



Hi3093 通用 MPU 芯片

用户指南

文档版本 00B01

发布日期 2023-11-30

版权所有 © 海思技术有限公司 2023。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：<http://www.hisilicon.com/cn/>

客户服务邮箱：support@hisilicon.com



前言

概述

本文档针对 Hi3093 的各类业务，从预备知识、信息采集方法、通用处理流程、常见故障处理方法和故障案例分析介绍了故障的处理过程。

本文档提供了 Hi3093 设备故障的处理流程和方法。

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 维护工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
 须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其他不可预知的结果。



符号	说明
	“须知”不涉及人身伤害。
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

修改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
2023-11-30	00B01	第 1 次临时版本发布。



目 录

前 言 i

目 录 iii

插图目录 xxxix

表格目录 l

手册使用约束 lx

1 芯片简介 1

 1.1 概述 1

 1.2 功能特性简述 2

 1.2.1 处理器核 2

 1.2.2 DDR 接口 3

 1.2.3 外部接口 3

 1.2.4 加速器功能 4

 1.2.5 显示功能 4

 1.2.6 安全功能 4

 1.2.7 其他接口 4

 1.3 硬件特性简述 5

 1.3.1 参考时钟 5

 1.3.2 封装尺寸 5

2 ARM-M3 6

 2.1 概述 6

 2.2 关键术语 7

 2.3 特性 7



2.4 功能	8
2.4.1 Cortex M3 内核	8
2.4.2 总线矩阵和接口	8
2.4.3 中断 NVIC	8
2.4.4 MPU	9
2.4.5 调试端口	9
2.4.6 WatchDog	10
2.4.7 核间通信	10
2.5 工作方式	10
2.5.1 安全启动	10
2.5.2 协处理器 M3 地址空间列表	11
.....	17
3 ARM-A55	18
3.1 概述	18
3.2 特性	18
3.3 功能描述	19
3.3.1 FCM 结构	19
3.3.2 Memory 接口	20
3.3.3 GIC CPU 接口	20
3.3.4 系统计数器	21
3.3.5 通用计时器	21
3.4 工作方式	22
3.4.1 启动	22
3.4.2 A55 地址空间列表	22
3.4.3 工作状态	22
3.4.4 异常等级	22
4 GIC	24
4.1 概述	24
4.2 特性	24
4.3 功能描述	25
4.3.1 关键术语	25
4.3.2 应用场景	26



4.3.3 中断流程.....	28
4.3.4 GICD 亲和路由	30
4.3.4.1 1/N SPI 亲和路由	31
4.3.4.2 SGI 亲和路由	32
4.3.5 中断触发方式	33
4.3.6 中断优先级	33
4.3.7 中断嵌套	34
4.3.8 中断唤醒核	34
4.3.9 核休眠	35
4.3.10 初始化	36
4.3.11 GICD 中断路由配置流程	37
4.3.11.1 亲和路由模式	37
4.3.11.2 改变 SPI 亲和路由设置流程	37
5 系统控制器	39
5.1 概述	39
5.2 特性	39
5.3 功能描述	40
5.3.1 系统运行模式	40
5.3.2 子系统外设控制	41
5.3.2.1 A55 子系统控制	41
5.3.2.2 DDR 子系统控制	41
5.3.2.3 高速外设 (HP) 子系统控制	42
5.3.2.4 低速外设 (LP) 子系统控制	42
5.3.2.5 HOST 子系统控制	43
5.4 工作方式	43
5.4.1 启动 PAD	43
5.4.2 启动真值表	43
5.4.3 系统启动	44
5.4.4 双 BIOS 启动功能描述	46
5.4.5 启动地址映射说明	47
5.4.6 软复位控制	49



5.4.7 其他功能.....	49
6 时钟.....	50
6.1 概述	50
6.2 特性	50
6.3 功能描述	51
6.3.1 时钟架构.....	51
6.3.2 TOP_CRG 描述	52
6.3.2.1 PLL 说明	53
6.3.2.2 PLL 配置	55
7 复位.....	60
7.1 概述	60
7.2 特性	60
7.3 功能描述	61
7.3.1 复位接口信号	61
7.3.2 复位种类.....	62
7.3.2.1 外部管脚硬复位.....	62
7.3.2.2 内部软复位	62
7.4 工作方式	63
7.4.1 整芯片复位	63
7.4.1.1 系统上电复位	63
7.4.1.2 整芯片热复位	64
7.4.2 协处理模式域单独复位.....	66
7.4.2.1 管脚触发协处理模式域复位.....	67
7.4.2.2 软件触发协处理模式域复位.....	68
7.4.2.3 WatchDog 触发协处理模式域复位	69
7.4.3 协处理模式非安全域单独复位	71
7.4.3.1 软件触发协处理模式非安全域复位.....	71
7.4.3.2 WatchDog 触发协处理模式非安全域复位	72
7.4.4 HOST 域单独复位.....	74
7.4.4.1 管脚触发 host 域单独复位.....	74



7.4.4.2 软件触发 host 域单独复位	75
7.4.4.3 PCIE0 相关模块单独复位	76
7.4.4.4 PCIE1 单独复位	77
7.4.5 模块软件复位	77
7.5 寄存器特殊复位	78
7.5.1 DMAC	78
7.5.2 SYSCtrl	79
7.5.3 SEC_CSR	80
7.5.4 PCIE	80
8 DDRC	84
8.1 概述	84
8.2 特性	84
8.3 关键术语	86
8.4 遵循协议	86
8.5 功能描述	86
8.5.1 相关模块	86
8.5.2 DDRC 支持的器件容量空间以及地址映射	87
8.5.2.1 DDRC 支持的器件容量	87
8.5.2.2 DDR4 独立通道模式下地址映射	87
8.5.2.3 DDRC 命令调度策略	89
8.5.2.4 DDR Memory ECC 保护	94
8.5.2.5 ECC 旁路访问	95
8.5.2.6 带内 ECC 规格	95
8.5.2.7 Memory BIST	95
8.5.2.8 Memory 内容初始化	96
8.5.2.9 DDR 自动刷新	96
8.5.2.10 DDR 自刷新功能	97
8.5.2.11 DDR Power Down 功能	97
8.5.2.12 DDR ZQ 自校准功能	98
8.5.2.13 DDRC 流量 Latency 统计功能	98
8.5.2.14 DDRPHY 异常检测功能	99
8.5.2.15 Address Parity 功能	99



8.5.2.16 Data Scrambling 功能	100
8.5.2.17 DDR 热复位数据保持功能	101
8.6 DFX	102
8.6.1 可靠性	102
8.6.1.1 自恢复	102
8.6.1.2 非自恢复	102
8.6.2 故障预防	103
8.6.3 模块维测	103
8.6.3.1 统计	103
8.6.3.2 Debug	104
9 LocalBus	105
9.1 概述	105
9.2 特性	105
9.3 功能描述	106
9.3.1 功能原理	106
9.3.1.1 外部 Ready 模式以及 Ready 超时保护	106
9.3.1.2 在外部 Ready 模式下支持防挂死特性	106
9.3.2 数据和地址线序	107
9.3.2.1 数据字节顺序	107
9.3.2.2 地址偏移	108
9.3.2.3 上电管脚采样	109
9.3.2.4 LocalBus 写保护功能	109
9.3.3 灵活的时序配置	110
9.3.3.1 LocalBus 时序参数配置	110
9.3.3.2 LocalBus 典型时序配置	112
9.4 工作方式	118
9.4.1 时钟门控	118
9.4.2 软复位	118
9.4.3 初始化	119
9.4.4 基地址配置	119
9.4.5 固定时序模式	120
9.4.6 异步等待模式	120



9.4.7 Page 读模式.....	121
9.4.8 写保护功能.....	121
9.4.8.1 普通写保护功能.....	121
9.4.8.2 高级写保护功能.....	122
9.4.9 数据读写方式	122
9.4.10 外部 Ready 模式	123
10 SFC.....	124
10.1 概述	124
10.2 特性	124
10.3 功能描述	125
10.3.1 Boot 模式	125
10.3.2 读写 Memory.....	125
10.3.2.1 读 Memory.....	125
10.3.2.2 写 Memory.....	125
10.4 工作方式	126
10.4.1 管脚复用配置	126
10.4.2 初始化	126
10.4.3 时钟门控	127
10.4.4 软复位	128
10.4.5 通过寄存器读 Flash（查询方式）	129
10.4.6 通过寄存器写 Flash（中断方式）	130
10.4.7 通过 DMA 方式读写 Flash 操作流程	130
10.4.8 通过寄存器方式擦除 FLASH.....	131
11 SMBUS	133
11.1 概述.....	133
11.2 特性.....	133
11.2.1 关键术语.....	134
11.3 功能描述	135
11.3.1 典型应用.....	135
11.4 工作方式	135
11.4.1 管脚复用配置.....	135



11.4.2 I2C 和 SMBUS 模式选择	136
11.4.3 时钟门控	136
11.4.4 软复位	136
11.4.5 TimeOut 超时处理	136
11.4.5.1 SMBUS 超时参数含义	137
11.4.5.2 SMBUS 超时处理流程	138
11.4.6 主动中断 I2C 传输	139
11.4.7 时钟配置	139
11.4.8 中断或查询方式下的数据传输	140
11.4.8.1 初始化	140
11.4.8.2 数据发送	141
11.4.8.3 数据接收	141
11.4.9 SMBUS 协议的支持	142
11.4.9.1 SMBUS 协议命令	142
11.4.9.2 SMBUS PEC	143
11.4.9.3 SMBUS ARP 命令支持	144
11.4.9.4 ARP 的配置流程	145
11.4.10 相关寄存器基址	147
11.4.11 状态和记录	148
11.4.12 内部关键信号	149
12 UART	150
12.1 概述	150
12.2 特性	151
12.3 功能描述	153
12.3.1 典型应用	153
12.3.1.1 UART 3 线模式	154
12.3.1.2 UART 5 线模式	155
12.3.1.3 UART 7 线模式	155
12.3.1.4 功能原理	156
12.3.1.5 自动流控功能	156
12.3.1.6 发送清除 (/CTS)	157



12.3.1.7 发送请求 (/RTS)	158
12.3.1.8 同步功能	160
12.4 工作方式	161
12.4.1 管脚复用配置	161
12.4.2 UART 全互联	163
12.4.2.1 UART 全互联场景	164
12.4.2.2 全互联的配置方法	165
12.4.3 时钟门控	166
12.4.3.1 打开时钟	167
12.4.3.2 关闭时钟	167
12.4.4 波特率配置	167
12.4.5 中断或查询方式下的数据传输	168
12.4.5.1 初始化	168
12.4.5.2 数据发送	168
12.4.5.3 数据接收	168
12.4.6 DMA 方式下的数据传输	169
12.4.6.1 初始化	169
12.4.6.2 数据发送	169
12.4.6.3 数据接收	169
12.4.7 配置 SOL	170
12.4.8 配置校验位	171
13 SPI	172
13.1 概述	172
13.2 特性	172
13.3 功能描述	173
13.3.1 应用框图	173
13.3.2 内部工作原理	173
13.3.2.1 发送数据	173
13.3.2.2 接收数据	174
13.3.3 SPI 典型时序配置	175
13.3.3.1 MotorolaSPI 帧格式 (spo=0、sph=0)	176



13.3.3.2 MotorolaSPI 帧格式 (spo=0、sph=1)	177
13.3.3.3 MotorolaSPI 帧格式 (spo=1、sph=0)	178
13.3.3.4 MotorolaSPI 帧格式 (spo=1、sph=1)	179
13.3.3.5 TI 同步串行帧格式	180
13.4 工作方式	181
13.4.1 管脚复用配置	181
13.4.2 时钟门控	181
13.4.2.1 打开门控时钟	181
13.4.2.2 关闭门控时钟	181
13.4.3 时钟配置	182
13.4.4 软复位	182
13.4.4.1 SPI0 软复位流程	182
13.4.4.2 SPI1 软复位流程	182
13.4.5 初始化	183
13.4.6 中断或查询方式下的数据传输	183
13.4.6.1 数据发送	184
13.4.6.2 数据接收	185
13.4.7 DMA 方式下的数据传输	185
14 eMMC	186
14.1 概述	186
14.2 特性	186
14.3 关键术语	187
14.4 功能	187
14.4.1 功能原理	187
14.4.1.1 指令与响应	187
14.4.1.2 数据传输	189
14.4.1.3 eMMC 启动 (boot 模式)	192
14.5 工作方式	193
14.5.1 工作时钟门控	193
14.5.2 软复位	194
14.5.3 时钟配置	194



14.5.3.1 接口时钟配置.....	194
14.5.4 初始化	195
14.5.5 非数据传输指令	196
14.5.6 单块或多块读数据	198
14.5.7 单块与多块写数据	200
14.5.8 流数据读写	201
14.5.9 内置 DMA 方式数据传输	201
14.5.9.1 描述符	202
14.5.9.2 初始化	205
14.5.9.3 发送	205
14.5.9.4 接收	206
14.5.10 Auto-stop 使用配置	206
14.5.11 停止数据传输	206
14.5.12 BOOT 模式 (仅 eMMC)	207
14.5.12.1 Original boot.....	207
14.5.12.2 Alternative boot	209
15 SDIO	212
15.1 概述	212
15.2 特性	212
15.3 关键术语	213
15.4 功能	213
15.5 工作方式	213
15.5.1 工作时钟门控	213
15.5.2 软复位	214
15.5.3 时钟配置	214
15.5.3.1 接口时钟配置.....	214
15.5.4 初始化	215
15.5.5 非数据传输指令	216
15.5.6 单块或多块读数据	218
15.5.7 单块与多块写数据	219
15.5.8 流数据读写	221
15.5.9 内置 DMA 方式数据传输	221



15.5.9.1 描述符	221
15.5.9.2 初始化	225
15.5.9.3 发送	225
15.5.9.4 接收	226
15.5.10 Auto-stop 使用配置	226
15.5.11 停止数据传输	226
15.5.12 Suspend 和 Resume 操作	227
15.5.12.1 Suspend 操作步骤	227
15.5.12.2 Resume 操作步骤	228
15.5.13 Readwait 操作.....	229
16 MDIO	230
16.1 概述	230
16.2 特性	230
16.3 遵循协议	230
16.4 功能描述	231
16.4.1 应用框图	231
16.4.2 关键术语	231
16.4.3 数据结构	231
16.4.3.1 Clause22 的数据帧格式	232
16.4.3.2 Clause45 的数据帧格式	233
16.4.4 接口时钟频率配置	234
16.4.5 中断.....	235
16.4.6 Clause22 章和 Clause45 章模式配置	235
16.4.7 PHY 状态自动查询.....	235
16.4.8 初始化	235
16.4.8.1 MDIO 管脚复用配置	235
16.4.8.2 模块初始化	235
16.4.8.3 复位	236
16.4.9 对接 PHY 的配置.....	236
16.4.9.1 Clause22 PHY 寄存器读操作.....	236
16.4.9.2 Clause22 PHY 寄存器写操作.....	237



16.4.9.3 Clause45 PHY 寄存器读数据操作	237
16.4.9.4 Clause45 PHY 寄存器写数据操作	238
16.4.9.5 Clause45 PHY 寄存器地址自增读数据操作	239
17 I2C	241
17.1 概述	241
17.2 关键术语	241
17.3 特性	242
17.4 功能描述	242
17.4.1 典型应用	242
17.4.2 功能原理	243
17.4.2.1 数据帧格式	243
17.4.2.2 复合数据传输格式	244
17.5 工作方式	245
17.5.1 管脚复用配置	245
17.5.2 I2C 和 SMBUS 模式选择	245
17.5.3 时钟门控	246
17.5.3.1 打开门控时钟	246
17.5.3.2 关闭门控时钟	246
17.5.4 时钟配置	247
17.5.5 软复位	248
17.5.5.1 复位操作	248
17.5.5.2 复位撤离	248
17.5.6 中断或查询方式下的数据传输	249
17.5.6.1 初始化	249
17.5.6.2 数据发送	250
17.5.6.3 数据接收	250
17.5.7 DMA 方式下的数据传输	250
17.5.7.1 初始化	250
17.5.7.2 数据发送	251
17.5.7.3 数据接收	251
17.5.8 对端器件挂死的处理	252



17.5.8.1 异常挂死场景.....	252
17.5.8.2 解决方案.....	252
18 GMAC.....	254
18.1 概述	254
18.2 特性	255
18.2.1 链路层处理	255
18.2.2 报文缓存管理	256
18.2.2.1 发送部分	256
18.2.2.2 接收部分	257
18.3 关键术语	259
18.4 功能	259
18.4.1 结构框图	259
18.4.2 接收报文	260
18.4.3 发送报文	261
18.4.4 接收缓存管理	262
18.4.4.1 获取接收缓存地址	262
18.4.4.2 接收报文缓存格式	263
18.4.4.3 接收多种 Tag 生成模式.....	266
18.4.5 发送缓存管理	274
18.4.6 接收报文解析	277
18.4.6.1 输入报文格式.....	277
18.4.6.2 报文协议信息解析	277
18.4.7 超长帧截断.....	283
18.4.7.1 发送方向超长帧截断.....	283
18.4.7.2 接收侧超长帧截断	284
18.4.8 PAD 剥离	285
18.4.8.1 报文内容大小端配置说明	286
18.4.8.2 报文描述符大小端配置说明.....	288
18.4.9 总线一致性操作.....	289
18.4.9.1 Cacheable 操作.....	289
18.4.9.2 Bufferable 操作	291



18.4.10 报文头字节匹配.....	291
18.4.11 流量控制.....	292
18.4.11.1 发送流控帧（全双工模式）	292
18.4.11.2 接收流控帧（全双工模式）	293
18.4.11.3 发送 JAM 帧（半双工模式）	294
18.4.12 端口模式配置	294
18.4.13 SerDes 自协商	297
18.4.14 SGMII 自协商	298
18.4.15 接收报文过滤	298
18.4.15.1 短帧过滤.....	298
18.4.15.2 流控帧过滤	299
18.4.15.3 广播帧过滤	300
18.4.15.4 多播帧过滤	300
18.4.15.5 单播帧过滤	301
18.5 工作方式	302
18.5.1 时钟门控	302
18.5.2 软复位	305
18.5.2.1 复位 GMAC	306
18.5.2.2 网口的各个时钟域的软复位寄存器	309
18.5.3 典型初始化配置流程	312
18.5.3.1 端口模式寄存器初始化配置（模式选择、通道使能）	312
18.5.3.2 报文过滤相关寄存器初始化配置（流量控制、过滤）	315
18.5.3.3 接收发送相关 FIFO 水线值寄存器初始化配置.....	318
18.5.3.4 接收报文控制相关寄存器初始化配置（截断（包括 MAC）、接收缓存大小、申请缓存方式、RX_CTRL）	320
18.5.3.5 接收报文内部总线操作相关寄存器初始化配置（cache 操作、cluster 通道选择） ...	321
18.5.3.6 接收报文解析相关寄存器初始化配置（RX_PKT_MODE、匹配等）	321
18.5.3.7 发送报文控制相关寄存器初始化配置（PAD、CRC 处理、流控帧、释放缓存失败超时、发送 pause 帧控制）	323
18.5.4 接收报文	324
18.5.5 发送报文	325
18.6 DFX	326



18.6.1 PCU 部分	326
18.6.1.1 发送接收总线错误地址查询	326
18.6.1.2 统计计数寄存器	326
18.6.1.3 状态机状态上报	327
18.6.1.4 FIFO 状态查询	327
18.6.1.5 Debug 寄存器	330
18.6.2 MAC 部分	331
18.6.2.1 SGMII 自协商状态查询	331
18.6.2.2 统计计数寄存器	332
18.6.3 环回测试	336
18.6.3.1 应用侧环回	336
18.6.3.2 线路侧环回	337
19 GPIO	339
19.1 概述	339
19.2 特性	339
19.3 关键术语	340
19.4 功能	341
19.4.1 典型应用框图	341
19.5 工作方式	342
19.5.1 时钟门控	342
19.5.1.1 打开门控时钟	342
19.5.1.2 关闭门控时钟	342
19.5.2 管脚复用配置	343
19.5.3 初始复位	343
19.5.4 输入输出配置	343
19.5.5 中断处理	344
19.5.6 毛刺滤除	345
20 PCIe	346
20.1 概述	346
20.2 特性	346
20.3 功能描述	348



20.3.1 应用框图	348
20.3.2 PCIe 系统体系结构	348
20.3.3 PCIe 接口应用框图	350
20.3.4 数据结构	350
20.3.5 PCIe 接口功能	355
20.3.5.1 PCIe 存储器请求事务	355
20.3.5.2 PCIe IO 请求事务	360
20.3.5.3 PCIe 配置请求事务	363
20.3.5.4 PCIe 完成请求事务	367
20.3.5.5 PCIe 消息请求事务	371
20.3.5.6 iATU 地址转换功能	374
20.4 工作方式	376
20.4.1 管脚复用配置	376
20.4.2 初始化	376
20.4.2.1 RC 模式下的初始化操作	376
20.4.2.2 EP 模式下的初始化操作	378
20.4.3 配置读/写操作（类型 0）	379
20.4.4 配置读/写操作（类型 1）	379
20.4.5 RC 模式下的存储器读/写操作（直接访问方式）	380
20.4.6 EP 模式下的存储器读/写操作（直接访问方式）	380
20.4.7 RC 模式下的 IO 读/写操作	380
20.4.8 iATU 功能 Inbound 模式	380
20.4.9 iATU 功能 Outbound 模式	384
20.4.10 DMA 使用初始化	385
20.4.11 DMA 写事务流程实例（非链表）	387
20.4.12 DMA 读事务流程实例（非链表）	390
20.4.13 DMA 写事务流程实例（链表）	392
20.4.14 DMA 读事务流程实例（链表）	397
20.4.15 XDMA HOST 流程	402
20.4.16 PCIe link down 处理流程	402
20.4.17 MSI 中断寄存器配置发送流程	403



20.4.18 MSI 中断接收配置流程	404
20.4.19 PCIe 环回测试	404
20.4.20 Local 环回测试流程	405
20.4.20.1 前提条件	405
20.4.20.2 进入 local PIPE loopback 的初始化流程	405
20.4.20.3 环回下的枚举和 BAR 启动	406
20.4.20.4 退出本地环回	406
20.4.21 Remote 环回测试流程	406
20.4.22 SerDes 初始化	408
20.4.23 软复位	408
20.5 性能	409
20.5.1 吞吐率 (Throughput)	409
20.6 调试 (Debug)	411
20.6.1 关键信号状态寄存器	411
20.6.2 PCIe 专用错误上报机制	411
20.6.3 PCIe 专用状态寄存器查询，链路建立，链路状态寄存器查询	412
20.7 注意事项	412
21 CANBUS	414
21.1 概述	414
21.2 特性	414
21.3 关键术语	415
21.4 功能描述	415
21.4.1 报文内存单元结构	415
21.4.2 CAN 所支持的四种帧格式	416
21.4.2.1 数据帧	416
21.4.2.2 远程帧	417
21.4.2.3 错误帧	418
21.4.2.4 过载帧	418
21.4.2.5 帧间隙	419
21.4.3 CAN 波特率设置	419
21.4.4 接收标识符过滤	420



21.4.5	BUFFLOCK 功能	422
21.4.6	接收功能描述	423
21.4.6.1	接收结构	423
21.4.6.2	接收程序	423
21.4.7	发送功能描述	424
21.4.7.1	发送优先级配置	424
21.4.7.2	发送程序	425
21.4.8	中断	425
21.4.9	时间戳计数器	426
21.4.10	诊断自检测试	427
21.4.10.1	配置环回测试	427
21.4.10.2	配置静音测试	427
21.4.10.3	配置环回静音测试	428
21.5	工作方式	429
21.5.1	管脚复用配置	429
21.5.2	时钟门控	429
21.5.2.1	打开门控时钟	429
21.5.2.2	关闭门控时钟	429
21.5.3	软复位	430
21.5.4	正常工作模式	430
21.5.5	诊断测试模式	431
21.5.5.1	配置 CAN 控制器自检测试	431
21.6	DFX	432
21.6.1	涉及寄存器基地址	432
21.6.2	常见配置检测	432
21.6.3	常见 Debug 手段	432
22	FAN	440
22.1	概述	440
22.2	特性	440
22.3	关键术语	440
22.4	功能	440



22.5 工作方式	441
22.5.1 管脚复用配置	441
22.5.2 读风扇转速操作	441
22.5.3 开启转速异常中断上报	442
22.5.4 FAN7~FAN0 的寄存器使用说明	442
22.5.5 FAN15~FAN8 的寄存器使用说明	444
23 PWM	446
23.1 概述	446
23.2 特性	446
23.3 关键术语	447
23.4 功能	447
23.5 工作方式	447
23.5.1 管脚复用配置	447
23.5.2 开启或关闭 PWM 输出	447
23.5.3 配置周期	447
23.5.4 配置占空比	448
23.5.5 配置复位	448
23.5.6 PWM5~PWM0 的寄存器使用说明	448
23.5.7 PWM11~PWM6 的寄存器使用说明	449
24 VGA	450
24.1 概述	450
24.2 特性	450
24.3 关键术语	451
24.4 功能	451
24.5 工作方式	452
24.5.1 初始化	452
24.5.2 软复位	452
25 USB	453
25.1 概述	453
25.2 特性	454
25.2.1 USB	454



25.2.2 功能.....	454
25.2.3 Host 模式功能描述.....	454
25.2.4 Device 模式功能描述.....	454
25.2.4.1 命令结构.....	457
Set Endpoint Configuration:DEPCFG.....	460
Set Endpoint Transfer Resource Configuration:DEPXFERCFG.....	462
Get Endpoint State:DEPGETSTATE.....	463
Set Stall and Clear Stall:DEPSSTALL,DEPCSTALL.....	463
Start Transfer:DEPSTRTXFER.....	464
Update Transfer:DEPUPDXFER.....	465
End Transfer:DEPENDXFER.....	465
Start New Configuration:DEPSTARTCFG.....	466
25.3 工作方式.....	470
25.3.1 时钟门控.....	470
25.3.2 USB 控制器软复位.....	471
25.3.3 USB Misc 模块的软复位.....	472
25.3.4 USB PHY 全局软复位.....	472
25.3.5 USB PHY 端口软复位.....	473
25.3.6 USB2.0/3.0 初始化流程.....	473
26 HOSTRTC	477
26.1 概述.....	477
26.2 特性.....	477
26.3 关键术语.....	477
26.4 功能.....	477
26.5 工作方式.....	478
26.5.1 初始化.....	478
26.5.2 读 HOSTRTC 操作.....	478
27 BMCRTC	480
27.1 概述.....	480
27.2 特性.....	480
27.3 关键术语.....	480
27.4 功能.....	480
27.5 工作方式.....	481



27.5.1 初始化	481
27.5.2 读 BMCRTC 操作	481
28 GPU	482
28.1 概述	482
28.2 特性	482
28.3 关键术语	483
28.4 功能	483
28.5 工作方式	484
28.5.1 时钟门控	484
28.5.1.1 打开门控时钟	484
28.5.1.2 关闭门控时钟	484
28.5.2 软复位	484
28.5.3 初始化	485
29 VCE	486
29.1 概述	486
29.2 概述	486
29.3 特性	486
29.4 功能	488
29.4.1 应用框图	488
29.4.2 相关模块	488
29.4.3 数据结构	489
29.4.4 VCE 功能	490
29.4.4.1 压缩功能	490
29.4.4.2 录屏功能	492
29.4.4.3 截屏功能	494
29.5 工作方式	494
29.5.1 软复位	494
29.5.1.1 正常软复位	494
29.5.1.2 初始化	495
29.5.1.3 VCE 模式二初始化	495
29.5.2 VCE 压缩数据读取	496



29.5.2.1 模式一	496
29.5.2.2 模式二	497
29.5.3 VCE 录屏读取	498
29.5.3.1 模式一读取录屏	498
29.5.3.2 模式二读取录屏	499
29.5.4 VCE 截屏	500
29.5.4.1 模式一截最后一屏	500
29.5.4.2 模式二截最后一屏	502
29.5.4.3 手动截屏	503
29.5.4.4 管脚截屏	506
29.5.5 压缩模式切换操作	508
29.5.5.1 模式一切换成模式二	508
29.5.5.2 模式二切换成模式一	509
29.6 DFX	509
29.6.1 提供智能心跳功能	509
29.6.2 支持内部所有寄存器可访问	510
29.6.3 支持统计寄存器可访问	510
29.6.4 支持关键信号观测	511
29.6.5 支持 CPU 模拟 Frame buffer 输入	511
30 DMAC	513
30.1 概述	513
30.2 特性	513
30.3 关键术语	514
30.4 功能	515
30.4.1 几个概念	515
30.4.2 控制器类型	515
30.4.3 外设请求分配	516
30.4.4 普通外设请求	518
30.4.4.1 请求产生机理	518
30.4.5 中断	519
30.4.5.1 中断产生	519



30.4.5.2 中断查询	520
30.5 工作方式	520
30.5.1 初始化	520
30.5.2 启动通道	520
30.5.3 中断处理	522
30.5.4 时钟门控	522
30.5.4.1 打开门控时钟	522
30.5.4.2 关闭门控时钟	522
30.5.5 UART 在 DMA 方式下的数据传输	523
30.5.5.1 初始化	523
30.5.5.2 数据发送	523
30.5.5.3 数据接收	523
30.5.6 SPI 在 DMA 方式下的数据传输	524
30.5.6.1 数据发送	524
30.5.6.2 数据接收	524
30.5.7 I2C 在 DMA 方式下的数据传输	525
30.5.7.1 初始化	525
30.5.7.2 数据发送	525
30.5.7.3 数据接收	525
30.6 DFX	526
31 TIMER.....	527
31.1 概述	527
31.2 特性	527
31.3 关键术语	528
31.4 功能	528
31.4.1 TICK.....	528
31.4.2 流量控制	528
31.4.3 计数范围和计数精度	529
31.4.4 TIMER 计数模式	529
31.5 工作方式	530
31.5.1 初始化	530
31.5.2 中断处理	531



32 WatchDog	532
32.1 概述	532
32.2 特性	532
32.3 关键术语	533
32.4 功能	533
32.5 工作方式	534
32.5.1 时钟门控	534
32.5.1.1 打开门控时钟	534
32.5.1.2 关闭门控时钟	534
32.5.2 中断处理过程	535
32.5.3 关闭 WatchDog	535
32.5.4 注意事项	535
33 JLC	537
33.1 概述	537
33.2 特性	537
33.3 关键术语	538
33.4 功能	538
33.4.1 模式 0	538
33.4.2 模式 1	539
33.4.2.1 自动 TMS 模式	539
33.4.2.2 手动 TMS 模式	540
33.4.3 延迟采样 TDO	540
33.5 工作方式	541
33.5.1 初始化	541
33.5.2 软复位	541
33.5.3 时钟门控	541
33.5.3.1 打开门控时钟	541
33.5.3.2 关闭门控时钟	542
33.5.4 模式 0	542
33.5.5 模式 1 自动 TMS	543
33.5.6 模式 1 手动 TMS	544



33.5.7 JLC RAM 初始化.....	545
33.6 DFX	545
34 SGPIO	546
34.1 概述	546
34.2 特性	546
34.3 关键术语	548
34.4 接口信号	548
34.5 功能描述	548
34.5.1 应用框图	548
34.6 工作方式	549
34.6.1 管脚复用配置	549
34.6.2 初始化配置	549
35 HPRE.....	553
35.1 关键术语	553
35.2 概述	553
35.3 特性	554
35.4 遵从协议	555
35.5 功能描述	555
35.5.1 软硬件交互界面.....	555
35.5.1.1 总体描述.....	555
35.5.1.2 队列说明.....	558
35.5.1.3 队列指针和循环空间.....	559
35.5.2 中断.....	561
35.5.2.1 中断寄存器	561
35.5.2.2 队列任务中断上报机制	562
35.5.3 椭圆曲线加密算法辅助运算.....	562
35.5.3.1 点加运算	562
35.5.3.2 点乘运算	562
35.6 编程接口	563
35.6.1 HPRE 输入 BD.....	563
35.6.1.1 数据结构.....	563



35.6.1.2 数据结构域段说明	565
35.6.2 HPRE 输出 BD	566
35.6.2.1 数据结构	566
35.6.2.2 数据结构域段说明	568
35.6.3 固件编程接口	570
35.6.4 应用编程接口	570
35.6.4.1 任务添加操作	570
35.6.4.2 任务完成操作	571
35.6.4.3 任务参数存储及访问方式	571
35.6.4.4 异常检测与处理	580
35.7 应用配置	582
35.7.1 ECC 算法核的任务数据配置约束	582
35.7.2 算法核切换流程	583
35.7.3 队列切换流程	583
35.8 初始化	584
35.8.1 初始化流程	584
35.8.2 HPRE_COMMON 页签初始化	584
35.8.3 HPRE_CLUSTER 页签初始化	585
35.8.4 QMV1_QUEUE_VM 页签初始化	585
35.8.5 HPRE_CLUSTER 页签初始化寄存器改配流程	586
35.8.6 QMV1_QUEUE_VM 页签初始化寄存器改配流程	586
35.9 时钟复位	587
35.9.1 时钟门控	587
35.9.2 软复位流程	587
35.10 性能指标	589
35.10.1 延迟 (Latency)	589
36 SEC	590
36.1 概述	590
36.1.1 相关模块	590
36.2 特性	591
36.3 关键术语	594



36.4 遵循协议	595
36.5 功能描述	596
36.5.1 任务	596
36.5.1.1 BD	596
36.5.1.2 WB_BD	596
36.5.2 任务流程	597
36.5.3 保序	598
36.5.3.1 保序完成	598
36.5.3.2 原子保序	598
36.5.4 数据类型	599
36.5.5 BD 数据结构	599
36.5.5.1 BD	600
36.5.5.2 BD 域说明	615
36.5.5.3 BD 域的依赖关系	627
36.5.5.4 透传包处理	628
36.5.6 存储结构	629
36.5.6.1 源数据 buffer 存储结构	629
36.5.6.2 目的数据 buffer 存储结构	630
36.5.7 数据格式	631
36.5.7.1 通用格式	631
36.5.8 地址和长度	633
36.5.8.1 报文地址和长度	633
36.5.8.2 回写报文的范围	633
36.5.8.3 Key 地址和长度	634
36.5.8.4 IV 地址和长度	634
36.5.8.5 随机数地址和长度	634
36.5.8.6 MAC 地址和长度	634
36.5.8.7 对齐	635
36.5.9 Key	635
36.5.10 IV	636
36.5.10.1 IV 的组成	636
36.5.10.2 IV 的地址和长度	637



36.5.10.3 IV 的获取	637
36.5.11 随机数	638
36.5.11.1 申请随机数	638
36.5.11.2 回写随机数	640
36.5.11.3 随机数的地址和长度	640
36.5.12 MAC	641
36.5.13 认证功能	641
36.5.14 机密功能	642
36.5.14.1 AES-CBC 加密	643
36.5.14.2 AES-CBC 解密	644
36.5.14.3 AES-CTR 加密	644
36.5.14.4 AES-CTR 解密	645
36.5.14.5 AES-GCM 加密	646
36.5.14.6 AES-GCM 解密	647
36.5.15 认证+机密组合	648
36.5.16 Hardcopy 功能	648
36.5.17 任务生成	649
36.5.18 任务结束	649
36.5.18.1 回写的目的地	649
36.5.18.2 回写的内容	650
36.5.18.3 查询结果	651
36.5.19 低功耗	652
36.5.20 中断控制	652
36.5.20.1 中断源	652
36.5.20.2 中断处理	652
36.6 操作流程	653
36.6.1 BD FIFO 接口	653
36.6.2 模块复位	654
36.6.3 模块暂停	655
36.6.4 资源回收	655
36.6.5 初始化	656



36.6.5.1 PF 初始化.....	657
36.6.5.2 VF 初始化.....	658
36.6.5.3 BD FIFO 初始化.....	659
36.6.6 执行任务	660
36.6.7 长数据 HASH	661
36.6.8 Flush 随机数缓存.....	662
36.6.8.2 SEC 常见异常处理	662
36.6.8.3 BD MEM 可视化.....	675
37 TRNG.....	676
37.1 概述	676
37.2 关键术语	678
37.3 特性	679
37.4 遵循协议	681
37.5 功能描述	682
37.5.1 生成随机数.....	682
37.5.1.1 NRBG.....	682
37.5.1.2 DRBG.....	682
37.5.1.3 混合随机数	682
37.5.2 随机数测试.....	682
37.5.3 中断.....	683
37.5.3.1 中断寄存器	683
37.5.3.2 中断处理.....	684
37.6 编程接口	684
37.6.1 固件编程接口	684
37.6.2 应用编程接口	684
37.6.2.1 获取随机数--不使用加速器专用硬件通道接口	684
37.6.2.2 获取随机数--使用 USER 私有接口时	685
37.6.2.3 查询方式调节 FRO delay	685
37.6.2.4 中断方式调节 FRO delay	686
37.6.2.5 SP 800-90A DRBG re-seed 操作流程.....	686
37.6.2.6 测试 AIS-31 Monobit,run,poker 测试电路操作流程.....	687
37.6.2.7 测试 repetition count, Adaptive Propotion 测试电路操作流程	688



37.6.2.8 测试 SHA256 加工功能操作流程.....	689
37.6.2.9 测试 SP 800-90A DRBG-AES_ECB_256 操作流程.....	689
37.6.2.10 测试标准的 NIST SP 800-90A DRBG 操作流程.....	691
37.6.2.11 获取噪声源数据操作流程.....	694
37.7 复位和初始化.....	695
37.7.1 复位.....	695
37.7.2 硬件初始化.....	695
37.7.3 软件初始化.....	696
37.7.3.1 软件初始化—不使用加速器专用硬件通道接口.....	696
37.7.3.2 软件初始化—使用加速器专用硬件通道接口时.....	697
37.7.3.3 RNG 初始化操作流程.....	698
37.8 注意事项.....	699
37.8.1 Debug.....	699
37.9 故障处理指南.....	699
37.10 性能.....	699
37.10.1 FRO 的熵 bit 速率.....	699
37.10.2 TRNG 随机数 bit 速率.....	699
38 TSENSOR	700
38.1 概述.....	700
38.2 特性.....	700
38.3 关键术语.....	700
38.4 功能.....	700
38.5 工作方式.....	701
38.5.1 TSensor 校准.....	701
39 SerDes	703
39.1 概述.....	703
39.2 特性.....	703
39.2.1 Combo phy 通用特性.....	703
39.2.2 Combo phy 独立特性.....	704
39.3 功能和工作方式.....	704
39.3.1 SGMII Combo Phy 初始化.....	704



39.3.2	---结束 PCIE Combo Phy 初始化.....	707
39.3.2.1	单板上电流程.....	708
39.3.2.2	PCIE 复位后启动流程	709
39.3.3	USB3 Combo Phy 初始化.....	709
39.3.4	ComboPhy 的 CR 接口	711
39.3.4.1	Combo Phy CR 接口控制寄存器.....	711
39.3.4.2	ComboPhy 的寄存器写操作	715
39.3.4.3	ComboPhy 的寄存器读操作	717
39.3.5	ComboPhy 的 powerdown	718
39.3.5.1	Powerdown 流程	718
39.3.5.2	Powerup 流程.....	718
39.3.5.3	相关寄存器	718
39.3.6	ComboPhy 的寄存器	719
39.3.6.1	ComboPhy 上电流程用到寄存器.....	719
39.3.6.2	ComboPhy SI 参数控制寄存器	722
39.3.7	ComboPhy 参数 Override	732
39.4	Serdes 环回工作模式	734
39.4.1	环回模式概述	734
39.4.2	Loopback 寄存器	735
39.4.2.1	Loopbak Override	735
39.5	Serdes 工作状态查询	735
39.5.1	SGMII combo phy.....	735
39.5.2	PCIE combo phy.....	737
39.5.3	USB3 combo phy.....	739
39.6	PRBS 测试	741
39.6.1	测试原理	742
39.6.2	测试寄存器	742
39.6.2.1	Pattern Generation	742
39.6.2.2	Pattern Matching.....	743
39.6.3	测试步骤	745
39.7	接收眼图	747
39.7.1	眼图参数设置	747
39.7.2	眼图数据采集	747



39.7.3 眼图图形生成	747
39.8 Serdes Tuning 启动流程.....	748
39.8.1 SGMII 和 USB 的 Tuning 启动流程	748
39.8.1.1 Tuning 初始化流程	749
39.8.1.2 简化流程参考	750
USB 流程.....	750
SGMII 流程.....	750
39.8.2 PCIE 的 Tuning 启动流程.....	750
39.8.2.1 Tuning 初始化流程	750
39.8.2.2 参考寄存器	751
40 ADC	755
40.1 概述	755
40.2 特性	755
40.3 关键术语	756
40.4 功能	756
40.5 工作方式	756
40.5.1 初始化	756
40.5.2 软复位	757
40.5.3 工作模式	758
40.5.3.1 ADC0 单独工作	758
40.5.3.2 ADC0 和 ADC1 同时工作	758
40.5.4 时钟频率配置	759
41 EFUSE.....	760
41.1 概述	760
41.2 特性	760
41.3 功能	762
41.3.1 应用框图	762
41.3.2 相关模块	764
41.4 工作方式	765
41.4.1 初始化	765
41.4.2 EFUSE 烧写.....	765



41.4.2.1 烧写方式	765
41.4.2.2 烧写时间	765
41.4.2.3 烧写基本流程	765
41.4.2.4 补充流程 1	767
41.4.2.5 补充流程 2 (烧写是否成功)	768
41.4.2.6 补充流程 3 (是否复位)	769
41.4.2.7 补充流程 4 (EFUSE 单独软复位需要加的步骤)	769
41.4.3 通过 DJTAG_MSTR 读 efuse	770
41.4.4 在线更新 EFUSE 中的 RIM 值	770
42 A55 低功耗	771
42.1 概述	771
42.2 低功耗特性	771
42.3 关键术语	772
42.4 遵循协议	772
42.5 功能描述	772
42.6 应用场景	773
42.7 工作流程	773
42.7.1 Core 硬件下电流程	774
42.7.2 Core 软件下电流程	775
42.7.3 Core 硬件上电流程-软件发起	776
42.7.4 Core 硬件上电流程-中断唤醒	777
42.7.5 Core 软件上电流程	778
42.7.6 Q-Channel 配置	779
43 JTAG	780
43.1 概述	780
43.2 特性	780
43.3 关键术语	781
43.4 功能	781
43.5 工作方式	782
43.5.1 JTAG 拓扑配置	782
43.5.2 SOC JTAG	784



43.5.3 ARM JTAG	785
43.5.4 DJTAG TAP	785
43.5.4.1 DJTAG 概述	785
43.5.4.2 DJTAG 指令	785
43.5.5 DJTAG Master	787
43.5.5.1 DJTAG 概述	787
43.5.5.2 DJTAG Master 访问配置流程	789
43.5.6 安全设计	790
44 SRAM/ROM	792
44.1 概述	792
44.2 特性	792
44.3 关键术语	793
44.4 功能	793
44.5 工作方式	793
44.5.1 初始化	793
44.5.2 单比特 ECC 错误回写使能和关闭	794
44.5.3 ECC 错误上报	794
44.6 DFX	795
45 HPM	796
45.1 概述	796
45.2 特性	796
45.3 关键术语	796
45.4 功能	797
45.5 工作方式	797
45.5.1 时钟门控	797
45.5.2 软复位	798
45.5.3 HPM 时钟控制	798
45.5.4 HPM 状态监控	799
46 IOCONFIG	800
46.1 概述	800
46.2 特性	800
46.3 功能描述	800



46.3.1 应用框图	800
46.3.2 管脚信号重用模块 IOMUX	801
46.3.3 启动功能	802
46.3.4 防对冲功能	802
46.3.5 管脚属性配置	802



插图目录

图 1-1 整芯片逻辑框图.....	2
图 2-1 Cortex M3 框图.....	6
图 2-2 JTAG-DP 的物理连接.....	9
图 3-1 FCM 结构框图.....	20
图 3-2 运行模式关系图.....	23
图 4-1 GIC 结构示意图	27
图 4-2 中断传递流程图.....	28
图 4-3 1/N SPI 中断亲和性路由示例 (IRM=0)	31
图 4-4 1/N SPI 中断亲和性路由示例 (IRM=1)	31
图 4-5 SGI 中断亲和性路由示例	32
图 4-6 SGI 中断广播路由示例.....	32
图 4-7 中断优先级结构示意图	34
图 5-1 系统启动流程图.....	46
图 5-2 A55 启动地址映射图	48
图 5-3 A55 启动双 bios 映射图	48
图 6-1 芯片时钟区域图.....	51
图 6-2 CRG 时钟架构	52
图 6-3 PLL 内部输出时钟频率关系示意图	53
图 7-1 复位管脚域内部模块对应关系图	61
图 7-2 上电复位时序图.....	63



图 7-3 热复位流程图	65
图 7-4 管脚触发协处理模式域复位时序	67
图 7-5 软件触发协处理模式域复位时序	68
图 7-6 WatchDog 触发协处理模式域复位时序	70
图 7-7 软件触发协处理模式非安全域复位时序	71
图 7-8 WatchDog 触发协处理模式非安全域复位时序	73
图 7-9 管脚触发 host 域单独复位时序	74
图 7-10 软件触发 host 域单独复位	75
图 7-11 PCIE0 相关模块单独复位时序	76
图 7-12 PCIE1 单独复位时序	77
图 8-1 DDR4 独立通道 RRBC 映射示意图	88
图 8-2 DDR4 独立通道 RRCBC 映射示意图	89
图 8-3 DDRC 调度示意图	90
图 8-4 优先级调度场景	91
图 8-5 效率优先调度场景	92
图 8-6 timeout 调度场景	93
图 8-7 timeout 调度场景 2	94
图 8-8 DDR4 CA Parity 时序图	100
图 8-9 DataScrambling 框图	100
图 9-1 8bit (cf_addr_offset=1) 或 8/16 (cf_addr_offset=0)	108
图 9-2 16bit (cf_addr_offset=1)	108
图 9-3 LocalBus 复用无外部 Ready 单次读操作	113
图 9-4 LocalBus 复用无外部 Ready 单次写操作	113
图 9-5 LocalBus 复用无外部 Ready 连续读操作	114
图 9-6 LocalBus 复用无外部 Ready 连续写操作	114
图 9-7 LocalBus 复用无外部 Ready 读写切换操作	115
图 9-8 LocalBus 复用无外部 Ready 写读切换操作	115



图 9-9 LocalBus 复用有外部 Ready 单次读操作	115
图 9-10 LocalBus 复用有外部 Ready 单次写操作	116
图 9-11 LocalBus 复用有外部 Ready 连续读操作.....	116
图 9-12 LocalBus 复用有外部 Ready 连续写操作	116
图 9-13 LocalBus 复用有外部 Ready 读写切换操作.....	117
图 9-14 LocalBus 复用有外部 Ready 写读切换操作.....	117
图 9-15 LocalBus 复用有外部 Ready 外部 Ready 超时.....	117
图 11-1 SMBUS 典型应用.....	135
图 11-2 I2C 和 SMBUS 控制器复用关系示意图.....	136
图 11-3 SMBUS TLOW_MEXT 和 TLOW_SEXT	138
图 11-4 SMBUS Master 超时处理流程	138
图 11-5 SMBUS Master ARP 配置流程.....	146
图 12-1 UART 控制器与接口管脚映射图	152
图 12-2 UART 3 线模式应用框图.....	154
图 12-3 UART 5 线模式应用框图.....	155
图 12-4 UART 7 线模式应用框图.....	155
图 12-5 UART 帧格式	156
图 12-6 UART 5 线模式结构框图.....	157
图 12-7 /CTS 功能时序图.....	158
图 12-8 /RTS 时序图（接收 FIFO 水线为 1/4/8Byte）	159
图 12-9 /RTS 时序图（接收 FIFO 水线为 14Byte）	159
图 12-10 数据采样过程	160
图 12-11 UART 的 IOMUX.....	162
图 12-12 UART 接口控制示意图.....	163
图 12-13 UART 全互联原理.....	164
图 12-14 PORT 和 UART 的非默认互联.....	164
图 12-15 UART 内部互联.....	165



图 12-16 PORT 的串入串出.....	165
图 12-17 通用 UART SOL 逻辑框图.....	170
图 13-1 SPI 应用框图.....	173
图 13-2 SPI 发送数据流程图.....	174
图 13-3 SPI 接收数据流程图.....	175
图 13-4 MotorolaSPI 单帧帧格式 (spo=0、sph=0)	176
图 13-5 MotorolaSPI 连续帧帧格式 (spo=0、sph=0)	176
图 13-6 MotorolaSPI 单帧帧格式 (spo=0、sph=1)	177
图 13-7 MotorolaSPI 连续帧帧格式 (spo=0、sph=1)	177
图 13-8 MotorolaSPI 单帧帧格式 (spo=1、sph=0)	178
图 13-9 MotorolaSPI 连续帧帧格式 (spo=1、sph=0)	178
图 13-10 MotorolaSPI 单帧帧格式 (spo=1、sph=1)	179
图 13-11 MotorolaSPI 连续帧帧格式 (spo=1、sph=1)	179
图 13-12 TI 同步串行单帧帧格式.....	180
图 13-13 TI 同步串行连续帧帧格式	180
图 13-14 数据传输流程图.....	184
图 14-1 eMMC 指令格式.....	187
图 14-2 eMMC 指令响应格式	188
图 14-3 eMMC 非数据指令操作.....	188
图 14-4 单块与多块读操作	189
图 14-5 单块与多块写操作	190
图 14-6 1bit 数据线传输模式下的块数据格式	191
图 14-7 4bit 数据线传输模式下的块数据格式	191
图 14-8 8bit 数据线传输模式下的块数据格式	191
图 14-9 Original Boot Operation 时序图	192
图 14-10 Alternative Boot Operation	193
图 14-11 双 buffer 结构示意图	202



图 14-12 链结构示意图	202
图 14-13 32bit 位宽的描述符的结构	203
图 15-1 双 buffer 结构示意图	222
图 15-2 链结构示意图	222
图 15-3 32bit 位宽的描述符的结构	223
图 16-1 MDIO 在 Hi3093 中的位置	231
图 17-1 I2C 典型应用电路原理图	242
图 17-2 I2C 数据传输时序图	243
图 17-3 I2C 数据传输帧格式图	243
图 17-4 I2C 复合数据传输格式	245
图 17-5 I2C 和 SMBUS 控制器复用关系示意图	245
图 17-6 I2C START 传输时序	252
图 18-1 网口 (GMAC_CLUSTER) 在 Hi3093 芯片中的位置	254
图 18-2 网口逻辑框图	260
图 18-3 接收报文流程示意图	261
图 18-4 发送报文流程示意图	262
图 18-5 接收报文存储格式	263
图 18-6 非分片报文 TAG 生成规则	269
图 18-7 分片 TAG 生成规则模式一	270
图 18-8 分片 TAG 生成规则模式二	271
图 18-9 多数据段报文发送数据组织	274
图 18-10 单数据段报文发送数据组织	274
图 18-11 非指令头模式下输入报文格式	277
图 18-12 硬件解析二层协议头	278
图 18-13 接收报文内容大小端配置说明	286
图 18-14 发送报文内容大小端配置说明	287
图 18-15 接收报文描述符大小端配置说明	288



图 18-16 发送报文描述符大小端配置说明	289
图 18-17 应用侧环回示意图	337
图 18-18 线路侧环回示意图	337
图 19-1 GPIO 模拟键盘的典型应用框图	341
图 20-1 PCIe 结构图	348
图 20-2 基于 PCIe 系统体系架构	349
图 20-3 PCIe 接口 EP 模式应用框图	350
图 20-4 PCIe 分层结构示意图	351
图 20-5 PCIe TLP 创建流程示意图	352
图 20-6 PCIe TLP 处理流程示意图	352
图 20-7 TLP 结构示意图	354
图 20-8 PCIe 存储器读写请求结构示意图	359
图 20-9 PCIe IO 读写请求结构示意图	363
图 20-10 PCIe 配置读写请求示意图	367
图 20-11 PCIe 完成请求结构示意图	371
图 20-12 PCIe 消息请求结构示意图	374
图 20-13 iATU 地址转换结构示意图	375
图 20-14 1 MB Write DMA Transfer from 0xBEE_BEE0 to 0xCAFE_CAF0	387
图 20-15 PCIe eDMA 写事务（非链表）应用流程参考图	388
图 20-16 1 MB Read DMA Transfer from 0xBEEF_BEE0 to 0xCAFE_CAF0	390
图 20-17 PCIe eDMA 读事务（非链表）应用流程参考图	390
图 20-18 PCIe eDMA 写事务（链表）应用流程参考图	392
图 20-19 PCIe eDMA 读事务（链表）应用流程参考图	397
图 20-20 XDMA HOST 流程图	402
图 20-21 PCIe 内部本地环回	405
图 20-22 Remote 环回测试	406
图 20-23 32-bit X4 Core @500Mhz:64-bit AXI @500Mhz:Memory Writes	410



图 20-24 32-bit X4 Core @500Mhz:64-bit AXI @500Mhz:Memory Reads	411
图 21-1 标准数据帧.....	417
图 21-2 扩展数据帧.....	417
图 21-3 标准远程帧.....	418
图 21-4 扩展远程帧.....	418
图 21-5 1 比特时间结构示意图	419
图 21-6 比特率时钟生成图.....	420
图 21-7 接受过滤结构图.....	421
图 21-8 接收不过滤 ID 示意图.....	421
图 21-9 接收过滤 ID 示意图	422
图 21-10 BUFFLOCK 打开后的数据储存示意图.....	422
图 21-11 接收过程示意图.....	423
图 21-12 应答间隙计数器示意图.....	427
图 21-13 环回测试	427
图 21-14 静音测试	428
图 21-15 静音环回测试	428
图 24-1 VDAC 应用图.....	451
图 25-1 USB 系统结构	453
图 25-2 TRB 结构	454
图 25-3 时钟选择逻辑	476
图 25-4 信号选择逻辑	476
图 25-5 观测值设置示例.....	476
图 25-6 trace 模式示例	476
图 28-1 GPU 应用框图.....	483
图 28-2 Gpu_dfx_awmisc_info 域段信息.....	485
图 29-1 VCE 应用图.....	488
图 29-2 压缩后的数据格式	490



图 29-3 块压缩格式	492
图 29-4 I/P 帧录屏存储格式	493
图 29-5 模式一截最后一屏软件处理流程图	500
图 29-6 模式二截最后一屏软件处理流程图	502
图 29-7 模式一 VC_CORE 手动截屏流程图	504
图 29-8 模式二 ENVC_CORE 手动截屏流程图	505
图 29-9 模式一 VC_CORE 管脚截屏流程图	506
图 29-10 模式二 ENVC_CORE 手动截屏流程图	507
图 30-1 普通外设请求产生机理	518
图 30-2 DMAC 中断产生示意图	519
图 33-1 串行 TDI&TMS 时 JTAG 链结构	539
图 33-2 手动发送模式	540
图 34-1 SGPIO 的应用框图	546
图 34-2 SGPIO 结构图	549
图 35-1 安全属性划分	551
图 35-2 DDR 的区域划分	551
图 35-3 安全子系统的结构	551
图 35-4 安全 GPIO、安全 UART、安全 I2C 和非安全 GPIO 的选通示意图	551
图 36-1 CER 模块应用流程简析	552
图 36-2 RIM 管理流程	552
图 36-3 根公钥校验	552
图 36-4 RIM 签名值认证流程	552
图 36-5 RIM 值更新流程	552
图 36-6 签名值认证流程	552
图 36-7 数据 HASH 计算流程	552
图 36-8 CER BD 存储示例	552
图 36-9 1024 比特大数按照 32bit 划分	552



图 36-10 1024bit 大数按照 CER 引擎要求的 32bit 大端存储结构	552
图 36-11 1024bit 大数按照 64bit 划分	552
图 36-12 1024bit 大数按照 CER 引擎要求的 64bit 大端存储结构	552
图 36-13 RSA 数字签名时的 PSS 填充方式	552
图 36-14 RIM 的明文输入数据结构(PKCS 1.5 填充).....	552
图 36-15 RIM 的明文输入数据结构(PSS 填充).....	552
图 36-16 长数据 HASH 数据拆分示意	552
图 36-17 根公钥值范例	552
图 36-18 根公钥 HASH 明文	552
图 36-19 ROTPK 值范例.....	552
图 36-20 RvkSubKeyCategory 值范例	552
图 36-21 E-Fuse 输出数据范例	552
图 36-22 RIM 明文	552
图 36-23 RIM 明文 HASH 值	552
图 36-24 RIM 签名值明文	552
图 36-25 RIM 签名值计算密钥.....	552
图 36-26 RIM 明文签名值	552
图 37-1 HPRE 逻辑框图	554
图 37-2 HPRE-CLUSTER 逻辑框图	554
图 37-3 Process Queue 的基本格式.....	556
图 37-4 Process Queue entry(HPRE BD)的基本格式	558
图 37-5 队列空图示.....	560
图 37-6 队列满图示.....	560
图 37-7 1024bit 大数按照 32bit 划分	571
图 37-8 1024bit 大数按照 HPRE 引擎要求的 32bit 大端存储结构	572
图 37-9 1024bit 大数按照 64bit 划分	572
图 37-10 1024bit 大数按照 HPRE 引擎要求的 64bit 大端存储结构	573



图 37-11 大数数据结构图例说明	573
图 37-12 椭圆曲线点乘计算参数数据结构	574
图 37-13 椭圆曲线点加计算参数数据结构	576
图 37-14 椭圆曲线点加计算参数数据结构	578
图 38-1 SEC 加速引擎交互模块	591
图 38-2 一个任务的 BD 和数据	597
图 38-3 源数据 buffer 存储结构	630
图 38-4 新数据 buffer 存储结构	631
图 38-5 报文地址长度解析	632
图 38-6 申请随机数路径	638
图 38-7 SEC 初始化分支流程图	657
图 39-1 TRNG 应用框图	676
图 39-2 TRNG 整体内部逻辑框图	677
图 39-3 接口时序	677
图 39-4 EIP-76d 的内部逻辑框图	678
图 39-5 TRNG_STATUS 状态寄存器	683
图 39-6 TRNG_CONTROL 控制寄存器	683
图 39-7 TRNG_INTACK 中断确认寄存器	684
图 39-8 NIST 测试向量	691
图 39-9 CTR_DRBG 测试向量	693
图 40-1 TSENSOR 在芯片中的位置图	702
图 41-1 Combo Phy 上电初始化时序图	705
图 41-2 上电时序	707
图 41-3 上电时序图	710
图 41-4 地址 Capture 传输 CR 时序	716
图 41-5 写操作 CR 时序	716
图 41-6 读操作 CR 时序	717



图 41-7 眼图绘制结果（举例）	748
图 42-1 ADC 应用图.....	756
图 43-1 EFUSE 应用图.....	762
图 43-2 JTAG_TAP 上其他 master	763
图 44-1 PMCTRL 结构框图	772
图 45-1 Hi3093 JTAG 拓扑结构图	781
图 45-2 Hi3093 JTAG 拓扑结构图	788
图 48-1 IOCONFIG 应用框图	801



表格目录

表 1-1 Hi3093 工作时钟	5
表 1-2 Hi3093 封装尺寸	5
表 2-1 ARM-M3 关键术语描述	7
表 2-2 协处理器 M3 地址空间映射表	11
表 2-3 安全核 M3 地址空间映射表	错误!未定义书签。
表 3-1 ARMv8-A 的异常等级	23
表 4-1 GIC 关键术语描述	25
表 4-2 GICD 中断说明	28
表 4-3 GICD 中断路由配置	29
表 4-4 GICD 中断亲和路由算法	30
表 4-5 安全软件视角中断优先级配置	33
表 4-6 非安全软件视角中断优先级配置	33
表 5-1 启动真值表	44
表 6-1 各 IP 时钟和 PLL 对应关系表	52
表 6-2 PLL 信号列表	54
表 6-3 CPLL 配置参考	57
表 6-4 PLL0 配置参考	57
表 6-5 PLL1 配置参考	58
表 6-6 PLL2 配置参考	59
表 7-1 软件复位模块表	78



表 7-2 DMAC 特殊复位寄存器	79
表 7-3 特殊复位寄存器	79
表 7-4 PCIE 特殊复位寄存器	80
表 8-1 DDRC 关键术语描述	86
表 8-2 16bit DDR4 器件组合	87
表 8-3 DDR 刷新命令数和带宽的关系	97
表 9-1 数据总线与物理管脚对应关系	107
表 9-2 数据总线字节顺序关系	107
表 9-3 地址总线与物理管脚对应关系 (cf_addr_offset=0, 1)	108
表 9-4 CS0 上电采样配置表	109
表 9-5 LocalBus 的可配置时序参数表	110
表 11-1 I2C 模块关键术语描述	134
表 11-2 SMBUS TimeOut 参数	137
表 11-3 IC_SS_SCL_HCNT/LCNT 典型配置值	139
表 11-4 IC_FS_SCL_HCNT 典型配置值	140
表 11-5 SMBus Bus 协议命令的使用	142
表 11-6 SMBus ARP 协议命令的实现	144
表 11-7 Hi3093 SMBUS 相关模块寄存器基地址	147
表 12-1 波特率误差	153
表 12-2 UART 接口模式	153
表 12-3 Hi3093 UART 相关模块寄存器基地址	171
表 14-1 eMMC 关键术语描述	187
表 14-2 eMMC 的工作模式频率要求	194
表 14-3 非数据传输指令时的寄存器 CMD 配置参考 (默认值)	197
表 14-4 单块或多块读数据时的寄存器 CMD 配置参考 (默认值)	199
表 14-5 单块或多块写数据时的寄存器 CMD 配置参考 (默认值)	200
表 14-6 DES0 各 bit 的含义	203



表 14-7 DES1 各 bit 的含义	204
表 14-8 DES2 各 bit 的含义	204
表 14-9 DES3 各 bit 的含义	205
表 15-1 SDIO 关键术语描述	213
表 15-2 非数据传输指令时的寄存器 MMC_CMD 配置参考（默认值）	217
表 15-3 单块或多块读数据时的寄存器 MMC_CMD 配置参考（默认值）	218
表 15-4 单块或多块写数据时的寄存器 MMC_CMD 配置参考（默认值）	220
表 15-5 DES0 各 bit 的含义	223
表 15-6 DES1 各 bit 的含义	224
表 15-7 DES2 各 bit 的含义	224
表 15-8 DES3 各 bit 的含义	225
表 15-9 Resume 操作时的寄存器 MMC_CMDARG 配置参考	228
表 16-1 MDIO 模块关键术语描述	231
表 16-1 MDIO 接口的 Clause22 数据帧格式	232
表 16-2 MDIO 接口的 Clause45 数据帧格式	233
表 16-3 MDIO 接口时钟频率.....	234
表 16-4 Hi3093 MDIO 相关模块寄存器基地址.....	240
表 17-1 I2C 模块关键术语描述	241
表 17-2 I2C_SS_SCL_HCNT 典型配置值	247
表 17-3 IC_SS_SCL_LCNT 典型配置值.....	247
表 17-4 IC_FS_SCL_HCNT 典型配置值	248
表 17-5 IC_FS_SCL_LCNT 典型配置值.....	248
表 17-6 Hi3093 I2C 相关模块寄存器基地址.....	253
表 18-1 GMAC 关键术语描述	259
表 18-2 Word1	264
表 18-3 Word2	265
表 18-4 IP 报文非分片包的 tag	266



表 18-5 从 2 层解析的非 TCP/UDP 报文或分片包的 TAG	266
表 18-6 从 3 层解析的非 TCP/UDP 报文或分片包的 TAG	267
表 18-7 从 3 层解析的非 IP 封装的非 TCP/UDP 报文或分片包的 TAG	267
表 18-8 提取方式计算得到的 Tag	268
表 18-9 默认方式的 TAG	268
表 18-10 报文描述符 WORD0 格式	274
表 18-11 报文描述符 WORD1 格式	275
表 18-12 报文描述符 WORD2 格式	275
表 18-13 报文描述符 WORD3 格式	276
表 18-14 WORD0	276
表 18-15 WORD1	276
表 18-16 Word1	278
表 18-17 Word2	279
表 18-18 Word3	279
表 18-19 Word4	279
表 18-20 Word5	280
表 18-21 Word6	282
表 18-22 发送方向截断报文及统计计数相关说明（1000M，MAX_FRM_SIZE 配置值为 N）	283
表 18-23 发送方向截断报文及统计计数说明（10/100M，MAX_FRM_SIZE 配置值为 N）	284
表 18-24 网口内部总线操作 Cache 类型说明	289
表 18-25 网口的门控时钟使能寄存器表 0x0CD0C000	303
表 18-26 网口的门控时钟禁止寄存器表 0x0CD0C000	304
表 18-27 网口的门控时钟状态寄存器表	305
表 18-28 网口的软复位寄存器表 0x0CD0C000	309
表 19-1 GPIO 关键术语描述	340
表 20-1 PCI Express 地址空间类型	354
表 20-2 4 DWords 存储器请求头字段	355



表 20-3 IO 请求头字段	360
表 20-4 配置请求头字段	364
表 20-5 完成请求头字段	368
表 20-6 4 DWords 消息请求头字段	372
表 20-7 iATU Register Map(offset address 0x700)	375
表 20-8 iATU Register Map(offset address 0x700)	381
表 20-9 Viewport (offset address 0x700+0x200)	381
表 20-10 Region Control1(offset address 0x700+0x204)	382
表 20-11 iATU Region Control2(offset address 0x700+0x208)	382
表 20-12 DMA 写通道相关配置寄存器	385
表 20-13 DMA 读通道相关配置寄存器	386
表 20-14 DMA 控制器寄存器配置流程（非链表写事务）	388
表 20-15 DMA 控制器寄存器配置流程（非链表读事务）	391
表 20-16 DMA 写初始化流程（链表）	393
表 20-17 DMA 控制器寄存器配置（链表写事务）	396
表 20-18 DMA 读初始化流程（链表）	398
表 20-19 DMA 控制器寄存器配置流程（链表读事务）	400
表 20-20 Remote loopback 模式下，iATU 配置（仅供参考）	407
表 20-21 PCIe 理想带宽利用率	409
表 20-22 TLP 报文格式	410
表 21-1 CANBUS 关键术语描述	415
表 21-2 报文内存单元结构表	415
表 21-3 所有 BUFFER 发送优先级配为 0 表	424
表 21-4 所有 BUFFER 发送优先级配不为 0 表	424
表 21-5 中断优先级编码列表	426
表 21-6 CANBUS DFX 用到寄存器基地址	432
表 21-7 CAN 控制器状态和诊断寄存器	433



表 22-1 FAN 关键术语描述	440
表 22-2 FAN7~FAN0 风扇转速寄存器和转速下限寄存器列表	443
表 22-3 FAN15~FAN8 风扇转速寄存器和转速下限寄存器列表	444
表 23-1 PWM 关键术语描述	447
表 24-1 VGA 关键术语描述	451
表 25-1 TRB 域段描述	455
表 25-2 Device 通用命令类型	458
表 25-3 DEPCFG	460
表 25-4 DEPXFERCFG	462
表 25-5 DEPSTRXFER	464
表 25-6 DEPENDXFER	466
表 25-7 Device Endpoint-n Events	467
表 25-8 Device-Specific Event	469
表 26-1 HOSTRTC 关键术语描述	477
表 27-1 BMCRTC 关键术语描述	480
表 28-1 GPU 关键术语描述	483
表 29-1 DVI 接口数据结构	489
表 29-2 Frame buffer 接口数据结构	489
表 30-1 DMAC 关键术语描述	514
表 30-2 外设请求分配表	516
表 31-1 TIMER 关键术语描述	528
表 31-2 Timer 计数范围和精度表	529
表 32-1 WATCHDOG 关键术语描述	533
表 33-1 JLC 关键术语描述	538
表 34-1 SGPIO 关键术语描述	548
表 34-2 SGPIO 接口信号描述	548
表 35-1 各个 master 对 slaver 的访问权限	551



表 35-2 各个 master 对系统寄存器的访问权限	551
表 35-3 启动管脚控制	551
表 36-1 CER 关键术语描述	552
表 36-2 RIM 每比特计算真值表	552
表 36-3 输入的 CER BD	552
表 36-4 输入的 CER BD 域段描述	552
表 36-5 BD 检查说明	552
表 36-6 回写的 CER BD 域段描述	552
表 36-7 RIM 数据结构	552
表 36-8 RIM 数据的域段说明	552
表 36-9 RIM 数据检查	552
表 36-10 RIM BD	552
表 36-11 普通 HASH 计算 BD	552
表 36-12 长数据 HASH 计算 Head BD	552
表 36-13 长数据 HASH 计算 Body BD	552
表 36-14 长数据 HASH 计算 Tail BD	552
表 36-15 签名值认证计算 BD	552
表 36-16 RSA 模幂运算计算延迟	552
表 36-17 签名校验计算延迟	552
表 36-18 总线低延迟或者零延迟情况下 SHA256 核计算延迟	552
表 37-1 HPRE 关键术语描述	553
表 37-2 HPRE 输入 BD	563
表 37-3 HPRE 输入 BD 域段说明	565
表 37-4 HPRE 输出 BD 数据结构	566
表 37-5 HPRE 输出 BD 域段说明	568
表 37-6 ECC 输入 BD 检查	580
表 37-7 ECC 任务参数检查	581



表 37-8 ECC 算法核的任务数据配置约束	582
表 37-9 椭圆曲线点加计算延迟 (随机 10 个点加)	589
表 37-10 椭圆曲线点乘计算延迟 (私钥 K 值满位宽, 每字节取 0xAA)	589
表 38-1 SEC 关键术语描述	594
表 38-2 BD4 结构 (机密+认证, 128B)	600
表 38-3 WB_BD4 结构 (机密+认证, 128B)	603
表 38-4 BD5 结构 (长数据 hash, BD 大小 128B)	606
表 38-5 WB_BD5 结构 (长数据 hash, BD 大小 128B)	609
表 38-6 {T_Len, T_TagValue, Dsc_Len}出错时的 WB_BD 结构 (BD 大小 128B)	612
表 38-7 BD 域的说明	615
表 38-8 WB_BD 域的说明	623
表 38-9 地址和长度的对齐要求	635
表 38-10 IV 的成分和格式	636
表 38-11 认证算法的属性摘要	641
表 38-12 回写结果数据的类型	650
表 38-13 BD FIFO 任务控制组合	653
表 38-14 SEC 异常分类列表	663
表 39-1 TRNG 关键术语描述	678
表 40-1 TSENSOR 关键术语描述	700
表 40-2 EFUSE 域段中的校准信息表	701
表 41-1 Serdes 接口应用表	703
表 41-2 上电流程中参数值表	705
表 41-3 上电时序	708
表 41-4 POR Timing Values	710
表 41-5 USB3PHY 的 CR 接口寄存器	711
表 41-6 PCIE PHY 的 CR 接口寄存器	712
表 41-7 SGMII PHY 的 CR 接口寄存器	714



表 41-8 PCIE PHY 的 phy reset.....	719
表 41-9 PCIE PHY 的 Pipe reset	720
表 41-10 USB3 PHY 的 Phy reset	721
表 41-11 USB3 PHY 的 Pipe reset	722
表 41-12 SGMII 参数寄存器 1.....	722
表 41-13 SGMII 参数寄存器 2.....	723
表 41-14 PCIE_PHYCTL0 寄存器（偏移地址 0x30）	725
表 41-15 PCIE_PHYCTL1 寄存器（偏移地址 0x34）	726
表 41-16 PCIE_PHYCTL2 寄存器（偏移地址 0x38）	727
表 41-17 USB 参数控制寄存器 1（偏移地址 0x1010）	728
表 41-18 USB 参数控制寄存器 2（偏移地址 0x1014）	729
表 41-19 USB 参数控制寄存器 3（偏移地址 0x1024）	731
表 41-20 TX 驱动参数重载寄存器（偏移地址 0x1002）	732
表 41-21 TX 驱动参数重载寄存器（偏移地址 0x1003）	733
表 41-22 RX 驱动参数重载寄存器（偏移地址 0x1006）	733
表 41-23 SGMII PHY 的状态信号	735
表 41-24 PCIE PHY 的状态信号.....	737
表 41-25 PCIE PHY 的状态信号.....	737
表 41-26 USB3 PHY 的状态信号.....	740
表 41-27 PRBS 测试原理图.....	742
表 41-28 寄存器 LANEN_DIG_TX_LBERT_CTL	742
表 41-29 LANEN_DIG_RX_LBERT_CTL.....	743
表 41-30 LANEN_DIG_RX_LBERT_ERR.....	744
表 42-1 ADC 关键术语描述.....	756
表 44-1 PMCTRL 关键术语描述	772
表 45-1 JTAG 关键术语描述	781
表 45-2 JTAG Chain 链配置表.....	783



表 45-3 DJTAG 指令列表.....	785
表 45-4 DJTAG Master 寄存器组.....	788
表 45-5 DJTAG 链上模块编号.....	790
表 46-1 SRAM 关键术语描述.....	793
表 47-1 HPM 关键术语描述	796



手册使用约束

须知

应用芯片时，需按照芯片手册注明的功能、性能、配置步骤等信息进行软硬件开发，未按手册注明的信息进行操作，可能会出现不可预知风险。切勿依商用器件经验推测，谢谢！



1 芯片简介

1.1 概述

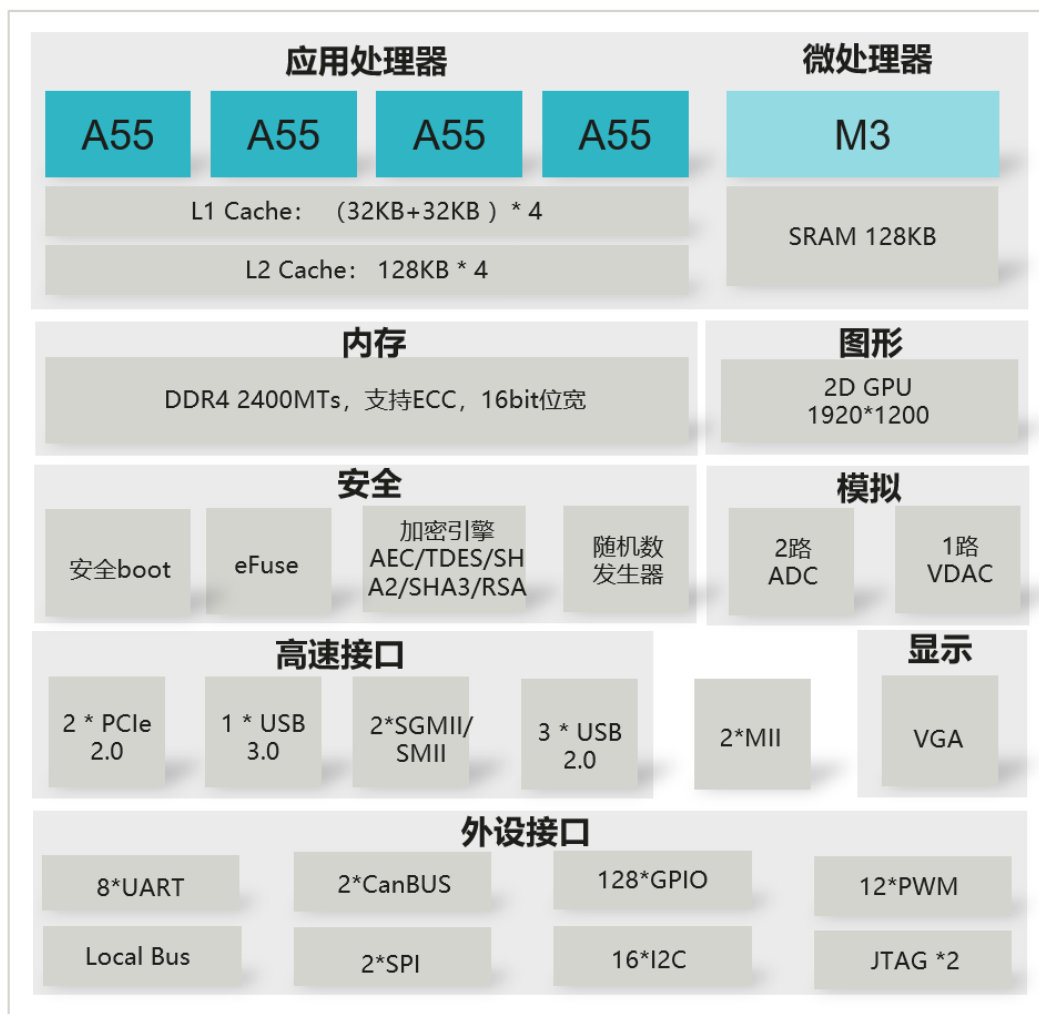
Hi3093 是一款针对工控、IOT 网关等领域的通用 MPU 芯片，包括一个主频最高 1GHz 的四核 A55 CPU，一个协处理 M3 主频 200MHz 及安全核 M3 主频 400MHz。芯片支持远程 KVM，支持 IPMI 管理接口，支持 PCIE 收发 MCTP 报文，支持本地显示 VGA，GE 网口、RMII 接口，以及其他丰富的板级管理和外设接口。

系统从功能维度分为 5 个子系统：

- A5X 子系统：包含 A55 和 GIC。
- 高速外设（HP）子系统：包含 SGMII/RMII、USB3.0、USB2.0、Serdes。
- 低速外设（LP）子系统：包含 M3、安全 M3、HPRE、CER、SEC、TRNG、efuse、DMAC、SFC、LocalBus、eMMC、ROM、RAM、UART、I2C、C80、KCS、mailbox、BT、SOL、WATCHDOG、SPI、SGPIO、Canbus、Sysctrl、MDIO、SDIO、GPIO、RTC、PWM、时钟、复位、(IOCONFIG 复用配置寄存器)、JLC、HOSTUART、AD。
- HOST 子系统：包含 PCIE0、PCIE1、GPU、VCE、HOSTRTC。
- DDR 子系统：包含 DDRC 和 DDR PHY。



图1-1 整芯片逻辑框图



1.2 功能特性简述

1.2.1 处理器核

- 集成一个 cluster, cluster 里 4 个 A55 处理器, 频率最高 1GHz, 可以静态配置分频。
- A55 为 64 位处理器, 兼容 ARM V8 指令集。
- 支持 CAS 指令。
- L1 Icache 32KB/L1 Dcache 32KB/L2 Cache 128KB。
- 支持 Crpto 加速/Dot Product 指令集、支持浮点 1.0Gflops/Core。



- 支持 L1/L2cache/TLB 保护。
- 处理器核内都带有硬件浮点处理单元，支持 IEEE754 标准单双精度浮点运算。
- 支持中断控制器 GIC，支持 GIC-V3 架构规范。

1.2.2 DDR 接口

- 集成单通道 DDR，支持 DDR4 16bits@2400MHz，支持片内 ECC 校验。
- 非安全区提供容量 128KB 的 SRAM。
- 安全区提供 1 块 256KB SRAM。

1.2.3 外部接口

- 集成 4 个 MAC 控制器，其中 2 路为 RMII 接口，另外 2 路为 SGMII 接口满足 10M/100M/1000M 应用。
- 集成 2 个 PCI-E 控制器，支持 PCI Gen2 标准。
- 集成 3 个 USB2.0 控制器，支持 DEVICE。
- 集成 1 个 USB3.0 控制器，支持 HOST 模式。
- 集成 SD2.0、MMC4.5 版本控制器，灵活解决可插拔存储卡应用。
- 集成 local bus 接口，支持 Nor Flash 启动。
- 集成 1 个 ADC 控制器，支持模拟电压检测。
- 集成 1 路 DAC，输出 VGA 信号。
- 集成 2 路 SFC 控制器，对接 SPI FLASH 器件，支持 SFC0 启动。
- 集成 1 个 CanBus 控制器。
- 集成 16 路 FAN 和 12 路 PWM。
- 集成 16 路 I2C，支持 SMBUS、IPMB 功能。
- 集成 8 路 UART，支持 1 路全功能模式，支持内部控制器全功能互联。
- 4 个 GPIO 控制器（129 根 GPIO 管脚与功能管脚复用）。
- 2 个 SPI。



1.2.4 加速器功能

- 支持密钥吊销。
- 提供了高性能并针对业务处理优化设计的 16 通道 DMAC。

1.2.5 显示功能

- 支持 VGA 接口显示输出。
- 集成视频图形数据压缩模块 VCE 对图像数据进行压缩处理，支持录屏、下电截屏功能。

1.2.6 安全功能

- 集成 M3 处理器，支持安全启动、BIOS 校验及强身份识别。
- 集成仅安全核可访问 ROM，容量为 32KB。
- 集成 eFuse 支持整体安全方案。
- 集成硬件加解密引擎，支持 RSA、ECDSA256、AES128-256、SHA256-512 及 HMAC 算法。
- 支持真随机数发生器，速率 16Kbps。

1.2.7 其他接口

- 集成协处理器 M3 进行协助处理实时性要求较高的业务。
- 集成 MCTP 管理部件传输协议，实现管理控制器如 CPU 和各设备之间的通信。
- 支持双 BIOS 启动，支持复位状态查询。
- 支持分业务模块复位，HOST 业务和 BMC 业务可独立复位。
- 集成 7 个 WatchDog。
- 集成 6 个 64bit Timer。
- 集成一个 HOSTRTC 和一个 BMCRTC。
- 支持 SOL 串口数据保存。
- 支持黑匣子功能，通过 PCIe 接口将数据写入 DDR 缓存。
- 支持处理器 JTAG 调试。
- 集成系统控制器，提供系统时钟、复位及系统模式切换功能。
- 支持 DRAM test。



1.3 硬件特性简述

1.3.1 参考时钟

表1-1 Hi3093 工作时钟

参考时钟	用途
25MHz	系统工作时钟。
100MHz	PCIE0、PCIE1、GMAC、USB3 SerDes 参考时钟（默认 100M）。

1.3.2 封装尺寸

表1-2 Hi3093 封装尺寸

芯片型号	封装尺寸	器件最大高度
Hi3093	17mm×17mm	-

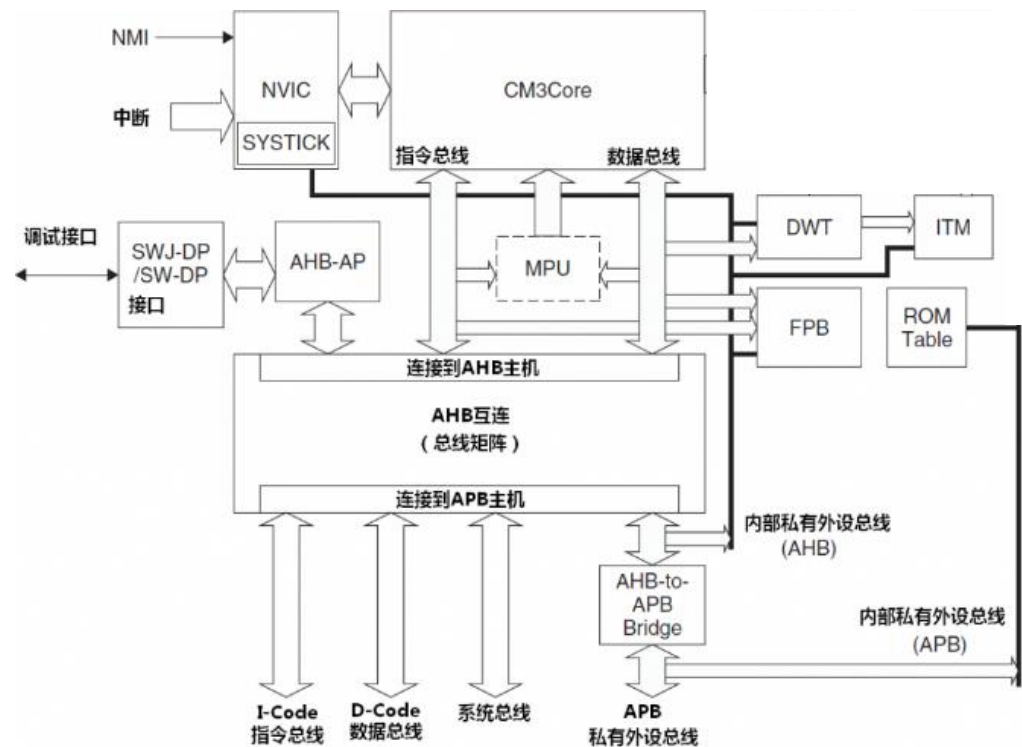
2 ARM-M3

2.1 概述

Hi3093 支持两个 Cortex M3 处理器，一个用于 IO 协处理器，一个用于安全处理器。

IO 协处理主要用于协助主核处理外设低速 IO 接口和 IP 的数据请求，中断和寄存器配置等业务。

图2-1 Cortex M3 框图





如图 2-1 所示，M3 对外访问的总线，有 Ibus，Dbus 和 System bus 三个 AHB 总线，分别用于取指，查表和读写操作。除此之外，在 M3 内部还有两个私有总线，分别是 AHB 和 APB，私有 AHB 总线相对来说用于 M3 内核内部的访问，主要用于访问中断控制（NVIC），闪存地址重载及断点单元（FPB），数据观察点单元（DWT），仪器化跟踪宏单元（ITM）等。私有的 APB 总线访问相对 M3 内核是访问外部的 rom table。

2.2 关键术语

表2-1 ARM-M3 关键术语描述

术语	描述
NVIC	Nested Vectored Interrupt Controller
MPU	Memory Protection Unit
FPB	Flash Patch and Breakpoint
DWT	Data Watchpoint
ITM	Instruction Trace Macrocell

2.3 特性

M3 核具有以下特性：

- M3 Core 工作时钟为系统 AHB 时钟。
- M3 Core 实现 ARMv7-M 架构。
- ARM Thumb 指令集和 Thumb-2 指令集。
- M3 Core 支持 4 种总线：I-Code，D-Code 和 System，M3 Core 通过这三种总线取指、访问数据，均为 AHB-Lite 协议总线；第 4 种总线为 PPB，私有外设总线，用于访问 debug 等内部私有外设。
- 支持 NVIC，支持 240 个外部中断，支持 8 级中断优先级配置。
- 支持 JTAG 使用 DAP 口完成 debug 功能。
- 支持级别是 full 的 debug：支持所有 debug 功能。
- 支持使用 MPU 配置 memory 属性。
- 芯片内部提供 WatchDog 监控，复位原因有记录。



- 支持 M3 访问 PECl 通道。

2.4 功能

2.4.1 Cortex M3 内核

Cortex-M3 内核基于哈佛架构，指令和数据各使用一条总线。由于指令和数据可以从存储器中同时读取，所以 Cortex-M3 处理器对多个操作并行执行，加快了应用程序的执行速度。

内核流水线分 3 个阶段：取指、译码和执行。当遇到分支指令时，译码阶段也包含预测的指令取指，这提高了执行的速度。处理器在译码阶段期间自行对分支目的地指令进行取指。在稍后的执行过程中，处理完分支指令后便知道下一条要执行的指令。如果分支不跳转，那么紧跟着的下一条指令随时可供使用；如果分支跳转，那么在跳转的同时分支指令可供使用，空闲时间限制为一个周期。

Cortex-M3 内核包含一个适用于传统 Thumb 和新型 Thumb-2 指令的译码器、一个支持硬件乘法和硬件除法的先进 ALU、控制逻辑和用于连接处理器其他部件的接口。

2.4.2 总线矩阵和接口

Cortex-M3 处理器总线矩阵把处理器和调试接口连接到外部总线；也就是把基于 32 位 AMBA® AHB-Lite 的 ICode、DCode 和系统接口连接到基于 32 位 AMBA APB™的专用外设总线 PPB (Private Peripheral Bus)。

32 位 ICode 接口用于获取代码空间中的指令，只有 CM3Core 可以对其访问。所有取指的宽度都是一个字，每个字里面的指令数目取决于所执行代码的类型及其在存储器中的对齐方式。32 位 DCode 接口用于访问代码存储空间中的数据，CM3Core 和 DAP 都可以对其访问。32 位系统接口分别获取和访问系统存储空间中的指令和数据，与 DCode 相似，可以被 CM3Core 和 DAP 访问。PPB 可以访问 Cortex-M3 处理器系统外部的部件。

2.4.3 中断 NVIC

NVIC 模块提供低延时可配置的中断处理能力，M3 支持 240 个 NVIC 中断接口，8 级优先级配置，可动态配置。有以下特性：

- 可嵌套中断支持
- 向量中断支持

- 动态优先级调整支持
- 低延迟
- 中断可屏蔽

NVIC 同时支持脉冲和电平类型的中断，电平类型中断可以由 ISR（Interrupt Service Routine）清除。脉冲类型中断需要保证可以被 Cortex-M3 的工作时钟 FCLK 的上升沿采集到。外设的中断会被送进 NVIC 中进行处理，此外 NVIC 还有提供一个 NMI 中断接口。对内核异常在中断前规划有 16 个系统异常，所以内部中断 0 的异常号为 16，依次类推。

2.4.4 MPU

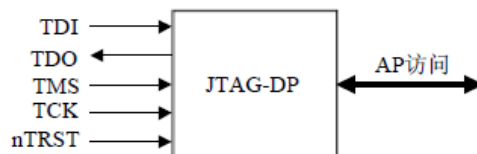
M3 包含 MPU，它通过保护用户应用程序中操作系统所使用的重要数据，分离处理任务（禁止访问各自的数据），禁止访问存储器区域，将存储器区域定义为只读，以及对有可能破坏系统的未知的存储器访问进行检测等方式来改善嵌入式系统的可靠性。支持以下特性：

- 保护区
- 重叠保护区
- 访问权限
- 将存储器属性输出到系统

2.4.5 调试端口

Hi3093 的 M3 支持用于调试访问的 AHB-AP，通过 DAP 口来实现。

图2-2 JTAG-DP 的物理连接



如图 2-2 所示，调试端口描述如下：

- JTAG 调试端口（JTAG-DP）。JTAG-DP 基于 IEEE 1149.1 测试访问端口（TAP）以及边界扫描结构，广义上称作 JTAG，它向 AHB-AP 端口提供 JTAG 接口。



- 串行线调试端口 (SW-DP)。SW-DP 向 AHB-AP 端口提供两个脚的接口 (时钟和数据)

一次只能使用一个 DP，只有当两个 DP 都没用的时候才可在这两个调试端口间进行切换。在上电复位时默认为 JTAG 模式，并且可以通过外部调试硬件所提供的控制序列进行协议的切换。

2.4.6 WatchDog

Hi3093 内部给两个 M3 核各一个 WatchDog。M3 核对应 WatchDog 挂死，输出复位指示，复位非 HOST 子系统，指示信号可屏蔽；复位指示信号到实际复位需做延时处理，延时从 0ms~10s 可配 (建议值 0ms, 10ms, 100ms, 500ms, 1s, 2s, 4s, 10s)，默认延时 500ms。

具体操作方式，请参见 [32 WatchDog](#) 章节。

2.4.7 核间通信

业务核 A55 与 IO 协处理器 M3 通过 DDR 或 SRAM 进行信息交换，信息交换在非安全区，除此之外提供可读写寄存器配合核间通信，通过这些寄存器实现互斥访问及信息交互；提供可读写寄存器用于核间中断机制通信，中断可屏蔽。

具体寄存器地址，请参见《Hi3093 寄存器手册》。

须知

核在发出锁的申请以后，自动排队等待获取锁的控制权限。安全核 M3 可同时占用多个锁，如果占用多个锁而且处理时间较长，则释放锁的时间也很长。在安全 M3 占完所有锁资源并处理时间很长一直不释放的极端情况下，会导致 A55 无法获得锁资源。

2.5 工作方式

2.5.1 安全启动

安全只从 ROM 启动，提供寄存器供软件记录校验状态。

安全启动需要 32KB ROM 和 256KB 的 SRAM。



2.5.2 协处理器 M3 地址空间列表

从 M3 协处理器视角看到的芯片内部的地址空间映射如表 2-2 所示。

说明

表 2-2 中未列出的地址范围为保留。此外，对于 M3，0xE0000000~0xE00FFFFFF 为内部私有总线的地址空间，所以与该部分重叠的 DDR 地址空间对 M3 不可见。

表2-2 协处理器 M3 地址空间映射表

起始地址	结束地址	功能	大小
0x0000_0000	0x07FF_FFFF	<ul style="list-style-type: none">local busSFCDDR	128MB
0x0800_0000	0x080F_FFFF	Reserved	1MB
0x0810_0000	0x081F_FFFF	SRAM	128KB
0x0820_0000	0x082F_FFFF	Reserved	1MB
0x0830_0000	0x083F_FFFF	Reserved	1MB
0x0840_0000	0x084F_FFFF	Reserved	1MB
0x0850_0000	0x085F_FFFF	Reserved	1MB
0x0860_0000	0x0860_FFFF	SFC1_REG	64KB
0x0861_0000	0x0861_FFFF	SDIO_REG	64KB
0x0862_0000	0x0862_FFFF	Reserved	64KB
0x0863_0000	0x0863_FFFF	EMMC_REG	64KB
0x0864_0000	0x0864_FFFF	SFC0_REG	64KB
0x0865_0000	0x086F_FFFF	Reserved	736KB
0x0870_0000	0x0870_1FFF	Timer01_REG	8KB
0x0870_2000	0x0870_3FFF	Timer23_REG	8KB
0x0870_4000	0x0870_5FFF	Timer45_REG	8KB
0x0870_6000	0x0870_6FFF	Reserved	4KB
0x0870_7000	0x0870_7FFF	I2C0_REG	4KB
0x0870_8000	0x0870_8FFF	I2C1_REG	4KB
0x0870_9000	0x0870_9FFF	I2C2_REG	4KB
0x0870_A000	0x0870_AFFF	I2C3_REG	4KB
0x0870_B000	0x0870_BFFF	I2C4_REG	4KB



起始地址	结束地址	功能	大小
0x0870_C000	0x0870_CFFF	I2C5_REG	4KB
0x0870_D000	0x0870_DFFF	I2C6_REG	4KB
0x0870_E000	0x0870_EFFF	I2C7_REG	4KB
0x0870_F000	0x0870_FFFF	Reserved	4KB
0x0876_0000	0x0876_0FFF	Reserved	4KB
0x0876_1000	0x0876_1FFF	sgpio0	4kB
0x0876_2000	0x0876_2FFF	sgpio1	4kB
0x0871_0000	0x0871_0FFF	UART2_REG	4KB
0x0871_1000	0x0871_1FFF	UART3_REG	4KB
0x0871_2000	0x0871_2FFF	Localbus_REG	4KB
0x0871_3000	0x0871_3FFF	SSP0_REG	4KB
0x0871_4000	0x0871_4FFF	DMA_REG	4KB
0x0871_5000	0x0871_5FFF	SSP1_REG	4KB
0x0871_6000	0x0871_6FFF	IPMB0_REG	4KB
0x0871_7000	0x0871_7FFF	IPMB1_REG	4KB
0x0871_8000	0x0871_8FFF	IPMB2_REG	4KB
0x0871_9000	0x0871_9FFF	IPMB3_REG	4KB
0x0871_A000	0x0871_CFFF	Reserved	12KB
0x0871_D000	0x0871_DFFF	lp_csr	4KB
0x0871_E000	0x0871_FFFF	Reserved	8KB
-	-	LPC_SUPERIO	64KB
0x0872_0000	0x0872_0FFF	Reserved	4KB
0x0872_1000	0x0872_1FFF	LPC_80_REG	4KB
0x0872_2000	0x0872_2FFF	LPC_KCS0_REG	4KB
0x0872_3000	0x0872_3FFF	LPC_KCS1_REG	4KB
0x0872_4000	0x0872_4FFF	LPC_KCS2_REG	4KB
0x0872_5000	0x0872_5FFF	LPC_KCS3_REG	4KB
0x0872_6000	0x0872_6FFF	LPC_HOSTWDG_REG	4KB
0x0872_7000	0x0872_7FFF	LPC_BT0_REG	4KB
0x0872_8000	0x0872_8FFF	LPC_BT1_REG	4KB



起始地址	结束地址	功能	大小
0x0872_9000	0x0872_9FFF	LPC_MAILBOX_REG	4KB
0x0872_A000	0x0872_AFFF	LPC_UART0_REG	4KB
0x0872_B000	0x0872_BFFF	LPC_UART1_REG	4KB
0x0872_C000	0x0872_CFFF	LPC_SOL_REG	4KB
0x0872_D000	0x0872_FFFF	Reserved	16KB
-	-	ESPI_SUPERIO	64KB
0x0873_0000	0x0873_0FFF	ESPI_REG	4KB
0x0873_1000	0x0873_1FFF	ESPI_80_REG	4KB
0x0873_2000	0x0873_2FFF	ESPI_KCS0_REG	4KB
0x0873_3000	0x0873_3FFF	ESPI_KCS1_REG	4KB
0x0873_4000	0x0873_4FFF	ESPI_KCS2_REG	4KB
0x0873_5000	0x0873_5FFF	ESPI_KCS3_REG	4KB
0x0873_6000	0x0873_6FFF	ESPI_HOSTWDG_REG	4KB
0x0873_7000	0x0873_7FFF	ESPI_BT0_REG	4KB
0x0873_8000	0x0873_8FFF	ESPI_BT1_REG	4KB
0x0873_9000	0x0873_9FFF	ESPI_MAILBOX_REG	4KB
0x0873_A000	0x0873_AFFF	ESPI_UART0_REG	4KB
0x0873_B000	0x0873_BFFF	ESPI_UART1_REG	4KB
0x0873_C000	0x0873_CFFF	ESPI_SOL_REG	4KB
0x0873_D000	0x0873_FFFF	Reserved	12KB
-	-	APB1_REG	768KB
0x0874_0000	0x0874_0FFF	SYSCTL_REG	4KB
0x0874_1000	0x0874_1FFF	CANBUS0_REG	4KB
0x0874_2000	0x0874_2FFF	CANBUS1_REG	4KB
0x0874_3000	0x0874_3FFF	UART4_REG	4KB
0x0874_4000	0x0874_4FFF	UART5_REG	4KB
0x0874_5000	0x0874_5FFF	IOCONFIG_REG	4KB
0x0874_6000	0x0874_6FFF	PECI_REG	4KB
0x0874_7000	0x0874_7FFF	MDIO_REG	4KB
0x0874_8000	0x0874_8FFF	Jtag Master0	4KB



起始地址	结束地址	功能	大小
0x0874_9000	0x0874_9FFF	Jtag Master1	4KB
0x0874_A000	0x0874_AFFF	GPIO0_REG	4KB
0x0874_B000	0x0874_BFFF	GPIO1_REG	4KB
0x0874_C000	0x0874_CFFF	GPIO2_REG	4KB
0x0874_D000	0x0874_DFFF	GPIO3_REG	4KB
0x0874_E000	0x0874_EFFF	Reserved	4KB
0x0874_F000	0x0874_FFFF	Reserved	4KB
0x0875_0000	0x0875_0FFF	I2C8_REG	4KB
0x0875_1000	0x0875_1FFF	I2C9_REG	4KB
0x0875_2000	0x0875_2FFF	I2C10_REG	4KB
0x0875_3000	0x0875_3FFF	I2C11_REG	4KB
0x0875_4000	0x0875_4FFF	I2C12_REG	4KB
0x0875_5000	0x0875_5FFF	I2C13_REG	4KB
0x0875_6000	0x0875_6FFF	I2C14_REG	4KB
0x0875_7000	0x0875_7FFF	I2C15_REG	4KB
0x0875_8000	0x0875_8FFF	Reserved	4KB
0x0875_9000	0x0875_9FFF	PWM_REG	4KB
0x0875_A000	0x0875_AFFF	BMC_RTC_REG	4KB
0x0875_B000	0x0875_BFFF	Reserved	4KB
0x0875_C000	0x0875_CFFF	Reserved	4KB
0x0875_D000	0x0875_DFFF	UART6_REG	4KB
0x0875_E000	0x0875_EFFF	UART7_REG	4KB
0x0875_F000	0x0876_2FFF	Reserved	2*4KB
0x0876_3000	0x0876_3FFF	watchdog0	4KB
0x0876_4000	0x0876_4FFF	watchdog1	4KB
0x0876_5000	0x0876_5FFF	watchdog2	4KB
0x0876_6000	0x0876_6FFF	watchdog3	4KB
0x0876_7000	0x0876_7FFF	watchdog4	4KB
0x0876_8000	0x0876_8FFF	watchdog5	4KB
0x0876_9000	0x0876_9FFF	ADC_CTRL0	4KB



起始地址	结束地址	功能	大小
0x0876_A000	0x0876_AFFF	ADC_CTRL1	4KB
0x0876_B000	0x0876_BFFF	PECI_SINGLE	4KB
0x0876_C000	0x0876_CFFF	TRNG1	4KB
0 x0876_D000	0x0876_FFFF	Reserved	3*4KB
0x0877_0000	0x0877_FFFF	A55_DJTAG_MSTR	64KB
0x0878_0000	0x0878_FFFF	DDRC_REG	16*4KB
0x0879_0000	0x087F_FFFF	Reserved	112*4KB
-	-	HOST_AXI	68MB
0x0880_0000	0x088F_FFFF	USB0_ahb_REG	1MB
0x0890_0000	0x089F_FFFF	USB1_ahb_REG	1MB
0x08A0_0000	0x08AF_FFFF	USB2_ahb_REG	1MB
0x08B0_0000	0x08B0_1FFF	USB0_apb_REG	8KB
0x08B0_2000	0x08B0_3FFF	USB1_apb_REG	8KB
0x08B0_4000	0x08B0_5FFF	USB2_apb_REG	8KB
0x08B0_6000	0x08B0_7FFF	Reserved	8kB
0x08B0_8000	0x08B0_BFFF	Reserved	16KB
0x08B0_C000	0x08B0_FFFF	VCE_REG	16KB
0x08B1_0000	0x08B1_FFFF	PCIe0 Reg	64KB
0x08B2_0000	0x08B2_0FFF	HOST_REG	4KB
0x08B2_1000	0x08B2_1FFF	PCIE0 MISC	4KB
0x08B2_2000	0x08BF_FFFF	Reserved	222*4K
-	-	HS_AXI	
0x0CC0_0000	0x0CCF_FFFF	USB3_ahb_REG	1MB
0x0CD0_0000	0x0CD0_0FFF	GMAC0_REG	4KB
0x0CD0_1000	0x0CD0_1FFF	GMAC1_REG	4KB
0x0CD0_2000	0x0CD0_2FFF	GMAC2_REG	4KB
0x0CD0_3000	0x0CD0_3FFF	GMAC3_REG	4KB
0x0CD0_4000	0x0CD0_4FFF	Reserved	4KB
0x0CD0_5000	0x0CD0_5FFF	Reserved	4KB
0x0CD0_6000	0x0CD0_7FFF	USB3_apb_REG(apb_sel2)	8KB



起始地址	结束地址	功能	大小
0x0CD0_8000	0x0CD0_9FFF	Reserved	8KB
0x0CD0_A000	0x0CD0_BFFF	Reserved	8KB
0x0CD0_C000	0x0CD0_CFFF	HP_WRAP_REG(apb_sel3)	4KB
0x0CD0_D000	0x0CD0_DFFF	Reserved	4KB
0x0CD0_E000	0x0CD0_EFFF	PCIE1 MISC	4KB
0x0CD0_F000	0x0CD0_FFFF	Reserved	4KB
0x0CD1_0000	0x0CD1_FFFF	PCIE1 Reg	64KB
0x0CD2_0000	0x0CD2_0FFF	HOSTRTC REG (apb_sel0)	4KB
0x0CD2_1000	0x0CD2_FFFF	Reserved	60KB
0x0CD3_0000	0x0CD3_0FFF	MCTP REG (apb_sel1)	4KB
0x0CD3_1000	0x0CDF_FFFF	Reserved	828KB
0x0CE0_0000	0x0CE3_FFFF	SERDES marco0	256KB
0x0CE4_0000	0x0CE7_FFFF	SERDES marco1	256KB
0x0CE8_0000	0x0CEB_FFFF	Reserved	256KB
0x0CEC_0000	0x0CEF_FFFF	Reserved	256KB
0x0CDF_0000	0x13FF_FFFF	Reserved	113MB
-	-	A5X_CORE	1MB
0x1400_0000	0x1400_0FFF	A5X_CSR	4kB
0x1400_1000	0x1400_1FFF	SHAPER REG	4kB
0x1400_2000	0x1400_3FFF	sys_cnts	8KB
0x1400_4000	0x1400_4FFF	A5X_ROM	4kB
0x1400_5000	0x1400_5FFF	LOCAL_CORE0	4kB
0x1400_6000	0x1400_6FFF	LOCAL_CORE1	4kB
0x1400_7000	0x1400_7FFF	LOCAL_CORE2	4kB
0x1400_8000	0x1400_8FFF	LOCAL_CORE3	4kB
0x1400_9000	0x1400_9FFF	PMCTRL	4KB
0x1400_A000	0x140F_FFFF	Reserved	-
0x1410_0000	0x143F_FFFF	Reserved	-
0x1440_0000	0x147F_FFFF	DAP_LITE	4MB
0x1480_0000	0x17FF_FFFF	Reserved	-



起始地址	结束地址	功能	大小
0x1800_0000	0x19FF_FFFF	GPU_DFX	32M
0x1A00_0000	0x1BFF_FFFF	PCIe0 Mem	32M
0x1C00_0000	0x23FF_FFFF	PCIe1 Mem	128M
0x2400_0000	0x25FF_FFFF	GIC	32MB
0x2600_0000	0x2FFF_FFFF	reserved	-
0x3000_0000	0x3FFF_FFFF	localbus_Mem0	256MB
0x4000_0000	0x4FFF_FFFF	localbus_Mem1	256MB
0x5000_0000	0x57FF_FFFF	localbus_Mem2	128MB
0x5800_0000	0x5FFF_FFFF	localbus_Mem2	128MB
0x6000_0000	0x67FF_FFFF	SFC0 片选 0	128MB
0x6800_0000	0x6FFF_FFFF	SFC0 片选 1	128MB
0x7000_0000	0x77FF_FFFF	SFC1 片选 0	128MB
0x7800_0000	0x7FFF_FFFF	SFC1 片选 1	128MB
0x8000_0000	0xFFFF_FFFF	DDR	2GB



3 ARM-A55

3.1 概述

Hi3093 提供 4 个 A55 核，其中两个 A55 用于硬件故障监控、全局业务计算、KVM 虚拟媒体计算和外部 IO 接口计算；另外两个 A55 用于智能运维中的推理计算。

A55 子系统由 4 个 A55 Core 和 1 个 DSU (DynamIQ Shared Unit) 组成，也称为 FCM (Flexible Cluster Microarchitechture system)。

3.2 特性

A55 Core 的特性：

- 完整支持 ARMv8.2-AA64, 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；支持指令预取。
- 容量 32KB 的一级指令 Cache 和 32KB 的一级数据 Cache。
- 容量 128KB 的二级 Cache。
- L1、L2 cache 和 TLB 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。
- 数据和指令支持小端模式。
- 最高频率 1.0GHz。频率可静态配置。
- 基于单线程结构。
- 支持 4 核 64 位 A55 处理器。
- 支持 CAS 指令。
- 提供 NEON&FPU 模块，支持 Crpto 加速/Dot Product 指令集、支持浮点运算 1.0Gflops/Core。
- AI 业务场景，提供不低于两个 A55 核的计算能力，以及 4G 内存，支持智能管理所需算力需求。

DSU 的特性：

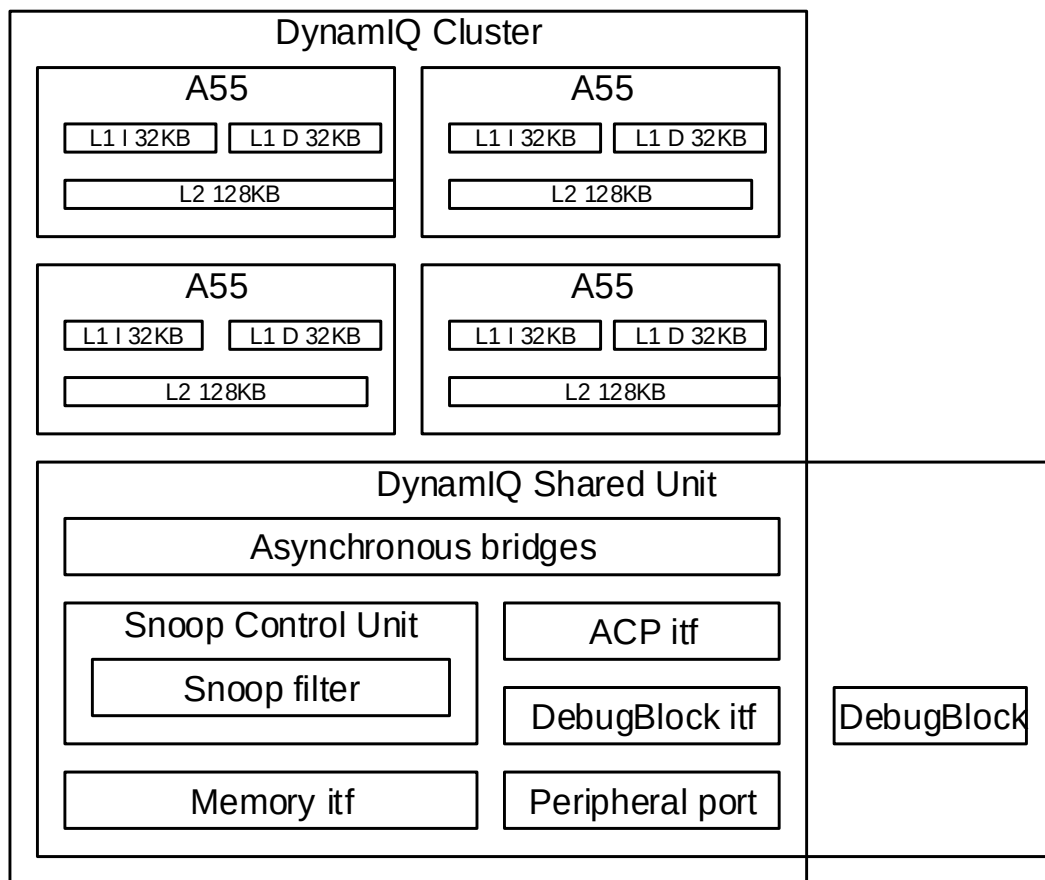
- 支持 1 个 ACE memory 接口。
- 支持 16T 物理地址空间（44 位物理地址）。
- 支持 Armv8.2-A debug 逻辑。
- 支持 RAS 特性。
- SCU RAM 支持 ECC 校验。
- 支持外接 DAPLite 调试部件。

3.3 功能描述

3.3.1 FCM 结构

DynamiQ Cluster 和 DebugBlock 一起，构成了 Hi3093 的主处理器单元，它的结构如图 3-1 所示。

图3-1 FCM 结构框图



没有 L3 cache, 4 个 A55 之间的一致性, 由 Snoop Controller Unit 维护。

3.3.2 Memory 接口

DSU 通过 ACE 接口访问外部 memory, 包含 Flash 和 DDR 等, 也可以访问系统中的所有 IP 寄存器空间, A55 通过这个接口可以访问整个系统地址空间。具体详见地址空间分配表。

3.3.3 GIC CPU 接口

DSU 支持 GICv3 架构, 包含 1 对 stream 接口。外部 GIC 通过 stream 接口, 将 interrupt 发给 A55 Core。DSU 保留了 nFIQ、nIRQ 等接口, 但是 Hi3093 不支持 GIC 和 DSU 通过这组接口传递 interrupt 信息。

A55 每个 Core 各包含 1 组 GIC CPU 接口系统寄存器, 分 3 类:

- ICC: GIC CPU interface System register



- ICV: Virtual GIC CPU interface System register
- ICH: Virtual interface control System register

3.3.4 系统计数器

系统计数器部署在 DynamIQ Cluster 外部，提供统一的系统时间，只在上电时复位，只能通过 memory-mapped 方式访问。

系统计数器提供 2 组访问接口：安全软件通过读写接口访问系统计数器，非安全软件通过只读接口访问系统计数器。

采用 64 位系统计数器，如果采用 PERIPHCLK 的 16 分频时钟，计数时长： $2^{64}/18.75/10^6/3600/24/366=31111$ 年，ARM 架构要求至少 40 年，满足要求。

须知

Hi3093 按 Core 复位场景下，system counter 值不会清零，芯片长时间运行后，system counter 会持续累加，软件中需要使用 system counter 值时（包括 CNTPCT 和 CNTVCT 这些和 system counter 同步的计数器）需要注意影响，避免使用 system counter 绝对值。

3.3.5 通用计时器

A55 每个 Core 有 5 个 timer:

- EL1 Non-Secure physical timer
- EL2 Hypervisor physical timer
- EL3 Secure physical timer
- Virtual timer
- Hypervisor virtual timer

各产生 1 个低有效的 PPI interrupt，送给 GIC，用于事件的调度。



3.4 工作方式

3.4.1 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

3.4.2 A55 地址空间列表

A55 地址空间列表同协处理器 M3 地址空间列表，请参见 [2.5.2 协处理器 M3 地址空间列表](#) 章节。

3.4.3 工作状态

由 ARMv8-A 架构决定，Cortex-A55 MP 的工作状态分为 4 个维度：

- 架构状态。Cortex-A55 支持所有模式下的 AArch32 和 AArch64。
- 指令集状态。由指令集支持决定，支持 A32、T32 和 A64 几种状态。
- 异常等级状态。分为 4 种，在后文详细描述。
- 安全状态。非安全和安全两种。

3.4.4 异常等级

ARMv8-A 异常等级如[表 3-1](#) 所示。

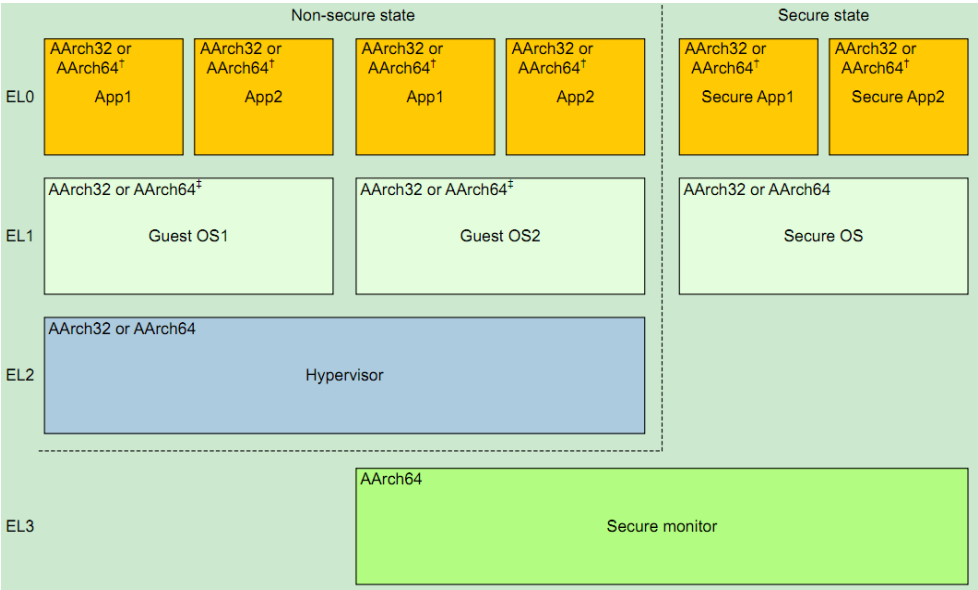


表3-1 ARMv8-A 的异常等级

处理器异常等级	描述
EL0	用户程序执行模式，非特权级，安全/非安全两套。
EL1	操作系统运行模式，特权级，安全/非安全两套。
EL2	用于虚拟化扩展功能，仅存在于非安全。
EL3	用于切换安全和非安全世界。

以上各种运行模式之间的对应关系如图 3-2 所示。

图3-2 运行模式关系图





4 GIC

4.1 概述

GIC (Generic Interrupt Controller) 模块是中断管理单元，遵循 ARM 的 GICv3 架构，主要包括如下功能：SGI, PPI 和 SPI 的使能、路由、优先级配置以及 ARM TrustZone 技术。

4.2 特性

GIC-600 的特性包含：

- 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。
- 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。
- 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。
- 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。
- 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。
- 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。
- 每个核提供内部器件的 PPI 中断，各核之间 PPI 中断相互独立。



- 支持 16 个 PPI。
- 系统支持最多 256 个 SPI 中断。
- 系统支持 16 个 SGI。
- 支持 TrustZone 扩展，输入中断可按照 Secure/Non-Secure 分组。
- 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（IRM 取 1）。
- ARE 固定为 1，SGI 和 PPI 只能在 GICR 寄存器空间编程。
- 支持 GIC 总线地址位宽 32bit。
- 支持通过中断唤醒低功耗模式的核。
- 片内 A55 可通过总线访问 GIC 寄存器。
- 20 路芯片外部中断，均支持管脚复用。此外，若其他管脚复用为 GPIO，也可以作为外部中断源。
- 外部中断触发方式可配置。
- 外部中断优先级可配。
- 芯片所有错误中断可以通过写寄存器或其他手段进行模拟注入。

4.3 功能描述

4.3.1 关键术语

表4-1 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。
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。



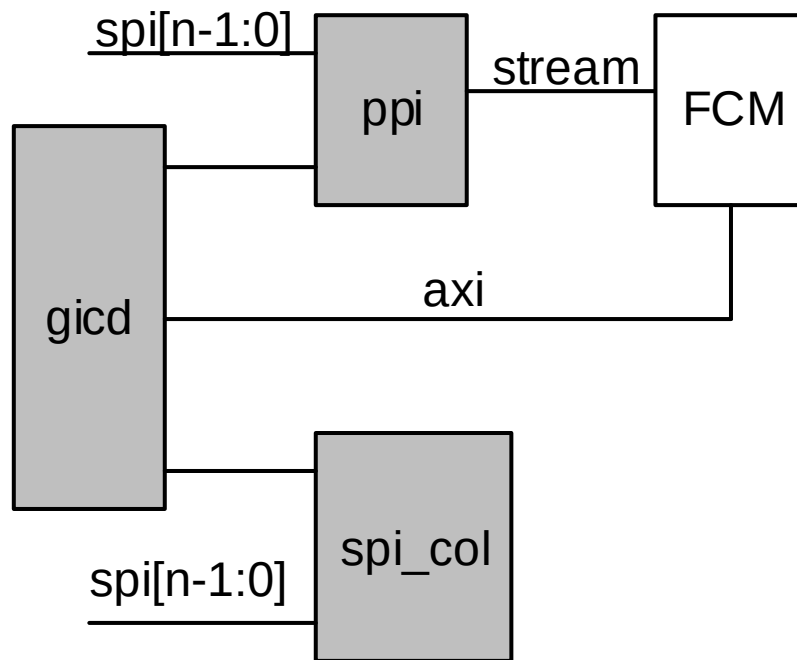
Terms	Description
SPI	Shared Peripheral Interrupt, 共享外设中断。
EOI	End of Interrupt, 中断结束。
WireInterrupt	通过物理硬连线直接发出的中断信号。
Cluster	多个 CPU Core 组成的, 复用相同 L2 Cache 的硬件集合。
Device	CPU Core 以外能产生中断请求的周边设备。
TargetList	表示多个中断目标 CPU Core 的一个合集, 可能采用 OneHot 编码或者多组地址合并表示多个目标 CPU Core。
CPU Core	中央处理器核, 处理器中用于指令解码、执行的核心硬件电路集合单元, 这里特指 A55。
Affinity Routing	亲和路由。
ARE	Affinity Routing Enable, 亲和路由使能。
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。
Distributor	分发器。
Re-Distributor	重分发器。

4.3.2 应用场景

Hi3093 的 GIC-600 包含 3 个部分 (如图 4-1 黄色的 block), 它接收 SGI, PPI 和 SPI, 通过 FCM stream 接口上报给 FCM 内的 Core。



图4-1 GIC 结构示意图



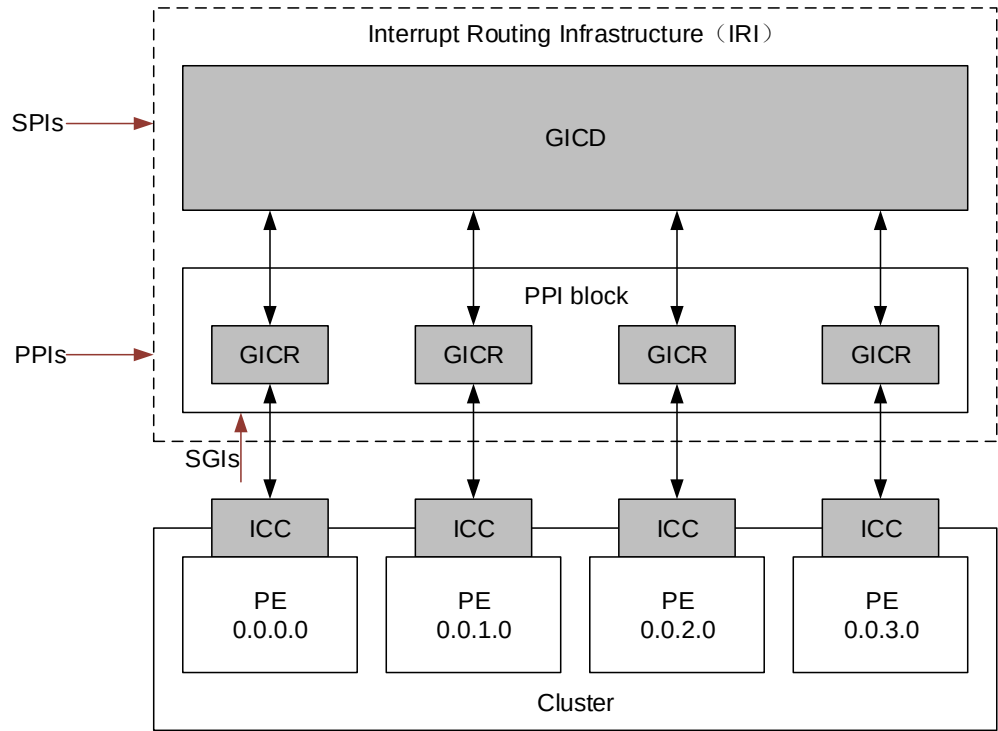
说明

ppi block 就是 redistributor，下文简称 gicr。



4.3.3 中断流程

图4-2 中断传递流程图



中断的传递流程如图 4-2 所示，每个中断，在其生命周期中，都要经过 6 个步骤。

步骤 1 中断产生：外设产生 SPIs 中断，通过线中断或消息中断，发送给 GICD。

PE 中特定事件产生 PPIs 线中断，发送给 GICR。

软件产生 SGIs 消息中断，发送给 GICR。

表4-2 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。	将软件抽象成一种外设，同普通外设一样上报中断。
Spi	外设中断线连接到 GICD。	大部分外设中断使用 SPI。



GICD 接收 中断类型	中断产生方法	说明
PPI	CPU Core 中断线连接到 GICD。	PPI 的源头是 Core 的私有 外设，如 timer 等。

步骤 2 中断分发：IRI 执行中断分组，中断优先级管理，并将中断分发给目标 PE 对应的 CPU Interface。

表4-3 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 寄存器时配置中断路由。	Sgi 支持 Targeted list 路由算法。
Spi	Spi 从软件配置的 GICD_IROUTER 寄存器获取路由配置的方式。	Spi 支持 aff, 1/N 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。

步骤 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。

步骤 4 PE 确认中断：PE 读 CPU Interface IAR 寄存器，读取中断 ID，告知已收到中断。

步骤 5 Priority drop：PE 写 CPU Interface EOI 寄存器，告知中断已处理完成。

步骤 6 Deactivation：PE 写 CPU Interface DIR 寄存器，触发 IRI 维护的中断状态机有 active 变为 inactive，中断处理完成。

----结束



说明

- 当 EOImode 为 0 时，步骤 5 的动作包含了步骤 5 和步骤 6 两部分含义，只需要执行到步骤 5。
- 当 EOImode 为 1 时，步骤 5 和步骤 6 的动作是分开且都需要先后执行的。

4.3.4 GICD 亲和路由

亲和性路由有 4 个层次：

- AL3，固定 0。
- AL2，该值等于系统 cluster 编号，Hi3093 该值只能取 0。
- AL1，该值等 CORE 编号，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，AL1 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。
 - 在 SGI 亲和路由配置中，AL1 等于 TargetList，此处的 TargetList 为 OneHot 编码格式，指示中断被路由到哪几个 CPU Core。
- AL0，线程 ID，Hi3093 只能配 0。

表4-4 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。	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。
SGI 路由	中断路由到亲和路由配置 “a.b.c.{ TargetList } ”指定的 CPU Core。	中断路由到系统中除自己以外的 所有 CPU Core。

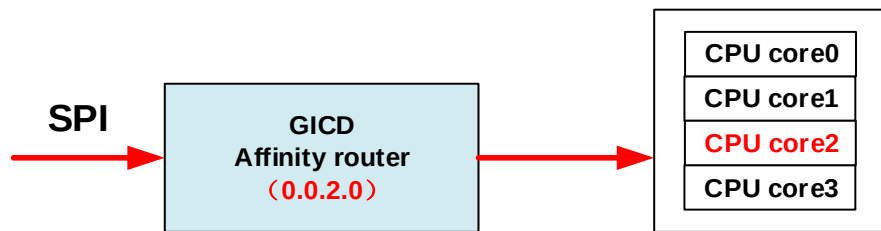
SPI 基于 GICD_IROUTER<32-1019>.IRM 确定路由算法，SGI 基于
SICC_SGI0R_EL1.IRM 和 ICC_SGI1R_EL1.IRM 确定路由算法。

4.3.4.1 1/N SPI 亲和路由

IRM 为 0 模式

假设该 SPI 中断亲和路由配置为 “0.0.2.0”，SPI 中断亲和性路由上报流程图如图 4-3 所示。

图4-3 1/N SPI 中断亲和性路由示例 (IRM=0)

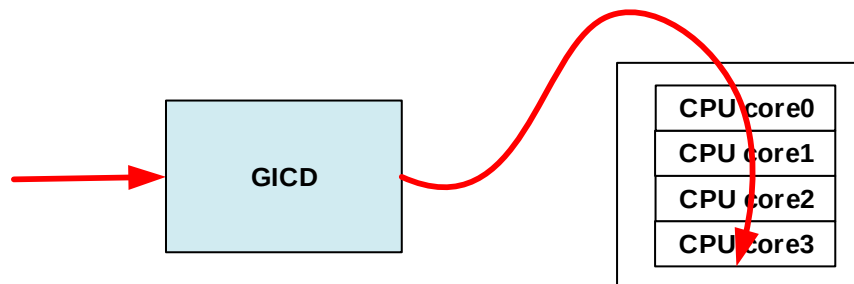


如图 4-3 红色箭头路由路径，根据亲和路由配置 “0.0.2.0” SPIs 会被路由给唯一目标 CPU Core。

IRM 为 1 模式

1. 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，如图 4-4 所示。

图4-4 1/N SPI 中断亲和性路由示例 (IRM=1)



2. 中断上报到 GICD 后，会按 Core ID 从小到大的顺序，遍历 CPU Core，寻找空闲的 CPU Core 并上报。

3. 当所有 CPU Core 均被遍历，且没有空闲的 Core 时，随机向一个 Core 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。

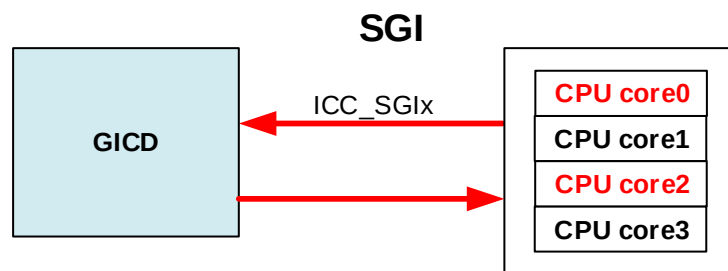
4.3.4.2 SGI 亲和路由

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 PE。TargetList 中，可以包含发起 SGI 的 PE。

SGI 中断亲和性路由上报流程图，如图 4-5 所示。

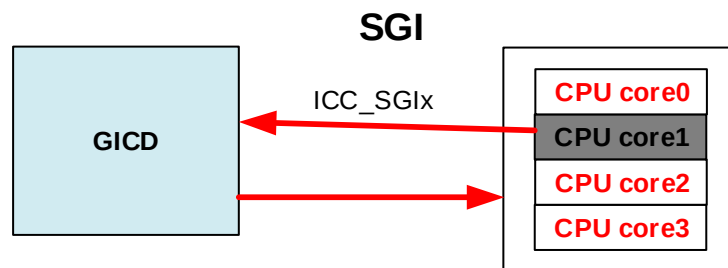
图4-5 SGI 中断亲和性路由示例



IRM 为 1 模式

该模式中中断会广播给除本 CPU Core 外的所有 CPU Core。

图4-6 SGI 中断广播路由示例





4.3.5 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 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。

4.3.6 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 4-5 所示。

表4-5 安全软件视角中断优先级配置

Type	Priority Register	Secure Priority	Non-Secure Priority
SGI PPI	GICR_IPRIORITY Rn	{IPRIORITYRn[7:3],3'b0}	{IPRIORITYRn[7:3],3'b0}
SPI	GICD_IPRIORITY Rn	{IPRIORITYRn[7:3],3'b0}	{IPRIORITYRn[7:3],3'b0}

说明

- 寄存器的值和安全视角看到的值一致。
- ARM 推荐安全写 non-secure 优先级时，IPRIORITYRn[7]等于 1。

目的：保证非安全中断的优先级落在级别较低的半个空间里。

表4-6 非安全软件视角中断优先级配置

Type	Priority Register	Secure Priority	Non-Secure Priority
SGI PPI	GICR_IPRIORITYRn	8'b0	{1'b1,IPRIORITYRn[7:4],3'b0}
SPI	GICD_IPRIORITYRn	8'b0	{1'b1,IPRIORITYRn[7:4],3'b0}

说明

- 非安全读 Non-Secure 优先级时，只读出 GICR(D)_PRIORITY[7:4]，低 4 位得 0。
- 非安全写 Non-Secure 优先级时，先右移 1 位，再高位补 0，最后存到域段里。

目的：保证非安全中断的优先级落在级别较低的半个空间里。



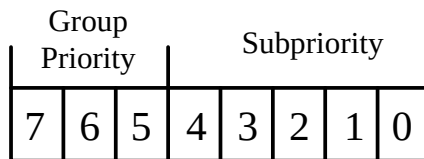
GIC 中断调度原则：

- 中断调度遵循时间优先，先到的中断先报。
- 相同时刻的中断遵循优先级调度，数字越小优先级越高。
- 中断优先级调度是基于安全视角的中断配置进行调度。
- 高于 GICC 下发的 PMR 的优先级门限的中断才能被调度。

4.3.7 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。可以利用 GICC_BPR.Binary point 配置 Group priority 和 Subpriority 组的位域，如图 4-7 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4-7 中断优先级结构示意图



Group Priority 的位数由寄存器 GICC_BPR、GICC_ABPR（非安全）、GICV_BPR（虚拟）可配。

4.3.8 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。
- 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 0。否则不可预期。
- 上面的步骤必须被遵守，如果在 GICR_WAKER.ProcessorSleep 置 1 后，下电步骤被中止。那么正常的上电或者复位可能不会发生。

4.3.9 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。

4.3.10 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。

Distributor 初始化步骤如下：

步骤 1 配置 GICR_PWRR 寄存器的 2 个字段：RDAG 等与 1，RDPD 等于 0。表示 Redistributor 已经上电，可以被访问了。

步骤 2 配置 Distributor 接口。

- 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。
- 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。
- 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。
- 配置寄存器 GICR_ICFGRn 初始化 PPIs 为电平敏感或者是边沿触发。
- 配置寄存器 GICD_ICFGRn 初始化 SPIs 为电平敏感或者是边沿触发。
- 配置寄存器 GICR_IPRIORITYRn 设置 SGI/PPI 中断的优先级。
- 配置寄存器 GICD_IPRIORITYRn 设置 SPI 中断的优先级。
- 配置寄存器 GICD_IROUTERn 设置 1/N SPIs 的目标处理器。

步骤 3 配置物理 GICC 接口。

- 配置寄存器 ICC_PMR 设置屏蔽优先级。



- 配置寄存器 ICC_BPR、ICC_ABPR 设置安全中断和非安全中断的 Group Priority。

步骤 4 使能 Distributor 接口。

配置寄存器打开相关中断 ID 使能位。

- 配置寄存器 GICD_ISENABLERn 使能 SPI 中断。
- 配置寄存器 GICR_ISENABLERn 使能 SGI/PPI 中断。

须知

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。

- 配置寄存器 GICD_CTLR.EnableGrp1、GICD_CTLR.EnableGrp1 使能 Group1 和 Group0 中断。

步骤 5 使能 GICC 接口。

配置寄存器 ICC_CTLR 使能处理器接口。

----结束

4.3.11 GICD 中断路由配置流程

4.3.11.1 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 对应路由寄存器的 1bit，配置步骤如下：

步骤 1 配置路由模式寄存器 GICD_IROUTERn.IRM 选择 SPI 中断 1/1 或 1/N 路由模式。

步骤 2 SGI 路由模式是通过 ICC_SGIXR_ELx 中的 IRM 进行选择的。

----结束

4.3.11.2 改变 SPI 亲和路由设置流程

软件必须按照以下步骤顺序执行：

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



5 系统控制器

5.1 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、软中断等），完成对外设的某些功能的配置与状态读取查询。

5.2 特性

系统控制器具有以下特性：

- 控制并监控系统的启动模式。
- 提供系统时钟控制和状态查询。
- 提供对外设时钟门控的控制、复位控制和查询。
- 提供通用外设控制寄存器对外设进行各种控制。
- 提供芯片的标识寄存器。
- 提供各个外设的功能配置接口。
- 启动所用 Flash 需寄存器指示。
- 启动的 Flash 区需寄存器指示（双 bios 指示）。
- 提供寄存器供软件启动结束填写下次启动是否切换 bios 区，除 poweron 复位外状态不丢失供 debug 使用。
- 提供寄存器供软件填写安全校验过程，除 poweron 复位外状态不丢失供 debug 使用；可以指示是否安全启动过程，有寄存器可供软件记录状态。



- A55 和 M3 协处理器之间的通信方式用共享内存的方式通信即可，共享内存的地址和大小可以配置（速率要求 10MB/s 即可）。
- 芯片内 A55 和 M3 协处理器之间实现内存互斥访问的机制。
- A55 和 M3 协处理器之间报文发送和接收可由查询寄存器或者中断方式通知，中断打关闭状态可以配置。
- 支持 16 位配置 STRAP 管脚，与其他数据管脚复用，逻辑使用 10bit。
- 启动 Flash 介质、双 bios 使能、双 bios 空间大小由 strap 管脚决定。
- 所有 STRAP 管脚值提供寄存器查询。
- 支持 48bits BootCfg 管脚，提供寄存器查询。
- 提供呼吸灯指示，指示芯片内 Core 是否挂死。
- 任意 WatchDog 挂死输出固定高电平。系统正常时输出方波，周期为 1s。
- 支持 SPI Flash 单片选双 BOOT 启动，启动空间大小为 1M/2M 管脚可配置。
- 支持 Local Bus 单片选双 BOOT 启动，启动空间大小为 1M/2M 管脚可配置。
- ROM 代码支持基于 eMMC 实现双 BOOT 引导，空间上支持两个区。
- A55 启动地址可以由可信启动安全核配置。
- A55 启动可由可信启动安全核控制复位释放。
- 非安全启动状态下，支持从不同存储介质（ROM+eMMC/SPI Flash/LocalBUS Flash）启动。
- 支持 SecureCore 先启动，然后再启动 A55。
- 提供 64bit 寄存器，安全核/A5X 均可读可写，整芯片复位。

5.3 功能描述

5.3.1 系统运行模式

Hi3093 系统可以工作在以下 2 种工作模式：

- NORMAL 模式

系统初始化完成后切换到 NORMAL 模式下。在此模式下，由片内 CPLL 的输出时钟驱动，处理器核与总线频率之比为 2:1。

- SLOW 模式



SLOW 模式是一种慢速模式。芯片上电后进入此模式，系统由外接晶振时钟 25MHz 驱动，此时系统处于初始化模式，只有部分片内外设可以工作，如系统控制器、Timer、LocalBus 等。所有对高速时钟有要求的模块在此时钟下无法工作，如 DDRC 等。

5.3.2 子系统外设控制

整芯片一共 5 个子系统，构成最小系统的 A55 子系统和 DDR 子系统的外部控制在一个模块中，其余三个子系统由独立的模块控制子系统外部配置。

5.3.2.1 A55 子系统控制

控制范围包含 A55 和 GIC。

5.3.2.2 DDR 子系统控制

控制范围包含 DDRC 和 DDR PHY。

系统控制寄存器 SYSCTRL 有详细描述，涵盖最小系统的外部配置，主要包含：

- Chip ID
- 芯片启动管脚配置状态：Flash 指定、安全选择、双 BIOS 使能、BIOS 区间大小、Flash 地址长度模式、启动偏移地址、启动接口位宽、JTAG 模式选择
- Bootcfg 初始化信息
- 启动定义控制
- 启动完成标识
- M3 启动地址
- 时钟分频控制
- PLL 配置
- PLL 锁定状态
- 软复位控制：芯片全局软复位、HOST 域软复位、软复位、安全区软复位、非安全区软复位、PLL 软复位
- 看门狗软复位控制：WDG0/1/2/3/4/5/6
- 复位历史记录：WDG0/1/2/3/4/5/6、HOST、非 HOST 域
- 复位历史记录清除
- DDR 外部控制
- DDR 状态



- Boot 启动记录
- A55 和 M3 之间通信寄存器

5.3.2.3 高速外设 (HP) 子系统控制

控制范围包含 SGMII/RMII, USB3.0, USB2.0。

系统控制寄存器 HP_SUB_Regs 有详细描述, 主要包含:

- 软复位 (USB0/1/2/3、GMAC0/1/2/3)
- 时钟使能
- 时钟选择
- 子系统内部桥的配置
- 子系统内部低功耗使能控制

5.3.2.4 低速外设 (LP) 子系统控制

控制范围包含 M3, 安全引擎, DMAC, SFC, Localbus, eMMC, ROM, RAM, UART, I2C/SMBUS, C80, KCS, mailbox, BT, SOL, WATHDOG, SPI, SGPIO, Canbus, Sysctrl, PECL, MDIO, SDIO, GPIO, RTC, PWM, 时钟、复位, IOCONFIG 复用配置寄存器, JLC, HOSTUART, AD。

系统控制寄存器 LP_SUBSYS 有详细描述, 主要包含:

- 高位地址选择 (SDIO/eMMC/SF)
- 软复位控制 (LPC、GPIO0/1/2/3/4、I2C0-15、SPI0/1、SFC0/1、localbus、SDIO、M3、外部 WatchDog、JLC、SGPIO0/1、UART2-7、MDIO、IPMB0/1/2/3、RTC、SOL、HOSTUART0/1/2、mailbox、C80、BT0/1、KCS0/1/2/3、canbus0/1、AD、PWM)
- SPI 外部配置控制
- MDIO 控制
- 中断电平控制
- M3 子系统控制
- UART 全互联控制
- GPIO 和 PWM 复位选择
- 时钟使能
- I2C 挂死恢复控制



- 时钟分频系数设置 (SFC0/1、emmc、SDIO)
- DMAC 输入矩阵的外设选择控制
- eMMC 的外部功能控制、时钟软复位控制
- 子系统内部低功耗使能控制

5.3.2.5 HOST 子系统控制

控制范围包含 PCIE0/1, USB2.0, GPU, VCE

系统控制寄存器 HOST_SUBSYS 有详细描述, 主要包含:

- 时钟门控: PCIE0/1、VCE、GPU
- 复位控制: PCIE0/1、VCE、GPU
- GPU 外部配置控制: DFX
- PCIE0/1 外部配置控制
- VCE 外部配置控制
- 子系统内部桥的配置

5.4 工作方式

5.4.1 启动 PAD

系统启动相关的一些初始配置需要通过管脚上下拉来给出固定值。在系统启动复位撤销前, 锁定到内部; 复位撤销后, PAD 的功能是默认 PAD 名称对应的功能。复位前 PAD 采样后的数值保存在系统控制器的 STRAP_STATUS_REG 寄存器, 具体启动 PAD 描述请参见《Hi3093 硬件设计指南》。

5.4.2 启动真值表

Hi3093 支持从外接 SFC、Localbus 接口 Flash 启动, 也支持 ROM 引导 eMMC 启动。EFUSE 里 NS-FORBID 位以及启动相关 STRAP 管脚控制启动核及启动介质, 真值表如表 5-1 所示。



表5-1 启动真值表

NS-FORBID[31:0]	Bootmode[1:0]	Us_boot_core	启动核	启动介质
非零	X	X	安全核	ROM
X	X	1'b1	安全核	ROM
0	2'b00	1'b0	A55	SFC Flash
0	2'b01	1'b0	A55	Localbus Flash
0	2'b10	1'b0	安全核	ROM
0	2'b11	1'b0	非法, 无法启动	非法, 无法启动

SFC Flash 启动的地址长度模式可以由 Strap 管脚 default_sfc0_addr_mode 设定, 该值为 0 时按 3Byte 地址模式启动, 为 1 时按照 4Byte 地址模式启动。

Localbus Flash 启动的地址位宽和偏远地址可以通过 strap 管脚 lb_boot_width、lb_boot_offset 控制。lb_boot_width 为 0 时启动位宽 8bit, 为 1 时启动位宽为 16bit。lb_boot_offset 为 0 是启动地址不偏移, 为 1 是启动地址偏移。

5.4.3 系统启动

系统启动流程如图 5-2 所示, 由系统控制器控制步骤如下:

- 步骤 1 系统稳定后释放复位, 根据启动安全性及启动核、启动介质释放对应处理器复位。如安全核启动, 释放其复位后执行步骤 7, 如 A55 启动释放其复位后执行步骤 2。安全启动指示由 EFUSE 指定, 启动核及介质由管脚指定, 仅非安全启动并设定 A55 启动且从多端口 SFC0 或 Localbus 启动, 整个启动流程由 A55 开始。
- 步骤 2 4 核 A55 均发出对 0 地址读命令, 硬件根据 boot 方式把 0 地址映射到不同的模块。系统支持 SFC、Local bus、ROM 启动。
- 步骤 3 判断是否需求切换到 BIOS1, 如需要则把 0~1M/2M 映射到 1M (2M) ~2M (4M)。切换到 BIOS1 的判断条件包括双 bios 使能 (STRAP_STATUS_REG 寄存器 boot_dbios 位)、双 bios 切换模式 (BOOT_DEFINE_REG 寄存器 dbios_swap_mode 位)、上次启动区记录 (BOOT_STATUS_REG 寄存器 pre_boot_area 位) 及上次启动成功标志 (BOOT_STATUS_REG 寄存器 boot_finish 位)。有 2 种情况会切换到 BIOS1 区: 双 bios 使能并双 bios 切换模式为 1; 双 bios



使能双 bios 切换模式为 0，但上次启动区为 bios0 且启动失败时进行切换。1M 或 2M 根据 strap 管脚指定。

步骤 4 A55 从对应模块取指执行代码，进行时钟配置、模块初始化。注意使用切换到 PLL 时钟需等待 PLL lock 后进行。

步骤 5 A55 配置协处理器 M3 的启动地址 (M3_BOOT_ADDR)，释放 M3 复位 (BOOT_DEFINE_REG 寄存器 m3_boot_en 位)，直到启动结束并配置启动成功标识 (BOOT_STATUS_REG 寄存器 boot_finish 位)。

步骤 6 M3 从启动地址取指开始代码执行直到启动结束。

步骤 7 安全核发出地址 0 读命令，地址 0 固定映射到 ROM，安全核开始软件代码执行。

步骤 8 安全核执行软件进行安全校验并把校验过程记录在 BOOT_STATUS_REG 寄存器。

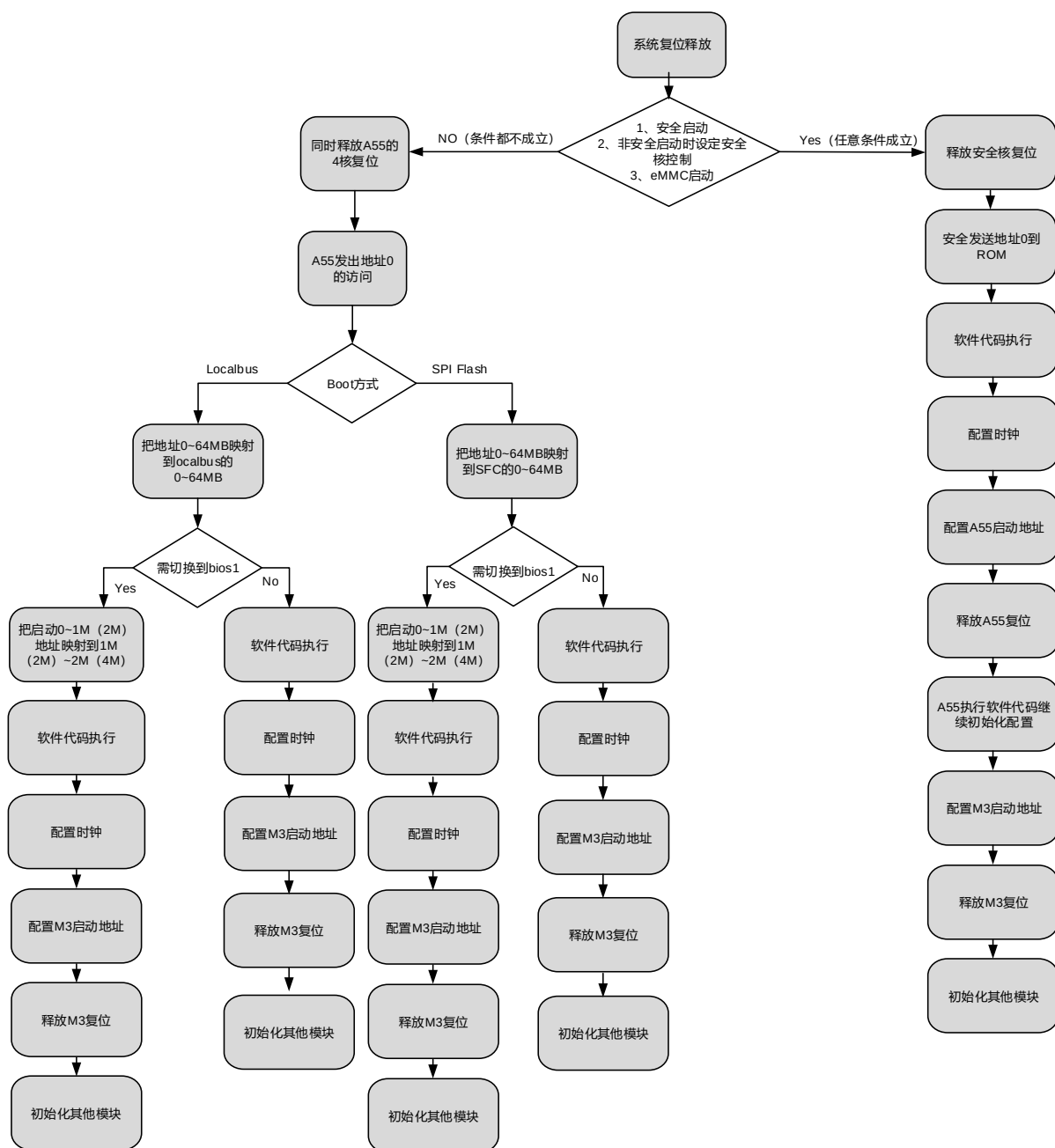
步骤 9 安全核配置时钟，配置 A55 启动地址 (在 A55 子系统中) 并是否 A55 复位 (该寄存器位于安全子系统，未包括在 sysctrl)。

步骤 10 A55 从指定起始地址开始代码执行，直到软件结束。

----结束



图5-1 系统启动流程图



5.4.4 双 BIOS 启动功能描述

双 BIOS 启动功能主要包括以下特点：



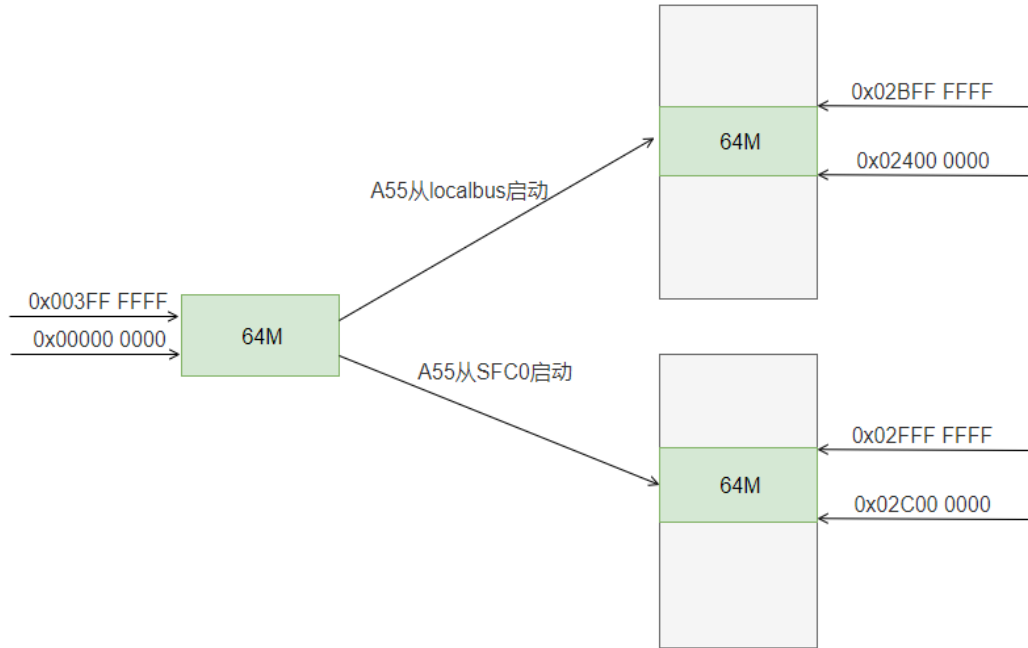
- 支持 LocalBus、SFC 双 BIOS 启动，上电默认从 BIOS0 启动，起始地址为 0x00000_0000。
- 支持 BIOS 空间大小为 1MB、2MB，通过管脚可配置。
- 支持两种切换模式：
 - 固定每次切换：在非掉电情况下，由整芯片管脚复位、片内狗复位、软件全局软复位产生的复位对 BIOS 区域进行切换，即上述复位每复位一次，BIOS 区域即从 BIOS0—>BIOS1 或 BIOS1—>BIOS0 切换一次；
 - 根据上次启动状态切换：触发条件与 1 相同，增加了判断上次启动状态逻辑，如上次启动成功则不进行切换，仅当上次启动失败进行切换。上次启动成功寄存器为软件标识，如不能正常标识该功能无法实现。
- 支持双 BIOS 功能通过管脚配置使能或禁止。
- 支持切换模式通过寄存器配置。
- 支持当前 BIOS 启动区域指示。
- 支持 BIOS 各区域启动次数统计。

5.4.5 启动地址映射说明

芯片支持 A55 或者安全核做启动核，同时支持 Localbus、SFC、eMMC 启动。在 A55 做启动核并从 Localbus 或 SFC 启动时，硬件会根据设置做对应的地址映射并硬件支持双 bios 功能。其他启动方式地址映射到安全 ROM 并且无双 bios 功能。硬件实现 A55 取指地址映射如图 5-2 所示。

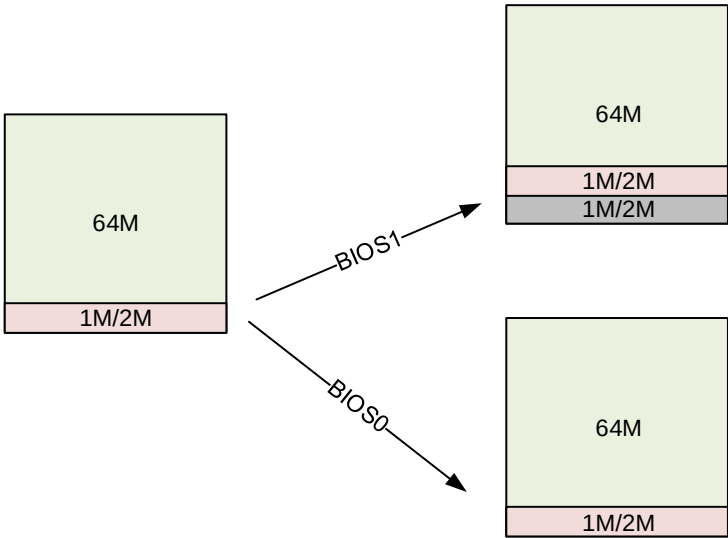


图5-2 A55 启动地址映射图



在切换到不同 Flash 空间后，对最低的 1M/2M 空间根据双 bios 设置进行 BIOS0、BIOS1 区切换。切换如图 5-3 所示。

图5-3 A55 启动双 bios 映射图





5.4.6 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：

- 当向系统复位请求寄存器 SYS_RST_REQ 进行写操作时，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。各子系统的复位控制寄存器有 HOST_RST_CFG, HP_SRST_REQ1/2, LPSC_SRST_REQ0/1/2, LPSC_EMMC_CRG_CTRL 等，具体寄存器描述参见《Hi3093 寄存器手册》。

须知

芯片软复位采用上升沿触发复位。先写 0，表示不复位；再写 1，表示复位。

5.4.7 其他功能

系统控制器支持芯片模式切换控制，内部模块的时钟门控控制，PLL 配置，以及内部模块设置，核间中断控制等请参见《Hi3093 寄存器手册》中的系统控制器的寄存器描述部分。



6 时钟

6.1 概述

芯片时钟模块接收管脚输入的时钟，产生芯片各功能模块以及管脚输出所需要的时钟，满足芯片的时钟需求。

6.2 特性

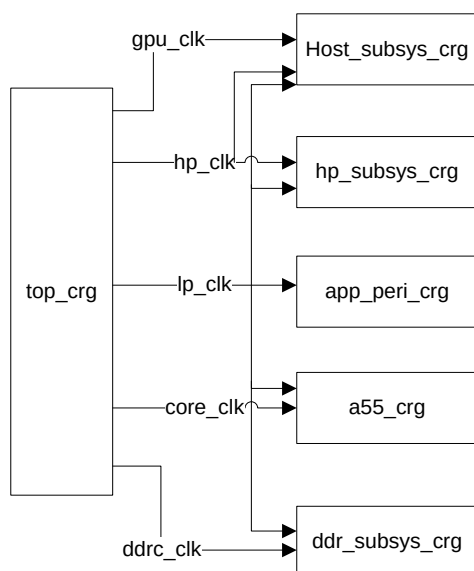
时钟管理模块对芯片时钟输入、时钟生成和控制进行统一的管理，包括：

- 提供接口时钟管理和控制，对端口时钟进行 DFT 控制和门控管理。
- 各 PLL 控制和管理，支持 PLL 频率配置，支持 PLL 时钟的 BYPASS 控制，支持 PLL 的 PD 控制。
- 提供业务模块的时钟生成和管理，支持各个业务模块的时钟分频、可测试性设计和时钟关断功能。
- 支持 Core 频率、AXI 总线频率比 2:1。
- 支持输入 SGMII 时钟：100MHz，差分， $\pm 300\text{ppm}$ ，无需支持 SSC。
- 支持输入 PCIe 时钟：100MHz，差分， $\pm 300\text{ppm}$ ，支持 SSC。
- 支持输入 NCSI 时钟：50MHz 单端，占空比 35%~65%，精度 100ppm。
- 支持输入系统时钟：25MHz 单端，占空比 45%~55%，精度 $\pm 50\text{ppm}$ 。
- 片外提供 SerDes 的参考钟。
- 片内提供 USB2 PHY 参考时钟。
- PCIe 的 Host 和 Device 100M CLK 不直连在一起，分开由外部提供。

6.3 功能描述

6.3.1 时钟架构

图6-1 芯片时钟区域图



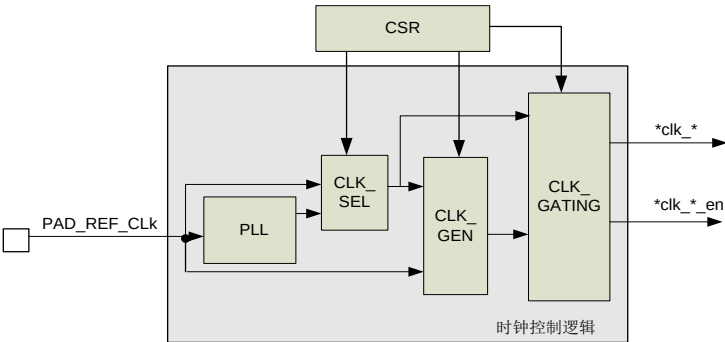
Hi3093 包含五个子系统时钟域：

- A55 子系统时钟域：覆盖 A55、GIC、A55_CRG、A55_CSR 等模块。
- LP 子系统时钟域：覆盖 I2C、M3、DMA、UART、SFC、eMMC、matrix 等模块。
- HP 子系统时钟域：覆盖 SGMII、RMII、USB3.0、USB2.0 等模块。
- HOST 子系统时钟域：覆盖 GPU、VCE、PCIe0、PCIe1 等模块。
- DDR 子系统时钟域：覆盖 DDR 模块。

CRG 整体架构如图 6-2 所示，CRG 内包括 PLL、时钟分频、Gating 等时钟单元。时钟控制由 CSR 模块完成。



图6-2 CRG 时钟架构



6.3.2 TOP_CRG 描述

TOP_CRG 包括 4 个 PLL，根据输入时钟和 PLL 配置生成各子系统 CRG 输入时钟、总线时钟及 DDR 时钟。

- CPLL：输出 1GHz 时钟，供 A55 Cluster 使用。
- PPLL0：输出 2GHz 时钟，供产生外设总线、工作时钟。
- PPLL1：输出 1.2GHz，用于产生 DDR 所需要时钟。
- PPLL2：用于产生 GPU 所需要的 DVI 时钟。

各输出时钟默认均 bypass 到参考时钟，上电流程需检查 PLL 锁定后切换到 PLL 输出时钟。切换是需逐级进行，先切换第一级（PLL 输出静态 mux）再切换后一级。

Hi3093 各功能模块时钟和 PLL 对应关系如表 6-1 所示，默认配置可支持系统正常工作。

表6-1 各 IP 时钟和 PLL 对应关系表

PLL	PLL 默认输出频率	各输出时钟	频率	说明
CPLL	1GHz	A55	1GHz	Core 工作时钟，可根据需求修改频率。
PPLL0	2GHz	HP_CLK	500MHz	用于 USB、PCIe、GMAC 等模块。
		LP_CLK	400MHz	用于总线及 eMMC、SDIO、SFC 等模块。

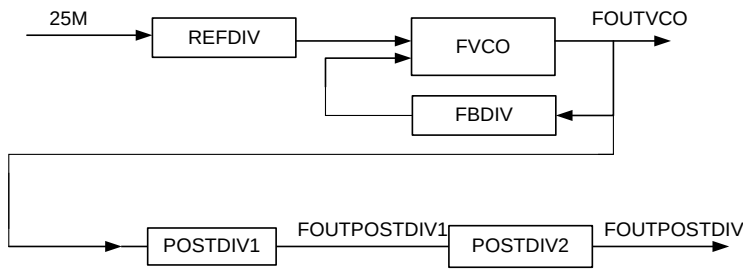


PLL	PLL 默认输出频率	各输出时钟	频率	说明
PLL1	1.2GHz	DDRC_CLK	600MHz	DDRC 工作频率，可根据需求修改频率。
PPLL2	200MHz	GPU_CLK	200MHz	PPLL2 的时钟由 GPU 控制，根据分辨率不同而变化，在测试模式可以从管脚输入时钟。

6.3.2.1 PLL 说明

芯片内部集成四个相同的 PLL，PLL 内部输出时钟频率关系如图 6-3 所示。

图6-3 PLL 内部输出时钟频率关系示意图



PLL 输出时钟的频率计算公式为如下：

- 当 dsmpd=0， $FOUTVCO = FREF / refdiv * (fbdiv + frac / 2^{24})$ ，小数分频模式。
- 当 dsmpd=1， $FOUTVCO = FREF / refdiv * fbdiv$ ，整数分频模式。
- $FOUTPOSTDIV = FOUTVCO / postdiv1 / postdiv2$

其中各参数如下：

- FREF：输入参考时钟频率。
- REFDIV：参考时钟分频比，取值范围 1~63，要求满足 $1MHz \leq FREF / REFDIV \leq 40MHz$ 。
- FBDIV：内部反馈分频器的分频比，当 PLL 工作在整数模式时，有效的取值为 8、9、12、13、14、16 ~ 4095；当 PLL 工作在小数模式时，有效的取值为 19 ~ 4092。



- POSTDIV1: 图 6-3 中的第一级分频器的分频比, 取值范围 1 ~ 7。
- POSTDIV2: 图 6-3 中的第二级分频器的分频比, 取值范围 1 ~ 7。
- FOUTPOSTDIV 是 PLL 输出管脚, 为其他电路提供工作时钟, FOUTVCO 和 FOUTPOSTDIV1 是 PLL 内部时钟。

须知

PLL 内部 VCO 输出时钟频率 FOUTVCO 的有效频率范围为 800MHz ~ 3200MHz。

PLL 控制信号和状态通过 SYSCtrl 的 CSR 寄存器配置及记录, 信号说明如表 6-2 所示。

表6-2 PLL 信号列表

PLL 信号名	方向	位宽	说明	备注
FREF	I	1	Reference Clock	参考时钟 (1MHz ~ 1.2GHz) , fractional mode 最小频率是 10MHz。
REFDIV	I	6	Reference Divide Value	参考分频值 (1 ~ 63) , 分频后时钟频率>1MHz 且<40MHz, fractional mode 下>10MHz。
FBDIV	I	12	Integer Feedback Divide Value	整数反馈分频值。
FRAC	I	24	Fractional Feedback Divide Value	小数反馈分频值。
POSTDIV1	I	3	First Post Divide Value	VCO 之后第一级分频值, 1~7。
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。



PLL 信号名	方向	位宽	说明	备注
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。
FOUTVCO PD	I	1	Bufferd VCO Clock Power Down	VCO 之后的 buffer power down。
LOCK	O	1	PLL Lock Signal	PLL 锁定指示信号。

6.3.2.2 PLL 配置

步骤 1 配置 PLL 的内外部 bypass 都使能。

配置系统控制器 CLK_BYPASS_CTRL_REG 寄存器为 0x0F。

步骤 2 配置 PLL 全局下电。

CPLL 配置系统控制器 CPLL_CTRL_REG_1 的 cpll_pd 位为 1。

PPLL0 配置系统控制器 PPLL0_CTRL_REG_1 的 ppll0_pd 位为 1。

PPLL1 配置系统控制器 PPLL1_CTRL_REG_1 的 ppll1_pd 位为 1。

步骤 3 配置分频参数。

系统控制器的 PLL 控制寄存器初始值按照参考时钟 25M 及默认频率设计, 如无需改变可跳过此步骤。如需改变工作时钟频率请参考 PLL 时钟计算公式及时钟架构配置如下寄存器。



CPLL: CPLL_CTRL_REG_0、CPLL_CTRL_REG_1。

PPLL0: PPLL0_CTRL_REG_0、PPLL0_CTRL_REG_1。

PPLL1: PPLL0_CTRL_REG_0、PPLL0_CTRL_REG_1。

步骤 4 延时 5us。

步骤 5 配置 PLL 全局上电。

CPLL 配置系统控制器 CPLL_CTRL_REG_1 的 cpll_pd 位为 0。

PPLL0 配置系统控制器 PPLL0_CTRL_REG_1 的 ppll0_pd 位为 0。

PPLL1 配置系统控制器 PPLL1_CTRL_REG_1 的 ppll1_pd 位为 0。

步骤 6 延时 100us。

步骤 7 查询 PLL 的锁定状态寄存器 PLL_LOCK_STATUS 的 bit[0]、bit[1]、bit[2]、bit[3]分别对应 CPLL、PPLL0、PPLL1、PPLL2。1 表示锁定，若未锁定，则认为锁相环锁定失败，软件打印相关信息。

步骤 8 配置 PLL 的第二级 bypass 都不使能；配置系统控制器 CLK_BYPASS_CTRL_REG 寄存器为 0x00。

步骤 9 配置系统进入正常模式，PLL 初始化结束。

----结束

须知

- PPLL2 的配置寄存器位于 GPU 模块内，由主机通过 PCIe 配；需要 GPU 驱动配置 PPLL2 的 PD 信号为 1，即 GPU 的寄存器（地址为 MMIO_base+0x802B8）VGA PLL2[31:0]=0x21000000。
- VCO≥2400MHz 时，高温+低压/常压 corner 下 Jitter 较大，超过指标值，且 VCO 频率提高，Jitter 会增大。如果 LTJ (long term jitter) 是关键指标，则需要规避；例如提升电压，限制 VCO 最高频率等。

PLL 的参考配置分别如表 6-3~表 6-6 所示。



表6-3 CPLL 配置参考

CPLL 配置	PLL 输出频率 (MHZ)	800
CPLL_CTRL_REG_0	整寄存器配置值	0xA000000
29:27	POSTDIV2	0x1
26:24	POSTDIV1	0x2
23:0	FRAC	0x0
CPLL_CTRL_REG_1	整寄存器配置值	0x401040
24	BYPASS	0x0
23	PD	0x0
22	DSM_PD	0x1
21	DAC_PD	0x0
20	FOUT4PHASE_PD	0x0
19	FOUTPOSTDIV_PD	0x0
18	FOUTVCO_PD	0x0
17:12	REFDIV	1
11:0	FBDIV	40

表6-4 PLL0 配置参考

PLL0 配置	PLL 输出频率 (MHZ)	2000
PLL0_CTRL_REG_0	整寄存器配置值	0x9000000
29:27	POSTDIV2	0x1
26:24	POSTDIV1	0x1
23:0	FRAC	0x0
PLL0_CTRL_REG_1	整寄存器配置值	0x401050
24	BYPASS	0x0
23	PD	0x0
22	DSM_PD	0x1
21	DAC_PD	0x0
20	FOUT4PHASE_PD	0x0



PLL0 配置	PLL 输出频率 (MHZ)	2000
19	FOUTPOSTDIV_PD	0x0
18	FOUTVCO_PD	0x0
17:12	REFDIV	1
11:0	FBDIV	50

表6-5 PLL1 配置参考

PLL1 配置	PLL 输出频率 (MHZ)	600
PLL1_CTRL_REG_0	整寄存器配置值	0x9000000
29:27	POSTDIV2	0x1
26:24	POSTDIV1	0x1
23:0	FRAC	0x0
PLL1_CTRL_REG_1	整寄存器配置值	0x401030
24	BYPASS	0x0
23	PD	0x0
22	DSM_PD	0x1
21	DAC_PD	0x0
20	FOUT4PHASE_PD	0x0
19	FOUTPOSTDIV_PD	0x0
18	FOUTVCO_PD	0x0
17:12	REFDIV	1
11:0	FBDIV	30



表6-6 PLL2 配置参考

PLL 配置寄存器描述	640x480 模式配置	800x600 模式配置	1024x768 模式配置
配置值[63:32]	206B851E	30000000	40000000
配置值[31:0]	23d40f02	23940801	23940d01
参考时钟	25	25	25
输出频率 (MHZ)	25.17499998	40	65

表6-7 PLL2 详细配置参考

PLL2 配置	配置寄存器描述	640x480 模式配置	800x600 模式配置	1024x768 模式配置
55:32	FRAC	6B851E	0	0
31	DACPD	0	0	0
30	gpu_bypass	0	0	0
29	BYPASS	1	1	1
28	FOUTVCOPD	0	0	0
27	FOUTPOSTDIVPD	0	0	0
26	FOUT4PHASEPD	0	0	0
25	DSMPD	1	1	1
24	PD	1	1	1
23:21	POSTDIV2	6	4	4
20:18	POSTDIV1	5	5	5
17:6	FBDIV	3C	20	34
5:0	REFDIV	2	1	1



7 复位

7.1 概述

Hi3093 包括 HOST 和非 HOST 两个独立复位域。各复位域支持管脚触发复位。非 HOST 复位域还支持内部 WatchDog 触发复位。同时 Hi3093 可通过寄存器触发各复位域复位，该功能作为 DFX 手段。非 HOST 域包括安全子系统和非安全子系统两部分，安全子系统复位时，非安全子系统同时也进行复位。非安全子系统单独复位时，安全子系统不受影响。Watchdog 触发的复位域可通过寄存器设置。

7.2 特性

- 支持管脚控制上电复位及热复位。
- 支持系统软件复位。
- 支持功能模块软件复位。
- 芯片分 3 个复位域：上电复位域、HOST 复位域及非 HOST 复位域，分别有管脚控制复位。
- PWM 模块复位可配置为非 HOST 域或者上电复位域。
- 支持 PCIe0/1 单独管脚复位，分别由 PCIE0_RST_INT 和 PCIE1_RST_INT 控制。
- 非 HOST 复位域内根据安全性细分为安全和非安全两个子域，如果安全域复位则非 HOST 域整体复位。
- WatchDog 复位域可配置（chip、协处理模式、非安全协处理模式三个域）、复位可屏蔽，复位延时可配。



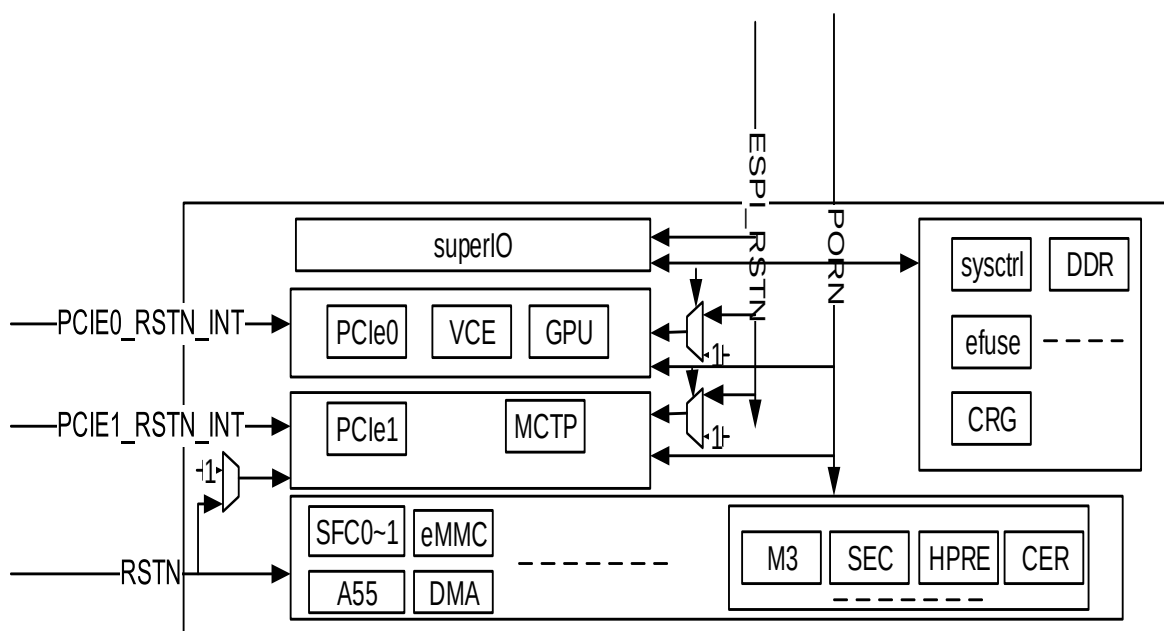
- 除上电复位，其他复位均有状态记录指示是否发生对应复位；提供协处理器 WatchDog 监控，复位原因有记录。
- 状态记录热复位不丢失。
- 芯片提供一个复位请求中断管脚。
- 协处理模式非安全域复位时输出指示信号（低有效），持续时间可配，默认 200ms。

7.3 功能描述

7.3.1 复位接口信号

复位管脚与内部模块对应关系如图 7-1 所示。

图7-1 复位管脚域内部模块对应关系图



须知

管脚复位输入有效时间需保持 5us 以上。复位间隔大于 1s。



7.3.2 复位种类

7.3.2.1 外部管脚硬复位

- **ESPI_RSTN 复位**: ESPI Slave 复位, 外部管脚输入。

默认复位范围:

PCIE0、PCIE1、PCIE Serdes、MCTP、GPU、除 SOL 外的整个 SUPERIO 子系统, 以及外围接口和相应总线。

说明

低速外设及 SuperIO 与 HOST 通讯接口随 HOST 复位。

- **RSTN 复位**: 热复位, 协处理模式域复位, 按键复位。

默认复位范围:

默认为 Hi3093 侧, 包括: USB3.0、USB2.0 0/1/2、SEC、GMAC、SGMII 0/1/2、RMII 0/1、A55、GIC、DMA、Local bus、SRAM、M3、SFC0、SFC1、SDIO、I2C、Timer、PECL、MDIO、GPIO (可选)、SPI、WDG、PWM、FAN、AD、通用 UART 以及 SYSCCTRL 的协处理器部分, 以及外围接口和相应总线。

- **PORN 复位**: 上电复位, 外部管脚输入, 对整个芯片进行复位。

默认复位范围:

芯片所有模块。其中 VCE、SOL、HOST RTC、DDRC、模块以及相关总线只在整芯片复位时复位, 不随 HOST 和 Hi3093 复位。

须知

DDR 仅在 PORN 上电复位时被复位。Sysctrl 支持软件复位整芯片时 DDR 数据不丢失, 当系统使用软件热复位整芯片需确保按照流程完成对应寄存器配置后, 再配置 DDRC_CTRL 的 ddr_reset_retention 为 0, 此时软件复位才可生效。DDR 复位保持数据流程参照 DDRC FS。

Sysctrl 支持 7 个 WDG 复位输入, 其中 0~3 对应 A55, 4 对应协处理器 M3, 5 对应安全核, 6 对应管脚输出。除 PAD_PORN 管脚复位及 WDG 复位均在 Reset_Source_CRG 寄存器记录, 热复位不丢失。

7.3.2.2 内部软复位

Hi3093 提供两种芯片内部复位:



- 内部狗 WDG 复位
- 寄存器触发软复位

内部 WatchDog 复位域可配置。各个复位域及功能模块均支持软复位，分别有寄存器对应。

7.4 工作方式

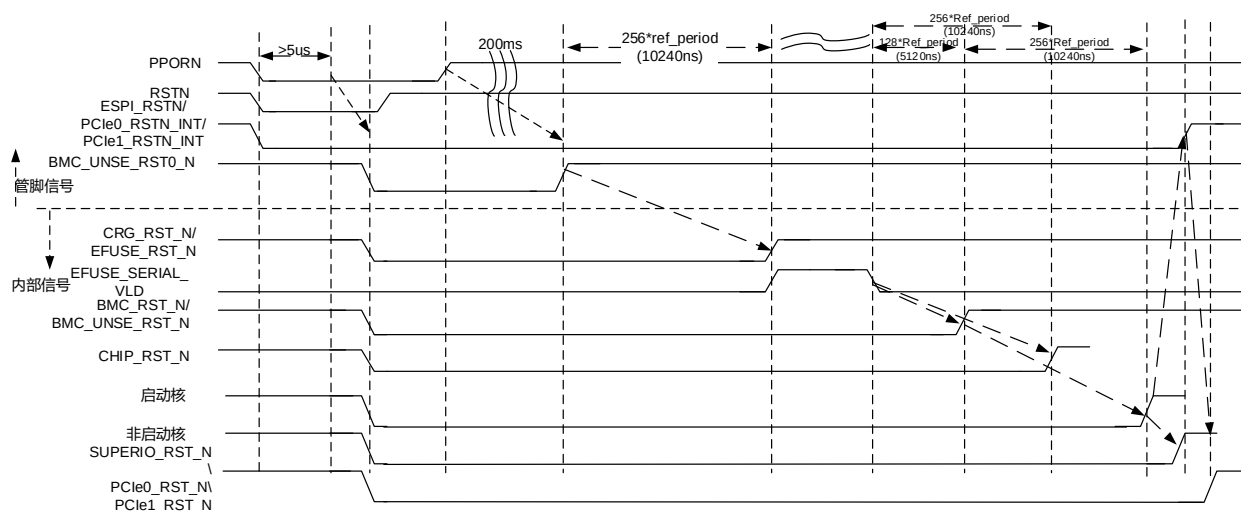
7.4.1 整芯片复位

整芯片包括系统上电复位及芯片热复位。

7.4.1.1 系统上电复位

系统上电复位需所有复位控制管脚配合完成，时序如图 7-2 所示。

图7-2 上电复位时序图



步骤 1 系统电源稳定后，拉低 PAD_PORN、PAD_RSTN、PAD_ESPI_RSTN、PAD_PClie0_RSTN_INT、PAD_PClie1_RSTN_INT 并保持超过 5us。

步骤 2 芯片收到管脚复位后经过滤毛刺、同步等模块后把芯片内所有复位信号拉低，同时将输出信号 PAD_BMC_UNSE_RST0_N 为 0。



- 步骤 3 系统先置 PAD_RSTN 无效，1us 后再置 PAD_PORN 无效，并保持 PAD_ESPI_RSTN、PAD_PCIE0_RSTN_INT、PAD_PCIE1_RSTN_INT 处于复位状态。
- 步骤 4 PAD_PORN 无效后芯片保持 200ms PAD_BMC_UNSE_RSTO_N 输出为 0 再将 PAD_BMC_UNSE_RSTN 置为无效。
- 步骤 5 经过 256 个参考时钟周期，硬件释放内部 crg_rst_n 及 efuse_rst_n 并等待 efuse_serial_vld 下降沿。
- 步骤 6 硬件收到 efuse_serial_vld 下降沿后等待 128 个参考时钟周期释放 bmc_rst_n 及 bmc_unse_rst_n 信号。
- 步骤 7 再过 128 个参考时钟周期硬件自动释放 chip_rst_n。
- 步骤 8 等待 128 个参考时钟周期释放启动核复位 (a55_rst_n 或 sc_rst_n)。
- 步骤 9 PORN 释放 1ms 后释放 PAD_ESPI_RSTN、PAD_PCIE0_RSTN_INT、PAD_PCIE1_RSTN_INT 复位。
- 步骤 10 芯片收到 PAD_ESPI_RSTN、PAD_PCIE0_RSTN_INT、PAD_PCIE1_RSTN_INT 信号无效后释放内部 superio_rst_n、pcie0_rst_n/pcie1_rst_n，整个上电复位过程结束。

----结束

非启动核的复位由启动核软件释放。安全核复位软件释放配置 sysctrl 模块的 BOOT_DEFINE_REG 寄存器的 sc_boot_en 位，协处理器 M3 软件释放配置 sysctrl 模块 BOOT_DEFINE_REG 寄存器的 m3_boot_en 位，A55 复位软件释放配置 SEC_CSR 模块的 A55_BOOT_EN 寄存器。

须知

PAD_POR_RST_N 仅用于上电复位控制，不允许单独使用控制整芯片热复位。

7.4.1.2 整芯片热复位

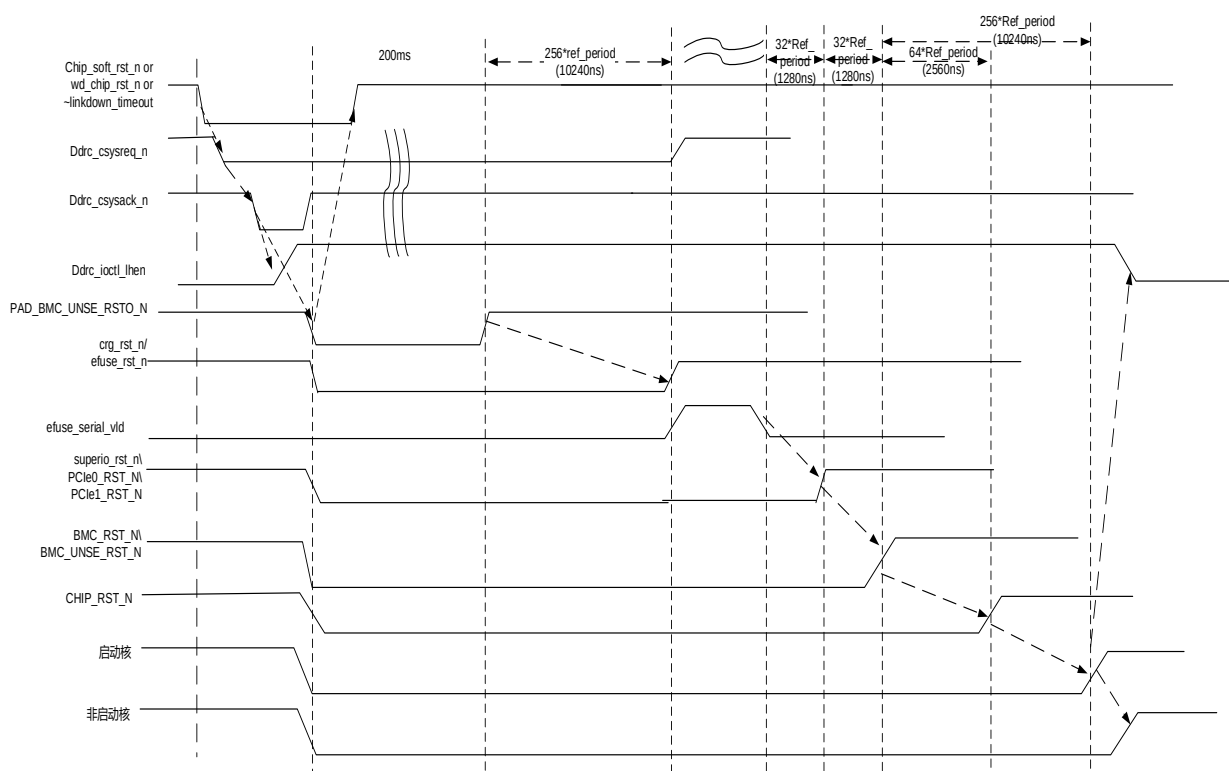
整芯片热复位源有三个：软件配置整芯片复位、WatchDog 触发整芯片复位、排空模块超时触发整芯片复位。芯片热复位是需系统保证和周边器件复位同步，否则将引起系统异常。芯片热复位时可通过配置 SYSCtrl 模块的 DDRC_CTRL 寄存器的 ddr_reset_retention 位来支持 DDR 数据不丢失。



系统热复位寄存器为 SYSCTRL 模块的 SYS_RST_REQ 寄存器 chip_rst_req。
WatchDog 复位控制可通过寄存器配置。非安全 WatchDog 复位控制配置 SYSCTRL 模块的 WD_CTRL_REG 寄存器，安全 WatchDog 复位控制配置 SEC_CSR 模块的 WD6_CTRL_REG 寄存器。如设置为 WatchDog 复位整芯片，则 WatchDog 挂死时触发整芯片复位。排空操作为子复位域单独复位时为保证其复位不影响其他逻辑而进行的硬件操作，如该过程发生异常未能完成，可配置进行整芯片复位。通过配置 SYSCTRL 模块的 XSEMP_TO_CTRL 寄存器，来设置超时时间及是否进行整芯片复位。

热复位流程如图 7-3 所示。

图7-3 热复位流程图



步骤 1 内部触发整芯片复位，如支持复位 DDR 数据不丢失则执行步骤 2，不支持则跳到步骤 4。

步骤 2 置 DDRC 的 ddr_csysreq_n 信号有效，等待 DDRC 发出 ddr_csysack_n。

步骤 3 硬件收到 ddr_csysack_n 后，自动置 ddr_ioctl_lthen 为 1。



- 步骤 4 触发内部复位有效，同时输出 PAD_BMC_UNSE_RSTO_N 有效并保持 200ms（是否输出及保持时间可通过 sysctrl 模块的 RST_OUT_CTRL_REG 及 RST_OUT_DLY_CFG 配置），系统可以利用该信号对周边器件进行复位。
- 步骤 5 经过 256 个参考时钟周期，硬件释放 crg_rst_n 和 efuse_rst_n 并等待 efuse_serial_vld 下降沿。如支持 DDR 数据保持同时置 ddrc_csysreq_n 信号无效。
- 步骤 6 收到 efuse_serial_vld 下降沿后硬件等待 32 个参考时钟周期自动释放 superio_rst_n/pcie0_rst_n/pcie1_rst_n 信号。
- 步骤 7 等待 32 个参考时钟周期硬件释放 bmc_rst_n 及 bmc_unse_rst_n 信号。
- 步骤 8 等待 64 个参考时钟周期硬件释放 chip_rst_n。
- 步骤 9 等待 192 个参考时钟周期释放启动核复位（a55_rst_n 或 sc_rst_n）。
- 步骤 10 软件配置释放非启动核复位。如支持 DDR 数据保存则先恢复 DDRC 配置（详细流程参考 DDRC FS 章节），然后配置 SYSCTRL 模块的 DDRC_CTRL 寄存器的 ddrc_ioctl_lhen_clr 位，置 ddrc_ioctl_lhen 为 0。整个复位过程结束。

----结束

须知

支持 DDR 数据保存需提前将 DDR 的配置保存在不受复位影响的存储区中（Flash 或 SRAM）。

7.4.2 协处理模式域单独复位

协处理模式域可通过管脚、软件、WatchDog 触发单独复位。触发协处理模式域单独复位管脚为 PAD_RSTN。软件通过配置 SYSCTRL 模块的 SYS_RST_REQ 寄存器 bmc_rst_req 位来触发协处理模式域单独复位管脚。WatchDog 复位控制可通过寄存器配置。非安全 WatchDog 复位控制配置 SYSCTRL 模块的 WD_CTRL_REG 寄存器，安全 WatchDog 复位控制配置 SEC_CSR 模块的 WD6_CTRL_REG 寄存器。如设置为 WatchDog 复位协处理模式域，则 WatchDog 挂死时触发协处理模式域单独复位。

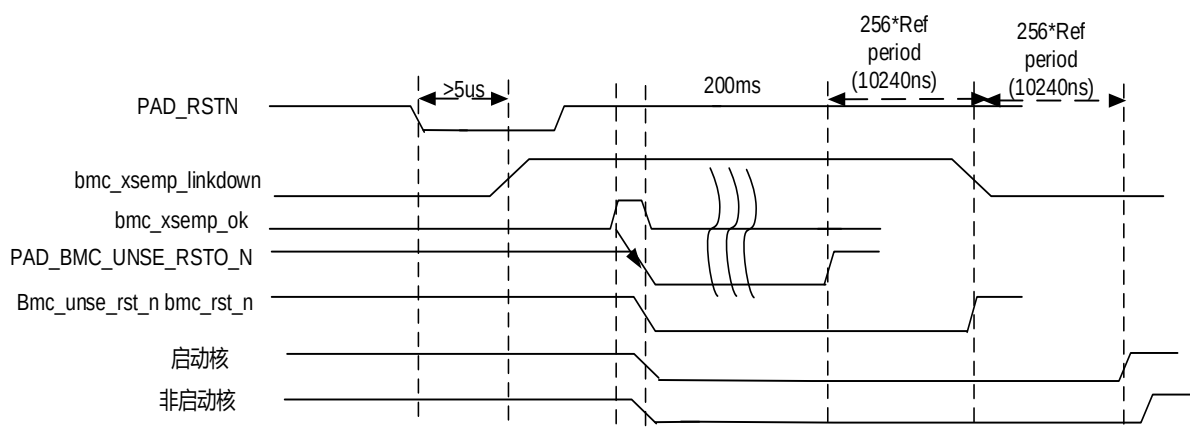
协处理模式域单独复位时不影响其他域正常工作。硬件通过触发域接口排空模块进行排空操作，排空后才进行真正的复位来实现域复位不对其他部分逻辑影响。在整个排空操作中如软件跨域进行读访问接口排空模块会自动返回全 1，写访问会被屏蔽。



7.4.2.1 管脚触发协处理模式域复位

系统通过 PAD_RSTN 触发协处理模式域复位，复位流程如图 7-4 所示。

图7-4 管脚触发协处理模式域复位时序



- 步骤 1 系统拉低 PAD_RSTN 超过 5us，触发协处理模式域复位流程。
- 步骤 2 硬件发出 bmc_xsemp_linkdown 信号，通知排空模块进行数据排空操作，并等待 bmc_xsemp_ok 返回。
- 步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 bmc_xsemp_ok 则输出 PAD_BMC_UNSE_RSTO_N 为 0，同时置 bmc_unse_rst_n、bmc_rst_n、m3_rst_n、a55_rst_n、sc_rst_n 有效。
- 步骤 4 PAD_BMC_UNSE_RSTO_N 保持时间为 RST_OUT_DLY_CFG 配置，也可通过 sysctrl 模块的 RST_OUT_CTRL_REG 设置跳过该步骤。
- 步骤 5 超过 XSEMP_TO_CTRL 设定时间则认为排空超时。排空超时根据 XSEMP_EN 寄存器的 bmc_xsemp_to_ctrl 设置选择进行复位行为控制。
- 步骤 6 硬件置 PAD_BMC_UNSE_RSTO_N 为 1，等待 256 个参考时钟周期，如此时 PAD_RSTN 无效则硬件自动释放 bmc_unse_rst_n 和 bmc_rst_n。如 PAD_RSTN 有效则等待 PAD_RSTN 无效时再置 bmc_unse_rst_n 和 bmc_rst_n 无效，同时置 bmc_xsemp_linkdown 无效。
- 步骤 7 释放 bmc_unse_rst_n 和 bmc_rst_n 无效后 256 个参考时钟周期，硬件置启动核 (A55 或者 sc) 复位无效。



步骤 8 软件配置释放其他核的复位。复位流程结束。

---结束

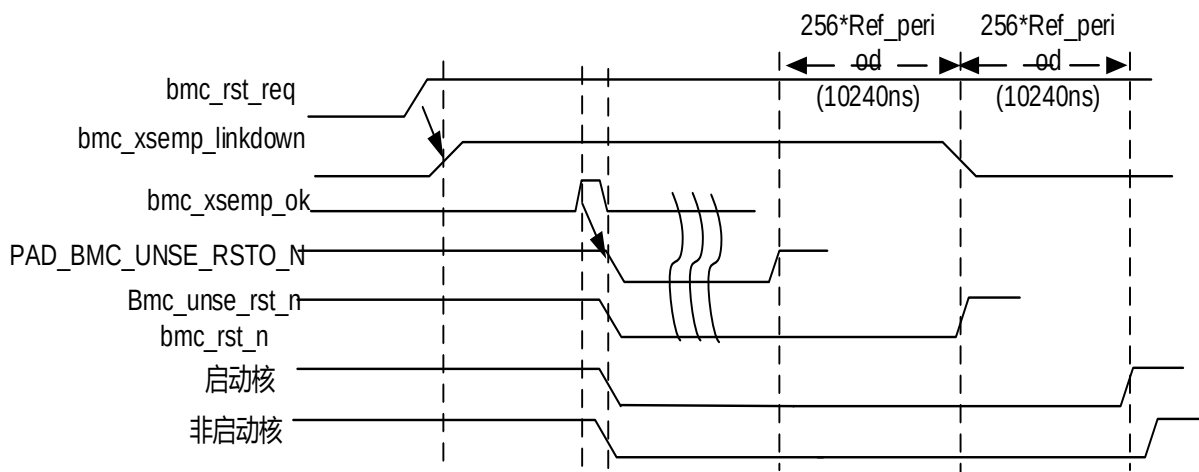
须知

系统需使用 PAD_BMC_UNSE_RSTO_N 复位周边相关器件保证系统工作正常。

7.4.2.2 软件触发协处理模式域复位

软件可配置 sysctrl 模块的 SYS_RST_REQ 寄存器 bmc_rst_req 触发协处理模式域复位。其流程如图 7-5 所示。

图7-5 软件触发协处理模式域复位时序



步骤 1 软件配置 bmc_rst_req 由 0 变为 1 触发协处理模式域复位流程。

步骤 2 硬件发出 bmc_xsemp_linkdown 信号，通知排空模块进行数据排空操作，并等待 bmc_xsemp_ok 返回。

步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 bmc_xsemp_ok 则输出 PAD_BMC_UNSE_RSTO_N 为 0，同时置 bmc_unse_rst_n、bmc_rst_n、m3_rst_n、a55_rst_n、sc_rst_n 有效。



- 步骤 4 PAD_BMC_UNSE_RSTO_N 保持时间为 RST_OUT_DLY_CFG 配置，也可通过 sysctrl 模块的 RST_OUT_CTRL_REG 设置跳过该步骤。
- 步骤 5 超过 XSEMP_TO_CTRL 设定时间则认为排空超时。排空超时根据 XSEMP_EN 寄存器的 bmc_xsemp_to_ctrl 设置进行复位控制。
- 步骤 6 硬件置 PAD_BMC_UNSE_RSTO_N 为 1，等待 256 个参考时钟周期，置 bmc_unse_rst_n 和 bmc_rst_n 无效，同时置 bmc_xsemp_linkdown 无效。
- 步骤 7 释放 bmc_unse_rst_n 和 bmc_rst_n 无效后 256 个参考时钟周期，硬件置启动核 (A55 或者 sc) 复位无效。
- 步骤 8 软件配置释放其他核的复位。复位流程结束。

----结束

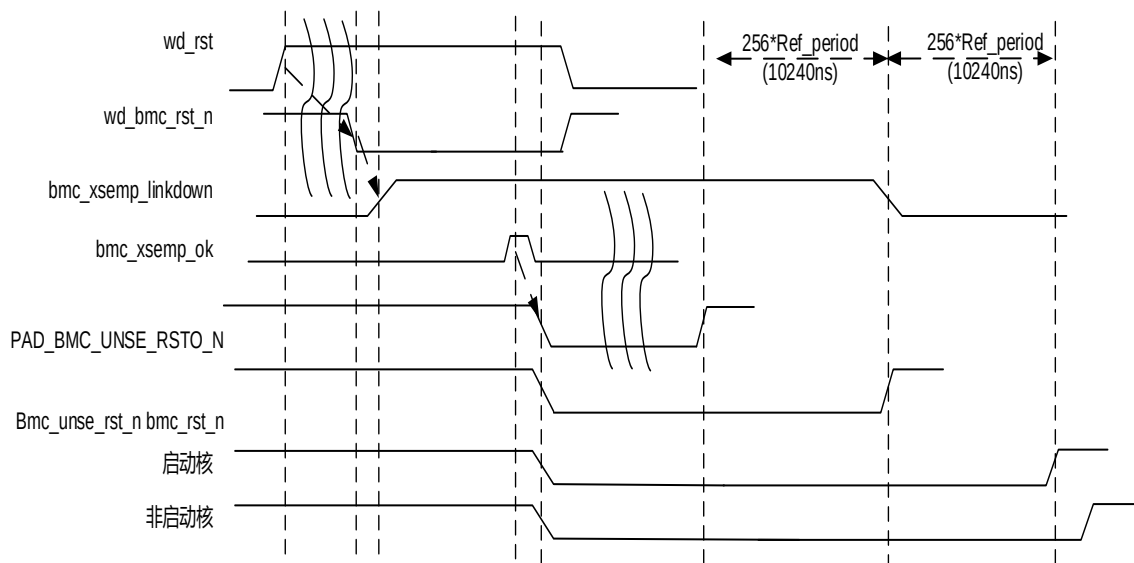
由于 bmc_rst_req 位于整芯片复位域，Hi3093 复位并不能清除该请求，需软件配置 SYS_RST_DREQ 寄存器 bmc_rst_dreq 位清除复位请求。系统需使用 PAD_BMC_UNSE_RSTO_N 复位相关周边器件保证系统工作正常。

7.4.2.3 WatchDog 触发协处理模式域复位

非安全 WatchDog 复位控制配置 SYSCTRL 模块的 WD_CTRL_REG 寄存器，安全 WatchDog 复位控制配置 SEC_CSR 模块的 WD6_CTRL_REG 寄存器。如设置为 WatchDog 复位协处理模式域，则 WatchDog 挂死时触发协处理模式域复位。复位流程如图 7-6 所示。



图7-6 WatchDog 触发协处理模式域复位时序



- 步骤 1 Watchdog 挂死后经过 WD_DLY_CFG 个参考时钟周期后，硬件根据配置置 wd_bmc_rst_n 为低，触发协处理模式域复位流程。
- 步骤 2 硬件发出 bmc_xsemp_linkdown 信号，通知排空模块进行数据排空操作，并等待 bmc_xsemp_ok 返回。
- 步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 bmc_xsemp_ok 则输出 PAD_BMC_UNSE_RSTO_N 为 0，同时置 bmc_unse_rst_n、bmc_rst_n、m3_rst_n、a55_rst_n、sc_rst_n 有效。
- 步骤 4 PAD_BMC_UNSE_RSTO_N 保持时间为 RST_OUT_DLY_CFG 配置，也可通过 sysctrl 模块的 RST_OUT_CTRL_REG 设置跳过该步骤。
- 步骤 5 超过 XSEMP_TO_CTRL 设定时间则认为排空超时。排空超时根据 XSEMP_EN 寄存器的 bmc_xsemp_to_ctrl 设置复位行为。
- 步骤 6 硬件置 PAD_BMC_UNSE_RSTO_N 为 1，等待 256 个参考时钟周期，置 bmc_unse_rst_n 和 bmc_rst_n 无效，同时置 bmc_xsemp_linkdown 无效；Watchdog 也被复位。
- 步骤 7 等待 256 个参考时钟周期，置 bmc_unse_rst_n 和 bmc_rst_n 无效，同时置 bmc_xsemp_linkdown 无效。



步骤 8 释放 bmc_unse_rst_n 和 bmc_rst_n 无效后 256 个参考时钟周期，硬件置启动核 (A55 或者 SC) 复位无效。

步骤 9 软件配置释放其他核的复位。复位流程结束。

----结束

系统需使用 PAD_BMC_UNSE_RSTO_N 复位相关周边器件保证系统工作正常。

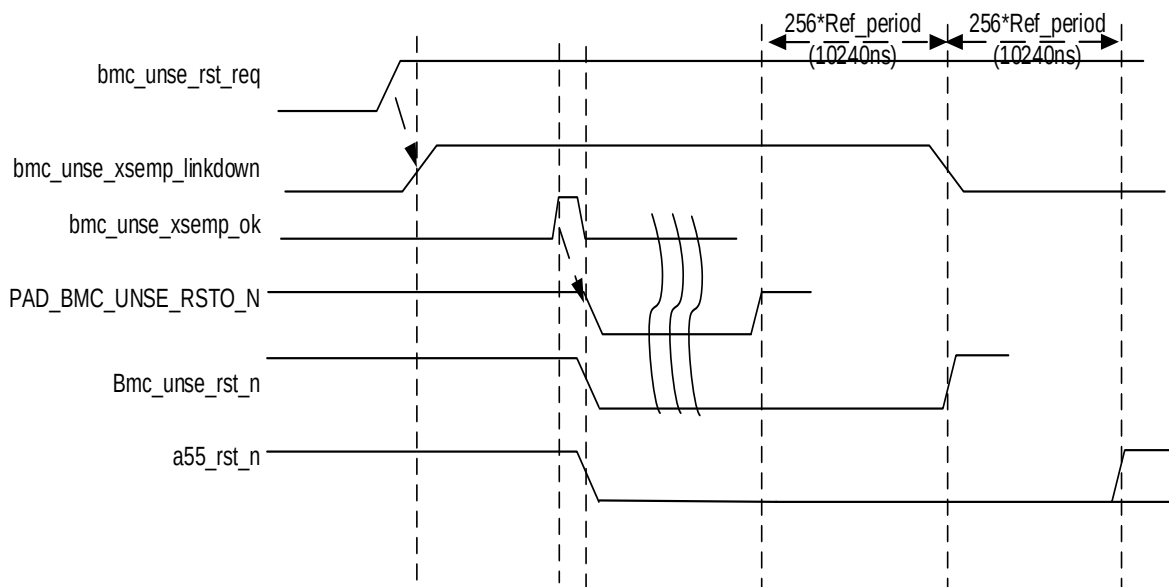
7.4.3 协处理模式非安全域单独复位

协处理模式非安全域可独立复位，触发非安全域复位的源包括软件复位、WatchDog 复位。协处理模式非安全复位会触发中断发给安全核，安全核需根据中断进行相应的软件操作已保证业务正常。系统需使用 PAD_BMC_UNSE_RSTO_N 复位相关周边器件保证系统工作正常。

7.4.3.1 软件触发协处理模式非安全域复位

软件可配置 sysctrl 模块的 SYS_RST_REQ 寄存器 bmc_unse_rst_req 触发协处理模式域复位。其流程如图 7-7 所示。

图7-7 软件触发协处理模式非安全域复位时序



步骤 1 软件配置 bmc_unse_rst_req 由 0 变为 1 触发协处理模式非安全域复位流程。



- 步骤 2 硬件发出 bmc_unse_xsemp_linkdown 信号，通知排空模块进行数据排空操作，并等待 bmc_unse_xsemp_ok 返回。
- 步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 bmc_unse_xsemp_ok 则输出 PAD_BMC_UNSE_RSTO_N 为 0，同时置 bmc_unse_rst_n、m3_rst_n、a55_rst_n 有效。
- 步骤 4 PAD_BMC_UNSE_RSTO_N 保持时间为 RST_OUT_DLY_CFG 配置，也可通过 sysctrl 模块的 RST_OUT_CTRL_REG 设置跳过该步骤。
- 步骤 5 超过 XSEMP_TO_CTRL 设定时间则认为排空超时。排空超时根据 XSEMP_EN 寄存器的 bmc_unse_xsemp_to_ctrl 设置选择进行整芯片或进行当前复位。
- 步骤 6 硬件置 PAD_BMC_UNSE_RSTO_N 为 1。等待 256 个参考时钟周期，硬件自动释放 bmc_unse_rst_n。
- 步骤 7 释放 bmc_unse_rst_n 无效后 256 个参考时钟周期，如 A55 为启动核，置启动核 a55_rst_n 复位无效，复位流程结束。如 A55 不为启动核，保持 a55_rst_n 复位有效，复位流程结束。

----结束

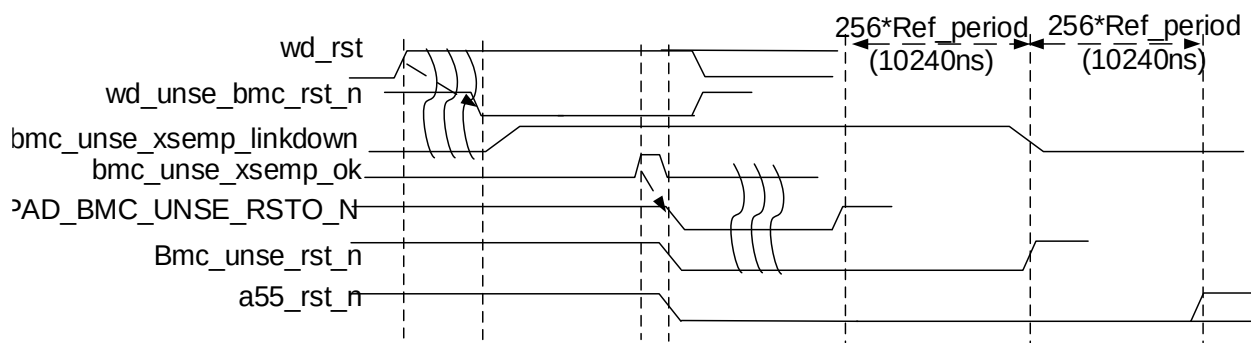
由于 bmc_unse_rst_req 位于整芯片复位域，协处理模式非安全域复位并不能清除该请求，需软件配置 SYS_RST_DREQ 寄存器 bmc_unse_rst_dreq 位清除复位请求。

7.4.3.2 WatchDog 触发协处理模式非安全域复位

非安全 WatchDog 复位控制配置 SYSCTRL 模块的 WD_CTRL_REG 寄存器，安全 WatchDog 复位控制配置 SEC_CSR 模块的 WD6_CTRL_REG 寄存器。如设置为 WatchDog 复位协处理模式非安全域，则 WatchDog 挂死时触发协处理模式非安全域复位。



图7-8 WatchDog 触发协处理模式非安全域复位时序



- 步骤 1 Watchdog 挂死后经过 WD_DLY_CFG 个参考时钟周期后，硬件根据配置置 wd_unse_bmc_rst_n 为低，触发协处理模式非安全域复位流程。
- 步骤 2 硬件发出 bmc_unse_xsemp_linkdown 信号，通知排空模块进行数据排空操作，并等待 bmc_unse_xsemp_ok 返回。
- 步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 bmc_unse_xsemp_ok 则输出 PAD_BMC_UNSE_RSTO_N 为 0，同时置 bmc_unse_rst_n、m3_rst_n、a55_rst_n 有效。
- 步骤 4 PAD_BMC_UNSE_RSTO_N 保持时间为 RST_OUT_DLY_CFG 配置，也可通过 sysctrl 模块的 RST_OUT_CTRL_REG 设置跳过该步骤。
- 步骤 5 超过 XSEMP_TO_CTRL 设定时间则认为排空超时。排空超时根据 XSEMP_EN 寄存器的 bmc_unse_xsemp_to_ctrl 设置选择进行整芯片或进行当前复位。
- 步骤 6 硬件置 PAD_BMC_UNSE_RSTO_N 为 1，等待 256 个参考时钟周期，置 bmc_unse_rst_n 无效，同时置 bmc_unse_xsemp_linkdown 无效。
- 步骤 7 释放 bmc_unse_rst_n 无效后 256 个参考时钟周期，如 A55 为启动核置启动核 a55_rst_n 复位无效复位流程结束。如 A55 不为启动核，保持 a55_rst_n 复位有效复位流程结束。

----结束

安全域 WatchDog 不受非安全域复位影响，需安全核主动清除。



7.4.4 HOST 域单独复位

Host 域逻辑用于与 Host 相连，包括 SuperIO 模块、PCIe0 相关模块、PCIe1 相关模块、LPC 模块、eSPI 模块。

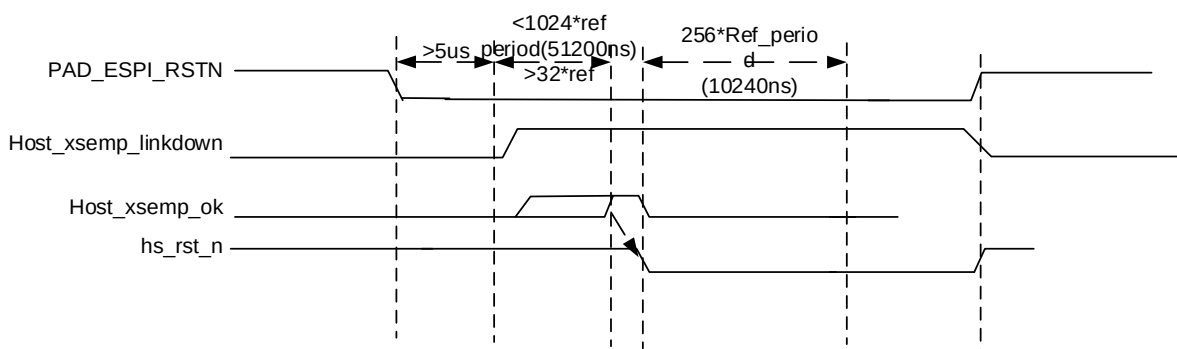
PAD_ESPI_RST_N 管脚控制 host 域复位。该信号有效时 SuperIO 一定复位。对 PCIe0 相关模块复位同时还受 STRAP 管脚 espi_rst_pcie 控制，仅 espi_rst_pcie 为 1 时 PCIe0 相关模块才受 PAD_ESPI_RST_N 管脚控制。PCIe0 相关模块还受 PAD_PCIE0_RSTN_INT 管脚控制。对 PCIe1 相关模块复位同时 STRAP 管脚 espi_rst_pcie 及寄存器 pcie1_rst_domain 控制，仅 espi_rst_pcie 为 1 时 PCIe0 且 pcie1_rst_domain 为 0 时相关模块才受 PAD_ESPI_RST_N 管脚控制，当 pcie1_rst_domain 为 1 时相关模块受 PAD_RST_N 管脚控制复位。PCIe1 相关模块还受 PAD_PCIE1_RSTN_INT 管脚控制。

Host 域复位需系统保证与 Host 侧芯片的配合，否则将导致系统工作不正常。

7.4.4.1 管脚触发 host 域单独复位

PAD_ESPI_RSTN 拉低超过 5us 则可触发 host 域复位，其复位流程如图 7-9 所示。

图7-9 管脚触发 host 域单独复位时序



步骤 1 PAD_ESPI_RSTN 拉低超过 5us 则可触发 host 域复位。

步骤 2 硬件产生 host_xsemp_linkdown 信号并等待 host_xsemp_ok 反馈。

步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 host_xsemp_ok 则触发 hs_rst_n 有效。如超过 XSEMP_TO_CTRL 设定时间则认为排空超时。排空超时根据 XSEMP_TO_CTRL 第 32bit 设置选择进行整芯片或进行当前复位。



步骤 4 Hs_rst_n 至少保持 256 个参考时钟周期，等待 PAD_ESPI_RSTN 无效释放 Hs_rst_n。复位流程结束。

----结束

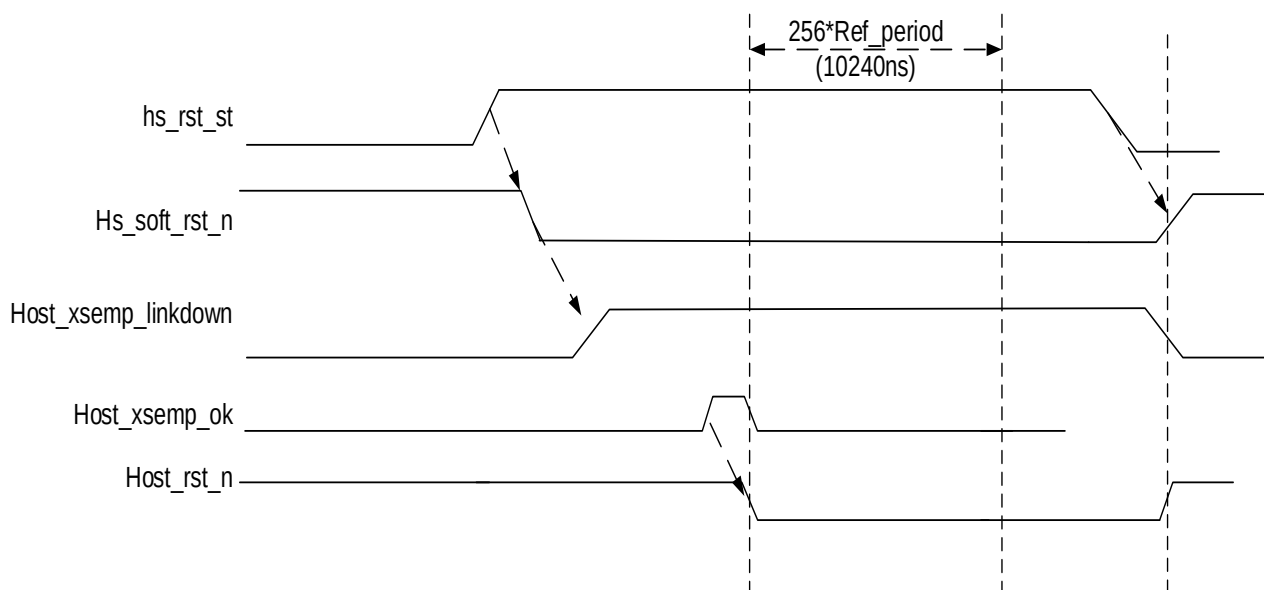
须知

PCIe0、PCIe1 相关模块由 strap 管脚及控制寄存器决定是否跟随 hs_rst_n 复位。

7.4.4.2 软件触发 host 域单独复位

软件可配置 sysctrl 模块的 SYS_RST_REQ 寄存器 hs_rst_req 触发 host 域复位。流程如图 7-10 所示。

图7-10 软件触发 host 域单独复位



步骤 1 软件配置 hs_rst_req 由 0 变为 1，硬件置 hs_soft_rst_n 有效触发 host 域复位流程。

步骤 2 硬件产生 host_xsemp_linkdown 信号并等待 host_xsemp_ok 反馈。

步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 host_xsemp_ok 则触发 hs_rst_n 有效。如超过 XSEMP_TO_CTRL 设定时间则认为排空超时。排空超时根据 XSEMP_TO_CTRL 第 32bit 设置选择进行整芯片或进行当前复位或不复位。



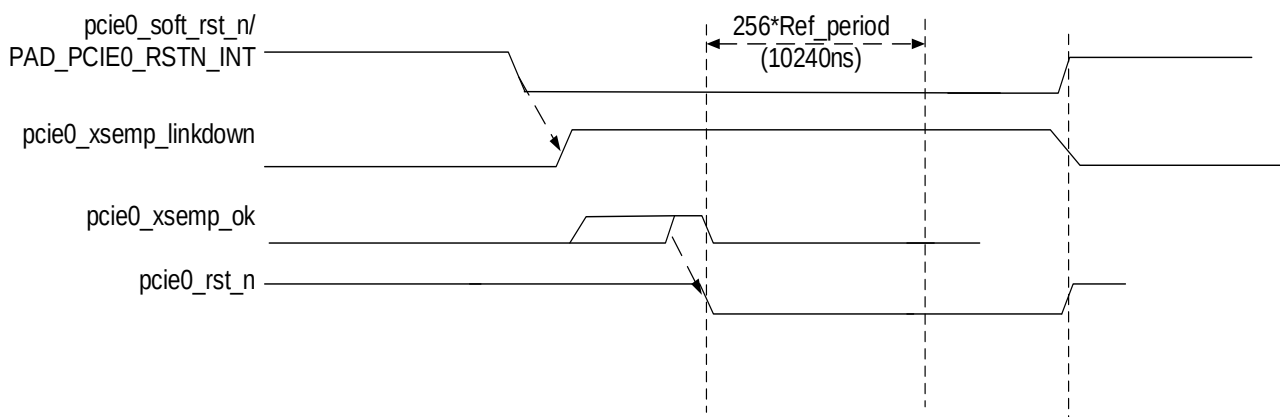
步骤 4 Hs_rst_n 至少保持 256 个参考时钟周期，等待软件配置 SYS_RST_DREQ 寄存器 hs_rst_dreq 位清除复位请求后释放 Hs_rst_n。复位流程结束。

----结束

7.4.4.3 PCIE0 相关模块单独复位

PCIE0 相关模块除了 strap 管脚 espi_rst_pcie 为 1 时受 PAD_ESPI_RSTN 管脚控制外，还支持管脚 (PAD_PCIE0_RSTN_INT) 控制复位及软件 (SYS_RST_REQ 寄存器 pcie0_rst_req) 触发复位，单独复位时需系统保证和对接 PCIe 设备同步复位。硬件对 PCIE0 复位同样使用了排空逻辑，保证其复位时不影响芯片其他逻辑工作。

图7-11 PCIE0 相关模块单独复位时序



步骤 1 软件配置 pcie0_rst_req 由 0 变为 1 或管脚 PAD_PCIE0_RSTN_INT 由 1 变成 0 超过 5us，硬件置 pcie0_soft_rst_n 有效触发 pcie0 相关模块复位流程。

步骤 2 硬件产生 pcie0_xsemp_linkdown 信号并等待 pcie0_xsemp_ok 反馈。

步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 pcie0_xsemp_ok 则触发 pcie0_rst_n 有效。如超过 XSEMP_TO_CTRL 设定时间则认为排空超时，超时后复位行为可通过 XSEMP_EN 寄存器的 pcie0_xsemp_to_ctrl 设置。

步骤 4 Pcie0_rst_n 至少保持 256 个参考时钟周期，等待软件配置 SYS_RST_DREQ 寄存器 pcie0_rst_dreq 位清除复位请求或管脚复位释放后释放 pcie0_rst_n。复位流程结束。

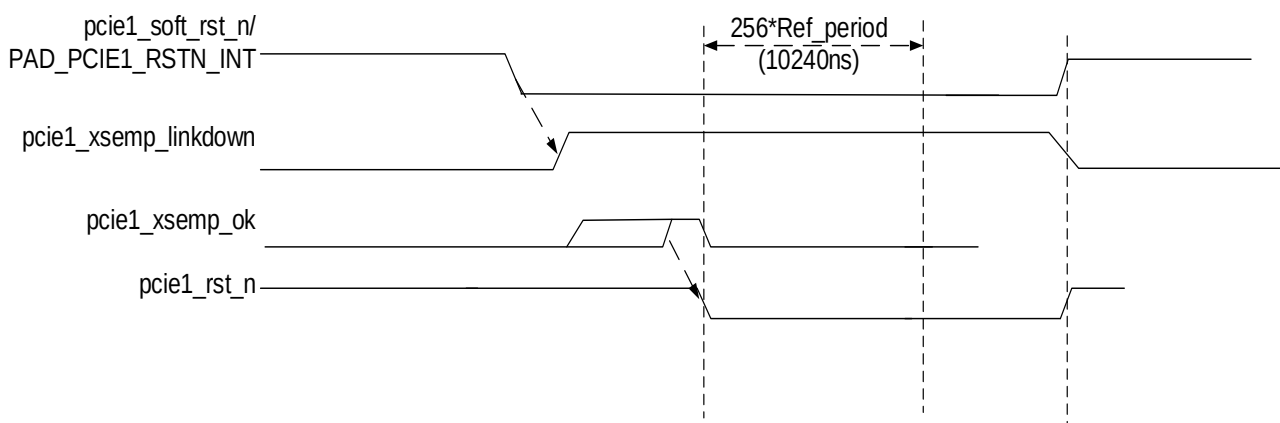
----结束



7.4.4.4 PCIe1 单独复位

PCIe1 相关模块除了随着 HOST 域或者协处理模式非安全域复位外，还可以单独被复位（PAD_PCIE1_RSTN_INT 及 SYS_RST_REQ 寄存器 pcie1_rst_req）。单独复位时需系统保证和对接 PCIe 设备同步复位。硬件对 PCIe1 复位同样使用了排空逻辑，保证其复位时不影响芯片其他逻辑工作。

图7-12 PCIe1 单独复位时序



- 步骤 1 软件配置 pcie1_rst_req 由 0 变为 1 或管脚 PAD_PCIE1_RSTN_INT 由 1 变成 0 超过 5us，硬件置 pcie1_soft_rst_n 有效触发 pcie1 相关模块复位流程。
- 步骤 2 硬件产生 pcie1_xsemp_linkdown 信号并等待 pcie1_xsemp_ok 反馈。
- 步骤 3 在 XSEMP_TO_CTRL 寄存器低 31bit 设定周期内收到 pcie1_xsemp_ok 则触发 pcie1_rst_n 有效。如超过 XSEMP_TO_CTRL 设定时间则认为排空超时，超时后复位行为通过 XSEMP_EN 寄存器的 pcie1_xsemp_to_ctrl 设置。
- 步骤 4 Pcie1_rst_n 至少保持 256 个参考时钟周期，等待软件配置 SYS_RST_DREQ 寄存器 pcie1_rst_dreq 位清除复位请求或管脚复位释放后释放 pcie1_rst_n。复位流程结束。

----结束

7.4.5 模块软件复位

各功能模块均提供软件复位寄存器，但使用软件复位需注意以下几点：

- 不支持对处理器相关模块单独软复位。
- 对功能模块软复位需确保在模块处于空闲的时候。
- 模块处于复位期间不能对其进行访问。



- 特定模块提供多个单独软件复位寄存器分别控制不同的逻辑复位，对模块整体复位时需同时配置不能遗漏，存在多个软件复位模块如表 7-1 所示。

表7-1 软件复位模块表

模块	复位寄存器
VCE	HOST_SUBSYS 模块 HOST_RST_ENALBE 寄存器
SGMII0~1	HP_SUBSYS 模块 HP 软复位使能寄存器
RMII0~1	
USB0~3	

- 特定模块的控制寄存器不受软件复位控制，包括 I2C、MDIO、WDG。

软件复位寄存器位于各个子系统的 CSR 模块内，详细说明参加对应《Hi3093 寄存器手册》。

须知

软复位需要系统保证当前业务全部完成并停止新的业务后再复位，同时复位期间不能访问被复位模块，否则可能会导致系统挂死。

7.5 寄存器特殊复位

7.5.1 DMAC

以下寄存器是使用 memory 构建的，不能复位这种寄存器的内容，其他寄存器可以正常复位（dmac_m 取值范围 0-15）：



表7-2 DMAC 特殊复位寄存器

名称	用途	偏移地址	说明
DMAC_Cm SRCADDR	通道 m 的源地地址寄存器。	0x0100+dmac_m*0x001	通道 m（链表）当前节点源地址（软件初始化时需要保证它是与实际数据宽度对齐）。数据搬移过程中硬件实时自动更新，完成一个链表节点传输后硬件自动加载下一个节点地址。
DMAC_Cm DESTADDR	通道 m 的目标地址寄存器。	0x0104+dmac_m*0x0010	通道 m（链表）当前节点目的地址（软件初始化时需要保证它是与实际数据宽度对齐）。数据搬移过程中硬件实时自动更新，完成一个链表节点传输后硬件自动加载下一个节点地址。
DMAC_Cm LLI	通道 m 的链表项寄存器。	0x108+dmac_m*0x10	表示链表项，是链表项地址的 bit[31:2]（链表项地址的 bit[1:0]为 0）。
DMAC_Cm CONTROL	通道 m 的控制寄存器。	0x010C+dmac_m*0x0010	-
DMAC_Cm CONFIG	通道 m 的配置寄存器。	0x0300+dmac_m*0x0004	-

7.5.2 SYSCTRL

以下寄存器 XEMPTY_STATUS（偏移地址 0x134），AXI Slave Empty Status。

各个字段含义如表 7-3 所示。

表7-3 特殊复位寄存器

字段	字段名称	说明
31:8	reserved	保留。



字段	字段名称	说明
7	pcie1_xsemp_timeout	PCIe1 AXI Slave 排空超时。 0: 没有超时; 1: 超时。
6	pcie0_xsemp_timeout	PCIe0 Slave 排空超时。 0: 没有超时; 1: 超时。
5	bmc_unse_xsemp_timeout	协处理模式非安全域 AXI Slave 排空超时。 0: 没有超时; 1: 超时。
4	bmc_xsemp_timeout	协处理模式域 AXI Slave 排空超时。 0: 没有超时; 1: 超时。
3:0	reserved	保留。

7.5.3 SEC_CSR

SEC_CSR 所有寄存器都是芯片域复位。

7.5.4 PCIE

PCIE 的表 7-4 配置寄存器内容是放在 RAM 里面的，在复位的时候无法清除存储值，只有下电才可以清除数据。

表7-4 PCIE 特殊复位寄存器

名称	用途	偏移地址	说明
DMAWRITEDONEIMWRADRESSLOW	DMA IMWR 远端低地址寄存器	0x09D0	DMA 使用此字段为 Done IMWr TLP 地址的生成【31:0】位。 注意：位【1:0】必须为“00”，因为此地址必须 DWORD 对齐。
DMAWRITEDONEIMWRAD	DMA IMWR 远	0x09D4	DMA 使用此字段为 Done IMWr



名称	用途	偏移地址	说明
DRESSHIGH	端高地址寄存器		TLP 地址的生成【63:32】位。
DMAWRITEA BORTIMWRA DDRESSLOW	DMA 写取消远 端低地址寄存器	0x09D8	DMA 使用此字段为 Abort IMWr TLP 地址的生成【31:0】 位。 注意：位【1:0】必须为“00”， 因此地址必须 DWORD 对齐。
DMAWRITEA BORTIMWRA DDRESSHIGH	DMA 写取消远 端高地址寄存器	0x09DC	DMA 使用此字段为 Abort IMWr TLP 地址的生成【63:32】 位。
DMAWRITEC HANNEL0IM WRDATA	DMA IMWR 写 通道数据寄存器	0x09E0	DMA 使用此字段为写通道 0 生成 Done or Abort IMWr TLPs 数据字 段。所有中断使用相同的数据类 型。
DMAREADD ONEIMWRAD DRESSLOW	DMA IMWR 远 端低地址寄存器	0x0A3C	DMA 使用此字段为 Done IMWr TLP 地址的生成【31:0】位。 注意：位【1:0】必须为“00”， 因此地址必须 DWORD 对齐。
DMAREADD ONEIMWRAD DRESSHIGH	DMA IMWR 远 端高地址寄存器	0x0A40	DMA 使用此字段为 Done IMWr TLP 地址的生成【63:32】位。
DMAREADAB ORTIMWRAD DRESSLOW	DMA 写取消远 端低地址寄存器	0x0A44	DMA 使用此字段为 Abort IMWr TLP 地址的生成【31:0】 位。 注意：位【1:0】必须为“00”， 因此地址必须 DWORD 对齐。
DMAREADAB ORTIMWRAD DRESSHIGH	DMA 写取消远 端高地址寄存器	0x0A48	DMA 使用此字段为 Done IMWr TLP 地址的生成【63:32】位。
DMAREADCH ANNELOIMW RDATA	DMA IMWR 写 通道数据寄存器	0x0A4C	DMA 使用此字段为读通道 0 生成 Done or Abort IMWr TLPs 数据字 段。所有中断使用相同的数据类 型。



名称	用途	偏移地址	说明
DMACHANNELCONTROLREGISTER	DMA 通道控制寄存器	0x0A70	地址转换 TLP 报头位 (Address Translation TLP Header Bit (AT))
DMATRANSFERSIZE	DMA 传输大小寄存器	0x0A78	<p>DMA 传输大小。</p> <p>使用 DMA 传输的大小对该寄存器进行编程。</p> <p>最大 DMA 传输大小为 4 GB。最小传输大小为一个字节 (0x1)。</p> <p>当 DMA 写入和读取通道传输进行时, 此字段由 DMA 自动递减。此字段指示剩余要传输的字节数。当所有字节都传输成功时, 当前传输大小为零。</p> <p>在 LL 模式下, DMA 用 LL 元素的相应 DWORD 覆盖此值。</p>
DMASARLOW	DMA 源低地址寄存器	0x0A7C	<p>源地址寄存器 (低 32 位)</p> <p>【1:0】位必须为'00', 因为此地址必须 DWORD 对齐。读取的下一个地址。DMA 会随着 DMA 传输的进行而递增 SAR。SAR 始终是 DWORD 对齐的, 即使传输大小是字节对齐的。</p> <p>在 LL 模式下, DMA 将用 LL 元素的相应 DWORD 覆盖它。</p> <p>DMA 读取 SAR 是远程内存的地址。</p> <p>DMA 写 SAR 是本地内存的地址。</p>
DMASARHIGH	DMA 源高地址寄存器	0x0A80	<p>源地址寄存器 (高 32 位)</p> <p>在 LL 模式下, DMA 用 LL 元素的相应 DWORD 覆盖此值。</p>
DMADARLOW	DMA 目的低地址寄存器	0x0A84	目标地址寄存器 (低 32 位)。



名称	用途	偏移地址	说明
	址寄存器		<p>【1:0】位必须为'00'，因为此地址必须 DWORD 对齐。下一个要写入的地址。DMA 在 DMA 传输时增加 DAR。</p> <p>DAR 始终是 DWORD 对齐的，即使传输大小是字节对齐的。</p> <p>在 LL 模式下，DMA 将用 LL 元素的相应 DWORD 覆盖它。</p> <p>DMA 读取 DAR 是本地内存的地址。</p> <p>DMA 写 DAR 是远程存储器的地址。</p>
DMADARHIG H	DMA 目的高地 址寄存器	0x0A88	<p>目的地址寄存器（高 32 位）。</p> <p>在 LL 模式下，DMA 用 LL 元素的相应 DWORD 覆盖此值。</p>
DMALINKEDL ISTPOINTER LOW	DMA 链表指针 低地址寄存器	0x0A8C	<p>链表传输列表在本地内存中的低位地址。仅在链表模式下使用。</p> <p>以在上一个元素被消费后指向传输列表中的下一个元素时由 DMA 更新。</p> <ul style="list-style-type: none">■当前元素为数据元素时，此字段增加 6。■当前元素是链接元素时，该字段被元素的 LL 元素指针覆盖。



8 DDRC

8.1 概述

DDRC (DDR3/4 DDR SDRAM Controller) 实现对 DDR3/4 DRAM 的存取控制，是整个芯片的数据存储中心，Hi3093 包括 1 组 DDRC，DDRC 里面包含 AXI_IF、QOSBUF、DMC。DMC 完成系统内部总线操作到 DFI 的时序转换，并配合系统内部总线上的 QOS 信息进行流量管理。

8.2 特性

DDRC 具有以下特性：

- 支持 JESD79-4B DDR4。
- 接口数据速率 DDR4：2133/2400 Mbps 可调节；对应的 DDRC 时钟速率为 533/600MHz，典型频点为 2133Mbps。
- DDR4 单个颗粒容量最大只支持到 16Gb (X8/X16 均适用)。
- 支持 X8，X16 器件颗粒，不支持不同数据位宽颗粒混用。
- 只支持 DDR Burst8 模式。
- DDRC 支持 1 Ranks。
- 支持 DMC 通道内部地址到 DDR 颗粒地址的 RRBC、RRCBC 映射模式。
- 支持 16bit DDR 数据位宽。
- 支持带内 ECC，支持 SEC/DED ECC 校验，支持出错场景下，第一次错误信息的记录（出错地址，期望数据、实际数据），其中带内 ECC 规格下，对于可用颗粒容量及效率带宽均会有影响。



- DDRC 不支持 ECC Write back，当读数据出现 1BIT ECC 错误时，需要软件来完成把纠正后的正确数据回写到 DDR 颗粒中。
- 支持 DDR 自动刷新，刷新周期可配置，支持 Postponed 模式且每次刷新的连续发送刷新的命令次数(1, 2, 4, 8)可配置。
- 支持 DDR 器件自刷新；以用于在 DDR 子系统在非掉电复位，DDR 数据不丢失的规格下，DDR 器件可进入自刷新来保证 DDR 数据不丢失。
- 支持 DDR 器件 Power Down，由软件通过配置寄存器使能。
- 支持 QOS，可以配合系统总线送过来的 QOS 信息进行流量管理；DDRC 可以根据 QOS 信息或者命令 ID 映射 Timeout 防止低优先级访问饿死问题。
- 支持 DDR4 的 Bank Group，通过有效的利用 Bank Group 来提高 DDR 的利用率。
- 支持 DDR4 的 DBI 功能，写 DBI 与 DM 功能互斥。
- 支持 ZQ 自校准。
- 支持 SEC 规格，整个地址空间上，不同的地址区间映射到了各自不同的权限属性。权限属性分为“命令的安全属性权限”。只有获得这种权限的命令，才可以正常访问 MDDRC 中对应的地址区间，DDR 提供安全属性，能够划分 10 个安全区域，总大小不小于 100MB，可配置。
- 支持 DDRC、DDRPHY 复位。
- DFX001：支持 Offline EXMBIST 测试，不支持 On line MBIST 测试。
- DFX002：支持对读写操作数量、基于 1 个 12Bits ID（可使用 ID Mask 映射任意几位 ID）进行读写操作延迟统计（读写操作最大最小延时，所有读写命令积累延时）以及 DDR 带宽利用率性能信息进行统计。
- DFX003：DDRC 的配置寄存器可以全部可读写。
- DFX004：支持 Debug Bus 接口，该接口可配置 DDRC 所有寄存器，可完成包括 DDR MBIST，DDRC 和 PHY 寄存器配置信息及统计信息的操作。
- DFX005：DDRC 能够检测 PHY 接口异常，并上报错误状态。
- DFX006：支持 DDRPHY Loop Back 功能，支持板级和 ATE 级的测试，I/O Pad 侧和 I/O Core 侧 Loop Back 可选择。
- DFX007：支持对 DDR4 的 Address parity 错误注入。
- DFX008：支持对 DDR4 的数据 CRC 错误注入。
- DFX009：针对内部 SRAM 进行 ECC 校验，并可以进行 1bit 或 2bit 错误注入，其中 WS MEM ECC 规格与 DDR ECC 功能耦合。
- DFX010：ECC 单 bit 错误可以定位到是具体 bit，即具体的 dq。



- RAS001: 支持 DDR4 的 CRC 功能, 寄存器可配置。
- RAS002: 支持 CA parity 和 CRC 的错误重传。
- RAS003: 支持 Data Scrambling, 将数据用 16 位地址线加扰写入 DRAM。
- RAS004: 支持 DDR4 的地址线 Parity 功能, 对命令/地址线的奇偶校验保护; 对数据线进行 CRC 保护。

8.3 关键术语

表 8-1 中的术语将在本章节表示特殊的含义。

表8-1 DDRC 关键术语描述

术语	描述
SEC/DED	Single error correct/Double error detect, 1bit 纠错, 2bit 检错。
PQ	Priority Queuing, 绝对优先级调度。
QOS	Quantity of service, 服务质量。
ODT	On die Terminate, 片上终端匹配。

8.4 遵循协议

JESD79-4B DDR4

8.5 功能描述

8.5.1 相关模块

在系统中, DDRC 模块会和下面多个模块进行交互, 分别描述如下:

- 系统 Master (CPU 等): DDRC 通过 Master 得到写命令和写数据访问 DDR 内存; 同时通过 Master 的读命令读取 DDR 内存数据返回给 Master。
- Debug Bus: 在 Debug 模式下, DDRC 可以通过 Debug bus 对 DDRC 全空间寄存器进行读写, 完成 DDR memory 初始化, DDR memory BIST 等功能。



- APB Bus: 在 APB 模式下, DDRC 可以通过 APB bus 对 DDRC 全空间寄存器进行读写, 完成 DDR memory 初始化, DDR memory BIST 等功能。

8.5.2 DDRC 支持的器件容量空间以及地址映射

8.5.2.1 DDRC 支持的器件容量

DDRC 支持外接 16bit 器件, DDRC 最大支持 1 个 Rank, DDRC 最大支持 4GB 的地址空间。

表8-2 16bit DDR4 器件组合

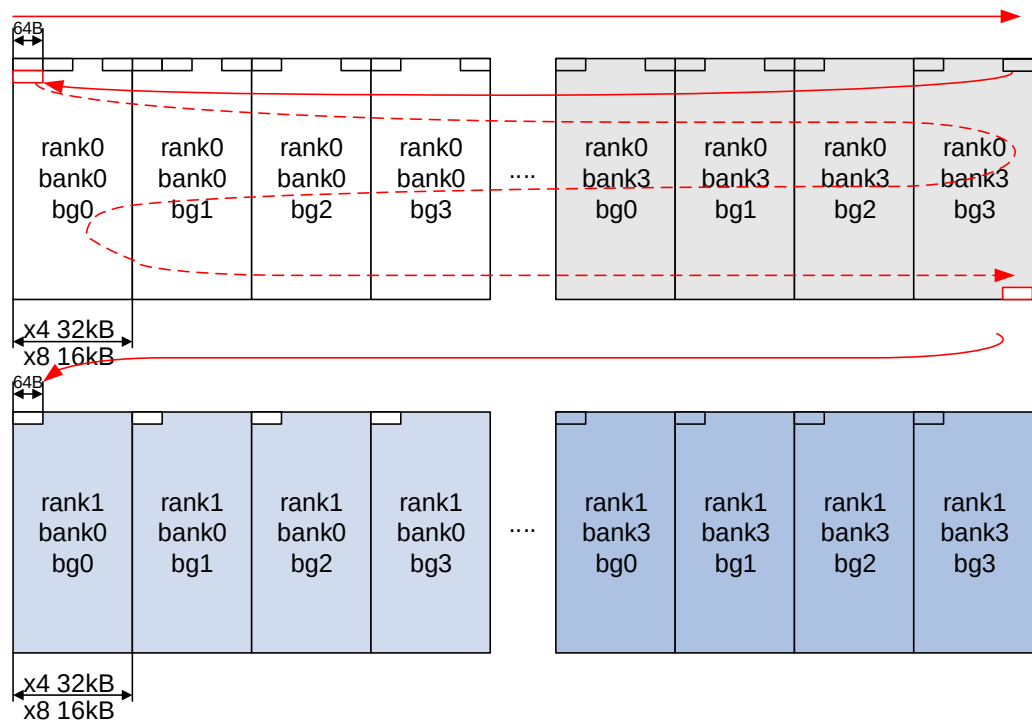
单 Rank 容量	单片器件类型	器件数量
1GB	512Mb*16	1
2GB	1Gb*8	2
2GB	1Gb*16	1
4GB	2Gb*8	2

8.5.2.2 DDR4 独立通道模式下地址映射

- RANK-ROW-BANK-COL 映射模式

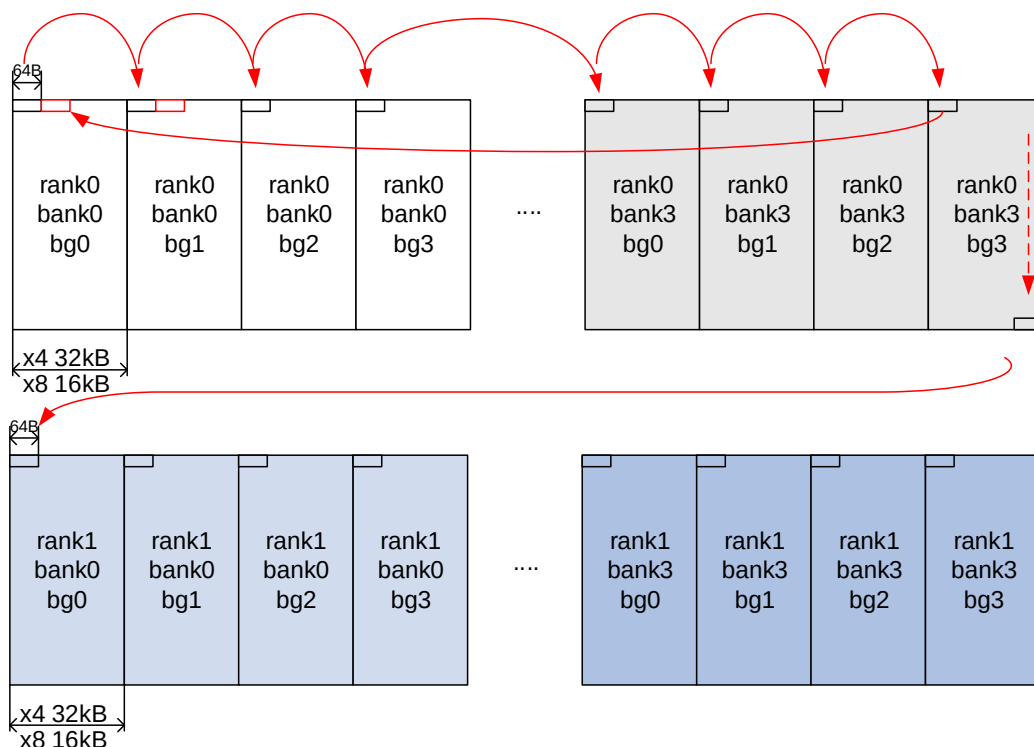


图8-1 DDR4 独立通道 RRBC 映射示意图



- RANK-ROW-COL-BANK-COL 映射模式

图8-2 DDR4 独立通道 RRCBC 映射示意图



须知

RRBC 的映射方式适用于访问 DDR 的地址空间是连续变化的，但是程序运行起来很难保证访问 DDR 的地址空间是连续变化的，容易出现较高的 bank 打开和关闭的概率，造成访问 DDR 的效率较低；

RRCBC 的映射方式通过将 bank 地址映射到更低位 (col 地址之间)，这样程序运行起来的时候可以同时打开更多的 bank，减小了 bank 打开和关闭的概率，也就提高了 DDR 的访问效率；

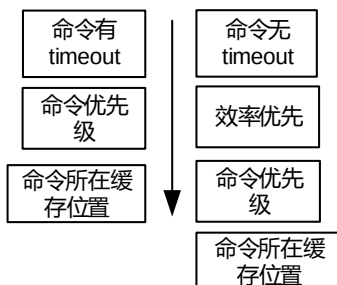
因此，单 Rank 的时候推荐适用 RRCBC 映射方式。

8.5.2.3 DDRC 命令调度策略

DDRC 根据 master 命令的 QOS 优先级给这个命令分配 timeout 值，timeout 值有两种，优先级小于等于 N 的访问分配的 timeout 值为 100 (可配置) DDRC tCK，优先级大于 N 的访问分配的 timeout 值为 1000 (可配置) DDRC tCK。命令进入到 DDRC 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。

DMC 可以缓存 12 个读写指令，DMC 调度方式如图 8-3 所示。

图8-3 DDRC 调度示意图



DDRC 将命令按照 PQ 算法 timeout，效率，优先级，命令所在缓存位置三种情况进行调度。

- 当有命令 timeout 的调度方法：
命令 timeout，则 timeout 的命令优先被调度。
若 timeout 不能够裁决出调度命令，根据命令优先级进行调度。
若命令优先级不能裁决出调度命令，根据命令所在缓存位置进行调度。
- 当没有命令 timeout 的调度方法：
若没有命令 timeout，则优先按照效率进行调度。
如果效率不能够裁决出命令，按照命令优先级进行调度。
命令优先级按照 QOS15>QOS14>QOS13...>QOS0 排列。

须知

DDRC 的调度只能解决 DDR 访问时序满足时的多个请求的调度，在高优先级请求 DDR 时序没有满足而低优先级请求 DDR 时序满足，DDRC 会按照效率优先调度低优先级的命令，timeout 的请求可以在请求没有满足 DDR 时序的情况下参与调度，但是此应用会降低 DDR 带宽的利用率，频繁的 timeout 命令会使得 DDR 带宽利用率下降。

每个 DMC 的命令有四种状态：

- Invalid：表示命令无效。

- Waiting: 表示命令在等待状态。当前命令与其他命令存在同 ID 读冲突, 或者地址一致性冲突, 或者被其他命令的 timeout 强行终止, 或者被 DDR 功能模块如 auto refresh/update/power down/sref refresh 等强行终止时, 进入 waiting 状态。
- Preparing: 表示命令进入 DDR 读写准备状态; 这个阶段将发送对应的 active、precharge 等 DDR 操作。
- Executing: 表示命令进入 DDR 读写状态。

为了提高 DDR 接口的效率, 只有 DMC 命令处于 preparing 和 executing 状态才可能提出仲裁请求。如果在其他状态提出仲裁请求, 而本身又不能发送任何命令, 就会浪费仲裁时隙, 降低效率。

调度的几种典型场景如图 8-4~图 8-7 所示。

图8-4 优先级调度场景

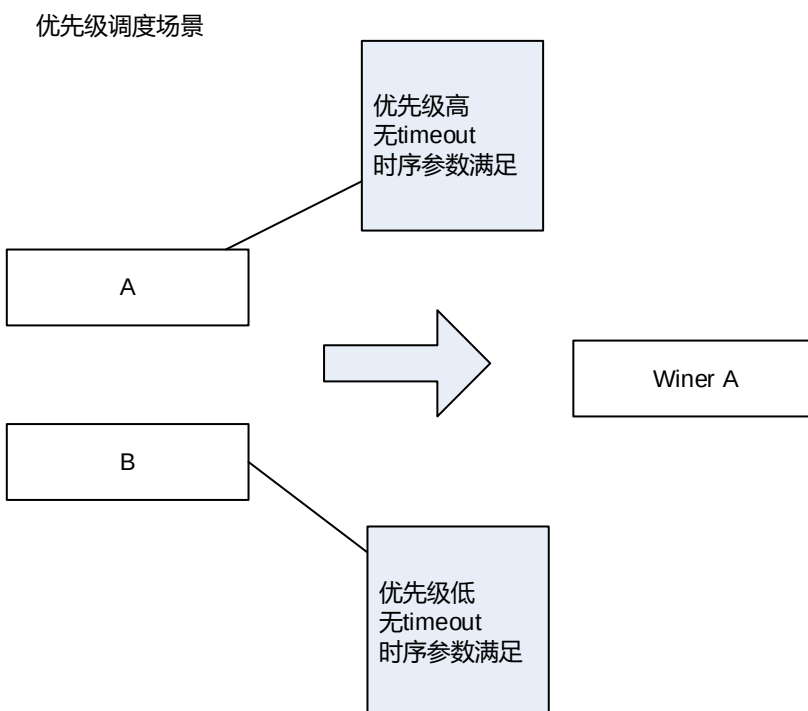




图8-5 效率优先调度场景

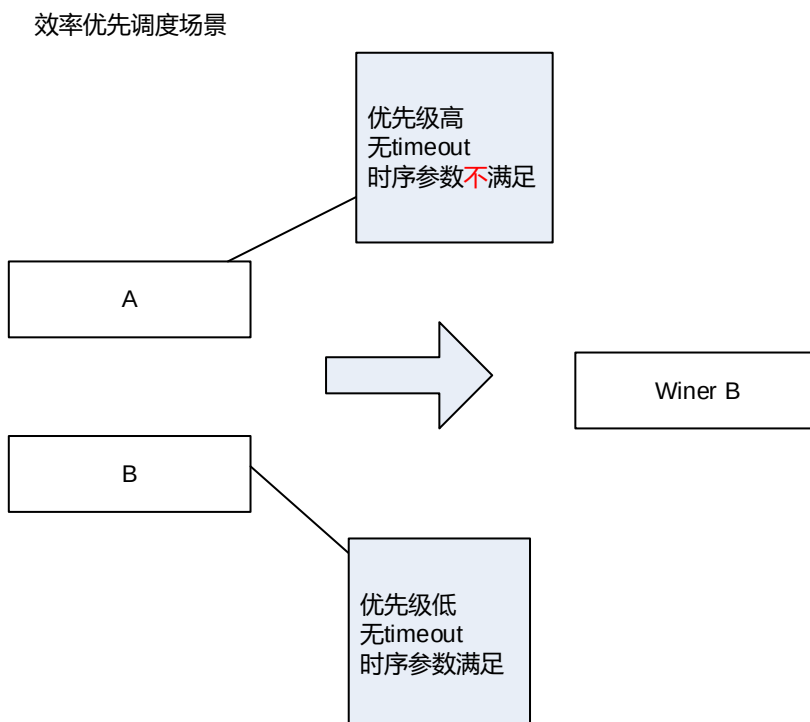




图8-6 timeout 调度场景

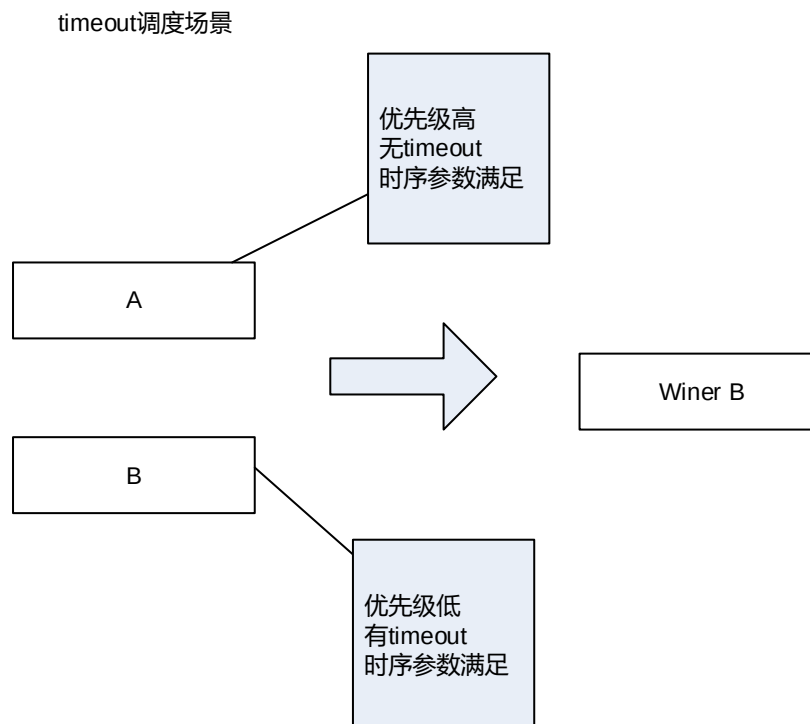
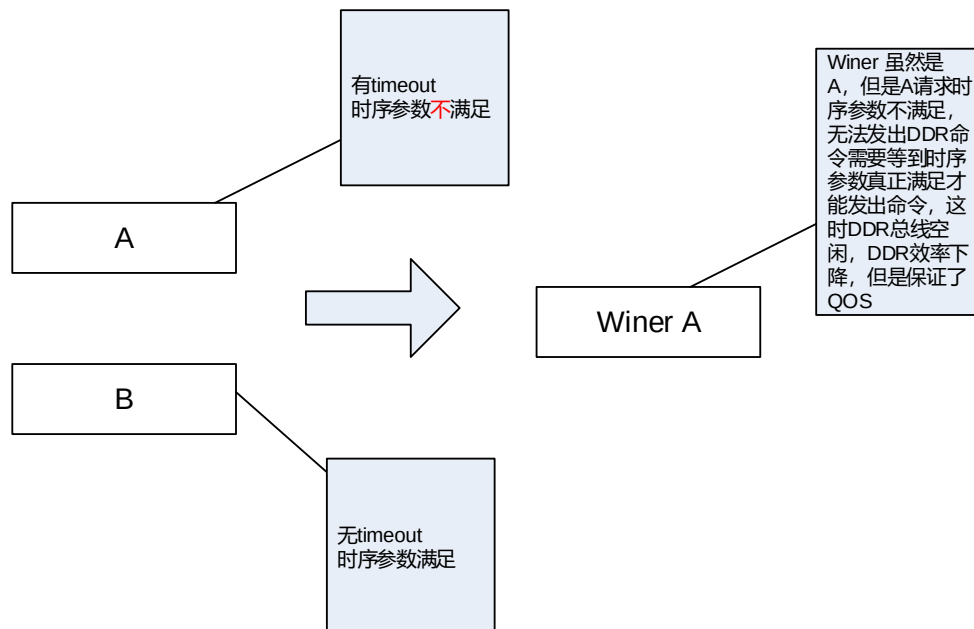


图8-7 timeout 调度场景 2

timeout调度场景 2



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。

8.5.2.4 DDR Memory ECC 保护

DDRC 支持 64/8bit 的 SEC/DED ECC 校验, 在 ECC 保护使能的模式下, 如果发现 1bit ECC 数据出错, DDRC 会将纠正的数据返回给系统 Master, 如果这个时候配置了 ecc writeback 使能, 会将正确数据回写入 DDR SDRAM。如果是 2bit 数据出错, DDRC 没有纠错能力, 会将错误数据直接返回给系统 Master, 并不会对 DDR SDRAM 进行数据回写, 需要系统读 ECC 错误地址寄存器, 对 DDR SDRAM 进行数据回写。

1bit ECC 数据出错的时候, DDRC 寄存器会记录 ECC 单 bit 出错的地址, 错误数据以及期待数据, 统计单 bit 出错的次数, 并且上报单 bit 错误中断。

2bit ECC 数据出错的时候, DDRC 寄存器会记录 ECC 2bit 出错的地址, 错误数据, 统计 2bit 出错的次数, 并且上报 2bit 错误中断。



DDRC 同时支持 ECC 错误数据 1bit/2bit 注入，支持连续错误注入，以及单次触发错误注入。

8.5.2.5 ECC 旁路访问

DDRC 支持 ECC 的旁路访问，通过配置 DDRC 的 SFC (Soft ware config) 寄存器可以直接访问 ECC memory 和 Data memory 的数据。

软件可以配置写入的数据的地址和数据，如果是读操作，可以从相应寄存器得到读数据值。

在 ECC 旁路访问过程中，ECC 生成模块和 ECC 检测模块被 Bypass，不会根据旁路访问的数据生成 ECC 码，以及对回读数据进行 ECC 校验。

8.5.2.6 带内 ECC 规格

带内 ECC 模块主要特性如下：

- 支持通过寄存器配置使能带内 ECC 功能。若使能关闭，总线上的信息与数据在模块内透传。
- 带有带内 ECC 的 DDR 空间，其地址容量利用率为 87.5%（即其中 7/8 的空间存放有效数据，1/8 空间存放 ECC 校验位）。
- 带内 ECC 可基于 AXI Port 使能，由寄存器配置。
- 除 ECC 校验位带来的空间压缩外，地址映射对于访问源不可见。
- 支持检测到 ECC 的 1bit 错误，可矫正该错误，并上报中断，计数并记录原始出错数据、出错地址与纠正后数据，纠正后数据回写需由软件实现。
- 支持检测到 ECC 的 2bit 错误，并上报中断，计数并记录原始出错数据与出错地址。

8.5.2.7 Memory BIST

DDRC 内部集成了美研开发的 EXMBIST 模块，当前版本仅支持 Off line bist 功能，不支持 On line bist 功能。

EXMBIST 支持灵活多变的算法，如：March C-，Checkerboard，MOVI，March LR，Butterfly，March-b/y/g/l/u，GALPAT 等。

用户可以指定测试 memory 的空间范围和大小。

支持多种不同的背景数据的生成。用户可以指定数据码型，而不仅仅限于全 0、全 1、5、A 这样的数据码型。



该模块应用非常广泛，可以用于硬件加速算法实现 BSP_TestBitspreadComparison、BITFLIP_COMPARISON 等算法，加快内存扫描的速度。

Memory BIST 支持如下错误检测：

- 固定逻辑故障 SAF - Stuck-at-Fault
- 固定开路故障 SOF - Stuck-Open Fault
- 状态转换故障 TF - Transition Fault
- 地址译码故障 AF - Address-Decoder Fault
- 状态耦合故障 CFst - State Coupling Fault
- 反转耦合故障 CFin - Inversion Coupling Fault
- 等幂耦合故障 CFid - Idempotent Coupling Fault
- 数据保持故障 DRF - Data Retention Fault
- 地址译码开路故障 ADOF - Address Decode Open Fault
- 伪读破坏故障 DRDF - Deceptive RDF
- 非破坏写故障 URDWF - Un-Restored Destructive Write Fault
- 慢写驱动故障 SWDF - Slow Write Driver Fault
- 动态两个单元错误读故障 d2clRF - Dynamic Two-cell Incorrect Read Fault

在进行 Memory BIST 操作之前，需要配置 DDRC 和 DDRPHY 的相应寄存器对 DDR memory 和 DDR PHY 进行正确的初始化。否则 Memory BIST 不能够正常执行。

8.5.2.8 Memory 内容初始化

DDRC 可以通过寄存器使能 DDR Memory 内容初始化，使能此功能 DDRC 可以将定义地址内的 memory 内容初始化为寄存器定义的 Pattern 内容，防止未写入地址数据读出时 ECC 误报错。

8.5.2.9 DDR 自动刷新

DDRC 支持 DDR 的 Postpone 的刷新方式，可配置每次连续发送自动刷新命令次数为 (1, 2, 4, 8)。

一次刷新的发送命令越少越能减少 DDR 命令访问的平均延迟，但是会降低整个 DDR 的访问带宽。一次发送的命令越多越会增加 DDR 命令访问的平均延迟，但会增加整个 DDR 的访问带宽。



若 DDR 的频率为 800MHz，则 tRFC 为 240tCK，tRP 为 11tCK，tRFI 为 3.9us(3120tCK)。

假设每次刷新之前都需要 Precharge，所以一次刷新需要的时长为 $11+n*240tCK$ 。

表8-3 DDR 刷新命令数和带宽的关系

每次发送刷新命令数	刷新时长 (tCK)	需要刷新次数	系统总带宽 (tCK)	刷新带宽消耗 (刷新时长*需要刷新次数) /系统总带宽
1	251	8	24960	$251*8/24960=8.04\%$
2	491	4	24960	7.8%
4	971	2	24960	7.78%
8	1931	1	24960	7.73%

DDRC 对于不同 Rank 采用串行刷新的方式，刷新完一个 Rank 再刷新另一个 Rank，刷新一个 Rank 的同时，另一个 Rank 的读写访问不受影响。

8.5.2.10 DDR 自刷新功能

DDRC 支持 DDR 自刷新功能 (Self Refresh)，用于低功耗应用，或保证对 DDR 子系统软复位但是需要保证 DDR 颗粒中的数据信息不丢失。此功能使能后 DDRC 会反压 Master 的命令，并且将内部剩余的命令处理完成后，发送自刷新命令使得 DDR 器件进行自刷新状态对访问命令返回 DMC_CMD_ERROR 指示信号。

DDRC 在进入自刷新后 Master 的读命令来到不会返回假读数据，Master 写命令遇到 Error 后，需要将相应的写数据丢弃，为了保证自刷新后系统访问不会挂死，访问 DDRC 的 Master 需要等待写数据被 DDRC 正常接收后，再发送对应的写数据。

8.5.2.11 DDR Power Down 功能

DDRC 支持 DDR Power Down 功能。此功能可以在 DDR 访问闲时降低 DDR 器件的功耗，使能此功能 DDRC 会在内部命令排空后开始计数，如果等到系统配置计数值到达后仍然没有新的访问进入 DDRC，DDRC 会将 DDR 颗粒的 CKE 拉低，进入 DDR Power Down 模式。当新的命令到来或者自动刷新时间到达，DDRC 会将 CKE 拉高退出低功耗模式，进行读写或者自动刷新操作。



8.5.2.12 DDR ZQ 自校准功能

随着环境、温度、电压的变化，器件的驱动和负载匹配也会随之变化，为了保证信号质量，DDR 器件提供了 ZQ_CL 和 ZQ_CS 命令进行 ZQ 校准。

DDRC 支持 ZQ_CL 和 ZQ_CS 命令，ZQ_CL 是长校准命令，用于 Full Calibration 驱动和配置，命令周期大约为 512tCK，一半用于 DDR 器件初始化过程，第一次校准 ZQ_CS 为短校准命令，校准命令时间比 ZQ_CL 短，命令周期大约为 64tCK，但是一次校准的能力比 ZQ_CL 小，每次 ZQ_CS 最小约 0.5% Impedance Error。ZQ_CS 命令一般用于 ZQ_CL 完成后的 ZQ 周期性校准。

DDRC 的 ZQ_CS 校准周期是寄存器可配置的，用户可以根据当前的应用环境对 ZQ_CS 校准周期进行配置，其计算方法如下：

$$\frac{0.5\%}{(TSens \times Tdriftrate) + (VSens \times Vdriftrate)}$$

其中：

$TSens = \max(dRTTdT, dRONdTM)$

$VSens = \max(dRTTdV, dRONdVM)$

TSens 为电阻温度敏感参数（如 1.5% °C），VSens 为电阻电压敏感参数（如 0.15%），其值可由器件手册得到。

Tdriftrate 为温度偏移率（单位 °C/sec），Vdriftrate 为电压偏移率（单位 mV/sec），这个值由系统和环境决定。

8.5.2.13 DDRC 流量 Latency 统计功能

DDRC 支持流量统计功能，可以统计接口读写流量，用以收集当前流量信息以决定是否需要进行 DFS 提高或者降低 DDRC 的频率。

- 支持指定 ID 访问读/写流量单独统计。
- 支持总体读/写流量统计。
- 支持命令 latency 检测。
- 支持指定 ID 访问读写最大 latency 统计，以及累积 latency 统计。

支持 DDR 接口利用率统计，统计计数器支持连续统计和单次统计，当为连续统计时计数器为非饱和计数器，计数器记到最大值后会卷绕，方便系统进行连续统计，所以需要系统在计数器未卷绕之前将计数器值读出。当为单次统计时，统计计数器在进行统计时间到达后停止统计，系统可以用此功能统计瞬时的流量以及 latency。



8.5.2.14 DDRPHY 异常检测功能

DDRC 可以对 DDRPHY 以下情况进行检测并上报中断：

- 读命令发出后，读数据不返回。

8.5.2.15 Address Parity 功能

在使用 DDR4 时，DMC 会随路地址信号送出一个偶校验信号，DDR 收到地址以后，会进行校验，并返回校验结果。如果发现有命令校验错误。对于 DDR4，可以由 REC 模块进行命令恢复。

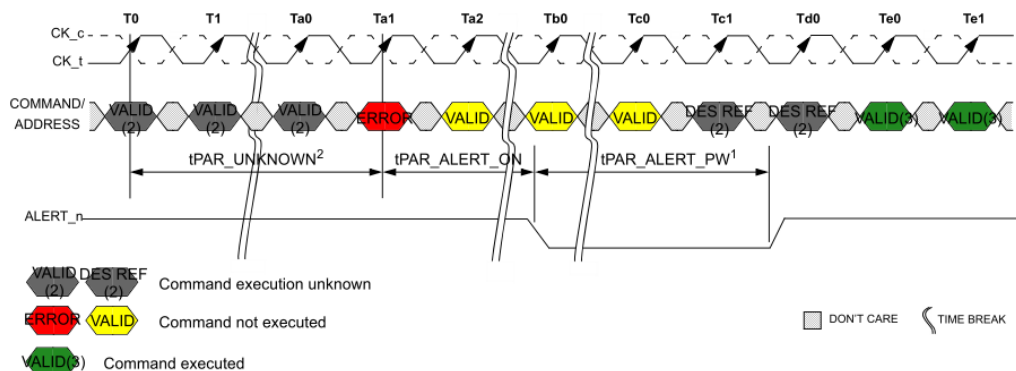
DDR4 偶校验由 ACT_n, RAS_n/A16, CAS_n/A15, WE_n/A14, BG0-BG1, BA0-BA1, A17-A0 产生。

当 DDR4 颗粒检测到一个 CA parity 错误后，会执行以下操作：

- 忽略该错误的命令。并且在此命令之前的 tPAR_UNKNOWN 窗口内的命令不一定会被执行。在这个窗口以内的读命令会被丢弃，DRAM 器件将不返回读数据。
- 将错误的命令和地址保存到 MPR 寄存器。
- 将模式寄存器的 Parity Error Status 标志位置 1。此标志位将会在 ALERT_n 信号释放以前被置位 (tPAR_ALERT_ON+tPAR_ALERT_PWmin)。
- 输出 ALERT_n 信号为低，并维持 tPAR_ALERT_ON 时间。
- 等待所有正在执行的命令完成。这些命令是在该错误命令前的 tPAR_UNKNOWN 窗口内接收的。假如在自刷新退出的过程中 (tXS_FAST 和 tXS 窗口内) 有错误命令，那么 ALERT_n 信号将可能延时撤离，因为此时在器件内部可能在执行 refresh 命令。
- 等待 tRAS_min 后关闭所有的 bank。DDR 器件将不执行在 tPAR_ALERT_ON+tPAR_ALERT_PW 窗口内的任何命令。
- 当 tPAR_ALERT_PW_min 时间满足以后，DDR 器件将撤销 ALERT_n 信号。
- 当 DDR 器件返回到 precharge 状态以后，DDR 器件将撤销 ALERT_n 信号。
- 当 tPAR_ALERT_ON+tPAR_ALERT_PW 时间满足后，DDR 器件可以接收新的命令。但是 DDR 控制器清空错误标志后（通过将 parity error status 寄存器写 0 来），才能发送新的命令。在错误标志没有清零前，DDR 器件将不会做 parity 检查，所有的错误命令都可能被执行。
- 在错误窗口以内的 refresh 命令也可能被丢弃，所以建议控制器再发送一次 autorefresh 命令。

- Parity error status 状态可以在 tPAR_ALERT_ON+tPAR_ALERT_PW 后任何时间被读取，用来判断哪个 DDR 器件有错。

图8-8 DDR4 CA Parity 时序图

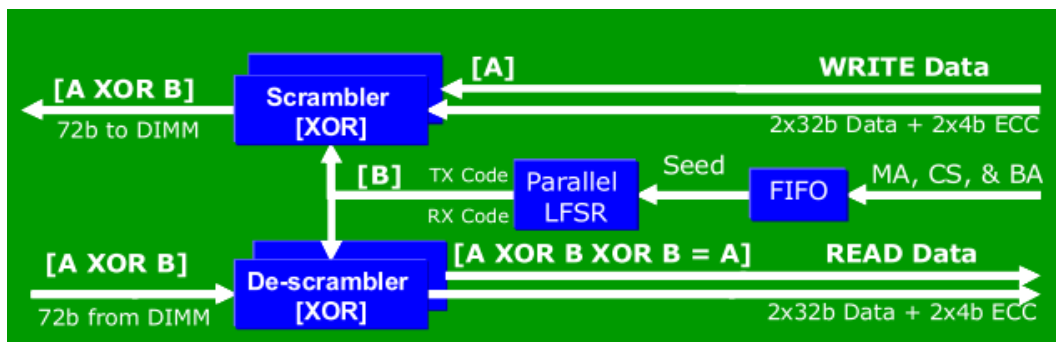


NOTE 1 DRAM is emptying queues, Precharge All and parity checking off until Parity Error Status bit cleared.
NOTE 2 Command execution is unknown the corresponding DRAM internal state change may or may not occur. The DRAM Controller should consider both cases and make sure that the command sequence meets the specifications.
NOTE 3 Normal operation with parity latency(CA Parity Persistent Error Mode disabled). Parity checking off until Parity Error Status bit cleared.

8.5.2.16 Data Scrambling 功能

DMC 可以将数据通过加扰，避免连续的 0 或 1 出现。

图8-9 DataScrambling 框图



由于设计原因 DataScrambling 使能不支持动态配置，配置使能后中途不能关闭该使能，除非重新复位。SEC 功能

- 命令的访问权限包括命令的安全属性及 MasterID（下文简称 MID）匹配两部分。
- 安全属性包括安全读/安全写/非安全读/非安全写四种，支持安全权限反转。
- 支持最多 16 个可叠加的拥有各自访问权限的地址区域，多个地址区域将整个地址空间划分为若干个地址区间，每个地址区间只允许具有相应权限的命令访问。



- 地址区域地址重叠时，以高优先级地址区域的权限为当前地址的命令访问权限。
- 每个地址区域可等分为 8 个可单独使能的子区域，若子区域未使能，则对应的访问权限由此地址区间内次高优先级的地址区域决定。
- 地址区间划分粒度为 32KB，考虑到子区域特性，划分粒度为 4KB。
- 部分关键寄存器支持寄存器访问保护，可被一次性锁定为只读，复位后才可被再次写入。
- 检测到越权命令时，可上报中断（高电平有效），并记录第一个越权命令的相应信息。
- 命令 outstanding 不受安全模块影响，其深度取决于 MDDRC 的支持深度。
- 模块 latency 为 1 个 DMC 时钟的时钟周期。
- 安全模块可整体屏蔽（RTL 级或寄存器级），MID 匹配功能可单独屏蔽（RTL 级或寄存器级）。

8.5.2.17 DDR 热复位数据保持功能

芯片热复位前，DDRC 让 DDR 进入自刷新状态，芯片复位时 DDR 中的数据不被复位，需要系统与 DDRC 共同配合实现热复位数据保持功能，实现如下：

- 步骤 1 热复位 DDR 前，系统发送 csysreq_n 请求给 DDRC。
- 步骤 2 系统保存 DDRPHY 的 training delay 值，保证复位后这部分 delay 数据也不会丢失。
- 步骤 3 DDRC 接收到 csysreq_n 请求后，DDRC 发起自刷新请求，设置 DDR 进入自刷新模式。
- 步骤 4 进入 DDR 自刷新。
- 步骤 5 DDR 进入自刷新状态 in_sref 后，通过将 csysack_n 信号拉低返回给系统。
- 步骤 6 系统检测到 csysack_n 的有效后，等待 1us 时间，生成 DDR 子系统复位信号，其中 csysack 状态需要系统寄存器可查。
- 步骤 7 如果系统超过 1ms 仍未检测到 csysack_n 有效后，则记录超时信息，同时生成 DDR 子系统复位信号。
- 步骤 8 配置 ioctl_lhen 为 1，让 IO 侧的 DDR_RESET CKE 为固定电平。
- 步骤 9 复位 DDRC/DDRPHY 后，解复位前 csysreq_n 有效状态清除（csysreq_n 置 1）。
- 步骤 10 DDR 子系统复位完成，系统时钟稳定后，再解复位 DDRC、DDRPHY。



步骤 11 重新配置 DDRC、DDRPHY 寄存器，将前面保存的 training delay 值重新 reload 进 DDRPHY。

步骤 12 配置 ioctl_lhen 为 0，退出 IO retention 模式。

步骤 13 通过配置让 DDR 退出自刷新模式，读出复位前的 DDR 数据。

---结束

8.6 DFX

8.6.1 可靠性

8.6.1.1 自恢复

- serr: DDR ECC 单 bit 错误，硬件自动纠错。
- sbram_serr: SBRAM 单 bit 错误，硬件自动纠错。
- recram_serr: RECRAM 单 bit 错误，硬件自动纠错。
- rpram_serr: RPRAM 单 bit 错误，硬件自动纠错。
- qosburam_serr: QOSBUF MEM 单 bit 错误，硬件自动纠错。

8.6.1.2 非自恢复

- merr: DDR ECC 多 bit 错误，上报中断，需要软件复位 DDR，并且重新做初始化。
- sref_err: 自刷新接口有命令访问，命令被丢弃，需要软件复位 DDR，并且重新做初始化。
- dimm_parity: DDR DIMM 奇偶校验错误，需要软件复位 DDR，并且重新做初始化；或者在 DDR4 时作为 CRC 校验错误。
- rdtimeout: DDRPHY 读数据超时，需要软件复位 DDR，并且重新做初始化。
- aref_alarm: DDR AREF 命令丢弃，如果连续发生多次可能造成数据丢失，需要软件复位 DDR，并且重新做初始化。
- sbram_merr: SBRAM 多 bit 错误，需要软件复位 DDR，并且重新做初始化。
- recram_merr: RECRAM 多 bit 错误，需要软件复位 DDR，并且重新做初始化。
- rpram_merr: RPRAM 多 bit 错误，需要软件复位 DDR，并且重新做初始化。



- qosbufram_merr: RPRAM 多 bit 错误, 需要软件复位 DDR, 并且重新做初始化。

8.6.2 故障预防

- 使能 ECC 功能, 出现 1bit ECC 错误会自动纠错, 出现 2bit ECC 错误只能检错。
- 使能 REC 重传功能, 出现 Alert 错误时命令自动重传。

8.6.3 模块维测

8.6.3.1 统计

DDRC 支持流量统计功能, 可以统计接口读写流量。

- 支持指定 ID 访问读/写流量单独统计。
- 支持总体读/写流量统计。

支持命令 latency 检测。

- 支持指定 ID 访问读写最大 Latency 统计, 以及累积 Latency 统计。

支持 DDR 接口利用率统计, 统计计数器支持连续统计和单次统计, 当为连续统计时计数器为非饱和计数器, 计数器记到最大值后会卷绕, 方便系统进行连续统计, 所以需要系统在计数器未卷绕之前将计数器值读出。当为单次统计时, 统计计数器在进行统计时间到达后停止统计, 系统可以用此功能统计瞬时的流量以及 Latency。

统计功能操作步骤如下:

- 步骤 1 通过 DDRC_CFG_PERF.perf_mode 配置统计的模式是连续触发还是单次触发, 如为单次触发还需要配置 perf_prd 域配置统计周期。
- 步骤 2 通过 DDRC_CFG_STAID 配置期望统计的 ID。
- 步骤 3 通过 DDRC_CFG_STAIDMSK 设置 ID 的 Mask 值, DDRC 会通过 sta_idmask&系统访问 ID 是否等于 DDRC_CFG_STAID 判断是否将当前访问纳入统计。使用这种方法可以统计多个 ID 的统计值。
- 步骤 4 配置 DDRC_CTRL_PERF.perf_en 使能统计。如为单次统计 perf_en 恢复为 0 时表示统计完成, 如为连续统计需要软件将 DDRC_CTRL_PERF.perf_en 写 0 停止统计。
- 步骤 5 通过 DDRC_HIS_FLUX_WR, DDRC_HIS_FLUX_RD, DDRC_HIS_FLUX_WCMD, DDRC_HIS_FLUX_RCMD, DDRC_HIS_FLUXID_WR, DDRC_HIS_FLUXID_RD, DDRC_HIS_FLUXID_WCMD, DDRC_HIS_FLUXID_RCMD,



DDRC_HIS_WLATCNT0, DDRC_HIS_WLATCNT1, DDRC_HIS_RLATCNT0, DDRC_HIS_RLATCNT1, DDRC_HIS_INHERE_RLAT_CNT 来观测统计结果。

----结束

8.6.3.2 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)。

- 提供 Debug 寄存器对 DDR 特定地址进行读写访问，能够访问 ECC 器件的数据。
- 提供 ECC 错误地址错误数据信息收集。
- 提供 Memory BIST 对 DDR SDRAM 进行内存硬件自检。



9 LocalBus

9.1 概述

LocalBus 对外提供异步静态存储器接口，可以连接 SRAM (Static Random Access Memory)、ROM (Read Only Memory)、NOR Flash 等异步静态存储器，用于实现系统启动、数据存储等功能。同时也可以用于连接带异步存储器接口（包括 Intel 时序和 Moto 时序）的控制芯片。

9.2 特性

LocalBus 控制器具有以下特性：

- 支持 4 个片选 (CS0~CS3)，每个片选独立配置存储器空间，独立配置时序参数。
 - CS0 支持的最大访问空间为 256MB；至少 128MB；
 - 片选、读使能、写使能的有效长度可配；
 - 片选有效电平之间的最小间隔可配；
 - 片选与读信号、写信号之间的间隔可配置。
- 数据位宽支持 16bit 或 8bit 可配置。
- 地址位宽 27bit。
- 支持防挂死特性。
- 支持 ALE 无效到数据输出时间可配置。
- 支持带 READY 和固定周期 2 种模式，每个片选独立配置。
 - 固定周期下根据内部寄存器配置的 Localbus 周期执行外部访问；



- READY 模式下，根据外部设备返回的 READY 信号握手方式执行外部访问。
- 支持地址线和数据线复用，减少外部管脚数量。
地址锁存信号 EBIALE 为高电平时，地址有效；EBIALE 为低电平时，数据有效。
- 支持地址偏移可配置。
- 支持 LocalBus Boot 功能。
 - 固定从 CS0 对应的 Flash 启动；
 - 启动时的数据位宽支持管脚可配置。

9.3 功能描述

9.3.1 功能原理

9.3.1.1 外部 Ready 模式以及 Ready 超时保护

支持外部 Ready 模式，通过配置片选控制寄存器 LB_CS0_CTRL bit[9] (物理地址为 0x08712020)，指示是否支持 Ready 模式。

在配置了支持外部 Ready 模式的情况下，配置 LB_CS0_CTRL bit[10] (物理地址为 0x08712020) 选择外部 Ready 的有效极性。配置 0、1 分别表示低、高电平有效。

在支持外部 Ready 模式下，为防止对接器件 Ready 长时间不响应，导致系统挂死，芯片支持超时保护。通过配置 LB_CS0_CTRL bit[16] (物理地址为 0x08712020)，关闭或开启超时保护功能。

9.3.1.2 在外部 Ready 模式下支持防挂死特性

在开启超时保护情况下，配置 LB_RDY_TOUT_0 bit[31:0] (物理地址为 0x087120AC)，超时保护时间。则 LocalBus 在片选有效后开始计数，如果计数到达 LB_RDY_TOUT_0 bit[31:0] 的配置值，未检测到有效的 Ready，则 Localbus 控制器向 ARM 内核送一个假的 Ready 信号结束本次操作，并上报中断给 CPU。同时控制器记录第一次超时的地址，超时地址通过 DBG_OVER_TIME_ADDR bit[31:0] (物理地址为 0x08712144) 地址可查询。

须知

上面只是介绍了 CS0 的情况，CS1~CS3 也同样支持。



9.3.2 数据和地址线序

支持数据和地址的线序选择：bit[0]为高位、还是 bit[0]为低位。通过寄存器配置 LB_CS0_CTRL[cf_data_order_0] (物理地址为 0x08712020，bit[7]) 选择。具体如表 9-1 所示。

表9-1 数据总线与物理管脚对应关系

总线 (MSB:LSB)	cf_data_order_0=0	cf_data_order_0=1
data[15:0]	PAD_LOCAL_BUS_ADDR bit[15:0]	PAD_LOCAL_BUS_ADDR bit[0:15]
data[7:0]	PAD_LOCAL_BUS_ADDR bit[7:0]	PAD_LOCAL_BUS_ADDR bit[0:7]

须知

- cf_data_order_0 与地址线序的关系如表 9-3 所示。
- 上面只是介绍了 CS0 的情况，CS1- CS3 也同样支持。
- Local bus 的数据和地址管脚复用。地址 27bit，数据 16bit，总共 27 根线，数据复用在低 16 根，总线由 ALE 信号指示当前为地址或者数据。这些管脚同时和其他功能有复用。

9.3.2.1 数据字节顺序

支持输出的字节顺序可调整，通过配置 LB_CS0_CTRL bit[8] (物理地址为 0x08712020) 选择，如表 9-2 所示。

表9-2 数据总线字节顺序关系

总线宽度	cf_byte_order_0 = 0	cf_byte_order_0 = 1
16bit	data[15:0]	{data[7:0],data[15:8]}
8bit	data[7:0]	data[7:0]



9.3.2.2 地址偏移

支持地址偏移，通过配置 LB_CS0_CTRL[cf_addr_offset_0] (物理地址为 0x08712020, bit[11]) 选择。cf_addr_offset_0 的不同配置对应的含义如表 9-3 所示。在不同配置下，与对接器件的连接方式如图 9-1、图 9-2 所示。

表9-3 地址总线与物理管脚对应关系 (cf_addr_offset=0, 1)

数据总线	地址总线 (cf_addr_offset=1)	地址总线 (cf_addr_offset=0)
8bit	addr[26:0]	addr[26:0]
16bit	{1'b0, addr[26:1]}	addr[26:0]

图9-1 8bit (cf_addr_offset=1) 或 8/16 (cf_addr_offset=0)

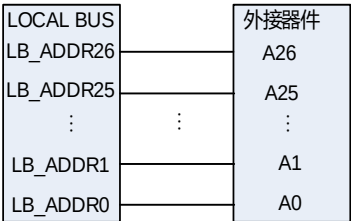
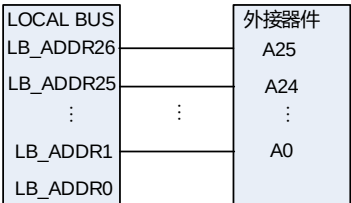


图9-2 16bit (cf_addr_offset=1)



当 cf_addr_offset=0 时，无论数据位宽是 8bit 还是 16bit，连接方式与图 9-1 相同。



9.3.2.3 上电管脚采样

芯片上电过程中，CS0 部分配置来自管脚，随后软件可以通过寄存器修改配置，如果软件不修改配置则保持上电管脚采样值不变。

表9-4 CS0 上电采样配置表

含义	管脚信号 (Pin)	寄存器
地址偏移	LB_ADDR_OFFSET	cf_addr_offset_0
数据接口位宽	LB_DATA_WID	cf_data_wid_0

须知

仅 CS0 上电部分配置可以来自管脚（用于通过 CS0 boot），CS1-3 不支持。

9.3.2.4 LocalBus 写保护功能

LocalBus 支持 2 种写保护功能：低级写保护和高级写保护。通过 LB_CTRL bit[15:0] (物理地址为 0x087120CC) 中的相应 bit 位来决定，例如地址段 0 的写保护使能，LB_CTRL bit[15:14] (物理地址为 0x087120CC) 配置为：

- 00：表示写保护功能无效；
- 01：表示低级写保护使能；
- 10：表示高级写保护使能，此时还需要配置写保护使能关键字 LB_HL_PROT_KEY0 bit[31:16] (物理地址为 0x08712130)。

写保护的地址空间可配置，起始地址通过 LB_PROT_ADDR_START_0 bit[31:16] (物理地址为 0x087120D0) 配置；结束地址通过 LB_PROT_ADDR_OVER_0 bit[31:16] (物理地址为 0x087120D4) 配置。保护的最小空间是 64KB。

如果写保护地址段中包含命令字地址，且需要对命令字地址可以进行写操作，需要配置命令字地址 LB_NOR_CMDi_ADDR(i=0-3) (偏移地址为 0x0110、0x118、0x120、0x128) 和相应的掩码 LB_NOR_CMDi_MASK(i=0-3) (偏移地址为 0x0114、0x11C、0x124、0x12C)，命令字地址和相应掩码共有四对，可根据写保护地址段的命令字数选择。



LocalBus 最多可以配置 8 个写保护地址段，如果使能了写保护功能，却对写保护地址进行了操作（命令字地址除外，命令字地址可以正常写入），会产生中断。

须知

LocalBus 写保护不只针对 NOR Flash，由于现有器件中只有 NOR Flash 涉及写保护，所以本处以 NOR Flash 为例说明。

NOR Flash 器件接口一般带有一个 WP#输入管脚，实现对 NOR Flash 的硬件写保护功能，芯片不提供 WP#输出管脚，单板可以通过 GPIO 来模拟。

9.3.3 灵活的时序配置

LocalBus 4 个 CS 对应的时序参数可独立配置；需要配置是 Intel 模式还是 Moto 模式，可直接根据具体器件配置出相应的时序。

须知

因上一版固定时序写 ALE 无效到数据有效间隔只有一个时钟周期，对单板设计有约束，本版本增加了间隔可配置，默认值为一个周期，如无单板设计约束可不配置，配置寄存器为 LB_CS0_T_SET2 bit[31:29]（物理地址 0x08712048），四个片选都可以分别配置，要求不大于 Tacs，否则会影响正常的读写时序。

9.3.3.1 LocalBus 时序参数配置

可配置的时序参数如表 9-5 所示。

表9-5 LocalBus 的可配置时序参数表

符号	说明	最小值	典型值	最大值	单位
Tacs	从地址有效到片选有效的时间。	1	2	32	Cycle
Tcsrp	在读操作时，表示片选有效时间。	2	8	511	Cycle
Taoe	地址有效到读使能（oe_n）有效的 时间。	1	3	32	Cycle



符号	说明	最小值	典型值	最大值	单位
Tcoen	如果读使能 (oe_n) 先于片选 (cs_n) 无效, 表示读使能 (oe_n) 无效到片选 (cs_n) 无效的时间; 如果片选 (cs_n) 先于读使能 (oe_n) 无效, 表示片选 (cs_n) 无效到读使能 (oe_n) 无效的时间。	0	0	63	Cycle
Tal	从地址信号有效到读写合一信号 (lbctl) 有效 (如: 默认为写, 假设为高, 则低有效) 时间。	1	1	32	Cycle
Tawe	从地址信号有效到写使能信号有效的时间。	1	3	32	Cycle
Tcln	如果读写合一信号 (lbctl) 先于片选(cs_n)无效, 表示从读写合一信号 (lbctl) 无效 (恢复默认值) 到片选 (cs_n) 无效时间; 如果片选 (cs_n) 先于读写合一信号 (lbctl) 无效, 表示从片选 (cs_n) 无效到读写合一信号 (lbctl) 无效的时间。	-	1	-	Cycle
Tcswp	写操作时, 片选有效时间。	2	8	511	Cycle
Tcwen	如果写使能 (we_n) 先于片选 (cs_n) 无效, 表示写使能 (we_n) 无效到片选 (cs_n) 无效的时间; 如果片选 (cs_n) 先于写使能 (we_n) 无效, 表示从片选无效到写使能无效的时间。	0	1	31	Cycle
Tcdn	从片选无效到写数据无效的时间。	0	0	31	Cycle



符号	说明	最小值	典型值	最大值	单位
Tcl	在复用情况下，表示从片选无效到 LALE 有效的时间。	1	4	63	Cycle
Tlgt_a_samp	从片选有效到开始采集外部 Ready (LGTA) 的时间。如果从片选有效开始采，则此值设置为 1。	1	1	63	Cycle
Tlale	在复用情况下，地址锁存的指示信号的有效时间。	0	3	31	Cycle
Tcsn	在复用情况下从 Ready 有效到片选无效时间。	0	2	63	Cycle

须知

在配置时序之前配置 LB_CFG_T_UPDATE_EN bit[0]（物理地址为 0x08712154）为 0。

然后配置时序参数，在所有的时序参数配置好后，查询 DBG_T_CFG（debug 状态寄存器 2），如果等于 0，再将此寄存器 LB_CFG_T_UPDATE_EN bit[0]（物理地址为 0x08712154）配置为 1。

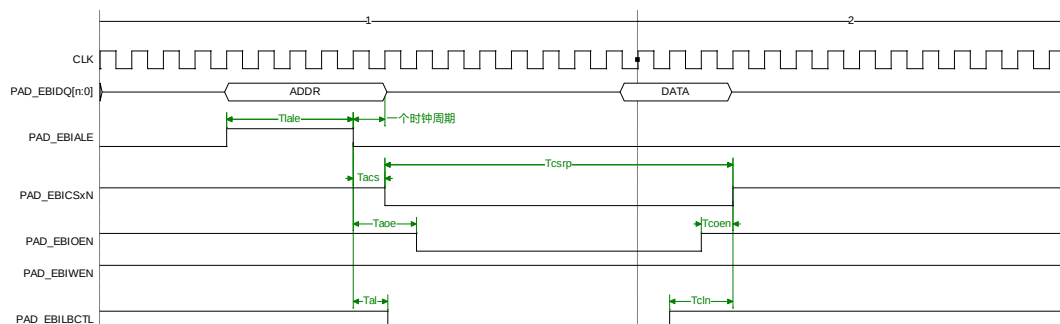
9.3.3.2 LocalBus 典型时序配置

本节根据 LocalBus 选择的固定周期或 Ready 模式，列出各种情况的时序图和配置参数。



复用、无外部 Ready 情况

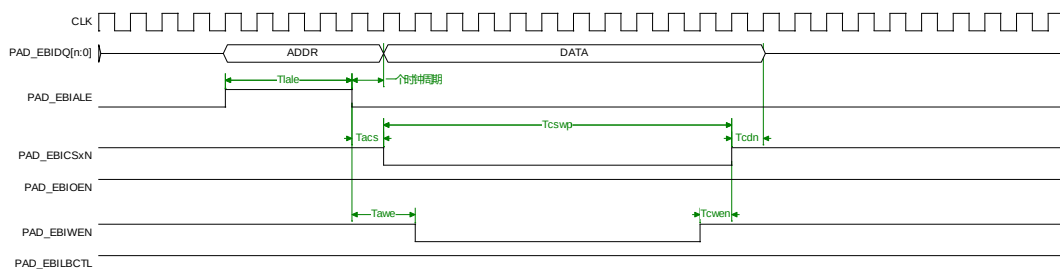
图9-3 LocalBus 复用无外部 Ready 单次读操作



须知

Localbus 读外部数据的操作，内部锁存读取的数据的时刻是片选 cs_n 或者读写合一信号 $lbctl$ 或者读使能 oe_n 的上升沿的前一个 cycle，以这三个信号最先无效的信号的上升沿为准。所以如果外部器件使用的是 oe_n 信号，要求 $Toen \geq Tln$ ；如果外部器件使用的是 $lbctl$ 信号，要求 $Tln \geq Toen$ 。

图9-4 LocalBus 复用无外部 Ready 单次写操作



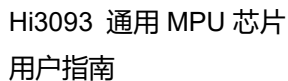
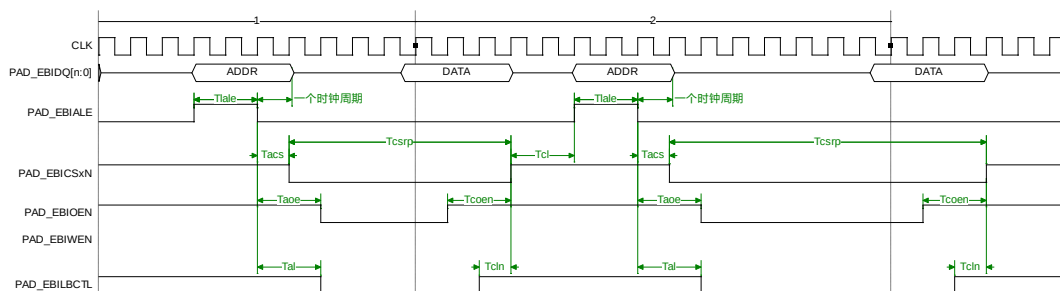


图9-5 LocalBus 复用无外部 Ready 连续读操作

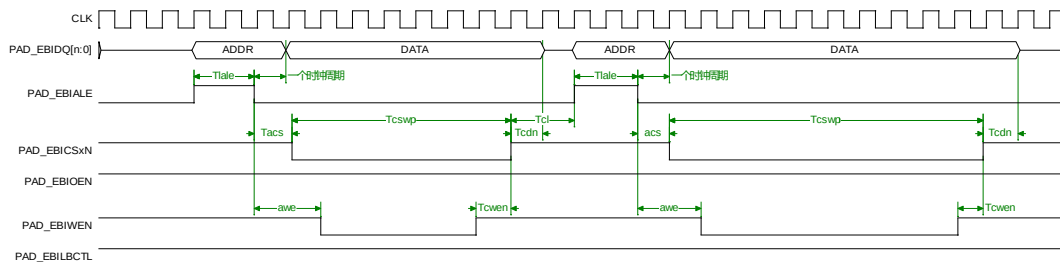


须知

Tcl 参数表示两次操作的间隔，Tcdn、Tcln、Twen、Tcoen 有效时间不能跨越两次操作，否则时序就不正确了。所以需要满足如下的要求：

- Tcd-3>=Tcdn; 保证数据保持时间不跨越两次操作。
- Tcd-3>=Tcln; 保证读写合一信号不跨越两次操作。
- Tcd-3>=Twen; 保证写使能信号不跨越两次操作。
- Tcd-3>=Toen; 保证读使能信号不跨越两次操作。

图9-6 LocalBus 复用无外部 Ready 连续写操作



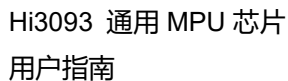
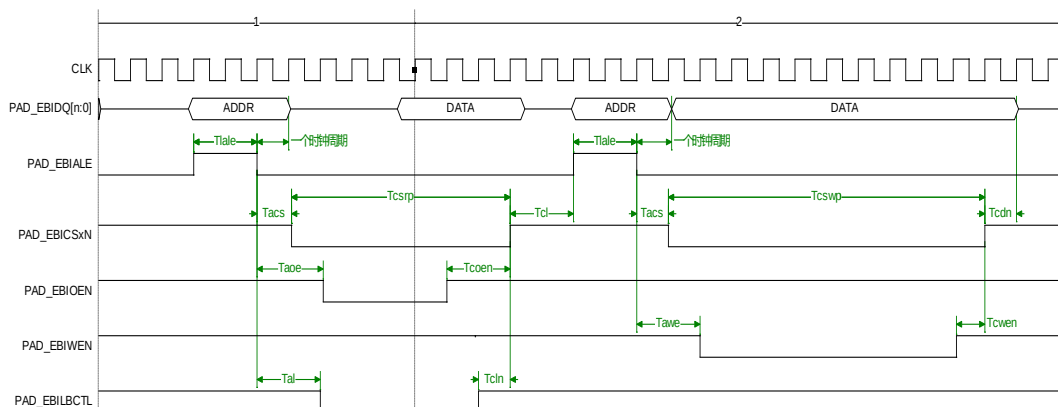
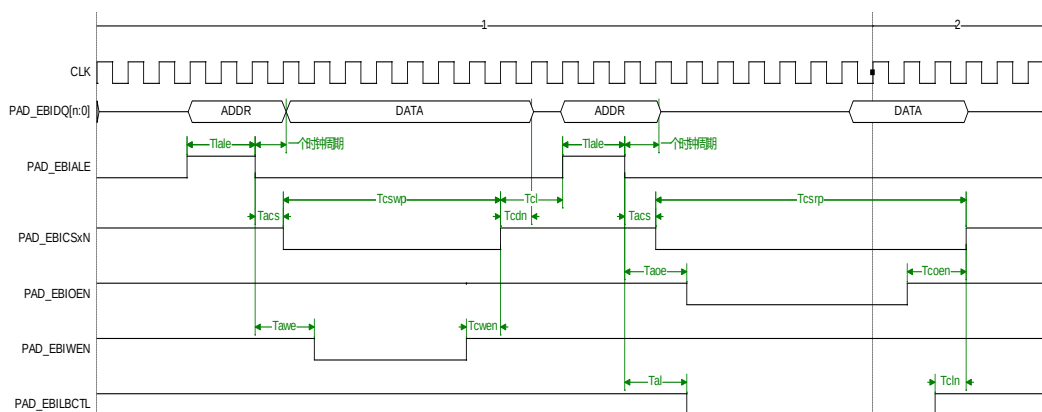


图9-8 LocalBus 复用无外部 Ready 写读切换操作



复用、有外部 Ready 情况

图9-9 LocalBus 复用有外部 Ready 单次读操作

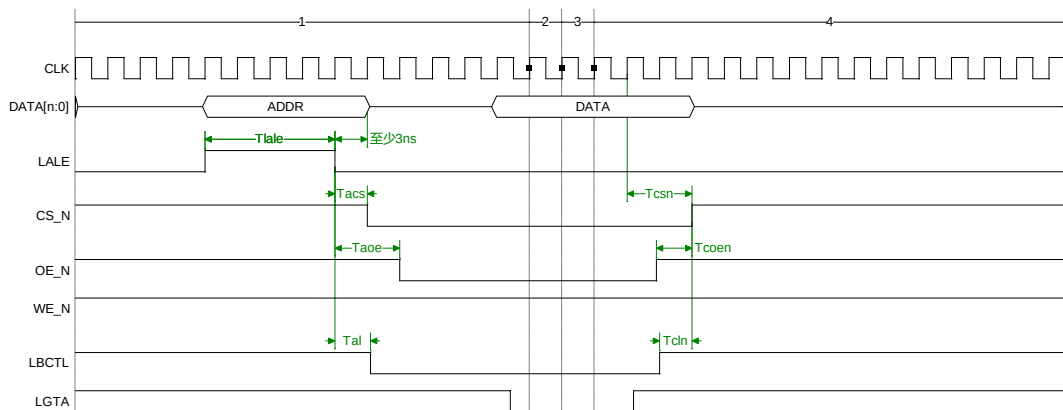




图9-10 LocalBus 复用有外部 Ready 单次写操作

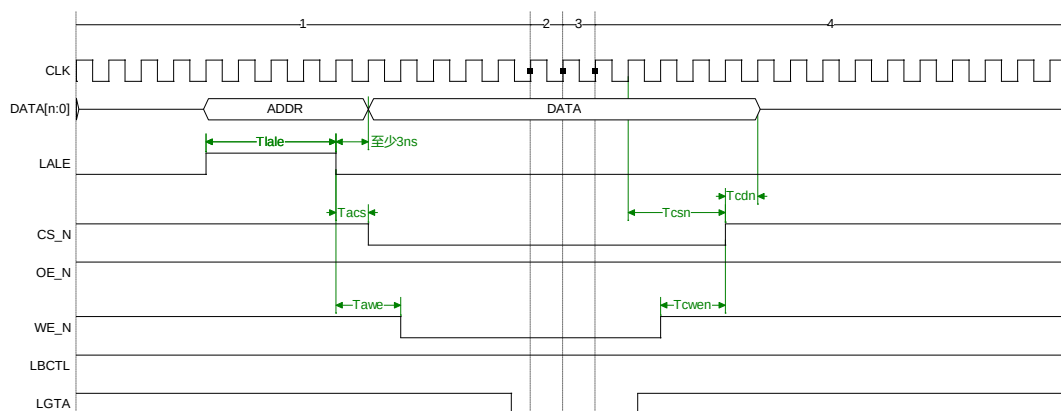


图9-11 LocalBus 复用有外部 Ready 连续读操作

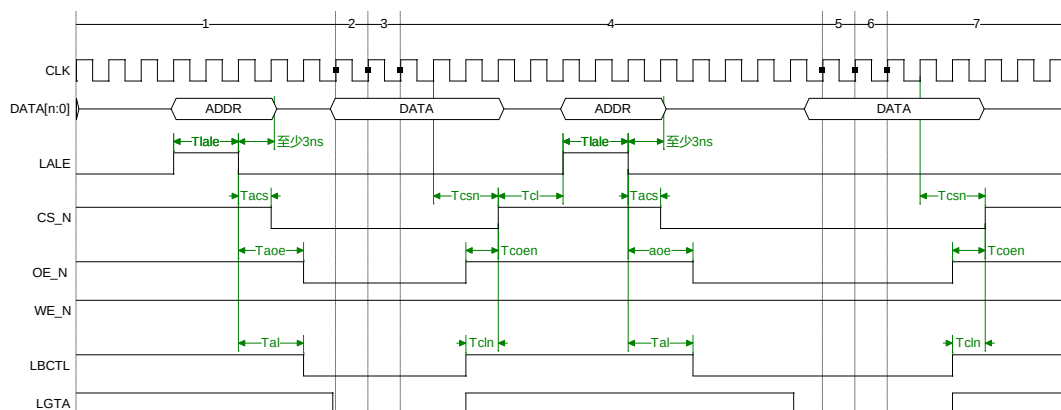


图9-12 LocalBus 复用有外部 Ready 连续写操作

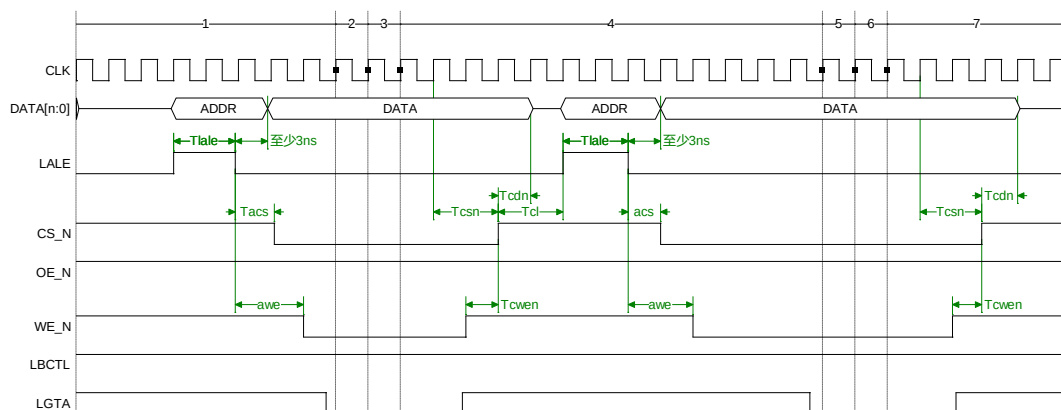




图9-13 LocalBus 复用有外部 Ready 读写切换操作

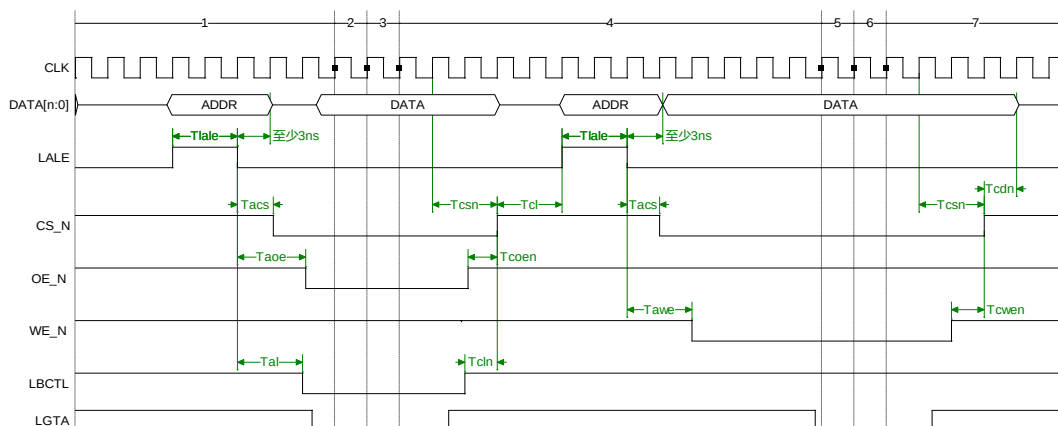


图9-14 LocalBus 复用有外部 Ready 写读切换操作

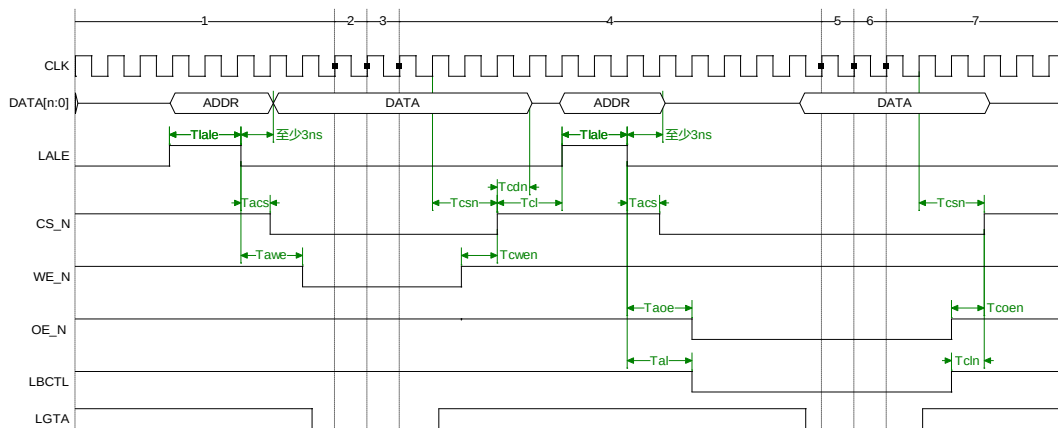
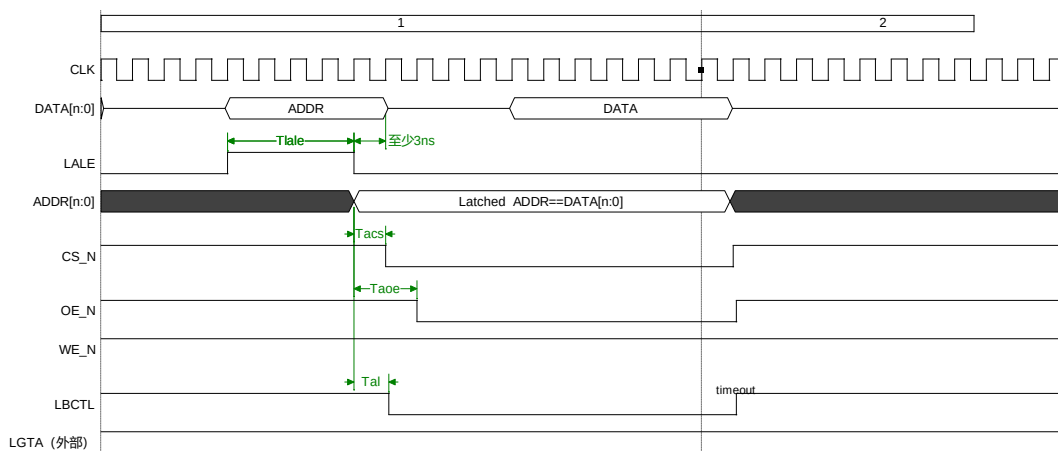


图9-15 LocalBus 复用有外部 Ready 外部 Ready 超时





9.4 工作方式

9.4.1 时钟门控

上电时 LOCALBUS 工作时钟默认为打开状态。在不使用 LOCALBUS 的情况下，可以关闭 LOCALBUS 工作时钟，以降低芯片功耗。

打开门控时钟具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS 控制寄存器 lb_clk_enb (LPSC_CLKEN_CTRL1 bit[5]，物理地址为 0x0871D014)为 1，打开 LOCALBUS 模块的时钟。

----结束

关闭门控时钟具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS 控制寄存器 lb_clk_dis (LPSC_CLKDIS_CTRL1 bit[5]，物理地址为 0x0871d1C0) 为 1，关闭 LOCALBUS 模块的时钟。

-----结束

同时可以通过读取 lb_clk_status (LPSC_CLKSTA_CTRL1 bit[5]，物理地址为 0x0871D1C4)来获取 Localbus 的时钟状态。

9.4.2 软复位

Localbus 软复位流程：

- 步骤 1 通过配置系统控制器 LPSC_SRST_REQ0 bit[4]（物理地址 0x0871D004）为 1，可实现对 Localbus 工作逻辑的单独软复位。

- 步骤 2 延时 2us。

- 步骤 3 配置系统控制器 LPSC_RESET_DREQ0 bit[4]（物理地址 0x0871D1FC）为 1，可实现对 Localbus 工作逻辑的单独软复位撤离。

----结束

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



9.4.3 初始化

需要根据 LOCALBUS 控制器外接存储器件的时序参数和特性，对 LOCALBUS 控制器进行正确配置，具体配置步骤如下（以配置片选 0 为例）：

- 步骤 1 根据外部器件配置片选 0 控制寄存器 LB_CS0_CTRL（物理地址为 0x08712020，bit[18:0]）。如配置是否支持外部 Ready 器件，是否支持 Page 读，Page 读大小，数据位宽等。
- 步骤 2 根据外部器件的时序特性，配置片选 0 的时序参数寄存器，LB_CS0_T_SET0（物理地址为 0x08712040），LB_CS0_T_SET1 物理地址为 0x08712044），LB_CS0_T_SET2 物理地址为 0x08712048）。
- 步骤 3 如果片选 0 访问的器件，存在写保护区域，则配置 LB_CTRL（偏移地址 0x00CC，bit[15:0]）使能写保护区域，例如使能保护地址段 0，配置写保护地址段 0 的起始地址 LB_PROT_ADDR_START_0（物理地址为 0x087120D0），配置写保护地址段 0 的结束地址 LB_PROT_ADDR_OVER_0（物理地址为 0x087120D4）。
- 步骤 4 如果在 1 中配置是带有外部 Ready 的器件，并且使能了超时保护功能，需要配置片选 0 的超时保护时间 LB_RDY_TOUT_0（物理地址为 0x087120AC）。

----结束

说明

如果通过片选 0（默认从片选 0 启动）从 norflash 启动，则需要设置管脚，选择适当的数据位宽，选择地址是否偏移。其他寄存器的配置值使用默认值。

9.4.4 基地址配置

4 个片选的基地址可以配置，具体步骤如下（以配置 CS0 为例）：

- 步骤 1 通过寄存器 LB_CSx_BASE_ADDR（物理地址为 0x08712000）配置 CSx 所挂载器件的基地址。
- 步骤 2 通过寄存器 LB_CSx_CTRL[cf_mem_size_x]（物理地址为 0x08712020）配置所挂载 memory 的存储容量。
- 步骤 3 读取调试寄存器 DBG_ST0 bit[0]（物理地址为 0x08712140），如果等于 1，说明有地址交叠，需要返回到[步骤 1](#)，或者[步骤 2](#)重新配置 memory 的地址分配。

----结束



须知

配置基地址要静态配置，最好在操作空闲期配置。

9.4.5 固定时序模式

需要根据 LocalBus 控制器外接存储器件的时序参数和特性，对 LocalBus 控制器进行正确配置，相关时序参数含义请参见 [9.3.3 灵活的时序配置](#) 章节。

具体配置步骤如下（以配置 CS0 为例）：

- 步骤 1 配置 LB_CFG_T_UPDATE_EN bit[0]（物理地址为 0x08712154）为 0。
- 步骤 2 根据外部器件的类型，配置 LB_CS0_CTRL（物理地址为 0x08712020）。
- 步骤 3 根据外部器件的时序要求，配置 LB_CS0_T_SET0、LB_CS0_T_SET1、LB_CS0_T_SET2（物理地址为 0x08712040、物理地址为 0x08712044、物理地址为 0x08712048）。
- 步骤 4 读取时序调试寄存器 DBG_T_CFG（物理地址为 0x08712148），如果等于 1，说明时序配置存在错误，需要返回到 [步骤 3](#)，重新配置时序参数，直到 DBG_T_CFG 的值为 0。
- 步骤 5 配置 LB_CFG_T_UPDATE_EN bit[0]（物理地址为 0x08712154）为 1。

----结束

须知

在上电启动时，采用的是 LocalBus 默认时序参数进行数据读取。这里用 CS0 举例，另外一个片选的操作方式相同。

9.4.6 异步等待模式

当 LocalBus 接带有外部 Ready 的器件时，需要进行如下配置（以片选 0 为例）：

- 步骤 1 配置 LB_CS0_CTRL[cf_rdy_mode_0]（物理地址为 0x08712020，bit[9]）为 1，使能外部 Ready。



- 步骤 2 配置 LB_CS0_CTRL[cf_rdy_pol_0] (物理地址为 0x08712020, bit[10]) , 配置外部 Ready 的有效极性, 为 0 表示低有效, 为 1 表示高有效。
- 步骤 3 配置 LB_CS0_CTRL[cf_rdy_tout_en_0] (物理地址为 0x08712020, bit[16]) , 配置为 0 关闭超时保护功能, 配置为 1 使能超时保护功能。
- 步骤 4 如果使能超时保护功能, 需要配置超时保护时间, LB_RDY_TOUT_0 (偏移地址为 00AC) 。
- 步骤 5 配置 LB_CS0_T_SET2[cf_t_lgta_samp_0] (物理地址为 0x08712048, bit[28:23]), 从片选有效到开始采集外部 Ready 的时间, 如果此参数小于等于 1, 则从片选有效开始采集。
- 步骤 6 配置 LB_CS0_T_SET2[cf_t_csn_0] (物理地址为 0x08712048, bit[5:0]), 外部 Ready 有效 3 拍到片选无效的时间 (异步模式特有时序参数) 。

----结束

9.4.7 Page 读模式

Page 读模式需要如下配置 (以片选 0 为例):

- 步骤 1 配置 LB_CS0_CTRL[cf_page_en_0] (物理地址为 0x08712020, bit[13]) , 配置为 1, 使能 Page 读模式。
- 步骤 2 配置 LB_CS0_CTRL[cf_Page_size_0] (物理地址为 0x08712020, bit[15:14]) , 配置 Page size 的大小。
- 步骤 3 配置 LB_CS0_T_SET0[cf_t_csrp_0] (物理地址为 0x08712040, bit[26:18]), 配置从地址有效到第一次读数据的时间。
- 步骤 4 配置 LB_CS0_T_SET2[cf_t_prc_0] (物理地址为 0x08712048, bit[10:6]), 配置两次读之间的时间。

----结束

9.4.8 写保护功能

9.4.8.1 普通写保护功能

LocalBus 的写保护功能需要如下配置 (以写保护地址段 0 为例):



- 步骤 1 配置 LB_CTRL[cf_wr_prot_0] (物理地址为 0x087120CC, bit[15:14]), 配置为 01 表示低级写保护使能。
- 步骤 2 配置写保护地址段 0 的起始地址 LB_PROT_ADDR_START_0 (物理地址为 0x087120D0), 配置写保护地址段 0 的结束地址 LB_PROT_ADDR_OVER_0 (物理地址为 0x087120D4)。
- 步骤 3 如果在保护地址段内存在命令字地址, 且需要使命令字写操作可以执行。则需要配置命令字地址 LB_NOR_CMDi_ADDR(i=0-3) (基地址为 0x08712000, 偏移地址为 0x0110、0x118、0x120、0x128) 和相应的掩码 LB_NOR_CMDi_MASK(i=0-3) (基地址为 0x08712000, 偏移地址为 0x0114、0x11C、0x124、0x12C), 命令字地址和相应掩码共有四对, 可根据写保护地址段 0 的命令字个数选择。

----结束

9.4.8.2 高级写保护功能

- 步骤 1 配置 LB_CTRL[cf_wr_prot_0] (物理地址为 0x087120CC, bit[15:14]), 配置为 10 表示高级写保护使能。
- 步骤 2 配置高级写保护使能关键字 LB_HL_PROT_KEY0[cf_hl_prot_key0] (物理地址为 0x08712130, bit[31:16])。
- 步骤 3 配置写保护地址段 0 的起始地址 LB_PROT_ADDR_START_0 (物理地址为 0x087120D0), 配置写保护地址段 0 的结束地址 LB_PROT_ADDR_OVER_0 (物理地址为 0x087120D4)。
- 步骤 4 如果在保护地址段内存在命令字地址, 且需要使命令字写操作可以执行。则需要配置命令字地址 LB_NOR_CMDi_ADDR(i=0-3) (基地址为 0x08712000, 偏移地址为 0x0110、0x118、0x120、0x128) 和相应的掩码 LB_NOR_CMDi_MASK(i=0-3) (基地址为 0x08712000, 偏移地址为 0x0114、0x11C、0x124、0x12C), 命令字地址和相应掩码共有四对, 可根据写保护地址段 0 的命令字个数选择。

----结束

9.4.9 数据读写方式

- 总线数据位宽 32bit, Localbus 接口数据位宽 8bit 时:
 - 写操作, 先发 bus_data[7:0], 其次发 bus_data[15:8]; 然后发 bus_data[23:16]; 最后发 bus_data[31:24]。



- 读操作，先读 lb_data[7:0]，其次读 lb_data[15:8]；然后读 lb_data[23:16]；最后读 lb_data[31:24]。在 Localbus 内部拼接成 lb[31:0]给总线。
- 总线数据位宽 32bit，Localbus 接口数据位宽 16bit 时：
 - 写操作，先发 bus_data[15:0]，再发 bus_data[31:16]。
 - 读操作，先读 lb_data[15:0]，再读 lb_data[31:16]。在 Localbus 内部拼接成 lb[31:0]给总线。

9.4.10 外部 Ready 模式

当 LocalBus 控制器外接带有 Ready 信号输出的存储芯片或者控制芯片时，可以将 LocalBus 控制器切换到 Ready 模式下，通过与外接芯片握手协商来进行数据传输。相关时序参数含义请参见 [9.3.3 灵活的时序配置](#) 章节。

具体配置步骤如下（以配置 CS0 为例）：

- 步骤 1 根据外部器件的类型，配置 LB_CS0_CTRL (物理地址为 0x08712020)，选定所需类型和开启 Ready 模式，并且可以配置 Ready 的有效极性，是否支持超时保护等功能。
- 步骤 2 如果[步骤 1](#)开启了超时保护，可以配置 LB_RDY_TOUT_0 (物理地址为 0x087120AC)，超时的时间。

----结束

完成以上步骤后，Localbus 控制器即可工作在 Ready 模式，并对等待信号的有效电平进行检测。



10 SFC

10.1 概述

SFC 逻辑是一个 SPI Flash 控制器，主要功能完成系统对 SPI Flash 的访问控制。

Hi3093 支持两个 SFC 控制器，分别为 SFC0 和 SFC1。

10.2 特性

SFC 的主要特性如下：

- 控制器支持 2 路片选。
- 支持 1, 2, 4 线模式。
- 支持接口时钟频率可配置，最高 50MHz。
- 支持两个片选 CS，单个片选存储空间最大可达 128Mbit（3B 地址模式）/8Gbit（4B 地址模式）。支持 Alias 方式从 SPI Flash 启动。
- 支持 StandardSPI、Dual-Output/Dual-InputSPI、Quad-Output/Quad-InputSPI、Dual-I/OSPI、Quad-I/OSPI、FullDIOSPI、FullQIOSPI 七种接口模式。上电后默认支持 StandardSPI 接口模式，可通过寄存器配置选择支持其他几种接口模式。两个片选所接 Flash 的接口模式必须相同。
- 支持 3Byte 和 4Byte 两种 Flash 地址模式。SFC1 上电后默认支持 3Byte 地址模式，可通过寄存器配置切换地址模式。
- 支持 WriteProtect 操作。
- 读写操作支持总线直接读写、寄存器编程读写和 DMA 方式读写 3 种方式。
- 支持 DMA，支持设置为安全或非安全区模式：



- 在安全区模式下支持从安全区配置 DMA 从安全或非安全 DDR 搬移数据写入 Flash;
- 在非安全区模式下从非安全区配置 DMA 从非安全 DDR 搬移数据写入 Flash (具体实现在安全方案中)。

说明

Hi3093 的 SFC 不支持 RESET/HOLD 复用的 Flash 器件。

10.3 功能描述

10.3.1 Boot 模式

当使用 SFC 控制器 Boot 时, 外接的存储器为 SPIFlash, 此时需要设置芯片外部管脚用于选择 Boot 的存储器, 两个控制器共用一个上电地址模式控制管脚:

- 通过设置管脚, 选择 Boot 时向 SPIFlash 存储器件发送的地址模式:
 - Default_flash_addr_mode (对应管脚 LB_ADDR6) =0, 选择 3B 模式。
 - Default_flash_addr_mode (对应管脚 LB_ADDR6) =1, 选择 4B 模式。

10.3.2 读写 Memory

10.3.2.1 读 Memory

提供两种读 Flash 的途径: 第一种通过总线读, 第二种通过设置 SFC 的配置寄存器来读 Flash。第一种读 Flash 操作效率高, SFC 模块自动将内部总线的读操作时序映射为 SPI 时序。第二种读 Flash 操作效率低, 需要软件配置寄存器。

对于内部总线的任何读 Memory 操作, SFC 模块都将其转换成 SPI 接口的读操作。上电后默认执行 Normalread 操作 (Opcode=0x3), 这种读操作对应的 DummyByte 个数为 0。上电系统 Boot 起来后, 可以通过软件配置读命令的 Opcode 和 DummyByte 个数来进行其他的读操作, 如 FastRead (Opcode=0xB), DummyByte 为 1。

10.3.2.2 写 Memory

提供两种写 Flash 的途径: 第一种通过内部总线写, 第二种通过设置 SFC 的配置寄存器来写 Flash。第一种写 Flash 操作效率高, SFC 模块自动将总线的一次写操作转换为 Flash 侧的 WREN 操作 (WriteEnable) + PP 操作 (PageProgram) + RDSR (ReadStatusRegister) 操作。第二种写 Flash 操作效率低, 需要软件配置寄存器。



在写 Memory 之前要对 Flash 进行 Erase 操作，这项操作必须要由软件通过配置 SFC 的寄存器来完成。

Flash 执行 PP 操作需要一定的时间，需要通过读 Flash 的状态寄存器来判断 PP 操作是否完成才能开始下一次写操作，否则会丢失数据。

PP 操作不能跨 Pageboundary，对于跨 Pageboundary 的写操作必须拆成 2 次来完成。CPU 通过总线写 Flash 时，SFC 模块根据地址和 Burst 长度自动判断此次操作是否会跨 Pageboundary，如果跨 Pageboundary 就拆成两次来处理。

其他操作

对 Flash 的其他操作如 Erase、进入 DeepPowerDown、读 DeviceID 等必须通过寄存器访问来实现。

10.4 工作方式

10.4.1 管脚复用配置

SFC0 复位后默认为 SFC0 的功能。

如果管脚复用寄存器有改写，SFC0 使用前需要配置管脚复用。配置 IOCFG 的 IOMG005 为 0x0。

SFC1 复位后默认为 SFC1 的功能。

如果管脚复用寄存器有改写，SFC1 使用前需要配置管脚复用。配置 IOCFG 的 IOMG006 为 0x0，IOMG007 为 0x0，IOMG008 为 0x0，IOMG009 为 0x0。

10.4.2 初始化

以 SFC1 为例，SFC0 的流程完全相同，只是寄存器的基地址和空间分配不同，请参考《Hi3093 寄存器手册》。

步骤 1（如果需要调整 Timing 参数）配置 Timing 寄存器（物理地址 0x08600110）。

步骤 2 配置 3Byte 与 4Byte 两种 Flash 地址模式初始化时，配置 GLOBAL_CONFIG bit[2]（物理地址 0x08600100）寄存器来选择相应的地址模式。



须知

SFC 上电时，会采样 LB_ADDR6 管脚来配置 SFC 工作在 3Byte or 4Byte 地址模式，所以即便使用 3Byte 地址模式，也应配置该寄存器。

- 步骤 3 配置总线操作方式寄存器 BUS_CONFIG1（物理地址 0x08600200）。
- 步骤 4 根据实际 Flash 大小配置 BUS_FLASH_SIZE（物理地址 0x08600210 可通过发 Read ID 命令给 Flash 查询获得器件型号获知）。
- 步骤 5 依据 Flash 直接映射到系统地址空间情况配置 BUS_BASE_ADDR_CS0（物理地址 0x08600214），BUS_BASE_ADDR_CS1（物理地址 0x08600218），BUS_ALIAS_ADDR_CS（物理地址 0x08600220）。映射空间应在系统总线分配给 SFC_MEM 的地址空间范围内。
- 步骤 6 通常 BUS_ALIAS_CS 对应从 Flash 启动时映射的地址，只默认值有意义，所以一般固定不修改。
- 步骤 7 有些器件要求进入非 Standard SPI 读写时序需要预先以特殊命令配置 Flash。根据器件需要，用寄存器方式发特定命令配置 Flash。
- 步骤 8 配置 BUS_CONFIG1/BUS_CONFIG2（物理地址 0x08600200 和 0x08600204），如果需要开启总线写操作，使能总线写，需配置 BUS_CONFIG1 bit[30]（物理地址 0x08600200）为 1。默认总线写功能是关闭着的。

----结束

须知

寄存器无需初始化，每次操作都需要重新配置。注意以上初始化流程仅做参考，请视器件差异进行调整。

10.4.3 时钟门控

上电时 SFC 工作时钟默认为打开状态。在不使用 SFC 的情况下，可以关闭 SFC 工作时钟，以降低芯片功耗。打开门控时钟具体配置步骤如下。

SFC0 门控流程：

- 步骤 1 配置 LP_SUBSYS 控制寄存器 sfc0_clk_enb（LPSC_CLKEN_CTRL0 bit[5]，物理地址为 0x0871D010）为 1，打开 SFC0 模块的时钟。



----结束

关闭门控时钟具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS 控制寄存器 sfc0_clk_dis (LPSC_CLKDIS_CTRL0 bit[5], 物理地址为 0x0871D1B8) 为 1, 关闭 SFC0 模块的时钟。

-----结束

同时可以通过读取 sfc0_clk_status (LPSC_CLKSTA_CTRL0 bit[5], 物理地址为 0x0871D1BC) 来获取 SFC0 的时钟状态。

SFC1 门控流程：

- 步骤 1 配置 LP_SUBSYS 控制寄存器 sfc1_clk_enb (LPSC_CLKEN_CTRL0 bit[6], 物理地址为 0x0871D010) 为 1, 打开 SFC1 模块的时钟。

----结束

关闭门控时钟具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS 控制寄存器 sfc1_clk_dis (LPSC_CLKDIS_CTRL0 bit[6], 物理地址为 0x0871D1B8) 为 1, 关闭 SFC1 模块的时钟。

-----结束

同时可以通过读取 sfc1_clk_status (LPSC_CLKSTA_CTRL0 bit[6], 物理地址为 0x0871D1BC) 来获取 SFC1 的时钟状态。

除了时钟门控之外, 还可以通过 LP_SUBSYS 寄存器 sfc0_clk_div_cnt (LPSC_CRG_DIV_CTRL bit[14:10], 物理地址为 0x0871D018) 和 sfc1_clk_div_cnt (LPSC_CRG_DIV_CTRL bit[19:15], 物理地址为 0x0871D018) 来控制 SFC0 和 SFC1 的接口时钟频率, 控制器和 NorFlash 的接口频率实际为这个分频系数基础上的二分频。在配置时钟频率时, 确认模块时钟处于关闭状态。

10.4.4 软复位

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

SFC0 软复位流程：



步骤 1 通过配置系统控制器 LPSC_SRST_REQ0 bit[5] (物理地址 0x0871D004) 为 1, 可实现对 SFC1 工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置系统控制器 LPSC_RESET_DREQ0 bit[5] (物理地址 0x0871D1FC) 为 1, 可实现对 SFC1 工作逻辑的单独软复位撤离。

----结束

SFC1 软复位流程:

步骤 1 通过配置系统控制器 LPSC_SRST_REQ0 bit[6] (物理地址 0x08740004) 为 1, 可实现对 SFC1 工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置系统控制器 LPSC_RESET_DREQ0 bit[6] (物理地址 0x0871D1FC) 为 1, 可实现对 SFC1 工作逻辑的单独软复位撤离。

----结束

10.4.5 通过寄存器读 Flash (查询方式)

通过寄存器读 Flash 的操作步骤如下:

步骤 1 配置 CMD_INS (物理地址 0x08600308) 寄存器。

步骤 2 配置 CMD_ADDR (物理地址 0x0860030C) 寄存器。

步骤 3 配置 CMD_CONFIG (物理地址 0x08600300) 寄存器的 data_cnt、ummy_byte_cnt、addr_en、rddata_en、sel_cs、start 位。

步骤 4 读 CMD_CONFIG (物理地址 0x08600300) 寄存器, 等待 start 为 0。

步骤 5 根据 data_cnt 读取寄存器 DATABUFFER1~8 (物理地址 0x08600400~0x0860041C)。

----结束



10.4.6 通过寄存器写 Flash（中断方式）

通过寄存器写 Flash 的操作步骤如下：

- 步骤 1 配置 CMD_INS（物理地址 0x08600308），发 WREN 操作。
- 步骤 2 配置 CMD_CONFIG（物理地址 0x08600300）寄存器的 addr_en、sel_cs 和 start 位。
- 步骤 3 等待中断发生。
- 步骤 4 写 INT_CLEAR（物理地址 0x0860012C）寄存器清除中断。
- 步骤 5 配置 CMD_INS（物理地址 0x08600308）寄存器，发 PP 操作。
- 步骤 6 根据写数据长度配置寄存器 DATABUFFER1~BUFFER8（物理地址 0x08600400~0x0860041C）。
- 步骤 7 配置 CMD_ADDR（物理地址 0x0860030C）寄存器（注意地址不能跨 PageBoundary）。
- 步骤 8 配置 CMD_CONFIG（物理地址 0x08600300）寄存器的 data_cnt、dummy_byte_cnt、addr_en、wrdata_en、sel_cs、start 位。
- 步骤 9 等待中断发生。
- 步骤 10 写 INT_CLEAR（物理地址 0x0860012C）寄存器清除中断。
- 步骤 11 配置 CMD_INS（物理地址 0x08600308）寄存器，发 RDSR 操作。
- 步骤 12 配置 CMD_CONFIG（物理地址 0x08600300）寄存器的 data_cnt、dummy_byte_cnt、addr_en、wrdata_en、sel_cs、start 位。
- 步骤 13 等待中断发生。
- 步骤 14 写 INT_CLEAR（物理地址 0x0860012C）寄存器清除中断。
- 步骤 15 确认 DATABUFFER1 bit[0]（物理地址 0x08600400）为 0。

----结束

10.4.7 通过 DMA 方式读写 Flash 操作流程

初始化流程：配置 SFC BUS_XX 参数。



DMA 操作与 BUS_XX 操作共用寄存器，需保证 DMA 操作中不更改 BUS_XX 寄存器。DMA 操作流程。

- 步骤 1 配置 DMA 操作的内存端起始地址 BUS_DMA_MEM_SADDR（物理地址 0x08600244）、Flash 端起始地址 BUS_DMA_FLASH_SADDR（Flash 偏移地址）、数据长度 BUS_DMA_LEN（物理地址 0x0860024C），这三个参数需要 4Byte 对齐，长度为 4Byte 的整数倍。
- 步骤 2 写 BUS_DMA_CTRL（物理地址 0x08600240）配置读写方向，操作 Flash 片选（Flash0 或 Flash1）和 DMA 使能。
- 步骤 3 等待 dma_done 中断触发（中断方式）或轮询 DMA 操作完成（DMA_CTRL bit[0]变为 0）。
- 步骤 4 DMA 操作时可以同时通过 Flash 寄存器命令操作。
- 步骤 5 DMA 操作时可以同时通过 AHB Slave 直接访问 Flash，但需保证中间不修改总线相关配置。

----结束

由于芯片总线地址位宽为 34 位，所以在配置 DMA 地址时如需用到高两位则在 LP_SUBSYS 控制器的 LPSC_ADDR_SEL bit[7:6]（物理地址为 0x0871D000）进行配置。

此外，DMA 搬运时请注意源地址，目的地址以及搬运长度需要 4Byte 对齐。

10.4.8 通过寄存器方式擦除 FLASH

BLOCK 擦除，以片选 0 为例，操作流程如下：

- 步骤 1 配置 CMD_INS（物理地址 0x08600308）为 0x6。
- 步骤 2 配置 CMD_CONFIG（物理地址 0x08600300）为 0x1，打开写使能。
- 步骤 3 配置 CMD_INS（物理地址 0x08600308）为 0xC7。（不同 Flash 器件擦除指令可能有所不同）
- 步骤 4 配置 CMD_CONFIG（物理地址 0x08600300）为 0x1，启动擦除。
- 步骤 5 读 Flash 内部状态寄存器判断擦除是否完成。

----结束



说明

上面以 SFC0 为例说明工作流程，SFC1 同理。



11 SMBUS

11.1 概述

SMBUS 控制器在 I2C 控制器新增了 SMBUS 协议相关特性。SMBUS 和 I2C 共用一套管脚和寄存器空间，通过配置寄存器来进行切换。

SMBUS 控制器和 I2C 控制器的个数相同，芯片提供 16 个 SMBUS 和 1 个安全 SMBUS。

11.2 特性

SMBUS 控制器具有以下特性：

- 支持标准 I2C、SMBUS 2.0 总线协议。
- 支持 SMBUS Master 设备操作。
- 支持 I2C Mater 和 Slave 设备操作。
- 支持 7bit 地址。
- 支持速率可配，支持标准速度（SS，0Kbps~100Kbps），快速（FS，100Kbps~400Kbps）。
- 支持 SMBUS2.0 协议命令。
- 支持 SMBUS Master ARP 配置流程。
- 支持 SCL 低电平超时检测门限配置，并上报超时中断。
- 支持 SMBUS 协议的 SEXT 和 Tlow：MEXT 门限配置，并上报超时中断。
- 支持以中断方式和查询方式进行数据搬运。
- 支持 TX/RX FIFO 深度为 128。



- SMBUS 控制器支持切换为 I2C 控制器。寄存器基地址、管脚与 I2C 保持一致。

11.2.1 关键术语

如表 11-1 术语将在本章节表示特殊的含义。

表11-1 I2C 模块关键术语描述

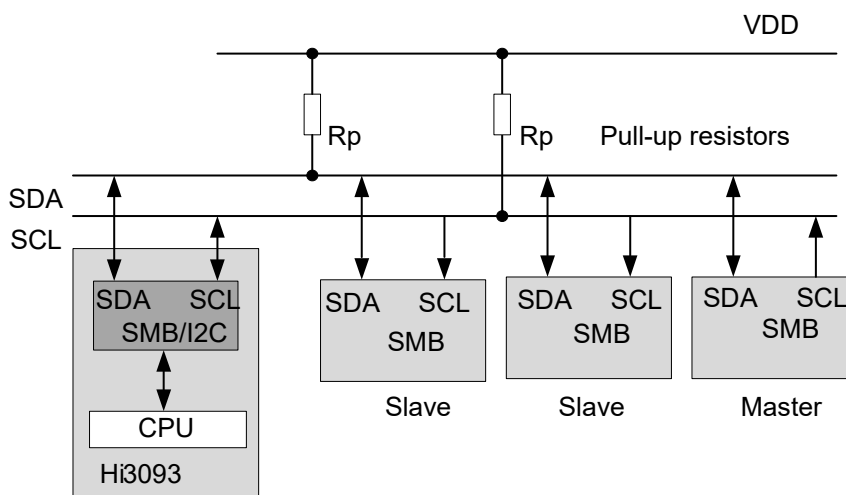
术语	描述
I2C	Inter-Integrated Circuit, I2C 总线是由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。是微电子通信控制领域广泛采用的一种总线标准。
SMBUS	The System Management Bus, SMBUS 是一种基于 I2C 的双线接口协议，通过 SMBUS 各系统芯片可以相互通信，并与系统的其余器件进行通讯。
ARP	The Address Resolution Protocol (ARP), 地址解析协议。SMBUS 可通过 ARP 协议分配 slave 地址。



11.3 功能描述

11.3.1 典型应用

图11-1 SMBUS 典型应用



SMBUS 模块支持 Master 模式。

支持中断或查询方式收发数据：发送数据时，在完成 I2C 初始化后，只需要将发送数据写入 FIFO 中，再配置寄存器 IC_DATA_CMD.cmd 启动发送，通过查询状态寄存器或检查中断判断数据发送是否完成；接收数据时，配置寄存器 I2C_DATA_CMD.cmd 为 1 后，通过查询状态寄存器或检查中断判断数据接收是否完成，如若完成，通过寄存器直接读取接收 FIFO 中的数据即可。

11.4 工作方式

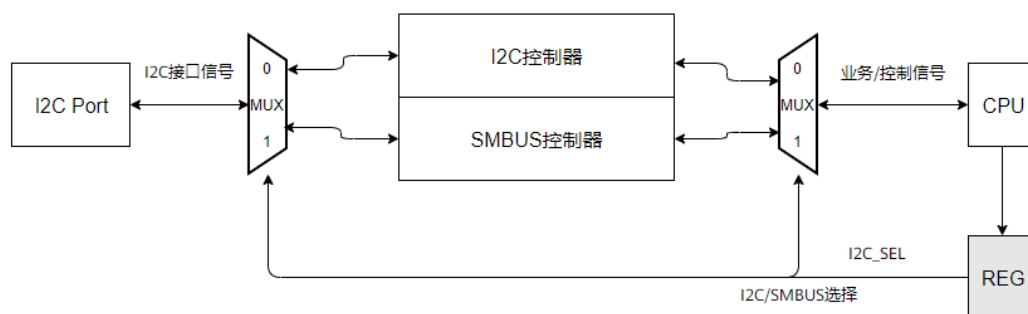
11.4.1 管脚复用配置

参考《Hi3093 硬件设计指南》管脚复用相关章节。

11.4.2 I2C 和 SMBUS 模式选择

I2C 和 SMBUS 控制器完全共用寄存器地址和 I2C 管脚，通过 LP_SUBSYS 的 I2Cx_SEL 寄存器控制二者的切换，启动默认为 I2C 控制器，如图 17-5 所示。

图11-2 I2C 和 SMBUS 控制器复用关系示意图



须知

I2C 和 SMBUS 不能同时工作。用户需上电后就决定控制器为 I2C 还是 SMBUS。不支持工作中进行 I2C 和 SMBUS 的模式动态切换。

11.4.3 时钟门控

请参见 17.5.5.1 步骤 1 和 17.5.3 时钟门控同 I2C。

11.4.4 软复位

请参见 17.5.5 软复位同 I2C。

11.4.5 TimeOut 超时处理

SMBUS 协议规定了 TimeOut 超时处理的机制，细节请参考《System Management Bus (SMBus) Specification Version 2.0》。协议规定的参数范围如表 11-2 所示。



表11-2 SMBUS TimeOut 参数

超时参数	描述	SMBUS /单位 ms		
		最小	最大	单位
Ttimeout	Detect clock low timeout	25	35	ms
TLOW_SEXT	cumulative clock low extend time (slave device)	-	25	ms
TLOW_MEXT	cumulative clock low extend time (master device)	-	10	ms

11.4.5.1 SMBUS 超时参数含义

SMBUS 控制器支持配置 SCL_STUCK_AT_LOW_TIMEOUT、SDA_STUCK_AT_LOW_TIMEOUT、TLOW_SEXT、TLOW_MEXT 超时门限的配置，并根据该门限来产生各种中断。

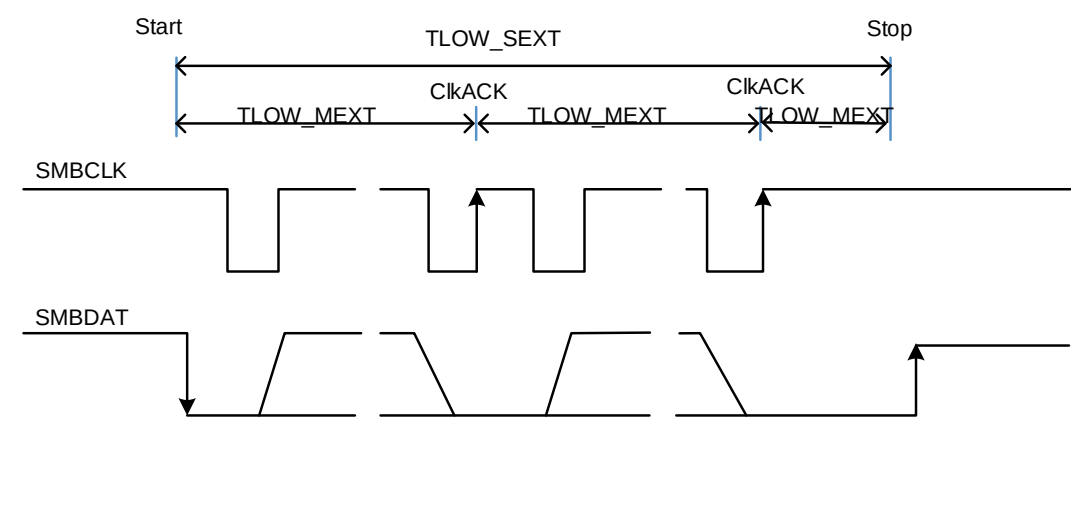
SCL_STUCK_AT_LOW_TIMEOUT 表示传输过程中，SCL 被持续拉低的超时门限。
SDA_STUCK_AT_LOW_TIMEOUT 表示传输过程中，检测 SCL 被持续拉低的超时门限。

TLOW_SEXT 表示一次传输过程中，START 到 STOP 中，检测到 SCL 被 Slave 扩展拉低的时间。

TLOW_MEXT 表示在一次传输过程中，Start to ACK、ACK to ACK、ACK to Stop，检测到 SCL 被 Master 扩展拉低的时间。



图11-3 SMBUS TLOW_MEXT 和 TLOW_SEXT



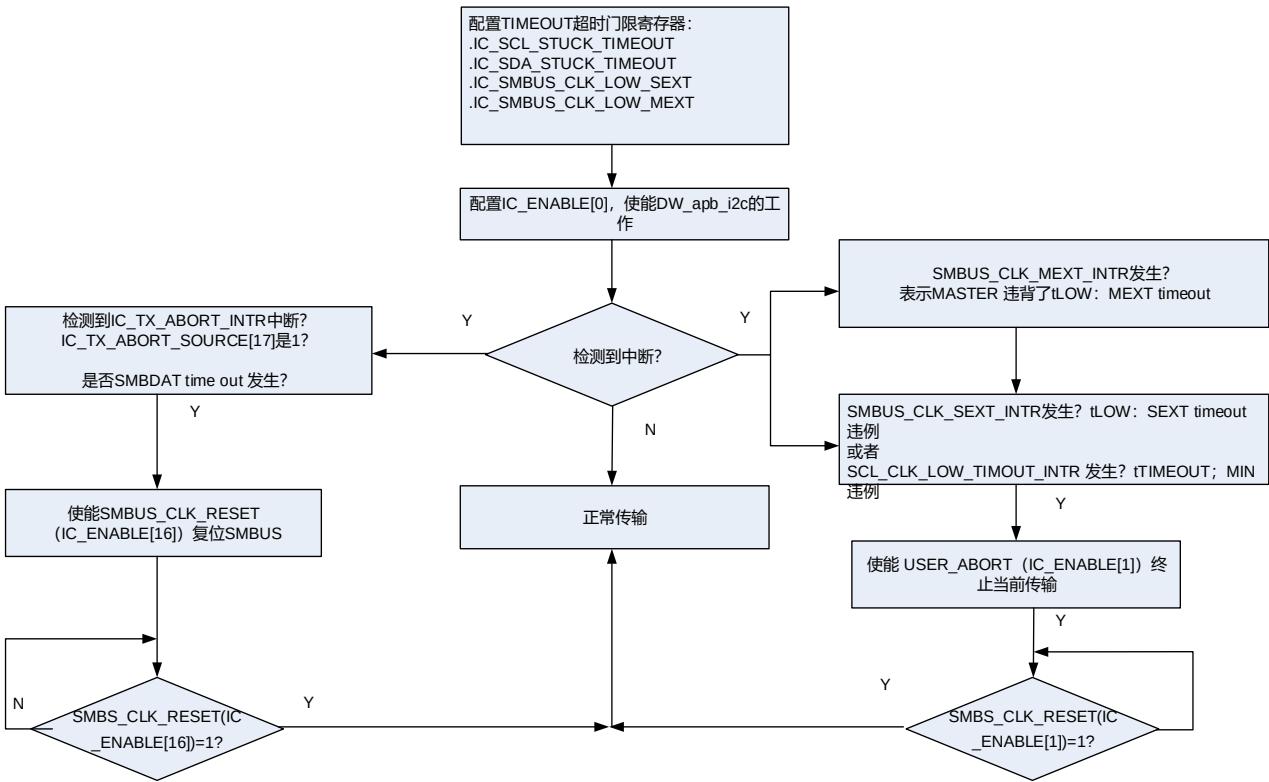
须知

上述超时参数和配置，同样适用于 SMBUS 控制器进行普通的 I2C 传输。

11.4.5.2 SMBUS 超时处理流程

SMBUS 作为 Master 的超时处理流程如图 11-4 所示。

图11-4 SMBUS Master 超时处理流程





11.4.6 主动中断 I2C 传输

- 步骤 1 停止操作寄存器 IC_DATA_CMD。
- 步骤 2 写寄存器 IC_ENABLE.abort 为 1。
- 步骤 3 等待 IC_RAW_INTR.r_tx_abrt 中断。
- 步骤 4 读取 IC_TX_ABORT_SOURCE 来识别 abort 来源 (bit[16]为高，表示用户配置的 tx abort) 。
- 步骤 5 读取 IC_CLR_TX_ABRT 清空中断。

----结束

11.4.7 时钟配置

I2C 作为主设备时需要通过配置寄存器来设置标准模式下 I2C 总线 SCL 信号的高低电平宽度，即相对于 I2C 工作时钟的周期数，寄存器 IC_SS_SCL_HCNT、IC_SS_SCL_LCNT 典型配置值如表 11-3 所示。

表11-3 IC_SS_SCL_HCNT/LCNT 典型配置值

速度模式	I2C 工作时钟 (MHz)	SCL 高电平宽度 (μ s)	SCL 高电平计数 值 (*_HCNT)	SCL 低电 平宽度 (μ s)	SCL 低电平计数 值 (*_LCNT)
SS (100Kbps)	100	5.2	0x190	4.7	0x1d6

I2C 作为主设备时需要通过配置寄存器来设置快速模式下 I2C 总线 SCL 信号的高低电平宽度，即相对于 I2C 工作时钟的周期数，寄存器 IC_FS_SCL_HCNT 和 IC_FS_SCL_LCNT 典型配置值如表 11-4 所示。



表11-4 IC_FS_SCL_HCNT 典型配置值

速度模式	I2C 工作时钟 (MHz)	SCL 高电平宽度 (μ s)	SCL 高电平计数 值 (*_HCNT)	SCL 低电 平宽度 (μ s)	SCL 低电平计数 值 (*_LCNT)
FS (400Kbps)	100	0.9	0x3c	1.6	0x82

11.4.8 中断或查询方式下的数据传输

本章节描述了基本的 I2C 传输流程。SMBUS 的协议命令传输也是基于该命令和流程。

11.4.8.1 初始化

步骤 1 配置 IC_SMBUS 选择开关, LP_SUBSYS 寄存器:

LPSC_SMBUS_I2C_MUX/smbus_i2c_selX (X=0~15) 配置为 1。

步骤 2 检测 IC_STATUS.activity, 若为 1, 则延时等待, 继续检测, 直到

IC_STATUS.activity 变为 0, 配置 IC_ENABLE.enable 为 0, 使 I2Cx 处于关闭状态。

步骤 3 配置 IC_CON.speed, 设定传输速率模式。

步骤 4 配置 IC_CON.restart_en 为 1, 使能发送 Re-start 指令。

步骤 5 若 I2C 为主设备, 配置 IC_CON.master 为 1。配置 IC_CON.slave_disable 为 1。

步骤 6 若 I2C 为主设备, 配置 IC_TAR.ic_tar, 设定 I2Cx 访问的 Slave 地址。

步骤 7 配置对应速度模式下的 I2Cx 总线时钟周期。

IC_SS_SCL_HCNT.ic_ss_scl_hcnt 与 IC_SS_SCL_LCNT.ic_ss_scl_lcnt、
IC_FS_SCL_HCNT.ic_fs_scl_hcnt 与 IC_FS_SCL_LCNT.ic_fs_scl_lcnt;

步骤 8 配置 IC_RX_TL.rx_tl、IC_TX_TL.tx_tl, 分别设定接收与发送 FIFO 的水线阈值。

步骤 9 若采用中断方式传输数据, 需配置 IC_INTR_MASK 相应的中断使能; 采用查询方式时应禁止产生相应中断信号。

步骤 10 配置 IC_ENABLE.enable 为 1, 使能 I2Cx, 完成初始化配置。

----结束



11.4.8.2 数据发送

- 步骤 1 将待发送的数据写入 IC_DATA_CMD.dat，并配置 IC_DATA_CMD.cmd 为 0，启动数据发送。在每发送一帧数据后，需要配置 IC_DATA_CMD.stop 或 IC_DATA_CMD.restart 发送 STOP 或者 restart 指令。
- 步骤 2 查询方式下，进行连续数据发送时，通过读取 IC_STATUS.tfe 与 IC_TXFLR.txflr，检测发送 FIFO 状态是否为空，tfe 为 1 表示空，txflr 表示发送 FIFO 中有效数据的个数。
- 步骤 3 中断方式下，通过读取 IC_INTR_STAT.r_tx_empty，检测发送 FIFO 状态是否为空。
- 步骤 4 在数据传送完成之前，既要保证发送 FIFO 中的数据没有溢出（IC_STATUS.tfnf 与 IC_INTR_STAT.r_tx_over，否则会造成数据丢失），同时也要保证发送 FIFO 中始终有数据（否则 I2C 会认为该次传输已全部完成而发出结束信号）。
- 步骤 5 查询方式下，通过检测 IC_STATUS.activity 是否为 0，判断 I2C 是否完成全部数据发送。
- 步骤 6 中断方式下，通过检测 IC_INTR_STAT.r_stop_det 是否为 1，判断 I2C 是否完成全部数据发送。

----结束

11.4.8.3 数据接收

- 步骤 1 配置 IC_DATA_CMD.cmd 为 1，启动数据接收。
- 步骤 2 进行连续数据接收时，首先要发送与接收数据数目相同的读命令到 IC_DATA_CMD 寄存器（例如要接收 3 个数据，则需发送 3 次读命令到 IC_DATA_CMD 寄存器）。在数据接收完成之前，既要保证发送 FIFO 中的数据没有溢出，又保证发送 FIFO 非空（否则 I2C 会认为没有新的数据需要接收而结束），另外在连续数据接收过程中还需要检测接收 FIFO 的状态（如果采用中断方式，则根据相应中断状态位检测），避免接收 FIFO 溢出。
- 步骤 3 通过检测 IC_STATUS.rff 是否为 1，如果为 1，则接收 FIFO 溢出，如果中断未屏蔽，检测 I2C.IC_INTR_STAT.r_rx_done，为 1，表示接收完成，以此判断 I2C 是否完成全部数据接收。

----结束



11.4.9 SMBUS 协议的支持

11.4.9.1 SMBUS 协议命令

SMBUS 协议命令 (2.0 定义了 11 个命令):

Quick Command, Send Byte, Receive Byte, Write Byte, Write Word, Read Byte, Read Word, Process Call, Block Read, Block Write, and Block Write-Block Read Process Call.

通过 I2C 的寄存器可实现上述命令。命令操作需参考 0。

表11-5 SMBus Bus 协议命令的使用

协议命令	Required TxFIFO commands (命令条数)	Command/data (IC_DATA_CMA[7:0])	CMD_bit (IC_DATA_CMA[8]) 1: READ 0: WRITE	Stop_bit (IC_DATA_CMA[9]) 1: 发送 STOP after command; 0: 不发 STOP after command	其他
Quick Command	1	Not Applicable	Set the command [RW]	Set to 1	Set IC_TAR[11] and IC_TAR[16] to 1
Send Byte	1	Data Byte	Set to 0	Set to 1	-
Receive Byte	1	Not Applicable	Set to 1	Set to 1	-
Write Byte	2	Command Code	Set to 0	Set to 0	-
		Data Byte	Set to 0	Set to 1	-
Write Word	3	Command Code	Set to 0	Set to 0	-
		Data Byte Low	Set to 0	Set to 0	-
		Data Byte High	Set to 0	Set to 1	-
Read Byte	2	Command Code	Set to 0	Set to 0	-
		Not Applicable	Set to 1	Set to 1	-
Read Word	3	Command Code	Set to 0	Set to 0	-
		Not Applicable	Set to 1	Set to 0	-
		Not Applicable	Set to 1	Set to 1	-
Process	5	Command Code	Set to 0	Set to 0	-



协议命令	Required TxFIFO commands (命令条数)	Command/data (IC_DATA_CMA[7:0])	CMD_bit (IC_DATA_CMA[8]) 1: READ 0: WRITE	Stop_bit (IC_DATA_CMA[9]) 1: 发送 STOP after command; 0: 不发 STOP after command	其他
Call		Data Byte Low	Set to 0	Set to 0	-
		Data Byte High	Set to 0	Set to 0	-
		Not Applicable	Set to 1	Set to 0	-
		Not Applicable	Set to 1	Set to 1	-
Block Write	N+2	Command Code	Set to 0	Set to 0	-
		Data Byte	Set to 0	Set to 0	-
		N+1) Data Byte N	Set to 0	Set to 1	-
Block Read	N+2	Command Code	Set to 0	Set to 0	-
		Not Applicable	Set to 0	Set to 0	-
		N+1) Not Applicable	Set to 0	Set to 1	-
Block Write Block Read Process Call	M+N+2	Command Code	Set to 0	Set to 0	-
		Data Byte 1	Set to 0	Set to 0	-
		M+1) Data Byte M	Set to 0	Set to 0	-
		M+2) Not Applicable	Set to 1	Set to 0	-
		M+3) Not Applicable	Set to 1	Set to 0	-
		M+N+1) Not Applicable	Set to 1	Set to 1	-

11.4.9.2 SMBUS PEC

SMBUS 协议引入了 PEC 字段，作为一帧数据的校验；对于 ARP 的命令强制要求需要传输 PEC 字段。

Master 的 PEC 的产生和校验，由软件来计算。计算结果作为最后一个 Byte 发送。

参考 CRC 计算多项式为 CRC8 $C(X)=X^8+X^2+X+1$ 。



11.4.9.3 SMBUS ARP 命令支持

SMBUS 定义了 128bit 的 Unique Device ID (UDID) 作为设备全网的唯一标识；IP 提供了 96bit 固定参数+32bit 寄存的方式来实现 SMBUS 的 UDID，参数在生成 IP 的过程确定。

表11-6 SMBus ARP 协议命令的实现

协议命令	TxFIFO commands	Command/data (IC_DATA_CMA[7:0])	CMD_bit (IC_DATA_CMA[8])	Stop_bit (IC_DATA_CMA[9])	操作说明
Prepare for ARP	2	Command = '0000 0001'	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Default Address (1100 001)
		PEC Byte	Set to 0	Set to 1	
Reset Device (General)	2	Command = '0000 0010'	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Default Address (1100 001)
		PEC Byte	Set to 0	Set to 1	
Get UDID (General)	20	Command = '0000 0011'	Set to 0	Set to 0	1.Set IC_TAR[6:0] to SMB Default Address (1100 001). 2. First Reads for the byte_cnts. 3.16 Reads to be performed for the 128 UDID bytes. 4. Last read command for the slave address.
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		PEC Byte	Set to 1	Set to 1	
Assign Address	20	Command = '0000 0100'	Set to 0	Set to 0	1. Set IC_TAR[6:0] to SMB Default Address (1100 001). 2. 16 Writes to be performed for the 128 UDID byte. 3. Last Write
		Byte Count = 17	Set to 0	Set to 0	
		UDID Byte 15	Set to 0	Set to 0	
		UDID Byte 14	Set to 0	Set to 0	
		Assigned Address	Set to 0	Set to 0	



协议命令	TxFIFO conmm ands	Command/data (IC_DATA_CMA[7:0])	CMD_bit (IC_DATA_CMA[8])	Stop_bit (IC_DATA_CMA[9])	操作说明
		PEC Byte	Set to 0	Set to 1	command for the Assigned slave address.
Get UDID (Directed)	19	Command = '00000011'	Set to 0	Set to 0	1. Set IC_TAR[6:0] to SMB Default Address (1100 001). 2. 16 Reads to be performed for the 128 UDID byte. 3. Last Read command for the slave address
		Slave address[6:0],1}	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		Not Applicable	Set to 1	Set to 0	
		PEC Byte	Set to 1	Set to 1	
Reset Device (Directed)	2	command = {slave address[6:0],0}	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Default Address (1100 001)
		PEC byte	Set to 0	Set to 1	
Notify ARP Master	3	Device Address ='1100 0010'	Set to 0	Set to 0	Set IC_TAR[6:0] to SMB Host Address (0001 000)
		Data Byte Low ='0000 0000'	Set to 0	Set to 0	
		Data Byte High ='0000 0000'	Set to 0	Set to 1	

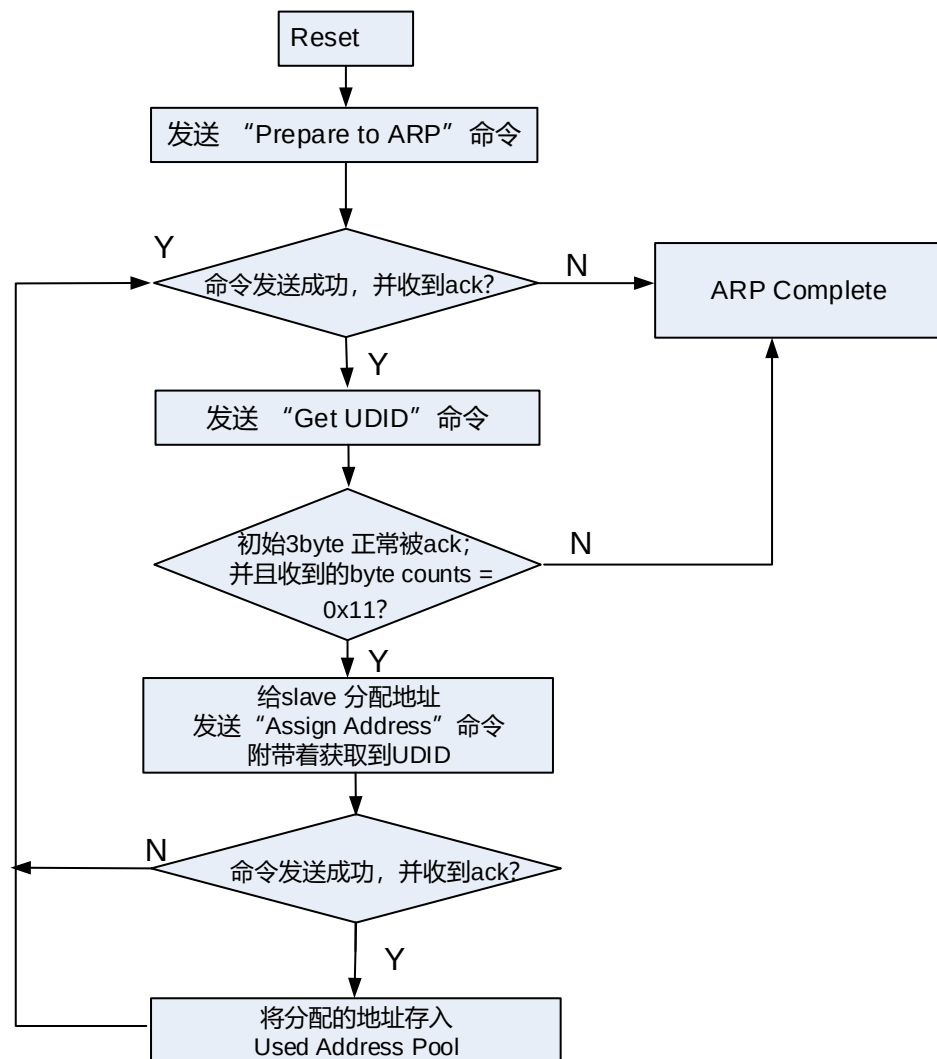
11.4.9.4 ARP 的配置流程

SMBUS 定义了 Host 和 Master，其中 Host 是进行 ARP 地址分配的 Master，并可监听 “Host Notify” 命令 (Notify ARP Master)。本文将进行 ARP 地址分配的 Master 或 Host 都称为 Host。ARP 处理流程如图 11-5 所示。

须知

本 IP 支持 Host、Master 和 Slave；由于 SMBUS 仅限于支持 Master，因此本文仅描述了 Master 侧的 ARP 流程，并对流程进一步做了简化以便于用户实现。

图11-5 SMBUS Master ARP 配置流程



ARP 的目的是 Host 将唯一地址分配给每个 Slave 设备，避免地址冲突，执行步骤如下：

- 步骤 1 复位或上电后，Master 发出 "Prepare to ARP" 命令通知其他设备将启动动态地址分配。Device 收到命令后将会清空 "host notify" 的命令发送。
- 步骤 2 Host 收到 "Prepare to ARP" 命令的 ACK，则表明在系统中存在 ARP 功能设备；必须发出"获取 UDID"命令。跳转到[步骤 3](#)。

如果收到 "Prepare to ARP" 命令的 NACK，表明 ARP 功能的设备不存在，或者所有的 slave 都有其 reserved 地址。在这种情况下，Host 跳转[步骤 8](#)。



NACK 是通过 IC_TX_ABRT_SOURCE 寄存器的 ABRT_7B_ADDR_NOACK 或 ABRT_TXDATA_NOACK 来判断的。

- 步骤 3 SMBus Host 发送 "Get UDID" 命令来获取 Slave 的 UDID 信息。
- 步骤 4 如果 "Get UDID" 命令的前三个 Byte 是 ACK，并且接收字节计数为 0x11，则 Host/Master 将发出 "Assign Address" 命令。否则跳转到步骤 8，以指示 ARP 已完成。
- NACK 是通过 IC_TX_ABRT_SOURCE 的 ABRT_7B_ADDR_NOACK 或 ABRT_TXDATA_NOACK 位来确认判断。
- 步骤 5 Host 发出 "Assign Address" 命令，将动态地址分配给前面获取到 UDID 对应的设备。
- 步骤 6 如果 Slave 对 "Assign Address" 命令回应了 ACK，则 Host 将从 Adress Pool 的“待分配的地址”删除该地址，然后移动到步骤 3 以获取另一个 Slave 的 UDID。如果数据包不是 ACK，则将不会从“地址池”中删除地址，并移动到步骤 3 以获取相同 Slave 或其他 Slave 的 UDID。
- 步骤 7 如果 Slave 对 "Assign Address" 命令回应了 ACK，则 Host 将该地址 + 对应设备 UDID 存入 Adress Pool 的“已分配地址”。
- 步骤 8 Host 步骤 3 将再次发出 "Gets UDID" 命令，以接收其他 Slave 的 UDID。如果它收到 NACK，则 ARP 完成。可开始正常数据收发。

----结束

须知

Adress Pool 地址池的维护和操作需软件实现。
更多处理细节请具体参考 SMBUS 协议。

11.4.10 相关寄存器基址

表11-7 Hi3093 SMBUS 相关模块寄存器基址

模块名称	模块基址
LP_SUBSYS_CSR	0x0871D000
SEC_CSR_REG	0x08533000



模块名称	模块基地址
I2C 0~15	IIC0:0x08707000 IIC1:0x08708000 IIC2:0x08709000 IIC3:0x0870A000 IIC4:0x0870B000 IIC5:0x0870C000 IIC6:0x0870D000 IIC7:0x0870E000 IIC8 :0x08750000 IIC9 :0x08751000 IIC10:0x08752000 IIC11:0x08753000 IIC12:0x08754000 IIC13:0x08755000 IIC14:0x08756000 IIC15:0x08757000
安全 I2C	SEC_IIC:0x08538000

须知

SMBUS 和 I2C 的寄存器为相同地址。

11.4.11 状态和记录

- 寄存器 I2C_ENABLE.enable 表示 I2C 是使能。
- 寄存器 IC_STATUS.i2c_activity 表示 I2C 的活跃状态。
- IC_STATUS.rff 表示 I2C 接收 FIFO 的满状态。
- IC_STATUS.rfne 表示 I2C 接收 FIFO 的空状态。
- IC_STATUS.tff 表示 I2C 接收 FIFO 的满状态。
- IC_STATUS.tfne 表示 I2C 接收 FIFO 的空状态。
- IC_TXFLR 提供接收 FIFO 中数据字节计数。
- IC_RXFLR 提供发送 FIFO 中数据字节计数。
- IC_TX_ABRT_SOURCE 提供了产生 TX_ABRT 的原因记录。



11.4.12 内部关键信号

寄存器 LP_SUBSYS_CSR.LPSC_I2C0_DEBUG~ LPSC_I2C15_DEBUG 提供 SMBUS (I2C) 一些关键信号的观测。



12 UART

12.1 概述

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

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

Hi3093 共包含 9 路 UART 控制器：

UART0、1 被 LPC/eSPI 控制和访问，称为 HOST UART。

UART2~7 被 A55 或 M3 控制和访问，称为通用 UART。

1 路 UART 仅被安全 M3 控制和访问，称为安全 UART。

本章节主要描述 UART 工作方式和全互联特性，包括了 6 路 UART 的工作描述。

须知

HOST UART 和安全 UART 不支持 DMA 传输方式。

安全 UART 不支持到其他物理端口的全互联映射。

安全 UART 的特性和接口复用等描述请参见 0 章节。



12.2 特性

UART 控制器具有以下特性：

- HOST UART (UART0~1) 和安全 UART 支持中断方式和查询方式进行数据搬运。
- 通用 UART (UART2~7) 支持 DMA 数据搬运，支持以中断方式和查询方式进行数据搬运。
- UART (UART2~7) 内部包括 128*8Byte 的发送 FIFO 和 128*10Byte 的接收 FIFO。HOST (UART0~1) 内部包括 32*8Byte 的发送和 32*10Byte 的接收。
- 支持 FIFO 禁止或使能。
- 支持 5/6/7/8bit 可编程数据位。
- 支持 1/1.5/2bit 可编程停止位。
- 支持奇、偶校验方式，支持无校验位，支持校验位为软件设定值。
- 支持一个组合中断连接到中断控制器，中断类型包括：接收 FIFO 中断、发送 FIFO 中断、接收超时中断、错误中断。
- UART 支持传送速率可编程，传送速率范围为 100Mbps/(16*N) 0<N<65536，N 为整数)。
- HOST 侧 UART 只支持 115200 及以下波特率，配置 HOST UART 侧波特率时按 X86 标准 1.8461M 时钟计数分频系数，1.8461Mbps/(16*N) (0<N<65536，N 为整数)。
- 支持关断 UART 时钟以降低功耗，UART2~7 能单独关断时钟，UART0、1 随 SUPERIO 整体关断时钟。
- UART1~UART4 每个控制器分别支持 1 路 SOL，4 路 SOL 的收发数据循环记录，缓存空间收发各 32MB，可配置为 0.25M、0.5M、1M、2M、4M、8M、16M、32M。
- 支持 8 个 UART 控制器和 8 路 UART 物理接口之间的映射关系可配置，如图 12-1 所示。UART 到 PORT 的 TX 方向全互联。
- 片内一共 8 个 UART 控制器：UART0~UART7，每个 UART 包括独立的地址和中断线。每个 UART 控制器对外均支持 7 线模式（兼容 3 线和 5 线）。
- 芯片对外的 UART 物理端口，请参见 46 IOCONFIG 章节。

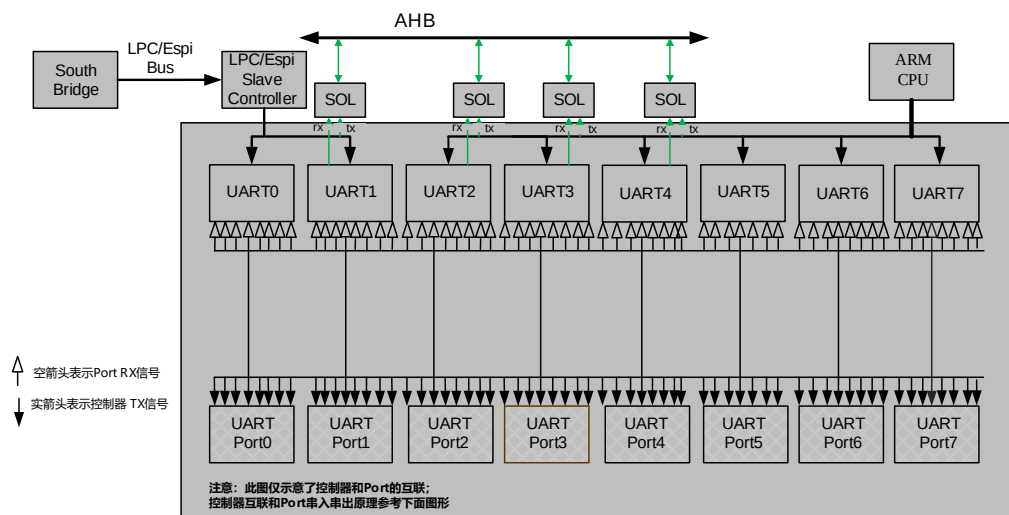


须知

4 路 SOL 中，3 路为通用 UART，另外 1 路在 HOST 侧支持 LPC/eSPI。

UART 接口控制示意图如图 12-1 所示。

图12-1 UART 控制器与接口管脚映射图



- UART0、UART1 支持 HOST 通过 LPC/Espi 访问。
- UART2、UART3...UART7 支持 ARM CPU 访问。
- UART0~UART7 控制器支持全功能模式。
- UART Port0 接口管脚支持全功能模式。
- UART0~UART7 控制器之间支持片内互联，且支持流控（不支持 7 线模式互联）。
- 支持对 UART1 进行串口记录，即支持将 UART1 的收发数据循环写入特定的内存空间。
- 支持各 UART 控制器与接口管脚的映射关系可配置。

由于工作频率固定，在支持某些标准的波特率时会有误差，表 12-1 给出以 100MHz 工作时的误差率，当速率越高时，误差将会越小。



表12-1 波特率误差

波特率 (bps)	分频比	误差率
9600	977	0.04%
56k	167	0.25%
187.5k	50	0.00%

说明

常用波特率 57600bps 可以通过分频比配置得到，但是不一定能完全精确到该速率。

12.3 功能描述

12.3.1 典型应用

UART 的典型应用分 3/5/7 线模式，其中 7 线兼容 3 线或 5 线，接口模式与管脚的对应关系。

表12-2 UART 接口模式

名称	方向	3 线模式	5 线模式	7 线模式
UART_RXD	I	接收信号	接收信号	接收信号。
UART_TXD	O	发送信号	发送信号	发送信号。
UART_DTR_N	O	不涉及	不涉及	本端准备好信号（可收发）。
UART_CTS_N	I	不涉及	对端接收准备好指示	半双工：RTS 的应答信号。 全双工：始终输入有效。
UART_DSR_N	I	不涉及	不涉及	Modem 准备好信号（可收发）。
UART_RTS_N	O	不涉及	本端接收准备好指示	半双工：发送切换请求信号。 全双工：始终输出有效。



名称	方向	3 线模式	5 线模式	7 线模式
UART_DCD_N	I	不涉及	不涉及	Modem 检测到数据载波指示信号。
UART_RI_N	I	不涉及	不涉及	振铃检测信号。

须知

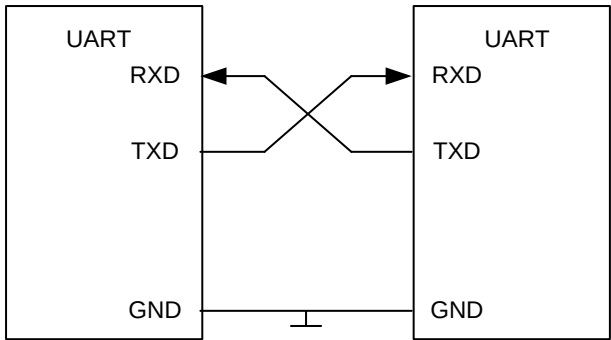
UART 接口 3/5/7 线模式均包括地线，表 12-2 中未列出。

UART 控制器支持 3/5/7 线模式。仅 UART Port0 保留了 7 线管脚。因此此芯片仅支持 UART Port0 作为全功能接口。其他 Port 仅支持 3 线/5 线模式。

具体请参考《Hi3093 硬件设计指南》管脚复用相关章节。

12.3.1.1 UART 3 线模式

图12-2 UART 3 线模式应用框图

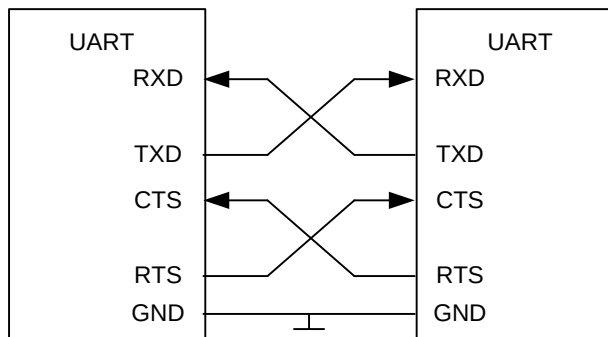


无流控，认为接收端总是处于准备好的状态，发送端有数据就启动发送。



12.3.1.2 UART 5 线模式

图12-3 UART 5 线模式应用框图

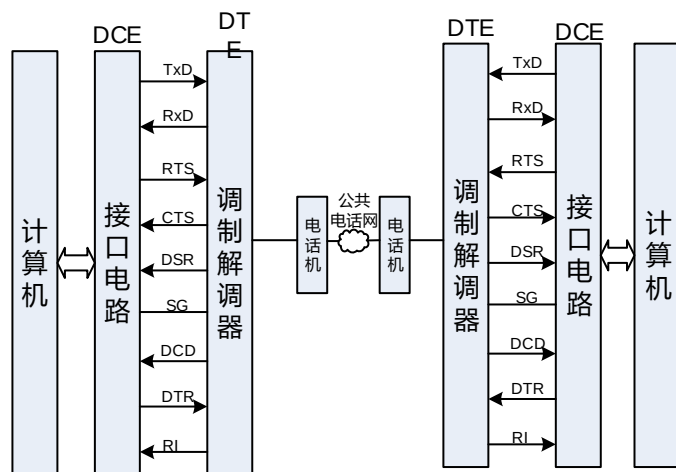


5 线模式下 RTS 和 CTS 用于流控：

- 本端的接收 FIFO 快满时，UART 控制器会自动把 RTS 设为无效，对端的 CTS 检测到 RTS 无效时，停止发送数据。
- 本端的接收 FIFO 变空时，UART 控制器会自动把 RTS 设为有效，对端的 CTS 检测到 RTS 有效后，开始发送数据。

12.3.1.3 UART 7 线模式

图12-4 UART 7 线模式应用框图



7 线模式下，Hi3093 和调制解调器 (Modem) 相连。



须知

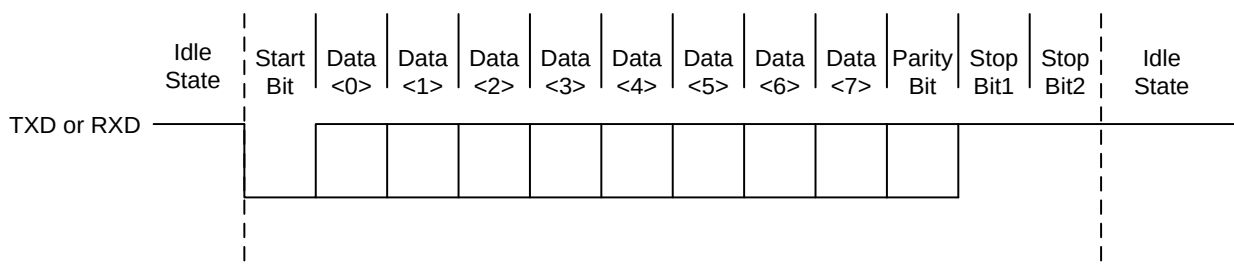
7 线模式下，RTS 和 CTS 的含义与 5 线模式完全不同，7 线模式下 RTS 和 CTS 用于发送请求和应答：

- 半双工时：RTS 用于通知对端，本端有数据要发送，CTS 用于对端应答，通知本端，对端可以接收。本端检测到 CTS 有效即开始发送数据。
- 全双工时：RTS 始终输出有效，CTS 始终输入有效。

12.3.1.4 功能原理

UART 的一次帧传输主要包括起始信号、数据、校验位和结束信号，如图 12-5 所示。数据帧从 UART 的 TXD 端输出，进入对接 UART 的 RXD 端输入。

图12-5 UART 帧格式



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

12.3.1.5 自动流控功能

UART 的自动流控功能 (AutoflowControl) 通过/RTS (发送请求) 和/CTS (发送清除) 实现，用于监控因 FIFO 的空/满而产生的溢出错误。

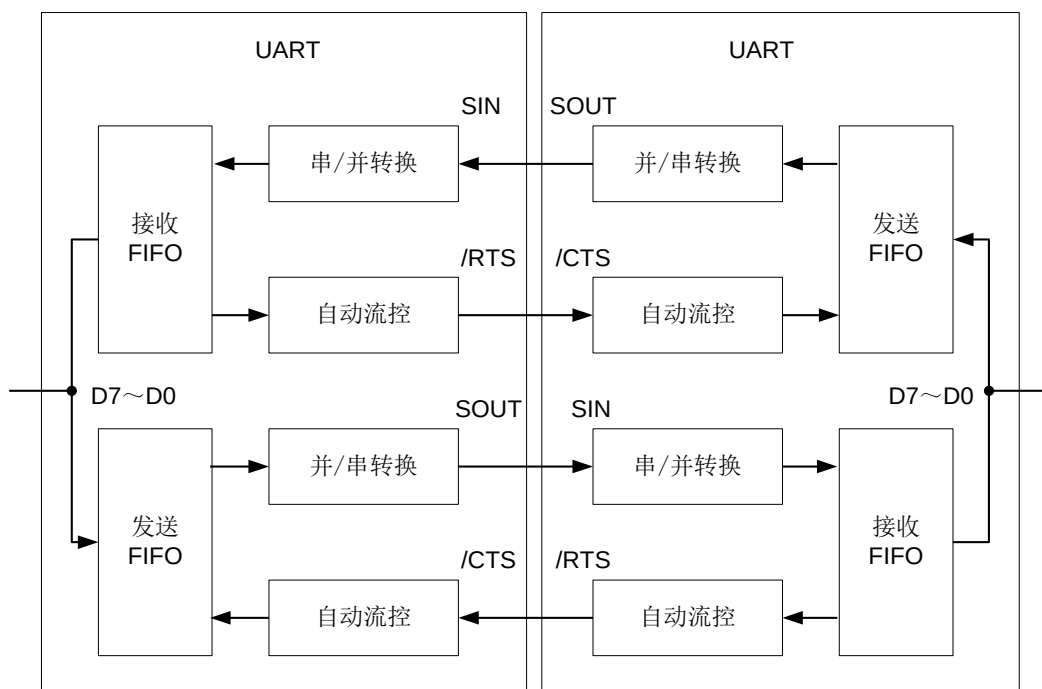


当发送 FIFO 发送数据或接收 FIFO 接收数据时，如果/CTS 或/RTS 有效（低电平），则可以避免在发送/接收数据过程中因出现 FIFO 空/满而产生的溢出错误。

可通过设置模式控制寄存器 UARTx_MCR bit[5]为 1 启动自动流控功能。设置 UARTx_MCR bit[1]为 1 通知对接器件 UART 已准备就绪，可以进行数据发送和接收操作。

UART 自动流控功能框图如图 12-6 所示。

图12-6 UART 5 线模式结构框图



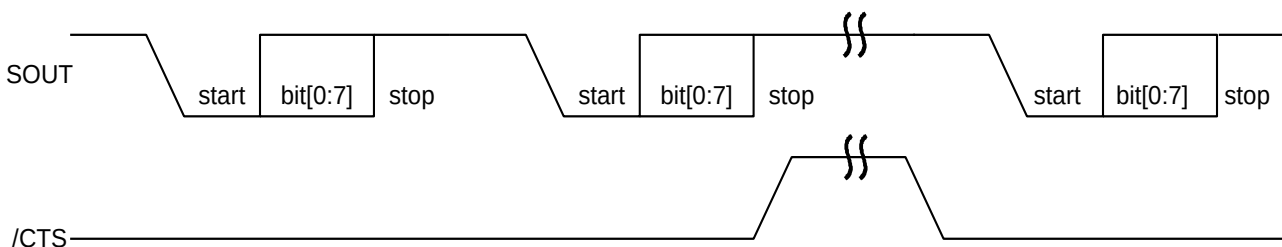
12.3.1.6 发送清除 (/CTS)

UART 发送机在发送数据之前，不断地检测/CTS 信号的状态。当/CTS 为低电平（有效）时，SOUT 发送数据；当/CTS 为高电平时，SOUT 停止发送数据。

/CTS 与 SOUT 的功能时序图如图 12-7 所示。



图12-7 /CTS 功能时序图



当/CTS 信号为低电平时，则 SOUT 持续发送串行数据；当/CTS 信号在 stop 信号发送过程中变为高电平，则 SOUT 发送完当前 Byte 后停止发送；当/CTS 信号由高电平变为低电平，则 SOUT 又开始发送数据。

12.3.1.7 发送请求 (/RTS)

/RTS 操作与 UART 接收 FIFO 的水线有关，/RTS 有效表示可以接收数据。

接收 FIFO 的水线可以设置为 1、1/4FIFO 深度、1/2FIFO 深度，FIFO 深度-2Byte，当接收 FIFO 达到水线时，接收机会置/RTS 无效，通知发送机停止发送数据。

须知

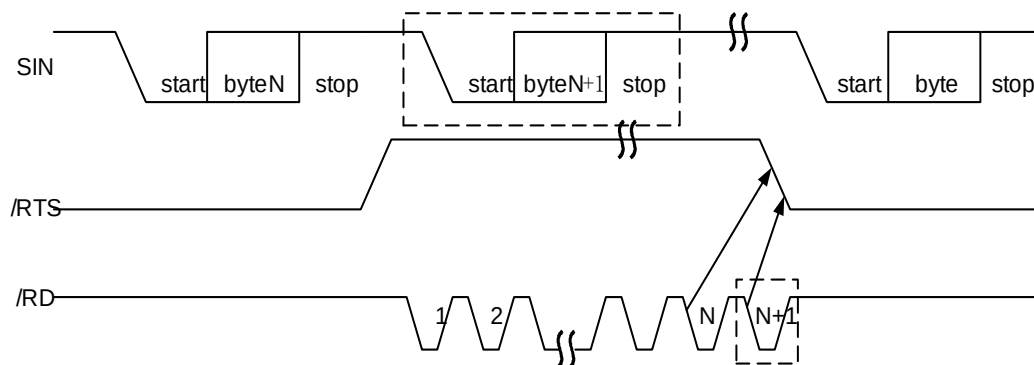
由于 UART 发送机检测到停止发送信息时，可能已经有 1Byte 数据处于发送状态，所以接收机置/RTS 无效后，还会收到 1Byte 数据，所以设置 FIFO 水线在 FIFO 满-2Byte 较合适。

当接收 FIFO 的水线为 14Byte 时，在发送机发送到第 16Byte 数据的第 1bit 时，/RTS 被拉高，发送完当前 Byte 后停止发送，当接收 FIFO 至少有 1Byte 空间时，再次启动 /RTS。

/RTS 与 SIN 的时序图如图 12-8 和图 12-9 所示。



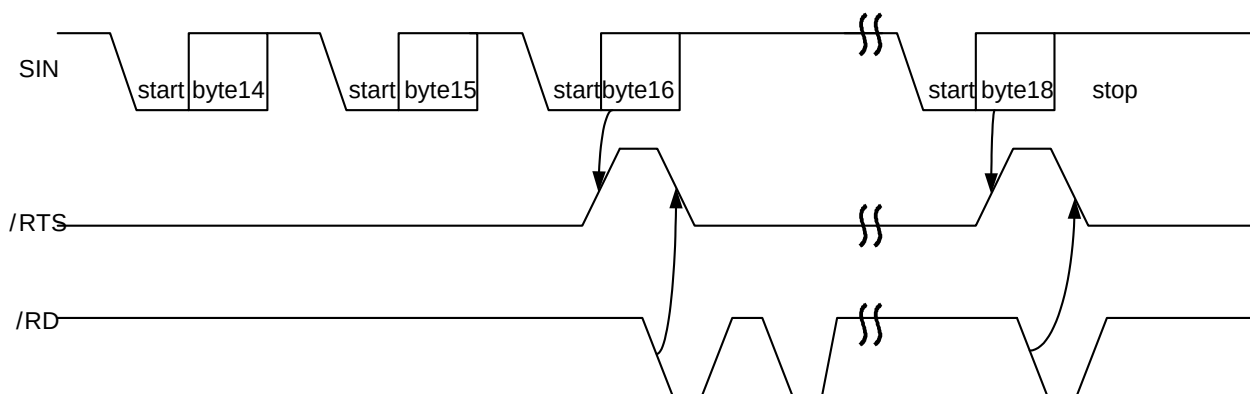
图12-8 /RTS 时序图 (接收 FIFO 水线为 1/4/8Byte)



须知

N 为接收 FIFO 的水线，为 1/4/8Byte，虚框中为发送机在达到接收 FIFO 水线后额外发送的 1Byte 数据。

图12-9 /RTS 时序图 (接收 FIFO 水线为 14Byte)



须知

/RTS 在 UART 接收到第 16Byte 的第 1bit 时开始失效（拉高），接收 FIFO 在接收完第 16Byte 后触发 FIFO 满中断。当接收 FIFO 有多于 1Byte 的空闲空间且没有正待接收的数据时，启动/RTS；当接收 FIFO 满时，通过读寄存器 PBR 再次启动/RTS。



12.3.1.8 同步功能

说明

UARTx 表示 UART0, UART1, UART2 或 UART3。

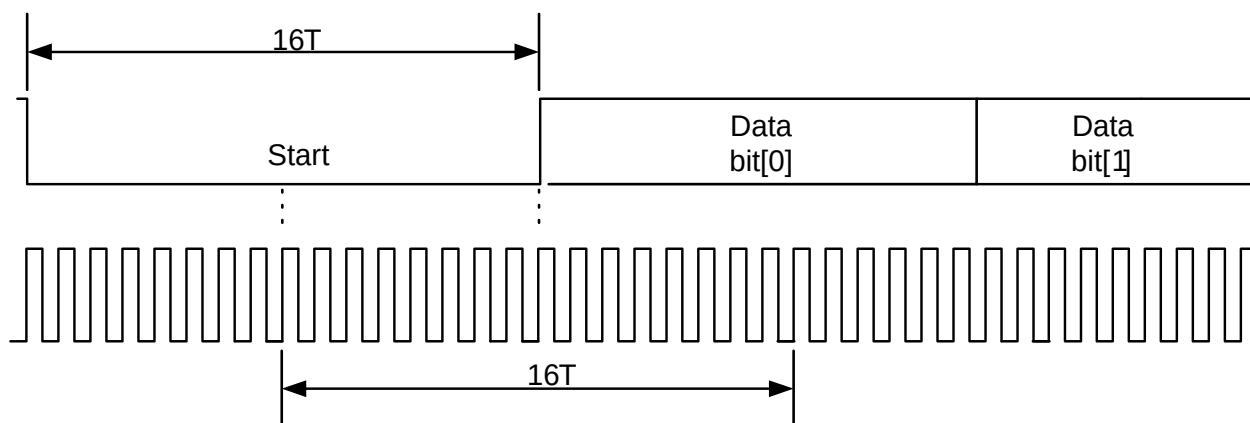
异步串行通信的关键是要保证通信双方收发同步。由于没有随路时钟，所以双方必须约定一个固定的频率，用于同步收发数据，这样才能保证数据传送的正确性。

UART 的同步就是寻找起始位来确定一帧数据的开始。UART 的同步过程如下：

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

数据采样过程如图 12-10 所示。

图12-10 数据采样过程



通过配置寄存器 UARTx_DLL 和 UARTx_DLH (x=0 或 1)，可设置 UART 工作的波特率，波特率计算公式如下：

波特率=总线时钟频率/(16*分频系数)。

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

例如当内部总线时钟为 144MHz 时，如果配置 UARTx_DLL 为 0x000000_4E，UARTx_DLH 为 0x0000_0000，按照波特率计算公式，则表示当前的波特率为 144/(78*16) = 115200bps。



UART 波特率配置的典型值有：9,600bps, 14,400bps, 19,200bps, 38,400bps, 57,600bps, 76,800bps, 115,200bps。

12.4 工作方式

说明

UARTx 表示 UART0、UART1、UART2.....UART7。

须知

UART 模块进行模式配置时，必须将数据收发工作停止，否则不能正确配置。

12.4.1 管脚复用配置

UART0~UART7 上电复位后默认功能为 UART，不需要配置。

如图 12-11 所示，UART 控制器到管脚之间存在三级复用：

- UART 全互联为第一级复用。
- UART_IOMUX 为第二级复用。
- IOCONFIG 为第三级复用。

IOCONFIG 中的 UART 端口：

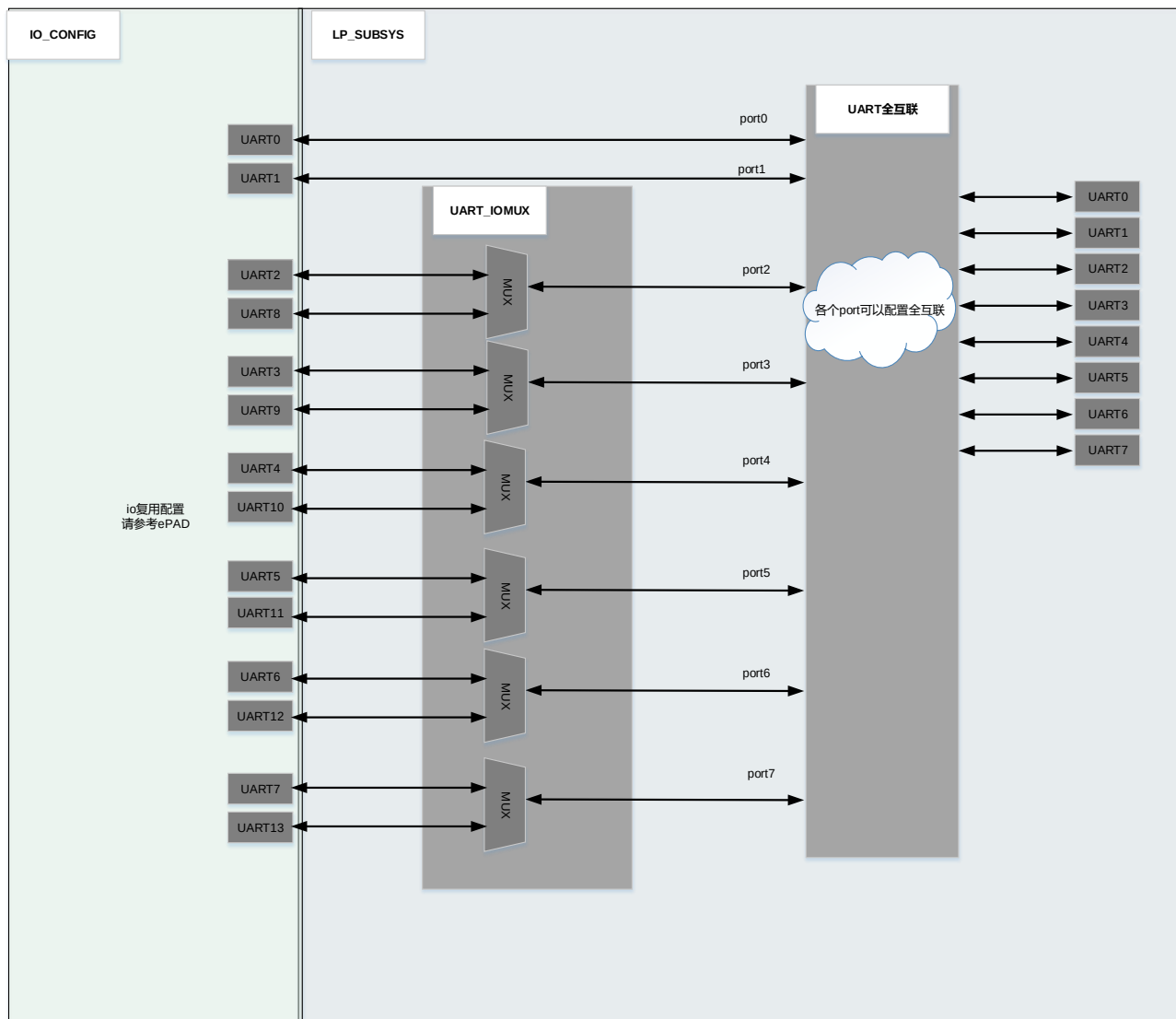
- UART0 为全功能端口。
- UART9~UART13 端口为 5 线模式端口。

通过 UART_IOMUX 可实现：

- UART2 控制器可以选择去 UART2 端口和 UART8 端口。
- UART3 控制器可以选择去 UART3 端口和 UART9 端口。
- UART4 控制器可以选择去 UART4 端口和 UART10 端口。
- UART5 控制器可以选择去 UART5 端口和 UART11 端口。
- UART6 控制器可以选择去 UART6 端口和 UART12 端口。
- UART7 控制器可以选择去 UART7 端口和 UART13 端口。

UART_IOMUX 请参考《Hi3093 硬件设计指南》。

图12-11 UART 的 IOMUX



须知

对外物理端口可能有 14 个，通过 UART_IOMUX 来实现，并和全互联模块实现分工和隔离。

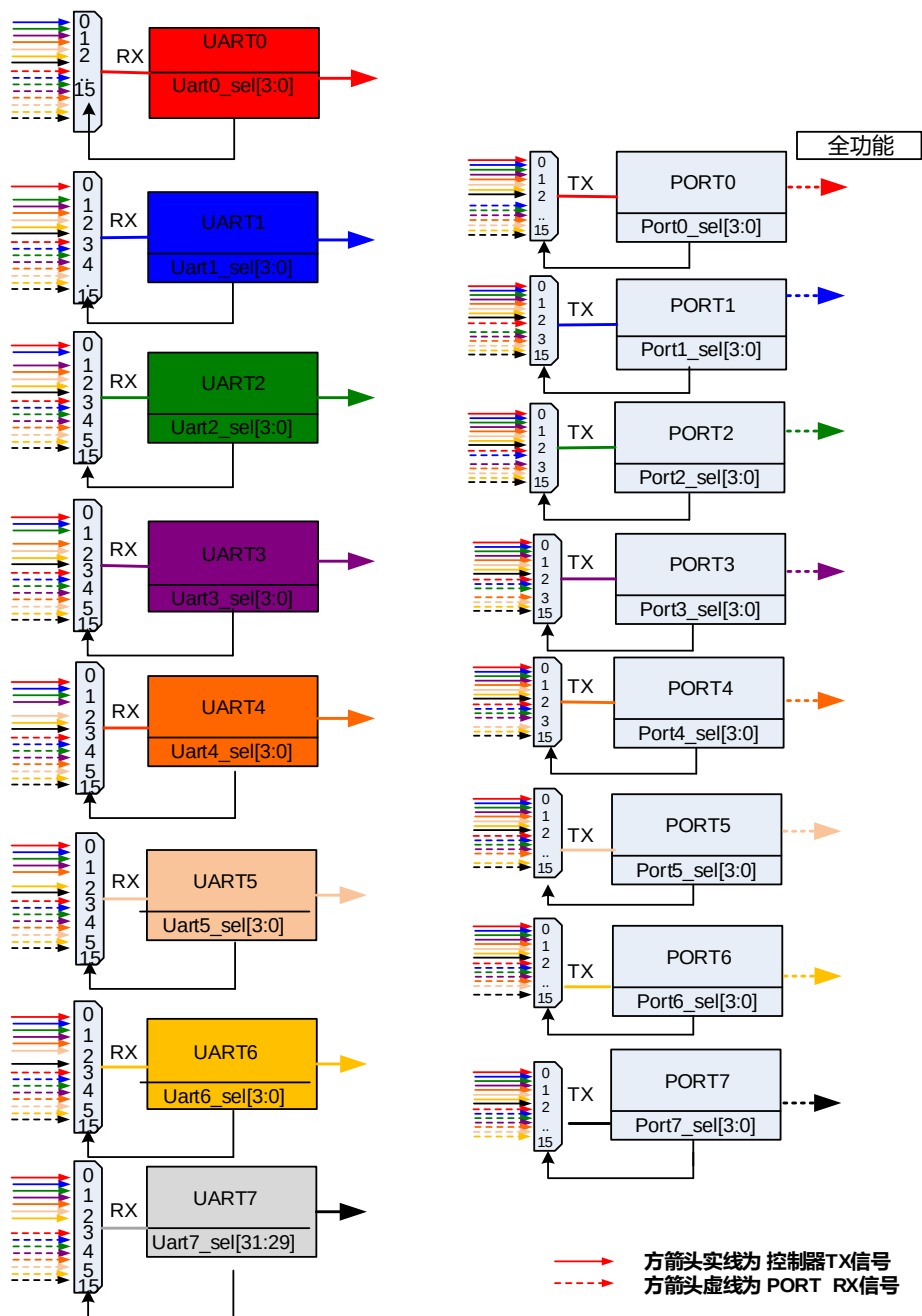
为避免混淆，全互联中仍按照 8 个 UART 物理端口进行描述，对应图 12-11 的 port0~7。



12.4.2 UART 全互联

UART 控制器和 UART PORT 直接可通过多路选择控制实现不同的映射关系；除此之外，还可实现 UART 控制器的内部互联，以及 UART PORT 的串入串出功能。如图 12-12 所示，这些互联选择通过 Uart_sel 和 Port_sel 寄存器控制。

图12-12 UART 接口控制示意图



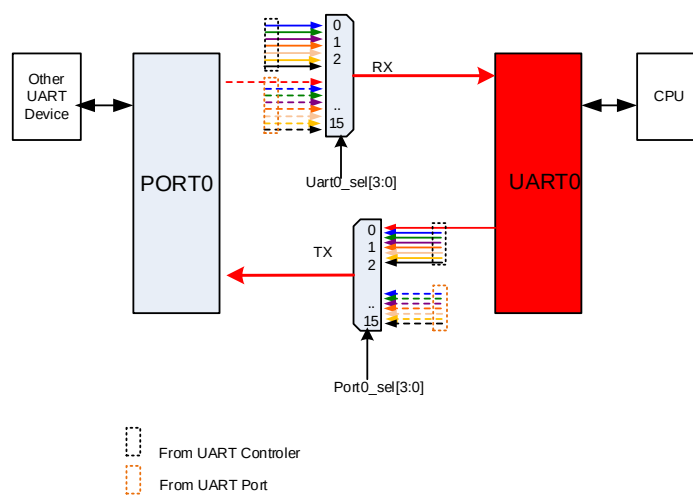


12.4.2.1 UART 全互联场景

上电复位后，默认 UART0~UART7 与物理端口 Port0~ Port7 一一对应连接。

同时 UART 和 PORT 之间有 2 个多路选择器，分别在 UART 到 PORT 的数据发送方向和 PORT 到 UART 的数据接收方向。通过寄存器 port_sel 和 uart_sel 可选择数据的来源。

图12-13 UART 全互联原理



通过全互联可实现 UART 到 Port 的全映射互联，UART 的内部互联和 UART Port 的串入串出。

图12-14 PORT 和 UART 的非默认互联

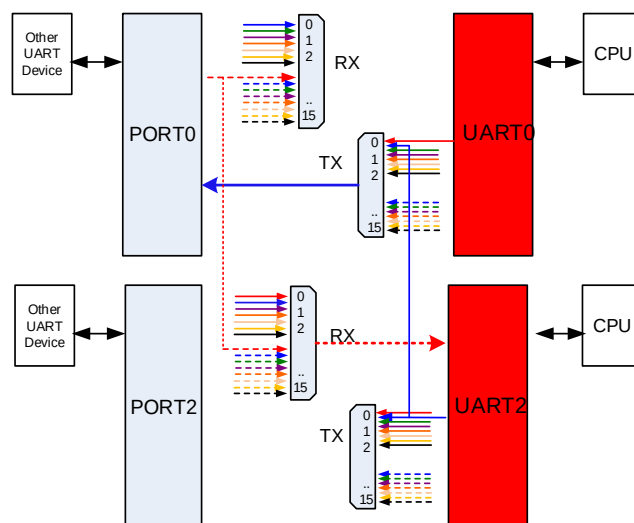


图12-15 UART 内部互联

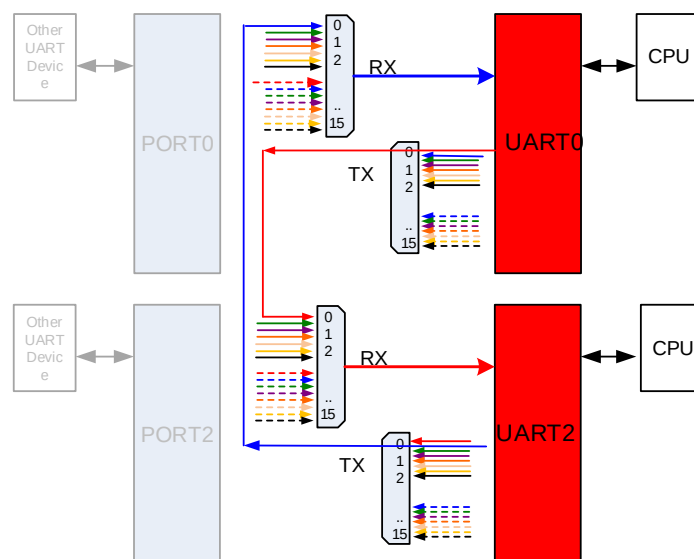
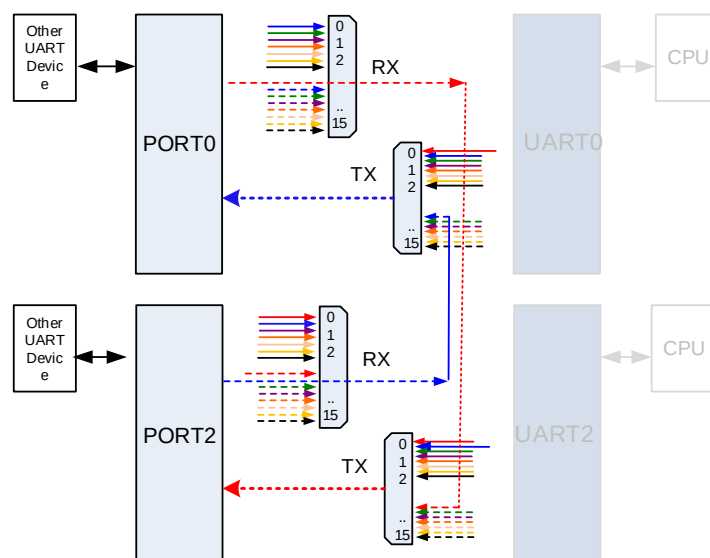


图12-16 PORT 的串入串出



12.4.2.2 全互联的配置方法

参考《Hi3093 寄存器手册》，UART 控制接收选择和 UART 物理 Port 接收选择对应 2 组寄存器，并有开关寄存器控制 UART 和 PORT 通路使能，当 PORT 或 UART 控制器不使用时，可关闭相应的通路。



UART（控制器）到 PORT 的互联选择，支持 UART 控制器的串联，支持 UART PORT 的串入串出。若要修改映射关系应整体进行修改，软件需保证配置关系不冲突，对不使用的 UART 或 Port 关闭其通路。

具体配置步骤如下：

步骤 1 配置 LP_SUBSYS.LPSC_UART_SEL.port_enable 为 0。

配置 LP_SUBSYS.LPSC_UART_SEL.uart_enable 为 0；关闭所有的 UART 控制器和 PORT 通路。

步骤 2 配置 LP_SUBSYS.LPSC_UART_CTRL1 寄存器。选择 PORT 的 RX 来源，详见寄存器具体描述。

步骤 3 配置 LP_SUBSYS.LPSC_UART_CTRL0 寄存器。选择 UART 控制器的 RX 来源，详见寄存器具体描述。

步骤 4 配置 LP_SUBSYS.LPSC_UART_SEL.port_enable。

配置 LP_SUBSYS.LPSC_UART_SEL.uart_enable。

使能使用的 uart 控制器和 port 物理口的通路选择。

---结束

须知

UART 和 PORT 的选择控制应该保证对应关系，比如 UART A 的发送选择送给 PORT B，则相应的 UART A 的接收应该也选择 PORT B。

当 UART 物理端口 Port A 和 B 进行串入串出配置，则对应的 UART 控制器 A 和 B 的 uart_enable 要配置成 0，关闭使用。

同理，当 UART 控制器 A 和 B 进行内部，那么对应的物理端口 Port A 和 B 的 port_enable 也需要配置成 0，关闭使用。

12.4.3 时钟门控

上电时 8 个 UART 时钟均默认为打开状态。UART 工作完成后，可以关闭 UART 时钟，以降低芯片功耗。UART2...UART7 使用不同的门控信号，它们的时钟可独立打开或关闭。UART0、UART1 整体随 SUPERIO 时钟关闭而关闭。



12.4.3.1 打开时钟

UART2~UART7 具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS_CSR.LPSC_CLKEN_CTRL1.uartX_clk_enb (X 为 2~7) 为 1，打开 UART X 模块的时钟门控。
- 步骤 2 读系统控制器寄存器 LP_SUBSYS_CSR.LPSC_CLKSTA_CTRL1.uartX_clk_status (X 为 2~7) 获取当前时钟门控信息。若为 0，则返回[步骤 1](#)；若为 1，则表示 UART X 时钟已经打开。

----结束

12.4.3.2 关闭时钟

- 步骤 1 配置 LP_SUBSYS_CSR.LPSC_CLKDIS_CTRL1.uartX_clk_dis (X 为 2~7) 为 1，关闭 UART X 模块的时钟门控。
- 步骤 2 读系统控制器寄存器 LP_SUBSYS_CSR.LPSC_CLKSTA_CTRL1.uartX_clk_status (X 为 2~7) 获取当前时钟门控信息。若为 1，则返回[步骤 1](#)；若为 0，则表示 UART X 时钟关闭。

-----结束

12.4.4 波特率配置

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

当前波特率=内部总线时钟频率/(16*分频系数)。

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

例如内部总线时钟为 100MHz，如果配置 UART_DLL 为 0x36，UART_DLH 为 0x00，按照波特率计算公式，则当前的波特率为 $100/(54*16)=115200\text{bps}$ 。

UART 波特率配置的典型值如下：

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



12.4.5 中断或查询方式下的数据传输

12.4.5.1 初始化

配置步骤如下：

- 步骤 1 配置线性控制寄存器 UART_LCR bit[7]为 1，使 UART_DLL 与 UART_DLH 可访问。
- 步骤 2 写相应的配置值到低位分频系数寄存器 UART_DLL 和高位分频因子寄存器 UART_DLH，配置传输速率。
- 步骤 3 配置寄存器 UART_LCR bit[6:0]，设定 UART 的工作模式。
- 步骤 4 配置 FIFO 控制寄存器 UART_FCR 使能或者关闭 FIFO，如果使能 FIFO，设定相应的发送及接收 FIFO 水线值，并配置配置寄存器 UART_LCR bit[3]为 1，使能 DMA 功能。
- 步骤 5 中断方式下配置中断使能寄存器 UART_IER 相应位为 1，使能相应中断；查询方式下配置中断使能寄存器 UART_IER 相应位为 0，禁止中断。

----结束

12.4.5.2 数据发送

具体配置步骤如下：

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

----结束

12.4.5.3 数据接收

具体配置步骤如下：



步骤 1 查询方式下，读取接收 FIFO 水线寄存 UART_RFL 和 UART 状态寄存器 UART_USR 检测接收 FIFO 的状态，根据接收 FIFO 的状态决定是否读取接收 FIFO 中的数据；中断方式下，根据相应中断状态位检测决定是否读取接收 FIFO 中的数据。

步骤 2 读取寄存器接收数据缓存寄存器 UART_RBR，获取接收的数据。

----结束

12.4.6 DMA 方式下的数据传输

12.4.6.1 初始化

具体配置步骤如下：

步骤 1 配置线性控制寄存器 UART_LCR bit[7]为 1，使 UART_DLL 与 UART_DLH 可访问。

步骤 2 写相应的配置值到低位分频因子寄存 UART_DLL 和高位分频因子寄存器 UARTx_DLH，配置传输速率。

步骤 3 配置寄存器 UART_LCR bit[6:0]，设定 UART 的工作模式。

步骤 4 配置寄存器 UART_FCR，使能或者关闭 FIFO。如果使能 FIFO，还需要设定相应的发送及接收 FIFO 水线值。

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

----结束

12.4.6.2 数据发送

具体配置步骤如下：

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

步骤 2 通过 DMA 中断上报，判断数据是否发送完成。

----结束

12.4.6.3 数据接收

具体配置步骤如下：

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

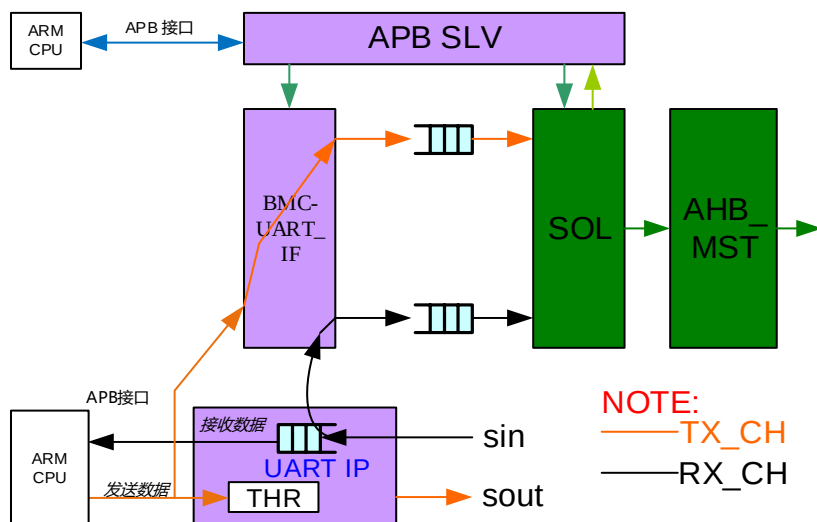
步骤 2 通过 DMA 中断上报，判断数据是否接收完成。

----结束

12.4.7 配置 SOL

通过 SOL 可记录 UART 串口收发两个方向的数据，并保存在 DDR 中预先分配好的位置。支持 SOL 的整体框图如图 12-17 所示。

图12-17 通用 UART SOL 逻辑框图



整体框图主要分成：APB_SLAVE、UART_IF、SOL 三部分。

- APB_SLAVE：解析 APB 时序配置 SOL 参数。
- UART_IF：将 UART 收/发数据各自合并成 32bit 写入 FIFO。
- SOL：控制 AHB MASTER 将 UART 收/发数据写入 DDR，并上报写 DDR 指针。

启动 SOL 操作步骤如下：

步骤 1 配置 SOL_TX_START_ADDR_L 寄存器，配置 TX 缓存区起始地址。

步骤 2 配置 SOL_TX_DDR_SIZE.tx_ddr_size 寄存器，配置缓存区大小。



步骤 3 配置 SOL_RX_START_ADDR_L.tx_start_addr_l 寄存器，配置 RX 缓存区起始地址。

步骤 4 配置 SOL_RX_DDR_SIZE.rx_ddr_size 寄存器，配置缓存区大小。

步骤 5 配置 SOL_START.sol_en 开工信号寄存器为 1，开始 SOL 数据缓存

----结束

读取 SOL 数据的步骤如下：

步骤 1 读 TX_DDR_WADDR_L bit[31:0]获取发送通道缓存当前写指针。

步骤 2 读 RX_DDR_WADDR_L bit[31:0]获取接收通道缓存当前写指针。

步骤 3 当前写指针表示最新的数据存储的位置，依据当前写指针从 DDR 中读取需要的数据。

----结束

须知

由于 SOL 内部收发单通道采用 BURST8 操作，所以在单通道未满足 32 个 Byte 数据量时，不会写入 DDR。也就是单通道 UART 收发数据中有可能残留 31 个 Byte 或者小于 31 个 Byte 在 SOL 内部。

12.4.8 配置校验位

校验位 (ParityBit，就是第 9bit)：校验位是 1bit 纠错信号。

UART 的校验位有奇校验、偶校验和固定校验，同时支持校验位的使能和禁止，详细描述请参见《Hi3093 寄存器手册》中的 UARTLCR 寄存器。

须知

寄存器在 uart busy 为 0 (空闲状态) 配置才生效。



13 SPI

13.1 概述

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

13.2 特性

SPI 具有以下特性：

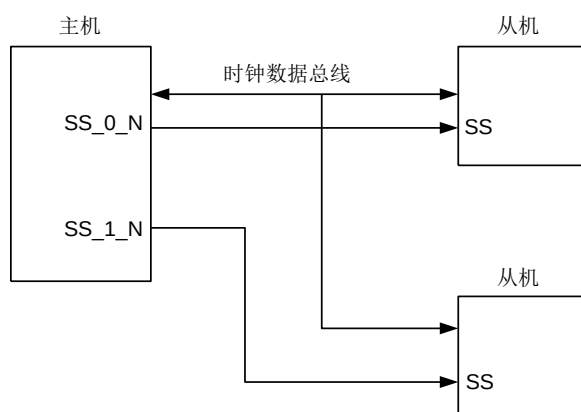
- 支持串行主设备操作。
- 支持可编程数据传输速率，传输速率配置范围为 200Mbps/N ($6 \leq N < 65534$ ，偶数)，最大传输速率可达到 20Mbps。
- 支持作为主设备，可以对接 2 个从设备。
- 支持接口时钟频率可编程。
- 支持 16bit*16 的接收 FIFO 和 16bit*16 的发送 FIFO。
- 支持 Moto 和 TI 帧格式
- 数据帧大小可编程：4bit ~ 16bit。
- 支持 DMA 功能。
- 支持 2 个片选。
- 支持 2 个 SPI 控制器
- 支持每个片选的有效极性可配置。

13.3 功能描述

13.3.1 应用框图

SPI 接口的应用框图如图 13-1 所示。

图13-1 SPI 应用框图



SPI 工作时，通过片选信号 SS 来选择相应的从设备，进行同步串行数据传输。SPI 按照数据高位在前，低位在后的方式传输。

13.3.2 内部工作原理

13.3.2.1 发送数据

SPI 使能后，SPI 控制器会自动将发送 FIFO 中的数据发送出去，操作流程如下：

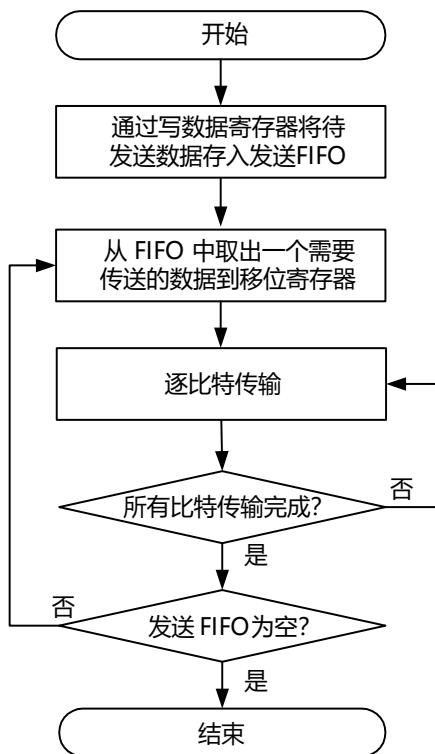
- 步骤 1 将待发送的数据从 SPI 数据寄存器 SPI_DR0~15 取出放在发送 FIFO 中。
- 步骤 2 将待发送数据的第一个数据从发送 FIFO 中取出，放到移位寄存器中。
- 步骤 3 由串行时钟控制，通过发送数据线 SPI0DO 逐比特发送 FIFO 中的第一个数据，高位在前，低位在后。
- 步骤 4 把第二个数据从发送 FIFO 中取出放到移位寄存器中发送，按照步骤 3 操作，直到发送 FIFO 为空。

----结束



SPI 发送数据的工作流程如图 13-2 所示。

图13-2 SPI 发送数据流程图



13.3.2.2 接收数据

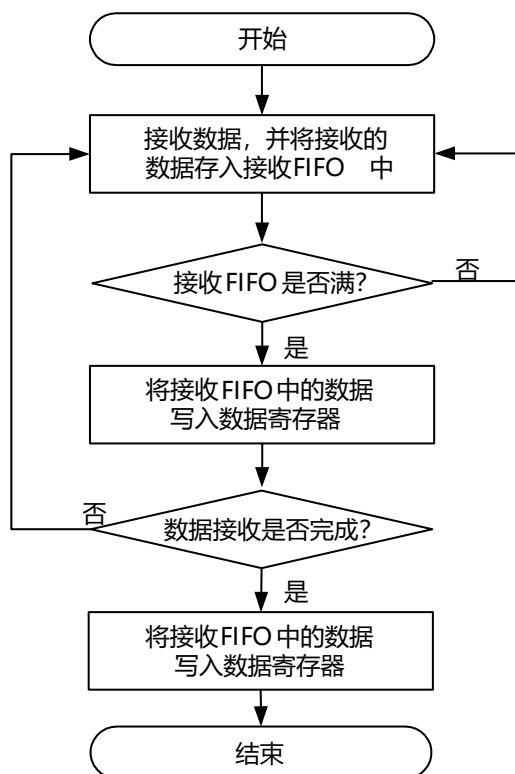
接收数据的操作流程如下：

- 步骤 1 通过接收数据线 SPI0DI 逐比特接收数据。
- 步骤 2 接收完一个数据之后存入到接收 FIFO，继续接收。
- 步骤 3 如果接收 FIFO 满，则将接收 FIFO 中的数据写入数据寄存器；如果接收 FIFO 未滿，则继续接收数据。
- 步骤 4 将接收 FIFO 中的数据写入 SPI 数据寄存器 SPI_DR0~15。

----结束

SPI 接收数据的工作流程如图 13-3 所示。

图13-3 SPI 接收数据流程图



须知

软件需参考寄存器 CTRL0 bit[9:8] tmod 配置与对接器件适配的工作模式。如寄存器描述：

- 发送与接收模式：发送 FIFO 和接收 FIFO 同时有效。
- 发送模式：该模式下接收 FIFO 中的数据无效，不会被存在接收 FIFO 中。在此模式下接收逻辑的相应中断应该被屏蔽。按照 [13.3.2.1 发送数据](#) 步骤工作。
- 接受模式：该模式下发送 FIFO 中的数据。在此模式下发送逻辑的相应中断应该被屏蔽。按照 [13.3.2.2 接收数据](#) 步骤工作。
- EEPROM 读：在该模式下参考 [13.4.6.2 数据接收](#) 步骤。

13.3.3 SPI 典型时序配置

图 13-4 中，spo 表示 SPICLKOUT 极性，sph 表示 SPICLKOUT 相位，spo、sph 不同取值由 MotorolaSPI 帧格式内部字段控制。



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

图13-4 MotorolaSPI 单帧帧格式 (spo=0、sph=0)

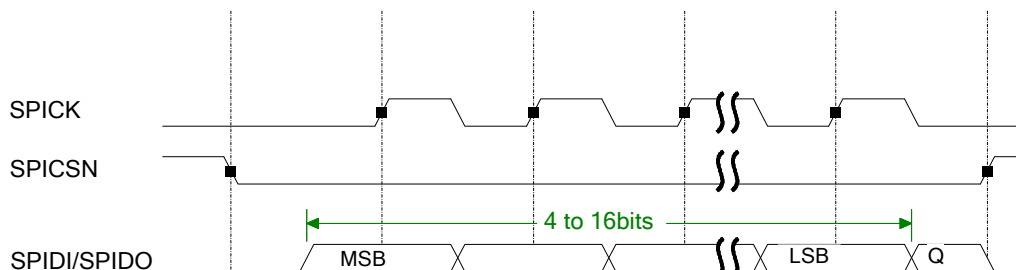
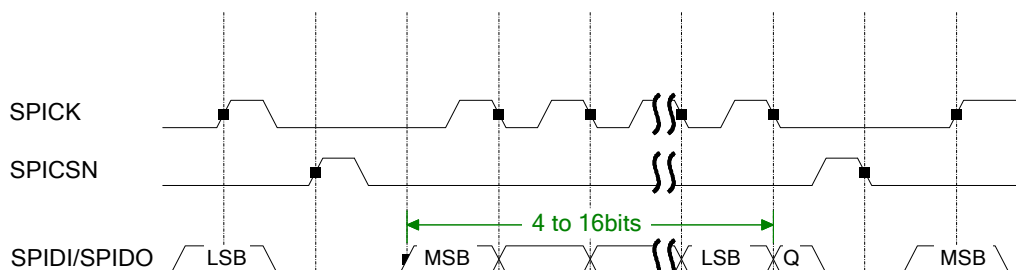


图13-5 MotorolaSPI 连续帧帧格式 (spo=0、sph=0)



在该模式下，当 SPI 处于空闲状态时：

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

当 SPI 处于使能状态，而且发送 FIFO 内部存在有效数据时，设置 SPICSNx 信号为低。半个 SPICK 时钟周期之后，有效的 Master 数据传输到 SPIDO 管脚。SPICK 时钟管脚会在接下来的半个 SPICK 时钟周期之后变化为高电平。数据就会在 SPICK 时钟的上升沿被捕获，在时钟的下降沿被传送。

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

如果是连续的传输，SPICSN 信号在 2 个 Word 传输之间必须将 SPICK 时钟拉高一个时钟周期。这是因为 sph 为 0 时，Salve 选择管脚会固定其内部串行设备寄存器的数据，使它不会变化。因此在连续传输时，主设备必须在每 2 个 Word 传输之间将

SPICSN 信号拉高。连续传输结束时，SPICSN 会在捕获到最后 1bit 数据之后的 1 个 SPICK 时钟周期之后恢复为高电平。

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

图13-6 MotorolaSPI 单帧帧格式 (spo=0、sph=1)

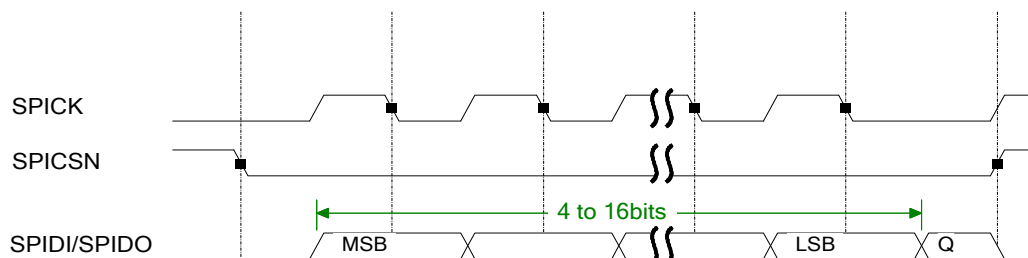
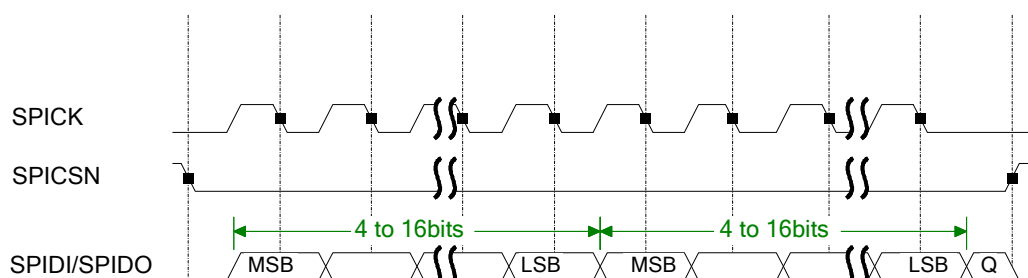


图13-7 MotorolaSPI 连续帧帧格式 (spo=0、sph=1)



在该模式下，当 SPI 处于空闲状态时：

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

当 SPI 处于使能状态，而且发送 FIFO 内不存在有效数据时，设置 SPICSNx 信号为低，表示开始传输数据。半个 SPICK 时钟周期之后，有效的 Master 数据传输到 SPIDO 管脚。SPICK 时钟管脚变化为低电平。数据就会在 SPICK 时钟的下降沿被捕获，在时钟的上升沿被传送。

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

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

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

图13-8 MotorolaSPI 单帧帧格式 (spo=1、sph=0)

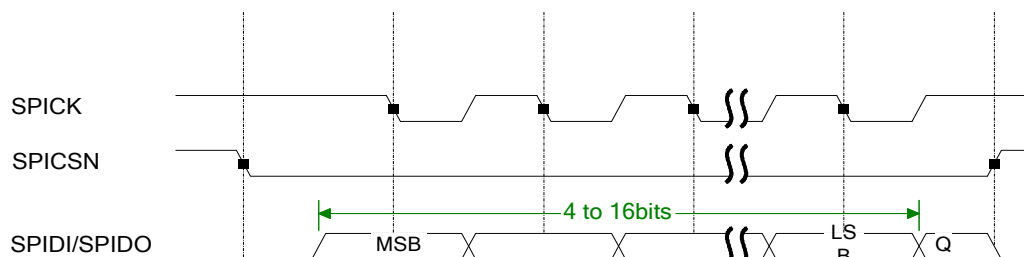
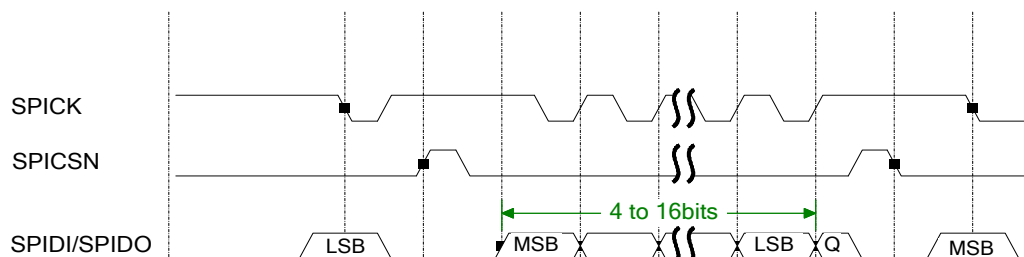


图13-9 MotorolaSPI 连续帧帧格式 (spo=1、sph=0)



在该模式下，当 SPI 处于空闲状态时：

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

当 SPI 为使能状态，而且在发送 FIFO 内部有有效数据时，设置 SPICSNx 信号为低表示开始传输数据。半个 SPICK 时钟周期之后，有效的 Master 数据传输到 SPIDO 管脚。SPICK 时钟管脚会在接下来的半个 SPICK 时钟周期之后变化为低电平。数据就会在 SPICK 时钟的下降沿被捕获，在时钟的上升沿被传送。

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

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

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

图13-10 MotorolaSPI 单帧帧格式 (spo=1、sph=1)

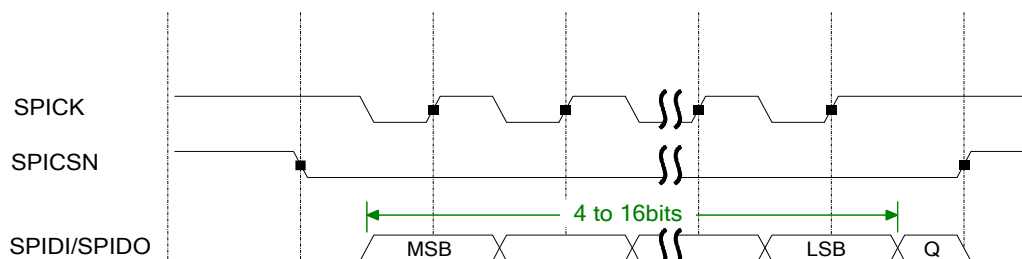
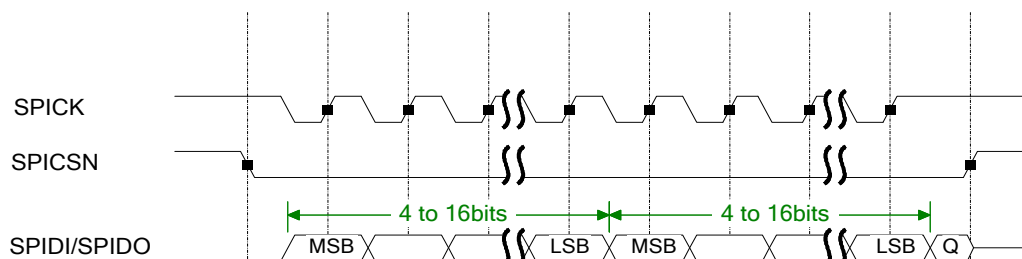


图13-11 MotorolaSPI 连续帧帧格式 (spo=1、sph=1)



在该模式下，当 SPI 处于空闲状态时：

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

当 SPI 为使能状态，而且在发送 FIFO 内部有效数据时，设置 SPICSNx 信号为低表示开始传输数据。半个 SPICK 时钟之后，有效的 Master 数据传输到 SPIDO 管脚。同时，时钟 SPICK 从一个下降沿开始有效。数据是在 SPICK 时钟的下降沿被捕获，在时钟的上升沿被传送。

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

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

13.3.3.5 TI 同步串行帧格式

图13-12 TI 同步串行单帧帧格式

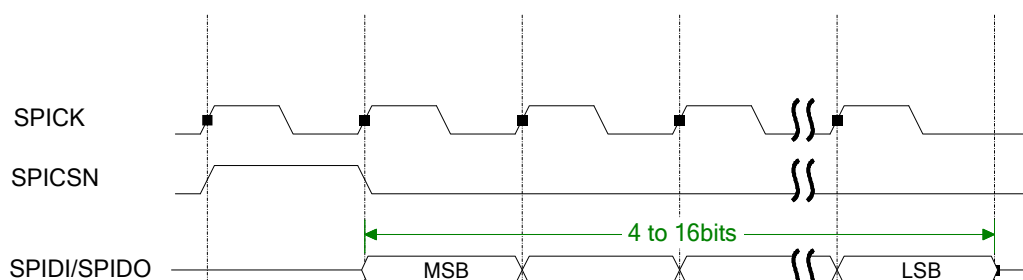
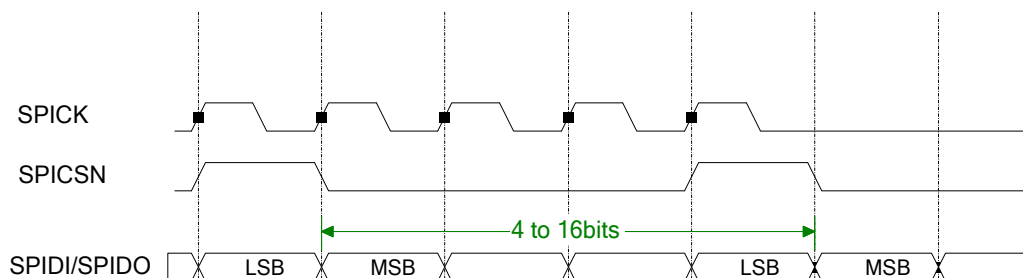


图13-13 TI 同步串行连续帧帧格式



在该模式下，当 SPI 处于空闲状态时：

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

一旦发送 FIFO 有数据，SPICSNx 就会产生一个 SPICK 时钟周期的高电平脉冲，将被发送的数据就会从发送 FIFO 传送到发送逻辑串行移位寄存器。在 SPICK 时钟的下一



个上升沿，4bit ~ 16bit 数据帧的 MSB 就会从 SPIDO 移位输出。同样，从外补串行 Slave 设准接收数据的 MSB 会从 SPIDI 管脚移位输入。

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

13.4 工作方式

13.4.1 管脚复用配置

SPI 的管脚其他管脚复用。使用 SPI 前，需要配置对应的寄存器，即将管脚设置为 SPI 模式。具体请参见《Hi3093 寄存器手册》。

说明

- 若要配置 3 线 SPI 模式，须在配置完管脚复用的基础上继续配置 LP_SUBSYS_CSR 的 LPSC_SPI_CTRL bit[3]（物理地址为 0x0871D01C）为 1，即可打开 3 线 SPI 模式。
- 若要配置片选的有效极性为高，可以对 LP_SUBSYS_CSR 的 LPSC_SPI_CTRL bit[2]（物理地址为 0x0871D01C）为 1，默认为低电平有效。

13.4.2 时钟门控

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

13.4.2.1 打开门控时钟

具体配置步骤如下：

步骤 1 配置 LB_SUBSYS 中的控制寄存器 LPSC_CLKEN_CTRL0 bit[8]为 1，打开 SPI0 模块的时钟；相应的配置 LPSC_CLKEN_CTRL0 bit[9]为 1，打开 SPI1 模块的时钟。

----结束

13.4.2.2 关闭门控时钟

具体配置步骤如下：

步骤 1 配置 LB_SUBSYS 中的控制寄存器 LPSC_CLKDIS_CTRL0 bit[8]为 1，关闭 SPI0 模块的时钟，相应的配置 LPSC_CLKDIS_CTRL0 bit[9]为 1，关闭 SPI1 模块的时钟。



-----结束

13.4.3 时钟配置

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

当前波特率=内部总线时钟频率/分频系数。

分频系数必须为偶数；内部总线时钟频率为 200MHz，同时也是 SPI 控制器内部工作时钟。

13.4.4 软复位

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

13.4.4.1 SPI0 软复位流程

步骤 1 通过配置 LP_SUBSYS 系统控制器 LPSC_SRST_REQ0 bit[7]为 1，可实现 SPI0 工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置 LP_SUBSYS 系统控制器 LPSC_RESET_DREQ0 bit[7]为 1，可实现对 SPI0 工作逻辑的单独软复位撤离。

----结束

13.4.4.2 SPI1 软复位流程

步骤 1 通过配置 LP_SUBSYS 系统控制器 LPSC_SRST_REQ0 bit[8]为 1，可实现 SPI1 工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置 LP_SUBSYS 系统控制器 LPSC_RESET_DREQ0 bit[8]为 1，可实现 SPI1 工作逻辑的单独接触软复位。

说明

复位后仅接收发送部分逻辑复位，各寄存器的值均保持不变。



13.4.5 初始化

初始化配置步骤如下，以 SPI0 为例，SPI1 相同：

步骤 1 向 SPI_SSIENR bit[0]写 0，关闭 SPI。

步骤 2 配置寄存器 SPI_CTRLR0。根据对接的从设备的时序来配置时钟极性和相位等参数。

须知

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

步骤 3 如果主设备需要从从设备接收数据则需要配置寄存器 SPI_CTRLR1，以确定接收数据的个数。

步骤 4 配置寄存器 SPI_BAUDR，设置传输的波特率。

步骤 5 配置发送 FIFO 阈值寄存器 SPI_TXFTLR 和接收 FIFO 阈值寄存器 SPI_RXFTLR，设置发送和接收 FIFO 的阈值水准。

步骤 6 配置寄存器 SPI_IMR，屏蔽不需要的中断。

步骤 7 配置寄存器 SPI_SER，来选择对接的从设备。

步骤 8 向 SPI_SSIENR bit[0]写 1，使能 SPI。

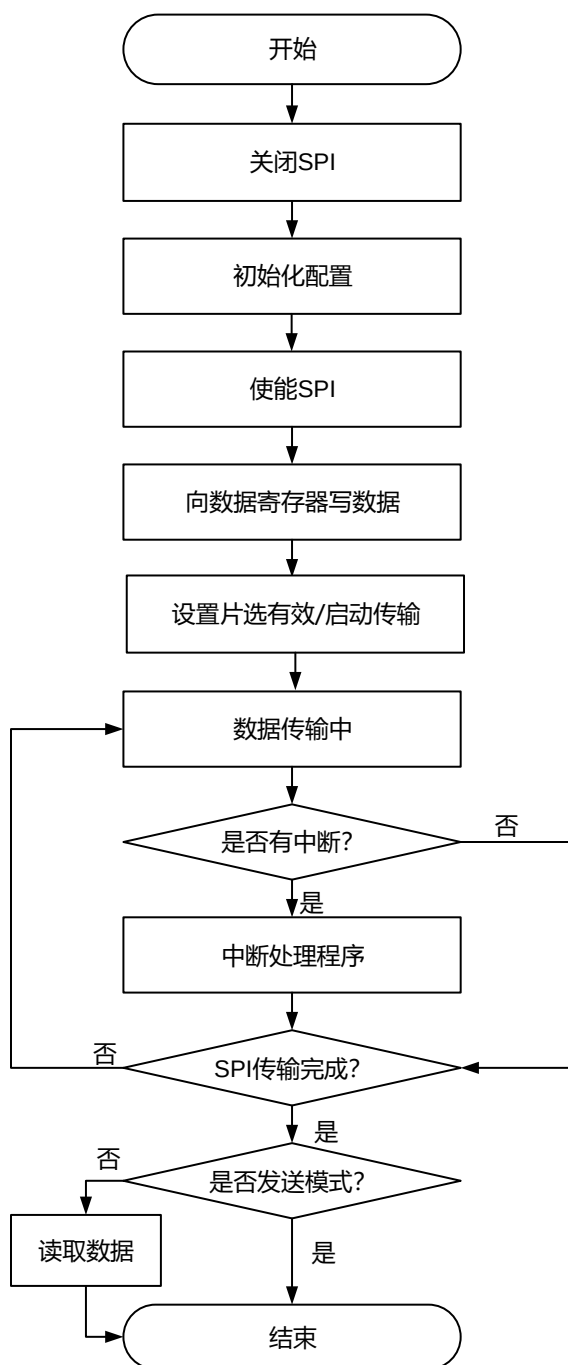
----结束

13.4.6 中断或查询方式下的数据传输

主设备模式下的数据传输流程图如[图 13-14](#) 所示。



图13-14 数据传输流程图



13.4.6.1 数据发送

具体配置步骤如下，以 SPI0 为例，SPI1 相同：



- 步骤 1 向 SPI 数据寄存器 SPI_DR0~15 (物理地址为 0x08713060~0x0871309c) 中写需要传输的数据。
- 步骤 2 配置 SPI_SER (物理地址为 0x08713010) 寄存器, 选中对接的从设备启动传输, 启动数据传输。
- 步骤 3 当产生发送 FIFO 空中断时, 如果还有需要传输的数据, 则需要向数据寄存器中写数据, 如果没有, 则传输完成。

----结束

13.4.6.2 数据接收

数据接收步骤如下, 以 SPI0 为例, SPI1 相同:

- 步骤 1 向 SPI 数据寄存器 SPI_DR0~15 (物理地址为 0x08713060~0x0871309c) 中写需要获取数据的地址和读命令。
- 步骤 2 配置 SPI_SER (物理地址为 0x08713010) 寄存器, 选中对接的从设备启动传输, 启动 SPI 数据传输。
- 步骤 3 当有接收 FIFO 满中断产生时, 则需要去读数据寄存器, 直到接收 FIFO 为空。

----结束

13.4.7 DMA 方式下的数据传输

请参见 [30.5.6 SPI 在 DMA 方式下的数据传输](#) 章节。



14 eMMC

14.1 概述

支持一个 eMMC 控制器，用于连接 eMMC 卡。

eMMC 控制器（以下简称 eMMC）用于处理对 eMMC 卡的读写等操作，eMMC 支持符合以下协议的设备：

- Embedded Multimedia Cards (eMMC - version 4.5)。

14.2 特性

- 接口时钟频率可编配置。
- 包含数据发送方向与接收方向的 2 个 FIFO，FIFO 深度为 256 Word。
- 支持 FIFO 阈值可配，DMA 传输时 Burst 大小可配。
- 支持 FIFO 上溢出与下溢出中断告警，防止数据传输错误。
- 支持命令与数据的 CRC 生成与校验。
- 支持低功耗模式关闭 eMMC 时钟和接口时钟。
- 支持 1Byte ~ 65535Byte 的块数据读写操作。
- 支持 eMMC 卡流数据读写方式。
- 支持命令完成中断、超时中断。
- 支持对 response 自动检测。
- 支持从 eMMC 进行 boot。
- 容量不低于 8GB。
- 数据位宽支持 1bit、4bit、8bit。



- 支持 eMMC4.5 版本。
- eMMC 控制器支持 1bit 和 2bit 的 ECC 错误注入。

14.3 关键术语

表14-1 eMMC 关键术语描述

术语	描述
eMMC	Embedded Multimedia Cards
CRC	Cyclical Redundancy Check, 循环冗余码

14.4 功能

14.4.1 功能原理

14.4.1.1 指令与响应

eMMC 与卡设备之间的所有交互操作均通过指令完成，包括卡初始化、寄存器读写、状态查询、数据传输等。

eMMC 指令为 48bit 的串行数据，由起始位、传输位、指令序号、指令参数、CRC 校验位和终止位组成。卡收到指令后，会根据指令类型返回 48bit 或 136bit 的响应。

图14-1 eMMC 指令格式

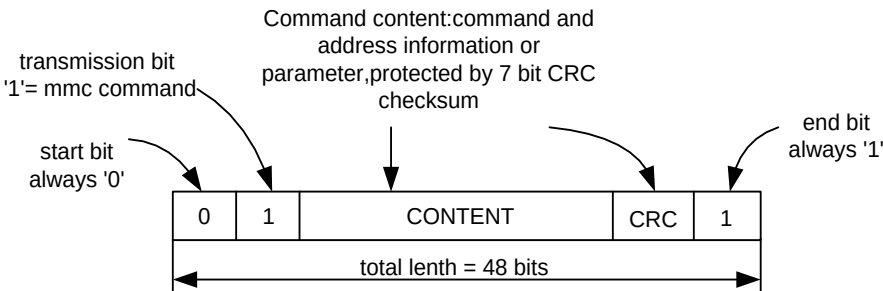
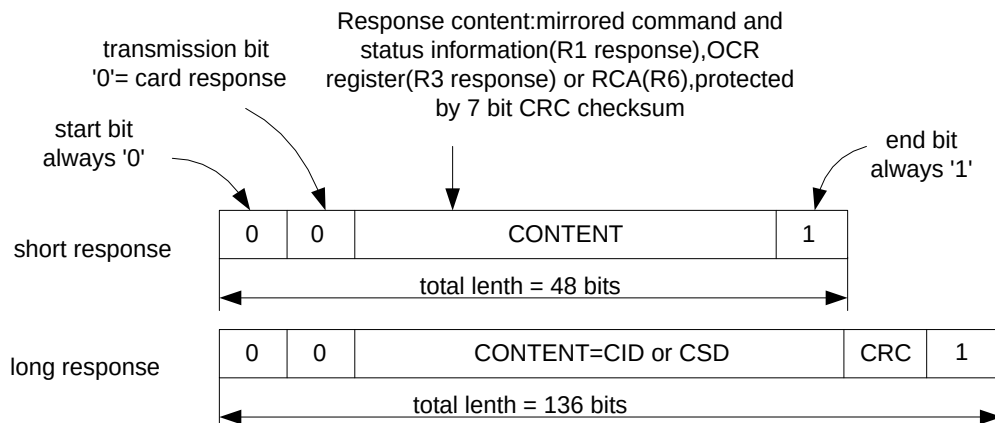


图14-2 eMMC 指令响应格式



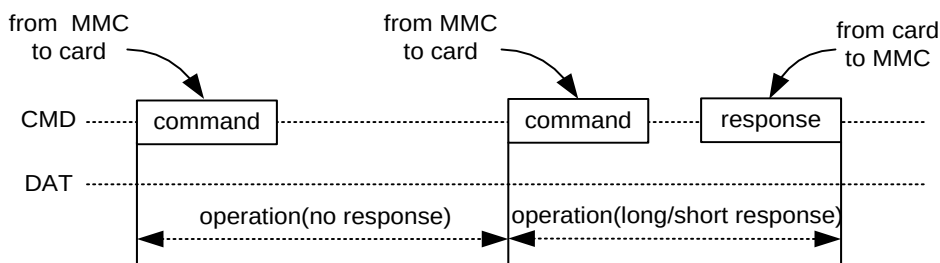
根据是否有数据传输，指令分为以下 2 种：

- 非数据传输指令

基于指令信号线 CMD，MMC 与卡采用串行方式进行指令发送与响应接收。

eMMC 与卡设备之间的非数据传输指令操作如图 14-3 所示。

图14-3 eMMC 非数据指令操作



- 数据传输指令

除指令线上的交互外，还有数据线 DAT0 ~ DAT7 上的数据传输。

eMMC 卡支持以下数据传输指令：

- 流数据读写指令，使用 1 根数据线（即 DAT0）进行数据传输，无 CRC 校验。
- 单块数据读写指令，一次传输完成一个块大小的数据，不需要使用停止命令结束一次数据传输。
- 多块数据读写指令。

多块数据读写有以下两种方式：

- predefinedblockcount 方式
在多块读写指令前，发送块数量指令指定待传输的数据量。
- openended 方式
发送读写指令后，在数据传输末尾，需使用停止指令来结束一次数据传输。

两种方式的差别在于 eMMC 通知卡结束一次传输的方式不同。

根据响应的类型，指令分为以下 3 种：

- 无响应指令：如卡复位指令。
- 短响应指令：数据传输指令、卡状态查询等均属于这类指令。
- 长响应指令：仅用于读取卡的寄存器 CID 和 CSD 信息。

14.4.1.2 数据传输

单块读写指令和多块读写指令为较常用的数据传输方式。通常 eMMC 卡数据传输的一个块大小为 512Byte。

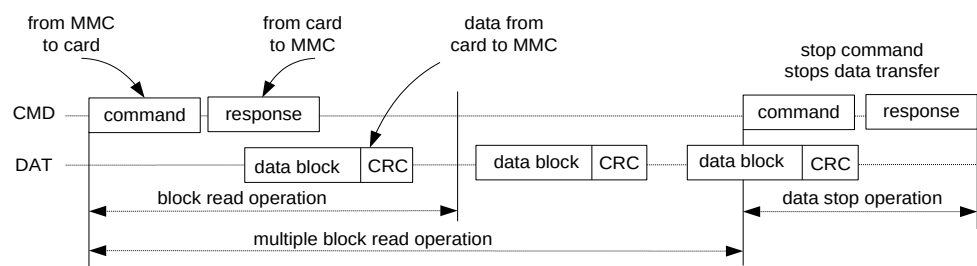
说明

以块读写指令方式进行数据传输时，传输数据总量必须为块大小的整数倍。

数据传输指令均为短响应指令，并伴随着数据线上的数据传输。指令、响应及数据线上的时序配合关系如图 14-4 和图 14-5 所示。

- 单块与多块读操作

图14-4 单块与多块读操作

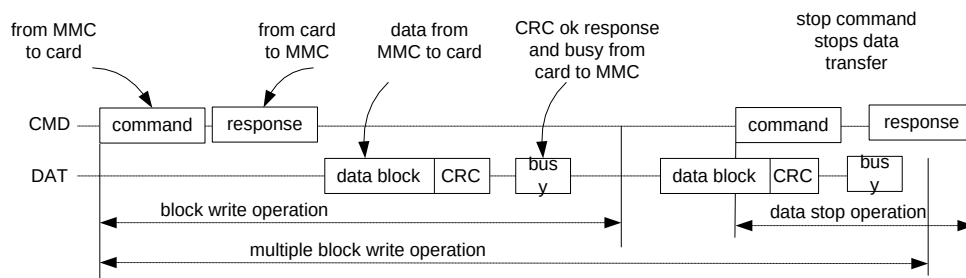


eMMC 向卡发送单块或多块读指令。在接收响应的过程中，接收以块为单位的数据，其中每块数据中均包含有 CRC 校验位，以保证数据传输的完整性。

单块读指令操作时，eMMC 在接收一块数据后完成一次数据传输；多块读指令操作时，eMMC 在接收多块数据后，需发送一条停止指令结束本次数据传输（仅 openended 多块读指令）。

- 单块与多块写操作

图14-5 单块与多块写操作



eMMC 往卡发送单块或多块写指令。在接收到响应后，往卡发送以块为单位的数据，其中每块数据中均包含有 CRC 校验位，卡会对每块数据进行 CRC 校验，并反馈 CRC 状态以确认数据传输的正确性。

单块写指令操作时，eMMC 在发送一块数据后完成一次数据传输；多块写指令操作时，eMMC 在发送多块数据后，需发送一条停止指令完成本次数据传输（仅 openended 多块读指令）。写操作结束后，卡可能会因为编程 Flash 而处于繁忙状态，eMMC 需查询 DAT0 状态，以确认卡退出繁忙状态后才能对卡进行下一步操作。

- 数据传输格式

块方式读写中，eMMC 与卡之间可采用 1bit、4bit、8bit 数据线方式进行数据传输。在发送数据传输指令之前，应分别设置 eMMC 与卡的数据传输位宽模式（1bit、4bit、8bit），使它们保持一致。eMMC 的数据位宽通过寄存器 CTYPE 设置，卡的数据位宽则通过发送相应的指令进行设置。

1bit、4bit、8bit 模式下的数据传输格式如[图 14-6](#)、[图 14-7](#) 和[图 14-8](#) 所示。



图14-6 1bit 数据线传输模式下的块数据格式

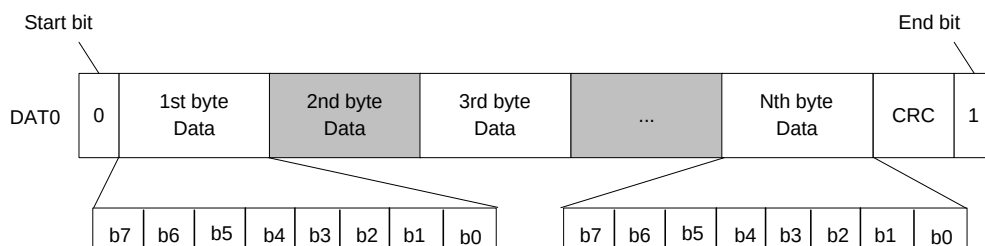


图14-7 4bit 数据线传输模式下的块数据格式

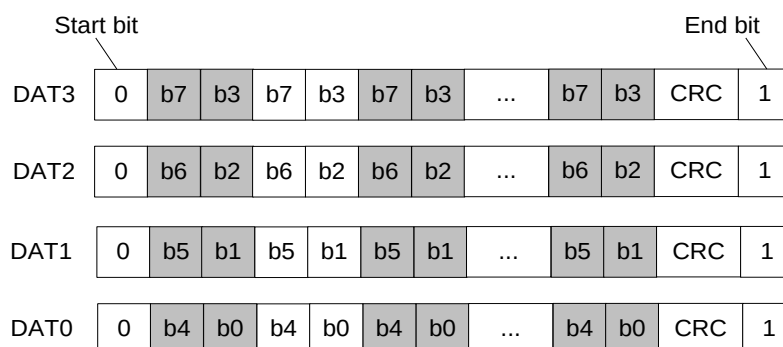
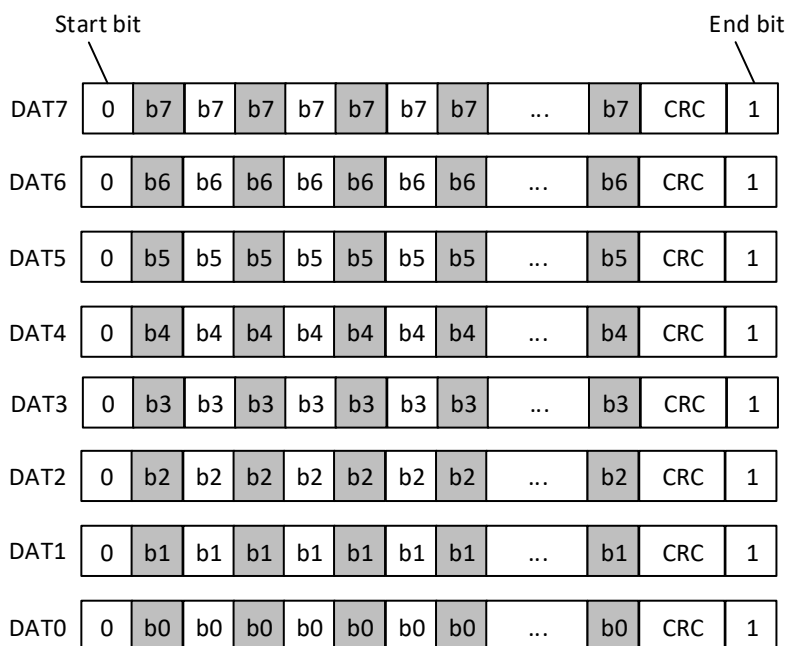


图14-8 8bit 数据线传输模式下的块数据格式





14.4.1.3 eMMC 启动 (boot 模式)

Boot Area 包含两个 Boot Area Partitions，主要用于存储 Bootloader，支持从 eMMC 启动系统。

两个 Boot Area Partitions 的大小是完全一致的，由 Extended CSD register 的 BOOT_SIZE_MULT Field 决定，计算公式如下：

$$\text{Size} = 128\text{Kbytes} \times \text{BOOT_SIZE_MULT}$$

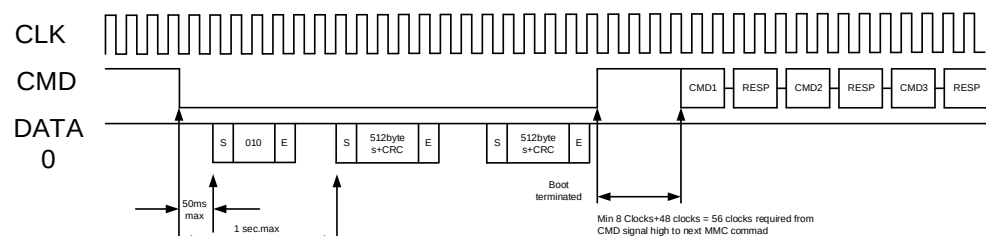
一般情况下，Boot Area Partition 的大小都为 4 MB，即 BOOT_SIZE_MULT 为 32。BOOT_SIZE_MULT 最大可以为 255，即 Boot Area Partition 的最大容量大小可以为 $255 \times 128 \text{ KB} = 32640 \text{ KB} = 31.875 \text{ MB}$ 。

eMMC 中定义了 Boot State，在 Power-up、HW reset 或者 SW reset 后，如果满足一定的条件，eMMC 就会进入该 State。进入 Boot State 的条件如下：

- Original Boot Operation

CMD 信号保持低电平不少于 74 个时钟周期，会触发 Original Boot Operation，进入 Boot State。

图14-9 Original Boot Operation 时序图

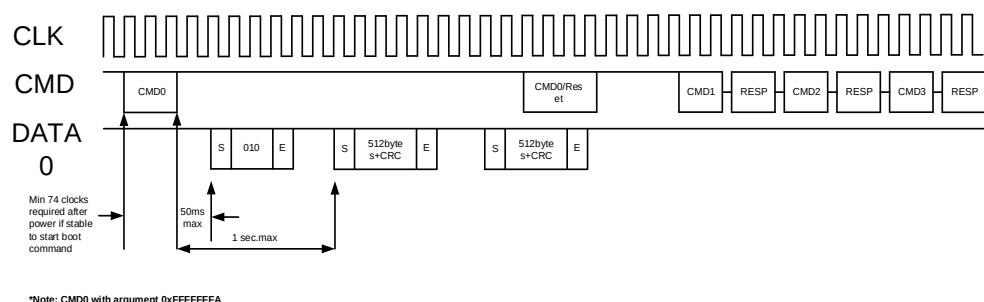


- Alternative Boot Operation

在 74 个时钟周期后，在 CMD 信号首次拉低或者 Host 发送 CMD1 之前，Host 发送参数为 0xFFFFFFFF 的 COM0 时，会触发 Alternative Boot Operation，进入 Boot State。



图14-10 Alternative Boot Operation



在 Boot State 下，如果有配置 BOOT_ACK，eMMC 会先发送“010”的 ACK 包，接着 eMMC 会将最大为 28KB x BOOT_SIZE_MULT 的 Boot Data 发送给 Host。传输过程中，Host 可以通过拉高 CMD 信号 (Original Boot 中)，或者发送 Reset 命令 (Alternative Boot 中) 来中断 eMMC 的数据发送，完成 Boot Data 传输。

14.5 工作方式

14.5.1 工作时钟门控

当 eMMC 控制器处于空闲状态时，可以关闭其工作时钟。具体步骤如下：

- 步骤 1 读寄存器 STATUS (物理地址为 0x08630048)。
- 步骤 2 若 STATUS[cmd_fsm_states]和 STATUS[data_state_mc_busy]均为 0，则向 CTRL (物理地址为 0x08630000) 写“0”，屏蔽 eMMC 中断、DMA 请求使能等，进入[步骤 3](#)。若其中有一个为非 0，则延时等待，返回[步骤 1](#)。
- 步骤 3 向 LP_SUBSYS 控制寄存器 LPSC_EMMC_CRG_CTRL (物理地址 0x0871d0f8) 对应位写“1”，打开时钟。
- 步骤 4 如果需要关闭工作时钟，向 LP_SUBSYS 控制寄存器 LPSC_EMMC_CRG_CTRL (物理地址 0x0871d0f8) 对应位写“0”。

----结束



14.5.2 软复位

在数据传输出现异常而导致 eMMC 无法回到空闲状态时，配置 LP_SUBSYS 寄存器对 eMMC 进行软复位：

步骤 1 通过配置 LP_SUBSYS 系统控制器 LPSC_EMMC_CRG_CTRL bit[1]（物理地址 0x0871d0f8）为 1，可实现对 eMMC 控制器工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置 LP_SUBSYS 系统控制器 LPSC_EMMC_CRG_CTRL bit[1]（物理地址 0x0871d0f8）为 0，可实现对 eMMC 控制器工作逻辑的单独软复位撤离。

----结束

说明

在使用 eMMC 控制器之前软复位控制器。

14.5.3 时钟配置

14.5.3.1 接口时钟配置

在使用 eMMC 前，需为其配置合适的工作时钟频率。

eMMC 的卡侧时钟可通过 LP_SUBSYS 配置分频，采样和驱动时钟的相位也可进行控制。

根据 eMMC 速率模式选择合适的分频系数，eMMC 控制器支持以下速率模式。

表14-2 eMMC 的工作模式频率要求

工作模式	频率
HS400	0~100MHz
HS200	0~100MHz
High Speed DDR	0~52MHz
High Speed SDR	0~52MHz
Backwards Compatibility	0~26MHz

eMMC 控制器的卡侧工作时钟 cclk_in 是由 LP_SUBSYS 的 TOP_CRG 输入，通过配置分频系数寄存器 emmc_clk_div_cnt（LPSC_CRG_DIV_CTRL bit[7:4]物理地址为



0x0871d018) 来控制 eMMC 时钟频率, eMMC 的 CRG 会对分频后时钟统一四分频送给控制器, 注意每种工作模式的频率上限。如下:

$$F_{CCLK_IN} = F_{LP_CLK} / (4 * emmc_clk_div_cnt)$$

在 eMMC 的使用过程中可以对卡侧驱动和采样时钟进行调相以满足 Timing 需求, 配置 LP_SUBSYS 控制器 LPSC_EMMC_FUNC_CTRL0 bit[6:4]对卡侧采样时钟进行调相, 配置 LPSC_EMMC_FUNC_CTRL0 bit[3:1]对卡侧驱动时钟进行调相。

在改变 eMMC 卡的时钟频率之前, 必须保证没有数据或指令正在传输。为了避免输出到 eMMC 卡的时钟产生毛刺, 在改变 eMMC 卡的时钟频率时应该遵照以下步骤:

- 步骤 1 关闭接口时钟。将寄存器 CLKENA (物理地址为 0x08630010) 配置为 0x0000_0000, 并将寄存器 CMD (物理地址为 0x0863002c) [start_cmd]、CMD[update_clock_registes_only]和 CMD[wait_prvdata_complete]置 “1”, 等待寄存器 CMD[start_cmd]被自动清零。
- 步骤 2 设置分频因子。根据所需要的时钟频率设置寄存器 CLKDIV (物理地址为 0x08630008), 并将寄存器 CMD (物理地址为 0x0863002c) [start_cmd]和 CMD[update_clock_registes_only]置 “1”, 等待寄存器 CMD[start_cmd]被自动清零。
- 步骤 3 重新使能接口时钟。
- 步骤 4 将寄存器 CLKENA (物理地址为 0x08630010) 配置为 0x0000_0001, 并将寄存器 CMD (物理地址为 0x0863002c) [start_cmd]和 CMD[update_clock_registes_only]置 “1”, 等待寄存器 CMD[start_cmd]被自动清零。

----结束

须知

- 只有当寄存器 CMD[start_cmd]和 CMD[update_clk_only]置 “1” 时, 寄存器 CLKDIV、CLKENA 的值才会被载入。当载入成功以后, eMMC 会自动清零寄存器 CMD[start_cmd]。如果此时有其他指令正在执行, 则会产生 HLE (HardwareLockedError) 中断。若产生 HLE 中断, 清除中断后重新发命令即可。
- 当有指令执行和数据传输时, 不能改变卡的时钟参数。

14.5.4 初始化

与卡进行命令和数据的交互前, 需要初始化 eMMC, 步骤如下:



- 步骤 1 配置 eMMC 工作时钟频率。
- 步骤 2 当卡上电，指令和数据信号线上拉稳定后，解复位控制器（跟随芯片同步复位撤离）。
- 步骤 3 清中断。将寄存器 RINTSTS (物理地址为 0x08630044) bit[15:0]所有位置“1”，清除原始中断状态位。
- 步骤 4 设置寄存器 INTMASK (物理地址为 0x08630024)。将寄存器 INTMASK bit[15:0]所有位置“1”，使能各中断源。若使用 DMA 方式进行数据传输，应将 INTMASK bit[4]、INTMASK bit[5]置“0”，以屏蔽接收/发送 FIFO 数据请求中断。
- 步骤 5 将寄存器 CTRL (物理地址为 0x08630000) [int_enable]置“1”，使能 eMMC 中断。
- 步骤 6 若使用 DMA 方式进行数据传输，应将寄存器 CTRL[use_internal_dmac]置“1”，使能 DMA 请求功能。
- 步骤 7 配置超时参数寄存器 TMOUT (物理地址为 0x08630014)。
- 步骤 8 配置 FIFO 参数寄存器 FIFOTH (物理地址为 0x0863004c)。
- 步骤 9 配置卡门限控制寄存器 CARDTHRCTL (物理地址为 0x08630100)，读卡门限配置为 512。如果要使用读门限，则 card_rd_thr_en 需要配置为 1。这个步骤仅 eMMC 控制器需要配置。

----结束

完成以上步骤后，就可以配置接口时钟，往卡发送指令了。

14.5.5 非数据传输指令

eMMC 在指令发送后，一旦收到响应（无论对错或超时），就会将寄存器 RINTSTS bit[2]置位。短响应保存到寄存器 RESP0 中，长响应保存到寄存器 RESP0 ~ RESP3 中，寄存器 RESP3 bit[31]为最高位，RESP0 bit[0]为最低位。当指令发出以后，其错误是由指令响应以及寄存器 RINTSTS 的错误位反映。发送非数据传输指令的步骤如下：

- 步骤 1 在寄存器 CMDARG (物理地址为 0x08630028)中设置相应的指令参数。
- 步骤 2 根据表 14-3 设置指令寄存器 CMD (物理地址为 0x0863002c)。
- 步骤 3 等待指令被 eMMC 执行。如果指令已执行，eMMC 自动清零寄存器 CMD[start_cmd]。



- 步骤 4 检查寄存器 RINTSTS bit[12] (物理地址为 0x08630044)是否产生 HLE 中断。
- 步骤 5 等待指令执行完毕。eMMC 收到响应 (无论对错或超时) 时将寄存器 RINTSTS bit[2] 置 “1” , 表示 CommandDone。
- 步骤 6 检查是否有响应异常, 必要时可读取响应值。
- 步骤 7 读取寄存器 RINTSTS bit[8]、RINTSTS bit[6]和 RINTSTS bit[1], 检查响应超时、响应 CRC 错误和响应错误。

---结束

须知

- 只有当寄存器 CMD[start_cmd]置 “1”、CMD[update_clock_registes_only]置 “0” 时, 寄存器 BYTCNT、BLKSIZ、CMDARG 和 CMD 的值才能被载入。载入成功后, eMMC 自动清零 CMD[start_cmd]。
- 如果有其他指令正在执行, 会产生 HLE 中断, 此时重新执行操作即可。在非数据传输指令执行时, 寄存器 BYTCNT 和 BLKSIZ 的值被忽略。

表14-3 非数据传输指令时的寄存器 CMD 配置参考 (默认值)

参数	取值	描述
start_cmd	1	启动指令发送。
update_clock_registes_only	0	非时钟参数更新指令。
data_transfer_expected	0	非数据传输指令。
card_number	0	-
cmd_index	cmdindex	指令序号。
send_initialization	0	当指令为卡复位时置 “1” , 如指令 CMD0。
stop_abort_cmd	0	当指令为停止数据传输时置 “1” , 如指令 CMD12。
rsponse_length	0	当响应为长响应类型时置 “1” 。



参数	取值	描述
rsponse_expect	1	当指令无响应时置“0”，如：指令 CMD0、指令 CMD4、指令 CMD15。
wait_prvdata_complete	1or0	在发送指令之前，eMMC 必须等待正在处理的数据传输指令结束。建议此位一直置“1”，除非该指令是为了在数据传输时查询卡状态或停止当前数据的传输。
check_response_crc	1or0	eMMC 是否检查响应的 CRC 校验位。

14.5.6 单块或多块读数据

读取单块或多块数据的步骤如下：

- 步骤 1 向寄存器 CTRL[fifo_reset] (物理地址为 0x08630000)写“1”，复位 FIFO 指针，查询等待该位自动清零。
- 步骤 2 向寄存器 BYTCNT (物理地址为 0x08630020) 写入待传输数据的字节数。
- 步骤 3 向寄存器 BLKSIZ (物理地址为 0x0863001c) 写入块的大小。
- 步骤 4 向寄存器 CMDARG (物理地址为 0x08630028) 写入读取数据的起始地址。
- 步骤 5 根据表 14-4 设置寄存器 CMD (物理地址为 0x0863002c)。

对于 eMMC 卡，分别使用指令 CMD17/CMD18 进行单块/多块读操作。

一旦寄存器 CMD 被写入，eMMC 就执行指令；当指令被送到总线上以后，产生 cmd_done 中断。

- 步骤 6 检查寄存器 RINTSTS bit[5]和 RINTSTS bit[10] (物理地址为 0x08630044)，如果其中之一为 1 或都为 1，则从寄存器 DATA 读取 FIFO 中的数据，以便 eMMC 接收后面的数据；同时检查数据错误中断，即寄存器 RINTSTS bit[7]、RINTSTS bit[9]、RINTSTS bit[13]和 RINTSTS bit[15]。此时，程序可以发送一个停止指令中止数据的传输。



步骤 7 当寄存器 RINTSTS bit[3]为 1 时，数据传输完成，从寄存器 DATA (物理地址为 0x08630200) 中读取残留在 FIFO 中的数据。

步骤 8 若执行指令时已将寄存器 CMD[send_auto_stop] (物理地址为 0x0863002c)置 “1” ， eMMC 自动发送停止指令结束一次数据传输，请参见 “[14.5.10 Auto-stop 使用配置](#)” 。

----结束

表14-4 单块或多块读数据时的寄存器 CMD 配置参考（默认值）

参数	取值	描述
start_cmd	1	启动指令发送。
update_clock_registes_only	0	非时钟参数更新指令。
card_number	0	-
send_initialization	0	当指令为卡复位时置 “1” ， 如指令 CMD0。
stop_abort_cmd	0	当指令为停止数据传输时置 “1” ， 如指令 CMD12。
send_auto_stop	0or1	请参见 “ 14.5.10 Auto-stop 使用配置 ” 中的 “Auto-stop 使用配置” 。
transfer_mode	0	块传输。
read/write	0	从卡中读取数据。
rsponse_length	0	数据指令均为短响应。
data_transfer_expected	1	数据传输指令。
rsponse_expect	1	当指令无响应时置 “0” ， 如：指令 CMD0、CMD4、CMD15。
cmd_index	Cmdindex	命令序号。
wait_prvdata_complete	1or0	在发送指令之前，主设备必须等待正在处理的数据传输指令结束。建议此位一直置 1，除非该指令是为了询问卡状态或停止当前数据的传输。



参数	取值	描述
check_response_crc	1or0	eMMC 是否检查响应的 CRC 校验位。

14.5.7 单块与多块写数据

写入单块或多块数据的步骤如下：

- 步骤 1 向寄存器 CTRL[fifo_reset] (物理地址为 0x08630000) 写 “1” ， 复位 FIFO 指针， 查询等待直至该位自动清零。
- 步骤 2 向寄存器 BYTCNT (物理地址为 0x08630020) 写入待传输数据的大小。
- 步骤 3 向寄存器 BLKSIZ (物理地址为 0x0863001c) 写入块的大小。
- 步骤 4 向寄存器 CMDARG (物理地址为 0x08630028) 写入数据的起始地址。
- 步骤 5 将数据写入 FIFO， 即写寄存器 DATA (物理地址为 0x08630200)， 通常在开始时应写满 FIFO。
- 步骤 6 根据表 14-5 设置寄存器 CMD (物理地址为 0x0863002c)。对于 eMMC 卡， 分别使用指令 CMD24/CMD25 进行单块/多块写操作；
- 步骤 7 检查寄存器 RINTSTS bit[4]和 RINTSTS bit[10] (物理地址为 0x08630044)， 如果其中之一为 1 或都为 1， 写寄存器 DATA 往 FIFO 填充数据； 同时应检查数据错误中断， 即检查寄存器 RINTSTS bit[7]、 RINTSTS bit[9]、 RINTSTS bit[13]和 RINTSTS bit[15]。如果有需要， 程序可以发送一个停止指令以中止数据的传输。当寄存器 RINTSTS bit[3]为 1， 数据传输结束。
- 步骤 8 若执行指令时已将寄存器 CMD[send_auto_stop] (物理地址为 0x0863002c)置 “1” ， eMMC 自动发送停止指令结束一次数据传输。请参见 “14.5.10 Auto-stop 使用配置” 。
- 步骤 9 查询并等待寄存器 STATUS[data_busy] (物理地址为 0x08630048)由 1 变为 0。

表14-5 单块或多块写数据时的寄存器 CMD 配置参考（默认值）

参数	取值	描述
start_cmd	1	启动指令发送。
update_clock_registes_only	0	非时钟参数更新指令。



参数	取值	描述
card_number	0	-
send_initialization	0	当指令为卡复位时置“1”，如指令 CMD0。
stop_abort_cmd	0	当指令为停止数据传输时置“1”，如指令 CMD12。
send_auto_stop	0or1	请参见 14.5.10 Auto-stop 使用配置 。
transfer_mode	0	块传输。
read_write	1	往卡写入数据。
rsponse_length	0	数据指令均为短响应。
data_transfer_expected	1	数据传输指令。
rsponse_expect	1	当指令无应时置“0”，如：指令 CMD0、CMD4、CMD15。
cmd_index	Cmdindex	-
wait_prvdata_complete	1or0	在发送指令之前，主设备必须等待直到正在处理的数据传输指令结束。建议此位一直置“1”，除非该指令是为了询问卡状态或停止当前数据的传输。
check_response_crc	1or0	eMMC 是否检查响应的 CRC 校验位。

14.5.8 流数据读写

流数据的读写方式，除了将寄存器 CMD[transfer_mode]置“1”外，其他与块数据的读写方式一致。对于流数据的传输，通常需要使用 Auto-stop 功能。

14.5.9 内置 DMA 方式数据传输

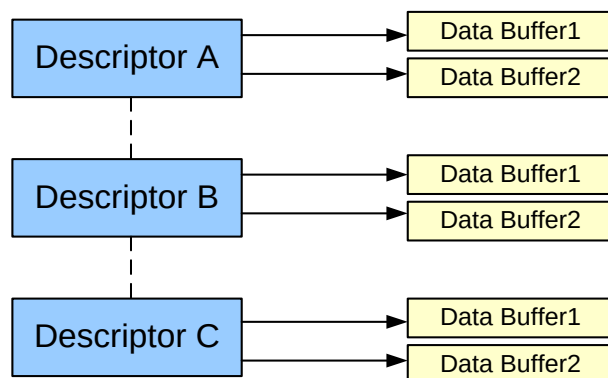
eMMC 控制器含有内置 DMA 控制器 (IDMAC)，IDMAC 可以根据指定的描述符把数据从原地址搬移到目的地址。

14.5.9.1 描述符

IDMAC 可以使用如下两种类型的描述符：

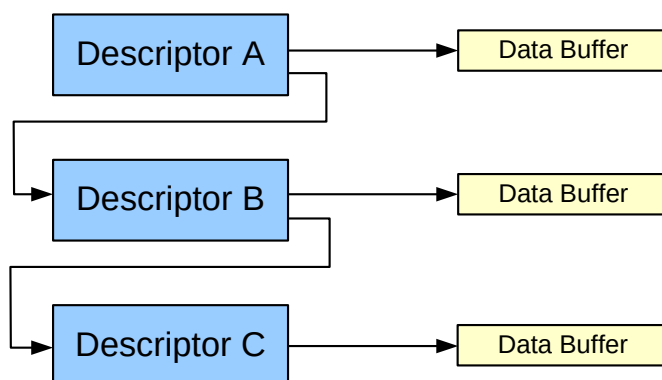
- 双 Buffer 结构：两个描述符之间的间隔根据 BMOD 寄存器的 DSL 位来指定。双 buffer 结构的示意图如图 14-11 所示。

图14-11 双 buffer 结构示意图



- 链结构：每个描述符指向唯一的 buffer 和下一个描述符。链结构的示意图如图 14-12 所示。

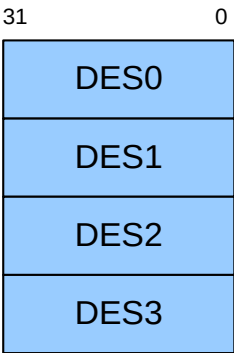
图14-12 链结构示意图



描述符必须以 Word 对齐，每个描述符包含 16Byte 的控制和状态信息。在链结构模式下每个描述符最多可传输 4KB 数据，在双 buffer 模式下每个描述符最多可传输 8KB 数据。描述符的内部结构如图 14-13 所示。



图14-13 32bit 位宽的描述符的结构



DES0 保护控制和状态信息，具体含义如表 14-6 所示。

表14-6 DES0 各 bit 的含义

位	名称	说明
31	OWN	描述符属性指示。 0：表示该描述符属于 CPU； 1：表示该描述符属于 IDMAC。 使用内部 DMA 完成数据传输后，IDMAC 将该位清 0。
30	CES	读写卡时出错的状态指示。 0：没有错； 1：出错。
29:6	RES	保留。
5	ER	描述符链路结束指示： 0：不是最后一个； 1：该链路最后一个描述符。 只对双 buffer 结构的描述符有意义。
4	CH	第二个地址（DES3 的地址）的含义： 0：DES3 中的第二个地址是指第二个 buffer 的地址； 1：DES3 中的第二个地址是指下一个描述符的地址。但该位为 1 时，DES1 bit[25:13]应该是 0。



位	名称	说明
3	FS	为 1 表示该描述符包含第一个数据 buffer，如果第 1 个数据 buffer 的大小为 0，那么下一个描述符包含起始数据。
2	LD	为 1 表示该描述符指向的 buffer 是最后一个数据 buffer。
1	DIC	为 1 时阻止数据传输结束上报中断。
0	RES	保留。

DES1 指示 buffer 大小，具体含义如表 14-7 所示。

表14-7 DES1 各 bit 的含义

位	名称	说明
31: 26	RES	保留。
25: 13	BS2	第二个数据 buffer 的 Byte 数，必须是 4 的整数倍。DES0 bit[4] 为 1 时该位无意义。
12: 0	BS1	第一个数据 buffer 的 Byte 数，必须是 4 的整数倍。

DES2 表示第一个数据 buffer 的地址指针，具体含义如表 14-8 所示。

表14-8 DES2 各 bit 的含义

位	名称	说明
31:0	BAP1	第一个数据 buffer 的物理地址，必须 Word 对齐。

DES3 表示第二个地址，具体含义如表 14-9 所示。



表14-9 DES3 各 bit 的含义

位	名称	说明
31:0	BAP2	当使用的是双 buffer 结构时，表示第二个数据 buffer 的物理地址。如果 DES0 bit[4]是 1，该地址表示下一个描述符的物理地址。

14.5.9.2 初始化

- 步骤 1 配置 BMOD (物理地址为 0x08630080) 寄存器，设置总线参数。
- 步骤 2 配置 IDINTEN (物理地址为 0x08630090) 寄存器，屏蔽不需要的寄存器。
- 步骤 3 创建发送或接收描述符链表，然后配置 DBADDR (物理地址为 0x08630088) 寄存器，设置起始地址。
- 步骤 4 IDMAC 尝试从描述符链表中获取描述符。

----结束

14.5.9.3 发送

- 步骤 1 CPU 创建描述符 (DES0~DES3) 并将 DES0 bit[31]位 OWN 置为 1，同时准备好数据 buffer。
- 步骤 2 把写数据命令写入 CMD (物理地址为 0x0863002c)寄存器。
- 步骤 3 通过 FIFOTH (物理地址为 0x0863004c) 寄存器设置 TX_Wmark。
- 步骤 4 IDMAC 获取描述符并判断 OWN 是否为 1，如果 OWN 不为 1 则需要等待 CPU 释放描述符，同时，IDMAC 会进入 suspend 状态，需要 CPU 配置 PLDMND (物理地址为 0x08630084)寄存器使 IDMAC 重新获取描述符。
- 步骤 5 OWN 为 1 时，IDMAC 从数据 buffer 中搬移数据到 eMMC 内部 FIFO。
- 步骤 6 如果中断已经使能，数据搬移完成后 IDMAC 状态寄存器 IDSTS (物理地址为 0x0863008c)的相应为将被更新，同时 OWN 位被清零。

----结束



14.5.9.4 接收

- 步骤 1 CPU 创建描述符 (DES0~DES3) 并将 DES0 bit[31]位 OWN 置为 1。
- 步骤 2 把读数据命令写入 CMD (物理地址为 0x0863002c)寄存器。
- 步骤 3 通过 FIFOTH (物理地址为 0x0863004c)寄存器设置 RX_Wmark。
- 步骤 4 IDMAC 获取描述符并判断 OWN 是否为 1, 如果 OWN 不为 1 则需要等待 CPU 释放描述符, 同时, IDMAC 会进入 suspend 状态, 需要 CPU 配置 PLDMND (物理地址为 0x08630084)寄存器使 IDMAC 重新获取描述符。
- 步骤 5 OWN 为 1 时, IDMAC 从 eMMC 内部 FIFO 中搬移数据到外部的数据 buffer。
- 步骤 6 如果中断已经使能, 数据搬移完成后 IDMAC 状态寄存器 IDSTS (物理地址为 0x0863008c)的相应位将被更新, 同时 OWN 位被清零。

----结束

14.5.10 Auto-stop 使用配置

在多块读写指令操作中, 需使用停止指令完成一次数据传输。停止指令的发送可以通过 [14.5.5 非数据传输指令](#)的方式发送, 也可以使用 Auto-stop 功能发送。

Auto-stop 功能的应用场合如下:

多块读写操作, 如指令 CMD18 和指令 CMD25。

建议使用控制器提供的 Auto-stop 功能, 配置过程如下:

- 步骤 1 在执行块数据传输指令操作中, 将寄存器 CMD[send_auto_stop] (物理地址为 0x0863002c)置 "1" 。
- 步骤 2 在所有数据传输完成后, eMMC 自动发送一次停止指令, 以便卡能返回相应的状态。
- 步骤 3 检查寄存器 RINTSTS[auto_cmd_done] (物理地址为 0x08630044), 判断该停止指令是否完成。其响应保存在寄存器 RESP1 中。

----结束

14.5.11 停止数据传输

停止指令用于打断 eMMC 与卡之间的数据传输, 中止指令用于打断 I/O 数据的传输 (仅用于 SDIO_IONLY 或 SDIO_COMBO)。



用法如下：

- 停止指令

该指令可以在数据传送的任何阶段进行发送。因为该指令用于停止数据传输，所以需要寄存器 CMD bit[5:0] 设为指令 CMD12，寄存器 CMD bit[14] 设为 1，寄存器 CMD bit[13] 设为 0。

获取已传输数据字节数。

- 中止指令

该指令仅用于 SDIO_IOONLY 或 SDIO_COMBO。为了中止数据的传输，需要通过指令 CMD52 设置对接器件 SDIO 卡的寄存器 CCCR[ASx] (SDIO 卡内部的寄存器)。

14.5.12 BOOT 模式 (仅 eMMC)

14.5.12.1 Original boot

步骤 1 为支持 boot 模式，首先要确认外接的 emmc 卡是支持 boot 操作的，通过 BOOT_PARTITION_ENABLE bit 位来体现。通过查询卡寄存器获得 BOOT_SIZE_MULT 和 boot 所用的数据传输位宽 (Extend CSD byte[177]bit[0:1])。

步骤 2 配置中断屏蔽寄存器 INTMASK (物理地址为 0x08630024)，配置 0x00。

步骤 3 配置 CTRL (物理地址为 0x08630000) 寄存器：

- 配置 enable_OD_pullup 使能 (command 输出一直在 open-drive mode，也就是说，只输出低电平或者高阻，不会输出高电平)；
- int_enable = 1'b1；
- 其他 bit 设置为 0；
- 如果需要用 IDMAC 搬运 boot 数据，需要按照 DMA 方式配置描述符，需要配置 use_internal_dmac 为 1。

步骤 4 配置时钟频率到 400kHz，参考 [13.4.3 时钟配置](#) 章节。

步骤 5 配置数据超时 DataTimeOut (可以使用默认值)。

步骤 6 配置 BLKSIZ (物理地址为 0x0863001c) 为 0x200 (512 Byte)。

步骤 7 根据 BOOT_SIZE_MULT 配置 BYTCNT (物理地址为 0x08630020) 寄存器。

步骤 8 配置 RXFIFO 的阈值 (物理地址为 0x0863004c)，一般设置为 FIFO_DEPTH/2-1。

步骤 9 配置 CMD 寄存器 (物理地址为 0x0863002c)。



- start_cmd = 1'b1
- enable_boot_ack = 1'b1 (为寄存器 expect_boot_ack 位)
- data_expected = 1'b1
- 其他配置为 0

步骤 10 如果 enable_boot_ack= 1'b1, 软件建立一个 50ms 的计时器。

- 计时器结束前, 控制器会发出 boot ack received (BAR) 中断。如果没有产生中断, 则软件必须发出如下 cmd (start_cmd= 1, disable_boot=1, 其他 bit 为 0), 控制器会发出 CD (command done) 中断。在 IDMAC 模式下, 描述符要被关闭, 配置 IDSTS (物理地址为 0x0863008c) 的 CES 位, 指示 BAR 超时, 不配置 RI。
- 如果收到 BAR 中断, 软件可以先清除中断, 然后发起一个 0.95s 的计时器, 在这个计时器结束前, 控制器发出 boot data start (BDS) 中断, 如果没有收到该中断, 软件必须发出如下 cmd (start_cmd= 1, disable_boot=1, 其他 bit 为 0), 在 IDMAC 模式下, 描述符要被关闭, 配置 IDSTS (物理地址为 0x0863008c) 的 CES 位, 指示 BDS 超时, 不配置 RI。
- 如果收到了 BDS 中断, 这说明卡侧的 boot 数据已经开始传输, 在非 IDMAC 模式下, 通过 RXDR (RINTSTS) 中断, 软件可以开始发起读操作。在 IDMAC 模式下, IDMAC 开始数据搬运。Boot 数据传输成功后, 可以查询以下寄存器 Command Done (RINTSTS), Data Transfer Over (RINTSTS)。如果是 IDMAC 模式, 则可查询 Receive Interrupt (IDSTS (物理地址为 0x0863008c))。
- 如果 boot ack pattern 或者 end bit 出错, 控制器则会拉高 CMD 线结束 boot, 产生 Command Done 中断, 也可以软件来结束 boot 传输。
- 在 IDMAC 模式下, 如果软件生成的描述符多于数据接收, 控制器不会自动关闭描述符。如果描述符少于数据接收, 控制器产生 (DU) Descriptor unavailable 中断, 不会进一步传输数据。

步骤 11 如果 enable_boot_ack=1'b0 (设为 0), 那么软件在[步骤 9](#)后应该发起 1 秒的计数器。

- 计数结束前, 控制器会发出 BDS (Boot Data Start) 中断。如果没有收到, 软件必须发出如下 CMD (start_cmd=1'b1, disable_boot=1, 其他 bit 为 0), 控制器会发出 CD (command done) 中断。在 IDMAC 模式下, 描述符要被关闭, 配置 IDSTS (物理地址为 0x0863008c) 的 CES 位, 指示 BDS 超时。
- 如果 BDS 中断被接收, 这表示 boot data 已经开始传输。在非 IDMAC 模式下, 软件可以根据 RXDR (RINTSTS) 中断, 开始读取数据。Boot 数据传输成功后,



可以查询以下寄存器 Command Done (RINTSTS), Data Transfer Over (RINTSTS)。如果是 IDMAC 模式, 则可查询 Receive Interrupt (IDSTS (物理地址为 0x0863008c))。

- 在 IDMAC 模式下, 如果软件生成的描述符多于数据接收, 控制器不会自动关闭描述符。如果描述符少于数据接收, 控制器产生 (DU) Descriptor unavailable 中断, 不会进一步传输数据。

----结束

14.5.12.2 Alternative boot

- 步骤 1 为支持 boot 模式, 首先要确认外接的 eMMC 卡是支持 alternative boot 操作的, 通过 BOOT_INFO bit 位来体现。通过查询卡寄存器获得 BOOT_SIZE_MULT 和 boot 所用的数据传输位宽 (Extend CSD byte[177]bit[0:1])。
- 步骤 2 配置中断屏蔽寄存器 (物理地址为 0x08630024), 配置 0x00。
- 步骤 3 配置 CTRL (物理地址为 0x08630000) 寄存器。
- 配置 enable_OD_pullup 使能 (command 输出一直在 open-drive mode, 也就是说, 只输出低电平或者高阻, 不会输出高电平);
 - int_enable = 1'b1;
 - 其他 bit 设置为 0;
 - 如果需要用 IDMAC 搬运 boot 数据, 需要按照 DMA 方式配置描述符, 需要配置 use_internal_dmac 为 1。
- 步骤 4 根据卡的 BOOT_INFO 配置时钟频率, 参考时钟配置章节, 不支持 DDR 模式。
- 步骤 5 等待一段时长大于 74 个卡时钟周期。
- 步骤 6 配置数据超时 DataTimeOut。
- 步骤 7 配置 BLKSIZ (物理地址为 0x0863001c) 为 0x200 (512 Byte)。
- 步骤 8 根据 BOOT_SIZE_MULT 配置 BYTCNT (物理地址为 0x08630020) 寄存器。
- 步骤 9 配置 RXFIFO 的阈值 (物理地址为 0x0863004c), 一般设置为 FIFO_DEPTH/2-1。
- 步骤 10 配置 CMDARG (物理地址为 0x08630028) = 0xFFFFFFFFFA。
- 步骤 11 配置 CMD 寄存器 (物理地址为 0x0863002c)。
- start_cmd = 1'b1
 - boot_mode = 1'b1
 - enable_boot_ack = 1'b1 (为寄存器 expect_boot_ack 位)



- data_expected = 1'b1
- cmd_index = 1'b0
- 其他配置为 0

步骤 12 软件等待 Command Done (CD) 中断。

步骤 13 如果 enable_boot_ack=1'b1, 软件建立一个 50ms 的计时器。

- 计时器结束前, 控制器会发出 boot ack received (BAR) 中断。如果没有产生中断, 则软件需要终止 boot 进程, 在 IDMAC 模式下, 描述符要被关闭, 配置 IDSTS (物理地址为 0x0863008c) 的 CES 位, 指示 BAR 超时, 不配置 RI。
- 如果收到 BAR 中断, 软件可以先清除中断, 然后发起一个 0.95s 的计时器, 在这个计时器结束前, 控制器发出 boot data start (BDS) 中断, 如果没有收到该中断, 软件需要终止 boot 进程, 在 IDMAC 模式下, 描述符要被关闭, 配置 IDSTS (物理地址为 0x0863008c) 的 CES 位, 指示 BDS 超时, 不配置 RI。
- 如果收到了 BDS 中断, 这说明卡侧的 boot 数据已经开始传输, 在非 IDMAC 模式下, 通过 RXDR (RINTSTS) 中断, 软件可以开始发起读操作。在 IDMAC 模式下, IDMAC 开始数据搬运。Boot 数据传输成功后, 可以查询以下寄存器 Command Done (RINTSTS), Data Transfer Over (RINTSTS)。如果是 IDMAC 模式, 则可查询 Receive Interrupt (IDSTS (物理地址为 0x0863008c))。
- 软件可以通过发送 CMD0 (CMDARG=0, start_cmd=1, cmd_index=0, 其他为 0) 来结束 boot 操作。
- 如果 boot ack pattern 或者 end bit 出错, 控制器不会产生 BAR 中断且会继续接受数据, 控制器产生 BDS 中断, 软件收到 BDS 中断后结束 boot 传输。
- 在 IDMAC 模式下, 如果软件生成的描述符多于数据接收, 控制器不会自动关闭描述符。如果描述符少于数据接收, 控制器产生 Descriptor unavailable 中断, 不会进一步传输数据。

步骤 14 如果 enable_boot_ack=1'b0, 软件建立一个 1s 的计时器。

- 计时器结束前, 控制器会发出 boot data start (BDS) 中断。如果没有产生中断, 则软件需要终止 boot 进程, 在 IDMAC 模式下, 描述符要被关闭, 配置 IDSTS (物理地址为 0x0863008c) 的 CES 位, 指示 BDS 超时, 不配置 RI。
- 如果收到了 BDS 中断, 这说明卡侧的 boot 数据已经开始传输, 在非 IDMAC 模式下, 通过 RXDR (RINTSTS) 中断, 软件可以开始发起读操作。在 IDMAC 模式下, IDMAC 开始数据搬运。



- 软件可以通过发送 CMD0 (CMDARG=0, start_cmd=1,cmd_index=0,其他为 0) 来结束 boot 操作。
- Boot 数据传输成功后, 可以查询以下寄存器 Command Done (RINTSTS), Data Transfer Over (RINTSTS)。如果是 IDMAC 模式, 则可查询 Receive Interrupt (IDSTS (物理地址为 0x0863008c))。
- 在 IDMAC 模式下, 如果软件生成的描述符多于数据接收, 控制器不会自动关闭描述符。如果描述符少于数据接收, 控制器产生 Descriptor unavailable 中断, 不会进一步传输数据。

----结束



15 SDIO

15.1 概述

支持一个 SDIO/MMC 控制器，用于连接板载的 SDIO 扩展模块（蓝牙，WIFI 等）。

SDIO 控制用于对外对接扩展的 SDIO 设备，支持以下协议：

Secure Digital I/O (SDIO-version 2.0)

15.2 特性

- 支持内部 DMA 数据传输方式。
- 包含数据发送方向与接收方向的 2 个 FIFO，FIFO 深度为 256Word。
- 支持 FIFO 阈值可配，DMA 传输时 Burst 大小可配。
- 支持 FIFO 上溢出与下溢出中断告警，防止数据传输错误。
- 支持命令与数据的 CRC 生成与校验。
- 接口时钟频率可编程，最大可配置为 50MHz。
- 支持低功耗模式关闭 SDIO 时钟和接口时钟。
- 数据位宽支持 1、4bit，可根据对接器件选择。
- 支持 1Byte ~ 512Byte 的块数据读写操作。
- 支持 SDIO 的 suspend 操作、resume 操作和 readwait 操作。
- 支持 SDIO 端口，占用 9bit 管脚。



15.3 关键术语

表15-1 SDIO 关键术语描述

术语	描述
SDIO	Secure Digital Input and Output, 安全数字输入输出。

15.4 功能

SDIO 控制用于对外对接扩展的 SDIO 设备，支持以下协议：

Secure Digital I/O (SDIO-version 2.0)。

15.5 工作方式

15.5.1 工作时钟门控

当 SDIO 控制器处于空闲状态时，可以关闭其工作时钟。

具体步骤如下：

步骤 1 读寄存器 STATUS (物理地址为 0x08610048)。

步骤 2 若 STATUS[cmd_fsm_states]和 STATUS[data_state_mc_busy]均为 0，则向 CTRL 写 (物理地址为 0x08610000) “0”，屏蔽中断、DMA 请求使能等，进入步骤 3。若其中有一个为非 0，则延时等待，返回步骤 1。

步骤 3 向 LP_SUBSYS 控制寄存器 LPSC_CLKEN_CTRL1 bit[26] (物理地址为 0x0871d014) 写 “1”，打开 SDIO 工作时钟。

步骤 4 如果需要关闭工作时钟，LP_SUBSYS 控制寄存器 LPSC_CLKEN_CTRL1 bit[26] (物理地址为 0x0871d014) 写 “0”。

----结束



15.5.2 软复位

在数据传输出现异常而导致 SDIO 无法回到空闲状态时，配置 LP_SUBSYS 寄存器 `sdio_srst_req` 对 SDIO 进行软复位。

步骤 1 通过配置 LP_SUBSYS 系统控制器 LPSC_SRST_REQ0 bit[3]（物理地址 0x0871d004）为 1，可实现 SDIO 控制器工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置 LP_SUBSYS 系统控制器 LPSC_SRST_REQ0 bit[3]（物理地址 0x0871d004）为 0，可实现 SDIO 控制器工作逻辑的单独软复位撤离。

----结束

须知

建议在使用 SDIO 控制器之前或拔插 SDIO 设备前后软复位控制器。

15.5.3 时钟配置

15.5.3.1 接口时钟配置

SDIO 控制器卡侧工作时钟来源于 TOP_CRG 经过 LP_SUBSYS 系统寄存器 `sdio_clk_div_cnt`（LPSC_CRG_DIV_CTRL bit[3:0]物理地址为 0x0871d018）分频得来：

$$F_{\text{CCLK_IN}} = F_{\text{LP_CLK}} / \text{sdio_clk_div_cnt}$$

控制器工作时钟 CCLK_IN 与接口时钟 SD_CLK 的频率关系为：

$$F_{\text{SD_CLK}} = F_{\text{CCLK_IN}} / (2 \times \text{clk_divider})$$

其中，分频因子 `clk_divider` 为寄存器 `MMC_CLKDIV[clk_divider]` 的值。不同卡类型支持的时钟频率不同，卡侧 $F_{\text{SD_CLK}}$ 最高不超过 50MHz。

在改变 SDIO 卡的时钟频率之前，必须保证没有数据或指令正在传输。为了避免输出到 SDIO 卡的时钟产生毛刺，在改变 SDIO 卡的时钟频率时应该遵照以下步骤：

步骤 1 关闭接口时钟。将寄存器 `MMC_CLKENA`（物理地址为 0x08610010）配置为 0x0000_0000，并将寄存器 `MMC_CMD`（物理地址为 0x0861002c）[Start_cmd]、



MMC_CMD[Update_clock_registes_only]和 MMC_CMD[Wait_prvdata_complete]置“1”，等待寄存器 MMC_CMD[Start_cmd]被自动清零。

步骤 2 设置分频因子。根据所需要的时钟频率设置寄存器 MMC_CLKDIV（物理地址为 0x08610008），并将寄存器 MMC_CMD[Start_cmd]和 MMC_CMD[Update_clock_registes_only]置“1”，等待寄存器 MMC_CMD[Start_cmd]被自动清零。

步骤 3 重新使能接口时钟。将寄存器 MMC_CLKENA（物理地址为 0x08610010）配置为 0x0000_0001，并将寄存器 MMC_CMD(物理地址为 0x0861002c) [Start_cmd]和 MMC_CMD[Update_clock_registes_only]置“1”，等待寄存器 MMC_CMD[Start_cmd]被自动清零。

----结束

须知

- 只有当寄存器 MMC_CMD[Start_cmd]和 MMC_CMD[Update_clk_only]置“1”时，寄存器 MMC_CLKDIV、MMC_CLKENA 的值才会被载入。当载入成功以后，MMC 会自动清零寄存器 MMC_CMD[Start_cmd]。如果此时有其他指令正在执行，则会产生 HLE（HardwareLockedError）中断。若产生 HLE 中断，清除中断后重新发命令即可。
- LP_SUBSYS 控制寄存器[mmc_cclk_samp_ctrl]（LPSC_MDIO_SDIO_CTRL[5]，物理地址为 0x0871d020）建议配置为 1。
- 当有指令执行和数据传输时，不能改变卡的时钟参数。

15.5.4 初始化

与卡进行命令和数据的交互前，需要初始化 SDIO 控制器，步骤如下：

步骤 1 配置 SDIO 工作时钟频率。

步骤 2 当卡上电，指令和数据信号线上拉稳定后，解复位控制器（跟随芯片同步复位撤离）。

步骤 3 清中断。将寄存器 MMC_RINTSTS（物理地址为 0x08610044）bit[15:0]所有位置“1”，清除原始中断状态位。



- 步骤 4 设置寄存器 MMC_INTMASK (物理地址为 0x08610024)。将寄存器 MMC_INTMASK bit[15:0]所有位置 “1” , 使能各中断源。若使用 DMA 方式进行数据传输, 应将 MMC_INTMASK bit[4]、MMC_INTMASK bit[5]置 “0” , 以屏蔽接收/发送 FIFO 数据请求中断。
- 步骤 5 将寄存器 MMC_CTRL (物理地址为 0x08610000) [int_enable]置 “1” , 使能 SDIO 中断。
- 步骤 6 若使用 DMA 方式进行数据传输, 应将寄存器 MMC_CTRL[use_internal_dmac]置 “1” , 使能 DMA 请求功能。
- 步骤 7 配置超时参数寄存器 MMC_TMOU (物理地址为 0x08610014)。
- 步骤 8 配置 FIFO 参数寄存器 MMC_FIFOTH (物理地址为 0x0861004c)。

----结束

15.5.5 非数据传输指令

SDIO 在指令发送后, 一旦收到响应 (无论对错或超时), 就会将寄存器 MMC_RINTSTS bit[2]置位。短响应保存到寄存器 MMC_RESP0 中, 长响应保存到寄存器 MMC_RESP0 ~ MMC_RESP3 中, 寄存器 MMC_RESP3 bit[31]为最高位, MMC_RESP0 bit[0]为最低位。当指令发出以后, 其错误是由指令响应以及寄存器 MMC_RINTSTS 的错误位反映。发送非数据传输指令的步骤如下:

- 步骤 1 在寄存器 MMC_CMDARG (物理地址为 0x08610028) 中设置相应的指令参数。
- 步骤 2 根据表 14-3 设置指令寄存器 MMC_CMD (物理地址为 0x0861002c)。
- 步骤 3 等待指令被 SDIO 执行。如果指令已执行, SDIO 自动清零寄存器 MMC_CMD[start_cmd]。
- 步骤 4 检查寄存器 MMC_RINTSTS[12] (物理地址为 0x08610044) 是否产生 HLE 中断。
- 步骤 5 等待指令执行完毕。SDIO 收到响应 (无论对错或超时) 时将寄存器 MMC_RINTSTS bit[2]置 “1” , 表示 CommandDone。
- 步骤 6 检查是否有响应异常, 必要时可读取响应值。
- 步骤 7 读取寄存器 MMC_RINTSTS bit[8]、MMC_RINTSTS bit[6]和 MMC_RINTSTS bit[1], 检查响应超时、响应 CRC 错误和响应错误。

----结束



须知

只有当寄存器 MMC_CMD[start_cmd]置 “1”、
MMC_CMD[update_clock_registes_only]置 “0” 时，寄存器 MMC_BYTCNT、
MMC_BLKSIZE、MMC_CMDARG 和 MMC_CMD 的值才能被载入。载入成功后，
SDIO 自动清零 MMC_CMD[start_cmd]。

如果有其他指令正在执行，会产生 HLE 中断，此时重新执行操作即可。在非数据传输
指令执行时，寄存器 MMC_BYTCNT 和 MMC_BLKSIZE 的值被忽略。

表15-2 非数据传输指令时的寄存器 MMC_CMD 配置参考（默认值）

参数	取值	描述
start_cmd	1	启动指令发送。
update_clock_registes_only	0	非时钟参数更新指令。
data_transfer_expected	0	非数据传输指令。
card_number	0	-
cmd_index	cmdindex	指令序号。
send_initialization	0	当指令为卡复位时置 “1” ，如指令 CMD0。
stop_abort_cmd	0	当指令为停止数据传输时置 “1” ，如 指令 CMD12。
rsponse_length	0	当响应为长响应类型时置 “1” 。
rsponse_expect	1	当指令无响应时置 “0” ，如：指令 CMD0、指令 CMD4、指令 CMD15。
wait_prvdata_complete	1or0	在发送指令之前，控制器必须等待正在 处理的数据传输指令结束。建议此位一 直置 “1” ，除非该指令是为了在数据 传输时查询卡状态或停止当前数据的传 输。
check_response_crc	1or0	控制器是否检查响应的 CRC 校验位。



15.5.6 单块或多块读数据

读取单块或多块数据的步骤如下：

- 步骤 1 向寄存器 MMC_CTRL[fifo_reset] (物理地址为 0x08610000) 写 “1” ， 复位 FIFO 指针， 查询等待该位自动清零。
- 步骤 2 向寄存器 MMC_BYTCNT (物理地址为 0x08610020) 写入待传输数据的字节数。
- 步骤 3 向寄存器 MMC_BLKSIZE (物理地址为 0x0861001c) 写入块的大小。
- 步骤 4 向寄存器 MMC_CMDARG (物理地址为 0x08610028) 写入读取数据的起始地址。
- 步骤 5 根据表 14-4 设置寄存器 MMC_CMD (物理地址为 0x0861002c)。
- 步骤 6 对于 SDIO 卡， 使用指令 CMD53 进行单块/多块读操作。
- 步骤 7 一旦寄存器 MMC_CMD 被写入， SDIO 就执行指令； 当指令被送到总线上以后， 产生 cmd_done 中断。
- 步骤 8 检查寄存器 MMC_RINTSTS bit[5]和 MMC_RINTSTS bit[10] (物理地址为 0x08610044)， 如果其中之一为 1 或都为 1， 则从寄存器 DATA 读取 FIFO 中的数据， 以便 SDIO 接收后面的数据； 同时检查数据错误中断， 即寄存器 MMC_RINTSTS bit[7]、 MMC_RINTSTS bit[9]、 MMC_RINTSTS bit[13]和 MMC_RINTSTS bit[15]。 此时， 程序可以发送一个停止指令中止数据的传输。
- 步骤 9 当寄存器 MMC_RINTSTS bit[3]为 1 时， 数据传输完成， 从寄存器 MMC_DATA (物理地址为 0x08610200) 中读取残留在 FIFO 中的数据。
- 步骤 10 若执行指令时已将寄存器 MMC_CMD[send_auto_stop] (物理地址为 0x0861002c)置 “1” ， SDIO 动发送停止指令结束一次数据传输， 请参见 14.5.10 Auto-stop 使用配置。

----结束

表15-3 单块或多块读数据时的寄存器 MMC_CMD 配置参考（默认值）

参数	取值	描述
start_cmd	1	启动指令发送。
update_clock_registes_only	0	非时钟参数更新指令。
card_number	0	-



参数	取值	描述
send_initialization	0	当指令为卡复位时置“1”，如指令 CMD0。
stop_abort_cmd	0	当指令为停止数据传输时置“1”，如指令 CMD12。
send_auto_stop	0or1	请参见 14.5.10 Auto-stop 使用配置中的“Auto-stop 使用配置”。
transfer_mode	0	块传输。
read/write	0	从卡中读取数据。
rsponse_length	0	数据指令均为短响应。
data_transfer_expected	1	数据传输指令。
rsponse_expect	1	当指令无响应时置“0”，如：指令 CMD0、CMD4、CMD15。
cmd_index	Cmdindex	命令序号。
wait_prvdata_complete	1or0	在发送指令之前，主设备必须等待正在处理的数据传输指令结束。建议此位一直置 1，除非该指令是为了询问卡状态或停止当前数据的传输。
check_response_crc	1or0	控制器是否检查响应的 CRC 校验位。

15.5.7 单块与多块写数据

写入单块或多块数据的步骤如下：

- 步骤 1 向寄存器 MMC_CTRL[fifo_reset] (物理地址为 0x08610000) 写“1”，复位 FIFO 指针，查询等待直至该位自动清零。
- 步骤 2 向寄存器 MMC_BYTCNT (物理地址为 0x08610020) 写入待传输数据的大小。
- 步骤 3 向寄存器 MMC_BLKSIz (物理地址为 0x0861001c) 写入块的大小。
- 步骤 4 向寄存器 MMC_CMDARG (物理地址为 0x08610028) 写入数据的起始地址。



- 步骤 5 将数据写入 FIFO，即写寄存器 MMC_DATA (物理地址为 0x08610200)，通常在开始时应写满 FIFO。
- 步骤 6 根据表 14-5 设置寄存器 MMC_CMD (物理地址为 0x0861002c)。对于 eMMC 卡，分别使用指令 CMD24/CMD25 进行单块/多块写操作；
- 步骤 7 检查寄存器 MMC_RINTSTS[4]和 MMC_RINTSTS bit[10] (物理地址为 0x08610044)，如果其中之一为 1 或都为 1，写寄存器 MMC_DATA 往 FIFO 填充数据；同时应检查数据错误中断，即检查寄存器 MMC_RINTSTS bit[7]、MMC_RINTSTS bit[9]、MMC_RINTSTS bit[13]和 MMC_RINTSTS bit[15]。如果有需要，程序可以发送一个停止指令以中止数据的传输。当寄存器 MMC_RINTSTS bit[3]为 1，数据传输结束。
- 步骤 8 若执行指令时已将寄存器 MMC_CMD[send_auto_stop] (物理地址为 0x0861002c)置“1”，eMMC 自动发送停止指令结束一次数据传输。请参见 14.5.10 Auto-stop 使用配置。
- 步骤 9 查询并等待寄存器 MMC_STATUS[data_busy] (物理地址为 0x08610048) 由 1 变为 0。

----结束

表15-4 单块或多块写数据时的寄存器 MMC_CMD 配置参考（默认值）

参数	取值	描述
start_cmd	1	启动指令发送。
update_clock_registes_only	0	非时钟参数更新指令。
card_number	0	-
send_initialization	0	当指令为卡复位时置“1”，如指令 CMD0。
stop_abort_cmd	0	当指令为停止数据传输时置“1”，如指令 CMD12。
send_auto_stop	0or1	请参见 14.5.10 Auto-stop 使用配置。
transfer_mode	0	块传输。
read_write	1	往卡写入数据。
rsponse_length	0	数据指令均为短响应。



参数	取值	描述
data_transfer_expected	1	数据传输指令。
rsponse_expect	1	当指令无应时置“0”，如：指令 CMD0、CMD4、CMD15。
cmd_index	Cmdindex	-
wait_prvdata_complete	1or0	在发送指令之前，主设备必须等待直到正在处理的数据传输指令结束。建议此位一直置“1”，除非该指令是为了询问卡状态或停止当前数据的传输。
check_response_crc	1or0	eMMC 是否检查响应的 CRC 校验位。

15.5.8 流数据读写

流数据的读写方式，除了将寄存器 MMC_CMD[transfer_mode]置“1”外，其他与块数据的读写方式一致。对于流数据的传输，通常需要使用 Auto-stop 功能。

15.5.9 内置 DMA 方式数据传输

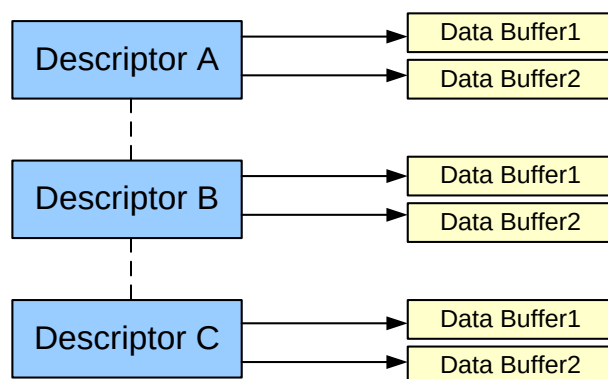
SDIO 控制器含有内置 DMA 控制器（IDMAC），IDMAC 可以根据指定的描述符把数据从原地址搬移到目的地址。

15.5.9.1 描述符

IDMAC 可以使用如下两种类型的描述符：

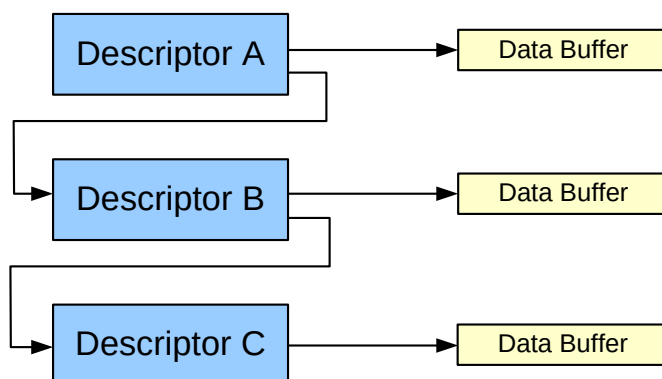
- 双 Buffer 结构：两个描述符之间的间隔根据 BMOD 寄存器的 DSL 位来指定。双 buffer 结构的示意图如图 15-1 所示。

图15-1 双 buffer 结构示意图



- 链结构：每个描述符指向唯一的 buffer 和下一个描述符。链结构的示意图如图 15-2 所示。

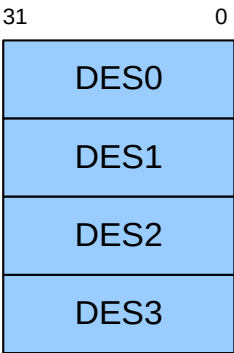
图15-2 链结构示意图



描述符必须以 Word 对齐，每个描述符包含 16Byte 的控制和状态信息。在链结构模式下每个描述符最多可传输 4KB 数据，在双 buffer 模式下每个描述符最多可传输 8KB 数据。描述符的内部结构如图 15-3 所示。



图15-3 32bit 位宽的描述符的结构



DES0 保护控制和状态信息，具体含义如表 15-5 所示。

表15-5 DES0 各 bit 的含义

位	名称	说明
31	OWN	描述符属性指示。 0：表示该描述符属于 CPU； 1：表示该描述符属于 IDMAC。 使用内部 DMA 完成数据传输后，IDMAC 将该位清 0。
30	CES	读写卡时出错的状态指示。 0：没有错； 1：出错。
29:6	RES	保留。
5	ER	描述符链路结束指示。 0：不是最后一个； 1：该链路最后一个描述符。 只对双 buffer 结构的描述符有意义。
4	CH	第二个地址（DES3 的地址）的含义。 0：DES3 中的第二个地址是指第二个 buffer 的地址； 1：DES3 中的第二个地址是指下一个描述符的地址。但该位为 1 时，DES1 bit[25:13]应该是 0。



位	名称	说明
3	FS	为 1 表示该描述符包含第一个数据 buffer，如果第 1 个数据 buffer 的大小为 0，那么下一个描述符包含起始数据。
2	LD	为 1 表示该描述符指向的 buffer 是最后一个数据 buffer。
1	DIC	为 1 时阻止数据传输结束上报中断。
0	RES	保留。

DES1 指示 buffer 大小，具体含义如表 15-6 所示。

表15-6 DES1 各 bit 的含义

位	名称	说明
31: 26	RES	保留。
25: 13	BS2	第二个数据 buffer 的 Byte 数，必须是 4 的整数倍。DES0 bit[4] 为 1 时该位无意义。
12: 0	BS1	第一个数据 buffer 的 Byte 数，必须是 4 的整数倍。

DES2 表示第一个数据 buffer 的地址指针，具体含义如表 15-7 所示。

表15-7 DES2 各 bit 的含义

位	名称	说明
31:0	BAP1	第一个数据 buffer 的物理地址，必须 Word 对齐。

DES3 表示第二个地址，具体含义如表 15-8 所示。



表15-8 DES3 各 bit 的含义

位	名称	说明
31:0	BAP2	当使用的是双 buffer 结构时，表示第二个数据 buffer 的物理地址。如果 DES0 bit[4]是 1，该地址表示下一个描述符的物理地址。

15.5.9.2 初始化

- 步骤 1 配置 BMOD (物理地址为 0x08610080)寄存器，设置总线参数。
- 步骤 2 配置 IDINTEN (物理地址为 0x08610090)寄存器，屏蔽不需要的寄存器。
- 步骤 3 创建发送或接收描述符链表，然后配置 DBADDR (物理地址为 0x08610088)寄存器，设置起始地址。
- 步骤 4 IDMAC 尝试从描述符链表中获取描述符。

----结束

15.5.9.3 发送

- 步骤 1 CPU 创建描述符 (DES0~DES3) 并将 DES0 bit[31]位 OWN 置为 1，同时准备好数据 buffer。
- 步骤 2 把写数据命令写入 CMD (物理地址为 0x0861002c)寄存器。
- 步骤 3 通过 FIFOTH (物理地址为 0x0861004c)寄存器设置 TX_Wmark。
- 步骤 4 IDMAC 获取描述符并判断 OWN 是否为 1，如果 OWN 不为 1 则需要等待 CPU 释放描述符，同时，IDMAC 会进入 suspend 状态，需要 CPU 配置 PLDMND(物理地址为 0x08610084)寄存器使 IDMAC 重新获取描述符。
- 步骤 5 OWN 为 1 时，IDMAC 从数据 buffer 中搬移数据到 eMMC 内部 FIFO。
- 步骤 6 如果中断已经使能，数据搬移完成后 IDMAC 状态寄存器 IDSTS (物理地址为 0x0861008c)的相应为将被更新，同时 OWN 位被清零。

----结束



15.5.9.4 接收

- 步骤 1 CPU 创建描述符 (DES0~DES3) 并将 DES0 bit[31]位 OWN 置为 1。
- 步骤 2 把读数据命令写入 CMD (物理地址为 0x0861002c)寄存器。
- 步骤 3 通过 FIFOTH (物理地址为 0x0861004c)寄存器设置 RX_Wmark。
- 步骤 4 IDMAC 获取描述符并判断 OWN 是否为 1, 如果 OWN 不为 1 则需要等待 CPU 释放描述符, 同时, IDMAC 会进入 suspend 状态, 需要 CPU 配置 PLDMND (物理地址为 0x08610084)寄存器使 IDMAC 重新获取描述符。
- 步骤 5 OWN 为 1 时, IDMAC 从 eMMC 内部 FIFO 中搬移数据到外部的数据 buffer。
- 步骤 6 如果中断已经使能, 数据搬移完成后 IDMAC 状态寄存器 IDSTS (物理地址为 0x0861008c)的相应位将被更新, 同时 OWN 位被清零。

----结束

15.5.10 Auto-stop 使用配置

在多块读写指令操作中, 需使用停止指令完成一次数据传输。停止指令的发送可以通过 [14.5.5 非数据传输指令](#)的方式发送, 也可以使用 Auto-stop 功能发送。

Auto-stop 功能的应用场合如下:

- 多块读写操作, 如指令 CMD18 和指令 CMD25。

建议使用控制器提供的 Auto-stop 功能, 配置过程如下:

- 步骤 1 在执行块数据传输指令操作中, 将寄存器 CMD[send_auto_stop] (物理地址为 0x0861002c)置 "1" 。
- 步骤 2 在所有数据传输完成后, 控制器自动发送一次停止指令, 以便卡能返回相应的状态。
- 步骤 3 检查寄存器 RINTSTS[auto_cmd_done] (物理地址为 0x08610044), 判断该停止指令是否完成。其响应保存在寄存器 RESP1 中。

----结束

15.5.11 停止数据传输

停止指令用于打断控制器与卡之间的数据传输, 中止指令用于打断 I/O 数据的传输 (仅用于 SDIO_IOONLY 或 SDIO_COMBO)。用法如下:



- 停止指令

该指令可以在数据传送的任何阶段进行发送。因为该指令用于停止数据传输，所以需要寄存器 CMD bit[5:0] 设为指令 CMD12，将寄存器 CMD bit[14] 设为 1，将寄存器 CMD bit[13] 设为 0。

获取已传输数据字节数。

- 中止指令

该指令仅用于 SDIO_IOONLY 或 SDIO_COMBO。为了中止数据的传输，需要通过指令 CMD52 设置对接器件 SDIO 卡的寄存器 CCCR[ASx] (SDIO 卡内部的寄存器)。

15.5.12 Suspend 和 Resume 操作

控制器可通过 Suspend 操作暂停某一功能设备的数据传输，将 SD 接口总线出让给另一个有更高优先级的功能设备。高优先级的功能设备完成数据传输后，通过 Resume 操作恢复前一功能设备未完成的传输。

Suspend 与 Resume 操作通过设置 SD 卡的寄存器 CCCR 来实现。读写寄存器 CCCR，使用指令 CMD52。

15.5.12.1 Suspend 操作步骤

步骤 1 查询寄存器 CCCR[SBS]，判断 SD 卡是否支持 suspend/resume 操作。

步骤 2 查询寄存器 CCCR[FSx]和[BS]，判断待暂停的功能设备是否正在进行数据传输。

步骤 3 如果[BS]为 1，则[FSx]位所指定的功能设备正在进行数据传输。

步骤 4 将寄存器 CCCR[BR]置“1”，暂停当前数据传输。

步骤 5 检测寄存器 CCCR[BS]和[BR]状态是否清零。

步骤 6 [BS] (Bus 状态) 在数据总线正被使用时保持为 1。[BR] (Bus 释放) 在总线完全释放之前保持为 1。当[BR]与[BS]都为 0 时，所选功能设备的数据传输被暂停。

步骤 7 如果暂停正在进行的读操作，在 suspend 操作成功完成以后，必须置位寄存器 MMC_CTRL[Abort_read_data]来复位 MMC 的数据传输功能。复位完成后，寄存器 MMC_CTRL[Abort_read_data]自动清零。

步骤 8 读寄存器 MMC_TCBCNT 获取已传输数据字节数。

----结束



15.5.12.2 Resume 操作步骤

- 步骤 1 检查卡是否处于非传输状态，以确认总线处于空闲状态。
- 步骤 2 如果卡处于 disconnect 状态，使用指令 CMD7 将它选中。卡的状态可以通过指令 CMD52/CMD53 获取。
- 步骤 3 检查待恢复的功能设备是否准备好进行数据传输（查询寄存器 CCCR[RF]）。如果 [RF]=1，则该功能设备已准备好进行数据传输。
- 步骤 4 为了恢复传输，使用指令 CMD52 将功能设备号写入寄存器 CCCR[FS]。发送指令 CMD52 的同时应启动 MMC 进入数据传输状态，即向寄存器 MMC_BLKSIZE 写入块的大小，向寄存器 MMC_BTDCNT 写入剩余待传输数据量。寄存器 MMC_CMDARG 的配置如表 15-9 所示，寄存器 MMC_CMD 的配置与块传输类似。
- 步骤 5 当指令 CMD52 成功发送以后，数据传输恢复正常。读取卡设备的 DF (ResumeDataFlag) 标志位，如果为 1，则在功能被恢复的同时，数据开始传输；如果为 0，则已无数据需要传输。
- 步骤 6 如果 DF 标志位为 0，在读数据的情况下，控制器会等待一段时间后产生数据超时错误中断。

----结束

表15-9 Resume 操作时的寄存器 MMC_CMDARG 配置参考

MMC_CMDARG	取值	描述
bit[31]	1	读写标志。
bit[30:28]	0	功能设备号，访问寄存器 CCCR。
bit[27]	1	实时标志，先写后读。
bit[26]	-	-
bit[25:9]	0x0D	寄存器地址。
bit[8]	-	-
bit[7:0]	被恢复的功能号	写数据。



须知

系统处于低功耗模式后不能通过 SDIO 唤醒。

15.5.13 Readwait 操作

Readwait 操作用于暂停 SD 卡当前功能设备的数据传输。MMC 根据需要决定暂停数据传输的时间长度。

Readwait 操作步骤如下：

- 步骤 1 检查卡是否支持 readwait 操作。
- 步骤 2 使用指令 CMD52 读取寄存器 CCCR[SRW]。如果为 1，则卡的所有功能设备都支持 readwait 操作。
- 步骤 3 如果卡支持 readwait 操作，将寄存器 MMC_CTRL[Read_wait]置“1”。
- 步骤 4 如需恢复数据传输，清零寄存器 MMC_CTRL[Read_wait]。

----结束



16 MDIO

16.1 概述

MDIO 是 Hi3093 对外部 PHY 芯片的接口控制模块，软件可以通过配置 MDIO 相应寄存器实现对 PHY 芯片内部寄存器的读写，最终实现对 PHY 芯片的控制和管理。例如复位控制、自环控制、速度选择、双工选择、冲突测试、自协商使能、物理层状态获取等等。

16.2 特性

- 支持 MDIO 接口时序，时钟频率可配置为 0.5/2.5/6.25MHz。
- 每个 MDIO 接口可独立配置为支持 IEEE802.3 协议的 Clause22 或 Clause45。
- 支持对 PHY 内部寄存器的读写访问，PHY 地址、寄存器地址等可配置。
- 支持 1 个 MDIO 模块。
- 支持对 GE PHY 内部寄存器的自动查询，每个 MDIO 接口最大支持 5 个 PHY 器件的状态查询，PHY 状态改变则上报中断（每个配置的 PHY 地址可以监控指定最多 2 个地址寄存器的任意指定位状态值的改变）。
- 支持 3.3V 电压。

16.3 遵循协议

遵循 IEEE802.3 协议。

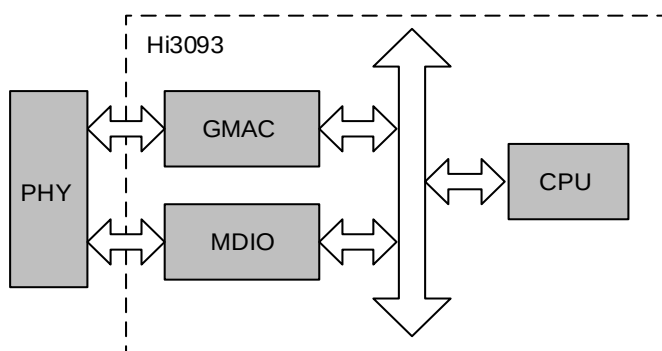


16.4 功能描述

16.4.1 应用框图

MDIO 是 Hi3093 对外部网口 PHY（后续所述 PHY 均指对接网口的 PHY）芯片的接口控制模块，软件可以通过配置 MDIO 相应寄存器实现对 PHY 芯片内部寄存器的读写，最终实现对 PHY 芯片的控制和管理。例如复位控制、自环控制、速度选择、双工选择、冲突测试、自协商使能、物理层状态获取等等，芯片内 1 个 MDIO 模块。

图16-1 MDIO 在 Hi3093 中的位置



16.4.2 关键术语

术语将在本章节表示特殊的含义。

表16-1 MDIO 模块关键术语描述

术语	描述
MDIO	Management Data Input/Output，管理数据输入输出接口。
GMAC	Gigabit Media Access Control，管理数据输入输出接口。
PHY	Physical Layer Device，管理数据输入输出接口。

16.4.3 数据结构

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



16.4.3.1 Clause22 的数据帧格式

表16-1 MDIO 接口的 Clause22 数据帧格式

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

数据帧各域的含义如下：

- PRE：帧同步码，32bit 位宽。
PRE 是 32 个连续的“1”，用于器件的同步通信。但该域不是必须的，为了提高效率，有些 PHY 器件的 MDIO 操作就没有此域。
- ST：帧开始标志。MAC 输出“01” bit 通知 PHY 表示帧操作开始。
- OP：帧操作码，2bit 位宽。
 - 01：表示此帧为一次写操作请求。
 - 10：表示此帧为一次读操作请求。
- PHY AD：PHY 器件地址，5bit 位宽。
每个 PHY 器件都把自己的地址与这 5bit 进行比较，若匹配则响应后面的操作；若不匹配，则忽略后面的操作。
5bit 决定了一个 MDIO 接口最多只能支持 32 个 PHY 器件的配置。
- REG AD：PHY 寄存器的地址，5bit 位宽。
用来选择 PHY 内部的 32 个寄存器中的某个寄存器，PHY 为物理层接口，每个 PHY 接口含有 32 个 16bit 的寄存器。
一般情况下，不同 PHY 芯片的前 16 个寄存器内容应该一致，后面 16 个是可由用户指定功能的寄存器。
- TA：状态转换域，2bit 位宽。
 - 读操作时，第 1bit 由 MDIO 送出高阻态，第 2bit 由 PHY 给 MDIO 信号置 0。
 - 写操作时，MDIO 信号由 MDIO 模块控制，连续输出“10”两个 bit。
- DATA：帧的寄存器数据域，16bit 位宽。



- 读操作时，为 PHY 送到 MDIO 模块的串行数据。
- 写操作时，为 MDIO 模块送到 PHY 的串行数据。
- IDLE：帧结束后的空闲状态。
此时 MDIO 无源驱动，处于高阻状态，但一般用上拉电阻使其处在高电平。

16.4.3.2 Clause45 的数据帧格式

表16-2 MDIO 接口的 Clause45 数据帧格式

操作方式	数据帧格式							
	PRE	ST	OP	PRTAD	DEVAD	TA	ADDRES S/DATA (payload)	IDL E
Address	1.....1	00	00	PPPPP	EEEEEE	10	AAAAAA AAAAAA AAAA	Z
Write	1.....1	00	01	PPPPP	EEEEEE	10	DDDDDD DDDDDD DDDD	Z
Read	1.....1	00	11	PPPPP	EEEEEE	Z0	DDDDDD DDDDDD DDDD	Z
Post-read- increment- address	1.....1	00	10	PPPPP	EEEEEE	Z0	DDDDDD DDDDDD DDDD	Z

数据帧各域的含义如下：

- PRE：帧同步码，32bit 位宽。
PRE 是 32 个连续的“1”，用于器件的同步通信。但该域不是必须的，为了提高效率，有些 PHY 器件的 MDIO 操作就没有此域。
- ST：帧开始标志。MAC 输出“00”比特通知 PHY 表示帧操作开始。
- OP：帧操作码，2bit 位宽。
 - 00：表示此帧为一次写地址操作，payload 中的内容为要访问的寄存器的地址。
 - 01：表示此帧为一次写数据操作，payload 中的内容为要写入的数据。
 - 11：表示此帧为一次读数据操作，payload 中的内容为读出的数据。



- 10：表示此帧为地址自增读数据操作，payload 中的内容为读出的数据。
- PRTAD：PHY 器件地址，5bit 位宽。
每个 PHY 器件都把自己的地址与这 5bit 进行比较，若匹配则响应后面的操作；若不匹配，则忽略后面的操作。
5bit 决定了一个 MDIO 接口最多只能支持 32 个 PHY 器件的配置。
- DEVAD：PHY 器件中的设备地址，5bit 位宽。
支持 IEEE802.3 Clause45 章的 1 个 PHY 器件内部可以支持最多 32 个设备，这个域就是表示 PHY 器件内部的设备地址。
- TA：状态转换域，2bit 位宽。
 - 读数据或者地址自增读数据操作时，第 1bit 由 MDIO 送出高阻态，第 2bit 由 PHY 给 MDIO 信号置 0。
 - 写地址或者写数据操作时，MDIO 信号由 MDIO 模块控制，连续输出 “10” 两个 bit。
- DATA：帧的寄存器数据域，16bit 位宽。
 - 读数据或者地址自增读数据操作时，为 PHY 送到 MDIO 模块的串行读数据。
 - 写地址操作时，为 MDIO 模块发出的要访问的串行某个寄存器地址。
 - 写数据操作时，为 MDIO 模块发出的串行写数据。
- IDLE：帧结束后的空闲状态。
此时 MDIO 无源驱动，处于高阻状态，但一般用上拉电阻使其处在高电平。

16.4.4 接口时钟频率配置

MDIO 接口时钟频率可通过配置寄存器 MDIO_clk_sel 选择接口频率，Hi3093 中 MDIO 支持 4 种频率，分别为 250KHz、1.667MHz、2.5MHz。支持的频率如表 16-3 所示。

表16-3 MDIO 接口时钟频率

工作时钟	分频比		
	40 分频	60 分频	400 分频
100MHz（默认）	2.5MHz	1.667MHz	250kHz
75MHz	1.875MHz	1.25MHz	187.5kHz



16.4.5 中断

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

16.4.6 Clause22 章和 Clause45 章模式配置

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

16.4.7 PHY 状态自动查询

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

16.4.8 初始化

16.4.8.1 MDIO 管脚复用配置

参考《Hi3093 硬件设计指南》管脚复用相关章节。

16.4.8.2 模块初始化

MDIO 初始化步骤如下：

- 步骤 1 配置寄存器 MDIO_COMMAND 选择接口的时钟。
- 步骤 2 配置寄存器 MDIO_COMMAND 选择接口模式为 GE 模式。
- 步骤 3 配置寄存器 MDIO_AUTO_CFG_0 到 MDIO_AUTO_CFG_2，配置自动查询模式下 PHY 地址值，计划扫描寄存器的地址（每个 PHY 支持 2 个寄存器）以及 bit 使能。
- 步骤 4 配置寄存器 MDIO_COMMAND [Mdio_AutoScan]，配置是否使能自动查询功能。

----结束



16.4.8.3 复位

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

- 步骤 1 配置寄存器 SC_MDIO_RESET_REQ[IP_INDEX]为 1，复位 MDIO[IP_INDEX]。
- 步骤 2 查询寄存器 SC_MDIO_RESET_ST[IP_INDEX]为 1，表示 MDIO[IP_INDEX]处于复位状态。
- 步骤 3 配置寄存器 SC_MDIO_RESET_DREQ[IP_INDEX]为 1，解复位 MDIO[IP_INDEX]。
- 步骤 4 查询寄存器 SC_MDIO_RESET_ST[IP_INDEX]为 0，表示 MDIO[IP_INDEX]处于解复位状态。

----结束

16.4.9 对接 PHY 的配置

16.4.9.1 Clause22 PHY 寄存器读操作

通过配置相关寄存器可实现对 PHY 外部寄存器的读访问，具体配置说明如下：

- 步骤 1 配置寄存器 MDIO_COMMAND[Mdio_Prtad]，配置相应的外部 PHY 地址。
- 步骤 2 配置寄存器 MDIO_COMMAND [Mdio_Devad]，配置相应的寄存器地址。
- 步骤 3 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x01，选择接口模式为 GE 模式。
- 步骤 4 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x10，表示读操作。
- 步骤 5 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1，启动读操作。
- 步骤 6 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0，如果为 0 表示读操作正常完成，进入下个步骤，不是 0 则继续查询。
- 步骤 7 查询寄存器 MDIO_RDATA_REG[Mdio_Rdata]，MDIO 读访问的读数据。

----结束

说明

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



16.4.9.2 Clause22 PHY 寄存器写操作

通过配置相关寄存器可实现对 PHY 内部寄存器的写访问，具体配置说明如下：

- 步骤 1 配置寄存器 MDIO_WDATA，配置需要写入 PHY 寄存器的数值。
- 步骤 2 配置寄存器 MDIO_COMMAND[Mdio_Prtad]，配置相应的外部 PHY 地址。
- 步骤 3 配置寄存器 MDIO_COMMAND [Mdio_Devad]，配置相应的寄存器地址。
- 步骤 4 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x01，选择接口模式为 GE 模式。
- 步骤 5 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x01，表示写操作。
- 步骤 6 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1，启动写操作。
- 步骤 7 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0，如果为 0 表示写地址操作正常完成，结束流程。不是 0 则在[步骤 7](#) 继续查询。

----结束

说明

上述步骤可以缩减为 3 个步骤，[步骤 2~步骤 6](#) 都是对同一个寄存器进行配置的，可以一次配置完成。

16.4.9.3 Clause45 PHY 寄存器读数据操作

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

- 步骤 1 配置寄存器 MDIO_ADDR [Mdio_Address]，配置需要读取的 PHY 中 DEV 中寄存器的地址。
- 步骤 2 配置寄存器 MDIO_COMMAND[Mdio_Prtad]，配置相应的外部 PHY 地址。
- 步骤 3 配置寄存器 MDIO_COMMAND [Mdio_Devad]，配置相应的设备地址。
- 步骤 4 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x00，选择接口模式为 XGE 模式。
- 步骤 5 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x00，表示写地址操作。
- 步骤 6 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1，启动写地址操作。
- 步骤 7 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0，如果为 0 表示写地址操作正常完成，进入[步骤 8](#)，不是 0 则在[步骤 7](#) 继续查询。



- 步骤 8 配置寄存器 MDIO_COMMAND[Mdio_Prtad], 配置相应的外部 PHY 地址。
- 步骤 9 配置寄存器 MDIO_COMMAND [Mdio_Devad], 配置相应的设备地址。
- 步骤 10 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x00, 选择接口模式为 XGE 模式。
- 步骤 11 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x11, 表示读操作。
- 步骤 12 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1, 启动读操作。
- 步骤 13 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0, 如果为 0 表示读操作正常完成, 进入[步骤 14](#), 不是 0 则继续查询。
- 步骤 14 查询寄存器 MDIO_RDATA_REG[Mdio_Rdata], MDIO 读访问的读数据。

----结束

说明

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

16.4.9.4 Clause45 PHY 寄存器写数据操作

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

- 步骤 1 配置寄存器 MDIO_ADDR [Mdio_Address], 配置需要写的 PHY 中 DEV 中寄存器的地址。
- 步骤 2 配置寄存器 MDIO_COMMAND[Mdio_Prtad], 配置相应的外部 PHY 地址。
- 步骤 3 配置寄存器 MDIO_COMMAND [Mdio_Devad], 配置相应的设备地址。
- 步骤 4 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x00, 选择接口模式为 XGE 模式。
- 步骤 5 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x01, 表示写地址操作。
- 步骤 6 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1, 启动写地址操作。
- 步骤 7 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0, 如果为 0 表示写地址操作正常完成, 进入[步骤 8](#), 不是 0 则在[步骤 7](#) 继续查询。
- 步骤 8 配置寄存器 MDIO_WDATA [Mdio_Wdata], 配置需要写入 PHY 寄存器的数值。
- 步骤 9 配置寄存器 MDIO_COMMAND[Mdio_Prtad], 配置相应的外部 PHY 地址。



- 步骤 10 配置寄存器 MDIO_COMMAND [Mdio_Devad]，配置相应的寄存器地址。
- 步骤 11 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x00，选择接口模式为 XGE 模式。
- 步骤 12 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x01，表示写操作。
- 步骤 13 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1，启动写操作。
- 步骤 14 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0，如果为 0 表示写地址操作正常完成。

----结束

说明

上述步骤可以缩减为 6 个步骤，步骤 2~步骤 6 都是对同一个寄存器进行配置的，可以一次配置完成。步骤 9~步骤 13 都是对同一个寄存器进行配置的，可以一次配置完成。

16.4.9.5 Clause45 PHY 寄存器地址自增读数据操作

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

- 步骤 1 配置寄存器 MDIO_ADDR [Mdio_Address]，配置需要读取的 PHY 中 DEV 中寄存器的地址。
- 步骤 2 配置寄存器 MDIO_COMMAND[Mdio_Prtad]，配置相应的外部 PHY 地址。
- 步骤 3 配置寄存器 MDIO_COMMAND [Mdio_Devad]，配置相应的设备地址。
- 步骤 4 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x00，选择接口模式为 XGE 模式。
- 步骤 5 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x00，表示写地址操作。
- 步骤 6 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1，启动写地址操作。
- 步骤 7 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0，如果为 0 表示写地址操作正常完成，进入步骤 8，不是 0 则在步骤 7 继续查询。
- 步骤 8 配置寄存器 MDIO_COMMAND[Mdio_Prtad]，配置相应的外部 PHY 地址。
- 步骤 9 配置寄存器 MDIO_COMMAND [Mdio_Devad]，配置相应的设备地址。
- 步骤 10 配置寄存器 MDIO_COMMAND[Mdio_St]为 0x00，选择接口模式为 XGE 模式。



- 步骤 11 配置寄存器 MDIO_COMMAND [Mdio_Op]为 0x10, 表示地址自增读操作。
- 步骤 12 配置寄存器 MDIO_COMMAND[Mdio_Start]为 0x1, 启动地址自增读操作。
- 步骤 13 查询寄存器 MDIO_COMMAND[Mdio_Start]是否为 0, 如果为 0 表示读操作正常完成, 进入[步骤 14](#), 不是 0 则继续查询。
- 步骤 14 查询寄存器 MDIO_RDATA_REG[Mdio_Rdata], MDIO 读访问的读数据。
- 步骤 15 继续读取下一个寄存器地址 (地址为之前访问的寄存器值+1) 的数据, 从[步骤 8](#) 开始循环到[步骤 15](#)。

----结束

说明

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



17 I2C

17.1 概述

I2C (The Inter-Integrated Circuit) 控制器实现标准 I2C 主设备功能，兼容 Philips I2C 总线协议，可完成对 I2C 总线上的从设备的数据发送和接收。

Hi3093 提供了 16 个非安全 I2C 控制器和 1 个安全 I2C 控制器。每个 I2C 包括独立的地址和中断线。

须知

安全 I2C 控制器是仅由安全核控制访问，其使用方法和其他 I2C 控制器相同；

安全 I2C 和其他安全接口存在复用，具体使用请参考《Hi3093 硬件设计指南》；

非安全 I2C 对外提供了 17 个 I2C port，因此存在 16 路非安全和 17 个 I2C port 的多路选择控制，具体使用请参考《Hi3093 硬件设计指南》。

17.2 关键术语

表17-1 I2C 模块关键术语描述

术语	描述
I2C	Inter-Integrated Circuit, I2C 总线是由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。是微电子通信控制领域广泛采用的一种总线标准。



17.3 特性

I2C 控制器具有以下特点：

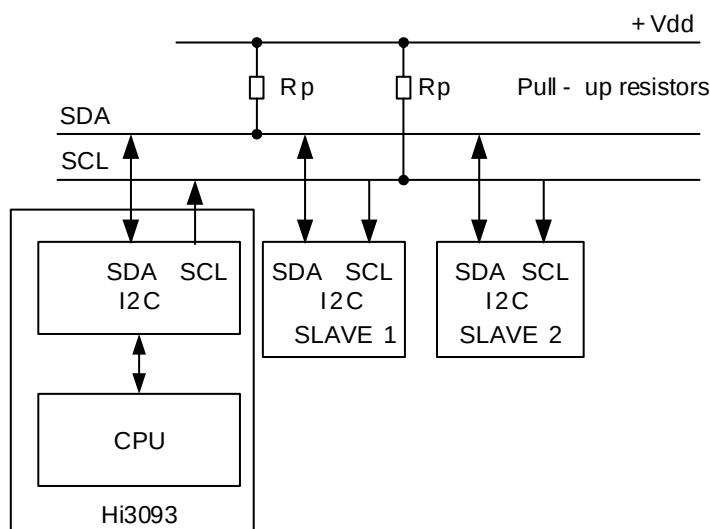
- 支持标准 I2C 总线协议。
- 支持主设备和从设备操作。
- 支持可编程时钟，可实现不同通讯速率（100Kbps/400Kbps）。
- 支持 DMA 数据搬运，其中安全 I2C 不支持 DMA 数据搬运。
- 支持中断和查询两种数据搬运方式。
- 支持 FIFO 深度 64 Byte。
- 支持防挂死功能。

17.4 功能描述

17.4.1 典型应用

I2C 总线典型应用电路如图 17-1 所示。

图17-1 I2C 典型应用电路原理图





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

17.4.2 功能原理

17.4.2.1 数据帧格式

一次典型的主模式 I2C 数据传输主要包括起始信号、从地址发送、数据传送、结束信号等部分。其时序如图 17-2 所示，数据帧格式如图 17-3 所示。

图17-2 I2C 数据传输时序图

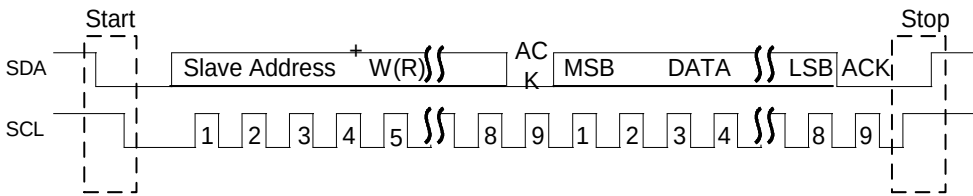
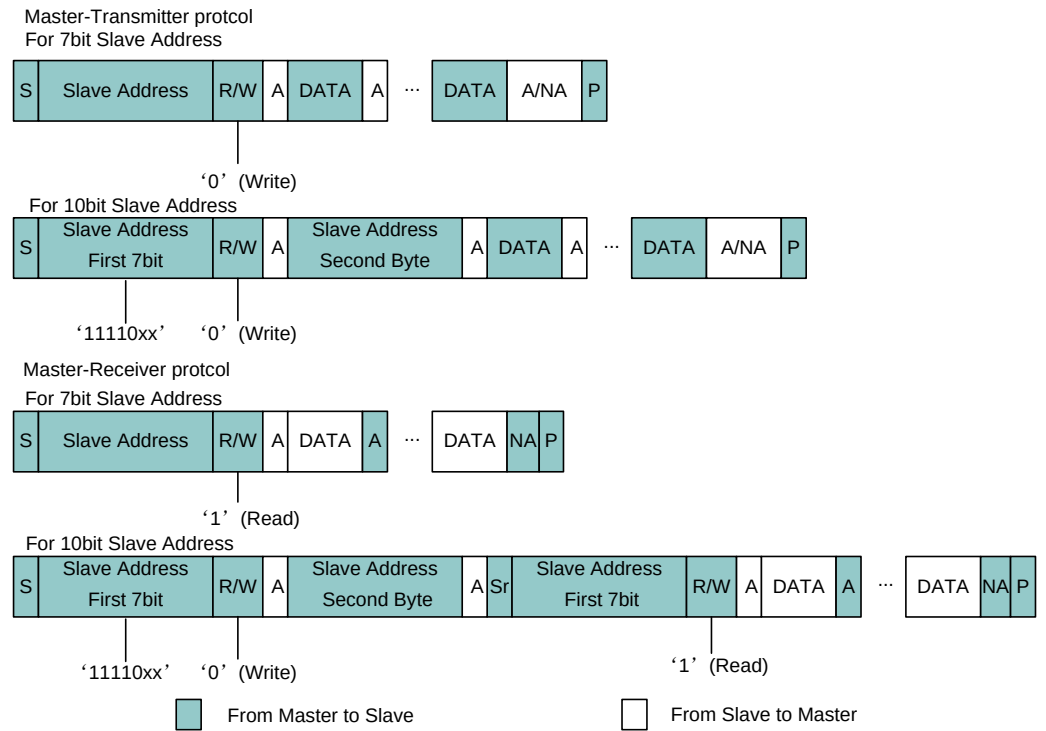


图17-3 I2C 数据传输帧格式图





说明

A: Acknowledge (SDALow)

NA: NoAcknowledge (SDAHigh)

S: StartCondition

Sr: RepeatedStartCondition

P: StopCondition

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

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

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

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

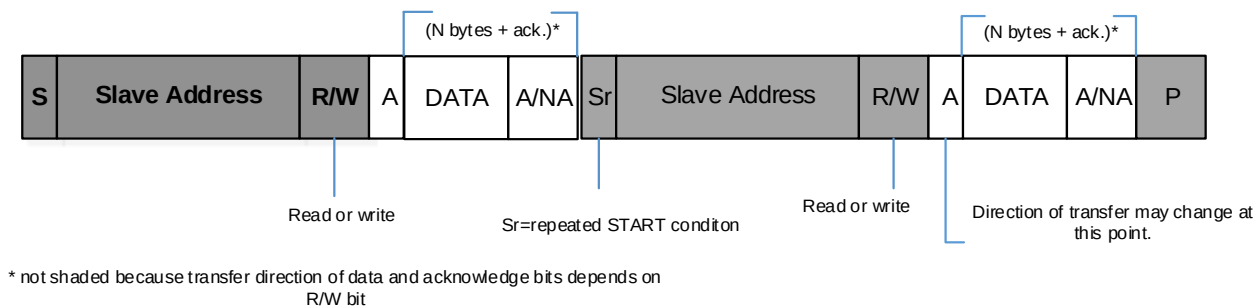
17.4.2.2 复合数据传输格式

I2C master 收发数据过程中，修改传输方向 (读写)，修改 Slave 地址，修改地址模式从 7bit 到 10bit 等情况，I2C 传输不需终止 Stop，直接 restart 改变后的传输，称为 Combined format，如图 17-4 所示。

Hi3093 I2C 仅支持读写操作的切换，不支持 Slave 地址或 7bit/10bit 地址模式的切换。



图17-4 I2C 复合数据传输格式



17.5 工作方式

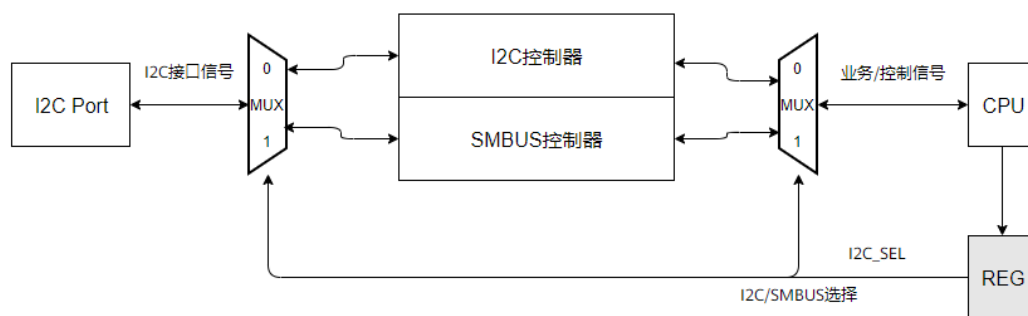
17.5.1 管脚复用配置

参考《Hi3093 硬件设计指南》管脚复用相关章节。

17.5.2 I2C 和 SMBUS 模式选择

I2C 和 SMBUS 控制器完全共用寄存器地址和 I2C 管脚，通过 LP_SUBSYS 的 I2Cx_SEL 寄存器控制二者的切换，启动默认为 I2C 控制器，如图 17-5 所示。

图17-5 I2C 和 SMBUS 控制器复用关系示意图



须知

I2C 和 SMBUS 不能同时工作，用户需上电后就决定控制器为 I2C 还是 SMUBS。不支持工作中进行 I2C 和 SMBUS 的模式动态切换。



17.5.3 时钟门控

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

17.5.3.1 打开门控时钟

以 I2C0 为例，具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS 寄存器 LPSC_CLKEN_CTRL0.I2C0_clk_enb 为 1，即打开 I2C 控制器的时钟门控。
- 步骤 2 读取 LP_SUBSYS 寄存器 LPSC_CLKSTA_CTRL0.I2C0_clk_status，获取当前时钟门控信息。
- 步骤 3 若为 I2C0_clk_status 为 0，则回到[步骤 1](#)；若为 1，则表示 I2C 控制器时钟已经打开，结束操作。

----结束

17.5.3.2 关闭门控时钟

以 I2C0 为例，具体配置步骤如下：

- 步骤 1 读 I2C0 的寄存器 I2C_STATUS，若 I2C_STATUS bit[0]为 0，则向 I2C_ENABLE bit[0]写 0，进入[步骤 2](#)；若 I2C_STATUS bit[0]为 1，则延时等待，返回[步骤 1](#)。
- 步骤 2 读取 LP_SUBSYS 寄存器 LPSC_CLKSTA_CTRL0.I2C0_clk_status，获取当前时钟门控信息。如果 I2C0_clk_status 为 1，则进入[步骤 3](#)；如果为 0，则表示 I2C 控制器时钟已经关闭，直接跳过下面的步骤，结束操作。
- 步骤 3 配置 LP_SUBSYS 寄存器 LPSC_CLKDIS_CTRL0.I2C0_clk_dis 为 0，即关闭 I2C 控制器模块的时钟门控。跳转[步骤 2](#)。

-----结束

说明

上述配置对应 I2C0~I2C15 均有对应的寄存器配置。



17.5.4 时钟配置

I2C Master 模式下需要通过配置 I2C_SS_SCL_HCNT、I2C_SS_SCL_LCNT、I2C_FS_SCL_HCNT、I2C_FS_SCL_LCNT 等寄存器可以设置标准模式和快速模式下 I2C 总线 SCL 信号的高低电平宽度，即相对于 I2C 工作时钟的周期数，其配置值的计算公式分别为： $*_HCNT = T_{scl_h} \times F_{I2C} - 8$ 和 $*_LCNT = T_{scl_l} \times F_{I2C} - 1$ 。

说明

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

I2C_SS_SCL_HCNT、I2C_SS_SCL_LCNT、I2C_FS_SCL_HCNT 和 I2C_FS_SCL_LCNT 典型配置值如表 17-2~表 17-5 所示。

表17-2 I2C_SS_SCL_HCNT 典型配置值

I2C 总线速率 (kbps)	I2C 工作时钟 (MHz)	SCL 高电平要求最小宽度 (μs)	SCL 高电平建议宽度 (μs)	IC_SS_SCL_HCNT (Cycle)
100	100	4.0	4	0x188

表17-3 IC_SS_SCL_LCNT 典型配置值

I2C 总线速率 (kbps)	I2C 工作时钟 (MHz)	SCL 低电平要求最小宽度 (μs)	SCL 低电平建议宽度 (μs)	IC_SS_SCL_LCNT (Cycle)
100	100	4.7	6	0x1D5



表17-4 IC_FS_SCL_HCNT 典型配置值

I2C 总线速率 (kbps)	I2C 工作时钟 (MHz)	SCL 高电平要求最小宽度 (μs)	SCL 高电平建议宽度 (μs)	IC_FS_SCL_HCNT (Cycle)
400	100	0.6	0.9	0x34

表17-5 IC_FS_SCL_LCNT 典型配置值

I2C 总线速率 (kbps)	I2C 工作时钟 (MHz)	SCL 低电平要求最小宽度 (μs)	SCL 低电平建议宽度 (μs)	IC_FS_SCL_LCNT (Cycle)
400	100	1.3	1.6	0x81

17.5.5 软复位

对接器件容许的情况下，可以对 I2C 控制器进行软复位，不会对系统造成影响。通过配置系统控制器实现对 I2C 控制器工作逻辑的单独软复位。

复位后仅 I2C 控制器接收发送部分逻辑复位，各寄存器的值均保持不变。

17.5.5.1 复位操作

以 I2C0 为例，具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS 寄存器 LPSC_SRST_REQ0. I2C0_srst_req 为 1，发送软复位请求。
- 步骤 2 查询 LP_SUBSYS 寄存器 LPSC_RESET_ST0. I2C0_srst_st，如果为 1，表示 I2C 已经进入软复位。如果为 0，则回到[步骤 1](#)。

----结束

17.5.5.2 复位撤离

以 I2C0 为例，具体配置步骤如下：



- 步骤 1 配置 LP_SUBSYS 寄存器 LPSC_RESET_DREQ0.I2C0_srst_dreq 为 1 为 0，可实现对 I2C0 工作逻辑的单独软复位撤离。
- 步骤 2 查询 LP_SUBSYS 寄存器 LPSC_RESET_ST0/I2C0_srst_st，如果为 0，表示 I2C 已经释放软复位。如果为 1，则回到[步骤 1](#)。

----结束

说明

上述配置对应 I2C0，I2C1 ~ I2C15 操作相应寄存器即可。

17.5.6 中断或查询方式下的数据传输

说明

下面配置过程以 I2C0 为例，I2C1~I2C15 同理。

17.5.6.1 初始化

以 I2C0 为例，具体配置步骤如下：

- 步骤 1 配置 LP_SUBSYS 寄存器 LPSC_SMBUS_I2C_MUX.smbus_i2c_sel0 配置为 0，选择 I2C 控制器模式。
- 步骤 2 设定超时标志，检测 I2C_STATUS bit[0]（判断 I2C 控制器是否空闲），如果是 1，则延时等待，继续检测，直到 IC_STATUS bit[0]变为 0 或者超时，向 I2C_ENABLE bit[0]写 0，使 I2C 处于关闭状态。
- 步骤 3 配置 I2C_CON，选择传输速率模式等参数。
- 步骤 4 将对接器件的从设备地址写入 I2C_TAR bit[9:0]。
- 步骤 5 配置 I2C_SS_SCL_HCNT、I2C_SS_SCL_LCNT 等寄存器，设定相应速率下 I2C 总线时钟周期。
- 步骤 6 配置 I2C_RX_TL、I2C_TX_TL，设定相应的发送及接收 FIFO 阈值。
- 步骤 7 如果驱动程序采用中断方式则需设定 I2C_INTR_MASK，使能相应中断信号。
- 步骤 8 采用查询方式时应禁止产生相应中断信号。
- 步骤 9 向 I2C_ENABLE bit[0]写 1，使能 I2C，完成初始化配置。

----结束



17.5.6.2 数据发送

具体配置步骤如下：

步骤 1 将发送数据写入 I2C_DATA_CMD，启动数据发送。

步骤 2 查询方式下，进行连续数据发送时通过读取 I2C_STATUS 和 I2C_TXFLR 检测发送 FIFO 状态；中断方式下，根据相应中断状态位检测。在数据传送完成之前，既要保证发送 FIFO 中的数据没有溢出（否则会造成数据丢失），同时也要保证发送 FIFO 中始终有数据（否则 I2C 会认为该次传输已全部完成而发出结束信号）。

步骤 3 通过检测 I2C_STATUS bit[2]是否为 1，判断 I2C 是否完成全部数据发送。

----结束

17.5.6.3 数据接收

具体配置步骤如下：

步骤 1 将读命令（0x100）写入 I2C_DATA_CMD，启动数据接收。

步骤 2 进行连续数据接收时，首先要发送次数与接收数据数目相同的读命令（0x100）到 I2C_DATA_CMD（例如要接收 3 个数据，则需发送 3 次读命令到 I2C_DATA_CMD 寄存器）。在数据接收完成之前，既要保证发送 FIFO 中的数据（即读命令 0x100）没有溢出，又保证发送 FIFO 非空（否则 I2C 会认为没有新的数据需要接收而结束），另外在连续数据接收过程中还需要检测接收 FIFO 的状态（如果采用中断方式，则根据相应中断状态位检测），避免接收 FIFO 溢出。

步骤 3 通过检测 I2C_STATUS bit[4]是否为 1，判断 I2C 是否完成全部数据接收。

----结束

说明

Master 模式下，可 combined Formal 传输；即在一次发送和接收过程中修改传输方向。配置 RESTART_EN 为 1。

17.5.7 DMA 方式下的数据传输

17.5.7.1 初始化

具体配置步骤如下：



- 步骤 1 配置 I2C_SMBUS 选择开关, LP_SUBSYS 寄存器:
LPSC_SMBUS_I2C_MUX/smbus_I2C_selX (X=0~15) 配置为 0。
- 步骤 2 设定超时标志, 检测 I2C_STATUS bit[0] (判断 I2C 控制器是否空闲), 如果是 1, 则延时等待, 继续检测, 直到 I2C_STATUS bit[0]变为 0 或者超时, 然后向 I2C_ENABLE bit[0]写 0, 使 I2C 处于关闭状态。
- 步骤 3 写相应的配置值到 I2C_CON, 配置主模式以及传输速率模式等参数。
- 步骤 4 如果是主模式, 将对接器件的设备地址写入 I2C_TAR bit[9:0]。
- 步骤 5 如果是从模式, 将本器件地址写入 I2C_SAR 寄存器。
- 步骤 6 配置 I2C_SS_SCL_HCNT、I2C_SS_SCL_LCNT 等寄存器, 设定相应的 I2C 总线时钟周期。
- 步骤 7 配置 I2C_DMA_TDLR、I2C_DMA_RDLR, 设置 DMA 发送 FIFO 及接收 FIFO 阈值。
- 步骤 8 向 I2C_ENABLE bit[0]写 1, 使能 I2C。

----结束

17.5.7.2 数据发送

具体配置步骤如下:

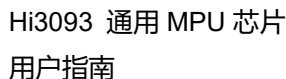
- 步骤 1 配置 DMA 数据通道, 包括数据传输源和目的地址、传输类型等参数。
- 步骤 2 向 I2C_DMA_CR bit[1]写 1, 使能 I2C 的 DMA 发送功能
- 步骤 3 读 I2C_INTR_STAT bit[4]寄存器 (发送 FIFO 低于阈值), 查询中断状态, 判断数据是否发送完成。如果完成则向 I2C_DMA_CR bit[1]写 0, 禁止 I2C 的 DMA 发送功能。

----结束

17.5.7.3 数据接收

数据接收包括数据发送和数据接收两步, 首先将读命令发送给 I2C 控制器, 然后从接收 FIFO 中读取接收数据, 具体配置步骤如下:

- 步骤 1 配置 DMA 数据通道, 包括数据传输源 (传输源内容包括读命令 0x100 等) 和目的地址、数据接收区地址、数据传输个数、传输类型等参数。
- 步骤 2 配置 I2C_DMA_CR bit[1:0]为 2'b11, 使能 I2C 的 DMA 发送和接收功能 (因数据接收时要先发送 0x100 到 I2C_DATA_CMD 寄存器, 所以要使能 DMA 发送功能)。

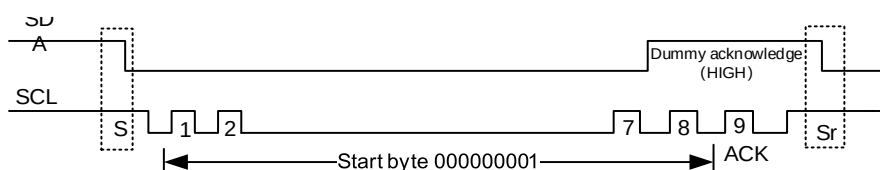


-----结束

17.5.8.1 异常挂死场景

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

图17-6 I2C START 传输时序



 说明

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

17.5.8.2 解决方案

防挂死操作使用的寄存器为 LP_SUBSYS_CSR.LPSC_I2C_CTRL0~LPSC_I2C_CTRL7, SEC_CSR_REG.SEC_I2C_CTRL。

以 I2C0 为例，具体配置步骤如下：

步骤1 配置 I2C0 clk mux sel 为 1, 表示软件接管 I2C SCL 的管脚行为。

步骤2 配置 I2C0 clk oe cfa 为 1, 配置 SCL 管脚为输出。

步骤3 配置 I2C0 scl cfg 为 1, 控制 I2C SCL 接口管脚输出高电平。



步骤 4 配置 I2C0_scl_cfg 为 0，控制 I2C SCL 接口管脚输出低电平。

步骤 5 重复步骤 3 和步骤 4 使得 I2C0 管脚电平不停翻转，同时软件插入等待时间，使得翻转频率为 10kHz，至少发送 9 个 SCL 时钟脉冲。

步骤 6 配置 I2C0_clk_mux_sel 为 0，恢复 I2C 管脚电平由 I2C 控制器来控制配置。

----结束



18 GMAC

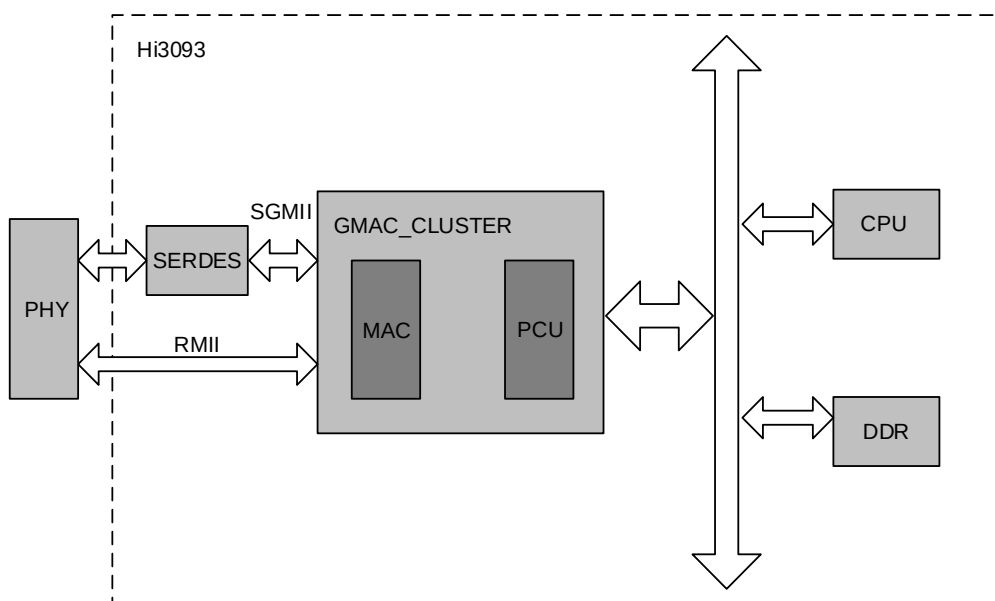
18.1 概述

Hi3093 支持 4 路网口数据通道，提供 SGMII/RMII 接口、数据链路层处理、报文缓存管理等功能模块。可实现网络报文接收、协议解析、缓存、发送等功能，网口在 Hi3093 中的位置如图 18-1 所示。

接收功能主要包括接收来自物理层（PHY）的数据，进行 CRC 校验，判断以太网数据帧的目的地址是否与本机地址相匹配，可以接收广播帧、流控帧等，并完成 3 层/4 层校验和检测，协议类型检测等。

发送功能主要包括接收来自应用层的数据，组装成数据帧，并根据配置决定是否添加 3 层和 4 层的校验和，是否产生 CRC 校验等，再通过物理层接口发送给 PHY 或 SerDes。

图18-1 网口（GMAC_CLUSTER）在 Hi3093 芯片中的位置





18.2 特性

18.2.1 链路层处理

- 支持 2 路 SGMII10M/100M/1000M 接口。
- 支持 2 路 RMII10M/100M 接口。
- 支持 SerDes1000M 接口。
- 通过 RMII 接口支持 NCSI 功能。

须知

GMAC0、1 支持 RMII；GMAC2-3 支持 SGMII。

- 支持符合 802.3 协议的帧发送处理功能：
 - 支持插入标准长度的前导码、SFD；
 - 支持根据配置决定是否对发送报文进行 PAD 填充，即把小于 64byte 的不带 FCS 的帧填充至 64byte；
 - 支持根据配置决定是否对发送报文添加 FCS；
 - 支持发送帧间隙可配置。
- 支持半/全双工从 PHY 侧至用户侧的帧接收功能：
 - 全双工支持 10/100MbpsRMII、10/100/1000MbpsSGMII 和 1000MbpsSerDes 接口；
 - 半双工支持 10/100MbpsRMII/SGMII 接口；
 - 支持识别流控帧，并将流控请求通知发送通道。
- 支持符合 802.3 协议的帧接收处理功能：
 - 支持剥离前导码和 SFD；
 - 支持剥离 PAD，剥离 PAD 的同时则必须剥离 FCS；如果不剥离 PAD，则可以配置决定是否剥离 FCS；
 - 支持帧错误标识功能，即帧接收过程中如果发现 FCS 校验 CRC 错误，则标识帧错误标志为有效；
 - 最多支持 6 个单播 MAC 地址匹配，过滤不匹配的帧；



- 支持 2 个带掩码的多播地址匹配，过滤不匹配的帧。
- 提供帧统计计数信息和状态信息：
 - 支持流控帧、广播帧、单播帧、组播帧、数据帧等收发数量统计信息；
 - 支持特殊帧或者错误帧收发统计信息，主要包括超长帧、超短帧、CRC 错误帧等。
 - 支持收发帧长统计，并对各个长度范围内收发帧进行个数统计。
- 支持 MAC 环回测试功能：
 - 支持内环回测试，即从应用侧经过发送 FIFO 后环回到接收 FIFO 然后再返回给应用侧；
 - 支持外环回测试，即从链路侧（PHY 侧）发出报文经过接收 FIFO 环回到发送 FIFO 后再发送给链路侧。
- 支持完备的功能配置选择：
 - 支持 MAC 源地址可配置；
 - 支持多播帧接收过滤地址可配置；
 - 支持 GMAC 的发送和接收使能可配置；
 - 支持流控响应使能可配置；
 - 支持接收帧 PAD 剥离操作使能可配置；
 - 支持接收帧 FCS 剥离操作使能可配置；
 - 支持发送帧添加 FCS 操作使能可配置；
 - 支持发送帧添加 PAD 操作使能可配置。
- 支持发送帧间隙长度可配置。

18.2.2 报文缓存管理

报文缓存管理主要包括链路侧和应用侧之间报文的缓存、丢包处理，协议解析，协议检查等功能。

18.2.2.1 发送部分

- 支持根据发送描述符的相关信息，主动搬运报文数据并进行发送。
- 支持两级发送描述符，第二级描述符为连续表项组织，存放在主存，可以实现分块报文发送。



- 支持 IP（三层）校验和计算（报文长度最大为 8032 Byte），按每包配置校验和计算使能，支持对 IP 版本号进行检查。
- 支持 TCP/UDP（四层）校验和计算（最大包长为 8032 Byte），按每包配置校验和计算使能（和 IP 校验和使能分离）。
- 支持对分片包的识别，分片包不计算网络 4 层协议校验和。
- 支持校验和计算时的协议检查，对于错误的报文不进行校验和计算，按原报文发送。
- 支持的协议检查（对于 IPv6 的分片包不支持检查四层头错误）包括：
 - IP、TCP/UDP 帧在 IP、TCP/UDP 头部中结束；
 - 帧长小于 TCP/UDP 的长度；
 - IP 头部长度域过短（小于 5）；
 - IP 报文长度域过短（小于 IP 头部长度）；
 - TCP 报文长度过短（IP 长度减去 IP 头部长度小于 20）；
 - UDP 报文长度过短（UDP 长度域小于 8）；
- 支持发送数据包时按每数据块设置读数据操作是否为 cachable 操作。
- 支持每包发送完成后向指定地址写 0，每数据块是否写 0 可配。

18.2.2.2 接收部分

- 支持的帧格式有 Ethernet、802.3、VLAN、VLANStack，支持 2 层 VLANStack。
- 支持 4 层的数据报文协议检查，检查结果写入报文描述符：
 - 4 层报文头部不正确（报文在 4 层头部中提前结束）；
 - UDP 长度与 IP 中长度不一致（UDP 长度大于 IP 净荷）；
 - UDP 长度域小于 8 Byte；
 - TCP 头部长度域小于 20 Byte；
 - TCP 头部长度域大于 IP 净荷长度；
 - 4 层校验和错误；
 - UDP 长度与 IP 中长度不一致；
 - 端口号为 0。
- 支持 3 层的数据报文协议检查，检查结果写入报文描述符：
 - IP 版本错误；



- IP 总长度小于 IP 头部长度;
- 报文头部长度域不正确, TCP/UDP 报文该长度域小于最小要求 (TCP 报文的净荷小于 20 Byte, UDP 报文的净荷小于 8 Byte);
- IPv4 报文头部的报文头部长度小于 5;
- 3 层校验和错误;
- 报文在 IP 头部中提前结束;
- IP 报文长度大于帧长;
- TTL 为 0;
- IP 存在选项字段 (IPv4 为存在选项字段, IPv4 为存在扩展头)。
- 支持 2 层的数据报文协议检查:
 - 寄存器配置超长包长度, 当报文大于该长度, 截断报文;
 - 接收 Error 信号有效;
 - 报文 CRC 错;
 - 报文在二层头部提前结束;
 - 2 层错误合并为 1 位写入报文描述符。
- 支持当接收 FIFO 满, 对继续到来的包进行丢弃处理并计数。
- 支持主动搬运数据到主存中。
- 支持接收方向缓存数据达到高水线时, 产生流控请求, 控制 MAC 发送流控帧, 水线可配置。
- 支持对申请到的接收缓存地址进行检测, 0 判断为无效地址。
- 支持以链表形式存放超过缓存大小的数据包。
- 支持第一个缓存头部空闲区域大小可配, 空闲区域为 32 Byte 的整数倍。
- 支持非第一个缓存头部空闲区域大小可配, 空闲区域为 32 Byte 的整数倍。
- 支持对报文进行协议解析, 结果写入第一个缓存头部空闲区域 (SKIP1) 之后, 大小为 32 Byte。
- 支持报文描述符后的一个空闲区域, 空闲区域大小可配, 为 32 Byte 的整数倍。
- 支持从寄存器指定偏移开始, 32 Byte 根据掩码提取进行匹配, 匹配结果在接收描述符中标记。
- 支持对 IPv4 的解析。
- 支持对 IPv6 的解析, 仅支持没有 IPv6 扩展头的 TCP/UDP 的解析。



- 支持接收方三层（IP）、四层（TCP/UDP）校验和计算，对分片包 TCP/UDP 不支持计算校验和。
- 支持对 IPV4 封装的 UDP 报文校验和全 0 的判断，该情况下不进行校验和检查。

须知

“支持报文解析跳过 SKIP 区域”的特性已删除，请软件人员不要关注相关配置。

当前 Hi3093 不支持 vlan 卸载，PCU 仅解析一层 vlan，多层 VLAN 需软件处理。

- 最小包长 64Byte，最大包长 7168 Byte。
- 支持报文过滤。
- 支持 VLAN 报文识别。

说明

关于长包接收性能：Hi3093 网口在不考虑流量影响时，GE 模式理论上发送最大能支持 64KB 报文，接收最大支持 32KB 报文。

18.3 关键术语

表18-1 GMAC 关键术语描述

术语	描述
GMAC	千兆以太网。
SGMII	Serial Gigabit Media Independent Interface，串行的千兆媒质无关接口。
RMII	Reduced Media Independent Interface，简化的媒质无关接口。

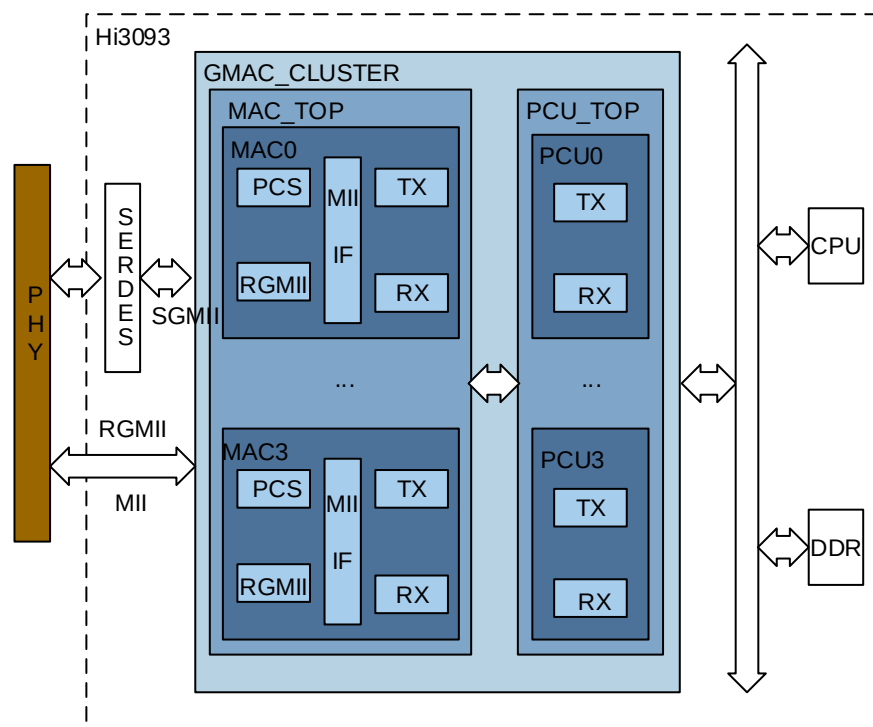
18.4 功能

18.4.1 结构框图

如图 18-2 所示，网口主要包括 MAC、PCU，各部分功能如下：

- MAC0-3 的功能主要是提供 SGMII 和 RMII 接口，与 PHY 对接，同时对接收或者发送的报文数据进行链路层处理。
- PCU0-3 的功能是作为硬件加速的一部分，对接收或者发送报文进行协议解析、缓存管理等。

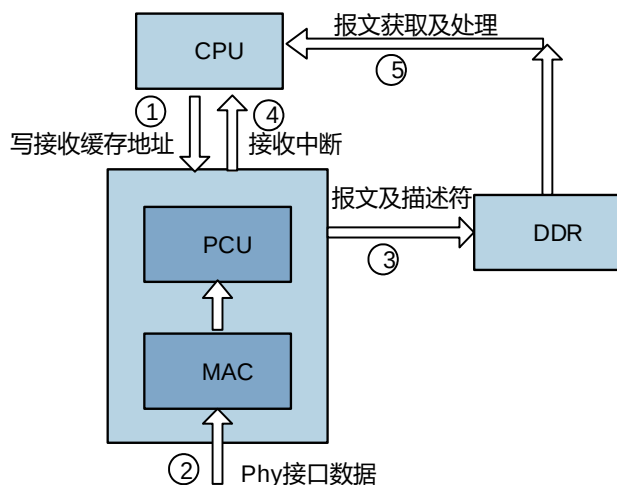
图18-2 网口逻辑框图



18.4.2 接收报文

网口接收报文的典型流程如[图 18-3](#)所示。

图18-3 接收报文流程示意图



步骤 1 软件通过 CPU 向 PCU 的接收缓存配置寄存器中写接收缓存地址。

步骤 2 接收从 Phy 收到的报文数据，接口模式需要软件初始化时配置。

步骤 3 报文经过 MAC、PCU 处理后，由 PCU 写入报文缓存地址，然后将报文解析结果即报文描述符随报文也一起写入。

步骤 4 当接收的数据包个数达到预定的数量或接收时间达到预定的时间，产生接收中断，通知 CPU。

步骤 5 CPU 接收到中断后，会从 DDR 中读取接收进来的数据并进行处理。

----结束

18.4.3 发送报文

网口发送的典型流程如图 18-4 所示。

步骤 1 软件通过 CPU 向 PCU 的发送配置 fifo 中写发送描述符。

步骤 2 PCU 根据发送描述符中描述的发送报文信息，从缓存中读取待发送报文。

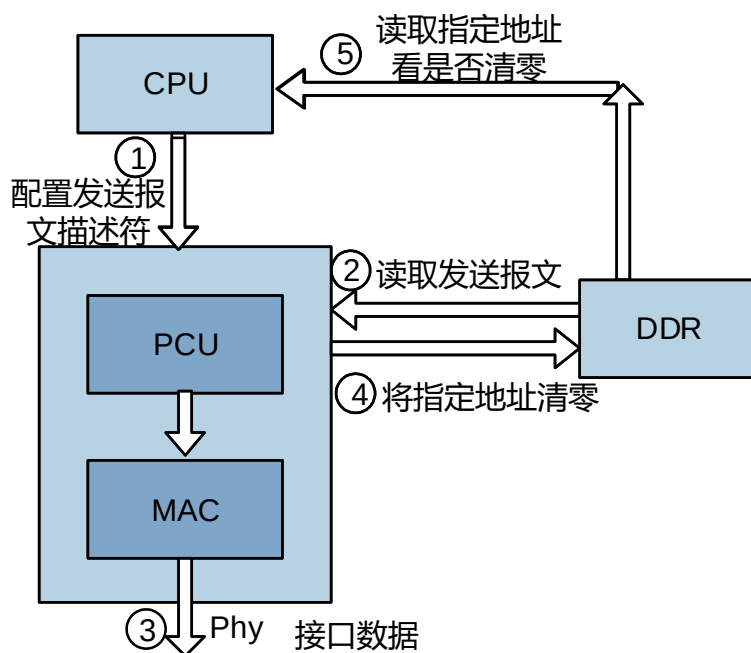
步骤 3 PCU 根据发送描述符中的配置信息，读取发送报文后经过 MAC 发送给 Phy。

步骤 4 PCU 完成报文发送后，会将发送描述符中的相应地址清零。

步骤 5 CPU 读取步骤 4 的清零地址，查看是否被清零，如果清零则表示报文发送完成。

----结束

图18-4 发送报文流程示意图



18.4.4 接收缓存管理

18.4.4.1 获取接收缓存地址

由软件写 PCU 的接收缓存配置寄存器，将接收缓存地址直接写入 PCU 接收缓存 FIFO。

须知

软件必须在配置 CF_CFG_FIFO_THRSLD 之后，CPU 再写缓存地址给 PCU，否则可能会出现无法预知的错误。

软件配置流程如下：

- 步骤 1 配置接收控制寄存器 RX_CTRL[cf_rx_get_addr_mode]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04F0，bit[12]）为 1，使能被动接收模式。
- 步骤 2 查询配置 FIFO 数据个数寄存器 CF_CFG_DATA_NUM[cf_rx_cfg_num]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x045C，bit[24:16]）是否小于等于 127，



只有当满足该条件时，才能继续下一步，否则一直查询，因为接收缓存地址 FIFO 最大只能存储 127 个接收缓存地址，如果超出则会导致接收缓存地址丢失。

步骤 3 将接收缓存地址写入接收配置 FIFO 地址寄存器 RX_CFF_ADDR（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04A0），然后重复步骤 2。

----结束

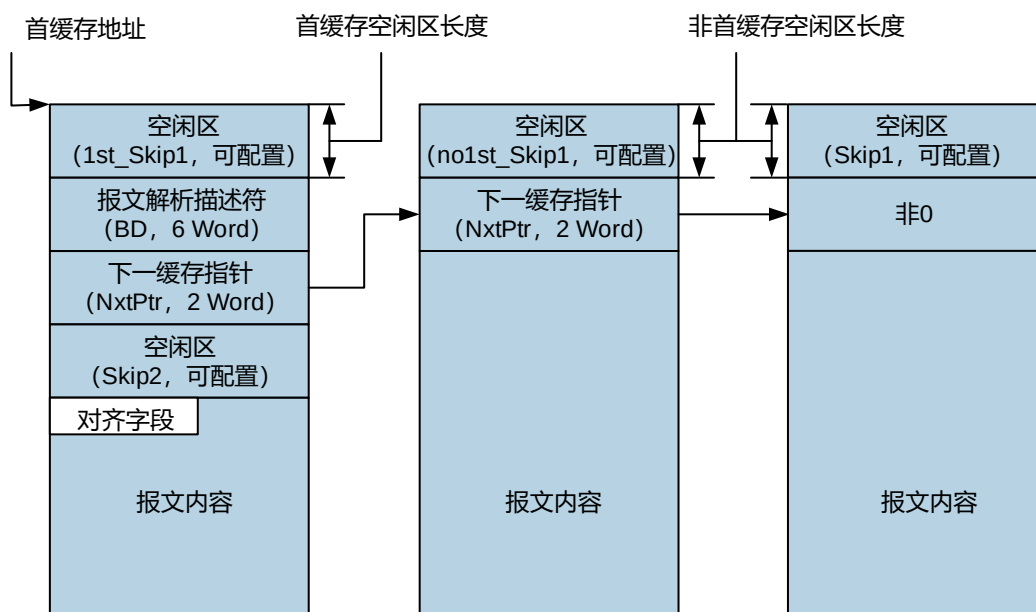
须知

缓存地址段不能跨 4G 边界，默认使用小于 4G 的地址段，若有特殊需要使用大于 4G 以上的地址段时，需要通过静态配置的方式实现，配置方式见缓存地址模式的切换。其他涉及缓存地址段的描述同此处，故不重复描述。

18.4.4.2 接收报文缓存格式

PCU 存储报文的格式如图 18-5 所示。

图18-5 接收报文存储格式



- 报文接收存储过程

当 PCU 接收数据 FIFO 中数据字数超过空水线，或 FIFO 中已经有完整帧，即启动数据搬运。从缓存地址 FIFO 读取缓存地址，然后跳过空闲区和描述符区（描



述符和下一缓存指针长度固定为 8Word，Skip 区长度由软件配置，因此硬件实际写报文内容的起始地址是缓存首地址开始偏移 skip1 加 skip2 空闲区长度再加 8 个 Word) 进行报文内容传输。如果一个缓存无法传输整帧，将再次申请一个缓存，继续存放数据，并在前一个缓存的“下一缓存地址”数据域描述后一个缓存的首地址。最后一个缓存的“下一缓存地址”为非 0，但无意义。在所有报文内容都完成传输后，硬件将报文描述符回写到首缓存地址开始的 8 个 Word。

• 接收报文存储格式

报文是以链表形式在内存中存放的，具体格式说明如下：

- 首缓存空闲区 (Skip1)，长度为寄存器可配，为 32 Byte 的整数倍；
- 报文解析描述符，长度为 24 Byte，6 个 Word；
- 下一缓存指针，2 个 Word，指向下一个缓存的“下一缓存指针”字段。报文解析描述符和下一缓存指针总共为 32 Byte，1 个 cacheline 长度；
- 缓存空闲区 2 (Skip2)，长度为寄存器可配，为 32 Byte 的整数倍；
- 对齐字段，寄存器可配，0~3 Byte；
- 报文内容。

当一块缓存无法放下整个数据报文时，下一缓存指针指向另一块缓存，该缓存格式为：

- 非首缓冲区，长度为寄存器可配，与首缓存空闲区长度独立；
- 下一缓存指针，2 个 Word，指向下一个缓存的“下一缓存指针”字段。如果该缓存为报文中最后一块缓存，该字段为非 0，但无意义；
- 报文内容；
- 下一缓存指针格式如表 18-2 和表 18-3 所示。

表18-2 Word1

成员范围	成员名称	成员说明
[31:25]	back	数据包下一块缓存中，数据指针所在 cacheline 距缓存头部的 cacheline 个数（cacheline 为 32 Byte）。
[24:16]	reserved	保留。



[15:0]	size	数据包下一块缓存中数据的长度，单位为 Byte。除最后一块缓存外，该长度指出对应缓存中有效的报文字节数。最后一块缓存中该长度为该缓存能够存放的最大报文字节数，可能大于真实存放的字节数。最后一块缓存的字节数需要软件自己算出。
--------	------	---

表18-3 Word2

成员范围	成员名称	成员说明
[31:0]	addr	下一缓存地址。

接收报文内容首字节对齐

软件通过配置接收控制寄存器 RX_CTRL[cf_rx_align_num]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04F0，bit[18:17]）可以控制接收报文写入内存时的字节对齐方式，例如配置值为 2，则报文内容从缓存首地址开始偏移 skip1 加 skip2 空闲区长度再加 8 个 Word，然后再偏移 2 个字节开始写入内存。

须知

以上 skip 长度配置的单位均为 cacheline 个数，即 32 Byte，例如如果配置值为 2，则表示 2 个 cacheline 长度，即 64 Byte。Skip 区长度可配置。

- 软件通过配置接收控制寄存器 RX_CTRL[cf_rxbuf_1st_skip_size]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04F0，bit[31:28]）可以控制首缓存空闲区 1（Skip1）的长度。
- 软件通过配置接收控制寄存器 RX_CTRL[cf_rxbuf_no_1st_skip_size]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04F0，bit[27:24]）可以控制非首缓存空闲区 1（Skip1）的长度。
- 软件通过配置接收控制寄存器 RX_CTRL[cf_rxbuf_1st_skip_size2]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04F0，bit[3:0]）可以控制首缓存空闲区 2（Skip2）的长度。



18.4.4.3 接收多种 Tag 生成模式

- 1. IP 报文非分片包
 - 1) 若是非分片的 IPv6 报文，且流标号不为 0 时，取流标号 20 bit 进行 16 位 CRC 运算得到 tag_crc[15:0]。
 - 2) 若是非分片的 IPv4 报文或者流标号为全 0 的非分片 IPv6 报文封装的 tcp/udp，采用下面取 5 元组信息方式生成 tag。

取 5 元组信息：协议类型，IP 源地址，IP 目的地址，TCP/UDP 源端口，TCP/UDP 目的端口，经过掩码选择相应的字段（对 IP 字段不需要按位掩码，直接对整个 IP 地址进行选择。没有选择的字段不参与运算），进行 16 位 CRC 运算得到 tag_crc[15:0]。

上述 IP 报文非分片包 tag[31:0]生成如表 18-4 所示。

表18-4 IP 报文非分片包的 tag

模式	IP 报文非分片包的 TAG			
tag_port_mode=0	tag[31:20]	tag[19:16]	tag[15:12]	tag[11:0]
	12'h0	port_num	port_num	tag_crc[11:0]
tag_port_mode=1	tag[31:24]	tag[23:16]	tag[15:12]	tag[11:0]
	8'h0	8'hff	4'hf	tag_crc[11:0]

- 2. 非 TCP/UDP 报文或分片包：
 - 1) 如果从 2 层解析，tag[31:0]生成如表 18-5 所示。

表18-5 从 2 层解析的非 TCP/UDP 报文或分片包的 TAG

模式	从 2 层解析的非 TCP/UDP 报文或分片包的 TAG			
tag_port_mode=0	tag[31:20]	tag[19:16]	tag[15:12]	tag[11:0]
	12'h0	port_num	port_num	L2 类型的低 12bit
tag_port_mode=1	tag[31:24]	tag[23:16]	tag[15:12]	tag[11:0]
	8'h0	8'hff	4'hf	L2 类型的低 12bit



2) 如果从 3 层解析，是 IP 报文 tag[31:0]的生成如表 18-6 所示，是非 IP 报文 tag[31:0]的生成如 0 所示。

表18-6 从 3 层解析的非 TCP/UDP 报文或分片包的 TAG

模式	从 3 层解析的 IP 封装的非 TCP/UDP 报文或分片包的 TAG				
tag_port_mode=0	tag[31:20]	tag[19:16]	tag[15:12]	tag[11:8]	tag[7:0]
	12'h0	port_num	port_num	4'h00	L3 协议类型 8bit
tag_port_mode=1	tag[31:24]	tag[23:16]	tag[15:12]	tag[11:8]	tag[7:0]
	8'h0	8'hff	4'hf	4'h0	L3 协议类型 8bit

表18-7 从 3 层解析的非 IP 封装的非 TCP/UDP 报文或分片包的 TAG

模式	从 3 层解析的非 IP 封装的非 TCP/UDP 报文或分片包的 TAG			
tag_port_mode=0	tag[31:20]	tag[19:16]	tag[15:12]	tag[11:0]
	12'h0	port_num	port_num	tag_def[11:0], 配置的默认值 16bit 的低 12bit
tag_port_mode=1	tag[31:24]	tag[23:16]	tag[15:12]	tag[11:0]
	8'h0	8'hff	4'hf	tag_def[11:0], 配置的默认值 16bit 的低 12bit

• 提取方式

从寄存器配置的偏移地址开始，用 64 位的掩码对报文的 64 Byte 进行提取，提取的数据进行 CRC16 运算得到 tag_crc[15:0]。tag[31:0]的生成如表 18-8 所示。



表18-8 提取方式计算得到的 Tag

模式	提取方式计算得到 Tag			
tag_port_mode=0	tag[31:20]	tag[19:16]	tag[15:12]	tag[11:0]
	12'h0	port_num	port_num	tag_crc[11:0]
tag_port_mode=1	tag[31:24]	tag[23:16]	tag[15:12]	tag[11:0]
	8'h0	8'hff	4'hf	tag_crc[11:0]

• 默认值方式

当采用非指令头模式，寄存器配置为不解析，采用默认方式生成 tag[31:0]。

默认方式生成 tag[31:0]如表 18-9 所示。

表18-9 默认方式的 TAG

模式	默认方式的 TAG			
tag_port_mode=0	tag[31:20]	tag[19:16]	tag[15:12]	tag[11:0]
	12'h0	port_num	port_num	tag_def[11:0], 配置的默认值 16bit 的低 12bit
tag_port_mode=1	tag[31:24]	tag[23:16]	tag[15:12]	tag[11:0]
	8'h0	8'hff	4'hf	tag_def[11:0], 配置的默认值 16bit 的低 12bit

• 增强型 TAG 生成方式

- 支持对分片、非分片、SEC 等报文类型的识别，且可按照软件配置决定报文的各协议域是否参与 TAG 生成。
- SEC 报文的配置寄存器独立为一组，其他类型报文为一组。
- 对于 IPV6 协议，最多支持 5 级 NextHeader 或者最大 256 Bytes 长度（从 IPV6 协议头开始算），只对解出的域做 CRC 参与 TAG 计算，此外对超出解析范围的报文做统计计数。



- 下述表格描述了当软件配置各协议域是否参与 TAG 计算后（对应配置寄存器为 V2_TAG_MODE），各类型报文相应的协议域实际参与 TAG 计算的情况。

说明

可配表示对于当前类型报文，支持该协议域是否参与 TAG 生成可配置。

配置无效表示对于当前类型报文，由于不存在该协议域，因此即使软件配置该协议域参与 TAG 生成，实际并不会生效。

不可配表示对于当前类型报文，不存在该协议域是否参与 TAG 生成的可配寄存器。

表18-10 非分片 IPV4 报文 TAG

协议	描述				
三层 IPV4	非分片				
四层	IP-SEC	TCP	UDP		其他
五层	-	-	GTPU	其他	
Protocol	可配	可配	可配	可配	可配
SIP	可配	可配	可配	可配	可配
DIP	可配	可配	可配	可配	可配
SPORT	不可配	可配	可配	可配	配置无效
DPORT	不可配	可配	可配	可配	配置无效
TEID	不可配	配置无效	可配	配置无效	配置无效
SPI	可配	不可配	不可配	不可配	不可配

表18-11 非分片 IPV6 报文 TAG

协议	描述				
三层 IPV6	非分片				
四层	IP-SEC	TCP	UDP		其它
五层	-	-	GTPU	其他	-



协议	描述				
Flow Label	可配	可配	可配	可配	可配
Protocol	可配	可配	可配	可配	可配
SIP	可配	可配	可配	可配	可配
DIP	可配	可配	可配	可配	可配
SPORT	不可配	可配	可配	可配	配置无效
DPORT	不可配	可配	可配	可配	配置无效
TEID	不可配	配置无效	可配	配置无效	配置无效
SPI	可配	不可配	不可配	不可配	不可配
三层 其他 或只有二层	软件配置默认 TAG 值				

表18-12 分片 IPV4 TAG 生成规则模式一

协议	描述					
三层 IPV4	分片（模式一，同 SIP+ID）					
四层	首片 UDP（先至）		首片 TCP（先至）	非首片（先至）	其他（先至）	全部（先至）
五层	GTPU	其他	-	-	-	-
Protocol	可配	可配	可配	可配	可配	同 SIP+ID 同 TAG
SIP	可配	可配	可配	可配	可配	
DIP	可配	可配	可配	可配	可配	
SPORT	可配	可配	可配	配置无效	配置无效	
DPORT	可配	可配	可配	配置无效	配置无效	
TEID	可配	配置无效	配置无效	配置无效	配置无效	
SPI	不可配	不可配	不可配	不可配	不可配	不可配



表18-13 分片 IPV6 TAG 生成规则模式一

协议	描述					
三层 IPV6	分片					
四层	首片 UDP（先至）		首片 TCP（先至）	非首片（先至）	其他（先至）	全部（先至）
五层	GTPU	其他	-	-	-	-
Flow Label	可配	可配	可配	可配	可配	同 SIP+ID 同 TAG
Protocol	可配	可配	可配	可配	可配	
SIP	可配	可配	可配	可配	可配	
DIP	可配	可配	可配	可配	可配	
SPORT	可配	可配	可配	配置无效	配置无效	
DPORT	可配	可配	可配	配置无效	配置无效	
TEID	可配	配置无效	配置无效	配置无效	配置无效	不可配
SPI	不可配	不可配	不可配	不可配	不可配	
三层 其他	软件配置默认 TAG 值					

表18-14 分片 IPV4 TAG 生成规则模式二

协议	描述				
三层 IPV4	分片 (模式二, 解析)				
四层	首片 UDP	首片 TCP	首片 SEC	首片其他	非首片



协议	描述					
五层	GTPU	其他	-	-	-	-
Protocol	可配	可配	可配	可配	可配	可配
SIP	可配	可配	可配	可配	可配	可配
DIP	可配	可配	可配	可配	可配	可配
SPORT	可配	可配	可配	配置无效	配置无效	配置无效
DPORT	可配	可配	可配	配置无效	配置无效	配置无效
TEID	可配	配置无效	配置无效	配置无效	配置无效	配置无效
SPI	不可配	不可配	不可配	不可配	不可配	不可配

表18-15 分片 IPV6 TAG 生成规则模式二

协议	描述
三层 IPV6	分片



四层	首片 UDP		首片 TCP	首片 SEC	首片其他	非首片
五层	GTPU	其他	-	-	-	-
Flow Label	可配	可配	可配	可配	可配	可配
Protocol	可配	可配	可配	可配	可配	可配
SIP	可配	可配	可配	可配	可配	可配
DIP	可配	可配	可配	可配	可配	可配
SPORT	可配	可配	可配	配置无效	配置无效	配置无效
DPORT	可配	可配	可配	配置无效	配置无效	配置无效
TEID	可配	配置无效	配置无效	配置无效	配置无效	配置无效
SPI	不可配	不可配	不可配	不可配	不可配	不可配
三层其他	软件配置默认 TAG 值					



18.4.5 发送缓存管理

发送报文时，需要 CPU 向发送描述符 FIFO 缓存地址寄存器 TX_CFF_ADDR（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0488—0x0494）写入发送描述符。

每个发送描述符代表一个数据帧，描述符可指向一个内存中的列表，每个表项指向一个发送数据段；也可指向一个只有一个数据段的数据帧（由发送描述符中的 Sd 标志位标识）。前者的数据组织如图 18-6 所示，后者的数据组织如图 18-7 所示。

图18-6 多数据段报文发送数据组织

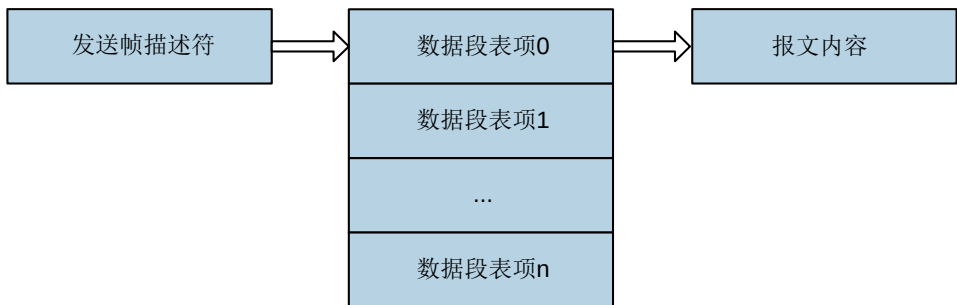
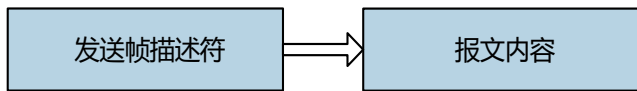


图18-7 单数据段报文发送数据组织



发送描述符占 4 个字，格式分别如表 18-16、表 18-17、表 18-18、表 18-19 所示。

表18-16 报文描述符 WORD0 格式

位	信号名称	描述
31	Reserved	保留。
30:26	IPoff	用来指明 IP 报文头在数据报中的起始偏移。
25	Sd	为 0 时，帧只包含一个数据段，直接指向数据段；为 1 时，帧包含多个数据段，指向数据段列表。
24:19	Ds	数据段个数，当 Sd=1 时有效。



位	信号名称	描述
18:3	Reserved	保留。
2	l3_cs	是否做 3 层 Checksum。
1	wb	Wb 为 1 时，当报文发送完成，向 WORD3 地址写 0。
0	l4_cs	是否做 4 层 Checksum。

表18-17 报文描述符 WORD1 格式

位	信号名称	描述
31:28	Reserved	保留。
27	I	读分配 L2cache。当 Sd=1 时，为数据表项所在缓存的属性。
26	Fm	发送完成释放缓存空间。当 Sd=1 时，为数据表项所在缓存的属性。
25:20	back	数据所在 cacheline 首地址与缓存首地址之间距离。当 Sd=1 时，为数据表项所在缓存的属性。
19:4	Len	发送长度。当 Sd=1 时，该字段无效。
3:0	Reserved	保留。

表18-18 报文描述符 WORD2 格式

位	信号名称	描述
31:0	ADDR[31:0]	地址 (当 Sd=1 时表示 Segmentlist 的首地址；当 Sd=0 时表示数据的首地址)。



表18-19 报文描述符 WORD3 格式

位	信号名称	描述
31:0	ADDR[31:0]	清 0 地址。

数据段表项格式如表 18-20、表 18-21 所示。

表18-20 WORD0

位	信号名称	描述
31:28	Reserved	保留。
27	I	读无效 L2 cache。
26	Fm	发送完成释放缓存空间。
25:20	Offset	数据所在 cache line 首地址与缓存首地址之间距离。
19-4	Len	发送长度。
3:0	pool	释放的 pool ID。

表18-21 WORD1

位	信号名称	描述
31:0	ADDR[31:0]	发送数据地址。

发送描述符 FIFO 深度为 256，可容纳 64 个描述符。通过查询配置 FIFO 数据个数寄存器 CF_CFF_DATA_NUM[cf_tx_cfg_num]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x045C，bit[8:0]）可以得到当前发送描述符 FIFO 中有效字个数。

须知

配置 FIFO 数据个数寄存器的状态值除以 4 表示当前的发送描述符个数（因为一个发送描述符占 4 个字），如果是 CPU 向 GMAC 添加发送描述符，需要在添加前查询当前 FIFO 个数是否还有 4 个字以上空位，否则可能会导致 FIFO 溢出。



18.4.6 接收报文解析

18.4.6.1 输入报文格式

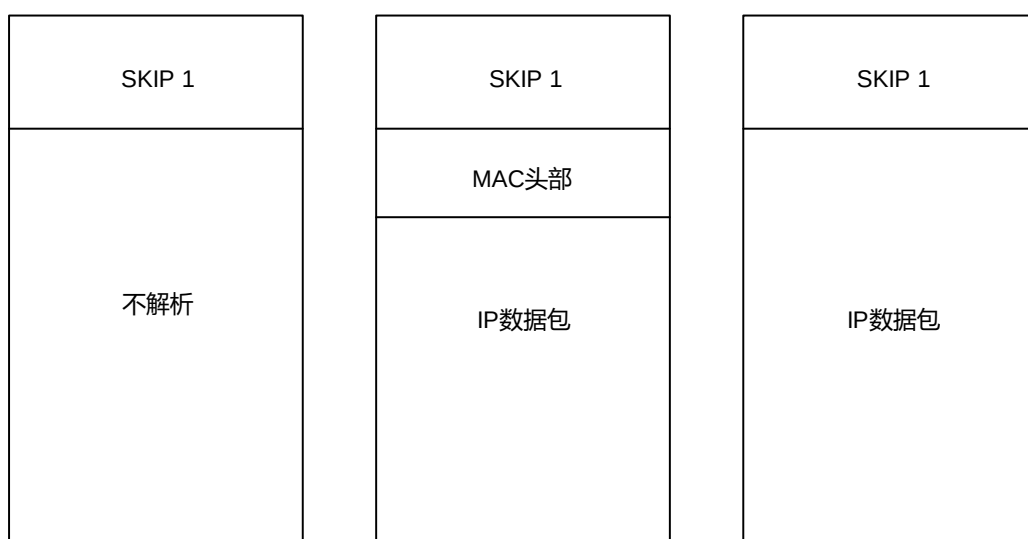
模块通过解析数据报文，产生两部分信息：协议分析结果和报文调度信息。通过报文协议解析生成。

当配置为非指令头模式，输入包格式有 3 种：

- 格式 1：SKIP1 大小寄存器可配，SKIP1 后面数据不解析。该格式中，协议分析结果由寄存器配置指出。
- 格式 2：SKIP1 大小寄存器可配，SKIP1 后面数据为第二层 MAC 数据。该格式中，协议分析结果由硬件分析报文生成。
- 格式 3：SKIP1 大小寄存器可配，SKIP1 后面数据为第三层 IP 数据。该格式中，协议分析结果由硬件分析报文生成。

这三种格式通过配置寄存器中的模式域指出，格式如[图 18-8](#) 所示。

图18-8 非指令头模式下输入报文格式



18.4.6.2 报文协议信息解析

当采用硬件解析报文协议时，配置为从 L2 开始解析模式下，能识别分析的 MAC 层协议如[图 18-9](#) 所示。



图18-9 硬件解析二层协议头

ethernet	6字节	6字节	2字节						
	目的地址	源地址	类型						
802.3	6字节	6字节	2字节	6字节	2字节				
	目的地址	源地址	<1501		类型				
ethernet vlan	6字节	6字节	2字节	2字节	2字节				
	目的地址	源地址	8100	VLAN	类型				
802.3 vlan	6字节	6字节	2字节	2字节	2字节	6字节	2字节		
	目的地址	源地址	8100	VLAN	<1501		类型		
ethernet vlan stack	6字节	6字节	2字节	2字节	2字节	2字节	2字节		
	目的地址	源地址	8100	VLAN	8100	VLAN	类型		
802.3 vlan stack	6字节	6字节	2字节	2字节	2字节	2字节	2字节	6字节	2字节
	目的地址	源地址	8100	VLAN	8100	VLAN	<1501		类型

协议分析结果 (decodefield) 为 6 个 Word，如表 18-22 ~ 表 18-27 所示。

表18-22 Word1

成员范围	成员名称	成员说明
[31:20]	reserve	保留。
[19:12]	Gen_id	该描述符的产生者编号，由 GMAC 寄存器配置。
[11:9]	Qos	服务质量。
[8:5]	Group	组号。
[4:3]	TT	保序类型。 00：不保序报文； 01：保序报文； 10：原子报文； 11：保留。
[2:0]	reserve	保留。



表18-23 Word2

成员范围	成员名称	成员说明
[31:0]	Tag	用于保序的 Tag 号。

表18-24 Word3

成员范围	成员名称	成员说明
[31:16]	Len	该数据包长度。
[15:12]	Port_num	数据包对应接收 GMAC 的物理端口号。
[11:9]	reserve	保留。
[8:0]	All_skip	输入格式中 skip1 长度+skip2 长度。

表18-25 Word4

成员范围	成员名称	成员说明
[31:24]	Buf_num	该数据包使用的缓存个数（如果数据报文需要不止一块缓存来存储，在每个缓存中要使用下一缓存指针将各个缓存串联起来）。
[23:16]	Ip_offset	报文头到 IP 头的偏移，单位为字节。
[15:0]	Vlan	Vlan 字段内容（当报文的 6 个字节的源地址和 6 个字节的地址之后的两个字节的值为 8100 时，表示是 vlan 报文）。



表18-26 Word5

成员范围	成员名称	成员说明
[31:30]	lp_tcp_udp	00: 非 IP 报文; 01: IP 报文; 10: Tcp 报文; 11: Udp 报文。
[29]	IPSE	Ipsec 报文。 0: Ipv4 头部协议类型字段或 Ipv6 头部下一条协议类型不为 50 或 51; 1: Ipv4 头部协议类型字段为 50 或 51, Ipv6 头部下一条协议类型为 50 或 51。
[28]	Ipv	Ipv4 或 Ipv6。 0: Ipv4; 1: Ipv6。
[27]	reserve	保留。
[26:23]	L4_error_code	0000: 正确帧; 0001: 4 层报文头部不正确, 包括: 报文在 4 层头部中提前结束; 0010: 长度错误; UDP 长度与 IP 中长度不一致 (UDP 长度大于 IP 净荷) ; UDP 长度域小于 8Byte; TCP 头部长度域小于 20Byte; TCP 头部长度域大于 IP 净荷长度; 0011: TCP/UDP 校验和出错; 0100: 端口号为 0; 其他: 保留。 注意: 当为分片包时, 该字段为 0, 无效。
[22]	Frag	该报文为分片包。



成员范围	成员名称	成员说明
[21]	Opt	IP 报文有选项字段（IPv4 为头部长度不为 5，小于 5 的情况在 l4_error_code 中指出；Ipv6 为存在扩展头）。
[20]	Ip_ver_err	IP 版本号非 4 和 6。
[19]	Brd_cst	广播报文。
[18]	Mul_cst	多播报文。
[17]	Arp	Arp 报文。
[16]	Rarp	Rarp 报文。
[15]	Icmp	ICMP 报文。
[14]	vlan_flag	Vlan 报文标记。
[13]	drop	缓存不足，该帧部分被丢弃。
[12:9]	L3_error_code	0000：正确帧； 0001：报文头部不正确，包括： 报文在 IP 头部中提前结束； IP 总长度小于 IP 头部长度； 报文头部长度域不正确，TCP/UDP 报文该长度域小于最小要求； Ipv4 报文头部的报文头部长度小于 5； 0010：3 层 checksum 错误； 0011：IP 报文长度大于帧长； 0100：TTL 为 0。
[8]	L2_error	当出现下面错误时，该位为 1： 寄存器配置超长包长度，当报文大于该长度，截断报文； 接收 error 信号有效； 报文 CRC 错。



成员范围	成员名称	成员说明
[7]	Key_match	报文头匹配正确。 0：匹配不正确； 1：匹配正确。
[6:5]	pm	报文的解析模式。 00：不解析报文； 01：从 L2 开始解析报文； 10：从 L3 开始解析报文； 11：保留。
[4:0]	Reserve	保留。

表18-27 Word6

成员范围	成员名称	成员说明
[31:25]	back	数据包第一块缓存中，数据指针所在 cacheline 距缓存头部的 cacheline 个数 (cacheline 为 32 Byte) 。
[24:16]	Reserved	保留。
[15:0]	size	数据包第一块缓存中有效数据的长度，单位为 Byte。

协议解析相关软件配置说明：

- 配置寄存器 RX_PKT_MODE[cf_instr_head_mode]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04F4，bit[31]）。为 0 表示非指令头模式，为 1 表示指令头模式。为非指令头模式时，解析信息由报文的协议生成。

须知

指令头模式已删除，请软件人员不要关注相关配置。



- 配置寄存器 RX_PKT_MODE[cf_skip_len] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F4, bit[31])。指明输入报文前 skip 区域的字节数, 即指明输入报文解析的起始位置。
- 配置寄存器 RX_PKT_MODE[cf_gen_id] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F4, bit[7: 0]), 指明该描述符产生者的 ID。
- 配置寄存器 RX_CTRL[cf_port_num] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[16:13]), 指明数据报文的物理端口号。

18.4.7 超长帧截断

18.4.7.1 发送方向超长帧截断

- 配置发送最大帧长寄存器 MAX_FRM_SIZE (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x003C, bit[15:0]) 值为 0x05EE, 设置发送数据帧的最大长度为 0x05EE 个字节。

表18-28 发送方向截断报文及统计计数相关说明 (1000M, MAX_FRM_SIZE 配置值为 N)

报文类型	CPU 发送的报文字节长度 M(无 CRC)	MAC 发送的报文字节长度 (有 CRC)	OCTETS_TRANSMITTED_BAD 统计值
VLAN 帧	$N+4 < M < N+10$	N+9	M+4
	$N+10 \leq M$		N+14
非 VLAN 帧	$N < M < N+6$	N+5	M+4
	$N+6 \leq M$		N+10

须知

OCTETS_TRANSMITTED_BAD (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0104, bit[14:0])。

- VLAN 帧允许的长度是该寄存器的配置值加上 4, 该寄存器最大可配值为 0xFFFFB, 大于则内部逻辑接收到 VLAN 帧时用该寄存器配置加 4 时内部逻辑会溢出异常。
- 最大长度的寄存器配置值必须是 4 Byte 的整数倍, 否则无法精确截断。



- GE 模式不考虑流量影响时，理论上发送支持 64K 以下长度报文，接受支持 32K 以下长度报文。

表18-29 发送方向截断报文及统计计数说明 (10/100M, MAX_FRM_SIZE 配置值为 N)

报文类型	CPU 发送的报文 字节长度 M(无 CRC)	MAC 发送的报文 字节长度 (有 CRC)	OCTETS_TRANSMITTE D _BAD 统计值
VLAN 帧	$M = N + 5$	N+9	N+9
	$M = N + 6$ 或 $N + 7$		N+10
	$M = N + 8$ 或 $N + 9$		N+11
	$M = N + 10$ 或 $N + 11$		N+12
	$M = N + 12$ 或 $N + 13$		N+13
	$N + 13 < M$		N+14
非 VLAN 帧	$M = N + 1$	N+5	N+5
	$M = N + 2$ 或 $N + 3$		N+6
	$M = N + 4$ 或 $N + 5$		N+7
	$M = N + 6$ 或 $N + 7$		N+8
	$M = N + 8$ 或 $N + 9$		N+9
	$N + 9 < M$		N+10

18.4.7.2 接收侧超长帧截断

接收的数据的最大长度可以通过寄存器配置，如果接收的数据长度大于配置值，则截断数据，并在接收描述符中将 drop 位置位，具体配置如下：

由软件配置接收超长限寄存器 MAX_FRAME_LEN (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0444, bit[14:0]), 设置接收数据帧的最大长度，此截断是在 PCU 实施的，因此不包括被 MAC 层剥离的长度。



18.4.8 PAD 剥离

802.3 帧在封装数据时，封装完的报文不足 64 Byte 会填充 PAD，Hi3093 支持这种 PAD 的剥离。

须知

VLAN 封装的帧及以太网帧无法剥离 PAD；剥离 PAD 时连 CRC 一起剥离，无法只剥离 PAD 不剥离 CRC。

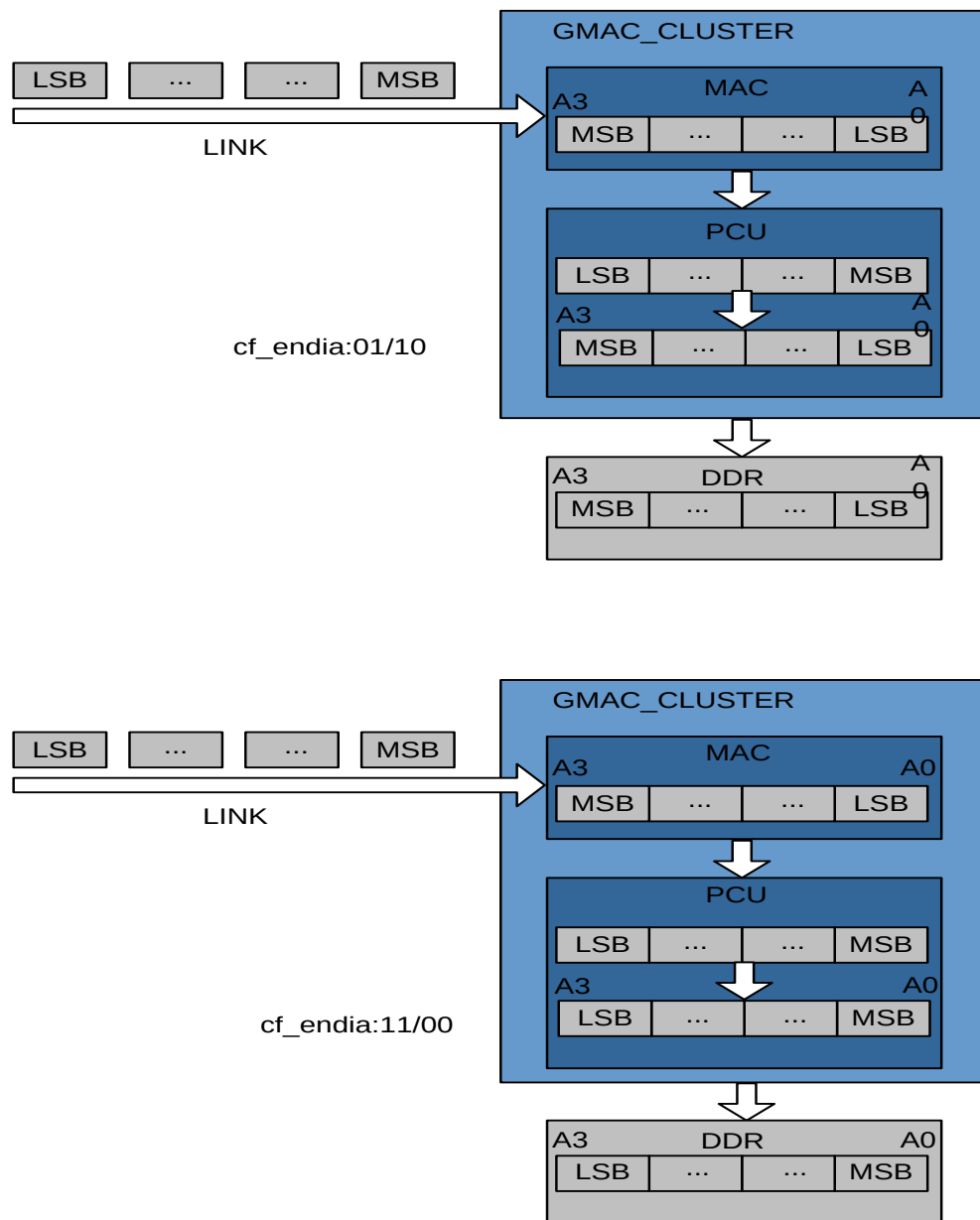
配置接收控制寄存器 RECV_CONTROL[strip_pad_en]（基地址 0xcd00000+1000*ch_num，偏移地址为 0x01E0，bit[3]）为 1。



18.4.8.1 报文内容大小端配置说明

接收报文

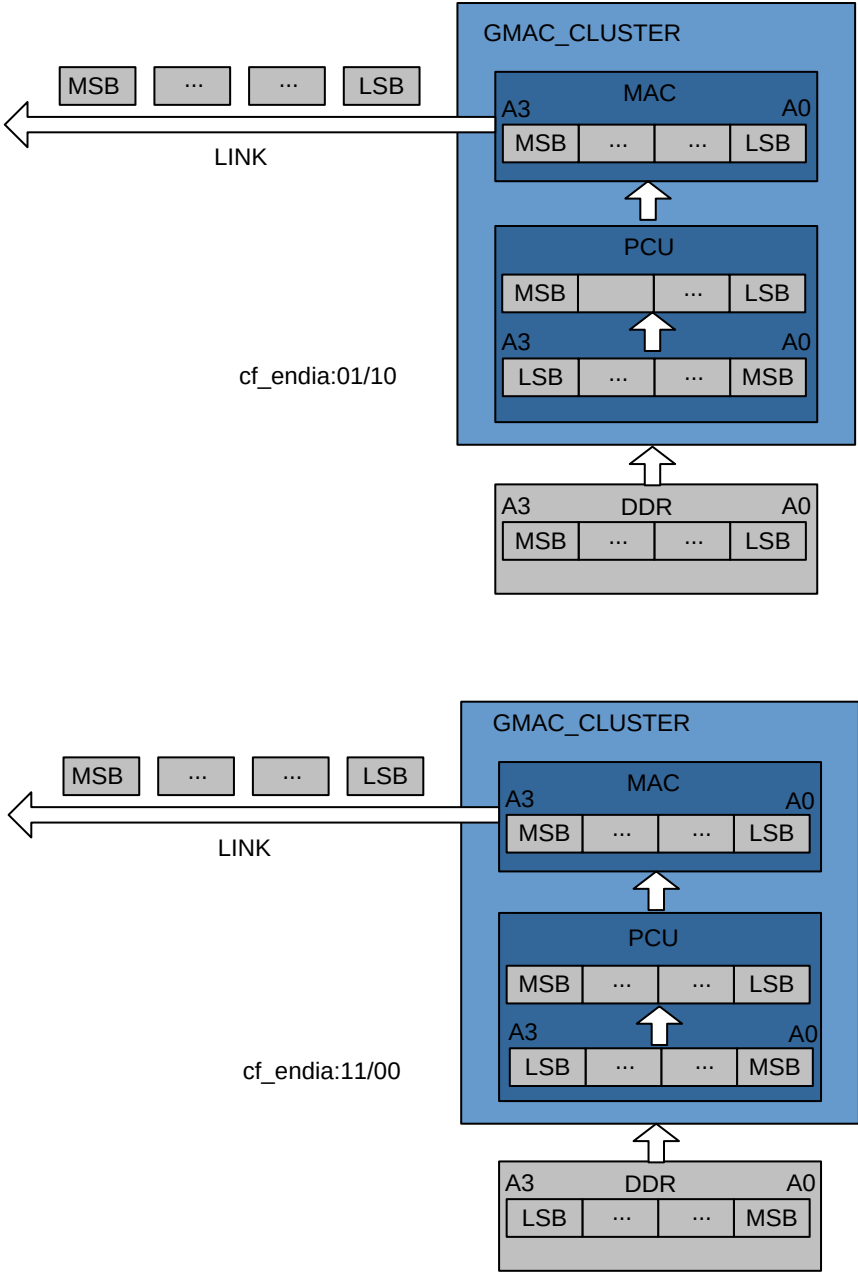
图18-10 接收报文内容大小端配置说明





发送报文

图18-11 发送报文内容大小端配置说明



说明

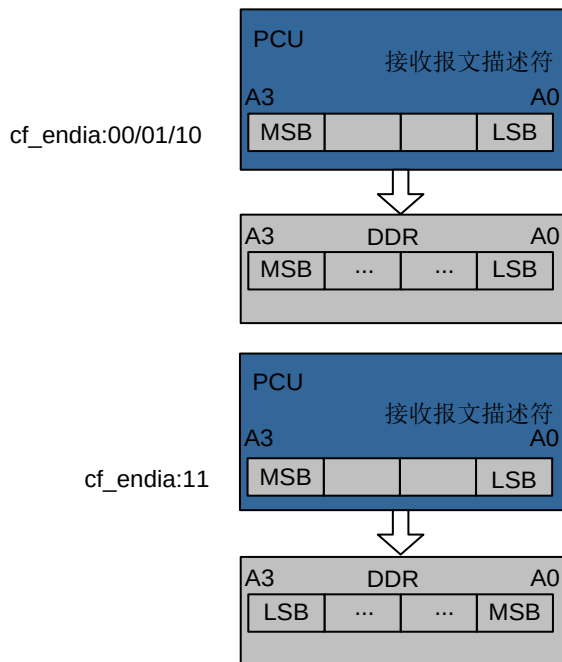
图 18-11 中 MSB (MostSignificantByte) 表示数据的高位, LSB (LeastSignificantByte) 表示数据的低位。A0 表示低位地址, A3 表示高位地址。



18.4.8.2 报文描述符大小端配置说明

接收报文

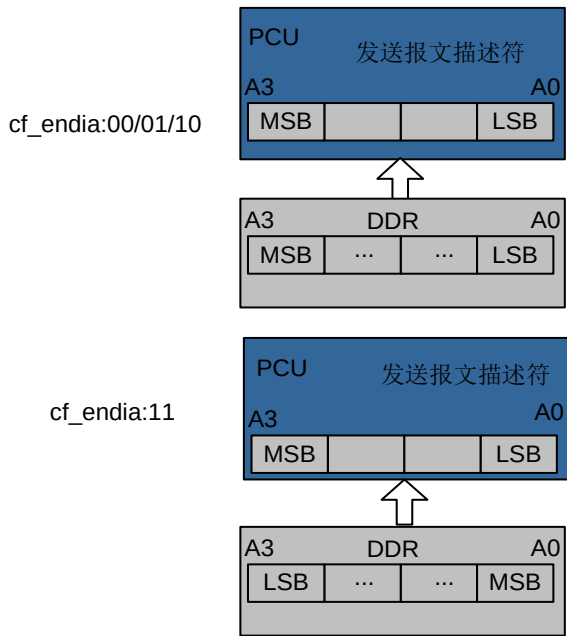
图18-12 接收报文描述符大小端配置说明





发送报文

图18-13 发送报文描述符大小端配置说明



18.4.9 总线一致性操作

18.4.9.1 Cacheable 操作

网口内部总线操作 Cache 类型说明如表 18-30 所示。

表18-30 网口内部总线操作 Cache 类型说明

总线传输类型	cache 类型
接收，将报文数据写入内存。	寄存器可配置。
发送，读出内存中报文数据。	发送描述符、寄存器组合配置。
发送完成后，对指定地址清零。	固定不可 cache。

接收报文写内存 Cacheable 操作配置说明

接收报文写内存 Cacheable 操作配置说明如下：



- 步骤 1 软件配置寄存器 BUS_CTRL[cf_rx_alloc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E8, bit[0]) 为 1。
- 步骤 2 软件配置寄存器 BUS_CTRL[huser] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E8, bit[5]) 为 1。
- 步骤 3 软件配置寄存器 BUS_CTRL[noncoh_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E8, bit[4]) 为 0, 上述寄存器组合配置作用是使能 Cacheable 操作。
- 步骤 4 软件配置寄存器 RX_CTRL[cf_cache_line_h]及 RX_CTRL[cf_cache_line_l] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[9:8]及 bit[6:4]), 配置值为软件需要的 Cachable 操作的报文长度, 单位是 cacheline 个数。

----结束

发送报文读内存 Cacheable 操作配置说明

发送报文读内存 Cacheable 操作配置说明如下:

- 步骤 5 软件配置寄存器 BUS_CTRL[cf_rx_alloc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E8, bit[0]) 为 1。
- 步骤 6 软件配置寄存器 BUS_CTRL[huser] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E8, bit[5]) 为 1。
- 步骤 7 软件配置寄存器 BUS_CTRL[noncoh_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E8, bit[4]) 为 0, 上述寄存器组合配置作用是使能 Cacheable 操作。
- 步骤 8 软件将发送描述符 WORD1[27] (发送描述符配置寄存器 TX_CFF_ADDR_1[cache], 基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x048C, bit[27]) 配置为 1, 上述寄存器组合配置后, 发送方向从内存中读取报文数据时均为 Cacheable 操作。

----结束

非 Cacheable 操作配置说明

软件配置寄存器 BUS_CTRL[noncoh_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E8, bit[4]) 为 1, 则无论发送还是接收均为非 Cacheable 操作。



须知

配置寄存器 BUS_CTRL 中 cf_rx_alloc、huser、noncoh_en 等 bit 除上述合法配置值外，均为非法配置值，会导致不可预期的错误。

18.4.9.2 Bufferable 操作

Bufferable 操作配置说明

Bufferable 操作配置说明如下：

- 步骤 1 软件配置寄存器 BUS_CTRL[cf_pd_buf](基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04E8，bit[6])为 0，配置禁止接收描述符及时间戳上报写内存操作为 bufferable 操作。
- 步骤 2 软件配置寄存器 BUS_CTRL[cf_buf](基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04E8，bit[3])为 1，配置使能接收报文内容写内存操作为 bufferable 操作。

----结束

须知

BUS_CTRL[cf_pd_buf]务必配置为 0，否则可能出现不可预期的错误。

18.4.10 报文头字节匹配

Hi3093 网口支持对报文头指定偏移地址的 32 Byte 进行匹配功能，软件可通过掩码寄存器对指定需要匹配的报文头字节位置及匹配值，硬件对接收报文根据配置值进行匹配，如果匹配成功则将描述符的相应位置 1，否则置 0。

报文头字节匹配流程如下：

- 步骤 1 由软件配置寄存器 RX_PKT_MODE[cf_match_offset](基地址 0x0CD00000+1000*ch_num，偏移地址为 0x04F4，bit[20:12])，设置字符串匹配在报文中的起始偏移。
- 步骤 2 由软件配置参与匹配的报文头部字节掩码表寄存器，MATCH_MASK_0(基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0538)，指明需要匹配的字节。



步骤 3 由软件配置报头匹配的字节表 MATCH_TBL_0 到 MATCH_TBL_7（基地址 0x0CD00000+1000*ch_num，偏移地址从 0x0540 到 0x055C），指明匹配的参考字节值。

---结束

18.4.11 流量控制

18.4.11.1 发送流控帧（全双工模式）

当接收方向的报文无法及时缓存时（例如总线繁忙或没有及时申请到缓存地址等），硬件支持发送流控帧通知对接设备控制流量，对接设备接收到流控帧后会进行相应的处理（通常会停止发送报文）。

Hi3093 支持硬件自动发送流控帧或软件配置发送流控帧。

硬件自动发送流控帧

硬件自动发送流控帧配置说明如下：

- 步骤 1 配置本机 MAC 地址 2 寄存器 STATION_ADDR_HIGH_2[station_addr_high_2]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0214，bit[15:0]）为发送流控帧的源 MAC 地址的高 16bit，配置本机 MAC 地址 2 寄存器 STATION_ADDR_LOW_2（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0210）为发送流控帧的源 MAC 地址的低 32bit。
- 步骤 2 配置流控帧类型域寄存器 FD_FC_TYPE[fd_fc_type]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x000C，bit[15:0]）为发送流控帧的流控帧类型域。
- 步骤 3 配置流控时间参数寄存器 FC_TX_TIMER[fc_tx_timer]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x001C，bit[15:0]）为发送流控帧的流控时间参数域。
- 步骤 4 配置发送流控帧间隙寄存器 PAUSE_THR[pause_thr]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0038，bit[15:0]）为发送流控帧的间隙，该参数的单位是：1000M 为 64 个 125M 时钟周期；100M 为 128 个 25M 时钟周期；10M 为 128 个 2.5M 时钟周期。
- 步骤 5 配置流控使能寄存器 PAUSE_EN[tx_fdfc]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0048，bit[1]）为 1。



----结束

软件发送流控帧

软件发送流控帧配置说明如下：

- 步骤 1 配置本机 MAC 地址 2 寄存器 STATION_ADDR_HIGH_2[station_addr_high_2]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0214，bit[15:0]）为发送流控帧的源 MAC 地址的高 16bit，配置本机 MAC 地址 2 寄存器 STATION_ADDR_LOW_2（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0210）为发送流控帧的源 MAC 地址的低 32bit。
- 步骤 2 配置流控帧类型域寄存器 FD_FC_TYPE[fd_fc_type]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x000C，bit[15:0]）为发送流控帧的流控帧类型域。
- 步骤 3 配置流控时间参数寄存器 FC_TX_TIMER[fc_tx_timer]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x001C，bit[15:0]）为发送流控帧的流控时间参数域。
- 步骤 4 配置发送流控帧间隙寄存器 PAUSE_THR[pause_thr]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0038，bit[15:0]）为发送流控帧的间隙，该参数的单位是：1000M 为 64 个 125M 时钟周期；100M 为 128 个 25M 时钟周期；10M 为 128 个 2.5M 时钟周期。
- 步骤 5 配置流控使能寄存器 PAUSE_EN[tx_fdfc]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0048，bit[1]）为 1。
- 步骤 6 配置发送流控控制寄存器 CF_TX_PAUSE[cf_tx_pause_auto]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0470，bit[1]）为 1。

----结束

18.4.11.2 接收流控帧（全双工模式）

接收方向收到流控帧后，可以响应流控帧，即自动暂停发送报文，具体配置说明如下：

- 步骤 1 配置流控帧目的地址寄存器 FD_FC_ADDR_HIGH[fd_fc_addr_high]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0024，bit[15:0]）为接收流控帧目的地址的高 16bit，配置流控帧目的地址寄存器 FD_FC_ADDR_LOW（基地址



0x0CD00000+1000*ch_num, 偏移地址为 0x0020) 为接收流控帧目的地址的低 32bit。

步骤 2 配置流控使能寄存器 PAUSE_EN[rx_fdfc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0048, bit[0]) 为 1。

----结束

须知

接收方向对流控帧地址检查时, 接收报文的目的地址和[步骤 1](#) 配置的流控帧目的地址寄存器值相同或是标准的流控帧目的地址 (48'h0180_C200_0001), 即认为流控帧的地址检查正确。

18.4.11.3 发送 JAM 帧 (半双工模式)

Hi3093 支持半双工模式发生冲突时发送 JAM 帧。

配置流控使能寄存器 PAUSE_EN[tx_hdfc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0048, bit[2]) 为 1。

18.4.12 端口模式配置

如[图 18-1](#) 所示, Hi3093 芯片外对接了 PHY, 所以需要配置 GMAC 的端口模式为 PHY 的工作模式才能正常通信。

须知

- 网口 GMAC0、1 支持 RMII10M/100M 接口, 网口 GMAC2~3 支持 SGMII10M/100M/1000M 接口;
- 1000M 不支持半双工。

端口模式配置如下:

步骤 1 配置端口通道使能寄存器 PORT_EN[tx_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0044, bit[2]) 为 0, 关闭发送通道。

步骤 2 配置端口通道使能寄存器 PORT_EN[rx_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0044, bit[1]) 为 0, 关闭接收通道。



步骤 3 配置全双工半双工模式寄存器 DUPLEX_TYPE[duplex_sel] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0008, bit[0]) 为对接 PHY 的双工模式。

步骤 4 配置端口模式寄存器 PORT_MODE[port_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0040, bit[3:0]) 为对接 PHY 的工作模式。

步骤 5 配置端口模式改变使能寄存器 MODE_CHANGE_EN[mode_change_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01B4, bit[0]) 为 1。

步骤 6 配置 MAC_CORE 的应用侧时钟软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[2];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[9];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[16];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[23]。

步骤 7 配置 MAC 的工作时钟软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[0];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[7];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[14];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[21]。

步骤 8 配置 MAC 的发送方向 SERDES 的时钟软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[6];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[13];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[20];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[27]。

步骤 9 配置 MAC 的接收方向 SERDES 时钟软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[5];



- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[12];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[19];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[26]。

步骤 10 配置 MAC_CORE 的应用侧时钟软复位寄存器为 0。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[2];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[9];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[16];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[23]。

步骤 11 配置 MAC 的工作时钟软复位寄存器为 0。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[0];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[7];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[14];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[21]。

步骤 12 配置 MAC 的发送方向 SERDES 的时钟软复位寄存器为 0。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[6];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[13];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[20];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[27]。

步骤 13 配置 MAC 的接收方向 SERDES 时钟软复位寄存器为 0。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[5];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[12];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[19];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0000, bit[26]。



- 步骤 14 配置端口通道使能寄存器 PORT_EN[tx_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0044, bit[2]) 为 1, 开启发送通道。
- 步骤 15 配置端口通道使能寄存器 PORT_EN[rx_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0044, bit[1]) 为 1, 开启接收通道。

----结束

18.4.13 SerDes 自协商

Hi3093 支持 802.3 协议规定的自协商功能, Hi3093 可以主动发起自协商或被动响应自协商, 主动发起自协商具体配置如下:

- 步骤 1 配置自协商本地配置寄存器 TX_LOCAL_PAGE (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x005C) 为 32'h1A0。
- 步骤 2 配置自协商链接时间寄存器 SIXTEEN_BIT_CNTR[sixteen_bit_cntr] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01CC, bit[15:0]) 为 16'h9896。
- 步骤 3 配置自协商链接时间更新寄存器 LD_LINK_COUNTER[lid_link_counter] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01D0, bit[1]) 为 1。
- 步骤 4 配置发送控制寄存器 TRANSMIT_CONTROL[an_enable] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0060, bit[5]) 为 1。
- 步骤 5 读取自协商状态寄存器 AN_NEG_STATE (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0058) 的值。当该寄存器的 bit[21]为 1 时, 表示自协商完成, 此时 bit[15]表示对接设备能否支持 next_page 操作, bit[13:12]表示对设备的链接状态, bit[8:7]表示对接设备对流控帧的支持能力, bit[6:5]表示对接设备对双工模式的支持能力。

----结束

须知

SerDes 自协商可以主动发起或被动响应自协商, 但协商结果不会改变 MAC 的状态。



18.4.14 SGMII 自协商

Hi3093 支持 SGMII 协议规定的自协商功能，使用该功能时，无须 MDIO 就能获取 PHY 的工作状态，软件可以读取自协商结果再配置 GMAC 的端口模式，使得 GMAC 和 PHY 可以正常通信。具体配置如下：

- 步骤 1 配置 SGMII 自协商本地配置寄存器 TX_LOCAL_PAGE（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x005C）为 32'h1。
- 步骤 2 配置自协商链接时间寄存器 SIXTEEN_BIT_CNTR[sixteen_bit_cntr]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x01CC，bit[15:0]）为 16'h3F。
- 步骤 3 配置自协商链接时间更新寄存器 LD_LINK_COUNTER[lid_link_counter]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x01D0，bit[1]）为 1。
- 步骤 4 配置发送控制寄存器 TRANSMIT_CONTROL[an_enable]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0060，bit[5]）为 1。
- 步骤 5 读取 SGMII 自协商状态寄存器 AN_NEG_STATE（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0058）的值。当该寄存器的 bit[21]为 1 时，表示自协商完成，此时 bit[15]表示 PHY 的线路侧链接状态，bit[12]表示 PHY 的双工模式，bit[11:10]表示 PHY 的工作速率。

----结束

须知

SGMII 自协商只会读取 PHY 的状态到本地寄存器，不会改变 MAC 的状态。

18.4.15 接收报文过滤

18.4.15.1 短帧过滤

MACCORE 的接收通道可以通过配置相应的寄存器选择是否透传短帧。如果该功能被使能，接收到的短帧不会被丢弃，并且被统计进接收有效帧字节统计寄存器。

须知

以下必须配置为短帧透传禁止。



配置短帧透传使能寄存器 RE_CV_CONTROL[runt_pkt_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01E0, bit[4]) 为 0, 短帧透传禁止。

须知

- 当同时使能短帧过滤和 PAD 剥离功能时, 对于接收到的 802.3 封装短包 (小于 64 Byte) 将无法过滤。
- 对于接收到的 802.3 封装短包 (小于 64 Byte, 且 CRC 等均正确), MAC 不会因为其报文长度小于 64 Byte 而上报 2 层解析错误。

18.4.15.2 流控帧过滤

MACCORE 的接收通道可以通过配置相应的寄存器对流控帧进行过滤。首先对接收到的流控帧 DA 域进行检查, 如果该流控帧的 DA 域与 01-80-C2-00-00-01 完全匹配, 或者与流控帧目的地址寄存器值完全匹配, 则该流控帧 MAC 地址匹配。继续判断该流控帧的长度/类型域是否为 88-08-00-01, 如果匹配在流控帧过滤使能的情况下则丢弃该流控帧, 并进行接收方向过滤掉的帧统计和接收方向过滤掉的字节统计。流控帧虽然被丢弃但流控时间被存储, 发送方向将按照流控时间停止发送报文流。

须知

接收流控帧过滤使能时, 当接收到流控帧该帧将被丢弃并统计进接收方向丢弃的帧和字节统计寄存器。全双工模式下相应流控帧使能, 发送方向才能按照流控时间停止发送报文, 并在 CRC 检测有效且 CRC 正确、帧长大于等于 64 Byte 的情况下被统计进接收到的流控帧统计寄存器。

流控帧过滤配置如下:

- 步骤 1 配置流控帧目的地址低 32 位寄存器 FD_FC_ADDR_LOW (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0020, bit[31:0]) , 配置流控帧目的地址高 16 位寄存器 FD_FC_ADDR_HIGH (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0024, bit[15:0]) , 配置流控帧目的地址。
- 步骤 2 配置接收方向相应流帧使能寄存器 PAUSE_EN[rx_fdfc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0048, bit[0]) 为 1, 全双工模式下响应流控帧使能。



步骤 3 配置流控帧过滤使能寄存器 REC_FILT_CONTROL[pause_frm_pass] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0064, bit[4]) 为 0, 流控帧过滤使能。

----结束

18.4.15.3 广播帧过滤

MACCORE 的接收通道可以通过配置相应的寄存器对接收到的广播帧进行过滤, 当该功能有效时检查接收帧的 DA 域是否全 1, 匹配则该帧将被丢弃, 并进行接收方向过滤掉的帧统计和接收方向过滤掉的字节统计。

配置广播帧过滤使能寄存器 REC_FILT_CONTROL[bc_drop_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0064, bit[2]) 为 1, 广播帧过滤使能。

18.4.15.4 多播帧过滤

MACCORE 的接收通道可以通过配置相应的寄存器对接收到的多播帧进行过滤, 当该功能有效时检查接收帧的 DA 域与 2 个多播帧地址匹配寄存器的配置值是否相等, 检查时会根据地址屏蔽寄存器配置值将相应地址 bit 位屏蔽掉 (即屏蔽掉的地址 bit 不进行比较), 如果不匹配, 则该帧将被丢弃, 并进行接收方向过滤掉的帧统计和接收方向过滤掉的字节统计。

须知

检查多播帧 DA 域才有使用带掩码 MAC 地址的需求, 因此带掩码的 MAC 地址用于多播帧 MAC 地址匹配。

多播帧过滤配置如下:

步骤 1 配置本机 MAC 地址 0 寄存器 1STATION_ADDR_LOW_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0200, bit[31:0]), 配置本机 MAC 地址 0 寄存器 2STATION_ADDR_HIGH_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0204, bit[15:0]), 第 8 位为 1, 其他位根据需求可选; 配置本机 MAC 地址 1 寄存器 2STATION_ADDR_LOW_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0208, bit[31:0]), 配置本机 MAC 地址 1 寄存器 2STATION_ADDR_HIGH_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x020C, bit[15:0]), 第 8 位为 1, 其他位根据需求可选。

步骤 2 配置本机 MAC 地址 0 寄存器 1 的掩码 STATION_ADDR_MSK_LOW_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0230, bit[31:0]); 配置本机 MAC 地址



0 寄存器 2 的掩码 STATION_ADDR_MSK_HIGH_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0234, bit[15:0]) ; 这两个寄存器与步骤 1 中所配的前两个寄存器按位与的结果构成一个带掩码的 MAC 地址。配置本机 MAC 地址 1 寄存器 1 的掩码 STATION_ADDR_MSK_LOW_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0238, bit[31:0]) ; 配置本机 MAC 地址 1 寄存器 2 的掩码 STATION_ADDR_MSK_HIGH_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x023C, bit[15:0]) , 第 8 位为 1, 其他位根据需求可选; 这两个寄存器与步骤 1 中所配的后两个寄存器按位与的结果构成一个带掩码的 MAC 地址。

步骤 3 配置多播帧过滤使能寄存器 REC_FILT_CONTROL[mc_drop_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0064, bit[1]) 为 1, 多播帧过滤使能。

----结束

18.4.15.5 单播帧过滤

MACCORE 的接收通道可以通过配置相应的寄存器对接收到的单播帧进行过滤, 当该功能有效时检查接收帧的 DA 域是否与 4 个 MAC 源地址中至少一个完全匹配, 否则该帧将被丢弃, 并进行接收方向过滤掉的帧统计和接收方向过滤掉的字节统计。

须知

- 一般情况下 4 个不带掩码的 MAC 源地址用于支持单播 MAC 地址匹配, 而 2 个带掩码的 MAC 地址用于支持多播 MAC 地址匹配;
- 当应用不需要多播 MAC 地址匹配时, 务必将其对应的掩码寄存器 (STATION_ADDR_MSK_LOW_0/1、STATION_ADDR_MSK_HIGH_0/1, 偏移地址为 0x0230—0x023C) 所有 bit 位配置为全 1。

单播帧过滤配置如下:

步骤 1 配置本机 MAC 地址 2 寄存器 1 STATION_ADDR_LOW_2 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0210, bit[31:0]) , 配置本机 MAC 地址 2 寄存器 2 STATION_ADDR_HIGH_2 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0214, bit[15:0]) , 第 8 位为 0, 其他位根据需求可选; 这两个寄存器的组合值为一个本机 MAC 源地址。配置本机 MAC 地址 3 寄存器 2 STATION_ADDR_LOW_3 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0218, bit[31:0]) , 配置本机 MAC 地址 3 寄存器 2 STATION_ADDR_HIGH_3 (基



地址 $0x0CD00000 + 1000 * ch_num$ ，偏移地址为 $0x021C$ ，bit[15:0])，第 8 位为 0，其他位根据需求可选；这两个寄存器的组合值为一个本机 MAC 源地址。配置本机 MAC 地址 4 寄存器 1STATION_ADDR_LOW_4 (基地址 $0x0CD00000 + 1000 * ch_num$ ，偏移地址为 $0x0220$ ，bit[31:0])，配置本机 MAC 地址 4 寄存器 2STATION_ADDR_HIGH_4 (基地址 $0x0CD00000 + 1000 * ch_num$ ，偏移地址为 $0x0224$ ，bit[15:0])，第 8 位为 0，其他位根据需求可选；这两个寄存器的组合值为一个本机 MAC 源地址。配置本机 MAC 地址 5 寄存器 2STATION_ADDR_LOW_5 (基地址 $0x0CD00000 + 1000 * ch_num$ ，偏移地址为 $0x0228$ ，bit[31:0])，配置本机 MAC 地址 5 寄存器 2STATION_ADDR_HIGH_5 (基地址 $0x0CD00000 + 1000 * ch_num$ ，偏移地址为 $0x022C$ ，bit[15:0])，第 8 位为 0，其他位根据需求可选；这两个寄存器的组合值为一个本机 MAC 源地址。

步骤 2 配置单播帧过滤使能寄存器 REC_FILT_CONTROL[uc_drop_en] (基地址 $0x0CD00000 + 1000 * ch_num$ ，偏移地址为 $0x0064$ ，bit[0]) 为 1，单播帧过滤使能。

----结束

18.5 工作方式

18.5.1 时钟门控

每个 GMAC 都支持时钟关闭功能。由于 GMAC 的工作时钟默认为开启模式，所以在系统应用不需要每个 GMAC 都工作的场景中，可以将不工作的 GMAC 的工作时钟关闭，来降低芯片的动态功耗。

下面以 GMAC0 为例说明关闭时钟和开启时钟的流程：

GMAC0 关闭时钟步骤如下：

- 步骤 1 读取系统寄存器 HP_CLK_CFG。
- 步骤 2 判断 GMAC0 的 AHB 总线时钟状态，若 HP_CLK_CFG bit[0]为 1，表示 GMAC0 有总线时钟，则执行[步骤 3](#)，否则执行[步骤 4](#)。
- 步骤 3 将 HP_CLK_DIS bit[0]置 1，关闭 GMAC0 的总线时钟。
- 步骤 4 判断 GMAC0 的工作时钟状态，若 HP_CLK_CFG bit[1]为 1，表示 GMAC0 有工作时钟，则执行[步骤 5](#)，否则结束。
- 步骤 5 将 HP_CLK_DIS bit[1]置 1，关闭 GMAC0 的工作时钟。



----结束

GMAC0 时钟开启步骤如下：

- 步骤 1 读取系统寄存器 HP_CLK_CFG。
- 步骤 2 判断 GMAC0 的 AHB 总线时钟状态，若 HP_CLK_CFG bit[0]为 1，表示 GMAC0 有总线时钟，则执行步骤 4，否则执行步骤 3。
- 步骤 3 将 HP_CLK_ENB bit[0]置 1，开启 GMAC0 的总线时钟。
- 步骤 4 判断 GMAC0 的工作时钟状态，若 HP_CLK_CFG bit[1]为 1，表示 GMAC0 有工作时钟，则结束，否则执行步骤 5。
- 步骤 5 将 HP_CLK_ENB bit[1]置 1，开启 GMAC0 的工作时钟。

----结束

表18-31 网口的门控时钟使能寄存器表 0x0CD0C000

寄存器名称	偏移地址	bit	成员名称	类型	描述	复位值
HP_CLK_ENB	0x020	[7]	gmac3_125m_clk_enb	WO	gmac3 工作时钟使能，高有效。	0x1
HP_CLK_ENB	0x020	[6]	Gmac3sys_clk_enb	WO	gmac3AHB 总线时钟使能，高有效。	0x1
HP_CLK_ENB	0x20	[5]	gmac2_125m_clk_enb	WO	gmac2 工作时钟使能，高有效。	0x1
HP_CLK_ENB	0x020	[4]	gmac2sys_clk_enb	WO	gmac2AHB 总线时钟使能，高有效。	0x1
HP_CLK_ENB	0x020	[3]	gmac1_125m_clk_enb	WO	gmac1 工作时钟使能，高有效。	0x1
HP_CLK_ENB	0x020	[2]	gmac1sys_clk_enb	WO	gmac1AHB 总线时钟使能，高有效。	0x1



HP_CLK_ENB	0x020	[1]	gmac0_125m_clk_enb	WO	gmac0 工作时钟使能，高有效。	0x1
HP_CLK_ENB	0x020	[0]	gmac0sy_s_clk_enb	WO	gmac0AHB 总线时钟使能，高有效。	0x1

表18-32 网口的门控时钟禁止寄存器表 0x0CD0C000

寄存器名称	偏移地址	bit	成员名称	类型	描述	复位值
HP_CLK_DIS ENB	0x01c	[7]	gmac3_125m_clk_dsb	WO	gmac3 工作时钟禁止，高有效。	0x0
HP_CLK_DIS ENB	0x01c	[6]	Gmac3sy_s_clk_dsb	WO	gmac3AHB 总线时钟禁止，高有效。	0x0
HP_CLK_DIS ENB	0x01c	[5]	gmac2_125m_clk_dsb	WO	gmac2 工作时钟禁止，高有效。	0x0
HP_CLK_DIS ENB	0x01c	[4]	gmac2sy_s_clk_dsb	WO	gmac2AHB 总线时钟禁止，高有效。	0x0
HP_CLK_DIS ENB	0x01c	[3]	gmac1_125m_clk_dsb	WO	gmac1 工作时钟禁止，高有效。	0x0
HP_CLK_DIS ENB	0x01c	[2]	gmac1sy_s_clk_dsb	WO	gmac1AHB 总线时钟禁止，高有效。	0x0
HP_CLK_DIS ENB	0x01c	[1]	gmac0_125m_clk_dsb	WO	gmac0 工作时钟禁止，高有效。	0x0
HP_CLK_DIS ENB	0x01c	[0]	gmac0sy_s_clk_dsb	WO	gmac0AHB 总线时钟禁止，高有效。	0x0



表18-33 网口的门控时钟状态寄存器表

寄存器名称	偏移地址	bit	成员名称	类型	描述	复位值
HP_CLK_CFG	0x018	[7]	gmac3_125m_clk_st	RO	gmac0 工作时钟状态，高有效。	0x1
HP_CLK_CFG	0x018	[6]	gmac3sys_clk_st	RO	gmac0AHB 总线时钟状态，高有效。	0x1
HP_CLK_CFG	0x018	[5]	gmac2_125m_clk_st	RO	gmac1 工作时钟状态，高有效。	0x1
HP_CLK_CFG	0x018	[4]	gmac2sys_clk_st	RO	gmac1AHB 总线时钟状态，高有效。	0x1
HP_CLK_CFG	0x018	[3]	gmac1_125m_clk_st	RO	gmac2 工作时钟状态，高有效。	0x1
HP_CLK_CFG	0x018	[2]	gmac1sys_clk_st	RO	gmac2AHB 总线时钟状态，高有效。	0x1
HP_CLK_CFG	0x018	[1]	gmac0_125m_clk_st	RO	gmac3 工作时钟状态，高有效。	0x1
HP_CLK_CFG	0x018	[0]	gmac0sys_clk_st	RO	gmac3AHB 总线时钟状态，高有效。	0x1

18.5.2 软复位

每个 GMAC 的每个时钟域都支持单独软复位的功能。

典型 GMAC 软复位应用场景：

- 在芯片上电后且 GMAC 对接的相应 serdes 锁定后，需按照 [18.5.2.1 复位 GMAC](#) 章节复位 GMAC；
- 改变 GMAC 端口模式时，请按照 [18.5.3.1 端口模式寄存器初始化配置（模式选择、通道使能）](#) 章节配置；
- 当 Phy 的速率发生切换时，请按照 [18.5.3.1 端口模式寄存器初始化配置（模式选择、通道使能）](#) 章节配置，确保 GMAC 正常工作；



- Serdes 配置参数刷新及复位后，请按照 [18.5.3.1 端口模式寄存器初始化配置（模式选择、通道使能）](#) 章节）配置，确保 GMAC 正常工作；
- 出现 FIFO 又空又满中断可以按照“[18.5.3.1 端口模式寄存器初始化配置（模式选择、通道使能）](#)”章节重新配置端口模式来恢复正常。

18.5.2.1 复位 GMAC

上电后或某些应用场景需要复位 GMAC，配置流程如下：

须知

在配置 GMAC 的软复位寄存器过程中，请不要读写 GMAC 的寄存器。

- 步骤 1 配置端口通道使能寄存器 PORT_EN[tx_en]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0044，bit[2]）为 1，开启发送通道，配置端口通道使能寄存器 PORT_EN[rx_en]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0044，bit[1]）为 0，关闭接收通道。
- 步骤 2 查询配置 FIFO 寄存器 CF_CFF_DATA_NUM[cf_tx_cfg_num]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x045c，bit[8:0]），若返回值为 0，进入 [步骤 4](#)；否则隔 10ms 查询该寄存器，若返回值改变为 0 进入 [步骤 4](#)，若返回值改变但不为 0，隔 10ms 重新执行该步骤，若返回值没有改变，则疑似系统总线接口数据传输异常，此时建议软件对系统进行软复位。

须知

进入复位流程后，软件务必停止发包，即禁止继续向 GMAC 添加发送描述符。

- 步骤 3 查询 FIFO_CURR_STATUS[rff_cf_empty]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0454，bit[6]）和 FIFO_CURR_STATUS[cff_cf_rx_empty]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0454，bit[8]），如果 rff_cf_empty 为 1，则进入 [步骤 5](#)，如果 rff_cf_empty 为 0，且 cff_cf_rx_empty 为 1，则进入 [步骤 6](#)，如果 rff_cf_empty 为 0，且 cff_cf_rx_empty 为 0，则进入 [步骤 7](#)。
- 步骤 4 延时 10ms，查询 PCU 的状态机寄存器 DEBUG_ST_MCH（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0450，bit[8:0]），返回值为 0，则进入 [步骤 8](#)，否则疑似系统总线接口数据传输异常，此时建议软件对系统进行软复位。



步骤 5 延时 10ms，查询 PCU 的状态机寄存器 `DEBUG_ST_MCH`（基地址 `0x0CD00000+1000*ch_num`，偏移地址为 `0x0450`，`bit[8:0]`），返回值为 0 或 20，则进入步骤 8，否则疑似系统总线接口数据传输异常，此时建议软件对系统进行软复位。

步骤 6 查询 `CF_CFF_DATA_NUM[cf_rx_cfg_num]`（基地址 `0x0CD00000+1000*ch_num`，偏移地址为 `0x045C`，`bit[24:16]`），返回值为 0，则进入步骤 4，返回值为非 0，则延时 10ms 重新查询，如果查询值未发生变化，则查询 `FIFO_CURR_STATUS[rff_cf_empty]`（基地址 `0x0CD00000+1000*ch_num`，偏移地址为 `0x0454`，`bit[6]`），如果为 1，则返回步骤 4，否则疑似系统总线接口数据传输异常，此时建议软件对系统进行软复位。

步骤 7 配置 MAC 的总线接口软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[4]`；
- GMAC1：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[11]`；
- GMAC2：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[18]`；
- GMAC3：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[25]`。

步骤 8 配置 MAC 的工作时钟软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[0]`；
- GMAC1：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[7]`；
- GMAC2：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[14]`；
- GMAC3：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[21]`。

步骤 9 配置 MAC_CORE 的配置寄存器软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[3]`；
- GMAC1：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[10]`；
- GMAC2：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[17]`；
- GMAC3：基地址 `0x0CD0C000`，偏移地址为 `0x0008`，`bit[24]`。

步骤 10 配置 MAC 的发送方向 SERDES 的时钟软复位寄存器为 1。



寄存器具体地址如下：

- GMAC0：基地址 0x0CD0C000，偏移地址为 0x0008，bit[6]；
- GMAC1：基地址 0x0CD0C000，偏移地址为 0x0008，bit[13]；
- GMAC2：基地址 0x0CD0C000，偏移地址为 0x0008，bit[20]；
- GMAC3：基地址 0x0CD0C000，偏移地址为 0x0008，bit[27]。

步骤 11 配置 MAC 的接收方向 SERDES 时钟软复位寄存器为 1。

寄存器具体地址如下：

- GMAC0：基地址 0x0CD0C000，偏移地址为 0x0008，bit[5]；
- GMAC1：基地址 0x0CD0C000，偏移地址为 0x0008，bit[12]；
- GMAC2：基地址 0x0CD0C000，偏移地址为 0x0008，bit[19]；
- GMAC3：基地址 0x0CD0C000，偏移地址为 0x0008，bit[26]。

步骤 12 配置 MAC 的总线接口软复位解除寄存器为 1。

寄存器具体地址如下：

- GMAC0：基地址 0x0CD0C000，偏移地址为 0x0010，bit[4]；
- GMAC1：基地址 0x0CD0C000，偏移地址为 0x0010，bit[11]；
- GMAC2：基地址 0x0CD0C000，偏移地址为 0x0010，bit[18]；
- GMAC3：基地址 0x0CD0C000，偏移地址为 0x0010，bit[25]。

步骤 13 配置 MAC 的工作时钟软复位解除寄存器为 1。

寄存器具体地址如下：

- GMAC0：基地址 0x0CD0C000，偏移地址为 0x0010，bit[0]；
- GMAC1：基地址 0x0CD0C000，偏移地址为 0x0010，bit[7]；
- GMAC2：基地址 0x0CD0C000，偏移地址为 0x0010，bit[14]；
- GMAC3：基地址 0x0CD0C000，偏移地址为 0x0010，bit[21]。

步骤 14 配置 MAC_CORE 的配置寄存器软复位解除寄存器为 1。

寄存器具体地址如下：

- GMAC0：基地址 0x0CD0C000，偏移地址为 0x0010，bit[3]；
- GMAC1：基地址 0x0CD0C000，偏移地址为 0x0010，bit[10]；
- GMAC2：基地址 0x0CD0C000，偏移地址为 0x0010，bit[17]；



- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[24]。

步骤 15 配置 MAC 的发送方向 SERDES 的时钟软复位解除寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[6];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[13];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[20];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[27]。

步骤 16 配置 MAC 的接收方向 SERDES 时钟软复位解除寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[5];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[12];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[19];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[26]。

----结束

须知

复位 GMAC 后需要重新初始化 GMAC。

18.5.2.2 网口的各个时钟域的软复位寄存器

表18-34 网口的软复位寄存器表 0x0CD0C000

寄存器名称	偏移地址	bit	成员名称	类型	描述	复位值
HP_SRST_R EQ1	0x0008	[27]	gmac3_tx_srst_req	RW	gmac3 SerDes 接口 tx 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[26]	gmac3_rx_srst_req	RW	gmac3 SerDes 接口 rx 软复位请求。高有效。	0x0



寄存器名称	偏移地址	bit	成员名称	类型	描述	复位值
HP_SRST_R EQ1	0x0008	[25]	gmac3_sys_srst_req	RW	gmac3 总线接口软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[24]	gmac3_cfg_srst_req	RW	gmac3 寄存器配置域软复位请求，高有效。	0x0
HP_SRST_R EQ1	0x0008	[23]	gmac3_macsys_srst_req	RW	gmac3 的 MAC_CORE 应用侧时钟软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[22]	gmac3_fifo_srst_req	RW	gmac3 fifo 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[21]	gmac3_eth_srst_req	RW	gmac3 工作域软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[20]	gmac2_tx_srst_req	RW	gmac2 SerDes 接口 tx 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[19]	gmac2_rx_srst_req	RW	gmac2 SerDes 接口 rx 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[18]	gmac2_sys_srst_req	RW	gmac2 总线接口软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[17]	gmac2_cfg_srst_req	RW	gmac2 寄存器配置域软复位请求，高有效。	0x0
HP_SRST_R EQ1	0x0008	[16]	gmac2_macsys_srst_req	RW	gmac2 的 MAC_CORE 应用侧时钟软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[15]	gmac2_fifo_srst_req	RW	gmac2 fifo 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[14]	gmac2_eth_srst_req	RW	gmac2 工作域软复位请求。高有效。	0x0



寄存器名称	偏移地址	bit	成员名称	类型	描述	复位值
HP_SRST_R EQ1	0x0008	[13]	gmac1_tx_srst_req	RW	gmac1 SerDes 接口 tx 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[12]	gmac1_rx_srst_req	RW	gmac1 SerDes 接口 rx 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[11]	gmac1_sys_srst_req	RW	gmac1 总线接口软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[10]	gmac1_cfg_srst_req	RW	gmac1 寄存器配置域软复位请求，高有效。	0x0
HP_SRST_R EQ1	0x0008	[9]	gmac1_macsys_srst_req	RW	gmac1 的 MAC_CORE 应用侧时钟软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[8]	gmac1_fifo_srst_req	RW	gmac1 fifo 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[7]	gmac1_eth_srst_req	RW	gmac1 工作域软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[6]	gmac0_tx_srst_req	RW	gmac0 SerDes 接口 tx 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[5]	gmac0_rx_srst_req	RW	gmac0 SerDes 接口 rx 软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[4]	gmac0_sys_srst_req	RW	gmac0 总线接口软复位请求。高有效。	0x0
HP_SRST_R EQ1	0x0008	[3]	gmac0_cfg_srst_req	RW	gmac0 寄存器配置域软复位请求，高有效。	0x0
HP_SRST_R EQ1	0x0008	[2]	gmac0_macsys_srst_req	RW	gmac0 的 MAC_CORE 应用侧时钟软复位请求。高有效。	0x0



寄存器名称	偏移地址	bit	成员名称	类型	描述	复位值
HP_SRST_REQ1	0x0008	[1]	gmac0_fifo_srst_req	RW	gmac0 fifo 软复位请求。高有效。	0x0
HP_SRST_REQ1	0x0008	[0]	gmac0_eth_srst_req	RW	gmac0 工作域软复位请求。高有效。	0x0

以 GMAC0 的工作时钟复位为例说明软复位的流程。具体步骤如下：

- 步骤 1 配置 HP_SRST_REQ1 为 1，复位 GMAC0 的工作时钟域。
- 步骤 2 等待 10ms，保证该时钟域已经被复位。
- 步骤 3 配置 HP_SRST_DIS1 为 1，撤离 GMAC0 的工作时钟域复位。

----结束

18.5.3 典型初始化配置流程

须知

芯片上电后，Serdes 锁定前的不稳定时钟可能会使 GMAC 进入异常状态，所以 Serdes 锁定后请务必按照 [18.5.2.1 复位 GMAC](#) 章节复位下 GMAC；改变 GMAC 模式请按照下面“端口模式寄存器初始化配置”章节操作，即改变模式后复位部分 GMAC。

18.5.3.1 端口模式寄存器初始化配置（模式选择、通道使能）

端口模式寄存器初始化配置如下：

- 步骤 1 配置端口通道使能寄存器 PORT_EN[tx_en]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0044，bit[2]）为 0，关闭发送通道。
- 步骤 2 配置端口通道使能寄存器 PORT_EN[rx_en]（基地址 0x0CD00000+1000*ch_num，偏移地址为 0x0044，bit[1]）为 0，关闭接收通道。



步骤 3 配置全双工半双工模式寄存器 DUPLEX_TYPE[duplex_sel] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0008, bit[0]) 为对接 PHY 的双工模式。

步骤 4 配置端口模式寄存器 PORT_MODE[port_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0040, bit[3:0]) 为对接 PHY 的工作模式。

须知

网口 GMAC0、1 支持 MII10M/100M, 网口 GMAC2 ~ 3 支持 SGMII10M/100M/1000M 接口; 1000M 不支持半双工。

步骤 5 配置端口模式改变使能寄存器 MODE_CHANGE_EN[mode_change_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01B4, bit[0]) 为 1。

步骤 6 配置 MAC_CORE 的应用侧时钟软复位寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[2];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[9];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[16];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[23]。

步骤 7 配置 MAC 的工作时钟软复位寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[0];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[7];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[14];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[21]。

步骤 8 配置 MAC 的发送方向 SERDES 的时钟软复位寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[6];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[13];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[20];



- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[27]。

步骤 9 配置 MAC 的接收方向 SERDES 时钟软复位寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[5];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[12];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[19];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0008, bit[26]。

步骤 10 配置 MAC_CORE 的应用侧时钟软复位解除寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[2];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[9];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[16];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[23]。

步骤 11 配置 MAC 的工作时钟软复位解除寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[0];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[7];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[14];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[21]。

步骤 12 配置 MAC 的发送方向 SERDES 的时钟软复位解除寄存器为 1。

寄存器具体地址如下:

- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[6];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[13];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[20];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[27]。

步骤 13 配置 MAC 的接收方向 SERDES 时钟软复位解除寄存器为 1。

寄存器具体地址如下:



- GMAC0: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[5];
- GMAC1: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[12];
- GMAC2: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[19];
- GMAC3: 基地址 0x0CD0C000, 偏移地址为 0x0010, bit[26]。

步骤 14 配置端口通道使能寄存器 PORT_EN[tx_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0044, bit[2]) 为 1, 打开发送通道。

步骤 15 配置端口通道使能寄存器 PORT_EN[rx_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0044, bit[1]) 为 1, 打开接收通道。

----结束

18.5.3.2 报文过滤相关寄存器初始化配置 (流量控制、过滤)

须知

半双工模式不使用流控帧, 可以忽略[步骤 1](#) ~ [步骤 8](#)。

报文过滤相关寄存器初始化配置如下:

- 步骤 1 配置本机 MAC 地址 2 寄存器 STATION_ADDR_HIGH_2[station_addr_high_2] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0214, bit[15:0]) 为发送流控帧的源 MAC 地址的高 16bit, 配置本机 MAC 地址 2 寄存器 STATION_ADDR_LOW_2 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0210) 为发送流控帧的源 MAC 地址的低 32bit。
- 步骤 2 配置流控帧类型域寄存器 FD_FC_TYPE[fd_fc_type] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x000C, bit[15:0]) 为发送流控帧的流控帧类型域。
- 步骤 3 配置流控时间参数寄存器 FC_TX_TIMER[fc_tx_timer] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x001C, bit[15:0]) 为发送流控帧的流控时间参数域, 该参数的单位是: 1000M 为 64 个 125M 时钟周期; 100M 为 128 个 25M 时钟周期; 10M 为 128 个 2.5M 时钟周期。
- 步骤 4 配置发送流控帧间隙寄存器 PAUSE_THR[pause_thr] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0038, bit[15:0]) 为发送流控帧的间隙, 该参数的单位是: 1000M 为 64 个 125M 时钟周期; 100M 为 128 个 25M 时钟周期; 10M 为 128 个 2.5M 时钟周期。



步骤 5 配置流控使能寄存器 PAUSE_EN[tx_fdfc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0048, bit[1]) 为 1。

须知

接收方向对流控帧地址检查时, 接收报文的目的地址和[步骤 1](#) 配置的流控帧目的地址寄存器值相同或是标准的流控帧目的地址 (48'h0180_C200_0001), 即认为流控帧的地址检查正确。

步骤 6 配置流控帧目的地址寄存器 FD_FC_ADDR_HIGH[fd_fc_addr_high] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0024, bit[15:0]) 为接收流控帧目的地址的高 16bit, 配置流控帧目的地址寄存器 FD_FC_ADDR_LOW (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0020) 为接收流控帧目的地址的低 32bit。

步骤 7 配置流控使能寄存器 PAUSE_EN[rx_fdfc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0048, bit[0]) 为 1。

步骤 8 配置流控使能寄存器 PAUSE_EN[tx_fdfc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0048, bit[1]) 为 1。

须知

全双工模式忽略[步骤 9](#)。

步骤 9 配置流控使能寄存器 PAUSE_EN[tx_hdfc] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0048, bit[2]) 为 1。

须知

下面的步骤必须配置为短帧透传禁止。

步骤 10 配置短帧透传使能寄存器 RE_CV_CONTROL[runt_pkt_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01E0, bit[4]) 为 0, 短帧透传禁止。

步骤 11 配置广播帧过滤使能寄存器 REC_FILT_CONTROL[bc_drop_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0064, bit[2]) 为 1, 广播帧过滤使能。



- 步骤 12 配置本机 MAC 地址 0 寄存器 1 STATION_ADDR_LOW_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0200, bit[31:0]) , 配置本机 MAC 地址 0 寄存器 2 STATION_ADDR_HIGH_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0204, bit[15:0]) ; 配置本机 MAC 地址 1 寄存器 2 STATION_ADDR_LOW_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0208, bit[31:0]) , 配置本机 MAC 地址 1 寄存器 2 STATION_ADDR_HIGH_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x020C, bit[15:0]) 。
- 步骤 13 配置本机 MAC 地址 0 寄存器 1 的掩码 STATION_ADDR_MSK_LOW_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0230, bit[31:0]) 最低位配置为 1, 其他位可选; 配置本机 MAC 地址 0 寄存器 2 的掩码 STATION_ADDR_MSK_HIGH_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0234, bit[15:0]) ; 这两个寄存器与步骤 1 中所配的前两个寄存器按位与的结果构成一个带掩码的 MAC 地址。配置本机 MAC 地址 1 寄存器 1 的掩码 STATION_ADDR_MSK_LOW_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0238, bit[31:0]) 最低位配置为 1, 其他位可选; 配置本机 MAC 地址 1 寄存器 2 的掩码 STATION_ADDR_MSK_HIGH_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x023C, bit[15:0]) ; 这两个寄存器与步骤 1 中所配的后两个寄存器按位与的结果构成一个带掩码的 MAC 地址。
- 步骤 14 配置多播帧过滤使能寄存器 REC_FILT_CONTROL[mc_drop_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0064, bit[1]) 为 1, 多播帧过滤使能。
- 步骤 15 配置本机 MAC 地址 0 寄存器 1 STATION_ADDR_LOW_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0200, bit[31:0]) , 配置本机 MAC 地址 0 寄存器 2 STATION_ADDR_HIGH_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0204, bit[15:0]) ; 配置本机 MAC 地址 1 寄存器 2 STATION_ADDR_LOW_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0208, bit[31:0]) , 配置本机 MAC 地址 1 寄存器 2 STATION_ADDR_HIGH_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x020C, bit[15:0]) 。
- 步骤 16 配置本机 MAC 地址 0 寄存器 1 的掩码 STATION_ADDR_MSK_LOW_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0230, bit[31:0]) 最低位配置为 1, 其他位可选; 配置本机 MAC 地址 0 寄存器 2 的掩码 STATION_ADDR_MSK_HIGH_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0234, bit[15:0]) ; 这两个寄存器与步骤 1 中所配的前两个寄存器按位与的结果构成一个带掩码的 MAC 地址。配置本机 MAC 地址 1 寄存器 1 的掩码 STATION_ADDR_MSK_LOW_1 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0238, bit[31:0]) 最低位配置为 1, 其他位可选; 配置本机 MAC 地址 1 寄存器 2 的掩码 STATION_ADDR_MSK_HIGH_1 (基



地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x023C$ ，bit[15:0])；这两个寄存器与步骤 1 中所配的后两个寄存器按位与的结果构成一个带掩码的 MAC 地址。

步骤 17 配置本机 MAC 地址 2 寄存器 1STATION_ADDR_LOW_2 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x0210$ ，bit[31:0])，配置本机 MAC 地址 2 寄存器 2STATION_ADDR_HIGH_2 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x0214$ ，bit[15:0])；这两个寄存器的组合值为一个本机 MAC 源地址。配置本机 MAC 地址 3 寄存器 2STATION_ADDR_LOW_3 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x0218$ ，bit[31:0])，配置本机 MAC 地址 3 寄存器 2STATION_ADDR_HIGH_3 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x021C$ ，bit[15:0])；这两个寄存器的组合值为一个本机 MAC 源地址。配置本机 MAC 地址 4 寄存器 1STATION_ADDR_LOW_4 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x0220$ ，bit[31:0])，配置本机 MAC 地址 4 寄存器 2STATION_ADDR_HIGH_4 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x0224$ ，bit[15:0])；这两个寄存器的组合值为一个本机 MAC 源地址。配置本机 MAC 地址 5 寄存器 2STATION_ADDR_LOW_5 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x0228$ ，bit[31:0])，配置本机 MAC 地址 5 寄存器 2STATION_ADDR_HIGH_5 (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x022C$ ，bit[15:0])；这两个寄存器的组合值为一个本机 MAC 源地址。

步骤 18 配置单播帧过滤使能寄存器 REC_FILT_CONTROL[uc_drop_en] (基地址 $0x0CD00000+1000*ch_num$ ，偏移地址为 $0x0064$ ，bit[0]) 为 1，单播帧过滤使能。

----结束

18.5.3.3 接收发送相关 FIFO 水线值寄存器初始化配置

接收发送相关 FIFO 水线值寄存器初始化配置如下：

步骤 1 配置发送 FIFO 水线寄存器 CF_TX_FIFO_THRSLD[cf_tx_p_full_th] (基地址为 $0x0CD000000x1000*ch_num$ ，偏移地址为 $0x0420$ ，bit[24:16])，值为 $0x154$ ，指明发送报文数据 FIFO 的高水线。在不做 checksum 的情况下，如果发送数据 FIFO 中缓存数据个数大于该水线配置值，或者缓存数据个数未超过该水线，但已经缓存了完整的报文（不论是否做 checksum），此时启动向 MAC 发包。



须知

发送 FIFO 水线寄存器 CF_TX_FIFO_THRSLD[cf_tx_p_full_th] (基地址为 0x0CD000000x1000*ch_num, 偏移地址为 0x0420, bit[24:16]) 的值必须小于 0x1EE

- 步骤 2 配置发送 FIFO 水线寄存器 CF_TX_FIFO_THRSLD[cf_tx_p_empty_th] (基地址为 0x0CD000000x1000*ch_num, 偏移地址为 0x0420, bit[8:0]), 值为 0x0AA, 指明发送报文数据 FIFO 的低水线。在发送配置 FIFO 不为空的情况下, 如果发送数据 FIFO 中缓存数据个数小于该水线配置值, 此时启动从内存取包。
- 步骤 3 配置接收 FIFO 水线寄存器 CF_RX_FIFO_THRSLD[cf_rx_p_full_th] (基地址为 0x0CD000000x1000*ch_num, 偏移地址为 0x0424, bit[25:16]), 值为 0x240, 指明接收报文数据 FIFO 的高水线。当 FIFO 中的有效数据个数超过该水线时, 即停止向 FIFO 中写入新数据包。

须知

当接收报文数据 FIFO 数据个数超出高水线时, 会停止写入新的数据包, 当前数据包的数据会继续写入, 直到 FIFO 全满。

- 步骤 4 配置接收 FIFO 水线寄存器 CF_RX_FIFO_THRSLD[cf_rx_p_empty_th] (基地址为 0x0CD000000x1000*ch_num, 偏移地址为 0x0424, bit[9:0]), 值为 0x190, 指明接收报文数据 FIFO 的低水线。当 FIFO 中的有效数据个数超过该水线时, 开始将 FIFO 中的数据取出写入内存。
- 步骤 5 配置配置 FIFO 水线寄存器 CF_CFG_FIFO_THRSLD[cf_tx_p_full_th] (基地址为 0x0CD000000x1000*ch_num, 偏移地址为 0x0428, bit[31:24]), 值为 0x10, 指明发送配置 fifo 的满水线。当 FIFO 中的有效数据个数超过该水线时, 产生中断上报软件或者产生硬件将满指示信号通知 CPU 停止写入新的发送描述符。
- 步骤 6 配置配置 FIFO 水线寄存器 CF_CFG_FIFO_THRSLD[cf_tx_p_empty_th] (基地址为 0x0CD000000x1000*ch_num, 偏移地址为 0x0428, bit[23:16]), 值为 0x01, 指明发送配置 fifo 的空水线。当 FIFO 中的有效数据个数低于该水线时, 上报发送描述符将空状态供软件查询。
- 步骤 7 配置配置 FIFO 水线寄存器 CF_CFG_FIFO_THRSLD[cf_rx_p_full_th] (基地址为 0x0CD000000x1000*ch_num, 偏移地址为 0x0428, bit[15:8]), 值为 0x10, 指明接收配置 fifo 的满水线。当 FIFO 中的有效数据个数超过该水线时, 上报接收描述符的将满状态供软件查询。



步骤 8 配置配置 FIFO 水线寄存器 CF_CFG_FIFO_THRSLD[cf_rx_p_empty_th] (基地址为 0x0CD00000x1000*ch_num, 偏移地址为 0x0428, bit[7:0]), 值为 0x01, 指明接收配置 fifo 的空水线。当 FIFO 中的有效数据个数低于该水线时, 产生中断上报软件当前接收缓存地址不足。

----结束

18.5.3.4 接收报文控制相关寄存器初始化配置 (截断 (包括 MAC)、接收缓存大小、申请缓存方式、RX_CTRL)

接收报文控制相关寄存器初始化配置如下 (截断 (包括 MAC)、接收缓存大小、申请缓存方式、RX_CTRL):

- 步骤 1 配置接收超长帧界限寄存器 MAX_FRAME_LEN(基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0444, bit[14:0]), 值为 0x05EE, 设置接收数据帧的最大长度为 0x05EE 个字节。
- 步骤 2 配置最大帧长寄存器 MAX_FRM_SIZE(基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x003C, bit[15:0])值为 0x05EE, 设置接收数据帧的最大长度为 0x05EE 个字节。

须知

VLAN 帧允许的长度是该寄存器的配置值加上 4, 该寄存器最大可配值为 0xFFFFB, 大于则内部逻辑接收到 VLAN 帧时用该寄存器配置加 4 时内部逻辑会溢出异常。

不考虑流量影响时, 发送最大支持 64K 以下长度报文, 接受支持 32K 以下长度报文。

- 步骤 3 配置接收缓存大小寄存器 RX_BUF_SIZE[cf_rx_buf_size] (基地址为 0x0CD00000x1000*ch_num, 偏移地址为 0x04E4, bit[15:0]), 值为 0x0200, 指明接收缓存大小。
- 步骤 4 配置接收控制寄存器 RX_CTRL[cf_rx_get_addr_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[12]), 值为 0x1, 使能被动接收模式。
- 步骤 5 配置接收控制寄存器 RX_CTRL[cf_rxbuf_1st_skip_size] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[31:28]), 值为 0x0。指明第一块缓存区中报文描述符之前保留 0 个 cacheline。



- 步骤 6 配置接收控制寄存器 RX_CTRL[cf_rxbuf_no_1st_skip_size] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[27:24]), 值为 0x0。指明非第一个接收缓存首部保留 0 个 cacheline。
- 步骤 7 配置接收控制寄存器 RX_CTRL[cf_time_inf_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[23]), 值为 0x0。指明禁止时间戳上报。
- 步骤 8 配置接收控制寄存器 RX_CTRL[cf_rx_align_num] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[18:17]), 值为 0x2, 指明接收缓存中数据报文头部对齐的空白字节数为 2。
- 步骤 9 配置接收控制寄存器 RX_CTRL[cf_port_num] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[16:13]), 值为 0x0, 指明该 GMAC 的物理端口号为 0。
- 步骤 10 配置接收控制寄存器 RX_CTRL[cf_rxbuf_1st_skip_size2] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[3:0]), 值为 0x8, 指明第一个接收缓存首部报文描述符之后保留的 cacheline 个数为 8。

----结束

18.5.3.5 接收报文内部总线操作相关寄存器初始化配置 (cache 操作、cluster 通道选择)

接收报文内部总线操作相关寄存器初始化配置如下 (cache 操作、cluster 通道选择):

- 步骤 1 配置接收控制寄存器 RX_CTRL[cf_cache_line_h]和 RX_CTRL[cf_cache_line_l] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[9:8]和 bit[6:4]), 值为 0x0, 指明数据包首部可 cache 的 cacheline 个数为 0。

----结束

18.5.3.6 接收报文解析相关寄存器初始化配置 (RX_PKT_MODE、匹配等)

接收报文解析相关寄存器初始化配置如下 (RX_PKT_MODE、匹配等):

- 步骤 1 配置接收报文输入模式寄存器 RX_PKT_MODE[cf_instr_head_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F4, bit[31]), 值为 0x0 表示非指令头模式。



- 步骤 2 配置寄存器 RX_PKT_MODE[cf_skip_len] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F4, bit[29:23])。值为 0x00, 指明输入报文前 skip 区域的字节数为 0, 即指明报文输入进来就开始解析。
- 步骤 3 配置寄存器 RX_PKT_MODE[cf_parse_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F4, bit[22:21])。值为 0x1, 表示在非指令头模式下, 从 L2 开始解析。
- 步骤 4 配置寄存器 RX_PKT_MODE[cf_match_offset] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F4, bit[20:12]), 值为 0x000, 指明字符串匹配在报文中的起始偏移为 0。
- 步骤 5 配置寄存器 RX_PKT_MODE[cf_gen_id] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F4, bit[7:0]), 值为 0x00, 指明该描述符产生者的 ID 号为 0。
- 步骤 6 配置 QOS 信息生成模式寄存器 QOS_MODE[qos_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F8, bit[31:30]), 为 00 表示 Qos 的生成由软件配置指定。
- 步骤 7 配置寄存器 QOS_MODE[port_qos] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F8, bit[29:27]), 值为 0x0, 指定该网口的 Qos 的值为 0。
- 步骤 8 配置 Group 生成模式寄存器 GRP_MODE_0[grp_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0520, bit[2:0]), 为 000 表示由软件配置寄存器指定端口的 group。
- 步骤 9 配置寄存器 GRP_MODE_0[port_grp] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0520, bit[29:26]), 值为 0x0, 指定该端口的 group 值为 0。
- 步骤 10 配置寄存器 TT_MODE[tt_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0528), 值为 0x0, 表示由协议分析生成 TT。
- 步骤 11 配置 TT_MODE (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0528, bit[29:20]), 值为 0x0, 指明非 IP 报文、IPv4 下 TCP 报文、IPv4 下非 TCP 报文、IPv6 下 TCP 报文以及 IPv6 下非 TCP 报文的 TT 类型都为 0x0。
- 步骤 12 配置 TAG_MODE[tag_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x052C, bit[31]), 为 0 表示通过解析报文生成 TAG。
- 步骤 13 配置 TAG_MODE (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x052C, bit[30:25]), 30 到 25 位值分别为 0、1、1、1、1、1, 指明 tag 的[23:16]位为端口号; 源和目的 TCP/UDP 端口, IP 源和目的地址以及协议类型参与分析。



步骤 14 配置参与匹配的报文头部字节掩码寄存器，MATCH_MASK_0 (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0538), 值为 0x00000000, 指明报文头部不需要匹配。

----结束

18.5.3.7 发送报文控制相关寄存器初始化配置 (PAD、CRC 处理、流控帧、释放缓存失败超时、发送 pause 帧控制)

发送报文控制相关寄存器初始化配置如下 (PAD、CRC 处理、流控帧、释放缓存失败超时、发送 pause 帧控制):

步骤 1 配置接收控制寄存器 RECV_CONTROL[strip_pad_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01E0, bit[3]), 值为 1, 指明剥离接收帧的 PAD。

须知

只有不带 VLAN 的 802.3 帧才可以剥离 PAD, 且不可以只剥离 PAD 而不剥离 CRC。

步骤 2 配置 CRC 剥离使能寄存器 CF_CRC_STRIP[cf_crc_strip] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01B0, bit[1]), 值为 1, 指明剥离接收帧的 CRC。

步骤 3 配置发送控制寄存器 TRANSMIT_CONTROL[pad_enable] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0060, bit[7]), 值为 0, 指明发送帧的不添加 PAD。

步骤 4 配置发送控制寄存器 TRANSMIT_CONTROL[crc_add] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0060, bit[6]), 值为 1, 指明发送帧的添加 CRC。

步骤 5 配置发送完成释放缓存失败后的超时寄存器 TX_REL_TIMER (基地址 0x0CD00000+1000*h_num, 偏移地址 0x0588, bit[15:0]), 值为 0x0000, 指明释放缓存失败后不需要等待就可以再次释放。

步骤 6 发送流控控制寄存器 CF_TX_PAUSE (基地址 0x0CD00000+1000*h_num, 偏移地址 0x0470, bit[1]), 值为 0x0, 指明 MAC 根据实际流量决定是否发送流控帧, 当 PCU 的接收 FIFO 满或将满时, MAC 会自动发送流控帧。



----结束

18.5.4 接收报文

接收报文配置如下：

- 步骤 1 配置接收控制寄存器 RX_CTRL[cf_rx_get_addr_mode] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04F0, bit[12]) 为 1, 使能被动接收模式。
- 步骤 2 配置接收缓存大小寄存器 RX_BUF_SIZE[cf_rx_buf_size] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04E4, bit[15:0]), 指明接收缓存大小。
- 步骤 3 查询配置 FIFO 数据个数寄存器 CF_CFF_DATA_NUM[cf_rx_cfg_num] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x045C, bit[24:16]) 是否小于等于 127, 只有当满足该条件时, 才能继续下一步, 否则一直查询, 因为接收缓存地址 FIFO 最大只能存储 127 个接收缓存地址, 如果超出则会导致接收缓存地址丢失。
- 步骤 4 将接收缓存地址写入接收配置 FIFO 地址寄存器 RX_CFF_ADDR (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x04A0), 然后重复步骤 3。
- 步骤 5 配置中断状态寄存器 CF_INTRPT_STAT[stat_rx_frm] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0434, bit[27]), 值为 1, 打开收包中断。当接收包的时候, 接包时间达到设定的时间或接包数量达到设定数量的时候, 产生中断, 通知 CPU 读取处理。
- 步骤 6 配置中断状态寄存器 CF_INTRPT_CLR[cf_intrpt_pkt] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x042C, bit[11:6]), 接收中断的包个数上限, 即当接收到的报文个数由大于该配置值时, 对应中断 stat_rx_frm 有效。
- 步骤 7 配置中断状态寄存器 CF_INTRPT_CLR[cf_intrpt_time] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x042C, bit[5:0]), 接收时间上限, 超过该值则认为接收超时, 对应中断 stat_rx_frm 有效。

须知

寄存器 cf_intrpt_time 和 cf_intrpt_pkt 的配置值需要在配置 cf_rx_frm_clr 寄存器后才能生效。

- 步骤 8 配置中断清零寄存器 CF_INTRPT_CLR[cf_rx_buf_size] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x0438, bit[27]), 值为 1, 指明将步骤 6



的接收包个数寄存器清零和步骤 7 接收包时间寄存器清零。CPU 在接收到步骤 6 或步骤 7 的中断后，读取内存中的数据进行处理，执行本步骤。

----结束

18.5.5 发送报文

发送报文配置如下：

- 步骤 1 配置发送描述符 FIFO 地址寄存器 TX_CFF_ADDR_0[*sd*]（基地址 0x0CD00000+1000**ch_num*，偏移地址为 0x0488，bit[25]），为 0 时，指明帧只包含一个数据段，直接指向数据段；为 1 时，指明帧包含多个数据段，指向数据段列表。
- 步骤 2 配置发送描述符 FIFO 地址寄存器 TX_CFF_ADDR_0[*ds*]（基地址 0x0CD00000+1000**ch_num*，偏移地址为 0x0488，bit[24:19]），指明数据段的个数。
- 步骤 3 配置发送描述符 FIFO 地址寄存器 TX_CFF_ADDR_2[*send_addr*]（基地址 0x0CD00000+1000**ch_num*，偏移地址为 0x0490），指明发送数据缓冲地址。当采用 SegmentList 模式，这里指的是 SegmentList 的首地址，且必须按照 Cache-Line 对齐；如果是 DataBuffer 模式，则是数据缓存起始地址，该地址可以按照 Byte 对齐。
- 步骤 4 配置发送描述符 FIFO 地址寄存器 TX_CFF_ADDR_1[*cl*]（基地址 0x0CD00000+1000**ch_num*，偏移地址为 0x048C，bit[*fm*]），指明报文发送完成释放缓存空间。
- 步骤 5 配置发送描述符 FIFO 地址寄存器 TX_CFF_ADDR_0[*cl*]（基地址 0x0CD00000+1000**ch_num*，偏移地址为 0x0488，bit[1]），指明帧发送完成之后需要将指定地址清 0。
- 步骤 6 配置发送描述符 FIFO 地址寄存器 TX_CFF_ADDR_3[*wb_addr*]（基地址 0x0CD00000+1000**ch_num*，偏移地址为 0x0494，bit[*fm*]），指明发送数据完成，回写清 0 的地址。CPU 查询到该地址被清零，就知道可以再度利用其释放的缓存空间了。

----结束



18.6 DFX

在实际应用和调试中，GMAC 模块的相关状态可以很好地辅助系统问题定位，同时，对于 GMAC 模块自身，提供多种 Debug 的基本手段。本章节将分 PCU 和 MAC 两个部分详细描述 Debug 相关寄存器。

18.6.1 PCU 部分

18.6.1.1 发送接收总线错误地址查询

在接收发送总线出错时，可以通过下面两个寄存器查找出错的地址：

- 接收方向，当 GMAC 通过 AHB 总线向 DDR 中写数据出错，通过出错总线地址寄存器 TX_BUS_ERR_ADDR[tx_bus_err_addr]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x043C）可以读取出错地址，读完地址清零。
- 发送方向，当 GMAC 通过 AHB 总线从 DDR 中读数据出错。通过出错总线地址寄存器 RX_BUS_ERR_ADDR[rx_bus_err_addr]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0440）可以读取出错地址，读完地址清零。

18.6.1.2 统计计数寄存器

在 PCU 中，有多种计数器来反映当前运行状态：

- 发送 FIFO 空而丢弃的包统计计数寄存器 TX_DROP_CNT[tx_drop_cnt]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0448）。
- 接收 FIFO 满而丢弃的包统计计数寄存器 RX_OVER_FLOW_CNT[over_flow_cnt]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x044C）。
- 发送添加校验和失败统计计数寄存器 TX_CS_FAIL_CNT[tx_trans_fail_pkg_cnt]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0460）。
- 下行搬运成功的包计数寄存器 RX_TRANS_PKG_CNT[rx_trans_pkg_cnt]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0464）。
- 上行搬运成功的包计数寄存器 TX_TRANS_PKG_CNT[tx_trans_pkg_cnt]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0468）。
- 写下行配置 FIFO 时，由于 FIFO 满而未成功写入的地址个数寄存器 RX_ADDR_OVERFLOW[rx_addr_overflow]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x046C）。



- 接收方向申请缓存错误计数寄存器 RX_BUFRQ_ERR_CNT[rx_bufreq_err_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x058C)。
- 发送方向释放缓存错误计数寄存器 TX_BUFRL_ERR_CNT[tx_bufrl_err_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0590)。
- 接收方向写工作描述符错误计数寄存器 RX_WE_ERR_CNT[rx_we_err_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0594)。

18.6.1.3 状态机状态上报

接收发送状态机状态查询：

- 接收状态机状态寄存器 DEBUG_ST_MCH[rx_st_mch] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0450, bit[8:5])。
- 发送状态机状态寄存器 DEBUG_ST_MCH[tx_st_mch] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0450, bit[4:0])。

18.6.1.4 FIFO 状态查询

本节主要涉及 FIFO 相关的状态查询，分为 FIFO 当前水线状态、FIFO 历史水线状态和配置 FIFO 数据个数三个方面介绍。

FIFO 当前水线状态

通过读取下列寄存器可以了解当前发送接收配置 FIFO、发送接收 FIFO 水线状态：

- 发送配置 FIFO 的将满信号状态寄存器 FIFO_CURR_STATUS[cff_cf_tx_p_full] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0454, bit[15]), 高有效。
- 发送配置 FIFO 的满信号状态寄存器 FIFO_CURR_STATUS[cff_cf_tx_full] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0454, bit[14]), 高有效。
- 发送配置 FIFO 的将空信号状态寄存器 FIFO_CURR_STATUS[cff_cf_tx_p_empty] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0454, bit[13]), 高有效。
- 发送配置 FIFO 的空信号状态寄存器 FIFO_CURR_STATUS[cff_cf_tx_empty] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0454, bit[12]), 高有效。
- 接收配置 FIFO 的将满信号状态寄存器 FIFO_CURR_STATUS[cff_cf_rx_p_full] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0454, bit[11]), 高有效。



- 接收配置 FIFO 的满信号状态寄存器 FIFO_CURR_STATUS[cff_cf_rx_full] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[10]), 高有效。
- 接收配置 FIFO 的将空信号状态寄存器 FIFO_CURR_STATUS[cff_cf_rx_p_empty] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[9]), 高有效。
- 接收配置 FIFO 的空信号状态寄存器 FIFO_CURR_STATUS[cff_cf_rx_empty] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[8]), 高有效。
- 接收 FIFO 的将空信号状态寄存器 FIFO_CURR_STATUS[rff_cf_p_empty] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[7]), 高有效。
- 接收 FIFO 的空信号状态寄存器 FIFO_CURR_STATUS[rff_cf_empty] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[6]), 高有效。
- 接收 FIFO 的将满信号状态寄存器 FIFO_CURR_STATUS[rff_cf_p_full] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[5]), 高有效。
- 接收 FIFO 的满信号状态寄存器 FIFO_CURR_STATUS[rff_cf_full] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[4]), 高有效。
- 发送 FIFO 的将空信号状态寄存器 FIFO_CURR_STATUS[tff_cf_p_empty] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[3]), 高有效。
- 发送 FIFO 的空信号状态寄存器 FIFO_CURR_STATUS[tff_cf_empty] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[2]), 高有效。
- 发送 fifo 的将满信号状态寄存器 FIFO_CURR_STATUS[tff_cf_p_full] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[1]), 高有效。
- 发送 FIFO 的满信号状态寄存器 FIFO_CURR_STATUS[tff_cf_full] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0454$, bit[0]), 高有效。

FIFO 历史水线状态

通过读取下列寄存器可以了解历史发送接收配置 FIFO、发送接收 FIFO 水线状态：

- 发送配置 FIFO 的将满信号状态寄存器 FIFO_HIS_STATUS[cff_cf_tx_p_full_h] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0458$, bit[15]), 高有效, 读取之后清零。
- 发送配置 FIFO 的满信号状态寄存器 FIFO_HIS_STATUS[cff_cf_tx_full_h] (地址为 $0x0CD00000+1000*ch_num$, 偏移地址为 $0x0458$, bit[14]), 高有效, 读取之后清零。



- 发送配置 FIFO 的将空信号状态寄存器 FIFO_HIS_STATUS[cff_cf_tx_p_empty_h] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[13]), 高有效, 读取之后清零。
- 发送配置 FIFO 的空信号状态寄存器 FIFO_HIS_STATUS[cff_cf_tx_empty_h] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[12]), 高有效, 读取之后清零。
- 接收配置 FIFO 的将满信号状态寄存器 FIFO_HIS_STATUS[cff_cf_rx_p_full_h] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[11]), 高有效, 读取之后清零。
- 接收配置 FIFO 的满信号状态寄存器 FIFO_HIS_STATUS[cff_cf_rx_full_h] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[10]), 高有效, 读取之后清零。
- 接收配置 FIFO 的将空信号状态寄存器 FIFO_HIS_STATUS[cff_cf_rx_p_empty_h] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[9]), 高有效, 读取之后清零。
- 接收配置 FIFO 的空信号状态寄存器 FIFO_HIS_STATUS[cff_cf_rx_empty_h] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[8]), 高有效, 读取之后清零。
- 接收 FIFO 的将空信号状态寄存器 FIFO_HIS_STATUS[rff_cf_p_empty] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[7]), 高有效, 读取之后清零。
- 接收 FIFO 的空信号状态寄存器 FIFO_HIS_STATUS[rff_cf_empty] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[6]), 高有效, 读取之后清零。
- 接收 FIFO 的将满信号状态寄存器 FIFO_HIS_STATUS[rff_cf_p_full] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[5]), 高有效, 读取之后清零。
- 接收 FIFO 的满信号状态寄存器 FIFO_HIS_STATUS[rff_cf_full] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[4]), 高有效, 读取之后清零。
- 发送 FIFO 的将空信号状态寄存器 FIFO_HIS_STATUS[tff_cf_p_empty] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[3]), 高有效, 读取之后清零。



- 发送 FIFO 的空信号状态寄存器 FIFO_HIS_STATUS[tff_cf_empty] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[2]), 高有效, 读取之后清零。
- 发送 FIFO 的将满信号状态寄存器 FIFO_HIS_STATUS[tff_cf_p_full] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[1]), 高有效, 读取之后清零。
- 发送 FIFO 的满信号状态寄存器 FIFO_HIS_STATUS[tff_cf_full] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0458, bit[0]), 高有效, 读取之后清零。

FIFO 数据个数

- 接收配置 FIFO 中数据字个数统计寄存器 CF_CFF_DATA_NUM[cf_rx_cfg_num] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x045C, bit[24:16])。
- 发送配置 FIFO 中数据字个数统计寄存器 CF_CFF_DATA_NUM[cf_tx_cfg_num] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x045C, bit[8:0])。

18.6.1.5 Debug 寄存器

PCU 中有 3 个 Debug 寄存器, 用于检测 AHB 总线工作状态和接收发送 FIFO 中报文的字数量和帧数量。

AHB 总线工作状态检测

- 接收报文模块 AHBMASTER 接口的 Ready 信号寄存器 DBG_ST0[hready_m0_dbg] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[31])。
- 发送报文模块 AHBMASTER 接口的 Ready 信号寄存器 DBG_ST0[hready_m1_dbg] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[30])。
- 接收报文模块 AHBMASTER 接口的 burst 信号寄存器 DBG_ST0[rx_bus_burst] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[29:27])。
- 接收报文模块 AHBMASTER 接口的 prot 信号寄存器 DBG_ST0[rx_bus_prot] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[26:23])。
- 接收报文模块 AHBMASTER 接口的 size 信号寄存器 DBG_ST0[rx_bus_size] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[22:20])。



- 接收报文模块 AHBMASTER 接口的 incr_num 信号寄存器 DBG_ST0[rx_bus_incr_num] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[19:15])。
- 发送报文模块 AHBMASTER 接口的 burst 信号寄存器 DBG_ST0[tx_bus_burst] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[14:12])。
- 发送报文模块 AHBMASTER 接口的 prot 信号寄存器 DBG_ST0[tx_bus_prot] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[11:8])。
- 发送报文模块 AHBMASTER 接口的 size 信号寄存器 DBG_ST0[tx_bus_size] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[7:5])。
- 发送报文模块 AHBMASTER 接口的 incr_num 信号寄存器 DBG_ST0[tx_bus_incr_num] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E4, bit[4:0])。
- 发送状态机读报文数据时一次 burst8 读操作中完成的读数据的个数寄存器 DBG_ST1[trans_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E8, bit[22:20])。

发送接收 FIFO 报文的字数量和帧数量统计

- 发送 FIFO 报文字计数寄存器 DBG_ST1[tx_fifo_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E8, bit[19:10])。
- 发送 FIFO 帧计数寄存器 DBG_ST1[txff_frame_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05E8, bit[9:0])。
- 接收 FIFO 报文字计数寄存器 DBG_ST2[fifo_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05EC, bit[21:11])。
- 接收 FIFO 帧计数寄存器 DBG_ST2[frame_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x05EC, bit[10:0])。

18.6.2 MAC 部分

18.6.2.1 SGMII 自协商状态查询

- 链接状态寄存器 AN_NEG_STATE[link_ok] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0058, bit[15])。当值为 0 时, 表示链接失败; 当值为 1 时, 表示链接成功。



- 工作模式寄存器 AN_NEG_STATE[duplex] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0058, bit[12])。当值为 0 时, 表示半双工; 当值为 1 时, 表示全双工。
- 链接速度寄存器 AN_NEG_STATE[speed] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0058, bit[11:10])。当值为 0b11: 保留; 值为 0b10 时, 速度为 1000M; 值为 0b01 时, 速度为 100M; 值为 0b00 时, 速度为 10M。

18.6.2.2 统计计数寄存器

MAC 中有多个统计计数器用于查询运行状态:

接收方向

- 接收有效帧 (范围包括 DA ~ FCS) 的字节统计寄存器 RX_OCTETS_TOTAL_OK[rx_octets_total_ok] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0080)。
- 接收错误帧 (包括 CRC 错误, 对齐错误等) 字节统计寄存器 RX_OCTETS_BAD[rx_octets_bad] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0084)。
- 接收单播帧数统计 (不包括 bad 帧) 寄存器 RX_UC_PKTS[rx_uc_pkts] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0088)。
- 接收多播帧数统计 (不包括 bad 帧) 寄存器 RX_MC_PKTS[rx_mc_pkts] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x008C)。
- 接收广播帧数统计 (不包括 bad 帧) 寄存器 RX_BC_PKTS[rx_bc_pkts] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0090)。
- 接收帧长为 64Byte 的帧数统计 (包括 bad 帧) RX_PKTS_64OCTETS[rx_pkts_64octets] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0094)。
- 接收帧长为 65Byte ~ 127Byte 的帧数统计 (包括 bad 帧) RX_PKTS_65TO127OCTETS[rx_pkts_65to127octets] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0098)。
- 接收帧长为 128Byte ~ 255Byte 的帧数统计 (包括 bad 帧) RX_PKTS_128TO255OCTETS[rx_pkts_128to255octets] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x009C)。



- 接收帧长为 256Byte ~ 511Byte 的帧数统计 (包括 bad 帧)
RX_PKTS_255TO511OCTETS[rx_pkts_256to511octets] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00A0)。
- 接收帧长为 512Byte ~ 1023Byte 的帧数统计 (包括 bad 帧)
RX_PKTS_512TO1023OCTETS[rx_pkts_512to1023octets] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00A4)。
- 接收帧长为 1024Byte ~ 1518Byte 的帧数统计 (包括 bad 帧)
RX_PKTS_1024TO1518OCTETS[rx_pkts_1024to1518octets] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00A8)。
- 接收帧长为 1519Byte ~ 最大 Byte 的帧数统计 (包括 bad 帧)
RX_PKTS_1519TOMAXOCTETS[rx_pkts_1519tomaxoctes] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00AC)。
- CRC 检测错误的帧数统计 (不包括短帧) RX_FCS_ERRORS[rx_fcs_errors] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00B0)。
- 接收带 TAG 的帧统计寄存器 RX_TAGGED[rx_tagged] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00B4)。
- 接收数据错误帧统计寄存器 RX_DATA_ERR[rx_data_err] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00B8)。
- 接收数据非字节对齐 (10M/100M 模式时接收到的帧长不是字节的整数倍) 错误帧统计寄存器 RX_ALIGN_ERRORS[rx_align_errors] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00BC)。
- CRC 正确且无非对齐错误的情况下, 帧长大于 max_frm_size 且不大于 2 倍的 max_frm_size 的帧数统计寄存器 RX_LONG_ERRORS[rx_long_errors] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00C0)。
- CRC 错误或 Byte 数不是整数的情况下, 帧长大于允许的最大长度的帧数统计寄存器 RX_JABBER_ERRORS[rx_jabber_errors] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00C4)。
- CRC 错误或非对齐错误情况下, 帧长大于 max_frm_size 且不大于 2 倍的 max_frm_size 的帧数统计寄存器
RX_PAUSE_MACCONTROL_FRAMCOUNTER[rx_pause_maccontrol_framecounter] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00C8)。
- 接收到的非流控控制帧帧数统计寄存器
RX_UNKNOWN_MACCONTROL_FRAMCOUNTER[rx_unknown_maccontrol_framecounter] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00CC)。



- 帧长大于 2 倍的 max_frm_size 的帧数统计寄存器
RX_VERY_LONG_ERR_CNT[rx_very_long_err_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00D0)。
- 帧长小于 64Byte, 但大于或等于 12Byte 的帧数统计寄存器
RX_RUNT_ERR_CNT[rx_runt_err_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00D4)。
- 帧长小于 12Byte 的帧数统计寄存器 RX_SHORT_ERR_CNT[rx_short_err_cnt]
(基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00D8)。
- 接收方向被过滤的包统计寄存器 RX_FILT_PKT_CNT[rx_filt_pkt_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00E8)。
- 接收方向被过滤的字节统计寄存器
RX_OCTETS_TOTAL_FILT[rx_octets_total_filt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x00EC)。
- GMAC 应用侧接收 FIFO 溢出次数统计寄存器
RX_OVERRUN_CNT[rx_overrun_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x01EC)。
- GMAC 剥离 PAD 使能时, 接收到帧长不等于 64 的含 PAD 的帧的个数统计寄存器
RX_LENGTHFIELD_ERR_CNT[rx_lengthfield_err_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x01F4)。
- 字节定界过程中同时出现两个或者两个以上 comma 码的个数统计寄存器
RX_FAIL_COMMA_CNT[rx_fail_comma_cnt] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x01F8)。

发送方向

- 发送成功的正确帧字节数统计寄存器
OCTETS_TRANSMITTED_OK[octets_transmitted_ok] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0100)。
- 发送成功的错误帧字节数统计寄存器
OCTETS_TRANSMITTED_BAD[octets_transmitted_bad] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0104)。
- 发送的单播帧数统计 (不包括 bad 帧) 统计寄存器 TX_UC_PKTS[tx_uc_pkts]
(基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x0108)。
- 发送的多播帧数统计 (不包括 bad 帧) 统计寄存器 TX_MC_PKTS[tx_mc_pkts]
(基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x010C)。



- 发送的广播帧数（不包括 bad 帧）统计寄存器 TX_BC_PKTS[tx_bc_pkts]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0110）。
- 发送的帧长为 64Byte 的帧数统计（包括 bad 帧）统计寄存器 TX_PKTS_64OCTETS[tx_pkts_64octets]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0114）。
- 发送帧长为 65Byte ~ 127Byte 的帧数统计（包括 bad 帧）统计寄存器 TX_PKTS_65TO127OCTETS[tx_pkts_65to127octets]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0118）。
- 发送帧长为 128Byte ~ 255Byte 的帧数统计（包括 bad 帧）统计寄存器 TX_PKTS_128TO255OCTETS[tx_pkts_128to255octets]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x011C）。
- 发送帧长为 256Byte ~ 511Byte 的帧数统计（包括 bad 帧）统计寄存器 TX_PKTS_255TO511OCTETS[tx_pkts_256to511octets]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0120）。
- 发送帧长为 512Byte ~ 1023Byte 的帧数统计（包括 bad 帧）统计寄存器 TX_PKTS_512TO1023OCTETS[tx_pkts_512to1023octets]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0124）。
- 发送帧长为 1024Byte ~ 1518Byte 的帧数统计（包括 bad 帧）统计寄存器 TX_PKTS_1024TO1518OCTETS[tx_pkts_1024to1518octets]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0128）。
- 发送帧长大于等于 1519Byte 的帧数（包括 bad 帧）统计寄存器 TX_PKTS_1519TOMAXOCTETS[tx_pkts_1519tomaxoctes]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x012C）。
- 发送帧长超过设定的最大帧长 max_frm_siz 而导致被截断的帧统计寄存器 TX_EXCESSIVE_LENGTH_DROP[tx_excessive_length_drop]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x014C）。
- 帧发送过程中发生内部错误而导致发送失败的次数统计寄存器 TX_UNDERRUN[tx_underrun]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0150）。
- 发送的 VLAN 帧数统计寄存器 TX_TAGGED[tx_tagged]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0154）。
- 发送帧长正确而 CRC 错误的帧数目统计寄存器 TX_CRC_ERROR[tx_crc_error]（基地址为 0x0CD00000+1000*ch_num，偏移地址为 0x0158）。



- 发送 PAUSE 帧数统计寄存器 TX_PAUSE_FRAMES[tx_pause_frames] (基地址为 0x0CD00000+1000*ch_num, 偏移地址为 0x015C)。

须知

上述接收或者发送报文统计个数寄存器在软件配置读清的同时, 如果有统计事项发生, 会导致该统计事项被漏统计, 因此不建议对上述报文统计寄存器的统计值进行连续读取并累加作为精确累计值。

18.6.3 环回测试

Hi3093 提供了应用侧环回和线路侧环回 2 种方法用于测试数据通路。应用侧环回可以用来定位芯片内部模块是否正常工作; 线路侧环回可以用来测试芯片外的设备是否正常工作。

须知

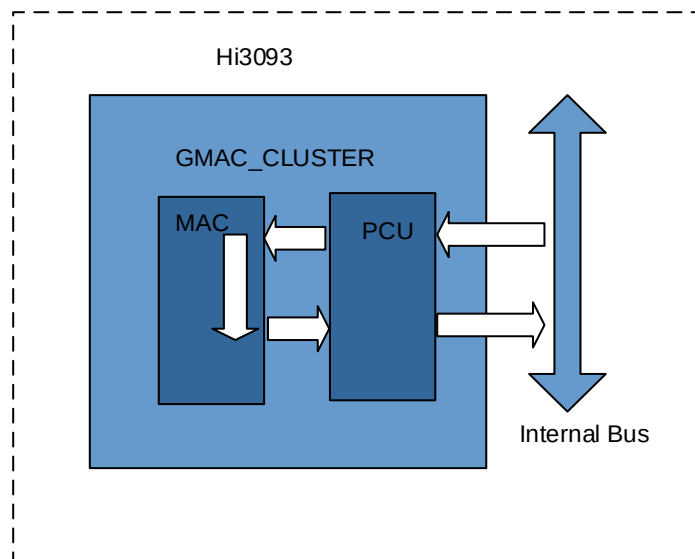
半双工模式不可以使用环回测试。

18.6.3.1 应用侧环回

如图 18-14 所示是应用侧环回的示意图, 数据流方向:

InternalBus->PCU->MAC->PCU->InternalBus, 通过应用侧环回可以测试数据流通过的模块是否正常工作。

图18-14 应用侧环回示意图

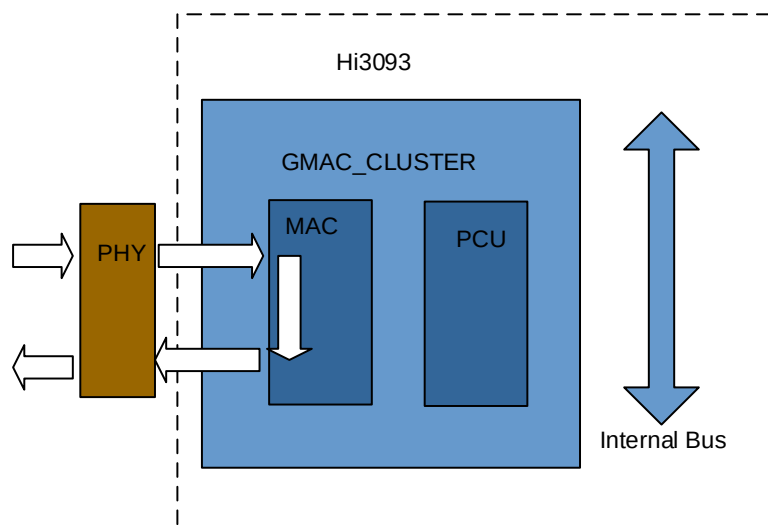


配置环回补充寄存器 LOOP_REG[cf2mi_lp_en] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01DC, bit[2]) 为 1。

18.6.3.2 线路侧环回

如图 18-15 所示是线路侧环回的示意图，数据流方向：对接设备 ->PHY->MAC->PHY->对接设备，通过线路侧环回可以测试外部设备、PHY 和 MAC 的配置是否正确、硬件连接是否正常等。

图18-15 线路侧环回示意图





- 步骤 1 配置环回补充寄存器 LOOP_REG[cf_ext_drive_lp] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01DC, bit[1]) 为 0。
- 步骤 2 配置线路侧环回寄存器 LINE_LOOP_BACK[line_loop_back] (基地址 0x0CD00000+1000*ch_num, 偏移地址为 0x01A8, bit[0]) 为 1。

----结束



19 GPIO

19.1 概述

GPIO (General Purpose Input/Output) 为通用输入输出接口。Hi3093 共有 5 组 GPIO 接口：分别为 GPIO0、GPIO1、GPIO2、GPIO3、GPIO4。

GPIO 每个管脚可以配置为输入或者输出。这些管脚用于生成特定应用的输出信号或者采集特定应用的输入信号。作为输入管脚时，可作用为中断源；作为输出管脚时，每个 GPIO 都可以独立地写 0 或写 1。大部分 GPIO 管脚与其他管脚复用，管脚复用说明请参见《Hi3093 寄存器手册》中的 IOCONFIG 寄存器配置说明。芯片上电默认所有 GPIO 为输入。

支持与 I2C, LocalBus, PWM 等慢速 IO 复用。

19.2 特性

GPIO 接口具有以下特性：

- 提供五组 GPIO 接口，每组可提供 32 个 GPIO。
- 对外提供 129 个 GPIO，其中 2 个支持安全属性。
- 其中 GPIO0~2 对外分别提供 32 路，GPIO3 对外提供 31 路 (0~30)，GPIO4 对外提供 2 路 GPIO (30~31) 和安全 GPIO 的 2 路 (0~1) 进行了复用。
- 支持串行数据总线和低于 32bit 的并行数据总线的模拟。
- 每根信号有独立的数据寄存器和输入输出控制寄存器。上电复位时默认为输入模式。



- 支持中断查询和中断屏蔽。
- 支持中断上报。
- GPIO_INTTYPE_LEVEL（中断类型寄存器）和 GPIO_INT_POLARITY（中断触发极性控制寄存器）两个寄存器共同决定了中断源的特性和触发类型。
- 当 GPIO 作为输入管脚时，可作为中断源，每个 GPIO 管脚都具有独立的中断控制。
- 当 GPIO 作为输出管脚时，每个 GPIO 管脚支持独立输出使能。
- 当 GPIO 作为输入管脚时，GPIO 支持滤毛刺功能。在芯片使用 25MHz 晶振时钟的场景下，可滤除宽度小于 2μs 的脉冲。

须知

支持安全属性的 GPIO，为 GPIO4 的 30~31 和 Sec GPIO 的 0~1 进行复用；
当寄存器配置成安全模式的时候，选择 Sec GPIO[1:0]；
当寄存器配置成非安全模式的时候，选择 GPIO4[31:30]。具体请参见 0 章节。

19.3 关键术语

表19-1 GPIO 关键术语描述

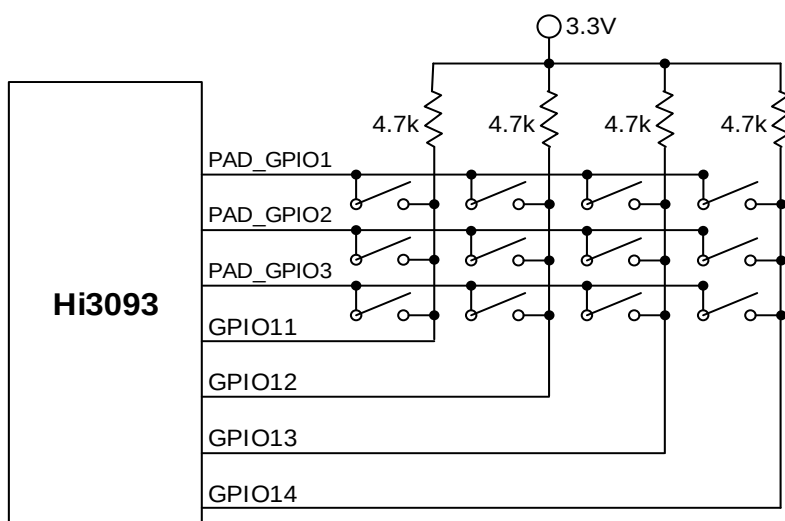
术语	描述
GPIO	General Purpose Input/Output，通用输入输出接口。当微控制器或芯片组没有足够的 I/O 端口，或当系统需要采用远端串行通信或控制时，GPIO 产品能够提供额外的控制和监视功能。



19.4 功能

19.4.1 典型应用框图

图19-1 GPIO 模拟键盘的典型应用框图



GPIO[3:1]的引脚作为行选择，输出低电平；GPIO[14:11]的引脚作为列选择，初始为高电平，接收输入。

确定按键的列值：当按键被按下时，行选择和列选择连通，引起GPIO[14:11]的某一引脚由高变低，触发中断，可以确定GPIO[14:11]的某一输入被按下，即确定按键的行值。

确定按键的行值：GPIO[3:1]行选择的3个引脚轮流发送低电平，扫描按键的行值。这样行和列选择就可以唯一的确定一个按键。



19.5 工作方式

19.5.1 时钟门控

上电时两个 GPIO 接口的时钟默认为打开状态。GPIO 工作完成后，可以关闭 GPIO 时钟，以降低芯片功耗。GPIO0、GPIO1、GPIO2、GPIO3 和 GPIO4 使用不同的门控信号，即它们的时钟可独立打开或关闭。

19.5.1.1 打开门控时钟

以 GPIO0 为例，操作 LP_SUBSYS 的寄存器，具体配置步骤如下：

步骤 1 配置 LPSC_CLKEN_CTRL0[gpio0_clk_enb]为 1，即打开 GPIO 控制器的时钟门控。

步骤 2 读 LP_SUBSYS 寄存器 LPSC_CLKSTA_CTRL0[gpio0_clk_status]，获取当前时钟门控信息。

若 gpio0_clk_status 为 0，则回到[步骤 1](#)；

若 gpio0_clk_status 为 1，则表示 GPIO 控制器时钟已经打开，结束操作。

----结束

19.5.1.2 关闭门控时钟

以 GPIO0 为例，操作 LP_SUBSYS 的寄存器，具体配置步骤如下：

步骤 1 配置 LPSC_CLKDIS_CTRL0[gpio0_clk_dis]为 0，即关闭 GPIO 控制器模块的时钟门控。

步骤 2 读 LP_SUBSYS 寄存器 LPSC_CLKSTA_CTRL0[gpio0_clk_status]，获取当前时钟门控信息。

若 gpio0_clk_status 为 1，则回到[步骤 1](#)；

若 gpio0_clk_status 为 0，则表示 GPIO 控制器时钟已经关闭，结束操作。

-----结束



19.5.2 管脚复用配置

GPIO 大部分均与其他管脚复用，使用 GPIO 前，需要配置 IOCONFIG 复用配置寄存器使能相应管脚的 GPIO 功能。

管脚复用方式请参见《Hi3093 硬件设计指南》管脚复用相关章节。

19.5.3 初始复位

说明

下面以 GPIO0 为例进行说明，其他 GPIO 只是基地址不一样。

上电复位时，所有管脚默认为输入。

复位信号有效时，GPIO 有如下反应：

- 所有的寄存器被清零。
- 所有管脚都被配置为输入。
- 所有中断被清除。
- 中断类型被设为电平触发。

19.5.4 输入输出配置

GPIO 的输入输出方向和输出数据均由软件控制，管脚的输入信号也通过软件读取 GPIO_EXT_PORT bit[31:0]来获得。

输入输出模式配置步骤如下：

步骤 1 选择要使用的 GPIO 管脚，配置 IOCONFIG 复用配置寄存器的相应管脚复用控制寄存器，选择 GPIO 功能，参见 IOCONFIG 复用配置寄存器的寄存器配置说明。

步骤 2 配置寄存器 GPIO_SWPORT_DDR，配置 GPIO 是输入还是输出。



说明

- 当 GPIO 用作输入时，外部信号通过 GPIO 管脚送进来，可通过寄存器 GPIO_EXT_PORT 读取输入信号值。
- 当 GPIO 用作输出时，通过写寄存器 GPIO_SWPORT_DR，GPIO 会将写进此寄存器的值通过 GPIO 输出。需要注意的是，此时若 GPIO 中断处于使能状态，则当输出信号满足触发条件时会产生中断。

---结束

19.5.5 中断处理

须知

每个 GPIO 有一个中断控制寄存器。当有一个或多个 GPIO 管脚产生了中断时，GPIO 汇聚成一个中断送到中断控制器。对于边沿触发的中断，软件必须清除该中断以使能其他的中断；对于电平触发的中断，外部中断源应该保持该电平直到处理器识别到该中断。

设置 GPIO 中断为电平触发时，如果配置该中断为异步中断，则该中断仍然可以在 GPIO 时钟关闭时传递中断到 GIC，如果配置该中断为同步中断，则该中断在 GPIO 时钟关闭时无法传递到 GIC。

GPIO 中断处理的配置步骤如下：

步骤 1 选择要使用的 GPIO 管脚，配置该管脚为输入（配置方法请参见 [19.5.4 输入输出配置](#)）。

步骤 2 保证 GPIO 数据线在这些操作的过程中保持稳定。

步骤 3 配置 GPIO_INTTYPE_LEVEL 对应位，选择边沿触发或电平触发。

- 如果选择电平触发，配置 GPIO_INT_POLARITY 对应位，选择高电平或低电平触发。否则跳过此步。
- 如果选择了边沿触发，配置 GPIO_INT_POLARITY 对应位，选择上升沿或下降沿触发。否则跳过此步。

步骤 4 通过往寄存器 GPIO_PORTA_EOI 对应位写 1，清中断。

步骤 5 配置 GPIO_INMASK，关闭中断屏蔽。



步骤 6 配置 GPIO_INTEN，使能中断。

步骤 7 发生中断时，查询 GPIO_INTSTATUS，获取中断信息。

步骤 8 中断处理完成。往寄存器 GPIO_PORTA_EOI 对应位写 1，清除中断。

----结束

19.5.6 毛刺滤除

配置步骤如下：

步骤 1 选择要使用的 GPIO 管脚，并配置为输入（配置方法请参见 [19.5.4 输入输出配置](#) 章节）。

步骤 2 配置 GPIO_DEBOUNCE 对应位为 1，使能毛刺滤除功能。

----结束



20 PCIe

20.1 概述

PCIe 接口是芯片的外部接口，提供高速、高性能、点到点、全双工（Dual Simplex）、串行差分信号链路来与片外设备进行互连通讯。通过该芯片 PCIe 接口可以实现对片外设备的存储空间（32bit 或者 64bit）、PCIe 配置空间进行访问。

Hi3093 芯片一共有 2 个 PCIe 2.0 X1 控制器，PCIe 接口控制器是一个 Dual Mode 控制器，支持 EP 工作模式，该工作模式需要在初始化的时候配置，不可动态切换。

20.2 特性

Dual Mode PCIe 控制器具有以下特性：

- 支持静态配置为 EP 和 RC 模式；默认配置是 EP 模式。
- 支持 PCI Express GEN1（2.5Gbps）、GEN2（5 Gbps）规范标准。
- 提供 AXI 总线接口，可以通过该数据访问接口访问 PCIE 外设。
- 提供 AXI 配置访问接口访问 PCIe 接口内部寄存器空间，并且支持内部地址转换机制（iATU）。
- 支持最大 Lane 数目 X1。
- 支持 MSI 和传统 Int 中断上报（EP 模式）。
- 支持 MSI 和传统 Int 中断接收（RC 模式）。
- 支持 CFG0 和 CFG1 类型配置空间。
- 支持 32 位存储器地址空间访问。
- 自带 DMA 功能。



- 支持链表访问模式;
 - 支持 32bit CPU 地址位宽访问;
 - 支持 64 bit CPU 数据位宽访问;
 - 独立于 PCI 控制器和 CPU 总线时钟;
 - 全双工模式, 读写事务可以同时进行。
- 其中 PCIE1 支持 MCTP。
 - 数据包最大支持 256 Bytes 的载荷。
 - 提供各种校验错的状态信息, 供诊断。
 - 要求 PCIe 输出关键的状态信号, 例如工作正常标记, 严重错误或故障等。
 - PCIe 协议在 TLP (事务层) 和 DLP (数据层) 两层都有 CRC 相关保护机制。
 - 其中 DLP 的 CRC 保护无论如何都应当存在, TLP 的 CRC 保护则取决于 CPU 选择的帧结构;
 - 要完成保护功能, 首先要求软件支持; 另外, 在 Serdes 接收端的软硬核当中, DLP 的校验始终打开, TLP 的校验需要配置打开 (由软件配置决定)。
 - 提供正常和各种故障的统计、查询、清除功能; 支持丢包信息记录功能, 各接口提供错误登记功能, 区分 8B/10B、错误, Parity 错误, 包 CRC 错误等指示。
 - 提供关键模块独立复位功能: 独立的复位寄存器; 提供手动、自动复位。
 - 集成 2 个 PCIe 控制器, 支持 PCIE2.0 X1, 独立 SerDes。
 - 支持 pcie 卡的热插拔 (linkdown), 不需要支持协议热插拔。
 - 其中一路专用于对接 GPU, 默认 EP 模式。另外一路支持 PCIE 的 RC 和 EP 模式可配置, RC 模式下, 用于 SMM 板上对接 LSW (只做管理用), 存储控制器对接 LSW。EP 模式下用于黑匣子、虚拟网口、MCTP、EDMA。EP 模式下用于黑匣子、虚拟网口、MCTP、EDMA。
 - PCIE BAR0/BAR1 地址空间大小可配置 (场景: 做 EP, 地址空间可配置缩小)。
 - 2 个 PCIE 控制器时钟分别控制, 不绑定。
 - PCIE 做 RC 的地址空间要求 64MB。
 - PCIe 控制器预留 BAR 供黑匣子业务使用, host 通过 PCIe 可把数据缓存到内存中, 循环覆盖。BAR 为 memory 属性。
 - 主控端 CPU 和 Hi3093 均能读取黑匣子内数据。
 - 黑匣子的 BAR 空间可配最大为 128MB, 默认 128MB。



- 安全核可以访问主控端 CPU 通过 PCIE 写到 DDR 的内容，该 PCIE 通道占用独立 BAR。
- 支持内环回和外环回测试。

须知

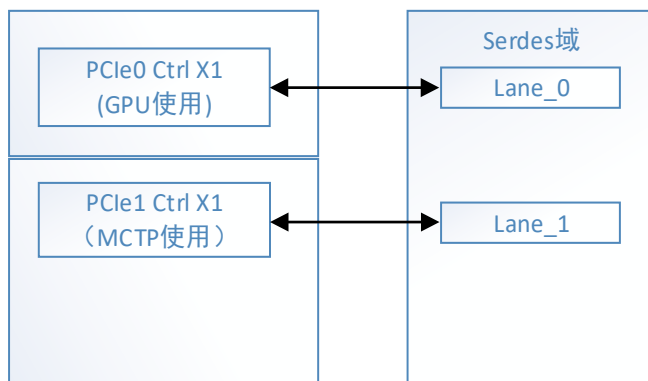
1. 不支持软件配置 PCI Express 电源管理机制和硬件自动状态电源管理机制 (ASPM)。
2. iATU 地址转换对地址有 64KB 对齐要求。
3. A55 和 M3 访问 PCIE DM 寄存器只支持 single 操作。A55 对外设寄存器空间的访问必须是最严格的 nGnRnE。

20.3 功能描述

20.3.1 应用框图

Hi3093 共支持 2 个 X1 的 PCIE 控制器 (PCIE0, PCIE1)，每个 PCIE 控制器可以互相独立，都通过 AXI 总线与系统总线互连。

图20-1 PCIe 结构图



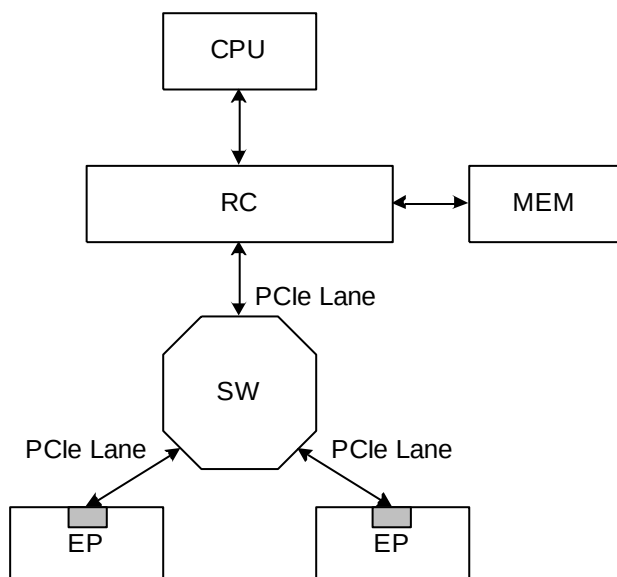
20.3.2 PCIe 系统体系结构

PCIe 是一种能够应用于移动设备、台式电脑、工作站、服务器、嵌入式计算机和通信设备等周边 I/O 设备互连的总线，属于第三代高性能 I/O 总线。能够提供高速、高性



能、点到点、双单工 (Dual Simplex)、串行、差分信号链路来进行设备互连。一个基于 PCIe 互连的典型系统体系架构如图 20-2 所示。

图20-2 基于 PCIe 系统体系架构

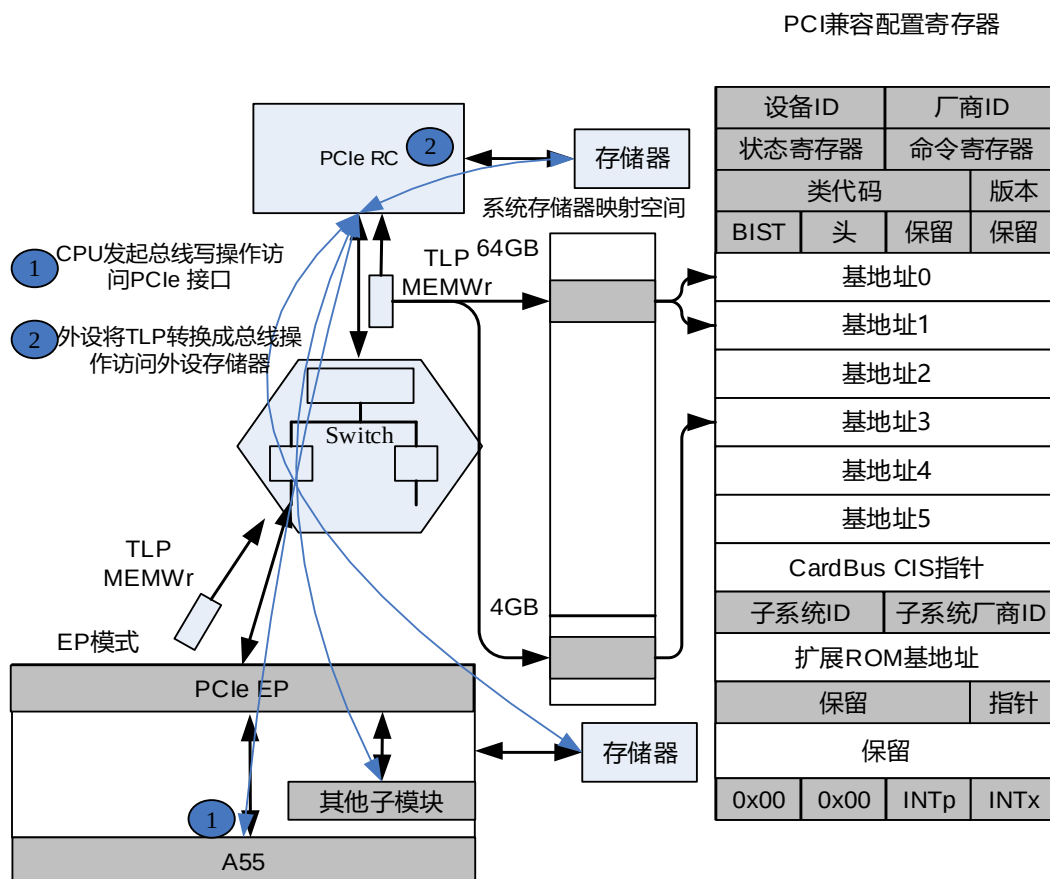


Hi3093 默认是 EP 模式的设置。Hi3093 PCIe 接口不同模式下的应用如图 20-3 所示。



20.3.3 PCIe 接口应用框图

图20-3 PCIe 接口 EP 模式应用框图



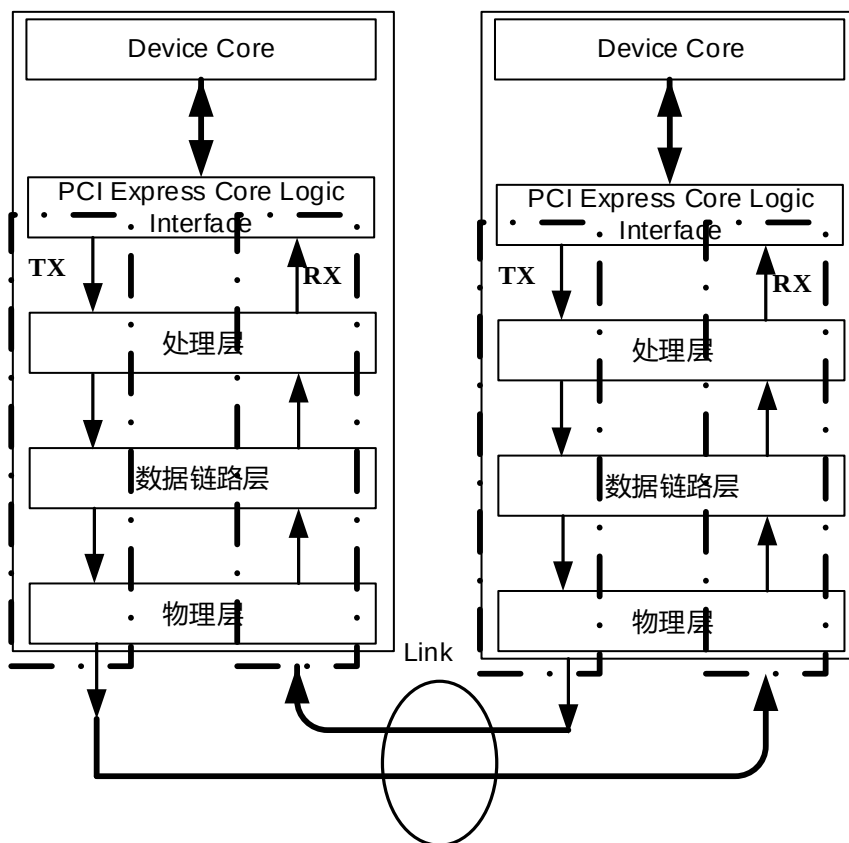
当 PCIe 接口作为 EP，对接器件要发起存储器读/写、IO 读/写操作前，需要 RC（外设）首先配置 Hi3093 的 PCI 配置头的 BAR0 ~ BAR5 寄存器，将存储器读写、IO 读写的基地址写入。其过程也是如图 20-3 所示。CPU 通过 PCIe 接口主动发起存储器写事务（MEMWr）访问外设存储器。首先，CPU 发起写操作访问 PCIe 接口，PCIe 接口会自动将该写操作转换成 MEMWr 的方式请求发送 TLP 访问外设。TLP 头字段里面包含了 PCIe 接口作为请求者的 ID，目标地址以及请求数据的大小。TLP 会先到达交换机，由交换机解码 TLP 中的存储器地址，如果地址匹配，则该 TLP 会被转发至对接器件接口，再由对接器件将该 TLP 转换成总线写操作访问存储器。

20.3.4 数据结构

PCI Express 规范定义了四层结构。为了保持 PCI Express 和目前主要硬件驱动程序以及应用程序的完全兼容。PCI Express 仍然采用了兼容 PCI 总线寻址的方式，并使

用了与定义 PCI 设备即插即用特性一样的标准规范来定义 PCI Express。PCI Express 规范定义的四层结构如图 20-4 所示。

图20-4 PCIe 分层结构示意图



在 PCI Express 结构中，首先由软件层产生数据的读写请求（如图 20-5 所示），然后数据将被传送到处理层（Transaction），处理层对数据进行封装之后再传到数据链路层（Data Link）。而数据链路层会为已封装的数据增加时序号（Sequence Number）和 CRC 纠错信号，以增加传输数据的正确性和可靠性，然后数据就到达最底层的物理层（Physical Layer），并通过物理层传送至另一个 PCI Express 设备，并各层进行数据包逆向（拆解）过程（如图 20-6 所示）。

图20-5 PCIe TLP 创建流程示意图

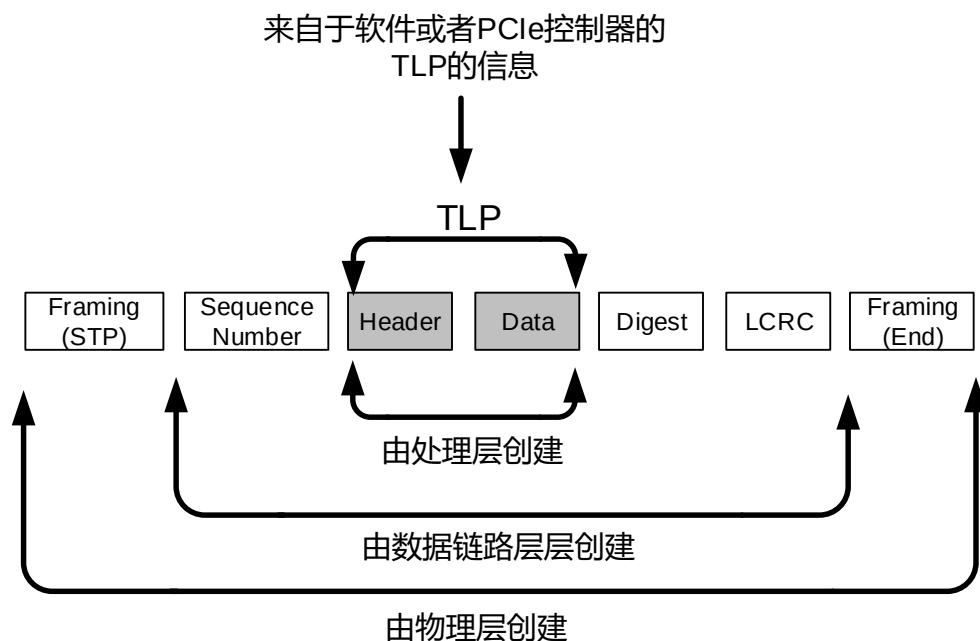
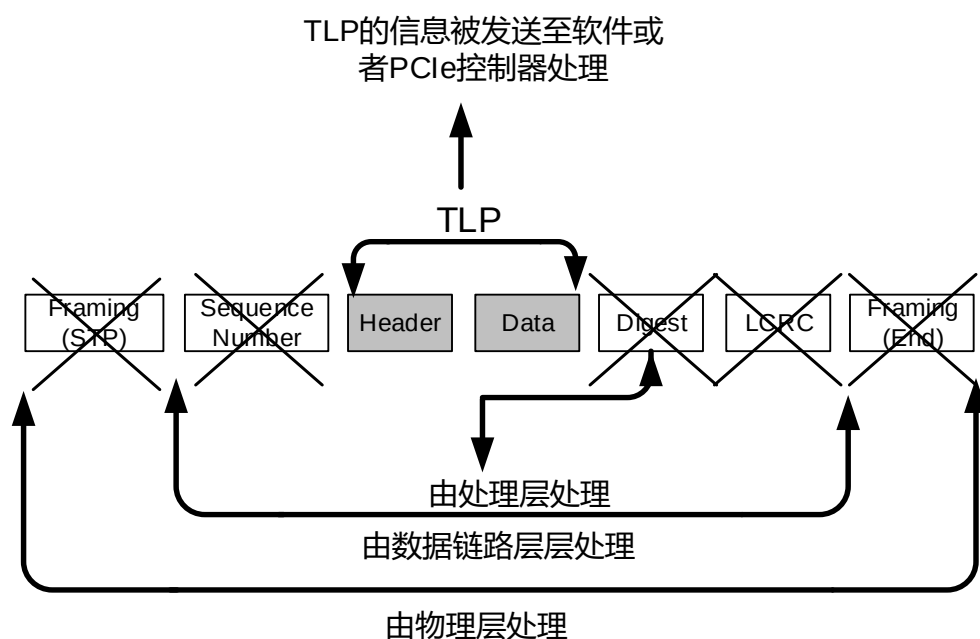


图20-6 PCIe TLP 处理流程示意图



- 物理层 (Physical Layer)



和局域网 OS 模型的物理层相似，PCI Express 总线架构中的物理层主要决定总线的物理特征。在未来的发展中，可以通过速度提升或更改编解码方式来进一步提升 PCI Express 总线性能，而这些变更都只会作用于物理层，不会影响其他结构，为升级带来了便利。物理层还有一个重要作用就是处理系统中断请求，PCI Express 结构的物理层将兼容两种中断模式，即 INTx 模式和 MSI 模式。其中 INTx 源自 PCIe 规范并予以保留，MSI (Message Signaled Interrupt, 信号信息中断) 中断方式将不必经由 CPU 请求而能够由各种外设直接发出，这样就比由主处理系统发出中断请求更为有效。

- 数据链路层 (Data Link Layer)

数据链路层的重要作用就是要保证 PCI Express 总线传输的数据包的正确性和可靠性。它会检查数据包封装是否完整正确，并将序列号和冗余校验代码 (CRC) 添加到数据中进行检测和纠错，并采用 ack/nack 握手协议进行检测纠错。

- 处理层 (Transaction Layer)

处理层的主要作用有两个，一是接受来自软件层送来的读、写请求，或者本身创建一个请求封装包传到数据链路层，这种数据包称为“处理层数据包” (Transaction Layer Packet, TLP 另一个作用是接受从链路层传来的响应数据包 (Data Link Layer Packet, DLL P)，并将其与相关的软件请求相关联，传送至软件层处理 (TLP 结构如图 20-7 所示)。

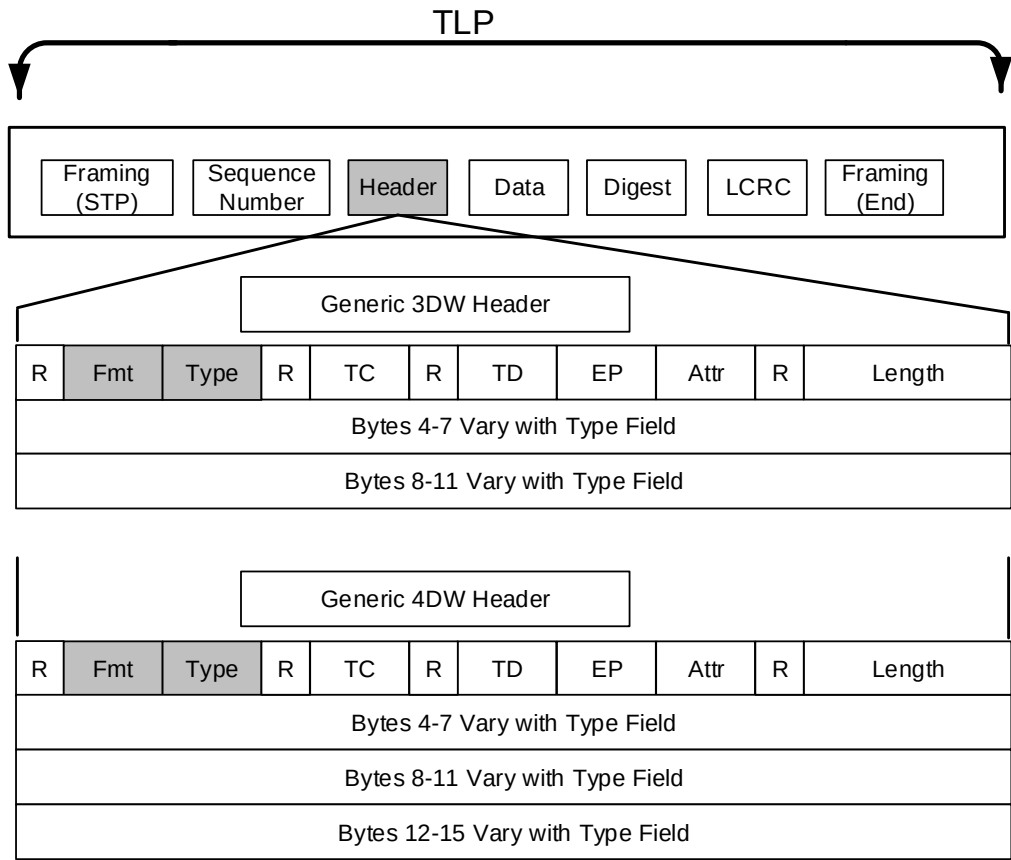
须知

详细的 TLP 报文结构说明会在 [20.3.5 PCIe 接口功能](#) 章节介绍。

PCIE 作为 EP 模式时，仅支持发送 INTa 中断。



图20-7 TLP 结构示意图



所有的数据包都包含有唯一的标识便于和相应的请求及响应相匹配（包括优先权的设置等），并确保对点通信的正确性。处理层可处理来自 4 个地址空间的数据。包括内存、I/O、配置以及信息地址空间。其中前 3 个地址空间来自 PCI 总线结构，最后的信息空间（MessageSpace）是 PCI Exp~m 规范新加的。上述地址空间类型如表 20-1 所示。

表20-1 PCI Express 地址空间类型

地址空间	处理方式	处理对象
内存	读、写。	来自于或者发送到内存中的数据。
I/O	读、写。	来自于或者发送到 I/O 的数据。
配置	读、写。	EP 相关设置。
消息	中断、供应商信息、电源管理、热插拔等。	事件信息，通用信息等。



- 软件层(Software Layer)

对于 PCI Express 而言，软件的兼容性是一个异常重要的因素。而软件的兼容性主要体现在两个方面：对硬件的检测和支持其运行。PCI 接口的外设在操作系统中能被系统自动发现，由系统分配相应的资源，如内存、I/O 空间和中断等，构建一个优化的系统运行环境。PCI Express 结构继续保持了 PCI 兼容的配置空间和 I/O 设备连接的规范。在软件方面和 PCI 的完全兼容，包括支持本地存储模式内存共享模式和在 PCI Express 平台中所有操作系统的引导程序等，所有的软件在 PCI Express 系统中执行都不会有所改变。

20.3.5 PCIe 接口功能

20.3.5.1 PCIe 存储器请求事务

PCI Express 存储器事务包括读请求、完成和写请求。表 20-2 描述了系统存储器映射及 3 DWords，4 DWords 存储器请求数据包格式。

3 DWords 和 4 DWords 存储器请求头各个字段的定义：表列出了在 4 DWords 存储器请求头中每个字段的位置和用途。

须知

3 DWords 和 4 DWords 头之间差别在于起始地址字段的位置和大小：

对于 3 DWords（32bit 寻址）：地址 bit[31:2]在 Byte8~11 中，Byte12~15 没有用。

对于 4 DWords（64bit 寻址）：地址 bit[31:2]在 Byte12~15 中，bit[63:32]在 Byte8~11 中二者其他的头字段相同。

表20-2 4 DWords 存储器请求头字段

字段名	头字节/比特	功能
Length[9:0]	Byte3 的 bit[7:0] Byte2 的 bit[1:0]	TLP 数据有效载荷大小，以 DWord 为单位。 最大为 10bit， $2^{10}=1024$ DWords(4KB)。编码： 00 0000 0001b=1DWord 00 0000 0010b=2DWord ...



字段名	头字节/比特	功能
		11 1111 1111b=1023 DWords 00 0000 0000b=1024 DWords
ATTR[1:0]	Byte2 的 bit[5:4]	bit[5]=灵活顺序。 设置为 1 时，对于此 TLP，允许 PCI-X 的灵活顺序。如果设置为 0，则使用 PCI 的严格顺序。 bit[4]=不侦测。 当设置为 1 时，表明请求者对于此 TLP 不存在缓存一致性问题。无需要系统硬件引起处理器缓存侦测一致性。 当设置为 0 时，要求进行 PCI 类型的缓冲侦测保护。
EP (Error Poisoned)	Byte2 的 bit[6]	如果设置为 1，尽管允许正常完成事务，但伴随次数据的数据应该被认为是无效的。
TD	Byte2 的 bit[7]	如果设置为 1，可选一个双字的 TLP 摘要 (Digest)字段包含在此 TLP 中。下面是一些规则： TD=1，但没有摘要字段的 TLP 作为 TLP 处理。 如果某设备在场 ECRC 校验并且 TD=1，则必须执行 ECRC 校验。 如果在最终目的地，某设备部不支持 ECRC 校验（可选），则该设备必须忽略此摘要字段。
TC[2:0]	Byte1 的 bit[6:4]	这 3 个比特用例编码应用于此 TLP 以及其相关的完成数据包（如果有的话）的流量类别。 000=流量类别 0（默认） ... 111=流量类别 7 TC0 是默认的类型，TC1~7 用来提供不同的服务。



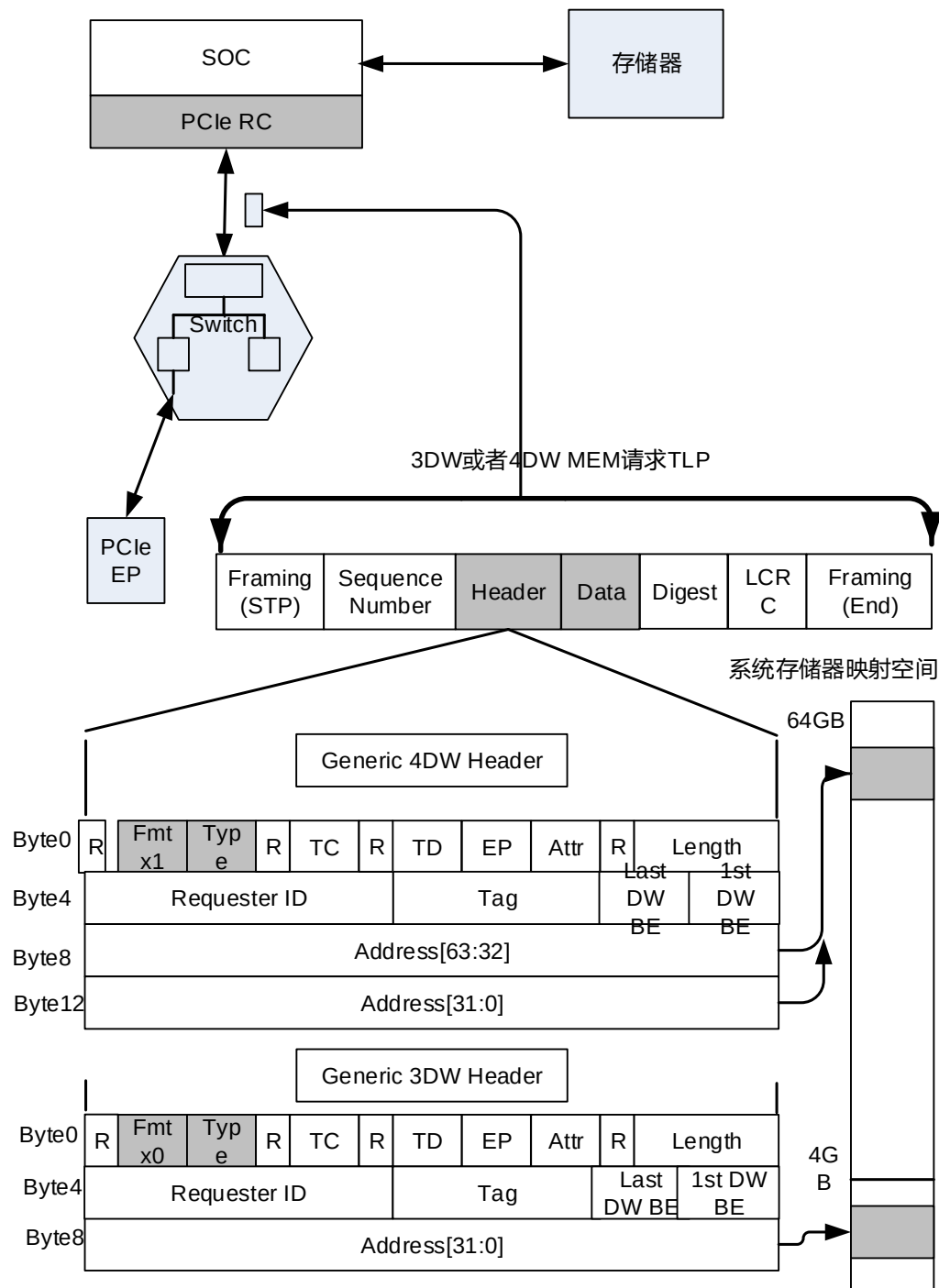
字段名	头字节/比特	功能
Type[4:0]	Byte0 的 bit[4:0]	TLP 数据包类别字段： 00000b=存储器读或者写 00001b=存储器读锁定
Fmt[1:0]	Byte0 的 bit[6:5]	数据包的格式： 00b=存储器读（3 DWords 带数据/无数据） 01b=存储器写（3 DWords 带数据） 10b=存储器读（4 DWords 带数据/无数据） 11b=存储器写（4 DWords 带数据）
1 st DWord BE[3:0]	Byte7 的 bit[3:0]	高电平比特对应的第一个 DWord 有效载荷中的字节。
Last DWord BE[3:0]	Byte7 的 bit[7:4]	高电平比特对应的最后一个 DWord 有效载荷中的字节。
Tag[7:0]	Byte6 的 bit[7:0]	用来标识请求者发出的每一个未完成的请求。 如果发送的非报告请求，则分配下一个连续的标志。 默认请求：只使用 bit[4:0]（在某一时刻，有 32 个为完成事务）。 如果 PCI Express 控制器寄存器中的扩展标记比特设置为 1，则可以使用全部 8 个比特（256 标记）。
Requester ID	Byte5 的 bit[7:0] Byte4 的 bit[7:0]	识别请求者，以便返回完成数据包等。 Byte4,7:0=总线号 Byte5,7:3=设备号 Byte5,2:0=功能号
Address[31: 2]	Byte15 的 bit[7:2] Byte14 的 bit[7:0] Byte13 的 bit[7:0] Byte12 的 bit[7:0]	存储器传送的 64bit 起始地址的低端 32bit。注意 32bit 地址的最低端 2bit（00b），强制起始地址为 DWord 对齐。



字段名	头字节/比特	功能
Address[63:32]	Byte11 的 bit[7:2] Byte10 的 bit[7:0] Byte9 的 bit[7:0] Byte8 的 bit[7:0]	存储器传送的 64bit 高端 32 地址。



图20-8 PCIe 存储器读写请求结构示意图



PCI Express 存储器请求事务的特点：

- 存储器传送决不允许跨越 4KB 边界。
- 所有的存储器映射写是要被报告的，因此可以产生更高的性能。（即是写后不管）



- 可以使用 32bit 或者 64bit 寻址。3 DWords 头格式支持 32bit 地址，而 4 DWords 头支持 64bit 地址。
- 可以利用突发传送的最大能力，其传送长度为 0~1024 DWords (0~4KB)。

20.3.5.2 PCIe IO 请求事务

为了兼容传统设备和可能依赖驻留在 IO 映射而不是存储器映射空间的兼容设备中的软件，PCIe 接口保留支持 IO 事务，并同时支持 32bit 地址 IO 范围，也支持低 16bit (64KB) 的 IO 范围。

须知

PCIe 接口 EP 模式时，可以配置传统的 PCI Express 设备，支持基址配置寄存器请求 IO 地址分配。

IO 请求头字段的定义。图 20-3 给出了一个 IO 请求头的每个字段的位置和用途。

表20-3 IO 请求头字段

字段名	头字节/比特	功能
Length[9:0]	Byte3 的 bit[7:0] 字节 2 的 bit[1:0]	TLP 数据有效载荷大小，以 DWord 为单位。 该字段总是为 1。
ATTR[1:0]	Byte2 的 bit[5:4]	bit[5]=灵活顺序。 在配置请求中总是为 0。 bit[4]=不侦测。 在 IO 请求中总是为 0。
EP (Error Poisoned)	Byte2 的 bit[6]	如果设置为 1，尽管允许正常完成事务，但伴随次数据的数据应该被认为是无效的。



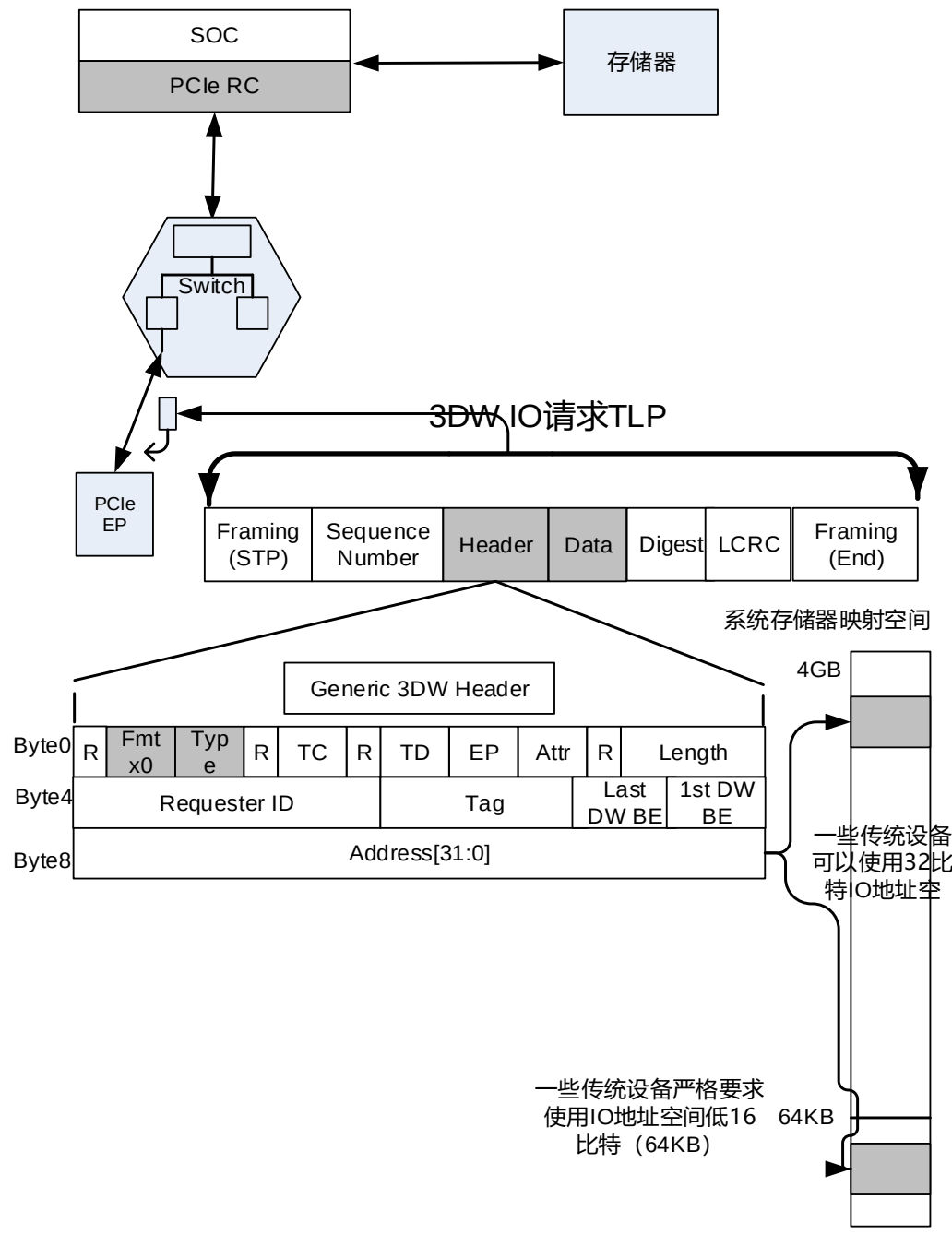
字段名	头字节/比特	功能
TD	Byte2 的 bit[7]	<p>如果设置为 1, 可选一个双字的 TLP 摘要 (Digest) 字段包含在此 TLP 中。下面是一些规则:</p> <p>TD=1, 但没有摘要字段的 TLP 作为 TLP 处理。</p> <p>如果某设备在场 ECRC 校验并且 TD=1, 则必须执行 ECRC 校验。</p> <p>如果在最终目的地, 某设备部不支持 ECRC 校验 (可选), 则该设备必须忽略此摘要字段。</p>
TC[2:0]	Byte1 的 bit[6:4]	<p>这 3 个比特用例编码应用于此 TLP 以及其相关的完成数据包 (如果有的话) 的流量类别。</p> <p>在 IO 请求中 TC=0。</p>
Type[4:0]	Byte0 的 bit[4:0]	TLP 数据包的类别字段。对于 IO 请求, 总是为 00001b。
Fmt[1:0]	Byte0 的 bit[6:5]	<p>数据包的格式:</p> <p>00b=IO 读 (3 DWords 无数据)</p> <p>10b=IO 写 (3 DWords 带数据)</p>
1 st DWord BE[3:0]	Byte7 的 bit[3:0]	高电平比特对应的第一个 DWord 有效载荷中的字节。
Last DWord BE[3:0]	Byte7 的 bit[7:4]	高电平比特对应的最后一个 DWord 有效载荷中的字节。



字段名	头字节/比特	功能
Tag[7:0]	Byte6 的 bit[7:0]	用来标识请求者发出的每一个未完成的请求。如果发送的非报告请求，则分配下一个连续的标志。 默认请求：只使用 bit[4:0]（在某一时刻，有 32 个为完成事务）。 如果 PCI Express 控制器寄存器中的扩展标记比特设置为 1，则可以使用全部 8 个比特（256 标记）。
Requester ID[15:0]	Byte5 的 bit[7:0] Byte4 的 bit[7:0]	识别请求者，以便返回完成数据包等。 字节 4,7:0=总线号 字节 5,7:3=设备号 字节 5,2:0=功能号
Address[31:2]	Byte11 的 bit[7:2] Byte10 的 bit[7:0] Byte9 的 bit[7:0] Byte8 的 bit[7:0]	IO 传送的 32bit 起始地址高端 30 比特。注意 32bit 地址的最低端 2bit（00b），强制起始地址为 DWord 对齐。



图20-9 PCIe IO 读写请求结构示意图



20.3.5.3 PCIe 配置请求事务

为了保持与 PCI 兼容，PCI Express 同时支持类型 0 和类型 1 配置周期。类型 1 周期向下游传播，直到它到达目标所在的总线的桥接口。在目的链路上，桥将配置事务从类型 1 转换成类型 0。图 20-10 举例说明类型 1 配置周期向下游传播的过程。在目的链路上，它将被转换成类型 0，并由端点设备声明。所有端点设备都在每个类型 0 配



置写周期中捕获由上游设备提供的总线号和设备号信息，该信息包含在配置请求头的 8~9Byte 中。

说明

配置请求总是使用 3 DWords 头格式，而且根据 ID 字段的内容来路由。

配置请求头各个字段定义，如表 20-4 所示。

表20-4 配置请求头字段

字段名	头字节/比特	功能
Length[9:0]	Byte3 的 bit[7:0] Byte2 的 bit[1:0]	TLP 数据有效载荷大小，以 DWord 为单位。该字段总是为 1。
ATTR[1:0]	Byte2 的 bit[5:4]	bit[5]=灵活顺序。 在配置请求中总是为 0。 bit[4]=不侦测。 在配置请求中总是为 0。
EP (Error Poisoned)	Byte2 的 bit[6]	如果设置为 1，尽管允许正常完成事务，但伴随次数据的数据应该被认为是无效的。
TD	Byte2 的 bit[7]	如果设置为 1，可选一个双字的 TLP 摘要(Digest)字段包含在此 TLP 中。下面是一些规则： TD=1，但没有摘要字段的 TLP 作为 TLP 处理。 如果某设备在场 ECRC 校验并且 TD=1，则必须执行 ECRC 校验。 如果在最终目的地，某设备部不支持 ECRC 校验（可选），则该设备必须忽略此摘要字段。
TC[2:0]	Byte1 的 bit[6:4]	这 3 个比特用例编码应用于此 TLP 以及其相关的完成数据包（如果有的话）的流量类别。



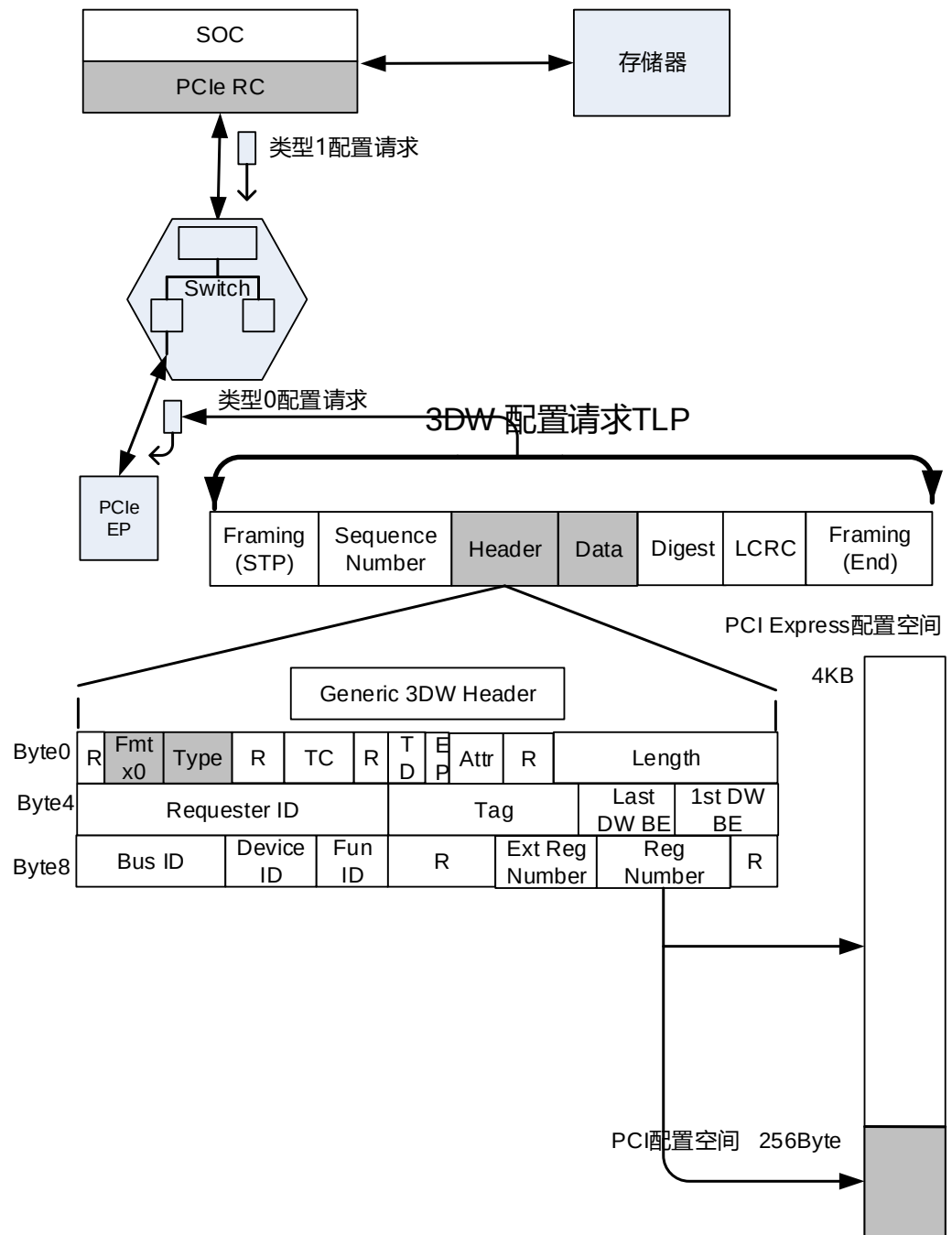
字段名	头字节/比特	功能
		在配置请求中 TC=0。
Type[4:0]	Byte0 的 bit[4:0]	TLP 数据包的类别字段： 00100b=配置 0 请求； 00101b=配置 1 请求。
Fmt[1:0]	Byte0 的 bit[6:5]	数据包的格式： 00b=配置读（3 DWords 无数据）； 10b=配置写（3 DWords 带数据）。
1 st DWord BE[3:0]	Byte7 的 bit[3:0]	高电平比特对应的第一个 DWord 有效载荷中的字节。
Last DWord BE[3:0]	Byte7 的 bit[7:4]	高电平比特对应的最后一个 DWord 有效载荷中的字节。
Tag[7:0]	Byte6 的 bit[7:0]	用来标识请求者发出的每一个未完成的请求。如果发送的非报告请求，则分配下一个连续的标志。 默认请求：只使用 bit[4:0]（在某一时刻，有 32 个为完成事务）。 如果 PCI Express 控制器寄存器中的扩展标记比特设置为 1，则可以使用全部 8 个比特（256 标记）。
Requester ID[15:0]	Byte5 的 bit[7:0] Byte4 的 bit[7:0]	识别请求者，以便返回完成数据包等。 字节 4,7:0=总线号 字节 5,7:3=设备号 字节 5,2:0=功能号
Register Number[5:0]	Byte11 的 bit[7:2]	配置空间偏移量（DWord）的低 6bit。 寄存器好与扩展寄存器号一起用来表示 1024 DWords（4096 字节）PCI Express 配置空间的 10bit 偏移量。
Ext Register Number[3:0]	Byte10 的 bit[3:0]	配置空间偏移量（DWord）的高 4bit。 扩展寄存器好与扩展寄存器号一起用来



字段名	头字节/比特	功能
		表示 1024 DWords (4096Byte) PCI Express 配置空间的 10bit 偏移量。
Complete ID[15:0]	Byte9 的 bit[7:0] Byte8 的 bit[7:0]	标识此配置周期要访问的完成者。设备可以在配置类型 0 写时，捕获这字段中的总线和设备。 Byte4,7:0=总线号 Byte5,7:3=设备号 Byte5,2:0=功能号



图20-10 PCIe 配置读写请求示意图



20.3.5.4 PCIe 完成请求事务

完成在以下每个非报告请求后返回：

- 存储器读请求可能返回带数据的完成（CplD）。
- IO 读请求可能导致带数据或者不带数据的完成（CplD）。



- IO 写请求可能导致没有数据的完成 (Cpl)。
- 配置读请求可能导致带数据的完成 (CplD)。
- 配置写请求可能导致没有数据的完成 (CplD)。
- 配置读请求。

完成请求头中的许多字段与相应的请求必须要有相同的值，包括流量类别，属性比特和用于路由完成数据返回原请求者的原请求者 ID。图 20-11 描述了在非报告请求之后返回的一个完成，以及 3 DWords 的完成头格式。

表 20-5 列出了完成头各字段的位置和用途。

表20-5 完成请求头字段

字段名	头字节/比特	功能
Length[9:0]	Byte3 的 bit[7:0] Byte2 的 bit[1:0]	TLP 数据有效载荷大小，以 DWord 为单位。 该字段反映与完成数据包相关的数据有效载荷的大小。
ATTR[1:0]	Byte2 的 bit[5:4]	属性 1：灵活顺序。 属性 0：不侦测比特 在完成数据包中，这些比特设置成和请求中的状态一样。
EP (Error Poisoned)	Byte2 的 bit[6]	如果设置为 1，表示数据有效载荷中毒了。
TD	Byte2 的 bit[7]	如果设置为 1，可选一个双字的 TLP 摘要 (Digest) 字段包含在此 TLP 中。下面是一些规则： TD=1，但没有摘要字段的 TLP 作为 TLP 处理。 如果某设备在场 ECRC 校验并且 TD=1，则必须执行 ECRC 校验。 如果在最终目的地，某设备部不支持 ECRC 校验（可选），则该设备必须忽略此摘要字段。
TC[2:0]	Byte1 的 bit[6:4]	表示包传送类别，对于完成数据包，TC 设置成和请求相同的值。



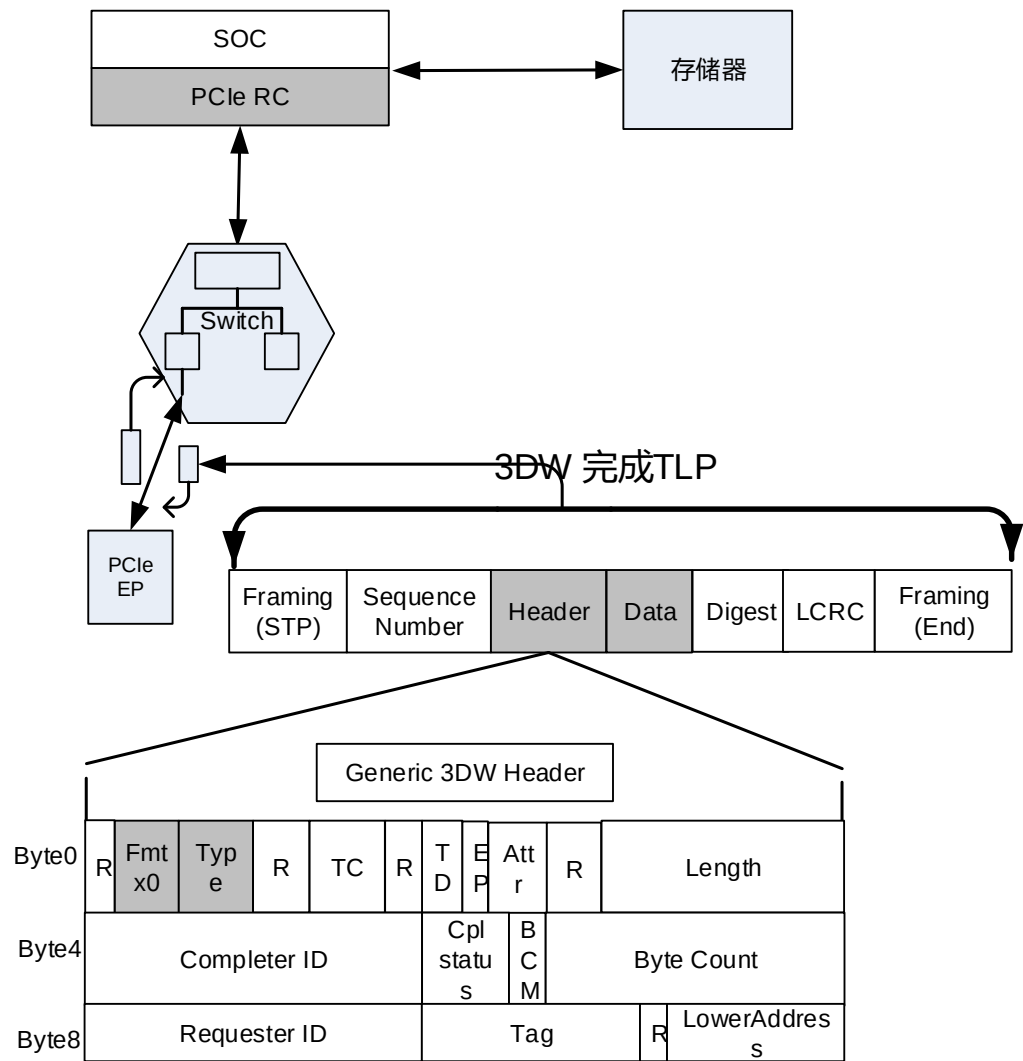
字段名	头字节/比特	功能
Type[4:0]	Byte0 的 bit[4:0]	对于完成数据包, TLP 数据包的类别字段总是 =01010b
Fmt[1:0]	Byte0 的 bit[6:5]	数据包的格式: 00b=无数据的完成数据包 (Cpl) 10b=带数据的完成数据包 (CplD)
Byte Counter	Byte7 的 bit[7:0] Byte6 的 bit[3:0]	读请求结束前的剩余字节数。通常, 可以由原请求的长度字段获得。
BCM	Byte6 的 bit[4]	修改的字节计数, 只能有 PCI-X 完成者设置为 1, 所以在应用中, PCIe 接口默认设置为 0。
CS [2:0]	Byte6 的 bit[7:5]	完成状况代码。 000b: (SC)成功完成的代码, 表示原请求在目标处已正确完成。 001b: (UR)不支持的请求, 表示原请求在目标处失败, 因为它以不支持的地址为目标、携带不支持的地址或请求等, 为不可修复的错误。 010b: (CRS) 配置请求重试状况, 表示目标暂时掉线, 应该尝试重试 (例如: 复位后的初始化延时等)。 100b: 完成中止代码, 表示完成者由于某一错误而掉线, 为不可修复的错误。 其他: 保留。
Complete ID[15:0]	Byte5 的 bit[7:0] Byte4 的 bit[7:0]	标识完成者。尽管路由完成数据包时不需要这个信息, 但在调试总线流量时, 这一信息非常有用。 Byte4,7:0=总线号 Byte5,7:3=设备号 Byte5,2:0=功能号
Low Address[6:0]	Byte11 的 bit[6:0]	一次读操作返回数据的第一允许字节的地址低端 7bit。可以由请求数据包中的长度和字节允许来计算。常用来确定下一合法读完成的边



字段名	头字节/比特	功能
		界。
Tag[7:0]	Byte6 的 bit[7:0]	设置这些比特以反映请求标记。请求者使用这一字段将入站的完成数据包与未完成的请求关联起来。
Requester ID[15:0]	Byte9 的 bit[7:0] Byte8 的 bit[7:0]	从请求中复制到此字段，用来路由完成数据包返回原请求者。 Byte4,7:0=总线号 Byte5,7:3=设备号 Byte5,2:0=功能号



图20-11 PCIe 完成请求结构示意图



20.3.5.5 PCIe 消息请求事务

消息请求可以代替早期总线协议使用的许多中断，错误和电源管理边带信号。所有的消息请求都使用 4 DWords 头格式，并且处理消息请求的过程与处理存储器写事务基本相同。消息可以使用地址路由，ID 路由或者隐式路由方法来路由。图 20-12 描述了消息请求头的格式。

消息请求头各字段的定义，如表 20-6 所示。



表20-6 4 DWords 消息请求头字段

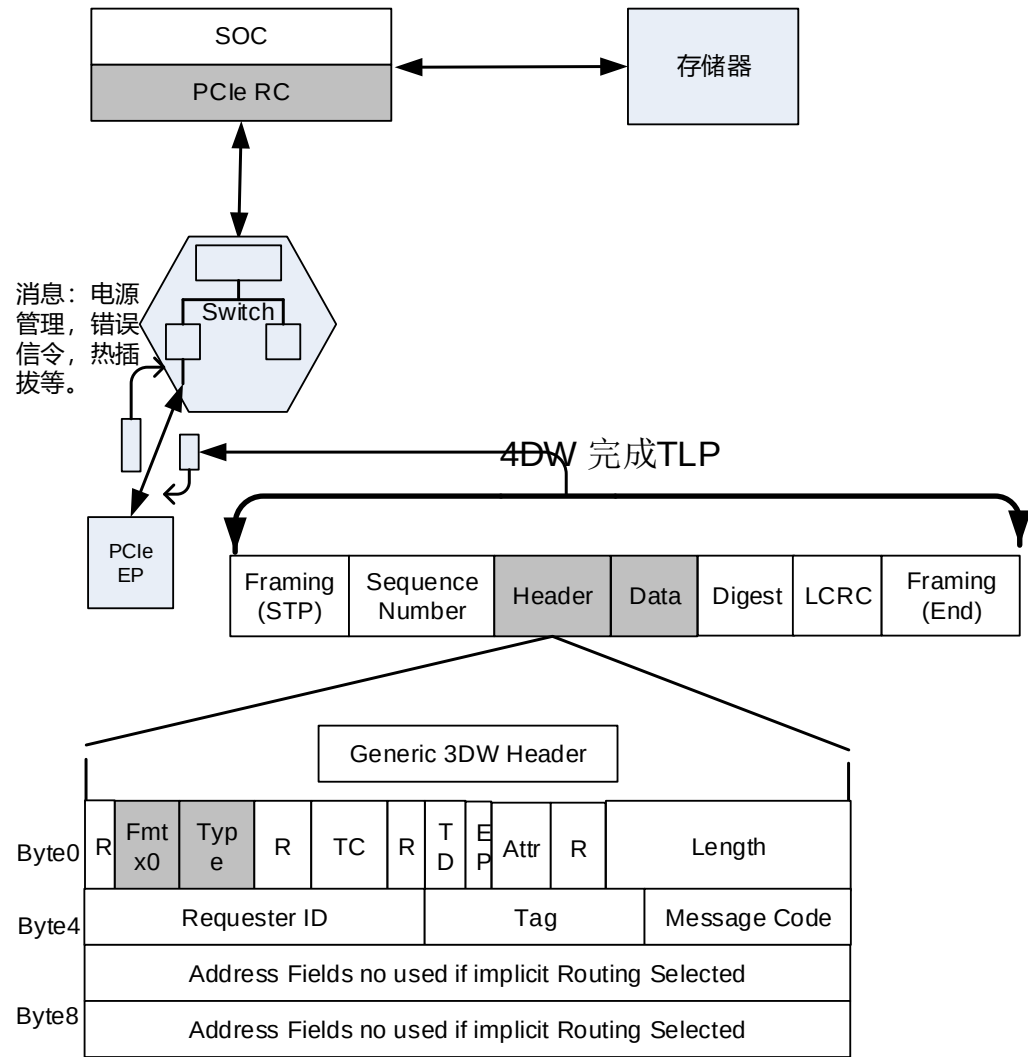
字段名	头字节/比特	功能
Length[9:0]	Byte3 的 bit[7:0] Byte2 的 bit[1:0]	TLP 数据有效载荷大小，以 DWord 为单位。 该字段总是为 1。
ATTR[1:0]	Byte2 的 bit[5:4]	属性 1：灵活顺序。 属性 0：不侦测比特。 在消息数据包中，这些比特总是设置 0。
EP (Error Poisoned)	Byte2 的 bit[6]	如果设置为 1，代表数据中毒了。
TD	Byte2 的 bit[7]	如果设置为 1，可选一个双字的 TLP 摘要 (Digest) 字段包含在此 TLP 中。下面是一些规则： TD=1，但没有摘要字段的 TLP 作为 TLP 处理。 如果某设备在场 ECRC 校验并且 TD=1，则必须执行 ECRC 校验。 如果在最终目的地，某设备部不支持 ECRC 校验（可选），则该设备必须忽略此摘要字段。
TC[2:0]	Byte1 的 bit[6:4]	这 3 个比特用例编码应用于此 TLP 以及其相关的完成数据包（如果有的话）的流量类别。 在消息请求中 TC=0。
Type[4:0]	Byte0 的 bit[4:0]	TLP 数据包的类别字段： bit[4:3]10=消息请求 bit[2:0]000b=路由至根联合体 001b=根据地址路由 010b=根据 ID 路由 011b=根联合体广播 Msg 100b=本地，终止于接收器



字段名	头字节/比特	功能
		101b=收集/路由至根联合体 其他保留。
Fmt[1:0]	Byte0 的 bit[6:5]	数据包的格式： 01b=4 DWords 无数据消息请求 11b=4 DWords 带数据消息请求
Msg Tag[7:0]	Byte6 的 bit[7:0]	该字段含有表明要发送的消息的类型的编码。 0000 0000b=解除锁定消息 0001 xxxxb=电源管理消息 0010 0xxxb=Intx 消息 0011 00xxb=出错消息 0100 0000b=热插拔消息 0101 0000b=插槽电源消息 0111 111xb=厂商类型 0 消息 0111 1111b=厂商类型 0 消息
Requester ID	Byte5 的 bit[7:0] Byte4 的 bit[7:0]	识别请求者，以便返回完成数据包等。 Byte4,7:0=总线号 Byte5,7:3=设备号 Byte5,2:0=功能号
Address[31: 2]	Byte15 的 bit[7:2] Byte14 的 bit[7:0] Byte13 的 bit[7:0] Byte12 的 bit[7:0]	如果消息选择地址路由方法，则此字段有效。传送的 64bit 起始地址的低端 32bit。注意 32bit 地址的最低端 2bit (00b)，强制起始地址为 DWord 对齐。
Address[63: 32]	Byte11 的 bit[7:2] Byte10 的 bit[7:0] Byte9 的 bit[7:0] Byte8 的 bit[7:0]	如果消息选择地址路由方法，则此字段有效。消息传送的 64bit 高端 32 地址。



图20-12 PCIe 消息请求结构示意图

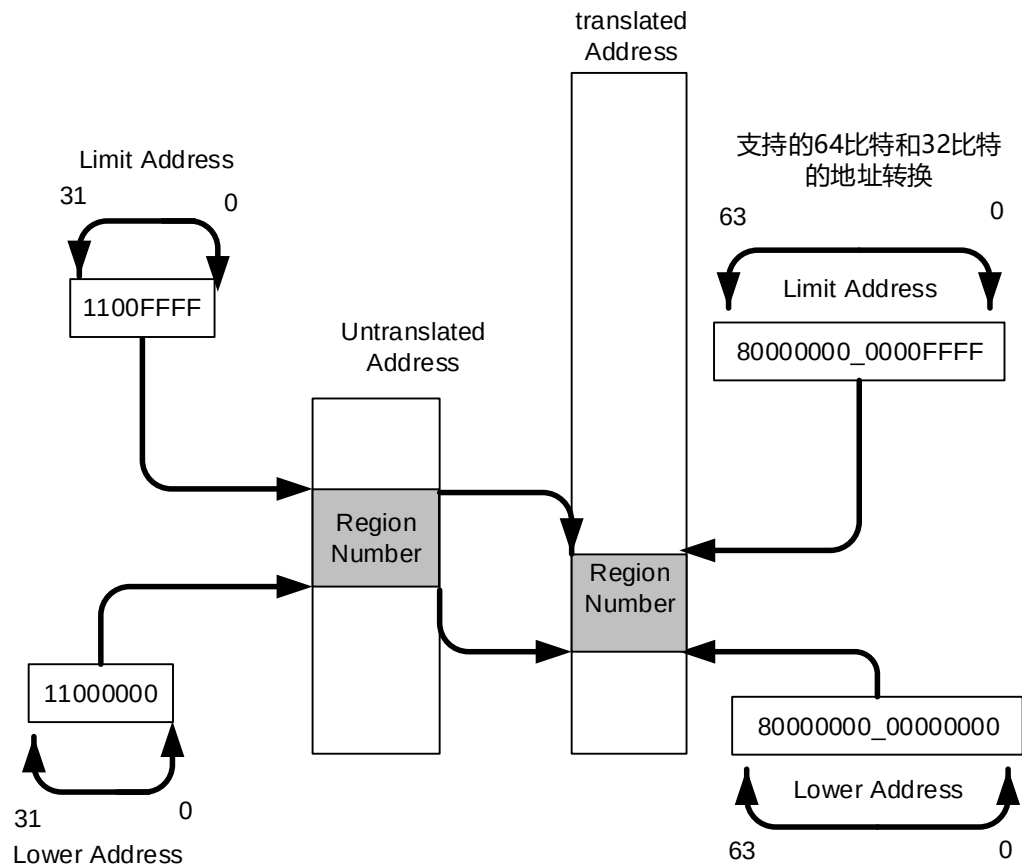


20.3.5.6 iATU 地址转换功能

PCIe 接口的 iATU 功能主要实现与外设进行读写访问时，能够自由进行地址空间的转换。iATU 支持 32bits 和 64bits 的地址空间的转换。其结构如图 20-13 所示。



图20-13 iATU 地址转换结构示意图



须知

iATU 寄存器访问两种途径：

1. 本地 CPU 读写，通过 DBI 总线访问，访问之前需要配置 PCIE_MISC.PCIE_CTRL0.bit[21]和 PCIE_MISC.PCIE_CTRL1.bit[21]为 1 选通 DBI 访问，配置完成后要将该比特重新配为 0；
2. 配置空间访问（RC 访问 EP），其基地址为配置空间地址 0x0。

iATU 配置转换的地址有 64K 对齐的要求（地址低 16 位为 0）。

表20-7 iATU Register Map(offset address 0x700)

Byte Offset	描述
+ 0x200	iATU Viewport 寄存器
+ 0x204	iATU Region Control1 寄存器



Byte Offset	描述
+ 0x208	iATU Region Control2 寄存器
+ 0x20c	iATU Region 基地址低寄存器
+ 0x210	iATU Region 基地址高寄存器
+ 0x214	iATU Region 基地址限制寄存器
+ 0x218	iATU Region 目标低地址寄存器
+ 0x21c	iATU Region 目标高地址寄存器

须知

当在 PCIe 模块初始化完成后，需要配置 iATU 的相关寄存器才能进一步进行发包其配置流程步骤分两种，一种是 Inbound，另外一种是 Outbound。对应 iATU 的两种模式，既可以单独使用其中一种，也可以两种一起使用。

20.4 工作方式

20.4.1 管脚复用配置

不涉及。

20.4.2 初始化

20.4.2.1 RC 模式下的初始化操作

当 PCIe1 需要工作在 RC 模式下，在从上电到链路建立之前，需要配置相关寄存器。

须知

由于 PCIe0 不支持 RC 模式，所以以下相关操作步骤都是针对 PCIe1 而言。

详细操作步骤如下：



- 步骤 1 把 PCIe 的模式配置为 RC (PCIE_MISC.PCIE_CTRL0 的 bit[31:28]=0x4) 。
- 步骤 2 初始化 SerDes 为 PCIe 模式。
- 步骤 3 使能 PCIe 的时钟 (HOST_SUBSYS_REG. HOST_CLK_ENABLE 偏移 0 寄存器 bit1=1) 。
- 步骤 4 撤销 PCIe LTSSM (PCIE_MISC.PCIE_CTRL7 的 bit[11]=0) 。
- 步骤 5 撤销 PCIe Core 的复位 (HOST_SUBSYS_REG. HOST_RST_DISABLE 偏移 0x10 寄存器 bit1=1) 。
- 步骤 6 使能 PCIe LTSSM (PCIE_MISC.PCIE_CTRL7 的 bit[11]=1) 。
- 步骤 7 确定 PCIe 已经建立链接。读取 PCIE_MISC.PCIE_STATS0 状态寄存器的 bit[15]和 bit[5], 如果值均为 1b1, 表明链接已经建立。
- 步骤 8 通过 DBI 设置 PCI 兼容配置头的命令寄存器、类代码。向 PCI 兼容配置空间的第 2 个 DWord 的 bit[2:0]写入 0x7, 向第 3 个 DWord 的 bit[31:8]写入 0x60000。

说明

设置 PCI 配置空间命令寄存器为 0x7, 则打开了 IO 空间使能, 存储器空间使能, 总线主控制器使能; 设置 PCI 配置空间类代码寄存器, 则设定了类代码为 0x60。详细寄存器描述请参考 PCIe 协议相关文档。

获取路径: <https://pcisig.com/>

- 步骤 9 执行标准的 PCI 枚举过程, 软件驱动根据下游芯片申请的资源进行资源分配, PCIe 实际上进行配置读写的过程。

须知

在步骤 9, 建议移植 OS 支持的标准 PCI 驱动程序执行 PCI 枚举过程。如果确定移植 OS 的标准 PCI 驱动程序存在风险, 根据实际典型应用, 软件知道 EP 只有一个, 而且也是 Hi3093, 可以跳过这一步骤。

设置事务类型为配置类型。

- 步骤 10 设置 EP 的总线号为 1、功能号为 0。
- 步骤 11 开始读取对端 EP 的厂商设备号。读取对端 EP 的 PCI 配置空间的第 1 个寄存器, 确保厂商设备号没有问题。



步骤 12 存储器空间配置，通过读取对端 EP 的基址 0 ~ 基址 5 寄存器（PCI 兼容配置头的第 4 个 DW 开始）。

说明

按照标准的 PCI 配置过程，应该先向该地址写入 0xFFFFFFFF，再读取该地址的值，根据其返回值从基址的最低有效比特（bit7）向上扫描，通过检查设置为 1 的最低有效比特，软件就可以确定所需存储器空间。例如：写入 0xFFFFFFFF，其返回值是 0xFFFF0000，软件可以知道，该存储器空间大小 1MB，软件可以根据实际情况，分配 1MB 地址空间给 EP。

步骤 13 IO 空间配置，读取对端 EP 的基址 0 ~ 基址 5 寄存器，如果其中有寄存器的 bit0 为 1，表示该基址寄存器申请的是 IO 空间。

步骤 14 关闭保护（配置 PCIE_MISC.PCIE_CTRL8 的 bit[17]=1）。

---结束

20.4.2.2 EP 模式下的初始化操作

当 PCIe1 需要工作在 EP 模式下，在从上电到链路建立之前，需要配置相关寄存器。

以下是 EP 模式操作步骤详细描述：

步骤 1 把 PCIe 的模式配置为 EP（PCIE_MISC.PCIE_CTRL0 的 bit[31:28]=0x0）。

须知

下面各个操作步骤同 RC 模式操作一致，相关寄存器描述请参考 RC 模式初始化相关表格描述。

步骤 2 配置 SerDes 的工作模式为 PCIe。

步骤 3 设置存储器事务对应的地址空间。根据系统需求，写入存储器事务的起始地址，结束地址。

步骤 4 复位 SerDes。

步骤 5 撤销 SerDes 的复位。

步骤 6 使能 PCIe 的时钟。



须知

对于 PCIe0，由于应用在 GPU 场景，所以默认不需要以下步骤，相关操作完全由主控端 CPU 来完成。

步骤 7 复位 PCIe Core。

步骤 8 撤销 PCIe Core 的复位。

步骤 9 使能 PCIe LTSSM。

----结束

20.4.3 配置读/写操作（类型 0）

当 PCIe 配置为 RC 模式时，PCIe 可以访问对接的 EP 的寄存器空间。应用场景是一个 RC，一个 EP 设备。配置读写的步骤如下：

步骤 1 进行 RC 模式的初始化操作，请参见 [20.4.2.1 RC 模式下的初始化操作](#) 章节。

步骤 2 设置事务类型为配置类型 0。

步骤 3 设置 EP 的总线号为 1、功能号为 0（实际按照 switch 实际用户手册进行操作）。

步骤 4 开始读取对端 EP 的厂商设备号。

步骤 5 如果 [步骤 5](#) 成功，则进行其他寄存器的读写操作。

----结束

20.4.4 配置读/写操作（类型 1）

当 PCIe 配置为 RC 模式时，PCIe 可以访问对接的 EP 的寄存器空间。应用场景是一个 RC，通过 Switch，访问多个 EP 设备。配置读写的步骤如下：

步骤 1 进行 RC 模式的初始化操作，请参见 [20.4.2.1 RC 模式下的初始化操作](#) 章节。

步骤 2 设置事务类型为配置类型 1。

步骤 3 设置 Switch 的总线号为 1、功能号为 0。

步骤 4 开始读取对端 Switch 的厂商设备号。

步骤 5 如果 [步骤 5](#) 成功，则进行其他寄存器的读写操作，枚举 Switch。

----结束



20.4.5 RC 模式下的存储器读/写操作（直接访问方式）

当 PCIe 配置为 RC 模式时，PCIe 可以访问对接的 EP 的存储器空间。步骤如下：

- 步骤 1 确保 PCIe 作为 RC 可以正确地读写对端 EP 的寄存器空间，请参见 [20.4.2.1 RC 模式下的初始化操作](#) 章节。
- 步骤 2 配置事务类型为 MEM 事务。
- 步骤 3 通过读写 MEM 事务的起始地址和结束地址就可以读写对端对应的存储器空间。

----结束

20.4.6 EP 模式下的存储器读/写操作（直接访问方式）

当 PCIe 配置为 EP 模式时，PCIe 可以访问对接的 RC 的存储器空间。步骤如下：

- 步骤 1 配置事务类型为 MEM 事务。
- 步骤 2 通过读写 MEM 事务的起始地址和结束地址就可以读写对端对应的存储器空间。

----结束

20.4.7 RC 模式下的 IO 读/写操作

当 PCIe 配置为 RC 模式时，如果对端 EP 支持 IO 操作，则可以发起 IO 读/写操作，步骤如下：

- 步骤 1 确保 PCIe 作为 RC 可以正确的读写对端 EP 的配置寄存器空间，请参见 [20.4.2.1 RC 模式下的初始化操作](#) 章节。
- 步骤 2 配置事务类型为 IO 事务。
- 步骤 3 通过读写 IO 事务的起始地址和结束地址就可以读写对端对应的 IO 空间。

----结束

20.4.8 iATU 功能 Inbound 模式

须知

iATU 寄存器访问两种途径：



- 本地 CPU 读写，通过 DBI 总线访问，访问之前需要配置 PCIE_MISC.PCIE_CTRL0.bit[21]和 PCIE_MISC.PCIE_CTRL1.bit[21]为 1 选通 DBI 访问，配置完成后要将该比特重新配为 0；
- 配置空间访问（RC 访问 EP），其基地址为配置空间地址 0x0。

iATU 配置转换的地址有 64K 对齐的要求（地址低 16 位为 0）。

表20-8 iATU Register Map(offset address 0x700)

Byte Offset	描述
+ 0x200	iATU Viewport 寄存器。
+ 0x204	iATU Region Control1 寄存器。
+ 0x208	iATU Region Control2 寄存器。
+ 0x20c	iATU Region 基地址低寄存器。
+ 0x210	iATU Region 基地址高寄存器。
+ 0x214	iATU Region 基地址限制寄存器。
+ 0x218	iATU Region 目标低地址寄存器。
+ 0x21c	iATU Region 目标高地址寄存器。

设置 Inbound Region2，将配置空间 BAR4 映射到 0x800000000_20000000 地址空间。操作步骤如下：

步骤 1 设置 ViewPort 寄存器

写 0x80000002 到地址（offset 0x700 + 0x200），则设置了 Inbound Region2 作为当前操作区域。

表20-9 Viewport (offset address 0x700+0x200)

位	名称	描述
3:0	Region Index	Region 索引号。 最大不能超过 7。
30: 4	-	保留。该域为只读域。



31	Region Direction	Region Direction。 0: Outbound; 1: Inbound。
----	------------------	--

步骤 2 设置目标地址寄存器

写 0x20000000 到地址 (offset 0x700 + 0x218)，则设置了目标低地址寄存器。

写 0x80000000 到地址 (offset 0x700 + 0x21C)，则设置了目标高地址寄存器。

步骤 3 通过 Region Control1 寄存器来配置相关设置

写 0x00000000 到地址 (offset 0x700 + 0x204)，则定义了该区域的数据是 MEM 空间地址数据，打开 BAR 模式匹配。

表20-10 Region Control1(offset address 0x700+0x204)

位	名称	描述
4:0	TYPE	PCI Express 类型代码 00000: MEM Request。

说明

TYPE 的具体定义请参考 PCI Express Base Specification 相关文档。

步骤 4 最后是打开使能，BAR 匹配设置。

写 0xC0000400 到地址 (offset 0x700 + 0x208)，则打开了 iATU 使能寄存器，匹配 BAR4 到目标地址寄存器映射的地址空间。

表20-11 iATU Region Control2(offset address 0x700+0x208)

位	名称	描述
31	Region Enable	iATU function 使能。 0: 禁止; 1: 使能。



位	名称	描述
30	Matche Mode	模式匹配。 Inbound MEM/IO： 0：地址模式； 1：BAR 模式。 注意：使用 BAR 匹配模式时，转换地址的地址对齐要求取决于：ATU 的地址对齐要求（64K）和 BAR 的地址对齐要求（BAR MASK 配置决定）两者较大者。 Inbound CFG0： 0：ID 路由模式； 1：接受模式。 Inbound MSG/MSGD： 0：地址模式； 1：Vendor ID。
10:8	Bar Machted	Bar 匹配配置。 000：BAR0； 001：BAR1； 010：BAR2； 011：BAR3； 100：BAR4； 101：BAR5； 111：保留。

----结束

设置 Inbound Region0，将 TLP 里面地址范围 0x11000000-0x11FFFFFF 映射到 0x000000000_80000000 至 0x000000000_80FFFFFF 地址空间。操作步骤如下：

步骤 1 设置 ViewPort 寄存器。



写 0x80000000 到地址 (offset 0x700 + 0x200), 则设置了 Inbound Region0 作为当前操作区域。

步骤 2 设置窗口寄存器的 Base Address 和 Limit Address 寄存器。

- 写 0x11000000 到地址 (offset 0x700 + 0x20c), 则设置了基地址低地址;
- 写 0x00000000 到地址 (offset 0x700 + 0x210), 则设置了基地址高地址;
- 写 0x11efffff 到地址 (offset 0x700 + 0x214), 则设置了地址限制寄存器。

步骤 3 设置目标地址寄存器。

- 写 0x80000000 到地址 (offset 0x700 + 0x218), 则设置了目标低地址寄存器;
- 写 0x00000000 到地址 (offset 0x700 + 0x21c), 则设置了目标高地址寄存器。

步骤 4 通过 Region Control1 寄存器来配置相关设置。

写 0x00000000 到地址 (offset 0x700 + 0x204), 则定义了该区域的数据是 MEM 空间且是地址匹配模式。

步骤 5 写 0x80000000 到地址 (offset 0x700 + 0x208), 则打开了 iATU 使能寄存器。

须知

在 iATU 的 inbound 功能一般情况下使用在 EP 模式, 兼容不同的系统的地址空间的转换。

----结束

20.4.9 iATU 功能 Outbound 模式

须知

对于通一块 Region Index, Outbound 和 Inbound 模式可以同时使用。

设置 outbound 的 Region0, 例如将 TLP 里面地址范围 0x11000000-0x117FFFFFF 映射到 0x000000000_90000000 至 0x000000000_907FFFFFF 地址空间。

步骤 1 设置 ViewPort 寄存器。



写 0x00000000 到地址 (offset 0x700 + 0x200)，则设置了 outbound Region0 作为当前操作区域。

步骤 2 设置窗口寄存器的 Base Address 和 Limit Address 寄存器。

- 写 0x11000000 到地址 (offset 0x700 + 0x20c)，则设置了基地址低地址；
- 写 0x00000000 到地址 (offset 0x700 + 0x210)，则设置了基地址高地址；
- 写 0x117fffff 到地址 (offset 0x700 + 0x214)，则设置了地址限制寄存器。

步骤 3 设置目标地址寄存器。

- 写 0x90000000 到地址 (offset 0x700 + 0x218)，则设置了目标低地址寄存器；
- 写 0x00000000 到地址 (offset 0x700 + 0x21c)，则设置了目标高地址寄存器。

步骤 4 通过 Region Control1 寄存器来配置相关设置。

写 0x00000000 到地址 (offset 0x700 + 0x204)，则定义了该区域的数据是 MEM 空间且是地址匹配模式。

步骤 5 写 0x80000000 到地址 (offset 0x700 + 0x208)，则打开了 iATU 使能寄存器。

----结束

20.4.10 DMA 使用初始化

DMA 初始化，主要是配置使能远端 RIE 中断时要使用的 MSI 中断地址和 MSI 中断向量号，中断又分为正常完成中断和异常 Abort 完成中断，对于 MSI 资源有效时，正常和异常完成的 MSI 地址可以配置成相同的，通过读状态寄存器来判断。

表20-12 DMA 写通道相关配置寄存器

Address (0x700+)	名称	参考值
0x2BC	DMA Write Interrupt Status	读中断状态
0x2C4	DMA Write Interrupt Mask	0x0
0x2C8	DMA Write Interrupt Clear	清中断寄存器
0x2CC	DMA Write Error Status	读异常状态
0x2D0	DMA Write Done IMWr Address Low	HOST 端 MSI 接收地址低 32 位



Address (0x700+)	名称	参考值
0x2D4	DMA Write Done IMWr Address High	HOST 端 MSI 接收地址高 32 位
0x2D8	DMA Write Abort IMWr Address Low	异常完成时 HOST 端 MSI 接收地址低 32 位
0x2DC	DMA Write Abort IMWr Address High	异常完成时 HOST 端 MSI 接收地址高 32 位
0x2E0	DMA Write Channels 1 and 0 IMWr Data	0x0 (MSI 中断号, 低 16 位有效, Hi3093 只有一个通道 0)
0x300	DMA Write Linked List Error Enable	0x1

表20-13 DMA 读通道相关配置寄存器

Address (0x700+)	名称	参考值
0x310	DMA Read Interrupt Status	读中断状态
0x318	DMA Read Interrupt Mask	0x0
0x31C	DMA Read Interrupt Clear	清中断寄存器
0x324	DMA Read Error Status Low	读异常状态低 32 位
0x328	DMA Read Error Status High	读异常状态高 32 位
0x33C	DMA Read Done IMWr Address Low	HOST 端 MSI 接收地址低 32 位
0x340	DMA Read Done IMWr Address High	HOST 端 MSI 接收地址高 32 位
0x344	DMA Read Abort IMWr Address Low	异常完成时 HOST 端 MSI 接收地址低 32 位
0x348	DMA Read Abort IMWr Address High	异常完成时 HOST 端 MSI 接收地址高 32 位



Address (0x700+)	名称	参考值
0x34C	DMA Read Channels 1 and 0 IMWr Data	0x0 (MSI 中断号, 低 16 位有效, Hi3093 只有一个通道 0)
0x334	DMA Read Linked List Error Enable	0x1

20.4.11 DMA 写事务流程实例（非链表）

图20-14 1 MB Write DMA Transfer from 0xBEE_BEE0 to 0xCAFE_CAF0

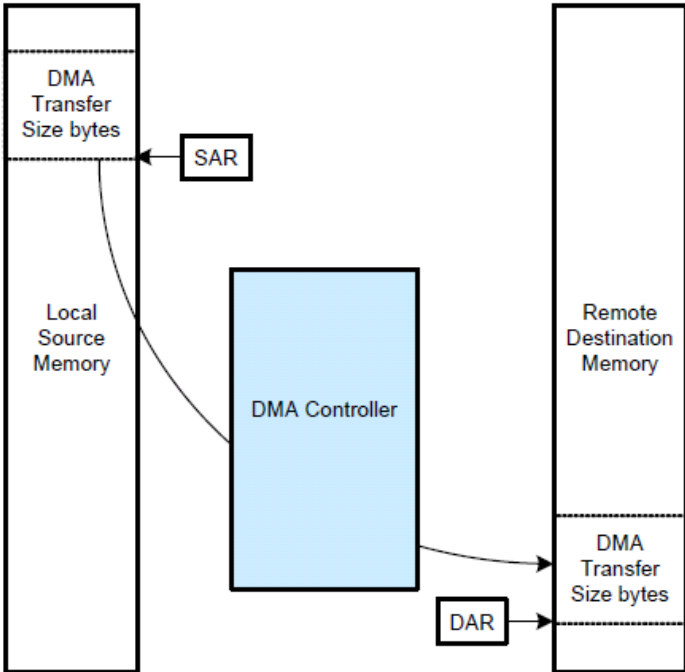
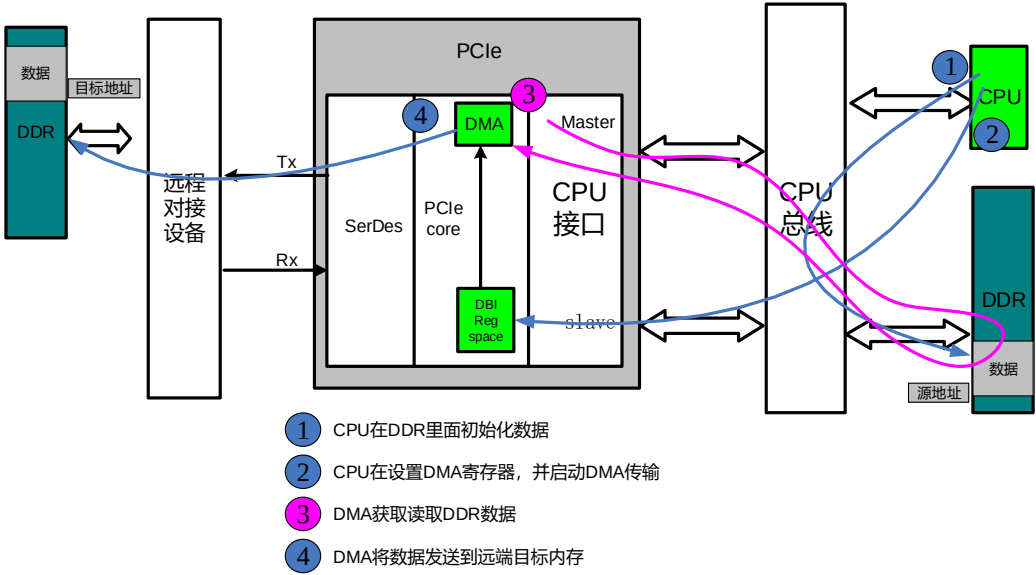




图20-15 PCIe eDMA 写事务（非链表）应用流程参考图



实例说明：

1. 需要搬移的数据都存放在本地内存（例如 DDR），源地址为 0xBEEF_BEE0。
2. 传输是长度 1MB
3. 数据需要搬移到远端设备内存空间，即目标地址是 0xCAFE_CAF0。
4. 先对 eDMA 进行初始化，由于初始化涉及到 eDMA 的寄存器，在访问 eDMA 寄存器之前，需要配置 PCIE_MISC.PCIE_CTRL0.bit[21]=1 和 PCIE_MISC.PCIE_CTRL1.bit[21]=1 选通 DBI 访问使能，具体初始化寄存器和步骤如表 20-14 所示。
5. 数据搬移完毕，PCIe 上报 eDMA_int 中断（各个 PCIE_n 控制器的对应的中断号分别是 242,250，n 取值为 0 和 1）。
6. 软件收到中断后，配置 pcie 系统控制器 0 和 pcie 系统控制器 1 的第 21bit 需要配置为 1 才能访问，即 PCIE_MISC.PCIE_CTRL0[21]=1 和 PCIE_MISC.PCIE_CTRL1[21]=1。
7. 读取 eDMA 的写通道状态寄存器（PCIe DBI 空间偏移 0x9BC）的 bit[1]，如果该 bit 为 1，表示 DMA 写事务结束。

表20-14 DMA 控制器寄存器配置流程（非链表写事务）

Address (0x700+)	名称	值
0x27c	DMA 写事务使能	0x1



Address (0x700+)	名称	值
0x2C4	DMA 写中断屏蔽	0x0
0x36C	DMA 通道索引号=0 DMA 事务方向=write	0x0
0x370	DMA 通道控制器 1 寄存器 本地中断使能 (LIE) =1	0x00000008
0x378	DMA 传输长度	0x00100000
0x37C	DMA 源地址低 32 位	0xBEEF_BEE0
0x380	DMA 源地址高 32 位	0x00000000
0x384	DMA 目标地址低 32 位	0xCAFE_CAF0
0x388	DMA 目标地址高 32 位	0x00000000
0x280	DMA 写事务传输启动 (该寄存器写 0, 即开始启动 DMA 写事务传输)	0x0

20.4.12 DMA 读事务流程实例（非链表）

图20-16 1 MB Read DMA Transfer from 0xBEEF_BEE0 to 0xCAFE_CAF0

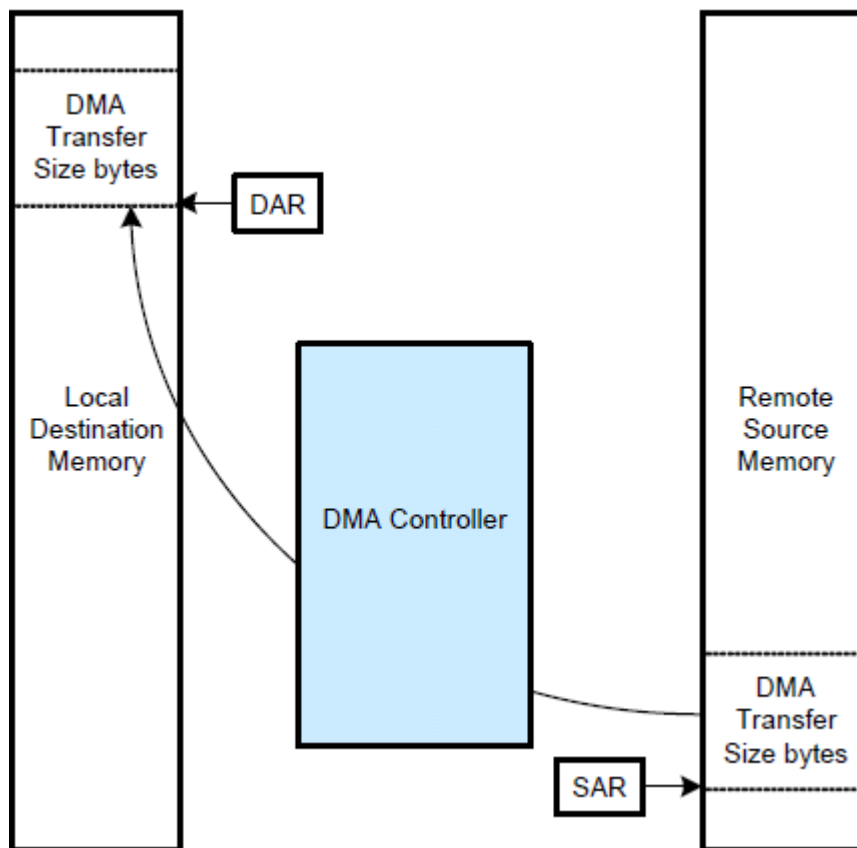
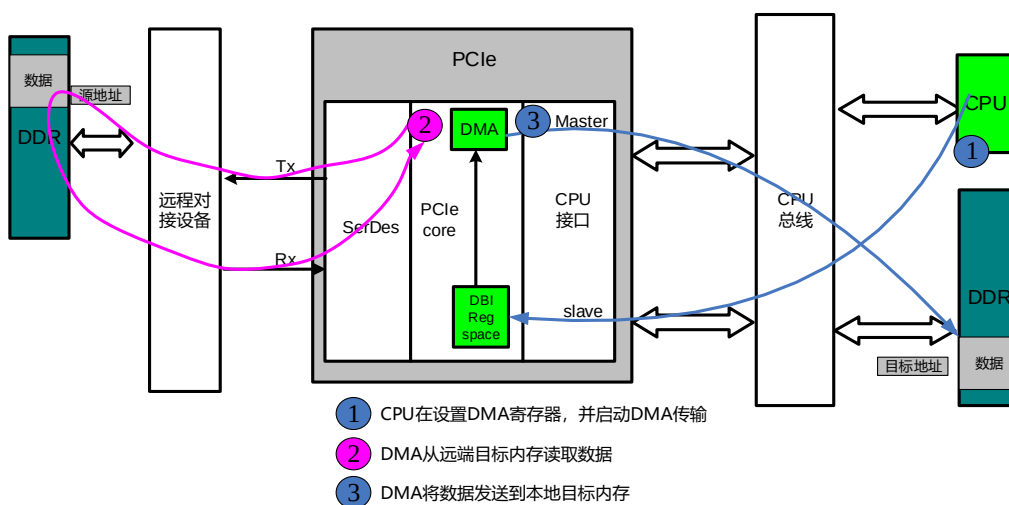


图20-17 PCIe eDMA 读事务（非链表）应用流程参考图





实例说明：

1. 需要搬移的数据都存放在远端设备内存空间，源地址为 0xBEEF_BEE0。
2. 传输是长度 1MB。
3. 数据需要搬移到本地 DDR，即目标地址是 0xCAFE_CAF0。
4. 先对 eDMA 进行初始化，由于初始化涉及到 eDMA 的寄存器，在访问 eDMA 寄存器之前，需要配置 pcie 系统控制器 0 和 pcie 系统控制器 1 的第 21bit 需要配置为 1 才能访问，即 PCIE_MISC.PCIE_CTRL0[21]=1 和 PCIE_MISC.PCIE_CTRL1[21]=1，具体初始化寄存器和步骤如表 20-15 所示。
5. 数据搬移完毕，PCIe 上报 eDMA_int 中断（各个 PCIE_n 控制器的对应的中断号分别是 242,250,n 取值为 0 和 1）。
6. 软件收到中断后，配置 pcie 系统控制器 0 和 pcie 系统控制器 1 的第 21bit 需要配置为 1 才能访问，即 PCIE_MISC.PCIE_CTRL0[21]=1 和 PCIE_MISC.PCIE_CTRL1[21]=1。
7. 读取 eDMA 的读通道状态寄存器（PCIe DBI 空间偏移 0xA10）的 bit[1]，如果该 bit 为 1，表示 DMA 读事务结束。

表20-15 DMA 控制器寄存器配置流程（非链表读事务）

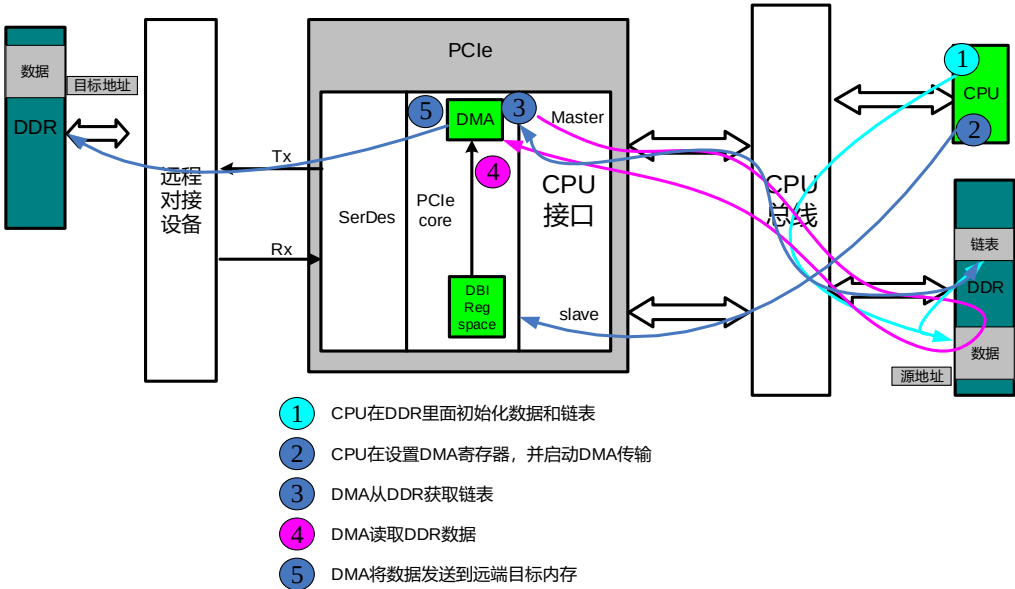
Address (0x700+)	名称	值
0x29c	DMA 读事务使能	0x1
0x318	DMA 读中断屏蔽	0x0
0x36C	DMA 通道索引号=0 DMA 事务方向=read	0x80000000
0x370	DMA 通道控制器 1 寄存器 本地中断使能 (LIE) =1	0x00000008
0x378	DMA 传输长度	0x00100000
0x37C	DMA 源地址低 32 位	0xBEEF_BEE0
0x380	DMA 源地址高 32 位	0x00000000
0x384	DMA 目标地址低 32 位	0xCAFE_CAF0



Address (0x700+)	名称	值
0x388	DMA 目标地址高 32 位	0x00000000
0x2A0	DMA 读事务传输启动 (该寄存器写 0，即开始启动 DMA 读事务传输)	0x0

20.4.13 DMA 写事务流程实例（链表）

图20-18 PCIe eDMA 写事务（链表）应用流程参考图



实例说明：

1. 链表头数据存放在本地 DDR 内存，偏移量 0x90000000 处。

须知

链表头数据只能存放在必须能够支持 burst 总线操作的 DDR 内存。

2. 该链表连接 10 个元素。
3. 每个元素对应着 1MByte 的写操作。



4. 每个元素对应着的源地址（需要搬移的数据存放的地址）分别是 0xA0000000, 0xA0100000, 0xA0200000, 以 1MB 递增，一直到 0xA0900000。
5. 每个元素对应着的目标地址（远端设备内存）分别是 0x40000000, 0x40200000, 0x40400000, 以 2MB 递增，一直到 0x41000000。
6. 软件在内存初始化链表数据，如表 20-16 所示。然后对 eDMA 进行初始化，由于初始化涉及到 eDMA 的寄存器，在访问 eDMA 寄存器之前，需要配置 pcie 系统控制器 0 和 pcie 系统控制器 1 的第 21bit 需要配置为 1 才能访问，即 PCIE_MISC.PCIE_CTRL0[21]=1 和 PCIE_MISC.PCIE_CTRL1[21]=1，具体初始化寄存器和步骤如表 20-17 所示。
7. 数据搬移完毕，PCIe 上报 eDMA_int 中断（各个 PCIE_n 控制器的对应的中断号分别是 242,250, n 取值为 0 和 1）。
8. 软件收到中断后，配置 pcie 系统控制器 0 和 pcie 系统控制器 1 的第 21bit 需要配置为 1 才能访问，即 PCIE_MISC.PCIE_CTRL0[21]=1 和 PCIE_MISC.PCIE_CTRL1[21]=1。
9. 读取 eDMA 的写通道状态寄存器（PCIe DBI 空间偏移 0x9BC）的 bit[1]，如果该 bit 为 1，表示 DMA 写事务结束。

表20-16 DMA 写初始化流程（链表）

Element	Address	Data	说明
0	0x90000000	0x00000001	通道控制
0	0x90000004	0x00100000	传输大小（1 MB）
0	0x90000008	0xa0000000	源低 32 位地址
0	0x9000000c	0x0	源高 32 位地址
0	0x90000010	0x40000000	目的低 32 位地址
0	0x90000014	0x0	目的高 32 位地址
1	0x90000018	0x00000001	通道控制
1	0x9000001c	0x00100000	传输大小（1 MB）
1	0x90000020	0xa0100000	源低 32 位地址
1	0x90000024	0x0	源高 32 位地址



Element	Address	Data	说明
1	0x90000028	0x40200000	目的低 32 位地址
1	0x9000002c	0x0	目的高 32 位地址
2	0x90000030	0x00000001	通道控制
2	0x90000034	0x00100000	传输大小 (1 MB)
2	0x90000038	0xa0200000	源低 32 位地址
2	0x9000003c	0x0	源高 32 位地址
2	0x90000040	0x40400000	目的低 32 位地址
2	0x90000044	0x0	目的高 32 位地址
3	0x90000048	0x00000001	通道控制
3	0x9000004c	0x00100000	传输大小 (1 MB)
3	0x90000050	0xa0300000	源低 32 位地址
3	0x90000054	0x0	源高 32 位地址
3	0x90000058	0x40600000	目的低 32 位地址
3	0x9000005c	0x0	目的高 32 位地址
4	0x90000060	0x00000001	通道控制
4	0x90000064	0x00100000	传输大小 (1 MB)
4	0x90000068	0xa0400000	源低 32 位地址
4	0x9000006c	0x0	源高 32 位地址
4	0x90000070	0x40800000	目的低 32 位地址
4	0x90000074	0x0	目的高 32 位地址
5	0x90000078	0x00000001	通道控制
5	0x9000007c	0x00100000	传输大小 (1 MB)
5	0x90000080	0xa0500000	源低 32 位地址
5	0x90000084	0x0	源高 32 位地址



Element	Address	Data	说明
5	0x90000088	0x40a00000	目的低 32 位地址
5	0x9000008c	0x0	目的高 32 位地址
6	0x90000090	0x00000001	通道控制
6	0x90000094	0x00100000	传输大小 (1 MB)
6	0x90000098	0xa0600000	源低 32 位地址
6	0x9000009c	0x0	源高 32 位地址
6	0x900000a0	0x40c00000	目的低 32 位地址
6	0x900000a4	0x0	目的高 32 位地址
7	0x900000a8	0x00000001	通道控制
7	0x900000ac	0x00100000	传输大小 (1 MB)
7	0x900000b0	0xa0700000	源低 32 位地址
7	0x900000b4	0x0	源高 32 位地址
7	0x900000b8	0x40e00000	目的低 32 位地址
7	0x900000bc	0x0	目的高 32 位地址
8	0x900000c0	0x00000001	通道控制
8	0x900000c4	0x00100000	传输大小 (1 MB)
8	0x900000c8	0xa0800000	源低 32 位地址
8	0x900000cc	0x0	源高 32 位地址
8	0x900000d0	0x41000000	目的低 32 位地址
8	0x900000d4	0x0	目的高 32 位地址
9	0x900000d8	0x00000007	通道控制
9	0x900000dc	0x0	保留
9	0x900000e0	0x90000000	下一个链表指针低 32 地址 (这里 0x90000000 代表数据终止)



Element	Address	Data	说明
9	0x900000e4	0x0	下一个链表指针高 32 地址

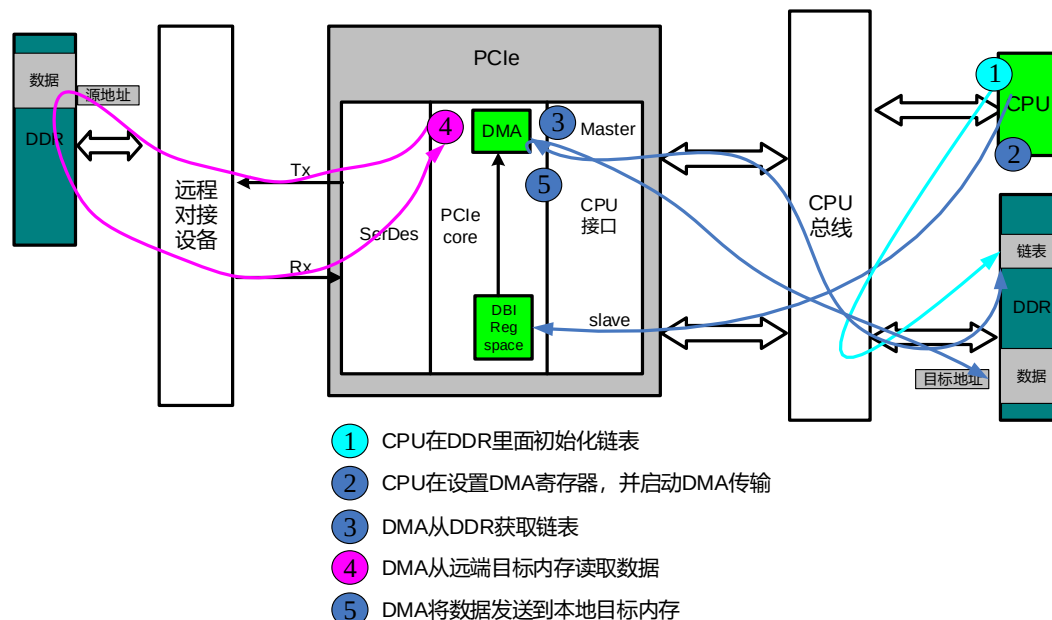
上面的链表初始化完成后，需要设置 DMA 相关寄存器，启动 DMA 传输。

表20-17 DMA 控制器寄存器配置（链表写事务）

Address (0x700+)	名称	值
0x27c	DMA 写事务使能	0x1
0x2C4	DMA 写中断屏蔽	0x0
0x300	DMA 链表写事务错误使能	0x1
0x36C	DMA 通道索引号=0 DMA 事务方向=write	0x0
0x370	DMA 通道控制器 1 寄存器 链表使能=1 循环状态使能=1	0x00000300
0x38C	DMA 链表指针低 32 位	0x90000000
0x390	DMA 链表指针高 32 位	0x00000000
0x280	DMA 写事务传输启动 (该寄存器写 0，即开始启动 DMA 写事务传输)	0x0

20.4.14 DMA 读事务流程实例（链表）

图20-19 PCIe eDMA 读事务（链表）应用流程参考图



实例说明：

1. 链表头数据存放在本地 DDR 内存，偏移量 0x90000000 处。

须知

链表头数据只能存放在必须能够支持 burst 总线操作的 DDR 内存。

2. 该链表连接 10 个元素。
3. 每个元素对应着 1MByte 的写操作。
4. 每个元素对应着的源地地址分别是 0xA0000000, 0xA0100000, 0xA0200000, 以 1MB 递增，一直到 0xA0900000。
5. 每个元素对应着的目标地址（本地内存空间，例如 DDR）分别是 0x40000000, 0x40200000, 0x40400000, 以 2MB 递增，一直到 0x41000000。
6. 软件在内存初始化链表数据，如表 20-16 所示。然后对 eDMA 进行初始化，由于初始化涉及 eDMA 的寄存器，在访问 eDMA 寄存器之前，需要配置 pcie 系统控制器 0 和 pcie 系统控制器 1 的第 21 比特需要配置为 1 才能访问，即



PCIE_CTRL0[21]=1 和 PCIE_CTRL1[21]=1，具体初始化寄存器和步骤如表 20-17 所示。

7. 数据搬移完毕，PCIe 上报 eDMA_int 中断（各个 PCIE_n 控制器的对应的中断号分别是 242,250，n 取值为 0 和 1）。
8. 软件收到中断后，配置 pcie 系统控制器 0 和 pcie 系统控制器 1 的第 21bit 需要配置为 1 才能访问，即 PCIE_MISC.PCIE_CTRL0[21]=1 和 PCIE_MISC.PCIE_CTRL1[21]=1。
9. 读取 eDMA 的读通道状态寄存器（PCIe DBI 空间偏移 0xA10）的 bit[1]，如果该 bit 为 1，表示 DMA 读事务结束。

表20-18 DMA 读初始化流程（链表）

Element	Address	Data	说明
0	0x90000000	0x00000001	通道控制
0	0x90000004	0x00100000	传输大小（1 MB）
0	0x90000008	0xa0000000	源低 32 位地址
0	0x9000000c	0x0	源高 32 位地址
0	0x90000010	0x40000000	目的低 32 位地址
0	0x90000014	0x0	目的高 32 位地址
1	0x90000018	0x00000001	通道控制
1	0x9000001c	0x00100000	传输大小（1 MB）
1	0x90000020	0xa0100000	源低 32 位地址
1	0x90000024	0x0	源高 32 位地址
1	0x90000028	0x40200000	目的低 32 位地址
1	0x9000002c	0x0	目的高 32 位地址
2	0x90000030	0x00000001	通道控制
2	0x90000034	0x00100000	传输大小（1 MB）
2	0x90000038	0xa0200000	源低 32 位地址
2	0x9000003c	0x0	源高 32 位地址



Element	Address	Data	说明
2	0x90000040	0x40400000	目的低 32 位地址
2	0x90000044	0x0	目的高 32 位地址
3	0x90000048	0x00000001	通道控制
3	0x9000004c	0x00100000	传输大小 (1 MB)
3	0x90000050	0xa0300000	源低 32 位地址
3	0x90000054	0x0	源高 32 位地址
3	0x90000058	0x40600000	目的低 32 位地址
3	0x9000005c	0x0	目的高 32 位地址
4	0x90000060	0x00000001	通道控制
4	0x90000064	0x00100000	传输大小 (1 MB)
4	0x90000068	0xa0400000	源低 32 位地址
4	0x9000006c	0x0	源高 32 位地址
4	0x90000070	0x40800000	目的低 32 位地址
4	0x90000074	0x0	目的高 32 位地址
5	0x90000078	0x00000001	通道控制
5	0x9000007c	0x00100000	传输大小 (1 MB)
5	0x90000080	0xa0500000	源低 32 位地址
5	0x90000084	0x0	源高 32 位地址
5	0x90000088	0x40a00000	目的低 32 位地址
5	0x9000008c	0x0	目的高 32 位地址
6	0x90000090	0x00000001	通道控制
6	0x90000094	0x00100000	传输大小 (1 MB)
6	0x90000098	0xa0600000	源低 32 位地址
6	0x9000009c	0x0	源高 32 位地址



Element	Address	Data	说明
6	0x900000a0	0x40c00000	目的低 32 位地址
6	0x900000a4	0x0	目的高 32 位地址
7	0x900000a8	0x00000001	通道控制
7	0x900000ac	0x00100000	传输大小 (1 MB)
7	0x900000b0	0xa0700000	源低 32 位地址
7	0x900000b4	0x0	源高 32 位地址
7	0x900000b8	0x40e00000	目的低 32 位地址
7	0x900000bc	0x0	目的高 32 位地址
8	0x900000c0	0x00000001	通道控制
8	0x900000c4	0x00100000	传输大小 (1 MB)
8	0x900000c8	0xa0800000	源低 32 位地址
8	0x900000cc	0x0	源高 32 位地址
8	0x900000d0	0x41000000	目的低 32 位地址
8	0x900000d4	0x0	目的高 32 位地址
9	0x900000d8	0x00000007	通道控制
9	0x900000dc	0x0	保留
9	0x900000e0	0x90000000	下一个链表指针低 32 地址 (这里 0x90000000 代表数据终止)
9	0x900000e4	0x0	下一个链表指针高 32 地址

表20-19 DMA 控制器寄存器配置流程 (链表读事务)

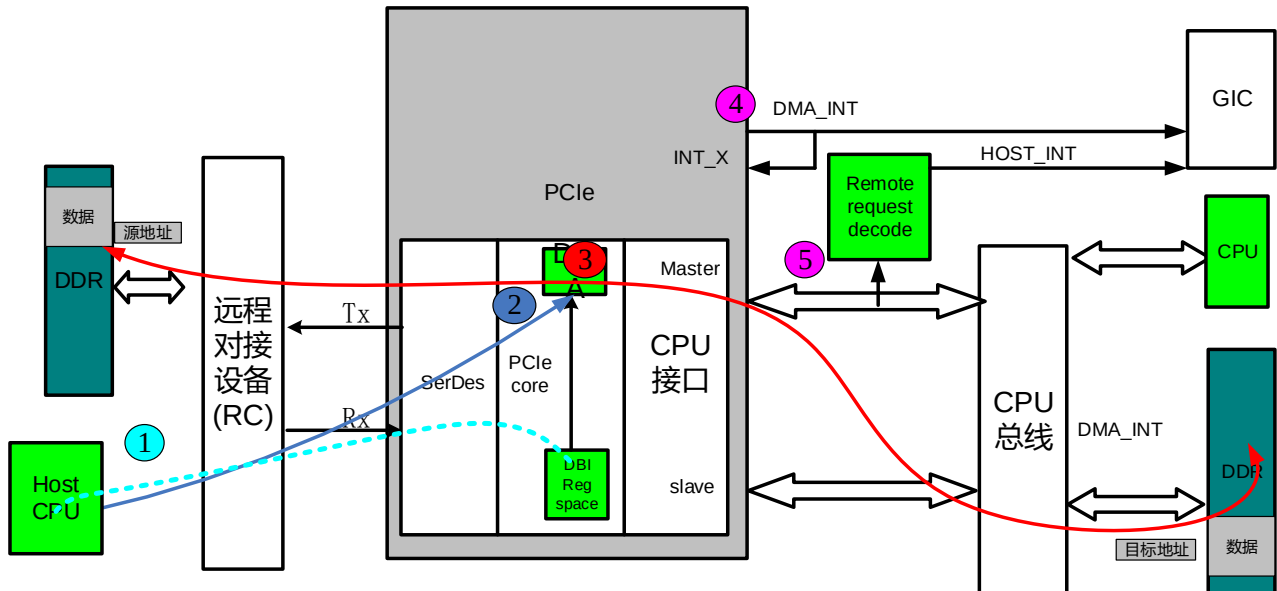
Address (0x700+)	名称	值
0x29c	DMA 读事务使能	0x1
0x318	DMA 读中断屏蔽	0x0



Address (0x700+)	名称	值
0x334	DMA 链表读错误使能	0x00010000
0x36C	DMA 通道索引号=0 DMA 事务方向=read	0x80000000
0x370	DMA 通道控制器 1 寄存器 链表使能=1 循环状态使能=1	0x00000300
0x38C	DMA 链表指针低 32 位	0x90000000
0x390	DMA 链表指针高 32 位	0x00000000
0x2A0	DMA 读事务传输启动 (该寄存器写 0, 即开始启动 DMA 读事务传输)	0x0

20.4.15 XDMA HOST 流程

图20-20 XDMA HOST 流程图



Part1: 数据获取

- ① Host CPU通过配置读写，初始化Hi3093 DMA寄存器
备注：这个方法有个弊端，读写性能可能比较差，因为CFG命令需要等待返回CPL，才能继续下发命令。有个方法是将DMA寄存器映射到BAR空间里面
- ② Host CPU通过配置读写，启动Hi3093 DMA
- ③ Hi3093 DMA将本地DDR数据搬运到远程RC侧DDR，或者从远程RC侧DDR读取数据到本地DDR
- ④ Hi3093 DMA结束，想本地GIC发出DMA_INT中断，通知本地Hi3093 CPU，同时也将中断给到INT_x (MSI_INT)，通知Host CPU

Part2: 两边CPU通讯

- ⑤ Host CPU收到Hi3093发过来的中断后，需要以中断方式通知Hi3093 CPU，可以通过如下方式操作：
通过写某个特定的BAR地址映射的寄存器 (该寄存器地址DDR某个地址)，该寄存器被命中后，给Hi3093 GIC发HOST_int
需要软件映射BAR，才能访问，Hi3093选择该方案。

20.4.16 PCIe link down 处理流程

步骤 1 软件在初始化步骤里面，先复位 pcie (地址偏移: 0x40 的 bit[20]=1)，关闭 PCIe ltssm (PCIE_MISC.PCIE_CTRL7 的 bit[11]=0)，再撤离 pcie 复位 (地址偏移: 0x40 的 bit[20]=0)。

步骤 2 使能 PCIe Ltssm 状态机 PCIE_MISC.PCIE_CTRL7 的 bit[11]=1。



- 步骤 3 读取 PCIE_MISC.PCIE_STATUS0 的 bit[15], bit[5], 如果这两个 bit 都是 1, 在代表 link up 成功之后; 然后设置 PCIE_MISC.PCIE_CTRL7 的 bit[31:29]=0x7, 设置 PCIE_MISC.PCIE_CTRL11 的 bit11=1 然后设置 PCIE_MISC.PCIE_CTRL11 的 bit11=0 (清除 bridge_flush_not 指示)。
- 步骤 4 收到 link down 中断。
- 步骤 5 底软件通知业务软件, PCIe 异常。
- 步骤 6 软件停止下发命令访问 host。
- 步骤 7 查询 PCIE_MISC.PCIE_LNKDN_INT (偏移 0x70) bit[2:0] 确认事件类型, 如果是 bit[0] 置 1 则是 link down 事件执行 [步骤 8](#), 否则 (bit[1] 是 AXI Slave 总线异常; bit[2] 是发生 ECC 错误, 此时查询 PCIE_MISC.PCIE_ECC_INT 偏移 0x50 获得具体状态, 单比特 ECC 可根据实际需求仅打印不复位控制器) 打印错误信息, 跳到 [步骤 11](#)。
- 步骤 8 清除 PCIe link down 中断使能 (PCIE_MISC.PCIE_CTRL11 的 bit[12]=1)。
- 步骤 9 读取 PCIE_MISC.PCIE_STATUS1 的 bit[20], 如果等于 1, 则设置 PCIE_MISC.PCIE_CTRL11 的 bit[11]=1, 然后设置 PCIE_MISC.PCIE_CTRL11 的 bit[11]=0 (清除 bridge_flush_not 指示), 等待 100ms 时间, 重复 [步骤 8](#) 3 次, 如果为 1 (确认 PCIe 总线内部 buffer 已经没有数据了), 则进行下一步骤。
- 步骤 10 关闭 PCIe ltssm (PCIE_MISC.PCIE_CTRL7 的 bit[11]=0)。
- 步骤 11 复位 pcie (地址偏移: 0x40 的 bit[20]=1), 设置 PCIE_MISC.PCIE_CTRL7 的 bit[31:29]=0x0, 再撤离 pcie1 复位 (地址偏移: 0x40 的 bit[20]=0), 读取 PCIE_MISC.PCIE_STATUS4 的 bit[5:0]=0x0, 确认 PCIe 状态机已经被彻底清除。
- 步骤 12 重新 [步骤 1](#), 则可以重新进行业务。

----结束

20.4.17 MSI 中断寄存器配置发送流程

PCIE 工作在 EP 模式时, 可以配置寄存器发送 MSI 中断, 具体操作步骤如下:

- 步骤 1 确认 PCIE 配置空间偏移 0x50 bit[16] 为 1, 即配置 MSI enable。确认 PCIE 配置空间偏移 0x54、0x58 分别配置了 MSI 中断地址的低位和高位。
- 步骤 2 配置 PCIE_MISC.PCIE_CTRL2 (偏移 0x8) .pcie_ven_msi_vector (bit[31:27]) 为要发送的中断号。



步骤 3 配置 PCIE_MISC.PCIE_CTRL2 (偏移 0x8) .pcie_ven_msi_req (bit[26]) 为 1 使能 MSI 发送; 再配置该寄存器为 0。

----结束

20.4.18 MSI 中断接收配置流程

PCIE 工作在 RC 模式时, 可以接收对端发送的 MSI 中断并转化成电平中断进行上报, 具体操作步骤如下:

步骤 1 配置 PCIE 配置空间偏移 0x820、0x824 为 MSI 中断地址的低位和高位。

步骤 2 配置 PCIE 配置空间偏移 0x828 为 0xFFFF 开启低 16 位中断使能。

步骤 3 配置 PCIE 配置空间偏移 0x82C 为 0xFFFF0000 关闭低 16 位中断屏蔽。

步骤 4 此时对端发送一个 MSI 中断后, 本地会上报电平中断, 之后查询 PCIE 配置空间偏移 0x830 查询中断状态。

---- 结束

20.4.19 PCIe 环回测试

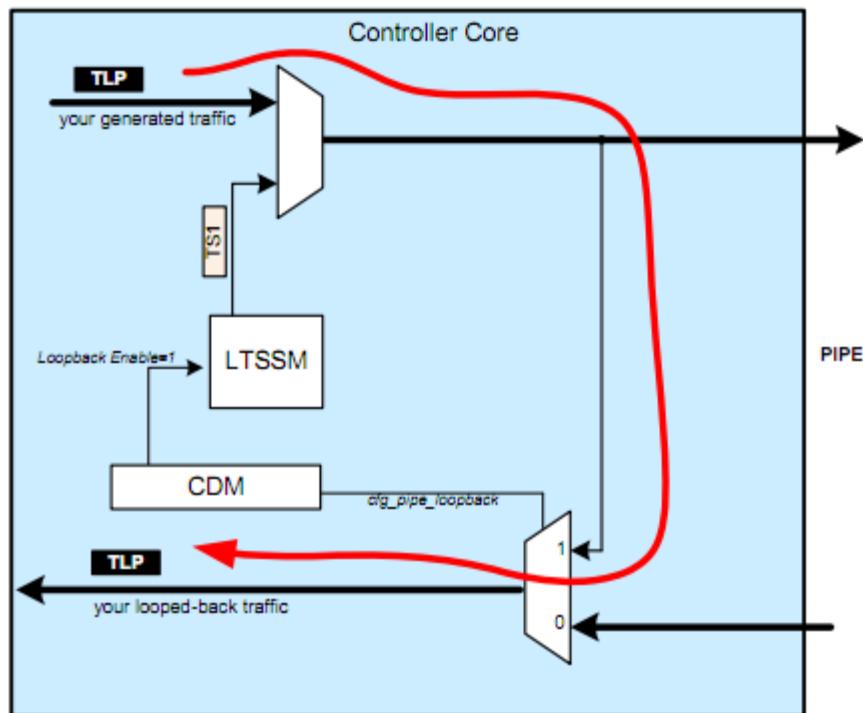
PCIe 支持两种 Local 环回模式。一个是 Local 环回, 一个是 remote 环回。

Local 环回, 不需要建立 link, 需要 PHY 的提供时钟即可, 环回路径不涉及 PHY, 仅在 PCIe 内部进行环回。

Remote 环回, 是属于 PCIe Spec 也不需要建立 link, 需要 PHY 的提供时钟即可, 需要链路稳定 OK, 环回路径涉及 PHY 和 remote PHY。

20.4.20 Local 环回测试流程

图20-21 PCIe 内部本地环回



20.4.20.1 前提条件

Local loopback 包括 PIPE RX 和 PIPE TX 信号，而且只在环回状态下运行，因为控制器自身是不能跟自己建立 link 而进入 L0 (xmlh_ltssm_state=0x11) 状态进行报文的收发。

20.4.20.2 进入 local PIPE loopback 的初始化流程

- 步骤 1 关闭建链使能，配置 PCIE_MISC.PCIE_CTRL7 (偏移 0x1C) bit[11] pcie_app_ltssm_enable 为 0。
- 步骤 2 设置 PCIe 内部 CFG 寄存器，偏移量为 0x700+0x1B8 的 PIPE Loopback Control 寄存器的 bit[31]为 1，使能 PIPE 环回。
- 步骤 3 设置 PCIe 内部 CFG 寄存器，偏移量为 0x700+0x10 的 Port Link Control 寄存器 bit[2] 为 1，使能 loopback。
- 步骤 4 打开建链使能，配置 PCIE_MISC.PCIE_CTRL7 (偏移 0x1C) bit[11] pcie_app_ltssm_enable 为 1。

----结束

20.4.20.3 环回下的枚举和 BAR 启动

在环回状态下，也必须要配置（通过 local CPU 去配置 PCIe 内部 CFG 寄存器空间）BAR 地址（EP），memory/IO 范围（downstream port），设置 PCI 标准 256Byte 配置空间头中的 memory 空间使能和总线 master 使能（即设置 PCIe 内部 CFG 寄存器，配置空间中偏移地址为 0x4 的 bit[0]、bit[1]和 bit[2]为 1）。这样的话，接收过滤才能正常接收 TLP 报文。

20.4.20.4 退出本地环回

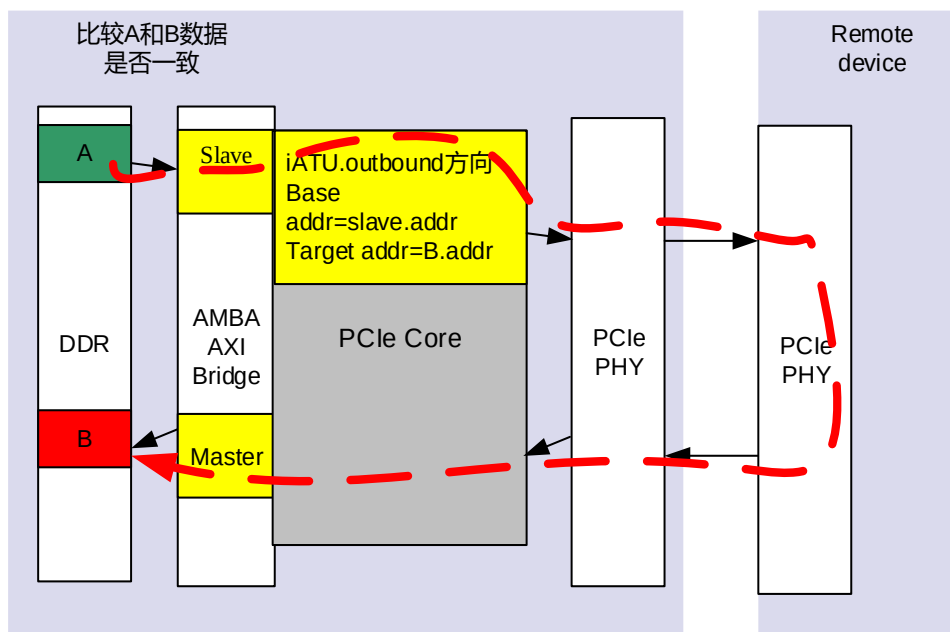
清除 DBI 空间中偏移量为 0x700+0x1B8 的 PIPE Loopback Control 寄存器的 bit31，写其为 0 即可。

清除 DBI 空间中偏移量为 0x700+0x10 的 PIPE Loopback Control 寄存器的 bit2，写其为 0 即可。

20.4.21 Remote 环回测试流程

用于 PCIe 链路检测，如图 20-22 所示。

图20-22 Remote 环回测试





软件往 PCIe 方向发送读写命令，经过环回后，在 DDR 内存里面进行比对，检测 PCIe 链路是否正常。

其配置过程如下：

- 步骤 1 PCIe 初始化成功。（包括 link up，设置 PCIe DBI 空间偏移 0x4 的命令寄存器，bit[2:0]=3b111）。
- 步骤 2 设置 PCIe DBI 空间寄存器：地址偏移为 0x700 + 0x10 的寄存器的 bit[2]设置成 1；loopback 使能。
- 循环读取 PCIe DBI 空间寄存器：地址偏移为 0x700 + 0x28 的寄存器的 bit[5:0]；如果 bit[5:0]为 0x1b(loopback.active)，则 PCIe 进入 loopback 状态。
- 步骤 3 软件申请两块内存，分别是 A 和 B，在 DDR 内存，初始化 A 地址和数据。
- 步骤 4 初始化 iATU，将 outbound 目标地址设置为 B 地址，报文类型是 MEM 类型，如表 20-20 所示。

表20-20 Remote loopback 模式下，iATU 配置（仅供参考）

iATU 寄存器名称	Region0 参考值
说明	访问外设 MEM 空间
VideindexRegister	0x0000_0000 (outbond)
RegionControl1Register	0x0000_0000
RegionControl2Register	0x8000_0000
LowerBaseAddressRegister	PCIe Slave 地址+软件自定义的外设 MEM 空间偏移
UpperBaseAddressRegister	0x0000_0000（高 32 位地址，软件可以自定义）
LimitAddressRegister	PCIe Slave 地址+软件自定义的外设 MEM 空间偏移+外设 MEM 空间大小
LowerTargetAddressRegister	DDR 内存 B 地址低 32 位
UpperTargetAddressRegister	DDR 内存 B 地址高 32 位



步骤 5 软件将 A 地址和数据写到 PCIe 侧。

步骤 6 软件比较 A 地址和 B 地址数据是否一致。

----结束

须知

PCIE 进入环回状态后，再取消环回使能，让其退出环回状态，但是 PCIE 是不能重新进入 L0 状态（即链路 linkup），而是 LinkDown，软件需要重新初始化 PCIE，PCIE 才重新正常工作。

20.4.22 SerDes 初始化

PCIe 复位后使用时初始化步骤如下：

步骤 1 设置 SerDes 的工作模式。

步骤 2 复位 SerDes。

步骤 3 撤销 SerDes 的复位。

步骤 4 查看 SerDes 的状态是否锁定。

----结束

20.4.23 软复位

对接器件容许的情况下，可以任意时间对 PCIe 工作逻辑进行软复位，不会对系统造成影响。通过配置相应的寄存器 bit，可实现对 PCIe 工作逻辑的单独软复位。

PCIe 软复位步骤如下：

步骤 1 配置相应的复位寄存器 bit 为 1，软复位 PCIe。

步骤 2 等待 1ms，保证 PCIe 完全复位。

步骤 3 配置相应的复位寄存器 bit 为 0，撤离 PCIe 的软复位。

步骤 4 初始化 PCIe，进入到正常的功能模式。

--结束



须知

复位控制寄存器位于 HOST_SUBSYS_REG (基地址 0x08B20000) .
HOST_RST_ENABLE (偏移 0xC), bit[0]对应 PCIE0 复位, bit[1]对应 PCIE1 复位,
bit[5]对应 MCTP 复位, PCIE1 复位时, MCTP 应该伴随一起复位。

20.5 性能

20.5.1 吞吐率 (Throughput)

有效的带宽计算非常复杂, 涉及到信用管理, 缓冲器大小和模式, 不同的数据事务处理, 包的长度, 最大载荷等。这里只给出比较纯粹的带宽计算, 仅供参考。

开销:

- 8b10b 编码机制会占用 20%开销。
- 在数据链路层的流控制更新数据包和握手数据包 (DLLP), 会大约占用 1%到 5% 开销。
- 设计参数的选择 (载荷, 缓存区大小, 软件操作等) 的影响非常大。下面给出仅从载荷的选择来看, 带宽的利用率情况。

表20-21 PCIe 理想带宽利用率

-	载荷	包头	ECRC	统计	带宽利用率		备注
					8b/10b 开销前	8b/10b 开销后	
最坏	16	16	4	16/44	36%	29%	有 ECRC
典型 1	128	16	0	128/152	84%	67%	没有 ECRC
典型 2	256	16	0	256/280	91%	73%	没有 ECRC
典型 3	512	16	0	512/536	96%	76%	没有 ECRC



-	载荷	包头	ECRC	统计	带宽利用率		备注
					8b/10b 开销前	8b/10b 开销后	
最好	4096	16	0	4096/ 4116	99%	79%	没有 ECRC

表20-22 TLP 报文格式

STP (DLLP)	SEQ (DLLP)	TLP Head	TLP Data	ECRC	LCRC (DLLP)	END (DLLP)
1Byte	2Byte	12/16Byte	16/128/2 56/512/ 4096 Byte	4Byte (可选)	4Byte	1Byte

图20-23 32-bit X4 Core @500Mhz:64-bit AXI @500Mhz:Memory Writes

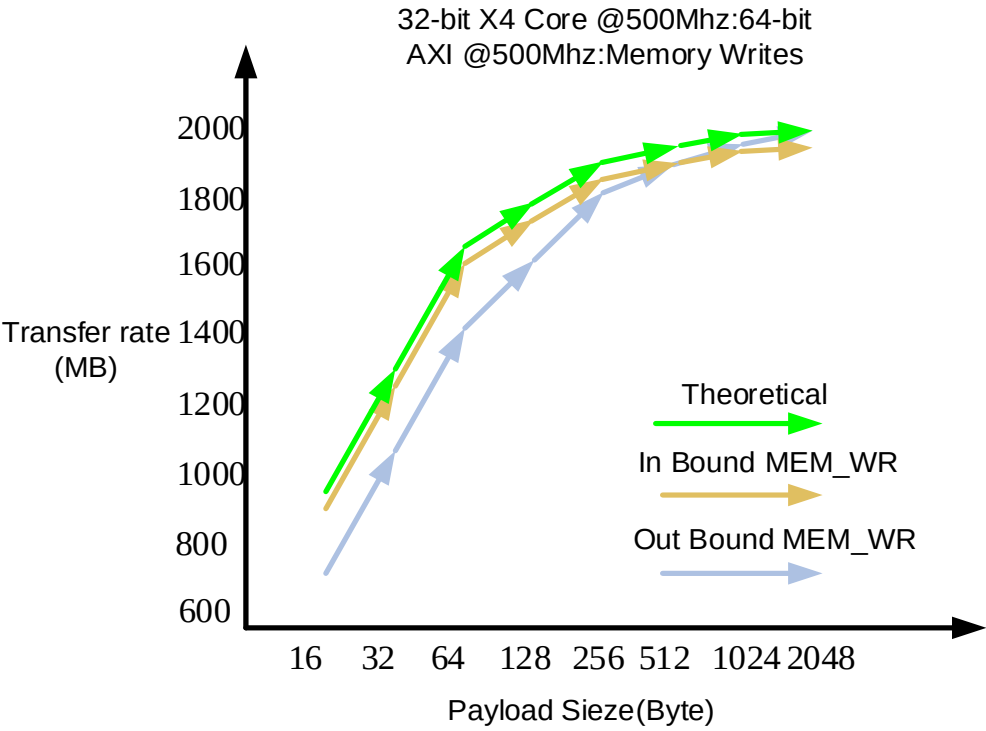
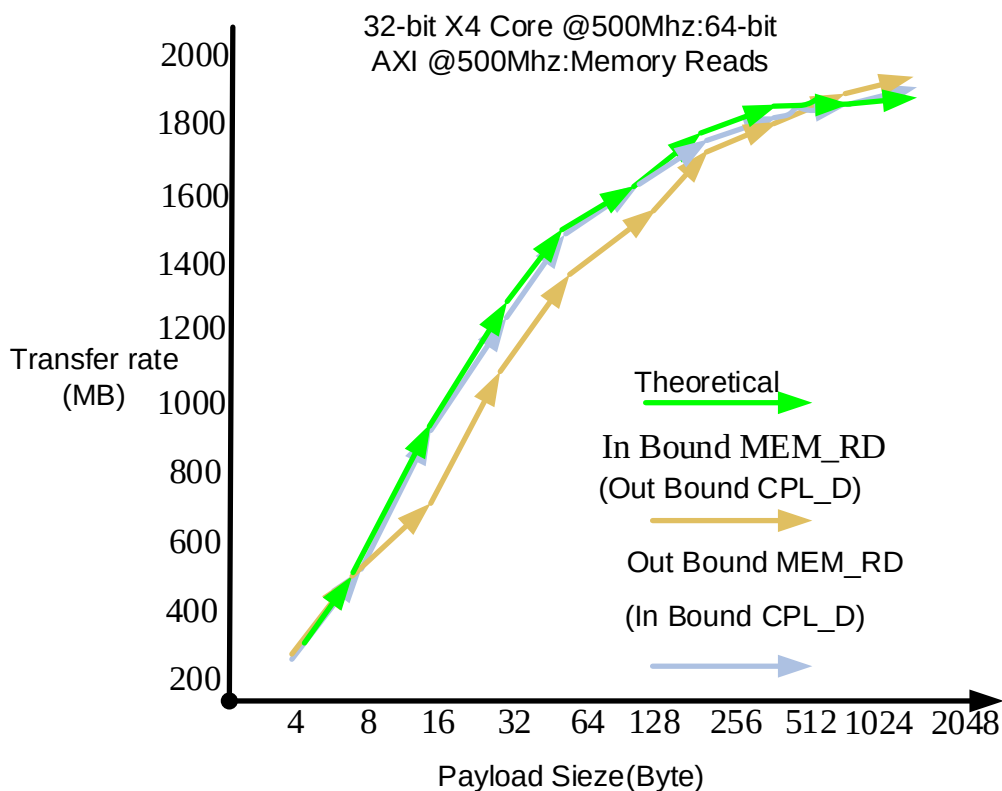




图20-24 32-bit X4 Core @500Mhz:64-bit AXI @500Mhz:Memory Reads



20.6 调试 (Debug)

20.6.1 关键信号状态寄存器

PCIE_MISC 包含关键的中断状态信息 (PCIE_STATUS0\PCIE_STATUS1)、LTSSM 状态机信息 (PCIE_STATUS4~7)、总线相关 DFX (PCIE_STATUS8~9)，在问题定位中经常使用。

20.6.2 PCIe 专用错误上报机制

PCIe Spec 使用的错误上报机制 (在 PCIe DBI 空间偏移 0x104 和 0x110)，在 PCIe 配置空间寄存器提供了相关寄存器接口和定义。



20.6.3 PCIe 专用状态寄存器查询，链路建立，链路状态机寄存器查询

PCIe 能够正确传输数据的前提是链路 linkup，当链路 linkdown 的时候，PCIe 将无法传输数据，PCIe 的链路状态查询操作如下：

步骤 1 查询 PCIE_MISC 寄存器 PCIE_STATUS0（偏移 0x80）的 xmlh_link_up 和 xtlh_link_up 的值均为 1 表示 PCIe 初始化完毕，进入正常 Link Up 工作状态。

----结束

20.7 注意事项

操作 PCIe 额外注意事项如下：

- PCIe 的初始化遵守 GRIO 和 RAB 的初始化操作流程。
- SerDes 的初始化遵守 SerDes 初始化操作流程。
- PCIe 的软复位遵守软复位操作流程。
- PCIe 的各个类型操作遵守对应操作流程。

须知

PCIE 控制器可能出现的挂死场景，软件使用需注意：

- 在 PIPE Loopback 模式下，通过 PCIE Slave 口访问地址在该 PCIe 控制器 Slave 口系统译码分配的地址空间范围之内时会造成总线挂死。
- 在 PCIe 没有建立链路之前对 PCIe Slave 口进行访问，会造成总线挂死。
- Cle 控制器未解复位或者 PCIe AXI Slave 口时钟被门控情况下，对 PCIe Slave 口进行访问会造成总线挂死。
- PCIe 控制器正在下发业务期间，对 PCIe 控制器进行复位可能会造成总线挂死。
- 在 DMA 正在搬运数据期间，对 DMA Context 寄存器进行重复配置，DMA 行为会产生异常，软件在配置 DMA 期间需要读取 DMA 状态寄存器，确认 DMA 未再数据搬运期间再进行寄存器的配置，或者使用链表模式进行数据的搬运，减少 DMA 寄存器配置次数。



- PCIe 链路不稳定，反复在 L0 和 recovery 快速跳变，或者一直在非 L0 下，访问 PCIe Slave 口可能会导致总线挂死。7.PCIE RC 模式时，对接器件异常（例如其芯片内部总线被复位住，总线时钟关断），通过 PCIe Flow Credit 反压 PCIe，访问 PCIe Slave 可能会导致挂死。为了避免该问题。软件需要设置 PCIE_MISC.PCIE_CTRL8 bit[17].pcie_no_credit_pro_dis=0。
-



21 CANBUS

21.1 概述

Hi3093 集成了两个独立的 CAN 控制器，内部挂接到 APB 总线上，外部通过的 CAN 收发器将 TX，RX 电平信号转为电压差分信号连接到 CAN 总线，与具有 CAN 接口的设备通讯。

21.2 特性

- 支持两路 CANBUS 接口。
- 支持 CANbus2.0 协议，A 和 B 部分，包含标准与扩展的数据和远程帧，0~8Byte 长度，最高支持 1Mbit/sec 速率。
- 极性可配置。
- 支持最大 Message Buffers 为 15，每个都可独立配置收发。
- 每个 Buffer 有单独的标识符屏蔽，支持全局屏蔽和独立屏蔽，其中 BUFFER 14 支持独立标识符屏蔽，BUFFER 0~13 支持全局标识符屏蔽。
- 同一个 ID 可以设置多重 Buffers。
- 支持 1~15 个中断源，每个中断源都可屏蔽，支持所有 Message buffers 收发或错误所产生的中断上报为一个中断输出。
- 当成功收发数据后支持 16 位计数器可配时间戳控制应答时间。
- 支持诊断模式下环回和只接收测试。



21.3 关键术语

表21-1 CANBUS 关键术语描述

术语	描述
CAN	Controller Area Network，控制器局域网。

21.4 功能描述

21.4.1 报文内存单元结构

CAN 控制器含有 MEM 的数据 BUFFER RAM，共有 8 个 32bit 寄存器，用于储存收到的报文或者待发送的 FIFO 报文，称之为报文内存单元。

须知

当芯片复位（不掉电的情况下），部分已配置的报文内存单元仍保持原值，故需要对所有的报文内存单元初始化。

表21-2 报文内存单元结构表

AD DR	Buf fer Re gis	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x 1c	ID 1	XI2 8 ID1 0	XI2 7 ID 9	XI2 6 ID 8	XI2 5 ID 7	XI2 4 ID 6	XI2 3 ID 5	XI2 2 ID 4	XI2 1 ID 3	XI2 0 ID 2	XI1 9 ID 1	XI1 8 ID 0	SR R RT R	ID E	XI1 7	XI1 6	XI 15
0x 18	ID 0	XI1 4	XI1 3	XI1 2	XI1 1	XI1 0	XI9	XI8	XI7	XI6	XI5	XI4	XI3	XI2	XI1	XI0	RT R
0x 14	DA TA 0	Da ta 1.7	Dat a 1.6	Da ta 1.5	Dat a 1.4	Da ta 1.3	Dat a 1.2	Da ta 1.1	Dat a 1.0	Da ta 2.7	Dat a 2.6	Da ta 2.5	Da ta 2.4	Da ta 2.3	Da ta 2.2	Da ta 2.1	Da ta 2.0



AD DR	Buf fer Re gis	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x 10	DA TA 1	Da ta 3.7	Dat a 3.6	Da ta 3.5	Dat a 3.4	Da ta 3.3	Dat a 3.2	Da ta 3.1	Dat a 3.0	Da ta 4.7	Dat a 4.6	Da ta 4.5	Da ta 4.4	Da ta 4.3	Da ta 4.2	Da ta 4.1	Da ta 4.0
0x 0c	DA TA 2	Da ta 5.7	Dat a 5.6	Da ta 5.5	Dat a 5.4	Da ta 5.3	Dat a 5.2	Da ta 5.1	Dat a 5.0	Da ta 6.7	Dat a 6.6	Da ta 6.5	Da ta 6.4	Da ta 6.3	Da ta 6.2	Da ta 6.1	Da ta 6.0
0x 08	DA TA 3	Da ta 7.7	Dat a 7.6	Da ta 7.5	Dat a 7.4	Da ta 7.3	Dat a 7.2	Da ta 7.1	Dat a 7.0	Da ta 8.7	Dat a 8.6	Da ta 8.5	Da ta 8.4	Da ta 8.3	Da ta 8.2	Da ta 8.1	Da ta 8.0
0x 04	TS TP	TS TP 1 5	TS TP 1 4	TS TP 1 3	TS TP 1 2	TS TP 1 1	TS TP 1 0	TS TP 9	TS TP 8	TS TP 7	TS TP 6	TS TP 5	TS TP 4	TS TP 3	TS TP 2	TS TP 1	TS TP 0
0x 00	CN TS T	DL C3	DL C2	DL C1	DL C0	reserved				PR I3	PR I2	PR I1	PR I0	ST 3	ST 2	ST 1	ST 0

RTR 的全称为“远程发送请求位 (Remote Transmission Request BIT)”，RTR 位在数据帧里必须为“显性”，而在远程帧里必须为“隐性”。

SRR 的全称是“替代远程请求位 (Substitute Remote Request BIT)”。

SRR 是一隐性位。它在扩展格式的标准帧 RTR 位位置，因此代替标准帧的 RTR 位。

IDE 的全称是“识别符扩展位 (Identifier Extension Bit)”。

IDE 位属于：

- 扩展格式的仲裁场。
- 标准格式的控制场。

标准格式里的 IDE 位为“显性”，而扩展格式里的 IDE 位为“隐性”。

21.4.2 CAN 所支持的四种帧格式

21.4.2.1 数据帧

数据帧将数据从发送器传输到接收器。按照所识别设备的范围，分标准数据帧（12 位设备标识）和扩展数据帧（29 位设备标识）。



图21-1 标准数据帧

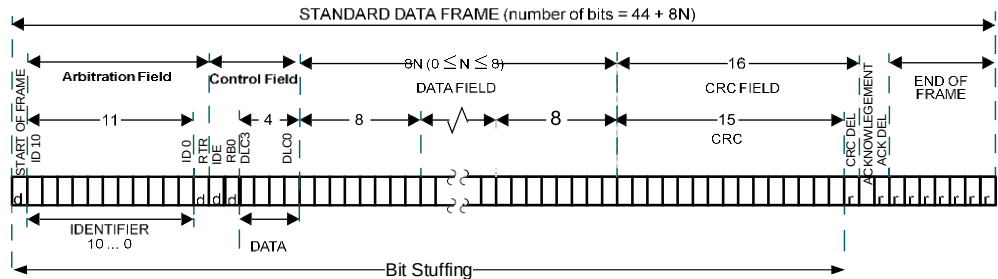
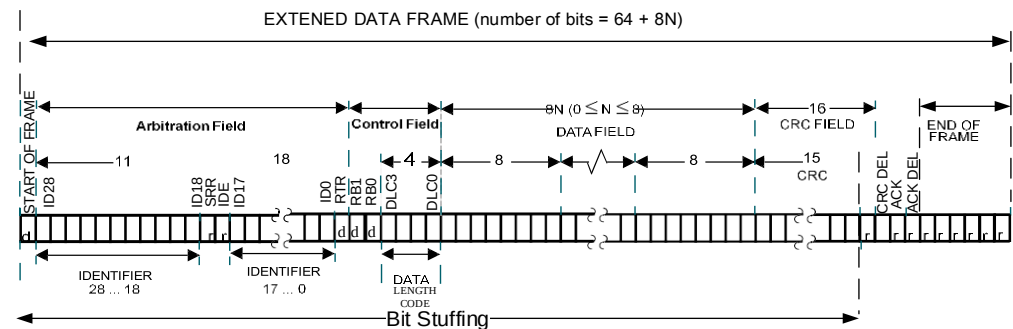


图21-2 扩展数据帧



须知

图 21-1、图 21-2 中 d=dominant 显性，r=recessive 隐性。

21.4.2.2 远程帧

通过发送远程帧，作为某数据接收器的节点通过其他资源节点对不同的数据传送进行初始化设置，请求其发送具有同一识别符的数据帧，数据帧和相应的远程帧是由相同的识别符命名的。



图21-3 标准远程帧

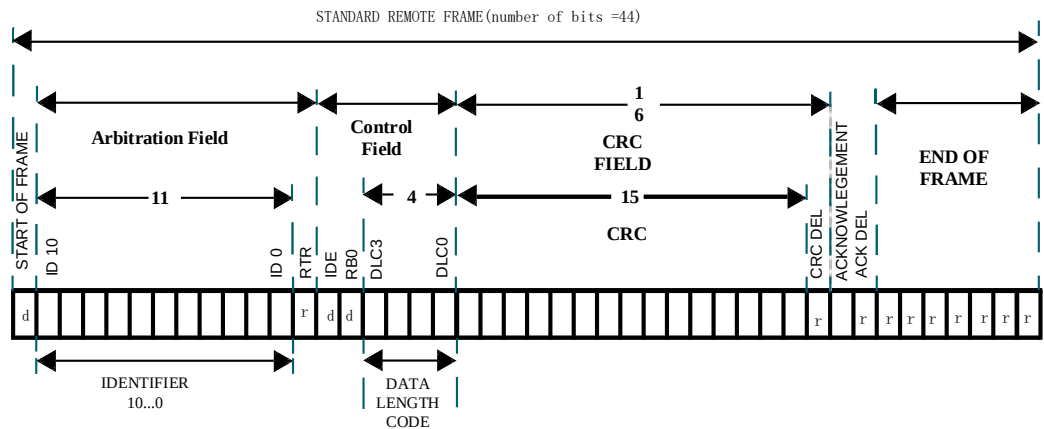
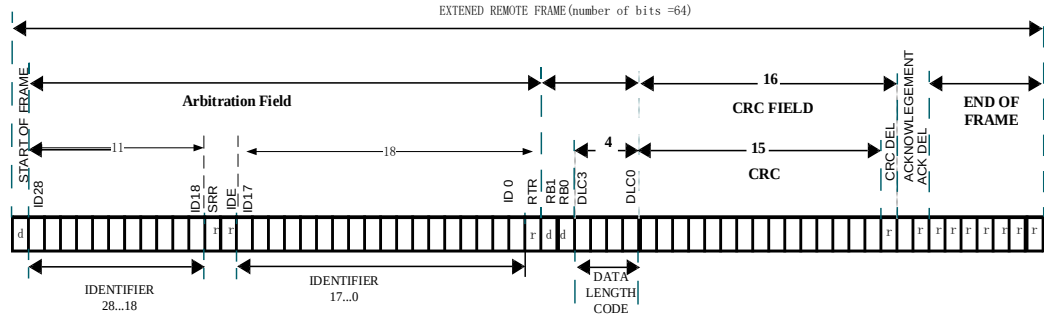


图21-4 扩展远程帧



须知

图 21-3、图 21-4 中 d=dominant 显性，r=recessive 隐性。

21.4.2.3 错误帧

任何单元检测到总线错误就发送错误帧。注意 CAN 协议中连续五个相同的位，第六个需要加入上一个的反码，而错误帧区别与正常数据就在于连续六个相同的位值。当 CAN 总线上某个节点接收数据错误后，会立马发送一个错误帧到总线，总线上所有接收节点放弃本次数据，而发送站则重新发送。

21.4.2.4 过载帧

过载帧用以在先行的和后续的数据帧（或远程帧）之间提供一附加的延迟。在上一帧结束后，插入六个相同极性的位值区别以下内容不是数据，和错误帧同理。



21.4.2.5 帧间隙

帧间隙用于分隔数据帧和远程帧前面得任何帧。帧间空间包括 Intermission（间歇）、Bus Idle（总线空闲）的位域。如果“错误被动”的站已作为前一报文的发送器时，则其帧空间除了间歇、总线空闲外，还包括称作挂起传送（SUSPEND TRANSMISSION）的位域。具体见 CANBUS 2.0 协议。

说明

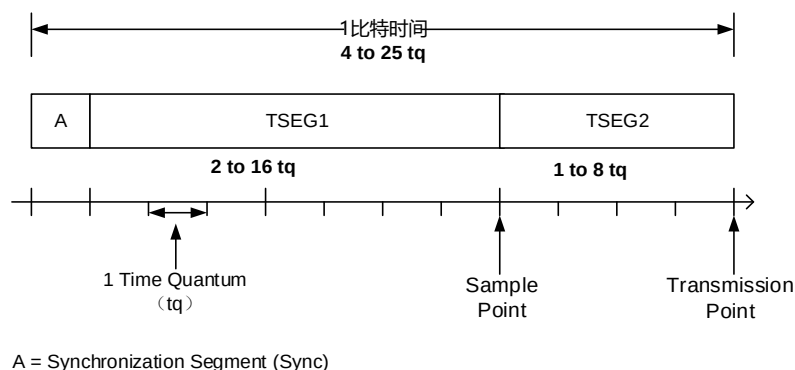
错误帧、过载帧和帧间隙是 canbus 控制器自动发送的，具体帧格式请参考协议文档《CAN Specification Version2.0》。

CANBUS 标准获取路径：<https://www.can-cia.org/>

21.4.3 CAN 波特率设置

在同一个 CAN 网络中，只能使用同一种波特率进行通信。由于 CAN 网络上有不同的节点，各节点控制器的晶振频率不一定相同，使得波特率设置成为 CAN 通信是否成功的重要工作。若某个节点波特率设置错误，将会影响整个 CAN 网络通信。在仲裁期间，当多个节点同时发送报文，若节点波特率不同，可能导致监听时采样错位，则其中一些发送节点由于位错误等不断累加，将会转变为“错误被动”节点而退出总线。相对其他网络，CAN 对波特率的容差能力较强，容许波特率设置存在微小的差别。

图21-5 1 比特时间结构示意图



各时间段解释如下：

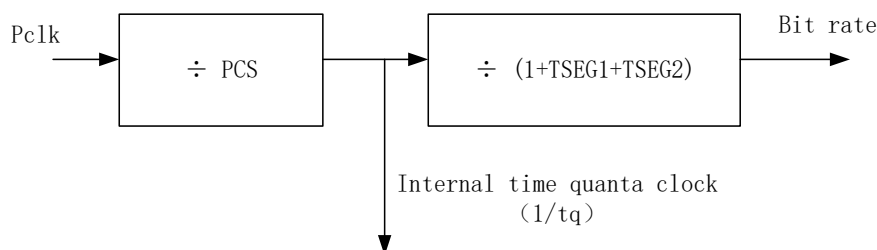
同步段（A）：比特时间的同步段用于同步总线上不同节点，时间长度固定，为 1tq。

相位缓冲段 1（TSEG 1）：用于补偿边沿阶段的错误，时间长度 2tq ~ 16tq。

相位缓冲段 2（TSEG2）：用于补偿边沿阶段的错误，时间长度 1tq ~ 8tq。

采样点 (Sample Point) 在 TSEG 1 和 TSEG 2 之间, 监听总线的电平即为采样点时刻所采到的电平。时间份额 t_q 可以通过 CTIM (CAN 波特率寄存器) 分频 APB 时钟得到。CAN 波特率最大可达 1Mbit/s。在配置波特率之前, 还需要描述 1 个参数: 重新同步跳转宽度 (SJW Synchronization Jump Width), 重新同步的结果, 使相位缓冲段 1 加长, 或使相位缓冲段 2 缩短, 相位缓冲段加长或缩短的数量有一个上限, 此上限由 SJW 给定。

图21-6 比特率时钟生成图



通过配置寄存器 CTIM (CAN 波特率寄存器) 中相关参数, 可得到希望的波特率:

寄存器 CTIM[2:0], TSEG2 的取值为 CTIM[2:0]+1。

寄存器 CTIM[6:3], TSEG 1 的取值为 CTIM [6:3]+1。

寄存器 CTIM[8:7], SJW 的取值为 $\min(4, \text{Phase_Seg1})-1$ 。

寄存器 CTIM[15:9], PSC 取值为 CTIM [15:9]+2, 有效值为 0~63。

公式为:

$$\text{busclock} = \frac{\text{pclk}}{\text{PCS} \times (1 + \text{TSEG1} + \text{TSEG2})}$$

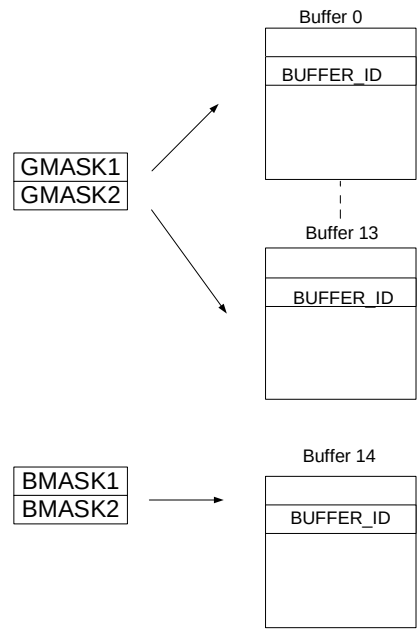
例如 pclk 为 50MHz, 寄存器 CTIM 配 0x1089, 便可配置波特率为 1Mbit/sec。

21.4.4 接收标识符过滤

支持两个 32 位掩码过滤 CAN 总线中不需要的标识符, 分别为 GMASK 和 BMASK。接收过滤 0~13 的数据 BUFFER 中的 ID 标识符为 Global Filtering Mask (GMASK), 单独接收过滤 BUFFER 14 中的 ID 标识符为 Separate Filtering Mask(BMASK)如图 21-7 所示。

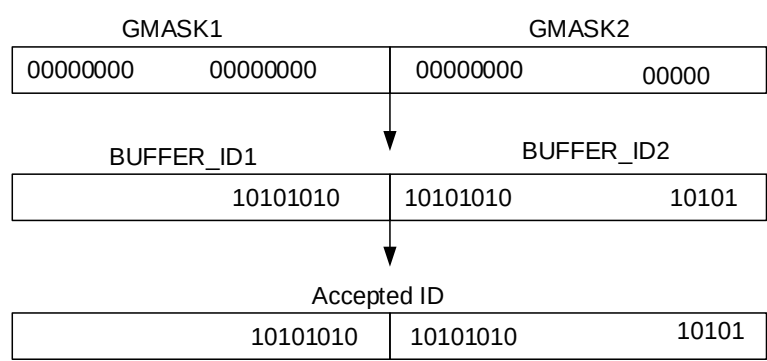


图21-7 接受过滤结构图



例如，当 GMASK 设置为 0，及对接受的 ID 标识符不进行掩码，CAN 所支持如图 21-8 所示。

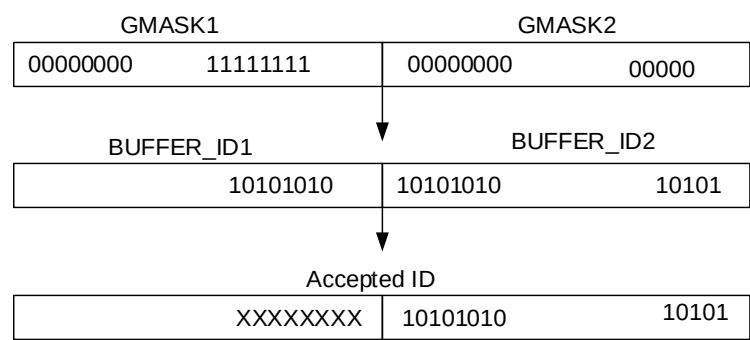
图21-8 接收不过滤 ID 示意图



当 GMASK 配置为 1，将部分 ID 进行掩码，如图 21-9 所示，“X” 表示改部分 ID 被屏蔽不识别。



图21-9 接收过滤 ID 示意图

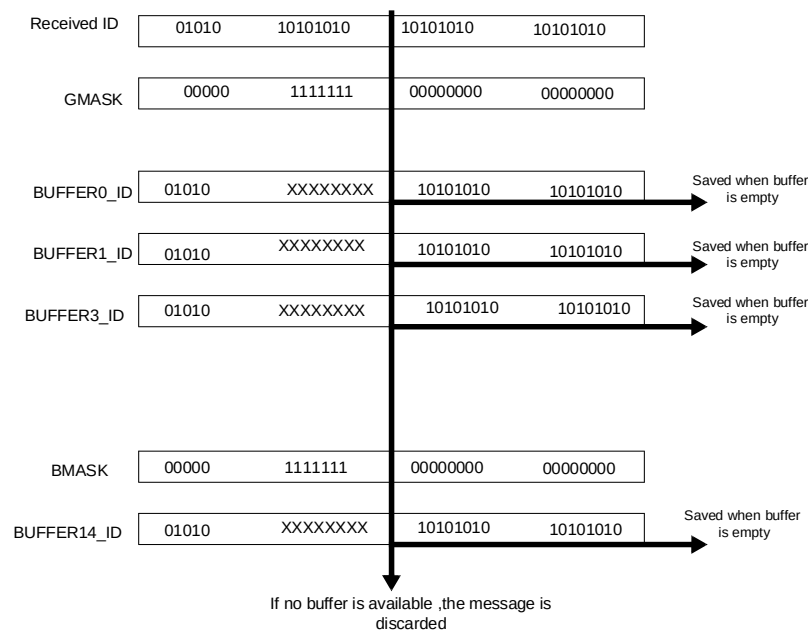


21.4.5 BUFFLOCK 功能

通过配置寄存器 CGCR bit[3]buff_lock，可开启 BUFFLOCK 功能，此功能可使得接收 BUFFER 在收到一个有效帧后自动锁定，当 CPU 读到数据并在 BUFFER 中写 RX_READY 状态，BUFFER 解锁。

例如图 21-10 所示，在打开 BUFFLOCK 功能后，若干个 messages 以相同的 ID 接收，GMASK 设置不掩码，第一个帧被保存在 BUFFER 0，BUFFER 0 被锁定，下一帧被保存在 BUFFER 1，一直到 BUFFER 13 被写入，过程中同一 ID 的数据不会覆盖上一个 BUFFER，就是 BUFFLOCK 的作用。

图21-10 BUFFLOCK 打开后的数据储存示意图



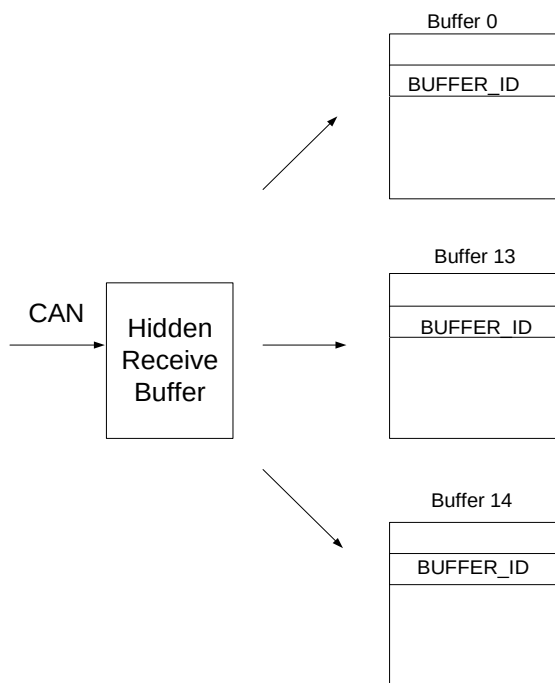


21.4.6 接收功能描述

21.4.6.1 接收结构

HCMB 寄存器为隐藏 BUFFER，所有接收的帧都会先储存在隐藏 BUFFER，用接收 ID 标识符对其他 BUFFER ID 标识符进行掩码、状态等比较，当接收的 EOF 部分到达后，隐藏 BUFFER 的内容便会被复制到所对应的 BUFFER ID 中，过程中 CPU 不可以对隐藏 BUFFER 进行访问。

图21-11 接收过程示意图



21.4.6.2 接收程序

步骤 1 配置接收的掩码寄存器（GMASK 或 BMASK 寄存器）。如果接收远程帧，配置 RTR/XRTR 为 0。

步骤 2 配置 BUFFER ID（ID0 和 ID1 寄存器）。

步骤 3 配置 BUFFER 状态为 RX_READY（CNSTAT 寄存器）。如果接收远程帧并回复数据帧，配置 BUFFRT 状态为 TX_RTR。

----结束



21.4.7 发送功能描述

21.4.7.1 发送优先级配置

CAN 控制器可支持发送 BUFFER 的优先级配置。可通过 CNSTAT 寄存器的 PRI bit[7:4]来配置 15 个 BUFFER ID 的优先级。当所有 BUFFER 的 PRI 都不配为 0，其优先级顺序已 BUFFER 0~14 递减排序，如表 21-3 所示。当所有 BUFFER 的 PRI 优先级都配有值，如表 23-3 所示，优先级顺序当 PRI 配为 14 时最低，配为 0 时最高。如果两个 BUFFER 的优先级相同，则按小 BUFFER 号的优先级更高。

表21-3 所有 BUFFER 发送优先级配为 0 表

TXPRI	Buffer Number	PRI	TX Priority
0	0	0	Highest
0	1	1	
...	...	...	...
0	14	14	lowest

表21-4 所有 BUFFER 发送优先级配不为 0 表

TXPRI	Buffer Number	PRI	TX priority
14	0	224	Lowest
13	1	209	-
12	2	194	-
11	3	179	-
10	4	164	-
9	5	149	-
8	6	134	-
7	7	119	-
6	8	104	-
5	9	89	-
4	10	74	-
3	11	59	-
2	12	44	-
1	13	29	-



TXPRI	Buffer Number	PRI	TX priority
0	14	14	highest

21.4.7.2 发送程序

- 步骤 1 配置 CNSTAT 寄存器，BUFFER 状态为 TX_NOT_ACTIVE。
- 步骤 2 加载 BUFFER ID 标识符和数据寄存器。
- 步骤 3 对于远程帧，识别符的 RTR 位应该被设置，接收掩码寄存器的 RTR/XRTR 设置成 1，并且加载数据字节可被忽略。
- 步骤 4 配置 CNSTAT 寄存器状态进行期望操作。
- 步骤 5 TX_ONCE：引发单个数据帧的传输。
- 步骤 6 TX_ONCE_RTR：引发单个数据帧的传输并等待接收远程帧去触发下一个数据帧。
- 步骤 7 TX_RTR：等待远程帧去触发数据帧的传输。

----结束

21.4.8 中断

CAN 控制器支持将一个中断向量上报给 CPU，这个中断向量来源于以下两个条件：

CAN 数据传输：

- 数据 BUFFER 接收完毕一个有效数据帧。
- 成功发送一个数据帧。
- 成功响应一个远程帧。
- 成功发送一个远程帧。

CAN 错误条件：

- 在根据了 CIPND 寄存器和 CEDIAG 寄存器中的响应的错误诊断上报。
- 每个 BUFFER 发送或接收的中断都可通过 CIEN 寄存器配置屏蔽，中断状态可在 CIPND 寄存器中读取，并且通过 CICLR 寄存器清除中断。



- 中断请求优先级配置，通过 CSTPND 寄存器的 IRQ[4] 和 IST[3:0] 的编码配置，来为各个 BUFFER 配置中断请求优先级，如表 21-5 所示，中断请求优先级由高到低。

表21-5 中断优先级编码列表

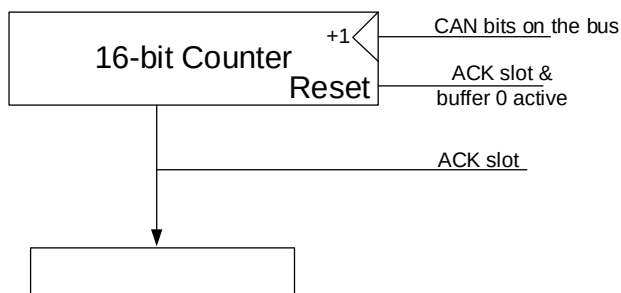
CAN interrupt Request	IRQ	IST[3]	IST[2]	IST[1]	IST[0]
No request	0	0	0	0	0
Error Interrupt	1	0	0	0	0
Buffer0	1	0	0	0	1
Buffer 1	1	0	0	1	0
Buffer2	1	0	0	1	1
Buffer3	1	0	1	0	0
Buffer4	1	0	1	0	1
Buffer5	1	0	1	1	0
Buffer6	1	0	1	1	1
Buffer7	1	1	0	0	0
Buffer8	1	1	0	0	1
Buffer9	1	1	0	1	0
Buffer10	1	1	0	1	1
Buffer11	1	1	1	0	0
Buffer12	1	1	1	0	1
Buffer13	1	1	1	1	0
Buffer14	1	1	1	1	1

21.4.9 时间戳计数器

CAN 控制器支持 16 位 TIMER 计数器，计数值通过配置 TSTP 寄存器来控制成功传输或接收数据后 ACK slot（应答间隙）的延迟时间。此计数器的作用在于同步 CAN 网络中接收或发送的时间。例如 BUFFER 的 TSTP 寄存器在应答间隙期间，捕获到现在 CTMR 的值，接着 CTMR 复位为 0。时间戳功能可配置 CGCR 寄存器的第[4]位 tstp_en 使能或关闭。



图21-12 应答间隙计数器示意图

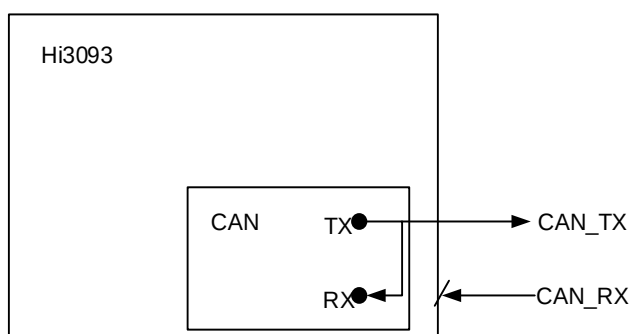


21.4.10 诊断自检测试

21.4.10.1 配置环回测试

配置 CGCR 寄存器第[10]位 diagen 为 1，进入诊断模式，配置 CGCR 寄存器[8]loopback 为 1，进入环回测试，如图 21-13 所示。

图21-13 环回测试

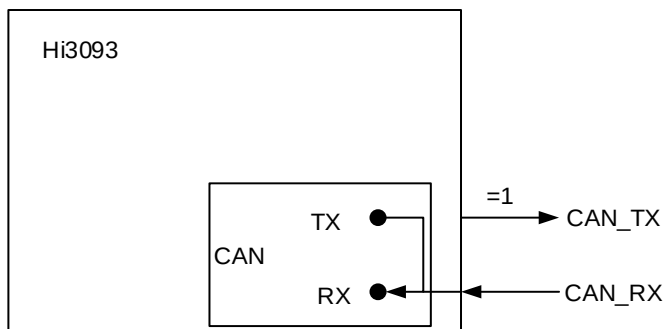


在回路模式下，控制器不接收总线 CAN_RX 信号，而接收 TX 的信号，同时 TX 信号发送到总线 CAN_TX 上。该模式用于自身测试，且忽略应答错误。

21.4.10.2 配置静音测试

配置 CGCR 寄存器第[10]位 diagen 为 1，进入诊断模式，配置 CGCR 寄存器[6]lo 为 1，进入静音测试，如下图 21-14 所示。

图21-14 静音测试

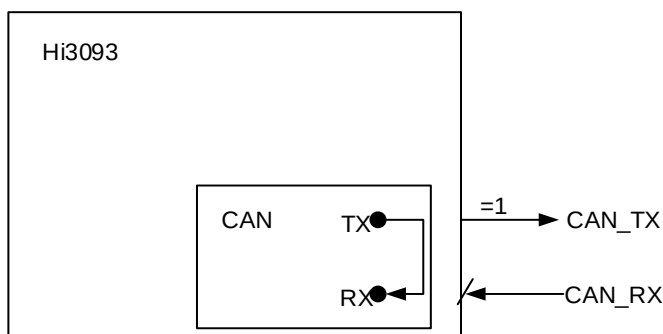


在静音模式下，CAN 可以接收其他节点发送的数据帧或者远程帧，但 CAN_TX 只能输出隐性电平，不能发送数据。TX 节点的应答信号会环回到 RX 端。静音模式用来分析总线负荷的情况，且能保证对总线不产生影响。此时需要有其他 CAN 设备反馈应答标识，否则会出现应答错误而接收不到数据报文。

21.4.10.3 配置环回静音测试

配置 CGCR 寄存器第[10]位 diagen 为 1,进入诊断模式，配置 CGCR 寄存器[8]loopback 为 1，配置 CGCR 寄存器[6]lo 为 1，进入静音环回测试，如图 21-15 所示。

图21-15 静音环回测试



静音回路模式相当于回路模式和静音模式的效果组合在一起，此模式用来自身热测试，且发送和接收均不影响 CAN 总线，也不受 CAN 总线影响。



21.5 工作方式

21.5.1 管脚复用配置

参考《Hi3093 硬件设计指南》管脚复用相关章节。

21.5.2 时钟门控

21.5.2.1 打开门控时钟

具体配置步骤如下：

- 步骤 1 读 LP_SUBSYS 寄存器 LPSC_CLKSTA_CTRL1.canbus0_clk_status，获取当前时钟门控信息。
- 步骤 2 如果 canbus0_clk_status 为 0，进入[步骤 4](#)。
- 步骤 3 如果 canbus0_clk_status 为 1，表示 CANBUS0 时钟已经打开，结束操作。
- 步骤 4 配置 LPSC_CLKEN_CTRL1.canbus0_clk_enb，为 1，打开 CANBUS0 模块的时钟门控，并返回[步骤 1](#)。

----结束

21.5.2.2 关闭门控时钟

具体配置步骤如下：

- 步骤 1 读 LP_SUBSYS 寄存器 LPSC_CLKSTA_CTRL1.canbus0_clk_status，获取当前时钟门控信息。
- 步骤 2 如果 canbus0_clk_status 为 1，进入[步骤 3](#)。
- 步骤 3 如果 canbus0_clk_status 为 0，表示 CANBUS0 时钟已经关闭，结束操作。
- 步骤 4 配置 LPSC_CLKDIS_CTRL1.canbus0_clk_dis 为 1，关闭 CANBUS0 模块的时钟门控，并返回[步骤 1](#)。

-----结束



说明

CANBUS1 软复位寄存器操作相应寄存器即可。

21.5.3 软复位

CANBUS0 软复位流程：

- 步骤 1 配置 LP_SUBSYS 的寄存器 LPSC_SRST_REQ1.canbus0_srst_req 为 1。
- 步骤 2 直到读 LP_SUBSYS 寄存器 LPSC_RESET_ST1.canbus0_srst_st，获取当前软复位状态为 1。表示软复位配置成功。
- 步骤 3 延时 2us。
- 步骤 4 配置 LP_SUBSYS 的寄存器 LPSC_SRST_DREQ1.canbus0_srst_dreq 为 1。
- 步骤 5 直到读 LP_SUBSYS 寄存器 LPSC_RESET_ST1.canbus0_srst_st，获取当前软复位状态为 0。表示软复位释放。

----结束

说明

CANBUS1 软复位寄存器操作相应寄存器即可。

21.5.4 正常工作模式

以 CanBus0 为例：

- 步骤 1 初始化 RAM，将所有寄存器 CMB0~CMB14 的所有 Memory 寄存器清零，分别对所有 CMB0~CMB14 的 CNSTAT 进行初始化配置。
- 步骤 2 配置 CTIM 寄存器配置波特率。
- 步骤 3 配置 CGCR 寄存器，选择忽视应答，数据方向，时间同步使能，BUFFLOCK 使能，发送与接收的极性配置。
- 步骤 4 配置 GMSKX 及 BMSKX 寄存器选择是否接收 ID 识别符掩码。
- 步骤 5 配置 CIEN 和 CICEN 设置中断是否屏蔽及中断请求优先级功能。
- 步骤 6 配置 CGCR 寄存器使能 CAN。
- 步骤 7 配置 CAN 的接收 BUFFER，包括 CNSTAT 状态控制寄存器，和 ID 寄存器。



步骤 8 配置 CAN 的发送 BUFFER，包括数据缓存寄存器 DATA3~0，ID1~0 寄存器和 CNSTAT 寄存器中 BUFFER 状态、数据长度和传输优先级。

步骤 9 等待发送与接收成功的中断。

----结束

21.5.5 诊断测试模式

21.5.5.1 配置 CAN 控制器自检测测试

步骤 1 初始化 RAM，将所有寄存器 CMB0~CMB14 的所有 Memory 寄存器清零，分别对所有 CMB0~CMB14 的 CNSTAT 进行初始化配置。

步骤 2 配置 CTIM 寄存器配置波特率。

步骤 3 配置 CGCR 寄存器，进行诊断模式的选择，需要给 digen，internal，loopback 写 1。

步骤 4 配置 GMSKX 及 BMSKX 寄存器选择是否接收 ID 识别符掩码，也可配置 CIEN 和 CICEN 测试中断功能。

步骤 5 配置 CGCR 寄存器使能 CAN。

步骤 6 配置 CAN 的接收 BUFFER，包括 CNSTAT 状态控制寄存器和 ID 寄存器。

步骤 7 配置 CAN 的发送 BUFFER，包括 CNSTAT 状态控制寄存器和 ID 寄存器，和 DATA3~0 数据缓存寄存器。

步骤 8 读取 CAN 接受 BUFFER 的数据寄存器 DATA3~0 是否与发送 BUFFER 中数据 DATA3~0 相同。

----结束



21.6 DFX

21.6.1 涉及寄存器基地址

表21-6 CANBUS DFX 用到寄存器基地址

寄存器模块名称	寄存器基地址	用到场景
CANBUS0	0x08741000	CAN 内部寄存器读写
CANBUS1	0x08742000	CAN 内部寄存器读写
LP_SUBSYS	0x0871D000	CAN 控制器软复位和时钟门控

21.6.2 常见配置检测

波特率配置，跟总线上的设备是否匹配。

收发极性配置，跟总线上的设备是否匹配。

21.6.3 常见 Debug 手段

请参见 [21.4.10 诊断自检测试](#) 章节，可进行环回等测试。

CAN 控制器支持错误中断的使能、上报和清除。具体参考寄存器手册。

本节仅给出 Debug 常用的状态观测寄存器：

步骤 1 CAN 节点状态查询：CSTPND.ns。

步骤 2 CAN 错误上报计数统计：CANEC.rec 或 tec。

步骤 3 CAN 错误诊断寄存器，可查询错误发生的域段和详细 bit 位，以及具体是 CRC 或填充错误；错误发生时，控制器是发送者还是接收者等信息。

----结束



表21-7 CAN 控制器状态和诊断寄存器

寄存器名称	寄存器描述	偏移地址	域段名称	BIT	域段属性	寄存器域段说明
CSTPND	CAN 中断或节点状态寄存器	0x228	reserved	31:16	RO	保留。
-			reserved	15:8	RO	保留。
			ns	7:5	RO	CAN 节点状态。 NS[2:0] : Node Status 000: Not Active; 010: Error active; 011: Error warning level; 10x: Error passive; 11x: Bus off。
			irq	4	RO	中断编码, 见 ist。 Eicen 进行使能。



寄存器名称	寄存器描述	偏移地址	域段名称	BIT	域段属性	寄存器域段说明
			ist	3:0	RO	中断编码； Eicen 进行使能； Can Interrupt: IRQ IST[3:0] no request 0 0000 Error int 1 0000 其他参考《Hi3093 寄存器手册》。
CANEC	CAN 错误计数寄存器	0x22C	reserved	31:16	RO	保留。
-			rec	15:8	RO	接收错误计数。接收错误的累计值，0~127，只描述了错误被动发生之前的错误统计状态，如果等于或者超过 128，则计数器保持前一次的数值。



寄存器名称	寄存器描述	偏移地址	域段名称	BIT	域段属性	寄存器域段说明
			tec	7:0	RO	发送错误计数。发送错误的累计值，0~255，发送计数器的值等于或者大于 128 时，设备进入错误被动；如果计数器的值大于等于 256，设备变为总线关闭。只描述了总线关闭发生之前的错误统计状态，如果等于或者超过 256，则计数器保持前一次的数值。
CEDIAG	CAN 错误诊断寄存器	0x230	reserved	31:16	RO	保留。



寄存器名称	寄存器描述	偏移地址	域段名称	BIT	域段属性	寄存器域段说明
-			drive	14	RO	驱动值 DRIVE 该 bit 给出了发发生错误时， CANTX 管脚的驱动值； 注意：接收端仅在 ACK 的时候，才会驱动总线。
			mon	13	RO	观测值 MONITOR 该 bit 给出了发生错误时， CANRX 管脚观测到的值；



寄存器名称	寄存器描述	偏移地址	域段名称	BIT	域段属性	寄存器域段说明
			crc	12	RO	CRC 错误：CRC Error - When set, this bit indicates that CRC is invalid. This bit should only be used if the EFID shows the code of the ACK field. 0：无 CRC 错误； 1：有 CRC 错误。 注意： EFID 指示为 ACK 域段是，该告警有效。
			stuff	11	RO	填充错误 Stuff Error 0：没有 bit 填充错误； 1：发生 bit 填充错误。 注意：某些域段无 bit 填充，则忽略该告警。



寄存器名称	寄存器描述	偏移地址	域段名称	BIT	域段属性	寄存器域段说明
			txe	10	RO	传输错误 Transmit Error 0: 错误发生时 CAN 控制器是接收者; 1: 错误发生时, CAN 控制器是发送者。
			ebid	9:4	RO	错误位标识符, 表明在出错帧里出错的 bit; 比如 EFID=1110b and EBID=111001b (57) ,表示错误发生在 data 域; 假设 DLC 表示 data 域为 8Byte, 则错误位置为 8x8-1 - 57=6, data 域的第 6 个 bit



	efid	3:0	RO	错误域标识符，表明最后发生的错误域段 Field EFID[3: 0] ERROR 0 ERROR DEL 1 ERROR ECHO 2 BUS IDLE 3 ACK 4 EOF 5 INTERMISSION 6 SUSPEND TRANSMISSION 7 SOF 8 ARBITRATION 9 IDE 10 EXTENDED ARBITRATION 11 R1/R0 12 DLC 13 DATA 14 CRC 15
--	------	-----	----	---



22 FAN

22.1 概述

FAN，风扇转速监控，通过检测 Fan 信号可以知道风扇的转速，FAN 信号的脉冲频率指示了风扇转速信息，脉冲频率和风扇转速是正比关系，通过测量固定时间内脉冲的个数就能知道风扇的转速。

22.2 特性

- 支持 16 个 Fan 信号输入。
- 支持 Fan 的上升沿检测或下降沿检测，可选择配置。
- 支持转速小于设定值或转速为 0 时上报异常中断，每一路输入可独立设置转速门限及中断屏蔽。
- 支持选择滤除宽度小于 125us，250us 或 500us 的脉冲。

22.3 关键术语

表22-1 FAN 关键术语描述

术语	描述
FAN	风扇。

22.4 功能

Fan，风扇转速监控模块，通过计算固定时间窗（默认 500ms）内的脉冲个数 num，根据公式 $\text{num} \times 2 \times 60 / (\text{风扇每转一圈产生脉冲个数})$ 得出风扇转速，单位为转/min。如



脉冲计数读数为 50，风扇转一圈产生两个脉冲，则当前转速为 $50 \times 2 \times 60 \div 2 = 3000$ 转/分钟。

须知

- 支持 16 个 FAN 和 12 个 PWM。
- FAN7~FAN0 和 FAN15~FAN8 使用的模块是一样的，使用的寄存器是一样的，只是地址空间不同；以 FAN7~FAN0 为例，需要使用偏移地址 0x98 的 bit[15: 0]。
- PWM5~PWM0 和 PWM11~PWM6 使用的模块是一样的，使用的寄存器是一样的，只是地址空间不同。
- FAN7~FAN0 和 PWM5~PWM0 的寄存器地址空间是 0x08759000~0x087597FF。
- FAN15~FAN8 和 PWM11~PWM6 的寄存器地址空间是 0x08759800~0x08759FFF。
- FAN7~FAN0 和 FAN15~FAN8 会分别上报中断给 GIC。软件收到中断后可以分别查询两个模块的寄存器。

22.5 工作方式

22.5.1 管脚复用配置

参考《Hi3093 寄存器手册》和《Hi3093 硬件设计指南》的相关章节。

22.5.2 读风扇转速操作

以 FAN0 为例说明，其他通道流程相同，改为操作该通道对应的寄存器即可。

步骤 1 读 FAN_SR0 寄存器的值。

步骤 2 判断 FAN_SR0 bit[8]的值，如果等于 0，则等待 0.5 秒（即前面描述的固定时间窗 500ms），跳[步骤 1](#)，否则到[步骤 33](#)。

步骤 3 用 FAN_SR0 bit[7:0]的值，通过换算得到当前风扇转速。

----结束



须知

- 测量的时间窗默认为 500ms，风扇转速值为 8bit，如果风扇每转一圈输出两个脉冲的话，最高能表示的转速为 $255 \times 60 = 15300$ 转/分钟。如果风扇的最高转速超过 15300 转/分钟，需要配置减小测量的时间窗的大小。
- 测量的时间窗通过寄存器 MEASURE_TIME bit[13:0]设置，默认值 3999，表示测量的时间窗大小为 4000 个 8kHz 时钟。该寄存器的值 N 和时间窗大小 T 的关系为： $T = (N+1) / 8$ ，单位为 ms。例如要设置为 250ms，则往 MEASURE_TIME 写入数据 1999，要设置为 400ms，则往 MEASURE_TIME 写入数据 3199。
- 修改时间窗后，转速计算公式相应发生变化。FAN_SR bit[7:0] $\times 60$ / 时间窗大小 / (风扇每转一圈产生脉冲个数) 得出风扇转速，这个公式中时间窗大小单位为“秒”，得到的风扇转速单位为“转/秒”。

22.5.3 开启转速异常中断上报

以 FAN0 为例说明，其他通道流程相同，改为操作该通道对应的寄存器即可。

步骤 1 写 FAN_TH0，设置风扇的转速下限值。

步骤 2 写 FAN_INT_EN 的 bit[24]为 1，开启风扇中断，写 FAN_INT_EN bit[0]为 1，开启通道 0 中断使能。

步骤 3 如果有中断上报，则表明转速小于设定的值，查询风扇状态寄存器 FAN_STATUS 判断是哪路风扇异常。

步骤 4 往 FAN_STATUS 相应位写 1，清中断。

----结束

须知

软件需要保证风扇已经到达平稳转动状态之后才能使能异常中断上报。发生异常之后，需要暂时关闭中断使能，等风扇再次进入正常工作状态后再开启中断使能。

22.5.4 FAN7~FAN0 的寄存器使用说明

步骤 1 FAN_CTRL 的 bit[15:0] (bit[15:14]对应 FAN7，bit[13:12]对应 FAN6，bit[11:10]对应 FAN5，bit[9: 8]对应 FAN4，bit[7:6]对应 FAN3，bit[5:4]对应 FAN2，bit[3:2]对应 FAN1，bit[1:0]对应 FAN0)。



步骤 2 FAN_STUTUS 的 bit[15:0] (bit[15:14]对应 FAN7, bit[13:12]对应 FAN6, bit[11:10]对应 FAN5, bit[9:8]对应 FAN4, bit[7:6]对应 FAN3, bit[5:4]对应 FAN2, bit[3:2]对应 FAN1, bit[1:0]对应 FAN0)。

步骤 3 FAN7~FAN0 的风扇转速寄存器和转速下限寄存器。

表22-2 FAN7~FAN0 风扇转速寄存器和转速下限寄存器列表

含义	物理地址
FAN0 的风扇转速寄存器	0x087590A4
FAN0 的风扇转速下限寄存器	0x087590A8
FAN1 的风扇转速寄存器	0x087590AC
FAN1 的风扇转速下限寄存器	0x087590B0
FAN2 的风扇转速寄存器	0x087590B4
FAN2 的风扇转速下限寄存器	0x087590B8
FAN3 的风扇转速寄存器	0x087590BC
FAN3 的风扇转速下限寄存器	0x087590C0
FAN4 的风扇转速寄存器	0x087590C4
FAN4 的风扇转速下限寄存器	0x087590C8
FAN5 的风扇转速寄存器	0x087590CC
FAN5 的风扇转速下限寄存器	0x087590D0
FAN6 的风扇转速寄存器	0x087590D4
FAN6 的风扇转速下限寄存器	0x087590D8
FAN7 的风扇转速寄存器	0x087590DC
FAN7 的风扇转速下限寄存器	0x087590E0

步骤 4 FAN_INT_EN 的 bit[15:0] (bit[15:14]对应 FAN7, bit[13:12]对应 FAN6, bit[11:10]对应 FAN5, bit[9:8]对应 FAN4, bit[7:6]对应 FAN3, bit[5:4]对应 FAN2, bit[3:2]对应 FAN1, bit[1:0]对应 FAN0) 和 bit[24] (表示 FAN7~FAN0 的中断总使能)。



----结束

须知

物理地址:0x087590E4~0x08759100 的寄存器，请不要配置和使用。

22.5.5 FAN15~FAN8 的寄存器使用说明

- 步骤 1 FAN_CTRL 的 bit[15:0] (bit[15:14]对应 FAN15, bit[13:12]对应 FAN14, bit[11:10]对应 FAN13, bit[9:8]对应 FAN12, bit[7:6]对应 FAN11, bit[5:4]对应 FAN10, bit[3:2]对应 FAN9, bit[1:0]对应 FAN8)。
- 步骤 2 FAN_STUTUS 的 bit[15:0] (bit[15:14]对应 FAN15, bit[13:12]对应 FAN14, bit[11:10]对应 FAN13, bit[9:8]对应 FAN12, bit[7:6]对应 FAN11, bit[5:4]对应 FAN10, bit[3:2]对应 FAN9, bit[1:0]对应 FAN8)。
- 步骤 3 FAN15~FAN8 的风扇转速寄存器和转速下限寄存器。

表22-3 FAN15~FAN8 风扇转速寄存器和转速下限寄存器列表

含义	物理地址
FAN8 的风扇转速寄存器	0x087598A4
FAN8 的风扇转速下限寄存器	0x087598A8
FAN9 的风扇转速寄存器	0x087598AC
FAN9 的风扇转速下限寄存器	0x087598B0
FAN10 的风扇转速寄存器	0x087598B4
FAN10 的风扇转速下限寄存器	0x087598B8
FAN11 的风扇转速寄存器	0x087598BC
FAN11 的风扇转速下限寄存器	0x087598C0
FAN12 的风扇转速寄存器	0x087598C4
FAN12 的风扇转速下限寄存器	0x087598C8
FAN13 的风扇转速寄存器	0x087598CC



含义	物理地址
FAN13 的风扇转速下限寄存器	0x087598D0
FAN14 的风扇转速寄存器	0x087598D4
FAN14 的风扇转速下限寄存器	0x087598D8
FAN15 的风扇转速寄存器	0x087598DC
FAN15 的风扇转速下限寄存器	0x087598E0

步骤 4 FAN_INT_EN 的 bit[15:0] (bit[15:14]对应 FAN15, bit[13:12]对应 FAN14, bit[11:10]对应 FAN13, bit[9:8]对应 FAN12, bit[7:6]对应 FAN11, bit[5:4]对应 FAN10, bit[3:2]对应 FAN9, bit[1:0]对应 FAN8) 和 bit[24] (表示 FAN15~FAN8 的中断总使能)。

----结束

须知

物理地址:0x087598E4~0x08759900 的寄存器，请不要配置和使用。



23 PWM

23.1 概述

PWM (Pulse Width Modulation)，为脉宽调制，是利用数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。一般应用中，PWM 的周期通常是固定的，通过占空比实时变化来承载信息；在用于风扇控制时，因为风扇在较短的时间段内是恒速的，所以占空比可以通过寄存器来配置。Hi3093 PWM 用于风扇转速控制。

23.2 特性

- 支持 12 个通道，可单独控制每个通道的开启与关闭，也可以控制 PWM 功能的使能和关闭。
- 输出为方波信号。
- 每个通道的 PWM 的周期独立可配置，范围是 1 ~ 228-1 个工作时钟周期。
- 每个通道的占空比独立可配置，范围 0%~100%。
- PWM 模块的工作时钟为 APB 时钟。
- 支持 PWM 管脚中 8 个独立管脚，剩余 4 路管脚和其他管脚复用。
- 芯片作为协处理模式复位时，要维持风扇当前转速，不能风扇失控。PWM 控制器是否复位可配置。



23.3 关键术语

表23-1 PWM 关键术语描述

术语	描述
PWM	Pulse Width Modulation，为脉宽调制，是利用数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。

23.4 功能

通过配置 PWM 周期、占空比等参数，PWM 模块产生 PWM 信号。支持 12 个 PWM 通道的信号输出。

23.5 工作方式

23.5.1 管脚复用配置

参考《Hi3093 寄存器手册》和《Hi3093 硬件设计指南》的相关章节。

23.5.2 开启或关闭 PWM 输出

以 PWM0 为例说明：

往 PWM_CHEN bit[12]写 0，关闭所有通道的输出；往 PWM_CHEN bit[12]写 1，往 PWM_CHEN bit[0]写 1，开启 PWM0 输出；往 PWM_CHEN bit[12]写 1，往 PWM_CHEN bit[1]写 1，开启 PWM1 输出，其余通道类似。

23.5.3 配置周期

以 PWM0 为例说明：

往 PWM_PRD0 bit[27:0]中写入数值 NUM，可将 PWM0 输出信号的周期改变为 NUM 个 PWM 模块的工作时钟周期，PWM 模块的工作时钟为 100MHz。如周期为 20000 (十进制)，那么 PWM_PRD0 配置为 19999 (十进制)。



其他通道类似，配置对应的 PWM_PRD 即可。

23.5.4 配置占空比

以 PWM0 为例说明：

步骤 1 配置 CH0_PWH bit[27:0]为 0；

步骤 2 配置 CH0_PWL bit[27:0]为周期 (PWM_PRD0+1) 和占空比的乘积；如周期为 20000 (PWM_PRD0=19999)，需要的占空比为 40%，则需要写入 CH0_PWL 的数为 $20000 \times 0.4 = 8000$ 。

----结束

须知

可动态改变占空比，即时生效。

23.5.5 配置复位

对于 PWM (两个 PWM 模块相同) 的复位可以进行配置控制：

步骤 1 配置 LPSC_RESET_SEL bit[0]为 0x1；可以将 PWM 的复位配置为：由协处理模式复位进行复位控制；默认情况下是协处理模式复位。

步骤 2 配置 LPSC_RESET_SEL bit[0]为 0x0；可以将 PWM 的复位配置为：由 chip 复位进行复位控制。

----结束

23.5.6 PWM5~PWM0 的寄存器使用说明

步骤 1 PWM_CHEN 的 bit[5:0] (bit[5]对应通道 PWM5，bit[4]对应通道 PWM4，bit[3]对应通道 PWM3，bit[2]对应通道 PWM2，bit[1]对应通道 PWM1，bit[0]对应通道 PWM0) 和 bit[12] (PWM5~0 通道的使能开关)。

步骤 2 PWM_INIT_VALUE 的 bit[5:0] (bit[5]对应通道 PWM5，bit[4]对应通道 PWM4，bit[3]对应通道 PWM3，bit[2]对应通道 PWM2，bit[1]对应通道 PWM1，bit[0]对应通道 PWM0)。



步骤 3 PWM5~PWM0 的周期寄存器、低电平起始寄存器、高电平起始寄存器 (物理地址:0x08759008~0x0875904C)。这几个寄存器的使用说明请参见 [23.5.4 配置占空比](#) 章节。

----结束

23.5.7 PWM11~PWM6 的寄存器使用说明

步骤 1 PWM_CHEN 的 bit[5:0] (bit[5]对应通道 PWM11, bit[4]对应通道 PWM10, bit[3]对应通道 PWM9, bit[2]对应通道 PWM8, bit[1]对应通道 PWM7, bit[0]对应通道 PWM6) 和 bit[12] (PWM11~6 通道的使能开关)。

步骤 2 PWM_INIT_VALUE 的 bit[5:0] (bit[5]对应通道 PWM11, bit[4]对应通道 PWM10, bit[3]对应通道 PWM9, bit[2]对应通道 PWM8, bit[1]对应通道 PWM7, bit[0]对应通道 PWM6)。

步骤 3 PWM11~PWM6 的周期寄存器和低电平起始寄存器、高电平起始寄存器 (物理地址:0x08759808~0x0875984C)。这几个寄存器的使用说明请参见 [23.5.4 配置占空比](#) 章节。

----结束

须知

物理地址:0x08759050~0x08759094 的寄存器, 请不要配置和使用。

物理地址:0x08759850~0x08759894 的寄存器, 请不要配置和使用。



24 VGA

24.1 概述

VGA 接口通过行场数字信号和 VDAC 模块 (Video Digital-to-Analog Converter, 图像数字模拟转换器) 模拟信号组成。

VDAC 模块在系统中用于将 GPU 输出的 RGB 数字格式信号转换为 VGA 输出的模拟信号。

24.2 特性

- 数字接口支持 10-bit 编码。
- 支持 Video 数字格式 RGB 输入, 转换为 VGA 模拟输出。
- 支持 1 ~ 300MSPS 转换速率。
- 3 路 DAC。
- 支持通道连接检测。
- 输出电压范围 0.56V~1.3V。
- 模拟电源: $1.8V \pm 10\%$ 。
- 数字电源: $0.9V \pm 10\%$ 。
- 支持 1 路本地视频输出 (VGA), 用于机架服务器前后面板物理端口。

24.3 关键术语

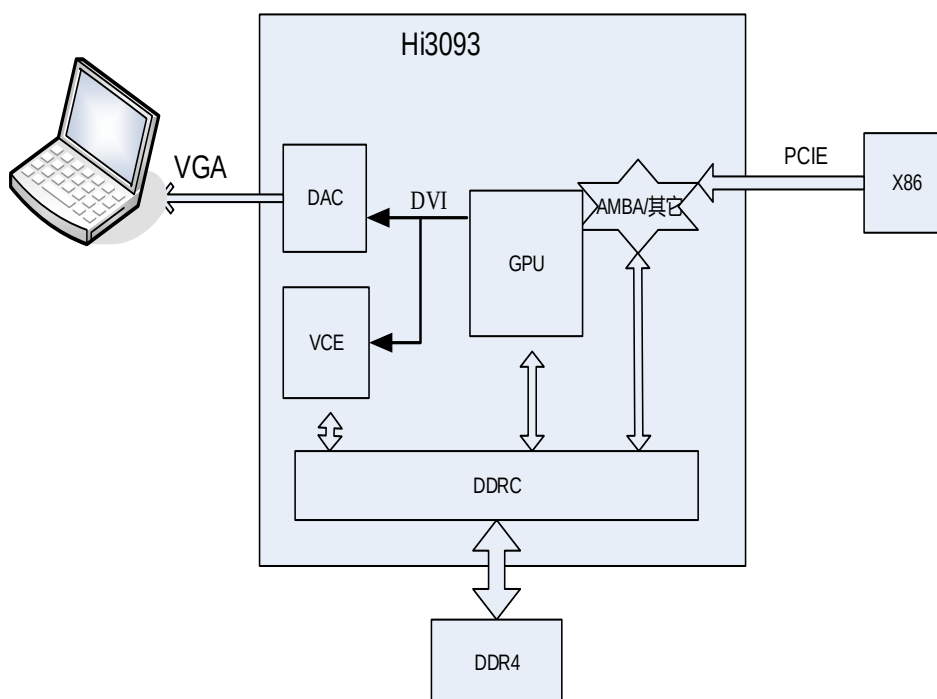
表24-1 VGA 关键术语描述

术语	描述
VGA	Video Graphics Array, 是 IBM 在 1987 年随 PS / 2 机一起推出的一种视频传输标准, 具有分辨率高、显示速率快、颜色丰富等优点, 在彩色显示器领域得到了广泛的应用。

24.4 功能

VDAC 的应用如图 24-1 所示。将 GPU 输出的 DVI 接口, RGB 数字格式信号, 转换为模拟 VGA 输出, 可以对接显示器或笔记本等。

图24-1 VDAC 应用图





24.5 工作方式

24.5.1 初始化

步骤 1 在供电电压 avdd3~0 稳定前，保持 endac3~0 为低（系统控制器寄存器）。

步骤 2 稳定后，设置 endac3~0 为高（地址 0x0874020c 的 bit2~0），使能 VDAC 的三个通道。

----结束

24.5.2 软复位

X86 系统软复位场景下，对 VDAC 子系统进行软复位；当 ARM 子系统进行软复位时，需要保证 VDAC 子系统处于正常工作状态，不受 ARM 子系统软复位影响。

通过 endac3~0 拉低，来对 VDAC 进行复位。



25 USB

25.1 概述

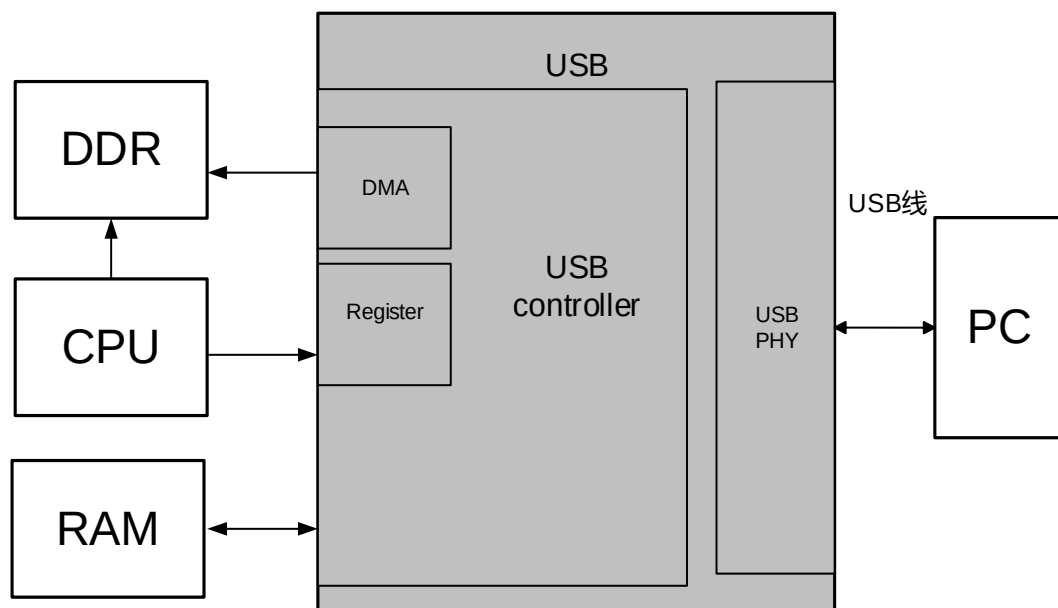
Hi3093 芯片支持 3 种类型的 USB 设备：USB 3.0 DRD，USB2.0 DRD，USB Device。DRD 设备可通过软件配置切换成为 Host 或 device 设备。

USB 3.0/2.0 的设备架构如下：

USB 系统包括了 USB 控制器、USB PHY，用于缓存数据和命令的 RAM 组成。

对外主要跟 CPU 和 DDR 进行交互，CPU 通过 AHB 总线配置 USB 控制器，USB 控制器通过 AXI 总线访问 DDR。

图25-1 USB 系统结构





25.2 特性

25.2.1 USB

DRD 模式，配置成 Host 模式：

- 支持 SuperSpeed（仅 USB3.0 控制器支持）/HighSpeed/FullSpeed/LowSpeed 四种设备类型。
- 支持 xHCI 1.0 协议。
- 支持 AMBA 配置架构。
- 支持四种传输：control、bulk，Isochronous 和 Interrupt。
- 支持内部 DMA。
- Host 模式下，通过连接 Hub 支持最多 64 个设备。
- 支持过流保护。

DRD/Device 模式配置成 Device 模式：

- 支持 SuperSpeed（仅 USB3.0 控制器支持）/HighSpeed/FullSpeed/LowSpeed 四种设备类型。
- 物理 EP 数量 18 个（9 个 IN 类型 EP；9 个 OUT 类型 EP；EP1~EP4 支持 Burst 传输，EP5~EP9 不支持 Burst 传输）。
- 支持四种传输：control、bulk，Isochronous 和 Interrupt。

25.2.2 功能

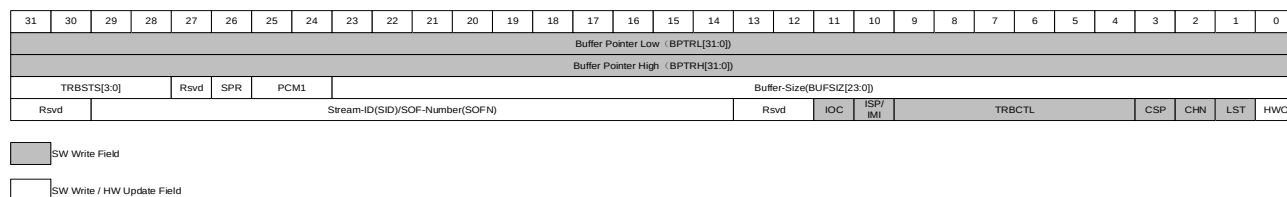
25.2.3 Host 模式功能描述

详细请参见 xHCI 协议规范。

25.2.4 Device 模式功能描述

TRB 结构的细节如图 25-2 所示。

图25-2 TRB 结构





详细域段的描述如表 25-1 所示。

表25-1 TRB 域段描述

Field	Description	Hardware Access
DW 03-00		
31:0	Buffer Pointer Low (BPTRL) 数据 buffer 指针的低 32bit。	R_W
DW 07-04		
31:0	Buffer Pointer High (BPTRH) 数据 buffer 指针的高 32bit。	R_W
DW 0B-08		
23:0	Buffer size (BUFSIZ) 当前 TRB 所指示 Buffer 剩余空间指示。	R_W
25:24	Packet Count M1 (PCM1) 在 High-Speed, 高带宽的 isochronous IN endpoint 通信过程中, 指示的是 Buffer 中总的 packet 数量减 1 的值。	R_W
26	SPR/Rsvd GUCTL1[DEV_TRB_OUT_SPR_IND]=1 时, 这个空间表明当前这个 TRB 的最后一个传输, 接收到一个短包。在 OUT 传输期间, 如果最后一个传输是一个短包, 则该 TRB 的这个位将会置 1。 GUCTL1[DEV_TRB_OUT_SPR_IND]=1 时, 该域段保留。	R_W
27	保留。	R_W



Field	Description	Hardware Access
31:28	TRB status (TRBSTS) 在释放该 TRB 之前，硬件更新该域段。 4'h0: OK; 4'h1: Missedloc:Isochronous 间隔丢失或未完成; 4'h2: SetupPending-在 control 传输的 data/status 阶段，又接收到一个 SETUP; 4'h4: TransferInProgress-传输过程中，接收到一个 end transfer 命令; 4'hF: ZLP_PENDING-只用于 NORMAL-ZLP 模式。表明 IN 传输的 zero-length-packet 仍然处于挂起状态。	R_W
DW 0F-0C		
0	Hardware Owner of Descriptor (HWO) 指示硬件获取 TRD 的控制权。 当软件创建一个 TRB 的时候就会将这个 bit 设置为 1，只有在硬件将这个 bit 修改为 0 以后，软件才能够回收这个 TRB。	R_W
1	Last TRB (LST) 表明这个 TRB 是整个 TRB 列表中最后一个 TRB。在完成传输操作以后，将会产生一个 XferComplete event。	R
2	Chain Buffer (CHN) 适用于 IN 和 OUT endpoint。 软件通过将这个 bit 设置为 1，用来指示这个 TRB 与下一个 TRB 之间的关系。	R
3	Continue on Short Packet (CSP) 适用于 OUT endpoint，且尽在接收到短包的时候有用。 如果为 1，在接收到短包的时候将会继续执行下一个传输。 isochronous endpoint 需要设置该 bit。 如果为 0，在接收到短包的时候会产生一个 XferComplete event 并且移除这个 stream。	R



Field	Description	Hardware Access
9:4	TRB Control (TRBCTL) 表明的是 TRB 类型： 1: Normal (Control-Data-2+/Bulk/Interrupt) -	R
10	Interrupt on Short Packet / Interrupt on Missed ISOC (ISP / IMI) 对于 OUT endpoint: 接收到一个短包的时候，并且 CSP = 1 且 LST = 0 的时候，如果该 bit 为 1，将会产生一个 XferInProgress event。 对于 Isochronous endpoint: 当产生 Missed Isoc 状态的时候，将产生一个 XferInProgress event。	R
11	Interrupt on Complete (IOC) 当该 bit 为 1 的时候，这个 TRB 对应的传输 buffer 的所有传输完成后，将会产生一个 XferInProgress event，并且 event status 的 IOC 位为 1。	R
13:12	保留。	R_W
29:14	Stream ID / SOF Number 如果是 stream-based bulk endpoint: 这个域段代表的是 Stream ID 如果是 isochronous endpoint: 这个域段代表的是最后一个 packet 的微帧编号	R_W
31:30	保留。	R_W

25.2.4.1 命令结构

命令结构是软件通过配置 DGCMD 和 DGCMDPAR 寄存器，来管理链路包的收发。
device 的命令结构分为：

- Device 通用命令结构。
- Device 物理 Endpoint 命令结构。



Device 通用命令结构

通用的 device 命令结构包括：

- Set Periodic Parameters
- Set Scratchpad Buffer Array Address Lo
- Set Scratchpad Buffer Array Address Hi
- Transmit Device Notification
- Selected FIFO Flush
- All FIFO Flush
- Set Endpoint NRDY
- Run SoC Bus LoopBack Test
- Restart After Disconnect

表25-2 Device 通用命令类型

Command	Description
02h	Set Periodic Parameters Parameter[9:0](SystemExitLatency)：软件应该将这个空间，配置成 host 通过 Set SEL 命令请求 device 所配置的值，单位是 ms。
04h	Set Scratchpad Buffer Array Address Lo 这个命令配置的是 scratchpad buffer 地址 bit[31:0]，这个 scratchpad 用于 save/restore。 在控制器处于 stop 模式下的时候（RunStop=0），如果发送了这个命令或 05h 命令，那么 CmdIOC 要配置为 0。
05h	Set Scratchpad Buffer Array Address Hi 这个命令配置的是 scratchpad buffer 地址 bit[63:32]，这个 scratchpad 用于 save/restore。 在控制器处于 stop 模式下的时候（RunStop=0），如果发送了这个命令或 05h 命令，那么 CmdIOC 要配置为 0。
07h	Transmit Device Notification 这个命令可以发送任何 device 类型的通知信息，通知类型和通知参数可以通过 DGCMDPAR 寄存器来指定。 DGCMDPAR[3:0]=通知类型 DGCMDPAR[31:4]=通知参数



Command	Description
09h	Selected FIFO Flush Parameter[4:0] = FIFO number Parameter[5] = 1, TX FIFO;0, RX FIFO
0Ah	All FIFO Flush 无参数。
0Ch	Set Endpoint NRDY 这个命令会令内核认为指定的 endpoint 处于 NRDY 状态。如果这个 endpoint 有可用的 buffer，那么控制器会立即发送一个 ERDY。 Parameter[4:0] = Physical Endpoint Number。
10h	Run SoC Bus LoopBack Test 使能的时候，可以让内核处于 SoC Bus Loopback test 模式，在没有任何设备与 PHY 连接的时候，发送端的数据会直接进入接收到端。 配置 Physical EP0，为一个 non-stream capable Bulk OUT endpoint，并且配置期望的 MaxPacketSize。 配置 Physical EP1，为一个 non-stream capable Bulk IN endpoint，并且将 TxFIFO 0 分配给这个 endpoint，并且配置期望的 MaxPacketSize。 发送这个命令，Parameter[0]配置为 1，使能 Loopback 模式。 发送 Start Transfer 给 EP0。 发送 Start Transfer 给 EP1。 内核从 IN buffer 读取数据，并写会给 OUT buffer。 Parameter[0] = 1，Enable Loopback 模式。 Parameter[0] = 0，Disable Loopback 模式。
11h	Restart After Disconnect 无效命令。

Device 物理 Endpoint 命令结构

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

Device 物理 endpoint 命令如下：



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

Set Endpoint Configuration:DEPCFG

这个命令用来设置物理 endpoint 的配置信息。

表25-3 DEPCFG

Field	Description
Parameter 2	
31:0	当 Parameter 0 的 Config Action = 1 的时候，这个空间设置的是 endpoint 的状态。否则，该空间保留。
Parameter 1	
31	FIFO-based 如果这个 isochronous endpoint 是一个 FIFO-based 数据流结构，需要将该 bit 配置为 1。
30	保留
29:25	USB Endpoint Number bit[29:26]:Endpoint Number bit[25]:Endpoint direction OUT IN 物理 endpoint0 (EP0) 一定要是 control endpoint 0 OUT 物理 endpoint1 (EP1) 一定要是 control endpoint 1 IN
24	Stream Capable(StrmCap) 只是这个 endpoint 是 stream-capable (MaxStreams !=0)



Field	Description
23:16	bInterval_m1 设置为 bInterval – 1 有效值是 0~13 bInterval 在 endpoint 描述符中反馈。工作在 Full-Speed 模式的时候，该 bit 配置为 0。
15	如果 endpoint 使用 External Buffer Control(EBC)模式。可用来限制 DMA 的 outstanding 数量。 保留。
14	在 bit15 = 1 的时候，才需要配置该 bit，否则，无需配置该 bit。 1:控制器不会回写 TRB 描述符（不更新 TRB 的 HWO）； 0:控制器回写 TRB 描述符（更新 TRB 的 HWO）。
13:8	Device Endpoint Specific Event Enable(DEPEVTEN) 如果某个 bit 配置为 0，那么相应的 event 就会被关闭。 Bit 13:Stream Event Enable Bit 12:Reserved Bit 11:Reserved Bit 10:XferNotReady Enable Bit 9:XferInProgress Enable Bit 8:XferComplete Enable
7:5	保留
4:0	Interrupt number(IntrNum) 必须要配置为 0。
Parameter 0	
31:30	Config Action 0：初始化 endpoint 状态； 1：重载 endpoint 状态； 2：修改 endpoint 状态。
29:26	保留



Field	Description
25:22	Burst Size (BrstSiz) 0: Burst length = 1 1: Burst length = 2, 以此类推 15: Burst length = 16
21:17	FIFO Number(FIFONum) 指示那个 TxFIFO 分配给这个 endpoint。 对于 Control endpoint, OUT endpoint 的 FIFONum 要与 IN endpoint 的相同。对于其他 OUT endpoint, 这个空间需要配置为 0。 虽然在 DRD 模式中, TxFIFO 的数量大于等于 16 个, 但是 device 模式必须要使用低 16 个 TxFIFO。
16:14	保留
13:3	Maximum Packet Size(MPS) 适用于 IN 和 OUT endpoint。
2:1	Endpoint Type(EPTypE) USB endpoint 的传输类型。 00: Control 01: Isochronous 10: Bulk 11: Interrupt
0	保留

Set Endpoint Transfer Resource Configuration:DEPXFERCFG

每个 endpoint 只能有一个传输资源。Start Transfer 会使用这个传输资源。End Transfer 或 XferComplete event 会释放这个传输资源。如果软件想要给一个 endpoint 分配多个传输资源, 那么将会导致 CmdStatus/EventStatus 产生 error。

表25-4 DEPXFERCFG

Field	Description
Parameter 2	



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

Get Endpoint State:DEPGETSTATE

在 Get Endpoint State 命令完成以后，DEPCMDPAR2 中存放的就是 endpoint 的状态信息。包括当前数据序列号，流控状态和控制传输状态（只针对 control endpoint）。

只有在控制器进入到 hibernation 状态的时候才会使用这个命令。因为本控制器不支持 hibernation 状态，所以该命令无效。

Set Stall and Clear Stall:DEPSSTALL,DEPCSTALL

只适用于 non-isochronous IN 和 OUT endpoint。

软件发送一个 Set Stall 命令来暂停 USB host 发送给这个 endpoint 的 token。如果 endpoint 处于 NRDY 状态，那么 STALL 状态抢占 NRDY 优先级。

如果当软件发送 Set Stall 命令的时候，当前刚好有业务需要处理，需要根据如下 endpoint 方向进行处理：

- 对于 OUT endpoint，当前包将会完成。控制器会在下一个 OUT DP 或 token 的时候使用 STALL 来做出响应。
- 对于 SuperSpeed 模式下的 IN endpoint，当前的突发业务会被完成，内核会在下一个 ACK TP 的时候使用 STALL 做出响应。对于其他速率，控制器会完成当前的包，并且在下一个 IN token，使用 STALL 作出响应。
- 对于 non-control endpoint，软件的职责就是通过 Set Stall/Clear Stall 命令来设置和清除 STALL。当软件清除了 STALL 的时候，endpoint 的数据序列号就会被复位成 0。
- 对于 control endpoint，软件在 OUT 方向的 endpoint 上且只有在 OUT 方向的 endpoint 上，只要发送 Set Stall 命令就可以了。当控制器接收到一个针对这个



endpoint 的 SETUP token 的时候，控制器会自动清除 STALL。软件不能在 control endpoint 上发送 Clear Stall 命令。

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

Start Transfer:DEPSTRTXFER

这个命令适用于 IN 和 OUT endpoint。

Device 模式的传输描述符 (TD) 是一个 TRB 链表，这个链表实现了若干个传输操作。软件通过 Start Transfer 命令来提供指向这个 TD 的指针，并且这个指针会在 XferComplete event 产生的时候或 End Transfer 命令产生的时候自动移除。

软件通过发送这个命令来通知控制器，已经在内存中构件号描述符。对于 IN endpoint，这个命令会触发控制器执行这个描述符，并且将数据从相应的 memory 空间搬到内部的 TxFIFO。对于 OUT endpoint，会触发控制器开始执行这个描述符，并且将接收到 RxFIFO 中的数据搬到相应的 memory 空间。

为了响应 Start Transfer 命令，硬件会为此传输分配一个资源索引编号 (XferRscldx)，并且将这个索引值反馈到 DEPCMDn 寄存器，以及 Command Complete event 中。这个 index 一定要用于后续的 Update 和 End Transfer 命令。

如果相同的 TD 已经存在与控制器中的 cache 中，那么针对这个 TD 发送 Start Transfer 命令也是合法的。

non-stream capable endpoint 规则：

- Stream ID 必须要为 0
- HighPri 空间保留

Stream-capable endpoint 规则：

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

表25-5 DEPSTRTXFER

Field	Description
Parameter 2	
31:0	保留



Field	Description
Parameter 1	
31:0	Transfer Descriptor Address(TDAddr Low)
Parameter 0	
31:0	Transfer Descriptor Address(TDAddr High)

Update Transfer:DEPUPDXFER

如果软件使用的是一个环形的 TRB buffer，并且更新一个 TRB 的 HWO =1，因为这个 TRB 的 HWO = 0，那么软件就必须执行 Update Transfer 命令。在 DEPCMD 中给出这个 TRB 的资源索引。硬件会使用这个信息来重新缓存这个 TRB。

软件可能会通过将 CmdAct=0，且 CmdIOC=0 的方式来发送一个 “No Response Update Transfer” 命令。这种情况下，硬件不会产生 Command Complete event，也不会将 CmdAct 设置为 0（因为这一 bit 已经为 0），并且软件可能会在这个命令以后，向相同的 endpoint 发送其他命令。当软件根据 XferNotReady event 来构建 TRB 的时候，可能不会使用这个 Update Transfer 命令。

End Transfer:DEPENDXFER

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

发送一个 End Transfer 命令以后，软件一定要将 CmdIOC 设置为 1，这样一来 Endpoint Command Complete event 就会在传输完成的时候产生。

对于 IN endpoint，这个命令可以用来停止描述符的执行操作，并且内核会停止预取针对这个 endpoint 新的数据，并且停止传输操作。内核可能会使用 DPPABORT 序列集来截取一个当前正在传输的包，而无需等待任何 ACK，并且无需更新 TRB 状态。

对于 OUT endpoint，这个命令可以用来停止描述符的执行操作。如果 FIFO 中有数据，那么接收 FIFO 中的数据会被搬到相应的 memory 空间。内核不会更新 TRB 状态。

在如下条件下使用这个命令：

- 当处理 USBReset 或 SetConfiguration 的时候，可以使用这个命令来关闭 endpoint。
- 在接收到一个 ClearFeature (STALL) 控制传输的时候，软件可以在 Clear Stall 命令后发送 End Transfer 命令，并且随后发送 Start Transfer 命令。



- 在 XferInProgress event 后，当一个 TRB 的 HWO 设置为 0 的时候，可以使用这个命令。
- 对于 isochronous endpoint，如果 host 在几个 interval 内都没有传输数据，软件可能会强制传输结束，并且等待 host 重新发起传输。

在 End Transfer 的时候，硬件不会发送 XferComplete event，但会发送 CommandComplete event。

表25-6 DEPENDXFER

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

Start New Configuration:DEPSTARTCFG

软件在如下情况下发送该命令：

- power-on-reset 后，XferRscldx=0，开始配置 endpoint 0 和 1 之前。
CmdIOC 必须要配置为 0，并且软件要轮询的方式查询 CmdAct，以便确定命令完成的时刻。
- XferRscldx = 2，当接收到 SetConfiguration 的时候，且在开始配置 Endpoint > 1 之前。

须知

CmdIOC 可能会配置为 0，也可能会配置为 1。

这个命令只能发送给 Endpoint0。

在接收到这个命令的时候，硬件会将传输资源复位成 XferRscldx 参数（一定要是 0 或 2）。



Device Event Buffer 结构

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

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

表25-7 Device Endpoint-n Events

Field	Description
31:16	Event Parameters(EventParam) Bulk Endpoint 上的 XferNotReady,XferComplete,和 Stream event: bit[31:16]:Stream ID. XferInProgress: bit[31:16]:Isochronous Microframe Number. Isochronous Endpoint 上的 XferNotReady event: bit[31:16]:Isochronous Microframe Number. EPCmdCmpl event: 对于所有的 EPCmdCmpl event bit[27:24]:Command Type 对于 Start Transfer 命令的 EPCmdCmpl event bit[22:16]:Transfer Resource Index



Field	Description
15:12	<p>Event Status(EventStatus)</p> <p>在 XferNotReady event 中:</p> <p>bit[15]:表明 XferNotReady event 产生的原因:</p> <p>0:XferNotActive</p> <p>1:XferActive</p> <p>bit[14]:Not Used</p> <p>bit[13:12]:对于控制 endpoint, 指示请求阶段:</p> <p>01:Control Data Request</p> <p>10:Control Status Request</p> <p>在 XferComplete 或 XferInProgress event 中:</p> <p>bit[15]:完成 TRB 的 LST 位</p> <p>bit[15]:MissedIsoc</p> <p>bit[14]:TRB 的 IOC 位</p> <p>bit[13]:只是 TRB 的完成, 是因为接收到一个短包, 或者是 isochronous interval 中最后一个包</p> <p>bit[12]:保留</p> <p>在 Stream Event 中:</p> <p>bit[15:12]:</p> <p>4'h2:StreamNotFound</p> <p>4'h1:StreamFound</p> <p>在响应 Start Transfer 命令的时候:</p> <p>bit[15:12]:</p> <p>4'h2:表明 Start Transfer 命令的总线时间已经溢出</p> <p>4'h1:表明没有有效的传输资源适用于这个 endpoint</p> <p>在响应 Set Transfer Resource 命令的时候:</p> <p>bit[15:12]:</p> <p>4'h1:表明有错误产生, 因为软件请求了多个传输资源</p> <p>在响应 End Transfer 命令的时候:</p> <p>bit[15:12]:</p> <p>4'h1:表明分配了无效的传输资源</p>
11:10	保留
9:6	4'h7: Endpoint Command Complete



Field	Description
9:6	4'h6: Stream Event
9:6	4'h5: 保留
9:6	4'h4: 保留
9:6	4'h3: XferNotReady Event
9:6	4'h2: XferInProgress Event
9:6	4'h1: XferComplete Event
9:6	4'h0: 保留
5:1	Physical Endpoint Number(0~31)
0	1'b0: 表明这个是一个 endpoint-specific event。

表25-8 Device-Specific Event

Field	Description
31:25	保留
24:16	Event Information Bits(EvtInfo) 对于一个 USB/Link 状态改变 Event，这个域段表明的是链路状态： EvtInfo[8:5]-HIRD 值，来自 LPM token EvtInfo[4]-SuperSpeed event。1, SS; 0, non-SS EvtInfo[3:0]-Link State
15:13	保留
12:8	16: ECC Error
	15: Reserved
	14: L1 Resume/RemoteWake Event
	13: Stopped On Disconnect
	12: Vendor Device Test LMP Received Event
	11: Event Buffer Overflow Event



Field	Description
	10: Generic Command Complete Event
	9: Erratic Error Event
	8: L1 Suspend Event
	7: Start of Frame
	6: USB Suspend Entry Event
	5: Hibernation Request Event
	4: Resume/Remote Wakeup Detected Event
	3: USB/Link State Change
	2: Connection Done
	1: USB Reset
	0: Disconnect Detected Event
7:1	7'h00: Device Specific Event
0	1'b1: Non-Endpoint-Specific Event

25.3 工作方式

25.3.1 时钟门控

上电时 USB 时钟默认为打开，不用时可以关闭时钟以降低芯片功耗，其关断和打开步骤如下：

关断时钟步骤如下：

- 步骤 1 配置 HP 子系统寄存器 HP_SUB_CSR.HP_CLK_DIS.usb0_clk_dis 为 1，关断提供给 USB0 DEV 控制器的总线时钟。
- 步骤 2 查询 HP 子系统寄存器 HP_SUB_CSR.HP_CLK_CFG.usb0_clk_cfg，如果为 0，则关断总线时钟成功，跳出；否则，回到步骤 1。



----结束

打开时钟步骤如下：

- 步骤 1 配置 HP 子系统寄存器 HP_SUB_CSR.HP_CLK_ENB.usb0_clk_en 为 0，打开提供给 USB0 DEV 控制器的总线时钟。
- 步骤 2 查询 HP 子系统寄存器 HP_SUB_CSR.HP_CLK_CFG.usb0_clk_cfg，如果为 1，则打开总线时钟成功，跳出；否则，回到[步骤 1](#)。

----结束

其他控制器参考相同寄存器的其他域段；其中 USB3 DRD 为 HP_CLK_ENB.
Usb3_sysclk_en 和 HP_CLK_DIS. Usb3_sysclk_dis。

25.3.2 USB 控制器软复位

控制器的软复位一般在初始化流程使用，通过配置系统控制器，可实现对 USB 控制器的软复位：

USB 控制器的软复位：

- 步骤 1 配置寄存器 HP_SUB_CSR.HP_SRST_REQ2.usb3_srst_req 为 1（USB3 DRD），复位 USB 控制器。
- 步骤 2 查询寄存器 HP_SUB_CSR.HP_SRST_ST2.usb3_srst_st 如果为 1，表示复位成功；否则回到[步骤 1](#)。

----结束

解除 USB 控制器的软复位：

- 步骤 1 配置寄存器 HP_SUB_CSR. HP_SRST_DIS2. usb3_srst_dis 为 1（USB3 DRD），解除复位 USB 控制器。
- 步骤 2 查询寄存器 HP_SUB_CSR. HP_SRST_ST2. usb3_srst_st 如果为 1，表示解复位成功；否则回到[步骤 1](#)。

----结束

其他控制器的寄存器参考相同寄存器的其他域段。



25.3.3 USB Misc 模块的软复位

控制器的软复位一般在初始化流程使用，通过配置系统控制器，可实现对 USB 控制器的软复位：

复位 USB Misc：

- 步骤 1 配置寄存器 HP_SUB_CSR.HP_SRST_REQ2.usb3_apb_rst_rst_req 为 1（USB3 DRD），复位 USB MISC 模块。
- 步骤 2 查询寄存器 HP_SUB_CSR.HP_SRST_ST2.usb3_apb_rst_rst_st 如果为 1，表示复位成功；否则回到[步骤 1](#)。

----结束

解除复位 USB Misc：

- 步骤 1 配置寄存器 HP_SUB_CSR.HP_SRST_DIS2.usb3_apb_rst_rst_dis 为 1（USB3 DRD），解复位 USB 控制器。
- 步骤 2 查询寄存器 HP_SUB_CSR.HP_SRST_ST2.usb3_apb_rst_rst_st 如果为 1，表示解复位成功；否则回到[步骤 1](#)。

----结束

其他控制器的寄存器参考相同寄存器的其他域段。

25.3.4 USB PHY 全局软复位

PHY 的软复位一般在初始化流程或者结合控制器进行复位时使用，通过配置系统控制器，可实现对 USB PHY 的软复位：

复位 USB PHY：

- 步骤 1 配置寄存器 HP_SUB_CSR.HP_SRST_REQ2.usb3_cmb_phy_rst_req 为 1（USB3 ComboPhy），复位 USB MISC 模块。
- 步骤 2 查询寄存器 HP_SUB_CSR.HP_SRST_ST2.usb3_cmb_phy_rst_st 如果为 1，表示复位成功；否则回到[步骤 1](#)。

----结束

解除复位 USB PHY：



- 步骤 1 配置寄存器 HP_SUB_CSR.HP_SRST_DIS2.usb3_cmb_phy_srst_dis 为 1 (USB3 ComboPhy) , 解复位 USB 控制器。
- 步骤 2 查询寄存器 HP_SUB_CSR.HP_SRST_ST2.usb3_cmb_phy_srst_st 如果为 1, 表示解复位成功; 否则回到[步骤 1](#)。

----结束

须知

USB3 控制器连接有 2 个 phy: ComboPhy (PIPE 端口) 和 femtoPhy (UTMI 端口)。其他 USB 控制器均为对接 femtoPhy。

其他控制器的控制器, 参考同寄存器的不同域段, 其中 USB3 的 femtophy 寄存器为 HP_SUB_CSR.HP_SRST_ST2.usb_phy_srst_st。

25.3.5 USB PHY 端口软复位

PHY 的软复位一般在初始化流程或者结合控制器进行复位时使用, 通过配置系统控制器, 可实现对 USB PHY 的软复位:

- 步骤 1 配置寄存器 USB3_GLB.PERI_USB3_GUSB2PHYCFG.N.u2_phy_soft_rst 为 1, 启动对 USB2 的 PHY UTMI 端口复位。

配置寄存器 USB3_GLB.PERI_USB3_GUSB3PIPECTLN.u3_phy_soft_rst 为 1, 启动对 USB3 的 PHY PIPE 端口复位。

- 步骤 2 配置寄存器 USB3_GLB/PERI_USB3_GUSB2PHYCFG.N/u2_phy_soft_rst 为 0 释放对 USB2 的 PHY UTMI 端口复位。

配置寄存器 USB3_GLB/PERI_USB3_GUSB3PIPECTLN/u3_phy_soft_rst 为 0 释放对 USB3 的 PHY PIPE 端口复位。

USB2 的 Device0 1 和 DRD 不涉及 PIPE 端口。

----结束

25.3.6 USB2.0/3.0 初始化流程

初始化主要用在芯片上电启动之后, 其包括复位与撤消复位两个过程, 复位的模块包含 PHY、Host 控制器、Device 控制器 0 ~ 3。



PHY 的全局复位要求时间最小 20us，因此上电复位步骤如下：

步骤 1 控制 USB Misc、USB 控制器、USB PHY 复位使能。

请参考 [25.3.2 USB 控制器软复位~25.3.3 USB Misc 模块的软复位](#) 章节。

步骤 2 释放 USB Misc 复位。

请参考 [25.3.3 USB Misc 模块的软复位](#) 章节。

步骤 3 通过 Misc 寄存器配置不使能 PHY 的模式选择：

如有不使用的 USB 端口，需隔离 PHY 的输出信号，并且将控制器给 PHY 的信号固定为 0，直接将 PHY 的电源口悬空；

不用的 PHY 的模式选择：配置寄存器 `usb2_phy_ctrl0 / usb3_phy_ctrl0` 为 0，选择 PHY 输入为 MISC 模块；

使用的 PHY 的模式选择：配置寄存器 `usb2_phy_ctrl0 / usb3_phy_ctrl0` 为 0，选择 PHY 输入为 Contrler。

步骤 4 通过 Misc 寄存器配置 VBUS 过流控制和极性选择：

根据单板上 VBUS 控制电路的极性，选择过流信号的极性控制和使能：

`USB3_MISC_USB3_CTRL/usb3_ctrl_ctrl2[25:24]` (0：低有效；1：高有效)。

使能对应端口的极性控制电路，将 `USB3_MISC_USB3_CTRL/usb3_ctrl_ctrl2[23:22]` 对应 bit 配置为 1。

步骤 5 通过 Misc 寄存器配置虚拟 VBUS 选择：

未连接 VBUS 信号，使用 MISC 寄存器来虚拟 Vbus 输入的行为：

配置 `USB3_MISC_USB2_PHY/usb2_phy_ctrl4[13]` 选择 Vbus 输入检测方式为虚拟 VBUS 控制；

将 `USB3_MISC_USB2_PHY/usb2_phy_ctrl1[10]` 配置为 1。

步骤 6 通过 Misc 寄存器配置 PHYVregbypass：

选择 $VDDH = 1.8\text{ V}$ ，配置 `USB3_MISC_USB2_PHY/usb2_phy_ctrl4/Vregbypass` 寄存器为 1。

步骤 7 配置 serdes 的 SI 参数（如果默认参数，需调整情况下，执行此步骤，具体请参考 [39 SerDes](#) 章节）。

步骤 8 释放 USB PHY 的全局复位。



请参考 25.3.4 USB PHY 全局软复位章节。

步骤 9 释放 USB 控制器复位；

请参考 25.3.2 USB 控制器软复位章节。

步骤 10 释放 USB PHY 端口的复位。

请参考 25.3.5 USB PHY 端口软复位章节。

步骤 11 RAM 初始化

(Mem 功能提供的初始化功能。在上电/USB 控制器重新复位、出现 ECC 错误的时候，需软件配置)

1. RAM0 初始化使能：usb3_ctrl_ctrl3[1] = 1；
2. RAM0 初始化完成：当 usb3_ctrl_status8[27]=1，进行下一步 (=1，表明初始化完成；=0，表明当前正处在初始化过程)；
3. RAM0 初始化去使能：usb3_ctrl_ctrl3[1] = 0。

步骤 12 Cache 属性配置

USB 控制器的 DMA 接口，其 Cache 属性可配置。对于 AXI 接口，分为写通路的 cache 属性：awcache；读通路的 cache 属性：arcache。

1. cache 控制来源的选择：

配置 usb3_ctrl_ctrl2/ xm_arcache_sel 和 usb3_ctrl_ctrl2/ xm_awrcache_sel：

0：通过配置 USB GLOBAL 寄存器来配置 cache 属性；

1：通过配置 MISC 寄存器来配置 cache 属性。

2. cache 属性的配置：

在 USB GLOBAL 寄存器的配置方法：

USB3_GLB/PERI_USB3_GSBUSCFG0/0xC100 寄存器来配置 cache 属性。

在 MISC 寄存器的配置方法：

通过 usb3_ctrl_ctrl2/xm_arcache 和 usb3_ctrl_ctrl2/ xm_awcache 来配置 cache 属性。

步骤 13 DMA 接口 PROT 属性配置：

USB 控制器的 DMA 接口，保密属性:prot 是固定的 3'b010。



可通过寄存器配置修改读通道的保密属性 arprot 和写通道的保密属性 awprot。类似 Cache 属性的配置。

1. prot 控制来源的选择，通过配置 usb3_ctrl_ctrl2/xm_arprot_sel 和 usb3_ctrl_ctrl2/xm_awprot_sel 来实现：
 - 0：通过 USB GLOBAL 寄存器控制 prot 属性。
 - 1：通过配置 MISC 寄存器来配置 prot 属性。
2. prot 属性的配置：
 - USB GLOBAL 寄存器：选择通过 USB 控制器本身来控制 prot 属性
 - MISC 寄存器：usb3_ctrl_ctrl2/xm_arprot 和 usb3_ctrl_ctrl2/xm_awprot。

----结束



26 HOSTRTC

26.1 概述

HOSTRTC 利用外部输出晶振时钟，提供计时功能。

26.2 特性

- 支持一个 64bit 的计数器，HOST 可通过 PCIE 接口访问。
- HOST 通过 PCIE 接口可设置计数器的值。
- 支持典型值 8kHz，2MHz，10MHz 等时钟输入。
- 芯片直接对输入时钟进行计数。

26.3 关键术语

表26-1 HOSTRTC 关键术语描述

术语	描述
-	-

26.4 功能

用 64bit 计数器对输入晶振时钟计数，供 HOST 读取，HOST 可对计数器的值进行实时设置。



HOSTRTC 支持 HOST 通过 PCIE 接口进行访问。

HOST 访问 HOSTRTC 地址由 HOST 配置 PCIE 寄存器决定，但需与 HOST SUBSYS 寄存器 HOSTRTC_ADDR_CFG 保持一致。该寄存器由 BMC 配置。

26.5 工作方式

26.5.1 初始化

上电初始化 64bit 计数器的值，否则从 0 开始计数。

步骤 1 配置计数器低 32 位 HOSTRTC_LTIME0 bit[31:0]初始值。

步骤 2 配置计数器高 32 位 HOSTRTC_LTIME1 bit[31:0]初始值。

----结束

须知

- 一次写操作包括写低 32 位和写高 32 位，中间不允许有读操作。
- 以上两个寄存器的写操作都完成之后，数据才被加载到计数器中。
- 只能 HOST 通过 PCIE 进行操作。

26.5.2 读 HOSTRTC 操作

步骤 1 读取计数器低 32 位 HOSTRTC_LTIME0 bit[31:0]。

步骤 2 读取计数器高 32 位 HOSTRTC_LTIME1 bit[31:0]。

----结束

须知

- 一次读操作包括读低 32 位和读高 32 位，中间不允许有写操作。



- 在读低 32 位时，高 32 位被保存备份，读高 32 位时再送出去，所以读到的数值为第一次读操作时的计数值。
-



27 BMCRTC

27.1 概述

BMCRTC 为 Hi3093 利用内部时钟做的计数器。

27.2 特性

- 支持秒级上报。
- 支持补偿/延迟参数可配。

27.3 关键术语

表27-1 BMCRTC 关键术语描述

术语	描述
-	-

27.4 功能

利用芯片内部 APB 时钟计数器计算至 1s 时，进位至秒值寄存器进行计数。APB 时钟固定为 100MHz。



27.5 工作方式

27.5.1 初始化

步骤 1 配置秒值寄存器 SEC bit[31:0]。

步骤 2 配置 CNT_1S bit[26:0]寄存器，该计数器根据工作时钟计数到 1 秒时产生进位信号并清零，SEC 寄存器加 1。

步骤 3 配置使能寄存器 RTC_EN bit[0]为 1'b1 开启 BMCRTC 功能；默认为关闭功能。

----结束

27.5.2 读 BMCRTC 操作

读取秒值寄存器 SEC bit[31:0]。



28 GPU

28.1 概述

GPU (Graphic Processing Unit) 图形处理器。

GPU 模块在系统中用于本地图形 UI 显示，GPU 支持芯片内部 DVI 接口输出，通过 DAC 转换，支持 VGA 输出。

28.2 特性

- 支持以下标准分辨率：
640*480、800*600、1024*768、1152*864、1280*720、1280*768、
1280*960、1280*1024、1440*900、1600*900、1600*1200、1920*1080、
1920*1200
- 支持标准 VG/VESA。
- 支持刷新率 60Hz。
- video modes 支持的每像素 bit 数 16/24 可配置。
- 支持 2D 加速。
- 支持标准 AMBA 接口访问 DDR4。
- 支持鼠标同步。
- 支持芯片内部标准 DVI 接口，支持 (RGB) 格式数据。
- GPU 所有寄存器信息 Hi3093 CPU 可读，支持图层像素数据 (鼠标形状) 和鼠标位置数据可读取。
- 芯片内部集成一个 GPU。



28.3 关键术语

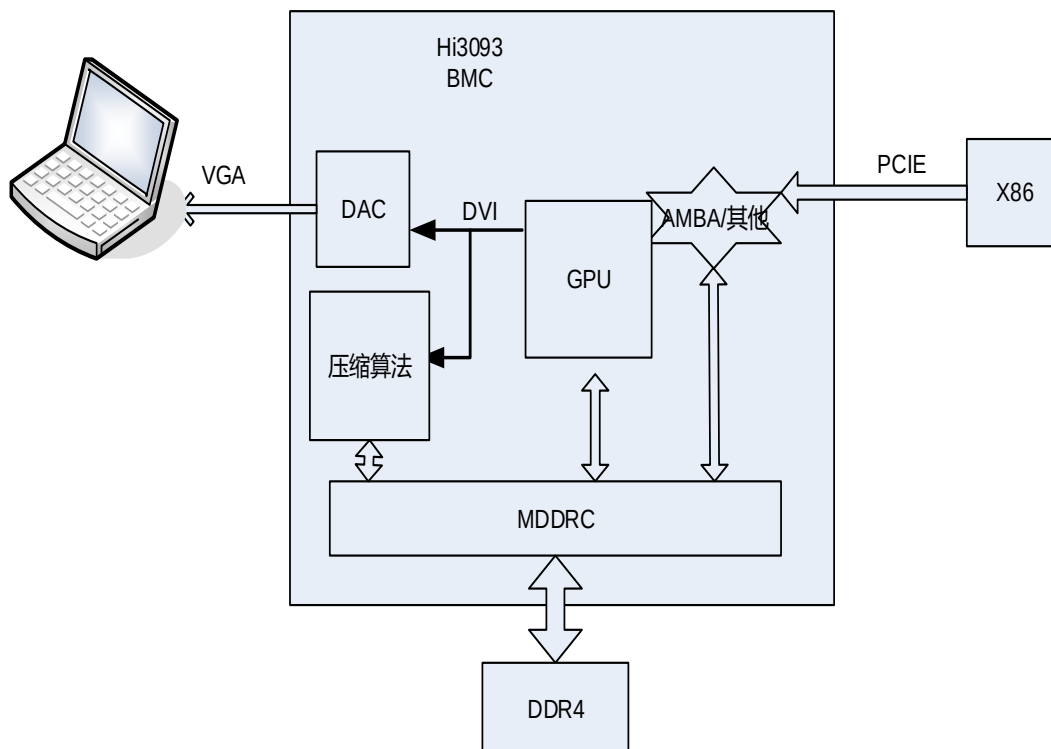
表28-1 GPU 关键术语描述

术语	描述
GPU	Graphic Processing Unit, 图形处理器。
DVI	Digital Video Interactive, 数字视频接口。

28.4 功能

GPU 的应用如图 28-1 所示。X86 通过 PCIE 接口配置 GPU 寄存器，完成对 GPU 的设置，并将需要显示的图像信息传递给 GPU。GPU 获取到绘图命令或图像数据后，完成数据处理，并通过 AMBA 接口把显示数据存储到内存 DDR，并通过 Display 模块，将数据通过 DVI 接口送出给 DAC 和图像压缩模块。

图28-1 GPU 应用框图





28.5 工作方式

28.5.1 时钟门控

上电时 GPU 控制器工作时钟默认为打开状态，GPU 时钟分为两部分，一为 GPU 总线时钟，二为 GPU DVI 时钟，这两个时钟都可以单独控制。下面以 GPU 总线时钟为例说明打开关闭方法。

28.5.1.1 打开门控时钟

步骤 1 读系统控制器寄存器 HOST_CLK_STATE bit[2]，获取当前时钟门控信息。

步骤 2 若 HOST_CLK_STATE bit[2]为 1'b0，则进入[步骤 3](#)；HOST_CLK_STATE bit[2]为 1'b1，则表示 GPU 控制器时钟已经打开，直接跳过下面的步骤，结束操作。

步骤 3 配置 HOST_CLK_ENABLE bit[2]为 1'b1，即打开 GPU 控制器的时钟门控。

----结束

28.5.1.2 关闭门控时钟

步骤 1 读系统控制器寄存器 HOST_CLK_STATE bit[2]，获取当前时钟门控信息。

步骤 2 若 HOST_CLK_STATE bit[2]为 1'b1，则进入[步骤 3](#)；HOST_CLK_STATE bit[2]为 1'b0，则表示 GPU 控制器时钟已经关闭，直接跳过下面的步骤，结束操作。

步骤 3 配置 HOST_CLK_DISABLE bit[2]为 1'b1，即关闭 GPU 控制器的时钟门控。

----结束

28.5.2 软复位

在正常工作下不要对 GPU 进行软复位，否则显示功能不可用，通过配置系统控制器 HOST_RST_CFG bit[3]对 GPU 模块进行软复位操作。

步骤 1 通过配置 HOST_RST_CFG bit[3]为 1'b1，可实现对 GPU 工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置配置 HOST_RST_CFG bit[3]为 1'b0，可实现对 GPU 工作逻辑的单独软复位撤离。

----结束



28.5.3 初始化

GPU 正常工作，需要配置 GPU 读写 DDR 的基地址和大小。

步骤 1 配置 HOST_SUBYSS_REG 的 GPU_CFG 寄存器 bit[15:8]为 GPU 访问 DDR 的地址高位，也即 bit[33:26]。

步骤 2 配置 HOST_SUBYSS_REG 的 GPU_CFG 寄存器 bit[1:0]为 2'b01，配置访问 DDR 的大小为 32MB。具体大小由软件指定。

----结束

须知

GPU 给 HOST 发送的中断，必须通过本地 CPU 处理，配置 PCIE0 相关寄存器触发 INTx 或者 MSI 中断上报给 HOST。配置 PCIE0 发送 INTx 或者 MSI 中断的流程请参见 [20.4.17 MSI 中断寄存器配置发送流程](#) 章节。



29 VCE

29.1 概述

29.2 概述

VCE 是指在视频数据传输过程中，为了在 Ethernet 上传输时获得更小的传输带宽，而使用对原始视频数据进行压缩的技术。

29.3 特性

- 支持 400M 和 200M 可配置，默认 400M。
- 支持的压缩帧率 30 帧每秒@60Hz，即 GPU 刷新频率为 60Hz 的情况下，对帧数进行下采样，压缩频率是 30 帧每秒。
- 支持两种压缩模式，模式一是支持变长游程编码压缩，模式二是支持定长游程和 JPEG 压缩。
- 压缩模式一显示用 8bit，模式二显示用 24bit。
- 录屏空间可配置 8MB、16MB、24MB 和 32MB。
- 支持录屏触发后再录空间可配置。
- 模式一在录屏打开的情况下支持黑屏检测，设置中断上报。
- 支持软件截屏。
- 提供管脚截屏，可屏蔽。
- 压缩模式一支持以下特性：



- 支持游程变长 RLE 编码。
- 支持 RGB 格式输入。
- 支持压缩数据输出格式是 RGB332。
- 支持分辨率和刷新频率为 800*600@60Hz、1024*768@60Hz、1152*864@75Hz、1280*720@60Hz、1280*768@60Hz、1280*960@60Hz、1280*1024@60Hz 的标准分辨率；VGA 模式下 640*480@60Hz 和 720*400@85Hz。
- 压缩模式二支持以下特性：
 - VCE.400：支持分辨率和刷新频率为 800*600@60Hz、1024*768@60Hz、1152*864@75Hz、1280*720@60Hz、1280*768@60Hz、1280*960@60Hz、1280*1024@60Hz、1440*900@60Hz、1600*900@60Hz、1600*1200@60Hz、1920*1080@60Hz、1920*1200@60Hz；VGA 模式下 640*480@60Hz 和 720*400@85Hz。
 - 支持 2 种视频压缩接口模式，寄存器可配置；一种是 DVI 接口模式，一种是 Frame Buffer 接口模式；两种模式可配置按照分辨率自动切换，大于某分辨率用 Frame Buffer 接口模式，反之用 DVI 接口模式，该分辨率可配置。
 - 支持 YCbCr444 压缩数据格式。
 - 可配置把 60 帧每秒的输入帧率压缩成每秒 10 帧、15 帧、20 帧或 30 帧。
 - 目标压缩帧率：YCbCr444 数据格式 1920*1200@60Hz 下支持 30 帧每秒的压缩。
 - 支持自动分区，根据分区内像素种类决定采用 RLE 或 JPEG 算法压缩，像素种类大于等于 n 种采用 JPEG 压缩，否则使用 RLE 无损压缩，其中 n 可配置 1、2、3、4 和 5。
 - 支持对图像自动分块处理，块大小 64x64。
 - 支持 CRC 计算的帧间压缩。
- VCE 模块支持单图片注入以及压缩后的图片导出。

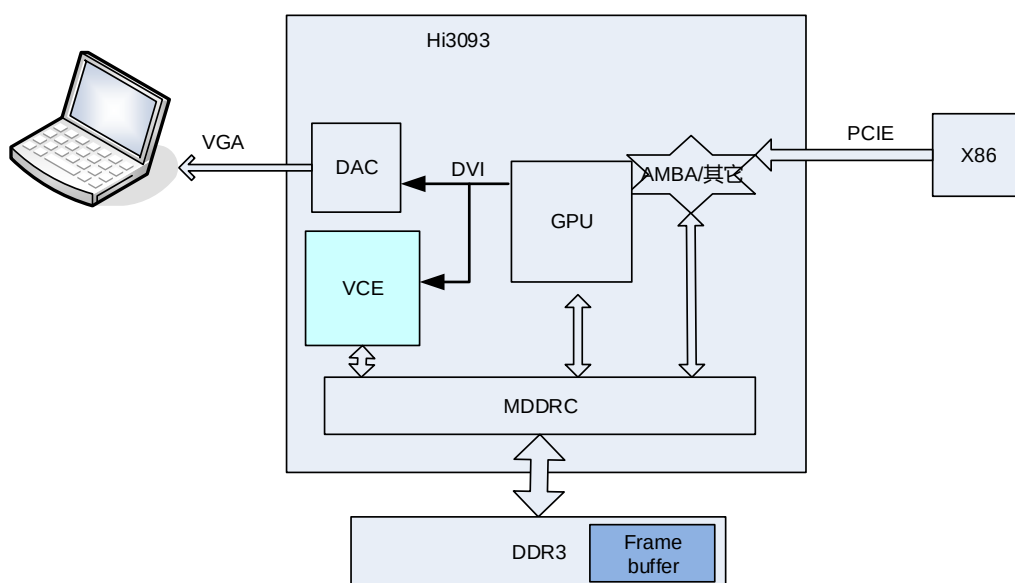


29.4 功能

29.4.1 应用框图

VCE 的应用如图 29-1 所示。GPU 把显示数据从 Frame buffer 读出，通过 DVI 口输出给显示器。VCE 可以从 DVI 口或者从 DDR 中的 Frame buffer 区域取数据，然后压缩。

图29-1 VCE 应用图



29.4.2 相关模块

在系统中，VCE 模块会和下面多个模块进行交互，分别描述如下：

- GPU：GPU 通过 DVI 接口将视频图像数据和 DVI 时序传给压缩模块，通过 Frame buffer 信息接口把 Frame buffer 格式、地址、分辨率和鼠标信息传给压缩模块。
- CPU：CPU 通过 APB 接口，配置 VCE 模式寄存器，读取 VCE 状态。
- DDR：VCE 通过 AXI 接口，读写 DDR。



29.4.3 数据结构

VCE 支持从 DVI 接口接收来自显卡的 RGB 数据，接口如表 29-1 所示；也支持直接从 DDR 读取数据，接口如表 29-1 所示。

表29-1 DVI 接口数据结构

Field Location	Field Name	Description
dvi_red_data[7:0]	R 数据	DVI 接口视频数据的 Red 数据。
dvi_green_data [7:0]	G 数据	DVI 接口视频数据的 Green 数据。
dvi_blue_data [7:0]	B 数据	DVI 接口视频数据的 Blue 数据。
dvi_hsync	行同步	DVI 接口视频数据的行同步信号，高有效。
dvi_vsync	帧同步	DVI 接口视频数据的帧同步信号，高有效。
dvi_data_en	数据使能	DVI 数据有效使能，高有效。

表29-2 Frame buffer 接口数据结构

Field Location	Field Name	Description
gpu_format[1:0]	数据格式	graphics frame buffer 存储数据格式。 00: 8-bit index color; 01: 16-bit 565 coclor; 10: 32-bit 888 coclor; 11: 禁用。
gpu_start_adr[27:6]	基地址	graphics frame buffer 存储视频数据的基地址，[31:28]由 CPU 通过 APB 配置，[5:0]VCE 自动补 0。
gpu_hde [11:0]	行像素点	行分辨率-1，屏幕 X 方向像素点个数减 1。

gpu_vde [10:0]	列像素点	列分辨率-1，屏幕 Y 方向像素点个数减 1。
gpu_offset[9:0]	行偏移	屏幕相邻 2 行的行起始地址的偏移量，单位为 16Byte，需要保证 gpu_offset[1:0]=0。

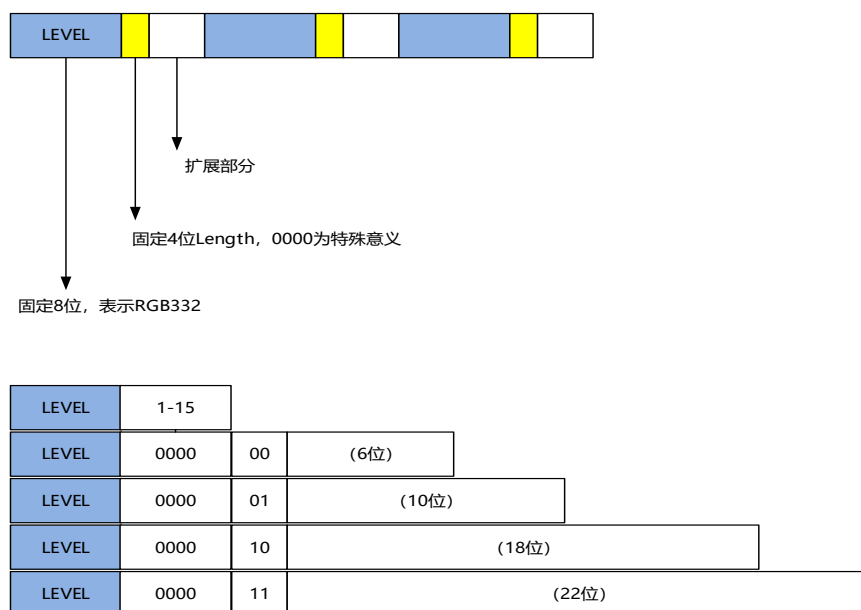
29.4.4 VCE 功能

29.4.4.1 压缩功能

模式一：

对于 I 帧压缩，把当前帧的 RGB888 数据截位成 RGB332，然后做变长 RLE 编码；对于 P 帧压缩，把前后两帧异或一下，做截位（和 I 帧一样的截位操作）和变长 RLE 编码。其压缩后的数据格式如图 29-2 所示。

图29-2 压缩后的数据格式



数据压缩的过程是不断记录连续相同的像素点的个数，如图 29-2 所示，我们把连续相同的像素点个数分成 5 个等级来表示，分别是：

- 相同个数范围为 1-15: LEVEL 表示像素点的值, 后 4bit 表示相同像素点的个数。



- 相同个数范围为 16-63: LEVEL 表示像素点的值, 接着 4bit 为 0000, 接着 2bit 表示数量等级 00, 最后 6 位表示相同像素点的个数。
- 相同个数范围为 64-1023: LEVEL 表示像素点的值, 接着 4bit 为 0000, 接着 2bit 表示数量等级 01, 最后 10 位表示相同像素点的个数。
- 相同个数范围为 1024- (256K-1): LEVEL 表示像素点的值, 接着 4bit 为 0000, 接着 2bit 表示数量等级 10, 最后 18 位表示相同像素点的个数;
- 相同个数范围为 256K-(4M-1): LEVEL 表示像素点的值, 接着 4bit 为 0000, 接着 2bit 表示数量等级 11, 最后 22 位表示相同像素点的个数。

模式二:

算法是 RLE 编码和 JPEG 编码, 一帧可以混用两种方式。对数据的处理是按照块的方式, 大小是 64*64。

- 块格式

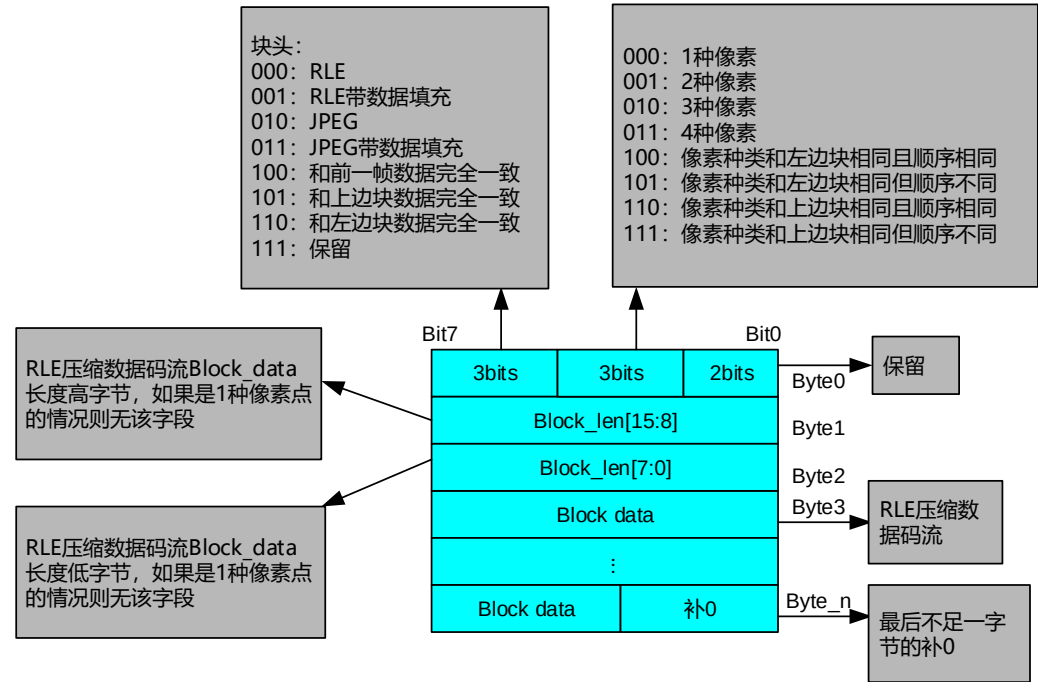
块格式如图 29-3 所示, 块头是前 3 个字节。

Byte0 的 bit [7:5]表示压缩模式; bit [4:2]表示 RLE 的像素种类, bit[4]为 1 表示该压缩块和左边块或上边块像素种类相同, 解压客户端可根据该字段从相关的块中获取到像素值及其顺序相关的信息, 因此压缩数据码流中就可以配置成不带像素值数据 (YUV 值), 以提高压缩比。对于 JPEG 压缩块则没有这个字段; bit[1:0]为保留位。

Byte1 为块长度的高 8 位, Byte2 为块长度的低 8 位。最后是 RLE 或 JPEG 压缩出来的数据码流, 由于压缩出来的码流不是字节对齐的, 所以在块的最后一个字节的无效数据部分填 0, 字节内的比特为大端对齐, 即高位为较早压缩出来的码流, 所以填 0 部分是字节的低位。



图29-3 块压缩格式

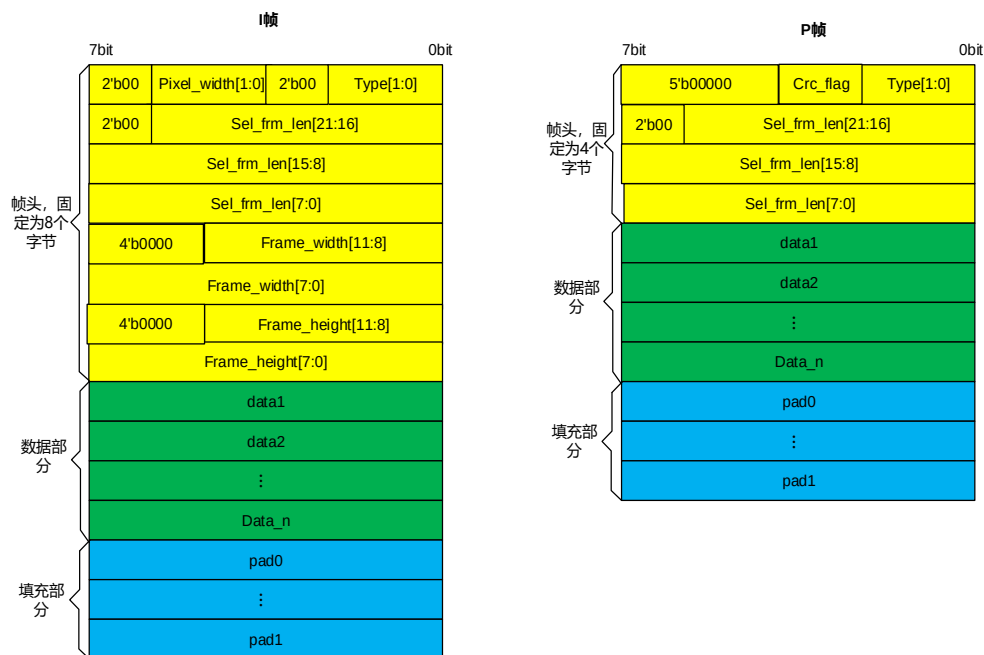


29.4.4.2 录屏功能

- 帧格式

录屏图像数据帧中, 包含有 I 帧和 P 帧, 它们有各自的存储格式, 如图 29-4 所示。

图29-4 I/P 帧录屏存储格式



- **Type:**
指示帧类型，2'b01: I 帧；2'b10: P 帧。
- **Crc_flag:**
CRC 相同标志（只在新算法中有效），1: 当前帧与上一帧完全相同；0: 当前帧与上一帧不相同；该字段只对 P 帧有效，对 I 帧无效。如果当前帧的 CRC 标志有效，则不存在压缩数据，所以长度字段为 0，类型字段 type 必定为 10（表示 P 帧），整帧在 DDR 中占 32Byte。
- **Pixel_width:**
指示压缩数的像素点的抽样：2'b0=6bit; 2'b01=7bit; 2'b10=8bit; 2'b11=4bit; 其中，8bit 表示 332 抽样，7bit 表示 223 抽样，6bit 表示 222 抽样，4bit 表示 112 抽样。该字段只用于压缩算法模式一（原来 KVM 逻辑的压缩算法），对于模式二，该字段为保留，其值设定为 2'b00。
- **Sel_frm_len:**
压缩帧长度，位宽 22bit，只计算有效的压缩数据部分，即图 29-4 的绿色部分。
- **frame_width:**
图像宽度，位宽 12bit。
- **frame_height:**



图像高度，位宽 12bit。

- Data (图 29-4 的绿色部分):

压缩帧数据内容

- Pad (图 29-4 的蓝色部分):

填充数据部分，由于写 DDR 是突发写，每次写 32Byte，为了方便地址的控制，每帧数据都占用 32Byte 整数倍的空间，有效数据不足 32Byte 整数倍的后面填充无用的数据，软件读取数据时只需要根据压缩帧长度把有效数据读走即可。

29.4.4.3 截屏功能

截屏有两种方式，软件截屏和管脚截屏。

29.5 工作方式

29.5.1 软复位

29.5.1.1 正常软复位

在 VCE 正常工作时软件不能随意对 VCE 模块单独进行软复位操作（否则可能会导致整个逻辑挂死），需要根据其工作模式，并按照一定的流程去复位。

模式一的复位步骤如下：

步骤 1 配置寄存器 FRM_CTRL_REG 为 0，关掉 DVI。

步骤 2 查询寄存器 VC_CORE_IDLE_FLAG 是否为 1，等待逻辑处理处于空闲状态，VC_CORE_IDLE_FLAG 为 1 时候，跳转至步骤 3。

步骤 3 复位整个 VCE 模块（将 HOST_SUBSYS 寄存器 HOST_RST_ENABLE 写 0x0000000C，读 HOST_RST_STATE 的 bit[3:2]为 2'b11 后，再将 HOST_RST_DISABLE 写 0x0000000C），复位后按照初始化规定的流程重新初始化 VCE。

----结束

模式二的复位步骤如下：

步骤 1 配置寄存器 ENVC_DVI_SWITCH 为 0，关掉 DVI。

步骤 2 查询寄存器 ENVC_CORE_IDLE_FLAG 是否为 1，等待逻辑处理处于空闲状态。



步骤 3 复位整个 VCE 模块（将 HOST_SUBSYS 寄存器 HOST_RST_ENABLE 写 0x0000000C，读 HOST_RST_STATE 的 bit[3:2]为 2'b11 后，再将 HOST_RST_DISABLE 写 0x0000000C），复位后按照初始化规定的流程重新初始化 VCE。

----结束

29.5.1.2 初始化

VCE 模式一初始化

- 步骤 1 配置系统控制器 SYSCTRL 的寄存器 MISC_CTRL 的 bit1 为 0，将 DVI 时钟来源、RGB 数据，以及行场数据有效使能连接内部输入。
- 步骤 2 把分配给 VCE 的基地址配置到寄存器 DDR_BASE_ADDR_HIGH_2BIT 和 DDR_BASE_ADDR，软件需要保证 DDR_BASE_ADDR 的低 6bit 为 0。
- 步骤 3 配置 VCE 访问 DDR 的大小端模式，寄存器 VCE_DDR_BIG_ENDIAN，默认 0 为小端模式；由于 Hi3093 项目只使用小端模式，这个寄存器禁止配置为 0x1。
- 步骤 4 配置寄存器 COMP_MODE 为 1，切换到模式一。
- 步骤 5 配置 KVM_EN 或 START_EN 为 0x1，打开 KVM 或录屏功能。
- 步骤 6 配置 FRM_CTRL_REG 的 bit[2]为 1，打开 DVI 口。如果步骤 5 对 KVM 和录屏都没有打开，就不能配置 FRM_CTRL_REG 的 bit[2]为 1，即不能打开 DVI 口。

----结束

29.5.1.3 VCE 模式二初始化

- 步骤 1 配置系统控制器寄存器 MISC_CTRL 的 bit[1]为 0，将 DVI 时钟来源、RGB 数据，以及行场数据有效使能连接内部输入。
- 步骤 2 把分配给 VCE 的基地址配置到寄存器 DDR_BASE_ADDR_HIGH_2BIT 和 DDR_BASE_ADDR，bit[31:6]为 DDR_BASE_ADDR 的高 26bit，DDR_BASE_ADDR 的低 6bit VCE 自动补 0，bit[5]配置为 1。
- 步骤 3 配置 VCE 访问 DDR 的大小端模式，寄存器 VCE_DDR_BIG_ENDIAN，默认 0 为小端模式；由于 Hi3093 项目只使用小端模式，这个寄存器禁止配置为 0x1。
- 步骤 4 配置寄存器 COMP_MODE 为 0，切换到模式二，默认为 0。



- 步骤 5 配置寄存器 FRM_BUF_DBG_EN 和 FRAME_BUF_MODE_EN 选择数据输入模式。
FRM_BUF_DBG_EN 配置为 0x1 则选择强制 Frame buffer 输入。
FRM_BUF_DBG_EN 配置为 0x0, 如果 FRAME_BUF_MODE_EN 配置为 0x0, 则强制 DVI 输入; FRM_BUF_DBG_EN 配置为 0x0, 如果 FRAME_BUF_MODE_EN 配置为 0x1, 则按配置分辨率自动切换, 切换分辨率配置 SWITCH_RESOLUTION。
- 步骤 6 配置 DDR_BASE_ADDR [5:1]选择是全部 JPEG 压缩, 还是 RLE 和 JPEG 混压。
DDR_BASE_ADDR [5]为 1, 如果块中出现 Y 值相同但 UV 不同的情况, 就用 JPEG 压缩, 不会统计像素种类, 只在 Frame buffer 输入起作用; DDR_BASE_ADDR [4:2]是 RLE 和 JPEG 混压的配置, 具体见寄存器描述, DVI 和 Frame buffer 输入都适用; 配置 DDR_BASE_ADDR [1]为 1, 全部用 JPEG 压缩, DVI 和 Frame buffer 输入都适用。但如果是 Frame buffer 输入且 DDR_BASE_ADDR bit[5]为 1, 那么 DDR_BASE_ADDR bit[5]先起作用。
- 步骤 7 配置 ENVC_CPU_RD_COMP_EN 或 ENVC_REC_EN 为 0x1, 打开 KVM 或录屏功能。
- 步骤 8 配置 ENVC_DVI_SWITCH 为 1, 打开 DVI 口。如果[步骤 7](#)对 KVM 和录屏都没有打开, 就不能配置 ENVC_DVI_SWITCH 为 1, 即不能打开 DVI 口。

---结束

29.5.2 VCE 压缩数据读取

29.5.2.1 模式一

由于受 DDR 带宽以及 DDR 延迟影响, 可能导致内部 FIFO 溢出。当接收 DVI 数据的 fifo 溢出时, VCE 会将 FRM_ERR_TYPE 的 bit[2]置 1。

模式一下读压缩数据的具体步骤如下:

- 步骤 1 CPU 每间隔 1ms 查询寄存器 SELFRM_STATUS 的 bit[0]是否为 1, 不为 1 时, 重复检测; 为 1 时, 跳转至[步骤 2](#)。也可以通过 data_rdy 中断来判断 data_rdy 是否为 1, 具体现象是中断状态有效后, VCE_INT_STATUS 的 bit[15]为 1; 如果用中断就需要清中断, 不用时开始就得设置 VCE_INT_EN 的 bit[13]屏蔽该中断。
- 步骤 2 CPU 检测 FRM_ERR_TYPE 的 bit[2]是否为 1, 如果为 1, 则直接将寄存器 CLR_SELFRM 的 bit[0]写 1 执行清帧操作, 跳转至[步骤 3](#); 如果为 0, 读 VC_CMP_STR_ADDR_HIGH_2BIT 和 VC_CMP_STR_ADDR 得到压缩数据起始地址, 读 SELFRM_STATUS 的 bit[2:1]得到帧类型, 读 SELFRM_ZIP_LEN 得到数据长



度，读 CURRFRM_COLUMN_LEN 和 CURRFRM_LINE_LEN 得到垂直分辨率和水平分辨率，执行步骤 3。

步骤 3 CPU 根据码流存放地址读取压缩码流，并组包发给客户端，发送完成后，将寄存器 CLR_SELFRM 的 bit[0]写 1 执行清帧操作，跳转至步骤 1。

----结束

29.5.2.2 模式二

由于不同分辨率和接口模式受 DDR 带宽以及 DDR 延迟影响，可能导致 VCE 内部 FIFO 溢出，以及 GPU 的时序异常都可能导致 VCE 接受到的原始视频数据错误，造成 VCE 压缩码流错误，由于该类原因引起的压缩码流错误，VCE 会将 ENVC_FRM_ERR_TYPE 的 bit[0]置 1，即如果接口模式为 frame buffer 模式，VCE 取 frame buffer 数据超时，VCE 将强制压缩结束，并将 ENVC_FRM_ERR_TYPE 的 bit[2]置 1，如果压缩码流的大小超出给定内存空间的大小 4MB，VCE 将寄存器 ENVC_FRM_ERR_TYPE 的 bit[3]置 1。

软件查询寄存器 ENVC_SELFRM_STATUS，在检测到 ENVC_SELFRM_STATUS 的 bit[0]为 1 时，再检测 ENVC_FRM_ERR_TYPE 寄存器。也可以通过 data_rdy 中断来判断 envc_comp_data_rdy 是否为 1。具体步骤如下：

步骤 1 CPU 每间隔 1ms 查询寄存器 ENVC_SELFRM_STATUS 的 bit[0]是否为 1，不为 1 时，重复检测；直到检测到 ENVC_SELFRM_STATUS 的 bit[0]为 1 时，跳转至步骤 2。也可以通过 data_rdy 中断来判断 envc_comp_data_rdy 是否为 1，具体现象是中断状态有效后，VCE_INT_STATUS 的 bit[15]为 1；如果用中断就需要清中断，不用的话开始就得设置 VCE_INT_EN 的 bit[0]屏蔽该中断。

步骤 2 CPU 检测 ENVC_FRM_ERR_TYPE 的 bit[3:0]，判断寄存器是否为 0，如果为 0，则读取码流存放地址寄存器 ENVC_CMP_STR_ADDR_HIGH_2BIT 和 ENVC_CMP_STR_ADDR，分辨率信息寄存器 ENVC_RESOLUTION，Frame buffer 数据格式寄存器 GPU_FORMAT，跳转至步骤 3。如果寄存器不为 0，则直接将寄存器 ENVC_CLR_COMP_FRM 的 bit[0]写 1 执行清帧操作，跳转至步骤 1。

步骤 3 CPU 根据码流存放地址读取压缩码流，并组包发给客户端，发送完成后，读取完成后将寄存器 ENVC_CLR_COMP_FRM 的 bit[0]写 1，执行清帧操作，跳转至步骤 1。

----结束



29.5.3 VCE 录屏读取

29.5.3.1 模式一读取录屏

CPU 读取录屏操作步骤如下：

- 步骤 1 配置寄存器 REC_SIZE_AFTER_TRIG，该寄存器是从触发使能下发开始继续录屏的数据量。
- 步骤 2 配置寄存器 TRI_ABN_EN 为 1，CPU 准备读录屏数据之前先下发触发使能通知 VCE 准备要读取录屏数据，VCE 要记录此刻的 I 帧数量；我们将此刻称作触发时刻，VCE 要实时更新从触发时刻开始录下的 I 帧数寄存器，并记录当前录屏地址，根据地址判断如果从触发使能下发开始录入了 REC_SIZE_AFTER_TRIG 配置的数据量后自动停止录入视频数据，CPU 可实时读取 I 帧个数寄存器。
- 步骤 3 CPU 收到 VCE_INT_STATUS 的 bit[19]为高的录屏停止中断或查询 REC_LOCK_FINISH 有效后配置 START_EN 为 0，关闭录屏。关闭录屏后等待 1ms，对于触发使能下发后录的数据，CPU 读寄存器 TRI_I_NUM 可以知道从下发触发使能时刻开始录下的 I 帧数，再配置 TRI_ABN_EN 为 0，关闭触发使能。如果用中断，用完后就清除对应位的中断；如果查询 REC_LOCK_FINISH，用完就得配置 CLR_REC_LOCK_FINISH 为 1 清除状态。另外，有可能存在 GPU 在输入在没录满再录空间就断了的情况，这个时候中断和状态都不会有效。就得检测视频丢失，如果视频丢失，就继续执行本步骤第 1 句话后面的操作，一直到步骤 6 完成读录屏，然后再进入截屏流程；没有丢失，继续检测 160ms，直到录屏截止或发现视频丢失。
- 步骤 4 CPU 读寄存器 ALL_I_NUM 得知 I 帧总数，假设为 x 个；配置 READ_NUM_P_FRM_EN 为 1 打开读开关，对寄存器 NUM_P_FRM 读 x 次，就得到每个 I 帧后面的 P 帧个数；再根据要读的录屏长度配置 READ_I_NUM；最后配置 READ_EN 为 1，把要读取数据对应的 DDR 空间的起始地址和长度锁存到寄存器。
- 步骤 5 CPU 读寄存器 REC_STR_ADDR_HIGH_2BIT、REC_STR_ADDR 和寄存器 READ_I_SIZE 获取录屏数据的起始地址和长度后，将整块数据从 DDR 中读出，然后配置 READ_EN 为 0。如果步骤 4 配置了 READ_NUM_P_FRM_EN 为 1，这里需要配置 READ_NUM_P_FRM_EN 为 0。
- 步骤 6 如果需要继续录屏则配置 START_EN 为 1，重新打开录屏使能，重新打开后，原来录入的数据将被清空，即录入的 I 帧数变为 0。

----结束



29.5.3.2 模式二读取录屏

CPU 读取录屏操作步骤如下：

- 步骤 1 配置寄存器 REC_SIZE_AFTER_TRIG，该寄存器是从触发使能下发开始继续录屏的数据量。
- 步骤 2 配置寄存器 ENVC_TRIG_EN 为 1，CPU 准备读录屏数据之前先下发触发使能通知 VCE 准备要读取录屏数据，VCE 要记录此刻的 I 帧数量；我们将此刻称作触发时刻，VCE 要实时更新从触发时刻开始录下的 I 帧数寄存器，并记录当前录屏地址，根据地址判断如果从触发使能下发开始录入了 REC_SIZE_AFTER_TRIG 配置的数据量的数据后自动停止录入视频数据，CPU 可实时读取 I 帧个数寄存器。
- 步骤 3 CPU 收到 VCE_INT_STATUS 的 bit[19]为高的录屏停止中断或查询 REC_LOCK_FINISH 有效后配置 START_EN 为 0，关闭录屏。关闭录屏后等待 1ms，对于触发使能下发后录的数据，CPU 读寄存器 ENVC_TRI_I_NUM 可以知道从下发触发使能时刻开始录下的 I 帧数，再配置 ENVC_TRIG_EN 为 0，关闭触发使能。如果用中断，用完后就得清对应位的中断；如果查询 REC_LOCK_FINISH，用完就得配置 CLR_REC_LOCK_FINISH 为 1 清状态。另外，有可能存在 GPU 在输入在没录满再录空间就断了的情况，这个时候中断和状态都不会有效。就得检测视频丢失，如果视频丢失，就继续执行本步骤第 1 句话后面的操作，一直到步骤 6 完成读录屏，然后再进入截屏流程；没有丢失，继续检测 160ms，直到录屏截止或发现视频丢失。
- 步骤 4 CPU 读寄存器 ENVC_ALL_I_NUM 得知 I 帧总数，假设为 x 个；配置 READ_NUM_P_FRM_EN 为 1 打开读开关，对寄存器 NUM_P_FRM 读 x 次，就得到每个 I 帧后面的 P 帧个数；再根据要读的录屏长度配置 ENVC_READ_I_NUM；最后配置 ENVC_REC_READ_EN 为 1，把要读取数据对应的 DDR 空间的起始地址和长度锁存到寄存器。
- 步骤 5 CPU 读寄存器 ENVC_REC_STR_ADDR_HIGH_2BIT、ENVC_REC_STR_ADDR 和寄存器 ENVC_READ_I_SIZE 获取录屏数据的起始地址和长度后，将整块数据从 DDR 中读出，然后配置 ENVC_REC_READ_EN 为 0；如果步骤 4 配置了 READ_NUM_P_FRM_EN 为 1，这里需要配置 READ_NUM_P_FRM_EN 为 0。
- 步骤 6 如果需要继续录屏则配置 ENVC_REC_EN 为 1，重新打开录屏使能，重新打开后，原来录入的数据将被清空，即录入的 I 帧数变为 0。

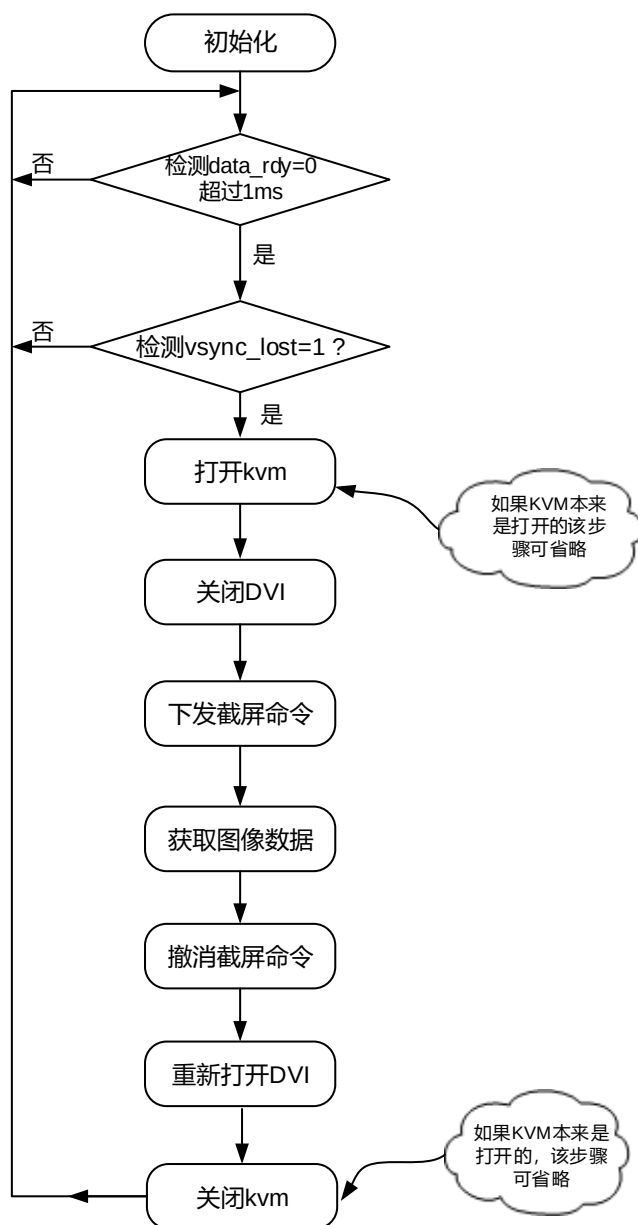
----结束



29.5.4 VCE 截屏

29.5.4.1 模式一截最后一屏

图29-5 模式一截最后一屏软件处理流程图



步骤 1 初始化：配置寄存器 START_EN 为 1，打开录屏，并保证不关闭，如果客户端关闭则需要关闭 KVM 使能（配置寄存器 KVM_EN 为 0）。

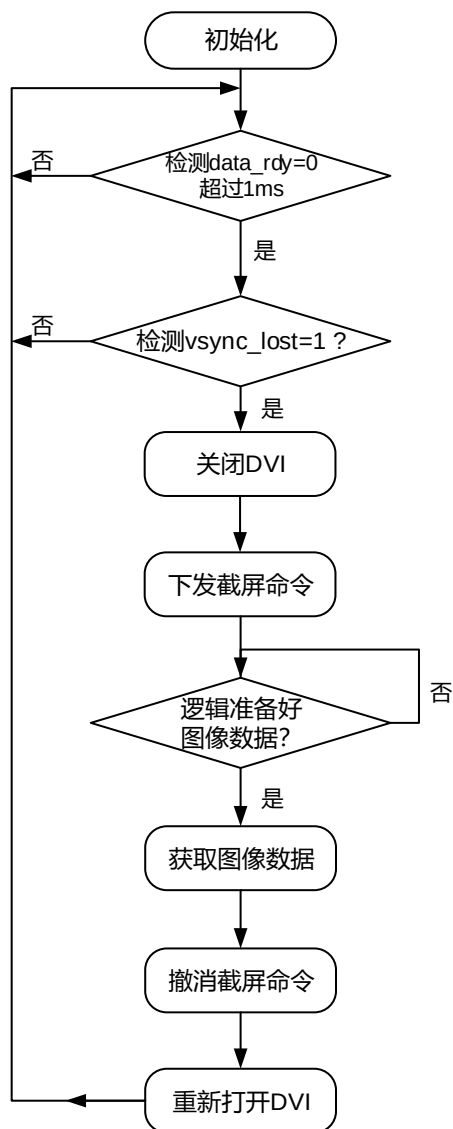


- 步骤 2 检测 data_rdy=0 是否超过 1ms: CPU 读寄存器 SELFRM_STATUS, SELFRM_STATUS 的 bit[0]是否持续为零 1ms 以上。也可以通过 data_rdy 中断来判断 data_rdy 是否为 1, 具体现象是中断状态有效后, VCE_INT_STATUS 的 bit[15]为 1。
- 步骤 3 检测 vsync_lost: CPU 读寄存器 FRM_ERR_TYPE, 判断 FRM_ERR_TYPE 的 bit[1] 是否为 1, 为 1 执行[步骤 4](#), 为 0 返回执行[步骤 2](#)。也可以通过 vsync_lost 中断来判断 vsync_lost 是否为 1, 具体现象是中断状态有效后, VCE_INT_STATUS 的 bit[18] 为 1。
- 步骤 4 打开 kvm: 配置寄存器 KVM_EN 为 1。
- 步骤 5 关闭 DVI: 配置寄存器 FRM_CTRL_REG 为 0。
- 步骤 6 下发截屏命令: 若截最后一屏, 配置寄存器 FRM_CTRL_REG 为 3; 若截倒数第二屏, 配置寄存器 FRM_CTRL_REG 为 1。
- 步骤 7 获取图像数据: 读取寄存器 SELFRM_ZIP_LEN、VC_CMP_STR_ADDR_HIGH_2BIT 和 VC_CMP_STR_ADDR 获取压缩数据基地址和长度, 然后读取压缩数据, 该步骤需要在下发截屏命令 300ns 后才能进行。
- 步骤 8 撤消截屏命令: 配置寄存器 FRM_CTRL_REG 为 0。
- 步骤 9 重新打开 DVI: 配置寄存器 FRM_CTRL_REG 为 4。
- 步骤 10 关闭 kvm: 配置寄存器 KVM_EN 为 0x0。

----结束

29.5.4.2 模式二截最后一屏

图29-6 模式二截最后一屏软件处理流程图



步骤 1 初始化：配置寄存器 ENVC_REC_EN 为 1，打开录屏，并保证不关闭，如果客户端关闭则需要关闭 KVM 使能，配置寄存器 ENVC_CPU_RD_COMP_EN 为 0。

步骤 2 检测 data_rdy=0 超过 1ms：读寄存器 ENVC_SELFRM_STATUS 判断 ENVC_SELFRM_STATUS 的 bit[0]是否持续为零 1ms 以上。也可以通过 data_rdy 中断来判断 data_rdy 是否为 1，具体现象是中断状态有效后，VCE_INT_STATUS 的 bit[15]为 1。



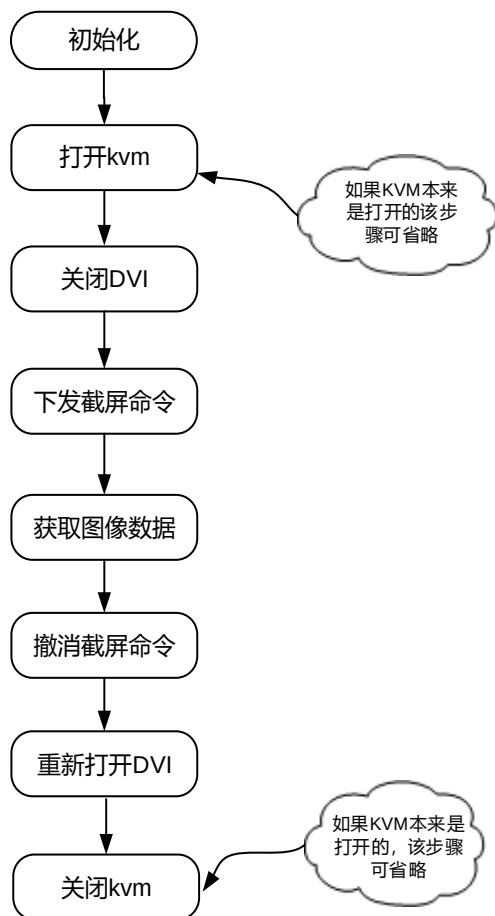
- 步骤 3 检测 vsync_lost: 读寄存器 ENVC_FRM_ERR_TYPE 判断 ENVC_FRM_ERR_TYPE 的 bit[1] 是否为 1, 为 1 执行步骤 4, 为 0 返回执行步骤 2。也可以通过 vsync_lost 中断来判断 vsync_lost 是否为 1, 具体现象是中断状态有效后, VCE_INT_STATUS 的 bit[18] 为 1。
- 步骤 4 关闭 DVI: 配置寄存器 ENVC_DVI_SWITCH 为 0。
- 步骤 5 下发截屏命令: 若截最后一屏, 配置寄存器 GET_LAST_FRM_CMD 为 1; 若截倒数第二屏, 配置寄存器 GET_LAST_FRM_CMD 为 2。
- 步骤 6 判断逻辑是否准备好图像数据: 读寄存器 GET_LAST_FRM_STATUS, 判断 GET_LAST_FRM_STATUS 的 bit[0], 为 1 表示准备好。
- 步骤 7 获取图像数据: 读取寄存器 LAST_FRM_START_ADDR_HIGH_2BIT 和 LAST_FRM_START_ADDR 获取原始 YUV 数据基地址, 读取寄存器 LAST_FRM_LINE 和 LAST_FRM_COLUMN 获取分辨率信息, 然后读取图像数据。
- 步骤 8 撤消截屏命令: 配置寄存器 GET_LAST_FRM_CMD 为 0。
- 步骤 9 重新打开 DVI: 配置寄存器 ENVC_DVI_SWITCH 为 1。

----结束

29.5.4.3 手动截屏

手动截屏是软件在逻辑压缩的过程中随机下发的截屏命令, 模式一 VC_CORE 的手动截屏与截最后一屏相比, 省去了 data_rdy 和 vsync_lost 的判断, 如图 29-7 所示。

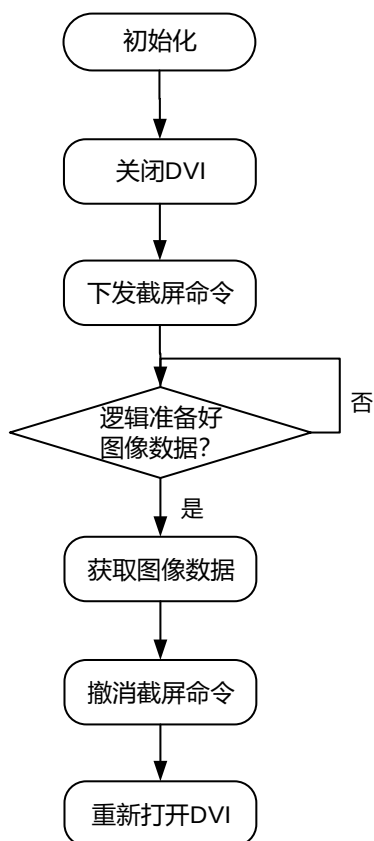
图29-7 模式一 VC_CORE 手动截屏流程图



每个步骤与模式一 VC_CORE 截最后一屏相同，但在关闭 DVI 后需要等待 100ms 以上再去截屏。

同样，模式二的手动截屏命令与截最后一屏相比，省去了 data_rdy 和 vsync_lost 的判断，如图 29-8 所示。

图29-8 模式二 ENVC_CORE 手动截屏流程图



每个步骤与模式二 ENVC_CORE 截最后一屏相同，但在关闭 DVI 后需要等待 100ms 以上再去截屏。

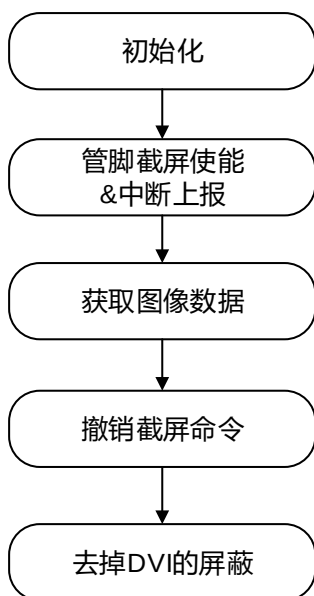
ENVC_CORE 在 frame_buffer 下载屏如果遇到 GPU 改变 frame_buffer 格式为 VGA 时可能会截出花屏，所以软件需要在下发截屏命令和逻辑准备好图像数据之间不断查询寄存器 gpu_format 是否为 0（为 0 表示 VGA 模式），如果出现过读出为 0 的情况则应该丢弃该截屏图像数据，重新截屏，直到读出 gpu_format 值不存在 0 的情况为止。



29.5.4.4 管脚截屏

模式一的管脚截屏

图29-9 模式一 VC_CORE 管脚截屏流程图



- 步骤 1 初始化：配置寄存器 START_EN 为 1，打开录屏，并保证不关闭；配置寄存器 KVM_EN 为 1，打开 KVM 使能；配置 SCREEN_SHOT_IO_EN 为 1，打开管脚截屏开关；若管脚截屏截最后一屏，配置寄存器 SCREEN_SHOT_FRM_TYPE_SEL 为 1；若截倒数第二屏，配置寄存器 SCREEN_SHOT_FRM_TYPE_SEL 为 0；。
- 步骤 2 管脚截屏使能&中断上报：截屏管脚触发后，CPU 可以通过 VCE_INT_STATUS 的 bit[14]的中断或者查询寄存器 SCREEN_SHOT_IO_VLD 得知。如果用中断，就需要清中断；如果查询寄存器，寄存器有效之后，就得配置寄存器 CLR_SCREEN_SHOT_IO_VLD 为 1 清除寄存器。这里需要软件关注，如果发生管脚截屏，硬件屏蔽 DVI，那么视频丢失信号就可能会置高，这个时候不能认为是视频丢失。
- 步骤 3 获取图像数据：CPU 等待 100ms+300ns，再读取寄存器 SELFRM_ZIP_LEN、VC_CMP_STR_ADDR_HIGH_2BIT 和 VC_CMP_STR_ADDR 获取压缩数据基址和长度，然后读取压缩数据。100ms 的原因如步骤 2 描述，300ns 的原因是这些信息需要截屏信号起作用 300ns 后才能读取。



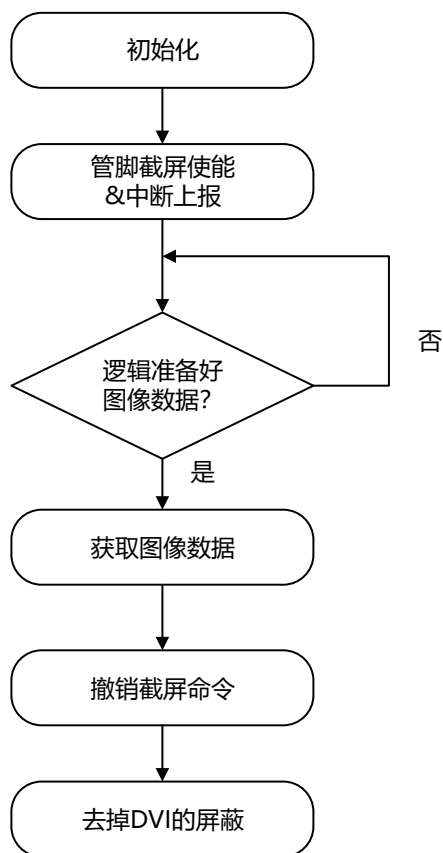
步骤 4 撤消截屏命令：配置 SCREEN_SHOT_IO_EVENT_FINISH 为 1。

步骤 5 去掉 DVI 的屏蔽：配置 REMOVE_MASK_DVI_BY_SCREEN_SHOT_IO 为 1，使 DVI 重新起作用。

----结束

模式二的管脚截屏

图29-10 模式二 ENVC_CORE 手动截屏流程图



步骤 1 初始化：配置寄存器 ENVC_REC_EN 为 1，打开录屏，并保证不关闭；如果客户端关闭则需要关闭 KVM 使能，配置寄存器 ENVC_CPU_RD_COMP_EN 为 00。配置 SCREEN_SHOT_IO_EN 为 1，打开管脚截屏开关；对于 DVI 模式，若管脚截屏截最后一屏，配置寄存器 SCREEN_SHOT_FRM_TYPE_SEL 为 1；若截倒数第二屏，配置寄存器 SCREEN_SHOT_FRM_TYPE_SEL 为 0。对于 frame buffer 模式，固定截最后一屏，寄存器 SCREEN_SHOT_FRM_TYPE_SEL 不起作用。



- 步骤 2 管脚截屏使能&中断上报：截屏管脚触发后，CPU 可以通过寄存器 VCE_INT_STATUS 的 bit[14]的中断或者查询寄存器 SCREEN_SHOT_IO_VLD 得知。如果用中断，就需要清中断；如果查询寄存器，寄存器有效之后，就得配置寄存器 CLR_SCREEN_SHOT_IO_VLD 为 1 清除寄存器。这里需要软件关注，如果发生管脚截屏，硬件会屏蔽 DVI，那么视频丢失信号就可能会置高，这个时候不能认为是视频丢失。
- 步骤 3 判断逻辑是否准备好图像数据：读寄存器 GET_LAST_FRM_STATUS，判断 GET_LAST_FRM_STATUS 的 bit[0]，为 0x1 表示准备好。
- 步骤 4 获取图像数据：读取寄存器 SELFRM_ZIP_LEN、VC_CMP_STR_ADDR_HIGH_2BIT 和 VC_CMP_STR_ADDR 获取压缩数据基地址和长度，然后读取压缩数据。
- 步骤 5 撤消截屏命令：配置 SCREEN_SHOT_IO_EVENT_FINISH 为 1。
- 步骤 6 去掉 DVI 的屏蔽：配置 REMOVE_MASK_DVI_BY_SCREEN_SHOT_IO 为 1，使 DVI 重新起作用。

----结束

29.5.5 压缩模式切换操作

模式一和模式二之间的切换，都需要先按照当前模式参照正常软复位流程对整个 VCE 进行复位后再重新按照初始化的流程对 VCE 进行初始化（复位后 VCE 所有寄存器的值恢复到初始值），具体操作如下。

29.5.5.1 模式一切换成模式二

- 步骤 1 配置寄存器 FRM_CTRL_REG 为 0，关掉 DVI。
- 步骤 2 查询寄存器 VC_CORE_IDLE_FLAG 是否为 1，等待逻辑处理处于空闲状态，VC_CORE_IDLE_FLAG 为 1 时候，跳转至[步骤 3](#)。
- 步骤 3 复位整个 VCE 模块(将 HOST_SUBSYS 寄存器 HOST_RST_ENABLE 写 0x0000000C，读 HOST_RST_STATE 的 bit[3:2]为 2'b11 后，再将 HOST_RST_DISABLE 写 0x0000000C)。
- 步骤 4 配置系统控制器寄存器 MISC_CTRL 的 bit[1]为 0，将 DVI 时钟来源、RGB 数据，以及行场数据有效使能连接内部输入。
- 步骤 5 把分配给 VCE 的基地址配置到寄存器 DDR_BASE_ADDR_HIGH_2BIT 和 DDR_BASE_ADDR，bit[31:6]为 DDR_BASE_ADDR 的高 26bit，低 6bit VCE 自动补 0，bit[5]配置为 1，用于 Frame buffer 模式解决 YUV 数据 Y 值冲突问题。



步骤 6 配置 VCE 访问 DDR 的大小端模式，寄存器 VCE_DDR_BIG_ENDIAN，默认 0 为小端模式；由于 Hi3093 项目只使用小端模式，这个寄存器禁止配置为 0x1。

步骤 7 配置寄存器 COMP_MODE 为 0，切换到模式二，默认为 0。

步骤 8 配置 ENVC_CPU_RD_COMP_EN 或 ENVC_REC_EN 为 1，打开 KVM 或录屏功能。

步骤 9 配置 ENVC_DVI_SWITCH 为 1，打开 DVI 口。如果步骤 8 对 KVM 和录屏都没有打开，就不能配置 ENVC_DVI_SWITCH 为 1，即不能打开 DVI 口。

----结束

29.5.5.2 模式二切换到模式一

步骤 1 配置寄存器 ENVC_DVI_SWITCH 为 0，关掉 DVI。

步骤 2 查询寄存器 ENVC_CORE_IDLE_FLAG 是否为 1，等待逻辑处理处于空闲状态。

步骤 3 复位整个 VCE 模块（将 HOST_SUBSYS 寄存器 HOST_RST_ENABLE 写 0x0000000C，读 HOST_RST_STATE 的 bit[3:2]为 2'b11 后，再将 HOST_RST_DISABLE 写 0x0000000C）。

步骤 4 配置系统控制器寄存器 MISC_CTRL 的 bit[1]为 0，将 DVI 时钟来源、RGB 数据，以及行场数据有效使能连接内部输入。

步骤 5 把分配给 VCE 的基地址配置到寄存器 DDR_BASE_ADDR_HIGH_2BIT 和 DDR_BASE_ADDR，软件需要保证 DDR_BASE_ADDR 的低 6bit 为 0。

步骤 6 配置 VCE 访问 DDR 的大小端模式，寄存器 VCE_DDR_BIG_ENDIAN，默认 0 为小端模式；由于 Hi3093 项目只使用小端模式，这个寄存器禁止配置为 0x1。

步骤 7 配置寄存器 COMP_MODE 为 1，切换到模式一。

----结束

29.6 DFX

29.6.1 提供智能心跳功能

VCE 提供智能心跳功能，可实时监控内部状态机和 CPU。智能心跳功能必须在录屏打开的情况下才能使用。



状态机智能心跳功能：对模块内部状态机进行超时监控，一旦超时，就上报中断。该流程需要在寄存器 STATE_MONITOR_TMOUT_SET 设置超时参数，配置 VCE_INT_EN 的 bit[13]为 1。

CPU 智能心跳功能：对 CPU 进行超时监控，CPU 定时对 VCE 特定寄存器进行写操作，每次写数据和前一次数据相反，当 VCE 检测到一定时间内 CPU 没有操作该寄存器，则表示超时，CPU 心跳停止，VCE 进入自动录屏状态。

CPU 智能心跳配置流程：

- 步骤 1 配置心跳定时计数器 KVM_HEARTBEAT_TMOUT_SET，设置心跳超时时间，单位为 1ms。如：配置 10，即心跳超时时间为 10ms。
- 步骤 2 配置寄存器 KVM_HEARTBEAT_WORK_EN 为 1 打开心跳使能，VCE 内部心跳计数器开始计数。
- 步骤 3 CPU 每隔 1ms 写一次 KVM_HEARTBEAT_CNT_CLEAR，如果在等待 KVM_HEARTBEAT_TMOUT_SET 寄存器设置的时间后，CPU 未曾写过 KVM_HEARTBEAT_CNT_CLEAR 寄存器，则 VCE 认为 CPU 已经挂死，此时 VCE 自动关闭 CPU 读压缩操作，保留录屏操作；在 KVM_HEARTBEAT_TMOUT_SET 配置的超时时间内（如 10ms 时间内），CPU 清除内部心跳计数器，即 KVM_HEARTBEAT_CNT_CLEAR 写 1 清除，则内部心跳计数器重新计数。
- 步骤 4 不需要心跳功能时，则关闭心跳功能，写 KVM_HEARTBEAT_WORK_EN 为 0，CPU 不再需要定时写 KVM_HEARTBEAT_CNT_CLEAR 寄存器。

----结束

29.6.2 支持内部所有寄存器可访问

模块内部所有配置寄存器可通过 APB 总线访问。

29.6.3 支持统计寄存器可访问

支持对压缩性能和调试统计寄存器的访问；

支持对压缩帧最大、最小和平均压缩时延统计。

记录压缩过程中单帧最大压缩延时、最小压缩延时，统计 1 秒内压缩帧平均压缩延时。

支持接收和压缩帧统计：



- 统计压缩过程中从 DVI 接收和 Frame Buffer 读取帧数，统计压缩过程中压缩帧数。
- 统计由于 DDR 延迟或压缩性能导致丢帧数，包括单独对整帧丢帧的统计和非整帧丢帧的统计。
- AXI 最大、最小和平均访问延迟统计。
- 每个压缩帧的压缩块统计。

29.6.4 支持关键信号观测

JPGE IP 压缩状态信号观测。

帧有效标记、帧压缩完成标记、RLE_JPEG 压缩使能、CRC 比较结果观测。

29.6.5 支持 CPU 模拟 Frame buffer 输入

在不依赖 GPU 的前提下，本模块支持 CPU 模拟 GPU 给 VCE 输入 Frame buffer 的数据。设计到的寄存器是 TEST_FRAME_BUFFER_EN、TEST_GPU_FORMAT、TEST_GPU_START_ADD、TEST_GPU_HDE、TEST_GPU_VDE、TEST_GPU_OFFSET 和 TEST_DVI_VSYNC。

其中，TEST_GPU_FORMAT、TEST_GPU_START_ADD、TEST_GPU_HDE、TEST_GPU_VDE、TEST_GPU_OFFSET 和 TEST_DVI_VSYNC 分别对应 GPU 给过来的 gpu_format、gpu_start_adr、gpu_hde、gpu_vde、gpu_offset 和 dvi_vsync。TEST_FRAME_BUFFER_EN 是对 CPU 配置和 GPU 给过来那些信号的选通。

Debug 过程如下：

- 步骤 1 把帧数据写到 DDR 中，起始地址需要是 64Byte 对齐，行偏移需要是 64Byte 整数倍。
- 步骤 2 把帧的数据格式、起始地址、水平分辨率、垂直分辨率、行偏移分别配置到 TEST_GPU_FORMAT、TEST_GPU_START_ADD、TEST_GPU_HDE、TEST_GPU_VDE、TEST_GPU_OFFSET。
- 步骤 3 配置 TEST_FRAME_BUFFER_EN 为 1。
- 步骤 4 配置 COMP_MODE 为 0，选择模式二。
- 步骤 5 配置 ENVC_CPU_RD_COMP_EN 为 1，打开压缩功能。
- 步骤 6 配置 ENVC_DVI_SWITCH 为 1，打开 DVI 口。
- 步骤 7 配置 TEST_DVI_VSYNC 为 1。（该寄存器是自清零寄存器）



步骤 8 VCE 硬件进行工作。步骤 7 完成后，过一帧的时间（以显示器播放时间为准），再进行步骤 1/步骤 2/步骤 7，进行下一帧配置。其他步骤不需要重复。

----结束

说明

测试模式时，如果在步骤 8 中，上一帧已经播放完了，下一帧的 TEST_DVI_VSYNC 没有及时配置，会导致视频丢失中断，这是正常的，不用关注。



30 DMAC

30.1 概述

DMA（直接存储器访问）是一种由硬件执行数据搬移的工作方式，一般用于高速传送成组或者成块的数据。

DMA 通过一个独立于 CPU 的总线控制器申请总线，根据预先配置，实现数据在存储器及外设之间自动搬移，减轻 CPU 搬移数据的负担。

Hi3093 DMAC 只支持小端模式，不支持大端模式。

30.2 特性

- 提供 16 个 DMA 通道，每个通道可用于一种单向传输。
- 提供 32 个 DMA 外设请求输入。
- 外设可使用单次传输（Single）DMA 和连续传输（Burst）DMA 两种请求，任何一个与 DMAC 连接的外设都可同时使用这两种请求信号。
- Burst 大小（1 ~ 16）可编程；通常 Burst 的长度设为外设的 FIFO 深度的一半。
- 支持四种 DMA 传输：存储器至外设，外设至存储器，外设至外设，存储器至存储器。DMA 不可访问安全空间。注意：在地址扩展模式下，只支持存储器至存储器。
- 支持 scatter/gather 存储（单节点、或多节点链表 LLI（LinkedListItem）以供 DMAC 一次传完）。
- DMA 传输长度可由 DMAC 本身控制。
- DMAC 作为流控器时，一个链表项对应的最大传输尺寸为 512K（Beat）。
- 每个 DMA 通道具有各自的硬件优先级，采用优先级循环或绝对优先级方式。



- 提供专门的编程接口，CPU 通过此接口对 DMAC 的寄存器进行配置。
- 源/目的端的地址均能选择在 DMA 传输过程中自动递增/不递增。
- DMAC 内部各个通道间共享两个 16×64bit 的 FIFO。
- 支持 8、16、32 和 64 位宽的传输，源/目的端可采用不同的位宽，但要求源/目的数据传输量要求一致。
- 支持外设发出的 OKAY、SLVERR、和 DECERR 响应。从设备发出 OKAY 响应表明传输已完成。总线 Interconnect 或者从设备发出 SLVERR 或者 DECERR 响应表明产生了一个错误，此时 DMAC 发出错误中断请求。
- 支持产生 DMAError 中断和 DMAcount 中断，也支持产生由这两者组合而成的组合中断。
- 支持三组独立的中断信号分别输出中断给不同的 CPU。Hi3093 中把 3 组中断全部连接给 A55、协处理器 M3 和安全处理器 M3。具体中断分配请参见 4 GIC 章节。不支持提供通过 DMAC 的安全和非安全空间之间的 DMA 操作。
 - DMACINTC，表示 DMA 传输完成。
 - DMACINTERR，表示传输发生了错误。
 - DMACINTR，由前面两个中断组合而成，当系统的中断控制器输入线较少时可使用此中断作为中断控制器的输入。
- 支持中断屏蔽，DMAError 和 DMAterminalcount 中断都可被屏蔽。

支持 DMAC 的原始中断状态（Rawinterruptstatus）读取，DMAError 和 DMAterminalcount 的原始中断状态都可读取。

30.3 关键术语

表30-1 DMAC 关键术语描述

术语	描述
DMA	Direct Memory Access，直接存储器访问。是一种由硬件执行数据搬移的工作方式，一般用于高速传送成组或者成块的数据。
DMAC	DMA Controller，DMA 控制器。



30.4 功能

30.4.1 几个概念

- 一次 DMA 传输：DMAC 模块有 16 个通道，每次传输软件选择一个空闲通道，配置完成相关参数后，配置通道寄存器 `DMAC_CMCONFIG[ce]=1` 启动 DMA 传输，随后 DMAC 无需 CPU 干预按照配置要求完成从源地址到目标地址的数据传输，传输完成后上报中断并自动置 `DMAC_CMCONFIG[ce]=0`，这个过程称作一次 DAM 传输。
- 链表：一次 DMA 传输，可以传输 $n(n \geq 1)$ 个地址不连续的数据块，从芯片角度 n 无最大限制，取决于系统内存， n 个数据块的控制信息组成 1 个链表。
- 链表节点：每个数据块的传输通过 1 个链表节点控制， n 个数据块的传输通过 1 个具有 n 个节点的链表控制。
- 循环优先级仲裁：从通道 0~15 优先级不固定，初始化时通道 0~15 优先级依次由高到低组成一个优先级队列，随后获得服务（一个 burst 传输）的通道优先级排到队尾。
- 固定优先级仲裁：通道 0~15 优先级固定，依次由高到低。固定优先级仲裁会把通道 0 的请求全部完成之后再服务通道 1。

说明

- 链表由软件创建，保存在内存中。
- 每个链表节点占 4 个 32bit，内容和 `DMAC_CmSRCADDR`、`DMAC_CmDESTADDR`、`DMAC_CmLLI`、`DMAC_CmCONTROL` 一一对应。每个链表节点都有下一个节点存放地址指示，因此链表节点在内存存放也可以地址不连续。
- 链表首节点由软件写入 DAMC 模块通道（链表）节点级 4 个配置寄存器中，随后 DMAC 在传输完成一个链表节点后会自动加载下一个链表节点。
- `sburst_length`、`dburst_length`：链表节点级 burst 长度配置。值越大，单次总线请求搬运的数据越多，效率越高。
- DMAC 要求数据宽度和 burst 长度严格满足如下关系：
 $swidth * sburst_length = dwidth * dburst_length$ ，错误配置可能导致 DMAC 模块挂死。

30.4.2 控制器类型

DMAC 的控制器类型，通过通道（链表）级配置寄存器 `DMAC_CmCONFIG[flowctrl]` 配置，DMAC 模块设计比较灵活，应用时可以简化：

- 000：DMA 作控制器（传输长度通过链表节点寄存器 `DMAC_CmCONTROL[transfersize]` 控制，单位为 beat（源数据宽度））；



- 其余配置：保留。

说明

- 控制器决定传输的控制方，如 DMA 做控制器，则传输控制方为 DMA，配置完成源目标地址、搬移长度后，DMA 控制直到传输完成。
- 传输数据源、目的地址与控制器类型无关，即 DMA 做控制器，也可以在存储器及外设间地址传输数据。

30.4.3 外设请求分配

外设类型有 SPI、I2C、UART、外部 DMA 共 4 类，其中 SPI、I2C、UART 外设置于芯片里面（普通外设），而外部 DMA 置于其他 IP 内，例如 eMMC 内的 DMA 等。由于芯片内的外设多于 32 个，而 DMAC 最多支持 32 个外设，在使用 DMAC 时，使用寄存器矩阵对每个 DMAC 外设端口进行配置；配置矩阵可以参考 LP_SUBSYS 的寄存器。

默认 DMAC 配置中的外设请求分配表如表 30-2 所示。

表30-2 外设请求分配表

外设 ID	外设信号	描述
0	ssp0_dma_tx_req	SSP 发送方向请求
1	ssp0_dma_rx_req	SSP 接收方向请求
2	I2C0_dma_tx_req	I2C0 发送方向请求
3	I2C0_dma_rx_req	I2C0 接收方向请求
4	I2C1_dma_tx_req	I2C1 发送方向请求
5	I2C1_dma_rx_req	I2C1 接收方向请求
6	Uart2_dma_tx_req	UART2 发送方向请求
7	Uart2_dma_rx_req	UART2 接收方向请求
8	Uart3_dma_tx_req	UART3 发送方向请求
9	Uart3_dma_rx_req	UART3 接收方向请求
10	Uart4_dma_tx_req	UART4 发送方向请求
11	Uart4_dma_rx_req	UART4 接收方向请求



外设 ID	外设信号	描述
12	I2C2_dma_tx_req	I2C2 发送方向请求
13	I2C2_dma_rx_req	I2C2 接收方向请求
14	I2C3_dma_tx_req	I2C3 发送方向请求
15	I2C3_dma_rx_req	I2C3 接收方向请求
16	I2C4_dma_tx_req	I2C4 发送方向请求
17	I2C4_dma_rx_req	I2C4 接收方向请求
18	I2C5_dma_tx_req	I2C5 发送方向请求
19	I2C5_dma_rx_req	I2C5 接收方向请求
20	I2C6_dma_tx_req	I2C6 发送方向请求
21	I2C6_dma_rx_req	I2C6 接收方向请求
22	I2C7_dma_tx_req	I2C7 发送方向请求
23	I2C7_dma_rx_req	I2C7 接收方向请求
24	I2C8_dma_tx_req	I2C8 发送方向请求
25	I2C8_dma_rx_req	I2C8 接收方向请求
26	I2C9_dma_tx_req	I2C9 发送方向请求
27	I2C9_dma_rx_req	I2C9 接收方向请求
28	I2C10_dma_tx_req	I2C10 发送方向请求
29	I2C10_dma_rx_req	I2C10 接收方向请求
30	I2C11_dma_tx_req	I2C11 发送方向请求
31	I2C11_dma_rx_req	I2C11 接收方向请求

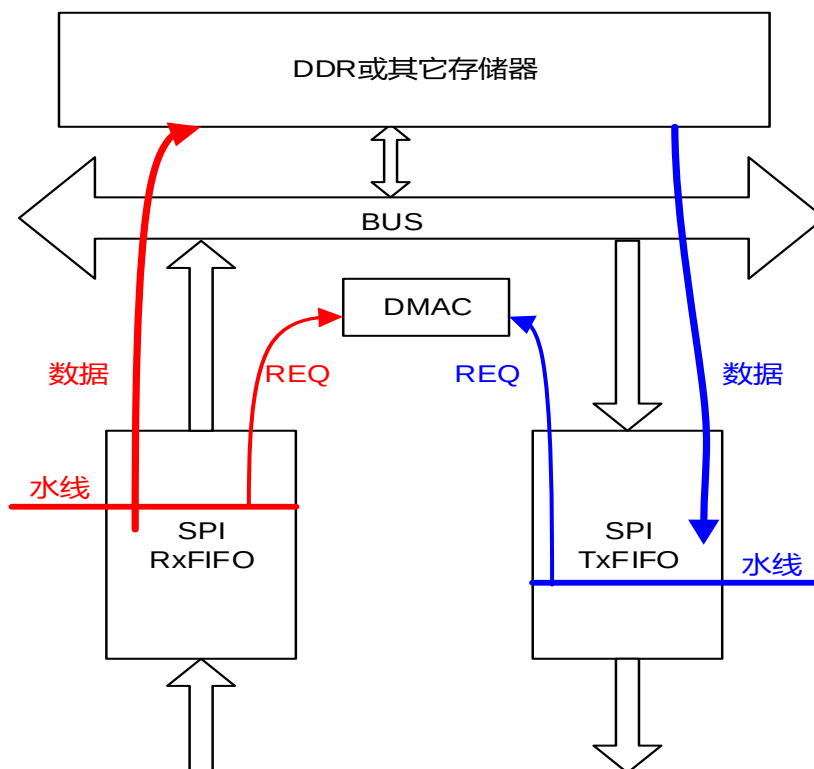
30.4.4 普通外设请求

30.4.4.1 请求产生机理

普通外设包含 SPI、I2C、UART，其内部和 DMA 模块相连只有 REQ 请求信号，没有 LREQ 请求信号。

普通外设请求产生机理如图 30-1 所示（以 SPI 为例）。

图30-1 普通外设请求产生机理



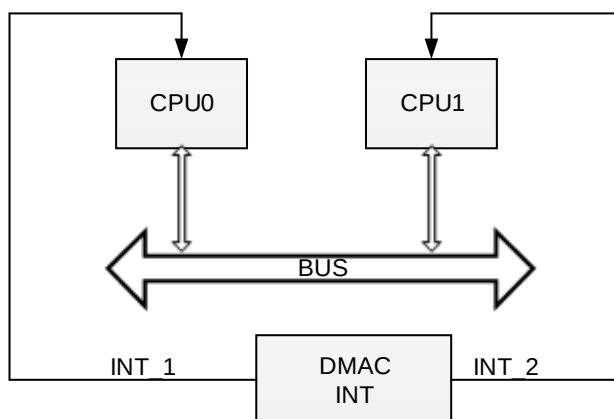
TX 方向，当 Tx FIFO 发送数据低于水线时，SPI 拉高 REQ，请求 DMAC 启动从 DDR 或其他存储器（由 DMAC 通道源地址决定）向 SPI Tx FIFO 搬移数据；当 Tx FIFO 发送数据不低于水线时，SPI 拉低 REQ。

RX 方向，当 Rx FIFO 接收数据高于水线时，SPI 拉高 REQ，请求 DMAC 从 SPI Rx FIFO 向 DDR 或其他存储器（由 DMAC 通道目标地址决定）搬移数据；当 Rx FIFO 接收数据不高于水线时，SPI 拉低 REQ。

30.4.5 中断

30.4.5.1 中断产生

图30-2 DMAC 中断产生示意图



- 芯片提供 1 个 DMAC 模块，16 个通道相互独立，每个通道可用于一种单向传输。
- 为方便多核间独立使用 DMAC 通道资源，芯片提供三组中断，每组中断可以对应到一个处理器上，对于多核 ARM，只需要将其中一组中断送给多核中断控制器 GIC 即可，寄存器见中断控制逻辑章节。Hi3093 中把 3 组中断全部连接给 A55、协处理器 M3 和安全处理器 M3。具体中断分配请参见 4 GIC 章节。
- 支持 DMA 错误中断请求和 DMA 传输完成中断请求，也支持两种中断组合的请求。DMA 错误中断请求是在传输过程中出现了错误发出的。
- 中断产生的原因如下：
 - DMA 传输出错：

DMA 传输过程中出错，总线 INTERCONNECT 或者 SLAVE 设备发出 SLVERR 或者 DECERR 响应时，DMAC 将向 CPU 发出 DMA 错误中断请求。
 - DMA 传输结束：

当 DMAC 作流控时，DMAC 中的计数器在传输过程中将自动作递减，当递减至 0 时表明一个链表节点传输完成，当最后一个链表节点计数递减至 0 时表明本次传输结束，置传输结束中断。



- 中断可配置为是否屏蔽。可在 DMAC_ITC_MASK_INT_m 和 DMAC_IE_MASK_INT_m (m=1~3) 寄存器对 16 个通道的中断屏蔽直接设置，3 组中断屏蔽信号可由对应的 3 个 CPU 配置。
- 可支持链表中的每个节点传输完成后是否向 CPU 发出中断，通过通道（链表）级配置寄存器 DMAC_CmCONFIG[itc_inten]进行配置，通道（链表）下全部节点共用同一配置，n 个链表节点传输完成总共上报 n 次中断。

30.4.5.2 中断查询

通过 DMAC 寄存器，可以查询各通道传输状态，在中断程序中可以通过逐通道查询获得通道中断类型或状态。

30.5 工作方式

30.5.1 初始化

初始化 DMAC 步骤如下：

步骤 1 配置 DMAC_G_EN 的 bit[1]设置通道优先级，bit[0]=1 使能 DMAC。

步骤 2 配置寄存器 DMAC_ITC_MASK_INT_x、DMAC_IE_MASK_INT_x (x=1~3) 均为 0x0000_0000 屏蔽中断。

步骤 3 配置寄存器 DMAC_INT_TC_CLR 和 DMAC_INT_ERR_CLR 均为 0x0000_FFFF，清除所有中断状态。

----结束

30.5.2 启动通道

DMAC 初始化完成之后，需要配置并启动 DMAC 通道后才可以进行 DMA 数据传输。
DMAC 通道配置和启动步骤如下：



须知

- 配置时保证链表各节点均满足： $\text{swidth} \times \text{sburst_length} = \text{dwidth} \times \text{dburst_length}$ ，考虑到软件设计效率和复杂度，在搬移数据足够多的情况下，burst 取值建议按照如下算法：如果 $\text{swidth} = \text{dwidth}$ ，则 $\text{sburst_length} = \text{dburst_length} = 16$ ；如果 $\text{swidth} < \text{dwidth}$ ，则 $\text{sburst_length} = 16$ ， $\text{dburst_length} = \text{swidth} \times \text{sburst_length} / \text{dwidth}$ ；如果 $\text{swidth} > \text{dwidth}$ 同理 $\text{dburst_length} = 16$ ， sburst_length 根据公式计算获得，burst 实际寄存器配置需要在上述计算值的基础上减 1。
- DMA 作为控制器，每个节点传输长度 transferSize（单位 beat，即源数据宽度）必须能被 sburst_length 整除；建议 transferSize 最大长度限定为 0x7FFF0（小于芯片寄存器支持的 0x7FFFF），从而在所有 sburst_length 取值下，transferSize 均能被 sburst_length 整除。
- 对于源设备待搬运的地址空间，不同通道之间必须都小于 4GB，或都大于 4GB；对于目的设备待写入的地址空间，不同通道之间必须都小于 4GB，或都大于 4GB。

- 步骤 1 读 DMAC_EN_CH_STATUS 寄存器，找出处于非传输状态的通道，并从中选择一个通道用于配置。
- 步骤 2 向 DMAC_INT_TC_CLR 和 DMAC_INT_ERR_CLR 的对应位写 1，清除选定通道的中断状态。
- 步骤 3 写通道寄存器 DMAC_CmSRCADDR，设置源设备访问首地址。
- 步骤 4 写通道寄存器 DMAC_CmDESTADDR，设置目的设备访问首地址。
- 步骤 5 如果配置通道用于单块数据传输，则将通道寄存器 DMAC_CmLLI 置为 0。
- 步骤 6 写寄存器 DMAC_CmCONTROL，设置访问源/目的设备所采用的 Master、源目的设备的位宽、Burst size、地址递增以及 Transfer size 等参数。
- 步骤 7 写寄存器 DMAC_CmCONTROL，设置本通道的 DMA 请求信号、流控方式及中断屏蔽。向 DMAC_CmCONFIG bit[0]写 0，暂不启动该通道。
- 步骤 8 向寄存器 DMAC_CmCONFIG bit[0]写 1，启动该通道。

----结束



30.5.3 中断处理

DMAC 通道配置启动传输完成之后或传输过程中出现错误，都会上报中断给中断控制器。中断程序的处理步骤如下：

- 步骤 1 读中断状态寄存器 INT_x(物理地址为 0x08714000~0x08714038)，找出发出中断请求的通道。
- 步骤 2 根据 DMAC_CmCONFIG bit[31]配置，查询中断类型（传输错误、链表节点传输完成后、还是整个链表传输完成）。
- 步骤 3 写寄存器 DMAC_INT_ERR_CLR、DMAC_INT_TC_CLR 通道对应位为 1，清除对应通道的中断状态。
- 步骤 4 退出中断服务程序。

----结束

30.5.4 时钟门控

DMAC 模块长时间不用，可以关闭门控，通过配置系统控制器寄存器实现。

30.5.4.1 打开门控时钟

具体配置步骤如下：

- 步骤 1 读系统控制器寄存器 LPSC_CLKSTA_CTRL1 bit[16]，获取当前时钟门控信息。
- 步骤 2 如果 LPSC_CLKSTA_CTRL1 bit[16]为 0，进入[步骤 3](#)。LPSC_CLKSTA_CTRL1 bit[16]为 1，表示 DMAC 时钟已经打开，直接跳过下面的步骤结束操作。
- 步骤 3 配置 LPSC_CLKEN_CTRL1 bit[16]为 1，打开 DMAC 模块的时钟门控。

----结束

30.5.4.2 关闭门控时钟

具体配置步骤如下：

- 步骤 1 读系统控制器寄存器 LPSC_CLKSTA_CTRL1 bit[16]，获取当前时钟门控信息。
- 步骤 2 如果 LPSC_CLKSTA_CTRL1 bit[16]为 0，则表示 DMAC 时钟已经关闭，直接跳过下面的步骤结束操作。



步骤 3 配置 LPSC_CLKDIS_CTRL1 bit[16]为 1，关闭 DMAC 模块的时钟门控。

----结束

30.5.5 UART 在 DMA 方式下的数据传输

说明

以 UART2、I2C0、DMA 以通道 0 为例，物理地址标注为通道 0。

30.5.5.1 初始化

步骤 1 配置线性控制寄存器 UART_LCR bit[7]为 1，使 UART_DLL 与 UART_DLH 可访问。

步骤 2 写相应的配置值到低位分频因子寄存 UART_DLL 和高位分频因子寄存器 UART_DLH，配置传输速率。

步骤 3 配置寄存器 UART_LCR bit[6:0]，设定 UART 的工作模式。

步骤 4 配置寄存器 UART_FCR，使能或者关闭 FIFO。如果使能 FIFO，还需要设定相应的发送及接收 FIFO 水线值。

----结束

30.5.5.2 数据发送

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

步骤 2 通过 DMAC 通道当前源地址（寄存器 DMAC_CmSRCADDR）减去初始源地址计算已经完成搬移的数据数，判断数据是否发送完成。

----结束

30.5.5.3 数据接收

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

步骤 2 通过 DMAC 通道当前目标地址（寄存器 DMAC_CmDESTADDR）减去初始目标地址计算已经完成搬移的数据数，判断数据是否接收完成。

----结束



30.5.6 SPI 在 DMA 方式下的数据传输

30.5.6.1 数据发送

- 步骤 1 配置寄存器 SPI_IMR 为 0，禁止产生中断。
- 步骤 2 配置 SPI_DMCCR bit[1]为 1，使能 SPI 的 DMA 发送功能。
- 步骤 3 获取一个 DMAC 的通道。
- 步骤 4 配置这个 DMAC 通道的配置寄存器和控制寄存器中的相关参数。
- 步骤 5 启动 DMAC_CmCONFIG bit[0]为 1，以响应 SPI 的 DMA 请求。
- 步骤 6 向 SPI_SSIENR bit[0]写 1，使能 SPI，完成初始化配置。
- 步骤 7 通过 DMAC 通道当前源地址（寄存器 DMAC_CmSRCADDR）减去初始源地址计算已经完成搬移的数据数，如果完成搬移则向 SPI_DMCCR bit[0]写 0，关闭 SPI 的 DMA 发送功能；否则继续等待。
- 步骤 8 向 SPI_SSIENR bit[0]写 0，关闭 SPI。

----结束

30.5.6.2 数据接收

- 步骤 1 配置 SPI_IMR 为 0，禁止产生中断。
- 步骤 2 向 SPI_DMCCR bit[0]写 1，使能 SPI 的 DMA 接收功能。
- 步骤 3 获取一个 DMAC 通道。
- 步骤 4 配置该 DMAC 通道的配置寄存器和控制寄存器中的相关参数。
- 步骤 5 启动 DMAC，以响应 SPI 的 DMA 请求。
- 步骤 6 向 SPI_SSIENR bit[0]写 1，使能 SPI，完成初始化配置。
- 步骤 7 通过 DMAC 通道当前目标地址（寄存器 DMAC_CmDESTADDR）减去初始目标地址计算已经完成搬移的数据数，如果完成搬移则向 SPI_DMCCR bit[0]写 0，关闭 SPI 的 DMA 接收功能；否则继续搬移。
- 步骤 8 向 SPI_SSIENR 写 0，关闭 SPI。

----结束



30.5.7 I2C 在 DMA 方式下的数据传输

30.5.7.1 初始化

- 步骤 1 设定超时标志，检测 I2C_STATUS[0]（判断 I2C 控制器是否空闲），如果是 1，则延时等待，继续检测，直到 I2C_STATUS bit[0]变为 0 或者超时，然后向 I2C_ENABLE bit[0]写 0，使 I2C 处于关闭状态。
- 步骤 2 写相应的配置值到 I2C_CON，配置主模式以及传输速率模式等参数。
- 步骤 3 将对接器件的从设备地址写入 I2C_TAR bit[9:0]。
- 步骤 4 配置 I2C_SS_SCL_HCNT、I2C_SS_SCL_LCNT 等寄存器，设定相应的 I2C 总线时钟周期。
- 步骤 5 配置 I2C_DMA_TDLR、I2C_DMA_RDLR，设置 DMA 发送 FIFO 及接收 FIFO 阈值。
- 步骤 6 向 I2C_ENABLE bit[0]写 1，使能 I2C。

----结束

30.5.7.2 数据发送

- 步骤 1 配置 DMA 数据通道，包括数据传输源和目的地址、传输类型等参数。
- 步骤 2 向 I2C_DMA_CR bit[1]写 1，使能 I2C 的 DMA 发送功能。
- 步骤 3 读 I2C_INTR_STAT bit[4]寄存器（发送 FIFO 低于阈值），查询中断状态，判断数据是否发送完成。如果完成则向 I2C_DMA_CR bit[1]写 0，禁止 I2C 的 DMA 发送功能。

----结束

30.5.7.3 数据接收

数据接收包括数据发送和数据接收两步，首先将读命令发送给 I2C 控制器，然后从接收 FIFO 中读取接收数据，具体配置步骤如下：

- 步骤 1 配置 DMA 数据通道，包括数据传输源（传输源内容包括读命令 0x100 等）和目的地址、数据接收区地址、数据传输个数、传输类型等参数。
- 步骤 2 配置 I2C_DMA_CR bit[1:0]为 2'b11，使能 I2C 的 DMA 发送和接收功能（因数据接收时要先发送 0x100 到 I2C_DATA_CMD 寄存器，所以要使能 DMA 发送功能）。



步骤 3 读 I2C_INTR_STAT bit[8]寄存器，查询中断状态，判断数据是否接收完成。如果完成则向 I2C_DMA_CR bit[1:0]写 1'b00，禁止 I2C 的 DMA 发送和接收功能。

----结束

30.6 DFX

当发现 DMAC 没有完成，有报错时，需要先确认是否满足下面关系：

DMAC 要求数据宽度和 burst 长度严格满足如下关系：

$swidth * sburst_length = dwidth * dburst_length$ ，错误配置可能导致 DMAC 模块挂死。



31 TIMER

31.1 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。

Hi3093 提供 3 个 Dual-Timer 模块，Dual-Timer0 ~ 2 包含两个功能完全相同的 64bitTimer。

Dual-Timer0 包括 Timer0、Timer1，它们共用同一个基地址（Timer0 和 Timer1 两个计数器在同一个模块中实现，模块寄存器的基地址相同，偏移地址不同），中断线独立。

Dual-Timer1 包括 Timer2、Timer3，它们共用同一个基地址（Timer2 和 Timer3 两个计数器在同一个模块中实现，模块寄存器的基地址相同，偏移地址不同），中断线独立。

Dual-Timer2 包括 Timer4、Timer5，它们共用同一个基地址（Timer4 和 Timer5 两个计数器在同一个模块中实现，模块寄存器的基地址相同，偏移地址不同），中断线独立。

31.2 特性

- 2 个带可编程 8 位预分频器的 64bit/32bit 减法定时器/计数器。
- 计数时钟是 100MHz 总线时钟。
- 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` ($x=0 \sim 7$) 和 `TIMERx_BGLOAD` 寄存器实现。



- 当前的计数值可随时读取。
- 当计数值减到 0 的当拍会产生中断。
- 支持 APB 总线接口。
- 支持 6 个 32bit 计数或 6 个 64bit 计数。
- 支持 1 个 timer 独立给安全核 M3 使用。

31.3 关键术语

表31-1 TIMER 关键术语描述

术语	描述
Timer	计时器。

31.4 功能

31.4.1 TICK

64/32bitTimer 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。

31.4.2 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

- 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。
- 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。
- 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。

31.4.3 计数范围和计数精度

Timer 计数范围和计数精度如表 31-2 所示。

表31-2 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
100MHz	32bitTimer	10ns	0~42s
	64bitTimer		0~5852 年（一年按照 365 天计算）

31.4.4 TIMER 计数模式

64bitTimer 基于一个 64bit/32bit（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。当计数值递减到零，Timer 将产生一个中断。

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 0 时又自动回转到其最大值，并继续计数。当计数长度为 32bits 时，最大值为 0xFFFF_FFFF。当计数长度为 64bit 时，最大值为 0xFFFF_FFFF_FFFF_FFFF。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 0 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 0 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 0 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（该分频的配置参考 Timer0_CONTROL 寄存器的 timerpre 域段）。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 0。然后载入 TIMERx_BGLOAD 寄存器中的新值开始计数。

31.5 工作方式

31.5.1 初始化

步骤 1 配置寄存器 Timer0_LOAD_H，为 Timer0 载入计数初值的高 32bit 值。

步骤 2 配置寄存器 Timer0_LOAD_L，为 Timer0 载入计数初值的低 32bit 值。必须先配置高 32 位，再配置低 32 位。

步骤 3 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 Timer0_BGLOAD_L 寄存器和 Timer0_BGLOAD_H，设置 Timer 的计数周期。否则跳过此步骤，进入步骤 3。

步骤 4 写系统控制器 LPSC_CLKEN_CTRL0 bit[2]为 1，配置 Timer 的时钟使能信号。

步骤 5 写寄存器 Timer0_CONTROL，配置 Timer 的计数模式、计数器长度、预分频因子及中断屏蔽，同时启动 Timer 计数。

----结束



31.5.2 中断处理

Timer 主要用于定时产生中断，因此 Timer 的中断处理主要是激活等待定时中断的进程。操作步骤如下：

步骤 1 配置 Timer0_INTCLR 寄存器，清除 Timer 中断。

步骤 2 激活等待该中断的进程，使其继续执行。

步骤 3 当所有等待该中断的进程完成或再次执行到等待中断的休眠状态时，恢复中断现场，继续执行当前被中断的程序。

----结束



32 WatchDog

32.1 概述

WatchDog (看门狗) 用于系统异常情况下, 一定时间内发出复位信号, 复位对应的复位域请参考 [7 复位](#) 章节。

32.2 特性

- 内部具有一个 32bit 减法计数器。
- 支持超时时间间隔 (即计数初值) 可配置。
- 支持锁定寄存器来防止寄存器被误改。
- 支持时钟使能, 可控制计数时间间隔。
- 支持超时中断产生, 若第一次中断未被响应, 第二次超时则产生复位信号。
- 如果 WatchDog 挂死, 输出指示信号到芯片外。
- 复位指示信号到实际复位需做延时处理, 延时从 0ms~10s 可配 (0ms, 10ms, 100ms, 300ms, 500ms, 1s, 2s, 4s, 10s), 默认延时 300ms。
- 片内集成 7 个 WatchDog (4 个 A55 核、业务 M3、安全 M3 分别使用 1 个, 1 个预留片外控制使用); 其中片外控制的 WatchDog 复位信号接出到片外, 芯片复位由片外控制。
- 和 Core 对应的 Watchdog 溢出, 输出复位指示, 复位整 CHIP 域、整协处理模式域或协处理模式非安全域可配, 指示信号可屏蔽。
- 用于片外控制的 WatchDog 溢出, 输出复位指示到芯片管脚, 指示信号可屏蔽; 其他 6 个 WatchDog 也可配置通过该管脚输出。
- 支持中断屏蔽功能。



- 支持时钟门控，以降低系统功耗。

32.3 关键术语

表32-1 WATCHDOG 关键术语描述

术语	描述
WATCHDOG	看门狗

32.4 功能

WatchDog 使用晶振时钟（25MHz），当 WatchDog 超时，将产生复位信号直接复位包括芯片在内的相关系统。

WatchDog 的运行基于 1 个 32bit 减法计数器，计数初值由寄存器 WDG_LOAD 载入。在 WatchDog 时钟使能情况下，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 0 的当拍，WatchDog 将产生一个中断。在下一个计数时钟上升沿，计数器又从寄存器 WDG_LOAD 中重新载入计数初值，开始递减计数。

如果计数器第二次计数递减到 0 的当拍，CPU 还没有清除 WatchDog 中断，则 WatchDog 将发出复位信号 WDG_RST，计数器停止计数。

根据实际应用需要，可通过配置 WDG_CONTROL 使能或者禁止 WatchDog 产生中断和复位信号：

- 当禁止产生中断时，计数器将停止计数。
- 当重新开启中断时，WatchDog 将从 WDG_LOAD 的设定值开始计数，而不是从计数器上次停止时的计数值开始计数。在中断到来之前，可以重新载入初值。

通过配置 WDG_LOCK 寄存器，可以禁止对 WatchDog 内部寄存器进行写操作：

- 向 WDG_LOCK 写入 0x1ACC_E551，可以打开所有 WatchDog 寄存器的写权限。
- 向 WDG_LOCK 寄存器写入其他任何值，可以关闭所有 WatchDog 寄存器（WDG_LOCK 寄存器除外）的写权限。



该特性保护 WatchDog 的寄存器不被软件错误地修改，从而使得在异常情况下，WatchDog 不致被软件错误地中止操作。

WatchDog 计数时间为： $T_{WDG} = \text{Value}_{WDG_LOAD} \times (1/f_{clk})$

- T_{WDG} 表示 WatchDog 计数时间。
- Value_{WDG_LOAD} 表示 WatchDog 计数初值。
- f_{clk} 表示 WatchDog 计数时钟频率。

WatchDog 在 25MHz，不同配置下的计数时间范围值：0~172s。

32.5 工作方式

32.5.1 时钟门控

WatchDog 模块长时间不用，可以关闭门控，通过配置系统控制器寄存器实现。以下以 Wathdog0 为例说明。

32.5.1.1 打开门控时钟

- 步骤 1 读系统控制器寄存器 LPSC_CLKSTA_CTRL1，获取当前时钟门控信息。
- 步骤 2 若 LPSC_CLKSTA_CTRL1 bit[17]为 0，则进入[步骤 3](#)。LPSC_CLKSTA_CTRL1 bit[17]为 1，则表示 WatchDog 控制器时钟已经打开，直接跳过下面的步骤，结束操作。
- 步骤 3 配置 LPSC_CLKEN_CTRL1 bit[17]为 1，即打开 WatchDog 控制器的时钟门控。

----结束

32.5.1.2 关闭门控时钟

- 步骤 1 读系统控制器寄存器 LPSC_CLKSTA_CTRL1，获取当前时钟门控信息。
- 步骤 2 如果 LPSC_CLKSTA_CTRL1 bit[17]为 1，则进入[步骤 3](#)；如果 LPSC_CLKSTA_CTRL1 bit[17]为 0，则表示 WatchDog 控制器时钟已经关闭，直接跳过下面的步骤，结束操作。
- 步骤 3 配置 LPSC_CLKDIS_CTRL1 bit[17]为 1，即关闭 WatchDog 控制器的时钟门控。



----结束

32.5.2 中断处理过程

收到 WatchDog 发出的中断后，应及时清除其中断状态，并使其载入计数初值重新开始计数。

WatchDog 中断处理的过程如下所示：

- 步骤 1 向锁定寄存器 WDG_LOCK 写 0x1ACC_E551，为 WatchDog 开锁。
- 步骤 2 向寄存器 WDG_INTCLR 写任意值，清除 WatchDog 的中断状态，同时也使 WatchDog 自动载入计数初值重新开始计数。
- 步骤 3 向寄存器 WDG_LOCK 写入 0x1ACC_E551 以外的任何值，给 WatchDog 上锁。

----结束

须知

对于中断使能的配置，要在产生中断前就配置好。

若中断先前被禁止，则当中断再次被使能时，计数器从 WDG_LOAD 中载入初值并重新开始计数。

WDG_CONTROL[inten]不仅控制中断屏蔽，也控制 WatchDog 计数器的启动/停止计数。WatchDog 发出中断时，将 INTEN 位清零只会屏蔽将要发生的 WatchDog 中断，但不会使现有的中断状态清零。当再次将 INTEN 置位时，虽然 WatchDog 的计数器载入初值重新开始计数，但 Watchdog 的中断输出 WDOGINT 会重新有效。所以应注意，清除中断只能通过写 WDG_INTCLR 寄存器来完成。

32.5.3 关闭 WatchDog

在系统进入 SLEEP 模式之前，必须关闭 WatchDog。向 WDG_CONTROL bit[0]写 1'b0，关闭 WatchDog；向该位写 1'b1，打开 WatchDog。

32.5.4 注意事项

使用 WatchDog 的注意事项如下：

- 一旦 Watchdog 的复位信号输出 WDGRST 有效后，就会一直保持有效，直到系统使用 BMC_UNSE_RST0_N 复位信号将 Watchdog 的内部寄存器清零，



WDGRST 的复位才会取消。因此系统的复位生成电路在接收到 WDGRST 信号后应发出 BMC_UNSE_RST0_N 复位信号将 Watchdog 复位，否则系统无法退出复位状态。另外，WatchDog 被复位的同时，系统也完成了复位，此时系统初始化程序应该重新对 WatchDog 进行初始化。

- 使用 WatchDog 软复位也可将 WatchDog 复位。
- 下面 4 种情况下 WatchDog 重新载入计数初值：
 - 写 WDG_LOAD 寄存器时；
 - WatchDog 的计数值第一次减到 0 时；
 - 写 WDG_INTCLR 寄存器时；
 - 写寄存器 WDG_CONTROL，使其 INTEN 位由无效转为有效时。
- 如果在 WatchDog 的 WDOGCLK 上升沿或时钟使能 WDOGCLKEN 到来之前，多次写 WDG_LOAD 寄存器，则 WatchDog 的计数器实际载入的值为最后一次写入 WDG_LOAD 寄存器中的值。



33 JLC

33.1 概述

加载控制器 JLC (JTAG Loader Controller) 是一个 JTAG 主控制器，接口符合 IEEE Std 1149.1-2001 标准。利用该设备，可以方便地控制符合 IEEE Std 1149.1-2001 标准的 JTAG 器件，完成 FLASH 加载、CPLD 加载、板级测试等功能。

33.2 特性

- 支持双缓存，缓存深度 128，链长最大 128。
- 支持 TMS、TDI 并行或串行发送操作。
- 支持 TCK 频率灵活配置，最大可达 25MHz。
- 支持自动、手动 TMS 模式。
- 芯片复位时所有输出高阻。
- 支持对输入 TDO 进行延迟采样。
- 支持 2 路 JTAG master 接口，用于板载 CPLD，双硬盘背板 CPLD 的升级。
- 其中 1 路 JTAG Master/Slave 切换可以配，默认模式取决于管脚 STRAP[8]配置。



33.3 关键术语

表33-1 JLC 关键术语描述

术语	描述
JLC	加载控制器 JLC 是一个 JTAG 主控制器，接口符合 IEEE Std 1149.1-2001 标准。利用该设备，可以方便地控制符合 IEEE Std 1149.1-2001 标准的 JTAG 器件，完成 FLASH 加载、CPLD 加载、板级测试等功能。

33.4 功能

在 Hi3093 芯片中，有 2 个 JLC 模块（JLC0 和 JLC1），作为 JTAG Master，用于板载 CPLD，双硬盘背板 CPLD 的升级。其中 JLC0 的信号，通过寄存器可以实现 JTAG Slave 功能；寄存器的配置可以参考 [43 JTAG](#) 章节。

JLC 有 2 种主要工作模式，通过寄存器 JLC_CR bit[0]控制：

- JLC_CR bit[0]=0x0，JLC 工作于模式 0。
- JLC_CR bit[0]=0x1，JLC 工作于模式 1。

33.4.1 模式 0

JLC 工作于模式 0 时，TDI 和 TMS 分别通过寄存器 JLC_TDI_DATA（TDI 发送数据寄存器）和 JLC_TMS_DATA（TMS 发送数据寄存器）发送，收到的 TDO 数据存放在寄存器 JLC_TDO_DATA（TDO 接收数据寄存器）。

在寄存器 JLC_TDI_DATA（TDI 发送数据寄存器）和 JLC_TMS_DATA（TMS 发送数据寄存器）都不为空时（JLC_STAT bit[1]=0x0 且 JLC_STAT bit[3]=0x0），数据开始发送，且 TDI 和 TMS 总是同时逐位发送。从管脚 DO 上接收到的 TDO 数据保存在寄存器 JLC_TDO_DATA（TDO 接收数据寄存器）。

模式 0 时，TDI 和 TMS 的数据位宽可以设置为 8bit 或 32bit。当设置为 8bit 时，只发送 JLC_TDI_DATA 和 JLC_TMS_DATA 的低 8bit，当设置为 32bit 时，JLC_TDI_DATA 和 JLC_TMS_DATA 的所有 32bit 都会发送。

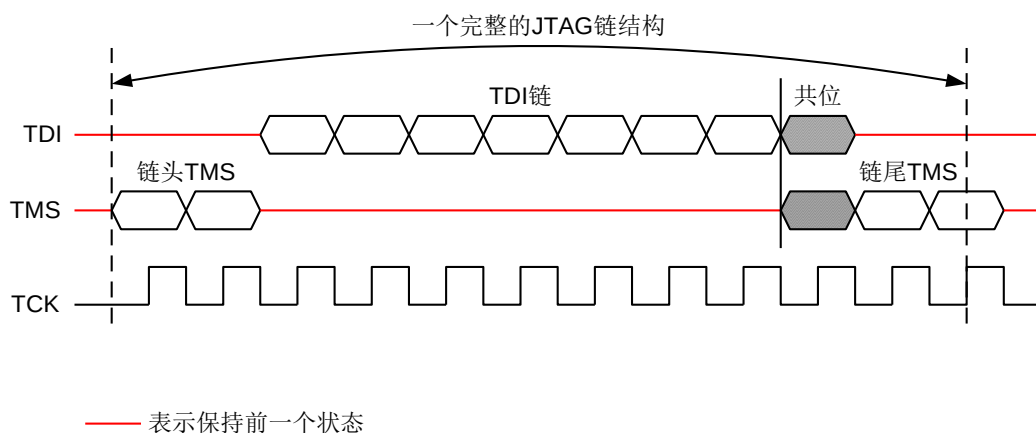
33.4.2 模式 1

JLC 工作于模式 1 时，TMS 和 TDI 数据分时发送。在发送 TDI 时，TMS 保持最后一个发送状态，在发送 TMS 时，TDI 亦保持最后一个发送状态，如图 33-1 所示。

在模式 1，根据链头、链尾 TMS 是否自动发送，又可以分成自动 TMS、手动 TMS 两种模式（用寄存器 JLC_CR bit[3]设置）。

在此模式下，JLC 可以使用缓存 1、缓存 2 分别发送 TDI 数据。

图33-1 串行 TDI&TMS 时 JTAG 链结构



33.4.2.1 自动 TMS 模式

自动 TMS 模式时，仅使用缓存发送 TDI 数据：

- 使用缓存 1 时，一条链发送的数据依次为：链头 TMS (JLC_TMS_DATA)、TDI 数据 (缓存 1)、链尾 TDI (JLC_TDI2_DATA)、链尾 TMS (JLC_TMS2_DATA)。
- 使用缓存 2 时，一条链发送的数据依次为：链头 TMS (JLC_TMS_DATA)、TDI 数据 (缓存 2)、链尾 TDI (JLC_TDI2_DATA2)、链尾 TMS (JLC_TMS2_DATA)。

说明

缓存 1 链和缓存 2 链使用相同的 TMS 数据和不同的 TDI 数据。

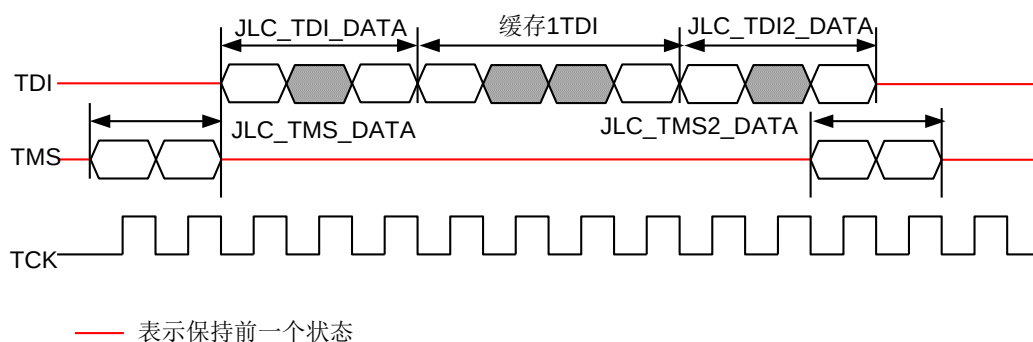
33.4.2.2 手动 TMS 模式

手动 TMS 模式时，除了寄存器 JLC_TDI2_DATA2 外，所有的数据寄存器、缓存均可以用来发送数据（TMS、TDI 或 TCK），但仅在写操作时启动数据发送。写多个数据寄存器时，JLC 会自动按照写操作的顺序对发送的数据排序，保证先写入的数据首先被发送。

例如：用户写寄存器的顺序为（在写操作时，各个数据寄存器都为空）：

JLC_TMS_DATA→JLC_TDI_DATA→JLC_START bit[1]（置位）→JLC_TMS2_DATA→JLC_TDI2_DATA。JLC 会按照图 33-2 顺序发送 TMS 和 TDI 数据。在发送 TDI 数据的同时，从 DO 管脚上收到的串行 TDO 数据会和发送的 TDI 保存在同一位置。

图33-2 手动发送模式



33.4.3 延迟采样 TDO

延迟采样配置步骤如下：

- 步骤 1 配置 JLC_ENABLE（JLC 使能寄存器）为 0 关闭 JLC 模块。
- 步骤 2 配置 JLC_SAM_DLY（延迟检测配置寄存器）为所需要的延迟参数。
- 步骤 3 配置 JLC_DLY_EN 为 1 使能延迟采样。

----结束



须知

JLC_SAM_DLY 寄存器的配置值需要与 JLC_TCK_RATE 值相匹配，最大支持 2 个 TCK 时钟周期。当 JLC_SAM_DLY 寄存器配置的值大于 2 个 TCK 时钟周期时，会自动修改为 2 个 TCK 时钟周期。

33.5 工作方式

33.5.1 初始化

JTAG 有 2 路。

使用 strap 管脚和寄存器进行第一路 JTAG MASTER 和 SLAVE 的切换，以及 MUX 的切换。寄存器地址是 0x08740208。使用 strap 管脚和寄存器进行 MUX 的切换。具体含义可以参考 SysCtrl 模块寄存器表单。

33.5.2 软复位

步骤 1 通过配置 LP_SUBSYS 模块寄存器 LPSC_SRST_REQ1 的 bit[30]和 bit[31]为 1，可实现对 JLC 工作逻辑的单独软复位。

步骤 2 延时 2us。

步骤 3 配置 LP_SUBSYS 模块寄存器 LPSC_SRST_REQ1 的 bit[30]和 bit[31]为 0，可实现对 JLC 工作逻辑的单独软复位撤离。

----结束

33.5.3 时钟门控

JLC 模块长时间不用，可以关闭时钟门控，通过配置系统控制器寄存器实现。以下以 JLC0 为例说明。

33.5.3.1 打开门控时钟

步骤 1 读系统控制器寄存器 LPSC_CLKSTA_CTRL0，获取当前时钟门控信息。

步骤 2 若 LPSC_CLKSTA_CTRL0 bit[0]为 0，则进入步骤 3；LPSC_CLKSTA_CTRL0 bit[0]为 1，则表示 JLC0 控制器时钟已经打开，直接跳过下面的步骤，结束操作。



步骤 3 配置 LPSC_CLKEN_CTRL0 bit[0]为 1，即打开 JLC0 控制器的时钟门控。

----结束

33.5.3.2 关闭门控时钟

步骤 1 读系统控制器寄存器 LPSC_CLKSTA_CTRL0，获取当前时钟门控信息。

步骤 2 如果 LPSC_CLKSTA_CTRL0 bit[0]为 1，则进入[步骤 3](#)；如果 LPSC_CLKSTA_CTRL0 bit[0]为 0，则表示 JLC0 控制器时钟已经关闭，直接跳过下面的步骤，结束操作。

步骤 3 配置 LPSC_CLKDIS_CTRL0 bit[0]为 1，即关闭 JLC0 控制器模块的时钟门控。

----结束

33.5.4 模式 0

JLC 模式 0 使用步骤如下：

步骤 1 往 JLC_ENBALE bit[0]写 0x0，关闭 JLC 模块。

步骤 2 如果需要 JLC_TDI_DATA 和 JLC_TMS_DATA 的数据位宽为 8bit，则往 JLC_CR bit[4:0]写 0x04，如果需要数据位宽为 32bit，则往 JLC_CR bit[4:0]写 0x00。

步骤 3 写 JLC_TCK_RATE bit[6:0]配置 TCK 输出时钟相对 APB 时钟的分频系数。

步骤 4 往 JLC_CH_EN bit[6:0]写 0x01，使能通道 0。

步骤 5 往 JLC_ENBALE bit[0]写 0x1，使能 JLC 模块。

步骤 6 检测 JLC_STAT bit[3]，如果为 1，说明 JLC_TMS_DATA 寄存器空，将需要发送到 TMS 线上的数据写入 JLC_TMS_DATA 中，否则等待 JLC_STAT bit[3]为 1。

步骤 7 检测 JLC_STAT bit[1]，如果为 1，说明 JLC_TDI_DATA 寄存器空，将需要发送到 TDI 线上的数据写入 JLC_TDI_DATA 中，否则等待 JLC_STAT bit[1]为 1。

步骤 8 检测 JLC_STAT bit[2]，如果为 1，说明 JLC_TDO_DATA 寄存器中有有效数据，读取该数据，否则等待 JLC_STAT bit[2]为 1。

步骤 9 重复[步骤 6](#)~[步骤 8](#)直到所有的数据收发完毕。

----结束



须知

- 只有 JLC_TMS_DATA 和 JLC_TDI_DATA 都不为空时才开始发送数据，发送数据的同时从 TDO 线上收到的数据存在 JLC_TDO_DATA 中。
- TDI 和 TMS 同时发送，先发低位。
- APB 时钟为 100M。

33.5.5 模式 1 自动 TMS

JLC 模式 1 自动使用步骤如下：

- 步骤 1 往 JLC_ENBALE bit[0]写 0x0，关闭 JLC 模块。
- 步骤 2 往 JLC_CR bit[4:0]写 0x0B，设置工作模式为模式 1 自动 TMS。
- 步骤 3 写 JLC_TCK_RATE bit[6:0]配置 TCK 输出时钟相对 APB 时钟的分频系数，TCK 输出时钟频率等于 APB 时钟频率除以两倍分频系数。
- 步骤 4 往 JLC_CH_EN bit[6:0]写 0x01，使能通道 0。
- 步骤 5 往 JLC_ENBALE bit[0]写 0x1，使能 JLC 模块。
- 步骤 6 往 JLC_TMS_SIZE bit[4:0]写链头 TMS 长度，0x00 表示有效位数为 32，0x01 ~ 0x1F 分别表示实际有效位数 1 ~ 31（从低位开始）。
- 步骤 7 往 JLC_TMS_DATA 写链头 TMS 数据。
- 步骤 8 写 JLC_BUF_LEVEL 设置 BUF 的深度。
- 步骤 9 把数据写入 BUF 中。
- 步骤 10 写 JLC_TDI_SIZE2 配置链尾 TDI 数据长度。
- 步骤 11 写 JLC_TDI_DATA2 配置链尾 TDI 数据。
- 步骤 12 写 JLC_TMS_SIZE2 配置链尾 TMS 数据长度。
- 步骤 13 写 JLC_TMS_DATA2 配置链尾 TMS 数据。
- 步骤 14 往 JLC_START bit[0]写 1，开始发送。
- 步骤 15 等待 JLC_STAT bit[9:8]等于 0x3，从 JLC_TDO_DATA2 和 BUF 中读出从线路收到的 TDO 数据。
- 步骤 16 重复步骤 6~步骤 15，直到所有的数据都收发完成。



---结束

须知

JLC 有两套 BUF，如果有必要可以乒乓操作，提高效率。BUF2 的使用和 BUF1 相同，和 BUF 相关的寄存器都有两套，具体参见《Hi3093 寄存器手册》。

33.5.6 模式 1 手动 TMS

手动 TMS 模式时，除了寄存器 JLC_TDI2_DATA2 外，所有的数据寄存器、缓存均可以用来发送数据（TMS、TDI），但仅在写操作时启动数据发送。写多个数据寄存器时，JLC 会自动按照写操作的顺序对发送的数据排序，保证先写入的数据首先被发送。

- 步骤 1 往 JLC_ENBALE bit[0]写 0x0，关闭 JLC 模块。
- 步骤 2 往 JLC_CR bit[4:0]写 0x03，设置工作模式为模式 1 手动 TMS。
- 步骤 3 写 JLC_TCK_RATE bit[6:0]配置 TCK 输出时钟相对 APB 时钟的分频系数，TCK 输出时钟频率等于 APB 时钟频率除以两倍分频系数。
- 步骤 4 往 JLC_CH_EN bit[6:0]写 0x01，使能通道 0；
- 步骤 5 往 JLC_ENBALE bit[0]写 0x1，使能 JLC 模块。
- 步骤 6 往 JLC_TMS_SIZE bit[4:0]写链头 TMS 长度，0x00 表示有效位数为 32，0x01 ~ 0x1F 分别表示实际有效位数 1 ~ 31（从低位开始）。
- 步骤 7 往 JLC_TMS_DATA 写链头 TMS 数据。
- 步骤 8 往 JLC_TDI_DATA 写链头 TMS 数据。
- 步骤 9 写 JLC_BUF_LEVEL 设置 BUF 的深度。
- 步骤 10 把数据写入 BUF 中。
- 步骤 11 往 JLC_START bit[0]写 1，开始发送。
- 步骤 12 写 JLC_TDI_SIZE2 配置链尾 TDI 数据长度。
- 步骤 13 写 JLC_TDI_DATA2 配置链尾 TDI 数据。
- 步骤 14 写 JLC_TMS_SIZE2 配置链尾 TMS 数据长度。



步骤 15 写 JLC_TMS_DATA2 配置链尾 TMS 数据。

步骤 16 等待 JLC_STAT bit[9:8]等于 0x3，从 JLC_TDO_DATA2 和 BUF 中读出从线路收到的 TDO 数据。

步骤 17 重复步骤 6~步骤 16，知道所有的数据都收发完成。

----结束

须知

- 步骤 6~步骤 8，先写 TMS 则先发 TMS，先写 TDI 则先发 TDI。
- 步骤 12~步骤 15，在 JLC_TMS_DATA2 和 JLC_TDI_DATA2 中都有数据的时候才会发送。
- 每一个数据寄存器都有状态位，见 JLC_STAT，在非空时无法写入，建议在写任意数据寄存器时，判断其空满状态。

33.5.7 JLC RAM 初始化

步骤 1 以 JLC0 为例，往 JLC_RAM_INI bit[1:0]写 0x3，对 JLC0 的 RAM 进行初始化。

步骤 2 读取 JLC_RAM_INI bit[5:4]，当读取到 JLC_RAM_INI bit[5:4]为 0x3 的时候，说明 JLC0 的 RAM 初始化完成，然后才能进行 JLC0 的模式 0 或模式 1 的工作。否则不能进行工作。

----结束

33.6 DFX

如果 JLC0 升级不成功，优先排查 SYSCTRL 模块的寄存器 JTAG0_Mode_SEL，看当前 JTAG0 是 master 还是 slave。



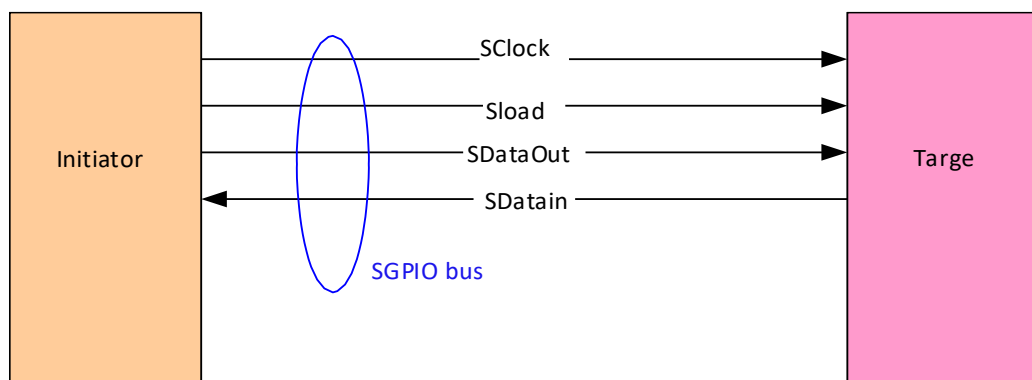
34 SGPIO

34.1 概述

SGPIO 是通用串行输入输出接口，是用来串行化通用 IO 的一种方式，该模式定义了一个 Initiator 和一个 target，通常用在将输出信号输出到并行的 LED 指示信号灯上。

在 Hi3093 中，SGPIO 控制器作为 Initiator，通过 SGPIO 接口控制外接的 target。如图 34-1 所示。一组 SGPIO 总共 4 个管脚：SGPMCK、SGPMLD、SGPMO、SGPMI，信号含义如表 34-2 所示。

图34-1 SGPIO 的应用框图



34.2 特性

- 支持 APB 接口。



- Hi3093 作为 Initiator，支持最大 80 路输入和 80 路输出，可配置为 8/16/24/32/40/48/56/72/80 路。
- 8 bit: SGPIOA
- 16bit: SGPIOA, SGPIOB
- 24bit: SGPIOA, SGPIOB, SGPIOC
- 32bit: SGPIOA, SGPIOB, SGPIOC, SGPIOD
- 40bit: SGPIOA, SGPIOB, SGPIOC, SGPIOD, SGPIOE
- 48bit: SGPIOA, SGPIOB, SGPIOC, SGPIOD, SGPIOE, SGPIOF
- 56bit: SGPIOA, SGPIOB, SGPIOC, SGPIOD, SGPIOE, SGPIOF, SGPIOG
- 64bit: SGPIOA, SGPIOB, SGPIOC, SGPIOD, SGPIOE, SGPIOF, SGPIOG, SGPIOH
- 72bit: SGPIOA, SGPIOB, SGPIOC, SGPIOD, SGPIOE, SGPIOF, SGPIOG, SGPIOH, SGPIOI
- 80bit: SGPIOA, SGPIOB, SGPIOC, SGPIOD, SGPIOE, SGPIOF, SGPIOG, SGPIOH, SGPIOI, SGPIOJ
- SGPIO 时钟基于 APB 时钟分频，分频系数可配，频率支持 1kHz 到 1MHz。
- 数据传输格式按照 normal 格式，数据内容由软件指定。
- 支持中断上报。
- 支持中断屏蔽。
- 支持的中断触发模式：高电平触发，低电平触发，上沿触发，下沿触发。
- 支持 SGPMCK、SGPMLD 取反。
- 每组 4 个管脚，3.3V，实现 2 组，IO 复用。
- 实现 host 侧功能，对接 CPLD，实现 GPIO 扩展。
- 数据内容由软件指定。

须知

Hi3093 的 SGPIO 只支持对接 CPLD，完成 GPIO 的扩展。



34.3 关键术语

表34-1 SGPIO 关键术语描述

术语	描述
SGPIO	Serial General Purpose Input Output

34.4 接口信号

表34-2 SGPIO 接口信号描述

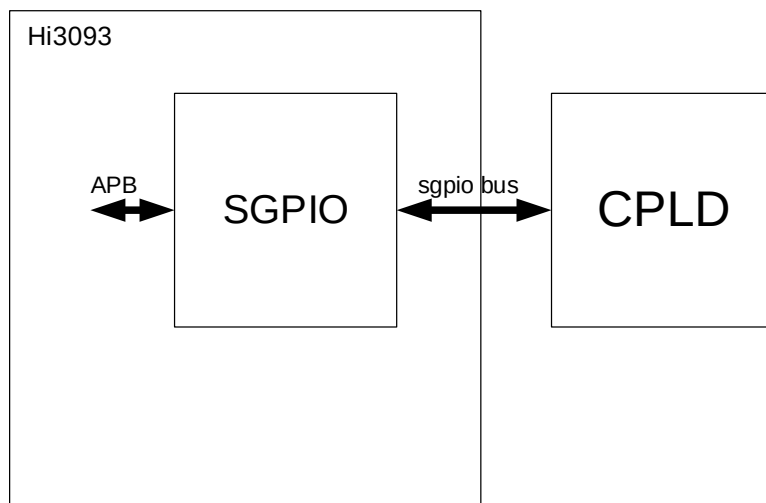
管脚	内部信号名称	方向	描述
SGPMCK	Sgpmck	O	Sgpio master clock
SGPMLD	Sgpmld	O	Sgpio master load
SGPMO	Sgpmo	O	Sgpio master output
SGPMI	Sgpmi	I	Sgpio master input

34.5 功能描述

34.5.1 应用框图

Hi3093 支持 SGPIO Initiator，通过 APB 接口与系统总线互连。

图34-2 SGPIO 结构图



34.6 工作方式

34.6.1 管脚复用配置

参考《Hi3093 寄存器手册》中 IOCONFIG 管脚复用寄存器。

34.6.2 初始化配置

- 步骤 1 配置 SGPMCK 分频参数，寄存器是 SGPIO_CFG bit[31:16]，SGPMCK 的时钟周期等于 apb 时钟周期*2*(SGPIO_CFG bit[31:16] +1)，SGPMCK 频率范围是 1kHz 到 1MHz。
- 步骤 2 配置 SGPIO pin 个数，寄存器是 SPGIO_CFG bit[15:12]，支持配置为 1 到 10，分别对应 1 到 10Byte。
- 步骤 3 配置 SGPMLD 是否取反，寄存器是 SGPIO_CFG bit[8]，0 不取反，1 取反。
- 步骤 4 配置 SGPMCK 是否取反，寄存器是 SGPIO_CFG bit[4]，0 不取反，1 取反。
- 步骤 5 用户如果需要使能中断检测，需执行如下配置，否则可以跳过；配置是否使能 pin 中断检测，对应寄存器 SGPIO_INT_ENB_0~2，1 使能中断，0 关闭中断；配置中断检测类型 1，对应寄存器 SGPIO_INT_SENS_TYPE1_0~2，1 电平触发，0 边沿触发；配置中断检测类型 2，对应寄存器 SGPIO_INT_SENS_TYPE2_0~2，1 上升沿触发或者高电平触发，0 下降沿触发或者低电平触发。



步骤 6 配置 SGPIO_CFG bit[0]为 1，使能 SGPIO。

步骤 7 把各 pin 的输出值对应写入到 SGPIO_DATA_OUT_0~2；读取 SGPIO_DATA_IN_0~2，可以得到对应 pin 的输入值。

步骤 8 读取 SGPIO_INT_STAT_0~2，可以得到对应 pin 的中断状态，读到 0 是没有中断，读到 1 是有中断，写 0 误操作，写 1 清除中断状态。

----结束

须知

用户如果需要修改步骤 1~步骤 5 的配置，需要配置 SGPIO_CFG bit[0]为 0，关闭 SGPIO 之后再修改，修改完成之后再配置 SGPIO_CFG bit[0]为 1，使能 SGPIO。







35 HPRE

35.1 关键术语

表35-1 HPRE 关键术语描述

术语	描述
HPRE	high performace RSA and ECC engine
DH	第一个发表的公开密钥算法出现在 Diffie 和 Hellman 的论文 “New Directions in Cryptography” 中，这篇论文定义了公开密钥密码编码学，而这种算法就是通常被称为 Diffie- Hellman 密钥交换。Diffie-Hellman 密钥交换算法可以让 A、B 双方在一公用网络上传输敏感数据。传输的双方各持有一个公钥和私钥，双方共享一个会话密钥，并以此来传输敏感数据。这样，除了 A 和 B 之外的其他人都无法知道会话密钥，由此可知，敏感数据的安全性得以保证。
ECC	Elliptic Curves Cryptography，椭圆曲线加解密。

35.2 概述

HPRE 是高性能椭圆曲线加解密算法辅助引擎，可以完成点加、点乘、点是否在曲线上的测试。HPRE 逻辑框图如图 35-1 和图 35-2 所示。

图35-1 HPRE 逻辑框图

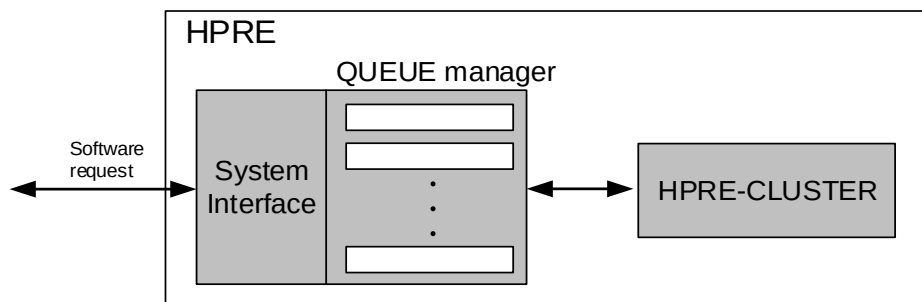
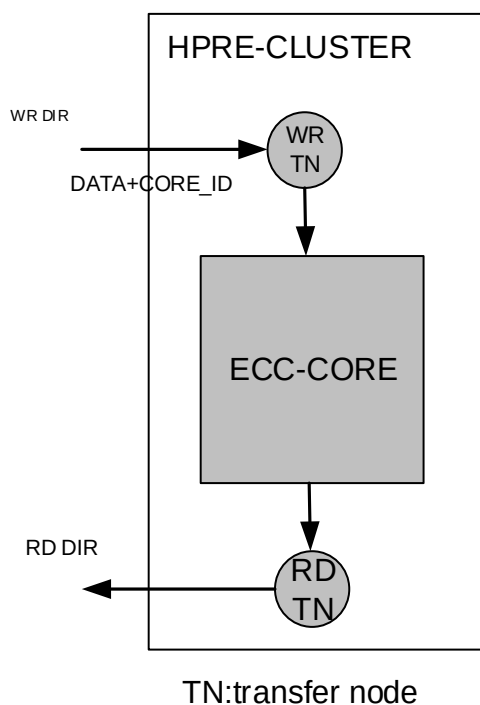


图35-2 HPRE-CLUSTER 逻辑框图



35.3 特性

HPRE 具有以下特性：

- 支持素数域 F_p 椭圆曲线加密算法辅助运算。
 - 支持素数域 F_p 椭圆曲线方程在仿射坐标系下形式为： $y^2=x^3+ax+b$ 。
 - 支持点加运算，支持位宽 128bit/192bit/256bit/320bit/384bit。
 - 支持点乘运算，支持位宽 128bit/192bit/256bit/320bit/384bit。



- 支持点在曲线上判定，支持位宽 128bit/192bit/256bit/320bit/384bit。
- 256bit 点乘运算性能达到 2K/s。
- 支持通过环形 BD 队列及相关指针进行软硬件交互。
 - 支持 16 个物理队列。
 - 每个队列可独立使能和关闭。
 - 每个队列深度可配，配置范围为 2 个~4K-1 个 BD。
 - 队列调度方式：BD 队列按照 WRR 调度。
 - 每个队列的中断处理流程独立。
 - 每个队列的寄存器按照 4K 页表隔离，队列寄存器相互独立，任何一个队列的功能配置，不影响其他队列的功能。
 - 每个队列都支持软件通过队列读写指针管理和查询队列。

35.4 遵从协议

DH 组：

- group 19/20：RFC5903。
- 椭圆曲线：RFC6090。

35.5 功能描述

35.5.1 软硬件交互界面

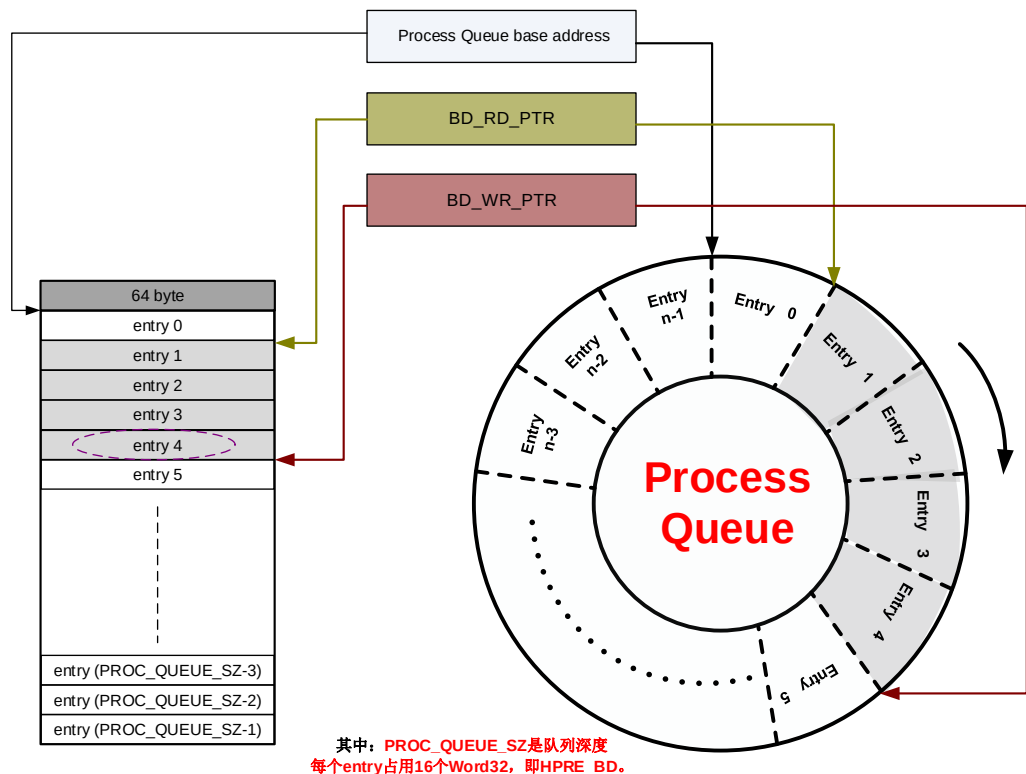
35.5.1.1 总体描述

软件通过向 HPRE 的处理队列（Process Queue）推送任务，来完成软件所需的硬件加速。

HPRE 系统支持 16 个软硬件交互队列，且每个队列可独立使能，其起始地址及长度支持独立配置，Process Queue 与核之间的对应关系由软件来控制。CPU 主要是通过对每个 Process Queue 的写指针 BD_WR_PTR 来告知控制器当前有待处理的 HPRE 任务，而控制器内部则依据 WRR 仲裁算法从各个 Process Queue 中搬取 Process Queue entry，然后对 Process Queue entry（内容为 HPRE BD）进行解析，从而完成所需计算。

软件与硬件逻辑之间的信息交互以 Process Queue entry 为基本单元，如前所述，为了保证信息交互的高效性，Process Queue entry 在 DDR 中以 Queue 的形式存在。其逻辑实现如图 35-3 所示。

图35-3 Process Queue 的基本格式



这是一个深度为 N 的环形队列。队列深度，队列基址，均由 CPU 在初始化时进行配置。队列内基本单元空间的大小固化为 64Byte。

上图中，BD_RD_PTR 是该队列的读指针；而 BD_WR_PTR 是队列的写指针。读指针是由读队列的一侧（HPRE）进行控制，写指针由写队列一侧（CPU）进行控制；读写双方都需要知道读写指针当前出在何种位置，以确定是否能够继续读写。

初始时，BD_RD_PTR 与 BD_WR_PTR 皆为 0，Queue 为空，因此 CPU 可以填入 Process queue entry，当填入若干个 entry 后，CPU 需更新 BD_WR_PTR 值，以告知 HPRE 有新的 Process queue entry 填入；此时 CPU 需要检查 BD_WR_PTR 和 BD_RD_PTR 来判定 Queue 是否为满。若 Queue 满，则不应继续填入 entry，否则可以继续填入。对于 HPRE 一侧，同样通过判断读写指针是否相等，即可判断 Queue 是否为空，只要非空，HPRE 即可启动 DMA，获取 entry 的信息。获取后，HPRE 需要更新 BD_RD_PTR 值。有关队列的空满说明，可以参考 35.5.1.3 队列指针和循环空间章节描述。



HPRE 则通过以下两种方式来告知软件各队列中任务的完成情况：

1. 根据各队列中的 BD_RD_PTR 值获知队列任务读取情况。
2. 每完成一定数量的任务，通过中断方式告知软件。具体完成任务数量可通过寄存器配置。

须知

有关内存空间申请和释放问题：

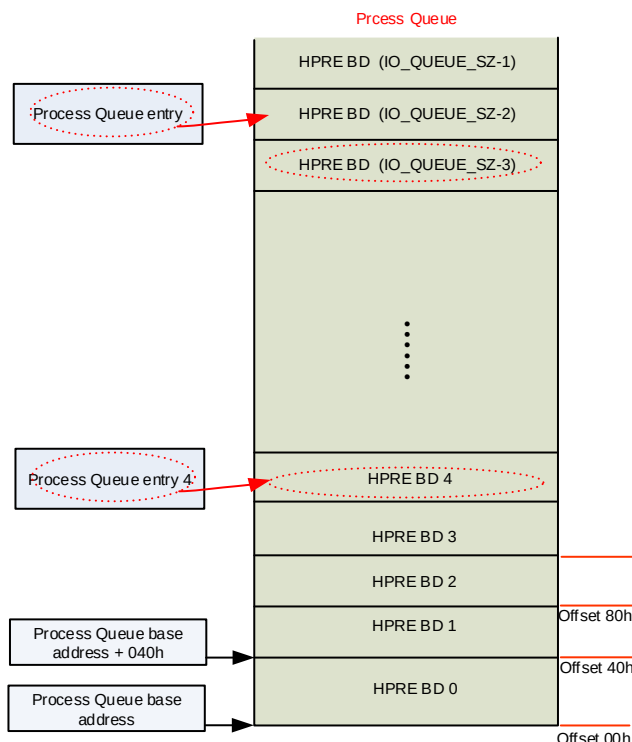
对于 HPRE 而言，每个 entry 就是一个任务。每个任务的数据缓存空间申请和释放均由 CPU（软件）完成。

Process Queue 是一个 HPRE BD 的队列，每一个 entry 对应一个 HPRE BD。控制器通过 BD_RD_PTR 与 Process Queue base address 寄存器的值便可以从队列里面获取对应的 HPRE BD。换言之，BD_RD_PTR = 0 与 HPRE BD 0 关联，BD_RD_PTR = 1 与 HPRE BD1 关联，依此类推。

Process Queue 存储在 DDR 中，其首地址 Process Queue base address 由 CPU 在初始化的时候，通过寄存器进行配置。

Process Queue entry(即：HPRE BD)数据结构如[图 35-4](#) 所示。

图35-4 Process Queue entry(HPRE BD)的基本格式



如图 35-4 所示，每个 Process Queue entry（即：HPRE BD）的长度固定（16 个 Word32，即 64Byte），且各个域的内容均事先约定，控制器可根据 PRC_QUEUE_RD_PTR 与 Process Queue base address 和队列深度计算出每一个 HPRE BD 在 DDR 中的存储地址，通过 DMA 进行获取。

须知

软件需要保证，队列写指针寄存器（BD_WR_PTR）不允许多核同时进行修改，允许多核查询。HPRE 加速器本身对此没有限制，需要软件做好保护机制。

35.5.1.2 队列说明

HPRE 的每个队列均为保序队列，交互接口如下：

写指针（BD_WR_PTR）：软件维护，RW 寄存器，nManager 寄存器名称：QMV1_Q_WR_PTR。

读指针（BD_RD_PTR）：HPRE 维护，RO 寄存器，nManager 寄存器名称：QMV1_Q_RD_PTR。



完成指针 (BD_PROC_PTR): HPRE 维护, RO 寄存器, 用于表征 HPRE 按照队列中的 BD 顺序, 完成到第几个 BD, nManager 寄存器名称: QMV1_Q_PRO_PTR。

软件完成指针 (QMV1_Q_SOFT_PROC_PTR): 软件维护, RW 寄存器, 用于表征软件按照队列中的 BD 顺序, 完成到第几个 BD, nManager 寄存器名称: QMV1_Q_SOFT_PROC_PTR。

接口交互方式

- 添加 BD

软件判定 BD 队列一个 BD 地址能否刷新 BD, 需要根据 BD_PROC_PTR 来处理, 添加完 BD 后, 才能更新 BD_WR_PTR。

软件处理完硬件回写的 BD 后, 需要修改 QMV1_Q_SOFT_PROC_PTR 指针。

- BD 完成查询和处理

查询 BD_PROC_PTR 获知当前已处理过的 BD 编号。

35.5.1.3 队列指针和循环空间

循环空间是 DDR 种开辟的一段空间。循环空间由基地址、深度 (Q_DEPTH)、元素 SIZE 决定。

空间地址计满至空间下边界时需要卷绕到空间基地址。

每个循环空间的软硬件交互接口就是读指针 (RD_PTR) 和写指针 (WR_PTR)。指针指向每个元素的首地址。

HPRE BD 空间, 元素 SIZE 是 64Byte。

HPRE 加速器对指针的理解是: 凡是软件配置的指针值, 都根据循环空间深度进行饱和和处理, 加速器维护的指针保证饱和在循环空间深度内。指针的饱和值是: 队列深度, 例如队列深度配置为 100, 那么指针饱和值就是 100, 就是说指针最大值是 100。

例如: 如果软件配置 HPRE 某个 BD 队列的深度是 100, 而随后 BD 队列的 WR_PTR 配置为 200, 那么 WR_PTR 就被饱和为 100。

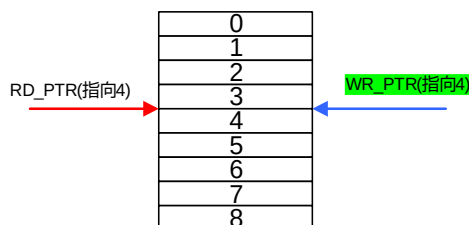
队列可写空间判定: RD_PTR 为 0 时, WR_PTR 可以写到 Q_DEPTH。RD_PTR>0 时, WR_PTR 可以写到 (RD_PTR-1)。

队列可读空间判定: RD_PTR 不断递增 (达到空间边界需要卷绕回 0), 直到 RD_PTR=WR_PTR。

队列空满描述

队列空: $RD_PTR = WR_PTR$ 。

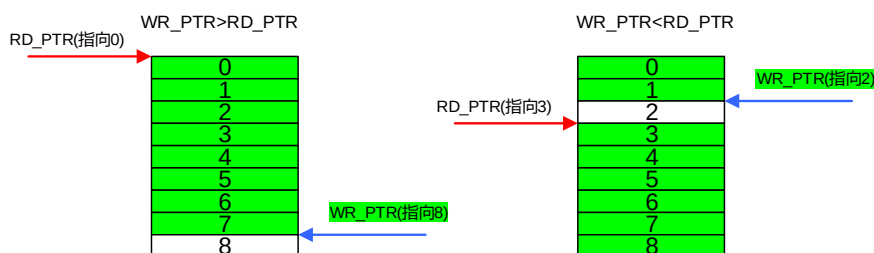
图35-5 队列空图示



队列满:

- $WR_PTR < RD_PTR$ 时, $WR_PTR + 1 = RD_PTR$;
- $WR_PTR > RD_PTR$ 时, $WR_PTR = RD_PTR + Q_DEPTH$; 其中 Q_DEPTH 是队列深度。

图35-6 队列满图示



如图 35-5 和图 35-6 所示, 无论如何, 队列都有一个空格。此处配置深度是 8, 指针最大值可以写到 8。

队列深度配置属性说明

每个队列深度配置寄存器为: `QMV1_Q_DEPTH_CFG`, 域 `hpre_q_depth` 位宽 12bit。深度范围是 0~4095 (元素个数)。软件需要提供 `hpre_q_depth+1` 的 BD 空间。

若配置深度为 0, 则该队列不使能。

读写指针的饱和值和队列深度值一致。若大于饱和值(饱和值为队列深度), 则自动卷绕至 0。例如配置深度是 1023, 读指针为 1023 时, 若读指针再增加 1, 则会卷绕至 0。

发生软件配置导致指针饱和现象的处理(DFX)



如果检测配置指针出现饱和，则加速器内部会把指针识别为饱和值（指针寄存器的配置值是保持不变），然后执行以下操作：

报出相关指针饱和中断。随后出现该类异常的队列将停止参与调度，对于已经读入的 BD，则继续把它们处理完毕。

35.5.2 中断

35.5.2.1 中断寄存器

公共中断

公共中断用于记录 HPRE 出现的公共异常，属于电平中断。

对应《Hi3093 寄存器手册》HPRE_COMMON 页签：

- 中断屏蔽寄存器 HPRE_COMMON_INT_MSK。
- 中断置位寄存器 HPRE_COMMON_INT_SET。
- 原始中断寄存器 HPRE_COMMON_RINT，对应比特写 1 清零。
- 中断状态寄存器 HPRE_COMMON_INT_ST。

队列中断

HPRE 最多支持 16 个物理队列，每个队列都包含一个队列任务中断和队列异常中断。其中，队列任务中断是脉冲中断，队列异常中断是电平中断。对应《Hi3093 寄存器手册》QMV1_QUEUE_VM 页签：

- 队列任务中断相关寄存器如下：
 - 中断屏蔽寄存器 QMV1_Q_FLOW_INT_MSK。
 - 中断置位寄存器 QMV1_Q_FLOW_INT_SET。
 - 原始中断寄存器 QMV1_Q_FLOW_RINT，对应比特写 1 清零。
 - 中断状态寄存器 QMV1_Q_FLOW_INT_STATUS。
- 队列异常中断：
 - 中断屏蔽寄存器 QMV1_Q_FAIL_INT_MSK。
 - 中断置位寄存器 QMV1_Q_FAIL_INT_SET。
 - 原始中断寄存器 QMV1_Q_FAIL_RINT，对应比特写 1 清零。
 - 中断状态寄存器 QMV1_Q_FAIL_INT_STATUS。



35.5.2.2 队列任务中断上报机制

每个队列任务中断上报由如下机制控制生成：

- 基于完成任务的数量生成任务中断。

在没有达到任务中断超时门限的情况下，队列每完成寄存器。

QMV1_Q_PROC_NUM_CFG 配置值个任务，就会生成任务中断。

- 基于超时生成任务中断。

当队列计时器计数至寄存器 QMV1_Q_OT_TH 配置值时，并且此时队列中已经有完成的任务没有上报，那么队列就会基于任务超时生成任务中断上报。

- 任务中断生成后，会清除队列超时计数器和队列完成但未上报任务的计数器。

35.5.3 椭圆曲线加密算法辅助运算

椭圆曲线加密算法辅助运算包括：点乘、点加。

通过支持这些算法，软件可以调用这些辅助运算，为椭圆曲线加密算法相关的算法流程提供硬件加速。

35.5.3.1 点加运算

椭圆曲线 E 上的两个点 P1 和 P2 的和。即：

- 输入点 P1 和 P2。
- 输出点 P3，记为 $P3=P1+P2$ 。

35.5.3.2 点乘运算

椭圆曲线 E 上点 P1 的 k 倍点。

- 输入点 P1 和整数 k。
- 输出点 P2，记为 $P2=[k]P1$ ，相当于 $P2=P1+P1+...P1$ （其中 P1 的个数是 k）。



35.6 编程接口

35.6.1 HPRE 输入 BD

35.6.1.1 数据结构

表35-2 HPRE 输入 BD

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WORD 0	DO NE_ FLA G	BD_RSV1														ERR_TYPE										ALG_TYPE						
WORD 1	BD_RSV2					reserv vd		MRT_TEST_NUM							TASK_LENTH2							TASK_LENTH1										
WORD 2	KEY_ADDR_L																															
WORD 3	KEY_ADDR_H																															
WORD 4	PLAINTEXT_ADDR_L																															
WORD 5	PLAINTEXT_ADDR_H																															
WORD 6	RESULT_ADDR_L																															



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R D 6																																
W O R D 7	RESULT_ADDR_H																															
W O R D 8	TAG																															
W O R D 9	reserved																															
W O R D 10	reserved																															
W O R D 11	reserved																															
W O R D 12	reserved																															
W O R D 13	reserved																															
W O R D 14	reserved																															



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4																																
WORD 15	reserved																															

35.6.1.2 数据结构域段说明

表35-3 HPRE 输入 BD 域段说明

域段	功能描述
ALG_TYPE (WORD0 bit[4:0])	算法类型，域值描述如下。 0xB：ECC 点在曲线上测试； 0xC：ECC 点加； 0xD：ECC 点乘； 其余：保留。
ERR_TYPE	错误类型，BD 回写会修改这个域，输入填 0x0。
BD_RSV1 (WORD0 bit[29:16])	保留域。
DONE_FLAG (WORD0 bit[31:30])	BD 回写时，任务完成标志。 2'b00：保留。 2'b01：未启动或未完成任务。 2'b10：任务完成失败； 2'b11：任务完成成功。
TASK_LENTH1 (WORD1 bit[7:0])	表示以 64bit 为单位的长度值减 1。例如 1024 对应的长度是 1024/64-1=15。配置范围为 1~15。
TASK_LENTH2 (WORD1 bit[15:8])	表示以 64bit 为单位的 P 值长度。例如 1024 对应的长度是 1024/64-1=15。(不需要使用，保留域)



域段	功能描述
MRT_TEST_NUM (WORD1 bit[23:16])	素数测试次数。(保留域)
BD_RSV2 (WORD1 bit[31:25])	保留域。
KEY_ADDR_L (WORD2 bit[31:0])	密钥地址的低 32bit。
KEY_ADDR_H (WORD3 bit[31:0])	密钥地址的高 32bit。
PLAINTEXT_ADDR_L (WORD4 bit[31:0])	明文地址的低 32bit。
PLAINTEXT_ADDR_H (WORD5 bit[31:0])	明文地址的高 32bit。
RESULT_ADDR_L (WORD6 bit[31:0])	结果地址的低 32bit。
RESULT_ADD_H (WORD7 bit[31:0])	结果地址的高 32bit。

35.6.2 HPRE 输出 BD

35.6.2.1 数据结构

表35-4 HPRE 输出 BD 数据结构

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
WORD 0	DO NE _F LA G	BD_RSV1														ERR_TYPE								ALG_TYPE											
WORD 1	BD_RSV2							r e s e r v d	MRT_TEST_NUM							TASK_LENTH2							TASK_LENTH1												
WORD	KEY_ADDR_L																																		



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R D 2																																
W O R D 3	KEY_ADDR_H																															
W O R D 4	PLAINTEXT_ ADDR_L																															
W O R D 5	PLAINTEXT_ADDR_H																															
W O R D 6	RESULT_ ADDR_L																															P J D G
W O R D 7	RESULT_ADDR_H																															
W O R D 8	TAG																															
W O R D 9	reserved																															
W O R D 10	reserved																															
W O R	reserved																															



D 11	21	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W O R D 12	reserved																															
W O R D 13	reserved																															
W O R D 14	reserved																															
W O R D 15	reserved																															

35.6.2.2 数据结构域段说明

表35-5 HPRE 输出 BD 域段说明

域段	功能描述
ALG_TYPE (WORD0 bit[4:0])	算法类型，域值描述如下。 0xB：ECC 点在曲线上测试； 0xC：ECC 点加； 0xD：ECC 点乘； 其余：保留。
ERR_TYPE	错误类型，域值描述如下。 ERR_TYPE 的 bit[10:9]：保留。 ERR_TYPE 的 bit[8]：回写任务结果，发生 AXI 错误，此处不包



域段	功能描述
	含回写 BD; ERR_TYPE 的 bit[7]: ECC 点加、ECC 点乘计算过程中出现无穷远点, 计算结果不可用; ERR_TYPE 的 bit[6]: BD 检查出错; ERR_TYPE 的 bit[5]: 任务参数检查出错; ERR_TYPE 的 bit[4]: 回写任务结果, 发生 ECC 2bit 错误, 此处不包含回写 BD; ERR_TYPE 的 bit[3]: 计算时发生 ECC 2bit 错误; ERR_TYPE 的 bit[2]: 读任务参数发生 AXI 错误; ERR_TYPE 的 bit[1]: 读 BD 发生 AXI 错误; ERR_TYPE 的 bit[0]: 读任务参数 ECC 2bit 错误。
BD_RSV1 (WORD0 bit[29:16])	保留域。
DONE_FLAG (WORD0 bit[31:30])	BD 回写时, 任务完成标志。 2'b00: 保留。 2'b01: 未启动或未完成任务。 2'b10: 任务完成失败; 2'b11: 任务完成成功。
TASK_LENTH1 (WORD1 bit[7:0])	表示以 64bit 为单位的长度值减 1。例如 1024 对应的长度是 $1024/64-1=15$ 。
TASK_LENTH2 (WORD1 bit[15:8])	表示以 64bit 为单位的长度值减 1。例如 1024 对应的长度是 $1024/64-1=15$ 。(不需要使用, 保留域)
MRT_TEST_NUM (WORD1 bit[23:16])	素数测试次数。(保留域)
BD_RSV2 (WORD1 bit[31:25])	保留域。
KEY_ADDR_L (WORD2 bit[31:0])	密钥地址的低 32bit。
KEY_ADDR_H (WORD3 bit[31:0])	密钥地址的高 32bit。



域段	功能描述
PLAINTEXT_ADDR_L (WORD4 bit[31:0])	明文地址的低 32bit。
PLAINTEXT_ADDR_H (WORD5 bit[31:0])	明文地址的高 32bit。
RESULT_ADDR_L (WORD6 bit[31:0])	结果地址的低 32bit。
RESULT_ADDR_H (WORD7 bit[31:0])	结果地址的高 32bit。
PJDG	点在曲线上的判定结果。其中 0x0 表示点不再曲线上，0x1 表示点在曲线上。仅 ALG_TYPE 等于 0xB 时，会输出这个值。

35.6.3 固件编程接口

暂不涉及。

35.6.4 应用编程接口

35.6.4.1 任务添加操作

保序队列任务添加流程如下：

- 步骤 1 软件查询队列寄存器 QMV1_Q_RD_PTR，根据软件自身需要维护的寄存器 QMV1_Q_WR_PTR 判定 BD 队列是否已满；若已满，结束任务添加。
- 步骤 2 软件查询队列寄存器 QMV1_Q_PRO_PTR，软件根据查询结果检索该地址及其下一个地址对应的 BD 是否已经被 HPRE 和软件处理完毕；处理完毕后转步骤 3。
- 步骤 3 软件修改队列寄存器 QMV1_Q_SOFT_PROC_PTR。

须知

保序队列不根据 QMV1_Q_PRO_PTR 和 QMV1_Q_SOFT_PROC_PTR 进行队列反压，因此软件刷新 BD 队列前，必须确保被刷新的 BD 已经被处理完毕。



----结束

35.6.4.2 任务完成操作

保序队列任务添加流程：

步骤 1 软件查询队列寄存器 QMV1_Q_PRO_PTR。

步骤 2 软件根据 QMV1_Q_PRO_PTR 判断队列 BD 完成情况。

----结束

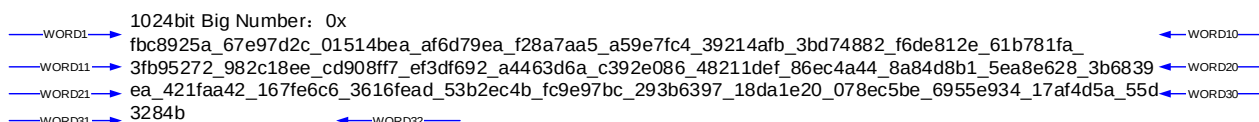
35.6.4.3 任务参数存储及访问方式

大数(Big Number)存储方式

支持数据大端存储方式，以 1024bit 为例，软件 32 位和 64 位处理方式分别列举如下：

- 32 位数据大端存储格式

图35-7 1024bit 大数按照 32bit 划分



32bitCPU 需要把整个 1024bit 大数按照 32 个 WORD (32bit) 来存储；大小端针对每个 WORD 而言，而非整个 1024bit 大数。对于大数而言，WORD1 (0xFBC8_925A) 是最高有效 bit 所在 WORD。



图35-8 1024bit 大数按照 HPRE 引擎要求的 32bit 大端存储结构

32bit Big Endian

Addr0+0x7C	WORD32	0x4b_28_d3_55
Addr0+0x78	WORD31	0x5a_4d_fa_17
Addr0+0x74	WORD30	0x34_e9_55_69
Addr0+0x70	WORD29	0xbe_c5_8e_07
Addr0+0x6C	WORD28	0x20_1e_da_18
Addr0+0x68	WORD27	0x97_63_3b_29
Addr0+0x64	WORD26	0xbc_97_9e_fc
Addr0+0x60	WORD25	0x4b_ec_b2_53

• • • • •

Addr0+0x1C	WORD8	0x82_48_d7_3b
Addr0+0x18	WORD7	0xfb_4a_21_39
Addr0+0x14	WORD6	0xc4_7f_9e_a5
Addr0+0x10	WORD5	0xa5_7a_8a_f2
Addr0+0xC	WORD4	0xea_79_6d_af
Addr0+0x8	WORD3	0xea_4b_51_01
Addr0+0x4	WORD2	0x2c_7d_e9_67

Addr0(Head Address)	WORD1	<p>HPRE gets: 0x5a_92_c8_fb, Byte map: 0x5a à Addr0+0x3; 0x92à Addr0+0x2; 0xc8à Addr0+0x1; 0xfbà Addr0+0x0;</p>
---------------------	-------	---

← first word to gain

- 64 位数据大端存储格式

图35-9 1024bit 大数按照 64bit 划分

1024bit Big Number: 0x
 —DWORD1→ fbc8925a67e97d2c_01514beaaf6d79ea_f28a7aa5a59e7fc4_39214afb3bd74882_f6de812e61b781fa_
 —DWORD6→ 3fb95272982c18ee_cd908ff7ef3df692_a4463d6ac392e086_48211def86ec4a44_8a84d8b15ea8e628_3b683
 —DWORD11→ 9ea421faa42_167fe6c63616fead_53b2ec4bfc9e97bc_293b639718da1e20_078ec5be6955e934_17af4d5a55
 —DWORD16→ d3284b

64bitCPU 需要把整个 1024bit 大数按照 16 个 DDWORD (64bit) 来存储; 大小端针对每个 DDWORD 而言, 而非整个 1024bit 大数。对于大数而言, DDWORD1 是最有效 bit 所在 DDWORD。



图35-10 1024bit 大数按照 HPRE 引擎要求的 64bit 大端存储结构

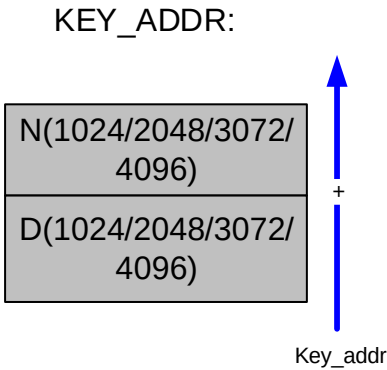
64bit Big Endian		
Addr0+0x78	DWORD16	0x4b_28_d3_55_5a_4d_fa_17
Addr0+0x70	DWORD15	0x34_e9_55_69_be_c5_8e_07
Addr0+0x68	DWORD14	0x20_1e_da_18_97_63_3b_29
Addr0+0x60	DWORD13	0xbc_97_9e_fc_4b_ec_b2_53
Addr0+0x58	DWORD12	0xad_fe_16_36_c6_e6_7f_16
Addr0+0x50	DWORD11	0x42_aa_1f_42_ea_39_68_3d
Addr0+0x48	DWORD10	0x28_e6_a8_5e_b1_d8_84_8a
Addr0+0x40	DWORD9	0x44_4a_ec_86_ef_1d_21_48
Addr0+0x38	DWORD8	0x86_e0_92_c3_6a_3d_46_a4
Addr0+0x30	DWORD7	0x92_f6_3d_ef_7e_f7_8f_90_cd
Addr0+0x28	DWORD6	0xee_18_2c_98_72_52_b9_3f
Addr0+0x20	DWORD5	0xfa_81_b7_61_2e_81_de_f6
Addr0+0x18	DWORD4	0x82_48_d7_3b_fb_4a_21_39
Addr0+0x10	DWORD3	0xc4_7f_9e_a5_a5_7a_8a_f2
Addr0+0x8	DWORD2	0xea_79_6d_af_ea_4b_51_01
Addr0(Head Address)	DWORD1	<div>HPRE gets: 0x2c_7d_e9_67_5a_92_c8_fb, which: 0x2càAddr0+0x7; 0x7dàAddr0+0x6; 0xe9àAddr0+0x5; 0x67àAddr0+0x4; 0x5aàAddr0+0x3; 0x92àAddr0+0x2; 0xc8àAddr0+0x1; 0xfbàAddr0+0x0;</div>

first double word to gain

←

大数数据结构图例

图35-11 大数数据结构图例说明





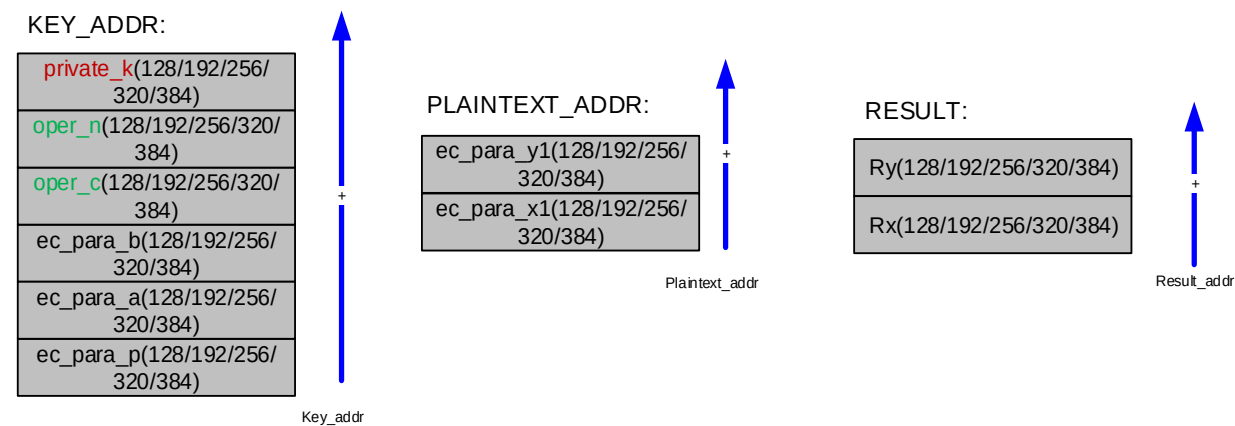
如图 35-11 所示，给定参数地址 Key_addr。

- N 和 D 表征 Key_addr 存储了两个参数，一个是参数 D，一个是参数 N。
- 1024/2048/3072/4096 表征参数的二进制长度。
- 箭头方向表征参数存储的地址递增方向。Key_addr 地址对应参数 D 的第一个字节存储的地址，参数 D 存储方式按照大数存储方式进行存储，具体参考 0 大数(Big Number)存储方式章节。

椭圆曲线点乘计算

图35-12 椭圆曲线点乘计算参数数据结构

ECC点乘:



- 图 35-12 中的所有参数存储方式参考 0 大数(Big Number)存储方式和 0 大数数据结构图例章节。
- 参数说明：

素数域 F_p 椭圆曲线方程在仿射坐标系下形式为： $y^2=x^3+ax+b$ ，其中素域 p 对应 ec_para_p ， ec_para_a 和 ec_para_b 分别对应曲线方程的参数 a 和 b ， $oper_n$ 是曲线方程基点的阶， $oper_c$ 是计算因子。 $private_k$ 是点乘运算的倍数， ec_para_x1/ec_para_y1 分别对应点乘运算的被乘点 x 和 y 坐标， Ry/Rx 分别对应点乘运算结果的 x 和 y 坐标。
- 图 35-12 中 $oper_c$ 需要软件预处理计算获得，计算方式如下：
 - 参数位宽是 128~256bit 的情况下的计算方式： $oper_c=2^{(2*64*4)} \bmod ec_para_p$ 。



- 参数位宽是 257~384bit 的情况下的计算方式: $\text{oper_c} = 2^{(2^{64 \times 6})} \bmod \text{ec_para_p}$ 。
- 参数位宽是 385~576bit 的情况下的计算方式: $\text{oper_c} = 2^{(2^{64 \times 9})} \bmod \text{ec_para_p}$ 。
- 参数位宽是 577~1024bit 的情况下的计算方式: $\text{oper_c} = 2^{(2^{64 \times 16})} \bmod \text{ec_para_p}$ 。

须知

软件请严格按以上计算得到的 oper_c 值进行算法任务参数配置, 否则算法运算可能出错。

- 图 35-12 中所有参数的二进制长度为 64bit 整数倍, 最大长度不超过 1024bit, 非有效位宽按照 0x0 进行填充。128/192/256/320/384 情况, TASK_LENTH1 对应配置为 0x1/0x2/0x3/0x4/0x5。
- BD 配置要求:
 - ALG_TYPE: 配置为 0xd。
 - TASK_LENTH1: ec_para_p/ ec_para_a/ ec_para_b/ ec_para_n/oper_c/private_k/ec_para_y1/ec_para_x1/Ry/Rx 的存储长度均为 64bit 整数倍, 最大长度不超过 1024bit, 非有效位宽按照 0x0 进行填充。128/192/256/320/384 情况, TASK_LENTH1 对应配置为 0x1/0x2/0x3/0x4/0x5。
 - ERR_TYPE: 填 0x0。
 - DONE_FLAG: 填 0x1。
 - KEY_ADDR_L: 填 KEY_ADDR 的低 32bit 值。
 - KEY_ADDR_H: 填 KEY_ADDR 的高 32bit 值。
 - PLAINTEXT_ADDR_L: 填 PLAINTEXT_ADDR 的低 32bit 值。
 - PLAINTEXT_ADDR_H: 填 PLAINTEXT_ADDR 的高 32bit 值。
 - RESULT_ADDR_L: 填结果地址低 32bit 值。
 - RESULT_ADDR_H: 填结果地址高 32bit 值。
 - 其余域段不关注, 填 0x0 即可。
- 计算结果位宽 1280bit。



须知

任务参数要求:

- 1、需要确保椭圆曲线参数 $ec_para_p/ ec_para_a/ ec_para_b/ ec_para_n/oper_c/private_k/ec_para_y1/ec_para_x1$ 是符合密码学应用要求。
- 2、参数 $y1$ 和 $x1$ 必须是曲线方程 $y^2=x^3+ax+b$ 上的一个点，否则计算结果可能会出错。

对于任务长度不满足 64bit 整数倍情况的处理算法:

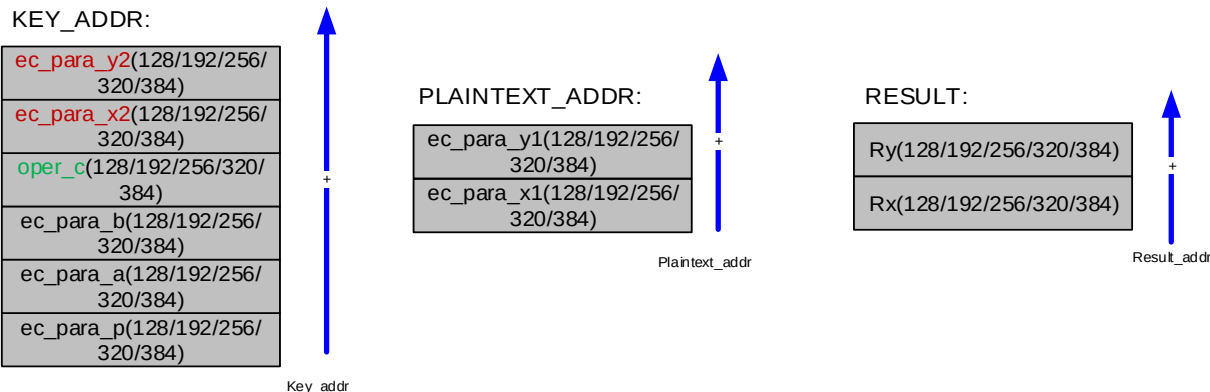
任务长度除以 64，然后进行上取整，然后再减 1；

举例：224bit 的情况， $224/64=3.5$ ，3.5 上取整是 4， $4-1=3$ ，所以 224bit 时，TASK_LENTH1 配置为 3。

椭圆曲线点加计算

图35-13 椭圆曲线点加计算参数数据结构

ECC点加:



- 图 35-13 中的所有参数存储方式参考 0 大数(Big Number)存储方式和 “0 大数数据结构图例” 章节；
- 参数说明：
素数域 F_p 椭圆曲线方程在仿射坐标系下形式为： $y^2=x^3+ax+b$ ，其中素域 p 对应 ec_para_p ， ec_para_a 和 ec_para_b 分别对应曲线方程的参数 a 和 b ， $oper_n$ 是曲线方程基点的阶， $oper_c$ 是计算因子。 $ec_para_x1/ec_para_y1/ ec_para_x2/ec_para_y2$ 分别对应点加运算的两个点的 x 和 y 坐标， Ry/Rx 分别对应点加算结果的 x 和 y 坐标。
- 图 35-13 中 $oper_c$ 需要软件预处理计算获得，计算方式如下：



- 参数位宽是 128~256bit 的情况下的计算方式: $\text{oper_c} = 2^{(2^{64 \times 4})} \bmod \text{ec_para_p}$ 。
- 参数位宽是 257~384bit 的情况下的计算方式: $\text{oper_c} = 2^{(2^{64 \times 6})} \bmod \text{ec_para_p}$ 。
- 参数位宽是 385~576bit 的情况下的计算方式: $\text{oper_c} = 2^{(2^{64 \times 9})} \bmod \text{ec_para_p}$ 。
- 参数位宽是 577~1024bit 的情况下的计算方式: $\text{oper_c} = 2^{(2^{64 \times 16})} \bmod \text{ec_para_p}$ 。

须知

软件请严格按以上计算得到的 oper_c 值进行算法任务参数配置, 否则算法运算可能出错。

- **图 35-13** 中所有参数的二进制长度为 64bit 整数倍, 最大长度不超过 1024bit, 非有效位宽按照 0x0 进行填充。128/192/256/320/384 情况, TASK_LENTH1 对应配置为 0x1/0x2/0x3/0x4/0x5。
- BD 配置要求:
 - ALG_TYPE: 配置为 0xC。
 - TASK_LENTH1: ec_para_p/ ec_para_a/ ec_para_b /oper_c/ ec_para_y1/ec_para_x1/ec_para_y2/ec_para_x2/Ry/Rx 的存储长度均为 64bit 整数倍, 最大长度不超过 1024bit, 非有效位宽按照 0x0 进行填充。128/192/256/320/384 情况, TASK_LENTH1 对应配置为 0x1/0x2/0x3/0x4/0x5。
 - ERR_TYPE: 填 0x0。
 - DONE_FLAG: 填 0x1。
 - KEY_ADDR_L: 填 KEY_ADDR 的低 32bit 值。
 - KEY_ADDR_H: 填 KEY_ADDR 的高 32bit 值。
 - PLAINTEXT_ADDR_L: 填 PLAINTEXT_ADDR 的低 32bit 值。
 - PLAINTEXT_ADDR_H: 填 PLAINTEXT_ADDR 的高 32bit 值。
 - RESULT_ADDR_L: 填结果地址低 32bit 值。
 - RESULT_ADDR_H: 填结果地址高 32bit 值。
 - 其余域段不关注, 填 0x0 即可。



- 计算结果位宽 1280bit。

须知

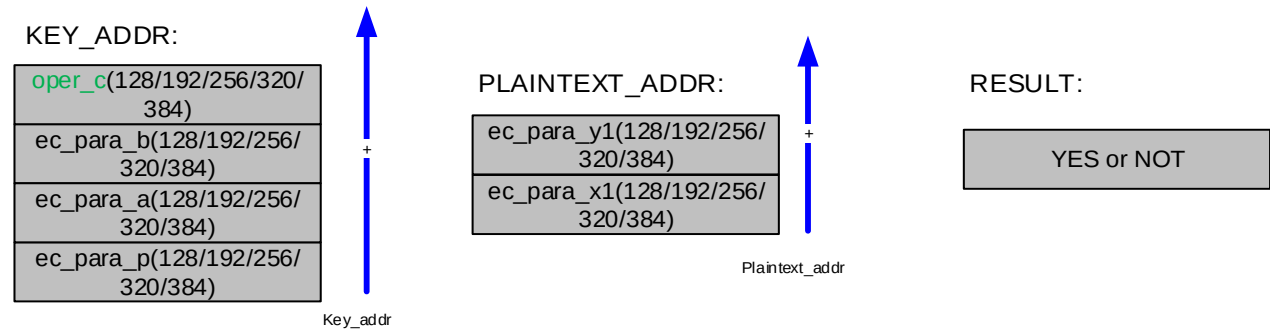
任务参数要求：

- 1、需要确保椭圆曲线参数 ec_para_p/ ec_para_a/ ec_para_b /oper_c/ ec_para_y1/ec_para_x1/ec_para_y2/ec_para_x2 是符合密码学应用要求。
- 2、参数(ec_para_y1,ec_para_x1)和(ec_para_x2,ec_para_y2)必须是曲线方程 $y^2=x^3+ax+b$ 上的点，否则计算结果可能会出错。
- 3、对于任务长度不满足 64bit 整数倍情况的处理算法：
任务长度除以 64，然后进行上取整，然后再减 1；
举例：224bit 的情况， $224/64=3.5$ ，3.5 上取整是 4， $4-1=3$ ，所以 224bit 时，TASK_LENGTH1 配置为 3。

椭圆曲线点在曲线上判定

图35-14 椭圆曲线点加计算参数数据结构

ECC点在曲线上判定:



- 图 35-14 中的所有参数存储方式参考 0 大数(Big Number)存储方式和 0 大数数据结构图例章节。
- 参数说明：
素数域 F_p 椭圆曲线方程在仿射坐标系下形式为： $y^2=x^3+ax+b$ ，其中素域 p 对应 ec_para_p， ec_para_a 和 ec_para_b 分别对应曲线方程的参数 a 和 b ， oper_n



是曲线方程基点的阶，oper_c 是计算因子。ec_para_x1/ec_para_y1 分别对应待测试点的 x 和 y 坐标。

- 图 35-14 中 oper_c 需要软件预处理计算获得，计算方式如下：
- 参数位宽是 128~256bit 的情况下的计算方式： $\text{oper_c} = 2^{(2 \times 64 \times 4)} \bmod \text{ec_para_p}$ 。
- 参数位宽是 257~384bit 的情况下的计算方式： $\text{oper_c} = 2^{(2 \times 64 \times 6)} \bmod \text{ec_para_p}$ 。
- 参数位宽是 385~576bit 的情况下的计算方式： $\text{oper_c} = 2^{(2 \times 64 \times 9)} \bmod \text{ec_para_p}$ 。
- 参数位宽是 577~1024bit 的情况下的计算方式： $\text{oper_c} = 2^{(2 \times 64 \times 16)} \bmod \text{ec_para_p}$ 。

须知

软件请严格按以上计算得到的 oper_c 值进行算法任务参数配置，否则算法运算可能出错。

- 图 35-14 中所有参数的二进制长度为 64bit 整数倍，最大长度不超过 1024bit，非有效位宽按照 0x0 进行填充。128/192/256/320/384 情况，TASK_LENTH1 对应配置为 0x1/0x2/0x3/0x4/0x5。
- BD 配置要求：
 - ALG_TYPE：配置为 0xB。
 - TASK_LENTH1：ec_para_p/ec_para_a/ec_para_b/oper_c/ec_para_y1/ec_para_x1 的存储长度均为 64bit 整数倍，最大长度不超过 1024bit，非有效位宽按照 0x0 进行填充。128/192/256/320/384 情况，TASK_LENTH1 对应配置为 0x1/0x2/0x3/0x4/0x5。
 - ERR_TYPE：填 0x0。
 - DONE_FLAG：填 0x1。
 - KEY_ADDR_L：填 KEY_ADDR 的低 32bit 值。
 - KEY_ADDR_H：填 KEY_ADDR 的高 32bit 值。
 - PLAINTEXT_ADDR_L：填 PLAINTEXT_ADDR 的低 32bit 值。
 - PLAINTEXT_ADDR_H：填 PLAINTEXT_ADDR 的高 32bit 值。
 - RESULT_ADDR_L：填结果地址低 32bit 值。



- RESULT_ADDR_H: 填结果地址高 32bit 值。
- 其余域段不关注, 填 0x0 即可。
- 计算结果通过 BD 判定。

须知

任务参数要求:

1. 需要确保椭圆曲线参数 ec_para_p/ ec_para_a/ ec_para_b/oper_c/ec_para_y1/ec_para_x1 是符合密码学应用要求。
2. 对于任务长度不满足 64bit 整数倍情况的处理算法:
任务长度除以 64, 然后进行上取整, 然后再减 1。
举例: 224bit 的情况, $224/64=3.5$, 3.5 上取整是 4, $4-1=3$, 所以 224bit 时, TASK_LENTH1 配置为 3。

35.6.4.4 异常检测与处理

BD 异常

表35-6 ECC 输入 BD 检查

检查项	检查规则	违背处理方式	BD 回写体现
队列类型 +ALG_TYPE+TASK_LENTH1+TASK_LENTH2	队列类型值是 0x2, 然后检查下面: 0xB、0xC、0xD 之外的 ALG_TYPE 都是违例, TASK_LENTH1 大于 0xf 或者等于 0 是违例, TASK_LENTH2 不检查。	任务失败, 不回写数据, 只回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[6]修改为 0x1, 其余 bit 为 0。



检查项	检查规则	违背处理方式	BD 回写体现
ERR_TYPE	要求等于 0	任务失败，不回写数据，只回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[6]修改为 0x1, 其余 bit 为 0。
DONE_FLAG	要求等于 0x1	任务失败，不回写数据，只回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[6]修改为 0x1, 其余 bit 为 0。
MRT_TEST_NUM	不检查	-	-
KEY_ADDR	不检查	-	-
PLAINTEXT_ADDR	不检查	-	-
RESULT_ADDR	不检查	-	-
TAG	不检查	-	-

数据异常

表35-7 ECC 任务参数检查

任务类型	检查规则	违背后的处理	BD 回写体现
点加	计算结果为无穷远点	任务失败，算法核生成回写数据（数据不可用），回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[7]修改为 0x1, 其余 bit 为 0。
点乘	计算结果为无穷远点	任务失败，算法核生成回写数据（数据不可用），回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[7]修改为 0x1, 其余 bit 为 0。



任务类型	检查规则	违背后的处理	BD 回写体现
点加	参数 p 必须为奇数	任务失败，不回写数据，只回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[5]修改为 0x1, 其余 bit 为 0。
点乘	参数 p 必须为奇数	任务失败，不回写数据，只回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[5]修改为 0x1, 其余 bit 为 0。
点乘	参数 k 必须满足[1,n-1] 区间	任务失败，不回写数据，只回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[5]修改为 0x1, 其余 bit 为 0。
点加	模逆不存在	任务失败，算法核生成回写数据（数据不可用），回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[5]修改为 0x1, 其余 bit 为 0。
点乘	模逆不存在	任务失败，算法核生成回写数据（数据不可用），回写 BD。	1. DONE_FLAG 改为 0x2; 2. ERR_TYPE[5]修改为 0x1, 其余 bit 为 0。

须知

表 35-7 只是对任务参数进行最基本的检查，但并非意味着按照表项的约束就能得到正确的结果，具体约束请参见 35.6.4.3 任务参数存储及访问方式章节。

35.7 应用配置

35.7.1 ECC 算法核的任务数据配置约束

表35-8 ECC 算法核的任务数据配置约束

算法	任务约束
点加	参数 p 必须为奇数
点乘	参数 p 必须为奇数



35.7.2 算法核切换流程

- 步骤 1 软件配置所有队列页签 QMV1_QUEUE_VM 中的 QMV1_QUEUE_ENB 为 0x0，关闭队列预取 BD 地址开关。
- 步骤 2 软件查询所有队列页签 QMV1_QUEUE_VM 中的 QMV1_Q_STATUS 为 0x3 后，再配置 HPRE_CLUSTER 页签的 HPRE_CORE_ENB，完成算法核调度使能开启。

----结束

35.7.3 队列切换流程

队列动态切换有两种情况：队列重用，队列关闭。

- 步骤 1 软件配置页签 QMV1_QUEUE_VM 中的 QMV1_QUEUE_ENB 为 0x0，关闭队列预取 BD 地址开关。
- 步骤 2 软件确认页签 QMV1_QUEUE_VM 中的 QMV1_Q_STATUS 为 0x3 后，可以重新配置队列属性，其中队列 STREAM ID、权重是在 HPRE_COMMON 页签配置。
- 步骤 3 队列属性配置完毕后，软件配置页签 QMV1_QUEUE_VM 中的 QMV1_Q_INIT，完成队列初始化配置。
- 步骤 4 软件配置页签 QMV1_QUEUE_VM 中的 QMV1_QUEUE_ENB 为 0x1，完成队列使能配置。

----结束

须知

使能队列前，需要队列所需的算法核已经被提前使能。禁止不使能算法核情况下，进行队列切换，因为 HPRE 的队列管理模块有预取功能，预取任务后，若没有算法核处理任务，将导致队列无法停止。



35.8 初始化

35.8.1 初始化流程

模块标准复位初始化流程需要按照如下顺序执行：

- 步骤 1 配置 HPRE_COMMON 页签，请参考 [35.8.2 HPRE_COMMON 页签初始化](#) 章节。
- 步骤 2 配置 HPRE_CLUSTER 页签，请参考 [35.8.3 HPRE_CLUSTER 页签初始化](#) 章节。
- 步骤 3 配置队列页签 QMV1_QUEUE_VM，请参考 [35.8.4 QMV1_QUEUE_VM 页签初始化](#) 章节。

----结束

35.8.2 HPRE_COMMON 页签初始化

如下初始化步骤不支持动态改配。若要修改，需要先完成模块复位。

- 步骤 1 配置 HPRE 模块的 AXI Master 接口读写 Qos 寄存器 HPRE_CFG_AXQOS。
- 步骤 2 配置 HPRE 模块的 AXI Master 接口 AWCACHE/ARCACHE 寄存器 HPRE_CFG_AXCACHE。
- 步骤 3 配置 HPRE 模块 BD 大小端寄存器 HPRE_CFG_BD_ENDIAN。
- 步骤 4 配置各个队列 VMID 寄存器 HPRE_Q_STREAMID_CFG，需要配置 16 个地址，从 HPRE_COMMON 页签偏移地址 0x104 开始。
- 步骤 5 配置各个队列权重寄存器 HPRE_Q_WEIGHT_CFG，需要配置 16 个地址，从 HPRE_COMMON 页签偏移地址 0x144 开始。
- 步骤 6 配置各个队列 AR USER 属性配置寄存器 HPRE_Q_ARUSER_CFG，需要配置 16 个地址，从 HPRE_COMMON 页签偏移地址 0x1C4 开始。
- 步骤 7 配置各个队列 AW USER 属性配置寄存器 HPRE_Q_AWUSER_CFG，需要配置 16 个地址，从 HPRE_COMMON 页签偏移地址 0x204 开始。
- 步骤 8 配置读通道 SRAM 初始化寄存器 HPRE_CFG_RDCHN_INI。
- 步骤 9 查询寄存器 HPRE_RDCHN_INI_ST 并确认寄存器值更新为 0x1 为止。



----结束

说明

以上配置步骤必须按照步骤顺序完成配置。

步骤 9 等待时间建议值：500ns。

步骤 1、步骤 2、步骤 6、步骤 7 有关配置，维持默认值即可，软件初始化可以不配置。

35.8.3 HPRE_CLUSTER 页签初始化

如下初始化步骤不支持动态配置，仅允许模块初始化的时候配置，若要改配，请参见 [35.8.5 HPRE_CLUSTER 页签初始化寄存器改配流程](#) 章节。

步骤 1 配置 HPRE 算法核使能寄存器 HPRE_CORE_ENB。

步骤 2 配置 HPRE 算法核初始化寄存器 HPRE_CORE_INI_CFG，本寄存器不管算法核是否使能，都会对所有算法核发起初始化操作。

步骤 3 查询 HPRE 算法核初始化状态寄存器 HPRE_CORE_INI_STATUS，直到确认寄存器值已更新为 0x1。

----结束

说明

以上配置步骤必须按照步骤顺序完成配置。

35.8.4 QMV1_QUEUE_VM 页签初始化

如下配置区分队列进行配置，初始化步骤不支持动态配置，仅允许模块初始化的时候配置，若要改配，请参见 [35.8.6 QMV1_QUEUE_VM 页签初始化寄存器改配流程](#) 章节。

步骤 1 配置队列属性配置寄存器 QMV1_Q_CFG，设置队列保序属性是保序，默认配置是保序。

步骤 2 配置队列完成 BD 中断配置寄存器 QMV1_Q_PROC_NUM_CFG。

步骤 3 配置队列最大深度寄存器 QMV1_Q_DEPTH_CFG。

步骤 4 配置环形队列基地址寄存器 QMV1_Q_BASE_HADDR 和 QMV1_Q_BASE_LADDR。



- 步骤 5 配置队列完成中断超时门限寄存器 QMV1_Q_OT_TH。
- 步骤 6 根据需要，配置队列的中断屏蔽控制寄存器 QMV1_Q_FAIL_INT_MSK 和 QMV1_Q_FLOW_INT_MSK。
- 步骤 7 配置寄存器 QMV1_Q_INIT，完成队列初始化配置。
- 步骤 8 配置寄存器 QMV1_QUEUE_ENB，完成队列使能配置。

须知

只有在完成队列的初始化配置后，软件才能去更新 QMV1_QUEUE_VM 页签中的 QMV1_Q_WR_PTR 寄存器。

QMV1_Q_CFG 禁止配置为非保序，即禁止配置为 0x1。

----结束

说明

以上配置步骤必须按照步骤顺序完成配置。

35.8.5 HPRE_CLUSTER 页签初始化寄存器改配流程

- 步骤 1 配置各个队列寄存器页签的队列使能寄存器 QMV1_QUEUE_ENB。
- 步骤 2 查询各个队列寄存器页签的队列状态寄存器 QMV1_Q_STATUS，确认所有队列值均为 0x3。
- 步骤 3 按照 [35.8.3 HPRE_CLUSTER 页签初始化](#) 章节，根据需求，修改及配置相应寄存器。

----结束

35.8.6 QMV1_QUEUE_VM 页签初始化寄存器改配流程

- 步骤 1 配置各个队列寄存器页签的队列使能寄存器 QMV1_QUEUE_ENB。
- 步骤 2 查询各个队列寄存器页签的队列状态寄存器 QMV1_Q_STATUS，确认所有队列值均为 0x3。



步骤 3 按照 [35.8.4 QMV1_QUEUE_VM 页签初始化](#) 章节，根据需求，修改及配置相应寄存器。

----结束

35.9 时钟复位

HPRE 的 master 和 slave 上电时钟默认为使能状态。

HPRE 的 master 和 slave 时钟共用一个复位请求，默认为去复位状态。

35.9.1 时钟门控

HPRE 算法核、队列调度、任务数据读写接口共用一个时钟门控，HPRE 寄存器接口逻辑使用一个时钟门控。

35.9.2 软复位流程

软件需要按照下述流程对 HPRE 进行软复位准备，各大步骤和小步骤，都要顺序执行。

步骤 1 任务排空和资源回收（如果不需要回收资源，可以跳过此步）：

1. 关闭队列使能。

把所有 QMV1_QUEUE_VM 页签中 QMV1_QUEUE_ENB（偏移地址 0x0018）配置为 0x0。

2. 查询队列状态。

查询到所有 QMV1_QUEUE_VM 页签中的 QMV1_Q_STATUS(偏移地址 0x0600) 均为 0x3 后，表征 HPRE 已排空所有任务。

3. 如果软件在自己规定时间内未查询到（2）中寄存器值变为 0x3，这预示 HPRE 可能已经挂死，需要直接跳到[步骤 2](#)。

须知

发生此种情况会出现资源丢失。

步骤 2 关闭总线接口，请参考《Hi3093 寄存器手册》中的相关寄存器描述，完成关闭总线接口操作：



1. 下发关闭总线接口请求。

把寄存器偏移地址 0x0 的[0]bit (域 ctrl_shutdown_req) 配置为 0x1, 则不再发起新的总线命令。

2. 查询总线命令处理状态。

查询 0x150, 当这个寄存器均为 0x3 时, 表征之前发出的所有总线操作都已经被 master 接口处理完毕。

3. 如果软件在自己规定时间内未查询到 (2) 中寄存器值变为 0x3, 这预示出现总线死锁, 复位准备无法完成, 建议记录各模块总线相关信息后复位整个芯片。

须知

发生此种情况预示出现总线死锁, 推荐做法是记录各模块总线相关信息后复位整个芯片。

当 HPRE 的复位准备完成后, 通过操作 SEC_CSR 中有关 HPRE 的寄存器, 来完成 HPRE 的复位操作:

步骤 3 关闭时钟。写 ACLK_DREQ (偏移为 0x004) 域 hpre_aclken_dreq 为 1。写 HCLK_DREQ (偏移为 0x0EC) 域 hpre_hclken_dreq 为 1。查询 ACLK_ST (0x008) 域 hpre_aclken_st 是否为 0。查询 HCLK_ST (偏移为 0x0F0) 域 hpre_hclken_st 是否为 0。

步骤 4 施加复位。写 SEC_RESET_REQ (偏移为 0x010) 域 hpre_reset_req 为 1; 查询 SEC_RESET_ST (偏移为 0x018) 域 hpre_reset_st 是否为 1, 确认 HPRE 复位施加成功。

----结束

通过操作 SEC_CSR 中有关 HPRE 的寄存器, 来完成 HPRE 的解复位操作:

步骤 1 解除复位。写 SEC_RESET_DREQ(偏移为 0x014)域 hpre_reset_dreq 为 1; 查询 SEC_RESET_ST (偏移为 0x018) 域 hpre_reset_st 是否为 0, 确认 HPRE 复位解除成功。

步骤 2 开启时钟。写 ACLK_REQ (偏移为 0x000) 域 hpre_aclken_req 为 1。写 HCLK_REQ (偏移为 0x0E8) 域 hpre_hclken_req 为 1。查询 ACLK_ST (0x008) 域 hpre_aclken_st 是否为 1。查询 HCLK_ST (偏移为 0x0F0) 域 hpre_hclken_st 是否为 1。



---结束

35.10 性能指标

35.10.1 延迟 (Latency)

以下是估计值，尚未进行实际测试，特别是素数对计算的时间波动比较大，延迟统计需要根据实测数据给出。

此处使用的统计单位是 HPRE 的计算 cycle，Hi3093 假定实现时钟是 400MHz，则这个统计单位折合为 2.5 ns。

此处仅提供算法核内部延迟，没有考虑总线延迟引入的影响，仅供参考。

表35-9 椭圆曲线点加计算延迟 (随机 10 个点加)

参数位宽(bit)	计算所需周期(cycles)	每秒次数 (K/s)
128	6775	59.04
192	8697	45.99
256	10404	38.45
320	14223	28.12
384	16207	24.68

表35-10 椭圆曲线点乘计算延迟 (私钥 K 值满位宽，每字节取 0xAA)

参数位宽(bit)	计算所需周期(cycles)	每秒次数 (K/s)
128	98322	4.07
192	146882	2.72
256	199596	2.00
320	355034	1.13
384	435449	0.92



36 SEC

36.1 概述

SEC 主要应用在数据认证和加密等安全应用中，用以优化安全算法处理能力。

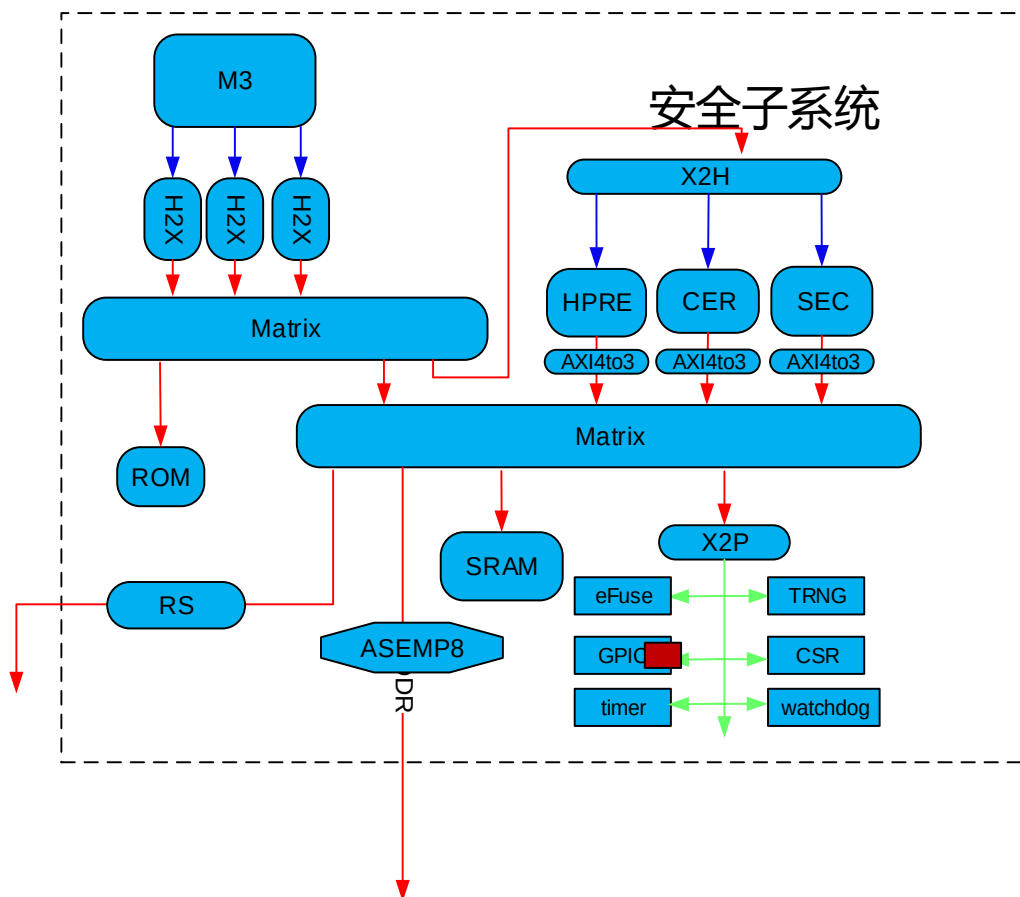
SEC 模块实现了认证算法 SHA-256 和 SHA-512，以及其 HMAC SHA-256 和 HMAC SHA-512。实现了机密算法 AES 的 CBC、CTR、GCM 模式。

36.1.1 相关模块

在系统中，SEC 模块会和下面多个模块进行交互，如图 36-1 所示。

- CPU：解析配置信息以及报文，生成 SEC BD，并下发到 SEC 的 BDFIFO 里。SEC 处理完任务后回写 BD（状态和结果），CPU 继续对任务做后处理。
- DDR/SRAM：存储 BD、报文信息，SEC 从这里中读取 BD、报文信息，处理完后再更新这里的内容。
- TRNG：SEC 可以从 TRNG 获取随机数，用于生成某些加密/认证算法的 IVIN。
- EFUSE：SEC 可以从 EFUSE 获取 HUK，在某些安全任务（如派生密钥）里用做 KEY。

图36-1 SEC 加速引擎交互模块



说明

本框图仅体现与 SEC 相关的主要模块，并不反映完整的系统架构图。

36.2 特性

- 支持 BD FIFO 工作方式（BD FIFO 工作方式的 SEC 是安全设备）。
 - BD FIFO 工作方式：仅用于安全世界软件通过 SEC 寄存器接口向 SEC 下发任务（一次配置一个 BD 的地址），多核并发时由安全世界软件完成调度。
- 提供 SEC BD 描述符解析功能。
 - 支持通过解析 SEC BD 描述符获得流信息、控制信息、地址信息、长度信息；
 - 一个 SEC BD 描述符可支持一次认证和一次机密性算法操作，机密性和认证的数据范围可独立配置。



- 支持处理报文。
 - SEC BD 的起始地址 64Byte 对齐;
 - 报文、MAC、IVIN、KEY 的起始地址字节对齐;
 - 块算法(block cipher)进行机密性处理的报文的长度按 blocksize 对齐 (AES 的 CBC 模式仅支持长度 16Byte 对齐);
 - 流算法(stream cipher)进行机密性处理的报文的长度字节对齐;
 - 完整性算法 (即认证) 进行处理的报文的长度字节对齐。
- 支持 AES 加密、解密运算, 符合 NIST FIPS-197 标准。
 - AES 密钥长度支持 128、256 bits;
 - AES 支持 CBC 和 CTR 工作模式, 工作模式符合 NIST 的 SP 800-38A 标准; AES-CBC with 128-bit keys 符合 RFC3602 协议标准; AES-CTR 符合 RFC3686 协议标准, 也符合 3GPP TS 33.401 标准;
 - 支持基于 AES 的 AES_GCM 算法, 即 Counter with GMAC 算法, 符合 RFC4106 协议标准;
 - AES_GCM 的 AAD 长度和机密运算报文长度不能同时为 0。
- 支持 SHA-256、SHA512 数据摘要算法, 符合 FIPS 180-4 标准要求;
- 支持 HMAC, 符合 RFC2104 标准, HMAC 是基于 SHA-256、SHA-512;
- 支持大小端特性如下:
 - SEC 寄存器只支持以小端模式配置;
 - SEC 通过 AXI Master 接口处理报文相关数据 (含报文、KEY、IVIN、MAC 等) 时只支持大端模式;
 - BD 解析描述符支持大小端模式寄存器可配; 且在解析 BD 时支持两种大小端转换类型: 对 BD 在 32bit 内进行字节序翻转 (32bit 之间的顺序保持不变) 后再进行解析, 和对 BD 在 64bit 内进行字节序翻转 (64bit 之间的顺序保持不变) 后再进行解析, 两种转换类型寄存器可配;
 - 大小端模式配置默认为大端。
- 支持动态功耗控制。
 - 支持各算法核在没有处理数据时时钟被硬件自动关断, 此功能有独立使能位;
 - 支持 ICV Check 功能, Check 结果可以通过 WB_BD 写入 DDR/SRAM 中, 供 CPU 查询。
- SEC 的任务保序完成、顺序执行。
- 支持从 TRNG 获取加密/认证算法所需 IVIN 的随机数部分。



- 支持处理结果输出到目的地址。
 - 支持由软件申请新 buffer 作为目的地址;
 - 目的地址 64Byte 对齐, 填写到 BD 中;
 - 输出到目的地址的内容包括 WB_BD 和结果数据。
- 支持长数据 HASH 功能。
 - 支持 SHA256、SHA-512、HMAC_SHA256、HMAC_SHA-512 算法;
 - 支持最大数据长度为 $(2^{61}-1)$ B;
 - 长数据 (报文) 划分成 ≥ 2 段, 每段生成 1 个任务, 软件原子保序;
 - 各段数据长度 >0 ;
 - 除最后一段, 各段数据长度必须为算法 blocksize 的整数倍。
- 支持对处理包的数量进行统计。
- 提供智能心跳功能, 实时监控内部核心状态机运行, 如果状态机超时无跳转, 则表示心跳停止, 将上报中断。
- 提供 AXI 端口的平均读写延迟查询和最大读写延迟查询。
- 支持 Debug Bus 接口, 该接口可读取 SEC 的所有外部可访问寄存器, 例如 SEC 的性能统计、Fabric 访问延迟统计、智能心跳功能。
- 提供故障检测和错误处理机制: 支持典型错误 BD 识别和处理。
- 提供故障告警功能。
 - 故障告警提供状态寄存器查询、中断上报功能;
 - 故障告警支持描述符中标示出异常状态。
- 支持 SEC 模块内部重要信息软件可见。
 - 提供重要状态寄存器查看功能, 比如: SAA 正在处理的任务数, 入口队列和出口队列里的任务数;
 - 状态机状态可见。
- 支持 BD FIFO 空满状态可查询, 支持 BD FIFO 空读满写行为的中断上报。
- 支持加速器暂停。
 - 支持 SEC 整模块暂停;
- 支持逻辑处理容错。
 - 模块耦合容错: 不挂死, 可恢复; 资源可回收;
 - 异常报文容错: 模块合理处理部分异常报文, 模块不挂死, 不影响后续模块处理。



- 支持 Memory 1bit ECC 错误纠错、统计并上报中断，2bit ECC 错误上报中断。

36.3 关键术语

如表 36-1 所示的术语将在本章节表示特殊的含义。

表36-1 SEC 关键术语描述

术语	描述
AAD	Additional authenticated data，附加认证数据。
AES	Advanced Encryption Standard，先进加密标准，一种加解密算法。
BD	Buffer Description，数据描述符。用于描述任务信息。
WB_BD	write-back BD，任务执行完成后，回写状态到 BD 中，称为回写 BD。
CBC	Cipher Block Chaining，密码分组链接，加解密算法的一种工作模式。
CTR	Counter，计数，加解密算法的一种工作模式。
ECB	Electronic Code Book，电子密码本，加解密算法的一种工作模式。
FIPS	Federal Information Processing Standards，(美国)联邦信息处理标准。
Flag	AES-CCM 的 IVIN 组成元素，据其他参数计算得到。
GCM	Galois/Counter Mode，伽罗华/计数模式，一种既有加解密又有认证的算法工作模式。
GMAC	Galois Message Authentication Code，一种基于加密算法的认证算法。
HASH	哈希，又称散列，一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。



术语	描述
HMAC	Hash-based Message Authentication Code, 密钥相关的哈希运算消息认证码。
HUK	存储硬件特定对称性密钥 (HUK-Hardware Unique Key) 是随机产生, 每个系统不同, 仅用于密钥迁移。由 HUK 派生的密钥用于 SST(trusted world Secure Storage 可信世界安全存储)文件的加解密, 可用于支持掉电存储, 采用例如 AES256 算法。
MAC	Message Authentication Codes, 消息认证码。
NIST	National Institute of Standards and Technology, 美国标准与技术研究所。
nonce	AES-CTR 算法下用于组成 IVIN。
SAA	Security Algorithm Accelerator, 安全算法加速模块。
Salt	盐值, AES-CCM/AES-GCM 算法下用于组成 IVIN。
SEC	Security Engine, 硬件安全加速模块 (也即本模块), 用以加强处理器的安全相关功能。
SHA	Security Hash Algorithm, 安全散列函数, 一种认证算法。

36.4 遵循协议

- AES 加、解密运算, 符合 NIST FIPS-197 标准;
- AES 支持 CBC 和 CTR 工作模式, 工作模式符合 NIST 的 SP 800-38A 标准;
- AES-CBC with 128-bit keys 算法, 符合 RFC3602 协议标准;
- AES-CTR 算法, 符合 RFC3686 协议标准;
- AES_GCM 算法, 即 Counter with GMAC 算法, 符合 RFC4106 协议标准;
- SHA-256、SHA512 算法, 符合 FIPS 180-4 标准要求;
- HMAC, 符合 RFC2104 标准, 基于 SHA-256、SHA512。



36.5 功能描述

本章节描述 SEC 的功能和工作模型，让读者了解它的基本操作方法。这里讲到的都是通用概念和操作方法，用于全部算法的理解和使用。有些算法有特殊之处，将在后面有专门章节另作描述，它所描述的是与通用方法的差异。

36.5.1 任务

SEC 处理的算法任务有两类：认证任务和机密任务。

- 认证任务

认证任务分为认证生成和认证校验。认证生成是对数据计算出 MAC 值，作为输出结果。认证校验是先对数据求出 MAC 值，再与期望 MAC 值比较；若相等则认为认证通过，否则不通过。

认证算法主要有 HASH 和 HMAC，以及 AEAD 类（例如 GCM）。HASH 无 key，HMAC 有 key。

- 机密任务

机密任务分为加密和解密。机密算法都需要 key，还有各种因算法而异的参数。

（日常人们喜欢将加密和解密统称为加密任务，但本文中有大量地方要区分二者，所以为避免混淆尽量使用机密作为统称）

SEC 还有一个 hardcopy 功能，将数据搬到新的地方。这个功能可以单纯地使用；或与算法任务组合使用，例如对数据执行认证生成，得到 MAC 值，并将数据搬到新的地方。这样能减少 CPU 的负荷，提高效率。

36.5.1.1 BD

SEC 处理算法任务。每一个任务由一个 BD 来描述。BD 是个数据结构，里面各个域指明了要执行什么算法、算法的参数、结果放在哪里等等。

一个 BD 能表达一个认证任务、或一个机密任务、或认证+机密组合任务。

BD 用于用户向 SEC 发送任务。

36.5.1.2 WB_BD

SEC 执行完运算后，会将结果状态、错误信息等填写到 BD 里，并将 BD 返回给用户，这称为 WB_BD。用户能通过查询它知道任务的状态。

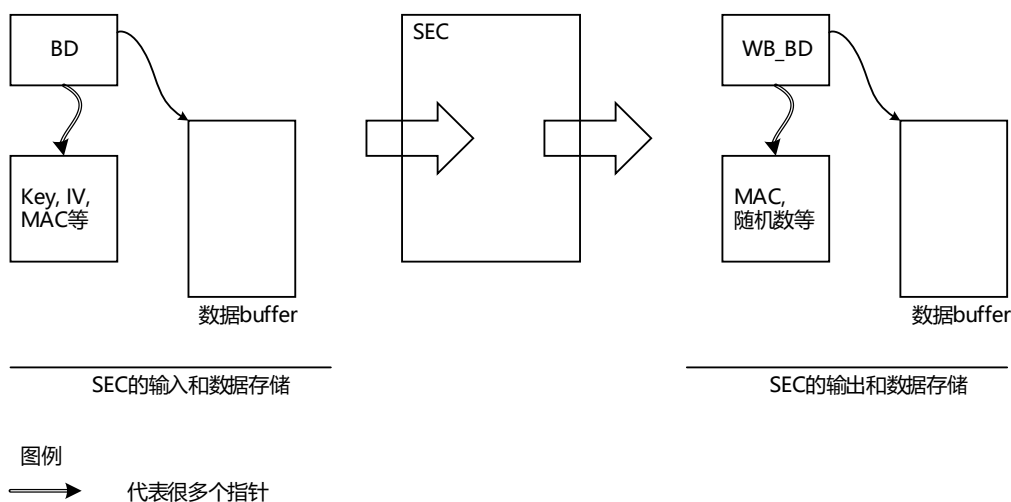
WB_BD 的位置（是在原 BD 还是新的地方），取决于回写方式和错误等因素，详细请参见 36.5.18.1 回写的目的地章节。

须知

用户不能通过查询 WB_BD 来获知任务是否完成，而是在得知任务完成后再查询 WB_BD。

36.5.2 任务流程

图36-2 一个任务的 BD 和数据



须知

本图 36-2 仅作示意，并不代表真实应用情况。例如，key/IV/MAC 等参数可以放在数据 buffer 中属于用户使用的区域，不一定要放在数据 buffer 之外。

用户有很多算法任务要处理，SEC 能提供加速处理。用户将任务信息和数据准备好，通知 SEC。SEC 在收到通知后执行任务。最后用户查询任务是否完成和完成的状态。

如图 36-2 所示，是一个任务经过 SEC 处理的前后。实际应用中有大量任务（BD 和数据）同时存在。BD 和数据都是存放在内存中。报文存放在数据 buffer 中。其余的数据主要是算法参数，或称为辅助数据，诸如 key/iv/mac 等。辅助数据的位置没有太多讲究，可以放在数据 buffer 内未占用的地方，也可以放在数据 buffer 外。不论辅助数据位于何处，都是通过指针与 BD 关联。



SEC 处理系统安全任务（密钥派生、安全备份等）大致的工作流程如下：

安全世界软件（CPU）根据任务的要求，生成任务 BD；将 BD 和各种数据准备好，放在 DDR 中。

步骤 1 安全世界软件通过 SEC 的寄存器接口，发送 BD 任务；

步骤 2 SEC 读取并解析 BD，执行任务（包括读取各种数据、运算、在运算过程中写出结果数据）；

步骤 3 SEC 将结果数据写回 DDR 中；写 WB_BD；

步骤 4 SEC 更新 BD FIFO 任务统计信息，软件通过查询统计寄存器和 WB_BD 获知任务处理情况。

----结束

36.5.3 保序

36.5.3.1 保序完成

SEC 保证在入口和出口，任务的顺序相同；亦即 SEC 接收任务和返回完成任务的顺序相同。

用户发送任务时，形成序。这个序与 SEC 接收任务的序相同。但在 SEC 内部，任务有可能是乱序执行。在最后返回完成任务给用户时，SEC 将任务重新排序，确保与接收任务的顺序相同。这就是保序完成（in-order completion）。

须知

Hi3093 的 SEC 内部，任务顺序执行。

36.5.3.2 原子保序

原子保序是指软件要等一个任务完成后再发送下一个任务。这通常是用于“前一个任务的结果作为下一个任务的输入”的情况。长数据 HASH 功能要求软件进行原子保序。



没有原子保序要求的任务可以批量发送给 SEC。这些任务缓存在 SEC 内部，乱序、流水线地执行。流水线地执行是指，前一个任务进行到某个阶段，下一个任务就开始执行；不是等前一个任务执行完毕才开始。所以它们之间不能有数据依赖关系。

36.5.4 数据类型

输入数据有以下几种：

- 报文
这就是通常所说的数据，例如等加密的数据。
- 辅助数据
包括算法所用到的各种参数，诸如 key，IV 等。
- 随机数
这个数据来源于 TRNG 或 DDR。属于辅助数据，请参考 [36.5.11.1 申请随机数](#) 章节。
- MAC
认证校验的任务将计算出来的 MAC 值与期望 MAC 值比较。所以需要将期望 MAC 作为输入数据

输出数据有以下几种：

- 报文
这就是通常所说的数据，例如等加密的数据。
- MAC
认证生成的任务会回写 MAC 值。
- 随机数
这是回写随机数，请参考 [36.5.11.2 回写随机数](#) 章节。

这些数据，无论是源数据还是结果数据，都是放在 DDR 中（有少数放在 BD 中，而 BD 也在 DDR 中）。在 BD 和 WB_BD 中有指针指向它们。

36.5.5 BD 数据结构

有 2 种 BD 数据结构，BD4 用于各种运算（除了长数据 HASH），BD5 只用于长数据 HASH 运算。（BD1-BD3 在 Hi3093 项目无效）



36.5.5.1 BD

表36-2 BD4 结构 (机密+认证, 128B)

-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Word0	PrevAddr[31:0]/TimeStamp_NanoSecond[31:0]																																
Word1	PrevAddr_Reserved/TimeStamp_Second[31:16]																PrevAddr[47:32]/TimeStamp_Second[15:0]																
Word2	NextAddr[31:0]																																
Word3	NextAddr_Reserved																NextAddr[47:32]																
Word4	RefItselfAddr[31:0]																																
Word5	RefItselfAddr_Reserved																RefItselfAddr[47:32]																
Word6	DataOffset																DataLength																
Word7	ChainOffset																T_Len=6'b011000						T_TagValue=6'b000010						Dsc_Len=4'b0010				
Word8	Reserved														Dst_Rls_Ind		Dst_Addr_Type		Src_Addr_Type		Scene				DE		SEQ	Auth		Cipher			
Word9	Reserved																C_S		Reserved		AI_GEN		CI_GEN		Reserved		HU	Reserved		Nonce_Len			



-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								K								
Word10	Reserved															A_Algo					AKey_Len					MAC_Len						
Word11	Reserved												C_Algo				C_Mode			CKey_Len		C_Width			C_ICV_Len							
Word12	IV_Offset[7:0]								Auth_Len																							
Word13	IV_Offset[15:8]								Cipher_Len																							
Word14	Cipher_Source_Offset															Auth_Source_Offset																
Word15	CS_UDP_Header_Offset															CS_IP_Header_Offset																
Word16	Salt[7:0]								Salt[15:8]								Salt[23:16]						Salt[31:24]									
Word17	Location		Vqi_Rsv		Vqi[9:0]										Err_Vqi_Rsv			Err_Vqi[9:0]								Reserved						
Word18	Afflnd	Order_Rsv				Order_Type		Order_TagValue[23:0]/Affinity_TagValue[23:0]																								
Word19	Padding Length								Next Header							Reserved			IP_Type		Mode		Protocol_Type			Deal_ESP_AH						
Word20	MAC_Addr[31:0]																															



-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Word21	Reserved															MAC_Addr[47:32]																	
Word22	Auth_Key_Addr[31:0]																																
Word23	Reserved															Auth_Key_Addr[47:32]																	
Word24	Cipher_Key_Addr[31:0]																																
Word25	Reserved															Cipher_Key_Addr[47:32]																	
Word26	Auth_IVIN_Addr[31:0]																																
Word27	Reserved															Auth_IVIN_Addr[47:32]																	
Word28	Cipher_IVIN_Addr[31:0]																																
Word29	Reserved															Cipher_IVIN_Addr[47:32]																	
Word30	Dst_Addr[31:0]																																
Word31	Reserved															Dst_Addr[47:32]																	



表36-3 WB_BD4 结构 (机密+认证, 128B)

	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
Word 0	复制自源 BD																															
Word 1																																
Word 2	NextAddr[31:0]																															
Word 3	NextAddr_Reserved															NextAddr[47:32]																
Word 4	RefItselfAddr[31:0]																															
Word 5	RefItselfAddr_Reserved															RefItselfAddr[47:32]																
Word 6	DataOffset															DataLength																
Word 7	ChainOffset															T_Len=6'b011000						T_TagValue=6'b000010						Dsc_Len=4'b0010				
Word 8	复制自源 BD																															
Word 9																																
Word 10																																



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



	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
Word 23																																
Word 24	Warning_Type								Error_Type								Reserved						Flag		CSC		ICV		Done			
Word 25	MAC_I[7:0]								MAC_I[15:8]								MAC_I[23:16]						MAC_I[31:24]									
Word 26	Reserved																Checksum_I															
Word 27	TimeStamp[31:0]																															
Word 28	Reserved																				TimeStamp[42:32]											
Word 29	Src_Addr[31:0]																															
Word 30	Reserved																Src_Addr[47:32]															
Word 31	复制自源 BD																															



表36-4 BD5 结构 (长数据 hash, BD 大小 128B)

-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word0	PrevAddr[31:0]/TimeStamp_NanoSecond[31:0]																															
Word1	PrevAddr_Reserved/TimeStamp_Second[31:16]															PrevAddr[47:32]/TimeStamp_Second[15:0]																
Word2	NextAddr[31:0]																															
Word3	NextAddr_Reserved															NextAddr[47:32]																
Word4	RefItselfAddr[31:0]																															
Word5	RefRbdBufAddr_Reserved															RefRbdBufAddr[47:32]																
Word6	DataOffset															DataLength																
Word7	ChainOffset															T_Len=6'b011000						T_TagValue=6'b000011						Dsc_Len=4'b0010				
Word8	Reserved															Reserved			Src_Addr_Type			Scene			Reserved		SEQ	Auth		Reserved		
Word9	Reserved															Reserved		NO_PAD		AI_GE_N		Reserved		HUK	Reserved			Reserved				
Word10	Reserved															A_Alg						AKey_Len						MAC_Len				



-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Word11	Reserved								Auth_Len																								
Word12	IV_Offset																Auth_Source_Offset																
Word13	Long_Auth_Data_Len[31:0]																																
Word14	Long_Auth_Data_Len[63:32]																																
Word15	Location		Vqi_Rsv		Vqi[9:0]										Err_Vqi_Rsv		Err_Vqi[9:0]										Reserved						
Word16	Affind	Order_Rsv					Order_Type		Order_TagValue[23:0]/Affinity_TagValue[23:0]																								
Word17	Reserved																																
Word18	MAC_Addr[31:0]																																
Word19	Reserved																MAC_Addr[47:32]																
Word20	Auth_Key_Addr[31:0]																																
Word21	Reserved																Auth_Key_Addr[47:32]																
Word22	Auth_IVIN_Addr[31:0]																																



-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word23	Reserved																Auth_IVIN_Addr[47:32]															
Word24	Warning_Type								Error_Type								Reserved				Flag		Reserved			ICV		Done				
Word25	MAC_I[7:0]								MAC_I[15:8]								MAC_I[23:16]				MAC_I[31:24]											
Word26	TimeStamp[31:0]																															
Word27	Reserved																				TimeStamp[42:32]											
Word28	Reserved																															
Word29	Reserved																															
Word30	Reserved																															
Word31	Reserved																															



表36-5 WB_BD5 结构 (长数据 hash, BD 大小 128B)

-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	复制自源 BD																															
Word 1																																
Word 2	NextAddr[31:0]																															
Word 3	NextAddr_Reserved															NextAddr[47:32]																
Word 4	RefItselfAddr[31:0]																															
Word 5	RefRbdBufAddr_Reserved															RefRbdBufAddr[47:32]																
Word 6	DataOffset															DataLength																
Word 7	ChainOffset															T_Len=6'b011000						T_TagValue=6'b000011						Dsc_Len=4'b0010				
Word 8	复制自源 BD																															
Word 9																																
Word 10																																



-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 11																																
Word 12																																
Word 13																																
Word 14																																
Word 15																																
Word 16																																
Word 17																																
Word 18																																
Word 19																																
Word 20																																
Word 21																																
Word 22																																



-	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
Word 23																																
Word 24	Warning_Type								Error_Type								Reserved						Flag		Reserve d		ICV		Do ne			
Word 25	MAC_I[7:0]								MAC_I[15:8]								MAC_I[23:16]						MAC_I[31:24]									
Word 26	TimeStamp[31:0]																															
Word 27	Reserved																				TimeStamp[42:32]											
Word 28	Reserved																															
Word 29																																
Word 30																																
Word 31																																

当{T_Len, T_TagValue, Dsc_Len}出错时的 WB_BD 如表 36-6 所示。这种错误造成 WB_BD.Done 等域写到了不同的地方；并且那个位置复用了 BD.MAC_Len，不能初始化。软件在查询 WB-BD 前，应该先检查原 BD 的{T_Len, T_TagValue, Dsc_Len}；如果正确才继续查看域 Done。



表36-6 {T_Len, T_TagValue, Dsc_Len}出错时的WB_BD 结构 (BD 大小 128B)

-	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	复制自源 BD																															
Word 1																																
Word 2	NextAddr[31:0]																															
Word 3	NextAddr_Reserved															NextAddr[47:32]																
Word 4	RefItselfAddr[31:0]																															
Word 5	RefItselfAddr_Reserved															RefItselfAddr[47:32]																
Word 6	DataOffset															DataLength																
Word 7	ChainOffset															T_Len=6'b001000						T_TagValue=6'b000100						Dsc_Len=4'b0001				
Word 8	复制自源 BD																															
Word 9																																
Word 10	Warning_Type								Error_Type								Reserved						Flag			Reserve d			ICV			Do ne



-	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
Word 11	TimeStamp[31:0]																															
Word 12	Reserved																					TimeStamp[42:32]										
Word 13	Src_Addr[31:0]																															
Word 14	Reserved															Src_Addr[47:32]																
Word 15	复制自源 BD																															
Word 16																																
Word 17																																
Word 18																																
Word 19																																
Word 20																																
Word 21																																
Word 22																																



-	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
Word 23																																
Word 24																																
Word 25																																
Word 26																																
Word 27																																
Word 28																																
Word 29																																
Word 30																																
Word 31																																



36.5.5.2 BD 域说明

下面是 BD 和 WB_BD 的各个域说明。表格之后有更多补充说明。

表36-7 BD 域的说明

说 明	
Dsc_Len	BD 的长度，单位：512bit，(0:256bit, 1:512bit)；该域在每个 BD 中为固定值。
T_TagValue	BD 的类型标识，该域在每个 BD 中为固定值。
T_Len	BD 的类型标识，该域在每个 BD 中为固定值。
PrevAddr	在 Hi3093 中无效。
NextAddr	固定为 0。
RefItselfAddr	数据 buffer 地址，不允许为 0。 RefItselfAddr、DataLength、DataOffset 一起使用，请参考 43.5.6 存储结构章节。
DataLength	在 Hi3093 中无效。
DataOffset	数据起始点相对于 RefItselfAddr 的偏移，单位：Byte。 RefItselfAddr、DataLength、DataOffset 一起使用，请参考 43.5.6 存储结构章节。
ChainOffset	在 Hi3093 中无效。
TimeStamp_NanoSecond[31:0]	在 Hi3093 中无效。
TimeStamp_Second[31:0]	在 Hi3093 中无效。
Cipher	机密算法功能。 0x0：无机密算法； 0x1：加密； 0x2：解密； 0x3：硬件拷贝。



说 明	
Auth	认证算法功能。 0x0: 无认证算法; 0x1: 认证生成; 0x2: 认证校验; 0x3: 保留。
SEQ	机密运算和认证运算的处理顺序: 0: 先机密后认证 (CtA) ; 1: 先认证后机密 (AtC) ;
DE	目标地址选择。 0x0: 原址回写; 0x1: 新址回写; 0x2: 在 Hi3093 中无效; 0x3: 保留。
Scene	固定为 1 (在 Hi3093 中无效) 。
Src_Addr_Type	固定为 0。
Dst_Addr_Type	固定为 0。
Dst_Rls_Ind	固定为 0。
Nonce_Len	固定为 11 (在 Hi3093 中无效) 。
HUK	选择 HUK。 0: 使用普通 key; 1: 使用 HUK。 (在 Hi3093 中固定为 0)
CI_GEN	机密运算 IV 的获取方式。



说 明	
	0x0: 非 GCM 算法时, 从 DDR 读取完整 IV; 0x0: GCM 算法时, 从 DDR 读取 salt+随机数, 用于组装 IV; 0x1: 加密时, 从 TRNG 读取随机数, 用于组装 IV; 0x1: 解密时, 从 DDR 读取随机数, 用于组装 IV; 0x2: 在 Hi3093 中无效; 0x3: 保留。
AI_GEN	认证运算 IV 的获取方式。 0x0: 使用算法 spec 规定的 IV (内置于硬件中) ; 0x1: 从 DDR 读取完整 IV; 0x2: 在 Hi3093 中无效; 0x3: 在 Hi3093 中无效。
NO_PAD	认证运算时, 是否对数据末尾做 padding。 0x0: 做 padding。该选项用于普通 HASH 功能和长数据 HASH 功能的末段。表示数据全部结束; 本次任务的计算结果是 HASH 值; 0x1: 不做 padding。该选项用于长数据 HASH 功能的首段和中间段。它表示数据并未结束, 还有数据接续于下一个任务; 本次任务的计算结果只是 HASH 的中间结果。 其他: 保留。
Long_Auth_Data_Len	HASH 算法 padding 里的数据长度值, 单位: bit。 仅用于长数据 HSAH 功能的尾段, 表示(进行 Hash 运算)的这笔数据的总长度。SEC 默认该域低 3bit 为 0, 不做检查。
C_S	固定为 0。
{A_Alg, AKey_Len, MAC_Len}	A_Alg: 认证算法类型。 AKey_Len: 认证 key 长度, 单位: Word; MAC_Len: 认证 MAC 长度, 单位: Word; A_Alg, AKey_Len, MAC_Len 的合法配置如下: 0x0_00xx (17'b0000000_000000_xxxxx) : 在 Hi3093 中无效。 0x0_08xx (17'b0000001_000000_xxxxx) : SHA-256 (该算法



说 明	
	<p>MAC 值长度为 256bit，允许的编码空间为 0x0~0x8，其中配置为 0 表示要输出全长，配置效果等同于配置 0x8)；</p> <p>0x0_10xx (17'b000010_000000_xxxxx)：在 Hi3093 中无效</p> <p>0x0_18xx (17'b000011_000000_xxxxx)：在 Hi3093 中无效</p> <p>0x0_20xx (17'b000100_000000_xxxxx)：在 Hi3093 中无效</p> <p>0x0_28xx (17'b000101_000000_xxxxx)：SHA-512 (该算法 MAC 值长度为 512bit，允许的编码空间为 0x0~0x10，其中配置为 0 表示要输出全长，配置效果等同于配置 0x10)；</p> <p>0x0_30xx (17'b000110_000000_xxxxx)：(在 Hi3093 中无效)</p> <p>0x0_38xx (17'b000111_000000_xxxxx)：(在 Hi3093 中无效)</p> <p>0x0_80xx (17'b010000_xxxxxx_xxxxx)：(在 Hi3093 中无效)</p> <p>0x0_88xx (17'b010001_xxxxxx_xxxxx)：HMAC-SHA256 (该算法 Key 的长度为 0~512bit，允许的编码空间为 0x0~0x10；该算法 MAC 值长度为 256bit，允许的编码空间为 0x1~0x8)；</p> <p>0x0_90xx (17'b010010_xxxxxx_xxxxx)：在 Hi3093 中无效</p> <p>0x0_98xx (17'b010011_xxxxxx_xxxxx)：在 Hi3093 中无效</p> <p>0x0_A0xx (17'b010100_xxxxxx_xxxxx)：在 Hi3093 中无效</p> <p>0x0_A8xx (17'b010101_xxxxxx_xxxxx)：HMAC-SHA512 (该算法 Key 的长度为 0~1024bit，允许的编码空间为 0x0~0x20；该算法 MAC 值长度为 512bit，允许的编码空间为 0x1~0x10)；</p>



说 明	
	0x0_B0xx (17'b010110_xxxxxx_xxxxx) : 在 Hi3093 中无效 0x0_B8xx (17'b010111_xxxxxx_xxxxx) : 在 Hi3093 中无效 0x1_0083 (17'b100000_000100_00011) : 在 Hi3093 中无效 0x1_0084 (17'b100000_000100_00100) : 在 Hi3093 中无效 0x1_088x (17'b100001_000100_xxxxx) : 在 Hi3093 中无效 0x1_090x (17'b100001_001000_xxxxx) : 在 Hi3093 中无效 0x1_10xx (17'b100010_xxxxxx_xxxxx) : 在 Hi3093 中无效 0x1_188 (17'b100011_000100_00001) : 在 Hi3093 中无效 0x1_2081 (17'b100100_000100_00001) : 在 Hi3093 中无效 0x1_21xx (17'b100100_001000_xxxxx) : 在 Hi3093 中无效 其他: 保留。
{C_Alg, C_Mode, CKey_Len, C_Width, C_ICV_Len}	C_Alg: 机密算法类型。 0x0: 在 Hi3093 中无效; 0x1: 在 Hi3093 中无效; 0x2: AES; 0x4: 在 Hi3093 中无效; 0x5: 在 Hi3093 中无效; 其他: 保留。 C_Mode: 机密算法模式。 0x0: 在 Hi3093 中无效; 0x1: CBC; 0x4: CTR; 0x5: 在 Hi3093 中无效; 0x6: GCM; 0x8: 在 Hi3093 中无效; 其他: 保留。 CKey_Len: key 长度。 0x0: AES 运算时, 表示 128 位密钥长度;



说 明	
	<p>0x1: AES 运算时, 表示 192 位密钥长度;</p> <p>0x2: AES 运算时, 表示 256 位密钥长度;</p> <p>0x3: 在 Hi3093 中无效;</p> <p>其他: 保留。</p> <p>C_Width: 固定为 0;</p> <p>C_ICV_Len: GCM 算法时, MAC 值长度, 单位: Byte;</p> <p>机密算法{C_Alg, C_Mode, CKey_Len, C_Width, C_ICV_Len}的合法配置如下。</p> <p>0x0_0200 (20'b0000_0000_001_000_000000): 在 Hi3093 中无效; 0x0_1200 (20'b0000_0001_001_000_000000): 在 Hi3093 中无效;</p> <p>0x1_0200 (20'b0001_0000_001_000_000000): 在 Hi3093 中无效; 0x1_0600 (20'b0001_0000_011_000_000000): 在 Hi3093 中无效;</p> <p>0x1_1200 (20'b0001_0001_001_000_000000): 在 Hi3093 中无效; 0x1_1600 (20'b0001_0001_011_000_000000): 在 Hi3093 中无效; 0x2_0000 (20'b0010_0000_000_000_000000): 在 Hi3093 中无效; 0x2_0200 (20'b0010_0000_001_000_000000): 在 Hi3093 中无效; 0x2_0400 (20'b0010_0000_010_000_000000): 在 Hi3093 中无效; 0x2_1000 (20'b0010_0001_000_000_000000): AES-CBC-128; 0x2_1200 (20'b0010_0001_001_000_000000): AES-CBC-192; 0x2_1400 (20'b0010_0001_010_000_000000): AES-CBC-256; 0x2_4000 (20'b0010_0100_000_000_000000): AES-CTR-128; 0x2_4200 (20'b0010_0100_001_000_000000): AES-CTR-192; 0x2_4400 (20'b0010_0100_010_000_000000): AES-CTR-256; 0x2_50xx (20'b0010_0101_000_000_000000): 在 Hi3093 中无效; 0x2_52xx (20'b0010_0101_001_000_000000): 在 Hi3093 中无效; 0x2_54xx (20'b0010_0101_010_000_000000): 在 Hi3093 中无效; 0x2_60xx (20'b0010_0110_000_000_000000): AES-GCM-128 (GCM C_ICV_Len 可配置值为: 0x8、0x9、0xA、0xB、</p>



说 明	
	<p>0xC、0xD、0xE、0xF、0x10) ;</p> <p>0x2_62xx (20'b0010_0110_001_000_xxxxxx): AES-GCM-192 (GCM C_ICV_Len 可配置值为: 0x8、0x9、0xA、0xB、0xC、0xD、0xE、0xF、0x10) ;</p> <p>0x2_64xx (20'b0010_0110_010_000_xxxxxx): AES-GCM-256 (GCM C_ICV_Len 可配置值为: 0x8、0x9、0xA、0xB、0xC、0xD、0xE、0xF、0x10) ;</p> <p>0x4_8000 (20'b0100_1000_000_000_000000): 在 Hi3093 中无效; 0x5_8000 (20'b0101_1000_000_000_000000): 在 Hi3093 中无效; 0x5_8400 (20'b0101_1000_010_000_000000): 在 Hi3093 中无效;</p> <p>其他: 保留。</p>
Auth_Len	<p>当普通认证算法时, 认证报文长度。单位: Byte。</p> <p>当 AES-GCM 算法时, 表示 AAD 的长度 (注意: AAD 是认证运算的一部分长度, 并非全部长度)。单位: Byte。</p>
Cipher_Len	<p>机密报文长度, 单位: Byte。</p> <p>AES 的 CBC 模式支持到 16Byte 对齐, 其他算法 (模式) 支持到 Byte 对齐。</p>
IV_Offset	随机数的偏移值, 用于形成指针。
Auth_Source_Offset	认证报文的偏移值, 用于形成指针。
Cipher_Source_Offset	机密报文的偏移值, 用于形成指针。
CS_IP_Header_Offset	固定为 0。
CS_UDP_Header_Offset	固定为 0。



说 明	
Salt	算法的 Salt 值 CI_GEN=1 且执行 AES-GCM/AES-CTR 算法时用来与随机数一起组成 Nonce。AES-GCM 时长度为 4 Byte，AES CTR 时长度为 4 Byte。
Err_Vqi[9:0]	在 Hi3093 中无效。
Err_Vqi_Rsv	在 Hi3093 中无效。
Vqi[9:0]	在 Hi3093 中无效。
Vqi_Rsv	在 Hi3093 中无效。
Location	在 Hi3093 中无效。
Order_TagValue[23:0]	在 Hi3093 中无效。
Affinity_TagValue[23:0]	在 Hi3093 中无效。
Order_Type	在 Hi3093 中无效。
Order_Rsv	在 Hi3093 中无效。
AffInd	在 Hi3093 中无效。
Deal_ESP_AH	固定为 0。
Protocol_Type	固定为 0（在 Hi3093 中无效）。
Mode	固定为 0（在 Hi3093 中无效）。
IP_Type	固定为 0（在 Hi3093 中无效）。
Padding Length	固定为 0（在 Hi3093 中无效）。
Next Header	固定为 0（在 Hi3093 中无效）。
Auth_Key_Addr	认证 key 的指针。
Auth_IVIN_Addr	认证 IV 的指针。



说 明	
MAC_Addr	MAC 值的指针。
Cipher_Key_Addr	机密 key 的指针。
Cipher_IVIN_Addr	机密 IV 的指针。
Dst_Addr	新址回写的目标地址，按照 64Byte 对齐。

表36-8 WB_BD 域の説明

说明	
Done	BD 任务处理完成标志：该域在任何情况有效。 0：处理未完成； 1：处理已完成。
ICV	认证校验结果。 0x0：无认证校验； 0x1：认证校验通过； 0x2：认证校验失败； 0x3：SEC BD、源 buffer 检测错误、或处理任务时遇到总线 ERR； 其他：保留。



说明	
Flag	处理结果标记。 bit[0]认证运算标志： 0：认证运算未处理； 1：认证运算已处理； bit[1]机密运算处理标志： 0：机密运算未处理； 1：机密运算已处理； bit[2]固定为 0（在 Hi3093 中无效）。
Error_Type	错误信息：该域在任何情况有效。



说明	
	<p>0x0: 无错误;</p> <p>0x1: 在 Hi3093 中无效;</p> <p>0x2: BD 中 TLV 错误;</p> <p>0x3: 算法选择非法: 加解密算法选择{A_Alg, AKey_Len, MAC_Len}选择了保留值, 或者认证算法{C_Alg, C_Mode, CKey_Len, C_Width, C_ICV_Len}选择了保留值;</p> <p>0x4: 配置了保留值: 比如 Auth 域配置了保留值, 或者 Cipher!=0 时 DE 域配置了保留值;</p> <p>0x5: 任务过程中 (非读 BD 时) 遇到总线读 ERROR;</p> <p>0x6: 读取 BD 时遇到总线 ERROR 或者存储时遇到 ECC2bit ERR;</p> <p>0x7: 配置 0 长度的非法报文;</p> <p>0x8: BD 解析时的源 buffer 配置非法;</p> <p>0x9: 在 Hi3093 中无效;</p> <p>0x10: 加解密选择 CBC/ECB 算法时长度配置非法;</p> <p>0xA: 在 Hi3093 中无效;</p> <p>0xB: SAA 申请 buf 出错;</p> <p>0xC: buffer 链中实际存储的报文长度小于 SEC 计算所需要的报文长度;</p> <p>0xD: 在 Hi3093 中无效;</p> <p>0xE: 地址对齐方式非法;</p> <p>0xF: CI_GEN、AI_GEN 配置类错误。</p>



说明	
	0x11: CCM/GCM/GMAC 算法配置错误; 0x12: 长度参数错误; 0x13: BD4 时 IV_Offset 非法; 0x14: DE 配置类错误; 0x15: 在 Hi3093 中无效; 0x16: 在 Hi3093 中无效; 0x17: 在 Hi3093 中无效; 0x18: 在 Hi3093 中无效; 0x19: 长数据哈希配置非法; 0x1A: 源 buffer 配置异常 (长度解析不够) ; 0x1D: 在 Hi3093 中无效; 0x1F(在 Hi3093 中无效; 其他: 保留。
Warning_Type	警告信息。 0x0: 无警告; 0x1: 在 Hi3093 中无效; 其他: 保留。
MAC-I	当有认证运算时, 表示 SEC 计算生成的 MAC 值 (截取最低的 word 填写) 。 其他情况下无效。 用于认证校验失败时维测, 软件将它与 UE 发送过来的 MAC 进行比较。
Checksum_I	在 Hi3093 中无效。
SEC_TimeStamp	全局时间戳, 单位: ns。
Src_Addr	源 BD 指针。
Reserved	保留域。



WB_BD 的域表示任务的完成状态，也有些域填写了部分结果数据用于 debug。这些域是在“任务完成且找到了回写的地方且 BD.Done=1”时有效。

BD.Done

表示这个是 WB_BD，当 SEC 回写时把它设置为 1。但该域并非在任何情况下有效，只有在“任务完成且找到了回写的地方”时有效。详细请参考 36.5.18.3 查询结果章节。

重要：若任务是新址回写，则在发送 BD 前，要初始化新 buffer 的 WB_BD.Done=0。若任务是原址回写，由于 WB_BD.Done 复用了 BD.Cipher_Key_Addr[0]，无法初始化。

BD.ICV

各种情况按优先级排列如下：

- 当 BD 是透传包时，BD.ICV=0
- 当出现错误时，BD.ICV=3
- 当有认证运算时，BD.ICV=1 或 2
- 其余情况，BD.ICV=0

BD.Flag

各种情况按优先级排列如下：

- 当 BD 是透传包或出现错误时，BD.Flag=0
- 其余情况，
 当有认证运算时，BD.flag[0]=1
 当有机密运算或 hardcopy 时，BD.flag[1]=1

36.5.5.3 BD 域的依赖关系

BD 中的域之间有依赖关系。有些很明显，用户能轻易看出来，例如认证运算的参数只有在 BD.Auth!=0 时才有效。有些是设计所限，用户不能推测出来，在这里列出。

记号 $A \Rightarrow B$ ，读作 A 成立时必然 B 成立。

记号 $A \Leftrightarrow B$ ，读作 A 与 B 等价，或者 A 与 B 同时成立。

$BD.SEQ \text{ 有效} \Leftrightarrow BD.Auth \neq 0 \ \&\& \ (BD.Cipher == 1 \ || \ BD.Cipher == 2)$

$BD.DE \text{ 有效} \Leftrightarrow BD.Cipher \neq 0$



$BD.Cipher==3 \Rightarrow BD.DE=1$

$BD.Dst_addr \text{ 有效} \Leftrightarrow BD.DE \text{ 有效} \ \&\& \ BD.DE=1$

36.5.5.4 透传包处理

SEC 支持软件生成的透传包 ($BD.Cipher=0$, $BD.Auth=0$, $BD.C_S=0$) 的处理: 只读写 SEC BD, 不读写任何其他数据 (含 IVIN、KEY、MAC、报文等), 并经 TAG 保序。



36.5.6 存储结构

BD 的 WORD0-WORD7 用来连结 BD 和数据 buffer，指明数据在内存中的分布。数据 buffer 中，数据区域占据其中的一部分。

前文讲过辅助数据由 BD 中的指针来关联（参考 36.5.2 任务流程）。为简明起见，这里的描述和图都将它略去。

36.5.6.1 源数据 buffer 存储结构

源数据 buffer 是指包含了源数据的 buffer。

源数据 buffer 的起始地址是 BD.RefltslfAddr。

源数据区域的起始地址是 BD.RefltslfAddr+BD.DataOffset。

源数据区域的长度没有对应的量来指定，只要数据区域不超过源数据 buffer 的范围即可。（注意：BD.DataLength 无效）

BD 可以在源数据 buffer 之内或之外，二者通过指针关联。其分布如图 36-3 所示。

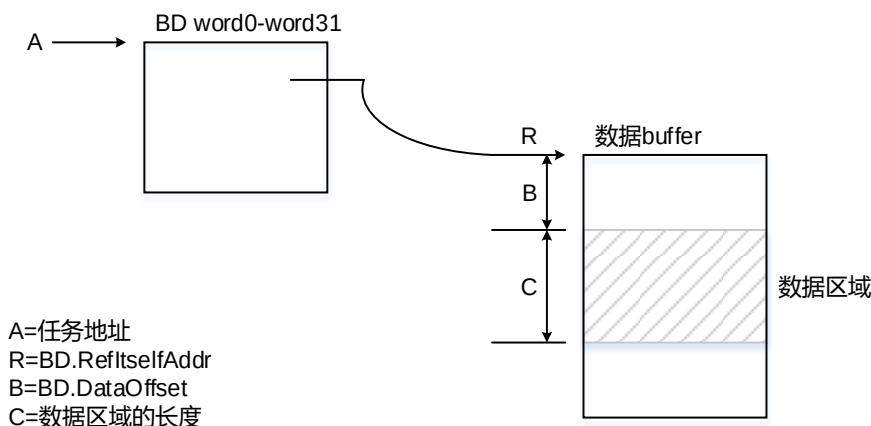
地址对齐的要求，详细请参考 36.5.8.7 对齐章节。

说明：源数据区域的起始地址和源数据 buffer 的起始地址不必相等，数据 buffer 内未占用的区域归用户使用。

须知

SEC 只知道源数据 buffer 的起始地址和源数据区域的起始地址，SEC 不知道源数据 buffer 的大小，不知道源数据区域的长度。在执行任务时 SEC 根据算法参数来读取数据。如果算法参数所指示的数据长度超出源数据 buffer 范围，SEC 并不知道越界，从而不会提示读取错误和结果错误。所以，软件在申请源数据 buffer 时要充分考虑其大小，保证容纳输入的内容。

图36-3 源数据 buffer 存储结构



36.5.6.2 目的数据 buffer 存储结构

原址回写的存储结构

如果结果是原址回写，WB_BD 和结果数据会被填写到原来的位置，分布保持不变。

软件在申请源数据 buffer 时要充分考虑其大小，能容纳所有输出的内容。如果数据 buffer 空间不够，SEC 输出结果会越界，而且不会提示错误，可能会造成破坏。

新址回写的存储结构

如果结果是新址回写，则新数据 buffer 内包括 WB_BD 和结果数据。其分布如图 36-4 所示。

WB_BD 的 WORD0-WORD7 由硬件自动填写，形成新的指针关系。

结果数据区域紧跟在 WB_BD 后面。所以 WB_BD.DataOffset=128。

新数据 buffer 的起始地址是 WB_BD.ReflSelfAddr。

结果数据区域的起始地址是 WB_BD.ReflSelfAddr+WB_BD.DataOffset。

结果数据区域的长度没有对应的量来指定，只要数据区域不超过新数据 buffer 的范围即可。（注意：WB_BD.DataLength 无效）

地址对齐的要求，详细请参考 36.5.8.7 对齐章节。

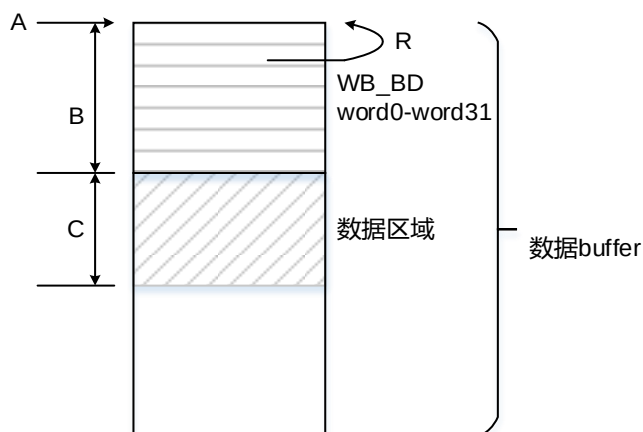
软件在申请新数据 buffer 时要充分考虑其大小，能容纳所有输出的内容。如果数据 buffer 空间不够，SEC 输出结果会越界，而且不会提示错误，可能会造成破坏。

须知

必须配置寄存器

SEC_BDFIFO_REG.SEC_BDF_CFG_SKIP_LENGTH[bdf_skip_length_cfg]=1, 才能得到该分布和正常工作。

图36-4 新数据 buffer 存储结构



WB_BD是回写BD
A=新buffer地址=BD.Dst_Addr
R=WB_BD.RefltSelfAddr
A=R
B=WB_BD.DataOffset
C=数据区域的长度

36.5.7 数据格式

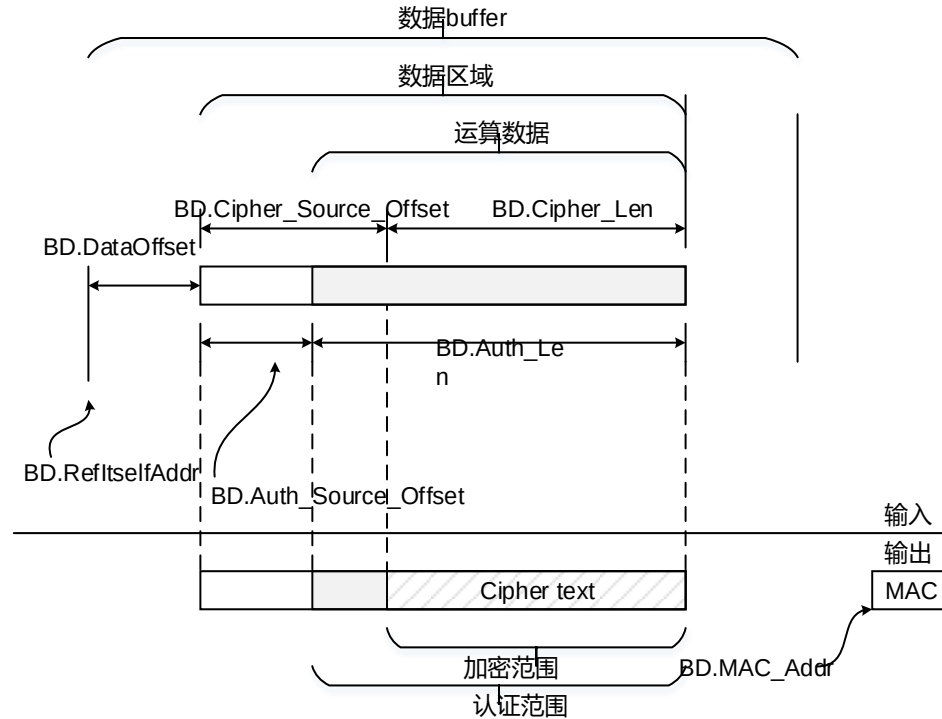
36.5.7.1 通用格式

普通算法的数据输入格式和输出格式。数据 buffer 中，数据区域占据其中的一部分。数据区域中，运算数据占据其中的一部分。通常讲的报文是指数据区域。

SEC 读取的范围是整个数据区域。虽然数据区域中有非运算数据，但它仍然会被读取；在回写时它也会被复制到目标地址。

认证生成+加密组合运算的报文解析如图 36-5 所示，其他算法依此类推。

图36-5 报文地址长度解析



须知

本图仅作示意，并不代表真实应用情况。例如，MAC 值可以放在数据 buffer 之内或之外。

定义 $\text{pkt} = \{\text{dummy}, \text{A0}, \text{IV}, \text{text}\}$

- pkt.dummy 是 pkt 的前面部分，不参与认证运算或机密运算。只会被回写。
如果应用中没有这部分，则不需要此域。
pkt.dummy 指针 = $\text{BD.Refitselfaddr} + \text{BD.Dataoffset}$
pkt.dummy 的长度 = $\text{BD.Auth_Source_Offset}$
- pkt.A0 只参与认证运算。如果没有认证运算，则不需要此域。
pkt.A0 指针 = $\text{BD.Refitselfaddr} + \text{BD.Dataoffset} + \text{BD.Auth_Source_Offset}$
- pkt.IV 是一个域，它是否存在与选取的算法有关。
若存在，它是否参与认证运算因算法而异，它的长度因算法而异。它不参与机密运算。它与回写随机数有关。详细见后面的内容。
pkt.IV 指针 = $\text{BD.Refitselfaddr} + \text{BD.Dataoffset} + \text{BD.IV_Offset}$
- pkt.text 是指 plaintext 或 ciphertext。如果没有机密运算，则不需要此域。



- 机密运算的范围是 pkt.text。
机密运算的起始点=pkt.text 指针
=BD.Refitselfaddr+BD.Dataoffset+BD.Cipher_Source_Offset
机密运算的长度=pkt.text 的长度=BD.Cipher_Len
- 认证运算的范围是{A0, IV, text}。
认证运算的起始点=pkt.A0 指针
=BD.Refitselfaddr+BD.Dataoffset+BD.Auth_Source_Offset
认证运算的长度=BD.Auth_Len

如果没有认证运算或机密运算时，可以认为整个报文都是 dummy，其他域不存在。
pkt={dummy}。

如果只有认证运算时，可以认为 pkt.IV 和 pkt.text 不存在。pkt={dummy, A0}。

如果只有机密运算时，pkt={dummy, IV, text}。

36.5.8 地址和长度

36.5.8.1 报文地址和长度

所有情况都按照下述来解析，不论是源数据或结果数据、原址回写或新址回写。对于结果数据，式中的 BD 用 WB_BD 代替。

数据区域的起始地址=BD.RefitselfAddr+BD.DataOffset

运算数据的地址以(BD.RefitselfAddr+BD.DataOffset)作为参考点来计算（并不是以 BD.RefitselfAddr 作为参考点）

认证运算的数据的起始地址
=(BD.RefitselfAddr+BD.DataOffset)+BD.Auth_Source_Offset

认证运算的数据的长度=BD.Auth_Len

机密运算的数据的起始地址
=(BD.RefitselfAddr+BD.DataOffset)+BD.Cipher_Source_Offset

机密运算的数据的长度=BD.Cipher_Len

36.5.8.2 回写报文的范围

SEC 读取的范围是整个数据区域。如果回写的结果数据涉及报文（例如机密运算和 hardcopy），则回写的范围是数据区域。SEC 读取整个范围，写到目标地址，并更新其内某些内容。详细请参考表 36-12。



36.5.8.3 Key 地址和长度

采用普通 key 时，由软件准备好放在 DDR 中，并在将指针配置到 BD 中。

key 地址的参考点是 0。

认证运算 key 的地址=BD.Auth_Key_Addr

认证运算 key 的长度=BD.AKey_Len

机密运算 key 的地址=BD.Cipher_Key_Addr

机密运算 key 的长度=BD.CKey_Len

36.5.8.4 IV 地址和长度

“从 DDR 里获取完整 IV”的地址参考点是 0。

认证运算 IV 的地址=BD.Auth_IVIN_Addr

机密运算 IV 的地址=BD.Cipher_IVIN_Addr

IV 的长度与具体算法有关，SEC 会决定读取长度。

36.5.8.5 随机数地址和长度

“从 DDR 里获取随机数”和“回写随机数到 DDR 里”的地址以
(BD.RefltslfAddr+BD.DataOffset BD.RefltslfAddr+BD.DataOffset) 作为参考点来计算（并不是以 BD.RefltslfAddr 作为参考点）。

随机数的地址=(BD.RefltslfAddr+BD.DataOffset)+BD.IV_Offset

随机数的长度与具体算法有关，SEC 会决定读取长度。

须知

BD.IV_Offset 不能顾名思义。虽然它的命名含有 IV，但其实是用于随机数。

36.5.8.6 MAC 地址和长度

MAC 的地址参考点是 0。

MAC 的地址=BD.MAC_Addr

MAC 的长度分几种情况：



- 普通认证算法，MAC 的长度=BD.MAC_Len。
- 当算法在机密那侧配置，又兼有认证功能时(例如 GCM)，MAC 的长度=BD.C_ICV_Len。BD.MAC_Len 无效。

36.5.8.7 对齐

表36-9 地址和长度的对齐要求

变量	对齐要求
BD 指针	64 Byte。
KEY、IV、随机数、MAC 指针	1 Byte。
BD.Auth_Source_Offset BD.Cipher_Source_Offset	1 Byte。
BD.Cipher_Len	AES-CBC 时，算法的 blocksize，16 Byte。
BD.Cipher_Len	其余情况，1 Byte。
BD.Auth_Len	长数据 HASH 的首段和中间段时，算法的 blocksize。
BD.Auth_Len	其余情况，1 Byte。
BD.Long_Auth_Data_Len	1 Byte。 该变量的单位是 bit。它表示的长度只能精确到 byte，LSB 3 bit 为 0。
BD.Dst_Addr	64 Byte。

36.5.9 Key

从安全角度来讲，有两种 key：普通 key 和 HUK。

普通 key 允许被安全世界软件访问到。软件将 key 放在 DDR 中，SEC 通过 BD 指针获取。

HUK 是安全级别更高的 key，它不允许被安全世界软件访问到，只有特定安全硬件能接触到它。HUK 的一种实现方法是放在 efuse 中，通过私有接口（硬连线/直接硬件接口）传递到使用它的模块。



SEC 能使用这两种 key，BD 中有相应的域来选择。每个任务都可以选择用一种 key。若选择 HUK，则认证和机密运算都使用 HUK（不会出现普通 key）。默认 HUK 的长度是 256 bit，当算法的 key 长度小于 256 bit 时，会截取低位使用；当算法的 key 长度大于 256 bit 时，会在高位填充 0 后使用。

BD 中相关的域：BD.HUK，BD.AKey_Len，BD.CKey_Len，BD.Auth_Key_Addr，Cipher_Key_Addr。

须知

Hi3093 SEC 取消了 HUK 功能，只使用普通 key。

36.5.10 IV

IV 也称为 IVIN，在不引起混淆的情况下可互换使用。

随机数属于 IV 的一部分。不过由于它内容较多，详细请参考 [36.5.11 随机数](#) 章节。

36.5.10.1 IV 的组成

每种算法定义了自己的 IV。它们之间差别很大；甚至同一种算法，在不同协议层次中对 IV 的定义和称呼也不尽相同。[表 36-10](#) 列出 IV 格式和主要成分，以方便用户查阅。要了解 IV 更详细的信息，请参考各个算法的 spec。

表36-10 IV 的成分和格式

算法	IV 格式	成分，及其在 IV 中的长度、位置、属性		
		随机数	Salt	Counter
AES_CBC	{随机数}	16B random/ unpredictable	-	-
AES_CTR	{Nonce, 随机数, Counter}	8B unique per key	4B nonce/ unpredictable (nonce 来自 BD.Salt)	4B 0x0000_0001
AES_GCM	软件角度的 IV：{salt, 随机数}	8B unique per key	4B random/ unpredictable	4B 0x0000_0001



算法	IV 格式	成分，及其在 IV 中的长度、位置、属性		
		随机数	Salt	Counter
	硬件角度的 IV: {salt, 随机数, Counter }			

36.5.10.2 IV 的地址和长度

关于 IV 的地址和长度解析，请参考 36.5.8.4 IV 地址和长度 章节。

关于随机数的地址和长度解析，请参考 36.5.8.5 随机数地址和长度 章节。

36.5.10.3 IV 的获取

由于不同算法的 IV 组成很不同，而且应用上有多种需求，SEC 通过 BD.AI_GEN 和 BD.CI_GEN 定义了很多种 IV 的获取方式。下面分情况逐一描述。

认证算法的 IV，有以下几种产生和获取方式，配置 BD.AI_GEN。

- 使用算法 spec 规定的 IV。
像 MD5、SHA 这些算法 spec 里都定义了 IV。SEC 的硬件内置了这些 IV，能直接使用，无需软件准备这些 IV。
它可以被选项“从 DDR 读取完整 IV”取代，只要软件将算法 spec 规定的 IV 放在 DDR 中即可。
- 从 DDR 读取完整 IV。
软件组装好 IV 放在 DDR 中，配置指针 BD.Auth_IVIN_Addr 指向之。SEC 从该地址读取。

机密算法的 IV，有以下几种产生和获取方式，配置 BD.CI_GEN。

- 从 DDR 读取完整 IV。
软件组装好 IV 放在 DDR 中，配置指针 BD.Cipher_IVIN_Addr 指向之。SEC 从该地址读取。

须知

该选项给非 GCM 算法使用。

- 从 TRNG 读取随机数，用于组装 IV。

有些算法的 IV 包含多种成份，这些成份来源各异。这个选项表示 SEC 从 TRNG 中获取随机数，将它和其他成分（可能来自 BD 的域、报文）一起组装成完整 IV。

- 从 DDR 读取随机数，用于组装 IV。

该选项与“从 TRNG 读取随机数”类似。这个选项表示 SEC 从 DDR 中获取随机数，将它和其他成分（可能来自 BD 的域、报文）一起组装成完整 IV。

随机数指针请参考 36.5.11.3 随机数的地址和长度章节。

- 从 DDR 读取 salt+随机数，用于组装 IV。

该选项与“从 DDR 读取随机数”类似。仅用于 GCM 算法。软件组装好 salt+随机数放在 DDR 中，配置指针 BD.Cipher_IVIN_Addr 指向之。SEC 从该地址读取。

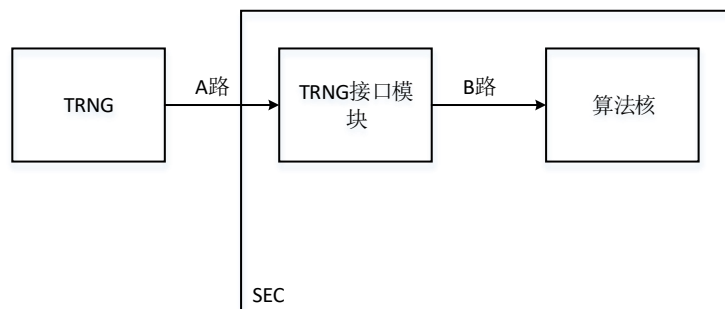
36.5.11 随机数

36.5.11.1 申请随机数

有些算法用到随机数。随机数可由用户提供或从 TRNG 获取。本章节描述从 TRNG 获取随机数。

TRNG 为系统中很多模块和软件提供服务，响应需要一定时间。如果 SEC 算法核有大量任务同时申请随机数，可能会出现超时。为此，SEC 采用缓存来减少延时。SEC 内部有一个 TRNG 接口模块，负责向外部 TRNG 读取随机数并缓存在本地。当算法核要使用随机数时，算法核向它请求随机数。如图 36-6 所示的 A 路和 B 路。

图36-6 申请随机数路径



TRNG 接口模块有使能开关。本地的缓存空间可以存放 20 个随机数。在使能后，它开始预取。它还有 DFX 功能，用于 flush 缓存内的随机数。



A 路和 B 路都会超时，都设置了超时中断。B 路出现超时，原因是 TRNG 接口模块的缓存没有随机数了，源头的原因是 A 路申请不到随机数。哪个超时先发生，取决于两路的超时阈值设置。

A 路和 B 路互相独立，即两路的发起申请和超时互不依赖。

A 路超时有使能开关。

- 当 TRNG 接口模块使能时，允许 A 路产生超时。
- 当 TRNG 接口模块不使能时，不允许 A 路产生超时。但并不阻止 B 路产生超时。

B 路超时有使能开关。

- 当使能时，在出现超时，SEC 会(1)产生中断，(2)取消申请，用任意数据代替该随机数并继续执行运算。WB_BD 不会标记错误。任务的结果数据到达对方后，能正确的解密和认证校验。在安全性方面，这个替代的随机数符合 AES-GCM/AES-CTR 算法的 IV 要求，但不符合 AES-CBC 算法的 IV 要求。所以当任务中有 AES-CBC 算法不使能该开关。
- 当不使能时，SEC(1)不会产生中断，(2)算法核保持申请，直到获得随机数为止。
- 当 TRNG 接口模块不使能时，SEC 提供了一个 DFX 功能，用于 flush TRNG 接口模块的缓存。flush 后缓存清空，要重新获取随机数后才有得用。这个功能是为谨慎起见，以防在系统复位后初期生成的随机数并不符合要求（可能是系统原因，也可能是 TRNG 原因），这个功能可以将这段时间内获取的随机数丢弃不用。操作流程请参考 [36.6.8 Flush 随机数缓存](#) 章节。

相关的寄存器：

- 使能开关
SEC_PF_REG.SEC_CONTROL.trng_en
SEC_PF_REG.SEC_CONTROL.channel_trng_req_timeout_en
- 超时中断
SEC_PF_REG.ACC_PF_ABNORMAL_INT_SOURC.req_trng_timeout_rint
SEC_PF_REG.ACC_PF_ABNORMAL_INT_SOURC.channel_req_rng_timeout_rint
- 超时阈值
SEC_PF_REG.SEC_REQ_TRNG_TIME_TH.sec_req_trng_time_th
SEC_PF_REG.SEC_CHANNEL_RNG_REQ_THLD.sec_channel_rng_req_thld
- flush 随机数缓存
SEC_PF_REG.DROP_RANDOM_CFG.drop_random_cfg
SEC_PF_REG.RANDOM_NUM_CNT.random_num_cnt



36.5.11.2 回写随机数

有些算法的 IV 包含随机数。按协议规定，在加密和认证生成时，获取随机数(用来组装 IV)，用来进行运算；将输出结果和随机数一齐发送给对方。在解密和认证校验时，提取随机数（用来组装 IV），用来进行运算。

将随机数发送给对方有多种方式，例如把它填入报文中、用其他通信通道传递等等。SEC 采用“将随机数填入报文中”的方式。它由软硬件协作完成。“将随机数填入报文中”的动作可由硬件或软件来做，依赖于 BD.AI_GEN 和 BD.CI_GEN 的配置，详细请参考 36.5.14 机密功能章节关于算法的各种配置。

如果“将随机数填入报文中”的动作由硬件完成，则它的过程大概如下。

- 报文中预留填写随机数的域。(请参考 36.5.7.1 通用格式章节)
- 在加密和认证生成时，软件先从报文中找到随机数的域，形成指针、配置 BD；SEC 获取随机数并开始运算，（根据指针）把随机数写到结果报文中。这称为回写随机数。
- 在解密和认证检验时，软件先从报文中找到随机数的域，形成指针、配置 BD；SEC（根据指针）提取随机数并开始运算。

在上述操作中，软件所做的事情是从报文中找到随机数的域、形成指针和配置 BD，此外无额外操作。

如果“将随机数填入报文中”的动作由软件完成，则它的过程大概如下。

- 报文中预留填写随机数的域。(请参考 36.5.7.1 通用格式章节)
- 在加密和认证生成时，软件先从报文中找到随机数的域，形成指针、配置 BD；在 SEC 执行完任务后，软件把随机数写到结果报文中。这称为回写随机数。
- 在解密和认证检验时，软件先从报文中找到随机数的域，形成指针、配置 BD；SEC（根据指针）提取随机数并开始运算。

在上述操作中，软件所做的事情是从报文中找到随机数的域、形成指针和配置 BD，还有回写随机数。

36.5.11.3 随机数的地址和长度

关于随机数的地址和长度解析，请参考 36.5.8.5 随机数地址和长度章节。



36.5.12 MAC

在应用中，认证功能的认证值有多种叫法，例如指纹、摘要、签名、hash 值、MAC (message authentication code)、ICV (integrity check value) 等。本文统一采用的叫法是 MAC。

通常 MAC 会截断使用。例如，SHA256 算法生成 256bit 的结果，用户根据应用中要求的认证强度，可适应地截取部分 LSB bit 来作为 MAC 值，例如 96bit。对方校验时也要相应地用截断的值来作为期望值。BD 中有域用于指定其长度。

在很多网络协议中，MAC 是报文 (packet) 的一个域，或在报文中间或在报文末尾。但 SEC 对 MAC 并无此限制；在 SEC 看来，MAC 是一个数据，可位于报文之内或之外。所以，如果应用场景不是某种协议的报文，则只要用指针将 BD 与 MAC 关联即可，不必特意指向报文的某个位置。

在认证生成时，MAC 是结果数据，由 SEC 回写。

在认证校验时，MAC 是输入、期望值、辅助数据，由用户准备好。

关于 MAC 的地址和长度，详细请参考 [36.5.8.6 MAC 地址和长度](#) 章节。

36.5.13 认证功能

表36-11 认证算法的属性摘要

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-256	$<2^{64}$	512	32	256
SHA-512	$<2^{128}$	1024	64	512

软件准备好算法参数，有些放在 BD 中，有些放在 DDR 中并由 BD 中的指针关联。认证生成时，计算结果为 MAC 值，输出到指针 BD.MAC_Addr 指定的位置。认证校验时，期望 MAC 值放在 DDR 中，指针配置在 BD.MAC_Addr；计算结果为通过或失败，填写在 WB_BD 的域中。

认证生成

要设置的 BD 域：

- BD.Auth
选择哪种认证功能。



- BD.SEQ
(可选) 用于认证+机密组合时设置算法顺序。详细请参考 [36.5.14.1 AES-CBC 加密](#) 章节。
- BD.Cipher=3 和 BD.DE=1 和 BD.Dst_Addr
(可选) 用于与 hardcopy 功能组合，详细请参考 [36.5.16 Hardcopy 功能](#) 章节。
- BD.HUK
指定选用哪种 key。
- BD.AI_GEN
选择认证 IV 的获取方式，详细请参考 [36.5.10.3 IV 的获取](#) 章节。
- {BD.A_Alg, BD.AKey_Len, BD.MAC_Len}
指定算法类型、key 长、MAC 长度。
BD.AKey_Len 依赖于 BD.A_Alg。普通 SHA 类算法没有 key，HMAC 类算法有 key。
- BD.Auth_Source_Offset 和 BD.Auth_Len
指定认证运算的报文参数。
- BD.MAC_Addr
MAC 指针。
- BD.Auth_Key_Addr
认证 key 指针，依赖于 BD.A_Alg。普通 SHA 类算法没有 key，HMAC 类算法有 key。
- BD.Auth_IVIN_Addr
认证 IV 指针，依赖于 BD.AI_GEN。

其中 BD.MAC_Addr 和 BD.MAC_Len 与输出结果有关。

认证校验

要设置的 BD 域与认证生成相同。

其中 BD.MAC_Addr 和 BD.MAC_Len 是输入参数。

长数据 HASH 功能还用到 BD.NO_PAD、BD.Long_Auth_Data_Len。详细请参考 [36.6.7 长数据 HASH](#) 章节。

36.5.14 机密功能

软件准备好算法参数，有些放在 BD 中，有些放在 DDR 中并由 BD 中的指针关联。结果报文输出到指定地方。机密算法都有 key。

要设置的 BD 域：



- BD.Cipher
选择哪种机密功能，BD.Cipher=1 或 BD.Cipher=2。
- BD.SEQ
(可选) 用于认证+机密组合时设置算法顺序。详细请参考 [36.5.14.1 AES-CBC 加密](#) 章节。
- BD.DE 和 BD.Dst_Addr
指定原址回写还是新址回写。
- BD.HUK
指定选用哪种 key。
- BD.CI_GEN
选择机密 IV 的获取方式，详细请参考 [36.5.10.3 IV 的获取](#) 章节和 [36.5.14.1 AES-CBC 加密~36.5.14.6 AES-GCM 解密](#) 章节。
- {BD.C_Alg, BD.C_Mode, BD.CKey_Len, BD.C_Width, BD.C_ICV_Len}
指定算法类型、模式、key 长、MAC 长度等。
- BD.Cipher_Source_Offset 和 BD.Cipher_Len
指定机密运算的报文参数。
- BD.Cipher_Key_Addr
机密 key 指针。

CI_GEN 提供了很多种用法，每种用法的差异很大。详细请参考 [36.5.14.1 AES-CBC 加密~36.5.14.6 AES-GCM 解密](#) 章节。这几个章节以 [36.5.7.1 通用格式](#) 的通用格式为基础，以认证+机密组合的角度来叙述，如果不使用组合，请相应调节。

36.5.14.1 AES-CBC 加密

输入报文按照下面的格式：

- pkt={dummy, A0, IV, text}
- 预留 pkt.IV 的位置。pkt.IV 的长度=16 Byte。
- pkt.IV 参与认证运算，不参与机密运算。不过，在输入报文中可以不填写它；只要算法的 IV 与 BD 关联了，硬件会去读取 IV、根据 pkt.IV 指针找到位置、用 IV 代替 pkt.IV、进行算法运算、输出到目的地 wb_pkt.IV。

输出报文，与输入报文有同样格式：

- wb_pkt={dummy, A0, IV, text}。
- wb_pkt.IV 由哪一方填写，依赖于 BD.CI_GEN。

BD.CI_GEN：



- 从 DDR 读取完整 IV。
软件组装好 IV 放在 DDR 中，位置可以在 pkt 之外或者 pkt.IV。IV 指针 =BD.Cipher_IVIN_Addr。
wb_pkt.IV 由软件填写。
- 从 TRNG 读取随机数，用于组装 IV。
wb_pkt.IV 由硬件填写。

配置满足下述约束关系：

- BD.Cipher_Len 是算法 blocksize(16 Byte)的整数倍。

36.5.14.2 AES-CBC 解密

输入报文按照下面的格式：

- pkt={dummy, A0, IV, text}。
- 预留 pkt.IV 的位置。pkt.IV 的长度=16 Byte。
- pkt.IV 参与认证运算，不参与机密运算。在输入报文中是否填写，与 BD.CI_GEN 选项有关。只要算法的 IV 与 BD 关联了，硬件会去读取 IV、根据 pkt.IV 指针找到位置、用 IV 代替 pkt.IV、进行算法运算、输出到目的地 wb_pkt.IV。

输出报文，与输入报文有同样格式：

- wb_pkt={dummy, A0, IV, text}。
- wb_pkt.IV 由软件填写。

BD.CI_GEN：

- 从 DDR 读取完整 IV。
软件组装好 IV 放在 DDR 中，位置可以在 pkt 之外或者 pkt.IV。IV 指针 =BD.Cipher_IVIN_Addr。
- 从 DDR 读取随机数，用于组装 IV。
随机数位置在 pkt.IV。

配置满足下述约束关系：

- BD.Cipher_Len 是算法 blocksize(16 Byte)的整数倍。

36.5.14.3 AES-CTR 加密

输入报文按照下面的格式：

- pkt={dummy, A0, IV, text}。



- 预留 pkt.IV 的位置。pkt.IV 的长度=8 Byte。
- pkt.IV 参与认证运算，不参与机密运算。不过，在输入报文中可以不填写它；只要算法的 IV 与 BD 关联了，硬件会去读取 IV、根据 pkt.IV 指针找到位置、用 IV 代替 pkt.IV、进行算法运算、输出到目的地 wb_pkt.IV。

输出报文，与输入报文有同样格式：

- wb_pkt={dummy, A0, IV, text}。
- wb_pkt.IV 由哪一方填写，依赖于 BD.CI_GEN。

BD.CI_GEN：

- 从 DDR 读取完整 IV。
软件组装好 IV 放在 DDR 中，位置在 pkt 之外。IV 指针=BD.Cipher_IVIN_Addr。
wb_pkt.IV 由软件填写。
- 从 TRNG 读取随机数，用于组装 IV。
nonce 来自 BD.Salt。
wb_pkt.IV 由硬件填写。

36.5.14.4 AES-CTR 解密

输入报文按照下面的格式：

- pkt={dummy, A0, IV, text}。
- 预留 pkt.IV 的位置。pkt.IV 的长度=8 Byte。
- pkt.IV 参与认证运算，不参与机密运算。在输入报文中是否填写，与 BD.CI_GEN 选项有关。只要算法的 IV 与 BD 关联了，硬件会去读取 IV、根据 pkt.IV 指针找到位置、用 IV 代替 pkt.IV、进行算法运算、输出到目的地 wb_pkt.IV。

输出报文，与输入报文有同样格式：

- wb_pkt={dummy, A0, IV, text}。
- wb_pkt.IV 由软件填写。

BD.CI_GEN：

- 从 DDR 读取完整 IV。
软件组装好 IV 放在 DDR 中，位置在 pkt 之外。IV 指针=BD.Cipher_IVIN_Addr。
- 从 DDR 读取随机数，用于组装 IV。
nonce 来自 BD.Salt。
随机数位置在 pkt.IV。



36.5.14.5 AES-GCM 加密

输入报文按照下面的格式：

- pkt={dummy, AAD, IV, text}。
- pkt.A0 换了名称，叫做 pkt.AAD。
AAD 指针=BD.Auth_Source_Offset, AAD 的长度=BD.Auth_Len。
(此时 BD.Auth_Len 并非表示认证运算的长度，而是表示 AAD 的长度。硬件会求出认证运算的长度)
- 预留 pkt.IV 的位置。pkt.IV 的长度=8 Byte。
- pkt.IV 在 GCM 中是指随机数。pkt.IV 不参与认证运算，不参与机密运算。不过，在输入报文中可以不填写它；只要算法的 IV 与 BD 关联了，硬件会去读取 IV、根据 pkt.IV 指针找到位置、用 IV 代替 pkt.IV、进行算法运算、输出到目的地 wb_pkt.IV。

输出报文，与输入报文有同样格式：

- wb_pkt={dummy, AAD, IV, text}。
- wb_pkt.IV 由哪一方填写，依赖于 BD.CI_GEN。

GCM 算法包含了认证和机密两种运算，它是 EtA(Encrypt then Authenticate)。不过它的 BD 配置主要放在机密那侧的参数，用到少量认证那侧的参数。需要如下参数：

- 选择算法 GCM。
BD.Cipher=1, 配置{BD.C_Alg, BD.C_Mode, BD.CKey_Len, BD.C_Width, BD.C_ICV_Len}。
- BD.Auth=0。BD.SEQ 无效。BD.MAC_Len 无效，由 BD.C_ICV_Len 取代。
- BD.MAC_Addr
MAC 指针
- BD.Cipher_Key_Addr
认证运算和机密运算共用一个 key。BD.Auth_Key_Addr 无效。
- IV
认证运算和机密运算共用一套 IV。配置依赖于 BD.CI_GEN。

BD.CI_GEN：

- 从 TRNG 读取随机数，用于组装 IV。
Salt 值来自 BD.Salt。
wb_pkt.IV 由硬件填写。



- 从 DDR 读取 salt+随机数, 用于组装 IV。
salt+随机数位置在 pkt 之外, 指针=BD.Cipher_IVIN_Addr
wb_pkt.IV 由软件填写。

配置满足下述约束关系:

- BD.Auth=0, 否则视为 BD 错误。
- AAD 长度和机密运算报文长度不能同时为 0。否则视为 BD 错误。
- AAD 和机密运算报文不能重叠, 即
 $BD.Auth_Source_Offset + BD.Auth_Length \leq BD.Cipher_Source_Offset$ 。否则视为 BD 错误。

36.5.14.6 AES-GCM 解密

输入报文按照下面的格式:

- $pkt = \{dummy, AAD, IV, text\}$ 。
- pkt.A0 换了名称, 叫做 pkt.AAD。
AAD 指针=BD.Auth_Source_Offset, AAD 的长度=BD.Auth_Len。
(此时 BD.Auth_Len 并非表示认证运算的长度, 而是表示 AAD 的长度。硬件会求出认证运算的长度)
- 预留 pkt.IV 的位置。pkt.IV 的长度=8 Byte。
- pkt.IV 是指 GCM IV 中的随机数。pkt.IV 不参与认证运算, 不参与机密运算。在输入报文中是否填写, 与 BD.CI_GEN 选项有关。只要算法的 IV 与 BD 关联了, 硬件会去读取 IV、根据 pkt.IV 指针找到位置、用 IV 代替 pkt.IV、进行算法运算、输出到目的地 wb_pkt.IV。

输出报文, 与输入报文有同样格式

- $wb_pkt = \{dummy, AAD, IV, text\}$ 。
- wb_pkt.IV 由软件填写。

GCM 算法包含了认证和机密两种运算, 它是 EtA (Encrypt then Authenticate)。不过它的 BD 配置主要放在机密那侧的参数, 用到少量认证那侧的参数。需要如下参数:

- 选择算法 GCM。
BD.Cipher=2, 配置{BD.C_Alg, BD.C_Mode, BD.CKey_Len, BD.C_Width, BD.C_ICV_Len}
- BD.Auth=0。BD.SEQ 无效。BD.MAC_Len 无效, 由 BD.C_ICV_Len 取代。



- BD.MAC_Addr
MAC 指针。
- BD.Cipher_Key_Addr
认证运算和机密运算共用一个 key。BD.Auth_Key_Addr 无效。
- IV
认证运算和机密运算共用一套 IV。配置依赖于 BD.CI_GEN。

BD.CI_GEN

- 从 DDR 读取随机数，用于组装 IV。
随机数位置在 pkt.IV。
Salt 值来自 BD.Salt。
- 从 DDR 读取 salt+随机数，用于组装 IV。
salt+随机数位置在 pkt 之外，指针=BD.Cipher_IVIN_Addr

配置的约束关系与加密时相同。

36.5.15 认证+机密组合

当任务配置有认证+机密算法时，可以认证算法先行或者机密算法先行，BD.SEQ 指定算法的顺序。

当只有一种算法时，认证或机密，BD.SEQ 无效。关于 BD.SEQ 的依赖关系，详细请参考 [36.5.5.3 BD 域的依赖关系](#) 章节。

须知

认证+机密只允许这些组合：认证生成+加密，认证校验+解密。其余的组合会导致硬件行为不确定，请勿使用。

36.5.16 Hardcopy 功能

机密运算总是要输出报文，所以它天然具有将（结果）报文写到指定地方的功能。但认证运算只是输出 MAC 值或检验结果。有些时候，用户希望在认证同时能将原报文搬到新的地方。所以，为了使功能更灵活，引入了 hardcopy 功能，用于以下情况：

- 无算法功能，BD.Cipher=3，实现数据 hardcopy，将原报文新址回写。
- 认证运算与 BD.Cipher=3 组合，实现数据 hardcopy，将原报文新址回写。



关于新址回写，详细请参考 [36.5.18.1 回写的目的地](#) 章节。关于 BD.DE 的依赖关系，详细请参考 [36.5.5.3 BD 域的依赖关系](#) 章节。

须知

BD.Cipher=3 $\Rightarrow$ BD.DE=1，hardcopy 禁止将原报文原址回写。

说明

机密算法与 BD.DE 组合，将结果报文写到指定地方。机密运算输出的数据不同于原报文，可能原址回写也可能新址回写，所以不叫 hardcopy，以免令人困惑。机密运算无法与 BD.Cipher=3 组合，因为机密算法本身要配置为 BD.Cipher=1 或 2。因此通过 BD.DE 来指定目标位置。

关于 hardcopy 的数据的范围，请参考 [36.5.8.2 回写报文的范围](#) 章节。

36.5.17 任务生成

关于任务流程，在前文已经介绍过，详细请参考 [36.5.2 任务流程](#) 章节。

前文介绍到，SEC 有好几个功能，而且可以组合。但并不是全部组合都能正常工作。SEC 支持的功能组合如 [表 36-12](#) 所示。

关于添加任务的操作流程，详细请参考 [36.6.6 执行任务](#) 章节。

36.5.18 任务结束

36.5.18.1 回写的目的地

当任务结束时，SEC 回写结果数据和 WB_BD。按目的地址来分，回写分为原址回写和新址回写。

须知

算法运算过程与回写结果数据同时进行。WB_BD 发生在任务后期，但 WB_BD 的完成时刻与回写结果数据的完成时刻（即真正到达目的内存空间）没有顺序关系。

原址回写是把输出写到原来的内存空间。生成 BD 时设置 BD.DE。存储结构请参考 [原址回写的存储结构](#) 小节。

新址回写是把输出写到新的内存空间。软件要预先申请 buffer（称为新 buffer），生成 BD 时把指针填写到 BD.Dst_Addr 和设置 BD.DE；SEC 会按照指示回写到新 buffer。存储结构请参考 [新址回写的存储结构](#) 小节。



回写的目的地按优先级排列如下：

- 如果任务是透传包，则 WB_BD 按照原址回写。忽略 BD.DE。无需回写结果数据。
- 如果任务正确执行完成（没有出现错误，包括 BD 解析和执行过程中），则按照 BD.DE 指示来回写。
- 如果任务有错误，
 - 如果在 BD 解析时检测到配置错误，则 WB_BD 按照原址回写，无需回写结果数据。
 - 如果在 BD 解析时无错误，但在处理过程中遇到某些错误（随机数申请（B 路）超时、AXI 总线读写错误），则 WB_BD 和结果数据按照 BD.DE 指示。
 - 如果在 BD 解析时无错误，但在处理过程中遇到其他错误，则 WB_BD 按照原址回写，结果数据按照 BD.DE 指示。

36.5.18.2 回写的内容

不同算法和算法组合会输出不同的数据内容，具体见表 36-12。

表36-12 回写结果数据的类型

功能组合			结果数据	报文相关的回写范围	备注
认证运算	机密运算	Hardcopy			
认证生成	-	-	MAC	-	-
认证校验	-	-	-	-	校验结果在 WB_BD
认证生成	-	Hardcopy	MAC 复制报文	复制整个数据区域	-
认证校验	-	Hardcopy	复制报文	复制整个数据区域	校验结果在 WB_BD
认证生成	加密	N/A	MAC 密文	复制整个数据区域，并更新机密报文那部分	-



功能组合			结果数据	报文相关的回写范围	备注
认证运算	机密运算	Hardcopy			
认证校验	解密	N/A	明文	复制整个数据区域，并更新机密报文那部分	校验结果在 WB_BD
-	加密	N/A	密文	复制整个数据区域，并更新机密报文那部分	-
-	解密	N/A	明文	复制整个数据区域，并更新机密报文那部分	-
-	N/A	Hardcopy	-	复制整个数据区域	

36.5.18.3 查询结果

SEC 有寄存器用于查询任务是否完成，之后再找出 WB_BD 在何处。具体操作流程请参考 36.6.6 执行任务章节。

注意：用户不能通过查询 WB_BD 来获知任务是否完成，而是在得知任务完成后再查询 WB_BD。

为什么要找出 WB_BD 在何处？因为 WB_BD 写到哪个地方受很多因素影响，详细情况请参考 36.5.18.1 回写的目的地章节。

如何找出 WB_BD 在何处？步骤如下：

- 步骤 1 若任务是原址回写，WB_BD 在原址处。
(原址处的 WB_BD.Done 复用了 BD.Cipher_Key_Addr[0]，无法初始化。因此不能通过查询 WB_BD 来获知任务是否完成。)
- 步骤 2 重要：若任务是新址回写，则在发送 BD 前，要初始化新 buffer 的 WB_BD.Done=0。
虽然任务指定为新址回写，但最终发生了新址回写还是原址回写，受很多因素影响。



当用户得知任务完成后，要找出 WB_BD 到底写到了哪里。这时读取一下新址处的 WB_BD.Done，就能帮助判断出来。

----结束

36.5.19 低功耗

SEC 提供了动态时钟门控 (dynamic clock gating) 功能。寄存器中有开关使能之。在使能后，当 SAA 无任务处理时，SEC 会自动关闭其时钟；当有任务到达 SAA 时，又会自动打开其时钟。

相关寄存器：

SEC_PF_REG.SEC_CONTROL.clk_gate_enable

36.5.20 中断控制

36.5.20.1 中断源

SEC 向外输出共 10 个中断信号，仅作为异常中断使用，包括 1 个 PF 中断信号、8 个 VF 中断信号和 1 个安全中断信号。

PF 中断信号共有 6 个中断源，中断源信息可在 ACC_PF_ABNORMAL_INT_SOURCE 寄存器 (PF 0x0208 地址) 查询，中断源按 bit 写 1 清 0。

(Hi3093 不涉及) VF 中断信号共有 10 个中断源，中断源信息可在 ACC_VF_ABNORMAL_INT_SOURCE 寄存器 (VF 0x0208 地址) 查询，中断源按 bit 写 1 清 0。

安全中断信号共有 5 个中断源，中断源信息可在 SEC_BDF_RINT 寄存器 (BD FIFO 0x0204 地址) 查询，中断源按 bit 写 1 清 0。

具体中断源，请参见《Hi3093 寄存器手册》描述。

36.5.20.2 中断处理

详细请参考《Hi3093 中断向量表》。



36.6 操作流程

36.6.1 BD FIFO 接口

BD FIFO 寄存器 SEC_BDF_EN（偏移地址为 0x0000）提供了 BD FIFO 任务入口和出口控制，其中 bdf_push_en 控制了 BD FIFO 的任务入口，bdf_pop_en 控制了 BD FIFO 的任务出口，详细配搭方式如下：

表36-13 BD FIFO 任务控制组合

bdf_pop_en	bdf_push_en	组合结果
0	0	关闭 BD FIFO 的任务出口和入口： 1、禁止安全 CPU 向 BD FIFO 下发任务； 2、不允许 SAA 获取新的 BD FIFO 任务； 3、SAA 可以返回已经完成的 BD FIFO 任务。 典型应用场景：SEC 安全复位，BD FIFO 任务回收。
0	1	仅关闭 BD FIFO 任务出口： 1、允许安全 CPU 向非满的 BD FIFO 下发任务； 2、不允许 SAA 获取新的 BD FIFO 任务； 3、SAA 可以返回已经完成的 BD FIFO 任务。 典型应用场景：NA。
1	0	仅关闭 BD FIFO 任务入口： 1、禁止安全 CPU 向 BD FIFO 下发任务； 2、允许 SAA 获取新的 BD FIFO 任务； 3、SAA 可以返回已经完成的 BD FIFO 任务。 典型应用场景：任务 BD FIFO 任务排空。



bdf_pop_en	bdf_push_en	组合结果
1	1	打开 BD FIFO 的任务出口和入口： 1、允许安全 CPU 向 BD FIFO 下发任务； 2、允许 SAA 获取新的 BD FIFO 任务； 3、SAA 可以返回已经完成的 BD FIFO 任务。 典型应用场景：安全运行。

36.6.2 模块复位

SEC 支持软件对整个 SEC 进行模块复位，该复位又称为软复位。下述各步骤需要顺次执行。

一、任务排空和资源回收（如果不需要排空任务和回收资源，可以直接跳到[步骤 4](#)；如果排空任务和回收资源失败，也请继续[步骤 4](#)，影响见本小节后的注意事项）：

步骤 1 先查询 SEC_PF_REG 的 BDF_EN_STATUS 寄存器（偏移地址 0x0A80），若查询到 0，即没开启 BD FIFO 模式，可直接跳到[步骤 3](#)；若查询到 1，则是已启用了 BD FIFO 模式，需要通知安全世界软件进行 BD FIFO 模式资源回收。

步骤 2 BD FIFO 模式资源回收的详细步骤请参见 [36.6.4 资源回收](#) 章节。

二、关闭总线接口

步骤 3 下发关闭总线接口请求：把 AXI MASTER DBW 的寄存器 AM_CTRL_GLOBAL（偏移地址 0x000）的[0]bit（域 ctrl_shutdown_req）都配置为 0x1，则 SEC 不再发出新的总线命令。

步骤 4 查询总线命令处理状态：循环查询 AXI MASTER DBW 的寄存器 AM_CURR_TRANS_RETURN（偏移地址 0x150），当 DBW 的 AM_CURR_TRANS_RETURN 寄存器都变为 0x3 时，表明之前发出的所有总线操作都已经被 master 接口处理完毕；如果超过 1ms 仍未查询到 DBW 的该寄存器值都变为 0x3，则表明总线接口上仍有未完成的总线操作，即关闭总线接口操作失败。

三、复位 SEC

步骤 5 产生复位。外部有时钟复位模块（CRG），它能产生复位信号。按照其要求操作流程即可复位 SEC。

----结束



须知

如果在复位前任务排空和资源回收步骤未完成，复位后可能会丢失资源；如果在复位前总线接口关闭未完成，复位后可能会挂死总线。

36.6.3 模块暂停

SEC 在 BDFOFP 工作方式的正常工作状态下，可以被整模块暂停。停止操作完成后，软件可重新配置一些基本配置，例如更改那些半动态配置寄存器（除非明确说明复位后只能配置 1 次的配置寄存器，否则都可以称之为“半动态配置寄存器”）。

步骤 1 参考 36.6.2 模块复位章节，直至执行完步骤 3。

步骤 2 此后，软件可更改 SEC 的一些基本配置。

步骤 3 完成配置更新后，软件可重新使能各个功能和重新工作。

----结束

36.6.4 资源回收

资源回收主要体现在未完成的 BD 任务需要被安全 CPU 及时回收。大致分为两个过程：

- 将入口处的任务排空。这些任务在 BD FIFO 内，未进入算法核执行。SEC 提供了 DFX 手段，软件能将这任务 POP 出来。
- 等待算法核内的任务排空。软件无法干预已进入算法核执行的任务，只能等待其结束。

具体步骤如下：

步骤 1 配置 SEC_BDF_EN 寄存器，偏移地址为 0x0000：配置为 0x0，关闭 BD FIFO 的入口和出口。跳至步骤 2。

步骤 2 查看 BD FIFO 内是否存有任务。

查询 SEC_BDF_FIFO_STATUS 寄存器，偏移地址为 0x128。

若域段 bdf_fifo_empty 为 0x0，表明 BD FIFO 未排空，跳至步骤 3；

若域段 bdf_fifo_empty 为 0x1，表明 BD FIFO 已排空，跳至步骤 5。



步骤 3 尝试将 BD FIFO 中的任务 POP 出来。

写 SEC_BDF_SW_POP_EN 为 1，偏移地址为 0x118：每写一次该寄存器产生一个 POP 动作。

须知

该寄存器不能顾名思义，它表示产生 POP 动作，而非 POP enable。

步骤 4 检查 POP 出来的任务。

查询 SEC_BDF_SW_POP_DAT_LOW 和 SEC_BDF_SW_POP_DAT_HIG，偏移地址为 0x120、0x124：POP FIFO 的数据存放于此。CPU 对此任务作相关处理。

(若数据为 0x5A5A5A5A_5A5A5A5A，表明软件 POP 动作与硬件 POP 动作出现竞争，从而软件 POP 动作失败，它不是一个任务指针)；

如果连续 1000 次查询出现 0x5A5A5A5A_5A5A5A5A，则资源回收失败，软件报错，跳至结束。

跳至[步骤 2](#)。

步骤 5 等待算法核内的任务排空。

循环查询 SEC_BDF_STATUS 寄存器，偏移地址为 0x0114。

若仍有任务，则继续等待和查询。

若已无任务，则资源回收成功，跳至结束。

该步骤限时 550ms，若超时则资源回收失败，跳至结束。

----结束

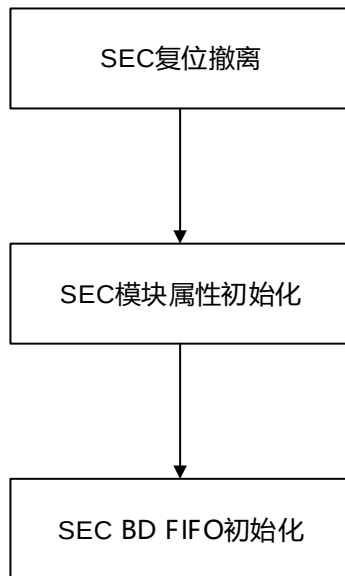
须知

软件执行资源回收的时候需要注意避免导致 BD FIFO 出现空读现象，否则将导致 SEC 产生 BD FIFO 空读异常原始中断。

36.6.5 初始化

无论是上电复位还是软复位，只要对 SEC 进行了模块复位，就需要进行模块初始化。这个过程涉及三个寄存器组。

图36-7 SEC 初始化分支流程图



36.6.5.1 PF 初始化

完成 PF 属性配置，操作寄存器组 SEC_PF_REG。

步骤 1 配置 BD_FIFO_EN_STATUS 寄存器，偏移地址 0x0A80：配置为 1，标明启用 BD FIFO 模式。

步骤 2 配置 SAA 门控统计窗口（如果不需要 SAA 的门控统计 DFX 功能，可以忽略该步骤）。

配置 ACC_GATING_WINDOW 寄存器，偏移地址 0x0620。

步骤 3 memory 相关功能配置。

配置 PF ACC_MEM_START_INIT 寄存器，偏移地址 0x0608：memory 自动初始化；

循环访问 PF ACC_MEM_INIT_DONE 寄存器，偏移地址 0x060C：检查 memory 自动初始化是否完成；如果超过 5us 还不能查询到 memory 自动初始化完成，则软件报错。

步骤 4 SEC 内部属性配置。

配置 PF SEC_CONTROL 寄存器，偏移地址 0x0800：配置 BD 大小端及各内部小功能的开关等；（加上描述，如果要用 TRNG 随机数的话，就打开 trng_en）



配置 PF SEC_BD_CS_PACKET_OST_CFG 寄存器, 偏移地址 0x0880: 配置 SEC 读取 BD 时的 Outstanding 深度;

配置 PF SEC_SAA_ENABLE 寄存器, 偏移地址 0x08A0: 配置 SAA 使能控制, 调度模块只给被使能的 SAA 分配任务。

步骤 5 SEC 中断、计数器门限相关配置。

配置 PF ACC_PF_ABNORMAL_INT_ENABLE 寄存器, 偏移地址 0x0200: 配置异常中断使能;

配置 PF FSM_MAX_CNT 寄存器, 偏移地址 0x0838: 配置内部关键状态机智能心跳计数上限值;

配置 SEC_CHANNEL_RNG_REQ_THLD, 偏移地址 0x0834: 配置随机数申请(B 路)超时阈值;

配置 SEC_REQ_TRNG_TIME_TH, 偏移地址 0x090C: 配置 SEC 向 TRNG 模块申请随机数的时间门限;

配置 PF SEC_PF_CNT_CLR_CE 寄存器, 偏移地址 0x0830: 配置公共计数器读清寄存器读清使能。

步骤 6 配置 AXI_MASTER_OOO 寄存器, 请参照《Hi3093 寄存器手册》。

----结束

36.6.5.2 VF 初始化

配置 VF 的业务相关的属性, 最后打开工作使能。操作寄存器组 SEC_PF_REG 和 SEC_VF_REG。

需要配置 VF 的原因: 虽然在应用上 BDFIFO 模式是独立于 VF 业务的模式, 但硬件实现时借用了 VF, 与之共享。所以需要使能一个 VF 来启动 BDFIFO 模式, 其编号等于 SEC_BDFIFO_REG.SEC_BDF_USE_VF_NUM.bdf_use_vf_num 的配置值。

步骤 1 超时相关配置。

配置 SEC_PF_REG.SEC_REQ_TRNG_TIME_TH 寄存器, 偏移地址 0x090C: 配置 SEC 申请 TRNG 的时间门限。

步骤 2 配置 SEC_PF_REG.SR_IOV_VF_OFFSET 寄存器, 偏移地址 0x012C: 为简单起见, 设置为 vfx_offset=x 即可。



步骤 3 配置 SEC_PF_REG.SR_IOV_CTL 寄存器，偏移地址 0x0108：在 vf_en_bitmap 域配置，使能一个 VF 用于 BDFIFO 模式，其余 VF 不使能。这是所选取的 VF。

配置 SEC_PF_REG.SEC_VF_WEIGHT 寄存器，偏移地址 0x08C0 0x08C4：配置对应 VF 处理任务的权重（为简单起见，用默认值即可）。

配置 SEC_PF_REG.SEC_VF_POWER 寄存器，偏移地址 0x0900 0x0904：配置 SEC 每个 VF 能调度的最大包个数；为简单起见，设置所选取的 vfx_msg_num=0xFF。

步骤 4 VF 计数器配置及中断配置。

配置 SEC_VF_REG.ACC_VF_ABNORMAL_INT_ENABLE 寄存器，偏移地址 0x0200：配置为不使能。

步骤 5 配置工作使能。

配置 SEC_VF_REG.VF_FUNC_EN 寄存器，偏移地址 0x0608：配置 VF 工作使能。

----结束

36.6.5.3 BD FIFO 初始化

配置 BD FIFO 相关的属性。操作寄存器组 SEC_BDFIFO_REG。

步骤 1 配置 BD FIFO 模式启用标识。

配置 BDF_EN_STATUS 寄存器，偏移地址 0x0004：配置为 1，标明启用 BD FIFO 模式。

步骤 2 BD FIFO 属性寄存器配置：

配置 SEC_BDF_USE_VF_NUM 寄存器，偏移地址 0x0014：配置选取哪个 VF 来处理 BDFIFO 任务。

配置 SEC_BDF_CNT_CLR_CE 寄存器，偏移地址 0x0020：配置 BD FIFO 模式寄存器读清使能。

配置 SEC_BDF_CFG_SKIP_LENGTH 寄存器，偏移地址 0x0028：配置 SEC 搬运数据到新 buf 时，需要预留的首片缓存 SKIP 区长度；为简单起见，设置 bdf_skip_length_cfg=1 即可。

步骤 3 BD FIFO 中断寄存器配置：



配置 SEC_BDF_INTENABLE 寄存器，偏移地址 0x0200：配置 BD FIFO 模式中断使能寄存器。

步骤 4 BD FIFO 任务开关配置：

配置 SEC_BDF_EN 寄存器，偏移地址 0x0000：配置 BD FIFO 模式使能，打开 BD FIFO POP 和 PUSH 使能。

----结束

36.6.6 执行任务

BD FIFO 的深度为 16。

SEC 的 BD FIFO 工作方式操作步骤如下：

- 步骤 1 安全世界软件查询 SEC_BDF_EN(0x0000)，若[bdf_push_en]为 0x0，需要配置为 0x1，若[bdf_pop_en]为 0x0，需要配置为 0x1。
- 步骤 2 安全世界软件生成任务 BD。如果是新址回写，则初始化新址处 WB_BD.Done=0。
- 步骤 3 安全世界软件查询 SEC_BDF_FIFO_STATUS(0x0128)，若[bdf_fifo_full]为 0x0，则可以通过配置 SEC_BDF_DATA_LOW (0x0100)和 SEC_BDF_DATA_HIG (0x0104) 下发任务 BD。

须知

写 SEC_BDF_DATA_HIG 触发通知“任务地址已填写好，SEC 有新任务到达”。所以要先填写 SEC_BDF_DATA_LOW 再填写 SEC_BDF_DATA_HIG。每个任务都要触发。

- 步骤 4 SEC 根据 BD FIFO 获取的 SEC 描述符地址，获取 SEC BD，并根据描述符的内容完成相关处理，然后完成 BD 的回写。
- 步骤 5 任务完成方式查询：当（一段时间内）BD FIFO 工作方式的任务都下发完成后，间隔若干时间（软件根据下发任务长度及数量自定义，至少要大于 0.1us），安全世界软件循环查询 SEC_BDF_PKG_DONE_CNT(0x0110)和 SEC_BDF_PKG_GET_CNT(0x010C)，在两个寄存器没发生卷绕情况下，两者值相等，表示 SEC 已完成之前所有 BD FIFO 工作方式下发的任务的处理——如果超过 5.5s 仍未查询到两寄存器值相等，软件报错。



步骤 6 安全世界软件找出 WB_BD 在哪里。

先查询 BD 的{T_Len, T_TagValue, Dsc_Len}。如果有错, 则 WB_BD 在原址, 格式如表 36-6 所示, 如果正确, 则继续下述查询。如果任务是原址回写, 则 WB_BD 在原址。如果任务是新址回写, 先查询新址处 WB_BD.Done 是否为 1。若是则 WB_BD 在新址处。若否则 WB_BD 在原址处。

步骤 7 安全世界软件根据 WB_BD 的信息作后续处理。

----结束

36.6.7 长数据 HASH

普通 HASH 用法是用一个 BD 为计算一笔数据的 HASH 值。长数据 HASH 是将这笔数据划分成多段, 每段生成一个 BD 任务, 最终计算出 HASH 值。前一段的计算结果作为后一段的输入, 所以要由软件对 BD 进行原子保序。

须知

长数据 HASH 功能属于认证功能, 它只能单独使用, 不能与机密算法或 hardcopy 组合。

步骤 1 软件把待处理的数据分成多段 (≥ 2)。

第一段称为首段, 最后一段称为尾段, 其余称为中间段。当总只有 2 段时, 无中间段。

首段和中间段的长度必须为算法 blocksize 的整数倍。每段的最大长度为 $(2^{24}-1)B$ 。

步骤 2 软件为每段数据生成 1 个 BD 任务。发送任务, 原子保序。

BD 配置与普通 HASH 算法相同, 除了下面有特别说明的域。

首段: 配置 BD.AI_GEN=0, BD.NO_PAD=1。

中间段: 配置 BD.AI_GEN=1, BD.NO_PAD=1。上一段的计算结果作为 IV。

末段: 配置 BD.AI_GEN=1, BD.NO_PAD=0。配置 BD.Long_Auth_Data_Len。上一段的计算结果作为 IV。

须知

BD.Long_Auth_Data_Len, 单位是 bit。它表示的长度只能精确到 Byte, LSB 3 bit 为 0。



首段和中间段的结果不能截断。(例如有些应用使用 SHA256 算法, 虽然算法输出结果为 256 bit, 但最终截取 192 bit 作为结果。用长数据 HASH 来计算时, 中间结果不能截断, 只有等最终结果才能截断)

步骤 3 等待当前 BD 任务处理完成。如果超过 550ms 仍未处理完毕, 软件报错。

步骤 4 待当前 BD 任务处理完并完成任务统计后, 软件判断是否已经把全部分段做完。如果已经做完, 则长数据 HASH 运算结束, 否则再回到[步骤 2](#)。

----结束

36.6.8 Flush 随机数缓存

TRNG 接口模块内有缓存用于存放随机数。该操作流程将缓存清空。

操作 SEC_PF_REG 寄存器组。

步骤 1 关闭 TRNG 接口模块的使能。

设置 SEC_CONTROL.trng_en=0

步骤 2 开启 flush 功能。

设置 DROP_RANDOM_CFG.drop_random_cfg=1。

步骤 3 循环查询缓存中余下的随机数数量, 直至为 0。

RANDOM_NUM_CNT.random_num_cnt

步骤 4 关闭 flush 功能。

设置 DROP_RANDOM_CFG.drop_random_cfg=0。

----结束

36.6.8.2 SEC 常见异常处理

常见的 PD、SEC BD、总线错误、ECC 等的处理方式, 分类列于[表 36-14](#)。



表36-14 SEC 异常分类列表

出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
BD 的 T_Len、 T_TagValue、 Dsc_Len 的 TLV 错误	识别成错误 BD， 上报 BD 错误中 断，（若 T_TagValue 合法 则按 T_TagValue 类型回写，否则按 BD 类型 2）回写 BD 时 ICV 域置 3，对报文不进行 处理	BD 解析	有， vf_bd_err_rint 中 断（VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE，偏移地址 0x0208）	0x2
Auth==3	识别成错误 BD， 上报 BD 错误中 断，回写 BD 时 ICV 域置 3，对报 文不进行处理	BD 解析	有， vf_bd_err_rint 中 断（VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE，偏移地址 0x0208）	0x4
Cipher!=0 时 DE==3	识别成错误 BD， 上报 BD 错误中 断，回写 BD 时 ICV 域置 3，对报 文不进行处理	BD 解析	有， vf_bd_err_rint 中 断（VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE，偏移地址 0x0208）	0x4
Src_Addr_Type != 0 或 1	识别成错误 BD， 上报 BD 错误中 断，回写 BD 时 ICV 域置 3，对报 文不进行处理	BD 解析	有， vf_bd_err_rint 中 断（VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE，偏移地址 0x0208）	0x4



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
NO_PAD!=0/1	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x4
^Cipher==1 时, 且非 ECB 模式, CI_GEN==3	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x4
^Auth==1 时, A_Alg, AKey_Length, Mac_Length 域值 组合不符合 0 的描 述	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x3
^Cipher==1 时, C_Alg, C_Mode, Ckey_Length, C_Width, T_Length 域值组 合不满足 0 的描述	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x3



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中断对应	ERROR_TYPE/ ARNING_TYPE
Cipher!=0, DE==1 时, Dst_Addr 为非 Cacheline(64B)对 齐地址	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xE
Cipher==3, DE==0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x14
选择了不合适的算 法	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x9
^Auth==1 时, Auth_Length==0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x7
^Cipher==1 时, Cipher_Length ==0 (BD4 的 gcm 算法除外)	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x7



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
Cipher==3, Auth==0 时, Cipher_Length ==0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x7
BD4, ^Cipher==1 时, 非 ECB, CI_GEN==2	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xF
^Cipher==1 时, 选择 AES-CBC AES-ECB 算法时 Cipher_Length 非 16Byte 对齐; 选 择 DES/3DES 算 法时 Cipher_Length 非 8Byte 对齐	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x10
BD4, ^Auth==1,GMAC 算法时, AI_GEN==0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xF



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
BD4, Auth==2,GMAC 算法时, AI_GEN==2	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xF
BD4, ^Auth==1, 非 GMAC 算法 时, AI_GEN! =0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xF
^Cipher==1, CCM/GCM 算法 时, ^Auth==1	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x11
^Auth==1, GMAC 算法时, ^Cipher==1	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x11
^Cipher==1, CCM 算法时, Auth_Len==0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x7



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中断对应	ERROR_TYPE/ ARNING_TYPE
^Cipher==1, GCM 算法时, Auth_Len==0, 且 Cipher_Len==0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x7
^Cipher==1, CCM/GCM 算法 时, Auth_Source_Off set+ Auth_Len> Cipher_Source_O ffset	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x12
^Cipher==1, CCM/GCM 算法 时, Auth_Source_Off set+ Auth_Len+32<Cip her_Source_Offse t	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x12
BD4, ^Cipher==1, ^Auth==1 时, Cipher_Source_O ffset< Auth_Source_Off set	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x12
BD4, ^Cipher==1, ^Auth==1 时, Cipher_Source_O ffset+Cipher_Len >Auth_Source_Of ffset+Auth_Len	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x12



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
BD4, 非 ECB, ^Cipher==1, CI_GEN==1 时, IV_Offset+IV_len > Cipher_Source_Offset	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x13
BD4, ^Auth==1, AI_GEN==2/3, GMAC 算法时, IV_Offset+IV_len > Auth_Source_Offset+Auth_Len	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x13
BD4, ^Auth==1, AI_GEN==2/3, GMAC 算法时, IV_Offset<Auth_S ource_Offset	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x13
BD5, Auth==2 时, NO_PAD==1	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x19



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
BD5, Auth==1, NO_PAD==1 时, sha160、sha224、sha256、md5 类的算法 Auth_Length 低 6bit 不为 0, 或 sha384、sha512、sha512/256、sha512/224 类的算法 Auth_Length 低 7bit 不为 0	识别成错误 BD, 上报 BD 错误中断, 回写 BD 时 ICV 域置 3, 对报文不进行处理	BD 解析	有, vf_bd_err_rint 中断 (VF 页签 ACC_VF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	0x19
BD5, ^Auth==1、AI_GEN==1、NO_PAD==0 时 (长数据 Hash 尾 BD), Long_Auth_Data_Len 低 3bit 不为 0	不上报错误 BD, SEC 按照低 3bit 为 0 进行 Hash_padding	BD 解析	不涉及	不涉及
BD5, ^Auth==1, AES_XCBC/AES_CMAC/GMAC 算法时, NO_PAD!=0	识别成错误 BD, 上报 BD 错误中断, 回写 BD 时 ICV 域置 3, 对报文不进行处理	BD 解析	有, vf_bd_err_rint 中断 (VF 页签 ACC_VF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	0x19



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
BD5, ^Auth==1, AI_GEN==2/3	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xF
BD5, ^Auth==1,GMAC 算法时, AI_GEN==0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xF
BD5, ^Auth==1, AES_XCBC/AES _CMAC 算法时, AI_GEN==1	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0xF
BD4, Cipher==1, CI_GEN==1, 非 ECB、CCM、 GCM 算法时, SAA 获取随机数 超时	使用上一次的随机 数+当前全局时戳 当做随机数, 上报 channel_req_rng_ timeout_rint 错误 中断, 后续报文走 正常处理流程, 不 识别为错误 BD (注: 环回测试可 能通过)	IVIN 模块	有, channel_req_rng_ timeout_rint 中断 (PF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	不涉及



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中断对应	ERROR_TYPE/ ARNING_TYPE
BD4, Cipher==0/3, Auth==1, AI_GEN==3, AES-GMAC 算法, SAA 获取随机数超时	使用上一次的随机数+当前全局时戳当做随机数, 上报 channel_req_rng_timeout_rint 错误中断, 后续报文走正常处理流程, 不识别为错误 BD (注: 环回测试可能通过)	VIN 模块	有, channel_req_rng_timeout_rint 中断 (PF 页签 ACC_VF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	不涉及
BD4, Cipher==1, CI_GEN==1, CCM、GCM 算法时, SAA 获取随机数超时	加解密/认证分别使用上一次的随机数或者上一次的随机数+当前全局时戳当做本次的随机数, 上报 channel_req_rng_timeout_rint 错误中断, 后续报文走正常处理流程, 不识别为错误 BD (注: 环回测试可能校验失败)	VIN 模块	有, channel_req_rng_timeout_rint 中断 (PF 页签 ACC_VF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	不涉及
Src_Addr_Type=1, NextAddr==0 时, 即单节点链, data_len 为 0 或者 RefItselfAddr 为 0	识别成错误 BD, 上报 BD 错误中断, 回写 BD 时 ICV 域置 3, 对报文不进行处理	BD 解析	有, vf_bd_err_rint 中断 (VF 页签 ACC_VF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	0x8



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中 断对应	ERROR_TYPE/W ARNING_TYPE
Src_Addr_Type== 0 时, RefltslfAddr==0	buffer 大小为 0: BD 回写时会识别 成错误 BD, 上报 buffer 大小为 0 中 断、buffer 错误中 断和 BD 错误中 断, ICV 域置 3	buffer 的解析	会上报 vf_chain_buff_len _all_0_rint 、 vf_chain_buff_err _rint 和 vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x1A
buffer 的 NextDataAddr 域 低 6Bit 不为 0	识别成错误 BD, 上报 BD 错误中 断, 回写 BD 时 ICV 域置 3, 对报 文不进行处理	BD 解析	有, vf_bd_err_rint 中 断 (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	0x8
读 BD 时, 发生总 线异常。	跟正常 BD 一样识 别, 但上报读总线 ERR 中断, 不过 可能对报文的部分 数据进行处理	总线	vf_axi_rresp_err_r int (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	不涉及
读 KEY、IVIN、 MAC、报文数据 时, 发生总线异 常。	跟正常 BD 一样识 别, 但上报读总线 ERR 中断, 不过 可能对报文的部分 数据进行处理	总线	vf_axi_rresp_err_r int (VF 页签 ACC_VF_ABNOR MAL_INT_SOUR CE, 偏移地址 0x0208)	不涉及



出错信息描述	处理策略	出错位置 (BD/数据)	是否有单独原始中断对应	ERROR_TYPE/ ARNING_TYPE
回写 BD、IVIN、MAC、报文数据时，发生总线异常，或者处理 buffer 时遇到 buffer 异常。	跟正常 BD 一样识别，但上报写总线 ERR 中断，不过可能对报文的部分数据进行处理	写总线返回异常。	vf_axi_bresp_err_rint (VF 页签 ACC_VF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	不涉及
MEM，如果遇到了 ECC 1bit 错误	纠错后按照正常流程处理；统计；上报 ecc1bit 中断	MEM	ecc_1bit_err_rint 中断 (PF 页签 ACC_PF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	不涉及
MEM，遇到 ECC 2bit 错误	立即关闭所有 OOO 接口（有寄存器开关控制），如果选择不关闭 OOO 接口，则错误现象未知；统计；上报 ecc2bit 中断	MEM	ecc_2bit_err_rint 中断 (PF 页签 ACC_PF_ABNORMAL_INT_SOURCE, 偏移地址 0x0208)	不涉及

- 以下情况称为透传包。
BD 中，Cipher==0 且 Auth==0 且 C_S==0;
- 透传包时，不检查 BD 内容，忽略一切异常配置；非透传包时，才检查 BD 异常、buffer 异常等。
- BD4 下含加解密、认证；BD5 下仅有认证，且 BD5 下不支持写到新地址、随机数拼 IVIN。
- BD4 下，IV_len 为随机数部分长度，ECB 模式为 0，DES/3DES CBC、AES CTR、CCM、GCM、GMAC 为 8，AES CBC 为 16。
- 设 buffer 链中实际存储的报文长度（Data_Length 之和）为 data_len_all。
- Cipher 为 0 时忽略 CI_GEN 配置，auth 为 0 时忽略 AI_GEN 配置。



- Cipher 为 0 时忽略 DE 配置。

36.6.8.3 BD MEM 可视化

软件可以通过寄存器接口，查询 SEC 内部缓存 BD 的 SRAM 中的内容；该 DFX 功能是为了方便原型验证、样片测试等非 EDA 场景的调试（商用时禁止使用该接口，以免引起不必要的错误中断）。

受限于 SRAM 的特性，需要在 SEC 处于空闲阶段再使用 SRAM 可视化，否则可能导致查询出非预期结果（甚至可能伴随错误中断上报）。

- BD SRAM 可视化具体使用方式：
 - 1. 配置 PF SEC_BDMEM_RD_CFG (偏移地址为 0xD1C);
 - 2. 通过读 PF SEC_BDMEM_RDATA(偏移地址为 0xD20~0xD3C)获得步骤 1 查询的 BD 内容。这 8 个寄存器会根据步骤 1 的配置而更新。注意：只有查询和配置操作相隔至少 20 拍 AHB Slave 时钟，才能查询出正确结果。
- 非预期结果排除：
 - 受限于 SRAM 处理机制，在 SEC 正常工作过程中，配置寄存器可视化操作可能会和 SEC 正常功能逻辑有冲突，此时查询结果有效状态位会置 0。
 - 在 SEC 工作期间进行查询操作，因为 SEC 在内部不断地存取 BD，所以即便是查询 8 个连续的地址，也可能查出的是不属于同一个 BD 的内容。



37 TRNG

37.1 概述

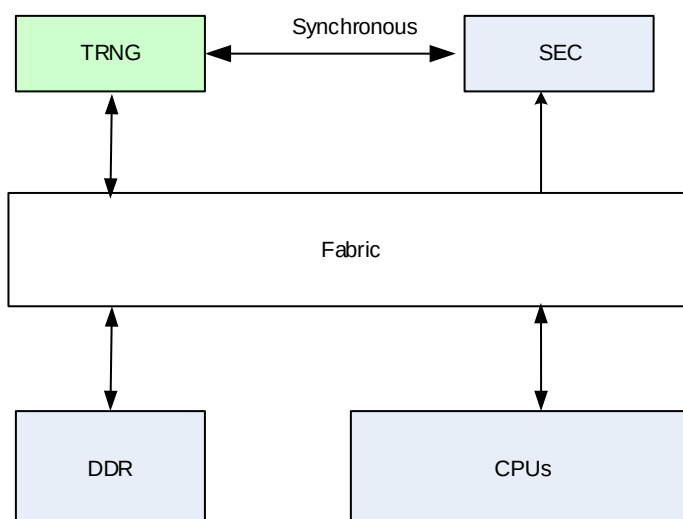
True Random Number Generator (TRNG) 是用来产生真随机数的单元。用于密钥对生成、SEC 加解密密钥以及 IV、以及其他系统中任何需要用到真随机数的场景。

在系统中，TRNG 模块会和如下模块进行交互，描述如下：

- CPU：CPU 从 TRNG 模块中获取真随机数，把真随机数读取到存入 DDR 中。
- DDRC 模块：TRNG 生成真随机数，CPU 通过 DDRC 把生成好的真随机数存储到 DDR 的安全空间。
- SEC 模块：SEC 的加解密使用的密钥，以及初始化向量 IV，是由 TRNG 模块生成的真随机数提供，和 SEC 之间采用私有接口，确保实现方便。

TRNG 的应用如图 37-1 所示。

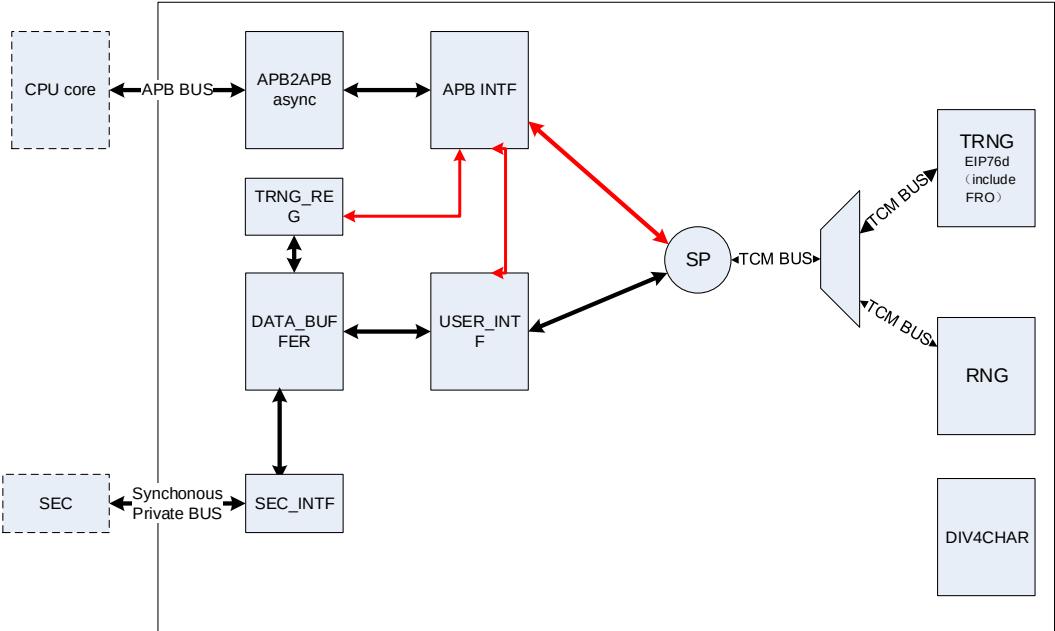
图37-1 TRNG 应用框图





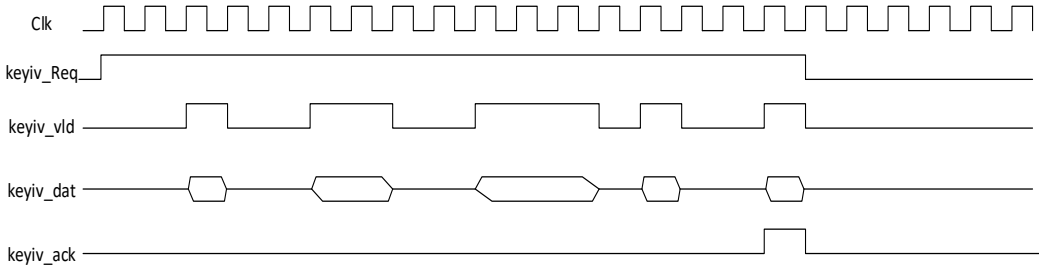
SEC 与 TRNG 的硬件接口是同步的。如图 37-2 所示。

图37-2 TRNG 整体内部逻辑框图



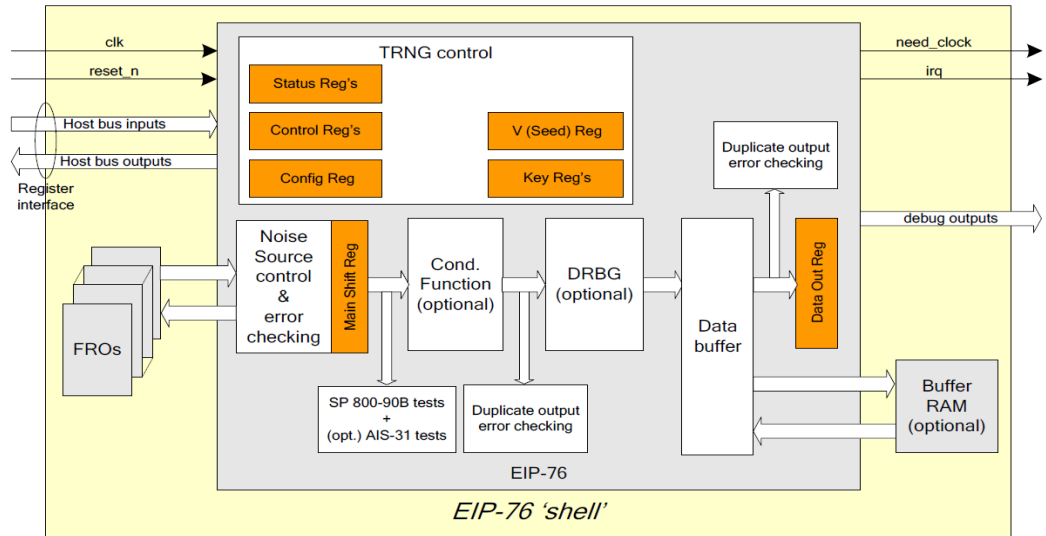
TRNG 和 SEC 的工作时钟保持一致，其私有接口时序如图 37-3 所示。

图37-3 接口时序



TRNG 使用了 IP 厂商 INSIDE 的真随机数 IP：EIP-76d。其随机物理源基于环形振荡器的物理抖动。IP 的逻辑框图如图 37-4 所示。

图37-4 EIP-76d 的内部逻辑框图



37.2 关键术语

表37-1 TRNG 关键术语描述

术语	描述
TRNG	True Random Number Generator, 真随机数产生器。
NRBG	Non-deterministic Random Bit Generator, 不确定性随机 bit 产生器。
DRBG	Deterministic Random Bit Generator, 确定性随机 bit 产生器。
NIST	National Institute of Standards and Technology, 国际标准和 技术协会。
SP	NIST 发布的特殊标准。
FIPS	Federal Information Processing Standard
FRO	Free Running Oscillator
Repetition Count	重复性计数。



术语	描述
Adaptive Propotion	自适应均衡。
Monobit test	测试 1 和 0 的数目是否相等。
Poker test	测试不交迭 4bit block 的出现。
Runs test	测试 0 或者 1 的数目，长度小于 6 或者大于等于 6。
Long runs test	搜索长度大约等于 34。
Noise source failure	噪声源失败测试。
AIS	Application notes and Interpretation of the Scheme (German IT standard series)
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
CTR_DRBG	Counter-Deterministic Random Bit Generator (SP 800-90A classification for EIP-76d DRBG)
IV	Initialization Vector
SHA	Secure Hash Algorithm
KAT	Known Answer Testing, 结果已知的测试，即已知正确的输出结果，检查经过计算的输出结果是否和已知正确的输出结果相等，相等即测试通过，否则测试失败。

37.3 特性

- 硬件实现的不确定性随机 bit 生成器 (NRBG) 并且与确定性随机 bit 生成器 (DRBG) 耦合。
- NRBG 中内嵌可选的硬件加工功能：
 - 与 SHA-256 兼容 ([SP 800 -90B draft] (和[FIPS140-2]/[FIPS140-3 draft]));
 - 加工功能把前面的输出追加到噪声源输入，平滑噪声源行为的变化。
- 支持确定性随机 bit 生成器 (DRBGs)：



- 与 AES-256 的 CTR_DRBG 兼容 ([SP 800-90A][FIPS 140-2][FIPS 140-3 draft]), 不带 DC_DF 功能 (需要加工功能, 但是允许在随机数生成中重新收集 re-seed 数据)。
- 可选的安全随机数数据缓存 buffer, 最大 16kbit 深度 (采用单口 SRAM, 最大为 512×32bit)。
- 支持 8 个 FRO。
- 支持 “随机性检测” 设计, 可配置:
 - 振荡器带有专门的测试电路, 用于检测对周期信号的锁定;
 - 在 NRBG 和 DRBG 上的重复输出数据的检测 ([FIPS 140-2][FIPS 140-3 draft]);
 - 硬件实现在噪声源的 “重复计数 (Repetition Count)” 和 “自适应均衡 (Adaptive Propotion)” 测试, 测试窗口为 64 和 4096 大小;
 - 支持对噪声源的连续测试 ([AIS-31]):
 - Monobit test;
 - Poker test;
 - Runs test;
 - Long runs test;
 - Noise source failure。
 - 安全随机数 buffer 支持读后擦除和回归零点功能 ([FIPS 140-2][FIPS 140-3 draft]);
 - 安全读模式, 只允许在读请求的有限时间内获取到数据, 这个有限时间可配置, 防止产生的数据被偷窥;
 - 支持致命错误自动关闭 (振荡频率与采样时钟同步);
 - 支持在加工功能中 DRBG 和自测试电路上各种在线和离线的完整测试, 以及 KAT (known-answer tests)。
- 输出的 Ready 和多个错误中断有独立的屏蔽位和 ack 位。
- 熵收集周期可以被控制, 使熵汇集时间适应基本的熵产生速率 (熵产生速率是随工艺变化的)。
- 支持密码学上的下述场景:
 - 密码学关键 key 的产生;
 - 初始化向量的产生;
 - Cookies、Nonces 和 Seeds 的产生。



- 支持非密码学的下述场景：
 - 统计抽样；
 - 通信协议中的 re-try 时间；
 - 噪声产生。
- 寄存器不区分安全属性，TRNG 所属子系统 CPU 可以对其所有寄存器进行访问。
- 支持多核无锁并发获取随机数，每次获取 32bit 随机数，获取者需要检查获取到的 32bit 随机数不为全 0，如果为全 0，需要丢弃重新获取 32bit。
- 支持硬件初始化，硬件自动 re-seed 功能，不需要软件协助做 re-seed，可配置。
- 性能要求：
 - 真随机数产生的速率为 8bit/clock；
 - TRNG 和 SEC 的硬件通道，随机数性能达到 1.6Gbps 的吞吐率（400MHz）。
- 支持 32bit 的随机数寄存器接口，支持软件按照 32bit、64bit、128bit 方式读取随机数。
- 支持真随机数 Ready 和错误指示输出中断，支持软件控制中断 mask 以及中断清零。
- MBIST 测试后，需要对 TRNG 复位后再使用。

37.4 遵循协议

- NRBG：符合[SP 800 -90B draft]（和[FIPS140-2]/[FIPS140-3 draft]）标准，加工函数采用 SHA-256。
- DRBG：符合[SP 800-90A](和[FIPS 140-2]/[FIPS 140-3] draft)标准，采用 AES-256 的 CTR_DRBG 模式生成随机数。
- NRBG 噪声源的重复性计数测试和自适应均衡测试兼容 SP800-90B draft 标准，自适应均衡测试同时进行窗口长度 64bit 采样和 4096bit 采样。
- NRBG 噪声源的连续测试包括 Monobit test, Poker test, Runs test 以及 Long Runs tests 四项，符合[AIS-31]标准，同时兼容[FIPS 140-2]标准。
- 安全随机数 buffer 支持读后擦除和回归零点功能（[FIPS 140-2]/[FIPS 140-3 draft]）。



37.5 功能描述

37.5.1 生成随机数

TRNG 可以生成 NRBG、DRBG 以及混合随机数，三者随机性强度比较如下：

NRBG>DRBG>混合随机数。

软件可以根据具体应用使用不同类型的随机数。

37.5.1.1 NRBG

NRBG 是通俗理解的真随机数，生成速度是三者中最慢的；

生成原理：算法核通过收集 8 个物理震荡环产生的熵源，通过 SHA256 算法计算获得的。

37.5.1.2 DRBG

DRBG 是通俗理解的满足密码学安全要求的伪随机数，生成速度相对 NRBG 要快。

TRNG 有独立寄存器控制 DRBG 的使能。

37.5.1.3 混合随机数

TRNG 内部除了 DRBG 生成器外，还集成了 RNG。其中，RNG 是一个高速伪随机发生器。混合随机数是 DRBG 结果以及 RNG 结果的混合，用于弥补 DRBG 性能不足以及在 re-seed 过程中随机数的缺失，满足软件对高性能随机数的需求。

TRNG 有独立寄存器控制混合随机数的使能。

37.5.2 随机数测试

TRNG 内部硬件电路对随机数的随机质量进行定期检查测试，测试范围涵盖：振荡器、噪声质量（包含重复计算、自适应均衡、连续测试(符合 AIS-31 的 monobit、poker、runs、long runs、noise source failure)）、NRBG/DRBG 的重复度。也可以通过软件配置完成对测试电路的 Known Answer Testing 测试。

通过软件读取噪声源寄存器 TRNG_RAW_L/H 数据，便于对芯片的熵进行评估。后处理之后的随机数据直接通过软件读取 TRNG_OUTPUT 寄存器中的随机数据。

说明

测试对象：振荡器、NRBG、DRBG。



37.5.3 中断

37.5.3.1 中断寄存器

TRNG 通过软件配置可以产生中断指示信号 IRQ，CPU 获取中断指示信号之后通过查询 TRNG_STATUS 状态寄存器来获取中断类型。中断类型包括：TRNG 数据 Ready 中断源、FROs 错误关断中断源 (shutdown_oflo)、DRBG 连续产生 2 个 128bits 相同随机数错误中断源 (stuck_out)、AIS 噪声源检测失效中断 (noise_fail)、AIS 游程检测失败中断源 (run_fail)、AIS 长游程检测失败中断源 (long_run_fail)、AIS 扑克检测失败中断源 (poker_fail)、AIS 单比特频数检测失败中断源 (monobit_fail)、NRBG 连续产生 2 个相同随机数错误中断源 (stuck_nrbg)、重复计数检测错误中断源 (repcnt_fail)、自适应均衡检测错误中断源 (aprop_fail)。

图37-5 TRNG_STATUS 状态寄存器

TRNG_STATUS (Read only), Byte Address Offset: 0x10																																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
need_clock	Reserved								blocks_available								test_stuck_out	aprop_fail	repcnt_fail	Reserved				stuck_nrbg	test_ready	monobit_fail	poker_fail	long_run_fail	run_fail	noise_fail	stuck_out	shutdown_oflo	ready
																	0	0	0					0	0	0	0	0	0	0	0	0	0

TRNG 控制寄存器 (TRNG_CONTROL)：TRNG_CONTROL 寄存器包含输出 Ready 和多个错误中断的独立 mask 屏蔽位。

图37-6 TRNG_CONTROL 控制寄存器

TRNG_CONTROL (Read / Write), Byte Address Offset: 0x14																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
data_blocks												Reserved				request_data	re_seed	aprop_fail_mask	repcnt_fail_mask	drbg_en	Reserved	enable_trng	stuck_nrbg_mask	test_mode	monobit_fail_mask	poker_fail_mask	long_run_fail_mask	run_fail_mask	noise_fail_mask	stuck_out_mask	shutdown_oflo_mask	ready_mask
																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

中断确认寄存器 (TRNG_INTACK)：通过软件置 1 来清除 TRNG_STATUS 状态寄存器相应的中断指示位。



图37-7 TRNG_INTACK 中断确认寄存器

TRNG_INTACK (Write only), Byte Address Offset: 0x10																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																test_stuck_out	aprop_fail_ack	report_fail_ack	Reserved			stuck_trng_ack	test_ready_ack	monobit_fail_ack	poker_fail_ack	long_run_fail_ack	run_fail_ack	noise_fail_ack	stuck_out_ack	shutdown_oflo_ack	ready_ack
																open_read_gate															
																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

中断置位寄存器 (TRNG_INT_SET): 软件可以通过写中断置位寄存器相应位为 1 来产生中断。同样需要通过软件置位 TRNG_INTACK 来清除中断。

37.5.3.2 中断处理

当检测到 TRNG_STATUS 寄存器 stuck_out、noise_fail 以及 shutdown_oflo 标志位为 1 时, 表明 TRNG 内部发生了致命错误 (噪声源失效), 这时需要软件置位 TRNG_CONTROL 寄存器中的标志位 enable_trng (先置 0 后置 1), 重新启动内部噪声源。

37.6 编程接口

37.6.1 固件编程接口

TRNG_CTRL0 寄存器复位初值为 0x0FFF4B67, 复位之后 TRNG 默认采用真随机数产生模式, SEC 专用硬件通道接口默认关闭, JTAG_AUTH、USER 通道接口和 Auto_Reseed 默认打开。根据应用需要, 可以通过固件编程重新配置寄存器 TRNG_CTRL0, 控制 SEC、等硬件接口的开关以及 auto-reseed 的周期数据块数量。

37.6.2 应用编程接口

37.6.2.1 获取随机数--不使用加速器专用硬件通道接口

步骤 1 (可选) Re-seed 操作 [37.6.2.5 SP 800-90A DRBG re-seed 操作流程](#) ([可分开作为一个单独循环步骤, 或者和获取随机数放在一起合成一个整体的大步骤])。

步骤 2 如果是中端方式, 中断上报之后, 查看 TRNG_STATUS.ready 是否为 1, 如果为 1 则进入下一个步骤。如果不是中断方式, CPU 在需要真随机数的时候, 去主动查看



TRNG_STATUS.ready 是否为 1，如果为 1 则进入下一个步骤，否则隔一段时间再来查询（[按照目前性能评估，CPU 在需要真随机数的时候，TRNG 一定可以准备好真随机数]）。

步骤 3 从寄存器 TRNG_OUTPUT_0, TRNG_OUTPUT_1, TRNG_OUTPUT_2, TRNG_OUTPUT_3 中依次读取，获得 128bit 的随机数。

步骤 4 往寄存器 TRNG_INTACK 寄存器中的 ready_ack 写 1，清除中断，使 TRNG 继续产生新的随机数。

步骤 5 把获得的 128bit 随机数写入指定内存的安全存放区域。

----结束

37.6.2.2 获取随机数--使用 USER 私有接口时

步骤 1（可选）Re-seed 操作 [37.6.2.5 SP 800-90A DRBG re-seed 操作流程](#)（[可分开作为一个单独循环步骤，或者和获取随机数放在一起合成一个整体的大步骤]）。

步骤 2 如果配置了硬件自动 re-seed，那么软件想获取随机数时，直接去 TRNG_RAN_DATA0, TRNG_RAN_DATA1, TRNG_RAN_DATA2, TRNG_RAN_DATA3 寄存器中去读取即可。

步骤 3 对获得的 32bit 随机数进行检查，如果是全 0，则丢弃重读，否则就写入指定内存的安全存放区域。

----结束

须知

软件从 TRNG_RAN_DATA0~3 寄存器获取随机数之后，硬件会自动更新寄存器中的随机值。

37.6.2.3 查询方式调节 FRO delay

步骤 1 软件以秒级间隔查看 TRNG_ALARM_MASK 寄存器，如果有被置为 1 的 bit，就进入[步骤 2](#)。

步骤 2 检查 TRNG_ALARM_STOP 寄存器对应 bit，如果也为 1，那么相应的 FRO 已经由于多个“alarm 事件”关断了。如果没有关断发生，进入[步骤 3](#)，否则进入[步骤 4](#)。

步骤 3 清除 TRNG_ALARM_MASK（认为是偶然性事件）。



步骤 4 如果一个或者多个 FROs 已经被关断了，软件可以通过配置 TRNG_FRODETUNE 修改这些 FROs 的 delay 参数。对于这类操作，“shutdown_threshold”应该设置低一些（例如设置为 2），且“shutdown_oflo”中断可以被用来标识异常操作或者 FROs 的故障。

----结束

37.6.2.4 中断方式调节 FRO delay

步骤 1 软件以设置“shutdown_threshold”寄存器，设置在调整措施必须实施之前允许关断的 FROs 数目。

步骤 2 如果有“shutdown_oflo”中断发生，那么就进入下面一系列步骤。

步骤 3 清除 TRNG_ALARMMASK。

步骤 4 清除 TRNG_ALARMSTOP。

步骤 5 把 TRNG_FRODETUNE 相关 bit 配置翻转。

步骤 6 配置 TRNG_FROENABLE，将关断的震荡环打开。

----结束

须知

软件必须跟踪中断的时间间隔，如果中断间隔时间太小（例如在分钟级别），就认为是一种异常操作或者 FROs 故障。

37.6.2.5 SP 800-90A DRBG re-seed 操作流程

步骤 1 检查 TRNG_CONTROL.data_blocks 是否为全 0，如果是全 0 就进入下一个步骤，否则到结束。

步骤 2 写 TRNG_CONTROL.re_seed 为 1，逻辑立刻把 TRNG_STATUS.ready 置为 0。

步骤 3 查询 TRNG_STATUS.test_ready 是否为 1，如果为 1，TRNG_PS_AI_*寄存器被复位到全 0，表示接受了“附加输入”的数据，进入下一步骤。

步骤 4 写入 TRNG_PS_AI_*, TRNG_PS_AI_11 作为最后一个寄存器写入。

步骤 5 写完 TRNG_PS_AI_11 之后，TRNG_CONTROL.re_seed 变为 0，re_seed 操作就开始了。



步骤 6 之后写 TRNG_CONTROL.data_blocks 为全 1, TRNG_CONTROL.request_data 为 1。

----结束

须知

Re-seed 开始之后, 使用 NRBG 产生的熵开始处理, 随机数缓存 buffer 被归零, 输出寄存器被清除, TRNG_BLOCKCNT.block_count 被清 0。

对于 SP 800-90A DRBG 来说, 在启动的时候不需要 re_seed, 因为启动过程自动执行一个等同于 re_seed 的初始化功能, 但是在每次 TRNG_CONTROL.data_blocks 变为 0 的时候, 就需要去做 re-seed 操作, CPU 需要定期检查 TRNG_CONTROL.data_blocks 是否为 0, 从而进行定期的 re-seed 操作, 在性能足够的情况下, CPU 可以在获取随机数之前先检查 TRNG_CONTROL.data_blocks 是否为 0 来启动 re-seed。

这些步骤可以放在获取随机数的前面。

37.6.2.6 测试 AIS-31 Monobit,run,poker 测试电路操作流程

- 步骤 1 设置 TRNG_CONTROL.test_mode 为 1, 设置 TRNG_TEST.test_known_noise 为 1。(逻辑主动把 TRNG_STATUS.test_ready 置为 1 表示数据可以被写入, 并且把 monobit test, run test, poker test 电路都复位到初始状态)
- 步骤 2 如果有需要, 可以设置 TRNG_CONFIG 寄存器中的时间设置, 加速测试。
- 步骤 3 把第 1 个 64bit 的测试数据经过向左循环移动 4 位之后写入 TRNG_RAW_L/H 寄存器。逻辑自动把 TRNG_STATUS.test_ready 拉低表示处理启动了, 测试数据在内部自动左移, 对于 monobit test 和 run test, bit[3]是第 1 个被测试的数据, bit[2]是第 2 个, bit[4]是最后一个。对于 poker test, bit[3:0]是第 1 个被测试的 nibble, bit[63:60]是第 2 个, bit[7:4]是最后一个。
- 步骤 4 查询 TRNG_STATUS.test_ready, 如果为 1, 表示可以接受下一个 64bit 测试数据, 这时之前的测试数据已经被处理完成, 可以检查计数器状态。
- 步骤 5 对于一个完整的 monobit test, run test 以及 poker test, 20032bit 的测试数据需要通过重复步骤 3 和步骤 4 写入, 这 313 个 64bit block 数据必须这样提供: 最后一个 block 的 bit[35:4]数据保持为 0。当 TRNG_RUN_CNT.run_test_count 域达到 20001 时, TRNG_STATUS.monobit_fail 会被更新 (除非 cont_poker 被设置为 1)。在计数器达到 20020 时, TRNG_STATUS 中的 run_fail 和 poker_fail 域被更新, 而且



monobit, poker 和 runs 测试状态会被锁定, 以供把它们读出。下一次测试的第 1 个 64bits 写入 TRNG_RAW_L/H 之后会清除状态寄存器而且启动本次的 64bit 测试---对于连续的测 TRNG_RAW_L/H 之后会清除状态寄存器而且启动本次的 64bit 测试---对于连续的测试, 建议在输入新的数据之前主动清除 monobit, poker 和 runs 测试状态寄存器 (通过给 TRNG_INTACK 的 monobit_fail_ack, run_fail_ack 或者 poker_fail_ack 中写 1) 。

步骤 6 把 TRNG_CONFIG 寄存器中的时间设置存储起来 (如果步骤 2 中有变化), 清除 TRNG_TEST.test_known_noise (逻辑立刻自动清楚 TRNG_STATUS.test_ready, 并且复位 monobit test, run test 和 poker test 电路), 清除 TRNG_CONTROL.test_mode。

----结束

37.6.2.7 测试 repetition count, Adaptive Propotion 测试电路操作流程

步骤 1 设置 TRNG_CONTROL 中 test_mode 和 enable_trng 为 1, 设置 TRNG_TEST 中 test_spb, test_known_noise, cont_poker 为 1。(设置 cont_poker 为 1 主要是为了防止在 AIS-31 monobit 和 poker 测试电路发现的错误阻塞本测试)。设置好之后, 逻辑立刻自动把 TRNG_STATUS.test_ready 置为 1 表示数据可以被写入。

步骤 2 如果有需要, 可以设置 TRNG_CONFIG 寄存器中的时间设置, 加速测试。

步骤 3 通过配置 TRNG_SPB_TESTS 寄存器控制不同的计数器和中断值, 注意: 如果在 TRNG_INTACK 中 repcnt_fail_ack 或 aprop_fail_ack 写 1 会把这里相应的计数器复位, 往 TRNG_SPB_TESTS 中的 bit[30:29]写 2'b11 会把 Adaptive Proportion 测试中的内部噪声采样计数器清除。

步骤 4 把第 1 个 64bit 的测试数据经过向左循环移动 2 位之后写入 TRNG_RAW_L/H 寄存器。逻辑自动把 TRNG_STATUS.test_ready 拉低表示处理启动了, 测试数据在内部自动左移, 对{bit[1:0],bit[63:58]}是第 1 个被测试的字节, 后面是 bit[57:50], 最后一个 bit[9:2]。

步骤 5 查询 TRNG_STATUS.test_ready, 如果为 1, 表示可以接受下一个 64bit 测试数据, 这时之前的测试数据已经被处理完成, 可以检查计数器状态。如果需要测试多个数据, 返回步骤 3。

步骤 6 把 TRNG_CONFIG 寄存器中的时间设置存储起来 (如果步骤 2 中有变化), 清除 TRNG_TEST 中 test_spb, test_known_noise 和 cont_poker, 清除 TRNG_CONTROL.test_mode。

----结束



37.6.2.8 测试 SHA256 加工功能操作流程

- 步骤 1 设置 TRNG_CONTROL 中 test_mode 和 enable_trng 为 1，设置 TRNG_TEST 中 test_cond_func，test_known_noise，cont_poker 为 1。（设置 cont_poker 为 1 主要是为了防止在 AIS-31 monobit 和 poker 测试电路发现的错误阻塞本测试）。设置好之后，逻辑立刻自动把 TRNG_STATUS.test_ready 置为 1 表示数据可以被写入。设置 test_cond_func 使能访问 TRNG_COND_*寄存器（当把它们清零之后）。
- 步骤 2 如果有需要，可以设置 TRNG_CONFIG 寄存器中的时间设置，加速测试。或者设置一个确定的 noise_blocks 值。Sample_cycles 的设置实际影响了测试数据加到引擎的方式。
- 步骤 3 对标准的哈希功能测试或者一个特定的 chaining 值，在 TRNG_COND_*中写哈希输入数据或者 chaining 值的最后 256bit。由于内部哈希核是小端方式，写入的字节流必须基于 32bit 字进行翻转：最后字节必须写入 TRNG_COND_7 的 bit[31:24]中。
- 步骤 4 第一部分输入数据必须被作为 noise_blocks*8 个 blocks（1 个 block 为 64bit）输入引擎。这些 blocks 应该把 bit[7:0]和 bit[63:56]进行交换。如果 sample_cycles 为 1，这 64bit 必须循环左移 2bit，否则要循环左移 1bit，移位的值要写入 TRNG_RAW_L (bit[31:0]) /H (bit[63:32]) 寄存器中，写入 TRNG_RAW_H 之后逻辑自动把 TRNG_STATUS.test_ready 拉低。
- 步骤 5 查询 TRNG_STATUS.test_ready，如果为 1，表示可以接受下一个 64bit 测试数据，如果需要输入多个 block 的数据，返回步骤 4。
- 步骤 6 查询 TRNG_STATUS.ready 为 1，表示加工功能哈希操作的结果已经完成，被写入了 TRNG_COND_*寄存器中（注意，此时并不会产生一个 irq 中断，即使 TRNG_CONTROL 寄存器中的 ready_mask 为 1）。这个结果是一个小端的，第一个字节的 bit[7:0]在 TRNG_COND_0 寄存器中，通过给 TRNG_INTACK.ready_ack 中写 1 获取结果。如果需要做更多的测试，返回步骤 3 或者步骤 4。
- 步骤 7 把 TRNG_CONFIG 寄存器中的时间设置存储起来（如果步骤 2 中有变化），清除 TRNG_TEST 中 test_cond_func，test_known_noise 和 cont_poker，清除 TRNG_CONTROL.test_mode。

----结束

37.6.2.9 测试 SP 800-90A DRBG-AES_ECB_256 操作流程

对于 SP 800-90A DRBG，有两个独立的 known-answer 测试，test_aes_256 仅仅测试 AES-256 Core，不关心后处理。Test_sp_800_90 测试所有的后处理逻辑，而且允



许使用 NIST 测试向量测试。Test_sp_800_90 测试会破坏后处理的状态，所以在产生新的随机数之前，需要有相当数量的“示例”操作。

- 步骤 1 设置 TRNG_CONTROL 中 drbg_en 为 1，设置 TRNG_TEST 中 test_aes_256 为 1（逻辑立刻自动清除 TRNG_STATUS.ready 且置 TRNG_STATUS.test_ready 为 1，指示 key 和数据向量可以被写入—置位 TRNG_STATUS.test_ready 需要一段时间，因为内部操作需要完成。注意在这个测试中 TRNG 不需要被使能）。
- 步骤 2 当 TRNG_STATUS.test_ready 为 1 时，把 256bit 的 key 写入 TRNG_KEY_0~TRNG_KEY_7 寄存器中，注意这并没有实际写入 SP 800-90A DRBG 的 key 寄存器中，所以并没有破坏状态。
- 步骤 3 在 TRNG_INPUT_0~TRNG_INPUT_3 中写入 128bit 测试数据用于加密，写完 TRNG_INPUT_3 之后，TRNG_STATUS.test_ready 将变为 0 表示处理开始了。
- 步骤 4 等待 TRNG_STATUS.test_ready 为 1 表示处理结束，AES-ECB 加密的结果可以在相应的 TRNG_OUTPUT_*寄存器中找到。
- 步骤 5 如果需要做多次操作，重复步骤 2，步骤 3，步骤 4。
- 步骤 6 清除 TRNG_TEST.test_aes_256，逻辑立刻自动清除 TRNG_STATUS.test_ready。如果 TRNG 是使能的，且在此时有 DRBG 数据请求，那么在 TRNG_STATUS.ready 为 1 之后，新的随机数据将被产生且载入 TRNG_OUTPUT_*寄存器中。由于 DRBG 状态没有丢失，所以不需要 re-seeding 操作。

----结束

图 37-8 是一个针对 AES-ECB-256 的标准的 NIST 测试向量。



图37-8 NIST 测试向量

```

"key = 603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4"
    → TRNG_KEY_0 = 0x603deb10
    → TRNG_KEY_1 = 0x15ca71be
    → TRNG_KEY_2 = 0x2b73aef0
    → TRNG_KEY_3 = 0x857d7781
    → TRNG_KEY_4 = 0x1f352c07
    → TRNG_KEY_5 = 0x3b6108d7
    → TRNG_KEY_6 = 0x2d9810a3
    → TRNG_KEY_7 = 0x0914dff4

"pt = 6bclbee22e409f96e93d7e117393172a"
    → TRNG_INPUT_0 = 0x6bclbee2
    → TRNG_INPUT_1 = 0x2e409f96
    → TRNG_INPUT_2 = 0xe93d7e11
    → TRNG_INPUT_3 = 0x7393172a (starts processing)

"ct = f3eed1bdb5d2a03c064b5a7e3db181f8"
    → TRNG_OUTPUT_0 = 0xf3eed1bd (result)
    → TRNG_OUTPUT_1 = 0xb5d2a03c (result)
    → TRNG_OUTPUT_2 = 0x064b5a7e (result)
    → TRNG_OUTPUT_3 = 0x3db181f8 (result)

```

37.6.2.10 测试标准的 NIST SP 800-90A DRBG 操作流程

对于 SP 800-90A DRBG，有两个独立的 known-answer 测试，test_aes_256 仅仅测试 AES-256 Core，不关心后处理。Test_sp_800_90 测试所有的后处理逻辑，而且允许使用 NIST 测试向量测试。Test_sp_800_90 测试会破坏后处理的状态，所以在产生新的随机数之前，需要有相当数量的“示例”操作。

- 步骤 1 设置 TRNG_CONTROL 中 drbg_en 为 1，设置 TRNG_TEST 中 test_sp_800_90 为 1（逻辑立刻自动清除 TRNG_STATUS.ready 且置 TRNG_STATUS.test_ready 为 1，指示第一次测试的输入数据可以被写入—置位 TRNG_STATUS.test_ready 的时间非常短，因为内部状态不需要保留。注意在这个测试中 TRNG 不需要被使能）。
- 步骤 2 当 TRNG_STATUS.test_ready 为 1 时，把 384bit 的向量（包含一个等于特征数据串异或熵输入的值）写入 TRNG_PS_AI_0~TRNG_PS_AI_11 寄存器中。写完 TRNG_PS_AI_11 之后，逻辑把 TRNG_STATUS.test_ready 置为 0，表示“示例”操作已经启动。
- 步骤 3 等到 TRNG_STATUS.test_ready 变为 1，表示“示例”操作完成。
- 步骤 4 把包含熵输入值的 384bit 向量（这个熵输入值在 NIST 中叫做 EntropyInputPR）写入 TRNG_PS_AI_0.....TRNG_PS_AI_11 寄存器中进行“产生”功能。逻辑把 TRNG_STATUS.test_ready 置为 0，表示“产生”操作已经启动。



- 步骤 5 等到 TRNG_STATUS.test_ready 变为 1，表示“产生”操作完成，单个 128bit 的结果输出在 TRNG_OUTPUT_* 中。
- 步骤 6 一个标准的 NIST SP 800-90A DRBG known-answer 测试丢弃第一次“产生”功能的结果，只检查第二次“产生”功能的结果，所以[步骤 4](#)和[步骤 5](#)通常需要再重复一次。如果需要更多的测试，重写 TRNG_TEST.test_sp_800_90 为 1，然后把测试逻辑带入[步骤 2](#)。
- 步骤 7 清除 TRNG_TEST.test_sp_800_90，逻辑立刻自动清除 TRNG_STATUS.test_ready，逻辑且擦除所有 SP 800-90A DRBG 内部的寄存器，并且使随机数数据 buffer 回归原点。
- 步骤 8 当所有的 DRBG 状态丢失之后，在新的随机数产生之前，“示例”操作必须被执行。“示例”操作需要把一个“特征数据串”写入 TRNG_PS_AI_* 中。如果此时使能 TRNG，“示例”操作将使 DRBG 运行到准备处理一个新的“产生”操作的点。

----结束

CTR_DRBG 测试向量如[图 37-9](#)所示。



图37-9 CTR_DRBG 测试向量

```

"EntropyInput =
0734e8ed5fbf5edde328c9bb758b1bc8ce0e967d4051ab99
b2171370d8352d71b1b999f6445d14529d2d95170576a7a6"
"PersonalizationString =
631d756e9cf22e45b8f93ca6a847e609130ac3ca5633d27c
bb63d86bf719fc71e81c9ffc8e24805f49cf0f5753c110df"
→ TRNG_PS_AI_0 = 0x0734e8ed XOR 0x631d756e (= 0x64299d83)
→ TRNG_PS_AI_1 = 0x5fbf5edd XOR 0x9cf22e45 (= 0xc34d7098)
→ TRNG_PS_AI_2 = 0xe328c9bb XOR 0xb8f93ca6 (= 0x5bd1f51d)
→ TRNG_PS_AI_3 = 0x758b1bc8 XOR 0xa847e609 (= 0xddccfdcl)
→ TRNG_PS_AI_4 = 0xce0e967d XOR 0x130ac3ca (= 0xdd0455b7)
→ TRNG_PS_AI_5 = 0x4051ab99 XOR 0x5633d27c (= 0x166279e5)
→ TRNG_PS_AI_6 = 0xb2171370 XOR 0xbb63d86b (= 0x0974cb1b)
→ TRNG_PS_AI_7 = 0xd8352d71 XOR 0xf719fc71 (= 0x2f2cd100)
→ TRNG_PS_AI_8 = 0xb1b999f6 XOR 0xe81c9ffc (= 0x59a5060a)
→ TRNG_PS_AI_9 = 0x445d1452 XOR 0x8e24805f (= 0xca79940d)
→ TRNG_PS_AI_10 = 0x9d2d9517 XOR 0x49cf0f57 (= 0xd4e29a40)
→ TRNG_PS_AI_11 = 0x0576a7a6 XOR 0x53c110df (= 0x56b7b779,
starts instantiate)

"EntropyInputPR =
aa6bbcabef45e339136cale7bcelc8819fa37b0963b53667
b36e0053a202ed814650d90d8eed6127666f24020dfd3af9"
→ TRNG_PS_AI_0 = 0xaa6bbcab
→ TRNG_PS_AI_1 = 0xef45e339
→ TRNG_PS_AI_2 = 0x136cale7
→ TRNG_PS_AI_3 = 0xbcelc881
→ TRNG_PS_AI_4 = 0x9fa37b09
→ TRNG_PS_AI_5 = 0x63b53667
→ TRNG_PS_AI_6 = 0xb36e0053
→ TRNG_PS_AI_7 = 0xa202ed81
→ TRNG_PS_AI_8 = 0x4650d90d
→ TRNG_PS_AI_9 = 0x8eed6127
→ TRNG_PS_AI_10 = 0x666f2402
→ TRNG_PS_AI_11 = 0x0dfd3af9
(starts generating 1st result:
1a55267d53af794b2db7cd802140fa87
which is discarded in a standard NIST test)

"EntropyInputPR =
35c1b7a10154c52bd5777390226a4fdb5f16080d06b68369
d0c93d003336e27f1abf2c37e6ab006ca4adc6e18e1907a2"
→ TRNG_PS_AI_0 = 0x35c1b7a1
→ TRNG_PS_AI_1 = 0x0154c52b
→ TRNG_PS_AI_2 = 0xd5777390
→ TRNG_PS_AI_3 = 0x226a4fdb
→ TRNG_PS_AI_4 = 0x5f16080d
→ TRNG_PS_AI_5 = 0x06b68369
→ TRNG_PS_AI_6 = 0xd0c93d00
→ TRNG_PS_AI_7 = 0x3336e27f
→ TRNG_PS_AI_8 = 0x1abf2c37
→ TRNG_PS_AI_9 = 0xe6ab006c
→ TRNG_PS_AI_10 = 0xa4adc6e1
→ TRNG_PS_AI_11 = 0x8e1907a2
(starts generating 2nd result, checked)
"ReturnedBits = b663b9f124943e1380f7dce5abala16f"
→ TRNG_OUTPUT_0 = 0xb663b9f1 (result)
→ TRNG_OUTPUT_1 = 0x24943e13 (result)
→ TRNG_OUTPUT_2 = 0x80f7dce5 (result)
→ TRNG_OUTPUT_3 = 0xabala16f (result)

```



37.6.2.11 获取噪声源数据操作流程

在 NIST 和 FIPS 的相关随机数认证中需要对噪声源产生的随机数据进行评估。噪声源数据直接来自于 FROs，没有经过后处理。具体的操作步骤如下：

- 步骤 1 设置寄存器 TRNG_CONTROL，写全 0，确保 TRNG 处在一个确定状态。
- 步骤 2 设置寄存器 TRNG_ALARMSTOP 和 TRNG_ALARMMASK，写全 0，清除警告标志位。
- 步骤 3 设置寄存器 TRNG_CONFIG.sample_div，设置从 FROs 采样的时钟间隔，0 表示每个时钟都从 FROs 采样，15 表示每 16 个时钟周期采样一次。设置 TRNG_CONFIG.sample_cycles，设置异或到一起生成 1bit 的 FRO 采样 bit 数目，异或结果再移位到主移位寄存器中。如果测试到的噪声源数据，随机效果不理想，则**增大 sample_div 和 sample_cycles 的值，有益于提高熵值。**
- 步骤 4 配置寄存器 TRNG_CONTROL，写入 0x00000500。其中设置 TRNG_CONTROL.test_mode=1 使能测试模式。设置 TRNG_CONTROL.enable=1 使能 trng，FRO 振荡输出。
- 步骤 5 配置寄存器 TRNG_TEST，写入 0x00002000。其中标志位 TRNG_TEST.test_noise=1 使能可以读取寄存器 TRNG_RAW_L/H 的值。TRNG_TEST.test_shiftreg=0 表示从 main shift register 的 buffer 中读取数据。Main shift register 每取到 64bit 熵源，会存到一个 64bit buffer 中。
- 步骤 6 CPU 查看寄存器 TRNG_STATUS。若 test_ready 标志位为 1，表示 64bit 噪声源数据已经准备好，进入下一个步骤。
- 步骤 7 读取寄存器 TRNG_RAW_L 和 TRNG_RAW_H 的 64bit 采样数据。
- 步骤 8 配置寄存器 TRNG_INTACK，写入 0x00000100，清楚 test_ready 信号。TRNG 继续产生新的随机数据。
- 步骤 9 重复**步骤 6 到步骤 8**，直到采样到足够需要的数据。

----结束



37.7 复位和初始化

37.7.1 复位

TRNG 上电时默认处于解复位状态，上电时时钟默认使能。支持软件复位。

软件在 TRNG 空闲时（所有访问完成），可以通过配置寄存器，按照操作流程对 TRNG 模块进行复位。

37.7.2 硬件初始化

TRNG_CTRL0 寄存器复位初值为 0x0FFF4B67，复位之后 TRNG 默认采用真随机数产生模式，SEC 专用硬件通道接口默认关闭，USER 通道接口和 Auto_Reseed 默认打开。

TRNG 支持复位之后硬件自动进行初始化设置。硬件初始化的参数是固定的，自动初始化的步骤如下：

- 步骤 1 配置寄存器 TRNG_CONFIG 为 0x00010008。这个设置表示每次需要 8 个 512bit 的噪声源数据 blocks 用于 SHA256 加工函数输出 256bit 输出。每一个 clk 采样 1bit 噪声源数据，产生熵的速度快，但品质低。非安全读模式。
- 步骤 2 配置寄存器 TRNG_CONTROL.drbg_en=1, TRNG_CONTROL.enable_trng=1, 使能 AES-256 DRBG 和使能 FRO 振荡。
- 步骤 3 配置 TRNG_CONTROL.data_blocks 为全 1，准备一轮生成 4096 个 block（128bit）随机数，配置 TRNG_CONTROL.request_data 为 1 表示使能生成 data_blocks 个随机数块。

----结束

须知

硬件自动初始化由硬件自动完成，不需要软件操作。硬件初始化完成之后，可以根据具体应用需要，软件再进行配置。



37.7.3 软件初始化

37.7.3.1 软件初始化—不使用加速器专用硬件通道接口

在不使用加速器专用硬件通道时，CPU 可以通过直接访问 TRNG IP 内部相关寄存器获取到随机数。

- 步骤 1 确保加速器专用硬件通道接口是关闭的，SEC 接口默认关闭。确保如下寄存器为 1，TRNG_CTRL0.user_interface_disable，TRNG_CTRL0.sec_interface_disable 默认值都是为 1 的。
- 步骤 2 配置寄存器 TRNG_CONTROL，写全 0，确保 TRNG 处在一个确定状态。
- 步骤 3 配置 TRNG_INTACK 为 0xff，清除起始的无效中断。
- 步骤 4 配置寄存器 TRNG_CONFIG.noise_blocks，设置为加工功能提供输入的 512bit blocks 的数目，根据[SP 800-90B draft]中描述输入熵信息量必须是输出的 2 倍。假设噪声源每 8bit 包含 1bit 熵，那么对于 SHA256 加工功能需要输出 256bit 的输出，就需要 $256 \times 8 \times 2 = 4096$ bit 的输入采样 bit，就等于 $4096 / 512 = 8$ blocks。对于初始化或者 re-seeding 的时候需要配置为 2 倍（即 $8 \times 2 = 16$ ），因为 SHA256 初始化和 re-seeding 的时候需要有两个加工功能输出。配置寄存器 TRNG_CONFIG.sample_div，设置从 FROs 采样的时钟间隔，0 表示每个时钟周期都从 FROs 采样，15 表示每 16 个时钟周期采样一次。配置寄存器 TRNG_CONFIG.read_timeout，设置安全读模式的读开放时间，0 表示关闭安全读模式，1~15 使能安全读模式，安全读的超时时间为 $\text{read_timeout} \times 16$ 个时钟周期。配置寄存器 TRNG_CONFIG.sample_cycles，设置异或到一起生成 1bit 的 FRO 采样 bit 数目，异或结果再移位到主移位寄存器中，这个配置必须保证每个移位的 8bit 中至少有 1bit 熵。（例如配置 TRNG_CONFIG 为 0x00010008 表示快速的但是低品质的熵产生速度）
- 步骤 5 配置寄存器 TRNG_ALARMCNT 寄存器，配置 alarm_threshold 来决定检测每个 FRO 重复 pattern 的采样窗口，默认值为全 1，最大容忍度。配置 shut_down_threshold，当 shut_sown_count 大于这个门限时，就产生 shutdown_oflo 中断。配置 shutdown_fatal 决定 shutdown_oflo 中断是否致命中断。
- 步骤 6 配置寄存器 TRNG_FROENABLE 寄存器，使能 FRO，只有使能的 FRO 才会产生熵以及中断信息。
- 步骤 7 配置寄存器 TRNG_CONTROL，配置*_mask 来控制这些事件是否产生中断 irq。
- 步骤 8 配置 TRNG_CONTROL.drbg_en 来决定是否使能 AES-256 (DRBG[SP 800-90A])，配置寄存器 TRNG.CONTROL.enable_trng 来使能 TRNG。



步骤 9 配置 TRNG_CONTROL.data_blocks 为全 1，准备一轮生成 4096 个 block 随机数，配置 TRNG_CONTROL.request_data 为 1 表示使能 data_blocks。

----结束

须知

TRNG 初始化中有清除一些寄存器的操作，所以正常工作过程中不能重新初始化。否则会导致 TRNG re-seed 状态机出问题。

37.7.3.2 软件初始化—使用加速器专用硬件通道接口时

在使用加速器专用硬件通道接口时，CPU 不能通过直接访问 TRNG IP 内部相关寄存器获取到随机数，需要通过访问 TRNG 控制寄存器接口去获取随机数。下面以安全 USER 为例来说明，非安全 USER 的方式类似。（当使用 USER 接口，复位之后 USER 会自动对寄存器 TRNG_CONTROL 进行配置，DRBG 随后开始产生随机数。软件配置步骤 1、2 的时候，需要保证在 DRBG 产生随机数之前配置完成，否则会出现错误。建议软件初始化直接跳过步骤 1、2 的配置，不会影响正常功能使用。）

步骤 1 配置寄存器 TRNG_CONTROL，写全 0，确保 TRNG 处在一个确定状态。

步骤 2 配置 TRNG_INTACK 为 0xff，清除起始的无效中断。

步骤 3 配置寄存器 TRNG_CONFIG.noise_blocks，设置为加工功能提供输入的 512bit blocks 的数目，根据[SP 800-90B draft]中描述输入熵信息量必须是输出的 2 倍。假设噪声源每 8bit 包含 1bit 熵，那么对于 SHA256 加工功能需要输出 256bit 的输出，就需要 $256 \times 2 = 512$ 的输入采样 bit，就等于 512/512=1 blocks。对于初始化或者 re-seeding 的时候需要配置为 2 倍（即 $8 \times 2 = 16$ ），因为 SHA256 初始化和 re-seeding 的时候需要有两个加工功能输出。配置寄存器 TRNG_CONFIG.sample_div，设置从 FROs 采样的时钟间隔，0 表示每个时钟周期都从 FROs 采样，15 表示每 16 个时钟周期采样一次。配置寄存器 TRNG_CONFIG.read_timeout，设置安全读模式的读开放时间，0 表示关闭安全读模式，1~15 使能安全读模式，安全读的超时时间为 $\text{read_timeout} \times 16$ 个时钟周期。配置寄存器 TRNG_CONFIG.sample_cycles，设置异或到一起生成 1bit 的 FRO 采样 bit 数目，异或结果再移位到主移位寄存器中，这个配置必须保证每个移位的 8bit 中至少有 1bit 熵。（例如配置 TRNG_CONFIG 为 0x00010008 表示快速的但是低品质的熵产生速度）

步骤 4 配置寄存器 TRNG_ALARMCNT 寄存器，配置 alarm_threshold 来决定检测每个 FRO 重复 pattern 的采样窗口，默认值为全 1，最大容忍度。配置 shut_down_threshold，



当 shut_sown_count 大于这个门限时，就产生 shutdown_oflo 中断。配置 shutdown_fatal 决定 shutdown_oflo 中断是否致命中断。

步骤 5 配置寄存器 TRNG_FROENABLE 寄存器，使能 FRO，只有使能的 FRO 才会产生熵以及中断信息。

步骤 6 配置寄存器 TRNG_CONTROL，配置*_mask 来控制这些事件是否产生中断 irq。

步骤 7 配置 TRNG_CONTROL.drbg_en 来决定是否使能 AES-256 (DRBG[SP 800-90A])，配置寄存器 TRNG.CONTROL.enable_trng 来使能 TRNG。

步骤 8 配置 TRNG_CONTROL.data_blocks 为全 1，准备一轮生成 4096 个 block 随机数，配置 TRNG_CONTROL.request_data 为 1 表示使能 data_blocks。

步骤 9 最后确保加速器专用硬件通道接口是打开的，确保如下相关寄存器为 0，TRNG_CTRL.user_interface_disable，TRNG_CTRL.sec_interface_disable。

----结束

须知

TRNG 初始化中有清除一些寄存器的操作，所以正常工作过程中不能重新初始化。否则会导致 TRNG re-seed 状态机出问题。

37.7.3.3 RNG 初始化操作流程

为了防止意外，由于工艺问题导致 TRNG 产生器不能正常工作，或者安全性要求不高的场景下，可以使用内部集成的 RNG 工作，除了 RNG 需要初始化之外，其他操作都和 TRNG 操作相同，下面列出如何启动 RNG 工作，以及 RNG 的初始化步骤。

步骤 1 设置 TRNG_CTRL 中 trng_mode 为 0，设置工作模式为 RNG 模式，默认为 TRNG 模式。

步骤 2 配置 RNG_CTRL.rng_en 为 1，打开 RNG，根据需要配置 RNG_CTRL 其他域。

步骤 3 配置 RNG_SEED，设置 RNG 的随机种子。

步骤 4 上述步骤配置好之后，再通过类似 TRNG 的初始化步骤启动。

----结束



37.8 注意事项

37.8.1 Debug

TRNG 由物理噪声源，加工功能，DRBG 和测试电路组成，测试电路可用于 DFX，具体使用方法请看工作方式中测试方面的描述。

37.9 故障处理指南

软件检测到 IRQ 输出之后，通过查询 TRNG_STATUS 寄存器状态，判断如果出现致命错误，则软件重新置位 TRNG_CONTROL 寄存器的‘enable_trng’标志位，硬件将自动把 output buffer 中的数据清零，并且强制 TRNG_STATUS 中的‘ready’标志位为 0。

37.10 性能

37.10.1 FRO 的熵 bit 速率

最大熵需要的采样频率为 $t_{\text{startup}} = \frac{4N^2d^2}{D} = \frac{1}{Df^2}$

其中 d 为 FRO 反相器的延时；N 为反相器的数目；f 为 FRO 的中心频率 $f = \frac{1}{2Nd}$

熵 bit 速率的基带带宽为 $\frac{\text{bits}}{\text{second}} = \frac{1}{t_{\text{startup}}} = Df^2$

37.10.2 TRNG 随机数 bit 速率

真随机数产生的速率为 8bit/clock，在 400MHz 时钟频率下的吞吐率为 3200Mbit/s，200MHz 时钟频率下的吞吐率为 1600Mbit/s。



38 TSENSOR

38.1 概述

Tsensor 是一个温度检测模块，可以检测芯片的温度。

38.2 特性

- 温度检测精度为 $\pm 3^{\circ}\text{C}$ ，范围是 $-40^{\circ}\text{C} \sim +125^{\circ}\text{C}$ ，采用 10 bit SARADC 量化温度，温度分辨率为 $0.207^{\circ}\text{C}/\text{LSB}$ 。
- 温度检测电路可设置为工作、不工作两种模式。

38.3 关键术语

表38-1 TSENSOR 关键术语描述

术语	描述
TSENSOR	温度检测模块。

38.4 功能

检测芯片 die 内的温度，并且把 die 内温度转换成 10bit 的二进制码输出。



38.5 工作方式

具体配置步骤如下：

步骤 1 通过寄存器 TSENSOR_OUT，可以得到 Tsensor 直接输出码（10bit 二进制）。

步骤 2 通过 10bit 二进制的值对温度℃单位换算公式，可以得到温度：

$$T_{\text{°C}} = [\text{BIN2DEC}(\text{TEMP_OUT}[9:0]) - 159] / (939 - 159) * [125 - (-40)] + (-40)$$

步骤 3 通过寄存器 TSENSOR_CTRL bit[2]，可以对 Tsensor IP 进行工作使能配置。配置为 1，表示打开 Tsensor 工作；配置为 0，Tsensor 不工作。

步骤 4 通过寄存器 TSENSOR_CTRL bit[1:0]，可以配置采样时间。芯片每 16 个采样间隔会进行 16 次采样，对于得到的数据相加，然后取平均值作为 IP 检测到的 die 温度。

----结束

38.5.1 TSensor 校准

TSensor 预留 2 个 11-bit EFUSE 做两点温度校准，分别对应 105℃和 25℃校准温度码。

表38-2 EFUSE 域段中的校准信息表

EFUSE 0	位宽	功能	信号明细
[299:289] [277:267]	22	TSensor Trim	[299:289] 常温 25C TSensor
			[277:267] 高温 105C TSensor

[299:289], [277:267]是 TSensor 温度校准值；可以在 SysCtrl 寄存器 DIEID9 和 DIEID8 进行查询。

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：



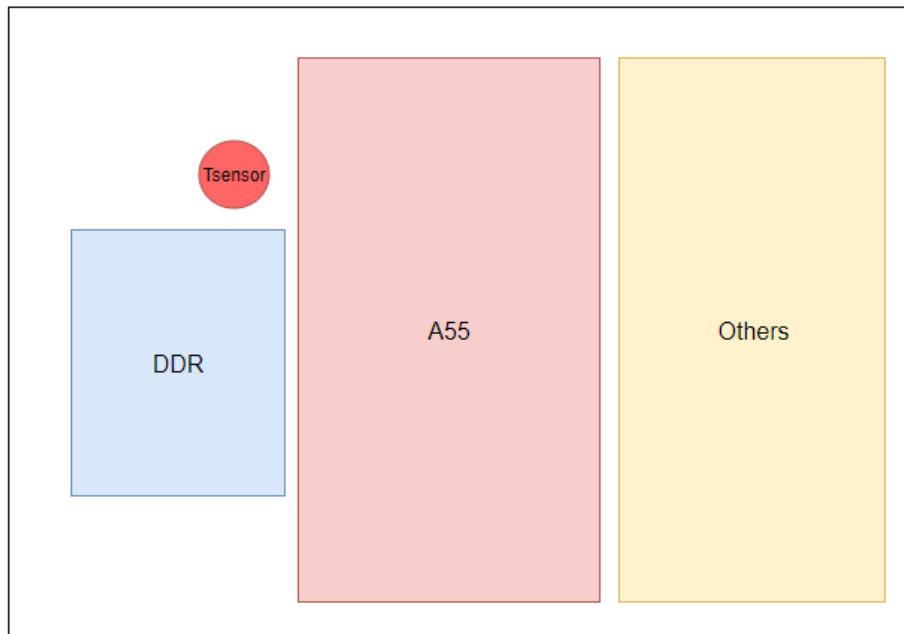
$$y = \left(\frac{80}{T_2 - T_1} \right) * x + 25 - \left(\frac{80}{T_2 - T_1} \right) * T_1$$

其中：

- T1 为 EFUSE 中 25°C 下的温度校准码值；
- T2 为 EFUSE 中 105°C 下的温度校准码值；
- X 为读取的温度码值；
- Y 为 die 真实结温。

Tsensor 在芯片中的位置如图 38-1 所示。

图38-1 TSENSOR 在芯片中的位置图





39 SerDes

39.1 概述

Hi3093 支持 SGMII（用于 GE 业务）、PCIE、USB3 三种高速接口。其中 GE 接口有两组；PCIE 接口有两组，USB3 有一组。其中，GMAC SerDes 与 GMAC 相连，PCIE SerDes 与 PCIE 相连，USB3 Serdes 和 USB3 控制器相连。各种接口支持的速率及用途如表 39-1 所示。

表39-1 Serdes 接口应用表

Serdes 通道	接口	速率	用途
PCIE SerDes	PCIE	5Gbps	GPU、MCTP 与 HOST 的通信通道
GMAC SerDes	GE	1.25Gbps	以太网通信
USB3.0 PHY	USB3.0	5Gbps	USB 3.0 PHY

39.2 特性

39.2.1 Combo phy 通用特性

- 支持输入均衡、输出预加重。
- Serdes 支持内环回，支持典型 PRBS 码流自发包，支持 PRBS 环回接收检测和统计功能（含误码统计）。
- 支持链路故障信息记录和查询功能。



- 提供数字眼图功能。
- IEEE 1149.6 (AC JTAG) 边界扫描。
- 支持环回测试。
- PHY 内集成了收发器, PLL, 数字 Core 和 ESD 保护电路。
- 支持 prbs 测试。
- 支持 ate,eq 等 SI 参数可配置。

39.2.2 Combo phy 独立特性

SGMII Combo Phy

1.25Gbps SGMII 接口传输速率, 支持 10bit 接口 125M 时钟速率。

PCIE Combo Phy

5Gbps 传输速率, PIPE3 PHY 接口, 提供可配置的软 PCS 层。

- 支持 16bit 接口 250MHz 时钟速率。
- 支持 1 lane PIPE。
- 支持 1.8V I/O 供电。

USB3 Combo Phy

5-Gbps SS USB3 数据传输速率。

- 支持展频时钟 (SSC) 范围为(+4,980 ppm through -4,980 ppm);
- 支持 PIPE3 USB 3.0 传输接口, 提供可配置的软 PCS 层;
- 支持 32bit 接口 125MHz 时钟速率;
- 支持 USB 3.0 测试模式;
- 支持 1.8-V I/O 供电。

39.3 功能和工作方式

39.3.1 SGMII Combo Phy 初始化

具体流程如图 39-1 所示。



图39-1 Combo Phy 上电初始化时序图

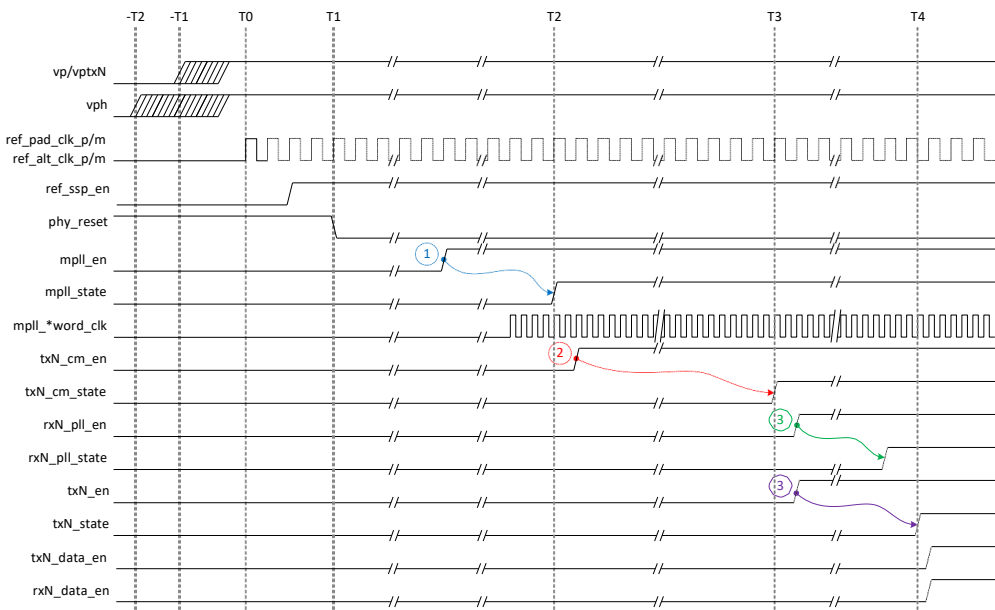


表39-2 上电流程中参数值表

Timing Parameter	Description	Value
RefClk	Differential reference clock input to the PHY. Equal to pins ref_alt_clk_p/ref_alt_clk_m or ref_pad_clk_m/ref_pad_clk_p depending on the setting of ref_use_pad.	–
-T2 to -T1	Maximum time vph is applied before vp/vptxN	< 100 ms
T0	Time when supplies are within specified limits and the reference clock is up and stable	–
T1	Time after T0 when ASIC can remove phy_reset	> 10 μ s
T2	Time after T1 when mpll_state asserts and the MPLL clocks are guaranteed to be up and stable	< 25 μ s
T3	Time after T2 where common mode is established and the PHY is in the common-mode or Quiet state	230 μ s (156.25-MHz reference clock) 204 μ s (125-MHz reference clock)



Timing Parameter	Description	Value
T4	Time after T3 where the PHY is in Active state and ready to transmit and receive data	2 μ s (156.25-MHz reference clock) 2 μ s (125-MHz reference clock)

SGMII Serdes 的上电流程如下，具体寄存器请参考 [39.3.6.1 ComboPhy 上电流程用到寄存器](#) 章节。

步骤 1 板级提供稳定的电源和参考时钟（上电后 phy_reset 默认为 1）。

步骤 2 serdes 的参数配置（仅在 phy_reset 为 1 情况下进行配置）。

步骤 3 serdes 的控制寄存器初始化：

txN_reset 和 rxN_reset 配置为 0；

rxN_invert 配置为 0（后续根据管脚 PN 反转情况配置）；

txN_vboost_en 配置为 0；

txN_detect_rx_req 配置为 0。

步骤 4 时钟稳定后等待大于 10us 时间后，配置 ref_ssp_en 和 phy_reset，使能参考时钟和解除 PHY 的复位。

步骤 5 配置 mpll_en；启动 PLL。

步骤 6 读取 mpll_state 的状态值，当其值为 1 时，表示 mpll 已经锁定。

步骤 7 等待大于 25us。

步骤 8 配置 tx0_cm_en 为 1。

步骤 9 读取 tx0_cm_state，当其值为 1 表示共模模式已经建立。

步骤 10 等待大于 230us。

步骤 11 配置 rx0_pll_en 和 tx0_en 为 1，启动 TX 和 RX 通道。

步骤 12 查询 rx0_pll_state 和 tx0_state 的状态，为 1 表示通道启动成功。

步骤 13 等待大于等于 4us。

步骤 14 配置 tx0_data_en 和 rx0_data_en。启动数据传输。



----结束

39.3.2 ----结束 PCIE Combo Phy 初始化

PCIE 所用的 Combo Phy 的上电时序要求请参考图 39-2。

图中的各个时序请参考表 39-3。使用的具体寄存器请参考 39.3.6.1 ComboPhy 上电流程用到寄存器章节。

图39-2 上电时序

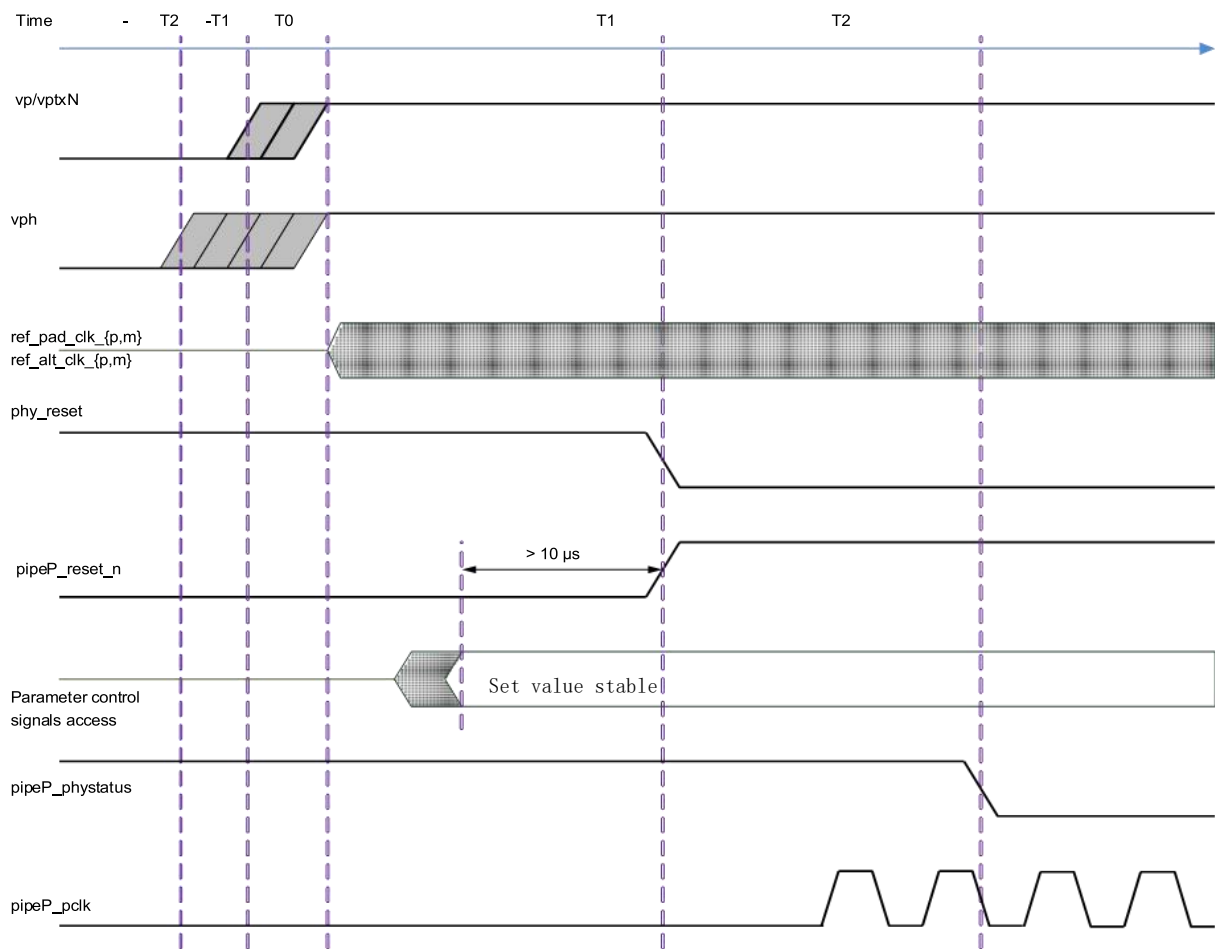




表39-3 上电时序

Timing Parameter	Description	Value
RefClk	Differential reference clock input to the PHY. Equal to pins ref_alt_clk_p/ref_alt_clk_m or ref_pad_clk_m/ref_pad_clk_p depending on the setting of ref_use_pad.	-
-T2 to -T1	Maximum time vph is applied before or after vp/vptxN	< 100 ms
T0	Time when supplies are within specified limits and the reference clock is up and stable	-
T1	Time after T0 when ASIC can remove phy_reset	> 10 μ s
T1 to T2	Time after T1 when pipeP_phystatus de-asserts to indicate that pipeP_pclk is guaranteed to be up and stable	255 μ s (100-MHz reference clock)

39.3.2.1 单板上电流程

- 步骤 1 板级提供稳定的电源和参考时钟。
- 步骤 2 时钟稳定后，解除 PCIE Core 的软复位 (HOST_SUBSYS_REG. HOST_RST_DISABLE. pcie0_core_rst_dis) 。
- 步骤 3 配置 SYS_CTRL 寄存器，复位 phy_reset (配置为 0，上电默认为 0) 。
- 步骤 4 配置关闭 PCIE 控制器的 ltssm_en 使能 (PCIE_MISC. PCIE_CTRL7. pcie_app_ltssm_enable 配置为 0) 。
- 步骤 5 配置 PCIE MISC 寄存器，复位 pipeP_reset_n (配置为 1，上电默认为 0) 。
- 步骤 6 配置 PCIE MISC 寄存器，配置 serdes 的参数 (参数推荐值，参考后续章节) 。
- 步骤 7 延时 10us，释放 phy_reset；释放 pipeP_reset_n。
- 步骤 8 配置打开 PCIE 控制器的 ltssm_en 使能 (配置为 1) 。

----结束

其中 pipeP_reset_n 的控制细化步骤为：



复位 pipeP_reset_n

步骤 1 配置 HOST_SUBSYS_REG.PCIE_PHY_RST_CFG.pcie_phy_rst_sel 为 1。

步骤 2 配置 PCIE_MISC.PCIE_CTRL8.power_up_rst_n_en 为 1。

----结束

释放 pipeP_reset_n

步骤 1 配置 HOST_SUBSYS_REG.PCIE_PHY_RST_CFG.pcie_phy_rst_sel 为 1。

步骤 2 配置 PCIE_MISC.PCIE_CTRL8.power_up_rst_n_en 为 0。

步骤 3 配置 HOST_SUBSYS_REG.PCIE_PHY_RST_CFG.pcie_phy_rst_sel 为 0。

----结束

39.3.2.2 PCIE 复位后启动流程

PCIE 控制器受 host 复位、PCIE 管脚复位控制。当 PCIE 控制器复位会导致 serdes 参数恢复为默认值。因此当 PCIE 控制器解复位后，需要重新初始化 serdes，流程请参考图 39-2。步骤如下：

步骤 1 PCIE 控制器解复位后，上报中断（参考 Hi3093 中断向量表）；

步骤 2 同单板上电流程。

----结束

须知

仅需上电后，电源和参考时钟稳定后至少 10us 后释放复位。

根据手册要求，通过控制器接口修改 Serdes 参数配置，需要在 serdes 复位释放前，完成配置，并且 serdes 工作过程中，参数值不能变动。

39.3.3 USB3 Combo Phy 初始化

USB3 用的 Combo Phy 的上电流程请参考图 39-3，流程中的参数请参考表 39-5。

使用的具体寄存器参考 39.3.6.1 ComboPhy 上电流程用到寄存器章节。



图39-3 上电时序图

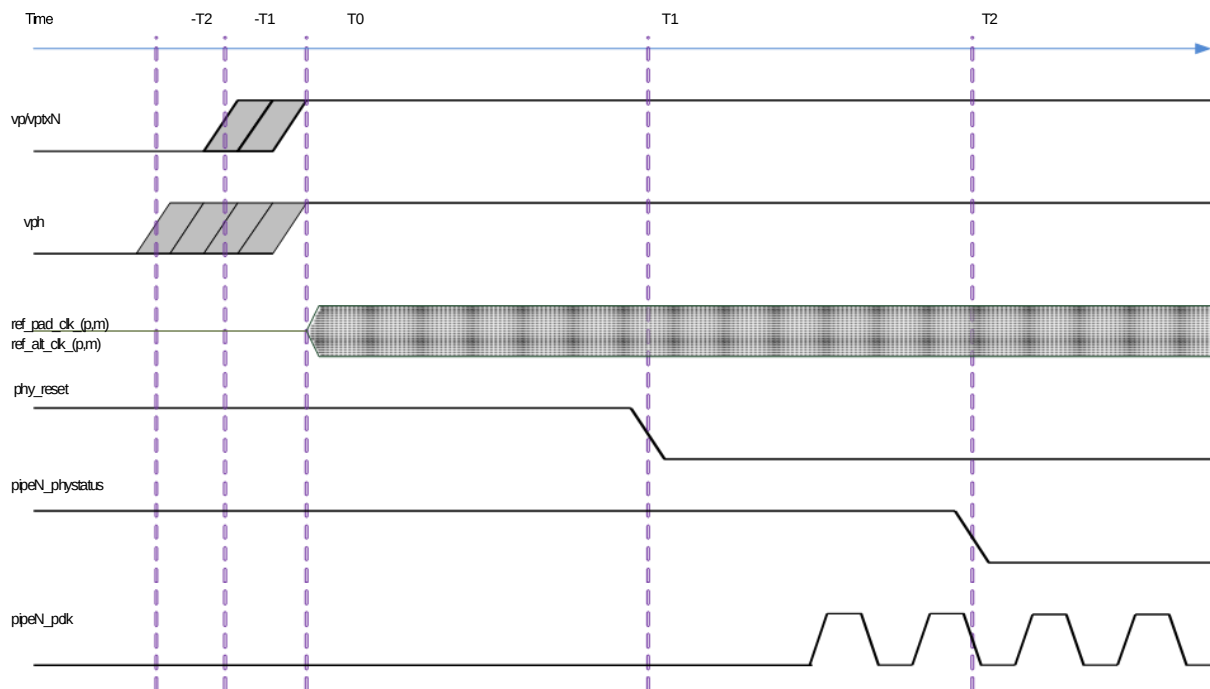


表39-4 POR Timing Values

Timing Parameter	Description	Value
RefClk	Differential reference clock input to the PHY. Equal to pins ref_alt_clk_p/ref_alt_clk_m or ref_pad_clk_m/ref_pad_clk_p depending on the setting of ref_use_pad.	-
-T2 to -T1	Maximum time vph is applied before or after vp/vptxN	< 100 ms
T0	Time when supplies are within specified limits and the reference clock is up and stable	-
T1	Time after T0 when ASIC can remove phy_reset	> 10 μ s
T1 to T2	Time after T1 when pipeN_phystatus de-asserts to indicate that pipeN_pclk is guaranteed to be up and stable	257 μ s (100-MHz reference clock)

上电流程如下：



- 步骤 1 板级提供稳定的电源和参考时钟。
- 步骤 2 时钟稳定后，配置 serdes 的参数。
- 步骤 3 Serdes 参数配置稳定后，至少 10us，可释放 phy_reset（需提前释放 USB MISC 复位）和 pipeP_reset_n（需提前释放 USB 控制器复位）。

----结束

由于 serdes 初始化和 USB 控制器配置有配合关系，整合后流程请参考 25 USB 章节。

须知

仅需上电后，电源和参考时钟稳定后至少 10us 后释放复位。

通过控制器接口修改 Serdes 参数配置，需要重新复位 serdes 才可以生效。

39.3.4 ComboPhy 的 CR 接口

本章节给出 Combo Phy 的寄存器 CR 控制接口，通过 CR 接口可读取和配置 ComboPhy 内部的寄存器。可操作的寄存器内容，请参考 Combo Serdes 的寄存器手册。

下文给出了 3 个 Combo Phy CR 接口控制寄存器。

39.3.4.1 Combo Phy CR 接口控制寄存器

USB3 PHY 的 CR 接口配置寄存器

下面给出了 USB3 PHY 的 CR 接口寄存器，详细请参考 MISC_USB3_PHY 寄存器手册。

表39-5 USB3PHY 的 CR 接口寄存器

寄存器名称	寄存器域段	Bit 位	R/W	含义说明
USB3_PHY_CTRL7 偏移地址：0x1024	cr_data_in	31:16	RW	CR 寄存器数据输出
	reserved	15:9	-	保留
	cr_cap_data	8	RW	CR 寄存器数据捕获



寄存器名称	寄存器域段	Bit 位	RW	含义说明
	cr_cap_addr	7	RW	CR 寄存器地址捕获
	reserved	6	-	保留
	cr_write	5	RW	CR 寄存器写控制
	cr_read	4	RW	CR 寄存器读控制
USB3_PHY_STATUS2 偏移地址：0x104c	reserved	31:17	RO	保留
	cr_ack	16	RO	CR ACK
	cr_data_out	15:0	RO	CR 应对数据

PCIE PHY 的 CR 接口配置寄存器

PCIE PHY 的 CR 接口操作寄存器如表 39-6 所示，详细请参考 PCIE_MISC 寄存器手册。

表39-6 PCIE PHY 的 CR 接口寄存器

寄存器名称	寄存器域段	Bit 位	属性	含义说明
PCIE_PHYCTL3	pciephy_cfg_ssc_ref_clk_sel	31:23	RW	ssc 参考时钟选择。
	pciephy_cfg_ssc_range	22:20	RW	Serdes ssc 的范围。
	phy_cr_read	19	RW	PHY 内部寄存器间接访问读指示信号。 注意：Combophy 不支持访问 Reserved 地址空间。
	phy_cr_write	18	RW	PHY 内部寄存器间接访问写指示信号。 注意：Combophy 不支持访问 Reserved 地址空间。



寄存器名称	寄存器域段	Bit 位	属性	含义说明
	phy_cr_cap_addr	17	RW	PHY 内部寄存器间接访问地址锁存指示信号。 注意：Combophy 不支持访问 Reserved 地址空间。
	phy_cr_cap_data	16	RW	PHY 内部寄存器间接访问数据锁存指示信号。 注意：Combophy 不支持访问 Reserved 地址空间。
	phy_cr_data_in	15:0	RW	PHY 内部寄存器间接访问地址/数据输入信号。 注意：Combophy 不支持访问 Reserved 地址空间。数据位宽不能超过 16bit。
PCIE_PHYSTAT0	pciephy_rtune_ack	31	RO	phy 级联信号
	reserved	30:17	RO	保留
	phy_cr_ack	16	RO	PHY Debug 应答信号
	phy_data_out	15:0	RO	PHY Debug 数据输出

SGMII PHY 的 CR 接口配置寄存器

参考 HP_SUBSYS 寄存器手册。

GMAC2 (SGMII0) 寄存器偏移地址为 0x17C~0x180,

GMAC3 (SGMII1) 寄存器偏移地址为 0x184~0x188。



表39-7 SGMIIIPHY 的 CR 接口寄存器

寄存器名称	寄存器域段	Bit 位	属性	含义说明
GMAC2 SEDES 的 CR 间接访问接口	reserved	31:21	RO	保留
	phy0_cr_cap_addr	20	RW	cr 间接访问 地址 使能, 1 为有效。
	phy0_cr_cap_data	19	RW	cr 间接访问 数据 使能, 1 为有效。
	phy0_cr_cap_read	18	RW	cr 间接访问 读操 作选择, 1 为有 效。
	phy0_cr_cap_write	17	RW	cr 间接访问 写操 作选择, 1 为有 效。
	phy0_cr_cap_en	16	RW	cr 间接访问接口使 能, 1 为有效。
	phy0_cr_cap_data_ req	15:0	RW	cr 间接访问 sds cr 接口 输出数据。
GMAC2 SEDES 的 CR 间接访问接口	reserved	31:17	RO	保留。
	phy0_cr_cap_ack	16	RO	cr 间接访问 sds cr 接口 ack。
	phy0_cr_cap_data_ ack	15:0	RO	cr 间接访问 sds cr 接口 反馈数据。
GMAC3 SEDES 的 CR 间接访问接口	reserved	31:21	RO	保留。
	phy1_cr_cap_addr	20	RW	cr 间接访问 地址 使能, 1 为有效。
	phy1_cr_cap_data	19	RW	cr 间接访问 数据 使能, 1 为有效。



寄存器名称	寄存器域段	Bit 位	属性	含义说明
	phy1_cr_cap_read	18	RW	cr 间接访问 读操作选择, 1 为有效。
	phy1_cr_cap_write	17	RW	cr 间接访问 写操作选择, 1 为有效。
	phy1_cr_cap_en	16	RW	cr 间接访问接口使能, 1 为有效。
	phy1_cr_cap_data_req	15:0	RW	cr 间接访问 sds cr 接口 输出数据。
GMAC3 SEDES 的 CR 间接访问接口	reserved	31:17	RO	保留。
	phy1_cr_cap_ack	16	RO	cr 间接访问 sds cr 接口 ack。
	phy1_cr_cap_data_ack	15:0	RO	cr 间接访问 sds cr 接口 反馈数据。

须知

GMAC PHY 使用 cr 间接访问接口需将 phy0_cr_cap_en 或 phy0_cr_cap_en 配置为 1。

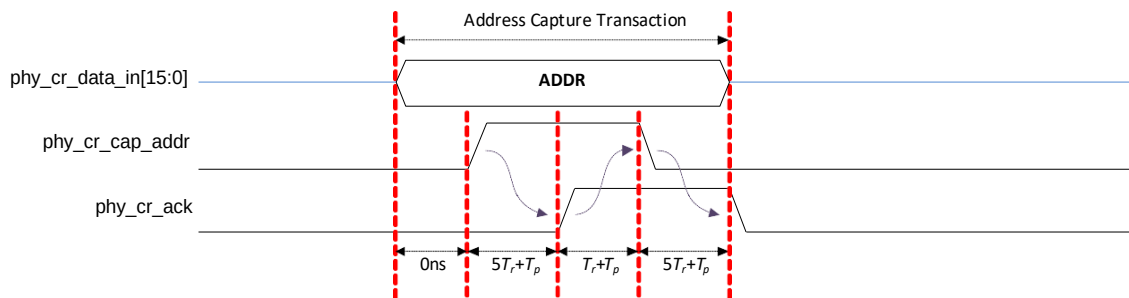
步骤中的 1us 为配置举例，软件可以配置延时范围为>1us。

39.3.4.2 ComboPhy 的寄存器写操作

写寄存器操作分为 2 部分，分别是地址的锁存和参数写入。



图39-4 地址 Capture 传输 CR 时序



说明

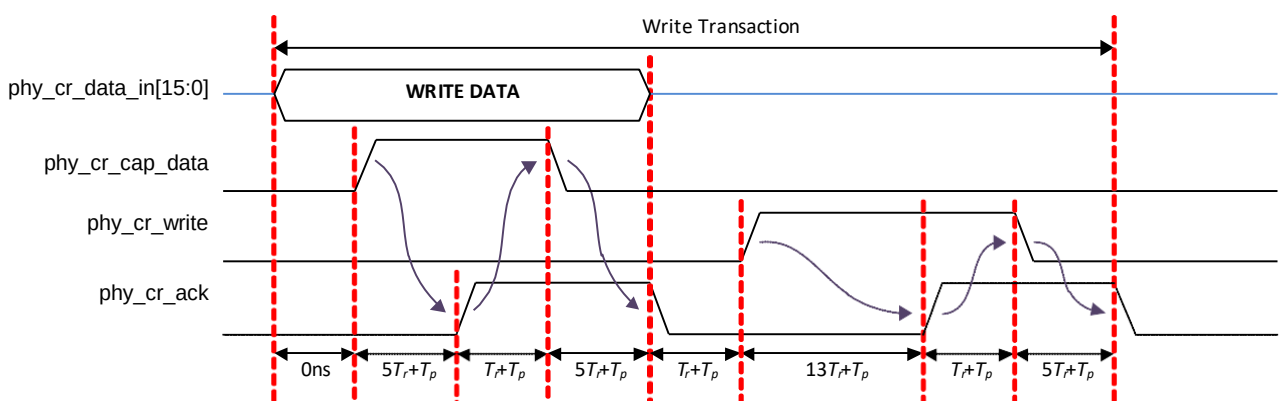
T_p 为传播延时 1ns, T_r 为参考时钟的周期 10ns (参考时钟为 100M)。

地址锁存操作请参考图 39-4, 步骤如下:

- 步骤 1 写入 phy_cr_data_in, 内容为目标参数的地址。
- 步骤 2 写 phy_cr_cap_addr 为 1, 延时 1us。
- 步骤 3 读 phy_cr_ack, 若为 1, 则写 phy_cr_cap_addr 为 0, 并延时 1us; 若为 0, 则表明 cr 接口异常, 停止操作。
- 步骤 4 读 phy_cr_cak, 若为 0, 则操作成功; 若为 1, 则表明 cr 接口异常, 停止操作。

----结束

图39-5 写操作 CR 时序



参数写入操作请参考图 39-5, 步骤如下:

- 步骤 1 写入 phy_cr_data_in, 内容为目标参数的“写入数据”。



步骤 2 写 phy_cr_cap_data 为 1, 延时 1us。

步骤 3 读 phy_cr_ack, 若为 1, 则写 phy_cr_cap_data 为 0, 并延时 1us; 若为 0, 则表明 cr 接口异常, 停止操作。

步骤 4 读 phy_cr_cak, 若为 0, 则进入下一步; 若为 1, 则表明 cr 接口异常, 停止操作。

步骤 5 写 phy_cr_cap_write 为 1, 延时 1us。

步骤 6 读 phy_cr_ack, 若为 1, 则写 phy_cr_cap_write 为 0, 并延时 1us; 若为 0, 则表明 cr 接口异常, 停止操作。

步骤 7 读 phy_cr_cak, 若为 0, 则操作成功; 若为 1, 则表明 cr 接口异常, 停止操作。

----结束

须知

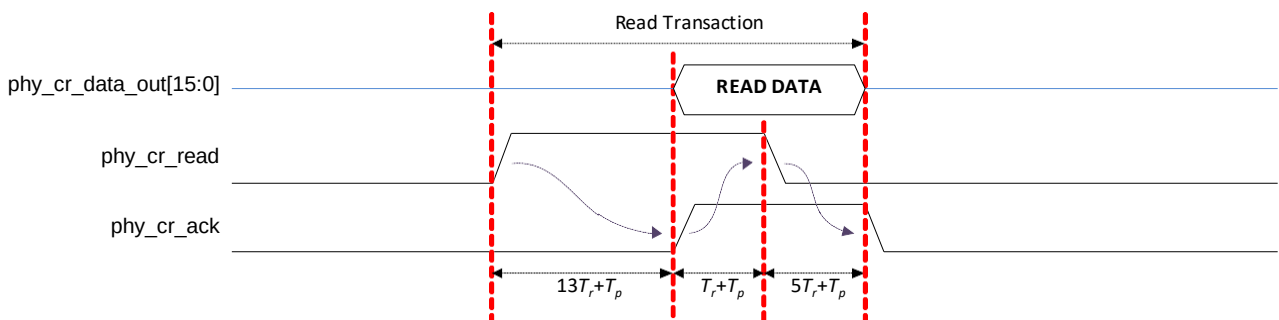
GMAC PHY 使用该 cr 接口步骤需将 phy0_cr_cap_en 或 phy0_cr_cap_en 配置为 1。

步骤中的 1us 为配置举例, 软件可以配置延时范围为 >1us。

39.3.4.3 ComboPhy 的寄存器读操作

读参数操作分为 2 部分, 分别为地址的锁存和参数的读出; 其中地址锁存的操作, 和写操作一致; 参数的读出请参考图 39-5。

图39-6 读操作 CR 时序



参数读出操作:

步骤 1 写 phy_cr_cap_read 为 1, 延时 1us。



步骤 2 读 phy_cr_ack, 若为 1, 读 phy_cr_data_out (获取参数值), 写 phy_cr_cap_read 为 0, 延时 1us; 若为 0, 则表明 cr 接口异常, 停止操作。

步骤 3 读 phy_cr_cak, 若为 0, 则进入下一步; 若为 1, 则表明 cr 接口异常, 停止操作。

----结束

须知

GMAC PHY 使用该 cr 接口步骤需将 phy0_cr_cap_en 或 phy0_cr_cap_en 配置为 1。
步骤中的 1us 为配置举例, 软件可以配置延时范围为>1us。

39.3.5 ComboPhy 的 powerdown

SGMII、PCIE、USB3 的 powerdown 流程是一样的, 均按照手册要求的流程。

39.3.5.1 Powerdown 流程

Powerdown 操作随时都可进行。

步骤 1 配置 powerdown 寄存器为 1。

----结束

39.3.5.2 Powerup 流程

Powerdown 后, 如果需要 powerup 需按照下面流程操作:

步骤 1 PHY 全局复位: 配置 phy_reset 为 1。

步骤 2 延时大于 50ns 后, 配置 powerdown 寄存器为 0。

步骤 3 延时大于 10us 后, PHY 解除复位, 配置 phy_reset 为 0。

----结束

39.3.5.3 相关寄存器

PCIE0: 0x8b21038, bit6, 配置为 1 为 test_powerdown;

PCIE1: 0xcd0e038, bit6, 配置为 1 为 test_powerdown;

GMAC2 地址 0xcd0c160, bit27, 配置为 1 为 test_powerdown;



GMAC3 地址 0xcd0c164, bit27, 配置为 1 为 test_powerdown;

USB3 地址 0xc0d7004, bit16, 配置为 1 为 test_powerdown;

PHY RESET 寄存器请参考 39.3.6 ComboPhy 的寄存器。

须知

USB0、1、2、3 的 USB2.0 PHY 不具备 powdown 功能。

39.3.6 ComboPhy 的寄存器

给出本节所用到的, ComboPhy 的寄存器和系统寄存器地址。

39.3.6.1 ComboPhy 上电流程用到寄存器

ComboPhy 的都有 Phy Reset。对于 PCIE 和 USB3 PHY 为 PIPE 接口, 还有 1 个 PiPe 接口的复位。

SGMII 寄存器

寄存器参考 HP_SUBSYS 的 GMAC_PHY_CFG1 和 GMAC_PHY_CFG2。

PCIE 寄存器

表 39-8 给出了 PCIE 的 phy reset, 偏移地址 0x218, 基地址参考 SYSCTRL 寄存器。

表39-8 PCIE PHY 的 phy reset

寄存器名称	域段名称	Bit 地址	属性	含义描述
PROBE_CTRL	Pcie1_phy_reset_n	31	RW	pcie1 phy 复位。 0: pcie1 phy 复位; 1: pcie1 phy 不复位。
0x218	pcie0_phy_reset_n	30	RW	pcie0 phy 复位。 0: pcie0 phy 复位; 1: pcie0 phy 不复位。

参考 HOST_SUBSYS 寄存器。



Pipe reset 上电默认为复位状态；默认为 crg 控制；

Pipe reset 操作中，复位和解复位都操作不同寄存器，并在操作完成查询复位状态。当复位状态跟预期不符，重新操作复位和解复位。

表39-9 PCIE PHY 的 Pipe reset

寄存器名称	域段名称	Bit 地址	属性	含义描述
PCIE_PHY_RST_CFG 0x190	reserved	31:2	RO	-
	pcie1_phy_rst_sel	1	RW	pcie1 phy 复位源选择。 0: 选择 crg 复位; 1: 选择控制器复位。 默认为 0。
	pcie0_phy_rst_sel	0	RW	pcie0 phy 复位源选择。 0: 选择 crg 复位; 1: 选择控制器复位。 默认为 0。
HOST_RST_ENABLE 0x1c	reserved	31:9	RO	-
	pcie1_phy_rst_en	8	WO	pcie1 phy 复位使能信号，写 1 有效。
	pcie0_phy_rst_en	7	WO	pcie0 phy 复位使能信号，写 1 有效。
HOST_RST_DISABLE 0x10	reserved	31:9	RO	-
	pcie1_phy_rst_dis	8	WO	pcie1 phy 复位关闭信号，写 1 有效。
	pcie0_phy_rst_dis	7	WO	pcie0 phy 复位关闭信号，写 1 有效。
HOST_RST_STATE	reserved	31:9	RO	-



寄存器名称	域段名称	Bit 地址	属性	含义描述
0x14	pcie1_rst_state	8	RO	pcie1 phy 复位状态。 0：复位无效； 1：复位有效。 默认为 1。
	pcie0_rst_state	7	RO	pcie0 phy 复位状态。 0：复位无效； 1：复位有效。 默认为 1。

USB3 寄存器

Phy reset 具体参考 HP_SUBSYS 寄存器；每组寄存器包括状态、复位请求和复位释放 3 个寄存器。

Phy reset 操作中，复位和解复位都操作不同寄存器，并在操作完成查询复位状态。当复位状态跟预期不符，重新操作复位和解复位。

表39-10 USB3 PHY 的 Phy reset

寄存器名称	域段名称	Bit 地址	属性	含义描述
HP_SRST_ST2 0x4	usb3_cmb_phy_rst_st	13	RO	usb3 drd 的 comb phy 的软复位查询。
HP_SRST_REQ2 0xc	usb3_cmb_phy_rst_req	13	WO	usb3 drd 的 comb phy 的软复位请求。
HP_SRST_DIS2 0x14	usb3_cmb_phy_rst_dis	13	WO	usb3 drd 的 comb phy 的软复位释放请求。

参考 MISC_USB3_PHY 寄存器。



表39-11 USB3 PHY 的 Pipe reset

寄存器名称	域段名称	Bit 地址	属性	含义描述
USB3_PHY_CTRL5 0x1014	pipe_reset_n	21	RW	pipe 复位

39.3.6.2 ComboPhy SI 参数控制寄存器

SGMII 寄存器

SGMII Combo Phy 不支持 DFE 功能，但相关的 SI 参数可配置的。

文中所给的 SI 配置值为 IP 推荐值，具体应用时，仍需要先根据链路情况调整，选择合适的参数方可保证链路的健壮性。

表39-12 SGMII 参数寄存器 1

寄存器名称	域段名称	Bit 地址	默认值	推荐值	含义描述
GMAC_PHY0_LOS_CFG	tx_vboost_lvl_0	17:15	0x4	默认 0x4	Vboost 助推电平门限设置。011: Corresponds to launch amplitude of 0.844 V; 100 (default): Corresponds to launch amplitude of 1.008 V; 101: Corresponds to launch amplitude of 1.156 V。
	rx0_los_filt_cnt_0	14:8	0x05	默认 0x5	LOS 信号过滤周期，单位为 20UI; 当检测到原始 LOS，需持续该时长，LOS 才会置为 1; 当配置为 0，则 LOS 信号过滤不使能。



寄存器名称	域段名称	Bit 地址	默认值	推荐值	含义描述
	los_bias_0	7:5	0x2	默认 0x2	LOS 的检测电平门限。 000: Reserved; 001: 45 mV; 010: 60 mV; 011: 75 mV; 100: 90 mV; 101: 105 mV; 110: 120 mV; 111: 135 mV。
	los_level_0	4:0	0x09	默认 0x9	los 的检测敏感度设置; 保持为默认, 禁止配置。

说明

GMAC0 的偏移地址为 0x158; GMAC1 的偏移地址为 0x15C。

表39-13 SGMII 参数寄存器 2

寄存器名称	域段名称	Bit 地址	默认值	推荐值	含义描述
GMAC_PHY0_CFG2	tx0_preemph_0	22:16	0x00	默认 0x0	Tx Pre_emphasis 默认值为 7' b0; 根据链路情况进行配置。
	tx0_detect_rx_req_0	15	0x1	默认 0x1	由内部寄存器控制, 默认值为 1。
	tx0_amplitude_0	14:8	0x00	默认 0x0	TX Amplitude 默认值为 7'b0; 根据链路情况进行配置。



寄存器名称	域段名称	Bit 地址	默认值	推荐值	含义描述
	phy_tx0_term_offset_0	7:3	0x00	默认 0x0	txN_term_offset: 范围 0~31 由控制寄存器控制; 默认值为 5'b00000; 后续根据链路情况进行配置。
	phy_rx0_eq_0	2:0	0x0	配置 0x2	RX 均衡设置; 推荐值为 2 由内部寄存器控制, 后续根据链路情况 进行配置。

说明

GMAC0 的偏移地址为 0x168; GMAC1 的偏移地址为 0x16C。

须知

模拟参数配置过程中, 需要提前将 Phy 复位住。在 Phy 工作期间, 参数不可改配。

PCIE 寄存器

Combo Phy 不支持 DFE 功能, 但相关的 SI 参数可配置的。

文中所给的 SI 配置值为 IP 推荐值, 具体应用时, 仍需要先根据链路情况选择合适的参数方可保证链路的健壮性。

须知

PCIE 参数配置使用到 PCIE MISC 寄存器,因此需要先解复位 pcie0_core_rst_en 和 pcie1_core_rst_en, 详细操作请参考 HOST_SUB 寄存器。

同时, 当 PCIE 控制器复位之后 (可能原因 host 复位, pcie 管脚复位) PCIE 的参数需要重新配置, 并且重新复位 serdes。



PCIE 复位会产生中断 Pice1_rst_pos_int 或 Pcie0_rst_pos_int, 参考《Hi3093 中断向量表》

表39-14 PCIE_PHYCTL0 寄存器 (偏移地址 0x30)

寄存器名称	默认值	字段	Bit	域段默认	建议配置值 PCIE PHY 手册提供	描述
PCIE_PHYCTL0	0x420539F3	pcs_tx_deemph_gen1	31:26	0x10	配置 0x18	当 pipeP_tx_deemph 为 1 时且在 Gen1 状态下通过此值设置预加重。
		pcs_tx_deemph_gen2_3p5db	25:20	0x20	配置 0x18	当 pipeP_tx_deemph 为 1 时且在 Gen2 状态下通过此值设置预加重。
		pcs_tx_deemph_gen2_6db	19:14	0x14	配置 0x23	当 pipeP_tx_deemph 为 0 时且在 Gen2 状态下通过此值设置预加重。
		pcs_tx_swing_full	13:7	0x73	配置 0x7c	当 pcs_tx_swing=0 时(full swing mode), 通过此值设置 Tx 信号的摆幅。
		pcs_tx_swing_low	6:0	0x73	配置 0x48	当 pcs_tx_swing=1 时(low swing mode), 通过此值设置 Tx 信号的摆幅。



表39-15 PCIE_PHYCTL1 寄存器 (偏移地址 0x34)

寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
PCIE_PHYCTL1	0x31924114	pipe0_tx2rx_loopbk	31	0x0	-	Tx to Rx 环回
		pipe0_tx_deemph_highbit	30	0x0	-	TX 预加重配置高 bit
		参考寄存器	29:19	0x0	-	保留
		phy_los_leve	18:14	0x09	默认 0x9	los 信号检测控制
		phy_tx0_term_offset	13:9	0x00	默认 0x0	TX 传输边界补偿
		pipe_en	8	0x1	-	PIPE 接口使能信号
		pcs_clk_req	7	0x0	-	mpll_world_clk 请求
		pcs_common_clocks	6	0x0	-	选择 PCS 数据接收通道和 PCIe link 使用共同的参考时钟
		phy_los_bias	5:3	0x2	默认 0x2	PHY loss 信号阈值控制
		phy_tx_vboost_lv	2:0	0x4	默认 0x4	Tx 电压阈值控制



表39-16 PCIE_PHYCTL2 寄存器 (偏移地址 0x38)

寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
PCIE_PHYCTL2	0x00010100	参考寄存器	31:10	0x0	-	
		phy_rx0_eq	9:7	0x2	默认 0x2	Rx 均衡
		phy_test_power down	6	0x0	-	环路电源控制, 在测试状态下, 关闭所有环路电源
		phy_test_bypass	5	0x0	-	PHY 测试 bypass 信号
		phy_test_burnin	4	0x0	-	PHY test 信号
		reserved_38_0	3	0x0	-	保留
		phy_rtune_req	2	0x0	-	PHY 参考电阻反转请求
		pciephy_cfg_ssc_en	1	0x0	-	Serdes ssc 使能, 高有效。
		phy_res_req_in	0	0x0	-	公用外部参考电阻应答请求

USB3 寄存器

Combo Phy 不支持 DFE 功能, 但相关的 SI 参数可配置的。



文中所给的 SI 配置值为 IP 推荐值，具体应用时，仍需要先根据链路情况选择合适的参数方可保证链路的健壮性。

参考 “dwc_ssp_phy_tsmc28hpc_09V18V_databook” 4.5 章节。

寄存器来自 USB3_MISC_USB3_PHY 的寄存器。

须知

USB PHY 参数配置使用到 USB MISC 寄存器,因此需要先解复位 usb3_apb_rst_srst, 详细操作请参考 HP_SUBSYS 寄存器。

表39-17 USB 参数控制寄存器 1 (偏移地址 0x1010)

寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
USB3_PHY_CTRL4	0x046C7D09	lane_tx_term_offset	31:27	0x00	默认	发送端端电阻偏移保留，固定值 5'b00000
		pcs_tx_deemph_6db	26:21	0x23	0x23	6db 时刻的去加重值 TX De-Emphasis at 6 dB Function: This static value sets the TX driver de-emphasis value when pipeP_tx_deemph[1:0] is set to 2'b10 (according to the PIPE3 specification). This bus is provided for completeness and as a second potential launch amplitude. Default: 23



寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
		pcs_tx_dee_mph_3p5db	20:15	0x18	0x18	3.5db 时刻的去加重值 TX De-Emphasis at 3.5 dB Function: This static value sets the TX driver de-emphasis value when Default: 22
		pcs_rx_los_mask_val	14:5	0x3E8	默认	可配置的 LOS 滤波宽度 Configurable Loss-of-Signal Mask Width For normal operation, set to a targeted mask interval of 10 us (value = 10 us / Tref_clk). If the ref_clkdiv2 signal is used, the value = 10 us / (2 * Tref_clk)
		los_level	4:0	0x09	0x9	LOS 检测敏感电平控制。 保留，固定位 0x09

表39-18 USB 参数控制寄存器 2 (偏移地址 0x1014)

寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
USB3_PHY_CTRL5	0x000000FF	pipe_tx_dee_mph	31:30	0x0	-	pipe 发送端预加重控制
		pipe_powerdown	29:28	0x0	-	pipe powerdown 控制



寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
		pipe_tx_ones_zeros	27	0x0	-	pipe 发送端 0/1 间隔向量的发送
		pipe_tx_elecidle	26	0x0	-	发送端发送空闲电平
		pipe_tx_detectrx	25	0x0	-	发送端接收电阻检测
		pipe_rx_termination	24	0x0	-	接收端电阻使能
		pipe_rx_polarity	23	0x0	-	接收端极性控制
		pipe_rx_eq_training	22	0x0	-	接收端均衡训练
		pipe_reset_n	21	0x0	-	pipe 复位
		pipe_elasticity_buf_mode	20	0x0	-	弹性 buffer 模式选择
		ssc_range	19:17	0x0	-	展频时钟范围
		ssc_ref_clk_sel	16:8	0x000	-	展频参考时钟相位移动
		ssc_en	7	0x1	-	展频使能设置为 1.



寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
		pcs_tx_swing_full	6:0	0x7F	默认	PCS 发送端摆幅 TX Amplitude (Full Swing Mode) Function: This static value sets the launch amplitude of the transmitter . To enable tuning at the board level for RX eye compliance, connect this signal to a register. Default: 120 (7b'1111000)

表39-19 USB 参数控制寄存器 3（偏移地址 0x1024）

寄存器名称	默认值	字段	Bit	域段默认	建议配置值	描述
USB3_PHY_CTRL9	0x00000610	参考寄存描述	31:8	-	-	保留
-	-	rx0_eq	7:5	0x0	-	接收端，均衡值

USB 手册推荐固定 RX 均衡，通过 CR 接口进行配置：

- 步骤 1 配置 RX_OVRD_IN_HI.RX_EQ_EN 为 0，寄存器位于[address 16'h1006: bit 6]。
- 步骤 2 配置 RX_OVRD_IN_HI.RX_EQ_EN_OVRD 为 1，寄存器位于[address 16'h1006: bit 7]。



步骤 3 配置 RX_OVRD_IN_HI.RX_EQ 为 2~4 的一个值，建议 2，寄存器位于[address 16'h1006: bits 8–10] [equalization setting]。

步骤 4 配置 1'b1 to RX_OVRD_IN_HI .RX_EQ_OVRD 为 1，寄存器位于[address 16'h1006: bit11]。

----结束

39.3.7 ComboPhy 参数 Override

Phy (serdes) 的参数配置可以通过配置内部寄存器进行重载入。当使能 serdes 内部重载寄存器时，系统寄存器的配置值就失效了，serdes 将使用内部寄存器配置的参数进行工作。

Override 寄存器可通过 ComboPhy 的 CR 接口继续访问和操作。

表39-20 TX 驱动参数重载寄存器（偏移地址 0x1002）

Bits	Name	Memory Access	Description
15	RESERVED_15_15	R	Description: Reserved for Future use and actual reset value is 0xX Value After Reset: 0x0 Exists: Always Reset Mask: 0x0
14	EN	R/W	Description: Enable override values for all inputs controlled by this register Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
13:7	PREEMPH	R/W	Description: Override value for transmit preemphasis. Value After Reset: 0x0 Exists: Always Reset Mask: 0x7f
6:0	AMPLITUDE	R/W	Description: Override value for transmit amplitude. Value After Reset: 0x0 Exists: Always Reset Mask: 0x7f



表39-21 TX 驱动参数重载寄存器 (偏移地址 0x1003)

Bits	Name	Memory Access	Description
15:6	RESERVED_15_6	R	Description: Reserved for Future use and actual reset value is 0xX Value After Reset: 0x0 Exists: Always Reset Mask: 0x0
5	EN	R/W	Description: Enable override values for all inputs controlled by this register Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
4:0	TERM_OFFSET	R/W	Description: Override value for tx_term_offset Value After Reset: 0x0 Exists: Always Reset Mask: 0x1f

表39-22 RX 驱动参数重载寄存器 (偏移地址 0x1006)

Bits	Name	Memory Access	Description
15:14	RESERVED_15_14	R	Description: Reserved for Future use and actual reset value is 0xX Value After Reset: 0x0 Exists: Always Reset Mask: 0x0
13	RX_RESET_OVRD	R/W	Description: Override enable for rx_reset Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
12	RX_RESET	R/W	Description: Override value for rx_reset Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
11	RX_EQ_OVRD	R/W	Description: Override enable for rx_eq Value After Reset: 0x0 Exists: Always Reset Mask: 0x1



Bits	Name	Memory Access	Description
10:8	RX_EQ	R/W	Description: Override value for rx_eq Value After Reset: 0x0 Exists: Always Reset Mask: 0x7
7	RX_EQ_EN_OVRD	R/W	Description: Override enable for rx_eq_en Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
6	RX_EQ_EN	R/W	Description: Override value for rx_eq_en Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
5	RX_LOS_FILTER_OVRD	R/W	Description: Override enable for rx_los_filter Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
4:3	RX_LOS_FILTER	R/W	Description: Override value for rx_los_filter Value After Reset: 0x0 Exists: Always Reset Mask: 0x3
2	RX_RATE_OVRD	R/W	Description: Override enable for rx_rate Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
1:0	RX_RATE	R/W	Description: Override value for rx_rate Value After Reset: 0x0 Exists: Always Reset Mask: 0x3

39.4 Serdes 环回工作模式

39.4.1 环回模式概述

Combo Phy 只有一种环回模式，分别为：Serial Loopback。



PCIE 和 USB 称为 pipeP_tx2rx_loopbk。

SGMII 称为 lane0_tx2rx_loopbk。

39.4.2 Loopback 寄存器

环回控制是通过管脚给到 PHY，因此 loopback 的寄存器在控制器或各个 subsys。

USB3 参考 MISC_USB3_PHY 的 USB3_PHY_CTRL1. lane_tx2rx_loopbk（偏移地址 0x1004，bit[5]）；

PCIE 参考 PCIE_MISC 的 PCIE_PHYCTL1. pipe0_tx2rx_loopbk（偏移地址 0x34，bit[31]）；

SGMII0，参考 HP_SUBSYS 的 GMAC_PHY0_CFG. lane0_loopbk_en_0（偏移地址 0x160，bit[15]）；

SGMII1，参考 HP_SUBSYS 的 GMAC_PHY1_CFG. lane0_loopbk_en_1（偏移地址 0x160，bit[15]）。

39.4.2.1 Loopbak Override

Loopback 也有 Override 寄存器，请参考 Serdes 寄存器中的 LANEN_DIG_TX_OVRD_IN_LO。

39.5 Serdes 工作状态查询

39.5.1 SGMII combo phy

参考 HP_SUBSYS 寄存器。

表39-23 SGMII PHY 的状态信号

寄存器名称	偏移地址	域段名称	Bit	含义
GMAC_PHY0_STS	0x0170	reserved	31:8	GMAC2 PHY 的状态信号。
-	-	rx_los_0	7	phy 的 los 信号。
		rx_valid_0	6	phy 的 valid 信号。1 表示信号锁定，RxData 为有效。



寄存器名称	偏移地址	域段名称	Bit	含义
		mpll_state_0	5	mpll 锁定状态上报。1 表示 PLL 锁定。
		rx0_pll_state_0	4	Receive DPLL 锁定状态上报 当为 0 表示 rx_clk 可能没工作或者不在目标频率。
		tx0_cm_state_0	3	TX 侧共模状态上报。 0: electirc idle 状态下, 发送差分对处于弱下拉 (通过一个高阻接地) ; 1: electirc idle 状态下, 发送差分对保持在 vptxN 的 1/2 电平。
		tx0_detect_rx_ack_0	2	detect 应答信号 当 txN_detectrx_req 置为 1, 该 bit 清 0。
		tx0_detect_rx_result_0	1	detect 结果上报。
		tx0_state_0	0	tx 方向发送时钟和数据采样状态。 1 表明 TX 准备好发送数据。
GMAC_PHY1_STS	0x0174	reserved	31:8	GMAC3 PHY 的状态信号。
-	-	rx_los_1	7	phy 的 los 信号。
		rx_valid_1	6	phy 的 valid 信号。
		mpll_state_1	5	mpll 锁定状态上报。
		rx0_pll_state_1	4	Receive DPLL 锁定状态上报 当为 0 表示 rx_clk 可能没工作或者不在目标频率。
		tx0_cm_state_1	3	TX 侧共模状态上报。 0: electirc idle 状态下, 发送差分对处于弱下拉 (通过一个高阻接地) ; 1: electirc idle 状态下, 发送差分对保持在 vptxN 的 1/2 电平。



寄存器名称	偏移地址	域段名称	Bit	含义
		tx0_detect_rx_ack_1	2	detect 应答信号。
		tx0_detect_rx_result_1	1	detect 结果上报。
		tx0_state_1	0	tx 方向发送时钟和数据采样状态; 1 表明 TX 准备好发送数据。

39.5.2 PCIe combo phy

表39-24 PCIe PHY 的状态信号

寄存器名称	偏移地址	域段名称	Bit	含义
PCIE_STATUS0	0x80	-	-	-
-	-	pcie_xmlh_link_up	15	PCIe (X1) 控制器。 PHY Link up/down 指示。 0: link down; 1: link up。
-	-	pcie_rdlh_link_up	5	PCIe (X1) 控制器。 数据 Link 层 up/down 指示。 0: link down; 1: link up。

表39-25 PCIe PHY 的状态信号

寄存器名称	偏移地址	域段名称	Bit	含义
PCIE_STATU S4	0x94	phy_mac_rxd ata	31:16	PHY 输出到 PCIe Controller 的 rxdata 信号。
		phy_mac_rxd atak	15:14	PHY 输出到 PCIe Controller 的 rxdatak 信号。
		phy_mac_rxv alid	13	PHY 输出到 PCIe Controller 的 rxvalid 信号。



寄存器名称	偏移地址	域段名称	Bit	含义
		phy_mac_phy_status	12	PHY 输出到 PCIe Controller 的 phystatus 信号。
		phy_mac_rxeidle	11	PHY 输出到 PCIe Controller 的 rxelecidle 信号。
		phy_mac_rxstatus	10:8	PHY 输出到 PCIe Controller 的 rxstatus 信号。 000: 接收数据有效; 001: 检测到 1 个 SKP 符号插入; 010: 检测到 1 个 SKP 符号清除; 011: Receiver detected; 100: 8B/10B 编码错误; 101: 弹性 buffer 上溢; 110: 弹性 buffer 下溢, 弹性 buffer 工作在 normal 模式, 该错误不会发生; 111: 极性错误。
		mac_phy_rate	7:6	指示当前 PCIe 控制器的速率。 00: 2.5Gbps; 01: 5.0Gbps; 10: 8.0Gbps;



寄存器名称	偏移地址	域段名称	Bit	含义																																
		xmlh_ltssm_state	5:0	指示当前 PCIe 控制器 ltssm 状态 link up 成功的状态是：0x11，具体状态如下：																																
				<table><tr><td>DETECT_QUIET 0x00</td><td>RCVRY_EQ1 0x21</td></tr><tr><td>DETECT_ACT 0x01</td><td>RCVRY_EQ2 0x22</td></tr><tr><td>POLL_ACTIVE 0x02</td><td>RCVRY_EQ3 0x23</td></tr><tr><td>POLL_COMPLIANCE 0x03</td><td>L0 0x11</td></tr><tr><td>POLL_CONFIG 0x04</td><td>L0S 0x12</td></tr><tr><td>PRE_DETECT_QUIET 0x05</td><td>L123_SEND_IDLE 0x13</td></tr><tr><td>DETECT_WAIT 0x06</td><td>L1_IDLE 0x14</td></tr><tr><td>CFG_LINKWD_START 0x07</td><td>L2_IDLE 0x15</td></tr><tr><td>CFG_LINKWD_ACEPT 0x08</td><td>L2_WAKE 0x16</td></tr><tr><td>CFG_LANENUM_WAIT 0x09</td><td>DISABLED_ENTRY 0x17</td></tr><tr><td>CFG_LANENUM_ACEPT 0x0A</td><td>DISABLED_IDLE 0x18</td></tr><tr><td>CFG_COMPLETE 0x0B</td><td>DISABLED 0x19</td></tr><tr><td>CFG_IDLE 0x0C</td><td>LPBK_ENTRY 0x1A</td></tr><tr><td>RCVRY_LOCK 0x0D</td><td>LPBK_ACTIVE 0x1B</td></tr><tr><td>RCVRY_SPEED 0x0E</td><td>LPBK_EXIT 0x1C</td></tr><tr><td>RCVRY_RCVRCFG 0x0F</td><td>LPBK_EXIT_TIMEOUT 0x1D</td></tr><tr><td>RCVRY_IDLE 0x10</td><td>HOT_RESET_ENTRY 0x1E</td></tr><tr><td>RCVRY_EQO 0x20</td><td>HOTRESET 0x1F</td></tr></table>	DETECT_QUIET 0x00	RCVRY_EQ1 0x21	DETECT_ACT 0x01	RCVRY_EQ2 0x22	POLL_ACTIVE 0x02	RCVRY_EQ3 0x23	POLL_COMPLIANCE 0x03	L0 0x11	POLL_CONFIG 0x04	L0S 0x12	PRE_DETECT_QUIET 0x05	L123_SEND_IDLE 0x13	DETECT_WAIT 0x06	L1_IDLE 0x14	CFG_LINKWD_START 0x07	L2_IDLE 0x15	CFG_LINKWD_ACEPT 0x08	L2_WAKE 0x16	CFG_LANENUM_WAIT 0x09	DISABLED_ENTRY 0x17	CFG_LANENUM_ACEPT 0x0A	DISABLED_IDLE 0x18	CFG_COMPLETE 0x0B	DISABLED 0x19	CFG_IDLE 0x0C	LPBK_ENTRY 0x1A	RCVRY_LOCK 0x0D	LPBK_ACTIVE 0x1B	RCVRY_SPEED 0x0E	LPBK_EXIT 0x1C	RCVRY_RCVRCFG 0x0F	LPBK_EXIT_TIMEOUT 0x1D
DETECT_QUIET 0x00	RCVRY_EQ1 0x21																																			
DETECT_ACT 0x01	RCVRY_EQ2 0x22																																			
POLL_ACTIVE 0x02	RCVRY_EQ3 0x23																																			
POLL_COMPLIANCE 0x03	L0 0x11																																			
POLL_CONFIG 0x04	L0S 0x12																																			
PRE_DETECT_QUIET 0x05	L123_SEND_IDLE 0x13																																			
DETECT_WAIT 0x06	L1_IDLE 0x14																																			
CFG_LINKWD_START 0x07	L2_IDLE 0x15																																			
CFG_LINKWD_ACEPT 0x08	L2_WAKE 0x16																																			
CFG_LANENUM_WAIT 0x09	DISABLED_ENTRY 0x17																																			
CFG_LANENUM_ACEPT 0x0A	DISABLED_IDLE 0x18																																			
CFG_COMPLETE 0x0B	DISABLED 0x19																																			
CFG_IDLE 0x0C	LPBK_ENTRY 0x1A																																			
RCVRY_LOCK 0x0D	LPBK_ACTIVE 0x1B																																			
RCVRY_SPEED 0x0E	LPBK_EXIT 0x1C																																			
RCVRY_RCVRCFG 0x0F	LPBK_EXIT_TIMEOUT 0x1D																																			
RCVRY_IDLE 0x10	HOT_RESET_ENTRY 0x1E																																			
RCVRY_EQO 0x20	HOTRESET 0x1F																																			
				注：该寄存器的值在运行时会动态变化。																																

39.5.3 USB3 combo phy

参考 MISC_USB3_PHY 寄存器。



表39-26 USB3 PHY 的状态信号

寄存器名称	偏移地址	域段名称	Bit	含义
USB3_PHY_STATUS1	0x1048	reserved	31:17	保留
-	-	rtune_ack	16	-
		ref_clkreq_n	15	-
		alt_clk_req	14	-
		pipe_rx_valid	13	PIPE Receive Data Valid Indicates symbol lock and valid data on RxData and RxDataK.
		pipe_power_present	12	PIPE Power Present This PIPE output indicates the presence of VBUS and is derived directly from the power_presentN input
		pipe_phystatus	11	PIPE PHY Status Communicates completion of several PHY functions including power management state transitions, rate change, and receiver detection. When this signal transitions during entry and exit from P2 state and PCLK is not running, signaling is asynchronous. In error situations (where the PHY fails to assert PhyStatus), the MAC can take MACspecific error recovery actions.
		reserved	10	-
		pipe_data_bus_width	9:8	-
		pipe_rx_elecidle	7	PIPE Receiver Electrical Idle Indicates receiver detection of electrical idle. If this signal is deasserted while the PHY is in the P2 state, indicates the detection of an LFPS or a beacon



寄存器名称	偏移地址	域段名称	Bit	含义
		pipe_rx_status	6:4	PHY 输出到 PCIe Controller 的 rxstatus 信号 000: 接收数据有效; 001: 检测到 1 个 SKP 符号插入; 010: 检测到 1 个 SKP 符号清除; 011: Receiver detected; 100: 8B/10B 编码错误; 101: 弹性 buffer 上溢; 110: 弹性 buffer 下溢, 弹性 buffer 工作在 normal 模式, 该错误不会发生; 111: 极性错误。
		pipe_rx_datak	3:0	PIPE RX Data-K 码。

39.6 PRBS 测试

Hi3093 中一共集成了 5 个 combo phy (SerDes), 每个 phy 有 1 条 lane。由于在 PRBS 模式下, serdes 的测试方法是一致的, 但对于状态信号的上报有所差异, 文中会单独列出。

须知

在下面的测试步骤中, 只是表面要进行正确测试, 在配置步骤中设置的值是在寄存器默认配置上增加的, 所以不要修改先前的配置值, 这些配置值是增量配置。

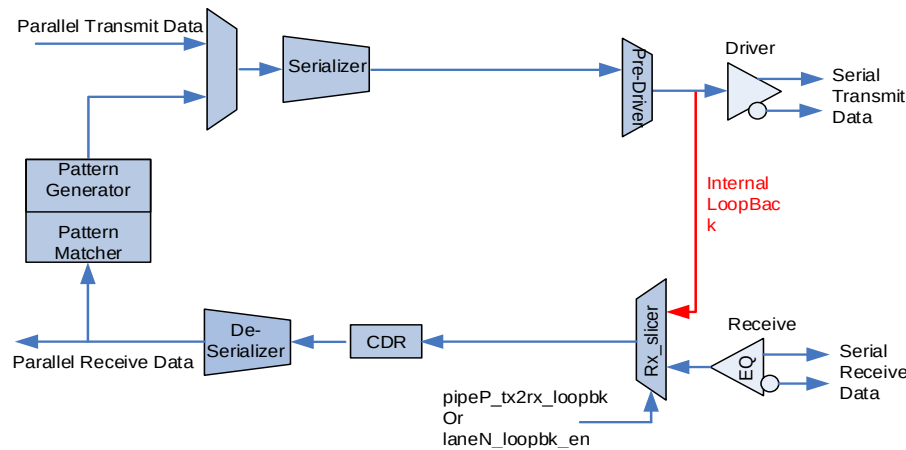
SerDes 本身提供了内部环回, 如果是外部管脚环回需要外部环境支持, 比如单板上有测试的 SMA 头, 可以将 TX 和 RX 对接, 或者和 SerDes 对接的 PHY 可以配置环回模式。在这里, 只描述内部环回模式下的 PRBS 测试方法。



39.6.1 测试原理

测试原理图如表 39-27 所示。

表39-27 PRBS 测试原理图



39.6.2 测试寄存器

39.6.2.1 Pattern Generation

可产生 PRBS-7, PRBS-15, 或者 10bit 字节或 20bit 字节测试数据。

寄存器使用请参考表 39-28。

表39-28 寄存器 LANEN_DIG_TX_LBERT_CTL

Bits	Name	Memory Access	Description
15	保留	R	保留
14:5	PAT0	R/W	Description: 10 bit pattern for modes using PAT0 (5-7). Ignored for other MODEs. Value After Reset: 0x0 Exists: Always Reset Mask: 0x3ff



Bits	Name	Memory Access	Description
4	TRIGGER_ERR	R/W	Description: Single shot inversion the LSB of the current symbol. Any write of a 1 to this bit will insert an error. Maintain mode and pattern from setup. Value After Reset: 0x0 Exists: Always Reset Mask: 0x1
3:0	MODE	R/W	0 - disabled 1 - lfsr31. $X^{31} + X^{28} + 1$ 2 - lfsr23. $X^{23} + X^{18} + 1$ 3 - lfsr15. $X^{15} + X^{14} + 1$ 4 - lfsr7. $X^7 + X^6 + 1$ 5 - fixed word (PAT0) 6 - DC balanced word (PAT0, $\sim$ PAT0) 7 - Fixed pattern: (000, PAT0, 3ff, $\sim$ PAT0) 8 - USB3.0 TSEQ (rx equalizer training ordered sets) Transmits Even, then Odd. 9-15 - RESERVED 注意：在切换测试数据发送模式前，都需要先配置 MODE 为 0。

39.6.2.2 Pattern Matching

可用于校验接收到的测试数据，支持异步的数据 pattern 匹配。寄存器使用请参考表 39-29。

表39-29 LANEN_DIG_RX_LBERT_CTL

Bits	Name	Memory Access	Description
15:5	RESERVED_15_5	R	保留不使用。



Bits	Name	Memory Access	Description
4	SYNC	R/W	Description: 是否匹配标识 写 1 将会清空 error 计数器。无 复位值: 0 Exists: Always Reset Mask: 0x1
3:0	MODE	R/W	Description: 测试数据接收模式 0 - disabled 1 - lfsr31: $X^{31} + X^{28} + 1$ 2 - lfsr23: $X^{23} + X^{18} + 1$ 3 - lfsr15: $X^{15} + X^{14} + 1$ 4 - lfsr7 : $X^7 + X^6 + 1$ 5 - $d[n] = d[n-10]$ 6 - $d[n] = !d[n-10]$ 7 - $d[n] = !d[n-20]$ 8 - 仅 pattern Generator 会使用; Value After Reset: 0x0 Exists: Always Reset Mask: 0xf 注意: 在切换测试数据接收模式前, 都需要先配置 MODE 为 0。

表39-30 LANEN_DIG_RX_LBERT_ERR

Bits	Name	Memory Access	Description
15	OV14	R/W	Description: A read of this register, or a sync of the Pattern Matcher resets the error count. If all bits of this register are 1, the error counter has saturated. If the status is active, multiply COUNT by 128. If OV14=1 and $COUNT=2^{15}-1$, signals overflow of counter (volatile and 2 reads needed to read value) Value After Reset: 0x0 Exists: Always Reset Mask: 0x1 Volatile: true



Bits	Name	Memory Access	Description
14:0	COUNT	R/W	Description: Current error count If OV14 field is active, then multiply count by 128 to get the actual count. (volatile and 2 reads needed to read value) Value After Reset: 0x0 Exists: Always Reset Mask: 0x7fff Volatile: true

39.6.3 测试步骤

通过 CR 接口来进行配置：

步骤 1 设置 RX_OVRD_OUT, 0x1007=0x32

```
VALID_OVRD = 1  
LOS = 1  
LOS_OVRD = 1
```

步骤 2 设置 enable intLBTX_OVRD_IN_LO, 0x1000 = 0x3

```
LOOPBK_EN = 1  
TX_LOOPBK_EN_OVRD = 1
```

步骤 3 #disable aligner

```
RX_OVRD_IN_LO, 0x1005=0x80  
RX_ALIGN_EN_OVRD = 1
```

步骤 4 #Program PG to LFSR7

```
DIG_TX_LBERT_CTL, 0x1015=0x4
```

步骤 5 #Program PM to LFSR7

```
DIG_RX_LBERT_CTL , 0x1016=0x4  
MODE = 0x4
```

步骤 6 #RESET RX

```
DIG_RX_OVRD_IN_HI, 0x1006=0x3000  
RX_RESET_OVRD = 1  
RX_RESET = 1;
```



步骤 7 #DISABLE RESET RX

```
DIG_RX_OVRD_IN_HI, 0x1006=0x2000  
RX_RESET_OVRD = 1  
RX_RESET = 0
```

步骤 8 #SYNC PM

```
DIG_RX_LBERT_CTL, 0x1016=0x14  
SYNC = 1  
MODE = 0x4
```

步骤 9 READ ERROR COUNT

```
RX_LBERT_ERR 0x1017  
bit[15] = saturated  
bit[14:0] = count
```

步骤 10 循环读取 3 次 (间隔>1ms)

```
READ ERROR COUNT  
RX_LBERT_ERR 0x1017  
bit[15] = saturated  
bit[14:0] = count
```

该处应该读到的 RX_LBERT_ERR = 0x0, 表示无误码。

步骤 11 错误注入 #Program PG to LFSR7

```
DIG_TX_LBERT_CTL, 0x1015=0x14  
DIG_TX_LBERT_CTL, 0x1015=0x4
```

步骤 12 循环读取 4 次 (间隔>1ms)

```
READ ERROR COUNT  
RX_LBERT_ERR 0x1017  
bit[15] = saturated  
bit[14:0] = count
```

第一次应该读到 RX_LBERT_ERR 为非 0 值;

后续读到的 RX_LBERT_ERR = 0x0, 表示无误码。

----结束



39.7 接收眼图

本节介绍如何通过 CR 接口来获取眼图数据，以及运用工具绘制眼图。

测试的原理：通过改变 CDR 的相位，调整各个相位上的电压余量。最多使用 64 个相位和 256 个电压，并且在每个点上计算接收到的 1 的数量。

眼图测试分为三步骤：眼图参数设置；眼图数据采集；眼图的绘制。

前两部由软件实现，第三步将采集数据导入工具生成眼图。

39.7.1 眼图参数设置

定义参数，包括 CDR 相位的步进和电压的步进；

例如，ph_inc 为 1，表示相位步进为 1；

Dac_inc 为 4，表示电压幅度的调整步进为 4；

VPH: VPH 电压 1.8V 或 2.5V；

Scope_sel: Before AFE sampling; 1: After AFE sampling。

39.7.2 眼图数据采集

读取 scope counter 值。生成 TDO_output_File 文件。

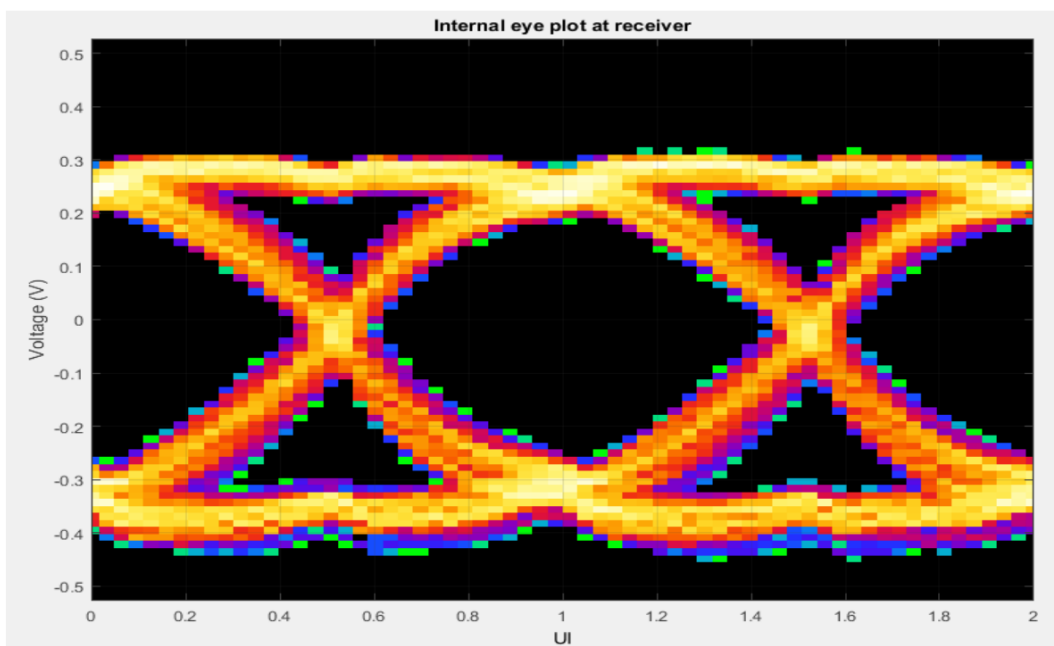
39.7.3 眼图图形生成

可运行 Matlab 脚本来对原始的 output_File 文件画出眼图。

软件可将 output_File 用点阵符号的格式，打印成眼图。

图 39-7 给出了运行 matlab 脚本的运行结果。

图39-7 眼图绘制结果（举例）



须知

GMAC 的 serdes 在正常业务码流下无法绘制眼图。

GMAC 的 serdes 需在接收 PRBS 码流的场景下绘制眼图。

USB 和 PCIE 的 serdes 均可在正常业务码流下绘制眼图。

39.8 Serdes Tuning 启动流程

本节仅涉及部分单板硬件未提供外部参考时钟给所有 serdes 的情况。

无此情况，不用关注此章节，不涉及 Tuning 及初始化流程。

SGMII 和 USB 的流程一致；PCIE 的流程单独体现。

39.8.1 SGMII 和 USB 的 Tuning 启动流程

上电第一次进行下面操作，后续不下电的情况，不用重新走此流程。

思路是所有的 serdes，第一次初始化切到内部参考时钟。

训练完成后，复位所有 serdes，再切到外部参考时钟进行第二次初始化。



39.8.1.1 Tuning 初始化流程

具体步骤：

步骤 1 serdes 初始化前，打开 serdes 内部参考时钟门控。

寄存器 0x08740064 (sys_ctrl) bit12 配置成 1。

步骤 2 serdes 选择内部参考时钟源。

GMAC HP寄存器：GMAC2: 0xcd0c160 bit 28 配置0

GMAC3: 0xcd0c164 bit 28 配置0

USB USB3PHY寄存器 (USB3 MISC寄存器要先解复位)

0xcd07004 bit10 配置成0

步骤 3 Serdes 初始化。

此次初始化的目的是保证 serdes tuning 成功，防止由于外部未供参考时钟，导致无法 tuning。

最后一个 serdes 初始化后再延时 1ms，等待 tuning 完成；

再配置 serdes phy reset 为复位状态。

步骤 4 恢复参考时钟源选择为外部参考，关闭内部参考时钟的门控。

下面寄存器 bit12 配置成 0；0x08740064

GMAC HP 寄存器：

GMAC2: 0xcd0c160 bit 28 配置1；

GMAC3: 0xcd0c164 bit 28 配置1；

USB USB3PHY寄存器 (USB3 MISC寄存器要先解复位)

0xcd07004 bit10 配置成1。

步骤 5 serdes 初始化。

----结束



39.8.1.2 简化流程参考

USB 流程

- 步骤 1 板级提供稳定的电源和参考时钟。
- 步骤 2 配置 HP SUBSYS 寄存器，复位 phy_reset（配置为 0，上电默认为 0）。
- 步骤 3 Serdes 参数配置稳定后，至少 10us，可释放 phy_reset。

----结束

SGMII 流程

- 步骤 1 板级提供稳定的电源和参考时钟。
- 步骤 2 配置 HP SUB 寄存器，复位 phy_reset（配置为 0，上电默认为 0）。
- 步骤 3 延时 10us，释放 phy_reset。

----结束

39.8.2 PCIE 的 Tuning 启动流程

和 SGMII 或 USB 的 Tuning 流程不同，PCIE 的流程是通过手动配置 Tuning 寄存器完成的。

原因是 PCIE 处于 Host 复位域，其初始化时间取决于 Host 上电。

39.8.2.1 Tuning 初始化流程

- 步骤 1 PCIE serdes 解复位，解复位 pcie Core（MISC 寄存器可配）。
- 步骤 2 配置 rtune_req 为高，等待 10us 再拉低；（不必等待 rtune_ack，建议将 rtune_ack 的值记录 log）。
- 此步骤，不关注内部或外部参考时钟。
- 步骤 3 Tuning 完成，进行 PCIE 的 serdes 初始化。

----结束



须知

PCIE0 和 PCIE1 的 tuning 过程，需要保证分开，不要同时进行。

39.8.2.2 参考寄存器

PCIE_MISC0 偏移地址:0x08B21000

PCIE_MISC1 偏移地址:0x0CD0E000

表39-31 描述 PCIE_PHYCTL0 寄存器

Bits	Name	Memory Access	Description
31:26	pcs_tx_deemph_gen1	RW	当 pipeP_tx_deemph 为 1 时且在 Gen1 状态下通过此值设置预加重。
25:20	pcs_tx_deemph_gen2_3p5db	RW	当 pipeP_tx_deemph 为 1 时且在 Gen2 状态下通过此值设置预加重。
19:14	pcs_tx_deemph_gen2_6db	RW	当 pipeP_tx_deemph 为 1 时且在 Gen2 状态下通过此值设置预加重。
13:7	pcs_tx_swing_full	RW	当 pipeP_tx_deemph 为 1 时且在 Gen2 状态下通过此值设置预加重。
6:0	pcs_tx_swing_low	RW	当 pcs_tx_swing=1 时(low swing mode)，通过此值设置 Tx 信号的摆幅。

寄存器 PCIE_PHYCTL0 偏移地址：0x30，寄存器位数：32bit

表39-32 描述 PCIE_PHYCTL2 寄存器

Bits	Name	Memory Access	Description
31:29	reserved_38_2	RW	保留。



Bits	Name	Memory Access	Description
28	app_mac_phy_pclkr eq_n_en	RW	高有效, app_mac_phy_pclkr eq_n_en 信号 使能。
27	app_mac_phy_pclkr eq_n	RW	低有效, 软件控制 app_mac_phy_pclkr eq_n 信号,该信号在 app_mac_phy_pclkr eq_n_en 高有效, 才能生效。
26	app_mac_phy_rxelec idle_disabl e_en	RW	高有效, app_mac_phy_rxelec idle_disabl e_en 信号 使能。
25	app_mac_phy_rxelec idle_disabl e	RW	低有效, 软件控制 app_mac_phy_rxelec idle_disabl e 信号,该信 号在 app_mac_phy_rxelec idle_disabl e_en 高有效, 才能生效。
24	app_mac_phy_txco mmonmo de_disabl e_en	RW	高有效, app_mac_phy_txco mmonmo de_disabl e_en 信号使能。
23	app_mac_phy_txco mmonmo de_disabl e	RW	低有效, 软件控制 app_mac_phy_txco mmonmo de_disabl e 信号,该信号在 app_mac_phy_txco mmonmo de_disabl e_en 高有效, 才能生效。
22	app_local_ref_clk_re q_n_en	RW	高有效, app_local_ref_clk_re q_n_en 信号 使能。
21	app_local_ref_clk_re q_n	RW	低有效, 软件控制 app_local_ref_clk_re q_n 信号,该信号在 app_local_ref_clk_re q_n_en 高有效, 才能生效。



Bits	Name	Memory Access	Description
20	app_clk_req_in_n_en	RW	高有效, clk_req_in 信号使能。
19	app_clk_req_in_n	RW	低有效, 软件控制 clk_req_in 信号,该信号在 app_clk_req_in_n_en 高有效, 才能生效。
18	phy_mac_pclkack_n	RW	低有效, 低功耗时钟响应确认。
17	app_clk_pm_en	RW	高有效, 软件控制 PM 使能。
16	pcie_phy_cfg_vreg_bypass	RW	cfg_vreg_bypass 1 有效。
15	pcie_phy_cfg_mac_clkreq	RW	phy_cfg_mac_clkreq 0 有效。
14:10	reserved_38_1	RW	保留。
9:7	phy_rx0_eq	RW	Rx 均衡使能。
6	phy_test_powerdown	RW	环路电源控制, 在测试状态下, 关闭所有环路电源。
5	phy_test_bypass	RW	PHY 测试 bypass 信号。
4	phy_test_burnin	RW	PHY test 信号。
3	reserved_38_0	RW	保留。



Bits	Name	Memory Access	Description
2	phy_rtune_req	RW	PHY 参考电阻反转请求。
1	pciephy_cfg_ssc_en	RW	Serdes ssc 使能，高有效。
0	phy_res_req_in	RW	公用外部参考电阻应答请求。

寄存器 PCIE_PHYCTL2 偏移地址：0x38，寄存器位数：32bit



40 ADC

40.1 概述

ADC (Analog -to- Digital Converter) 模拟数字转换器。

ADC 模块在系统中用于将芯片外部输出的模拟信号，如电压值，进行自动采集转换为数据信号，供内部 CPU 进行读取。

40.2 特性

- 支持两个 ADC，各自动收集 8 路模拟输入，一共 16 路。
- 支持根据水线上报告警。
- 支持采集数据 10bit 和 12bit 编码输出。
- 支持 5MHz Conversion Rate。
- 复位信号为高有效。
- 支持采集数据异常上报。
- 支持软件配置 ADC 模块工作时钟为可配置。
- Digital Power Supply: 1.0V±10%。



40.3 关键术语

表40-1 ADC 关键术语描述

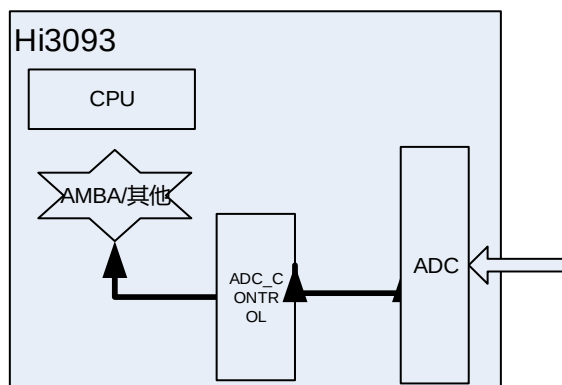
术语	描述
ADC	Analog -to- Digital Converter，模拟数字转换器。

40.4 功能

ADC 如图 40-1 所示。将外部输入给芯片的模拟信号值，如电压，转换为数字值，供内部 CPU 进行读取。

ADC 的参考电压支持从芯片外部 PAD（PAD_ADCVREF）输入或使用内部参考电压。

图40-1 ADC 应用图



40.5 工作方式

40.5.1 初始化

在使用 ADC 之前必须要对 ADC 进行一系列的初始化。以 ADC0 为例，使用外部参考电压的初始化流程如下：

- 步骤 1 使能 ADC0，配置 LP 控制器的 LPSC_ADC_CTRL bit[1:0]为 2'b00，即
adc_selref=0,adc_selbg=0，配置 LPSC_ADC_CTRL bit[4]为 1，即 adc_enadc=1。



须知

必须在使能 `adc_enadc` 前通过配置 LP 控制器的 `LPSC_ADC_CTRL` bit[3:2]为 2'b10 或 2'b11 来配置 ADC0 的分辨率为 12bit 或 10bit，该寄存器默认值不能工作。

步骤 2 复位 ADC0，配置 LP 控制器的 `LPSC_SRST_REQ1` bit[1]为 1。

步骤 3 复位撤离，配置 LP 控制器的 `LPSC_RESET_DREQ1` bit[1]为 1。

----结束

使用内部参考电压的初始化流程如下：

步骤 1 使能 ADC，配置 LP 控制器的 `LPSC_ADC_CTRL` bit[1:0]为 2'b11，即 `adc_selref=1`，`adc_selbg=1`，配置 `LPSC_ADC_CTRL` bit[4]为 1，即 `adc_enadc=1`

须知

必须在使能 `adc_enadc` 前通过配置 LP 控制器的 `LPSC_ADC_CTRL` bit[3:2]为 2'b10 或 2'b11 来配置 ADC0 的分辨率为 12bit 或 10bit，该寄存器默认值不能工作。

步骤 2 等待 2US，等待内部电压稳定。

步骤 3 复位 ADC0，配置 LP 控制器的 `LPSC_SRST_REQ1` bit[1]为 1。

步骤 4 复位撤离，配置 LP 控制器的 `LPSC_RESET_DREQ1` bit[1]为 1。

----结束

须知

如果需要对 ADC 的工作模式（参考电压，分辨率等）进行修改，需按照流程重新初始化。

40.5.2 软复位

软复位会对 ADC 和 ADC CONTROL 同时进行复位。步骤请参考 [40.5.1 初始化](#) 章节。



40.5.3 工作模式

40.5.3.1 ADC0 单独工作

步骤 1 初始化 ADC0。

步骤 2 配置扫描周期（地址 0X08769050）。

步骤 3 配置中断使能寄存器（地址 0X08769008）。

步骤 4 配置 ADC_CONCTRL 模块开始工作（地址 0x08769000）。

步骤 5 读数据寄存器 SAMPLE_DAT0~ SAMPLE_DAT3（地址 0x08769010~0x0876901C）
得到 8 个通道的采样值。

----结束

须知

可以设置 LLB_CH0~LLB_CH7（地址 0x08769020~0x0876903C）来单独设置每个通道的低阈值和高阈值，从而实现数据的异常上报功能。

40.5.3.2 ADC0 和 ADC1 同时工作

步骤 1 初始化 ADC0，ADC1，请将初始化参数设置成相同参数。

步骤 2 配置 ADC0 和 ADC1 控制器扫描周期（地址 0x08769050 和地址 0x0876A050），
设置为相同参数。

步骤 3 配置中断使能寄存器（地址 0x08769008 和地址 0x0876A008）。

步骤 4 配置 ADC1_CONCTRL 模块开始工作（地址 0x0876A000）。

步骤 5 配置 ADC0_CONCTRL 模块开始工作（地址 0x08769000）。

步骤 6 读数据寄存器 SAMPLE_DAT0~ SAMPLE_DAT3（地址 0x08769010~0x0876901C 和
地址 0x0876A010~0x0876A01C）得到 16 个通道的采样值。

----结束



须知

可以设置 LLB_CH0~LLB_CH7（地址 0x08769020~0x0876903C 和地址 0x0876A020~0x0876A03C）来单独设置每个通道的低阈值和高阈值，从而实现数据的异常上报功能。

须知

ADC1 无法单独工作。

40.5.4 时钟频率配置

可以通过配置寄存器 LPSC_CRG_DIV_CTRL 的 bit[28:24]来调整 ADC 的频率，默认为 50M。

须知

配置频率的时候请关闭 ADC，待配置完成后重新使能。

两个 ADC 频率受同一分频系数控制。



41 EFUSE

41.1 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：

- 参数化 EFUSE：用于调整、校准、使能/去使能片上的功能。
- 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；用于满足产品专用的安全需求。
- 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。

Hi3093 采用了 1 块的 EFUSE 模块，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。

41.2 特性

- 提供 256bit 的 UDS 作为根密钥，装备时烧写，限制只能烧写一次，且复位后仅启动安全核时 BootROM 能访问，限定访问一次且提供锁定（提供永久关闭调试时读取 UDS 的功能）。
- 按照公司安全启动流程提供相应 EFUSE 字段。
- 提供 32bit 用于启动配置，装备时烧写。
- 提供 128bit 用于固件版本吊销。
- 系统支持安全启动和非安全启动方式，启动方式由 EFUSE 指示。
- 支持 Die-ID 可查询。
- 支持 EFUSE 一致的读写控制逻辑。



- 支持 SOC_JTAG、A55 和安全 M3 读写；支持 CER 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。

须知

EFUSE 作为电熔丝逻辑，在供电、读和烧写需要注意：

- 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。
- 烧写后的 EFUSE 读取值为 “1”，未烧写的 EFUSE 为 “0”。
- 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。
- 所有字段烧写后，都需要复位 EFUSE 才可以生效。该复位包括上电复位、单独软复位 EFUSE 等各种能引起 EFUSE 复位的复位源。



41.3 功能

41.3.1 应用框图

图41-1 EFUSE 应用图

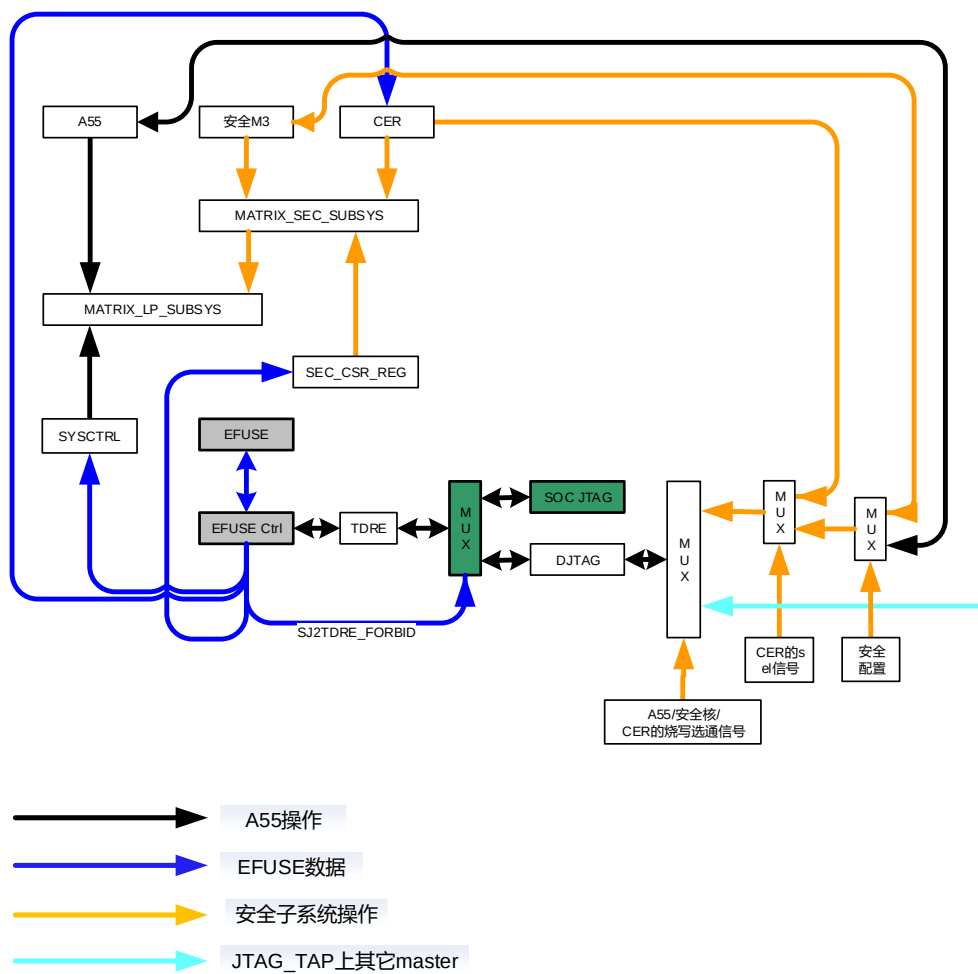
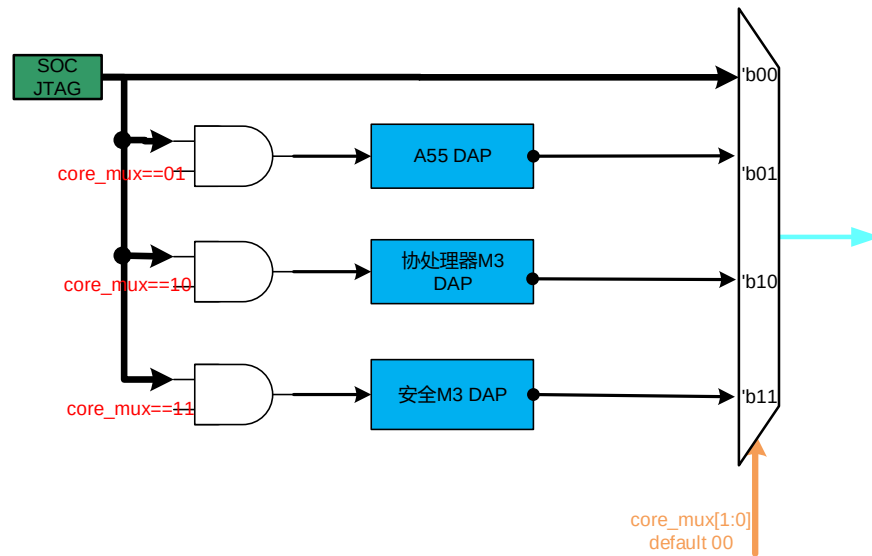


图41-2 JTAG_TAP 上其他 master



如图 41-1 到 EFUSE 的通路有 5 条，分别是 SOC JTAG、A55、安全 M3、CER 和 JTAG_TAP 上其他 master。如图 41-2，JTAG_TAP 上其他 master 是 SOC JTAG、A55 DAP、协处理器 M3 DAP 和安全 M3 DAP，4 个 master 经过选通到 DJTAG TAP。

访问权限设置如下：

- SOC JTAG 在装备阶段后不能访问 efuse，通过 efuse 中 SJ2TDRE_FORBID 禁掉。
- A55 的 djtag_mstr 在装备阶段后除了不能读 UDS，其它都可读，但 efuse 所有字段都不可写。这是通过 efuse 中 KEY_CFGED 在 djtag_mstr 中设置禁止访问读写地址实现的。
- A55 也可以通过 SYSCTRL 读出 efuse 中的信息，不能读安全计算使用的信息。
- 安全 M3 的 djtag_mstr 在装备阶段后可读所有信息，但对 UDS，如果禁止读 UDS 的寄存器有效后，读出来的是无效值（该寄存器是 SEC_CSR_REG 的 UDS_RD_ONCE_SET，需要安全 M3 配置，Hi3093 复位解除状态）；在装备阶段后，可写除 Revoked Subkey ID Mask 的现网字段，其他都不可写。和上面 A55 的 djtag_mstr 一样，是通过 efuse 中 KEY_CFGED 在 djtag_mstr 中设置禁止访问读写地址实现的。
- 安全 M3 可以通过 SEC_CSR_REG 读出安全计算使用的信息；对于 UDS，如果禁止读 UDS 的寄存器有效后，读出来的是无效值（该寄存器是 SEC_CSR_REG



的 UDS_RD_ONCE_SET, 需要安全 M3 配置, Hi3093 复位解除状态)。可以 SYSCTRL 读出其他剩余信息。

- CER 仅能烧写 Revoked Subkey ID Mask。
- JTAG_TAP 上其他 master 不能访问 EFUSE, 在青色箭头进来的 MUX 上用 A55/安全 M3/CER 访问 EFUSE 的 sel 信号选通, 当 A55/安全 M3/CER 访问时可以选通, 当 JTAG_TAP 上其他 master 时不通。

对于 A55、安全 M3 和 CER 过来的信号, A55 和安全 M3 是通过配置 SEC_CSR_REG 中的 A55_M3_DJTAG_MSTR_SEL 选通的。选通结果再与 CER 做选通, 选通信号是 CER 访问 efuse 的 sel, 表示 CER 有访问 efuse 的优先权。

须知

对于安全 M3/A55 烧写进去的值, 需要复位 EFUSE 后才能从 djtag_mstr 读到; 检验是否烧写成功, 不要从寄存器中读 EFUSE 信息。安全 M3 通过配置 SEC_CSR_REG 中的 efuse_reset_req 为 0x1 来实现软复位 efuse; A55 通过配置 LP_SUBSYS 中的 efuse_srst_req 为 0x1 来实现软复位 efuse。

41.3.2 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, SOC_JTAG 和 CPU 通过 TDRE 完成基本的 EFUSE 读或烧写操作。
- A55: 可通过 a55_djtag_mstr 烧写 efuse; 可通过 a55_djtag_mstr 和 SYSCTRL 读 efuse。
- 安全 M3: 可通过 sec_m3_djtag_mstr 烧写 efuse; 可通过 sec_m3_djtag_mstr、SYSCTRL 和 SEC_CSR_REG 读 efuse。
- CER: CER 会更新二级密钥吊销信息。
- SEC_CSR_REG: 安全数据通过该模块供安全 M3 读。
- SYSCTRL: DIE ID 之类的数据通过该模块供安全 M3 和 A55 读。



41.4 工作方式

41.4.1 初始化

EFUSE 的初始化流程参考 Hi3093 初始化流程描述。正常的 EFUSE 初始化流程不需要软件参与。

41.4.2 EFUSE 烧写

41.4.2.1 烧写方式

烧写有 SOC JTAG 和 CPU DJTAG 两种方式。

41.4.2.2 烧写时间

烧写 n bit 用的时间为: $16672 + 306 * n(\text{cycle})$ 。

41.4.2.3 烧写基本流程

EFUSE 烧写方式为 CPU DJTAG。

CER 对 EFUSE 的烧写也是经过 DJTAG 进行，只是 CER 直接输出了 DTAG 信号。由于 CER 只是对 Revoked Subkey ID Mask 进行烧写，对其不做介绍；SOC JTAG 流程请参考 Hi3093 DFT EFUSE 烧写流程描述。下面介绍安全 M3 烧写 EFUSE 流程；A55 的烧写流程也是一样的，只是 DJTAG Master 对应的基地址不一样。

CPU DJTAG 通过安全 M3 对 DJTAG Master 操作，控制 DJTAG TAP 控制器访问 BISR 的 TDRE 接口，完成烧写 EFUSE。

烧写前需要通过管脚打开 efuse 烧写电压，烧写完再关闭。

欲烧写的位置为 EFUSE[617:612]，烧写值为 0x3F，对应 SEC_CSR_REG 寄存器为 ROOTKEYHASH_VALUE6，SEC_CSR_REG 基地址 0x08533000。下面是对 SEC_SUBSYS_DJTAG_SUBCTRL 的寄存器进行操作。

- 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。
- 步骤 2 CPU 配置寄存器 SC_DJTAG_MSTR_CFG（偏移地址 0xD818）为 0x203F0000，选中 BISR 的 TDRE 链，并配置写使能。



- 步骤 3 CPU 配置寄存器 SC_DJTAG_MSTR_ADDR (偏移地址 0xD810) 为 0x8019, 选中偏移地址为 0x8019 (即 [639:608]) 。
- 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[639:608]==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 DJTAG Master 寄存器 SC_DJTAG_OP_ST (偏移地址 0xE828) , 如果域段 djtag_op_done 读出值为 0x1, 继续; 否则等待, 直到读出值为 0x1。
- 步骤 7 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 8 CPU 配置寄存器 SC_DJTAG_MSTR_CFG (偏移地址 0xD818) 为 0x203F0000, 选中 BISR 的 TDRE 链, 并配置写使能。
- 步骤 9 CPU 配置寄存器 SC_DJTAG_MSTR_ADDR (偏移地址 0xD810) 为 0x7, 选中偏移地址为 0x7。
- 步骤 10 CPU 配置寄存器 SC_DJTAG_MSTR_DATA (偏移地址 0xD814) 为 0x1, 配置数据为 0x1。
- 步骤 11 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 12 CPU 读 DJTAG Master 寄存器 SC_DJTAG_OP_ST (偏移地址 0xE828) , 如果域段 djtag_op_done 读出值为 0x1, 继续; 否则等待, 直到读出值为 0x1。
- 步骤 13 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 14 CPU 配置寄存器 SC_DJTAG_MSTR_CFG (偏移地址 0xD818) 为 0x203F0000, 选中 BISR 的 TDRE 链, 并配置写使能。
- 步骤 15 CPU 配置寄存器 SC_DJTAG_MSTR_ADDR (偏移地址 0xD810) 为 0x0, 选中偏移地址为 0x0。
- 步骤 16 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 17 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 18 CPU 读 DJTAG Master 寄存器 SC_DJTAG_OP_ST (偏移地址 0xE828) , 如果域段 djtag_op_done 读出值为 0x1, 继续; 否则等待, 直到读出值为 0x1。
- 步骤 19 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 20 CPU 配置寄存器 SC_DJTAG_MSTR_CFG (偏移地址 0xD818) 为 0x203F0000, 选中 BISR 的 TDRE 链, 并配置写使能。
- 步骤 21 CPU 配置寄存器 SC_DJTAG_MSTR_ADDR (偏移地址 0xD810) 为 0x0, 选中偏移地址为 0x0。
- 步骤 22 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 23 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 24 CPU 读 DJTAG Master 寄存器 SC_DJTAG_OP_ST (偏移地址 0xE828) , 如果域段 djtag_op_done 读出值为 0x1, 继续; 否则等待, 直到读出值为 0x1。

----结束

41.4.2.4 补充流程 1

在线更新 EFUSE 额外流程如下, 该流程必须在“烧写完成后”和“软复位前”之间做:

- 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。
- 步骤 2 CPU 配置寄存器 SC_DJTAG_MSTR_CFG (偏移地址 0xD818) 为 0x203F0000, 选中 BISR 的 TDRE 链, 并配置写使能。
- 步骤 3 CPU 配置寄存器 SC_DJTAG_MSTR_ADDR (偏移地址 0xD810) 为 0x0。
- 步骤 4 CPU 配置寄存器 SC_DJTAG_MSTR_DATA (偏移地址 0xD814) 为 0x800。
- 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 DJTAG Master 寄存器 SC_DJTAG_OP_ST (偏移地址 0xE828) , 如果域段 djtag_op_done 读出值为 0x1, 继续; 否则等待, 直到读出值为 0x1。

----结束



41.4.2.5 补充流程 2（烧写是否成功）

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 1 CPU 配置 DJTAG Master 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 2 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_CFG（偏移地址 0xD818）为 0x203F0000，选中 BISR_S 的 TDRE 链，并配置写使能。
- 步骤 3 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_ADDR（偏移地址 0xD810）为 0x0。
- 步骤 4 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_DATA（偏移地址 0xD814）为 0x800。
- 步骤 5 CPU 配置 DJTAG Master 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 DJTAG Master 寄存器 SC_DJTAG_OP_ST（偏移地址 0xE828），如果域段 djtag_op_done 读出值为 0x1，继续；否则等待，直到读出值为 0x1。
- 步骤 7 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_CFG（偏移地址 0xD818）为 0x00200000，选中 HP 的 TDRE 链。
- 步骤 8 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_START_EN（偏移地址 0xD81C）为 0x1，启动 DJTAG Master 访问。
- 步骤 9 CPU 配置 SEC_CSR_REG 中的寄存器 SEC_RESET_REQ 的 bit13 为 0x1。
- 步骤 10 CPU 读取 SEC_CSR_REG 中的寄存器 SC_BISR_RESET_ST 的 bit13，若值为 0x1，表示 BISR_S 处于复位状态。
- 步骤 11 CPU 配置 SEC_CSR_REG 中的寄存器 SEC_RESET_DREQ 的 bit13 为 0x1。
- 步骤 12 CPU 读取 SEC_CSR_REG 中的寄存器 SEC_RESET_ST，若值为 0x0，表示 BISR_S 已解除复位状态。
- 步骤 13 CPU 读取 SEC_CSR_REG 中的寄存器 EFUSE_INFO_DONE，若为 0x1，表示 BISR-S 初始化完成。
- 步骤 14 CPU 读取 SEC_CSR_REG 中的寄存器 REVOKED_SUBKEY_IDMASK 的值，跟预期值比对。

----结束



41.4.2.6 补充流程 3（是否复位）

以下步骤是 SC_DJTAG_MSTR_DATA（偏移地址 0xD814）烧写为 0x846 后，没有进行 chip 复位的情况下，要再次烧写需要添加的额外流程：

- 步骤 1 CPU 配置 DJTAG Master 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 2 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_CFG（偏移地址 0xD818）为 0x203F0000，选中 BISR_S 的 TDRE 链，并配置写使能。
- 步骤 3 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_ADDR（偏移地址 0xD810）为 0x0。
- 步骤 4 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_DATA（偏移地址 0xD814）为 0x800。
- 步骤 5 CPU 配置 DJTAG Master 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 DJTAG Master 寄存器 SC_DJTAG_OP_ST（偏移地址 0xE828），如果域段 djtag_op_done 读出值为 0x1，继续；否则等待，直到读出值为 0x1。

----结束

须知

41.4.2.3 步骤 1~41.4.2.3 步骤 12 是完成 EFUSE 待烧写数据配置，通过重复这个过程，可以完成 EFUSE 中多个字段的配置（可以把多个字段的值都经过 41.4.2.3 步骤 1~41.4.2.3 步骤 12 写完，再进行后面的步骤烧写）。41.4.2.3 步骤 13~41.4.2.3 步骤 24 是完成烧写操作下发。如果缺少了补充流程 1，在线烧写 EFUSE 后，将无法通过软复位，读出 EFUSE 的存储值。

41.4.2.7 补充流程 4（EFUSE 单独软复位需要加的步骤）

EFUSE 单独软复位前和安全 M3/A55/CER 读写完 EFUSE 之后都需要把 TDRE 链切到 HP 上，如下步骤。EFUSE 单独软复位有三个源：安全 M3 通过配置 SEC_CSR_REG 中的 efuse_reset_req 为 0x1；A55 通过配置 LP_SUBSYS 中的 efuse_srst_req 为 0x1；CER 做二级密钥更新。

- 步骤 1 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_CFG（偏移地址 0xD818）为 0x00200000，选中 HP 的 TDRE 链。



步骤 2 CPU 配置 DJTAG Master 寄存器 SC_DJTAG_MSTR_START_EN (偏移地址 0xD81C) 为 0x1, 启动 DJTAG Master 访问。

---结束

41.4.3 通过 DJTAG_MSTR 读 efuse

欲读的位置为 EFUSE[639:608]。

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。

步骤 2 CPU 配置寄存器 SC_DJTAG_MSTR_CFG (偏移地址 0xD818) 为 0x003F0000, 选中 BISR 的 TDRE 链, 并配置写使能。

步骤 3 CPU 配置寄存器 SC_DJTAG_MSTR_ADDR (偏移地址 0xD810) 为 0xC019, 选中偏移地址为 0xC019 (即 [639:608]) 。

步骤 4 CPU 配置寄存器 SC_DJTAG_MSTR_DATA (偏移地址 0xD814) 为 0x0。

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 DJTAG Master 寄存器 SC_DJTAG_OP_ST (偏移地址 0xE828) , 如果域段 djtag_op_done 读出值为 0x1, 继续; 否者等待, 直到读出值为 0x1。

---结束

41.4.4 在线更新 EFUSE 中的 RIM 值

步骤和注意实现请参见 0 章节。



42 A55 低功耗

42.1 概述

A55 的低功耗实现由 PMCTRL 模块来控制完成，其功耗管理主要包括 CLUSTER 动态门控和 CORE 上下电管理。

42.2 低功耗特性

A55 的低功耗设计主要实现如下特性：

- 实现 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。
- 实现 A55 Core 独立动态上下电功能，4 个 Core 独立上下电控制，Core 的下电流程可由本 Core 或其他 Core 操作完成。
 - 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 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。



42.3 关键术语

表42-1 PMCTRL 关键术语描述

术语	描述
PMCTRL	Power Management Controller，控制器。
PDC	Power Down Controller
FCM	Flexible Cluster Microarchitecture

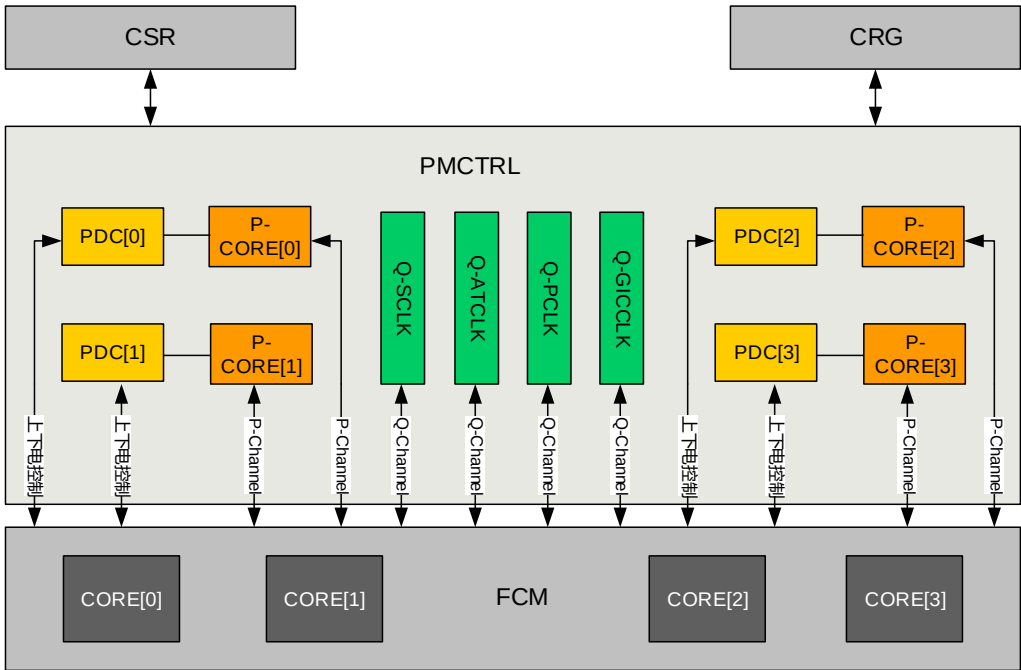
42.4 遵循协议

IHI0068B_low_power_interface_spec

42.5 功能描述

功耗管理模块 PMCTRL 结构框图如图 42-1 所示，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42-1 PMCTRL 结构框图





4 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。

42.6 应用场景

Core 的上下电场景：4 个 A55 Core 默认上电，然后软件根据业务需要，选择将 n 个 Core (n=1,2,3,4) 进行下电，下电后，仍可以重新上电。对多个 Core 进行上下电时，按照流程串行依次上下电。

对于 4 个 A55 Core 全部下电的情况，比较特殊，在重新上电时，必须有 M3 控制上电。

Cluster 时钟动态门控场景：完全由硬件自动控制完成，在特定时钟域 (SCLK/ATCLK/GICCLK/PCLK) 逻辑空闲时，硬件自动将对应时钟关闭。

42.7 工作流程

这里的流程主要包括 Core 硬件下电、Core 软件下电、Core 硬件上电和 Core 软件上电 4 种主要场景，这 4 种场景都是在软件主控下进行的。

其中，Core 硬件下电和 Core 硬件上电流程，是在使能 PMCTRL 情况下，需要 PMCTRL 逻辑参与，在相应的步骤中自动实现与 Core 的握手，以及 MTCMOS，Isolation 开关控制、复位控制等操作。

Core 软件下电和 Core 软件上电流程，是在关闭 PMCTRL 情况下，不需要 PMCTRL 逻辑参与，仅通过芯片提供的寄存器接口，完全有软件来实现与 Core 的握手，以及 MTCMOS，Isolation 开关，复位等的控制。

Core 硬件上电又分为软件发起的上电和中断唤醒的上电两种子场景：

- 软件发起的上电场景，是在 Core 下电状态下，因软件需求，需要主动让 Core 进行的上电流程；
- 中断唤醒的上电场景，是在 Core 下电状态下，因某些中断事务触发，芯片自主进行的上电流程。



42.7.1 Core 硬件下电流程

步骤 1 初始化

- 软件根据所处的安全状态设置 A55 访问权限。
设置 **ACTLR_EL3.PWREN** (BIT7) 为 1, 打开 EL2/EL1 Secure 模式下对低功耗寄存器的访问权限;
设置 **ACTLR_EL2.PWREN** (BIT7) 为 1, 打开 EL1 Non-secure 模式下对低功耗寄存器的访问权限。
- 设置 Core[n]重新上电的启动地址。
配置 A5X 寄存器 **A55_CORE[n]_RVBA**, 指向 Core 重新上电的启动程序。
- 打开 PMCTRL 中的 P-Channel 控制器和上下电控制器。
设置 **PMC_CORE[n]_PCTRL.core[n]_p_channel_enable** (bit[31]) 为 1, 使能 Core[n] P-Channel 控制器;
设置 **PMC_CORE_PDC_CTRL.core[n]_pdc_en** 为 1, 使能 Core[n] PDC 控制器。

步骤 2 中断相关配置。

- 配置 GIC 寄存器 **GICD_ICENABLER**, 关闭以 core[n]为 target 的中断;
- 配置 GIC 寄存器 **GICD_ISENABLER**, 使能用于唤醒 core[n]的中断;
- 配置 GIC, 设置 **GICR[n]**的寄存器 **GICR_WAKER.ProcessorSleep** 为 1, 指示在有需要 core[n]处理的中断时 GIC 会通过 wake_request 唤醒该 Core。

步骤 3 开启 A55 Core 可下电使能。

- 配置 core[n], 设置 **CPUPWRCTLR_EL1.CORE_PWRDN_EN** (BIT0) 为 1, 指示在 Core 进入 WFI 模式后会向 PMCTRL 请求下电。

步骤 4 软件发起下电准备。

- 配置 PMCTRL, 设置 **PMC_CORE_POWER_DOWN_REQ.core[n]_pwrnd_req** 为 1, 启动下电状态机进入等待下电状态。

步骤 5 软件执行指令, core[n]进入 WFI。

- 执行 ISB 指令;
- 执行 DSB SY 指令;
- 执行 WFI 指令。



步骤 6 硬件自动完成 P-Channel 握手，执行一系列下电动作。

- Core 通过 P-Channel 与 PMCTRL 完成握手；
- 使能 Isolation；
- 复位有效；
- 关闭 MTCMOS，Core 区下电。

----结束

须知

Core 硬件下电完成后，PMCTRL 会同时发出完成中断给 M3 和 GIC，同时会上报状态给 SUBCTRL 供软件查询。

42.7.2 Core 软件下电流程

步骤 1 初始化。

- 软件根据所处的安全状态设置 A55 访问权限
设置 **ACTLR_EL3.PWREN** (bit[7]) 为 1，打开 EL2/EL1 Secure 模式下对低功耗寄存器的访问；
设置 **ACTLR_EL2.PWREN** (bit[7]) 为 1，打开 EL1 Non-secure 模式下对低功耗寄存器的访问。
- 设置 core[n]重新上电的启动地址
配置 A5X 寄存器 **A55_CORE[n]_RVBA**，指向 Core 重新上电的启动程序。

步骤 2 中断相关配置。

- 配置 GIC 寄存器 **GICD_ICENABLER**，关闭以 core[n]为 target 的中断；
- 配置 GIC 寄存器 **GICD_ISENABLER**，使能用于唤醒 core[n]的中断；
- 配置 GIC，设置 **GICR[n]**的寄存器 **GICR_WAKER.ProcessorSleep** 为 1，指示在有需要 core[n]处理的中断时 GIC 会通过 wake_request 唤醒该 Core。

步骤 3 开启 A55 Core 可下电使能。

- 配置 core[n]，设置 **CPUPWRCTLR_EL1.CORE_PWRDN_EN** (bit[0]) 为 1，指示在 Core 进入 WFI 模式后会向 PMCTRL 请求下电。



步骤 4 软件执行指令，core[n]进入 WFI。

- 执行 ISB 指令；
- 执行 DSB SY 指令；
- 执行 WFI 指令。

步骤 5 软件 P-Channel 握手。

- 软件查询 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PSTAT_FEEDBACK 寄存器 bit [17:0]，直到 PACTIVE 值变为 0；
- 软件配置 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PCTRL 的 bit [21:16]为 0；
- 软件配置 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PCTRL 的 bit [24]为 1，发起 P-Channel 下电请求；
- 软件查询 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PSTAT_FEEDBACK 的 bit [20]，直至寄存器值为 1，指示 core[n]接受请求；
- 软件配置 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_P_CTRL 的 bit [24]为 0，拉低 PREQ；
- 软件查询 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PSTAT_FEEDBACK 的 bit [20]，直至寄存器值为 0，表示 P-Channel 握手完成。

步骤 6 软件操作 Core 下电。

- 配置 Cluster SUBCTRL 寄存器 SC_CLUS_CRG_RESET_REQ bit[7:0]对应 bit 为高，将对应核复位 ([3:0]对应核上电复位，[7:4]对应核非上电复位，软件根据需要配置特定核复位)；
- 配置 Cluster SUBCTRL 寄存器 PMC_CORE_PDC_CTRL bit[11:8]对应 bit 为高，打开 Core Isolation 隔离（每 bit 对应一个 Core）；
- 配置 Cluster SUBCTRL 寄存器 PMC_CORE_PDC_CTRL bit[15:12]对应 bit 为低，关闭 Core MTCMOS 关闭 Core 电源（每 bit 对应一个 Core）。

----结束

42.7.3 Core 硬件上电流程-软件发起

步骤 1 软件发起上电准备。



- 设置 PMC_CORE_POWER_UP_REQ.core[n]_pwrup_req 为 1，启动上电状态机。

步骤 2 硬件自动执行一系列上电动作，Core 重新启动。

- 打开 MTCMOS，Core 区上电；
- 关闭 Isolation；
- 解复位；
- PMCTRL 主动发起 P-Channel 握手，与 Core 完成状态同步。

步骤 3 在启动程序中完成状态恢复。

- 设置 GICR_WAKER.ProcessorSleep 为 0，指示在有中断 target core[n]时 GIC 不会再 assert wake_request；
- 将之前下电过程中对中断的修改恢复。

----结束

须知

Core 硬件上电完成后，PMCTRL 会同时发出完成中断给 M3 和 GIC，同时会上报状态给 SUBCTRL 供软件查询。

42.7.4 Core 硬件上电流程-中断唤醒

步骤 1 GIC 收到唤醒中断，通知 PMCTRL。

- PMCTRL 收到 GIC 的唤醒通知，启动上电状态机。

步骤 2 硬件自动执行一系列上电动作，Core 重新启动。

- 打开 MTCMOS，Core 区上电；
- 关闭 Isolation；
- 解复位；
- PMCTRL 主动发起 P-Channel 握手，与 Core 完成状态同步。

步骤 3 在启动程序中完成状态恢复。

- 设置 GICR_WAKER.ProcessorSleep 为 0，指示在有中断 target core[n]时 GIC 不会再 assert wake_request；



- 将之前下电过程中对中断的修改恢复。

----结束

须知

Core 硬件上电完成后，PMCTRL 会同时发出完成中断给 M3 和 GIC，同时会上报状态给 SUBCTRL 供软件查询。

42.7.5 Core 软件上电流程

步骤 1 设置 core[n]重新上电的启动地址。

- 配置 A5X 寄存器 A55_CORE[n]_RVBA，指向 Core 重新上电的启动程序。

步骤 2 关闭 PMCTRL。

- 关闭 PMCTRL 中核[n] 对应的 P-Channel 控制器。

设置 **PMC_CORE[n]_PCTRL**.core[n]_p_channel_enable (BIT31) 为 0，关闭 Core[n] P-Channel 控制器。

- 关闭 PMCTRL 中核[n]对应的上下电控制器。

设置 **PMC_CORE_PDC_CTRL**.core[n]_pdc_en 为 0，关闭 Core[n] PDC 控制器。

步骤 3 初始化 P-Channel 握手。

- 软件配置 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PCTRL 的 bit [21:16]为 0x8；
- 软件配置 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PCTRL 的 bit [24]为 1，发起 P-Channel 上电请求。

步骤 4 软件操作一系列 Core 上电动作，Core 开始启动。

- 配置 Cluster SUBCTRL 寄存器 PMC_CORE_PDC_CTRL bit[15:12]对应 bit 为 1，打开 Core MTCMOS 打开 Core 电源（每 bit 对应一个 Core）；
- 配置 Cluster SUBCTRL 寄存器 PMC_CORE_PDC_CTRL bit[11:8]对应 bit 为 0，关闭 Core Isolation 隔离（每 bit 对应一个 Core）；



- 配置 Cluster SUBCTRL 寄存器 SC_CLUS_CRG_RESET_DREQ bit[7:0]对应 bit 为高，撤离核复位 (bit[3:0]对应核上电复位，bit[7:4]对应核非上电复位，软件根据需要配置特定核复位)。

步骤 5 完成 P-Channel 握手。

- 软件查询 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PSTAT_FEEDBACK 的 bit [20]，直到寄存器值为 1；
- 软件配置 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PCTRL 的 bit [24]为 0，拉低 PREQ；
- 软件查询 CLUSTER SUBCTRL 寄存器 PMC_CORE[n]_PSTAT_FEEDBACK 的 bit [20]，如果寄存器值为 0，表示 P-Channel 握手完成。

步骤 6 在启动程序中完成状态恢复。

- 设置 **GICR_WAKER.ProcessorSleep** 为 0，指示在有中断 target core[n]时 GIC 不会再 assert wake_request；
- 将之前下电过程中对中断的修改恢复。

----结束

42.7.6 Q-Channel 配置

如果软件需要是能 Q-Channel 动态门控，可以在 Cluster 上电之后配置 Cluster 的 Q-Channel 使能，具体配置：

配置 CLUSTER SUBCTRL 的寄存器 PMC_QCTRL_EN 对应 bit 为 1 (bit[0]对应 SCLK；bit[1]对应 ATCLK；bit[2]对应 PCLK；bit[3]对应 GICCLK)。



43 JTAG

43.1 概述

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

按照功能的不同共包含有 3 类 JTAG TAP 控制器：

- 第一类是 ARM TAP (Test Access Port) 控制器，主要功能是连接 ARM 仿真器用于 Cortex Coresight 调试系统；包括 A55 和 M3 的 TAP 控制器；
- 第二类是 DJTAG TAP 控制器，主要功能是连接可编程仿真器用于寄存器访问和部分寄存器初始化配置；
- 第三类是 DFT (Design For Test) TAP 控制器，主要功能用于边界扫描。

43.2 特性

- 板级提供标准的 JTAG 接口，每个 DIE 上封装一组标准 JTAG 接口作为 Slave 调试口，兼容 IEEE1149.1 JTAG 标准。
- JTAG 接口支持外接调试工具，用于程序调试和单板制造测试。
- JTAG 接口支持工作在 25MHz 及以下的速率。
- 基于 ARM V8 调试系统的 CoreSight，提供多核调试能力。
- 支持独立的调试总线 Debug Bus，提供总线异常情况下的问题定位手段。
- 支持 ATE (Automatic Test Equipment) 测试、样片调试场景统一的 JTAG 接口。
- 支持管脚和内部 JTAG 寄存器不同配置，选择内部 DFT JTAG、ARM JTAG 和 DJTAG 在 JTAG 的通路上的有无。



- 支持安全调试。
- 支持 IEEE1149.1 JTAG 标准；支持 IEEE1149.6 JTAG 标准。

43.3 关键术语

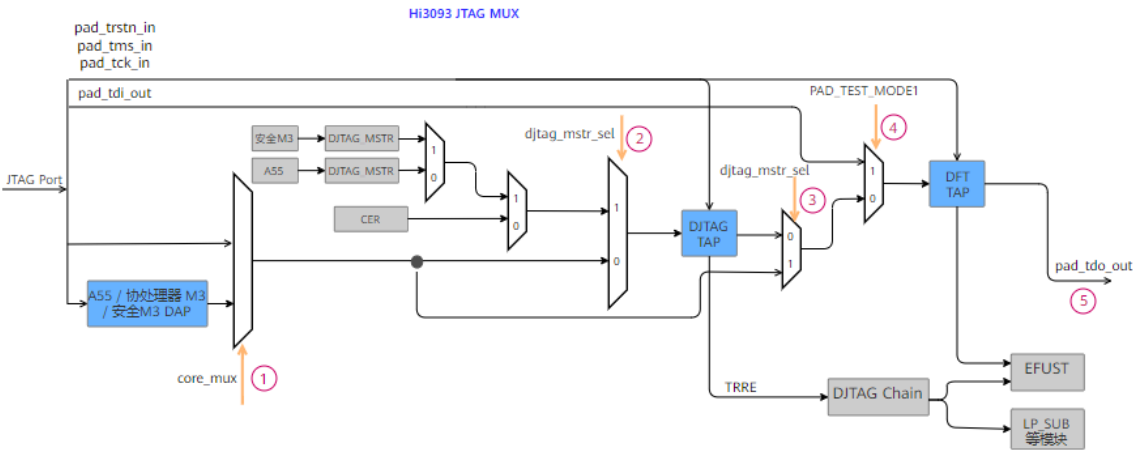
表43-1 JTAG 关键术语描述

术语	描述
JTAG	Joint Test Action Group, JTAG 是联合测试工作组（Joint Test Action Group）的简称，是在名为标准测试访问端口和边界扫描结构的 IEEE 的标准 1149.1 的常用名称。
DJTAG	Debug JTAG, 符合 JTAG 标准的调试模块。
TAP	Test Access Port, 测试接入端口，通过 TAP 可以访问芯片提供的所有数据寄存器和指令寄存器。TAP 总共包括 5 个信号接口 TCK, TMS, TDI, TDO 和 TRST。

43.4 功能

Hi3093 的 JTAG 拓扑结构包含有 5 个 JTAG TAP 控制器。5 个 TAP 控制器的连接结构图如图 43-1 所示，通过芯片的 JTAG 管脚可以连接仿真器调试。

图43-1 Hi3093 JTAG 拓扑结构图





各个 TAP 控制器的描述如下：

- ARM TAP 控制器 (A55 TAP)：指用于 A55 的 coresight 调试系统的 TAP 控制器，支持系统级的实时调试；
- 协处理器 M3 TAP：指用于协处理器 M3 的调试系统的 TAP 控制器；
- 安全 M3 TAP：指用于安全 M3 的调试系统的 TAP 控制器；
- DJTAG TAP 控制器：指连接可编程仿真器用于寄存器访问的 TAP 控制器，当 DJTAG TAP 控制器在菊花链上未被使能时，其可被 CPU 访问，用于芯片内部 Memory 的 MBIST 测试，当 DJTAG TAP 控制器在菊花链上被使能时，外部仿真器可通过 DJTAG TAP 访问芯片内部寄存器；
- DFT TAP 控制器：指用于边界扫描的 TAP 控制器。

43.5 工作方式

43.5.1 JTAG 拓扑配置

5 个 TAP 以首尾菊花链的形式连接，连接形式通过管脚 TEST_MODE、JTAG 配置信号 core_mux、djtag_onchain 的组合配置 JTAG chain。JTAG 的 chain 链和对应 TEST_MODE1、TEST_MODE0 管脚，以及 dft 配置信号的关系如表 43-2 所示。

对于 core_mux，djtag_onchain，可以有寄存器配置，DFT TAP 产生，Strap 管脚产生这三种方式。

参考表 43-2 的管脚配置，JTAG 典型应用场景如下：

- ATE 测试：
使用配置方式 1 (Only DFT TAP)，DFT JTAG 通过 JTAG 接口配置，且 ARM JTAG 和 DJTAG 的 TAP 控制器处于复位状态。
- 样片测试：
使用配置方式 1 (Only DFT TAP) 或配置方式 4 (only DFT 菊花链)，用于 JTAG 扫描；
使用配置方式 2 (Only ARM TAP)，用于 ARM 仿真器调试。使用 DFT TAP 配置完 mux 的选择信号后，DFT TAP 将不再使用。CPU 可通过 SYSCCTRL 访问 DJTAG 配置和 debug 寄存器；



使用配置方式 7 (ARM TAP、DJTAG TAP、DFT TAP 在线)，用于 ARM 仿真器调试，ARM JTAG、DJTAG 和 DFT JTAG 通过菊花链连接；可通过 JTAG 指令将 DJTAG 和 DFT JTAG bypass 掉，也可通过 ARM 仿真器配置 DJTAG 访问 DJTAG 链上模块寄存器；

使用配置方式 5 (DJTAG TAP 和 DFT TAP 在线)。使用 DFT TAP 配置 mux 的选择信号，使 DJTAG TAP 使用 PAD_TDI 信号，可以通过外部管脚 PAD_TDI 通过 DJTAG TAP 访问 DJTAG 链上模块寄存器。

- 装备测试：

使用配置方式 4 (only DFT 菊花链)，用于 JTAG 扫描。处理器通过 DJTAG 也可以访问 DJTAG 链上模块寄存器或 efuse。

表43-2 JTAG Chain 链配置表

场景编号	TEST_MODE1	TEST_MODE0	core_mux (default 00)	djtag_onchain (default 0)	JTAG 连接模式
1	1	1	*	*	only DFT；TDI-DFT-TDO。
2	0	0	*	*	only A55；TDI-A55-TDO。
3	1	0	*	*	only 安全 M3。
4	0	1	00	0	菊花链连接方式：TDI-DFT-TDO；同时 DJTAG 可以通过 CPU 控制进行工作。
5	0	1	01	0	菊花链连接方式：A55 JTAG 和 DFT JTAG 在菊花链上；同时 CPU 可通过 DJTAG 配置寄存器。
6	0	1	10	0	菊花链连接方式：IO M3 TAP 和 DFT JTAG 在菊花链上；同时 CPU 可通过 DJTAG 配置寄存器。



场景 编号	TEST_ MODE1	TEST_ MODE0	core_mux (default 00)	djtag_o nchain (default 0)	JTAG 连接模式
7	0	1	11	0	菊花链连接方式：SEC M3 TAP 和 DFT JTAG 在菊花链上；同时 CPU 可通过 DJTAG 配置寄存器。
8	0	1	01,10,11	1	菊花链连接方式：A55 或 IO M3 或 SEC M3 TAP、DJTAG 和 DFT JTAG 在菊花链上，TDI-ARM-DJTAG-DFT。
9	0	1	00	1	TDI-DJTAG-DFT-TDO。

43.5.2 SOC JTAG

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

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

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

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



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

更多详细内容请参考 IEEE 1149.6。

43.5.3 ARM JTAG

ARM JTAG 即指 ARM TAP 控制器，Hi3093 的 Coresight 调试系统提供了系统级的实时调试。Coresight 的功能特性主要有：

- 多核调试能力。
- 整个系统的可视化调试能力。
- 成熟的调试工具支持。
- 更高的数据压缩率。
- 工具支持的标准编程模型。

43.5.4 DJTAG TAP

43.5.4.1 DJTAG 概述

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

43.5.4.2 DJTAG 指令

表43-3 DJTAG 指令列表

指令名	Code	JTAG 寄存器
EXTEST	00000000	标准 BSD 指令。
BYPASS	11111111	标准 BSD 指令。
SAMPLE	00000001	标准 BSD 指令。
PRELOAD	00000001	标准 BSD 指令。
CLAMP	00000010	标准 BSD 指令。



指令名	Code	JTAG 寄存器
HIGHZ	00000011	标准 BSD 指令。
IDCODE	00000100	标准 BSD 指令。
DBG_CHAIN0	00100000	DJTAG Debug 链 0
DBG_CHAIN1	00100001	DJTAG Debug 链 1
DBG_CHAIN2	00100010	DJTAG Debug 链 2 LP Subsys 安全模块
DBG_CHAIN3	00100011	DJTAG Debug 链 3 HOST Subsys
DBG_CHAIN4	00100100	DJTAG Debug 链 4 HP Subsys
DBG_CHAIN5	00100101	DJTAG Debug 链 5 A5X Subsys
DBG_CHAIN6	00100110	DJTAG Debug 链 6 SysCtrl
DBG_CHAIN7	00100111	DJTAG Debug 链 7
DBG_CHAIN8	00101000	DJTAG Debug 链 8
DBG_CHAIN9	00101001	DJTAG Debug 链 9
DBG_CHAIN10	00101010	DJTAG Debug 链 10
DBG_CHAIN11	00101011	DJTAG Debug 链 11
DBG_CHAIN12	00101100	DJTAG Debug 链 12
DBG_CHAIN13	00101101	DJTAG Debug 链 13
DBG_CHAIN14	00101110	DJTAG Debug 链 14
DBG_CHAIN15	00101111	DJTAG Debug 链 15 Efuse
DBG_CHAIN16	00110000	DJTAG Debug 链 16



指令名	Code	JTAG 寄存器
DBG_CHAIN17	00110001	DJTAG Debug 链 17 DDR
DBG_CHAIN18	00110010	DJTAG Debug 链 18 LP Subsys 非安全 LP_Subsys 的寄存器
.....		DJTAG Debug 链 N
DBG_CHAIN63	01011111	DJTAG Debug 链 63

须知

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

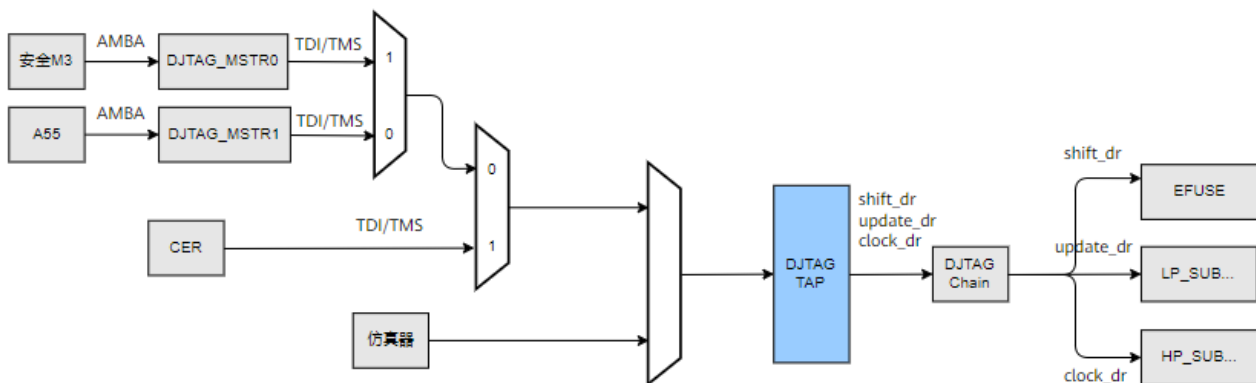
43.5.5 DJTAG Master

43.5.5.1 DJTAG 概述

在 Hi3093 中，集成 2 个 DJTAG Master。DJTAG Master0 为安全 M3 可访问的 DJTAG Master，主要为 Efuse 访问。DJTAG Master1 为 A55 专用 DJTAG Master，当系统挂死时，A55 配置寄存器将 DJTAG TAP 的访问源头从 DJTAG Master0 切换至 DJTAG Master1，因此该配置为静态配置；切换的寄存器为 A55_M3_DJTAG_MSTR_SEL。



图43-2 Hi3093 JTAG 拓扑结构图



DJTAG 的寄存器组（参考表 43-4）需要软件保证原子性操作，在一次 DJTAG 访问结束前，不允许再次发起 DJTAG 访问操作。DJTAG Master0 和 DJTAG Master1 分别有一组寄存器。

表43-4 DJTAG Master 寄存器组

寄存器名	偏移地址	类型
SC_DJTAG_SEC_ACC_EN	0xD800	RW
SC_DJTAG_FAST_MODE	0xD804	RW
SC_DJTAG_MSTR_ADDR	0xD810	RW
SC_DJTAG_MSTR_DATA	0xD814	RW
SC_DJTAG_MSTR_CFG	0xD818	RW
SC_DJTAG_MSTR_START_EN	0xD81C	RW
SC_DJTAG_MSTR_PIPE_CFG	0xD820	RW
SC_DJTAG_TMOUT	0xD840	RW
SC_DJTAG_SEC_MODE	0xD844	RW
SC_DJTAG_RD_DATA0	0xE800	RO
SC_DJTAG_RD_DATA1	0xE804	RO
SC_DJTAG_RD_DATA2	0xE808	RO
SC_DJTAG_RD_DATA3	0xE80C	RO
SC_DJTAG_RD_DATA4	0xE810	RO
SC_DJTAG_RD_DATA5	0xE814	RO
SC_DJTAG_RD_DATA6	0xE818	RO



寄存器名	偏移地址	类型
SC_DJTAG_RD_DATA7	0xE81C	RO
SC_DJTAG_RD_DATA8	0xE820	RO
SC_DJTAG_RD_DATA9	0xE824	RO
SC_DJTAG_OP_ST	0xE828	RO

43.5.5.2 DJTAG Master 访问配置流程

DJTAG Master 访问 Debug bus 链上模块的配置步骤如下：

若本次访问涉及 CER 寄存器，则需要 CPU 配置寄存器 SC_DJTAG_SEC_ACC.djtag_sec_acc_en（偏移地址 0xD800）为 1。若本次访问不涉及 CER 寄存器，则无需配置该寄存器。

- 步骤 1 CPU 配置寄存器 SC_DJTAG_MSTR_ADDR.djtag_mstr_addr（偏移地址 0xD810），写入被访问模块的内部寄存器的偏移地址；（注意：对于 A55 寄存器的访问，配置的地址为 A5X 的基地址(0x14000000) + 内部寄存器的偏移地址）
- 步骤 2 CPU 配置寄存器 SC_DJTAG_MSTR_DATA.djtag_mstr_data（偏移地址 0xD814），写入写数据（如果是读则省略该步骤）。
- 步骤 3 CPU 配置寄存器 SC_DJTAG_MSTR_CFG（偏移地址 0xD818），完成本次 DJTAG 的相关设置。
 - debug_module_sel：写入需要操作的模块对应的地址，模块的地址见表 43-5；
 - djtag_mstr_wr：该次 DJTAG 操作的读写属性，0 为读，1 为写；
 - djtag_mstr_disable：禁止该次 DJTAG 操作，默认为 0，允许本次 DJTAG 操作；
- 步骤 4 CPU 配置寄存器 SC_DJTAG_MSTR_START_EN.djtag_mstr_start_en（偏移地址 0xD81C）为 1，使能 DJTAG 访问。
- 步骤 5 CPU 访问寄存器 SC_DJTAG_OP_ST（偏移地址 0xD828）中的 djtag_op_done，为 1，表示 DJTAG 访问结束。访问寄存器 SC_DJTAG_OP_ST（偏移地址 0xE828）中的 unit_conflict，为 0 表示访问正常，不为 0 表示访问存在异常。读数据通过寄存器 SC_DJTAG_RD_DATA0（偏移地址 0xE800）读取。



须知

当内部 PLL bypass 到参考时钟后，参考时钟为 25M，此时与 PLL 产生的给 DJTAG 交互的时钟为 1.25M，此时 PAD_JTAG_TCK 的时钟应该也需要降至 0.625M 方可以确保正常通信。

----结束

表43-5 DJTAG 链上模块编号

模块名	编号
LP_SUBSYS	0x51
CER	0x2
HPRE	0x03
SEC_CSR	0x04
SEC	0x05
HOST_SUBSYS	0x10
VCE	0x11
HP_SUBSYS	0x20
A5X	0x27
SYSCTRL	0x30
EFUSE	0x3F
DDR	0x50

43.5.6 安全设计

安全设计需要考虑下列有安全需求的点。

- sec_1 表示从 DJTAG Chain 过来访问 efuse，只有安全核配置 djtag_mstr、A55 配置 djtag_mstr 和 CER 过来的命令可以访问，其他命令都禁止。
- sec_2 表示装备阶段后要禁掉 DJTAG Chain 访问安全子系统。
- sec_3 表示装备阶段后要把安全 M3 从链上去掉。
- sec_4 表示装备阶段后把 DFT 从 PAD 直接访问 efuse 的通路禁掉。



- sec_5 表示装备阶段后，安全 M3 对 efuse 中固定信息区域的通路禁掉；A55 到 efuse 和安全子系统的通路禁掉。



44 SRAM/ROM

44.1 概述

SRAM (Static Random Access Memory), 随机存取存储器的一种类型, 只要电源持续供电, 就可以保留其内容。它不像动态随机存取存储器, 不需要不断刷新。

ROM (Read Only Memory) 只读存储器, 用于存放启动相关信息, 掉电信息不丢失。

44.2 特性

Hi3093 SRAM 具有以下特性:

- 支持 AXI 访问。
- 支持两个 SRAM, 分别在安全域和非安全域。
- 支持 SRAM 初始化。
- 支持单比特 ECC 纠错和回写。
- 支持单比特和多比特 ECC 错误上报和错误注入。
- 安全域 SRAM 提供大小为 256KB。
- 非安全域 SRAM 提供大小为 128KB。

Hi3093 ROM 具有以下特性:

- 支持 AXI 访问, 仅支持读。
- 支持安全核访问, 存放安全启动相关代码。
- ROM 大小为 32KB。
- 支持从 ROM 引导启动。



44.3 关键术语

表44-1 SRAM 关键术语描述

术语	描述
SRAM	静态随机存取存储器，随机存取存储器的一种类型，只要电源持续供电，就可以保留其内容。它不像动态随机存取存储器，不需要不断刷新。
ROM	Read Only Memory，只读存储器，用于存放启动相关信息，掉电信息不丢失。

44.4 功能

SRAM 可供内部软件直接读写缓存数据。

ROM 支持安全核访问，存放安全启动相关代码。

44.5 工作方式

44.5.1 初始化

在使用 SRAM 之前必须要对 SRAM 进行初始化。

非安全 SRAM 初始化流程如下：

- 步骤 1 开始初始化，配置 LP 控制器的 LPSC_SRAM_INI bit[0]（地址 0x0871D264）为 1'b1，这个信号为脉冲信号，自动回零，SRAM 开始初始化。
- 步骤 2 等待 LP 控制器的 LPSC_SRAM_INI bit[1]（地址 0x0871D264）为 1。则 SRAM 初始化结束。

----结束

安全 SRAM 初始化流程如下：



- 步骤 1 开始初始化，配置 SEC 控制器的 SRAM_MEM_INIT_START bit[0]（地址 0x085330A0）为 1'b1，这个信号为脉冲信号，自动回零，SRAM 开始初始化。
- 步骤 2 等待 SEC 控制器的 SRAM_MEM_INIT_DONE bit[0]（地址 0x085330A4）为 1。则 SRAM 初始化结束。

----结束

须知

非安全 SRAM 的初始化结束信号会随着协处理模式非安全域复位，但 SRAM 内部值不变，安全 SRAM 的初始化结束信号只会跟随 chip 域复位，需要软件注意使用上的不同。

44.5.2 单比特 ECC 错误回写使能和关闭

提供单比特 ECC 错误回写使能和关闭的寄存器，非安全 SRAM 的单比特 ECC 错误回写功能控制寄存器为 LP 控制器的 LPSC_SRAM_WR_BACK（地址 0x0871D3BC），默认使能，高有效，配置为 1'b0 时关闭该功能。

安全 SRAM 的单比特 ECC 错误回写功能控制寄存器为 SEC 控制器的 SRAM_1BIT_WR_BACK_EN（地址 0x0853320C），默认使能，高有效，配置为 1'b0 时关闭该功能。

44.5.3 ECC 错误上报

非安全 SRAM 的 ECC 错误中断上报寄存器为 LP 控制器的 LPSC_ECC_INT_RAW（地址 0x0871D320），bit[20]对应多比特 ECC 错误，bit[21]对应单比特 ECC 错误，写 1 清楚中断状态。

可以通过配置 LPSC_ECC_INT_MASK（地址 0x0871D324）bit[20]和 bit[21]来屏蔽中断，配置为 1'b1 为不屏蔽中断，配置为 1'b0 为屏蔽中断。默认屏蔽。

安全 SRAM 的 ECC 错误中断上报寄存器为 SEC 控制器的 RAM_ECC_ERROR_INT_STATUS（地址 0x085330B8），bit[1]对应多比特 ECC 错误，bit[0]对应单比特 ECC 错误，写 1 清除中断状态。

可以通过配置 RAM_ECC_ERROR_INT_EN（地址 0x085330B4）bit[0]和 bit[1]来屏蔽中断，配置为 1'b1 为不屏蔽中断，配置为 1'b0 为屏蔽中断。默认屏蔽。



提供非对齐写使能和关闭的寄存器，非安全 SRAM 控制寄存器为 LP 控制器的 LPSC_ECO_NONUSE0[0] (地址 0x0871D39C)，默认关闭，高有效，配置为 1'b1 时打开该功能。

安全 SRAM 控制寄存器为 SEC 控制器的 ROOM_FOR_SECURE_CORE_7 bit[1] (地址 0x0853336C)，默认关闭，高有效，配置为 1'b1 时打开该功能。

默认关闭的情况下，SRAM 写只支持 1,2,4,8Byte 对齐的写操作。打开后支持非对齐写。

操作该功能时请确保对 SRAM 没有进行操作。

44.6 DFX

支持 1bit ECC 错误回写和 2bit ECC 错误上报。



45 HPM

45.1 概述

Hi3093 芯片中放置两个 HPM，通过 HPM 模块的延迟反映芯片的工艺分布和温度变化。两个 HPM 的位置分别位于 LP SUBSYS 和 A55 SUBSYS。

45.2 特性

检测芯片的工艺分布和温度变化。

45.3 关键术语

表45-1 HPM 关键术语描述

术语	描述
HPM	Hardware Performance Monitor，监控单元，用来检测芯片不同 PVT 条件下的时延信息，通过 HPM 模块的延迟反映芯片的工艺分布和温度变化。



45.4 功能

HPM 为多个反相器收尾想接构成震荡环路，作为测试芯片工艺的传感器集成在芯片中。环路速度主要有 HPM 的电压和环路的阈值电压决定，与本身时钟频率无关。震荡速度越快，输出码值越大。

45.5 工作方式

45.5.1 时钟门控

LP 子系统 HPM 打开时钟流程如下：

步骤 1 配置 LPSC_CLKEN_CTRL bit[23]（地址为 0x0871D0014）为 1，打开时钟，默认打开。

步骤 2 HPM 关闭时钟流程如下：

配置 LPSC_CLKDIS_CTRL bit[23]（地址为 0x0871D1C0）为 1，关闭时钟。

步骤 3 HPM 查询时钟状态流程如下：

读 LPSC_CLKSTA_CTRL bit[23]（地址为 0x0871D1C4），1 表示时钟打开，0 表示关闭时钟。

----结束

A55 子系统 HPM 打开时钟流程如下：

步骤 1 配置 A55_CLK_EN bit[14]（地址为 0x14000064）为 1，打开时钟，默认打开。

步骤 2 HPM 关闭时钟流程如下：

配置 A55_CLK_DIS bit[14]（地址为 0x14000068）为 1，关闭时钟。

步骤 3 HPM 查询时钟状态流程如下：

读 A55_CLK_STATUS bit[23]（地址为 0x1400006C），1 表示时钟打开，0 表示关闭时钟。

----结束



45.5.2 软复位

LP 子系统 HPM 模块软复位流程如下：

- 步骤 1 配置 LPSC_SRST_REQ2 bit[6] (地址为 0x0871D00C) 为 1, 软复位请求有效。
- 步骤 2 等待 2us。
- 步骤 3 配置 LPSC_RESET_DREQ2 bit[6] (地址为 0x0871D204) 为 1, 撤消软复位请求。

----结束

A55 子系统 HPM 模块软复位流程如下：

- 步骤 1 配置 A55_SOFT_RESET_REQ bit[23] (地址为 0x14000000) 为 1, 软复位请求有效。
- 步骤 2 等待 2us。
- 步骤 3 配置 A55_SOFT_RESET_DREQ bit[23] (地址为 0x0871D204) 为 1, 撤消软复位请求。

----结束

45.5.3 HPM 时钟控制

LP 子系统 HPM 模块可以通过如下寄存器进行控制：

- 步骤 1 可以配置 LPSC_HPM_CTRL bit[7] (地址为 0x0871D314) 选择 HPM 输入时钟。
- 步骤 2 可以配置 LPSC_HPM_CTRL bit[6:1] (地址为 0x0871D314) 配置时钟分频。
- 步骤 3 配置 LPSC_HPM_CTRL bit[0] (地址为 0x0871D314) 为 1 使能 HPM 模块。

----结束

A55 子系统 HPM 模块可以通过如下寄存器进行控制：

- 步骤 1 可以配置 A55_HPM_CLK_CFG bit[7] (地址为 0x140000E8) 选择 HPM 输入时钟。
- 步骤 2 可以配置 A55_HPM_CLK_CFG bit[6:1] (地址为 0x140000E8) 配置时钟分频。
- 步骤 3 配置 A55_HPM_CLK_CFG bit[0] (地址为 0x140000E8) 为 1 使能 HPM 模块。

----结束



45.5.4 HPM 状态监控

- LP 子系统内 HPM:
 - 软件可以通过读取 LPSC_HPM_STATE (地址为 0x0871D318) hpm_pc_0_org 和 hpm_pc_1_org 来获取 LP 子系统内 HPM 两个通道的监控值, 这两个值在 hpm_pc_valid 为高时有效。
 - 当 HPM 时钟计数器溢出的时候, 会通过 ESPI 错误中断进行上报。可以在 ESPI_ECO4[5:4] (物理地址 0x8730028) 寄存器上进行查询, 通过 ESPI_ERR_STAT[24](物理地址为 0x87300A4)写 1 进行中断清除。只有 LP 子系统内 HPM 时钟计数器溢出可以上报中断。
- A55 子系统内 HPM:
 - 软件可以通过读取 A55_HPM_STATE (地址为 0x140000EC) hpm_pc_0_org 和 hpm_pc_1_org 来获取 A55 子系统内 HPM 两个通道的监控值, 这两个值在 hpm_pc_valid 为高时有效。

须知

LP 子系统内 HPM 时钟计数器溢出的中断是与 ESPI 的错误中断复用。当检测到 ESPI 的错误中断时, 需要读取 ESPI_ECO4 寄存器判断是哪个中断源。

对于只使用 LPC 的场景, ESPI 的错误中断只体现 HPM 的时钟计数器溢出中断源。



46 IOCONFIG

46.1 概述

IOCONFIG 是 Hi3093 的数字管脚复用控制模块，可以配置管脚复用和数字管脚工作特性。

46.2 特性

IOCONFIG 具有以下特性：

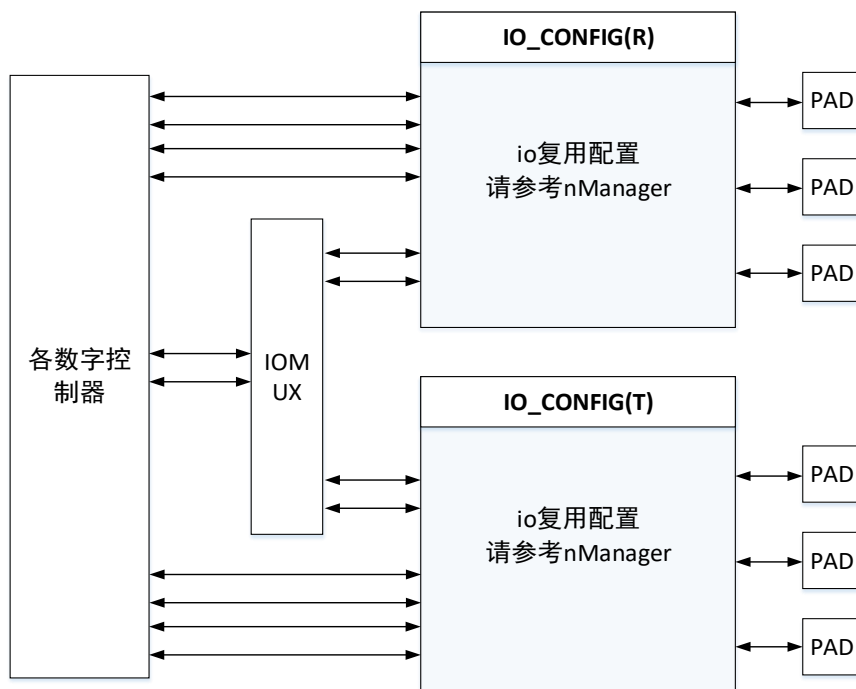
- 支持两个 IOCONFIG。
- 支持防对冲设计。
- 支持 APB 总线。
- 支持管脚工作特性配置。
- 支持管脚上电采样寄存。

46.3 功能描述

46.3.1 应用框图

IOCONFIG 的应用框图如图 46-1 所示。

图46-1 IOCONFIG 应用框图



由于芯片内部逻辑位置原因，IOCONFIG 被分为 IOCONFIG(T)和 IOCONFIG(R)，这两个部分共同组成了 Hi3093 的管脚复用模块。

IOCONFIG(T)和 IOCONFIG(R)的基址分别为 0x08745000 和 0x0876E000。分别控制各自管脚的复用关系和管脚特性配置。具体请参考《Hi3093 寄存器手册》

46.3.2 管脚信号重用模块 IOMUX

为了增加管脚配置的灵活性，I2C，UART，SPI 和 JTAG 还进行 IOMUX 选择。

其中 SPI1，JTAG1 和 IPMB0~3 分别通过 IOMUX 同时复用到了 IOCONFIG(T)和 IOCONFIG(R)。

通过 IOMUX，I2C8~I2C12 控制器中选择一个复用到 I2C16 端口上面，默认 I2C8~12 端口和控制器一一对应，I2C16 端口悬空，复用状态下可以选择 I2C8~12 控制器任一个复用到 I2C16 端口上，同时原来的端口悬空。

通过 IOMUX，UART2 控制器可以选择去 UART2 端口和 UART8 端口，UART3 控制器可以选择去 UART3 端口和 UART9 端口，UART4 控制器可以选择去 UART4 端口和 UART10 端口，UART5 控制器可以选择去 UART5 端口和 UART11 端口，UART6 控制器可以选择去 UART6 端口和 UART12 端口，UART7 控制器可以选择去 UART7 端口和 UART13 端口。



SPI1, IPMB0~3, I2C, UART 的 IOMUX 请参考 LP_SUBSYS 控制器 LPSC_IOMUX_CTRL (物理地址为 0x0871D334)。

JTAG1 请参考 SYSCtrl 的 PROBE_CTRL bit[4] (物理地址为 0x08740218)。

46.3.3 启动功能

上电后具有上电采样功能的管脚会自动对外部的高低电平进行采样锁存，其中 STRAP 管脚为芯片内部逻辑使用，BOOTCFG 管脚只进行锁存提供给软件进行读取。

采样过后，启动相关的管脚会根据 STRAP 采样所选择的启动方式自动切换改模块为默认管脚。

启动管脚具体请参考《Hi3093 硬件设计指南》中的管脚复用列表。

46.3.4 防对冲功能

芯片管脚有和外部对接器件的防对冲设计。在使用所需模块前，需要对相关管脚进行配置，这里需要注意的是，即使模块的信号为管脚默认功能，也需要对该管脚配置 0 以解除防对冲状态。

46.3.5 管脚属性配置

IOCONFIG 中提供寄存器控制 PAD 的 Schmitt, 上下拉, 驱动能力等属性，具体请参考《Hi3093 寄存器手册》。