Hi309a 通用 MPU 芯片

用户指南 01-CPU 小系统

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CPU小系统的各项基本功能，常见应用模式的配置方法。

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。
⚠ 危险	表示如不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。

版权所有 © 海思技术有限公司

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。

版权所有 © 海思技术有限公司

目 录

前 言 i
1 产品简介 1
1.1 产品族 1
1.2 功能描述 2
1.2.1 处理器特性 3
1.2.2 存储特性 4
1.2.3 加速器特性 4
1.2.4 接口描述 4
1.2.5 安全特性 5
1.2.6 遵从的标准与协议 6
1.3 电气特性 9
1.4 应用领域 9
1.5 订购须知 10
2 全局数据流 11
2.1 关键术语 11
2.2 概述 11
2.3 LSW 快速转发业务 11
2.4 基础网络转发业务 12
2.5 快速加解密转发业务 13
2.6 增强转发业务 15
2.7 控制报文处理 16
2.7.1 控制报文 RX 处理流程 17
2.7.2 控制报文 TX 处理流程 18
2.8 PCIe 接口应用场景数据流 18
2.8.1 PCIe 设备向 Hi309a 发送报文的数据流 19
2.8.2 Hi309a 对从 PCIe 接收到的报文的处理流程 21
2.8.3 完成 PCIe 设备间的数据交换 22
2.8.4 查询模式的缺点和局限 22
2.8.5 下行方向数据报文流处理（SPPE--->PCIe） 22
3 A55 Cluster 23
3.1 概述 23
3.2 特性 23

版权所有 © 海思技术有限公司

3.3 功能描述 24
3.3.1 数据流 24
3.3.2 FCM 结构 24
3.3.3 时钟 24
3.3.4 复位 25
3.4 启动 25
3.5 注意事项 25
4 时钟 26
4.1 概述 26
4.2 关键术语 26
4.3 功能特性 26
4.4 功能描述 27
4.5 应用配置 28
4.5.1 PLL 配置 28
4.5.1.1 PLL 控制信号 29
4.5.1.2 PLL 频率配置公式 31
4.5.1.3 PLL 配置流程 31
4.5.1.4 PLL 典型配置值 32
4.5.2 网络同步以太 33
4.6 初始化 34
4.7 注意事项 34
4.8 DFX 35
4.9 性能指标 35
5 复位 36
5.1 概述 36
5.2 关键术语 36
5.3 功能特性 36
5.3.1 复位接口信号 37
5.3.1.1 网络场景下复位接口信号 37
5.4 复位类型 38
5.4.1 芯片上电复位 39
5.4.2 网络外狗场景芯片非上电复位 39
5.4.2.1 外部复位源输入 39
5.4.2.2 全局软复位 40
5.4.2.3 内部狗溢出复位 40
5.4.2.4 芯片非上电复位流程 40
5.4.2.5 CPU 系统软复位 41
5.4.3 网络内狗场景芯片非上电复位 41
5.4.3.1 外部复位源输入 41
5.4.3.2 全局软复位 42
5.4.3.3 内部狗溢出复位 42
5.4.3.4 芯片非上电复位流程 43

 版权所有 © 海思技术有限公司

5.4.3.5 CPU 系统软复位 43
5.5 模块软复位流程 44
5.6 CPU 异常复位 LANSW 正常转发流程 45
5.7 逻辑喂狗 48
5.8 双 BIOS 启动跟 CRG 相关的介绍 48
5.9 RST_OUT_N 49
5.10 复位约束 49
5.11 复位原因记录 49
5.11.1 Hi309a 网络络场景下支持以下复位类型 50
6 系统初始化 51
6.1 概述 51
6.2 关键术语 51
6.3 特性 52
6.4 遵循协议 53
6.5 功能描述 53
6.5.1 上电复位 53
6.5.2 热复位 54
6.5.3 eFuse 加载和硬件初始化 54
6.5.4 启动真值表 55
6.5.5 双 BOOT 启动方案 70
6.5.5.1 Hi309a 的硬件逻辑切换的双 BOOT 70
6.5.5.2 Hi309a 网络软件双 BOOT 74
6.5.6 Hi309a 网络启动流程 74
6.5.6.1 HISS 启动流程 74
6.5.6.2 A55 启动流程 74
6.5.7 软复位 75
6.6 编程接口 76
6.6.1 系统初始化 76
6.7 注意事项 78
6.7.1 Cache 一致性配置注意事项 78
6.7.2 配置注意事项 78
6.7.3 软件约束 79
6.7.4 BSBC 软件需求 79
7 地址控制 80
7.1 概述 80
7.2 特性 80
7.3 关键术语 81
7.4 默认地址分布 82
7.5 DAW 82
7.5.1 DAW 介绍 82
7.5.1.1 一级 DAW 82
7.5.1.2 二级 DAW 83

 版权所有 © 海思技术有限公司

7.5.1.3 三级译码 83
7.5.1.3.1 DAW 和设备实际空间/有效空间映射关系 83
7.5.2 DAW 配置方式 83
7.6 多模隔离 84
7.6.1 多模空间介绍 84
7.6.2 多模空间配置方式 85
8 系统性能 87
8.1 概述 87
8.2 关键术语 87
8.3 特性 87
8.4 功能描述 89
8.4.1 应用框图 90
8.4.2 相关模块 91
8.4.3 芯片数据流图 92
9 系统反压和流量控制 96
9.1 概述 96
9.2 关键术语 97
9.3 功能特性 99
9.4 遵循协议 101
9.5 功能描述 101
9.5.1 应用框图 101
9.5.2 相关模块 103
9.5.3 数据结构 104
9.5.3.1 POE 对加速器的有任务通知 104
9.5.3.2 BMU 的 QoS 反压 105
9.5.3.3 POE 对 TM 的反压 105
9.5.4 HAC 的总线接入 105
9.5.5 系统中 QoS 的传递 106
9.5.5.1 HAC 107
9.5.5.2 AXI Scheduler 107
9.5.5.3 SMMU 108
9.5.5.4 AA 与 Cross Station 108
9.5.5.5 LLC 108
9.5.5.6 Dispatch 108
9.5.5.7 DDRC 及 HHA 109
9.5.6 DDR 调度 109
9.5.7 DDR 拥塞管理 109
9.5.8 对外的反压流控 110
9.5.9 内部反压和流控的基本原则 111
9.5.9.1 加速器对 CPU 操作响应的原则 111
9.5.9.2 加速器间反压的基本原则 111
9.5.9.3 加速器间的指示 112

 版权所有 © 海思技术有限公司

9.5.10 CPU 添加加速器反压和异常处理 112
9.5.10.1 CPU 添加 POE 的处理方式 112
9.5.10.2 CPU 向 POE 获取任务的处理方式 113
9.5.10.3 CPU 向 BMU 申请和释放缓存 113
9.5.10.4 CPU 添加 TM 的处理方式 114
9.5.10.5 CPU 添加 SEC 的处理方式 114
9.5.10.6 CPU 添加 PPE 的处理方式： 114
9.5.11 加速器间访问失败处理 114
9.5.11.1 模块间反压基本原则 114
9.5.11.2 加速器添加 POE 115
9.5.11.3 加速器向 POE 获取任务 115
9.5.11.4 加速器添加 TM 115
9.5.11.5 加速器向 BMU 申请缓存 116
9.5.11.6 加速器向 BMU 释放缓存 116
9.5.11.7 加速器添加 PPE 116
9.5.12 包任务和 Buffer 传送 outstanding 机制各加速器间的互相访问关系及支持的 Outstanding 117
9.5.13 报文 QoS 117
9.5.13.1 报文 QoS 生成与调度 117
9.5.13.2 报文 QoS 与总线 QoS 117
9.6 软件指南 117
9.7 性能 117
9.7.1 延迟（Latency） 117
9.8 DFX 117
9.8.1 可靠性 117
9.8.1.1 自恢复 117
9.9 注意事项 118
10 系统功耗 119
10.1 关键术语 119
10.2 概述 120
10.3 特性 120
10.4 Compliance 121
10.5 功能描述 121
10.5.1 电源域 121
10.5.2 时钟域 123
10.5.3 功耗管理 124
10.5.4 CPU 功耗 124
10.5.4.1 Q-Channel 控制器 124
10.5.4.2 P-Channel 控制器 126
10.5.4.3 CORE 上下电控制器 128
10.5.5 IP 功耗 130
10.5.5.1 PLL 130

 版权所有 © 海思技术有限公司

10.5.5.2 eFuse..... 130
10.5.5.3 TSensor..... 130
10.5.5.4 DDR..... 131
10.5.5.5 Hilink..... 131
10.5.6 温度控制..... 131
10.5.7 AVS..... 131
10.5.7.1 AVS 静态工艺补偿 AVS0..... 132
10.6 Initialization Example..... 133
10.6.1 AVS 应用流程..... 133
10.6.2 PMU 芯片初始化..... 133
10.6.2.1 TPS544C24..... 134
10.6.2.2 VT7410..... 137
10.6.3 PMBus 初始化..... 139
10.6.4 PMBus 软件调压..... 147
10.6.5 PMBus 软件读电压流程..... 149
10.6.6 AVS 动态温度补偿配置流程..... 150
10.6.7 算法旁路 HPM 检测配置..... 154
10.6.8 Adaptive Clock..... 159
10.6.9 SPPE 掉电流程..... 165
10.6.10 HISS 掉电流程..... 166
10.6.11 相应模块 CLK GATING 配置流程..... 167
10.6.12 SERDES MACRO 的 power gating 配置流程..... 167
11 系统 DFX..... 172
11.1 关键术语..... 172
11.2 概述..... 172
11.3 特性..... 172
11.4 遵循协议..... 181
11.5 功能描述..... 181
11.5.1 应用框图..... 181
11.5.2 相关模块..... 182
11.5.3 内部信号监测..... 182
11.5.4 ALL-SCAN..... 198
11.5.5 功能 MBIST..... 200
11.5.6 Memory 软失效保护..... 200
11.5.7 Crack 检测..... 201
11.5.7.1 正常功能验证..... 201
11.5.7.2 强制注错功能验证..... 202
11.6 软件指南..... 203
11.6.1 Debug Bus 写 JTAG 寄存器..... 203
11.6.2 Debug Bus 读 JTAG 寄存器..... 204
11.6.3 HiLink Serdes 功能 BIST..... 204
11.6.4 功能 MBIST 测试流程..... 204

 版权所有 © 海思技术有限公司

11.6.4.1 CPU Cluster MBIST 测试流程 204
11.6.4.2 LLC MBIST 测试流程 205
11.6.4.3 DDRC BIST 测试流程 206
11.6.4.4 其他 SUBSYS BIST 测试流程 206
11.6.5 ALL-SCAN 测试流程 208
12 GIC 209
12.1 概述 209
12.2 关键术语 209
12.3 特性 211
12.4 遵循协议 213
12.5 功能描述 213
12.5.1 应用框图 213
12.5.2 相关模块 215
12.5.2.1 ITS 子模块架构 215
12.5.2.1.1 GICD 子模块架构 216
12.5.2.1.2 GICR 子模块架构 217
12.5.2.2 典型数据处理流程 217
12.5.2.2.1 GICD 中断上报流程 217
12.5.2.2.2 跨 GICD 间中断上报流程 219
12.5.2.2.3 ITS 中断上报流程 220
12.5.2.2.4 GICR 中断上报流程 222
12.5.2.3 基础特性描述 223
12.5.2.3.1 中断向量表 223
12.5.2.3.2 LPI 中断设备号 223
12.5.2.3.3 地址分配 223
12.5.2.3.4 MSI 224
12.5.2.3.5 GICD 亲和路由 225
12.5.2.3.6 1/N SPI 亲和路由 226
12.5.2.3.6.1 IRM 为 0 模式 226
12.5.2.3.6.2 IRM 为 1 模式 227
12.5.2.3.7 N/N SPI 路由 227
12.5.2.3.8 SGI 亲和路由 228
12.5.2.3.8.1 IRM 为 0 模式 228
12.5.2.3.8.2 IRM 为 1 模式 229
12.5.2.3.9 中断触发方式 229
12.5.2.3.10 LPI Config Table 229
12.5.2.3.11 中断优先级 230
12.5.2.3.12 中断嵌套 230
12.5.2.3.13 中断唤醒核 231
12.5.2.3.14 核休眠 232
12.6 应用配置 233
12.6.1 BIOS 配置说明 233

 版权所有 © 海思技术有限公司

12.7 初始化 233
12.7.1 GIC 初始化 233
12.7.1.1 ITS 初始化 233
12.7.1.2 GICR 初始化 235
12.7.2 Distributor 初始化 235
12.7.3 GICR\GICD 软件复位流程 236
12.7.4 GICR 被软件发现流程 236
12.7.5 LPI 中断路由配置流程 239
12.7.6 ITS 中断路由改配流程 240
12.7.6.1 ITS 配置接口 240
12.7.6.2 ITS Remapping or Unmapping 中断 241
12.7.6.3 ITS Remapping or Unmapping 设备 242
12.7.6.4 ITS Remapping or Unmapping 中断分组 242
12.7.6.5 ITS Remapping or Unmapping VCPU 242
12.7.7 GICD 中断路由配置流程 242
12.7.7.1 亲和路由模式 243
12.7.7.2 改变 SPI 亲和路由设置流程 243
12.7.8 GICD 中断跨片路由配置流程 243
12.7.8.1 SPI 中断跨 GICD 传递配置流程 243
12.7.8.2 SGI 中断跨 GICD 传递配置流程 244
12.7.9 GICR 中断转移流程 244
12.7.9.1 MOVALL 流程 244
12.7.9.2 MOVI 流程 245
12.7.9.3 虚拟中断重映射流程 246
12.7.9.4 虚拟核重映射流程 246
12.7.10 GICR 中断改配流程 246
12.7.11 虚拟机切换相关流程 246
12.7.11.1 改变虚拟 CPU 的 Residency 流程 246
12.7.11.2 Re-Distributor 对于 Hyeprvisor 切换 VCPU 的工作流程 246
12.7.11.3 软件切换 Guest OS 需要执行动作 247
12.7.12 VMOVI 和 VMOVP 相关流程 247
12.7.12.1 VMOVI 中断迁移流程 247
12.7.12.2 VMOVP 中断迁移流程 248
12.7.13 注意事项 248
12.7.13.1 GIC SMMU 属性配置说明 248
12.7.13.2 GIC 内存分配说明 248
12.7.13.3 SPI/SGI/PPI 中断配置变更说明 249
12.7.13.4 ITS 中断注册说明 249
12.7.13.5 版本兼容性 249
12.7.13.6 多 P 场景中断上报路径说明 249
12.7.13.7 虚拟机中断优先级配置说明 250
12.7.13.8 GIC/ITS 解复位说明 250

 版权所有 © 海思技术有限公司

12.7.13.9 NNSPI 中断说明 250
12.7.13.10 空中断处理说明 250
12.8 内部数据流图 250
12.9 典型问题定位流程 250
12.9.1 中断丢失问题定位 250
12.9.1.1 检查 ITS 中断错误码 251
12.9.1.2 LPI 中断丢失定位实例 257
12.10 时钟复位 260
13 SysCtrl 261
13.1 概述 261
13.2 关键术语 261
13.3 特性 261
13.4 功能描述 262
13.4.1 应用场景 263
13.4.2 相关模块 263
13.4.3 软复位控制 264
13.4.4 模块时钟门控 264
13.4.5 模块软复位控制 264
13.4.6 安全控制 265
13.4.7 功能模式 MBIST 265
13.4.8 DIE ID 读取 265
13.4.9 TSensor 接口 266
13.4.10 外设的功能配置接口 266
13.4.11 寄存器锁定功能（安全） 266
13.4.12 PLL 失锁检测记录寄存器 266
13.4.13 DJTAG Master 266
13.4.14 AXI Master 268
13.4.15 SYSTEM COUNTER 同步 269
13.4.16 WFE 269
13.4.17 功放保护及掉电告警 269
13.4.18 引脚滤波功能 272
13.4.19 添加软件可控的复位记录 272
13.4.20 NAND 掉电保护 273
13.4.21 通用外部中断 274
13.5 工作流程 274
13.5.1 初始化 274
13.5.2 复位 274
13.5.3 功能模式 MBIST 操作 274
13.5.4 DJTAG 274
14 LLC 描述 275
14.1 概述 275
14.2 关键术语 275

 版权所有 © 海思技术有限公司

14.3 特性 276
14.4 遵循协议 278
14.5 功能描述 278
14.5.1 应用框图 278
14.5.2 模块内部结构 279
14.5.3 多 Bank 交织 279
14.5.4 Cache 行为判定 280
14.5.5 替换算法 280
14.5.6 Lockdown/Unlock 280
14.5.7 Maintain 操作 281
14.5.8 预取操作 282
14.5.9 性能跳水 282
14.5.10 RAM ECC 校验 283
14.5.11 虚拟化 283
14.5.12 数据 Bypass 功能 284
14.5.13 Cacheline 压缩技术 284
14.5.14 Evicted-Address Filter（EAF） 284
14.5.15 Cache Stash 284
14.5.16 低峰值功耗模式 285
14.5.17 统计功能 285
14.5.18 执行状态显示 288
14.5.19 自动初始化 290
14.5.20 SRAM 功能 290
14.6 软件指南 291
14.6.1 初始化 291
14.6.2 软复位 291
14.6.3 Maintain 操作 291
14.6.3.1 安全全局 Clean 292
14.6.3.2 安全全局 Invalidate 292
14.6.3.3 安全全局 Clean+Invalidate 292
14.6.3.4 安全按地址区段 Clean 292
14.6.3.5 安全按地址区段 Invalidate 292
14.6.3.6 安全按地址区段 Clean+Invalidate 293
14.6.3.7 安全按 Cluster Clean 293
14.6.3.8 安全按 Cluster Invalidate 293
14.6.3.9 安全按 Cluster Clean+Invalidate 294
14.6.3.10 安全按 Way Clean 294
14.6.3.11 安全按 Way Invalidate 294
14.6.3.12 安全按 Way Clean+Invalidate 294
14.6.3.13 非安全全局 Clean 294
14.6.3.14 非安全全局 Invalidate 295
14.6.3.15 非安全全局 Clean+Invalidate 295

 版权所有 © 海思技术有限公司

14.6.3.16 非安全按地址区段 Clean 295
14.6.3.17 非安全按地址区段 Invalidate 295
14.6.3.18 非安全按地址区段 Clean+Invalidate 296
14.6.3.19 非安全按 Cluster Clean 296
14.6.3.20 非安全按 Cluster Invalidate 296
14.6.3.21 非安全按 Cluster Clean+Invalidate 297
14.6.3.22 非安全按 Way Clean 297
14.6.3.23 非安全按 Way Invalidate 297
14.6.3.24 非安全按 Way Clean+Invalidate 297
14.6.4 Lockdown 操作 298
14.6.4.1 按 Way Shutdown 298
14.6.4.2 安全按地址段 Lockdown 298
14.6.4.3 安全按地址段 unLock 299
14.6.4.4 非安全按地址段 Lockdown 299
14.6.4.5 非安全按地址段 Unlock 299
14.6.5 虚拟化操作 300
14.6.6 预取操作 300
14.6.6.1 基于所有 dsid 预取操作 300
14.6.6.2 基于配置的部分 dsid 预取操作 300
14.6.6.3 基于配置的部分{lpid，srcid}预取操作 301
14.6.7 性能跳水操作 301
14.6.7.1 基于所有 dsid 性能跳水操作 301
14.6.7.2 基于配置的部分 dsid 性能跳水操作 301
14.6.7.3 基于配置的部分{lpid，srcid}性能跳水操作 301
14.6.8 地址段统计 DFX 操作 302
14.6.9 Cluster 统计 DFX 操作 302
14.6.10 Dsid 统计 DFX 操作 302
14.6.11 LLC 内部 Buffer 状态信息查询配置流程 302
14.6.12 LLC 特定 DSID 在 LLC 所在容量查询配置流程 303
14.6.13 强制地址段为 Inclusive 结构 的 LLC 配置流程 303
14.6.14 LLC 不同工作模式的配置流程 303
14.7 性能 304
14.8 DFX 应用指导 304
14.8.1 故障在线检测 304
14.8.2 模块维测 304
14.8.2.1 功能统计 304
14.8.3 中断上报 307
14.8.4 典型定位案例 307
14.9 注意事项 307
15 SRAM 模块 308
15.1 概述 308
15.2 特性 308

 版权所有 © 海思技术有限公司

15.3 关键术语 309
15.4 遵循协议 310
15.5 功能描述 310
15.5.1 Normal 空间操作 310
15.5.2 Exclusive 空间操作 311
15.5.3 安全属性操作 311
15.5.4 ECC 功能 311
15.6 软件指南 312
15.6.1 配置 SRAM 初始化流程 312
15.6.2 正常 ECC 使用流程 312
15.6.3 构造 ECC 错误的推荐方法 313
15.6.4 CPU 因 Data ECC 异常进入 Abort 状态配置 313
15.6.5 软复位 313
15.7 性能 313
15.8 DFX 313
15.8.1 故障注入 314
15.9 注意事项 314
16 SMMU 315
16.1 Overview 315
16.2 Features 316
16.3 Key Terms 320
16.4 Compliance 321
16.5 功能描述 321
16.5.1 Stage1 和 Stage2 应用场景 321
16.5.2 SMMU 页表格式 321
16.5.3 StreamID 的组成 321
16.5.4 Configuration 查找 321
16.5.4.1 Stream Table Entry 的查找 322
16.5.4.2 Context Descriptor 的查找 323
16.5.5 地址转换流程简述 324
16.5.5.1 单 Stage 地址转换流程 324
16.5.5.2 Stage1+Stage2 转换流程 326
16.5.6 全流程简述 326
16.5.7 属性转换 327
16.5.7.1 属性叠加规则 328
16.5.8 Command Queue 329
16.5.8.1 支持的 Command 类型 329
16.5.8.2 Command 格式 330
16.5.8.3 Command 域段 333
16.5.9 Event Queue 333
16.5.9.1 支持的 Event 类型 333
16.5.9.2 Event 格式 334

 版权所有 © 海思技术有限公司

16.5.9.3 Event 域段 337
16.5.10 Streaming Ordered Write 337
16.5.10.1 使用规则 338
16.5.11 IO-Cache 338
16.5.11.1 使用规则 339
16.5.12 Fault 和中断 339
16.5.12.1 Fault 类型 339
16.5.12.2 Fault 处理方式 339
16.5.12.3 中断处理 340
16.5.13 PCIe 的 Atomic 操作 340
16.5.13.1 Atomic 命令类型 340
16.5.13.2 其他限定 340
16.5.14 RAS 341
16.5.14.1 Correct Error 341
16.5.14.2 Unrecoverable Uncorrected Error 341
16.5.14.3 Recoverable Uncorrected Error 341
16.6 数据结构 341
16.6.1 Stream Table Entry 配置描述 341
16.6.2 Context Descriptor 配置描述 349
16.7 应用场景配置 354
16.7.1 STE 和 CD 建立 354
16.7.1.1 STE 建立流程 354
16.7.1.2 CD 建立流程 355
16.7.2 STE 透传配置 355
16.7.3 非透传转换配置 356
16.7.4 User 域 Bypass 模式属性转换 358
16.7.5 SMMU 整体透传模式属性转换 359
16.7.6 STE 透传模式属性转换 361
16.7.7 非透传属性转换 364
16.7.7.1 Stage1 Only 属性转换 364
16.7.7.2 Stage2 Only 属性转换 365
16.7.7.3 Stage1 + Stage2 属性转换 365
16.7.8 SMMU 同步操作 365
16.7.9 Global Error 处理配置 366
16.7.10 Commad Queue 配置与功能 366
16.7.10.1 CMD Queue 寄存器配置 366
16.7.10.2 CMD Queue 的读写 367
16.7.10.3 CMD Queue Error 367
16.7.10.4 CMD Queue 溢出 368
16.7.10.5 CMD Queue 相关的中断 368
16.7.11 Event Queue 配置与功能 368
16.7.11.1 Event Queue 寄存器配置 368

 版权所有 © 海思技术有限公司

16.7.11.2 Event 的记录 368
16.7.11.3 Event Queue 溢出 369
16.7.11.4 Event Queue 相关的中断 369
16.7.11.5 Event Queue 与 Fault Configuration 369
16.7.12 PRI Queue 370
16.7.12.1 功能 370
16.7.12.2 PRIQ 寄存器配置 370
16.7.12.3 PRIQ 相关数据格式 371
16.7.12.4 PRIQ 溢出 371
16.7.12.5 PRIQ 相关的中断 371
16.7.13 Streaming Ordered Write 配置流程 372
16.7.14 IO-Cache 配置流程 372
16.8 初始化 372
16.9 接口定义 374
16.9.1 AXI 接口 374
16.9.2 Skyros 接口及约束 375
16.9.3 其他接口信号 376
16.10 Performance Monitor 以及芯片调试 DFX 功能特性描述 376
16.10.1 Performance Monitor 376
16.10.2 Performance Monitor 配置方法 377
16.10.3 DFX Debug 377
16.10.4 中断上报 377
16.11 典型问题定位流程 377
16.12 时钟复位 378
17 Timer 379
17.1 概述 379
17.2 特性 379
17.3 关键术语 380
17.4 遵循协议 380
17.5 功能描述 380
17.5.1 典型应用 380
17.5.1.1 TICK 380
17.5.1.2 波特率控制 380
17.5.1.3 流量控制 381
17.5.1.4 计数范围和计数精度 381
17.5.2 Timer 功能原理 381
17.5.3 多模隔离 382
17.5.4 原子特性 382
17.6 软件指南 383
17.6.1 Timer 初始化 383
17.6.2 中断设置 385
17.7 性能 385

 版权所有 © 海思技术有限公司

17.8 DFX 385
17.9 注意事项 385
18 TSensor 387
18.1 片上温度监控 TSensor 387
18.1.1 概述 387
18.1.2 特性 387
18.1.3 关键术语 387
18.1.4 遵循协议 388
18.1.5 功能描述 388
18.1.5.1 应用框图 388
18.1.5.2 相关模块 389
18.1.5.2.1 模块框图 389
18.1.5.3 TSensor 温度探测 389
18.1.5.4 TSensor 结温修正 389
18.1.5.4.1 TSensor 两点校准 389
18.1.6 软件指南 390
18.1.6.1 初始化 390
18.1.7 性能 390
18.1.7.1 应用场景 390
19 WatchDog 391
19.1 关键术语 391
19.2 概述 391
19.3 特性 391
19.4 遵循协议 392
19.5 功能描述 392
19.5.1 内部比较器机制 392
19.5.1.1 比较器刷新条件 393
19.5.1.1.1 软件更新 WDOG 比较器（Explicit watchdog refresh） 393
19.5.1.1.2 首次超时刷新（TimeoutRefresh） 393
19.5.1.2 WDOG 溢出事件 393
19.5.1.2.1 第一次 WDOG 溢出 WS0 393
19.5.1.2.2 第二次 WDOG 溢出 WS1 393
19.5.2 中断 394
19.5.2.1 中断寄存器 394
19.5.2.2 中断类型 394
19.6 编程接口 394
19.7 典型应用配置 394
19.8 初始化 394
19.8.1 看门狗初始化 394
19.9 注意事项 395
19.10 差异说明 395
19.11 时钟复位 395

 版权所有 © 海思技术有限公司

20 Cross Station 模块（AA）..... 396
20.1 概述..... 396
20.2 特性..... 396
20.3 关键术语..... 397
20.4 遵循协议..... 398
20.5 功能描述..... 398
20.5.1 典型应用..... 398
20.5.2 Link 层通信..... 399
20.5.3 汇聚分发..... 400
20.5.4 路由选择..... 401
20.5.5 防 Deadlock 机制..... 401
20.5.6 地址交织..... 402
20.5.6.1 L3 Cache Bank 交织..... 402
20.5.7 DAW 配置..... 403
20.5.8 数据统计..... 404
20.5.8.1 事件统计..... 405
20.5.8.2 延时统计..... 407
20.5.8.3 状态统计..... 407
20.5.9 Trap 传输信息..... 407
20.6 软件指南..... 408
20.6.1 初始化..... 408
20.6.2 软复位..... 408
20.6.3 DAW 配置..... 408
20.6.4 性能统计操作..... 409
20.7 性能..... 409
20.7.1 吞吐率（Throughput）..... 409
20.7.1.1 数据吞吐率..... 409
20.7.1.2 指令吞吐率..... 410
20.7.2 延迟（Latency）..... 410
20.8 DFX..... 410
20.8.1 Trap..... 410
20.8.2 Cross Station 挂死定位..... 410
20.9 注意事项..... 411
21 Dispatch..... 412
21.1 概述..... 412
21.2 关键术语..... 412
21.3 功能特性..... 412
21.4 遵循协议..... 413
21.5 功能描述..... 413
21.5.1 应用框图..... 413
21.5.2 相关模块..... 414
21.5.3 工作频率..... 414

 版权所有 © 海思技术有限公司

21.5.4 接口协议类型与数据位宽 414
21.5.5 Early WriteAcknowledge 414
21.5.6 DAW 功能 414
21.5.7 Barrier 处理 414
21.5.8 Qos 和 Retry 414
21.5.9 编程接口 415
21.6 应用配置 415
21.6.1 滑动地址窗口（DAW）配置更新 415
21.7 初始化 415
21.8 注意事项 415
21.9 DFX 416
21.9.1 故障在线检测 416
21.9.1.1 非法操作记录 416
21.9.1.1.1 异常命令中断状态记录 416
21.9.1.1.2 SLV 端返回错误响应中断记录 416
21.9.1.2 未完成操作统计及 Trace 416
21.9.2 故障离线检测 417
21.9.2.1 Trap 417
21.10 性能指标 417
21.10.1 延迟（Latency） 418
22 Scheduler 419
22.1 概述 419
22.2 关键术语 419
22.3 功能特性 419
22.4 遵循协议 420
22.5 功能描述 420
22.5.1 应用框图 420
22.5.2 相关模块 421
22.5.3 工作频率 421
22.5.4 接口数据位宽 421
22.5.5 跨 Cacheline 切分 421
22.5.6 WRR 调度 421
22.5.7 Master 端口 ID 设置 421
22.5.8 DJTAG 接口 422
22.5.8.1 编程接口 422
22.5.8.2 应用配置 422
22.6 初始化 422
22.7 注意事项 422
22.8 DFX 422
22.9 性能指标 423
22.9.1 延迟（Latency） 423
23 MN 424

版权所有 © 海思技术有限公司

23.1 概述 424
23.2 特性 425
23.3 遵从协议 426
23.4 功能描述 426
23.4.1 广播请求的接力 426
23.4.2 MN 节点配置说明 426
23.4.2.1 Barrier 操作处理 427
23.4.2.2 Dvm 操作处理 428
23.5 编程接口 428
23.5.1 应用配置 428
23.5.1.1 广播请求配置 428
23.5.1.2 动态关闭 cluster，MN 配置流程 428
23.5.1.3 动态关闭 core 429
23.5.1.4 动态关闭 cluster 429
23.6 初始化 429
23.7 注意事项 430
23.8 芯片调试 DFX 功能特性描述 430
23.8.1 MN 的统计信息 430
23.8.2 中断上报 430
23.8.3 消息上报 431
23.8.4 Debug 431
23.9 故障处理指南 431
23.9.1 软硬件接口故障 431
23.9.2 硬件内部数据完整性错误 431
23.10 典型问题定位流程 432
23.11 性能指标 432
23.11.1 延迟（Latency） 432
24 HHA 模块 433
24.1 概述 433
24.2 特性 433
24.2.1 关键术语 435
24.2.2 遵循协议 435
24.2.3 功能描述 435
24.2.3.1 应用框图 436
24.2.3.2 相关模块 436
24.2.3.3 工作频率 437
24.2.3.4 保序模型 437
24.2.3.5 TrustZone 437
24.2.3.6 DDR 接口 437
24.2.3.7 CPU 数据直接转发 437
24.2.3.8 Exclusive Monitor 437
24.2.3.8.1 Exclusive 传输 1 437

 版权所有 © 海思技术有限公司

24.2.3.8.2 Exclusive 传输 2 438
24.2.3.9 MN 功能描述 438
24.2.3.10 Barrier 操作处理 438
24.2.3.11 Dvm 操作处理 439
24.2.3.12 Default Slave 操作处理 439
24.2.3.13 MN 的 DFX 信息 439
24.3 软件指南 440
24.3.1 初始化 440
24.3.2 Mbist 测试操作 441
24.3.3 软复位 441
24.3.3.1 广播请求配置 442
24.3.3.2 动态关闭 cluster，MN 配置流程 442
24.3.4 性能 442
24.3.4.1 吞吐率（Throughput） 443
24.3.4.2 延迟（Latency） 443
24.4 DFX 443
24.4.1 模块维测 443
24.4.1.1 统计 443
24.4.1.2 中断上报 446
24.4.1.3 消息上报 446
24.4.1.4 Debug 447
24.5 注意事项 447
25 eFuse 448
25.1 eFuse 448
25.1.1 概述 448
25.1.2 特性 448
25.2 关键术语 449
25.3 应用框图 449
25.4 相关模块 450
25.5 模块框图 451
25.6 数据结构 451
25.7 软件指南 451
25.7.1 初始化 451
25.7.1.1 eFuse 烧写 451
25.7.1.2 在线更新 eFuse 中的 RIM 值 455
26 FTE 456
26.1 概述 456
26.2 关键术语 456
26.2.1 特性 457
26.3 遵循协议 458
26.4 功能描述 458
26.4.1 写操作功能 458

 版权所有 © 海思技术有限公司

26.4.2 读操作功能 458
26.4.3 ADD 和 GET POE 459
26.4.4 结果及统计功能 460
26.5 应用配置 460
26.6 初始化 460
26.7 注意事项 461
26.8 DFX 462
26.9 性能指标 462
26.9.1 吞吐率（Throughput） 462
27 Coresight 463
27.1 概述 463
27.2 特性 465
27.3 关键术语 466
27.4 遵循协议 466
27.5 功能描述 466
27.5.1 控制总线通路类模块 467
27.5.1.1 DP 467
27.5.1.2 AP 467
27.5.1.3 DAPBUSIC 468
27.5.1.4 APBIC 468
27.5.1.5 APBASYNCBRIDGE 468
27.5.1.6 APBSYNCBRIDGE 468
27.5.2 trace 数据总线模块 468
27.5.2.1 FUNNEL 468
27.5.2.2 REPLICATOR 469
27.5.2.3 TPIU 469
27.5.2.4 TMC 469
27.5.2.5 ATBUPSIZER，ATBDOWNSIZER 469
27.5.2.6 此类别的其他模块 470
27.5.3 传递 trigger 信息的 CT 模块 470
27.5.3.1 CTI 470
27.5.3.2 CTM 470
27.5.4 产生时间戳的 tsgen 模块 470
27.5.4.1 TSGEN 470
27.5.4.2 TSE 470
27.5.4.3 TSREPLICATOR 471
27.5.4.4 TSD 471
27.5.5 此类别的其他模块 471
27.6 工作方式 471
27.6.1 内部的 Counter 模块的使能方式 471
27.6.1.1 Counter 模块的寄存器 471
27.6.1.2 CNTCR 471

 版权所有 © 海思技术有限公司

27.6.1.3 CNTSR....472
27.6.1.4 CNTCVL....473
27.6.1.5 CNTCVU....473
27.6.1.6 CNTFID0....474
27.6.1.7 修改 counter 原值的步骤....475
27.6.2 各模块的偏移地址....475
27.7 性能....476
27.7.1 应用场景....477
27.7.2 注意事项....478
28 IPC....479
28.1 概述....479
28.2 依赖关系....479
28.3 操作步骤....480
28.3.1 单个目的的处理器通信....480
28.3.2 多个目的的处理器通信....484
28.3.3 自动应答模式....485
28.3.4 自动链接模式....490
28.3.5 IPC 软复位配置步骤....495
28.4 配置约束....495
28.5 应用说明....495
28.6 可维可测....495
29 SEM....496
29.1 概述....496
29.2 依赖关系....496
29.3 操作步骤....497
29.3.1 查询模式....497
29.3.2 队列模式....497
29.3.3 组合模式....497
29.3.4 SEMAPHORE 软复位配置步骤....498
29.4 配置约束....498
29.5 应用说明....498
29.6 可维可测....498

版权所有 © 海思技术有限公司

插图目录

图 1-1 整芯片逻辑图 2
图 2-1 LSW 快速转发业务处理流程 12
图 2-2 基础网络转发业务处理流程 13
图 2-3 快速加解密转发业务处理流程 14
图 2-4 增强转发业务处理流程 15
图 2-5 控制报文 RX 处理流程 17
图 2-6 控制报文 TX 处理流程 18
图 2-7 使用 PCIe 接口的主要数据流 19
图 2-8 PCIe 设备向 Hi309a 发送报文的数据流 20
图 2-9 传送网场景上行方向业务板到 CPU 控制信令处理流程 21
图 3-1 FCM 结构 24
图 4-1 PLL 功能结构图（12nm IP） 29
图 4-2 软件正常配置流程图 34
图 5-1 Hi309a 的复位结构图 37
图 5-2 上电复位时序图 39
图 5-3 逻辑喂狗 48
图 5-4 RST_OUT_N 时序 49
图 6-1 Hi309a 双 BOOT 方案图 72
图 6-2 SC_CRG_RST_ENABLE 复位源使能寄存器（Sysctrl） 73
图 6-3 SC_DBIOS_EN 双 BIOS 使能（PERI Subctrl） 73
图 6-4 SC_BSBC_BIOS_CFG 记录 BSBC 校验成功的 ESBC 分区寄存器（Sysctrl） 73
图 6-5 SC_WDOG_ST（Sysctrl） 73
图 6-6 SC_PORESET_CFG 校验分区镜像寄存器（Sysctrl） 73
图 6-7 Hi309a 网络场景启动流程 75
图 6-8 寄存器图 79
图 6-9 espi_out_dly_sel 域段配置参数 79
图 7-1 DAW 窗口更新示例 84
图 8-1 芯片应用总体示意图 90
图 8-2 AR 设备典型应用场景 91
图 8-3 LSW 快速转发业务芯片数据流图 92
图 8-4 基础网络转发业务处理流程 93
图 8-5 快速加解密转发业务处理流程 94
图 8-6 增强转发业务处理流程 95

版权所有 © 海思技术有限公司

图 9-1 Hi309a 的总线拓扑 102
图 9-2 业务模块接口架构图 103
图 9-3 Subsys 的总线接入 106
图 9-4 QoS 传递示意 107
图 9-5 对外反压流控 110
图 10-1 Hi309a 电源合并方案 122
图 10-2 电源域 123
图 10-3 时钟域 123
图 10-4 PMCTRL 结构框图 124
图 10-5 Q-Channel 接口 125
图 10-6 Q-Channel 接口时序图 125
图 10-7 Q-Channel 控制器状态机 126
图 10-8 P-Channel 接口 127
图 10-9 P-Channel 接口时序图 127
图 10-10 Core P-Channel 控制器状态跳转 128
图 10-11 单 Core 上下电路流程 129
图 10-12 温度控制 131
图 10-13 Hi309a AVS 方案图 132
图 10-14 不同 PC 芯片电压-频率对应图 133
图 10-15 DATASLICE SHUT DOWN 参考代码 1 167
图 10-16 DATASLICE SHUT DOWN 参考代码 2 168
图 10-17 DATASLICE SHUT DOWN 参考代码 3 169
图 10-18 DATASLICE RESTART 参考代码 1 169
图 10-19 DATASLICE RESTART 参考代码 2 170
图 10-20 DATASLICE RESTART 参考代码 3 171
图 10-21 DATASLICE RESTART 参考代码 4 171
图 11-1 系统 DFX 应用图 182
图 11-2 ALL-SCAN 时钟控制图 199
图 11-3 ALL-SCAN 配置流程 200
图 12-1 GIC 在系统中的位置 214
图 12-2 ITS 架构框图 216
图 12-3 GICD 中断上报流图 217
图 12-4 跨 GICD 间中断上报流图 219
图 12-5 ITS 中断上报流图 220
图 12-6 GICR 中断上报流图 222
图 12-7 亲和性层次图 225
图 12-8 1/N SPI 中断亲和性路由示例 226
图 12-9 1/N SPI 中断亲和性路由示例 227
图 12-10 N/N SPI 路由配置示例 228
图 12-11 SGI 中断亲和性路由示例 228
图 12-12 N/N SGI 中断亲和性路由示例 229
图 12-13 中断嵌套示意图 231

版权所有 © 海思技术有限公司

图 12-14 Priority grouping by binary point 231
图 12-15 物理中断配置命令交互流程 240
图 12-16 虚拟中断配置命令交互流程 241
图 13-1 SysCtrl 应用图 263
图 13-2 DJTAG Master 应用框图 267
图 13-3 System Counter 结构框图 269
图 13-4 ECPRI 场景功放保护框图 271
图 13-5 CPRI 场景功放保护框图 272
图 13-6 NAND 掉电保护 273
图 14-1 LLC 应用图 278
图 14-2 LLC 内部结构示意图 279
图 15-1 SRAM 应用框图 310
图 16-1 smmu structure 316
图 16-2 Linear Stream Table 322
图 16-3 2-level Stream Table with SPLIT=8 323
图 16-4 一级 STE 和单个 CD 结构的 Context Bank 323
图 16-5 一级 STE 和 Linear Context Descriptor 结构 324
图 16-6 混合结构，2-level STE，各个 STE 指向的 CD 结构可以不同 324
图 16-7 4k 页表 Stage1 转换流程举例 325
图 16-8 Stage1+Stage2 地址转换流程 326
图 16-9 Configuration and translation lookup sequence 327
图 16-10 Command 数据结构 1 330
图 16-11 Command 数据结构 2 330
图 16-12 Command 数据结构 3 330
图 16-13 Command 数据结构 4 330
图 16-14 Command 数据结构 5 330
图 16-15 Command 数据结构 6 330
图 16-16 Command 数据结构 7 330
图 16-17 Command 数据结构 8 331
图 16-18 Command 数据结构 9 331
图 16-19 Command 数据结构 10 331
图 16-20 Command 数据结构 11 331
图 16-21 Command 数据结构 12 331
图 16-22 Command 数据结构 13 331
图 16-23 Command 数据结构 14 331
图 16-24 Command 数据结构 15 331
图 16-25 Command 数据结构 16 332
图 16-26 Command 数据结构 17 332
图 16-27 Command 数据结构 18 332
图 16-28 Command 数据结构 19 332
图 16-29 Command 数据结构 20 332
图 16-30 Command 数据结构 21 332

 版权所有 © 海思技术有限公司

图 16-31 Command 数据结构 22 332
图 16-32 Command 数据结构 23 332
图 16-33 Event 数据结构 1 334
图 16-34 Event 数据结构 2 334
图 16-35 Event 数据结构 3 334
图 16-36 Event 数据结构 4 334
图 16-37 Event 数据结构 5 334
图 16-38 Event 数据结构 6 335
图 16-39 Event 数据结构 7 335
图 16-40 Event 数据结构 8 335
图 16-41 Event 数据结构 9 335
图 16-42 Event 数据结构 10 335
图 16-43 Event 数据结构 11 336
图 16-44 Event 数据结构 12 336
图 16-45 Event 数据结构 13 336
图 16-46 Event 数据结构 14 336
图 16-47 Event 数据结构 15 336
图 16-48 Event 数据结构 16 337
图 16-49 Streaming Ordered WriteUniques transaction flow 337
图 16-50 IO-Cache 338
图 16-51 STE 数据结构 341
图 16-52 Context Descriptor 数据结构 349
图 16-53 SMMU 整体透传流程 359
图 16-54 SMMU 非透传属性转换流程 364
图 16-55 TLB 同步流程示意图 366
图 16-56 PRIQ page request 格式 371
图 17-1 Timer 中断产生时序图 381
图 18-1 TSensor 应用图 388
图 20-1 Ring 拓扑结构图 399
图 20-2 Link 握手过程 399
图 20-3 Ring 数据传输过程 400
图 20-4 传输的汇聚与分发 400
图 20-5 Ring 上各节点的循环传输 402
图 20-6 DAW 配置有效值 404
图 21-1 Dispatch 应用图 413
图 22-1 Scheduler 应用图 420
图 23-1 MN 在系统中的位置与交互 424
图 24-1 HHA 应用图 436
图 25-1 eFuse 应用图 450
图 26-1 FTE 结构示意图 456
图 27-1 Arm 的仿真器硬件 463
图 27-2 Ds-5 软件界面 464

 版权所有 © 海思技术有限公司

图 27-3 仿真器连接示意图。 465
图 27-4 Hi309a 芯片的 CPU CLUSTER coresight 系统 467
图 27-5 Timestamp generator 471
图 27-6 CNTCR 472
图 27-7 CNTSR 473
图 27-8 CNTCVL 473
图 27-9 CNTCVU 474
图 27-10 CNTFID0 475
图 27-11 各模块偏移地址 476
图 27-12 Effect of trace configuration 477
图 27-13 TPIU 寄存器 478

版权所有 © 海思技术有限公司

表格目录

表 1-1 功能描述 2
表 1-2 Hi309a 封装尺寸和订购须知 10
表 2-1 关键术语 11
表 4-1 关键术语 26
表 4-2 时钟列表 27
表 4-3 PLL 控制信号列表 29
表 4-4 PLL 配置参数说明 32
表 4-5 网络同步以太链路模式对应分频系数配置表 33
表 5-1 关键术语 36
表 5-2 复位信号描述 38
表 5-3 各种类型的复位范围描述 38
表 5-4 模块软复位信号 44
表 5-5 CPU 异常复位时转发通路相关模块时钟/复位处理 46
表 6-1 系统初始化关键术语描述 51
表 6-2 AP（A55）启动真值表 55
表 6-3 启动模式真值表 57
表 6-4 AP 侧启动真值表 58
表 6-5 HISS 侧启动真值表 60
表 6-6 启动晶振频率真值表 62
表 6-7 SFC 接口时钟频率真值表 62
表 6-8 NAND BOOT 模式真值表 62
表 6-9 NANDC 启动模式控制 70
表 6-10 Hi309a 硬件双 BOOT 启动方案 71
表 6-11 启动备份大小真值表 71
表 6-12 启动校验分区真值表（软件使用） 74
表 7-1 Memory Map 关键术语描述 81
表 7-2 VF 空间地址配置说明 85
表 7-3 HAC 索引号 85
表 8-1 系统性能关键术语描述 87
表 9-1 系统反压和流量控制关键术语描述 97
表 10-1 System Power 术语表 119
表 10-2 TPS544C24 配置表 134
表 10-3 VT7410 配置表 137

版权所有 © 海思技术有限公司

表 10-4 HPM_RELATED_CFG0 寄存器 156
表 11-1 系统 DFX 术语表 172
表 11-2 probe 信号列表 183
表 11-3 probe signal 分配关系 183
表 12-1 GIC 关键术语描述 209
表 12-2 组件之间通过总线传递消息的通达关系 214
表 12-3 GICD 中断说明 217
表 12-4 GICD 中断路由配置 218
表 12-5 支持 GICD 间转移中断类型说明 219
表 12-6 GICDn 中断路由配置 220
表 12-7 ITS 中断产生说明 221
表 12-8 ITS 中断路由配置 221
表 12-9 GICR 中断源说明 222
表 12-10 GICR 中断配置 223
表 12-11 GIC Device ID 223
表 12-12 GICR 和 GICD 地址分配 223
表 12-13 GIC 支持的 MSI 中断接口 224
表 12-14 GICD 中断亲和路由算法 225
表 12-15 安全软件视角中断优先级配置 230
表 12-16 非安全软件视角中断优先级配置 230
表 12-17 GICR_TYPER_H 寄存器（Re-distributor affinity 层级） 237
表 12-18 GICR_TYPER_L 寄存器（Re-distributor） 238
表 12-19 ITS DFX 寄存器地址 251
表 12-20 ITS 错误编码 0 251
表 12-21 ITS 错误编码 1 254
表 12-22 ITS 错误编码 2 256
表 12-23 Device/VCPU/PCPU Table Pointer Table Entry 数据结构表 257
表 12-24 Device Table Entry 数据结构表 258
表 12-25 ITT Table Entry 数据结构表 258
表 12-26 VCPU Table Entry 数据结构表 259
表 12-27 Collection Table Entry 数据结构表 259
表 13-1 SysCtrl 关键术语描述 261
表 13-2 信号名称说明 270
表 13-3 电源异常判定真值表 273
表 13-4 写保护判定真值表 273
表 14-1 llc 术语表 275
表 14-2 LLC Cacheline 压缩的 Pattern 284
表 14-3 LLC 内部统计事件功能列表 285
表 14-4 LLC 入口队列状态寄存器列表（注：高有效） 288
表 14-5 LLC 处理队列状态寄存器列表（注：高有效） 288
表 14-6 LLC 处理队列状态寄存器列表（注：高有效） 289
表 14-7 LLC 处理队列状态寄存器列表（注：高有效） 290

 版权所有 © 海思技术有限公司

表 14-8 功能统计 304
表 15-1 SRAM 术语表 309
表 16-1 Description of SMMU key terms 320
表 16-2 Skyros MemAttr 328
表 16-3 Skyros SnpAttr 328
表 16-4 属性强弱表 328
表 16-5 Atomic 命令类型 340
表 16-6 Level1 Stream Table Descriptor 数据结构 342
表 16-7 Stream Table Entry 数据结构 342
表 16-8 Context Descriptor 数据结构 349
表 16-9 V8 架构粒度 T*SZ 配置表 356
表 16-10 V8 架构粒度 T*SZ 配置表 357
表 16-11 V8 架构粒度 T*SZ 配置表 357
表 16-12 acfg 转换表 358
表 16-13 ACFG 转换表 359
表 16-14 透传 mtcfg 为 0 时输出 Memory 属性转换表 360
表 16-15 Mattr 属性转换表 360
表 16-16 透传 mtcfg 为 1 时输出 Memory 属性转换表 361
表 16-17 STE.ALLOCCFG 转换表 361
表 16-18 STE.MTCFG 为 0 时输出 Memory 属性转换表 362
表 16-19 Mattr 属性转换表 362
表 16-20 STE.MTCFG 为 1 时输出 Memory 属性转换表 363
表 16-21 SMMU 定义下的 SMMU_(S)_CMDQ_CONS.ERR 367
表 16-22 AXI 非保序命令的数据规格 375
表 17-1 Timer 术语表 380
表 17-2 Timer 计数范围和精度表 381
表 17-3 TIMERx_LOAD 寄存器描述 383
表 17-4 TIMERx_BGLOAD 寄存器描述 383
表 17-5 TIMERx_CONTROL 寄存器描述 384
表 18-1 TSensor 关键术语描述 388
表 18-2 TSensor 预留 2 组 11-bit eFuse 做两点温度校准 389
表 19-1 WDOG 关键术语描述 391
表 20-1 Cross Station 术语表 398
表 20-2 Cross Station 内部统计事件功能列表 405
表 20-3 Cross Station 存取延迟（Latency） 410
表 21-1 延迟统计 418
表 22-1 Scheduler 关键术语描述 419
表 22-2 AXI Master ID 数值 422
表 22-3 Scheduler 统计事件功能列表 422
表 22-4 Scheduler 模块各通道传输延迟 423
表 23-1 MN 内部统计事件功能列表 430
表 23-2 MN 状态寄存器列表（注：高有效） 431

 版权所有 © 海思技术有限公司

表 23-3 MN 广播延迟（Latency）.....432
表 24-1 Hydra HA 关键术语描述.....435
表 24-2 MN 内部统计事件功能列表.....439
表 24-3 MNBuffer 状态寄存器列表（注：高有效）.....440
表 24-4 Special ID 配置具体说明.....442
表 24-5 HHA 存取延迟（Latency）.....443
表 24-6 HHA 内部统计事件功能列表.....443
表 24-7 MN 内部统计事件功能列表.....445
表 24-8 MN 状态寄存器列表（注：高有效）.....445
表 24-9 HHA 中断向量表.....446
表 25-1 eFuse 关键术语描述.....449
表 26-1 FTE 关键术语描述.....456
表 26-2 FTE 定义的包格式.....459
表 27-1 CoreSight 关键术语描述.....466
表 28-1 IPC 中断映射表.....480
表 29-1 SEM 中断映射表.....496
表 29-2 错误类型说明.....498

 版权所有 © 海思技术有限公司

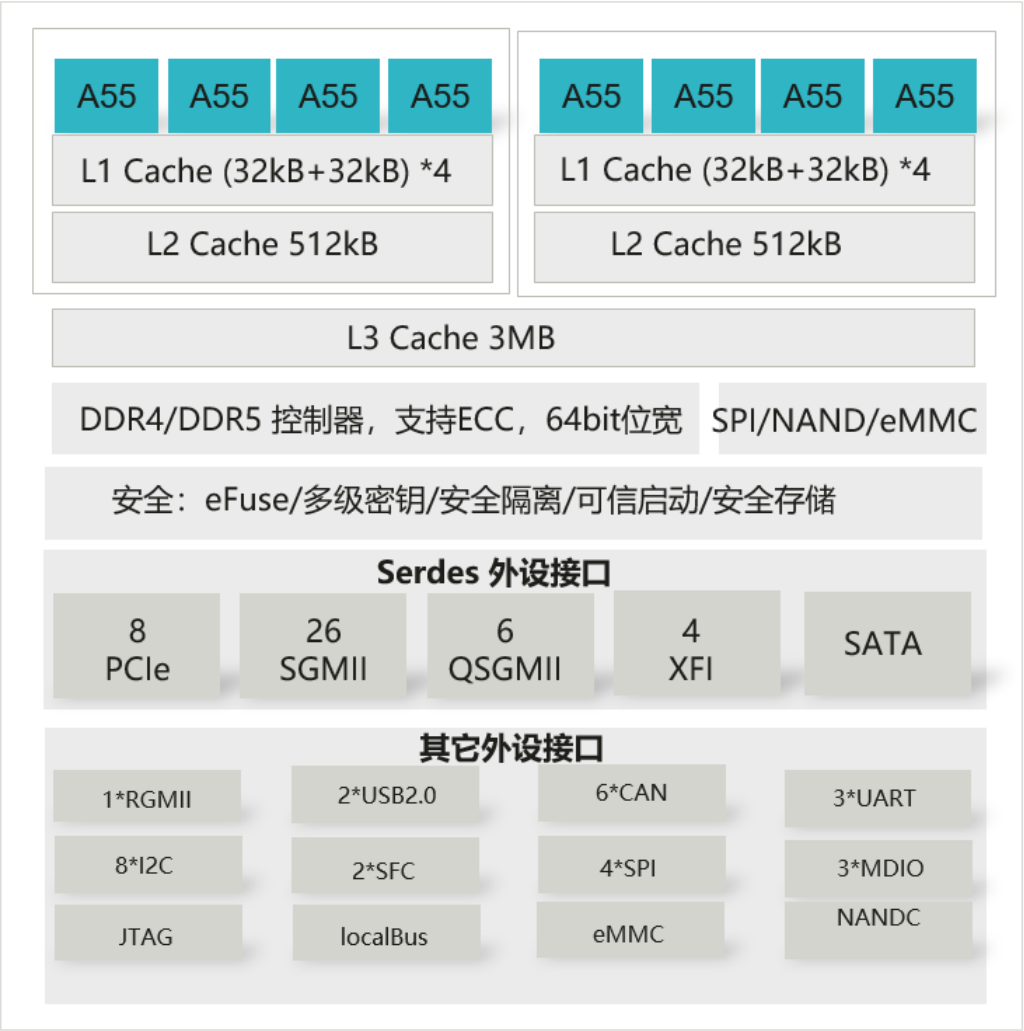
1 产品简介

1.1 产品族

Hi309a是海思面向通信、工控等领域推出的多核处理器方案，主要应用于交换机、网关等网络设备场景，包括两组最高支持2.2GHz的四核A55处理器，总计八核A55。集成了1通道DDR4，速率最高可达2666/3200MT/s；为了减小操作延迟和保证数据一致性，提供了多级Cache结构和基于硬件的全局数据一致性解决方案；为了加速业务处理能力及留出更多的CPU处理性能以支持灵活的业务扩展，集成了网络通讯应用所需要的多种加速引擎，包括硬件包解析、硬件队列管理、硬件QoS管理、硬件缓存管理，及IPSec、HPRE安全处理等。集成了包括XGE、GE等在内多种网络接口，支持板内、背板、光模块等多种连接；此外还集成了丰富的通用处理器接口，包括PCIe、USB、SATA、Local Bus、SPI、UART等。

版权所有 © 海思技术有限公司

图 1-1 整芯片逻辑图



1.2 功能描述

表 1-1 功能描述

特性	规格
处理器	Cortex-A55，8*2.2GHz
Cache	L1：32kB+32kB独占 L2：每Cluster内共享512kB L3：共享3MB
DDR控制器	64bit DDR4存储器接口 DDR4最高数据率2933Mbps

版权所有 © 海思技术有限公司

特性	规格
FLASH接口	1个NANDC接口，可以对接NAND FLASH 1个eMMC接口 2个SFC接口，可以对接SPI FLASH
Serdes接口	
PCIe	支持8个PCIe接口，支持8*X1和2*X2和1*X4，每Lane速率支持5G/2.5Gbps
SGMII	最多支持26个SGMII GE接口
QSGMII	最多支持6个QSGMII
XFI	最多支持4个10GE（BASE-R/BASE-R with BRFEC）
SATA	2个SATA接口
RGMII	1个独立的RGMII接口
通用接口	
USB接口	2个USB2.0 Host接口（可兼容USB3.0 Device）
CAN	6个CANbus 2.0接口
SPI	4个SPI接口，每个SPI提供2个片选
其他接口	UART/I2C/JTAG/localBus/MDIO

1.2.1 处理器特性

- 完整支持ARMv8.2-A A64，A32和T32指令集。
- 支持RAS extension，ARMv8.3-A extension LDAPR指令，ARMv8.4-A extension Dot Product指令。
- 所有异常等级（EL0-3）支持2种执行态（AArch32和AArch64）。
- 支持顺序执行的流水线。
- 支持direct和indirect分支预测。
- 每个CPU core支持容量32kB的一级指令Cache和32kB的一级数据Cache。
- 每个CPU core支持容量128kB的二级Cache。
- L1和L2 Cache支持ECC或奇偶校验。
- 集成2级页表硬件转换MMU，支持虚拟地址到物理地址的映射。
- 支持TrustZone安全技术。
- 支持SIMD和浮点指令。
- 支持Cryptographic指令。
- 内部集成了GIC CPU 接口，支持外接GICv3。
- 支持外接64位系统计数器。

版权所有 © 海思技术有限公司

- 指令仅支持小端模式，数据支持大小端模式可配置。
- 支持16T物理地址空间（44位物理地址）。
- 基于单线程结构。

1.2.2 存储特性

- 支持3级Cache，每个CPU Core集成一个32kB大小的一级指令Cache和一个32kB大小的数据Cache；CPU Cluster（含4个CPU Core）集成一个512kB大小的二级Cache。
- 集成3MB的L3 Cache。
- 支持基于Skyros协议的互联，基于MOESI一致性协议，以Directory结构支持片内L1/L2 Cache、Memory、IO、Accelerator的完全数据一致性互联。
- 提供1个64 bits DDR4存储器接口，DDR4最高数据率2933Mbps（2rank支持最高速率为2666Mbps）。
- 支持全局的双指令（Exclusive）原子操作，可完成加、减、与、或、SWAP读-改-写原子操作。
- 提供256kB的片上SRAM，SRAM额外支持加、减、与、或的单指令原子操作加速（单次写完成原子操作），支持配置固定空间作为片上安全存储。
- 提供独立的QDMA单元，加速各Memory/IO之间的数据搬运。
- 全系统存储空间的访问支持QoS进行流量管理。
- 提供独立的内存管理单元BMU，支持缓存动态申请和释放，BMU以链表存储块的方式管理内存，并针对报文存储资源提供基于通道和端口的反压与丢弃。

1.2.3 加速器特性

- 提供统一的包保序模块POE，用以保证系统中Pipeline或RTC模式处理下包的顺序，并均衡各个处理单元的包处理负载。
- 提供硬件包解析模块PPE，用以在数据通路的GE/XGE接口线速对包基本类型进行解析，将解析结果按照规定格式写入包描述符中，可根据系统缓存状态，对高低优先级报文在端口上采用不同的丢弃策略。
- PPE模块集成44Gbps的LAN Switch，支持基本的L2特性，部分L2+特性；支持MAC侧26个端口，L2表最大支持6k深度，支持入口和出口流表共享768*80b的TCAM表项。
- 提供统一的流量管理模块TM，该模块可用于系统出口，实现出口TM功能；可对数据流进行QoS处理、队列管理和调度，完成尾丢弃、WRED/RED算法，避免队列拥塞，完成流量整形，包超时丢弃。
- 提供硬件加解密加速模块SEC，可加速认证、数据加解密等安全应用，包括IPSec ESP、AH等协议的安全算法处理；模块提供了DES/3DES、AES、SHA-1、SHA-256/224和MD5算法，及基于SHA-1、SHA-256/224、MD5的HMAC运算、基于AES-CBC的MAC运算、基于AES的CCM运算，DES/3DES和AES分别支持ECB、CBC、CFB、OFB和ECB、CBC、CFB、OFB、CTR几种工作模式。
- 提供模幂运算加速模块HPRE（High Speed RSA Engine），可为RSA加解密（含签名认证）、RSA密钥对生成、DH密钥交换提供硬件加速。

1.2.4 接口描述

- 最多支持4个10GE（BASE-R/BASE-R with BRFEC）。

版权所有 © 海思技术有限公司

- 最多支持26个GE接口（Hi309a可用的serdes lane只有10个），不支持半双工模式；1000BASE-X/SGMII支持提频到3.125Gbps。
- 支持6路QSGMII与2个XGE并发。
- 最大支持4路5GBASE-R/ 5GBASE-KR。
- 支持8个PCIe接口，支持8*X1和2*X2和1*X4，每Lane速率支持5G/2.5Gbps。
- 支持10 Lane Serdes，由XGE、GE、PCIe复用。
- 提供丰富的通用/调试接口包括：
  - 1个NANDC接口，可以对接NAND FLASH。
  - 1个eMMC接口。
  - 2个SFC接口，可以对接SPI FLASH。
  - 3个独立的10M/100Mbps MII/RMII接口。
  - 1个独立的RGMII接口。
  - 2个USB 2.0 Host接口（可兼容USB3.0 Device）。
  - 6个CANBUS2.0接口。
  - 2个UART接口。
  - 10个I2C接口。
  - 4个SPI接口，每个SPI提供4个片选。
  - 7个ESPI接口，提供2个片选。
  - 2个HDLC接口。
  - 4个TDM接口。
  - 1个JTAG接口，支持IEEE1149.1/IEEE1149.6边界扫描，CPU软件调试及芯片内硬件调试，通过JTAG接口进行芯片内硬件调试。
  - 1个AVS电源接口，支持PMBUS1.2/IMVP6.5电源协议。
  - 3个MDIO接口。

1.2.5 安全特性

- Application Processor
  - ARMv8架构处理器核，支持Trustzone扩展，即可实现安全/非安全模式切换，只有安全核且访问地址的页表属性为安全（NSbit为0），才能对该地址发起安全访问（总线NSbit为0）；本文为了表述方便，简称为AP，并且与HISS区分开。
- HISS
  - 独立增强安全子系统，外部无法访问子系统内部。
  - HISS内集成1个CORTEX-M3架构处理器核SC300。
  - HISS内集成抗侧信道攻击的密码算法引擎、密钥管理模块以及SOC设计。
  - HISS启动和运行均独立于主系统。
  - 具备独立的安全启动ROM code。
  - 具备独立的安全固件或OS。
  - 具备向主系统提供秘钥申请、身份认证等安全应用服务的能力。
  - 支持部署安全启动、软件/固件/密钥/证书的安全升级。
  - 支持镜像加解密和安全校验。

版权所有 © 海思技术有限公司

  - 结合高性能CryptoEngine实现业务密钥加密、业务数据安全存储。
  - JTAG调试功能支持Hi309a通过eFuse单独关闭。
- 片上存储体系
  - FLASH对于安全而言，主要是存储需要掉电保存的加密过的关键信息，但是FLASH本身不具备安全特性。
  - NANC具备按区域划分访问权限控制的功能。
  - IFC具备按区域划分访问权限控制的功能。
  - DDR通过HHA提供了数据存储的安全性划分。
- GIC
  - 支持ARM AArch64安全扩展的中断管理控制器。
- TZPC
  - TZPC属于安全世界设备，通过TZPC可以将外设配置为安全、非安全或安全可配置设备，分别控制安全访问权限，Hi309a只是控制了TIMER。
- CryptoEngine
  - 对称密码算法加速器SEC：

    用于完成对称密码学相关的运算。
  - 非对称密码算法加速器HPRE（High Performance RSA and ECC engine）：

    用于完成RSA、ECC等非对称密码学相关的运算。
- JTAG-AUTH
  - JTAG鉴权加速器，用于控制JTAG调试开关。
- WatchDog Timer
  - 整个系统提供2个安全WDT（系统和HISS各一个）。

1.2.6 遵从的标准与协议

Hi309aV100支持以下标准协议：

- PCIe遵从如下标准：
  - PCI Express Base Specification Revision 2.1 Specification.09/2006
  - PIPE Specification for PCI Express 2.0 Version. 07/2007
- DDRC遵从如下标准：
  - JESD79-4；Sep 2012
  - JESD79-3E；July 2010
- RSA遵从如下标准：
  - FIPS PUB 186-3 Digital Signature Standard；June 2009
  - RFC3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)；May 2003
  - RFC2631 Diffie-Hellman Key Agreement Method；June 1999
  - PKCS #1 v2.1: RSA Cryptography Standard；June 2002
- SEC遵从如下标准：
  - 算法标准：
    - FIPS PUB 46-3 DATA ENCRYPTION STANDARD(DES)；Oct 1999

版权所有 © 海思技术有限公司

- FIPS PUB 197 ADVANCED ENCRYPTION STANDARD(AES)；Nov 2001
- FIPS PUB 180-2 SECURE HASH SIGNATURE STANDARD(SHA)；Aug 2002
- FIPS PUB 198 The Keyed-Hash Message Authentication Code(HMAC)；Mar 2002
- NIST Special Publication 800-38A (2001): "Recommendation for Block Cipher Modes of Operation"
- NIST Special Publication 800-38B (2001): "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication"
- RFC 1321 The MD5 Message-Digest Algorithm；Apr 1992
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication；Feb 1997
- RFC 2202 Test Cases for HMAC-MD5 and HMAC-SHA-1；Sep 1997
- RFC 2403 The Use of HMAC-MD5-96 within ESP and AH；Nov 1998
- RFC 2404 The Use of HMAC-SHA-1-96 within ESP and AH；Nov 1998
- RFC 3566 The AES-XCBC-MAC-96 Algorithm and its Use With Ipsec；Sep 2003
- RFC 4493 The AES-CMAC Algorithm；Jun 2006
- RFC4309 Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)；Dec 2005
- RFC 4106 The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)；Jun 2005
- RFC 2451 The ESP CBC-Mode Cipher Algorithms；Nov 1998
- RFC 3602 The AES-CBC Cipher Algorithm and its Use with IPsec；Sep 2003
- RFC 3686 Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)；Jan 2004
- RFC 2405 The ESP DES-CBC Cipher Algorithm With Explicit IV；Nov 1998
- 3GPP TS 35.215: "Confidentiality and Integrity Algorithms UEA2 & UIA2；Document 1: UEA2 and UIA2 specifications"(SNOW 3G)
- 3GPP TS 35.202: 3G Security；Specification of the 3GPP confidentiality and integrity algorithms；；Document 2。

– 协议标准：

版权所有 © 海思技术有限公司

  - RFC4301 Security Architecture for the internet Protocol；Dec 2005
  - RFC4303 IP Encapsulating Security Payload(ESP)；Dec 2005
  - RFC4305 Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)；Dec 2005
  - 3GPP TS 36.323 V10.1.0 (2011-03) Technical Specification
  - 3GPP TS 33.401 V9.6.0 (2010-12) Technical Specification。

- GE & XGE& PPE & SPPE遵从如下标准：
  - HiTag
  - RFC894: A Standard for the Transmission of IP Datagrams over Ethernet Networks, Apirl 1984
  - RFC2516: A Method for Transmitting PPP Over Ethernet (PPPoE), February 1999
  - IEEE Std 802.3-2008: Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, December 2009
  - IEEE Std 802.1Q-2005: Virtual Bridged Local Area Networks, May 2006
  - IEEE Std 802.1ah-2008: Virtual Bridged Local Area Networks Amendment 7: Provider Backbone Bridges, August 2008
  - IEEE Std 802.1ad-2005: Virtual Bridged Local Area Networks Amendment 4: Provider Bridges, May 2006
  - IEEE Std 802.3ah-2004: Amendment: Media Access Control
  - Parameters, Physical Layers, and Management Parameters for Subscriber Access Networks, September 2004
  - IEEE Std 802.1ag-2007: Virtual Bridged Local Area Networks
  - Amendment 5: Connectivity Fault Management, December 2007
  - IEEE Std 802.1AB-2009: Station and Media Access Control Connectivity Discovery, September 2009
  - ITU Y.1731-200802: OAM functions and mechanisms for Ethernet based networks, 02/2008
  - RFC3031: Multiprotocol Label Switching Architecture, January 2001
  - RFC3032: MPLS Label Stack Encoding, January 2001
  - RFC826: An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware, November 1982
  - RFC903: A Reverse Address Resolution Protocol, June 1984
  - RFC791: INTERNET PROTOCOL, September 1981
  - RFC2460: Internet Protocol, Version 6 (IPv6) Specification, December 1998
  - RFC1701: Generic Routing Encapsulation (GRE), October 1994
  - RFC2784: Generic Routing Encapsulation (GRE), March 2000
  - RFC4302: IP Authentication Header, December 2005

版权所有 © 海思技术有限公司

  - RFC4303: IP Encapsulating Security Payload (ESP), December 2005
  - RFC793: TRANSMISSION CONTROL PROTOCOL, September 1981
  - RFC768: User Datagram Protocol, August 1980
  - RFC4960: Stream Control Transmission Protocol, September 2007
  - RFC792: INTERNET CONTROL MESSAGE PROTOCOL, September 1981
  - RFC4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, March 2006
  - RFC3376: Internet Group Management Protocol, Version 3, October 2002
  - RFC2131: Dynamic Host Configuration Protocol, March 1997
  - RFC951: BOOTSTRAP PROTOCOL (BOOTP), September 1985
  - 3GPP TS 29.060 V10.1.0 (2011-03): GPRS Tunnelling Protocol(GTP) across the Gn and Gp interface(Release 10)
  - IEEE Std 1588-2008: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, July 2008
  - ITU Q.922-199202: ISDN DATA LINK LAYER SPECIFICATION FOR FRAME MODE BEARER SERVICES, 1992
- JTAG遵从如下标准：
  - 1149.6TM IEEE Standard for Boundary-Scan Testing of Advanced Digital Networks

1.3 电气特性

Hi309aV100芯片具有以下电气特性：

- 芯片Core电压0.8V、1.0V。
- I/O电压如下：
  - DDR4：1.2V。
  - RGMII/MII0：1.8V。
  - MDIO/MII1：1.8V。
  - PLL：1.8V。
  - 其他：1.8V。
- 芯片功耗 ≤15W。
- 封装FCBGA，尺寸25mmx25mm。
- 可工作环境温度：-40℃~+85℃。
- 长期工作结温：-40℃~+105℃。

1.4 应用领域

Hi309aV100芯片适用于以下领域：

- 网络AR/FW/WLAN A等企业网表、园区线卡和盒式主控控制面应用。
- 嵌入式工控MPU场景。
- 通用MPU场景。

版权所有 © 海思技术有限公司

1.5 订购须知

表 1-2 Hi309a 封装尺寸和订购须知

产品型号	下单型号	丝印	封装类型
Hi309a	PanguM900NPHi309a	pangu M900 NP	FCBGA 25mm×25mm

版权所有 © 海思技术有限公司

2 全局数据流

2.1 关键术语

如表2-1所示术语将在本章节表示特殊的含义。

表 2-1 关键术语

术语	描述
HRPE	高性能RSA和ECC非对称加解密算法引擎。
POE	Packet Order Enforcer，包保序模块。
PPE	Packet Parse Engine，包解析引擎模块。
SPPE	Software Packet Parse Engine。
SA	Sevice Awareness，业务感知。
SEC	Security Engine，对称加解密算法引擎。

2.2 概述

本文档主要介绍Hi309a芯片各种应用场景下报文数据处理的典型流程，以便读者对芯片内各个顶层模块之间的配合有一个快速的了解。

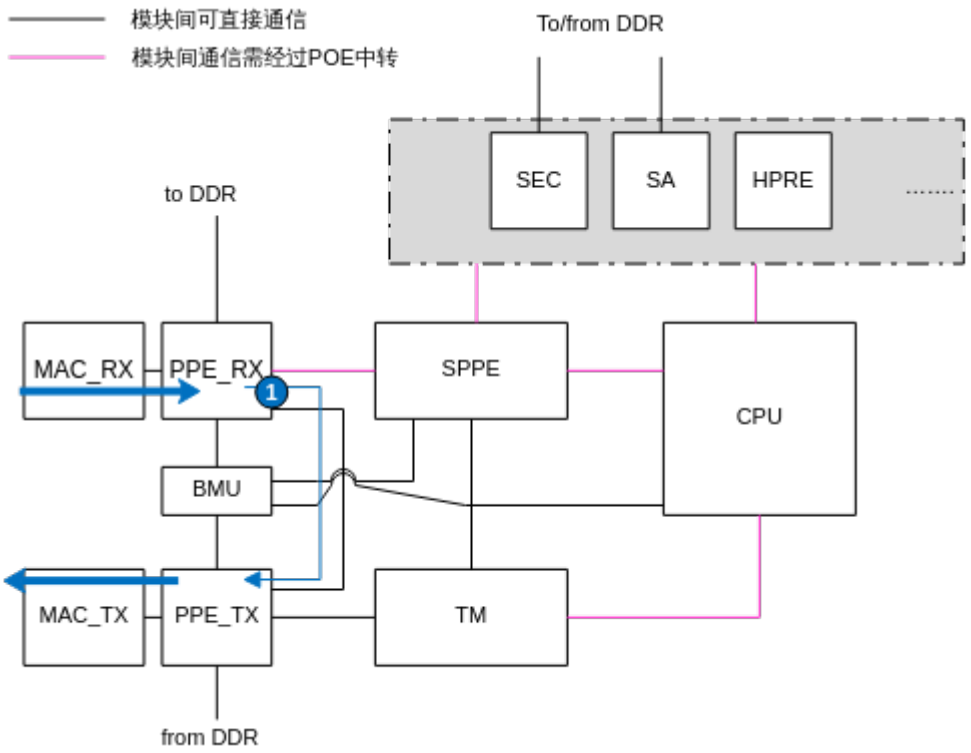
2.3 LSW 快速转发业务

典型应用：AR产品作为LAN侧的LSW/HUB使用，完成LAN侧交换/本地L3网关等基础业务。

仅适用对于表项规格要求较低的简单业务（有k级别MAC表，ACL实现LPM），可以由PPE直接完成，如图2-1所示。

版权所有 © 海思技术有限公司

图 2-1 LSW 快速转发业务处理流程



接收到报文后，PPE对报文进行解析，对LSW快速转发业务，PPE直接完成包头编辑然后发送出去。

2.4 基础网络转发业务

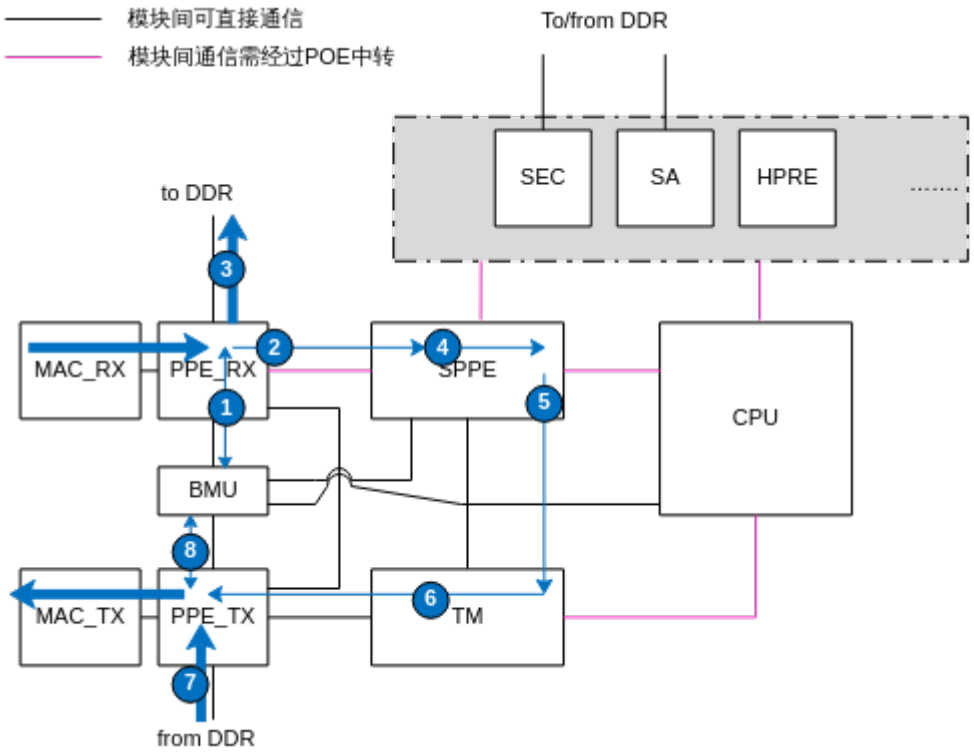
典型应用：AR/FW/WLAN AC产品，实现基础应用业务（对外测试基本指标），包括包转发（AR，逐包进行）和流转发（FW/WLAN AC，逐流进行）两类，基础转发业务主要由SPPE完成。对于流转发，在SPPE建立流表之前的首N个报文需要上送到CPU进行处理，完成后下发流表到SPPE，之后的报文在SPPE查流表命中直接转发。

基础转发业务主要包括以下典型业务：

- 二层bridge类业务（L2 bridge）
- 三层IPv4/v6双栈满规格基本转发传统路由场景
- CAPWAP+L2/L3
- MPLS业务（支持p/pe/6pe基本转发场景，支持ldp/l2vpn/l3vpn/te/frr等业务流程）
- IPv4/v6 + NAT + FW 快速转发

版权所有 © 海思技术有限公司

图 2-2 基础网络转发业务处理流程



步骤1 PPE预先向BMU申请好内存Buffer缓存。

步骤2 接收到报文后，PPE对报文进行解析，并生成包描述符。

步骤3 PPE将解析结果随报文一起缓存到DDR，同时将包描述符送给POE。

步骤4 SPPE完成报文合法性检查，提取包头做各种转发查找处理。

步骤5 SPPE生成报文的编辑命令，和包描述符一起交TM处理。

步骤6 TM完成调度、整形后将报文的包描述符推送给PPE。

步骤7 PPE从DDR中读出包数据，完成包头编辑后发送出去。

步骤8 PPE向BMU释放缓存。

----结束

2.5 快速加解密转发业务

典型应用：AR/FW/WLAN AC产品，实现快速加解密业务，主要由SPPE调用SEC引擎完成。

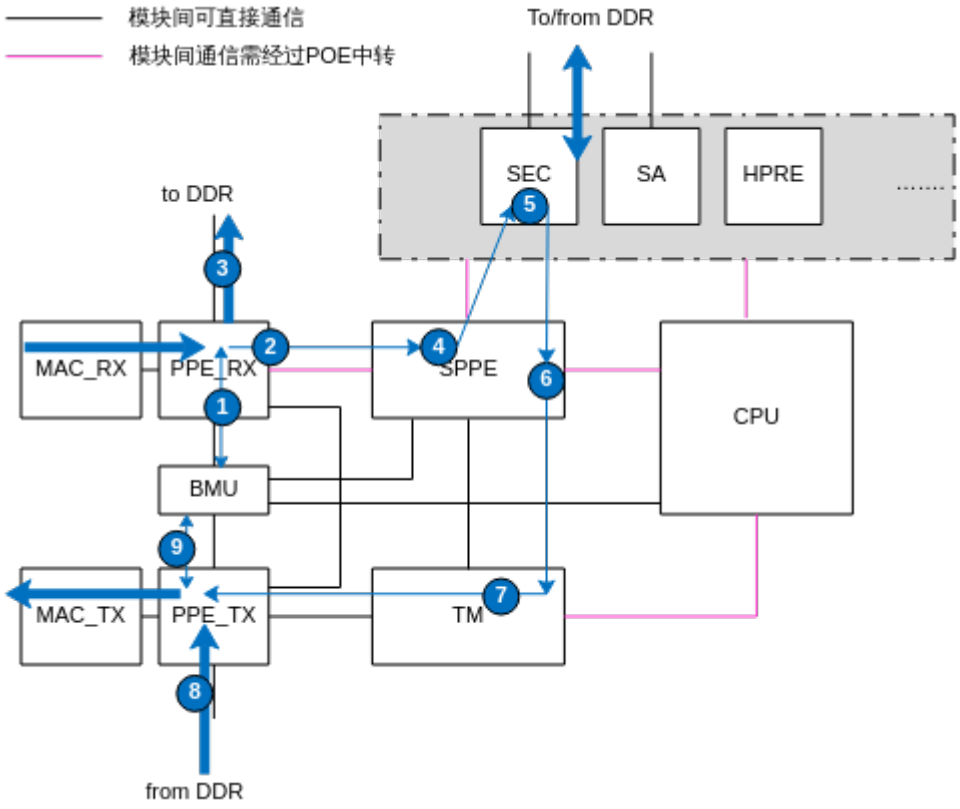
对于AR还是逐包转发；但对于FW/AC还是建流表场景，流表由CPU创建后下发到SPPE，SPPE的流表只负责转发行为，加解密是分立的独立操作，在流转发行为结束后调用SEC引擎进行。

快速加解密转发业务主要包括以下典型场景：

版权所有 © 海思技术有限公司

- IPSEC快转（ipv4/v6+ipsec）
- CAPWAP+DTLS（解密：AESCBC then SHA1 2次SEC，加密：SHA1 & AESCBC，1次SEC）+IPV4/V6
- SDWAN（支持dpi/net stream/dsvpn/ipsec/hqos/ipfpm/tcpfpm/spr等若干复杂业务的组合）

图 2-3 快速加解密转发业务处理流程



步骤1 PPE预先向BMU申请好内存Buffer缓存。

步骤2 接收到报文后，PPE对报文进行解析，并生成包描述符。

步骤3 PPE将解析结果随报文一起缓存到DDR，同时将包描述符送给POE。

步骤4 SPPE完成报文合法性检查，提取包头做各种转发查找处理。

步骤5 SPPE完成IPSec前处理后，将报文的包描述符推送给SEC加速器进行加密/解密处理；SEC从DDR中读取数据，完成加/解密后，将包描述符推送回SPPE处理。

步骤6 SPPE 完成 IPSEC 后处理，包括各种必要的合法性检查和转发查找处理，然后生成编辑命令，和包描述符一起交TM处理。

（WLAN AC的AESCBC需要重复**步骤5**~**步骤6**）

步骤7 TM完成调度、整形后将报文的包描述符推送给PPE。

步骤8 PPE从DDR中读出包数据，完成包头编辑后发送出去。

版权所有 © 海思技术有限公司

步骤9 PPE向BMU释放缓存。

----结束

2.6 增强转发业务

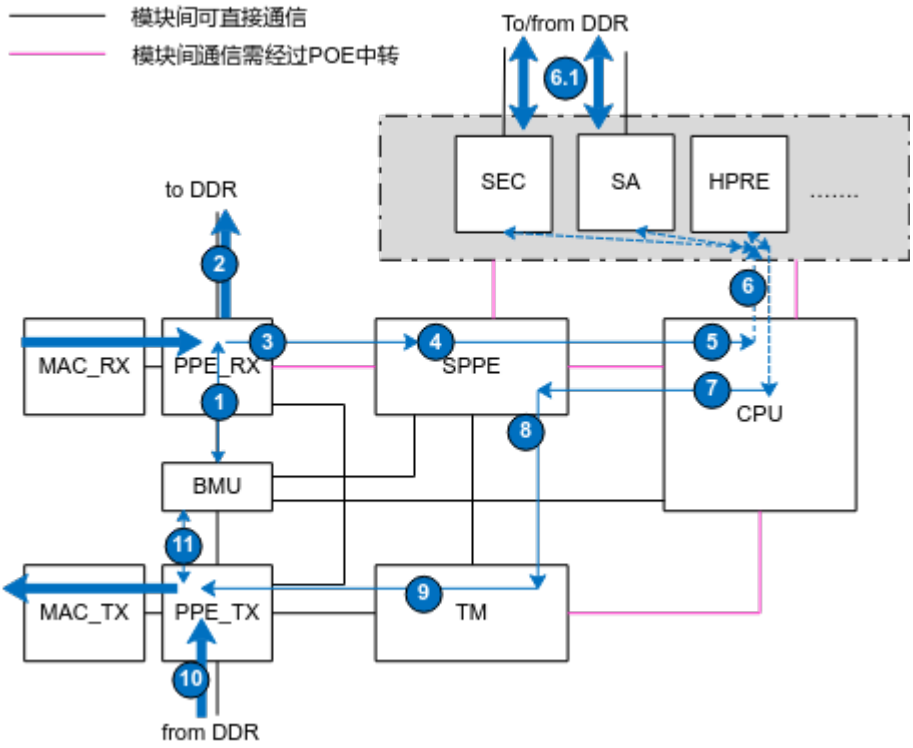
典型应用：AR/FW/WLAN AC产品，实现NGE业务，主要由CPU调用SEC引擎完成。SPPE一般做前处理，CPU处理完后数据一般不会再回SPPE。

对一些复杂业务，SPPE不能单独完成，要有CPU参与处理。增强转发业务中主要的处理（通常在4层以上）由CPU完成，SPPE一般只做L2/L3处理（产品开发具体业务时，SPPE也可以做更多的处理，例如调用SEC加速器做加解密，但在本流程中不体现），如图2-4所示。FW业务基于IPS策略，分快转和满转，快转场景在SPPE建立流表之前的首N个报文需要上送到CPU进行处理，完成后下发流表到SPPE，之后的报文在SPPE查流表命中直接转发。如果不开快转，那么所有报文都在CPU完成。

增强转发业务主要包括以下典型场景：

- SDWAN（支持dpi/net stream/dsvpn/ipsec/hqos/ipfpm/tcpfpm/spr等若干复杂业务的组合）
- 基于包解析的数据流转发（NGFW/IPS）
- CAPWAP+NGE
- Telemetry

图 2-4 增强转发业务处理流程



步骤1 PPE预先向BMU申请好内存Buffer缓存。

版权所有 © 海思技术有限公司

步骤2 接收到报文后，PPE对报文进行解析，并生成包描述符。

步骤3 PPE将解析结果随报文一起缓存到DDR，同时将包描述符送给POE。

步骤4 SPPE完成报文合法性检查，提取包头做各种转发查找处理。

（如果是快转业务，可以直接跳到**步骤8**）

步骤5 CPU收到报文后处理 。

步骤6 CPU按需要，可以调用加速器协助处理，可以多次调用：

- SEC：对称加解密
- SA：业务感知
- HPRE：非对称加解密

步骤7 CPU完成处理后，将包描述符送给SPPE。

步骤8 SPPE 完成必要的转发查找，生成报文的编辑命令，和包描述符一起交TM处理。

步骤9 TM完成调度、整形后将报文的包描述符推送给PPE。

步骤10 PPE从DDR中读出包数据，完成包头编辑后发送出去。

步骤11 PPE向BMU释放缓存。

----结束

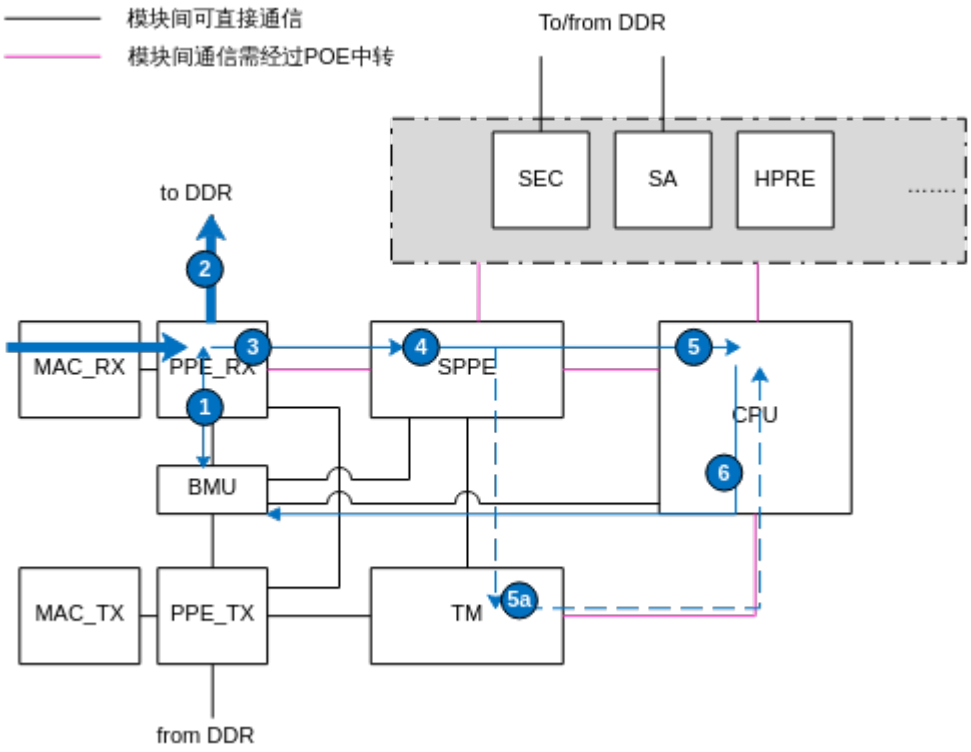
2.7 控制报文处理

这里的控制报文泛指在本地终结的报文，各种基于流/连接转发的业务，如果需要将首包交CPU处理建流/连接，这种首包的处理流程也归到控制报文的处理。

版权所有 © 海思技术有限公司

2.7.1 控制报文 RX 处理流程

图 2-5 控制报文 RX 处理流程



步骤1 PPE预先向BMU申请好内存Buffer缓存。

步骤2 接收到报文后，PPE对报文进行解析，并生成包描述符。

步骤3 PPE将解析结果随报文一起缓存到DDR，同时将包描述符送给POE。

步骤4 SPPE完成报文合法性检查，提取包头做各种转发查找处理。

步骤5 CPU收到报文后处理。

5a SPPE可以选择将报文送给TM，由TM做流控后再交CPU。

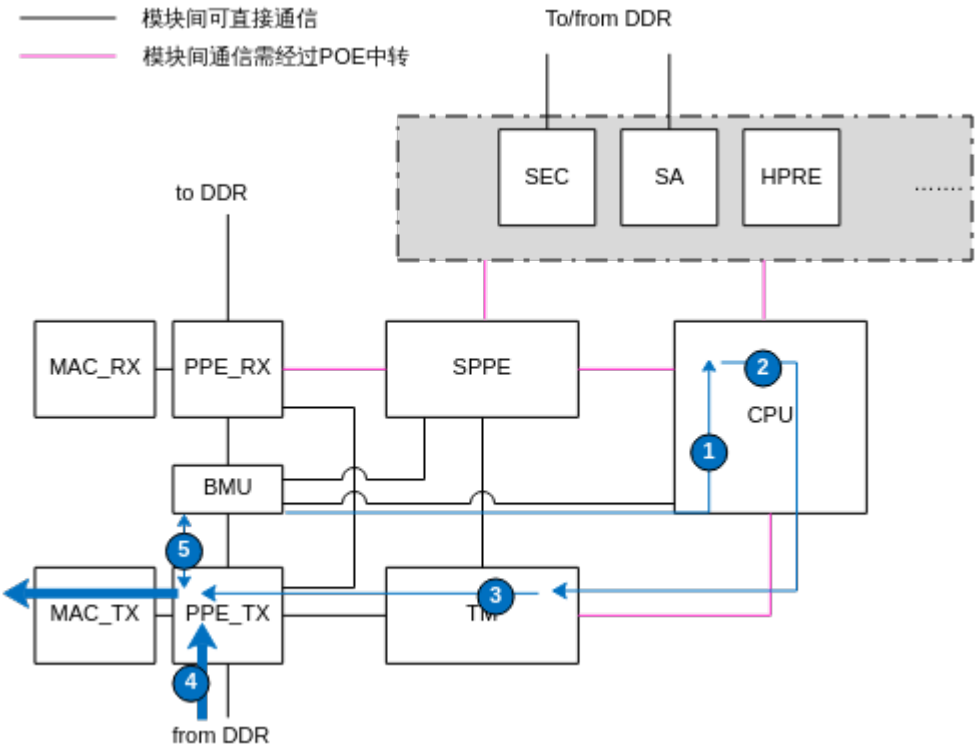
步骤6 CPU向BMU释放缓存。

----结束

版权所有 © 海思技术有限公司

2.7.2 控制报文 TX 处理流程

图 2-6 控制报文 TX 处理流程



步骤1 CPU从BMU 申请缓存。

步骤2 CPU准备好数据后生成包描述符并送给TM（如果有需要也可以由SPPE完成后续流程）。

步骤3 TM完成调度、整形后将报文的包描述符推送给PPE。

步骤4 PPE从DDR中读出包数据，完成包头编辑后发送出去。

步骤5 PPE向BMU释放缓存。

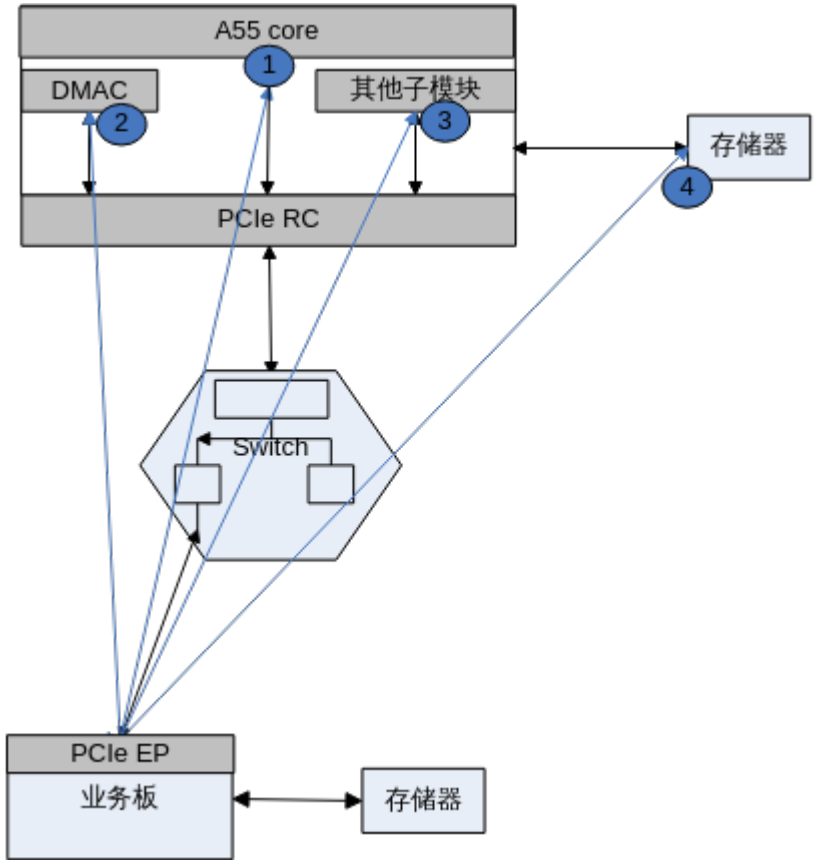
----结束

2.8 PCIe 接口应用场景数据流

如图2-7所示为Hi309a使用PCIe接口的主要数据流。

版权所有 © 海思技术有限公司

图 2-7 使用 PCIe 接口的主要数据流



1 Hi309a CPU读写寄存器：

CPU->PCIe RC-> PCIe EP->业务板存储空间

2 Hi309a 发包到业务板：

系统DMA->PCIe RC-> PCIe EP->业务板存储空间

2 Hi309a 下版本到业务板：

系统DMA-> PCIe RC-> PCIe EP->业务板存储空间

3 其他模块访问业务板：

模块->PCIe RC-> PCIe EP->业务板存储空间

4 业务板发包到Hi309a：

业务板DMA->PCIe EP-> PCIe RC-> Hi309a存储空间

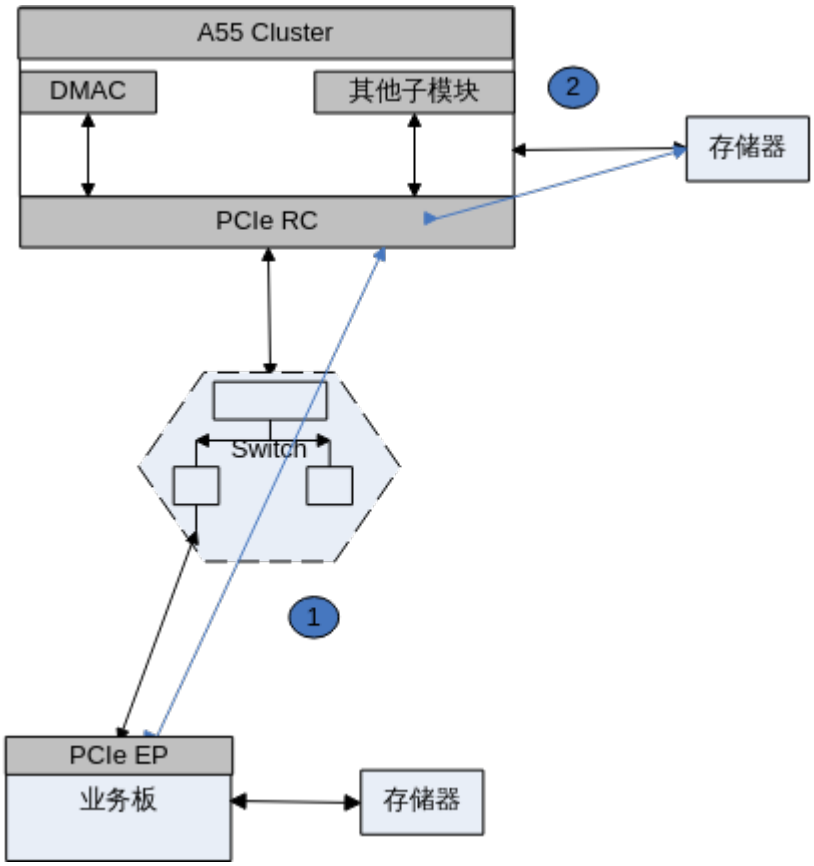
2.8.1 PCIe 设备向 Hi309a 发送报文的数据流

PCIe设备向Hi309a发送报文时，由PCIe设备发起DMA。

版权所有 © 海思技术有限公司

当作为EP的PCIe设备对Hi309a要发起存储器读/写（通过存储器读写发送报文）或IO读/写操作前，需要Hi309a首先配置对应设备PCI配置头的BAR0～BAR5寄存器，将存储器读写、IO读写的基地址写入。当业务板上的PCIe设备根据配置对Hi309a发起MEMWr时，TLP头字段里面包含了PCIe接口作为请求者的ID，目标地址以及请求数据的大小。TLP到达Hi309a的PCI控制器（RC模式），PCIe将该TLP转换成总线写操作访问存储器。

图 2-8 PCIe 设备向 Hi309a 发送报文的数据流



1 业务板的PCIe设备受到报文后根据配置通过MEMWr操作，将报文数据送到Hi309a的PCIe控制器。

2 Hi309a的PCIe控制器作为AXI master根据配置的地址发起对主存的写操作，将数据写到Hi309a的主存。

须知

这一操作并不需要系统DMA的参与。

如果Hi309a的主存访问拥塞，Hi309a的PCIe控制器可以产生流控信息，使业务板停止发送以避免丢包。

版权所有 © 海思技术有限公司

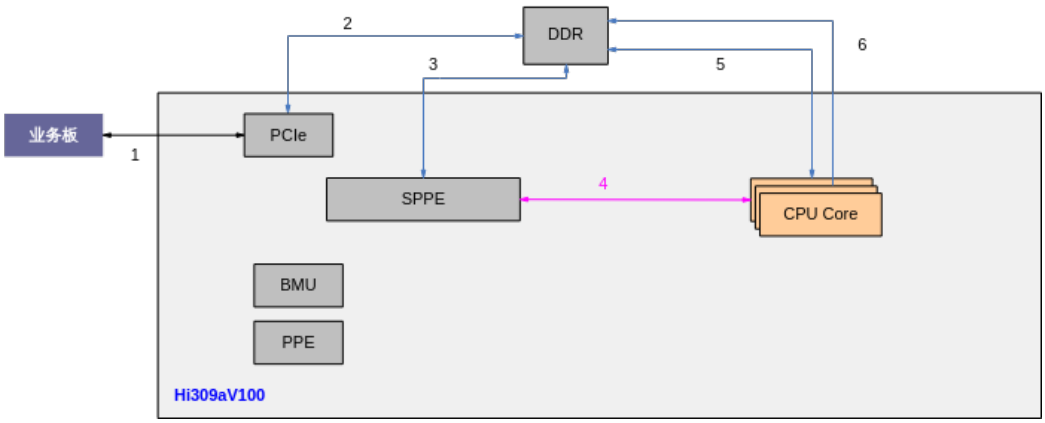
2.8.2 Hi309a 对从 PCIe 接收到的报文的处理流程

通过PCIe设备向Hi309a发送的报文被PCIe模块根据配置的地址写入主存。

因为不经过PPE模块，因此PCIe设备向Hi309a发送的报文无法被PPE解析、生成描述符，并推送任务到POE。

如下图2-9所示为传送网场景（1800）上行方向业务板到CPU处理流程：

图 2-9 传送网场景上行方向业务板到 CPU 控制信令处理流程



详细处理流程如下：

步骤1 业务报文通过PCIe向Hi309a发起MEMWr操作。

步骤2 PCIe控制器接收从设备的报文写到主存（为专用区域）。

步骤3 SPPE定时（逻辑）发起对主存的查询（微码），检查是否有新的报文被写入Hi309a的PCIe数据缓存。

- 每次完成一个报文的处理，SPPE的线程可以立刻发起一次查询操作（两个或条件，定时或者循环线程）。
- 如果对报文处理的性能要求较高，可以对步骤进行拆分成流水处理。

步骤4 SPPE对接收到的报文进行解析并生成描述符（微码实现），经POE推送给CPU核。

步骤5 CPU核解析包描述符，读出报文进行处理。

步骤6 CPU完成报文处理后释放对应的主存（如果是循环buf则不用释放主存）。

须知

PCIe收到的报文并未向BMU申请Buffer，PCIe所使用的缓存由CPU自行管理，不可以对BMU释放。

----结束

版权所有 © 海思技术有限公司

2.8.3 完成 PCIe 设备间的数据交换

如果存在PCIe设备间的数据交换，则SPPE可以在完成解析后，识别转发目的，通过EMIO配置系统DMA，将报文发送到目的PCIe接口连接的业务板。

2.8.4 查询模式的缺点和局限

查询模式下时延较大，且不稳定。定时查询的间隔就是处理的最大抖动范围。进一步，因为抖动较大，在这一场景下1588的处理方案要更加谨慎并充分考虑抖动的因素。

SPPE的查询延时包括：查询定时（μs级），SPPE访问主存的延时（SPPE对主存读访问，在不拥塞的情况下约为80ns~100ns，在拥塞的场景下可能高达2000ns）。

如果不用查询方式，改用循环方式，通过标志来实现SPPE和PCIe之间的正确交付。

2.8.5 下行方向数据报文流处理（SPPE--->PCIe）

反方向上，支持SPPE向PCIe输送数据报，PCIe端口再向外发出数据报文。

这一操作，对于SPPE而言，就是微码通过EMIO对对应PCIe的地址空间直接发起读写操作。

版权所有 © 海思技术有限公司

3 A55 Cluster

3.1 概述

CORTEXA55 是基于ARMv8-A架构的一款mid-range、low-power的处理器。在Hi309a芯片中，共集成2个CORTEXA55 Cluster，每个Cluster包含4个ca55_cpu 和一级、二级Cache系统。

3.2 特性

A55 Core的特性：

- 完整支持ARMv8.2-A A64，A32和T32指令集。
- 支持RAS extension，ARMv8.3-A extension LDAPR指令，ARMv8.4-A extension Dot Product指令。
- 所有异常等级（EL0-3）支持2种执行态（AArch32和AArch64）。
- 支持顺序执行的流水线。
- 支持direct和indirect分支预测。
- 容量32kByte的一级指令Cache和32kByte的一级数据Cache。
- 容量128kByte的二级Cache。
- L1和L2 cache支持ECC或奇偶校验。
- 支持TrustZone技术。
- 支持SIMD和浮点指令。
- 支持Cryptographic指令。
- 内部集成了GIC CPU 接口，支持外接GICv3。
- 支持外接64位系统计数器。
- 指令仅支持小端模式，数据支持大小端模式可配置。
- Hi309a支持2.2GHz。
- 基于单线程结构。

版权所有 © 海思技术有限公司

3.3 功能描述

3.3.1 数据流

处理器子系统的数据流，主要有4条：

1. FCM通过CHI接口访问DDR控制器，或外设。
2. FCM通过AXI Peripheral port（简称pp口）配置GIC-600，控制SGI，PPI和SPI的priority，routing等属性。Peripheral port能访问的地址范围，由一对寄存器配置，再接到FCM的一对pin 上：ASTARTMP bit[39:20]，AENDMP bit[39:20]。处理器如果发现要访问的memory-mapped地址落ASTARTMP和AENDMP之间，就会把访问命令从Peripheral Port上发出。配置GIC的addr空间的寄存器字段：

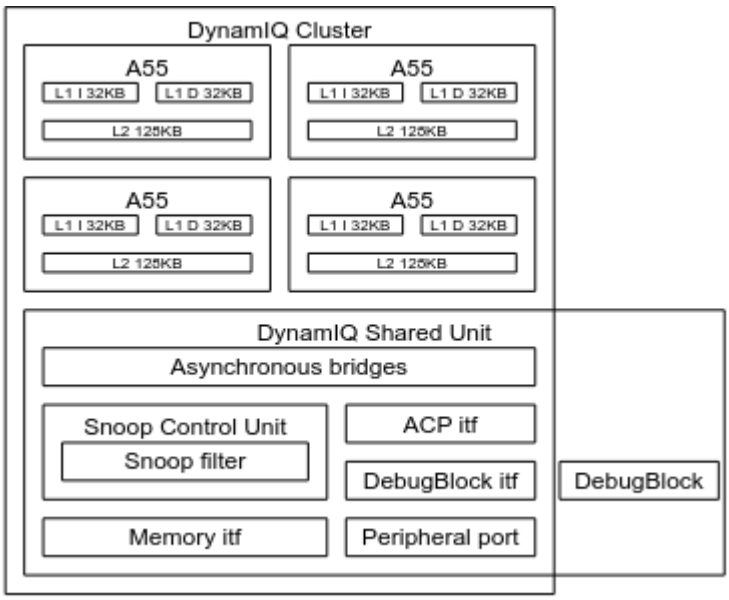
   a55_pport_astart bit[19:0]

   a55_pport_aend bit[19:0]
3. FCM和GIC之间，通过stream接口上报和响应interrupt。
4. 外部调试器通过Jtag口访问FCM的debug寄存器。

3.3.2 FCM 结构

DynamIQ Cluster和DebugBlock一起，构成了Hi309a的主处理器单元，它的结构固定成如图3-1所示。

图 3-1 FCM 结构



没有L3 cache，4个a55之间的一致性，由Snoop Controller Unit维护。

3.3.3 时钟

CPU子系统只有1个主时钟：TOP CRG出来的clk_arm_ini。其他时钟都是它的分频钟。此外，还有1个配置通路的时钟，即外面interconnect的工作时钟。

版权所有 © 海思技术有限公司

CPU子系统的软复位，参考《 Hi309a 寄存器手册 》文档。

3.3.4 复位

CPU子系统输入2个复位：rst_chip_n和rst_bmc_n，前者复位寄存器配置模块和分频器，后者复位其它逻辑。

CPU子系统的软复位，参考《 Hi309a 寄存器手册 》文档。

3.4 启动

A55的启动步骤：

步骤1 系统上电，CRG启动eFuse读并做Repair。

步骤2 安全核启动，配置PLL及A55有关的寄存器。

步骤3 安全核解除1个A55核或者所有A55核的热复位。

步骤4 A55开始启动。

步骤5 A55启动完成后，开始工作。

步骤6 电源控制器根据设备请求，控制各个core进出各级低功耗状态。

----结束

3.5 注意事项

Hi309a总线不支持混合属性一致性，流程约束，不允许多核并发下电，只能串行。

Hi309a多核抢锁说明：

A55执行Exclusive抢锁的三个步骤：

步骤1 发起ReadShared读取锁变量（读）。

步骤2 发起CleanUnique获取写权限改写锁变量（改）。

步骤3 发起写请求释放锁变量（写）。

----结束

基于当前A55和DUI总线的实现，必然会出现，有的CORE 抢锁访问的延迟比其他的CORE 速度快，该CORE总是发送CleanUnique将其他的CORE 的CACHELINE INVALID。

在这个流程里，Cluster1发起CleanUnique获得写权限之前，Cluster0发起的CleanUnique先到LLC，从而抢锁失败，并造成有状态无数据（empty E）的情况。但是CPU连续两次抢锁（Store）之间执行操作的间隔t大于一般Load+Store的间隔（500ns-860ns），可以缓解这种不公平现象。

对软件要求如下：软件如果追求绝对的公平，需要使用公平性更高的排队锁（ticket-lock）。

版权所有 © 海思技术有限公司

4 时钟

4.1 概述

时钟管理模块对芯片时钟输入、时钟生成和控制进行统一的管理。

Hi309a接收外部的晶振时钟输入，利用内部的锁相环（PLL：Phase_ Locked Loop）及时钟电路产生芯片内部需要的各种工作时钟。

Hi309a内部一共有10个PLL，用于产生芯片内各个模块所需要的各种工作时钟。

4.2 关键术语

表 4-1 关键术语

PLL	Phase-Locked Loop，锁相环。 将芯片输入的高稳定度的晶振时钟倍频到内部逻辑使用的高频时钟
Slow模式	系统工作在慢频即晶振时钟域。
Normal模式	系统工作在高频即PLL IP输出时钟域。
PD功能	PowerDown下电功能。

4.3 功能特性

时钟管理模块特性：

- 支持多路管脚时钟输入，作为芯片内部的时钟源头。
- 支持一路25MHz/50MHz/66.667MHz系统时钟输入。
- 支持一路156.25MHz差分时钟输入作为GE/XGE Serdes的参考时钟。
- 支持一路100MHz差分时钟输入作为PCIE/SATA Serdes的参考时钟（当芯片最多只使用到PCIE GEN2时可以使用156.25MHz差分时钟输入作为参考时钟）。
- 支持多类IO接口时钟输入。

版权所有 © 海思技术有限公司

- GE口支持RGMII模式下由内部PLL产生250MHz的时钟送至RGMII，由RGMII产生125MHz的tx时钟输出PAD至对端。
- 支持同步以太需求，通过软件配置选择输出链路，并将选择出来的时钟按如下分频后送到片外，对时钟质量无特殊要求。其中：

  XGE：并口时钟257.8125MHz，分频输出8kHz。

  GE： 并口时钟125MHz，分频输出8kHz。

  FE： 并口时钟125MHz，分频输出8kHz 。
- 支持配置和管理PLL生成各频点的时钟源头。
- 支持对各PLL进行独立频点配置，BYPASS，以及PD功能。
- 支持对PLL生成的时钟源头进行分频生成各业务模块主时钟。
- 支持可选模块和接口的时钟可软件控制时钟打开或关断。
- 支持PLL时钟作为Hilink IP的测试时钟，相关时钟设计均有ICG可关断。
- 支持时钟可测试性设计及DFX设计。
- 支持功能模式下的时钟满足DFT模式下可控。
- 支持提供寄存器读取逻辑时钟状态有无，对关键时钟信号（PLL以及IP输出时钟）可通过管脚进行检测。
- 支持提供锁相环锁定状态检测，并提供CPU接口可查询的历史状态记录寄存器。
- 支持检测内部锁相环是否失锁，可通过失锁指示管脚或辅助接口在CPU时钟域异常情况下获取关键PLL状态。
- 支持提供Serdes PLL失锁中断上报功能，并且可以查看Serdes core的PLL当前和历史锁定状态。
- 支持Serdes每条链路恢复时钟可通过管脚进行观测。
- 支持提供Serdes 参考源时钟丢失检测中断上报功能。
- 支持功能mbist时钟可bypass到芯片内部PLL分频时钟。
- 支持所有Hilink输出的mclk/rxoclk时钟均可门控（默认关闭），初始化完成后再打开时钟。

4.4 功能描述

Hi309a根据需要集成了9个SCPLL及1个FNPLL，其中DDR_WRAP内部分别集成1个SCPLL+1个FNPLL。各PLL作用的时钟域如表4-2所示。

表 4-2 时钟列表

时钟	频率	模块
PLL0	1200MHz@0.8V/（典型） 1400MHz@1.0V/ 900MHz@0.7V（BSBC默认配置）	SPPE SUB/POE SUB/ETH SUB RING/CPU/LLC/HHA/BMU/POE

版权所有 © 海思技术有限公司

时钟	频率	模块
PLL1	2400MHz	CPU/PERI SUB/HPRE/TM/TRING/ESPI/CER/SECL TRNG/SEC/IFC/BOOTROOM/EMMC
PLL2	800MHz	PPU MAM TOP/PPP SSU PA TOP PPE MAG TOP/ETH SUB/ROCEE/MAG TOP/MAG XSEC/ MDIO/I2C/PMBUS/HACSPI
PLL3	1000MHz	RGMII/NANDC/MAG TOP/SATA SERDES SUB
PLL4	850MHz	SPPE/SPPE SUB
PLL5	1600MHz@0.8V/（典型） 2200MHz@1.0V/ 1200MHz@0.7V/（BSBC默认配置）	CPU
PLL6	24MHz	USB
PLL7	1000MHz	ETH SUB
DDR	3200/2400/2666MHz	DDR

4.5 应用配置

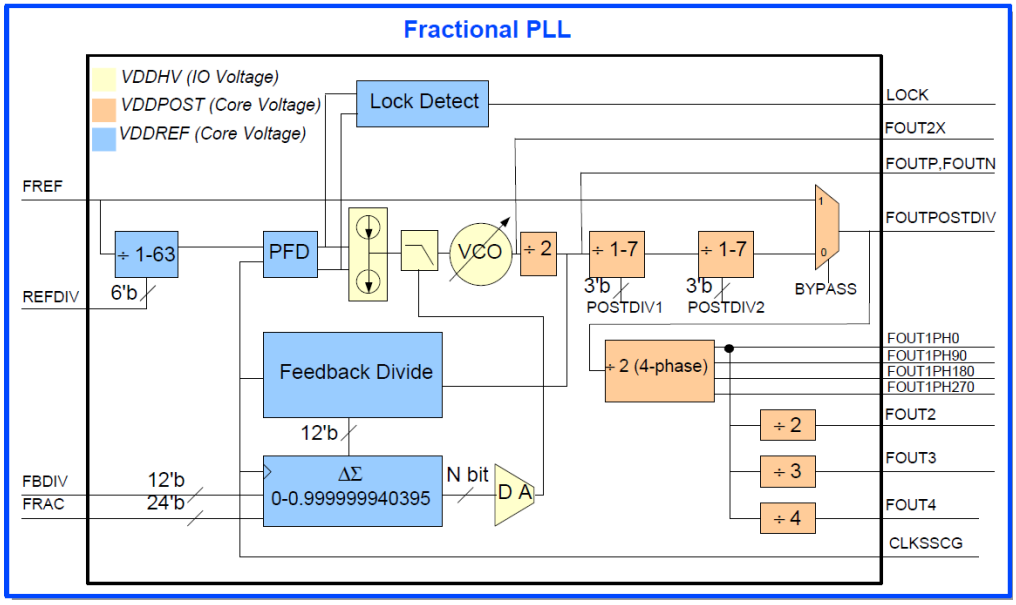
时钟应用配置在以下小节介绍。

4.5.1 PLL 配置

版权所有 © 海思技术有限公司

4.5.1.1 PLL 控制信号

图 4-1 PLL 功能结构图（12nm IP）



须知

- 5MHz ≤ FREF ≤ 1200MHz(Int)，10MHz ≤ FREF ≤ 1200MHz(Frac)。
- PLL内部VCO输出时钟FOUT2X的有效频率范围为1600MHz~6400MHz。
- PLL输出时钟FOUTPOSTDIV的有效频率范围为16MHz～3200MHz。

PLL控制信号如下表4-3所示。

表 4-3 PLL 控制信号列表

PLL信号名	方向	位宽	说明	备注
FREF	I	1	Reference Clock	Int mode 参考时钟（1MHz～1.2GHz），Frac mode 参考时钟10MHz~1.2GHz。
REFDIV	I	6	Reference Divide Value	参考分频值（1～63）。
FBDIV	I	12	PLL Feedback Divide Value	Int mode反馈分频值（16~3200），Frac mode反馈分频值（20~320）。
FRAC	I	24	Fractional portion Feedback Divide Value	小数反馈分频值。

版权所有 © 海思技术有限公司

PLL信号名	方向	位宽	说明	备注
POSTDIV1	I	3	First Post Divide Value	VCO之后第一级分频值，1~7，注：POSTDIV1始终应“大于等于”POSTDIV2。
POSTDIV2	I	3	Second Post Divide Value	VCO之后第二级分频值，1～7。 （输出值是FVCO/POSTDIV1/POSTDIV2）
PD	I	1	Global Power Down	全局power down信号，高有效。
BYPASS	I	1	FREF Bypass	参考时钟bypass，有效时，FOUTPOSTDIV=FREF。
DSMPD	I	1	Delta Sigma Power Down	为高时，为整数模式。
DACPD	I	1	Cancellation DAC Power Down	正常模式为0。
FOUT4PHASEPD	I	1	4 Phase Clock Power Down	四相位输出时钟的power down。同时对FOUT2，FOUT3，FOUT4有效。
FOUTPOSTDIVPD	I	1	Post Divide Power Down	VCO之后的分频器power down，同时对四相位输出时钟，FOUT2，FOUT3，FOUT4有效。
FOUT2XPD	I	1	2X VCO rate output Clock Power Down	VCO输出时钟power down。
FOUT2X	O	1	2X VCO rate Output Clock	PLL输出的时钟。
FOUTP	O	1	Positive phase VCO rate output Clock	PLL输出的时钟。
FOUTN	O	1	Negative phase VCO rate output Clock	PLL输出的时钟。
FOUTPOSTDIV	O	1	Divided VCO Output Clock	PLL输出的时钟。
FOUT1PH0	O	1	Phase 0 of 4-phase output	PLL输出的时钟。

版权所有 © 海思技术有限公司

PLL信号名	方向	位宽	说明	备注
FOUT1PH90	O	1	Phase 90 of 4-phase output	PLL输出的时钟。
FOUT1PH180	O	1	Phase 180 of 4-phase output	PLL输出的时钟。
FOUT2PH270	O	1	Phase 270 of 4-phase output	PLL输出的时钟。
FOUT2	O	1	Divide-by-2 Output Clock	PLL输出的时钟。
FOUT3	O	1	Divide-by-3 Output clock	PLL输出的时钟。
FOUT4	O	1	Divide-by-4 Output clock	PLL输出的时钟。
LOCK	O	1	PLL Lock Signal	PLL锁定指示信号。
CLKSSCG	O	1	Clock for external spread spectrum modulator	小数分频模式展频输出的时钟。

4.5.1.2 PLL 频率配置公式

PLL的输出频率配置公式如下：

若：DSMPD=0b1

FOUT2X = 2*FREF/REFDIV*FBDIV

FOUTP/N = FREF/REFDIV*FBDIV

FOUTPOSTDIV= FREF/REFDIV*FBDIV/POSTDIV1/POSTDIV2

若：DSMPD=0b0

FOUT2X = 2* (FREF/REFDIV)*(FBDIV + FRAC/2^24)

FOUTP/N =(FREF/REFDIV)*(FBDIV + FRAC/2^24)

FOUTPOSTDIV = (FREF/REFDIV)*(FBDIV + FRAC/2^24)/POSTDIV1/POSTDIV2

4.5.1.3 PLL 配置流程

PLL产生不同频率的时钟，需要对PLL的输入进行不同的配置。以50MHz参考时钟产生1200MHz的时钟配置为例，推荐配置步骤如下：

版权所有 © 海思技术有限公司

步骤1 PLL上电并输入PLL的参考时钟，待上电稳定后即表示完成。

步骤2 在Slow模式，通过软件配置系统控制器寄存器SC_PLLFCTRL0，以产生1200MHz的时钟频率。依据PLL频率配置公式章节描述，确定配置参数如下：

1. 配置寄存器SC_PLLFCTRL0 bit[5:0]为0x2，使FREFDIV=2。
2. 配置寄存器SC_PLLFCTRL0 bit[17:6]为0x30，使FBDIV=48。
3. 配置寄存器SC_PLLFCTRL0 bit[18:23] 为0x9，使POSTDIV1为1，POSTDIV2为1。
4. 配置寄存器SC_PLLFCTRL0 bit[25]为0b1，即配置PLL power down。

步骤3 等待超过1us后，将Power down信号PD置为低电平，即：

1. 配置寄存器SC_PLLFCTRL0 bit[25]为0。
2. 等待PLL Lock。

步骤4 查询系统控制器寄存器SC_PLL_LOCK_STATUS bit[0]，若置1，则表明PLL0已经Lock。

----结束

其他PLL的配置可通过相应系统控制器寄存器SC_PERCTRL*实现，步骤同上。

通过读取系统控制器寄存器SC_PLL_LOCK_STATUS的值查询各PLL的Lock信息。

须知

DDR_WRAP内部分别集成了1个SCPLL+1个FNPLL，两者共用同一组配置寄存器SC_DDR_PLLCTRL0/SC_DDR_PLLCTRL1。但为避免使用SCPLL（FNPLL）时，另外一个PLL空转浪费功耗，故使用SC_DDR_PLLCTRL0 bit[31]bit，对应FNPLL的全局PD信号（依然使用SC_DDR_PLLCTRL0 bit[25]bit，对应SCPLL的全局PD信号），从而在使用DDR内部FNPLL配置时，区别于上述流程的其中两条：

- 配置寄存器SC_PLLFCTRL0 bit[31]为0b1，即配置PLL power down。
- 等待超过1μs后，配置寄存器SC_PLLFCTRL0 bit[31]为0，即将Power down信号PD配置为低电平。

  等待PLL LOCK。

4.5.1.4 PLL 典型配置值

Hi309a根据需要集成了9个SCPLL及1个FNPLL；其中DDR_WRAP内部分别集成1个SCPLL+1个FNPLL。

以FREF=50M为例，各PLL的典型配置参考如表4-4所示。

表 4-4 PLL 配置参数说明

需求频点		xxMHz	
成员名称	**配置值**	**成员名称**	**配置值**
SC_PLLFCTRL0	-	pll0_pd	0b1/0b0
pll0_foutvco_pd	0b1	pll0_bypass	0b0
pll0_foutpostdiv_pd	0b0	pll0_postdiv2	A2

版权所有 © 海思技术有限公司

需求频点		xxMHz	
成员名称	配置值	成员名称	配置值
pll0_fout4phase_pd	0b0	pll0_postdiv1	A1
pll0_dac_pd	0b0	pll0_fbdiv	B
pll0_dsm_pd	0b1/0b0	pll0_refdiv	C

说明

1. 斜体字体部分均使用默认值即可。
2. pll_dsm_pd通常情况下使用整数模式默认值0b1，在动态调频等使用小数模式默认值0b0。
3. pll_pd上电使用默认值0b1，上电稳定即1us后，软件改配为0b0。
4. 需求频点 = (Freq/C) * B /A2/A1，以需求频点1.2G，晶振时钟50M为例：1200M = (50/C) *B /A1/A2，选取5 适当的值C=2，B=48，A2=1，A1=1或C=2，B=96，A2=1，A1=2。

注：

- pll_postdiv2 ≤ pll_postdiv1，即A2 ≤ A1。
- 800MHz ≤ (Freq/C) *B ≤ 3200MHz 。

4.5.2 网络同步以太

网络同步以太配置流程，支持从所有ETH口任意选择一路端口恢复时钟，然后分频输出8kHz时钟。注：时钟占空比可以为非1:1。

上电完成后，通过配置ETH SUBCTRL的SC_NET_SYNC_LANE_SEL0~2寄存器bit[3:0]，在多路恢复时钟中选择其中一路。然后根据所选恢复时钟的时钟频率，相应配置ETH SUBCTRL的SC_NET_SYNC_CFGB0~2及SC_NET_SYNC_CFGC0~2寄存器，即软件配置该恢复时钟对应的分频系数（具体配置值见《Hi309a寄存器手册》），由CRG分频电路分频输出8kHz时钟。

同时，为了保证时钟正常输出，在同步以太电路中增加了“源恢复时钟有无检测”电路。这样在检测到所选恢复时钟已丢失时，可关闭8kHz时钟输出。

须知

可通过配置ETH SUBCTRL的SC_NET_SYNC_TEST_EN bit[5:3]分别屏蔽3路“源恢复时钟有无检测”电路对时钟输出的影响。

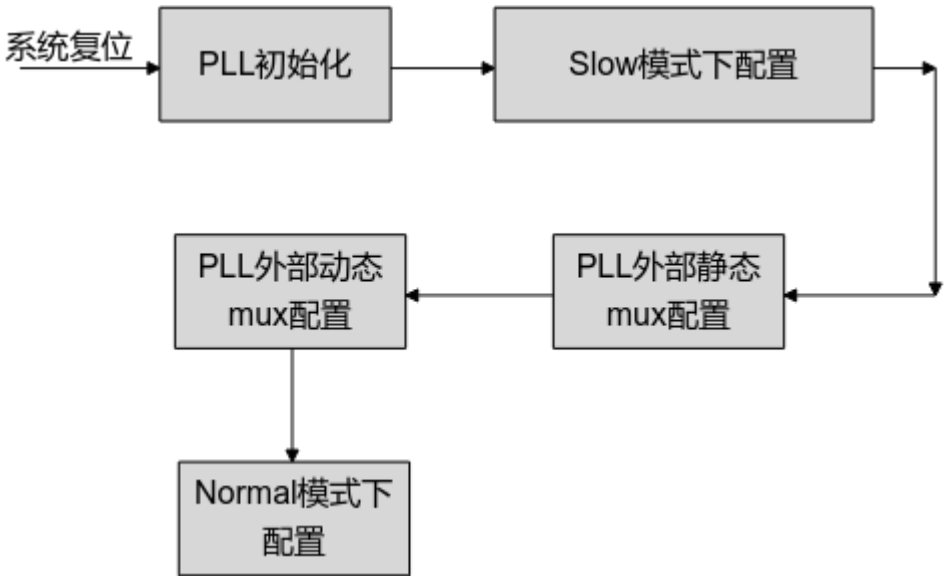
表 4-5 网络同步以太链路模式对应分频系数配置表

链路模式	SC_NET_SYNC_CFGB	SC_NET_SYNC_CFGC	备注
GE	9h2	22h3d09	OR要求
10GE	9h20	22h7de29	

版权所有 © 海思技术有限公司

4.6 初始化

图 4-2 软件正常配置流程图



芯片进入整芯片复位（上电复位/非掉电复位），系统自动进入Slow模式，即整个系统时钟来自晶振；

在上电复位场景下：由于系统控制器寄存器SC_PLL_CLK_BYPASS0被复位为初始默认值0，即各 PLL外部静态MUX有效，输出晶振时钟；在非掉电复位场景下：由于系统控制器寄存器SC_PLL_CLK_BYPASS0及PLL IP不被复位，即各 PLL外部静态MUX无效，输出PLL IP时钟；

而系统控制器寄存器SC_PLL_CLK_BYPASS1无论在上电复位或非掉电复位场景下，都会被复位为初始默认值0，即各PLL动态MUX有效，输出晶振时钟；

1. 参考“PLL配置流程”对各PLL进行初始化配置，待各PLL时钟锁定后，配置系统控制器寄存器SC_PLL_CLK_BYPASS0相应bit位为0b1（实际SC_PLL_CLK_BYPASS0本身并没有被复位，这一步是为了将上电初始化和非上电初始化流程做一致），此时各PLL第一级外部bypass无效，第一级MUX输出PLL IP时钟。
2. 配置系统控制器寄存器SC_PLL_CLK_BYPASS1相应bit位为0b1，此时各PLL第二级外部bypass无效，第二级MUX输出PLL IP时钟； 至此系统进入NORMAL模式工作。
3. 配置SYSCTRL寄存器SC_SYSMODE_CTRL[2:0]为0b1XX（X为任意值），此时等待系统进入NORMAL模式，整芯片开始正常工作。
4. 查询SYSCTRL的寄存器SC_SYSMODE_STATUS，值为0x4,表明系统已经切换到normal 状态。

4.7 注意事项

无。

版权所有 © 海思技术有限公司

4.8 DFX

时钟检测主要实现关键时钟的有无及频率的检测。主要覆盖的时钟包括IP的端口时钟、通过IO输入和输出的时钟、PLL输出的时钟以及各业务模块的关键时钟，实现方式如下：

将关键时钟通过分频到低速时钟（小于50M）输出到端口进行检测；

通过软件读取系统控制器PLL LOCK寄存器，确定PLL时钟状态；

通过寄存器可上报PLL UNLOCK状态；

对关键时钟（包括：serdes参考时钟、PLL IP输出时钟等）进行时钟有无检测，并提供寄存器进行状态读取。

4.9 性能指标

无。

版权所有 © 海思技术有限公司

5 复位

5.1 概述

Hi309a复位管理模块对整个芯片的复位、各功能模块的复位进行统一管理，生成芯片内部各功能模块的复位信号。

5.2 关键术语

表 5-1 关键术语

术语	描述
外部狗	外部看门狗芯片。

5.3 功能特性

复位管理模块的主要特性如下：

- 支持上电复位和非上电复位两种方式。
- 支持上电复位可将芯片内所有模块进行复位。
- 支持非上电复位在保障DDR数据保持功能的前提下，复位剩余的所有模块，涉及到不复位的模块需要经过分析。
- 支持按键复位、CAN复位、外部狗溢出复位、全局软复位等作为芯片非上电整芯片复位源。
- 支持上述非上电整芯片复位源均可通过软件控制使能。
- 支持上述非上电整芯片复位源通过“相与”逻辑产生外部706芯片MR信号。
- 支持外部狗溢出复位可产生不可屏蔽中断，同时支持延时产生MR信号，延时时间可配置（1s~10s可配置，步进1ms，默认500ms）。
- 支持非上电复位场景下DDRPHY和DMC均被复位。
- 支持CPU自主独立复位并保持NP转发正常；支持CPU异常复位时内置LANSwitch转发正常。

版权所有 © 海思技术有限公司

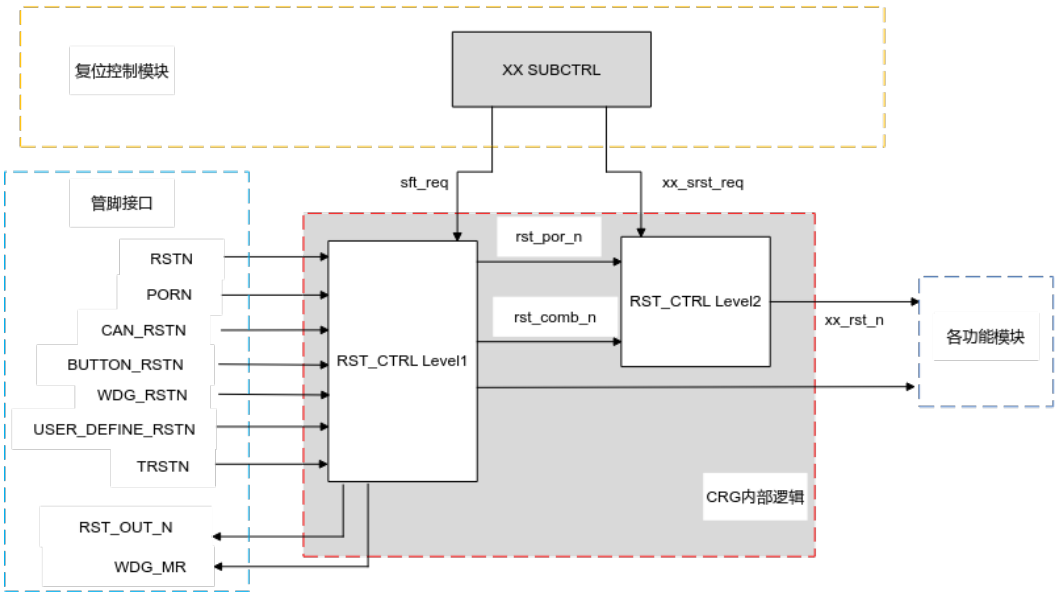
- 支持CPU自主独立复位并保持NP转发正常，完全属于软件流程，通过软件配置各core复位实现，不需要CRG硬件逻辑参与。
- 支持CPU异常复位时内置LANSwitch转发正常；需要配置相关寄存器实现CPU异常时，对应逻辑不复位，从而实现内置LANSwitch转发正常。
- 支持各业务模块软复位及各IP的独立复位。
- 支持芯片内部所有PLL可独立复位（Power Down），包括系统内部以及HLINK内部PLL。
- 支持芯片各接口、通道、异步FIFO，Interlaken接口、PCIe接口，重要状态机等独立复位。
- 支持芯片复位原因记录。
- 支持硬件完成五次复位原因记录，不需要软件控制。
- 支持复位原因记录包括：上电复位，按键复位，CAN控制复位，全局软复位，外部狗溢出复位。
- 支持复位原因记录寄存器不受复位端控制。
- 支持复位的可测试性设计及DFX设计。
- 支持功能模式下的复位满足DFT模式下可控。
- 支持关键复位信号（IP，数字逻辑，以及CRG部分的复位）可通过管脚进行检测。
- 支持芯片内部集成看门狗溢出复位。

5.3.1 复位接口信号

5.3.1.1 网络场景下复位接口信号

Hi309a的复位结构如图5-1所示。

图 5-1 Hi309a 的复位结构图



版权所有 © 海思技术有限公司

表 5-2 复位信号描述

名称	方向	描述
RSTN	I	系统复位输入，低电平有效，用于设计复位整芯片逻辑。
PORN	I	系统上电复位管脚，低电平有效；当发生系统复位时，PORN为低表示是上电复位，上电完成后PORN始终为高。
CAN_RSTN	I	CAN复位输入，低电平有效。
BUTTON_RSTN	I	按键复位输入，低电平有效。
WDG_RSTN	I	外部狗溢出复位输入，低电平有效。
USER_DEFINE_RSTN	I	用户自定义复位输入，低电平有效。
TRSTN	I	JTAG复位输入，低电平有效。
RST_OUT_N	O	CPU复位输出，可用于复位CPU周边器件。在内部处理器撤离复位前拉高，默认CPU复位撤离前200us拉高。
WDG_WDI	O	专用的喂狗信号。
WDG_MR	O	芯片内部复位外部看门狗信号，输出低电平持续时间必须大于1ms，连接外部看门狗MR信号。
WDOG_MODE	I	0：芯片使用内部看门狗复位模式； 1：芯片使用外部看门狗复位模式。

首先，明确Hi309a芯片复位输入管脚包括：PORN、RSTN、CAN_RSTN、BUTTON_RSTN、WDG_RSTN、USER_DEFINE_RSTN；复位相关输出管脚包括：硬件喂狗WDI、芯片复位外部看门狗WDG_MR、RST_OUT_N等。

表 5-3 各种类型的复位范围描述

复位类型	复位方法	解复位方法	复位范围描述
上电复位	PORN管脚低电平有效	PORN管脚低电平撤离	复位整芯片。
非上电复位	上述外部复位源任一个低电平有效	对应外部复位源低电平撤离	除CRG部分逻辑、系统控制器部分逻辑等其他部分均被复位。

5.4 复位类型

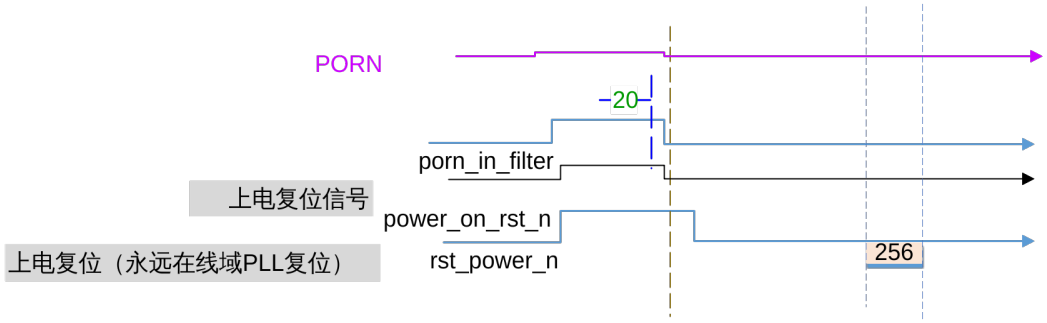
版权所有 © 海思技术有限公司

5.4.1 芯片上电复位

Hi309a上电复位必须满足如下条件：

- 芯片的电源上电稳定后（除EFUSE_VQPS以外）。
- 上电复位管脚PORN输入低电平。
- 晶振时钟输入稳定正常。
- 复位管脚PORN信号为低并大于20个晶振周期。

图 5-2 上电复位时序图



须知

porn_in_filter、power_on_rst_n、rst_power_n为芯片内部逻辑产生的信号。

5.4.2 网络外狗场景芯片非上电复位

5.4.2.1 外部复位源输入

Hi309a网络外狗场景外部复位源输入导致的非上电复位必须满足如下条件：

- 网络场景，管脚BOOT_SEL5在芯片封装时上拉。
- 内狗外狗模式管脚WDOG_MODE上拉。
- 上电复位管脚PORN保持高电平。
- 任意复位源管脚CAN_RSTN/BUTTON_ RSTN/WDG_RSTN/ USER_DEFINE_RSTN输入低电平，且低电平维持时间大于20个晶振周期。
- 晶振时钟输入稳定正常。
- 经“与”逻辑以及内部延时电路后送出WDG_MR低电平信号，并引入RSTN低电平输入，从而触发芯片复位。

须知

RSTN低电平持续时间要求至少50000个晶振cycle（对应25M晶振时钟，约2ms），因为CRG内部逻辑实现：芯片外围FLASH晚于CPU复位的时间差为50000个CYCLE，因此如果低电平持续宽度不够，会导致CPU复位了，但外围FLASH没有复位成功。

版权所有 © 海思技术有限公司

5.4.2.2 全局软复位

Hi309a全局软复位导致的非上电复位步骤如下：

步骤1 通过配置系统SYSCTRL寄存器，为特定值“32h5555_5555”。

步骤2 由SYSCTRL送出glb_sft_req高电平有效信号，送至系统CRG。

步骤3 系统CRG采样glb_sft_req信号，但不直接触发芯片复位，而是与其他外部复位源（外部复位源要经过有效输入判断电路及延时电路处理）经“与”逻辑后送出WDG_MR低电平信号，并引入RSTN低电平输入，从而触发芯片复位。

----结束

5.4.2.3 内部狗溢出复位

内部狗关键特性：

- 特性一：内部狗具有定时的溢出能力，当1.2s @ 66.667MHz,1.6s @ 50MHz，1.6s @ 25MHz（同时时间可调）内没有喂狗信号时，则内部狗溢出，产生一次内部狗复位。
- 特征二：内部狗计时的清0方式，包括：“软件清0”、“掉电复位”、“逻辑喂狗”、“内部狗自身溢出”四种。
- 特性三：“内部狗定时溢出”，最终内部狗的复位功能必须支持软件可配；即只有使能该功能时，内部狗溢出才会引起芯片复位。
- 特性四：内部狗必须依赖系统参考时钟。
- 特性五：WDOG_MODE为外狗模式时（WDOG_MODE上拉），内部狗默认不工作，需要软件配置使能启动。
- 特性六：任意有效复位源撤离时，会触发内部狗重新计数。

Hi309a内部狗溢出导致的非上电复位步骤如下：

步骤1 芯片内部看门狗定时溢出触发内部狗送出复位请求信号inter_wdg_srst_req_n低电平有效信号。

步骤2 CRG采样到inter_wdg_srst_req_n低电平有效后，并不直接触发芯片复位，而是与芯片与其他外部复位源（外部复位源要经过有效输入判断电路及延时电路处理）及系统软复位请求信号，经“与”逻辑以及内部延时电路后送出WDG_MR低电平信号，并引入RSTN低电平输入，从而触发芯片复位。

----结束

5.4.2.4 芯片非上电复位流程

芯片上电完成后，PORN管脚保持高电平，此时以上任意类型的非上电复位形式有效后（外部复位源输入，需要经过有效外部复位源检测以及延时输出等电路处理），送出WDO低电平有效信号，从而触发单板硬件电路向Hi309a送入RSTN低电平有效，触发芯片发生非上电复位，具体流程如下：

- Hi309a输入RSTN有效低电平。
- 系统CRG电路采样到RSTN低电平有效信号后，向DDRC发出自刷新请求。

  ”ddr_warm_rst_req_n“低电平有效信号；系统CRG启动保护计时，默认50000个晶振cycle，等待DDRC返回握手ack信号。

版权所有 © 海思技术有限公司

- 待DDRC正常返回握手信号”ddr_wram_rst_ack_n “低电平有效信号，或者保护计时满足50000个cycle计时后，系统CRG触发非上电复位有效。
- 复位有效后，系统CRG电路延迟64个周期将”ddr_warm_rst_req_n “置高电平无效，完成本次请求握手；同时整个系统时钟被bypass到晶振时钟，系统进入Slow模式。
- RSTN低电平有效持续200ms左右（由单板硬件电路典型如外置硬件看门狗芯片决定），撤离RSTN低电平；系统CRG电路采样到RSTN低电平撤离后，即将非上电复位信号置无效，从而完成本次复位流程。

5.4.2.5 CPU 系统软复位

CPU系统软复位支持硬件流程和软件流程两种方式的独立复位；其中硬件流程等同于”芯片非上电复位流程“介绍，即只要单板硬件电路将 RSTN管脚独立置低电平有效即可；在此不再赘述。

对于CPU系统软件流程触发的复位，具体步骤如下：

步骤1 通过配置系统SUBCTRL寄存器（具体见可寄存器表单描述）为特定值“0x5555_5555”；由系统控制器送出sc_srst_ap_req高电平有效信号，送至系统CRG。

步骤2 系统CRG采样到sc_srst_ap_req有效后，启动50000个（可配置）晶振cycle计数；计满后自动撤离复位。

步骤3 系统CRG采样到sc_srst_ap_req有效信号后，向DDRC发出自刷新请求“ddr_warm_rst_req_n”低电平有效信号；系统CRG启动保护计时，默认50000个晶振cycle，等待DDRC返回握手ack信号。

步骤4 待DDRC正常返回握手信号“ddr_wram_rst_ack_n”低电平有效信号，或者保护计时满足50000个cycle计时后，系统CRG触发CPU系统复位。

步骤5 待复位请求撤离后，至此完成CPU系统软件流程触发的复位操作。

----结束

须知

网络场景CPU系统软复位主要用于调试场景。如果用于产品场景需要软件保证先做临终遗言记录，再触发复位。

5.4.3 网络内狗场景芯片非上电复位

5.4.3.1 外部复位源输入

Hi309a网络内狗场景外部复位源输入导致的非上电复位必须满足如下条件：

- 网络场景，管脚BOOT_SEL5在芯片封装时上拉。
- 内狗外狗模式管脚WDOG_MODE下拉。
- 上电复位管脚PORN保持高电平。
- 任意复位源管脚CAN_RSTN/BUTTON_ RSTN/WDG_RSTN/ USER_DEFINE_RSTN输入低电平，且低电平维持时间大于20个晶振周期。

版权所有 © 海思技术有限公司

- 晶振时钟输入稳定正常。

须知

RSTN低电平持续时间要求至少50000个晶振cycle（对应25M晶振时钟，约2ms），因为CRG内部逻辑实现：芯片外围FLASH晚于CPU 复位的时间差为50000个CYCLE，因此如果低电平持续宽度不够，会导致CPU复位了，但外围FLASH没有复位成功。

5.4.3.2 全局软复位

Hi309a系统软复位导致的非上电复位步骤如下：

步骤1 通过配置系统SYSCTRL寄存器，为特定值“0x5555_5555”；由SYSCTRL送出glb_sft_req高电平有效信号，系统CRG电路采样到glb_sft_req高电平有效信号后，向DDRC发出自刷新请求。

步骤2 ”ddr_warm_rst_req_n“低电平有效信号；系统CRG启动保护计时，默认50000个晶振cycle，等待DDRC返回握手ack信号。

步骤3 待DDRC正常返回握手信号”ddr_wram_rst_ack_n “低电平有效信号，或者保护计时满足50000个cycle计时后，系统CRG触发非上电复位有效。

步骤4 复位有效后，系统CRG电路延迟64个周期将“ddr_warm_rst_req_n”置高电平无效，完成本次请求握手；同时整个系统时钟被bypass到晶振时钟，系统进入Slow模式。

步骤5 glb_sft_req高电平有效持续50000个周期（可配置），撤离glb_sft_req低电平；系统CRG电路采样到glb_sft_req电平撤离后，即将非上电复位信号置无效，从而完成本次复位流程。

----结束

5.4.3.3 内部狗溢出复位

内部狗关键特性：

- 特性一：内部狗具有定时的溢出能力，当1.2s @ 66.667MHz，1.6s @ 50MHz，1.6s @ 25MHz（同时时间可调）内没有喂狗信号时，则内部狗溢出，产生一次内部狗复位。
- 特征二：内部狗计时的清0方式，包括：“软件清0”、“掉电复位”、“逻辑喂狗”、“内部狗自身溢出”四种。
- 特性三：“内部狗定时溢出”，最终内部狗的复位功能必须支持软件可配；即只有使能该功能时，内部狗溢出才会引起芯片复位。
- 特性四：内部狗必须依赖系统参考时钟。
- 特性五：WDOG_MODE为内狗模式时（WDOG_MODE下拉），内部狗默认工作。
- 特性六：内部狗溢出产生的复位会触发临终遗言中断。
- 特性七：任意有效复位源撤离时，会触发内部狗重新计数。

Hi309a内部狗溢出导致的非上电复位步骤如下：

步骤1 芯片内部看门狗定时溢出触发内部狗送出复位请求信号inter_wdg_srst_re_n低电平有效信号。

版权所有 © 海思技术有限公司

步骤2 CRG采样到inter_wdg_srst_req_n低电平有效后，向DDRC发出自刷新请求。

步骤3 “ddr_warm_rst_req_n”低电平有效信号。系统CRG启动保护计时，默认50000个晶振cycle，等待DDRC返回握手ack信号。

步骤4 待DDRC正常返回握手信号”ddr_wram_rst_ack_n “低电平有效信号，或者保护计时满足50000个cycle计时后，系统CRG触发非上电复位有效。

步骤5 复位有效后，系统CRG电路延迟64个周期将“ddr_warm_rst_req_n“置高电平无效，完成本次请求握手；同时整个系统时钟被bypass到晶振时钟，系统进入Slow模式。

步骤6 inter_wdg_srst_req_n低电平有效持续200ms（不可配置），系统CRG电路采样到inter_wdg_srst_req_n电平撤离后，即将非上电复位信号置无效，从而完成本次复位流程。

----结束

5.4.3.4 芯片非上电复位流程

芯片上电完成后，PORN管脚保持高电平，此时以上任意类型的非上电复位形式有效后（外部复位源输入，需要经过有效外部复位源检测以及延时输出等电路处理），触发芯片发生非上电复位，具体流程如下：

步骤1 Hi309a输入RSTN有效低电平。

步骤2 系统CRG电路采样到外部复位源检测以及延时输出等电路输出低电平有效信号后，向DDRC发出自刷新请求。

步骤3 “ddr_warm_rst_req_n”低电平有效信号。系统CRG启动保护计时，默认50000个晶振cycle，等待DDRC返回握手ack信号。

步骤4 待DDRC正常返回握手信号“ddr_wram_rst_ack_n”低电平有效信号，或者保护计时满足50000个cycle计时后，系统CRG触发非上电复位有效。

步骤5 复位有效后，系统CRG电路延迟64个周期将“ddr_warm_rst_req_n”置高电平无效，完成本次请求握手；同时整个系统时钟被bypass到晶振时钟，系统进入Slow模式。

步骤6 系统CRG电路采样到输入复位源低电平撤离后，即将非上电复位信号置无效，从而完成本次复位流程。

----结束

5.4.3.5 CPU 系统软复位

CPU系统软复位支持硬件流程和软件流程两种方式的独立复位；其中硬件流程等同于“芯片非上电复位流程”介绍，即只要单板硬件电路将 RSTN管脚独立置低电平有效即可；在此不再赘述。

对于CPU系统软件流程触发的复位，具体步骤如下：

步骤1 通过配置系统SUBCTRL寄存器（具体见可寄存器表单描述）为特定值“0x5555_5555”；由系统控制器送出sc_srst_ap_req高电平有效信号，送至系统CRG。

步骤2 系统CRG采样到sc_srst_ap_req有效后，启动50000个（可配置）晶振cycle计数；计满后自动撤离复位。

版权所有 © 海思技术有限公司

步骤3 系统CRG采样到sc_srst_ap_req有效信号后，向DDRC发出自刷新请求“ddr_warm_rst_req_n”低电平有效信号；系统CRG启动保护计时，默认50000个晶振cycle，等待DDRC返回握手ack信号。

步骤4 待DDRC正常返回握手信号“ddr_wram_rst_ack_n”低电平有效信号，或者保护计时满足50000个cycle计时后，系统CRG触发CPU系统复位。

步骤5 待复位请求撤离后，至此完成CPU系统软件流程触发的复位操作。

----结束

须知

网络场景CPU系统软复位主要用于调试场景，如果用于产品场景需要软件保证先做临终遗言记录，再触发复位。

5.5 模块软复位流程

各功能模块都有自己的软复位请求信号，有系统控制器发出模块软复位请求信号，从而实现各模块的复位功能。软复位撤离顺序由软件来保证。

通常要求模块软复位与模块时钟门控成对配合使用，建议按照如下复位流程：

步骤1 先发起模块软复位。

步骤2 再关闭模块时钟门控。

步骤3 再撤离模块软复位。

步骤4 最后打开模块时钟门控。

----结束

这样既能保证有些模块内部同步复位逻辑生效，也能避免寄存器在模块软复位撤离的时候，高频收敛复位时序的困难。

Hi309a通过以上软复位流程实现复位低频收敛的模块主要有以下几个模块。

表 5-4 模块软复位信号

模块	软复位信号	时钟门控信号
HHA_WRAP	srst_req_hha	icg_en_hha
SA	srst_req_sa_m srst_req_sa_s srst_req_sa_m_2x	icg_en_sa_m icg_en_sa_s icg_en_sa_m_2x
SEC	srst_req_sec_m srst_req_sec_s	icg_en_sec_m icg_en_sec_s

版权所有 © 海思技术有限公司

模块	软复位信号	时钟门控信号
TOP CORESIGHT	srst_req_top_coresight_tresetn srst_req_top_coresight_psysresetn	icg_en_top_coresight_traceclkin icg_en_top_coresight_pclksys
FTE	srst_req_fte	icg_en_fte_apb
GIC	srst_req_gicd	icg_en_gicd
HISS	srst_req_sce_m srst_req_sac_m	icg_en_sce_m
LLC	srst_req_llc	icg_en_llc
QDMA	srst_req_qdma	icg_en_qdma
RMII	srst_req_rmii_800m	icg_en_rmii_800m
TIMER	srst_req_timer	icg_en_timer
EMMC	srst_req_emmc	icg_en_emmc
ESPI	srst_req_espi_axi	icg_en_espi_axi
IFC	srst_req_ifc	icg_en_ifc
SRAM	srst_req_sram srst_req_sram_reg	icg_en_sram icg_en_sram_reg
DDR	srst_req_ddr_ahb srst_req_ddr_1x_bpg srst_req_phy srst_req_status srst_req_ddrc	Normal模式下禁止复位
SMMU	srst_req_poe_smmu_ras srst_req_sppe_smmu_ras	Normal模式下禁止复位

5.6 CPU 异常复位 LANSW 正常转发流程

Hi309a芯片支持CPU异常复位但LANSW正常转发流程，本章节只介绍跟CRG设计相关的内容。该特性仅在网络模式下支持（boot_sel5拉高）。

步骤1 首先，CPU异常复位只针对包括“按键复位、CAN复位、外部狗溢出复位、用户自定义复位、内部狗溢出复位”在内的5种外部复位源触发的CPU复位，不包括“全局软复位、CPU系统软复位”触发的CPU复位。

步骤2 其次，上述5种类型的外部复位触发CPU复位前，由CRG给PPE送出一根高电平有效的指示信号“rst_indi_for_ppe”，通知PPE进行相关流程配置；该指示信号有效后500ms（支持可配，默认500ms）CPU进入复位；之后CPU撤离复位，软件先对PPE进行相关流程配置，然后再对指示信号“rst_indi_for_ppe”进行软件清除操作。

版权所有 © 海思技术有限公司

步骤3 再次CPU异常复位LANSW正常转发，要求整个转发通路包括Serdes在内所有涉及模块以及相关配置都不能被复位；因而在CRG设计上，对应这些模块的时钟在该场景下不能断流，复位不能生效，从而在设计上，对以下时钟复位信号的产生，通过寄存器配置实现：支持该场景时这些时钟复位的产生逻辑模块不能连同CPU复位；不支持该场景时这些时钟复位的产生连同CPU一起复位。列表如表5-5所示。

表 5-5 CPU 异常复位时转发通路相关模块时钟/复位处理

涉及模块	非上电复位时模块时钟不切频	非上电复位时模块复位不复位	配置寄存器名称	SUBCTRL
HILINK SDS CRG	打拍及MAC相关时钟 clk_sds*_pma_rx/ clk_sds*_pma_tx/ clk_sds*_ge_pcs_rx_irs/ clk_sds*_ge_pcs_tx_irs/ clk_ge*_pcs_tx_pre/ clk_ge*_pcs_rx_pre	打拍及MAC相关复位 rst_sds*_pma_rx_n rst_sds*_pma_tx_n rst_sds*_ge_pcs_rx_irs_n rst_sds*_ge_pcs_tx_irs_n	sds*_lansw_comb_rst_inde	hilink_mux subctrl
HILINK SDS CRG	Hilink ahb时钟 clk_hilink_ahb hilink mux sbuctrl时钟 clk_hilink_subctrl	Hilink ahb复位 rst_hilink_ahb_n hilink mux sbuctrl复位 rst_hilink_subctrl_n	lansw_comb_rst_indi_crg	ETH SUBCTRL
PPP_SSU_PA_TOP_CRG	模块内所有时钟	模块内所有复位	lansw_comb_rst_indi_crg	-
PPE_MAG_TOP_CRG	模块内所有时钟	模块内所有复位	lansw_comb_rst_indi_crg	-
MAG_XSEC_CRG	模块内所有时钟	模块内所有复位	lansw_comb_rst_indi_crg	-
MAG_TOP_CRG	模块内所有时钟 mac内的spi时钟，复位期间无时钟，解复位后切换回晶振分频。	模块内所有复位	lansw_comb_rst_indi_crg	-

版权所有 © 海思技术有限公司

涉及模块	非上电复位时模块时钟不切频	非上电复位时模块复位不复位	配置寄存器名称	SUBCTRL
ETH_SUB_CTRL	lansw_comb_rst_indi_crg	默认0b1，使用eth_sub_crg/rst_subctrl_network_online_n复位域。		
SYSCTRL	PLL2/PLL7配置域上电复位域 hi309a_top_crg/sc_pll2_pllfctrl0 hi309a_top_crg/sc_pll2_pllfctrl1 hi309a_top_crg/pll2_bypass_external_n hi309a_top_crg/pll2_peri_mode hi309a_top_crg/sc_pll7_pllfctrl0 hi309a_top_crg/sc_pll7_pllfctrl1 hi309a_top_crg/pll7_bypass_external_n hi309a_top_crg/pll7_peri_mode			

CPU异常复位LANSW正常转发，要求整个转发通路包括Serdes在内所有涉及模块以及相关配置都不能被复位，由于SATA和PCIE相关的serdes会被复位，启动后需要重新初始化serdes，而由于Serdes在内所有涉及模块以及相关配置都不能被复位，需要软件在解复位之后做serdes的初始化前将serdes相关时钟的门控和复位请求配置为该寄存器上电时的默认值，将Serdes相关时钟关闭，初始化serdes完成后再打开Serdes相关时钟。

说明

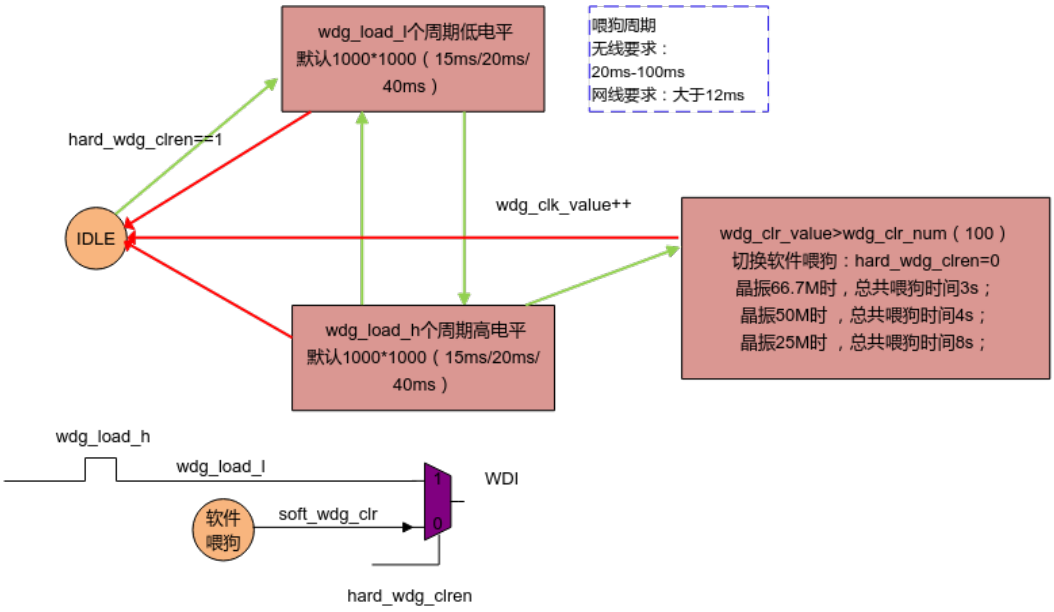
1. 上述列表当中各配置寄存器默认为0b1表示不支持该场景，如果需要支持LANSW独立复位场景，则需要在系统初始化第一步首先配置该类寄存器，配置且回读正确后，再继续其他初始化动作。
2. 上述列表中lansw_comb_rst_indi_crg 为最高优先级，即列表中其他任意一个配置为0b0支持该场景，这两个信号必须先配置支持该场景。

----结束

版权所有 © 海思技术有限公司

5.7 逻辑喂狗

图 5-3 逻辑喂狗



芯片上电解复位后，芯片逻辑喂狗状态机开始工作，输出喂狗信号，喂狗周期20ms~100ms，软件接管前默认执行逻辑喂狗，超时未接管停止喂狗。超时时间3s/4s/8s，对应晶振（66.7MHz/50MHz/25MHz）。

若发生非上电复位，在复位撤离时，切换到逻辑喂狗，保证启动时由芯片逻辑喂内部狗或者外部狗。

5.8 双 BIOS 启动跟 CRG 相关的介绍

- 双BIOS支持在非掉电情况下，由复位管脚RSTN、软件全局软复位产生的复位对BIOS区域进行切换，即上述复位每复位一次，BIOS区域即从BIOS0—>BIOS1或BIOS1—>BIOS0切换一次。
- 支持芯片内部狗溢出复位通过软件可配置选择，是固定BIOS0启动，或是从BIOS0—>BIOS1或BIOS1—>BIOS0切换一次（默认使能切换）。

  说明

  复位管脚RSTN、软件全局软复位可通过软件禁止BIOS切换，默认均为使能。
- 支持双BIOS启动区域指示，通过读sysctrl的寄存器获得该信息。
- 支持BIOS（各区域）启动次数统计，通过读sysctrl的寄存器获得该信息。

说明

本章节只描述与“时钟复位设计”相关的双BIOS启动内容。

版权所有 © 海思技术有限公司

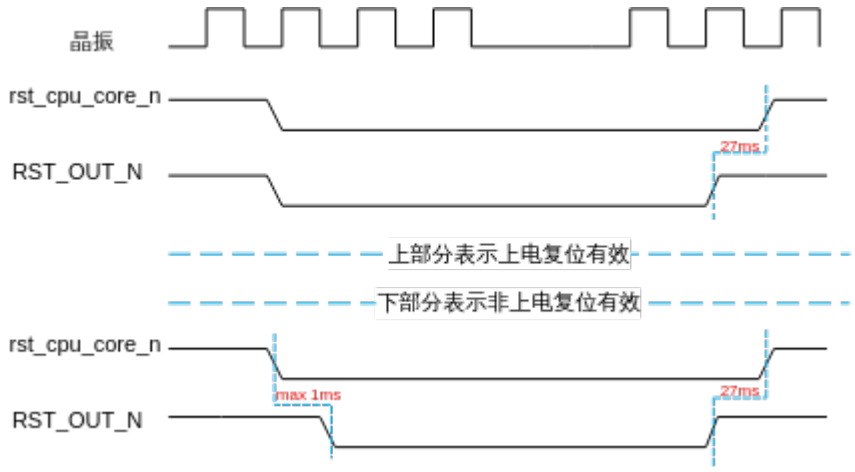
5.9 RST_OUT_N

对于Hi309a芯片外围小芯片诸如NOR FLASH等，建议通过使用RST_OUT_N信号来保证CPU同NOR FLASH之间的复位顺序：CPU复位---->NOR FLASH复位----->NOR FLASH解复位----->CPU解复位。

上电复位有效时：CPU与NOR FLASH同时进入复位；非上电复位有效时，需要保证CPU先进入复位，然后发出RST_OUT_N，启动NOR FLASH复位。

上电复位撤离时或非上电复位撤离时，由设计保证NOR FLASH在芯片管脚RSTN低电平撤销后（约128）*20ns撤离复位；CPU在芯片管脚RSTN低电平撤销后至少27ms撤离复位。

图 5-4 RST_OUT_N 时序



须知

rst_cpu_core_n为芯片内部逻辑产生的信号。

5.10 复位约束

RSTN低电平持续时间要求至少50000个晶振cycle（对应25M晶振时钟，约2ms），因为CRG内部逻辑实现：芯片外围FLASH晚于CPU 复位的时间差为50000个CYCLE，因此如果低电平持续宽度不够，会导致CPU复位了，但外围FLASH没有复位成功。

CAN_RSTN与BUTTON_RSTN低电平宽度至少要保持在2μs以上。

5.11 复位原因记录

复位原因记录寄存器不受复位端控制；软件只能读取寄存器，不能有任何的写操作。复位原因记录，芯片掉电/重新上电可清。

版权所有 © 海思技术有限公司

硬件实现芯片最近5次的复位原因记录，通过SC_RESET_TYPE寄存器记录，最近一次复位原因记录在bit[2:0]，倒数第二次复位原因记录在bit[5:3]，以此类推。

5.11.1 Hi309a 网络络场景下支持以下复位类型

- 0b001：上电复位。
- 0b010：全局软复位（CPU主动触发外狗复位，上报临终遗言中断）。
- 0b011：CAN控制复位（CAN主动触发外狗复位，上报临终遗言中断）。
- 0b100：按键复位（按键触发外狗复位，上报临终遗言中断）。
- 0b101：外部狗溢出复位（外狗溢出复位，上报临终遗言中断）。
- 0b110：内部狗溢出复位（内狗溢出复位，上报临终遗言中断。
- 0b111：CPU系统软复位（不触发外部狗复位，直接写寄存器复位CPU，不触发临终遗言）。
- 0b000：用户自定义复位。

版权所有 © 海思技术有限公司

6 系统初始化

6.1 概述

本章内容描述系统初始化相关特性，并提供一种典型的系统初始化流程作为参考，实际产品的初始化流程可能会有不同。重点描述整个系统的初始化要求，具体模块初始化流程参考模块章节描述。描述与芯片相关的硬件初始化，软件系统的初始化不作为重点。

同以往芯片不同的是：引入了HISS子系统，并区分了HISS和AP的概念。Hi309a的HISS是一个独立的SC300子系统，AP则是传统的业务处理器，对于Hi309a而言就是8个A55 core。

6.2 关键术语

表 6-1 系统初始化关键术语描述

术语	描述
小系统	满足操作系统运行的最小系统。
BootLoader	引导加载程序，是系统加电后运行的第一段软件代码，主要用于引导并启动操作系统。
BSBC	Boot ROM Secure Boot Code，安全启动代码。
eFuse	Electrical-Fuse，一次性的电可编程的Fuse。
AP	Application Processor，在Hi309a中指A55。
BMC	Baseboard Management Controller。
eFuse	Electrical-Fuse，一次性的电可编程的Fuse。
ESBC	External Secure Boot Code，片外安全启动代码。

版权所有 © 海思技术有限公司

术语	描述
ESBC A/B	指网络产品的双点备份机制，ESBC A和ESBC B是完全相同的两个镜像备份，安全启动的时候先从ESBC A启动，若校验失败则跳转ESBC B继续校验。ESBC A为0地址存放的镜像分区，ESBC B为远离0地址存放的镜像分区。
FIQ	Fast Interrupt Request，快速中断请求。
HISS	High Security Subsystem。
IRQ	Interrupt Request，中断请求。
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，从提出到现在已近二十年，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。
SBSA	Server Base System Architecture。
CRG	Clock and Reset Generation module，时钟复位产生模块。
L3D SRAM	LLC工作模式选择SRAM工作模式，作为SRAM使用。

6.3 特性

系统初始化相关特性有：

- 上电启动方式由芯片管脚配置和Efuse一起决定。由于Hi309a支持AP侧与HISS侧两种不同的安全方案，两套方案不共存，所以对应的启动模式也区分AP侧与HISS侧场景，由EFuse域段chip_mode选择。
  - AP侧启动具体支持的启动方式有：
    - AP直接从SPI NOR FLASH非安全启动（仅限调试）。
    - AP直接从NAND FLASH非安全启动（仅限调试）。
    - AP从AP BootRom 安全/非安全启动，然后引导SPI NOR FLASH安全/非安全启动（主要模式）。
    - AP从AP BootRom 安全/非安全启动，然后引导NAND FLASH安全/非安全启动（主要模式）。
    - 芯片提供硬件管脚用于区分主从，从片支持安全启动。
    - 支持主片通过ESPI接口加载从片ESBC。
  - HISS侧具体支持的启动方式有：
    - HISS从HISS BootRom 安全/非安全启动，然后引导AP安全/非安全从SPI NOR FLASH启动。
    - HISS从HISS BootRom 安全/非安全启动，然后引导AP安全/非安全从NAND FLASH启动。

版权所有 © 海思技术有限公司

- SPI Nor、NAND关键启动参数可通过芯片管脚配置，详见功能描述里的**表6-7**、**表6-8**、**表6-9**。
- 支持25M、50M、66.667M晶振。网络场景默认在25M晶振下启动。
- SPI NOR FLASH启动空间支持32MB。
- 提供一组（16个32bits）CPU复位不清零的寄存器给初始化软件使用。
- 片内BOOTROM启动失败原因可查询，过程关键信息通过串口输出，启动失败信息存储到HISS SUBCTRL的noreset 寄存器中。
- 提供软件可控的硬件喂狗信号，输出给单板使用；BOOTROM要实现启动阶段喂狗功能。
- NAND FLASH启动空间支持32MB。
- NAND FLASH启动支持：
  - BootRom NAND多点启动，BootRom支持8份镜像校验。
  - 支持128kByte、256kByte、512kByte BLOCK大小的芯片。
- 支持安全启动，片内BOOTROM启动时间（上电复位撤离到执行片外Nor FLASH Boot程序）要小于2s。
- 支持上电自动加载eFuse，用于memory修复、安全启动、AVS等。
- 只支持核0正常（本核软件主动发起的复位）复位启动，异常复位启动不保证。
- CPU只支持64位启动，启动地址可配置。
- 支持双BOOT启动功能：
  - 企业网固网要求支持单芯片双boot区乒乓启动模式。包括SPI NOR FLASH（1+1模式），NAND FLASH（4+4模式）启动设备。
  - 双区启动区模式为乒乓式互相备份的启动模式，在热启动复位信号触发下，设备启动地址由地址逻辑或FLASH控制器自动在BOOTA和BOOTB间切换。
  - 复位启动后支持通过配置寄存器恢复FLASH全地址访问功能。
  - 双BOOT和多分区启动模式要有寄存器记录当前启动区。
- 启动备份大小支持通过管脚选择，SPI NOR FLASH启动备份大小4M、8M可选，NAND FLASH启动备份大小2M、4M可选。
- 非上电复位支持Efuse内容重新读取到寄存器中。
- 内置HISS协处理器，跟A55核解耦，复位和启动不依赖A55核。
  - 不支持HISS的单独复位。
- 通过AP/HISS管理DDRC内部MCU单元的启动。

6.4 遵循协议

不涉及。

6.5 功能描述

6.5.1 上电复位

AP和HISS同时上电复位。POR_N控制上电复位。RSTN为系统复位输入。

版权所有 © 海思技术有限公司

1. 上电后，整芯片一起进行上电复位，硬件自动完成eFuse加载和相应硬件自动初始化。
2. CRG需要保证启动用到的模块（如SFC）比CPU（AP主核或者HISS内部SC300）先撤离复位（SFC至少27ms）。

CPU启动，由eFuse信息和管脚信息一起确定启动地址，取指进入系统初始化阶段。

在chip_mode为非0xF，系统复位撤离，不管RST_MODE管脚是什么状态，HISS处于掉电状态，AP主核撤离复位状态；

在chip_mode为0xF，正常工作模式下，上电后AP由HISS固件解复位。具体如下：

使用HISS引导AP启动。

上电后，AP和HISS一起进行上电复位，上电复位撤离后硬件自动完成eFuse加载和相应硬件自动初始化； RST_MODE为”0”，所有AP核上电复位保持，HISS启动后引导AP启动。

6.5.2 热复位

非上电系统复位时，不做eFuse动作，部分模块保留复位前的寄存器值，其他模块和上电复位一样（SFC不受影响）。

对于SFC boot要注意：SFC中有部分处于上电复位域的寄存器，这些寄存器需要BSBC启动前进行初始化配置，防止系统异常后热复位启动时SFC寄存器状态不正确。对于不经过BSBC启动直接从SPI NOR FLASH启动的场景，不涉及安全寄存器的清除，但是涉及总线访问相关的寄存器。

6.5.3 eFuse 加载和硬件初始化

上电复位完成后，复位撤离，硬件自动加载eFuse，用于memory修复、安全启动控制等。当Efuse阶段完成后（通过约10ms的延迟确保完成），硬件自动撤离CPU的复位，CPU开始从启动地址取指令进行系统初始化配置。

eFuse阶段的开始由上电复位触发，eFuse阶段结束会给出结束标志信号efuse_done。在正常功能模式下（芯片内部信号dft_mode=0，非外部可见管脚）时，需要保证在上电复位撤离之后能够准确的采样到efuse_done的上升沿。

effuse_done信号在复位时，为高电平，在读写过程中为低电平，读写完成后，efuse_done由低变高。eFuse阶段结束后才允许进入芯片初始化阶段（撤离CPU复位，CPU开始取值执行软件）。

上电复位期间芯片状态如下：

1. 默认Core电压由单板电源芯片提供，并遵循Hi309a电源要求。
2. 所有输出管脚保持三态（复位输出等个别管脚除外），所有双向管脚默认为输入。
3. 芯片所有PLL处于复位状态，上电复位后默认输出为参考时钟。
4. 所有时钟分频器处于复位状态并且无时钟输出。
5. 除了需要软件配置来初始化的寄存器外，其他寄存器均为默认值。
6. 所有的接口都不工作。

上电复位撤离后芯片自动完成如下初始化动作：

1. 芯片上电后自动进入Slow模式，此时系统由外接晶振时钟驱动。

版权所有 © 海思技术有限公司

2. PLL使用参考时钟和上电默认配置自动锁定，可以读SysCtrl里的寄存器SC_PLL_LOCK_STATUS查看是否锁定。
3. 只有部分片内Device可以工作，如总线（Slow 模式）、SYSCTRL、Timer、SFC，需要高速时钟的DDRC、PCIe等无法工作。
4. 芯片上电后使用默认地址映射。启动空间根据管脚信息决定是映射到SPI NOR FLASH、NANDFLASH还是BOOTROM。
5. CPU自动从启动地址开始取指令，开始BOOT启动流程，逻辑使用eFuse和管脚信息，自动判断是否安全启动等。
6. 上电复位撤离后，JTAG TAP控制器可以采样管脚信息正常工作。

6.5.4 启动真值表

Hi309a的正常工作模式是，HISS启动场景：HISS执行BOOTROM（HISS）的BSBC后，再引导AP（A55）从外部的FLASH介质（ESBC）加载；AP启动场景：AP执行BOOTROM（AP）的BSBC后，再引导AP（A55）从外部的FLASH介质（ESBC）加载。

Hi309a也支持直接从外接的SPI NOR FLASH启动加载，但仅作为调试模式。需要设置eFuse里的efuse_ns_forbid位，以及芯片外部管脚的BOOT_SEL0和RST_MODE，用于选择Boot模式，真值表如下表6-2所示。

表 6-2 AP（A55）启动真值表

名称	efuse_chip_mode	efuse_ns_forbid	RST_MODE	BOOT_SEL0	启动场景
说明	**芯片模式，位于Efuse中。含义：非“0xF”AP侧启动；“0xF”HISS侧启动。**	**efuse_ns_forbid，位于AP/HISS 侧Efuse中。在内部启动逻辑中优先级最高。含义：“0”非安全启动模块；非“0”安全启动模式。**	**RST_MODE为0，AP不能自己启动，需要HISS启动后引导A55启动。**	**上电采样得到。含义：启动介质为“0” SPI NOR FLASH；“1” NAND FLASH。**	
真值表	非0xF	0	X	0	AP侧非安全启动（调试模式），AP直接从SPI NOR FLASH非安全启动。
	非0xF	0	X	1	AP侧非安全启动（调试模式），AP直接从NAND FLASH非安全启动。
	非0xF	1	X	0	AP侧安全启动：AP从BSBC安全启动，再引导SPI NOR FLASH启动。

版权所有 © 海思技术有限公司

名称	efuse_chip_mode	efuse_ns_forbid	RST_MODE	BOOT_SEL0	启动场景
说明	芯片模式，位于Efuse中。 含义： 非“0xF”AP侧启动； “0xF”HISS侧启动。	efuse_ns_forbid，位于AP/HISS 侧Efuse中。在内部启动逻辑中优先级最高。含义： “0”非安全启动模块； 非“0”安全启动模式。	RST_MODE为0，AP不能自己启动，需要HISS启动后引导A55启动。	上电采样得到。含义：启动介质为“0” SPI NOR FLASH；“1” NAND FLASH。	
	非0xF	1	X	1	AP侧安全启动：AP从BSBC安全启动，再引导NAND FLASH启动。
	0xF	0	0	X	HISS侧非安全启动（调试模式）：HISS从BSBC非安全启动，AP需要由HISS引导启动。
	0xF	0	1	0	预留，当前非法，禁止使用。
	0xF	0	1	1	预留，当前非法。
	0xF	1	X	X	HISS侧安全启动：HISS从BSBC安全启动，HISS启动后引导AP启动。

备注：目前的设计，efuse_ns_forbid有32bit，全零是非安全启动，非零均为安全启动。

此外还需要设置芯片外部管脚BOOT_MODE，用于选择Boot的存储器件。

版权所有 © 海思技术有限公司

表 6-3 启动模式真值表

efuse_chip_mode（Efuse）芯片模式，位于Efuse中。 **含义：** **非“0xF”AP侧启动；** **“0xF”HISS侧启动。**	**efuse_ns_forbid（Efuse）** **含义：非“0”安全启动模式；“0”非安全启动模块。**	**BOOT_MODE** **含义：“1”片内BSBC启动；“0”片外SPI Nor FLASH/NAND FLASH启动。**	**安全启动场景**
非0xF	0	0	A55主核从片外介质启动（调试用）。
非0xF	0	1	A55主核从BSBC非安全启动。
非0xF	1	X	A55主核从BSBC安全启动。
0xF	0	0	预留，当前非法。
0xF	0	1	HISS从BSBC非安全启动。
0xF	1	X	HISS从BSBC安全启动。

从上述真值表可以看出，eFuse控制安全还是非安全启动，芯片设计要保证Efuse的控制优先级最高。

如下表6-4所示，是完整的安全启动真值表。

版权所有 © 海思技术有限公司

表 6-4 AP 侧启动真值表

名称	efuse_chip_mode	efuse_ns_forbid	BOOT_SEL1	BOOT_MODE	BOOT_SEL0	启动场景
说明	芯片模式，位于Efuse中。 含义： 非“0xF”AP侧启动； “0xF”HISS侧启动。	efuse_ns_forbid，位于Efuse中。在内部启动逻辑中优先级最高。 含义： “0”非安全启动模块； 非“0”安全启动模式。	主从片模式启动模式脚，1bit，上电采样管脚。含义： “0”主片； “1”从片。	内部外部启动模式脚，1bit，上电采样管脚。仅在efuse_ns_forbid为0时有意义，含义： “0”片外介质启动； “1”片内BSBC启动。	上电采样得到。含义：启动介质为“0”SPI NOR FLASH；“1”NAND FLASH。	
真值表	非0xF	0	X	0	0	AP侧非安全启动（调试模式）：AP从SPI NOR FLASH非安全启动。
	非0xF	0	X	0	1	AP侧非安全启动（调试模式）：AP从NAND FLASH非安全启动。
	非0xF	0	X	1	0	AP侧非安全启动：AP从BSBC非安全启动，再引导SPI NOR FLASH启动。

版权所有 © 海思技术有限公司

名称	efuse_chip_mode	efuse_ns_forbid	BOOT_SEL1	BOOT_MODE	BOOT_SEL0	启动场景
说明	芯片模式，位于Efuse中。 含义： 非“0xF”AP侧启动； “0xF”HISS侧启动。	efuse_ns_forbid，位于Efuse中。在内部启动逻辑中优先级最高。 含义： “0”非安全启动模块； 非“0”安全启动模式。	主从片模式启动模式脚，1bit，上电采样管脚。含义： “0”主片； “1”从片。	内部外部启动模式脚，1bit，上电采样管脚。仅在efuse_ns_forbid为0时有意义，含义： “0”片外介质启动； “1”片内BSBC启动。	上电采样得到。含义：启动介质为“0”SPI NOR FLASH；“1”NAND FLASH。	
	非0xF	0	X	1	1	AP侧非安全启动：AP从BSBC非安全启动，再引导NAND FLASH启动。
	非0xF	1	X	X	0	AP侧安全启动：AP从BSBC安全启动，再引导SPI NOR FLASH启动。
	非0xF	1	X	X	1	AP侧安全启动：AP从BSBC安全启动，再引导NAND FLASH启动。

版权所有 © 海思技术有限公司

表 6-5 HISS 侧启动真值表

名称	efuse_chip_mode	efuse_ns_forbid	RST_MODE	BOOT_SEL1	BOOT_MODE	BOOT_SEL0	启动场景
说明	**芯片模式，位于Efuse中。含义：非“0xF”AP侧启动；“0xF”HISS侧启动。**	**efuse_ns_forbid，位于Efuse中。在内部启动逻辑中优先级最高。含义：“0”非安全启动模块；非“0”安全启动模式。**	**RST_MODE为0，AP不能自己启动，需要HISS启动后引导A55启动。**	**主从片模式启动模式脚，1bit，上电采样管脚。含义：“0”主片;“1”从片。**	**内部外部启动模式脚，1bit，上电采样管脚。仅在efuse_ns_forbid为0时有意义，含义：“0”片外介质启动;“1”片内BSBC启动。**	**上电采样得到。含义：启动介质为“0”SPI NOR FLASH;“1”NAND FLASH。**	
-	0xF	0	0	X	0	0	**预留，当前非法。**
	0xF	0	0	X	0	1	**预留，当前非法。**
	0xF	0	0	X	1	0	HISS侧非安全启动（调试模式）：HISS从BSBC非安全启动，再引导SPI NOR FLASH启动。
	0xF	0	0	X	1	1	HISS侧非安全启动（调试模式）：HISS从BSBC非安全启动，再引导NAND FLASH启动。
	0xF	0	1	X	0	0	**（预留，当前非法，禁止使用）**
	0xF	0	1	X	0	1	预留，当前非法。

版权所有 © 海思技术有限公司

名称	efuse_chip_mode	efuse_ns_forbid	RST_MODE	BOOT_SEL1	BOOT_MODE	BOOT_SEL0	启动场景
说明	芯片模式，位于Efuse中。 含义： 非“0xF”AP侧启动； “0xF” HISS侧启动。	efuse_ns_forbid，位于Efuse中。在内部启动逻辑中优先级最高。含义： “0”非安全启动模块； 非“0”安全启动模式。	RST_MODE为0，AP不能自己启动，需要HISS启动后引导A55启动。	主从片模式启动模式脚，1bit，上电采样管脚。含义： “0”主片； “1”从片。	内部外部启动模式脚，1bit，上电采样管脚。仅在efuse_ns_forbid为0时有意义，含义： “0”片外介质启动； “1”片内BSBC启动。	上电采样得到。含义：启动介质为 “0”SPI NOR FLASH； “1”NAND FLASH。	
	0xF	0	1	X	1	X	预留，当前非法。
	0xF	1	X	X	X	0	HISS侧安全启动：HISS从BSBC安全启动，HISS启动后引导AP从SPI NOR FLASH启动。
	0xF	1	X	X	X	1	HISS侧安全启动：HISS从BSBC安全启动，HISS启动后引导AP从NAND FLASH启动。

参照上述Boot模式真值表，芯片设计要保证：

1. eFuse控制安全还是非安全启动，芯片设计要保证eFuse的控制优先级最高。efuse_ns_forbid为非“0”时，efuse_chip_mode为非“0xF”时，不管其他PAD是什么值，必须要AP从BSBC安全启动；efuse_ns_forbid为非“0”时，efuse_chip_mode为“0xF”时，不管其他PAD是什么值，必须要HISS从BSBC安全启动，然后引导AP启动，AP不能自己启动。
2. efuse_chip_mode为非“0xF”时，不管RST_MODE是什么值，HISS都不启动，A55自行启动；efuse_chip_mode为“0xF”，RST_MODE为“0”时，不管BOOT SEL是什么值，A55都不能自己启动，需要HISS启动后引导A55启动。

版权所有 © 海思技术有限公司

3. Hi309a HISS中SC300核只支持从片上Boot ROM启动。

表 6-6 启动晶振频率真值表

BOOT_SEL4	BOOT_SEL3	安全启动场景
0	0	50M
0	1	66.667M
1	X	25M

须知

1. 单板硬件需要保证BOOT_SEL符合上述真值表。
2. 逻辑代码需要使用BOOT_SEL和来自eFuse的efuse_ns_forbid、efuse_chip_mode位，进行地址映射译码，且Efuse的efuse_ns_forbid位优先。
3. BOOTROM里的软件，需要使用efuse_ns_forbid、efuse_chip_mode和 BOOT_SEL一起选择跳转分支。

说明

当前Hi309a仅支持SFC 3Bytes 单线boot模式。

SPI NOR FLASH boot时支持如下时序选择：

表 6-7 SFC 接口时钟频率真值表

BOOT_CFG_N1	BOOT_CFG_N0	SFC 分频	备注
0	0	6	SPI NOR FLASH工作在50MHz。
0	1	8	SPI NOR FLASH工作在37MHz。
1	0	12	SPI NOR FLASH工作在25MHz。
1	1	24	SPI NOR FLASH工作在12.5MHz。

表 6-8 NAND BOOT 模式真值表

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0000000	001	011	00	0	0	2k page size，16bits/1024Byte ecc，64page/block，4个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0000001	001	010	00	1	0	2k page size，8bits/1024Byte ecc，64page/block，5个地址周期。
0000010	001	010	01	1	0	2k page size，8bits/1024Byte ecc，128page/block，5个地址周期。
0000011	001	010	10	1	0	2k page size，8bits/1024Byte ecc，256page/block，5个地址周期。
0000100	001	011	00	1	0	2k page size，16bits/1024Byte ecc，64page/block，5个地址周期。
0000101	001	011	01	1	0	2k page size，16bits/1024Byte ecc，128page/block，5个地址周期。
0000110	001	011	10	1	0	预留，当前非法。
0000111	001	100	00	1	0	2k page size，24bits/1024Byte ecc，64page/block，5个地址周期。
0001000	001	100	01	1	0	2k page size，24bits/1024Byte ecc，128page/block，5个地址周期。
0001001	001	100	10	1	0	2k page size，24bits/1024Byte ecc，256page/block，5个地址周期。
0001010	001	111	00	1	0	2k page size，28bits/1024Byte ecc，64page/block，5个地址周期。
0001011	001	111	01	1	0	2k page size，28bits/1024Byte ecc，128page/block，5个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0001100	001	111	10	1	0	2k page size，28bits/1024Byte ecc，256page/block，5个地址周期。
0001101	010	010	00	1	0	4k page size，8bits/1024Byte ecc，64page/block，5个地址周期。
0001110	010	010	01	1	0	4k page size，8bits/1024Byte ecc，128page/block，5个地址周期。
0001111	010	010	10	1	0	4k page size，8bits/1024Byte ecc，256page/block，5个地址周期。
0010000	010	011	00	1	0	4k page size，16bits/1024Byte ecc，64page/block，5个地址周期。
0010001	010	011	01	1	0	4k page size，16bits/1024Byte ecc，128page/block，5个地址周期。
0010010	010	011	10	1	0	4k page size，16bits/1024Byte ecc，256page/block，5个地址周期。
0010011	010	100	00	1	0	4k page size，24bits/1024Byte ecc，64page/block，5个地址周期。
0010100	010	100	01	1	0	4k page size，24bits/1024Byte ecc，128page/block，5个地址周期。
0010101	010	100	10	1	0	4k page size，24bits/1024Byte ecc，256page/block，5个地址周期。
0010110	010	111	00	1	0	4k page size，28bits/1024Byte ecc，64page/block，5个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0010111	010	111	01	1	0	4k page size，28bits/1024Byte ecc，128page/block，5个地址周期。
0011000	010	111	10	1	0	4k page size，28bits/1024Byte ecc，256page/block，5个地址周期。
0011001	011	100	00	1	1	8k page size，24bits/1024Byte ecc，64page/block，randmizer，5个地址周期。
0011010	011	100	00	1	0	8k page size，24bits/1024Byte ecc，64page/block，5个地址周期。
0011011	011	100	01	1	1	8k page size，24bits/1024Byte ecc，128page/block，randmizer，5个地址周期。
0011100	011	100	01	1	0	8k page size，24bits/1024Byte ecc，128page/block，5个地址周期。
0011101	011	100	10	1	1	8k page size，24bits/1024Byte ecc，256page/block，randmizer，5个地址周期。
0011110	011	100	10	1	0	8k page size，24bits/1024Byte ecc，256page/block，5个地址周期。
0011111	011	100	11	1	1	8k page size，24bits/1024Byte ecc，512page/block，randmizer，5个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0100000	011	100	11	1	0	8k page size，24bits/1024Byte ecc，512page/block，5个地址周期。
0100001	011	111	00	1	1	8k page size，28bits/1024Byte ecc，64page/block，randmizer，5个地址周期。
0100010	011	111	00	1	0	8k page size，28bits/1024Byte ecc，64page/block，5个地址周期。
0100011	011	111	01	1	1	8k page size，28bits/1024Byte ecc，128page/block，randmizer，5个地址周期。
0100100	011	111	01	1	0	8k page size，28bits/1024Byte ecc，128page/block，5个地址周期。
0100101	011	111	10	1	1	8k page size，28bits/1024Byte ecc，256page/block，randmizer，5个地址周期。
0100110	011	111	10	1	0	8k page size，28bits/1024Byte ecc，256page/block，5个地址周期。
0100111	011	111	11	1	1	8k page size，28bits/1024Byte ecc，512page/block，randmizer，5个地址周期。
0101000	011	111	11	1	0	8k page size，28bits/1024Byte ecc，512page/block，5个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0101001	011	101	00	1	1	8k page size，40bits/1024Byte ecc，64page/block，randmizer，5个地址周期。
0101010	011	101	00	1	0	8k page size，40bits/1024Byte ecc，64page/block，5个地址周期。
0101011	011	101	01	1	1	8k page size，40bits/1024Byte ecc，128page/block，randmizer，5个地址周期。
0101100	011	101	01	1	0	8k page size，40bits/1024Byte ecc，128page/block，5个地址周期。
0101101	011	101	10	1	1	8k page size，40bits/1024Byte ecc，256page/block，randmizer，5个地址周期。
0101110	011	101	10	1	0	8k page size，40bits/1024Byte ecc，256page/block，5个地址周期。
0101111	011	101	11	1	1	8k page size，40bits/1024Byte ecc，512page/block，randmizer，5个地址周期。
0110000	011	101	11	1	0	8k page size，40bits/1024Byte ecc，512page/block，5个地址周期。
0110001	011	110	00	1	1	8k page size，64bits/1024Byte ecc，64page/block，randmizer，5个地址周期。
0110010	011	110	00	1	0	8k page size，64bits/1024Byte ecc，64page/block，5个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0110011	011	110	01	1	1	8k page size，64bits/1024Byte ecc，128page/block，randmizer，5个地址周期。
0110100	011	110	01	1	0	8k page size，64bits/1024Byte ecc，128page/block，5个地址周期。
0110101	011	110	10	1	1	8k page size，64bits/1024Byte ecc，256page/block，randmizer，5个地址周期。
0110110	011	110	10	1	0	8k page size，64bits/1024Byte ecc，256page/block，5个地址周期。
0110111	011	110	11	1	1	8k page size，64bits/1024Byte ecc，512page/block，randmizer，5个地址周期。
0111000	011	110	11	1	0	8k page size，64bits/1024Byte ecc，512page/block，5个地址周期。
0111001	100	101	00	1	1	16k page size，40bits/1024Byte ecc，64page/block，randmizer，5个地址周期。
0111010	100	101	00	1	0	16k page size，40bits/1024Byte ecc，64page/block，5个地址周期。
0111011	100	101	01	1	1	16k page size，40bits/1024Byte ecc，128page/block，randmizer，5个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
0111100	100	101	01	1	0	16k page size，40bits/1024Byte ecc，128page/block，5个地址周期。
0111101	100	101	10	1	1	16k page size，40bits/1024Byte ecc，256page/block，randmizer，5个地址周期。
0111110	100	101	10	1	0	16k page size，40bits/1024Byte ecc，256page/block，5个地址周期。
0111111	100	101	11	1	1	16k page size，40bits/1024Byte ecc，512page/block，randmizer，5个地址周期。
1000000	100	101	11	1	0	16k page size，40bits/1024Byte ecc，512page/block，5个地址周期。
1000001	100	110	00	1	1	16k page size，64bits/1024Byte ecc，64page/block，randmizer，5个地址周期。
1000010	100	110	00	1	0	16k page size，64bits/1024Byte ecc，64page/block，5个地址周期。
1000011	100	110	01	1	1	16k page size，64bits/1024Byte ecc，128page/block，randmizer，5个地址周期。
1000100	100	110	01	1	0	16k page size，64bits/1024Byte ecc，128page/block，5个地址周期。

版权所有 © 海思技术有限公司

BOOT_CFG_W6～0	page size	ecc_type	block_size	addr_num	randmizer_en	启动模式
1000101	100	110	10	1	1	16k page size，64bits/1024Byte ecc，256page/block，randmizer，5个地址周期。
1000110	100	110	10	1	0	16k page size，64bits/1024Byte ecc，256page/block，5个地址周期。
1000111	100	110	11	1	1	16k page size，64bits/1024Byte ecc，512page/block，randmizer，5个地址周期。
1001000	100	110	11	1	0	16k page size，64bits/1024Byte ecc，512page/block，5个地址周期。
1001001~1111111	X	X	X	X	X	预留，当前非法。

另外，对于NANDC启动有额外的管脚控制对应模式。

NAND FLASH不支持同步NAND BOOT。

表 6-9 NANDC 启动模式控制

BOOT_CFG_W7	自适应模式	备注
0	自适应启动无效（BOOT_CFG_W4~0）	上电默认值。
1	自适应启动有效	-

6.5.5 双 BOOT 启动方案

6.5.5.1 Hi309a 的硬件逻辑切换的双 BOOT

双BOOT也称为双BIOS，基本原理主要继承自Hi30xx系列芯片。主要的不同在于：HISS的引入。

版权所有 © 海思技术有限公司

并且因为HISS的引入额外增加了BSBC双bios。

表 6-10 Hi309a 硬件双 BOOT 启动方案

双BOOT区乒乓启动模式	企业网固网要求支持单芯片双BOOT区乒乓启动模式。包括SPI NOR FLASH（1+1模式），NAND FLASH（4+4模式）启动设备。 双区启动区模式为乒乓式互相备份的启动模式，在热启动复位信号触发下，设备启动地址由FLASH控制器自动在BOOTA和BOOTB间切换。上电启动是默认从BOOTA启动（物理0地址）。
上电启动	固定BOOTA区（固定SPI NOR FLASH0）。
内部狗溢出	可软件可配置固定BOOTB区或BOOT A区，可切换（默认值切换）。
双区翻转触发条件	必须同时满足以下条件： 1. BOOT_SEL2上拉。 2. 复位源使能寄存器SC_CRG_RST_ENABLE对应类型复位切换使能为1。 3. 双bios切换使能寄存器SC_DBIOS_EN对应硬切换FLASH类型使能为1。
硬件清狗（WDI）	硬件清狗，清除内部与外部狗； Hi309a输出到外部706芯片的WDI引脚固定。
启动区域指示	SysCtrl内部具有寄存器SC_BSBC_BIOS_CFG表示本次正常启动时所选择的BOOTA区域（BOOTA或BOOTB）。
FLASH全地址访问	复位启动后支持通过配置寄存器恢复FLASH全地址访问功能。

表 6-11 启动备份大小真值表

模块	BOOT_CFG_W8	BOOT_CFG_N3	BOOT_CFG_N4	备份大小	备注
SPI Nor FLASH	NA	0	0	切换到SFC片选1	-
	NA	0	1	切换到SFC片选1	-
	NA	1	0	4M	SFC片选0。
	NA	1	1	8M	SFC片选0。
NAND FLASH	0	NA	NA	2M	-
	1	NA	NA	4M	-

版权所有 © 海思技术有限公司

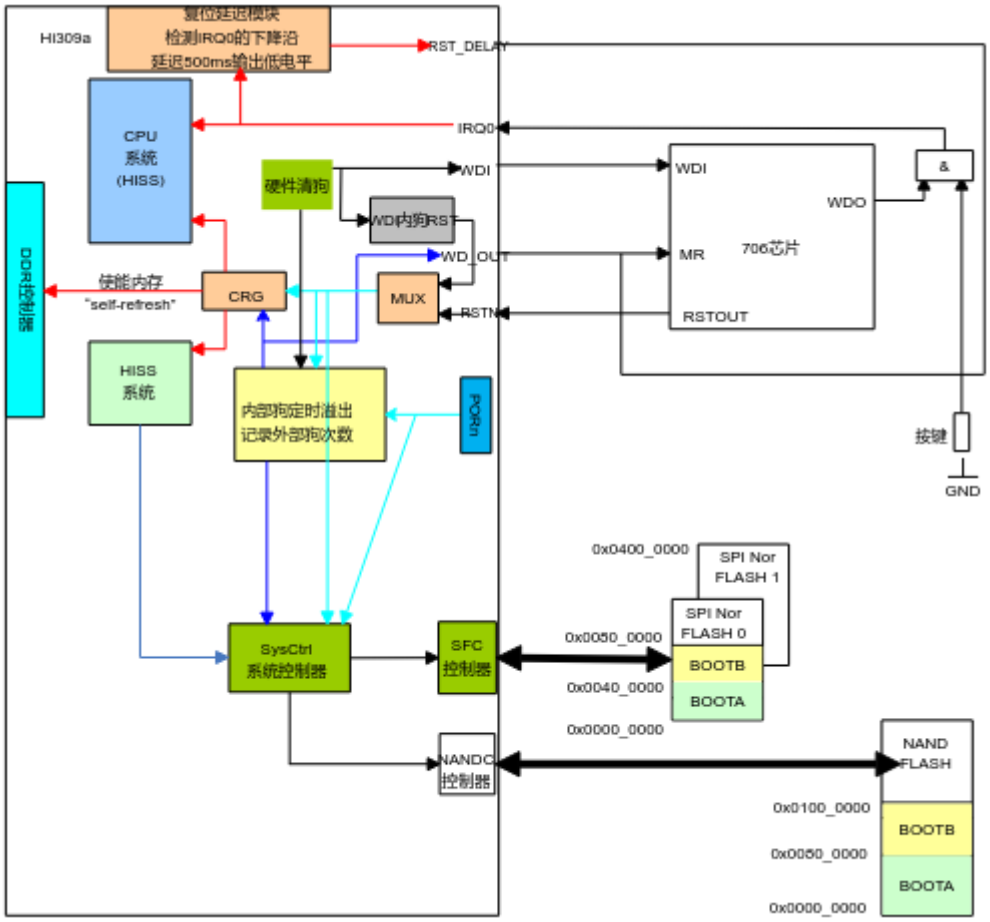
NANDC FLASH使用BOOT_CFG_W8来控制备份大小选择；SPI NOR FLASH使用BOOT_CFG_N3控制备份区在片选0或片选1切换，BOOT_CFG_N4控制备份大小选择。

说明

企业网固网要求支持单芯片双BOOT区乒乓启动模式。

硬件双BOOT框图如图6-1所示。

图 6-1 Hi309a 双 BOOT 方案图



Hi309a支持2种双BOOT启动方式：

- SPI NOR FLASH启动，芯片管脚BOOT_SEL0为“0b0”。
- NAND FLASH启动，芯片管脚BOOT_SEL0为“0b1”。
- 双BOOT启动功能通过上电采样BOOT_SEL2输入值来使能和禁止，对于不需要支持双BOOT启动的产品应用场景，需要将 BOOT_SEL2配置“下拉”完成上电字采样，即禁止该功能。
- 双BOOT支持在非掉电情况下，由复位管脚RSTN、软件全局软复位产生的复位对BOOT区域进行切换，即上述复位每复位一次，BOOT区域即从BOOTB—>BOOTA

版权所有 © 海思技术有限公司

或BOOTA—>BOOTB切换一次。支持芯片内部狗溢出复位通过软件可配置选择，是固定BOOTB启动，或是从BOOTA—>BOOTB或BOOTB—>BOOTA切换一次。

- 支持双BOOT启动区域指示，通过读sysctrl的寄存器获得该信息。
- 支持BOOT（各区域）启动次数统计，通过读sysctrl的寄存器获得该信息。

附：相关寄存器。

图 6-2 SC_CRG_RST_ENABLE 复位源使能寄存器（Sysctrl）

SC_CRG_RST_ENABLE	复位源使能寄存器		0x2300	32			0x00007FF9	reserved	31:15	-	0x00000		保留		
								porn_enable	14	RW_POR	0x1		porn复位使能寄存器 1：无效； 0：有效；		
								inter_wdg_dbios_enable	13	RW_POR	0x1		内部狗复位触发双BIOS切换使能位 0：无效； 1：有效		
								rstn_can_dbios_enable	12	RW_POR	0x1		CAN复位触发双BIOS切换使能位 0：无效； 1：有效		
								rstn_button_dbios_enable	11	RW_POR	0x1		BUTTON复位触发双BIOS切换使能位 0：无效； 1：有效		
								rstn_wdog_dbios_enable	10	RW_POR	0x1		外部狗复位触发双BIOS切换使能位 0：无效； 1：有效		
								rstn_user_define_dbios_enable	9	RW_POR	0x1		user_define复位触发双BIOS切换使能位 0：无效； 1：有效		
								rstn_glb_sft_dbios_enable	8	RW_POR	0x1		全局软复位复位触发双BIOS切换使能位 0：无效； 1：有效		
								rstn_ap_sft_dbios_enable	7	RW_POR	0x1		AP软复位触发双BIOS切换使能位 0：无效； 1：有效		
								glb_sft_rstn_enable	6	RW_POR	0x1		全局软复位使能位 0：无效； 1：有效		
								rstn_user_define_enable	5	RW_POR	0x1		user_define复位触发使能位 0：无效； 1：有效		
								rstn_wdog_enable	4	RW_POR	0x1		外部狗复位使能位 0：无效； 1：有效		
								rstn_button_enable	3	RW_POR	0x1		BUTTON复位使能位 0：无效； 1：有效		
								porn_ip_enable	2:1	RW_POR	0x0		MISS porn_ip_in使能信号，默认2'b00		
								rstn_can_enable	0	RW_POR	0x1		芯片CAN复位使能寄存器；		

图 6-3 SC_DBIOS_EN 双 BIOS 使能（PERI Subctrl）

SC_DBIOS_EN	双BIOS使能		0x230	32			0x00000003	reserved	31:2	-	0x00000000		保留
								dbios_en_ifc	1	RW	0x1		ifc双Bios使能： 1：表示使能 0：表示不使能
								dbios_en_nandc	0	RW	0x1		NAND双Bios使能： 1：表示使能 0：表示不使能

图 6-4 SC_BSBC_BIOS_CFG 记录 BSBC 校验成功的 ESBC 分区寄存器（Sysctrl）

SC_BSBC_BIOS_CFG	记录BSBC校验成功的BSBC分区寄存		0x22F0	32			0x00000000	reserved	31:2	-	0x00000000		保留
								bsbc_bios_cfg	1:0	RW_POR	0x0		记录BSBC校验成功的BSBC分区 00:默认值，BSBC没启动 01: BSBC从分区A启动 10: BSBC从分区B启动 11: 保留

图 6-5 SC_WDOG_ST（Sysctrl）

SC_WDOG_ST	外部狗复位次数寄存器（在上电复位和内存自刷新复位时该寄存器会被清成复位值。）		0x2948	32			0x00000000	reserved	31:15	-	0x00000		保留。
								bios_sel_pos_cnt	14:10	RO	0x00		bios_sel的下降计数
								bios_sel_neg_cnt	9:5	RO	0x00		bios_sel的下降计数
								reserved	4:2	-	0x0		保留
								bios_sel	1	RO	0x0		双BIOS选择： 0：选择分区1； 1：选择分区2；
								rstn_external_active_src	0	RO	0x0		临终遗言中断请求，低电平有效

图 6-6 SC_PORESET_CFG 校验分区镜像寄存器（Sysctrl）

SC_PORESET_CFG	校验分区镜像寄存		0x22F4	32			0x00000000	sc_poreset_cfg	31:0	RW_POR	0x00000000		用于区分BSBC先校验L2/L1分区镜像

版权所有 © 海思技术有限公司

说明

硬件双bios的典型使用方法是假如当前是由biosB启动的，则先升级biosA，并使能复位切换bios。则升级结束复位后将会由升级了的biosA启动，biosA启动成功则继续把biosB也刷新，如果biosA启动失败则会再次复位后由biosB使用升级前的bios固件启动。

6.5.5.2 Hi309a 网络软件双 BOOT

网络场景下，当不使能硬件双BIOS时，即BOOT_SEL2为0，支持BSBC软件双BIOS。BSBC通过读取BOOT_SEL2和寄存器SC_PORESET_CFG，确定从哪个分区开始加载。

表 6-12 启动校验分区真值表（软件使用）

BOOT_SEL2	SC_PORESET_CFG	分区选择
1	X	ESBC A
0	0	ESBC B
	1	ESBC A

6.5.6 Hi309a 网络启动流程

6.5.6.1 HISS 启动流程

步骤1 上电后，芯片上电复位，硬件自动完成eFuse加载和相应硬件自动初始化。

步骤2 HISS内部逻辑等待efuse_done后， 撤离SC300的复位，根据eFuse信息和管脚信息一起确定启动方式，取指进入上电启动过程。

步骤3 逻辑先判断来自eFuse的efuse_ns_forbid信号，如果为安全启动则从BOOTROM进行BSBC安全启动，然后跳转到HISS内部的SRAM；

步骤4 如果来自eFuse的efuse_ns_forbid信息指示为非安全启动，则判断来自管脚采样信息BOOT_MODE如果为“1”，则从BOOTROM进行BSBC非安全启动，然后跳转到HISS内部的SRAM；BOOT_MODE如果为“0”，则从片外SPI NOR FLASH启动（此模式仅用作BSBC代码有问题Debug调试，正式产品不采用），如果此HISS所在的芯片上不带FLASH，BOOT_MODE必须为“1”。

步骤5 SC300负责HISS系统内相关模块。

步骤6 HISS跳转到内部SRAM，运行HISS固件代码。

步骤7 执行完HISS的固件代码后，由AP CPU执行初始化。

步骤8 系统初始化完成后，L3D SRAM恢复为L3 cache。

----结束

6.5.6.2 A55 启动流程

步骤1 上电后，芯片进行上电复位，硬件自动完成eFuse加载和相应硬件自动初始化。

版权所有 © 海思技术有限公司

步骤2 如果是安全启动，所有AP核保持复位状态，HISS启动后初始化LLC为SRAM使用，并将启动代码搬移到LLC，解复位AP主核从L3D安全空间启动；主核启动后引导其他核启动（打开LLC代码需要在DDR运行）。

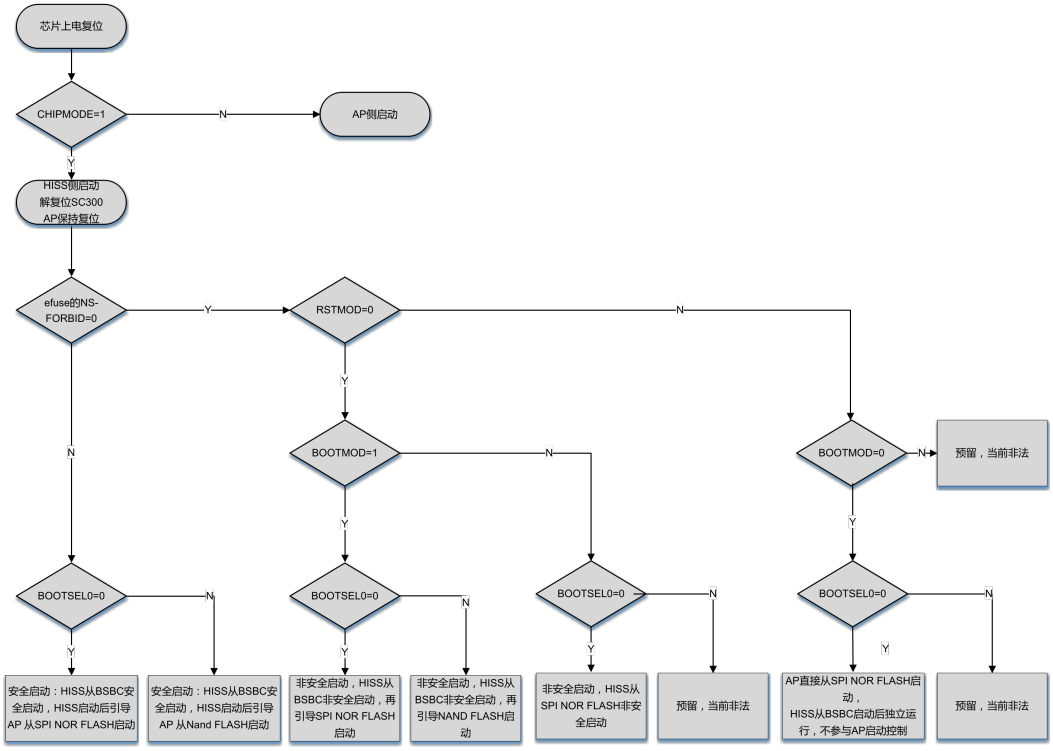
步骤3 如果是非安全片内启动，RST_MODE为”0”，所有AP核也保持复位状态；HISS启动后，解复位AP主核从SFC或NAND启动；主核启动后引导其他核启动。

步骤4 如果是从片外介质启动，等待efuse_done后，AP主核撤离复位，其他核保持复位状态或进WFI状态，主核启动后引导其他核启动。

步骤5 AP主核负责AP系统内相关模块的BIOS初始化。

----结束

图 6-7 Hi309a 网络场景启动流程



6.5.7 软复位

A55单核软复位流程

步骤1 配置System Control Register中的C位为0，关闭Cache，避免新的数据allocate到Cache中。

步骤2 配置CPUECTLR_EL1寄存器，将[15:13]位配置为0，将[12:10]位配置为0b101，关闭Cache的预取功能。

步骤3 执行ISB，确保CPUECTLR_EL1寄存器的修改生效。

步骤4 执行DSB，确保关闭预取前的访存结束。

步骤5 对DCache进行Clean and invalidate操作，确保其他CORE不会到本CPU CORE的DCache中获取数据。

版权所有 © 海思技术有限公司

步骤6 刷LLC，将LLC的数据刷到DDR中，确保后续CPU发出nocacheable访问时直接从DDR中获取的数据是正确的。

步骤7 配置中断控制器，确保中断控制器不再向本CPU CORE发送中断。

步骤8 配置DBGOSDLR寄存器的DLK位为1，禁止调试。

步骤9 执行ISB，确保之前的操作全部完成。

步骤10 执行DSB，确保DCache维护等操作完成。

步骤11 执行WFI指令。

步骤12 以下动作需其他核执行（非当前复位核）。

步骤13 持续读取系统控制器寄存器SC_CPU_STANDBYWFI_ST中相应位，直到读取为1，确保被复位核已进入WFI模式。

步骤14 配置SC_CLUS_CRG_RESET_REQ寄存器中的srst_req_cpu_cluX[x]和srst_req_cpu_por_cluX[x]（X表示0-1,x表示0-3）位为1，产生单核软复位。

----结束

须知

禁止cluster复位，cluster复位后会断链，之后无法正常建链。

6.6 编程接口

6.6.1 系统初始化

上电复位后整芯片工作在Slow模式（晶振时钟），CPU可以从默认启动地址取系统初始化代码完成整系统的初始化配置，下面列举了系统初始化流程以及需要初始化的模块：

1. CPU内部初始化。
2. 根据产品地址映射规划，可以使用默认值DAW配置，也可以改配DAW进行地址空间划分，细节请参见“**地址控制**”章节。
3. 初始化UART。
4. 初始化SRAM，细节请参见“**SRAM模块**”章节。
5. 如果PLL默认配置不满足要求，则配置PLL并等待其锁定。
6. 初始化DDRC。

   说明

   Hi309a DDRC集成有MCU，如果不影响启动时间，可以将DDR PHY的初始化程序放到DDR_SUBSYS 内部集成的MCU核执行。
7. 初始化必要的系统配置（不分先后），主要是时钟和复位以及IO复用关系的配置。
   a. IO 复用关系，细节请参照IOConfig的描述。

版权所有 © 海思技术有限公司

b. Sysctrl必要配置，打开需要模块的时钟，并撤离其复位以及Event等其他控制，细节请参照“SysCtrl”章节。
c. SubCtrl必要配置，打开需要模块的时钟，并撤离其复位以及其他控制。

8. 初始化总线通路模块。
   a. 初始化GIC，细节请参见“GIC”。
   b. 初始化SMMU。

须知

SMMU为Bypass， 需要参照SMMU章节的User域Bypass模式属性转换配置。

   c. 初始化HHA/MN。
   d. 初始化Scheduler。

须知

通过配置scheduler各个端口的最大otd个数来实现schedule的口2~口7的最大otd不能超过16（保序的约束）。

   e. 初始化Dispatch。
9. 将引导程序拷贝到DDR中，初始化UBOOT划分的各个数据段，然后跳转到DDR继续运行。
10. 初始化SFC或者NAND FLASH。
11. 初始化功耗管理。
12. 根据需要，初始化如下全部或部分模块（个别模块顺序可以按需求调整）。
    a. 初始化TIMER。
    b. 初始化WDOG。
    c. 初始化Tsensor。
    d. 初始化SEM。
    e. 初始化TRNG。
    f. 初始化L2 CoreSight。
    g. 初始化DMAC。
    h. 初始化BMU。
    i. 初始化SEC。
    j. 初始化HPRE。
    k. 初始化Tring。
    l. 初始化PPE。
    m. 初始化TM。
    n. 初始化POE。
    o. 初始化SPPE。
    p. 初始化RoCEE。
    q. 初始化MAILBOX。

版权所有 © 海思技术有限公司

r. 初始化RTC。

s. 需要将所有和POE交互的模块初始化完后，最后初始化POE，因为POE初始化后就会发消息到这些模块，如果发到还没初始化的模块，会导致该模块异常。

13. 根据需要，初始化其他外部接口。

a. 初始化MDIO。

b. 初始化GE。

c. 初始化XGE。

d. 初始化XSEC。

e. 初始化USB。

f. 初始化PCIe。

g. 初始化SA。

h. 初始化TDM。

i. 初始化CodecTDM。

j. 初始化SATA。

k. 初始化MII/RMII。

l. 初始化RGMII。

m. 初始化ESPI。

n. 初始化HAC_SPI。

o. 初始化SPI。

p. 初始化I2C。

q. 初始化GPIO。

14. CPU根据业务需要，使能外部接口，开始正常业务处理。

完成系统初始化后，Hi309a就可以进行正常的业务处理了，用户必须先使能接口的发送，再使能接收。先使能内部模块，再使能外部模块。下面会对初始化涉及多个地方的模块再增补进行描述。

6.7 注意事项

6.7.1 Cache 一致性配置注意事项

Hi309a芯片支持多核Cache一致性，如果要保证Cache一致性，需要配置正确的页表属性，并根据需要配置相关寄存器。下面的步骤是必须要做的配置项，配置的顺序请参照各启动流程，软件人员需要确认下面的注意事项是否都配置，来保证一致性。

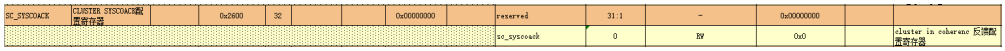
6.7.2 配置注意事项

步骤1 根据页表所需的粒度，配置MMU和SMMU的属性，保证同一段物理地址的页表具有相同属性。页表详细配置请参照ARM CPU的使用手册和SMMU的配置说明。需要注意，对于Shareble属性的地址，系统保证一致性；对于non-shareble属性的地址，系统不保证一致性，软件不能臆测对non-shareble地址操作的先后关系。

步骤2 根据使用的Cluster数目和加速器，使能对应的MMU和SMMU。系统需要在打开MMU的cache之前配置CPU_CLUSTER Subctrl的寄存器sc_syscoack为1，否则会挂死。

 版权所有 © 海思技术有限公司

图 6-8 寄存器图



SC_SYSCOACK	CLUSTER SYSCOACK配置寄存器		0x2600	32			0x00000000		reserved	31:1	-	0x00000000		
									sc_syscoack	0	RW	0x0		cluster in coherenc 反馈配置寄存器

步骤3 使能CPU Cluster，并使能对应的L1和L2Cache。

步骤4 根据使能的Cluster个数，使能对应的Fabric的FAB_SF_MODE寄存器的cluster_en域（该寄存器的bit[2:0]），每一个bit对应1个Cluster（例如使能了Cluster0、1，则需要配置FAB_SF_MODE寄存器的bit[1:0]为0x3）。

步骤5 LLC上电自动使能，不需要软件配置。

----结束

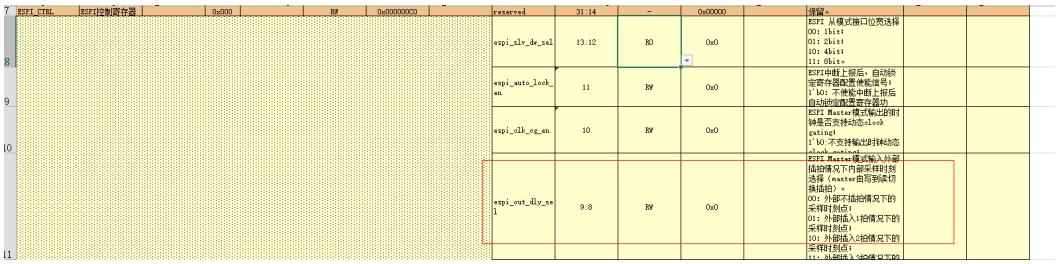
6.7.3 软件约束

1. 针对Non-Cacheable属性地址，LDADD等原子操作指令不可用，请软件规避,若使用会导致Data Abort异常；
2. 由PRFM指令引起的Stash类操作，上ring前会被转换成Cleanshared操作，CPU也不会收到任何的snpstash操作；
3. IFC模块放在HISS系统外，AP侧和HISS侧都能访问及配置，当AP侧和HISS侧同时对IFC模块发起配置时，无法确定以谁的配置为准，需要上层软件做强保护；IFC配置属性，包括IFC各种静态属性，需要做到HISS或AP只配置一次。

6.7.4 BSBC 软件需求

使用ESPI接口进行主从片通信时，当前ESPI调相打拍功能默认开启，需配置主片ESPI寄存器ESPI_CTRL的espi_out_dly_sel域段值为2，否则ESPI接口通信异常。

图 6-9 espi_out_dly_sel 域段配置参数



7	ESPI_CTRL	ESPI控制器寄存器	0x000		RW	0x00000000		reserved	31:14	-	0x00000		保留。	
8								espi_slv_dw_sel	13:12	RO	0x0		ESPI 从模式接口位宽选择 00：1bit 01：2bit 10：4bit 11：8bit。	
9								espi_auto_lock_en	11	RW	0x0		ESPI中断上报后，自动锁定寄存器配置使能信号： 1'b0：不使能中断上报后自动的锁配置寄存器功	
10								espi_clk_cg_en	10	RW	0x0		ESPI Master模式输出的时钟是否支持动态clock gating! 1'b0：不支持输出时钟动态	
11								espi_out_dly_sel	9:8	RW	0x0		ESPI Master模式输入外部插拍情况下内部采样时刻选择（master由写到读切换插拍）： 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；	

版权所有 © 海思技术有限公司

7 地址控制

7.1 概述

采用DAW（Dynamic Address Window）方式对地址空间进行映射。该功能可以将各个Master或CPU对任意设备或内存空间的访问，重新映射到新的设备或内存空间，并且可以选择地址映射的窗口大小。系统内部使用的总线寻址模式为36bits，可寻址的地址空间为64GBytes。

须知

- 在CPU Cluster的MMU中，支持2级页表转换在外设的SMMU中，支持2种模式的页表转换：其中操作系统使用的是VA（Virtual Address），VA通过一级页表翻译为临时的IPA（Intermediate Physical Address），然后IPA通过二级页表翻译为最终的PA（Physical Address）。这里的存储映射是指最终的PA地址。
- 与CPU Cluster相同的转换过程，即VA至IPA，再到PA的过程。
- 仅支持1级页表转换，其中处理的是IPA（Intermediate Physical Address）对PA（Physical Address）的转换，因此在该模式下CPU配置外设寄存器时，需要使用IPA进行配置，外设发出的IPA再由SMMU转换为PA。
- CPU MMU翻译后的PA地址与外设 SMMU翻译后的PA地址应该是一一对应完全相同的，即为本章节描述的存储地址。

7.2 特性

通过DAW的地址映射包含以下特性：

- 各个设备的地址空间可配置。
  - 各个设备的起始地址可配置。
  - 各个设备的地址空间大小可配置。
- 采用3级地址映射方式，一级二级为DAW，三级为固定译码。
  - 支持16个1级滑动窗口配置，每个窗口控制的空间为16MB～8TB，在各个Subsys的AA进行配置。

版权所有 © 海思技术有限公司

  - 每个1级滑动窗口的起始地址寄存器可配置，且起始地址必须与空间大小对齐。
  - 针对每个1级地址映射空间，支持8个2级滑动窗口配置，每个窗口控制的空间为1MB～1TB，在各个Subsys中的DISPATCH进行配置。
  - 每个2级地址映射空间的起始地址可配置，且起始地址必须与空间大小对齐。
  - 针对部分2级地址映射空间（APB接口以及AHB接口），支持3级译码，译码地址固定不可变。
  - 对于3级地址映射空间，每个映射空间的大小如果没有特殊需求，则默认为64k，对应的设备固定。
- 初始状态下，默认DDR/SRAM/外设集中在0GB～3GB范围内分布。
- 启动后，主核可根据应用设定各个TgtID在系统中实际地址分布。
- 对于单VF空间为64kByte的HAC模块，其VF空间统一规划，由独立的逻辑进行地址映射：
  - 默认全部集中在4GByte最高的16MB空间内，一共5组VF，每组占2MByte空间。
  - 每组VF支持32个64kByte大小的HAC空间。
  - 5组VF地址连续不可改变，每组VF对应的HAC类型可配。
  - 整体16MB空间的基地址可配置。
  - 访问16MB的VF空间，但无实际对应HAC的地址由总线返回DECERR，并上报中断，请参见“**注意事项**”。
- 如果访问DAW没有覆盖的地址空间，总线返回DECERR，并上报中断，请参见“**注意事项**”。
- 如果DAW配置有同一地址段映射到不同设备，将上报中断，且记录出错的DAW序号，供调试修正。

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 Memory Map 关键术语描述

术语	描述
DAW	Dynamic Address Window，动态地址窗口，该窗口可灵活定义某个TgtID设备的实际物理地址存储空间。
TgtID	Target ID。目的ID，即说明操作需要到达Fabric中的Ring的目的结点ID。 TGTID=[SKTID,DIEID,PORTID]
SktID	芯片ID，表示单板上物理芯片的编号，当前仅支持单片系统。
DieID	晶元ID，表示一个物理芯片（Socket）内部集成的各晶元（DIE）的编号。2bit指示。
PortID	端口ID，表示一个晶元（DIE）内的总线接口编号，每个接口对应一个Subsys/Memory/CPU Cluster。

版权所有 © 海思技术有限公司

术语	描述
Remap	地址重映射，CPU复位时，默认从0x00_0000_0000地址取第一条指令，而该地址通常是DDR空间，为保证CPU能够正确从指定的外部存储空间取指，可以修改CPU的默认值，使CPU上电后从指定地址取指。
HAC	Hisilicon Accelerate Controller，自研硬件加速器模块。
PF	Physical Function，物理功能。在单模系统即指整芯片的各个功能；在多模系统则对应公共资源以及仍为单模特性的功能。
VF	Virtual Function，虚拟功能。在多模系统指各中模式独享的私有资源。

7.4 默认地址分布

系统上电后的默认地址分布即可满足应用需求，如需调整，则通过修改DAW和多模空间基地址完成。

7.5 DAW

7.5.1 DAW 介绍

系统的地址映射采用3级译码方式进行映射，每一级的映射空间粒度依次由粗到细对整个地址空间进行划分。DAW根据接收到的地址，将传输发送到特定的设备上。

每个DAW为一个独立寄存器，配置信息包括基地址（即窗口起始地址）、窗口大小、TgtID或者Region信息。其中1级DAW寄存器分布在Access Agent中，译码过程也在Access Agent模块完成；而2级DAW寄存器则分布在各个Subsys里面的Subctrl当中，译码过程由Dispatch模块完成。

配置完成各个DAW寄存器后，需要开启DAW使能寄存器的使能位，对应的DAW配置才能生效。

DAW只是一个地址译码窗口，并不会做任何地址转换。1级DAW用来判定CPU或ACC、IO的传输请求地址，如果在窗口中命中，则将该传输引导到对应的TgtID Subsys上。CPU的MMU或外设的SMMU在配置页表时，都需要基于DAW定义的有效地址空间来设定。

7.5.1.1 一级 DAW

一级DAW采用32个滑动窗口，每个窗口的粒度为16MB～8TB，每个窗口通过两组配置寄存器配置。

1级DAW寄存器在系统控制器中配置。

版权所有 © 海思技术有限公司

须知

- 严禁将同样的地址映射到不同的TgtID。
- 严禁配置寄存器表单指定的TgtID以外的非法ID。
- 违反配置要求将导致系统挂死。

7.5.1.2 二级 DAW

对于1级DAW中配置为Subsys的空间，支持2级DAW窗口。每个Subsys的2级DAW窗口通过DISPATCH进行配置。

2级DAW寄存器在各个子系统的DISPATCH中配置（具体偏移地址另见各子系统的DIAPATCH nManager）。

须知

当Dispatch下挂的AXI模块需要复位时，为避免系统访问异常，需要将对应AXI口的2级DAW关闭，使复位过程对该模块的访问由Dispatch下挂的Default_slave处理，保证系统的正常运行。

7.5.1.3 三级译码

对于每个Subsys中都包含了APB Slaves以及AHB Slaves Region，这两个Region都包含了多类低速设备，因此需要第三次译码进行地址空间映射。三级译码为偏移译码，不可配置，每个设备的寄存器空间若无特殊需求，均为64kByte。

说明

整系统地址分配有多级的冗余空间，未分配的地址默认返回OKAY响应，可通过配置寄存器在Debug模式下返回DECERR响应，方便定位问题。

- DAW1未分配的空间，统一由MN处理，对于CPU访问默认返回OKAY，加速器访问返回DECERR（具体查看**MN**章节）。
- DAW2未分配的空间，有各个Subsys的Def_Slv处理，默认返回OKAY，通过配置可以选择返回DECERR。
- DEC3未分配的空间，由APB/AHB的Decoder负责默认返回OKAY，通过配置可以选择返回DECERR。
- 各个Slave内部未使用的空间，Slave负责默认返回OKAY，通过配置可以选择返回DECERR。

7.5.1.3.1 DAW 和设备实际空间/有效空间映射关系

DAW允许定义的窗口小于设备的有效空间，此时地址窗口为设备有效窗口基于配置地址的偏移地址窗口。

如果设备的实际空间小于设备的有效空间，或者DAW定义的空间大于设备有效空间，访问模式均为卷绕。

7.5.2 DAW 配置方式

DAW配置一般在Boot前期完成，且配置完成后一般不作修改。为了减少芯片的物理实现资源，一级DAW采用DJTAG方式访问，速度较慢（μs级别），所以配置完成后建议

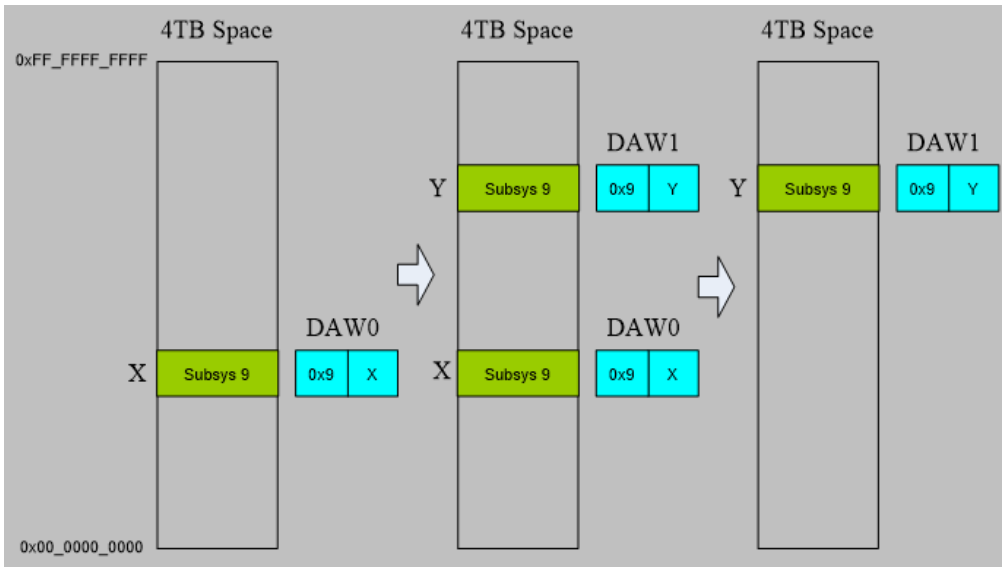
版权所有 © 海思技术有限公司

读取最后配置的一个窗口，确定配置完成再执行后续操作。二级DAW采用APB接口访问，速度较快（100ns级别），配置完成后需要再配置同步寄存器，才能使新的DAW值生效。

一级DAW要求全局统一，各个SUBSYS的Access Agent的DAW配置必须统一，Boot过程需要对所有的DAW窗口进行刷新。若确定某些窗口未使用或后续使用，可以在使用前再更新。

对于未使用的窗口改配置，可以直接修改，无需流程保证。对于已经在使用的窗口，特别是Subsys 的DAW配置更新必须以TgtID重复映射的方式进行，即Subsys 初始地址为X，要修改为地址Y，需要先选择一个空DAW寄存器，先将Subsys 的TgtID和地址Y写入空DAW寄存器并使能，此时Subsys 同时具有两个地址映射，访问地址X和访问地址Y均可访问设备A，此时将对设备A的X地址的访问迁移到地址Y，然后再关闭Subsys在地址X上的DAW寄存器，如图7-1所示。

图 7-1 DAW 窗口更新示例



7.6 多模隔离

7.6.1 多模空间介绍

集成的外部加速器（HAC）具有多模特性，即模块内部资源被多种模式共享使用，需要对这些资源进行合理划分和隔离，对于地址控制则需要对送往HAC的地址进行特殊处理。

每个HAC内部分为PF地址空间和VF地址空间，PF空间大小不一，属于多模共享的公共地址空间。每个HAC一般配置8段VF空间，且每段VF空间对于大部分的HAC均统一为64kByte大小。

在运行操作系统或线上业务的过程中，通过减少页表的替换可以提升硬件系统的处理性能。但页表条目的缓存空间（TLB）有限，所以需要尽可能将不同HAC的同一VF集中到一条页表项内。

当前最常使用的页表项粒度为2MByte，所以大于2MB的VF空间单独处理，小于2MB的VF空间（统一要求限制在64kByte以内），则将多组合并到2MB页表内。

版权所有 © 海思技术有限公司

原有的DAW配置完成所有PF的地址空间映射，以及VF空间大于2MB的地址映射。另外，对于小于2MB的VF空间整系统统一规划，并集中到统一的16MB多模空间内，该组多模空间不在DAW中映射，由新增逻辑独立映射。多模空间内部分布固定不可配置，多模空间16MB整体基地址可配置。

须知

VF专用的16MB空间同属于全局物理空间，即该16MB空间配置不能和已有DAW配置重合，否则译码出错，系统将出现不可预知的错误。

7.6.2 多模空间配置方式

多个HAC的多模空间集中到统一的一段物理地址空间内，减少页表隔离时所使用的页表条目数，提升页表命中率，从而提升性能。

多模空间也属于整系统物理空间的一部分，通过实际的物理地址访问，所以多模空间的配置将占用实际的物理空间，并且不能和已有的被分配的物理空间重叠。

单个HAC群占用2MB的物理地址空间，一共支持5个AC群，即总共占用16MB物理地址空间。总HAC群的基地址通过寄存器配置，默认为低4GB空间内的最高16MB空间，即基地址默认为0x000_FF00_0000。每个HAC群之间的地址连续排列不可改变，单个HAC群内部具体64kByte映射到具体HAC可配。

表 7-2 VF 空间地址配置说明

Addr[35:24]	Addr[23:21]	Addr[20:16]
HAC群基地址	HAC群VF号	单VF内HAC索引号
可配	不可配	可配

默认的HAC索引号如表7-2所示。

表 7-3 HAC 索引号

HAC Offset [20:16]	64K HAC reg
0x1F	保留
……	保留
0x13	MAILBOX
0x12	POE_ACC
0x11	保留
0x10	ROCEE
0x0F	保留
0x0E	保留

版权所有 © 海思技术有限公司

HAC Offset [20:16]	64K HAC reg
0x0D	保留
0x0C	保留
0x0B	TM_CPU
0x0A	TM_CPU
0x09	TM_CPU
0x08	TM_CPU
0x07	保留
0x06	TRING
0x05	SEC
0x04	保留
0x03	保留
0x02	保留
0x01	TM_ACC
0x00	BMU

系统最多支持32个HAC的VF空间配置，需要配置Access Agent和Dispatch（Dispatch寄存器集成在Subctrl中）的相关寄存器，才可以正确访问各VF空间。配置VF_BASEADDR寄存器修改HAC群的基地址，配置VFXX_ID寄存器则修改对应索引号将指向的HAC，配置VF_ENABLE则使能对应索引号的HAC空间。访问未使能的VF空间则返回DECERR。

说明

在Access Agent的VF_ID配置，仅为HAC所在的Subsys的ID；在Dispatch中有类似的配置寄存器，表示HAC所挂载的Dispatch Port编号。只有两级译码过程均配置正确，访问才能到达最终HAC的VF空间。

须知

对于无需使用VF空间的应用场景，可以将VF的16MB全部映射到PF空间，所以需要将所有的VF enable禁用，以免软件误操作访问了VF空间。

版权所有 © 海思技术有限公司

8 系统性能

8.1 概述

本芯片（Hi309a）主要面向网络中低端主控。Hi309a在网络场景下，主要是应用在AR/FW/WLAN AC等企业网关、园区线卡和盒式主控控制面应用，满足芯片安全可信基线2.0。

Hi309a支持8核ARM Cortex-A55 CPU，支持2.2GHz频率。支持1通道DDR4；支持10 Lanes Serdes， PCIE采用Gen3；支持最大26个通道以太网口，4*XGE/6*QSGMII内置MAC；支持2*SATA；支持2*USB2；内置集成44G LSW，基本L2特性交换；集成15G SEC引擎；HPRE引擎，支持RSA 4K OPS，ECDH+ECDSA算法套、ECDHE+RSA算法套；集成6G SA引擎。集成TM，包保序（POE），缓存管理（BMU）等通用加速引擎。在安全方面，支持安全特性， 支持macsec，支持DICE，网络安全。

8.2 关键术语

表 8-1 系统性能关键术语描述

术语	描述
POE	Packet Order Enforcer，包保序模块。
SEC	Security Engine，安全引擎模块。
FW	Fire Wall，防火墙。
线性度	n核总性能/（单核性能*n）。

8.3 特性

系统性能相关的特性：

- 接口流量性能需求：
  - 支持PPE最大1518Bytes @ 20Gbps收发规格。

版权所有 © 海思技术有限公司

  - 02MAC LSW支持44Gbps 64B线速；
  - AR转发性能（非自然散热） 10Gbps@330B/4.3Mpps@64B
  - SDWAN转发性能：
    - 自然散热 SDWAN性能1Gbps@330B。
    - 非自然散热 SDWAN性能1.5Gbps@330B。
  - 防火墙（自然散热）传统转发性能 10Gbps@330B/9Mpps@64B， NG墙SA +IPS（8*1.6G）： 1.3G @ EnterpriseMix。
  - AC转发性能（自然散热）：
    - 基础转发场景：10Gbps @ 512B。
    - NGE业务场景：6Gbps @ 512B。
- PPE性能需求：
  - 单TNL单BD性能：单个网口一个报文仅有一个BD的场景：PPE支持以3.8Mpps/10Gbps进行报文收发，即长度330Bytes以上的报文性能达到10Gbps，长度330Bytes以下的报文性能达到3.8Mpps。
  - 单TNL多BD性能：多BD报文性能：是对应长度的单BD报文性能的至少80%。对接收侧而言，此时要求可用的Buffer长度（除去skip）都是1.5kByte以上。对发送侧而言，除尾片外，其他片典型包长1536B。
  - 多TNL单BD性能：所有通道同时工作时，所有通道支持的同时收发流量是20Gbps，不超过总流量时，任意通道满足带宽。
  - 转发性能： MAC侧之间转发长度为64Bytes及以上的报文支持44Gbps线速收发；CPU侧之间转发长度为330Bytes及以上的报文，且一个报文仅有一个BD的支持20Gbps线速收发。
  - 延迟：包长，500Byte，309a（Tx）->309a（Rx）serdes口环回端到端时延：
    - Latency MIN （空载）： 1.5us。
    - Latency MAX（DMA引擎为空，但内部FIFO数据Buffer满）： 3us。
- DDR接口：
  - DDR接口对于连续地址访问流水。
  - DDR/PHY training的时间保底1S，挑战0.5S。
- SPI FLASH接口：
  - SFC IP读写速率不低于SD7601。SD7601 NOR单软测试数据如下：
    - 单线模式：擦除4.51MByte/s、读取12.93MByte/s、写入2.87MByte/s。
    - 四线模式：擦除6.17MByte/s、读取35.25MByte/s、写入3.15MByte/s。
- ESPI接口性能：
  - 8bit数据接口@50Mhz速率。
  - single写速率：>10MB/S， burst写速率：>20MB/S。
  - single读速率：>5MB/S， burst读速率：>10MB/S。
- CPU片内BOOTROM启动时间要求：
  - CPU片内BOOTROM启动时间（CPU解复位到执行外部BOOTROM程序的第一条指令）要求小于1秒（以NOR/NAND安全启动，Bin为2MB计算）。

版权所有 © 海思技术有限公司

- Nand接口性能需求：
  - 异步nand器件，单片，使用正常操作，不使用提高性能操作时，在裸测情况下随机写性能不低于5MByte/s，读性能不低于16MByte/s。
  - 异步nand器件，以器件参数tPROG=300us，tR=30us为评估标准，读写性能如下：
    - 使用cache连续读操作，性能不低于24MByte/s。
    - cache 连续写操作，性能不低于8MByte/s。
    - cache过程进行interleave，时写性能不低于11MByte/s。
  - nand器件写busy时间不作为瓶颈的情况下，cache和interleave同时使用的情况下，写性能达到22MByte/s。
- POE性能需求：
  - POE在700MHz频率下提供的最大加包性能是：80Mpps（128bits任务描述符），50Mpps（256bits任务描述符）。
  - POE提供的最大取包性能不低于：60Mpps（非保序任务），15Mpps（保序任务）。
  - POE提供的推包性能不低于：15Mpps（按照一次推包访问时延200ns计算）。
- BMU性能：
  - BMU缓存的申请释放带宽需求均为不小于15Mpps，可以支持CPU和HAC的并发访问。
- TRNG性能：
  - TRNG和SEC的硬件通道，混合随机数性能达到1.6Gbps的吞吐率；数据位宽为16bits，性能达到15Mpps。
  - 软件访问TRNG的接口，混合随机数性能达到640Mbps的吞吐率。
- HPRE性能：
  - ECDHE+ECDSA算法套：（代理场景）4Kops。
  - ECDHE+RSA算法套：（代理场景）4Kops。

8.4 功能描述

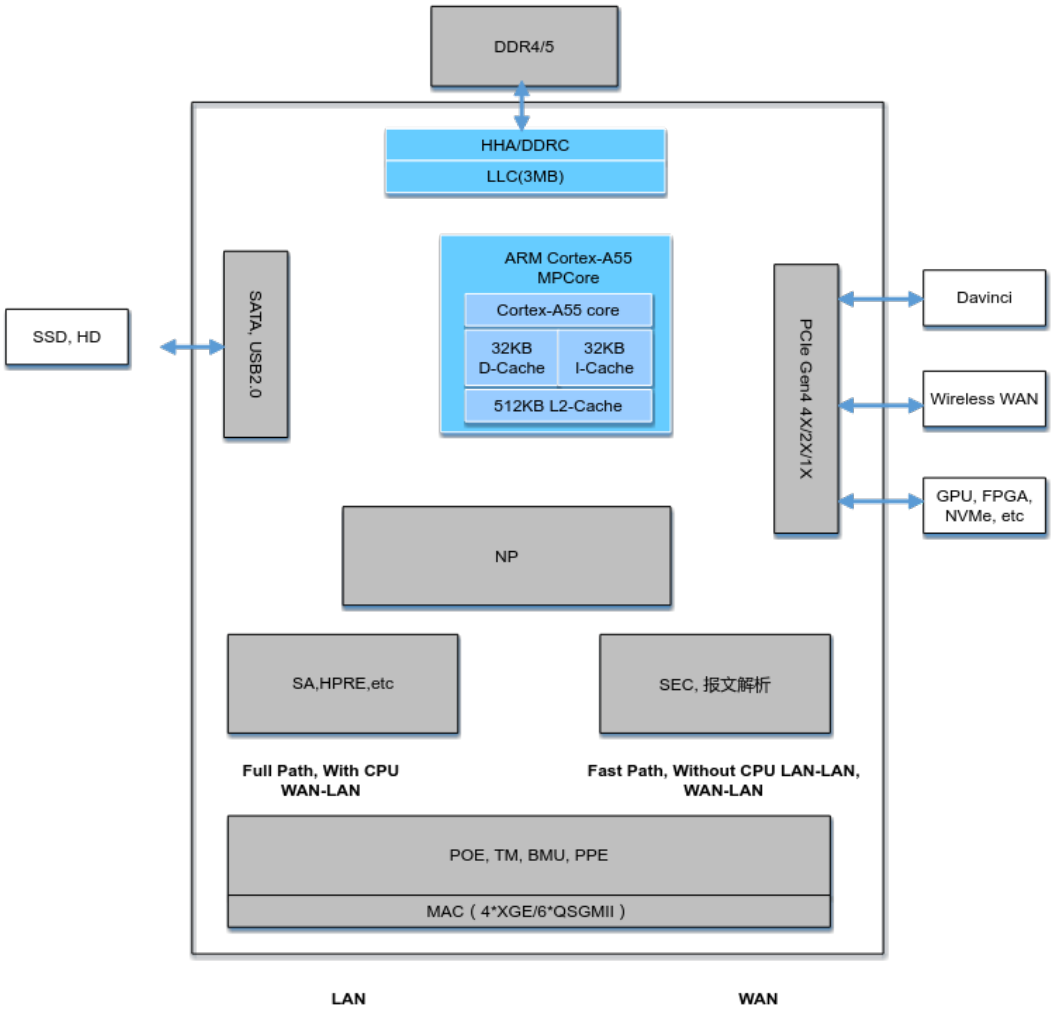
本节功能描述包括如下内容：

1. 性能规格：业务场景性能规格、系统性能规格和单模块性能规格
2. 性能调优方法（芯片硬件）
3. 性能DFX：性能监测，性能调试

版权所有 © 海思技术有限公司

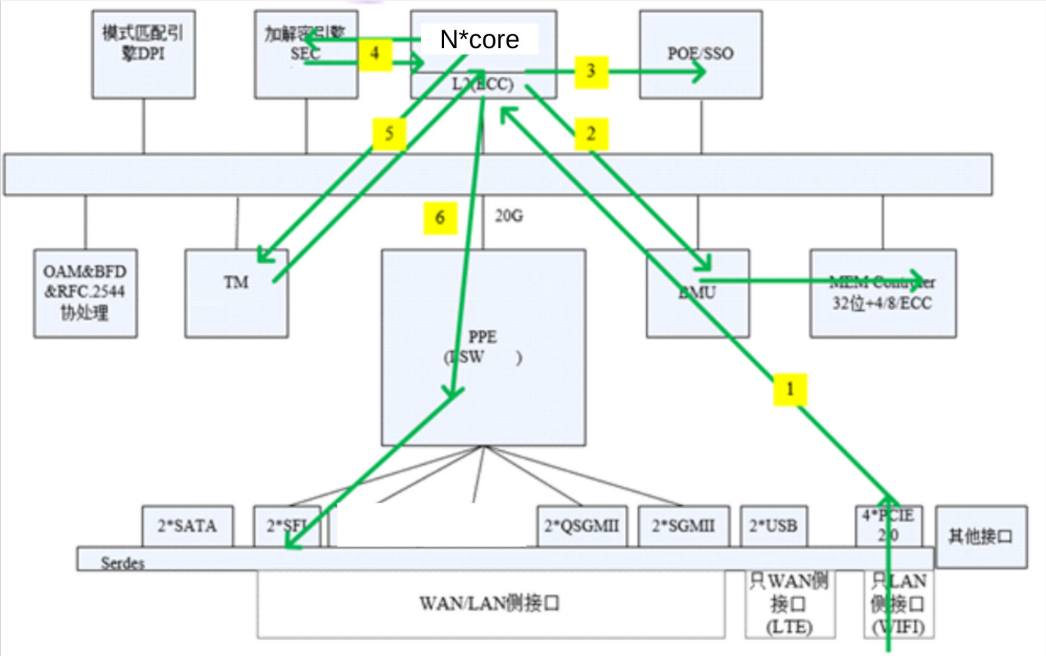
8.4.1 应用框图

图 8-1 芯片应用总体示意图



版权所有 © 海思技术有限公司

图 8-2 AR 设备典型应用场景



8.4.2 相关模块

在芯片应用场景中，如下模块会涉及到系统性能，分别描述如下：

- Cortex-A55：完成业务软件通用处理。
- MMU：根据软件配置在DDR内部的页表，完成Core存储访问的属性检查、地址映射；MMU内部含有两级TLB用于缓存部分页表。
- Ring：完成系统所有组件的数据传输。
- HHA：HHA中包含一个特殊的Ring节点，用于处理Barrier和DVM操作；HHA还负责对DDR命令进行调度。
- DDRC：负责将系统总线命令转换成对片外DDR Memory的访问命令。
- Scheduler：将若干路总线Master的操作汇聚成一路。
- SMMU：根据软件配置在DDR内部的页表，完成加速器及IO存储访问的属性检查、地址映射；SMMU内部含有两级TLB用于缓存部分页表；SMMU支持对需要总线保序的Memory访问命令的返回传输进行保序，支持对需要总线保序的Device访问的命令传输保序。
- Dispatch：将总线的访问操作根据地址分配信息分发给各个加速器、IO。
- FTE：支持对DDR/L3、部分加速器发送测试命令，用于性能测试和评估。
- POE：系统中进行队列缓存、队列调度的模块，支持对CPU处理任务进行保序。
- BMU：管理系统中用于IO接口模块收发报文的缓存；支持CPU、加速器、IO申请和释放缓存。
- PPE（MAC）：支持网口报文收发以及硬件解析。
- NP：可编程包解析模块，内部集成NP及运行所需的指令、数据等存储器和查表加速器。
- TM：支持网口发送报文的整形和流控。

版权所有 © 海思技术有限公司

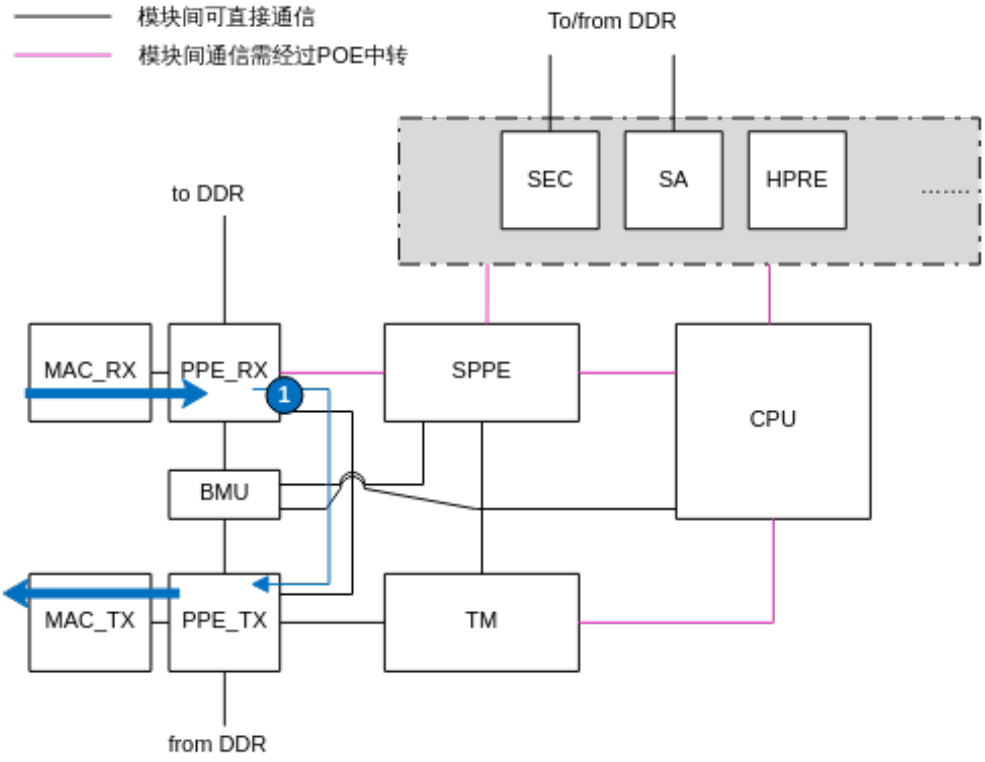
- SEC：支持数据的加密、解密、认证和校验。
- SA：进行模式匹配的IP模块。
- USB 3.0：支持OTG。
- PCIe ： 扩展网口、AI智能芯片、FPGA、存储等。
- NANDC/SFC：外接FLASH等存储器件。

8.4.3 芯片数据流图

Hi309a主要的数据流图如图8-3所示。

系统以CPU Core和NP为处理核心，从网口接收来自片外的报文，在硬件加速器的协助下完成各种处理，最后将报文通过网口发送出去。

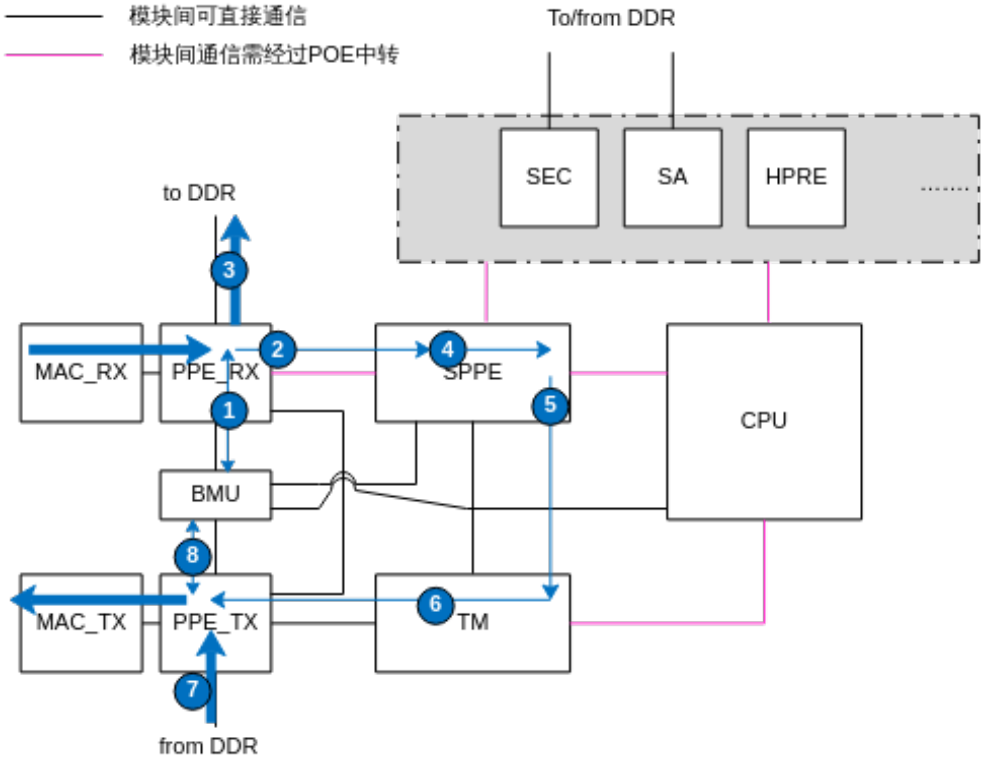
图 8-3 LSW 快速转发业务芯片数据流图



步骤1 接收到报文后，PPE对报文进行解析，对LSW快速转发业务，PPE直接完成包头编辑然后发送出去。

版权所有 © 海思技术有限公司

图 8-4 基础网络转发业务处理流程



步骤2 PPE预先向BMU申请好内存Buffer缓存。

步骤3 接收到报文后，PPE对报文进行解析，并生成包描述符。

步骤4 PPE将解析结果随报文一起缓存到DDR。同时将包描述符送给POE。

步骤5 SPPE完成报文合法性检查，提取包头做各种转发查找处理。

步骤6 SPPE生成报文的编辑命令，和包描述符一起交TM处理。

步骤7 TM完成调度、整形后将报文的包描述符推送给PPE。

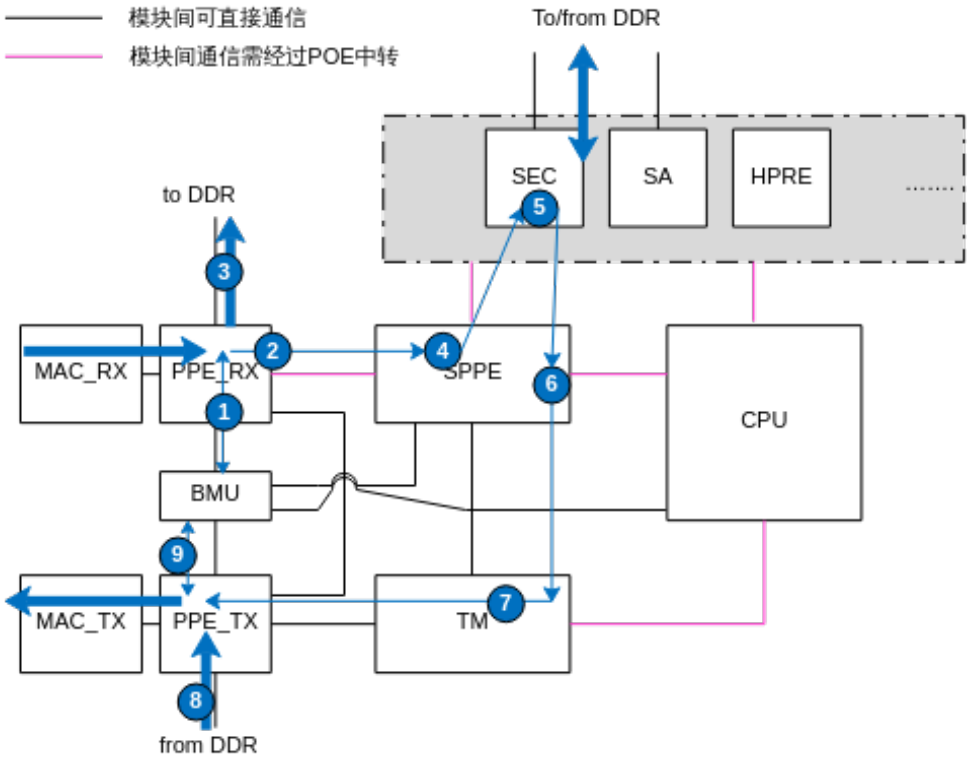
步骤8 PPE从DDR中读出包数据，完成包头编辑后发送出去。

步骤9 PPE向BMU释放缓存。

----结束

版权所有 © 海思技术有限公司

图 8-5 快速加解密转发业务处理流程



步骤1 PPE预先向BMU申请好内存Buffer缓存。

步骤2 接收到报文后，PPE对报文进行解析，并生成包描述符。

步骤3 PPE将解析结果随报文一起缓存到DDR。同时将包描述符送给POE。

步骤4 SPPE完成报文合法性检查，提取包头做各种转发查找处理。

步骤5 SPPE完成IPSec前处理后，将报文的包描述符推送给SEC加速器进行加密/解密处理；SEC从DDR中读取数据，完成加/解密后，将包描述符推送回SPPE处理。

步骤6 SPPE 完成 IPSEC 后处理，包括各种必要的合法性检查和转发查找处理，然后生成编辑命令，和包描述符一起交TM处理。

步骤7 TM完成调度、整形后将报文的包描述符推送给PPE。

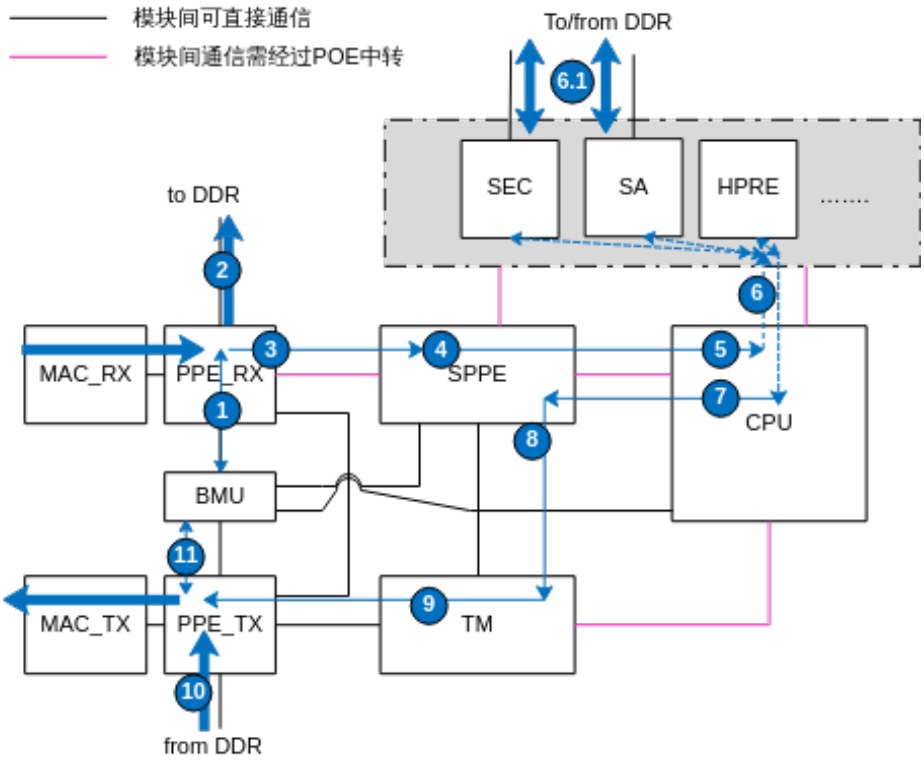
步骤8 PPE从DDR中读出包数据，完成包头编辑后发送出去。

步骤9 PPE向BMU释放缓存。

----结束

版权所有 © 海思技术有限公司

图 8-6 增强转发业务处理流程



步骤1 PPE预先向BMU申请好内存Buffer缓存。

步骤2 接收到报文后，PPE对报文进行解析，并生成包描述符。

步骤3 PPE将解析结果随报文一起缓存到DDR。同时将包描述符送给POE。

步骤4 SPPE完成报文合法性检查，提取包头做各种转发查找处理。

步骤5 CPU收到报文后处理。

步骤6 CPU按需要，可以调用加速器协助处理，可以多次调用：

- SEC：对称加解密。
- SA：业务感知。
- HPRE：非对称加解密。

步骤7 CPU完成处理后，将包描述符送给SPPE。

步骤8 SPPE 完成必要的转发查找，生成报文的编辑命令，和包描述符一起交TM处理。

步骤9 TM完成调度、整形后将报文的包描述符推送给PPE。

步骤10 PPE从DDR中读出包数据，完成包头编辑后发送出去。

步骤11 PPE向BMU释放缓存。

----结束

版权所有 © 海思技术有限公司

9 系统反压和流量控制

9.1 概述

本节主要是对Hi309a的系统反压和流量控制的综述，以及对各模块反压、流控行为的约束。规范各个加速器间、CPU与加速器之间的反压的行为。对于异常访问（如地址错误）等的处理不在本节描述。Hi309a的系统反压和流量控制机制的目的是保证全系统在复杂的突发和共享资源冲突下，协调Hi309a各部件，保证SOC高效工作。

Hi309a的系统反压和流量控制机制的主要目的是：

- 在SOC处于正常工作负载时，避免出现局部的阻塞，引入性能瓶颈。
- 在SOC高负荷工作时，根据QoS的分级机制优先保证较高优先级任务的处理，避免出现系统性拥塞。
- 在系统发生短时拥塞的时候尽量保证业务不受影响，在发生系统性拥塞的时候保证资源不泄漏。
- 保证全局的QoS识别，总线、LLC、Scheduler、dispatch及集成队列管理模块的HAC支持基于QoS的调度，在拥塞的时候保证高QoS的带宽和延迟。
- 在SoC内负载将要达到处理极限时，能够反压接口Ingress流量，避免处理过载和性能跳水；系统能够及时响应接口Egress流量受到的反压。
- 保证加速器模块对输入进行合法性检查，保证模块间协同的流控反压机制，接口器件异常的保护处理。

在芯片内部，处理器、加速器及IOC（IO Controller，如PCIe的Controller ）对系统存储资源的访问可分为下述4个层次：

- 第一级：处理器，加速器/IOC。
- 第二级：一个cluster内对4个处理器的调度，若干个HAC/IOC挂在1个Scheduler/SMMU下。
- 第三级：系统总线（Ring Fabric）。
- 第四级：由LLC和DDRC构成的存储系统，Dispatch及下挂的XSlave。

要保证达到系统反压和流控的目的，需要上述各级都支持灵活的QoS配置、端到端的按QoS调度和按QoS反压。

系统主要的业务处理是对报文流的接收、处理和发送，报文进入芯片首先由PPE进行解析，识别出不同的业务报文流，然后存放在由POE管理的入口队列；报文在芯片内部

版权所有 © 海思技术有限公司

处理过程中，存储报文的Buffer由BMU管理，报文各个处理单元之间的交互队列由POE管理；最后，待发送报文存放在由TM管理的出口队列中。

为了保证系统内共享资源的高效实用和保证高QoS业务的处理性能，反压的机制：

- 总线Link层反压
- 总线Network层反压：Error、Retry、Credit
- 应用层：Data

硬连线ON/OFF：

流控的方法：在系统共享资源瓶颈点（主要是LLC和DDRC），采用Analysis-Identify-Monitor-Control方式进行资源管理。

须知

上面所说的存储系统不能简单的理解为主存，而应该理解为存储空间。主存、HAC的寄存器及内嵌RAM、IO空间都应被视为存储空间的一部分。

9.2 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 系统反压和流量控制关键术语描述

术语	描述
AA	ACCESS AGENT的简称，用作Subsys内部数据流的汇聚和分发，实现Subsys的聚合，共享Ring的一个Node。
AXID	总线的访问ID值，本文特指AWID、ARID。
AA_SUBSYS	用于各个Subsys与Ring对接的Access Agent，实现Subsys内部Master和Slave与Ring之间的分发汇聚，以及解决上下Ring的分频问题。
BMU	Buffer Management Unit。
Cross Station	Ring上的节点，根据路由信息对传输进行汇聚和分发。每个节点最多有3组接入口，每组接入口支持4对收发。
Dispatch	将Device Station接收到的操作转发至ACC/IO/Register从设备目的端，对于非法地址的操作则送至Default从设备，且返回DECERR信息。
DRR	Deficit Round-Robin，赤字轮询，一种调度算法。
FTE	Fabric Test Engine，总线性能冲击模块，集成在每个Subsys内，可以在不同数据通路上对总线进行性能冲击测试。传输属性可灵活配置，对读写操作分别进行统计。
LLC	Last Level cache。

版权所有 © 海思技术有限公司

术语	描述
Master	指发送命令并接收响应的主体，一个模块可以仅是Master或仅是Slave或同时具有Master和Slave。
MOESI	用Modified、Owned、Exclusive、Shared、Invalid的5种状态来表示Cache中数据共享状态，从而保证共享数据一致性的一种协议。对应ACE协议中的UD、SD、UC、SC、I。
PD	Packet Descriptor，包描述符，PD/包描述符也经常被称为包任务。
POE	包保序（Packet Order Enforcer）。
PPE	Packet Parser Engine。
PQ	Priority Queue，一般指严格优先级（SP）调度。
PQ+DRR	严格优先级队列（PQ）+赤字轮询（DRR）TM等模块支持PQ+DRR调度算法。
PQ+Timeout	LLC（L3 Cache）和DDRC支持PQ+Timeout调度。命令进入到Cmd Queue后在没有执行之前各个命令的Timeout计数器将会递减，递减为0后这个命令定义为Timeout命令，Timeout命令具有调度的最高优先级。
RED	Random Early Detection ，随机早期检测。
Ring Bus	Ring Bus通常指系统总线。
RR	Fabric的Ring Station，对相同方向的传输汇聚时，采用RR算法进行轮询调度，支持2x Virtual Channel。
Scheduler	ACC和IO的传输进入Ring Bus前的预汇聚模块，将功能类似、总带宽和Ring Bus Station匹配的一组ACC和IO进行调度汇聚，然后再通过SMMU接入Ring Bus。
Slave	接收命令并产生响应的主体，一个模块可以仅是Master或仅是Slave或同时具有Master和Slave。
SLLC	Skyros Link Layer Controller。
SMMU	系统存储管理单元。
WRR	Weighted Round-Robin加权轮询算法。
逻辑CPU	物理CPU Core虚拟出的逻辑CPU，一个物理CPU Core对应多个逻辑CPU。要注意的是对于CPU物理实体而言，并不存在类似多线程CPU那样的“逻辑CPU”。
假满	POE/TM模块有内部缓存来存储待处理的任务，如果内部缓存满，同时因为总线或主存访问拥塞等原因无法及时把待处理的任务写到主存空间，此时称为假满。
真满	POE/TM模块有内部缓存来存储待处理的任务，如果内部缓存满，HAC会把无法在内部缓存的任务缓存到主存中，可使用的主存空间是软件配置的。如果主存中对应的缓存也被存满，这时就没有空间存储待处理的任务了，这时就达到了真满状态。

版权所有 © 海思技术有限公司

9.3 功能特性

系统反压与流控具有以下特性：

- 支持QoS分级（最多支持16级），可以在系统的不同层次对不同QoS的访问、数据传输进行调度和管理。
  - 整系统的内部操作支持多调度，第一级调度在CPU cluster，或Scheduler/SMMU上，第二级调度在Fabric的Ring Station上，第三级在LLC/Dispatch，第四级调度在DDRC。
  - 加速器支持对不同类型的访问携带不同的QoS，支持不同的VF配置不同的QoS；加速器及Scheduler支持16个优先级（0~15，越大的值表示越高的优先级），可以在Scheduler上对Scheduler各个接口配置使用总线操作携带的QoS或Scheduler上该接口配置的QoS；Scheduler对各个接口间支持WRR调度（不支持根据加速器送上的QoS进行调度）；Scheduler支持通过限制各个端口的Outstanding来进行各端口流量限制。
  - 模块内部汇聚节点支持按照端口WRR，每个端口支持1种QoS流量。
  - Ring fabric支持Skyros协议的4个虚通道（实现上是4个物理通道），各虚通道的反压独立，虚通道间的调度方式为RR调度；支持逃生Path。
  - LL cache支持QoS[3:0]的传递，区分4个优先级的调度，命令处理按照QoS调度，替换算法支持QoS； DDRC支持4级QoS，支持PQ+Timeout调度策略。
  - DDRC等支持带超时的SP（Timeout SP）调度，当一个访问请求超时后标记为timeout的命令，timeout的命令优于非timeout的命令调度；多个timeout的命令按照优先级调度。
  - Dispatch支持4级Qos，入口队列按照QoS调度；低优先等级操作长时间调度失败后，会自动按照最高优先等级Qos进行重新调度。
- 支持CPU连续访问POE、BMU（TM在业务流程中采用POE Forward操作，不由CPU直接添加任务），不需要查询Valid状态决定访问是否成功。通过中断反馈是否失败。
  - CPU直接访问POE添加任务：该中断保留作为应急处理（如配置错误）手段，但要求正常操作下，不会产生该中断。
  - CPU通过Ring Buffer的方式向POE添加任务。
  - CPU直接访问POE获取任务。
  - CPU通过Ring Buffer（内存队列）的方式从POE获取任务。
  - CPU访问BMU申请Buffer。
  - CPU访问BMU释放Buffer。
  - CPU对HAC/IO/外设的访问，不允许总线返回Error，如有异常则支持对应的中断或异常上报。
- 支持加速器访问BMU时通过总线返回Error反压。
  - HAC之间Error反压统一为不可配的固定值。
- 支持配置BMU的16个pool*9个Qos双门限反压PPE。在反压有效期间，PPE丢弃需要在相应pool中存储的该QoS的报文。
- 支持配置PPE的门限反压TM的访问。
  - 各个PPE通道的反压独立配置。
  - 当PPE开始反压TM时，TM最多可以再向反压的以太网口发出不超过5个报文的描述符。

版权所有 © 海思技术有限公司

- 支持配置PPE接收FIFO的流控Threshold和解流控Threshold（双Threshold），通过流控帧反压对接芯片。
  - 支持超时解除反压（如果此时仍然未达到解流控门限，则可能导致丢包）。
  - 支持通过软件配置是否使能对对端的pause帧反压。
- TM支持反压功能。
  - TM Slave加满时，通过不返回Credit，使各Master的Credit耗尽反压添加动作。
  - 支持丢弃Buffer地址给BMU时AXI总线返回ERROR的反压重传机制；若BMU返回假满超过重传次数（假满重传次数由软件静态配置），TM内部丢弃任务并上报中断，记录任务信息。
  - 支持推送PD给PPE时，目的加速器对TM的硬反压机制；PPE反压包含推送接口的端口级反压，对于PPE反压，TM对应端口接收到其硬反压后停止发送PD，当反压撤销后，继续推送动作。
  - 支持POE对TM的硬连线反压，目的加速器对TM的6bit硬连线反压，TM将其反压映射到发送端口上或捕获fifo上。
  - 支持软件基于TRUNK节点动态配置软反压机制，当某个TRUNK被反压后，停止对应节点发送PD，当反压撤销后，继续推送动作。
  - 支持软件查询硬反压的实时状态和反压次数。
- 支持 PPE反压CPU功能（jam反压）：
  - PPE的jam反压按照OOO的计数产生。
  - 支持连续反压时间值可配，连续反压CPU Station超时后，强制结束反压。
  - 对CPU Station的反压有效时，系统会减小CPU的outstanding。
  - 该反压是否使能软件可以静态配置。
- POE支持入口反压：
  - POE为每个Group提供一组反压PPE任务添加的水线配置寄存器POE_GRP_BP_TH，当Group中的任务个数大于等于水线值时，此Group不再接收接口（PPE）的任务添加，丢弃PPE添加的任务，POE负责释放Buffer；大于等于Group水线时，只反压PPE的任务添加，不影响其他处理单元的任务添加。
  - POE缓存队列空间（内部缓存空间和DDR空间）满时，上报真满中断，任务丢弃，统计丢弃次数；若是加速器添加，POE通过总线B通道固定返回OK响应，POE负责释放Buffer；若是CPU添加， POE通过总线B通道固定返回OK响应。
  - POE出现假满状态时，反压加速器的添加操作，POE通过总线B通道固定返回OK响应，POE负责主动重传添加任务，若重传超过最大重传次数则丢弃并释放Buffer，并上报中断，统计丢弃计数。
  - POE任务队列使能关闭时，反压加速器的添加操作，POE通过总线B通道固定返回OK响应，任务丢弃，上报中断，统计丢弃计数。
  - POE出现假满状态时，反压CPU的添加操作，POE通过总线B通道返回OK，POE负责主动重传添加任务，若重传超过最大重传次数则丢弃并释放Buffer，并上报中断，统计丢弃计数。
  - POE任务队列使能关闭时，反压CPU的添加操作，POE通过总线B通道返回OK，任务丢弃，上报中断，统计丢弃计数。
  - 支持反压状态可查询，支持反压次数（8bit饱和计数、读清）统计可查询。
- POE支持出口反压：

版权所有 © 海思技术有限公司

  - 支持添加任务到POE/TM若发生真满时，由推送的目的加速器进行释放Buffer处理。
  - 支持释放Buffer到BMU时总线反馈假满，POE重试添加操作，重试次数到达配置次数时，POE上报中断，Buffer丢弃。
  - 支持反压状态可查询，支持反压次数统计（8bit饱和计数、读清）可查询。
- PPE支持流控功能（SSU完成）
  - 支持基于内部缓冲mem的使用状态进行流控解流控。
  - 支持基于BMU的Buffer的使用状态进行流控解流控。
  - 支持流控软件指示进行流控。
  - 支持超时功能，停止收到报文达到流控反压时间达到门限后退出流控，直到再次收到报文或流控状态解除后，才会重新触发流控。
  - 支持端口级流控。
- BMU等HAC支持区分不同模式的反压流控
  - BMU的假满支持区分不同VF/PF的反压，支持的VF数量参见各模块的FS。
  - 被反压的模块要能够区分VF响应反压，不同的VF之间没有有相互影响（考虑到假满是瞬时的，不强制要求在部分VF假满时，未收到假满的VF可以继续发送消息/PD）。

以下是系统反压和流量控制相关的DFX特性：

- 支持提供当前反压状态。
- 支持提供反压历史状态。
- 支持提供反压次数统计（饱和计数）。
- 支持每个物理队列达到满时，上报中断。
- 支持反压超时时上报中断。
- 支持CPU对POE的内部队列和其它FIFO的空满状态查询。

9.4 遵循协议

ARM CHIE（Skyros）协议；

AXI协议。

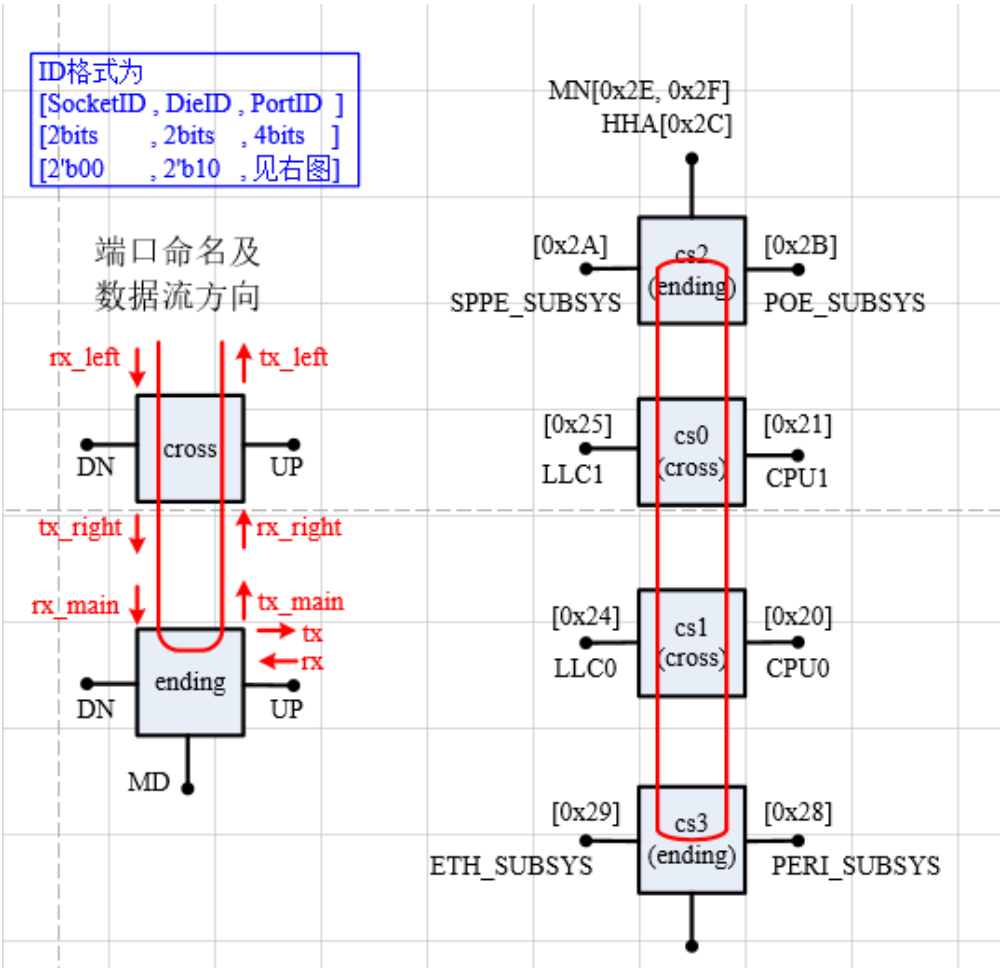
9.5 功能描述

9.5.1 应用框图

系统总线拓扑如图9-1所示。

版权所有 © 海思技术有限公司

图 9-1 Hi309a 的总线拓扑



如图9-1所示，Hi309a包含2个CPU cluster，LLC（Last level cache），DDRC，POE、thernet、PCIe等接口，PPE、TM、BMU等加速器

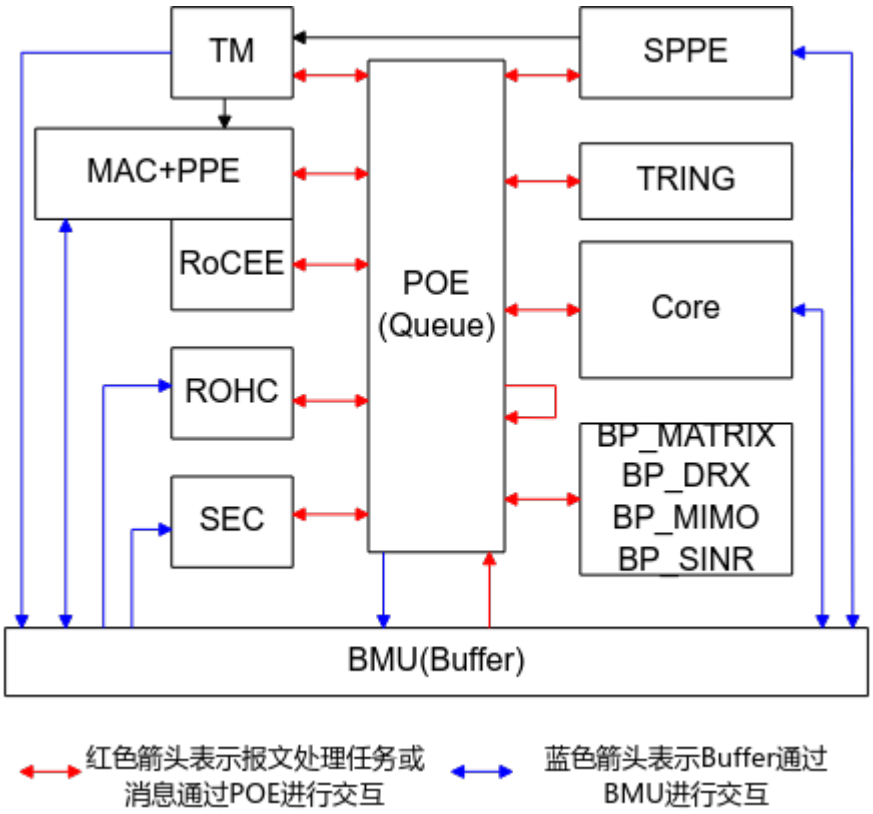
Ring属于传输系统的链路层（Link），通过Credit的方式进行数据传输。Ring节点为Bufferless结构，传输无阻塞。

当业务突发时，比如XGE、GE口同时突发接收报文时，主存可能无法同时支撑2个CPU cluster以及HAC/IO die需要的带宽。此时，PPE可以产生对CPU cluster的反压，以优先保证报文的收发。这一反压功能可关闭。

基于系统数据流和业务处理流程，各业务模块间的接口关系请参见图9-2。

版权所有 © 海思技术有限公司

图 9-2 业务模块接口架构图



BMU管理系统中的Buffer（一段连续内存），需要使用Buffer的模块通过与BMU交互完成Buffer的申请和释放，从上图9-2可以看出，系统中需要申请释放Buffer的模块主要是PPE、SEC、RoCEE、Core和NP（SPPE），其他模块不应该与BMU有交互接口。

POE负责管理Core之间、Core和HAC/IO之间、HAC/IO与HAC/IO之间的交互队列，起到一个On-Chip Message Bus的作用。

9.5.2 相关模块

在系统中，如下模块会涉及到系统反压和流量控制，分别描述如下：

- CPU Core：CPU本身无法直接感知业务反压，但可以被总线/AA反压，并可通过软件轮询、中断感知业务反压以及加速器的拥塞状态。
- LLC Cache：LLC是重要的共享资源，支持一定的拥塞管理和调度。
- Fabric：是各子系统间数据交互的主通路，通常Fabric的带宽应当有足够的“加速比”但后续需要分析Ringbus的各节点间的带宽是否会发生拥塞。
- SMMU：纯粹的SMMU和系统的QoS、反压流控关系相对小一些，SMMU承载着对AXI master发出操作的缓存能力，会反压Scheduler，并可能产生额外的针对需要总线保序的业务的反压。
- DDRC：同LLC一样，LLC是重要的共享资源，支持一定的拥塞管理和调度，Hi230的DDRC也是系统的主要性能瓶颈之一。
- PPE：响应网口的反压，并可以触发总线产生对CPU。在总线带宽不足的情况下会主动丢包，PPE的集成LAN Switch针对内外端口支持COS队列和调度。

版权所有 © 海思技术有限公司

- TM：系统的出口队列管理，支持层次化调度，可以响应PPE和POE的反压，并支持队列尾丢弃、RED丢弃。
- POE：在QoS方面起到入口队列的作用，作为入口队列模块，无法完全规避队头阻塞，可以反压TM，但对于PPE的反压因为存在“多（Group）对1（调度器）”，仅在部分场景可以使能。
- 其他HAC：可以响应Scheduler的反压，反压Dispatch。
- 网口：解析PAUSE帧，反压PPE。

9.5.3 数据结构

Hi309a的反压消息及任务通知均采用简化的消息格式。传送的方式是：消息的发送方写接收方的特定寄存器，不同的PF/VF有各自独立的寄存器。

当消息小于32位时，简化的消息支持32/64/128位操作。但128位只有低32位数据有效；当消息大于32位但不超过64位时，简化的消息支持64/128位操作。但128位只有低64位数据有效。

说明

1. 完整的反压消息计划携带源、目的、校验位等信息，共128位。头56位是消息的header（FCBP_MSG_HEAD），中间64位为消息的payload，末8位是校验位（CRC-8）。在Hi309a中，因为采用反压源通过AXI master接口写被反压方寄存器的方式、通过系统总线传递反压，因此消息的header、校验位均可以忽略。
2. FCBP_MSG_HEAD的格式如下：{16位目的地址，16位源地址，4位的RSV/版本号，4位的消息类型，4位的QoS，4位的RSV，8位的VFID}
3. 完整的消息格式如下：{FCBP_MSG_HEAD[55:0]，payload[63:0],CRC[7:0]}。

下面区分各反压消息时只介绍后64位的反压信息。

9.5.3.1 POE 对加速器的有任务通知

POE模块提供任务处理通知功能触发加速器到POE获取任务，POE会向加速器发送消息以通知各个加速器来取包。

POE通过PF统一发送消息给各加速器的PF，发送该消息的地址由POE中PF制式的寄存器POE_MSG_ACC_PF_ADDR_H_CFG、POE_MSG_ACC_PF_ADDR_L_CFG。POE添加各加速器的任务通知命令的Outstanding个数可通过寄存器POE_SEND_MSG_OTD_NUM配置。

POE添加各加速器的任务通知命令数据结构见《Hi309a全局数据结构汇总》DIE间消息标签页“POE给各加速器的任务状态的消息数据结构”。数据格式如下：

{Rsv[23:0]，Current_Task_State[7:0]};

Current_Task_state：POE中各VF当前任务状态，若为0b1表示有任务，0b0表示无任务。Rsv暂无使用，填为0。

bit0~bit7的任务状态指示需要加速器使用加速器的SR_IOV_VF_OFFSET寄存器中的vf*_offset作为索引，索引该加速器的各VF的任务指示，例如，若SEC的VF0的vf0_offset为5，则使用Current_Task_State bit[5]作为VF0的任务状态指示。

POE给加速器发送任务处理通知命令的触发条件：

- POE中一个循环计时器，达到定时寄存器POE_MSG_TMOUT_CFG 配置值（该门限值由PF配置，不分制式）时，给各个加速器遍历发送一次任务通知。

版权所有 © 海思技术有限公司

- POE中加速器可GET_POE的Group任务从无到有和从有到无时，发送任务通知命令更新加速器中任务通知寄存器的任务状态。

须知

当启用一个加速器的一个制式（VF0~4）业务功能时，必须保证该加速器的PF制式已经能够接收来自于POE的任务通知消息的写寄存器操作时，才可通过配置POE的相应VF0~4下的POE_GRP_ACC_MASK和POE_MSG_ACC_EN对应bit位为1，其中POE_MSG_ACC_EN按VF可配置加速器使能。

9.5.3.2 BMU 的 QoS 反压

BMU对PPE，支持对16个逻辑Pool，每个逻辑Pool的9个Qos单独配置双门限丢弃水线，水线单位粒度为64个Buffer，Qos反压的产生和撤销需要检查逻辑Pool和物理Pool的资源情况，当逻辑Pool的Buffer总数小于其设置的Qos低水线产生反压指示，指示接口模块（PPE）丢弃对应报文；当逻辑Pool的Buffer总数大于其设置的Qos高水线，撤销反压接口模块（PPE）的指示，采用写消息反压PPE。

须知

BMU的Qos反压非绝对准确，因为在发生Block在逻辑POOL和物理POOL之间迁移以及Buffer在LIFO和逻辑POOL之间溢出回读时，会导致剩余Buffer计数更新不准确，产生误反压。

在BMU工作时如果要关闭写消息反压，需要先关闭PPE的QoS接收使能，再关闭BMU的QoS发送使能。

9.5.3.3 POE 对 TM 的反压

TM推送POE的任务，实际上是通过POE送给CPU的。TM推送POE的消息格式中包含POE的VQI bit[9:0]，最终映射到Group。而TM的调度是基于TM的L1/L2/L3/L4等信息的，调度出一个PD后再根据对应的信息得出相应的VQI信息。因此对于TM而言，实际需要的是被反压的L1/L2/L3/L4，但对于POE而言，目前并没有Group对应TM的L1/L2/L3/L4信息。因此POE的反压仍然基于物理Group产生，由TM完成反压映射到端口。POE对TM的反压连线为6组，高电平有效。

9.5.4 HAC 的总线接入

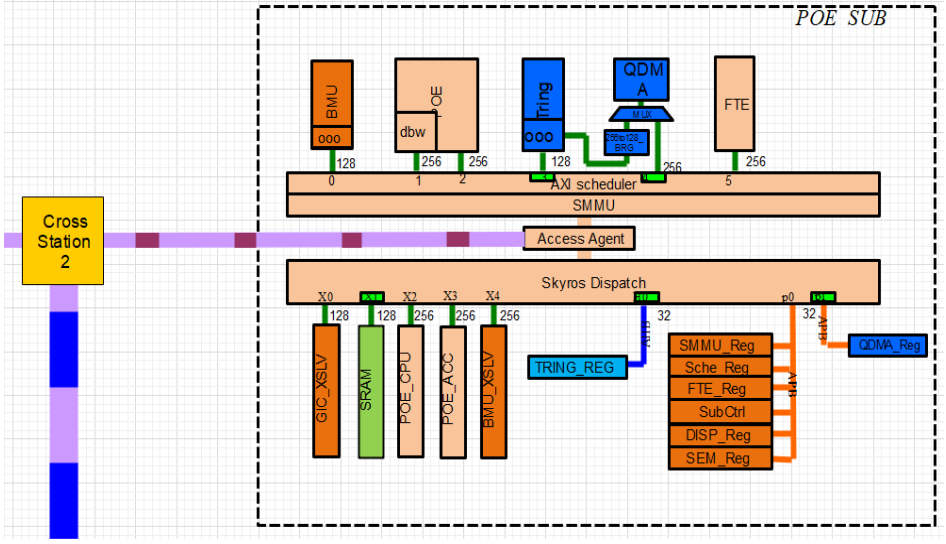
下面图9-3是Subsys接入系统RingBus的拓扑。

Access Agent（AA）模块为Cross Station节点接入前的Buffer模块；

SMMU具有保序通道和非保序通道。SMMU接收到的AXI保序命令数据必须在一个Cacheline范围以内，即不跨越512bit边界。

版权所有 © 海思技术有限公司

图 9-3 Subsys 的总线接入



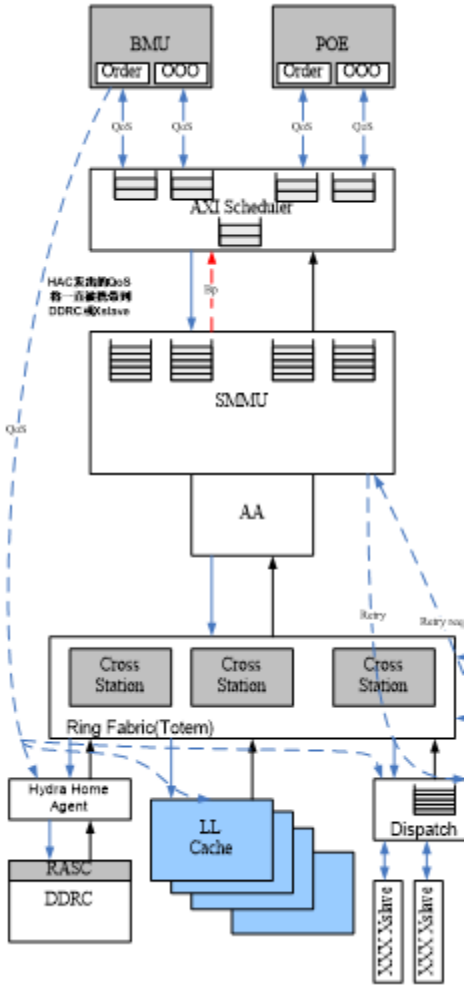
其他加速器/接口控制器接入RingBus的拓扑与上述类似。

9.5.5 系统中 QoS 的传递

在系统中，LLC、Ring Fabric的峰值带宽较大，对于长期业务可以认为基本是无阻塞的（存在瞬时阻塞的可能，这里不作进一步分析）。

版权所有 © 海思技术有限公司

图 9-4 QoS 传递示意



9.5.5.1 HAC

HAC可以根据业务的不同，配置不同类型AXI master访问的QoS（目前支持的位宽是4bit），该QoS将会被一路携带到LLC，DDRC或AXI slave设备；

HAC内部对不同AXI master访问的调度不作统一要求，要求保证业务性能，不会相互阻塞。

9.5.5.2 AXI Scheduler

对加速器/接口控制器送过来的操作所携带的QoS透传；

AXI Scheduler上不区分请求是否需要保序，入口每个端口命令Buffer深度为2，AXI Scheduler的出口命令Buffer深度为2。数据Buffer 的深度为8。AXI Scheduler对HAC发出的操作不区分是否保序，依赖本地配置的权重对不同的AXI master接口作WRR调度。

AXI Scheduler的操作频率典型是总线的半频，也可以根据对应Subsys的情况与总线异步。AXI Scheduler本身的最大带宽必须可以支持下挂的加速器所需要的总线访问的带宽需求，不允许因AXI Scheduler本身的处理能力引起长期反压。

版权所有 © 海思技术有限公司

AXI Scheduler可响应SMMU的停止调度保序读命令、要求停止调度保序写命令、停止调度非保序读命令、停止调度非保序写命令的反压。

9.5.5.3 SMMU

在SMMU 进行保序与不保序的Buffer分离。SMMU的最大总outstanding 和总Buffer深度为192，但写操作能占据的Buffer的深度最大不允许超过128。

对保序操作，Infinite上的SMMU可缓存的读写操作数量各为24；Totem上的SMMU读写分别可配为16/20/24（不能读写分别配置）。

SMMU可反压Scheduler，读写分离，是否保序分离。不会因一种命令堵死。

SMMU会对QoS[3:0]透传，SMMU本身可以区分高/低优先级，高优先级优先调度。相同优先级的不同操作类型RR。

SMMU工作在系统总线的半频。

SMMU与Dispatch间支持retry（详见Dispatch节）。

9.5.5.4 AA 与 Cross Station

在Hi309a中，Ring Fabric的峰值带宽较大，基本上可以认为是无阻塞的，不是QoS及反压流控分析的主要关注点。

AA透传QoS[3:0]，不根据QoS调度；Ring上没有反压，Cross Station也没有基于QoS的调度。AA及Cross Station对过环和上/下环操作的调度方式是：上环和过环之间过环绝对优先，双向下环之间RR。

9.5.5.5 LLC

LLC把QOS分为4个等级：HH，H，M，L。 4个优先级使用QoS[3:2]来区分，但当cache不命中时，完整的QoS[3:0]会被传递下去。

不同QoS共享16深的Buffer。使用绝对优先级且防饿死算法。高优先级的优先仲裁进入流水线处理，低优先级的优先被retry，但长时间未被执行的操作优先级会升高，如果一个请求在LLC内部等待时间timout，等级就会升高。

LLC的cache替换算法，也可考虑QoS，这里不做详细描述。

须知

各加速器要注意使用QoS[3]，如果只使用QoS[2:0]，在LL cache就只能映射出2个优先级。

9.5.5.6 Dispatch

Dispatch入口与RingBus对接，Dispatch的操作频率为总线频率的一半。

Dispatch出口与各个Register和IO Device相连，Dispatch与这些设备间可以工作在1/N同步模式或任意异步频率模式。

Dispatch有入口队列，共16深，对读写操作可区分QoS，采用SP调度，优先调QoS最高的访问。同时，有timeout机制，对长期未被调度出去的操作会提高该操作的优先

版权所有 © 海思技术有限公司

级。不同QoS 的Buffer存满会retry，优先retry 低QoS的操作。对于要Retry的操作，Dispatch返回对应的ID，SMMU根据ID重传（Retry）该操作。

Dispatch会对相同地址范围内的读写操作进行保序。

9.5.5.7 DDRC 及 HHA

DDRC配合系统内部总线上的QoS信息进行流量管理，DDRC支持16级QoS，命令优先级按照QOS15>QOS14>QOS13…>QOS0排列；

DDRC同时支持两种QoS调节功能：

命令优先级自适应功能：命令在N个周期（见寄存器DDRC_CFG_QOS）没有得到仲裁，则自身命令会提高一个优先级。

命令老化功能：命令如果自身因为DDR的时序参数不满足而得不到仲裁，在N（见寄存器DDRC_CFG_QOS）个周期后可以强制提起请求申请仲裁。

Hydra Home Agent QoS透明，透传QoS。Hydra Home Agent会调整地址顺序以最大化DDRC带宽。

9.5.6 DDR 调度

DDR调度效率分三个维度：

- 效率：DDR数据总线有效传输时间和总时间比较值，在lmbenchfrd项达88%以上。
- 优先级：每个DDR访问命令可带一个优先级，期望优先调度高优先级命令。
- 公平性：某个命令或某个ID的命令被其他命令阻塞的严重程度，超过一个阈值之后认为触发公平性问题。

DDR调度在效率上遵从以下原则：

- FRFS原则：DMC内部的命令只有当时序满足之后才能参与调度，即First Ready First Serve（FRFS）原则。
- 读写优先原则：当DMC有Rd、Wr、Act、Precharge这四类命令可参与调度时，调度优先级分为两档：Rd &Wr > Act & Precharge，称为读写优先原则。
- 写命令汇聚原则：DDRC内的写命令达到一定的数量之后再集中发送写命令，减少写读切换的频率降低写读切换带来的效率下降。同时避免读写命令一起在DMC内相互争抢Bank资源的带来的效率下降。
- BG均衡原则：DDRC实现对已调度的各BG的Burst长度进行统计，精确到Burst长度的BG均衡调度算法，使DMC能够很容易实现BG轮转，以减少由同BG命令连续发送造成的效率损失。

9.5.7 DDR 拥塞管理

优先级传递原则

- 当新进入DMC的高优先级命令发现已有命令阻塞它的发送时，将把高优先级命令的优先级传递给阻塞它的命令。
- 造成阻塞的原因可以是ID冲突、地址冲突等。

根据效率优先策略，没有满足时序的命令即使优先级再高也可能遇到长时间无法发送，例如命令遇到其他大量Rowhit命令，使得命令Latency非常大甚至系统挂死。完全遵循效率优先策略可能会导致这种不公平的问题，所以DDRC提供了公平性策略：

版权所有 © 海思技术有限公司

1. Timeout机制
   a. DDRC为每个命令配置一个Timeout值，当计数达到时标记为Timeout命令，然后不管是否满足时序都强行参与调度。
   b. Timeout的命令将优先于其他非Timeout命令被调度，多个Timeout命令则遵循优先级和先进先出原则。
   c. Timeout功能可以关闭。
2. Row-hit打断机制

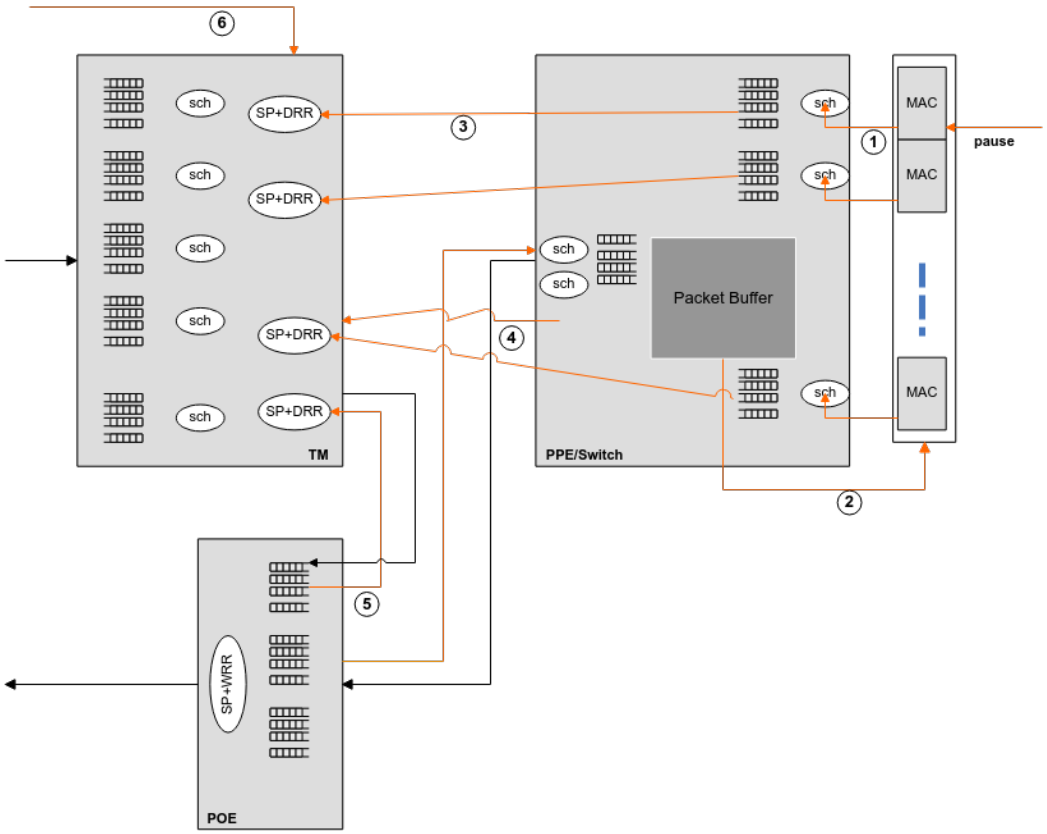
   DDRC内只允许若干连续的Rowhit的命令发送，之后如果有同Bank的其他row的命令，可以打断Rowhit的命令的连续发送。
3. 绿色通道

   DDRC为QoS最高的命令提供绿色通道，属于绿色命令不受效率优先的限制，以尽可能小的lattency返回响应。对于lattency敏感且流量不大的业务，适合配置绿色通道。

9.5.8 对外的反压流控

Hi309a的队列和调度主要部署在PPE，POE和TM，因此业务可见的反压流控也主要由这三个模块体现：

图 9-5 对外反压流控



反压1：MAC接收并解析pause帧后，会在发完当前包后直接反压（或通知PPE内部Switch的端口调度器停止调度）；

版权所有 © 海思技术有限公司

反压2：当PPE内部Switch的片上包缓存达到阈值时可以触发MAC产生pause帧（包缓存采用共享管理，但会统计每个端口收到包所占用的缓存），pause功能可以关闭；

反压3：当PPE内部Switch的队列拥塞时，可以对TM反压（前提是TM的调度层次包含物理端口）；

反压4：当PPE内部Switch的片上包缓存达到阈值时可以反压TM；

反压5：送CPU的业务可以在TM整形后再送CPU，此时POE可以在对应Group队列深度达到阈值时反压TM的对应调度节点；

反压6：CPU可以根据业务需求对TM置“软反压”。

9.5.9 内部反压和流控的基本原则

系统业务层面的队列主要部署在2个模块：POE，TM。其中，TM的队列主要作为出口队列，POE的队列主要作为入口队列。

对于出口队列（TM），一般情况下不对前级反压，队列满则丢弃，并支持通过RED等手段避免队列满（full）。

对于入口队列（POE），可以选择对前级反压。

除TM和POE外，PPE和BMU也会反压前级模块。

9.5.9.1 加速器对 CPU 操作响应的原则

系统支持如下与加速器的交互方式：

POE，CPU直接访问（如不作特殊说明，本节的访问一般是指添加任务，获取任务，而不是寄存器访问）；

CPU通过POE进行Forward_TM访问时，POE接收到CPU的写操作（将任务写入POE的接收FIFO）会对CPU发出总线应答。

对于TM和PPE，CPU的直接添加任务操作仅用于调试（寄存器还是CPU直接读写）。

加速器对于CPU访问，应满足以下要求：

支持按照业务流程CPU需要访问的加速器（POE、BMU），连续访问时不需要查询Valid状态决定访问是否成功（CPU访问PPE是用于DEBUG，允许先添加再查询）；

HAC对CPU的反馈，不允许总线返回Error（除非访问异常）。真满丢弃，上报中断，释放Buffer；对于假满，采取内部重传或反压AXI slave口的FIFO，在保证只是瞬时反压的情况下，当AXI slave口的Buffer（一般是FIFO）满时可以反压总线。但建议不要同时反压所有的PF/VF，以及不要同时反压CPU和加速器。

对于POE，当内部反压重传的次数达到预设的最大值时丢弃任务，上报中断，记录第一个任务（任务/消息）。

目前POE为每个Group提供一个双Threshold配置（GRP_BP_TH寄存器），当队列中的任务个数达到Threshold时，此队列不再接收接口模块（PPE）的任务，但仍可接收CPU的包任务。

9.5.9.2 加速器间反压的基本原则

加速器间的反压，假满和真满可以由发起方和目的方联合处理，也可以只由目的方处理而发起方不感知。反压通过总线响应完成的。

版权所有 © 海思技术有限公司

假满是加速器的片上Buffer满，真满是加速器不仅片上Buffer满，而且片外Buffer也已全部被占用。

当假满和真满可以由发起方和目的方联合处理时，假满需要由发起方发起重传，重传可配置最大次数，当达到预设的最大重传次数限制时，丢弃任务，上报中断。考虑到除非发生异常不会达到预设的最大重传次数限制，因此在涉及Buffer的情况下，假满重传达到次数要求时不要求释放Buffer（建议释放，但不作要求，也可以不释放）；

真满丢弃任务，并需要释放Buffer。添加的目的方总线反馈真满，上报中断（中断区分pf/vf），任务丢弃，发送方进行释放Buffer处理。

异常错误统计由发送方完成。

当假满和真满只由目的方处理而发起方不感知时，假满由目的方将PD缓存在目的方模块内部，先返回OK给发起方，然后进行内部重传，重传可配置最大次数，当达到预设的最大重传次数限制时，丢弃任务，上报中断。考虑到除非发生异常不会达到预设的最大重传次数限制，因此在涉及Buffer的情况下，假满重传达到次数要求时不要求释放Buffer（建议释放，但不作要求，也可以不释放）；

真满丢弃任务，上报中断（中断区分pf/vf），记录PD信息，由硬件释放Buffer。

异常错误统计由目的方完成。

对于属于输出队列性质的TM，真满直接丢弃，且不对前级模块反压。而对于其他加速器，应尽量（不作强制要求）通过硬连线或反压消息通知前级模块，避免真满丢弃。

PPE反压TM是通过硬连线反压，POE反压TM是通过硬连线反压，BMU通知PPE按QoS丢包是通过消息的方式。TM、PPE不会返回真满。TM不会返回真满的原因已作过说明。PPE是因为PPE会对TM反压，因此不存在队列溢出操作，PPE也不会假满（因此也不会返回假满）。

9.5.9.3 加速器间的指示

模块间有2类指示：

反压消息/硬连线反压，反压消息传递的方式是发送方将反压消息通过AXI master接口写到对端的特定寄存器。硬连线反压指的是同一个Die上的加速器间的反压通过连线传递。

POE的“有任务指示”，表现为POE通知相应加速器有任务时全用消息传递方式。

9.5.10 CPU 添加加速器反压和异常处理

9.5.10.1 CPU 添加 POE 的处理方式

POE可配置为两种方式：“查询模式”、“非查询模式”（通过寄存器POE_VCPU_ADD_MODE配置，详见POE章节的描述）。目前的业务流程中主要采用“非查询模式”。

非查询模式时，CPU添加任务不需要查询状态确认添加是否成功，出现添加失败时，POE会上报中断给添加任务/消息的逻辑CPU。

查询模式时，CPU添加一个任务后必须读取CPU对应的状态寄存器VCPU_ADD_ST确认是否添加成功。

对于逻辑CPU添加POE包任务或消息，POE通过总线反馈OK，查询模式中通过查询寄存器报告状态，非查询模式时（任务入队、真满、虚拟Group号错误、队列配置为禁

版权所有 © 海思技术有限公司

止入队、VFThreshold反压、内部RAM校验错误）都标记为添加成功，并有中断上报，错误的任务会写入“任务失败缓存”中。

CPU和加速器添加的任务响应处理是分开的，不会相互阻塞。虽然POE的slave口只有一个，如果CPU加包造成总线反压，会造成加速器的命令也进不来。但该反压只是瞬时的，可以接受。

CPU添加任务区分VF，不同的VF间不会相互阻塞。当假满时同一个VF的添加不同Group/队列的任务/消息可能存在相互阻塞。

CPU添加任务到POE时，POE可能会“真满”和“假满”：

- 真满：POE的当前添加的队列，片上和主存的溢出空间全满，不能添加任务。

  逻辑CPU添加任务队列的片内和DDR空间全满，任务丢弃，上报真满中断给对应制式的逻辑CPU，添加失败的任务写入失败缓存空间，按制式记录失败次数。VCPU_ADD_REAL_FULL_CNT寄存器；
- 假满：在出现总线拥塞时，溢出到DDR的操作不及时，可能导致后面溢出的操作被堵塞，引起POE的假满状态。

  逻辑CPU添加任务，队列假满状态时，POE内部重试添加，直到重试时间到达寄存器VCPU_ADD_RETRY_TIME_CFG所配置的假满重试时间时，任务丢弃（不要求释放Buffer），上报假满重试中断给对应制式的逻辑CPU，添加失败的任务写入失败缓存空间。

  POE支持推送核间消息或任务到Cacheable的存储空间，支持针对Cacheable的存储空间的队列轮询和反压机制（仅限GET_POE操作），用于优化提升CPU获取包任务或消息的性能。

  POE通过总线对Cacheable的主存操作与处理器交互。

9.5.10.2 CPU 向 POE 获取任务的处理方式

CPU可指定POE中Group/队列获取包任务/核间消息。每一个逻辑CPU都对应64组256bit的读寄存器用于GET_GROUP。每一个逻辑CPU都对应512组256bit的读寄存器VCPU_Q_WORD{0~7}_GET，用于指定队列获取任务（GET_QUEUE）。

如果POE中有待处理的任务，则调度出队返回给CPU 1个待处理的任务；如果当前对应的Group/队列没有待处理的任务，POE反馈给处理单元的数据中“Node_Code[6:0]”域值为“0x7F”（即获取失败标识）。

POE也支持轮询Cacheable的主存空间，任务有效时，POE从Cache获取CPU要添加的任务，支持针对Cacheable的存储空间的队列的轮询和反压机制，用于优化提升CPU添加包任务或消息的性能。

POE通过总线对Cacheable的主存操作与处理器交互。CPU和POE共同在内存中维护1-16个环形队列，每个队列8深。作为标准的生产者-消费者模型。

9.5.10.3 CPU 向 BMU 申请和释放缓存

BMU对于CPU申请缓存的处理机制：

- 申请成功：返回Buffer指针。
- 逻辑Pool没有缓存：资源耗尽，给加速器和CPU返回数据全0。
- 逻辑Pool的LIFO空： 内部LIFO空，待回读补充Buffer，返回给CPU可配置的值（默认为全“0”， 寄存器BMU_EMPTY_DATA可配置），返回给加速器全0。
- 逻辑Pool复位或者BMU独立复位时，返回CPU可配置值（BMU_RST_BP_DATA，对于加速器，此时返回全0）。

版权所有 © 海思技术有限公司

- CPU申请Buffer时，BMU不会返回Error响应（BMU也不会对加速器返回Error响应）。

CPU向BMU释放缓存：

- 释放单Buffer：BMU通过总线返回OKAY（0b00）响应。
- 释放链式Buffer：BMU通过总线返回OKAY（0b00）响应。

在BMU中提供16任务缓存，用来吸收总线上2个cluster的64个和加速器的（不超过）64个outstanding写操作，将添加任务全部接收，但暂时不反馈response，完成地址译码后返回response；CPU和加速器添加的任务响应处理是一起的，会相互阻塞。

9.5.10.4 CPU 添加 TM 的处理方式

如前所述，Hi309a的业务流程中，CPU不会直接对TM添加任务，而是通过POE进行Forward_TM的操作。CPU直接添加TM的机制只可以作为调试机制。下述CPU添加TM的机制仅供参考：

CPU添加任务到TM时，可能会添加失败，失败的原因和上报方式如下：

- 假满：总线拥塞，TM来不及溢出，当前添加任务的队列片上Buffer满，但DDR还有空间，即假满。假满的任务会阻塞后续操作，并可能在进一步引起入口Buffer满后反压总线。
- VCPU添加任务失败（含真满）：添加任务失败，给CPU返回OK，上报中断给对应的VM，真满时TM要向BMU释放Buffer，其他添加任务失败不释放Buffer给BMU，记录第一个错误的源VFID、目的VFID、VCPU ID号。
- 中断上报给对应的VF。
- CPU和加速器添加的任务响应处理有分开的Buffer，不会相互引起头阻塞。
- 一个VF的所有命令如果第一个命令阻塞，后面的所有命令一起阻塞（即一个逻辑端口或物理端口阻塞，其他的一起阻塞）。

9.5.10.5 CPU 添加 SEC 的处理方式

SEC支持BD Fifo模式，该模式下SEC分配了1个地址给CPU添加任务，该模式仅用于安全启动时。CPU先查询再添加，不支持outstanding的添加。满丢弃，上报中断（此时尚无VF/PF的区分），记录最早的一次。

9.5.10.6 CPU 添加 PPE 的处理方式：

CPU添加PPE主要用于调试，不作为正常业务流程。

分配了16个地址给CPU添加任务，CPU向某一个地址添加1次任务后查询是否成功，同一组地址不支持outstanding的添加。

9.5.11 加速器间访问失败处理

9.5.11.1 模块间反压基本原则

任务推送者收到假满（即收到DECERR响应），如果没有达到连续失败次数（软件可配置），暂停一段时间（时间间隔软件可配置）后重新添加；如果达到连续失败次数，丢弃任务，记录最早的一次任务（存储格式和推送任务的数据结构相同）、按制式进行统计（8bit饱和计数器，读清），不要求释放缓存，（可选）上报中断给对应的PF/VF；

版权所有 © 海思技术有限公司

在接收到假满反压时，建议做到（考虑到假满是瞬时的，不做强制要求）各个VF的发送通路相互不堵塞。例如，当推送VF1的任务收到“假满”反压时，推送其他VF的任务不应受影响。

对于支持outstanding>1推送的场景，如果某个任务被反压，该任务后已发送出去的任务都要进行重传。

事实上，加速器不会收到真满响应——TM直接丢弃，BMU的真满意味着缓存出现错误。当POE接收到任务发现队列真满时（即队列所有存储空间满，也包括队列未使能），POE上报中断给对应的VF，丢弃任务，每个VFID记录一个最早的GRP、QoS（VFID通过地址区分）、进行统计（8bit饱和计数器，读清），由硬件释放缓存；

内置NP（SPPE）、TM、PPE推送POE，内置NP（SPPE）推送TM，在Hi309a采用Credit机制。

9.5.11.2 加速器添加 POE

支持向POE添加任务的加速器有SEC/TM/PPE/集成NP（SPPE）/TRING /POE；添加任务均为256bit single操作；

POE为每个Group提供一个Threshold配置（GRP_BP_TH寄存器），当Group中的任务个数达到Threshold时，此Group不再接收接口（PPE）的任务添加，丢弃PPE添加的任务，POE释放Buffer； 到达GroupThreshold时，只反压PPE的任务添加，不影响其他处理单元的任务添加；

POE缓存队列空间（内部缓存空间和DDR空间）满时，反压所有处理单元的添加操作，上报真满中断，任务丢弃，由软件释放Buffer；

POE出现假满状态时，不反压加速器的添加操作，通过总线反馈OK给加速器；任务缓存在POE内部，POE负责重传（Retry），重传达到次数限制时丢弃该任务，记录最早的一次任务（存储格式和推送任务的数据结构相同）、按制式进行统计，不释放缓存，上报中断给对应的VF；

POE任务队列使能关闭时，通过总线反馈OK给加速器，由POE上报中断，任务丢弃，硬件不释放Buffer，由软件释放Buffer。

9.5.11.3 加速器向 POE 获取任务

当加速器在POE中绑定的队列有包任务时，POE通过消息通知加速器。加速器收到消息后，通过总线向POE获取任务。

POE收到加速器的任务获取操作后，如果POE中有待处理的任务，则调度出队返回给加速器1个待处理的任务；如果当前对应的Group/队列没有待处理的任务，POE反馈给加速器数据中“Node_Code[6:0]”域值为“0x7F”（即获取失败标识）。

9.5.11.4 加速器添加 TM

可能添加TM的加速器为：POE，SPPE（EMIO/PKIO）。

加速器添加PD到TM共用支持最大Outstanding数为12的Buffer，其中SPPE-EMIO/SPPE-PKIO添加PD最大Outstanding 16，POE添加PD最大Outstanding 8，剩余为预留值；

加速器通过MASTER添加包任务到TM，且队列缓存已满时，TM会发生NO_BUFFER丢弃，将PD对应的Buffer释放给BMU，并基于队列记录是否发生过NO_BUFFER丢弃的状态值（寄存器TM_FQ_NOBUFFER_DROP_STATE），一共512bits，其中每个bit状态值可分别进行写1清操作，且必须由PF来进行清除状态操作，对于VF只读；

版权所有 © 海思技术有限公司

其它参考模块间反压基本原则。

9.5.11.5 加速器向 BMU 申请缓存

当加速器需要向BMU申请Buffer时，该模块通过读对应逻辑Pool的寄存器地址来向指定的逻辑Pool申请Buffer，BMU根据地址解析出对应的逻辑Pool编号，从对应逻辑Pool的LIFO中读出Buffer指针反馈给申请模块。BMU对于申请缓存的处理机制：

- 申请成功：返回Buffer指针。
- 逻辑Pool没有缓存：资源耗尽，给加速器返回数据全0。
- 逻辑Pool的LIFO空： 内部LIFO空，待回读补充Buffer，返回给加速器全0。
- 逻辑Pool复位或者BMU独立复位时，返回全0给加速器。

建议发起申请缓存的加速器处理机制：收到返回数据为0后认为申请失败，暂停可配置时间后再申请，连续失败次数达到可配置次数后上报中断。

9.5.11.6 加速器向 BMU 释放缓存

加速器向BMU释放缓存：

释放单Buffer：BMU通过总线返回OKAY（0b00）响应。

释放链式Buffer：如果链式Buffer释放成功，BMU通过总线反馈OKAY响应，否则返回假满（0b11）响应，加速器需要重传返回假满响应的Buffer。

释放分离式Buffer：如果分离式Buffer释放成功，BMU通过总线反馈OKAY响应，否则返回假满（0b11）响应，加速器需要重传返回假满响应的Buffer。

BMU中提供足以吸收总线上来自CPU的outstanding写操作和加速器的outstanding写操作的缓存（缓存深度暂定为128），可将添加任务全部接收。但接收后不马上返回总线response，而是在完成地址译码后返回response。CPU和加速器添加的任务响应处理是一起的，会相互阻塞。

须知

若释放操作的总线握手后，必须保证此次总线操作完成后才停止释放动作，避免总线挂死。

9.5.11.7 加速器添加 PPE

可能添加PPE的加速器为： TM。其中TM添加PPE是业务流程的主要操作方式。

PPE不会返回“真满”及“假满”，对于TM，PPE可通过硬连线在达到设置的阈值时反压TM，TM看到反压状态为有效则不添加，如果TM未及时响应反压，PPE上报中断。PPE反压包含推送接口的端口级反压，对于PPE反压，TM对应端口接收到其硬反压后停止发送PD，当反压撤销后，继续推送动作

其它参考模块间反压基本原则。

版权所有 © 海思技术有限公司

9.5.12 包任务和 Buffer 传送 outstanding 机制各加速器间的互相访问关系及支持的 Outstanding

无。

9.5.13 报文 QoS

9.5.13.1 报文 QoS 生成与调度

Hi309a支持根据L2的COS域段、L3的TOS域段等获取报文优先级的信息。并支持由报文的优先级信息映射出内部的QoS信息、队列号等信息。

9.5.13.2 报文 QoS 与总线 QoS

报文QoS可以映射到总线QoS。理论上，是可能减小高优先级报文的端到端处理时延的，但目前尚无使用先例，也无成熟和完整的机制，并可能影响DDR的带宽。

9.6 软件指南

软件需要关注Hi309a的反压机制中各加速器、IO控制器对于CPU添加任务的响应方式，以及相关的中断；

对于加速器间的反压，软件无需直接关注，但要注意可能发生的异常以及中断；

Hi309a的各级模块的QoS、调度配置的默认值会尽可能满足产品的要求，但将Hi309a用于其它场景时，软件需要根据应用场景核对是否需要修改相关配置。

9.7 性能

通过消息传递的反压，会尽量使用事件触发方式，只有当反压状态变化时发出反压消息，以降低对系统总线和带宽的影响。

POE的“有任务”指示使用事件触发结合定时发送的方式。

9.7.1 延迟（Latency）

相较于硬连线反压，反压消息的延时较大，跨die的消息传递时延典型情况下预计在40-60ns之间，设置反压阈值时要考虑到这一点。

POE的“有任务”指示同样可能存在时延，当向POE发出取任务操作时，POE实际可能已经没有在缓存中的任务了，此时POE会返回“空”。

9.8 DFX

9.8.1 可靠性

9.8.1.1 自恢复

目前非数据完整性错误（即ECC2bit错）是否会丢Buffer的情况如下：

版权所有 © 海思技术有限公司

1. NP（SPPE）：释放链式Buffer假满重传到最大次数，会泄露Buffer。
2. TM：释放链式Buffer假满重传到最大次数; 还有一个调试型中断回读qid校验错误。
3. SEC：不会丢包，不会泄露缓存。
4. SA：不会丢包，不会泄露缓存。
5. HPRE：不会丢包，不会泄露缓存。
6. BMU：不会丢包，不会泄露缓存。
7. PPE：如下情况会泄露Buffer：

描述符非首片refltselfaddr为0时之后的节点的Buffer

描述符非首片DateLength为0后续节点Buffer

描述符首片pkt_len=0对应的报文Buffer

报文pkt_len小于15Byte的报文Buffe

OWN域非0xA，会泄露报文和描述符Buffer

L3报文的L2LEN/L3LEN错误，会泄露Buffer

pkt_len超出最大长度范围，泄露Buffer

pkt_len不等于datalength累加长度的任务，后续节点Buffer会泄露

出现上述描述符错误时，会泄露Buffer。

9.9 注意事项

Na。

版权所有 © 海思技术有限公司

10 系统功耗

10.1 关键术语

表10-1为与System Power相关的关键术语表。

表 10-1 System Power 术语表

Terms	Description
AVS	Adaptive Voltage Scaling 自适应电压调节技术。
HPM	片上集成的硬件速度监控单元（HPM，Hardware Performance Monitor）。
PMU	Power Manage unit，外部电源管理芯片。
PPU	Power Policy Unit，供电策略单元。
WFI	Wait For Interrupt，CPU等待中断唤醒状态。
WFE	Wait For Event，CPU等待事件（SEV指令或者终端）唤醒状态。
CP	Circuit Probe，即Wafer测试。
FT	Final Test，即芯片测试。
PC	Process Class，根据芯片工作速度确定的芯片等级。
Isolation Cell	隔离逻辑单元，用于Power Gating时钳位被Power Down的模块的输出到固定电平。
Level Shifter	电平转移器。接口物理电平不一样，但是逻辑电平一样。
Power Gating	门控电源技术。一种通过关闭电路电源降低功耗的技术。
Clock Gating	门控时钟技术。一种通过关闭电路时钟降低功耗的技术。

版权所有 © 海思技术有限公司

10.2 概述

系统功耗描述Hi309a各业务场景的功耗、功耗分布、功耗管理方案、电源方案、功耗优化技术和使用流程等。

主要内容：

- Hi309a应用场景功耗要求。
- Hi309a的功耗组成。
- Hi309a功耗管理方案。
- Hi309a电源方案。
- Hi309a功耗管理流程。

10.3 特性

系统功耗具有以下特性：

- 支持模块级时钟门控，软件可以配置每个模块关闭和打开时钟。
- 支持A55 Cluster软件可配置的动态门控功能，包括SCLK、ATCLK、GICCLK和PCLK，动态门控由Cluster Q-Channel接口实现。
- 支持A55 Core独立动态上下电功能，8个Core独立上下电控制。
  - 支持基于A55 Core P-Channel的功耗管理接口。
    - 支持A55上下电硬件支持，由硬件自动完成MTCMOS开关、ISO开关、复位控制等。
    - 支持A55上下电的软件流程，由软件接管硬件完成Core/Cluster的上下电控制。
    - 支持Core下电状态下的中断唤醒功能。
    - 支持Core下电状态下的软件唤醒功能。
- 支持Power Gating。
  - 支持网络场景特有模块SPPE_TOP Power Gating，由软件配置。
  - 支持网络场景特有模块HISS Power Gating，由软件配置。
  - eFuse在初始化完成后自动进入Power Down模式。
  - HILINK每Macro和每Lane均提供Power Gating功能。
- 支持数字电源域multi-VDD，CPU和总线模块电压分0.7V/ 0.8V/ 1.0V三种场景。
- 支持CPU和总线模块电压域AVS工艺补偿。
- 支持multi-Vth，降低lvt/ulvt器件比例。
- 支持PLL0、PLL5模块调频，RING/AA/CPU CLUSTER，根据其应用场景调频。
- 支持除总线类型LLC/AA/SMMU/DISPATCH/SCHDULER/RING总线类模块外其他模块的clk gating；支持XSEC、MAC、RoCE单独clk gating。
- 支持单个SERDES MACRO的power gating。

版权所有 © 海思技术有限公司

10.4 Compliance

Power Management Bus 1.2

10.5 功能描述

Hi309a根据封装内CPU核数、频率、Serdes Lane数等配置不同，满足不同场景下的芯片性能功耗规格，达成系统散热、供电和可靠性等设计需求。

为了满足不同场景下功耗目标和省电需求，芯片采用了多种低功耗设计和相应的工作模式降低芯片的静态功耗和动态功耗：Clock gating、Power gating，AVS等。不同应用场景下软、硬件配合完成低功耗模式配置和应用。

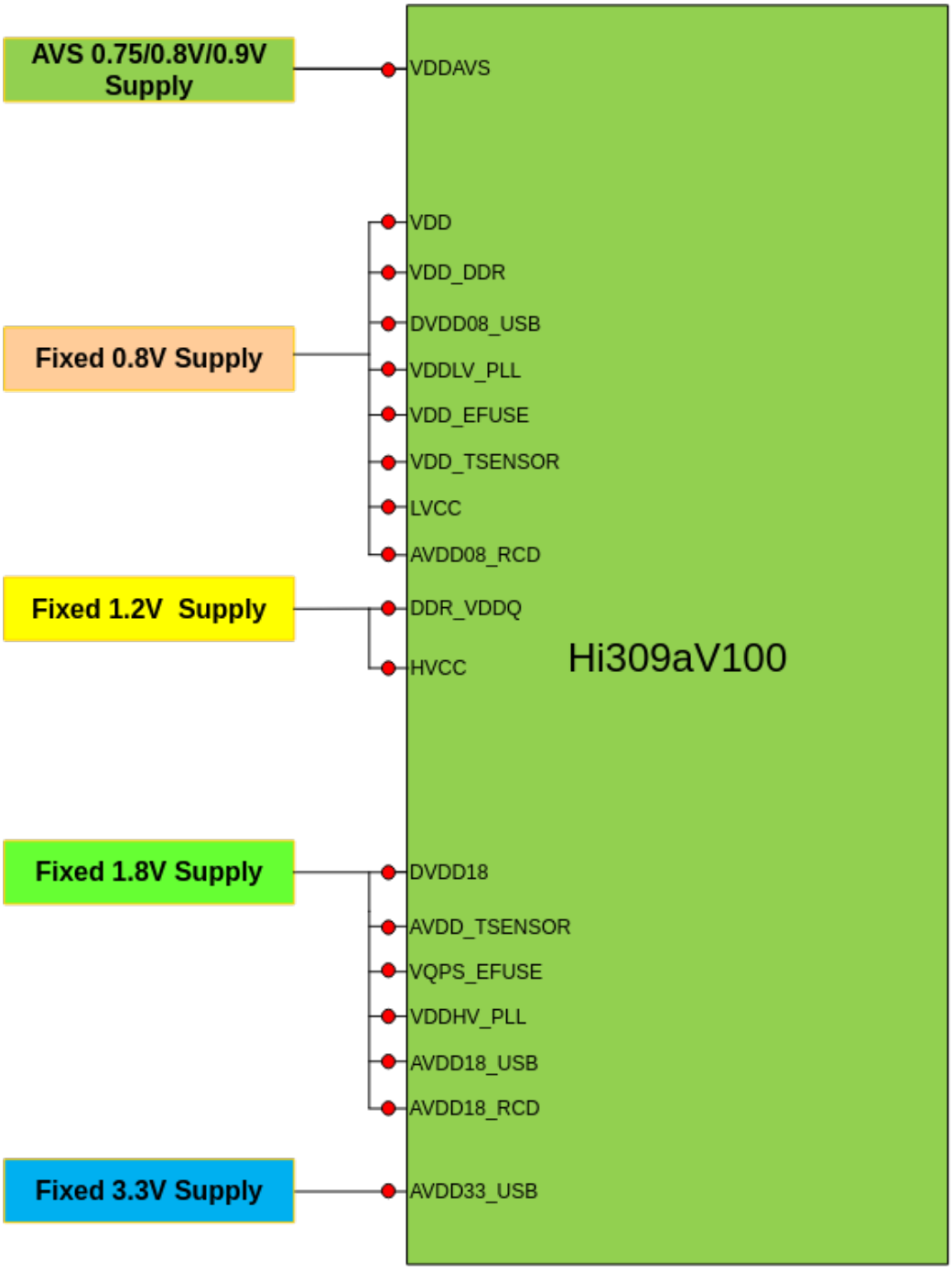
10.5.1 电源域

Hi309a的电源列表参考《 Hi309a 硬件设计指南 》。

单板电源设计方案可参考图10-1。单板AVS 0.8V Supply主要给Core AVS域供电，SIOE模块也由AVS 0.8V Supply供电，可以支持0.75V到0.9V范围内的调压；Fixed 0.8V Supply给Core非AVS逻辑和IP的数字电源供电；HILIN 1.2V电源由Fixed 1.2V Supply电源供电；DDR4/5 1.2V/1.1V电源由Fixed 1.2V/1.1V Supply电源供电；Fixed 3.3V Supply给USB供电； IP的模拟1.8V以及IO由Fixed 1.8V Supply电源供电。

版权所有 © 海思技术有限公司

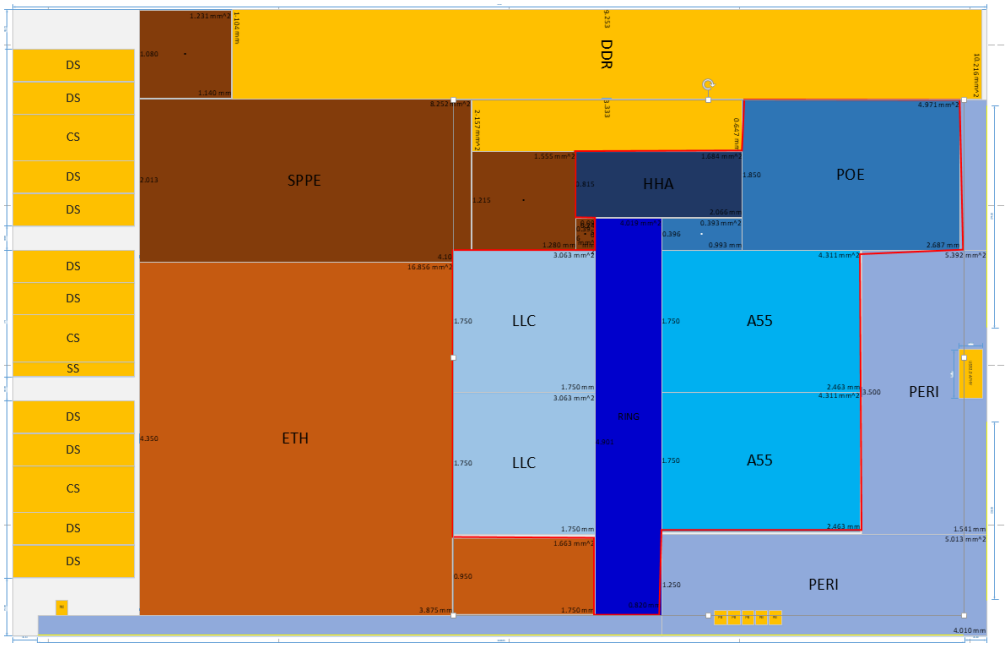
图 10-1 Hi309a 电源合并方案



如图10-2所示，Hi309a数字逻辑中红线框区域内A55、LLC、POE、RING和HHA是一个AVS电源域，由外部电源管理芯片供电。

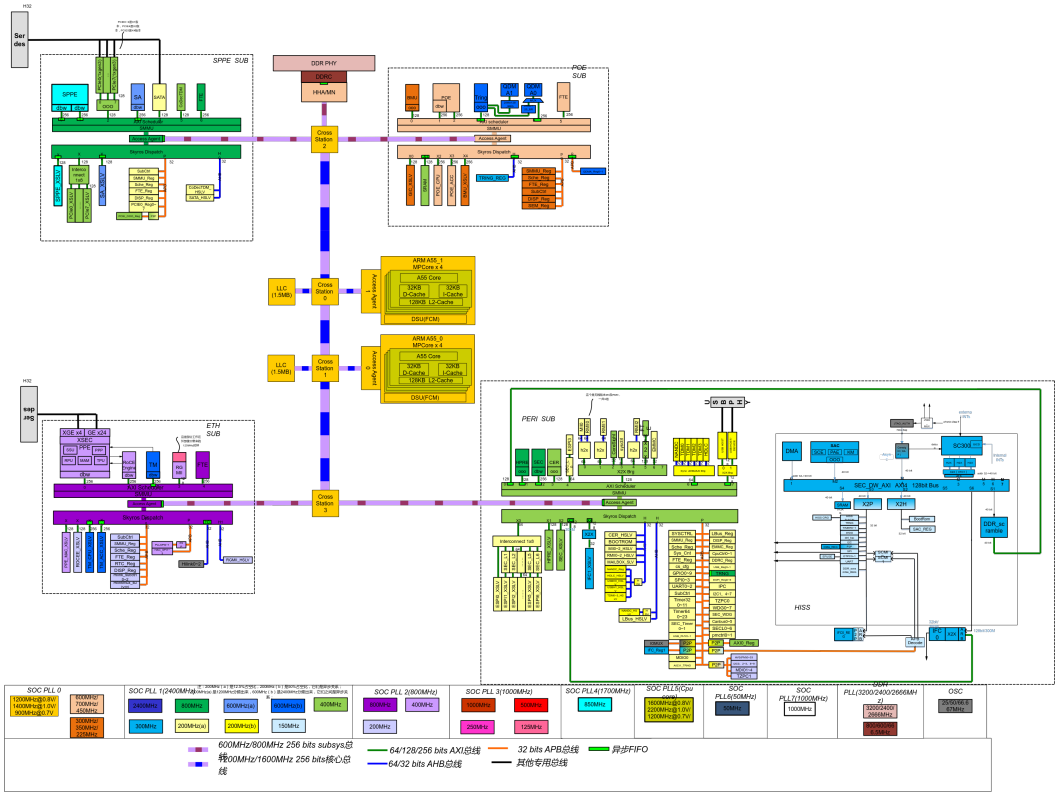
版权所有 © 海思技术有限公司

图 10-2 电源域



10.5.2 时钟域

图 10-3 时钟域



CPU子系统PLL0时钟域支持的典型时钟频率为1.6GHz。

版权所有 © 海思技术有限公司

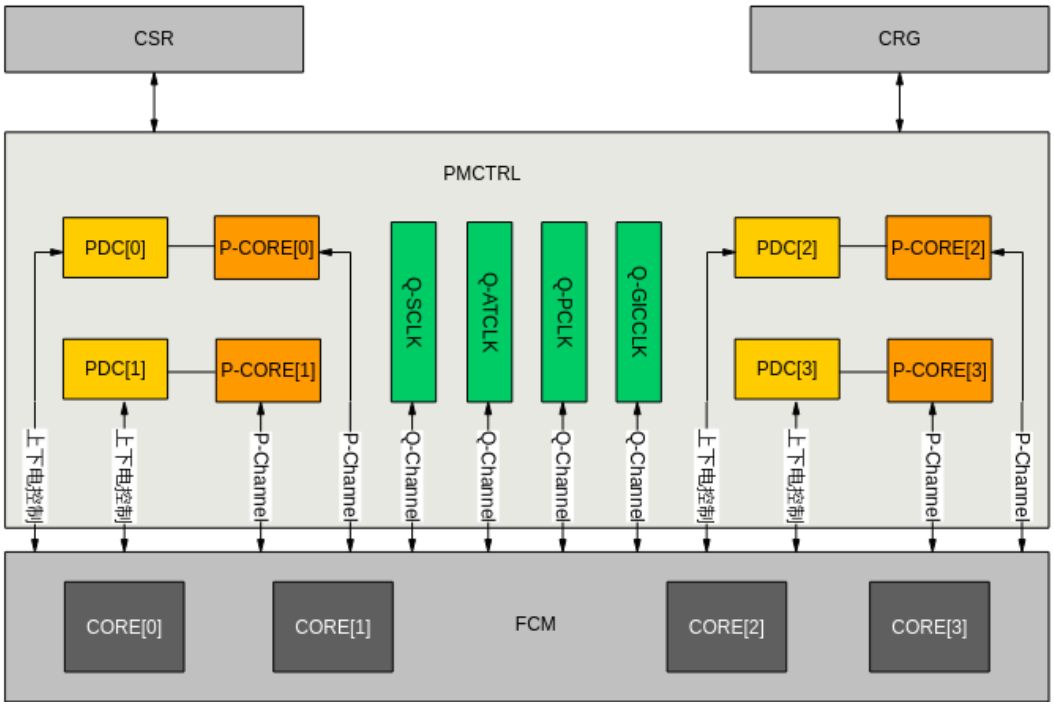
10.5.3 功耗管理

芯片功耗分为动态功耗和静态功耗，CPU子系统、总线、DDR、CT/IT加速器及其业务接口模块是芯片功耗的主要组成部分。各模块和系统的支持的功耗模式与AVS（PMBUS）逻辑构成的软硬件系统可以满足各场景降功耗需求。

10.5.4 CPU 功耗

CPU功耗管理模块PMCTRL主要组成单元是4个CORE_PDC上下电控制模块、4个Core P-Channel控制器、SCLK Q-Channel控制器、ATCLK Q-Channel控制器、GICCLK Q-Channel控制器和PCLK Q-Channel控制器组成。

图 10-4 PMCTRL 结构框图



四路Q-Channel控制器通过Q-Channel接口与FCM低功耗控制单元交互，实现FCM SCLK、ATCLK、GICCLK和PCLK的动态门控；四路Core P-Channel控制器通过P-Channel接口与四个Core的P-Channel接口交互实现4个Core的独立上下电功能。

10.5.4.1 Q-Channel 控制器

Q-Channel接口如图10-5所示，接口信号有QACTIVE、QREQn、QACCEPTn和QDENY。其中QACTIVE由器件给出，低有效，用于通知控制器器件侧当前可申请对应时钟关断；QREQn由控制器给出，低有效，用于向器件申请时钟关断；QACCEPTn由器件给出，低有效，用于通知控制器器件侧同意相应时钟门控关断；QDENY信号由器件给出，高有效，用于通知控制器当前不能门控关断所申请的时钟。具体时序如图10-6所示，在Q_ACCEPTED接口需要门控关断对应的工作时钟。

版权所有 © 海思技术有限公司

图 10-5 Q-Channel 接口

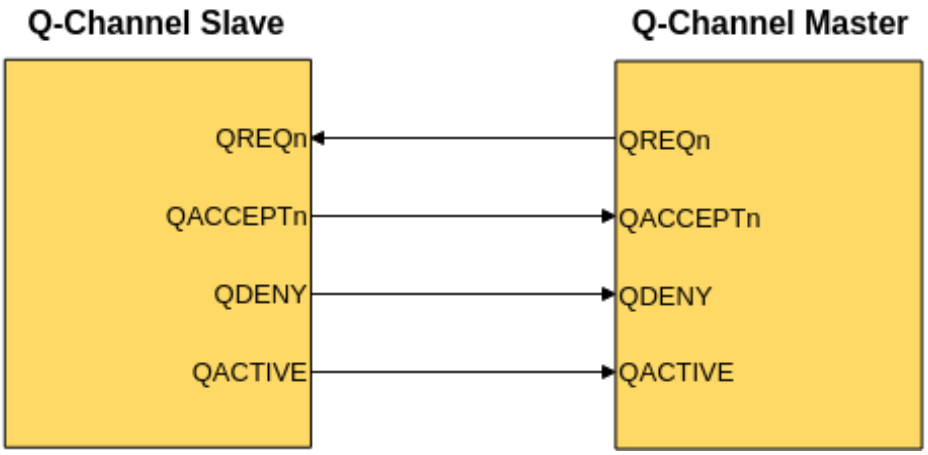
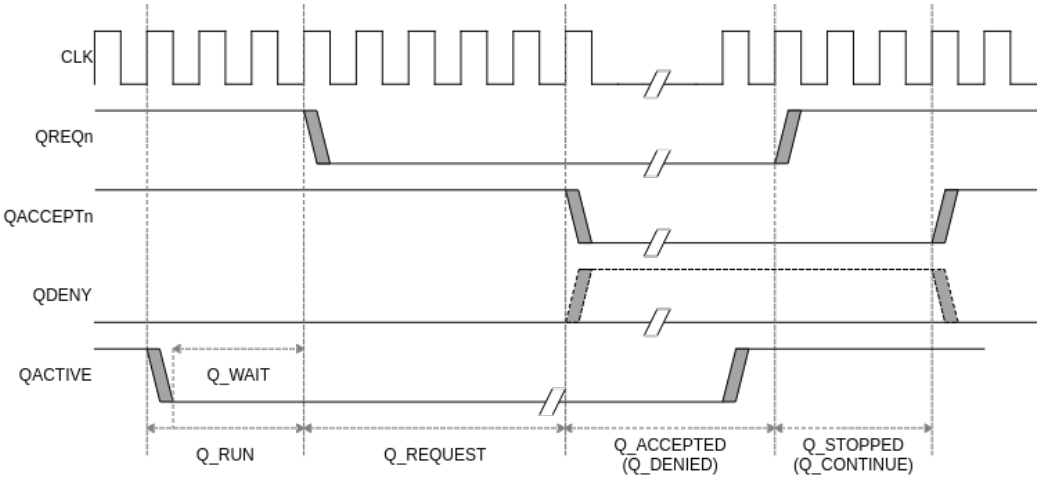


图 10-6 Q-Channel 接口时序图

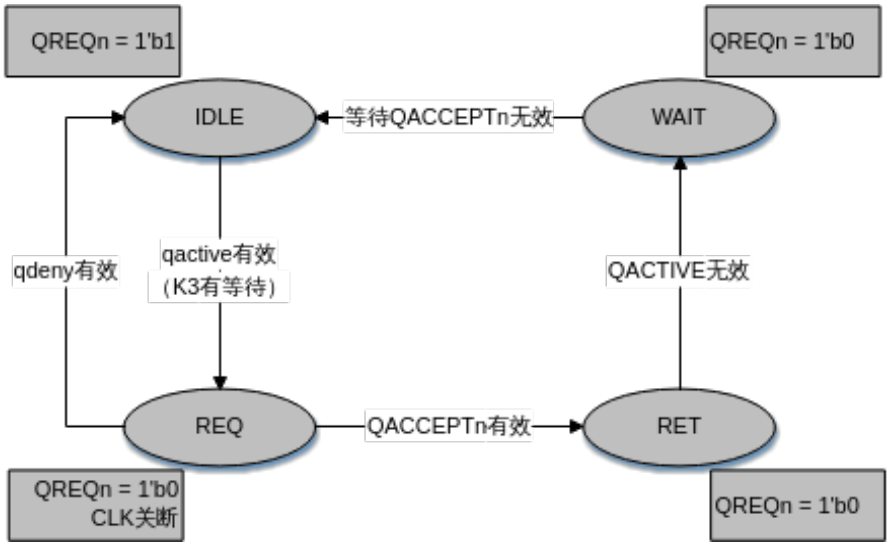


Q-Channel控制器主要实现的功能是接受QACTIVE指示，发起并完成Q-Channel握手实现相应时钟动态门控功能，所有的工程都是通过图10-7所示的Q-Channel控制器状态机控制，具体过程：

1. QACTIVE信号拉低。
2. Q-Channel控制器进入计数等待。
3. 计数超时，状态机进入REQ态并拉低QREQn，发出握手请求信号。
4. 器件拉低QACCEPTn信号表示可以时钟关断，状态机进入RET态并门控工作时钟。
5. QACTIVE信号拉高。
6. 状态机进入WAIT中间态。
7. 状态机返回IDLE态并拉高QREQn信号。

版权所有 © 海思技术有限公司

图 10-7 Q-Channel 控制器状态机



说明

- 模块的工作时钟为对应器件侧需要动态关断的时钟，复位为相应的复位。
- 解复位初始态，状态机上电解复位后为IDLE态，对应的输出QREQn为高，时钟门控打开。
- 按照ARM低功耗接口要求解复位完成后QACCEPTn和QDENY必须拉低。对应QACTIVE建议拉低。

10.5.4.2 P-Channel 控制器

P-Channel接口如图10-8所示，接口信号包括PACTIVE、PSTATE、PREQ、PACCEPT和PDENY。其中PACTIVE又器件给出，为多bit信号，用于通知控制器可以申请的状态；PSTATE由控制器给出，为多bit信号，用于指示器件控制器希望申请的状态；PREQ由控制器给出，高有效，为控制器向器件侧发出的握手请求信号；PACCEPT由器件给出，高有效，用于器件通知控制器器件侧接受控制器的功耗状态请求；PDENY由器件给出，高有效，用于器件拒绝控制器功耗状态请求。图10-9所示为P-Channel典型握手的时序图。

版权所有 © 海思技术有限公司

图 10-8 P-Channel 接口

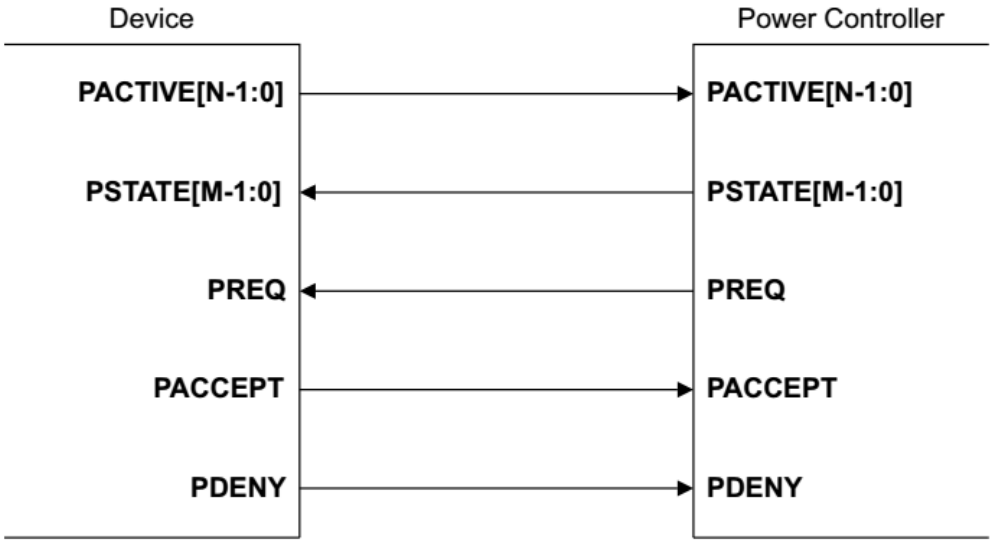
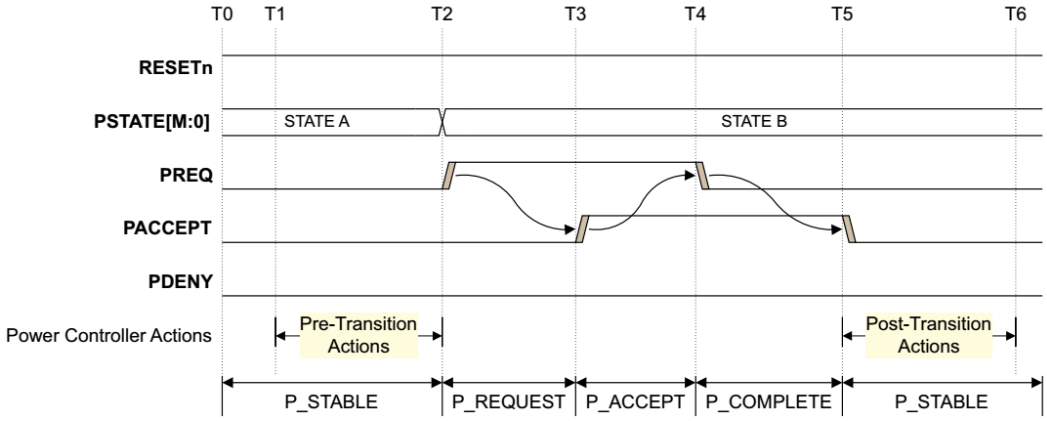


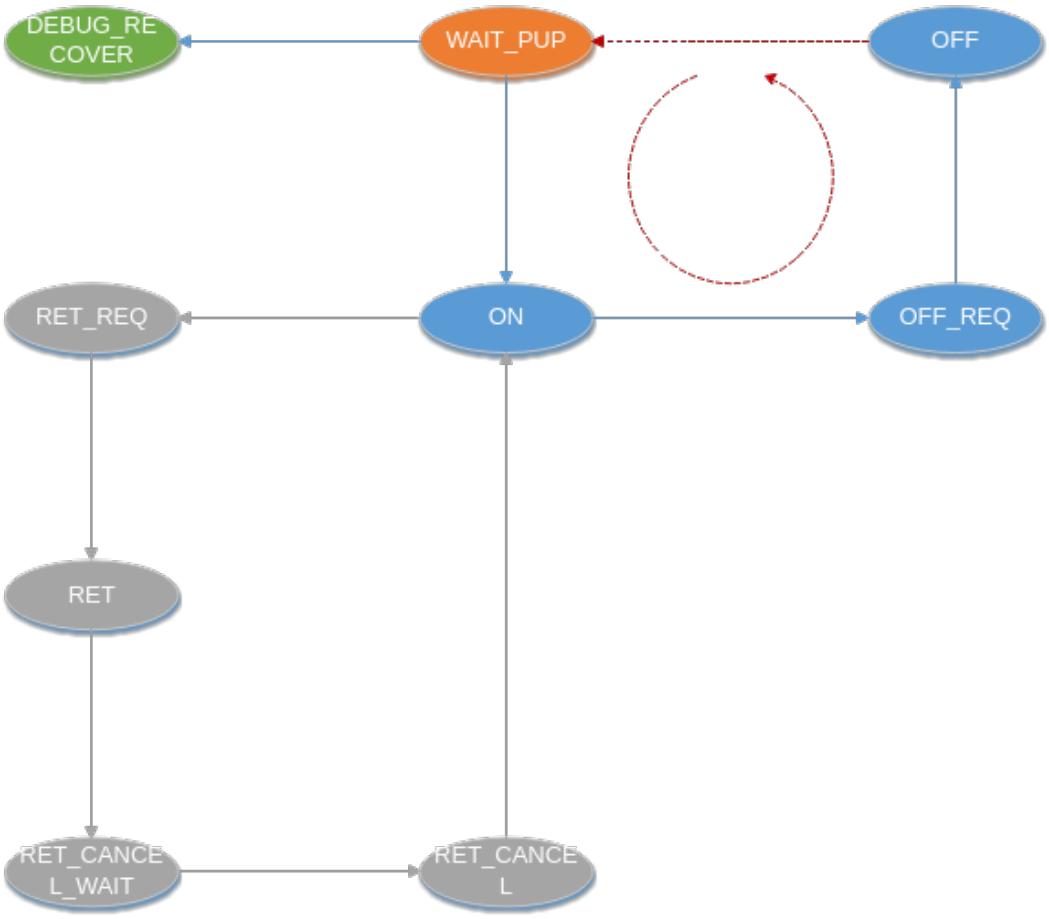
图 10-9 P-Channel 接口时序图



Core的P-Channel控制器的功能主要是根据Core所支持的功耗状态通过P-Channel接口握手实现Core不同功耗状态之间的切换。目前Hi309a项目A55核所支持的功能状态由DEBUG_RECOVER、ON、OFF态，对于RETENTION态目前没做规划由于代码本身支持，所以图10-10所示控制器状态机都有体现，其中灰色部分为目前不支持的状态跳转。对于中跳转条件为P-Channel协议规定的PACCEPT或者PDENY条件具体参看协议描述。

版权所有 © 海思技术有限公司

图 10-10 Core P-Channel 控制器状态跳转



控制器上电初始态为WAIT_PWRUP态，控制器会根据当前配置寄存器指示的状态申请跳转到DEBUG_RECOVER或者ON状态（对于上电解复位后的跳转不需要PACTIVE指示，由控制器主动发起握手请求并完成状态跳转）

说明

解复位各信号初始值：

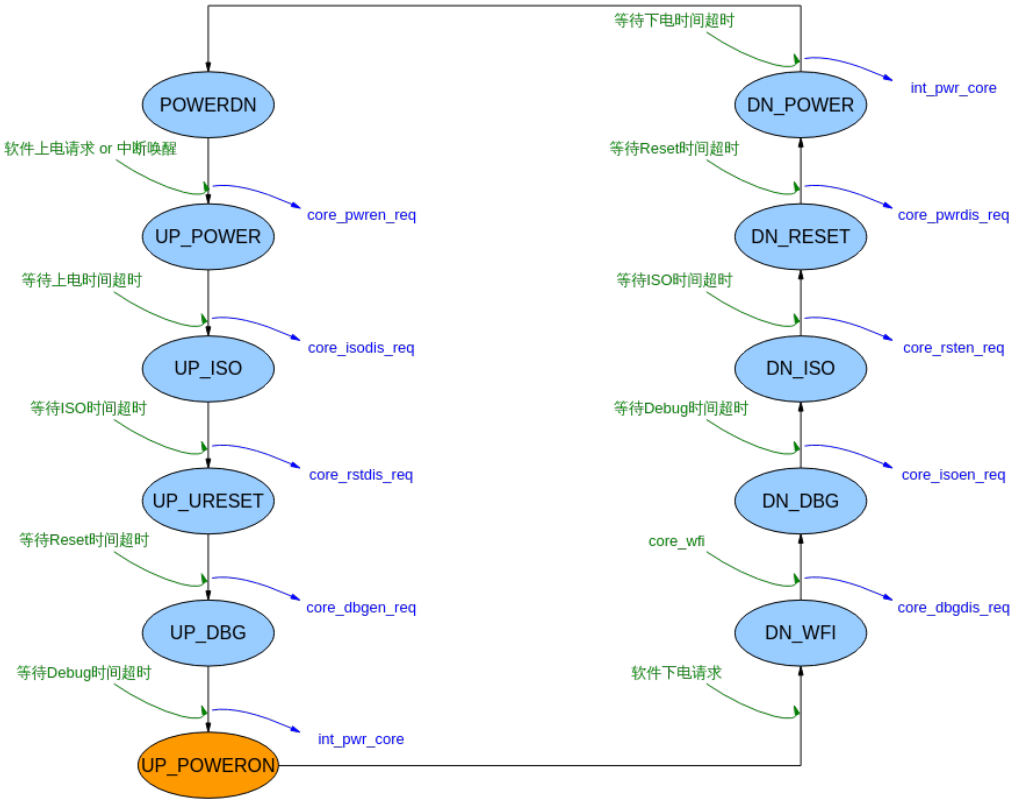
- 状态机 = WAIT_PWRUP
- Corepstate = ON（由控制器控制）
- Corepreq = 1（未使能，来自系统控制器）
- mem_ctrl_o = H00（未使能，受控系统控制器）
- Corepowerdown = 0（未使能为0，只有在状态机处于OFF态才会为1）
- Core_idle = 0（只有在状态机处于OFF或者RET态是才会为1）

10.5.4.3 CORE 上下电控制器

Core P-Channel控制器如果完成Core OFF状态的请求就会给出对应Core可下电指示信号，LPM3在监测到此信号拉高时，会通知对应CORE上下电状态机启动上下电流程。具体状态跳转如图10-11所示。

版权所有 © 海思技术有限公司

图 10-11 单 Core 上下电路流程



Hi309a每个A55 Core支持独立上下电并由片上MTCMOS控制。整个上下电流程受控于CORE_PDC控制器。下电过程在HISIA52监测到Core P-Channel状态机OFF状态给出下电请求启动，上电过程可以由硬件自动完成也可以由软件完成，硬件过程由对应Core的中断唤醒信号启动，软件过程由HISIA52给出上电请求启动。具体：

- 上电过程：

  上电请求或者中断请求有效

  a. 打开MTCMOS，对应Core上电（注意上电完成指示）
  b. 打开Isolation
  c. 解复位（同时需要注意此时需要复位P-Channel控制器）
  d. Core DBG使能
  e. 发送Core上电完成中断给HISI A52

- 下电过程：

  a. 下电请求有效
  b. 等待Core P-Channel握手完成并WFI指示有效
  c. 无效DBG
  d. 使能Isolation
  e. 复位有效
  f. 关闭MTCMOS，Core区下电

版权所有 © 海思技术有限公司

说明

- Core在系统上电初始为ON，系统根据需要对特定core完成下电
- 复位初始化PDC为ON态，P-Channel初始为WAIT_PWRUP态，解复位完成需要首先完成一次P-Channel握手，确定Core进入ON或者DEBUG_RECOVER态
- PDC工作时钟为PERIPHCLK，复位为对应的复位
- 解复位各信号初始值：
- 状态机 = ON
- corei_pwr_en = 1
- corei_iso_en = 0
- corei_rst_req = 0

10.5.5 IP 功耗

10.5.5.1 PLL

Hi309a PLL的功耗主要分步在四个部分：

- VCO，主要依赖于VCO的频率。
- PLL Core，主要依赖于PLL输出频率和Core的负载电容大小。
- Charge Pump，主要依赖于VDDHV电源的DC电流。
- Digital Power，数字逻辑的功耗。

PLL模块PD端口，提供PLL全局Power Down的功能。在这种低功耗模式下，所有模拟电路的电源关闭，PLL输出处于低电平。

10.5.5.2 eFuse

eFuse支持Power Down模式。eFuse控制器在上电后读出内部存储的值，然后自动将eFuse配置为Power Down模式。

10.5.5.3 TSensor

TSensor可设置为运行模式和停机模式，若TSesnor管脚TEMP_EN=0，TSensor处于停机模式：

- TEMP_EN=1，TSensor运行模式下：40℃，TSensor AVDD工作电流=500 μA，TSensor DVDD工作电流很小，不会超过500 μA。
- TEMP_EN=0，TSensor停机模式下：40℃，TSensor AVDD工作电流=10 μA，TSensor DVDD工作电流很小，不会超过500 μA。

下面是TSensor集成时需要关注以下几点：

1. TSensor的时钟：25MHz固定时钟。
2. 2个，具体位置待补充。
3. 温度采样和中断都在systermctrl中做。
4. TSENSOR的温度校准，eFuse需求有，这个最终会体现在eFuse那个表格上。

注： 第四点补充说明： 这个温度校准是在ATE环节做的，由自动流程直接烧eFuse。芯片应用时软件读这个eFuse值来使用。（ TSENSOR硬件不感知这个值 ）

版权所有 © 海思技术有限公司

10.5.5.4 DDR

DDR支持的低功耗特性：

- 支持DDR器件 Power Down，由软件通过配置寄存器使能。
- 支持DDR4复位信号可控，保证非掉电复位的状态下，RAM内的数据不丢失，即如果DDR器件在DDRC复位前进入自刷新状态，则DDRC复位后仍旧保证器件处在自刷新状态DDR器件不会被复位。

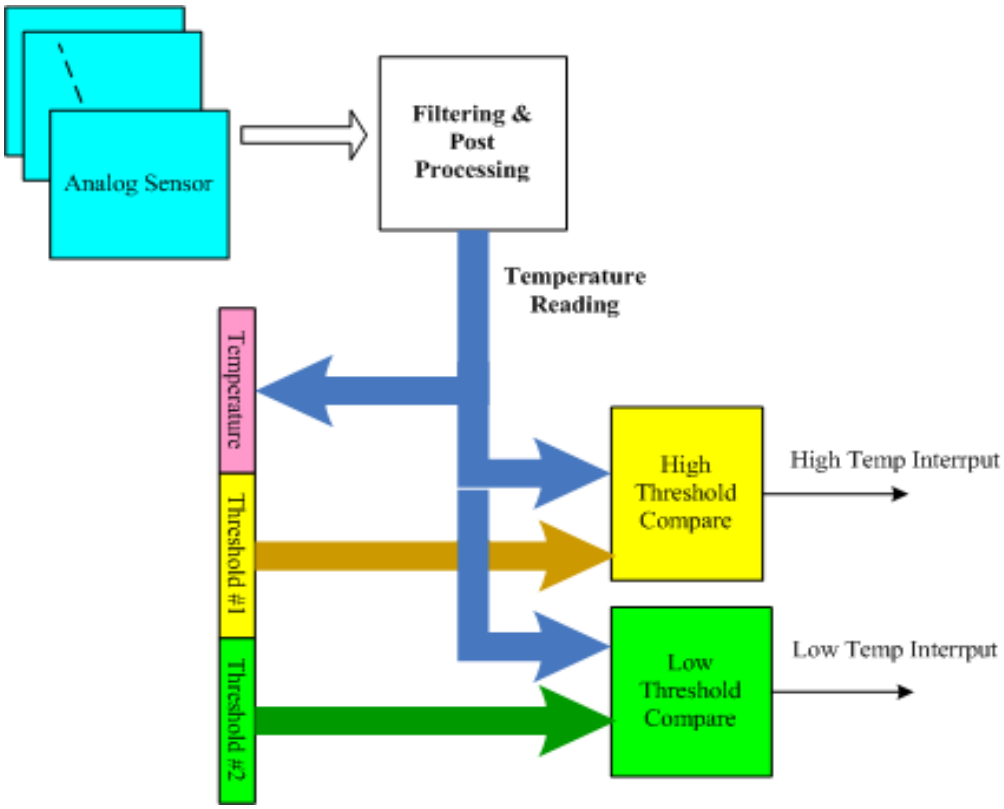
10.5.5.5 Hilink

集成了x个Hilink Macro，每个Hilink Macro都支持全局或者每Lane TX或者RX的Power Down功能。

10.5.6 温度控制

Hi309a分别集成了2个TSensor监控芯片的结温，系统可以实时采样芯片结温。软件设置TSensor的温度门槛，进行高温度预警。

图 10-12 温度控制



10.5.7 AVS

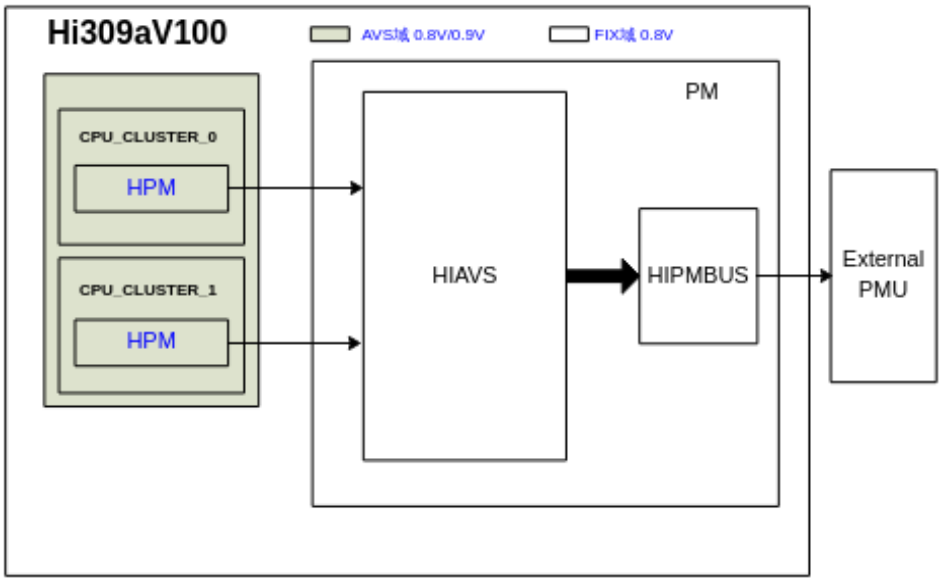
HIAVS，HPM和HIPMBUS构成了AVS方案的硬件系统。

Hi309a支持AVS静态工艺补偿和AVS动态温度补偿，通过微调电压来降低动态功耗和静态功耗。AVS静态工艺补偿和AVS动态温度补偿都依赖片上的HPM模块，通过HPM模块的延迟反应芯片的工艺分布和温度变化。

版权所有 © 海思技术有限公司

如图10-13所示，Hi309a集成了2组HPM模块，位于CPU Cluster模块中。HIAVS用于HPM使能和数据处理，并生成调压指令；HIPMBUS支持PMBus1.2接口，接受HIAVS发出的调压指令，通过PMBus接口对外部PMU芯片执行调压和监控操作。

图 10-13 Hi309a AVS 方案图



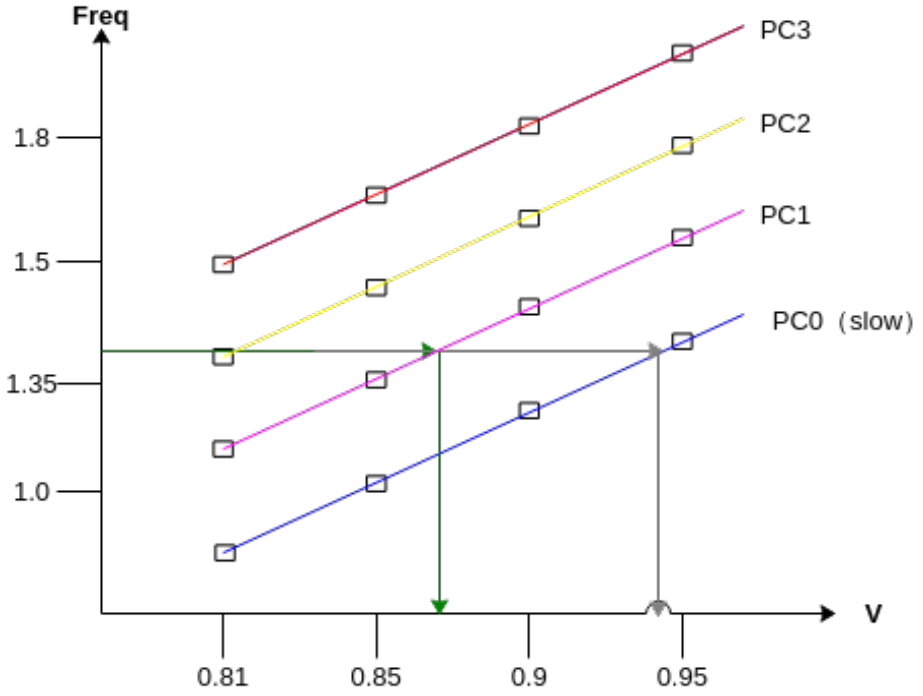
10.5.7.1 AVS 静态工艺补偿 AVS0

AVS静态工艺补偿AVS0，即支持为不同PC的芯片选择不同的默认工作电压来降低芯片的功耗。如图10-14所示，归属不同PC的芯片在相同的工作频率下可以工作在不同的电压下，快PC芯片相对慢PC芯片可以工作在更低的电压下。调整的精度受限于外部电源管理芯片每步调压的精度和芯片能够划分的PC粒度。

静态补偿主要是通过片上集成的HPM和片上eFuse完成，在生产测试阶段确定芯片所属的PC，确定最低的工作电压，烧写入eFuse。芯片启动时BIOS根据eFuse烧写的信息配置芯片的工作电压。

版权所有 © 海思技术有限公司

图 10-14 不同 PC 芯片电压-频率对应图



10.6 Initialization Example

10.6.1 AVS 应用流程

Hi309a的软件调压和AVS3动态温度补偿功能,默认都是有CPU和HIAVS控制器配置电源接口完成。软件调压和AVS动态温度补偿都需要初始化HIAVS控制器，PMBus接口和外部PMU器件。初始化默认PLL处于Lock。

如下寄存器如果没有特别说明，均指AVS寄存器。

10.6.2 PMU 芯片初始化

说明

PMU的初始化流程与单板使用的电源芯片相关，不同的电源芯片使用的初始化流程会有差别，请以PMU datasheet 以及使用场景要求为准。请根据实际情况选择合适的初始化流程。

单板应用中，电源芯片的地址会有差别。

所有对PMU的操作都是通过PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD进行的（偏移地址：0x010）。

下面以slave_addr代表电源芯片的地址。

版权所有 © 海思技术有限公司

10.6.2.1 TPS544C24

表 10-2 TPS544C24 配置表

TPS544C24 2.0芯片								
PMBUS地址	**0x24**							
说明	**PMBUS命令**	**命令名称**	**寄存器比特数**	**默认值**	**配置值**			**备注**
-	-	-	-	-	SD6182	Hi1381	SD6602	-
解保护	10h	WRITE_PROTECT	8bit	0x00 不保护	0x00 不保护	0x00 不保护	0x00 不保护	-
输出电压最小值 限制	A4h	MFT_VOUT_MIN	16bit	0. 5v	Vmin EFUSE*0.95	Vmin EFUSE*0.95	Vmin EFUSE*0.95	按照Vmin_EFUSE值的95%设置，如果海思暂无Vmin值，该值可用0x0171（0.72V）
输出电压最大值 限制。（2.0芯片）	24h	VOUT_MAX	16bit	1. 5v	0x0233 (1.1V)	0x0233 (1.1V)	0x0233 (1.1V)	-
过压故障门限	40h	VOUT_OV_FAULT_LIMIT	16bit	0x288 (1.265V)	0x288 (1.265V)	0x288 (1.265V)	0x288 (1.265V)	1.1*1.15=1.265V
过压告警门限	42h	VOUT_OV_WARN_LIMIT	16bit	0x26C (1.21V)	0x26C (1.21V)	0x26C (1.21V)	0x26C (1.21V)	1.1*1.1=1.21V

版权所有 © 海思技术有限公司

TPS544C24 2.0芯片								
PMBUS地址	0x24							
说明	PMBUS命令	命令名称	寄存器比特数	默认值	配置值			备注
输出欠压保护关断门限	44h	VOUT_UV_FAULT_LIMIT	16bit	0x139（0.612V）	Vmin_EFUSE*0.85	Vmin_EFUSE*0.85	Vmin_EFUSE*0.85	按照Vmin_EFUSE值的85%设置，如果海思未提供Vmin值，先按照0x0157（0.67V）配置
输出欠压保护告警门限	43h	VOUT_UV_WARN_LIMIT	16bit	0x14C（0.648V）	Vmin_EFUSE*0.90	Vmin_EFUSE*0.90	Vmin_EFUSE*0.90	按照Vmin_EFUSE值的90%设置，如果海思未提供Vmin值，先按照0x0171（0.72V）配置
过流告警门限设置	4Ah	IOUT_OC_WARN_LIMIT	16bit	34A	default	default	0xF81D（14.5A）	-
过流故障门限设置	46h	IOUT_OC_FAULT_LIMIT	16bit	36A	default	default	0xF81E（15A）	-

版权所有 © 海思技术有限公司

TPS544C24 2.0芯片								
PMBUS地址	0x24							
说明	PMBUS命令	命令名称	寄存器比特数	默认值	配置值			备注
调压斜率设置1.0的样片默认值为0.067 mv/μs	27h	VOUT_TRANSITION_RATE	16bit	Imv/μs	0xC006 (0. mv/μs)	0xC006 (0. mv/μs)	0xC006 (0. mv/μs)	-
修改PMU采样模式，只采样电压值。	E5h	MFR_SPECIFIC_21	16bit	00C7h	-	-	00CFH	-
清楚错误	03h	CLEAR_FAULTS	-	-	-	-	-	-
写保护	10h	WRITE_PROTECT	8bit	0x00 不保护	0x20 写保护	0x20 写保护	0x20 写保护	-
说明：配置参数有顺序，按照从上道下的顺序配置。灰色为默认值满足要求，不需要设置的。								

步骤1 配置寄存器PMBUS_WR_OPEN为0x5A5A_5A5A完成对PMBUS_PROC寄存器的写保护解锁，默认为锁定状态；

步骤2 配置寄存器I2C_LOCK为0x5A5A_5A5A完成对PMBUS接口模块寄存器的写保护解锁，默认为解锁状态；

步骤3 PMU寄存器解保护。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x10和0x400。

步骤4 配置PMU输出电压最小值MFR_VOUT_MIN为0.72V（0.67V对应于PMU配置电压编码为0x171）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0xA4、0x71和0x401。

步骤5 配置PMU输出电压最大值VOUT_MAX为1.1 V（1.1V对应于PMU配置电压编码为0x233）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x24、0x33和0x402。

步骤6 配置PMU过压故障门限VOUT_OV_FAULT_LIMIT为1.265 V（1.265V对应于PMU配置电压编码为0x288）。

版权所有 © 海思技术有限公司

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x40、0x88和0x402。

步骤7 配置PMU过压告警门限VOUT_OV_WARN_LIMIT为1.21 V（1.21V对应于PMU配置电压编码为0x26C）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x42、0x6C和0x402。

步骤8 配置PMU输出欠压保护关断门限VOUT_UV_FAULT_LIMIT为0.67V（0.67V对应于PMU配置电压编码为0x157）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x44，0x57和0x401。

步骤9 配置PMU输出欠压保护告警门限VOUT_UV_WARN_LIMIT为0.72V（0.72V对应于PMU配置电压编码为0x171）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x43，0x72和0x401。

步骤10 配置PMU器件调压斜率为0.1mV/μs（目的：减小输出电压的过冲）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x27，0x06，和0xD0。

步骤11 清除PMU器件的异常状态（在PMU设置斜率后，保护PMU前，清除PMU异常信息，目的是清除PMU器件的异常状态，包括ALERT信号，以免对后续调压状态的判断产生影响。）

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}和0x403。

步骤12 修改PMU配置，只保留PMU对电压的采样（目的：提高电压调整速度，由默认的56ms提高到11ms，针对TPS544C24）

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0xE5，0xCF和0x400。

步骤13 打开PMU寄存器保护功能（在首次解保护PMU，配置完PMU寄存器后，再将PMU进行保护，防止软件再次修改，产品线通常都这样操作。）

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x10和0x420。

----结束

10.6.2.2 VT7410

表 10-3 VT7410 配置表

VT7410							
PMBUS地址	**0x57**						
PMBUS命令地		**命令名称**	**寄存器比特数**	**默认值**	**配置值**		**备注**
-	-	-	-	-	Hi1381	SD6602	-
解保护	10h	WRITE_PROTECT	6bit	20h（写保护）	0h（解保护）	0h（解保护）	-

版权所有 © 海思技术有限公司

VT7410							
PMBUS地址	0x57						
PMBUS命令地		命令名称	寄存器比特数	默认值	配置值		备注
输出电压最小值限制	D1h	MFT_VOUT_MIN	16bit	-	Vmin_EFUSE*0.95	Vmin_EFUSE*0.95	如果暂无Vmin值，可先用0x0171（0.72V）值配置
输出电压最大值限制	24h	VOUT_MAX	16bit	-	0x0233（1.1V）	0x0233（1.1V）	-
过流故障门限设置	D2h[7:5]	MFR_DEVSET1	8bit	35A	default	00（20A）	-
-	10h	WRITE_PROTECT	8bit	-	20h（写保护）	20h（写保护）	-

步骤1 配置寄存器PMBUS_WR_OPEN为0x5A5A_5A5A完成对PMBUS_PROC寄存器的写保护解锁，默认为锁定状态。

步骤2 配置寄存器I2C_LOCK为0x5A5A_5A5A完成对PMBUS接口模块寄存器的写保护解锁，默认为解锁状态。

步骤3 PMU寄存器解保护。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x10和0x400。

步骤4 配置PMU输出电压最小值MFR_VOUT_MIN为0.72V（0.67V对应于PMU配置电压编码为0x171）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0xA4、0x71和0x401。

步骤5 配置PMU输出电压最大值VOUT_MAX为1.1 V（1.1V对应于PMU配置电压编码为0x233）。

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x24、0x33和0x402。

步骤6 打开PMU寄存器保护功能（在首次解保护PMU，配置完PMU寄存器后，再将PMU进行保护，防止软件再次修改，产品线通常都这样操作。）

依次向I2C_DATA_CMD寄存器写{3h2,slave_addr}，0x10和0x420。

----结束

版权所有 © 海思技术有限公司

10.6.3 PMBus 初始化

步骤1 配置寄存器AVS_WR_OPEN完成对AVS全局参数寄存器的写保护解锁，默认为锁定状态。

Bits	寄存器说明	配置值
AVS_WR_OPEN：偏移地址0x004		
[31:0]	写寄存器： 向本寄存器地址写入指定值AVS_WR_OPEN（默认为0x5A5A_5A5A，该宏可修改）后打开写使能；向本寄存器地址写入非指定值后则锁定写使能。保护地址范围为：13'h030~13'h7FF。 读寄存器： 寄存器读出值表示寄存器的锁定状态： 0x0000_0000表示寄存器处于锁定状态； 0x0000_0001表示寄存器处于解锁状态。	0x5A5A_5A5A（解锁） 0x0000_0000（锁定）

步骤2 配置寄存器PMBUS_WR_OPEN完成对PMBUS_PROC寄存器的写保护解锁，默认为锁定状态。

Bits	寄存器说明	配置值
PMBUS_WR_OPEN：偏移地址0xA04		
[31:0]	写寄存器： 向本寄存器地址写入指定值PMBUS_WR_OPEN（默认为0x5A5A_5A5A，该宏可修改）后打开写使能；向本寄存器地址写入非指定值后则锁定写使能。保护地址范围为：13'h800~13'hEFF。 读寄存器： 寄存器读出值表示寄存器的锁定状态： 0x0000_0000表示寄存器处于锁定状态； 0x0000_0001表示寄存器处于解锁状态。	0x5A5A_5A5A（解锁） 0x0000_0000（锁定）

步骤3 配置寄存器I2C_LOCK完成对PMBUS接口模块寄存器的写保护解锁，默认为解锁状态。

Bits	寄存器说明	配置值
I2C_LOCK：偏移地址0x8AC		

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[31:0]	写寄存器： 向Lock寄存器写32bits密钥（密钥宏定义配置，请查看宏定义文件），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。 读寄存器： Lock寄存器读出值表示寄存器的锁定状态： 32h0000_0000表示寄存器处于解锁状态； 32h0000_0001表示寄存器处于锁定状态。	0x5A5A_5A5A（解锁） 0x0000_0000（锁定）

步骤4 确认硬件调压关闭TRIGGER_CFG为0x0。

Bits	寄存器说明	配置值
AVS_REG_GEN.TRIGGER_CFG：偏移地址0x0B8		
[31:17]	保留	-
[16]	AVS调整启动信号，写1产生脉冲信号。	0x0
[15:1]	保留	-
[0]	DVFS调整启动信号，写1产生脉冲信号。	0x0

步骤5 软件配置I2C工作使能寄存器，去使能PMBus。

Bits	寄存器说明	配置值
PMBUSIF_REG_GEN.I2C_ENABLE：偏移地址0x06C		
[31:1]	保留	-
[0]	I2C使能。	0x0

步骤6 设置PMBus为快速模式。

Bits	寄存器说明	配置值
PMBUSIF_REG_GEN.I2C_CON：偏移地址0x0		
[31:7]	保留	-
[6]	I2C从模式使能	0x0
[5]	主模式下Restart指令发送使能	0x1
[4:3]	保留	-
[2:1]	速度选择。	0x2。
[0]	I2C主模式使能	0x1

版权所有 © 海思技术有限公司

步骤7 配置I2C的SCL高电平和低电平配置寄存器，需要根据PMBus的速度模式配置不同的寄存器。

Bits	寄存器说明	配置值
PMBUSIF_REG_GEN .I2C_FS_SCL_LCNT：偏移地址0x020		
[31:16]	保留	-
[15:0]	设置快速模式下SCL时钟低电平时间。 要求： 快速模式下SCL输出低电平最小时间为1.3μs。	HIAVS工作在200MHz，配置为400;

Bits	寄存器说明	配置值
PMBUSIF_REG_GEN .I2C_FS_SCL_HCNT：偏移地址0x01C		
[31:16]	保留	-
[15:0]	设置快速模式下SCL时钟高电平周期数。 要求： 快速模式下SCL输出高电平时间最小为0.6μs。	HIAVS工作在200MHz，配置为200;

步骤8 配置I2C中断屏蔽寄存器I2C_INTR_MASK，使能中断，当相应的bit为配置为0b1时屏蔽，配置为0b0时不屏蔽该中断。

Bits	寄存器说明	配置值
I2C_INTR_MASK：偏移地址0x830		
[31:18]	保留。	-
[17]	master 一次命令传输结束（包括异常结束）中断屏蔽。 0：不屏蔽; 1：屏蔽。	-
[16]	SCL高电平超时中断屏蔽。 0：不屏蔽; 1：屏蔽。	-
[15]	SCL低电平超时中断屏蔽。 0：不屏蔽; 1：屏蔽。	-

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[14]	Slave接收地址信息中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[13]	Slave接收命令、数据信息中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[12]	alert_detect中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[11]	一次general call请求被接收的中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[10]	start detect中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[9]	stop detect中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[8]	activity中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[7]	接收完成完成中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[6]	发送中止中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[5]	接收请求中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[4]	TX_FIFO到达或者低于阈值中断屏蔽。 0：不屏蔽； 1：屏蔽。	-

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[3]	TX_FIFO溢出中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[2]	RX_FIFO达到或超过阈值中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[1]	RX_FIFO溢出中断屏蔽。 0：不屏蔽； 1：屏蔽。	-
[0]	读数据溢出（即RX_FIFO为空时CPU对FIFO进行读取）中断屏蔽。 0：不屏蔽； 1：屏蔽。	-

步骤9 配置I2C SDA保持时间配置寄存器I2C_SDA_HOLD。

Bits	寄存器说明	配置值
PMBUSIF_REG_GEN .I2C_SDA_HOLD：偏移地址0x07C		
[31:16]	保留。	-
[15:0]	标准模式和快速模式的SDA保持时间设置，以I2C工作频率i2c_clk为单位。 说明： 该值为SCL下降沿到SDA变化的延时计数值，即当SCL下降沿以后sda_hold_fs个i2c_clk周期数据线SDA才能发生变化。	200

步骤10 配置I2C low timeout 低电平超时寄存器I2C_LOW_TIMEOUT。

Bits	寄存器说明	配置值
PMBUSIF_REG_GEN .I2C_LOW_TIMEOUT：偏移地址0x108		
[31:0]	PMBus的SCL低电平超时值（PMBus协议规定为25~35ms）配置寄存器，单位为PMBus的工作时钟cycle，需要根据PMBus的工作频率配置。	0x5B8D80

步骤11 配置I2C SCL高电平空闲值配置寄存器I2C_PMBUS_IDCNT。

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
PMBUSIF_REG_GEN .I2C_ PMBUS_IDCNT：偏移地址0x130		
[31:15]	保留。	-
[14:0]	PMBus的SCL高电平空闲值（（PMBus协议规定为50us））配置寄存器，单位为PMBus的工作时钟cycle，需要根据PMBus的工作频率配置。	0x2710

步骤12 配置PMBUSIF_REG_GEN SCL检测寄存器I2C_PMBUS_SCL_DET，使能I2C低电平超时检测。

Bits	寄存器说明	配置值
I2C_PMBUS_SCL_DET：偏移地址0x92C		
[31:2]	保留。	-
[1]	SCL低电平超时检测使能寄存器。 0：超时检测不使能； 1：超时检测使能。 **注意：** **只有在I2C接口禁止（I2C_ENABLE=0）时才可以进行写操作。**	0x1
[0]	SCL高电平空闲检测使能寄存器。 0：空闲检测不使能； 1：空闲检测使能。 **注意：** 1. **master模式，如果作为正常的I2C通信，为匹配I2C协议，不能使能SCL空闲检测。slave模式使能空闲检测无效** 2. **只有在I2C接口禁止（I2C_ENABLE=0）时才可以进行写操作。**	0x1

步骤13 配置PMBUSIF_REG_GEN PMBUS控制寄存器I2C_PMBUS_CTRL为0x6，打开PEC和alert功能。

Bits	寄存器说明	配置值
I2C_PMBUS_CTRL：偏移地址0x104		
[31:3]	保留。	-
[2]	PEC功能使能。	0x1
[1]	Alert功能使能。	0x1
[0]	保留。	-

版权所有 © 海思技术有限公司

步骤14 配置PMBUSIF_REG_GEN I2C工作使能寄存器I2C_ENABLE，使能PMBUS。

Bits	寄存器说明	配置值
I2C_ENABLE：偏移地址0x86C		
[31:1]	保留。	-
[0]	I2C使能。 0：禁止； 1：使能。	0x1

步骤15 配置PMBUS其他相关配置寄存器PMBUS_OTHER_CFG。

Bits	寄存器说明	配置值
PMBUS_OTHER_CFG：偏移地址0xA2C		
[31:25]	保留	-
[24]	pmbus_proc硬件使能。 0：不使能； 1：使能。	0x1
[23:21]	保留	-
[20]	pmbus硬件调压超时检测使能。 0：不使能； 1：使能。	0x1
[19:17]	保留	-
[16]	调压流程中是否发送READ_VOUT命令。 0：不发送； 1：发送。 **注意：若本寄存器配置为0，则表示通过计数来保证电压调整完成，因此软件必须保证的值足够大。**	0x1
[15:13]	保留	-
[12]	调压流程中是否发送PAGE命令。 0：不发送； 1：发送。 **注意：若对接的电源芯片不支持PAGE命令，则本寄存器必须配置为0**	0x0
[11:10]	保留	-

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[9:8]	PMBus处理模块使用READ_VOUT命令时，求取平均采用的回读电压的次数。 00：回读4次做平均； 01：回读8次做平均； 10：回读16次做平均； 11：回读32次做平均。	0x0
[7:0]	PMBus 处理模块发送命令错误时，重新发送命令的次数。	0x0

步骤16 配置电压域相关的地址配置寄存器VOLTAGE_ADDR_CFG。

Bits	寄存器说明	配置值
VOLTAGE_ADDR_CFG：偏移地址0xA30		
[31:24]	保留。	-
[23:16]	READ_VOUT回读电压时，允许的噪声容限值，需要结合电源芯片的电压测试误差值计算得出。	根据电源芯片稳态输出时波动V_{noise}和电源芯片N值决定（V_{noise}/（2^N*1000））以20Mv和N值为-9为例，值为0xA
[15:8]	电压域对应PAGE地址。	0x0
[7]	保留。	-
[6:0]	电压域对应电源器件地址。	配置为PMBUS接口挂接的PMU芯片的地址。

步骤17 配置电压域电压值转换相关寄存器VOLTAGE_CONVERT_CFG。

Bits	寄存器说明	配置值
VOLTAGE_CONVERT_CFG：偏移地址0xA34		
[31:24]	保留。	-
[23:8]	电压偏移值，用于将8bit电压值转成PMBus电压的16bit电压值。	0x0
[7:0]	电压最小步长值，用于将8bit电压值转成PMBus电压的16bit。	由步进值大小Vstep（单位mv）和电源芯片N值决定（Vstep/（2^N*1000）），以步进6.25Mv和N值-9为例，配置值为0x3

版权所有 © 海思技术有限公司

步骤18 配置PMBUS电压稳定时延寄存器PMBUS_VOLTAGE_STABLE。

Bits	寄存器说明	配置值
PMBUS_VOLTAGE_STABLE：偏移地址0xA20		
[31:24]	保留。	-
[23:0]	PMBUS 电压调整稳定门限值，对应的最大值为33.5ms@500MHz。在发送完调整电压命令后，等待本寄存器所配置的时延后，才发送READ_VOUT命令（使能READ_VOUT命令）。	0x2DC6C0（15ms@200MHz）

步骤19 配置重新发送READ_VOUT间隔寄存器READ_VOUT_INTERVAL。

Bits	寄存器说明	配置值
READ_VOUT_INTERVAL：偏移地址0xA28		
[31:20]	保留。	-
[19:0]	发送READ_VOUT命令的时间间隔，也即做读取电压操作的间隔，对应的最大值为2.0972ms@500MHz，该值必须大于电源器件的Output voltage monitoring interval值。	0x30D40（1ms@200MHz）

步骤20 配置PMBUS硬件调压超时门限值寄存器PMBUS_PROC_TIMEOUT_TH。

Bits	寄存器说明	配置值
PMBUS_PROC_TIMEOUT_TH：偏移地址0xA1C		
[31:26]	保留。	-
[25:0]	PMBus 硬件电压调整处理超时门限值，对应的最大值为134ms@500MHz。	0x3FFFFFF（335ms@200MHz）

----结束

10.6.4 PMBus 软件调压

确认PMBus和PMU初始化完成。

步骤1 查询PMBUS_PROC状态上报相关寄存器STATUS_RPT，确认PMBUS_PROC硬件逻辑是否空闲，bit[25:21]值为0表示硬件逻辑当前空闲。

步骤2 根据eFuse信息，通过查找表获取芯片的初始电压信息。

步骤3 配置PMU为期望的电压值exp_vol，PMU的16-bit配置值Vol[15:0] = （exp_vol*512）/10000，其中exp_vol的单位是0.1mV。具体步骤：

1. 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为，打开写使能，假定PMU器件地址为0x24。
2. 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为，启动调压操作。

版权所有 © 海思技术有限公司

3. 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD0xVol[7:0]，配置为电压的低电压值Vol[7:0]。
4. 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为，配置为电压的高8-bit电压值Vol[15:8]。

步骤4 软件查询I2C原始中断状态寄存器I2C_INTR_RAW，确认当前命令正确执行完成。

Bits	寄存器说明	查询值
I2C_INTR_RAW：偏移地址0x834		
[31:18]	保留。	-
[17]	master 一次命令传输结束（包括异常结束）中断。 0：无中断； 1：一次命令传输结束。	0x1
[16]	保留	-
[15]	SCL低电平超时中断状态。 0：无中断； 1：SCL低电平值超过了I2C_LOW_TIMEOUT中i2c_low_timeout值。	0x0
[14:13]	保留。	-
[12]	Master或Slave检测到alert产生的中断状态。 0：无中断； 1：Master或Slave检测到了alert条件。	0x0
[11]	保留。	-
[10]	start detect中断状态，指示I2C总线接口是否有START或RESTART条件。 0：无中断； 1：Master或Slave检测到了START或RESTART条件。	x
[9]	stop detect中断状态，指示I2C总线接口是否有STOP条件。 0：无中断； 1：Master或Slave检测到了STOP条件。	x
[8]	activity中断状态，记录I2C活动状态，直至被清除。 0：Master或者Slave处于闲状态； 1：Master或者Slave处于忙状态。	x
[7]	保留。	-

版权所有 © 海思技术有限公司

Bits	寄存器说明	查询值
[6]	发送中止中断状态。 0：无中断； 1：有中断。	0x0
[5]	保留。	-
[4]	TX_FIFO到达或者低于阈值中断状态。 0：无中断； 1：有中断。	x
[3]	TX_FIFO溢出中断状态。 0：无中断； 1：有中断	x
[2]	RX_FIFO到达或者低于阈值中断状态。 0：无中断； 1：有中断。	x
[1]	RX_FIFO溢出中断状态。 0：无中断； 1：有中断。	x
[0]	读数据溢出（即RX_FIFO为空时CPU对FIFO进行读取）中断状态。 0：无中断； 1：有中断。	x

步骤5 本次发送命令为写命令时，配置I2C原始中断状态寄存器I2C_INTR_RAW，配置值为0xFFFFFFF，清除所有中断结束本次访问；本次发送命令为读命令时，读取I2C数据操作寄存器I2C_DATA_CMD获取返回数据后清除所有中断结束本次访问。

----结束

10.6.5 PMBus 软件读电压流程

读电压流程如下：

步骤1 PMBus初始化完成。

步骤2 在软件读取电压前，通过读取PMBUSIF_REG_GEN I2C RX_FIFO有效数据指示寄存器I2C_RXFLR，确认I2C RX FIFO为空。

步骤3 PMBus读取电压。

- 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为0x224，打开写使能，假定PMU器件地址为0x24。
- 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为0x8B，启动PMU输出电压读取操作。
- 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为0x324，从PMU读取电压，假定PMU器件地址为0x24。

版权所有 © 海思技术有限公司

- 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为0x100，读取电压的低8bit电压数据。
- 配置PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD为0x500，读取电压的高8bit电压数据。

步骤4 读取PMBUSIF_REG_GEN I2C RX_FIFO有效数据指示寄存器PMBUSIF_REG_GEN I2C_RXFLR和PMBUSIF_REG_GEN I2C屏蔽后中断状态寄存器I2C_INTR_STAT（pmbus_cmd_finish，scl_low_tout，alert_det和start_det域段），如果分别为0x2和0x20000，则表示电压读取正常；否则，电压读取失败；连续两次读取PMBUSIF_REG_GEN I2C数据操作寄存器I2C_DATA_CMD的低8bit数据，获取电压的16bit数值Vol，则芯片输出的实际电压为（Vol*1000/512）mV。

----结束

10.6.6 AVS 动态温度补偿配置流程

确认PMBus和PMU已经初始化，且AVS3功能关闭ENABLE_AND_DVFS_VOL_LMT.avs_en 为0x0。

配置步骤如下：

步骤1 配置PMU相关寄存器PMU_CFG，设置PMU电压掉电时对应的VID值和PMU电压调整延时稳定计数值。

Bits	寄存器说明	配置值
PMU_CFG：偏移地址0x040		
[31:24]	PMU电压掉电对应的VID值。 **注意：具体数值必须与所使用的供电器件匹配，否则可能导致芯片烧坏。**	0x0；
[23:0]	PMU电压调整延时稳定计数值，最大等待稳定时延为84ms@200MHz。 **注意：具体数值必须与所使用的供电器件匹配，否则可能导致芯片无法工作。硬件对最小值做了处理，若配置值小于24'd10时，则逻辑内部将使用24'd10做为稳定计数值，否则将采用实际配置的值。当为VID6格式时，每次更新VID时，输出到PAD上的bit4（即为VID_S,select line）持续为低电平的时间为稳定计数值的一半。**	根据使用PMU参数配置

步骤2 配置使能和DVFS配置的电压门限寄存器ENABLE_AND_DVFS_VOL_LMT。

Bits	寄存器说明	配置值
AVS_REG_GEN. ENABLE_AND_DVFS_VOL_LMT：偏移地址0x044		
[31:29]	保留。	-
[28]	AVS使能。	0b1
[27:25]	保留。	-
[24]	DVFS触发AVS使能。	0b0

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[23:21]	保留。	-
[20]	DVFS使能。	0b0
[19]	保留。	-
[18]	VID调整方向。	0b0
[17]	保留。	-
[16]	电源接口为VID模式时。	0b0
[15:8]	DVFS输出的电压上限对应的VID值。	根据电源芯片参数配置
[7:0]	DVFS输出的电压下限对应的VID值。	根据电源芯片参数配置

步骤3 配置AVS调整稳定后检测间隔寄存器AVS_DONE_INTERVAL。

Bits	寄存器说明	配置值
AVS_DONE_INTERVAL：偏移地址0x050		
[31:20]	保留。	-
[19:0]	AVS调整到HPM的输出在噪声范围之内后HPM检测间隔周期，以AVS工作时钟周期为单位，最大为5.243ms@200MHz。	根据测试数据给出

步骤4 配置AVS检测相关寄存器AVS_CHECK_CFG。

Bits	寄存器说明	配置值
AVS_REG_GEN. AVS_CHECK_CFG：偏移地址0x04C		
[31:29]	保留。	-
[28]	HPM取值方式。	0b1
[27:8]	AVS调整过程中HPM检测间隔周期，以AVS工作时钟周期为单位，最大为2.0972ms@500MHz。	3000，根据测试数据给出
[7:5]	保留。	-
[4]	AVS计算调整旁路。	0b0
[3:0]	AVS调整过程中每次调整需对HPM采样的次数。	0b0，根据测试数据给出

步骤5 配置AVS其他配置寄存器AVS_MISCELL。

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
AVS_REG_GEN. AVS_MISCELL：偏移地址0x054		
[31:28]	AVS稳态增益值。即对应于电压小幅度调整时的步长。	0x1
[27:24]	AVS饱和增益值。即对应于电压大幅度调整时的步长。	0x2
[23:16]	AVS调整中对HPM进行屏蔽，8bit从低到高依次对应电压域内的HPM0~7。	根据测试数据给出
[15:8]	AVS输出的电压上限对应的VID值。	根据电源芯片参数配置
[7:0]	AVS输出的电压下限对应的VID值。	根据电源芯片参数配置

步骤6 配置DVFS请求输出的电压值寄存器DVFS_NEXT_VOL。

Bits	寄存器说明	配置值
AVS_REG_GEN. DVFS_NEXT_VOL：偏移地址0x048		
[31:8]	保留。	-
[7:0]	DVFS请求输出的电压值。	根据电压模式配置 配置值为（V_{cfg}/（2^N*1000））/电压最小步长值，其中V_{cfg}为配置电压值（单位mv），电压最小步长值参见寄存器VOLTAGE_CONVERT_CFG。35

步骤7 配置HPM相关配置0寄存器HPMn_RELATED_CFG0。

Bits	寄存器说明	配置值
HPM*n*_RELATED_CFG0：偏移地址0x058+ *n**0x00C *n*=0~7		
[31:26]	AVS噪声下限值，本参数为有符号数。	根据测试数据给出
[25:20]	AVS噪声上限值，本参数为有符号数。	根据测试数据给出
[19]	保留。	-
[18:16]	HPM检测到的原始编码值缩放系数。	根据测试数据给出
[15:12]	HPM采样结果PC_ORG_0缩放系数，以16为底。	根据测试数据给出
[11:8]	HPM采样结果PC_ORG_1缩放系数，以16为底。	根据测试数据给出
[7:6]	保留。	-

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[5:0]	AVS的HPM时钟分频系数。 **注意：配置为0表示关闭HPM的时钟，因此若需要HPM正常工作时，不能配为0。**	根据测试数据给出

步骤8 配置HPM相关配置1寄存器HPMn_RELATED_CFG1。

Bits	寄存器说明	配置值
HPM*n*_RELATED_CFG1：偏移地址0x05C+ *n**0x00C *n*=0~7		
[31:22]	AVS稳态下限值，本参数为有符号数。	根据测试数据给出
[21:12]	AVS稳态上限值，本参数为有符号数。	根据测试数据给出
[11:10]	保留。	-
[9:0]	AVS参考校准值，作为HPM检测结果的调整参考值。	根据测试数据给出

步骤9 配置HPM相关配置2寄存器HPMn_RELATED_CFG2。

Bits	寄存器说明	配置值
HPM*n*_RELATED_CFG2：偏移地址0x060+ *n**0x00C *n*=0~7		
[31:26]	保留。	-
[25:16]	AVS饱和下限值，本参数为有符号数。	根据测试数据给出
[15:10]	保留。	-
[9:0]	AVS饱和上限值，本参数为有符号数。	根据测试数据给出

步骤10 配置DVFS和AVS启动寄存器TRIGGER_CFG。

Bits	寄存器说明	配置值
AVS_REG_GEN. TRIGGER_CFG：偏移地址0x0B8		
[31:17]	保留。	-
[16]	AVS调整启动信号。	0b1
[15:1]	保留。	-
[0]	DVFS调整启动信号。	0b0

----结束

版权所有 © 海思技术有限公司

10.6.7 算法旁路 HPM 检测配置

软件可配置算法旁路条件下单独启动AVS的HPM检测并通过相应寄存器上报获取HPM的检测值。

步骤1 配置寄存器AVS_WR_OPEN完成对AVS全局参数寄存器的写保护解锁，默认为锁定状态。

Bits	寄存器说明	配置值
AVS_WR_OPEN：偏移地址0x004		
[31:0]	写寄存器： 向本寄存器地址写入指定值AVS_WR_OPEN（默认为0x5A5A_5A5A，该宏可修改）后打开写使能；向本寄存器地址写入非指定值后则锁定写使能。保护地址范围为：0x030~0x7FF。 读寄存器： 寄存器读出值表示寄存器的锁定状态： 0x0000_0000表示寄存器处于锁定状态； 0x0000_0001表示寄存器处于解锁状态。	0x5A5A_5A5A（解锁） 0x0000_0000（锁定）

步骤2 配置寄存器PMBUS_WR_OPEN完成对PMBUS_PROC寄存器的写保护解锁，默认为锁定状态。

Bits	寄存器说明	配置值
PMBUS_WR_OPEN：偏移地址0xA04		
[31:0]	写寄存器： 向本寄存器地址写入指定值PMBUS_WR_OPEN（默认为0x5A5A_5A5A，该宏可修改）后打开写使能；向本寄存器地址写入非指定值后则锁定写使能。保护地址范围为：0x800~0xEFF。 读寄存器： 寄存器读出值表示寄存器的锁定状态： 0x0000_0000表示寄存器处于锁定状态； 0x0000_0001表示寄存器处于解锁状态。	0x5A5A_5A5A（解锁） 0x0000_0000（锁定）

步骤3 配置寄存器I2C_LOCK完成对PMBUS接口模块寄存器的写保护解锁，默认为解锁状态。

Bits	寄存器说明	配置值
I2C_LOCK：偏移地址0x8AC		

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[31:0]	写寄存器： 向Lock寄存器写32bits密钥（密钥宏定义配置，请查看宏定义文件），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。 读寄存器： 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。	0x5A5A_5A5A（解锁） 0x0000_0000（锁定）

步骤4 配置AVS检测相关寄存器AVS_CHECK_CFG。

Bits	寄存器说明	配置值
AVS_CHECK_CFG：偏移地址0x04C		
[31:29]	保留	-
[28]	平均模式使能。 0：最小值模式，选择观察窗范围内的HPM的最小值作为判断是否需要调压； 1：平均模式，选择观察窗范围内的HPM的平均值作为判断是否需要调压。	-
[27:8]	AVS调整过程中HPM检测间隔周期，以AVS工作时钟周期为单位，最大为2.0972ms@500MHz	-
[7:5]	保留	-
[4]	AVS计算调整旁路选择。 0：完整的AVS流程，即在完成HPM检测后，硬件完成计算和电压调整； 1：AVS计算调整旁路，仅使用HPM检测功能，即在完成HPM检测后，软件根据HPM检测结果来完成计算和并通过DVFS来完成电压调整。	0x1
[3:0]	AVS调整过程中每次调整需对HPM采样的次数。 0x0：表示采样2^（0）=1次 0x1：表示采样2^（1）=2次 … 0xF：表示采样2^（15）=32768次	-

步骤5 配置AVS其他配置寄存器AVS_MISCELL，配置HPM屏蔽位选择HPM。

Bits	寄存器说明	配置值
AVS_MISCELL：偏移地址0x054		

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[31:28]	AVS稳态增益值，即对应于电压小幅度调整时的步长	-
[27:24]	AVS饱和增益值，即对应于电压大幅度调整时的步长	-
[23:16]	AVS调整中对HPM进行屏蔽，8bit从低到高依次对应电压域内的HPM0~7，电平信号： 0：HPM开展检测（即AVS调整时启动该HPM）； 1：HPM不做检测（即不启动该HPM）。	根据测试需求进行配置
[15:8]	AVS输出的电压上限对应的VID值 **注意：本寄存器必须配置成保证芯片不被烧坏的安全电压值，否则可能导致芯片烧坏。**	-
[7:0]	AVS输出的电压下限对应的VID值 **注意：本寄存器必须配置成保证芯片不被烧坏的安全电压值，否则可能导致芯片烧坏。**	-

步骤6 配置HPM相关配置0寄存器HPMn_RELATED_CFG0，配置AVS的HPM时钟分频系数，如表10-4所示。

表 10-4 HPM_RELATED_CFG0 寄存器

Bits	寄存器说明	配置值
HPM*n*_RELATED_CFG0：偏移地址0x058+ *n**0x00C *n*=0~7		
[31:26]	AVS噪声下限值，本参数为有符号数。	-
[25:20]	AVS噪声上限值，本参数为有符号数。	-
[19]	保留。	-
[18:16]	HPM检测到的原始编码值缩放系数。	-
[15:12]	HPM采样结果PC_ORG_0缩放系数，以16为底。	-
[11:8]	HPM采样结果PC_ORG_1缩放系数，以16为底。	-
[7:6]	保留。	-
[5:0]	AVS的HPM时钟分频系数。 **注意：配置为0表示关闭HPM的时钟，因此若需要HPM正常工作时，不能配为0。**	根据测试需求进行配置

步骤7 配置使能和DVFS配置的电压门限寄存器ENABLE_AND_DVFS_VOL_LMT，配置AVS使能位有效。

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
ENABLE_AND_DVFS_VOL_LMT：偏移地址0x044		
[31:29]	保留。	-
[28]	AVS使能，电平信号。 0：AVS功能失效； 1：AVS功能使能，支持DVFS硬件触发和软件触发。	0x1
[27:25]	保留	-
[24]	DVFS触发AVS使能，电平信号。 0：DVFS调整结束不触发AVS调整； 1：DVFS调整结束将触发AVS调整。	0x0
[23:21]	保留。	-
[20]	DVFS使能，电平信号。 0：DVFS功能不使能； 1：DVFS功能使能，支持软件触发模式。	0x0
[19]	保留	-
[18]	VID调整方向。 0：增加VID值，表示调高电压（即VID值与电压值同相，增加VID值，表示调高电压，减少VID值，表示降低电压）； 1：减少VID值，表示调高电压（即VID值与电压值反相，增加VID值，表示降低电压，减少VID值，表示调高电压）。	-
[17]	保留。	-
[16]	电源接口为VID模式时，三态控制线使能，电平信号。 0：三态控制线无效，电压由外部的上/下拉电阻决定； 1：三态控制线有效，电压由AVS输出的VID值控制。	-
[15:8]	DVFS输出的电压上限对应的VID值。 **注意：本寄存器必须配置成保证芯片不被烧坏的安全电压值，否则可能导致芯片烧坏。**	-
[7:0]	DVFS输出的电压下限对应的VID值。 **注意：本寄存器必须配置成保证芯片不被烧坏的安全电压值，否则可能导致芯片烧坏。**	-

步骤8 配置AVS启动寄存器TRIGGER_CFG，启动AVS调整。

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
TRIGGER_CFG：偏移地址0x0B8		
[31:17]	保留。	-
[16]	AVS调整启动信号，写1产生脉冲信号。 **注意：当相应电压域的ENABLE_AND_DVFS_VOL_LMT.avs_en=0b1时，且AVS的状态机空闲（AVS_PMUCTRL_RPT.avs_fsm_state=0b0）时，配置本寄存器才会触发AVS，否则逻辑不保证处理的正确性。**	0x1
[15:1]	保留。	-
[0]	DVFS调整启动信号，写1产生脉冲信号。 **注意：当相应电压域的ENABLE_AND_DVFS_VOL_LMT.dvfs_en=0b1且DVFS的状态机空闲（DVFS_RPT.dvfs_fsm_state=3'd0）时，配置本寄存器才会触发DVFS，否则该启动将被丢弃。**	0x0

步骤9 查询AVS和PMUCTRL模块状态上报寄存器AVS_PMUCTRL_RPT（偏移地址0x530）bit[9:8]，确认HPM检测完毕。

Bits	寄存器说明	查询值
AVS_PMUCTRL_RPT：偏移地址0x504		
[31:20]	保留	-
[19:16]	PMU_CTRL模块当前相应的请求。 0x1：电源芯片上电请求； 0x2：电源芯片掉电请求； 0x4：DVFS发出的调压请求； 0x8：AVS发出的调压请求。	-
[15:10]	保留	-
[9:8]	AVS调整状态机状态上报。 00：空闲状态； 01：HPM检测状态； 10：AVS计算状态； 11：AVS等待状态。	0x0
[7:0]	AVS输出的电压对应的VID值。	-

步骤10 读取HPMn检测检测结果上报寄存器HPMn_SAMPLE_RESULT，获取HPM的采样结果。

版权所有 © 海思技术有限公司

Bits	寄存器说明	查询值
HPM*n*_SAMPLE_RESULT：偏移地址0x50C+ *n**0x004 *n*=0~7		
[31:26]	保留。	-
[25:16]	AVS采样得到的HPM0返回的通道1的检测数据，对应的Ring Oscillator中延时单元采用LVT库。	-
[15:10]	保留。	-
[9:0]	AVS采样得到的HPM0返回的通道0的检测数据，对应的Ring Oscillator中延时单元采用HVT库。	-
[7:0]	AVS输出的电压对应的VID值。	-

----结束

10.6.8 Adaptive Clock

芯片上电后，系统处于pre-normal模式（模式频率-50MHz），初始化Adaptive Clock。

步骤1 各CPU和LLC CPM模块初始化。

Bits	寄存器说明	配置值
SysCtrl.SC_CPM_CLUSTER0_CFG0：偏移地址0x3610 SysCtrl.SC_CPM_CLUSTER1_CFG0：偏移地址0x3620 SysCtrl.SC_CPM_CLUSTER2_CFG0：偏移地址0x3630 SysCtrl.SC_CPM_CLUSTER3_CFG0：偏移地址0x3640 LLC.LLC_CPM_CFG0：偏移地址0x0700（x4）		
[31:30]	保留	-
[29:28]	CPM码字的输出模式。 00：直接输出； 01：最小值输出； 10：最大值输出； 11：不翻转。	0x0
[27:25]	保留。	-
[24:20]	设定CPM检测的门限。	0x00
[19:17]	保留。	-
[16]	电压跌落指示输出禁止位。 0：允许输出电压跌落指示； 1：禁止输出电压跌落指示。	0x0
[15:12]	SVT长线级联，级数配置PIN。	0x0

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[12:8]	SVT短线级联，级数配置PIN。	0x0
[7:4]	LVT长线级联，级数配置PIN。	0x0
[3:0]	LVT短线级联，级数配置PIN。	0x0

步骤2 ACG模块初始化；

Bits	寄存器说明	配置值
POE Subctrl.SC_ACG_CFG0：偏移地址0x3600		
[31:20]	保留。	-
[19:18]	频率降低比率。 00：10%； 01：18.18%； 10：25%； 11：reserve	0x1
[17:16]	电压检测模式选择。 00：reserve（不降频）； 01：IP内部电压检测（CPM）； 10：IP外部电压检测（ND）； 11：ND+CPM。	0x0
[15:14]	保留。	-
[13]	使能信号，工作模式不翻转（低功耗）。 0：默认； 1：表示64分频模块工作。	0x0
[12]	用于DFT量产筛选。 Glitch测试使能。 0：关闭； 1：打开。	0x0
[11]	用于DFT量产筛选。 匹配达标检测使能，1有效； 工作状态，要保持为0。	0x0
[10]	Charge pump电流大小配置。	0x0

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[9]	逐级升频模式下，间隔选择。 0：64个cycle； 1：128个cycle or。	0x0
[8]	退出降频模式选择。 0：直接升频； 1：逐级升频。	0x0
[7:0]	降频使能信号的选择。 对应ENABLE_FFS[3:0]； 0：表示禁用； 1：表示启用。	0xFF

Bits	寄存器说明	配置值
POE Subctrl.SC_ACG_CFG1：偏移地址0x3604		
[31]	监控路径复位信号的脉宽选择。	0x0
[30:26]	CPM下降沿检测设定的门限值。	0x0
[25:21]	CPM上升沿检测设定的门限值。	0x0
[20]	极限值输出状态下，使能信号。 0：清除前一次求值； 1：使能极限求值。	0x0
[19:18]	CPM码字的输出模式。 00：直接输出； 01：最小值输出； 10：最大值输出； 11：不翻转。	0x0
[17]	0：上升沿检测； 1：双沿检测（上升沿+下降沿）。	0x1
[16]	电压检测采样周期。 0：1cycle； 1：2cycle。	0x1
[15:12]	LVT长线级联，级数配置PIN。	0x0
[11:8]	LVT短线级联，级数配置PIN。	0x0
[7:4]	SVT长线级联，级数配置PIN。	0x0
[3:0]	ULVT短线级联，级数配置PIN。	0x0

版权所有 © 海思技术有限公司

步骤3 配置PLL0 OCC时钟源MUX，采用ACG输出的时钟。

Bits	寄存器说明	配置值
POE Subctrl.SC_ACG_BYPASS：偏移地址0x3650		
[31:1]	保留。	-
[0]	ACG模块bypss控制。 0：不使用ACG，时钟来自PLL； 1：使用ACG模块之后的时钟。	0x1

步骤4 CPU配置各PLL，待各PLL锁定。

步骤5 撤离ACG模块的FFS复位。

Bits	寄存器说明	配置值
POE Subctrl.SC_ACG_RESET_REQ：偏移地址0xD28		
[31:1]	保留。	-
[0]	软复位请求。 0：无意义（写0无效）； 1：软复位请求有效。	0x0

Bits	寄存器说明	配置值
POE Subctrl.SC_ACG_RESET_DREQ：偏移地址0xD2C		
[31]	保留	-
[0]	软复位请求撤离。 0：无意义（写0无效）； 1：去复位请求有效，逻辑撤离复位。	0x1

步骤6 读取CPM码字。

Bits	寄存器说明	配置值
SysCtrl.SC_CPM_CLUSTER0：偏移地址0x6610		
SysCtrl.SC_CPM_CLUSTER1：偏移地址0x6620		
SysCtrl.SC_CPM_CLUSTER2：偏移地址0x6630		
SysCtrl.SC_CPM_CLUSTER3：偏移地址0x6640		

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
LLC.LLC_CPM_ST：偏移地址 0x0708（x4）		
[31:12]	保留。	-
[11:8]	cpm_test_out。	-
[7:6]	保留。	-
[5:0]	CPM码字的输出。	-

Bits	寄存器说明	配置值
POE Sysctrl.SC_ACG_ST：偏移地址 0x6604		
[31:24]	保留。	-
[23:20]	预留输出测试PIN。	-
[19:17]	保留。	-
[16]	DROP FLAG输出。	-
[15:13]	保留。	-
[12:8]	上升沿检测，CPM码字的输出。	-
[7:5]	保留。	-
[4:0]	下降沿检测，CPM码字的输出。	-

步骤7 设置CPM检测的门限

Bits	寄存器说明	配置值
SysCtrl.SC_CPM_CLUSTER0_CFG0：偏移地址0x3610 SysCtrl.SC_CPM_CLUSTER1_CFG0：偏移地址0x3620 SysCtrl.SC_CPM_CLUSTER2_CFG0：偏移地址0x3630 SysCtrl.SC_CPM_CLUSTER3_CFG0：偏移地址0x3640 LLC.LLC_CPM_CFG0：偏移地址0x0700（x4）		
[31:30]	保留	-
[29:28]	CPM码字的输出模式。 00：直接输出； 01：最小值输出； 10：最大值输出； 11：不翻转。	0x0

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[27:25]	保留。	-
[24:20]	设定CPM检测的门限。	-
[19:17]	保留。	-
[16]	电压跌落指示输出禁止位： 0：允许输出电压跌落指示； 1：禁止输出电压跌落指示。	0x0
[15:12]	SVT长线级联，级数配置PIN。	0x0
[12:8]	SVT短线级联，级数配置PIN。	0x0
[7:4]	LVT长线级联，级数配置PIN。	0x0
[3:0]	LVT短线级联，级数配置PIN。	0x0

步骤8 设置电压检测模式,Adaptive Clock正常工作。

Bits	寄存器说明	配置值
POE Subctrl.SC_ACG_CFG0：偏移地址0x3600		
[31:20]	保留。	-
[19:18]	频率降低比率。 00：10%； 01：18.18%； 10：25%； 11：保留。	0x1
[17:16]	电压检测模式选择。 00：保留（不降频）； 01：IP内部电压检测（ACG CPM）； 10：IP外部电压检测（CPU/LLC CPM）； 11：CPM+ACG CPM。	0x3
[15:14]	保留。	-
[13]	使能信号，工作模式不翻转（低功耗）。 1：表示64分频模块工作。	0x0
[12]	用于DFT量产筛选。 Glitch测试使能。 0：关闭； 1：打开。	0x0

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[11]	用于DFT量产筛选。 匹配达标检测使能，1有效； 工作状态，要保持为0。	0x0
[10]	Charge pump电流大小配置。	0x0
[9]	逐级升频模式下，间隔选择。 0： 64个cycle； 1：128个cycle or。	0x0
[8]	退出降频模式选择。 0：直接升频； 1：逐级升频。	0x0
[7:0]	降频使能信号的选择。 对应ENABLE_FFS[3:0]； 0：表示禁用； 1：表示启用。	0xFF

----结束

10.6.9 SPPE 掉电流程

软件可配置单独下电SPPE，完全有软件来实现MTCMOS，ISO开关等的控制。

步骤1 配置ISO钳位，配置寄存器SC_PWR_CFG的iso_en_sub（bit3），使能SPPE_TOP的钳位，钳位输出。

Bits	寄存器说明	配置值
SC_PWR_CFG：偏移地址0x1580		
[31:0]	写寄存器： iso_en_sub（bit3）位写1表示打开钳位；（默认不钳位） 向本寄存器地址（默认为0x0000_0005）写入0x0000_000D打开SPPE钳位。	0x0000_0005（默认） 0x0000_000D（钳位）

步骤2 配置MTCMOS关断，配置寄存器SC_PWR_CFG的pwrup_sub（bit2），关断MTCMOS，使SPPE_TOP掉电。

Bits	寄存器说明	配置值
SC_PWR_CFG：偏移地址0x1580		

版权所有 © 海思技术有限公司

Bits	寄存器说明	配置值
[31:0]	写寄存器： iso_en_sub（bit2）位写0表示MTCMOS关断；（默认打开） 再向本寄存器地址写入0x0000_0009关断MTCMOS，SPPE下电。	0x0000_000D（钳位） 0x0000_0009（下电）

步骤3 查询关断状态，软件查询PERI SUBCTRL寄存器SC_PWR_ST的pwrup_ack_sub位。

Bits	寄存器说明	配置值
SC_PWR_ST：偏移地址0x1584		
[31:0]	读寄存器： 读取pwrup_ack_sub位（bit1）的值，为0表示SPPE下电。	N/A

----结束

10.6.10 HISS 掉电流程

软件可配置单独下电HISS，完全有软件来实现MTCMOS，ISO开关等的控制。

步骤1 配置ISO钳位，配置寄存器SC_PWR_CFG的iso_en_hiss（bit1），使能HISS的钳位，钳位输出。

Bits	寄存器说明	配置值
SC_PWR_CFG：偏移地址0x1580		
[31:0]	写寄存器： iso_en_hiss（bit1）位写1表示打开钳位；（默认不钳位） 向本寄存器地址（默认为0x0000_0005）写入0x0000_0007打开HISS钳位。	0x0000_0005（默认） 0x0000_0007（钳位）

步骤2 配置MTCMOS关断，配置寄存器SC_PWR_CFG的pwrup_hiss（bit0），关断MTCMOS，使HISS掉电。

Bits	寄存器说明	配置值
SC_PWR_CFG：偏移地址0x1580		
[31:0]	写寄存器： iso_en_hiss（bit0）位写0表示MTCMOS关断；（默认打开） 再向本寄存器地址写入0x0000_0006关断MTCMOS，HISS下电。	0x0000_0005（钳位） 0x0000_0006（下电）

版权所有 © 海思技术有限公司

步骤3 查询关断状态，软件查询PERI SUBCTRL寄存器SC_PWR_ST的pwrup_ack_hiss位。

Bits	寄存器说明	配置值
SC_PWR_ST：偏移地址0x1584		
[31:0]	读寄存器： 读取pwrup_ack_sub位（bit 0）的值，为0表示HISS下电。	N/A

----结束

10.6.11 相应模块 CLK GATING 配置流程

如果软件需要对相应模块进行clk gating，详见各模块寄存器表单，具体配置：

配置该模块的寄存器icg_en对应bit为关闭，关闭时钟。

10.6.12 SERDES MACRO 的 power gating 配置流程

如果软件需要对DATASLICE SHUT DOWN，具体配置参考如下：

图 10-15 DATASLICE SHUT DOWN 参考代码 1

```
//==========================================================================================
==
// Function: LaneShutDown
// - This function is used for lane power down and reset to save power energy when the lane does not need work.
// - dsIndex: The number of DS that should be use lane power down.
// - dsMask: The mask for DS, assert one bit to enable corresponding DS. Eg. Set dsMask to 0x81 to enable DS7 and
DS0.
// - Type: 0: ds_rstb can not be in reset status and after need start-up and DS Calibration can be skip;
//         1: ds_rstb can be in reset status and save more power but after need start-up and DS Calibration can not be
skip
// - The return value can be 0/1, 0 means reset timeout, user need to double check the refclk and power supply, etc.
//==========================================================================================
==
bool LaneShutDown(unsigned short dsMask, unsigned short Type)
{
    unsigned char dsIndex;
    unsigned short checkCount;
    unsigned char doneMask;
    unsigned char dsIdNorthCs;    //The ID of DS on the north side of CS.
    unsigned char dsNum;    // Total DS number in current macro.

    // Get the Ds Number in current macro
    dsNum = DS_ID_NORTH (CSDIG_CSR0 bit[15:12]) + DS_ID_SOUTH (CSDIG_CSR0 bit[11:8]);

    if ((0 == CORE_WAIT(CP_CSR28 bit[0])) && ( 1 == MCURSTB (CP_CSR11 bit[0])))
    {
        for (dsIndex=0; dsIndex<dsNum; dsIndex++)
        {
            if (0x1 == ((dsMask >> dsIndex) & 0x1))
```


版权所有 © 海思技术有限公司

图 10-16 DATASLICE SHUT DOWN 参考代码 2

```
            {
                // Step1: lane soft reset
                DS_SOFT_RESET_API (DSAPI_CSR0 bit[4]) = 0x1;
            }
        }

        // Wait the FW Soft Reset Done
        doneMask = 0;
        checkCount = 0;
        dsIndex = 0;
        do
        {
            checkCount++;
            wait (100us);

            if ((0x1 == ((dsMask >> dsIndex) & 0x1)) && (0x0 == ((doneMask >> dsIndex) & 0x1)))
            {
                doneMask |= (DS_SOFT_RESET_STATUS_API (DSAPI_CSR13 bit[0]) << dsIndex);
            }
            if(++dsIndex >= dsNum)
            {
                dsIndex = 0;
            }
        } while ((doneMask != dsMask) && (1000 > checkCount));

        if (1000 <= checkCount)
        {
            print("Dataslice firmware reset timeout!");
            return false;
        }
    }

    // Power down and Reset
    for (dsIndex=0; dsIndex<dsNum; dsIndex++)
    {
        if (0x1 == ((dsMask >> dsIndex) & 0x1))
        {
            // Step2: Power down Rx/Tx Part of DS
            if (false == DsPowerDownWoClkDis(dsIndex))
            {
                return false;
```


版权所有 © 海思技术有限公司

图 10-17 DATASLICE SHUT DOWN 参考代码 3

```
                }

                // set ds_rstb or not
                if (1 == Type)
                {
                    ds_rstb[dsIndex](pin) = Low Voltage;
                }
            }
        }

        return true;
}
```

如果软件需要对DATASLICE RESTART，具体配置参考如下：

图 10-18 DATASLICE RESTART 参考代码 1

```
//==========================================================================================
// Function: LaneRestart
// - This function is used for recovery current lane from hardware lane reset status to work again.
// - dsMask: The mask for DS, assert one bit to enable corresponding DS. Eg. Set dsMask to 0x81 to enable DS7 and
DS0.
// - The return value can be 0/1, 0 means reset timeout, user need to double check the refclk and power supply, etc.
//==========================================================================================
bool LaneRestart(unsigned short dsMask)
{
    unsigned char dsIndex, doneMask;
    unsigned short checkCount;
    unsigned char dsIdNorthCs;    //The ID of DS on the north side of CS.
    unsigned char dsNum;    // Total DS number in current macro.
    unsigned char dsResetMask;

    //Step1: Do CS PLL calibration if necessary
    if (need to change PLL frequency)
    {
        if (false == CsVcoSwitch(pllIndex)) // including Reset Soft FW
        {
            print("Clockslice PLL re-load fail!");
            return false;
        }
    }

    // Get the Ds Number in current macro
    dsNum = DS_ID_NORTH (CSDIG_CSR0 bit[15:12]) + DS_ID_SOUTH (CSDIG_CSR0 bit[11:8]);
```


版权所有 © 海思技术有限公司

图 10-19 DATASLICE RESTART 参考代码 2

```
// Step2: reset the lane firmware
if ((0 == CORE_WAIT(CP_CSR28 bit[0])) && ( 1 == MCURSTB (CP_CSR11 bit[0])))
{
    for(dsIndex=0; dsIndex<dsNum; dsIndex++)
    {
        // Step1: Dataslice firmware reset.
        DS_SOFT_RESET_API (DSAPI_CSR0 bit[4]) = 0x1;
        DS_INIT_ADAPT_EN_API (DSAPI_CSR0 bit[2]) = 0x0;
        DS_CONT_ADAPT_EN_API (DSAPI_CSR0 bit[3]) = 0x0;
    }

    // Wait the FW Soft Reset Done
    doneMask = 0;
    checkCount = 0;
    dsIndex = 0;
    do
    {
        checkCount++;
        wait (100us);

        if ((0x1 == ((dsMask >> dsIndex) & 0x1)) && (0x0 == ((doneMask >> dsIndex) & 0x1)))
        {
            doneMask |= (DS_SOFT_RESET_STATUS_API (DSAPI_CSR13 bit[0]) << dsIndex);
        }
        if (++dsIndex >= dsNum)
        {
            dsIndex = 0;
        }
    } while ((doneMask != dsMask) && (1000 > checkCount));

    if (1000 <= checkCount)
    {
        print("Dataslice firmware reset timeout!");
        return false;
    }
}

//Step3: Power up and release the lane
dsResetMask = 0;
for (dsIndex=0; dsIndex<dsNum; dsIndex++)
```


版权所有 © 海思技术有限公司

图 10-20 DATASLICE RESTART 参考代码 3

```
    {
        if (0x1 == ((dsMask >> dsIndex) & 0x1))
        {
            if (0x0 == ds_rstb[dsIndex](pin))
            {
                // Step1: Release lane reset
                ds_rstb[dsIndex](pin) = High Voltage;
                dsResetMask |= (1 << dsIndex);

                // Step2: Power up DS without high speed distribution.
                if (false == DsPowerUpWoClkDis(dsIndex))
                {
                    return false;
                }

            }
        }
    }


    //Step4: Do DS calibration: All calibrations except DSCLK,case3/4
    if (false == OnLineDsCalib(dsMask))
    {
        print("Dataslice calibration fail!");
        return false;
    }

    //Ste5: Configure adaptation API
    for (dsIndex=0; dsIndex<dsNum; dsIndex++)
    {
        if (0x1 == ((dsMask >> dsIndex) & 0x1))
        {
            DsAdaptationApiConfig();
        }
    }

    //Step6: redo Adaptation
    if (false == RedoAdaptation(dsMask))
    {
        print("Dataslice adaptation fail!");
        return false;
```

图 10-21 DATASLICE RESTART 参考代码 4

```
    }

    return true;
}
```


版权所有 © 海思技术有限公司

11 系统 DFX

11.1 关键术语

表 11-1 系统 DFX 术语表

术语	描述
FIT	Failure In Time，10^9设备1小时下（例如一千个零件运转百万小时，一百万个零件运转一千小时……等）预期的失效次数。
MBIST	Memory内建自测试，即自动实现存储器单元或阵列的RTL级内建自测试电路。
ALL-SCAN	基于SCAN结构的板级ATPG方案，用于Dump芯片所有的FF值。
DCM	Die Crack Monitor，Die开裂监控电路。

11.2 概述

系统DFX从系统角度描述了芯片支持的DFX功能，通过软硬件结合，支撑芯片问题定位和关键信息搜集。

11.3 特性

系统DFX具有以下特性：

- 提供集中式的调试控制器用于系统调试和测试，兼容IEEE1149.1JTAG标准、IEEE1149.6JTAG标准。
  - 调试模式和测试模式共用JTAG控制器和接口。
  - 多种JTAG配置方式，支持内部DFT JTAG、ARM JTAG和DJTAG配置通路，适应不同的调试场景和应用场景。
  - 支持通过JTAG连接调试器。

版权所有 © 海思技术有限公司

  - 支持多芯片JTAG串联，支持芯片内多个JTAG控制器串链或独立成链共用JTAG管脚。
- 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。
  - JTAG接口支持工作在25MHz及以下的速率。
  - 支持安全调试。
  - 提供JTAG鉴权。
  - JTAG管脚选用带施密特触发的IO。
- 片上Memory都支持MBIST。
  - 片上所有Memory，都支持JTAG启动和功能MBIST测试，部分较大的Memory支持自修复。
  - At-speed MBIST测试。
  - 灵活的MBIST功能启动模式，避免功耗过大问题。
  - 功能MBIST测试并行度可配置。
  - 功能MBIST测试完成时间不长于440毫秒（不包含DDR）。
  - DDR ECC MBIST支持的算法是基于ExMbist上的优化。
  - 增加测试应力。
  - 优化算法中使用的背景数（针对颗粒位宽为8bit、16bit）。
  - 测试算法集合补充：
    - bit均衡算法。
    - 总线走步算法。
    - Margin测试。
  - 功能MBIST不依赖于外部器件初始化。
  - 支持检测内部RAM读写和bit异常翻转的异常并上报。
  - MBIST测试错误提供中断上报，错误次数统计，提供出错地址和数据（回读值，期望值）信息，能够通过软件控制MBIST测试模式（出错后即停止或出错后继续运行）。
  - 支持各模块的MBIST功能直接按By Memory上报错误，并且能够区分模块，通过寄存器可查询。
  - 芯片支持通过简单配置数个寄存器即能启动芯片MBIST功能。
  - 支持在正常启动和装备流程中使用MBIST。
  - ASIC内建逻辑对内置存储器进行检测，即支持在功能模式下启动Mbist测试。
- 支持软失效保护和相关特性。
  - Hi309a漏检的软失效指标为80FIT（挑战目标40FIT），检测到但无法纠错的软失效指标为400FIT（挑战目标300FIT）。
  - SRAM和DRAM都支持ECC软失效保护，错误后记录出错地址信息并上报中断。
  - 支持错误注入功能。
    - 支持L1/2 Cache和SRAM、DDR模块单bit、多bit故障注入功能。（L3不支持故障注入）。
  - DDR ECC支持ECC回写和出错地址记录，片上SRAM ECC支持ECC出错地址记录。

版权所有 © 海思技术有限公司

  - DDR接口支持ECC错误地址、数据和期望数据记录。
  - 芯片内部各模块提供Memory ECC/ Parity功能开启功能接口（功能默认开启）。
  - 出错之后的记录信息需要区分到RAM的位置。
  - ECC支持出错回写功能，避免ECC单bit积累。
  - 所有模块内部SRAM采取统一的ECC设计原则：单bit ECC错误自动纠错，不上报中断，2bits ECC错误上报中断。
  - 针对部分单bit错误且提供了出错地址的，不强制要求回写，要求响应相应中断并采用软件回写。
  - 支持每个核 能访问到其他Core的L1/L2 Cache ECC/Parity 错误的相关状态。
  - 支持内存巡检功能，周期性或者按照需要刷新内存，避免单bit失效积累成多bit失效。
- 数据通路模块、总线、接口模块和CPU都包括丰富的统计功能寄存器和故障告警功能，支持实时监控芯片数据流、控制流。
  - 统计位宽与MIB标准一致，即对帧的统计采用32位寄存器，对字节和bit的统计采用64位寄存器。
  - 高速端口，统计需要支持64bit位宽，保证不翻转。包括TM队列和端口，GE/XGE端口，PPE。
  - 内部所有加速器支持收发字节数（只有感知字节数的加速器才支持字节数统计）、包数统计（含正确/错误）。
  - 硬件提供性能统计计数器，用于评估硬件实际性能的发挥程度，比如CPU占有率、CACHE命中率、IPC；提供流量统计功能。
  - 支持各物理口收发报文的统计。
  - 支持转发报文的统计。
  - 支持记录第一个报文的key。
  - 支持靠MAC侧，基于MAC粒度配置内环和外环。
  - 所有丢包地方都有基于丢包类型的统计。
  - 配置类TCAM支持背景扫描校验ECC，并单独上报中断状态。
  - 支持配置类MEM/TCAM出现1bit ECC错误纠错和检错功能，由软件对该mem全部回读并重写。
  - 支持动态门控，模块没有业务时，自动关闭时钟。
  - 支持IPG可配置。
- 高速接口如PCIE、Serdes和DDR等满足了可靠性设计需求。
- PCIE可靠性需求。
  - 提供数据校验功能，并提供校验错统计功能。
  - 关键的状态信号支持查询功能。
  - PCIe支持Trace功能，PCIe Trace可以跟踪PCIe的状态机跳转过程。
- Serdes可靠性需求。
  - 支持PRBS功能。
  - 支持接口和链路检测功能。
  - 提供链路故障信息记录和查询功能。
  - 提供数字眼图功能。

版权所有 © 海思技术有限公司

  - 支持环回功能等。
- DDR可靠性需求。
  - 支持SEC/DED ECC校验。
  - 支持DDR MBIST测试，可自动完成DDR颗粒遍历测试。
  - CPU外挂DDR ECC颗粒内存单元测试，芯片提供MBIST测试，支持多种软件配置，硬件测试算法设计包含比如PARC20种（支持BITSPREAD_COMPARISON和BITFLIP_COMPARISON），能对ECC做全覆盖测试。增加：
    - Mbist中增加行相关性、列相关性需求。
    - 合入次优先级算法：March C-、March C-R、March U、March Y,March SR、MarchC+、MEMTEST_MATS+、MEMTEST_MATS++、MEMTEST_Marching 1/0、March SSE、March28n、MSCAN、MEMTEST_MOVI（PMOVI)、MEMTEST_UARBW、MEMTEST_PMOVI PRBS ALL、MEMTEST_PRBS ALIGN DATA、MEMTEST_PRBS FLIP DATA、MEMTEST_PRBS ALIGN ADDR和MEMTEST_PRBS FLIP ADDR。
  - 对ECC错误提供出错地址和数据信息，应用能根据此信息定位到DDR内存颗粒。
  - 要求实现DDRC的地址线和数据线的在线环回，当前数据线已经支持（双向），支持区分DDRC、DDR PHY和DDR颗粒的问题定位。
  - 支持DDR ECC颗粒可直接访问测试。
- 支持性能分析。
  - 支持全局时间戳，用于性能分析。
  - 支持PMU统计。
  - 增加各模块访问总线/DDR延迟统计（最大，最小，平均），处理流量和带宽统计， 负载统计。
- 支持通过管脚和寄存器配置两种方式选择内部关键信号输出到管脚观测。
- 支持环回测试功能。
  - 接口模块如GE，XGE，PCIE，SERDES，SPI等支持环回功能。
  - 支持各通信接口物理内外环回，对于集成MAC和 PHY的要分层环回。 支持MAC层到PCS层环回，及PCS层到Serdes的环回。
  - 支持无损环回（无损的意思是在线环回，即在有业务运行的状态下，CPU可以指定其中一个报文在GMAC控制器中并行环回，不影响其他正常业务的收发）。
  - 提供命令完成指定业务通道的指定环回测试（完成该环回测试，包括的步骤为：设置环回、发送数据、接收数据进行比较、解除环回，类似批处理）。
  - 各个GE端口能独立设置、解除和查询。
  - 支持对指定端口进行单独环回测试。
  - 支持在线环回功能，对于复杂的模块要支持多级环回。
  - 支持XGE报文无损环回功能。
  - 支持内部加速器模块的智能心跳功能。
- 提供器件版本寄存器，通常标示器件主版本号+次版本号，如：表示V1.02版本，可通过器件版本信息，分析不同版本器件差异来协助问题定位。

版权所有 © 海思技术有限公司

- 支持芯片的温度检测功能。
  - 提供DIEID寄存器，给每个芯片提供唯一的Die ID，可通过JTAG和CPU读取。
  - 提供器件版本寄存器，通常标示器件主版本号+次版本号，如：表示V1.02版本，可通过器件版本信息，分析不同版本器件差异来协助问题定位。
  - 支持芯片的温度检测功能。
  - 根据实际物理布局和芯片热点位置确定温度传感器放置位置。
  - 支持过热保护功能，在结温超过设定阈值后，可以上报结温超标告警。
  - Tensor检测精度支持校正后±3℃。
  - 支持检测温度范围-40℃ ~ +120℃。
  - 片内集成温度传感器，探测芯片温度，支持125℃。
  - 温控传感器的精度要求高结温区间（80度到125度）±2C。其他温度范围内±4C。
  - 支持芯片复位时，通过JTAG直接读取TSensor获取当前结温。
  - 实际温度可读取，过温门限可配置。
- 支持时钟和复位检测和状态查询功能。
  - 支持时钟有无检测，锁相环锁定检测，并提供可查询的状态寄存器。
  - 各锁相环支持独立复位，包括系统内部和Serdes内部。
  - 提供Serdes PLL失锁中断上报功能，并且可以查看Serdes的历史失锁状态。
  - 提供Serdes参考源时钟丢失检测中断上报功能。
  - 支持检测内部锁相环是否失锁,通过寄存器上报。
- 提供AVS0维测接口。
  - 支持调压结果上报。
  - 支持当前电压监测。
  - 支持手动调压配置。
  - 在关闭AVS功能和开启AVS功能的情况下，都可以进行AVS的裕量测试。
- 提供压力测试机制。
  - 支持片内构造满流量冲击各加速器和CPU，建议压力测试要能够达到性能能力的4倍，负载可调0~4倍，被测试模块能够观测到负载状态。
  - 支持芯片内模块（如：各加速器、HAC等）压力测试，压力不小于产品实际应用。可以分模块单独测试，根据实际应用场景将相关联模块串联起来做系统级测试。模块并发软件来实现，硬件支持激励源的需求。
  - 支持SERDES接口测试，遵从压力测试规范。
- 满足海思ASIC工程设计基线V07.00。
  - 芯片硬失效率要求至少满足海思工程基线的要求（最好小于20FIT）。
  - ASIC内建逻辑对内置存储器进行检测，即支持在功能模式下启动Mbist测试。
- L1、L2可以关闭，支持Dram无缓存测试。
- 支持内存巡检功能，周期性或者按照需要刷新内存，避免单bit失效积累成多bit失效。
- 支持芯片内核间互联接口支持检测与故障上报。
- 支持Cache一致性测试。
- 满足海思DFx需求基线V03.00。

版权所有 © 海思技术有限公司

- 支持SERDES接口测试，遵从压力测试规范：
  - 提供发包激励源，支持近端环回和远端环回。
  - 多个接口可以同时测试。通过配置多个数据流，完成多个接口同时测试。
  - 满规格发包。每个接口都可以模拟发送报文，收发流量不小于设计规格，报文内容要求根据硬件布线确定测试报文的内容，能够遍历关键连线的状态变化，能够对接口流量模型进行模拟测试，例如报文内容采用PRBS填充。能够模拟不同长度的报文。
  - 对于模拟报文的进行收发包数和字节数统计，判断是否丢包，是否达到设计规格。
- 支持总线互联Trace和Debug。
  - 总线和互联支持Trace和Debug（不包含DSPS），CPUS继承原1382的coresight。
- 所有模块均分配地址空间，并能够保证被DJTAG访问。
- 支持模块运行状态检查及获取。
  - 支持加速器异常上报、加速器工作状态获取、加速器独立复位、模块处理时延和加速器状态检查。
  - 支持包处理、I/O类加速器的内部队列内容获取、清空和状态复位。
  - 支持调度类加速器处理的输入、输出跟踪。
  - 支持核、加速器、I/O故障检测机制，所有内存、队列资源、寄存器状态可以被软件读取。
  - 对于需要硬件动态申请使用外部DDR缓存的加速引擎，需要提供一个可读寄存器指示目前占用了多少buf，便于内存泄露问题的定位。
  - 网口自协商成功与否，网口link up状态，片上有明确的寄存器正确指示信息。
- 每个源中断尽量具有独立的中断号，避免软件中断处理流程中查询多个寄存器才能识别中断源。
- 芯片故障检测能力。
  - 故障检测识别率达到95%。
  - 故障检测能明确识别故障原因。

    芯片故障的原因可分为：

    - 芯片逻辑BUG, 芯片物理缺陷。
    - 软失效。
    - 软件操作错误。
    - 外部接口输入错误。
  - 故障检测支持隐性故障检测识别。
  - 故障检测粒度精确到最小可恢复粒度，故障出现后，定位信息能区分到故障源，不同后处理的故障源要分开。
  - 内部加速器支持心跳功能检测，实时监控内部核心状态，并在出现异常时上报中断。
  - DFX 记录异常操作信息包含源MASTER的信息。
  - 支持检测内部RAM读写和bit异常翻转的异常并上报。

版权所有 © 海思技术有限公司

- 芯片管脚个别数据线/地址线拉死，能检测出来。

- 芯片提供一些关键KPI的测量信息，软件定期查询并进行数据分析，提前预警，达到预警门限则告警。

  KPI的测量信息包括但不限于：

  - RAM 和DDR可自愈的软失效发生的频度。
  - 芯片物理参数检测：电路速度、温度和电压等。
  - 关键模块处理资源的负载：
    - BMU的最大、平均使用率、队列的最大、平均使用率。
    - NP 的线程利用率。
    - DDR 带宽占用率。
    - 访问性能，延迟的最大，最小，平均值统计等。

    系统软件，知道业务负载的情况，可以判断是否异常；使用设计的极限值数据判断是否异常。

    接口模块各层的异常频度统计： 物理层误码，协议层RETRY 报文，链路状态闪断频度等。

    KPI采样后，软件根据预判算法，计算KPI的趋势，有恶劣倾向的进行预警。

- 故障上报机制规范。
  - 故障都支持中断上报。
  - 支持以异常方式上报。以该方式上报的，能够在CPU的寄存器内识别异常上报的源和错误编码。
  - 无法上报的，记录在死机日志中。
  - 硬件死机现场状态保存，可以确定死机原因。

- 故障上报和记录可靠性。
  - 任何操作异常情况下，故障信息都能够被CPU或者某个模块读取，并且能够记录日志。
  - 如果上述中确实有无法保证系统不挂死的参数配置，必须加上配置开关保护，防止配置错误。
  - 如果上述还是不能保证的特例情况，将故障信息记录在复位不清除的位置，复位之后读取故障信息。
  - 出现异常或者发生特殊事件时，保持状态和事件等信息，软件可以查询和清除。
  - 当出现导致软件无法运行的异常时，需要硬件自主保存异常信息（非掉电不可失寄存器内）。软件复位后读取并清除该异常状态信息。
  - 系统复位前备份关键信息到存储空间中，在复位期间数据被保持不丢失。

- 故障上报信息标准。
  - 故障必须记录伴随信息、错误的次数和地址。
  - 故障按照影响明确分级，分为致命、严重、一般和提示，分别对应：系统不可用、功能不可用、单报文或单次操作错误和功能无影响。
  - 故障按照后处理标准化为暂定：系统复位、单模复位、软件恢复配置和提示告警。
  - 故障分类上报在寄存器接口级别规范化定义，软件可以直接读取信息进行判断处理。

 版权所有 © 海思技术有限公司

  - 总线异常时，硬件支持记录异常地址、ID、错误状态等定位信息。
- 故障恢复规范。
  - 故障恢复时间指标。
    - 芯片复位不超过5s。
    - 单模复位不超过10s。
    - 模块复位不超过100ms。
    - 单个核，单线程，单引擎复位的不超过100ms。
    - 软件修复配置参数的不超过100ms。
  - 故障恢复可靠性。故障恢复措施在逻辑实现的部分，必须保证可以完成。
  - 不同层次快速复位功能。Hi309a上承载的传输和交换功能，支持这些功能在线程、ATOM、Core及加速器在不同场景下不同层次的快速复位、数据面整体复位和业务恢复，快速复位业务中断时间不多于1ms，整体复位时间不多于2s。
  - 单核和单模块复位加载，支持各核和IP：
    - 复位、加载、调试、使能和去使能。
    - 任意核运行异常时，不影响其他核的运行，可单独复位、加载、调试、使能和去使能。
    - 各模块复位要安全进行，避免相关关联导致的影响和复位范围不合理。
    - 单核复位时，要求共享资源可回收。
    - SPPE支持除了支持单模复位，还要支持设备级独立复位，1ms之内完成。
- 支持NP基于ATOM、线程的复位及SPPE与关键加速器的整体复位功能
  - 支持基于线程的独立复位功能，线程复位不影响业务运行，文无丢失。
  - 支持基于ATOM的独立复位功能，ATOM复位不影响其他ATOM的运行，在典型流量下报文无丢失。
  - 支持基于NP系统及与之相关高速通道加速器的整体复位功能，NP整体复位影响Core的运行，业务中断时间控制在10ms以内。
- 故障预防规范。
  - 输入输出参数检查：
    - 静态参数：
      - 错误后果严重的静态配置参数，必须增加32bit的PATTERN 开关保护，防止误配置。
      - 存放在MEM中的参数，开关难以保护的，支持校验，防止参数被误踩。
      - 配置参数的检查，只对可能导致模块功能完全不可用，或者系统挂死的进行检查。并且必须在配置流程里，体现在状态寄存器上，供配置软件查询。
    - 动态参数：

版权所有 © 海思技术有限公司

    - CPU动态输入模块的参数，可能导致模块功能完全不可用，或者系统挂死的必须检查并告警。
    - 加速器动态输入模块的参数，可能导致模块功能完全不可用，或者系统挂死的必须检查并告警。
    - 动态参数经过外部MEM存储过的，支持校验，防止误踩。
  - 支持模块间协同的流控反压机制。
  - HAC支持参数配置保护，确保任意配置不会导致HAC挂死不可恢复。
  - 软件操作流程可靠性，操作流程必须状态可控，闭环：
    - 操作流程都必须有明确的完成状态，不允许有DELAY一个不固定完成时间的配置要求。DELAY时间只能用来做异常保护。
    - 状态指示需要明确区分：成功/DONE ；失败。失败之后给出明确的故障分类原因。
  - KPI指标预警，触发预防措施，功能降额使用。
  - 内存防踩保护。
    - DDR防踩内存保护：支持检测到外部DDR空间越界时报中断，同时如果后续再进来有写操作则反馈error，不会毁坏其他DDR空间的数据。
    - 提供对关键内存区域的保护：关键内存区域是指存放重要业务数据或系统数据的内存区域（如内存控制块、消息控制块、队列控制块、定时器控制块等内存区域），需要对越界和非法写操作等异常操作进行预防。要求对产品设计规格中指定的内存区域可以预防测试中模拟的越界和非法写操作对内存的破坏，并对异常操作进行记录。
  - 错误输入不能伤及硬件。
    - 未初始化访问不能挂死。
    - 硬件管理内存增加校验（如各类溢出空间，内部ram等）；
    - 各种错误都有故障信息记录，便于快速定位原因。
    - 故障隔离及恢复（故障后影响降低到最低，能在线自愈修复）。
    - 错误输入不能伤及硬件（正确错误包混合输入，硬件都能处理正确，不能一个错包后，正确的包也不能正确处理）。
    - 出错后能及时上报，无法中断上报的，采用硬件定期巡检。
  - HAC支持业务相关维测信息统计与输出，业务相关的硬件加速器支持业务相关的维测信息统计与输出。
- 支持故障隔离规范。
  - 错误输入不能扩散：正确错误混合输入，正确的能正确处理，错误的输入不能导致后面正确的报文不能正确处理。
  - 故障后影响降低到最低，能够以最低代价恢复。
  - 加速器模块必须对输入进行合法性检查，支持模块间协同的流控反压机制，接口器件异常的保护处理。
- 支持过载控制，芯片各模块支持过载状态获取、上报、反压、流控处理。
  - 支持软件获取I/O器件的负载状态，当CPU过载时，软件执行对入口任务添加的反压处理。

版权所有 © 海思技术有限公司

  - 支持软件获取CPU负载状态，当CPU过载时，软件执行对入口任务添加的反压处理。
  - 支持软件获取总线负载状态，包括流量、时延。
  - 支持软件获取加速器负载状态；支持当加速器过载时，软件执行对入口任务添加的反压处理。
- 芯片接口总线防挂死功能，支持独立复位：芯片接口故障不扩散，不导致系统（包括框内或者跨框）其他单板故障。
- 芯片支持crack检测，便于在生产装备环节检测芯片裂纹问题。
- 芯片可以不借助片外其他器件实现对内部模块功能性检测，也可做故障注入，并检测，测试完毕后恢复。
- 支持MACSEC的装备对接测试，在无对接设备的情况下支持自测试功能。
- 在Hi309a的SYS_CTRL的寄存器中实现IDCODE以提供芯片制造及版本信息，CPU/JTAG只读。
  - Serdes提供业务端口（或通道）单独打开/关闭的功能。
  - Serdes提供校验和错误处理机制,例如Serdes接口链路校验。
  - Serdes提供 PLL锁定指示当前状态，芯片顶层逻辑并记录其历史状态信号寄存器可查询，且历史。
  - 从管脚把SerDes每条链路的恢复时钟引出芯片观测。
- 对复位寄存器增加访问控制功能，控制写权限避免误操作。
- 片内温度阈值门限可配置，当温度达到此门限时，芯片会自动上报中断。
- 支持将SerDes的输出接口rxoclk（最高）在顶层要做一个分频，通过GPIO引出去芯片可观测。
- 支持eFuse。
  - 支持eFuse信息采用double bit。
  - 支持安全efuse烧写后不需要芯片复位即可回读确认安全efuse烧写值，efuse烧写流程需要考虑便于装备测试。

11.4 遵循协议

IEEE 1149.1/6。

11.5 功能描述

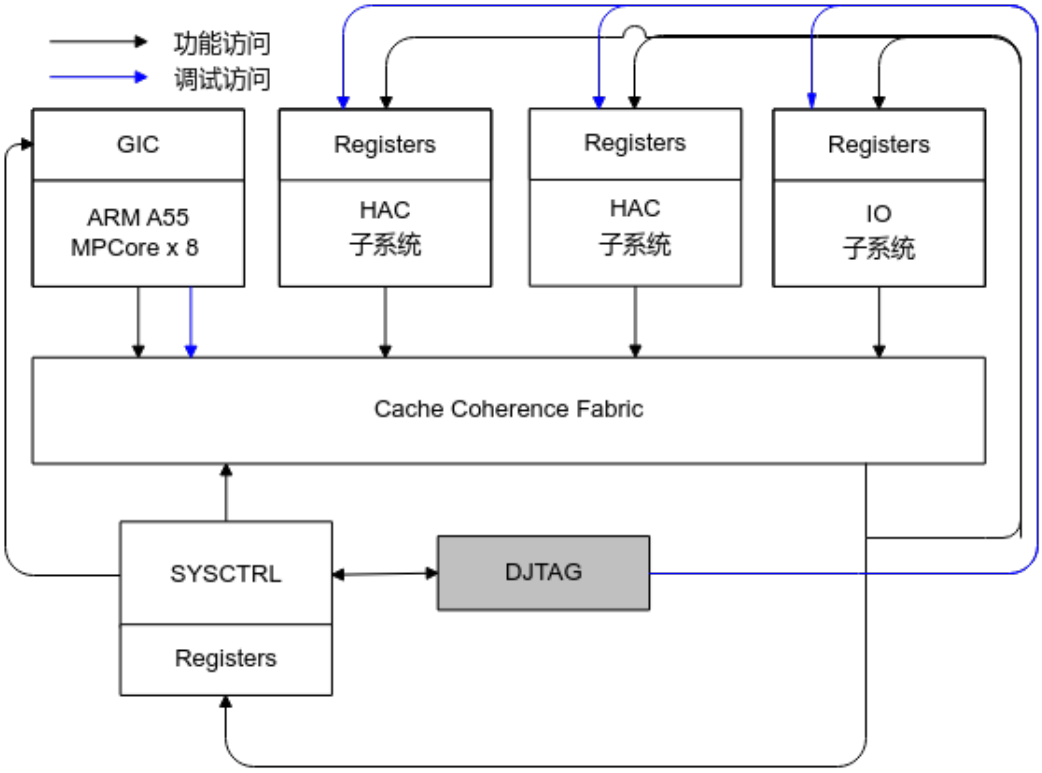
无。

11.5.1 应用框图

系统DFX的应用如图11-1所示。

版权所有 © 海思技术有限公司

图 11-1 系统 DFX 应用图



芯片集成了JTAG和DJTAG调试模块。Fabric，HAC子系统和IO子系统内的各个模块都包含Debug Bus模块。DJTAG和Debug bus是独立于功能总线的调试系统，外部仿真器通过DJTAG接口可以对集成Debug bus的模块进行实时检测和调试。

11.5.2 相关模块

在系统中，系统DFX会和下面多个模块进行交互，分别描述如下：

- CPU子系统/ACC子系统/IO子系统：CPU子系统、HAC子系统和IO子系统都集成调试逻辑，可以完成整芯片，子系统和模块的功能、性能等调试、定位。
- 安全逻辑：安全逻辑通过等级设置、鉴权对系统DFX逻辑功能分级、使能和禁止。

11.5.3 内部信号监测

芯片有16个关键信号可以通过管脚观测，如**表11-2**所示。在Probe_sel（GPIO146，静态选择信号，只在复位信号撤离时采样输入信号决定是否将芯片内部的关键状态拉出来在管脚上可观测）有效时，probe信号输出到管脚probe_out15~probe_out0上。

当probe_sel无效（即不是出于probe模式）时，如需观测内部信号，可配置相应的IO复用寄存器。

版权所有 © 海思技术有限公司

表 11-2 probe 信号列表

probe信号	内部关键信号	外部管脚
probe_out0	clk_probe_grp[grp_index][0]	SCL1
probe_out1	clk_probe_grp[grp_index][1]	PMBUS0_DATA
probe_out2	clk_probe_grp[grp_index][2]	PMBUS0_CLK
probe_out3	clk_probe_grp[grp_index][3]	PMBUS1_DATA
probe_out4	clk_probe_grp[grp_index][4]	PMBUS1_CLK
probe_out5	clk_probe_grp[grp_index][5]	PMBUS2_DATA
probe_out6	clk_probe_grp[grp_index][6]	SPICK0
probe_out7	clk_probe_grp[grp_index][7]	SPIDI0
probe_out8	rst_probe_grp[grp_index][0]	SPIDO0
probe_out9	rst_probe_grp[grp_index][1]	SPICSN0_0
probe_out10	rst_probe_grp[grp_index][2]	SPICSN0_1
probe_out11	rst_probe_grp[grp_index][3]	SDA0
probe_out12	rst_probe_grp[grp_index][4]	SCL0
probe_out13	rst_probe_grp[grp_index][5]	SDA1
probe_out14	rst_probe_grp[grp_index][6]	URXD0
probe_out15	rst_probe_grp[grp_index][7]	UTXD0

表 11-3 probe signal 分配关系

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
0	pad_probe_out[0]	efuse_hard_repair_done0	efuse0加载完成指示。
	pad_probe_out[1]	efuse_hard_repair_done1	efuse1加载完成指示。
	pad_probe_out[2]	efuse_hard_repair_done2	efuse2加载完成指示。
	pad_probe_out[3]	-	-

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[4]	-	-
	pad_probe_out[5]	-	-
	pad_probe_out[6]	efuse_hard_repair_done_final	eFuse加载完成指示。
	pad_probe_out[7]	jtag_auth_djtag_disable	jtag 鉴权结果，disable djtag指示信号。
	pad_probe_out[8]	jtag_auth_dft_disable	jtag 鉴权结果，disable DFT指示信号。
	pad_probe_out[9]	probe_system_counter_flag	AP的system_counter计数到指定值的指示信号。
	pad_probe_out[10]	-	-
	pad_probe_out[11]	-	-
	pad_probe_out[12]	-	-
	pad_probe_out[13]	-	-
	pad_probe_out[14]	-	-
	pad_probe_out[15]	-	-
1	pad_probe_out[0]	clk_tcxo	晶振时钟。
	pad_probe_out[1]	clk_utmi_ref[0]	USB0控制器100M参考时钟。
	pad_probe_out[2]	clk_utmi_ref[1]	USB1控制器100M参考时钟。
	pad_probe_out[3]	clk_utmi_60m[0]	USB PHY0控制器60M时钟。
	pad_probe_out[4]	clk_utmi_60m[1]	USB PHY1控制器60M时钟。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[5]	clk_p0_utmi[0]	utmi接口采样时钟。
	pad_probe_out[6]	clk_p0_utmi[1]	utmi接口采样时钟。
	pad_probe_out[7]	pad_espi_sclk_in	espi3 slave模式输入时钟。
	pad_probe_out[8]	clk_mii_tx_in	mii tx时钟。
	pad_probe_out[9]	clk_mii_rx_in	mii rx时钟。
	pad_probe_out[10]	clk_rmii_ref_pre	rmii参考时钟。
	pad_probe_out[11]	clk_rgmii_mii_rx_in	rgmii rx时钟。
	pad_probe_out[12]	clk_rgmii_mii_tx_in	rgmii tx时钟。
	pad_probe_out[13]	clk_pll7_foutpostdiv	pll7输出时钟。
	pad_probe_out[14]	clk_pll6_foutpostdiv	pll6输出时钟。
	pad_probe_out[15]	clk_pll5_foutpostdiv	pll5输出时钟。
2	pad_probe_out[0]	clk_pll4_foutpostdiv	pll4输出时钟。
	pad_probe_out[1]	clk_pll3_foutpostdiv	pll3输出时钟。
	pad_probe_out[2]	clk_pll2_foutpostdiv	pll2输出时钟。
	pad_probe_out[3]	clk_pll1_foutpostdiv	pll1输出时钟。
	pad_probe_out[4]	clk_pll0_foutpostdiv	pll0输出时钟。
	pad_probe_out[5]	clk_pll_ddr_probe_out	ddr的pll输出。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[6]	clk_sds2_mclk[0]	sds mclk输出。
	pad_probe_out[7]	clk_sds2_mclk[1]	sds mclk输出。
	pad_probe_out[8]	clk_sds1_mclk[0]	sds mclk输出。
	pad_probe_out[9]	clk_sds1_mclk[1]	sds mclk输出。
	pad_probe_out[10]	clk_sds0_mclk[0]	sds mclk输出。
	pad_probe_out[11]	clk_sds0_mclk[1]	sds mclk输出。
	pad_probe_out[12]	clk_sds2_rxoclk[0]	sds rxo恢复时钟。
	pad_probe_out[13]	clk_sds2_rxoclk[1]	sds rxo恢复时钟。
	pad_probe_out[14]	clk_sds2_rxoclk[2]	sds rxo恢复时钟。
	pad_probe_out[15]	clk_sds2_rxoclk[3]	sds rxo恢复时钟。
3	pad_probe_out[0]	clk_sds1_rxoclk[0]	sds rxo恢复时钟。
	pad_probe_out[1]	clk_sds1_rxoclk[1]	sds rxo恢复时钟。
	pad_probe_out[2]	clk_sds0_rxoclk[0]	sds rxo恢复时钟。
	pad_probe_out[3]	clk_sds0_rxoclk[1]	sds rxo恢复时钟。
	pad_probe_out[4]	clk_sds0_rxoclk[2]	sds rxo恢复时钟。
	pad_probe_out[5]	clk_sds0_rxoclk[3]	sds rxo恢复时钟。
	pad_probe_out[6]	rtc_25m_clkin	25M输入时钟。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[7]	clk_sds0_core_clk	sds0 core时钟。
	pad_probe_out[8]	clk_sds1_core_clk	sds1 core时钟。
	pad_probe_out[9]	clk_sds2_core_clk	sds2 core时钟。
	pad_probe_out[10]	-	-
	pad_probe_out[11]	-	-
	pad_probe_out[12]	-	-
	pad_probe_out[13]	-	-
	pad_probe_out[14]	-	-
	pad_probe_out[15]	-	-
4	pad_probe_out[0]	pll7_lock	pll7锁定指示。
	pad_probe_out[1]	pll6_lock	pll6锁定指示。
	pad_probe_out[2]	pll5_lock	pll5锁定指示。
	pad_probe_out[3]	pll4_lock	pll4锁定指示。
	pad_probe_out[4]	pll3_lock	pll3锁定指示。
	pad_probe_out[5]	pll2_lock	pll2锁定指示。
	pad_probe_out[6]	pll1_lock	pll1锁定指示。
	pad_probe_out[7]	pll0_lock	pll0锁定指示。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[8]	fn_pll_ddr_lock	DDR的FNPLL锁定指示。
	pad_probe_out[9]	sc_pll_ddr_lock	DDR的SCPLL锁定指示。
	pad_probe_out[10]	-	-
	pad_probe_out[11]	-	-
	pad_probe_out[12]	-	-
	pad_probe_out[13]	-	-
	pad_probe_out[14]	-	-
	pad_probe_out[15]	0b0	-
5	pad_probe_out[0]	rst_jtag_auth_comb_n	jtag auth combo复位。
	pad_probe_out[1]	rst_jtag_auth_pow_n	jtag auth power复位。
	pad_probe_out[2]	porn_in	porn_in输入。
	pad_probe_out[3]	wdg_mr	wdg mr复位。
	pad_probe_out[4]	rstn_in	rstn_in复位输入。
	pad_probe_out[5]	rstn_can_ctrl	rstn_can_ctrl。
	pad_probe_out[6]	rstn_button	rstn_button。
	pad_probe_out[7]	rstn_wdog	rstn_wdog。
	pad_probe_out[8]	rstn_user_define	rstn_user_define。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[9]	rstn_out_chip_n	rstn_out_chip_n。
	pad_probe_out[10]	rst_power_n	POWER复位。
	pad_probe_out[11]	rst_glb_sft_n	全局软复位。
	pad_probe_out[12]	inter_wdg_srst_n	内部狗复位。
	pad_probe_out[13]	rstn_ext_active_src_dly	外部复位。
	pad_probe_out[14]	hard_repair_done	eFuse 加载完成指示。
	pad_probe_out[15]	rst_por_n	POR复位。
6	pad_probe_out[0]	rst_comb_n	combo复位。
	pad_probe_out[1]	glb_srst_req	全局软复位请求。
	pad_probe_out[2]	ap_srst_req	ap软复位请求。
	pad_probe_out[3]	rst_alone_n	独立复位指示。
	pad_probe_out[4]	0b0	-
	pad_probe_out[5]	0b0	-
	pad_probe_out[6]	0b0	-
	pad_probe_out[7]	0b0	-
	pad_probe_out[8]	0b0	-
	pad_probe_out[9]	0b0	-

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[10]	0b0	-
	pad_probe_out[11]	0b0	-
	pad_probe_out[12]	0b0	-
	pad_probe_out[13]	0b0	-
	pad_probe_out[14]	0b0	-
	pad_probe_out[15]	0b0	-
7	pad_probe_out[0]	0b0	-
	pad_probe_out[1]	0b0	-
	pad_probe_out[2]	0b0	-
	pad_probe_out[3]	0b0	-
	pad_probe_out[4]	0b0	-
	pad_probe_out[5]	0b0	-
	pad_probe_out[6]	0b0	-
	pad_probe_out[7]	0b0	-
	pad_probe_out[8]	0b0	-
	pad_probe_out[9]	0b0	-
	pad_probe_out[10]	probe_pfo_n_test[0]	dying gasp输出测试。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[11]	probe_pfo_n_test[1]	dying gasp输出测试。
	pad_probe_out[12]	usb2phy0_ref_output_clk	已经给出USB2.0的参考钟。
	pad_probe_out[13]	usb2phy1_ref_output_clk	已经给出USB2.0的参考钟。
	pad_probe_out[14]	-	-
	pad_probe_out[15]	-	-
8	pad_probe_out[0]	probe_bootrom_ecc_error	bootrom ecc错。
	pad_probe_out[1]	probe_bootrom_multi_err	bootrom多bit错误。
	pad_probe_out[2]	-	-
	pad_probe_out[3]	-	-
	pad_probe_out[4]	-	-
	pad_probe_out[5]	-	-
	pad_probe_out[6]	-	-
	pad_probe_out[7]	-	-
	pad_probe_out[8]	-	-
	pad_probe_out[9]	-	-
	pad_probe_out[10]	-	-
	pad_probe_out[11]	-	-

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[12]	-	-
	pad_probe_out[13]	-	-
	pad_probe_out[14]	-	-
	pad_probe_out[15]	-	-
9	pad_probe_out[0]	sc_hilink0_lossofrefclk[0]	serdes参考钟丢失。
	pad_probe_out[1]	sc_hilink0_lossofrefclk[1]	serdes参考钟丢失。
	pad_probe_out[2]	sc_hilink1_lossofrefclk[0]	serdes参考钟丢失。
	pad_probe_out[3]	sc_hilink1_lossofrefclk[1]	serdes参考钟丢失。
	pad_probe_out[4]	sc_hilink2_lossofrefclk[0]	serdes参考钟丢失。
	pad_probe_out[5]	sc_hilink2_lossofrefclk[1]	serdes参考钟丢失。
	pad_probe_out[6]	probe_usb_test_bus_hst[0]	usb测试。
	pad_probe_out[7]	probe_usb_test_bus_hst[1]	usb测试。
	pad_probe_out[8]	probe_usb_test_bus_hst[2]	usb测试。
	pad_probe_out[9]	probe_usb_test_bus_hst[3]	usb测试。
	pad_probe_out[10]	probe_usb_test_bus_drd[0]	usb测试。
	pad_probe_out[11]	probe_usb_test_bus_drd[1]	usb测试。
	pad_probe_out[12]	probe_usb_test_bus_drd[2]	usb测试。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[13]	probe_usb_test_bus_drd[3]	usb测试。
	pad_probe_out[14]	-	-
	pad_probe_out[15]	-	-
10	pad_probe_out[0]	sc_hilink0_plloutoflock[0]	serdes pll失锁。
	pad_probe_out[1]	sc_hilink0_plloutoflock[1]	serdes pll失锁。
	pad_probe_out[2]	sc_hilink1_plloutoflock[0]	serdes pll失锁。
	pad_probe_out[3]	sc_hilink1_plloutoflock[1]	serdes pll失锁。
	pad_probe_out[4]	sc_hilink2_plloutoflock[0]	serdes pll失锁。
	pad_probe_out[5]	sc_hilink2_plloutoflock[1]	serdes pll失锁。
	pad_probe_out[6]	probe_clk_sds0_rxi[0]	sds rxi时钟。
	pad_probe_out[7]	probe_clk_sds0_rxi[1]	sds rxi时钟。
	pad_probe_out[8]	probe_clk_sds0_rxi[2]	sds rxi时钟。
	pad_probe_out[9]	probe_clk_sds0_rxi[3]	sds rxi时钟。
	pad_probe_out[10]	probe_clk_sds1_rxi[0]	sds rxi时钟。
	pad_probe_out[11]	probe_clk_sds1_rxi[1]	sds rxi时钟。
	pad_probe_out[12]	probe_clk_sds2_rxi[0]	sds rxi时钟。
	pad_probe_out[13]	probe_clk_sds2_rxi[1]	sds rxi时钟。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[14]	probe_clk_sds2_rxi[2]	sds rxi时钟。
	pad_probe_out[15]	probe_clk_sds2_rxi[3]	sds rxi。
11	pad_probe_out[0]	probe_serdes_clk_out0	同步以太输出。
	pad_probe_out[1]	probe_serdes_clk_out1	同步以太输出。
	pad_probe_out[2]	probe_serdes_clk_out2	同步以太输出。
	pad_probe_out[3]	0b0	-
	pad_probe_out[4]	0b0	-
	pad_probe_out[5]	0b0	-
	pad_probe_out[6]	0b0	-
	pad_probe_out[7]	clk_csys_trace	cpu cluster0 debug clk。
	pad_probe_out[8]	clk_dbg_clk	cpu cluster0 debug clk。
	pad_probe_out[9]	csys_dbg_rst_n	cpu cluster0 复位测试点。
	pad_probe_out[10]	core_l2_rst_n	cpu cluster0 复位测试点。
	pad_probe_out[11]	core3_por_n	cpu cluster0 复位测试点。
	pad_probe_out[12]	core2_por_n	cpu cluster0 复位测试点。
	pad_probe_out[13]	core1_por_n	cpu cluster0 复位测试点。
	pad_probe_out[14]	core0_por_n	cpu cluster0 复位测试点。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[15]	csys_trace_rst_n	cpu cluster0 复位测试点。
12	pad_probe_out[0]	不使用	-
	pad_probe_out[1]	不使用	-
	pad_probe_out[2]	不使用	-
	pad_probe_out[3]	不使用	-
	pad_probe_out[4]	不使用	-
	pad_probe_out[5]	不使用	-
	pad_probe_out[6]	不使用	-
	pad_probe_out[7]	clk_csys_trace	cpu cluster0 debug clk。
	pad_probe_out[8]	clk_dbg_clk	cpu cluster0 debug clk。
	pad_probe_out[9]	csys_dbg_rst_n	cpu cluster1 复位测试点。
	pad_probe_out[10]	core_l2_rst_n	cpu cluster2 复位测试点。
	pad_probe_out[11]	core3_por_n	cpu cluster3 复位测试点。
	pad_probe_out[12]	core2_por_n	cpu cluster4 复位测试点。
	pad_probe_out[13]	core1_por_n	cpu cluster5 复位测试点。
	pad_probe_out[14]	core0_por_n	cpu cluster6 复位测试点。
	pad_probe_out[15]	csys_trace_rst_n	cpu cluster7 复位测试点。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
13	pad_probe_out[0]	不使用	-
	pad_probe_out[1]	不使用	-
	pad_probe_out[2]	不使用	-
	pad_probe_out[3]	不使用	-
	pad_probe_out[4]	不使用	-
	pad_probe_out[5]	不使用	-
	pad_probe_out[6]	不使用	-
	pad_probe_out[7]	cmd0_boot_start	EMMC内部测试点。
	pad_probe_out[8]	auto_resp_done	EMMC内部测试点。
	pad_probe_out[9]	cmd_send_done_tmp	EMMC内部测试点。
	pad_probe_out[10]	cmd_resp_done	EMMC内部测试点。
	pad_probe_out[11]	cmd_fsm_state[0]	EMMC内部测试点。
	pad_probe_out[12]	cmd_fsm_state[1]	EMMC内部测试点。
	pad_probe_out[13]	cmd_fsm_state[2]	EMMC内部测试点。
	pad_probe_out[14]	auto_cmd_start	EMMC内部测试点。
	pad_probe_out[15]	cmd_start	EMMC内部测试点。
14	pad_probe_out[0]	cmd_valid_ff2	EMMC内部测试点。

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[1]	int_hiss[0]	hiss中断。
	pad_probe_out[2]	int_hiss[1]	hiss中断。
	pad_probe_out[3]	int_hiss[2]	hiss中断。
	pad_probe_out[4]	int_hiss[3]	hiss中断。
	pad_probe_out[5]	int_hiss[4]	hiss中断。
	pad_probe_out[6]	int_hiss[5]	hiss中断。
	pad_probe_out[7]	int_hiss[6]	hiss中断。
	pad_probe_out[8]	int_hiss[7]	hiss中断。
	pad_probe_out[9]	int_hiss[8]	hiss中断。
	pad_probe_out[10]	int_hiss[9]	hiss中断。
	pad_probe_out[11]	int_hiss[10]	hiss中断。
	pad_probe_out[12]	int_hiss[11]	hiss中断。
	pad_probe_out[13]	int_hiss[12]	hiss中断。
	pad_probe_out[14]	int_hiss[13]	hiss中断。
	pad_probe_out[15]	int_hiss[14]	hiss中断。
15	pad_probe_out[0]	int_hiss[15]	hiss中断。
	pad_probe_out[1]	0b0	-

版权所有 © 海思技术有限公司

probe分组（probe_sel）	probe信号	内部关键信号	测试点描述
	pad_probe_out[2]	0b0	-
	pad_probe_out[3]	0b0	-
	pad_probe_out[4]	0b0	-
	pad_probe_out[5]	0b0	-
	pad_probe_out[6]	0b0	-
	pad_probe_out[7]	0b0	-
	pad_probe_out[8]	0b0	-
	pad_probe_out[9]	0b0	-
	pad_probe_out[10]	0b0	-
	pad_probe_out[11]	0b0	-
	pad_probe_out[12]	0b0	-
	pad_probe_out[13]	0b0	-
	pad_probe_out[14]	0b0	-
	pad_probe_out[15]	0b0	-

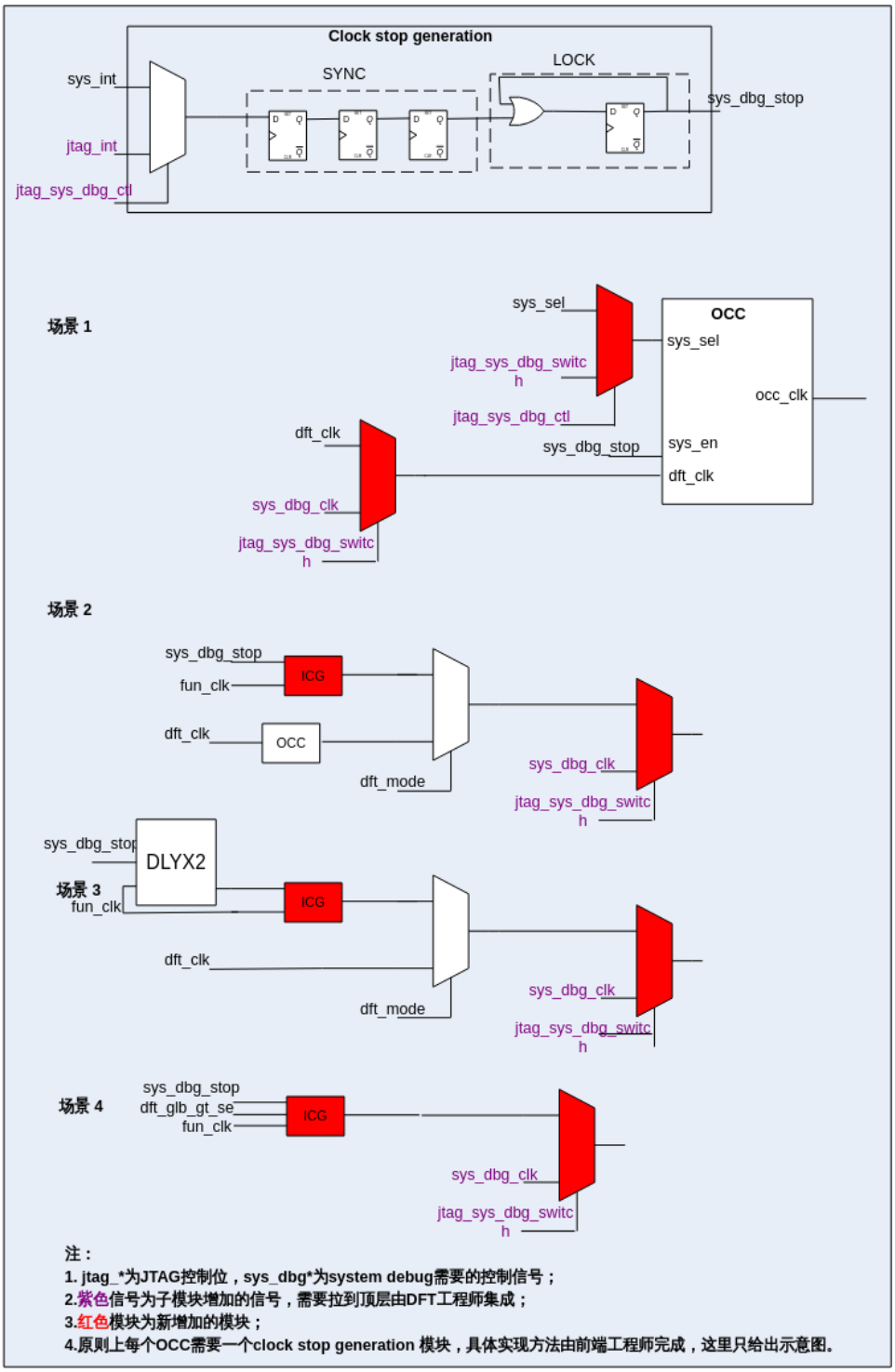
11.5.4 ALL-SCAN

在ALL-SCAN模式下，所有的系统时钟、模块主、从时钟都关闭，系统从功能模式切换到扫描模式，通过JTAG端口可以串行移出系统所有的FF的值，根据实现的复杂度，ALL-SCAN可以支持两种工作方式

版权所有 © 海思技术有限公司

- 不可恢复的ALL-SCAN，功能模式切到ALL-SCAN模式后，导出芯片FF的值后，芯片不能切回正常工作模式。

图 11-2 ALL-SCAN 时钟控制图



进入ALL_SCAN模式可以通过功能中断，也可以通过JTAG方式。两种方式进入ALL_SCAN测试模式的场景如图11-3所示。

版权所有 © 海思技术有限公司

图 11-3 ALL-SCAN 配置流程

场景一		
all_scan测试流程：（由sys_int中断）		
	在dft_mode=0时，实现时钟关断，以及切换到sys_dbg_clk上	
1	sys_int由1->0	all_scan_ctrl寄存器默认值为0，选择sys_int实现时钟关断
2	jtag_sys_dbg_switch配置为1	occ前面的dft_clk切换到sys_dbg_clk
3	jtag_sys_dbg_ctrl配置为1	occ输出切换到sys_dbg_clk
4	kerneal_clk_ctrl配置为13'h0FC0	将CPU_CLUSTER内部rst_mbist_n置为无效
5	配置mode_ctrl[4:3]为2'b11	将dft_lgc_rst_n置为无效
	在dft_mode=1时，使能sys_dbg_shift实现移位过程	
6	mode_ctrl配置为11_1111_1000	将dft_mode、scan_mode、dft_oen_ctrl、dft_ctrl_mbist、edt_bypass置为1
7	配置clock_mesh寄存器	打开所有all_scan的时钟
8	配置IEEE1500 scan_mode为1	保证shift时候所有ICG都是打开的，mbist电路的复位是可控的
9	配置all_scan_ctrl寄存器	将需要测试模块的使能打开
10	配置sys_dbg_shift	实现移位过程

场景二		
all_scan测试流程：（由jtag_int中断）		
	在dft_mode=0时，实现时钟关断，以及切换到sys_dbg_clk上	
1	jtag_sys_dbg_ctrl配置为1	jtag_int默认值为0，实现时钟关断
2	jtag_sys_dbg_switch配置为1	occ前面的dft_clk切换到sys_dbg_clk，occ输出切换到sys_dbg_clk
3	kerneal_clk_ctrl配置为13'h0FC0	将CPU_CLUSTER内部rst_mbist_n置为无效
4	配置mode_ctrl[4:3]为2'b11	将dft_lgc_rst_n置为无效
	在dft_mode=1时，使能sys_dbg_shift实现移位过程	
5	mode_ctrl配置为11_1111_1000	将dft_mode、scan_mode、dft_oen_ctrl、dft_ctrl_mbist、edt_bypass置为1
6	配置clock_mesh寄存器	打开所有all_scan的时钟
7	配置IEEE1500 scan_mode为1	保证shift时候所有ICG都是打开的，mbist电路的复位是可控的
8	配置all_scan_ctrl寄存器	将需要测试模块的使能打开
9	配置sys_dbg_shift	实现移位过程

说明

上述图11-3中场景一第4步骤和场景二第3步骤，对kerneal_clk_ctrl处理Hi309a不涉及。

ALL-SCAN的实现可以支持在单板上跑ATPG，进一步提高ATPG pattern的测试质量。

绿色部分位于各个子系统，红色部分位于Hi309a顶层。TDI作为Scan Input，TDO作为Scan Output。

ALLSCAN模式，包括切换、移位、暂停三种过程。

切换过程：由功能模式切换到ALLSCAN模式直到稳定的过程；

移位过程：在ALLSCAN模式中，将内部寄存器值的移出的过程（类似scan的shift过程）；

暂停过程：在移位过程前后，可以暂停，等待任意时长，然后重新开始移位。

11.5.5 功能 MBIST

Hi309a支持功能MBIST测试功能。芯片的上电自检可以启动功能MBIST，也可以在系统调试时启动。在启动功能MBIST时，Hi309a不能进行正常业务。

说明

由于部分模块初始化后处于复位状态，在功能MBIST测试前，需要先要解除模块的复位，初始处于复位的模块请参考“**复位**”章节。

11.5.6 Memory 软失效保护

Hi309a各个内嵌SRAM的模块都支持Parity或ECC功能。

版权所有 © 海思技术有限公司

11.5.7 Crack 检测

Hi309a支持Die Crack、BUMP Crack和BALL Crack检测，其中DIE Crack和BUMP Crack由内部DCIP实现，BALL Crack依赖板级配合，应用方案与产品提供方案吻合。

Hi309a DCIP寄存器在sysctrl_cfg页签，基地址为0xFA0A0000。

11.5.7.1 正常功能验证

步骤1 配置时钟，0x390配置为0x5。

SC_DCIP_CLK_EN	SC_DCIP_CLK_EN的时钟使能寄存器。	0x390	32	0x00000005	reserved	31:3	-	0x00000000	保留
					icg_en_dcip_core	2	NREG_SET_WO	0x1	工作时钟使能 1：工作时钟被使能； 0：无意义（写0无效）。
					reserved	1	-	0x0	保留
					icg_en_dcip_bcbist0	0	NREG_SET_WO	0x1	工作时钟使能 1：工作时钟被使能； 0：无意义（写0无效）。

步骤2 复位解复位：先复位，0xD60配置为0x11;再解复位，0xD64配置为0x11。

SC_DCIP_RESET_REQ	DCIP的复位请求信号。	0xD60	32	0x00000011	reserved	31:5	-	0x0000000	保留
					srst_req_dcip_core	4	NREG_SET_WO	0x1	软复位请求 1：软复位请求有效； 0：无意义（写0无效）。
					reserved	3	-	0x0	保留位。
					reserved	2	-	0x0	保留位。
					reserved	1	-	0x0	保留位。
					srst_req_dcip_bcbist0	0	NREG_SET_WO	0x1	软复位请求 1：软复位请求有效； 0：无意义（写0无效）。
SC_DCIP_RESET_DREQ	DCIP的复位去请求信号。	0xD64	32	0x00000000	reserved	31:5	-	0x0000000	保留
					srst_dreq_dcip_core	4	NREG_CLR_WO	0x0	软复位请求撤离 1：去复位请求有效，逻辑撤离复位； 0：无意义（写0无效）。
					reserved	3	-	0x0	保留位。
					reserved	2	-	0x0	保留位。
					reserved	1	-	0x0	保留位。
					srst_dreq_dcip_bcbist0	0	NREG_CLR_WO	0x0	软复位请求撤离 1：去复位请求有效，逻辑撤离复位； 0：无意义（写0无效）。

步骤3 配置ENABLE有效。

0xF200 bit[2:0]配置为3b111，其他bit为默认值；

0xF204 保持默认值，不需要配置；

0xF208 BIT4配置为1b1，其他bit为默认值。

SC_DCIP_CORE_CTRL1	DCIP CORE控制寄存器1	0xF200	32	0x00000000	reserved	31:21	-	0x000	保留
					dff_ignore_core	20:7	RW	0x0000	DFF模式有效，忽略检测当前所有的DFF电路检测。设置从1开始，0表示不屏蔽。
					detect_period_core	6:5	RW	0x0	Crack检测的频度（相当于50M）： 00：0ms； 01：1ms； 10：100ms； 11：1000ms；
					clk_div_cfg_core	4:3	RW	0x0	时钟分频，debug时使用： 2'b00:1分频； 2'b01:4分频； 2'b10:16分频； 2'b11:64分频；
					detect_enable_core	2:0	RW	0x0	Crack检测使能信号： bit0：inner DC； bit1：mid DC or DFF； bit2：outer DC； 注意：CORE DC与DFF分开测试，都先配置成3'b5，查看flag，然后配置成3'b2，查看flag

SC_DCIP_BIST_CTRL0	DCIP BIST控制寄存器	0xF208	32	0x00000002	reserved	31:9	-	0x000000	保留
					detect_period_bist	8:7	RW	0x0	Crack检测的频度（相当于50M）： 00：0ms； 01：1ms； 10：100ms； 11：1000ms；
					clk_div_cfg_bist	6:5	RW	0x0	时钟分频，debug时使用： 2'b00:1分频； 2'b01:4分频； 2'b10:16分频； 2'b11:64分频；
					detect_enable_bist	4	RW	0x0	Bump crack检测使能信号，高有效： bit0：inner DC； bit1：mid DC or DFF； bit2：outer DC；
					bypass_enable_bist	3	RW	0x0	CORE bypass SENSOR进行测试，默认值0
					dc_tsel_bist	2:1	RW	0x1	DC方案档位，默认为2'b01，测试与用户至少覆盖2'b00与2'b01；
					force_err_enable_bist	0	RW	0x0	CORE模拟crack进行测试，默认值0；

版权所有 © 海思技术有限公司

步骤4 等待5ms，判断输出所有FLAG是否为0；0xF20C和0xF210为0时代表正常，为非0时代表异常。异常时，打印出这两个寄存器值。

SC_DCIP_ST1	DCIP状态寄存器1	0xF20C	32	0x00000000	reserved	31:30	-	0x0	保留
					test_out_core	29:22	RO	0x00	TEST OUT，预留；
					dff_state_core	21:20	RO	0x0	指示crack导致DFF电路失效状态： 00：无crack，功能正常； 01：timing_out； 10：read 0 fail； 11：read 1 fail；
					dff_location_core	19:6	RO	0x0000	DFF模式下有效，指示发生crack发生的全局位置；精度10u
					mid_flag_now_core	5	RO	0x0	mid DC/DFF crack实时标志位；
					mid_flag_core	4	RO	0x0	mid DC/DFF crack历史标志位；
					outer_flag_now_core	3	RO	0x0	outer DC crack实时标志位；
					outer_flag_core	2	RO	0x0	outer DC crack历史标志位；
					inner_flag_now_core	1	RO	0x0	inner DC crack实时标志位；
					inner_flag_core	0	RO	0x0	inner DC crack历史标志位；
SC_DCIP_ST2	DCIP状态寄存器2	0xF210	32	0x00000000	reserved	31:2	-	0x00000000	保留
					flag_now_bcbist	1	RO	0x0	实时标志位；
					flag_bcbist	0	RO	0x0	历史标志位；

步骤5 返回测试结果。（如果是提供被装备测试用的用例，在异常时，需要将出错的0xF20C/0xF210两个寄存作为errinfo信息传到HUMP系统）

----结束

11.5.7.2 强制注错功能验证

步骤1 配置时钟，0x390配置为0x5。

SC_DCIP_CLK_EN	SC_DCIP_CLK_EN的时钟使能寄存器。	0x390	32	0x00000005	reserved	31:3	-	0x00000000	保留
					icg_en_dcip_core	2	MREG_SET_WO	0x1	工作时钟使能 1：工作时钟使能有效； 0：无意义（写0无效）。
					reserved	1	-	0x0	保留
					icg_en_dcip_bcbist0	0	MREG_SET_WO	0x1	工作时钟使能 1：工作时钟使能有效； 0：无意义（写0无效）。

步骤2 复位解复位：先复位，0xD60配置为0x11;再解复位，0xD64配置为0x11。

SC_DCIP_RESET_REQ	DCIP软复位请求信号。	0xD60	32	0x00000011	reserved	31:5	-	0x00000000	保留
					srst_req_dcip_core	4	MREG_SET_WO	0x1	软复位请求 1：软复位请求有效； 0：无意义（写0无效）。
					reserved	3	-	0x0	保留位。
					reserved	2	-	0x0	保留位。
					reserved	1	-	0x0	保留位。
					srst_req_dcip_bcbist0	0	MREG_SET_WO	0x1	软复位请求 1：软复位请求有效； 0：无意义（写0无效）。
SC_DCIP_RESET_DREQ	DCIP软复位去请求信号。	0xD64	32	0x00000000	reserved	31:5	-	0x00000000	保留
					srst_dreq_dcip_core	4	MREG_CLR_WO	0x0	软复位请求撤离 1：去复位请求有效，逻辑撤离复位； 0：无意义（写0无效）。
					reserved	3	-	0x0	保留位。
					reserved	2	-	0x0	保留位。
					reserved	1	-	0x0	保留位。
					srst_dreq_dcip_bcbist0	0	MREG_CLR_WO	0x0	软复位请求撤离 1：去复位请求有效，逻辑撤离复位； 0：无意义（写0无效）。

步骤3 配置force_err_enable有效, 0xF204 bit21配置为0b1；0xF208 bit0配置为0b1（跟正常的正常功能验证步骤相比，增加的配置）。

SC_DCIP_CORE_CTRL1	scDCIP CORE控制寄存器1	0xF204	32	0x00049068	reserved	31:30	-	0x0	保留
					test_in_core	29:22	RW	0x00	TEST IN，预留；
					force_err_enable_core	21	RW	0x0	CORE模拟crack进行测试，默认值0；
					bypass_enable_core	20	RW	0x0	CORE bypass SRIOR进行测试，默认值0
					dc_tsel_core	19:18	RW	0x1	DC方案档位，默认为2' b01，测试与用片里少覆盖2' b00与2' b01；
					dff_tsel_core	17:15	RW	0x1	DFF方案档位，默认为3' b001，测试与用片里少覆盖3' b000与3' b001；
					dff_test_s_core	14	RW	0x0	DFF模式下，当ting_out发生后，可以使用此功能定位具体的location；
					dff_limit_core	13:0	RW	0x1068	SRIOR电路中DFF电路的数量 默认值根据预留最真实的die size进行设置，其值如下：die周长/10u+100；

SC_DCIP_BIST_CTRL0	scDCIP BIST控制寄存器	0xF208	32	0x00000001	reserved	31:9	-	0x000000	保留
					detect_period_bist	8:7	RW	0x0	Crack检测的频度（标准于50M）： 00：0ns； 01：1ns； 10：100ns； 11：1000ns；
					clk_div_cfg_bist	6:5	RW	0x0	时钟分频，debug时使用： 2' b00:1分频； 2' b01:4分频； 2' b10:16分频； 2' b11:64分频；
					detect_enable_bist	4	RW	0x0	Bump crack检测使能信号，有效位： bit0：inner DC； bit1：mid DC or DFF； bit2：outer DC；
					bypass_enable_bist	3	RW	0x0	CORE bypass SRIOR进行测试，默认值0
					dc_tsel_bist	2:1	RW	0x1	DC方案档位，默认为2' b01，测试与用片里少覆盖2' b00与2' b01；
					force_err_enable_bist	0	RW	0x0	CORE模拟crack进行测试，默认值0；

版权所有 © 海思技术有限公司

步骤4 配置ENABLE有效:

0xF200 bit[2:0]配置为0b111，其他bit为默认值；

0xF204 保持默认值，不需要配置；

0xF208 bit4配置为0b1，其他bit为默认值。

SC_DCIP_CORE_CTRL1	HIDCIP CORE控制寄存器1	0xF200	32	0x00000000	reserved	31:21	-	0x000	保留
					dff_ignore_core	20:7	RW	0x0000	DFF模式有效，忽略检查当前所有的DFF电路检测。设置从1开始，0表示不忽略；
					detect_period_core	6:5	RW	0x0	Crack检测的频度（相当于50U）： 00：0ms; 01：1ms; 10：100ms; 11：1000ms;
					clk_div_cfg_core	4:3	RW	0x0	时钟分频，debug时使用： 2' b00:1分频; 2' b01:4分频; 2' b10:16分频; 2' b11:64分频;
					detect_enable_core	2:0	RW	0x0	Crack检测使能信号 ： bit0：inner DC; bit1：mid DC or DFF; bit2：outer DC; 注意：CPU DC号和FF分开测试，都先配置成3' b0，查看flag，然后配置成3' b7，查看flag

SC_DCIP_BIST_CTRL0	HIDCIP BIST控制寄存器	0xF208	32	0x00000002	reserved	31:9	-	0x000000	保留
					detect_period_bist	8:7	RW	0x0	Crack检测的频度（相当于50U）： 00：0ms; 01：1ms; 10：100ms; 11：1000ms;
					clk_div_cfg_bist	6:5	RW	0x0	时钟分频，debug时使用： 2' b00:1分频; 2' b01:4分频; 2' b10:16分频; 2' b11:64分频;
					detect_enable_bist	4	RW	0x0	Bump crack检测使能信号，两有效 ： bit0：inner DC; bit1：mid DC or DFF; bit2：outer DC;
					bypass_enable_bist	3	RW	0x0	CDFE bypass SIn\SOn进行测试，默认值0
					dc_tsel_bist	2:1	RW	0x1	DC方案指位，默认为2' b01，测试专用产品必须是2' b00与2' b01；
					force_err_enable_bist	0	RW	0x0	CDFE模拟crack进行测试，默认值0；

步骤5 等待5ms，判断输出所有FLAG是否为1；0xF20C为0x3F和0x60E4为0x3时代表正常，其他值代表异常。

SC_DCIP_ST1	DCIP状态寄存器1	0xF20C	32	0x00000000	reserved	31:30	-	0x0	保留
					test_out_core	29:22	RO	0x00	TEST OUT，预留；
					dff_state_core	21:20	RO	0x0	指示crack导致DFF电路失效状态： 00：无crack，功能正常； 01：timing_out; 10：read 0 fail; 11：read 1 fail;
					dff_location_core	19:6	RO	0x0000	DFF模式下有效，指示发生crack发生的全局位置，精度10u
					mid_flag_now_core	5	RO	0x0	mid DC/DFF crack实时标志位；
					mid_flag_core	4	RO	0x0	mid DC/DFF crack历史标志位；
					outer_flag_now_core	3	RO	0x0	outer DC crack实时标志位；
					outer_flag_core	2	RO	0x0	outer DC crack历史标志位；
					inner_flag_now_core	1	RO	0x0	inner DC crack实时标志位；
					inner_flag_core	0	RO	0x0	inner DC crack历史标志位；
SC_DCIP_ST2	DCIP状态寄存器2	0xF210	32	0x00000000	reserved	31:2	-	0x00000000	保留
					flag_now_bchist	1	RO	0x0	实时标志位；
					flag_bchist	0	RO	0x0	历史标志位；

步骤6 返回测试结果。

----结束

11.6 软件指南

11.6.1 Debug Bus 写 JTAG 寄存器

Debug Bus配置JTAG寄存器经过以下步骤：

步骤1 通过JTAG管脚TCK，TMS控制TAP进入指令加载通路，把“100001”DBG_PTS这样的指令通过TDI管脚串行移入指令寄存器。

步骤2 指令寄存器译码后选通DBG_PTS的Debug Bus通路，JTAG管脚TCK，TMS控制TAP进入数据加载通路，通过TDI把JTAG数据（寄存器地址）串行移入Debug Bus中。

版权所有 © 海思技术有限公司

步骤3 JTAG寄存器需要Debug Bus两次加载才能完成地址和数据的加载，重复步骤2，通过TDI把JTAG数据（寄存器数据）串行移入。

----结束

11.6.2 Debug Bus 读 JTAG 寄存器

Debug Bus读JTAG寄存器经过以下步骤：

步骤1 通过JTAG管脚TCK，TMS控制TAP进入指令加载通路，把“00101001”DBG_POE这样的指令通过TDI管脚串行移入指令寄存器。

步骤2 指令寄存器译码后选通DBG_POE的Debug Bus通路，JTAG管脚TCK，TMS控制TAP进入数据加载通路，通过TDI把JTAG数据（寄存器地址）串行移入Debug Bus中。

步骤3 地址并行输出到APB总线上，下一个APB周期把寄存器地址对应的寄存器值捕捉到JTAG寄存器中，通过TDO端口串行移出。

----结束

11.6.3 HiLink Serdes 功能 BIST

步骤1 配置内部环回模式：寄存器RX_CSR0的第10bit TXRXSERLPBKEN配置成1（如果不配置这个寄存器，Serdes工作在正常模式下，TX为串行输出端口，RX为串行输入端口）。

步骤2 RX寄存器CSR14的PRBSCHKEN配置成1，PRBSCHECKPATSEL设置为选择的PRBS测试向量。

步骤3 TX寄存器CSR2的TXDATASEL配置成“10”。

步骤4 TX CSR4的PRBSGENEN配置成1，使能产生PRBS，PRBSGENPATSEL模式选择，根据需求选择。

步骤5 等待一段时间后，查看RX CSR18寄存器检测PRBS校验结果。

说明

1. 测试需要保证时钟和IP引脚的配置正确。
2. 码产生和校验模式一致。

----结束

11.6.4 功能 MBIST 测试流程

测试MBIST前，注意模块的时钟、Memory门控和复位状态。

11.6.4.1 CPU Cluster MBIST 测试流程

以下测试过程中DDR不可用，所以以下测试程序需要运行在FLASH中或者内部SRAM。

一个CPU Cluster的MBIST测试配置由另外一个CPU Cluster完成。

步骤1 配置SysCtrl寄存器SC_MBIST_REQ为0x1。

步骤2 配置CPU_CLUSTER Subctrl寄存器SC_CLUS_MBIST_RESET_REQ为0x1。

步骤3 配置CPU_CLUSTER Subctrl寄存器SC_CLUS_MBIST_RESET_DREQ为0x1。

版权所有 © 海思技术有限公司

步骤4 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

步骤5 配置SysCtrl寄存器SC_CRG_RESET_DREQ中srst_dreq_func_mbist为0x1。

步骤6 配置SysCtrl寄存器SC_TOP_CRG_CLK_EN为0x1。

步骤7 配置SysCtrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x1。

步骤8 配置SysCtrl寄存器SC_MBIST_CPUI_ENABLE为0x1。

步骤9 等待200ns。

如下为Harden模块,如下如无特别说明，需要测试模块替换

CPU_CLUSTER0~CPU_CLUSTER1

步骤10 具体的测试向量，由DFT同事提供，软件按照向量步骤操作进行即可。

如下步骤为Restore流程：

步骤11 配置CPU_CLUSTER Subctrl寄存器SC_CLUS_MBIST_RESET_REQ为0x1。

步骤12 配置Sysctrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x0。

步骤13 配置Sysctrl寄存器SC_MBIST_CPUI_ENABLE为0x0。

步骤14 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

----结束

11.6.4.2 LLC MBIST 测试流程

步骤1 配置SysCtrl寄存器CFG_MBIST_LLC中srst_req_func_mbist为0xFF。

步骤2 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

步骤3 配置SysCtrl寄存器SC_CRG_RESET_DREQ中srst_dreq_func_mbist为0x1。

步骤4 配置SysCtrl寄存器SC_TOP_CRG_CLK_EN为0x1。

步骤5 配置SysCtrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x1。

步骤6 配置SysCtrl寄存器SC_MBIST_CPUI_ENABLE为0x1。

等待200ns。

向量运行阶段。

步骤7 具体的测试向量，由DFT同事提供，软件按照向量步骤操作进行即可。

Restore阶段。

步骤8 配置SysCtrl寄存器SC_MBIST_CPUI_ENABLE为0x0。

步骤9 配置SysCtrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x0。

步骤10 配置SysCtrl寄存器SC_TOP_CRG_CLK_EN为0x0。

步骤11 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

----结束

版权所有 © 海思技术有限公司

11.6.4.3 DDRC BIST 测试流程

步骤1 配置DDRC TOP寄存器DDRC_CFG_MBIST中mbist_rank为0x3。

步骤2 配置DDRC DMC0/1寄存器DDRC_CFG_CLKON（寄存器地址0x FB034240/0xFB03C240）为0x3FF。

步骤3 配置DDRC MRCR寄存器MRCR_CNTRL0（寄存器地址0x FB036000/0xFB03E000）中cfg_mrcr_ck_en为0x1。

步骤4 配置DDRC RASC寄存器RASC_CFG_MODE（寄存器地址0x FB035020/0xFB03D020）中cfg_mrcr_ck_en为0x1。

步骤5 配置DDRC SCTRL寄存器UCE_CRG_GT_EN为0xFFFF。

步骤6 配置DDRC SCTRL寄存器UCE_CRG_RST_EN为0x1。

步骤7 配置DDRC SCTRL寄存器UCE_CRG_RST_DIS为0x1。

步骤8 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

步骤9 配置SysCtrl寄存器SC_CRG_RESET_DREQ中srst_dreq_func_mbist为0x1。

步骤10 配置SysCtrl寄存器SC_TOP_CRG_CLK_EN为0x1。

步骤11 配置SysCtrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x1。

步骤12 配置SysCtrl寄存器SC_MBIST_CPUI_ENABLE为0x1。

等待200ns。

向量运行阶段。

步骤13 具体的测试向量，由DFT同事提供，软件按照向量步骤操作进行即可。

Restore阶段。

步骤14 配置SysCtrl寄存器SC_MBIST_CPUI_ENABLE为0x0。

步骤15 配置SysCtrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x0。

步骤16 配置SysCtrl寄存器SC_TOP_CRG_CLK_EN为0x0。

步骤17 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

----结束

11.6.4.4 其他 SUBSYS BIST 测试流程

Before Init阶段，读时钟复位状态并保存。

步骤1 读PERI_SUBCTRL、POE_SUBCTRL、ETH_SUBCTRL、SPPE_SUBCTRL、HILINKMUX_SUBCTRL中各时钟复位状态寄存器（*_ICG_ST*、*RESET_ST*寄存器，*代表模块或编号），并保存相应状态。

Init阶段，打开模块时钟复位，和配置MBIST相关寄存器。

步骤2 配置PERI_SUBCTRL中所有时钟使能寄存器（SC_*_ICG_EN）和解复位寄存器（SC_*_RESET_DREQ）；具体可参考《Hi309a 寄存器手册》。

步骤3 配置POE_SUBCTRL中所有时钟使能寄存器（SC_*_ICG_EN）和解复位寄存器（SC_*_RESET_DREQ）；具体可参考《Hi309a 寄存器手册》。

版权所有 © 海思技术有限公司

步骤4 配置ETH_SUBCTRL中所有时钟使能寄存器（SC_*_ICG_EN）和解复位寄存器（SC_*_RESET_DREQ）；具体可参考《Hi309a 寄存器手册》。

步骤5 配置SPPE_SUBCTRL中所有时钟使能寄存器（SC_*_ICG_EN）和解复位寄存器（SC_*_RESET_DREQ）；具体可参考《Hi309a 寄存器手册》。

步骤6 配置HILINKMUX_SUBCTRL中所有时钟使能寄存器（SC_*_ICG_EN）和解复位寄存器（SC_*_RESET_DREQ）；具体可参考《Hi309a 寄存器手册》。

步骤7 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

步骤8 配置SysCtrl寄存器SC_CRG_RESET_DREQ中srst_dreq_func_mbist为0x1。

步骤9 配置SysCtrl寄存器SC_TOP_CRG_CLK_EN为0x1。

步骤10 配置SysCtrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x1。

步骤11 配置SysCtrl寄存器SC_MBIST_CPUI_ENABLE为0x1。

等待200ns。

向量运行阶段

步骤12 具体的测试向量，由DFT同事提供，软件按照向量步骤操作进行即可。

Restore阶段

步骤13 配置SysCtrl寄存器SC_MBIST_CPUI_ENABLE为0x0。

步骤14 配置SysCtrl寄存器SC_MBIST_CPUI_SMS_FUNC_RESET为0x0。

步骤15 配置SysCtrl寄存器SC_TOP_CRG_CLK_EN为0x0。

步骤16 配置SysCtrl寄存器SC_CRG_RESET_REQ中srst_req_func_mbist为0x1。

时钟复位Restore阶段

步骤17 根据保存的PERI_SUBCTRL、POE_SUBCTRL、ETH_SUBCTRL、SPPE_SUBCTRL时钟复位状态值，进行Restore配置。

----结束

CPUWrite（data）操作流程说明：

步骤1 配置 SysCtrl寄存器SC_MBIST_CPUI_DATAIN为data。

步骤2 等待40x2 ns，配置 SysCtrl寄存器SC_MBIST_CPUI_WREITE_EN为0x1。

步骤3 等待40x4ns，配置 SysCtrl寄存器SC_MBIST_CPUI_WREITE_EN为0x0。

步骤4 CPUWait（wait_cycle）操作流程说明。

步骤5 等待*wait_cycle*个functional MBIST clock。

步骤6 CPURead（exp_data）操作流程说明。

步骤7 读取 SysCtrl寄存器SC_MBIST_CPUI_DATAOUT寄存器，与exp_data比较。

----结束

版权所有 © 海思技术有限公司

11.6.5 ALL-SCAN 测试流程

步骤1 通过JTAG或CPU停止功能时钟（参考DFT方案和时钟方案），并等待稳定。

步骤2 通过JTAG配置dft_mode为1（此时dft_clk来自PAD，必须要保证PAD不跳变）。

步骤3 通过JTAG配制lpct_en为1，将寄存器时钟切换到lpct_dft_clk。

步骤4 JTAG TAP进入shift-DR状态，SE置高，打通移位链。

步骤5 移位时钟传入，开始移位，每个TCK周期移一位。

ALL-SCAN的测试流程采用JTAG-Studio和Chip Explorer工具。工具的使用流程参考JTAG-Studio和海思CAD ChipEX工具用户手册。

----结束

版权所有 © 海思技术有限公司

12 GIC

12.1 概述

海思GIC是芯片基于GIC Specification提出的一种全局中断管理架构，它主要包括如下功能：

1. 使能、禁用、选择所有的SGI、PPI、SPI以及LPI。
2. 支持将输入的中断路由到任一指定的CPU core。
3. 支持中断优先级配置
4. 支持ARM AArch64安全化扩展。
5. 支持ARM AArch64虚拟化扩展。

基于GIC Specification，海思GIC主要进行如下功能扩展：

扩展通过N/N路由模式上报SPI中断，详细描述见“**N/N SPI路由**”。

12.2 关键术语

表12-1的术语将在本章节表示特殊的含义。

表 12-1 GIC 关键术语描述

Terms	Description
GIC	Generic Interrupt Controller，特指ARM定义的通用中断控制器。
SGI	Software Generated Interrupt，软件中断，具体中断映射信息请参考**中断向量表**。
PPI	Private Peripheral Interrupt，私有外设中断，通过WireConnection连接的中断，只能发给特定CPU core，具体中断映射信息请参见“**中断向量表**”。
SPI	Shared Peripheral Interrupt，共享外设中断，具体中断映射信息请参见“**中断向量表**”。

版权所有 © 海思技术有限公司

Terms	Description
LPI	Locality-specific Peripheral Interrupt， 特定区域外设中断，具体中断映射信息请参见“**中断向量表**”。
SEI	System error interrupt，系统错误中断。
MSI	Message signaled interrupt，消息中断，基于信息形式上报的中断，本文的详细描述请参见“**GICD中断上报流程**”。
ISR	Interrupt Service Routine，中断服务程序。
ICT	Information Communication Technology，信息通信融合技术。
EOI	End of Interrupt，中断结束。
WireInterrupt	通过物理硬连线直接发出的中断信号。
ITS	Hyper Interrupt Translation Service，超级中断路由服务。
ITT	Interrupt Translation Table，中断转换表。
ITE	Interrupt Translation Entry，中断传递表项。
SRE	-
Cluster	多个CPU core组成的，复用相同L2 Cache的硬件集合，本手册提及的每个Cluster包括4个CPU core组成。
Device	CPU core以外能产生中断请求的周边设备。
Symmetric Distributed Architecture	表示一个功能特性由一个或多个相似或相同的实例来实现，每个实例间是对等的。
MonolitGIC Architecture	和Symmetric Distributed Architecture相对，表示一个功能特性由一个完整的实例实现。
GIC Specification	ARM GIC Architecture Specification，ARM全局中断控制器设计规范。
GICV3	指ARM GIC Specification第三版文档。
GICV2	指ARM GIC Specification第二版文档。
PERI Subctrl	GICD硬件所在的子系统的子控制器模块。
TargetList	表示多个中断目标CPU core的一个合集，可能采用OneHot编码或者多组地址合并表示多个目标CPU core。
CPU core	中央处理器核，处理器中用于指令解码、执行的核心硬件电路集合单元。
Affinity Routing	亲和路由，本文的详细描述见“**GICD亲和路由**”。
ARE	Affinity Routing Enable，亲和路由使能。

版权所有 © 海思技术有限公司

Terms	Description
OneHot	独热码，一种按bit位表示多个属性状态的编码方式，每个属性的真假由对应bit位值的1或0表示。
IRM	Interrupt Routing Mode，中断路由模式，路由配置寄存器中的一个bit位，用于指示具体路由模式，本文的详细描述请参见“**GICD亲和路由**”。
IRQ Domain	作为一个多个中断的集中管理模块，拥有自己的子中断映射空间，负责子中断的分配、映射、翻译，统一响应属于该空间内的所有中断调用，经过适当转换后调用对应的子中断处理函数，详细内容请参看内核源码*kernel/irq/irqdomain.c*。
Re-Distributor	重分发器。
SCL	Super cluster。
SCCL	含义同socket。
ECAM	Enhanced Configuration Access Mechanism。

12.3 特性

GIC具有如下特性：

- GIC支持以下中断类型：软件中断SGIs、私有外设中断PPIs、共享外设中断SPIs，不支持LPI\vLPI（文中LPI\vLPI和ITS相关内容请忽略）。
- GIC.SPIs支持电平敏感和边沿触发方式中断，中断源的触发方式可单独配置，电平触发方式时高电平有效，表示有中断产生，边沿触发方式时为上升沿表示有中断产生。
- GIC支持物理中断虚拟化，经过Hypervisor调度之后，由GICC根据配置产生虚拟中断Virtual FIQ和Virtual IRQ。
- GIC支持虚拟中断直接上报到Guest OS。Guest OS在位时，虚拟中断可以直接上报到Guest OS；Guest OS不在位时，需要Hypervisor配合唤醒目标Guest OS，完成虚拟中断发送到Guest OS。
- GIC每个SCCL一个GICD，每个GICD支持最多32核的中断分发调度，系统支持最多8个SCCL，对于多GICD的系统，软件需配置主GICD。
- GIC支持SGI/SPI中断通过路由配置路由到不同的CPU Core，每个中断有独立的路由配置，SGI/SPI支持IRM为0和IRM为1的路由模型。
- 系统内主GICD控制1-N SPI中断的配置，并广播给其他从GICD。
- GICSPI支持MSI方式上报到GICD，兼容PCIe系统的MSI/MSI-X中断。
- GIC支持中断单独使能，支持中断状态设置、清除。
- 支持中断状态查询。
- GIC物理中断支持优先级屏蔽，屏蔽优先级门限（Mask Priority）由CPU配置，优先级高于门限的中断才能上报。在GICR硬件，虚拟中断不支持中断优先级屏蔽，虚拟中断的优先级屏蔽是在CPU Core中实现的。

版权所有 © 海思技术有限公司

- GIC支持中断嵌套，对物理中断，中断嵌套的深度由Group Priority决定，只有Group Priority较高的中断才能引发中断嵌套，Group Priority的级数可配，最多支持32级中断嵌套。
- GIC每个核提供内部器件的PPI中断，各核之间PPI中断相互独立，当前实现的PPI ID范围为20~31。
- GIC系统支持最多640个SPI中断，其中608个SPI可工作在1-N路由模式，N-N路由模式SPI要求是边沿触发类型中断，由GICD_ITARGETR寄存器配置中断路由。
- 系统内每个GICD单独配置自己的N-N SPI路由，软件可见所有N-N配置。
- 系统内每个GICD单独维护自己的N-N SPI的pending状态，根据路由配置向亲和DIE上报中断，并由亲和DIE清除pending状态。
- 系统内每个GICD单独维护自己的N-N SPI的active状态，软件配置本GICD的N-N SPI的active状态，不会广播给其他GICD。
- 支持NNSPI的enable配置独立（仅限于Hi1383版本），且多片NNSPI enable配置对软件可见，软件需要分别对多片NNSPI进行配置。
- GIC系统支持16个SGI中断。
- GIC支持TrustZone扩展，输入中断可按照Secure/Non-Secure分组。
- GIC支持中断按优先级上报，中断优先级的值越小说明该中断优先级越高。安全中断支持32个优先级，非安全中断支持16个优先等级，其中PPI/SGI/SPI支持安全和非安全中断类型。
- 支持SPI 1-N中断优先选空闲核上报机制。
- GIC.ARE固定为1，支持GICV3/GICV4特性，不兼容GICV2。
- GIC支持GIC总线地址位宽扩展到48bit。
- GIC支持GIC访问内存（LPI\VLPI的配置表，pending表）的shareable属性可配置。
- GIC支持核下电进入休眠状态，支持通过中断唤醒进入低功耗模式的核。
- GIC可选支持1NSPI唤醒休眠核功能，1NSPI唤醒休眠特性使能时，休眠核比跨SCCL核有更高优先级被选中。
- GIC支持多SICL上的PCIE INTx共享SPI中断号。
- 片内CPU Core可通过总线访问GIC寄存器接口、片外JTAG接口可通过DJTAG BUS访问GIC的寄存器接口。
- 提供模块内部常用信息监控与异常信息调试手段：
  - FIFO空满条件记录。
  - 模块内部各级状态机状态全部寄存器可读。
  - 模块内部当前水线状态可读。
  - 模块之间交互的命令，操作类型，发送方提供上报中断数量记录，接收方提供接收中断数量记录。
- 模块提供总线访问延时统计：
  - 支持统计总线访问最大延迟。
  - 支持接口流量控制，性能统计。
- 选择任意一个中断号进行统计：
  - 该中断被ITS接收的次数。
  - 该中断被GIC接收的次数。

版权所有 © 海思技术有限公司

- 该中断被SET的次数。
- 该中断被CPU Release的次数。
- 该中断被CPU Active的次数。
- 该中断被CPU Deactive的次数。
- 该中断在GICR合并的次数。
- 该中断在GIC CLR的次数。

须知

海思GIC V3.0架构基本遵守GIC Specification要求，但仍然有部分内容不同。与*GIC Specification*相比，除以下特性点不同外，其余特性完全一致，相同的特性点请参考*GIC Specification*相关文档。

当前硬件不兼容GICV2。

亲和路由机制实现不同，GICD支持三层亲和路由关系，详细请参见“**GICD亲和路由**”，GIC Specification定义最多支持4层亲和路由关系。

支持SPI N/N路由模式，详细请参见“**N/N SPI路由**”。

说明

为了读者更好地理解本文的相关设计，建议读者阅读本手册之前，先通读以下文档：

《IHI0048B_gic_architecture_specification 》

《GICv3_prd03_genc_010745_35.0 》

《GICv3_multichip_document 》

须知

反压状态理解为ITS流水线状态机的非IDLE状态。

12.4 遵循协议

架构遵守GIC Specification要求，相关特性点请参考GIC Specification相关文档。

12.5 功能描述

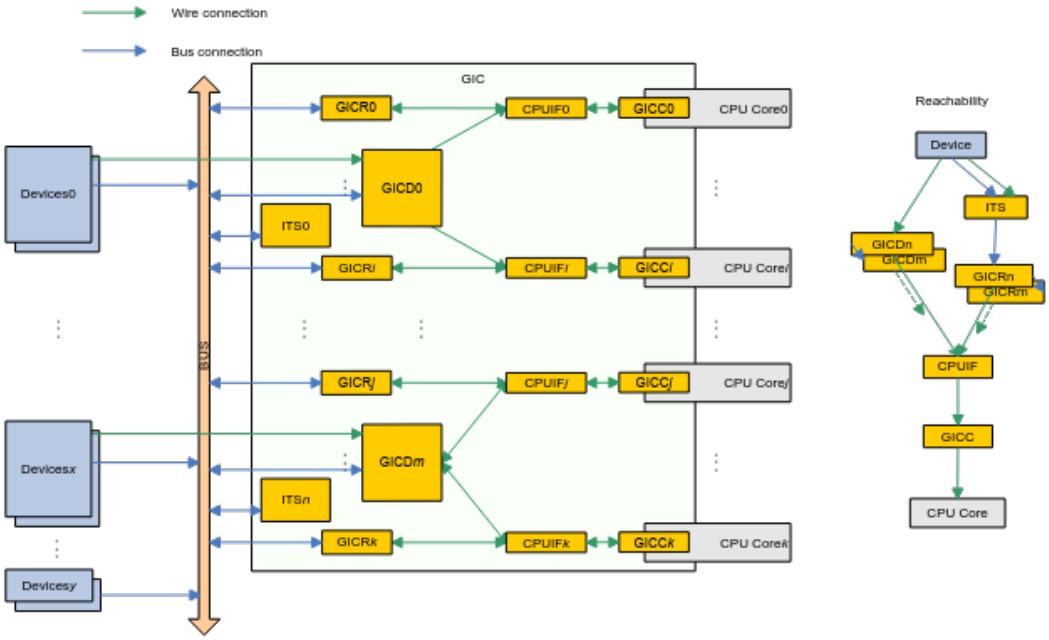
本章将按照层次递进的方式描述GIC的组成结构和各功能子模块架构及相互关系，详细内容如下。

12.5.1 应用框图

GIC组成结构：GIC由中断分发器（GICD）、中断传递服务（ITS）、中断重分发器（GICR）、CORE接口（CPUIF）、GICC组成。此处GICR、CPUIF、GICC与系统中的CPU core数量一致，并一一对应。ITS、GICD为Symmetric Distributed Architecture，一个系统根据需要可以有多个ITS和GICD。GIC各组件在系统中的位置及多组ITS、GICD之间的关系如图12-1所示。

版权所有 © 海思技术有限公司

图 12-1 GIC 在系统中的位置



连接方式：构成GIC所有的组件之间通过如下两类方式相互连接。

WireConnection：把两个组件通过物理硬连线进行连接的方式。具有WireConnection连接关系的组件必须在物理上属于同一个SCL。或者虽然并非通过物理硬连线，但逻辑上要求相互连接的组件必须位于同一个SCL的连接方式在图中也被表示为WireConnection。

BusConnection：多个组件通过总线的方式交换消息。只要接入总线的组件，理论上都可以相互交换消息，没有物理位置的限制。但即使通过总线，在逻辑上，有些组件只能单向发消息，有些组件可双向收发消息，有些组件只能在特定类型的组件之间收发消息，不同GIC组件之间传递消息的通达关系如表12-2所示。

表 12-2 组件之间通过总线传递消息的通达关系

From\To	ITS	GICR	GICD
Device	√	×	√
ITS	×	√	×
GICR	×	√	×
GICD	×	×	√

子模块简介：以下是图中涉及到的GIC各主要子模块及周边模块的功能简介及相互关系。

Device：一些能够产生中断请求的周边设备，它们在必要时产生中断，并以WireConnection或MSI方式将中断请求上报到ITS或GICD。

ITS：类似于ARM GIC V3中ITS，参见“处理器特性”子章节，主要负责接收PCIE或Device通过BusConnection发出的MSI，并依据中断路由配置信息，将中断以MBI的方

版权所有 © 海思技术有限公司

式重定向到合适的GICR进行后继处理。该模块可以通过软件动态修改路由配置，将中断路由到系统中任一GICR。详细描述请参见“**相关模块**”。

GICD：类似于ARM GIC V3中GICD，参见“**处理器特性**”章节，主要负责接收通过WireConnection连接到GICD模块的外设发出的PPI /SPI WireInterrupt和CPU core下发的SGI中断，并根据中断的路由配置信息，把中断分发到与之相连的合适的CPUIF进行后继处理，其中CPU core下发SGI中断需通过CPUIF接收并通过内部总线转发到GICD。同时，该模块也可以由软件配置，利用总线在系统中所有GICD之间相互转发、接收MBI，有关GICD的详细描述请参见“**GICD子模块架构**”。由于GICD与Device和CPUIF之间都是通过WireConnection相连，所以与每一个GICD直接相连的Device、CPUIF及其后对应的GICC、Processor Core都必须位于同一个Chip。

GICR：类似于ARM GIC V3的GICR，主要负责接收任一ITS上报的MBI，并将它转发给与之相连的CPUIF或在其他GICR之间重路由。如图所示，每一个Processor Core对应一个CPUIF和一个GICR。GICR不仅可以转发MBI到对应CPUIF，还可以通过软件配置利用BusConnection在相互之间转发、接收MBI，从而完成中断的清除、转移等操作。GICR接收ITS发送的命令分为物理中断和虚拟中断命令，详细命令接口描述请参见GICV规范文档3 5.4子章节。GICR中断处理流程请参见“**GICR中断上报流程**”。

CPUIF：等同于ARM GIC V3的CPUIF，参见“**处理器特性**”*章节*，主要功能是接收GICD和GICR上报的中断，并依据软件配置信息，调度出优先级最高的中断，将它转换为Processor Core接口格式，上报到对应Processor Core的GICC；另外CPUIF还会接收到CPU core通过GICC下发的SGI中断，CPUIF收到SGI中断则通过内部总线转发到GICD。CPUIF与GICC之间交互命令格式为AXI Stream命令，命令格式参见“**处理器特性**”*章节*。

GICC：接收CPUIF上报的中断，并通过IRQ或FIQ交由相应的Processor Core执行对应的中断处理程序；提供给软件SGI中断接口，并转发CPU core产生的SGI中断到CPUIF。CPUIF与GICC之间交互命令格式为AXI Stream命令，命令格式参见“**处理器特性**”章节。

典型处理流程：下面给出几种系统典型的中断上报流程，虽然不全面，但具有代表性。

1. 通过GICD上报中断流程，详细处理请参见“**GICD中断上报流程**”。
2. 跨GICD间上报中断流程，详细处理请参见“**跨GICD间中断上报流程**”。
3. 通过ITS上报中断流程，详细处理请参见“**ITS中断上报流程**”。
4. GICR间转发上报中断流程，详细处理请参见“**GICR中断上报流程**”。

抽象与实际的映射：上述内容是一个抽象的GIC组成结构及相互关系，一般在一颗设计封装完成的Socket中，Device与各个ITS和GICD的连接逻辑会被固化。

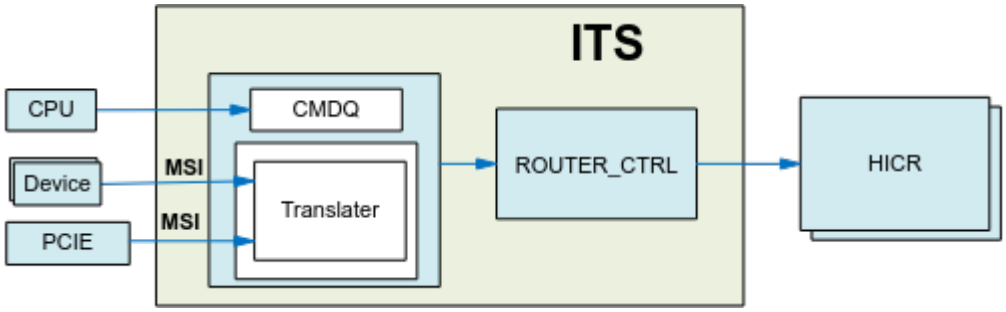
12.5.2 相关模块

12.5.2.1 ITS 子模块架构

ITS功能是提供LPI中断重定向服务，将LPI中断路由到不同的CPU core，其中中断和CPU core的路由关系支持动态配置，如**图12-2**所示。

版权所有 © 海思技术有限公司

图 12-2 ITS 架构框图



ITS由以下模块组成：

CMDQ：完成CPU core下发的ITS命令解析。

Translater：用于捕捉外设写ITS Translater寄存器操作所产生MSI中断，并转发中断给路由表控制模块，该模块对软件完全透明。

ROUTER_CTRL：该模块对软件透明，模块的功能如下：

处理MSI中断，并将中断路由到对应的GICR，每个中断路由的目标GICR地址信息请参考

1. 地址分配章节。
2. 处理CPU core下发的ITS命令，完成路由表改配或者向GICR转发中断维护命令。

几个有关ITS的典型操作流程如下：

1. ITS初始化流程，详细处理请参阅**ITS初始化**。
2. ITS LPI中断路由配置流程，详细处理请参阅**LPI中断路由配置流程**。
3. ITS中断路由改配流程，详细处理请参阅**ITS中断路由改配流程**。
4. ITS中断上报处理流程，详细处理请参阅**ITS中断上报流程**。

12.5.2.1.1 GICD 子模块架构

海思GICD为分布式结构，系统中可以有多个对等的GICD，其中单个GICD中断处理流程与GIC Specification定义的GICD处理流程一致。物理上GICD是分布式实现，对软件只看到一个主GICD，软件只需配置GICD即可。

由于GIC Specification未说明分布式GICD间的中断路由的应用实现，GICD自定义了一套GICD间中断传递的流程，具体流程见**跨GICD间中断上报流程**章节。

另外海思GICD在GIC Specification的亲和路由特性基础上扩展了N/N模型的SPI中断，具体实现方法见**N/N SPI路由**。路由配置时需要了解的可接入GICD的中断信息请参考**中断向量表**章节。

有关GICD的典型操作流程如下：

1. GICD初始化流程，详细处理请参阅**GIC初始化**。
2. GICD中断处理流程，详细处理请参阅**GICD中断上报流程**。
3. GICD间中断转移流程，详细处理请参阅**跨GICD间中断上报流程**。
4. GICD中断路由配置流程，详细过程请参阅**GICR中断上报流程**。

版权所有 © 海思技术有限公司

12.5.2.1.2 GICR 子模块架构

海思GICR和GIC Specification定义的GICR设计完全兼容，GICR的详细工作流程请参考GIC SpecificationGICR设计。

有关GICR的典型操作流程如下：

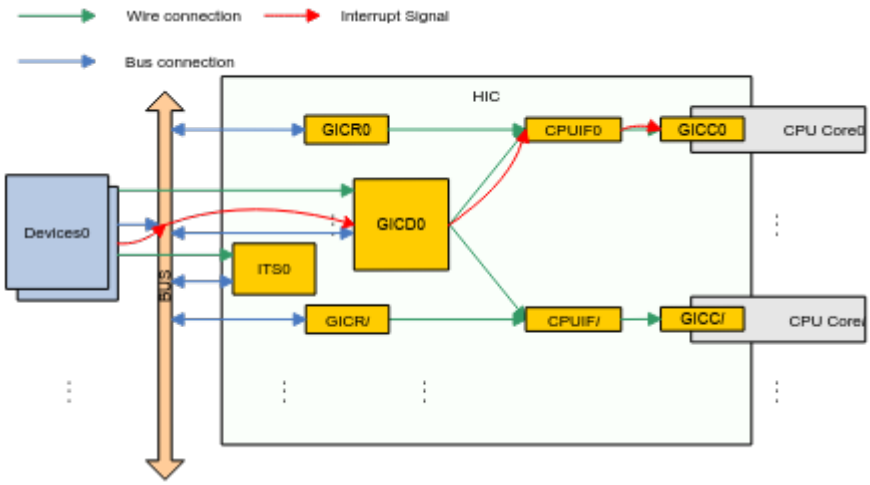
1. GICR初始化流程，详细处理请参阅**GICR初始化**。
2. GICR中断处理流程，详细处理请参阅**GICR中断上报流程**。

12.5.2.2 典型数据处理流程

本章将描述中断在GIC中的典型数据处理流程，并重点阐述GIC中断收集接口和GIC中断路由配置方法。

12.5.2.2.1 GICD 中断上报流程

图 12-3 GICD 中断上报流图



1. 外设发起中断请求，将中断发送到GICD。
   GICD支持接收的中断类型：PPI/SGI/SPI，GICD中断输入接口如**表12-3**所示。

表 12-3 GICD 中断说明

GICD接收中断类型	中断产生方法	说明
SGI	软件写ICC_SGIx寄存器方式 其他GICD写GICD_CROSSSCL_SETR方式	将软件和其他GICD均抽象成一种外设，同普通外设一样上报中断。

版权所有 © 海思技术有限公司

GICD接收中断类型	中断产生方法	说明
SPI	外设写GICD_SETSPI_NSR/GICD_SETSPI_SR方式 或者 其他GICD写GICD_CROSSSCL_SETR方式 或者 外设中断线连接到GICD，中断通过连线上报到GICD方式	将其他GICD抽象成一种外设，同普通外设一样上报MSI中断。
PPI	CPU core中断线连接到GICD，中断通过连线上报到GICD方式	-

GICD接收的中断类型、与外设连接关系、中断触发方式参见**中断向量表**。

2. GICD根据路由配置，调度出优先级最高的中断，发送到CPUIF。

表 12-4 GICD 中断路由配置

GICD接收中断类型	中断路由算法获取	说明
SGI	软件写ICC_SGIx寄存器时获取中断路由配置。	SGI支持Targeted list路由算法，详见**GICD亲和路由**。
SPI	1-N模型SPI从软件配置的GICD_IROUTER寄存器获取路由配置的方式。 N-N模型SPI从软件配置的GICD_ITARGETSR_SPI寄存器指定。	SPI支持1/N、N/N方式路由算法，详见**GICD亲和路由**。
PPI	只由特定CPU core进行处理，不能路由给其他CPU core。	不可配置路由目标。

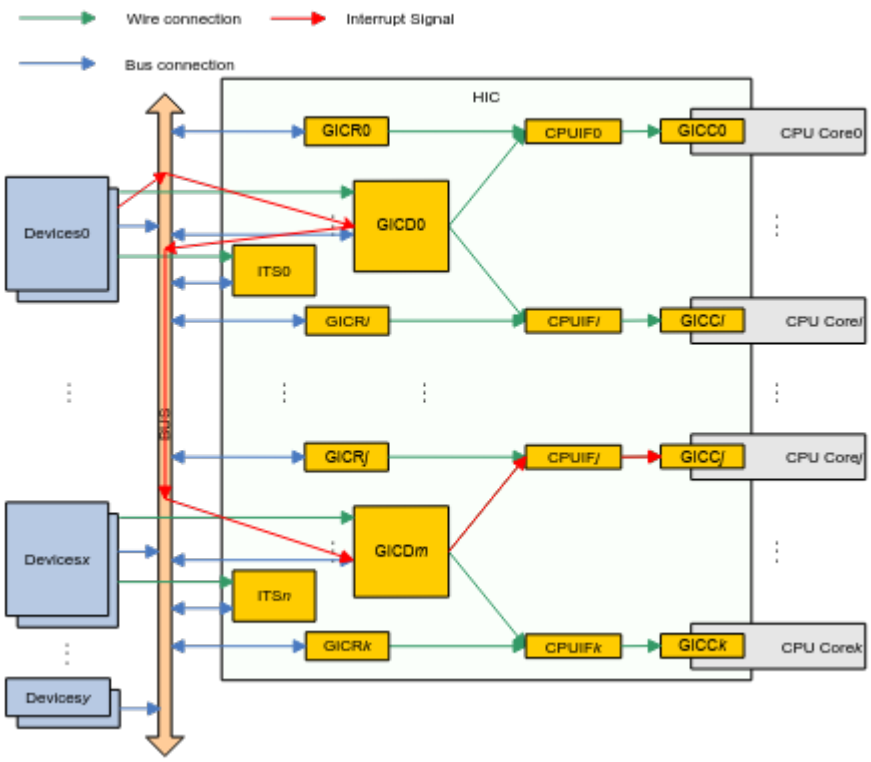
GICD的路由模式参见**GICD亲和路由**，GICD的配置方法参见**GICD中断路由配置流程**。

3. CPUIF从输入中断中调度出优先级最高的中断，通过AXI4 Stream协议给GICC发送SET中断上报命令，把中断发送到GICC。

版权所有 © 海思技术有限公司

12.5.2.2.2 跨 GICD 间中断上报流程

图 12-4 跨 GICD 间中断上报流图



1. 外设发起中断请求，将中断直接发送到GICDn。
   支持GICD间转移中断类型：SGI/SPI

   表 12-5 支持 GICD 间转移中断类型说明

中断类型	支持中断转移的中断产生方法	说明
SGI	软件写ICC_SGIx寄存器命令产生SGI中断方式。	不支持已经从GICD转发的中断，再转发到其他GICD。
SPI	外设SPI中断源直接连线到GICD方式 外设写GICD_SETSPI_NSR/GICD_SETSPI_SR方式	不支持已经从GICD转发的中断，再转发到其他GICD。

   GICD接收的中断类型、与外设连接关系、中断触发方式见**中断向量表**章节。
   GICD的路由模式参见**GICD亲和路由**，GICD的配置方法参见**GICD中断跨片路由配置流程**。
2. GICDn根据路由配置，调度出优先级最高的中断，发送到另一个GICDm。

版权所有 © 海思技术有限公司

表 12-6 GICDn 中断路由配置

中断类型	中断路由配置获取途径	说明
SGI	在软件写ICC_SGIx寄存器命令产生SGI中断命令时获取中断路由的方式，见GICV3 *5.7.29章节*;	亲和路由使能ARE**GICD亲和路由**为真时，1/N、N/N路由模式的SGI中断支持在GICD间转移，详见**GICD亲和路由**。
SPI	从软件配置的GICD_IROUTER寄存器获取路由配置的方式。	亲和路由使能ARE**GICD亲和路由**为真时，1/N路由模式的SPI中断支持在GICD间转移，详见**GICD亲和路由**。

GICD的路由模式参见**GICD亲和路由**，GICD跨越GICD转发中断的配置方法参见**GICD中断跨片路由配置流程**。

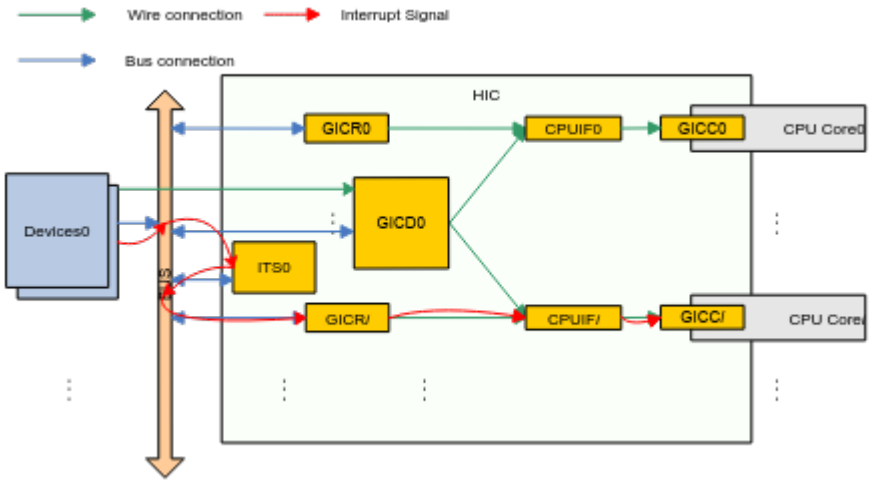
3. GICDm处理方式同**GICD中断上报流程**小节描述的GICD中断上报处理流程，具体流程略。

说明

海思GIC多片实现符合ARM GIC Multichip Implementation规范，具体请参考“GICv3_multichip_document.pdf”。

12.5.2.2.3 ITS 中断上报流程

图 12-5 ITS 中断上报流图



1. 外设发起中断请求，将中断可以直接发送到ITS，如**表12-7**所示。

版权所有 © 海思技术有限公司

表 12-7 ITS 中断产生说明

中断类型	中断产生方法	说明
LPI	外设写ITS_Translater寄存器产生方式。	-

ITS支持中断类型：LPI/VLPI，被路由到物理CPU core的ITS中断为LPI；被路由到虚拟CPU core的ITS中断为VLPI。

其中连线方式上报的LPI中断需要软件配置，具体配置方法见**LPI中断路由配置流程**。

2. ITS根据路由配置找到对应的GICR基地址，不同GICR的基地址参见"**MSI**”，以MSI形式发送给GICR（写GICR_SETLPIR/ GICR_VSETLPIR寄存器）。

   每次写操作对应一次MSI消息，ITS给GICR的中断只支持沿中断类型，如果该中断未被及时处理，而又来新的MSI消息，中断会在GICR进行合并。

表 12-8 ITS 中断路由配置

中断类型	中断路由配置获取途径	说明
LPI	软件配置ITS三级中断路由表，配置命令见GICV3 *4.9.8章节*。	LPI路由表配置方法请参见“**LPI中断路由配置流程**”。

3. GICR从接收到的中断中调度出最高优先级的中断，发送给相应的CPUIF。

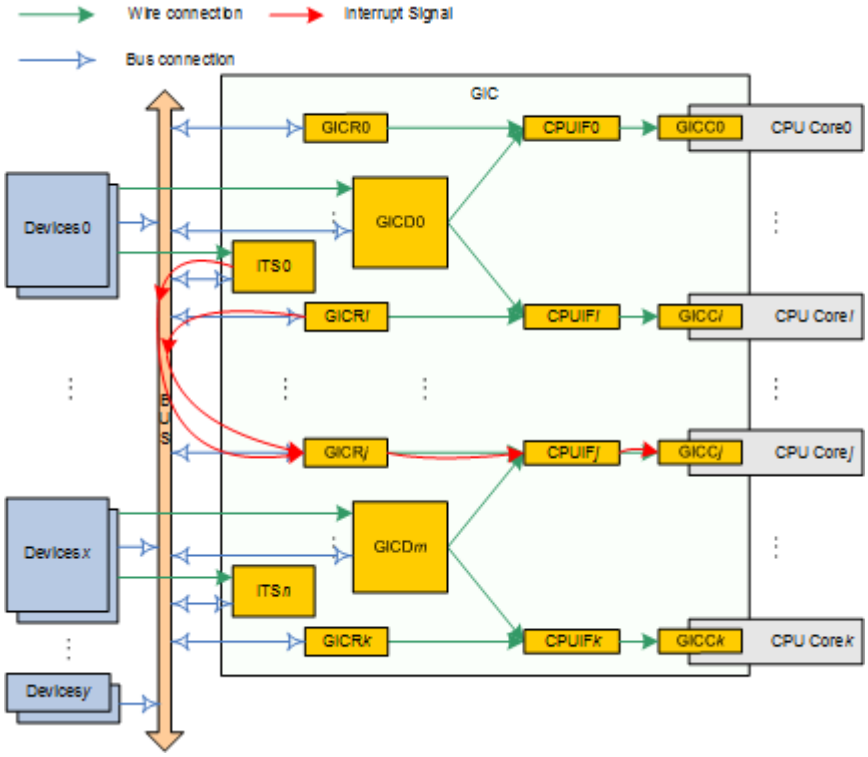
   GICR具体配置方法请参见“**GICR\GICD软件复位流程**”。

4. CPUIF从输入中断中调度出优先级最高的中断，再通过AXI4 Stream协议给GICC发送SET中断上报命令，把中断发送到GICC。

版权所有 © 海思技术有限公司

12.5.2.2.4 GICR 中断上报流程

图 12-6 GICR 中断上报流图



1. GICR作为子分发器，可以接收下列模块发出的中断请求。

表 12-9 GICR 中断源说明

GICR中断源	中断传递方法	说明
ITS	ITS写GICR_SETLPIR/GICR_SETVLPIR寄存器，产生MSI中断方式。	-
GICR	GICR执行MOV/MOVALL命令，写另一个GICR的GICR_SETLPIR/GICR_SETVLPIR寄存器，产生MSI中断方式。	MOV/MOVALL命令的应用场景描述“**GICR中断转移流程**”。
软件	GICR_SETLPIR/GICR_SETVLPIR寄存器对软件可见，软件可以直接操作该寄存器产生LPI中断。	-

2. GICR根据配置从接收到的中断中调度出最高优先级的中断，发送给CPUIF。

版权所有 © 海思技术有限公司

表 12-10 GICR 中断配置

配置类型	配置内容	说明
LPI Config table	GICR LPI/VLPI中断优先级配置由软件分配，单个配置大小为8bit，格式如下： { Priority = 0x3F， Group = 0b1 ， Enable = 0b0}；	GICR配置方法请参见“**GICR\GICD软件复位流程**”。

3. CPUIF、GICC、CPU core的详细交互流程见GICV3 7.6.*1章节*。

12.5.2.3 基础特性描述

本章将详细阐述GIC与GIC规范在具体子模块上的实现差异和特有特性，以及GIC规范定义的亲和路由在GIC的具体实现。

12.5.2.3.1 中断向量表

中断类型主要包括SGI/PPI/SPI/LPI四种，由于不同Socket拥有不同外设和GIC硬件，因此不同的Socket的中断源、中断类型以及与GIC硬件连接各不相同，它们被分别描述如下各子节。此处描述针对全部可能的Socket和SCL划分情况，实际的芯片依据具体的封装，这些SCL的组合情况可能各有不同。

Hi309a芯片详细的外设与GIC之间的连接关系及中断向量分配请参考《 Hi309a 中断向量表 》。

12.5.2.3.2 LPI 中断设备号

表 12-11 GIC Device ID

bit18~21	bit16~17	bit0~15
SCL ID	ECAM NUM，具体参考PCCU文档	SMMU Stream ID，具体参考SMMU文档。

12.5.2.3.3 地址分配

在Hi309a Socket的GIC架构中，每32个GICR和1个GICD共用一片配置空间，每个ITS有一片独立的配置空间。GICR和GICD共用配置空间地址分配如**表12-12**所示。

表 12-12 GICR 和 GICD 地址分配

模块名称	子模块	配置基地址（23:16)	偏移地址（15:0)
GICD		0x00	0x0000~0xFFFF
GICR0	GICr0_lpi	0x10	0x0000~0xFFFF
	GICr0_sgi	0x11	0x0000~0xFFFF

版权所有 © 海思技术有限公司

模块名称	子模块	配置基地址（23:16)	偏移地址（15:0)
	GICr0_vlpi	0x12	0x0000~0xFFFF
	ReserveD	0x13	0x0000~0xFFFF
GICR1~GICR30	…….	…….	…….
	…….	…….	…….
GICR31	GICr31_lpi	0x8C	0x0000~0xFFFF
	GICr31_sgi	0x8D	0x0000~0xFFFF
	GIC31_vlpi	0x8E	0x0000~0xFFFF
	Reserved	0x8F	0x0000~0xFFFF
Reserved		0x90~0xFF	0x0000~0xFFFF

有关GICD子模块内部的配置空间说明将在下一小节展开说明。

12.5.2.3.4 MSI

随着CPU core数量的增加，中断的数量也大量增加，为了解决专用上报中断线过多的问题，GICv3增加支持Message形式上报中断的机制：外设通过总线向特定地址空间写特定数据值，来产生特定中断。采用该机制产生的中断被称为MSI中断。

GIC支持MSI中断的接口如表12-13，若MSI接口对外设可见，则表示外设可以写该接口产生MSI中断。

表 12-13 GIC 支持的 MSI 中断接口

MSI中断接口	数据格式	说明
GICD_SETSPI_NSR/ GICD_SETSPI_SR	同GIC Specification定义	该接口对外设、软件可见。 软件写该寄存器产生SPI MSI中断。
ICC_SGIRx	同GIC Specification定义	该接口对软件设可见。 软件写该寄存器产生SGI MSI中断到GICD。
ITS_Translater	ITS_Translater bit[15:0].EvevtID	该接口对外设、软件可见。 软件写该寄存器产生LPI MSI中断到GICR，该方式仅用于测试。
GICR_SETLPIR/ GICR_SETVLPIR	同GIC Specification定义	该接口对ITS、软件、GICR可见。 软件写该寄存器产生LPI MSI中断到GICR，该方式仅用于测试。

版权所有 © 海思技术有限公司

说明

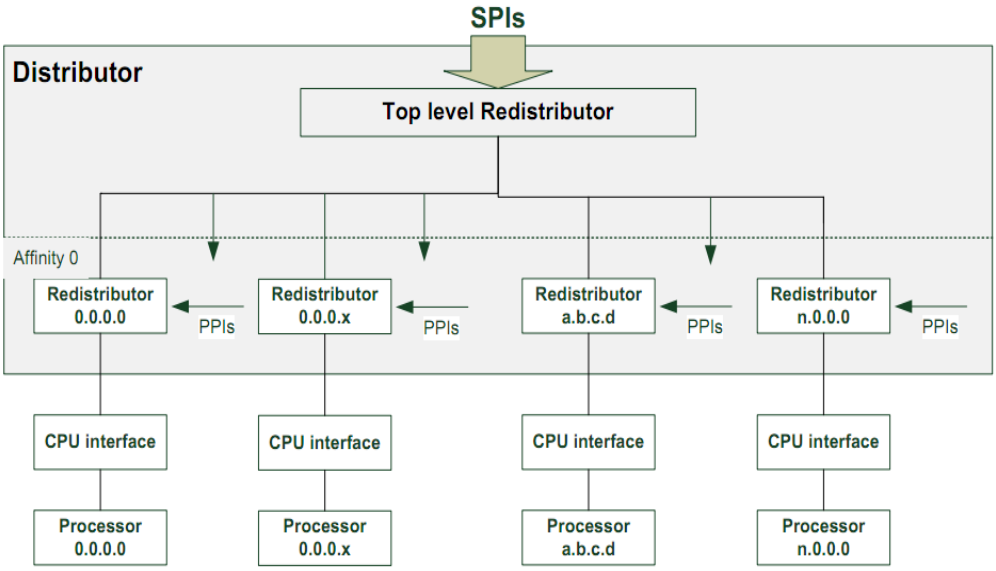
GICD_CROSSSCL_SETR寄存器为GICD为支持跨GICD中断转移特性扩展定义。

MSI中断接口均对软件可见，必要时软件直接写该接口产生中断。

12.5.2.3.5 GICD 亲和路由

GIC Specification亲和性路由分级可达4个层次，如图12-7所示。

图 12-7 亲和性层次图



GIC亲和路由设计遵循GIC Specification，但路由关系只有三个层次，GIC亲和路由配置定义如下：

- AL3，该层次未定义，硬件不使用该字段。
- AL2，该值 bit[5:2]等于SCLID，SCLID表示在系统中SCL的编号，指示中断被路由到哪个SCL；该值 bit[1:0]等于ClusterID，ClusterID表示某SCL内的CPU Cluster编号，指示中断被路由到哪个Cluster。
- AL1，该值bit[2:0]等于CPUID，CPUID表示在某CPU Cluster内CPU core编号，指示中断被路由到哪个CPU core。
- AL0，该层次在SGI和SPI路由配置中定义不同：

  在SPI亲和路由配置中，AL0表示Thread Num；目前参照A55格式，AL0 tied to Zero。

  在SGI亲和路由配置中，AL0等于TargetList；该值并不构成路由信息，该值[0]bit指示SGI路由是否有效。

表 12-14 GICD 中断亲和路由算法

路由模式	IRM=0	IRM=1
1/N SPI路由	中断路由到亲和路由配置“a.b.c.d”指定的CPU core，请参见“1/N SPI亲和路由”。	中断发送到SCL内所有CPU core，但只有其中一个CPU core处理该中断。

版权所有 © 海思技术有限公司

路由模式	IRM=0	IRM=1
N/N SPI路由	中断路由到合并的路由配置指定的CPU core	中断路由到SCL内的所有CPU core。
S/N SGI路由	中断路由到亲和路由配置“a.b.c.{ TargetList }“指定的CPU core，请参见“1/N SPI亲和路由”。	中断路由到系统中除自己以外的所有CPU core。

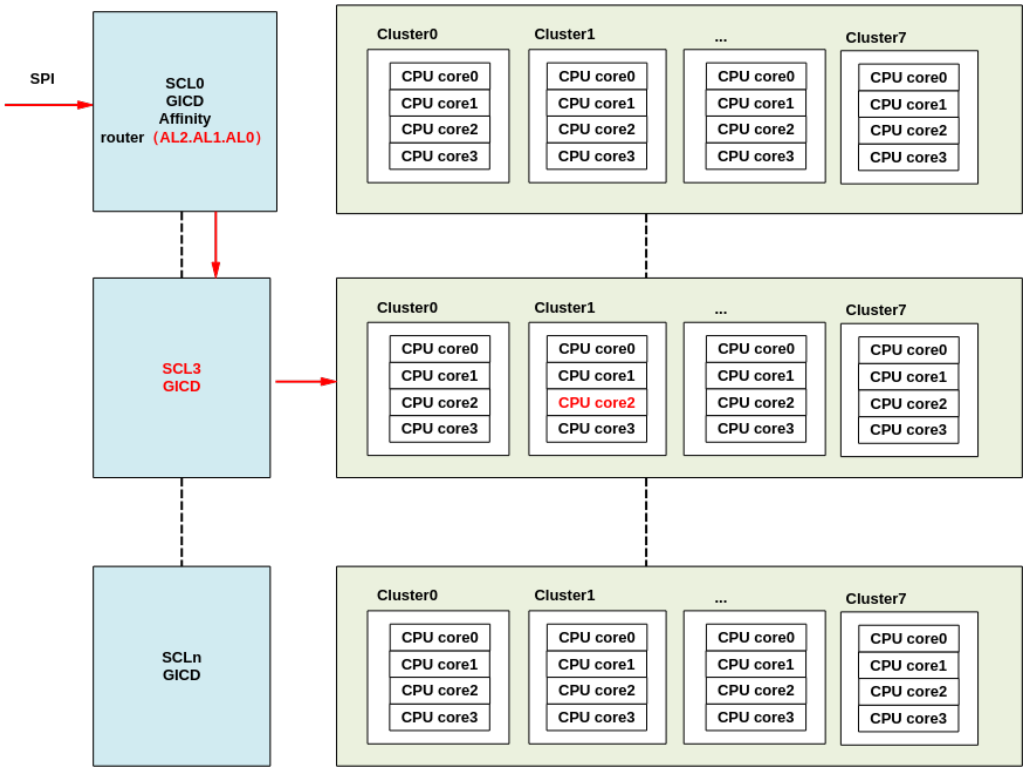
GICD亲和路由相关配置寄存器可查询寄存器子章节提供的芯片寄存器手册得到。

12.5.2.3.6 1/N SPI 亲和路由

12.5.2.3.6.1 IRM 为 0 模式

该模式SPI中断为1/1路由模式，支持跨SCL中断传递。以连接在SCL0 GICD上的SPI为例，假设该SPI中断亲和路由配置为“0.3.1.2”，SPI中断亲和性路由上报流程图如图12-8所示。

图 12-8 1/N SPI 中断亲和性路由示例



如图12-8红色箭头路由路径，根据亲和路由配置“0.3.1.2”SPIs会被1/1路由给唯一目标CPU core。亲和路由配置的AL2层可以指定系统中的任意SCL为目标SCL，当亲和路由配置的目标SCL不是SPI硬件连接所在的SCL时，将产生跨SCL SPI中断传递。

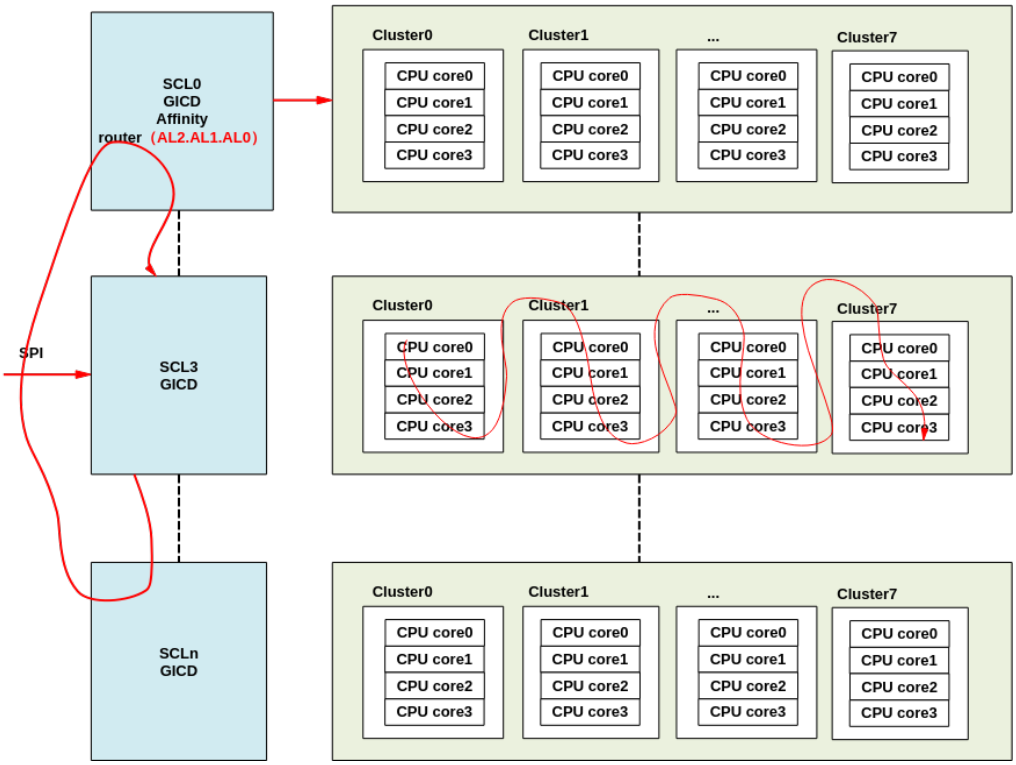
版权所有 © 海思技术有限公司

12.5.2.3.6.2 IRM 为 1 模式

该模式SPI中断为1/N路由模式，支持跨SCL中断传递。该模式中断将会被发送给系统内空闲CPU core，但只有一个核会处理该中断，1/N上报的流程如下：

1. 中断上报到GICD后，会上报到SCCL内的随机CPU。
2. 如果本SCCL的所有CPU均繁忙，则按SCLID递增顺序发到其他SCCL的GICD上报，SCLID到最大值时会发生卷绕。
3. 在其他的SCCL GICD中，中断也被优先上报到该SCCL内的CPUID较小的空闲CPU。
4. 当所有SCCL均被遍历后，则随机向某一个核发起CPU请求，并开启超时计数器，如果CPU在响应超时，则会随机向下一个核发起CPU请求，直到中断被CPU处理。

图 12-9 1/N SPI 中断亲和性路由示例



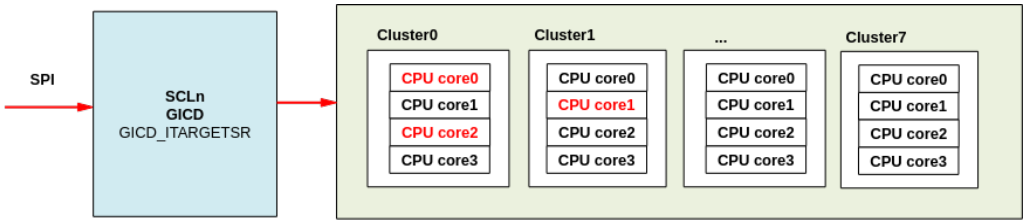
12.5.2.3.7 N/N SPI 路由

N/N类型中断，表示中断将被发送到SCL内多个目标CPU core，每个CPU core独立处理该中断，当一个CPU core响应该中断时，只会清除该CPU core的该SPI中断的状态，不影响其他CPU core的该SPI中断的状态。N/N模式SPI中断不支持电平触发类型中断，电平方式输入的SPIs当作沿触发中断处理。

该模式下，SCL内所有N/N中断可以单独配置到不同的目标CPU core，路由目标CPU core的是通过GICD_ITARGETSR_SPIn进行配置的，如图12-10所示。如果系统存在多个SCL，并且SCL的GICD是使能的，则还需要将中断在该SCL内进行N/N广播，广播的路由配置是目标SCL的GICD_ITARGETSR_SPIn进行配置的。

版权所有 © 海思技术有限公司

图 12-10 N/N SPI 路由配置示例



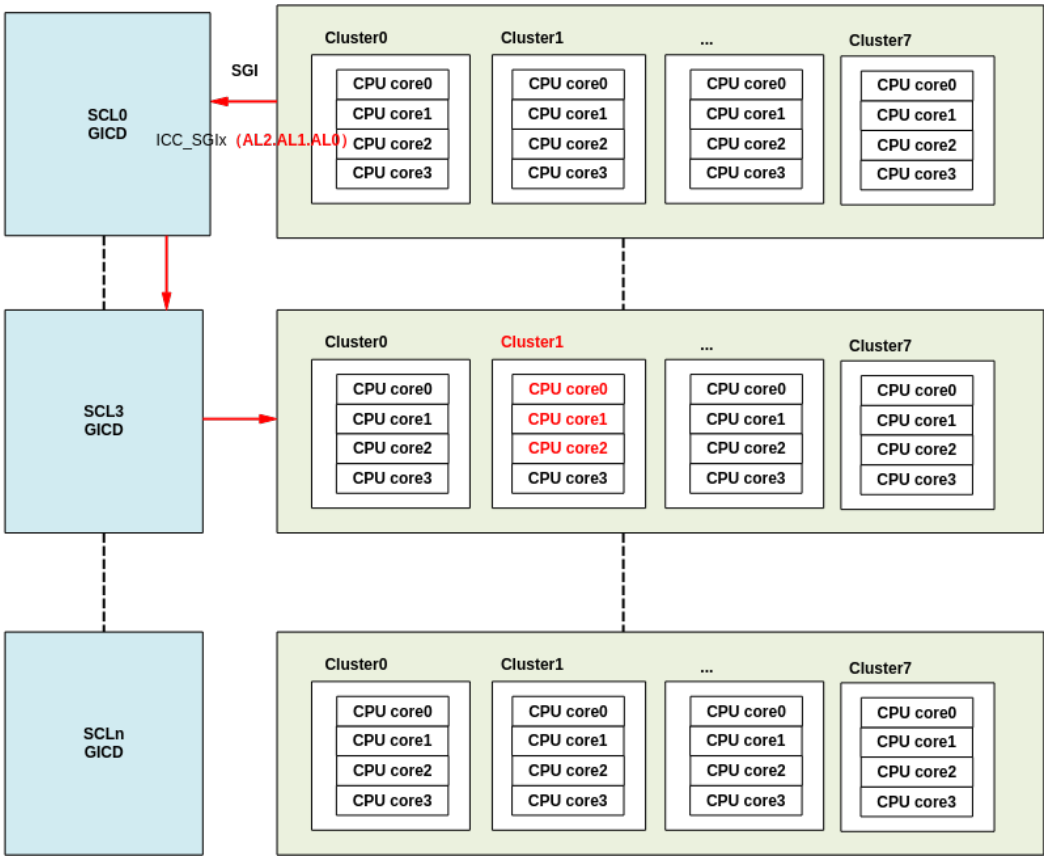
12.5.2.3.8 SGI 亲和路由

12.5.2.3.8.1 IRM 为 0 模式

SGIs亲和路由配置的格式为“a.b.c.{TargetList}”，TargetList字段是OneHot类型，TargetList的每一位表示一个CPU core，中断会路由给“a.b.c”层次下“TargetList”中的CPU core。该中断上报模型为S/N，即中断可根据亲和路由上报到指定的多个CPU core。

以连接在SCL0 GICD上的SGI中断为例，假设该SGI中断亲和路由配置为“0.3.1.7”，SGI中断亲和性路由上报流程图如图12-11所示。

图 12-11 SGI 中断亲和性路由示例



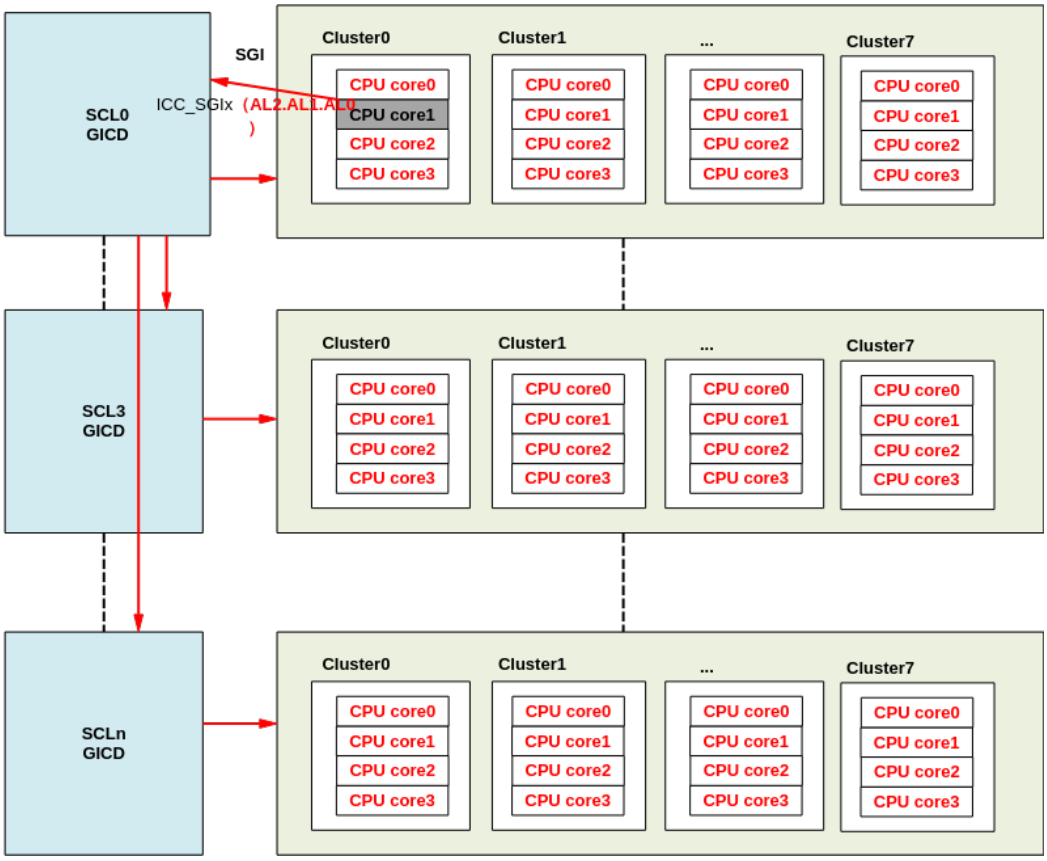
如上图12-11红色路由路径，SGIs会被路由给最多4个CPU core。

版权所有 © 海思技术有限公司

12.5.2.3.8.2 IRM 为 1 模式

该模式中断会广播给除本CPU core外的所有SCL的CPU core，如图12-12所示。

图 12-12 N/N SGI 中断亲和性路由示例



12.5.2.3.9 中断触发方式

中断触发方式包含电平敏感方式和边沿触发方式，GIC对各种中断类型的触发方式进行了具体定义：

- 对于SGIs/LPIs是固定为边沿触发方式。
- 对于PPIs和1/N模型SPIs是可配置的，复位后为电平敏感方式。
- 对于N/N模型SPIs的中断则仅支持边沿触发方式。

12.5.2.3.10 LPI Config Table

1. LPI配置表地址通过GICRn_（V）PROPBASER.PA进行配置。
   配置格式为：{Priority bit[5:0]，Group，Enable}，每个中断配置占8bit。

须知

Group必须固定配置为1，否则可能出现异常。

版权所有 © 海思技术有限公司

2. 每个GICR有独立的LPI CFG配置寄存器，软件保证SCL内的GICR共享LPI配置。
3. 本硬件GICRn_TYPERn.CommonLPIAff固定为0x2，表示软件需为不同SCCL的GICR分配不同LPI CFG，避免为LPI配置分配跨SCL内存。

12.5.2.3.11 中断优先级

不同的安全状态下，访问优先级配置会得到不同的值，具体如表12-15所示。

表 12-15 安全软件视角中断优先级配置

中断类型	配置寄存器	安全中断优先级	非安全中断优先级
SGI/PPI	GICR_IPRIORITYRn	{IPRIORITYRn[7:3]，0b0}	{1b1，IPRIORITYRn[6:4]，0b0}
SPI	GICD_IPRIORITYRn	{IPRIORITYRn[7:3]，0b0}	{1b1，IPRIORITYRn[6:4]，0b0}
LPI	LPI Config Table	无安全中断	{1b1，IPRIORITYRn[6:2]，0b0}

说明

以上配置读写均由安全软件发起。

表 12-16 非安全软件视角中断优先级配置

中断类型	配置寄存器	安全中断优先级	非安全中断优先级
SGI/PPI	GICR_IPRIORITYRn	0b0	{IPRIORITYRn[7:4]，0x0}
SPI	GICD_IPRIORITYRn	0b0}	{IPRIORITYRn[7:4]，0x0}
LPI	LPI Config Table	无安全中断	{IPRIORITYRn[7:2]，0b0}

说明

以上配置读写均由非安全软件发起。

GIC中断调度原则：

1. 中断调度遵循时间优先，先到的中断先报。
2. 相同时刻的中断遵循优先级调度。
3. 优先级相同时，LPI采用RR调度，SPI遵循ID越小优先上报，SPI/LPI之间是SPI优先上报。
4. 中断优先级调度是基于安全视角的中断配置进行调度。
5. 高于GICC下发的PMR的优先级门限的中断才能被调度。

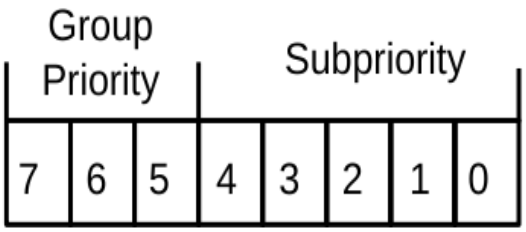
12.5.2.3.12 中断嵌套

GIC支持中断嵌套，中断嵌套的深度由Group Priority的级数决定，最多支持128级中断嵌套。8bit中断优先级域Priority{7,0}分为两部分，Group priority和Subpriority。可以

版权所有 © 海思技术有限公司

利用GICC_BPR.Binary point配置 Group priority和Subpriority组的位域，如图12-13所示，将Binary point设置为4，则Group priority为3bit位宽，表示支持8级中断嵌套，Sub priority设置为5bit位宽，如果中断的组优先级相同，中断是不可以嵌套的。

图 12-13 中断嵌套示意图



Group Priority的位数由寄存器 GICC_BPR、GICC_ABPR（非安全）、GICV_BPR（虚拟）可配。

图 12-14 Priority grouping by binary point

Binary point value	Interrupt priority field [7:0]		
	Group priority field	Subpriority field	Field with binary point
0	[7:1]	[0]	ggggggg.s
1	[7:2]	[1:0]	gggggg.ss
2	[7:3]	[2:0]	ggggg.sss
3	[7:4]	[3:0]	gggg.ssss
4	[7:5]	[4:0]	ggg.sssss
5	[7:6]	[5:0]	gg.ssssss
6	[7]	[6:0]	g.sssssss
7	No preemption	[7:0]	.ssssssss

12.5.2.3.13 中断唤醒核

CPU时钟允许被复位或关断，CPU时钟关断是一种核深度休眠的状态。GIC允许通过中断触发系统控制器打开休眠CPU时钟，CPU核时钟打开后，CPU将自动进行初始化，进入正常工作状态，该步骤称为中断唤醒核。中断唤醒的步骤如下：

步骤1 GICR_WAKER.ChildrenAsleep表示关联核的休眠状态，初始状态为1。

步骤2 用于唤醒功能的中断始终被配置为使能。

步骤3 当唤醒中断pending被置位，被GIC调度给某休眠核时，GIC会向系统控制器发起该核的WakeRequest。

步骤4 系统控制器收到WakeRequest后，控制CPU时钟解复位。

步骤5 CPU解复位后执行代码开始初始化复位向量，CPU进入工作态。

版权所有 © 海思技术有限公司

步骤6 软件写GICR_WAKER. ProcessorSleep为0，唤醒GICR。

步骤7 GICR下发GICD_CTLR.DS配置到GICC，唤醒GICC。

步骤8 GICR收到GICC的配置应答后表示GICC被唤醒，GICR设置GICR_WAKER.ChildrenAsleep为0。

步骤9 软件在GICR_WAKER.ChildrenAsleep为1之前访问GIC寄存器都是非预期的。

步骤10 软件轮询GICR_WAKER.ChildrenAsleep直到该返回值为0，软件启动配置GICC中断分组。

----结束

须知

如果软件写GICR_WAKER.ProcessorSleep为1后，就必须等GICR_WAKER.ChildrenAsleep为1，才能再写GICR_WAKER.ProcessorSleep为为0.否则不可预期。注意：上面的步骤必须被遵守，如果在GICR_ProcessorSleep置1后，下电步骤被中止。那么正常的上电或者或复位可能不会发生。

12.5.2.3.14 核休眠

CPU时钟允许被复位或关断，被关断时钟的核可以看做进入深度休眠状态。进行核休眠控制时，必须保证该核对于的GICC处于静默状态，GIC规范定义一套软硬件协同机制，保证核可以正常进入休眠的状态。

初始状态GICR_WAKER.ChildrenAsleep为0，GICC处于工作态，通过如下步骤完成核休眠操作：

步骤1 软件关闭GICC中断分组使能。

步骤2 软件设置GICR_WAKER.ProcessorSleep为1，通知GICR进入休眠。

步骤3 GICR发送Quiesce消息通知GICC进入休眠状态。

步骤4 GICC释放未处理中断，正在处理的中断需要正常完成。

步骤5 软件将释放的中断转移到其他核处理。

步骤6 GICC返回Quiesce完成应答给GICR。

步骤7 GICR设置GICR_WAKER.ChildrenAsleep为1，表示GICC进入休眠。

步骤8 软件轮询到GICR_WAKER.ChildrenAsleep为1之后，配置系统控制逻辑，完成核时钟关断或者复位。

----结束

须知

软件在GICR_WAKER.ProcessorSleep置1后，还需等待；GICR_WAKER.ChildrenAsleep状态拉高，并且必须确保GICD不会发出中断到静默的CPU。否则不可预期。

版权所有 © 海思技术有限公司

12.6 应用配置

12.6.1 BIOS 配置说明

GIC/ITS解复位后，软件按照以下章节顺序对GIC/ITS各模块进行BIOS配置初始化：

如果芯片是多SCL合封，则需要在BIOS阶段初始化跨SCL配置，如果是单SCL封装，则不需要关注本章节，BIOS初始化步骤如下：

步骤1 初始化跨SCL中断目标地址：

根据系统规划地址，将各个GICD的系统基地址配置到GICD_CROSSDIE_BASEADDR寄存器。

步骤2 初始化SCL接收使能：

配置GICD_DIEEN_CROSS寄存器，设置可以接收跨SCL中断的SCL。

步骤3 初始化SCL N2NSGI中断接收使能：

配置GICD_DIEEN_CROSS_SGI_N2N寄存器，设置可以接收跨SCL NNSGI的SCL。

步骤4 配置GICD_DIEEN_CROSS_SPI_12N寄存器，设置可以接收跨SCL 1NSPI的SCL初始化SCL 12NSPI中断接收使能。

步骤5 初始化SCL N2NSPI中断接收使能：

配置GICD_DIEEN_CROSS_SPI_N2N寄存器，设置可以接收跨SCL NNSPI的SCL。

----结束

12.7 初始化

本章节主要描述GIC的各种典型应用场景的软件配置流程。

12.7.1 GIC 初始化

BIOS初始化后，软件按照以下章节顺序对GIC各模块进行初始化。

12.7.1.1 ITS 初始化

GIC初始化后，如果需要使用ITS功能，那么必须对各个ITS进行初始化，各ITS在系统中的基地址分配请参见“地址分配”。

须知

ITS使能必须在GIC初始化完成之后。

步骤1 路由表初始化：

配置ITS_BASERn寄存器完成ITS路由表初始化，ITS路由表所需内存分配算法如下：

版权所有 © 海思技术有限公司

1. Device table（简称DT表）内存配置

   DT表项内存页分配为4kByte的对齐，每页支持设备号个数为4kByte/8Byte=512的倍数。

   ITS支持的设备ID号位宽为did_width， DT size配置为ffff>>（16+8-did_width）；分配的内存大小为（DT size +1）* 4kByte;

   例1：若ITS支持的设备ID号范围0~511，则did_width=8，DT size = ffff>>（16+8-did_width） =0，分配内存大小（（DT size +1）*4kByte = 4kByte；

   例2：若ITS支持的设备ID号范围0~1023，则did_width=9，DT size = ffff>>（16+8-did_width） =1，分配内存大小（（DT size +1）*4kByte = 8kByte；

2. collection table（简称CT表）内存分配

   CT表项内存页分配为4kByte的对齐，每页支持设备号个数为4kByte/8Byte=512的倍数。

   ITS支持的Collection ID号位宽有效为cid_width，则CT size配置为ffff>>（16+8-id_width）；（目前cid_width位宽为7（8bit），支持256个中断分组，而每内存页支持cid_width位宽为8（9bit），因此CT size为0即可）。

3. virtual collection table（简称VCT表）内存分配

   VCT表项内存页分配为4kByte的对齐，每页支持设备号个数为4kByte/8Byte=512的倍数。

   ITS支持的VCPU ID位宽有效为vcpu_width，则VCT size配置为ffff>>（16+8-id_width）；（目前vcpu_width位宽为7（8bit），支持256个虚拟*CPU core*，每内存页支持vcpu_width位宽为8（9bit），因此VCT size为0即可）。

软件根据系统需要分配好相应的Table内存，内存须初始化为0（否则将产生随机配置异常）；再将分配的内存指针配置到对应ITS_BASERn.pa寄存器中，配置对应内存访问属性，根据下面公式完成表项ITS_BASERn.size配置，将对应表的ITS_BASERn.valid置1，完成路由表初始化。

须知

如果软件未使能LPI虚拟化特性，第三步初始化可以省略。

步骤2 CMDQ初始化：

配置ITS_CBASER、ITS_CWRITER、ITS_CREADR寄存器，完成命令队列内存分配，外部命令队列写指针初始化。外部命令队列内存页分配为4kByte的对齐，每页支持命令个数为4KB/32Byte=128的倍数。可分配最大内存大小为4KB*256=2M，最多支持命令数32K；外部命令队列不支持中途重新初始化。卷绕指针为Wrap =（cmdq_size+1）<<7 - 1。

步骤3 ITS_CTRL.enable置为1使能ITS，如需支持虚拟中断功能，还需使能将ITS_CTRL.vlpienable置为1。如果系统中有多个ITS，则由系统为该ITS分配唯一的ITS编号，软件将编号配置到ITS_CTRL.ITS_number，ITS硬件会根据ITS_number编号使用哪个GICR_VMOVLPIRn寄存器，其中n等于ITS_number号。

----结束

版权所有 © 海思技术有限公司

12.7.1.2 GICR 初始化

配置GICR_（V）PROPBASER寄存器完成（V）LPI Config table初始化，完成（V）LPI中断优先级、中断分组、中断禁能设置；配置GICR_（V）PENDBASER寄存器完成（V）LPI PT/CPT表初始化，设置（V）LPI中断待发送队列初始状态。

初始化步骤如下：

步骤1 软件初始化时给每个Re-Distributor分配8kByte大小的PendingTable内存空间和每个VM分配一个64KB的大小的config table内存空间。

步骤2 配置初始化中断Pending状态的pending table，简称PT，PT分为粗粒度中断Corase PT（CPT）和PT两部分：

对于PT：1kByte~64kByte中断Pending状态初始化为Pending = 0b0，实际用到的是8kByte~64kByte空间；

对于CPT：0~1kByte的粗粒度中断映射pending状态初始化为：

{unused = 0b0，Priority = 0x3F，Pending = 0b0}

其中Pending = 0b0，表示非Pending状态。

步骤3 配置初始化Config table存放的64k中断配置每Byte为：

{ Priority = 0x3F， Group = 0b1 ，Enable = 0b0}

步骤4 配置GICR_CTRL.enable_lpis为1，打开（V）LPI中断使能。

----结束

12.7.2 Distributor 初始化

1. 配置Distributor接口。
   a. 配置寄存器GICD_CTRL使能亲和路由位、全局安全和全局非安全分组。
   b. 配置寄存器GICR_IGROUPRn/GICR_IGRPMODRn将SGI/PPI中断分组为Secure Group 0，Non-secure Group 1，Secure Group 1。
   c. 配置寄存器GICD_IGROUPRn/GICD_IGRPMODRn将SPI中断分组为Secure Group 0，Non-secure Group 1，Secure Group 1。
   d. 配置寄存器GICR_ICFGRn初始化PPIs为电平敏感或者是边沿触发。
   e. 配置寄存器GICD_ICFGRn初始化SPIs为电平敏感或者是边沿触发。
   f. 配置寄存器GICR_IPRIORITYRn设置SGI/PPI中断的优先级。
   g. 配置寄存器GICD_IPRIORITYRn设置SPI中断的优先级。
   h. 配置寄存器GICD_ITARGETSRn设置N/N SPIs的目标处理器。
   i. 配置寄存器GICD_IROUTERn设置1/N SPIs的目标处理器。
2. 配置物理GICC接口。
   a. 配置寄存器ICC_PMR设置屏蔽优先级。
   b. 配置寄存器ICC_BPR、ICC_ABPR设置安全中断和非安全中断的Group Priority。
3. 使能Distributor接口。
   配置寄存器打开相关中断ID使能位。

版权所有 © 海思技术有限公司

a. 配置寄存器GICD_ISENABLERn使能SPI中断。
b. 配置寄存器GICR_ISENABLERn使能SGI/PPI中断。

   注意：每个中断都有1bit对应的使能位，需要在中断配置完成后，再打开中断使能。如需修改中断配置，应先关闭中断对应的使能，配置修改完成后再重新使能。
c. 配置寄存器GICD_CTLR.EnableGrp1、GICD_CTLR.EnableGrp1使能Group1和Group0中断。

4. 使能GICC接口。

   配置寄存器ICC_CTLR使能处理器接口。

12.7.3 GICR\GICD 软件复位流程

由于GIC与CPU core有命令交互，与总线及各外设中断线有交互，因此GIC工作过程中不支持中途单独软复位。

初始化阶段，在CPU core撤离复位后，CPU core通过系统控制器模块对GIC进行解除与复位，具体GIC解复位步骤如下：

步骤1 通过系统控制器进行软件复位撤离，复位撤离需要禁能GIC工作时钟，配置《Hi309a寄存器手册》中的GIC时钟禁能寄存器SC_GIC_CLK_DIS（0x3C4）值为1，等待10us后读寄存器SC_GIC_CLK_ST（0x53C0）是否为0，若是则继续往下，若不是则继续读取，多次读取不为0则视为异常。

步骤2 解复位整个GIC，在系统控制器模块里有一个解复位，配置《 Hi309a 寄存器手册》中的GIC软复位去请求控制寄存器SC_GIC_RESET_DREQ值为1。

步骤3 在配置完**步骤2**寄存器之后等待10μs，读取《 Hi309a 寄存器手册》中的GIC 软复位请求状态寄存器SC_GIC_RESET_ST（0x5A80）进行确认，读取该寄存器中对应bit为0时，表示相应的解复位成功。

步骤4 在查询完GIC软复位请求状态为0，GIC解复位成功后，使能GIC工作时钟，配置《Hi309a 寄存器手册》中的GIC时钟使能控制寄存器SC_GIC_CLK_EN（0x3C0）值为1，等待10us后读寄存器SC_GIC_CLK_ST（0x53C0）是否为1，若是则继续往下，若不是则继续读取，多次读取不为1则视为异常。这样，GIC正式进入工作状态。

复位整个GIC，在系统控制器模块里有一个复位，配置《 Hi309a 寄存器手册》中的GIC软复位请求控制寄存器SC_GIC_RESET_REQ对GIC进行软复位。

----结束

12.7.4 GICR 被软件发现流程

每个CPU core对应的Re-Distributor地址空间发现是通过软扫描GICR_TYPER寄存器，并与CPU core配置值进行匹配，从而确定该Re-Distributor与CPU core的对应关系，具体软件发现流程为：

步骤1 目前一个SCL之间的Re-distributor之间基地址为0x10/0x14/0x18/0x1c…/0x8C，不同SCL之间需要重新找到主CPU core对应的Re-distributor起始地址，同一个SCL的地址空间在一片连续的地址页表空间。

步骤2 软件从主CPU core对应的Re-distributor基地址开始，读以该基地址+GICR_TYPER_L偏移地址寄存器，每次读64bit（包括GICR_TYPER_L、GICR_TYPER_H信息）得到该Re-distributor与CPU core的关系及Re-distributor信息。

步骤3 当软件读到某一个地址的GICR_TYPER_L.GICR_Typer_last == 1时，表示该片连续空间结束，需要从另一片地址空间继续获取Re-distributor信息。

版权所有 © 海思技术有限公司

GICR_TYPER.A3~GICR_TYPER.A0指示的路由层级如表12-17所示。

表 12-17 GICR_TYPER_H 寄存器（Re-distributor affinity 层级）

寄存器名称	成员名称	成员范围	属性	成员初值	成员说明	寄存器名称
GICR_TYPER_H	GICR配置信息寄存器[63:32]	GICR_Typer_a3	31:24	RO	0x00	保留。
-	-	GICR_Typer_a2	23:16	RO	0x00	Re-Distributor的A2的值，bit[2:0]位有效，指示8个*SCL*，其中： bit[5:4]指示*Socket* ID，固定为0； bit[3:2]指示DIE DI，固定为2. bit[1:0]指示clu ID，范围0~3。
-	-	GICR_Typer_a1	15:8	RO	0x00	Re-Distributor的A1的值，指示CPU ID，范围0~7。
-	-	GICR_Typer_a0	7:0	RO	0x00	保留。

A3保留，A2由芯片引脚配置决定，A1/A0表示一个SCL内部的16个CPU core，与16个Re-distributor关系固定一一对应；

Re-Distributor在系统中信息在GICR_TYPER_L寄存器中指示，GICR_TYPER_L.GICR_Typer_procnum指示该CPU core在系统中的唯一序号，具体内容如表12-18所示。

版权所有 © 海思技术有限公司

表 12-18 GICR_TYPER_L 寄存器（Re-distributor）

寄存器名称	成员名称	成员范围	属性	初值	成员说明	寄存器名称
GICR_TYPER_L	GICR配置信息寄存器 bit[31:0]	reserved	31:24	RO	0x00	保留。
-	-	GICR_Typer_procnum	23:8	RO	0x0000	当前CPU core号指示。
-	-	reserved	7:5	RO	0x0	保留。
-	-	GICR_Typer_last	4	RO	0x0	指示在一片连续的地址页表中该Re-Distributor是否是最后的，在一个SCL的地址页表空间是连续的。 0：该Re-Distributor不是最后的； 1：该Re-Distributor是最后的。
-	-	GICR_Typer_distributed	3	RO	0x1	指示是否支持分布式寄存器。 0：不支持分布式寄存器； 1：支持分布式寄存器。
-	-	reserved	2	RO	0x0	保留。

版权所有 © 海思技术有限公司

寄存器名称	成员名称	成员范围	属性	初值	成员说明	寄存器名称
-	-	GICR_Typer_vlpis	1	RO	0x1	指示是否支持VLPIs中断。 0：不支持VLPIs中断； 1：支持VLPIs中断。
-	-	GICR_Typer_lpis	0	RO	0x1	指示是否支持LPIs中断。 0：不支持LPIs中断； 1：支持LPIs中断。

----结束

12.7.5 LPI 中断路由配置流程

步骤1 根据Device个数，进行MAPD命令操作，把所有的Device进行一一映射，ITS就有了Device table 和ITT的对应关系。

步骤2 接着对每个中断进行MAPI/VMAPI/MAPVI/VMAPVI的命令操作，把中断ID映射到Collection ID上，这样每个中断ID就有了中断ID和Collection ID的对应关系。

须知

通过MBI上报的中断只能用MAPVI/VMAPVI进行映射，PCIE上报的中断不受此限制。

1. MBI上报的中断ID为外设私有中断号Device Priavte ID简称（DPID）， MBI上报的中断其中断向量为{DID, DPID }，每个管脚对应的中断号可通过中断向量表查询得到。
2. 外设驱动软件初始化中断时，会根据中断向量表，ITS硬件会通过MAPVI/VMAPVI命令为该中断分配全局的ITS中断号，ITS中断号由系统随机分配，不可指定。
3. 在系统层，该中断会在软件绑定一个Linux irq（软件中断号），Linux irq对应的中断描述符会给出这个中断的ITS硬件中断号、中断控制器以及中断流控等。

步骤3 然后对Collecton 进行MAPC的命令操作，把Collection ID映射到TA（Re-Distributor Target Address）上，这样每个Collection ID就有了Collection ID与目的Core地址的映射关系。

版权所有 © 海思技术有限公司

步骤4 对于需要注册的中断ID，配置该中断ID对应Config table存放的对应Byte：

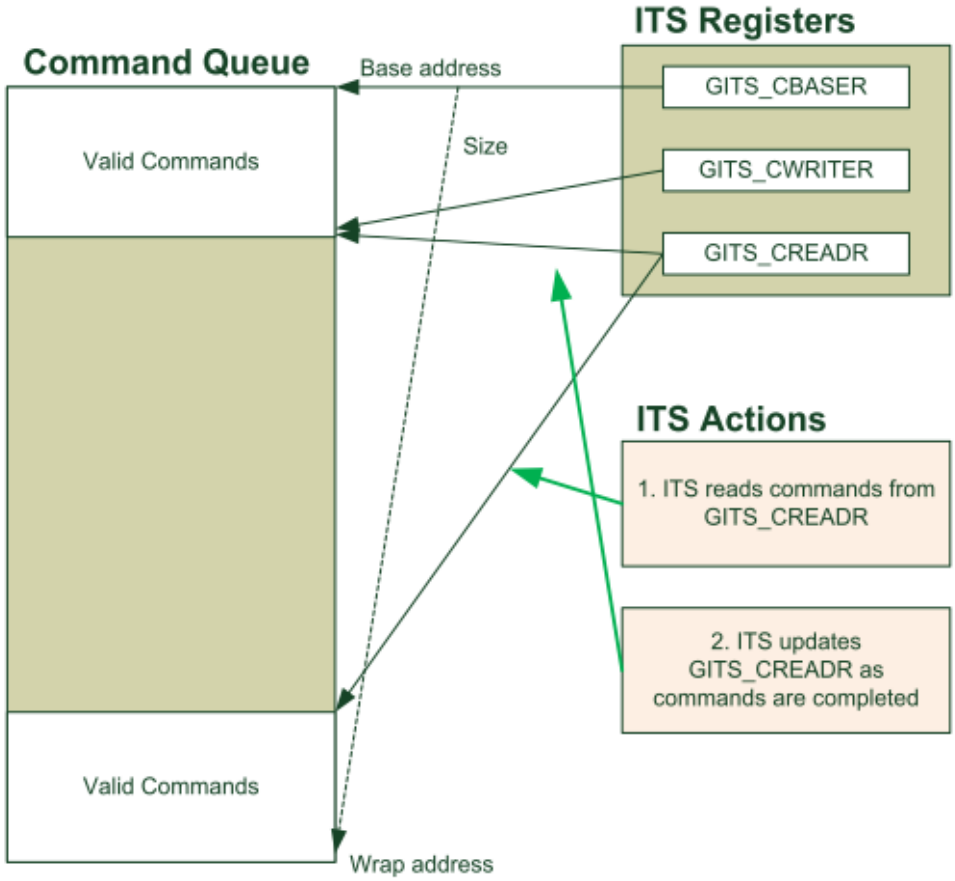
对应Config table内容为：{ Priority（6）， Group = 0b1 ，Enable（1）};

----结束

12.7.6 ITS 中断路由改配流程

12.7.6.1 ITS 配置接口

图 12-15 物理中断配置命令交互流程



如图12-15所示，ITS和CPU通过一个共享的命令队列完成配置命令交互。共享命令队列是循环队列，保存在内存中，由ITS和CPU共同维护，CPU负责命令写入和队列的写指针维护，而ITS则负责读和读指针维护。

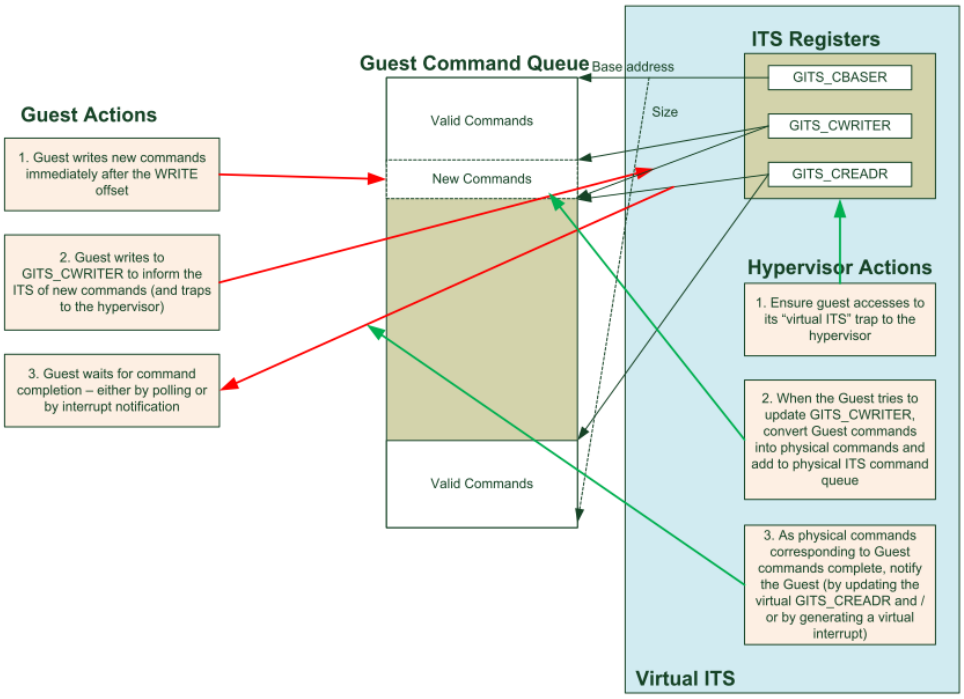
命令队列的处理流程如下：

1. 初始化时外部命令队列Cwrite/Cread均指向0，若Cwrite指针等于Cread指针则表示命令队列空，若Cwrite指针紧挨着Cread指针之后，则表示命令队列满；Cwrite由软件维护，并对硬件可见，Cread由硬件维护，软件可读。
2. CPU检查命令队列空满，若命令队列未满则将配置命令写入Cwrite指针处内存，并将Cwrite指针增1写入ITS Cwrite寄存器。

版权所有 © 海思技术有限公司

3. ITS硬件检查Cwrite寄存器和Cread寄存器是否相等，若不相等表示命令队列有等待处理的命令，则将Cread处的内容读出，并将Cread增1更新到Cread寄存器，如此循环直到Cwrite等于Cread，即命令队列空。

图 12-16 虚拟中断配置命令交互流程



如图12-16所示，Guest os和Host os执行的ITS命令相同，Guest os发出的ITS命令会被Hypervisor捕获，Hypervisor将捕获的Guest OS命令转换成ITS对应的虚拟命令，并将命令写入命令队列，然后更新命令队列写指针，并将写指针发送给硬件。

其中软件是通过CMDQ操作锁实现各个OS的操作互斥，保证各个OS不会同时发起CMDQ队列操作。

12.7.6.2 ITS Remapping or Unmapping 中断

ITS初始化后，如果需要使用中断重定向，那么必须进行如下的操作：

步骤1 确保原来的目的中断已经关闭。

步骤2 发送一个Mapping中断的命令，或者是解除Mapping中断的discard命令。

步骤3 可以通过cread指针、执行SYNC命令和INT中断方式，判断配置命令是否生效。

步骤4 查询到路由配置命令生效后，新中断才能根据新路由发送到新的目的地。

----结束

提醒0：软件必须在查询到路由生效之后，再发中断才能确保中断使用预期的配置上报中断。

提醒1：软件应保证ITT有效建表数占用的内存不大于给ITT表分配的内存大小。

版权所有 © 海思技术有限公司

12.7.6.3 ITS Remapping or Unmapping 设备

ITS初始化后，如果需要使用中断重定向，那么必须进行如下的操作：

步骤1 确保原来旧设备中断已经关闭。

步骤2 发送一个Mapping设备的命令，或者是解除Mapping设备的命令（与Mapping是同一个命令，只是valid 为0）。

步骤3 可以通过Cread指针、执行SYNC命令和INT中断方式，判断配置命令是否生效。

步骤4 完成后，所有中断将不再以旧的Mapping作用。

----结束

提醒0：软件必须在查询到路由生效之后，再发中断才能确保中断使用预期的配置上报中断。

提醒1：采用MAPD命令修改路由时，对应设备的所有中断也需要重新配置。

12.7.6.4 ITS Remapping or Unmapping 中断分组

ITS初始化后，如果需要使用中断分组重定向，那么必须进行如下的操作：

步骤1 发送一个Mapping设备的命令（MAPC），或者UN-Mapping设备的命令（与Mapping是同一个命令（MAPC），只是valid 为0）。

步骤2 如果中断分组被重新映射，则需要发出一个MOVALL命令到GICR。

步骤3 如果中断分组被重新映射，还需要发出一个SYNC命令到GICR。

步骤4 等待SYNC命令完成（可以查询或者中断的方式）。

步骤5 完成后，所有中断将不再以旧的Mapping作用。

----结束

提醒：软件必须在查询到路由生效之后，再发中断才能确保中断使用预期的配置上报中断。

12.7.6.5 ITS Remapping or Unmapping VCPU

ITS初始化后，如果需要使用VCPU重定向，那么必须进行如下的操作：

步骤1 确保所有到该VCPU的中断已经关闭。

步骤2 发送一个Mapping设备的命令（VMAPP），或者UN-Mapping设备的命令（与Mapping是同一个命令（VMAPP），只是valid 为0）。

步骤3 等待之前的命令完成（可以查询或者中断的方式）。

步骤4 完成后，所有中断将不再以旧的Mapping作用。

----结束

提醒：软件必须在查询到路由生效之后，再发中断才能确保中断使用预期的配置上报中断。

12.7.7 GICD 中断路由配置流程

版权所有 © 海思技术有限公司

12.7.7.1 亲和路由模式

由于本硬件不支持GICV2，因此ARE固定为1，使用亲和路由模式，每个中断ID分配1bit由寄存器：

- 配置路由模式寄存器GICD_IROUTERn.IRM选择SPI中断或1/N路由模式。
- SGI路由模式是通过ICC_SGIxR_ELx中的IRM进行选择的。

12.7.7.2 改变 SPI 亲和路由设置流程

软件必须按照以下顺序执行：

步骤1 中断必须置为无效和没有中断处于Active状态。

步骤2 关闭该中断使能。

步骤3 软件通过查询GICD_CTLR.RWP/GICR_CTLR.RWP寄存器确定关断中断使能生效。

步骤4 软件修改SPI中断路由。

步骤5 打开该中断使能。

步骤6 软件通过查询GICD_CTLR.RWP/GICR_CTLR.RWP寄存器确定中断使能生效。

----结束

12.7.8 GICD 中断跨片路由配置流程

支持跨GICD路由的中断类型由两种分别是SPI和SGI，它们的配置流程分别如下。

12.7.8.1 SPI 中断跨 GICD 传递配置流程

SPI跨GICD流程如下：

步骤1 配置ARE使能，只在亲和路由模式下才支持跨SCL中断传递。

步骤2 配置GICD_DIEEN_CROSS寄存器，指示每个SCL是否支持跨SCL中断功能。

步骤3 配置GICD_DIEEN_CROSS_SPI_12N、GICD_DIEEN_CROSS_SPI_N2N寄存器，指示对应的SPI中断是否支持跨SCL中断功能。

步骤4 软件读寄存器GICD_DIEIDR，获取当前SCL的SCLID编号，此SCLID定义和GICR_TYPER_H中A2域定义一致。

步骤5 软件配置中断路由模式（SPI使用GICD_IROUTERN寄存器，指定每个中断ID的跨SCL路由目标SCL和目标CPU core。

- SPI中断：当irm=0，a2为指定的目标SCL号，a1为目标SCL内指定的cluster编号（0~7），a0为指定cluster内的CPU *core*号（0~3）（a0为二进制编码，0~15）；当irm=1，不指定SCL号，选取系统内任意CPU core发送。

步骤6 硬件根据路由模式：

- 如果目标SCL为本SCL，则直接通过stream接口上报。
- 如果目标SCL为其他SCL，通过AXI总线将中断从SourceSCL发给目标SCL，目标SCL收到跨SCL中断之后进行优先级调度，再上报给指定CPU core。

步骤7 软件读IAR寄存器获取中断信息，进入中断处理流程，在中断处理结束之前，对SPI电平中断，需要清除source SCL的中断源。最后写EOI（DIR）寄存器结束中断。

版权所有 © 海思技术有限公司

步骤8 目标SCL的CPU core收到EOI信息后，产生deactivate命令将中断结束信息返回给目标SCL上的GICD，该GICD通过解析清中断信息中的中断ID清除对应的中断pend/active状态。

----结束

12.7.8.2 SGI 中断跨 GICD 传递配置流程

SGI跨GICD流程如下：

步骤1 配置ARE*使能*，只在亲和路由模式下才支持跨SCL中断传递。

步骤2 配置GICR_CROSSSCL_EN寄存器，指示每个SCL是否支持跨SCL中断功能。

步骤3 配置GICD_DIEEN_CROSS_SGI_N2N寄存器，指示SGI中断是否支持跨SCL中断功能。

步骤4 软件读寄存器GICD_SCLIDR，获取当前SCL的SCLID编号，此SCLID定义和GICR_TYPER_H中A2域定义一致。

步骤5 软件配置中断路由模式（SGI使用ICC_SGI0R_EL1, ICC_SGI1R_EL1 and ICC_ASGI1R_EL1寄存器），指定每个中断ID的跨SCL路由目标SCL和目标CPU core。

SGI中断：当irm=0，al2[5:2]为指定的目标SCL号，al2[1:0]为目标SCL内指定的cluster编号（0~7），al1[2:0]为目标SCL内指定的cluster内0~3个CPU core号，TargetList等于al0，该值并不构成路由信息，该值[0]bit只是SGI路由是否有效。

步骤6 同SPI中断**步骤6**。

步骤7 同SPI中断**步骤7**。

步骤8 同SPI中断**步骤8**。

须知

AXI总线在将跨SCL中断写给目标SCL的过程中，需要知道目标SCL的接收地址，由于每个SCL的基地址是可指定的，所以跨SCL中断传递要求将每个SCL的基地址配置到寄存器GICR_CROSSDIE_BASEADDR中，AXI总线写操作的44bit地址构成：{28bit基地址，16bit跨SCL接收/清除寄存器偏移地址}。

每个CPU core针对不同的中断分组有分别的使能控制，如果将中断上报给不存在的CPU core，或者上报给某个在位CPU core但对应使能没打开，都会导致中断不响应会响应超时，造成后续中断阻塞，所以硬件在中断上报给目标CPU core前，会检查对应的使能状态。

----结束

12.7.9 GICR 中断转移流程

12.7.9.1 MOVALL 流程

当要关闭一个物理CPU core时，需要把当前CPU core处理的中断转移到另外一个运行CPU core上去处理，这时候需要执行MOVALL命令，把一个CPU core的全部中断转移到另一个CPU core，中断转移具体按照以下流程进行：

版权所有 © 海思技术有限公司

步骤1 软件给ITS发送MAPC命令，ITS把中断路由关系改配到另一个CPU core。

步骤2 软件给ITS发送MOVALL命令，ITS再给目标Redistributor写MOVALL命令寄存器。

步骤3 Redistributor收到MOVALL命令，解释到目标TA地址，把当前Redistributor处理的所有中断，转移到目标TA。

----结束

须知

软件通过ITS配置转移中断MOVALL命令，有以下几种情况需要注意，否则会造成系统挂死：

1. 对于Redistributor之间的中断转移，软件需要在系统层配置中断转移路径，不能同一时刻让中断转移在各个CPU core之间绕成环路，比如：

   CPU core0-> CPU core1 -> CPU core2 -> CPU core0。
2. 为了保证以上情况不会出现，软件需要全局进行改配，一个MOVALL转移命令发出后，要发出SYNC命令保证MOVALL命令被执行完毕，才能再发出下一个MOVALL转移命令。
3. 软件通过ITS配置MOVALL命令所指定的目标TA地址，与源TA地址不能配置相同，比如：

   CPU core -> CPU core0
4. 如果配置相同，ITS内部会把该MOVALL命令丢弃，并记录错误状态。
5. 软件如果调试Redistributor的MOVALL命令，直接给Redistributor写MOVALL寄存器，也禁止配置的目标TA等于当前Redistributor的TA地址，并且每次的MOVALL命令需要通过读SYNC寄存器保证执行完毕后再发送下一个MOVALL命令。

12.7.9.2 MOVI 流程

当本SCL区ITS的某个中断需要从一个CPU core转移到另一个CPU core上处理时，软件可以通过给ITS发送MOVI命令来实现。

步骤1 软件给ITS发送MOVI命令，ITS执行Re-MAPI把中断路由关系改配到另一个CPU core。

步骤2 ITS执行完Re-MAP后，ITS再给目标Redistributor写MOVI命令寄存器。

步骤3 Redistributor收到MOVI命令，解释到目标TA地址，把当前Redistributor处理的该中断ID转移到另一个CPU core上处理。

----结束

须知

MOVI命令可以支持中断的跨SCL转移，把一个芯片的CPU core上处理的中断转移到另一个芯片的CPU core上处理，但是使用上有以下限制：

整个系统只允许有一个跨SCL MOV命令在执行，不能同时有两个ITS进行跨芯片的MOVI命令，否则有可能造成系统挂死。

版权所有 © 海思技术有限公司

12.7.9.3 虚拟中断重映射流程

描述详情见GICV章节。

12.7.9.4 虚拟核重映射流程

描述详情见GICV章节。

12.7.10 GICR 中断改配流程

（V）LPI config table软件必须执行以下流程：

步骤1 关闭需要修改中断配置的中断使能。

步骤2 修改（V）LPI Config Table中的该中断配置。

步骤3 发生如下命令确保Re-Distributor使用最新的中断配置：

步骤4 当少量的LPI中断配置信息被重新配置，需要下发INV命令。

步骤5 当大量的LPI中断配置信号被重新配置，需要下发INVALL命令。

----结束

说明

当以上的命令没有执行完毕时，中断上报仍然按照旧的配置进行。

12.7.11 虚拟机切换相关流程

12.7.11.1 改变虚拟 CPU 的 Residency 流程

为了让VCPU not-resident，软件必须写GICR_VPENDBASER.Valid = 0；

为了让VCPU resident，软件必须执行以下流程：

步骤1 确保没有其他的VCPU正处于resident状态。（注：即把GICR_VPENDBASER.Valid = 0，不能直接改变GICR_VPENDBASER.Paddr地址）

步骤2 写Cfg Table地址到GICR_VPROPBASER寄存器。

步骤3 写Pending Table地址到GICR_VPENDBASER寄存器。

步骤4 写GICR_VPENDBASER.Valid = 1。

----结束

12.7.11.2 Re-Distributor 对于 Hyeprvisor 切换 VCPU 的工作流程

步骤1 软件配置写GICR_VPENDBASER.Valid = 0，仍然使用outgoing的VCPU的GICR_VPENDBASER.Paddr。

Re-Distributor收到该写操作，执行以下操作：

步骤2 把GICR_VPENDBASER.dirty寄存器置成1，表示正在处理GICR_VPENDBASER.Valid = 0硬件需要做的切换工作。

版权所有 © 海思技术有限公司

步骤3 发Clear命令给CPU接口，等待CPU回Release响应，清除正在上报中断。

步骤4 使用outgoing的GICR_VPENDBASER.Paddr把Cache中数据写回内存（相当于INVALL命令）。

步骤5 以上**步骤1**、**步骤2**、**步骤3**操作完成后把GICR_CTLR.dirty寄存器置成0，告诉软件可以切换Hypervisor。

步骤6 软件读到GICR_VPENDBASER.dirty = 0，写GICR_VPENDBASER.Valid = 1和设置incoming的GICR_VPENDBASER.Paddr进行切换Hyperviso操作。

步骤7 Re-Distributor收到GICR_VPENDBASER.Valid = 1，使用incoming的GICR_VPENDBASER.Paddr进行内存Pending搜索，把搜索到的中断上报。

----结束

12.7.11.3 软件切换 Guest OS 需要执行动作

在GICV3和GICV4，软件处于non-secure EL1时改变Guest OS时Hyperviso必须运行以下操作：

1. 确保Hyeper运行在EL2和没有让即将停止运行的Guest OS转换到non-secure EL1的事件。
2. 在GICV4，写GICR_VPENDBASER.Valid = 0，确保任何虚拟LPI中断被CPU interface释放掉。
3. 保存即将停止运行的Guest OS的中断上下文：
   保存ListRegister内容。
   保存ICH_AP0Rn_EL2内容。
   保存ICH_AP1Rn_EL2内容。
   保存ICH_VMCR_EL2内容。
   保存ICC_SRE_EL1的non-secure备份内容。
4. 重载新的Guest OS的中断上下文：
   重载HCR_EL2内容。
   重载ICC_SRE_EL1的non-secure备份内容。
   重载ICH_VMCR_EL2内容。
   重载ICH_AP1Rn_EL2内容。
   重载ICH_AP0Rn_EL2内容。
   重载List Register寄存器值。
5. 确保新的Guest OS以下事件被重新执行：
   重新写GICR_VPROBASER地址。
   重新写GICR_VPENDBASER地址。

12.7.12 VMOVI 和 VMOVP 相关流程

12.7.12.1 VMOVI 中断迁移流程

VMOVI命令的作用是实现一个中断从一个VCPU到另一个VCPU的迁移。

版权所有 © 海思技术有限公司

步骤1 正确配置该中断的ITS路由表。使用MAPD,VMAPI/VMAPTI，VMAPC命令配置DT,ITT,VCPU entry。

步骤2 软件向CMD Queue里面写VMOVI命令。硬件执行VMOVI修改ITT entry。

步骤3 等待新路由生效，即VMOVI执行完毕。

步骤4 接收中断。ITS硬件产生doorbell中断，GICR硬件判断VCPU是否Resident。

如果Resident就上报该VLPI中断；如果没有Resident，则上报doorbell中断给hypervisor。之后参见**改变虚拟CPU的Residency流程~软件切换Guest OS需要执行动作**小节完成VCPU Resident。

----结束

12.7.12.2 VMOVP 中断迁移流程

VMOVP命令的作用是完成虚拟核从一个物理核到另一个物理核的迁移。

步骤1 正确配置中断的ITS路由表。使用MAPD、VMAPI/VMAPTI，VMAPC命令配置DT、ITT、VCPU entry。

步骤2 软件向CMD Queue里面写VMOVP命令。硬件执行VMOVP修改VCPU entry。

步骤3 等待新路由生效，即VMOVP执行完毕。

步骤4 接收中断。ITS硬件产生doorbell中断，GICR硬件判断VCPU是否Resident；

如果Resident就上报该VLPI中断；如果没有Resident，则上报doorbell中断给hypervisor。之后参见**改变虚拟CPU的Residency流程~软件切换Guest OS需要执行动作**小节完成VCPU Resident。

----结束

12.7.13 注意事项

12.7.13.1 GIC SMMU 属性配置说明

- GIC所有寄存器对应的地址属性均为Device NonCacheable Nonbufferable，因此SMMU中该地址的属性需配置成Device NonCacheable Nonbufferable，否则可能出现未知错误。
- GICR访问SCL内DDR的Cache属性由GICR_PENDBASER、GICR_PROBASER、GICR_VPENDBASER、GICR_VPROBASER寄存器的Cacheability位与SMMU配置共同决定，推荐经过SMMU转换后访问DDR数据的Cache属性为Normal。
- 在SMMU Bypass情况下，GIC访问外设或跨SCL总线操作的Cache属性固定为Device Noncacheable Nonbufferable属性，而访问DDR属性由GICR_PENDBASER、GICR_PROBASER、GICR_VPENDBASER、GICR_VPROBASER寄存器的Cacheability位决定，我们系统在Bypass SMMU模式下的访问DDR Cache属性支持配置为Normal。

12.7.13.2 GIC 内存分配说明

- GIC分配的内存原则：分配的内存和使用该内存的模块在同一个片区，不推荐出现内存跨SCL访问。

版权所有 © 海思技术有限公司

- GIC如果出现跨SCL访问DDR，跨SCL访问DDR的Cache属性必须配置为cacheable，且只允许在一个时刻上经过PA的caccheable队列或noncache队列只有一个访问源头。

12.7.13.3 SPI/SGI/PPI 中断配置变更说明

在配置SPI/SGI/PPI中断过程中，为了避免上报的中断使用到半新半旧的配置，要求软件遵守以下流程：

步骤1 确保没有处于active状态的中断。

步骤2 确保配置修改的中断处于disabled：可以通过写GICD_CTRL寄存器来清除各group中断使能bit，也可以通过写GICD（R）_ICENABLERn清除某个中断ID对应的使能bit。

步骤3 中断去使能之后，软件轮询GICD_CTLR.rwp状态为0后，再进行路由、优先级、group等配置修改。

步骤4 配置完成后再打开中断使能。

----结束

说明

软件配置下发后到硬件生效有一定延迟，如果软件需要确保配置成功才能执行下一步操作，那么需要在配置后增加DSB指令。（DSB指令有另外的专门文档说明，这里不再赘述）。

须知

multiGICD场景配置注意：

软件保证系统中只有一个GICD可以做为主GICD，并且保证在主GICD配置过程中，即GICD_CTLR.rwp为1期间不会切换主GICD，通过从GICD发送配置请求。否则可能出现系统挂死等问题。

12.7.13.4 ITS 中断注册说明

为避免由于ITS中断路由缺失导致的中断丢失，软件必须保证在ITS中断注册之前，中断均处于屏蔽状态，请软件严格遵守如下操作顺序：

步骤1 根据ARM/X86中断设计要求，外设硬件解复位后，中断MASK位必须为1，即默认中断屏蔽。

步骤2 外设驱动调用ITS驱动接口，完成中断注册，建立ITS中断路由表。

步骤3 外设驱动将该中断的MASK位置为0，取消中断屏蔽。

----结束

12.7.13.5 版本兼容性

本硬件不兼容GICV2，使用GICV2模式工作可能出现不可预期的结果。

12.7.13.6 多 P 场景中断上报路径说明

芯片多P封装场景下，中断上报路径选择注意事项：

版权所有 © 海思技术有限公司

1. 建议外设中断通过SCL内的ITS上报，即软件配置外设的LPI中断目标地址为该外设所在的SCL的ITS。
2. 如果不能遵循条件1，请软件保证上报到某ITS的所有LPI中断的中断向量在该ITS内是唯一的。

12.7.13.7 虚拟机中断优先级配置说明

建议配置虚拟中断优先级低于系统核心中断，否则可能出现虚拟机挂死时，可能会导致虚拟中断无法处理，堵塞核心中断上报。

12.7.13.8 GIC/ITS 解复位说明

GIC/ITS的解复位必须在UEFI中控制，否则访问GIC/ITS寄存器时，可能出现总线挂死问题。

12.7.13.9 NNSPI 中断说明

1. 物理位置在Inifinity的N/N SPI中断支持广播给该Inifinity同片的所有核，物理位置在Totem的N/N SPI中断则支持广播给全系统所有核。
2. 该中断配置为多核共享，并非多核独享，因此只有PF权限才可配置NNSPI中断配置，VF不可更改NNSPI中断配置。

12.7.13.10 空中断处理说明

某些特殊场景下，OS可能收到空中断，为保证软件的健壮性，需要软件支持通过硬件状态识别空中断，并退出中断处理。

12.8 内部数据流图

请参见“**典型数据处理流程**”章节描述。

12.9 典型问题定位流程

本小节介绍中断丢失问题定位流程。

12.9.1 中断丢失问题定位

本小节介绍ITS中断错误码并举例说明。

版权所有 © 海思技术有限公司

12.9.1.1 检查 ITS 中断错误码

系统错误状态查询

表 12-19 ITS DFX 寄存器地址

寄存器名称	地址	信号名	位宽	初始值	详细描述
系统错误状态0	0x20010	sys_err0	31:0	0x0	对应系统错误码bit[31:0]，每bit表示ITS出现的一种错误。 0：无错误产生； 1：有错误产生。
系统错误状态1	0x20014	sys_err1	31:0	0x0	
系统错误状态2	0x20018	sys_err2	8:0	0x0	

系统错误寄存器 0

表 12-20 ITS 错误编码 0

地址0x20010	Mnemonic	bit[n]	Descriptions
sys_err0[0]	MAPD_DEVICE_OOR	N = 0	Sys err寄存器bit0表示MAPD命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[1]	MAPI_DEVICE_OOR	1	Sys err寄存器bit1表示MAPI命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[2]	MAPVI_DEVICE_OOR	2	寄存器bit2表示MAPVI命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[3]	MOVI_DEVICE_OOR	3	寄存器bit3表示MOVI命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[4]	DISCARD_DEVICE_OOR	4	寄存器bit4表示DISCARD命令的设备号超出DT.SIZE和device_idbits支持的范围。

版权所有 © 海思技术有限公司

地址0x20010	Mnemonic	bit[n]	Descriptions
sys_err0[5]	INV_DEVICE_OOR	5	INV命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[6]	INT_DEVICE_OOR	6	INT命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[7]	CLEAR_DEVICE_OOR	7	CLEAR命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[8]	VMAPI_DEVICE_OOR	8	VMAPI命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[9]	VMAPVI_DEVICE_OOR	9	VMAPVI命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[10]	VMOVI_DEVICE_OOR	10	VMOVI命令的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[11]	LPI_DEVICE_OOR	11	LPI中断的设备号超出DT.SIZE和device_idbits支持的范围。
sys_err0[12]	MAPI_ID_OOR	12	MAPI命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[13]	MAPVI_ID_OOR	13	MAPVI命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[14]	MOVI_ID_OOR	14	MOVI命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[15]	DISCARD_ID_OOR	15	DISCARD命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[16]	INV_ID_OOR	16	INV命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[17]	INT_ID_OOR	17	INT命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。

版权所有 © 海思技术有限公司

地址0x20010	Mnemonic	bit[n]	Descriptions
sys_err0[18]	CLEAR_ID_OOR	18	Spec遗漏。 CLEAR命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[19]	VMAPI_ID_OOR	19	VMAPI命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[20]	VMAPVI_ID_OOR	20	VMAPVI命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[21]	VMOVI_ID_OOR	21	VMOVI命令的ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[22]	LPI_ ID_OOR	22	LPI中断ID号超出LPI范围或超出对应ITT表支持的范围（ITT.SIZE）。
sys_err0[23]	MAPVI_PHYSICALID_OOR	23	MAPVI命令的PHYSICALID号超出LPI范围且不是一个doorbell中断号，或超出该ITS支持的中断支持范围（idbits）。
sys_err0[24]	VMAPI_PHYSICALID_OOR	24	VMAPI命令的PHYSICALID号超出LPI范围且不是一个doorbell中断号，或超出该ITS支持的中断支持范围（idbits）。
sys_err0[25]	VMAPVI_PHYSICALID_OOR	25	VMAPVI命令的PHYSICALID号超出LPI范围且不是一个doorbell中断号，或超出该ITS支持的中断支持范围（idbits）。
sys_err0[26]	VMOVI_PHYSICALID_OOR	26	VMOVI命令的PHYSICALID号超出LPI范围且不是一个doorbell中断号，或超出该ITS支持的中断支持范围（idbits）。
sys_err0[27]	VMAPVI_VIRTUALID_OOR	27	VMAPVI命令的VIRTUALID号超出LPI范围或超出该ITS支持的中断支持范围（idbits）。
sys_err0[28]	MOVALL_ERR	28	MOVALL命令的DST TA与SRC TA相同。

版权所有 © 海思技术有限公司

地址0x20010	Mnemonic	bit[n]	Descriptions
sys_err0[29]	MAPI_COLLECTION_OOR	29	MAPI命令的COLLECTION号超出CT.SIZE支持的范围。
sys_err0[30]	MAPVI_COLLECTION_OOR	30	MAPVI命令的COLLECTION号超出CT.SIZE支持的范围。
sys_err0[31]	MOVI_COLLECTION_OOR	31	MOVI命令的COLLECTION号超出CT.SIZE支持的范围。

系统错误寄存器 1

表 12-21 ITS 错误编码 1

地址0x20014	Mnemonic	bit[n]	Descriptions
sys_err1[0]	INVALL_COLLECTION_OOR	32	INVALL命令的COLLECTION号超出CT.SIZE支持的范围。
sys_err1[1]	VMOVP_VCPU_OOR	33	VMOVP命令的VCPU号超出VCT.SIZE支持的范围。
sys_err1[2]	VMAPI_VCPU_OOR	34	VMOVP命令的VCPU号超出VCT.SIZE支持的范围。
sys_err1[3]	VMAPVI_VCPU_OOR	35	VMOVP命令的VCPU号超出VCT.SIZE支持的范围。
sys_err1[4]	VINVALL_VCPU_OOR	36	VMOVP命令的VCPU号超出VCT.SIZE支持的范围。
sys_err1[5]	VSYNC_VCPU_OOR	37	VSYNC命令的VCPU号超出VCT.SIZE支持的范围。
sys_err1[6]	VMAPP_VCPU_OOR	38	VMAPP命令的VCPU号超出VCT.SIZE支持的范围。
sys_err1[7]	VMOVI_ VCPU _OOR	39	SPEC错误应该是VCPU_OOR: VMOVI命令的VCPU号超出VCT.SIZE支持的范围。
sys_err1[8]	MAPD_ITTSIZE_OOR	40	MAPD命令的ITT.SIZE超出该ITS支持的中断支持范围（idbits）。
sys_err1[9]	VMAPP_VPTSIZE_OOR	41	VMAPP命令的ITT.SIZE超出该ITS支持的中断支持范围（idbits）。

版权所有 © 海思技术有限公司

地址0x20014	Mnemonic	bit[n]	Descriptions
sys_err1[10]	MAPI_UNMAPPED_DEVICE	42	MAPI映射某设备中断，该设备DT表项无效。
sys_err1[11]	MAPVI_UNMAPPED_DEVICE	43	MAPVI映射某设备中断，该设备DT表项无效。
sys_err1[12]	MOVI_UNMAPPED_DEVICE	44	MOVI映射某设备中断，该设备DT表项无效。
sys_err1[13]	DISCARD_UNMAPPED_DEVICE	45	DISCARD映射某设备中断，该设备DT表项无效。
sys_err1[14]	INV_UNMAPPED_DEVICE	46	INV映射某设备中断，该设备DT表项无效。
sys_err1[15]	INT_UNMAPPED_DEVICE	47	INT映射某设备中断，该设备DT表项无效。
sys_err1[16]	CLEAR_UNMAPPED_DEVICE	48	CLEAR映射某设备中断，该设备DT表项无效。
sys_err1[17]	VMAPI_UNMAPPED_DEVICE	49	VMAPI映射某设备中断，该设备DT表项无效。
sys_err1[18]	VMAPVI_UNMAPPED_DEVICE	50	VMAPVI映射某设备中断，该设备DT表项无效。
sys_err1[19]	VMOVI_ UNMAPPED_DEVICE	51	VMOVI映射某设备中断，该设备DT表项无效。
sys_err1[20]	LPI_ UNMAPPED_DEVICE	52	LPI中断的设备DT表项无效。
sys_err1[21]	MOVI_UNMAPPED_INTERRUPT	53	MOV的源中断ID对应的ITT表项无效。
sys_err1[22]	INV_UNMAPPED_INTERRUPT	54	INV的中断ID对应的ITT表项无效。
sys_err1[23]	INT_UNMAPPED_INTERRUPT	55	INT的中断ID对应的ITT表项无效。
sys_err1[24]	CLEAR_UNMAPPED_INTERRUPT	56	CLEAR的中断ID对应的ITT表项无效。
sys_err1[25]	VMOVI_UNMAPPED_INTERRUPT	57	VMOVI的中断ID对应的ITT表项无效。
sys_err1[26]	LPI_ UNMAPPED_INTERRUPT	58	LPI中断ID对应的I。 TT表项无效。
sys_err1[27]	MOVI_UNMAPPED_COLLECTION	59	需MOVI操作的CID缺少配置。

版权所有 © 海思技术有限公司

地址0x20014	Mnemonic	bit[n]	Descriptions
sys_err1[28]	INVALL_UNMAPPED_COLLECTION	60	需INVALL操作的CID缺少配置。
sys_err1[29]	INV_ITE_INVALID	61	修改为INV_CTE_OR_VCTE_INVALID。
sys_err1[30]	INT_ITE_INVALID	62	修改为INT_CTE_OR_VCTE_INVALID。
sys_err1[31]	CLEAR_ITE_INVALID	63	修改为CLEAR_CTE_OR_VCTE_INVALID。

系统错误寄存器 2

表 12-22 ITS 错误编码 2

地址0x20018	Mnemonic	bit[n]	Descriptions
sys_err2[0]	LPI_CTE_OR_VCTE_INVALID	64	CTE_OR_VCTE_INVALID。
sys_err2[1]	VINVALL_VCPU_INVALID	65	需VINVALL操作的VCPU缺少配置。
sys_err2[2]	VSYNC_VCPU_INVALID	66	需SYNC操作的VCPU缺少配置。
sys_err2[3]	VMOVP_VCPU_INVALID	67	需修改的VCPU配置项不存在。
sys_err2[4]	VMOVI_ITEVCPU_INVALID	68	需VMOVI操作的VCPU缺少配置。
sys_err2[5]	VMOVI_CMDVCPU_INVALID	69	命令中的配置无效。
sys_err2[6]	MOVI_ID_IS_VIRTUAL	70	用物理命令修改虚拟路由表。
sys_err2[7]	VMOVI_ID_IS_PHYSICAL	71	用虚拟命令修改物理路由表。
sys_err2[8]	MOV_DEST_SRC_IS_SAME	72	源和目的地址相同的配置错误。
sys_err2[9]	MOVP_DEST_SRC_IS_SAME	73	源和目的地址相同的配置错误。

版权所有 © 海思技术有限公司

12.9.1.2 LPI 中断丢失定位实例

以SCCL ITS上报中断到SCCL GICR为例，假设its_ba=0x0004C000000其他子系统请更换基址。

1. 查询外设发出的中断的统计。
2. 查询ITS寄存器，偏移地址0x20008，观察ITS收发统计。
3. GICR收发检查。
   a. 读GICR 地址0x3700，观察GICR是否正在发送状态。
   b. 读GICR 地址0x3708，观察GICR状态机中保存的LPIID。
   c. 读GICR 地址0x3710，观察GICR接收到的中断次数。
   d. 读GICR 地址0x3718，观察GICR中被合并的中断次数。
4. CPUIF收发检查。
   a. 读GICD 地址0x3510，观察中断被GICC接收的次数。
   b. 读GICD地址0x3514，观察中断被CPU接收的次数。
   c. 读GICD地址0x3518，观察中断被CPU处理完成的次数。
5. 检查以上统计是否相等；如果不相等，则可确认中断在该路径上丢失。
6. 如果ITS 0x20008偏移寄存器显示中断丢失，则可查ITS 0x4c020010/0x4c020014/ 0x4c020018看是否发生系统异常，确认ITS路由配置缺失问题。
7. 如果ITS错误码显示有错误，则需通过软件查DDR路由表数据，确定路由缺失的原因和数据：

   例如：某lpi中断（deviceid/eventid）上报不上，ITS错误码显示错误配置，定位具体路由缺失原因：

   查0x0104/0x0100 获得devicetable_ba。

须知

GITS_BASERn. Indirect为0则指示本表为大表，GITS_BASERn. PA为某路由表的基地址；GITS_BASERn. Page_Size

GITS_BASERn. Indirect为1时，GITS_BASERn. PA为某路由表pointer table的基地址，某路由表的pointer table的表项指向device table；GITS_BASERn. Page_Size指定每个表项指向内存的大小，本硬件固定为4k，某路由表pointer table entry的格式如表12-23所示。

表 12-23 Device/VCPU/PCPU Table Pointer Table Entry 数据结构表

数据结构	字段范围	描述
l2t.pa	47：12	2级页表的物理地址的bit[12]~bit[47]
Res	11：0	2级页表的物理地址的bit[0]~bit[11]固定为0，该字段被当做0处理；

版权所有 © 海思技术有限公司

查device table:地址{devicetable_ba,12’B0}+ {deviceid，0b0}的内容，格式如下表12-24：

表 12-24 Device Table Entry 数据结构表

数据结构	字段范围	描述
Device Valid	45	宽度：1bit指示 Device Table Entry是否有效；
ITT Size	44：40	宽度：5bit指示 ITT Table的大小，为支持中断ID数的bit数减1；
ITT Address	39：0	宽度：40bit 指示ITT Table的基地址，48bits的物理地址，取bit[47:8]作为有效寻址。

从Device Table Entry中获取ITT Address，示例：
Device table addr is 0xFFFFFFC01AC35008
00000000: 10 cd 1a 00 f0 02 00 00（大端格式） 12 cd 1a 00 f0 02 00 00
00000010: 14 cd 1a 00 f0 02 00 00 16 cd 1a 00 f0 02 00 00
查ITT table: 地址{itt_ba,8’B0}内容，格式如表12-25所示。

表 12-25 ITT Table Entry 数据结构表

数据结构	字段范围	描述
ITT Table Valid	49	宽度：1bit指示ITT Entry是否有效。
IRQ TYPE	48	宽度：1bit指示中断类型，读取对应的配置字段。 0：表示物理中断映射表； 1：表示虚拟中断映射表。
Virtual ID	47:32	宽度：16bit虚拟中断号。
VCID	31:16	宽度：16bit IRQ TYPE为1时，VCID表示虚拟中断上报的VCPU号，系统最大支持256VCPUs； IRQ TYPE为0时，VCID表示物理中断上报的CID分组号。
Physical ID	15:0	宽度：16bit物理中断号。

ITT table addr is 0xFFFFFFC01ACD1000
00000000: 00 00 00 00 00 00 00 00 00（异常）00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

版权所有 © 海思技术有限公司

ITT table addr is 0xFFFFFFC01ACD1600

00000000: 8f 32 00 00 00 00 00 02 00 00 00 00 00 00 00 00

00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

a. 如上发现ite数据为空，即说明配置异常。

b. 查0x0114/0x0110 获得ct_ba。

c. 查0x010c/0x0108 获得vct_ba。

d. ITT表bit48为1时，根据VCID查询CT table: 地址{vct_ba,12’B0}+ {VCID，0b0}内容，格式如表12-26所示。

表 12-26 VCPU Table Entry 数据结构表

数据结构	字段范围	描述
VCPU Table Valid	69	宽度：1bit VCPU Table Entry是否有效；
VPTS	68：64	宽度：5bit虚拟中断Pending TableTable的大小，为支持中断ID数的bit数减1；
VPTA	63：32	宽度：32bit 某VCPU对应的虚拟中断Pending Table基地址。48bits的物理地址，取 bit[47:16] 作为有效寻址。
TA Address	31：0	宽度：32bit GICR的基地址，48bits的物理地址，取 bit[47:16] 作为有效寻址。GICR与GICC是一一对应的，该地址表示VCPU目前运行在哪个物理CPU Core上。

e. ITT表bit bit[48]为0时，根据VCID查询CT table: 地址{vct_ba,12’B0}+ {VCID，0b0}内容，格式如下表12-27所示。

表 12-27 Collection Table Entry 数据结构表

数据结构	字段范围	描述
Collection Table Valid	32	宽度：1bit Collection Table Entry是否有效；
TA Address	31：0	宽度：32bit GICR的基地址，48bits的物理地址，取 bit[47:16] 作为有效寻址。GICR与GICC是一一对应的，该地址表示VCPU目前运行在哪个物理CPU Core上。

如果GICR接受和CPUIF发送统计不相等：

版权所有 © 海思技术有限公司

则确认该LPI中断配置是否正确：

1. 配置地址为：GICRn_PROPBASER.PA +（ LPIID-0x2000 ）。
2. 每个中断配置占8bit，配置格式为：{Priority[5:0]，Group，Enable}，注意Group必须固定配置为1，否则可能出现异常。
3. 并确认该LPI中断Pending状态是否被置位。
4. Pending table地址为：GICRn_PENDBASER.PA +（ LPIID/0x8 ）。
5. 每个中断状态占1bit，表示中断是否需要发送。

CPUIF发送和接收统计不相等：则需查询GICD_CPUIF状态确认GICC是否存在反压：

DFX地址：0x7000+0x0040*cpu_num。

12.10 时钟复位

- GIC/ITS时钟与CPU时钟同源，频率为CPU时钟的2或3分频，默认配置为2分配。
- GIC/ITS仅支持上电复位、全局系统复位。
- GIC/ITS不支持模块单独软复位，跟随SCL的复位或整芯片复位。

版权所有 © 海思技术有限公司

13 SysCtrl

13.1 概述

系统控制器主要用于控制系统运行的模式，控制系统运行状态，管理系统中的重要模块（如时钟、复位、软复位、安全调试控制等），完成对外设的某些功能的配置与状态读取查询。在芯片中，除了一个SysCtrl外，每个子系统还对应有一个子系统控制器。

13.2 关键术语

表 13-1 SysCtrl 关键术语描述

术语	描述
SysCtrl	System Controller，系统控制器。
SubCtrl	Subsystem Controller，子系统控制器。
MBIST	Memory Build-In SelfTest，片上SRAM内建自测试。
DIE ID	芯片生产阶段烧写在eFuse中的每颗die的唯一标示。
Normal World	Normal World与Secure World对应，是CPU为了支持系统安全特性而区分的两种处理器状态。
DAW	Dynamic Address Window，滑动地址窗口。
SD5011	本文中提到的SD5011均指Hi309a，不包括PLC die。

13.3 特性

系统控制器具有以下特性：

- 控制并监控系统的运行模式。

版权所有 © 海思技术有限公司

- 提供系统时钟控制和状态查询。
- 提供外设的时钟源选择、时钟门控的控制、复位请求的控制,以及相应状态的查询。
- 提供CPU工作模式控制。
- 提供系统安全调试等级控制。
- 支持功能模式下调用DFT MBIST。
- 部分SubCtrl支持TSensor工作状态指示、温度查询、高低温门限配置和过热/过冷保护中断。
- SysCtrl提供芯片的DIE ID寄存器。
- SysCtrl提供各个外设的功能配置接口。
- SysCtrl和各个SubCtrl提供寄存器锁定功能。
- SysCtrl提供PLL失锁检测记录寄存器，支持PLL失锁上报中断。
- 提供不同CPU DIE的CoreSight内System Counter的同步功能。
- 支持与板级SD6137、BBL DPLL配合的整体时钟方案，包括1588相关中断，时钟选源切换，PLL配置。
- 支持通过管脚输入获得板级异常状态，在汇总后对外输出功放保护信号来控制功放关闭。
- 支持RGPS串口通信，支持2个RGPS_1PPS中断，支持RGPS_1PPS、RGPS_TX、RGPS_RX信号状态寄存器可读，RGPS_TX支持发送电平可配置，支持RGPS_RX上升沿检测计数。
- 支持掉电告警，SD6225将掉电告警汇聚后通过中断上报给Hi309a。
- 支持标识管脚（AGING_MARK、PWR_DEVICE_MARK、FT_IND_IN、FT_IND_OUT）及指示灯管脚（LED_RUN、LED_ALM、LED_ACT、LED_AAU_BBL_RUN、LED_AAU_BBL_ALM、LED_AAU_BBL_ACT），支持指示灯源选择。
- 支持CPU配置远程下电管脚来控制整个单板下电。

DFX特性：

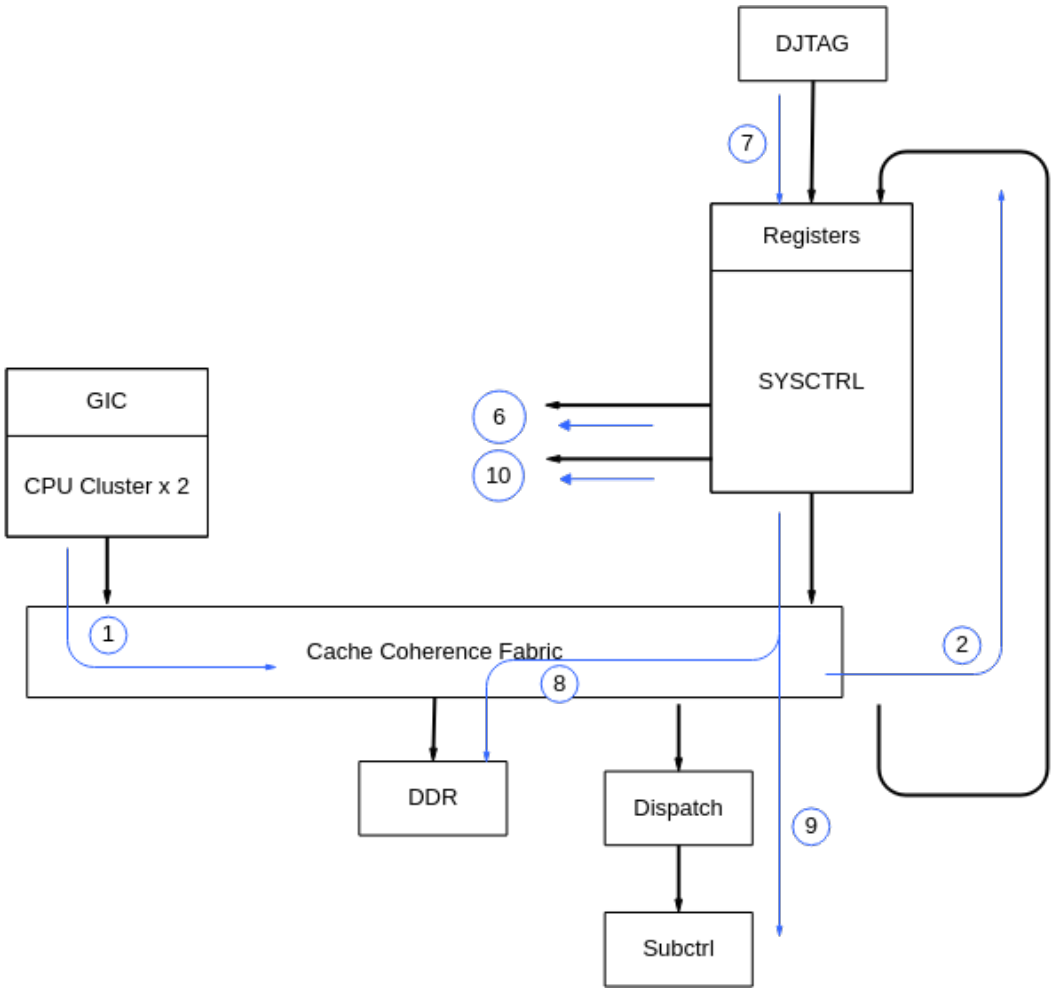
- 支持AXI Master接口，软件可下发指令，控制AXI Master口发出对DDR内存的访问操作。
- 支持DJTAG Master接口，软件可下发指令，控制DJTAG Master口发出对系统中各个模块寄存器的读写操作。
- 支持Debug Bus接口，外部调试设备可通过DJTAG TAP控制器访问该Debug Bus接口，配置并访问SYSCTRL所有寄存器，也可通过DJTAG TAP控制器访问Debug Bus接口控制AXI Master启动。
- SysCtrl提供芯片上电采样到的单板配置信息查询。

13.4 功能描述

版权所有 © 海思技术有限公司

13.4.1 应用场景

图 13-1 SysCtrl 应用图



SysCtrl控制着整个系统的运行，所有模块的时钟和复位等。如图13-1所示，CPU通过①和②配置SysCtrl， SysCtrl根据输入的各种状态和配置信息，通过⑥以硬连线的方式输出各种控制信息，DJTAG通过⑦访问Debug Bus的方式可配置和访问SysCtrl内寄存器，SysCtrl以AXI Master的方式通过⑧访问DDR内存，通过⑨配置和访问SubCtrl内的寄存器、广播Event、同步System Counter，SysCtrl以DJTAG Master的方式通过⑩访问系统各模块的寄存器。

13.4.2 相关模块

在系统中，SysCtrl模块会和下面多个模块进行交互，分别描述如下：

- CPU Core：CPU Core通过系统总线发送各种配置信息、获取各种状态；SysCtrl通过硬连线的方式控制CPU Core的运行模式。
- DJTAG：DJTAG可配置和访问SysCtrl内所有寄存器。
- DDRC：SysCtrl可通过AXI Master接口访问DDR。
- 各模块：SysCtrl通过硬连线的方式提供各外设模块配置信息。

版权所有 © 海思技术有限公司

13.4.3 软复位控制

系统控制器支持对芯片全局以及局部模块进行软复位：

- 配置系统状态寄存器SC_GLB_RESET_REQ时，系统控制器将发出全局软复位请求给片内复位模块。
- 当配置系统控制器的相应模块软复位控制位后，系统控制器将控制片内复位模块对这个模块进行复位。相应的模块软复位控制位请参见系统控制器的寄存器描述部分。

13.4.4 模块时钟门控

通过SysCtrl和SubCtrl可以控制每个模块的时钟的开启和关闭。

- SC_XX_CLK/ICG_EN为时钟使能寄存器，WO类型，高有效，写入1表示开启XX模块的时钟，写入0，时钟状态不改变。
- SC_XX_CLK/ICG_DIS为时钟禁止寄存器，WO类型，高有效，写入1表示关闭XX模块的时钟，写入0，时钟状态不改变。
- SC_XX_CLK/ICG_ST为时钟状态寄存器，RO类型，表示XX模块时钟开启状态，读出1表示XX模块的时钟处于开启状态，读出0表示XX模块的是在处于关闭状态。

如当需要开启BMU时钟时，向SC_BMU_CLK_EN写入1，经过3~5个时钟周期后，BMU模块的时钟被开启；当需要关闭BMU时钟时，向SC_BMU_CLK_DIS写入1，经过3~5个时钟周期后，BMU模块的时钟被关闭。

须知

1. 在模块初始化阶段，首先需要配置寄存器SC_XX_CLK/ICG_EN打开模块的时钟。
2. 单个模块时钟被门控后，系统不保证访问其slave口的操作不挂死，需要软件保证不会有访问该slave口的操作。
3. 如果同一个Dispatch口下的所有模块均不需要工作，需要软件先配置相关模块的软复位请求，再配置相应Dispatch二级DAW将其窗口使能（dispatch_daw_en的相应窗口使能）关闭，其后再配置相关模块时钟门控寄存器将模块时钟关闭，这样可以保证CPU访问该Dispatch口下面的所有slave模块不挂死，响应默认返回OK，读数据返回全F。

13.4.5 模块软复位控制

通过SysCtrl和SubCtrl可以控制每个模块的软复位。

- SC_XX_RESET_REQ为复位请求控制寄存器，WO类型，高有效，写入1表示向XX模块的CRG发出软复位请求，写入0，不改变复位状态。
- SC_XX_RESET_DREQ为复位去请求控制寄存器，WO类型，高有效，写入1表示向XX模块的CRG撤销软复位请求，写入0，不改变复位状态。
- SC_XX_RESET_ST为复位状态寄存器，RO类型，表示XX模块复位状态，读出1表示XX模块的时钟处于复位状态，读出0表示XX模块的是在处于正常工作状态。

如当需要将BMU置为复位状态时，向SC_BMU_RESET_REQ写入1，BMU模块的复位信号立即有效；当需要将BMU置为正常工作状态时，向SC_BMU_RESET_DREQ写入1，经过3~5个时钟周期后，BMU模块的复位信号将无效。

版权所有 © 海思技术有限公司

说明

在访问模块Slave之前，需要先配置对应寄存器SC_XX_RESET_DREQ将该模块解复位。

13.4.6 安全控制

SC_DBG_AUTH_CTRL[cpu_spniden]、SC_DBG_AUTH_CTRL[cpu_spiden]、SC_DBG_AUTH_CTRL[cpu_niden]、SC_DBG_AUTH_CTRL[cpu_dbgen]分别控制各个核的安全特权模式non-invasive调试、安全特权模式invasive调试、非安全特权模式non-invasive调试和非安全特权模式invasive调试。该寄存器高有效，1表示可以通过调试访问系统内安全寄存器或控制CPU运行。

上电复位和自刷新复位后，该寄存器默认状态是开启安全调试模式，因此，在CPU安全启动后，进入Normal模式前，需要将该寄存器清零，关闭安全调试。

寄存器SC_AXI_MSTR_SEC_ACC_EN控制SysCtrl发起的AXI安全访问。

寄存器SC_DJTAG_SEC_ACC_EN控制SysCtrl发起的DJTAG Debug bus安全访问。

13.4.7 功能模式 MBIST

芯片中集成了若干MBIST Controller，均由DFT工具产生，其中CLUSTER、LLC可实现repair功能，所有MBIST Controller的启动均由TDRE电路实现，用于在DFT模式下对片上Memory做自测试。在功能模式下通过SysCtrl也可以启动这些MBIST Controller，用于在系统运行前对MBIST进行自检。

在功能模式下可通过SysCtrl启动MBIST Controller，首先需要将SysCtrl寄存器配置的CPU时序转换成JTAG时序，再由TDRE启动MBIST Controller。

DJTAG TAP控制器可实现LLC、HHA的MBIST或Repair功能，配置流程同DJTAG流程：

DFT集成MBIST Controller也可通过SysCtrl启动，启动方式为配置SysCtrl寄存器，控制DFT的TDRE启动MBIST。

- 寄存器SC_CRG_RESET_REQ的srst_req_func_mbist（SysCtrl偏移地址0xA80）控制MBIST复位，配置为1，撤离复位。
- 寄存器SC_MBIST_CPUI_ENABLE（SysCtrl偏移地址0xF30C）控制CPU启动MBIST信号。
- 寄存器SC_MBIST_CPUI_DATAIN（SysCtrl偏移地址0xF310）控制TMB TDRE电路数据和指令。
- 寄存器SC_MBIST_CPUI_WRITE_EN（SysCtrl偏移地址0xF314）控制TDRE CPU接口读写。
- 寄存器SC_MBIST_CPUI_SMS_FUNC_RESET（SysCtrl偏移地址0xF318）控制MBIST电路是否处于复位状态。
- 寄存器SC_MBIST_CPUI_DATAOUT（SysCtrl偏移地址0xF320）CPU启动MBIST模式，TMB电路测试结果观测接口。

具体配置流程请参考“**系统DFX**”。

13.4.8 DIE ID 读取

Hi309a一共有3块功能eFUSE，通过SYSCtrl寄存器EFUSE_SLT_VERSION~EFUSE2_2047_2016可以获取烧入到3块eFUSE的可读的DIE ID信息。

版权所有 © 海思技术有限公司

13.4.9 TSensor 接口

通过POE SubCtrl、ETH SubCtrl、SPPE SubCtrl、SERDES SubCtrl和CPU_CLUSTER SUBCTRL寄存器SC_TSENSORx_CTRL配置片上温度传感器工作参数，通过寄存器SC_TSENSORx_TEMP_SAMPLE可以读取片上温度传感器的工作指示和温度值。

可通过设定温度门限，以中断方式实现超高温、高温过热保护和过冷保护功能。

13.4.10 外设的功能配置接口

通过SysCtrl/SubCtrl可提供对各个外设接口的功能配置。

13.4.11 寄存器锁定功能（安全）

- 往SysCtrl或SubCtrl的寄存器SC_SYSCTRL_LOCK（偏移0xF100）写入0x5a5a_a5a5，此时，SysCtrl或SubCtrl的所有寄存器为锁定状态，无法写入，只能读取；往该寄存器写入其他值，对SysCtrl或SubCtrl的状态不产生影响，所有寄存器继续保持原有的状态。该寄存器只有安全操作才能进行访问，默认为0，状态为未锁定。
- 往SysCtrl或SubCtrl的寄存器SC_SYSCTRL_UNLOCK（偏移0xF110）写入0xa5a5_5a5a，此时，SysCtrl的所有寄存器被解锁；往该寄存器写入其他值，对SysCtrl或SubCtrl的状态不产生影响，所有寄存器继续保持原有的状态。该寄存器只有安全操作才能进行访问，默认为0，状态为未锁定。

说明

寄存器锁定功能在Hi309a中对除了时钟门控和软复位以外的寄存器生效。

13.4.12 PLL 失锁检测记录寄存器

SYSCTRL寄存器SC_PLL_SRC_INT记录PLL失锁。

当硬件检测到PLL失锁后，SC_PLL_SRC_INT会被置位，CPU可以通过写1清除。在芯片非上电复位过程中，该寄存器的值保持不变。

寄存器SC_PLL_SRC_INT可获取PLL的原始锁定状态，寄存器SC_PLL_LOCK_STATUS可获取PLL失锁中断状态。

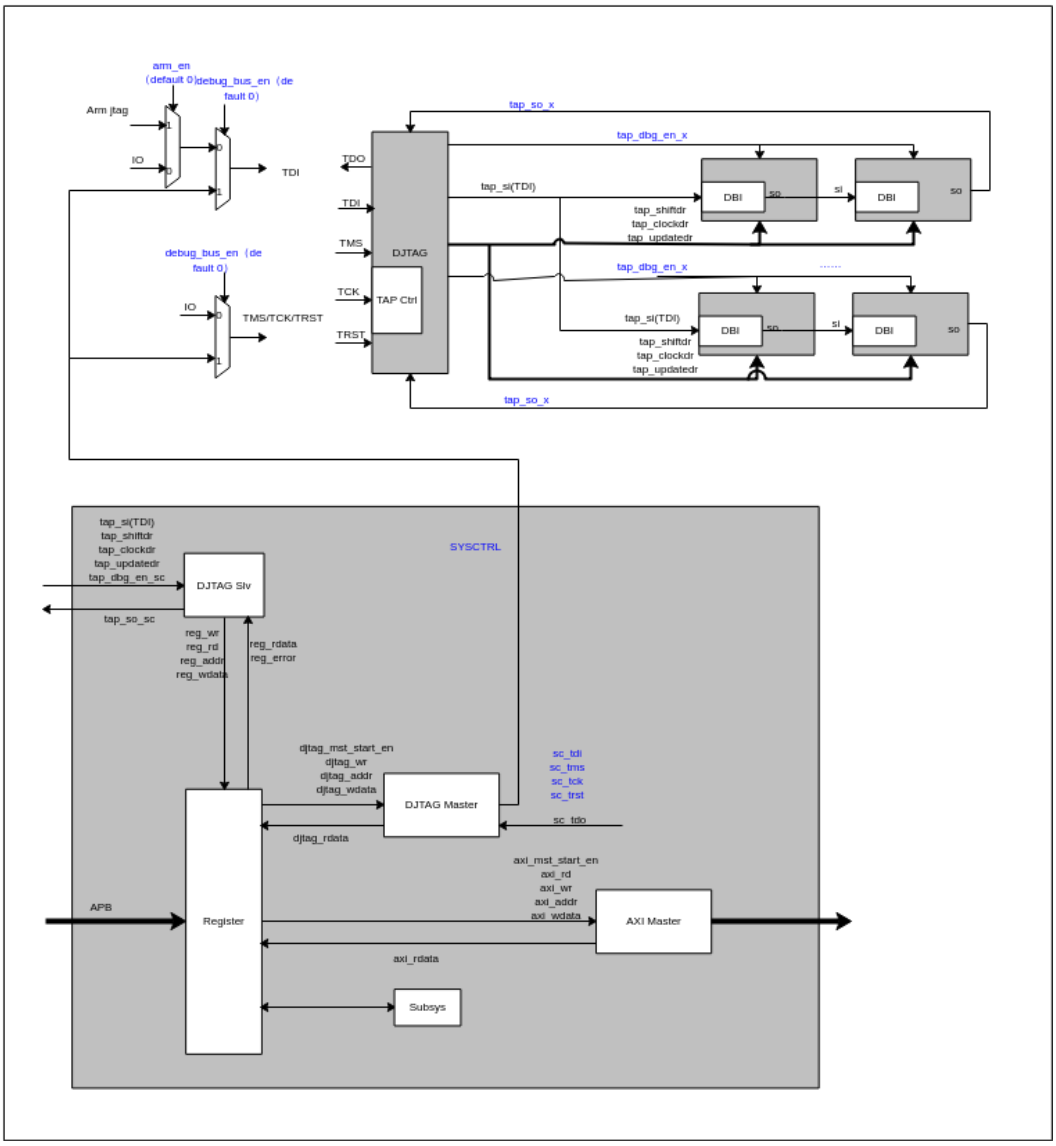
13.4.13 DJTAG Master

DJTAG Master为SysCtrl模块内发起JTAG TAP口时序的Master控制器，主要应用为CPU通过APB总线配置SysCtrl内的DJTAG组相关寄存器（偏移地址0x6300~0x6440），SysCtrl按照该组寄存器对应的模块地址，产生模块对应的访问时序，访问和配置对应模块的寄存器。

对应的应用框图如图13-2所示。

版权所有 © 海思技术有限公司

图 13-2 DJTAG Master 应用框图



不同的模块串联在不同的扫描链中，各个模块在DJTAG Master中对应的地址见地址映射表中的DJTAG章节，将该地址配置到寄存器SC_DJTAG_MSTR_CFG / debug_module_sel[7:0]。

总线模块Cross Station、Acess Agent使用CFG_BUS访问，CFG_BUS具有Broadcast特性，可以通过DJTAG Mstr配置CFG_BUS的Broadcast Chain，即只需要配置一次寄存器，就可以将该配置应用于同一类总线模块，具体请参考地址映射表中的CFG_BUS章节。

DJTAG Master访问Debug bus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（偏移地址0x6300）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 CPU配置寄存器SC_DJTAG_MSTR_ADDR（偏移地址0x6308），写入被访问模块的内部寄存器的偏移地址。

版权所有 © 海思技术有限公司

步骤3 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0x630C），写入写数据（如果是读则省略该步骤）。

步骤4 CPU配置寄存器SC_DJTAG_MSTR_CFG（偏移地址0x6310），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，模块的地址可从架构图的DJTAG章节获得。
3. djtag_mstr_wr：该次DTJTAG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5 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0x6314）为1，使能DJTAG访问。

步骤6 CPU访问寄存器SC_DJTAG_OP_ST（偏移地址0x6428）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0x6428）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_DATA0~6（偏移地址0x6400~0x6418）寄存器内，否则读数据通过寄存器SC_DJTAG_RD_DATA0（偏移地址0x6400）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，EMU和FPGA会裁剪掉某些模块导致版本中的DJTAG链和文档描述的DJTAG链不一致时需要特别注意链长。

----结束

13.4.14 AXI Master

AXI Master为SysCtrl模块内发起AXI时序的Master控制器，主要应用为CPU通过APB总线配置SysCtrl内的AXI Master相关寄存器，SysCtrl按照该组寄存器对应的模块地址，产生模块对应的访问时序，访问和配置对应模块的寄存器。

AXI Master访问配置步骤如下：

步骤1 CPU配置寄存器SC_AXI_MSTR_USER（偏移地址0xDA04），设置本次AXI操作的属性。

步骤2 CPU配置寄存器SC_AXI_MSTR_DATA0~3（偏移地址0xDA08~0xDA14），写入写数据，若为读操作，则忽略该步骤。

步骤3 CPU配置寄存器SC_AXI_MSTR_WR_USER（偏移地址0xDA18），设置本次AXI操作的读写属性，0为读，1为写。

步骤4 CPU配置寄存器SC_AXI_MSTR_ADDR0~1（偏移地址0xDA1C~0xDA20），写入需要访问的地址。

步骤5 CPU配置SC_AXI_MSTR_START_EN（偏移地址0xDA24）为1，使能AXI Master发出操作。

步骤6 CPU访问，寄存器 SC_AXI_MSTR_START_EN（偏移地址0xDA24）为0则表示AXI访问结束，如果为读操作，读取寄存器SC_AXI_RD_DATA0~3（偏移地址

版权所有 © 海思技术有限公司

0xEA00~0x0xEA0C ），获取读数据，访问寄存器SC_AXI_MSTR_ST（偏移地址0xEA10 ），判断bit[3:1]是否为0，为0表示访问正常，不为0表示访问存在异常。

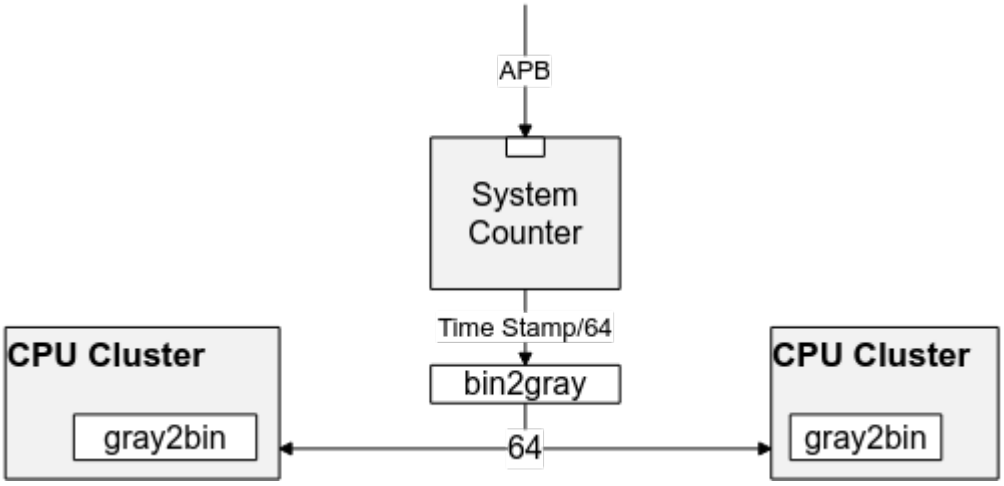
----结束

13.4.15 SYSTEM COUNTER 同步

System Counter为CPU Core提供time stamp值，软件可用于记录某个事件的持续时间，调度事件并基于一个增量的计数值触发中断等。

Hi309a有一个System Counter， Hi309a内所有CPU Core共用这个Counter。System Counter提供一组APB接口，基地址请参见地址映射表，内部寄存器信息请参考System Counter寄存器描述，时钟默认为晶振时钟，可配置为晶振时钟二分频。

图 13-3 System Counter 结构框图



System Counter在芯片解复位后自启动。

13.4.16 WFE

Hi309a内的Core Wait for Event采用Event Ctrl网络传输，软件无需进行任何配置。

例如，当一个Core执行SEV指令，Hi309a内的其他Core都能收到Event消息，处于WFE状态的Core因此退出WFE low-power state。

13.4.17 功放保护及掉电告警

功放保护输入支持可配置电平或边沿触发，电平及边沿状态可配置。

功放保护汇聚输出信号支持汇聚所有功放保护源，并支持对任一输入源独立屏蔽，芯片需要支持输入和输出信号的多对多映射逻辑。

功放保护及掉电告警信号如下表13-2所示。

版权所有 © 海思技术有限公司

表 13-2 信号名称说明

序号	信号名称	信号说明
1	PWR_ALERT	二次电源异常告警信号，电平有效，低电平表示有告警。支持作为功放保护输入源，默认不使能功放保护。需要支持可配0~15个晶振周期的毛刺滤波，默认为0（不过滤毛刺）。**高低电平均需要滤毛刺。** **注意：此信号同时需要通过中断上送CPU处理。**
2	PA_OFF_PWR	板级电源监控输入，掉电触发功放保护**及NAND写保护**，同步作为中断上报软件，电平有效，高电平表示有功放保护需求。需要最高支持100ns毛刺滤波，滤波精度要求≤5ns，默认不过滤毛刺。
3	PA_OFF_IN	主从片间功放保护交互，接收另一片送过来的功放保护信号，同步作为中断上报软件，电平有效，高电平表示有功放保护需求。检测到有效信号后内部记录为片间功放汇聚输入，支持进一步汇聚输出。需要支持10ns毛刺滤波，支持滤波功能开启或关闭，默认不过滤毛刺。
4	PA_OFF_OUT	主从片间功放保护交互，可汇聚自身收到的各功放保护源，输出有效方式（上升沿、下降沿、高电平、低电平）可配置，默认输出高电平表示有功放保护需求。
5	PA_OFF[0:1]_BBL	接收BBL送过来的功放保护信号，高电平表示有功放保护需求，在识别到功放保护后，除了对功放保护进行汇聚输出，还需要支持作为中断上报软件；需要支持10ns毛刺滤波，支持滤波功能开启或关闭，默认不过滤毛刺。
6	INT_BBL[0:1]_WDG	BBL软件异常中断上报，高电平表示有中断请求，支持作为功放保护源。需要支持10ns毛刺滤波，支持滤波功能开启或关闭，默认不过滤毛刺。
7	PA_OFF[0:1]_SYS	功放保护汇聚输出，可汇聚自身收到的各中断源，输出有效方式（上升沿、下降沿、高电平、低电平）可配置，默认高电平表征有功放保护请求。最大支持1驱3。

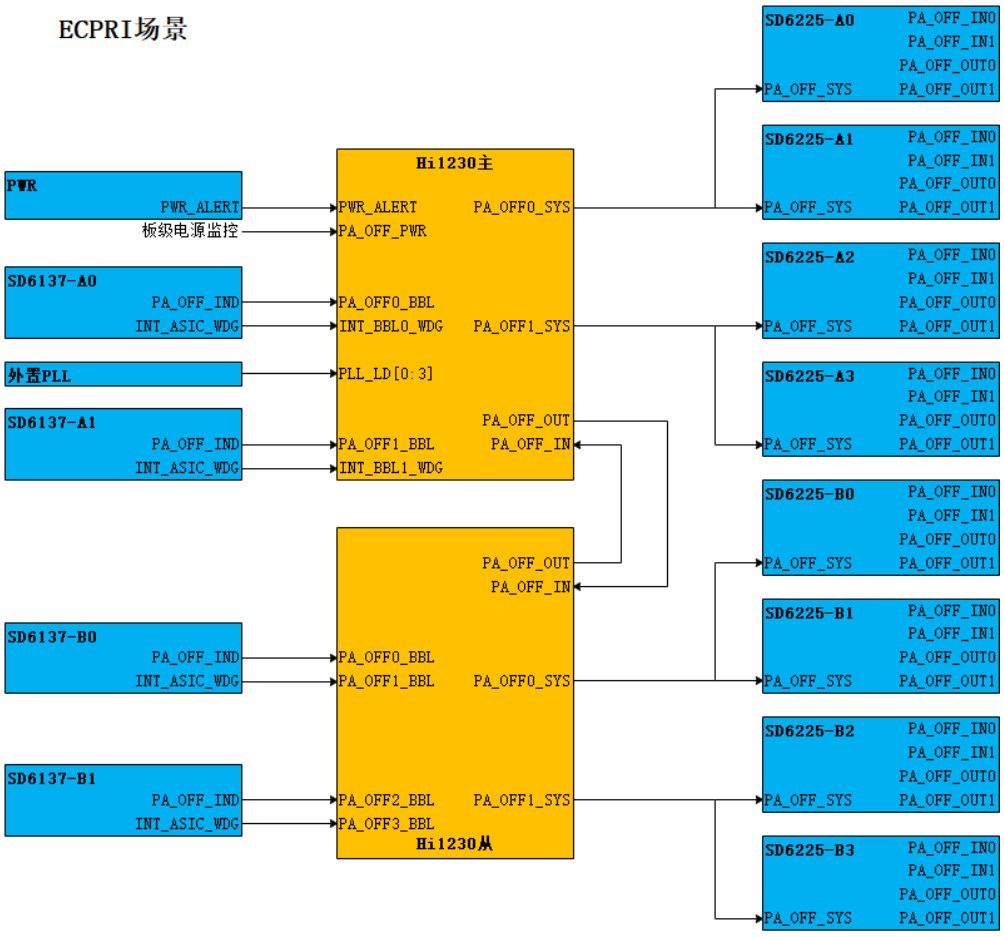
版权所有 © 海思技术有限公司

序号	信号名称	信号说明
8	PLL_LD[0:5]	外置PLL锁定指示信号，电平有效，低电平表示失锁。支持作为功放保护输入源，默认不使能功放保护。需要支持10ns毛刺滤波，支持滤波功能开启或关闭，默认不过滤毛刺。 **注意：此信号同时需要通过中断上送CPU处理。**

功放保护在ECPRI与CPRI场景下设计有轻微区别，主要表现在CPRI场景下少了来自于BBL的功放保护源，以及主从Hi309a片间的功放保护，且对外输出的PA_OFF1_SYS最大化场景需要1驱3。

ECPRI场景功放保护框图如下图13-4所示。

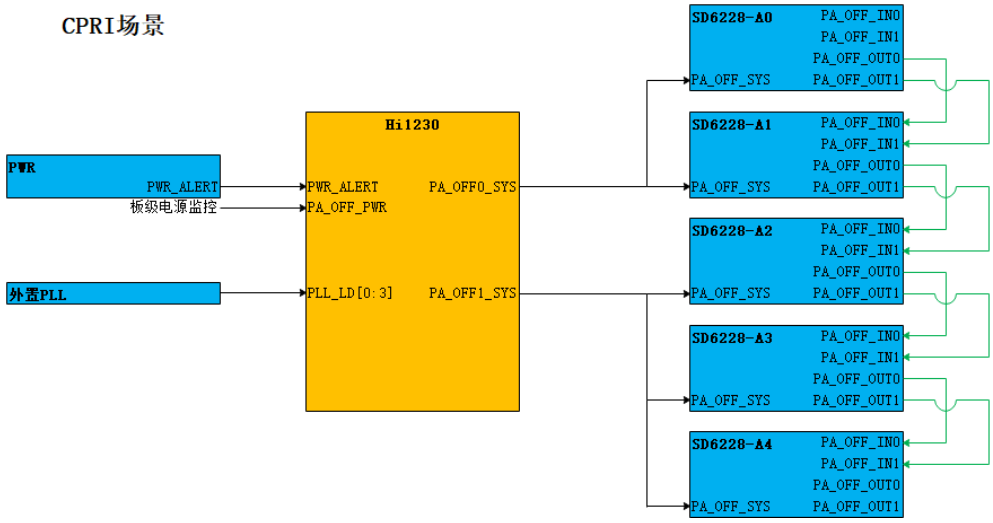
图 13-4 ECPRI 场景功放保护框图



CPRI场景功放保护框图如下图13-5所示。

版权所有 © 海思技术有限公司

图 13-5 CPRI 场景功放保护框图



芯片内部需要实现如下功能：

1. 功放保护管脚全部工作在晶振时钟域（含晶振PLL输出时钟）。
2. 功放保护输入支持可配置电平或边沿触发，电平及边沿状态可配置。
3. 功放保护汇聚输出信号支持汇聚自身所有功放保护输入源，并支持对任一输入源独立屏蔽，输出有效方式（上升沿、下降沿、高电平、低电平）可配置。

   输出为电平时候：上升沿：高脉冲，高电平持续100ns，其他时候都是低电平（低电平>100ns）；

   下降沿：地脉冲，低电平持续100ns，其他时候都是高电平（高电平>100ns）
4. 功放保护输入支持当前及历史状态查询（历史状态需要支持统计计数16bit，计满翻转。

   a. 普通中断，有1bit记录即可。

   b. 告警信号、功放保护信号按照16bit记录。

   c. 同时是中断、告警或功放保护的信号，按照高的要求设计。
5. 输入管脚需要支持毛刺滤波，具体滤波需求参考以上信号说明中表格。
6. Hi309a从检测到正确功放保护输入到输出功放保护信号的延时在增加滤波前为9.92ns~29.05ns之间。

13.4.18 引脚滤波功能

1. I2C、PMBUS接口要求：支持毛刺滤波，精度5ns，支持0~1275ns可配置，默认配置50ns滤波。
2. GPIO管脚作为中断时，要求支持10ns毛刺滤波。
3. 时钟选源切换信号CLK_SEL需求支持10ns毛刺滤波（HACSPI）。

13.4.19 添加软件可控的复位记录

添加16个可读写的寄存器，寄存器只能通过上电复位清除。

版权所有 © 海思技术有限公司

13.4.20 NAND 掉电保护

芯片增加电源异常判定、写保护判定逻辑，配合产品板级电源掉电检测，实现硬件写保护功能。

图 13-6 NAND 掉电保护

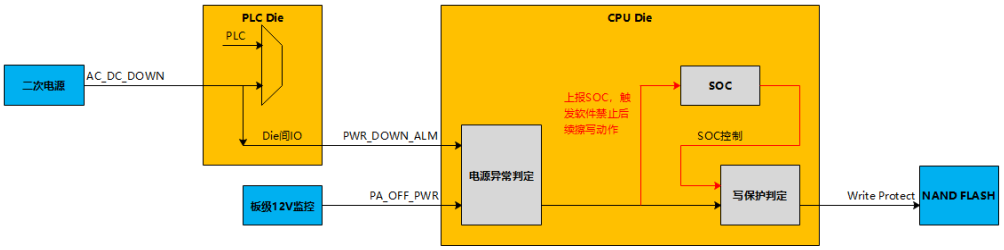


表 13-3 电源异常判定真值表

电源异常判定真值表		
PWR_DOWN_ALM	PA_OFF_PWR	判定结果
有掉电	X	有掉电
X	有掉电	有掉电
无掉电	无掉电	无掉电

表 13-4 写保护判定真值表

写保护判定真值表		
电源异常判定	SOC控制	WP输出
有掉电	X	写保护
X	写保护	写保护
无掉电	不写保护	不写保护

1. 增加PWR_DOWN_ALM信号（复用Die间IO，支持毛刺滤波，不新增管脚）。
2. 增加电源异常判定逻辑，检测到电源异常时传递到写保护判定逻辑，同步上报软件（通过中断上报，支持中断状态可查）；电源异常判定输入源支持软件独立屏蔽，默认使能。
3. 增加写保护判定逻辑，出现掉电时，硬件直接对NAND进行写保护；写保护判定输入源支持软件独立屏蔽，电源异常分支默认不使能，SOC控制分支默认使能。
4. 电源异常判定触发写保护的设计，执行逻辑关（写保护），软件开（去写保护），掉电不保存。

版权所有 © 海思技术有限公司

13.4.21 通用外部中断

SI文档中和中断映射表中INT0~INT9为通用外部中断，该中断默认状态为1，低电平触发中断。

13.5 工作流程

13.5.1 初始化

SysCtrl不需要特殊的初始化步骤，但整个系统的初始化（如时钟配置、模块初始化等）需要访问SysCtrl模块的寄存器。

13.5.2 复位

SysCtrl不支持模块独立复位。

13.5.3 功能模式 MBIST 操作

关于MBIST的使用请参考“**功能MBIST**”。

13.5.4 DJTAG

支持通过DJTAG配置或读取SysCtrl/SubCtrl内部的寄存器。

版权所有 © 海思技术有限公司

14 LLC 描述

14.1 概述

LLC是系统级片上三级高速缓存，该Cache被CPU和各加速器、IO共用，用以临时缓存在一段时间内会被多次使用的数据，这可以有效提高处理器及其他加速模块对外部存储器的访问带宽和访问效率。LLC也是片内MOESI一致性协议Skyros的Home Node，维护各个Cluster的L2 Cache之间的Cache一致性。LLC同时也是片间MESI一致性协议HYDRA的Cache Agent，处理HYDRA的一致性请求，完成HYDRA域的一致性维护

14.2 关键术语

表14-1为与LLC相关的关键术语表。

表 14-1 llc 术语表

术语	描述
CHI/Skyros	Coherent Hub Interface ARM的一种总线协议。
RR	Round Robin。
LLC	Last Level Cache最后一级缓存。
RA	Read Allocate，指示读操作如果缺失，则需要产生Allocate操作。
WA	Write Allocate，指示写操作如果缺失，则需要产生Allocate操作。
SECDED	单bit纠错，双bit检错。
DECTED	双bit纠错，三bit检错。
PR	Pseudo Random Algorithm，随机算法。
LRU	Least Recently Used Algorithm，最近最少使用算法。

版权所有 © 海思技术有限公司

14.3 特性

LLC模块具有以下特点：

- Cache容量为3MB，采用2 Banks Distributed结构，Cache运行频率和Ring Bus相同。
  - 每Cycle最大可并行处理2个请求查询。
  - 每Cycle最大可并行读/写2个256 bits的数据。
  - Ring Bus按照地址全地址交织或CacheLine交织分配Banks间地址交织方式，保证2个Bank带宽均衡。
- Cache为Skyros Home Node。
  - 在CHI协议中充当HNF的角色。
  - 支持Skyros MOESI Cache一致性协议，完成CPU Cluster 的L2 Cache之间，及L2 Cache和系统所有IO、Accelerator主设备之间的数据一致性。
  - 在Skyros一致性域承担PoS（Point of Serialization）和PoC（Point of Coherence）角色。
  - 支持Shareable属性的数据读写访问一致性。
  - 支持处理一致性协议中的Maintenance操作透传处理（即对于上一级的maintain操作会将数据彻底刷到DDR中）。
  - 为提升系统容错能力，支持将Snoop操作的Invalid操作转换为Clean+Invalid操作（寄存器可配），避免Dirty数据丢失导致一致性错误。
- 提供Skyros Directory，Shareable操作可在Directory查询CPU Cluster L2 Cache中是否存有某Cache Line的信息，避免Snoop Broadcast。
  - Directory为Sparse结构，容量为L2 Cache Tag的9/2倍。
  - 支持Direcotry溢出后进行Back-Invalid。
  - Directory支持MSI三种状态。
  - Direcotry支持CPU Cluster数量为1~8，最大支持8 CPU Cluster。
  - Direcotry支持全局Invalid/Clean，即支持将全部L2、LLC数据刷出，然后全芯片复位或掉电。
  - Direcotry支持按cluster Maintain操作,即支持按cluster将L2、LLC数据刷出。
  - Direcotry支持按way Maintain操作，即支持按way将L2、LLC数据刷出。
  - 支持对溢出CacheLine的Victim，精确执行，不丢弃Evict。
- Cache判定（包括Read/Write Allocate）可灵活编程。
  - 可自动判定CPU的MemSet操作，对连续冲击数据进行递减比例Allocate。
  - 可自动判定Packet连续冲击，对连续冲击数据进行递减比例Allocate。
- Cache Line存储大小固定为128 Bytes（32 Words/1024 bits），支持CPU 64Byte的CacheLine访问，支持与HHA 64 Bytes和128Bytes的交互。
- Cache TAG/Directory Block为18路组相连结构，Data Block为12路组相连结构。
  - TAG和Data之间采用动态链表结构（即TAG和DATA之间一一映射）。
  - 对Skyros协议的M和E状态采用尽量Exclusive结构，即只记录TAG/Directory，不存放Data。
  - 对Skyros协议的S状态采用Inclusive结构，即同时记录TAG/Directory和Data。

版权所有 © 海思技术有限公司

  - 支持将部分 Way进行关闭以节省功耗（即way shutdown功能）。
- 基于物理地址的36 bits地址结构。
- 支持随机替换算法，伪LFU，DSID QoS三种替换算法且寄存器可配。
- 支持Cache Line Lockdown和unlock操作。
  - 支持按地址区段的Lockdown。
  - 支持按地址区段进行Unlock。
- 支持TrustZone。
  - 在每条Cache Line中有一个bit保存Secure信息，安全操作无法查询到非安全操作的信息，非安全操作也无法查询到安全操作的信息。
  - 对于寄存器配置涉及secure信息的，如地址段maintain和Lock操作，所操作的地址段的安全信息会通过寄存器接口的安全信息位传递过来并使用；传递的信息是安全的则对安全地址进行操作，传递的信息是非安全的则对非安全地址进行操作。
- 固定为Write-Back策略。
- 支持基于物理地址的Cache 维护（Maintain）操作。
  - 包括Clean、Invalid、Clean+Invalid三种操作。
  - 支持全系统（包括L2和L1）进行Invalid、Clean、Invalid+Clean。
  - 支持按地址区段模式进行Invalid、Clean、Invalid+Clean。
  - 支持按cluster 模式进行Invalid、Clean、Invalid+Clean，即支持按cluster将L2、LLC数据刷出。
  - 支持按way模式进行Invalid、Clean、Invalid+Clean，即支持按way将L2、LLC数据刷出。
- 支持预取功能,即支持当出现连续地址规律且预取使能的情况下会将CPU可能需要的CacheLine提前放入Cache中。
- 支持上电自动初始化TAG/Directory。
- 支持在没有访问时自动关闭内部Bank的SRAM及逻辑的工作时钟。
- 支持数据Bypass功能（即在配置寄存器中的控制器，使部分满足条件的请求的数据直接由DDR返回，而不经过LLC）。

  可以将满足MISS，数据状态为非S态条件的ReadShared，ReadUnqiue由DDR直接返回RN数据而不经过LLC。
- 支持CacheLine压缩技术，即对于有规律的数据可以根据Tag信息直接回数据而无需访问Datram。
- 支持Evicted-Address Filter（EAF）。
- 支持CacheStash功能，（即支持L2 Cache预取信息传递），具体行为为接收POE特殊Opcode并往L2中发出snoop以告知L2预取的该地址信息。
- 支持虚拟化，可以配置不同的Dsid所使用不同的Way，在allocate或替换时会优先选择所配置的Way（即支持12个DSID QoS，每个Way独立配置）。
- 支持低峰值功耗模式，即LLC内部Bank访问Memory错开访问。
- 支持LLC作为上电启动空间使用，最大可用空间为3M。
- 支持Mbist功能。

  支持在LLC关闭的情况下通过软件启动Mbist测试。
- 每个Bank提供8个32bits Counter，可对Cache读写操作数量、读写命中数量等性能信息进行统计，支持对某个地址段进行DFX统计，支持对某个cluster或加速器进行DFX统计。

版权所有 © 海思技术有限公司

- 片内CPU可通过DJTAG BUS访问LLC的寄存器接口、片外JTAG接口可通过DJTAG BUS访问LLC的寄存器接口。
- Cache内TAG SRAM支持单bit，两bitECC错误信息可统计和两bit ECC上报中断；DATA SRAM支持单bit ECC纠错可配，并上报中断，两bitECC错误信息可统计，并上报中断。
  - 对于TAGSRAM单bit ECC出错，支持将纠错后的数据回写至Cache；对于DATA SRAM单bit出错，纠错可配，并支持回写可配，并上报中断。
  - 对于TAGSRAM两bits ECC出错，支持上报中断，总线返回错误数据，不返回总线Error；对于DATASRAM两bits ECC出错，支持上报中断，总线返回错误数据，不返回总线Error。
  - 记录Tag部分出错单/双bit ecc 出错的Way，Index等信息和Data部分出错单/双bit ecc出错的Way，Index等信息。

14.4 遵循协议

CHI协议。

Hydra 协议。

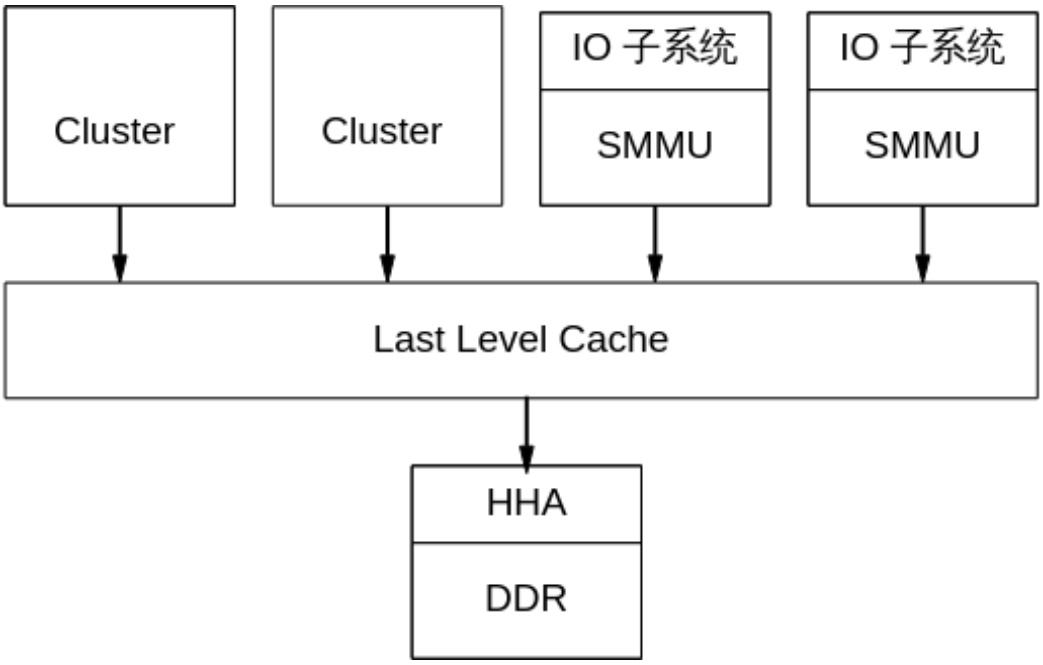
14.5 功能描述

无。

14.5.1 应用框图

LLC的应用如图14-1所示。

图 14-1 LLC 应用图



版权所有 © 海思技术有限公司

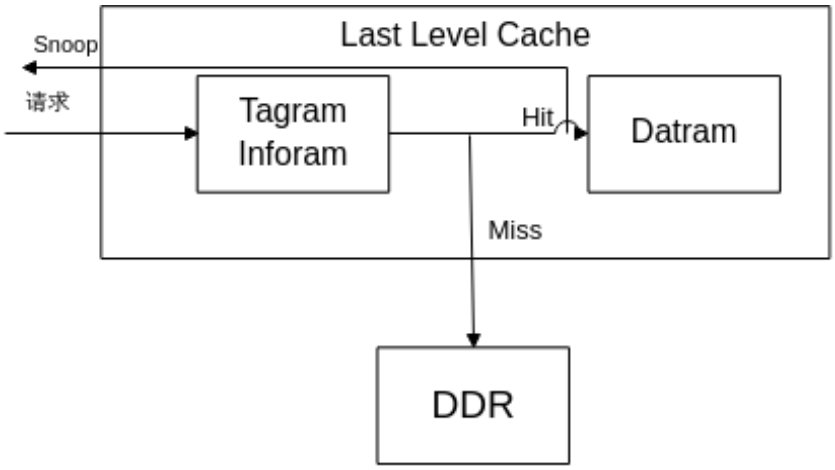
各CPU Cluster、SMMU在读写DDR cacheable空间时，会发出请求先通过Ring Bus到达LLC，然后进行LLC查询，如果LLC命中，则读写在LLC中完成；如果读写缺失，则LLC通过Skyros接口经过Cross Station读取DDR的数据。基于数据的时间和空间局部性特性，即CPU、SMMU会在读写某个数据后有较高概率在较短时间内再次读写该数据的特性，LLC主要存放近期内被CPU、SMMU读写过的数据，从而较快的响应读写，以达到提升这些数据被再次访问时的读写效率的目的。

LLC的内部流水线与Ring以同步方式工作，工作频率与Ring相同。LLC的外部反压条件均属协议反压。

14.5.2 模块内部结构

LLC的内部结构示意图如图14-2所示。

图 14-2 LLC 内部结构示意图



读请求进入LLC后，会先查询tagram以判断地址是否hit，查询inforam以判断状态是否正确。如果地址hit且状态显示LLC数据为最新数据,则可以直接读datram；如果地址hit且状态显示LLC数据不是最新数据,则发出snoop往L2 cache中获取数据；如果地址是miss且状态为无效，则发出请求往DDR中获取数据。

写请求进入LLC后，会先查询tagram以判断地址是否hit，查询inforam以判断状态是否正确。如果地址hit且状态显示该CPU可以改写数据，则可以直接写datram；如果地址hit且状态显示该CPU没有改写权利，则发出snoop 使该CPU获得写权利后再改写数据。如果地址是miss且状态为无效，则需要判断请求是否需要allocate，如果需要allocate，则往DDR发出读操作，获取数据后合并写入datram中；如果不需要allocate，则直接写到DDR中。

14.5.3 多 Bank 交织

LLC为Distributed Cache结构，按照不同的地址交织方式，将物理空间均匀分布在2个Bank之间。各Bank互相独立，各自处理本Bank的传输。

Ring Bus支持两种交织方式：cacheline交织和全地址交织。

Cacheline交织是通过全地址的第7bit进行交织，将所有地址均匀的分配到2个Bank。

全地址交织是通过全地址的异或操作生成1bit信息进行交织。

版权所有 © 海思技术有限公司

14.5.4 Cache 行为判定

LLC通过对当前操作的操作类型，Allocate属性和是否需要BackInvlid来进行判断是否需要Allocate。

对于涉及一致性的部分操作，包括ReadShared，ReadUnique，ReadClean，CleanUnqiue，MakeUnique，强制会Allocate。

对于写操作，和其他部分操作，包括ReadOnce，ReadNosnp，WriteUnqiueFull，WriteUnqiuePtl，WriteNosnpFull，WriteNosnpPtl，会根据Allocate属性和该操作是否需要Backinvlid来决定是否需要Allocate；如果是Allocate且不需要Backinvlid，则Allocate；否则不Allocate。

对于其他操作，包括CMO操作，CopyBack操作等，则不区分Allocate属性。

14.5.5 替换算法

当一个命令Miss且需要做Allocate时，如果所有的Way都被占用，需要按照相应的替换算法选择一个Way作为替换Way。此时，需要先将替换Way中的Cacheline从Data Ram中读出写回到DDR（如果是dirty数据才需要，否则可以发出evict），再把新的Cacheline从DDR中读到并写进Data Ram。为了保证操作的顺序性和效率，将被替换的Cacheline从Data Ram中读出后（不需要保证将Cacheline写到DDR），才能执行读取新的Cacheline的操作。

LLC支持的替换算法为随机（Pseudo Random）算法，伪LFU，DSID QoS三种替换算法。

- 随机算法是轮询调度算法来实现随机。
- 伪LFU算法即近似最不经常使用页置换算法。具体做法是，对于每一个Allocate的操作，会根据请求的QoS记录其优先级，每访问一次则优先级增高，每次替换优先级最低的那条way。
- DSIDQoS算法。具体的做法是，通过寄存器配置每一个Dsid相应使用的Way，当该Dsid的操作到来时会优先在寄存器配置的Way中使用伪LFU选择优先级最低的那一条Way进行替换。至于没有属于单独配置的Dsid的请求，则使用公共配置Way寄存器中的Way（即有12个Dsid可以单独配置，其余的统一使用一种配置）。

14.5.6 Lockdown/Unlock

LLC可以通过Lockdown/Unlock操作来对指定的地址区段进行Lockdown/Unlock。地址区段必须以Cache Line对齐，地址段区间从1个cacheline到整体地址段空间（Cacheline粒度为128Byte）。

Lockdown:

用寄存器配置指定Lockdown的地址区段内的地址和所需Lockdown的Way，那么该地址段会LLC 发出出操作，主动将该地址段的Cacheline Allocate进所配置的Way。一旦被Allocate进Last Level Cache，就不会被替换出去。任意地址进行读写操作时，需要Lockdown的地址区段可以在LLC打开时动态切换，但在切换前需要将原来的地址段进行Unlock。

版权所有 © 海思技术有限公司

须知

如果Lockdown的地址对应的允许用的Way中没有可用的Way时（均被Lock不可替换），该Lockdown操作默认失败，所以Lockdown一段地址，只是尽力lockdown进入LLC，不可以依赖于一定会Lock进LLC中。

Unlock:

LLC的Unlock配置和Lockdown类似，不同之处在于不用配置Way信息；Unlock操作将标志为Lock的Cache Line进行解锁操作，被Unlock的Cache Line可被替换；

配置寄存器的操作分为安全和非安全两种，由此产生了安全Lock、安全Unlock操作和非安全Lock、非安全Unlock操作，以配置LLC_LOCK_CTRL*时的安全属性为准。安全的配置操作只能对安全的地址进行Lock，Unlock操作，非安全的配置操作只能对非安全的地址进行Lock，Unlock操作。

为保证Lock操作的原子性，LLC支持同时对8组地址段进行Lock和Unlock操作。8组Lock/Unlock寄存器各自独立，配置流程完全相同，软件配置流程中的是举其中一个配置例子，其他的相同。

14.5.7 Maintain 操作

LLC的Maintain操作包括Clean、Invalid、Clean+Invalid三种（具体可以通过LLC_MAINT_CTRL中的maint_range进行配置）：

- Clean操作将Cache的Dirty Cache Line更新到主存中，使主存中存储最新的数据，随后将Cache中该Line的Dirty指示置零但Valid保持有效。
- Invalidate操作将Cache的数据Line直接丢弃，随后将Cache中该Line的Dirty和Valid指示都置零。
- Clean+Invalid操作将Cache的Dirty Cache Line更新到主存中，使主存中存储最新的数据，随后将Cache中该Line的Dirty和Valid指示都置零。

LLC支持全局 Maintain操作、按地址区段Maintain操作、按Cluster Maintain操作和按Way Maintain操作（具体可以通过LLC_MAINT_CTRL中的maint_type进行配置）：

- 地址段Maintain操作：

  按地址段Maintain操作指对特定物理地址进行Clean、Invalidate、Clean+Invalidate操作，通过写LLC的Maintain寄存器配置对应的地址段和Maintain使能，即可将该地址段在LLC内部存的Cacheline进行Maintain操作（地址段区间从1个Cacheline到整体地址空间）（Cacheline粒度为64Byte）。
- 全局Maintain操作：

  全局 Maintain操作指对LLC所有Bank里的所有Line进行Clean、Invalidate、Clean+Invalidate操作，通过配置LLC的Maintain寄存器配置Maintain使能，即可将LLC内部存的所以Cacheline进行Maintain操作。
- Cluster Maintain操作：

  Cluster Maintain操作指对特定的Cluster进行Clean、Invalidate、Clean+Invalidate操作，通过写LLC的Maintain 寄存器配置对应的Cluster和Maintain使能，即可将该Cluster在LLC内部存的Cacheline进行Maintain操作。
- Way Maintain操作：

  Way Maintain 操作指对特定的Way进行Clean、Invalidate、Clean+Invalidate操作，通过写LLC的Maintain 寄存器配置对应的Way和Maintain 使能，即可将该Way存的Cacheline进行Maintain操作。

版权所有 © 海思技术有限公司

配置寄存器的操作分为安全和非安全两种，由此产生了安全Maintain和非安全Maintain操作，以配置LLC_MAINT_CTRL*时的安全属性为准。安全的配置操作只能对安全的地址进行Maintain操作，非安全的配置操作只能对非安全的地址进行Maintain操作。

对于全局Maintain、Cluster Maintain及Way Maintain，为了保证Maintain效率，所以在安全Maintain的配置下，可以一次性将LLC内部安全和非安全的满足条件的Cacheline均刷到DDR中。而非安全Maintain的配置下，只能刷掉LLC内部非安全的满足条件的Cacheline。此时为了防止安全的Invalid操作将非安全的地址Invalid掉，所以在全局Maintain下，会将Invalid操作强制改为Cleaninvalid操作。

须知

maintain操作是对于LLC,L1,L2全局的操作,即会将 LLC,L1,L2中对应的信息都刷到下一级（不包含nosnp的读写操作）。

14.5.8 预取操作

LLC的预取操作主要用于解决在memory copy场景下，读操作发生连续Miss而造成性能下降的问题。

预取功能通过写LLC的Auxiliary Control寄存器：LLC_AUCTRL来开启。在使能预取功能前，需要配置好LLC_PREFETCH寄存器。LLC_PREFETCH寄存器用于配置LLC在检测到多少个连续地址读Miss后自动进行预取操作，预取成功多少个地址hit后继续预取，预取失败多少个地址miss后重新寻找规律，预取跳过多少个地址开始预取，预取规律消失多久后预取Buffer被释放，以及预取多少条Cache Line。

预取分两种模式：1.基于dsid进行预取。2.基于LPID+SRCID进行预取。两种模式各自独立，只能配其中的一种模式工作，不允许动态切换。

须知

由于系统在页表的最小粒度是4kByte，因此预取操作只在4kByte的范围内进行，不会跨越4kByte边界。另外，预取操作只针对非安全的操作进行，对于安全操作，不会触发预取操作。

预取建议2个bank的LLC一起配置，尽量保持一致，保证预取平衡。

14.5.9 性能跳水

LLC的性能跳水操作主要用于解决在Memory Copy场景下，读/写操作发送连续Miss而造成性能下降的问题（即CPU在处理业务的过程中，如果加速器或其他CPU在对连续地址进行大量的写/读操作,如果这些操作都是Allocate的，则会使LLC内部出现大量的替换，从而影响CPU对业务的处理，所以需要将这些连续地址的大量冲击变为NonAllocate（即性能跳水），从而减少替换，减少对CPU处理业务的影响）。

性能跳水功能通过写LLC的Auxiliary Control寄存器：LLC_AUCTRL来开启。在使能性能跳水前，需要配置好LLC_ALLOC_FREQ寄存器。LLC_ALLOC_FREQ寄存器用于配置LLC在检测到多少个连续地址写Miss后自动进行性能跳水操作，性能跳水规律消失多久后性能跳水的Buffer被释放。

版权所有 © 海思技术有限公司

性能跳水主要针的操作有：

- WriteUnqiue
- WriteNosnp
- ReadNosnp
- ReadOnce

性能跳水分两种模式：

- 基于dsid进行性能跳水。
- 基于LPID+SRCID进行性能跳水。

两种模式各自独立，只能配其中的一种模式工作，不允许动态切换。

须知

性能跳水只针对非安全的操作进行，对于安全操作，不会触发性能跳水。

14.5.10 RAM ECC 校验

LLC支持所有RAM的ECC校验，该功能提供高速Memory访问检错纠错能力。对Data RAM，ECC纠错基于每条Cacheline的每128 bits进行；对Tag RAM，基于每个Tag进行ECC纠错。

对于Tag Sram，如果有1bit校验错误，则ECC逻辑将自动纠正错误，并可配置将纠正之后的数据回写进Tag RAM，并通过Interrupt信号通知系统（如不需要上报中断，可配置对应的中断屏蔽）；如果错误为1bit以上，ECC逻辑将无法纠正，此时LLC通过Interrupt信号通知系统（如不需要上报中断，可配置对应的中断屏蔽）。

对于Data Sram，如果有1bit校验错误，则可配置是否纠正错误，并可配置是否将纠正之后的数据回写进DATA RAM，并通过Interrupt信号通知系统（如不需要上报中断，可配置对应的中断屏蔽）；如果错误为1bit以上，ECC逻辑将无法纠正，此时LLC通过Interrupt信号通知系统（如不需要上报中断，可配置对应的中断屏蔽）。

14.5.11 虚拟化

为了减少不同制式的业务在LLC中的相互影响，需要实现每个制式在LLC中的数据之间的虚拟化。对于每个虚拟机，可以区分的粒度是Dsid。所以通过Dsid信息进行虚拟化（LLC所得到的Dsid是从请求带过来的）。

虚拟化功能通过写LLC的Auxiliary Control寄存器：LLC_AUCTRL来开启。在使能虚拟化前，需要配置好LLC_DSID_PART*寄存器和LLC_DSID_WAY_MASK_PART*。

LLC_DSID_PART*代表所配置需要虚拟化的DSID，一共可以配置12个DSID。LLC_DSID_WAY_MASK_PART*代表配置的DSID所对应优先使用的Way。每个DSID可以配置使用的Way有12条。

不在LLC_DSID_PART*中配置的Dsid的请求使用LLC_DSID_WAY_PART8所配置的way。

可配置的way的个数最少是1条way，最多是12条way。

版权所有 © 海思技术有限公司

须知

如果某dsid的请求到来时，它所配置可使用的way均已被使用且不可替换时，可以使用其他的way。虚拟化所配置的way是优先使用。

LLC的寄存器建议用公共平台或Hypervisor配置，不建议虚拟系统直接配置。

14.5.12 数据 Bypass 功能

可以将满足MISS，数据状态为非S态条件的ReadShared，ReadUnqiue Bypass，由DDR直接返回RN数据而不经过LLC。

14.5.13 Cacheline 压缩技术

LLC将会对于一些比较简单规律的数据进行压缩，将该规律信息存储在Tagram中，则下次访问LLC时，可以直接通过Tagram的信息直接生成该数据，而无需到Datram中寻找数据。

表 14-2 LLC Cacheline 压缩的 Pattern

Num	Pattern	Description	Compress Num
0	{XXX，XXX}	无规律	00
1	{{XXX},{0b0}}	低256bit数据为全0	01
2	{{0b0},{XXX}}	高256bit数据为全0	10
3	{0b0}	512bit数据为全0	11

注：表格中XXX是表示非0b0的任意数据。

14.5.14 Evicted-Address Filter（EAF）

LLC每次替换时都会在EAF中记录替换地址的信息（通过哈希函数映射），如果LLC在短时间内（每隔一段时间，LLC会将EAF重新刷新，在刷新周期内即算短时间）再次访问发现Miss且在EAF中命中，则说明该CacheLine被踢掉是不准确的，所以该CacheLine将会以较高优先级放入LLC中，如果该CacheLine是Miss且在EAF中不命中，则说明该CacheLine最近没有访问，所以该CacheLine将会以较低优先级放入LLC中。

哈希函数有两种：

1. 根据Tag地址的低6bit直接生成64bit 的Onehot信号。
2. 根据Tag地址的低18bit（DDR的地址空间有限，20bit以上不会变所以不考虑），每隔6bit进行异或算法（即映射后bit0=映射前bit0^bit6^bit12），以此类推，生成一个6bit的数，在将这个6bit的数变成64 bit 的Onehot即可。

14.5.15 Cache Stash

LLC会收到POE发出的特殊Opcode，该请求中将携带地址和Cpuid信息。对于shareable的cachestash，LLC将会对该Cpuid发出该地址特殊的snoop，以通知CPU预取该地址的信息；对于nonshareable的cachestash，LLC不会发出snoop。

版权所有 © 海思技术有限公司

14.5.16 低峰值功耗模式

LLC支持寄存器配置选择低峰值功耗模式。打开该模式，则LLC内部Bank访问Datram将会强制错开，即一个周期内只有一个Bank的Datram将被访问。关闭该模式，则没有限制。（该模式下会影响LLC的性能，具体需要看场景）。

14.5.17 统计功能

LLC提供Monitor统计功能，以统计Last Level Cache相关性能数据。

LLC的每个BANK提供8个32bit位宽的统计寄存器，8个统计寄存器均可编程为任一统计事件。计数器为加法计数器，在系统写入计数值（默认为全零），并使能Monitor后计数，当计数到全1时将重新从全0开始计数，并产生一个中断通知CPU。

LLC Monitor按照Bank对事件进行统计，每Bank内部统计的事件共65个（以下表14-3格中的NUM不代表统计事件配置值，具体配置值，参考nManager）。

表 14-3 LLC 内部统计事件功能列表

Num	LLC Event	Description	备注
0	event_alloc_read	读Allocate请求个数。	-
1	event_alloc_write	写Allocate请求个数。	-
2	event_noalloc_read	读noAllocate请求个数。	-
3	event_noalloc_write	写noAllocate请求个数。	-
4	event_read_hit	读命中个数。	-
5	event_write_hit	写命中个数。	-
6	event_maintain	收到cmo请求个数。	-
7	event_copyback	收到copyback请求个数。	-
8	event_hydra_snoop	收到hydra snoop请求个数。	-
9	event_smmu_req	收到smmu请求个数。	-
10	event_excl_success	收到exclusive成功操作个数。	-
11	event_excl_fail	收到exclusive失败操作个数。	-
12	event_overflow	Cacheline drop溢出个数。	-
13	event_backinvld	Cacheline backinvld溢出个数。	-
14	event_prefetch	预取操作个数。	-
15	event_retry	retry操作个数。	-
16	event_tag _2bit_ecc	Tag ram 2bit及2bit以上ecc 错误个数。	-
17	event_data_2bit_ecc	Data ram 2bit及2bit以上ecc 错误个数。	-

版权所有 © 海思技术有限公司

Num	LLC Event	Description	备注
18	event_dirovf	HHA目录溢出操作个数。	-
19	event_dirovf_retry	HHA目录溢出retry个数。	-
20	event_tag _1bit_ecc	Tag ram 1bit ecc错误个数。	在大量，连续出错时统计不准。
21	event_data _1bit_ecc	Data ram 1bit ecc错误个数。	在大量，连续出错时统计不准。
22	event_snoop_cpu0	往cluster0发出的snoop个数。	-
23	event_snoop_cpu1	往cluster1发出的snoop个数。	-
24	event_snoop_cpu2	往cluster2发出的snoop个数。	-
25	event_snoop_cpu3	往cluster3发出的snoop个数。	-
26	event_snoop_cpu4	往cluster4发出的snoop个数。	-
27	event_snoop_cpu5	往cluster5发出的snoop个数。	-
28	event_snoop_cpu6	往cluster6发出的snoop个数。	-
29	event_snoop_cpu7	往cluster7发出的snoop个数。	-
30	event_dsid0_capability	LLC内部DSID0所占容量。	-
31	event_dsid1_capability	LLC内部DSID1所占容量。	-
32	event_dsid2_capability	LLC内部DSID2所占容量。	-
33	event_dsid3_capability	LLC内部DSID3所占容量。	-
34	event_dsid4_capability	LLC内部DSID4所占容量。	-
35	event_dsid5_capability	LLC内部DSID5所占容量。	-
36	event_dsid6_capability	LLC内部DSID6所占容量。	-
37	event_dsid7_capability	LLC内部除了dsid0~6其他DSID所占容量。	-
38	event_rs_rc	收到readshared/readclean个数。	-
39	event_allocate_std _sub	相邻64B需要allocate到std_way的请求个数。	-
40	event_allocate _ext	allocate到ext_way的请求个数。	-
41	event_ru_mu_cu	收到readunique/makeunique/cleanunique个数。	-
42	event_allocate_ext_sd	rs allocate到ext_way个数。	-

版权所有 © 海思技术有限公司

Num	LLC Event	Description	备注
43	event_allocate _std	allocate到std_way的请求个数。	-
44	event_wu	收到writeuniquefull/writeuniqueptl个数。	-
45	event_ext2std	从ext_way转移到std_way的次数。	-
46	event_wns	收到writenosnpfull/writenosnpptl个数。	-
47	event_ro	收到readonce个数。	-
48	event_std2ext	从std_way转移到ext_way的次数。	-
49	event_evt_wrevt	收到evict/writeevictfull次数。	-
50	event_rns	收到readnosnp个数。	-
51	event_ru_mu_cu_hit	readunique/makeunique/cleanunique命中次数。	-
52	event_wr_bypass	WriteuniqueFull/Ptl和WritenosnoopFull/Ptl bypass 次数。	-
53	event_wb_wc	收到WritebackFull /Ptl/和WritecleanFull/Ptl个数。	-
54	event_rs_rc_hit	readshared/readclean命中次数。	-
55	event_comrepss	数据压缩次数。	-
56	event_mu_cu	收到makeunique/cleanunique个数。	-
57	event_mu	收到makeunique个数。	-
58	event_wr_ext_hit	写ext_way 命中次数。	-
59	event_rns_bypass	readnosnp bypass次数。	-
60	event_stash	收到cachestash个数。	-
61	event_cs	收到cleanshared个数。	-
62	event_rd_ext_hit	读ext_way 命中次数。	-
63	event_dup	duplicate次数。	-
64	event_pkg_capability	pkg请求的LLC使用容量统计。	-

除此之外，LLC支持按地址段统计DFX，即配置LLC的地址段统计DFX配置寄存器中的地址段寄存器，再配置相应统计使能，则将会只对属于这个地址段发生的事件（tag/dat SRAM 1bit/2bit ECC 出错、retry和dirovf_retry除外）进行统计，其他的均不统计。

版权所有 © 海思技术有限公司

LLC支持按cluster或加速器统计DFX，即配置LLC的cluster或加速器统计DFX配置寄存器中的cluster id寄存器，再配置相应统计使能，则将会只对属于这个id的请求进行统计，其他的均不统计。（其中retry、hydra snoop 请求，预取请求，tag/ dat SRAM 1bit/2bit ECC 出错，dirovf请求及dirovf_retry由于和cluster 并无直接联系，所以均无法统计）。

LLC支持按dsid 统计DFX，即配置LLC的DSID DFX配置寄存器中的dsid寄存器，再配置相应统计使能，则将会只对这个dsid的内容进行统计，其他均不统计。（按dsid统计无法统计retry，hydra snoop 请求，overflow，backinvld，容量统计，dirovf请求及dirovf_retry，tag/ dat SRAM 1bit/2bit ECC 出错）。

14.5.18 执行状态显示

Last Level Cache提供Monitor状态功能，以指示LLC的工作情况。共有13个状态寄存器，LLC_INGRESS_EVENT_STATUS0和LLC_BANK*_EVENT_STATUS*（bank0~bank3,status0~status3）；

LLC_INGRESS_EVENT_STATUS主要指示LLC入口队列的空满情况和阻塞情况。

表 14-4 LLC 入口队列状态寄存器列表（注：高有效）

Num	LLC Event	Description
0	Conflict_full	流水线内部的retry队列是否满。
1	Conflict_empty	流水线内部的retry队列是否空。
2	Block_txrsp	写请求是否能发生dbid，读请求是否能发生read receipt。
3	Block_cor_queue	处理一致性的Buffer是否满。
4	Block_cor_table	请求是否被流水线组相关阻塞。
5	Block_lock	请求是否被相同地址的请求Lock阻塞。
6	Ingress_full	入口队列是否满。
7	Ingress_empty	入口队列是否空。

LLC_BANK*_EVENT_STATUS*主要指示LLC 处理队列的遗留请求中最早未处理完的请求的信息；

表 14-5 LLC 处理队列状态寄存器列表（注：高有效）

Num	LLC Event	Description
0	Queue_empty	处理队列是否为空。

版权所有 © 海思技术有限公司

Num	LLC Event	Description
1	Sky_snprsp	是否该请求snp未完成。
2	Hydra_victim_cmp	是否该请求往HA发出的victim请求未完成。
3	Hydra_opt_cmp	是否该请求往HA发出的operation请求未完成。
4	Hydra_ack	是否该请求hydra的ack响应未发送。
5	Sky_resp	是否该请求sky的resp响应未发送。
6	Sky_ack	是否该请求sky的ack响应未收到。
7	Buff_vit	是否该请求申请的victim Buffer未释放。
8	Buff_opt	是否该请求申请的operation Buffer未释放。
9	Buff_hit	是否该请求申请的hit Buffer未释放。
10	Hydra_resp	是否该请求的hydra的响应未发送。
11	Hydra_dync	是否该请求的hydra的pgrant未收到。
12	保留	保留。
13	Srcid	该请求的srcid。
14	Opcode	该请求的Opcode。

表 14-6 LLC 处理队列状态寄存器列表 （注：高有效）

Num	LLC Event	Description
0	Sky_snprsp_sub	相邻64Byte cacheline的snoop响应是否完成。
1	Hydra_victim_cmp_sub	相邻64Byte cacheline的victim req是否完成。
2	Hydra_opt_comp_sub	相邻64Byte cacheline的opt req是否完成。
3	Hydra_ack_sub	相邻64Byte cacheline的hydra ack是否完成。

版权所有 © 海思技术有限公司

Num	LLC Event	Description
4	Buff_vit_sub	相邻64Bytecacheline的victim buff是否未释放。
5	Buff_opt_sub	相邻64Byte cacheline的opt buff是否未释放。
6	Hydra_dync_sub	相邻64Byte cacheline的grant是否未收到。
7	Curr_st	该请求的buff 状态。
8	Curr_st_sub	相邻64Byte cacheline的buff状态。
9	Curr_st_dbid	该请求是否收到dbid。
10	Curr_st_dbid_sub	相邻64Byte cacheline 是否收到dbid。
11	Curr_st_bypass_done	该请求的bypass替换是否完成。
12	Curr_st_bypass_done_sub	相邻64Byte cacheline的bypass 替换是否完成。
13	Curr_st_update	该请求是否收齐写数据。
14	Curr_st_ram_acc_sub	该请求是否正在进行相邻64Byte cacheline的memory操作。

表 14-7 LLC 处理队列状态寄存器列表 （注：高有效）

Num	LLC Event	Description
0	Index	该请求的index。
1	Tag	该请求的tag。

14.5.19 自动初始化

由于Tag RAM的初始状态为未知数据，因此Last Level Cache在使能之前需要将Tag RAM进行初始化，即使用Invalidate操作将Tag RAM所有Valid位都置零。

复位后，Last Level Cache自动进行初始化操作。软件可以通过查询寄存器判断初始化是否结束。初始化完成后可使能Last Level Cache。

14.5.20 SRAM 功能

LLC支持寄存器配置选择SRAM功能。当选择SRAM功能时，LCC的3M存储空间可以作为系统启动期间的部分数据的SRAM使用，负责在系统启动期间存储系统要求的内容。

版权所有 © 海思技术有限公司

14.6 软件指南

14.6.1 初始化

LLC上电初始化步骤如下：

步骤1 复位后，LLC 自动开始初始化操作。

步骤2 查询LLC_INVLD寄存器第2bit（ llc_invld_done ），判断初始化是否结束。若为1，则表示初始化已经结束。

步骤3 清除中断（ 中断不带复位端 ）。

----结束

LLC系统工作中途初始化步骤如下:

步骤1 将LLC_INVLD寄存器的第写1，初始化开始。

步骤2 查询LLC_INVLD寄存器第2bit（ llc_invld_done ），判断初始化是否结束。若为1，则表示初始化已经结束。

步骤3 清除中断（ 中断不带复位端 ）。

----结束

须知

LLC 有SRAM模式和CACHE模式可使用。模式的配置流程参考“**LLC不同工作模式的配置流程**”。

须知

对LLC的任何操作，包含LLC的寄存器配置，功能模式下Mbist测试等，都必须等待LLC初始化完成后进行。

对LLC进行中途初始化，必须保证LLC已处于IDLE（ 可读reg_llc_status寄存器，读其值为1，说明LLC已处于IDLE ），否则很可能导致挂死；如果不希望数据丢失，可进行一次全局maintain，等全局maintain结束后，等到reg_llc_status寄存器的值为1，再进行中途初始化。

14.6.2 软复位

LLC不支持单独软复位。

14.6.3 Maintain 操作

版权所有 © 海思技术有限公司

14.6.3.1 安全全局 Clean

安全全局 Clean的具体配置步骤如下：

步骤1 安全方式写寄存器LLC_MAINT_CTRL配置为全局 Clean模式（mnt_range为00，mnt_type为0b01），并启动（mnt_en为1）。

步骤2 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.2 安全全局 Invalidate

安全全局 Invalidate的具体配置步骤如下：

步骤1 用安全方式写寄存器LLC_MAINT_CTRL配置为全局 Invalidate模式（mnt_range为00，mnt_type为0b10），并启动（mnt_en为1）。

步骤2 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.3 安全全局 Clean+Invalidate

安全全局 Clean+Invalidate的具体配置步骤如下：

步骤1 用安全方式写寄存器LLC_MAINT_CTRL配置为全局 Clean+Invalidate模式（mnt_range为00，mnt_type为0b11），并启动（mnt_en为1）。

步骤2 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.4 安全按地址区段 Clean

安全按地址 Clean的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_START_H和LLC_MAINT_START_L，配置需要执行Maintain操作的起始物理地址。

步骤2 写寄存器LLC_MAINT_AREA_H和LLC_MAINT_AREA_L，配置需要执行Maintain操作的物理地址区间。

步骤3 用安全方式写寄存器LLC_MAINT_CTRL，配置为按地址Clean模式（mnt_range为01，mnt_type为0b01），并配置启动位（mnt_en为1）。

步骤4 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.5 安全按地址区段 Invalidate

安全按地址 Invalidate的具体配置步骤如下：

版权所有 © 海思技术有限公司

步骤1 写寄存器LLC_MAINT_START_H和LLC_MAINT_START_L，配置需要执行Maintain操作的起始物理地址。

步骤2 写寄存器LLC_MAINT_AREA_H和LLC_MAINT_AREA_L，配置需要执行Maintain操作的物理地址区间。

步骤3 用安全方式写寄存器LLC_MAINT_CTRL，配置为按地址Invalidate模式（mnt_range为01，mnt_type为0b10），并配置启动位（mnt_en为1）。

步骤4 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.6 安全按地址区段 Clean+Invalidate

安全按地址 Clean+Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_START_H和LLC_MAINT_START_L，配置需要执行Maintain操作的起始物理地址。

步骤2 写寄存器LLC_MAINT_AREA_H和LLC_MAINT_AREA_L，配置需要执行Maintain操作的物理地址区间。

步骤3 用安全方式写寄存器LLC_MAINT_CTRL，配置为按地址Clean+Invalidate模式（mnt_range为01，mnt_type为0b11），并配置启动位（mnt_en为1）。

步骤4 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.7 安全按 Cluster Clean

安全按Cluster Clean的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_CLUSTER，配置需要执行Maintain操作的Cluster。

步骤2 用安全方式写寄存器LLC_MAINT_CTRL，配置为按Cluster Clean模式（mnt_range为10，mnt_type为0b01），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.8 安全按 Cluster Invalidate

安全按Cluster Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_CLUSTER，配置需要执行Maintain操作的Cluster。

步骤2 用安全方式写寄存器LLC_MAINT_CTRL，配置为按Cluster Invalidate模式（mnt_range为10，mnt_type为0b10），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

版权所有 © 海思技术有限公司

14.6.3.9 安全按 Cluster Clean+Invalidate

安全按Cluster Clean+Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_CLUSTER，配置需要执行Maintain操作的Cluster。

步骤2 用安全方式写寄存器LLC_MAINT_CTRL，配置为按Cluster Clean+Invalidate模式（mnt_range为10，mnt_type为0b11），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.10 安全按 Way Clean

安全按Way Clean的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_WAY，配置需要执行Maintain操作的WAY。

步骤2 用安全方式写寄存器LLC_MAINT_CTRL，配置为按Way Clean模式（mnt_range为11，mnt_type为0b01），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.11 安全按 Way Invalidate

安全按Way Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_WAY，配置需要执行Maintain操作的WAY。

步骤2 用安全方式写寄存器LLC_MAINT_CTRL，配置为按Way Invalidate模式（mnt_range为11，mnt_type为0b10），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.12 安全按 Way Clean+Invalidate

安全按Way Clean+Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_WAY，配置需要执行Maintain操作的WAY。

步骤2 用安全方式写寄存器LLC_MAINT_CTRL，配置为按Way Clean+Invalidate模式（mnt_range为11，mnt_type为0b11），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.13 非安全全局 Clean

非安全全局 Clean的具体配置步骤如下：

版权所有 © 海思技术有限公司

步骤1 用非安全方式写寄存器LLC_MAINT_CTRL配置为全局 Clean模式（mnt_range为00，mnt_type为0b01），并启动（mnt_en为1）。

步骤2 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.14 非安全全局 Invalidate

非安全全局 Invalidate的具体配置步骤如下：

步骤1 用非安全方式写寄存器LLC_MAINT_CTRL配置为全局 Invalidate模式（mnt_range为00，mnt_type为0b10），并启动（mnt_en为1）。

步骤2 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.15 非安全全局 Clean+Invalidate

非安全全局 Clean+Invalidate的具体配置步骤如下：

步骤1 用非安全方式写寄存器LLC_MAINT_CTRL配置为全局 Clean+Invalidate模式（mnt_range为00，mnt_type为0b11），并启动（mnt_en为1）。

步骤2 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.16 非安全按地址区段 Clean

非安全按地址 Clean的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_START_H和LLC_MAINT_START_L，配置需要执行Maintain操作的起始物理地址。

步骤2 写寄存器LLC_MAINT_AREA_H和LLC_MAINT_AREA_L，配置需要执行Maintain操作的物理地址区间。

步骤3 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按地址Clean模式（mnt_range为01，mnt_type为0b01），并配置启动位（mnt_en为1）。

步骤4 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.17 非安全按地址区段 Invalidate

非安全按地址 Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_START_H和LLC_MAINT_START_L，配置需要执行Maintain操作的起始物理地址。

版权所有 © 海思技术有限公司

步骤2 写寄存器LLC_MAINT_AREA_H和LLC_MAINT_AREA_L，配置需要执行Maintain操作的物理地址区间。

步骤3 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按地址Invalidate模式（mnt_range为01，mnt_type为0b10），并配置启动位（mnt_en为1）。

步骤4 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.18 非安全按地址区段 Clean+Invalidate

非安全按地址 Clean+Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_START_H和LLC_MAINT_START_L，配置需要执行Maintain操作的起始物理地址。

步骤2 写寄存器LLC_MAINT_AREA_H和LLC_MAINT_AREA_L，配置需要执行Maintain操作的物理地址区间。

步骤3 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按地址Clean+Invalidate模式（mnt_range为01，mnt_type为0b11），并配置启动位（mnt_en为1）。

步骤4 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.19 非安全按 Cluster Clean

非安全按Cluster Clean的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_CLUSTER，配置需要执行Maintain操作的Cluster。

步骤2 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按Cluster Clean模式（mnt_range为10，mnt_type为0b01），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.20 非安全按 Cluster Invalidate

非安全按Cluster Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_CLUSTER，配置需要执行Maintain操作的Cluster。

步骤2 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按Cluster Invalidate模式（mnt_range为10，mnt_type为0b10），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

版权所有 © 海思技术有限公司

14.6.3.21 非安全按 Cluster Clean+Invalidate

非安全按Cluster Clean+Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_CLUSTER，配置需要执行Maintain操作的Cluster。

步骤2 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按Cluster Clean+Invalidate模式（mnt_range为10，mnt_type为0b11），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.22 非安全按 Way Clean

非安全按Way Clean的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_WAY，配置需要执行Maintain操作的WAY。

步骤2 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按Way Clean模式（mnt_range为11，mnt_type为0b01），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.23 非安全按 Way Invalidate

非安全按Way Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_WAY，配置需要执行Maintain操作的WAY。

步骤2 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按Way Invalidate模式（mnt_range为11，mnt_type为0b10），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

14.6.3.24 非安全按 Way Clean+Invalidate

非安全按Way Clean+Invalidate的具体配置步骤如下：

步骤1 写寄存器LLC_MAINT_WAY，配置需要执行Maintain操作的WAY。

步骤2 用非安全方式写寄存器LLC_MAINT_CTRL，配置为按Way Clean+Invalidate模式（mnt_range为11，mnt_type为0b11），并配置启动位（mnt_en为1）。

步骤3 查询LLC_MAINT_CTRL寄存器最低位（域 mnt_en），判断Maintain操作是否结束。若为0，则表示Maintain操作已经结束；若为1，则表示还在进行中。

----结束

版权所有 © 海思技术有限公司

须知

非安全maintain和安全maintain的区别在于操作的地址属性不同。

对于地址段maintain，非安全maintain只能maintain非安全地址，安全maintain只能maintain安全地址；例如需要有一个非安全地址在LLC中，使用安全地址段maintain不能刷出该cacheline。

对于全局maintain，按way maintain以及按cluster maintain，非安全maintain只能maintain非安全地址，安全maintain会maintain安全和非安全地址（原因详见1.1.5.7节）。

此外，对于地址段maintain，需要注意：1）maintain大段地址会占用LLC的处理能力，所以建议不进行大段地址的maintain，如果需要，建议分解成几段进行maintain操作。2）如果maintain的地址段大于LLC容量，并且曾经确实访问过大量在该地址段的操作，建议直接进行全局maintain，可大大提升maintain速度。

14.6.4 Lockdown 操作

14.6.4.1 按 Way Shutdown

按Way Shutdown可支持将LLC的2个bank中，一条Way或多条Way关闭，在某些情况下可以降低功耗。因此，一共提供了2个寄存器分别独立控制2个bank各自内部的16个std_way的打开或关闭状态。LLC_WAY_SHUTDOWN属于静态配置，需要在初始化流程中配置，且不能动态修改

下面以将第0个bank中第4、5条Way为例，其具体的配置步骤如下：

步骤1 配置Bank0中的寄存器LLC_WAY_SHUTDOWN为0x0030，关闭std_way4～5，将其他Way打开。

步骤2 正常访问LLC，则此时bank0的LLC的可用容量变为了0.875M。

----结束

14.6.4.2 安全按地址段 Lockdown

对一段安全地址进行Lockdown的具体配置步骤如下：

步骤1 写寄存器LLC_LOCK_START_L*和LLC_LOCK_START_AREA_H*的start_h，配置需要执行Lock操作的起始物理地址。

步骤2 写寄存器LLC_LOCK_AREA_L*和LLC_LOCK_START_AREA_H*的area_h，配置需要执行Lock操作的物理地址区间。

步骤3 写寄存器LLC_LOCK_CTRL*的lock_way_en，配置希望将Lock的地址放到LLC的哪些way中。（可选，如果不配，将指定低10条way中任意一条way进行Lock）

步骤4 用安全方式写寄存器LLC_LOCK_CTRL*，配置Lock地址的domain 属性（lock_domain），并配置启动位（lock_en为1，unlock_en为0）。

步骤5 查询LLC_LOCK_CTRL*寄存器最低位（域lock_en），判断Lock操作是否结束。若为0，则表示Lock操作已经结束；若为1，则表示Lock操作还未结束。

----结束

版权所有 © 海思技术有限公司

14.6.4.3 安全按地址段 unLock

对一段安全地址进行Lockdown的具体配置步骤如下：

步骤1 写寄存器LLC_LOCK_START_L*和LLC_LOCK_START_AREA_H*的start_h，配置需要执行Unlock操作的起始物理地址。

步骤2 写寄存器LLC_LOCK_AREA_L*和LLC_LOCK_START_AREA_H*的area_h，配置需要执行Unlock操作的物理地址区间。

步骤3 用安全方式写寄存器LLC_LOCK_CTRL*，配置Unlock地址的domain 属性（lock_domain），并配置启动位（lock_en为0，unlock_en为1）。

步骤4 查询LLC_LOCK_CTRL*寄存器第二bit（域 unlock_en），判断Unlock操作是否结束。若为0，则表示Unlock操作已经结束；若为1，则表示Unlock操作还未结束。

----结束

14.6.4.4 非安全按地址段 Lockdown

对一段非安全地址进行Lockdown的具体配置步骤如下：

步骤1 写寄存器LLC_LOCK_START_L*和LLC_LOCK_START_AREA_H*的start_h，配置需要执行Lock操作的起始物理地址。

步骤2 写寄存器LLC_LOCK_AREA_L*和LLC_LOCK_START_AREA_H*的area_h，配置需要执行Lock操作的物理地址区间。

步骤3 写寄存器LLC_LOCK_CTRL*中的lock_way_en，配置希望将Lock的地址放到LLC的哪些way中。（可选，如果不配，将指定低10条way中任意一条way进行Lock）

步骤4 用非安全方式写寄存器LLC_LOCK_CTRL*，配置Lock地址的domain 属性（lock_domain），并配置启动位（lock_en为1，unlock_en为0）。

步骤5 查询LLC_LOCK_CTRL*寄存器最低位（域lock_en），判断Lock操作是否结束。若为0，则表示Lock操作已经结束；若为1，则表示Lock操作还未结束。

----结束

14.6.4.5 非安全按地址段 Unlock

对一段非安全地址进行Lockdown的具体配置步骤如下：

步骤1 写寄存器LLC_LOCK_START_L*和LLC_LOCK_START_AREA_H*的strat_h，配置需要执行Unlock操作的起始物理地址。

步骤2 写寄存器LLC_LOCK_AREA_L*和LLC_LOCK_START_AREA_H*的area_h，配置需要执行Unlock操作的物理地址区间。

步骤3 用非安全方式写寄存器LLC_LOCK_CTRL*，配置Unlock地址的domain 属性（lock_domain），并配置启动位（lock_en为0，unlock_en为1）。

步骤4 查询LLC_LOCK_CTRL*寄存器第二bit（域 unlock_en），判断Unlock操作是否结束。若为0，则表示Unlock操作已经结束；若为1，则表示Unlock操作还未结束。

----结束

版权所有 © 海思技术有限公司

须知

非安全Lock/Unlock和安全Lock/Unlock的区别在于操作的地址属性不同，即非安全Lock/Unlock只能Lock/Unlock非安全地址，安全Lock/Unlockn只能Lock/Unlock安全地址；

大段地址的Lock/unlock会占用LLC的处理能力，所以建议不进行大段地址的Lock/Unlock，如果需要，建议分解成几段进行Lock/Unlock操作。

对于Lock/Unlock操作，如果需要Lock 8段以上的地址段进LLC，则涉及一组Lock寄存器进行多次Lock的情况，下面是此种情况的操作简介：

如需要用一组Lock寄存器lock 0~2k，4k~6k，9k到10k这三个地址段，那么可以需要配置三次Lock操作。第一次配置起始地址0，区间2k，开启Lock使能，等待Lock结束。Lock结束后重新配置起始地址为4k，区间2k，开启Lock使能，等待Lock结束。Lock结束后重新配置起始地址为9k，区间为1k，开启Lock使能，等待Lock结束。等第三次Lock结束后，LLC已经将0~2k，4k~6k，9k到10k这三个地址段全部Lock进llc了。

14.6.5 虚拟化操作

对虚拟化的具体配置步骤如下：

步骤1 写寄存器LLC_DSID_PART*，配置需要虚拟化的dsid。

步骤2 写寄存器LLC_DSID_WAY_MASK_PART*，配置dsid所使用的way。

步骤3 写寄存器LLC_AUCTRL，配置启动位（dsid_way_mask_part*en）。

----结束

14.6.6 预取操作

14.6.6.1 基于所有 dsid 预取操作

基于所有dsid预取操作的具体配置步骤如下：

步骤1 写寄存器LLC_PREFTCH，配置预取相关信息（具体信息见寄存器表单），也可直接使用默认值。

步骤2 写寄存器LLC_AUCTRL，配置启动位（prefetch_en）。

----结束

14.6.6.2 基于配置的部分 dsid 预取操作

基于配置的部分dsid预取操作的具体配置步骤如下：

步骤1 写寄存器LLC_PREFTCH，配置预取相关信息（具体信息见寄存器表单），也可直接使用默认值。

步骤2 写寄存器LLC_CTRL，配置预取基于dsid模式。

步骤3 写寄存器LLC_REG_ID_PART*，配置所需预取的dsid。

步骤4 写寄存器LLC_REG_ID_PART*，配置所需预取的dsid对应的预取使能。

----结束

版权所有 © 海思技术有限公司

14.6.6.3 基于配置的部分{lpid，srcid}预取操作

基于配置的部分{lpid，srcid}预取操作的具体配置步骤如下：

步骤1 写寄存器LLC_PREFTCH，配置预取相关信息，也可直接使用默认值。

步骤2 写寄存器LLC_CTRL，配置预取基于{lpid，srcid}模式。

步骤3 写寄存器LLC_REG_ID_PART*，配置所需预取的{lpid，srcid}。

步骤4 写寄存器LLC_REG_ID_PART*，配置所需预取的{lpid，srcid}对应的预取使能。

----结束

14.6.7 性能跳水操作

14.6.7.1 基于所有 dsid 性能跳水操作

基于所有dsid性能跳水操作的具体配置步骤如下：

步骤1 写寄存器LLC_ALLOC_FREQ，配置性能跳水相关信息（具体信息见寄存器表单），也可直接使用默认值。

步骤2 写寄存器LLC_AUCTRL，配置启动位（alloc_freq_en）。

----结束

14.6.7.2 基于配置的部分 dsid 性能跳水操作

基于配置的部分dsid性能跳水操作的具体配置步骤如下：

步骤1 写寄存器LLC_ALLOC_FREQ，配置性能跳水相关信息（具体信息见寄存器表单），也可直接使用默认值。

步骤2 写寄存器LLC_CTRL，配置性能跳水基于dsid模式。

步骤3 写寄存器LLC_REG_ID_PART*，配置所需性能跳水的dsid。

步骤4 写寄存器LLC_REG_ID_PART*，配置所需性能跳水的dsid对应的性能跳水使能。

----结束

14.6.7.3 基于配置的部分{lpid，srcid}性能跳水操作

基于配置的部分{lpid，srcid}性能跳水操作的具体配置步骤如下：

步骤1 写寄存器LLC_ALLOC_FREQ，配置性能跳水相关信息（具体信息见寄存器表单），也可直接使用默认值。

步骤2 写寄存器LLC_CTRL，配置性能跳水基于{lpid，srcid}模式。

步骤3 写寄存器LLC_REG_ID_PART*，配置所需性能跳水的{lpid，srcid}。

步骤4 写寄存器LLC_REG_ID_PART*，配置所需性能跳水的{lpid，srcid}对应的性能跳水使能。

----结束

版权所有 © 海思技术有限公司

14.6.8 地址段统计 DFX 操作

对一段地址进行DFX统计的具体配置步骤如下：

步骤1 写寄存器LLC_BANK_EVENT_TYPE0和LLC_BANK_EVENT_TYPE1，配置需要统计的事件类型。

步骤2 写寄存器LLC_DFX_START，配置需要执行地址段DFX操作的起始物理地址。

步骤3 写寄存器LLC_DFX_AREA，配置需要执行地址段DFX操作的物理地址区间。

步骤4 写寄存器LLC_CTRL中的dfx_addr_en和和LLC_AUCTRL中的event_bus_en，即可对该地址段进行DFX统计。

----结束

14.6.9 Cluster 统计 DFX 操作

对一个cluster进行DFX统计的具体配置步骤如下：

步骤1 写寄存器LLC_BANK_EVENT_TYPE0和LLC_BANK_EVENT_TYPE1，配置需要统计的事件类型。

步骤2 写寄存器LLC_DFX_CLUSTER，配置需要执行Cluster DFX操作的Cluster。

步骤3 写寄存器LLC_CTRL中的dfx_cluster_en和LLC_AUCTRL中的event_bus_en，即可对该Cluster进行DFX统计。

----结束

14.6.10 Dsid 统计 DFX 操作

对一个dsid进行DFX统计的具体配置步骤如下：

步骤1 写寄存器LLC_BANK_EVENT_TYPE0和LLC_BANK_EVENT_TYPE1，配置需要统计的事件类型。

步骤2 写寄存器LLC_DSID_PART*，配置所需统计的具体DSID的值。

步骤3 写寄存器LLC_DFX_DSID，配置所需统计的DSID的位置。（ LLC_DFX_DSID中的0b000~0b111对应LLC_DSID_PART*中的dsid0~dsid7 ）

步骤4 配置LLC_CTRL中的llc_dfx_dsid_en即可对该Dsid进行DFX统计。

----结束

14.6.11 LLC 内部 Buffer 状态信息查询配置流程

LLC内部Buffer状态信息查询的具体配置步骤如下：

步骤1 如果需要查询最早请求的Buffer信息，则配置LLC_CTRL中的earliest_en为1，查询LLC_BANK*_EVENT_STATUS*即可读出各个bank的该Buffer状态即可。

步骤2 如果需要查询特定的Buffer信息，则配置LLC_CTRL中的earliest_en为0，再配置LLC_DFX_QUEUEID配置Buffer号，查询LLC_BANK*_EVENT_STATUS*即可读出各个bank的该Buffer状态即可。

----结束

版权所有 © 海思技术有限公司

14.6.12 LLC 特定 DSID 在 LLC 所在容量查询配置流程

LLC特定DSID在LLC所在容量查询配置步骤如下：

步骤1 写寄存器LLC_DSID_PART*，配置所需统计的具体DSID的值。

步骤2 配置LLC_BANK_EVENT_TYPE*为统计DSID容量。

步骤3 配置LLC_AUCTRL中的reg_event_en，LLC即会对该时刻以后的LLC对配置DSID的容量进行统计（统计的是该时刻以后对该dsid的容量统计，若开启reg_event_en之前该Dsid已经在LLC中占有一部分容量，可能出现容量为0还有cacheline溢出的场景，此时容量保持为0）。

----结束

14.6.13 强制地址段为 Inclusive 结构 的 LLC 配置流程

强制地址段为Inclusive结构的LLC配置步骤如下：

步骤1 配置LLC_INCLUSIVE_START_H_*和LLC_INCLUSIVE_START_L_*，配置地址段的起始地址。

步骤2 配置LLC_INCLUSIVE_SIZE_*，配置地址段的size。即可将该地址段强制为Inclusive结构。（size为0表示关闭）

----结束

14.6.14 LLC 不同工作模式的配置流程

SRAM工作模式配置步骤如下：

步骤1 查询LLC_INVLD寄存器第2bit（llc_invld_done），等待初始化完成。若为1，则表示初始化已经结束。

步骤2 配置LLC_CTRL_S寄存器第1bit（l3d_sram_en）为1。

步骤3 配置ACCESS的DAW（18）窗口，SRAM对应的3M空间去LLC的节点。（节点号是24和25）。

步骤4 配置ACCESS_CTRL_S寄存器第1bit（l3d_sram_en）为1。

步骤5 软件写LLC 3M SRAM全空间为0进行初始化。

步骤6 配置LLC_BANK_INTC寄存器，清除中断（中断不带复位端）。

步骤7 确认中断清除后，即可开始SRAM工作模式，进行数据存储。

----结束

Cache工作模式配置步骤如下：

步骤1 查询LLC_INVLD寄存器第2bit（llc_invld_done）等待初始化完成。

步骤2 配置ACCESS_CTRL_S寄存器第1bit（l3d_sram_en）为0。

步骤3 关闭ACCESS的DAW（18）窗口。

步骤4 配置LLC_CTRL_S寄存器第1bit（l3d_sram_en）为0。

步骤5 将初始化寄存器LLC_INVLD寄存器的第1 bit（llc_invld_start）再重新写1（重新进行一次初始化）。

版权所有 © 海思技术有限公司

步骤6 确认初始化完成后（查询LLC_INVLD寄存器第（llc_invld_done）为1），即可开始Cache工作模式，进行请求处理。

----结束

须知

如果上电后直接进入cache模式或者从sram模式切换到cache模式时，请务必配置ACCESS_CTRL_S寄存器和LLC_CTRL_S寄存器第1bit为0，否则出现一致性出错或者挂死。

14.7 性能

无。

14.8 DFX 应用指导

无。

14.8.1 故障在线检测

- RAM出现ECC 2bit错误，报中断，硬件无法恢复，芯片可复位。
- RAM ECC 1bit 错误内部自动纠错，数据正常使用，可配置报中断，计数软件可写清。

14.8.2 模块维测

14.8.2.1 功能统计

LLC 支持多种类型事件的统计，支持对8个事件进行同时统计，通过配置LLC_BANK_EVENT_TYPE*来确定指定计数器统计的内容，通过读取寄存器：LLC_BANK_EVENT_CNT*来得出统计值。

表 14-8 功能统计

可用Cnt编号	统计事件号	事件名称
所有Cnt均可统计	00000000	读Allocate请求；
	00000001	写Allocate请求；
	00000010	读noAllocate请求；
	00000011	写noAllocate请求；
	00000100	读命中；
	00000101	写命中；

版权所有 © 海思技术有限公司

可用Cnt编号	统计事件号	事件名称
	00000110	cmo操作；
	00000111	copyback请求；
	00001000	hydra snoop请求；
	00001001	smmu请求；
	00001010	exclusive成功操作；
	00001011	exclusive失败操作；
	00001100	溢出drop操作；
	00001101	溢出backinvalid操作；
	00001110	收到预取请求
	00001111	Retry请求；
	00010000	TAG RAM ECC出2bit错；
	00010001	DAT RAM ECC出2bit错；
	00010010	目录溢出请求个数；
	00010011	目录溢出请求被retry个数；
	00010100	TAG RAM ECC出1bit错；
	00010101	DAT RAM ECC出1bit错；
	00010110	往cluster00发出的snoop次数；
	00010111	往cluster01发出的snoop次数；
	00011000	往cluster02发出的snoop次数；
	00011001	往cluster03发出的snoop次数；
	00011010	往cluster04发出的snoop次数；
	00011011	往cluster05发出的snoop次数；
	00011100	往cluster06发出的snoop次数；
	00011101	往cluster07发出的snoop次数；
	00110000	DSID00的LLC使用容量统计；
	00110001	DSID01的LLC使用容量统计；
	00110010	DSID02的LLC使用容量统计；
	00110011	DSID03的LLC使用容量统计；
	00110100	DSID04的LLC使用容量统计；

版权所有 © 海思技术有限公司

可用Cnt编号	统计事件号	事件名称
	00110101	DSID05的LLC使用容量统计；
	00110110	DSID06的LLC使用容量统计；
	00110111	DSID07的LLC使用容量统计；
	00111000	pkg请求的LLC使用容量统计；
Cnt0	01000000	readshared/readclean；
	01000001	allocate std sub 请求个数；
	01000010	allocate ext 请求个数；
	01000011	reserved0；
Cnt1	01000000	readunique/makeunique/cleanunique；
	01000001	allocate ext SD 请求个数；
	01000010	allocate std 请求个数；
	01000011	reserved2；
Cnt2	01000000	writeuniquefull/writeuniqueptl；
	01000001	reserved8；
	01000010	ext转移到std；
	01000011	reserved3；
Cnt3	01000000	writenosnpfull/writenosnpptl；
	01000001	readonce；
	01000010	std转移到ext；
	01000011	reserved4；
Cnt4	01000000	evict/writeevictfull；
	01000001	readnosnp；
	01000010	readunique/makeunique/cleanunique命中；
	01000011	write bypass 次数；
Cnt5	01000000	writebackfull /writebackptl/ writecleanfull/writecleanptl；
	01000001	reserved6；
	01000010	readshared/readclean命中；
	01000011	数据压缩次数；
Cnt6	01000000	makeunique/cleanunique；

版权所有 © 海思技术有限公司

可用Cnt编号	统计事件号	事件名称
	01000001	makeunique；
	01000010	写ext 命中；
	01000011	readnosnp bypass次数；
Cnt7	01000000	cachestash；
	01000001	cleanshared；
	01000010	读ext 命中；
	01000011	duplicate次数；

14.8.3 中断上报

LLC支持两种中断的上报：

- 统计事件溢出中断。
- 异常中断。

第一种中断在计数器到全f的时候即会上报一个中断,可以用于热点函数的捕捉和LLC DSID的容量限制。热点函数的捕捉即可以设置寄存器的初值，然后按地址或cluster进行关心的时间统计，溢出则上报一个中断，通过多次统计即可捕捉到热点函数。
LLCDSID 容量限制即将统计事件配为DSID的容量统计，然后设定初值，如果报中断，则说明该DSID的容量超过预期，则可以通过软件调节即可以保证容量的限制。

第二种中断显示LLC处于不正常的状态或收到不正常的请求，可用于问题定位。

须知

由于中断没有复位端，所以在上电复位后需要清除中断,以免造成错误的中断上报。

14.8.4 典型定位案例

NA。

14.9 注意事项

LLC不支持混合属性的访问。

版权所有 © 海思技术有限公司

15 SRAM 模块

15.1 概述

SRAM模块主要实现片上Memory的访问，支持ECC功能，支持快速原子加、原子减操作，提高多核的Exclusive 操作效率。

15.2 特性

SRAM具有以下特性：

- SRAM物理空间为256KByte，支持三种不同属性空间的配置，分别为Normal空间，原子加空间，原子减空间。
- SRAM整体所占的地址空间范围为其物理空间的4倍，并且按物理空间大小分成4部分地址空间，从基地址开始依次按照Normal空间，原子加空间，原子减空间，寄存器空间进行地址空间排序。
- 空间属于Normal操作的行为：
  - Normal操作最大空间为256KBytes。
  - 支持Unaligned Transfers，即首地址不对齐Burst操作。
  - Normal空间接收到Exclusive操作，返回OKAY响应表示异常，忽略Lock功能。
  - 读写Size最大支持128bits，对于大于128bits的操作，返回SLVERR。
  - 最大支持Burst 16操作。
  - 收到不支持的操作则返回SLVERR响应。
- 空间属于硬件原子操作（Exclusive）的行为：
  - 硬件原子操作最大空间为256KBytes。
  - 操作数支持快速加n，减n，n为8/16/32/64bits位宽（对于64bits原子操作需要CPU支持64bits的Single操作）。
  - 被操作数与操作数同位宽。
  - 支持设置被操作数中的任意bit。
  - 原子加减操作溢出不做判断，由软件保证，加到最大或减到最小时，会进行卷绕。

版权所有 © 海思技术有限公司

  - 原子操作不支持Burst操作，均为Single操作。
  - 原子操作支持Swap大小端配置，默认工作在小端模式。
- 空间属于寄存器操作的行为：
  - SRAM寄存器可以通过DJtag方式进行访问，也可以通过AXI方式进行访问。
  - 通过AXI方式访问SRAM寄存器时，按照访问SRAM寄存器空间形式来实现，即寄存器空间的基地址加上寄存器的偏移地址方式访问。
- 支持ECC特性。
  - ECC功能支持“纠一检二”（SECDED）。
  - 单bit ECC错误不返回SLVERR响应，2bits ECC错误可通过配置选择是否返回SLVERR响应。
  - 支持ECC错误统计计数器，单bit ECC错误数统计，2bit ECC错误数统计。
  - ECC错误统计使用64k饱和计数器，计数满停止，直到软件清零。
  - 支持第一次ECC错误地址记录，可通过CPU写清，记录下一次ECC错误地址。
  - 单bit ECC错误自动纠错，不上报中断，2bits ECC错误上报中断。
  - ECC单bit出错后自动纠错并回写到SRAM中。
- 支持安全访问控制。
  - 在非安全模式下，无法访问安全区域内的数据，写操作无法写入，读操作回读全零，Resp不会反馈异常。
  - 安全访问空间的地址配置必须为16B对齐。
  - SRAM的安全空间需要分配在Normal区域内，不可分配在原子加减区域内。
  - SRAM的默认配置为全空间只接受安全访问。
- 支持动态时钟门控功能。
  - 当SRAM不使用时，支持SRAM整模块的时钟关断。（注：Hi309a不支持此功能）
  - 支持Memory的时钟自动门控，即有Memory读写才将时钟打开，否则将时钟关闭。
- 支持SRAM寄存器启动硬件自动初始化。
- 支持SRAM故障注入测试，支持单bit和2bit ECC错误注入。
- SRAM提供四组32bit DFX计数器，可统计SRAM接收操作数、原子操作数等信息。
- 支持MBIST功能，在SRAM不使用时，可启动MBIST功能。
- SRAM提供两种寄存器访问接口，可通过AXI总线访问SRAM的寄存器，也可通过DJTAG BUS访问SRAM的寄存器。

15.3 关键术语

表15-1为与SRAM相关的关键术语表。

表 15-1 SRAM 术语表

术语	描述
AHB	Advanced High-performance Bus；高级高性能总线。

版权所有 © 海思技术有限公司

术语	描述
AXI	Advanced eXtensible Interface；高级可扩展接口。

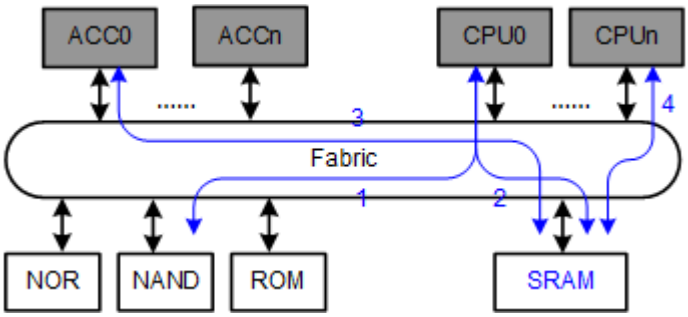
15.4 遵循协议

SRAM接口为标准AXI4接口。

15.5 功能描述

SRAM有效存储空间共256kBytes，同时支持Normal属性的操作和Exclusive属性的操作。对于同一个物理地址空间，同一时刻仅支持一种属性的操作。需要软件在使用前规划好Normal属性的空间和Exclusive属性的空间大小，避免数据踩踏导致软件出错。

图 15-1 SRAM 应用框图



数据流1，2：主CPU负责启动流程，将储存设备（NOR/NAND/ROM等）中的指令和数据读入并执行，为了提高启动效率，将指令和数据拷贝到SRAM中存放，供后续启动使用；另外在系统运行过程中，可以将常用的数据存储在SRAM中，避免频繁的DDR读写。

数据流3：加速器或外设可以对SRAM的数据进行访问，实现数据共享；加速器或外设可以对SRAM的原子空间进行操作，更新统计信息。

数据流4：多个CPU之间通过对SRAM的原子空间访问，获取统计信息；可以使用SRAM做Flag的判断，控制业务流程。

15.5.1 Normal 空间操作

Normal空间的SRAM一般用于数据的存储，即作用相当于一个较大的Buffer空间。CPU或加速器将一些重要的中间数据放在SRAM，减少对DDR的读写次数，可以提升命令处理性能。

SRAM地址空间默认为Normal属性，支持Unaligned Increment操作，最大支持INCR 16的Burst操作。对于Unaligned的读操作，非对齐部分返回无用数据，如果非对齐部分为非安全访问安全属性，则返回全零数据；对于Unaligned的写操作，根据Strobe信号决定更新的数据，如果非对齐部分非安全访问安全属性，则安全部分无法写入。非对齐操作的Resp均为OKAY。当接收到的操作Size大于128bits或Burst大于16时，则返回SLVERR响应。

版权所有 © 海思技术有限公司

15.5.2 Exclusive 空间操作

Exclusive操作即原子操作，分为硬件原子加和原子减两类操作，SRAM空间对应地划分为两类空间。两类空间支持的特性和操作方式类似，访问原子加空间触发加操作，访问原子减空间触发减操作。

根据CPU发出的Size属性，判断当前操作数的位宽，支持8/16/32/64bits不同位宽的原子操作。SRAM收到CPU访问原子空间的读操作，直接返回对应地址的数据；如果为写操作，则先读取该地址的数据，加上或减去写操作数，再回写到SRAM中。对于原子加减，计数器溢出时直接做卷绕处理。

SRAM原子操作空间为Device空间，所以接收到的操作均为Single操作，如果接收到非Single操作或size大于64bit则返回SLVERR。

15.5.3 安全属性操作

SRAM的安全属性通过判断AXI接口的NS域实现，NS即Non-secure指示，1表示非安全，0表示安全。NS域的取值由SMMU/MMU产生。

安全访问可以访问任意空间，即包含安全空间和非安全空间；非安全访问只能访问非安全空间。若非安全读访问安全空间，则返回全零数据；若非安全写访问安全空间，则无法写入。SRAM的默认配置为全空间只接受安全访问。

安全空间仅在Normal属性的空间有效，原子操作空间均为非安全空间。安全空间的地址范围通过配置寄存器配置，地址必须按照16B对齐。

说明

对于非对齐操作，如果某非安全访问的空间，部分涉及安全属性，则安全属性部分按照“读全零，写失败”处理，非安全部分仍可完成。

非安全空间访问安全空间，Resp为OKAY类型。

SRAM全空间只接受安全访问的默认配置由全局安全空间使能信号（SRAM_SEC_START[17]）控制。

15.5.4 ECC 功能

ECC功能覆盖整个SRAM空间，并且提供统计计数器和中断上报功能。ECC功能为“纠一检二”（SECDED）级别，即可以识别并纠正1bit ECC错误，只识别不纠正2bit ECC错误，对于2bit以上ECC错误，当前算法无法支持，可能导致统计计数值出错或中断上报失效。

1-bit ECC

提供状态查询，显示当前SRAM是否发生ECC错误，若状态显示有ECC错误，则可以查询首个出错的地址，以及总共出错的次数。通过写清，将出错地址和出错次数复位。1bit ECC错误不上报中断，纠错后直接回写到SRAM中。

2-bit ECC

提供状态查询，显示当前SRAM是否发生ECC错误，若状态显示有ECC错误，则可以查询首个出错的地址，以及总共出错的次数。通过写清，将出错地址和出错次数复位。2bit ECC错误上报中断，无法纠错。

版权所有 © 海思技术有限公司

须知

若出现2bit ECC错误，可能导致CPU运行出错，系统挂死。通过复位后读取中断寄存器判断是否产生2bit ECC错误。

若先发生1bit ECC错误，后发生2bit ECC错误，ECC状态寄存器会记录第一次2bit ECC地址。

15.6 软件指南

15.6.1 配置 SRAM 初始化流程

1. SRAM_CTRL寄存器（偏移地址为0x30_0000）先写0x0000_0000，再写0x0000_0001，启动SRAM的memory初始化，SRAM不支持上电自动初始化，需软件配置进行初始化，上升沿启动初始化。
2. 配置SRAM_CLR_STAT寄存器（偏移地址为0x30_0070）为0x0000_01FF，清除SRAM状态寄存器和DFX计数器，配置SRAM_INTCLR（偏移地址为0x10_004C）为0x0000_0FFF，清除中断状态。
3. 配置SRAM工作方式：根据功能需求配置SRAM_CTRL寄存器（偏移地址为0x30_0000），bit[0]为init，初始化使能，低电平变为高电平启动初始化；bit[1]为low_power，低功耗配置，高电平有效，使能后可动态关闭SRAM时钟，默认使能；bit[2]为ecc_en，使能ECC纠检错，高电平有效，默认高电平；bit[3]为multi_ecc_report，多bit ECC错误上报slave error配置，高电平有效，默认有效；bit[4]为excl_endien，原子空间大小端配置，低电平为小端，高电平为大端，默认低电平；bit[5]为cnt_en，DFX计数器使能，高电平有效，默认低电平，ECC计数器不受该寄存器控制；bit[6]地址译码错误返回decode error使能，高电平有效，默认低电平；bit[8]为基地址使能，高电平有效，默认低电平，配置该使能前，应先在reg_base_addr配好SRAM基地址，再拉高基地址使能；bit[9]原子加空间使能，高电平有效，默认高电平；bit[10] 原子减空间使能，高电平有效，默认高电平。
4. 查询SRAM状态寄存器SRAM_STATE（偏移地址为0x30_0004），初始化结束后（bit0为1），SRAM可正常访问。

须知

- SRAM_CTRL寄存器中基地址使能默认为低电平，如果配置基地址使能，当出现地址卷绕访问情况时，就会上报错误。
- 如果不配置基地址使能，当出现地址卷绕访问情况时，会正常处理不上报，因此如果要防止此情况发生，需要例化sram的模块去约束不会出现地址卷绕访问的情况。

5. 每次解复位和开启时钟后，必须读取时钟和复位状态，确保模块已经完成解复位和开时钟，然后才能启动模块功能。

15.6.2 正常 ECC 使用流程

在使用ECC之前，需要先清除所有ECC统计状态，初始化所有地址，然后才能正常使用ECC。

版权所有 © 海思技术有限公司

15.6.3 构造 ECC 错误的推荐方法

1. 打开ECC使能，配置SRAM_CTRL中的ecc_en位为0b1。
2. 配置ECC注入寄存器SRAM_ECC_INJECT（偏移地址为0x0060），使能ECC注入，并选择ECC注入bit；（改步可配置注入单bit或2bit ECC错误）。
3. 向需要注入ECC错误的地址写入数据。
4. 读取3中注入ECC错误的地址。

须知

在ECC已开启的情况下，需要软件保证遵循先初始化，后使用的步骤。

15.6.4 CPU 因 Data ECC 异常进入 Abort 状态配置

1. 配置寄存器SRAM_CTRL（偏移地址为0x0）的multi_ecc_report错误传输使能位为0b1；
2. 在CPU初始化的时候配置CPU控制信号cpsr的Abit位（Asynchronous abort mask bit）为0；指令为CPSIE a,#19；
3. 使能ECC控制，配置SRAM_CTRL中的ecc_en位为0b1。

15.6.5 软复位

1. 正常工作阶段，SRAM内可能存在正在处理的命令，不能直接进行SRAM模块的软复位，需等待无命令时才能软复位。
2. 如果要求模块软复位，需要与模块时钟门控成对配合使用，建议按照如下复位流程：先发起模块软复位--->再关闭模块时钟门控--->再撤离模块软复位--->最后打开模块时钟门控。
3. 每次解复位和开启时钟后，必须读取时钟和复位状态，确保模块已经完成解复位和开时钟，然后才能启动模块功能。

15.7 性能

SRAM的工作频率最高位625MHz，接口位宽为128bit。

- 对于Normal空间，读写吞吐率共计（最高）为：

  128bits×625MHz =80Gbps =10GB/s
- 对于Exclusive空间，数据位宽为64bit，读吞吐率为：

  64bits×625MHz÷2=40Gbps =5GB/s
- 对于Exclusive空间，数据位宽为64bit，写操作需要进行读-改-写，带宽减半，写吞吐率为：

  64bits×625MHz= 20Gbps = 2.5GB/s

15.8 DFX

版权所有 © 海思技术有限公司

15.8.1 故障注入

SRAM支持1bit故障注入测试：

步骤1 使能ECC控制，配置SRAM_CTRL中的ecc_en位为0b1。

步骤2 配置寄存器SRAM_ECC_INJECT（偏移地址为0x60），仅使能bit1_en，不使能bit2_en，配置bit1_sel选择注入故障的bit位置。

步骤3 产生写操作，向SRAM中写入数据。

步骤4 产生读操作，读取**步骤2**中的写操作地址。

步骤5 ECC单bit错误统计加1。

SRAM支持2bit故障注入测试。

步骤6 使能ECC控制，配置SRAM_CTRL中的ecc_en位为0b1。

步骤7 配置寄存器SRAM_ECC_INJECT（偏移地址为0x60），同时使能bit1_en和bit2_en，配置bit1_sel和bit2_sel选择注入故障的bit位置，bit1_sel和bit2_sel值不能相等。

步骤8 产生写操作，向SRAM中写入数据。

步骤9 产生读操作，读取**步骤2**中的写操作地址。

步骤10 ECC多bit错误统计加1，且上报中断。

----结束

15.9 注意事项

SRAM对应的页表属性在原子加减操作时需要配置为Device属性。

版权所有 © 海思技术有限公司

16 SMMU

16.1 Overview

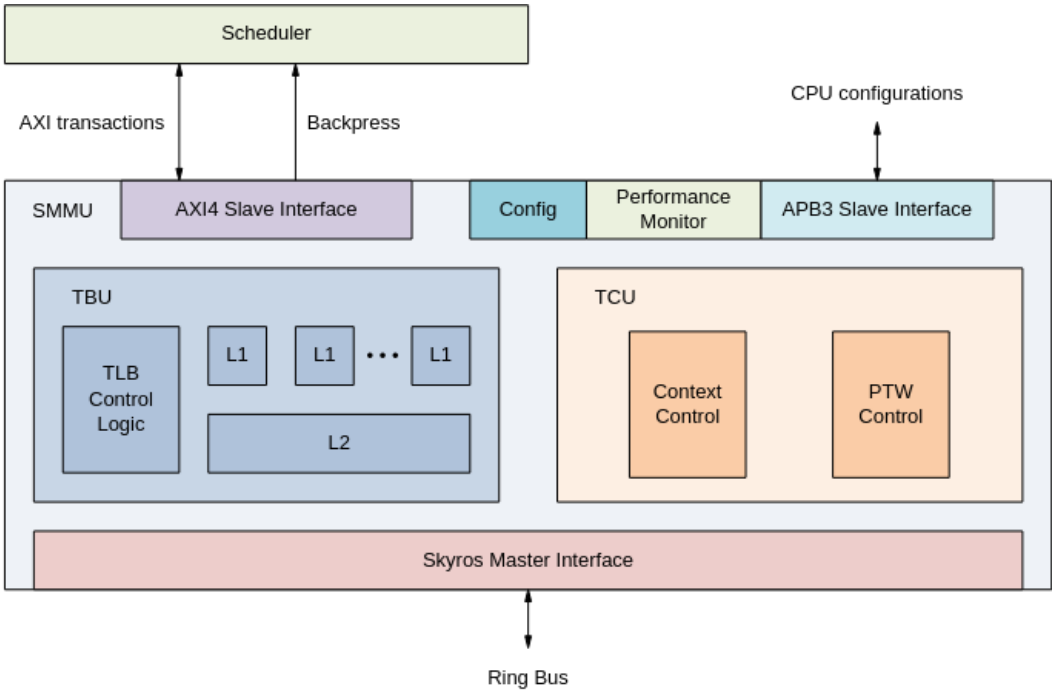
系统存储管理单元（SMMU）是ARM架构下实现虚拟化扩展（Virtualization Extensions）所提供的一个重要组件。该结构为虚拟化具体的实现提供了一种灵活的框架，其可以应用于I/O及加速引擎的虚拟化。在AArch32模式下通过LPAE（Large Physical Address Extension）可实现对40位物理地址空间的访问，在AArch64模式下可实现对48位物理地址空间的访问。

系统存储管理单元（SMMU）是在多个Guest OS被一个Hypervisor管理的虚拟化系统中使用的，主要支持虚拟地址（VA）到中间物理地址（IPA）以及中间物理地址（IPA）到物理地址（PA）的地址转换。

系统存储管理单元（SMMU）可以同时处理来自多个设备的Transactions，并且根据这些Transactions的Stream ID来使用不同的Stage1和Stage2的页表进行地址和属性转换。

版权所有 © 海思技术有限公司

图 16-1 smmu structure



16.2 Features

- SMMU.FS01：SMMU完成Scheduler到Ring Bus之间的跨接和地址转换，完成设备实际对Ring Bus的接入：
  - SMMU和Scheduler采用同步接口。
  - SMMU通过同步接口与Dispatch和Ring Bus接入模块相连。
  - SMMU运行频率为Ring Bus频率的1/2。
- SMMU.FS02：SMMU包括如下总线接口：
  - SMMU和Scheduler采用AXI4兼容接口。
  - SMMU配置采用APB3兼容接口和DJTAG接口。
  - SMMU和Ring Bus采用CHIE的RN-D接口。
- SMMU.FS03：支持Stage1虚拟地址（VA）到中间物理地址（IPA）的地址转换。
- SMMU.FS04：支持Stage2中间物理地址（IPA）到物理地址（PA）的地址转换。
- SMMU.FS05：支持Stage1+Stage2（虚拟地址VA到物理地址PA）转换。
- SMMU.FS06：支持配置Stage1透传、Stage2透传和整体透传模式。
- SMMU.FS07：支持当前多种页表格式：
  - 支持AArch64的 ARMv8-A长页表格式。
  - 支持AArch32的ARMv7-A LPAE长页表格式。
- SMMU.FS08：AArch32模式下支持最小页表颗粒，AArch64模式下支持4kByte、16kByte和64kByte三种最小页表颗粒。
- SMMU.FS09：Stage1地址转换支持最大对48-bit IPA空间。

版权所有 © 海思技术有限公司

- SMMU.FS010：Stage2地址转换支持最大对48-bit PA 空间。
- SMMU.FS011：SMMU提供分布式最大和统一的2048-entry 4-way set associative L2 TLB，L1 TLB为每个上一级Scheduler接入的Master独享，两级TLB采用exclusive策略。
- SMMU.FS012：TLB entry支持ARMv7-A LPAE长页表格式和ARMv8-A页表格式，TLB页表颗粒粒度包括、2MB、1GB，16kByte、32MByte以及64kByte、512MByte。
- SMMU.FS013：支持页表Contiguous属性，提升TLB利用率。
- SMMU.FS014：支持转换页表大小端可配（只是转换页表，不包括STD，STE，CD，Command Queue，Event Queue等数据结构）。
- SMMU.FS015：支持处理DVM消息（TLB invalid and sync）。
  - 不支持DVM发往扩展端口（即TP口）。
- SMMU.FS016：SMMU支持同时存在多个活跃的上下文场景进行地址转换，上下文场景根据命令的VMID和ASID可以进行动态切换，支持16bitsASID。
  - 不支持相同的VMID和ASID组合对应不相同的页表基地址。
- SMMU.FS017：支持地址转换的错误处理，在发生错误的情况下可以产生错误报告或错误中断，支持Fault命令stall功能。
- SMMU.FS018：支持地址空间读写权限管理，当出现地址空间访问权限错误，支持产生错误报告或错误中断。
- SMMU.FS019：支持TrustZone扩展。
- SMMU.FS020：支持通过Command Queue接收处理软件Command。
  - 支持TLB无效化命令。
  - 支持STE无效化命令。
  - 支持CD无效化命令。
  - 支持CMD_ATC_INV。
  - 支持CMD_RESUME。
  - 支持CMD_STALL_TERM。
  - 支持CMD_SYNC。
  - 支持CMD_ FREFETCH_CONFIG。
- SMMU.FS021：支持通过Event Queue通知软件处理配置错误。
- SMMU.FS022：支持Walk cache存储Page Table Walk过程中的table页表，、16k、64k页表粒度的每级PTW各8个entry。
- SMMU.FS023：每次PTW读取半个Cache line，对于最后一级PTW，支持将多余的页表存储下来做下次Page Table Walk查找，提高连续地址空间的访问效率。
- SMMU.FS024：支持Context Cache功能，对转换数据结构（STE、CD）进行缓存。
  - Context Cache深度可配，支持16、64和128深度可配。
- SMMU.FS025：支持Memory ECC校验，支持纠正，2bit ECC上报中断。存在ECC的memory有：L2 TLB，两个AXI通道的write buffer，Context Cache。
  - 支持寄存器ECC注入功能。
- SMMU.FS026：支持最大读写各的保序命令的处理。
- SMMU.FS027：支持PCIE ATS功能。

版权所有 © 海思技术有限公司

- SMMU.FS028：支持SubstreamID功能。
  - 对于使用substreamid的请求，需要将ssv置1；对于不使用substreamid的请求，需要将ssv置0。
  - 支持对ssv=0时的substreamid作默认值处理：STE.S1DSS配置为0b00时，对该请求进行检测; STE.S1DSS配置为0b01时，对该请求进行stage1 bypass处理; STE.S1DSS配置为0b10时，将substreamid当做0处理（此时ssv=1，且SubstreamID为0的请求会被检测）;1280 SMMU驱动需根据加速器的行为满足此配置需求。
- SMMU.FS029：
- SMMU.FS030：支持PCIE的原子操作。
- SMMU.FS031：支持Stash操作，支持将加速器发过来的Stash 逻辑核号按照每个CPU Die 16核转为物理核号。
  - 不支持stash的跨片，每个chip的逻辑核号只能为0~31，无论单片还是多片。
  - 不满足allocate属性的stash操作将发往default slave，包括配置FNA强制为non-allocate的情况。
- SMMU.FS032：支持SMMUv3协议规定的RAS功能。
  - 单bit ECC错误报CE。
  - SMMUv3协议中Fetch Error相关的Event报UER。
  - 两bit ECC错误报UEU。
- SMMU.FS033：支持WFE中断唤醒功能。
  - 当CMD_SYNC完成，且要求SIG_SEV时，发送WFE中断。
  - 当CMDQ从满到非满时，发送WFE中断。
- SMMU.FS034：支持PCIE的SO功能。
  - 支持IO cache功能解决PCIE的SO。
  - 支持Streaming Order Write功能解决PCIE的SO。
  - 支持寄存器控制两种模式的切换。
- SMMU.FS035：支持双端口SMMU。
- SMMU.FS036：支持AXI数据通道。
- SMMU.FS037：支持保序的粒度的AXI wrap读操作。
- SMMU.FS038：支持MSI类型中断。
- SMMU.FS039：支持两个TLB相关的开关：支持通过寄存器配置使能STE bypass场景TLB allocate，提升多VM场景下STE bypass 的性能；支持通过寄存器配置使能L1 miss，L2 hit，从L2 allocate到L1的功能。
- SMMU.FS040：支持将ReadOnce转成ReadShared发给L3。
  - 支持该功能寄存器使能。
  - 支持发送的ReadShared Expack=0, Order=Request Order,需要返回ReadReceipt。
  - 在使能该功能的情况下，不支持配置sky_bypass_en或者sky_bypass_rd_force为1；不支持同时使能AxUser[FA]和AxUser[FNA]；不支持加速器发大于128Byte的读请求。
  - 在使能该功能的情况下，对于Userbypass/Global bypass的操作，需要确保memattr的一致性，也即不支持Noncache或者Device属性，但是SnpAttr=1的请求。

版权所有 © 海思技术有限公司

  - 如果没有L3，不支持该功能，需要将该配置关闭。
- SMMU.DFX001：支持SMMUv3的Performance Monitor功能。Performance Monitor包含一组PMCG，个64位的计数器（计满卷绕），可配置触发MSI中断，通过配置可以支持如下事件的统计功能，每类统计可以区分StreamID、非安全和安全属性：
  - SMMU的每64个时钟周期计数加1（不可区分StreamID和非安全属性）。
  - SMMU接收的命令总数。
  - TLB miss总数。
  - Context Cache miss总数。
  - Page Table Walk次数。
  - Context Bank fetch次数。
  - PCIe ATS请求的次数。
  - PCIe ATS translated请求次数。
  - SMMU SP口接收读命令。
  - SMMU TP口接收读命令。
  - SMMU SP口接收写命令。
  - SMMU TP口接收写命令。
  - SMMU SP口发出命令（不可区分StreamID）。
  - SMMU TP口发出命令（不可区分StreamID）。
  - SMMU SP口发出读命令（不可区分StreamID）。
  - SMMU TP口发出读命令（不可区分StreamID）。
  - SMMU SP口发出写命令（不可区分StreamID）。
  - SMMU TP口发出写命令（不可区分StreamID）。
  - SMMU L1 TLB接收命令。
  - SMMU L2 TLB接收命令。
  - SMMU L1 TLB Hit。
  - SMMU L2 TLB Hit。
  - SMMU SP口被retry的命令个数统计。
  - SMMU TP口被retry的命令个数统计。
- SMMU.DFX002：片内CPU core可通过总线的访问SMMU寄存器接口、片外JTAG接口可通过DJTAG BUS访问SMMU的寄存器接口。
- SMMU.DFX003：SMMU包含AXI接口状态监控寄存器，可以查询接口反压状态和被反压的MSTID。
- SMMU.DFX004：SMMU包含skyros_req_queue条目查询寄存器接口，可以查询sky_req_queue任意条目的TxnID，Mstid，StreamID，状态，地址，Skyros Size等信息。
- SMMU.DFX005：SMMU包含CMD Queue状态查询寄存器。
- SMMU.DFX006： SMMU包含正在处理命令统计寄存器。
- SMMU.DFX007：SMMU包含两个skyros队列中stall命令的Streamid，Txnid，NS信息查询。

版权所有 © 海思技术有限公司

16.3 Key Terms

表 16-1 Description of SMMU key terms

Terms	Description
Virtualization	虚拟化。
VMID	Virtual Machine ID。
ASID	Address Space ID。
Context, Context Bank	上下文，使用不同页表的Transaction称其有不同上下文（包括STE和Context Descriptor）。
PTW	Page Table Walk，SMMU从DDR中读取页表的操作。
Ring Bus	系统总线。
TLB	Translate Lookaside Buffer，SMMU内部缓存的页表信息，用于加速地址和属性转换。
DVMOp	Distributed Virtual Memory Operation，对SMMU发出的正常DVMOp为TLB无效和Sync操作。
Transaction	总线上的命令和数据传输。
CD	Context Descripter。
STE	Stream Table Entry。
VA	Virtual Address。
PA	Physical Address。
IPA	Intermediate Physical Address。
StreamID	数据流ID，标识transaction。
SubstreamID	子数据流ID，标识transaction，区分不同stage1转换。
S1_TTD	Stage1 Translation Table。
S2_TTD	Stage2 Translation Table。
MT	Memory Attribute。
RA	Read Allocate。
WA	Write Allocate。
TR	Transient。

版权所有 © 海思技术有限公司

Terms	Description
SH	Share Attribute。
TTB	Translation Table Base Address。

16.4 Compliance

The SMMU complies with, or implements, the specifications described in this section.

遵循“ARM System MMUv3 specification”。

16.5 功能描述

16.5.1 Stage1 和 Stage2 应用场景

Stage1完成VA到IPA的地址转换，Stage2完成IPA到PA的地址转换。

Stage1 only的场景应用于只存在一个OS，不需要用Stage2进行虚拟化。

Stage2 only的场景下，Hypervisor管理Stage2完成各个OS所属的IPA到PA的转换，而VA到IPA的地址转换和管理则由OS实现。

Stage1和Stage2同时使能的场景下，Hypervisor管理Stage2完成各个OS的IPA到PA的地址转换，而OS通过SMMU的Stage1完成各个Device的VA到IPA的地址转换。

16.5.2 SMMU 页表格式

SMMU可以和MMU使用相同的页表，也可以使用自建页表。

SMMU支持AArch64的 ARMv8-A长页表格式及AArch32的ARMv7-A LPAE长页表格式。页表格式详见“ARM Architecture Reference Manual”D4部分和G4部分。

16.5.3 StreamID 的组成

Stream ID = {SubstreamID, Request ID};

不同的StreamID可以根据需要映射相同或者不同的VMID+ASID。VA+VMID+ASID 决定了最终的PA，如果想要映射不同的PA，这三个变量中至少有一个不一样。

16.5.4 Configuration 查找

SMMU的转换配置信息放在内存中，STE中存放了地址转换的初始信息，包括决定是否进行Stage1或者Stage2转换，或者Stage1+Stage2转换，以及怎样查找Stage1和Stage2配置信息，STE中还包含了Stage2 的转换信息，Context Descriptor 包含了Stage1的转换信息。所以SMMU中总共有两种不同的结构，第一种是配置结构，将StreamID和转换页表，转换配置信息对应起来，第二种是转换页表结构，用来进行Stage1（VA到IPA）和Stage2（IPA到PA）转换。

版权所有 © 海思技术有限公司

16.5.4.1 Stream Table Entry 的查找

输入命令首先要查找STE（Stream Table Entry），STE中包含了Stage2的转换信息及CD（包含Stage1转换的配置信息）的基地址STE.S1ContextPtr和CD的格式信息。

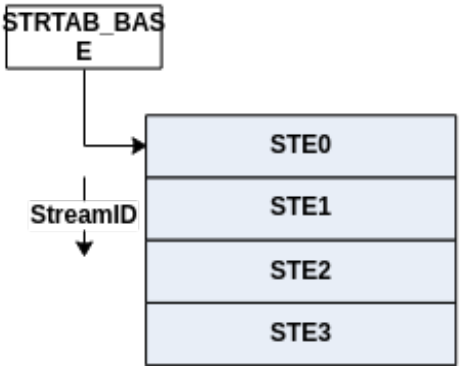
STE有两种格式，一种是Linear Stream Table，一种是2-level Stream table。与配置STE格式相关的信息有：首先SMMU_（S）_IDR0.ST_LEVEL决定SMMU是否支持2-level STE，在硬件支持的条件下，配置SMMU_（S）_STRTAB_BASE_CFG.FMT，决定了STE是linear还是2-level。

Linear Stream Table是一组连续的STE，通过SMMU_（S）_STRTAB_BASE.ADDR[47:6]作为基地址，StreamID作为索引来查找，每个Stream Table Entry占用64B的存储空间，故StreamID递增1时，Stream Table Entry地址递增64。

2-level Stream Table包含两级Table，第一级Table，即STD（Stream Table Descriptor）包含了指向二级STE的基地址STD.L2Ptr和表征第二级STE数目的STD.Span，第二级STE是Linear Stream Table。查找第一级Table方法为，SMMU_（S）_STRTAB_BASE.ADDR作为基地址，StreamID[n:SPLIT]作为索引，n为Stream的最高bit，SPLIT通过SMMU_（S）_STRTAB_BASE_CFG.SPLIT进行配置。第二级STE Table是由第一级STD中的STD.L2Ptr作为基地址加上StreamID[SPLIT-1:0]作为偏移进行索引的。

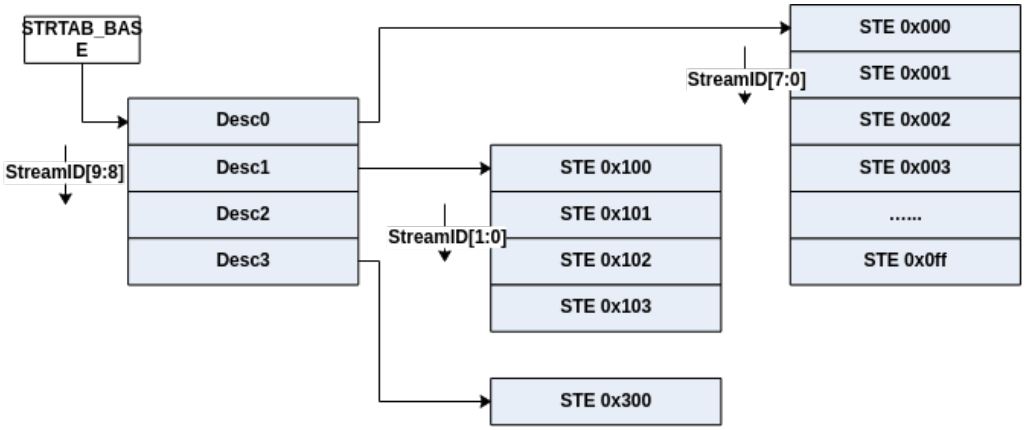
寄存器所配置的SPLIT参数决定了二级Table的最大值，可能的取值为6,8,10.对应的二级Table大小为4kByte，16kByte，64kByte。以SPLIT=6为例，表示一个STD所指向的二级Table最大的entry数为2^6个，每个二级Table占有64Byte空间，所以二级Table占有的最大空间即为4KB。STD中的SPAN决定了该STD所指向的二级Table实际entry数目为2^（SPAN-1）。所以，SPAN与SPLIT之间必须满足的关系是，StreamID[SPLIT-1:0]<2^（SPAN-1）。当不满足次条件时，会产生C_BAD_STREAMID的Event。

图 16-2 Linear Stream Table



版权所有 © 海思技术有限公司

图 16-3 2-level Stream Table with SPLIT=8

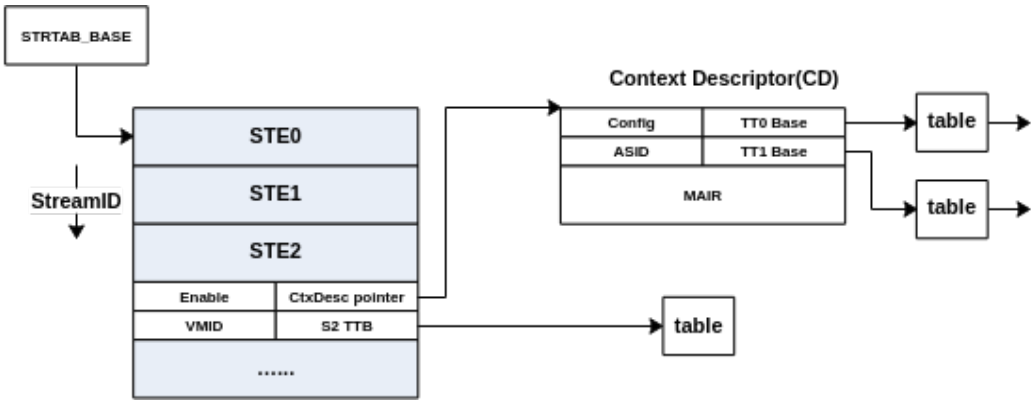


16.5.4.2 Context Descriptor 的查找

CD包含Stage1的转换配置信息，包括Stage1转换的也表基地址等。CD存在三种可能的格式：一个CD，直接由STE中的指针定位；一级linear CD，通过STE的指针作为基地址，SubstreamID[S1CDMax-1:0]作为索引定位到CD；二级CD。通过STE的指针作为基地址，SubstreamID[S1CDMax-1:n]作为索引定位到第一级CD，再由第一级CD中的基地址加SubstreamID[n-1:0]作为索引定位到二级CD。其中n的可能取值为6和10，分别表示4kByte和64kByte二级CD。

STE中的S1CDMax和S1Fmt共同决定了该STE对应的CD的格式：当S1CDMax=0时，表示只有一个CD，则忽略S1Fmt；当S1CDMax不为0且S1Fmt=0时，表示一级linear CD；当S1CDMax不为0且S1Fmt=1时，表示二级CD，且二级CD的大小为4kByte；当S1CDMax不为0且S1Fmt=2时，表示二级CD的大小为64kByte。

图 16-4 一级 STE 和单个 CD 结构的 Context Bank



版权所有 © 海思技术有限公司

图 16-5 一级 STE 和 Linear Context Descriptor 结构

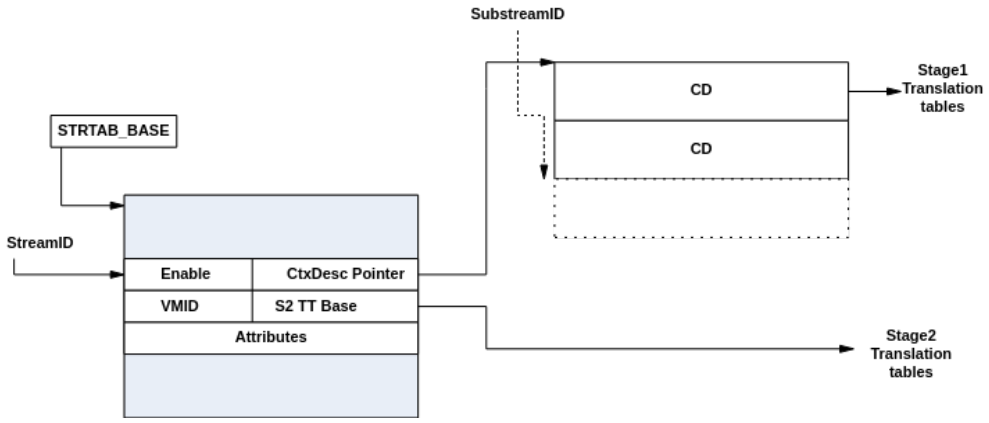
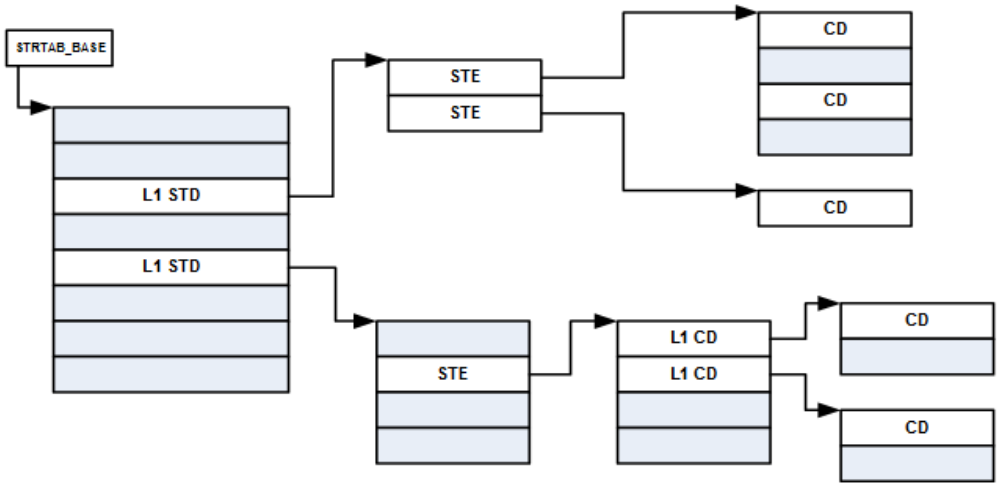


图 16-6 混合结构，2-level STE，各个 STE 指向的 CD 结构可以不同



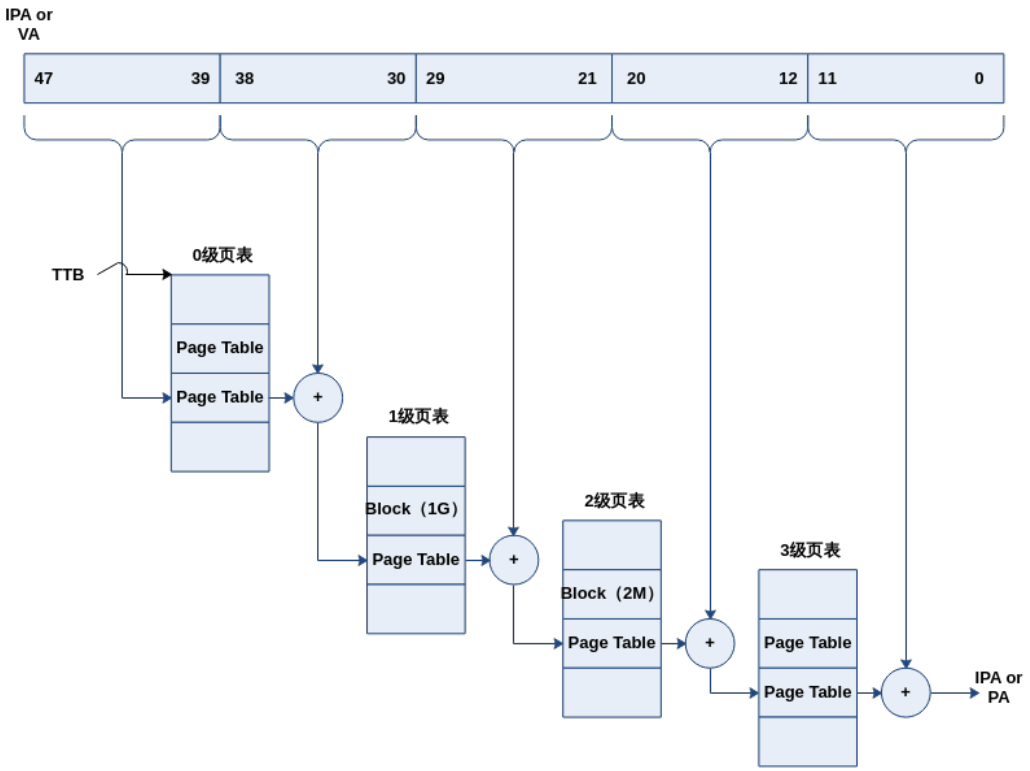
16.5.5 地址转换流程简述

16.5.5.1 单 Stage 地址转换流程

如图16-7所示，描述了Stage1地址转换流程，以4k粒度页表为例，Stage2转换同理。

版权所有 © 海思技术有限公司

图 16-7 4k 页表 Stage1 转换流程举例

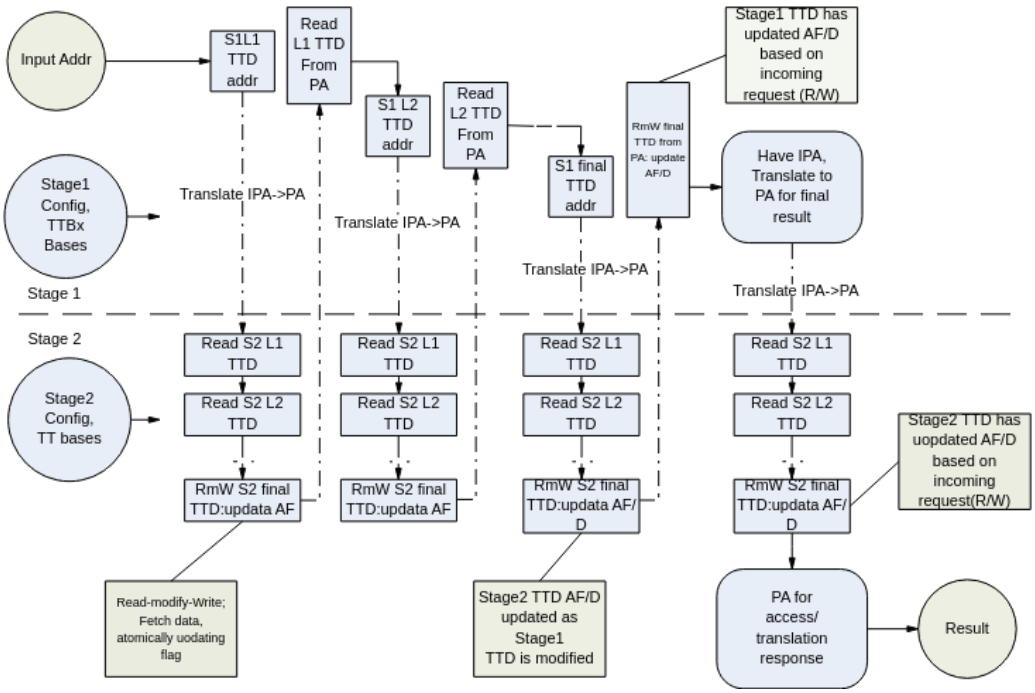


1. TTB和VA[47:39]组成获取Level0页表的地址PA。
2. 读取Level0页表中的Next-Level Table address，与VA[38:30]组成获取Level1页表的地址PA。
3. 读取Level1页表中的Next-Level Table address，与VA[29:21]组成获取Level2页表的地址PA。
4. 读取Level2页表中的Next-Level Table address，与VA[20:12]组成获取Level3页表的地址PA。
5. 读取Level3页表中的Output Address，与VA[12:0]组成获取最后的转换地址。

版权所有 © 海思技术有限公司

16.5.5.2 Stage1+Stage2 转换流程

图 16-8 Stage1+Stage2 地址转换流程



如图16-8所示，简单描述了Stage1和Stage2都使能情况下的地址转换流程。值得注意的是，在Stage2使能的情况下，STE.ContextPtr，CD.TTBx和Stage1PTW获取的各级页表基地址都属于IPA空间。所以从实际物理内存中获取CD或页表时，必须先经过Stage2的地址转换，将IPA转换为PA。

以Stage1+ Stage2都有效场景为例，地址转换流程主要步骤为：

步骤1 通过SMMU_STRTAB_BASE和StreamID获取STE。

步骤2 对STE.ContextPtr进行Stage2的地址转换，得到CD指针的物理地址（Stage2的PTW格式和起始Level由STE.S2T0SZ,STE.S2SL0,STE.S2TG决定）。

步骤3 通过CD指针和SubstreamID获取CD。

步骤4 用CD.TTBx和VA组成Stage1转换Start Level的IPA。

步骤5 对IPA进行Stage2转换，获取页表的物理地址。

步骤6 读取页表中的Next-Level Table address，与VA组成获取下一级转换页表的IPA。

步骤7 重复步骤5和步骤6，直到最后一级PTW，完成VA到IPA的转换。

步骤8 通过Stage2完成IPA到PA的转换，地址转换结束。

----结束

16.5.6 全流程简述

在Configuration查找和功能描述章节中，分别描述了Configuration数据结构的查找和地址转换的流程。SMMU的全流程可划分为这两个主要步骤。

版权所有 © 海思技术有限公司

如图16-9，SMMU收到上游Scheduler发来的AXI读写命令后，先进行Configuration的查找，再进行translation查找。

步骤1 通过StreamID和相应寄存器的值获取STE。通过STE判断是否使能了该StreamID，是否使能了Stage1和Stage2。

步骤2 如果使能，通过SubStreamID和STE获取Context Descriptor。至此，Configuration查找完毕。

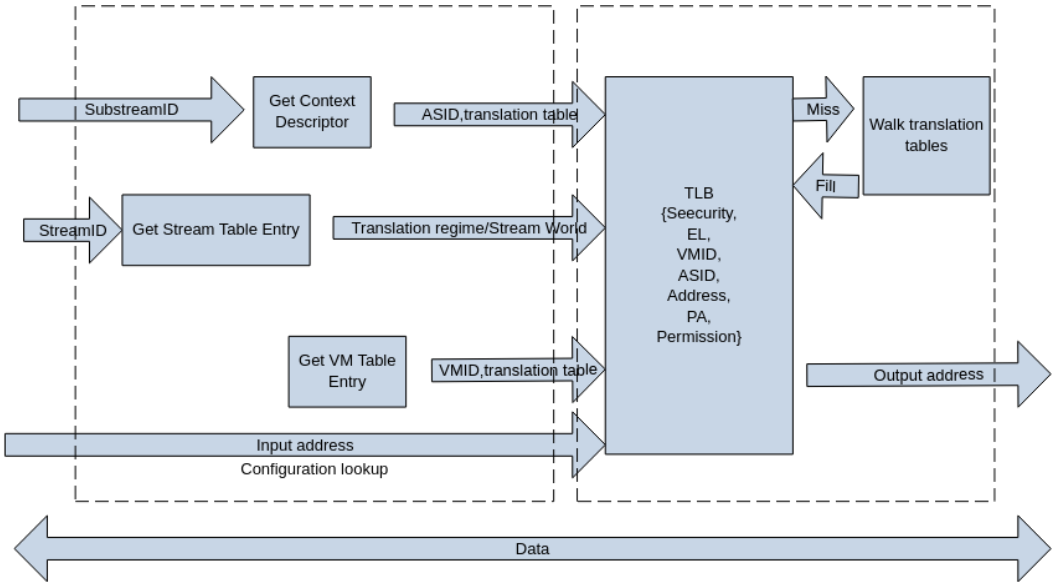
SMMU通过Configuration查找获取了Translation所需的各个Stage的Page base pointer，ASID,VMID和Page Table Walk所需的属性

步骤3 SMMU进行地址转换：首先对TLB进行查找，在TLB miss的情况下，通过page table walk流程进行translation table lookup和地址转换。

步骤4 属性转换详见“**属性转换**”、“**Configuration查找**”至“**Commad Queue配置与功能**”。

----结束

图 16-9 Configuration and translation lookup sequence



16.5.7 属性转换

SMMU在不同的地址转换过程中，属性转换也会不同。

SMMU的输入命令的memory属性为AXI4协议的Awcache或Arcache，在后续转换中统一为Axcache。其中，如果是读命令，Axcache就是Arcache；如果是写命令，Axcahe为{Awcache[2],Awcache[3],Awcache[1:0]}。

SMMU转换后的请求命令中Memory属性由Skyros MemAttr域和Skyros SnpAttr域体现。

版权所有 © 海思技术有限公司

表 16-2 Skyros MemAttr

MemAttr[3:0]	Description
[0]	Early Write Acknowledge bit. 0：Early Write Acknowledge disallowed 1：Early Write Acknowledge allowed
[1]	Memory type bit. 0：Normal Memory Type 1：Device Memory Type
[2]	Cacheable bit. 0：Non-cacheable 1：Cacheable
[3]	Allocate hint. 0：Do not Allocate 1：Allocate

表 16-3 Skyros SnpAttr

SnpAttr[0]	Description
[0]	Snpable bit. 0:Non-snpable 1:Snpable

Skyros MemAttr只有在Early Write Acknowledge allowed且Normal Memory Type时才可能Cacheable，只有在Cacheable时才可能Allocate。Skyros SnpAttr只有在Skyros MemAttr为Cacheable时才可能Snpable。

SMMU Memory属性配置中Write Through（即Normal Memory Type，Cacheable Non-Bufferable）都会被降级处理为Non-Cacheable（即Normal Memory Type，Non-Cacheable Bufferable）。

16.5.7.1 属性叠加规则

对于任何属性叠加，叠加输出为输入中的最强属性。各属性强弱等级如表16-4所示。

表 16-4 属性强弱表

-	**Weakest** **Strongest**						
MemAttr	Normal-WB	Normal-WT	Normal-NC	Device-GRE	Device-nGRE	Device-nGnRE	Device-nGnRnE

版权所有 © 海思技术有限公司

-	**Weakest** **Strongest**
Shareable	NSH ISH OSH
Allocate	Allocate No-allocate
Transient	Non-transient Transient

16.5.8 Command Queue

Command Queue用于软件与SMMU的硬件交互，软件写命令到Command Queue，SMMU从Command Queue中读取命令处理。Command Queue采用了circular Queue结构。在有效的index基础上，多使用一个bit来表示当前的index是否wrap。当consumer的index（下文成为CONS.RD）和producer的index（下文成为PROD.WR）相同，并且wrap位相同时，表示当前Queue为空；当CONS.RD和PROD.WR相同，但wrap位相反时，表示当前Queue为满。Circular的特点要求CONS和PROD两个指针只能递增，不能向后递减。当index达到最大值时，再递增则index归零，wrap位取反即可。

Circular Queue有空间的条件是：producer和consumer的index不同，或者index相同但wrap不同。

CONS与PRODS两个指针分别由consumer和producer管理，对于Command Queue，consumer为SMMU，producer为软件，CONS指向的是下一次会读到的entry，PROD指向的是下一次会写到的entry。

16.5.8.1 支持的 Command 类型

- CMD_CFGI_STE, CMD_CFGI_STE_RANGE;
- CMD_CFGI_CD, CMD_CFGI_CD_ALL, CMD_CFGI_ALL;
- CMD_TLBI_NH_ALL, CMD_TLBI_NH_ASID, CMD_TLBI_NH_VA, CMD_TLBI_NH_VAA;
- CMD_TLBI_EL3_ALL, CMD_TLBI_EL3_VA,
- CMD_TLBI_S12_VMALL;
- CMD_TLBI_S2_IPA;
- CMD_TLBI_NSNH_ALL;
- CMD_ATC_INV;
- CMD_PRI_RESP;
- CMD_RESUME;
- CMD_STALL_TERM;
- CMD_SYNC;
- CMD_PREFETCH_CONFIG;
- CMD_PREFETCH_ADDR。

版权所有 © 海思技术有限公司

16.5.8.2 Command 格式

图 16-10 Command 数据结构 1

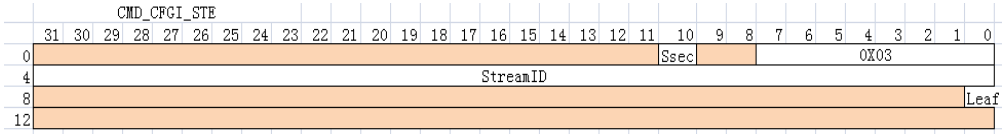


图 16-11 Command 数据结构 2

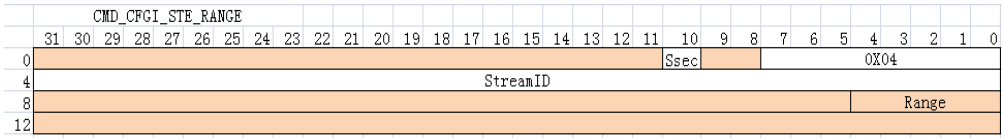


图 16-12 Command 数据结构 3

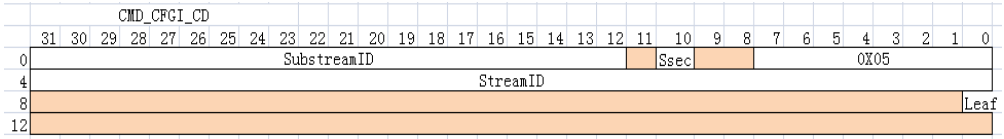


图 16-13 Command 数据结构 4

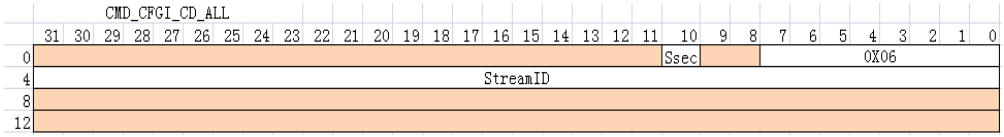


图 16-14 Command 数据结构 5

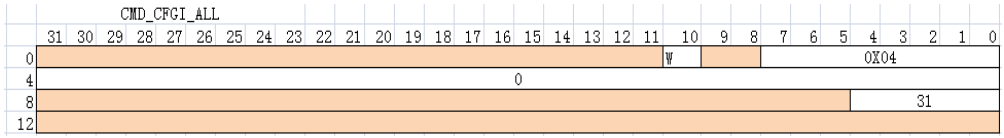


图 16-15 Command 数据结构 6

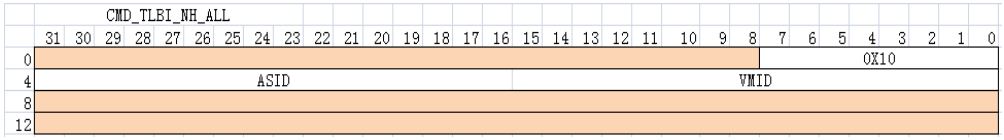
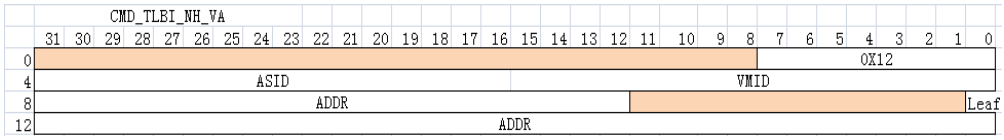


图 16-16 Command 数据结构 7



版权所有 © 海思技术有限公司

图 16-17 Command 数据结构 8

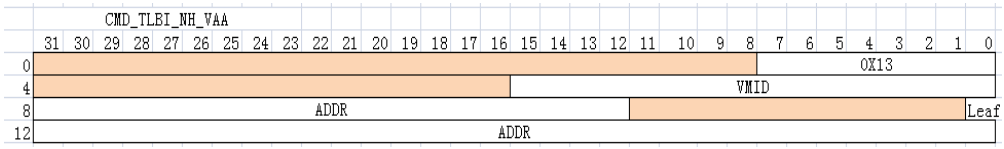


图 16-18 Command 数据结构 9

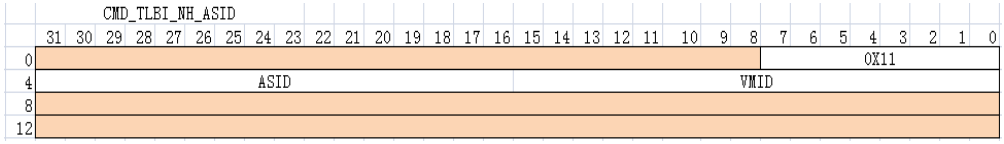


图 16-19 Command 数据结构 10

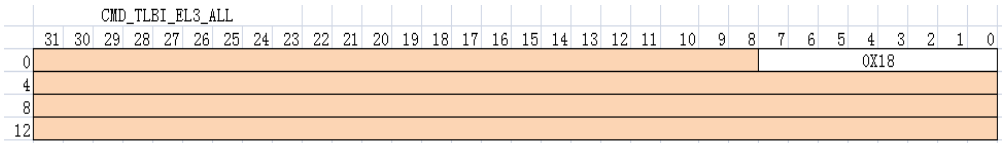


图 16-20 Command 数据结构 11

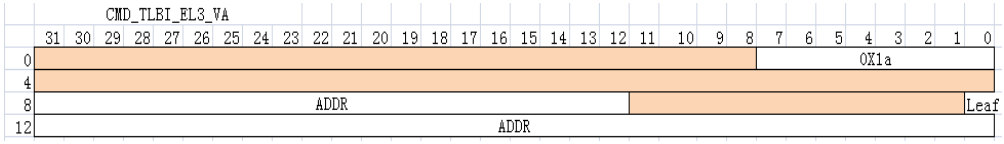


图 16-21 Command 数据结构 12

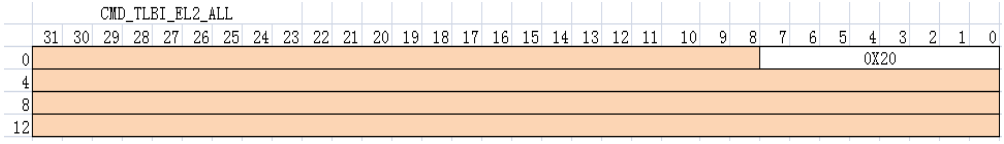


图 16-22 Command 数据结构 13

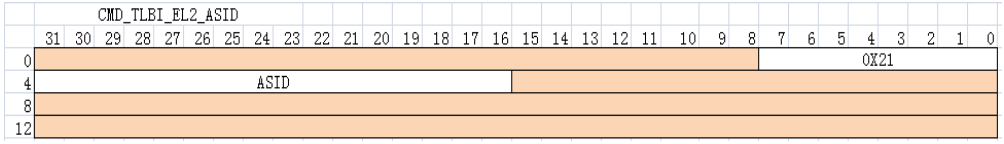


图 16-23 Command 数据结构 14

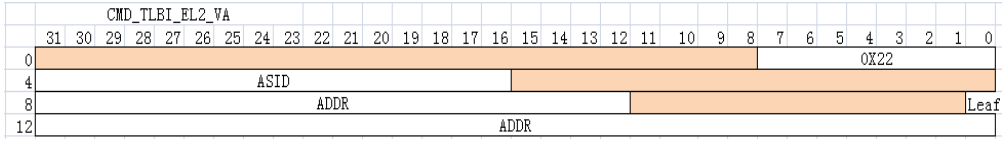
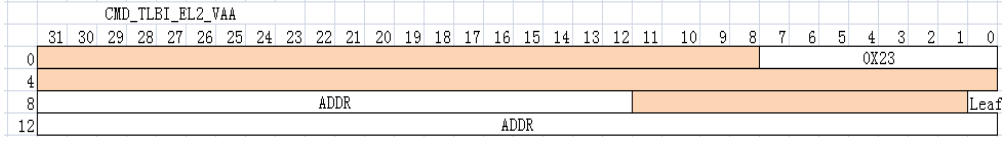


图 16-24 Command 数据结构 15



版权所有 © 海思技术有限公司

图 16-25 Command 数据结构 16

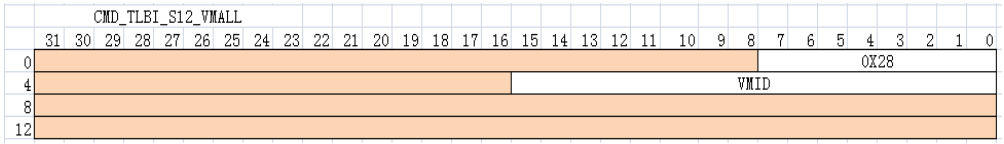


图 16-26 Command 数据结构 17

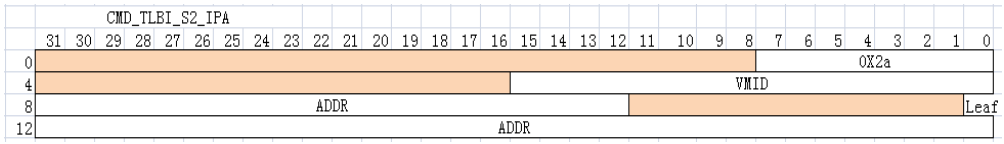


图 16-27 Command 数据结构 18

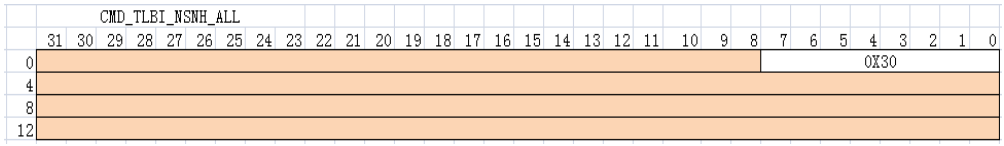


图 16-28 Command 数据结构 19

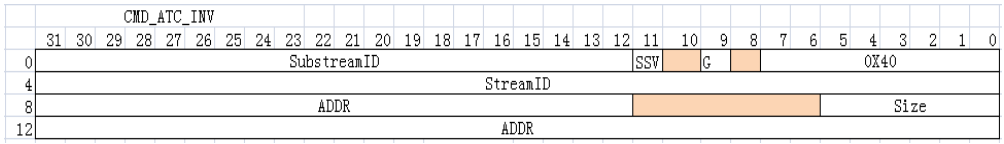


图 16-29 Command 数据结构 20

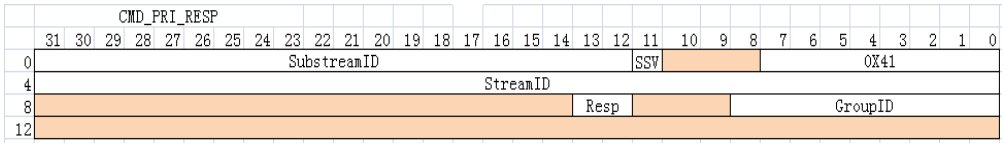


图 16-30 Command 数据结构 21

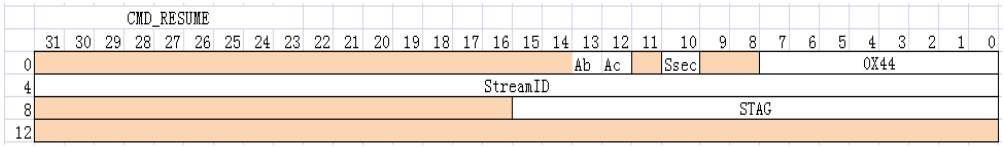


图 16-31 Command 数据结构 22

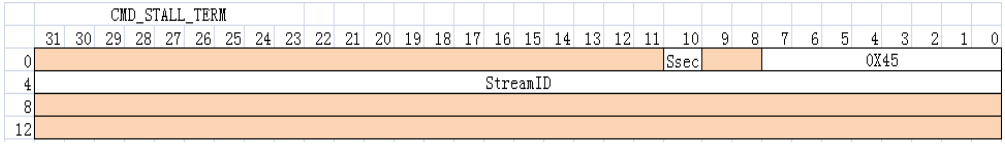
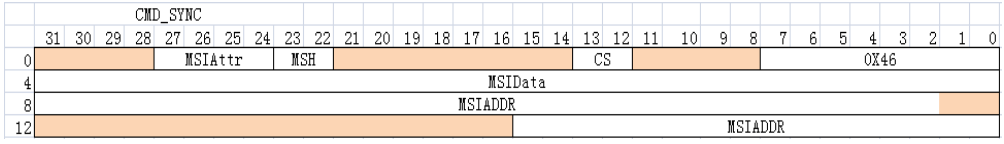


图 16-32 Command 数据结构 23



版权所有 © 海思技术有限公司

16.5.8.3 Command 域段

1. SSec：安全StreamID。在non-secure CMD Queue出现SSec=1是非法的。
2. SSV：SubstreamID valid。
3. Leaf：Leaf=1表示invalidate相关的命令只会影响最后一级CD或STE，不影响第一级。
4. Range：在invalidate相关的命令中，表示作用的范围。
5. Global：ATC invalidate命令中的Global flag。
6. GroupID：CMD_PRI_RESP中用来匹配page request。
7. Resp：CMD_PRI_RESP中用来表示Resp类型：Success，Fail，Deny。
8. STAG：CMD_RESUME中用来匹配stalled transaction。
9. Action：CMD_RESUME中用来表示stalled transaction是Retry还是Terminate。
10. Abort：CMD_RESUME中用来表示Terminate的行为，Abort或RAZ/WI。
11. ComplSignal：CMD_SYNC中用来表示CMD_SYNC完成后如何通知软件。

16.5.9 Event Queue

Event Queue用于SMMU发生软件配置错误的状态信息记录，SMMU将配置错误信息写入到Event Queue中，软件通过读取Event Queue获得配置错误信息并进行配置错误处理。Event Queue采用了circular Queue结构。在有效的index基础上，多使用一个bit来表示当前的index是否wrap。当consumer的index（下文成为CONS.RD）和producer的index（下文成为PROD.WR）相同，并且wrap位相同时，表示当前Queue为空；当CONS.RD和PROD.WR相同，但wrap位相反时，表示当前Queue为满。Circular的特点要求CONS和PROD两个指针只能递增，不能向后递减。当index达到最大值时，再递增则index归零，wrap位取反即可。

Circular Queue有空间的条件是：producer和consumer的index不同，或者index相同但wrap不同。

CONS与PRODS两个指针分别由consumer和producer管理，对于Event Queue，consumer为软件，producer为SMMU，CONS指向的是下一次会读到的entry，PROD指向的是下一次会写到的entry。

16.5.9.1 支持的 Event 类型

- C_BAD_STREAMID；
- C_BAD_SUBSTREAMID
- F_STE_FETCH；
- C_BAD_STE；
- F_BAD_ATS_TREQ；
- F_CD_FETCH；
- C_BAD_CD；
- F_WALK_EABT；
- F_TRANSlATION；
- F_ACCESS；
- F_PERMISSION；

版权所有 © 海思技术有限公司

16.5.9.2 Event 格式

图 16-33 Event 数据结构 1

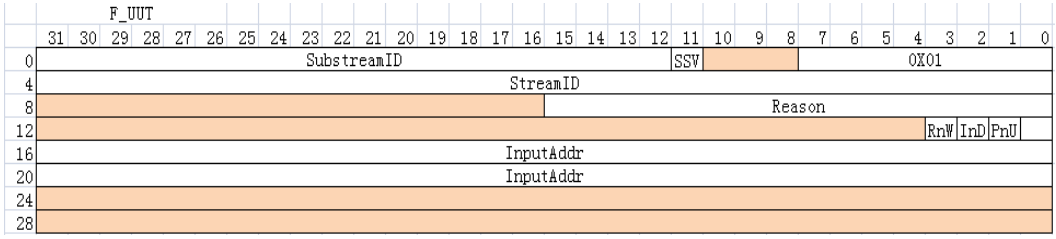


图 16-34 Event 数据结构 2

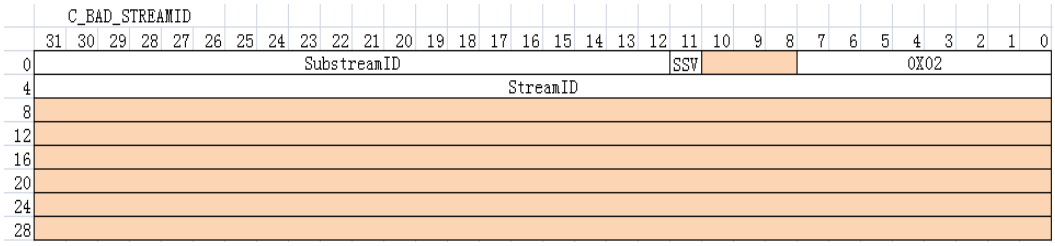


图 16-35 Event 数据结构 3

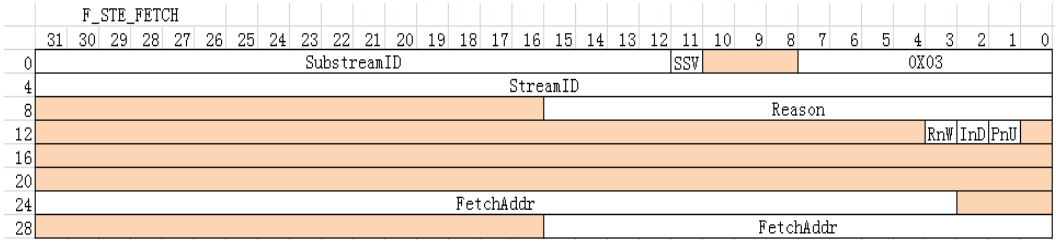


图 16-36 Event 数据结构 4

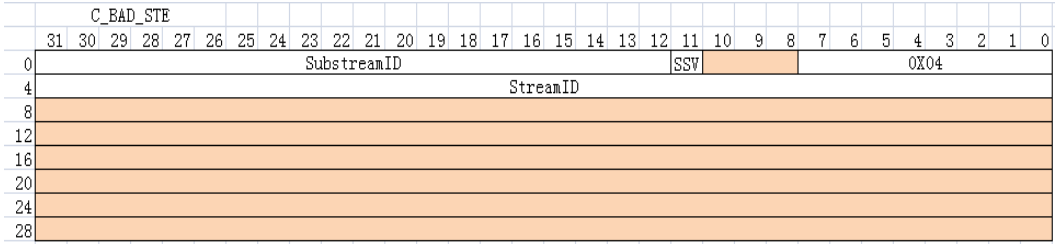
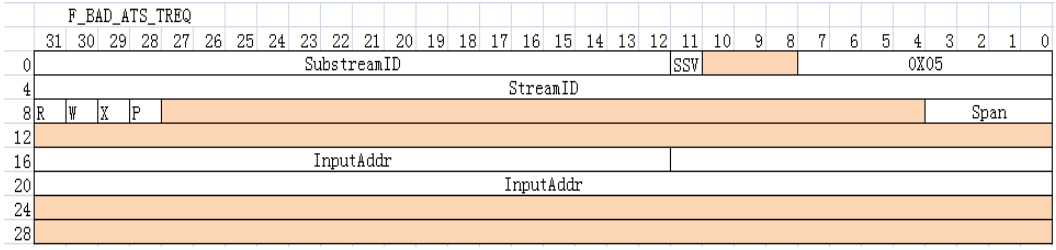


图 16-37 Event 数据结构 5



版权所有 © 海思技术有限公司

图 16-38 Event 数据结构 6

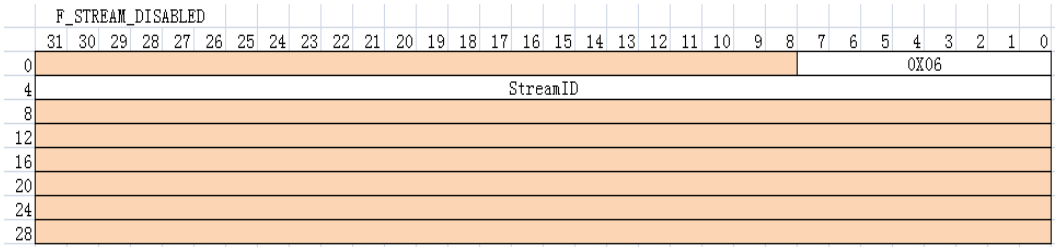


图 16-39 Event 数据结构 7

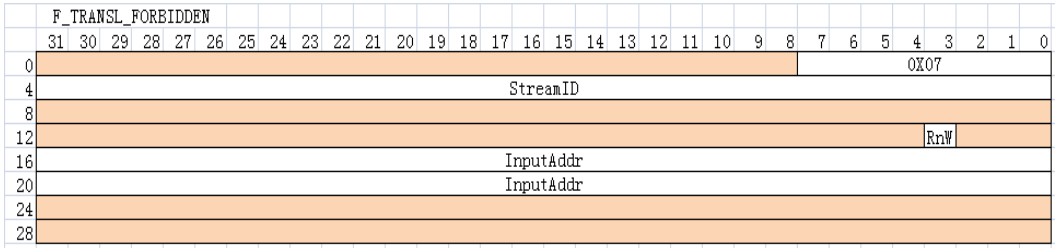


图 16-40 Event 数据结构 8

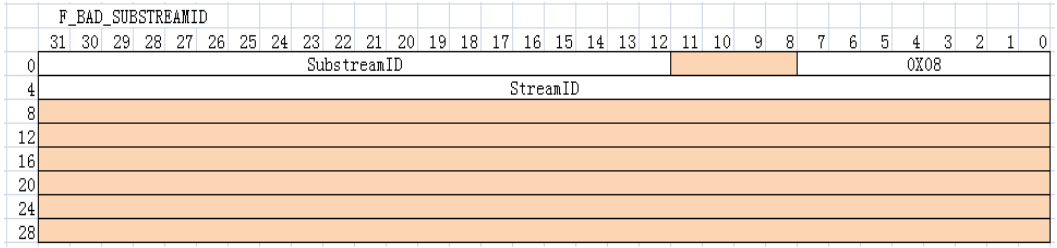


图 16-41 Event 数据结构 9

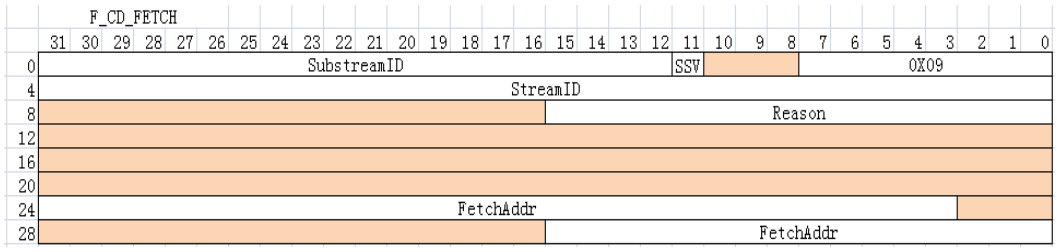
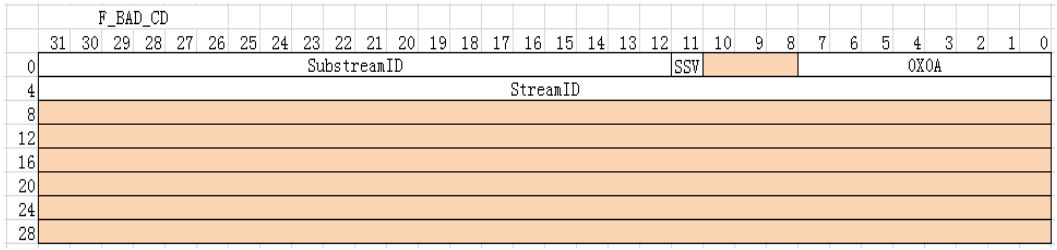


图 16-42 Event 数据结构 10



版权所有 © 海思技术有限公司

图 16-43 Event 数据结构 11

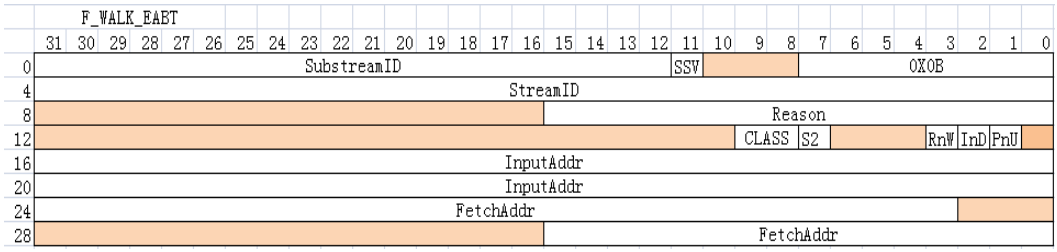


图 16-44 Event 数据结构 12

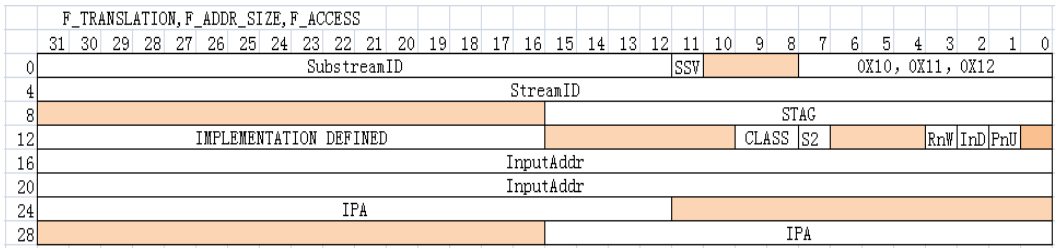


图 16-45 Event 数据结构 13

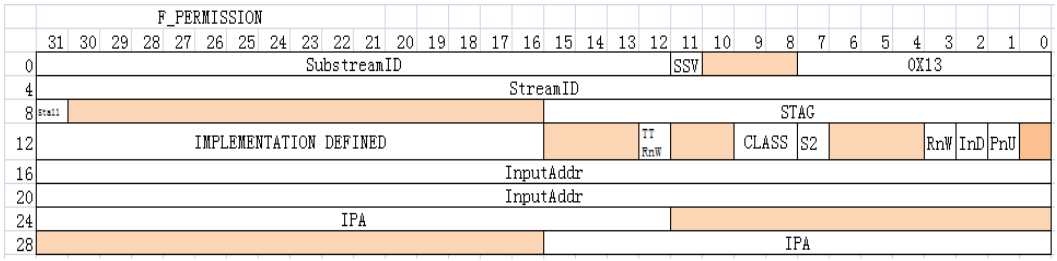


图 16-46 Event 数据结构 14

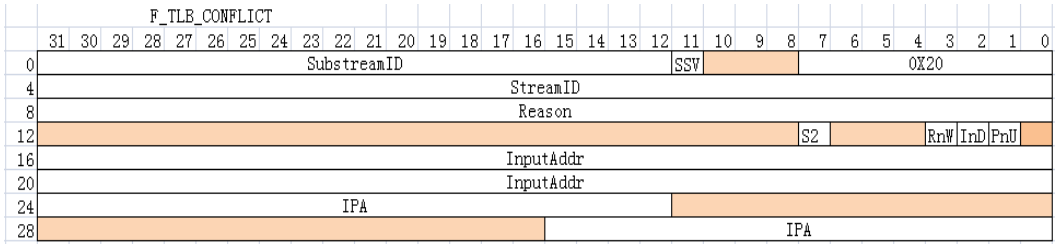
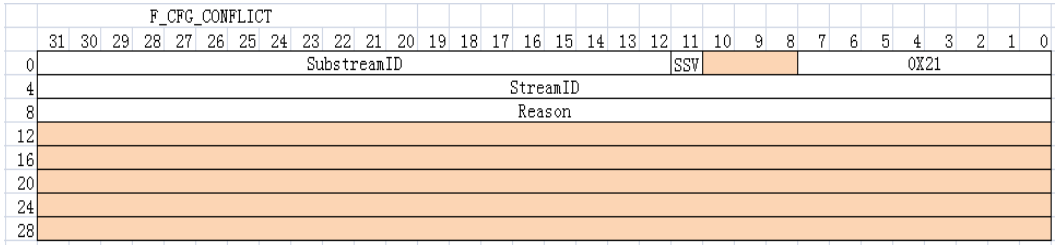
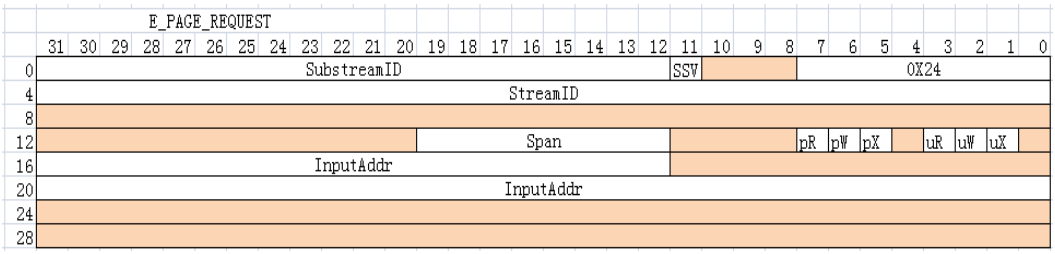


图 16-47 Event 数据结构 15



版权所有 © 海思技术有限公司

图 16-48 Event 数据结构 16



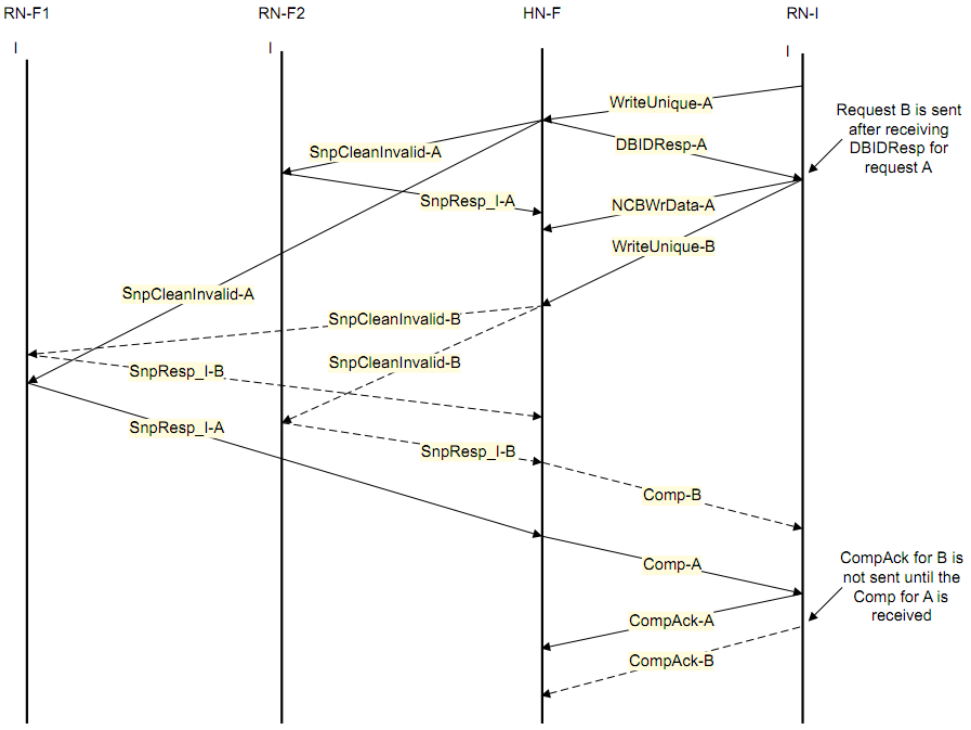
16.5.9.3 Event 域段

1. SSV：SubstreamID valid。
2. CLASS：00表示CD Fetch，01表示TTD fetch，10表示Input Address Caused Fault。
3. RnW：Read/Write。
4. PnU: Priviledge/Unpriviledged。
5. InD: Instruction/Data。
6. S2: S2=1表示由于Stage2引起的Fault。
7. pR、pW、pX、uR、uW、uX：E_PAGE_REQUEST中用来表示预期的read/write/execute权限。

16.5.10 Streaming Ordered Write

IO-Cache为SMMU为了满足PCIE SO写操作之间保续而实现的功能。

图 16-49 Streaming Ordered WriteUniques transaction flow



版权所有 © 海思技术有限公司

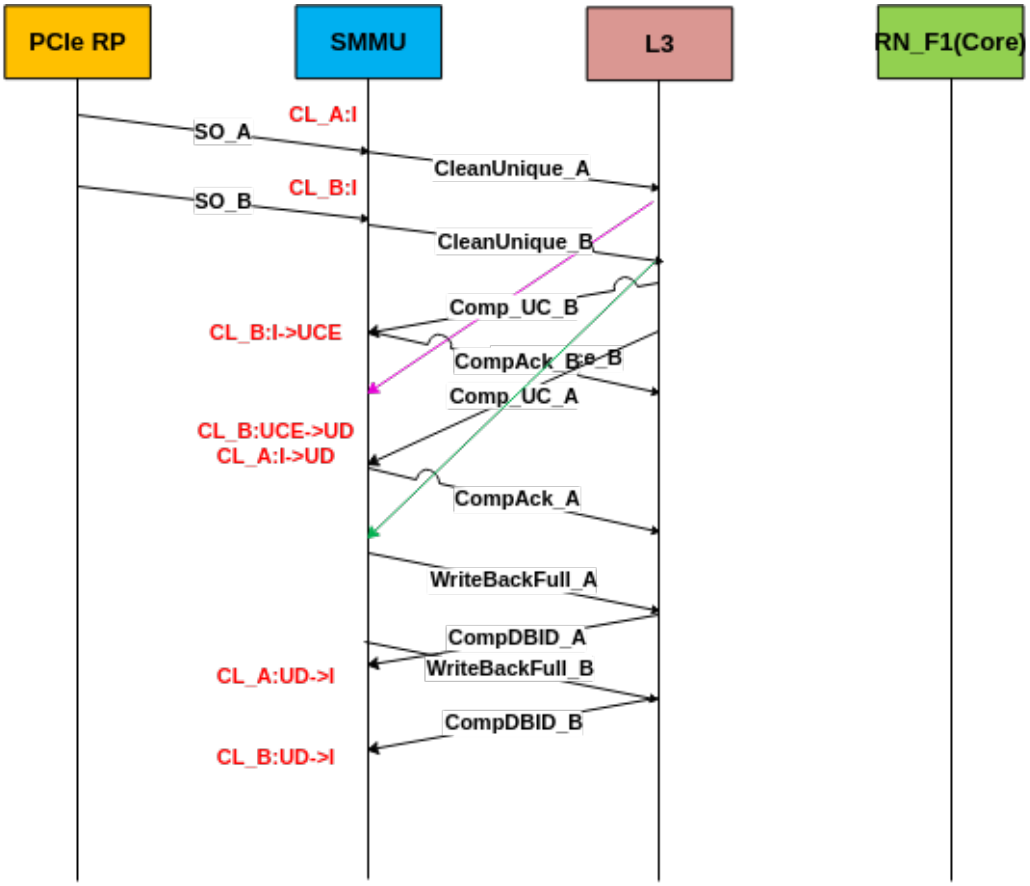
16.5.10.1 使用规则

1. SO命令类型，非Atomic、TH类型的Norder的写操作；如果属性为C+S（不区别byp和非byp）的SO操作，可以走Stream Write优化流程。其他类型不走Stream Write流程。
2. 若SO操作满足Stream Write条件：在前面所有满足依赖关系的SO或RO收到DBID后，发出WriteUnique，ExpAck拉高；在前面所有满足依赖关系的SO或RO收到COMP后，发出CompAck。
3. 若SO操作不满足Stream Write条件：在前面所有满足依赖关系的SO或RO收到COMP后，发出WriteNoSnp。
4. StreamWrite模式下，所有的RO和SO都必须去到HA（非Atomic写操作），DEC都等于NCA，Atomic操作去L3。
5. WriteUnique必须是Request Order。
6. StreamWrite的CompAck，Rsv.dec=STREAMWR，给PA。
7. StreamWrite模式下DBID和Comp的源头不一样，需要记录两个DBID和SrcID。

16.5.11 IO-Cache

IO-Cache为SMMU为了满足PCIE SO写操作之间保续的性能而实现的功能。

图 16-50 IO-Cache



版权所有 © 海思技术有限公司

16.5.11.1 使用规则

1. SO命令类型，非Atomic、TH类型的Norder的写操作；如果属性为C+S（不区别byp和非byp），skyros size不超过128B，发CU+WB；不满足上述条件的SO或RO操作都只会发WriteUnique或WriteNoSnp。
2. CleanUnique操作一定置位ExpAck，SMMU收到CompUC后返回（无其他约束）。
3. SMMU在收到CompUC之前收到Snp，直接回SnpResp，SMMU在发出CompAck后收到Snp，返回SnpResp，回到I态，重新发送CleanUnique（HA收到CompAck之后才发Snp）；SMMU在发出WriteBack还没有收到CompDBID之间收到Snp，阻塞Snp，直到收到CompDBID之后再回SnpResp。
4. SMMU在发出WB时就可以向AXI回Bresp（可配置，可配置为收到compdbid后返回AXI的Bresp）。
5. 保续条件：同Master同SOID的RO-SO，SO-SO之间的写保续。
6. SO-SO之间：
   a. 若W0为C+S，W1为C+S，W1必须等到W0发出WB后才能发WB；同地址的SO保序，前一个SO的WB收到CompDBID回到I态后，后一个才能发出CU。
   b. 若W0为C+S，W1为非C+S，W1必须等到W0发出WB后才能发Write。
   c. 若W0为非C+S，W1为C+S，W1必须等到W0收到Comp才发WB。
   d. 若W0为非C+S，W1为非C+S，W1必须等到W0收到Comp才能发Write。
7. RO-SO之间：
   a. SO只和早于自己的RO有保续关系，和晚于自己的RO操作没有保续关系。
   b. 若SO为C+S的，则必须等到所有RO收到Comp才能发W0的WB。
   c. 若SO为非S的，则必须等到所有RO收到Comp才能发Write。
8. 允许是Stash的，带有Stash的WriteBack，在SMMU配置寄存器使能spill的情况下，可以发NCA。其他情况下的WriteBack发FH。
9. IO cache收到的Snp，SrcID来自于PA，不能默认SrcID与HA一致，要单独记录。
10. 不支持长atomic（ReadUnique+WriteBack）访问CCIX SA地址空间。

16.5.12 Fault 和中断

16.5.12.1 Fault 类型

SMMU有两种类型的Fault，一种是配置错误及Command格式错误等，记录在Event Queue中，一种是将Event写入Event Queue过程中发生的错误，记录在SMMU_GERROR寄存器中。Fault类型及Event格式详见《ARM System MMUv3 specification》文档 chapter 9。

16.5.12.2 Fault 处理方式

1. Terminate模式

   SMMU停止引发Fault的transaction的转换，读操作会返回0，写操作会被忽略。SMMU会根据STE[R]来决定是否上报Event。

2. Stall模式

   SMMU实现了Fault命令stall功能，STE[STALLD]及CD[S]域段来控制是否允许stage1 Fault命令stall。如果判断命令被允许为stall，SMMU会把引发Fault的

版权所有 © 海思技术有限公司

transaction暂停，然后stall。被stall的transaction之后会根据CMD_RESUME及CMD_STALL_TERM命令来决定stall命令retry还是terminate。只有4中Fault类型允许被stall，分别是F_TRANSLATION，F_ADDR_SIZE，F_ACCESS及F_PERMISSION。

16.5.12.3 中断处理

系统中每个SMMU上报6个中断，安全和非安全中断各3个，分别为：

1. Event中断：当Event Queue 记录Event时产生中断。Global Error中断及CMD_SYNC命令完成中断。
2. Global Error中断：Global Error记录在SMMU_GERROR寄存器中时上报中断。
3. CMD_SYNC中断：Command Queue中CMD_SYNC命令完成时产生中断。

16.5.13 PCIe 的 Atomic 操作

SMMU支持PICe设备发出的Atomic请求。PCIe的Atomic操作类型在AWUSER中的ATYPE域段体现。PCIe固定从AW通道发送Atomic命令。SMMU收到Atype不为0的写请求，认为是有效的Atomic命令，经过转换后，向下游发送标准的CHIE Atomic请求。在收到下游返回的CompData后，SMMU通过AXI 的R通道向PCIE返回读数据，结束一次Atomic请求。

16.5.13.1 Atomic 命令类型

根据SMMU与PCIe协之间的协商结果，所支持的Atomic命令类型如表16-5所示。其中，数据量指的是outbound数据量。对于FetchAdd和Swap这两种类型而言，outbound size等于inbound size。而对于CAS类型，由于outbound包括CompareData和SwapData两个部分，因此outbound size是inbound size的两倍。

根据CHIE协议的规定，AXI的SIZE字段应当设置为等于outbound size。在CAS类型的非对齐场景中，尽管AXI的ADDR需要与inbound size对齐，但并不一定对齐outbound size。此时，AXI的ADDR应始终指向CompareData。然而，在这种非对齐场景下，wstrb的低位需要准确地反映SwapData的状态，这与AXI标准协议的要求不符，因为标准协议规定在非对齐情况下，低位的wstrb的值应当应当全部为0。

表 16-5 Atomic 命令类型

PCIe端命令名称	CHIE命令	CHIE Opcode	ATYPE	数据量（Outbound）
FetchAdd	AtomicLoad.LDADD	0x30	1	4B/8B
Swap	AtomicSwap	0x38	3	4B/8B
CAS	AtomicCompare	0x39	2	8B/16B/32B

16.5.13.2 其他限定

要求Atomic请求固定NO=1，SO=0，CTYPE=0或1。

版权所有 © 海思技术有限公司

16.5.14 RAS

SMMU遵循ARM架构，支持Fault handling Interrupt和Error Recovery Interrupt中断上报。

16.5.14.1 Correct Error

当SMMU内部Memory发生1bit ECC Error的时候，会出发Correct Error，并在对应使能位的情况下上报Fault handling Interrupt。

16.5.14.2 Unrecoverable Uncorrected Error

当SMMU内部Memory发生2bit及以上 ECC Error的时候，会出Unrecoverable Uncorrected Error，并在对应使能位的情况下上报Fault handling Interrupt及Error Recovery Interrupt。

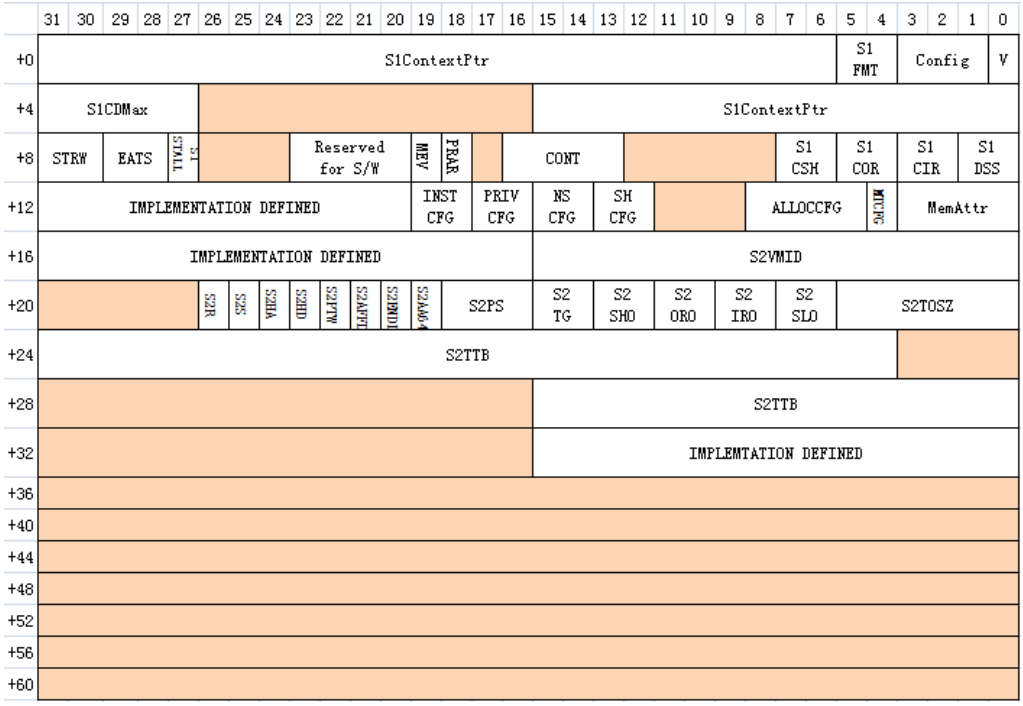
16.5.14.3 Recoverable Uncorrected Error

当SMMU发生F_WALK_EABT，F_STE_FETCH，F_CD_FETCH等对DDR读取或者写操作时放回的ABORT时产生Recoverable Uncorrected Error，并在对应使能位的情况下上报Fault handling Interrupt及Error Recovery Interrupt。

16.6 数据结构

16.6.1 Stream Table Entry 配置描述

图 16-51 STE 数据结构



版权所有 © 海思技术有限公司

表 16-6 Level1 Stream Table Descriptor 数据结构

Field Name	Position	Width	Description
Span	4:0	5	Span的域段说明。 [31:18]:Reserved; [17:1]:Level 2 array contains 2（Span-1）STEs; [0]:L2Ptr is invalid; the StreamIDs matching this descriptor are all invalid.
L2Ptr	47:5	43	Pointer to start of Level-2 array.
Reserved	63:48	16	Reserved.

表 16-7 Stream Table Entry 数据结构

Field Name	Position	Width	Description
V	0	1	Valid. 0:Structure contents are invalid,other STE fields are IGNORED; 1:Structure contents are valid.
Config	3:1	3	Stream configuration. 0xx:Report abort to device,no event recorded; 100:Bypass; 101:Stage 1 Context with Stage 2 Bypass; 110:Stage 2 Context; 111:Stage 1 Context with Stage 2 Context.
S1Fmt	5:4	2	S1ContextPtr points to.
S1ContextPtr	47:6	42	Stage1 Context Descriptor pointer bits[47:6],remainder of bits 0.
Reserved	58:48	8	Reserved.
S1CDMax	63:59	5	Number of CDs pointed to by S1ContextPtr, 2CDMax.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
S1DSS	65:64	2	When substreams are enabled （CDMax != 0）, this field determines the behaviour of a transaction that arrives without an associated substream. 00:Transaction is terminated; 01:Transaction bypasses Stage 1; 10:Transaction is translated using the CD associated with Substream 0. Other transactions arriving from Substream 0 are terminated; 11:Reserved（behaves as 00）.
S1CIR	67:66	2	S1ContextPtr memory Inner Region attribute. 00:Normal, non-cacheable; 01:Normal, Write-Back cacheable, read-allocate; 10:Normal, Write-Through cacheable; 11:Normal, Write-Back cacheable, no read-allocate.
S1COR	69:68	2	S1ContextPtr memory Outer Region attribute: 00:Normal, non-cacheable; 01:Normal, Write-Back cacheable, read-allocate; 10:Normal, Write-Through cacheable; 11:Normal, Write-Back cacheable, no read-allocate.
S1CSH	71:70	2	S1ContextPtr memory Shareability attribute. 00:Non-shareable; 01:Reserved（behaves as 00）; 10:Outer shareable; 11:Inner shareable.
CONT	80:77	4	Contiguous Hint.
Reserved	81	1	Reserved.
PPAR	82	1	PRI Page request Auto Responses s. 0:Auto-generated responses on PRIQ overflow do not include a PASID TLP prefix; 1:Auto-generated responses on PRIQ overflow include a PASID TLP prefix.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
MEV	83	1	Merge Events arising from terminated transactions from this stream. 0:Do not merge similar Fault records; 1:Permit similar Fault records to be merged.
Res S/W	87:84	4	Reserved for software use, this field is IGNORED by the SMMU.
Reserved	90:88	3	Reserved.
STALLD	91	1	Stall disable: 0:Allow stalling Fault model for Stage 1（configured in CD）; 1:Disallow stalls to be configured（Faults terminate immediately）.
EATS	93:92	2	Enable ATS translation and traffic. 00:Disallow ‘Translated’ traffic to bypass the SMMU; 01:ATS Translation Requests are serviced by translating at all enabled stages of translation. ‘Translated’ traffic is allowed to bypass; 10:ATS Translation Requests are serviced using Stage 1 translations only; subsequent ‘Translated’ traffic is translated in the SMMU using Stage 2 translations; 11: Reserved（behaves as 00）.
STRW	95:94	2	Select the translation regime/exception level controlling this stream.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
MemAttr	99:96	4	Memory Attributes, When MTCFG=1 0x0: Device-nGnRnE; 0x1: Device-nGnRE; 0x2: Device-nGRE; 0x3: Device-GRE; 0x4: Reserved; 0x5: Normal outer non-cacheable, inner non-cacheable; 0x6: Normal outer non-cacheable, inner write-through; 0x7: Normal outer non-cacheable, inner write-back; 0x8: Reserved; 0x9: Normal outer write-through, inner non-cacheable; 0xA: Normal outer write-through, inner write-through; 0xB: Normal outer write-through, inner write-back; 0xC: Reserved; 0xD: Normal outer write-back, inner non-cacheable; 0xE: Normal outer write-back, inner write-through; 0xF: Normal outer write-back, inner write-back.
MTCFG	100	1	Memory Type configuration: 0:Use incoming type/cacheability; 1:Use type/cacheability defined by MemAttr field.
ALLOCCFG	104:101	4	Allocation hints override: 0xxx:Use incoming RA, WA, TR hints; 1RWT:Hints are overridden to given values: Read Allocate = R Write Allocate = W Transient = T
Reserved	107:105	3	Reserved

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
SHCFG	109:108	2	Shareability override. 00:Non-shareable; 01:Use incoming shareability attribute; 10 :Outer shareable; 11:Inner shareable.
NSCFG	111:110	2	NS attribute configuration. 00:Use incoming NS attribute; 01: Reserved （ behaves as 00 ）; 10:Secure; 11:Non-Secure.
PRIVCFG	113:112	2	User/privileged attribute configuration. 00:Use incoming PRIV attribute; 01:Reserved （ behaves as 00 ）; 10:Unprivileged; 11:Privileged.
INSTCFG	115:114	2	Inst/Data attribute configuration. 00 :Use incoming INST attribute; 01:Reserved （ behaves as 00 ）; 10:Data; 11:Instruction.
IMPDEF	127:116	12	Implementation-defined per-stream configuration.
S2VMID	143:128	16	the index into the VMTable of the VMTE used for Stage 2 translation/Stage1 VMID tag
IMPDEF	159:144	12	Implementation-defined per-stream configuration.
S2T0SZ	165:160	6	The Size offset of the S2_TTBR0 addressed region, encoded as a 6 bit unsigned number giving the size of the region as 264-T0SZ

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
S2SL0	167:166	2	4kB Page Starting Level for TTBR0 addressed region. 00:Level2; 01:Level1; 10:Level0; 11:UNPREDICTABLE. 16kB or 64kB Page Starting Level for TTBR0 addressed region: 00:Level3; 01:Level2; 10:Level1; 11:UNPREDICTABLE.
S2IR0	169:168	2	Inner Cacheability attributes for the memory associated with the translation table walks using TTBR0. 00:Non-Cacheable; 01:Write-Back，Allocate; 10:Write-Trough; 11:Write-Back，No Allocate.
S2OR0	171:170	2	Outer Cacheability attributes for the memory associated with the translation table walks using TTBR0. 00:Non-Cacheable; 01:Write-Back，Allocate; 10:Write-Trough; 11:Write-Back，No Allocate.
S2SH0	173:172	2	Sharebility attributes for the memory associated with the translation table walks using TTBR0. 00:Non-shareable; 01:Reserved; 10:Outer-shareable; 11:Inner-shareable.
S2TG	175:174	2	Page Granule Size. 00:4kByte 01:64kByte 10:16kByte 11:Reserved

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
S2PS	178:176	3	Physical address size 000:2 bits; 001 :36 bits; 010 :40 bits; 011 :42 bits; 100:44 bits; 101 :48 bits.
S2AA64	179	1	S2AA64域字段说明。 0:VM translation table is AArch32（LPAE）; 1:VM translation table is AArch64.
S2ENDI	180	1	Translation table endianness. 0:Little Endian; 1:Big Endian.
S2AFFD	181	1	Access Flag Fault Disable. 0:An AccessFlag Fault occurs（behaviour controlled by ARS bits）; 1:An AccessFlag Fault never occurs; the TTD.AF bit is considered to be always ‘1’.
S2PTW	182	1	S2PTW域字段说明。 0:Stage 1 translation table walks allowed to any valid Stage 2 address; 1:Stage 1 translation table walks to Stage 2 addresses mapped as any Device. are terminated; a Stage 2 Permission Fault is recorded.
S2HA, HD	184:183	2	Hardware Translation Table Update of Access/Dirty flags. 00:HTTU disabled; 10:Update of Access flag enabled; 01:Reserved – behaves as 00; 11:Update of Access and Dirty flags enabled.
S2{R,S}	186:185	2	Stage 2 Fault behaviour.
Reserved	195:187	8	Reserved.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
S2TTB	239:196	44	Address of Translation Table base. bit[3:0] and bit[63:48] of the address are implied as zero.

16.6.2 Context Descriptor 配置描述

图 16-52 Context Descriptor 数据结构

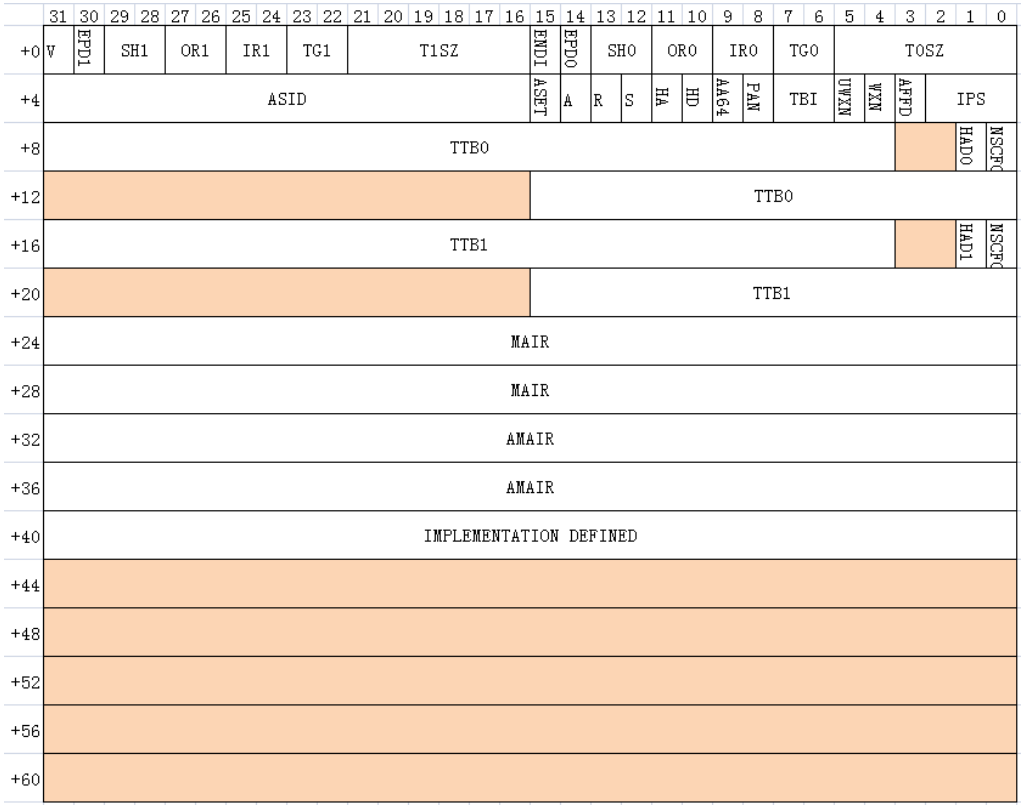


表 16-8 Context Descriptor 数据结构

Field Name	Position	Width	Description
T0SZ	5:0	6	The Size offset of the TT0 addressed region, encoded as a 6 bit unsigned number giving the size of the region as 264-T0SZ.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
TG0	7:6	2	TT0 Translation Granule size, when AA64=1. 00:4kB; 01:64kB; 10:16kB; 11:Reserved.
IR0	9:8	2	Inner region cacheability for TT0 access: 00:Non-cacheable; 01:Write-back cacheable, read-allocate, write-allocate; 10:Write-through cacheable, read-allocate; 11:Write-back cacheable, read-allocate, no write-allocate.
OR0	11:10	2	Outer region cacheability for TT0 access: 00:Non-cacheable; 01:Write-back cacheable, read-allocate, write-allocate; 10:Write-through cacheable, read-allocate; 11:Write-back cacheable, read-allocate, no write-allocate.
SH0	13:12	2	Shareability for TT0 access: 00:Non-shareable; 01:Reserved （behaves as 00）; 10:Outer shareable; 11:Inner shareable.
EPD0	14	1	TT0 translation table walk disable. 0:Perform translation table walks using TT0; 1:A TLB miss on an address that is translated using TT0 causes a Translation Fault. No translation table walk is performed.
ENDI	15	1	Translation table endianness: 0:Little Endian 1:Big Endian

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
T1SZ	21:16	6	The Size offset of the TT1 addressed region, encoded as a 6 bit unsigned number giving the size of the region as 264-T1SZ
TG1	23:22	2	TT1 Translation Granule size: 00:Reserved; 01:16kByte; 10:4kByte; 11:64kByte.
IR1	25:24	2	Inner region cacheability for TT1 access: 00:Non-cacheable; 01:Write-back cacheable, read-allocate, write-allocate; 10:Write-through cacheable, read-allocate; 11:Write-back cacheable, read-allocate, no write-allocate.
OR1	27:26	2	Outer region cacheability for TT1 access: 00:Non-cacheable; 01:Write-back cacheable, read-allocate, write-allocate; 10:Write-through cacheable, read-allocate; 11:Write-back cacheable, read-allocate, no write-allocate.
SH1	29:28	2	Shareability for TT1 access: 00:Non-shareable; 01:Reserved （behaves as 00）; 10:Outer shareable; 11:Inner shareable.
EPD1	30	1	TT1 translation table walk disable: 0:Perform translation table walks using TT1; 1:A TLB miss on an address that is translated using TT1 causes a Translation Fault. No translation table walk is performed.
V	31	1	Context Descriptor Valid. 0:Invalid; 1:Valid.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
IPS	34:32	3	Intermediate Physical Size: 000:32 bits; 001:36 bits; 010:40 bits; 011:42 bits; 100:44 bits; 101:48 bits.
AFFD	35	1	Access Flag Fault Disable. 0:Access flag Faults are reported; 1:Access flag Faults are not reported.
WXN	36	1	Write eXecute Never: 0:Instruction read is allowed as normal; 1:Instruction read to writable page raises a Permission Fault.
UWXN	37	1	Unprivileged Write eXecute Never: 0:Instruction read is allowed as normal; 1:Instruction read from user-writable page raises a Stage 1 Permission Fault.
TBI0	38	1	Top Byte Ignore for TTB0.
TBI1	39	1	Top Byte Ignore for TTB1.
PAN	40	1	Privileged Access Never:
AA64	41	1	Translation table format: 0:TTB{0,1} point to AArch32-format translation tables; 1:TTB{0,1} point to AArch64-format translation tables.
HA, HD	43:42	2	Hardware Translation Table Update of Access/Dirty flags for TT0 and TT1: HA/HD 00:HTTU disabled; 10:Update of Access flag enabled; 01:Reserved – behaves as 00; 11:Update of Access and Dirty flags enabled.
{A,R,S}	46:44	3	Stage 1 Fault behaviour.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
ASET	47	1	ASID Set. 0:ASID in ‘shared set’ 1:ASID in ‘non-shared set’
ASID	63:48	16	Address Space Identifier:
NSCFG0	64	1	Non-secure attribute for the memory associated with the first-level translation table to which TTB0 points: 0:TTB0’s first-level descriptor is fetched using NS=0 access; 1:TTB0’s first-level descriptor is fetched using NS=1 access.
HAD0	65	1	Hierarchical Attribute Disable for the TTB0 region: 0:Hierarchical attributes are enabled; 1:Hierarchical attributes are disabled.
Reserved	67:66	2	Reserved.
TTB0	111:68	44	Address of TT0 base[47:4]. [3:0] and [63:48] of the address are zero.
Reserved	127:112	16	Reserved.
NSCFG1	128	1	Non-secure attribute for the memory associated with the first-level translation table to which TTB1 points: 0:TTB0’s first-level descriptor is fetched using NS=0 access; 1:TTB0’s first-level descriptor is fetched using NS=1 access.
HAD1	129	1	Hierarchical Attribute Disable for the TTB1 region: 0:Hierarchical attributes are enabled; 1:Hierarchical attributes are disabled.
Reserved	131:130	2	Reserved.
TTB1	175:132	44	Address of TT1 base[47:4]. [3:0] and [63:48] of the address are zero.
Reserved	191:176	16	Reserved.

版权所有 © 海思技术有限公司

Field Name	Position	Width	Description
MAIR0	223:192	32	bit[31:24]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b011; bit[23:16]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b010; bit[15:8]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b001; bit[7:0]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b000.
MAIR1	255:224	32	bit[31:24]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b111; bit[23:16]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b110; bit[15:8]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b101; bit[7:0]:Stage 1 Memory Attributes when S1 Page AttrIndex is 0b100.
AMAIR0	287:256	32	Equivalent to ARM CPU Auxiliary Memory Attribute Indirection Registers.
AMAIR1	319:288	32	Equivalent to ARM CPU Auxiliary Memory Attribute Indirection Registers.
IMPDEF	351:320	32	Reserved.
Reserved	511:352	2	Reserved.

16.7 应用场景配置

16.7.1 STE 和 CD 建立

16.7.1.1 STE 建立流程

步骤1 首先读SMMU_（S）_IDR0判断SMMU硬件是否支持2-level STE。

步骤2 配置SMMU_（S）_STRTAB_BASE_CFG.FMT，决定Stream Table的格式为linear或2-Level（在硬件支持的前提下）。

步骤3 配置SMMU_（S）_STRTAB_BASE_CFG.LOG2SIZE[5:0]。在linear STE格式下，LOG2SIZE表示STE的entry数为2^LOG2SIZE，也表示StreamID的有效取值范围为[2^LOG2SIZE-1:0]，也表示在定位STE时，直接用StreamID[LOG2SIZE-1:0]作为索引。在2-level格式下，LOG2SIZE表示二级STE的entry数目的最大值。此时，L1 STD的entry数为2^（LOG2SIZE-SPLIT），用StreamID[LOG2SIZE-1:SPLIT]作为索引定位L1 STD。二级STE的最大entry数由SPLIT决定，实际entry数由STD.SPAN决定。索引用StreamID[SPLIT-1:0].需要注意的是，LOG2SIZE的取值必须满足LOG2SIZE<=SMMU_(S)_IDR1.SIDSIZE。

版权所有 © 海思技术有限公司

步骤4 在2-level结构下，配置SMMU_(S)_STRTAB_BASE_CFG.SPLIT。允许的取值为6,8,10.分别对应L1STD能够指向的最大二级STE空间为4kByte，16kByte，64kByte。

步骤5 配置SMMU_(S)_STRTAB_BASE寄存器中的RA和ADDR域段。RA表示STE的read allocate hint。ADDR[47:6]存储Stream Table的基地址，由于STE的大小为64Byte，所以基地址为64Byte对齐，低6bit固定为0.在Linear结构下，要求基地址按照Stream Table大小对齐，即要求ADDR[LOG2SIZE+5:0]=0。在2-Level结构下，要求对齐64Byte空间和L1STD空间中的较大者，即要求ADDR[MAX(5,LOG2SIZE-SPLIT-1+3):0]=0。

步骤6 至此，STE的基地址，结构，各级大小已经确定。只要StreamID在有效的范围内，即可定位到L1STD和L2STE。STE各个字段的解释见“**数据结构**”章节。

----结束

16.7.1.2 CD 建立流程

步骤1 首先，通过SMMU_(S)_IDR1.SSIDSIZE判断SMMU硬件是否支持SubstreamID。

步骤2 配置STE.S1ContextPtr[47:6]，表示CD的基地址。当Stage1和Stage2都有效的情况下，该地址为IPA空间，需要通过Stage2转换才能从PA空间获取CD。CD存储大小为64Byte，所以地址的低6bit固定为0。

步骤3 在硬件支持的前提下，配置STE.S1CDMax。当S1CDMax=0，表示该STE只有一个CD，则SubstreamID无效，直接用S1ContextPtr获取唯一的CD。当S1CDMax不为0时，表示S1ContextPtr指向的CD数目为2^S1CDMax。

步骤4 在S1CDMax不为0的情况下，配置STE.S1Fmt。

S1Fmt=0表示Linear结构的CD。CD的获取方法为S1ContextPtr基地址加SubstreamID[S1CDMax-1:0]索引。此时，要求地址对齐，即S1ContextPtr[S1CDMax+5:0]=0。

当S1Fmt=1时，表示2-levelCD结构，且第二级的大小为4kByte，即第二级CD的entry数为2^6。此时，L1CD索引为SubstreamID[S1CDMax-1:6]，L2CD的索引为SubstreamID[5:0]。L2CD的基地址必须4kByte对齐，所以必须配置L1CD.L2Ptr[11:0]=0。L1CD的基地址必须对齐64Byte和L1CD空间中的较大者，即要求S1ContextPtr[MAX(5,S1CDMax-6+2):0]=0。

当S1Fmt=2时，表示二级CD大小为64kByte，则L1CD的索引为SubstreamID[S1CDMax-1:10]，L2CD的索引为SubstreamID[9:0]。L2CD地址必须为64kByte对齐，即必须配置L1CD。L2Ptr[15:0]=0。L1CD的基地址必须对齐64Byte和L1CD空间中的较大者，即要求配置S1ContextPtr[MAX(5,S1CDMax-10+2):0]=0。

步骤5 在已经配置了2-levelCD的情况下（S1CDMax不为0），配置STE.S1DSS。S1DSS决定了在2-levelCD结构，但是没有提供SubstreamID的行为。

步骤6 至此，CD的结构，各级CD的大小和基地址已经建立。在STE成功获取的情况下，通过SubstreamID（如果存在），即可获取Stage1转换相关的CD。CD各个字段的具体解释见“**数据结构**”章节。

----结束

16.7.2 STE 透传配置

步骤1 在主存建立Stream Table Entry，配置[CONFIG]位为0b100，根据属性转换需求合理配置STE.MemAttr，STE.Mtcfg，ALLOCCFG，STE.SHCFG，STE.NSCFG，STE.PRIVCFG，STE.INSTCFG。

版权所有 © 海思技术有限公司

步骤2 配置SMMU_STRTAB_BASE/SMMU_STRTAB_BASE_CFG寄存器配置如何查找STE。

----结束

16.7.3 非透传转换配置

步骤1 根据地址转换类型的不同在主存建立各级页表。

- 在Stage1 and Stage2 Bypass类型时，无需建立页表。
- 在Stage1 and Stage2 Nested类型时，如果进行Stage1和Stage2地址转换，那么建立Stage1页表及Stage2页表。
- 如果只进行Stage1地址转换，那么建立Stage1页表，无需建立Stage2页表。
- 如果只进行Stage2地址转换，那么建立Stage2页表，无需建立Stage1页表。

步骤2 如果Stage2 Only类型，跳过此步骤，进行**步骤3**。如果Stage1 and Stage2 Bypass或者Stage1 and Stage2 Nested类型，那么需要在主存建立对应的CD。

- 如果需要进行Stage1地址转换，根据属性转换需求配置CD.MAIR0(CD.MAIR1)；如果不需要进行Stage1地址转换，则无需配置；
- 如果需要进行Stage1地址转换，CD.TG0（CD.TG1）位根据内存页表粒度是4kByte、16kByte还是64KB而进行配置；根据VA输入的地址位宽合理配置CD.T0SZ（CD.T1SZ）位，决定Level0、Level1、Level2还是Level3为起始页表；根据页表的地址空间属性配置CD的Memory属性域段和安全非安全属性域段；如果不需要进行Stage1地址转换，无需配置。
- 根据Stage1页表在内存中的位置配置CD.TTB0（CD.TT1），根据访问页表的属性配置CD.IR0（CD.IR1），CD.OR0（CD.OR1），CD.SH0（CD.SH1）。
- 根据Stage1 Fault处理模式配置CD.A，CD.R，CD.S（网口和PCIe不支持stall功能）。
- 根据AArch32模式或AArch64模式配置CD. AA64域，根据页表在内存的大小端配置CD.ENDI位。

说明

T*SZ配置说明：

在AArch64 结构下输入地址的范围和T*SZ关系为$2^{64-T*SZ}$。V8架构下，对于Stage1转换，T*SZ支持配置区间16~39。

对于Stage2转换，T*SZ支持配置区间16~39，并且由于转换起始level是由SL0决定。

在V7架构下，对于Stage1输入地址最大支持32bit，所以T*SZ支持配置区间为32~39，对于Stage2输入地址最大支持40bit，所以T*SZ支持配置区间为24~39，并且由于转换起始level是由SL0决定。

表 16-9 V8 架构粒度 T*SZ 配置表

Level of Table	Input Address Bits used for index for a full table	Size of Block Entry	Permitted T*SZ values for starting at this level			
			Stage1 table		Stage2 table	
			Min T*SZ	Max T*SZ	Min T*SZ	Max T*SZ
0	47:39:00	N/A	16	24	16	24
1	38:30:00	1GBytes	25	33	21	33

版权所有 © 海思技术有限公司

Level of Table	Input Address Bits used for index for a full table	Size of Block Entry	Permitted T*SZ values for starting at this level			
			Stage1 table		Stage2 table	
			Min T*SZ	Max T*SZ	Min T*SZ	Max T*SZ
2	29:21:00	2MBytes	34	39	30	39
3	20:12	4kBytes	N/A		N/A	

表 16-10 V8 架构粒度 T*SZ 配置表

Level of Table	Input Address Bits used for index for a full table	Size of Block Entry	Permitted T*SZ values for starting at this level			
			Stage1 table		Stage2 table	
			Min T*SZ	Max T*SZ	Min T*SZ	Max T*SZ
0	47	N/A	16	16	N/A	
1	46:36:00	64GBytes	17	27	16	27
2	35:25:00	32MBytes	28	38	24	38
3	24:14:00	16kBytes	39	39	35	39

表 16-11 V8 架构粒度 T*SZ 配置表

Level of Table	Input Address Bits used for index for a full table	Size of Block Entry	Permitted T*SZ values for starting at this level			
			Stage1 table		Stage2 table	
			Min T*SZ	Max T*SZ	Min T*SZ	Max T*SZ
1	47:42:00	N/A	16	21	16	21
2	41:29:00	512MBytes	22	34	18	34
3	28:16:00	64kBytes	35	39	31	39

步骤3 在内存中建立STE。

- 如果需要进行Stage2地址转换，根据内存页表粒度是4kByte、16kByte还是64KB配置STE.TG；根据输入IPA的有效地址位宽配置STE.T0SZ位和STE.SL0位，决定Level0、Level1、Level2还是Level3为起始页表。

版权所有 © 海思技术有限公司

- 配置STE.SHCFG，STE.NSCFG，STE.PRIVCFG，STE.INSTCFG的属性域段。
- 根据Stage2页表的地址空间属性配置STE.S2IR0，STE.S2OR0，S2SH0。
- 根据Stage2 Fault处理模式配置STE.S2R，STE.S2S。
- 根据AArch32模式或AArch64模式配置STE.AA64，根据页表在内存的大小端配置STE. ENDI。
- 根据转换使用CD在内存中的位置配置STE.S1Fmt，STE.S1ContextPtr，S1CDMax；根据CD所在内存中的属性配置CD.S1CIR，CD.S1COR，CD.S1CSH。
- 根据转换类型配置STE.Config。

步骤4 配置SMMU_STRTAB_BASE/SMMU_STRTAB_BASE_CFG寄存器确定转换对应的STE。

----结束

16.7.4 User 域 Bypass 模式属性转换

当加速器使用User域Bypass功能时，SMMU将透传输入Memattr属性，转换到对应Skyros Memattr域为{acfg, (axcache_i[2] | axcache_i[3]), ~axcache_i[1], (axcache_i[2] | axcache_i[3])}。其中acfg为表16-12值。

表 16-12 acfg 转换表

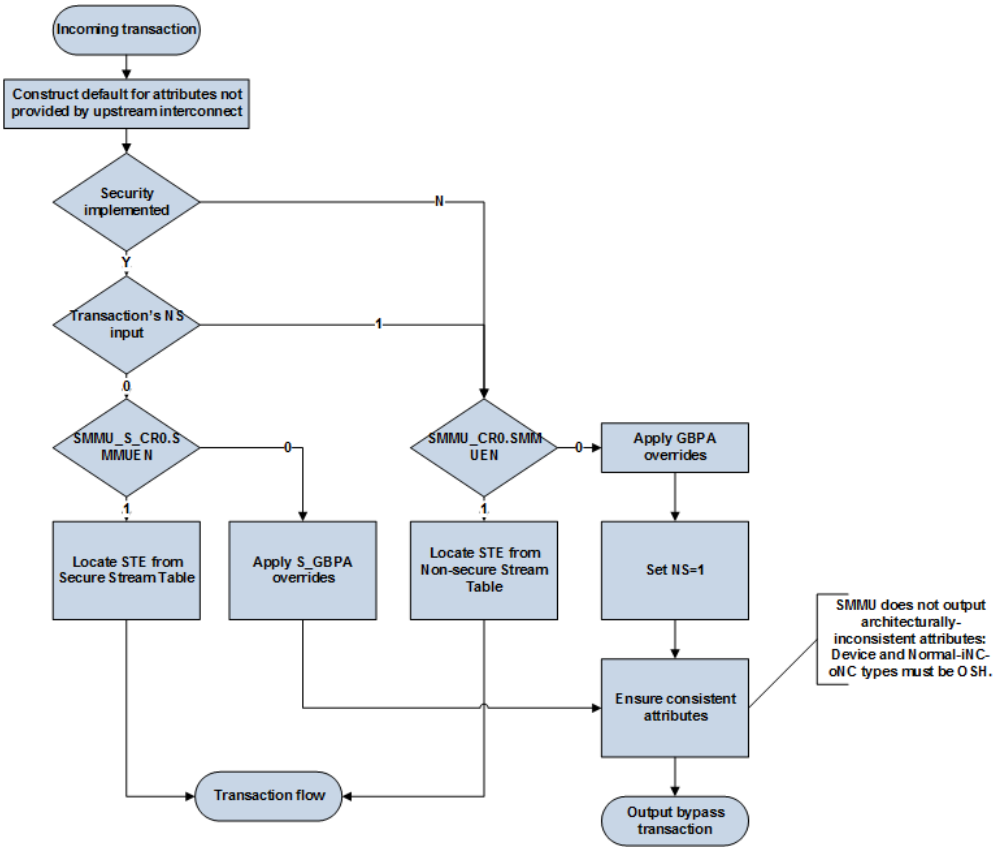
acfg	数据类型
Axcache_i [2]	读操作
Axcache_i [3]	写操作

Snpattr属性当命令为normal属性时使用SMMU配置寄存器值SMMU_USER_CONFIG[bypass_snpattr]，当命令为Device时，Snpattr属性固体为0b00。

版权所有 © 海思技术有限公司

16.7.5 SMMU 整体透传模式属性转换

图 16-53 SMMU 整体透传流程



SMMU CR0/SCR0 Bypass需要配置SMMU_GBPA/SMMU_S_GBPA寄存器，采用SMMU_GBPA/SMMU_S_GBPA寄存器的属性域段。

根据寄存器的mtcfg域段，透传模式属性转换可以分为两种情况。

1. 当SMMU_GBPA.MTCFG/SMMU_S_GBPA.MTCFG为0时

   如果输入Axcache_i[3:2]为0b00时，那么输出Memory属性等于输入Memory属性，为non-cacheable，转换到对应Skyros Memattr域为{0b00, ~axcache_i [1], axcache_i [0]}。

   如果输入Axcache_i [3:2]不为0b00时，寄存器的alloccfg将会生效，当alloccfg==0xxx

表 16-13 ACFG 转换表

SMMU_GBPA.ALLOCCFG/ SMMU_S_GBPA.ALLOCCFG	数据类型	ACFG
0b0xxx	任意	0b00
0b1RWT	读操作	{1,~R}
0b1RWT	写操作	{1,~W}

版权所有 © 海思技术有限公司

最后转换出的Skyros输出Memory属性等于。

表 16-14 透传 mtcfg 为 0 时输出 Memory 属性转换表

ACFG	Skyros Memattr
0b00	{axcache[2]& axcache[0]，axcache[0], 0b0, axcache[0] }
0b10	{ axcache [0]，axcache [0], 0b0, axcache [0]}
0b11	{ 0b0，axcache [0], 0b0, axcache[0]}

2. 当SMMU_GBPA.MTCFG/SMMU_S_GBPA.MTCFG为1时

   SMMU_GBPA.MemAttr/SMMU_S_GBPA.MemAttr域段生效，首先会按Table 1-16将产生输出命令的ACFG。SMMU_GBPA.MemAttr/SMMU_S_GBPA.MemAttr域段进行转换。

表 16-15 Mattr 属性转换表

SMMU_GBPA.MemAttr / SMMU_S_GBPA.MemAttr	MATTR
0x0	0x0
0x1	0x1
0x2	0x2
0x3	0x3
0x4	0x0
0x5	0x5
0x6	0x5
0x7	0x5
0x8	0x0
0x9	0x5
0xA	0x5
0xB	0x5
0xC	0x0

版权所有 © 海思技术有限公司

SMMU_GBPA.MemAttr / SMMU_S_GBPA.MemAttr	MATTR
0xD	0x5
0xE	0x5
0xF	0xF

最后转换出的Skyros输出Memory属性等于。

表 16-16 透传 mtcfg 为 1 时输出 Memory 属性转换表

ACFG	MATTR	Skyros Memattr
0bxx	0x0	0x2
0bxx	0x1	0x3
0bxx	0x2	0x3
0bxx	0x3	0x3
0bxx	0x5	0x1
0b00	0xF	{axcache [2], 0b101}
0b10	0xF	0xD
0b11	0xF	0x5

16.7.6 STE 透传模式属性转换

根据STE.MTCFG域段，透传模式属性转换可以分为两种情况。

1. 当STE.MTCFG域段为0时

   如果输入Axcache_i[3:2]为0b00时，那么输出Memory属性等于输入Memory属性，为non-cacheable，转换到对应Skyros Memattr域为{0b00, ~axcache_i [1], axcache_i [0]}。

   如果输入Axcache_i [3:2]不为0b00时，寄存器的alloccfg将会生效，当alloccfg==0xxx。

表 16-17 STE.ALLOCCFG 转换表

STE.ALLOCCFG	数据类型	ACFG
0x0xxx	任意	0b00
0x1RWT	读操作	{1,~R}

版权所有 © 海思技术有限公司

STE.ALLOCCFG	数据类型	ACFG
0x1RWT	写操作	{1,~W}

最后转换出的Skyros输出Memattr等于。

表 16-18 STE.MTCFG 为 0 时输出 Memory 属性转换表

ACFG	Skyros Memattr
0b00	{axcache[2]& axcache[0]，axcache[0], 0b0, axcache[0] }
0b10	{ axcache [0]，axcache [0], 0b0, axcache [0]}
0b11	{ 0b0，axcache [0], 0b0, axcache[0]}

2. 当STE.MTCFG为1时

   STE.MemAttr生效，首先会产生输出命令的ACFG。然后将STE.MemAttr进行转换。

表 16-19 Mattr 属性转换表

STE.MemAttr	MATTR
0x0	0x0
0x1	0x1
0x2	0x2
0x3	0x3
0x4	0x0
0x5	0x5
0x6	0x5
0x7	0x5
0x8	0x0
0x9	0x5
0xA	0x5
0xB	0x5
0xC	0x0
0xD	0x5
0xE	0x5

版权所有 © 海思技术有限公司

STE.MemAttr	MATTR
0xF	0xF

最后转换出的Skyros输出Memory属性等于。

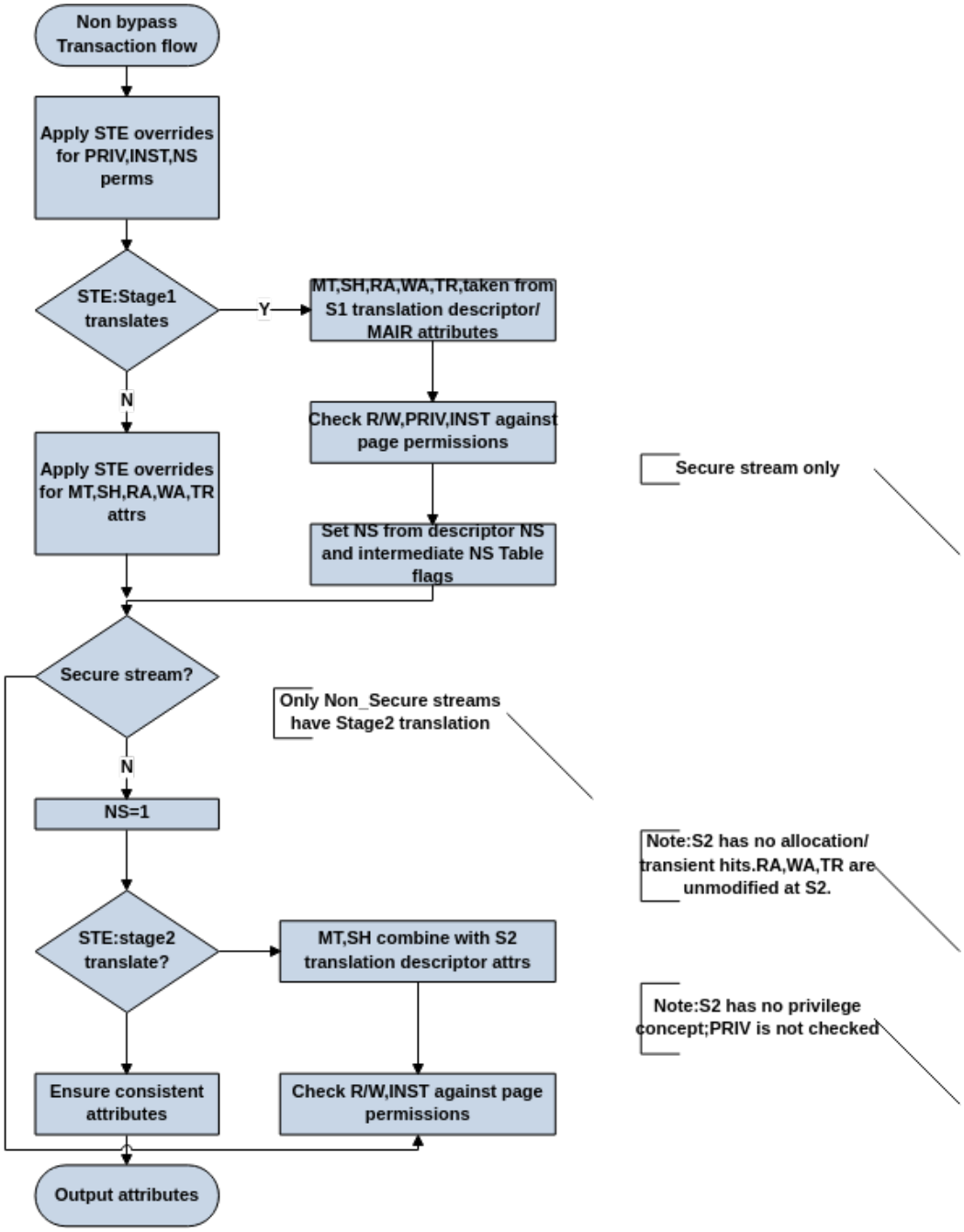
表 16-20 STE.MTCFG 为 1 时输出 Memory 属性转换表

ACFG	MATTR	Skyros Memattr
0bxx	0x0	0x2
0bxx	0x1	0x3
0bxx	0x2	0x3
0bxx	0x3	0x3
0bxx	0x5	0x1
110b00	0xF	{axcache [2], 0b101}
0b10	0xF	0xD
0b11	0xF	0x5

版权所有 © 海思技术有限公司

16.7.7 非透传属性转换

图 16-54 SMMU 非透传属性转换流程



16.7.7.1 Stage1 Only 属性转换

- 输出Instruction属性：等于STE.INSTCFG值。
- 输出Privilege属性：等于STE.PRIVCFG值。
- 输出NS属性：如果输入为secure Stream，则输出NS属性为S1_TTD.NS，S1_TTD.NStable及CD.NSCFG有效值的叠加；如果输入为non-secure Stream，则输出NS属性固定为1。

版权所有 © 海思技术有限公司

- 输出MT属性：等于CD.MAIR[S1_TTD.AttrIndx]。
- 输出RA，WA，TR属性：如果MT属性根据STE.MTCFG和STE.MemAttr进行转换后的MT属性为Normal –Cacheable，则输出RA，WA，TR属性为STE.ALLOCFG与CD.MAIR[S1_TTD.AttrIndx] 叠加后的结果，如果MT属性根据STE.MTCFG和STE.MemAttr进行转换后的MT属性为Normal –Cacheable外其他属性，则输出RA/WA/TR属性则使用CD.MAIR[S1_TTD.AttrIndx]。
- 输出SH属性：等于S1_TTD.SH值。

16.7.7.2 Stage2 Only 属性转换

- 输出Instruction属性：等于STE.INSTCFG值。
- 输出Privilege属性：等于STE.PRIVCFG值。
- 输出NS属性：输出NS属性固定为1。
- 输出MT属性：等于STE {MemAttr，MTCFG}和S2_TTD.MemAttr的叠加。
- 输出RA，WA，TR属性：等于STE.ALLOCFG与S2_TTD.MemAttr的叠加。
- 输出SH属性：等于STE.SHCFG与S2_TTD.SH的叠加。

16.7.7.3 Stage1 + Stage2 属性转换

- 输出Instruction属性：等于STE.INSTCFG值。
- 输出Privilege属性：等于STE.PRIVCFG值。
- 输出NS属性：输出NS属性固定为1。
- 输出MT属性：等于CD.MAIR[S1_TTD.AttrIndx]和S2_TTD.MemAttr的叠加。
- 输出RA，WA，TR属性：如果MT属性根据STE.MTCFG和STE.MemAttr进行转换后的MT属性为Normal –Cacheable，则输出RA，WA，TR属性为STE.ALLOCFG与CD.MAIR[S1_TTD.AttrIndx] 叠加后的结果，如果MT属性根据STE.MTCFG和STE.MemAttr进行转换后的MT属性为Normal –Cacheable外其他属性，则输出RA/WA/TR属性则使用CD.MAIR[S1_TTD.AttrIndx]。
- 输出SH属性：等于S1_TTD.SH与S2_TTD.SH的叠加。

16.7.8 SMMU 同步操作

重新配置页表，导致内存中页表改变且影响到SMMU转换处理时，需要进行TLB同步。当与MMU共用页表发生改变时，需要通过DVM操作无效化TLB。步骤如下：

步骤1 CPU更新页表，将新的页表写入内存。

步骤2 CPU执行TLB无效化指令，发送DVM TLB无效化请求。

步骤3 SMMU收到DVM TLB无效化请求产生的SnpDVMOp，返回SnpResp。

步骤4 SMMU执行无效化TLB操作。

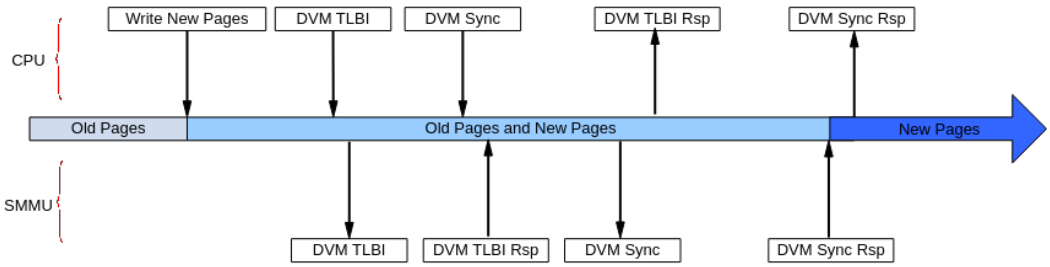
步骤5 CPU发送DVMOp（Sync）请求。

步骤6 SMMU在无效化TLB结束且收到DVMOp（Sync）请求产生的SnpDVMOp（Sync）后，停止地址转换处理，等待SnpDVMOp（Sync）前所有的Transaction结束后，返回SnpResp（Sync），之后启动后续Transaction的地址转换处理。

步骤7 CPU收到DVMOp（Sync）请求的Response。

版权所有 © 海思技术有限公司

图 16-55 TLB 同步流程示意图



CPU更新新页表前，所有SMMU命令使用旧页表；写入新页表后，SMMU命令可能新页表也可能旧页表; 发出DVM Sync Rsp后，所有SMMU命令都是使用新页表。

----结束

16.7.9 Global Error 处理配置

步骤1 配置SMMU_IRQ_CTRL.GERROR_IRQEN寄存器使能Global中断。

步骤2 配置SMMU_GERROR_IRQ_CFG0.ADDR，配置MSI中断的目的地址。

步骤3 配置SMMU_GERROR_IRQ_CFG1.DATA，配置MSI中断数据。

步骤4 配置SMMU_GERROR_IRQ_CFG2.SH及SMMU_GERROR_IRQ_CFG2.MemAttr，配置MSI中断写地址的属性。

步骤5 软件收到Global Error中断，读SMMU_GERROR寄存器值，确定发生错误类型，软件完成对应错误处理后，写SMMU_GERRORN寄存器与SMMU_GERROR对应位一致；Global中断处理完成。

----结束

16.7.10 Commad Queue 配置与功能

软件通过写DDR中的Command Queue与SMMU进行命令交互。SMMU通过SMMU_CMDQ_BASE寄存器获取Command Queue的基地址读取Command命令，Command Queue一个循环FIFO结构，有两个index寄存器SMMU_COMDQ_CONS和SMMU_COMDQ_PROD指向输入命令和取出命令的位置。SMMU_COMDQ_CONS由SMMU写CPU读，SMMU_COMDQ_PROD由CPU写SMMU读取。Command格式见ARM System MMUv3 specification。

16.7.10.1 CMD Queue 寄存器配置

步骤1 读SMMU_(S)_IDR1.CMDQS[4:0]，这个域表示硬件支持的Command Queue index最大bit数。协议规定最大值为19，即最大2^19个entry配置。

步骤2 当SMMU_(S)_IDR1.QUEUES_PRESET=1时，表示Queue的基地址是固定的。这种情况下，ADDR不可配置，寄存器为read only。

在可配置的情况下，SMMU_(S)_CMDTQ_BASE是被SMMU_(S)_CR0.CMDQEN保护的，只有CMDQEN=0时，才能配置。

步骤3 配置SMMU_(S)_CMDQ_BASE寄存器。其中ADDR[47:5]表示Event Queue在物理内存中的基地址。虽然一个Command entry固定为16Byte，协议规定ADDR[4:0]为0，即

版权所有 © 海思技术有限公司

32Byte对齐。LOG2SIZE表示实际CMD Queue index的bit数。其取值必须满足LOG2SIZE<=SMMU_IDR1.CMDQS。

步骤4 配置SMMU_(S)_CMDQ_CONS和PROD寄存器的初始值。

步骤5 配置SMMU_(S)_CR0.CMDQEN=1，使寄存器配置生效。

----结束

16.7.10.2 CMD Queue 的读写

CMD Queue由软件通过PROD.WR指针写入，由SMMU通过CONS.RD指针读取。

CMD queue读写的正确行为应该是：

步骤1 软件读CONS和PROD寄存器，确认CMD Queue非满。读GERROR，确认没有active的CMDQ_ERR。

步骤2 通过BASE和PROD指针，按照协议规定的CMD格式，向CMD Queue写入新的命令。软件只有在确认命令彻底写入内存，对SMMU可见时，才能更新PROD寄存器。

步骤3 SMMU只有在CR0.CMDQEN=1时才会处理CMD Queue。SMMU通过CONS寄存器从内存中获取CMD，更新CONS的指针。若发生error或abort，则更新CONS.ERR和GERROR并触发中断。若CMD正常，则按照协议进行相应。

----结束

16.7.10.3 CMD Queue Error

软件通过PROD.WR指针向CMD Queue写入命令，SMMU在处理命令时，有可能发出error和abort。当Error或abort产生时，SMMU会更新SMMU_(S)_CMDQ_CONS.ERR域段，并更新GERROR.CMDQ_ERR，发出GERROR类型的MSI中断。SMMU停止处理CMD，此时的CONS.RD指针指向发生错误的CMD。

软件纠正错误后，应当配置GERRORN.CMDQ_ERR，使其与GERROR.CMDQ_ERR相同，以此通知SMMU错误已经解除。SMMU会从当前的CONS.RD指针开始重新fetch CMD。

SMMU定义下的SMMU_(S)_CMDQ_CONS.ERR见**表16-21**。

表 16-21 SMMU 定义下的 SMMU_(S)_CMDQ_CONS.ERR

-	-	-
0x00	CERROR_NONE	没有Error
0x01	CERROR_ILL	软件写入的CMD不能被正确解析。Opcode不存在或CMD格式不正确
0x02	CERROR_ABT	SMMU读CMD发生了external abort
0x03	CERROR_ATC_INV_SYNC	CMD_SYNC命令执行失败。由于CMD_SYNC之前发出的CMD_ATC_INV发生了Timeout

版权所有 © 海思技术有限公司

16.7.10.4 CMD Queue 溢出

CMD Queue由软件控制写入。软件通过CONS和PROD寄存器可以判断CMD Queue是否有空间，故不应该触发溢出。

16.7.10.5 CMD Queue 相关的中断

与CMD Queue相关的中断有两种。

当SMMU处理CMD发生错误时，SMMU更新CONS.ERR和GERROR寄存器，触发GERROR类型的MSI中断。要求软件提前配置好SMMU_(S)_GERROR_IRQ_CFG{0,1,2}寄存器。

当CMD_SYNC命令完成时，SMMU发出MSI类型的中断。要求软件提前配置好SMMU_(S)_CMDQ_IRQ_CFG{0,1,2}寄存器。

16.7.11 Event Queue 配置与功能

16.7.11.1 Event Queue 寄存器配置

步骤1 读SMMU_(S)_IDR1.EVENTQS[4:0]，这个域表示硬件支持的Event Queue index最大bit数。协议规定最大值为19，即最大2^19个entry。

步骤2 当SMMU_(S)_IDR1.QUEUES_PRESET=1时，表示Queue的基地址是固定的。这种情况下，ADDR不可配置，寄存器为read only。

在可配置的情况下，SMMU_(S)_EVENTQ_BASE是被SMMU_(S)_CR0.EVENTQEN保护的，只有EVENTQEN=0时，才能配置。故软件需要先设置EVENTQEN=0。

步骤3 配置SMMU_(S)_EVENTQ_BASE寄存器。其中ADDR[47:5]表示Event Queue在物理内存中的基地址。LOG2SIZE表示实际Event Queue index的bit数。其取值必须满足LOG2SIZE<=SMMU_IDR1.EVENTQS。

步骤4 配置CONS和PROD寄存器的初始值。

步骤5 配置EVENTQEN=1，使寄存器的配置生效。

----结束

16.7.11.2 Event 的记录

Event Queue的可写是收到SMMU_(S)_CR0.EVENTQEN控制的，只有在EVENTQEN=1时，SMMU才能向Queue中写入Event。当软件关闭了Event Queue的写使能时，非stall transaction引起的Event会被SMMU直接丢弃不做通知。Stall transaction引起的Event不会被直接丢弃，而是在重新使能后，由SMMU再次写入Event Queue。

当Event Queue溢出时，SMMU会翻转SMMU_(S)_EVENTQ_PROD.OVFLG，以此通知软件发生了溢出。

当写Event Queue引起external abort时，SMMU会改变GERROR相应域段，通知软件做出相应处理。

版权所有 © 海思技术有限公司

16.7.11.3 Event Queue 溢出

当EVENTQEN=1，且当且状态下Event Queue已满，且已经有新的Event因为Queue满而产生丢弃，则SMMU触发Event Queue溢出。由于stall transaction相关的Event不会被丢弃，所以stall transaction相关的Event不会引起溢出。

当溢出条件满足时，SMMU翻转SMMU_(S)_EVENTQ_PROD.OVFLG。软件在每次读PROD寄存器的时候，都应该将OVFLG保存。软件通过比较OVFLG的当前值与上一次读OVFLG所得到的值来判断是否有溢出发生。

软件需要将SMMU_(S)_EVENTQ_CONS.OVACKFLG配置成与OVFLG相同值来通知SMMU当前的溢出已经解除。

SMMU在OVFLG和OVACKFLG值不同的情况下，即有未处理的溢出时，不会产生新的溢出。

16.7.11.4 Event Queue 相关的中断

与Event Queue相关的中断有两种。

- 当Event Queue由空到不空时，会触发Event Queue中断。SMMU会读SMMU_(S)_EVENTQ_IRQ_CFG{0,1,2}的值发出MSI（message signaling interrupt）给GIC。软件需要提前配置好SMMU_(S)_EVENTQ_IRQ_CFG{0,1,2}。
- 当写Event Queue发生external abort时，SMMU会翻转SMMU_(S)_GERROR.EVENTQ_ABT_ERR相应域段。再通过SMMU_(S)_GERROR_IRQ{0,1,2}的值，发出MSI中断。软件需要提前配置IRQ寄存器。同时。软件在检测到GERROR后，采取相应处理，再通过配置GERRORN=GERROR来通知SMMU外部abort已经清除掉。

16.7.11.5 Event Queue 与 Fault Configuration

STE和CD中与Fault configuration相关的域段有：STE.S2R，STE.S2S，CD.A,CD.R,CD.S。

当SMMU发生了translation相关的Fault时，其Fault处理行为受上述字段控制。Translation相关的Fault Event有：F_TRANSLASTION, F_ACCESS, F_ADDR_SIZE, F_PERMISSION.

{A,R,S}的解释为：

A	Terminate时是否abort	当入口的transaction产生了上述四种Fault，若A=1，则向上游的client返回terminate with abort。若A=0，则transaction complete with RAZ/WI，不向client返回abort error。
R	Record	R=1，Fault会写入Event Queue。R=0，Fault不会写入Event Queue。当S=1时，默认R=1。即R只有在S=0时起作用。
S	Stall	Stall=1，SMMU不返回response，将Fault写入Event Queue，分配STAG。软件处理Event Queue，fix问题，向CMD Queue写入CMD_RESUME或CMD_STALL_TERM命令。SMMU读CMD Queue，决定对stalled transaction进行retry还是terminate。

版权所有 © 海思技术有限公司

发生上述四种Fault的处理流程可总结为：

步骤1 Fault产生后，SMMU首先判断S是否为1。若为1，则将Fault写入Event Queue。等待软件处理Event，并向CMD Queue写入RESUME或STALL_TERM命令。

步骤2 若软件发出RESUME命令，SMMU会对匹配到的stalled transaction进行retry。Stall transaction的匹配需要软件提供StreamID，SSec，STAG等信息，详情参考CMD_RESUME命令的格式。若为STALL_TERM命令，SMMU将所有stalled transaction终止。

步骤3 若S=0，SMMU将终止产生Fault的transaction。SMMU判断R是否为1.若为1，则将Fault写入Event Queue，若为0，则不写。

步骤4 若A=1，针对已经被终止的transaction，SMMU会向上游返回error信息。若A=0，则不会返回任何response。

----结束

16.7.12 PRI Queue

16.7.12.1 功能

SMMU支持协议规范的PRIQ（Page Request Queue）功能。工作流程如下：

步骤1 SMMU接收PCIe RC发送的PPR（PRI page request），将PPR以PRIQ的格式写入Memory，并更新PROD寄存器。

步骤2 软件读取PRIQ，更新指针。建立好页表后，向CMDQ写入CMD_PRI_RESP命令，通知SMMU建立页表结束。

步骤3 SMMU根据CMD_PRI_RESP生成PRI response返回给PCIE RC。PPR周期结束。

SMMU的PRIQ采用circular fifo结构，SMMU作为producer，软件侧作为consumer。Queue结构、寄存器管理、空满判断与EVENTQ完全相同。

PRIQ和ATS只支持非安全流。

----结束

16.7.12.2 PRIQ 寄存器配置

PRIQ配置过程中，必须保证SMMU_CR0.PRIQEN配置为0。PRIQ寄存器配置流程如下：

步骤1 配置SMMU_PRIQ_BASE寄存器。Log2size bit[4:0]配置为PRIQ的size，要求不能超过只读寄存器SMMU_IDR1.PRIQS的值。

步骤2 配置SMMU_PRIQ_BASE.addr bit[47:5]，即PRIQ的物理基地址。

步骤3 配置SMMU_PRIQ_BASE.wa，表示PRIQ的write allocate属性。

步骤4 配置SMMU_PRIQ_CONS和SMMU_PRIQ_PROD的初始值。

步骤5 配置SMMU_CR0.PRIQEN=1，使配置生效，使能SMMU写PRIQ。

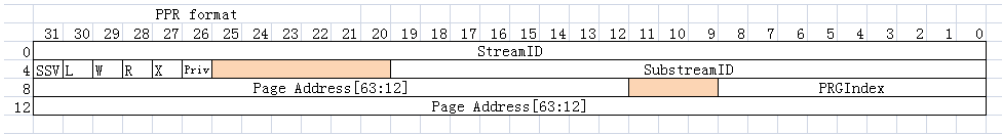
----结束

版权所有 © 海思技术有限公司

16.7.12.3 PRIQ 相关数据格式

与PRIQ相关的数据格式有，SMMU向PRIQ写入的page request格式和软件处理完PPR后向CMDQ写入的CMD_PRI_RESP。以下介绍page request格式。

图 16-56 PRIQ page request 格式



对部分域段做出说明：

1. SSV：SubstreamID Valid
2. PRGIndex[8:0]：PCIE设备发送的Page Request Group的Index。PCIE设备发送的Page Request一般成组出现，PRGIndex用来表示一组request。软件建立页表后发送的CMD_PRI_RESP命令应该包含有想应该PRGIndex。
3. L：一组PRG的Last标志，表示该request为PRG的最后一个request。
4. W,R,X,Priv：表示page request的页表请求权限。分别表示Writable, Readable, Executable, Priviledge。

16.7.12.4 PRIQ 溢出

当PRIQ满，且SMMU收到新的Page Request时，PRIQ发生溢出。

与EVENTQ的处理相同，SMMU在溢出时会翻转SMMU_PRIQ_PROD.ovflg。软件在处理PRIQ时，读ovflg和ovackflg，当发现两个域段取值相反时，表示PRIQ处于溢出的状态。软件应该尽快消化PRIQ中的page request，移动CONS指针，并翻转ovackflg以解除溢出状态。

SMMU硬件在溢出没有解除的情况下，不会再向PRIQ写入新的page request。对于溢出状态下接受到的page request，SMMU的处理方法为：

- 若Last位为1，SMMU会直接向PCIE RC返回response。
- Last位为0的page request，SMMU会直接丢弃。

对于Last位为1的page request，SMMU的response可能有两种：

- 当StreamID所对应的STE有效时，直接返回Success的response。PCIE设备在收到建立页表success的响应后，很有可能会再次发送地址转换请求。该请求会再次产生缺页的相应，PCIE设备有可能再次发送Page Request。
- 当SMMUEN=0，或StreamID超出范围（C_BAD_STREAMID），或发生F_STE_FETCH，或STE无效（C_BAD_STE）时，返回Failure的response。

需要注意的是，由于溢出的发生，最后写入的PRG有可能没有写完，即Last标志位为1的Page Request还没有写入PRIQ就发生了溢出。这种情况下，SMMU硬件必定已经返回了相应的Response。所以要求硬件在处理PRIQ时，只在遇到Last标志位为1的Page Request时才产生CMD_PRI_RESP。

16.7.12.5 PRIQ 相关的中断

与PRIQ相关的中断有：

版权所有 © 海思技术有限公司

1. SMMU写PRIQ发生external abort，SMMU触发GERROR.PRIQ_ABT_ERR中断。软件需要配置GERROR的MSI中断相关寄存器。
2. 当配置SMMU_PRIQ_IRQ_CFG2.LO=0时，SMMU在每次PRIQ由空到满时产生MSI类型的中断。
3. 当配置SMMU_PRIQ_IRQ_CFG2.LO=1时，SMMU在每次收到Last位为1的page request时发送MSI类型的中断。
4. SMMU发送MSI类型中断发生external abort时，触发GERROR的MSI中断。

16.7.13 Streaming Ordered Write 配置流程

步骤1 配置SMMU_USER_CONFIG[stream_req_opt]，使能SO Write保续功能。

步骤2 配置SMMU_USER_CONFIG[stream_write_mode_sp]为0。

步骤3 配置SMMU_USER_CONFIG[stream_write_mode_tp]为0。

----结束

16.7.14 IO-Cache 配置流程

步骤1 配置SMMU_USER_CONFIG[stream_req_opt]，使能SO Write保续功能。

步骤2 配置SMMU_USER_CONFIG[stream_write_mode_sp]为1。

步骤3 配置SMMU_USER_CONFIG[stream_write_mode_tp]为1。

----结束

16.8 初始化

SMMU复位后，使用时需要进行初始化配置。

步骤1 如果需要非透传的地址转换，请按照“19.7.3 非透传转换配置”流程建立页表和Context Bank，配置SMMU_STRTAB_BASE/SMMU_STRTAB_BASE_CFG寄存器；如果需要STE透传，请按照**STE透传配置** ，流程建立Context Bank，配置SMMU_STRTAB_BASE/SMMU_STRTAB_BASE_CFG寄存器；否则无需生成页表和Context Bank，进行**步骤4**。

步骤2 配置SMMU_(S_)CMDQ_BASE寄存器，配置SMMU_(S_)CMDQ_CONS 和SMMU_(S_)CMDQ_PROD寄存器的值相等；配置SMMU_CR0/SMMU_S_CR0.CMDQEN为1，当SMMU_(S_)CR0ACK.CMDQEN==1时，完成CMDQ的配置。

步骤3 配置SMMU_(S_)EVENTQ_BASE寄存器，配置SMMU_(S_)EVENTQ_CONS 和SMMU_(S_)EVENTQ_PROD寄存器的值相等；配置SMMU_CR0/SMMU_S_CR0.EVENTQEN为1，当SMMU_(S_)CR0ACK.EVENTQEN==1时，完成EVENTQ的配置。

步骤4 配置SMMU_CR0/SMMU_S_CR0寄存器。

- 配置SMMU_CR0/SMMU_S_CR0.SMMUEN，如果透传，则配置SMMU_CR0/SMMU_S_CR0.SMMUEN为，否则为1。
- 配置SMMU_CR0/SMMU_S_CR0.CMDQEN 、.EVENTQEN 、.PRIQEN，完成是否开启CMDQ、EVENTQ及PRIQ。

版权所有 © 海思技术有限公司

步骤5 配置SMMU_USER_CONFIG0寄存器。

- 配置SMMU_USER_CONFIG0.L2_TLB_EN，是否使能L2 TLB。
- 配置SMMU_USER_CONFIG0.VICTIM_TLB_EN，是否使能Victim TLB。
- 配置SMMU_USER_CONFIG0.WALK_CACHE_EN，是否使能Walk Cache。
- 配置SMMU_USER_CONFIG0.DEVICE_TREE_EN，是否使能Device tree mode。
- 配置SMMU_USER_CONFIG0.L2_TLB_GS_SET，配置L2 TLB页表粒度。
- 配置SMMU_USER_CONFIG0.SKY_BYPASS_EN，是否使能命令透传LLC功能。
- 配置SMMU_USER_CONFIG0.SKY_BYPASS_WR_FORCE，是否使能写命令透传LLC功能。
- 配置SMMU_USER_CONFIG0.SKY_BYPASS_RD_FORCE，是否使能读命令透传LLC功能。
- 配置SMMU_USER_CONFIG0.PAGE_CACHE_ENABLE，是否使能Page cache。
- 配置SMMU_USER_CONFIG0.STREAM_REQ_OPT决定Strong order命令是否发送保序。
- 配置SMMU_USER_CONFIG0.STREAM_ACK_OPT决定Sky命令是否需要返回ACK。

须知

通过配置SMMU_USER_CONFIG0 [SKY_BYPASS_EN]可以强制使得Cacheable 操作可以不访问LLC，对于短包，当使能SMMU_USER_CONFIG0 [SKY_BYPASS_EN]，并且转换后的操作属性是not allocate的，则操作会直接访问DDR而不去LLC，如果不使能SMMU_USER_CONFIG0 [SKY_BYPASS_EN]，则Cacheable的操作会访问LLC；对于长包，则不管配置如何，都直接访问DDR而不去LLC。

步骤6 配置SMMU_USER_CONFIG1寄存器；

- 配置SMMU_USER_CONFIG0.master0_gs配置L1TLB Master0 的pending粒度；
- 配置SMMU_USER_CONFIG0.master1_gs配置L1TLB Master1 的pending粒度；
- 配置SMMU_USER_CONFIG0.master2_gs配置L1TLB Master2 的pending粒度。
- 配置SMMU_USER_CONFIG0.master3_gs配置L1TLB Master3 的pending粒度。
- 配置SMMU_USER_CONFIG0.master4_gs配置L1TLB Master4 的pending粒度。
- 配置SMMU_USER_CONFIG0.master5_gs配置L1TLB Master5 的pending粒度。
- 配置SMMU_USER_CONFIG0.master6_gs配置L1TLB Master6 的pending粒度。
- 配置SMMU_USER_CONFIG0.master7_gs配置L1TLB Master7 的pending粒度。

须知

大于64k的页表粒度可以配置对应gs值为64k，性能会比较好。

步骤7 配置SMMU_CR1/SMMU_S_CR1寄存器，配置读取STE和CMDQ的属性。

版权所有 © 海思技术有限公司

步骤8 如果CR0/S_CR0 Bypass模式，配置SMMU_GBPA/SMMU_S_GBPA的属性配置域段，完成CR0/S_CR0 Bypass模式下的非安全/安全Transaction属性转换的配置；如果非CR0/S_CR0 Bypass模式，则无需配置这些域段。

步骤9 配置SMMU系统性能统计及DFX配置寄存器, 完成SMMU的性能统计及DFX配置。

步骤10 配置SMMU_IRQ_CTRL对应位使能中断。

步骤11 配置连接到SMMU的Master发相应的操作，SMMU即可对Master发出的操作进行地址转换。

----结束

须知

1. SMMU支持DJTAG和APB单独访问寄存器，APB和DJTAG不可以同时访问SMMU寄存器。
2. SMMU所有配置寄存器只允许在空闲状态时候配置，在SMMU工作状态正在命令转换时，不可以更改寄存器配置。

16.9 接口定义

16.9.1 AXI 接口

SMMU AXI Burst Type只支持INCR类型。

AXI命令的要求及SMMU相应整形规则如下：

1. 保序通道

   SMMU接收到的AXI保序命令数据必须在一个Cacheline范围以内，即不跨越512bit边界。

   如果AXI保序命令的Single的，AxLen为0，那么SMMU会保持原有命令，不做整形处理。

   如果AXI保序命令AxLen为1，AxSize为4，AxAddr为256bit对齐，即地址对齐的两个128bit的AXI transfer，那么SMMU会整形该命令为半个Cacheline的操作，转化后的Skyros命令的Size为5，首地址以半个Cacheline对齐，在AXI接口上仍然按照AXI协议返回读数据或写响应。

   如果AXI保序命令是其他格式的多transfer命令，AxLen不为0，那么SMMU会整形该命令为一个Cacheline的操作，转化后的Skyros命令的Size为Cacheline，首地址Cacheline对齐，但在AXI接口上仍然按照AXI协议返回读数据或写响应。

   SMMU支持128bit乘4拍的Wrap保序读操作。

2. 非保序通道

   SMMU接收到的AXI非保序命令数据可以接受最大为256 Byte长包，但是AXI非保序命令的AxSize、AxLen和AxAddr需要满足表16-22的规格。其中Data Size是最终命令访问的数据量大小。AxAddr需要地址对齐而不能跨越Data Size的地址边界。

版权所有 © 海思技术有限公司

表 16-22 AXI 非保序命令的数据规格

AxSize	AxLen	AxAddr	Data Size	Skyros Size
0b000	0	1Byte对齐	1Byte	0x
0b001	0	2Byte对齐	2Byte	0x1
0b010	0	4Byte对齐	4Byte	0x2
0b011	0	8Byte对齐	8Byte	0x3
0b100	0	16Byte对齐	16Byte	0x4
0b100	1	32Byte对齐	32Byte	0x5
0b100	3	64Byte对齐	64Byte	0x6
0b100	7	128Byte对齐	128Byte	0x7
0b100	11	192Byte对齐	192Byte	0x8
0b100	15	256Byte对齐	256Byte	0x9
0b101	0	32Byte对齐	32Byte	0x5
0b101	1	64Byte对齐	64Byte	0x6
0b101	3	128Byte对齐	128Byte	0x7
0b101	5	192Byte对齐	192Byte	0x8
0b101	7	256Byte对齐	256Byte	0x9

SMMU在返回AXI非保序命令读数据时，同一个命令的读数据以128bit的颗粒粒度乱序，且每一个128bit读数据的rlast都为高。读数据通道RUSER的DataID指示当前128bit数据在Cacheline内的位置。

须知

1. SMMU AXI非保序命令在AxSize为0b100，AxLen为1时，即为两个128bit数据transfer时，这两个128bit数据应该在同一个256bit内，而不可以跨256bit边界。
2. 除128*4的保序读wrap操作外，SMMU AXI Burst Type不支持FIXED和WRAP类型。
3. SMMU不支持Exclusive操作。
4. 对于Device空间的保序读操作，只支持半个cacheline（即axsize=0b100，axlen=1，axaddr[4:0]=0）或者整个cacheline（即axsize=0b100，axlen=3，axaddr[5:0]=0）或者axlen=0的读，其他数据量的read均不支持。

16.9.2 Skyros 接口及约束

在Skyros RN-D接口基础上，SMMU有以下几处自定义：

版权所有 © 海思技术有限公司

1. 发送的命令请求TxReqFlit中， Order域在Device nGnRnE或Device nGnRE时为0b11，同时保证请求保序和终点保序；其他情况下为0b10，保证请求保序；不存在No Ordering Required，读必然会收到ReadReceipt。
2. 发送的命令请求TxReqFlit中， Opcode增加ReadUnique，但该命令不向HN返回CompAck。
3. 发送的命令请求TxReqFlit中， TgtID域固定为0。
4. 发送的命令请求TxReqFlit中， PCrdType域固定为0。
5. 发送的命令请求TxReqFlit中，Size域在ReadOnce、ReadUnique时不一定为64Bytes，可以和ReadNoSnp一样取值。
6. 删除Linkactive和Sactive接口信号。

16.9.3 其他接口信号

与Scheduler之间流量控制的电平信号：

1. block_axi_mstnum，8bits

   基于Master Port的反压阻塞，8个Master依次对应smmu_master_block从低到高的8个bit。此信号某bit高时，Scheduler停止调度该bit对应的Master命令到SMMU。

2. block_axi_rd_order，1bits

   保序读命令的反压阻塞，此信号高时， Scheduler停止调度保序读命令到SMMU。

3. block_axi_wr_order，1bits

   保序写命令的反压阻塞，此信号高时， Scheduler停止调度保序写命令到SMMU。

4. block_axi_rd_norder，1bits

   非保序读命令的反压阻塞，此信号高时， Schedule停止调度非保序读命令到SMMU。

5. block_axi_wr_norder，1bits

   非保序写命令的反压阻塞，此信号高时， Schedule停止调度非保序写命令到SMMU。

16.10 Performance Monitor 以及芯片调试 DFX 功能特性描述

16.10.1 Performance Monitor

按照SMMUv3协议实现Performance Monitor，完成相应的DFX功能。Performance Monitor包含的统计事件已经在1.2章节列出，具体事件的编码详见nManager定义。

SMMU实现一组PMCG（Performance Monitor Counter Group），共8个64位计数器。实现对每个寄存器的分别配置，可配置项目有：匹配的安全属性，能够匹配的StreamID，Event类型，计数使能，中断使能。

版权所有 © 海思技术有限公司

16.10.2 Performance Monitor 配置方法

步骤1 配置EVTYPERn寄存器域段。配置ovfcap为1，当该计数器溢出时，会触发一次对所有计数器的capture。

步骤2 配置filter_sec_sid，filter_sid_span，表示该计数器统计对象的Streamid的安全非安全属性和匹配模式。

步骤3 配置event域段，表示该计数器目标event类型。

步骤4 配置SMRn寄存器，performance monitor根据filter相关域段生成可匹配的Streamid范围。

步骤5 配置CR寄存器，global使能。

步骤6 配置CNTENSET对应bit位，使能计数器，计数器开始统计。

----结束

16.10.3 DFX Debug

本模块提供了以下方式用于内部状态查询，方便系统调试。

1. 查询两个AXI接口的反压状态和被反压的MSTID。
2. 查询两个skyros Queue任意entry的状态以及数据。
3. 查询Commond Queue状态。
4. 查询正在转换命令统计。
5. 查新skyros Queue中stall命令状态。

16.10.4 中断上报

按照协议实现了MSI中断机制。可通过寄存器配置各个计数器在计满溢出时触发MSI中断。再通过OVSSET[7:0]确定发生溢出的计数器。

16.11 典型问题定位流程

任务描述符异常中断的建议定位流程如下：

步骤1 查询SMMU_HIS_CMD_NUM寄存器，确认是否有命令接收但未处理完成。

1. 查询SMMU_HIS_CMD_NUM寄存器，记录所有命令的状态。
2. 记录当前时间戳和系统状态。

步骤2 若存在未处理完成命令，查询DFX sky Queue相关寄存器。

1. 检查DFX sky Queue的状态标志位，记录队列中的命令数量和状态。
2. 记录当前时间戳和系统状态。

步骤3 若存在stall，解析event Queue，软件发送cmd解除stall。

1. 解析event Queue，记录事件类型、发生时间和上下文信息。
2. 根据解析结果，发送相应的命令解除stall。
3. 记录当前时间戳和系统状态。

步骤4 若ptw Queue状态机挂死，进一步推理挂死原因。

版权所有 © 海思技术有限公司

1. 尝重启ptw Queue状态机或重置相关寄存器。
2. 分析ptw Queue的状态机状态，记录相关寄存器值。
3. 记录当前时间戳和系统状态。

----结束

16.12 时钟复位

SMMU的软复位请求及撤离需要与门控配合使用（寄存器请参考各subctrl寄存器描述表单）：

1. 配置SMMU软复位请求寄存器。
2. 配置SMMU门控寄存器关闭，关闭时钟。
3. 配置SMMU软复位撤离寄存器。
4. 配置SMMU门控寄存器打开，使能时钟。

版权所有 © 海思技术有限公司

17 Timer

17.1 概述

Timer模块主要实现定时、计数功能，可以供操作系统用作系统时钟，以及供应用程序使用。

芯片提供38个Timer模块，Timer 0～11为功能完全相同的32 bits Timer，Timer12~35为功能完全相同的64 bits Timer，Timer0~35Timer为系统公用，Timer36~37固定只给安全OS使用。

17.2 特性

Timer模块具有以下特点：

- 带可编程8位预分频器的64 bits/32 bits/16 bits减法定时器/计数器；
- 计数时钟可配置，Timer的工作时钟为系统总线时钟的二分频，以及它的分频时钟(1/16/256分频)；
- 支持3种计数模式：自由运行模式、周期模式和单次计数模式；
- 有2种载入计数初值的方法，分别通过TIMERx_LOAD和TIMERx_BGLOAD寄存器实现；
- 当前的计数值可随时读取；
- Timer中断上报：
  - 当计数值减到1时会产生中断，各个Timer之间中断独立；
  - 中断支持电平中断和脉冲中断可配，中断上报类型为N-N或1-1由系统控制器Sysctrl的配置决定；
  - 可通过配置peri_cubctrl寄存器SC_TIMER32_SPI_EN/SC_ TIMER64_SPI_EN来选择是否上报中断。
- 通过页表配置支持多模隔离，1个Timer仅能分配给一个模式使用；
- 支持原子读取需求；
  - 支持读取64bit定时器值的原子特性；
  - 支持多核或者多进程访问64bit定时器值不需要加锁；

版权所有 © 海思技术有限公司

- Timer0~23的寄存器配置权限受系统外设安全控制器TZPC的控制，Timer模块内部没有安全和非安全的区分，Timer24~25的寄存器固定为安全寄存器。

17.3 关键术语

Timer相关的关键术语表如表17-1所示。

表 17-1 Timer 术语表

术语	描述
DFS	Dynamic Frequency Scaling
TZPC	TrustZone Protection Controller

17.4 遵循协议

Timer接口为APB2.0接口。

17.5 功能描述

17.5.1 典型应用

17.5.1.1 TICK

32/64 bits Timer可用于给操作系统提供TICK时钟，控制操作系统运行快慢。一般要求是：1秒内触发60~100次中断，推荐采用10ms触发一次中断。

说明

- 操作系统内部对任务的调度管理是在TICK节拍下进行（可以把TICK就当作操作系统的工作时钟）。
- TICK跑的越快，那么操作系统就运行得越快，任务调度、上下文切换越频繁，操作系统实时性越高，但对CPU的处理能力要求越高！
- TICK跑得慢，对CPU处理能力要求不高，但操作系统实时性降低，需要考虑能否满足业务对实时性的要求。
- 硬件定时器精度可到μs，甚至ns级。

17.5.1.2 波特率控制

32/64 bits Timer作为硬件定时器，可用于主从串口通信，波特率为187500bps。

版权所有 © 海思技术有限公司

说明

1. 当主串口（Master）向从串口（Slave）发送命令的同时，启动该Timer进行计时。
2. 在10μs内，如果Master没有收到Slave的反馈，则认为该次传输失败，Master将重传，并重置Timer。
3. 软件会记录重传次数，超过设置阈值，则认为该接口失效，转交接口异常处理。

17.5.1.3 流量控制

Timer作为硬件定时器，可用于以太网口防大流量冲击场景。

说明

1. 目前网络安全堪忧，需要独善其身。
2. 针对从以太网口进来的IP数据包，如果需要转交TCP/IP协议栈处理，将会占用CPU部分资源。
3. 在ARP攻击场景中，大量无效ARP包对以太网口进行冲击，频繁的占用CPU资源，导致CPU负载过重，无法处理正常业务，甚至宕机、业务下线等异常。
4. 在网口增加一个Timer，限定在一定时间内，只能从以太网口传输一定的IP数据包。如果超过该设置，将关闭网口三层功能一段时间，采用二进制指数退避算法，随机退避一段时间后，再开启。有效防止在大流量网口冲击情况下，由于CPU拥塞导致的业务失效问题。

17.5.1.4 计数范围和计数精度

Timer计数范围和计数精度如表17-2所示。

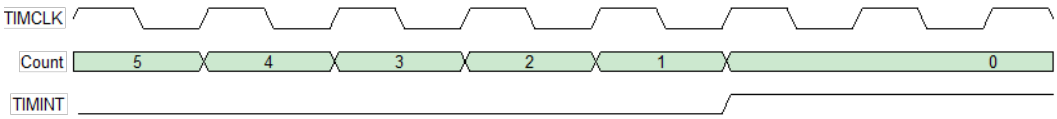
表 17-2 Timer 计数范围和精度表

计数时钟	计数位宽	计数精度	计数范围
200MHz	16 bits Timer	5ns	0 ~ 327μs
	32 bits Timer		0 ~ 21s
	64 bits Timer		0 ~ 2926年（一年按照365天计算）
50MHz	16 bits Timer	20ns	0 ~ 1310μs
	32 bits Timer	-	0 ~ 84s
	64 bits Timer	-	0 ~ 11704年（一年按照365天计算）

17.5.2 Timer 功能原理

32 bits Timer基于一个32 bits/16 bits（可配置）减法计数器。64 bits Timer基于一个64 bits/32 bits（可配置）减法计数器。计数器的值在每个计数时钟的上升沿减1。当计数值递减到1，Timer 将在下一拍产生一个中断，中断产生时序如图17-1所示。

图 17-1 Timer 中断产生时序图



版权所有 © 海思技术有限公司

Timer有以下3种计数模式：

- 自由运行模式

  定时器持续计数，当计数值减到1时又自动回转到其最大值，并继续计数。当计数长度为32 bits时，最大值为0xFFFF_FFFF。当计数长度为16 bits时，最大值为0xFFFF。在自由模式下，也可以载入计数值，并立即从载入值减计数，但计到1时回转到其最大值。

- 周期模式

  定时器持续计数，当计数值减到1时从TIMERx_BGLOAD 寄存器中再次载入初值并继续计数。

- 单次计数模式

  向定时器中载入计数初值。当定时器的计数值减到1时就停止计数，直到重新被载入新值且定时器处于使能状态，才再次开始计数。

每个Timer具有一个预分频计数器，可将工作时钟在Timer内部再次进行1分频、16分频或256分频。进一步提高计数时钟频率的选择灵活性。

对定时器载入计数初值的方法如下：

- 通过写TIMERx_LOAD寄存器可对定时器载入计数初值。当定时器处于工作状态时，如果向TIMERx_LOAD寄存器写入值，会导致定时器立刻从新值开始重新计数。适用于所有计数模式。
- 通过写TIMERx_BGLOAD寄存器可以设定周期计数模式的计数周期。写该寄存器不会立刻影响定时器的当前计数，定时器会继续计数直到计数值减到1。然后载入TIMERx_BGLOAD寄存器中的新值开始计数。

17.5.3 多模隔离

每个Timer的寄存器空间最大为64kBytes，实际使用小于4kBytes。通过页表划分，将多个模式对Timer访问映射到到不同的Timer基地址，从而实现Timer模块在物理上的隔离。

须知

一个Timer仅能分配给一种模式使用，否则会导致计数值错乱影响软件运行。

17.5.4 原子特性

对于64bits Timer，需要支持原子特性，确保Master读取计时信息的正确性。

每个64bits Timer内部有4组record寄存器，当收到某个Master的读操作时，根据地址返回32bits数据，另外32bits数据寄存在record寄存器，等待下次同一Master的读取。所以，一个Timer最大支持4个Master的并发操作。

多个64bits Timer之间可以通过同一个使能寄存器开启计数。当多个64bits Timer的初始值设定一致后，配置该使能寄存器，多个64bits Timer可以同时启动，即实现4个以上的Master并发访问。

版权所有 © 海思技术有限公司

17.6 软件指南

17.6.1 Timer 初始化

复位后使用时初始化：

一个标准系统至少需要有两个定时器，其中一个供操作系统使用，另一个供应用程序调用。因此系统初始化时应对Timer进行初始化。

Timer初始化时，需要配置TIMERx_LOAD，TIMERx_BGLOAD和TIMERx_CONTROL寄存器。

表 17-3 TIMERx_LOAD 寄存器描述

位	名字	描述
[31:0]	timerx_load	Timer0的计数初值。 0x0：保留。 0x1～0xFFFF_FFFF：n个计数器工作时钟周期。 注意：该寄存器最小写入有效值为1；若写入0，则Timerx立即产生中断；当向TIMERx_BGLOAD寄存器写入值时，TIMERx_LOAD的值也会被覆盖，但计数器的当前值不会受到影响。 “x”表示0～1。

表 17-4 TIMERx_BGLOAD 寄存器描述

位	名字	描述
[31:0]	timerx_bgload	Timerx的计数初值。 0x0：保留。 0x1～0xFFFF_FFFF：n个计数器工作时钟周期。 在周期模式下，当定时器的计数值递减到0时重新载入此计数初值。 当向TIMERx_BGLOAD寄存器写入值时，TIMERx_LOAD的值也会被覆盖，但定时器计数的当前值不会受到影响。 “x”表示0~1。

版权所有 © 海思技术有限公司

表 17-5 TIMERx_CONTROL 寄存器描述

位	名字	描述
[31:8]	-	-
[7]	timeren	Timer32使能。 0：禁止； 1：使能。
[6]	timermode	Timer32计数模式选择。 0：自由运行模式，该模式下计数器递减计数到0后，载入计数器最大值（16bit模式时为0xFFFF，32bit模式时为0xFFFF_FFFF）。 1：周期模式，该模式下计数器递减计数到0后，载入Time32_BGLOAD设置值（16bit模式时为低16bit数据，32bit模式时为寄存器设置值）。 **注意：16bit/32ibt模式可通过[timersize]域设置。**
[5]	intenable	Timer32中断屏蔽。 0：屏蔽； 1：不屏蔽。
[4]	int_type	Timer32中断类型。 0：电平中断； 1：脉冲中断； **注意：Timer的中断类型需要和GIC配合使用，默认为1-1电平中断，当为N-N时，需要配置为脉冲中断，同时配置SYSCTRL中的SC_GIC_CTRL寄存器。**
[3:2]	timerpre	Timer32预分频因子设置。 00：不预分频； 01：4级预分频，将Timer32时钟频率除以16； 10：8级预分频，将Timer32时钟频率除以256； 11：保留。
[1]	timersize	Timer32的16bit/32bit计数器选择。 0：16bit计数器； 1：32bit计数器。
[0]	oneshot	选择Timer32的计数模式为一次性计数还是回卷计数。 0：回卷（wrapping）计数；计数器计数到0后，重新载入新值进行计数。 1：一次性（one-shot）计数，计数器计数到0后，停止计数。

初始化单个Timer时应按以下步骤进行配置：

版权所有 © 海思技术有限公司

步骤1 写TIMERx_LOAD寄存器，载入Timer计数初值。

步骤2 当需要Timer工作在周期计数模式下、且计数周期与载入Timer的计数初值不同时，写TIMERx_BGLOAD寄存器，设置Timer的计数周期。否则跳过此步骤，进入**步骤3**。

步骤3 写SC_CTRL寄存器，按照**表17-5**方式配置Timer的工作时钟。

步骤4 写寄存器TIMERx_CONTROL，配置Timer的计数模式、计数器长度、预分频系数及中断使能，启动Timer计数。

----结束

须知

针对64bit timer，支持多个timer同时启动。在Subctrl控制寄存器中有timer同时使能的开关配置。当需要多个64bits timer同时启动时，可配置PeriSubctrl中SC_TIMER64_EN_CTRL[15:0]寄存器，每个bit对应一个64bit timer，配置多个bit同时为1，可使多个64bit timer同时启动。

17.6.2 中断设置

Timer主要用于产生定时中断，因此Timer的中断处理任务主要是激活等待定时中断的进程。Timer的中断默认为电平中断类型，通过系统控制器Sysctrl的配置，最多可以支持10个SPI脉冲中断类型。

以软件响应Timer0中断为例，其操作步骤如下：

步骤1 向TIMER0_INTCLR写入任何值，清除Timer0的中断。

步骤2 激活等待该中断的进程，使其继续执行。

----结束

17.7 性能

无。

17.8 DFX

无。

17.9 注意事项

Timer64_0_VALUE0_L [31:0] 和Timer64_0_VALUE0_H [31:0]为Timer的实时计数值，需要通过下面的步骤来获取一个正确的值。

步骤1 读Timer64_0_VALUE0_H的值。

步骤2 读Timer64_0_VALUE0_L的值。

版权所有 © 海思技术有限公司

步骤3 读Timer64_0_VALUE0_H的值；如果两次Timer64_0_VALUE0_H的值不一致，则丢弃当前计数值，软件重新发起一次访问。

----结束

使用Timer内部的16/256分频功能时，可能会有一个周期的计数误差。

版权所有 © 海思技术有限公司

18 TSensor

18.1 片上温度监控 TSensor

本段落主要是对系统级TSensor的功能和应用做详细描述。

18.1.1 概述

Hi309a集成了2个TSensor用于实时监测芯片结温，并进行数字量化输出。CPU和JTAG可以通过CPU_CLUSTER SUBCTRL模块读取TSensor送出的10-bit量化信息。

18.1.2 特性

TSensor具有以下特性：

- 检测温度范围： -40℃～+125℃，采用10 bits SARADC量化温度，温度分辨率为0.205℃/LSB。
- 绝对温度检测精度： -40℃~+125℃范围内校准前±7.5℃，校准后±3℃，超出温度范围的精度会有损失。
- 1M时钟TSensor，±10%精度以内，占空比40%～60%以内。
- 支持统计温度校准。
- 电源需求：
  - DVDD，IP端0.8V数字电源，-20%~+10%精度（AC+DC，AC<±30mv）；功耗小于0.5mA。
  - AVDD，IP端1.8V模拟电源，-5%~+10%精度（AC+DC），AC<±20mv。
- 可以根据软件设置的高低温水线上报中断。
- 支持CPU和JTAG接口读取TSensor。

18.1.3 关键术语

表18-1中的术语将在本章节表示特殊的含义。

版权所有 © 海思技术有限公司

表 18-1 TSensor 关键术语描述

术语	描述
TSensor	Temperature Sensor，温度传感器。
ATC	Automatic Temperature Controller，自动温度控制仪。

18.1.4 遵循协议

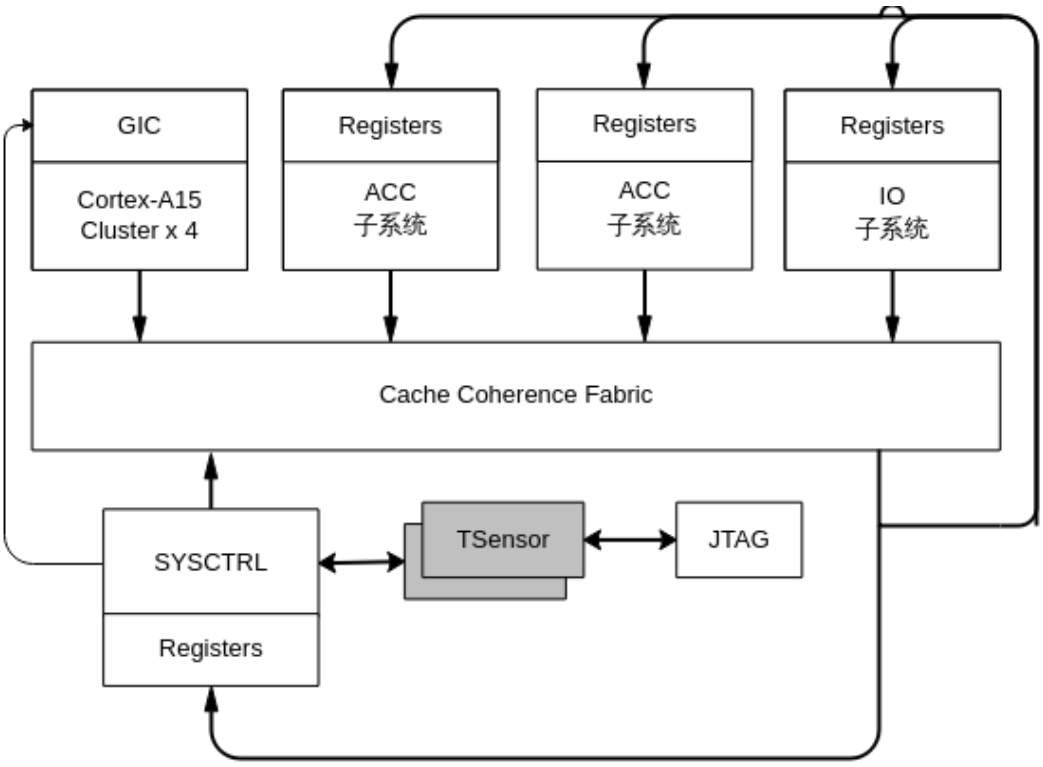
不涉及。

18.1.5 功能描述

18.1.5.1 应用框图

TSensor的应用如图18-1所示。

图 18-1 TSensor 应用图



TSensor主要有两个用途：

- 功耗测试的结温监控。

  在功耗测试中，通过读取TSensor来调节外部温度控制仪（例如ATC或者Thermal Jet），使芯片结温达到功耗测试要求的结温。

版权所有 © 海思技术有限公司

须知

功耗测试分为静态功耗测试和动态功耗测试。静态功耗测试芯片处于低功耗模式，功能时钟全部关闭，通过JTAG读取TSensor来通知外部温控设备调整芯片的结温；动态功耗测试时可以通过CPU，也可以通过JTAG读取TSensor，根据芯片的实时结温通知外部温控设备调整芯片的结温。

- 正常工作的结温监控。

  芯片正常启动后，CPU通过读取TSensor实时监控芯片的结温，并通过预先设置的高低温水线，对超出芯片允许的工作结温的场景进行预警。

18.1.5.2 相关模块

在系统中，TSensor模块会和下面多个模块进行交互，分别描述如下：

- SUBCTRL：CPU访问SubCtrl设置TSensor的工作参数，获取TSensor测量的结温。
- JTAG：JTAG可以直接配置TSensor使能，并获取TSensor测量的结温。

18.1.5.2.1 模块框图

参考TSensor IP手册。

18.1.5.3 TSensor 温度探测

芯片上电后使能TSensor，芯片逻辑检测到TEMP_READY边沿，开始采样temp_out数据，连续采样16或16的整数倍周期的temp_out，取平均值作为一次检测到CPU温度。

18.1.5.4 TSensor 结温修正

TSensor需要校准才可以使用。

18.1.5.4.1 TSensor 两点校准

每个TSensor预留2组11-bit eFuse做两点温度校准，分别对应105℃和25℃校准温度码。

表 18-2 TSensor 预留 2 组 11-bit eFuse 做两点温度校准

eFuse 1	位宽	功能	信号明细
[789:768]	22	TSensor0 Trim	[10:0] 常温25℃ TSensor0
			[21:11] 高温105℃ TSensor0
[811:790]	22	TSensor1 Trim	[10:0] 常温25℃ Tsensor1
			[21:11] 高温105℃ Tsensor1

- 【21:0】TSensor0温度校准值。

  【21:0】对应SubCtrl寄存器SC_EFUSE_TSENSOR0_TRIM。

版权所有 © 海思技术有限公司

- 【43:22】TSensor1温度校准值。
  【43:22】对应SubCtrl寄存器SC_EFUSE_TSENSOR1_TRIM。

EFUSE 11- bit中MSB为符号位（1表示负，0表示正），为0.25℃（每bit），有效数值为[-1023，1023]，温度可烧写范围为 [-255.75℃，255.75℃]。

ATC设置外部温度为105℃和25℃后，读取TSensor输出码，烧入对应的eFuse字段，作为高温和常温校准温度码。

装备和软件读取eFuse校准温度码和TSensor值，通过两点校准公式进行校准补偿得到校准后的温度值：

$$y=(80/(T2-T1))*x+25-(80/(T2-T1))*T1$$

其中，T1为eFuse中25℃下的温度校准码值；

T2为eFuse中105℃下的温度校准码值；

X为读取的温度码值；

Y为Die真实结温。

18.1.6 软件指南

18.1.6.1 初始化

1. 配置系统控制器寄存器SC_TSENSOR_ALARM，根据产品场景设置高低温水线。
2. 配置TSensor采样次数寄存器SC_TSENSOR_SAMPLE_NUM，建议配置为16的整数倍。
3. 配置系统控制器寄存器SC_TSENSOR_CTRL的temp_en为1，使能TSensor。
4. 读取系统控制器SC_TEMP_SAMPLE_AVERAGE寄存器中的tsensor_valid信号，当该信号为1时（配置Tsensor使能，到tsensor_valid有效，至少有SC_TSENSOR_SAMPLE_NUM*0.512+1（ms）的时延，软件设计时要注意增加等待时延）， 读取SC_TEMP_SAMPLE_AVERAGE的tsensor_sample获取温度值（实际的结温还需要温度校准）。
5. 配置系统控制器寄存器SC_TSENSOR_CTRL的temp_en为0，完成本次结温数据获取。

18.1.7 性能

18.1.7.1 应用场景

如“**应用框图**”所示，TSensor主要应用在两个场景下：

- 功耗测试的结温监控。静态功耗模式下JTAG能够使能并读取TSensor测量的结温值；在动态功耗模式下JTAG和CPU能够使能并读取TSensor测量的结温值。
- 正常应用的结温监控。TSensor在芯片正常启动后，CPU能够实时使能并读取TSensor测量的温度值，并通过预先设置的高低温水线，对超出芯片（或者IP）允许的工作结温的场景进行预警。

版权所有 © 海思技术有限公司

19 WatchDog

19.1 关键术语

表 19-1 WDOG 关键术语描述

Terms	Description
SBSA	Server Base System Architecture服务器系统架构。
WDOG	WatchDog看门狗。
AMBA	Advanced Microcontroller Bus Architecture。

19.2 概述

WDOG（系统看门狗）用于系统异常情况下，一定时间内发出异常指示信号或者复位信号，以复位整个系统。

在指定的一个时间阈值内，如果软件没有更新WDOG内的比较器，系统看门狗发出中断指示信号上报CPU，通知CPU更新系统看门狗。此时如果系统看门狗内的比较器在一个时间阈值内仍然没有被CPU刷新，则产生一个复位信号输出复位该CPU核或整系统，或者产生一个更高优先级的1-N SPI中断。

WDOG具有两个寄存器地址段，一个为系统看门狗刷新寄存器段，另一个为系统看门狗控制寄存器段。这样就可以实现将看门狗控制寄存器段分配给高权限的内核软件使用；而低权限的用户软件只能使用刷新寄存器段。

19.3 特性

WDOG具有以下特性:

- 编程接口符合AMBA APB协议标准，易于集成到SOC实现中。
- 架构实现符合ARM SBSA通用架构要求，易于移植到其他系统中。
- 系统需要提供安全和非安全两种WDOG。

版权所有 © 海思技术有限公司

- WDOG第一次超时溢出后，非安全WDOG产生LPI/SPI中断告警上报CPU；安全WDOG产生1-N SPI安全中断上报CPU。
- WDOG第一次超时溢出后，在下一个时间阈值内CPU仍然没有刷新WDOG内的比较器，产生第二次超时溢出。第二次超时溢出时，非安全WDOG产生1-N SPI中断上报CPU；安全WDOG产生复位信号复位系统。
- 每个WDOG需要128kByte的寄存器空间，低64kByte给内核态使用，高64KB给用户态使用。
- WDOG工作逻辑只在上电复位期间才可以被复位，不支持非上电复位。
- 系统非上电复位，不会清除WDOG内配置和状态寄存器。
- WDOG使用系统时间SYS_CNT计数值作为比较器的基准时间。
- 支持超时时间间隔offset_value和比较器初始值WCV软件可配置，一旦系统时间SYS_CNT计数值大于软件配置的时间比较阈值，则比较器溢出。
- 支持看门狗的比较器状态软件可查询。
- 看门狗阈值寄存器WCV_L和WCV_H支持单线程原子性读/写操作。

须知

WDOG中对64bit寄存器（WCV_L[31:0]和WCV_H[31:0]）支持单线程原子操作。

原子操作机制：

1. 一次全新的读访问，如果操作低32bit寄存器后，则同时将高32bit寄存器值保持在模块内部，在下次访问高32位数据时，返回高位数据；如果继续访问低位数据，则返回更新后的低位数据，重新锁存高32bit数据。同理，如果先访问高32位数据，则同时会将低32位数据保存在模块内部，在下次访问低32位数据时返回该值。在一次连续的低32位和高32位被访问之后，则本次原子访问操作结束。
2. 一次全新的写访问，只有在低32位和高32位被连续写访问后，才会将64位数据更新到寄存器中。

19.4 遵循协议

AMBA APB.

19.5 功能描述

多核处理器的发展趋势是高度整合，在一个芯片中集成越来越多的功能模块和CPU核。一方面功能模块和CPU核的增加、总线架构的日益复杂都给整系统的稳定性带来了新的挑战。另一方面，软件架构的日益复杂性，软件操作异常导致CPU异常的概率逐渐增大。为了监测系统的异常错误行为，系统看门狗应运而生。

19.5.1 内部比较器机制

WDOG的运行基于一个64bit的比较器和一个64bit的加法器，比较器的初始值由WCV看门狗比较阈值寄存器载入。比较器的参考比较值为系统计数器提供的系统时间sys_cnt[63:0]。一旦系统时间超过了WCV阈值寄存器，立即产生一个WS0。在下一个时钟上升沿，WCV寄存器更新为当前时间与WOR的和，重新开始比较。

版权所有 © 海思技术有限公司

如果比较器在第二次检测到系统时间超过了WCV阈值，CPU还没有刷新WDOG比较器，则立即产生WS1，比较器停止工作直到被再次刷新。

说明

WS0表示比较器第一次溢出事件；WS1表示比较器第二次溢出事件。

19.5.1.1 比较器刷新条件

19.5.1.1.1 软件更新 WDOG 比较器（Explicit watchdog refresh）

1. Refresh寄存器WRR被软件读操作。
   使用sys_cnt[63:0]+wdog_offset_value[31:0]作为新的比较阈值CompareValue。
2. Offset补偿寄存器WOR被软件写操作。
   使用sys_cnt[63:0]+wdog_offset_value[31:0]作为新的比较阈值CompareValue。
3. 控制与状态寄存器WCS被软件写操作。
   使用sys_cnt[63:0]+wdog_offset_value[31:0]作为新的比较阈值CompareValue。
4. 比较初始阈值寄存器WCV被软件写操作。
   如果新的配置的compare_value[63:0]大于当前时间，则将新配置的compare_value[63:0]作为新的比较阈值CompareValue。否则，将使用sys_cnt[63:0]+wdog_offset_value[31:0]作为新的比较阈值CompareValue。

须知

WOR.offset_value[31:0]在sys_cnt 计数频率400MHz，最大的计数时间阈值大约10s。如果在需要大延时的时候，直接向64bit compare_value中写入新的时间阈值即可。

19.5.1.1.2 首次超时刷新（TimeoutRefresh）

当发生比较器第一次溢出（TimeoutRefresh）事件时，系统使用sys_cnt[63:0]+wdog_offset_value[31:0]的结果作为新的比较阈值CompareValue。

19.5.1.2 WDOG 溢出事件

比较器溢出事件TimeoutRefresh触发条件=(sys_cnt[63:0]> CompareValue[63:0])。

19.5.1.2.1 第一次 WDOG 溢出 WS0

1. 系统上电复位期间，WS0恢复默认值0。
2. 看门狗使能信号（WCS.enable）无效时，WS0跳变0。
3. 软件更新WDOG（Explicit watchdog refresh）时，WS0跳变为0。
4. 比较器第一次溢出（WS0=0&TimeoutRefresh）时，WS0跳变1。

19.5.1.2.2 第二次 WDOG 溢出 WS1

1. 系统上电复位期间，WS1恢复默认值0。
2. 看门狗使能信号（WCS.enable）无效时，WS1跳变0。

版权所有 © 海思技术有限公司

3. 软件更新WDOG（Explicit watchdog refresh）时，WS1跳变为0。
4. 比较器第二次溢出（WS0=1&TimeoutRefresh）时，WS1跳变1。

19.5.2 中断

19.5.2.1 中断寄存器

中断寄存器包括中断状态寄存器和中断清除寄存器。

- 中断状态寄存器WCS[ws0_status]和WCS[ws1_status]：读为高表示有产生中断；读为低表示未产生中断。
- 中断清除寄存器WRR：写任意值可清除WDOG的中断并重新开始计数比较。

19.5.2.2 中断类型

中断类型描述如下:

- 第一次溢出ws0_status：对非安全WDOG而言，WS0上报为LPI中断。对安全WDOG而言，WS0上报为1-N SPI中断，软件指定该中断运行于EL3中断层级。
- 第二次溢出ws1_status：对非安全WDOG而言，WS1不使用。对安全WDOG而言，WS1上报为1-N SPI中断，软件指定该中断运行于EL3中断层级。

19.6 编程接口

软件通过写WRR看门狗Refresh寄存器来更新看门狗比较阈值和清除中断，对该寄存器写入任意值可清除WatchDog的中断，并使WatchDog重新载入offset_value +sys_cnt_value值作为新的compare_value阈值开始和系统时间比较。

19.7 典型应用配置

无。

19.8 初始化

19.8.1 看门狗初始化

看门狗在上电复位撤离后初始化配置如下步骤。

步骤1 配置WDOG使能寄存器WCS[enable]为0，关闭看门狗。

步骤2 配置WDOG阈值补偿寄存器WOR。

步骤3 配置WDOG初始比较阈值寄存器WCV_L和WCV_H 。

步骤4 配置WDOG使能寄存器WCS[enable]为1，打开看门狗。

----结束

版权所有 © 海思技术有限公司

19.9 注意事项

在软件切换SYS_CNT计数频率期间，需要软件先无效看门狗的WCS[enable]关闭看门狗，在配置完成后重新使能看门狗，看门狗根据新的compare值开始监控。

19.10 差异说明

无。

19.11 时钟复位

- WDOG工作逻辑时钟上电后默认为打开状态。
- 各寄存器的值均不受系统非上电复位影响。

版权所有 © 海思技术有限公司

20 Cross Station 模块（AA）

20.1 概述

Ring Bus是芯片系统内部各个Subsys进行交互访问控制的总线模块。Ring是由多个节点互联而成，这些节点即为Cross Station模块，每个Cross Station（Ring节点）主要完成的功能是保证芯片系统内部多个主设备按照一定的顺序正确且高效的访问从设备。

Access Agent（AA）模块则为Cross Station节点接入前的Buffer模块，对于Subsys完成传输的汇聚和分发，对于CPU Cluster则完成数据位宽的合并与分拆。

20.2 特性

Cross Station（包含Access Agent模块）具有以下特性：

- 时钟与复位规格
  - Cross Station不支持单独软复位，仅支持管脚复位或整系统软复位。
  - CPU Cluster与Cross Station的频率比为1:1。
  - Access Agent与Cross Station同频运行。
- 支持基于CHI协议的定义，完成Core、IO及Accelerator与L3 Cache、DDR及Register之间的互联互通。
- 每个Cross Station节点提供1~3个完全对等的CHI接口。
  - 每个CPU Cluster独享1个CHI接口。
  - 每个LLC独享1个CHI接口。
  - 每个HHA独享1个CHI接口。
  - 各个Subsys内部有汇聚和分发模块，使整个Subsys收敛到统一的CHI接口。
- 实现36bits物理地址的Cacheline交织或全地址交织，将流量均分到各个LLC。
  - 支持Cacheline交织（128Bytes粒度，使用addr[7]判断）。
  - 支持全地址交织（addr[35:7]进行异或运算）。
- Cross Station上数据交换（data channel）位宽为256bits。

版权所有 © 海思技术有限公司

- 支持32个动态地址窗口（DAW），支持DDR、Subsys等设备的地址灵活映射到64GB的任意位置；系统空间按照DDR、Subsys分类，通过Target ID（TgtID）区分。
  - 任意动态窗口可定义任意TgtID设备在64GB空间中，16MB~64GB大小的范围。
  - 启动后，主核可根据应用设定各个TgtID在系统中实际地址分布。
  - 当命令访问的地址命中2个或以上不同TgtID的DAW时，上报中断，并提供问题窗口的序号供软件查询。
  - 当命令访问的地址未命中任何DAW时，上报中断,在mn有应答。
- 通过对CPU Cluster状态判断等软件流程，可以在对应Cluster无访问时关闭该Cluster与Ring节点的链路，然后关闭Cluster的时钟，满足低功耗要求。
- 每个Station提供4个48bits的Counter，支持对如下的传输统计功能（详细请参考Cross nManager）。
  - 支持对4类常规操作的个数汇总统计：Read，Write，Copyback，Maintain。
  - 支持对2类特殊操作的个数汇总统计：Barrier，DVM。
  - 支持操作响应个数的统计。
  - 支持各个指定Opcode的操作个数统计。
  - 各个计速器计满会卷绕，并上报中断。
- 支持实时捕捉功能（Trap）。
  - 支持Station端口RX方向4个CHI通道的信息捕捉。
  - 支持Station端口TX方向4个CHI通道的信息捕捉。
  - 信息捕捉以Addr，TgtID，SrcID，LPID等信息作为精确匹配的条件，也可以进行部分信息的模糊匹配。
  - Trap成功后会上报中断。
- 支持传输延时的统计。
  - 延时统计包括最大延时统计和平均延时统计。
  - 统计指定类型操作的传输延时，延时统计起始为请求进入Ring端口直到响应返回输出Ring端口。
  - 统计方式为采样统计，在实际中使用需要对一段业务时间进行采样的结果才有参考意义，采样时间越长，结果越准确。
  - 平均延时的统计结果通过采样的总传输延时和采样总个数计算得出。
  - 总延时统计和总个数统计计数器计满后会卷绕，并上报中断。
  - 总延时统计和个数统计相互匹配，当读取延时信息将锁定个数信息；同理，当读取个数信息，则锁定延时信息；只有当两组值都被读取后，统计值将继续更新。
- 支持各个Station工作状态的获取。
  - 支持各个Port的工作状态获取，包括credit，Buffer等状态。
  - 支持各个Cross内部的工作状态获取，包括credit，传输valid等状态。

20.3 关键术语

Cross Station相关的关键术语表如**表20-1**所示。

版权所有 © 海思技术有限公司

表 20-1 Cross Station 术语表

术语	描述
Link Active	链路状态，通过REQ/ACK握手方式激活链路后，Link进入活动状态，通过ACTIVE信号表示。
Link Credit	链路传输容限，由Rx端向Tx端发送Credit，以单Cycle为单位，一个Credit对应一次传输，Tx接收到的Credit总数表示Tx可以发送传输的次数。
Cross Station	Ring上的节点，根据路由信息对传输进行汇聚和分发。每个节点最多有3组接入口，每组接入口支持4对收发。
地址交织	对Master发出命令的物理地址进行移位或重映射，使得Master对各个L3 Bank的访问流量尽可能均衡，交织的性能与业务模型强相关。
DAW	动态地址窗口，对某个Device（Mem）可访问的空间大小进行配置。“动态”指上电启动后可配置，在业务运行过程中禁止改变已配置窗口的大小。
AA_CPU	用于CPU Cluster与Ring对接的Access Agent，主要用于数据位宽的转换。
AA_SUBSYS	用于各个Subsys与Ring对接的Access Agent，实现Subsys内部Master和Slave与Ring之间的分发汇聚，以及解决上下Ring的分频问题。

20.4 遵循协议

Cross Station各接口遵循标准的ARM CHI协议，支持Req、Snp、Rsp、Dat 4对收发通道，完成Link层的对接。

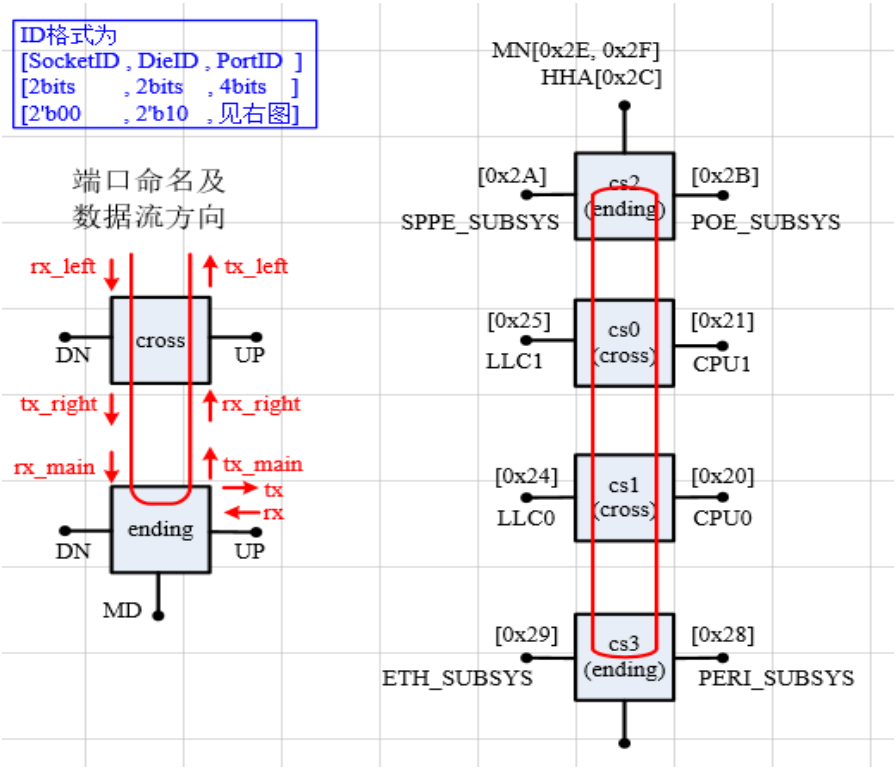
20.5 功能描述

20.5.1 典型应用

图20-1为Ring拓扑结构。每一个Cross Station对应Ring的一个交换节点。Cross Station之间的数据传输通过图中的left/right通道完成，称作中继口。Cross Station支持1至3组Port接入，即每一个节点都可以挂载1至3个Subsys/Cluster/L3 Bank，称作接入口。

版权所有 © 海思技术有限公司

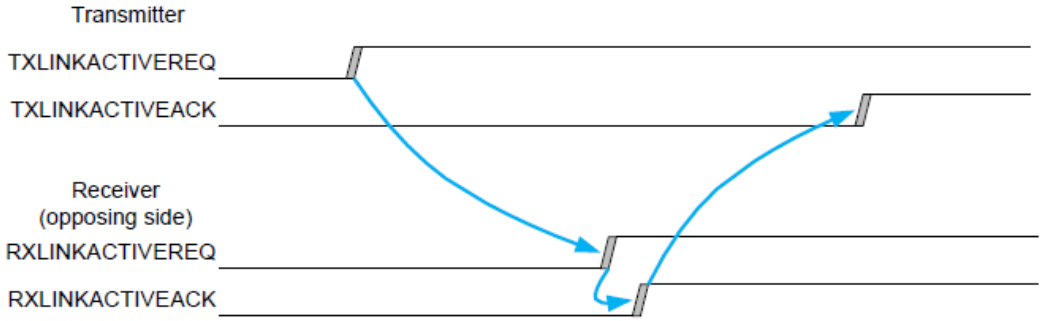
图 20-1 Ring 拓扑结构图



20.5.2 Link 层通信

Ring属于传输系统的链路层（Link），在传输开始前需要通过握手方式建立链接。如图20-2中所示，Tx发起链接Req，Rx端在接收到Req的一段时间后，返回Ack，Tx接收到Ack后则握手完成，表示图20-2当前两个节点之间的通路已激活。

图 20-2 Link 握手过程

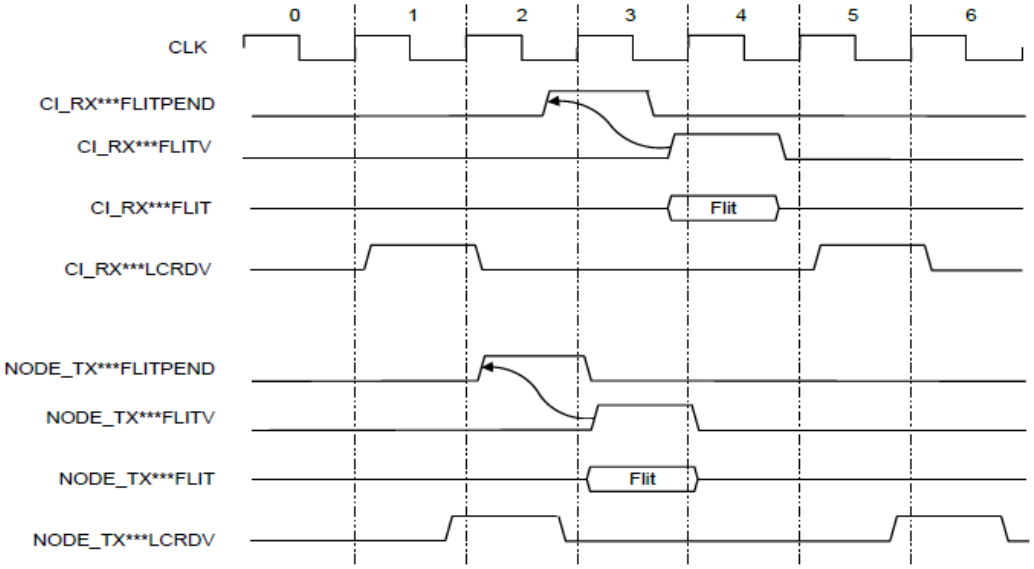


当Rx和Tx端的链接建立后，则通过Credit的方式进行数据传输。Rx端向Tx端发送单拍Credit脉冲，Tx收到Credit后产生Pend信号表示Tx可能有传输发出，Tx在Pend置低后发出Valid和Flit表示当前传输已发出，Rx将收到Valid和Flit存在本节点，如果Rx接收到的传输准备发至下一节点，则Rx继续向Tx发出Credit脉冲。

如图20-3表示Rx当前仅有一个空闲Buffer，所以Credit也仅有一个Cycle有效。如果Rx有多个空闲Buffer，则Rx可以一直发送Credit给TX。即Credit的发送和接收与实际数据传输（Flit传输）是分离的，仅需满足Tx有Credit的时候发出Flit至Rx即可。

版权所有 © 海思技术有限公司

图 20-3 Ring 数据传输过程



须知

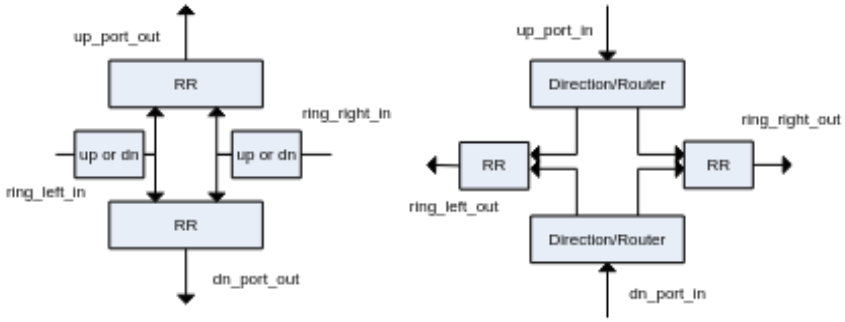
系统上电后并解复位后，Cluster和Subsys均会自动建链；对于断链功能主要用于Cluster复位。Cluster的断链（掉电，所有Core及L2进入WFI状态等）请查阅Cluster相关文档。

20.5.3 汇聚分发

如图20-4所示，图左为下Ring的逻辑，传输在各个节点间通过中继口传输。当传输送达目的地时，判断是发往Up Port或者Dn Port（对于Ending节点还有Md Port），然后将传输寄存到出口的Buffer。Left/right两边的中继口有独立的寄存Buffer，两路汇聚后，经过RR仲裁将传输送出Ring。

图右为上Ring的逻辑，传输从各个接入口发往中继口传输。一个新的传输根据源ID（SrcID）和目的ID（TgtID）的判断比较，选择最短路由，从left/right中继口送往下一节点。Up/Dn Port如果同时发往同一方向，则进行RR仲裁选其一进行传输，未被仲裁的传输则继续寄存在入口的Buffer等待下一次仲裁。

图 20-4 传输的汇聚与分发



版权所有 © 海思技术有限公司

说明

对于直通传输，如Up Port的传输是直接发往当前节点的Dn Port，因为物理实现的限制，无法直接透传。直通传输过程，和跨节点传输类似，传输先经过仲裁上Ring节点，进入节点后判断为到达，然后立即寄存至下Ring的Buffer后出Ring。即直通过程一般在Ring节点上额外寄存1拍即可。

20.5.4 路由选择

路由选择通过对源ID（SrcID）和目的ID（TgtID）计算得到。每个SrcID / TgtID均为8bits信号，其中ID[7:6]表示Socket ID，ID[5:4]表示DIE ID，ID[3:0]表示同一DIE内的Port ID。通过ID信息即可判断出数据传输的方向和传输所经过中继节点的次数。

考虑简化组合逻辑实现的复杂度，尽可能以最短路径为原则进行传输，由此决定传输进入Ring以后的传输方向。

由于Ring的无阻塞传输方式（即每一拍都在传输，没有缓存暂停），所以传输每到一个新的节点都必须比对当前节点的ID和TgtID是否相同，相同则尝试落地，不同则继续传输。

说明

目前Hi309a不支持多Socket或者多DIE的互连，一个DIE内统一使用4bits Port ID区分各个接入端口。

20.5.5 防 Deadlock 机制

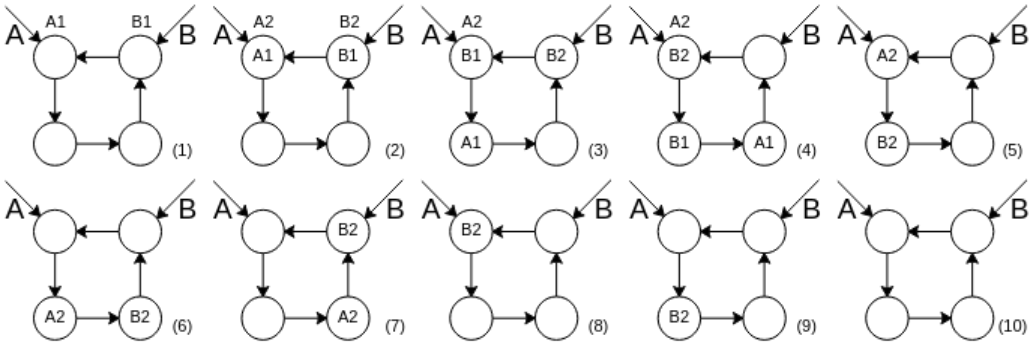
对于Ring的设计，首要保证Ring不会出现死锁导致整系统挂死。而Ring出现死锁的本质就是传输路径头尾相接形成回路。解决Ring死锁方式有多种，当前Ring设计采用的是循环传输的过程，如图**图20-5**所示。

图20-5示意了Ring的传输机制。假设两个节点A，B均向对角节点传输两个操作，分别为A1，A2，B1，B2。子图（2）中，Ring节点接收了A1，B1两个操作，A1，B1在后续的每一拍时钟都向下一个节点跳步；子图（3）中，A1，B1各向前移动一个节点，此时A节点被B1占据，所以A2无法传输上Ring，并在入口处等待空闲；子图（4）中，A1，B1都到达目的地，则下一拍各自落地，此时A节点被B2占据，A2继续等待；子图（5）中，A2等待到空闲，则发送至Ring传输，B2到达目的地，但因为目的地反压，B2未能落地，所以下一拍继续向下一节点跳步，循环传输。子图（6）-（9）显示B2在Ring上重传的过程，直到B2再次回到目的地判断可以落地，则B2送出Ring。

所以，只要保证Ring上的传输不断地向下一节点跳步，则Ring不会出现挂死。如果到达目的地不能落地，则继续向前跳，重新循环。未能落地的传输每次经过出口Port时均会尝试下Ring，当出口空闲可以接收数据时，该传输才能从对应Port输出。反压时间不定，取决于出口对接模块的接收能力。当传输下Ring后，空闲的节点可以接收新的传输。

版权所有 © 海思技术有限公司

图 20-5 Ring 上各节点的循环传输



20.5.6 地址交织

系统中高速存储模块，如DDR/LLC均为分布式结构，即通过多通道/多处理单元对数据进行并行处理，提升带宽性能。为了充分发挥分布式结构的优势，尽可能地将访问按照地址交错平均分配到各个单元。

如当前系统有2个LLC Bank，如果按照连续地址的高位进行划分，有可能所有操作都会集中到同一个Bank中，有效理论带宽仅为最大带宽的50%。

通过地址交织，可以将软件进程中一段时间内访问的集中地址分散到各个通道/处理单元，实现数据的并行处理。

20.5.6.1 L3 Cache Bank 交织

一般而言，多Bank结构按照高位地址划分，而一段软件程序经常连续对某段Memory地址进行连续访问。对于Cacheable的操作，这一段连续访问很有可能就直接落在同一个Bank中，其结果就是一个Bank的访问量很大，造成访问拥塞，平均访问延时增大、带宽下降，其他Bank则处于空闲状态。为了充分发挥多Bank结构的优势，需要将软件可能出现的地址连续访问的行为尽可能分布到不同Bank上。地址交织则实现将连续物理地址交织分散到不同的LLC Bank中。

全地址空间为36bit，L3 Cache的最小访问单元为1个Cacheline，即128Byte，所以需要对高29bit地址进行交织。地址交织过程在Access Agent中完成，共支持两种模式：Cacheline交织及全地址交织，默认使用全地址交织。

1. 全地址交织

   使用全地址交织方式，将所有地址进行两两异或（XOR）操作后，物理地址重映射到2个L3 Cache Bank上。

   tmp_num[1:0] = Addr[35:34]^ Addr[33:32]^ ……^ Addr[9:8]^ [Addr[7],0b0][1]

   Bank_num = tmp_num[1] ^ tmp_num[0]

   说明

   [1]表示：电路逻辑因为CPU发出的地址还是44bits，所以全地址交织按照44bits实现，软件要保证地址高8位为0。

   全地址交织将任意粒度的连续地址均匀分配到2个L3 Cache，达到均衡流量的目的。

2. Cacheline交织

版权所有 © 海思技术有限公司

Cacheline交织即以Cacheline为粒度进行地址交织。Cacheline为128Bytes，使用Addr[7]可将连续Cacheline访问的操作分配到不同的L3 Cache中。

Bank_num = Addr[7]

须知

各个Access Agent的交织配置需要统一，否则会导致地址访问的错乱。Access Agent的交织配置会连接至L3 Cache和HCCS Home Agent共同使用，确保全局的统一。

20.5.7 DAW 配置

DAW功能可以将各个Master或CPU对任意设备或内存空间的访问，重新映射到新的设备或内存空间，并且可以选择地址映射的窗口大小。

DAW功能集成在Access Agent中的Request通道，输入36bit的Address，使用[35:24]位进行TgtID的生成。地址窗口大小从16MB~64GB可配，窗口大小按照16MB乘以2的N次幂计算。

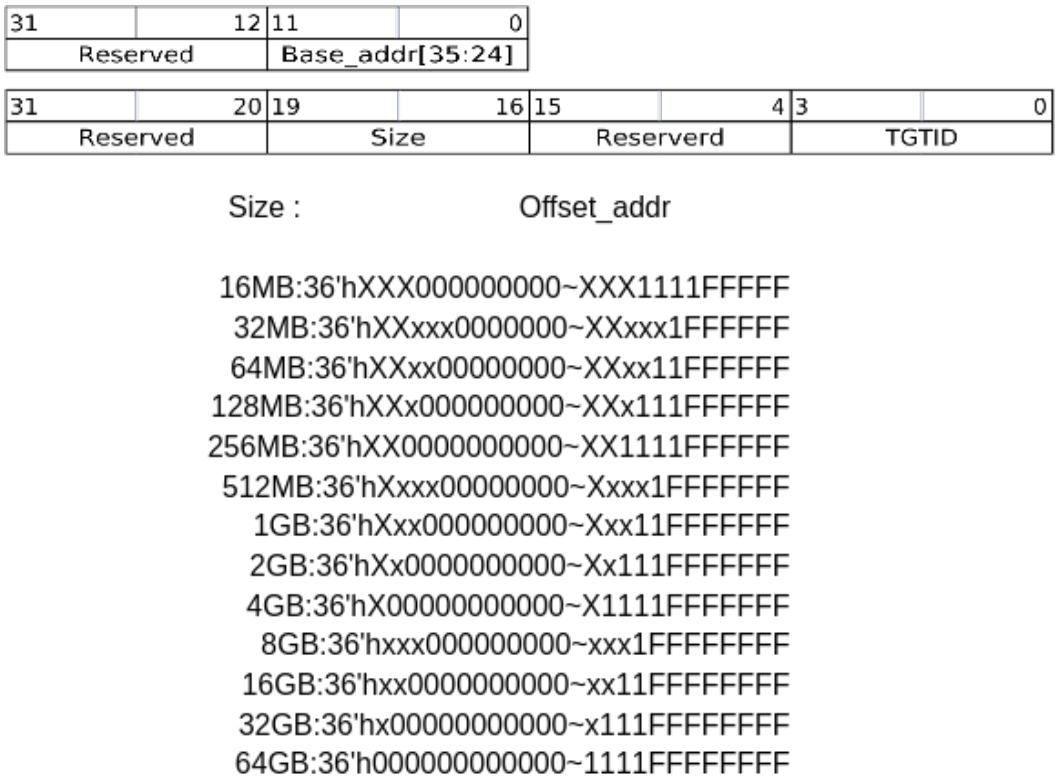
须知

Access Agent共计32组DAW配置寄存器，可同时映射32个地址段到不同的设备，包括DDR，Subsys。地址段由起始地址和映射窗口大小决定，且与窗口大小对齐。如不同设备的DAW有重叠，可能使命令访问错误的设备导致不可预知的后果。

配置DAW除了不能有相同地址段映射到不同设备，还需要注意映射窗口大小和所配置基地址的匹配问题，基地址并非任意，必须与窗口大小边界对齐。通过图20-6说明DAW的配置有效值。

版权所有 © 海思技术有限公司

图 20-6 DAW 配置有效值



DAW配置寄存器为两组32bits寄存器，图20-6中小写“x”表示可配置单bit信息，大写“X”表示可配置4bits信息。为了清楚表示有效配置，通过下划线标识的红色字体表示将1个十六进制数展开为4bits二进制数形式。

Size和Base_addr的匹配是指，Size指定的大小要和Base_addr对齐。当Size域配置为设定值后，某些Base_addr位已经固定为0，即非对齐的地址位无论写入任何初值都认为是0。所以，建议地址划分时，先决定各个Device的空间大小再确定基地址的配置。

以Size为0b1010为例，即地址空间为16GB，其可配置的基地址只有Addr[35:34]，Addr[33:24]无论配置何值，都默认为0。假设一组DAW的配置，基址为0x0000_0600，size和id为0x000A_000C。配置期望基址为0x6_0000_0000以上的16GB空间均访问TGTID为C的设备，但因为基址和size不对其，所以实际上该配置生效的基址为0x4_0000_0000而不是0x6_0000_0000，且地址空间即有效偏移地址为0x4_0000_0000至0x7_FFFF_FFFF。

说明

一级DAW译码生成的TgtID为整芯片统一规划，且粒度到DDR或某一个Subsys。Subsys内部由Dispatch的二级DAW完成译码。具体可参考地址映射文档。

对于DAW发现冲突时，将上报中断，并记录发生冲突的DAW窗口号，供查询分析。

对于未在一级DAW中配置的地址段，将全部映射到MN的DeFault Slave上。对于未在二级DAW中配置的地址段，将映射到当前Subsys的Default Slave返回DECERR响应。

20.5.8 数据统计

Cross Station提供额外的统计功能，以统计传输相关的性能数据，以及在系统异常时提供状态信息用于定位问题。

版权所有 © 海思技术有限公司

说明

DFX功能有一个总使能开关，实际上该使能开关是控制DFX主体逻辑的时钟门控，所以只有时钟门控使能后，各个DFX子功能才可以开始配置和使用。

20.5.8.1 事件统计

每个Station提供4个48bits的事件统计计数器，每个计数器有公共统计的事件，也有各自独立的统计事件。

Cross Station内部可统计的事件，如表20-2所示。

说明

每个事件统计计数器都可以配置初始值，计数器清零则通过配置全零值实现。每个计数器都有独立的使能开关，在事件类型配置完成后，开启计数器开关，开始统计相应事件。计数器为48bits，计数满后会卷绕，并上报中断。

须知

统计寄存器和事件的绑定，可以在系统运行中任意变换，但是切换统计事件前，需要先停止统计，将统计值清空再切换，避免统计值混乱。

表 20-2 Cross Station 内部统计事件功能列表

Cnt 编号	统计事件号	事件名称	备注
公共事件，所有Cnt均可统计	0	读命令	命令包括 ReadShared,ReadClean,ReadOnce,ReadNoSnp,ReadUnique, CleanUnique,MakeUnique
	1	写命令	命令包括 WriteUniquePtl,WriteUniqueFull, WriteNoSnpPtl,WriteNoSnpFull
	2	Copyback	命令包括（Copyback） Evict,WriteEvictFull,WriteCleanPtl, WriteCleanFull,WriteBackPtl,WriteBackFull
	3	Maintain	命令包括（Cache Maintenance Operation）CleanShared, CleanInvalid,MakeInvalid
	4	Barrier	命令包括EOBarrier，ECBarrier
	5	DVM	命令包括DVMOp

版权所有 © 海思技术有限公司

Cnt 编号	统计 事件号	事件名称	备注
	6	snp命令	命令包括 snpshared,snpclean,snponce,snounique,snpcleanshared, snpcleaninvalid,snpmakeinvalid
	7	snpdvm	-
Cnt0	8	读响应	指RN发出的CompAck
	9	readshared	-
	10	readunique	-
	11	writenosnp	partial and full
	12	cleanshared	-
	13	snpshared	-
	14	snpinv	指HCCS的SPECUNLOCK
	15	snpcleanshared	-
Cnt1	8	写响应	指SN发出的Comp或CompDBID
	9	readclean	-
	10	cleanunique	-
	11	evict	-
	12	cleaninvalid	-
	13	eobarrier	-
	14	snpclean	-
	15	snp响应	-
Cnt2	8	Retry	-
	9	readonce	-
	10	makeunique	-
	11	writeevictfull	-
	12	writeback	partial and full
	13	makeinvalid	-
	14	snponce	-

版权所有 © 海思技术有限公司

Cnt 编号	统计 事件号	事件名称	备注
	15	snpmakeinvalid	-
Cnt3	8	PcrdGrant	-
	9	readnosnp	-
	10	writeunique	partial and full
	11	writeclean	partial and full
	12	ecbarrier	-
	13	snpunique	-
	14	snpcleaninvalid	-

20.5.8.2 延时统计

在每个Station节点的各个入口都有一组延时统计寄存器，对指定类型的操作进行传输延时统计。统计结果包括传输最大延时，传输延时总和，传输个数总和，平均延时可通过延时总和除以个数总和计算得出。

可配置统计的类型：读操作，写操作，CopyBack操作，Maintain操作，Snp操作。

说明

对传输延迟的统计，都是基于不断对传输进行抽样统计获得的统计值，其最终值和真实值存在一定统计误差，但随着传输发起数量的增加，误差逐渐减少。

CPU可以在长时间运作中实时监控传输平均延迟和最大延迟，在稳定运行中，这两个值也将基本保持稳定；CPU可以设定一定的水线进行监控，如果发现有异常的延迟变化，则系统也相应出现了性能或流量的异常波动。

20.5.8.3 状态统计

Ring作为传输通道，不对数据进行处理，仅负责将传输从一个点到另一个点转发。一般而言，Station的状态是无需关注的。当系统异常时，可以通过Station的状态判断是否有传输驻留在Ring，从而判断可能出现问题的Master或Slave。

Ring内部可读取的状态包括：

- 上Ring之前的缓存Buffer内部是否有传输无法上Ring。
- Ring节点间是否仍有有效传输未下Ring。
- 下Ring后的缓存Buffer内部是否有传输无法发出到对接设备。
- 建断链以后，各个节点的Credit数量是否正常。

20.5.9 Trap 传输信息

所有Cross Station的各个接入口均可独立配置其RX和TX通道的Trap功能。设置期望匹配传输的Address，SrcID，TgtID，LPID，Opcode信息，可以进行精确匹配和模糊匹

版权所有 © 海思技术有限公司

配（Masked）。当符合预设条件的传输发生时，Station可触发Trap动作，将传输的主要信息记录。Trap动作发生时触发中断通知CPU，通过读取寄存器获取Trap的传输信息进行问题定位分析。

说明

Ring端口的RX和TX方向均可触发Trap功能，每个方向的4个CHI通道均可独立触发。对于RX方向，预设的匹配ID为传输的TgtID；对于TX方向，预设的匹配ID为传输的SrcID。

20.6 软件指南

20.6.1 初始化

Cross Station和Access Agent无需初始化默认即可全速工作。如果软件部署涉及到地址段的划分修改，则需要在系统Boot过程中，先修改好DAW，L3 Bank交织，DDR交织等工作方式后，再加载操作系统和业务软件。

须知

DAW，L3 Bank交织模式不可动态切换，只能在Boot程序中修改；

同一芯片内，各个节点的DAW和L3 Bank交织方式都必须统一。

违反以上原则都将导致数据读写出错，系统崩溃。

20.6.2 软复位

除非整系统复位，否则在系统正常运行Cross Station不允许软复位。

20.6.3 DAW 配置

一般情况无需配置DAW，只有当业务对地址空间有特定需求的时候，才会按照新的地址空间分配进行修改。

一级DAW配置在各个Access Agent的寄存器中，仅能通过DJTAG接口访问，配置流程如下：

步骤1 选择Access Agent所在的DJTAG链。

步骤2 使能所选链上所有的Access Agent模块，同时配置所有寄存器。

步骤3 对所选链的操作配置为写操作。

步骤4 向DJTAG“偏移地址寄存器”写入DAW的偏移地址值。

步骤5 向DJTAG“配置数据寄存器”写入DAW的具体数值。

步骤6 触发DJTAG功能的访问开关，开始自动写DAW寄存器。

步骤7 查询访问开关，看是否已完成一次写操作。

步骤8 重复**步骤4**-**步骤7**，直到所有DAW配置完成。

----结束

版权所有 © 海思技术有限公司

需要查找SysCtrl或SubCtrl对应的nManager文档确定各个DIE的DJTAG寄存器基地址和偏移地址；查找Cross Station（AA）对应的nManager确定各个DAW配置寄存器的偏移地址。

20.6.4 性能统计操作

通过对系统访问延时的采样统计，可以获取系统性能数据。寄存器的读写流程均由DJTAG完成。Ring性能统计的步骤如下：

步骤1 选择需要统计的节点，使能dfx_en开启DFX时钟门控。

步骤2 配置延时统计的传输类型。

步骤3 使能statistic_en，开始统计传输延时；延时统计为采样统计，统计时间越长，统计值越接近真实。

步骤4 读取总采样延时，总采样个数，最大延时信息。

步骤5 计算得出平均延时。（最大延时直接读取无需计算）

----结束

例如要统计CPU Cluster0的Wr延时信息，下述步骤直接对Cross Station（CS）寄存器进行说明：

步骤1 选择序号为0x0的链，即CS链。

步骤2 往偏移为0x0的CS寄存器bit[0]的dfx_en域写0b1，开启DFX时钟门控。

步骤3 因为CPU Cluster0在CS0的UP口，所以往偏移为0x70的CS寄存器bit[2:0]的写0b001。

步骤4 往偏移为0x0的CS寄存器bit[2]的statistic_en域写0b1，启动统计功能。

步骤5 按需保持业务运行一段时间后，分别读取CS寄存器0xA00（总延时低32bits），0xA04（总延时高32bits），0xA08（总延时对应命令个数），0xA0C（最大延时）。

步骤6 最大延时已有，平均延时由总延时64bits除以总命令个数得到。

----结束

20.7 性能

20.7.1 吞吐率（Throughput）

说明

吞吐率的计算均为理论最大值，仅作参考。实际情况因阻塞和Outstanding能力会导致带宽下降，并且在不同的业务场景中，运行频率不同，总线的带宽吞吐率也不同。本小节按照典型频率400MHz计算各项指标。

20.7.1.1 数据吞吐率

按照升压后运行频率计算：

数据通道为256bits位宽，Ring采用单通道结构，Ring各个节点之间的数据传输为单拍传输，Ring节点间单方向最大数据传输理论带宽为：

256bits×1200MHz = 300Gbps = 38.4GB/s

版权所有 © 海思技术有限公司

20.7.1.2 指令吞吐率

Req和Snp均为单通道结构，Ring各个节点之间的数据传输为单拍传输，Ring节点间单方向最大指令吞吐率为：

1200MHz/1 = 1200Mcmd/s

20.7.2 延迟（Latency）

Cross Station对于存取访问的延迟（Latency），延迟计时点从命令或数据进入Cross Station到响应离开Cross Station，不包含Master、Slave的处理延迟。由于传输可能跨DIE甚至跨Socket，所以延时无法一一列举，表20-3说明了延时的计算方法。

表 20-3 Cross Station 存取延迟（Latency）

行为	延迟（CPU Cycles）
上Ring寄存	1
Ring节点之间传输	1
下Ring寄存	1
起点到终点的总延时	2+1*N，N为经过的总节点数

说明

以上延迟数据均为Ring非阻塞时的统计值，Access Agent会增加频率转换的延时。并且当Ring严重阻塞时，可能由上游Master或下游Slave引起，延迟无法估计。

20.8 DFX

20.8.1 Trap

通过Trap进行定位分析的步骤如下：

步骤1 配置dfx_ex域，开启DFX的时钟门控。

步骤2 配置需要匹配的传输信息和Mask位。

步骤3 运行程序，查询中断。

步骤4 中断触发后，读取Info寄存器获取传输信息。

步骤5 清除中断，准备下一次Trap动作。

----结束

20.8.2 Cross Station 挂死定位

如果出现Cross Station挂死，可通过如下两种方式进行调试定位：

- 使用JTAG接口，调用DJTAG模块访问Cross Station寄存器获取当前状态；

版权所有 © 海思技术有限公司

- 锁定程序范围后，配置Cross Station的Trap功能抓取req/snp/rsp/dat信息，对挂死出现前的命令逐条排查，最终定位到未完成的指令。

20.9 注意事项

操作Cross Station需要额外注意事项如下：

- Cross Station的Trap可以在芯片运行中任意时刻进行。
- Cross Station的地址交织、Low Power模式仅仅能在系统启动，DDRC的关断，需要在还未访问DDR空间前配置。
- 访问L3 Bank的地址交织无特殊使用限制。
- DAW的配置不能将同一段地址映射到两个不同的设备。
- 对于各个Master和Slave在进行软复位之前，必须完成当前驻留在Cross Station中的操作，如发出的操作一定要收到响应，接收到操作一定要给响应等；同理，在关闭CPU Cluster之前，软件必须确保所有操作已完成。

版权所有 © 海思技术有限公司

21 Dispatch

21.1 概述

Dispatch是Ring Bus的输出模块，对从Ring Bus收到的请求，向一组（N<=8）功能类似、总带宽和Ring Bus Station匹配的Register或IO Device进行传输分发和缓存。

21.2 关键术语

无。

21.3 功能特性

Dispatch具有以下特性：

- 1-to-N（N<=8）的Dispatch，将Ring Bus对各个Register和IO Device的访问进行分发和缓存，使得操作可以正确到达目的端。
  - 支持任意1/N（N=1,2,4,8）同步频率的设备分发。
  - 支持任意异步频率的设备分发。
  - 支持最多1个APB、2个AHB、5个AXI接口设备同时挂接在一个Dispatch上，APB、AHB只支持32bit数据位宽，AXI支持64、128、256bits数据位宽。
- Dispatch支持MEM属性Early WriteAcknowledge为有效的操作（Write Buffer），当Dispatch收到此类操作后，将直接返回CompDBIDResp的响应。
- 支持最多为8个的滑动地址窗口（DAW功能）：
  - 通过滑动窗口配置，可将Dispatch所挂接的TgtID设备分配在1MB~1TB空间中，窗口大小范围可配。
- 支持Barrier操作：
  - Dispatch收到EOBar后，直接返回Comp响应，顺序由Dispatch保证。
  - Dispatch收到ECBar后，Dispatch会根据操作的Srcid和Lpid来判断发送该Barrier操作的CPU核，Dispatch等待该CPU核在ECBar之前发送的所有操作完成后，再返回Comp响应。
- 支持SKYROS侧扩展SIZE为128B/192B/256B的读写操作。

版权所有 © 海思技术有限公司

- Dispatch会对相同地址范围内的读写操作进行保序，保序地址范围寄存器可配。
- Dispatch支持4级Qos（优先级），Qos等级划分从低到高为最低优先级（0~3），次低优先级（4~7），次高优先级（8~11），最高优先级（12~15）。
  - 操作进入Dispatch的入口队列后，Qos（优先级）高的操作会优先下发。
  - 当Dispatch对接的Slave端口出现拥塞，导致操作无法下发时，操作将会被Retry， Retry将按照操作的Qos（优先级）进行，Qos（优先级）低的操作将先被Retry。
- Dispatch收到非法操作后，将会上报中断，Dispatch记录第一个非法操作的操作类型信息和地址的第[40:24]信息，以及该操作是否来自于CPU。
  - Dispatch如果收到DAW窗口或VF窗口以外的地址操作，会认为是非法操作，操作不会下发到子端口，如果是读操作，则返回读数据全1，如果是写操作，写数据将直接丢弃。
  - Dispatch可以支持的操作类型为ReadNoSnp、WriteNoSnoPtl、WriteNoSnpFull、ECBarrier、EOBarrier, Dispatch如果收到不支持的操作类型，认为是非法操作，Dispatch只保证按照协议要求返回响应，不保证数据正确性。
- Dispatch收到非法操作后，返回的响应类型为ERROR或OK寄存器可配置。
- Dispatch支持各Slave端口未完成操作个数统计。

21.4 遵循协议

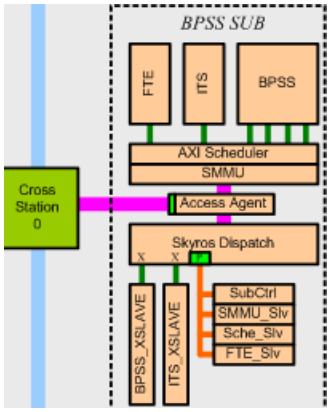
Dispatch与AA的接口为Skyros协议，Dispatch支持与 遵循AXI4 APB4 AHB协议的Slave口对接。

21.5 功能描述

21.5.1 应用框图

Dispatch的应用如图21-1所示。

图 21-1 Dispatch 应用图



版权所有 © 海思技术有限公司

Dispatch在总线中与各加速器的Slave对接，实现对各加速器的访问。一个Dispatch最多可以同时对接1个APB Slave，2个AHB Slave和5个AXI Slave。系统中可以根据加速器的种类和数量，例化多个Dispatch。

21.5.2 相关模块

Access Agent，以及所有的加速器Slave。

21.5.3 工作频率

Dispatch入口与RingBus对接，两者间采用同频同步接口，为总线频率的一半。

Dispatch出口与各个Register和IO Device相连，Dispatch与这些设备间可以工作在1/N同步模式或任意异步频率模式。

21.5.4 接口协议类型与数据位宽

Dispatch入口支持Skyros协议，数据位宽为256bit。

Dispatch出口支持最多1个APB、2个AHB、5个AXI接口设备同时挂接，APB、AHB只支持32bit数据位宽，AXI支持64、128bits、256bit数据位宽。

对于APB和AHB接口，地址位宽为32bit，Skyros的命令为44bit地址，命令地址在经过DAW完成译码后，地址高位会被直接截断，作为APB、AHB接口的地址下发。

21.5.5 Early WriteAcknowledge

Dispatch支持MEM属性Early WriteAcknowledge为有效的操作（Write Buffer），当Dispatch收到此类操作后，将直接返回Compdbid的响应。

其他写操作Dispatch将先返回DBID，等待真实写操作响应返回后，再返回Comp响应。

21.5.6 DAW 功能

子系统DAW分为两级，第一级在Ring Bus中实现，第二级在Dispatch中实现。Ring Bus将下发到Dispatch的命令归为一类，命令到达Dispatch后，由Dispatch根据DAW配置确定端口id，将命令最终归分发到各Slave中。

如果Disaptch接收到DAW配置以外的非法地址操作，Dispatch会将该操作送至内部Dafault从设备，并根据配置返回ERROR或OKAY响应，同时给出中断指示。

21.5.7 Barrier 处理

CPU发送的Barrier操作会传递到Dispatch。

Dispatch收到EOBar后，将直接返回Comp Resp，顺序由Dispatch保证。

Dispatch收到ECBar后，Dispatch会根据操作的Srcid和Lpid来判断发送该Barrier操作的CPU核，Dispatch等待该CPU核在ECBar之前发送的所有操作完成后，再返回Comp Resp。

21.5.8 Qos 和 Retry

Dispatch支持4级Qos（优先级），Qos等级划分从低到高为最低优先级（0~3），次低优先级（4~7），次高优先级（8~11），最高优先级（12~15）；

版权所有 © 海思技术有限公司

所有发往Dispatch的操作将首先进入Dispatch的入口队列，操作进入Dispatch的入口队列后，Qos（优先级）高的操作会优先下发。

当Dispatch对接的Slave端口出现拥塞，导致操作无法下发时，操作将会被Retry，Retry将按照操作的Qos（优先级）进行，Qos（优先级）低的操作先被Retry。

Barrier会被认为是拥有最高优先级的一类操作，Barrier操作也会被Retry。

Dispatch最多可Retry 64个不同源头的操作，对于同一个源头发送的操作，Disaptch最多可以Retry 256个。

系统中向Dispatch发送Barrier操作的Master只有MN模块，且MN除Barrier外不会发送其他操作到Dispatch，此类MN模块在系统中只有一个。Dispatch对于Barrier操作的Retry进行独立控制。

21.5.9 编程接口

无。

21.6 应用配置

21.6.1 滑动地址窗口（DAW）配置更新

Dispatch的控制寄存器存放在对应的Subctrl寄存器中，为了保证DAW的正确切换，D以更新窗口0为例，DAW的更新配置流程为：

步骤1 配置Subctrl的SC_DISPATCH_DAW_ARRAY0[8]为1，打开配置使能。

步骤2 配置SC_DISPATCH_DAW_EN[0]，打开或关闭窗口的DAW功能。

步骤3 配置SC_DISPATCH_DAW_ARRAY0[31:13]、SC_DISPATCH_DAW_ARRAY0[7:3]、SC_DISPATCH_DAW_ARRAY0[2:0]，在DAW窗口使能的时候，配置DAW窗口的DID，窗口大小，窗口基地址。

步骤4 配置Subctrl的SC_DISPATCH_DAW_ARRAY0[8]为0，关闭配置使能。

步骤5 为确保DAW修改成功，可再读取Subctrl的SC_DISPATCH_DAW_ARRAY0[8]，如果读到配置值为0，则DAW已修改成功。

----结束

其他窗口配置流程类似。

21.7 初始化

无。

21.8 注意事项

无。

版权所有 © 海思技术有限公司

21.9 DFX

Dispatch收到非法操作后，将会上报中断，Dispatch记录第一个非法操作的操作类型信息和地址的第[40:24]信息。

Dispatch如果收到DAW配置以外的非法地址操作，会认为是非法操作，操作不会下发到子端口，如果是读操作，则返回读数据全1，如果是写操作，写数据将直接丢弃。

Dispatch如果收到不支持的操作类型，认为是非法操作，Dispatch只保证按照协议要求返回响应，不保证数据正确性。

Dispatch收到非法操作后，返回的响应类型为ERROR或OK寄存器可配置。

Dispatch支持各Slave端口未完成操作个数统计。

21.9.1 故障在线检测

21.9.1.1 非法操作记录

21.9.1.1.1 异常命令中断状态记录

1. 读取DISP_INTRAW0/DISP_INTSTAT0（0x800/0x808）确认原始中断状态和屏蔽后中断状态。
2. 读取异常命令中断上报时记录的信息：

   a：读取DISP_ERR_ADDR_L/DISP_ERR_ADDR_H（0x840-0x844），查询错误地址状态信息。

   b：读取DISP_ERR_INFO0/DISP_ERR_INFO1（0x848-0x84c），查询异常命令的其他信息：包括Srcid/Lpid/Opcode等。
3. 确认异常命令类型：是disp不支持的Opcode还是DAW没有命中。

21.9.1.1.2 SLV 端返回错误响应中断记录

1. 读取DISP_INTRAW1/DISP_INTSTAT1（0x810/0x818）确认原始中断状态和屏蔽后中断状态。
2. 根据DISP_INTRAW1[15:8]，判断是哪个端口返回了error response。

21.9.1.2 未完成操作统计及 Trace

1. 读取DISP_BUSY（0x1000）的值，该值表示DISP内部状态。
   a. 如果读出结果为全0，表示DISP已经处于空闲状态；否则需要根据具体具体位进行分析。
   b. 如果DISP_BUSY[10]为1，则表示DISP内部Barrier fifo仍处于busy状态。
   c. 如果DISP_BUSY[9]为1，则表示DISP内部入口队列仍处于busy状态。
   d. 如果DISP_BUSY[8]为1，则表示DISP内部default slave仍处于busy状态。
   e. 如果DISP_BUSY[7:0]某位为1，则表示DISP内部对应端口仍处于busy状态。
2. 读DISP_PORTx_STAT （0x900-0x920），该值表示端口内部未完成命令计数。
   a. Port0:0x900；Port1:0x904；Port2:0x908；Port3:0x90c；Port4:0x910。

版权所有 © 海思技术有限公司

b. Port5:0x914；Port6:0x918；Port7:0x91C；Default Slave:0x920。
c. 如果DISP_PORTx_STAT[15:0]值为全0，表示对应端口没有未完成的命令。
d. 如果DISP_PORTx_STAT[15:8]值不为0，则表示对应端口未完成的读命令个数为该值。
e. 如果DISP_PORTx_STAT[7:0]值不为0，则表示对应端口未完成的写命令个数为该值。

3. 读DISP_PORTx_CUR_ADDR_L /DISP_PORTx_CUR_ADDR_H/ DISP_PORTx_CUR_ID（0x2000-0x2078），该值表示端口当前出口命令信息。
   a. Port0:0x2000-0x2008；Port1:0x2010-0x2018；Port2:0x2020-0x2028；Port3:0x2030-0x2038；Port4:0x2040-0x2048； Port5:0x2050-0x2058；Port6:0x2060-0x2068； Port7:0x2070-0x2078。
   b. 根据DISP_PORTx_CUR_ADDR_H/L，表示对应端口当前出口命令的地址。
   c. 如果DISP_PORTx_CUR_ID，表示对应端口当前出口命令的Srcid+Lpid。
4. 初步分析：
   a. 如果1、2读出的结果都为0，表示DISP命令都处理完了，且内部已经处于Idle状态。
   b. 如果1读出的结果不为0，2的结果也不为0。表示对应端口有未完成的命令，那么需要关注对应端口下接的slave状态；同时根据3得到当前出口命令的信息。
   c. 如果1读出的结果不为0，2的结果为0。那么表示DISP内部各端口的命令都已经完成，但各端口内部仍有fifo或者Buffer有数据残留。

21.9.2 故障离线检测

21.9.2.1 Trap

对于已经定位到具体操作的问题，可以通过DISP的TRAP功能跟踪特定的传输。

步骤1 配置DISP_TRAP_SEL，选择需要TRAP的端口及读写类型。

步骤2 配置DISP_TRAP_REVERSE和DISP_TRAP_ORDER，指定TRAP的工作模式。

步骤3 根据期望捕捉的数据包，配置DISP_TRAP_CMD_COMPx、DISP_TRAP_CMD_MASKx、DISP_TRAP_RSP_COMP0和DISP_TRAP_RSP_MASK0，作为匹配判断的依据。

步骤4 配置DISP_DFX_CTRL. disp_dfx_icg_en为1 ，打开TRAP模块功能时钟。

步骤5 配置DISP_TRAP_CTRL打开TRAP功能。

步骤6 如果匹配成功，则有中断状态记录。待中断上报，或者读取DISP_INTRAW1[1:0]判断是否已经捕捉成功。

步骤7 TRAP成功后，读取DISP_TRAP_CMD_INFOx和DISP_TRAP_RSP_INFO0寄存器获取命令的详细信息。

----结束

21.10 性能指标

版权所有 © 海思技术有限公司

21.10.1 延迟（Latency）

表 21-1 延迟统计

评估场景描述	评估数据（unit cycle）		备注
	Min	Max	
RNS_Req -> Arvalid	2	-	延迟统计基于入口和出口为同频同步的场景。
RNS_Req -> ReadReceipt	2	-	
Rvalid -> RNS_Data	3	-	
WNS_Req->Awvalid	2	-	
WNS_req->DBID	3	-	
Rx_Data->Wvalid	3	-	
Bvalid->Comp_Rsp	3	-	

版权所有 © 海思技术有限公司

22 Scheduler

22.1 概述

Scheduler是对一组（N<=8）功能类似、总带宽和Ring Bus Station匹配的Accelerator或IO进行传输调度和汇聚，然后再通过SMMU接入Ring Bus的接入模块。

22.2 关键术语

表 22-1 Scheduler 关键术语描述

术语	描述
Scheduler	device的传输进入总线环前的预汇聚模块，将一组总带宽和总线环节点匹配的device进行调度汇聚，然后再接入总线环。
WRR	Weighted Round-Robin加权轮询算法。
Outstanding	Outstanding表示设备发出请求后，在收到响应前能继续发出其它请求的数量。

22.3 功能特性

Scheduler具有以下特性：

- 支持N-to-1（N<=8）端口调度和汇聚，然后再通过SMMU接入Ring Bus。
  - 支持任意1/N（N=1,2,4,8）同步频率的设备接入。
  - 支持任意异步频率的设备接入。
  - 支持32bit、64bit、128bit数据位宽的设备接入。
  - Scheduler和SMMU之间采用同步接口。
- 根据接入设备的需求，可在Scheduler入口处配置Splitter组件，将跨Cacheline地址的操作（数据量可以大于1个Cacheline大小）切分成多个Cacheline地址内的子操作，保证到总线的实际操作都在一个Cacheline内部。

版权所有 © 海思技术有限公司

- 支持WRR调度算法，每个端口权重寄存器可配置。
- 支持端口Qos（优先级）可配置，该配置默认不使能，此时Scheduler直接透传加速器接口信号自带的Qos（优先级）信息。
- 支持每个加速器接口读写outstanding 0~255独立可配置。
- 每个Scheduler支持丰富的传输统计功能。
  - 支持对Scheduler每个端口的未完成的读写操作进行统计。
  - 每个Scheduler上支持对总传输数量进行统计。
  - 每个Scheduler上支持对总线阻塞的次数进行统计，包括：调度失败导致的阻塞次数、总线拥塞导致的阻塞次数等。
  - 每个Scheduler上支持对发生split的传输数量进行统计。
- 片内CPU Core可通过总线的访问Scheduler的寄存器接口、片外JTAG接口可通过DJTAG BUS访问Scheduler的寄存器接口。

22.4 遵循协议

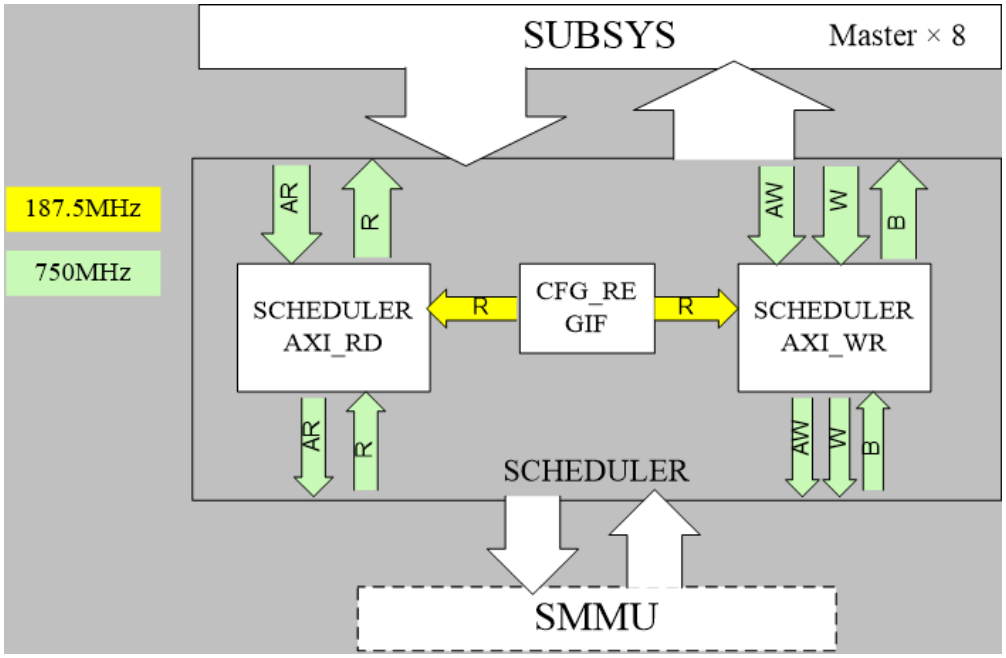
Scheduler接口遵循APB3 、AXI4协议。

22.5 功能描述

22.5.1 应用框图

Scheduler的应用如图22-1所示。

图 22-1 Scheduler 应用图



版权所有 © 海思技术有限公司

Scheduler是AXI Master接口的合路仲裁模块，用以将多个AXI Master设备合并为一个端口，进行WRR带宽调度，最后转换传输为ACE-Lite传输至SMMU再到Ring上，如图22-1所示。

22.5.2 相关模块

在系统中，Scheduler会和所有加速器的AXI Master对接。

22.5.3 工作频率

ACC和IO可以通过1/N同步模式或任意异步频率接入Scheduler。Scheduler根据配置在入口处对各时钟域的信号进行处理。

Scheduler命令通道和数据通道工作在同步模式，频率比为1:1。

Scheduler出口与SMMU对接，两者间采用同步接口。

22.5.4 接口数据位宽

Scheduler支持数据位宽为32bit、64bit、128bit的设备接入，出口数据位宽为128bit，对于数据位宽为32bit和64bit的设备，Scheduler内部完成对相关数据的搬移处理。

22.5.5 跨 Cacheline 切分

根据接入设备的需求，可在Scheduler入口处配置Splitter组件，将跨Cacheline地址的操作（数据量可以大于1个Cacheline大小）切分成多个Cacheline地址内的子操作，保证到总线的实际操作都在一个Cacheline内部。

22.5.6 WRR 调度

Scheduler支持WRR调度。

IO和ACC子系统内部均为类型相似的传输设备，Scheduler以传输结构相似的模块为子系统分组进行调度，以流量进行权重Weight分配，然后进行WRR调度。

Scheduler中各端口Weight均可配置，上电时均默认为F，正常使用前，初始化程序需要对系统Weight进行初始配置。

Scheduler将多个主设备的请求通过WRR仲裁后汇聚为一路请求，经过SMMU发送到RingBus 。

WRR算法说明如下：

WRR： 权重不为零的端口比权重为0的端口优先级高，权重不为0的端口之间采用RR仲裁，获得仲裁的端口对应权重值减1，当权重为0的端口获得仲裁，所有端口的权重值重新刷新为初始值。

22.5.7 Master 端口 ID 设置

Scheduler模块将输入的多路数据汇聚成1路输出，将ID位宽增加3bit，标识源路径信息。

版权所有 © 海思技术有限公司

表 22-2 AXI Master ID 数值

PORT编号	0	1	2	3
ID编号	{0b000,SID}	{0b001,SID}	{0b010,SID}	{0b011,SID}
PORT编号	4	5	6	7
ID编号	{0b100,SID}	{0b101,SID}	{0b110,SID}	{0b111,SID}

22.5.8 DJTAG 接口

DJTAG接口是独立Debug Bus接口，该接口通过JTAG接口外接JTAG Studio设备，可以不受CPU、Fabric的限制，独立对Scheduler所有寄存器进行访问。

22.5.8.1 编程接口

无。

22.5.8.2 应用配置

无。

22.6 初始化

Scheduler不需要特别的初始化流程。

22.7 注意事项

无。

22.8 DFX

每个Scheduler为AR/AW通道分别提供4个32 bits的统计寄存器，共计8组统计寄存器，用于记录4个统计事件。

统计值以首个传输时刻点开始，每次事件发生时+1，当计数满时则保持计数值，读取统计寄存器值后，相应统计寄存器清零。

Scheduler可统计的事件共4个，如表22-3所示。

表 22-3 Scheduler 统计事件功能列表

Module	Num	Event	Description
Scheduler	0	event_trans_cnt	Scheduler上发生的总传输个数统计。

版权所有 © 海思技术有限公司

Module	Num	Event	Description
	1	event_sch_fail_cnt	Scheduler上调度失败次数统计，此处认为当有多端口竞争总线优先权时，一定有端口无法获得仲裁即认为这些端口调度失败。
	2	event_bus_block_cnt	Scheduler上总线拥塞导致的拥塞次数统计。
	3	event_split_trans_cnt	Scheduler上发生split传输个数统计。

22.9 性能指标

22.9.1 延迟（Latency）

Scheduler模块个通道延迟如表22-4所示。

表 22-4 Scheduler 模块各通道传输延迟

通道	延迟周期数（无SPLIT模块）	延迟周期数 有SPLIT模块	备注
AW	2	3	所有延迟统计，未考虑接口为异步的情况，如果端口配置为异步模式，延时会增大。
W	3	4	
B	2	3	
AR	2	3	
R	3	4	

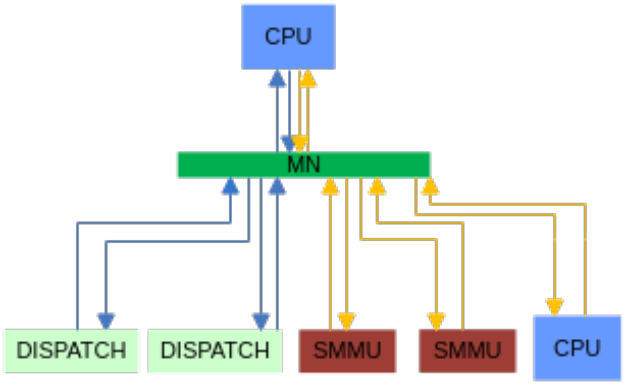
版权所有 © 海思技术有限公司

23 MN

23.1 概述

MN（Miscellaneous Node）是CHI协议中定义的混合节点，挂接在RING环上，主要负责BARRIER和DVM请求的广播。具体流程为：接收来自RN（Request Node）的EOBBarrier、ECBarrier和DVM请求，然后分发到设定的目标HNI节点或能处理DVM操作的RN；待HNI节点和能处理DVM操作的RN完成操作后，负责接收完成响应并向请求RN节点返回结束信息。

图 23-1 MN 在系统中的位置与交互



系统数据流：

1. Barrier数据流：由CPU发出barrier操作，MN根据配置广播到各个dispatch，然后收集所有的dispatch响应后，返回响应给CPU；（如蓝色指针）。
2. DVM数据流：由CPU发出dvm操作，MN根据配置广播到各个SMMU和其他的CPU（发出请求CPU除外），然后收集所有响应后，返回响应给CPU；（如黄色指针）。

版权所有 © 海思技术有限公司

须知

1. MN无法识别系统的节点分布，广播节点均有配置决定。
2. 若配置节点错误，MN广播给错误节点且不回响应，该请求挂死，无法完成。

23.2 特性

MN具有以下特性：

- 每个DIE上面有一个MN，MN运行频率为RING 的1/2。
  - 处理系统中的barrier，dvm操作的广播和响应的收集。
- MN为一个可扩展模块，在多个DIE或者多个SOCKET中可集成多个MN，形成一个广播网络。
  - 每个DIE的MN负责自己所在MN节点的广播，多个DIE之间的广播通过接力完成。
  - 支持广播配置自动屏蔽请求节点，防止广播风暴。
- MN为系统的混合节点，用于处理Barrier的广播。
  - 支持ECBarrier，EOBarrier独立配置处理，广播节点寄存器可配置。
  - 支持多SOCKET系统，ECBarrier，EOBarrier 由请求RN所在的MN直接广播给PA，不进行接力，也可配置，不广播给PA。
- MN为系统的混合节点，用于处理DVM的广播。
  - 支持DVM操作的独立配置处理，广播节点寄存器可配。
  - 支持DVM操作由LLC代替广播，由配置dvm_snp_mask寄存器，表示被mask，并且配置需要广播的目标节点由LLC来广播。
  - 支持DVM操作不对发起者进行selfsnoop，如果由LLC代替广播，则一定不能关闭selfsnoop。
  - 支持多SOCKET系统，DVM由请求RN所在的MN直接广播给主MN（主MN为DVM在系统中负责广播和接收片间请求的MN，通常为离PA最近的MN节点，寄存器可配置），再由主MN广播给PA，PA把请求传给其他SOCKET的主MN，再由其他SOCKET的主MN进行该SOCKET内的广播。
  - 普通dvm操作（非dvmsync）的outstanding可配，其中请求的outstanding:Hi309a MN的配置范围为（1~10）；其中SNP的oustanding的配置范围为（1~8），根据taishan的支持力度,最大配置为5，dvmsync操作的snoop的outstanding为1，普通dvm和dvmsync不相互依赖。
  - MN的Buffer个数可以接收所有的dvmsync操作，保证dvmnonsync操作的完成不依赖于任何其他操作（即大于cpu core的数量）。
- MN支持socket内请求retry机制。
  - 支持对socket内请求进行retry，并在有空处理的时候返回pgrant。
- MN支持配置寄存器完成动态关闭cluster中MN的相关处理。
  - 支持在运行过程中，按照流程可以动态关闭cluster而不影响MN的正常处理。
- MN支持对于DVM的snp广播返回错误响应可屏蔽。
  - 支持收到错误的dvm请求（即dvm type为reserved），不广播，上报中断。
  - 支持寄存器可配置，对于部分RN-D返回的error的snpresp屏蔽掉（避免广播给SMMU不能处理dvm操作后返回resperr导致CPU异常）。

版权所有 © 海思技术有限公司

- 每个MN提供4个48bits Counter，可对MN收到的操作数量等性能信息进行统计。
- 支持通过DJTAG BUS访问MN的寄存器。
- 支持两种中断上报：
  - Debug中断，即收到错误响应和收到有resperr的dvm侦听响应。
  - 统计计数溢出（溢出后翻转）中断。
- 支持MN Buffer 状态查询和残留请求信息查询。
  - 支持查询Buffer寄存器可配置，即可以通过寄存器连续配置看到所有Buffer的状态。
  - 支持查询的信息包括请求是否完成，停留在哪个阶段，是什么请求，由于什么原因没有完成。

23.3 遵从协议

CHI_V4.0协议。

CHI-Enhancement_V4.0 协议。

23.4 功能描述

23.4.1 广播请求的接力

1. 单SOCKET单DIE系统：

   请求RN发请求到MN，由MN直接广播给本DIE配置的所有节点。

2. 单SOCKET多DIE系统：

   请求RN发请求到请求RN所在DIE的MN（本地MN），本地MN广播所在DIE的所有配置节点，同时复制请求发往其他DIE的MN，再由其他DIE的MN广播给自己所在DIE的所有节点，各个MN收齐响应后按请求广播的反向方向返回，最终汇集一个响应返回给请求RN。

3. 多SOCKET多DIE系统的Barrier请求：

   请求RN发请求到请求RN所在DIE的MN（本地MN），本地MN广播所在DIE的所有配置节点，同时复制请求发往其他DIE的MN和所有PA节点，再由其他DIE的MN广播给自己所在的DIE的所有节点。各个MN收起响应后按请求广播的反向方向返回，同时PA也会返回响应，最终汇集一个响应返回给请求RN。

4. 多SOCKET多DIE系统的DVM请求：

   请求RN发请求到请求RN所在DIE的MN（本地MN），本地MN广播所在DIE的所有配置节点，同时复制请求发往其他DIE的MN和所有PA节点，再由其他DIE的MN广播给自己所在的DIE的所有节点。PA节点将DVM请求发给对片的主DIE的MN（主DIE的MN由寄存器配置决定），对片主MN广播自己所在DIE的所有节点，同时负责DVM请求发给其他DIE的MN，再由其他DIE的MN广播给自己所在的DIE的所有节点。最终，各个MN收起响应后按请求广播的反向方向返回，最终汇集一个响应返回给请求RN。

23.4.2 MN 节点配置说明

MN 无论是请求广播（barrier的广播和dvm跨die，跨socket广播）还是snp广播（dvm的die内广播）的配置，均有一个寄存器，总共32bits，对应的就是系统中的{ portid}

版权所有 © 海思技术有限公司

（注：portid为一个totem/nimbus/infinite内部各个节点的id号）。比如说需要广播的节点为10和13，把它译码成32bit的one hot信号，即第16bit为1和第19bit为1，则配置=32’b0000_0000_0000_1001_0000_0000_0000_0000;即寄存器配置为32’h0009_0000。

由于一个L3 partitioned内部有4个core共用一个ID_PORT，所以MN的dvm广播snp需要做特殊处理，即增加了MN_DVMSNP_CFG_0，MN_DVMSNP_PART0和MN_DVMSNP_PART1的配置，分别对应8个cluster广播core0,core1，core2，core3；举例说明：如果需要广播本DIE的所有core，首先需要配置MN_DVMSNP_CFG_0，使DVM能发送至8个L3 partitioned；其次需要配置MN_DVMSNP_PART0和MN_DVMSNP_PART1的cluster*_coreid为0xF。

对于跨DIE的配置，则需要配置MN_MNID来告诉模块系统中的各个DIE的MN的ID号，同时配置MN_DIEVLD，来决定哪些DIE需要广播；举例说明：如果是1Totem +1Nimbus的封装，则需要将Totem和Nimbus的MN_MNID都配置为7D5D3D1D（其中3D1D必须配置，7D5D在这种结构下不需要），然后将MN_DIEVLD配置为0011，则两个DIE将会进行跨DIE接力广播。

对于跨CHIP的配置，则需要配置MN_PAID来告诉模块系统中的其他CHIP的PA的ID号，同时配置MN_CHIPVLD，来决定哪些CHIP需要广播；举例说明：如果是4P的封装，则需要将4P的MN_PAID都配置为0A0908（其中配置的是其他CHIP的ID，所以只需要3个，不包含本CHIP），然后将MN_CHIPVLD配置为111，则4个CHIP将会进行跨CHIP接力广播。

须知

由于MN需要对于MN和PA节点有特殊广播，所以需要识别系统中的MN和PA节点，这个由两个寄存器配置决定（MN_MNID和MN_PAID）。

MN_MNID为配置系统MN节点识别寄存器，可配置四个节点ID，分别为MN0、MN1、MN2、MN3的ID。其中主MN ID在寄存器MN_MAIN_MNID配置。

MN_PAID为配置系统PA节点识别寄存器，可配置三个节点ID，分别为PA0、PA1、PA2的ID。

23.4.2.1 Barrier 操作处理

MN 负责对于EC barrier和EO barrier操作根据寄存器进行广播处理，并接收齐所有响应后返回RN 响应来结束操作。

对于多SOCKET系统，ECBarrier，EOBarrier 由请求RN所在的MN直接广播给PA，不进行接力。

所以EOBarrier和ECBarrier的每个die的配置均只需配：

1. 本die需要广播的HNI节点。
2. 本Socket内其他die的MN节点。
3. 各个PA的节点。

须知

片间barrier请求发给PA后，PA负责排空之前的操作后返回响应，不会广播到其他片。

版权所有 © 海思技术有限公司

23.4.2.2 Dvm 操作处理

MN负责对于所有类型的DVM操作根据寄存器进行广播处理，支持对于需要广播的RN-D进行snoop操作广播并收集响应，支持对于需要广播的MN节点和PA节点进行req广播并进行接力。

对于多Socket系统，dvm操作需要进行多次接力，所以需要识别主MN和从MN之分。

该区分由寄存器配置中MN_MAIN_MNID的最低7bit决定，这7bit所配置的MN节点ID号即为主MN的ID号。

所以Dvmsnp的配置需要配置本die需要广播的RN-D和CPU的节点。

Dvmreq的配置需要配置：

1. 本Socket的其他die的节点。
2. 各个PA的节点。

23.5 编程接口

23.5.1 应用配置

23.5.1.1 广播请求配置

MN广播请求步骤如下：

步骤1 配置MN_MNID中的MN节点ID号，配置MN_PAID中的PA的节点ID号（两个配置无先后要求，该配置默认值大部分场景可用，可用先检视，再决定是否需要配置）。

步骤2 配置MN_EOBARRIER_CFG_0，MN_ECBARRIER_CFG_0，MN_DVMSNP_CFG_0，MN_DVMREQ_CFG_0，MN_DVMSNP_PART0，MN_DVMSNP_PART1来确定需要广播的节点（配置内容方法看上面的配置说明章节）。

步骤3 正常发送barrier，dvm即可。

----结束

须知

MN_MAIN_MNID,MN_MNID,MN_PAID的默认值大部分场景可用，可用先检视后决定是否配置。

23.5.1.2 动态关闭 cluster，MN 配置流程

版权所有 © 海思技术有限公司

须知

在MN是和L3T做接力广播的情况下，动态关cluster会引起配置的不同步，导致dvm请求的错误，因此提供以下两种选择：

1. 配置MN和L3T做接力广播，则不允许动态关cluster。
2. 不配置MN和L3T做接力广播，则允许动态关cluster。

23.5.1.3 动态关闭 core

动态关闭core的具体配置步骤如下（以关闭cluster0 core0为例）：

步骤1 修改MN_DVMSNP_PART0（0x0040）的值，从原始的0x0F变为0x0E，回读该配置，确认配置成功。

须知

在配置MN_DVMSNP_PART0域段cluster*_maskid和cluster*_coreid时，cluster*_maskid至少有1bit为0b0（如果全为0b1则代表没有SNP可以发出），且cluster*_coreid对应的bit要为0b1。例子：当配置域段cluste0_maskid为0b1110时，cluster0_coreid[0]一定要配为0b1。

步骤2 配置MN_DYNAMIC_CTRL中dynamic_en位为1，反复读dynamic_en的值，待其重新变为0，则说明动态关闭cluster中MN部分的处理完成。

----结束

23.5.1.4 动态关闭 cluster

动态关闭cluster的具体配置步骤如下（以关闭cluster0 为例，相当于关闭4个core）：

步骤1 修改MN_DVMSNP_PART0（0~3）的值，从原来的0x0F变为0x00，回读该配置，确认配置成功。

步骤2 配置MN_DYNAMIC_CTRL中dynamic_en位为1，反复读dynamic_en的值，待其重新变为0，则说明动态关闭cluster中MN部分的处理完成。

----结束

须知

该流程需合入系统关闭cluster的流程。

23.6 初始化

不涉及。

版权所有 © 海思技术有限公司

23.7 注意事项

无。

23.8 芯片调试 DFX 功能特性描述

23.8.1 MN 的统计信息

MN提供Monitor统计功能，以统计MN相关性能数据。

每个MN提供4个48bit位宽的统计寄存器，4个统计寄存器均可编程为任一统计事件。计数器为加法计数器，在系统写入计数值（默认为全零），并使能Monitor后开始计数，当计数到全F时将重新从全零开始计数，并产生一个中断通知CPU。

每个MN Monitor内部统计的事件共5个。

表 23-1 MN 内部统计事件功能列表

Num	MN Event	Description	备注
0	event_eobar	eobarrier请求	-
1	event_ecbar	ecbarrier请求。	-
2	event_dvmop	dvmop请求	-
3	event_dvmsync	dvmsync请求	-
4	event_retry	Retry请求	-

23.8.2 中断上报

MN支持两种中断的上报：

1. 统计事件溢出中断。
2. 异常中断。

第一种中断在计数器到全f的时候即会上报一个中断,可以用于热点函数的捕捉。

第二种中断显示MN处于不正常的状态或收到不正常的请求，可用于问题定位。

须知

MN的CE错误属于事件统计溢出中断，已经划分为PERF中断，上报的是冗余，在RAS_CTRL中保持屏蔽不开启即可；

MN的UE错误由于MN没有标准的RAS信息记录，所以中断复位信号没有用RAS中断；因此在复位后会清除。所以复位后看复位信息需要看RAS_CTRL中的MN的UE中断信息，由于中断源只有一个所以可以代表该错误。

版权所有 © 海思技术有限公司

23.8.3 消息上报

23.8.4 Debug

MN提供Monitor状态功能，以指示MN的工作情况；该状态可以配置Buffer号，进行查看每一个Buffer的完成情况。

表 23-2 MN 状态寄存器列表（注：高有效）

Num	MN Event	Description
0	Valid	是否有请求没有完成。
1	Txdbid	是否没有返回dbid。
2	Getdata	是否没有收到数据。
3	Req_inner	非片间请求是否完成。
4	Req_outer	片间请求是否完成。
5	Snp	侦听请求是否发出。
6	Txcomp	完成响应是否发出。
7	Dynamic_ctrl	动态关cluster是否完成。
8	Retryack	是否有retryack没有完成。
9	Rx_pgnt	是否收到pgrant。
18:10	Srcid	该Buffer请求的srcid。
24:19	Opcode	该Buffer请求的Opcode。
27:25	Lpid	该Buffer请求的lpid。

23.9 故障处理指南

23.9.1 软硬件接口故障

- 命令格式硬件不支持。
- 命令参数超界。
- 响应超时未响应。

23.9.2 硬件内部数据完整性错误

- 控制信息的Parity、ECC1bit/2bit错误。
- 数据信息的Parity、ECC1bit/2bit错误。
- 静态配置表项的Parity、ECC1bit/2bit错误。

版权所有 © 海思技术有限公司

- 【以上为内部设计feature，输出文档时去掉】

23.10 典型问题定位流程

MN是否挂死及挂死原因定位流程如下：

步骤1 确认查看MN_STATUS状态寄存器（0x1030），看是否MN有残留命令。

步骤2 若有命令未完成，则配置MN_DFX_CTRL寄存器（0x0408），配置Buffer ID为00000。

步骤3 查看DFX状态信息（0x1040~0x1060），若0x1040中的第1bit为1，则记录这些寄存器的信息。

步骤4 重复**步骤2**、**步骤3**，遍历Buffer ID从0x 00000到0x3FFFF。

步骤5 将获取的信息根据nManger每1bit的含义，即可知道MN当前的挂死状态。

----结束

波形定位流程：

步骤1 查看req_inner_buff_r ,req_outer_buff_r,tgtid_num_req_vld。做一个信号 =~（req_inner_buff_r | req_outer_buff_r）& tgtid_num_req_vld。你会惊奇的发现至少有1bit为1。第几bit为1说明Buffer几的req发出去了但是没有收到响应。在req_inner_buff_r | req_outer_buff_r）拉低的附近去找txreq，你会惊奇的发现这个请求发出去的N个，但是没有收到N个响应。

步骤2 查看snp_buff_r , tgtid_num_snp_vld。做一个信号 =~snp_buff_r & tgtid_num_snp_vld。你会惊奇的发现只有1bit为1。第几bit为1说明Buffer几的snp发出去了但是没有收到响应。在snp_buff_r拉低的附近去找txsnp，你会惊奇的发现这个请求发出去的N个，但是没有收到N个响应。

----结束

23.11 性能指标

23.11.1 延迟（Latency）

MN广播延迟（Latency）如**表23-3**所示。

表 23-3 MN 广播延迟（Latency）

行为	延迟（Latency） Value
收到Barrier请求到广播发出	4拍
收到Dvm请求到广播发出	4拍
收齐请求响应到返回comp响应	3拍

版权所有 © 海思技术有限公司

24 HHA 模块

24.1 概述

Hydra Home Agent（HHA）位于DDR与Ring Bus之间，通过对DDR读写命令进行重排序，为系统提供高带宽低延迟的DDR读写访问。

此外，在HHA中还包含了MN（Miscellaneous Node）。MN（Miscellaneous Node）是Ring 环中的混合节点，能接收来自RN（Request Node）的EOBarrier、ECBarrier和DVM请求，然后分发到设定的目标保序HNI节点或有需要dvm操作的RN；待HNI节点和需要DVM操作的RN完成操作后，负责接受完成响应并向请求RN节点返回结束信息。

24.2 特性

HHA具有以下特性：

- HHA运行频率为Ring Bus频率的半频，HHA和DDRC采用异步接口，和Ring Bus为同步接口，符合Skyros协议。
- 在Hydra域承担PoS（Point of Serialization）角色。
  - 在CHI协议中充当SN-F的角色。
  - 支持对DDR空间的直接访问。
  - 支持Non-Cacheable操作Exclusive传输，为系统提供Non-Cacheable操作的Exclusive Monitor。
  - 支持CPU单操作原子锁。
- 支持CPU读操作在L3 cache miss情况下直接转发，将数据直接返回至CPU。
- 提供64 Entry Request Queue存放正在处理的请求, 提供64 Entry Read/Write Data Queue存放收到但还没有及时处理数据, 提供128 Command Buffer，存放不能及时处理的请求。
- HHA支持DDR空间TrustZone扩展，最大可支持8段安全地址空间，非安全操作无法访问安全地址空间。
- HHA与DDR通过DMC接口对接，数据位宽为256bit，地址位宽为33bit。
- 支持Mbist功能。

版权所有 © 海思技术有限公司

  - HHA所有Memory支持在无访问情况下通过软件启动Mbist测试。
- 每个HHA提供12个48bit Counter，可对Home Agent操作数量、读写命中数量、DDR读写操作数量等信息进行统计。
  - 支持HHA接收命令数量统计，包括HHA接收命令总数、HHA Retry命令总数、Copyback类型命令数量。
  - 支持DDR端口读操作数量、写操作数量统计。
- 支持HHA内部队列状态查询，包括pend Queue、lookup Queue、snoop Queue有效命令统计和有效命令状态查询。
- 支持HHA状态查询，包括初始化结束和有无命令状态。
- 片内CPU可通过总线访问HHA的寄存器接口、片外JTAG接口可通过DJTAG BUS访问HHA的寄存器接口。
- HHA内SRAM支持SECDED ECC，ECC错误信息可统计或上报，只支持上报第一次ECC出错地址。
  - 对于单bit ECC出错，支持纠正出错数据，并统计单bit ECC出错次数，其他RAM支持将出错数据纠正使用。
  - 对于两bits以上ECC出错，支持上报中断，总线返回错误数据，其他SRAM记录出错地址。
  - 可以通过配置寄存器的方式向各RAM注入单bit ECC错误。
- 支持中断事务上报，包括多bit ECC错误、非法地址访问等。
  - 支持多bit ECC错误上报中断，同时记录SRAM出错地址。
  - 支持非安全属性访问安全属性DDR空间上报中断。
  - 支持接收非法命令上报中断。

MN具有以下特性：

- 一个L2总线上面有1个MN，每个MN有32个Buffer，根据所处的Totem和请求的来源分配Buffer ID，MN运行频率与Ring Bus 相同。
  - 每Cycle最大可并行处理1个请求。
  - MN用于处理所有的default slave操作和系统中的barrier，dvm操作。
- MN为系统的混合节点，用于处理barrier和dvm的广播。
  - 支持ECBarrier，EOBarrier统一处理，outstanding为可使用Buffer总数量。
  - 非dvmsync操作和dvmsync操作的snoop的outstanding分别为1，且不相互依赖，MN的Buffer个数可以接收所有的dvmsync操作，保证dvmsync操作的完成不依赖于任何其他操作。
  - 支持可配置的广播barrier的req，dvm的snoop和req。
- MN为系统的default slave的统一处理模块。
  - 支持所有正常的skyros协议的Opcode的处理，按协议正确回响应保证不挂死。
  - default slave除了Buffer阻塞不依赖于任何其他操作，保证一定能完成。
  - 支持可配置的返回resperr。
  - 支持snoop resperr 的透传处理。
- MN支持socket内请求retry机制。
  - 支持对socket内请求进行retry，并在有空处理的时候返回pgrant。
- MN支持配置寄存器完成动态关闭cluster中MN的处理。

版权所有 © 海思技术有限公司

  - 支持在运行过程中，按照流程可以动态关闭cluster而不影响MN的处理。
- 每个MN提供4个32bits Counter，可对MN收到的操作数量等性能信息进行统计。
- 片内CPU可通过DJTAG BUS访问MN的寄存器接口、片外JTAG接口可通过DJTAG BUS访问MN的寄存器接口。
- 支持统计溢出和default slave 上报中断。
  - 对于default slave，支持上报中断并返回总线Error。
- 支持MN Buffer 状态查询和残留请求信息查询。
  - 支持MN内部Buffer状态，包括是否有操作没有完成，且没有完成操作状态的记录。
  - 支持MN残留请求信息查询，MN会记录一个没有回响应请求的信息，信息包括srcid，txnid，Opcode，未回响应的port id。
- 支持第一个default slave信息和最后一个default slave信息记录，信息包括srcid，txnid，Opcode，addr。

24.2.1 关键术语

表 24-1 Hydra HA 关键术语描述

术语	描述
Hydra	一致性协议，分为协议层（protocol layer）、网络层（network layer）、链路层（link layer）和物理层（physical layer），可用于多个SOCs、HAC、IO bridge等的互联。
HA	Home Agent，一致性home节点，接收并响应来自CA的request，产生snoop并收集Snoop response。
CA	Cache Agent，需要进行一致性维护的Cache节点，产生一致性的协议层Transaction，接收并响应来自HA的Snoop，接收来自HA的Response。
MN	Miscellaneous Node 混合节点。
RN	Request Node 请求节点。
HNI	IO Home Node。
LLC	Last Level Cache，在Totem中为L3 Cache。
ACC	Accelerator，加速器device。

24.2.2 遵循协议

ARM CHI 协议，Hydra协议。

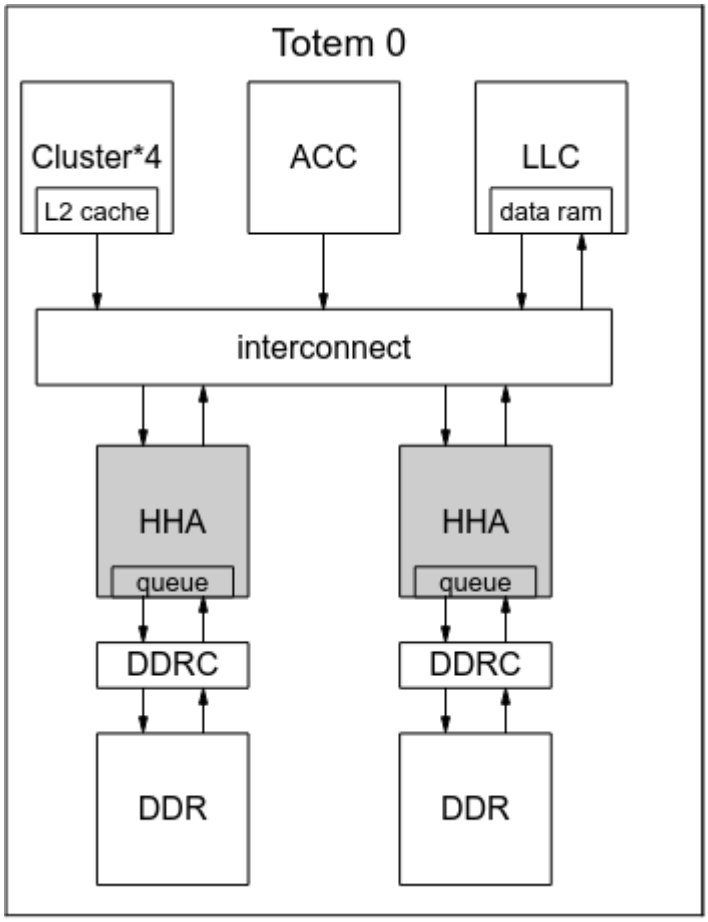
24.2.3 功能描述

版权所有 © 海思技术有限公司

24.2.3.1 应用框图

HHA的应用如图24-1所示。HHA位于Ring Bus与DDRC之间，为系统提供DDR读写访问接口。

图 24-1 HHA 应用图



各CPU Cluster、ACC在读写DDR Non-cacheable空间时，会通过Interconnect的Ring Bus将读写请求发到HHA，HHA发起对DDR空间的读写，最后由HHA将Response通过Interconnect返回给发起请求的CPU Cluster、ACC。

各CPU Cluster、ACC在读写DDR Cacheable空间时，先会通过Interconnect的Ring Bus将读写请求发到本片LLC查询，若查询命中，直接由LLC将Response返回给发起请求的CPU Cluster、ACC；若查询未命中，LLC将请求转发给对应DDR地址空间的HHA， HHA发起对DDR的读写，并将Response返回给LLC。

HHA位于Ring bus和DDR之间，任何DDR读写访问均需要通过HHA，HHA为DDR读写访问提供了请求重排序功能，根据DDR效率和QOS优先级，对DDR读写请求进行调度，获得高带宽低延迟的内存读写。

24.2.3.2 相关模块

在系统中，HHA模块会和下面多个模块进行交互，分别描述如下：

CPU Cluster/ ACC /LLC：CPU Cluster/ACC/LLC作为master，通过Interconnect对HHA发起访问。

版权所有 © 海思技术有限公司

DDRC：DDR控制器，通过DDRC读写DDR；

Ring Bus/Interconect：Hydra HA与LLC均挂在Ring Bus上，Hydra HA与LLC通过Ring bus进行交互。

24.2.3.3 工作频率

Hydra Home Agent运行频率与Ring Bus相同，Hydra Home Agent和DDRC采用异步接口，和Ring Bus为同步接口。

24.2.3.4 保序模型

对于cacheable类型操作，Hydra HA与CA之间通过Completion Response保证多命令之间的顺序性。

对non-cacheable类型操作，Hydra HA需要及时给出RdReceipt或DBIDResp响应，协同请求者完成多命令间的ordering。HHA需要对接收操作进行相关性检查，避免多个同地址请求对相同地址空间的访问冲突，并保证多个源访问数据的一致性。

24.2.3.5 TrustZone

Hydra HA支持TrustZone扩展，通过HHA寄存器最大可配置八段DDR安全空间，安全属性操作可访问DDR中安全和非安全空间；非安全属性操作可访问DDR中非安全空间，但不能访问安全空间，读操作返回0，写操作无法写入DDR。

24.2.3.6 DDR 接口

HHA与DDR为DMC接口，数据位宽为256bit，地址位宽为33bit，不支持exclusive、lock操作，HHA发出操作的length最大为3（即burst4）、size最大为256bit，仅支持INCR和WRAP操作，不支持FIXED操作。HHA支持INCR和WRAP类型读操作，支持读操作out-of-order，支持读数据间插。HHA仅支持INCR类型写操作，写数据不会间插。

HHA支持64的命令处理outstanding，同时最多可看到64个读写DDR命令，可对DMC发出的命令outstanding为64。 HHA发往DMC的命令outstanding支持可配。

24.2.3.7 CPU 数据直接转发

Totem采用cache分级架构，当CPU发起cacheable操作请求数据时，首先将操作发至LLC，若LLC命中，则LLC直接将数据返回给CPU，当LLC未命中时，由LLC发起操作向HHA请求数据，HHA读取DDR数据返回至LLC，再由LLC将数据返回给CPU。

CPU访问LLC miss后，HHA支持数据快速转发。当CPU发起cacheable操作请求数据时，首先仍然将操作发往LLC，若LLC命中，则直接由LLC将数据返回至CPU，若LLC未命中，则由LLC发起操作向HHA请求数据，HHA读取DDR数据返回给CPU，而仅返回response到LLC，减少CPU获取数据的latency。

24.2.3.8 Exclusive Monitor

HHA支持non-cacheable属性Exclusive操作，每个HHA为Exclusive操作提供一个Exclusive System monitor。HHA支持以下两种Exclusive传输。

24.2.3.8.1 Exclusive 传输 1

对于送入本模块的Exclusive传输，HHA在流水线上内置一个Exclusive Monitor，以支持CPU原子锁操作。Exclusive Monitor可同时监控32个cacheline的Exclusive访问。对

版权所有 © 海思技术有限公司

每一个ID（srcid+lpid），只记录一个Exclusive操作的地址，且记录的是最新操作地址，当同ID、同地址（单个HHA最大支持8GB，关注地址位宽为33bit，即addr[32:0]）、同NS的Exclusive写操作与该记录匹配时，返回Exokey。Exclusive Monitor的监控粒度为一条Cache Line大小，同cacheline的非exclusive写操作或者同cacheline的Exclusive成功写操作会清除该cacheline的所有记录，exclusive失败的写操作不清除记录。Exclusive读操作仅当地址与size非对齐时会失败，返回Okey。

当监控的32个ID的Exclusive访问全部没有结束，又接受到第33个Exclusive访问读操作，会返回EXOKAY响应；当接收到对应的写操作时，返回OKAY相应，指示操作失败。

Exclusive读操作到来后，仅当满足以下条件时，Exclusive写操作返回Exokey：

1. Exclusive读操作地址与Size对齐。
2. Exclusive读操作进入Monitor时， Exclusive monitor未满。
3. Exclusive写操作到来前，没有同cacheline地址的非Exclusive写操作，且没有同cacheline地址的exclusive成功操作。
4. Exclusive写操作到来前，没有同ID不同地址的Exclusive读操作访问。

Exclusive写操作与之前的Exclusive读操作同ID（srcid+lpid）、同NS、同地址（addr[32:0]）。

须知

非安全操作对安全空间抢锁可能成功，并且可能影响安全操作抢锁的效率，但非安全操作无法访问到安全空间内容。

24.2.3.8.2 Exclusive 传输 2

HHA还可支持另一种特殊Exclusive操作处理，可以在多个CPU之间实现快速的Exclusive操作。各个CPU使用Exclusive读操作抢锁，使用Exclusive写操作释放锁。对于收到的某个master的Exclusive读操作，当目的地址中的数据为0时，将数据1写入该地址，并向该master返回数据1和EXOK响应，该master认为抢锁成功，可以对某个共享变量执行操作，操作完成后使用Exclusive写操作写入数据0，将锁释放；当目的地址中的数据为1时，表示锁被另外的master占用，此时不改写数据，并向该master返回数据0和EXOKEY响应，表示抢锁失败，该master不能对共享变量执行操作，必须再次发出Exclusive读操作尝试抢锁，直到抢到为止。对于收到的Exclusive写操作，向Master返回EXOKEY响应，数据1的格式是大端还是小端可通过寄存器进行配置，默认为大端，建议与CPU当前使用的模式配成相同。

该传输方式的size为32bit，通过判断目的地址指向的32bit数据决定是否抢锁成功。且非exclusive写操作可写入目的地址中。

这两种Exclusive传输可通过寄存器配置进行选择。

24.2.3.9 MN 功能描述

24.2.3.10 Barrier 操作处理

MN 负责对于EC barrier和EO barrier操作根据寄存器进行广播处理，并接收齐所有响应后返回RN 响应来结束操作。

版权所有 © 海思技术有限公司

对于请求来自RN的请求，则会广播除源头RN节点外的其他所有配置节点。

对于请求来自MN的请求，则会广播除源头和其他MN节点之外的其他所有配置节点。

24.2.3.11 Dvm 操作处理

MN负责对于所有类型的DVM操作根据寄存器进行广播处理，支持对于snoop操作广播并收集响应，支持对于req进行totem之间进行接力。

Snoop广播：

对于来自RN，MN，PA的请求，均会广播除源头RN节点之外的其他所有配置节点。

Req 接力：

对于请求来自RN的请求，则会广播除源头RN节点外的其他所有配置节点。

对于请求来自MN的请求，则会广播除源头和其他MN节点之外的其他所有配置节点。

须知

由于MN判断是否请求来自于其他MN，是根据寄存器配置中special ID进行判断的，所以该配置一定要准确，如果有MN的节点ID不在special ID配置内部，则MN会将这个请求看成是来自RN的请求，可能会造成广播风暴。

24.2.3.12 Default Slave 操作处理

对于系统中地址译码错误的请求，都会发到MN，由MN统一回响应，并上报中断，返回resperr。

对于SMMU发出的部分操作会根据传的信息中的RSVDC_FRE域段，来决定是否返回resperr。

支持任意size的读写操作，并正确回数据。

Default slave在MN0处理，和barrier，dvm操作无任何依赖关系。

24.2.3.13 MN 的 DFX 信息

MN提供Monitor统计功能，以统计MN处理的相关数据。

每个MN提供4个32bits Counter，可对MN收到的操作数量等性能信息进行统计；

MN monitor内部统计事件共8个。

表 24-2 MN 内部统计事件功能列表

Num	MN Event	Description	备注
0	Eobar	Eobar请求	-
1	Ecbar	Ecbar请求	-
2	Dvmop	Dvmop操作	包括dvmsync

版权所有 © 海思技术有限公司

Num	MN Event	Description	备注
3	Dvmsync	Dvmsync操作	-
4	Read	读请求	-
5	Write	写请求	-
6	Copyback	Copyback请求	-
7	Else	其他请求	-

MN提供Monitor状态功能，以记录MN处理的最终状态。

表 24-3 MNBuffer 状态寄存器列表（注：高有效）

Num	MN Event	Description
0	Valid	是否请求没有完成。
1	Txdbid	是否有请求没有返回dbid。
2	Txrdrpt	是否有请求没有返回readreceipt。
3	Rxdbid	是否有请求没有收到dbid。
4	Getdata	是否有请求没有收到数据。
5	Txdat_cnt	是否有请求没有发往数据。
6	Rxdat_cnt	是否有请求没有收齐数据。
7	Req	是否有请求没有发出请求。
8	Snp	是否有请求没有发出snp。
9	Tgtid_num	是否有请求没有收到响应。
10	Tgtid_snp	是否有snoop没有收到响应。

此外，MN还支持记录一个MN内部未回响应请求的信息，信息包括srcid，txnid，Opcode，未回响应的port id；MN还支持第一个default slave信息和最后一个default slave信息记录，信息包括srcid，txnid，Opcode，addr；

24.3 软件指南

24.3.1 初始化

HHA初始化步骤如下：

版权所有 © 海思技术有限公司

步骤1 复位撤离后，配置HHA对应DDR空间基地址HHA_BA（8GB对齐）、安全空间起始地址寄存器HHA_SECn_SA（4kByte对齐，n为0~7，需启用安全空间时配置）和结束地址寄存器HHA_SECn_EA（4kByte对齐，n为0~7，需启用安全空间时配置）。

步骤2 根据当前HHA对应DDR结构及DDRC配置，配置HHA_DDR_MODE寄存器和HHA_SCH_CTRL寄存器设置DDR调度工作方式；HHA_DDR_MODE寄存器与DDRC配置相同即可。

步骤3 配置HHA状态清除寄存器HHA_CLR_STAT，清除HHA状态寄存器、HHA错误状态寄存器和DFX计数器；（上电复位初始化需要执行该步骤）。

步骤4 清除HHA中断；（上电复位初始化需要执行该步骤）。

步骤5 配置HHA控制寄存器HHA_CTRL，设置HHA工作方式，包括RAM ECC使能、DFX统计计数器使能等。

----结束

须知

HHA_SECn_SA、HHA_SECn_EA几个寄存器不支持动态配置；HHA_CTRL寄存器各域段默认不使能，使能相应功能时需在初始化阶段使能各域段，除link_down外，其余域段均不支持动态配置。

HHA中断状态寄存器、HHA状态寄存器、HHA ECC错误状态寄存器、HHA错误命令信息寄存器、HHA DFX统计计数器均为不带复位端寄存器，复位不能清除状态，上电后需要通过软件配置初始化。

每次解复位和开启时钟后，必须读取时钟和复位状态，确保HHA模块已经完成解复位和开时钟，然后才能启动模块功能。

24.3.2 Mbist 测试操作

Mbist操作的具体配置步骤如下：

步骤1 配置寄存器mbist_reset_select，选择是用模块复位还是用TDRE配置复位。

步骤2 如果是TDRE配置复位，则需要写mbist_reset_select[0]撤离复位。

步骤3 配置algsel_scan_in、dft_mem_gt_se等寄存器，选择测试算法和所测试的memory等信息。

步骤4 配置mbist_clk_en，打开Mbist的时钟。

步骤5 配置mbist_en，开始Mbist测试。

步骤6 等读到mbist_done后再读mbist_fail寄存器，看Mbist测试是否通过。

----结束

24.3.3 软复位

正常工作阶段，HHA内可能存在正在处理的命令，不能直接进行Hydra HA单个模块的软复位，需等待HHA无命令时才能软复位，即业务场景下HHA禁止软复位。

HHA允许整个系统软复位时进行同步软复位。

版权所有 © 海思技术有限公司

如果要求模块软复位，需要与模块时钟门控成对配合使用，建议按照如下复位流程：先发起模块软复位--->再关闭模块时钟门控--->再撤离模块软复位--->最后打开模块时钟门控。

24.3.3.1 广播请求配置

MN广播请求步骤如下：

步骤1 看系统是几个TOTEM和几个INFINITE的结构，则把special ID的cpudie0的配置变为该MN的id_port，而其他两个cpudie1，cpudie2配置成其他几个MN的ID（注意MN1对应MN1的，MN0对应MN0的）；以三个TOTEM和一个INFINITE为例，则三个TOTEM的MN1的special ID配置为83F2F1F（说明1F需要接收片间请求），三个TOTEM的MN0的specialID配置为83E2E1E。

步骤2 配置EOBAR_CFG_TARGET，ECBAR_CFG_TARGET，DVMSNP_CFG_TARGET，DVMREQ_CFG_TARGET，来决定需要广播的节点。

步骤3 正常发送barrier，dvm和default slave即可。

----结束

表 24-4 Special ID 配置具体说明

结构	MN1配置	MN0配置
totemA+Infinite	83f2f1f	83e2e1e
totemC+Infinite	83f1f2f	83e1e2e
totemA+totemC+Infinite	83f2f1f	83e2e1e
totemA+totemB +totemC_Infinite	83f2f1f	83e2e1e
Infinite+totemA+totemC +Infinite	83f2f1f	83e2e1e

24.3.3.2 动态关闭 cluster，MN 配置流程

动态关闭 cluster

动态关闭cluster的具体配置步骤如下：

步骤1 修改DVMSNP_CFG_TARGET的值，变为需要重新配置后的新的snp target的值。

步骤2 配置DVMSNP_CFG_TARGET完成后，配置DYNAMIC_CTRL中dynamic_en位为1，反复读dynamic_en的值，待其重新变为0，则说明动态关闭cluster值，MN部分的处理完成。

----结束

24.3.4 性能

版权所有 © 海思技术有限公司

24.3.4.1 吞吐率（Throughput）

Hydra HA的吞吐率跟DDR效率、总线位宽等有关。

理论最大读数据吞吐率为：工作频率×256bit。

理论最大写数据吞吐率为：工作频率×256bit。

24.3.4.2 延迟（Latency）

HHA对于存取访问的延迟（Latency）如下表24-5所示。

表 24-5 HHA 存取延迟（Latency）

行为	延迟（Latency） Value
接收读操作到DDR端口最小latency	时钟周期*4
接收DDR读数据到txdat最小latency	时钟周期*5
接受命令到发出retry最小latency	时钟周期*2
接收命令到发出comp最小latency	时钟周期*4

24.4 DFX

24.4.1 模块维测

24.4.1.1 统计

HHA提供Monitor统计功能，以统计HHA相关性能数据。

每个HHA提供12个32bit位宽的统计寄存器，8个统计寄存器均可编程为任一统计事件，通过配置HHA_EVENT_CNTTYPE_L和HHA_EVENT_CNTTYPE_H寄存器控制每一个统计寄存器的事件类型。计数器为加法计数器，在系统写入计数值（默认为全零），并使能Monitor后计数，当计数到全F时将重新从全零开始计数，并产生一个中断通知CPU。

每个HHA提供以下计数事件，事情类型见表24-6。

表 24-6 HHA 内部统计事件功能列表

Num	HHA Event	Description
0	event_rxreq	HHA接收的非linkflit命令统计事件。
1	event_retry	HHA发送的retry统计事件。

版权所有 © 海思技术有限公司

Num	HHA Event	Description
2	event_pgnt	HHA发送的P-Grant统计事件。
3	event_outer	HHA接收的outer属性操作统计事件。
8	event_dmc_rdcmd	HHA发送DDR读操作统计事件。
9	event_dmc_wrcmd	HHA发送DDR写操作统计事件。
10	event_secc	HHA发生单bit ECC错误统计事件。
12	event_internal	HHA接收的本片操作统计事件。
13	event_cback	HHA接收的copyback操作统计事件。
14	event_sq_rxreq	HHA已经处理或正在处理操作统计事件。
15	event_ sq_rd	HHA已经处理或正在处理读操作统计事件。
16	event_ sq_wr	HHA已经处理或正在处理写操作统计事件。
17	event_rdlen	HHA已经处理或正在处理128B读操作统计事件。
18	event_wrlen	HHA已经处理或正在处理128B写操作统计事件。
19	event_sq_rdddr	HHA处理队列发送DDR读操作统计事件；同0x8。
20	event_sq_wrddr	HHA处理队列发送DDR写操作统计事件；同0x9。
22	event_dup	HHA接收的duplicate操作统计事件。
23	event_dup_succ	HHA接收的duplicate成功操作统计事件。
26	event_rowhit	HHA发送给DDR的rowhit操作统计事件。
27	event_rowhitrd	HHA发送给DDR的rowhit读操作统计事件。
28	event_bankcft	HHA发送给DDR的bankconflict操作统计事件。
29	event_bankcftrd	HHA发送给DDR的bankconflict读操作统计事件。

版权所有 © 海思技术有限公司

Num	HHA Event	Description
30	event_timeout	HHA发送给DDR的timeout操作统计事件。
37	event_cancel	请求地址冲突cancel统计事件。

MN提供Monitor统计功能，以统计MN相关性能数据。

每个MN提供4个32bit位宽的统计寄存器，4个统计寄存器均可编程为任一统计事件。计数器为加法计数器，在系统写入计数值（默认为全零），并使能Monitor后计数，当计数到全F时将重新从全零开始计数，并产生一个中断通知CPU。

每个MN Monitor内部统计的事件共9个。

表 24-7 MN 内部统计事件功能列表

Num	MN Event	Description	备注
0	event_eobar	eobarrier请求。	-
1	event_ecbar	ecbarrier请求。	-
2	event_dvmop	dvmop请求。	-
3	event_dvmsync	dvmsync请求。	-
4	event_rd	读请求。	-
5	event_wr	写请求。	-
6	event_copyback	收到copyback请求。	-
7	event_else	收到除了读写copyback，barrier，dvm的其他请求。	-
8	event_retry	Retry请求。	-

MN提供Monitor状态功能，以指示MN的工作情况。共有1个状态寄存器。

表 24-8 MN 状态寄存器列表（注：高有效）

Num	MN Event	Description	备注
0	Valid	是否有请求没有完成。	-
1	Txdbid	是否没有返回dbid。	-
2	Txrdrpt	是否没有返回rdrpt。	-
3	Rxdbid	是否没有收到dbid。	-

版权所有 © 海思技术有限公司

Num	MN Event	Description	备注
4	Getdata	是否没有收到数据。	-
5	Txdat_cnt	是否还有读数据没有发送。	-
6	Rxdat_cnt	是否还有写数据没有接收。	-
7	Req	是否还有广播req没有发出。	-
8	Snp	是否还有广播snp没有发出。	-
9	Tgtid_req	是否还有请求响应没有收齐。	-
10	Tgtid_snp	是否还有snp响应没有收齐。	-
11	Rx_retryack	是否收到了retryack没有处理。	-
12	Rx_pgrant	是否收到了pgrant没有处理。	-

24.4.1.2 中断上报

HHA支持中断类型如表24-9所示。

表 24-9 HHA 中断向量表

类型	描述
ecc_err_int	不可纠正多bit ECC错误中断状态位。
ns_err_int	非安全操作访问安全区域中断状态位。
rxreq_op_err_int	接收非法request错误中断状态位。
rxrsp_op_err_int	接收非法response错误中断状态位。
rxdat_op_err_int	接收非法data错误中断状态位。
tgtid_err_int	接收错误tgtid请求中断状态位。
cnt(0~11)_overflow_int	DFX计数器溢出中断状态位。

24.4.1.3 消息上报

版权所有 © 海思技术有限公司

24.4.1.4 Debug

本模块提供了以下方式用于内部状态查询，以方便系统调试（Debug）。

- 本模块提供给了Debug Bus接口，该接口可配置和可读取寄存器配置信息及统计信息。
- HHA内部的所有寄存器，均可以通过Debug Bus接口进行读写访问。

24.5 注意事项

操作Hydra HA额外注意事项如下：

1. HHA模块中HHA_SECn_SA、HHA_SECn_EA几个寄存器不支持动态配置。
2. HHA_CTRL寄存器各域段默认不使能，使能相应功能时需在初始化阶段使能各域段，除link_down外，其余域段均不支持动态配置。

版权所有 © 海思技术有限公司

25 eFuse

25.1 eFuse

本段落主要是对eFuse的功能和应用做详细描述。

25.1.1 概述

eFuse用来实现需要永久保存信息的功能。eFuse保存的永久信息用于实现专用的功能和系统安全等。eFuse的主要功能如下：

- SECL bypass eFuse：用于调整、校准、使能/去使能片上的功能。
- 安全eFuse：用于满足产品专用的安全需求。
- DIEID eFuse：为每个芯片提供唯一的识别作用，可以用来追溯芯片的整个生产信息。
- Memory冗余eFuse：用于“纠正”片上Memory的硬失效。这些硬失效在生产过程中检测出来，对应的eFuse也接着被烧写。Hi309a有只有1块eFuse 做Memory Repair。
- Hi309a采用了4块的eFuse模块，efuse0（DIEID eFuse）、efuse1（安全eFuse）、efuse2（SECL bypass eFuse）和ram冗余 eFuse。除RAM冗余eFuse，其他eFuse都采用double-bit设计提高可靠性。

下文描述中的eFuse除了eFuse IP外，还包括用于读写eFuse的控制器。

25.1.2 特性

eFuse具有以下特性：

- 支持eFuse一致的读写控制逻辑。
- 支持eFuse JTAG和CPU读写。
- 支持eFuse安全读写。
- eFuse读写控制器自动进入低功耗模式。
- Double-bit设计提高烧写的可靠性。
- eFuse读/写的时间小于100 μs/macro（50M时钟，128x32）。
- 支持指定域段在线烧写，不需要上电复位立即生效。

版权所有 © 海思技术有限公司

说明

eFuse作为电熔丝逻辑，在供电、读和烧写需要注意：

1. eFuse由两种电源供电，即读电源VDD和烧写电源VQPS。如果不烧写，VQPS接地或者悬空。
2. 烧写后的eFuse读取值为“1”，未烧写的eFuse为“0”。
3. eFuse不允许重复烧写，烧写过程分为三步。
    1. Prefuse Read，烧写前读取eFuse信息，确保欲烧写字段无烧写过。
    2. Fuse，烧写。
    3. Postfuse Read，烧完后读取eFuse，确保烧写成功。
4. 除了指定域段支持在线烧写并软复位生效外，其他字段烧写完成后，需要重新上电生效。

25.2 关键术语

如表25-1所示的术语将在本章节表示特殊的含义。

表 25-1 eFuse 关键术语描述

术语	描述
eFuse	Electrical-FUSE，一次性的电可编程的FUSE。

25.3 应用框图

eFuse的应用如图25-1所示。

版权所有 © 海思技术有限公司

图 25-1 eFuse 应用图

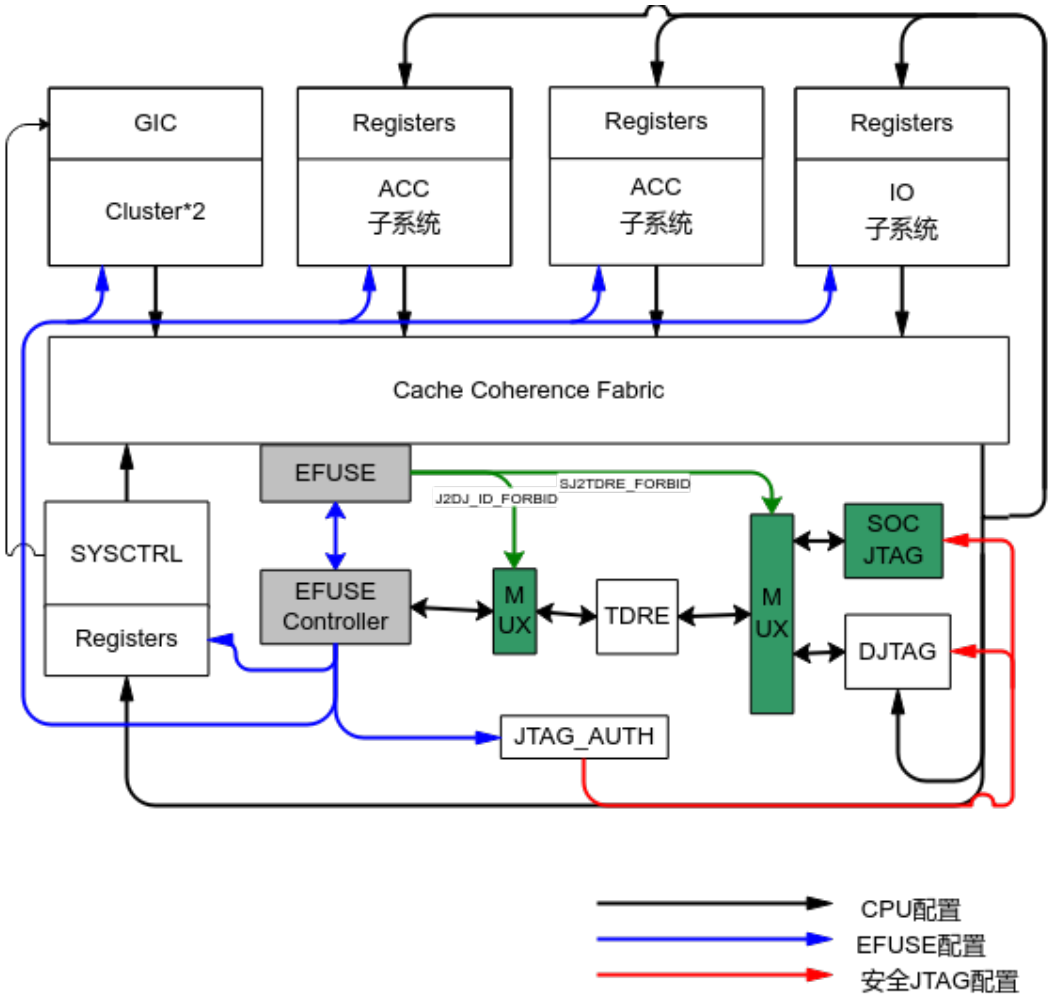


图25-1中J2DJS_ID_FORBID分别控制两块非安全eFuse（efuse0、efuse2）,烧写为非0值后，eFuse信息将不能够通过tdre接口访问。

安全eFuse（efuse1）提供了高优先级的控制信号SJ2TDRE_FORBID_2，烧写为非0后，将无法通过dft_jtag 烧写eFuse，当产品评估不需要dft_jtag烧写返修域段的时候可以烧写这个域段。

eFuse信息在ATE测试、装备阶段烧写完成。芯片初始化阶段读取eFuse，配置芯片的工作状态。

25.4 相关模块

在系统中，eFuse模块会和下面多个模块进行交互，分别描述如下：

- TDRE：TDRE用于控制eFuse Controller，JTAG和CPU通过TDRE完成基本的eFuse读或烧写操作。
- SYSCTRL、PERI SUBSYS等：eFuse存储的信息，如TSensor校准、AVS参数、安全和Memory Repair等分布到各个相关模块。

版权所有 © 海思技术有限公司

- JTAG_AUTH：根据eFuse烧写的安全等级，控制JTAG操作。

25.5 模块框图

eFuse模块框图参考图25-1。eFuse逻辑主要包括eFuse和eFuse Controller。eFuse存储需要永久保存的信息；eFuse Controller完成对eFuse的读操作和烧写操作。

25.6 数据结构

Hi309a集成了3个的eFuse Macro，eFuse 0，eFuse1和eFuse2。eFuse采用集中的方式实现，eFuse内容分布到各个相应的硬件模块中。这些eFuse内容可以通过寄存器访问。eFuse值的烧写除了安全相关的信息是在装备测试后烧写外，其他都是在ATE测试的CP或者FT测试阶段完成，不同字段的访问模式也存在差异。系统初始化阶段会触发eFuse的读操作，当所有eFuse内容读取完后系统复位才会撤离。eFuse读操作完成后自动进入Power Down模式。

说明

装备字段的烧写顺序为：其他字段 -> NS_FORBID -> KEY_CFGED。

25.7 软件指南

25.7.1 初始化

eFuse的初始化流程参考Hi309a初始化流程描述。正常的eFuse初始化流程不需要软件参与。

25.7.1.1 eFuse 烧写

eFuse烧写方式分为三种：

- SoC JTAG（DFT烧写方式）
- CPU DJTAG
- JTAG DJTAG

CPU DJTAG和JTAG DJTAG分别通过CPU和JTAG接口访问Sysctrl的DJTAG寄存器组，发起DJTAG Master操作，控制DJTAG TAP控制器访问BISR的TDRE接口，完成烧写eFuse。

烧写流程：

软件烧写流程都是调用djtag_master来进行烧写。

SYSCTRL中，DJTAG_MASTER 基地址0xFA0A_0000。

烧写举例：

欲烧写eFuse【415:384】，数据0x3F0，对应eFuse域段在sysctrl寄存器EFUSEx_415_384

步骤1 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

版权所有 © 海思技术有限公司

步骤2 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20B1FFFF选中BISR0的TDRE链，并配置写使能。

步骤3 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x800C选中偏移地址为0x800c（即 EFUSE0_415_384）

步骤4 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x3F0，配置数据为0x3F0（即EFUSE0_415_384==0x3F0）

步骤5 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤6 CPU读寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C），如果读到值为0x0，进入**步骤7**，否则继续等待，直到读出值为0x0。

步骤7 完成前面**步骤1**~**步骤6**代表数据已经load到待烧写寄存器内部，继续**步骤5**。

步骤8 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤9 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20B1FFFF 选中BISR0的TDRE链，并配置写使能。

步骤10 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x7，选中偏移地址为0x7（即Bit0:dieid_enable）。

步骤11 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x1，配置数据为0x1（即Bit0:dieid_enable）。

步骤12 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤13 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit8）值为1，进入**步骤14**，否则继续等待。

步骤14 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤15 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20B1FFFF 选中BISR0的TDRE链，并配置写使能。

步骤16 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

步骤17 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x840，配置数据为0x840。

步骤18 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤19 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit8）值为1，进入**步骤20**，否则继续等待。

步骤20 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤21 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20B1FFFF 选中BISR0的TDRE链，并配置写使能。

步骤22 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

版权所有 © 海思技术有限公司

步骤23 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x846，配置数据为0x846（即Bit1：hard_repair，Bit2：dft_mode）。

步骤24 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤25 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit8）值为1，进入**步骤26**，否则继续等待。

以下步骤是软件烧写eFuse1前，需要添加的额外流程：

步骤26 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤27 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20B2FFFF 选中BISR1的TDRE链，并配置写使能。

步骤28 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

步骤29 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x800，配置数据为0x800。

步骤30 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤31 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit8）值为1，进入**步骤32**，否则继续等待。

步骤32 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤33 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20B2FFFF 选中BISR0的TDRE链，并配置写使能。

步骤34 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

步骤35 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x800，配置数据为0x800。

步骤36 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤37 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit8）值为1，进入**步骤38**，否则继续等待。

步骤38 CPU配置SYSCTRL寄存器SC_EFUSE1_RESET_REQ(偏移值D70)，配置值0x1，复位BISR1。

步骤39 CPU读取SYSCTRL 寄存器SC_EFUSE1_RESET_ST(偏移地址5D70)，读取值为0x1，代表BISR1处于复位状态。

步骤40 CPU配置SYSCTRL寄存器SC_EFUSE1_RESET_DREQ(偏移值D74)，配置值0x1，解复位BISR1。

步骤41 CPU读取SYSCTRL 寄存器SC_EFUSE1_RESET_ST(偏移地址5D70)，读取值为0x0，代表BISR1处于解复位状态。

版权所有 © 海思技术有限公司

步骤42 cpu读取SYSCTRL寄存器SC_EFUSE_HARD_REPAIR_DONE(偏移地址0xE098) bit0~2分别指示eFuse0~2的移位结束状态，为1 表示移位结束。等待bit1变为1。BISR1 初始化完成。

步骤43 执行烧写**步骤1**~**步骤25**，在线烧写eFUSE1的域段。

以下步骤是软件调用CER在线更新EFUSE_S前，需要添加的额外流程。

步骤44 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤45 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20b2FFFF 选中BISR1的TDRE链，并配置写使能。

步骤46 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

步骤47 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x800，配置数据为0x800。

步骤48 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤49 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit[8]）值为1，进入**步骤50**，否则继续等待。

步骤50 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤51 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x20b2FFFF 选中BISR1的TDRE链，并配置写使能。

步骤52 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

步骤53 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x800，配置数据为0x800。

步骤54 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤55 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit[8]）值为1，进入**步骤56**，否则继续等待。

步骤56 软件调用CER执行在线烧写eFuse1的任务。

以下步骤是CER烧写完成后，软件比对RIM值是否烧写成功，需要添加的额外流程：

步骤57 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤58 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x204FFFFF选中BISR1的TDRE链，并配置写使能。

步骤59 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

步骤60 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x800，配置数据为0x800。

版权所有 © 海思技术有限公司

步骤61 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤62 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit[8]）值为1，进入**步骤63**，否则继续等待。

步骤63 CPU配置寄存器SC_DJTAG_SEC_ACC_EN（偏移地址0xD800）为0x1，配置操作为可访问安全数据。

步骤64 CPU配置SC_DJTAG_MSTR_CFG（偏移地址0xD818）为0x204EFFFF选中BISR0的TDRE链，并配置写使能。

步骤65 CPU配置SC_DJTAG_MSTR_ADDR（偏移地址为0xD810）为0x0，选中偏移地址为0x0。

步骤66 CPU配置寄存器SC_DJTAG_MSTR_DATA（偏移地址0xD814）为0x800，配置数据为0x800。

步骤67 CPU配置寄存器SC_DJTAG_MSTR_START_EN（偏移地址0xD81C）为0x1，启动DJTAG Master访问。

步骤68 CPU读寄存器SC_DJTAG_OP_ST（偏移地址0xE828），如果读到域段djtag_op_done（bit[8]）值为1，进入**步骤68**，否则继续等待。

步骤69 CPU配置SYSCTRL寄存器SC_EFUSE1_RESET_REQ（偏移值D70），配置值0x1，复位BISR1。

步骤70 CPU读取SYSCTRL 寄存器SC_EFUSE1_RESET_ST（偏移地址5D70），读取值为0x1，代表BISR1处于复位状态。

步骤71 CPU配置SYSCTRL寄存器SC_EFUSE1_RESET_DREQ（偏移值D74），配置值0x1，解复位BISR1。

步骤72 CPU读取SYSCTRL 寄存器SC_EFUSE1_RESET_ST（偏移地址5D70），读取值为0x0，代表BISR1处于解复位状态。

步骤73 CPU读取SYSCTRL寄存器SC_EFUSE_REPAIR_DONE_ST（偏移地址0x E828）bit[0]~bit[2] 分别指示eFuse0~2的移位结束状态，为1表示移位结束。等待bit[1]变为1。BISR1 初始化完成。

步骤74 CPU读取SYSCTRL中eFuse1对应域段判断烧写值是否正确。

----结束

说明

步骤1~**步骤6**是完成eFuse待烧写数据配置，通过重复这个过程，可以完成eFuse中多个域段的配置。**步骤7**~**步骤25**是完成烧写操作下发。如果缺少了**步骤25**~**步骤30**，在线烧写eFuse后，将无法通过软复位，读出eFuse的存储值。

25.7.1.2 在线更新 eFuse 中的 RIM 值

步骤和注意实现参见《Hi309a 通用MPU芯片 用户指南02-加速引擎》12 CER章节。

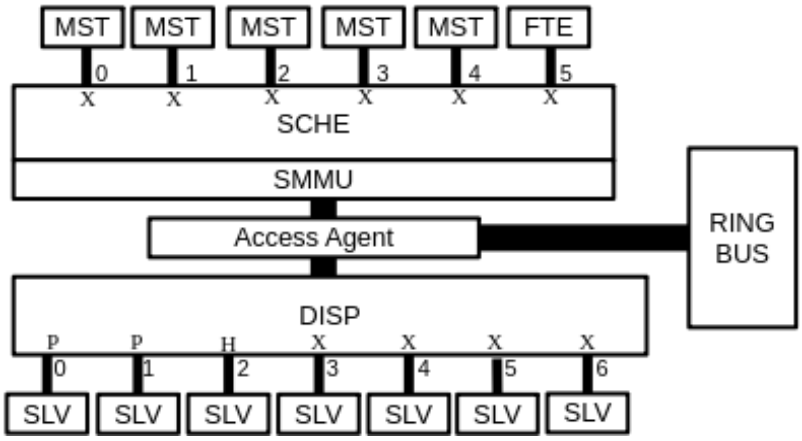
版权所有 © 海思技术有限公司

26 FTE

26.1 概述

FTE为总线测试模块，作为独立的master，可以通过挂接在各SCHE的端口上，对总线的不同数据通路进行测试。

图 26-1 FTE 结构示意图



FTE主要实现的功能：

1. 协助原型验证，对总线进行冲击测试，统计多种延迟数据。
2. 满足筛片要求，在筛片时能产生向量和各种不同激励，覆盖关键路径。

26.2 关键术语

表 26-1 FTE 关键术语描述

术语	描述
FTE	Fabric Test Engine 总线测试模块。

版权所有 © 海思技术有限公司

26.2.1 特性

- FTE支持4种读写命令发送方式：只读模式、只写模式、读写比对模式和读写非对比模式。
- FTE发送的操作数支持指定数量模式（该模式下发送操作数量可配置）和无限发送模式。

须知

非保序模式下，不支持读写对比模式。在读写对比模式下，FTE必须在收到写响应的情况下，再发出相同地址的读操作，读写对比才能通过。即FTE在读写对比模式下，SMMU要负责保序。

- FTE读写操作AXID可配置；支持两种变化方式：AXID固定和AXID递增模式，AXID递增模式时，ID变化的范围可配置。
- FTE读写操作中，QoS、prot、Lock、size、len、cache、User属性可配置， FTE发包过程中所有操作该类属性固定，不支持动态配置。
- FTE读写操作地址基值及变化方式可配置；地址基值可配置，支持所有地址可按比特位mask；地址变化分为4种模式：地址固定模式、地址递增模式、地址循环移位模式0（每个地址数据变化一次）、地址循环移位模式1（地址循环移位一轮数据变化一次）。地址递增模式分为两种：正常增量模式和固定增量模式。正常增量模式下，每次发送命令后地址增量按照len和size决定；固定增量模式下，每次发送命令后地址按配置的固定值递增，此模式下，需要保证地址增量的配置值大于等于由len和size计算得到的增量值，否则为防止地址倾轧，地址仍将按照len和size计算得到的增量值变化。可配置地址的某几bit变化，可以固定冲击L3的某个INDEX。
- FTE工作在只写模式时，写命令的strobe信号支持假写模式（此时USER[FP]必须配置为1），此模式下，FTE发送的所有写命令的strobe信号全为0。
- FTE发出的写数据基值及写数据变化方式可配置；写数据基值通过配置一个字节数据， FTE按照所配置的字节对所有数据位以字节为单位进行复制扩展；FTE基于写数据基值，支持3种数据变化方式：数据固定模式，数据递增模式，数据取反模式，变化过程以所配置的字节为基础，按照字节进行递增、取反操作，然后以字节为单位进行复制扩展；FTE写数据变化频度按照写命令为单位，每发送完一个命令，写数据变化一次。
- FTE寄存器支持DJTAG访问。
- 在读写对比模式下，支持读命令发送是否连续可配置选择。
- 记录对比模式下出错的地址和数据，以及记录最后一次读操作的数据。
- FTE发送操作的间隔周期可配置，读写操作独立可配。
- FTE支持时钟门控，只有在FTE测试时打开FTE时钟。
- FTE支持可编程测试，软件可以在内存中按照FTE定义的可编程命令格式存储一段操作指令，FTE从该段地址依次取指，解析后发送测试命令。
- FTE构造压力测试，压力不小于产品的实际应用，单独启动FTE测试，也可以根据实际场景将相关联的模块串链起来做系统级测试。
- FTE可以发送安全操作，但必须先由安全CPU打开FTE的读写操作安全属性配置使能寄存器，默认情况下FTE发送的所有操作为非安全。

版权所有 © 海思技术有限公司

- FTE工作在ADD和GET POE包模式时，支持512-bit和256-bit长度的数据包，可以通过配置寄存器来实现。
- FTE支持对发送的操作数量和延迟进行统计，可以对错误信息进行记录。
  - FTE对发出的操作数，接收的响应数进行统计，读写分别统计。
  - FTE对操作的平均延迟和最大延迟进行统计，读写分别统计。
  - FTE在读写对比模式下，会对第一个出错的命令的地址和出错数据进行记录，如果操作返回ERROR响应，FTE也会记录出错的命令的地址和出错数据，出错后暂停或者继续发操作可配置。
  - FTE对未完成操作的个数进行统计，读写分别统计。

26.3 遵循协议

AXI APB协议。

26.4 功能描述

26.4.1 写操作功能

支持写数据及其变化方式可配置，包括固定模式、递增模式和取反模式。

- 支持写操作ID及其变化方式可配置，包括固定模式和递增模式。
- 支持写操作的prot、Lock、size、len、cache、User通过寄存器静态配置。
- 支持写地址64位可配置，支持所有地址可按bit位mask。
- 地址变化分为4种模式：地址固定模式、地址递增模式、地址循环移位模式0（每个地址数据变化一次）、地址循环移位模式0（地址循环移位一轮数据变化一次）。
- 地址递增模式分为两种：正常增量模式和固定增量模式。
  - 正常增量模式下，每次发送命令后地址增量按照len和size决定。
  - 固定增量模式下，每次发送命令后地址按配置的固定值递增，此模式下，需要保证地址增量的配置值大于等于由len和size计算得到的增量值，否则为防止地址倾轧，地址仍将按照len和size计算得到的增量值变化。
- 支持写操作命令发送个数可配置，有无限发送模式和指定发送命令个数两种模式。

26.4.2 读操作功能

- ARID初值可配置：

  在读写对比模式下，ARID支持两种变化方式：ARID固定和ARID递增，读操作属性与写操作一致。

  在只读模式下，ARID支持两种变化方式：ARID固定和ARID递增。

  读写对比模式时，FTE对读操作进行数据比对，并记录错误地址、返回的错误数据及其类型。

版权所有 © 海思技术有限公司

26.4.3 ADD 和 GET POE

FTE工作在ADD和GET POE包模式时，软件需要将FTE ADD和GET 包的数据按照FTE定义的包格式准备好。

表 26-2 FTE 定义的包格式

bit位		含义	备注
256bit模式	[511:256]	GET或者ADD HAC时256bit包数据	具体参考各HAC相应的包数据结构。
512bit模式	[767:256]	GET或者ADD HAC时512bit包数据	
[255:157]		保留	-
[156:89]		GET或者ADD HAC时命令User属性	-
[88:85]		GET或者ADD HAC时命令重复次数	发送配置值+1个命令。
[84:82]		发送命令prot属性	-
[81:78]		发送命令QoS属性	-
[77:74]		发送命令cache属性	-
[73:68]		GET或者ADD HAC时使用的ID配置	-
[67:65]		发送命令包size类型配置	ADD HAC模式时，如果该size小于256bit，则将DDR获取的一个包数据内大于SIZE的高位数据丢弃。 配置范围0~5。
[64]		读写操作指示。 0：读操作； 1：写操作。	-
[63:0]		64bit数据地址	-

FTE工作在ADD/GET POE包模式时，需要将FTE发送操作的首地址配置为包数据的起始地址，分两种情况：

1. 包长为256bit，FTE将固定按照SIZE=256bit，LEN=1（2拍）的格式从外存中将软件准备好的数据读回，注意需要配置User[N0]=0（使用SMMU的Reorder Buffer）。FTE解析读到的512bit数据，将[67:65]作为SIZE，将[63:0]作为地址，[511:256]作为数据，判断[64]区分读写后，以SINGLE操作访问POE，直到该操作完成。

版权所有 © 海思技术有限公司

2. 包长为512bit，FTE将分两次从外存中将软件准备好的数据读回。每次读操作固定按照SIZE=256bit，LEN=1（2拍）的格式从外存中将软件准备好的数据读回，注意需要配置User[N0]=0（使用SMMU的Reorder Buffer）。第一次中的第一拍256bit数据作为命令被FTE解析，将[67:65]作为SIZE，将[63:0]作为地址；第一次中的第二拍256bit和第二次中的第一拍256bit读到的数据作为数据，判断第一次中第一拍256bit数据的[64]区分读写后，以INCR操作访问POE，直到该操作完成。

须知

在POE-512bit模式下，外存中的数据（包括第一次第一拍读取的命令、第一次第二拍读取的数据，和第二次读取的两拍数据）应按照128B的块存放，FTE不使用第二次第二拍的数据。从外存中的两次数据读取时，读地址应该按照64B地址对齐。

3. FTE从外存读数据的地址保存在其输出信号araddr或awaddr中，该信号为一个64bit信号。

   FTE继续从从外存中获取下一个包数据，依次循环。

26.4.4 结果及统计功能

- 在命令无限发送模式时，只能通过通过配置寄存器FTE_START为0，才能产生fte_test_finish信号；中途发生数据比对错误时，只会记录第一个错误地址及其类型，数据比对信号FTE_STAT. data_err_flag会拉高，但是不会因此而停止操作。
- 支持通过配置寄存器FTE_START为0，可以在任何时候停止FTE测试。
- 支持统计发出的写操作个数、接收到写响应的个数、写操作最大延时、写操作平均延时,在重新start后清零，重新统计计数，只有在停止后读取的统计值才是准确的值。
- 支持统计发出的读操作个数、接收到读响应的个数、读操作最大延时、读操作平均延时，在重新start后清零，重新统计计数，只有在停止后读取的统计值才是准确的值。

26.5 应用配置

不涉及。

26.6 初始化

FTE为总线冲击模块，FTE的所有行为都由软件配置。实际配置根据软件测试场景不同而不同。

在此给出一个通用的测试流程配置：

步骤1 配置Subctrl中FTE的时钟使能寄存器SC_FTE_CLK_EN为1，打开时钟。

步骤2 配置Subctrl中FTE的软复位去请求控制寄存器SC_FTE_RESET_DREQ，解除FTE复位。

步骤3 配置FTE_START的fte_clke_en域为1，打开FTE模块时钟。

步骤4 如果要使用POE测试模式，则需要配置读写操作模式寄存器FTE_MODE域poe_test_mode为1，否则使用默认值，直接执行**步骤5**。

版权所有 © 海思技术有限公司

步骤5 配置读写操作模式寄存器FTE_MODE域rw_mode，配置FTE的工作模式，如果是POE测试模式，则该域必须配置为读写非对比模式。

步骤6 配置读写操作模式寄存器FTE_MODE域addr_common_mode,addr_incr_mode,addr_increment，配置FTE地址变化模式。

步骤7 配置FTE_MODE域cmd_endless选择是否为无限发送模式；如果为无限模式，则跳过**步骤3**。

步骤8 配置FTE_CMD_NUM，设置发送命令个数的初始值。

步骤9 配置FTE_ADDR_EN_H/L、FTE_ADDR_H/L，设置FTE初始地址。

步骤10 配置写操作控制寄存器FTE_AW_CTRL、FTE_AR_CTRL，配置AXI接口传输属性，如果是POE测试模式，特别注意需要根据测试空间的mem属性分别对awcache和arcache信号进行配置（读写的cache属性可能不同）。

步骤11 配置写数据寄存器FTE_WDATA_CFG，配置写数据的初始值和数据变化模式。

步骤12 配置FTE_START寄存器fte_test_start域，先拉低再拉高，启动扫描测试。

步骤13 读FTE_START寄存器，确认该寄存器为1，测试开始。

步骤14 如果为指定命令个数发送模式，则通过读取结果寄存器，查看fte_test_finish域为高，则表示发送完成，通过读取FTE_AW_RNUM和FTE_AR_RNUM的值看知否和配置的发包个数相等，如果相同则表示操作接收完成。

步骤15 如果为无限发送模式，fte_test_finish信号不会拉高，需要配置FTE_START寄存器fte_test_start域为低来停止FTE发包，然后读取并判断FTE_AW_CNUM=FTE_AW_RNUM和FTE_AR_CNUM=FTE_AR_RNUM，表示FTE所有发送的命令已经接收完成。

步骤16 确认FTE所有命令接受完成后，可重复上述**步骤1**~**步骤11**，重新开始新一轮测试。

----结束

26.7 注意事项

1. FTE工作在读写对比模式时，如果发送的操作为非保序操作（User中的no域为1），需要注意配置：

   a.操作的id变化方式需要配置为递增模式，id变化的范围使用默认值。

   b.地址变化方式不支持地址循环移位方式。

   c.发送操作的size需要固定为4（128bit），不能发送narrow操作。

2. 对于保序操作，SMMU对于长包（数据量大于64B的操作，仅支持128B，192B,256B）操作不能保证顺序， 因此需要限制FTE发包的格式，建议使用AXI size为4（128bit），len为3（4拍）的操作类型。
3. 对于软件的异常配置，如寄存器中未定义或者reserve的值，会导致FTE发出不可预知的操作，这些操作不会造成FTE挂死，但是可能会造成系统异常。
4. 针对POE测试模式，因为地址可以任意配置，所以测试应用场景中可以不限于对POE进行ADD和GET包操作。但是针对扩展的测试场景，该测试模式不支持ADD（写）和GET（读）操作存在依赖的情况，这种场景下会有因互锁而造成挂死发生。例如：测试ITS中断时，利用FTE的POE模式写ITS，ITS的写完成依赖于GIC的读，FTE从ddr中读回来的数据因为写没法下发导致无法接收读操作，进而阻塞GIC的读完成，形成互锁。如果要进行该类测试，测试使用的FTE需要与被测试的加速器master不能在同一个Scheduler中，以避免互锁问题的发生。

版权所有 © 海思技术有限公司

5. 使用FTE冲击POE、BMU、TM、AQM的Slave口时，由于SMMU的资源有限，如果FTE发送操作的Outstanding太大，超过了SMMU的资源上限，则有可能导致挂死，为了避免这一现象，同时保证压力测试正常进行，在测试POE、BMU、TM、AQM时，不要使用和这些加速器同Subsys的FTE。

某些场景下，不得不使用同Subsys的FTE，此时需要限制FTE的Outstanding，具体限制的大小要与SMMU的资源进行匹配。一般情况下，需要将FTE的读写的Outstanding限制在24（读写各24）以下，该约束需要通过配置Scheduler模块内部对应FTE接口的outstanding limit寄存器实现。

6. 如果使用FTE发送读操作进行冲击测试，当每拍读命令对应返回多拍读数据时，由于总线频率比FTE工作频率高且位宽是FTE的2倍，造成总线返回的读数据FTE不能马上接收，当冲击的读数据数量累积后，会阻塞其他端口的读数据的返回，而影响其他端口的读操作响应速率，因此需要对FTE的读操作的冲击强度进行限制。

   建议当使用FTE读操作进行冲击测试时，将读命令的发送间隔进行控制，具体约束与发包长度进行匹配。

   下面给出几种典型的配置参考：

   Ar_len=0，可以不约束读命令发送间隔。

   Ar_len=1，配置FTE_INTERVAL_CYCLE中的[15:0]为1。

   Ar_len=3，配置FTE_INTERVAL_CYCLE中的[15:0]为3。

26.8 DFX

FTE支持对发送的操作数量和延迟进行统计，可以对错误信息进行记录。

FTE在读写对比模式下，会对第一个出错的命令的地址和出错数据进行记录，如果操作返回ERROR响应，FTE也会记录出错的命令的地址和出错数据，出错后暂停或者继续发操作可配置

FTE对未完成操作的个数进行统计，读写分别统计。

在没有错误的情况下，记录最后一次读操作的地址和数据。

在非保序读写对比场景下，FTE记录的错误地址并非最终的错误地址，最终的错误地址按照如下方式计算得到：

```
err_addr_mid=err_addr + rid*(burst_size);
err_addr_final =（err_addr_mid & ~addr_en）|（fst_addr_cfg & addr_en）。
```

26.9 性能指标

26.9.1 吞吐率（Throughput）

FTE可以按照最大频率无间隔的发送命令和数据，在下方接口无反压的情况下，800MHz，128bit数据位宽的情况下，发送速率为：

128bit x 800M=100Gbits

版权所有 © 海思技术有限公司

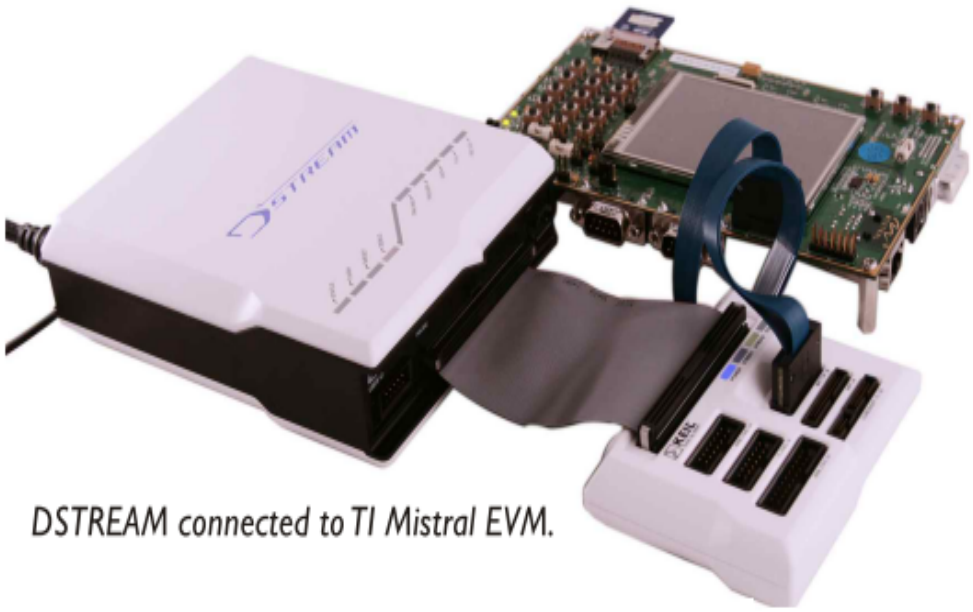
27 Coresight

27.1 概述

仿真器基础知识介绍

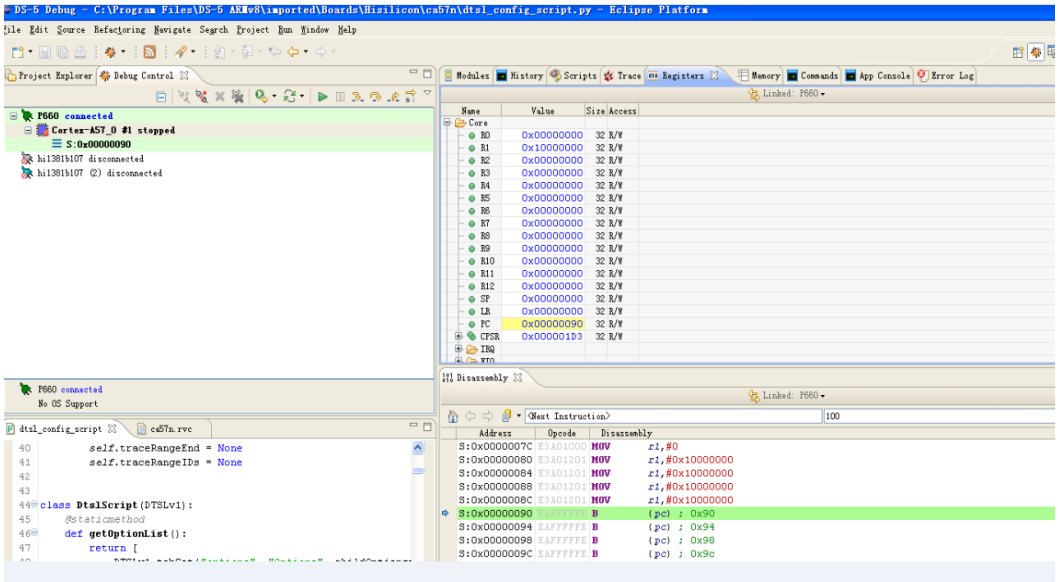
仿真器是由arm ， 劳德巴赫等公司生产的，通过一套jtag接口连接CPU，控制cpu的运行暂停，读取CPU状态等功能的硬件。和这样一套硬件配套的都会有一个基于PC或者工作站的图形界面的软件（ arm叫ds-5，劳德巴赫的叫trace32 ），通过这个软件界面，工程师可以让仿真器连接上芯片（ 或者FPGA，或者emulator ），然后让CPU暂停，读取CPU的寄存器，让CPU单步执行等。这样就可以让软件工程师在芯片，FPGA等硬件上看到CPU内部的状态，加快软件调试的速度。

图 27-1 Arm 的仿真器硬件



版权所有 © 海思技术有限公司

图 27-2 Ds-5 软件界面



调试方式分为如下两种：

1. debug 侵入式调试

   只能打断点，然后停下来查看当前CPU状态寄存器值等信息，然后单步运行等。这种调试方式只需要仿真器有一个jtag口和网口连接上芯片和PC即可。

2. trace 非侵入式调试

   在debug基础上，一次性跑很长一段程序，停下来之后还能够追溯CPU以前执行程序的顺序。以及各个时刻CPU的状态寄存器值等信息。不干扰CPU的运行。这种调试方式需要仿真器有一个jtag口连接芯片发送命令，一个网口连接PC，一个专门的trace数据传输的接口和连接线（在芯片里面我们称之为ATB总线，是个单向传输的总线）。

   图27-3为仿真器的连接示意图，PC和仿真器通过网线连接到同一个局域网，然后通过网口通信，仿真器通过jtag连接到芯片上的coresight逻辑，coresight逻辑把jtag接口转换成apb接口连接到各个需要通过仿真器调试的设备（可以同时连接到多个不同的CPU，其他需要调试的模块等）。如果需要trace功能，那就需要芯片上面有一个ATB总线把trace数据从CPU传递到芯片的管脚，然后通过专门的线连接到芯片上（或者通过芯片中集成一个sram存下数据或者集成一个专门的模块把trace数据存入ddr，然后通过jtag口导出去）。

   由于考虑到有的时候需要从CPU以外的设备过来读取CPU的pmu中统计的数据的需求，因此arm的CPU中的PMU模块一般都是支持从coresight的APB口直接过来访问的，也就是在这个apb口处挂接了个读写口以方便其他设备读写。

   如果这个CPU的配置支持trace的话，那CPU中就会有一个产生trace数据的模块PTM（在A7，A57等CPU也叫做ETM作用差不多，但是貌似A7的ETM不只是支持程序流trace，还支持数据trace，这个模块会直接从core逻辑中提取需要的trace数据，然后通过ATB总线送出去。如果这个CPU配置不支持trace的话，就不会有这个模块，那整个ATB通路的模块就都不需要了。

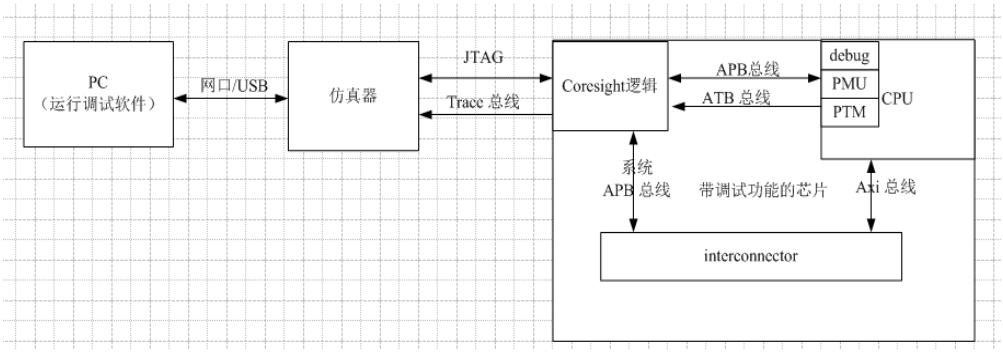
   在任何配置下，CPU都会有一个debug模块，这个模块里面的寄存器一般都提供两个读写口，一个是通过cp14配置，另一个是从coresight的APB总线读写。在CPU的运行中，可以通过coresight 的APB写一个寄存器（DBGDSCR 寄存器，CP14访问方式为MCR P14，0,r0,c0,c4,2），让CPU进入一种特殊的状态叫做debug状态，这个时候CPU暂停执行指令，但是CPU的各个寄存器的值都可以读出来，在

版权所有 © 海思技术有限公司

这个时候可以通过coresight的APB总线向debug模块的一个寄存器写入一条指令，然后通过其他寄存器让CPU在暂停的状态单步执行一下这条指令。仿真器的单步执行功能就是通过不停的写这些寄存器实现的。在CPU进入暂停的debug状态的时候CPU顶层信号dbgack会变成1，退出debug态的时候为0。

既然coresight是通过一个APB口连接到CPU里面的debug，PMU，PTM等模块的，那么，这个APB的源头也支持两种，第一种就是传统的仿真器使用的Jtag口，另一种就是可以让coresight提供一个APB slave口连接的芯片的系统总线上，这样CPU自己或者其他master也可以通过这个slave口发送命令到coresight逻辑，然后coresight逻辑发送到debug，PMU，PTM等模块中去访问相应的寄存器。值得注意的是，为了防止安全数据泄露，这两种方式能够访问的寄存器是有区别的，这种区别会体现在APB地址的最高位上。

图 27-3 仿真器连接示意图。



Coresight逻辑是通过arm发布的一个IP包，但是arm只是提供了各个小小的功能相对封闭的小模块，我们需要根据我们的系统搭建出来一个符合特定需求的coresight逻辑系统，然后arm还提供了一个工具叫做ambadesigner，这个工具中可以按照arm提供的各个封闭的小模块，然后通过手动连线，搭建出来我们想要的coresight系统，然后通过这个工具生成rtl代码。

27.2 特性

Hi309a Coresight 具有以下特性：

- 支持普通单核调试功能。能够用仿真器接上之后控制一个核的暂停，运行等。
- 支持多核MP调试功能。能够用仿真器连上之后控制多个核同时暂停，同时运行等。
- 支持单核/多核trace功能。
- 支持trace功能，支持芯片把Trace信息存储到片上SRAM\DDR或ETM设备。
- 支持内存数据硬断点，断点方式包括单地址、地址范围，可同时设置6个数据硬断点；当内存数据访问匹配上述地址或地址范围时，触发异常或中断通知软件记录现场信息；
- 支持代码的模块加载及单步调试。
- 预留一组CTM接口给顶层；预留ddr triger相关接口信号。
- DAPAPBAP模块拉到顶层；即本cssys不处理jtag总线。
- 集成了Counter模块，使用见下述章节。

版权所有 © 海思技术有限公司

27.3 关键术语

如表27-1的术语将在本章节表示特殊的含义。

表 27-1 CoreSight 关键术语描述

术语	描述
ECC	Error Checking and Correcting，错误检测和纠正。
MBIST	Memory Build In Self Test，内存自检。
JTAG	Joint Test Action Group。
DJTAG	Debug JTAG。
Serdes	英文Serializer（串行器）/Deserializer（解串器）的简称。

27.4 遵循协议

遵守ARM DEBUG V8协议规范要求。

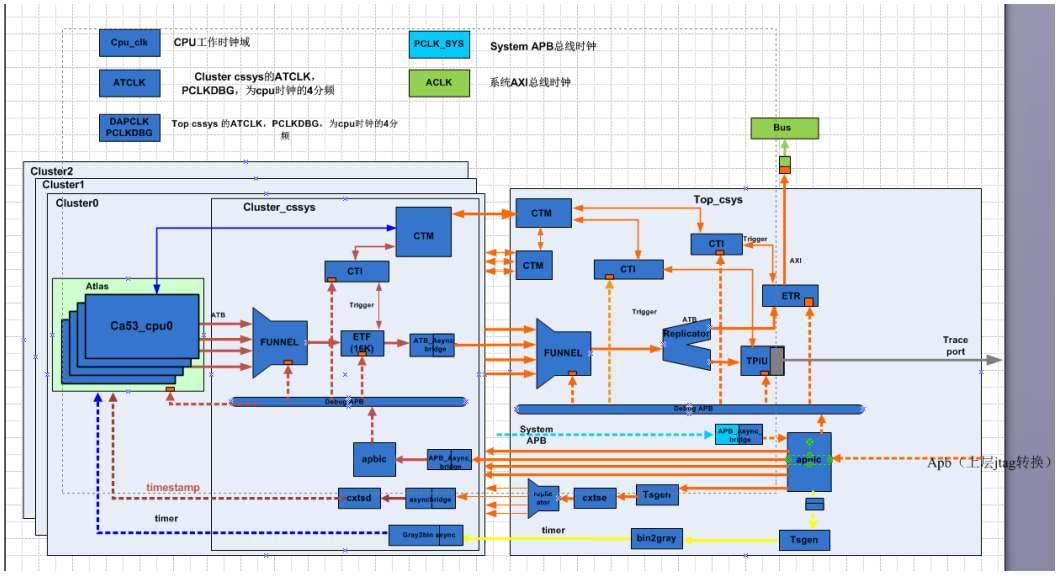
27.5 功能描述

Coresight组件可以简单的划分为控制总线通路模块（下图中3号框中的模块），trace数据源模块（下图27-4中的ca55 CPU中的ETM），trace数据总线模块（下中2号框中的模块），传递trigger信息的CT模块（下中1号框中的模块），产生时间戳的tsgen模块（下中4号框中的模块）等。

在不同的coresight组件IP包中所含有的模块种类和数量都不太完全相同，具体哪个版本的coresight组件IP包中含有哪些组件就需要查看相应的文档。但是，基本功能的组件一般都会有的。

版权所有 © 海思技术有限公司

图 27-4 Hi309a 芯片的 CPU CLUSTER coresight 系统



27.5.1 控制总线通路类模块

这一类模块基本上就是把JTAG协议转换成内部定义的DAP总线然后通过一系列不同的模块转换成不同的诸如APB，AXI，JTAG等通用总线协议。如果一个芯片只想要有debug功能，那就只需要有这一类组件搭建一个从JTAG转换成APB然后接到CPU里面的debug模块就可以了。下面就逐一介绍这一类型的各个模块的功能。

27.5.1.1 DP

这类模块是用来把从仿真器送过来的JTAG转换成coresight内部通用的DAP总线（这种总线非常类似APB总线）的。但是，由于支持的master口的协议不同可以分为三种：

1. JTAG-DP，只能把JTAG协议转换成DAP总线。

   主要接口： JTAG slave；DAP master；

2. SW-DP，只能把SW协议的接口转换成DAP总线。这里的SW总线是另外一种调试总线，一共就只有2跟线比JTAG还省线，但是，目前我们还没见哪个仿真器是用这种总线连接芯片然后进行调试的。

   主要接口： DAP master；SW slave

3. SWJ-DP，可以把SW协议和JTAG协议转换成DAP总线，我们绝大部分的芯片用的是这个DP。在这个模块的接口处提供了两套协议的信号，但是我们绝大部分芯片都是把SW总线的线和JTAG的那些线复用了，这样还是只需要JTAG的那一套线了。

   主要接口： JTAG slave ； DAP master；SW slave

27.5.1.2 AP

这类模块是用来把内部DAP总线转换成为各类标准协议的master的模块。

1. APB-AP： 用来把DAP总线转换成一个APB的master口。

   主要接口： DAP slave；APB master ；

版权所有 © 海思技术有限公司

2. AHB-AP： 用来把DAP总线转换成一个AHB的master口。
   主要接口： DAP slave；AHB master；
3. JTAG-AP： 用来把DAP总线转换成一个JTAG的master口。
   主要接口： DAP slave；JTAG master；

27.5.1.3 DAPBUSIC

这个模块是一个DAP总线的interconnector，把一个DAP总线的master（通常就是DP），MUX到多个DAP总线的slave（各种AP等）。模块本身的master口可配， slave口固定为1，值得注意的是，听说就算只有一个DP一个AP，最好也在ambadesigner中连接的时候在他们中间加一个1mater-1slave的DAPBUSIC。不然会扫描不到组件。

主要接口： DAP master；DAP slave ；

27.5.1.4 APBIC

这个模块就是APB的一个interconnector，可以挂接各种APB master和slave设备。值得注意的是，这个模块配置的时候需要给各个slave配置需要的地址空间和基地址，而且，各个地址空间必须是对齐的。就是说低N位必须是从0到全1。

主要接口： APB master；APB slave ；

27.5.1.5 APBASYNCBRIDGE

这是一个APB异步桥

主要接口： APB master；APB slave ；

27.5.1.6 APBSYNCBRIDGE

这是一个APB同步桥

主要接口： APB master；APB slave ；

trace 数据源模块

这一类模块主要是CPU内部的PTM，ETM等，但是，arm还在coresight IP包中提供了一个特殊的用于trace SOC的STM模块，这个模块提供一个AXI接口挂接在SOC的系统总线上，然后通过一个ATB口把数据传到通用的coresight trace通路上去，然后发送到仿真器上去。但是，arm只提供这样一个硬件通路，并没有固定这个通路中写的数据格式，和协议等，用户可以自己按照一定的格式通过AXI口把数据写入STM，然后导入到仿真器之后自己再把这些数据解析一下，就可以得到想要的信息了。

27.5.2 trace 数据总线模块

这一类模块负责把trace数据源模块产生的trace数据，用ATB总线一级级的传递到芯片的管脚然后发送到仿真器。

27.5.2.1 FUNNEL

这个模块是一个把多路数据路合并为一路的模块。

这个模块的主要可配项有：

版权所有 © 海思技术有限公司

1. ATB总线数据位宽（8bit，16bit，32bit，64bit，128bit）
2. slave端口数量（2-8）

   主要接口： ATB master；ATB slave ； APB slave；

27.5.2.2 REPLICATOR

这个模块是用来把一路数据分发到多路ATB总线的模块，使得一路trace 数据可以同时送到多个不同地方，通过不同方式保存trace数据，或者实时导出。

这个模块的主要可配项有：

- ATB总线数据位宽（8bit，16bit，32bit，64bit，128bit）

  主要接口： ATB master；ATB slave

27.5.2.3 TPIU

这个模块是用来把ATB总线格式的trace数据转换成32bit或者16bit的数据输出到芯片管脚的模块。这个模块的输出数据固定为32bit，但是内部有寄存器可配，配置之后可以改为16bit模式，这种情况下高16bit就可以悬空。

主要接口： ATB slave； APB slave；trace 数据口

27.5.2.4 TMC

这个模块又可以配置成3种功能的模块，

1. ETB

   这个模式就是一个trace专用的片上Buffer，需要集成一个sram，把trace数据存到这个sram中，然后停止trace之后，通过APB口读出到DP模块，然后通过JTAG口读到仿真器里面去，实现trace功能。

   主要接口：ATB slave； APB slave； sram接口

2. ETF

   配置成这种功能之后，除了能够满足上面ETB配置的功能外，还能充当缓冲trace数据流的作用。因此工作原理就是接受ATB发来的数据然后缓存到sram，然后再通过ATB发送给下一级，可以在数据量带宽不稳定的时候，削平波峰的作用。

   主要接口：ATB master；ATB slave； APB slave； sram接口

3. ETR

   配置成这种功能之后，这个模块就不会需要集成sram，但是会多一个AXI接口，他的功能就变成把ATB的trace数据通过AXI协议写到总线上，最后写入ddr等SOC的存储空间上去，然后，再通过APB-JTAG，或者其他方式读到仿真器上面去。这样，可存储的trace数据量就不受限于这里集成的sram大小了。但是，需要注意的是，这个模块的AXI master口的五个通道都是没有ID的。

   主要接口： ATB slave ； APB slave； AXI master

   这个模块的其他可配置选项：

   a. 集成的sram大小（512Byte到4G）；
   b. 内部一个Buffer的大小（4-32个entry）。

27.5.2.5 ATBUPSIZER，ATBDOWNSIZER

这两个模块是用来转换ATB的总线数据位宽的，不详细介绍了。

版权所有 © 海思技术有限公司

27.5.2.6 此类别的其他模块

其他模块还有ATB同步桥，ATB异步桥之类的模块不详细介绍了。

27.5.3 传递 trigger 信息的 CT 模块

这类模块是用来传递trigger信息的，通过CTM连成一个大的网络，然后通过CTI把这个网络的各个节点连接到各个需要传递trigger信息的模块中，这样任何一个模块有需要传递的trigger信息时，只要通过他的CTI一发送，整个系统中的其他任何模块都可以收到这些信息了。

例如，trace数据流中的ETB模块发现数据存满了，他就会触发他的CTI然后发送到CTM网络，然后送到PTM的CTI让PTM别再产生trace数据了。

再例如，在MP系统的debug模式中，仿真器只连接到了一个core，但是通过这个core也可以让其他所有的core进入debug模式，这时候仿真器会设置这个core的CTI的寄存器触发一个trigger，然后通过CTM网络，进入其他core的CTI，让其他core也进入到debug模式。

因此这一类一共就两个模块：CTI，CTM。

27.5.3.1 CTI

可以产生trigger信号也可以传递他的宿主的trigger信号。一般来说是所有需要传递trigger信号的模块，每个模块会配一个CTI，这些需要trigger信号的模块主要是trace源模块和trace数据总线模块等。

主要接口： APB slave； event interface in； event interface out ；channel-interface master

27.5.3.2 CTM

联系系统中所有的CTI，CTM等，让所有的trigger信号可以传递到任何终点去。

主要接口： channel-interface master； channel-interface slave

27.5.4 产生时间戳的 tsgen 模块

这一类模块主要是用来产生一个不停增加的计数值，用来给ATB总线传递的trace数据打上一个时间戳的作用，同时，这个时间戳还可以用来同步不同ATB通路上的trace数据，确保trace的不同核的指令是具有同步性的。

由于这个是产生一个不停加1的计数值的，而且位宽是64bit，因此，可以当做CPU的counter 产生原始的计数值用。

27.5.4.1 TSGEN

这个模块就是产生一个不停加1的64bit的计数值的。但是还提供两个暂停信号。

主要接口： APB slave；64bit计数值

27.5.4.2 TSE

把64位的TS计数值encoder成10bit的值的模块。

主要接口：10bit计数值；64bit计数值

版权所有 © 海思技术有限公司

27.5.4.3 TSREPLICATOR

把10bit的计数值复制成多份的模块。

27.5.4.4 TSD

把10位的TS计数值decoder成64bit的值的模块。

27.5.5 此类别的其他模块

这一类模块也有同步桥和异步桥之类的模块。

27.6 工作方式

27.6.1 内部的 Counter 模块的使能方式

本coresight中集成了两个Counter，给CPU作为系统Counter的计数器 在使能之后会不停的加数，一直工作到芯片报废，可以计数十年以上。另外一个是timestamp的计数器，timestamp的Counter支持外面控制使能，两个Counter均可以寄存器使能和置位，使用方式见下。

27.6.1.1 Counter 模块的寄存器

Couter模块中有如下寄存器：

图 27-5 Timestamp generator

Timestamp generator register summary

Offset	Name	Type	Description
PSELCTRL region			
0x000	CNTCR	RW	*Counter Control Register, CNTCR* on page 3-205
0x004	CNTSR	RO	*Counter Status Register, CNTSR* on page 3-205
0x008	CNTCVL	RW	*Current Counter Value Lower register, CNTCVL* on page 3-206
0x00C	CNTCVU	RW	*Current Counter Value Upper register, CNTCVU* on page 3-206
0x020	CNTFID0	RW	*Base Frequency ID register, CNTFID0* on page 3-207

27.6.1.2 CNTCR

这个是控制寄存器，给bit0 写入1或者0可以控制counter的打开与关闭。

系统Counter使能方法是，在安全模式下，没连接仿真器，向地址（coresight基地址 + 0x8000）写入数值3，进入debug态的时候，就会停止计数，如果写入1，进入debug时候不会停止计数；当CPU已经进入debug态，还需要使能计数的时候，只能向这个寄存器写1，否则不会计数；

timestamp使能方法是，向地址（coresight基地址 + 0x6000）写入数值3，就会开始计数；

版权所有 © 海思技术有限公司

图 27-6 CNTCR

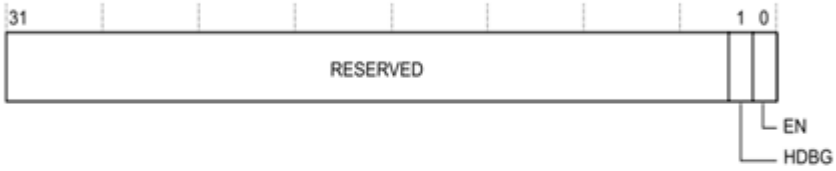
Counter Control Register, CNTCR

The CNTCR characteristics are:

Purpose Controls the counter increments.

Usage constraints This register is not accessible to the read-only programming interface.

shows the bit assignments.



CNTCR bit assignments

shows the bit assignments.

CNTCR bit assignments

Bits	Name	Function	
[31:2]	UNK/SBZP	Reserved	
[1]	HDBG	Halt on Debug.	
		0	Do not halt on debug. **HLTDBG** signal into the counter has no effect.
		1	Halt on debug, when **HLTDBG** is driven HIGH, the count value is held static.
[0]	EN	Enable.	
		0	The counter is disabled and not incrementing.
		1	The counter is enabled and is incrementing.

27.6.1.3 CNTSR

这个是只读寄存器，可以读出当前counter是否被halt住了。

系统counter访问地址是（coresight基地址 + 0x8000+0x004）.

Ts counter访问地址是（coresight基地址 + 0x6000+0x004）.

版权所有 © 海思技术有限公司

图 27-7 CNTSR

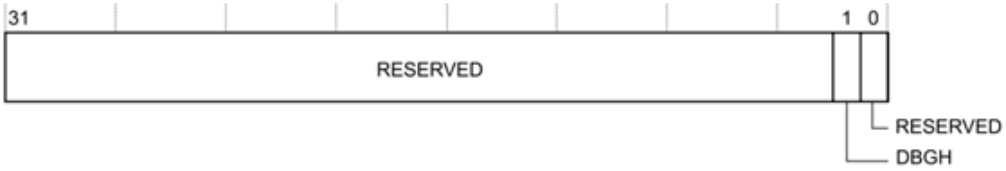


Table 3-248 CNTSR bit assignments

Bits	Name	Function
[31:2]	UNK/SBZP	Reserved.
[1]	DBGH	Debug Halted.
[0]	UNK/SBZP	Reserved.

27.6.1.4 CNTCVL

这个是counter的值的低位配置寄存器，可以人工修改counter的值到某一个数，或者读出当前counter值的低位。

系统counter访问地址是（coresight基地址 + 0x8000+0x008）.

TS counter访问地址是（coresight基地址 + 0x6000+0x008）.

图 27-8 CNTCVL

Current Counter Value Lower register, CNTCVL

The CNTCVL register characteristics are:

Purpose Reads or writes the lower 32 bits of the current counter value.

Usage constraints The read-only programming interface can read but not write to this register. The control interface must clear the CNTCR.EN bit before writing to this register.

CNTCVL register bit assignments

CNTCVL register bit assignments

Bits	Name	Function
[31:0]	CNTCVL_L_32	Current value of the timestamp counter, lower 32 bits. To change the current timestamp value, write the lower 32 bits of the new value to this register before writing the upper 32 bits to CNTCVU. The timestamp value is not changed until the CNTCVU register is written to.

27.6.1.5 CNTCVU

这个是counter的值的高位配置寄存器，可以人工修改counter的值到某一个数，或者读出当前counter值的高位。

版权所有 © 海思技术有限公司

系统counter访问地址是（coresight基地址 + 0x8000+0x00c）.

TS counter访问地址是（coresight基地址 + 0x6000+0x00c）.

图 27-9 CNTCVU

Usage constraints The read-only programming interface can read but not write this register. The control interface must clear the CNTCR.EN bit before writing to this register.

shows the bit assignments.

CNTCUV register bit assignments

shows the bit assignments.

CNTCUV register bit assignments

Bits	Name	Function
[31:0]	CNTCVU_U_32	Current value of the timestamp counter, upper 32 bits. To change the current timestamp value, write the lower 32 bits of the new value to CNTCVL before writing the upper 32 bits to this register. The 64-bit timestamp value is updated with the value from both writes when this register is written to.

27.6.1.6 CNTFID0

这个是counter 当前工作的频率寄存器，可读可写，但是这个值仅仅是软件可读，没有任何实际作用。建议根据当前的频率配置上，然后软件人员就可以只读这个寄存器就知道这个counter的频率是多少。但是，不配上也没任何问题，那软件人员就得查询相关文档才知道频率是多少。

系统counter访问地址是（coresight基地址 + 0x8000+0x020）.

版权所有 © 海思技术有限公司

图 27-10 CNTFID0

Base Frequency ID register, CNTFID0

The CNTFID0 register characteristics are:

Purpose You must program this register to match the clock frequency of the timestamp generator, in ticks per second. For example, for a 50 MHz clock, program `0x02FAF080`.

Usage constraints This register is not accessible to the read-only programming interface.

CNTFID0 register bit assignments

CNTFID0 register bit assignments

Bits	Name	Function
[31:0]	FREQ	Frequency in number of ticks per second. You can specify up to 4GHz.

27.6.1.7 修改 counter 原值的步骤

例如当前counter 计数到了100，而由于系统同步的需求需要立即同步到10000.那配置系统counter步骤如下：

步骤1 往地址（coresight基地址 + 0x8000+0x000）写入0x0，停止当前counter的计数。

步骤2 往地址（coresight基地址 + 0x8000+0x008）写入新值‘d10000。

步骤3 往地址（coresight基地址 + 0x8000+0x000）写入0x3，重新使能 counter的计数。

----结束

TS counter修改办法，寄存器配置方式和上述类似，只是TS counter可读写基地址为0x6000，不过多说明；外部控制TS counter使能的使用限制如下：

Clear：先暂停TS counter计数（寄存器配置禁能，外部encnt无效），然后向地址coresight基地址 + 0x6000+0x008）写入新值0，就实现了置位为0功能（置位为其他值类似），想要再次使能计数，可以通过拉高外部控制信号encnt，就会从0重新开始计数。

27.6.2 各模块的偏移地址

图27-11是coresight系统中各个模块的偏移地址的图，一共是三级结构，总共需要32M地址空间，下图中第三个图是第一次map图，第四个图上半部分是第二级map图，第四个图下半部分是第三级map图：

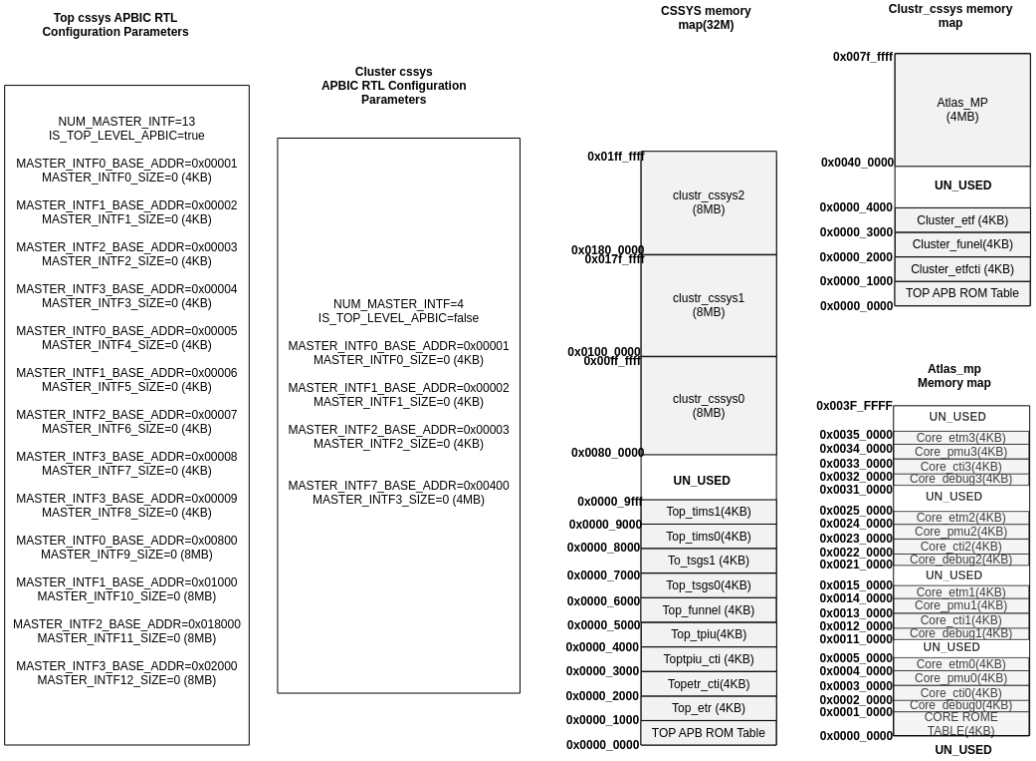
因此，从图27-11可以看出cluster0_cpu0（core_debug0）的偏移地址是: 0x00800000 + 0x00400000 + 0x00010000 = 0x00c10000（实际上，这个偏移地址的最高位经常用来区分是不是jtag访问，因此仿真器访问的地址经常需要把最高位置1，因此就是0x10c10000）

从图27-11可以看出cluster1_cpu2（core_debug2）的偏移地址是: 0x01000000 + 0x00400000 + 0x00210000 = 0x01610000（实际上，这个偏移地址的最高位经常用

版权所有 © 海思技术有限公司

来区分是不是jtag访问，因此仿真器访问的地址经常需要把最高位置1，因此就是0x11610000 ）

图 27-11 各模块偏移地址



TOP CSSYS的组件tsgs0 tsgs1是tscounter产生组件，区别是tsg0是可读写，tsg1只读；times0 times1是系统counter产生组件，0和1区别与上述类似；a55采用v8架构。

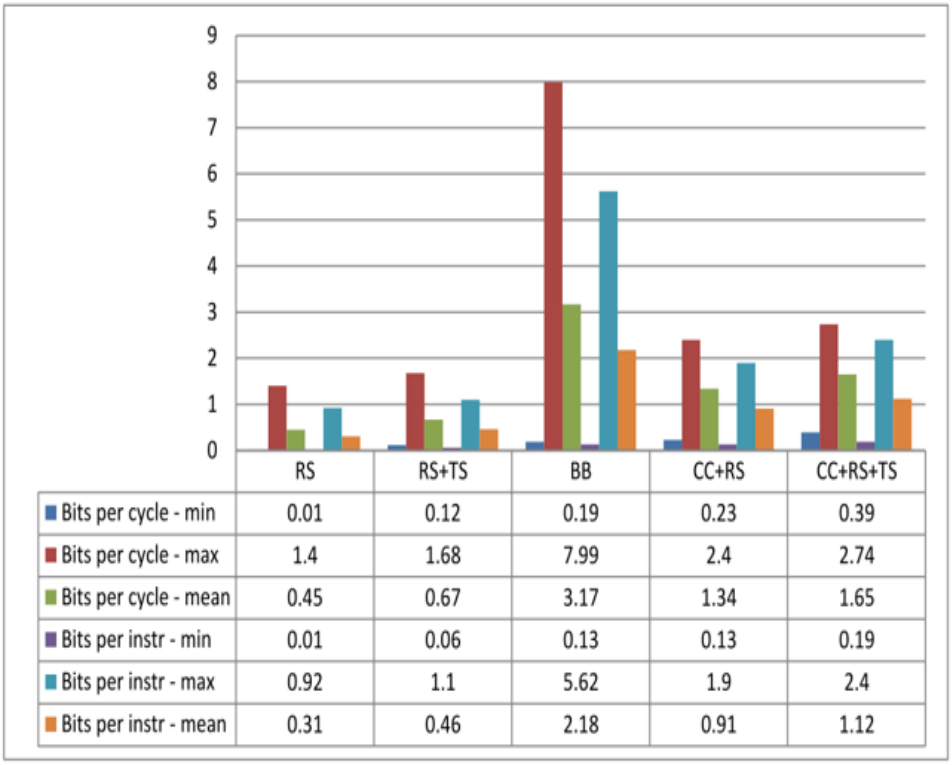
27.7 性能

版权所有 © 海思技术有限公司

27.7.1 应用场景

图 27-12 Effect of trace configuration

The effect of varying the trace configuration is shown in the following diagram. The graph shows the minimum, maximum and mean values for the bits of trace per cycle and per instruction with no overflows.



	RS	RS+TS	BB	CC+RS	CC+RS+TS
Bits per cycle - min	0.01	0.12	0.19	0.23	0.39
Bits per cycle - max	1.4	1.68	7.99	2.4	2.74
Bits per cycle - mean	0.45	0.67	3.17	1.34	1.65
Bits per instr - min	0.01	0.06	0.13	0.13	0.19
Bits per instr - max	0.92	1.1	5.62	1.9	2.4
Bits per instr - mean	0.31	0.46	2.18	0.91	1.12

Legend：

RS：Return stack enabled

TS：Timestamping enabled

BB：Branch broadcasting enabled

CC：Cycle counting enabled

TPIU数据输出bit位宽为32bit，如果CPU频率和IO PAD翻转率比例为2.4G：200M（100M的PAD 上下沿采样）。

则如果使用通常建议的RS+TS模式下，则：

0.68*N（Number of core）*12=32

N=3.98

除去ATB总线的损耗，大概在这种模式下可同时trace 3核，数据不丢失。

但是，实际上我们的coresight系统中还有一个4k的fifo，如果同时4核以上，也是可以trace相当长一段程序流的，便于调试程序。此时建议关掉TS，减少trace数据量。

版权所有 © 海思技术有限公司

27.7.2 注意事项

本coresight的TPMAXDATASIZE 接的是0x1F，表示最大支持32bit trace，Hi309a接出去16根trace数据线。

TPIU 有两个寄存器：

1. 第一个寄存器表示本芯片支持的trace管脚是多少根，由输入的TPMAXDATASIZE信号决定。
2. 第二个寄存器表示当前使用多少根trace线在传输trace数据（默认值是1bit）。

因此需要注意，如果芯片实际接出去的线和TPMAXDATASIZE 值不同，那就需要在连接上仿真器之后，用户配置模式的时候将寄存器2配置为16bit。

图 27-13 TPIU 寄存器

0x000	Supported_Port_Sizes	RO	0x00000001	*Supported Port Size register* on page 3-99
0x004	Current_port_size	RW	0x00000001	*Current Port Size register* on page 3-103

版权所有 © 海思技术有限公司

28 IPC

28.1 概述

IPC可用于核间、进程间的通信。当多个进程需要共享外设或Memory及各进程之间互发中断时，可以实现进程之间的通信。主要完成以下功能：

1. 发送中断给其他的进程
2. 传递少量数据到其他的进程

Mailbox既可以作为进程之间不固定的资源，也可以作为某些特殊进程的固定资源，并且源进程可以有多个Mailbox同时并行发送消息。

Hi309a的IPC模块支持A55_MP4四个核之间的通信，以及1个UVP核和A55核之间的通信，包含16个Mailbox，每个Mailbox包含8个数据寄存器。

Hi309a中针对IPC，CPU和DSP的编号为：

ARM_0：bit0；

ARM_1：bit1；

ARM_2：bit2；

ARM_3：bit3；

ARM_4：bit4；

ARM_5：bit5；

ARM_6：bit6；

ARM_7：bit7。

28.2 依赖关系

1. 包含1个APB配置口用于内部寄存器的配置和状态上报。
2. 输出8中断信号分别送给A55_MP4。

IPC模块送给ARM CPU分配的中断编号如**表28-1**所示，详细信息请参见《Hi309a 中断向量表》。

版权所有 © 海思技术有限公司

表 28-1 IPC 中断映射表

序号	中断编号	中断源	描述	输入源
0~7	764~771	INT_IPC	IPC的组合中断。	IPC

28.3 操作步骤

28.3.1 单个目的的处理器通信

为了便于理解，以处理器0，1做为源处理器，处理器2做为目的处理器，使用Mailbox4进行说明，其他组合依此类推。具体步骤如下：

步骤1 处理器0通过写源处理器申请获得Mailbox4的控制，并且通过设置源寄存器将CORE0或者CORE0~1任意几个设置为源处理器，从而实现1到1和多到1的应用场景。

说明

处理器在此用例中作为主控处理器，可以通过一个Mailbox完成多到1的通信，也可以申请多个Mailbox完成多到1的通信。

bit	Access	Name	寄存器描述	配置值
MBX_SOURCE: i×0x40，i为Mailbox编号，取值范围0~15				-
[31:6]	-	reserved	reserved	0x0
[5]	RW	source	预留处理器bit位，软件固定配置为0	0x0
[4:0]	RW	source	邮箱通道占用源处理器配置，可配置多个处理器共同占用此邮箱通道，每一bit对应一个占用通道的处理器。 bit[3]：ARM_4； bit[2]：ARM_3； bit[1]：ARM_1； bit[0]：ARM_0。 … 配置后写相同值释放该通道邮箱，该邮箱除数据寄存器、模式寄存器之外的寄存器均清0，配置后写其他值无意义。	0x3

步骤2 处理器0回读源寄存器，确认配置正确。

步骤3 处理器0通过配置中断屏蔽寄存器的相应的bit0到bit4位，不屏蔽处理器0~4的中断。

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
MBX_IMASK: 0x014 + i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	reserved	0x0
[5]	RW	int_mask	预留处理器bit位，软件固定配置为0	0x0
[4:0]	RW	int_mask	邮箱通道到处理器的中断屏蔽设置，每一bit对应一个处理器。 0：不屏蔽； 1：屏蔽。	0x0

步骤4 处理器0通过配置目的寄存器来定义处理器2为目的处理器。

bit	Access	Name	寄存器描述	配置值
MBX_DSET: 0x004 + i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	reserved	0x0
[5]	WO	dest	预留处理器bit位，软件固定配置为0	0x0
[4:0]	WO	dset	邮箱通道目的处理器配置寄存器，即通信中断的接收处理器设置，可配置多个处理器同时接收中断，每一bit对应一个处理器。 0：无影响； 1：设置为目的处理器。 bit[3]：ARM_4。 bit[2]：ARM_3； bit[1]：ARM_1； bit[0]：ARM_0； …	0x4

步骤5 处理器0、1根据每个处理器需要的写入消息数据到Mailbox4数据寄存器MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）。

说明

如果之前每个源处理器没有使用同一个Mailbox，那么此时需要注意相应处理器的消息数据写入到对应Mailbox数据寄存器中，同时Hi309a Mailbox包含MBX_DATA0~7共个数据寄存器。

步骤6 处理器0置位发送寄存器的bit0来使Mailbox4发中断到处理器2；处理器1置位发送寄存器的bit1来使Mailbox4发中断到处理器2。

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
MBX_SEND: 0x01C + i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	reserved	0x0
[5]	WO	send	预留处理器bit位，软件固定配置为0	0x0
[4:0]	WO	send	中断发送寄存器，每一bit对应一个处理器。通信过程中，配置此寄存器，只置位bit有效。 当源发中断给目的时，send寄存器配置值与源寄存器值相等时发中断给目的处理器；当目的发应答中断给源时，send寄存器配置值与目的相等时发应答中断给源。自动链接时，被链接邮箱此寄存器不须配置；自动应答时，目的清除中断后，此寄存器自动置位，不须配置。 bit[3]：ARM_4； bit[2]：ARM_3； bit[1]：ARM_1； bit[0]：ARM_0。 ...	各处理器置位对应位

说明

每个处理器自己独立配置各自的发送寄存器选择什么时候发送，只有每个处理器都完成发送后，中断才会送出。

步骤7 处理器2收到中断后，读取对应的中断屏蔽后状态寄存器CPU_IMST（0x800+j*8，j为处理器编号）检测发起中断的Malibox。

步骤8 处理器2从Mailbox4中MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）读消息数据。

步骤9 处理器2可选择地更新Mailbox4中的应答消息。

步骤10 处理器2清除对应Mailbox4的中断状态清除寄存器MBX_ICLR（0x018+i×0x40，i为Mailbox编号）中的bit位来清除中断。

bit	Access	Name	寄存器描述	配置值
MBX_ICLR: 0x018+i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	RWC	int_clear	预留处理器bit位，软件固定配置为0	0x0

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
[4:0]	RWC	int_clear	邮箱通道到处理器屏蔽后中断状态，每一bit对应一个处理器，写1清除。 0：无中断； 1：有中断。	各处理器置位对应位

步骤11 置位发送寄存器中bit位产生应答中断给处理器0、1。

bit	Access	Name	寄存器描述	配置值
MBX_SEND: 0x01C + i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	WO	send	预留处理器bit位，软件固定配置为0	0x0
[4:0]	WO	send	中断发送寄存器，每一bit对应一个处理器。通信过程中，配置此寄存器，只置位bit有效。 当源发中断给目的时，send寄存器配置值与源寄存器值相等时发中断给目的处理器；当目的发应答中断给源时，send寄存器配置值与目的相等时发应答中断给源。自动链接时，被链接邮箱此寄存器不须配置；自动应答时，目的清除中断后，此寄存器自动置位，不须配置。	各处理器置位对应位

步骤12 处理器0、1收到中断后，读中断屏蔽后状态寄存器CPU_IMST（0x800+j*8，j为处理器编号）检测发起中断的Mailbox。

步骤13 处理器0、1读Mailbox4中的应答消息数据。

步骤14 处理器0、1清除对应Mailbox4的中断状态清除寄存器MBX_ICLR（0x018+i*64，i为Mailbox编号）中的bit位来清除中断。

步骤15 处理器0通过清除源寄存器来释放对Mailbox4的占用。

bit	Access	Name	寄存器描述	配置值
MBX_SOURCE: i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	RW	source	预留处理器bit位，软件固定配置为0	0x0

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
[4:0]	RW	source	邮箱通道占用源处理器配置，可配置多个处理器共同占用此邮箱通道，每一bit对应一个占用通道的处理器； 配置后写相同值释放该通道邮箱，该邮箱除数据寄存器、模式寄存器之外的寄存器均清0，配置后写其他值无意义。	0x3

说明

处理器0也可以不清除源处理器寄存器，保持对Mailbox的占用，从而可以继续发送消息到其他目的处理器。

----结束

28.3.2 多个目的的处理器通信

此例的工作过程如下：

步骤1 处理器0获通过写源处理器申请获得Mailbox4的控制，并且通过设置源寄存器将CORE0设置为源处理器，从而实现1到多和多到多的应用场景。

说明

处理器0在此用例中作为主控处理器，可以通过一个Mailbox完成多到多的通信，也可以申请多个Mailbox完成多到多的通信。

步骤2 处理器0回读源寄存器，确认配置正确。

步骤3 处理器0通过配置中断屏蔽寄存器的相应的bit0到bit4位，不屏蔽处理器0~4的中断。

步骤4 处理器0通过配置目的寄存器来定义处理器1、2为目的处理器。

步骤5 处理器0根据每个处理器需要的写入消息数据到Mailbox4数据寄存器。

说明

如果之前每个源处理器没有使用同一个Mailbox，那么此时需要注意相应处理器的消息数据写入到对应Mailbox数据寄存器中，同时每个Mailbox的数据寄存器由宏定义MBX_DATA0~7可配。

步骤6 处理器0置位发送寄存器的bit0来使Mailbox发中断到处理器1、2。

说明

每个处理器自己独立配置各自的发送寄存器选择什么时候发送，只有每个处理器都完成发送后，中断才会送出。

步骤7 处理器1、2收到中断后，读取对应的中断屏蔽后状态寄存器检测发起中断的Malibox。

步骤8 处理器1、2读消息数据。

步骤9 处理器1、2可选择地更新应答消息。

版权所有 © 海思技术有限公司

步骤10 处理器1、2清除对应Mailbox4的中断状态清除寄存器中的bit位来清除中断，以及置位发送寄存器中bit位产生应答中断给处理器0。

步骤11 处理器0收到中断后，读中断屏蔽后状态寄存器检测发起中断的Mailbox。

步骤12 处理器0读应答消息数据。

步骤13 处理器0清除对应Mailbox4的中断状态清除寄存器中的bit位来清除中断。

步骤14 处理器0通过清除源寄存器来释放对Mailbox4的占用，同时清除Mailbox4的其他寄存器信息。

说明

处理器0也可以不清除源处理器寄存器，保持对Mailbox的占用，从而可以继续发送消息到其他目的处理器。

----结束

28.3.3 自动应答模式

此例的工作过程如下：

步骤1 处理器0获通过写源处理器申请获得Mailbox4的控制，并且通过设置源寄存器将CORE0设置为源处理器。

bit	Access	Name	寄存器描述	配置值
MBX_SOURCE: i×0x40，i为Mailbox编号，取值范围0~7				-
[31:6]	-	reserved	保留。	0x0
[5]	RW	source	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RW	source	邮箱通道占用源处理器配置，可配置多个处理器共同占用此邮箱通道，每一bit对应一个占用通道的处理器； 配置后写相同值释放该通道邮箱，该邮箱除数据寄存器、模式寄存器之外的寄存器均清0，配置后写其他值无意义。	0x1

步骤2 处理器0回读源寄存器，确认配置正确。

步骤3 处理器0通过配置中断屏蔽寄存器的相应的bit0到bit4位，不屏蔽处理器0~4的中断。

bit	Access	Name	寄存器描述	配置值
MBX_IMASK: 0x014＋i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
[5]	RW	int_mask	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RW	int_mask	邮箱通道到处理器的中断屏蔽设置，每一bit对应一个处理器。 0：不屏蔽； 1：屏蔽。 位宽与宏定义IPC_CPU_NUM一致。	0x0

步骤4 处理器0通过配置目的寄存器来定义处理器1为目的处理器。

bit	Access	Name	寄存器描述	配置值
MBX_DSET: 0x004 + i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	WO	dest	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	WO	dset	邮箱通道目的处理器配置寄存器，即通信中断的接收处理器设置，可配置多个处理器同时接收中断，每一bit对应一个处理器。 0：无影响； 1：设置为目的处理器。	0x2

步骤5 处理器0通过配置Mailbox4的通道模式状态机状态寄存器bit 0位为1，启动自动应答模式。

bit	Access	Name	寄存器描述	配置值
MBX_MODE: 0x010 + i×0x40，i为Mailbox编号				-
[31:8]	-	reserved	保留。	0x0

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
7:4	RO	state_status	状态机状态； 0x1：空闲状态，邮箱空闲； 0x2：源状态，此状态下，邮箱通道已占用，还未发中断给目的处理器； 0x4：目的状态，此状态下，目的处理器收到源发中断，未全部清除或返回应答中断； 0x8：应答状态，此状态下，源收到应答中断未清除完毕； 其他：异常值。	0x0
3:2	RO	reserved	保留。	0x0
1	RW	auto_link	Mailbox自动链接使能。 自动链接，将几个临近的Mailboxes链接在一起，从而可以将多个消息连续的发送出去，而不需要在消息与消息之间发送中断给源处理器。当所有信息都已经传递完毕，应答中断可以由一个目的处理器配置产生，也可以通过自动应答模式来产生。链的顺序是固定的，Mailbox 0 链接到 Mailbox 1 ，依次最高到最后一个邮箱，最后一个邮箱的自动链接功能配置无效。 0：禁止； 1：使能。	0x0
0	RW	auto_answer	Mailbox自动应答使能。 自动应答，即当目的处理器中断被清除0后，mailbox自动发出一个回复中断给源处理器。 0：禁止； 1：使能。	0x1

步骤6 处理器0根据每个处理器需要的写入消息数据到Mailbox4数据寄存器MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）。

步骤7 处理器0置位发送寄存器的bit0来使Mailbox4发中断到处理器1。

bit	Access	Name	寄存器描述	配置值
MBX_SEND: 0x01C＋i×0x40，i为Mailbox编号				-

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
[31:6]	-	reserved	保留。	0x0
[5]	WO	send	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	WO	send	中断发送寄存器，每一bit对应一个处理器。通信过程中，配置此寄存器，只置位bit有效。 当源发中断给目的时，send寄存器配置值与源寄存器值相等时发中断给目的处理器；当目的发应答中断给源时，send寄存器配置值与目的相等时发应答中断给源。自动链接时，被链接邮箱此寄存器不须配置；自动应答时，目的清除中断后，此寄存器自动置位，不须配置。	各处理器置位对应位

步骤8 处理器1收到中断后，读取对应的中断屏蔽后状态寄存器CPU_IMST（0x800+j*8，j为处理器编号）检测发起中断的Malibox。

步骤9 处理器1读消息数据MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）。

步骤10 处理器1可选择地更新应答消息。

步骤11 处理器1清除对应Mailbox4的中断状态清除寄存器MBX_ICLR（0x018+i*64，i为Mailbox编号）中的bit位来清除中断。

bit	Access	Name	寄存器描述	配置值
MBX_ICLR: 0x018+i*64，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	RWC	int_clear	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RWC	int_clear	邮箱通道到处理器屏蔽后中断状态，每一bit对应一个处理器，写1清除。 0：无中断； 1：有中断。 位宽与宏定义IPC_CPU_NUM一致。	各处理器置位对应位

步骤12 处理器0收到中断后，读中断屏蔽后状态寄存器CPU_IMST（0x800+j*8，j为处理器编号）检测发起中断的Mailbox，此时自动应答使能，不需要配置send寄存器回应答中断。

步骤13 处理器0读应答消息数据MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）。

版权所有 © 海思技术有限公司

步骤14 处理器0清除对应Mailbox4的中断状态清除寄存器中的bit位来清除中断。

bit	Access	Name	寄存器描述	配置值
MBX_ICLR: 0x018+i*64，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	RWC	int_clear	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RWC	int_clear	邮箱通道到处理器屏蔽后中断状态，每一bit对应一个处理器，写1清除。 0：无中断； 1：有中断。 位宽与宏定义IPC_CPU_NUM一致。	各处理器置位对应位

步骤15 处理器0通过清除源寄存器来释放对Mailbox4的占用，同时清除Mailbox4的其他寄存器信息。

bit	Access	Name	寄存器描述	配置值
MBX_SOURCE: i×0x40，i为Mailbox编号，取值范围0~7				-
[31:6]	-	reserved	保留。	0x0
[5]	RW	source	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RW	source	邮箱通道占用源处理器配置，可配置多个处理器共同占用此邮箱通道，每一bit对应一个占用通道的处理器； 配置后写相同值释放该通道邮箱，该邮箱除数据寄存器、模式寄存器之外的寄存器均清0，配置后写其他值无意义。 位宽与宏定义IPC_CPU_NUM一致。	0x0

说明

处理器0也可以不清除源处理器寄存器，保持对Mailbox的占用，从而可以继续发送消息到其他目的处理器。

----结束

版权所有 © 海思技术有限公司

28.3.4 自动链接模式

此例的工作过程如下：

步骤1 处理器0获通过写源处理器使处理器0申请获得Mailbox4的控制，并且通过设置源寄存器将CORE0为源处理器。

bit	Access	Name	寄存器描述	配置值
MBX_SOURCE: i×0x40，i为Mailbox编号，取值范围0~7				-
[31:6]	-	reserved	保留。	0x0
[5]	RW	source	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RW	source	邮箱通道占用源处理器配置，可配置多个处理器共同占用此邮箱通道，每一bit对应一个占用通道的处理器； 配置后写相同值释放该通道邮箱，该邮箱除数据寄存器、模式寄存器之外的寄存器均清0，配置后写其他值无意义。	0x1

步骤2 处理器0回读Mailbox4的源寄存器，确认使处理器0获得Mailbox4的控制。

步骤3 处理器0获通过写源处理器申请处理器0获得Mailbox5的控制，并且通过设置源寄存器将CORE0为源处理器。

bit	Access	Name	寄存器描述	配置值
MBX_SOURCE: i×0x40，i为Mailbox编号，取值范围0~7				-
[31:5]	-	reserved	保留。	0x0
[5]	RW	source	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RW	source	邮箱通道占用源处理器配置，可配置多个处理器共同占用此邮箱通道，每一bit对应一个占用通道的处理器； 配置后写相同值释放该通道邮箱，该邮箱除数据寄存器、模式寄存器之外的寄存器均清0，配置后写其他值无意义。	0x1

步骤4 处理器0回读Mailbox5的源寄存器，确认处理器0获得Mailbox5的控制。

步骤5 处理器0通过配置中断屏蔽寄存器的相应的bit0到bit4位，不屏蔽处理器0~4的中断。

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
MBX_IMASK: 0x014 + i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	RW	int_mask	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RW	int_mask	邮箱通道到处理器的中断屏蔽设置，每一bit对应一个处理器。 0：不屏蔽； 1：屏蔽。	0x0

步骤6 处理器0通过配置Mailbox4目的寄存器来定义处理器1为目的处理器，Mailbox5目的寄存器来定义处理器2为目的处理器。

bit	Access	Name	寄存器描述	配置值
MBX_DSET: 0x004 + i×0x40，i为Mailbox编号，此处为4				-
[31:5]	-	reserved	保留。	0x0
[5]	WO	dset	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	WO	dset	邮箱通道目的处理器配置寄存器，即通信中断的接收处理器设置，可配置多个处理器同时接收中断，每一bit对应一个处理器。 0：无影响； 1：设置为目的处理器。	0x2

bit	Access	Name	寄存器描述	配置值
MBX_DSET: 0x004 + i×0x40，i为Mailbox编号，此处为5				-
[31:5]	-	reserved	保留。	0x0
[5]	WO	dset	预留处理器bit位，软件固定配置为0	0x0
[4:0]	WO	dset	邮箱通道目的处理器配置寄存器，即通信中断的接收处理器设置，可配置多个处理器同时接收中断，每一bit对应一个处理器。 0：无影响； 1：设置为目的处理器。	0x4

版权所有 © 海思技术有限公司

步骤7 处理器0通过配置邮箱通道模式状态机状态寄存器的bit 1位为1，分别将Mailbox4、5链接起来；

bit	Access	Name	寄存器描述	配置值
MBX_MODE: 0x010 + i×0x40，i为Mailbox编号				-
[31:8]	-	reserved	保留。	0x0
[7:4]	RO	state_status	状态机状态； 0x1：空闲状态，邮箱空闲； 0x2：源状态，此状态下，邮箱通道已占用，还未发中断给目的处理器； 0x4：目的状态，此状态下，目的处理器收到源发中断，未全部清除或返回应答中断； 0x8：应答状态，此状态下，源收到应答中断未清除完毕； 其他：异常值。	0x0
[3:2]	RO	reserved	保留。	0x0
[1]	RW	auto_link	Mailbox自动链接使能。 自动链接，将几个临近的Mailboxes链接在一起，从而可以将多个消息连续的发送出去，而不需要在消息与消息之间发送中断给源处理器。当所有信息都已经传递完毕，应答中断可以由一个目的处理器配置产生，也可以通过自动应答模式来产生。链的顺序是固定的，Mailbox 0 链接到 Mailbox 1 ，依次最高到最后一个邮箱，最后一个邮箱的自动链接功能配置无效。 0：禁止； 1：使能。	0x1
[0]	RW	auto_answer	Mailbox自动应答使能。 自动应答，即当目的处理器中断被清除0后，mailbox自动发出一个回复中断给源处理器。 0：禁止； 1：使能。	0x0

步骤8 处理器0需要的写入消息数据到Mailbox4、5数据寄存器MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）。

步骤9 处理器0置位发送寄存器的bit0来使Mailbox4发中断到处理器1。

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
MBX_SEND: 0x01C + i×0x40，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	WO	send	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	WO	send	中断发送寄存器，每一bit对应一个处理器。通信过程中，配置此寄存器，只置位bit有效。 当源发中断给目的时，send寄存器配置值与源寄存器值相等时发中断给目的处理器；当目的发应答中断给源时，send寄存器配置值与目的相等时发应答中断给源。自动链接时，被链接邮箱此寄存器不须配置；自动应答时，目的清除中断后，此寄存器自动置位，不须配置。	各处理器置位对应位

说明

自动链接模式下，只需要配置首个Mailbox的发送寄存器。

步骤10 处理器1收到中断后，遍历中断屏蔽后状态寄存器检测发起中断的Malibox。

bit	Access	Name	寄存器描述	配置值
MBX_ICLR: 0x018+i*64，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	RWC	int_clear	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RWC	int_clear	邮箱通道到处理器屏蔽后中断状态，每一bit对应一个处理器，写1清除。 0：无中断； 1：有中断。	各处理器置位对应位

步骤11 处理器1读消息数据MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）。

步骤12 处理器1可选择地更新应答消息。

步骤13 处理器1清除对应Mailbox4的中断状态清除寄存器中的bit位来清除中断，以及置位发送寄存器中bit位产生应答给处理器0。

版权所有 © 海思技术有限公司

说明

自动链接模式下，中间过程中不产生应答中断。

步骤14 自动链接到Mailbox5，处理器2收到中断后，遍历中断屏蔽后状态寄存器检测发起中断的Malibox。

步骤15 处理器2读消息数据。

步骤16 处理器2可选择地更新应答消息。

步骤17 处理器2清除对应Mailbox5的中断状态清除寄存器中的bit位来清除中断，以及置位发送寄存器中bit位产生应答给处理器0。

步骤18 处理器0收到中断后，读中断屏蔽后状态寄存器CPU_IMST（0x800+j*8，j为处理器编号）检测发起中断的Mailbox，此时自动应答使能，不需要配置send寄存器回应答中断。

步骤19 处理器0读应答消息数据MBX_DATA0～MBX_DATA7（0x020+i*64～0x03C+i*64，i为Mailbox编号）。

步骤20 处理器0清除对应Mailbox4的中断状态清除寄存器中的bit位来清除中断。

bit	Access	Name	寄存器描述	配置值
MBX_ICLR: 0x018+i*64，i为Mailbox编号				-
[31:6]	-	reserved	保留。	0x0
[5]	RWC	int_clear	预留处理器bit位，软件固定配置为0。	0x0
[4:0]	RWC	int_clear	邮箱通道到处理器屏蔽后中断状态，每一bit对应一个处理器，写1清除。 0：无中断； 1：有中断。	各处理器置位对应位。

步骤21 处理器0通过清除源寄存器来释放对Mailbox4的占用，同时清除Mailbox4的其他寄存器信息。

bit	Access	Name	寄存器描述	配置值
MBX_SOURCE: i×0x40，i为Mailbox编号，取值范围0~7				-
[31:6]	-	reserved	保留。	0x0
[5]	RW	source	预留处理器bit位，软件固定配置为0。	0x0

版权所有 © 海思技术有限公司

bit	Access	Name	寄存器描述	配置值
[4:0]	RW	source	邮箱通道占用源处理器配置，可配置多个处理器共同占用此邮箱通道，每一bit对应一个占用通道的处理器； 配置后写相同值释放该通道邮箱，该邮箱除数据寄存器、模式寄存器之外的寄存器均清0，配置后写其他值无意义。	0x0

说明

处理器0也可以不清除源处理器寄存器，保持对Mailbox的占用，从而可以继续发送消息到其他目的处理器。

----结束

28.3.5 IPC 软复位配置步骤

步骤1 软件保证IPC已经空闲，并且复位过程中不再使用IPC；

步骤2 对于IPC，配置SOFT_RST0_CTL（0x44B6_2000+0x0044）的第15bit为0b1（其他位使用回读值或者0x0）,再配置为0b0（其他位使用回读值或者0x0），可实现对IPC的单独软复位。

----结束

须知

软复位功能仅供调试，禁止软件使用。

28.4 配置约束

Hi309a仅包含了5个处理器（ARM0~ARM3、UVP），因此源寄存器、中断屏蔽寄存器、目的寄存器、发送寄存器均只支持bit4~bit0可配，其余bit须配置为0。

28.5 应用说明

无。

28.6 可维可测

无。

版权所有 © 海思技术有限公司

29 SEM

29.1 概述

SEMAPHORE用于多核共享资源的调用，Hi309a SEMAPHORE支持32个信号量。
Hi309a中针对SEMAPHORE，CPU的编号为：

ARM_0：0b000；

ARM_1：0b001；

ARM_2：0b010；

ARM_3：0b011；

ARM_4：0b100；

ARM_5：0b101；

ARM_6：0b110；

ARM_7：0b111。

29.2 依赖关系

包含1个APB配置口用于内部寄存器的配置和状态上报；

输出8bit中断信号送给ARM CPU，SEMAPHORE模块送给ARM CPU分配的中断编号如表29-1所示，详细信息请参见《Hi309a 中断向量表》。

表 29-1 SEM 中断映射表

序号	中断编号	中断源	描述	输入源
1	202~209	INT_SEM_ABN	SEM的异常中断，每个核一个。	SEMAPHORE
2	210~217	INT_SEM_NOR	SEM的正常中断，每个核一个。	SEMAPHORE

版权所有 © 海思技术有限公司

29.3 操作步骤

SEMAPHORE支持查询模式、队列模式和组合模式，推荐使用队列模式。

29.3.1 查询模式

信号量获取使用查询工作模式时，配置流程如下：

步骤1 CPU通过读信号量的SEM_DIRECT寄存器申请获取对应信号量资源。

步骤2 若申请时信号量空闲且模式正确，读寄存器返回值为1，CPU直接获取到信号量资源。

步骤3 若申请时信号量忙，读寄存器返回值为0，继续读取Direct寄存器，直到前一CPU释放信号量后获取到信号量。

步骤4 操作申请信号量对应的共享资源。

步骤5 通过写1到SEM_DIRECT寄存器释放信号量。

----结束

29.3.2 队列模式

信号量获取使用队列工作模式时，

步骤1 配置SEM_CONFIG使能队列模式和中断使能。

步骤2 CPU通过写{cpu id，1}到信号量的SEM_INDIRECT寄存器申请获取信号量资源。

步骤3 若申请时信号量空闲，模块直接报中断给CPU，通过CPU_GRANTL中断状态寄存器查询成功获取到那个信号量。

步骤4 若申请时信号量忙，此CPU的申请操作记录到队列中，队列中此CPU前面的信号量申请释放后，再上报中断给此CPU，通过CPU_GRANTL中断状态寄存器查询成功获取到那个信号量。

步骤5 收到信号量获取中断后，配置SEM_GRANT_EOI清除中断，并操作申请信号量对应的共享资源。

步骤6 通过写正确的CPU id和0到SEM_INDIRECT寄存器释放信号量。

----结束

29.3.3 组合模式

信号量获取使用组合工作模式时：

步骤1 配置SEM_CONFIG使能组合模式和中断使能。

步骤2 CPU通过写CPU id和1到信号量的SEM_COMBINED寄存器申请获取信号量资源。

步骤3 若申请时信号量空闲，类似查询模式的处理方式，读SEM_COMBINED寄存器返回模式下未通过中断获取到信号量的CPU id和信号量空闲状态。

步骤4 若申请信号量忙，类似队列模式的处理方式，将此CPU的申请操作记录到队列中，队列中此CPU前面的信号量申请释放后，再上报中断给此CPU，通过CPU_GRANTL、中

版权所有 © 海思技术有限公司

断状态寄存器查询成功获取到那个信号量；收到信号量获取中断后，配置SEM_GRANT_EOI清除中断。

步骤5 信号量获取成功后，操作申请信号量对应的共享资源。

步骤6 通过写CPU id和0到SEM_COMBINED寄存器释放信号量。

----结束

29.3.4 SEMAPHORE 软复位配置步骤

软件保证SEMAPHORE已经空闲，并且复位过程中不再使用Semaphore；

步骤1 对于SEMAPHORE，配置SOFT_RST1_CTL（0x44B62000+0x0048）的第10bit为0b1（其他位使用回读值或者0x0），再配置为0b0（其他位使用回读值或者0x0），可实现对SEMAPHORE的单独软复位。

----结束

须知

软复位功能仅供调试，禁止软件使用。

29.4 配置约束

无。

29.5 应用说明

无。

29.6 可维可测

在信号量的申请释放出现错误，且错误中断使能情况下，信号量会报错误中断给CPU，CPU通过查询其相应的信号量错误中断状态寄存器（CPU_ERR_STATUS），确认出现错误的信号量id及具体的错误类型，错误类型分类如表29-2所示。

表 29-2 错误类型说明

错误类型	错误说明	备注
err_type1	队列模式和组合模式下，信号量获取申请已在队列中重复申请。	错误中断上报给进行错误操作的CPU。
err_type2	队列模式和组合模式下，信号量已占用重复申请。	错误中断上报给进行错误操作的CPU。
err_type3	队列模式和组合模式下，释放信号量的CPU id与占用CPU id不一致。	错误中断上报给进行错误操作的CPU。

版权所有 © 海思技术有限公司

错误类型	错误说明	备注
err_type4	队列模式和组合模式下，申请获取和释放信号量的CPU id超出CPU个数范围。	上报错误中断给所有CPU。
err_type5	非查询模式下读Direct寄存器。	上报错误中断给所有CPU。
err_type6	非队列模式下写InDirect寄存器。	错误中断上报给进行错误操作的CPU。
err_type7	非组合模式下写Combined寄存器。	错误中断上报给进行错误操作的CPU。

实际应用中，在方案软件调试阶段，有可能会出现错误中断；在产品化方案软件代码中，无芯片异常情况下，错误中断不会产生。

说明

有错误中断且未清除情况下，新的信号量操作错误中断及错误类型无法上报，直到错误中断清除后，新的错误操作中断和类型才会更新。

版权所有 © 海思技术有限公司