

**RDMA 小核**

**详细设计方案**

|  |  |
| --- | --- |
| 项目名称 |  |
| 项目编号 |  |
| 内部代号 |  |
| 拟 制 |  |
| 批 准 |  |
| 审 核 |  |
| 日 期 |  |

**中科驭数（北京）科技有限公司**

版权所有 不得复制

**历史版本**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 作者 | 修改日期 | 修改内容及原因 |
| V1.00 | 陈国涛 | 2024/3/5 | 初稿 |
| V1.01 | 蒋帅 | 2024/7/25 | 增加 地址空间分配, 增加中断分配. |
|  |  |  |  |
|  |  |  |  |

目录

[1. 引言 1](#_Toc28774)

[1.1. 范围 1](#_Toc21106)

[1.2. 读者对象 1](#_Toc1522)

[1.3. 参考资料 1](#_Toc25593)

[1.4. 术语与缩略语 1](#_Toc4091)

[2. 模块汇总 1](#_Toc27296)

[2.1. 模块列表 1](#_Toc27621)

[2.2. 模块数据流 2](#_Toc27883)

[2.2.1. Host to IMP 2](#_Toc29193)

[2.2.2. IMP to Host 3](#_Toc17940)

[2.2.3. PIPE to IMP 3](#_Toc27635)

[2.2.4. PCC 4](#_Toc28613)

[2.2.5. imp packet 5](#_Toc12721)

[2.2.6. Pipe2imp 5](#_Toc15033)

[2.2.7. PIPE ERR 6](#_Toc20074)

[2.2.8. CMD Err 7](#_Toc30061)

[2.2.9. Modify\_qp 7](#_Toc20179)

[2.3. 模块功能描述 7](#_Toc14656)

[2.3.1. 虚拟化 8](#_Toc12360)

[2.3.2. Channel 8](#_Toc6584)

[2.3.3. Qmap 8](#_Toc23938)

[2.3.4. 初始化 9](#_Toc3779)

[2.3.5. M3 9](#_Toc20048)

[2.3.6. DMA 12](#_Toc31278)

[2.3.7. 地址划分 12](#_Toc18870)

[2.3.8. Ring\_buffer 12](#_Toc27048)

[2.3.9. Jtag 12](#_Toc1629)

[2.3.10. Axi4\_bus 13](#_Toc525)

[2.4. 模块接口时序 13](#_Toc833)

# 引言

## 范围

本文档主要介绍小核模块的设计与仿真要求，是仿真人员进行仿真模型设计的一个参考文档，文档面向FPGA设计人员、验证人员。

## 读者对象

本文档读者对象为FPGA工程设计人员，以及与本项目相关的边缘人员。

## 参考资料

| **资料名称**  **[标识符]** | **出 版 单 位** | **作 者** | **日 期** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 术语与缩略语

| **术语、缩略语** | **解 释** |
| --- | --- |
| NVIC | Nested Vectored Interrupt Controller |
|  |  |
|  |  |
|  |  |

# 模块汇总

## 模块列表

|  |  |
| --- | --- |
| M3 | ARM context M3 cpu |
| A2X | Synopsys DW库中的axi 转换 为AHB，或者AHB转换为AXI |
| AXI4 BUS | Synopsys DW库中提供的AXI4 CrossBar |
| Host | Host 主机，Root complex 口 |
| DMA | M3 可以通过DMA访问片上寄存器，大片SRAM或者HOST |
| DB | 环回DoorBell，例如TX pipe需要还回某个队列，RX RSP需要发送等 |
| PF & VF space | 寄存器地址划分 |
| Ring\_buffer | RDMA PIPE主动上送的任务描述符或者HOST 和小核通信的任务描述符等。 |
| Jtag | Debug 模块 |

## Feature list

1. M3 --> ITCM, DTCM通路，
2. M3 --> S6 通路；

|  |  |  |
| --- | --- | --- |
| 序号 | 通路 | 功能点 |
| 1 | M3 --> ITCM, DTCM  （axi bus port M1--M5） | 1. M3 ahb lite 支持 AMBA 3 协议标准； 2. M3 ahb lite Icode port， 只支持读操作；   2，M3 ahb lite D code port， 支持读写操作；  3，M3 ahb lite system port， 支持读写操作； |
| 2 | PLDA-->common reg | 1. 小核工程版本号，默认值：0.0.0 2. 小核工程日期：默认值：20240512 |
| 3 | S6 --> s1 itcm | 1, dma 从 flash 中将 m3 启动文件搬到 s1 对应的itcm中 |
| 4 | S6 --> s2 itcm | 同上; |
| 5 | S6 --> s3 itcm | 同上; |
| 6 | S6 --> s4 itcm | 同上; |
| 7 | S6 --> s5 itcm | 同上; |
| 8 | S7<-->s9 | Dma 在 s7 的ring buf 和 s9 union buf 之间互相搬移数据; |
| 9 | S7<-->s10 | Dma 在s7 口的 ring buf 和 rdma pipe 之间互相搬移数据; |
| 10 | S7<-->s1,s2,s3,s4,s5 | Dma 在s7口的ring buf 和dtcm 之间互相搬移数据; |
| 11 | S9<-->s1,s2,s3,s4,s5 | Dma 在s9口的union buf 和dtcm 之间互相搬移数据; |
| 12 | S7<--> host(s8) | Dma 在s7口的ring buf 和 host 之间互相搬移数据; |
| 13 | M1,m2,m3,m4,m5 <--> s6 | M3 处理器打印 时访问uart; 后续支持uart命令行发送命令到 m3 处理器; |

## 模块数据流

M3 通过内部总线 队列AXI4 bus访问DMA和RDMA Slave口空间。Host 经过PCIe EP的控制器转换后，相关访问通过AXI4 bus路由到相应的地址空间上。还回DB也通过AXI4 BUS 路由到DoorBell地址上。M3需要通过DMA访问HOST和片上大片SRAM的能力。和小核交互的共享小核的数据空间DTCM。小核/PIPE/HOST有多条数据流。

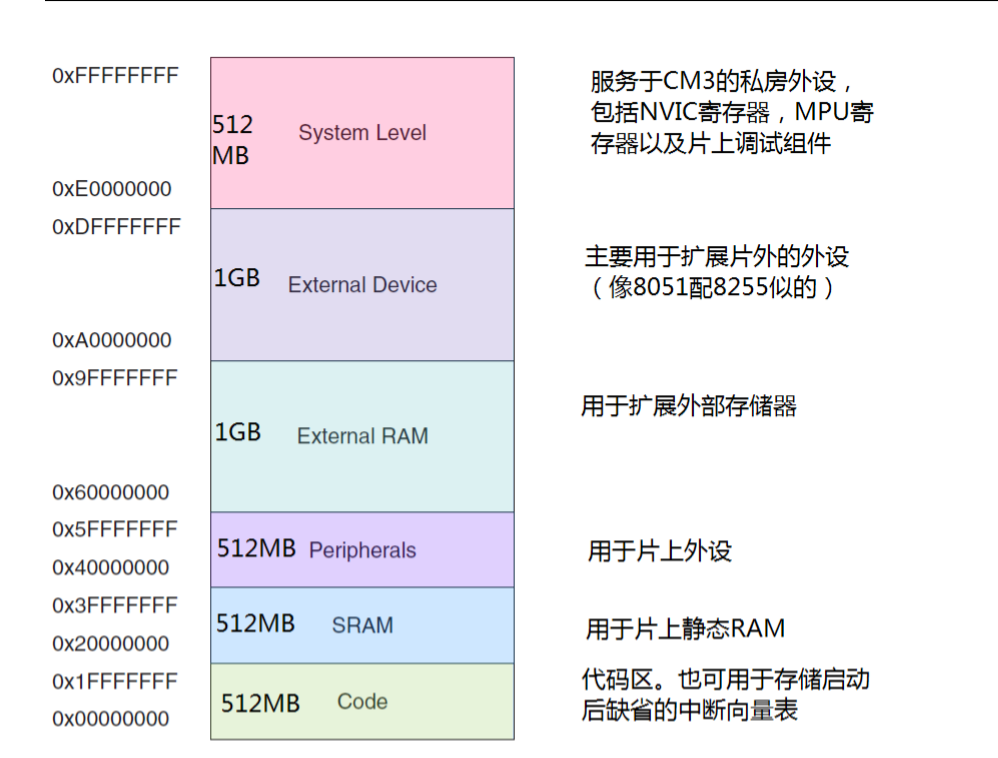


图 1片上默认空间分配

### Host to IMP

1. Host 将mailbox数据下发到VM bar空间的地址上（位于bar4的第2个4K的第一个64B），mailbox中应该包含返回命令的地址信息，PCIe等信息。该mailbox经过PCIe EP control，需要经过地址remap；该信息经过axi bus路由，有可能间插；数据到达slave口上；实现一个1K\*64B的sram，每个64kB是16个4B信息，当16个4B全部达到时，host下发中断，imp主动将数据搬移出这个SRAM；
2. 硬件将该64B搬移到Up队列，UP队列；更新UP队列PI指针，
3. 通过中断告诉小核有任务
4. 小核处理
5. 小核将回写信息更新进入DW队列，该数据结构为64B，前16B为控制信息，告诉硬件将该消息路由到host的位置和中断使能等
6. 小核调度DMA或者硬件回写64B，回写该信息，根据要求是否发送中断。



### IMP to Host

Host 为IMP主动上送的mailbox分配一段空间（位于HOST DDR中），IMP自己维护，下发到DW队列的数据结构和host to imp的返回数据结构相同

硬件执行第6步，路由该信息去host DDR空间中，并选择是否发送中断。

### PIPE to IMP

1. 硬件PIPE发现异常后，通过硬连线中断通知IMP；
2. IMP来查询该PIPE的寄存器；



### PCC

1. 接收端将流控信息{time，oop，rtt，ack，nak，ecn，cnp}等放入up队列
2. 通过中断通知PCC核或者PCC核自己轮询
3. PCC通过HQOS的叶子节点的rate影响流量

### imp packet

小核当成一个host，发送端完成报文填写，接收端完成报文解析。报文直接访问EPB或者IPB寄存器，获取报文或者发送报文。

### Pipe2imp

1. PIPE发现异常后，停止该条流
2. PIPE更新错误信息进入VOQ RAM的错误域
3. 该条PIPE最终会反压IPB



### PIPE ERR

1. Client 代表本地报文，Client tx端是一个生产者，即更新Egress Chain和CRRL，Client Rx端是一个消费者模型，即消费Egress Chain 和CRRL
2. Server代表对端报文，即Server Rx更新Ingress Chain和SRRL，Server Tx 消费Ingress Chain和SRRL
3. 当Client Tx PIPE出现Err后，其它的三条链路继续工作，Client Tx Pipe负责将该信息路由个IMP核，此时Client Rx继续处理报文，如果小核有报文发送，需要 （小核组包发给epb） ~~Client Tx侧路由~~
4. 小核配置qpc.Client Rx Err，Pipe每一次RTC开始时，读取 qpc和IMP\_inf,如果知道错误后，则随着pipe路由该报文给IMP。Client rx drop信号控制报文是否drop或者上送IMP，如果PIPE上有同一个队列的报文，该报文则需要进入相同的虚通道
5. 小核配置Server Tx出错，回收Server Tx权限
6. 小核配置Server Rx出错，回收Server Rx权限
7. 小核接管所有后续报文
8. 处理完异常后，小核负责协议的后续事件
9. 小核回收硬件资源



### CMD Err

Host 通过mailbox通知IMP更改QP\_State状态，小核配置Client Tx Err/Client Rx Err/Server Tx Err/Server Rx Err，小核接管权限，并负责清理硬件资源和软件资源。

### Modify\_qp

驱动或者小核想修改qpc时，需要暂停调度，查询PIPE状态，然后修改相关qpc。具体流程如下：

1. 配置队列暂停
2. Imp查询TX PIPE状态，
3. 无效qpc
4. 修改qpc
5. 重新开始任务。

修改过程中，造成接收端的丢包，通过重传解决。或者通过更上层的握手保证业务暂停后，修改QPC。~~无法支持在线修改qpc。~~

## 模块功能描述

IMP小核处理RDMA核初始化和异常，完成RDMA核和驱动的通信。驱动通过Verbs调用RDMA 队列/Channel/QPC资源时，该资源由小核管理。小核需要完成QP State状态。当PIPE发生异常时，通过中断上送小核，小核需要完成PIPE VOQ的接管，回收硬件资源，完成err\_status的查找，以及相关信息的打印等。

当用户希望在线修改QPC或者PCIe FLR时，IMP小核需要协助走完硬件PIPE清空流程，完成相关更改或者复位。

需要按照协议完成错误WQEs的上报，flush cqes的上报等。如果硬件某一个QP损坏，IMP需要查找到该QP的虚拟机，复位该虚拟机下面的所有资源。如果上层用户注销或者热迁移虚拟机，小核需要维护该虚拟机的断点信息。

PCC完成用户可编程流控。硬件上送流控控制信号，小核按照流控控制信息完成流控算法，并转换为发送端的流量信息。

### 虚拟化

SRIOV的虚拟化如下所示，一般是PF+VF的格式，其中VF&BDF的编码连续，地址连续，VF的空间是等间距的，PF则可以不通过。PF空间开放片上sram，让host可读，屏蔽所有写操作。



### Channel

Channel的初始化由小核完成。Channel表项查阅全局数据结构文档。全局使用Channel\_id + qpn 作为队列唯一标识。

本版本不实现qmap功能。

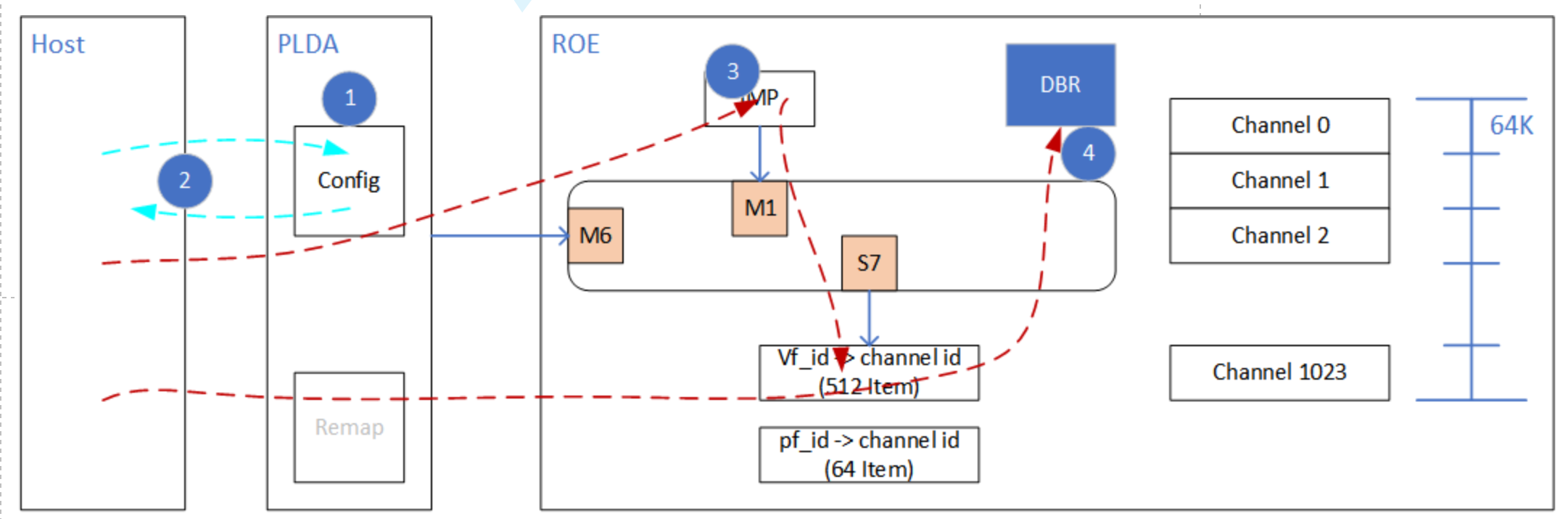
### 初始化

#### Qp初始化

Verbs调用create\_qp, 该信息封装成mailbox，通过mailbox通道下发到小核，小核和驱动共同完成Channel表项，qpc的初始化。完成qp\_state的维护

#### Vm初始化

1. Pcie link后，硬件会预设置其相关参数；每个虚拟机都会有自己的config空间和bar空间；小核应该指定vf id 去访问其对应的config 空间并具有解析其config空间的能力；
2. Host驱动获取pcie配置后，需要通过 bar4，bar5对应的地址空间的前64kB下发mailbox信息（4K地址以后）
3. Host驱动通过mailbox下发建立虚拟机的命令，命令里会带func id，固件会根据命令返回channel id。并更新{hostid，pf id，vf id} --> channel id的映射表；以及channel id -- > {hostid，pfid, vfid}的映射表；
4. host 驱动通过bar空间的第一个64k下发dbr；硬件会主动根据 其接口的边带信号携带的func id 和64K 偏移信息查询 func id--> channel id映射表；转换得到channel id，pipe 根据channel id和qpid ，其可得到真正的qpc信息；



#### HQOS初始化

产生虚拟机和qp时，需要驱动允许相关的代码，产生qp/cos,vm/Prority，qp的流量信息，vm的流量信息，port信息，即驱动和小核需要更新HQOS的class信息。

### M3

#### M3架构图

M3 需要一组时钟复位信号，复位为异步复位。需要看门口功能WIC。需要一组AHB BUS。需要JTAG能访问。



#### M3 接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 位宽 | 描述 |
| Clk | 1 | 时钟信号 |
| Rst\_n | 1 | 异步复位信号 |
| Interrupt | 40 | Host使用8个中断，每个host两个  Pipe使用32个中断，PIPE每一级一个 |
| Tck | 1 | Jtag 时钟 |
| Tms | 1 | Jtag控制信息 |
| tdo | 1 | Jtag输出信号 |
| tdi | 1 | Jtag输入信号 |
| Jrst\_n | 1 | Jtag复位信号 |
| AHB bus | 3组 | Master 口 |
| AHB bus | 1组 | Slave口 |

#### M3 ITCM & DTCM

ITCM和DTCM为外部例化的SRAM，M3不使用CS的时候，用户可以使用相关CS信息，直接读写数据。减少M3的访问延迟。ITCM和DTCM可以根据本身timing问题，设置multcycle。

#### Mailbox寄存器

一个channel分配1组mailbox 寄存器。数据结构如下，该寄存器为8份：

|  |  |  |
| --- | --- | --- |
| 信号名 | 宽度 | 描述 |
| Ch0\_dbr | 64 | 8B门铃寄存器 |
| Ch0\_dbr(32b) | 32 | 4B门铃寄存器 |
| Cmd\_depth | 4 | 2^n;  0000:表示可以存放一个command |
| Ch0\_mailbox | 512 | Mailbox , 16个4B; |
| Ch0\_mailbox\_dbr | 8 | 生产者指针 |
|  |  |  |

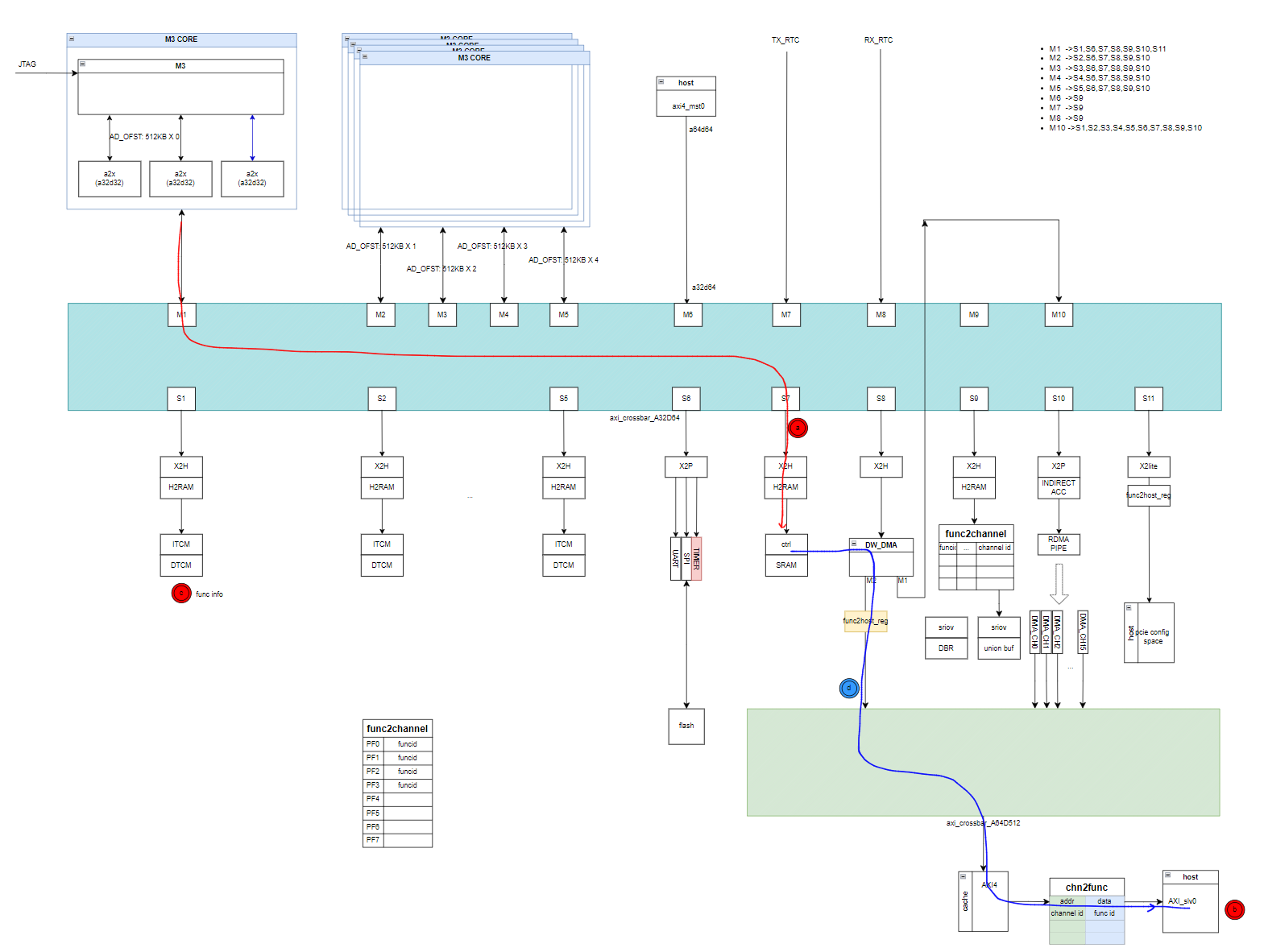
注意: 初版代码设计未使用upq 和 dwq , host 和m3 通信以 dma 的方式进行;

**通信过程:**

Imp寄存器: s07 口 基地址+ 128KB , 地址空间大小: 64K

Host DBR寄存器位置: s09口基地址 + 128KB , 地址空间大小: 128M

1, **imp --> host ;**



a, M3 先将需要发送的数据写入 s7口对应的sram 中, 记录好地址和长度;

b, host为小核分配 一个4K 的内存空间(以channel为单位); 并将此内存空间以mailbox 的方式告知小核, 同时包括 channel 信息, ;

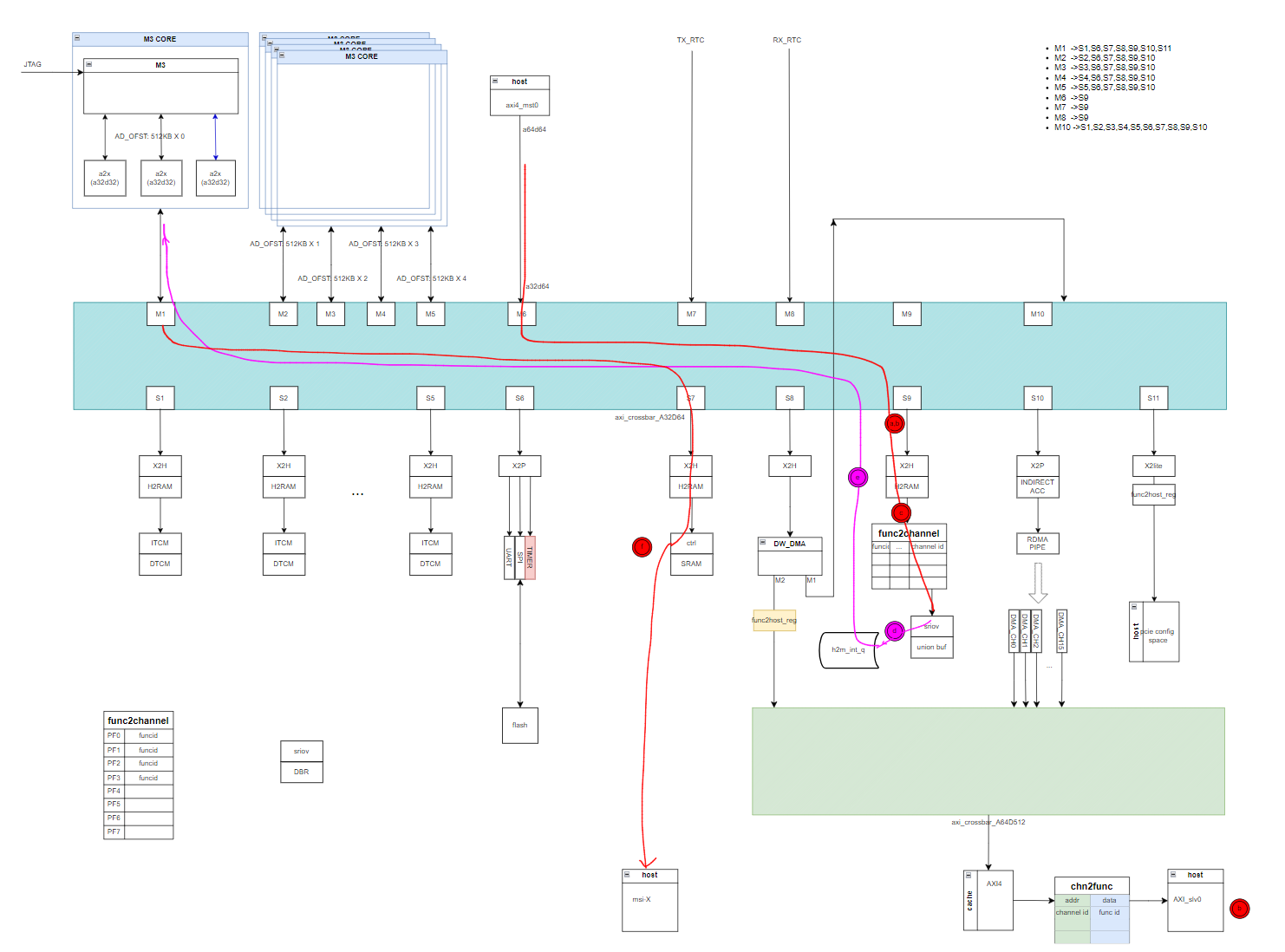
c, M3 根据记录的源地址和长度, 以及 host 侧的内存空间的channel信息反查得到 func信息;

d, M3根据上一步得到的func信息,配置 func info 2host 寄存器, 并根据源地址和长度, 目的地址, 启动dma ; 完成imp 到 host的 数据搬移; (此步骤为原子操作;)

e, 根据channel 信息, 以及func信息, 写to host int reg 中断通知寄存器, int vector 为m3 申请(每个channel 给M3 使用时至少分配一个), 硬件触发一次msi-x 中断通知 host ; (中断通知寄存器需要软件主动清0)

f , host 根据中断向量号 和 func信息,从内存中读取数据;

**2,Host ---> imp**



a , host 将要发送的mailbox 数据, 写入对应channel 的mailbox 中(硬件需要查询 func2channel 表);

b , 按对应channel 的 mailbox门铃(硬件需要查询 func2channel 表);

c , 硬件根据 s09口边带信号携带的 func 信息, 查询 func2channel 表, 得到channel id; 并将这些信息,填充到 h2m\_int\_q中(硬件需要维护一个中断请求的队列), 当中断请求队列非空时, 硬件主动向 m3 发送一次 中断, 表示host 有数据需要 m3 搬移;

d , m3 收到 中断后, 读取 h2m int q deep 获得当前中断请求队列的当前深度; 并根据此深度从h2m int q 中 读出对应数量的中断信息( func和channel 信息); 中断请求队列出口地址 : s07口基地址+ 128K 即: 0x2002\_0000;

e , m3 根据 中断请求队列的 func 信息和 channel 信息; 从对应channel 的 host mailbox 中读取 数据; 从而完成数据搬移;

f , 如需要双方握手; m3 可通过 申请另外的中断向量号, 通过 s07口 的 to host int reg 向host 再次发送中断;

#### Command structure

Host 与IMP的数据结构由软件定义。

上送报文和下发报文的16B 需要定义

上送报文的数据结构：

|  |  |  |
| --- | --- | --- |
| 信号名 | 位宽 | 描述 |
| Voq\_num | 6+2rsv | Voq 的编号，即那个voq上送的任务 |
| Chanal\_id + qpn | 22(10+12) | Qpc的硬件的序号，小核根据该序号访问qpc |
| func id+host\_id | 16 | Pcie的function编码，DMA需求 |
| Pkt\_Pb\_index | 16 | 该报文位于pb的位置 |
| Pkt\_len | 16 | B为单位  可以描述64KB的报文长度 |

下发报文的数据结构：

|  |  |  |
| --- | --- | --- |
| 信号名 | 位宽 | 描述 |
| Voq\_num | 6+2rsv | voq的编码，即协助哪个voq发送报文 |
| Channel\_id+qpn | 22(10+12) | 发送那个硬件队列的报文 |
| func id+host\_id | 16 | Pcie的function编码，DMA需求 |
| Rdma\_pkt |  | Rdma 报文位于64B的后面，长度最大8K |

PIPE需要开发IMP访问片上buffer的结构，允许IMP按照硬件结构访问数据，按照收发报文信息更新qpc & crrl & egress & ingress & srrl。PIPE需要有bypass路径上送或者下发报文。

PCC由流控定义

#### M3 中断分配表

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **Bit** | **说明** | **备注** |
| **0** | **0** | **Pcie mst 口 写数据 完成中断** | **Host mailbox 通知中断;** |
| 1 | 4:1 | 待分配 |  |
| 2 | 5 | Uart 0 中断 |  |
| 3 | 7:6 | 待分配 |  |
| 4 | 8 | I2C0 中断 |  |
| 5 | 36:9 | 待分配 |  |
| 6 | 37 | I2C1 中断 |  |
| 7 | 63:47 | 未分配 |  |
| 8 | 46 | S08 dma 中断 |  |
| 9 | 239:45 | 未分配 |  |

#### Channel 表：

Channel表规则如下：

1. 每个func 都有bar4的地址空间，其中，每64KB可分配一个channel；
2. 每个func需要指定可以分配的channel个数；固定前面的连续64KB可以分配channel；
3. Func 内部Channel id分配必须连续；不支持乱序分配。（减小硬件复杂度）
4. 64KB空间的前4K为dbr，其余可供mailbox使用；
5. Channel 表分为 func 到 channel ， 以及 channel 到 func的映射

**数据结构：（func 到channel）**

pf ：{pfid，valid，start\_channelid, channel\_num}

Vf ：{valid，start\_channelid, channel\_num}

Valid: 指示此func到channel id的映射生效；

Start channel id: 此func 内，如果bar空间大于128KB，可以分配多个channel时，指定的起始channel id；

Channel num: 以start channel id 为起始， 共有channel num个 channel ;即：以start channel id + channel num -1 为本func 内结束的channel id；

**数据结构：（channel 到 func）**

Channel:{valid, func};

Valid: 指示此方向映射的表项生效；

Func: 此channel 时，对应的 func id （host id + pfid+vfid）；

**实现方式：**

Pf id -- channel id 表采用 reg 实现，每组reg 需要小核填写 pf id + {valid，start\_channelid, channel\_num}； 共8组，对应8个pf；

每次下发dbr时，进行全匹配查找。

Vfid -- channel id 表采用 ram 实现；ram内部数据结构如下图所示：



表 1 vfid 到channel id 映射 表

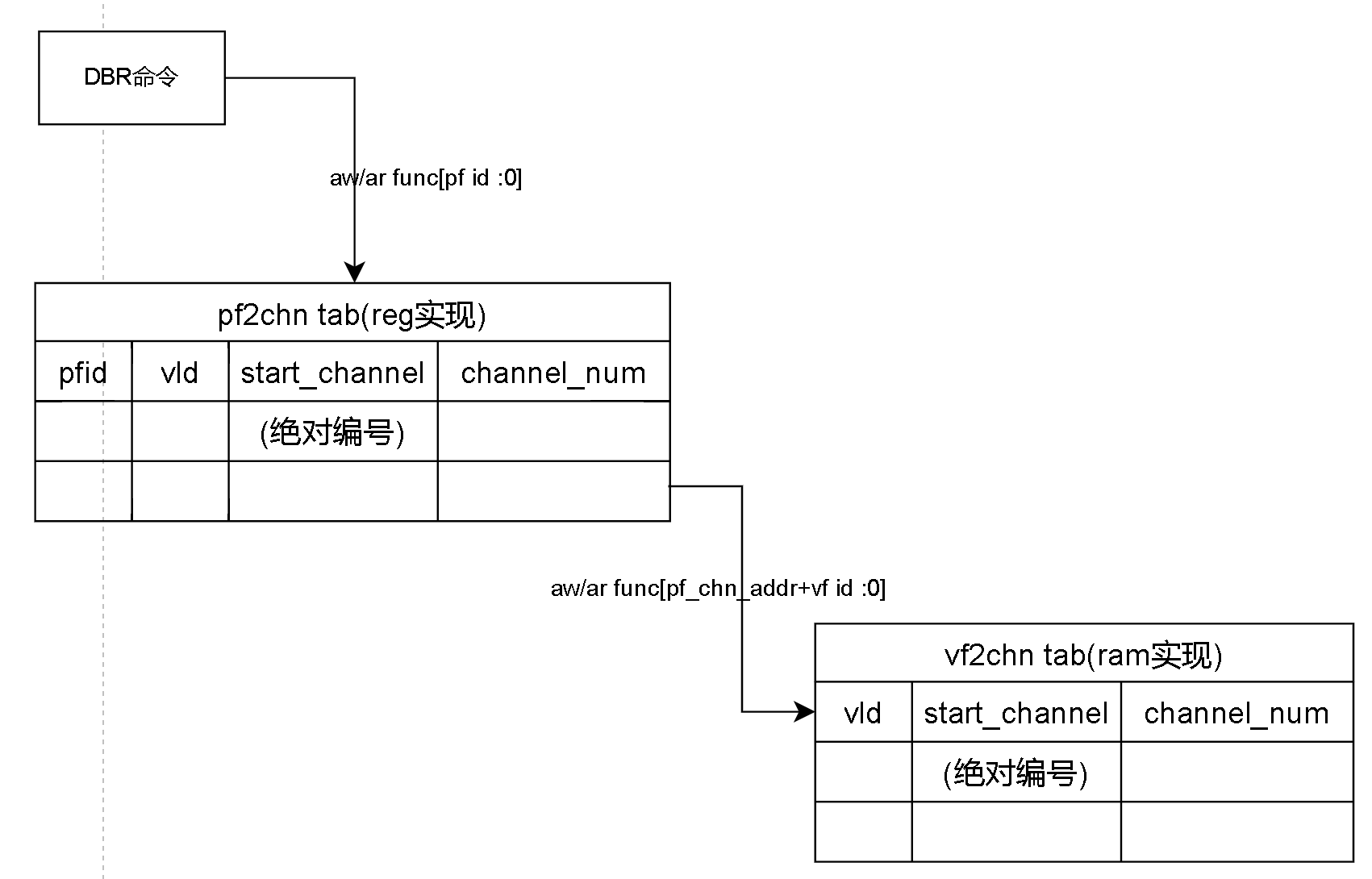


图 2 func到channel 表的查表流程

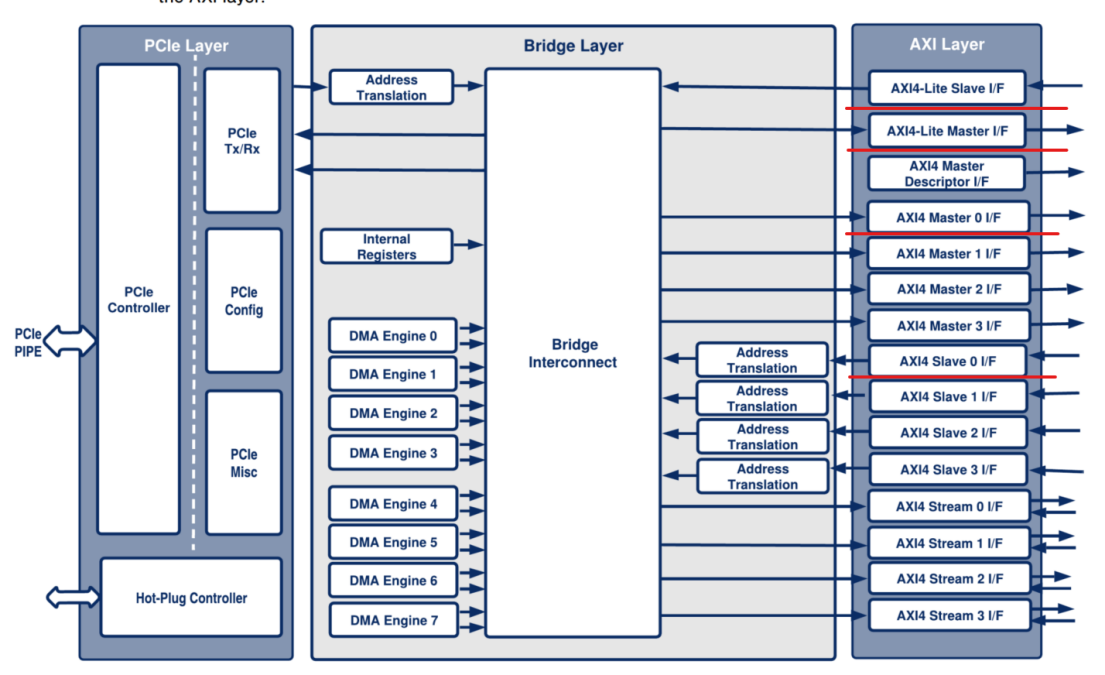
查表过程说明：

Dbr到来时，根据axi4接口边带信号携带的func id 信息，

1. 查询 8个pfid 到 channel id的映射表，进行全匹配； 得到pf下channl id 以及8个寄存器的地址vf\_grp，以及表项是否有效指示。
2. 如果vf id=9‘h0，则返回查表结果；
3. 如果vf id |= 9’h0，根据上一步得到的地址vf\_grp和vfid查询vf表；
4. 根据axi 接口的aw/ar addr[17:8] ,以及上一步的返回结果，换算得到channel id
5. 如果valid 都不生效，或者对应的 channel id 没有落入有效区间；则上报异常中断；

### PCIE

Pcie 采用公司 的PLDA, 接口如下:



IMP 部分接口描述(与 plda 接口)

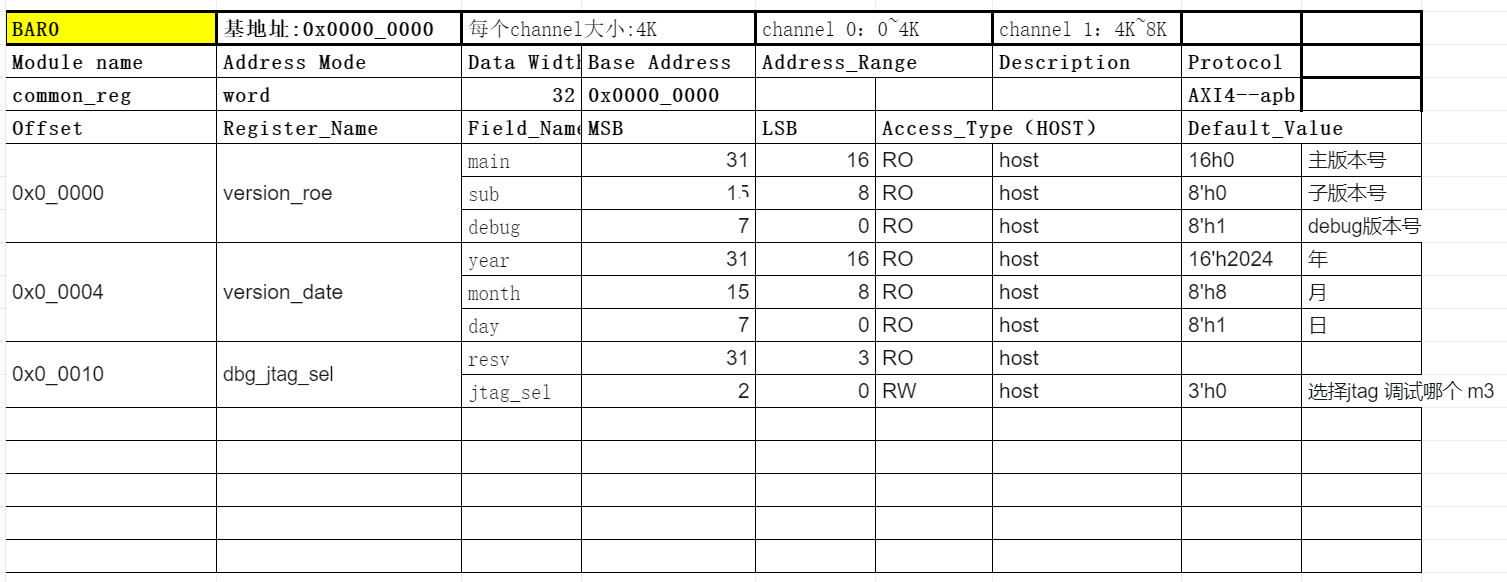
|  |  |  |
| --- | --- | --- |
| 信号名 | BAR | 描述 |
| Axi4\_slvl\_\* | 0 | Axi4 lite slv 口,用于接收 pcie axi4 mst lite 口的读写请求;(例如: VERSION\_ROE等) |
| Axi4\_mstl\_\* | 0 | Axi4 lite mst 口, 用于m3 发起读写 pcie config 空间(获取 bdf 号,见下表pcie bar 空间与地址空间描述); 与 pcie axi4 lite slv 口相连; |
| Axi4\_pcislv\_\* | 4 | Axi4 slave接口,用于接收 pcie 发起的门铃;(与 pcie axi 桥的 master 口相连); |
| Axi4\_dmamst\_\* |  | Axi4 master 接口,用于m3 发起dma 从 host mem 读写数据用; (与 rdma pipe 的 dma 接口的 crossbar相连; 单用于m3 的调试工程时, 需要与 plda的 axi4 桥的 slave 口相连) |
| Axis\_intmst\_\* |  | Axi stream 接口;  Plda 的中断接口,与 k2 pro的中断接口保持一致;见表 3; |

#### 接口功能描述:

1, axi4\_slvl \*

接口采用 axi4 lite协议, slave 模式, 被动接收 plda bar0 的访问; 基地址偏移: 0x0000\_0000;

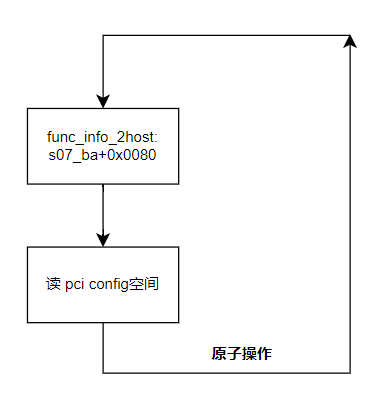
寄存器如下:



2, Axi4\_mstl\_\*

接口采用 axi4 lite 协议, master 模式; 主动发送 向 plda的 读写命令; 目前用于读取 plda 配置空间;

**工作流程:**



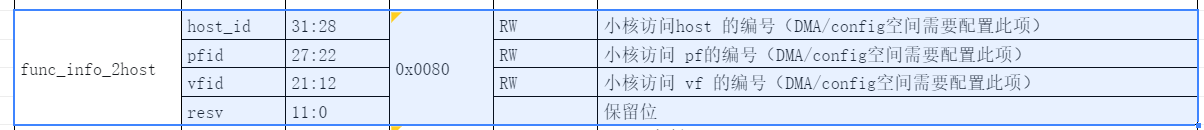
1, M3 配置 访问host的func info2host 寄存器, 指定需要访问的func 信息;

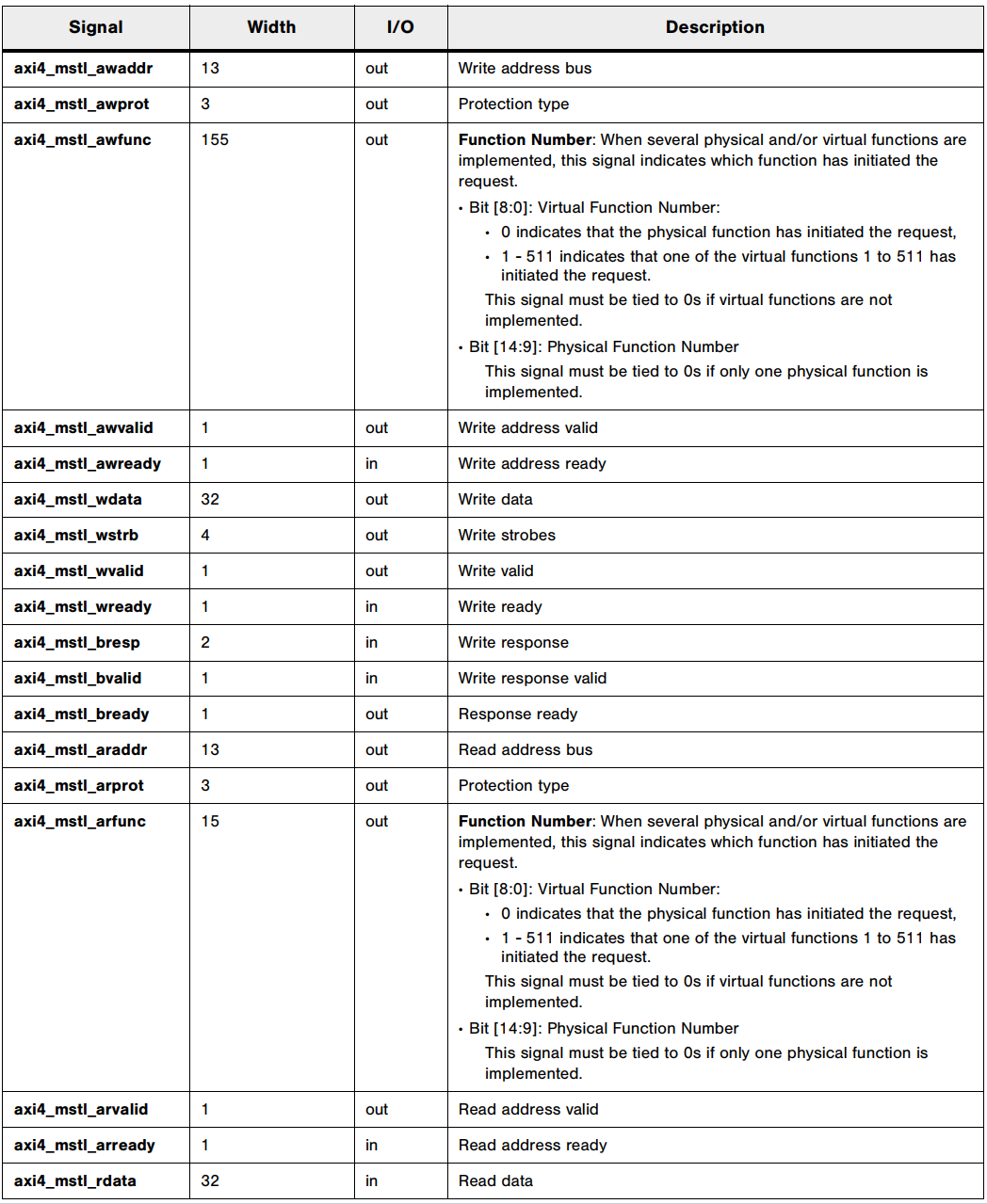
2, M3 通过 axi crossbar 的 s11 (地址偏移: 0x3800\_0000)口的 访问 pcie config 空间, 其地址偏移: 0x1000;

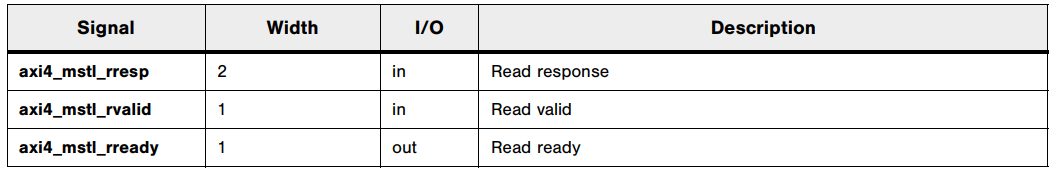
3, s11 口标准 axi4 协议不包含 awfunc / arfunc 信号; 硬件将func info 2host 寄存器转换为 plda 所需的 awfunc/ arfunc 信号;

4, 软件执行 读pcie config 空间操作, 即可得到对应 func 对应的pcie config 空间信息;

**需要注意的是: 两步读取的操作 尽量采用原子操作. 防止接口上awfunc / arfunc 信号的改变导致 读数据出错;**







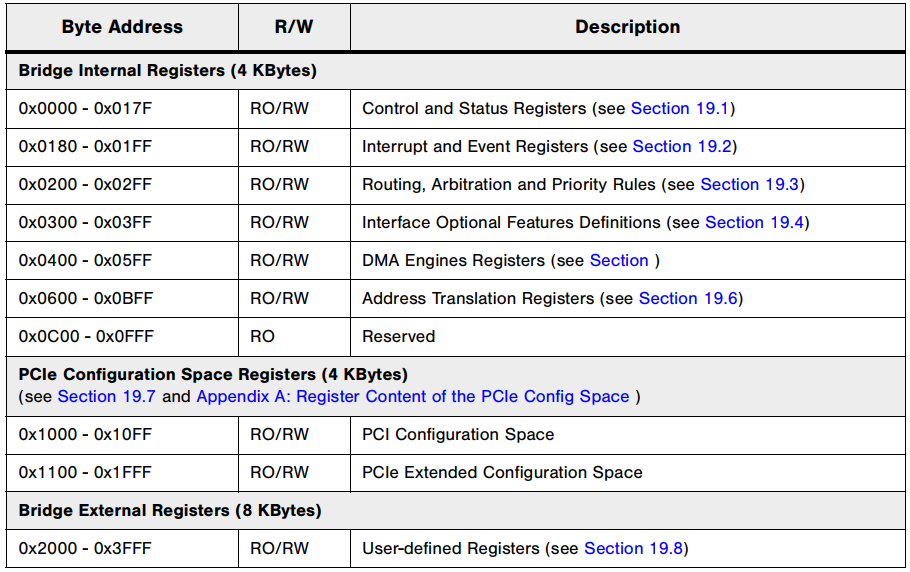


表 2 pcie axi lite slv 口 地址空间 (config 空间)

3，Axi4\_pcislv\_\*：

此接口映射为plda bar4， 基地址为0x0000\_0000; 功能为host驱动执行dbr操作，以及下发 mailbox使用；

门铃分为64bit门铃和32bit门铃，驱动应当根据情况使用相应的门铃地址；

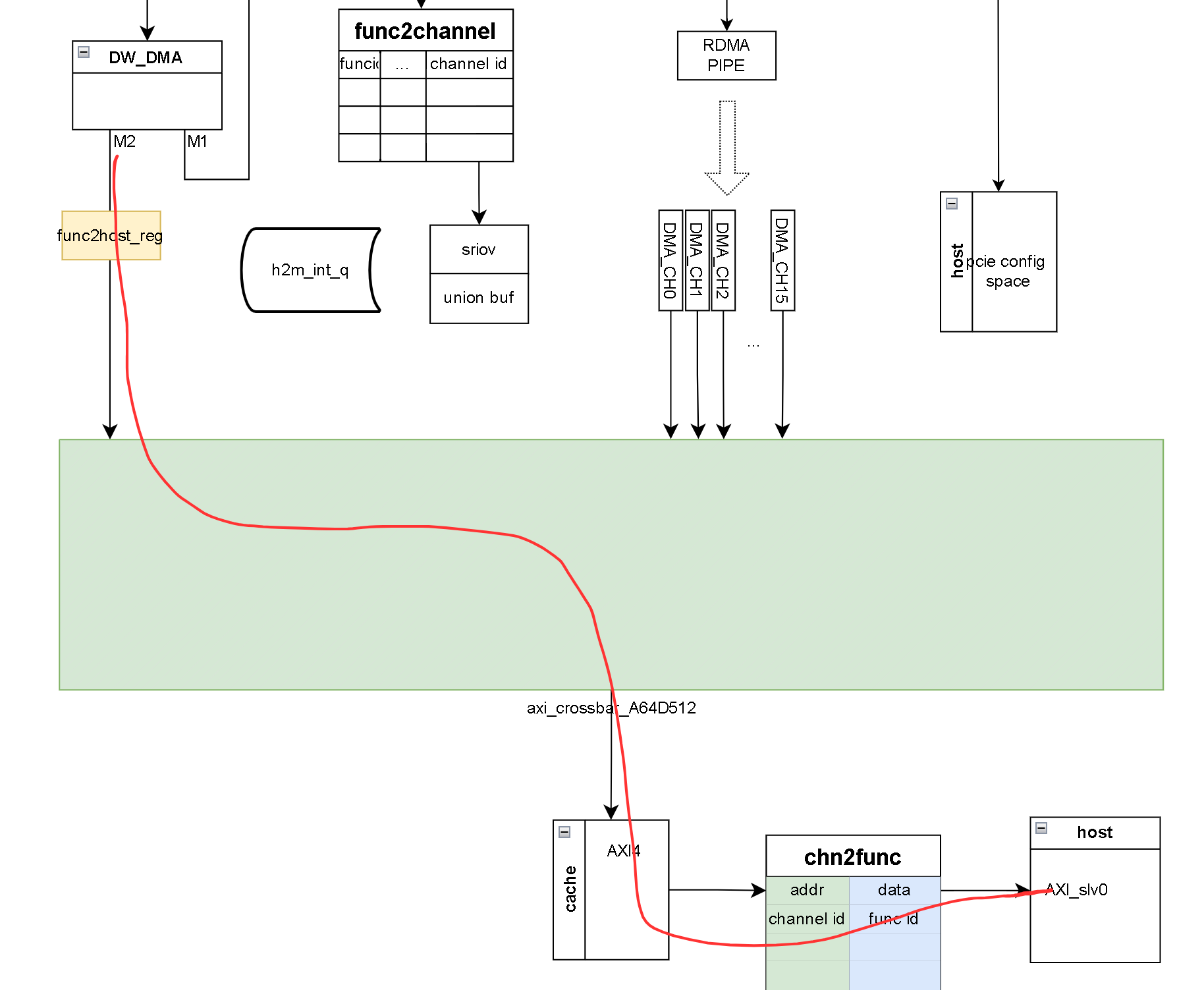
逻辑实现时， 硬件会主动偏移 0x2100\_0000，以适配内部bus 总线地址空间的划分；

4，Axi4\_dmamst\_\*

此接口为dma 接口，最终与plda axi桥层相连；

在ROE工程中，需要从pipe上的dma 模块的bus上分配一个 axi4接口；如下图所示。

注意：因为plda的dma接口上具有 awfunc/arfunc信号，所以小核启动dma 从host内存或者向host内存 搬移数据时，应该指定 本次dma 的func id和host id 信息；（寄存器func info 2host）



5，Axis\_intmst\_\*

此接口为二次封装接口，其数据结构如下图所示。

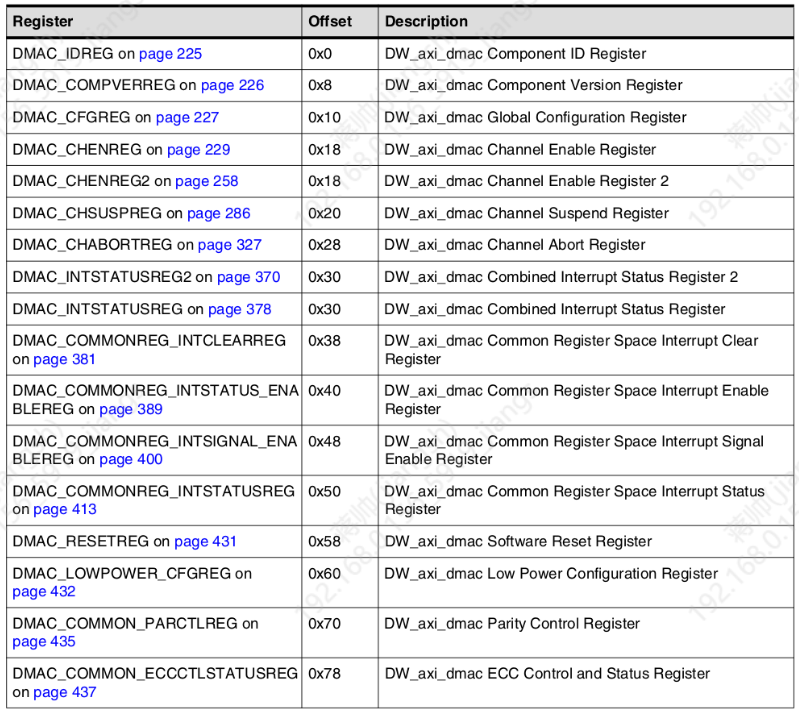
用于向host侧发送中断；接口执行axi4 stream协议；

Host驱动需要向pcie申请int vector，并与固件协同定义 不同int vector对应的含义；



### DMA

通用DMA逻辑，可以使用项目组DMA或者DW\_DMA逻辑。

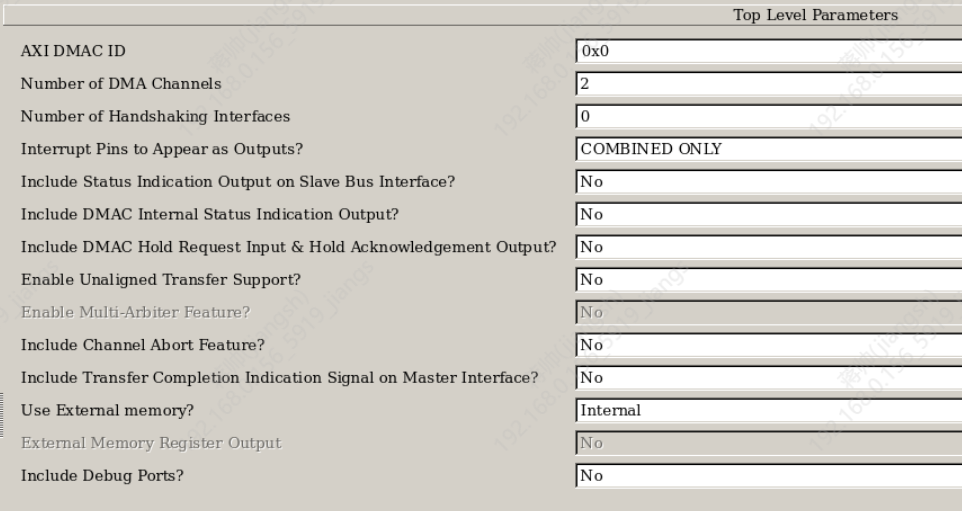


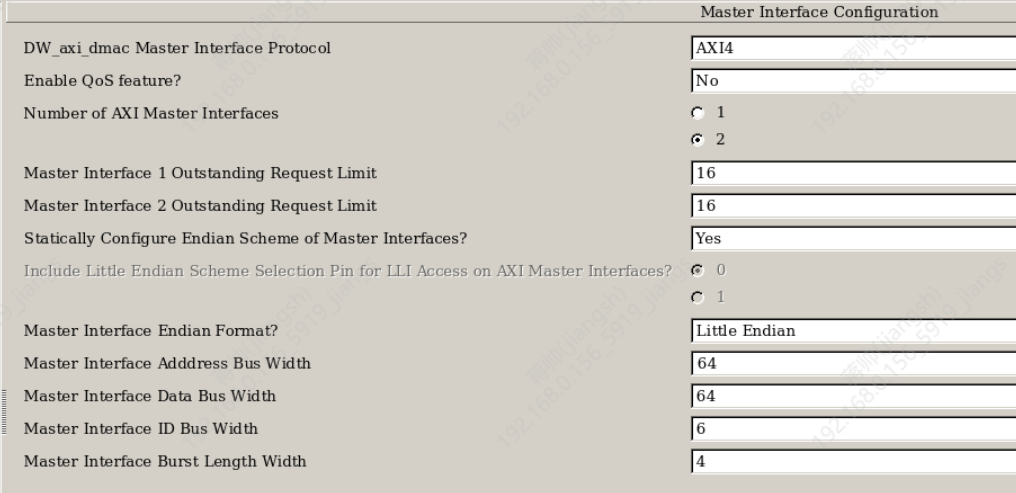
DMA寄存器描述：

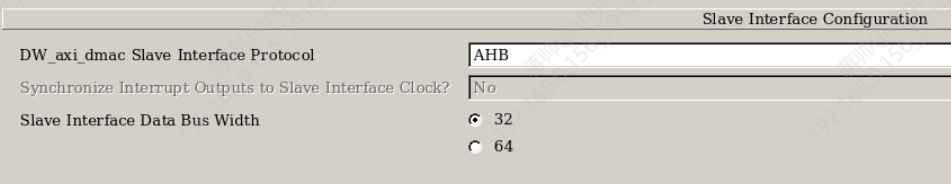


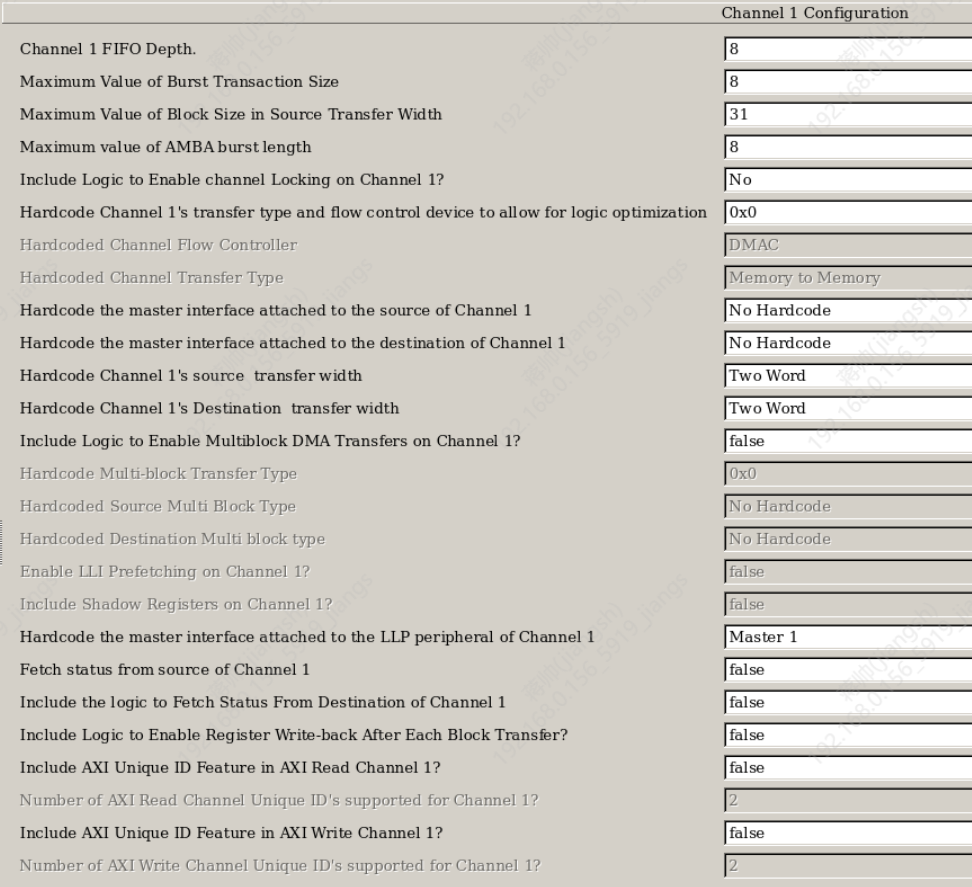
寄存器描述部分见 文档内6.2章节；

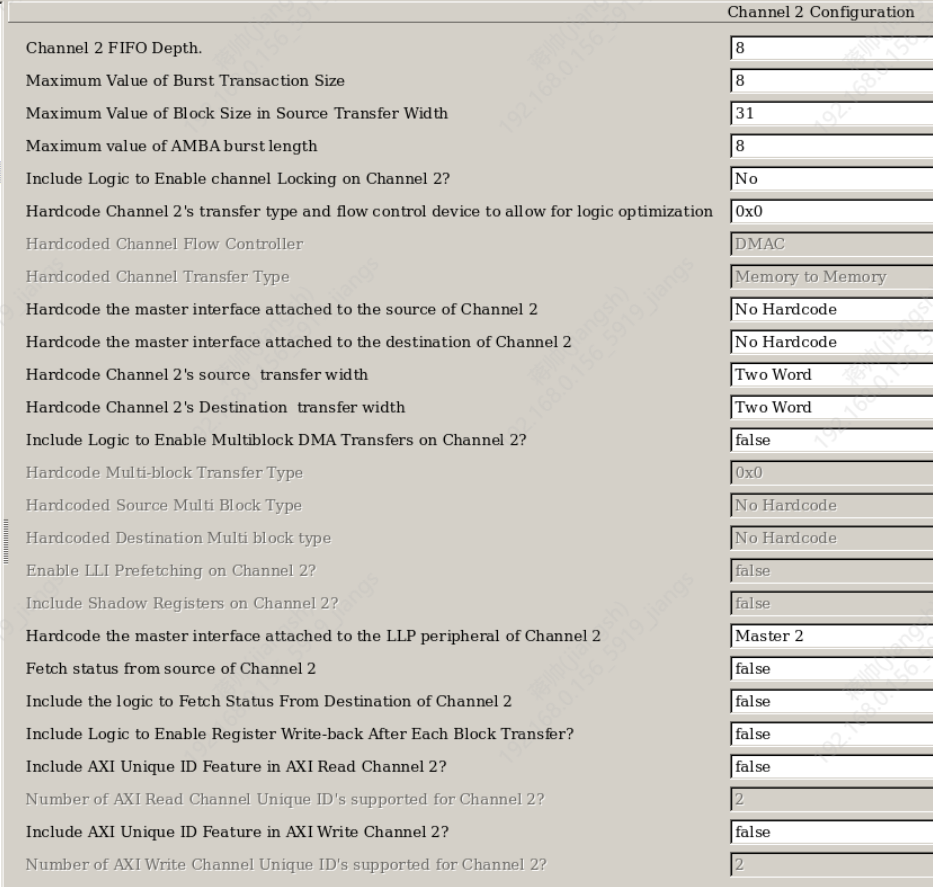
DMA规格:











执行Mailbox 时的Dma 的方式与 访问 config 空间类似,需要先配置本次与host 通信的func id;

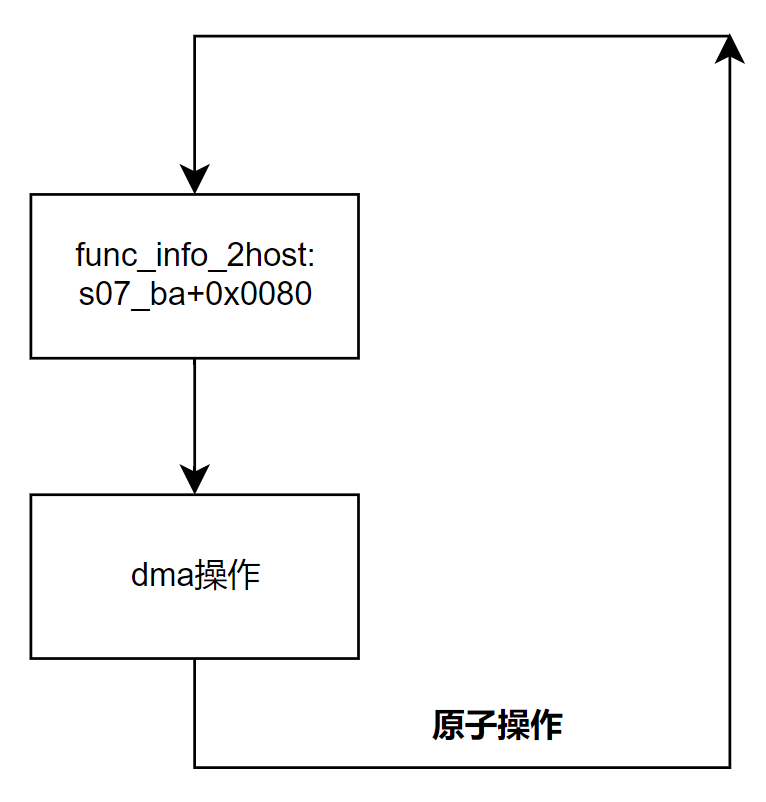


图 3 dma 执行步骤

### 地址划分

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Slv num** | **Num of regions** | **Start addr** | **End addr** | **Note** |
| Slv 1 | 1（） | 0x0 | 0x7\_ffff | 512kB |
| Slv 2 | 1 | 0x8\_0000 | 0xf\_ffff | 512kB |
| Slv 3 | 1 | 0x10\_0000 | 0x17\_ffff | 512kB |
| Slv 4 | 1 | 0x18\_0000 | 0x1f\_ffff | 512kB |
| Slv 5 | 1 | 0x20\_0000 | 0x27\_ffff | 512kB |
| Slv 6 | 1 | 0x4000\_0000 ~~0x28\_0000~~ | 0x4007\_ffff ~~0x2f\_ffff~~ | 512kB??? |
| Slv 7 | 1 | 0x2000\_0000 ~~0x30\_0000~~ | 0x20ff\_ffff ~~0x37\_ffff~~ | 16MB |
| Slv 8 | 1 | 0x4008\_0000 ~~0x38\_0000~~ | 0x400f\_ffff ~~0x3f\_ffff~~ | 512kB |
| Slv 9 | 1 | 0x2100\_0000 ~~0x2000\_0000~~ | 0x28ff\_ffff ~~0x3fff\_ffff~~ | 128MB |
| Slv 10 | 1 | 0x3000\_0000 ~~0x4000\_0000~~ | 0x33ff\_ffff ~~0x5fff\_ffff~~ | 64MB |
| Slv 11 | 1 | 0x3800 0000 ~~0x6000\_0000~~ | 0x38ff ffff ~~0x6fff\_ffff~~ | 16MB  前8KB空间, 用于访问 pcie 空间; 其中 0x1000--0x1ffff 用于访问 pcie config 空间; |
| Slv 12 | 1 | 0x3900 0000 ~~0x7000\_0000~~ | 0x39ff ffff ~~0x7fff\_ffff~~ | 16MB |

### Ring\_buffer

共享M3的DTCM，减少M3访问DTCM的延迟。

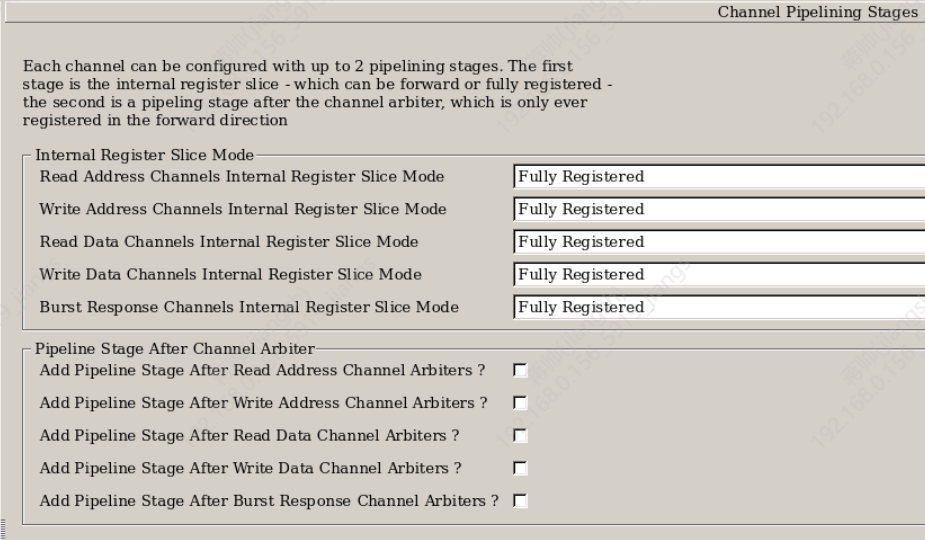
### Jtag

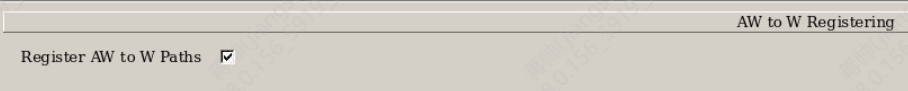
Jtag逻辑

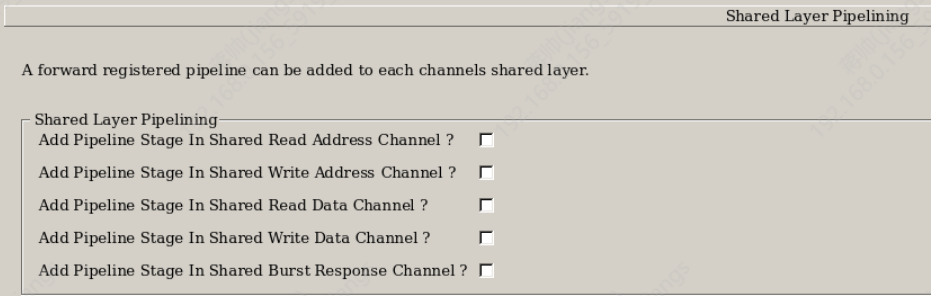
### Axi4\_bus

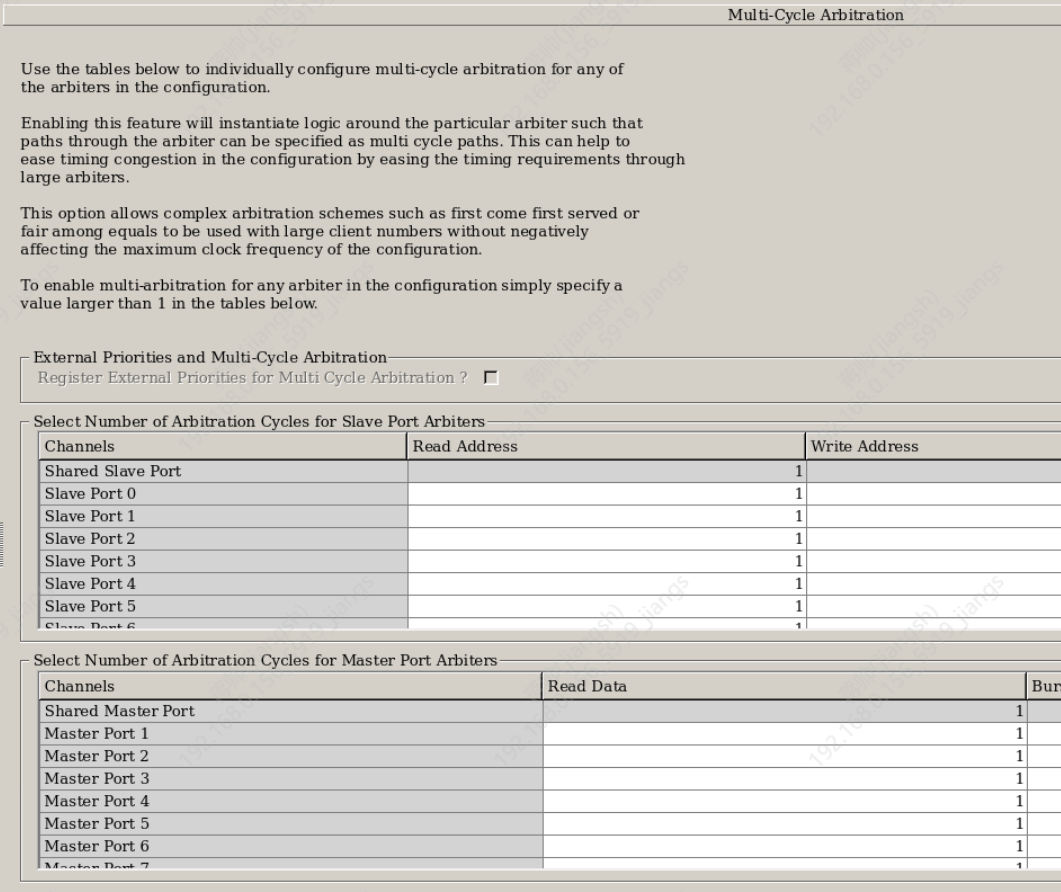
使用AXI4协议，需要12个master口，12个slave口;

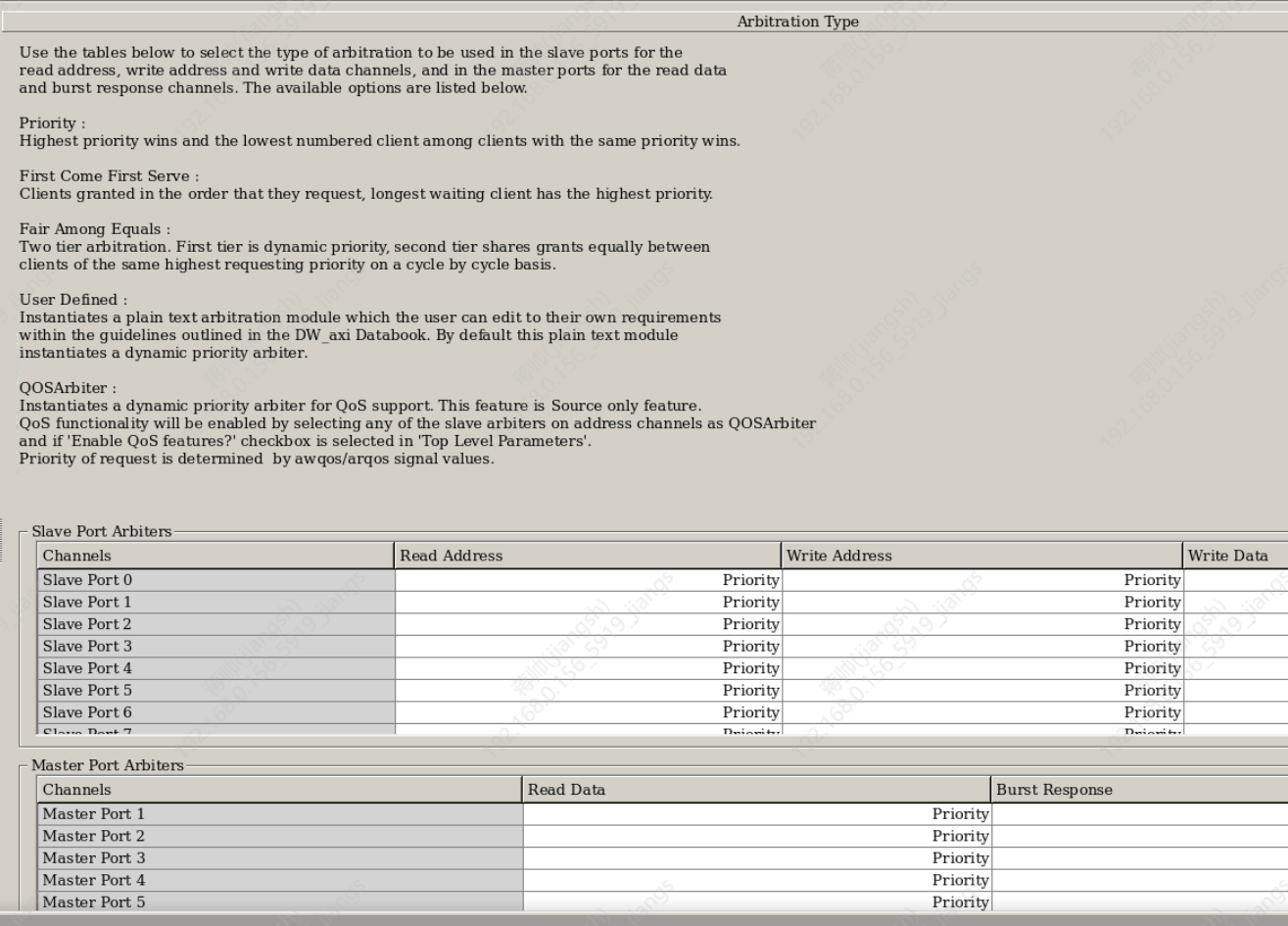


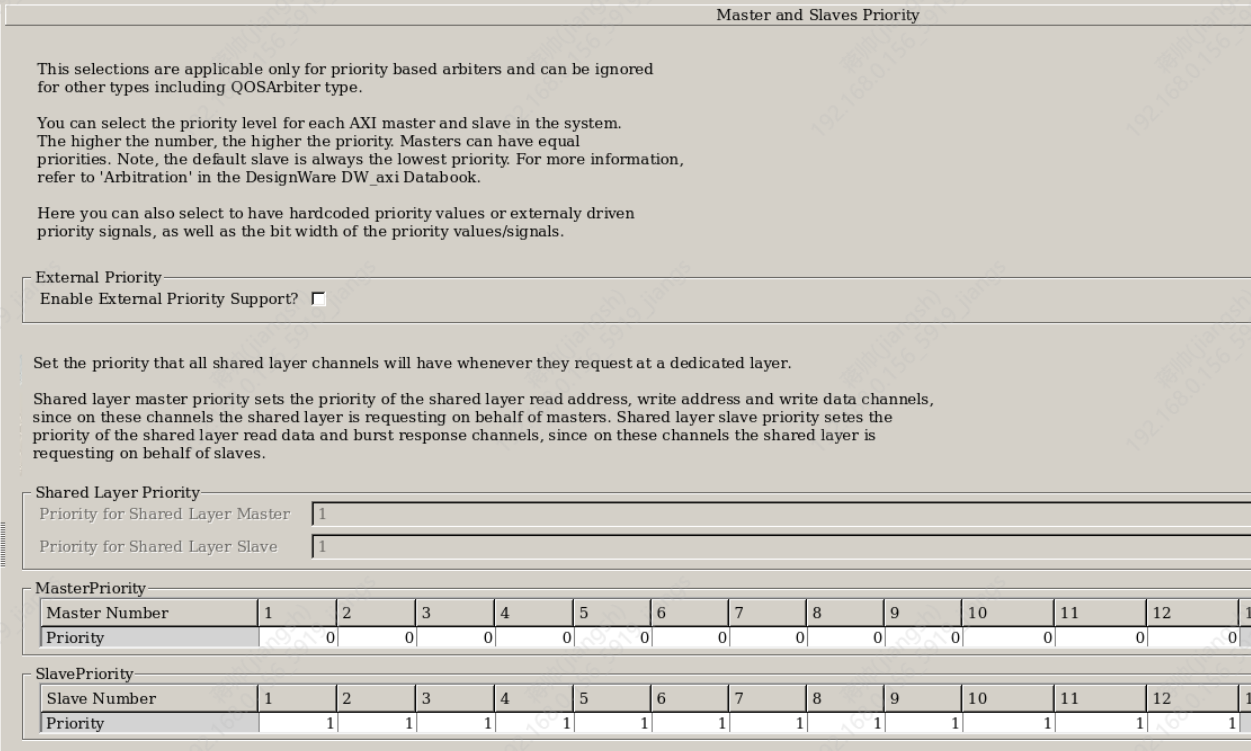


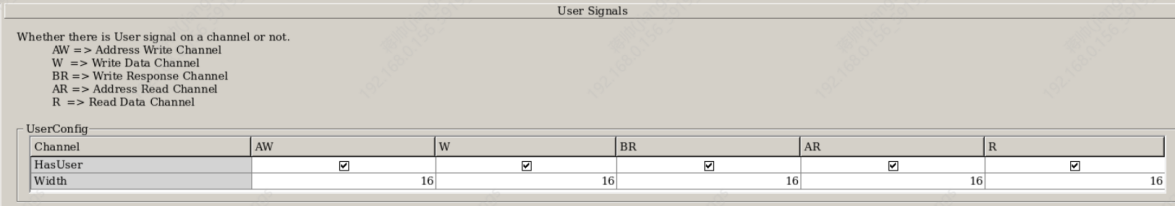


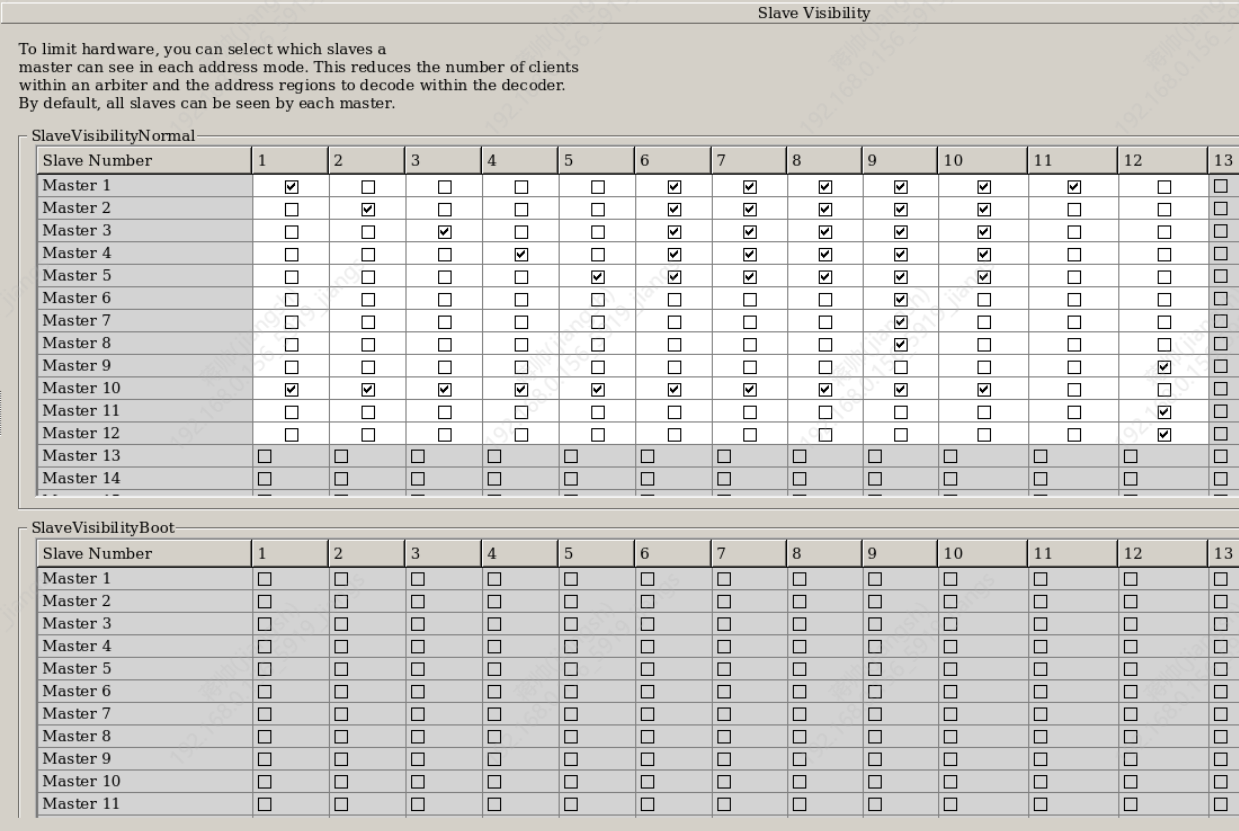


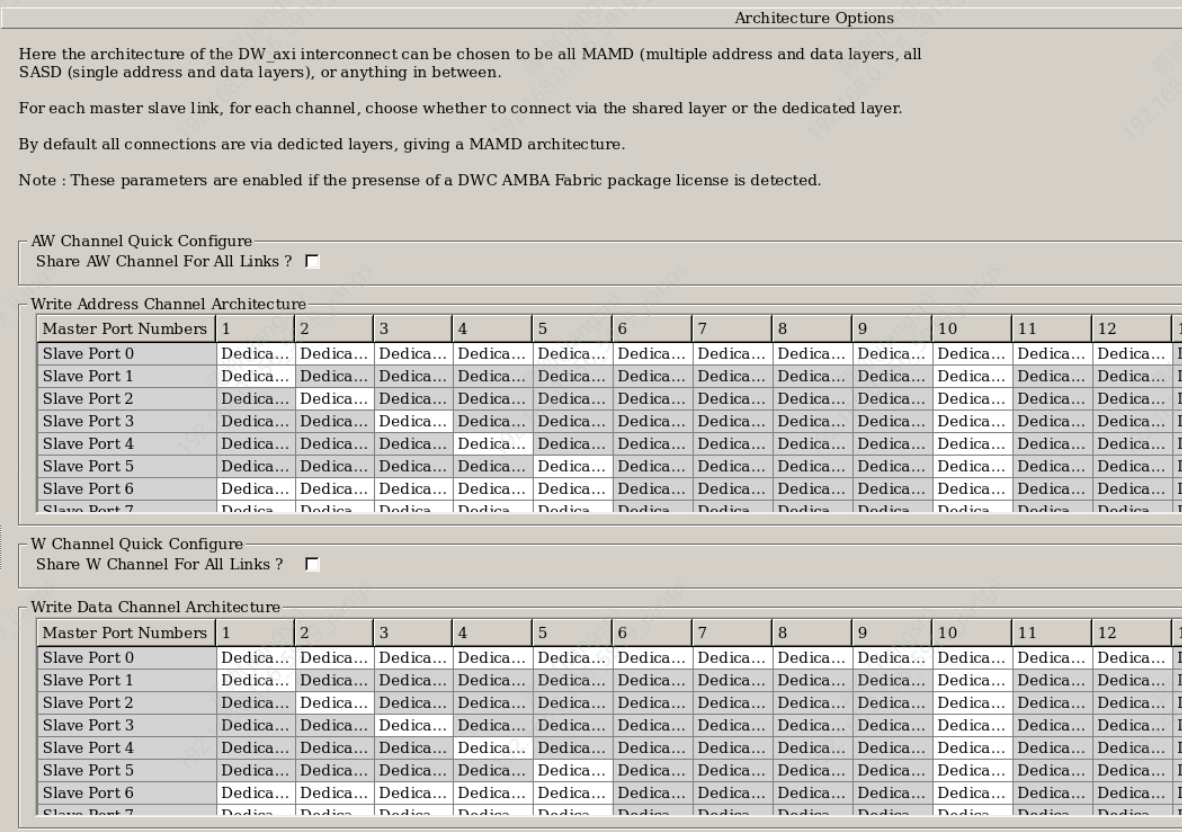


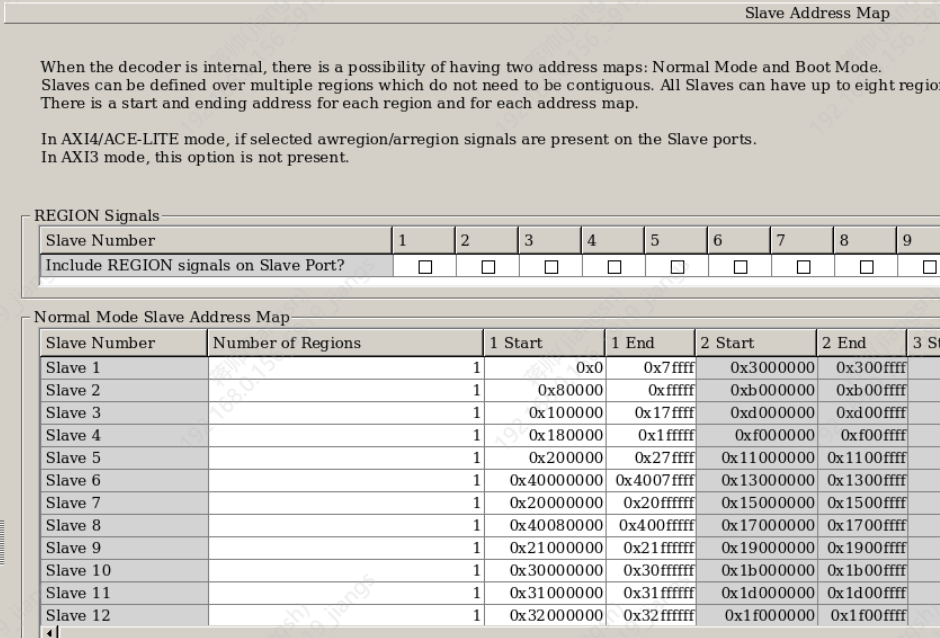


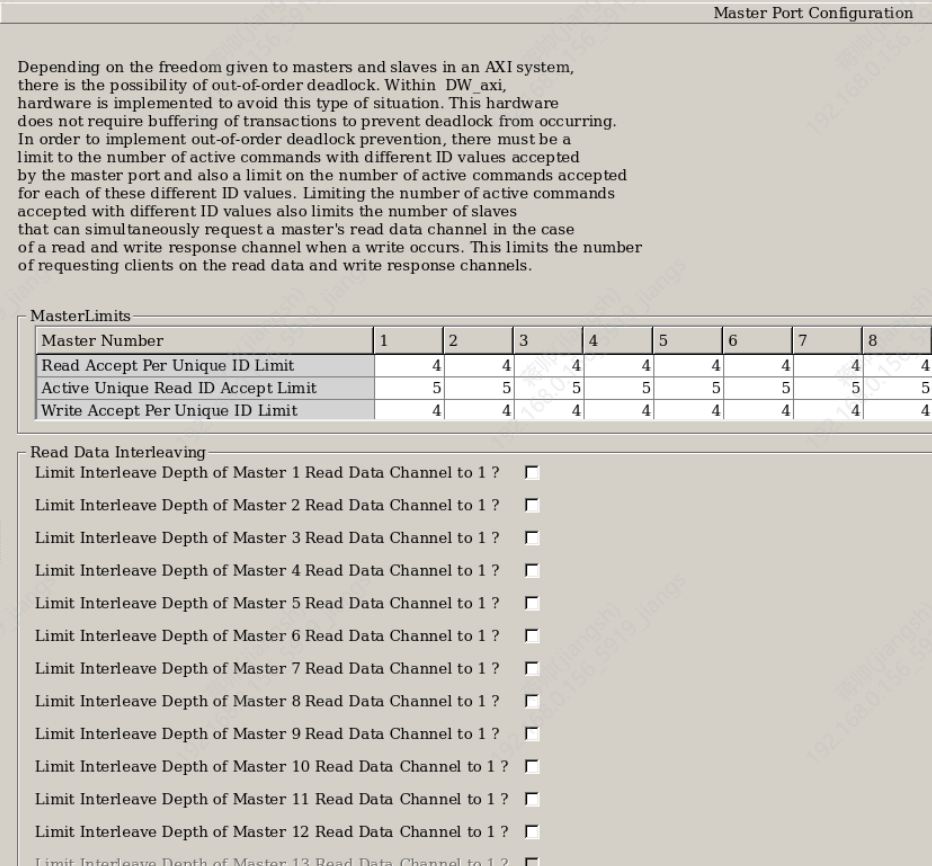


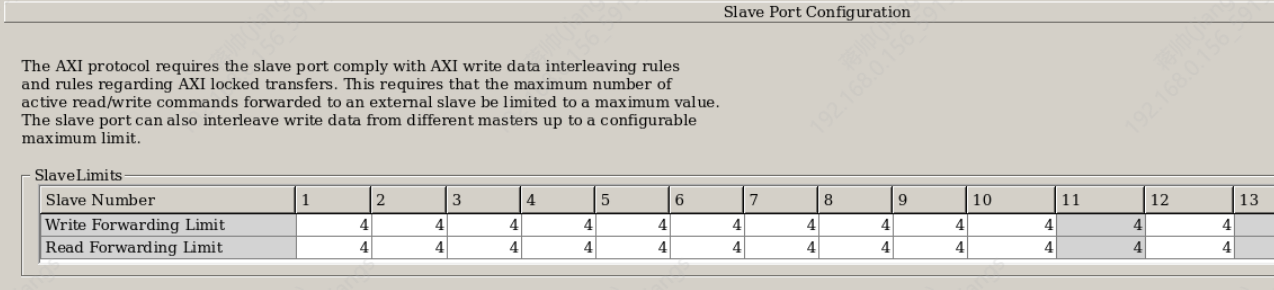












## 模块接口时序