表项。 0x80~0xFF：保留。 对于接收方向的表项，该配置值需要加上RxTSA的基地，得到RxTSA的表项地址。 对于发送方向的表项，该配置值需要加上TxTSA的基地址，得到TxTSA的表项地址。

须知

- 配置TSM表项时，CPU只能按照16bits的数据模式对表项进行配置。
- TSM表项中的环回模式，当且仅当对发送方向的表项进行配置才能生效。
- TSM表项中，TSM_ENTRY[wrap]只能在TSM表的最后一个表项配置有效，用来指示下一个TSM表项是TSM的第一个表项。

现以TDM总线工作在8.192MHz的模式为例，图12-10示意了TSM表在数据流中完成的工作。

图 12-10 TSM 应用示意图



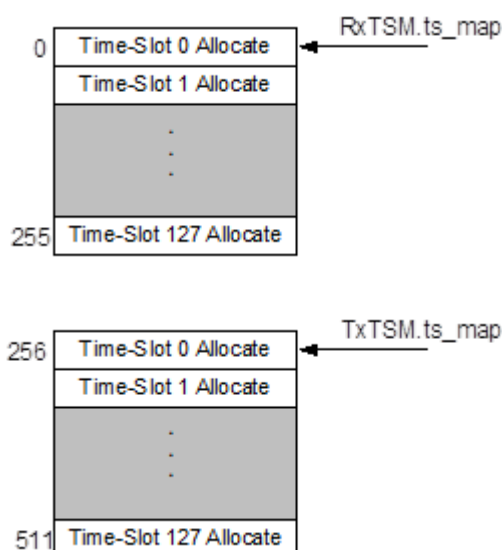
在TSM的配置中，可以将多个时隙同时映射到一个逻辑通道，由此来实现大数据流的传输。

12.4.6.3 TSA 配置

TSA表实现将TSM映射后的时隙与HDLC逻辑通道关联，是数据通路上的第二级时分复用控制。TSA表可以工作在两种模式：静态分配模式和动态分配模式。

静态分配模式下，TSA表可最大包含256个发送表项和256个接收表项。这种模式只能应用在路由表不需要切换的场景。在实际的应用场景下，用户可以场景来配置接收和发送方向的表项。TSA表项静态模式示意图如图12-11所示。

图 12-11 TSA 表静态模式



动态分配模式下，TSA表最大只能同时使用128个发送表项和128个接收表项。TSA表的动态模式需要和TSM表的动态模式一起工作，来完成整个数据通路的通道增加和删除。在实际的应用场景中，用户可以场景来配置接收和发送方向的表项个数。TSA表项动态分配模式示意图如图12-12所示。

图 12-12 TSA 表动态模式

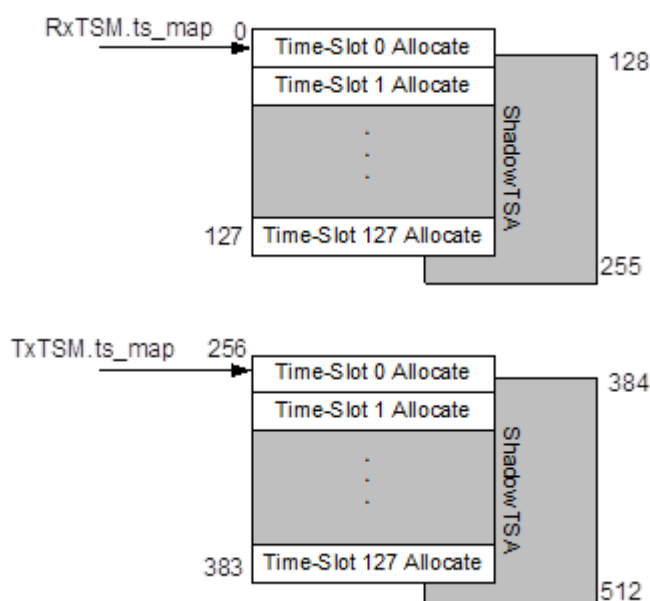
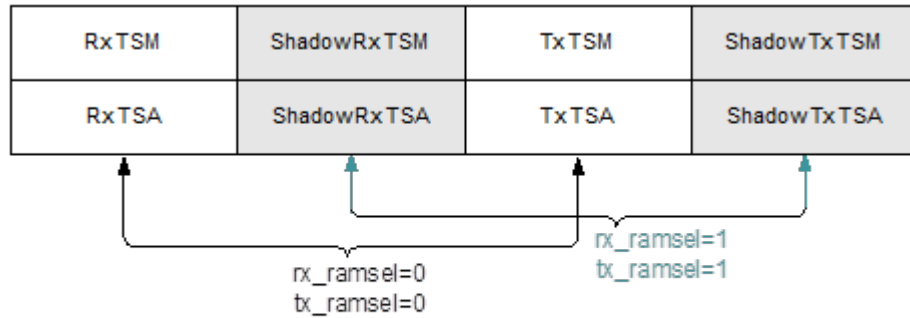


图12-13示意了在动态切换网表的场景中，TSM和TSA表的切换方式。通过配置PMC_CONFIG[rx_ramsel]和PMC_CONFIG[tx_ramsel]选择新的TSM和TSA表作为当前的数据路由表，完成数据通路的动态改变。

图 12-13 TSM 和 TSA 表动态切换模式示意图



Current Rout TSM/TSA decided by the 'rx_ramsel' and 'tx_ramsel'.
PMC_CONFIG register defined the parameter.

TSA的表项的定义如表12-9所示，其基地址参考“寄存器配置”章节的描述。

表 12-9 TSA 表项描述

位	名称	描述
[15]	Reserved	保留。
[14:8]	ch_ptr	逻辑通道指针。 每个逻辑通道分配了64字节，对应128个逻辑通道。
[7:0]	mask	时隙位使能。 0：屏蔽该位； 1：使能该位。 注意： 当发送方向该时隙的[strobe_en]使能时，该信息同时用于指示当前时隙的TDM_STROBE信息。

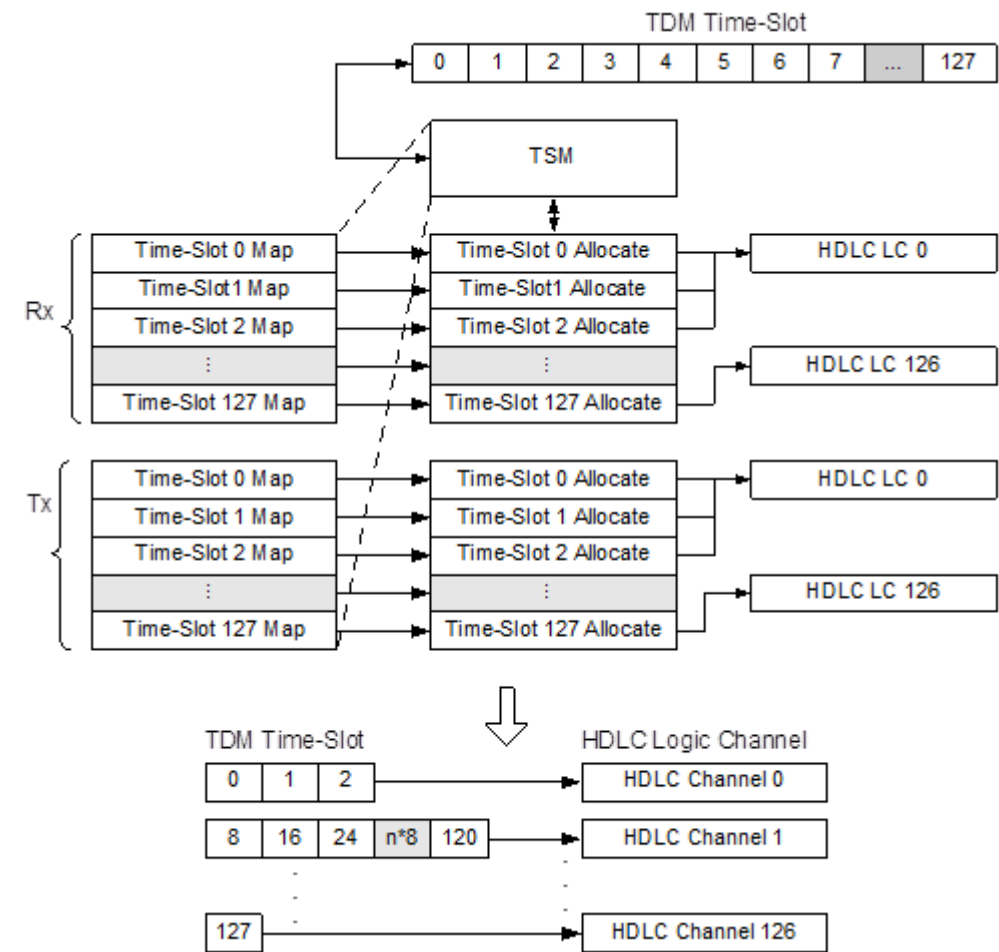
须知

1. 配置TSA表项时，CPU只能按照16bits位宽模式对表项进行配置。
2. 在配置TSA表项[ch_ptr]域时，只需要配置逻辑通道号即可。内部逻辑会自动在配置的[ch_ptr]值后补0b000000，用于指向每个通道特性参数的起始地址。

现以TDM总线工作在8.192MHz的模式为例，图12-14示意了TSA表在数据流中完成的工作。



图 12-14 TSA 应用示意图



12.4.6.4 通道参数配置

PMC的每个HDLC通道都有特性参数，该参数配置了HDLC的工作模式、HDLC协议处理方式等。每个通道的特性参数定义如表12-10所示。

表 12-10 HDLC 通道参数表

偏移地址	名称	描述
0x0	TX_ADDR	通道发送方向BD指针。该指针是基于PMC_BASE的偏移地址。
0x4	CHNL_MODE	通道模式配置寄存器。
0x8	TXSTATE	发送状态机状态信息。
0xC	TX_DPTR	当前发送数据的物理地址。 注意：由于发送会预取帧，发送指针会比BD表的地址要大。



偏移地址	名称	描述
0x10	TUPACK	正在发送的数据。TDM总线的一个时隙为8bit，所以每个时隙只传输1个字节，该寄存器用于缓存32bits的发送数据。
0x14	ZISTATE	发送方向，HDLC协议处理缓存。
0x18	TCRC	发送暂存CRC值。
0x1C	TX_TEMP	发送中断和状态控制。
0x20	RX_ADDR	通道接收方向BD指针。该指针是基于PMC_BASE的偏移地址。
0x24	RX_MODE	接收方向模式配置。
0x28	RXSTATE	接收状态机状态信息。
0x2C	RX_DPTR	当前接收数据的物理地址。
0x30	RPACK	缓存的接收数据。
0x34	ZDSTATE	插0状态。
0x38	RCRC	发送暂存CRC值。
0x3C	RX_TMP	内部使用的计数器缓存。

须知

PMC的通道参数寄存器都需要按照32bit的方式进行操作，否则不能写入或者读出。

- TX_ADDR配置描述
TX_ADDR用于配置每个HDLC通道的发送方向BD表的基地址。寄存器描述如表12-11所示。

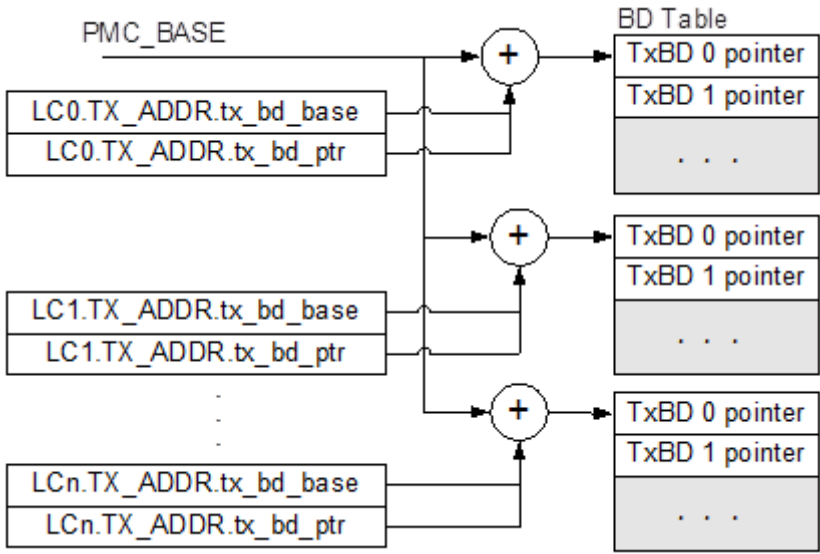
表 12-11 TX_ADDR 寄存器描述

位	名称	描述
[31:16]	tx_bd_base	发送方向BD表的基地址，单位字节。需要按照DWORD（8Byte）对齐。 PMC_BASE+tx_bd_base为外存物理地址。
[15:0]	tx_bd_ptr	当前使用的发送BD的地址，单位Byte。 PMC_BASE+tx_bd_ptr为外存物理地址。

系统上电或者软复位之后，初始化PMC时，需要将每个通道的TX_ADDR地址配置为其对应BD表的首地址，此时tx_bd_base和tx_bd_ptr是相同的配置值。地址配置示意图如图12-15所示。



图 12-15 发送方向 TX_ADDR 配置示意图



说明

软件可以通过TX_ADDR[tx_bd_ptr]和软件当前指向BD地址来维护一个HDLC通道接收方向剩余的BD个数。

• CHNL_MODE配置描述

CHNL_MODE寄存器用于配置每个通道的中断控制信息，以及工作模式信息和发送报文控制。寄存器的描述如表12-12所示。

表 12-12 CHNL_MODE 寄存器描述

位	名称	描述
[31:22]	-	保留。
[21]	mrf_mask	超长帧中断屏蔽。 0：屏蔽中断； 1：使能中断。
[20]	rxf_mask	接收完一帧的中断屏蔽。 0：屏蔽中断； 1：使能中断。
[19]	rxb_mask	接收完一个BD对应数据的中断屏蔽。 0：屏蔽中断； 1：使能中断。
[18]	txb_mask	发送完一个BD对应数据的中断屏蔽。 0：屏蔽中断； 1：使能中断。



位	名称	描述
[17]	unrxb_mask	没有接收Buffer的中断屏蔽。 0：屏蔽中断； 1：使能中断。
[16]	untxb_mask	没有发送数据中断屏蔽。 0：屏蔽中断； 1：使能中断。
[15:12]	num_of_flag	帧头的FLAG个数。 配置范围n为0x0~0xF，表示n+1个FLAG。
[11]	ch_rx_stop	通道接收停止配置。 0：正常工作； 1：停止通道的接收。 配置通道停止工作后，该发送通道将当前的帧的数据接收完成之后，停止数据的接收。
[10]	ch_tx_stop	通道发送停止配置。 0：正常工作； 1：停止通道的发送。 配置通道停止工作后，该发送通道将当前的帧的数据发送完成之后，停止数据的发送。
[9:8]	crc_mode	CRC模式配置。 00：无CRC校验； 01：16bitCCITT-CRC校验； 10：32bitCCITT-CRC校验； 11：保留。 该参数为通道参数，同时配置了通道的发送和接收方向的CRC模式。
[7:4]	pad	帧间FLAG/IDLE的个数。 0x0~0x1：1个FLAG/IDLE； 0x2~0xF：n个FLAG/IDLE。 该参数修改后，需要进行回读，确认修改是否成功。
[3]	-	保留。
[2]	pol	发送BD查询使能。 0：禁止； 1：使能。 将该位置1，发送逻辑会到外存获取BD，查询其状态，并启动传输。当发送逻辑从外部获取BD后，将该位清零。 软件每次填完一帧或多帧的BD和数据后，需要将该位置1，保证发送逻辑将内存中的数据发送出去。



位	名称	描述
[1]	idle_mode	通道发送IDLE选择。 0：禁止IDLE发送； 1：使能IDLE发送。 在IDLE使能的模式下，帧间发送的IDLE（0xFF）个数有发送BD中的pad决定。 当idle_mode禁止时，num_of_flag为两帧之间的FLAG个数；pad为帧结束之后的FLAG个数（包括帧结束的FLAG）；此时要求pad<num_of_flag。 当idle_mode使能时，num_of_flag为帧头的FLAG个数；pad为帧尾FLAG之后的IDLE个数。此时num_of_flag、pad无关。 该参数修改后，需要进行回读，确认修改是否成功。
[0]	-	保留。

CHNL_MODE[ch_rx_stop]和CHNL_MODE[ch_tx_stop]的配置可以实现禁止当前HDLC通道的数据传输。详细的使能和禁止通道的操作流程请参考[通道重新启动](#)章节的描述。

须知

当配置了CHNL_MODE[ch_rx_stop]或者CHNL_MODE[ch_tx_stop]参数后，当前的HDLC通道逻辑会在接收完当前的HDLC帧后，才会停止当前数据的接收或者发送。

CHNL_MODE[crc_mode]用于配置当前HDLC通道的CRC模式，该参数对通道的发送方向和接收方向都有效。

须知

PMC寄存器中共享了CRC的结果比较值，所以所有的HDLC通道只能同时都配置为16bitCCITT-CRC校验或者32bitCCITT-CRC校验。

CHNL_MODE[num_of_flag]与CHNL_MODE[pad]、CHNL_MODE[idle_mode]的相互关系如[图12-16](#)和[图12-17](#)所示。

图 12-16 帧间 7E 配置（idle_mode=0，pad=2，num_of_flag=4）

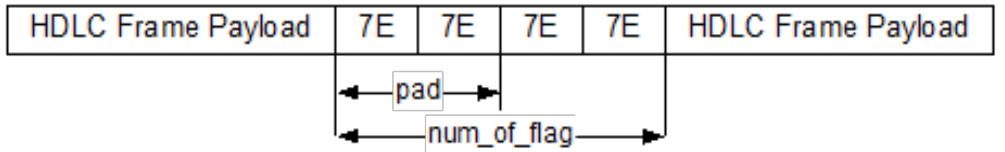
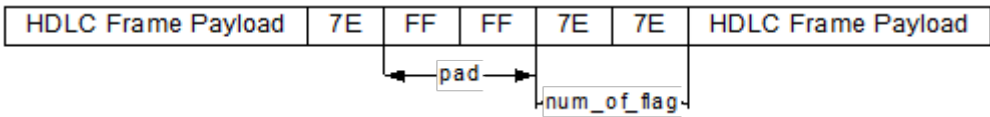




图 12-17 帧间 7E 配置（idle_mode=1，pad=2，num_of_flag=2）



须知

在CHNL_MODE[idle_mode]为0时，CHNL_MODE[pad]的配置值一定要小于CHNL_MODE[num_of_flag]的配置。

CHNL_MODE[pol]用于软件控制PMC发送报文。软件在配置完一帧数据后，对该位置1，通知HDLC通道开始发送报文。详细的配置流程请参考[报文接收](#)章节的描述。

- TXSTATE配置描述
TXSTATE描述了HDLC通道发送状态机的状态信息，其描述如[表12-13](#)所示。

表 12-13 TXSTATE 寄存器描述

位	名称	描述
[31:8]	tmp	内部缓存变量。
[7:4]	tx_state	发送状态机状态。
[3]	tx_stop	发送停止状态指示。 当发生异常时该位为1。上电时需要配置为0。
[2]	tx_bd_rdy	通道发送BD准备完成。 0：没有准备好的BD； 1：BD配置完成。 在应用中，在帧内检测到BD没有准备好，则发送Abort。然后处于停止数据发送的状态，相应间隙发送IDLE。此时该位自动置为0。
[1]	tx_init_done	发送通道初始化完成指示。 0：未完成； 1：完成初始化。 上电配置时，需要将该位配置为0。 注意：必须配置有效的发送BD和发送数据，发送初始化才算完成。
[0]	ch_tx_en	通道发送使能。 0：禁止； 1：使能。



对于TXSTATE的操作请参考[工作方式](#)章节的描述。在各种相关模式下，TXSTATE的配置方式和配置值都进行了描述。

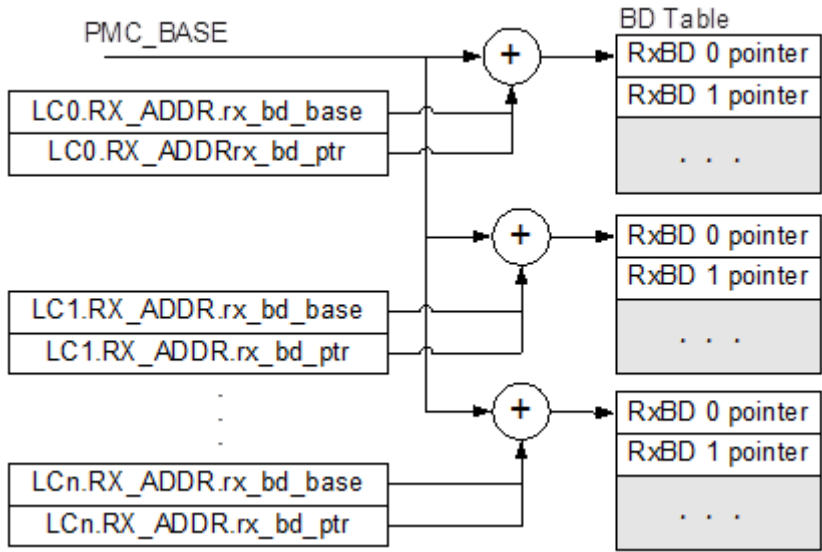
- RX_ADDR配置描述
RX_ADDR用于配置每个HDLC通道的接收方向BD表的基地址。寄存器描述如[表12-14](#)所示。

表 12-14 RX_ADDR 寄存器描述

位	名称	描述
[31:16]	rx_bd_base	接收方向BD表的基地址，单位字节。需要按照DWORD（8Byte）对齐。 PMC_BASE+tx_bd_base为外存物理地址。
[15:0]	rx_bd_ptr	接收方向，当前使用的BD地址，单位Byte。 PMC_BASE+tx_bd_ptr为外存物理地址。

系统上电或者软复位之后，初始化PMC时，需要将每个通道的RX_ADDR地址配置为其对应BD表的首地址，此时rx_bd_base和rx_bd_ptr需要配置为相同值。地址配置示意图如[图12-18](#)所示。

图 12-18 接收方向 RX_ADDR 配置示意图



说明

软件可以通过RX_ADDR[rx_bd_ptr]和软件当前指向BD地址来维护一个HDLC通道接收方向剩余的BD个数。

- RX_MODE配置描述
RX_MODE寄存器用于配置接收方向的属性，寄存器的描述如[表12-15](#)所示。PMC还提供了一个全局的超长帧配置寄存器PMC_RFRMCFG。如果配置了PMC_RFRMCFG，则每个通道的RX_MODE[max_rx_frm_len]将无效。



表 12-15 RX_MODE 寄存器描述

位	名称	描述
[31:16]	rx_datacnt	内部缓存变量，计数帧中接收到的数据个数。
[15:0]	max_rx_frm_len	最大接收帧长配置寄存器，单位Byte。 注意： 1. 这里配置时需要包含CRC的长度的。 2. 如配置最长帧的长度为3000字节，使用CRC16的模式，则该寄存器配置为3004。 3. 该值的配置需要小于等于0xfff0。 4. 当传输帧长大于等于该值时，将被判断为超长帧。

寄存器PMC_RFRMCFG的定义如表12-16所示。

表 12-16 PMC_RFRMCFG 寄存器描述

位	名称	描述
[31:16]	max_rx_frm_len	最大接收帧长；单位Byte。 0x0008 ~ 0xFFF0：n个Byte； 0x0：使用RX_MODE[max_rx_frm_len]配置。使用RX_MODE[max_rx_frm_len]配置。 其他：保留。
[15:0]	min_rx_frm_len	最小接收帧长；单位Byte。 0x0000 ~ 0xFFF0：n个Byte； 注意： 1. 该配置值需要包含CRC的长度。 2. 如果需要配置最短帧的长度为2Byte，使用CRC16的模式，则该寄存器配置为4。 3. 当传输帧长小于等于该值时，将被判定为超短帧。



须知

RX_MODE[max_rx_frm_len]的配置约束：

- 该配置需要包含CRC的长度。
如配置最长帧的长度为3000字节，使用CRC16的模式，则该寄存器配置为3002。
- 该配置值需要小于等于0xFFF0。
- 当接收帧的帧长大于等于该值时，将被逻辑判断为超长帧。

PMC处理超长帧时采用了8字节对齐的方法。在[max_rx_frm_len]的配置不为8*n（n为正整数）时，PMC在接收数据长度达到等于[max_rx_frm_len]后继续接收帧数据，直到构成一个8字节的数据段。在超过[max_rx_frm_len]的数据接收过程中，PMC仍然会检查接收帧的各种状态信息。

- RXSTATE配置描述
RXSTATE寄存器描述HDLC通道接收方向的状态信息，其定义如表12-17所示。

表 12-17 RXSTATE 寄存器描述

位	名称	描述
[31:12]	tmp_st0	接收方向内部临时状态缓存。
[11:8]	rx_state	接收状态机。
[7:3]	tmp_st1	接收方向内部临时状态缓存。
[2]	stop	通道停止状态指示。 0：正常传输； 1：通道停止接收数据。 在下面情况下，通道会停止传输。 1. 接收的BDFIFO空溢出； 2. 接收的DataFIFO满溢出。 3. BD写FIFO满溢出。 4. 没有有效BD，即内存中该通道的所有BD都已存满数据，需要软件进行数据的读取和更新BD的操作。
[1]	init_done	通道初始化完成指示。 0：未初始化； 1：初始化完成。
[0]	ch_rx_en	读通道使能。 0：禁止； 1：使能。

对于RXSTATE的操作请参考[工作方式](#)章节的描述。在各种相关模式下，TXSTATE的配置方式和配置值都进行了描述。



12.4.6.5 帧统计信息

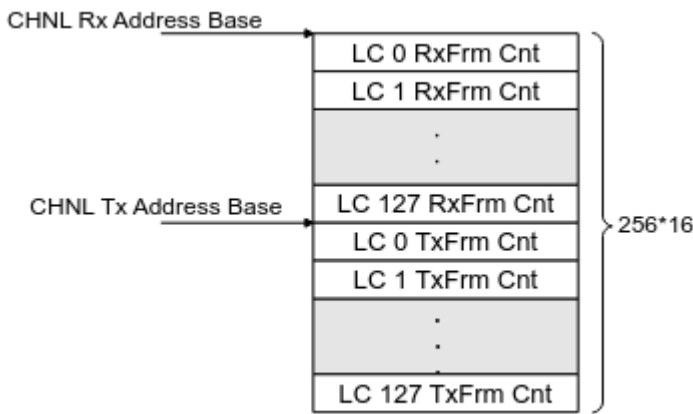
PMC实现了对每个HDLC通道发送正确帧和接收正确帧统计的功能。每个通道分配了16bit的发送帧计数器和16bits的接收帧计数器。

须知

在计数器加溢出的情况下，计数器将溢出到零，并继续累加。以此保存通道接收帧数和发送帧数的细节信息。

图12-19说明了PMC帧统计计数器在内部RAM中的存储结构。软件可以通过帧统计计数器和对接设备的帧统计计数器进行比较来判断在设备通信过程中是否存在帧丢弃或者错误发生。

图 12-19 PMC 帧统计计数器存储结构



说明

LC: 表示HDLCLogicChannel。

帧统计计数器的定义如表12-18所示。

表 12-18 PMC 接收帧计数器描述

位	名称	描述
[15:0]	chnl_rxf_cnt	接收通道正确帧统计。通过软件写零清零。



说明

接收正确帧是指当前接收帧没有发生如下错误。

- CRC错误。
- ABORT错误。
- 超长帧错误。
- 超短帧错误。
- 字节非对齐错误。

表 12-19 PMC 发送帧计数器描述

位	名称	描述
[15:0]	chnl_txf_cnt	发送通道正确帧统计。通过软件写零清零。

说明

发送正确帧是指当前发送帧没有发生由于BD没有准备好或者数据准备好而发送的ABORT帧。

须知

读写PMC的帧统计计数器时，需要按照16bits的方式进行操作，否则不能正确的读写该计数器。

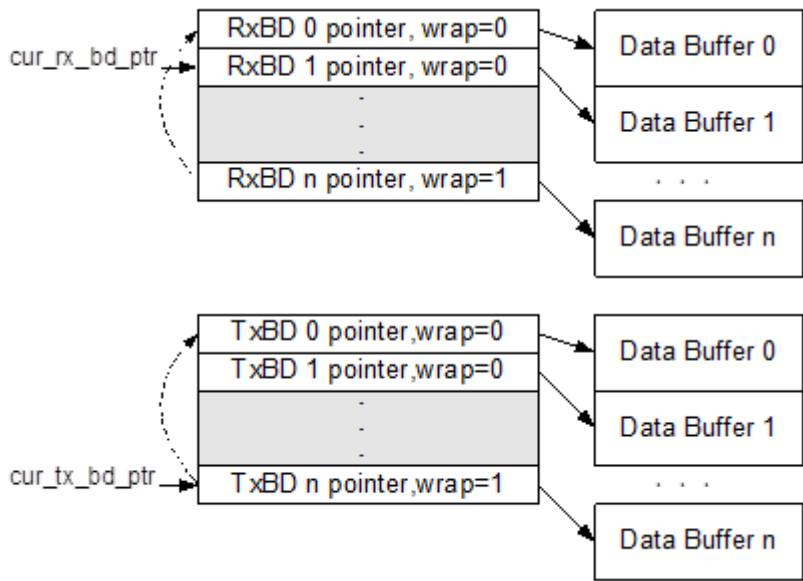
12.4.7 缓存描述符

12.4.7.1 组织框图

缓存描述符（BufferDescriptor）用于组织HDLC数据在内存中的存储，其示意图如图12-20所示。



图 12-20 BD 应用结构示意图



每个HDLC通道都会维护一个接收BD表和发送BD表，通过BD指向接收数据缓存和发送数据缓存，从而完成对HDLC报文的存储。软件通过BD表就可以读取接收到的帧数据或者配置需要发送的帧数据到DataBuffer中。

须知

- 1. 每个通道的接收BD表和发送BD表都必须是一段连续的存储空间，用于连续存放BD。
- 2. PMC的BD表最大空间为64K字节。

12.4.7.2 发送 BD

发送BD定义了当前BD对应的数据缓存首地址，数据长度以及当前发送帧信息。发送BD的描述如表12-20所示。

表 12-20 TxBD 描述

位	名称	描述
[63:32]	txbuf_ptr	发送数据缓存的首地址，单位字节。 注意：DataBuffer的首地址需要按照8字节对齐。
[31:16]	txdata_len	发送数据缓存中需要发送的数据长度，单位字节。 0x0 ~ 0x1：1个数据； 0x2 ~ 0xFFFF：n个数据。
[15:7]	-	保留。



位	名称	描述
[6]	-	保留。
[5]	tx_crc	发送数据添加CRC码使能。 0：禁止帧尾发送CRC； 1：使能帧尾发送CRC。 该值仅在TxBD[lb]为1时，才有效。用户若在HDLC帧尾增加CRC码，则需要同时将TxBD[tx_crc]和TxBD[lb]配置为1。
[4]	lb	一帧的最后一个BD。 0：不是最后一个BD； 1：最后一个BD。 PMC在检查到该信息后，发送完该BD的数据后发送CRC码和帧结束标志（0x7E）。
[3]	-	保留。
[2]	int_en	单个BD的数据发送完成中断使能。 0：禁止； 1：使能。 在该位使能时，每个BD的数据发送完成之后，都会产生BD发送完成的中断； 在该位禁止时，仅在一帧的最后一个BD的数据和CRC都发送完成之后才产生一帧发送完成的中断。
[1]	wrap	通道发送BD表环回边界指示。 0：非边界发送BD； 1：该通道最后一个发送BD，在读取完该BD后，逻辑自动跳转到该通道发送BD表首，读取BD。
[0]	rdy	发送数据准备状态。 0：没有发送数据； 1：数据已准备好。

须知

1. 在发送BD配置中，需要关注一帧最后一个BD的配置，如果需要在帧尾添加CRC码，则需要配置TxBD[tx_crc]和TxBD[lb]同时为1。
2. 如果一帧使用多个BD，则只能在该帧最后一个BD中配置TxBD[lb]为1。
3. 建议发送方向的[int_en]不使能。

12.4.7.3 接收 BD

接收BD定义了当前BD对应的数据缓存首地址，数据长度以及当前接收帧信息。接收BD的描述如表12-21所示。在接收BD中，记录一个HDLC帧的多种状态信息，软件可以根据这些状态信息进行帧处理以及帧统计。



表 12-21 RxBD 描述

位	名称	描述
[63:32]	rxbuf_ptr	接收数据缓存的首地址，单位字节。 注意：DataBuffer的首地址需要按照8字节对齐。
[31:16]	rxdata_len	接收数据缓存中有效数据长度，单位字节。 0x0 ~ 0xFFFF: n个数据。 注意：在特定场景下，PMC会存储数据长度为0的BD，此时软件不需要读取该DataBuffer中的数据。
[15:13]	-	保留。
[12]	crc_err	接收帧CRC错误状态。 0：无CRC错误； 1：有CRC错误。
[11]	abort	接收帧Abort错误。 0：无Abort错误； 1：有Abort错误。
[10]	nonoctet	接收帧字节非对齐错误。 0：无字节非对齐错误； 1：有字节非对齐错误。
[9]	lv	接收帧为超长帧状态。 0：正常帧； 1：超长帧。
[8]	sv	接收帧为超短帧状态。 0：正常帧； 1：超短帧。
[7]	-	保留。
[6]	-	保留。在完成一个BD的数据存储之后，PMC对RxBD[empty]置0。
[5]	fb	一帧的第一个BD。 0：不是第一个BD； 1：第一个BD。 PMC在接收到帧头时，会将第一个BD的RxBD[fb]为1，软件根据RxBD[fb]和RxBD[lb]来获取一个完整的HDLC帧。



位	名称	描述
[4]	lb	一帧的最后一个BD。 0：不是最后一个BD； 1：最后一个BD。 PMC在接收到帧尾时，会将该BD的RxBD[fb]为1。
[3]	-	保留。
[2]	int_en	单个BD的数据接收完成中断使能。 0：禁止； 1：使能。 在该位使能时，每个BD的数据接收完成之后，都会产生BD接收完成的中断； 在该位禁止时，仅在一帧的最后一个BD的数据接收完成之后才产生一帧发送完成的中断。
[1]	wrap	通道BD表环回边界指示。 0：非边界发送BD。 1：该通道最后一个接收BD，在读取完该BD后，逻辑自动跳转到该通道接收BD表首读取BD。
[0]	empty	数据接收完成状态。 0：没有接收数据； 1：接收数据完成。

须知

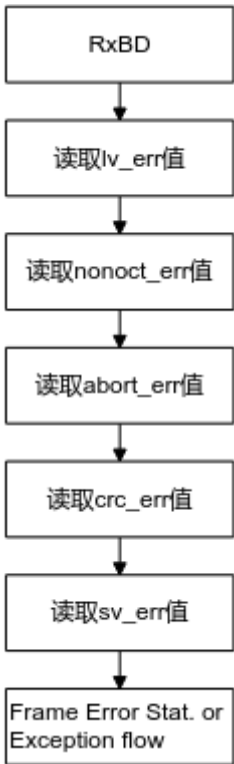
接收方向，PMC将HDLC帧的CRC数据也作为帧数据存储到外存中。软件在处理帧数据时需要注意该条件的影响。

1. 在16bitCCITT-CRC模式下，帧真实长度为RxBD[rxdata_len]-2字节。
2. 在32bitCCITT-CRC模式下，帧真实长度为RxBD[rxdata_len]-4字节。

接收帧的错误状态可以按照如[图12-21](#)所示的顺序进行处理，来达到最优的处理流程。



图 12-21 接收帧错误处理流程图



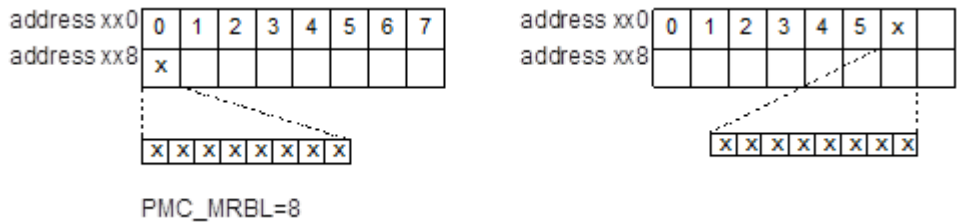
在初始化PMC时，软件需要分配DataBuffer空间。PMC通过PMC_MRBL寄存器来配置每个接收DataBuffer的大小，寄存器定义如表12-22所示。

表 12-22 PMC_MRBL 寄存器描述

位	名称	描述
[31:16]	-	保留。
[15:0]	max_rx_buf_len	接收数据缓存大小，单位字节。需要按照DWORD（8字节）对齐配置。

图12-22说明了接收方向出现数据非对齐的帧时，PMC存储数据的方式。在字节非对齐的传输场景中，如果非对齐字节刚好出现在一个接收DataBuffer的末尾，PMC将会把这个字节放在该DataBuffer的PMC_MRBL[max_rx_buf_len]+1的地址。所以软件在分配DataBuffer的空间时，需要分配PMC_MRBL[max_rx_buf_len]+8个字节空间给DataBuffer。

图 12-22 字节非对齐数据存储图



须知

1. 由于PMC对超长帧的处理机制的约束，要求PMC_MRBL[max_rx_buf_len]必须配置为8*n（n为正整数）。
2. 由于PMC对于非对齐报文的处理机制的约束，要求软件在分配接收方向DataBuffer的大小为PMC_MRBL[max_rx_buf_len]+8字节。

12.4.7.4 BD 应用

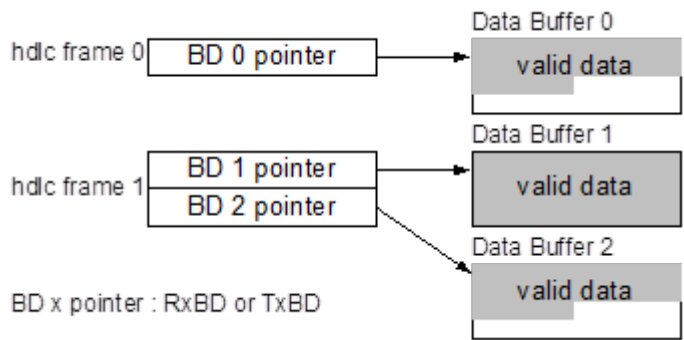
PMC采用了BD预取的实现架构，以极大的降低设备对外存访问延迟的要求。PMC的每个HDLC通道都可以缓存两个发送BD和两个接收BD。在PMC初始化时，当通道的发送或者接收使能后，逻辑会自动从外存中连续获取两个BD。在通道开始接收或者发送数据后，每完成一个BD的数据接收和发送时，逻辑自动从外存中获取下一个BD，从而达到流水处理能力。

须知

由于PMCBBD预取架构的约束，要求软件至少给每个HDLC通道分配4个接收BD和4个发送BD。同时为了限制BD越界，PMC增加约束逻辑。这要求软件给每个HDLC通道最多分配128个接收BD和128个发送BD。

在PMC的接收方向，每个HDLC逻辑通道接收到数据后，会通过BD将数据存储到DataBuffer中。每存储完一帧的数据后，无论当前DataBuffer是否完全利用，都会从下一个BD开始存储接收数据或者发送数据。在多BD多帧接收场景中，PMC对BD的处理方式如图12-23所示。

图 12-23 BD 应用示意图





12.4.8 接口时序

PMC支持多种接口时序模式，以满足多种产品场景的应用。PMC接口时序与工作模式无关，用于定义TDM总线的数据传输时序。

说明

本章节以PMC4线模式为示例，描述TDM总线接口时序模式。用户可以从时序示意图中理解PMC的时序配置效果，从而应用到其他场景下。

PMC的时序配置寄存器PMC_CONFIG定义如表12-23所示。

表 12-23 PMC_CONFIG 寄存器描述

位	名称	描述
[31:25]	-	保留。
[24]	fsp_pol	FSP信号极性。 0：高电平有效； 1：低电平有效。
[23]	byte_order	HDLC接收数据拼接和发送数据拆分方式选择。 0：接收时，先进入字节放在32bits数据最低字节； 发送时，先取32bit数据的最低字节进行发送； 1：接收时，先进入字节放在32bits数据最高字节； 发送时，先取32bist数据的最高字节进行发送。
[22]	crc_bit_invt	CRC数据按位取反使能。 0：禁止； 1：使能。
[21]	crc_inv	CRC字节bit顺序反转模式使能。 0：禁止； 1：保留。 在PMC中，仅使用按照bit流顺序的CRC计算方式。所以该值必须配置为0。
[20]	-	保留。
[19]	tdm_tx_initval	TDM_TX默认值配置。 0：传输0； 1：传输1。 在没有使能TDM时，TDM_TX的值由该寄存器确定。



位	名称	描述
[18]	echo_mode	TDM外环回模式。 0：正常模式； 1：外环回模式。 在该模式下，将TDM_RX数据线直接环回到TDM_TX数据线。同时外环回模式要求PMC工作同相共享模式。
[17]	tdm_mode	PMC的工作模式选择。 0：TDM模式； 1：保留。
[16]	tdm_iocfg	TDM4线模式和7线模式选择。 0：4线模式； 1：7线模式。
[15:12]	ts_num	TDM每帧的时隙数。 0x3 ~ 0x8：表示2**n个时隙。 其他：保留。 例如：0x7表示2**7=128个时隙。 注意：该设计中，每个时隙固定为8bits数据，所以TDM总线频率不同时，需要配置ts_num为相应的时隙数。 当超出配置范围时，寄存器使用默认值0x7。
[11:10]	-	保留。
[9:8]	tx_delay	发送方向帧同步信号到发送数据bit[0]的延迟。 00：0延迟； 01：1bit延迟； 10：2bit延迟； 11：3bit延迟。
[7:6]	-	保留。
[5:4]	rx_delay	接收方向帧同步信号到接收数据bit[0]的延迟。 00：0延迟； 01：1bit延迟； 10：2bit延迟； 11：3bit延迟。
[3:2]	-	保留。
[1]	clock_edge	数据采样时钟沿。 0：上升沿发送数据，下降沿接收数据； 1：下降沿发送数据，上升沿接收数据。



位	名称	描述
[0]	frame_edge	帧同步信号采样时钟沿。 0：时钟下降沿； 1：时钟上升沿。

须知

用户将PMC_CONFIG[rx_delay]和PMC_CONFIG[tx_delay]配置为相同的值。

PMC_CONFIG[11:0]定义了PMC接口时序的配置值。根据寄存器的定义可以将PMC接口时序分为两个大类：零延迟模式和非零延迟模式。

PMC接口时序非零延迟模式，即PMC_CONFIG[rx_delay]和PMC_CONFIG[tx_delay]都配置为非零值。PMC接口时序如图12-24和图12-25所示，该时序图对PMC_CONFIG[tx_delay]和PMC_CONFIG[rx_delay]都配置为1的场景进行描述。根据该时序图可以得到PMC接口时序中对[tx_delay]的定义，从而得到在[rx_delay]和[tx_delay]都大于1的接口时序图。

图 12-24 PMC 接口时序（clock_edge 为 0，数据延迟为 1）

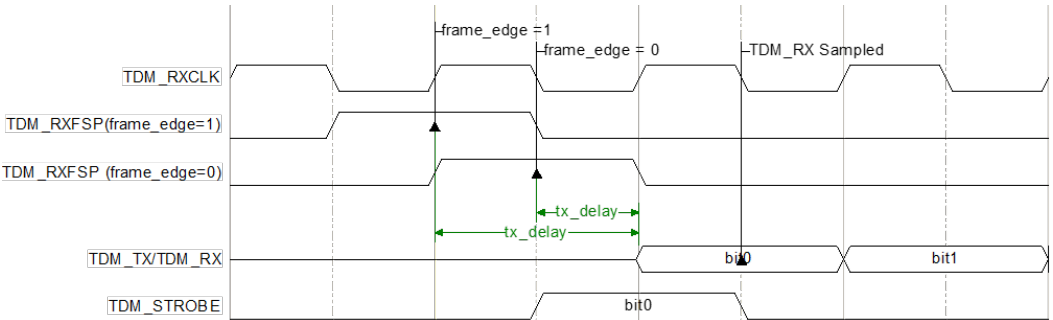
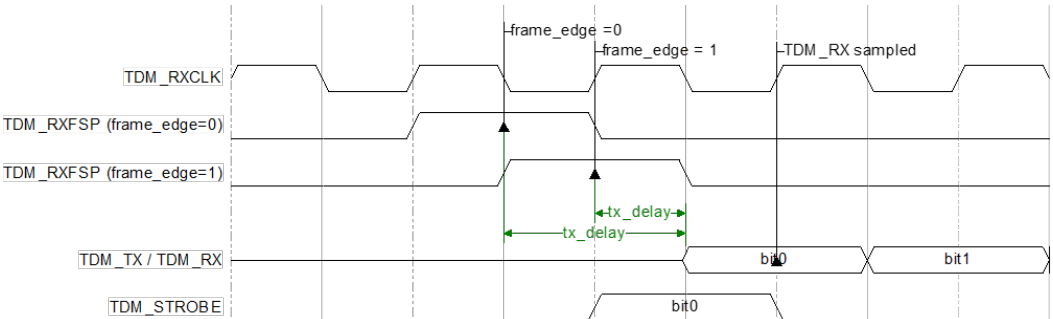


图 12-25 PMC 接口时序（clock_edge 为 1，数据延迟为 1）



PMC接口时序零延迟模式，即PMC_CONFIG[rx_delay]和PMC_CONFIG[tx_delay]都配置为0的模式。在该模式下，根据PMC_CONFIG[frame_edge]和PMC_CONFIG[clock_edge]的配置有可以得到4种接口时序类型。



图 12-26 PMC 零延迟接口时序（ frame_edge 为 0， clock_edge 为 0 ）

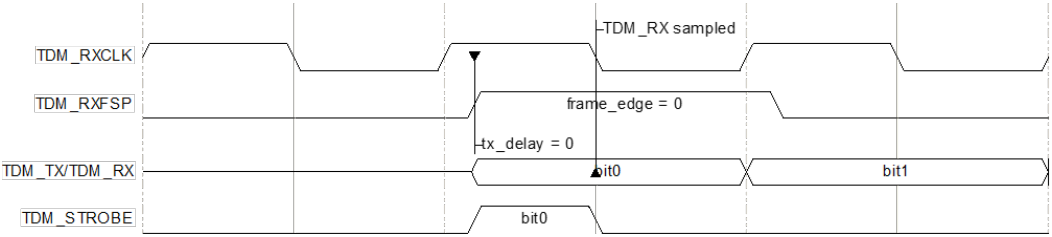


图 12-27 PMC 零延迟接口时序（ frame_edge 为 0， clock_edge 为 1 ）

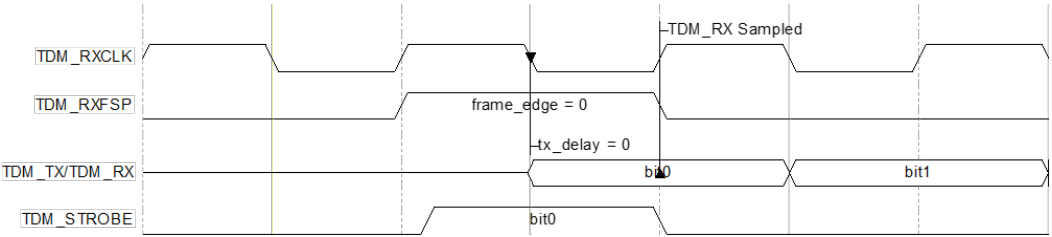


图 12-28 PMC 零延迟接口时序（ frame_edge 为 1， clock_edge 为 0 ）

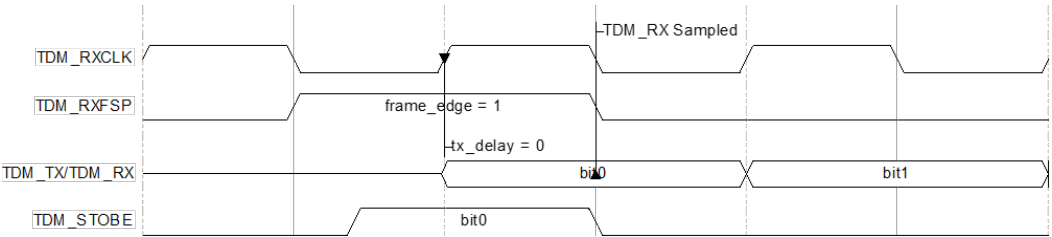
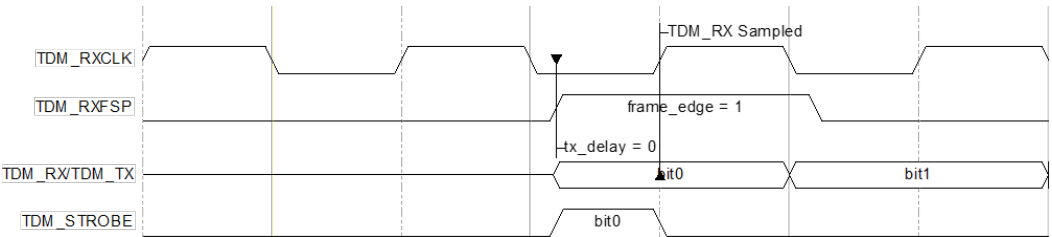


图 12-29 PMC 零延迟接口时序（ frame_edge 为 1， clock_edge 为 1 ）



12.4.9 中断处理

12.4.9.1 中断概述

PMC有两种中断类型：全局中断和通道中断。PMC根据发生事件的严重级别定义为全局中断还是通道中断。PMC在检测到不可恢复的事件发生时，通过中断寄存器上报中断，要求系统进行PMC进行即时处理。而基于通道的帧发送信息和状态则通过普通中断和软件进行交互，完成报文的接收和发送。



PMC全局中断寄存器PMC_INT的定义如表12-24所示。

表 12-24 PMC_INT 寄存器描述

位	名称	描述
[31:6]	-	保留。
[5]	pari_err	RAM偶校验错误中断。 0：没有错误； 1：偶校验错误。 电平中断方式，需要软件写1清中断。
[4]	bus_err	总线出错。 0：总线正常； 1：总线返回Error。 电平中断方式，需要软件写1清中断。
[3]	int_ovfl	中断向量表满溢出指示。 0：未溢出； 1：溢出。 电平中断方式，需要软件写1清中断。
[2]	g_int	中断向量表空状态指示。 0：为空； 1：中断向量表不为空。 电平中断方式，需要软件写1清中断。
[1]	g_tx_unfl	发送方向FIFO空溢出。 0：未溢出； 1：溢出。 电平中断方式，需要软件写1清中断。 由于内部总线繁忙，导致内部发送数据FIFO空溢出。当发生这种情况时，需要重新初始化PMC，进行数据的传输。
[0]	g_rx_ovfl	接收方向FIFO满溢出。 0：未溢出； 1：溢出。 电平中断方式，需要软件写1清中断。 由于内部总线繁忙，导致内部接收数据FIFO满溢出。当发生这种情况时，需要重新初始化PMC，进行数据的传输。

PMC用PMC_INT[pari_err]检测内部RAM软失效的状态，软件接收到该中断后，完成对PMCRAM软失效的处理。处理的方式参见[总线错误与RAM失效处理](#)章节的描述。

PMC用PMC_INT[bus_err]检测外部总线错误状态。当PMC检测到外部总线异常时上报中断，软件检测到该中断后，停止PMC的工作。详细的处理步骤参见[总线错误与RAM失效处理](#)章节的描述。



PMC_INT[g_tx_unfl]和PMC_INT[g_rx_ovfl]为PMC内部异常中断，当发生该中断时软件需要重新初始化PMC来恢复业务数据流。

PMC基于通道的中断采用了中断向量表的方式进行管理。通道中断的定义如表12-25所示。

表 12-25 PMC 通道中断（INT_ENTRY）描述

位	名称	描述
[15]	valid	中断有效状态。 0：无效； 1：有效。
[14]	ovfl	延迟溢出中断指示。 0：正常中断； 1：FIFO溢出中断指示。 注意： 该位和unrxb/untxb进行组合指示中断状态。在以下几种情况下，触发该中断。 该中断使用PMC_INT[txunfl_mask]、PMC_INT[rxovfl_mask]作为屏蔽使能。
[13]	mrf	接收到超长帧中断。
[12]	rxfr	接收完一帧中断。
[11]	rxbr	BD数据接收完成中断。 该中断受RxBD[int_en]的控制。
[10]	txbr	BD数据发送完成或帧数据发送完成中断。 该中断受TxBD[int_en]的控制。 当TxBD[int_en]为1时，该中断为BD数据发送完成中断； 当TxBD[int_en]为0时，该中断为帧数据发送完成中断。 对于BD数据发送完成的该中断，该中断受帧间CHNL_MODE[pad]配置的影响。在一帧最后一个BD时，数据发送完成之后，需要等到CHNL_MODE[pad]个FLAG发送到TxFIFO中，才会置位中断。
[9]	unrxb	接收数据缓存没准备好中断。 当[ovfl]为1时，该中断表示接收方向的FIFO溢出中断； 当[ovfl]为0时，该中断表示接收方向没有DataBuffer的中断。 当发生[ovfl]为1的中断后，PMC的该通道将进入停止数据接收状态，直到软件重新启动该通道。



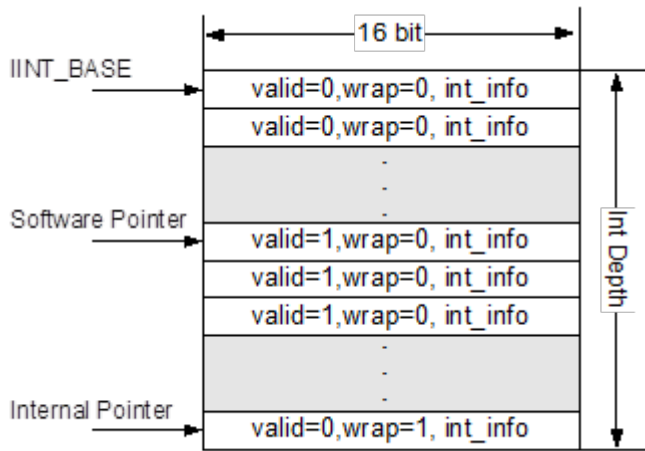
位	名称	描述
[8]	untxb	没有发送数据中断。 当[ovfl]为1时，该中断表示发送方向的FIFO溢出中断； 当[ovfl]为0时，该中断表示发送方向没有数据发送的中断。 当发送该中断后，硬件发送1个0x7F（HDLC的Abort传输），然后进入到IDLE（0xFF）发送状态，直到软件重新配置BD进行数据发送。
[7:0]	ch_num	发送中断的通道号。

须知

PMC中断向量表中，通过INT_ENTRY[ovfl]作为标志位，复用了INT_ENTRY[unrxb]和INT_ENTRY[untxb]的信息。软件在进行中断处理时需要关注该中断的具体含义。

图12-30说明了PMC中断的存储结构。中断向量表的首地址配置参见“寄存器配置”章节的描述。中断向量表的读指针由内部逻辑控制，其控制方式参见“通道中断处理”章节的描述。内部中断写指针则由通道的控制逻辑进行控制。

图 12-30 PMC 中断存储结构图



中断向量表的深度可以通过PMC_INTCTRL寄存器配置，该寄存器的描述如表12-26所示。该寄存器同时定义了全局中断的屏蔽位和中断向量深度的配置。

表 12-26 PMC_INTCTRL 寄存器描述

位	名称	描述
[31]	-	保留。



位	名称	描述
[30]	parierr_mask	偶校验错误中断屏蔽。 0：屏蔽中断； 1：使能中断。
[29]	buserr_mask	总线错误中断屏蔽。 0：屏蔽中断； 1：使能中断。
[28]	intovfl_mask	中断向量表溢出中断屏蔽。 0：屏蔽中断； 1：使能中断。
[27]	gint_mask	Gint中断屏蔽。 0：屏蔽中断； 1：使能中断。
[26]	txunfl_mask	发送方向FIFO溢出中断屏蔽。 0：屏蔽中断； 1：使能中断。
[25]	rxovfl_mask	接收方向FIFO溢出中断屏蔽。 0：屏蔽中断； 1：使能中断。
[24:16]	int_depth	中断向量深度。 0x0：保留； 0x1 ~ 0x1FF：n+2个中断深度。 注意：该值仅在PMC初始化时可配置，运行过程中，不能动态的修改该配置。
[15:0]	g_rx_frm_th	全局中断的门限。当中断向量表中有g_rx_frm_th个中断后，PMC_INT[gint]中断有效。 0x0 ~ 0xFF：n+1个中断深度； 其他：保留。 注意：该配置必须小于int_depth的配置值。

通过配置PMC_INTCTRL[int_depth]，分配中断向量表的存储空间。配置PMC_INTCTRL[g_rx_frm_th]控制PMC上报通道中断的方式。

须知

PMC_INTCTRL[int_depth]不能超出内部存储空间的边界。存储空间分配参见[存储结构](#)的描述。



12.4.9.2 异常中断处理

系统运行过程中，瞬态带宽超负荷或者大流量业务冲击系统时，会导致PMC对外存的访问延迟超出内部FIFO的容忍门限。PMC在该场景下，产生异常中断上报CPU，从而保证业务被正常处理。

全局中断的使能配置参考[表12-24](#)。

- PMC_INT[g_tx_unfl]中断产生和处理机制。

PMC在出现以下场景时，会上报g_tx_unfl中断。

- 发送方向读数据FIFO空溢出。
- 发送方向读BDFIFO空溢出。
- 发送方向写BDFIFO满溢出。
- 发送方向状态机处理超时。

对于FIFO溢出的错误类型，PMC也提供基于通道的中断。通过INT_ENTRY[ovfl]来指示该通道是否出现了FIFO溢出的中断。在通道中断上报了PMC异常的场景中，需要按照通道中断异常中断的处理方式完成第一步的处理，然后通过定时器中断进行第二步的中断处理。这种中断的处理方式可以在系统瞬态带宽不够，导致PMC异常的场景中，提高PMC的鲁棒性。即仅对某一个出现问题的通道进行异常处理，就可以恢复PMC到正常的工作状态。

当整个系统长期处于带宽超负荷的场景时，则需要进行第二步的中断处理，复位PMC并重新初始化。软件的处理流程可以参考“[中断处理流程图](#)”章节的描述。

对于发送方向状态机处理超时的错误，以及内部RAM软失效的错误，都是PMC的致命错误中断，要求软件立即停止PMC的操作，复位PMC并重新初始化，启动帧传输。

发送方向状态机处理超时的错误，一般是由于TDM总线的工作频率超出了PMC支持的范围才会导致。所以这样的场景发生的时候，需要先分析TDM总线的工作场景是否在PMC支持的范围。

异常场景的软件处理流程可以参考“[中断处理流程图](#)”章节的描述。

- PMC_INT[g_rx_ovfl]中断产生和处理机制。

PMC在出现以下场景时，会上报g_rx_ovfl中断。

- 接收方向写数据FIFO满溢出。
- 接收方向读BDFIFO空溢出。
- 接收方向写BDFIFO满溢出。
- 接收方向状态机处理超时。

对于FIFO溢出的中断处理方式和发送方向的[g_tx_unfl]一致，需要先处理通道的异常中断，在根据实际情况进行全局的中断处理。

对于接收方向状态机处理超时的错误，要求软件立即停止PMC的操作，复位PMC并重新初始化，启动帧传输。

- PMC_INT[int_ovfl]中断产生和处理机制。

软件不能及时处理PMC的通道中断，从而导致PMC的中断向量表溢出。这种场景说明了软件的处理性能不够。此时软件需要重新分析PMC的应用方式。

在PMC发生中断溢出的场景下，新产生的通道中断将被丢弃，中断向量表中维持未被处理的中断。在流量突发的场景下，出现了PMC通道中断溢出的情况，可以不对该中断进行任何处理，通过后续相同通道的中断，对完成对该通道的所有帧的处理。这种处理方式存在丢帧的风险。



12.4.9.3 通道中断处理

PMC通道中断实现硬件和软件之间的通信，完成HDLC报文的传输。通道中断的定义参见表12-25，通道中断的使能参见表12-12。

PMC定义了中断读取接口寄存器PMC_INTRD，软件通过读取该寄存器获取PMC内部的中断。PMC_INTRD定义如表12-27所示。

表 12-27 PMC_INTRD 寄存器描述

位	名称	描述
[31:16]	-	保留。
[15:0]	int_val	中断值。 通过该寄存器将PMC内部中断向量表的中断顺序读出。

软件在接收到PMC_INT[gint]中断后，通过连续读PMC_INTRD寄存器获得PMC内部的中断向量，直到读出中断向量的INT_ENTRY[valid]为0时，则停止读中断操作。在处理完所有的通道中断后，写PMC_INT[gint]位，清除该中断。

软件每次通过PMC_INTRD读取内部中断后，PMC逻辑会自动将该中断的INT_ENTRY[valid]清零，将中断置为无效。

通道中断的处理方式请参见“[中断处理流程图](#)”章节的描述。

12.4.9.4 中断处理流程图

图12-31描述了异常中断的处理流程。

图 12-31 异常中断处理流程图

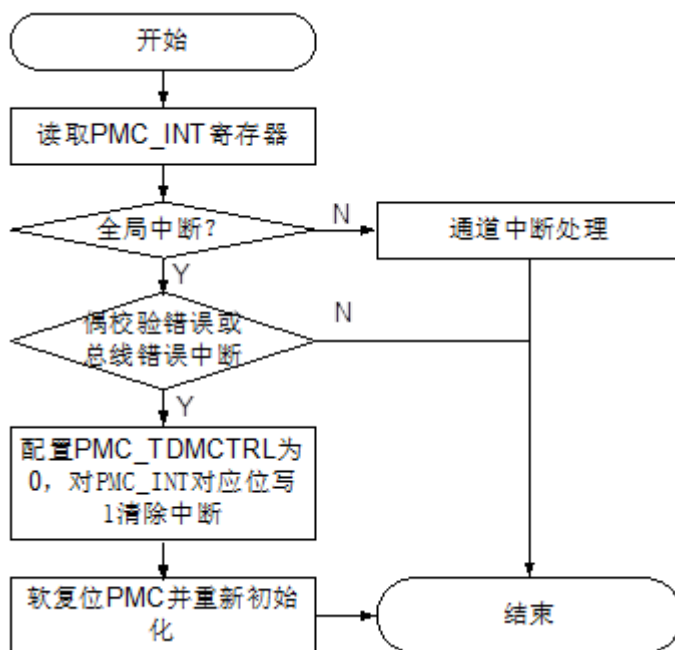
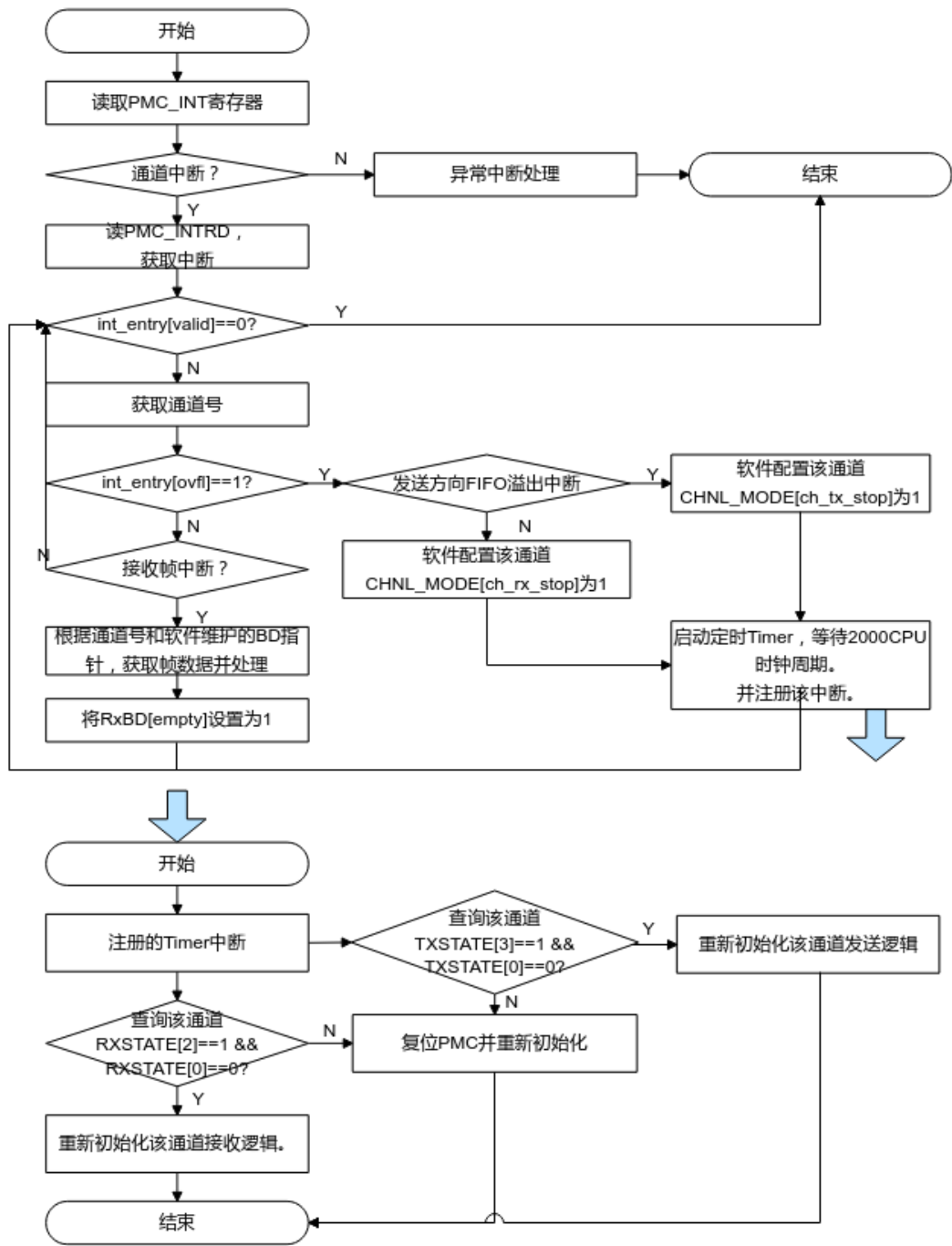


图12-32描述了通道中断的处理流程。

图 12-32 通道中断处理流程图



12.4.10 大小端工作模式

说明

本系统中的大端模式都是指BE-8模式。



12.4.10.1 DDR 访问的大小端控制

配置寄存器PMC_CONFIG[pmc_endian]，0是大端模式（默认值），1是小端模式。

对于32位数据（即放在CPU寄存器中的数据）0x11223344，如果按照小端方式写入内存，则按照8位、16位或32位写入内存，其在内存中的存放方式均如表12-28所示，如果从0x00地址读取32位，则为0x11223344，如果是从0x00地址读取16位，则为0x3344。

表 12-28 32 位数据小端存储

32位数据小端存储		
地址	值	字节标记
Byte_Addr3	0x11	MSB
Byte_Addr2	0x22	MSB -1
Byte_Addr1	0x33	LSB + 1
Byte_Addr0	0x44	LSB

对于32位数据0x11223344，如果按照大端方式写入内存，则按照8位、16位或32位写入内存，其在内存中的存放方式如下表12-29所示，如果从0x00地址读取32位，则为0x11223344，如果是从0x00地址读取16位，则为0x1122。

表 12-29 32 位数据大端存储

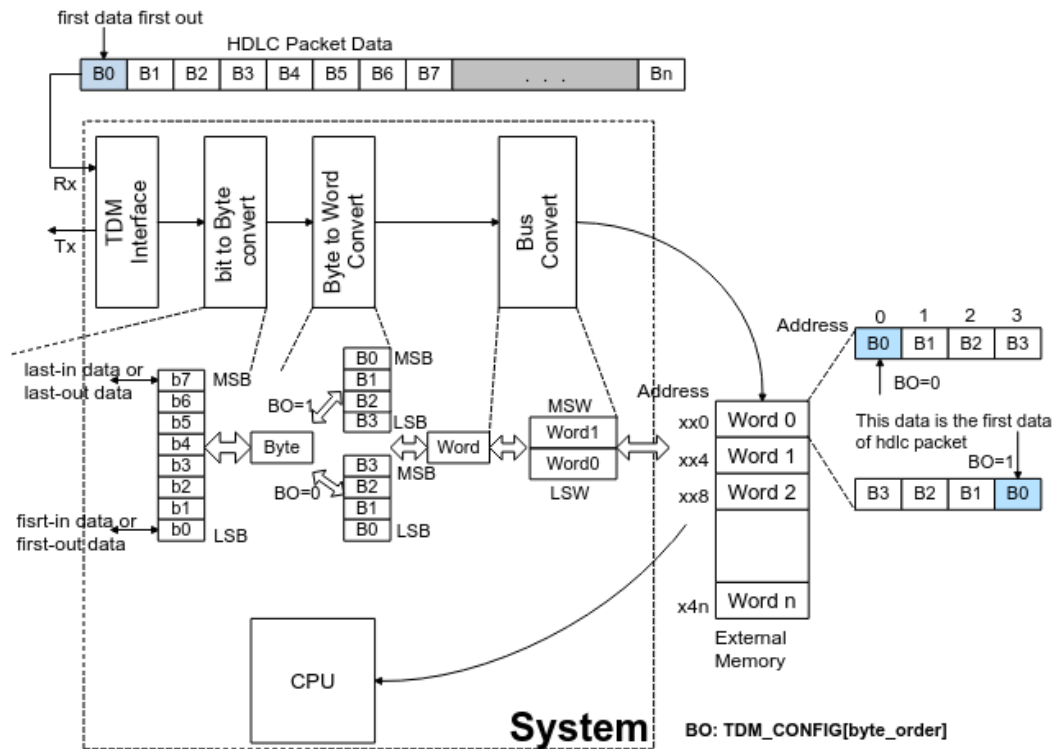
32位数据大端存储		
地址	值	字节标记
Byte_Addr3	0x44	LSB
Byte_Addr2	0x33	LSB +1
Byte_Addr1	0x22	MSB -1
Byte_Addr0	0x11	MSB

12.4.10.2 数据流框图

TDM可以用在大端模式的系统中，也可以应用在小端模式系统中。

图12-33说明了TDM在整个数据处理和存储过程中的处理方式。

图 12-33 TDM 数据处理图



12.4.10.3 大端模式（Big-Endian）

系统运行在BE-8模式时，软件特殊处理TDM的配置，包括TDM的寄存器配置方式、BD的配置方式和报文的处理方式。

TDM的寄存器和BD始终是小端模式的配置方式。即软件在配置寄存器和BD时，需要将WORD数据按照字节反序后，在写入到相应的寄存器和BD地址。

对于报文的处理，则只需要将TDM_CONFIG[byte_order]配置为0。则存放在外存中的数据已经是BE-8大端序，程序可以直接处理。

12.4.10.4 小端模式（Little-Endian）

系统运行在BE-8模式时，软件需要特殊处理TDM接收和发送的报文。在小端模式下，TDM的寄存器和BD都可以通过软件直接配置，所以这里不需要进行特殊处理。

对于TDM接收的报文，为了提升CPU处理的性能，可以配置TDM_CONFIG[byte_order]为1。TDM会对接收或者发送的报文按照WORD对齐的方式进行数据大小端转换。在这种场景下，CPU可直接按照WORD的方式读或写外存的报文。

须知

在小端系统中，TDM配置了大小端转换功能时，软件只能按照WORD的方式对报文进行读或者写。

12.4.11 性能约束

由于PMC使用了在外存中存放报文数据和BD的方式，所以外存的访问延迟约束PMC的性能。

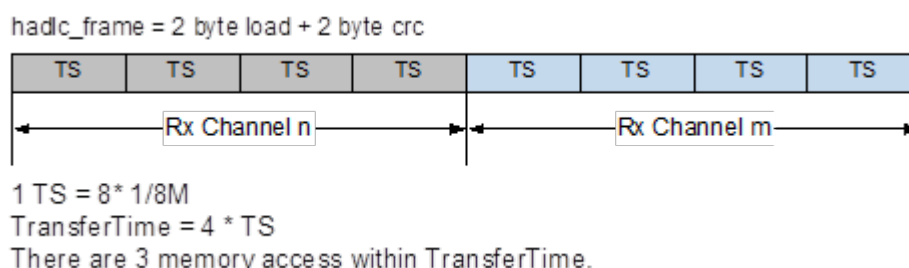
PMC接收方向帧存储访问可以分为3个阶段：预取BD阶段，一次外存读操作；写报文数据阶段，多次外存写操作；更新外存中BD状态阶段，一次外存写操作。所以一个帧数据接收至少需要3次外存访问。

在图12-34的应用场景下，连续多个时隙分配同一个通道，导致了在TransferTime的时间内必须完成3次外存操作。这对外存的访问延迟要求很高。以PMC工作在8MHz，内部总线工作在400MHz的模式为例，可以得到3次外存操作的总延迟为：

$$TotalDelay = \frac{TransferTime}{Period_{bus}} = 1600(BusCycle)$$

而平均每次操作的延迟约为500个总线时钟周期。PMC在这种场景下，抗系统突发能力会很差。

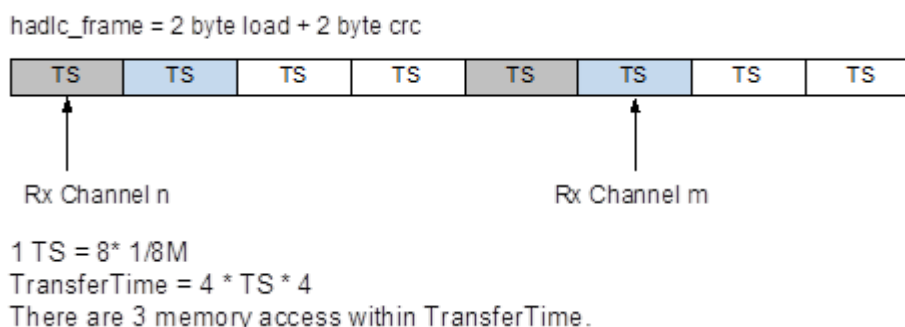
图 12-34 PMC 连续时隙性能示意图



在图12-35的应用场景下，将一帧的数据划分到不同的时隙进行数据的传输，利用了总线的空闲时隙进行数据传输。从而将TransferTime提升为场景一的4倍，而外存访问的次数仅提升了2倍，所以每次外存访问的平均延迟为1000总线时钟周期。

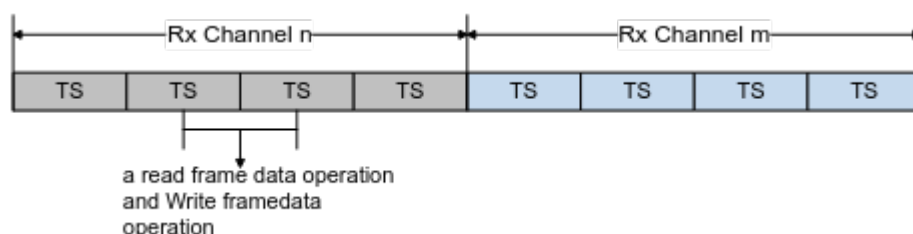
这说明通过应用方式的改变，提高了PMC抗系统突发能力。所以在应用场景中，需要重点考虑时隙的分配方式，从而利用PMC的最优性能。

图 12-35 PMC 间插时隙性能示意图



现以TDM总线工作在8.192MHz的模式为例，图12-36示意了TDM对系统延迟最敏感的场景。该场景下，最极端的延迟要求是在连续的2个时隙内完成2次存储访问。从而要求存储访问的延迟为：

图 12-36 TDM 性能约束示意图



结合上面的分析和PMC的内部总线原理，要求在1μs内完成2次总线操作，即平均的存储延迟为500ns。此时可以通过配置DDRC的Timeouts属性来保证该性能要求。

须知

在多时隙分配到1个通道时，建议采用时隙间插的方式来减少对系统延迟要求。

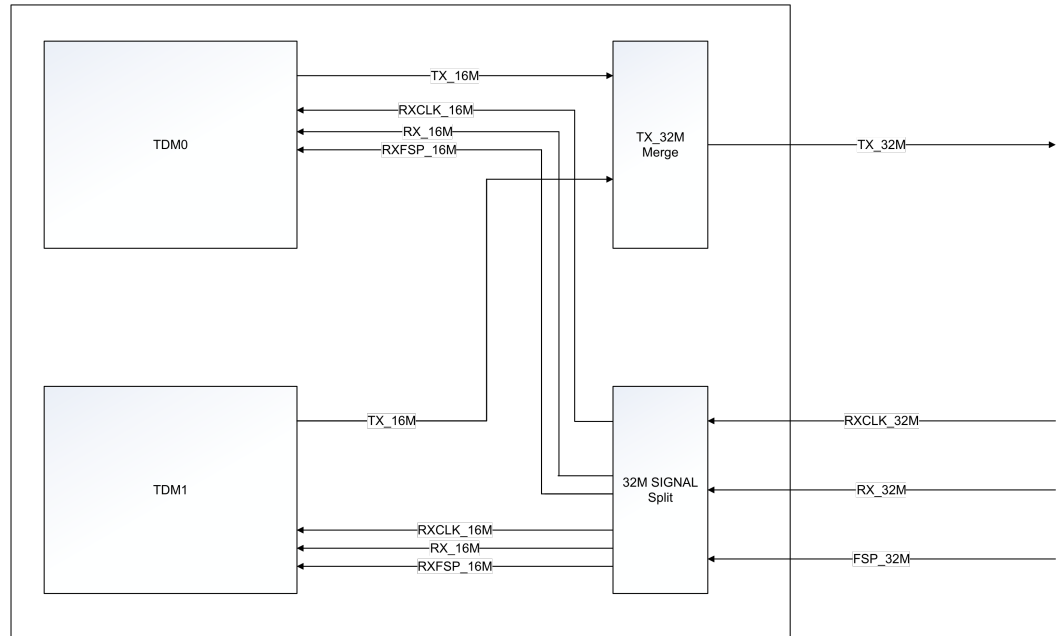
12.4.12 32M TDM 模式

Hi309a中有4个TDM（TDM0~3），通过使用两个TDM组成一对，实现在芯片接口传输32M TDM数据的特性（TDM0/1为一组，TDM2/3为一组）。

下面介绍32M功能时，主要是以TDM0/1进行介绍。TDM2/3一组的功能与TDM0/1一致。

32M TDM的工作原理框图如图12-37。内部逻辑将接口进来的32M RXCLK分成两个16M的RXCLK分别送给TDM0/1，FSP/RX信号也是经过处理成两路16M的FSP送给TDM0/1，TX则是基于TDM0/1的16 M TX组成成一路32M的TX送出。具体时序图请参见“32M TDM接口功能时序”章节。

图 12-37 TDM 32M 的工作原理



须知

要使用32M模式，需要将PERI_SUBCTRL中的寄存器SC_TDM_32M_SEL[tdm_32m_sel]配置为1。

如果使用32M模式，则TDM0/1，TDM/3/2均为32M模式。不支持32M和16M/8M模式混用。

12.4.13 32M TDM 接口功能时序

如图12-37所示，内部逻辑将接口进来的32M RXCLK分成两个16M的RXCLK分别送给TDM0/1，FSP信号也是如此，因此FSP信号相对于32M时钟分出来的16M时钟会有不同相位的情况。所以时序图有两种。

下图中带IO后缀的信号为管脚信号。其余信号为内部信号。

图12-38波形中内部的CLK_en与RxFSP在同一个上升沿为高，此时内部分发出的2个16MTDM通道的RxData（RxData0，RxData1）与RxFSP_16M的时序相同的（都晚0拍，或都晚1拍，...或都晚N拍），TxData在RxFSP后5（5-0）拍有效。



图 12-38 TDM 接口时序图 0

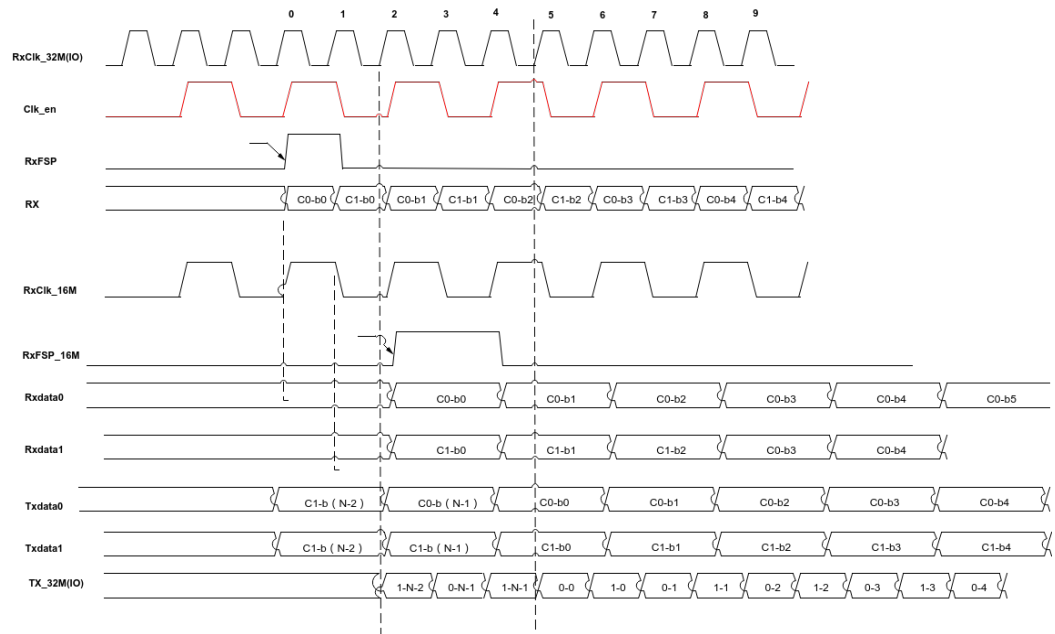
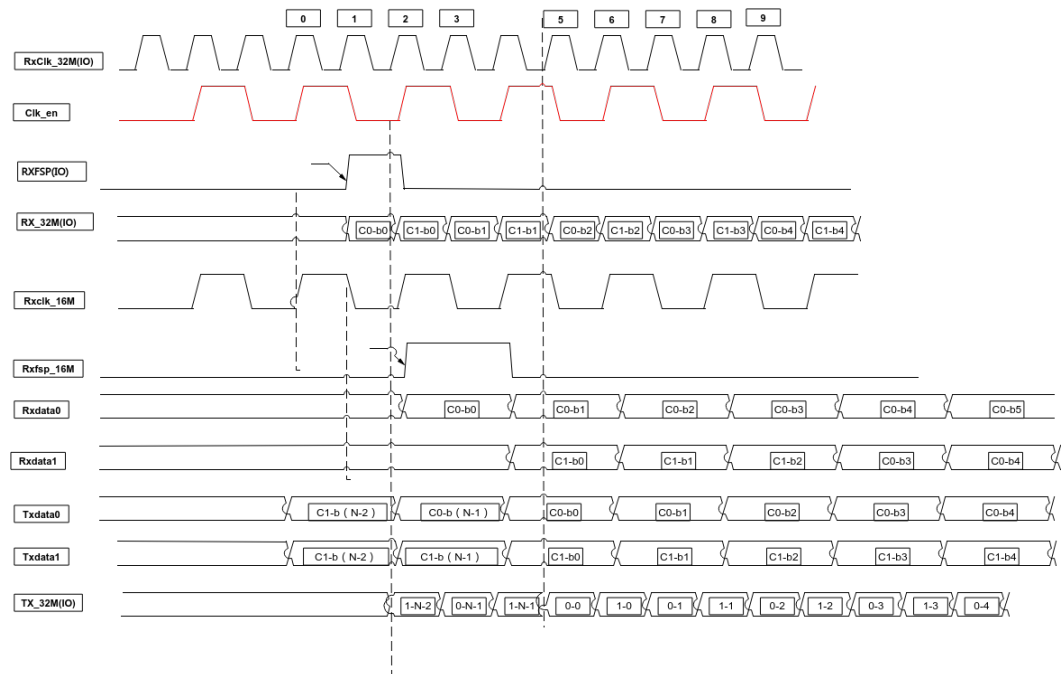


图12-39波形中内部的CLK_en与RxFSP不在同一个上升沿为高，此时内部分发出的2个16MTDM通道的RxData（RxData0，RxData1）与RxFSP_16M的时序不同，RxData1要相比RxData0晚一拍（假如RxData0晚0拍，RxData1晚1拍，...RxData0晚N拍,RxData1晚N+1拍），TxData在RxFSP后4（5-1）拍有效。

图 12-39 TDM 接口时序图 1





12.5 工作方式

12.5.1 管脚复用配置

PMC所有管脚都与GPIO复用，复用关系如表12-2所示。PMC工作之前，需要确认相应的TDM管脚都工作在PMC模式。PMC0需要配置IO——CONFIG的寄存器IOMG114~IOMG117为00，PMC1需要配置IOMG118~IOMG121为00。

须知

寄存器IOMG114~121的上电复位值为01，所以在启动PMC功能时，需要先配置IOMG113~121为00，完成TMD管脚应用为PMC的模式。

12.5.2 初始化

PMC需要经过初始化，才能进行HDLC报文的接收和发送。软件在进行PMC初始化配置时，首先需要明确PMC的应用场景，包括时隙复用方式，通道传输速率，总线工作频率等等因素。

1. 软件需要根据PMC的应用场景确认需要使用的时隙数，并根据“[存储原理](#)”章节的描述划分好PMC的存储结构。再根据“[缓存描述符](#)”章节的描述完成外存中BD的分配以及数据缓存的分配。
2. 结合PMC的应用场景，确认时隙的复用方式和HDLC通道的利用方式。
3. 硬件根据应用场景和“[板级应用](#)”章节描述，完成硬件对接逻辑设计。

上面的分析是完成PMC应用场景建立，然后就可以初始化PMC。

这以TDM总线工作在8.192MHZ，PMC工作在同相共享模式为例，对PMC的初始化配置步骤进行示例描述。

须知

该示例下，PMC工作在同相共享模式，通道采用了CCITT16bit的算法。

步骤1 读取PMC_TDMST寄存器，确认PMC_TDMST[ram_init_done]是否为1。若不为1，则循环执行**步骤1**，否则执行**步骤2**。

步骤2 根据PMC的应用场景，配置PMC_CONFIG寄存器。该示例配置为0x00487110。



说明

- [fsp_pol]: TDM_RXFSP/TDM_TXFSP极性配置，这个跟对接芯片使用的FSP信号极性相关。推荐使用高电平有效的应用模式。
- [byte_order]: PMC工作在大端模式还是小端模式的配置项。配置方式和工作原理参见“[大小端工作模式](#)”章节的描述。
- [crc_bit_invt]和[crc_inv]: 配置HDLC报文的FCS的计算方式。推荐使用默认值，完成标准的FCS算法。
- [tdm_tx_initval]: 默认状态下，TDM总线输出值。推荐使用默认值，使得在默认状态下，PMC发送总线传输数据为，即为HDLC的IDLE标志。
- [echo_mode]: PMC测试模式配置，用于外环回总线测试，详细使用方式参见章节的描述。正常功能模式下，该位配置为0。
- [tdm_mode]: PMC控制器和HDLC控制器管脚复用配置。
- [tdm_iocfg]: PMC的互联模式配置，根据应用场景进行配置，配置原理参见“[板级应用](#)”。该示例配置为0。
- [ts_num]: TDM总线工作频率配置，用于适配多种工作场景。该示例配置为7。
- [tx_delay]、[rx_delay]、[clock_edge]、[frame_edge]: PMC接口时序配置，配置原理参见“[接口时序](#)”章节的描述。推荐[tx_delay]和[rx_delay]配置为1，便于PMC和对接芯片的时序设计。该示例配置为：1,1,0,0。

步骤3 根据PMC内部存储空间分配，配置PMC的相关地址寄存器。它包括PMC_TSMCFG、PMC_TPTRCFG、PMC_RPTRCFG和PMC_INTBASE寄存器。

说明

- PMC_TSMCFG: PMCTSM表的地址配置，配置原理参见“[寄存器配置](#)”章节的描述。该示例配置为，即使用主TSM表作为当前工作表，发送和接收都分配了128个主TSM表和128个备份TSM表。
- PMC_TPTRCFG、PMC_RPTRCFG: PMCTSA表的地址配置，配置原理参见“[寄存器配置](#)”章节描述。该示例中PMC_RPTRCFG配置为，PMC_TPTRCFG配置为0x17001600。
- PMC_INTBASE: PMC中断向量表的基地址，配置原理参见“[寄存器配置](#)”章节描述。该示例中PMC_INTBASE配置为0x00001800。

步骤4 根据PMC外存中BD和Buffer的分配，配置PMC相关的寄存器。它包括PMC_BASE、PMC_MRBL和PMC_BDCFG寄存器。PMC_BDCFG寄存器必须配置为0x1ff。

说明

- PMC_BASE: PMC在外存中BD总表的基地址，配置原理参见“[缓存描述符](#)”章节的描述。BD表可以分配在非Cacheable空间，从而保证数据的一致性。该寄存器的配置完全取决于应用场景中软件对外存空间的管理和分配。
- PMC_MRBL: PMC每个接收数据Buffer大小。配置原理参见“[缓存描述符](#)”章节的描述。配置该寄存器时需要注意PMC对其约束条件。该寄存器的配置也完全取决于应用场景的要求，一般情况下，PMC_MRBL可以配置为一个HDLC报文的长度，需要考虑CRC的报文长度和PMC逻辑对数据Buffer的要求。
- PMC_BDCFG: PMC对每个通道的最大BD个数的约束。这里要求软件将该寄存器配置为0x1ff。

步骤5 结合PMC内部存储的分配，配置TSM和TSA表项。TSM和TSA的配置原理参见“[TSM配置](#)”和“[TSA配置](#)”章节的描述。

说明

- TSM和TSA表在配置时，需要从HDLC逻辑通道的角度分析，将TDM总线的时隙通过TSM表和TSA的配置，最后映射到正确的HDLC逻辑通道上。
- TSM配置时需要关注最后一个表项[wrap]的配置。



步骤6 配置PMC逻辑通道的ZISTATE寄存器和ZDSTATE寄存器。ZISTATE寄存器需要配置为0xFFFF00A0，ZDSTATE寄存器需要配置为0xff000000。

说明

ZISTATE和ZDSTATE寄存器在PMC的每个HDLC逻辑通道都有，所以这里仅需要配置需要使用的HDLC逻辑通道的参数即可。

步骤7 配置HDLC的CRC算法。包括PMC_PRECRC和PMC_RESCRC。

说明

- PMC_PRECRC：FCS初始值配置，推荐配置为。
- PMC_RESCRC：FCS比较值配置，CCITT16bit时配置为；CCITT32bit时配置为0xDEBB20E3。

步骤8 配置PMC_RFMCFG寄存器，约束HDLC的超长报文和超短报文。

说明

该寄存器需要根据PMC的应用场景进行配置。

步骤9 写PMC_INT寄存器，清零中断状态。

说明

PMC采用复位后仍然保留中断状态的方案，以方便在出现异常时，可以保留错误信息。所以在初始化PMC时，需要先将该中断寄存器写，清除默认的中断状态。

步骤10 配置PMC_INTCTRL寄存器，配置中断向量表深度和使能全局中断。该示例中配置为0x7FFF_0001。

说明

- [parierr_mask]：内部RAM软失效中断的屏蔽配置。推荐配置为，使能该中断。
- [buserr_mask]：总线错误中断的屏蔽配置。推荐配置为，使能该中断。
- [intovfl_mask]：中断向量表溢出中断的屏蔽配置。推荐配置为，使能该中断。
- [gint_mask]：通道中断的屏蔽配置。必须配置为1，使能该中断，完成通道报文的接收和发送。

----结束

[txunfl_mask]：发送方向FIFO溢出中断，配置原理参见“[接口时序](#)”章节的描述。推荐按配置为1，使能该中断。

[rxovfl_mask]：接收方向FIFO溢出中断，配置原理参见“[接口时序](#)”章节的描述。推荐配置为1，使能该中断。

[int_depth]：中断向量表深度配置，配置原理参见“[接口时序](#)”章节的描述。该示例配置为0x1FF。

[g_rx_frm_th]：通道中断触发门限，配置原理参见“[接口时序](#)”章节的描述。该示例配置为1。

步骤1 配置通道的BD表，所有需要使用的逻辑通道的接收方向和发送方向，都分配相应的BD表，以及BD对应的数据Buffer空间。BD的配置原理参见“[缓存描述符](#)”章节的描述。

步骤2 配置HDLC逻辑通道的参数。包括BD表指针，通道中断使能，以及通道使能。



说明

- TX_ADDR: 发送方向BD地址配置, 配置原理参见“[通道参数配置](#)”章节的描述。该寄存器配置和软件分配的BD空间相关。
- CHNL_MODE: 通道中断配置和通道模式配置, 配置原理参见“[通道参数配置](#)”章节的描述。软件根据实际的使用场景进行配置。[ch_rx_stop]和[ch_tx_stop]在初始化时必须配置为, [num_of_flag]可以配置为2, [pad]可以配置为1, [idle_mode]配置为0。该示例下寄存器配置为0x132110。
- TXSTATE: 发送方向状态机启动寄存器。配置该寄存器为, 即可使能该通道的发送逻辑。
- RX_ADDR: 接收方向BD地址配置, 配置原理参见“[通道参数配置](#)”章节的描述。该寄存器配置和软件分配的BD空间相关。
- RX_MODE: 接收方向通道超长帧配置。当全局寄存器PMC_RFMCFG寄存器配置后, 该寄存器可以不用配置。
- RXSTATE: 接收方向状态机启动寄存器。配置该寄存器为, 即可使能该通道的接收逻辑。

步骤3 配置PMC_TDMCTRL寄存器为1, 启动PMC进入功能模式。

说明

在配置PMC_TDMCTRL寄存器使能PMC后, PMC需要使用几个TDM帧的时间, 来完成接收通道的BD预取。当BD预取完成之后, PMC就可以接收HDLC报文了。

发送方向则需要软件每次有报文时, 按照发送报文的流程进行配置, 启动PMC的报文发送。

----结束

须知

通道的中断使能需要根据实际的应用情况来进行处理。接收BD完成中断、发送BD完成中断和超长帧中断一般用作统计功能, 可以不用开启。即CHNL_MODE[mrf_mask]、CHNL_MODE[rxb_mask]和TxB[int_en]都置为0。

12.5.3 软复位

PMC支持单独软复位的功能。在PMC应用场景中, 为了保证PMC的可靠性工作, 在特殊的应用模式或者异常处理时, 都需要使用PMC的软复位功能, 使得PMC错误可恢复。

PMC软复位步骤为:

步骤1 配置PMC_TDMCTRL为0x2。

步骤2 等待0.5ms, 保证PMC传输完成。

步骤3 配置SC_PMC_RESET_REQ各bit为1, 复位各PMC。

步骤4 等待0.5ms, 保证PMC完全复位。

步骤5 配置SC_PMC_RESET_DREQ各bit为1, 撤离各PMC复位。配置撤离软复位后, 需要等待10μs查看软复位状态寄存器SC_PMC_RESET_ST, 若为0进行下一步, 否则认为存在异常。

步骤6 初始化PMC, 在进入到正常的功能模式。

----结束



12.5.4 时钟门控

PMC支持时钟关闭功能。由于PMC的工作时钟默认为开启模式，所以在不使用PMC的场景中，可以将PMC的工作时钟关闭，来降低芯片的动态功耗。

PMC时钟关闭步骤为：

步骤1 读取系统寄存器SC_PMC_CLK_ST。

步骤2 判断PMC时钟状态，若SC_PMC_CLK_ST 各bit为1，表示PMC有工作时钟，则执行**步骤3**。

步骤3 将SC_PMC_CLK_DIS相应bit置1，关闭PMC工作时钟。

----结束

PMC时钟开启步骤为：

步骤1 读取系统寄存器SC_PMC_CLK_ST。

步骤2 判断PMC时钟状态，若SC_PMC_CLK_ST相应bit为0，表示PMC没有工作时钟，则执行**步骤3**。

步骤3 将SC_PMC_CLK_EN相应bit置1，开启PMC工作时钟。等待10μs后查看门控状态寄存器SC_PMC_ICG_ST，若响应bit为0，执行后续流程，否则认为存在异常。

----结束

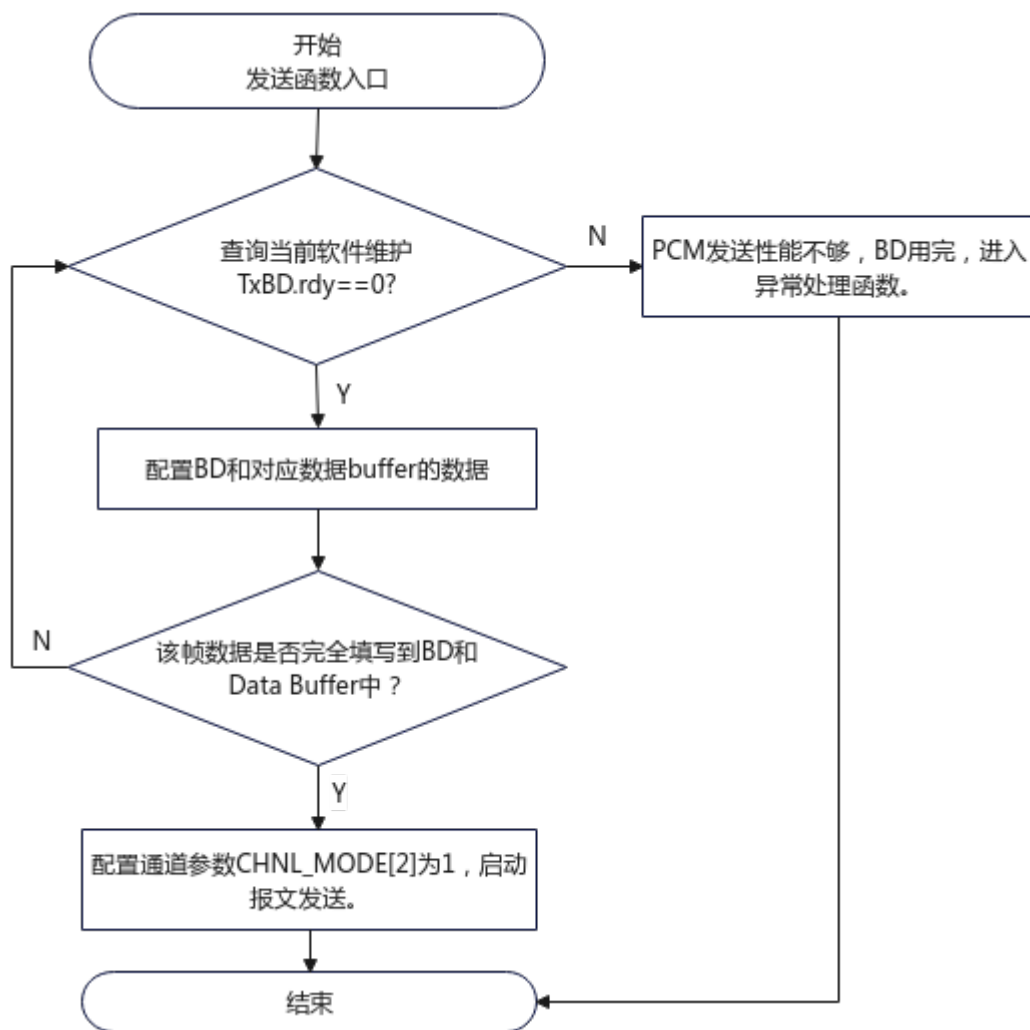
12.5.5 报文发送

PMC的报文发送流程很简单，在软件准备好报文数据和BD后，对发送通道配置报文准备好的信息，PMC自动对报文进行发送。发送流程如**图12-40**所示。

须知

在进行PMC报文发送时，要求软件在准备好一帧HDLC报文数据，才调用PMC的发送函数，进行报文发送。

图 12-40 PMC 报文发送软件配置流程



PMC报文发送的步骤为：

- 步骤1** 软件已经准备好一帧HDLC报文。
- 步骤2** 查询对应HDLC通道的TxBd[rdy]是否为0。若为0，则表示当前通道的发送性能不够，需要进入异常处理进行上层协议处理，跳转到结束。若为1，则执行**步骤3**。
- 步骤3** 将帧数据填写到该TxBd对应的数据Buffer中，然后根据帧信息配置TxBd。

说明

发送方向的写TxBd的指针由软件维护，软件负责向该BD和DataBuffer填写帧信息。硬件逻辑维护读取TxBd的指针，将该BD对应的数据发送出去。

[txdata_len]、[tx_crc]、[lb]则需要根据当前的帧长度进行配置。配置方式参见“[发送BD](#)”章节的描述。

- 步骤4** 数据是否已经完全配置到BD和DataBuffer中。若已经完成数据配置，则执行**步骤5**，否则执行**步骤2**。
- 步骤5** 配置该通道的CHNL_MODE[2]为1，启动数据发送。

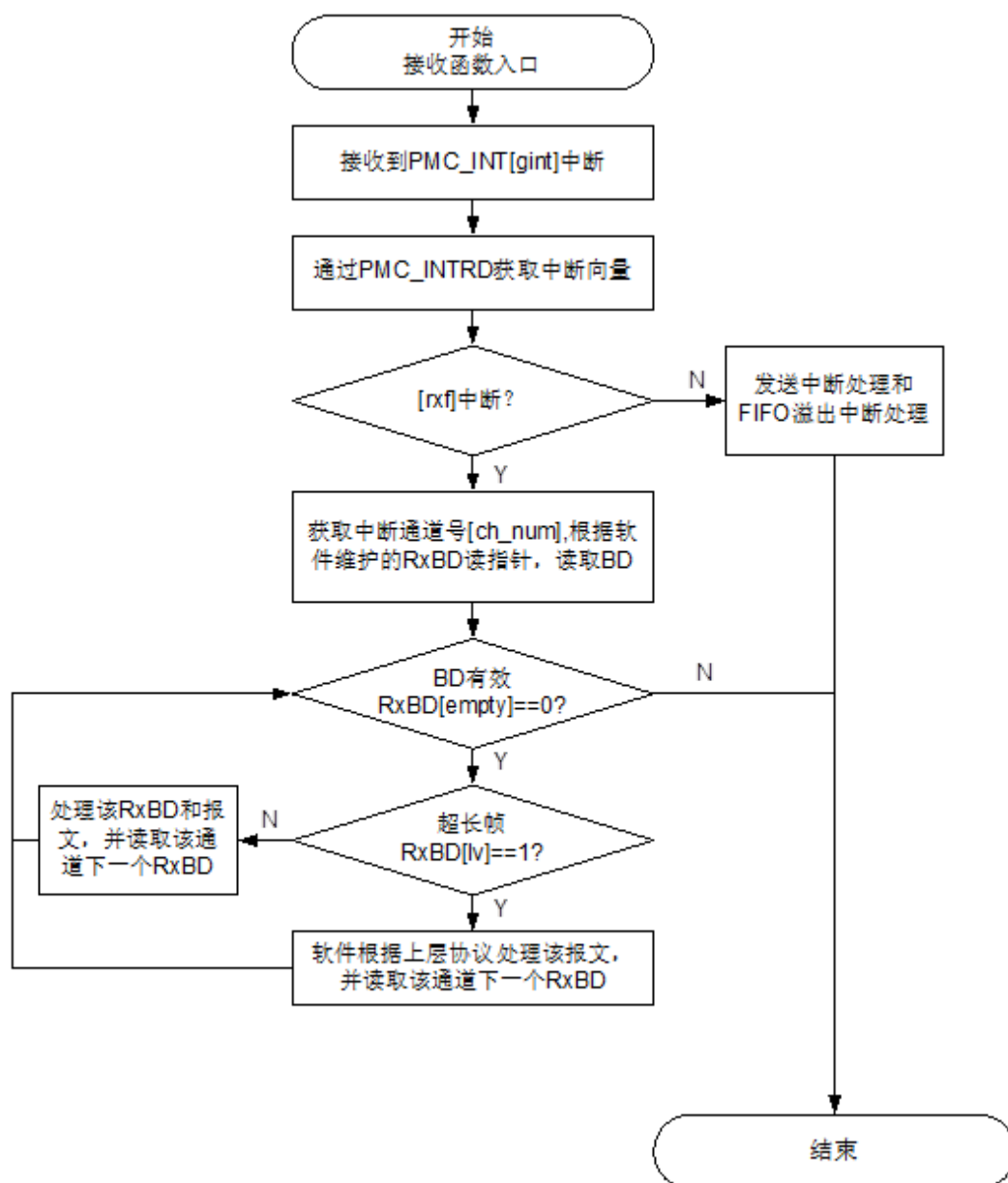
----结束



12.5.6 报文接收

PMC根据中断完成报文接收流程。PMC在接收到一帧报文后，根据中断配置上报中断，软件则根据中断对报文进行处理，以及配置RxBD，释放数据空间。PMC报文接收的流程图如图12-41所示。

图 12-41 PMC 接收报文流程图



PMC报文接收步骤如下：

- 步骤1** 软件接收到PMC_INT[gint]中断。
- 步骤2** 通过PMC_INTRD读取中断向量，若当前中断INT_ENTRY[rxf]有效，则进行**步骤3**；否则进行其他中断的处理模式，并结束。



说明

在读取到的中断向量为PMC异常帧中断时，则按照图12-32和中断处理章节描述进行处理。

步骤3 软件从INT_ENTRY[ch_num]中获取通道信息。然后根据软件维护的当前通道的BD指针读取BD和数据。然后执行步骤4。

说明

软件维护RxBD的读指针，逻辑维护RxBD的写指针。

步骤4 判断该RxBD[empty]是否为0。若为0，则执行步骤5，否则结束。

步骤5 判断该RxBD[lv]是否为1，即是否为超长帧。若为超长帧，软件根据上层协议进行处理，再执行步骤6，否则直接执行步骤6。

步骤6 处理该BD的数据后，仅将该RxBD[empty]配置为1。并读取下一个RxBD，然后执行步骤4。

说明

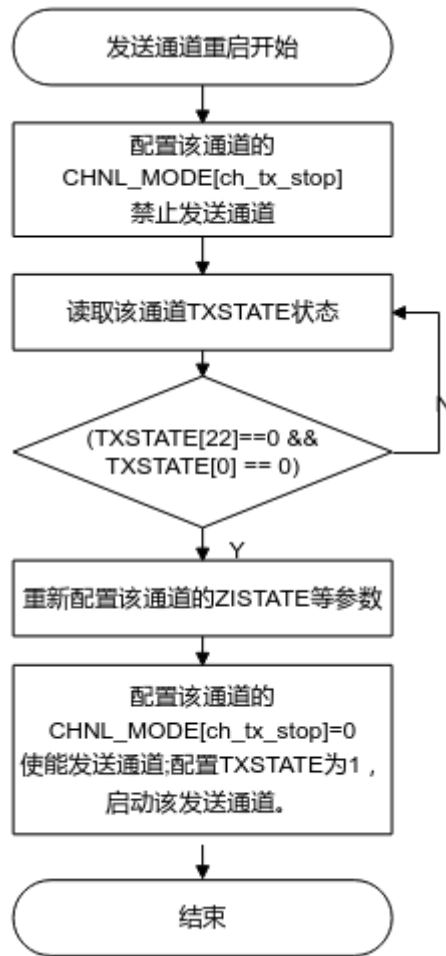
通过循环读取该通道的RxBD和数据，并置RxBD[empty]为1，完成对该接收通道的所有报文处理。

----结束

12.5.7 通道重新启动

在PMC发送工作的过程中，当出现发送方向FIFO溢出中断时，或者其他场景下，需要使用到通道重新启动的过程。图12-42说明了发送通道重启的操作流程。

图 12-42 PMC 发送通道重启流程图



发送通道重启的步骤为：

步骤1 配置该发送通道的CHNL_MODE[ch_tx_stop]为1，停止该通道的发送。

说明

如果是由于发送FIFO溢出导致进行通道重启的场景下，该操作没有影响。通道应该已经处于停止状态。

PMC仅支持基于帧结束的通道禁止功能。即在配置了通道禁止后，PMC该通道仍然会继续发送帧数据，直到该帧数据发送完成（包括正常结束和异常结束）。

步骤2 读取该通道的TXSTATE寄存器，判断表达式（TXSTATE[22]==0 && TXSTATE[0] == 0）是否为1。如果为1，则执行**步骤3**，否则执行**步骤2**。

步骤3 等待1ms的延时。

步骤4 配置该通道的ZISTATE参数。ZISTATE寄存器需要配置为0xFFFFF00A0。

说明

ZISTATE为发送方向协议处理状态机，必须配置为0xFFFFF00A0。说明：ZISTATE为发送方向协议处理状态机，必须配置为0xFFFFF00A0。

步骤5 配置该发送通道的CHNL_MODE[ch_tx_stop]为0，使能该通道的发送。



步骤6 配置该通道的TXSTATE为1，启动发送通道。

----结束

12.5.8 通道使能

PMC通道使能功能，主要应用于增加通道的场景。

PMC通道发送使能步骤：

步骤1 TDM总线是否已经通过TSM和TSA表对该HDLC通道分配了物理总线。否则需要先进行网表切换操作，给该通道分配物理时隙。

步骤2 配置发送通道的ZISTATE为0xFFFF00A0。

步骤3 配置通道的TX_ADDR和TxBD表。

说明

TX_ADDR指示了当前通道的BD基地址和当前发送BD的读指针，软件可以根据TxBD[tx_bd_ptr]得到逻辑准备操作的BD的读指针。然后根据该指针进行BD的初始化。

步骤4 配置CHNL_MODE[ch_tx_stop]为0。

步骤5 配置TXSTATE为1，使能通道发送功能。

----结束

PMC通道接收使能步骤：

步骤1 TDM总线是否已经通过TSM和TSA表对该HDLC通道分配了物理总线。否则需要先进行网表切换操作，给该通道分配物理时隙。

步骤2 配置接收通道ZDSTATE为0xFF000000。

步骤3 配置通道的RX_ADDR和RxBd表。

说明

RX_ADDR指示了当前通道的BD基地址和当前接收BD的写指针，软件可以根据RxBd[rxbd_ptr]得到逻辑准备操作的BD的写指针。然后根据该指针进行BD的初始化。

步骤4 配置CHNL_MODE[ch_rx_stop]为0。

步骤5 配置RXSTATE为1，使能通道接收功能。

----结束

PMC通道禁止功能，主要应用于通道删除的场景。

PMC通道接收禁止步骤：

步骤1 配置该通道CHNL_MODE[ch_rx_stop]为1。

步骤2 设置全局变量，指示该接收通道进入禁止模式。并等待通道中断触发。

步骤3 当该通道的接收中断发生时，表明该通道接收功能已经被禁止。配置该通道CHNL_MODE[ch_rx_stop]为0。

----结束



PMC通道发送禁止步骤：

步骤1 配置该通道CHNL_MODE[ch_tx_stop]为1。

步骤2 设置全局变量，指示该发送通道进入禁止模式。并等待通道中断触发。

当该通道的发送中断产生时，表明该通道发送功能已经被禁止。配置该通道CHNL_MODE[ch_tx_stop]为0。

----结束

12.5.9 网表切换

PMC支持TSM和TSA表动态切换功能。在这种模式下，TSM和TSA表工作在动态路由模式。PMC的内部存储分配方式参见“[TSM配置](#)”章节描述。

须知

网表切换时，如果要动态改变已有通道占用的TDM时隙，则需要PMC和对接模块都进行切换，且存在帧丢失的可能。所以网表切换一般应用于通道增加和删除的场景。

通用的切换步骤为：

步骤1 读取PMC_TDMST寄存器，PMC_TDMST[1]指示当前发送方向使用的TSM和TSA表；PMC_TDMST[0]指示了当前接收方向使用的TSM和TSA表。

步骤2 根据应用场景的需求，配置另外一份TSM和TSA表。

步骤3 根据通道使能和通道禁止的方式，使能或关闭不使用的通道。

步骤4 配置PMC_TSMCFG[24]为PMC_TDMST[1]的相反值；配置PMC_TSMCFG[8]为PMC_TDMST[0]的相反值。

----结束

PMC通道增加是一个典型的网表切换场景。可以分别增加发送通道和接收通道。

PMC动态增加发送通道的步骤：

步骤1 读取PMC_TDMST，根据PMC_TDMST[1]知道当前使用的TxTSM和TxTSA表，计算得到闲置TxTSM和TxTSA表的地址。

说明

TSM和TSA表的主网表地址和备份网表地址参见“[寄存器配置](#)”章节的描述。

步骤2 拷贝当前的TxTSM和TxTSA信息到闲置的TxTSM和TxTSA表。根据软件分配的时分复用关系，修改TxTSM和TxTSA表，以增加新通道的配置信息。

步骤3 配置新增发送通道的ZISTATE为0xFFFF00A0。

步骤4 初始化新增发送通道的BD表，以及分配相应的DataBuffer空间。

步骤5 根据该通道的TxBD表，配置TX_ADDR。

步骤6 配置TXSTATE为1，使能该发送通道。



步骤7 配置PMC_TSMCFG[24]为PMC_TDMST[1]的相反值，切换网表，增加HDLC发送通道。

----结束

PMC动态增加接收通道的步骤：

步骤1 读取PMC_TDMST，根据PMC_TDMST[0]知道当前使用的RxTSM和RxTSA表，计算得到闲置RxTSM和RxTSA表的地址。

说明

TSM和TSA表的主网表地址和备份网表地址参见“[寄存器配置](#)”章节的描述。

步骤2 拷贝当前的RxTSM和RxTSA信息到闲置的RxTSM和RxTSA表。根据软件分配的时分复用关系，修改RxTSM和RxTSA表，以增加新通道的配置信息。

步骤3 配置新增接收通道的ZDSTATE为0xFF000000。

步骤4 初始化新增接收通道的BD表，以及分配相应的DataBuffer空间。

步骤5 根据该通道的RxBd表，配置RX_ADDR。

步骤6 配置RXSTATE为1，使能该发送通道。

步骤7 配置PMC_TSMCFG[8]为PMC_TDMST[0]的相反值，切换网表，增加HDLC接收通道。

----结束

PMC通道删除需要基于通道禁止的操作来进行。

通道删除的步骤为：

步骤1 根据通道禁止章节的描述，禁止需要删除的发送或者接收通道。

步骤2 读取PMC_TDMST寄存器，获得当前使用的TSA和TSM表。

步骤3 拷贝TSA和TSM表到闲置的TSM和TSA表，并修改TSM表，将需要删除的通道的TSM表项无效（TSM_ENTRY[valid]=0）。

步骤4 配置PMC_TSMCFG[24]为PMC_TDMST[1]的相反值；或者配置PMC_TSMCFG[8]为PMC_TDMST[0]的相反值。切换网表，删除HDLC通道。

----结束

12.5.10 多时隙应用

PMC支持单个逻辑通道16kbps、32kbps、48kbps和64kbps的数据传输率。在数据传输率小于64kbps时，需要使用到子时隙模式。

子时隙的关键配置在于TSA表项的配置，配置原理参见“[TSM配置TSA配置](#)”章节描述。

这里以16kbps的数据传输速率为例，将16kbps的通道的TSA_ENTRY[mask]配置为0x03（也可以是其他bit，如配置为0x6）。

PMC还可以支持大于64kbps的数据速率的应用场景。在该场景下，需要将多个时隙都分配给同一个HDLC逻辑通道。应用场景图可以参见[图12-14](#)的描述。



须知

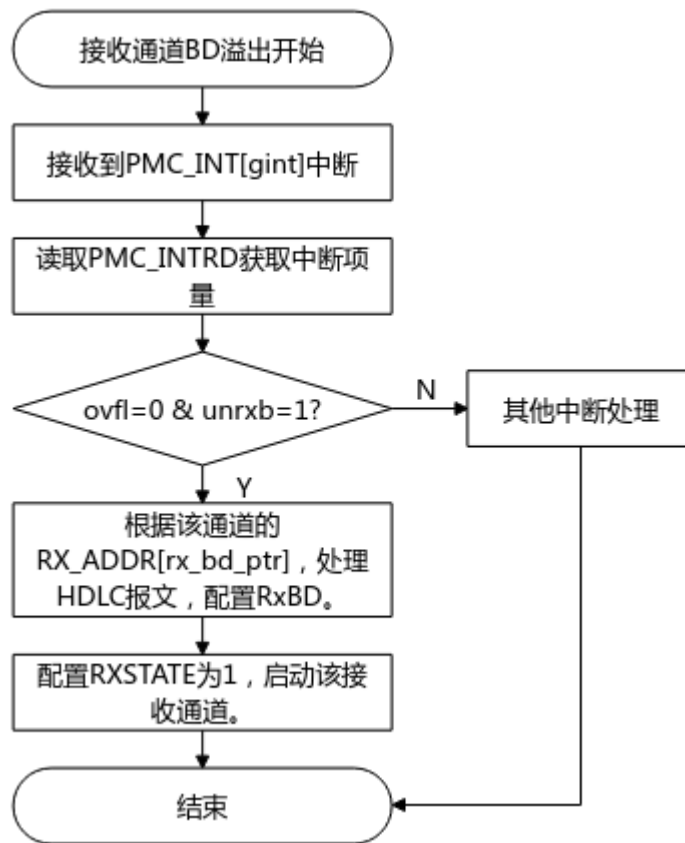
在使用连续时隙时，需要靠PMC的性能约束。请参见“[性能约束](#)”章节描述。

12.5.11 BD 溢出异常处理

PMC发送报文时，软件按照报文发送流程进行处理，则不会出现没有数据发送的情况，即不会出现发送BD溢出异常。

PMC接收报文时，在PMC接收性能大于软件处理BD性能的场景下，会出现INT_ENTRY[unrxb]的中断（即接收BD溢出异常），即一帧HDLC报文没有接收完成时，外存中没有DataBuffer存放帧数据。接收BD溢出异常处理流程见[图12-43](#)。

图 12-43 PMC 接收 BD 溢出处理流程图



软件处理步骤为：

步骤1 接收到PMC_INT[gint]中断后，读取PMC_INTRD，获得通道的中断向量。

说明

通过通道的中断向量，获得当前中断的通道号[ch_num]，用于后续通道参数寻址。

步骤2 判断当前中断向量是否为没有RxBD的中断。如果是[unrxb]中断，则执行[步骤4](#)，否则执行[步骤3](#)。

说明

没有RxBD的中断即是[ovfl]=0且[unrxb]=1。

步骤3 进入其他中断的处理流程。然后结束。

步骤4 根据该逻辑通道的RX_ADDR[rx_bd_ptr]指针，处理通道的报文和数据，然后配置RxBD可以为空，以接收报文。

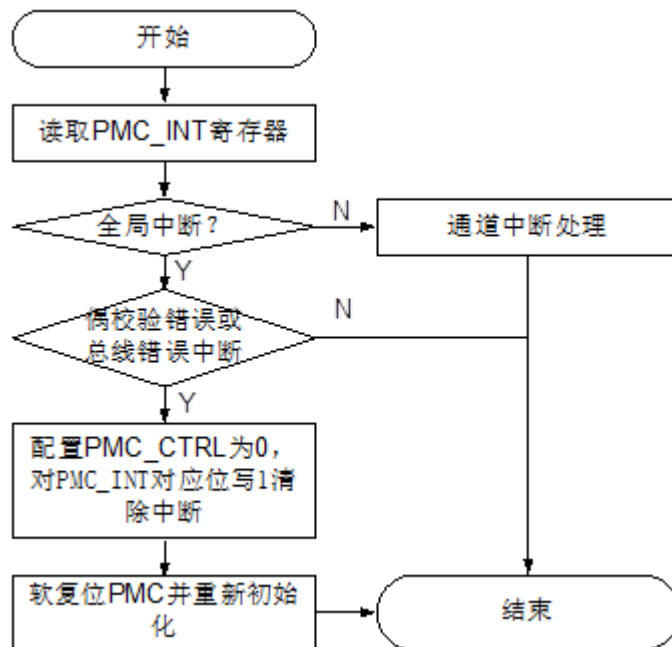
步骤5 配置该通道的RXSTATE为1，启动报文接收。

----结束

12.5.12 总线错误与 RAM 失效处理

PMC工作中，发生总线错误和RAM失效，都是致命的错误类型。此时软件需要及时的对PMC进行处理。软件操作的流程图如图12-44所示。

图 12-44 PMC 全局异常中断处理流程图



软件操作步骤为：

步骤1 读取PMC_INT中断，判断是否为[pari_err]或者[bus_err]中断。是，则执行**步骤3**；否则执行**步骤2**。

步骤2 进入通道中断处理流程，并结束。

步骤3 配置PMC_CTRL为0，立即禁止PMC工作。

步骤4 对PMC_INT对应位写1，清除中断。

步骤5 按照13.5.3 软复位流程进行配置，将PMC复位。

步骤6 按照13.5.2 初始化流程重新初始化PMC。

----结束



12.6 调试（Debug）

说明

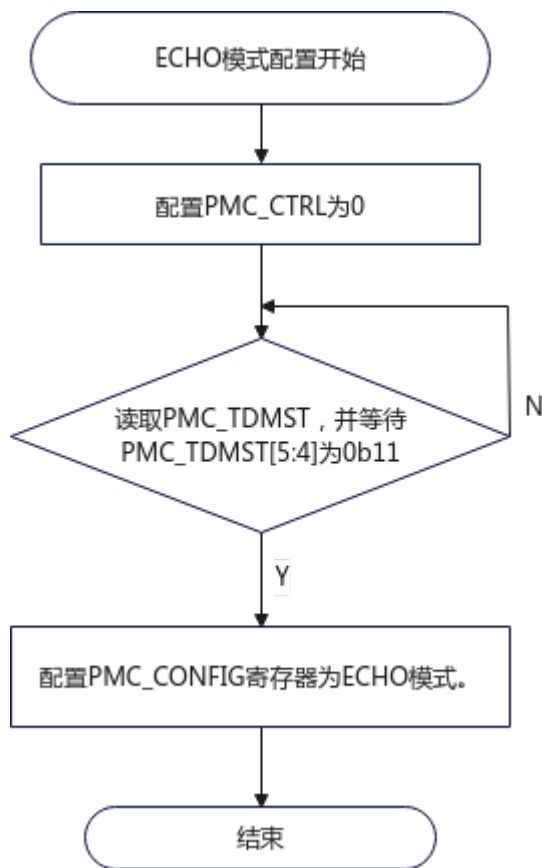
测试模式下，要求PMC工作在同相共享模式。

12.6.1 外环回测试模式

外环回模式主要用于对端逻辑测试线路是否正确的模式。

图12-45说明了外环回模式的配置流程。

图 12-45 外环回测试模式配置流程图



外环回模式配置步骤为：

步骤1 配置PMC_CTRL为0，禁止TDM总线。

步骤2 读取PMC_TDMST，判断PMC_TDMST[5:4]是否为0b11，用于TDM总线是否已经被禁止。若已被禁止，则执行**步骤3**，否则执行**步骤3**。

步骤3 配置PMC_CONFIG[18]为1，将PMC配置为外环回模式。

----结束



12.6.2 通道环回测试模式

PMC支持基于通道的内环回模式。该模式的配置步骤为：

- 步骤1** 按照通用切换模式，仅将发送通道的TSM和TSA进行配置。
- 步骤2** 将需要内环回通道的TSM表项配置为环回模式，即TSM_ENTRY[mode]为2' b10。
- 结束

12.6.3 心跳检测

PMC不支持纯硬件的心跳检测，只能提供软硬件结合的方式对PMC进行周期性检测，判断PMC是否正常工作。为了实现该功能，软件需要维护一个PMC正在工作的通道列表。基于该列表，可以通过定时任务或者外部触发，来启动心跳检测程序。

心跳检测的步骤为：

- 步骤1** 根据软件维护的通道列表，读取该通道的TXSTATE或RXSTATE。

说明

每个逻辑通道有发送和接收两个方向，软件需要维护发送和接收的信息。因为部分通道可以仅发送功能，部分通道仅接收功能；部分通道可同时接收和发送。

- 步骤2** 判断TXSTATE[0]是否为1，是则表示该发送通道正常工作；否则执行**步骤4**。

- 步骤3** 判断RXSTATE[0]是否为1，是则表示该接收通道正常工作；否则执行**步骤4**。

- 步骤4** 启动定时，等待10s后，再执行**步骤2**。通过全局计数器，计数当前是第几次检查操作。若当前是第10次该操作，则执行**步骤6**。否则执行**步骤5**。

说明

循环多次对同一个通道的检测，保证对通道的检测是正确的。

- 步骤5** 清零全局计数器。根据通道列表，读取下个通道状态，跳转到**步骤2**。

- 步骤6** 检测到PMC异常，通知CPU对PMC进行软复位。

----结束

12.7 注意事项

通道切换过程中，产生碎帧的解决措施。

在业务中如果动态增删通道，容易出现碎帧的情况。

解决方案如下：

- 通过在PMC和上层函数接口处，停止PMC接收数据上送和发送数据下发。硬件PMC和驱动的中断函数仍然维持通道数据的传输。

解决方案影响：

- 在通道禁止的时候，仍然会受到小部分的数据；但由于增删通道不会频繁发生，且PMC通道的流量很小，所以对CPU的利用率无影响。
- PMC通道可以正常的处理误码帧，且对软件的改动小。



13 CodecTDM

13.1 概述

CodecTDM（Codec Interface TDM）是TDM接口的数据缓存处理模块，主要功能是实现TDM接口与ARM多核之间的数据转换及缓存，也可以实现时隙之间的数据交叉功能。

CodecTDM模块支持1路TDM接口，最高速率32.768Mbps。提供最大512个时隙，每个时隙8bit的数据接收发送处理。

CodecTDM模块内部收发各提供2个5ms的数据乒乓缓存，内部集成DMA功能。接收每存满5ms数据，DMA到主存。发送每发完5ms数据，读取主存中的数据。

CodecTDM模块发送数据提供时隙交叉功能，可以根据时隙交叉表项TSCT完成接收时隙数据到发送时隙数据的交叉功能。

CodecTDM模块中从TDM侧到ARM侧定义为上行；从ARM侧到TDM侧定义为下行。

13.2 关键术语

表 13-1 CodecTDM 关键术语描述

术语	描述
CodecTDM	Codec Interface TDM 编解码接口处理TDM。
TDM	Time Division Multiplexing时分复用模式。
TSCT	TimeSlot Cross Table时隙交叉表项。
DMA	Direct Memory Access 直接内存存取。
AHB	Advanced High-Performance BUS 高性能总线。
AXI	Advanced eXtensible Interface 是ARM公司在APB和AHB总线基础上发展出来的第三代高性能系统总线，它面向的是高速高性能的SOC互联。



13.3 功能特性

CodecTDM模块支持以下特性：

- 提供1路TDM总线接口，仅支持4线模式，且由芯片外部提供TDM总线时钟和帧同步信号。TDM总线接口参数可配。
- 支持最大512个时隙，每个时隙8bit的数据串并、并串转换功能。
- 支持最大512路时隙通道的数据缓存功能。收发各缓存10ms（80帧）TDM数据。
- 支持集成硬件DMA。内部接收发送缓存数据通过AXI master总线与CPU交互。
- 支持512个时隙的数据交叉功能。
- CodecTDM模块工作时钟400M。TDM接口时钟作为内部信号处理，不作为时钟信号。
- CodecTDM模块复位信号由外部输入，低有效。
- CodecTDM模块因TDM接口的时序差异，提供输出数据tdm_tx在fsp的前1拍，后1、2拍、当拍时钟输出的功能（默认当拍输出）。
- CodecTDM模块TDM接口速率最大32.768Mbps。
- 模块上下行数据处理的典型延时为5.125ms+256μs=5.381ms。
- 交叉数据延时范围125μs~250μs。

DFX特性：

- 支持TDM接口内环回模式和外环回模式。在环回模式下，TDM总线必须工作在4线模式，及发送和接收方向的频率和时序相同。
- 支持基于时隙的内部缓存后的数据环回模式。通过配置时隙交叉表项TSCT实现。将各时隙上行数据环回到下行，也可以将下行数据环回到上行。
- 支持帧个数统计，提供不饱和计数器进行统计。
- 支持帧同步信号FSP的异常检测，并通过寄存器记录错误信息供查询。
- 支持上行数据CPU未及时处理的数据踩踏检测，提供踩踏统计饱和计数器，软件可清零；同时上报数据踩踏中断及DDR踩踏地址。
- 支持下行数据未准备好检测，提供断流饱和计数器，软件可清零。同时上报数据断流中断及DDR断流地址。
- 因TDM侧时序固定，内部处理经过40帧则会固定切换乒乓MEM处理；而上行数据写入DDR受AXI master总线响应时间限制，如果总线出现拥塞，在5ms时间未完成DDR数据写入，则丢弃数据并提供数据丢失中断及统计，同时上报DDR数据丢失地址。
- 下行处理在TDM侧乒乓切换时，AXI master读总线因总线拥塞未完成乒乓数据读任务时丢弃数据并提供下行DDR数据丢失中断及统计，同时上报DDR数据丢失地址。
- 支持接收RDMEM、发送TDMEM、TSCT表项、CXDM的MEM的ECC告警统计及中断上报。支持1bit和多bit的ECC中断上报及错误地址指示。
- 对上行写DDR数据总线返回错误时提供总线写响应中断及统计。同时上报首次出现error的总线错误类型及地址信息。
- 对下行读DDR数据总线返回错误时提供总线读响应中断及统计。同时上报首次出现error的总线地址信息。

- 提供上行接收40帧数据的帧失步中断，同时上报帧失步的DDR地址及指示失步帧数。
- 支持TDM_CLK的异常检测，提供时钟错误饱和计数器及时钟丢失饱和计数器。

须知

CodecTDM模块中所有接收（RX）、发送（TX）都是针对TDM侧定义。

13.4 功能描述

13.4.1 应用框图

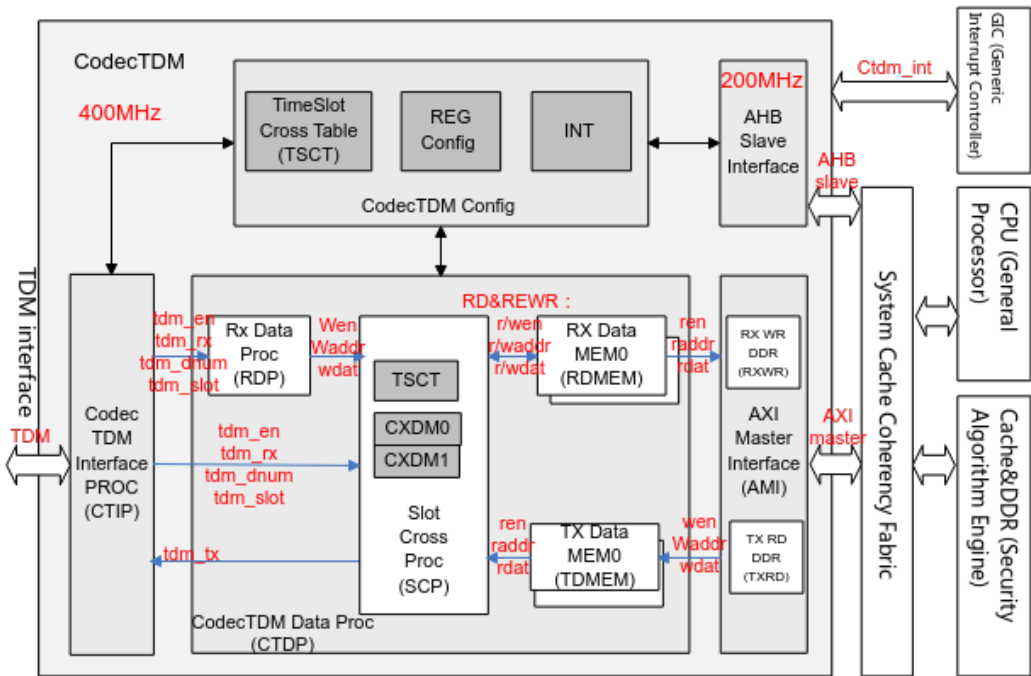
CodecTDM的应用如图13-1所示。

CPU通过AHB Slave总线，以及AHB slave模块和CTDM Config模块配置CodecTDM模块的寄存器及表项，完成对CodecTDM模块的设置。

CodecTDM模块根据配置，完成TDM接口数据的转换、存储及交叉。

CodecTDM模块通过AXI master总线操作DDR空间完成和CPU之间的数据交互。

图 13-1 CodecTDM 应用图



CodecTDM模块是IO处理模块，外部IO接口为TDM总线接口，最大支持32Mbps。IO接口信号见表13-2。

CodecTDM模块内部接口有AXI master总线接口、AHB slave总线接口。



CodecTDM模块内部出现交互需CPU干扰或者错误时通过中断信号经GIC模块上报CPU。

13.4.2 相关接口

在系统中，CodecTDM模块是IO处理模块，外部IO接口是TDM总线接口；
CodecTDM模块内部接口有AXI master总线接口、AHB slave总线接口。
分别描述如下：
TDM接口：通过TDM总线IO接口接收发送TDM数据；TDM接口仅支持4线模式，信号描述如表13-2所示。

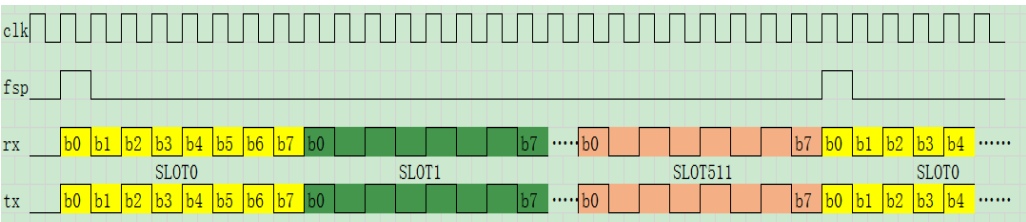
表 13-2 TDM 接口信号描述

名称	方向	描述	对应管脚
CTDM_CLK	I	CodecTDM接收方向总线时钟信号，在4线模式下，发送方向也使用该时钟信号。	CTDM_CLK
CTDM_FSP	I	CodecTDM接收方向帧同步信号，在4线模式下，发送方向也使用该帧同步信号。	CTDM_FSP
CTDM_RX	I	CodecTDM接收数据线。	CTDM_RX
CTDM_TX	O	CodecTDM发送数据线。	CTDM_TX

13.4.3 内部存储功能

CodecTDM模块的TDM入口时序如图13-2所示。

图 13-2 CodecTDM 接口 IO 时序图



须知

该时序是FSP高有效，且时钟为32.768MHz，帧头数据和帧同步信号的对齐关系dfsp=0，接收时钟和数据上升沿对齐的时序。
rx，tx数据根据实际数据变化。



在TDM的1帧中时序固定，512个时隙时分复用，每个时隙都接收发送8bit数据。

根据接口时序，内部MEM各时隙数据按地址独立存储。

每帧内各时隙写入MEM的8bit数据偏移bit位相同，可以使用同1个bit位标记或者掩码表示。

MEM读写数据分为乒乓存储，当写pp=0的MEM时，pp=1的MEM数据可以读，读写控制根据TDM侧数据处理40帧数据完成切换乒乓标记。

TDM时隙数据到MEM的位序可配，先到bit存入高位还是低位可配。

DDR中的字节序支持三种可配，分别为DDR数据大端、DDR数据32bit小端、DDR数据64bit小端。如图13-3所示。

发送数据MEM处理和接收数据类似，读写侧处理路径刚好相反。

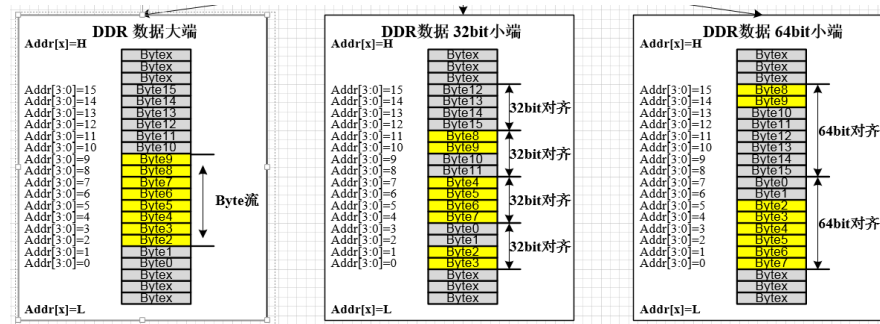
图 13-3 CodecTDM 接口内部 TDM 数据乒乓 MEM 存储格式

乒乓ind	addr[9:0]	frm39	frm38	frm37		frm1	frm0
pp0	slot0	8bit	8bit	8bit		8bit	8bit
	slot1	8bit	8bit	8bit		8bit	8bit
	slot2	8bit	8bit	8bit		8bit	8bit
							
	slot509	8bit	8bit	8bit		8bit	8bit
	slot510	8bit	8bit	8bit		8bit	8bit
	slot511	8bit	8bit	8bit		8bit	8bit
pp1	slot0	8bit	8bit	8bit		8bit	8bit
	slot1	8bit	8bit	8bit		8bit	8bit
	slot2	8bit	8bit	8bit		8bit	8bit
							
	slot509	8bit	8bit	8bit		8bit	8bit
	slot510	8bit	8bit	8bit		8bit	8bit
	slot511	8bit	8bit	8bit		8bit	8bit

图 13-4 MEM 存储结构图

	5ms=32*8=40帧		
DDR起始地址	512*320bit		DDR中单块5ms数据存储
tdm_base_addr	5ms	int	slot0-40byte (24byte null)
	5ms	int	slot1-40byte (24byte null)
	5ms	int	slot2-40byte (24byte null)
	5ms	int	
	5ms	int	slot509-40byte (24byte null)
tdm_mid_addr	5ms	int	slot510-40byte (24byte null)
	5ms	int	slot511-40byte (24byte null)
	5ms	int	
	5ms	int	
tdm_full_addr	5ms	int	

图 13-5 DDR 缓存块数据字节序



- 接收方向：
 - 内部MEM写数据：根据TDM时序，每帧只写MEM0或MEM1的1列8bit数据，按照帧计数frm_cnt依次写入，写完40帧，单块MEM写完，切换乒乓标记。
 - 内部MEM读数据：根据时隙有效配置及AXI master总线时序，每次读出乒乓MEM中的1个时隙的40帧数据320bit写入DDR相应地址。写入DDR时按照64字节写入，只使用前40字节，后24字节填0。
 - 上行数据每存满5ms数据，搬移到主存；完成后上报整块5ms数据写完中断及DDR中本次5ms块起始地址。
- 发送方向：
 - 内部MEM写数据：根据时隙有效配置及AXI master总线格式，每次读出DDR中相应地址的1个时隙的40帧数据320bit写入MEM对应时隙地址。DDR内部缓存分时隙按照64字节存储，只使用前40字节，后24字节填0。
 - 内部MEM读数据：根据TDM时序，每次读出有效时隙的8bit数据发送到TDM总线。TDM侧处理完40帧数据，切换乒乓标记。
 - 下行每发送完5ms数据，读取主存中的数据；完成后上报整块5ms数据读完中断及DDR中本次5ms块起始地址。
- 配置改变：
 - 数据存储处理和TDM接口速率相关，上述描述均是配置为32.768Mbps的处理。如果配置为8.192Mbps，16.384Mbps时，时隙相应减少为128和256，则MEM空间分配不变，只使用前128或者256个地址。
 - DDR中的空间分配也不变，只是使用每块的前128或者256个地址。

须知

TDM侧数据发送的无效时隙及CPU未及时准备好数据时发送寄存器配置的固定数据，该数据需要提前写入MEM。

13.4.4 数据交叉功能

CodecTDM模块同时根据CPU的表项配置，完成接收数据到发送数据的时隙交叉。



表 13-3 时隙交叉表项 TSCT 格式

addr[9]	addr[8:0]	bit10	bit[9:0]
rx=0	0	ts_cx_en	ts_cx_slot
	1	ts_cx_en	ts_cx_slot
	2	ts_cx_en	ts_cx_slot
	-		
	-		
	509	ts_cx_en	ts_cx_slot
	510	ts_cx_en	ts_cx_slot
	511	ts_cx_en	ts_cx_slot
tx=1	0	ts_cx_en	ts_cx_slot
	1	ts_cx_en	ts_cx_slot
	2	ts_cx_en	ts_cx_slot
	-		
	-		
	509	ts_cx_en	ts_cx_slot
	510	ts_cx_en	ts_cx_slot
	511	ts_cx_en	ts_cx_slot

表 13-4 时隙交叉表项 TSCT 描述

信号	比特位	描述
ts_cx_en	[10]	时隙交叉的使能配置。该配置和CTDM_TS_VLD的配置一致。 0：时隙交叉不使能。配置不使能时，TDM总线发送CTDM_UNVLD_DATA寄存器配置值； 1：时隙交叉使能。
ts_cx_slot	[9:0]	时隙交叉的时隙号配置，支持1k个时隙交叉。 TDM上行时隙号范围0-511； TDM下行时隙号范围512-1023。

ts_cx_en配置如下：

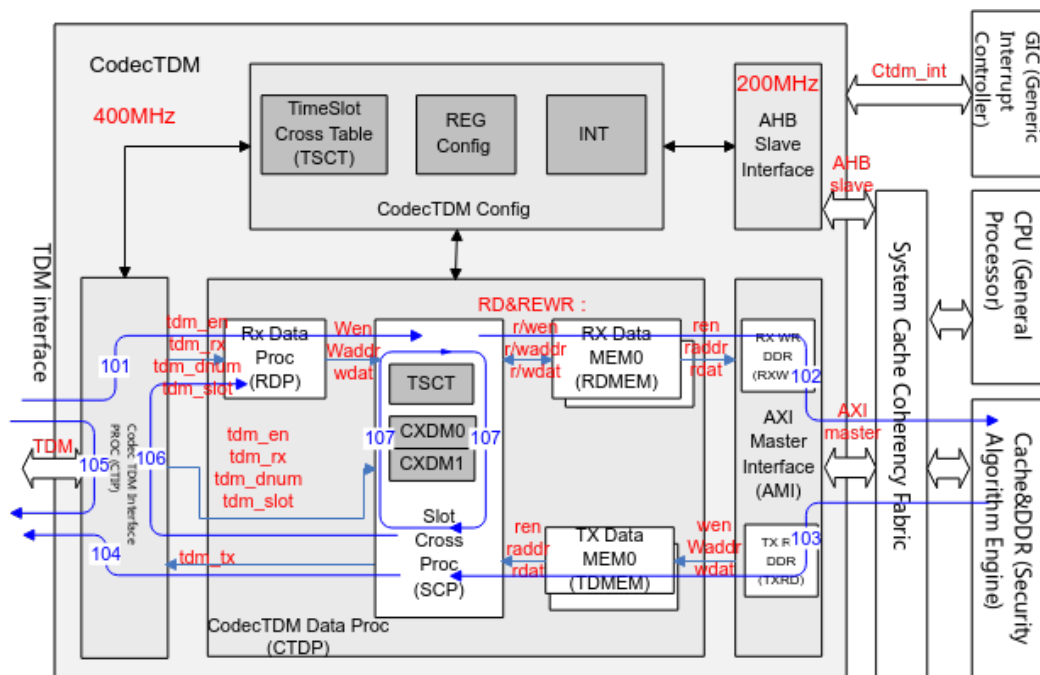
1：ts_cx_en=0：

时隙交叉不使能，表示时隙无效，该时隙发送CTDM_UNVLD_DATA寄存器配置的数据；

2: ts cx en=1:

时隙交叉使能, 该输出时隙地址发送ts cx slot交叉时隙的数据;

图 13-6 CodecTDM 接口内部数据流



数据交叉功能实现:

数据交叉功能集中在SCP模块。SCP接收侧模块入口TDM，出口TDM编号为时隙0-511；发送侧模块入口TDM，出口TDM编号为时隙512-1023。该模块根据时隙交叉表项TSCT的配置完成共1K个时隙之间的任意时隙的数据交叉功能。配置交叉不使能的时隙输出TDM UNVLD DATA寄存器配置的数据。

须知

AXI master总线只操作有效时隙的数据，减少总线浪费。所以DDR中的无效时隙的数据值可能是不确定数据。

13.5 编程接口

13.5.1 软复位

对接器件容许的情况下，可以任意时间对CodecTDM工作逻辑进行软复位，不会对系统造成影响。复位通过输入复位信号控制CodecTDM模块复位。



安全复位流程：

- 步骤1** CPU发起复位请求，配置CTDM_RST_REQ寄存器ctdm_rst_req为1。
- 步骤2** CodecTDM模块检测到ctdm_rst_req=1，对逻辑内部数据处理结束后（根据CTDM_RST_MODE配置是否在tdm_enb使能的情况下继续接收TDM侧的数据处理），将ctdm_rst_ready置为1，表示模块IDLE，可以复位。
- 步骤3** CPU将CodecTDM模块输入信号rst_n信号拉低，对ctdm_rst_req对CodecTDM进行复位。
- 结束

13.5.2 接收数据写 DDR

上行接收数据写入DDR：

- 步骤1** 接收数据乒乓切换变化后，查看本次ddr块的起始地址（根据配置的rd_base_addr和rd_ddr_len计算）对应CTDM_RX_DATA_READY的寄存器（最大支持32个DDR块，共32个寄存器0-31）的rx_data_ready值，若为1，表示CPU还未将本块上次数据读走，发生踩踏，上报数据踩踏中断，不影响后续操作。
- 步骤2** CodecTDM模块读取时隙有效配置ctdm_ts_vld，查找到第1个有效时隙。
- 步骤3** CodecTDM模块读出第1个有效时隙的RDMEM数据根据AXI master总线时序写入DDR地址，地址根据本次写入ddr块的起始地址及有效时隙计算。
- 步骤4** 写入结束后，继续执行**步骤2**和**步骤3**，直到写完最大512个时隙。
- 步骤5** 本次5ms数据写入完成后，CodecTDM模块将CTDM_RX_DATA_READY寄存器中和DDR中的5ms块对应的寄存器值置1，表示本块5ms接收数据准备完毕，CPU可以读取使用。同时上报接收DDR主存5ms数据块写完中断及DDR本次5ms块起始地址。
- 步骤6** CPU响应中断，读取对应CTDM_RX_DATA_READY寄存器中的值及5ms块起始地址，读取需要的5ms块数据，读完后将CTDM_RX_DATA_READY寄存器对应块的寄存器值清零，本次数据块搬移完成。
- 结束

表 13-5 CTDM_RX_DATA_RDY 寄存器描述（0x0500+dn*0x4）共 32 个寄存器（具体参见寄存器描述）

reserved	31:1	RO	0x00000000	保留。
rx_data_ready	0	CHRW	0x0	该寄存器的1bit代表主存中的1个5ms数据地址块。寄存器0表示接收主存中的起始块，地址0-31对应主存的0-31块。根据rd_ddr_len寄存器的块配置使用对应寄存器。 CodecTDM模块每写完1个5ms数据块，将对应寄存器bit置1；CPU将数据读走后，将对应寄存器清零。



13.5.3 发送数据读 DDR

下行发送数据从DDR读出：

- 步骤1 CPU按照配置的速率准备好DDR中的发送数据，每次写入DDR数据时，要查看DDR块空间对应的CTDM_TX_DATA_READY寄存器的值（最大支持32个DDR块，共32个寄存器0-31。CTDM模块内部按照5ms块依次处理，CPU可以根据读完成中断的附带DDR地址得到CTDM模块当前处理的5ms块），如果对应tx_data_ready为0可以写入，为1表示数据还未处理，CPU需等待。
- 步骤2 CPU写入结束后，将CTDM_TX_DATA_READY寄存器的值置1，表示数据准备好。
- 步骤3 CodecTDM模块数据乒乓切换后，查看本次ddr块的起始地址对应CTDM_TX_DATA_READY寄存器的值，为1执行步骤4；CTDM_TX_DATA_READY寄存器值若为0，表示CPU未准备好数据，上报数据断流中断，再执行步骤4（同时注意步骤5，不读取DDR数据，写入TDMEM的数据为TDM_NRDY_DATA寄存器配置的值）。
- 步骤4 CodecTDM模块读取时隙有效配置ctdm_ts_vld，查找到第1个有效时隙。
- 步骤5 根据AXI master总线时序发起第1个有效时隙的DDR数据读取，地址根据本次操作的ddr块的起始地址及有效时隙计算，读出数据写入TDMEM对应时隙地址；若对应CTDM_TX_DATA_READY寄存器值若为0，写入的数据为TDM_NRDY_DATA寄存器配置的值。
- 步骤6 读出数据结束后，继续执行步骤4和步骤5，直到读完最大512个时隙。
- 步骤7 本次5ms数据处理完成后，将CTDM_TX_DATA_READY寄存器中和DDR中的5ms块对应的寄存器值置0，表示本块5ms数据读完成，CPU可以继续写入。同时上报发送DDR主存5ms数据块读完中断及DDR本次5ms块起始地址。

----结束

表 13-6 CTDM_TX_DATA_READY 寄存器描述（0x0580+dn*0x4）共 32 个寄存器（具体参见寄存器描述）

reserved	31:1	RO	0x00000000	保留。
tx_data_ready	0	CHRW	0x0	该寄存器的1bit代表主存中的1个5ms数据地址块。寄存器0表示发送主存中的起始块，地址0-31对应主存的0-31块。根据td_ddr_len寄存器的块配置使用对应寄存器CPU每写完1个5ms数据块，将对应寄存器bit置1；CodecTDM模块将数据读走后，将对应寄存器清零。

13.6 应用配置

无。

13.7 初始化

CodecTDM在上电复位或软复位后，在使用前需初始化：



- 步骤1** 对CodecTDM内部RAM进行初始化，配置寄存器CTDM_MEM_INIT_START中各MEM的初始化使能为1。
- 步骤2** 检查CodecTDM内部RAM初始化是否完成，检查状态寄存器CTDM_MEM_INIT_DONE。如果寄存器值相应bit为0，表示仍在初始化，如果寄存器值为1，表明初始化已完成。
- 步骤3** 配置时隙交叉表项TSCT，包括时隙交叉使能和交叉时隙。
- 步骤4** 配置CodecTDM内部模式及处理相关的配置寄存器；包括中断使能、TDM位序、DDR字节序列模式、时隙无效或者断流时的数据配置、读/写DDR空间基地址、DDR空间长度、MEM ECC使能、时隙有效配置寄存器。
- 步骤5** 配置AXI总线的相关寄存器。
- 步骤6** 配置TDM接口模式CTDM_TDM_MODE寄存器，包括TDM频率，数据采样打拍沿，fsp有效电平，fsp和数据对齐关系等，以及重定帧的帧数配置。
- 步骤7** 配置TDM接口使能（在CTDM_TDM_EN寄存器），模块开始工作。

----结束

13.8 注意事项

无。

13.9 DFX

CodecTDM 模块环回

支持TDM侧总线接口内部环回和外部环回。

CodecTDM 查询，提供状态机统计查询

- 统计
 - 支持帧同步信号FSP的异常检测，并通过寄存器记录错误信息供查询，参见TDM_FSP_LOF_CNT寄存器和TDM_FSP_ERR_CNT寄存器；
 - 支持帧个数统计，提供不饱和计数器，参见TDM_FRM_CNT寄存器
 - 支持接收发送MEM的ECC告警统计及查询，参见CTDM_ECC_1BIT_CNT / CTDM_ECC_1BIT_CNT寄存器。
 - 支持总线读写次数统计，提供不饱和计数器。
 - 支持状态机状态查询。

正常中断

- 接收DDR主存块写完成中断。
模块接收数据存储到DDR主存空间，当写完1个5ms数据块时，上报接收DDR主存5ms数据块写完中断及DDR中本次5ms块起始地址。
- 发送DDR主存块读完成中断。
模块发送数据从DDR主存空间读出，当读完1个5ms数据块时，上报发送DDR主存5ms数据块读完中断及DDR中本次5ms块起始地址。



异常中断

- 上行数据踩踏中断。
支持上行DDR数据软件未及时处理的数据踩踏检测，提供踩踏统计饱和计数器，软件可清零；同时上报数据踩踏中断及DDR踩踏地址。
- 下行数据断流中断。
支持下行DDR数据未准备好检测，提供断流饱和计数器，软件可清零。同时上报数据断流中断及DDR断流地址。
- 上行写DDR数据丢失中断。
上行处理在TDM侧5ms数据乒乓切换时，AXI master写总线未完成乒乓数据写任务时上报上行写DDR数据丢失中断，同时上报DDR数据丢失地址。
- 下行读DDR数据丢失中断。
下行处理在TDM侧5ms数据乒乓切换时，AXI master读总线未完成乒乓数据读任务时上报下行读DDR数据丢失中断，同时上报DDR数据丢失地址。
- 各个MEM的1bit的ECC中断。
提供TSCT、CXDM、RDMEM、TDMEM的1bit的ECC中断。同时上报首次出现错误的MEM及地址。如果同时出现，按照优先级上报MEM地址。
- 各个MEM的2bit的ECC中断。
提供TSCT、CXDM、RDMEM、TDMEM的2bit的ECC中断。同时上报首次出现错误的MEM及地址。如果同时出现，按照优先级上报MEM地址。
- 上行数据写总线error中断。
总线写请求返回error中断。同时上报首次出现error的总线错误类型及地址信息。
- 下行读数据error中断。
总线读请求返回error中断。同时上报首次出现error的总线地址信息。
- 上行数据帧失步中断。
提供上行接收40帧数据的帧失步中断，同时上报帧失步的DDR地址及指示失步帧数。

13.10 性能指标

13.10.1 吞吐率（Throughput）

CodecTDM模块性能较低，由TDM接口带宽决定，收发最大32.768Mbps。

13.10.2 延迟（Latency）

- 上行、下行数据存在内部乒乓5ms缓存、交叉1帧缓存延时及总线延时，典型延时为 $5.125\text{ms} + 256\mu\text{s} = 5.381\text{ms}$ 。
- 交叉数据存在 $125\mu\text{s} \sim 250\mu\text{s}$ 的延时。



14 HDLC

14.1 关键术语

无

14.2 概述

高层数据链路控制HDLC（High_Level Data Link Control）是数据链路控制协议，包括面向比特的同步HDLC和面向字节的异步HDLC。

14.3 特性

HDLC的主要特性如下：

- 支持HDLC模块与CPU之间通过AHB总线，用BD（Buffer Descriptor）方式交互数据。
 - 需要注意：HDLC的寄存器空间属性必须是“Strong Order”。
- 支持CPU通过AHB总线控制HDLC工作，下发配置信息以及回读工作状态的上报信息。
- 包括一个总线Master AHB接口，总线位宽为32bits。
- 提供2个HDLC通道（每通道包括1个发送通路和1个接收通路），分别为通道0和通道8，每个通道均可独立配置自身特性。
 - 支持单个通道使能和无效。
 - 支持单个通道立即停止工作。
 - 支持单个通道独立复位（同步复位信号，通过寄存器配置）。
 - 支持单个通道复位后重新启动。
- 支持2个HDLC通道的通道仲裁，仲裁策略支持同等优先级和固定优先级可配。
- 提供2个HDLCBus，对应2个HDLC通道。
- 支持3种可屏蔽组合中断：通道0中断/通道8中断/通道仲裁中断。
- 支持7.68 Mbps，兼容3.84Mbps。



- HDLC.DFX001：提供多种通道内部数据环回、端口PN18测试和数据采样测试功能。

说明

HDLC的AHB slave接口只有hready_out，无hready_in，所以在集成时，需要在外部做处理，解决背靠背访问HDLC和相邻AHB模块时出现问题。

14.4 信号描述

表 14-1 HDLCBUS 接口信号描述

名称	方向	描述
HDLC_RXD0	I	HDLC通道8接收信号。
HDLC_TXD0	O	HDLC通道8发送信号。
HDLC_CLK0	I	HDLC时钟0。
HDLC_CLK1	I	HDLC时钟1。
HDLC_RXD1	I	HDLC通道0接收信号。
HDLC_TXD1	O	HDLC通道0发送信号。

须知

请注意接口名称与HDLC通道的对应关系。

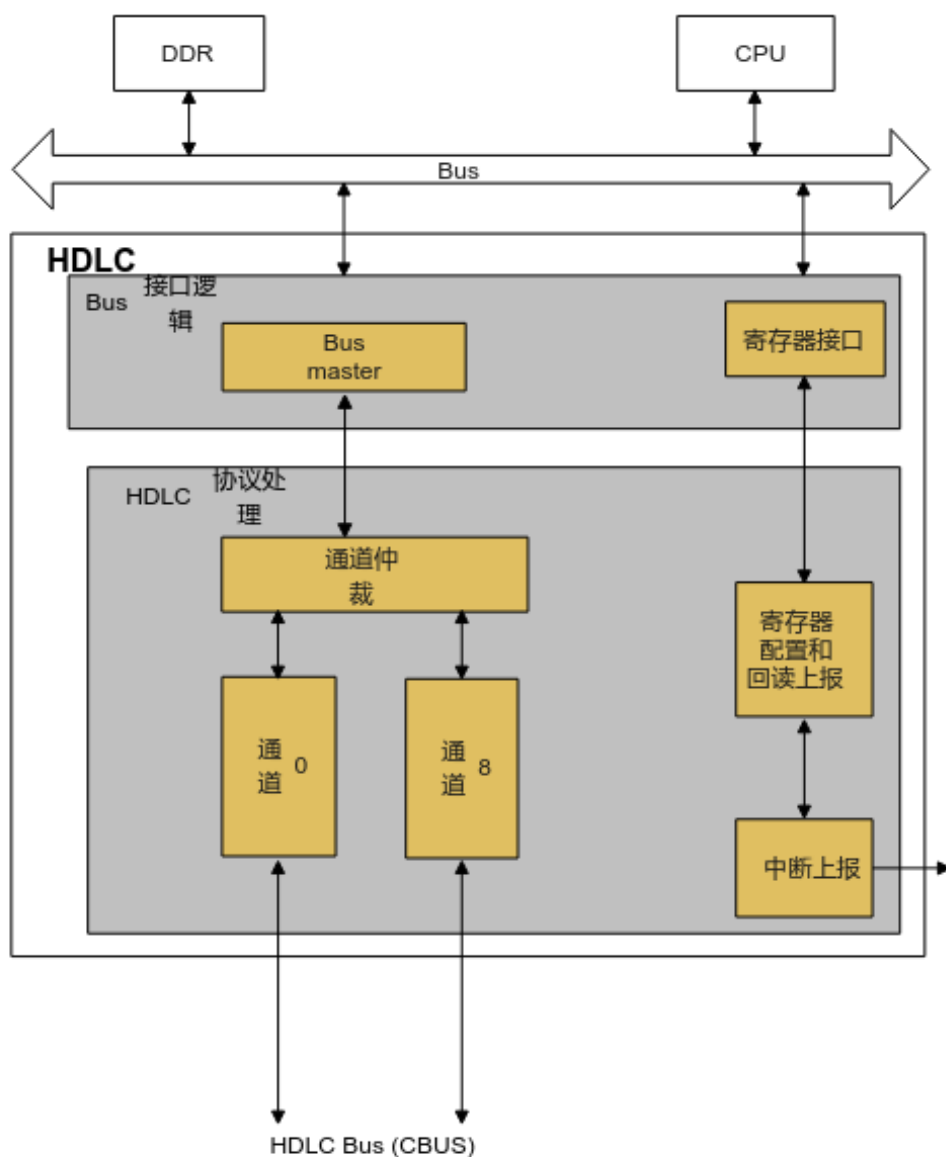
HDLC对于大于内部主工作时钟周期的接口时钟毛刺不能容忍，一定会引起数据收发问题。对于小于内部主工作时钟周期的接口时钟毛刺不一定能容忍，如果这个毛刺正好在内部时钟的上升沿也将引起收发问题。总之，HDLC对接口时钟要求较高，几乎无毛刺容忍能力。

14.5 功能描述

14.5.1 结构框图

HDLC的结构框图如图14-1所示。

图 14-1 HDLC 的应用框图





说明

HDLC单个发送通路对AHB总线延迟有一定要求：

- 点对点模式下，最大容忍延迟的理论值为 $8 \times 32 \times$ （内部主时钟频率/接口时钟频率）（单位：内部主时钟周期），Hi309a HDLC AHB总线访问内存效率需要仿真评估后得出。
- BUS模式下，如果非连续发帧（非连续发帧是指发送完一帧后下一个burst8数据并没有写入发送源FIFO，发送源FIFO空。），最大容忍延迟的理论值为 $n \times$ （内部主时钟频率/接口时钟频率）（单位：内部主时钟周期，n为高优先级获取总线需要等待连续逻辑1的个数。）如果是连续发帧，最大容忍延迟的理论值为 $8 \times 30 \times$ （内部主时钟频率/接口时钟频率）（单位：内部主时钟周期）。在容忍范围内，HDLC需要获得数据，否则会发生断流或发送通路停止。发生此类问题可以通过软复位发送通路恢复。

HDLC单个接收通路对AHB总线延迟有一定要求：

- 最大容忍延迟的理论值为 $8 \times 33 \times$ （内部主时钟频率/接口时钟频率）（单位：内部主时钟周期）。在容忍范围内，HDLC需要将数据写回内存，否则会发生接收丢帧。对于这种情况，HDLC一般会统计为接收丢帧，并回写BD，但如果帧尾行长度小于CRC长度，则会被统计为碎帧，并不会上报BD。

14.5.2 HDLC 通道协议处理

HDLC通道模块完成数据流的上行HDLC组帧和下行HDLC解帧功能，每个HDLC通道的协议处理功能如下：

- 支持同步HDLC协议功能。
 - 发送支持帧头/尾标志0x7E的生成和插入。
 - 接收支持帧头/尾标志0x7E的删除。
 - 发送支持FCS（16/32bits可配）的生成和插入，使用16/32bits CRC-CCITT。
 - 接收支持FCS（16/32bits可配）的校验删除。
 - 支持CRC初始值/比较值可配。
 - 支持CRC字节反向可配。
 - 发送支持“0”插入。
 - 接收支持“0”删除。
- 支持2帧之间空闲时插入“Flag”或“Idle”可配。
- 发送支持2帧之间（帧头之前）插入（1~16）可配数量的“Flag”，调配发送速率。
- 接收支持相邻2帧之间共享1个“Flag”。
- 接收支持相邻2个“Flag”之间共享1个0。
- 支持帧长检测：
 - 支持超长帧和超短帧检测。
 - 超长帧帧边界可配，可配范围：2Byte~（65535-CRC）Byte。
 - 超短帧帧边界可配，可配范围：2Byte~（65535-CRC）Byte。
- 支持Abort帧的发送和接收。

说明

发送abort帧时，发送通路有时会停止有时会继续发送数据，这跟发送协议状态机回到IDLE状态时是否数据回来有关。因此希望发送abort帧产生中断时，软件能在处理中断时复位发送通路。

- 支持通过发送/接收BD跟CPU交互组帧前/解帧后的数据。



- 发送通路根据发送BD的数据缓存区地址和数据长度，取得裸数进行HDLC组帧。
- 接收通路根据接收BD中接收数据缓冲区地址和CPU通过寄存器配置的接收缓冲区长度，将解帧后的裸数送DDR；根据接收HDLC帧的解帧情况，更新接收BD的帧长和错误类型等接收帧信息。
- 支持1帧对应1个BD，不支持1帧对应多个BD。
- 支持各种类型帧的统计上报。
 - 发送通路支持已发帧/Abort帧（使用普通外部接口时，发送源FIFO一帧之内数据断流造成的发送Abort帧）/发送超长帧/发送超短帧，计数上报。
 - 接收通路支持碎帧/丢帧（使用普通外部接口时，接收目的FIFO满造成的丢帧）/已收帧/Abort帧/字节未对齐帧/CRC错误帧/超长帧/超短帧，计数上报。
- 接收支持自动过滤碎帧。

📖 说明

碎帧：已经检测到帧尾并且解帧去“0”处理后裁剪CRC之前，帧长小于1Byte的帧，或者帧长小于CRC长度的帧。

当HDLC打开碎帧过滤使能，且CRC长度为0，则对小于1字节且尾缀为0x7f的帧，会识别为abort帧，而碎帧计数器将不会进行统计。但是对其中的0x7e后紧跟0x7f或0xff的帧，也不会统计为abort帧，事实上，对这样的帧无任何统计。

- 提供可屏蔽组合中断。
 - 发送Abort帧。
 - 发送源缓存完成。
 - 接收目的缓存完成。
 - 发送帧完成。
 - 接收帧完成。
 - 发送BD没有准备好。
 - 接收BD没有准备好。
 - 发送超长帧。
- 提供测试功能，包括：
 - 支持关键状态机状态和主要FIFO信息上报。
 - 支持协议处理前、协议处理后和端口环回。
 - 支持端口处进行PN18测试。
 - 支持单通道数据采样和实时业务的数据采样。

14.5.3 通道仲裁

HDLC模块采用多通道设计，多个通道同时发起总线操作时需要进行通道仲裁，来选择其中某一通道。HDLC通道仲裁模块具有如下特性：

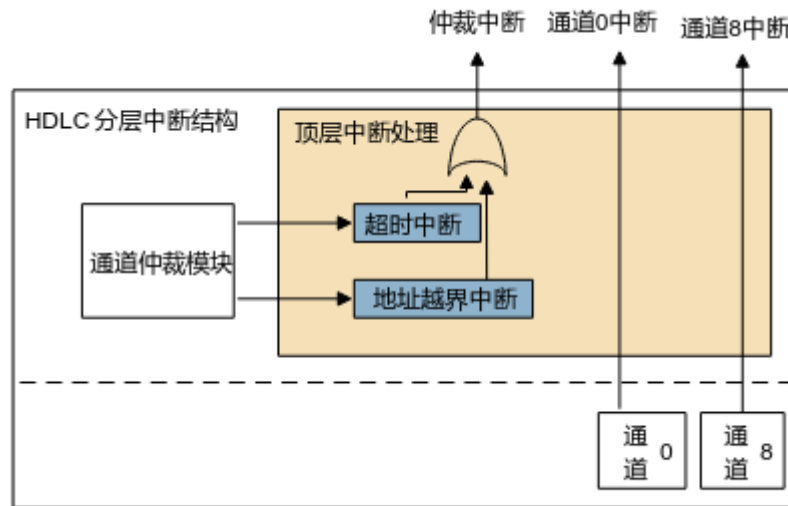
- 支持2个通道（共计4个通路，每个通道均包括一个接收通路和一个发送通路）的HDLC通道仲裁。
- 支持同等优先级和固定优先级可配。
- 同等优先级仲裁，采用轮询方式。
- 固定优先级仲裁，高优先级优先处理，同一优先级内部采用轮询方式仲裁。
- 支持单通路超时检测功能。

14.5.4 中断结构

HDLC模块提供通道0、通道8和通道仲裁三种可屏蔽组合中断上报。为方便用户使用，对中断处理结构和内容进行简要说明，如图14-2所示。

HDLC中断分层结构特点：各通道中断处理和顶层中断处理。

图 14-2 HDLC 分层中断结构



在图14-2中，每个通道提供一组可屏蔽组合中断，即包括中断状态/屏蔽/清除寄存器；每通道送一根高电平有效的中断信号给顶层处理。每通道提供的可屏蔽组合中断所包括的中断类型如下：

- 发送Abort帧。
- 发送源缓存完成。
- 接收目的缓存完成。
- 发送帧完成。
- 接收帧完成。
- 发送BD没有准备好。
- 接收BD没有准备好。
- 发送超长帧。

每种中断的具体含义请参见“[发送中断管理](#)”和“[接收中断管理](#)”。

在图14-2中，顶层中断处理实现以下2种功能：

- 生成通道0、通道8和通道仲裁三种中断。
 - 通道0中断：将通道0中断直接送中断管脚。
 - 通道8中断：将通道8中断直接送中断管脚。
 - 通道仲裁中断：提供通道仲裁模块的单通路处理超时中断处理，各包括一组中断状态/屏蔽/清除寄存器；单通路处理超时中断，用中断状态寄存器的4bits分别表示4个仲裁通路是否发生超时。

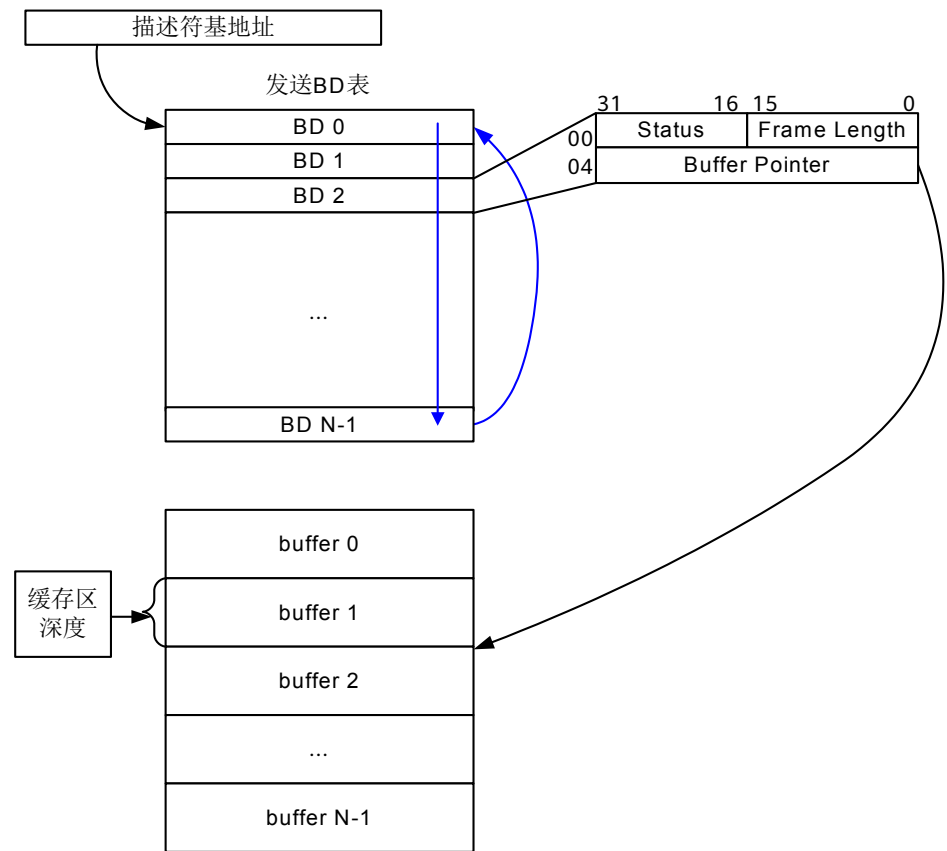
- 对以上3种中断，分别提供中断处理退出保护。

14.5.5 BD 数据结构

14.5.5.1 BD 管理结构

HDLC的缓存区结构是将BD放在某内存区域，数据放在另一内存区域中。BD的地址指针指向DDR中的数据缓存区。这种结构的最大好处是只要空间充足，可以自由的分配数据缓存区个数和缓存区的大小。BD管理数据缓冲区结构如图14-3所示。

图 14-3 HDLCBD 管理数据缓冲区



下面以发送为例进行说明：

- BD表

从图14-3可以看出，CPU通过发送BD基地址寄存器和接收BD基地址寄存器配置BD表的基地址。BD的基地址指向BD表的第一个BD的首地址，这个地址可以是HDLC可以访问空间的任何地址，其余的BD则可以通过如下公式进行计算：

$$ADDR_n = n * 8 + ADDR_0$$

说明

ADDR₀为接收/发送BD表首地址，n表示BD表中的第几个BD。例如：BD表大小为10，则n取值为0~9。

BD表中每一个BD依次排列，当处理到最后一个BD结束之后，再返回到表头。



- **BD结构：**
每个BD由8Bytes组成。其中，前2Bytes用来表示该缓存区的状态；其后2Bytes是用来描述帧长；最后4Bytes是数据缓存区指针，用来指向当前BD描述的数据缓存区的首地址。
- **发送方向工作过程：**
CPU将待发送的一帧数据放在片外DDR的数据缓存区中，将缓存区地址和帧长度等信息更新到发送BD表的下一个BD中，并将该BD的R比特配置为1。
HDLC在检测到CPU的启动脉冲之后，会自动读取下一个BD的内容。如果检测到BD中的R比特为1则视其为有效BD，并根据其中的缓存区地址和长度，到片外DDR中读取数据，进行HDLC组帧，送端口发送；当端口发送完成后，HDLC更新发送帧信息并将这个BD的R比特配置为0，交还CPU可用。
- **接收方向工作过程：**
 - CPU将片外DDR的空闲的数据缓存区分成段，每段接收缓存区长度为固定值，通过接收BD所指缓存大小寄存器配置给HDLC，CPU依次将每段缓存区的首地址更新到接收BD表的下一个BD中，并将该BD的R比特配置为1。
 - HDLC将端口接收到的HDLC帧进行解帧操作，同时检测接收BD表中是否有空闲有效的BD可用，将当前接收的帧长/错误类型等信息更新到有效的接收BD中，并将R比特配置为0，交还CPU使用。

说明

- 发送启动脉冲和接收启动脉冲，即CPU通知HDLC通道，发送/接收BD准备好了。
- 由于CPU与HDLC之间是通过按顺序操作BD来交互操作数据的，HDLC需要CPU在有新的BD准备好时，通知HDLC继续工作。如果CPU不通知HDLC，而通过HDLC搜索BD表检查有效BD的方式；将会在CPU无数据发送或者没有空闲有效的接收BD的情况下，造成HDLC频繁申请总线。

14.5.5.2 发送 BD 表

发送的源缓冲区采用BD管理。每个BD指出该缓冲区地址，长度，是否是帧内最后一个缓冲区等信息，如表14-2、表14-3所示。

表 14-2 HDLC 发送缓冲区 BD（Word0）

信号	bit	描述	配置描述
R	[31]	缓冲内容有效标识。 0：无效； 1：有效。	CPU检查内容无效时，可以更改BD，更改完成写入1，写入后在该位为0之前不应更改BD。 HDLC控制器检查内容有效时，在传送完成后更新BD，清零该位。
I	[30]	该BD结束后产生中断。	CPU写入，HDLC只读。1有效。
-	[29]	保留。	CPU写入，HDLC只读（不使用）。
LF	[28]	帧中最后一个BD。	CPU写入，HDLC只读。1有效。
LB	[27]	最后一个BD。	CPU写入，HDLC只读。1有效。
SL	[26]	超长帧。	HDLC写入，CPU只读。1有效。



信号	bit	描述	配置描述
SS	[25]	超短帧。	HDLC写入，CPU只读。1有效。
-	[24]	保留。	HDLC写入，CPU只读（全0）。
UN	[23]	发送数据没有准备好。	HDLC写入，CPU只读。1有效。
-	[22:16]	保留。	HDLC写入，CPU只读（全0）。
TLEN	[15:0]	发送缓冲长度。	CPU写入，HDLC只读。

表 14-3 HDLC 发送缓冲区 BD（Word1）

信号	bit	描述	配置描述
TDTPT	[31:0]	数据指针。	CPU写入，HDLC只读。 发送BD指向数据缓存区的首地址。

14.5.5.3 接收 BD 表

接收的目的缓冲区采用BD管理。每个BD指出环缓冲区地址、长度，是否是帧内最后一个缓冲区等信息，如表14-4、表14-5所示。

表 14-4 HDLC 接收缓冲区 BD（Word0）

信号	bit	描述	配置描述
R	[31]	缓冲为空标识。 0：不为空； 1：为空，可写。	CPU检查内容无效时，可以更改BD，更改完成写入1，写入后在该位为0之前不应更改BD。 HDLC控制器检查内容有效时，在传送完成后更新BD，清零该位。
I	[30]	该BD结束后产生中断。	CPU写入，HDLC只读。1有效。
-	[29]	保留。	HDLC写入，CPU只读。 （不关心0和1的变化。）
LF	[28]	帧中最后一个BD。	HDLC写入，CPU只读。1有效。
LB	[27]	最后一个BD。	CPU写入，HDLC只读。1有效。
SL	[26]	超长帧。	HDLC写入，CPU只读。1有效。
SS	[25]	超短帧。	HDLC写入，CPU只读。1有效。
NO	[24]	非字节对齐。	HDLC写入，CPU只读。1有效。
AB	[23]	Abort帧。	HDLC写入，CPU只读。1有效。



信号	bit	描述	配置描述
RC	[22]	CRC出错（帧中最后一个BD才有效）。	HDLC写入，CPU只读。1有效。
OV	[21]	接收缓存没有准备好。	HDLC写入，CPU只读。1有效。
-	[20:16]	保留。	HDLC写入，CPU只读（全0）。
RLEN	[15:0]	接收缓冲长度。	HDLC写入，CPU只读。

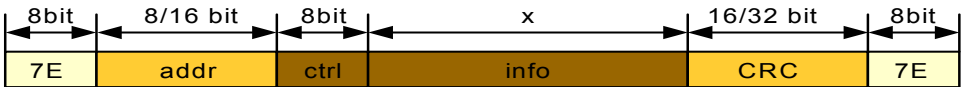
表 14-5 HDLC 接收缓冲区 BD（Word1）

信号	bit	描述	配置描述
RDTPT	[31:0]	数据指针。	CPU写入，HDLC只读。 接收BD指向数据缓存区的首地址。

14.5.6 标准 HDLC 帧与简化 HDLC 帧

标准的HDLC帧结构，包括完整的标志位、地址、控制字、信息和CRC校验。帧结构如图14-4所示。

图 14-4 HDLC 标准帧



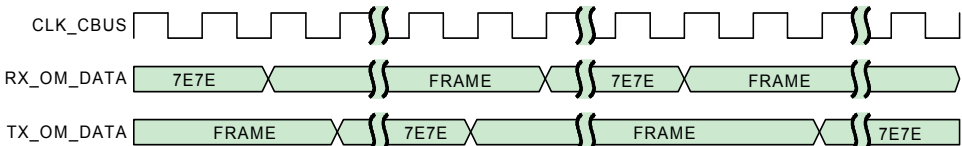
简化的HDLC帧结构，即HDLC不需要进行地址匹配检查，简化帧结构如图14-5所示。

图 14-5 HDLC 简化帧



工作在全双工模式下的时序如图14-6所示。

图 14-6 HDLCOM 接口全双工工作时序





14.6 工作方式

HDLC模块主要是配合CPU工作，完成对数据的HDLC协议处理。工作性质决定HDLC工作过程的特点如下：

- 启动、停止、复位等动作，以及各种静态工作参数，都是由CPU配置HDLC的寄存器进行控制。
- 工作过程中，依靠接收/发送BD与CPU交互HDLC数据。
- 工作过程中，通过中断和状态寄存器，向CPU报告工作状态。

14.6.1 管脚复用配置

HDLC管脚和GPIO复用。

- 使能HDLC通道，需要配置IOCONFIG模块寄存器IOMG182[1:0]为0b01。
- 使能HDLC0通道，需要配置IOCONFIG模块寄存器IOMG183[1:0]为0b01；配置IOCONFIG模块寄存器IOMG184[2:0]为0b001。
- 使能HDLC1通道，需要配置IOCONFIG模块寄存器IOMG185[2:0]为0b001；配置IOCONFIG模块寄存器IOMG186[0]为0b1。

14.6.2 软复位

对接器件容许的情况下，可以任意时间对HDLC Channel逻辑进行软复位，不会对系统造成影响。

- 通过配置系统控制器SC_HDLC_RESET_REQ[srst_req_hdlc_ahb]为1，可实现对HDLC AHB接口的软复位。
- 通过配置系统控制器SC_HDLC_RESET_REQ[srst_req_hdlc_ch0]为1，可实现对HDLC通道8工作逻辑的单独软复位。
- 通过配置系统控制器SC_HDLC_RESET_REQ[srst_req_hdlc_ch1]为1，可实现对HDLC通道0工作逻辑的单独软复位。

软复位操作仅对接收/发送部分逻辑复位，各寄存器的值均保持不变。

上述软复位寄存器配置完成后，需要延时10 μ s后分别读取软复位状态寄存器（SC_HDLC_RESET_ST），确定软复位解除，如果状态寄存器值错误，则认为存在异常。

14.6.3 操作流程

14.6.3.1 整体工作流程

HDLC的整体工作流程简单概括如下：

步骤1 复位。

根据具体情况使用如下2种复位：

- 硬复位
HDLC模块整体复位，来自上电异步复位。



- 软复位
单通道独立复位，片内CPU配置系统控制器相应复位寄存器实现，同步复位。

步骤2 初始化。

- 顶层初始化：通路仲裁模块功能初始化。
- 通道初始化：需要使用通道的静态配置参数初始化。

步骤3 启动。

每个使用通道的收/发通路，需要CPU通过寄存器配置使能信号和一个启动脉冲，HDLC通道在使能条件下检测到启动脉冲后，开始自动连续工作。

步骤4 工作。

- 通道仲裁功能：通道仲裁、单通路超时保护。
- 每通道功能：数据流管理、BD流管理、实现HDLC协议。

步骤5 完成。

- 如果传输正常完成，等CPU再次有数据发送时或者CPU处理已经接收到的数据后有新的空闲接收BD，返回**步骤3**。
- 如果发生传输异常，CPU识别异常，并记录错误定位信息后，返回**步骤1**。
- 如果CPU不需要HDLC的某通道再传送数据了，就停止该通道。

说明

接收通道，只要有空闲接收BD就可以自动继续工作，无需每次都配置接收启动脉冲。

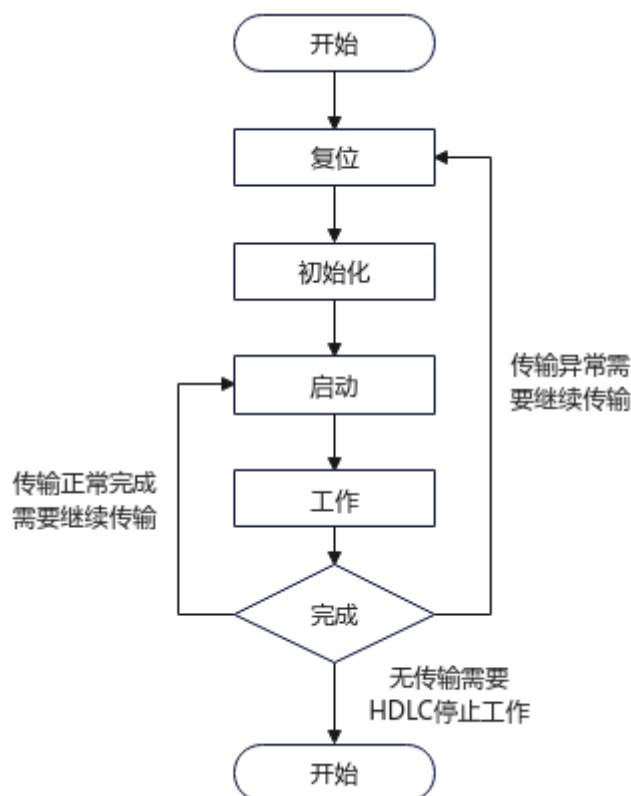
通道软复位，不会改变CPU配置寄存器的值，如果不需要重新配置则可以省略**步骤2**。

----结束

HDLC的整体工作流程如**图14-7**所示。



图 14-7 HDLC 整体工作流程



14.6.3.2 标准复位

HDLC的整体复位流程如下：

- 步骤1** CPU配置命令寄存器，关闭所有接收/发送通道使能。
- 步骤2** 等待一次总线操作最长时间。
- 步骤3** CPU控制异步复位输入信号，下发HDLC整体复位。

----结束

单通道独立复位流程如下：

- 步骤1** CPU配置命令寄存器，无效该通道收/发通路使能。
- 步骤2** CPU记录关键寄存器的值。

📖 说明

主要用于错误定位，根据需要可省略。

- 步骤3** CPU配置命令寄存器的收/发通道复位比特位，进行单通路软复位。

----结束



说明

- 上电复位，一定使用整体复位。
- 单通道传输异常后，使用通道软复位。

14.6.3.3 初始化配置

当通道首次使用或通道未使能时，才可以对HDLC进行初始化，即CPU配置HDLC静态工作参数。具体步骤如下：

步骤1 配置HDLC顶层寄存器。

须知

顶层初始化，必须在各个通道都没有使能的条件下进行；单个通道的初始化，必需保证该通道使能无效。如果在通道使能的条件下，CPU改变顶层或使能通道的静态参数配置，则可能导致HDLC工作异常。因此，禁止在通道使能的条件下，配置顶层和使能通道的静态参数。

- 设置顶层工作模式，具体配置请参见顶层工作模式寄存器。
- 启用有优先级仲裁模式（可选），具体配置请参见优先级配置寄存器。
- 启用单通路处理超时检测功能（可选），具体配置请参见超时上限寄存器。
- 顶层中断掩码，具体配置请参见通路超时中断状态 / 屏蔽 / 清除寄存器。

步骤2 配置单个HDLC通道。

- BD操作相关寄存器，具体配置请参见：
 - 发送BD基地址寄存器。
 - 接收BD基地址寄存器。
 - 接收BD所指缓存大小寄存器。
- 数据流操作相关寄存器，具体配置请参见：
 - 总线操作类型寄存器。
 - 模式寄存器（数据流操作相关控制比特）。
 - 模式寄存器/接口模式比特。
 - 模式寄存器/编码类型比特。
 - 模式寄存器/时钟采样类型比特。
- 协议处理相关寄存器，具体配置请参见：
 - 最长帧长度寄存器。
 - 最短帧长度寄存器。
 - CRC预设值寄存器。
 - CRC比较值寄存器。
 - 模式寄存器/CRC类型比特。
 - 模式寄存器/CRC字节反转比特。
 - 模式寄存器/CRC比特按位反转使能。
 - 模式寄存器/帧间Flag个数比特。



- 模式寄存器/帧间填充数据类型比特。
- 模式寄存器/丢弃小于1Byte碎帧比特。
- 接收地址寄存器。
- 接收地址掩码寄存器。

说明

当HDLC帧的地址段设置为8bit时，对应掩码要配置为0x00ff，且地址寄存器中配置的地址值的高8bit必须为0，否则会识别为地址不匹配帧。

- 通道内中断掩码，具体配置请参见通道内中断状态寄存器。

----结束

14.6.3.4 启动

须知

由于HDLC通道内，接收/发送一致，BD管理是2个BD乒乓工作的（开始预取2个，之后处理一个预取一个）。因此，启动通道时，每张BD表中必需有2个或以上BD。

CPU通过配置每个通道的命令寄存器，可以使收/发通路分别使能/启动/停止/复位，各种命令对HDLC通道的影响，请参见“[发送命令](#)”和“[接收命令](#)”。在整体复位和初始化配置之后，要使HDLC模块工作，CPU需要进行如下操作：

步骤1 准备发送数据、更新发送BD表；分配接收缓存区、更新接收BD表；

步骤2 配置命令寄存器通路使能和启动脉冲。

说明

- 根据CPU底层软件设计者的需要，[步骤1](#)也可以在初始化配置之前进行。
- 由于[步骤2](#)的2种配置同在命令寄存器中，可以同时配置。

----结束

14.6.3.5 交互机制

HDLC与CPU之间通过轮流控制/更新BD信息，来传递收/发数据信息。

14.6.3.5.1 CPU 发送数据交互

CPU主要完成发送数据的管理和发送BD表的维护等工作，具体操作步骤如下：

步骤1 管理发送数据缓冲区，将需要发送的数据放在空闲位置，确定数据长度。

须知

CPU必须进行发送数据长度检查，保证在配置给HDLC的最大和最小帧长范围之内；发送数据长度绝对不能为0，否则HDLC发送通道会在不通知CPU的情况下，停止工作。

步骤2 维护发送BD表，依次更新发送BD信息。



- 更新发送BD信息主要包括发送数据在缓冲区中的首地址和数据长度。
- 比特配置为1，表示CPU配置完成，HDLC可用。

步骤3 配置发送启动脉冲。

步骤4 中断维护/异常检测。

- 如果检测到异常，是中断（例如检测到发送Abort帧/发送超长帧中断）就进入相应的中断服务程序，返回**步骤1**；是非中断类的检测异常，则进入**步骤5**。
- 如果发送正常完成，发送源缓冲完成中断/发送帧完成中断/发送BD没准备好中断，进入相应的中断服务程序，返回**步骤1**。

步骤5 停止HDLC发送通道，保存现场的关键定位信息，复位HDLC发送通路。确定断点，清空发送BD表，将下一个更新指针指向发送BD表基地址，返回**步骤1**。

----结束

14.6.3.5.2 HDLC 发送数据交互

HDLC通道的发送通路，主要完成发送BD流管理/发送数据流管理/HDLC协议组帧等工作，具体操作步骤如下：

步骤1 在通路使能的条件下，如果检测到CPU配置的发送启动脉冲信号，发送通道根据当前发送BD指针读取发送BD，当读取的BD有效时，进入**步骤2**，如果读取的BD无效则进入**步骤7**。

说明

当前发送BD指针，复位后指向发送BD表基地址，发送通道无数据发送时，该指针指向第一个无效发送BD。

步骤2 根据发送BD中的地址和长度信息，读取缓存区中的裸数据。

步骤3 根据同步HDLC协议组帧，从CPU配置的端口发送。

步骤4 当端口完成该帧数据发送后，更新发送BD信息，回写到发送BD表中。

更新发送BD信息包括：

- 发送帧信息，即超长帧/超短帧/发送源数据一帧之内断流。
- 比特配置为0，表示HDLC处理完成，该发送BD可以给CPU使用。

步骤5 当BD回写完成之后，上报发送帧完成中断，通知CPU有HDLC帧发送完成。

须知

- 如果发送超长帧，上报发送超长帧中断，发送通道视为传输异常，停止工作，等待CPU复位。
- 如果发送源数据一帧之内断流，上报发送Abort中断，发送通道视为传输异常，停止工作，等待CPU复位。

步骤6 当前BD指针加8，读取下一个BD。

- 如果读取新的BD为有效BD时，则重复**步骤2** ~ **步骤6**的操作。
- 如果读取的BD为无效BD时，则进入**步骤7**。



说明

当上一次操作的BD指示为BD表中最后一个BD时，下一个读取的BD则为BD表的第一个BD，完成BD表的卷绕操作。发送BD用LB比特位有效，标志发送BD表尾。

步骤7 上报发送BD未准备好中断，发送通道停止读取新的BD，将已经读到的有效BD处理完，则停止发送数据，返回**步骤1**，等待CPU再有数据发送时，重新配置一次发送通道启动脉冲。

----结束

14.6.3.5.3 CPU 接收数据交互

CPU主要完成接收缓冲区的管理和接收BD表的维护等工作，具体操作步骤如下：

步骤1 管理接收数据缓冲区。

- 顺序读取接收BD，检测帧错误类型、缓冲区首地址和接收数据长度。
- 缓冲区首地址和接收数据长度，将已经收到的帧数据取走。
- 建议将正确帧交给上层；将错误帧丢弃，按错误类型进行错误帧计数。
- 按照配置给HDLC的接收缓冲区长度，分配新的空闲接收缓冲区。

步骤2 维护接收BD表，依次更新接收BD信息。

更新接收BD信息主要包括：

- 下一段接收数据缓冲区的首地址。
- 清空HDLC上报的信息，帧错误信息/接收数据帧长。
- 比特配置为1，表示CPU处理完成，HDLC可用。

步骤3 配置接收启动脉冲。

说明

- 接收通道初始化完成，接收数据缓冲区和接收BD准备好，使能接收通道准备接收数据时，需要配置接收启动脉冲。
- 检测到接收BD没准备好中断后，再次接收数据时，必须重新配置接收启动脉冲。
- 每次更新好空闲接收BD，都可以配置一次接收启动脉冲，不影响HDLC工作；但从简化CPU处理过程角度考虑，在接收控制流正常工作时，可省略**步骤3**。

步骤4 中断维护/异常检测。

- 如果检测到接收异常，是中断（例如，检测到接收BD没准备好中断）就进入相应的中断服务程序，返回**步骤1**；是非中断类的检测异常，则进入**步骤5**。
- 如果接收正常完成，即检测到接收目的缓冲完成中断/接收帧完成中断，按类进入中断服务程序，返回**步骤1**。

步骤5 停止HDLC接收通道，保存现场的关键定位信息，复位HDLC接收通路。

确定断点，清空接收BD表，将下一个更新指针指向接收BD表基地址，返回**步骤1**。

----结束

14.6.3.5.4 HDLC 接收数据交互

HDLC通道的接收通路，主要完成接收BD流管理/接收数据流管理/HDLC协议解帧等工作，具体操作步骤如下：



步骤1 在通路使能的条件下，如果检测到CPU配置的接收启动脉冲信号，接收通道根据当前接收BD指针读取接收BD，如果读取的BD无效则进入**步骤6**，当读取的BD有效时，进入**步骤2**。

说明

当前接收BD指针，复位后指向接收BD表基地址，接收通道无数据接收时，该指针指向下一个需要读取的接收BD。

步骤2 从端口接收的HDLC数据流中，检测帧头/帧尾标志，将头/尾标志之间的部分取出，根据同步HDLC协议完成解帧，解帧后的数据根据BD表提供的缓存区地址写入到片外DDR的数据缓存区中。

步骤3 当一帧数据全部写入数据缓存区后，完成一个完整帧的数据接收；将帧信息，顺序更新到下一个有效空闲接收BD，并回写到接收BD表中。

更新接收BD信息包括：

- 接收帧信息，即多种接收帧错误类型/接收帧长/是否发生由于接收目的缓冲不足而产生丢帧。
- 比特置0，表示HDLC处理完成，该接收BD可以给CPU使用。

步骤4 当BD回写完成之后，上报接收帧完成中断，通知CPU有HDLC帧接收完成。

步骤5 当前接收BD指针加8，读取下一个接收BD。

- 如果读取新的BD为有效BD时，则重复**步骤2** ~ **步骤5**的操作。
- 如果读取的BD为无效BD时，则进入**步骤6**。

须知

- 在连续工作时，HDLC上报接收BD未准备好中断，说明CPU的处理速度与HDLC不匹配。此时，HDLC接收通路内部的缓冲区很快就会被填满，HDLC会将后继接收的数据直接丢弃，直至CPU有空闲可用BD，再次配置接收启动脉冲时，才会继续处理缓存帧。
- 遇到这种情况，考虑接收异常之后简化同步操作，建议CPU直接复位HDLC接收通路。

步骤6 上报接收BD未准备好中断，接收通道停止读取新的BD，返回**步骤1**。

----结束

14.6.3.6 完成条件

BD更新后回写完成，才认为一帧处理完成。HDLC的通道会自动按顺序读取BD进行处理，发送通道直至已有有效BD操作全部完成或出错。接收通道始终检测接口是否有HDLC帧，同时按顺序检测接收BD表中是否有空闲可用BD，将解帧数据放入DDR后，完成BD更新回写，即完成一帧接收，直至无HDLC帧接收或无有效空闲BD。

- 发送通道
 - 正常完成条件：检测到无效发送BD。
 - 传输异常条件：CPU检测到异常，下发立即停止命令；发送帧超长/发送源FIFO帧中数据断流，通道自动停止。



- 接收通道
 - 正常完成条件：有空闲接收BD，接口无HDLC帧。
 - 传输异常条件：CPU检测到异常，下发立即停止命令；检测到无效接收BD，通道自动等待。

说明

接收数据帧错误，HDLC上报错误类型，也是正常完成。

14.6.4 发送通道使用说明

14.6.4.1 发送源缓冲区管理

发送的源缓冲区采用BD管理。每个BD指出该缓冲区地址、长度，是否是帧内最后一个缓冲区等信息。下面将发送缓冲区BD的关键控制比特说明如下。为描述方便，引述“表14-2”内容。

表 14-6 发送缓冲区 BD 的关键控制比特说明

信号	bit	信号描述	配置描述
R	[31]	缓冲内容有效标识。 0：无效； 1：有效。	CPU检查内容无效时，可以更改BD，更改完成写入1，写入后在该位为0之前不应更改BD。 HDLC控制器检查内容有效时，在传送完成后更新BD，清零该位。
I	[30]	该BD结束后产生中断。	CPU写入，HDLC只读。1有效。
-	[29]	保留。	CPU写入，HDLC只读（不使用）。
LF	[28]	帧中最后一个BD。	CPU写入，HDLC只读。1有效。
LB	[27]	最后一个BD。	CPU写入，HDLC只读。1有效。
SL	[26]	超长帧。	HDLC写入，CPU只读。1有效。
SS	[25]	超短帧。	HDLC写入，CPU只读。1有效。
-	[24]	保留。	HDLC写入，CPU只读（全0）。
UN	[23]	发送数据没有准备好。	HDLC写入，CPU只读。1有效。
-	[22:16]	保留。	HDLC写入，CPU只读（全0）。
TLEN	[15:0]	发送缓冲长度。	CPU写入，HDLC只读。

发送BD使用方式：BD必须连续存放，配置发送BD基地址寄存器为第一个BD地址。BD中BD尾标志（LB位）为1的表项为最后一个BD。

BD采用预读1个的方式，因此BD表中至少需要2个或以上BD表项。



须知

CPU必须进行发送数据长度检查，保证在配置给HDLC的最大和最小帧长范围之内；发送数据长度绝对不能为0，否则HDLC发送通道会在不通知CPU的情况下，停止工作。

当通道使能后，配置命令寄存器的“开始发送”位，HDLC开始读取BD，并根据BD读取数据缓冲区数据，按照BD中指示发送帧数据。发送完成后将状态更新回BD相应域。

- R

HDLC模块读取BD中的R位：

- R位为0表示该BD无效，发送通道立即上报中断，并停止读取新的BD。
- R位为1表示该BD内容有效，可以发送缓冲区中数据。

由于BD采用预读方式，发生BD无效中断时，HDLC模块可继续发送上一个BD缓冲区内容，并在发送完成后通道停止。CPU在填写缓冲区和BD时，需要检查R位：

- R位为0表示CPU可以填写该BD和缓冲区。
- R位为1表示CPU不能填写该BD和缓冲区。

如果CPU接收到BD无效中断，根据需要填写新的BD，并重新配置命令寄存器的“开始发送”位，使HDLC模块重新读入BD。

如果CPU没有接收到BD无效中断，也可以填充无效的BD，填充完成后配置命令寄存器的“开始发送”位。每次填充完成后总是配置命令寄存器的“开始发送”位，可以保证HDLC模块发送新填充的数据，而不会因为读到无效BD而停止。

说明

由于通道停止时配置“开始发送”位会启动通道读取BD；而通道正常运行时，配置“开始发送”位不会影响通道的正常发送。因此，多次配置命令寄存器的“开始发送”位没有影响。

- I

配置I位的BD，在相应数据缓冲中数据完成发送后产生中断。可以通过适当配置该位来决定发送中断的频率，并在中断程序中进行其他操作，如检查已完成BD状态或填充新BD等。

- LF

配置LF代表该缓冲区内容为1帧中最后一块数据，发送完该缓冲区数据后即发送帧尾标志。BD中没有帧头标志，启动后第一个BD或帧尾BD后第一个BD为帧头BD。需要注意，如果通道停止（如Abort），并在所有数据发送完成后进入IDLE状态（可查询发送状态寄存器），再次启动，此时BD也将作为帧头。

说明

Hi309a的HDLC，不论接收/发送通道，只支持一个BD接收/发送一帧的使用方式，不支持一帧放在多个发送BD中发送或者接收一帧放入多个接收BD中的使用方式。因此，发送BD中的LF比特，CPU在保证一帧一BD的前提下，只需要配置为1即可，即每个BD都是1帧中的最后一块数据。

- LB

LB位为1的BD为最后一个BD。BD应连续存放，并顺序完成，当LB为1的BD完成后，将卷绕到第一个BD继续读取。由于采用预读BD的方式，BD表中至少需要有2个或以上BD表项。

- SL



超长帧标志。当发送源数据（封帧前，数据不加CRC）长度大于超长帧寄存器中的配置时，HDLC模块会停止发送该帧，并发送Abort序列，在数据截止的BD中将SL位配置为1，停止发送通道，上报发送超长帧中断。

- SS

超短帧标志。当发送源数据帧长度小于超短帧寄存器中的配置时，HDLC模块会完成该帧，在BD中将SS位置1，并继续后面BD的发送。

- UN

发送数据未准备好。在发送帧过程中（已经开始发送），如果无法读取后继数据导致无数据可发送，HDLC模块会停止发送该帧，并发送Abort序列，将数据截止BD的UN位配置为1，停止发送通道，上报发送Abort中断。

产生发送数据未准备好是由于总线延迟无法及时读回数据。

- TLEN

发送缓冲区长度。该域最小单位为Byte，但HDLC模块总是按总线配置读取数据（如Burst8，32bits位宽），并丢弃多余的数据。在缓冲区附近有外设时，需要注意该行为。

14.6.4.2 发送中断管理

通道内与中断相关的寄存器有中断状态寄存器、中断清除寄存器、中断屏蔽寄存器。中断发生后需要察看“中断状态寄存器”，判断中断源，进入相应中断程序。

与发送相关的中断有发送Abort、发送源缓冲完成、发送帧完成、发送BD未准备好、发送超长帧。

- 发送Abort

发送通道使用普通外部接口，由于发送源缓冲不足，发生一帧之中数据断流，发送Abort帧，BD写回更新后，产生该中断。

- 发送源缓冲完成

当发送BD缓冲区数据完成发送，且BD的I位为1时，BD写回更新后，产生该中断。

- 发送帧完成

当发送BD缓冲区数据完成发送，且BD的LF位为1时，BD写回更新后，产生该中断。

- 发送BD未准备好

BD采用预读1个的方式，当预读入的BD为未准备好时，立即产生中断，此时上一个BD的内容可以继续发送。

- 发送超长帧

当发送源数据长度超过最长帧长度寄存器的配置值，中止当前发送，发送Abort序列，通道停止。BD写回更新后，产生该中断。

发送通道发送Abort帧的有以下3种原因：

1. 通道正在发送数据时，CPU配置立即停止命令。这种情况属CPU行为，无需上报。
2. 发送通道使用普通外部接口，由于发送源缓冲不足，发生一帧之中数据断流，在断流尾添加Abort序列，即发送Abort帧。BD更新后，上报发送Abort中断。
3. 发送帧超长，被截断为Abort帧。BD更新后，上报发送超长帧中断。



14.6.4.3 发送命令

与通道相关的概念。

- 通道使能
通道发送满足HDLC协议的数据，数据帧或者帧间填充数据。
- 通道发送
通道顺序读取BD，发送数据；通道停止时，接口发送帧间填充数据，停止读取BD，需要重新启动发送，开始读入BD。

📖 说明

帧间填充数据，根据CPU配置的帧间填充物类型，可以是“0x7E”或者“0xFF”。

发送相关命令。

- 停止发送
 - 立即停止发送数据帧，如果正在发送帧数据，停止数据发送，发送Abort序列，通道停止发送数据帧；接口为IDLE状态，发送“0x7E”或“0xFF”。
- 通道使能
 - 通道使能时，接口上数据满足HDLC协议，发送数据帧或IDLE状态数据至外部接口。
 - 通道不使能时，HDLC模块不工作，不发送任何数据。
- 启动发送
 - 通道停止时，以当前BD地址（当前BD寄存器）为帧头，开始发送数据。通道未停止时，没有作用。
- 通道复位

📖 说明

通道软复位的完整的复位流程：先停止发送通道使能，发送通道复位，发送通道使能，发送通道重新启动。禁止在通道使能的条件下复位通道。

通道复位后，停止发送，内部缓存清空，当前BD指针复位到BD基地址。配置寄存器不会复位。

14.6.5 接收通道使用说明

14.6.5.1 接收通道 BD 配置说明

接收的目的缓冲区采用BD管理。每个BD指出环缓冲区地址，长度，是否是帧内最后一个缓冲区等信息。下面将接收缓冲区BD的关键控制比特说明如下。为描述方便，引述“表14-4”内容。



表 14-7 接收缓冲区 BD 的关键控制比特说明

信号	bit	描述	配置描述
R	[31]	缓冲为空标识。 0: 不为空; 1: 为空, 可写。	CPU检查内容无效时, 可以更改BD, 更改完成写入1, 写入后在该位为0之前不应更改BD。 HDLC控制器检查内容有效时, 在传送完成后更新BD, 清零该位。
I	[30]	该BD结束后产生中断。	CPU写入, HDLC只读。1有效。
-	[29]	保留。	HDLC写入, CPU只读。 (不关心01变化)
LF	[28]	帧中最后一个BD。	HDLC写入, CPU只读。1有效。
LB	[27]	最后一个BD。	CPU写入, HDLC只读。1有效。
SL	[26]	超长帧。	HDLC写入, CPU只读。1有效。
SS	[25]	超短帧。	HDLC写入, CPU只读。1有效。
NO	[24]	非字节对齐。	HDLC写入, CPU只读。1有效。
AB	[23]	Abort帧。	HDLC写入, CPU只读。1有效。
RC	[22]	CRC出错(帧中最后一个BD才有效)。	HDLC写入, CPU只读。1有效。
OV	[21]	接收缓存没有准备好。	HDLC写入, CPU只读。1有效。
-	[20:16]	保留。	HDLC写入, CPU只读(全0)。
RLEN	[15:0]	接收缓冲长度。	HDLC写入, CPU只读

接收BD使用方式:

- BD必须连续存放, 配置接收BD基地址寄存器为第一个BD地址。BD中BD尾标志为1的表项为最后一个BD。
- BD采用预读1个的方式, 因此BD表中至少需要2个或以上BD表项。

当通道使能后, 配置命令寄存器的“开始接收”位, HDLC开始读取BD, 并根据BD存放数据缓冲区数据, 接收完成后将状态更新回BD相应域。

- R

HDLC模块读取的BD中R位:

- R位为1表示该BD可用, 可以将数据写入缓冲区中。
- R位为0表示该BD无效, 接收通道立即上报中断, 并停止读取新的BD。

由于BD采用预读方式, 发生BD无效中断时, HDLC模块可继续接收上一个BD缓冲区内容, 并在接收完成后通道停止。

CPU在填写缓冲区和BD时, 需要检查R位:

- R位为1表示CPU不能填写该BD和读取缓冲区。



- R位为0表示CPU可以填写该BD和读取缓冲区。

如果CPU接收到BD无效中断，根据需要填写新的BD，并重新配置命令寄存器的“开始接收”位，使HDLC模块重新读入BD。

如果CPU没有接收到BD无效中断，也可以填充无效的BD，填充完成后配置命令寄存器的“开始接收”位。每次填充完成后总是配置命令寄存器的“开始接收”位，可以保证HDLC模块会将数据写入缓冲区，而不会因为读到无效BD而停止。

说明

由于通道停止时配置“发送启动”位会启动通道读取BD；而通道正常运行时，配置“开始接收”位没有任何作用，因此总是配置命令寄存器的“开始接收”位没有其他副作用。

- I
配置I位的BD，在相应数据缓冲中数据完成接收后产生中断。可以通过适当配置该位来决定接收中断的频率，并在可在中断程序中进行其他操作，如检查已完成BD状态或填充新BD等。
- LF
配置LF代表该缓冲区内容为1帧中最后一块数据，由HDLC模块写入。当检测到一帧结束（包括正常帧，Abort帧，超长帧，超短帧，CRC错帧，字节非对齐帧，缓存不足被截断帧），将BD的LF位置为1。
由于只支持1帧对应一个BD的应用，所以要求CPU在配置接收BD所指缓存大小寄存器时，必须保证配置值大于最长帧长度寄存器的配置，即接收缓冲区长度，必须可以容纳最长帧。
- LB
LB位为1的BD为最后一个BD。BD应连续存放，并顺序完成，当LB为1的BD完成后，将卷绕到第一个BD继续读取。由于采用预读BD的方式，BD表中至少需要有2个或以上BD表项。
- SL
超长帧标志，由HDLC置位。当接收数据（解帧后，数据加CRC）长度大于最长帧长度寄存器的配置+CRC长度时，HDLC模块会停止接收该帧，并丢弃该帧剩余数据，在数据截止的BD中将SL位置1，并继续等待接收下一帧。
- SS
超短帧标志，由HDLC置位。当接收数据帧长度（解帧后，数据加CRC）小于超短帧寄存器中的配置+CRC长度时，HDLC模块会完成该帧，在BD中将SS位置1，并继续接收下一帧。
- NO
非字节对齐帧标志，由HDLC置位。当接收到的数据为非对齐时，更新BD该位为1，并继续接收下一帧。最后一字节中有效数据在低位，高位连续1后1个0均为无效数据。
- AB
接收Abort帧标志，由HDLC置位。当接收到Abort帧时，更新BD该位为1，并继续接收下一帧。
- RC
CRC出错标志，由HDLC置位。当接收到帧尾时，计算CRC值，如果发生错误，更新BD该位为1，并继续接收下一帧。
- OV
接收缓存没有准备好标志，由HDLC置位。当HDLC模块内部接收缓存满，而无法将数据写出，导致新数据无法写入，则丢弃新写入数据。当缓存中数据有机会写



回时，完成该帧，更新BD该位为1，并继续接收下一帧。可能会产生接收丢帧计数器累加。

- RLEN

须知

在设置接收BD所指缓存大小寄存器，即缓冲区长度时需要注意：配置值>最长帧长度；同时，配置值必须为32Bytes的整数倍。

接收缓冲区中数据长度。该域最小单位为Byte，但是HDLC模块总是按总线配置写出数据，如Burst8，32bits位宽，用接收BD中数据长度（RLEN）指示有效数据，多余的写出数据无效。

14.6.5.2 接收中断管理

通道内与中断相关的寄存器有中断状态寄存器，中断清除寄存器，中断屏蔽寄存器。中断发生后需要查看“中断状态寄存器”，判断中断源，进入相应中断程序。与接收相关的中断有：接收目的缓冲完成，接收帧完成，接收BD未准备好。

- 接收目的缓冲完成；
当接收BD缓冲区数据完成接收，且BD的LF位为1时，BD写回更新后，产生该中断。
- 接收帧完成；
当接收BD缓冲区中有帧尾的数据时，BD写回更新后，产生该中断。
- 接收BD未准备好；
BD采用预读1个的方式，当预读入的BD为未准备好时，立即产生中断，此时上一个BD的内容可以继续接收。

14.6.5.3 接收命令

与接收通道相关的概念如下：

- 通道使能；
通道使能时内部缓存可以接收数据；通道不使能时，内部缓存清空，HDLC模块不工作。
- 通道接收；
通道在接收状态下，顺序读取BD，接收数据。通道停止接收时（BD未准备好造成），内部缓存仍可有数据，可以重新启动接收，开始读入BD，才能将数据写出。

接收相关命令如下：

- 停止接收；
立即停止接收数据帧，包括当前内部缓冲中的数据被清空。
通道使能：通道使能后，模块内部可以接收数据，但是需要有BD才能将数据写出；通道不使能时，整个模块不工作。
- 启动接收；
通道开始读取BD，当通道读入的BD未准备好时，会停止继续读取BD，CPU准备好BD后，需要重新启动接收开始读入BD。



- 通道复位

须知

通道软复位的完整的复位流程：先停止接收通道使能，接收通道复位，接收通道使能，接收通道重新启动。禁止在通道使能的条件下复位通道。

通道复位后，停止接收，内部缓存清空，当前接收BD指针复位到BD基地址。配置寄存器不会复位。

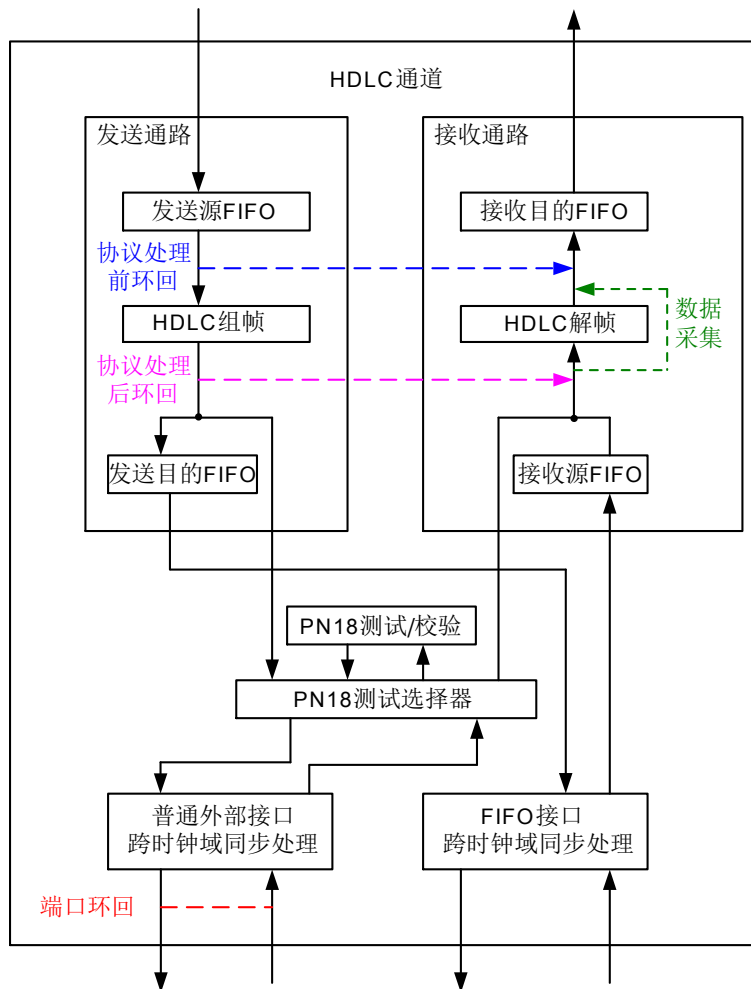
14.6.6 测试功能使用说明

HDLC支持的测试功能如下：

- 协议处理前环回
- 协议处理后环回
- 端口环回
- PN18测试
- 数据采样

各个测试功能在通道内的位置如图14-8所示。

图 14-8 HDLC 测试功能位置说明图



14.6.6.1 协议处理前环回

须知

虽然协议前环回不进行组帧/解帧操作，但是为了接收侧统一处理，发送通路还是会给每个发送BD的数据标志结尾，一起环回给接收通道做帧尾处理。所以，协议前环回有发送帧完成中断上报，也有接收帧完成中断上报。

HDLC提供通道收发数据协议处理前环回测试模式。该模式下，数据不通过协议处理，直接将发送数据通过接收通路回写缓存区。具体操作步骤如下：

- 步骤1** 正常配置HDLC顶层寄存器，正常配置通道各类参数。
- 步骤2** 配置通道的工作模式为协议前环回。
- 步骤3** 发送通路根据BD读取缓存中的帧数据，该数据未经协议处理直接透传给接收通路。



步骤4 接收通路接收透传过来的数据，根据BD将未经协议处理的数据直接回写缓存。

步骤5 CPU完成发送和接收数据比对，查看协议前环回通路是否正常工作。

----结束

14.6.6.2 协议处理后环回

HDLC提供通道收发数据协议处理后环回测试模式。该模式下，发送通路根据发送BD读取缓存数据，并完成HDLC组帧处理，组帧处理后的数据通过环回通道发送给接收通路，接收通路将数据解帧处理后的数据根据接收BD回写缓存。具体操作步骤如下。

步骤1 正常配置HDLC顶层寄存器，正常配置通道各类参数。

步骤2 配置通道的工作模式为协议处理后环回。

步骤3 发送通路完成数据组帧处理，并将协议处理后数据透传给接收通路。

步骤4 该通道将有接收帧完成中断上报，并根据配置的接收BD信息将环回数据回写缓存。

步骤5 CPU完成发送和接收数据比对，查看协议后环回通路是否正常工作。

----结束

须知

在由协议处理后环回切换到协议处理前环回时，需要遵循以下操作流程：

设置协议处理后环回=>通道软复位=>配置通道的工作模式为普通外部接口=>设置协议处理前环回。

14.6.6.3 端口环回

HDLC通道的普通外部接口支持以下2种端口环回：

- 向内环回
向CPU侧环回。
- 向外环回
将外部端口接收数据直接送发送端口。如果HDLC模块通过HDLCBus管脚与对接HDLC接口相连，所以也可以称为向对接HDLC接口侧环回。

向CPU侧的环回可以用来支持HDLC收/发通路功能正常检测、片内PN18测试功能正确性检测。

向对接HDLC接口侧的环回，预留给对接HDLC接口功能扩展时的测试环回。例如：对接HDLC接口也可以发送测试序列，通过HDLC端口环回，测试硬件链路工作是否正常。

例如：利用端口向内环回进行HDLC收/发通路功能正常检测。在向CPU侧环回模式下，发送通路根据发送BD读取缓存数据，并完成HDLC组帧处理，组帧处理后的数据送普通外部接口，在外送端口上环回给接收通路，接收通路类似端口收到数据一样，将数据解帧处理后的数据根据接收BD回写缓存。具体操作步骤如下：

步骤1 正常配置HDLC顶层寄存器，正常配置通道各类参数。



- 步骤2** 配置通道的工作模式为普通外部接口。
- 步骤3** 发送通路完成数据组帧处理后，将协议处理后数据送普通外部接口发送。
- 步骤4** 配置端口环回为向CPU侧环回模式，在端口处将发送数据直接送给接收通道。
- 步骤5** 接收通道将有接收帧完成中断上报，并根据配置的接收BD信息将环回数据回写缓存。
- 步骤6** CPU完成发送和接收数据比对，查看端口环回通路是否正常工作。
- 结束

14.6.6.4 PN18 测试

PN18测试，用来测试Hi309a芯片和对接HDLC接口之间的硬件连接情况是否正常。具体操作步骤如下：

- 步骤1** 通道配置为普通外部接口模式；端口配置为对通工作方式，不要配置其他环回。
- 步骤2** 对接HDLC接口配置针对HDLC接口环回操作。
- 步骤3** 通道配置PN18测试模式使能。

须知

PN18校验启动信号配置时间要稍晚于环回数据到达时间，否则将采样到错误的PN序列种子。

- 步骤4** 通道配置PN18校验启动信号。
- 步骤5** 读取PN18测试相关的状态和上报寄存器，查看PN18测试结果。

----结束

14.6.6.5 单通道数据采样

数据采样功能，是指接收通道不进行HDLC协议处理，直接将端口数据收集下来，送给CPU。当接收到异常帧时，该功能可以用来采样端口异常数据，提供分析定位数据。

单通道数据采样，即单个通道接收通路的数据采样功能。具体操作步骤如下：

- 步骤1** 初始化配置时，将需要通道配置为数据采样模式。
- 步骤2** CPU配置接收BD表时，需要配置I比特有效，即每个BD结束后产生中断。
- 步骤3** 接收通道将端口采样数据，不进行协议处理，直接有效空闲BD写入DDR缓冲区，直至当前BD所指接收缓冲区被填满，上报接收目的缓冲完成中断给CPU。
- 步骤4** CPU检测接收目的缓冲完成中断有效，逐个BD处理接收数据。

----结束

14.6.6.6 实时业务数据采样

实时业务数据采样，是指当前业务通道的数据正常接收，同时CPU可以通过另外一个空闲通道，对当前业务通道的接收端口数据进行直接采样。

以通道8为例，具体操作步骤如下：

步骤1 根据典型配置流程配置通道8。

步骤2 配置通道0的工作模式为数据采样模式。

步骤3 通道8正常接收数据，通道0采样通道8的端口接收数据，CPU检测通道0的接收缓冲完成中断，根据该中断处理采样到的通道8数据。

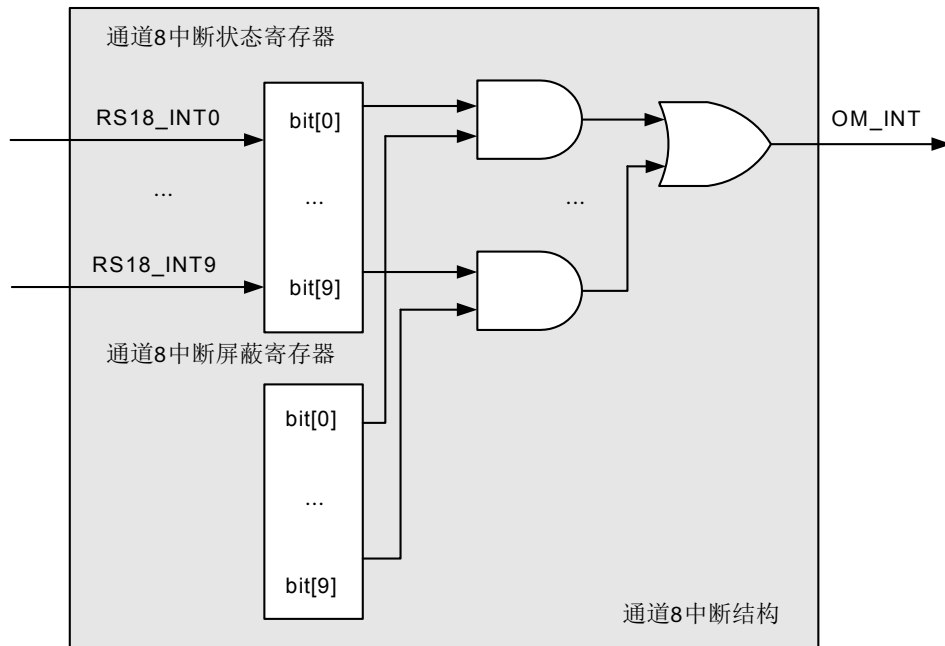
----结束

14.6.7 中断操作维护

14.6.7.1 通道 8 中断处理

通道8中断结构如图14-9所示。通道0与通道8都是例化相同的HDLC通道，只是需要传输的数据不同，各通道内的中断类型和中断的处理方法完全一致。下面以通道8为例进行说明。

图 14-9 HDLC 通道 8 中断树结构示意图



由图14-9可知，HDLC为每个通道都提供了8个类型的中断。这8类中断操作方式如下：

- 发送Abort（没有足够缓存）中断。
- 发送源缓冲完成。
- 接收目的缓冲完成。
- 发送帧完成。
- 接收帧完成。
- 发送BD没有准备好。
- 接收BD没有准备好。

- 发送超长帧。

当CPU检测到某通道产生中断时，需将该通道的中断状态寄存器读出，根据CPU配置的通道内中断屏蔽寄存器的状态，提取CPU关心的中断，按优先级依次进入中断服务程序。

说明

HDLC模块提供中断事件，可由应用决定采用哪些事件来进行帧发送/接收。

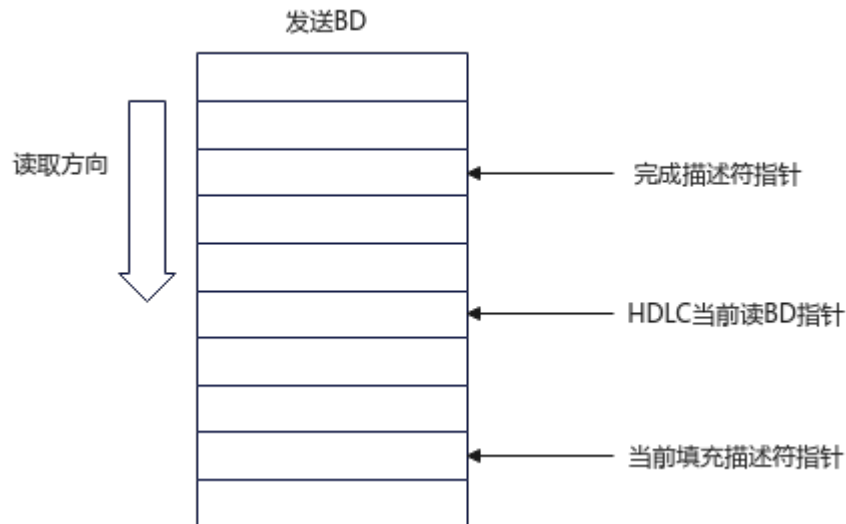
该方案提供参考，具体操作方式可根据应用对事件触发的需要进行修改。

方案中需要软件记录2个指针：

- 当前填充BD指针，用于记录当前CPU填充数据使用的BD地址。
- 完成BD指针，用于记录当前CPU检查已完成BD的地址。

发送通道各指针关系如图14-10所示。

图 14-10 HDLC 发送通道指针关系



完成BD指针记录已检查的BD，当前填充BD指针指向CPU下个填写数据所用的BD，HDLC当前读BD指针是HDLC模块下一个读取的BD。CPU不需要等待HDLC发送完成，只要有BD空间即可填充数据。

14.6.7.2 发送源缓冲完成中断/发送帧完成中断处理建议

发送源缓冲完成中断与发送帧完成中断（具体含义请参见“[发送中断管理](#)”）表示有帧发送完成。请根据CPU需要使用，通常情况下，建议屏蔽。

当CPU检测到发送源缓冲中断或发送帧中断时，建议进行如下处理：

- 步骤1** 软件记录当前完成BD指针，检查当前完成BD的R位，如果值为1，该BD未完成，退出。
- 步骤2** 检查当前完成BD的状态，进行错误处理。

错误处理：发送BD中HDLC会上报3种错误，超长帧/超短帧/发送数据没准备好（即当通道使用普通外部接口时，发送源FIFO一帧之内断流）。各种错误含义如下：



- 超长帧错误上报发送超长帧中断，检测发送超长帧中断进行处理。
- 发送超短帧错误不上报专类中断，需要检测发送源缓冲中断/发送帧中断进行处理。例如进行发送超短帧计数。
- 发送数据没准备好错误会上报发送Abort中断，检测发送Abort中断进行处理。

步骤3 当前完成BD指针加8，指向下一个表项，返回**步骤1**。

----结束

14.6.7.3 发送 BD 未准备好中断处理建议

发送BD未准备好中断（具体含义请参见“[发送中断管理](#)”）表示要发送的BD都已经发送完成，HDLC已经读到无效发送BD，无数据可发，通道停止等待。请根据CPU需要使用，建议CPU关注检测。

当CPU检测到发送BD未准备好中断时，记录发送BD未准备好状态（再有数据需要发送时，必须配置发送启动脉冲），退出。

14.6.7.4 发送 Abort 中断/发送超长帧中断处理建议

发送Abort中断或发送超长帧（具体含义请参见“[发送中断管理](#)”）表示发送通道已经发生异常。请根据CPU需要使用，CPU必须关注检测，在通道内中断类型中，这2个中断的优先级较高，请第一时间处理。

须知

由于发送Stop命令、Abort帧、超长帧等非BD未准备好原因，导致发送通道停止，如果有预读入BD，通道停止后预读入的BD丢失，再次启动时该预读BD被跳过。因此重新继续启动会读入错误停止BD的下一个或下2个BD，建议出错后复位通道，复位BD指针。

当检测到发送Abort中断或者发送超长帧中断时，建议进行如下处理：

步骤1 检查当前完成BD的状态，有错误信息，则进行错误处理；无错误信息，则进入**步骤2**。

错误处理举例：错误帧分类计数。

步骤2 当前完成BD指针加8，指向下一个表项，如果BD数据HDLC有效或当前完成BD地址等于HDLC当前读BD地址，则进入**步骤3**，否则返回**步骤1**。

步骤3 此时错误处理完成，可以初始化通道，重新发送；也可从当前BD地址处继续发送。

----结束

14.6.7.5 接收目的缓冲完成中断/接收帧完成中断处理建议

接收目的缓冲完成中断或接收帧完成中断（具体含义详参见“[接收中断管理](#)”）表示有一个接收数据缓冲区被装满/有一帧接收完成。请根据CPU需要使用，建议CPU关注检测。

当发生接收目的缓冲中断或接收帧中断时，建议进行如下处理：



- 步骤1** 软件记录当前完成BD指针，检查当前完成BD的R位，如果值为1，该BD未完成，退出。
- 步骤2** 检查当前完成BD的状态，进行错误处理，如果接收帧正确，读取缓冲区数据。
错误处理举例：错误帧分类计数，处理错误帧数据（定位时保留/非定位时丢弃）。
- 步骤3** 当前完成BD指针加8，指向下一个表项，返回**步骤1**。
- 结束

14.6.7.6 接收 BD 未准备好中断

接收BD未准备好中断（具体含义请参见“[接收中断管理](#)”）表示已经没有2个或以上空闲有效的接收BD（可能还有一个预读的接收BD尚未或正在使用），即CPU没有及时处理接收缓存中的数据。请根据CPU需要使用，要求CPU必须关注检测。

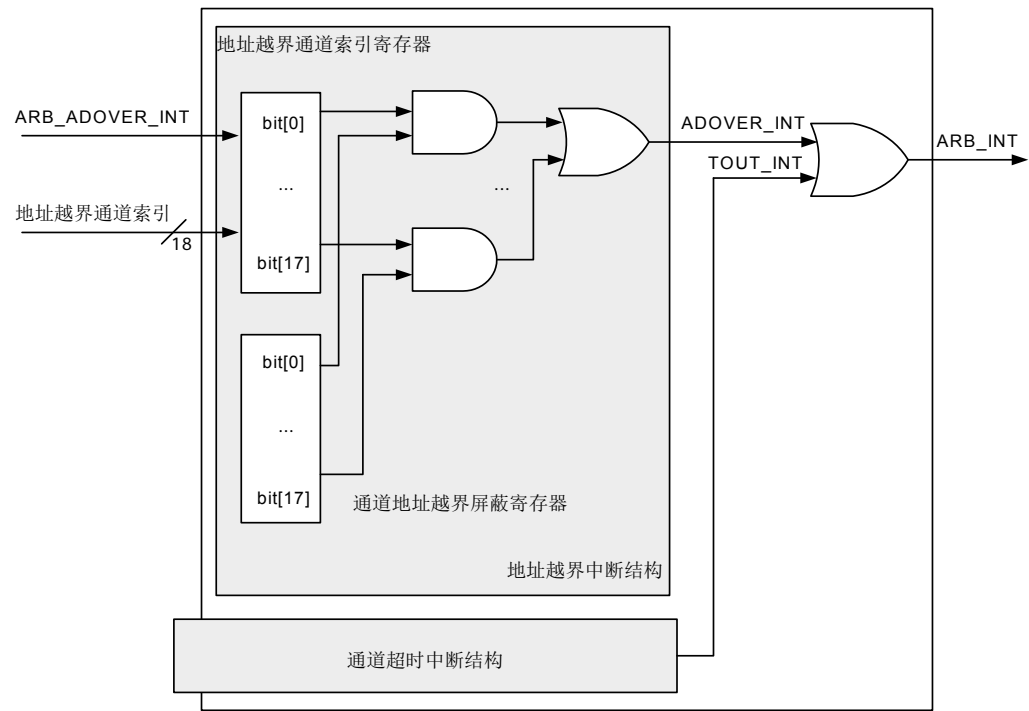
当发生接收BD未准备好中断时，建议进行如下处理：

- 步骤1** 停止HDLC接收通道使能。
如果需要，记录关键上报寄存器，复位HDLC接收通路。
- 步骤2** 及时将已经接收到的HDLC的数据，自动按序全部处理完成。
- 步骤3** 清空接收BD表，将当前BD指针指向接收BD表基地址。
- 结束

14.6.7.7 通道仲裁中断处理

通道仲裁中断树结构如[图14-11](#)所示。

图 14-11 HDLC 通道仲裁中断树结构





通道仲裁包括单通路处理超时中断上报，中断上报采用一级中断树的结构。当发送 ARB_INT 中断时，需要查询单通路处理超时中断状态寄存器。

14.6.7.8 通道超时中断

通路超时中断的含义：参与通道仲裁的通路中，当前占用master的通路处理时间过长，已经超过CPU配置的门限值。HDLC模块支持单通路仲裁超时检测防止因为单个通路处理异常而无响应，影响其他通路的正常工作，导致整个HDLC无响应的情况发生。请根据CPU需要使用，建议CPU启用超时检测，并优先响应。

当检测到通路超时中断时，建议进行如下处理：

- 确认超时通道。
- 如果检测到少数通路发生处理超时，只需复位相应通道即可。
- 如果是多个通路产生超时（例如：已经超过半数），CPU检测到中断后，建议先确认外部总线是否正常，再进行HDLC整体复位。

通道超时中断不提供超时类型定位寄存器。因此，不管任何异常情况引起的通道超时，都必须重新复位启动发生超时的通道。

14.7 HDLCBUS 模式

14.7.1 概述

HDLCBUS模式主要用于多个HDLC之间互相通信的场景。当每个HDLC都配置为HDLCBUS模式时，可以使用开漏输出将多个节点通过片外总线连接在一起。HDLCBUS模式只支持半双工操作，同一时刻只允许总线上有一个HDLC发送数据，而可以有多个HDLC接收数据。

14.7.2 特性

HDLC工作于BUS模式具有以下特性。

14.7.2.1 整体特性

- HDLC通道0和通道8在正常工作情况下（非环回测试模式），可以配置成普通外部接口（点对点工作模式），也可以配置成HDLCBUS模式。
- 在一种特定的应用场景下，只能选择使用普通外部接口或者HDLCBUS模式，并且作为静态参数在初始化时配置完成，两种模式无法进行动态切换。
- HDLCBUS模式下CBUS接口只有3个有意义的信号，分别是CLK_CBUS、RXD、TXD。

说明

HDLCBUS模式不使用RTS_N信号。

- CLK由芯片外部或者BUS上的任一芯片统一提供，所有连接在一个外部BUS上的HDLCcontroller接口共用同一个时钟。
- HDLCBUS模式只能支持半双工通信。
- 正常情况下，外部连接需要保证组成LAN的任何一个HDLCcontroller发送逻辑0，BUS上的实际数据都是逻辑0，而不论是否有其他HDLCcontroller同时发送逻辑1。



- HDLCBUS模式只支持同步HDLC协议，不支持异步HDLC协议。

14.7.2.2 配置特性

- 使用HDLCBUS模式时，帧间填充物类型必须配置为0xFF，而非0x7E；在点对点模式下，帧间填充物类型保持0xFF和0x7E可配。
- 使用HDLCBUS模式时，帧间FLAG个数必须配置为0；在点对点模式下，帧间FLAG个数保持0~15可配。
- 使用HDLCBUS模式时，编码类型必须配置为NRZ；点对点模式保持NRZ和NRZI编码类型可配。
- 使用HDLCBUS模式时，总线操作类型寄存器的发送源和接收目的都必须配置为地址自增、数据位宽32bit和burst8。
- HDLCBUS模式，采样延迟周期由0~7可配改为0~9可配，以降低保持时间的最低要求。
- HDLCBUS模式，发送延迟周期0~8可配，以配合采样时间的配置达到发送落后采样两个内部主工作时钟周期的目的。
- HDLCBUS模式，是否和MPC8360对接可以通过外部寄存器配置，如果LAN中存在MPC8360，那这个配置寄存器必须配置为有效。

说明

如果配置为不和MPC8360连接，发送延迟必须配置为比采样延迟寄存器小2的值；如果配置为和MPC8360连接，发送延迟必须配置为8。

14.7.2.3 功能特性

- 上次未成功发送整帧或者从未发送帧的HDLC检测到外部总线n个（此值可以通过寄存器配置）连续的逻辑1，即认为总线空闲可以开始发送数据。如果检测过程中遇到一个逻辑0，则检测计数器清0。
- 发送过程中如果TXD为1，而RXD为0，则判定为总线发生冲突。
- 发送过程中如果TXD为0，而RXD为1，则认为链路发生异常。

说明

- HDLC在遇到链路异常回退，若是发生时间为接口上正在发送净荷的前x比特（ $x \leq \text{CRC长度}$ ），则碎帧计数器有可能会有统计值。
- 当遭遇链路异常时，HDLC监测总线上连续1个数的计数器bus_idle_cnt会清零，因此该HDLC在下一轮对总线使用权的竞争中，一定会败于同等优先级的那些HDLC，也可能败于低优先级的HDLC，这取决于清除bus_idle_cnt时bus_idle_cnt的值。
- 如果帧长大于32Byte，冲突和链路异常检测窗口为发送每帧的前32Byte；如果帧长小于等于32Byte，冲突和链路异常检测窗口为整个净荷部分。
- 在冲突检测窗口内发生冲突或者链路异常，HDLC会准备重传此帧；如果实际发生冲突或链路异常在检测窗口之外，HDLC会一直将此帧发送完成，而不论是否正常接收。
- 上次完成一帧发送的HDLC需要等待总线连续的m个（ $m > n$ ，m值可以通过寄存器配置）逻辑1才能重新启动一次发送。如果检测过程中遇到一个逻辑0，则检测计数器清0。
- HDLC分别统计总线冲突和链路异常引起的重传次数。
- 无论是普通外部接口模式还是HDLCBUS模式支持上升沿发送下降沿接收、下降沿发送上升沿接收、上升沿收发和下降沿收发。

- HDLCBUS模式下，采样延迟周期由0~7可配改为0~9可配，以降低保持时间的最低要求。
- HDLCBUS模式下，发送延迟周期0~8可配，以配合采样时间的配置达到发送落后采样两个内部主工作时钟周期的目的。
- 连接在HDLCBUS上的HDLController不允许连续发帧，也不能两帧之间共享一个FLAG，一次获取总线只能发送一帧。
- HDLCBUS模式，芯片复位期间，TXD发送高电平。

14.7.3 接口信号

HDLCBUS模式接口信号同点对点模式。

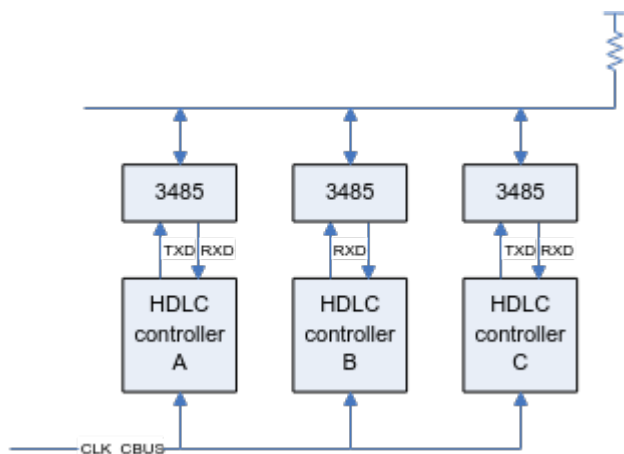
影响HDLC工作于BUS模式时的时序的因素主要有三个：收发模式、收发延迟、对接器件。此三个因素均为寄存器可以配置。

1. 收发模式：HDLC可以配置为上升沿收发、下降沿收发、上升沿发下降沿收、下降沿发上升沿收四种收发模式，收发模式不同接口时序不同。
2. 收发延迟：HDLC的接收和发送的延迟周期（以HDLC内部主时钟周期为单位）也可以通过外部寄存器配置，收发模式不同接口时序不同。

14.7.4 功能描述

14.7.4.1 功能概述

图 14-12 HDLCBUS 模式应用框图



如图14-12，HDLCBUS是一种连接多个HDLC的片外总线的实体又表示HDLC控制工作于总线模式。实际应用中HDLC需要通过3485总线连接到HDLCBUS上。连接在BUS上的每个HDLC无间断的监听总线，并根据一定的规则获取总线、检测冲突和回退重传。

14.7.4.2 工作规则

获取总线。获取总线的前提是总线空闲（判断当前时刻没有HDLC正在发送数据）。HDLC通过RXD监测总线数据。对于当前有帧待发且处于高优先级的HDLC，如果检测到n（可以配置）个连续的逻辑1，则可以认为总线空闲。对于处于低优先级且有帧待



发的HDLC，如果检测到m（m>n，可以配置）个连续的逻辑1，则可以认为总线空闲。判断到总线空闲即可以获取总线并立即启动一次帧发送。对于当前有帧待发的HDLC，如果没有检测到连续的n或者m个逻辑1而突然检测到逻辑0，那么cnt清零，重新开始检测总线。

冲突检测。在发送帧的前32Bytes，HDLC持续监测总线数据。如果出现TXD为1而RXD为0的情况，则判定为发生冲突。如果出现TXD为0而CTS_N为1的情况，则判定为链路异常。如果没有发生以上两种情况则可以继续发送直至此帧结束，并降低优先级返回获取总线状态。具体检测规则如表14-8。

表 14-8 冲突分析规则

TXD	RXD (CTS_N)	冲突状态
1	1	没有冲突，继续发送。
0	0	没有冲突，继续发送。
1	0	发生冲突，停止发送，准备重传。
0	1	链路异常，停止发送，准备重传。

回退重传。如果发生冲突或者链路异常，则停止发生并以高优先级回退到获取总线的状态，准备重传此帧。HDLCcontroller会分别对发生冲突和链路异常引起的帧重传次数进行统计。

14.7.5 寄存器配置说明

HDLC的寄存器配置完全参考Hi309a nManager的HDLC_CFG和HDLC_IF_SEL部分，分别配置HDLC通道和HDLC顶层。

HDLCBUS模式下HDLC的寄存器配置可以参考工作方式一节的描述，其中HDLCBUS模式下需要特殊配置的寄存器和Debug的寄存器在下面特别说明。

14.7.5.1 配置寄存器

HDLCBUS模式需要特别说明的配置寄存器全部在nManager的HDLC_CFG中。

表 14-9 模式寄存器（MODE）偏移地址：0x138

bit	名称	类型	描述	复位值
[31:28]	nflag	RW	帧间flag个数。 （初始时配置，范围从0到15可配。BUS模式配置为0。）	0x0



bit	名称	类型	描述	复位值
[27:25]	ifm	RW	接口模式。 000：普通外部接口； 010：BusMode外部接口； 011：协议处理后环回模式； 101：协议处理前环回模式； 其他：保留。 （初始时配置，BUS模式配置为010。）	0x0
[24]	flgt	RW	帧间填充物类型。 0:0x7E； 1:0xFF。 （初始时配置，根据使用约定，BUS模式OM通道半双工模式，配1；其他，均配0。）	0x0
[19]	codt	RW	编码类型。 0：NRZI编码，该码字中，用比特翻转代表1，不翻转代表0； 1：NRZ编码，该码字中，用高电平代表1，低电平代表0； （初始时配置，固定使用NRZ编码。） BUS模式配置为1。	0x1



bit	名称	类型	描述	复位值
[18]	clkt	RW	需要和模式寄存器的 clkt_low组合 成 {clkt,clkt_low} 配合表示时钟 采样类型。 00: 上升沿发 送，下降沿采 样 01: 上升沿发 送，上升沿采 样 10: 下降沿发 送，上升沿采 样 11: 下降沿发 送，下降沿采 样 (初始时配 置，点对点模 式固定配置为 0b00，BUS模 式固定配置为 2'b01。)	0x0



bit	名称	类型	描述	复位值
[17:15]	txdly	RW	<p>在HDLCBUS模式下，需要和模式寄存器的txdly_high组合成{txdly_high, txdly}配合表示发送延迟周期。</p> <p>可配范围[0,8]，延迟单位同主工作时钟周期。</p> <p>（注意：配0延时10个HDL主工作时钟周期，配8延时2个HDL主工作时钟周期，其他有效值以此类推，配置非法值9~15等同于配置为8。</p> <p>HDLCBUS模式，如果配置cf_8360_on_lan有效，需要配置为0b1000，如果配置cf_8360_on_lan无效，需要保证此寄存器配置比采样延迟周期寄存器小2。点对点模式配置该寄存器对发送时刻无影响。总之，采样延迟周期的实际有效配置值—发送延迟周期的实际有效配置值>=2。）</p>	0x4



bit	名称	类型	描述	复位值
[14:12]	rxdly	RW	<p>需要和模式寄存器的 rxdly_high 组合成 {rxdly_high, rxdly} 配合表示 RX 采样延迟周期。</p> <p>可配范围 [0,9]，延迟单位同主工作时钟周期。</p> <p>（注意：配 0 延时 10 个 HDLC 主工作时钟周期，配 9 延时 1 个 HDLC 主工作时钟周期，其他有效值依次类推，配置非法值 10~15 等同于配置为 9。无论 HDLCBUS 模式还是点对点模式，如果配置 cf_8360_on_lane 有效，则鉴沿后立即采样与采样延迟寄存器配置无关。配置 cf_8360_on_lane 无效时，HDLCBUS 模式默认配置为 0b1001，点对点模式默认配置为 0b0000。总之，采样延迟周期的实际有效配置值—发送延迟周期的实际有效配置值 >= 2。）</p>	0x0



bit	名称	类型	描述	复位值
[3]	clkt_low	RW	需要和模式寄存器的clkt组合成{clkt,clkt_low}配合表示时钟采样类型。 00: 上升沿发送，下降沿采样 01: 上升沿发送，上升沿采样 10: 下降沿发送，上升沿采样 11: 下降沿发送，下降沿采样 (初始时配置，点对点模式固定配置为0b00，BUS模式固定配置为0b01。)	0x0



bit	名称	类型	描述	复位值
[2]	rxdly_high	RW	<p>需要和模式寄存器的 rxdly_high 组合成 {rxdly_high, rxdly} 配合表示 RX 采样延迟周期。</p> <p>可配范围 [0,9]，延迟单位同主工作时钟周期。</p> <p>（注意：配 0 延时 10 个 HDLC 主工作时钟周期，配 9 延时 1 个 HDLC 主工作时钟周期，其他有效值依次类推，配置非法值 10~15 等同于配置为 9。无论 HDLCBUS 模式还是点对点模式，如果配置 cf_8360_on_lane 有效，则鉴沿后立即采样与采样延迟寄存器配置无关。配置 cf_8360_on_lane 无效时，HDLCBUS 模式默认配置为 0b1001，点对点模式默认配置为 0b0000。总之，采样延迟周期的实际有效配置值—发送延迟周期的实际有效配置值 >= 2。）</p>	0x0



bit	名称	类型	描述	复位值
[1]	nottxd	RW	在HDLCBUS模式下，配置该寄存器可以决定TXD是否取反后发送。 0：TXD不取反直接发送； 1：TXD取反后发送。 配置为0，禁止配置为1。	0x0



bit	名称	类型	描述	复位值
[0]	txdly_high	RW	<p>在HDLCBUS模式下，需要和模式寄存器的txdly_high组合成{txdly_high, txdly}配合表示发送延迟周期</p> <p>可配范围[0,8]，延迟单位同主工作时钟周期。</p> <p>（注意：配0延时10个HDL主工作时钟周期，配8延时2个HDL主工作时钟周期，其他有效值以此类推，配置非法值9~15等同于配置为8。HDLCBUS模式，如果配置cf_8360_on_lan有效，需要配置为0b1000，如果配置cf_8360_on_lan无效，需要保证此寄存器配置比采样延迟周期寄存器小2。点对点模式配置该寄存器对发送时刻无影响。总之，采样延迟周期的实际有效配置值—发送延迟周期的实际有效配置值>=2。）</p>	0x0



表 14-10 总线操作类型寄存器（BUSOPMD）偏移地址：0x13C

bit	名称	类型	描述	复位值
[31]	tsai	RW	发送源地址自增。 0：地址固定； 1：地址自增。 （初始时配置，固定配为地址自增。 BUS模式固定配置为1。）	0x0
[30:29]	tsbw	RW	发送源地址总线位宽。 00：8bit； 01：16bit； 10：32bit； 11：无效。 （初始时配置，典型配置32比特。 BUS模式固定配置为10。）	0x0
[28]	tsbt	RW	发送源burst配置。 0：single； 1：burst8。 （初始时配置，典型配置burst8。BUS 模式固定配置为1。）	0x0
[7]	rdai	RW	接收目的地址自增。 0：地址固定； 1：地址自增。 （初始时配置，固定配为地址自增。 BUS模式固定配置为1。）	0x0
[6:5]	rdbw	RW	接收目的地址总线位宽。 00：8bit； 01：16bit； 10：32bit； 11：无效。 （初始时配置，典型配置32比特。 BUS模式固定配置为10。）	0x0
[4]	rdbt	RW	接收目的burst配置。 0：single； 1：burst8。 （初始时配置，典型配置burst8。BUS 模式固定配置为1。）	0x0



表 14-11 获取总线优先级寄存器（CTS_PRI）偏移地址：0x1100

bit	名称	类型	描述	复位值
[31:24]	cf_cts_high_pri	RW	高优先级的HDLCcontroller获取总线需要检测的连续逻辑1的个数 配置为00010111。	0x17
[23:16]	cf_cts_low_pri	RW	低优先级的HDLCcontroller获取总线需要检测的连续逻辑1的个数 配置为00011001。	0x19
[15]	cf_8360_on_lan	RW	在HDLCBUS模式，配置该寄存器表示对接器件中是否有MPC8360。 0：不需要和MPC8360对接； 1：需要和MPC8360对接。 注意，无论BUS模式还是点对点模式，该寄存器配置为1时，HDLC内部鉴沿即接收，此时采样的时刻与采样延迟周期寄存器配置无关。	0x0

14.7.5.2 Debug 寄存器

为HDLCBUS模式特别增加的Debug寄存器全部在nManager的HDLC_CFG中。

表 14-12 冲突重传计数器（COL_RETRAN_CNT）偏移地址：0x1104

bit	名称	类型	描述	复位值
[31:0]	col_retran_cnt	RW	发生冲突引起重传的次数统计	0x00000000

表 14-13 链路异常重传计数器（ABN_RETRAN_CNT）偏移地址：0x1108

bit	名称	类型	描述	复位值
[31:0]	abn_retran_cnt	RW	链路异常引起重传的次数统计	0x00000000



表 14-14 地址不匹配帧计数器（ADDUN_CNT）偏移地址：0x110C

bit	名称	类型	描述	复位值
[31:0]	addun_cnt	RW	接收到的地址不匹配帧统计计数器。接收到帧的目的地址字段时和本地地址比较，如果不匹配则立刻进行地址不匹配统计，因此无论是否完整的帧只要目的地址不是本地地址，都会统计。	0x00000000

表 14-15 单帧发送时间计数器（ONEFR_CNT）偏移地址：0x1110

bit	名称	类型	描述	复位值
[31:0]	onefr_cnt	RO	从帧头移入移位寄存器到帧发送完成持续时间。 单位同主工作时钟。	0x00000000

表 14-16 同步物理接口信息寄存器（SYN_PHY_INFO）偏移地址：0x1114

bit	名称	类型	描述	复位值
[31]	bus_idle_vali d	RO	总线空闲指示。 0：总线繁忙； 1：总线空闲。	0x0
[30:23]	bus_idle_cnt	RO	记录到总线连续1的个数。	0x00
[22]	col_detect	RO	检测到冲突。 0：正常； 1：冲突。	0x0
[21]	abn_detect	RO	检测到链路异常。 0：链路正常； 1：链路异常。	0x0
[20:19]	tx_bd_in_src _fifo_cnt	RO	发送源FIFO中存在的BD数。	0x0
[18:15]	tx_src_frame _cnt	RO	发送源FIFO中的帧数。	0x0
[14:10]	bus_tx_src_fi fo_cnt	RO	发送源FIFO中当前有多少行数据未发送。	0x00



bit	名称	类型	描述	复位值
[9]	pre_frame_end	RO	帧尾是否在发送源移位寄存器中。 0：帧尾不在发送移位寄存器； 1：帧尾在发送移位寄存器。	0x1
[8]	fifo_nd_fill	RO	是否启动一次burst8写数据进发送源FIFO。 0：不启动burst8的请求； 1：启动一次burst8的请求。	0x1
[7:0]	Reserved	RO	保留。	0x00

表 14-17 发送源 FIFO 信息寄存器（TX_SRC_FIFO_INFO）偏移地址：0x1118

bit	名称	类型	描述	复位值
[31]	rec_valid	RO	记录读指针有效。	0x0
[30]	rec_clr_wdw	RO	检测时间窗过期清除记录读指针有效。 0：没有因检测窗口过期无效记录读指针； 1：因检测窗口过期无效记录读指针。	0x0
[29]	rec_clr_end	RO	帧尾发送完毕清除记录读指针有效。 0：没有因发送帧完成无效记录读指针； 1：因发送帧完成无效记录读指针。	0x0
[28:25]	rec_rd_ptr	RO	记录读指针。	0x0
[24:20]	cover_cnt	RO	发送源FIFO中不能被覆盖的行数。	0x00
[19:16]	send_cnt	RO	记录读指针有效期间已经发送的行数。	0x0
[15:11]	tx_src_sh_cnt	RO	移位寄存器中数据个数。	0x00
[10]	tx_src_reg_empty	RO	移位寄存器空信号。 0：移位寄存器不空； 1：移位寄存器空。	0x1
[9]	retran_for_col	RO	冲突重传一次。 0：没有因冲突发生重传； 1：冲突重传。	0x0



bit	名称	类型	描述	复位值
[8]	retran_for_a bn	RO	链路异常重传一次。 0：没有因链路异常发生重传； 1：链路异常重传。	0x0
[7:0]	Reserved	RO	保留。	0x00

说明

在HDLCBUS模式下，full信号（偏移地址为0x01E0+0x0100*H，第30bit）不具有指示发送源FIFO空满的意义，另有cover_cnt信号（偏移地址为0x1118+0x0100*H，第24:20bit）指示发送源FIFO中有多少行被占用。



15 GPIO

15.1 关键术语

表 15-1 GPIO 关键术语描述

术语	描述
GPIO	General Purpose Input/Output，通用输入输出接口。当微控制器或芯片组没有足够的I/O端口，或当系统需要采用远端串行通信或控制时，GPIO产品能够提供额外的控制和监视功能。

15.2 概述

GPIO（General Purpose Input/Output）为通用输入输出接口。Hi309a共例化10组GPIO接口：GPIO0~9。每组分别提供最大32个可编程的输入输出管脚（注：网络部分不出管脚，请以SI requirement表为准）。具体如下。

控制器编号	区域	管脚数量	管脚编号
GPIO0~9	PERI	304	GPIO001~304

每个管脚可以配置为输入或者输出。这些管脚用于生成或采样特定应用的输出或输入信号。作为输入管脚时，可作为中断源；作为输出管脚时，每个GPIO都可以独立地清0或置1。

须知

Hi309a（网络）：GPIO247~GPIO255管脚输入方向功能（含GPIO中断）不可用。



15.3 特性

GPIO接口具有以下特点：

- 提供10组GPIO接口，每组GPIO接口有独立的地址和中断线。
- 支持串行数据总线和低于32bit的并行数据总线的模拟。
- 每根信号有独立的数据寄存器和输入输出控制寄存器，默认为输入模式。
- 支持中断查询和中断屏蔽功能。GPIO_INTEN中断使能寄存器，GPIO_INMASK中断屏蔽寄存器，GPIO_INTSTATUS屏蔽后状态寄存器，GPIO_RAW_INTSTATUS原始中断状态寄存器，GPIO_PORTA_EOI中断清除寄存器。
- GPIO给每个GPIO端口提供一个独立的中断，每组GPIO提供32个独立的中断和一个汇聚的组合中断，独立中断支持可配置电平和脉冲方式上报。
- GPIO_INTTYPE_LEVEL（中断类型寄存器）、GPIO_INT_POLARITY（中断触发极性控制寄存器）两个寄存器共同决定了中断源的特性和触发类型。
- 当GPIO作为输出管脚时，每个GPIO管脚支持独立输出使能或关闭。
- 当GPIO作为输入管脚时，GPIO支持滤毛刺功能，可滤除宽度小于2.56μs的脉冲，该功能默认关闭，可通过寄存器GPIO_DEBOUNCE开启和关闭滤毛刺功能。
- 当GPIO作为输入管脚时，可作为中断源，每个GPIO管脚都具有独立的中断触发方式控制；中断支持电平触发和边沿触发两种方式，每个中断触发方式可独立配置为高电平、低电平、上升沿和下降沿触发。
- 部分PAD.GPIO管脚支持上电采样功能，采样结果存储在芯片寄存器中，具体请参见《SI requirement.xls》。
- GPIO支持两种工作模式，即自研GPIO工作模式和SYN工作模式，两种工作模式寄存器偏移不同，两种工作模式通过系统拉控制信号进行静态切换。
- SYN工作模式下，GPIO控制寄存器支持多核并发，提高软件效率。
- 自研、Synopsys和无线多核并发三种工作模式下，GPIO控制寄存器都支持双边沿触发中断。
- GPIO控制寄存器支持三套地址访问。

注：Hi309a只使用自研模式，下文不再赘述。

15.4 遵循协议

GPIO 与总线的交互均遵循下面协议：

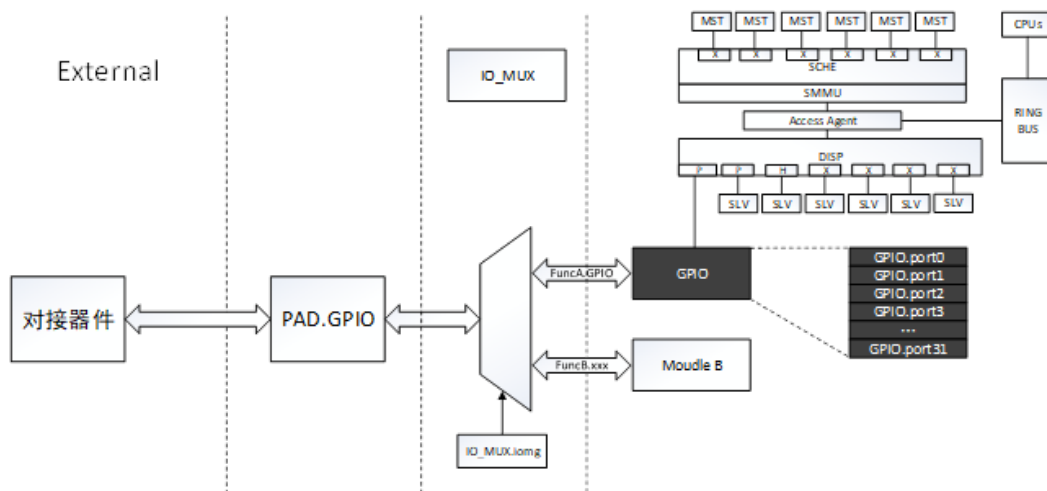
AMBA 3 APB Protocol v1.0（ARM IHI 024B）

15.5 功能描述

15.5.1 应用场景

GPIO应用框图如图15-1所示。

图 15-1 GPIO 应用框图



当芯片没有足够的I/O端口，或当系统需要采用远端串行通信或控制时，CPU可配置GPIO提供额外的控制和监视功能。

须知

GPIO应用时，需要根据GPIO复用模式，配置相应的IO_MUX寄存器，保证GPIO工作在正确的功能模式，下面章节描述的功能为上图中深色框图中GPIO模块的功能。

15.5.2 相关模块

在系统中，GPIO模块会和下面多个模块进行交互，分别描述如下：

- **CPU**: CPU通过总线访问GPIO的寄存器接口，操作属性必须为Device或Strongly-Ordered。CPU首先需要配置GPIO的输入或输出特性，如果是输出，则给出输出的数据，如果是输入，则读取输入数据。
- **GIC**: GPIO可配置为中断信号，在对外信号发送电平或脉冲变化时，将触发GPIO中断产生中断信号给GIC模块，通过GIC将中断发送给对GPIO具有控制权的CPU。
- **IO_MUX**: 芯片管脚PAD.GPIO通过IO_MUX做管脚复用控制后，使得一个PAD管脚对应多个功能端口，GPIO和其他功能端口复用一個芯片的PAD管脚。

15.5.3 功能原理

每组GPIO提供32个可编程的输入输出管脚。每个管脚可以配置为输入或者输出。这些管脚用于生成或采样特定应用的输出或输入信号。

作为输入管脚时，可作为中断源；作为输出管脚时，每个GPIO都可以独立地清0或置1。GPIO可以根据电平或跳变值产生可屏蔽的中断。GPIOINTR信号给中断控制器一个指示，表示有中断发生。用户可以配置中断为电平触发、单沿触发或双沿触发。

作为输入管脚时，GPIO还支持滤毛刺功能，可滤除脉冲宽度小于 $2.56\mu\text{s}$ 的毛刺。



15.6 工作流程

15.6.1 初始化

说明

GPIOx表示GPIO0、GPIO1和GPIO2。

上电复位时，所有管脚默认为输入。

复位信号有效时，GPIO有如下反应：

- 所有的寄存器被清零。
- 所有管脚都被配置为输入。
- 原始中断都被清零。
- 清除寄存器GPIOx_INTEN中相应的比特，使中断无效。
- 中断自动切换为电平触发的中断。

15.6.2 管脚复用配置

GPIO大部分均与其他管脚复用，使用GPIO前，需要配置IO_MUX使能相应管脚的GPIO功能。GPIO0、GPIO1和GPIO2的管脚复用方式请参见IO_MUX的寄存器配置说明。

15.6.3 软复位

对接器件容许的情况下，可以任意时间对GPIO工作逻辑进行软复位，不会对系统造成影响。

- 通过配置PERI_SUBCTRL中的 SC_GPIO_RESET_REQ相应位为0b1，可实现对GPIO0、GPIO1、GPIO2工作逻辑的单独软复位。
- 通过配置PERI_SUBCTRL中的SC_GPIO_RESET_DREQ的相应位为0b1，可实现对GPIO0、GPIO1、GPIO2工作逻辑的单独软复位撤离。

15.6.4 时钟门控

上电时GPIO接口的时钟默认为打开状态。GPIO工作完成后，可以关闭GPIO时钟，以降低芯片功耗。每组GPIO使用不同的门控信号，即它们的时钟可独立打开或关闭。

须知

上电初始化时，若配置软复位寄存器和门控寄存器解除软复位和门控，需要等待对应的软复位和门控状态寄存器状态完成才能下一步动作，否则认为存在异常。（等待时间10μs）

15.6.4.1 打开时钟

GPIO0、GPIO1、GPIO2的具体配置步骤如下：



- 步骤1** 读PERI_SUBCTRL系统控制器寄存器SC_GPIO_CLK_ST，获取当前时钟门控信息。
- 步骤2** 若SC_GPIO_CLK_ST [0]为0，则进入**步骤5**；SC_GPIO_CLK_ST [0]为1，则表示GPIO0时钟已经打开。
- 步骤3** 若SC_GPIO_CLK_ST [1]为0，则进入**步骤6**；SC_GPIO_CLK_ST [1]为1，则表示GPIO1时钟已经打开。
- 步骤4** 若SC_GPIO_CLK_ST [2]为0，则进入**步骤7**；SC_GPIO_CLK_ST [2]为1，则表示GPIO2时钟已经打开。
- 步骤5** 配置SC_GPIO_CLK_EN[0]为1，打开GPIO0模块的时钟门控。
- 步骤6** 配置SC_GPIO_CLK_EN[1]为1，打开GPIO1模块的时钟门控。
- 步骤7** 配置SC_GPIO_CLK_EN[2]为1，打开GPIO2模块的时钟门控。
- 结束

15.6.4.2 关闭时钟

GPIO0、GPIO1、GPIO2的具体配置步骤如下：

- 步骤1** 读PERI_SUBCTRL系统控制器寄存器SC_GPIO_CLK_ST，获取当前时钟门控信息。
- 步骤2** 若SC_GPIO_CLK_ST [0]为1，则进入**打开时钟的步骤5**；SC_GPIO_CLK_ST [0]为0，则表示GPIO0时钟已经关闭。
- 步骤3** 若SC_GPIO_CLK_ST [1]为1，则进入**打开时钟的步骤6**；SC_GPIO_CLK_ST [1]为0，则表示GPIO1时钟已经关闭。
- 步骤4** 若SC_GPIO_CLK_ST [2]为1，则进入**打开时钟的步骤7**；SC_GPIO_CLK_ST [2]为0，则表示GPIO2时钟已经关闭。
- 步骤5** 配置SC_GPIO_CLK_EN[0]为0，关闭GPIO0模块的时钟门控。
- 步骤6** 配置SC_GPIO_CLK_EN[1]为0，关闭GPIO1模块的时钟门控。
- 步骤7** 配置SC_GPIO_CLK_EN[2]为0，关闭GPIO2模块的时钟门控。
- 结束

15.6.5 模式选择

GPIO0、GPIO1、GPIO2的具体配置步骤如下：

- 步骤1** 写PERI_SUBCTRL系统控制器寄存器SC_GPIO_SYN_EN，配置GPIO模式。
- 步骤2** 若SC_GPIO_SYN_EN为0，则GPIO工作在自研模式下，支持自研寄存器地址访问。
- 步骤3** 若SC_GPIO_SYN_EN为1，则GPIO工作在SYN模式下，支持syn寄存器 或 多核并发寄存器地址访问。
- 结束

15.6.6 输入输出配置

说明

GPIO_x表示GPIO0、GPIO1、和GPIO2。



GPIO的输入输出方向和输出数据均由软件控制，管脚的输入信号也通过软件读取GPIOx_EXT_PORT[31:0]来获得。（自研模式读GPIO_SWPORT_DR）

输入输出模式配置步骤如下：

步骤1 选择要使用的GPIO管脚，配置IO_MUX的相应管脚复用控制寄存器，选择GPIO功能。

步骤2 配置寄存器GPIOx_SWPORT_DDR，配置GPIO是输入还是输出。

----结束

说明

- 当GPIO用作输入时，外部信号通过GPIO管脚送进来，可通过寄存器GPIOx_EXT_PORT（自研模式读GPIO_SWPORT_DR）读取输入信号值。
- 当GPIO用作输出时，通过写寄存器GPIOx_SWPORT_DR，GPIO会将写进此寄存器的值通过GPIO输出。需要注意的是，此时若GPIO中断处于使能状态，则当输出信号满足触发条件时会产生中断。

15.6.7 中断处理

15.6.7.1 自研模式或 syn 模式（syn 寄存器）

说明

GPIOx表示GPIO0、GPIO1和GPIO2。

须知

- 每个GPIO有一个中断控制寄存器。当有一个或多个GPIO管脚产生了中断时，GPIO汇聚成一个中断或者单独送到中断控制器。对于边沿触发的中断，软件必须清除该中断以使能其他的中断；对于电平触发的中断，外部中断源应该保持该电平直到处理器识别到该中断。
- 设置GPIO中断为电平触发时，如果配置该中断为异步中断（配置寄存器GPIO_LS_SYNC为0），则该中断仍然可以在GPIO时钟关闭时传递中断到GIC，如果配置该中断为同步中断（配置寄存器GPIO_LS_SYNC为1），则该中断在GPIO时钟关闭时无法传递到GIC。

GPIO中断处理的配置步骤如下：

步骤1 选择要使用的GPIO管脚，配置该管脚为输入（配置方法请参见“[输入输出配置](#)”）。

步骤2 保证GPIO数据线在这些操作的过程中保持稳定。

步骤3 配置GPIOx_DEBOUNCE为，使能毛刺滤除功能，否则跳过此步。

步骤4 配置GPIOx_INTTYPE_LEVEL对应位，选择边沿触发或电平触发。（自研模式下，可配置GPIOx_INT_DEEDGE为双边沿触发模式）

步骤5 配置GPIO_INT_POLARITY对应位，选择触发的有效电平或有效沿，通过往寄存器GPIOx_PORTA_EOI对应位写，清中断。

步骤6 配置GPIOx_INTMASK，关闭中断屏蔽。



步骤7 配置GPIOx_INTEN，使能中断。

步骤8 发生中断时，查询GPIOx_INTSTATUS，获取中断信息。

步骤9 中断处理完成。往寄存器GPIOx_PORTA_EOI对应位写，清除中断。

----结束

15.6.7.2 syn 模式（多核并发寄存器）

说明

GPIOx表示GPIO0、GPIO1和GPIO2。

须知

- 每个GPIO有一个中断控制寄存器。当有一个或多个GPIO管脚产生了中断时，GPIO汇聚成一个中断或者单独送到中断控制器。对于边沿触发的中断，软件必须清除该中断以使能其他的中断；对于电平触发的中断，外部中断源应该保持该电平直到处理器识别到该中断。
- 设置GPIO中断为电平触发时，如果配置该中断为异步中断（配置寄存器GPIO_LS_SYNC为0），则该中断仍然可以在GPIO时钟关闭时传递中断到GIC，如果配置该中断为同步中断（配置寄存器GPIO_LS_SYNC为1），则该中断在GPIO时钟关闭时无法传递到GIC。

GPIO中断处理的配置步骤如下：

步骤1 选择要使用的GPIO管脚，配置该管脚为输入（配置方法请参见“[输入输出配置](#)”）。

步骤2 保证GPIO数据线在这些操作的过程中保持稳定。

步骤3 配置GPIOx_DEBOUNCE_SET为，使能毛刺滤除功能，否则跳过此步。

步骤4 配置GPIOx_INTTYPE_LEVEL_SET或GPIOx_INTTYPE_LEVEL_CLR对应位，选择边沿触发或电平触发。

步骤5 配置GPIO_INT_POLARITY_SET或GPIO_INT_POLARITY_CLR对应位，选择触发的有效电平或有效沿，通过往寄存器GPIOx_PORTA_EOI对应位写，清中断。

步骤6 配置GPIOx_INTMASK_CLR，关闭中断屏蔽。

步骤7 配置GPIOx_INTEN_SET，使能中断。

步骤8 发生中断时，查询GPIOx_INTSTATUS，获取中断信息。

步骤9 中断处理完成。往寄存器GPIOx_PORTA_EOI对应位写1，清除中断。

----结束

15.6.8 GPIO 毛刺过滤

15.6.8.1 GPIO 消抖动

消抖动说明



- GPIO消抖动可以过滤掉小于 $2.56\mu\text{s}$ 的脉冲毛刺。
- 对于脉冲宽度在 $2.56\mu\text{s}\sim 5.12\mu\text{s}$ 之间的脉冲是否被过滤决定于输入与消抖动时钟间的相位关系。
- 对于脉冲宽度大于 $5.12\mu\text{s}$ 的输入脉冲，则可以稳定的被采样输入。

GPIO输入消抖动配置步骤如下：

步骤1 选择要使用的GPIO管脚，并配置为输入（配置方法请参见“[输入输出配置](#)”）。

步骤2 配置GPIOx_DEBOUNCE为1，使能毛刺滤除功能。

----结束

说明

GPIO_x表示GPIO0、GPIO1和GPIO2。



16_{QDMA}

16.1 IP 功能及特性描述

QDMAC（Queue Direct Memory Access Controller），基于Queue结构的直接内存访问控制器，通过处理器简单的控制即可独立完成系统内部存储器之间的数据搬移，有效的减轻了处理器的负担。HiQDMAV120具有如下特性：

- 基于Queue的任务管理方式：软件可编程为1、2、4、8、16个Queue，每个Queue具有一套独立的软件管理入口，包括任务配置、任务查询、Queue暂停、任务取消以及中断状态；每个Queue可配置的任务个数为512/Queue。
- 基于普通BD（Buffer Descriptor）的任务描述方式：普通BD描述了QDMAC需要执行任务的详细参数，结构见[图16-4](#)，普通BD置放于QDMAC外部的存储器中，软件配置普通BD所存储的地址（下文中统称普通BD指针）到QDMAC的Queue中，QDMAC读取存储器中的普通BD解析后执行数据搬移。
- 支持多维搬移：支持1维、2维、3维搬移，用于完成多块规则数据的搬移，具体参数见[表16-4](#)、[表16-5](#)、[表16-6](#)、[表16-7](#)，实例见[实例——普通BD一维M2M搬移](#)、[实例——普通BD一维F2M搬移](#)、[实例——普通BD一维F2F搬移](#)、[实例——普通BD三维M2M A-sync搬移](#)、[实例——普通BD三维F2M AB-sync搬移](#)。
- 支持BD链接：当前BD携带下一个BD指针信息，当前BD任务完成以后，可以直接从存储器取下一个BD执行，而不需要从Queue获取下一个BD指针，类似于链表操作，具体参数见[表16-8](#)，实例见[实例——普通BD链接任务](#)。
- 支持定浮点转换任务：QDMAC可以完成对搬移的数据进行定点数与浮点数之间的转换，具体参数见[表16-9](#)，实例见[实例——普通BD定浮点转换任务](#)。
- 支持压缩解压缩：QDMAC可以完成对搬移的数据进行压缩或者解压缩转换，具体参数见[表16-9](#)，实例见[实例——普通BD定浮点转换任务](#)。
- 支持清固定值任务：QDMAC可以直接向目的地址写固定的值，用于完成存储器的初始化，固定的值由软件配置，具体参数见[表16-1](#)、[表16-5](#)，实例见[实例——普通BD清固定值任务](#)。
- 支持任务完成写回：任务数据搬移完成并且没有错误，QDMAC可以向当前BD指针指向的地址写一个完成标志供软件查询，具体参数见[表16-1](#)，同一个Queue中的任务写回顺序严格按照任务的配置顺序进行，即同一个Queue的写回保序。
- 支持任务完成后发送消息：任务数据搬移完成并且没有错误，QDMAC可以向软件指定的地址发送一条消息，消息长度软件可编程，具体参数见[表16-11](#)、[表](#)

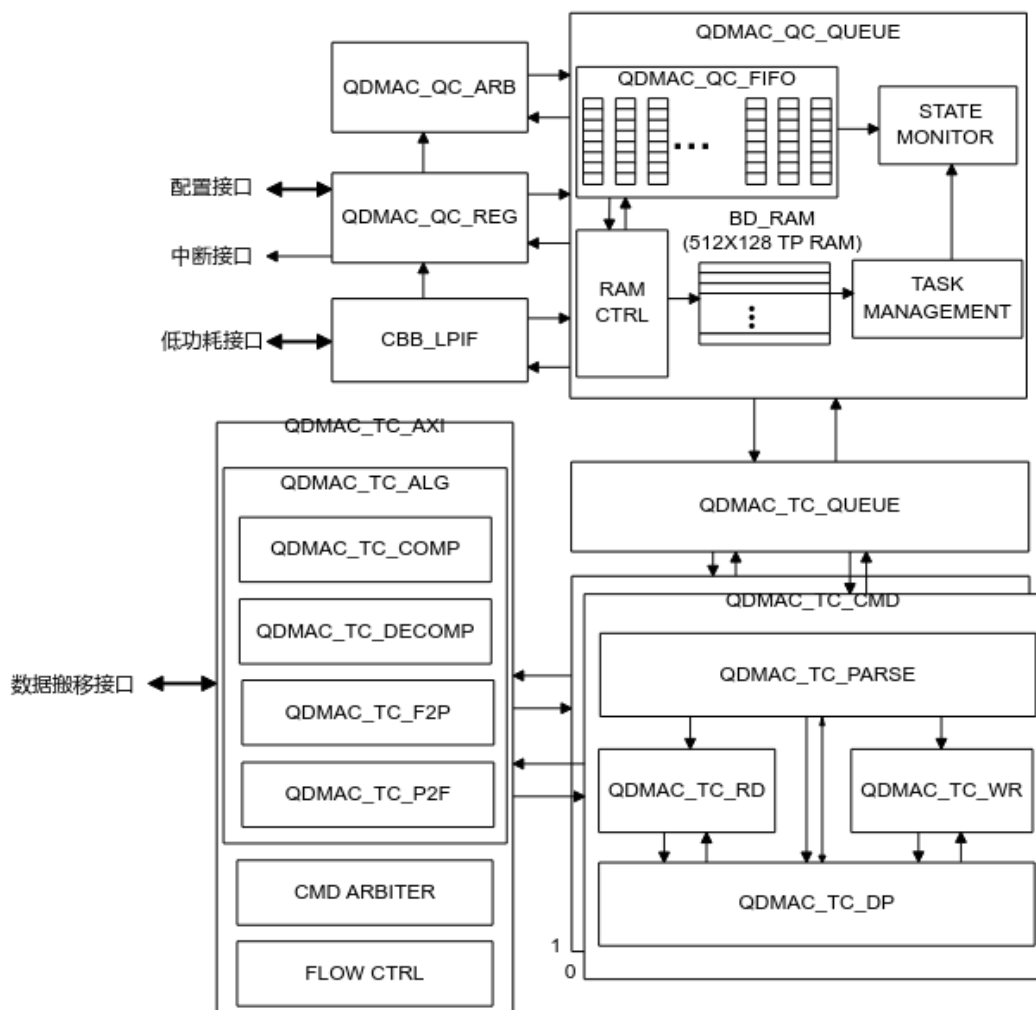


16-12、表16-13、表16-14、表16-15；同一个Queue中的任务消息发送顺序严格按照任务的配置顺序进行，即同一个Queue的消息发送保序。

- 支持任务完成后发送中断：任务数据搬移完成并且没有错误，写回和消息发送完成，QDMAC可以触发任务完成中断，并且记录完成BD的指针供软件查询，具体参数见表16-1；同一个Queue中的任务完成中断触发顺序严格按照任务的配置顺序进行，即同一个Queue的完成中断触发保序。
- 支持任务失败错误中断：当任务配置失败，任务参数配置错误，任务数据搬移错误，BD读取错误，写回错误，消息发送错误会触发任务失败中断，具体描述见中断描述。
- 支持低功耗功能：软件配置LPC（Low Power Controller）发起低功耗请求，QDMAC会自动阻塞软件继续配置任务，当QDMAC内部所有已经配置的任务执行完成并且所有产生的中断都已经被清除以后进入低功耗状态，系统可以对QDMAC进行任何低功耗操作。
- 支持处理能力隔离：软件按照业务需求将DMA的Queue分成多个通道组，多个通道组共享DMA的Master带宽，但是各个通道组对应的业务处理能力需求各异，为了保证各个通道组之间的带宽比例，软件对各个通道组进行了带宽限制，在多个通道组竞争的时候，如果参与竞争的通道组已经达到了带宽限制条件，则对应通道组的任务不能被调度。
- 支持读命令发送间隔：软件配置QDMAC_GLB_EN寄存器的rdctrl_en域配置读命令发送间隔使能，控制读命令发送间隔的目的在于减少系统被访问汇聚点的累积命令，从而减少高优先级Master命令在汇聚点的响应时间。
- 支持寄存器多模隔离：QDMAC支持对可写寄存器进行VMID保护，当进行寄存器访问所携带的VMID为配置的有效的VMID时，才能操作成功，否则，即违反VMID检查，上报pslverr，同时上报VMID不匹配中断；VMID=0时表示超级权限，不受任何保护；寄存器是否进行VMID检查可通过寄存器配置；QDMAC模块的可写寄存器可分为COMM类、通道组类、通道类、正常中断类参数，不同类别的参数根据寄存器配置VMID值进行保护检查。
- 不支持短BD。

QDMAC模块内部结构框图如图16-1所示。

图 16-1 QDMAC 模块内部结构框图



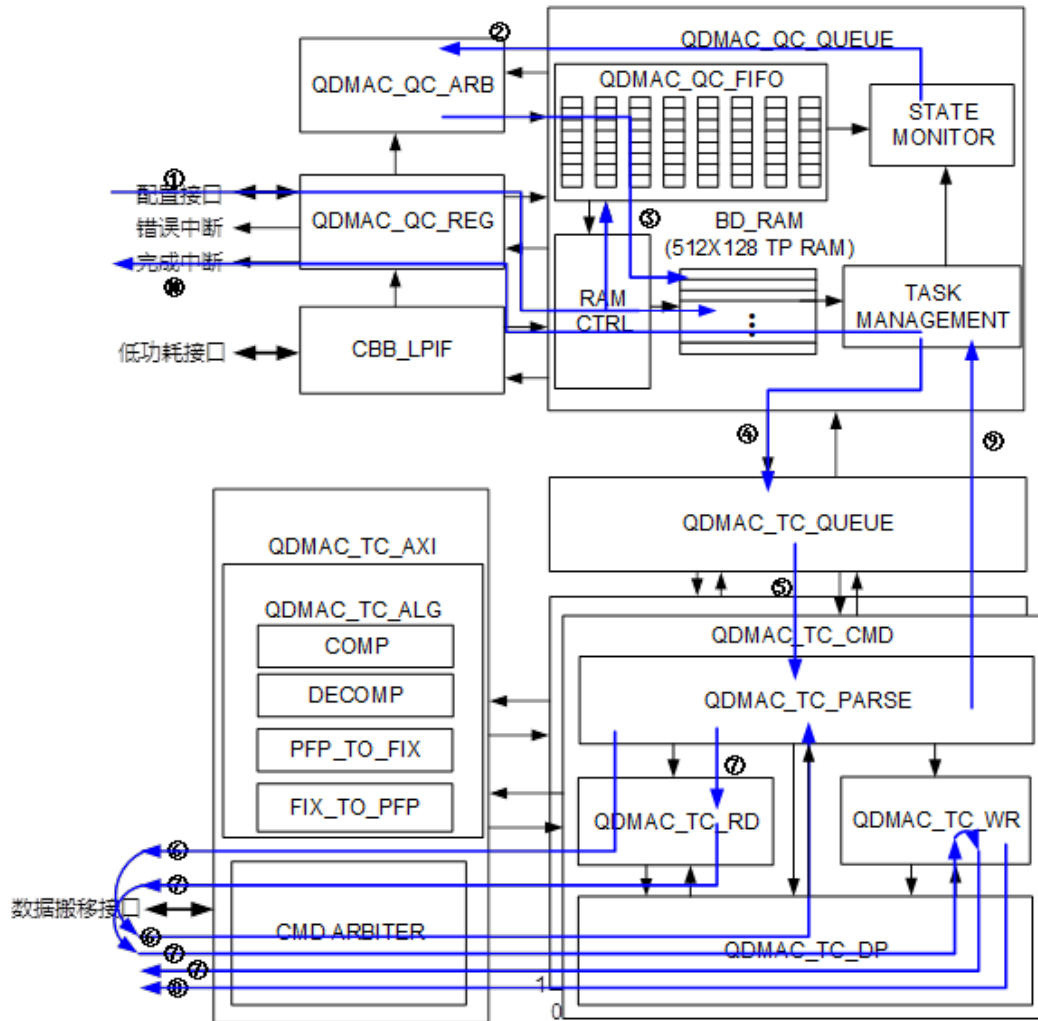
QDMAC由图16-1所示的模块和逻辑构成，主要包括：

- QDMAC_QC_REG：寄存器模块，实现QDMAC的软件配置和查询功能。
- QDMAC_QC_ARB：仲裁模块，实现Queue的请求仲裁，当软件下发到任务到Queue以后，Queue会发出传输申请，多个Queue的传输申请由该模块仲裁出一个获胜的Queue进行传输。
- QDMAC_QC_QUEUE：Queue的管理和任务收发模块，实现Queue的管理和维护，软件下发任务的管理，传输请求的产生以及获得仲裁任务的分发，同时维护Queue的状态供软件查询和低功耗控制。
- QDMAC_QC_FIFO：用于维护各个Queue的读写指针。
- BD_RAM：用于存放软件下发的任务。
- RAM CTRL：用于根据各个Queue的读写指针产生读写BD_RAM的控制信号。
- STATE MONITOR：监控Queue的状态。
- TASK MANAGEMENT：任务的分发和回收管理，监控任务的执行状态。
- QDMAC_TC_QUEUE：QDMAC两个物理通道的任务分发管理。
- QDMAC_TC_CMD0~1：QDMAC的物理通道，一共两个，每个物理通道都可以独立完成一个任务的参数解析，数据读取，数据缓存，数据写出功能。

- QDMAC_TC_PARSE：参数解析模块，用于完成BD参数的解析和读写状态机的启动。
- QDMAC_TC_RD：数据读取命令处理模块，用于完成数据读取命令的拆分。
- QDMAC_TC_WR：数据写出命令处理模块，用于完成数据写出命令的拆分。
- QDMAC_TC_DP：数据缓存模块，用于完成数据读取以后的缓存，达到一定数据量以后触发写出操作。
- QDMAC_TC_AXI：Master接口模块，同时实现两个物理通道的传输请求仲裁以及定浮点转换。
- CMD ARBITER：两个物理通道的请求仲裁逻辑。
- QDMAC_TC_ALG：算法处理模块，用于控制数据读取以后的算法处理。
- QDMAC_TC_F2P：实现浮点数转定点数。
- QDMAC_TC_P2F：实现定点数转浮点数。
- QDMAC_TC_COMP：实现压缩处理。
- QDMAC_TC_DECOMP：实现解压缩处理。

正常任务执行数据流如图16-2所示。

图 16-2 QDMAC 工作数据流图

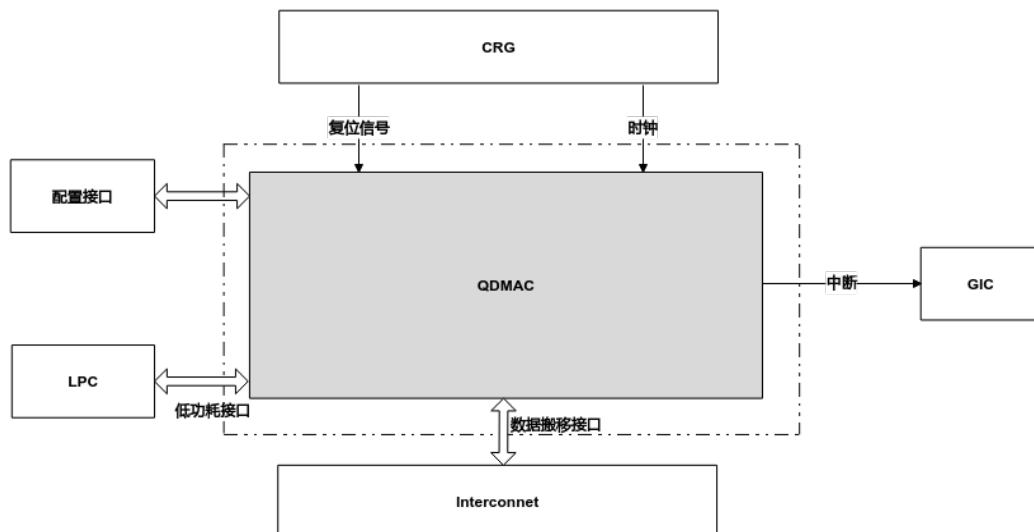


- 步骤1 软件配置任务到QDMAC内部的Queue，QDMAC会更新对应Queue的状态以及将BD指针存入内部RAM。
 - 步骤2 根据Queue的状态向Arbiter申请任务执行请求。
 - 步骤3 Arbiter根据Queue设定的优先级，仲裁出一个优先级最高的Queue。
 - 步骤4 读取获得仲裁的BD指针，并将其分发到传输控制模块。
 - 步骤5 为任务分配空闲的硬件资源（QDMAC_TC_CMD）。
 - 步骤6 读取BD具体参数。
 - 步骤7 根据解析的结果，将指定源地址数据搬移到目的地址。
 - 步骤8 数据搬移完成以后，QDMAC发起写回和消息。
 - 步骤9 完成内部任务执行状态更新。
 - 步骤10 触发任务完成中断。
- 结束

16.2 应用范围

QDMAC与芯片中各模块之间的连接关系如图16-3所示。

图 16-3 QDMAC 应用框图



CRG负责产生QDMAC所需要的时钟和复位；

配置接口用于软件配置QDMAC，任务下发以及状态查询；

LPC负责控制QDMAC的低功耗接口；

Interconnect用于连接QDMAC和各个存储器，QDMAC所有的搬移命令和数据都需要经过Interconnect；

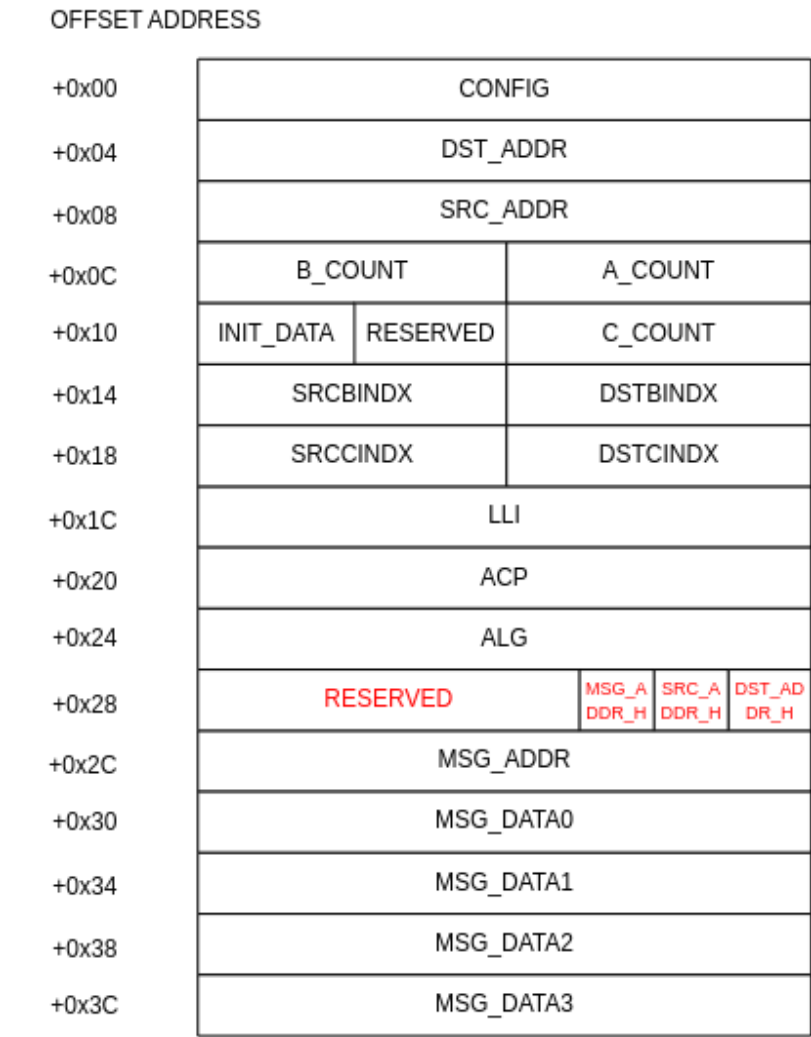
GIC负责将QDMAC产生的中断路由给各个处理器。



16.3 协议或包格式说明

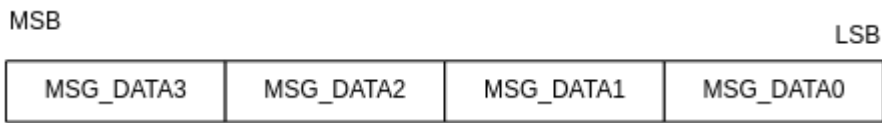
QDMAC的普通BD结构如图16-4所示。

图 16-4 QDMAC 的普通 BD 结构图



QDMAC的消息结构如图16-5所示。

图 16-5 QDMAC 的消息结构图



QDMAC的普通BD各个域说明：



表 16-1 普通 BD CONFIG 域说明

CONFIG		
Bits	Name	Description
[31]	wb_flag	<p>写回完成状态指示。</p> <p>0: 任务未执行；</p> <p>1: 任务已执行（当write_back_en使能后，DMA对该BD已经进行过处理，即使发生了BD参数解析告警、BD数据搬移告警、BD写消息告警该bit也会置1（如果BD读取告警、BD写回告警、Queue溢出告警，该bit不会置1）。</p> <p>注意：软件在配置BD时必须将该bit置为0，否则不能起到完成指示的作用。如果软件配置了write_back_en，该bit配置为1会触发配置告警中断。</p>
[30]	wb_info	<p>写回信息指示，DMA自动填写，对于软件为只读属性。</p> <p>0: 任务执行成功（数据搬移成功，即使发生了BD写消息告警、BD写回告警）；</p> <p>1: 任务执行失败（包括BD参数解析告警、BD数据搬移告警；不包括BD读取告警、BD写消息告警、BD写回告警、Queue溢出告警）。</p>
[29:28]	reserved	保留
[27]	bd_type	<p>BD类型指示，DMA自动填写，对于软件为只读属性。</p> <p>0: 普通BD；</p> <p>1: 短BD；</p>
[26]	fixed_init_en	<p>清固定值使能。</p> <p>0: 不使能清固定值，BD为正常的数据搬移任务；</p> <p>1: 使能清固定值，BD为只写目的端的清固定值任务。</p> <p>注意：清固定值不支持定点转浮点、浮点转定点、压缩、解压缩功能，如果配置上述四种功能使能有效会触发配置告警中断。</p>
[25:24]	msg_num	<p>QDMAC发送的消息word数目定义。</p> <p>00: BD搬移完成后将msg_data0、msg_data1、msg_data2、msg_data3作为消息发送；</p> <p>01: BD搬移完成后将msg_data0、msg_data1作为消息发送；</p> <p>10: BD搬移完成后将msg_data0作为消息发送；</p> <p>11: 保留。</p> <p>注意：如果msg_en有效，配置为11时会触发配置告警中断，同时，如果msg_num为10，则发生告警以后强制不发送消息。</p>



CONFIG		
[23]	si	源地址递增设置。 0：源地址不递增； 1：源地址递增。
[22]	di	目的地址递增设置。 0：目的地址不递增； 1：目的地址递增。
[21]	smode	源地址的递增模式。 0：A-sync模式，三维的地址偏移是一个frame的最后一个array的首地址到下一个frame的第一个array的首地址间隔。 1：AB_sync模式，三维的地址偏移是一个frame的第一个array的首地址到下一个frame的第一个array的首地址的间隔。
[20]	dmode	目的地址的递增模式。 0：A-sync模式，三维的地址偏移是一个frame的最后一个array的首地址到下一个frame的第一个array的首地址间隔。 1：AB_sync模式，三维的地址偏移是一个frame的第一个array的首地址到下一个frame的第一个array的首地址的间隔。
[19:16]]	sl	源传输burst长度。 0001：burst length为2； 0011：burst length为4； 0111：burst length为8； 1111：burst length为16； 其余字段保留。 注意：SD6801里面burst length固定为8，配置该域无效。
[15:12]]	dl	目的传输burst长度。 0001：burst length为2； 0011：burst length为4； 0111：burst length为8； 1111：burst length为16； 其余字段保留。 注意：SD6801里面burst length固定为8，配置该域无效。
[11]	reserved	保留。



CONFIG		
[10:8]	sw	源数据宽度。 000: burst size为8bit; 001: burst size为16bit; 010: burst size为32bit; 011: burst size为64bit; 100: burst size为128bit; 101: burst size为256bit; 110、111: reserved; 注意: SD6801里面burst size固定与总线位宽一致, 配置该域无效。
[7]	reserved	保留。
[6:4]	dw	目的数据宽度。 000: burst size为8bit; 001: burst size为16bit; 010: burst size为32bit; 011: burst size为64bit; 100: burst size为128bit; 101: burst size为256bit; 110、111: reserved; 注意: SD6801里面burst size固定与总线位宽一致, 配置该域无效。
[3]	chain_en	BD链接使能: 0: 链接使能无效, 该BD执行完毕以后不会链接到下一个BD; 1: 链接使能有效, 该BD执行完毕以后会链接到下一个BD继续执行。
[2]	intr_tc_en	搬移完成中断使能, 指示当BD搬移完成是否需要产生中断。 0: BD执行完毕以后不产生完成中断; 1: BD执行完毕以后产生完成中断。
[1]	write_back_en	写回BD使能, 当BD执行完成以后, 指示是否需要修改BD表中的完成指示。 0: BD执行完毕以后不会修改BD表中的完成指示; 1: BD执行完毕以后向BD表的CONFIG域的31bit写1。 注意: 如果BD发生了错误, 也可能写回, 具体参见写回一节说明。



CONFIG		
[0]	msg_en	message发送使能，当BD执行完成以后，指示是否需要向MSG_ADDR域指示的地址写入message。 0：BD执行完毕以后不需要发送message； 1：BD执行完毕以后需要发送message。 注意：如果BD发生了错误，也可能写Message，具体参见写Message一节。

表 16-2 普通 BD DST_ADDR 域说明

DST_ADDR		
Bits	Name	Description
[31:0]	dst_addr	搬移目的地址的低32bit，即QDMAC向对应地址写数据。 注意：目的地址不递增时，目的地址必须与burst size对齐，否则会上报配置告警中断，地址递增的时候支持非对齐。

表 16-3 普通 BD SRC_ADDR 域说明

SRC_ADDR		
Bits	Name	Description
[31:0]	src_addr	搬移源地址的低32bit，即QDMAC从对应地址取数据。 注意：源地址不递增时，源地址必须与burst size对齐，否则会上报配置告警中断，地址递增的时候支持非对齐。

表 16-4 普通 BD B_COUNT/A_COUNT 域说明

B_COUNT/A_COUNT		
Bits	Name	Description
[31:16]	b_count	二维传输长度配置（0～65535）； 即一个Frame包含的Array的个数。Frame的array个数为b_count的值加1。
[15:0]	a_count	一维传输长度配置（1～65535）； 即一个Array包含的最小Byte数目。Acount配置为0会触发配置错误中断。 注意：配置为0x0000会触发配置错误中断。



表 16-5 普通 BD INIT_DATA/C_COUNT 域说明

INIT_DATA/C_COUNT		
Bits	Name	Description
[31:24]	init_data	清固定值数据指示，当CONFIG中的fixed_init_en有效时，QDMAC会将init_data复制以后写到目的端。
[23:16]	reserved	保留。
[15:0]	c_count	三维传输长度配置（0~65535）； 即一个Block包含的Frame的个数，0表示没有三维传输。Frame的个数为c_count的个数加1。

表 16-6 普通 BD BINDX 域说明

SRCBINDX/DSTBINDX		
Bits	Name	Description
[31:16]	src_bindx	二维源地址偏移量配置，按照补码进行配置，偏移地址范围-32768~32767Byte。
[15:0]	des_bindx	二维目的地址偏移量配置，按照补码进行配置，偏移地址范围-32768~32767Byte。

表 16-7 普通 BD CINDX 域说明

SRCCINDX/DSTCINDX		
Bits	Name	Description
[31:16]	src_cindx	三维源地址偏移量配置，按照补码进行配置，偏移地址范围-32768~32767Byte。
[15:0]	des_cindx	三维目的地址偏移量配置，按照补码进行配置，偏移地址范围-32768~32767Byte。

表 16-8 普通 BD LLI 域说明

LLI		
Bits	Name	Description
[31:30]	reserved	保留。



LLI		
[29:0]	nxt_bd_addr	<p>下一个BD表地址的[35:6]位，BD地址始终是512-bit对齐。即BD表存储起始地址只能为0、64、128、192、256等等。</p> <p>注意：如果使能了链接，同时地址与当前BD地址相同，会上报BD参数配置错误，BD链接会在当前任务执行完毕后终止。</p>

表 16-9 普通 BD ALG 域说明

ALG		
Bits	Name	Description
[31:24]	reserved	保留。
[23:22]	decomp_type	<p>解压缩类型指示：</p> <p>00：8bit解压到16bit；</p> <p>01：4bit解压到16bit；</p> <p>10：4bit解压到8bit；</p> <p>11：reserved。</p> <p>注意：如果decomp_en为1，配置reserved值会上报BD参数告警中断；</p>
[21]	decomp_en	<p>解压缩使能：</p> <p>0：不使能；</p> <p>1：使能。</p> <p>注意：压缩和解压缩只能使能一种，或者都不使能，否则上报BD参数告警中断。</p>
[20]	comp_en	<p>压缩使能（只支持16bit到8bit的压缩算法）：</p> <p>0：不使能；</p> <p>1：使能。</p> <p>注意：压缩和解压缩只能使能一种，或者都不使能，否则上报BD参数告警中断。</p>
[19:18]	reserved	保留。
[17:12]	fix_total_width	定点数的有效位宽指示，有效配置2~32，配置其他值QDMAC不会做检查，由软件保证。
[11:10]	reserved	保留。
[9:4]	fix_int_width	定点数的整数部分位宽指示，有效配置-31~31，配置为-32 QDMAC不会做检查，由软件保证。
[3]	reserved	保留。



ALG		
[2]	pfp_flag	浮点位宽指示： 0：半精度浮点，16bit浮点数； 1：单精度浮点，28bit浮点数。
[1]	pfp_to_fix_en	浮点转定点使能： 0：不使能； 1：使能。 注意：定点转浮点和浮点转定点只能使能一种，或者都不使能，否则上报BD配置错误中断。
[0]	fix_to_pfp_en	定点转浮点使能： 0：不使能； 1：使能。 注意：定点转浮点和浮点转定点只能使能一种，或者都不使能，否则上报BD配置错误中断。

须知

支持半精度浮点数与定点数互相转换（定点数固定为16bit），支持单精度浮点数与定点数互相转换（定点数固定为32bit），不支持半精度浮点数到32bit定点数的转换。

表 16-10 普通 BD MSG_ADDR_H/SRC_ADDR_H/DST_ADDR_H 域说明

Bits	Name	Description
[31:12]	reserved	保留。
[11:8]	msg_addr_h	msg_addr的高位地址。
[7:4]	src_addr_h	src_addr的高位地址。
[3:0]	dst_addr_h	dst_addr的高位地址。



表 16-11 普通 BD MSG_ADDR 域说明

Bits	Name	Description
[31:0]	msg_addr	message输出的低位地址，当QDMAC执行完BD指示的搬移后，由msg_en决定是否需要向该地址写对应的message。msg_addr对齐原则： msg_num域为0b00：msg_addr按照128bit对齐； msg_num域为0b01：msg_addr按照64bit对齐； msg_num域为0b10：msg_addr按照32bit对齐。 注意：如果msg_addr没有按照要求的对齐，会触发BD错误中断，同时QDMAC不会进行数据搬移。

表 16-12 普通 BD MSG_DATA0 域说明

Bits	Name	Description
[31:0]	msg_data0	QDMAC写出的message内容，即QDMAC从BD表中解析出该msg_data0，当BD执行完毕以后直接将msg_data0作为消息内容写到对应的地址。

表 16-13 普通 BD MSG_DATA1 域说明

MSG_DATA1		
Bits	Name	Description
[31:16]	msg_data1	QDMAC写出的message内容，即QDMAC从BD表中解析出该msg_data1，当BD执行完毕以后直接将msg_data1作为消息内容写到对应的地址。
[15:0]	dma_info	DMA填写的硬件信息域： bit[3]~bit[15]：reserved，读出值为全0； bit[2]：0表示参数解析正常，1表示参数解析告警（Message配置错误除外）； bit[1]：0表示数据搬移正常，1表示数据搬移告警； bit[0]：0表示正常消息，1表示告警消息。

表 16-14 普通 BD MSG_DATA2 域说明

Bits	Name	Description
[31:0]	msg_data2	QDMAC写出的message内容，即QDMAC从BD表中解析出该msg_data2，当BD执行完毕以后直接将msg_data2作为消息内容写到对应的地址。



表 16-15 普通 BD MSG_DATA3 域说明

Bits	Name	Description
[31:0]	msg_data3	QDMAC写出的message内容，即QDMAC从BD表中解析出该msg_data3，当BD执行完毕以后直接将msg_data3作为消息内容写到对应的地址。

16.4 IP 配置流程

16.5 寄存器写保护设置

16.5.1 场景描述

QDMAC的寄存器根据被操作的频度划分为静态寄存器和动态寄存器，对静态寄存器进行了写保护，避免对静态寄存器的误操作导致系统异常。

动态寄存器包括：

- Queue的任务配置寄存器QDMAC_QUEUE_TASK，偏移地址0x45000 + 0x1000×qn，qn=0~15。
- 原始完成中断寄存器QDMAC_QUEUE_INT_TC_RAW，偏移地址0x45240 + 0x1000×qn，qn=0~15。
- 原始错误中断寄存器QDMAC_QUEUE_INT_ALARM_RAW，偏移地址0x45250 + 0x1000×qn，qn=0~15。
- 写保护寄存器QDMAC_LOCK，偏移地址0x000。

16.5.2 配置流程

寄存器写保护设置配置流程：

步骤1 解除写保护：配置写保护寄存器QDMAC_LOCK为0x5A5A_5A5A完成对寄存器的写保护解锁，默认为解锁状态，如表16-16所示。

表 16-16 QDMAC_LOCK 寄存器

Bits	寄存器说明	配置值
QDMAC_LOCK：偏移地址0x000		



Bits	寄存器说明	配置值
[31:0]	COMM域写Lock寄存器。 向Lock寄存器写32bits密钥（密钥为0x5A5A_5A5A），解锁寄存器，表示所有寄存器可写；向Lock寄存器写其他任意值，锁定寄存器，Lock保护范围内的寄存器不能写，Lock保护范围外的寄存器可以写； 读任何寄存器不受Lock寄存器的影响，任何时刻都可以读。 读Lock寄存器（Lock寄存器读出值表示寄存器的锁定状态）： 0x0000_0000表示寄存器处于解锁状态； 0x0000_0001表示寄存器处于锁定状态。 Lock寄存器锁定范围请见每个寄存器的描述。 受COMM VMID保护。	0x5A5A_5A5A

步骤2 锁定写保护：配置写保护寄存器QDMAC_LOCK为0x0000_0000（不等于0x5A5A_5A5A均可）完成对寄存器的写保护锁定，如表16-17所示。

表 16-17 QDMAC_LOCK 寄存器

Bits	寄存器说明	配置值
QDMAC_LOCK：偏移地址0x000		
[31:0]	COMM域写Lock寄存器。 向Lock寄存器写32bits密钥（密钥为0x5A5A_5A5A），解锁寄存器，表示所有寄存器可写；向Lock寄存器写其他任意值，锁定寄存器，Lock保护范围内的寄存器不能写，Lock保护范围外的寄存器可以写； 读任何寄存器不受Lock寄存器的影响，任何时刻都可以读。 读Lock寄存器（Lock寄存器读出值表示寄存器的锁定状态）： 0x0000_0000表示寄存器处于解锁状态； 0x0000_0001表示寄存器处于锁定状态。 Lock寄存器锁定范围请见每个寄存器的描述。 受COMM VMID保护。	0x0000_0000

----结束

16.5.3 配置约束



须知

只有COMM寄存器受QDMAC_LOCK寄存器的写保护，COMM寄存器除了QDMAC_COMM_INT_MSK、QDMAC_COMM_INT_RAW和QDMAC_LOCK寄存器在写保护处于锁定状态时可以进行写操作以外，其余所有寄存器在处于写保护状态下进行写操作都会导致异常。

16.6 寄存器多模隔离设置

16.6.1 场景描述

QDMAC的寄存器根据功能特性划分成了四大类：COMM寄存器、INT寄存器、CHG寄存器、CH寄存器，不同类别的寄存器根据配置的不同VMID值进行保护检查；COMM类寄存器是通用配置寄存器，INT寄存器是中断配置寄存器，CHG寄存器是通道组寄存器，CH是通道寄存器。

- COMM类寄存器的写操作受COMM_VMID的保护，只有当访问COMM类寄存器的操作所携带的VMID为配置的COMM_VMID或者0时，才能操作成功，否则，即违反VMID检查，上报pslverr，同时上报VMID不匹配中断。
- INT寄存器的写操作受INT_VMID的保护，只有当访问INT寄存器的操作所携带的VMID为配置的INT_VMID或者0时，才能操作成功，否则，即违反VMID检查，上报pslverr，同时上报VMID不匹配中断。
- CHG寄存器的写操作受CHG_VMID的保护，只有当访问CHG寄存器的操作所携带的VMID为配置的CHG_VMID或者0时，才能操作成功，否则，即违反VMID检查，上报pslverr，同时上报VMID不匹配中断。
- CH寄存器的写操作受它所属的通道组的CHG_VMID的保护，只有当访问CH寄存器的操作所携带的VMID为配置的CHG_VMID或者0时，才能操作成功，否则，即违反VMID检查，上报pslverr，同时上报VMID不匹配中断。

16.6.2 寄存器多模隔离使能配置流程

寄存器多模隔离使能配置流程：

- 步骤1** 获取COMM_VMID：读取QDMAC_COMM_VMID寄存器，如表16-17所示，读取到的值为COMM_VMID的值，记为comm_vmid。
- 步骤2** 解除写保护：使用comm_vmid，即寄存器操作携带的VMID值为comm_vmid，配置写保护寄存器QDMAC_LOCK为0x5A5A_5A5A完成对寄存器的写保护解锁，默认为解锁状态，如表16-16所示。

表 16-18 QDMAC_LOCK 寄存器

Bits	寄存器说明	配置值
	QDMAC_LOCK：偏移地址0x000	



Bits	寄存器说明	配置值
[31:0]	<p>写Lock寄存器。</p> <p>向Lock寄存器写32bits密钥，解锁寄存器，表示所有寄存器可写；向Lock寄存器写其他任意值，锁定寄存器，Lock保护范围内的寄存器不能写，Lock保护范围外的寄存器可以写；</p> <p>读任何寄存器不受Lock寄存器的影响，任何时刻都可以读。</p> <p>读Lock寄存器：</p> <p>Lock寄存器读出值表示寄存器的锁定状态：</p> <p>0x0000_0000表示寄存器处于解锁状态；</p> <p>0x0000_0001表示寄存器处于锁定状态。</p> <p>Lock寄存器锁定范围请见每个寄存器的描述。</p> <p>受COMM VMID保护</p>	0x5A5A_5A5A

步骤3 配置COMM_VMID：使用comm_vmid，即寄存器操作携带的VMID值为comm_vmid，配置QDMAC_COMM_VMID为所需要的值，如表16-17所示。

表 16-19 QDMAC_COMM_VMID 寄存器

Bits	寄存器说明	配置值
QDMAC_COMM_VMID：偏移地址0x004		
[31:3]	保留。	0x0000_0000
[2:0]	<p>COMM的VMID配置寄存器。</p> <p>受Lock写锁保护和COMM VMID保护</p>	0x1 ~ 0x7

步骤4 配置INT_VMID：使用comm_vmid，即寄存器操作携带的VMID值为comm_vmid，配置QDMAC_INT_VMID为所需要的值，如表16-20所示。

表 16-20 QDMAC_INT_VMID 寄存器

Bits	寄存器说明	配置值
QDMAC_INT_VMID：偏移地址0x00100 + 0x4×in (QDMAC_GLB: in = 0 ~ 33)，其他配置 (in = 0 ~ 9)		
[31:3]	保留。	0x0000_0000
[2:0]	<p>INT in的VMID配置寄存器，QDMA用于保护传输完成中断配置寄存器。</p> <p>受Lock写锁保护和COMM VMID保护。</p>	0x1 ~ 0x7

步骤5 配置CHG_VMID：使用comm_vmid，即寄存器操作携带的VMID值为comm_vmid，配置QDMAC_CHG_VMID为所需要的值，如表16-21所示。



表 16-21 QDMAC _ CHG _ VMID 寄存器

Bits	寄存器说明	配置值
QDMAC _ CHG _ VMID: 偏移地址0x00200 + 0x20×gn (gn = 0 ~ 3)		
[31:3]	保留。	0x0000_0000
[2:0]	通道组gn的VMID配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x1 ~ 0x7

步骤6 配置多模隔离保护使能：使用comm_vmid，即寄存器操作携带的VMID值为comm_vmid，配置QDMAC_GLB_EN bit[2]的值为0x1，该bit默认值为0，其他bit的配置值保持不变，如表16-22所示。

表 16-22 QDMAC_GLB_EN 寄存器

Bits	寄存器说明	配置值
QDMAC_GLB_EN: 偏移地址0x000		
[31:4]	保留。	0x000_0000
[3]	AXI读命令间隔使能。 写： 0：未使能，AXI读命令连续发出； 1：使能，AXI读命令会间隔arlen发出。 受Lock写锁保护和COMM VMID保护。	-
[2]	QDMAC VMID检查使能。 写： 0：未使能，不检查VMID访问的合法性； 1：使能，对各类参数的APB写访问进行VMID合法性检查。 受Lock写锁保护和COMM VMID保护。	0x1
[1]	QDMAC告警消息发送使能。 写： 0：未使能，如果BD发生了告警事件，即使消息使能也不会发送告警消息； 1：使能，如果BD发生了告警事件，且消息使能，则发送告警消息。 受Lock写锁保护和COMM VMID保护。	-



Bits	寄存器说明	配置值
[0]	QDMAC处理能力使能寄存器。 写： 0：未使能（如果是撤销使能，则处理能力隔离功能立即失效，各通道组的累计使用时间清0）； 1：使能。 受Lock写锁保护和COMM VMID保护。	-

----结束

16.6.3 寄存器多模隔离关闭配置流程

寄存器多模隔离关闭配置流程：

- 步骤1** 获取COMM_VMID：读取QDMAC_COMM_VMID寄存器，如表16-19所示，读取到的值为COMM_VMID的值，记为comm_vmid。
- 步骤2** 解除写保护：使用comm_vmid，即寄存器操作携带的VMID值为comm_vmid，配置写保护寄存器QDMAC_LOCK为0x5A5A_5A5A完成对寄存器的写保护解锁，默认为解锁状态，如表16-23所示。

表 16-23 QDMAC_LOCK 寄存器

Bits	寄存器说明	配置值
QDMAC_LOCK：偏移地址0x000		
[31:0]	写Lock寄存器。 向Lock寄存器写32bits密钥，解锁寄存器，表示所有寄存器可写；向Lock寄存器写其他任意值，锁定寄存器，Lock保护范围内的寄存器不能写，Lock保护范围外的寄存器可以写； 读任何寄存器不受Lock寄存器的影响，任何时刻都可以读。 读Lock寄存器。 Lock寄存器读出值表示寄存器的锁定状态： 0x0000_0000表示寄存器处于解锁状态； 0x0000_0001表示寄存器处于锁定状态。 Lock寄存器锁定范围请见每个寄存器的描述。 受COMM VMID保护。	0x5A5A_5A5A

- 步骤3** 配置多模隔离保护关闭：使用comm_vmid，即寄存器操作携带的VMID值为comm_vmid，配置QDMAC_GLB_EN bit[2]的值为0x0，该bit默认值为0，其他bit的配置值保持不变，如表16-24所示。



表 16-24 QDMAC_GLB_EN 寄存器

Bits	寄存器说明	配置值
QDMAC_GLB_EN：偏移地址0x000		
[31:4]	保留。	0x000_0000
[3]	AXI读命令间隔使能。 写： 0：未使能，AXI读命令连续发出； 1：使能，AXI读命令会间隔arlen发出。 受Lock写锁保护和COMM VMID保护。	-
[2]	QDMAC VMID检查使能。 写： 0：未使能，不检查VMID访问的合法性； 1：使能，对各类参数的APB写访问进行VMID合法性检查。 受Lock写锁保护和COMM VMID保护。	0x0
[1]	QDMAC告警消息发送使能。 写： 0：未使能，如果BD发生了告警事件，即使消息使能也不会发送告警消息； 1：使能，如果BD发生了告警事件，且消息使能，则发送告警消息。 受Lock写锁保护和COMM VMID保护。	-
[0]	QDMAC处理能力使能寄存器。 写： 0：未使能（如果是撤销使能，则处理能力隔离功能立即失效，各通道组的累计使用时间清0）； 1：使能。 受Lock写锁保护和COMM VMID保护。	-

----结束

16.6.4 配置约束

无

16.7 Queue 的初始化



16.7.1 场景描述

QDMAC是基于Queue的结构对任务进行管理，每个Queue的软件管理地址相互独立，包括任务配置、任务查询、任务撤销、任务完成状态、Queue的撤销，每个Queue的深度（即每个Queue允许最大的配置任务数）为QDMAC允许配置的最大任务数除以Queue的个数，QDMAC最大支持的任务数为512个，Queue的个数软件可以配置为1、2、4、8、16，每个Queue可以单独配置优先级；

对于不同的业务场景，需要的Queue的个数不一样，每个Queue的优先级也可能不一样，因此，在系统上电以后或者切换业务以后，软件首先应该完成QDMAC中Queue的初始化（个数配置，优先级设置）。

16.7.2 配置流程

Queue的初始化配置流程：

- 步骤1** 解除写保护，请参见“[配置流程的步骤1](#)”。
- 步骤2** 查询全局状态寄存器QDMAC_STAT bit[4]为0b0，表示QDMAC的所有任务均已完成且产生的所有中断均已清除，如[表16-25](#)所示。

表 16-25 QDMAC_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_STAT：偏移地址0x324		
[31:5]	保留。	-
[4]	DMAC的忙闲指示信号。 0b0：DMAC处于闲状态； 0b1：DMAC处于忙状态。	0b0
[3:1]	保留。	-
[0]	Memory的低功耗状态指示： 0b0：Memory处于Light Sleep状态； 0b1：Memory未进入Light Sleep状态。	-

- 步骤3** 配置全局控制寄存器QDMAC_QUEUE_NUM bit[2:0]完成Queue的数目配置，默认为1个Queue；例如配置为0b11，表示配置为8个Queue，每个Queue的深度为512/8=64，如[表16-26](#)所示。

表 16-26 QDMAC_QUEUE_NUM 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_NUM：偏移地址0x300		
[31:3]	保留。	-



Bits	寄存器说明	配置值
[2:0]	Queue的个数，QDMAC将任务通道按照等分到各个Queue。 000：1个Queue； 001：2个Queue； 010：4个Queue； 011：8个Queue； 1xx：16个Queue。 受Lock写锁保护和COMM VMID保护	0b11（8个Queue）

步骤4 配置通道组选择配置寄存器QDMAC_QUEUE_CHG_IDX bit[1:0]完成通道组的选择配置，该寄存器的默认值为0，即默认选择通道组0；例如配置QDMAC_QUEUE_CHG_IDX2为0b11，表示配置为Queue2属于通道组3，如表16-27所示。

表 16-27 QDMAC_QUEUE_CHG_IDX 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_CHG_IDX：偏移地址0x00290 + 0x4×qn（qn = 0 ~ 15）		
[31:3]	保留。	-
[2:0]	Queue qn的通道组选择配置寄存器。 受Lock写锁保护和COMM VMID保护。	0b11（属于通道组3）

步骤5 配置优先级控制寄存器QDMAC_QUEUE_PRI bit[1:0]分别完成每个Queue的优先级设置。如表16-28所示。

表 16-28 QDMAC_QUEUE_PRI 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_PRI qn：偏移地址0x45220+0x1000*qn		
[31:2]	保留。	-
[1:0]	第qn个Queue的优先级配置。 00：优先级为0（最高）； 01：优先级为1； 10：优先级为2； 11：优先级为3（最低）。 受Queue qn的VMID保护。	0b00

步骤6 锁定写保护，请参见“配置流程的步骤2”；

----结束

16.7.3 配置约束

须知

软件必须保证切换Queue的数目时QDMAC的所有任务均已完成且产生的所有中断均已清除，否则切换失败并且会导致异常。

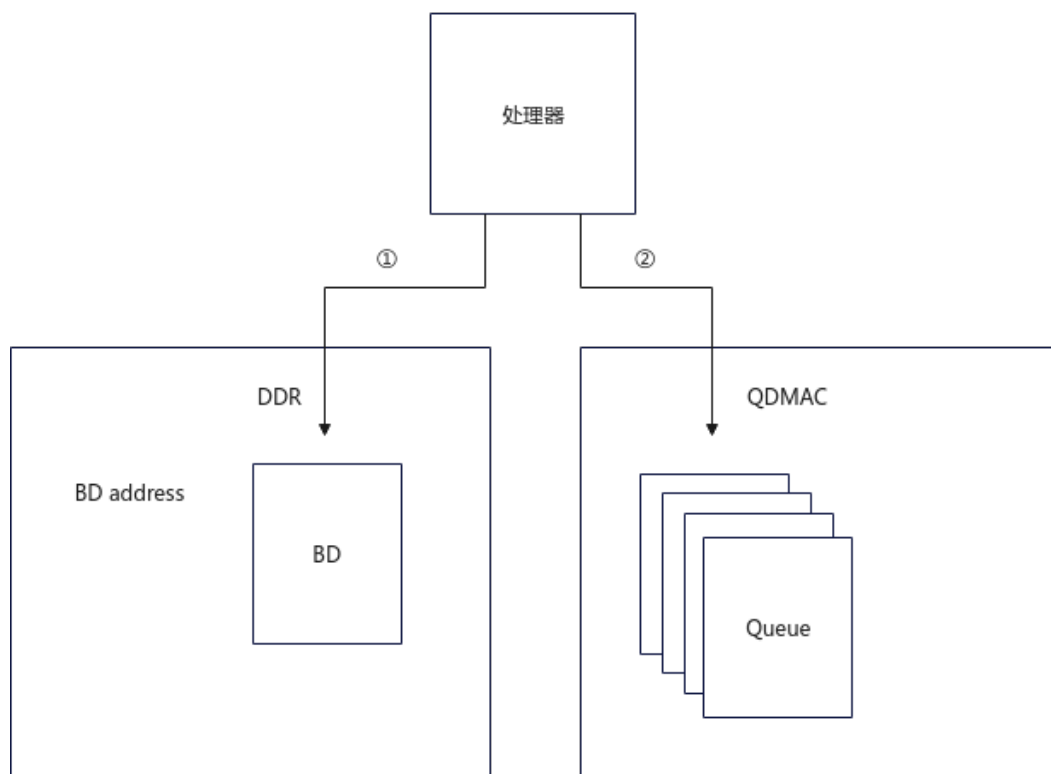
16.8 Queue 的任务配置

16.8.1 普通 BD 任务场景描述

任务即软件分配给QDMAC的一次数据搬移任务，该搬移任务可能只包含一个BD，也可能包含多个BD（即BD链接任务）。软件只需将BD的指针配置到QDMAC的Queue中，QDMAC则认为接收到了一个任务，同时到BD指针指示的地址读回具体的搬移参数，解析以后执行数据搬移。

配置流程如图16-6所示。

图 16-6 Queue 任务配置流程图



步骤1 软件首先准备好BD参数，并将其存储于系统的存储器中。

步骤2 将BD address配置到QDMAC的某个Queue中实现任务下发。

----结束



16.8.2 普通 BD 任务配置流程

Queue的任务配置流程：

- 步骤1** 软件将BD写到系统的存储器中，并记录地址（BD指针）；例如，BD被写到存储器的0x2000_0000。
- 步骤2** 查询QDMAC_QUEUE_STAT qn 寄存器的bit[9:0]获取第 qn 个Queue中是否还有空间存放任务；例如软件需要向Queue1配置任务，则查询QDMAC_QUEUE_STAT1（偏移地址0x46208）寄存器的bit[9:0]，如果不为0则表示可以配置任务，如表16-29所示。

表 16-29 QDMAC_QUEUE_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_STAT qn : 偏移地址0x45208+0x1000* qn , $qn=0\sim15$		
[31:13]	保留。	-
[12]	第 qn 个Queue中的所有任务是否都已执行完毕状态指示： 0：Queue中的所有任务均已执行完毕； 1：Queue中的任务还未执行完毕。	-
[11:10]	保留。	-
[9:0]	第 qn 个Queue中空闲的任务通道数目。 0x000：Queue中无空闲通道； 0x001：Queue中有1个空闲通道； ... 0x200：Queue中有512个空闲通道。 注意：初始化后，还未配置Queue的个数，则默认为一个Queue，Queue空闲的通道数目为总的任务数目，如果配置了Queue的个数，则根据个数平分总的任务数目	0x001（还有一个剩余任务空间）

- 步骤3** 配置BD指针到QDMAC_QUEUE_TASK qn 寄存器完成任务配置。例如将0x2000_0000配置到QDMAC_QUEUE_TASK1（偏移地址0x46000），则表示将BD存放于0x2000_0000的任务配置到了QDMAC的第1个Queue中，如表16-30所示。

表 16-30 QDMAC_QUEUE_TASK 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_TASK qn : 偏移地址0x45000+0x1000* qn , $qn=0\sim15$		
[31:30]	保留。	-
[29:0]	第 qn 个Queue的任务配置寄存器，将任务的BD表高30bit（[35:6]）地址配置到该寄存器。 受Queue qn 的VMID保护	0x0080_0000



----结束

16.8.3 配置约束

须知

1. 软件必须首先将BD写到系统的存储器中，再将BD指针配置到QDMAC的Queue中。

2. BD存放的地址必须按照512bit对齐，即地址的低6bit必须为0，否则将会导致QDMAC解析BD错误。

3. 软件配置任务时需要保证Queue中有剩余空间可以存储任务，否则会触发异常中断。

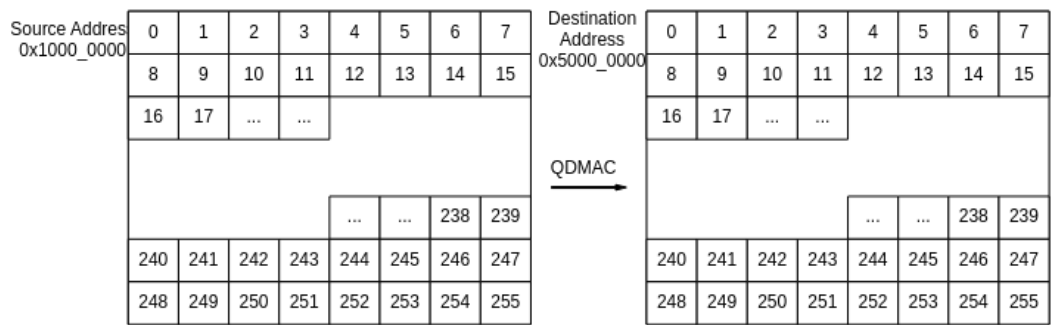
16.8.4 配置约束

无

16.8.5 实例——普通 BD 一维 M2M 搬移

一维M2M（Memory到Memory）搬移即将一块连续地址区域的数据搬移到另外一块连续的地址区域，图16-7为一维源和目的都以递增的方式完成256Byte数据的搬移：

图 16-7 一维 M2M 搬移示意图



假设数据搬移完成以后写回以及写消息，消息地址为0xF000_0000，消息数据为0x5A5A_5A5A，0xA5A5_A5A5，0x5555_5555，0xAAAA_AAAA，按表16-31组织BD：

表 16-31 一维 M2M BD 配置表

BD结构（请参见“协议或包格式说明”章节）	配置值
CONFIG	0x00C0_0003
DST_ADDR	0x5000_0000
SRC_ADDR	0x1000_0000

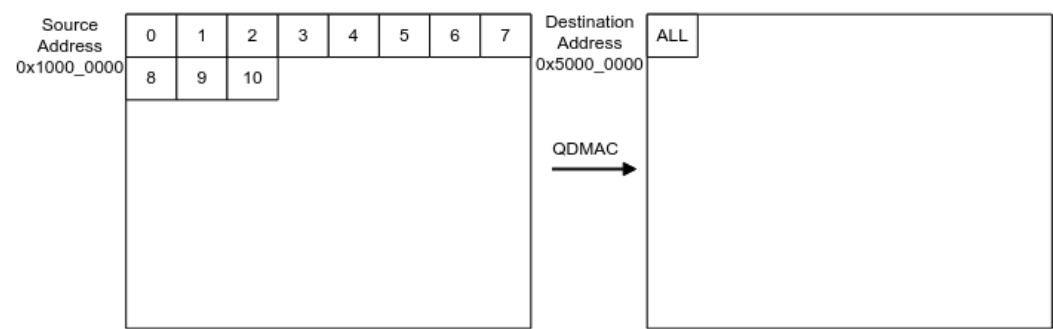


BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
B_COUNT			A_COUNT	0x0000	0x0100
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVE D	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0xF000_0000	
MSG_DATA0				0x5A5A_5A5A	
MSG_DATA1				0xA5A5_A5A5	
MSG_DATA2				0x5555_5555	
MSG_DATA3				0xAAAA_AAAA	

16.8.6 实例——普通 BD 一维 M2F 搬移

一维M2F（Memory到FIFO）搬移即将一块连续地址区域的数据搬移到另外一块固定地址区域，[图16-8](#)为一维源地址递增，目的地址固定的方式完成11Byte数据的搬移：

图 16-8 一维 M2F 搬移示意图



假设数据搬移完成以后只写回，按[表16-32](#)组织BD：



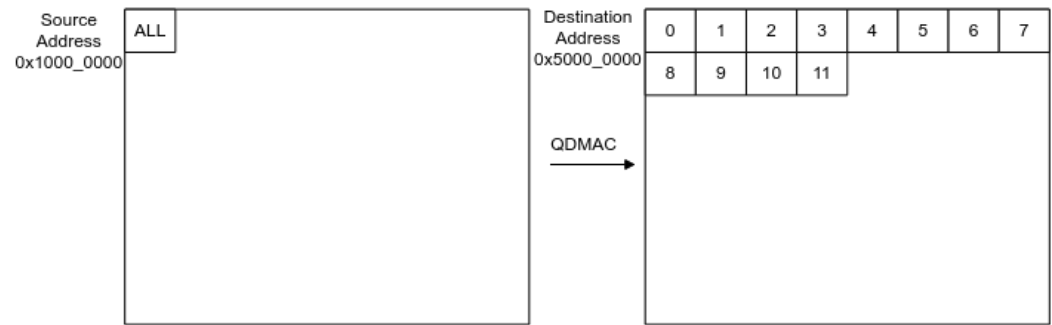
表 16-32 一维 M2F BD 配置表

BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x0080_0002	
DST_ADDR				0x5000_0000	
SRC_ADDR				0x1000_0000	
B_COUNT			A_COUNT	0x0000	0x000B
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0x0000_0000	
MSG_DATA0				0x0000_0000	
MSG_DATA1				0x0000_0000	
MSG_DATA2				0x0000_0000	
MSG_DATA3				0x0000_0000	

16.8.7 实例——普通 BD 一维 F2M 搬移

一维F2M（FIFO到Memory）搬移即将一块固定地址区域的数据搬移到另外一块连续地址区域，图16-9为一维源地址固定，目的地址递增的方式完成11Byte数据的搬移：

图 16-9 一维 F2M 搬移示意图





假设数据搬移完成以后只写消息，消息地址为0xF000_0000，消息数据为0x5A5A_5A5A，0xA5A5_A5A5，0x5555_5555，0xAAAA_AAAA，按表16-33组织BD：

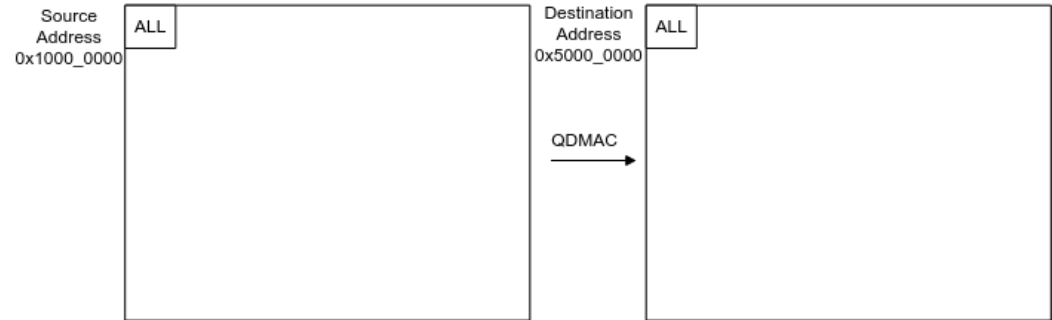
表 16-33 一维 F2M BD 配置表

BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x0040_0001	
DST_ADDR				0x5000_0000	
SRC_ADDR				0x1000_0000	
B_COUNT			A_COUNT	0x0000	0x000B
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0xF000_0000	
MSG_DATA0				0x5A5A_5A5A	
MSG_DATA1				0xA5A5_A5A5	
MSG_DATA2				0x5555_5555	
MSG_DATA3				0xAAAA_AAAA	

16.8.8 实例——普通 BD 一维 F2F 搬移

一维F2F（FIFO到FIFO）搬移即将一块固定地址区域的数据搬移到另外一块固定地址区域，图16-10为一维源地址和目的地址都固定的方式完成16Byte数据的搬移：

图 16-10 一维 F2F 搬移示意图





假设数据搬移完成以后只触发完成中断，按表16-34组织BD：

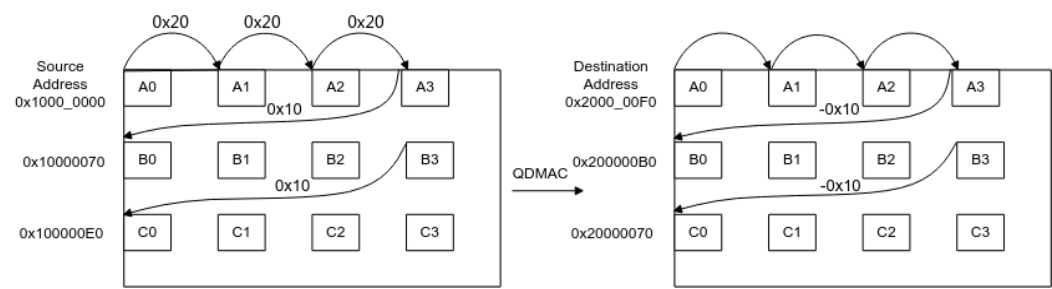
表 16-34 一维 F2F BD 配置表

BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x0000_0004	
DST_ADDR				0x5000_0000	
SRC_ADDR				0x1000_0000	
B_COUNT			A_COUNT	0x0000	0x0010
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0x0000_0000	
MSG_DATA0				0x0000_0000	
MSG_DATA1				0x0000_0000	
MSG_DATA2				0x0000_0000	
MSG_DATA3				0x0000_0000	

16.8.9 实例——普通 BD 三维 M2M A-sync 搬移

由于二维与三维比较类似，因此这里以三维为例进行说明。三维M2M A-sync模式可以完成多块规则的连续地址数据的搬移，图16-11为三维源地址和目的地址都递增的方式完成12Byte的数据的搬移：

图 16-11 三维 M2M A-sync 搬移示意图





假设数据搬移完成以后触发完成中断并且写消息，消息地址为0xF000_0000，消息数据为0x5A5A_5A5A，0xA5A5_A5A5，0x5555_5555，0xAAAA_AAAA，按表16-35组织BD。

表 16-35 三维 M2M A-sync 模式 BD 配置表

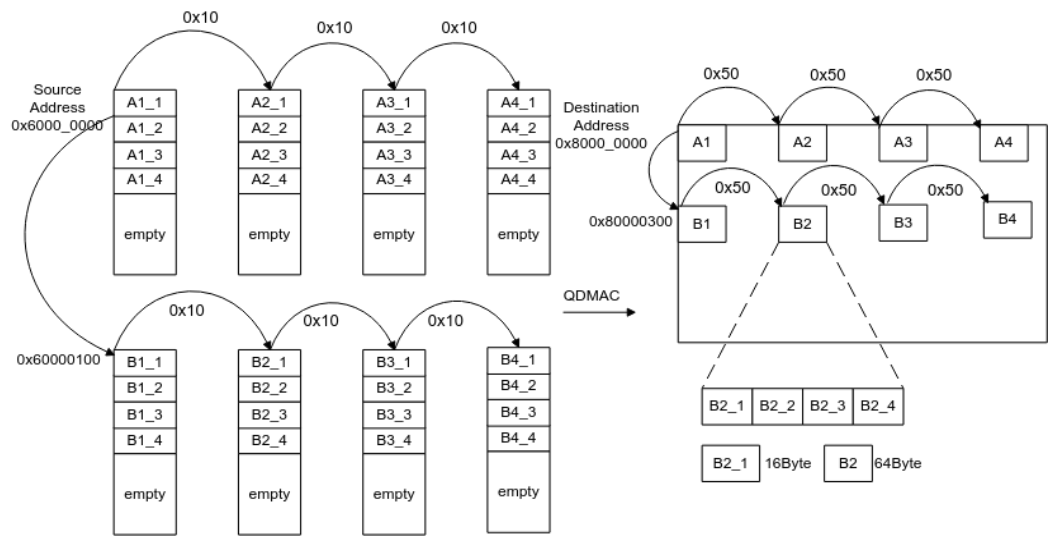
BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x00C0_0005	
DST_ADDR				0x2000_00F0	
SRC_ADDR				0x1000_0000	
B_COUNT			A_COUNT	0x0003	0x0001
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0002
SRCBINDX			DSTBINDX	0x0020	0xFFFF0
SRCCINDX			DSTCINDX	0x0010	0xFFFF0
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0xF000_0000	
MSG_DATA0				0x5A5A_5A5A	
MSG_DATA1				0xA5A5_A5A5	
MSG_DATA2				0x5555_5555	
MSG_DATA3				0xAAAA_AAAA	

16.8.10 实例——普通 BD 三维 F2M AB-sync 搬移

三维F2M AB-sync模式可以完成多块规则的固定地址（类似于FIFO）数据的搬移到多块连续地址空间，图16-12为三维源地址固定，目的地址递增的方式完成128Byte的数据的搬移：



图 16-12 三维 F2M AB-sync 搬移示意图



假设数据搬移完成以后触发完成中断并且写回，按表16-36组织BD。

表 16-36 三维 F2M AB-sync 模式 BD 配置表

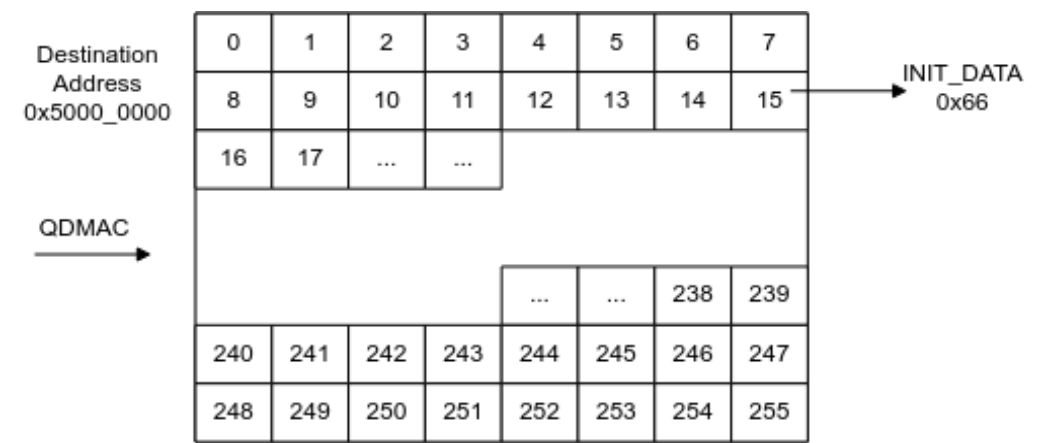
BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x0070_0006	
DST_ADDR				0x8000_0000	
SRC_ADDR				0x6000_0000	
B_COUNT			A_COUNT	0x0003	0x0040
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0001
SRCBINDX			DSTBINDX	0x0010	0x0050
SRCCINDX			DSTCINDX	0x0100	0x0300
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0x0000_0000	
MSG_DATA0				0x0000_0000	
MSG_DATA1				0x0000_0000	
MSG_DATA2				0x0000_0000	
MSG_DATA3				0x0000_0000	



16.8.11 实例——普通 BD 清固定值任务

清固定值任务不关心源地址，将BD中的INIT_DATA复制到目的地址，搬移的维度可以实现一维、二维、三维，可以实现Memory（目的地址递增）的初始化以及FIFO（目的地址固定）的数据填充。[图16-13](#)是一个一维初始化Memory的应用，初始化空间为256Byte，初始化数据为0x66：

图 16-13 清固定值任务搬移示意图



假设数据搬移完成以后写回以及写消息，消息地址为0xF000_0000，消息数据为0x5A5A_5A5A，0xA5A5_A5A5，0x5555_5555，0xAAAA_AAAA，按[表16-37](#)组织BD：

表 16-37 清固定值任务 BD 配置表

BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x0440_0003	
DST_ADDR				0x5000_0000	
SRC_ADDR				0x0000_0000	
B_COUNT		A_COUNT		0x0000	0x0100
INIT_DATA	RESERVED		C_COUNT	0x6600	0x0000
SRCBINDX		DSTBINDX		0x0000	0x0000
SRCCINDX		DSTCINDX		0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0000_0000	



BD结构（请参见“协议或包格式说明”章节）				配置值
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000
MSG_ADDR				0xF000_0000
MSG_DATA0				0x5A5A_5A5A
MSG_DATA1				0xA5A5_A5A5
MSG_DATA2				0x5555_5555
MSG_DATA3				0xAAAA_AAAA

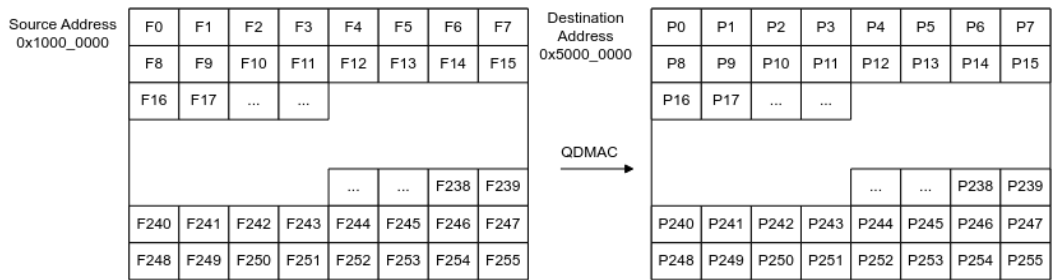
须知

清固定值任务使能时不能使能定浮点转换或者压缩解压缩功能，否则会导致配置异常中断。

16.8.12 实例——普通 BD 定浮点转换任务

定浮点任务即QDMAC将源数据实现定点转浮点或者浮点转定点以后再写往目的地址，支持半精度浮点数（位宽16bit）与定点数（位宽 16bit）互相转换，支持单精度浮点数（位宽28bit）与定点数（位宽 32bit）互相转换。软件通过BD ALG域实现转换使能和参数的配置。搬移的维度可以实现一维、二维、三维，源和目的地址都可以为递增或者固定模式。图16-14是一个半精度转定点数的应用。

图 16-14 定浮点转换任务搬移示意图



假设数据搬移完成以后写回以及写消息，消息地址为0xF000_0000，消息数据为0x5A5A_5A5A，0xA5A5_A5A5，0x5555_5555，0xAAAA_AAAA，按表16-38组织BD：

表 16-38 定浮点转换任务 BD 配置表

BD结构（请参见“协议或包格式说明”章节）		配置值
CONFIG		0x00C0_0003
DST_ADDR		0x5000_0000



BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
SRC_ADDR				0x1000_0000	
B_COUNT			A_COUNT	0x0000	0x0100
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0001_0101	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0xF000_0000	
MSG_DATA0				0x5A5A_5A5A	
MSG_DATA1				0xA5A5_A5A5	
MSG_DATA2				0x5555_5555	
MSG_DATA3				0xAAAA_AAAA	

须知

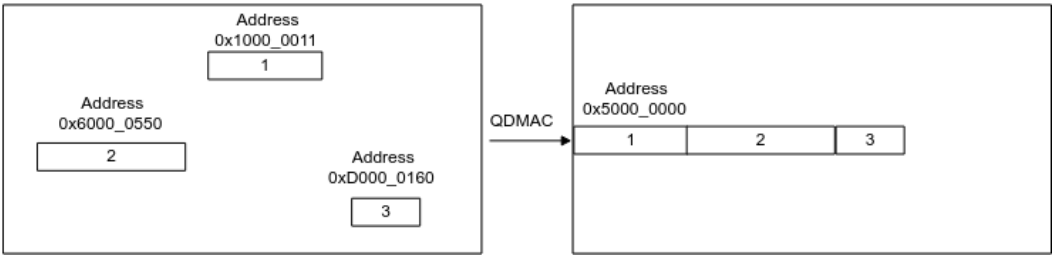
1. 对于单精度浮点转定点，软件保证将源数据扩展到32bit，QDMAC取高28bit参与、转换，转换完成后输出32bit的定点数；
2. 对于定点转单精度浮点，转换输出的浮点数为28bit，QDMAC在低位拼4bit 0组合出32bit数据；
3. 对于半精度转换，源和目的地址以及多维地址偏移必须以16bit对齐，长度必须为16bit整数倍，否则任务不被执行并会触发配置异常中断；
4. 对于单精度转换，源和目的地址以及多维地址偏移必须以32bit对齐，长度必须为32bit整数倍，否则任务不被执行并会触发配置异常中断。

16.8.13 实例——普通 BD 链接任务

QDMAC内部一个任务可以配置一个BD指针，该BD除了指示QDMAC搬移的参数，同时还可以指示是否需要再链接到下一个BD，同时下一个BD指针也包含在该BD中。因此，当前BD指示的数据搬移完成以后，QDMAC会自动读取下一个BD的内容进行解析并执行数据搬移，链的长度原则上可以无限制，当BD中的链接使能无效时链表结束。BD链接中的任务可以是清固定值任务、定浮点转换任务、普通任务。[图16-15](#)是三个BD任务链接的应用：



图 16-15 BD 链接任务搬移示意图



假设第一个BD任务为一维16个Byte的搬移，搬移完成后只写回，第二个BD任务为一维24个Byte的搬移，搬移完成后只写回，第三个BD为最后一个节点，一维8个Byte的搬移，搬移完成写回并且发送消息，三个节点的数据被搬移到目的地址为0x5000_0000开始的连续区域，第二、三个BD所在地址为0x0000_0040和0x0000_0080，BD1配置如表16-39所示，BD2配置如表16-40所示，BD3配置如表16-41所示。

表 16-39 BD 链接任务 BD1 配置表

BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x00C0_000A	
DST_ADDR				0x5000_0000	
SRC_ADDR				0x1000_0011	
B_COUNT			A_COUNT	0x0000	0x0010
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0001	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0x0000_0000	
MSG_DATA0				0x0000_0000	
MSG_DATA1				0x0000_0000	
MSG_DATA2				0x0000_0000	
MSG_DATA3				0x0000_0000	



表 16-40 BD 链接任务 BD2 配置表

BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x00C0_000A	
DST_ADDR				0x5000_0010	
SRC_ADDR				0x6000_0550	
B_COUNT			A_COUNT	0x0000	0x0018
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0002	
RESERVED				0x0000_0000	
ALG				0x0000_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0x0000_0000	
MSG_DATA0				0x0000_0000	
MSG_DATA1				0x0000_0000	
MSG_DATA2				0x0000_0000	
MSG_DATA3				0x0000_0000	

表 16-41 BD 链接任务 BD3 配置表

BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
CONFIG				0x00C0_0003	
DST_ADDR				0x5000_0028	
SRC_ADDR				0xD000_0160	
B_COUNT		A_COUNT		0x0000	0x0008
INIT_DATA	RESERVED		C_COUNT	0x0000	0x0000
SRCBINDX			DSTBINDX	0x0000	0x0000
SRCCINDX			DSTCINDX	0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	

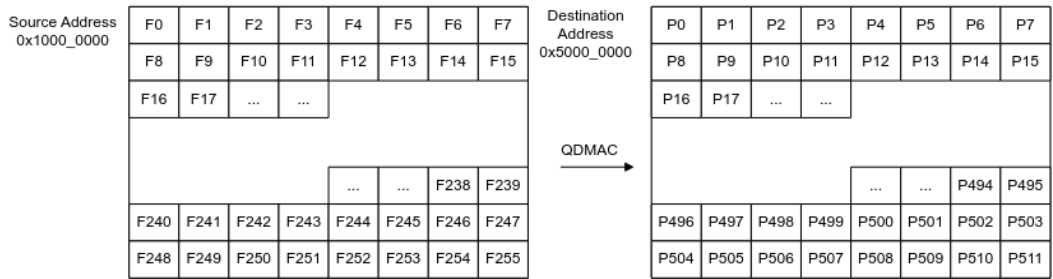


BD结构（请参见“协议或包格式说明”章节）				配置值
ALG				0x0000_0000
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000
MSG_ADDR				0xF000_0000
MSG_DATA0				0x5A5A_5A5A
MSG_DATA1				0xA5A5_A5A5
MSG_DATA2				0x5555_5555
MSG_DATA3				0xAAAA_AAAA

16.8.14 实例——普通 BD 压缩解压缩任务

压缩解压缩即QDMAC将源数据实现压缩或者解压缩以后再写往目的地址，支持4-16bit解压缩、4-8bit解压缩、8-16bit解压缩、16-8压缩。软件通过BD ALG域实现压缩解压缩使能和参数的配置。搬移的维度只支持一维，源和目的地址都为递增模式。
图16-14是一个8-16bit解压缩的应用：

图 16-16 压缩解压缩任务搬移示意图



假设数据搬移完成以后写回以及写消息，消息地址为0xF000_0000，消息数据为0x5A5A_5A5A，0xA5A5_A5A5，0x5555_5555，0xAAAA_AAAA，按表16-38组织BD：

表 16-42 定浮点转换任务 BD 配置表

BD结构（请参见“协议或包格式说明”章节）				配置值
CONFIG				0x00C0_0003
DST_ADDR				0x5000_0000
SRC_ADDR				0x1000_0000
B_COUNT		A_COUNT	0x0000	0x0100
INIT_DATA	RESERVED	C_COUNT	0x0000	0x0000



BD结构（请参见“ 协议或包格式说明 ”章节）				配置值	
SRCBINDX		DSTBINDX		0x0000	0x0000
SRCCINDX		DSTCINDX		0x0000	0x0000
LLI				0x0000_0000	
RESERVED				0x0000_0000	
ALG				0x0020_0000	
RESERVED	MSGADDR_H	SRCADDR_H	DSTADDR_H	0x0000_0000	
MSG_ADDR				0xF000_0000	
MSG_DATA0				0x5A5A_5A5A	
MSG_DATA1				0xA5A5_A5A5	
MSG_DATA2				0x5555_5555	
MSG_DATA3				0xAAAA_AAAA	

须知

1. 压缩和解压缩只能使能一种，不能同时使能压缩和解压缩，否则任务不被执行并行会触发配置异常中断。
2. 16bit的待压缩数据存储必须按照128bit对齐，8bit的待解压缩数据存储必须按照64bit对齐，4bit的待解压缩数据存储必须按照32bit对齐，否则任务不被执行并行会触发配置异常中断。
3. 只支持一维搬移，并且软件需要保证RB的完整性，即：16bit待压缩数据必须是384bit的整数倍，8bit待解压缩数据必须是192bit的整数倍，4bit待解压缩数据必须是96bit的整数倍，否则任务不被执行并行会触发配置异常中断。
4. 压缩和定点转浮点不能同时使能，解压缩和浮点转定点不能同时使能，否则任务不被执行并行会触发配置异常中断。
5. 4-16bit解压缩，a_count[15:14]必须为0，8-16bit和4-8bit解压缩，a_count[15]必须为0，否则任务不被执行并行会触发配置异常中断。

16.9 Group ID 设置

16.9.1 场景描述

系统中采用ID来标识Master（QDMAC即被称为Master）的身份，根据ID的不同可以对不同的Master开放不同的访问权限，但是由于系统中的Master数量越来越多，导致MPU（Memory Protect Unit）对ID的识别逻辑增加，同时也不便于软件管理，考虑到很多Master具备相同的访问权限，因此，采用Group ID来将具有相同访问权限的Master归为一个Group进行管理。



16.9.2 配置流程

Group ID配置流程：

- 步骤1** 完成QUEUE的初始化，请参见“配置流程”的 [步骤1](#)、“配置流程”的 [步骤2](#)、“配置流程”的 [步骤3](#)。
- 步骤2** 配置寄存器QDMAC_QUEUE_CHG_IDX qn 的bit[1:0]完成第 qn 个Queue通道组选择配置，该寄存器的默认值为0，即默认选择通道组0；例如写QDMAC_QUEUE_CHG_IDX1[1:0]为0x01，QUEUE1配置为属于通道组1。

表 16-43 QDMAC_QUEUE_CHG_IDX 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_CHG_IDX qn : 偏移地址0x00290 + 0x4× qn , $qn=0 \sim 15$		
[31:2]	保留。	-
[1:0]	Queue qn 的通道组选择配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x01

- 步骤3** 配置寄存器QDMAC_CHG_GRPID gn 的bit[12:8]和bit[4:0]完成第 gn 个通道组的写通道和读通道的Group ID配置；例如写QDMAC_CHG_GRPID1的值为0x208，通道组1的写通道Group ID为0x02，读通道Group ID为0x08。

表 16-44 QDMAC_CHG_GRPID 寄存器

Bits	寄存器说明	配置值
QDMAC_CHG_GRPID qn : 偏移地址0x00204 + 0x4× qn , $qn=0 \sim 15$		
[31:13]	保留。	-
[12:8]	通道组 gn 的写命令通道Group ID配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x02
[7:5]	保留。	-
[4:0]	通道组 gn 的读命令通道Group ID配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x08

- 步骤4** 锁定写保护，请参见“配置流程”的 [步骤2](#)。

----结束

16.9.3 配置约束



须知

配置通道组Group ID一定要在对应通道组空闲状态下进行配置，否则无法确定哪些BD任务会使用新的Group ID，建议在模块初始化时完成配置。

映射寄存器QDMAC_QUEUE_CHG_IDX必须在对应的Queue处于IDLE，且原始完成中断和原始告警中断全部清除以后才能配置，否则配置不成功并且上报pslverr。

16.10 QoS 设置

16.10.1 场景描述

为了不同业务下可以调整系统各部分的带宽比重，提供了QoS寄存器供软件配置，用于调整QDMAC总线接口在系统中获得响应的延时。

16.10.2 配置流程

QoS配置流程：

- 步骤1** 解除写保护，请参见“配置流程”的步骤1。
- 步骤2** 查询全局状态寄存器QDMAC_STAT bit[4]为0b0，表示QDMAC的所有任务均已完成且产生的所有中断均已清除，如表16-45所示。

表 16-45 QDMAC_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_STAT：偏移地址0x324		
[31:5]	保留。	-
[4]	QDMAC的忙闲指示信号。 0b0：QDMAC处于闲状态； 0b1：QDMAC处于忙状态。	0b0
[3:1]	保留。	-
[0]	Memory的低功耗状态指示。 0b0：Memory处于Light Sleep状态； 0b1：Memory未进入Light Sleep状态。	-

- 步骤3** 配置寄存器QDMAC_QOS的bit[5:4]和bit[1:0]完成第mn个Master总线接口写通道和读通道的QoS配置；例如写通道QoS为0x2，读通道QoS为0x1，如表16-46所示：

表 16-46 QDMAC_QOS 寄存器

Bits	寄存器说明	配置值
QDMAC_QOS：偏移地址0x334		



Bits	寄存器说明	配置值
[31:6]	保留。	-
[5: 4]	写通道的QoS配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x2
[3:2]	保留。	-
[1:0]	读通道的QoS配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x1

步骤4 锁定写保护，请参见“配置流程”的步骤2。

----结束

16.10.3 配置约束

须知

配置QoS一定要在QDMAC空闲状态下进行配置，否则可能导致异常，建议在模块初始化时完成配置。

16.11 USER 设置

16.11.1 场景描述

为了不同业务下可以调整系统功能，提供了USER寄存器供软件配置。

16.11.2 配置流程

USER配置流程：

步骤1 具体寄存器含义见《Hi309a 寄存器手册》的SC_DMAC_USER_CFG0和SC_DMAC_USER_CFG1两个寄存器。

图 16-17 SC_DMAC_USER_CFG0 和 SC_DMAC_USER_CFG1 寄存器

SC_DMAC_USER_CFG0	DMAC的USER配置0	0x4300	32		0x40000000	dmac_user_po	31	BF	0x0	Reserved: 保留的DMAC User的Buffer = 0: 使用DMAC的Buffer 1: 不使用DMAC的Buffer
SC_DMAC_USER_CFG1	DMAC的USER配置1	0x4304	32		0x00000001	reserved	31:20	-	0x0	Reserved

----结束

16.11.3 配置约束



须知

配置USER一定要在QDMAC空闲状态下进行配置，否则可能导致异常，建议在模块初始化时完成配置。

16.12 QDMA AXI 信号输出路径选择

16.12.1 场景描述

为了不同业务下可以调整系统功能，QDMA0 AXI信号可选输出至SCHEDULER，或输出至TRNG0。QDMA1 AXI信号固定输出至TRING1。

16.12.2 配置流程

QDMA0 AXI信号输出选择配置流程：

- 步骤1** 具体寄存器含义见《Hi309a寄存器手册》的SC_DMAC0_AXI_SEL寄存器。如果选择TRNG0，需要确保TRING0处于解复位。
- 步骤2** 配置SC_DMAC0_USER_CFG0寄存器的dmac0_user_no。如果**步骤1**选择TRING，则配置为不使用SMMU的Reorder Buffer，如果选择schedule，则配置为使用SMMU的Reorder Buffer。

QDMA1 AXI信号固定输出至TRING1（需要确保TRING1处于解复位）。

- 步骤3** 配置SC_DMAC1_USER_CFG0寄存器的dmac0_user_no，配置为不使用SMMU的Reorder Buffer。

----结束

16.12.3 配置约束

须知

配置AXI信号输出选择一定要在QDMAC空闲状态下进行配置，否则可能导致异常，建议在模块初始化时完成配置。

16.13 Outstanding 设置

16.13.1 场景描述

为了调试QDMAC总线接口的性能，QDMAC提供了Outstanding控制寄存器，用于控制QDMAC发出的总线命令个数。

16.13.2 配置流程

Outstanding配置流程：



- 步骤1 “配置流程”的步骤2。
- 步骤2 配置寄存器QDMAC_OSD_CTRL的bit[11:8]和bit[3:0]分别完成读通道和写通道Outstanding配置，例如配置读通道Outstanding配置为8个，写通道Outstanding配置为8个，如表16-47所示。

表 16-47 QDMAC_OSD_CTRL 寄存器

Bits	寄存器说明	配置值
QDMAC_OSD_CTRL：偏移地址0x30C		
[31:12]	保留。	-
[11:8]	读通道最大的outstanding配置： 0000：1个Outstanding； 0001：2个Outstanding； 0010：3个Outstanding； 0011：4个Outstanding； 0100：5个Outstanding； 0101：6个Outstanding； 0110：7个Outstanding； 0111~1111：8个Outstanding； 注意：配置小于7的值读出值为配置值，否则读出值为7。 受Lock写锁保护和COMM VMID保护。	0x7
[7:4]	保留。	-
[3:0]	写通道最大的outstanding配置： 0000：1个Outstanding； 0001：2个Outstanding； 0010：3个Outstanding； 0011：4个Outstanding； 0100：5个Outstanding； 0101：6个Outstanding； 0110：7个Outstanding； 0111~1111：8个Outstanding； 注意：配置小于7的值读出值为配置值，否则读出值为7。 受Lock写锁保护和COMM VMID保护。	0x7

- 步骤3 锁定写保护，请参见“配置流程”的步骤2。
- 结束

16.13.3 配置约束



须知

- 配置的Outstanding数不能大于8个，即配置值不能大于0x7，建议维持配置为0x7不变。
- 必须在QDMAC空闲时才能配置，否则会导致异常，建议在模块初始化时完成配置。

16.14 读命令间隔配置

16.14.1 场景描述

控制读命令发送间隔的目的在于减少系统汇聚点的命令累积，从而减少高优先级Master命令在汇聚点的响应时间。

16.14.2 配置流程

读命令间隔配置流程：

- 步骤1** 解除写保护，请参见“配置流程”的**步骤1**。
- 步骤2** 配置寄存器QDMAC_GLB_EN的bit[3]的值为0x1完成AXI读名利间隔的使能，该bit默认值为0x1，其他bit的值保持不变；如表16-48所示。

表 16-48 QDMAC_GLB_EN 寄存器

Bits	寄存器说明	配置值
QDMAC_GLB_EN：偏移地址0x314		
[31:4]	保留。	0x000_0000
[3]	AXI读命令间隔使能。 写： 0：未使能，AXI读命令连续发出； 1：使能，AXI读命令会间隔arlen发出。 受Lock写锁保护和COMM VMID保护。	0x1
[2]	QDMAC VMID检查使能。 写： 0：未使能，不检查VMID访问的合法性； 1：使能，对各类参数的APB写访问进行VMID合法性检查。 受Lock写锁保护和COMM VMID保护。	-



Bits	寄存器说明	配置值
[1]	QDMAC告警消息发送使能。 写： 0：未使能，如果BD发生了告警事件，即使消息使能也不会发送告警消息； 1：使能，如果BD发生了告警事件，且消息使能，则发送告警消息。 受Lock写锁保护和COMM VMID保护。	-
[0]	QDMAC处理能力使能寄存器。 写： 0：未使能（如果是撤销使能，则处理能力隔离功能立即失效，各通道组的累计使用时间清0）； 1：使能； 受Lock写锁保护和COMM VMID保护。	-

步骤3 锁定写保护，请参见“[配置流程](#)”的**步骤2**。

----结束

16.14.3 配置约束

无

16.15 处理能力隔离配置

16.15.1 场景描述

软件按照业务需求将DMA的Queue分成多个通道组，多个通道组共享DMA的Master带宽，但是各个通道组对应的业务处理能力需求各异，为了保证各个通道组之间的带宽比例，软件对各个通道组进行了带宽限制，在多个通道组竞争的时候，如果参与竞争的通道组已经达到了带宽限制条件，则对应通道组的任务不能被调度。

16.15.2 配置流程

处理能力隔离配置流程：

步骤1 完成QUEUE的初始化，请参见“[配置流程](#)”的**步骤1**、“[配置流程](#)”的**步骤2**、“[配置流程](#)”的**步骤3**。

步骤2 配置寄存器QDMAC_QUEUE_CHG_IDX qn 的bit[1:0]完成第 qn 个Queue通道组选择配置，该寄存器的默认值为0，即默认选择通道组0；例如写QDMAC_QUEUE_CHG_IDX1[1:0]为0x01，QUEUE1配置为属于通道组1，如[表16-49](#)所示。



表 16-49 QDMAC_QUEUE_CHG_IDX 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_CHG_IDX qn : 偏移地址 $0x00290 + 0x4 \times qn$, $qn=0 \sim 15$		
[31:2]	保留。	-
[1:0]	Queue qn 的通道组选择配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x01

步骤3 配置寄存器QDMAC_CHG_RATE gn 的bit[8:0]完成第 gn 个通道组的处理能力配比的配置，该寄存器的默认值为0；例如通道组之间的流量配比CHG0: CHG1: CHG2: CHG3=2: 4: 8: 16，需要配置QDMAC_CHG_RATE0的值为 $512/2=256$ ，QDMAC_CHG_RATE1的值为 $512/4=128$ ，QDMAC_CHG_RATE2的值为 $512/8=64$ ，QDMAC_CHG_RATE3的值为 $512/16=32$ ；QDMAC_CHG_RATE gn 的描述如表16-50所示。

表 16-50 QDMAC_CHG_RATE 寄存器

Bits	寄存器说明	配置值
QDMAC_CHG_RATE gn : 偏移地址 $0x00210 + 0x20 \times gn$ ($gn = 0 \sim 3$)		
[31:9]	保留。	-
[8:0]	分配给通道组 gn 的DMA搬移能力比率，根据通道组 gn 的流量配比的倒数乘以512得到。 注意：如果计算出来的值为512，则只能配置为511；如果配置为0，通道组不受执行时间的影响，优先级一直为最高。 受Lock写锁保护和COMM VMID保护。	0x001 ~ 0x1FF

步骤4 配置寄存器QDMAC_CHG_MAXTIME gn 的bit[31:0]完成第 gn 个通道组的最大使用时间的配置，该寄存器的默认值为0；QDMAC_CHG_MAXTIME gn 的描述如表16-51所示。

表 16-51 QDMAC_CHG_MAXTIME 寄存器

Bits	寄存器说明	配置值
QDMAC_CHG_MAXTIME gn : 偏移地址 $0x00214 + 0x20 \times gn$ ($gn = 0 \sim 3$)		
[31:0]	通道组 gn 的最大使用时间配置寄存器，在处理能力隔离中，如果通道组 gn 的累积使用时间达到或者超过该值，则所有通道组的累计使用时间清0。通道组 gn 的累计使用时间=实际累计使用时间*QDMAC_CHG_RATE。 受Lock写锁保护和COMM VMID保护。	0x0000_0001 ~ 0xFFFF_FFFF

步骤5 配置寄存器QDMAC_EXEC_THLD的bit[5:0]完成处理能力隔离中任务执行时间累加单位的配置；例如配置QDMAC_EXEC_THLD为0x01，则任务执行过程中每2个cycle执行时间累加一次；QDMAC_EXEC_THLD的描述如表16-52所示。



表 16-52 QDMAC_EXEC_THLD 寄存器

Bits	寄存器说明	配置值
QDMAC_EXEC_THLD：偏移地址0x00308		
[31:6]	保留。	-
[5:0]	处理能力隔离中任务执行时间累加单位。 000000：1个cycle加一次； 000001：2个cycle加一次； ... 111111：64个cycle加一次。 受Lock写锁保护和COMM VMID保护。	0x01

步骤6 配置寄存器QDMAC_GLB_EN的bit[0]的值为0x1完成处理能力隔离的使能，该bit的默认值为0x0，其他bit的值保持不变；QDMAC_GLB_EN的描述如表16-53所示。

表 16-53 QDMAC_GLB_EN 寄存器

Bits	寄存器说明	配置值
QDMAC_EXEC_THLD：偏移地址0x00308		
[31:4]	保留。	0x000_0000
[3]	AXI读命令间隔使能。 写： 0：未使能，AXI读命令连续发出； 1：使能，AXI读命令会间隔arlen发出。 受Lock写锁保护和COMM VMID保护。	-
[2]	QDMAC VMID检查使能。 写： 0：未使能，不检查VMID访问的合法性； 1：使能，对各类参数的APB写访问进行VMID合法性检查。 受Lock写锁保护和COMM VMID保护。	-
[1]	QDMAC告警消息发送使能。 写： 0：未使能，如果BD发生了告警事件，即使消息使能也不会发送告警消息； 1：使能，如果BD发生了告警事件，且消息使能，则发送告警消息。 受Lock写锁保护和COMM VMID保护。	-



Bits	寄存器说明	配置值
[0]	QDMAC处理能力使能寄存器。 写： 0：未使能（如果是撤销使能，则处理能力隔离功能立即失效，各通道组的累计使用时间清0）； 1：使能； 受Lock写锁保护和COMM VMID保护。	0x01

步骤7 锁定写保护，请参见“配置流程”的**步骤2**。

----结束

16.15.3 配置约束

须知

QDMAC_CHG_RATE qn 的配置值最大为511，如果通道组 gn 的流量配比的倒数乘以512得到的值为512，需要按照511去配置QDMAC_CHG_RATE qn 寄存器；

映射寄存器QDMAC_QUEUE_CHG_IDX必须在对应的Queue处于IDLE，且原始完成中断和原始告警中断全部清除以后才能配置，否则配置不成功并且上报pslverr。

16.16 Stream ID 设置

16.16.1 场景描述

系统中采用Stream ID来标识Master（QDMAC即被称为Master）的身份，根据ID的不同可以对不同的Master开放不同的访问权限。

16.16.2 配置流程

Stream ID配置流程：

- 步骤1** 完成QUEUE的初始化，请参见配置流程的**步骤1**、配置流程的**步骤2**、配置流程的**步骤3**。
- 步骤2** 配置寄存器QDMAC_QUEUE_CHG_IDX qn 的bit[1:0]完成第 qn 个Queue通道组选择配置，该寄存器的默认值为0，即默认选择通道组0；例如写QDMAC_QUEUE_CHG_IDX1[1:0]为0x01，QUEUE1配置为属于通道组1，如表16-54所示。



表 16-54 QDMAC_QUEUE_CHG_IDX 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_CHG_IDX qn : 偏移地址0x00290 + 0x4× qn , $qn=0 \sim 15$		
[31:2]	保留。	-
[1:0]	Queue qn 的通道组选择配置寄存器。 受Lock写锁保护和COMM VMID保护。	0x01

步骤3 配置寄存器QDMAC_CHG_STREAMID gn 的bit[23:0]完成第 gn 个通道组的Stream ID配置；例如写QDMAC_CHG_STREAMID1的值为0x208，通道组1的Stream ID为0x208，如表16-55所示。

表 16-55 QDMAC_CHG_STREAMID 寄存器

Bits	寄存器说明	配置值
QDMAC_CHG_STREAMID gn : 偏移地址0x00208 + 0x4× gn , $gn=0 \sim 15$		
[31:24]	-	reserved
[23:0]	RW	chg_streamid

步骤4 锁定写保护，请参见“配置流程的步骤2”。

----结束

16.16.3 配置约束

须知

配置通道组Group ID一定要在对应通道组空闲状态下进行配置，否则无法确定哪些BD任务会使用新的Group ID，建议在模块初始化时完成配置；
映射寄存器QDMAC_QUEUE_CHG_IDX必须在对应的Queue处于IDLE，且原始完成中断和原始告警中断全部清除以后才能配置，否则配置不成功并且上报pslverr。

16.17 IP 可维可测

无。

16.18 Queue 的任务查询

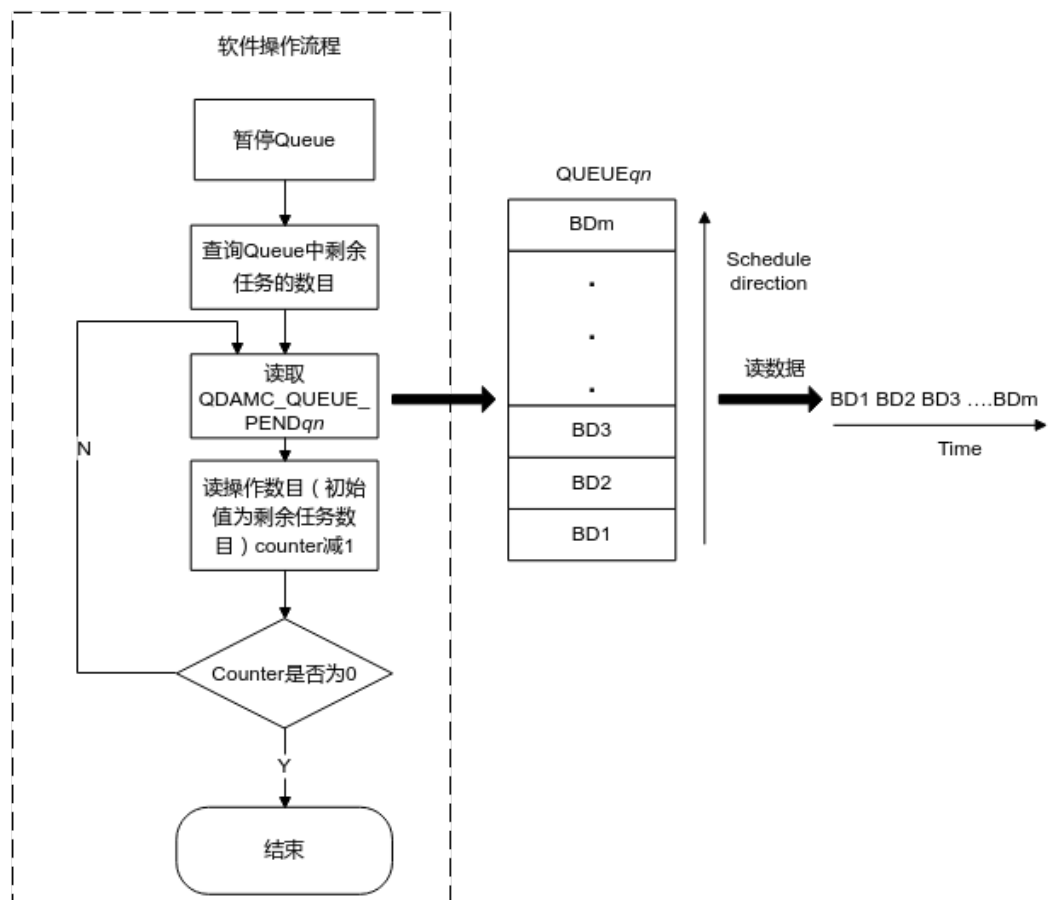
16.18.1 场景描述

QDMAC为每个Queue提供了一套查询机制，可以获取Queue的状态，包括Queue中剩余的空间，Queue中剩余的任务（还未被执行的任务的BD指针）。

对于Queue中剩余任务的查询，QDMAC为每个Queue只提供了一个只读寄存器，软件需要查询到Queue中剩余任务的数目m，然后读取m次该只读寄存器，第一次读出Queue中排在最前面的任务，第二次读出Queue中排在次前面的任务，依次类推。

图16-18读取剩余任务的操作流程示意图，假设Queue qn 中的任务全部未被调度，即剩余任务数为m：

图 16-18 剩余任务查询流程图



16.18.2 配置流程

Queue的任务查询流程：

步骤1 解除写保护，请参见“[配置流程](#)”的**步骤1**。

步骤2 配置Queue暂停寄存器QDMAC_QUEUE_PAUSE qn 的bit[0]为0b1，暂停Queue qn ；例如软件需要暂停Queue1，配置0x0000_0001到QDMAC_QUEUE_PAUSE1（偏移地址0x46200），如[表16-56](#)所示。



表 16-56 QDMAC_QUEUE_PAUSE 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_PAUSE qn : 偏移地址 $0x45200 + 0x1000 \times qn$ ($qn = 0 \sim 15$)		
[31:1]	保留。	-
[0]	第 qn 个Queue暂停控制。 写: 0: 撤销暂停, Queue继续工作; 1: 暂停, Queue停止工作。 读: 0: Queue处于运行状态。 1: Queue已经暂停; 受Queue q QDMAC_QUEUE_STAT寄存器 n 的VMID保护。	0b1

步骤3 查询Queue的状态寄存器QDMAC_QUEUE_STAT qn 寄存器的bit[9:0]获取Queue qn 中剩余的空间, 然后用Queue qn 的深度减去剩余的空间得到剩余的任务数, Queue的深度计算见[场景描述](#); 例如软件需要计算Queue1中剩余的任务, 则先查询QDMAC_QUEUE_STAT1 (偏移地址 $0x46208$) 寄存器的bit[9:0], 获取Queue1剩余的空间 $0x001$ (假设还有一个剩余任务空间), 假设Queue的深度为128, 则剩余任务为 $128-1=127$, 如[表16-57](#)所示。

表 16-57 QDMAC_QUEUE_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_STAT qn : 偏移地址 $0x45208 + 0x1000 \times qn$ ($qn = 0 \sim 15$)		
[31:13]	保留。	-
[12]	第 qn 个Queue中的所有任务是否都已执行完毕状态指示。 0: Queue中的所有任务均已执行完毕; 1: Queue中的任务还未执行完毕。	-
[11:10]	保留。	-
[9:0]	第 qn 个Queue中空闲的任务通道数目。 0x000: Queue中无空闲通道; 0x001: Queue中有1个空闲通道; ... 0x200: Queue中有512个空闲通道。 注意: 初始化后, 还未配置Queue的个数, 则默认为一个Queue, Queue空闲的通道数目为总的任务数目, 如果配置了Queue的个数, 则根据个数平分总的任务数目。	0x001 (还有一个剩余任务空间)



- 步骤4** 判断剩余任务数是否为0，不为0则用当前Queue中剩余任务数初始化查询次数，执行**步骤5**，否则退出流程，根据需要执行**步骤7**。
- 步骤5** 查询QDMAC_QUEUE_PEND qn ，获取剩余任务，如**表16-58**所示。

表 16-58 QDMAC_QUEUE_PEND 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_PEND qn : 偏移地址0x4520C + 0x1000× qn （ $qn = 0 \sim 15$ ）		
[31:0]	第 qn 个Queue的剩余任务查询寄存器，读取该寄存器会依次得到Queue中还未被执行的BD。 对于普通BD返回BD地址[35:6]，低位为全0； 注意：该地址必须在QDMAC_QUEUE_PAUSE配置为暂停时才能读取，否则返回为0，且上报APB总线访问错误（pslverr=1）。查询次数达到剩余任务个数以后再读取该寄存器，重新从第一个剩余任务开始返回。	-

- 步骤6** 查询次数减一，判断查询次数是否为0，不为0则执行**步骤5**继续查询，否则结束查询退出流程，根据需要执行**步骤7**。
- 步骤7** 配置Queue暂停寄存器QDMAC_QUEUE_PAUSE qn 的bit[0]为0b0，解除Queue qn 暂停；例如软件需要解除Queue1暂停，配置0x0000_0000到QDMAC_QUEUE_PAUSE1（偏移地址0x46200），如**表16-59**所示。

表 16-59 QDMAC_QUEUE_PAUSE 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_PAUSE qn : 偏移地址0x45200 + 0x1000× qn （ $qn = 0 \sim 15$ ）		
[31:1]	保留。	-
[0]	第 qn 个Queue暂停控制。 写： 0：撤销暂停，Queue继续工作； 1：暂停，Queue停止工作。 读： 0：Queue处于运行状态； 1：Queue已经暂停。 受Queue qn 的VMID保护	0b0

- 步骤8** 锁定写保护，请参见“**配置流程**的**步骤2**”。
- 结束



16.18.3 配置约束

须知

1. 查询剩余的任务必须在Queue处于暂停时才能够查询，否则会导致异常。
2. 剩余任务查询寄存器必须在读出的剩余任务数不为0时才能够读取，否则会导致异常。
3. 查询不会导致任务撤销，解除暂停以后任务可以继续被调度执行。
4. 如果读取剩余任务查询寄存器的次数超过了剩余任务数，查询到的任务会重新从第一个未被执行的任务开始循环，建议查询的次数按照图16-18约束执行。

16.19 Queue 的任务撤销

16.19.1 场景描述

QDMAC为每个Queue提供一个任务撤销寄存器，该寄存器用于撤销Queue中还未被调度的所有任务，已经调度出Queue的任务不会被撤销，被撤销的任务不会被执行，任务撤销以后Queue处于空的状态。

16.19.2 配置流程

Queue的任务撤销流程：

- 步骤1** 解除写保护，见“配置流程的步骤1”。
- 步骤2** 配置Queue暂停寄存器QDMAC_QUEUE_PAUSE qn 的bit[0]为0b1，暂停Queue qn ；例如软件需要暂停Queue1，配置0x0000_0001到QDMAC_QUEUE_PAUSE1（偏移地址0x46200），如表16-56所示。
- 步骤3** 配置Queue任务撤销寄存器QDMAC_QUEUE_CANCEL qn （偏移地址0x45204 + 0x1000 $\times$ qn ）的bit0为0b1，撤销Queue qn 中的任务，然后根据需要选择是否执行步骤4；例如软件需要撤销Queue1的任务，配置0x0000_0001到QDMAC_QUEUE_CANCEL1（偏移地址0x46204）。

表 16-60 QDMAC_QUEUE_CANCEL 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_CANCEL qn : 偏移地址0x45204 + 0x1000 $\times$ qn （ $qn = 0 \sim 15$ ）		
[31:1]	保留。	-
[0]	第 qn 个Queue任务撤销。 写： 0：对Queue的任务无影响； 1：撤销Queue中还未执行的任务。 注意：读无意义，且读出值为0。 受Queue qn 的VMID保护。	0b1



步骤4 配置Queue暂停寄存器QDMAC_QUEUE_PAUSE qn 的bit[0]为0b0，解除Queue qn 暂停；例如软件需要解除Queue1暂停，配置0x0000_0000到QDMAC_QUEUE_PAUSE1（偏移地址0x46204），如表16-59所示。

步骤5 锁定写保护，请参见“配置流程”的步骤2。

----结束

16.19.3 配置约束

须知

撤销Queue中的任务必须在Queue处于暂停时才能够撤销，否则会导致异常。

16.20 QDMAC 的状态查询

16.20.1 场景描述

QDMAC提供了内部工作状态的查询寄存器，方便软件在需要时了解QDMAC内部状态，包括：

- Queue的忙闲状态：配置到Queue的所有任务数据搬移都执行完毕以后，Queue为空闲，否则为忙；
- QDMAC的忙闲状态：所有Queue的所有任务数据搬移都执行完毕，并且，QDMAC产生的所有中断都已清除，QDMAC为空闲，否则为忙；
- Memory的低功耗状态：Memory自动进入Light Sleep以后，Memory为Light Sleep状态，否则为Normal状态；
- BD临终遗言：任务执行时BD会记录在QDMAC内部，如果出现了QDMAC任务无法完成的情况，可以查询任务的BD信息，方便定位。

16.20.2 配置流程

- Queue的忙闲状态查询：

步骤1 查询Queue的状态寄存器QDMAC_QUEUE_STAT qn 的bit[12]为0b0表示Queue qn 为空闲，即Queue qn 的所有任务均已执行完毕，为1则表示Queue qn 为忙，即Queue qn 还有任务未完成；例如软件需要获取Queue1的任务是否已经全部完成，则查询寄存器QDMAC_QUEUE_STAT1（偏移地址0x46208），判断bit[12]是否为0b0则可获取Queue1是否空闲，如表16-61所示。

表 16-61 QDMAC_QUEUE_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_STAT qn : 偏移地址0x45208 + 0x1000× qn （ $qn = 0 \sim 15$ ）		
[31:13]	保留。	-



Bits	寄存器说明	查询值
[12]	第qn个Queue中的所有任务是否都已执行完毕状态指示。 0: Queue中的所有任务均已执行完毕; 1: Queue中的任务还未执行完毕。	0b1/0b0
[11:9]	保留。	-
[8:0]	第qn个Queue中空闲的任务通道数目。 0xh000: Queue中无空闲通道; 0xh001: Queue中有1个空闲通道; ... 0xh200: Queue中有512个空闲通道。 注意: 初始化后, 还未配置Queue的个数, 则默认为一个Queue, Queue空闲的通道数目为总的任务数目, 如果配置了Queue的个数, 则根据个数平分总的任务数目。	-

须知

Queue的空状态并不代表Queue空闲, 空状态即Queue的剩余空间等于Queue的深度。

----结束

- QDMAC的忙闲状态查询:

步骤1 查询QDMAC的状态寄存器QDMAC_STAT的bit[4]为0b0表示QDMAC为空闲, 即所有Queue的所有任务均已执行完毕并且所有中断都已清除, 为0b1则表示QDMAC为忙, 如表16-62所示。

表 16-62 QDMAC_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_STAT: 偏移地址0x324		
[31:5]	保留。	-
[4]	DMAC的忙闲指示信号。 0: DMAC处于闲状态; 1: DMAC处于忙状态。	0b0/0b1
[3:1]	保留。	-
[0]	Memory的低功耗状态指示: 0: Memory处于Light Sleep状态; 1: Memory未进入Light Sleep状态。	-



----结束

- Memory的低功耗状态查询：

步骤1 查询QDMAC的状态寄存器QDMAC_STAT的bit[0]为0b0表示QDMAC的Memory处于Light Sleep状态，即QDMAC在所有任务执行完毕以后进入了Light Sleep状态，为0b1则表示QDMAC的Memory处于Normal状态，表示Memory处于正常访问状态，如表16-61所示。

表 16-63 QDMAC_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_STAT：偏移地址0x324		
[31:5]	保留。	-
[4]	DMAC的忙闲指示信号。 0b0：DMAC处于闲状态； 0b1：DMAC处于忙状态。	-
[3:1]	保留。	-
[0]	Memory的低功耗状态指示： 0b0：Memory处于Light Sleep状态； 0b1：Memory未进入Light Sleep状态。	0b0/0b1

----结束

- BD临终遗言查询：

步骤1 查询QDMAC_MIF_BD_WORD0 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD CONFIG域的配置信息，如表16-64所示。

表 16-64 QDMAC_MIF_BD_WORD0 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD0 $mncn$ ：偏移地址 $0x800+0x80mn+0x40cn$ ， $mn=0$ ， $cn=0\sim1$		
[31]	BD Word0的软件配置信息：CONFIG域的配置信息	0x0000_0000~0xFFFF_FFFF

步骤2 查询QDMAC_MIF_BD_WORD1 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD DST_ADDR域的配置信息，如表16-65所示。



表 16-65 QDMAC_MIF_BD_WORD1 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD1 <i>mncn</i> : 偏移地址 $0x804+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word1的软件配置信息: DST_ADDR域的配置信息	32bit系统地址

步骤3 查询QDMAC_MIF_BD_WORD2 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD SRC_ADDR域的配置信息, 如表16-66所示。

表 16-66 QDMAC_MIF_BD_WORD2 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD2 <i>mncn</i> : 偏移地址 $0x808+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word1的软件配置信息: DST_ADDR域的配置信息	32bit系统地址

步骤4 查询QDMAC_MIF_BD_WORD3 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD B_COUNT/A_COUNT域的配置信息, 如表16-67所示。

表 16-67 QDMAC_MIF_BD_WORD3 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD3 <i>mncn</i> : 偏移地址 $0x80C+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word3的软件配置信息: CNT0域的配置信息	0x0000_0000~0xFFFF_FFFF

步骤5 查询QDMAC_MIF_BD_WORD4 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD INIT_DATA/C_COUNT域的配置信息, 如表16-68所示。

表 16-68 QDMAC_MIF_BD_WORD4 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD4 <i>mncn</i> : 偏移地址 $0x810+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word4的软件配置信息: CNT1的配置信息	0x0000_0000~0xFFFF_FFFF

步骤6 查询QDMAC_MIF_BD_WORD5 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD BINDX域的配置信息, 如表16-69所示。



表 16-69 QDMAC_MIF_BD_WORD5 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD5 <i>mncn</i> : 偏移地址 $0x814+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word5的软件配置信息: BINDX的配置信息	0x0000_0000~0xFFFF_FFFF

步骤7 查询QDMAC_MIF_BD_WORD6 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD CINDX域的配置信息, 如表16-70所示。

表 16-70 QDMAC_MIF_BD_WORD6 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD6 <i>mncn</i> : 偏移地址 $0x818+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word6的软件配置信息: CINDX的配置信息	0x0000_0000~0xFFFF_FFFF

步骤8 查询QDMAC_MIF_BD_WORD7 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD LLI域的配置信息, 如表16-71所示。

表 16-71 QDMAC_MIF_BD_WORD7 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD7 <i>mncn</i> : 偏移地址 $0x81C+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word7的软件配置信息: LLI的配置信息	0x0000_0000~0xFFFF_FFFF

步骤9 查询QDMAC_MIF_BD_WORD8 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD RESERVE域的配置信息, 如表16-72所示。

表 16-72 QDMAC_MIF_BD_WORD8 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD8 <i>mncn</i> : 偏移地址 $0x81C+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word8的软件配置信息: 暂时预留	0x0000_0000~0xFFFF_FFFF

步骤10 查询QDMAC_MIF_BD_WORD9 *mncn* 获取第 *mn* 个Master接口的第 *cn* 个物理通道的BD ALG域的配置信息, 如表16-73所示。



表 16-73 QDMAC_MIF_BD_WORD9 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD9 $mncn$: 偏移地址 $0x824+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word9的软件配置信息：算法的配置信息	0x0000_0000~0xFFFF_FFFF

步骤11 查询QDMAC_MIF_BD_WORD10 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD RESERVE域的配置信息，如表16-74所示。

表 16-74 QDMAC_MIF_BD_WORD10 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD10 $mncn$: 偏移地址 $0x82C+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:12]	BD Word10的软件配置信息：暂时预留	0x0_0000~0xF_FFFF
[11:8]	msg_addr的高4bit地址	4bit高位地址
[7:4]	src_addr的高4bit地址	4bit高位地址
[3:0]	dst_addr的高4bit地址	4bit高位地址

步骤12 查询QDMAC_MIF_BD_WORD11 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD MSG_ADDR域的配置信息，如表16-75所示。

表 16-75 QDMAC_MIF_BD_WORD11 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD11 $mncn$: 偏移地址 $0x82C+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		
[31:0]	BD Word11的软件配置信息：MSG_ADDR的配置信息	32bit系统地址

步骤13 查询QDMAC_MIF_BD_WORD12的软件配置信息：MSG_ADDR的配置信息 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD WORD12配置信息，具体描述如表16-76所示。

表 16-76 QDMAC_MIF_BD_WORD12 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD12 $mncn$: 偏移地址 $0x830+0x80mn+0x40cn$, $mn=0$, $cn=0\sim1$		



Bits	寄存器说明	查询值
[31:0]	BD Word12的软件配置信息： 普通BD：MSG_DATA0的配置信息；	0x0000_0000~0xFFFF_FFFF

步骤14 查询QDMAC_MIF_BD_WORD13 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD WORD13域的配置信息，如表16-77所示。

表 16-77 QDMAC_MIF_BD_WORD13 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD13 $mncn$ ：偏移地址 $0x834+0x80mn+0x40cn$ ， $mn=0$ ， $cn=0\sim1$		
[31:0]	BD Word13的软件配置信息： 普通BD：MSG_DATA1的配置信息；	0x0000_0000~0xFFFF_FFFF

步骤15 查询QDMAC_MIF_BD_WORD14 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD WORD14域的配置信息，如表16-78所示。

表 16-78 QDMAC_MIF_BD_WORD14 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD14 $mncn$ ：偏移地址 $0x838+0x80mn+0x40cn$ ， $mn=0$ ， $cn=0\sim1$		
[31:0]	BD Word14的软件配置信息： 普通BD：MSG_DATA2的配置信息；	0x0000_0000~0xFFFF_FFFF

步骤16 查询QDMAC_MIF_BD_WORD15 $mncn$ 获取第 mn 个Master接口的第 cn 个物理通道的BD WORD15域的配置信息，如表16-79所示。

表 16-79 QDMAC_MIF_BD_WORD15 寄存器

Bits	寄存器说明	查询值
QDMAC_MIF_BD_WORD15 $mncn$ ：偏移地址 $0x83C+0x80mn+0x40cn$ ， $mn=0$ ， $cn=0\sim1$		
[31:0]	BD Word15的软件配置信息： 普通BD：MSG_DATA3的配置信息；	0x0000_0000~0xFFFF_FFFF



须知

- 上面的所有步骤可以只选择其中的一步或者多步，并且查询顺序没有任何要求。
- 查询到的结果为QDMAC从BD指针处读取的参数并且解析以后的结果。
- 查询上述寄存器的值需要在QDMAC停止状态（异常停止、正常执行完毕、HALT状态）下进行才能起到Debug作用。
- 由于存在两个物理通道，软件是不清楚任务到底在哪个物理通道执行，因此Debug时需要读取两个物理通道的参数进行分析。

----结束

16.20.3 配置约束

无。

16.21 QDMAC 的 HALT 操作

16.21.1 场景描述

HALT功能即整个QDMAC的暂停功能，当QDMAC进入HALT状态后，不再执行数据搬移操作，系统可以对QDMAC进行单独复位，也可以解除HALT状态，QDMAC继续执行未完成的数据搬移任务。

16.21.2 配置流程

QDMAC的HALT配置流程：

- 步骤1** 解除写保护，请参见“配置流程”步骤1。
- 步骤2** 配置HALT寄存器QDMAC_HALT的bit[1]为0b1，发起HALT请求，如表16-80所示。

表 16-80 QDMAC_HALT 寄存器

Bits	寄存器说明	配置值
QDMAC_HALT：偏移地址0x318		
[31:2]	保留。	-
[1]	QDMAC halt请求寄存器。 写： 0：撤销halt或者解除halt状态； 1：发起halt请求； 受Lock写锁保护和COMM VMID保护。	0b1



Bits	寄存器说明	配置值
[0]	QDMAC HALT响应寄存器。 读： 0b1：QDMAC已经进入了HALT状态； 0b0：QDMAC未进入HALT状态。 注意：解除HALT后，该bit清零。	-

步骤3 查询HALT寄存器QDMAC_HALT的bit[0]，如表16-81所示。

表 16-81 QDMAC_HALT 寄存器

Bits	寄存器说明	查询值
QDMAC_HALT：偏移地址0x318		
[31:2]	保留。	-
[1]	QDMAC halt请求寄存器。 写： 1：发起halt请求； 0：撤销halt或者解除halt状态； 受Lock写锁保护和COMM VMID保护	0b1
[0]	QDMAC HALT响应寄存器。 读： 0b1：QDMAC已经进入了HALT状态； 0b0：QDMAC未进入HALT状态。 注意：解除HALT后，该bit清零。	0b0~0b1

步骤4 判断查询到的值是否为0b1，如果为0b1则表示QDMAC已经进入HALT状态，软件可以对QDMAC做出期望的操作，否则执行步骤3；

步骤5 配置HALT寄存器QDMAC_HALT（偏移地址0x318）的bit1为0，退出HALT状态，QDMAC继续正常运行，如表16-82所示：

表 16-82 QDMAC_HALT 寄存器

Bits	寄存器说明	配置值
QDMAC_HALT：偏移地址0x318		
[31:2]	保留。	-



Bits	寄存器说明	配置值
[1]	QDMAC halt请求寄存器。 写： 1：发起halt请求； 0：撤销halt或者解除halt状态； 受Lock写锁保护和COMM VMID保护	0b0
[0]	QDMAC HALT响应寄存器。 读： 0b1：QDMAC已经进入了HALT状态； 0b0：QDMAC未进入HALT状态。 注意：解除HALT后，该bit清零。	-

步骤6 锁定写保护，请参见“配置流程”的步骤2。

----结束

16.21.3 配置约束

说明

一旦执行了“配置流程”的步骤3，则必须等待QDMAC成功进入HALT状态以后才能执行“配置流程”的步骤5，否则会导致HALT功能异常。

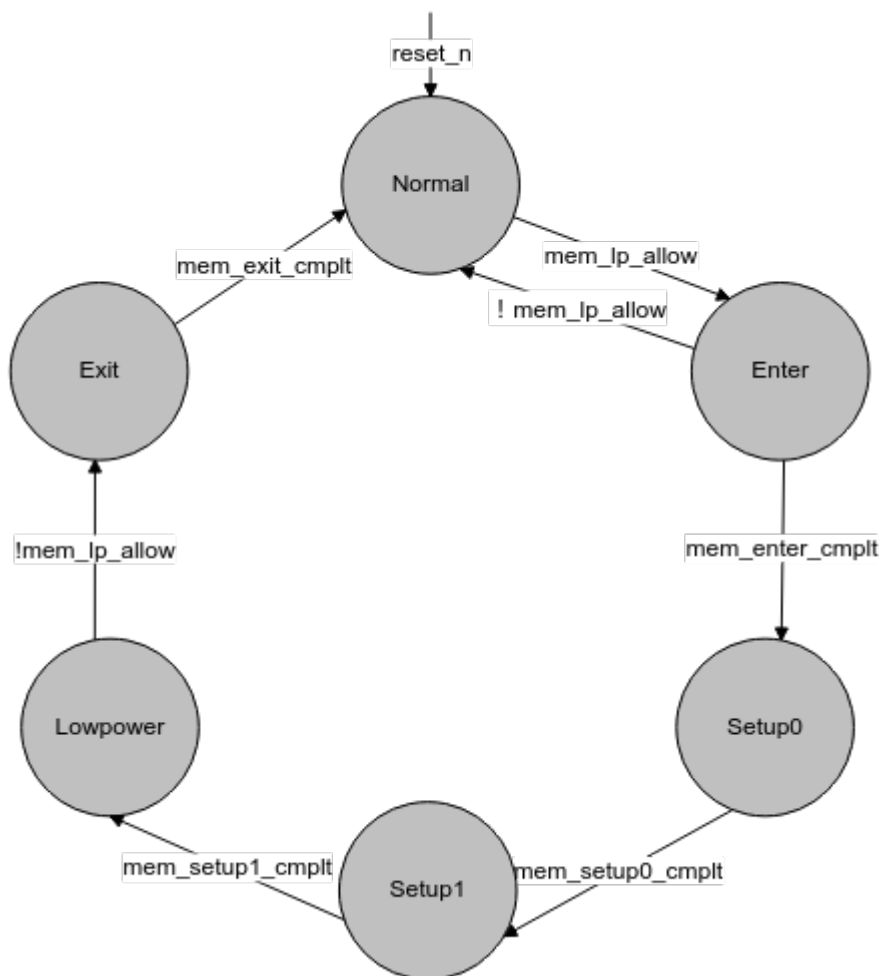
16.22 IP 低功耗

16.23 Memory 低功耗

16.23.1 场景描述

QDMAC内部包括BD指针存储RAM和数据Buffer RAM，为了降低功耗，QDMAC对这些RAM进行了自动Light Sleep控制，控制流程如图16-19所示。

图 16-19 Memory 低功耗处理流程图



- mem_lp_allow定义：Memory低功耗使能并且LPC未发起低功耗请求并且Memory无访问并且QDMAC的所有Queue空闲。当mem_lp_allow为0时即Memory退出低功耗的条件。
- mem_enter_cmplt定义：Memory低功耗准备阶段结束指示。
- mem_setup0_cmplt、mem_setup1_cmplt定义：Memory低功耗进入建立状态结束指示。
- mem_exit_cmplt定义：Memory低功耗退出阶段结束指示。

步骤1 QDMAC初始处于Normal状态。

步骤2 QDMAC完成所有任务数据搬移以后，判断mem_lp_allow是否允许进入低功耗，如果条件满足则进入Enter状态准备进入低功耗。

步骤3 在Enter状态下会进行计数，如果记到软件配置的阈值mem_lp_allow条件仍然满足，则进入Setup0状态开始建立Memory低功耗时序；如果在Enter状态下mem_lp_allow条件不满足，则立即回到Normal状态。

步骤4 进入Setup0、Setup1状态以后，Memory的低功耗时序就已经开始建立了，此过程不可逆，一旦进入以后就必须到达Memory低功耗状态。

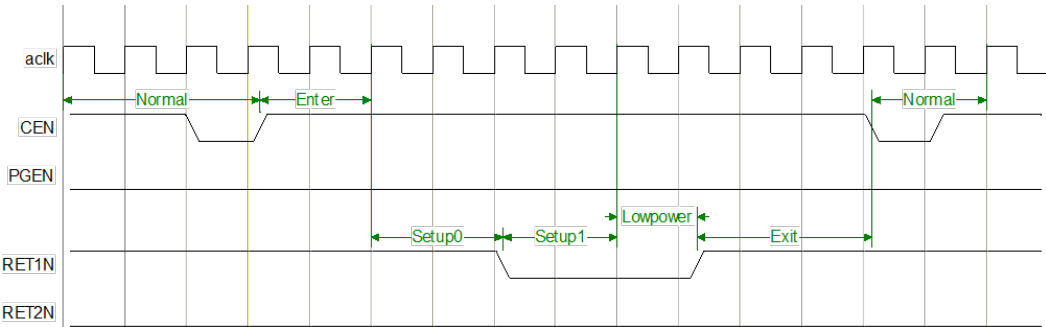


- 步骤5** 时序建立完毕以后进入Light Sleep状态，在该状态下Memory处于低功耗的Light Sleep状态，一直持续到mem_lp_allow条件不满足以后到达Exit状态。
- 步骤6** 在Exit状态下完成退出时序的建立以后回到Normal状态。

----结束

各阶段对应的Memory控制信号如图16-20所示。

图 16-20 Memory 低功耗控制时序图



- 图16-20中的信号解释如下：
- aclk：QDMAC的工作时钟信号。
 - CEN：Memory的片选信号。
 - PGEN：Memory的Power Gating使能信号。
 - RET1N：Memory的Retention使能信号1。
 - RET2N：Memory的Retention使能信号2。

16.23.2 配置流程

进入Memory低功耗流程：

- 步骤1** 解除写保护，请参见“配置流程”步骤1。
- 步骤2** 配置寄存器QDMAC_MEM_LP的bit[31:28]、bit[27:24]、bit[21:12]、bit[11:8]、bit[0]，分别完成Setup1、Setup0、Enter、Exit状态下计数阈值的配置以及使能的配置，计数器的计数单位为aclk；例如，需要在所有任务完成以后256个aclk cycle就进入Light Sleep，则Enter状态计数器的阈值为0xFF，Setup0、Setup1、Exit状态下的计数阈值与Memory的时序特性相关，假设都为2个aclk，则计数阈值为0x1，配置0x110FF101到QDMAC_MEM_LP表示QDMAC空闲以后可以自动控制Memory进入Light Sleep，如表16-83所示。

表 16-83 QDMAC_MEM_LP 寄存器

Bits	寄存器说明	配置值
QDMAC_MEM_LP：偏移地址0x31C		
[31:28]	Setup1状态的计数时间配置，用于控制Memory Light Sleep控制信号稳定时间。单位为工作时钟 cycle。	0x1



Bits	寄存器说明	配置值
[27:24]	Setup0状态的计数时间配置，用于控制模块空闲超时后（Enter状态后）到Memory控制低功耗信号之间的建立时间。单位为工作时钟cycle。	0x1
[23:22]	保留。	-
[21:12]	Enter状态的计数时间配置，用于控制QDMAC空闲以后，等待进入Memory Light Sleep的时间，即空闲以后在该时间内没有任何触发QDMAC继续工作的操作，QDMAC则开始控制Memory进入Light Sleep。单位为工作时钟cycle。	0xFF
[11:8]	Exit状态的计数时间配置，用于控制Memory从Light Sleep状态唤醒需要的时间，单位为工作时钟cycle。	0x1
[7:1]	保留。	-
[0]	Memory Light Sleep使能控制。 0：QDMAC不会进行Memory Light Sleep控制； 1：QDMAC自动进行Memory Light Sleep控制。	0b1

步骤3 锁定写保护，请参见“[配置流程的步骤2](#)”。

----结束

退出Memory低功耗流程：

步骤1 解除写保护，请参见“[配置流程的步骤1](#)”。

步骤2 配置寄存器QDMAC_MEM_LP的bit[31:28]、bit[27:24]、bit[21:12]、bit[11:8]、bit[0]，分别完成Setup1、Setup0、Enter、Exit状态下计数阈值的配置以及使能的配置，计数器的计数单位为aclk；例如，需要在所有任务完成以后256个aclk cycle就进入Light Sleep，则Enter状态计数器的阈值为0xFF，Setup0、Setup1、Exit状态下的计数阈值与Memory的时序特性相关，假设都为2个aclk，则计数阈值为0x1，配置0x110FF101到QDMAC_MEM_LP表示QDMAC空闲以后可以自动控制Memory进入Light Sleep，如[表16-83](#)所示。

表 16-84 QDMAC_MEM_LP 寄存器

Bits	寄存器说明	配置值
QDMAC_MEM_LP：偏移地址0x31C		
[31:28]	Setup1状态的计数时间配置，用于控制Memory Light Sleep控制信号稳定时间。单位为工作时钟cycle。	0x1



Bits	寄存器说明	配置值
[27:24]	Setup0状态的计数时间配置，用于控制模块空闲超时后（Enter状态后）到Memory控制低功耗信号之间的建立时间。单位为工作时钟cycle。	0x1
[23:22]	保留。	-
[21:12]	Enter状态的计数时间配置，用于控制QDMAC空闲以后，等待进入Memory Light Sleep的时间，即空闲以后在该时间内没有任何触发QDMAC继续工作的操作，QDMAC则开始控制Memory进入Light Sleep。单位为工作时钟cycle。	0xFF
[11:8]	Exit状态的计数时间配置，用于控制Memory从Light Sleep状态唤醒需要的时间，单位为工作时钟cycle。	0x1
[7:1]	保留。	-
[0]	Memory Light Sleep使能控制。 0：QDMAC不会进行Memory Light Sleep控制； 1：QDMAC自动进行Memory Light Sleep控制。	0b1

步骤3 锁定写保护，请参见“配置流程”的步骤2。

----结束

16.23.3 配置约束

说明

- 1. Setup0、Setup1、Exit状态下的计数阈值需要查询芯片Memory相关手册进行配置，建议尽量配置较大的值，避免引起Memory异常。
- 2. Enter状态下的计数阈值需要根据业务场景计算，例如10倍于配置任务的间隔时间。

16.24 QDMAC 低功耗接口

16.24.1 场景描述

QDMAC提供一个低功耗握手接口，用于与LPC配合完成QDMAC的低功耗行为。当QDMAC处于低功耗状态时，可以对QDMAC进行时钟门控或者掉电等低功耗行为。

QDMAC的低功耗进入处理流程：

- 步骤1 软件迫使QDMAC的Memory退出低功耗状态。
- 步骤2 LPC发起低功耗请求。
- 步骤3 QDMAC阻止软件配置Queue的任务寄存器。



- 步骤4** QDMAC等待所有任务执行完成，包括写消息，写回完成。
- 步骤5** QDMAC等待所有原始中断被清除。
- 步骤6** 上述条件满足后QDMAC进入低功耗，与LPC握手成功。

----结束

QDMAC的低功耗退出流程：LPC发起退出低功耗请求，QDMAC收到请求以后立即退出低功耗状态，软件可以正常访问QDMAC。

须知

如果QDMAC出现异常导致一直无法与LPC握手成功，LPC会保证在超时以后自动撤销低功耗请求，QDMAC可以被软件正常访问。

16.24.2 配置流程

QDMAC的低功耗操作流程：

- 步骤1** 解除写保护，请参见“配置流程”的**步骤1**。
- 步骤2** 配置QDMAC_MEM_LP寄存器的bit[0]为0，关闭Memory低功耗使能，如**表16-83**。
- 步骤3** 等待 200 个时钟周期后，查询寄存器QDMAC_STAT的bit[0]为1，表示Memory处于Normal状态，如**表16-85**所示：

表 16-85 QDMAC_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_STAT：偏移地址0x324		
[31:5]	保留。	-
[4]	DMAC的忙闲指示信号。 0：DMAC处于闲状态； 1：DMAC处于忙状态。	-
[3:1]	保留。	-
[0]	Memory的低功耗状态指示： 0：Memory处于Light Sleep状态； 1：Memory未进入Light Sleep状态。	0b1

- 步骤4** 配置LPC发起低功耗请求，具体见LPC用户手册。
- 步骤5** 查询LPC中QDMAC的低功耗状态，等待QDMAC进入低功耗状态，在等待过程中如果QDMAC有中断产生，需正常处理中断（至少需要保证清除中断），中断清除请参考“处理流程”，“处理流程”。
- 步骤6** 确认QDMAC进入低功耗状态以后，可以配置LPC或者系统控制对QDMAC进行低功耗操作，具体见LPC用户手册。



步骤7 需要将QDMAC从低功耗状态唤醒时，配置LPC发起退出低功耗请求。

步骤8 查询LPC中QDMAC的低功耗状态，确认QDMAC已经退出低功耗。

----结束

16.24.3 配置约束

说明

1. LPC发起低功耗请求以后，QDMAC会阻止软件配置任务，否则会导致任务配置异常。
2. LPC可以在QDMAC空闲或者忙时发起请求，QDMAC会保证在空闲以后才会进入低功耗状态。
3. 即使LPC已经发起了低功耗请求，QDMAC产生的中断也需要被正常处理，否则QDMAC无法完成低功耗握手。

16.25 时钟复位

16.26 时钟复位步骤

每次解复位和开时钟之后，必须读取CRG状态寄存器确认模块已经解复位和开时钟，然后启动模块功能。

步骤1 关闭模块时钟，配置SC_QDMA0_ICG_DIS各有效域段值为1。

步骤2 循环查询SC_QDMA0_ICG_ST，若各有效域段查询值为0，则表示模块对应时钟关闭，进行下一步操作。

步骤3 复位模块，配置SC_QDMA0_RESET_REQ各有效域段值为1。

步骤4 循环查询SC_QDMA0_RESET_ST，若查询到各有效域段值为1，则表示模块处于软复位状态，进行下一步操作。

步骤5 打开模块时钟，配置SC_QDMA0_ICG_EN各有效域段值为1。

步骤6 循环查询SC_QDMA0_ICG_ST，若查询各有效域段值为1，则表示模块对应时钟打开，进行下一步操作。

步骤7 等待500ns。关闭模块时钟，配置SC_QDMA0_ICG_DIS各有效域段值为1。

步骤8 循环查询SC_QDMA0_ICG_ST，若查询各有效域段值为0，则表示模块对应时钟关闭，进行下一步操作。

步骤9 软件判断软复位已经完成，执行以下步骤进行解复位和开时钟。

步骤10 对模块进行解复位，配置SC_QDMA0_RESET_DREQ各有效域段值为1。

步骤11 循环查询SC_QDMA0_RESET_ST，若查询各有效域段值为0，则表示模块处于解复位状态。

步骤12 打开模块时钟，配置SC_QDMA0_ICG_EN各有效域段值为1。

步骤13 循环查询SC_QDMA0_ICG_ST，若查询各有效域段值为1，则表示模块对应时钟打开。

完成以上步骤后，软件就可以操作模块工作。

----结束



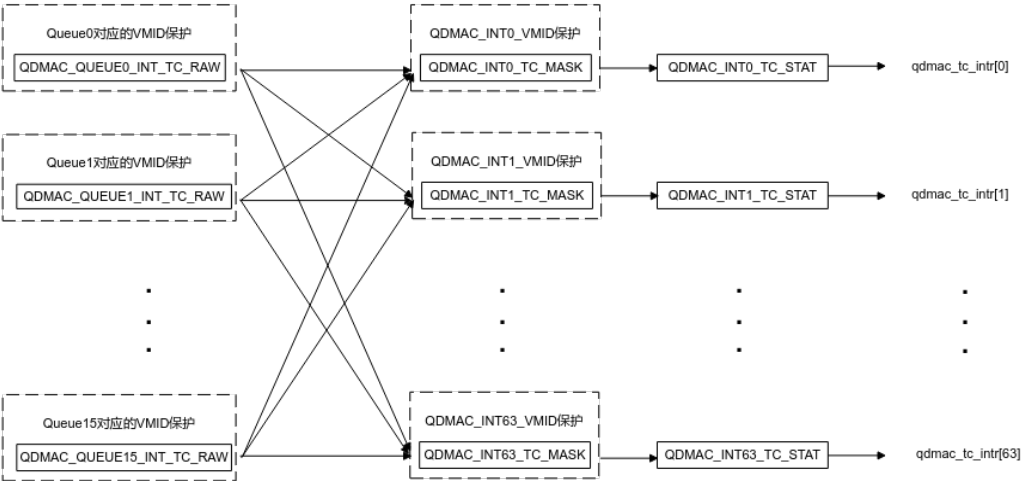
16.27 中断处理

16.28 QDMAC 完成中断

16.28.1 中断描述

QDMAC支持任务完成中断上报，即任务搬移完成以后通过完成中断线上报，中断状态按照Queue上报，原始中断状态寄存器、屏蔽后中断状态寄存器和中断屏蔽寄存器的bit[15:0]分别表示Queue15~Queue0。QDMAC_GLB支持34（其他宏定义支持10根）根完成中断线上报，34根中断线可分别路由到34个处理器上，每根中断线由单独的中断屏蔽寄存器控制，每个处理器单独配置为其分配的中断屏蔽寄存器实现对不关心Queue中断的屏蔽，结构如图16-21所示。

图 16-21 完成中断结构示意图



16.28.2 处理流程

QDMAC完成中断处理流程：

- 步骤1 解除写保护，请参见“配置流程”的步骤1。
- 步骤2 配置完成中断屏蔽寄存器QDMAC_INT_TC_MASK in 的bit[15:0]，设置完成处理器 in 的中断屏蔽；例如处理器1只关心Queue1和Queue2的完成中断，则配置0x0000_00F9到QDMAC_INT_TC_MASK1（偏移地址0x2004）禁止Queue1和Queue2的完成中断屏蔽，如表16-86所示。

表 16-86 QDMAC_INT_TC_MASK 寄存器

Bits	寄存器说明	配置值
QDMAC_INT_TC_MASK in : 0x01004 + 0x1000 $\times in$ (QDMAC_GLB: $in = 0 \sim 33$, QDMAC_OTHER: $in = 0 \sim 9$)		



Bits	寄存器说明	配置值
[31:16]	保留。	-
[15:0]	INT in的各个Queue的BD数据搬移完成中断屏蔽寄存器，bit15~0分别对应Queue15~0的中断屏蔽 0：中断屏蔽无效； 1：中断屏蔽有效（默认）。 受INT in的VMID保护。	0x00F9

- 步骤3** 锁定写保护，请参见[配置流程步骤2](#)；如果是普通BD，进行[步骤4](#)，如果是短BD，进行[步骤5](#)。
- 步骤4** 准备BD，并将BD CONFIG域的bit[2]置1（具体描述见[表16-1](#)），按照“[普通BD任务配置流程](#)”完成任务配置。
- 步骤5** QDMAC的完成中断触发以后，查询寄存器QDMAC_INT_TC_STAT in ，获取处理器 in 屏蔽后的中断状态；例如按“[普通BD任务配置流程](#)”例子查询QDMAC_INT_TC_STAT1获取的结果应该为0x0000_0002（因为任务被配置到了Queue1），如[表16-87](#)所示。

表 16-87 QDMAC_INT_TC_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_INT_TC_STAT in : 0x01000 + 0x1000 $\times in$ (QDMAC_GLB: $in = 0 \sim 33$, QDMAC_OTHER: $in = 0 \sim 9$)		
[31:16]	保留。	-
[15:0]	INT in的各个Queue的BD数据搬移完成屏蔽后中断状态，bit15~0分别对应Queue15~0的中断状态 0：未产生中断； 1：已产生中断。	0x02

- 步骤6** 查询QDMAC_QUEUE_DONE_BD qn ，获取第 qn 个Queue的完成BD指针；例如按“[普通BD任务配置流程](#)”例子查询QDMAC_QUEUE_DONE_BD1获取的结果为0x2000_0000，如[表16-88](#)所示。

表 16-88 QDMAC_QUEUE_TC 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_TC qn : 0x280+0x10 qn , $qn=0\sim7$		



Bits	寄存器说明	查询值
[31:0]	Queue qn的最新完成任务信息寄存器，记录BD地址。 注意： 1、普通BD每一个BD都会更新该寄存器，短BD只有链表的非首节点BD才会更新该寄存器 2、除了溢出以外，只要DMA处理过的BD都会更新该寄存器。	0x2000_0000

步骤7 清除中断，向寄存器QDMAC_QUEUE_INT_TC_RAWqn对应bit[0]写0x1表示清除中断；例如“[普通BD任务配置流程](#)”例子向地址0x46240写0x0000_0001清除中断，如[表16-89](#)所示。

表 16-89 QDMAC_QUEUE_INT_TC_RAW 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_INT_TC_RAWqn: 偏移地址0x45240 + 0x1000×qn （qn = 0 ~ 15）		
[31:1]	保留。	-
[0]	第qn个Queue的BD数据搬移完成中断状态： 读： 0：未产生中断； 1：已产生中断。 写： 0：无影响，保持原值； 1：清除对应的中断状态。 受Queue qn的VMID保护。	0x01

----结束

须知

1. 同一个Queue的完成中断产生顺序保持与任务配置到Queue中的顺序一致，不同Queue之间的中断顺序没有限制。
2. 完成中断只要在数据搬移过程没有出现错误就会上报，但是需要等待写消息和写回完成以后才会产生。
3. 如果Queue的上一次原始完成中断未被清除，又需要产生新的完成中断，新的完成中断信息会覆盖旧的中断信息，即每个Queue的QDMAC_QUEUE_DONE_BD寄存器只保留最新的完成中断信息。
4. QDMAC允许多核用同一个Queue，但任意一个核清除中断都会导致中断线变为无效，所以在这种场景下，由软件来保证中断的正确处理，如果无法做到多核间的通信，建议一个核用一个Queue，不采用中断方式没有此限制。



16.29 QDMAC 告警中断

16.29.1 中断描述

QDMAC支持任务错误中断上报，即任务发生异常以后通过错误中断线上报，中断状态按照Queue上报，原始中断状态寄存器、屏蔽后中断状态寄存器和中断屏蔽寄存器的bit[15:0]分别表示Queue15~Queue0。QDMAC支持4根完成中断线上报，每根中断对应一个通道组，每根中断线由单独的中断屏蔽寄存器控制，每个通道组单独配置为其分配的中断屏蔽寄存器实现对不关心Queue中断的屏蔽，其结构可以参考图16-21。

错误中断包括六类错误：BD参数错误、BD数据搬移错误、BD写消息错误、BD写回错误、BD读取错误、Queue溢出错误，只要出现了六类错误中的一类便会触发错误中断。

BD参数错误：QDMAC根据Queue中的BD指针读取BD的具体内容，如果在参数解析中发生以下情况会上报BD参数错误中断，并且发生错误后任务不会执行。

普通BD告警类型：

- 配置Account为0。
- 支持narrow传输时，配置burst size的reserved值；例如配置的AXI总线位宽为128bit，配置burst size为256bit。
- burst length可配置时，配置BD表CONFIG域的sl和dl为非法字段。
- FIX传输时，配置地址、多维传输的地址偏移与burst size不对齐。
- 当msg_en为1时，msg_addr没有按照要求对齐。
- 当msg_en为1时，msg_num配置为0b11。
- 当write_back_en为1时，CONFIG的bd_flag域为配置为1。
- 支持定浮点转换时，地址、多维传输的地址偏移以及数据量未按要求对齐。
- 支持压缩解压缩时，地址和数据量未按要求对齐。
- 支持压缩解压缩时，Bcount或者Ccount为非0，即多维传输。
- 同时使能了定点转换浮点和浮点转换定点。
- 同时使能了压缩和解压缩。
- 同时使能了压缩和定点转换浮点。
- 同时使能了解压缩和浮点转换定点。
- 清固定值使能时，打开了定浮点转换或者压缩解压缩使能。

发生BD参数解析告警以后，该BD指示的数据搬移任务不会被执行，但是如果写回使能，需要写回，指示任务执行失败；如果写Message使能，需要写Message，同时指示任务执行失败（如果Message配置错误或者Message只发送32bit则不会发送Message）。

BD数据搬移告警：QDMAC在数据搬移时，如果AXI总线反馈不为OKAY，QDMAC认为数据搬移失败，则上报BD数据搬移告警中断。发生了数据搬移告警后，QDMAC会一直持续搬移到该BD指示的数据搬移任务结束，触发数据搬移告警中断，不会触发数据搬移完成中断；如果写回使能，需要写回，指示任务执行失败；如果写Message使能，需要写Message，同时指示任务执行失败。



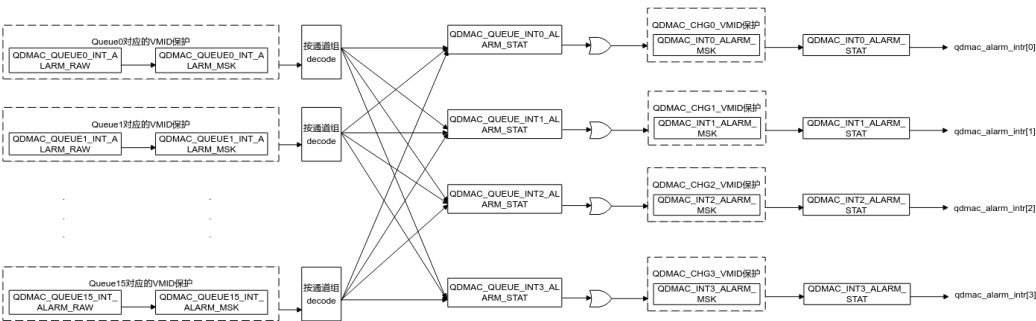
BD写消息告警：QDMAC在BD搬移完成以后通过AXI Master接口写消息时总线反馈不为OKAY，QDMAC认为消息发送失败，则上报BD写消息告警中断，不会再次发出消息，但是不影响写回操作，也不影响数据搬移完成中断的产生。

BD写回告警：QDMAC在BD搬移完成以后通过AXI Master接口写BD完成标志时总线反馈不为OKAY，QDMAC认为写回失败，则上报BD写回告警中断，不会再次发出写回操作，不影响数据搬移完成中断的产生。

BD读取告警中断：QDMAC在读取BD时，如果AXI总线反馈不为OKAY，QDMAC认为BD读取失败，则上报BD读取告警中断，BD不会被执行，同时强制不发Message和写回。

Queue溢出中断：当Queue满以后，软件还试图向该Queue配置BD任务，QDMAC会丢弃该任务，同时上报BD溢出告警中断。

图 16-22 告警中断结构示意图



须知

如果任务为链表任务，发生BD参数解析告警、BD数据搬移告警、BD写消息告警、BD写回告警、BD读取告警都会导致链表后面的BD任务丢弃，即发生了上述告警中断，链表任务在当前BD任务完成后立即终止。

16.29.2 处理流程

QDMAC错误中断处理流程：

- 步骤1 解除写保护，见“配置流程”步骤1。
- 步骤2 完成Queue的初始化，见“配置流程”的步骤1、步骤2；
- 步骤3 配置错误中断屏蔽寄存器QDMAC_QUEUE_INT_ALARM_MSKqn的bit[0]，设置queue n中断屏蔽；例如打开queue1的错误中断屏蔽，配置QDMAC_QUEUE_INT_ALARM_MSK1（偏移地址0x46254）的值为0x0，如表16-90所示。

表 16-90 QDMAC_QUEUE_INT_ALARM_MSK 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_INT_ALARM_MSKqn: 0x45254 + 0x1000×qn（qn = 0 ~ 15）		



Bits	寄存器说明	配置值
[31:1]	保留。	-
[0]	第qn个Queue的BD告警中断屏蔽寄存器： 0：中断屏蔽无效； 1：中断屏蔽有效（默认）。 受Queue qn的VMID保护。	0x0

步骤4 配置通道组gn的告警中断屏蔽寄存器QDMAC_INT_ALARM_MSK gn 的bit[0]，设置通道组 gn 的中断屏蔽；例如打开通道组1的错误中断屏蔽，则配置0x0000_0000到QDMAC_INT_ALARM_MSK1（偏移地址0x42004），如表16-91所示。

表 16-91 QDMAC_INT_ALARM_MSK 寄存器

Bits	寄存器说明	配置值
QDMAC_INT_ALARM_MSK in : 0x41004 + 0x1000 $\times gn$ ($gn = 0 \sim 3$)		
[31:1]	保留。	-
[0]	通道组gn的告警中断屏蔽寄存器 0：中断屏蔽无效； 1：中断屏蔽有效（默认）。 受通道组gn的VMID保护。	0x0

步骤5 锁定写保护，见“配置流程”的步骤2。

步骤6 QDMAC的错误中断触发以后，查询寄存器QDMAC_INT_QUEUE_ALARM_STAT gn ，获取通道组 gn 屏蔽后的中断状态；例如通道组1的Queue1发生了错误中断，则查询QDMAC_INT_QUEUE_ALARM_STAT1（偏移地址0x42008）得到的值为0x0000_0002，如表16-92所示。

表 16-92 QDMAC_INT_QUEUE_ALARM_STAT 寄存器

Bits	寄存器说明	查询值
QDMAC_INT_QUEUE_ALARM_STAT in : 0x4108 + 0x1000 $\times gn$ ($gn = 0 \sim 3$)		
[31:16]	保留。	-
[15:0]	通道组gn的各个Queue的告警屏蔽中断状态， bit15~0分别对应Queue15~0的中断状态 0：未产生中断； 1：已产生中断。	0x02

步骤7 查询QDMAC_QUEUE_ALARM_BD gn ，获取第 gn 个Queue的错误BD指针；例如Queue1发生了错误中断，则查询QDMAC_QUEUE_ALARM_BD1（偏移地址0x46260）获取的结果为0x2000_0000，如表16-93所示。



表 16-93 QDMAC_QUEUE_ALARM_BD 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_ALARM_BD qn : 0x45260 + 0x1000 $\times$ qn ($qn = 0 \sim 15$)		
[31:30]	保留。	-
[29:0]	Queue qn 的BD告警中断信息寄存器： 普通BD：记录BD地址[35:6]，低位为全0。	0x80_0000

步骤8 查询QDMAC_QUEUE_ALARM_TYPE qn ，获取第 qn 个Queue的错误BD的错误类型，如果错误类型是参数配置错误，进行**步骤9**，否则跳到**步骤10**；例如Queue1发生的错误类型为参数配置错误，则查询QDMAC_QUEUE_ERR_TYPE1（偏移地址0x46264）获取的结果应该为0x0000_0010，如表16-94所示。

表 16-94 QDMAC_QUEUE_ALARM_TYPE 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_ALARM_TYPE qn : 0x45264 + 0x1000 $\times$ qn ($qn = 0 \sim 15$)		
[31:6]	保留。	-
[5]	Queue溢出错误指示。 0：未发生Queue溢出错误； 1：发生了Queue溢出错误。	0b0
[4]	参数配置错误指示。 0：未发生参数配置错误； 1：发生了参数配置错误。	0b1
[3]	写回BD错误指示。 0：未发生写回BD错误； 1：发生了写回BD错误。	0b0
[2]	写message错误指示。 0：未发生写message错误； 1：发生了写message错误。	0b0
[1]	读BD错误指示。 0：未发生读BD错误； 1：发生了读BD错误。	0b0
[0]	BD数据搬移错误指示。 0：未发生数据搬移错误； 1：发生了数据搬移错误。	0b0

步骤9 查询QDMAC_QUEUE_ALARM_CFGERR qn ，获取第 qn 个Queue的参数配置错误的类型；例如Queue1发生的错误类型为参数配置错误里面的解压缩acnt数据量配置溢出错误



误，则查询QDMAC_QUEUE_ALARM_CFGERR1（偏移地址0x46268）获取的结果应该为0x0000_0100，如表16-95所示。

表 16-95 QDMAC_QUEUE_ALARM_CFGERR 寄存器

Bits	寄存器说明	查询值
QDMAC_QUEUE_ALARM_CFGERRqn: 0x45268 + 0x1000×qn（qn = 0~15）		
[31:14]	保留。	-
[13]	算法使能配置错误指示。 0：未发生错误； 1：发生了错误。	0b0
[12]	定浮点转换数据量和地址对齐错误指示。 0：未发生错误； 1：发生了错误。	0b0
[11]	压缩解压缩数据量不是RB整数倍配置错误指示。 0：未发生错误； 1：发生了错误。	0b0
[10]	压缩解压缩多维配置错误指示。 0：未发生错误； 1：发生了错误。	0b0
[9]	压缩解压缩地址非对齐错误指示。 0：未发生错误； 1：发生了错误。	0b0
[8]	解压缩acnt数据量配置溢出错误指示。 0：未发生错误； 1：发生了错误。	0b1
[7]	解压缩类型配置错误指示。 0：未发生错误； 1：发生了错误。	0b0
[6]	Message配置错误指示。 0：未发生错误； 1：发生了错误。	0b0
[5]	写回标志配置错误指示。 0：未发生错误； 1：发生了错误。	0b0
[4]	burst size配置错误指示。 0：未发生错误； 1：发生了错误。	0b0



Bits	寄存器说明	查询值
[3]	burst length配置错误指示。 0: 未发生错误; 1: 发生了错误。	0b0
[2]	format配置错误指示。 0: 未发生错误; 1: 发生了错误。	0b0
[1]	源目的地址配置错误指示。 0: 未发生错误; 1: 发生了错误。	0b0
[0]	Acount配置为0错误。 0: 未发生错误; 1: 发生了错误。	0b0

步骤10 清除中断，向QDMAC_QUEUE_INT_ALARM_RAW qn 的bit[0]写0x1表示清除中断；例如清除Queue1的错误中断则写0x0000_0001清除中断，如表16-96所示：

表 16-96 QDMAC_QUEUE_INT_ALARM_RAW 寄存器

Bits	寄存器说明	配置值
QDMAC_QUEUE_INT_ALARM_RAW qn : 偏移地址00x45250 + 0x1000 $\times$ qn ($qn = 0 \sim 15$)		
[31:1]	保留。	-
[0]	第 qn 个Queue的BD告警中断状态。 读： 0: 未产生中断; 1: 已产生中断。 写： 0: 无影响，保持原值; 1: 清除对应的中断状态。 该中断可能由以下原因触发。 1. BD参数错误。 2. BD数据搬移错误。 3. BD写消息错误。 4. BD写回错误。 5. BD读取错误。 6. Queue溢出错误。 受Queue qn 的VMID保护。	0x1



----结束

须知

1. QDMAC_QUEUE_ALARM_BD寄存器和QDMAC_QUEUE_ALARM_TYPE寄存器存储的是第一次BD错误的信息，同时，QDMAC还会执行已经配置的任务，如果后面的任务发生了错误，前面的错误原始中断还未清除，则错误信息会被丢弃。
2. 同一个Queue可能存在两个任务同时发生错误，QDMAC_QUEUE_ALARM_B寄存器记录的是配置顺序排在前面的任务的BD指针，QDMAC_QUEUE_ALARM_TYPE寄存器记录的是配置顺序排在前面的任务发生错误的类型。
3. BD链接任务可能发生当前任务搬移时发生了数据搬移错误和下一个任务读取错误，上报的错误类型包括数据搬移错误和读取BD错误，记录的地址为当前任务的BD指针。
4. 如果Queue溢出和其余错误同时发生，记录是其余错误的BD指针。
5. 如果软件清除错误中断和QDMAC内部产生错误中断同时发生，中断线无法正常清除，同时QDMAC记录的错误信息也不会更新，此处应用需要特别小心。

16.30 性能分析

16.31 性能分析

16.31.1 性能分析依据

影响DMA搬移性能主要有以下几个方面，性能分析着重从以下几个方面分析：

- 搬移的数据量Size。
- 搬移的源和目的设备在系统中位置，即访问路径的latency，源定义为Source Cluster，目的定义为Destination Cluster。
- 搬移的源和目的设备的处理时延，源定义为Source Memory，目的定义为Destination Memory。

定义性能分析公式：

- 实际时延：从读取数据开始到中断产生的时延。
- 理想时延：搬移数据量 / 16。
- 效率 = 理想时延 / 实际时延。
- 带宽 = 256 * 效率。

16.31.2 性能分析数据

QDMAC的性能分析数据：

- 启动开销：



- 从配置任务开始到AXI发出BD读取命令发出不超过8个cycle。
- 从BD内容返回到开始传输第一个读数据的命令6个cycle。
- 数据写完（写响应返回）到发出写message 3个cycle。
- 中断开销：BD执行完毕以后，到完成中断的开销为2个cycle。
- message开销：不考虑Slave写响应的开销，message开销为3个cycle。
- 数据搬移流水条件：
 - 不支持定浮点转换功能时，读数据的时延在43个cycle内可流水。
 - 只支持定浮点转换功能时，读数据的时延在41个cycle内可流水。
 - 只支持解压缩时，读数据的时延在30个 cycle内可流水（解压后的数据流水）。
 - 只支持压缩时，读数据的时延在53个cycle内可流水。
 - 支持定点转浮点和解压缩时，读数据的时延在28个cycle内可流水（解压后的数据流水）。
 - 支持浮点转定点和压缩压缩时，读数据的时延在52个cycle内可流水。

16.32 接口时序要求

无。



17 CAN

17.1 功能描述（feature）

Hi309a芯片集成了6个独立的CAN（Controller Area Network，控制器局域网）设备，连接在APB总线上。可以与具有CAN接口的设备通讯，也可以多个芯片组成控制网络，完成信息的传输、交互和控制功能。

- 符合ISO 11898-1:2015与ISO 11898-4标准。
- 波特率最高5Mbit/s。
- 每帧报文支持最大64Byte。
- 支持事件同步时间触发通讯。
- CAN错误记录。
- 支持AUTOSAR。
- 支持SAE J1939。
- 2个可配置的接收FIFO。
- 单独指令下接收高优先级消息。
- 高达64个专用的接收BUFFER。
- 高达32个专用的发送BUFFER。
- 支持可编程发送FIFO、发送队列、发送事件FIFO。
- 支持DMA访问主机CPU。
- 支持多个CAN共享相同的消息RAM。
- 支持可编程的环回自测试功能。
- 可屏蔽模块中断。
- SLV接口支持8/16/32数据访问。
- 支持双时钟域（CAN时钟域与HOST时钟域）。
- 支持电源掉电。
- 支持调试CAN。
- 根据CiA603支持硬件时间戳。