FAST 交换机设计文档

（版本：0）

FAST 技术开发团队

2016 年 10 月

目录

1 简介

FAST（FPGA bAsed SDN swiTching）是一种基于 FPGA 的可重构交换架构——将报文处理流程拆解成多个独立报文处理阶段，每个阶段由一个或者多个模块实现，并分别为每个报文处理阶段建立相应的模块库，来实现灵活的修改、替换等操作。FAST 允许使用者根据实际使用需求，自由选择需要的处理模块，用于快速重构报文处理流水线。开发者要在对 SDN 交换机设计增加新功能时也不再需要重新构建报文处理流程和代码结构，而是按照 FAST 的设计规范在对模块库中增加、选择新的模块，并使用所提供的标准接口与其他模块相连，能够大幅度降低网络应用服务开发的难度和网络设备的开发周期。

FAST 的设计思想是利用 FPGA 的可编程性，将报文处理流水线的每个步骤都变得可重构，具体思路如下：

（1）FAST 架构将报文解析转发流程拆解成多个带有接口的独立功能模块，使用者可以根据实际使用需求，自由选择流水线的每个模块，再对相应的文件进行编译，建立一个可重构的报文处理机制，足以适应需求不一的网络应用场景，提升交换机的灵活性和可扩展性。

（2）FAST 允许开发人员添加新模块，只需要使用标准定义的接口，即可与其他模块连接交互，大大缩短开发周期，适合于时间优先的原型设计方式，能够应对复杂变化的网络协议标准和不断增长的新需求。

（3）由于 FAST 根据功能需求动态的加载不同的处理模块，不会存在模块运行冗余或利用率低下的问题，可以有效降低设备功耗，延长使用周期。

如何增强转发平面的灵活性和可扩展性，关键在于对功能库中模块的自由选择，而按照标准扩建模块库，正是建立开源社区的意义所在。在 FAST 的开源过程中，不仅仅是公开代码和设计文档，而是既能将现有工作进行整合复用，也能给开源数据平面设计提供技术支持，推而广之，FAST 实际代表了一种规范化的 FPGA 交换机的开发模式。

2 系统架构

FAST 是一个高度模块化，低耦合性，具备灵活多变的组织形式。软件架构主要包括 FAST 核心模块部分与底层平台相关的适配部分；硬件同样是包括 FAST 核心的硬件模块与平台相关的适配模块。FAST 的软硬件核心模块抽象出了与平台相关的公共接口，不同的硬件平台根据此抽象接口实现所需功能，即可与 FAST 核心模块进行通信，组成一个完整的系统解决方案。

2.1 软件架构

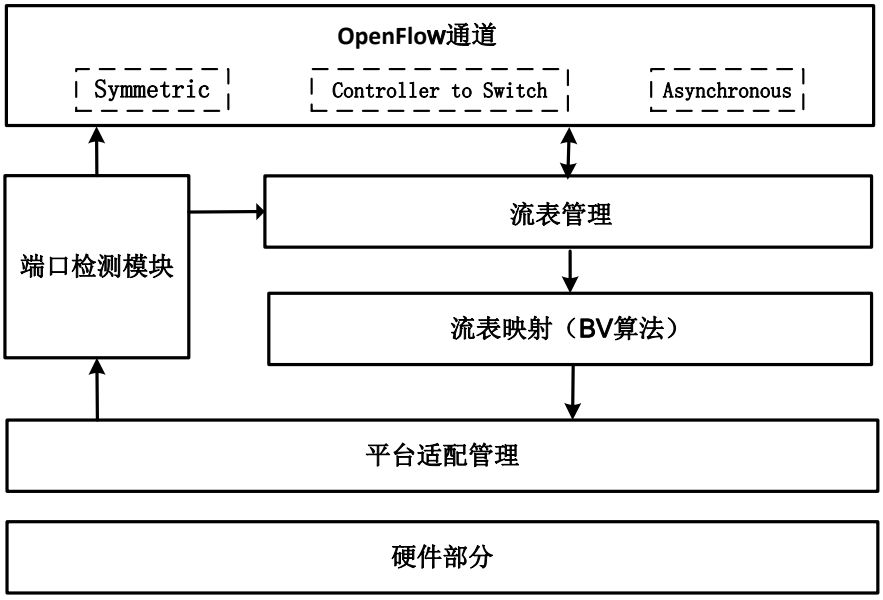


图 2.1FAST 软件架构图

FAST 软件部分主要包括 OpenFlow 通道、端口监测模块、流表管理模块、流表映射模块（BV 算法）、NPE 驱动。

1. OpenFlow_Protocol (OFP 模块)

主要负责与 Floodlight 控制器建立连接，传递流表，流量统计信息和未命中报文通过 OpenFlow 通道上传。

2. 链路状态探测模块

对各端口链路状态进行实时探测。发现链路状态改变后将探测到的链路信息通过 OFP_PORT_STAUS 消息上报至控制器。

3. 流表管理模块

此模块用来动态维护控制器下发给交换机的流规则信息，并实时把新添加的流规则或者更新的流规则经过流表映射 BV 算法，NPE 驱动写入到硬件 RAM 中，硬件根据表项准发，处理报文；另外此模块还提供删除流的接口，用来支持删除此模块中的流规则信息以及调用 bv 接口删除硬件中对应的流规则信息。

4. 流表映射模块（BV 算法）

此模块是将上面介绍的两个模块中的表项以及接收到的命令等分割成多个部分，通过 BV 算法进行映射后，在配置到硬件当中去。这样，每部分与对应的规则进行查找可以实现并行运算，以提高查表效率。

5. 平台适配管理

此模块主要负责对接下面不同的硬件平台，实现对应平台的数据收发，状态管理及其他接口实现。

2.2 硬件架构

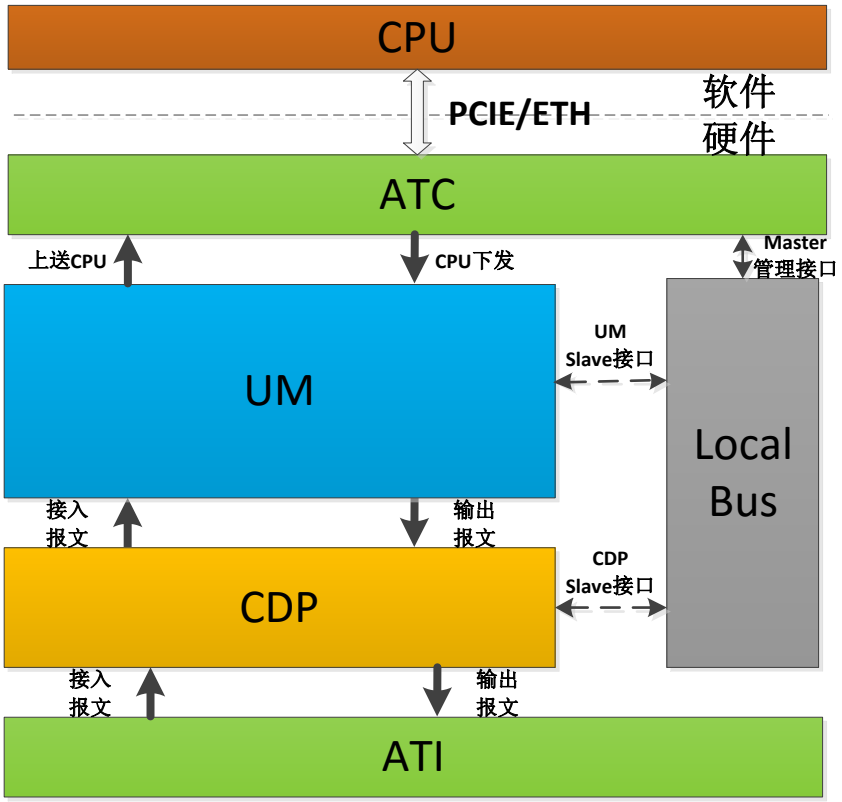


图 2.2FAST 硬件结构

FAST 根据功能和应用需求的区别，将硬件逻辑分为了如上图所示，其中各组件的主要功能为：

- ATC：Adapter To CPU，针对不同平台适配通往 CPU 的接口。带 CPU 的 NetMagic Pro 平台可使用 PCIE_DMA 模块，不带 CPU 的 NetMagic 08 可使用封装以太网从管理口输出到上层控制器的 CPU。
- CAB：管理组件，采用 Localbus 协议，管理硬件地址空间，解析软件的控制信息并分发至硬件各个部分。
- CDP：数据接入组件，主要管理所有硬件物理接口，汇聚从端口接收的报文，同时将处理完成的报文报文分发至各指定端口。
- UM：用户自定义组件，包含丰富的硬件模块库，用户可自行选择需要的模块定义内部功能。
- ATI：Adapter To Interface，针对不同硬件适配平台的数据 I/O 通路。

3 基础模块库

FAST 的基础模块库分为软件模块库与硬件模块库两部分。其中软件模块库包括 FAST 构建、数据收发、流表管理、计数管理、端口监测和 OFP 通道等模块库；硬件模块库包括报文解析、关键字提取、查表、动作执行、公共数据通路和用户编程模块。

3.1 软件模块库

软件模块库的主要功能是完成上层软件系统构建，根据用户配置完成硬件模块关联，并实现对应硬件平台与软件接口之间的耦合。

3.1.1 FAST 构建

FAST 构建模块主要负责建立起一个 FAST 标准的框架工程，将软硬件所必须的标准模块库进行组装。在用户构建 FAS 工程时，用户需要选择底层硬件使用哪款平台进行适配，上层软件选用哪些软件模块和应用来进行组装。最后，项目创建工具根据工程配置文件的设置，将所需要的软硬件代码进行拼装，生成对应的硬件项目代码与软件项目代码。用户分别编程软硬件工程代码即可完成整个 FAST 项目的构建。

3.1.2 流表管理

3.1.2.1OFP 底层流表管理

此模块的主要功能是动态维护控制器下发的流规则信息，实现对控制器下发给交换机的流表进行统一管理，提供后续对硬件进行流规则的添加、老化、更新操作，以及提供流信息来响应控制器的流统计请求。

1. 添加流表
   - 接口函数：void add_flow(struct _flow_table flow_table[],u8 *data,u32 key,u32 inport,u32 outport,u32 datalength)

   参数说明：
   - flow_table[]：指向本地流表的结构体指针；
   - data：指向 match 域的 u8*指针；
   - key：match 域+acting 动作的 MD5 值
   - inport：输入端口号
   - outport：输出端口号
   - datalength：data 的有效数据长度
   - 功能：向本地流表添加一条新的流规则，同时调用 BV 接口把这条规则下发到硬件
2. 删除流表
   - 接口函数：void del_flow_by_port(u32 port)

   参数说明：
   - port：探测到 linkdown 的端口号；
   - 功能：根据端口号删除本地流表中与此端口号相关的所有流规则信息，同时调用 BV 接口删除硬件中与此端口号相关的所有流规则信息
   - 接口函数：voiddel_a_flow(u32 key)

   参数说明：
   - key：match 域+acting 动作的 MD5 值
   - 功能：删除本地流表中指定的流规则，同时调用 BV 接口把硬件中

的这条流规则也删除掉

3. 更新流表

- 接口函数：void update_flow(struct _flow_table flow_table[],u8 *data,u32 key,u32 inport,u32 outport,u32 datalength)

参数说明：

- flow_table[]：指向本地流表的结构体指针；
- data：指向 match 域的 u8*指针；
- key：match 域+acting 动作的 MD5 值
- inport：输入端口号
- outport：输出端口号
- datalength：data 的有效数据长度
- 功能：向本地流表更新一条已有的流规则，同时调用 BV 接口把这条更新规则下发到硬件

3.1.2.2BV 算法流表管理

1. 增加规则

- 函数原型：add_rule(struct fast_table *fast)
- 功能：添加规则
- 返回值：若存在此规则，则返回 0；若添加了这条规则，则返回此次添加规则的行数

2. 删除规则

- 函数原型：delete_rule(struct fast_table *fast)
- 功能：删除规则
- 返回值：删除成功，返回 1；否则返回 0（没有找到相应的规则）

3.1.2.3添加规则接口模块

1. 流程图描述

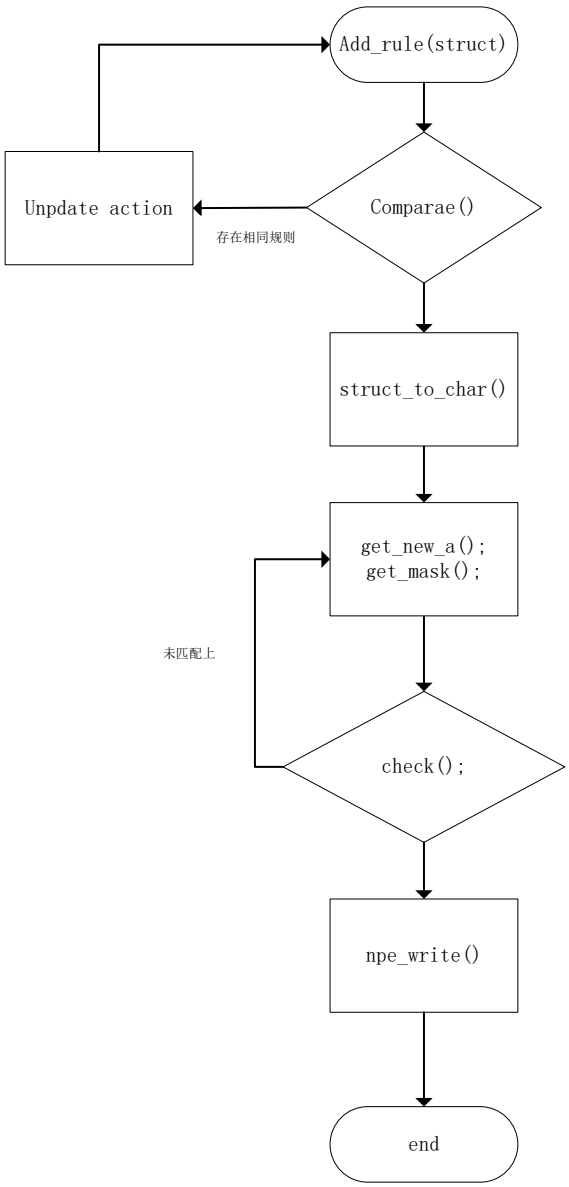


图 3.1 添加规则流程图

2. 比较模块

当添加一条新的规则时，需要通过 compare()函数进行比较，如果已经存在此规则，则只需要更新 action，若不存在此规则，那么就进行规则写入。

3. 规则结构体转换模块

通过函数 struct_to_char(struct fast_table *fast)将收到的结构体转化成为字符串

然后通过 get_new_a(char *rule)与 get_new_mask(char *rule)将此字符串进一步转化成为 32*sizeof(int)大小的规则数组 rule[32]与掩码数组 mask[32]。

4. 地址空间建立模块

为了实现 BV 算法查表，需要建立一个 0~511 的地址段，每一个地址都存放着 最终的 BV 向量。

5. 匹配模块

调用 check()对规则数组，掩码数组和地址空间的 512 个地址通过(rule[i]^addr[i])&mask[i]进行匹配,匹配上了就把地址空间里存入的 BV 向量增加 pow(2,row),其中 row 代表现在正添加规则的行数；

6. 数据结构

```
struct fast_table {
        Struct sw_flow *sw_flow_rule;
        Struct sw_flow *sw_flow_mask;
};
Struct sw_flow{
    Struct{
        U_int8_t src[ETH_ALEN];
        U_int8_t dst[ETH_ALEN];
        U_int16_t type;
    }eth;
    Struct{
        U_int32_t src;
        U_int32_t dst;
        U_int8_t proto;
    }ip;
    Struct{
        U_int16_t src;
        U_int16_t dst;
    }tp;
    U_int8_t in_port;
    U_int32_t priority;
    Struct{
        U_int32_t actions;
    }action;
};
```

3.1.4.3 删除规则接口模块

1. 流程图

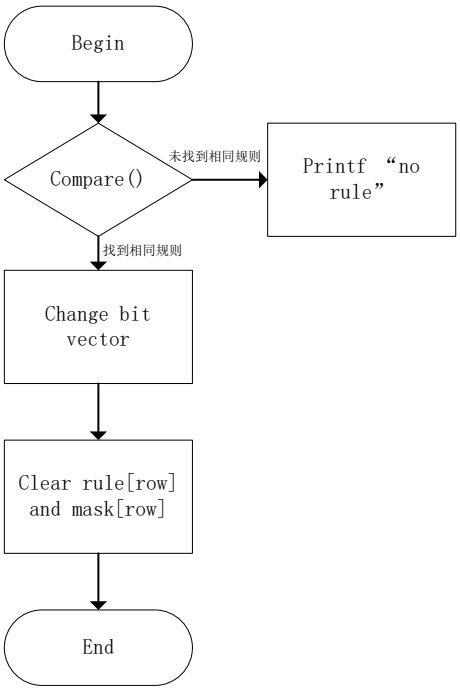


图 3.2 删除规则流程图

2. 查找规则模块

通过 compare()函数，如果没找到规则，则返回 0；若找到了，则返回此规则所在的行数。

3. 删除

找到行数之后，取出之前存好的这条规则 rule 数组和 mask 数组，通过 check()函数进行匹配，匹配上的，对相信的内存处 BV 减少 pow(2,row)，其中 row 表示此规则的行数。

4. 清空规则

将 rule 数组与 mask 数组清零。

3.1.3 计数管理

3.1.4 端口状态监测

此模块主要作用是监听端口的链路状态，监听端口 UP/DOWN，十百千兆连接方式等。一旦发现端口状态发生改变，调用 OFP 模块中 OFP_PORT_STATUS

消息发送发送异步状态信息到控制器。

1. 探测端口状态
   - 接口函数：U32 detect_port_status(u32 port)
   - 功能：探测端口状态，u32 端口号，返回一个 u32 位的端口状态参数。其中第 15 位代表端口 UP、DOWN 状态，第 10 位 11 位 10Mbps、100Mbps、1Gbps 状态。
2. 端口状态监测线程
   - 接口函数：Pthread_t port_status_detection();
   - 功能：每隔 1 秒循环探测一次各个端口状态，如发现有改变则上报调用 send_port_states_message()接口函数，上报改变的信息。实现控制器的拓扑展示的功能。

3.1.5 OFP 通道

1. 通用接口
   1) 打印报文
      接口函数：void pkt_print(u8* pkt, int len)
   2) 处理总的 OpenFlow 消息接口
   - 接口函数：static enum ofperr handle_openflow(struct ofp_buffer *ofpbuf,int len)
   - 功能：将接收到的存在 ofpbuf 中的报文，根据 TYPE 类型进行筛选，进行对应类型函数的处理。参数 len 为 ofp 消息长度，返回对应的错误类型。
2. OpenFlow 各类消息接口
   1) 处理 Hello 消息接口
   - 接口函数：static enum ofperr handle_hello(struct ofp_buffer *ofpbuf)
   - 功能：处理 Hello 消息，判断是否为 OpenFlow1.3 协议，若不是则返回错误，断开连接。
   2) 处理 Error 消息接口
   - 接口函数：static enum ofperr handle_error(struct ofp_buffer *ofpbuf)
   - 功能：相对应的 ERROR 消息。
   3) 处理 Echo 消息接口

- 接口函数：static enum ofperr handle_echo_request(struct ofp_buffer *ofpbuf)
- 功能：对控制器发来的 OFPT_ECHO_REQUEST 报文，回应 OFPT_ECHO_REPLY 报文，保持连接。

4) 处理 Features 消息接口

- 接口函数：static enum ofperr handle_features_request(struct ofp_buffer *ofpbuf)
- 功能：处理 OFPT_FEATURES_REQUEST 消息，回复 OFPT_FEATURES_REPLY 其中，uint32_t capabilities 字段，告知控制器是否具备：FLOW_STATS、TALBE_STATS、PORT_STATS、GROUP_STATS、IP_REASM、QUEUE_STATS、PORT_BLOCKED 等功能。

5) 处理 Get_config 消息接口

- 接口函数：Static enum ofperrhandle_get_config_request(struct ofp_buffer *ofpbuf)
- 功能：获取交换机 config 信息，回复 OFPT_GET_CONFIG_REPLY 告知控制器。

6) 处理 Set_config 消息接口

- 接口函数：static enum ofperr handle_set_config(struct ofp_buffer *ofpbuf,int len)
- 功能：配置交换机 config 信息，通常与 barrier 消息和 get_config_request 消息一起发送，以检查是否配置成功。

7) 处理 Packet_out 消息接口

- 接口函数：static enum ofperr handle_packet_out(struct ofp_buffer *ofpbuf)
- 功能：接收 Packet_out 消息，指导某个具体的报文如何转发，以及实现 LLDP。

8) 处理 Flow_mod 消息接口

- 接口函数：static enum ofperr handle_flow_mod(struct ofp_buffer *ofpbuf)

- 功能：通过 FLOW_MOD 消息，接受流表信息，将流表信息存储到流表管理模块，并通过 BV 转化下发到硬件。

9) 处理 Multipart 消息接口

- 接口函数：static enum ofperr handle_multipart_request(struct ofp_buffer *ofpbuf)
- 功能：用于流量统计，及基本信息的获取。

10) 处理 Multipart 子类型消息接口

- 描述交换机：OFPMP_DESC = 0
- 接口函数：static enum ofperr handle_ofpmp_desc(struct ofp_buffer *ofpbuf)
- 单独流统计：OFPMP_FLOW = 1
- 接口函数：static enum ofperr handle_ofpmp_flow_stats(struct ofp_buffer *ofpbuf)
- 总的流统计：OFPMP_AGGREGATE = 2
- 接口函数：static enum ofperr handle_ofpmp_aggregate(struct ofp_buffer *ofpbuf)
- 流表统计：OFPMP_TABLE = 3
- 接口函数：static enum ofperr handle_ofpmp_aggregate(struct ofp_buffer *ofpbuf)
- 端口统计：OFPMP_PORT_STATS = 4
- 接口函数：static enum ofperr handle_ofpmp_port_stats(struct ofp_buffer *ofpbuf)
- 表特征：OFPMP_TABLE_FEATURES = 12
- 接口函数：static enum ofperr handle_ofpmp_table_features(struct ofp_buffer *ofpbuf)
- 端口描述：OFPMP_PORT_DESC = 13
- 接口函数：static enum ofperr handle_ofpmp_port_desc(struct ofp_buffer *ofpbuf)

11) 处理 Barrier 消息接口

- 接口函数：static enum ofperr handle_barrier_request(struct ofp_buffer

*ofpbuf)

12) 处理 Role 消息接口

- 接口函数：static enum ofperr handle_role_request(struct ofp_buffer *ofpbuf)
- 功能：告知控制器角色消息，回复确认。

13) 发送 Hello 消息接口

- 接口函数：void send_hello_message(uint32_t xid)
- 功能：建立完 TCP 连接后向控制器发送 Hello 消息，告知自己版本情况。

14) 发送 Error 消息接口

- 接口函数：void send_error_message(uint32_t xid)
- 功能：根据交换机运行情况，向控制器发送对应的错误消息。

15) 发送 PORT_STATUS 消息接口

- 接口函数：void send_port_status_message(u32 port, u32 state, u32 current_value)
- 功能：报告端口状态改变的异步消息，u32 port 为端口号，u32 state 为 UP、DOWN 状态信息。

16) 发送 Packet_in 消息接口

- 接口函数：void send_packet_in_message(uint32_t xid)
- 功能：当硬件查询流表不匹配后，将不匹配的报文送入软件，软件将控制器将报文封装在 Packet_IN 消息中，上报控制器。触发控制器完成指定操作。

3.2 硬件模块库

硬件模块库是 FAST 流水线模块库的代码，这是实现不同报文处理逻辑的关键，不涉及用什么具体电路元件来实现，是通用且可编程的功能模块，模块使用 FAST 推荐的标准接口与其他模块相连。用户可根据应用环境的改变而重新选择或改良流水线模块的内容，编译出不同的案例，达到可编程网络的需求。

报文进入 FAST 后，一边用 Buffer 存储报文（将报文存入 RAM 表，再将地址作为 Buffer_id 存入 FIFO），一边提取报文的关键字，生成规则信息。规则信

息在报文尾到达前生成完毕，然后动作模块将报文从 Buffer 中取出后根据规则信息对报文进行处理再输出。

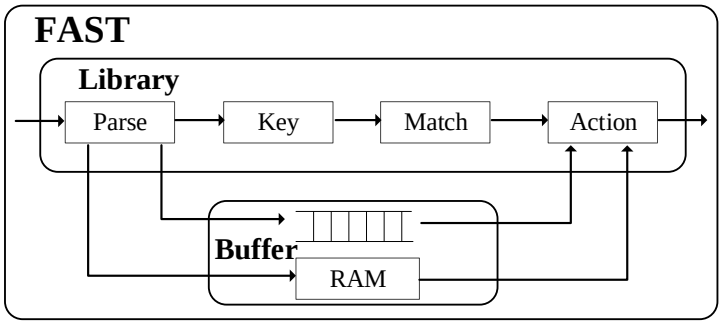


图 3.3FAST 架构

每类功能模块都包含有一个实现库。实现库是相同功能模块的集合。每个模块在 Verilog 代码中都拥有一个.v 文件，当使用者选择某个模块，就编译对应的.v 文件。同时，允许往每个库中增加新模块或修改原有模块以适应开发者的需求。

下图表示 FAST 可重构流水线库目前的规划情况，从每个库中选中所需的功能模块进行编译，即可满足许多应用场景的转发需求。不同的用例可以挑选不同的模块组合成不同的处理流程。Use case 就是 FAST 架构最基本的应用模式案例，代表层叠网中报文的处理流程。

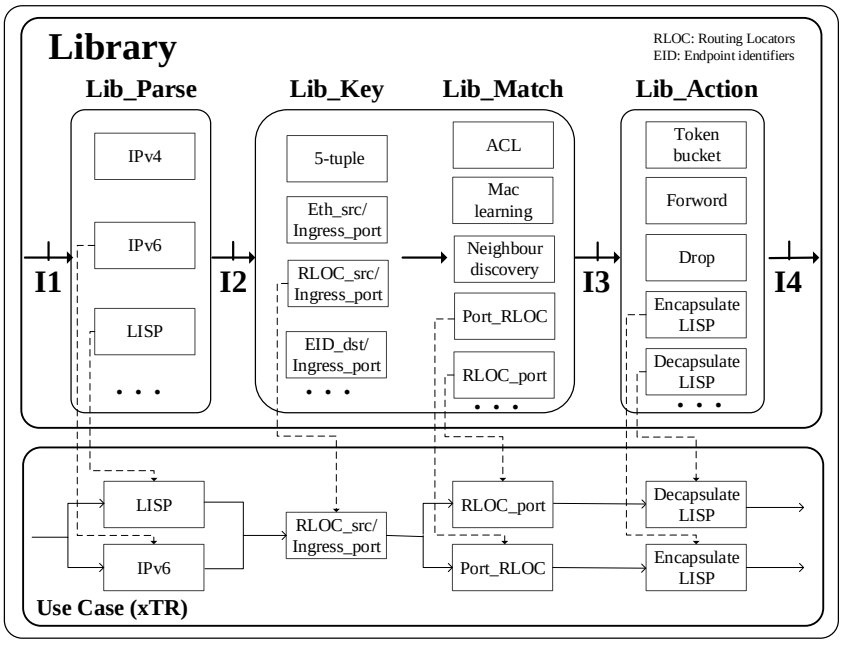


图 3.4FAST 可重构流水线库

3.2.1 报文解析

Parse 代表报文解析模块，负责根据数据包的类型进行解析。据统计，对一

个数据包而言，携带八个或以上的头部信息很常见，不同数据包之间的长度和格式差异很大，必须有针对性的进行识别，层层剥离头部获得报文类型的信息。

目前报文解析模块库已包含有 IPv4、LISP、IPv6 几类解析模块，之后会进行更加细粒度的划分。报文解析模块 Parser 实现报文的九元组解析和提取，并按照指定顺序拼接成查表关键字，同时将解析后的报文转发至执行模块 Executer 进行处理。

3.2.1.1 Parser 模块关键字格式

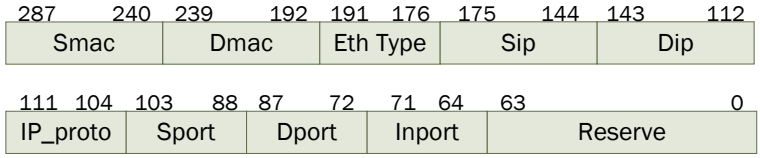


图 3.5 关键字格式

关键字共包含 9 个字段，各字段定义如下表所示：

表 3.1 关键字字段含义

字段名称	位宽(bit)	含义
Smac	48	以太网源 MAC 地址
Dmac	48	以太网目的 MAC 地址
Eth_type	16	以太网帧类型
Sip	32	IPv4 报文源 IP
Dip	32	IPv4 报文目的 IP
IP_proto	8	IPv4 报文协议字段，或者 ARP opcode 的低 8bit
Sport	16	TCP/UDP 源端口号
Dport	16	TCP/UDP 目的端口号
Inport	8	报文输入端口号
Reserve	64	保留位，默认所有 bit 位全为 1
共计	288	

3.2.1.2 Parser 模块接口定义

表 3.2Parser 接口定义

信号名称	方向	位宽	说明
Port2parser_data_wr	In	1	CDP 端口送往 Parser 模块的报文数据写使能
Port2parser_data	In	134	CDP 端口送往 Parser 模块的报文数据
Port2parser_valid_wr	In	1	CDP 端口送往 Parser 模块的报文有效写使能，在每次报文传输随着报文尾部置高

信号名称	方向	位宽	说明
Port2parser_valid	In	134	CDP 端口送往 Parser 模块的报文有效标志。 0：传输报文有错误，是无效报文 1：传输报文完整无误
Parser2port_alf	Out	1	Parser 模块给 CDP 端口的 almostfull 信号。 0：Parser 模块目前无法接收来自 CDP 端口的数据 1：Parser 模块目前可以接收来自 CDP 端口的数据
Parser2lookup_key_wr	Out	1	Parser 模块解析并送往 Lookup 模块的关键字写使能
Parser2lookup_ key	Out	134	Parser 模块解析并送往 Lookup 模块的关键字数据
Parser2next_data_wr	Out	1	Parser 模块送往下一个模块（Executer）的报文数据写使能
Parser2next_data	Out	134	Parser 模块送往下一个模块（Executer）的报文数据
Next2parser_alf	In	1	下一个模块（Executer）给 Parser 模块的 almostfull 信号。 0：下一个模块（Executer）目前无法接收来自 Parser 模块的数据 1：下一个模块（Executer）目前可以接收来自 Parser 模块的数据

3.2.1.3 Parser 模块实现细节

Parser 模块采用流水线的方式提取九元组，并按照格式组织成 Key。其主要实现方式如下所示。

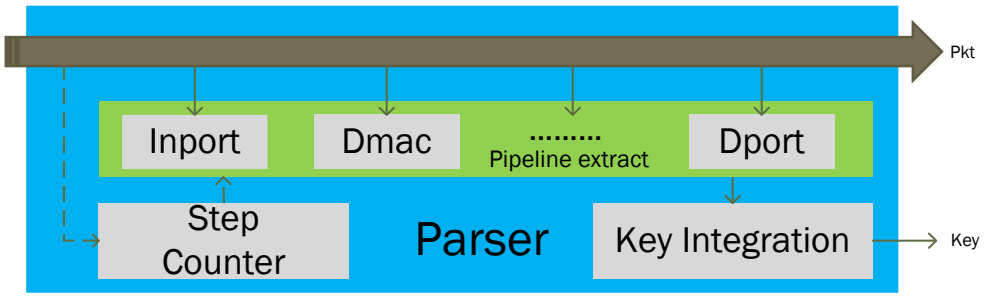


图 3.6Parser 模块功能结构

1. Step_Counter

Step_Counter 是一个 8bit 计数器，每次接收输入报文头部时，将计数器清零，并随着报文行的输入进行累加。。

通过对输入 Parser 模块的报文的每一行（报文数据在 UM 内部按照 128bit 为 1 行进行传输，具体可参照 4.1.1 节）进行计数，从而实现对报文的每一行进行编号。后续的流水线中即可参照当前传输的报文行号提取关键字。

2. Pipeline Extract

根据待提取的关键字在报文中所处的位置，统计处于同一报文行的关键字，以此划分流水线层级，当前各关键字段所处报文行如下：

表 3.3 关键字所处行号

字段名称	位宽(bit)	含义	行号
Smac	48	以太网源 MAC 地址	2
Dmac	48	以太网目的 MAC 地址	2
Eth_type	16	以太网帧类型	2
Sip	32	IPv4 报文源 IP	3
Dip	32	IPv4 报文目的 IP	3&4
IP_proto	8	IPv4 报文协议字段，或者 ARP opcode 的低 8bit	3
Sport	16	TCP/UDP 源端口号	4
Dport	16	TCP/UDP 目的端口号	4
Inport	8	报文输入端口号	0
Reserve	64	保留位，默认所有 bit 位全为 1	/
共计	288		

3. Key Integration

当报文传输到尾部最后一行时，Key Integration 开始对流水线的所有提取关键字按照 4.2.1 节所示的格式进行封装，并通最后一行一同输出。

在封装的同时，Key Integration 会根据 Eth_type/IP_proto 等字段判断诸如 Sip/Dport 等字段是否有效，并对无效字段在封装 Key 所处的位置进行置 1 处理。

3.2.2 查表（BV）

Lookup 主要进行根据软件定义，对报文进行查表匹配和规则查询，是进行报文的处理和决定转发方向的核心功能。

- 采用 BV 算法对关键字进行查表
- 根据查表返回的结果进行对比
  - 命中则输出对应规则 rule
  - 未命中则输出默认规则 rule

- 由于 BV 算法和 Rule 管理采用了不同的配置协议，因此接口各自独立，后期会考虑统一

3.2.2.1 Lookup 模块接口定义

表 3.4Lookup 接口定义

信号名称	方向	位宽	说明
Cfg2entry_cs_n	In	1	User_mgmt_slave 模块给 Lookup 模块表项配置的片选信号，低有效
Cfg2entry_rd_wr	In	1	User_mgmt_slave 模块给 Lookup 模块表项配置读写选择信号 0：写 1：读
Cfg2entry_data	In	32	User_mgmt_slave 模块给 Lookup 模块表项配置的写数据
Cfg2entry_ale	In	1	User_mgmt_slave 模块给 Lookup 模块表项配置的地址锁存信号，高有效
Cfg2entry_ack_n	Out	1	Lookup 模块返回给 User_mgmt_slave 模块的表项配置响应信号，低有效 与 Cfg2entry_cs_n 共同构成握手机制
Entry2cfg_data_out	Out	32	Lookup 模块返回给 User_mgmt_slave 模块的表项配置读请求响应数据
Parser2lookup_key_wr	In	1	Parser 模块解析并送往 Lookup 模块的关键字写使能
Parser2lookup_ key	In	134	Parser 模块解析并送往 Lookup 模块的关键字数据
Cfg2rule_cs_n	In	1	User_mgmt_slave 模块给 Lookup 模块 Rule 配置的片选信号，低有效
Cfg2rule_rd_wr	In	1	User_mgmt_slave 模块给 Lookup 模块 Rule 配置读写选择信号 0：读 1：写
Cfg2rule_data	In	32	User_mgmt_slave 模块给 Lookup 模块 Rule 配置的写数据
Cfg2rule_ale	In	1	User_mgmt_slave 模块给 Lookup 模块 Rule 配置的地址锁存信号，高有效
Cfg2rule_ack_n	Out	1	Lookup 模块返回给 User_mgmt_slave 模块的 Rule 配置响应信号，低有效 与 Cfg2rule_cs_n 共同构成握手机制
Rule2cfg_data_out	Out	32	Lookup 模块返回给 User_mgmt_slave 模块的 Rule 配置读请求响应数据
Lookup2exe_rule_wr	Out	1	Lookup 模块查表完成后送往 executer 模块

信号名称	方向	位宽	说明
			的规则写使能
Lookup2exe_rule	Out	32	Lookup 模块查表完成后送往 executer 模块的规则数据

3.2.2.2 Lookup 模块数据格式

1. 输入关键字

参见上述查表关键字格式。

2. 查表规则

图 3.7 报文控制信息格式

说明：

- 报文控制[31:29]：

  0：丢弃

  1：指定线程上送 CPU

  2：轮询分撒上送 CPU

  3：端口转发

  其余保留

- 保留位置[28:8]
- 输出端口[7:0]：
  - 端口转发时为输出端口
  - 指定线程上送 CPU 时为 CPU 线程号

3.2.2.3 BV_Search_Engine 实现细节

1. BV 搜索整体框架

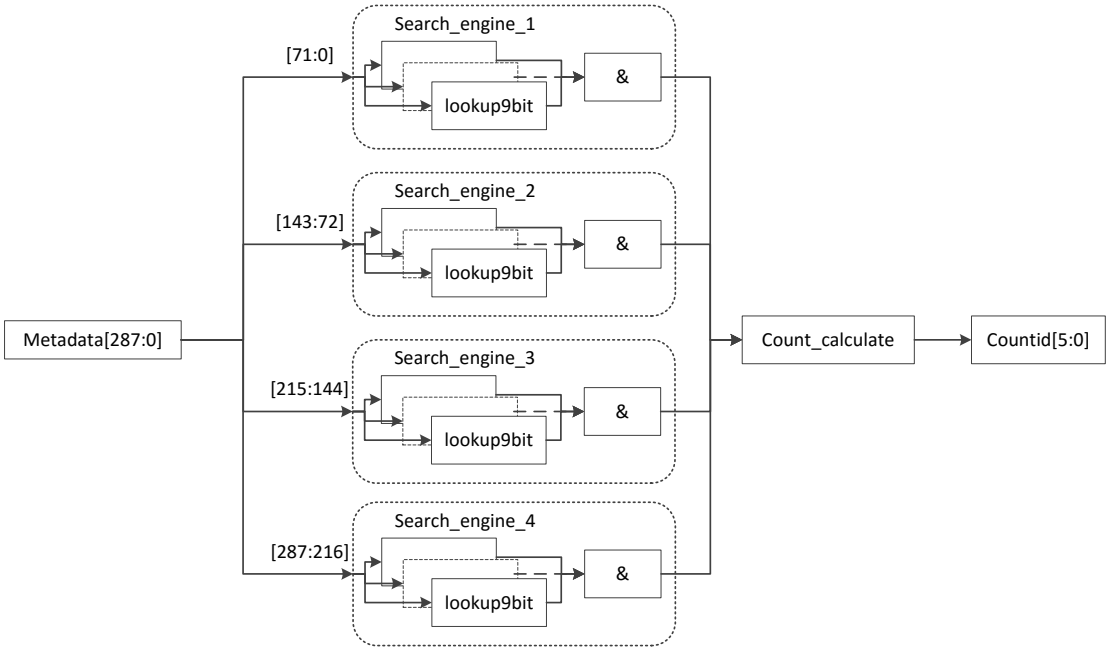


图 3.8BV 搜索整体框架图

BV 模块主要实现了 288bit 的任意匹配查找，用户通过主控 CPU 配置表项和结果，根据报文输入的关键字信息（metadata），输出相应匹配的规则 id（countid）。

2. BV 接口定义

表 3.5BV 接口信号定义

信号名称	方向	位宽	说明
Localbus			
localbus_cs_n	In	1	Localbus 总线片选信号，低有效
localbus_rd_wr	In	1	Localbus 总线读写选择信号 0：写 1：读
localbus_data	In	32	Localbus 总线写数据
localbus_ale	In	1	Localbus 总线地址锁存信号，高有效
localbus_ack_n	Out	1	UM 返回给 Localbus 总线的响应信号，低有效 与 localbus_cs_n 共同构成握手机制
localbus_data_out	Out	32	UM 返回给 Localbus 总线的读请求响应数据
metadata 接口			
Metadata_valid	In	1	查找关键字使能
Metadata	In	288	查找关键字信息
Countid 接口			
Counted_valid	In	1	匹配规则 id 写使能
Counted	In	6	匹配规则 id

3.2.2.4 Rule_Access 实现细节

Rule_Access 子模块主要实现查表命中的规则存储和管理，同时对规则配置和读取进行调度。

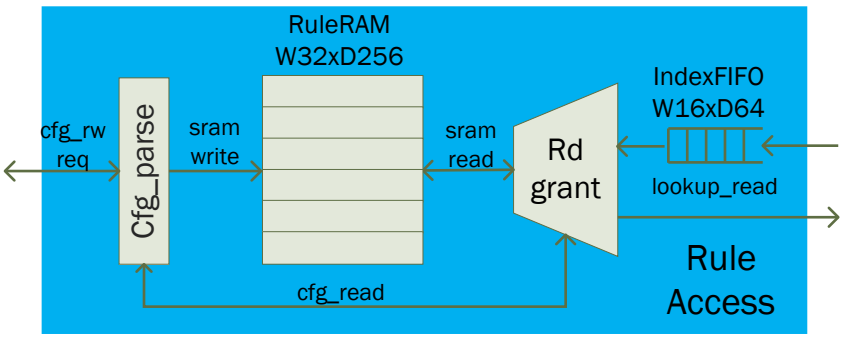


图 3.9Rule_Access 子模块功能结构

其主要有 Cfg_parse、Rd_grant 模块组成，使用了 RAM 缓存规则，同时，在增加 FIFO 缓存查表命中后的规则索引 index，以实现高性能流水化读取规则。

1. Cfg_parse

按照 Localbus 协议，解析从 User_mgmt_slave 模块送入的表项配置请求，将其分为写请求和读请求两类。

由于实现需求中，只有表项配置时才需要进行写请求，因此 Cfg_parse 解析的 RAM 写请求可独占 RAM 写端口。

而读请求则需要与查表命中后的规则索引竞争 RAM 读端口。并在竞争成功后将读回的数据通过 Localbus 协议返回 User_mgmt_slave 模块。

2. Rd_grant

仲裁来自配置和查表两个方向的 RAM 读请求。其仲裁原则是配置请求优先。

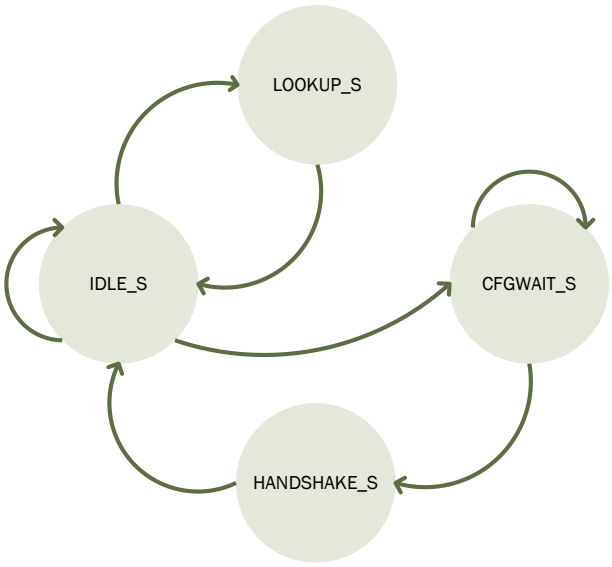


图 3.10Rd_grant 状态跳转

- IDLE_S: 判断来自查表匹配和表项配置两个方向的读请求
  - 只要有表项配置读请求，就提取地址，发起 SRAM 读请求，并跳转 CFGWAIT_S 等待完成读请求
  - 如果只有查表匹配的 FIFO 有索引，读出该索引，并跳转到 LOOKUP_S
  - 如果都没有，则继续停留在 IDLE_S
- LOOKUP_S: 利用读出的索引发起 SRAM 读请求，并跳转回 IDLE_S
  - 由于是流水线读取且 RAM 读取延迟固定，可在状态机外部通过延迟读请求的方式得到返回数据，无需在状态机中等待数据返回，
- CFGWAIT_S: 等待读请求数据返回，并跳转至 HANDSHAKE_S 完成此次配置读请求
- HANDSHAKE_S：将返回的数据写入读返回总线，并将 Localbus 的响应信号为有效，等待请求释放后返回 IDLE_S，没有释放前，继续在 HANDSHAKE_S 等待

由于配置请求采用了 Localbus，当读请求未返回时请求一直存在，因此必须等待数据返回并完成握手后才能开始下一次读调度。

3.2.3 动作执行

Action 代表动作执行模块，指示对数据包的处理。一个 Action 代表的动作可以分成几个子模块完成。目前动作执行模块库已包含有令牌桶处理、LISP 封装、LISP 解封装、丢弃等多种动作处理方式。

动作执行模块根据规则 rule 对报文进行处理，例如：报文输出方向和端口/丢弃/TTL-1/校验和重计算等（目前仅实现这报文输出方向和端口和丢弃这两项）。

3.2.3.1 Executer 模块接口定义

表 3.6Executer 模块接口定义

信号名称	方向	位宽	说明
Parser2exe_data_wr	In	1	Parser 模块送往下一个模块（Executer）的报文数据写使能
Parser2exe_data	In	134	Parser 模块送往下一个模块（Executer）的报文数据
Exe2parser_alf	Out	1	下一个模块（Executer）给 Parser 模块的 almostfull 信号。 0：下一个模块（Executer）目前无法接收来自 Parser 模块的数据 1：下一个模块（Executer）目前可以接收来自 Parser 模块的数据
Lookup2exe_rule_wr	In	1	Lookup 模块查表完成后送往 executer 模块的规则写使能
Lookup2exe_rule	In	32	Lookup 模块查表完成后送往 executer 模块的规则数据
Sys_max_cpuid	In	6	用户指定的 CPU 最大线程，用于 executer 分配线程时作为参考（线程 id 必须小于最大线程）
Exe2disp_direction_req	Out	1	Executer 模块送往 Dispath 模块的报文转发方向请求
Exe2disp_direction	Out	1	Executer 模块送往 Dispath 模块的报文转发方向 0：上送 CPU 1：端口下发
Exe2disp_data_wr	Out	1	executer 模块送往 dispatch 模块的报文数据写使能
Exe2disp_data	Out	134	executer 模块送往 dispatch 模块的报文数据
Exe2disp_valid_wr	Out	1	executer 模块送往 dispatch 模块的报文有效写使能，在每次报文传输随着报文尾部置高
Exe2disp_valid	Out	134	executer 模块送往 dispatch 模块的报文有效标志。 0：传输报文有错误，是无效报文 1：传输报文完整无误
Disp2exe_alf	In	1	dispatch 模块给 executer 模块的 almostfull 信号。 0：dispatch 模块目前无法接收来自 executer 模块的数据 1：dispatch 模块目前可以接收来自 executer 模块的数据

3.2.3.2 Executer 模块实现细节

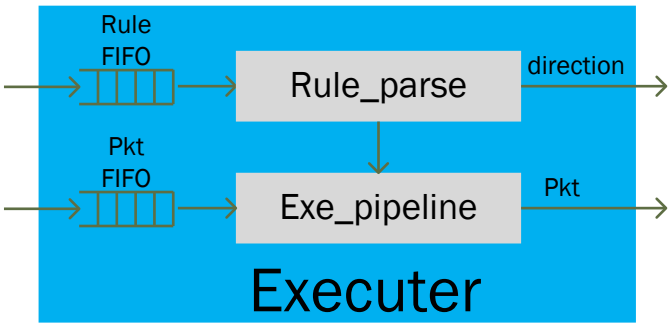


图 3.11Executer 模块功能结构

1. Rule_parse

Rule_parse 子模块主要功能为解析规则中的各处理字段，确认当前报文的转发方向，并构造处理请求，送往 Exe_pipeline 中的各功能块进行报文处理。

目前仅实现了解析并确定报文转发方向。

规则 RuleFIFO 采用 Showaehad 模式，即只要该 FIFO 中缓存有 Rule，无需读出即可看到第一条规则，因此可在报文发送前即可构造报文转发方向信号与下一个模块（dispatch）进行协商。

在报文已经开始发送并且 Dispatch 模块已经开始转发后，读出规则 FIFO 中的规则。并开始判断下一个报文的方向。

2. Pkt Pipeline

目前仅实现报文转发功能，并在报文 metadata 中增加输出端口/线程 ID 等字段。后续会以流水线的形式增加处理模块。

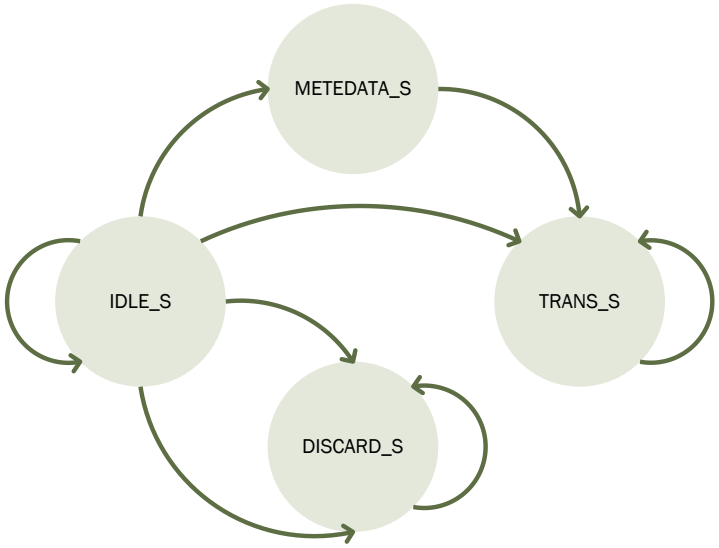


图 3.12Pktpipeline 状态跳转

- IDLE_S: 根据 Rule_parse 提供的请求决定跳转方向
  - 丢弃请求，则从 FIFO 中读出报文，并跳转到 DISCARD_S
  - 转发请求，则从 FIFO 中读出报文，并跳转到 METEDATA_S
  - 如果没有请求，则继续停留在 IDLE_S
- METEDATA_S: 根据 Rule_parse 提供的信息，替换读出的报文 METEDATA 中的线程 ID/输出端口号/接口槽号等字段并输出到下一个模块 Dispatch，然后跳转到 TRANS_S 发送剩下的报文。
- TRANS_S: 读出并转发 FIFO 中的报文，并判断报文中的位置标志
  - 当位置标志为报文尾时，即报文已经完整读出，停止读 FIFO，输出报文尾和有效 Valid 信号，并跳回 IDLE_S
  - 不是报文尾，则继续重复 TRANS_S
- DISCARD_S：读出 FIFO 中的报文，并判断报文中的位置标志
  - 当位置标志为报文尾时，即报文已经完整读出并丢弃，停止读 FIFO 并跳回 IDLE_S
  - 不是报文尾，则继续重复 DISCARD_S

3.2.4 CDP

为有效简化用户自定义报文处理功能的开发实现，FAST 设计了通用数据通路 CDP(Common Data Path)功能，负责供报文的接收、发送、复制等基本通用处理原语。CDP 模块的功能包括网络接口处理、如 GMII 发送和接收、CRC 产生和校验、数据通路报文产生、输入输出缓冲和调度等。通常情况下 CDP 模块的逻辑已经固定，用户无需改变其功能，只需下载并调用这个现成的模块即可。

处理流程包括接收和发送两个阶段，接收阶段包含三个步骤：

1、公共数据路径 CDP 收到命令报文后，检查以太网帧头信息，查看目的 MAC 地址是否为自己的 MAC 地址，若是则进行 CRC 校验；

2、CRC 校验无误后，在 CDP 模块中对报文进行拆封，剥去 MAC 头，查看 IP 头中的协议标识是否为 NMAC 协议，若是则剥去 IP 头，取出 NMAC 命令送管理逻辑 ML 的命令解析模块；

3、ML 中的命令解析模块解析 NMAC 命令，将相应的局部总线时序信号发送到用户自定义模块 UM，完成相应的操作。

发送阶段包含两个步骤：

1、命令在 UM 中完成后，产生返回结果，UM 将这些结果通过局部总线发送到 ML 的命令产生模块，命令产生模块按照协议规定的命令格式对返回结果进行封装，并将封装好的数据送入报文封装模块；

2、在 CDP 模块中对数据进行 IP 封装和 MAC 封装，并生成 CRC，按照物理层接口时序发送报文。

CDP 模块由 Triple Speed Ethernet 核、SGMII_RX、SGMII_TX、SGMII_MUX、SGMII_DMUX、INPUT_CTRL、OUTPUT_CTRL 模块组成，主要实现将外部设备送入的报文解析并转换成硬件规定的报文格式，并进行汇总。

3.2.5 UM

1. UM 简介

UM 模块是 FAST 演示 DEMO 中的用户定义模块，基于 OpenFlow1.0 协议开发，模块内部查表采用 BV（Bit Vector）算法，支持软件定义查表/转发功能。

2. UM 逻辑结构

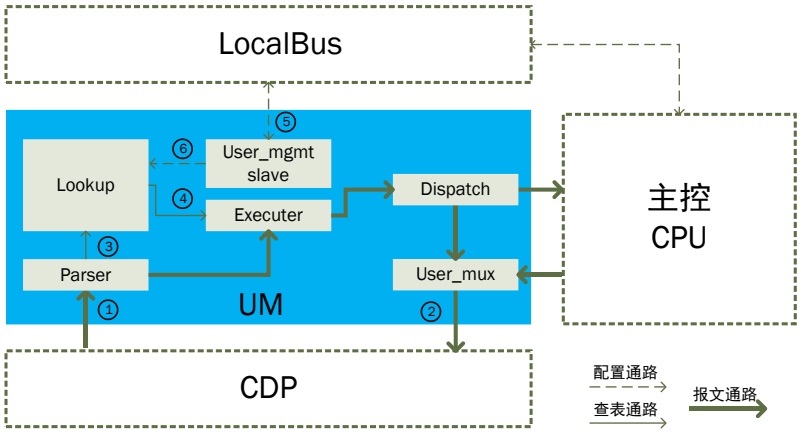


图 3.13UM 逻辑结构

UM 模块主要实现了报文的十三元组查表匹配，用户通过主控 CPU 配置表项和结果，UM 模块则通过查表结果对报文进行处理，同时决定报文的流向（直接从端口转发或上送 CPU）。

3. UM 接口定义

表 3.7UM 接口定义

信号名称	方向	位宽	说明
Localbus			
localbus_cs_n	In	1	Localbus 总线片选信号，低有效
localbus_rd_wr	In	1	Localbus 总线读写选择信号 0：写 1：读
localbus_data	In	32	Localbus 总线写数据
localbus_ale	In	1	Localbus 总线地址锁存信号，高有效
localbus_ack_n	Out	1	UM 返回给 Localbus 总线的响应信号，低有效 与 localbus_cs_n 共同构成握手机制
localbus_data_out	Out	32	UM 返回给 Localbus 总线的读请求响应数据
CDP 接口			
cdp2um_data_wr	In	1	CDP 模块送往 UM 模块的报文数据写使能
cdp2um_data	In	134	CDP 模块送往 UM 模块的报文数据
cdp2um_valid_wr	In	1	CDP 模块送往 UM 模块的报文有效写使能，在每次报文传输随着报文尾部置高
cdp2um_valid	In	1	CDP 模块送往 UM 模块的数据报文有效标志。 0：传输报文有错误，是无效报文 1：传输报文完整无误
um2cdp_alf	out	1	UM 模块给 CPD 的 almostfull 信号。 0：UM 模块目前无法接收来自 CDP 模块的

信号名称	方向	位宽	说明
			数据 1：UM 模块目前可以接收来自 CDP 模块的数据
um2cdp_data_wr	In	1	UM 模块送往 CDP 模块的报文数据写使能
um2cdp_data	In	134	UM 模块送往 CDP 模块的报文数据
um2cdp_valid_wr	In	1	UM 模块送往 CDP 模块的报文有效写使能，在每次报文传输随着报文尾部置高
um2cdp_valid	In	1	UM 模块送往 CDP 模块的数据报文有效标志。 0：传输报文有错误，是无效报文 1：传输报文完整无误
cdp2um_alf	out	1	CDP 模块给 UM 的 almostfull 信号。 0：CDP 模块目前无法接收来自 UM 模块的数据 1：CDP 模块目前可以接收来自 UM 模块的数据
主控 CPU			
npe2um_data_wr	In	1	主控 CPU 送往 UM 模块的报文数据写使能
npe2um_data	In	134	主控 CPU 送往 UM 模块的报文数据
npe2um_valid_wr	In	1	主控 CPU 送往 UM 模块的报文有效写使能
npe2um_valid	In	1	主控 CPU 送往 UM 模块的数据报文有效信号。 0：传输报文有错误，是无效报文 1：传输报文完整无误
um2npe_alf	out	1	UM 模块主控 CPU 给的 almostfull 信号。 0：UM 模块目前无法接收来自主控 CPU 的数据 1：UM 模块目前可以接收来自主控 CPU 的数据
um2npe_data_wr	In	1	UM 模块送往主控 CPU 的报文数据写使能
um2npe_data	In	134	UM 模块送往主控 CPU 的报文数据
um2npe_valid_wr	In	1	UM 模块送往主控 CPU 的报文有效写使能，在每次报文传输随着报文尾部置高
um2npe_valid	In	1	UM 模块送往主控 CPU 的数据报文有效信号。 0：传输报文有错误，是无效报文 1：传输报文完整无误
npe2um_alf	out	1	主控 CPU 给 UM 的 almostfull 信号。 0：主控 CPU 目前无法接收来自 UM 模块的数据 1：主控 CPU 目前可以接收来自 UM 模块的数据

4 平台适配库

4.1 NetMagic Pro

NetMagic Pro 设计平台是基于通用多核处理器+FPGA 架构设计实现。NetMagic Pro 通过搭载 inter i5/i7 CPU，并通过可编程硬件（FPGA）的辅助，通过基于 UM 的软件,硬件框架代码设计,使用户在软件和硬件的开发更方便,更简单。另外，由于 NetMagic Pro 在灵活性及可编程方面的良好折衷，使其具有了灵活的可编程性、并行的数据处理能力、高速的数据转发能力、智能的数据控制能力以及良好的功能扩展能力。

4.1.1 平台简介

NetMagic Pro 为一款开放式设计平台，基于 NetMagic Pro 平台软硬件的灵活可编程性，使其可以支持多种诸如防火墙、虚拟专网、区分服务、基于策略的路由以及 QoS 等功能的实现，同时也可以迎合网络协议的发展变化。

目前本款 NetMagic Pro 可支持快速路由转发、报文分类以及基于传送描述符的 QoS 管理功能等，NetMagic Pro 可实现基于 Run-to-Completion 机制的报文处理以及无中断的报文 DMA 传输处理机制，从而使报文可以有效的进行软硬件的交互处理。

NetMagic Pro 的 FPGA 设计采用分层结构的设计方法，将不同的硬件处理逻辑分层实现通过简单高速的接口信号进行连接以达到高内聚低耦合的设计目标。

1. 平台特点

- 可实现与多款 inter CPU 的适配
- 可实现基于 Run-to-Completion 机制的报文处理
- 可实现基于 Metadata 的报文数据处理
- 可实现无中断的报文 DMA 传输处理
- 可实现基于块的报文描述符缓冲区管理(ABM)
- 可实现基于 localbus 的访问控制(NPE_CAB)

2. 平台组成

- NPE_DMA：主要负责报文链表的构建与回收、DMA 地址池的维护，以及通过报文描述符实现报文的间接保序。
- NPE_PCIE：主要负责 PCIE TLP 报文与 NPE 内部格式报文之间的转换。
- NPE_CAB：主要实现硬件与软件间寄存器的读写、硬件计数器的维护以及控制命令的交换。
- NPE_PPS：报文处理支撑部件，主要实现对接口报文以及 NPE 报文间的格式转换和快速路由转发等功能。
- NPE_IF：接口部件，主要实现接收各端口的报文并拼装成接口报文并对各端口报文进行汇聚。
- UM：主要用于用户进行功能开发。

4.1.2 网络 I/O 接口

4.1.2.1 OUTPUT_CTRL 模块

OUTPUT_CTRL 模块主要接收从上层来的报文，对报文进行解析，根据槽号判断报文的子卡流向。

OUTPUT_CTRL 模块主要实现了两个功能：

- 解析上层报文，判断报文接口子卡流向；
- 统计分别发送到两个接口子卡的报文数；

4.1.2.2 SGMII_DMUX 模块

SGMII_DMUX 模块负责对上层接口下来的报文进行解析，向对应端口分发报文，并对报文计数。

SGMII_DMUX 模块主要实现三个功能：

- 解析从 OUTPUT_CTL 模块接收的报文；
- 剥掉前两拍 metadata 报文头；
- 按照报文头中的端口号将报文发到相应的端口。

4.1.2.3 SGMII_MUX 模块

SGMII_MUX 模块主要实现三个功能：

- 将五个端口（四个 SGMII 接口以及一个 E1 接口）的输入报文合并成一路报文向 INPUT_CTRL 模块转发；
- 按端口对接收报文进行计数；
- 按端口对接收报文添加序列号
- 构造两拍 metadata 数据，并且在其中填充报文的长度，端口号，槽号等信息。

4.1.2.4 INPUT_CTRL 模块

INPUT_CTRL 模块主要实现两个功能：

- 将两个槽的接口子卡的报文合并；
- 对接收的报文进行计数；

4.1.2.5 SGMII_TX 模块

SGMII_TX 模块主要实现的功能：将 134 位位宽的报文解析成 8 位位宽的报文，并输出给 sgmii 核。

4.1.2.6 SGMII_RX 模块

SGMII_RX 模块负责对下层写上来的 8 位位宽的报文拼成 134 位位宽的报文，其中包括 128 位的报文数据、2 位报文位置标识以及 4 位数据无效标识位，并输出给 SGMII_MUX 模块。

4.1.3 上行 CPU 接口

4.1.3.1 UMTOCPU 模块实现的功能

INGRESSTOCPU 模块主要功能是:

- IP 对齐偏移填充。
- 64 字节报文对齐填充。
- 同时根据 UM 模块提供的关键字信息对报文头部进行填充。

4.1.3.2 UMTOCPU 模块连接关系图

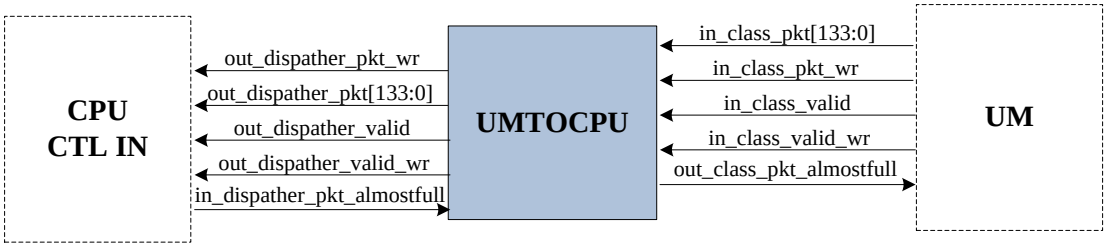


图 4.1UMTOCPU 模块连接关系图

4.1.3.3 UMTOCPU 模块的接口信号

表 4.1UMTOCPU 模块的接口信号

信号名称	宽度	方向	功能
out_dispather_pkt_wr	1	output	输出至 Dispather 模块的报文写信号
out_dispather_pkt	134	output	输出至 Dispather 模块的报文数据
out_dispather_valid_wr	1	output	输出至 Dispather 模块的完整报文标识写信号
out_dispather_valid	1	output	输出至 Dispather 模块的完整报文标识信号
in_dispather_pkt_almostfull	1	input	Dispather 模块输入的缓冲 FIFO 将满标识
in_um_pkt_wr	1	input	UM 模块输入的报文写信号
in_um_pkt	134	input	UM 模块输入的报文数据
in_um_valid_wr	1	input	UM 模块输入的完整报文标识写信号
in_um_valid	1	input	UM 模块输入的完整报文标识信号
out_um_pkt_almostfull	1	output	输出至 UM 模块的缓冲 FIFO 将满标识

4.1.4 CPU 下行接口

4.1.4.1 CPUTOUM 模块实现的功能

- 对各个不同路径的报文进行汇聚，并处理成接口指定的报文格式。
- 对经过软件处理再由硬件从内存中读取后输出的报文按照输出模块定义的报文格式修改，如根据软件指定位置截取报文等，以便输出端口解析并发送报文。

- 对报文 metadata 进行解析，按照不同线程对报文进行计数（计数器模块并不在其中，仅输出计数器更新信号）。

4.1.4.2 CPUTOUM 模块连接关系图

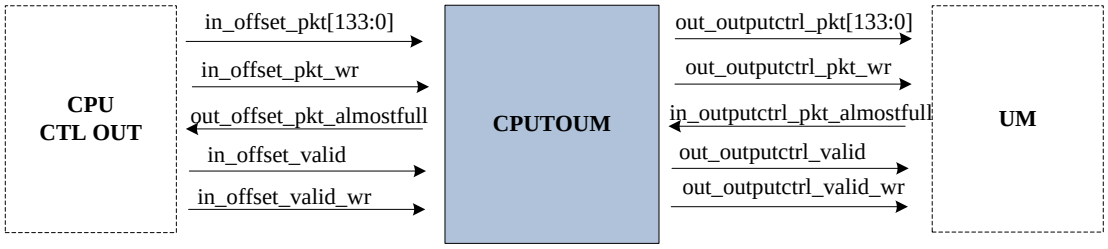


图 4.2EGRESS_CTRL 模块的连接关系图

4.1.4.3 CPUTOUM 模块的接口信号

表 4.2EGRESS_CTRL 模块的接口信号

信号名称	宽度	方向	功能
in_offset_pkt_wr	1	input	EGRESS_OFFSET 模块输入的报文写信号
in_ offset _pkt	134	input	EGRESS_OFFSET 模块输入的报文数据
in_ offset _valid_wr	1	input	EGRESS_OFFSET 模块输入的完整报文标识写信号
in_ offset _valid	1	input	EGRESS_OFFSET 模块输入的完整报文标识信号
out_ offset _pkt_almostfull	1	output	输出至 EGRESS_OFFSET 模块的缓冲 FIFO 将满标识
out_outputctrl_pkt_wr	1	output	输出至 OUTPUT_CTRL 模块的报文写信号
out_ outputctrl _pkt	134	output	输出至 OUTPUT_CTRL 模块的报文数据
out_ outputctrl _valid_wr	1	output	输出至 OUTPUT_CTRL 模块的完整报文标识写信号
out_ outputctrl _valid	1	output	输出至 OUTPUT_CTRL 模块的完整报文标识信号
in_outputctrl _pkt_almostfull	1	input	OUTPUT_CTRL 模块输入的缓冲 FIFO 将满标识
in_fastpath_pkt_wr	1	input	FAST_PATH 模块输入的报文写信号
in_fastpath _pkt	134	input	FAST_PATH 模块输入的报文数据
in_fastpath_valid_wr	1	input	FAST_PATH 模块输入的完整报文标识写信号
in_fastpath_valid	1	input	FAST_PATH 模块输入的完整报文标识信号
out_fastpath_pkt_almostfull	1	output	输出至 FAST_PATH 模块的缓冲 FIFO 将满标识
in_memctrl_pkt_wr	1	input	MEM_CTRL 模块输入的报文写信号
in_ memctrl _pkt	134	input	MEM_CTRL 模块输入的报文数据
in_ memctrl _valid_wr	1	input	MEM_CTRL 模块输入的完整报文标识写信号
in_ memctrl _valid	1	input	MEM_CTRL 模块输入的完整报文标识信号
out_memctrl_pkt_almostfull	1	output	输出至 MEM_CTRL 模块的缓冲 FIFO 将满标识

4.1.5 NPE 驱动

NPE 为 NetMagic Pro 设备的系统驱动，主要为操作系统内核提供标准以太网设备驱动的功能，其向上层应用提供一个标准的以太网设备，可以通过该设备的操作函数实现对网络报文的发送、端口计数、接收模式设置、MAC 地址设置和 MTU 设置等功能；向下则是将上层的调用参数传递给硬件驱动实现，硬件驱动将上层参数的功能要求，转化为硬件可识别、可操作的功能参数，通过底层硬件寄存器操作的方式将参数信息配置到硬件，实现上层应用功能。

4.1.5.1主要接口实现

在驱动中实现标准网络设备，必须申请和注册，而且每个设备都有唯一的网络标识，如接口名称和接口索引等。

1. 申请网络设备
   a) 函数：alloc_etherdev(sizeof(struct npe_adapter))
   b) 参数：sizeof(struct npe_adapter)，网络设备私有数据的空间大小
   c) 返回值：网络设备数据结构 struct net_device
   d) 函数主要功能：此函数只是申请一个标准网络设备的数据结构，并对数据结构中的私有数据做了空间预留，私有数据主要用来保存驱动中对每个网络设备存储的一些驱动内部的变量值。
2. 注册网络设备
   a) 函数：register_netdev(netdev_npe)
   b) 参数：netdev_npe 就是上面申请得到的网络设备数据结构指针
   c) 返回值：0 表示注册成功，非 0 表示注册失败
   d) 函数主要功能：完成了网络设备基本属性赋值的数据结构可以申请注册，系统内核会根据网络设备的相关参数查找是否存在冲突，数据是否合法，相关的操作函数指针是否非空等，所有判断成功后便返回注册成功。

4.1.5.2报文接收函数实现

NPE 驱动在报文处理线程启动只需要提供线程绑定的 CPU 号，线程运行起来后根据 CPU 参数绑定到指定的 CPU 核上独立运行。每个报文处理线程中都有一个报文链的首地址，报文首地址由线程启动时传入的 CPU 号决定，线程根据 CPU 到描述符地址中提取对应的报文地址作为轮询链的首地址。

a) 实现函数：static int npe_recv_poll(void *data)
b) 参数：data 表示线程参数
c) 返回值：0 表示线程执行结束
d) 函数主要功能：轮询报文链，通过 netif_receive_skb(skb)函数将报文送到上层协议栈处理

4.1.5.3网络设备操作函数实现

在标准网络设备接口中，其实只需要实现报文的发送，端口计数器等工作就可以了。一般用户都会根据硬件功能再实现其他一些标准功能和特殊功能，如：设备端口的打开和关闭，设置多个 MAC 地址，端口接收模式设备等。

网络设备的一般操作函数如下：

1. 端口打开接口
   e) 实现函数：int npe_open(struct net_device *dev)
   f) 参数：dev 表示网络设备
   g) 返回值：0 表示成功；非 0 表示错误
   h) 函数主要功能：将当前网络设备对应的物理端口状态置 1，表示端口使能，硬件允许报文从此端口输入与输出，硬件将此端口 LED 灯置为打开状态。启动网络接口队列，允许上层协议发送报文
2. 端口关闭接口
   a) 实现函数：int npe_stop(struct net_device *dev)
   b) 参数：dev 表示网络设备
   c) 返回值：返回值为 0
   d) 函数主要功能：将当前网络设备对应的物理端口状态置 0，表示端口

关闭，硬件不允许此端口上的报文收发，硬件将此端口的 LED 灯置为关状态。停止网络接口队列，上层协议不可以再发送报文。

3. 报文发送接口
   a) 实现函数： netdev_tx_t npe_xmit_frame(struct sk_buff *skb,struct net_device *dev)
   b) 参数：skb 表示要发送报文指针，dev 表示发送报文的网络设备
   c) 返回值：返回值为 NETDEV_TX_OK
   d) 函数主要功能：将上层协议的 skb 报文发送到硬件。由于 NPE 的特殊性，此类型报文需要通过格式转换或其他方式直接 DMA 到硬件，详细内容请参考后面内容。
4. 计数器接口
   a) 实现函数：struct net_device_stats* npe_get_stats(struct net_device *dev)
   b) 参数：dev 表示网络设备
   c) 返回值：返回当前网络设备的标准计数器数据结构地址
   d) 函数主要功能：此函数主要用来获取当前网络设备的各类型报文计数，计数器结构定义在网络设备的私有数据 adapter 中，每个端口的报文计数保存在硬件寄存器中，每次报文统计，从指定端口读取相应寄存器值后返回，由于读多个寄存器存在时延，实时计数器值可能存在一小点误差，但当报文停止收发时再读取，则不存在误差。
5. 接收模式接口
   a) 实现函数：void npe_set_rx_mode(struct net_device *dev)
   b) 参数：dev 表示网络设备
   c) 返回值：空
   d) 函数主要功能：此函数主要用来设置接口的报文接收模式，如混杂模式与非混杂模式和多播地址设置等
6. MAC 地址设置接口
   a) 实现函数：int npe_set_mac_address(struct net_device *dev,void *addr)
   b) 参数：dev 表示网络设置，addr 为 sockaddr 数据结构，内部保存了 MAC 地址

c) 返回值：0 表示设置成功，非 0 表示不成功，主要是 MAC 地址不正确

d) 函数主要功能：设置网络设备的 MAC 地址，设备的 MAC 地址在初始化时已经配置，支持用户对物理端口的 MAC 地址进行修改。

7. MTU 设置接口

a) 实现函数：int npe_change_mtu(struct net_device *dev, int new_mtu)

b) 参数：dev 表示网络设备，new_mtu 表示新的 MTU 值大小

c) 返回值：0 表示设置成功，非 0 表示设置失败，一般是 MTU 值大小不正确

d) 函数主要功能：设置物理端口的 MTU 值大小，在 NPE 驱动中，报文的最大长度为 2048 字节，报文 meta data 头为 32 字节，IP 对齐 2 字节，私有数据空间 18 字节，所以数据报文最大长度为 2048-32-2-18=1996 字节，即 MTU 最大为 1996 字节，MTU 最小为 64 字节。

8. 发送报文超时接口

a) 实现函数：void npe_tx_timeout(struct net_device *dev)

b) 参数：dev 表示网络设备

c) 返回值：空

d) 函数主要功能：此函数使用在一般以太网驱动中，当报文发送较慢时，报文描述符环未写满，但第一个报文写入描述符环已经超时（由 dev->watchdog 确定执行周期，使用时间函数调用）。NPE 不存在发送超时，但可以利用此函数做一些其他工作，如做性能统计分析。

4.1.5.4平台配置接口

NPE 驱动提供了一个字符设备驱动，设备路径为/dev/npe_debug。驱动实现了此字符设备的主要基本操作函数。主要包括：open、release、read 和 write。驱动设备主要负责用户空间与内核空间的数据读写、NPDK 核心驱动的数据控制与调试、平台配置等。

1. 设备打开接口

a) 实现函数：int npe_debug_open(struct inode *inode,struct file *filp)

b) 参数：inode 表示文件系统索引节点，filp 表示文件描述符

c) 返回值：0 表示成功，-1 表示失败

d) 函数主要功能：打开文件设备

2. 设备释放接口

a) 实现函数：int npe_debug_release(struct inode *inode,struct file *filp)

b) 参数：inode 表示文件系统索引节点，filp 表示文件描述符

c) 返回值：0 表示成功，-1 表示失败

d) 函数主要功能：设备关闭时，释放设备相关资源

3. 读设备接口

a) 实现函数：ssize_t npe_debug_read(struct file *filp,char __user *buf,size_t count,loff_t *f_pos)

b) 参数：filp 表示文件描述符，buf 表示读取所需相关信息，count 表示 buf 大小，f_pos 表示读取位置偏移

c) 返回值：buf 的大小

d) 函数主要功能：读函数主要从 buf 指针提取操作 CPU 号，如果 CPU 为 0，则调用 NPDK 核心驱动调试代码，将 buf 后的数据作为参数输入。如果 CPU 号为其他，则提取当前 CPU 号函数的读操作函数指针，调用并得到返回数据，将数据复制到 buf 并返回。

4. 写设备接口

a) 实现函数：ssize_t npe_debug_write(struct file *filp,const char __user *buf,size_t count,loff_t *f_pos)

b) 参数：filp 表示文件描述符，buf 表示写入相关信息，count 表示 buf 大小，f_pos 表示写入位置偏移

c) 返回值：buf 的大小

d) 函数主要功能：写函数主要将 buf 中的数据写入对应内核模块。首先从 buf 提取操作 CPU 号，如果 CPU 为 0 则调用 NPDK 核心驱动调试代码，否则根据此 CPU 提取对应的写操作函数指针，将 buf 数据复制作为输入参数，调用写指针函数实现对内核数据的写操作。

4.1.6 平台配置接口

4.1.6.1Localbus 格式

配置通路为 UM 模块与主控 CPU 之间的 Localbus 通路，其数据格式可参考 NetMagic08 中的 Localbus 标准协议。

4.1.6.2BV 配置格式

主要为用户对 BV 查表引擎的表项和规则 rule 配置，二者的格式与定义可 BV 的信息接口格式。

附录 A　BV 算法基本原理

BV 算法的基本原理是在每次搜索返回一个结果向量：BV（bit-vector）向量，其中每条规则在向量中对应 1-bit，并且用“1”表示搜索内容满足该“1”位置所对应的规则，“0”表示不满足。例如有规则集 R={R1,R2,R3,R4}，当输入的 key 满足 R’={R3,R4}时，返回的 BV 向量为“0011”。整个的查找过程如图 1-a 所示，输入搜索 key=“10”，搜索输入 key=“1”返回的 BV’={0011}，搜索输入 key=“0”返回的 BV”={1011}，最后按位求&得到 BV={0011}。

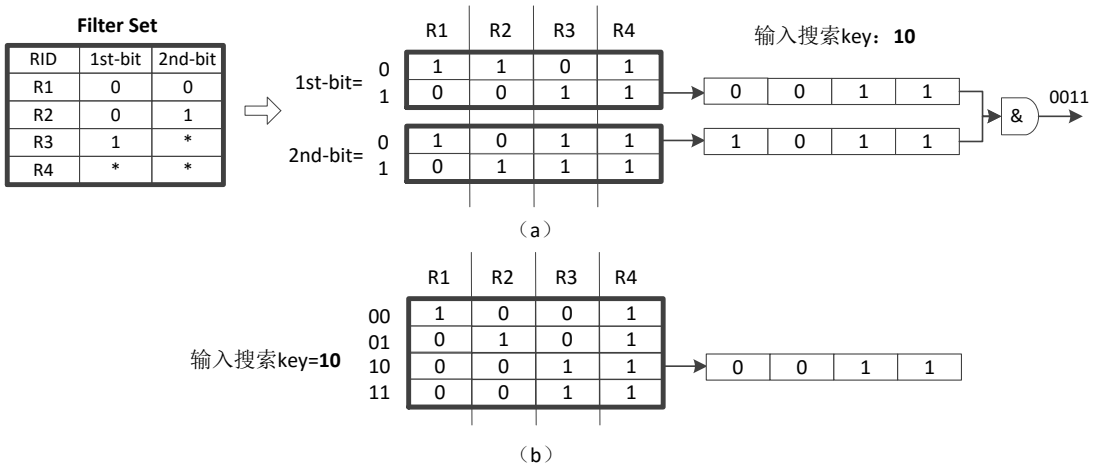


图 A-1 BV 算法

BV 算法在最后阶段，需要判断匹配了哪些规则，并且从中选出优先级最高的匹配规则，上面的例子中则是需要比较 R3，R4 的优先级，如果采用逐位比较的方式，则时间代价为 O(N)，其中 N 为规则数量。但是如果规则按照优先级降序的方式排列，则可以采用 2 分法查找，时间开销为 log N。例如：规则优先级对应关系为

R={　R1→priority: 4；

R2→priority：3；

R3→priority：2；

R4→priority：1}（其中数字越大，优先级越低）

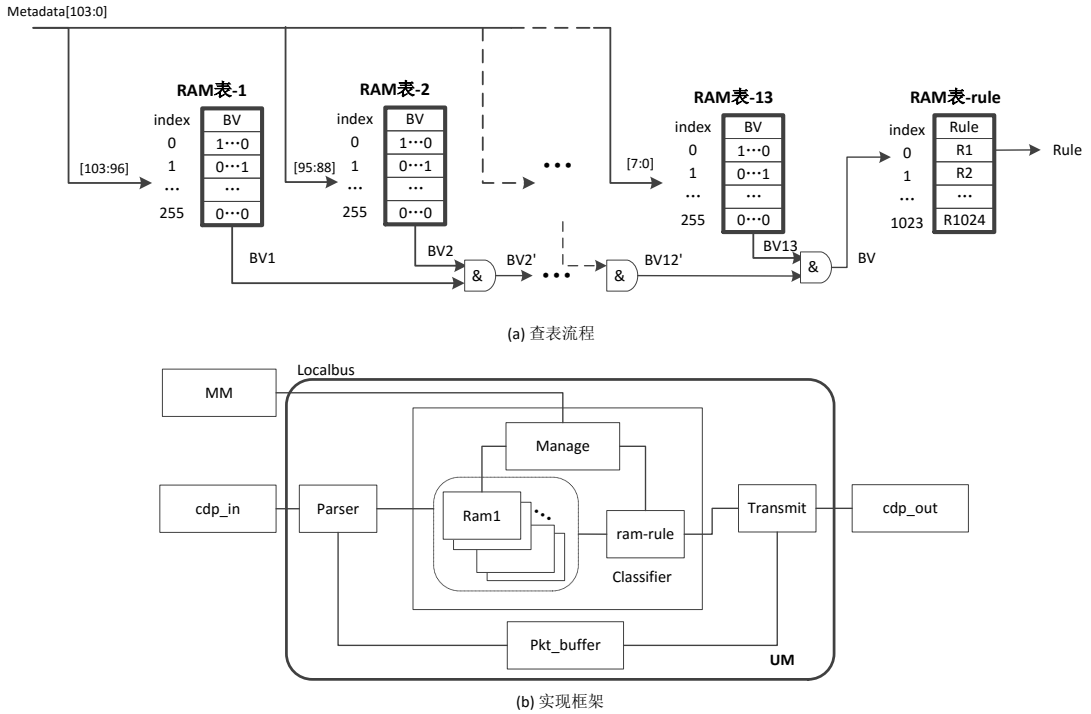
当输出的BV={1111}时，逐位比较的过程为，①获得R1规则，②获得R2规则，比较与R1的优先级大小，保留较小的，③获得R3规则，与前一阶段剩下的规则比较优先级大小，保留较小的，④获得R4规则，与前一阶段剩下的规则比较优先级大小，输出较小的。而将规则降序排列得到R={R4,R3,R2,R1}，采用2分法查找，其过程为：①判断1111是否为“0”（不为0）；②取前2bit，判断“11”是否为“0”（不为0）；③取前1bit，判断“1”是否为“0”（不为0），得到最高优先级规则为第一条规则，即R4。

上述方式存在的缺点是，搜索过程中，每次只比较1-bit，导致搜索延迟较高。可以增加比较的步伐，如图1-b所示，采用stride=2，则一次就可以获得BV={0011}。

以传统5元组查表为例（源IP地址：32-bit，目的IP地址：32-bit，源端口号：16-bit，目的端口号：16-bit，协议类型：8-bit），总共104bit。支持1K表项，把BV向量的长度设置为1kb。采用stride=8bit步幅长度。

BV算法搜索流程如图A-2所示，分成14阶流水，前13级是搜索字段，最后1阶段搜索匹配规则。每一个搜索阶段结束后都会返回一个BV向量，与之前输入的BV向量求&操作，得到新的BV向量，作为下一阶段的输入。最后根据BV向量搜索规则寄存器，获得相应的匹配规则。

具体的FPGA实现框架如图2-b所示，manage模块用于管理配置匹配阶段的13个RAM表以及规则RAM表。



(a) 查表流程

(b) 实现框架

图 A-2 算法的 FPGA 实现

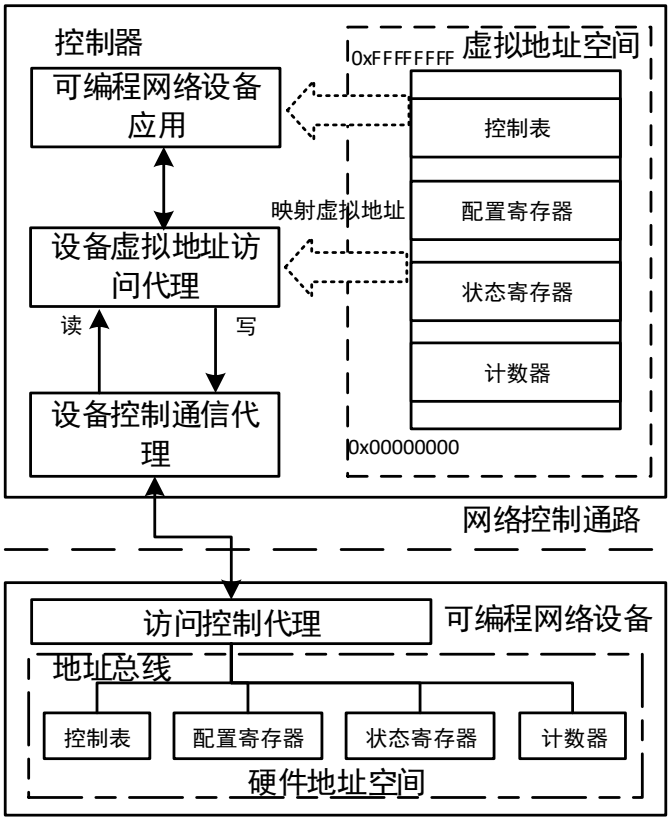
附录 B 虚拟地址空间分配规范

编程网络是将控制平面与转发平面相分离的构架网络，基于这种可编程网络思想的基础上，我们将转发平面的资源抽象出来如寄存器，存储器，计数器等，提供给控制平面的是类似于计算机内部的“虚拟地址空间”，控制平面读写虚拟地址空间，转发平面组织这些虚拟地址空间形成转发策略。同时，利用中间控制协议使得控制平面与转发平面进行交互，控制平面不需要了解转发平面的工作原理，只需注重控制端软件逻辑开发。

可编程实验网络设备可用于网络教学，网络科研，专网实验等领域，加速网络创新的发展。通过这种管理控制方式，转发平面的硬件使用 FPGA 使得硬件可重构，软件可编程使得网络平台灵活可扩展，可加速新协议，新网络体系架构的开发、验证以及部署。

FPGA 转发设备将提供给软件访问的寄存器、片内 RAM 空间以及片外

得存储器空间等映射到同一个虚拟地址空间中，对于控制端可见的是这些虚拟地址空间，控制器通过对这些虚拟地址空间的 Read，Write 命令或函数就可以访问到所有被管理的资源，对可编程网络设备进行管理控制，硬件将控制端传送的虚拟地址进行映射实际可编程资源的地址，提取出映射内容，对实际硬件资源进行读写操作，其虚拟地址空间表达方式可参考图 1 内虚拟地址空间，其控制端的虚拟地址空间需要与可编程网络设备内的虚拟地址空间保持一致性。



附录 C　NMAC API

NMAC API 是实现 FAST 平台无关设计的基础，交换机软件通过调用 NMAC API 访问底层的 FPGA 硬件。不论 FPGA 硬件是通过 PCIe 总线与 CPU 连接，如 NetMagic Pro，还是通过网络与 CPU 连接，如 NetMagic08。即 NMAC 库对上层软件屏蔽了 FPGA 与 CPU 的通信方式，使得 FAST 控制软件具有跨平台的可移植性。

NMAC API 定义的函数如下所示。

u_int16_t init(struct controller*, struct nmac_dev*)，初始化函数，将配置文件中的参数配置写入相应的数据结构中，查看网络接口状态，返回初始报文序列号；

void startup(struct controller ctr, struct nmac_dev dev , u_int8_t flag, u_int16_t seq)。参数描述：ctr，controller 结构对象；dev，nmac_dev 结构对象；flag，表示标志位值；seq，16 位序列号；本函数构造握手报文并发送，实现 controller 与 NetMagic 设备间建立连接，返回连接是否成功消息；

void write_data(struct controller ctr, struct nmac_dev dev, u_int8_t num, u_int32_t addr, u_int32_t data, u_int16_t seq)。参数描述：ctr，controller 结构对象；dev，nmac_dev 结构对象；num，表示连续请求数（目前为 1，只支持单个地址写）；addr，32 位地址，要写入数据的地址；data，32 位数据，要写入的数据；seq，16 位序列号；本函数实现单个地址的写数据功能，构造并发送写请求数据包，捕获写响应数据包，返回是否写成功消息。

u_in32_t read_data(struct controller ctr, struct nmac_dev dev, u_int8_t num, u_int32_t addr, u_int32_t data, u_int16_t seq)。参数描述：ctr，controller 结构对象；dev，nmac_dev 结构对象；num，表示连续请求数（目前为 1，只支持单个地址读）；addr，32 位地址，要读数据的地址； seq，16 位序列号；本函数实现单个地址的读数据功能，构造并发送读请求数据包，捕获写响应数据包，返回读

的数据。

u_int32_t keep_alive(struct controller ctr, struct nmac_dev dev, u_int16_t seq)。参数描述：ctr，controller 结构对象；dev，nmac_dev 结构对象；seq，16 位报文序列号；nmac_header 中 burst 为 0；根据 nmac_type 识别报文类型；本函数实现 keep alive 功能，构造并发送 keep alive 报文并发送，捕获 keep alive 响应数据包，返回响应的内容。

void sample_pkt(struct controller ctr, struct nmac_dev dev, u_int8_t port_id, u_int16_t seq)。参数描述：ctr，controller 结构对象；dev，nmac_dev 结构对象；port_id，表示报文采样的端口，seq，16 位报文序列号；本函数实现报文采样功能，构造并发送报文采样请求报文并发送，捕获报文采样响应数据包，将报文采样的内容保存到变量中。

void disconnect(struct controller ctr, struct nmac_dev dev , u_int8_t flag, u_int16_t seq)。参数描述：ctr，controller 结构对象；dev，nmac_dev 结构对象；flag，表示标志位，seq，16 位报文序列号；本函数实现连接释放功能，构造并发送连接释放请求报文并发送，捕获释放确认数据包，返回是否成功消息。

附录 E 文档主要贡献者

厉俊男、严锦立、何璐蓓、邱为好、刘长鑫、孙小添、曾强、刘晓骏。