FSL91030(M)芯片

原理文档

手册版本: V1.2

武汉飞思灵微电子技术有限公司

2022 年 12 月

目 录

1 端口描述

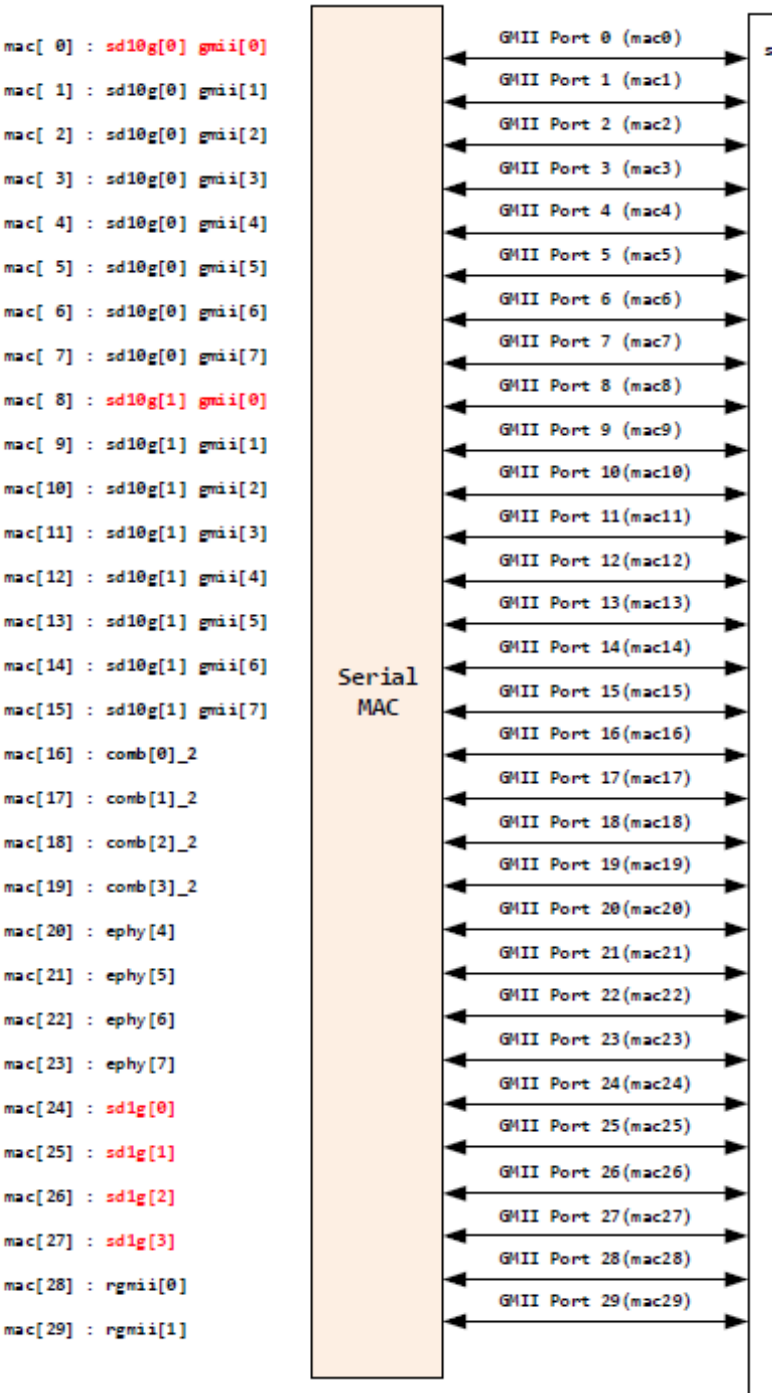
端口配置是入口管道的第一阶段，主要目的是获取数据包进入到芯片相关的端口配置信息。由于FSL91030M 芯片具有不同的端口概念，即物理端口和数据包处理端口。广义上将物理端口 id 号和 mac id 一一对应，数据包处理端口和逻辑端口（同时和面板口）一一对应。

FSL91030M 在上电后通过应用场景和配置文件信息首先确定芯片的工作模式和端口形态，进而来确定端口的配置信息。

下表为 FSL91030M 单芯片常用的几种工作模式和端口配置信息：

业务口 物理端口/mac_id 端口形态		4 电口		8 电口		6*1G 光口		2*10G 光口 + 8 电口		4*1G 光口 + 8 电口		2*10G 光口 + 4 电口 + 4comb	
0		GE0	16	GE0	16	GE0	0	XE0	0	GE0	16	XE0	0
1		GE1	17	GE1	17	GE1	8	XE1	8	GE1	17	XE1	8
2		GE2	18	GE2	18	GE2	24	GE0	16	GE2	18	Comb0	16
3		GE3	19	GE3	19	GE3	25	GE1	17	GE3	19	Comb1	17
4				GE4	20	GE4	26	GE2	18	GE4	20	Comb2	18
5				GE5	21	GE5	27	GE3	19	GE5	21	Comb3	19
6				GE5	22			GE4	20	GE6	22	GE0	20
7				GE7	23			GE5	21	GE7	23	GE1	21
8								GE6	22	GE8	24	GE2	22
9								GE7	23	GE9	25	GE3	23
10										GE10	26		
11										GE11	27		
31	CPU												
28	RGMII0												
29	RGMII1												

FSL91030M 上有两个 10G serdes 四个 1G serdes 以及八个 GEPHY，在物理上与 mac 对接的拓扑如下：



2 VLAN 功能

2.1 术语

术语	说明
Stag	Service VLAN Tag(外层 vlan tag)
Ctag	Customer VLAN Tag(内层 vlan tag)
SOT	Single outer tag
SIT	Single inner tag
DT	Double tag
UT	Untagged
Vid	Vlan ID
Cos	优先级（对应 tag 的 priority）
Cfi	规范为 0，非规范为 1
inIsLag	指示输入的内部逻辑端口号是否为 LAG 端口，高有效
inLport	输入的内部逻辑端口号，当 inIsLag 有效时表示 LAG 端口，否则表示普通端口。
outIsLag	指示输出的内部逻辑端口号是否为 LAG 端口，高有效
outLport	输出的内部逻辑端口号，当 outIsLag 有效时表示 LAG 端口，否则表示普通端口。

2.2 入口 VLAN

入口 Vlan 功能主要包括 4 个部分，即 VlanTag 检查、Vlan 翻译、VLan 查找、Vlan 滤波，如图 2-1 所示。报文进来之后，根据报文 Tag 类型和 VlanGet 模块来获取初始 Vlan 信息，即 ScosInit、ScfiInit、SvidInit、CcosInit、CcfiInit、CvidInit。然后会进入 Vlan 编辑模块来获取 upd.{svid、scos、scfi、cvid、ccos、ccfi}，当 iVtPortSrm.iVtEditEn=0 时，不使能 Vlan 编辑，Vlan 信息来自报文原始信息；当 iVtPortSrm.iVtEditEn=1 时，使能 Vlan 编辑功能，Vlan 编辑需要知道 VlanOpIdx 和用来添加或替换的 vlan 信息（svid、scos、scfi、cvid、ccos、ccfi）。这些信息的获取有四种途径，分别是 ProtoVlan、FlowVlan、XlateVLan、Port，通过查找对应的表项 iVtProtoVlanSrm、iVtFlowVlanTcmSrm、iVtXlateSrm（包含左 0-3 和右 0-3 共 8 个表项，用来防止哈希冲突）、iVtPortSrm 获取。可以根据需求使能相应的 Vlan 编辑模块来获取 VlanOpIdx，通过索引 VlanOpIdx 查询 iVtVlanOpSrm 来对原始报文进行编辑。在进行 Vlan 查找时，可以使用 PktSvid 或者 UpdSvid，通过设置 iNetPortSrm.usePktSvid=1 或 iNetPortSrm.useUpdSvid=1 来实现，从而获取内部 Vlan Id。

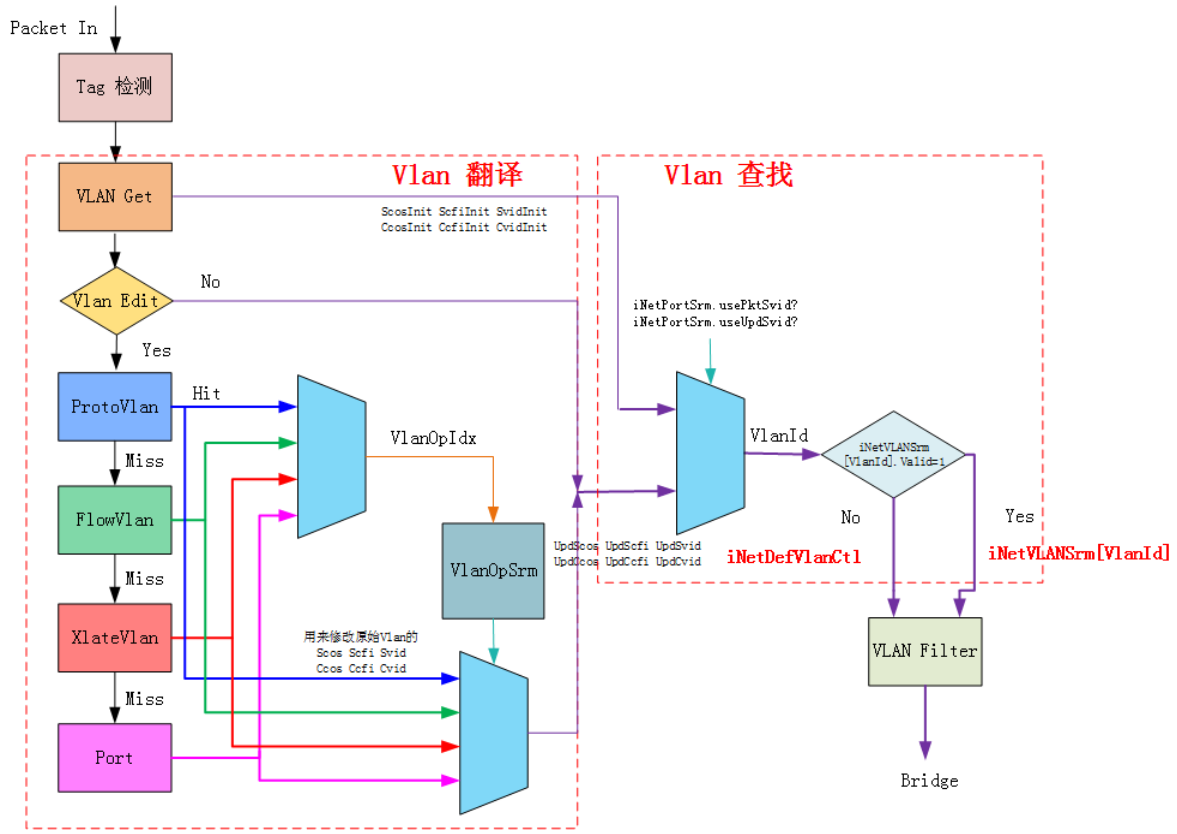


图 2-1 入口 VLAN 功能示意图

根据 Vlan Id 查找对应 iNetVlanSrm[VlanId]表，当 iNetVlanSrm[vlanId]表项无效时，使用 iNetDefVlanCtl 来获取 Vlan 的广播域等信息。因此，在芯片默认配置下（不设置 Vlan 滤波），iNetVlanSrm 表项无效，使用 iNetDefVlanCtl.portbmp 进行洪泛，可以正常进行报文转发。当 iVtPortSrm.VlanFilterEn=1 时，使能入口 Vlan 滤波功能，如果报文输入端口不在 Vlan 广播域内，则将该报文丢弃。具体实现方式详见各小节。

2.2.1 Tag 检测

Tag 检测示意图如图 2-2 所示，当报文进来之后，首先对其 12、13 字节进行检查，当其满足等式 1 时，说明其带 stag，即带有外层 Vlan，然后对报文 16、17 字节进行检查，满足等式 2 时，说明报文同时携带内外两层 Vlan，即 DT；如果不满足等式 2，则只带有一层外层 Vlan，即 SOT。而当报文的 12、13 字节不满足等式 1 而满足等式 2 时，则只带有一层内层 Vlan，即 SIT，当报文的 12、13 字节既不满足等式 1 又不满足等式 2 时，报文不带 Vlan，即 UT。

Pcket.stagTpid = Ipr0StagTpidCtl.stagTpidx (x=1、2、3、4，Stpid0 默认值是 0x88A8，Ipr0StagBmpCtl.StagBmp[InPort*4+3:InPort*4] 对应 bit 位使能)

Packet.ctagTpid = Ipr0CtagTpidCtl.ctagTpid (Stpid0 默认值是 0x8100)

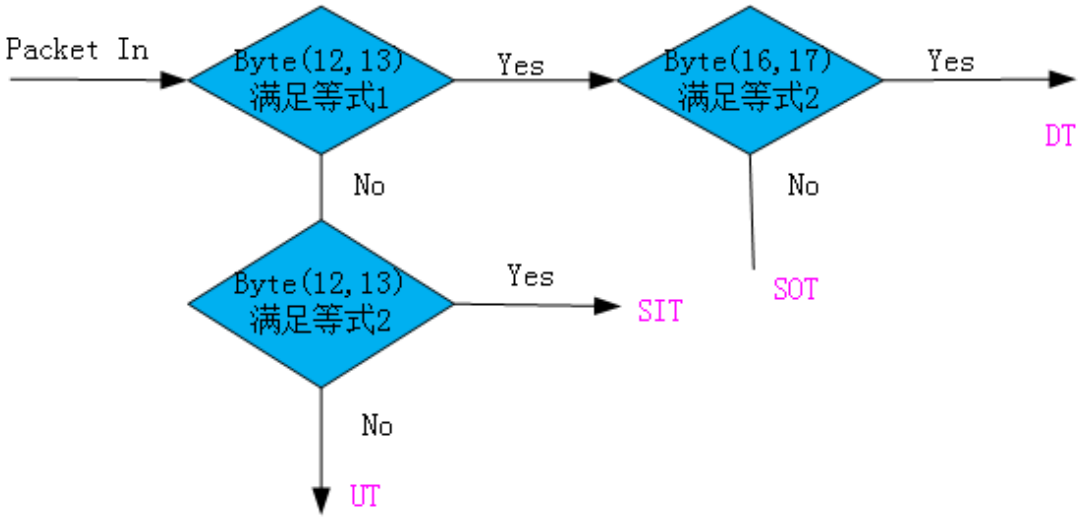


图 2-2 Tag 检测示意图

Tag 检测相关寄存器和表项

iPr0StagBmpCtl

Name	Width	Bits	R/W	Def	Description
stagBmp	140	139:0	R/W	0x1111111111111111 1111111111111111111	各端口的 stag bmp，每个端口 4 个比特

iPr0StagTpidCtl

Name	Width	Bits	R/W	Def	Description
stagTpid3	16	63:48	R/W	0x0	stagTpid3
stagTpid2	16	47:32	R/W	0x0	stagTpid2
stagTpid1	16	31:16	R/W	0x0	stagTpid1
stagTpid0	16	15:0	R/W	0x88a8	stagTpid0

iPr0CtagTpidCtl

Name	Width	Bits	R/W	Def	Description
ctagTpid	16	15:0	R/W	0x8100	ctagTpid

2.2.2 Vlan 翻译

Vlan 翻译包括 Vlan Get、Proto Vlan、FLowVlan、XlateVlan 和 Vlan 编辑几个部分，下面分别对其功能进行介绍。

2.2.2.1 Vlan Get

通过 Tag 检测获取报文 vlan 类型之后，可以得到报文的 vlan 信息，vlan 信息来自原始报文或者对应端口的默认 vlan 信息，如下表所示。

Vlan 类型	stag	ctag

DT	svidinit: packet.stag.svid scosinit: packet.stag.scos scfiinit: packet.stag.scfi	cvidinit: packet.ctag.cvid ccosinit: packet.ctag.ccos ccfiinit: packet.ctag.ccfi
SOT	svidinit: packet.stag.svid scosinit: packet.stag.scos scfiinit: packet.stag.scfi	cvidinit: iVtPortSrm.defcvid ccosinitcos: iVtPortSrm.defcos ccfiinit: iVtPortSrm.defcfi
SIT	svidinit: iVtPortSrm.defsvid scosinit: iVtPortSrm.defcos scfiinit: iVtPortSrm.defcfi	cvidinit: packet.ctag.cvid ccosinit: packet.ctag.ccos ccfiinit: packet.ctag.ccfi
UT	svidinit: iVtPortSrm.defsvid scosinit: iVtPortSrm.defcos scfiinit: iVtPortSrm.defcfi	cvidinit: iVtPortSrm.defcvid ccosinit: iVtPortSrm.defcos ccfiinit: iVtPortSrm.defcfi

2.2.2.2 ProtoVlan

如图 2-3 所示，Protocal Vlan 查找使用遍历的方式，在 iVtPortSrm.protoVlanEn=1 的条件下，命中条件如下所示。

inLport=iVtProtoVlanCtl[i].inLport

inIsLag=iVtProtoVlanCtl[i].inIsLag （等式 1）

ethType=iVtProtoVlanCtl[i].ethType (i 可取 0-15 之间)

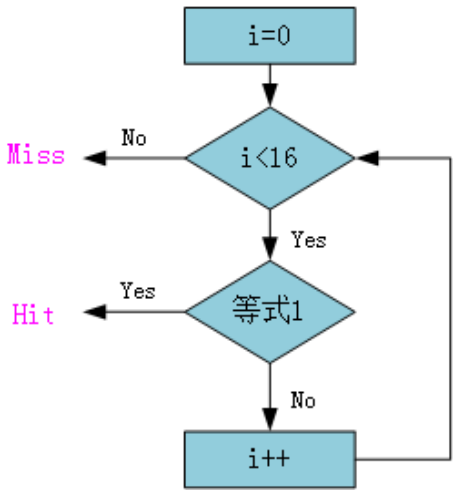


图 2-3 Protocal Vlan 遍历方式

通过遍历方式获取命中时的 iVtProtoVlanSrm 表项索引，从而查询对应表项来获取替换或添加的 svid、scos、scfi、cvid、ccos、cscfi、vlanOpIdx 值，用于 VLan 编辑使用。

2.2.2.3 Flow Vlan

Flow Vlan 查找方法有四种，即 vlanKey、macKey、ipv4Key、和 ipv6Key，在 iVtPortSrm.flowVlan0En=1 时，通过 Key 值和 iVtFlowVlanTcm 表项匹配可以得到对应的 iVtFlowVlanTcmSrm 的索引编号，从而获取替换或添加的 svid、scos、scfi、cvid、ccos、cscfi 值和 VlanOpIdx。具体采用哪种方式和端口配置以及包的类型有关，如图 2-4 所示。

具体的查表策略和查找过程可参照第四章 ACL 原理。

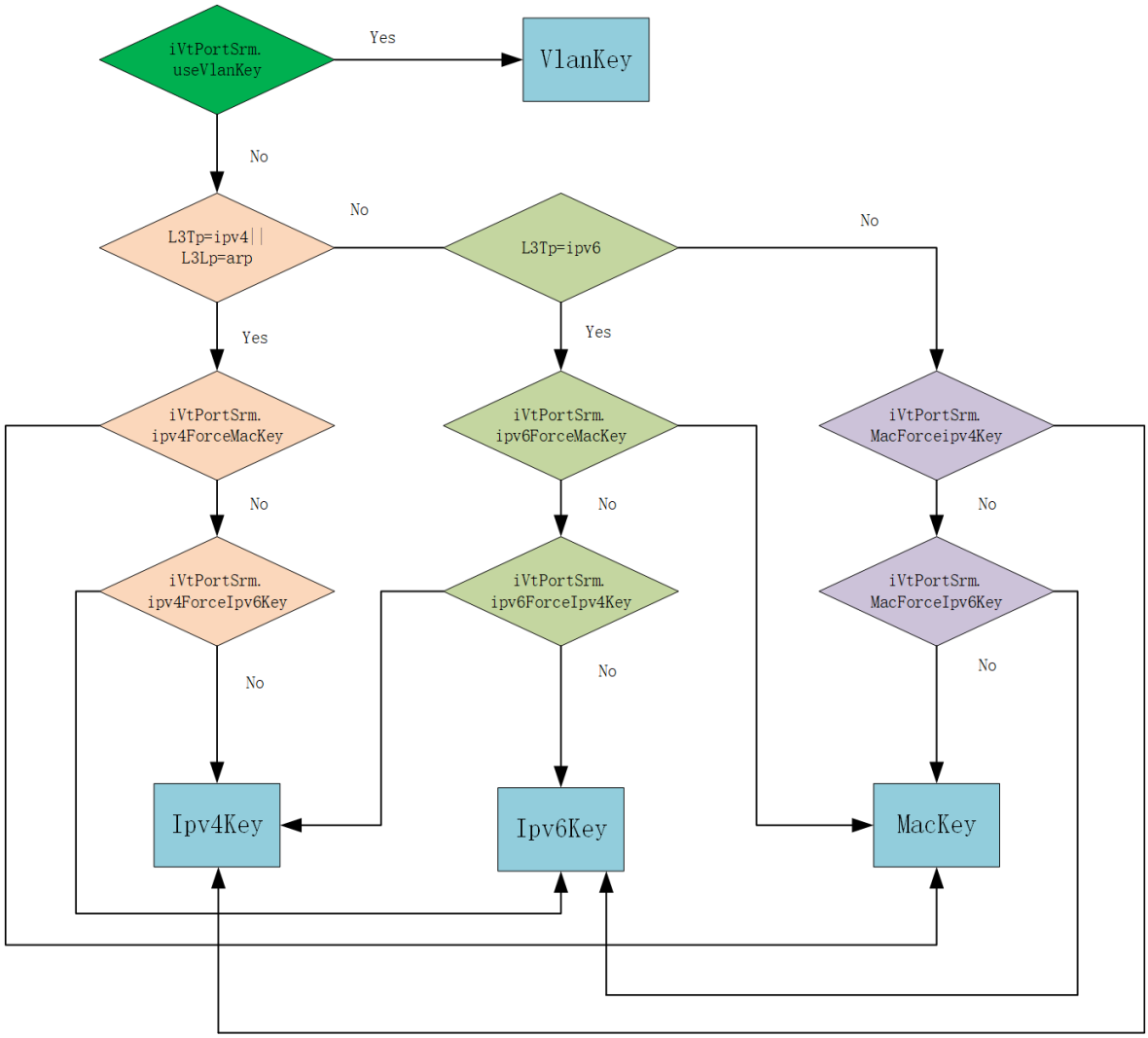


图 2-4 Flow Vlan 端口配置

vlanKey 由以下域组成：

Name	Width	Bits
2'b11	2	139:138
meta.sip[17:0]	18	137:120
phv.eth.sa	48	119:72
meta.ethType	16	71:56

meta.l2Tp	2	55:54
meta.l3Tp	4	53:50
meta.l4Tp	3	49:47
meta.l5Tp	3	46:44
meta.oamTp	3	43:41
phv.stag.{vid,cos,cfi}	16	40:25
phv.stag.isvalid	1	24:24
phv.ctag.{vid,cos,cfi}	16	23:8
phv.ctag.isvalid	1	7:7
meta.inIsLag	1	6:6
meta.inLport	6	5:0

MacKey 由以下域组成：

Name	Width	Bits
2'b00	2	279:278
res0	36	277:242
phv.ctag.isvalid	1	241:241
meta.l4Tp	3	240:238
meta.ethType	16	237:222
meta.l5Tp	3	221:219
meta.oamTp	3	218:216
phv.stag.{vid,cos,cfi}	16	215:200
phv.ctag.{vid,cos,cfi}	16	199:184
phv.l2_slow.subType	8	183:176
meta.ptpMsgType	4	175:172
meta.l3Udf0	8	171:164
meta.l3Udf1	8	163:156
meta.l4Udf0	8	155:148
meta.l4Udf1	8	147:140
2'b00	2	139:138
phv.stag.isvalid	1	137:137
meta.l3Tp	4	136:133
meta.l2Tp	2	132:131
phv.eth.da	48	130:83
sa	48	82:35
portBmp	35	34:0

iPv4Key 由以下域组成：

Name	Width	Bits
2'b01	2	279:278
res2	7	277:271
meta.ipOption	1	270:270
meta.ipHdrErr	1	269:269

meta.ipLen	16	268:253
phv.l3.ipv4.id	16	252:237
phv.l4.tcp.flag	8	236:229
meta.l4SrcPort	16	228:213
meta.l4DstPort	16	212:197
meta.l5Tp	3	196:194
meta.oamTp	3	193:191
meta.ttl	8	190:183
phv.ipv4.ihl	4	182:179
phv.ipv4.flg	3	178:176
meta.ptpMsgType	4	175:172
meta.l3Udf0	8	171:164
meta.l3Udf1	8	163:156
meta.l4Udf0	8	155:148
meta.l4Udf1	8	147:140
2'b01	2	139:138
meta.icmpType	8	137:130
meta.icmpCode	8	129:122
meta.tos	8	121:114
meta.l3Tp	4	113:110
meta.l4Tp	3	109:107
meta.ipProto	8	106:99
meta.sip	32	98:67
meta.dip	32	66:35
portBmp	35	34:0

iPv6Key 由以下域组成：

Name	Width	Bits	R/W	Def
2'b10	2	559:558	R/W	0x0
meta.ptpMsgType	4	557:554	R/W	0x0
phv.eth.da	48	553:506	R/W	0x0
phv.l4.tcp.flag	8	505:498	R/W	0x0
meta.ipLen	16	497:482	R/W	0x0
phv.l3.ipv4.id	16	481:466	R/W	0x0
meta.l5Tp	3	465:463	R/W	0x0
meta.oamTp	3	462:460	R/W	0x0
meta.l4SrcPort	16	459:444	R/W	0x0
meta.l4DstPort	16	443:428	R/W	0x0
meta.ipv6ExtHdrTp	8	427:420	R/W	0x0
2'b10	2	419:418	R/W	0x0
phv.stag.{vid,cos,cfi}	16	427:412	R/W	0x0
phv.ctag.{vid,cos,cfi}	16	401:386	R/W	0x0
phv.ctag.isvalid	1	385:385	R/W	0x0
phv.stag.isvalid	1	384:384	R/W	0x0

meta.icmpType	8	383:376	R/W	0x0
meta.icmpCode	8	375:368	R/W	0x0
meta.l3tp	4	367:364	R/W	0x0
meta.tos	8	363:356	R/W	0x0
meta.ttl	8	355:348	R/W	0x0
phv.ipv6.ext.isValid	1	553:553	R/W	0x0
meta.l3Udf0	8	346:339	R/W	0x0
meta.l3Udf1	8	338:331	R/W	0x0
meta.l4Udf0	8	330:323	R/W	0x0
meta.l4Udf1	8	322:315	R/W	0x0
portBmp	35	314:280	R/W	0x0
2'b10	2	279:278	R/W	0x0
meta.ipOption	1	277:277	R/W	0x0
meta.ipHdrErr	1	276:276	R/W	0x0
meta.ipProto	8	275:268	R/W	0x0
meta.sip	128	267:140	R/W	0x0
2'b10	2	139:138	R/W	0x0
meta.l4Tp	3	137:135	R/W	0x0
meta.dip	128	134:7	R/W	0x0
meta.inIsLag	1	6:6	R/W	0x0
meta.inLport	6	5:0	R/W	0x0

iVtFlowVlanTcm 深度是 128，包含的域如下表所示。由上面 key 表的宽度可知，vlanKey 只需要占用 1 表项，macKey 和 ipv4Key 占用 2 个表项，ipv6Key 则要占用 4 个表项。

iVtFlowVlanTcm 表项

Name	Width	Bits	R/W	Def	Description
validMask	1	281:281	R/W	0x0	有效指示掩码
keyMask	140	280:141	R/W	0x0	查找 key 掩码
valid	1	140:140	R/W	0x0	有效指示
key	140	139:0	R/W	0x0	查找 key

因此，iVtFlowVlanTcmSrm 最多支持 128 个 vlanKey 或者 64 个 macKey 或者 64 个 iPhv4Key 或者 32 个 ipv6Key，如果采用混合 Key 的方式则根据实际分配情况确定，通过配置寄存器 iVtFlowVlanCtl 实现。

iVtFlowVlanCtl 寄存器

Name	Width	Bits	R/W	Def	Description
ipv6KeyRstCtl	16	63:48	R/W	0x0	同 macKeyRstCtl
ipv4KeyRstCtl	16	47:32	R/W	0x0	同 macKeyRstCtl
vlanKeyRstCtl	16	31:16	R/W	0x0	{indexBase[6:0],keySize[1:0],tableBase[6:0]}
macKeyRstCtl	16	15:0	R/W	0x0	{indexBase[6:0],keySize[1:0],tableBase[6:0]}

iVtFlowVlanTcmSrm 的索引计数方式如下：

flowVlanIndex = (Index-indexBase)>>keySize +tableBase

其中 Index 是通过 hashKey 计算出的索引值。

举例说明：

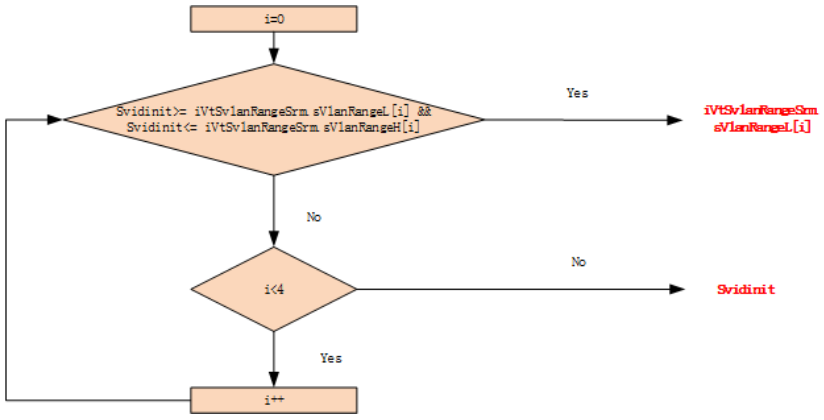
假设配置 iPv4 和 ipv6 两种类型的 Tcm 表，可配置 32 个 ipv4 和 16 个 ipv6Key 表。因此可配置 ipv4KeyRstCtl.indexBase=0 ， ipv4KeyRstCtl.keySize=1 ， ipv4KeyRstCtl.tableBase=0 ； ipv6KeyRstCtl.indexBase=64，ipv6KeyRstCtl.keySize=2，ipv6KeyRstCtl.tableBase=32。则 ipv4Key 索引值在 0-31 之间；ipv6Key 索引值在 32-47 之间。

2.2.2.4 XlateVlan

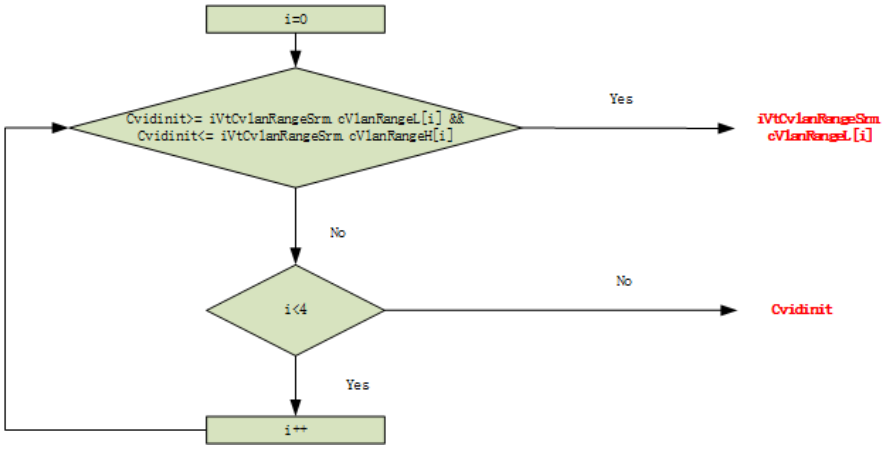
XlateVlan 包含 Xlate0 和 Xlate1 两部分，每部分又包含 8 个哈希表，Xlate0 包含为 iVtXlateLeftSrm[x](x 在 0-3 之间)、iVtXlateRightSrm[y](y 在 0-3 之间)（8 个哈希表是为了防止哈希冲突）。在生成 XlateVlanKey 之前，可以通过 iVtPortSrm.svlanRangeEn 和 iVtPortSrm.cvlanRangeEn 设置 VlanRange 功能，通过 iVtPortSrm.scosMapEn 和 iVtPortSrm.ccosMapEn 设置 CosMap 功能。通过生成的 XlateKey 和设置的哈希算法（通过 iVtPortSrm.AlgTp 设置），可以算出对应 XlateSrm 表的索引，从而得到用于替换或添加的 svid、scos、scfi、cvid、ccos、cscfi 值以及 VlanOpIdx，用于进行 Vlan 编辑。

1. VlanRange

VlanRange 即将一定区间的 Vlan 转换为相同的 vlan 使用，包括 SvidRange 和 CvidRange 两部份操作，SvidRange 通过设置 iVtSvlanRangeSrm 表实现，表项深度为 16。具体使用哪个表项通过 iVtPortSrm.svlanRangeIdx 获取，然后通过下图获取 SvidRange 值。



同理，通过 iVtPortSrm.cvlanRangeIdx 得到对应的 iVtSvlanRangeSrm 表，使用下图获取 CvidRange 值。



2. CosMap

CosMap 即对报文的优先级进行映射，来修改报文的优先级，包含 ScosMap 和 CcosMap 两部分，分别通过设置 iVtScosMapSrm 和 iVtCcosMapSrm 实现，表项深度为 8。iVtPortSrm.scosMapIdx 和 iVtPortSrm.ccosMapIdx 分别对应 Cos 映射表的索引值，根据 Vlan Get 获取的 ScosInit、ScfiInit、CcosInit、CcfiInit 值来进行相应的映射，如表 2-1、表 2-2、表 2-3 和表 2-4 所示。

表 2-1 ScosInit 映射关系

ScosInit	ScosMap
0	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos0
1	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos1
2	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos2
3	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos3
4	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos4
5	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos5
6	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos6
7	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cos7

表 2-2 ScfiInit 映射关系

ScfiInit	SfiMap
0	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cfi0
1	iVtScosMapSrm[iVtPortSrm.scosMapIdx].cfi1

表 2-3 CcosInit 映射关系

CcosInit	CcosMap
0	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos0
1	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos1
2	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos2
3	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos3
4	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos4
5	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos5
6	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos6
7	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cos7

表 2-4 CcfiInit 映射关系

CcfiInit	CcfiMap
0	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cfi0
1	iVtCcosMapSrm[iVtPortSrm.ccosMapIdx].cfi1

3. XlateKey

每个哈希表的索引值由对应的 XlateKey 表计算所得，XlateKey 表有 3 种，分别是 VlanKey、MacKey 和 MacIPBandKey 下面分别进行介绍。

xlateKey 表为 VlanKey 时，哈希表如下：

Name	Width	Bits	R/W	Def	Description
keyTp	2	60:59	R/W	0x0	xlate0 的查找类型：0：vlan xlate； 1：Vlan Mac; 2:macip bind
keyMode	3	58:56	R/W	0x0	key 组成类型
res	16	55:40	R/W	0x0	预留
inIsLag	1	39:39	R/W	0x0	指示输入的内部逻辑端口号是否为 LAG 端口，高有效
inLport	6	38:33	R/W	0x0	输入的内部逻辑端口号，当 inIsLag 有效时表示 LAG 端口，否则表示普通端口。
ccos	3	32:30	R/W	0x0	ccos 值
ccfi	1	29:29	R/W	0x0	ccfi 值
cvid	12	28:17	R/W	0x0	cvid 值
scos	3	16:14	R/W	0x0	scos 值
scfi	1	13:13	R/W	0x0	scfi 值
svid	12	12:1	R/W	0x0	svid 值
valid	1	0:0	R/W	0x0	有效指示

xlateKey 表为 MacKey 时，哈希表如下：

Name	Width	Bits	R/W	Def	Description
keyTp	2	60:59	R/W	0x0	xlate0 的查找类型：0：vlan xlate； 1：Vlan Mac; 2:macip bind
keyMode	3	58:56	R/W	0x0	key 组成类型
inIsLag	1	55:55	R/W	0x0	指示输入的内部逻辑端口号是否为 LAG 端口，高有效
inLport	6	54:49	R/W	0x0	输入的内部逻辑端口号，当 inIsLag 有效时表示 LAG 端口，否则表示普通端口。
smac	48	48:1	R/W	0x0	源 mac 地址
valid	1	0:0	R/W	0x0	有效指示

xlateKey 表为 MacIPBandKey 时，哈希表如下：

Name	Width	Bits	R/W	Def	Description
keyTp	2	60:59	R/W	0x0	xlate0 的查找类型：0：vlan xlate； 1：Vlan Mac; 2:macip bind

ipv4Ind	1	58:58	R/W	0x0	0:IPv6 的 SIP；1：ipv4 的 SIP
sip	57	57:1	R/W	0x0	ipv4 时，低 32 比特为 sip；ipv6 时为 SIP 的低位
valid	1	0:0	R/W	0x0	有效指示

XlateKey 表中包含的域信息获取 iVtXlateKeyCtlx(x 属于 0-7)的配置进行获取，可以配置 8 种不同的 Key 表生成模式，通过 iVtPortSrm 表中的 XlateKeyMode 进行选择。iVtXlateKeyCtlx 的配置信息如下：

Name	Width	Bits	R/W	Def	Description
usePort1	1	17:17	R/W	0x0	xlate1 查找是是否使用端口作为 key。1：使用
useCrange1	1	16:16	R/W	0x0	xlate1 查找时是否使用 cvid range 处理后的值。1：使用
useSrange1	1	15:15	R/W	0x0	xlate1 查找时是否使用 svid range 处理后的值。1：使用
useSvid1	1	14:14	R/W	0x0	xlate1 查找是是否使用 svid 作为 key。1：使用
useScos1	1	13:13	R/W	0x0	xlate1 查找是是否使用 scos 作为 key。1：使用
useScfi1	1	12:12	R/W	0x0	xlate1 查找是是否使用 scfi 作为 key。1：使用
useCvid1	1	11:11	R/W	0x0	xlate1 查找是是否使用 cvid 作为 key。1：使用
useCcos1	1	10:10	R/W	0x0	xlate1 查找是是否使用 ccos 作为 key。1：使用
useCcfi1	1	9:9	R/W	0x0	xlate1 查找是是否使用 ccfi 作为 key。1：使用
usePort0	1	8:8	R/W	0x0	xlate0 查找是是否使用端口作为 key。1：使用
useCrange0	1	7:7	R/W	0x0	xlate0 查找时是否使用 cvid range 处理后的值。1：使用
useSrange0	1	6:6	R/W	0x0	xlate0 查找时是否使用 svid range 处理后的值。1：使用
useSvid0	1	5:5	R/W	0x0	xlate0 查找是是否使用 svid 作为 key。1：使用
useScos0	1	4:4	R/W	0x0	xlate0 查找是是否使用 scos 作为 key。1：使用
useScfi0	1	3:3	R/W	0x0	xlate0 查找是是否使用 scfi 作为 key。1：使用
useCvid0	1	2:2	R/W	0x0	xlate0 查找是是否使用 cvid 作为 key。1：使用
useCcos0	1	1:1	R/W	0x0	xlate0 查找是是否使用 ccos 作为 key。1：使用
useCcfi0	1	0:0	R/W	0x0	xlate0 查找是是否使用 ccfi 作为 key。1：使用

XlateKey 的生成如图 2-5 所示，通过 iVtPortSrm.XlateEn 来使能对应的 Xlate 表，iVtPortSrm.XlateTp 决定使用哪种 XlateKey，iVtPortSrm.XlateKeyMode 来获取 iVtXlateKeyCtl 的表项，从而一起决定 XlateKey 的生成。得到 XlateKey 之后，通过哈希算法计算出 XlateSrm 表的索引值，通过查表获取用来添加或删除的 Vlan 信息和 VlanOpIdx，从而来对 vlan 进行编辑。

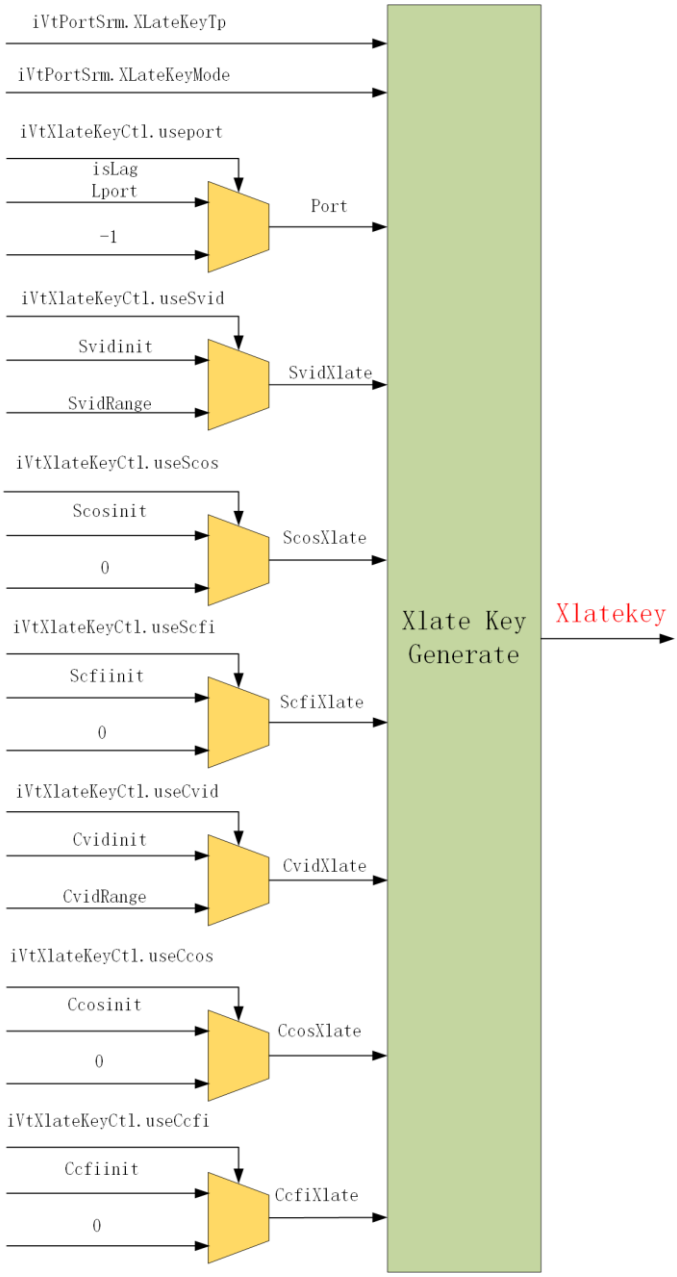


图 2-5 XlateKey 生成示意图

2.2.2.5 Port

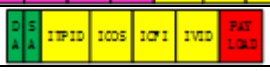
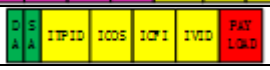
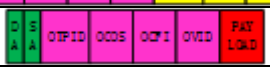
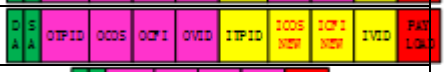
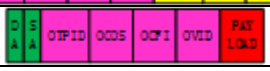
当 ProtoVlan、FlowVlan、和 XlateVlan 都没有命中时，使用端口默认的用于替换或添加的 svid、scos、scfi、cvid、ccos、cscfi 值和 VlanOpIdx 进行 VLan 编辑，通过端口索引查询表项 iVTPortSrm 获取。

2.2.3 Vlan Edit

当 iVtPortSrm 表中 iVtEditEn 位为 0 是，vlan 信息由 Vlan Get 获取；而当其为 1 时，通过 VlanOpIdx 索引来查找 iVtVlanOpSrm 表项来决定对应的编辑操作。需要添加或者替换的 vlan 信息(即 Newsvid、Newscos 、 Newscfi 、 Newcvid 、 Newccos 、 Newccfi) 和 VlanOpIdx 来自 iVtProtoVlanSrm 、iVtFlowVlanTcmSrm、iVtXlateLeftSrm[x](x 可取 0-3)、iVtXlateRightSrm[y](y 可取 0-3)，iVtPortSrm 中的一个，具体实现见相应章节，下面对 iVtVlanOpSrm 中 Vlan 编辑操作详细介绍。

2.2.3.1 DT Edit

DT 报文编辑之后的 Vlan 信息如下表所示，其中 defcvid 和 difsvid 来之 iVtPortSrm 的两个域。

Vlan 编辑之前报文	iVtVlanOpSrm 域	域值	Vlan 编辑之后报文
	dtPovid （packet.stag.vid=0） dtOvid （packet.stag.vid !=0）	0（nop）	报文不变
		1(copy)	
		2(replace)	
		3(delete)	
	dtOpri	0（nop）	报文不变
		1(copy)	
		2(replace)	
		3(delete)	
	dtPIvid （packet.ctag.vid =0） dtIvid （packet.ctag.vid !=0）	0（nop）	报文不变
		1(copy)	
		2(replace)	
		3(delete)	
	dtIpri	0（nop）	报文不变
		1(copy)	
		2(replace)	
		3(delete)	

2.2.3.2 SOT Edit

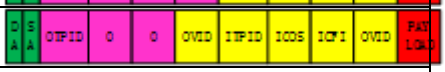
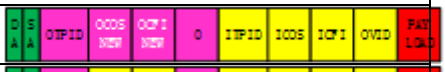
SOT 报文编辑之后的 Vlan 信息如表所示，其中 defcvid 和 difsvid 来之 iVtPortSrm 的两个域。

Vlan 编辑之前报文	iVtVlanOpSrm 域	域值	Vlan 编辑之后报文
	SotPovid （packet.stag.vid=0） SotOvid （packet.stag.vid !=0）	0（nop）	报文不变
		2(replace)	
		3(delete)	

	SotOpri	0（nop）	报文不变
		2(replace)	
		3(delete)	
	SotIvid	0（nop）	报文不变
		1(add)	
		2(copy)	
	SotIpri	0（nop）	报文不变
		1(add)	
		2(copy)	

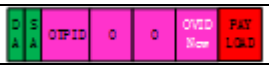
2.2.3.3 SIT Edit

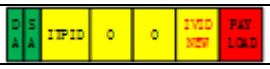
SIT 报文编辑之后的 Vlan 信息如表所示，其中 defcvid 和 difsvid 来之 iVtPortSrm 的两个域。

Vlan 编辑之前报文	iVtVlanOpSrm 域	域值	Vlan 编辑之后报文
	SitOvid	0（nop）	报文不变
		1（add）	
		2(copy)	
	SitOpri	0（nop）	报文不变
		1(add)	
		2(copy)	
	SitPivid（packet.stag.vid=0） SitIvid（packet.stag.vid !=0）	0（nop）	报文不变
		2(replace)	
		3(delete)	
	SotIpri	0（nop）	报文不变
		2(replace)	
		3(delete)	

2.2.3.4 UT Edit

UT 报文编辑之后的 Vlan 信息如表所示，其中 defcvid 和 difsvid 来之 iVtPortSrm 的两个域。

Vlan 编辑之前报文	iVtVlanOp Srm 域	域值	Vlan 编辑之后报文
	UtOvid	0（nop）	报文不变
		1（add）	
	UtOpri	0（nop）	报文不变

		1(add)	
	UtIvid	0（nop）	报文不变
		1（add）	
	UtIpri	0（nop）	报文不变
		1(add)	

2.2.4 Vlan 查找

经过 Vlan 翻译之后，得到 SvidInit 和 updSvid 两种不同的 Vlan 值，根据 iNetPortSrm.usePktSvid 和 iNetPortSrm.useUpdSvid 来进行选择，通过 Vlan 值去查询 iNetVlanSrm，从而获取该 Vlan 的一些配置信息，如图 2-6 所示。

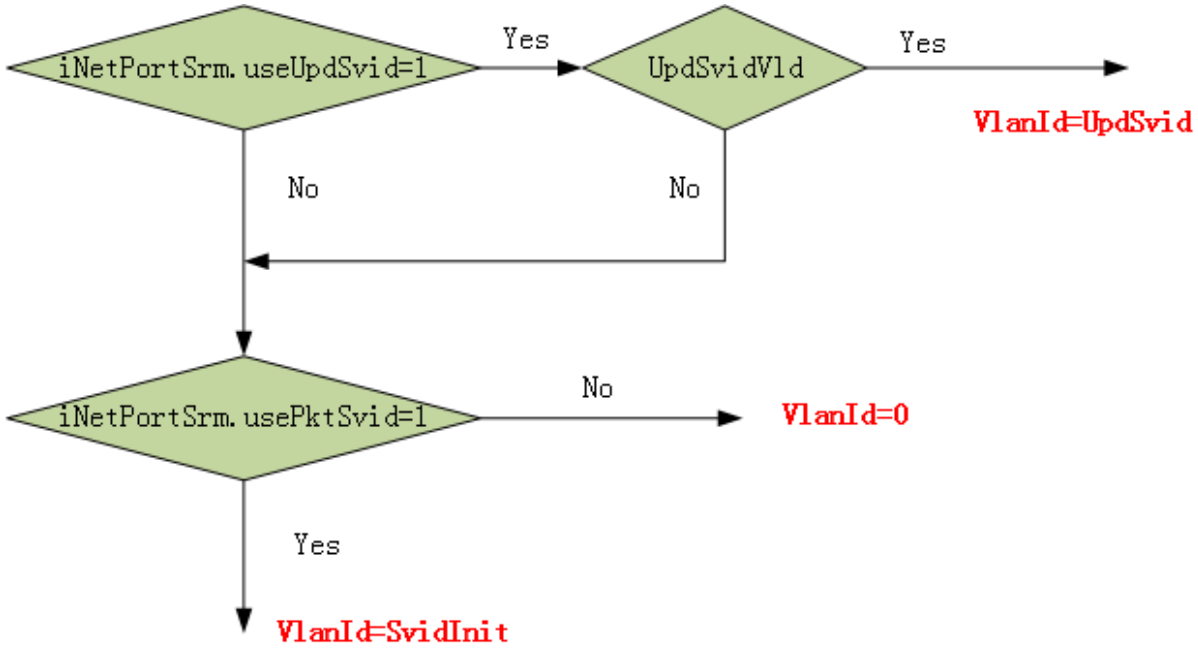


图 2-6 Vlan 查找功能流程图

通过 Vlan 编号可以查询对应的 iNetVLANSrm 表，来获取 Vlan 的各种属性值，如设置 Stp、Erps 功能防止环路，配置 Pbmp 限制广播域，qosProfileVld、qosProfileIdx 进行网络质量管理等。

当 iNetVLANSrm.valid=0 时，表示没有配置相应的 Vlan 属性表，为了报文正常转发，将使用默认的 Vlan 属性表，即 iNetDefVlanCtl。

2.2.5 Vlan Filter

当 iNetPortSrm.vlanFilterEn=1 时，使能入口 Vlan 滤波功能，如图 2-7 所示。当 isLag 置 1 时，表示该端口是 Lag 口，iNetPortSrm.lagBmp 中第 Lport 位为 0 时，将该端口的报文进行丢弃；当 isLag 置 0 时，表示该端口是普通口，当 iNetPortSrm.PortBmp 中第 Lport 位为 0 时，将该端口的报文进行丢弃。

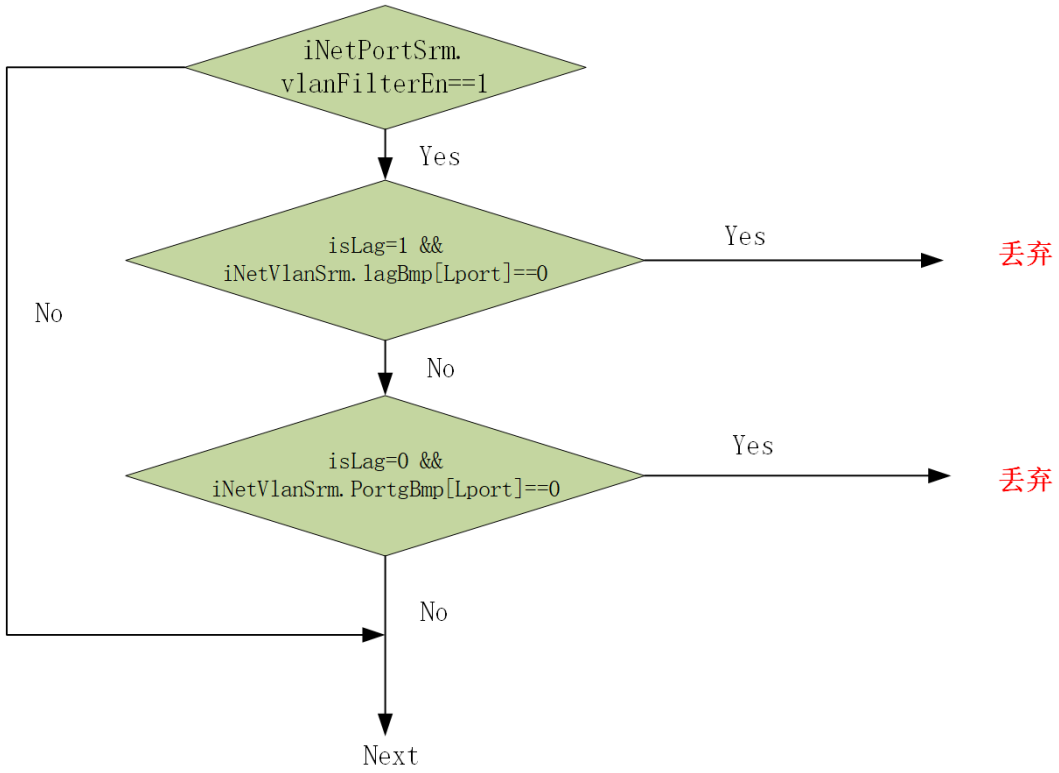


图 2-7 Vlan 滤波功能流程图

2.3 出口 Vlan

出口 Vlan 功能主要包括 EVlan 滤波、EVlan Get、EVlan 编辑几个部分功能，如图 2-8 所示。经过入口 Vlan 操作获取内部 VLan ID 之后，报文首先通过内部 VlanId 查找 eEeVLanSrm，当 eEeVlanSrm.valid=0 时，使用默认 Vlan 属性表 eEeDefVlanCtl。

当 eEePortSrm.vlanFilterEn=1 时，使能 EVlan 滤波，不在 Evlan 广播域的报文将被滤除。然后根据报文是否带 Tag 信息来获取 Evlan 信息，当报文缺少 EVlan 信息时，使用 eEeportSrm 中的默认 Vlan 信息。最后，根据 eEePortSrm.eVtEditEn 决定是否进行 EVlan 编辑，Evlan 编辑有 EXlateVlan 和 Eport 两种方式，通过查找对应的表项获取 VlanOpIdx 和用来添加或替换的 vlan 信息（svid、scos、scfi、cvid、ccos、ccfi），通过索引 VlanOpIdx 查询 iVtVlanOpSrm 来对原始报文进行编辑。各模块详细功能见下面各章节。

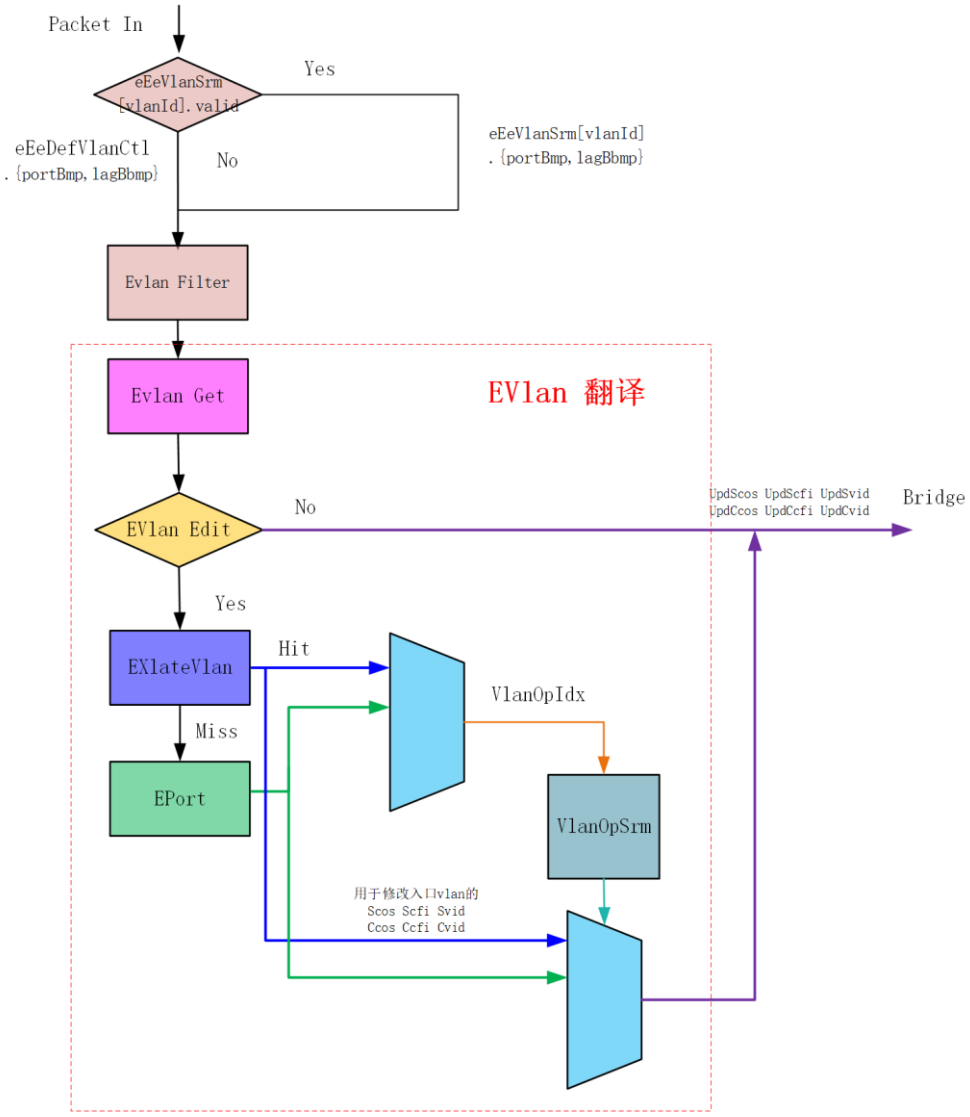


图 2-8 出口 VLAN 功能示意图

2.3.1 EVlan 滤波

当 eEePortSrm.vlanFilterEn=1 时，使能入口 Vlan 滤波功能，如图 2-9 所示。当 isLag 置 1 时，表示该端口是 Lag 口，eEePortSrm.lagBmp 中第 Lport 位为 0 时，将该端口的报文进行丢弃；当 isLag 置 0 时，表示该端口是普通口，当 eEePortSrm.PortBmp 中第 Lport 位为 0 时，将该端口的报文进行丢弃。

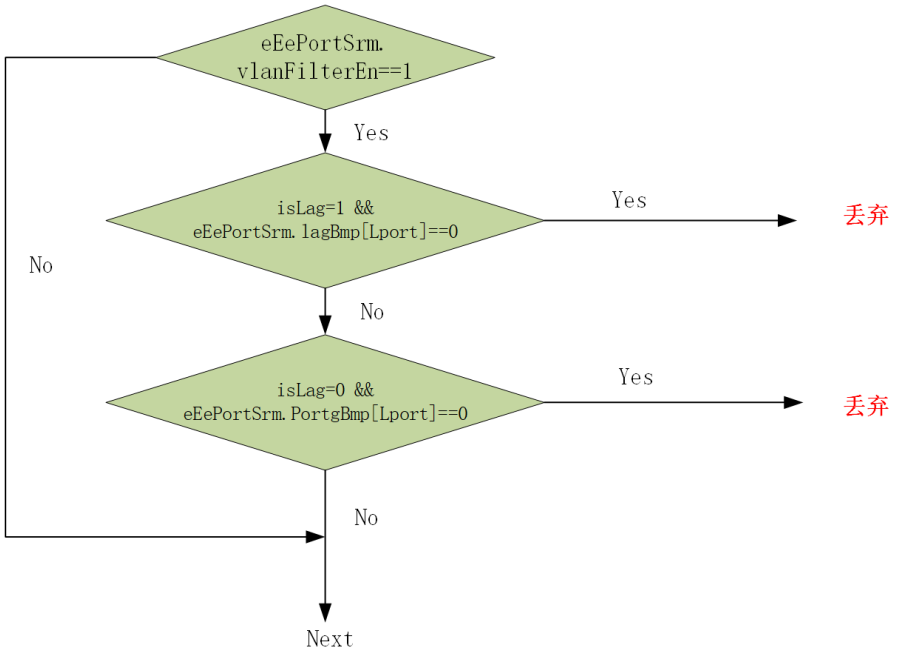


图 2-9 EVlan 滤波功能流程图

2.3.2 EVlan 翻译

2.3.2.1 Evlan Get

报文经过入口 VLan 相关操作之后，得到 Upd.{Svid、Scos、Scfi、Cvid、Ccos、Ccfi}，如果缺少 Stag 或者 Ctag，则使用对应出口的默认 vlan 信息，通过查表 eEePortSrm 获得。

2.3.2.2 EXlateVlan

EXlateVlan 和 XlateVlan 类似，也包含 EXlate0 和 EXlate1 两部分，每部分又包含 8 个哈希表，即 eEeXlateLeftSrm[x](x 在 0-3 之间)、eEeXlateRightSrm[y](y 在 0-3 之间)（8 个哈希表是为了防止哈希冲突）。但是不再包含 VlanRange 和 CosMap 功能，而且 Key 表有所不同，只有基于 Vlan 的 Key。除此之外，额外添加了 EXlateTcm 表功能。

对于 EXlate0 和 EXlate1，通过 EXlateKey 和设置的哈希算法（eEePortSrm.AlgTp），可以算出对应 EXlateSrm 表的索引，从而得到用于替换或添加的 svid、scos、scfi、cvid、ccos、cscfi 值以及 VlanOpIdx。而对于 EXlateTcm 表，可以通过 ExlateKey 和 EXlateTcm 进行掩码查找，得到 EXlateTcmsrm 表的索引值，从而得到用于替换或添加的 svid、scos、scfi、cvid、ccos、cscfi 值以及 VlanOpIdx。

1. EXlate

EXlate0 和 EXlate1 对应的 Key 表如下：

Name	Width	Bits	R/W	Def	Description
eVtTp	2	41:40	R/W	0x0	evt 的 key 类型
outIsLag	1	39:39	R/W	0x0	指示输出的内部逻辑端口号是否为 LAG 端口，高有效

outLport	6	38:33	R/W	0x0	输出的内部逻辑端口号，当 outIsLag 有效时表示 LAG 端口，否则表示普通端口。
ccos	3	32:30	R/W	0x0	ccos
ccfi	1	29:29	R/W	0x0	ccfi
cvid	12	28:17	R/W	0x0	cvid
scos	3	16:14	R/W	0x0	scos
scfi	1	13:13	R/W	0x0	scfi
svid	12	12:1	R/W	0x0	svid
valid	1	0:0	R/W	0x0	有效指示

EXlateKey 表中包含的域信息获取 eEeXlateKeyCtlx(x 属于 0-3)的配置进行获取，可以配置 4 种不同的 Key 表生成模式，通过 EXlateKey 表中的 eVtTp 进行选择。eEeXlateKeyCtlx 的配置信息如下：

eEeXlateKeyCtlx

eVtEn0	1	16:16	R/W	0x0	eVt0 查找使能，高有效
eVtEn1	1	15:15	R/W	0x0	eVt1 查找使能，高有效
eVtTcamEn	1	14:14	R/W	0x0	eVt tcam 查找使能，高有效
xlate0UsePort	1	13:13	R/W	0x0	xlate0 是否使用 port，高有效
xlate1UsePort	1	12:12	R/W	0x0	xlate1 是否使用 port，高有效
xlate0UseSvid	1	11:11	R/W	0x0	xlate0 是否使用 svid，高有效
xlate1UseSvid	1	10:10	R/W	0x0	xlate1 是否使用 svid，高有效
xlate0UseScos	1	9:9	R/W	0x0	xlate0 是否使用 scos，高有效
xlate1UseScos	1	8:8	R/W	0x0	xlate1 是否使用 scos，高有效
xlate0UseScfi	1	7:7	R/W	0x0	xlate0 是否使用 scfi，高有效
xlate1UseScfi	1	6:6	R/W	0x0	xlate1 是否使用 scfi，高有效
xlate0UseCvid	1	5:5	R/W	0x0	xlate0 是否使用 cvid，高有效
xlate1UseCvid	1	4:4	R/W	0x0	xlate1 是否使用 cvid，高有效
xlate0UseCcos	1	3:3	R/W	0x0	xlate0 是否使用 ccos，高有效
xlate1UseCcos	1	2:2	R/W	0x0	xlate1 是否使用 ccos，高有效
xlate0UseCcfi	1	1:1	R/W	0x0	xlate0 是否使用 ccfi，高有效
xlate1UseCcfi	1	0:0	R/W	0x0	xlate1 是否使用 ccfi，高有效

EVlanXlate Key 的生成如图 2-10 所示，通过 EVlan Get 得到 Upd.{Svid、Scos、Scfi、Cvid、Ccos、Ccfi}信息后，通过 eEeXlateKeyCtl 的相应配置信息可以得到 EVlanXlate Key 表，通过哈希算法得到 eEeXlateSrm 的索引信息，通过查表获取用来添加或删除的 Vlan 信息和 VlanOpIdx，从而来对 Evlan 进行编辑。

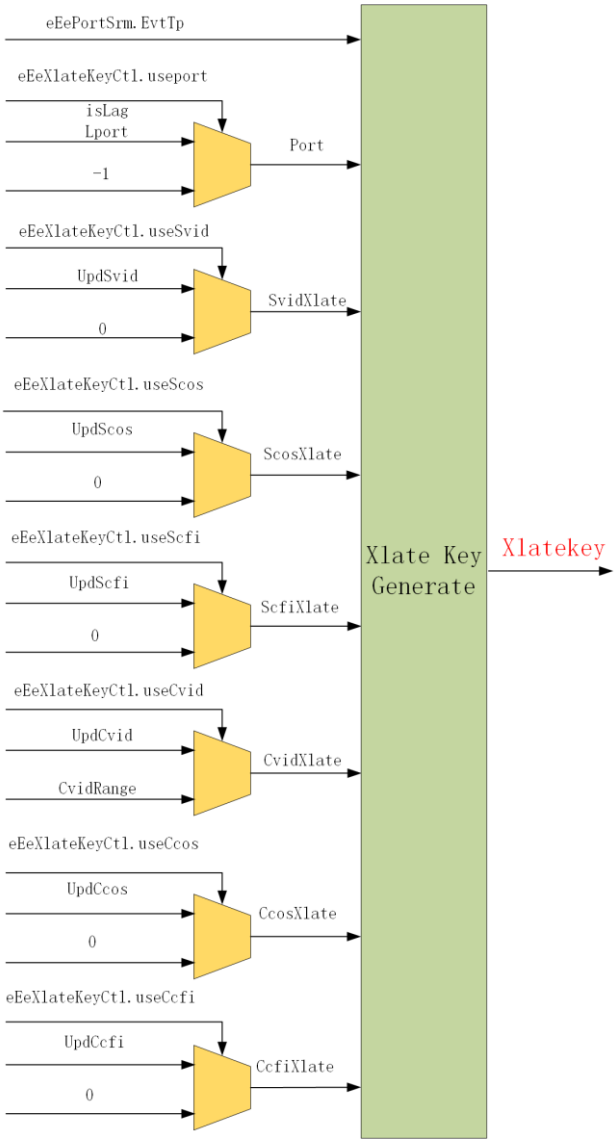


图 2-10 EVlanXlate Key 的生成方式

2. EXlateTcm

通过 eEeXlateKeyCtlx.eVtTcamEn=1 来使能 Tcam 表查找，TcamKey 包含的域如下表所示，其中 Vlan 相关域来自 EVlan Get。当 TcamKey 和 eEeXlateTcm 匹配时，得到 eEeXlateTcmSrm 的索引值，通过查询对应 eEeXlateTcmSrm 得到用来修改的 Vlan 信息和 VlanOpIdx。

TcamKey 域构成

res	3	0x0	
outIsLag	1	0x0	指示输出的内部逻辑端口号是否为 LAG 端口，高有效
outLport	6	0x0	输出的内部逻辑端口号，当 outIsLag 有效时表示 LAG 端口，否则表示普通端口。
svid	12	0x0	Upd.svid

scos	3	0x0	Upd.scos
scfi	1	0x0	Upd.scfi
cvid	12	0x0	Upd.cvid
ccos	3	0x0	Upd.ccos
ccfi	1	0x0	Upd.ccfi
eVtTp	2	0x0	evt 的 key 类型

2.3.2.3 EPort

当 EXlateVlan 没有命中时，使用端口默认的用于替换或添加的 svid、scos、scfi、cvid、ccos、cscfi 值和 VlanOpIdx 进行 EVLan 编辑，通过端口索引查询表项 eEePortSrm 获取。

2.3.3 EVLAN Edit

EVLAN 编辑功能和 VLAN Edit 一样，也是对 VLan 进行添加、修改、删除等操作，只不过是由入口 Vlan 变成出口 Vlan，这里不再赘述。

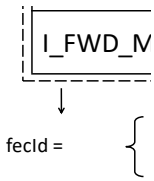
3 L2 转发

报文经过前面的 vlan 处理后，会进入到 l2 转发模块。总体来说，目前所获得的关键信息是：报文 smac，报文 dmac，vlan 信息，（准确来说是 I_NET_VLAN_SRM 的 fid 信息）等。结合这些信息，再加上我们关于 l2 功能的相关配置，报文会在芯片内进行 l2 转发。L2 转发总的来说，即 1）转发 2）学习 3）老化 的功能，以及衍生出的站点移位，黑白名单等功能，下面就分章节一一描述。

3.1 Bridging

如下图所示，报文进入到转发模块时，会提取 dmac+vid(fid)进行转发。通过 hash，得到一个 hash index 去查询 mac_key_srm。 由于 hash 涉及到 hash 冲突，所以这里采用左 4 加右 4 共 8 个槽位去解决冲突问题。所以得到的 hash index 会在这 8 个槽位对应的 index 位置都查找一遍，只要查到有对应表项的 valid 为 1 的，说明表项有效，然后再把 hash key(dmac+fid)和表项中的 mac key 对比一遍，相同的话，说明 I_FWD_MAC_KEY_SRM 表命中，报文走已知单播逻辑。接着就会去通过 I_FWD_MAC_KEY_SRM 命中表项的 index 去取 I_FWD_MAC_SRM 表，（I_FWD_MAC_SRM 和 I_FWD_MAC_KEY_SRM 是一一对应的）从其中去取对应的如转发出端口等信息指导后续的转发行为（mac 表同样可以标记丢弃，指示颜色等行为，这些域可以通过配置静态 mac 表插入，而不是通过学习的方式指定。）；如果 I_FWD_MAC_KEY_SRM 表未命中，后续则走泛洪逻辑。这里另外说明下，2 层组播逻辑同样也是通过插入组播类型的 mac 表来实现。但这里查出的 I_FWD_MAC_SRM.fecpath 的含义就是组播组 id，后续会用来作为 iDstMcGrpSrm 表的索引得到 portBmp 进行洪泛转发。

Mac 地址的组织分为 svl 和 ivl 两种模式。Ivl 指的是在不同 vlan 里，mac 地址互相独立；svl 指的是所有 vlan 共用同一张 mac 地址表。我们的实现方式是：在 I_NET_VLAN_SRM 表里，有一个叫做 fid 的域，形成了一个 vid--->fid 的映射。然后转发查找 mac key 时，是采用的 fid 作为 hash key。当希望实现 ivl 模式时，可采用 vid--->fid 一对一映射的方式来实现；当希望实现 svl 模式时，可采用 vid--->fid 多对一映射的方式来实现。



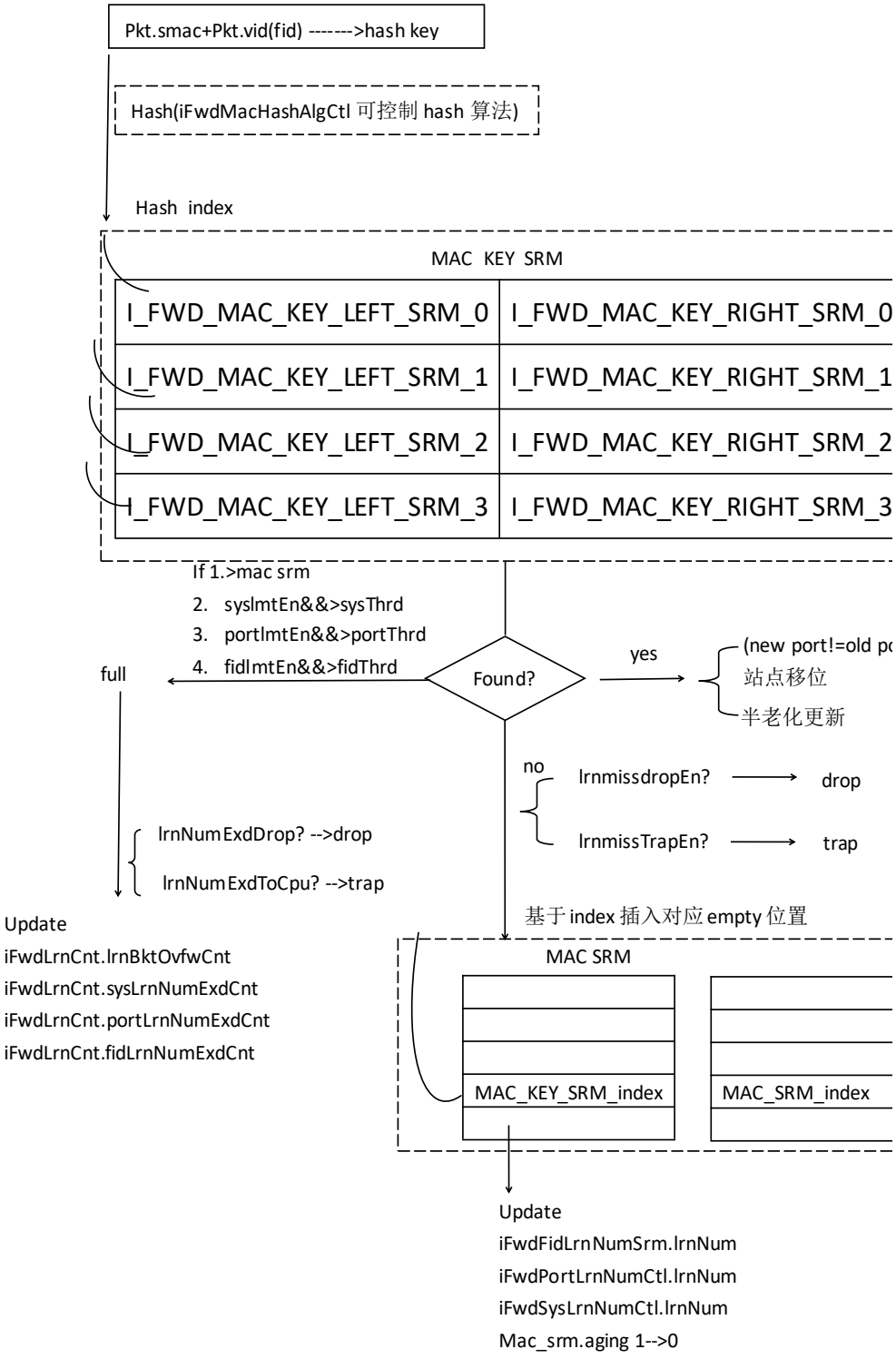
3.2 Learning

如下图所示，学习过程开始时，和转发类似，也是要查找 mac 表。报文进来后，会基于报文的 smac 地址去查询 mac 表。

命中：会判断是否站点移位或者老化更新，这个在后面会再介绍；

未命中且未溢出：会将 smac 和入端口插入对应的表项，即为学习。同时更新相关学习计数的表项。

未命中且溢出：如果此时 mac 表项已满，或者超出了基于 port，fid 以及 system 设置的 limit，则不会被学习，会将对应的溢出计数加一。也可以基于全局控制是否关闭学习。



3.3 Aging

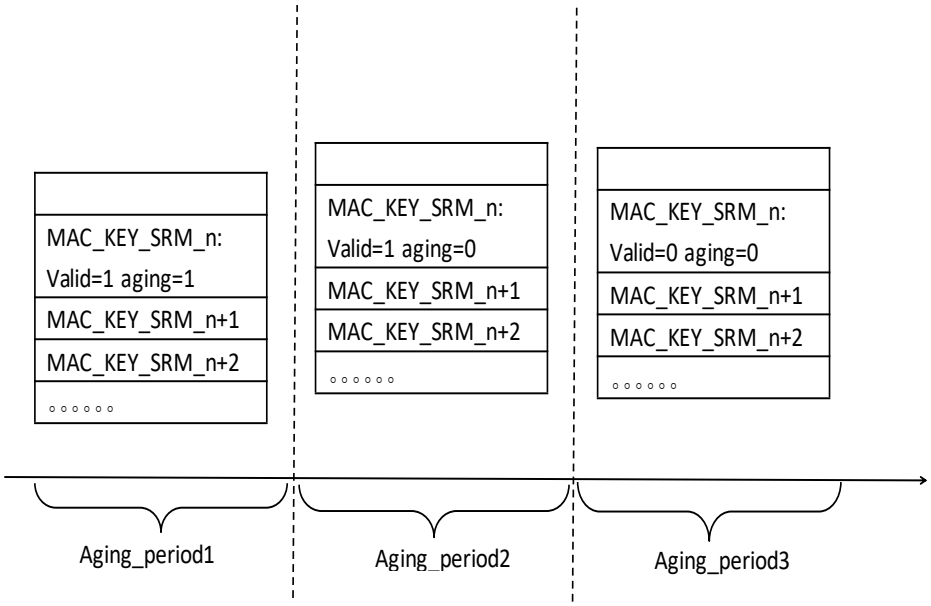
老化过程如下图所示。由于老化周期是到了老化周期时间点，统一判断，所以为了避免某些表项刚学习到就马上碰到了老化时间点，通常老化采用两步法进行老化：

如下图，每到了一个老化周期节点，芯片做两件事：1）会将表项中所有 Valid=1 且 aging=1 的项，将其 aging 置为 0；1）会将表项中所有 Valid=1 且 aging=0 的项，将其 Valid 置为 0（注：Valid=1，则 mac 表表示有效，会在 learning 和 forwarding 阶段起作用；Valid=1 表示 mac 无效，相当于该条目被老化。）那么 mac 表中所有学习到的表项都会经历：1）刚学习到时 Valid 和 aging 都被置 1；2）碰到第一次老化周期时间点时，会将 aging 置为 0，Valid 保持为 1；2）碰到第二次老化周期时间点时，Valid 被拉为 0，达到老化。

老化时间：

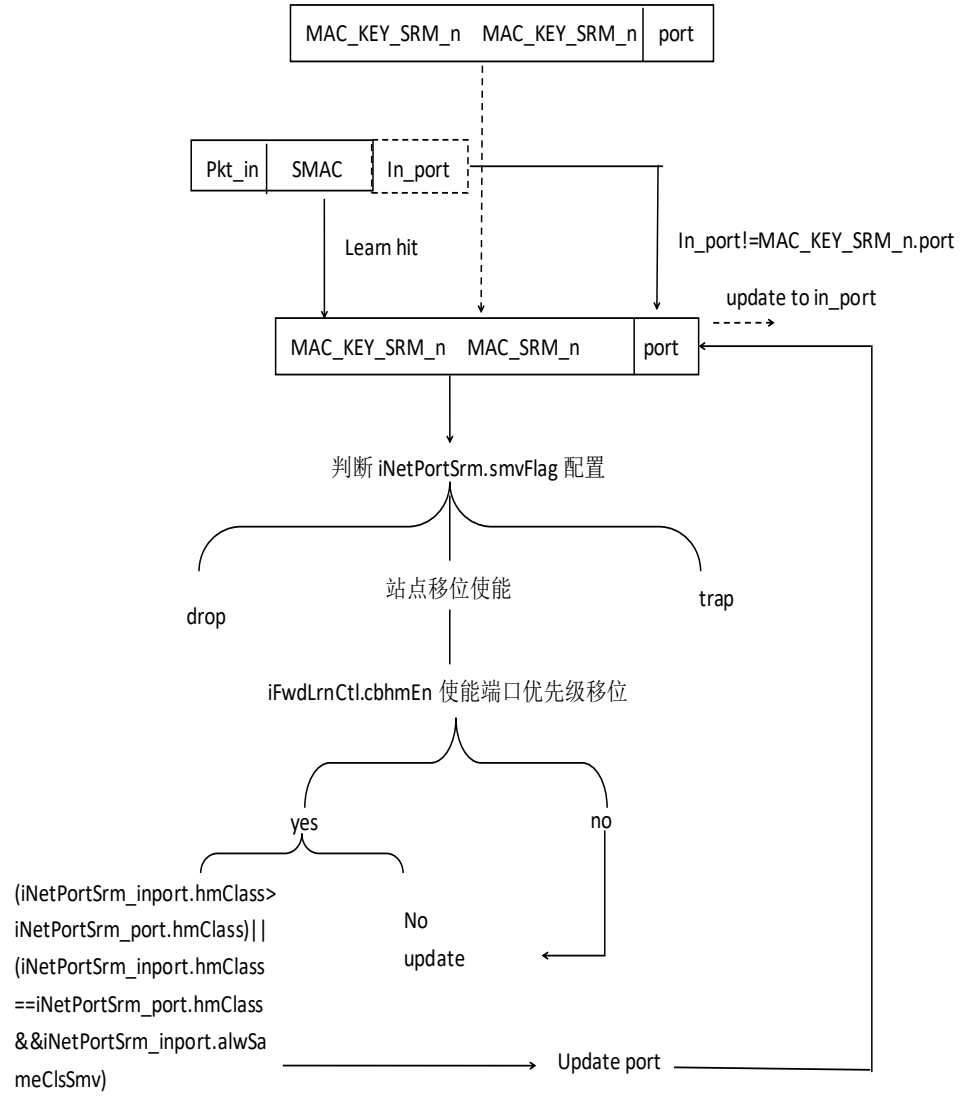
iFwdAgingTimerCtl.counterThrd 和 iFwdAgingCtl.sCounterThrd 决定了老化时间（默认 5 分钟），这个值不是直接等于时间，这个比例关系 sdk 代码里已处理好。

可以基于全局或者端口控制是否关闭老化，iFwdAgingCtl.agingEn 以及 iFwdAgingPortCtl.en(位图)。另外，芯片也支持快速老化，iFwdFastAgingCtl.fastAgingEn 写 1 即可老化掉当前所有学习到的 mac 地址（关闭老化的端口除外）。另外，静态地址（MAC_SRM.static ==1）的地址不会被老化。



3.4 站点移位

当入报文走学习逻辑时，smac 命中了原有 mac 表项，但入端口和表项中的端口不一致时，就会产生站点移位逻辑。如上图所示，根据配置，可决定此时是否进行站点移位，或者是否进行端口优先级站点移位，或者是否将尝试站点移位的报文 trap(to cpu).(注：此处决定是否 trap，但不决定 trap 的具体端口。可采用 dp trap set port=31 表示指定 trap 为 cpu 口，31 口为我们的内部 cpu 口)。此外，上述的 drop 和站点移位使能两者是冲突的，如果 drop 置位了，也不会进行站点移位。


MAC_KEY_SRM_n MAC_KEY_SRM_n port
Pkt_in SMAC In_port
Learn hit
In_port!=MAC_KEY_SRM_n.port
update to in_port
MAC_KEY_SRM_n MAC_SRM_n port
判断 iNetPortSrm.smvFlag 配置
drop
站点移位使能
trap
iFwdLrnCtl.cbhmEn 使能端口优先级移位
yes
no
(iNetPortSrm_inport.hmClass>
iNetPortSrm_port.hmClass)||
(iNetPortSrm_inport.hmClass
==iNetPortSrm_port.hmClass
&&iNetPortSrm_inport.alwSa
meClsSmv)
No
update
Update port

4 Acl 原理

芯片包括多个流识别引擎，其中包括 ACL 和 EACL。数据流经过 ACL 模块时会先确定哪些字段作为匹配项，并以 Key 的形式传递给查找引擎，只有存在与 Key 匹配的条目时，才执行相应的策略引擎行为。

4.1 Key 组成

Key 可以包含从数据包中解析出的信息，如 MACSA/MACDA、SRCIP/DSTIP、VLANID 或者 TCP/UDP 端口号等。还可以包含反映包交换状态的字段，如 DROP 指示报文、回环报文类型（如组播环回、镜像环回）等。

ACL 可以基于端口、L2、L3、L4 的内容进行流分类。其中针对不同的输入包类型，可以有不同的 Key 组合，包含 MacKey、Ipv4Key、Ipv6Key、MixKey，各种 Key 包含的具体内容如下表所示。

表 4-1 ACL_KEY

包类型	内容
Mac	DMAC[47:0]，SMAC[47:0]，STAG[15:0]，CTAG[15:0]， ETHERTYPE[15:0]，PORTS[34:0]，L3UDF[7:0]等
IPv4	DIP[31:0]，SIP[31:0]，PROTOCOL[7:0]，TOS[7:0]，PORTS[34:0]， L4SrcPort[15:0]，L4DsrPort[15:0]，L3UDF[7:0]，L4UDF[7:0]等
IPv6	DIP[127:0]，SIP[127:0]，PROTOCOL[7:0]，TOS[7:0]， PORTS[34:0]，L4SrcPort[15:0]，L4DsrPort[15:0]，L3UDF[7:0]，L4UDF[7:0]，TTL[7:0]， DMAC[47:0]，TTL[7:0]，TCPFLAG[7:0]，STAG[15:0]，CTAG[15:0]等
Mix	DIP[127:0]，SIP[127:0]，PROTOCOL[7:0]，TOS[7:0]， PORTS[34:0]，L4SrcPort[15:0]，L4DsrPort[15:0]，L3UDF[7:0]，L4UDF[7:0]，TTL[7:0]， DMAC[47:0]，SMAC[47:0]，ICMPCODE[7:0]，ICMPTYPE[7:0]，TTL[7:0]，TCPFLAG[7:0]， STAG[15:0]，CTAG[15:0]、IPOPTION[0:0]，IPHDRERR[0:0]，L3TP[3:0]，L4TP[2:0]等

4.2 查找引擎

查找引擎采用的是 TCAM 查表方式。每个 TCAM 表有多个条目，每个条目（有时称为规则）根据 Key 检查数据，该条目包含一个相应的逐位掩码字段，允许在匹配过程中忽略数据的任何比特位（Don't Care）。这很重要，因为用户可以由此确定给定字段的哪一部分是关键的，而其他部分将被忽略。只有当所有未忽略位匹配时，查找引擎才会认为匹配。

匹配过程从查找引擎的第一个条目开始，如果第一个条目未匹配则查找下一条目，直至查找到匹配条目为止，这时查找引擎将相应条目的偏移量发送到策略引擎。

查找引擎的条目由四个 TCAM 组成，这四个 TCAM 彼此垂直地排列在一起。这些 TCAM 一起决定了查找引擎的深度（例如，512 个条目），这种默认模式称为单宽模式。为了提供更大的灵活性，TCAM 还可以配置为双宽模式和四宽模式。

双宽模式允许将底部 TCAM 配置在上部 TCAM 的右侧，形成左右定位。为了让给定的条目匹配，Key 必须匹配 TCAM A 和 TCAM B。这种配置的优点是 Key 和匹配项的宽度是原来的两倍，允许在匹配条件中使用更多的位。但是这个配置意味着可用的条目数减半。所以如果单宽模式提供 512 个条目，那么双宽模式将有 256 个条目。

同理，四宽模式下 Key 的宽度是单宽模式的四倍，只有所有的四个 TCAM 同时匹配才能执行相应行为动作，此时可用的条目数为单宽模式的四分之一。

4.3 策略引擎

策略引擎包含所有与查找引擎的匹配条目相关联的操作。每个策略引擎包含的条目数和查找引擎相同。查找引擎中匹配条目的偏移量（或索引）指示被用作策略引擎的索引，以获取行为动作信息。

策略引擎的行为包含添加、修改、删除 VLANTAG，命中计数统计，丢弃报文，重定向报文，上送 CPU，采样，指定队列号，指定基于流的限速模板，指定 QOS 模板等。

4.4 查找过程

ACL 查找过程如图 4-1 所示。

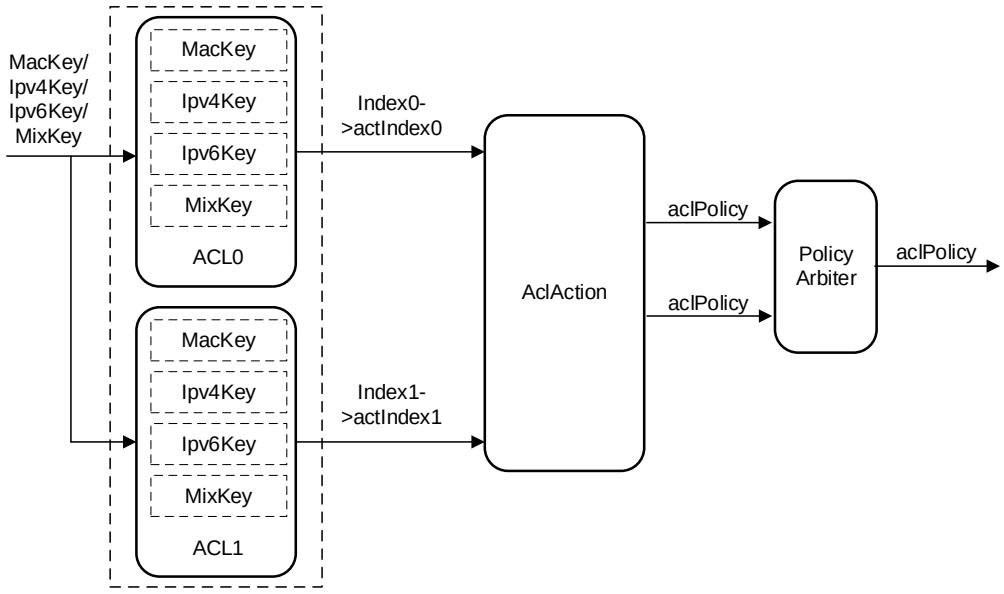


图 4-1 ACL 查找过程

ACL 包括两个流查找引擎 ACL0 和 ACL1，共享所有的 ACL 查找条目，条目数为 512 条。可以根据配置将 iAclTcm 表划分为 MacKey、Ipv4Key、Ipv6Key 和 MixKey。Key 类型是由 iAclTcm[index].key[159:158] 这 2bit 决定的，取值为 0 时，表示当前 index 条目为 MacKey；为 1，表示为 Ipv4Key；为 2，表示为 Ipv6Key；为 3，表示为 MixKey。每种 Key 类型分配的条目数通过 iAclCtl 寄存器配置， iAclCtl 寄存器组成如图 4-2 所示。

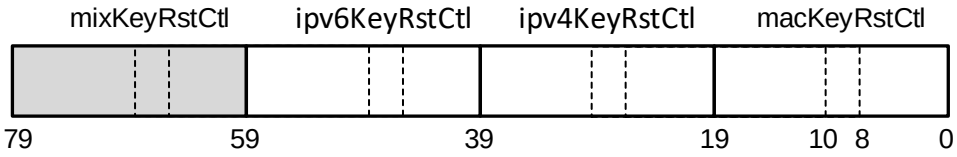


图 4-2 iAclCtl 寄存器组成示意图

iAclCtl 寄存器由四个域组成，分别为 macKeyRstCtl、ipv4KeyRstCtl、ipv6KeyRstCtl、mixKeyRstCtl，每个域有 20bit。这些域的值能确定 iAclTcm 表各种 Key 的条目分配。以 macKeyRstCIt 域为例进行说明，组成为{indexBase[8:0]，keySize[1:0]，tableBase[8:0]}，其中 indexBase 表示 MacKey 类型在 iAclTcm 表中的基址，keySize 表示 Key 宽模式（0 表示单宽模式，1 表示双宽模式，2 表示四宽模式），tableBase 表示该 Key 类型对应的 AclAction 行为表（iAclTcmSrm）的基址（一般情况下 tableBase 值与 indexBase 相等）。根据相邻 Key 类型的 indexBase 之差可以计算出某 Key 类型占用的条目数。例如，假设 MacKey 的 indexBase 为 0，与之相邻的 Ipv4Key 的 indexBase 为 100，则 MacKey 所占用 iAclTcm 条目数为 100 条。

ACL 的查找过程是根据数据报文的不同类型和配置选择 ACL0 和 ACL1 两个流引擎的查找 Key（MacKey/Ipv4Key/Ipv6Key/MixKey），其中 ACL0 和 ACL1 的查找 Key 可以分别包含不同的 ACL 模板。查找操作完成后，如果报文匹配了 ACL0 和 ACL1 的条目，则分别返回索引 index0 和 index1，通过 actIndex = (index-indexBase)>>keySize+tableBase 运算获取到行为表 iAclTcmSrm 的查找索引 actIndex0 和 actIndex1，其中 iAclTcmSrm 表中存储该流的处理行为。根据 TCAM 的特性，每个查找引擎只会输出最先匹配的项，因此在应用时需要根据条目的优先级进行配置，将优先级高的条目配置在靠前的条目中。

两个 ACL 查找引擎会获取该业务流的两条处理行为，此时需要进行仲裁，仲裁的原则是如果两条处理行为项之间存在冲突，则以 ACL1 查找引擎对应的处理行为为准。

ACL TCAM 查找的详细业务处理流程如图 4-3 所示。

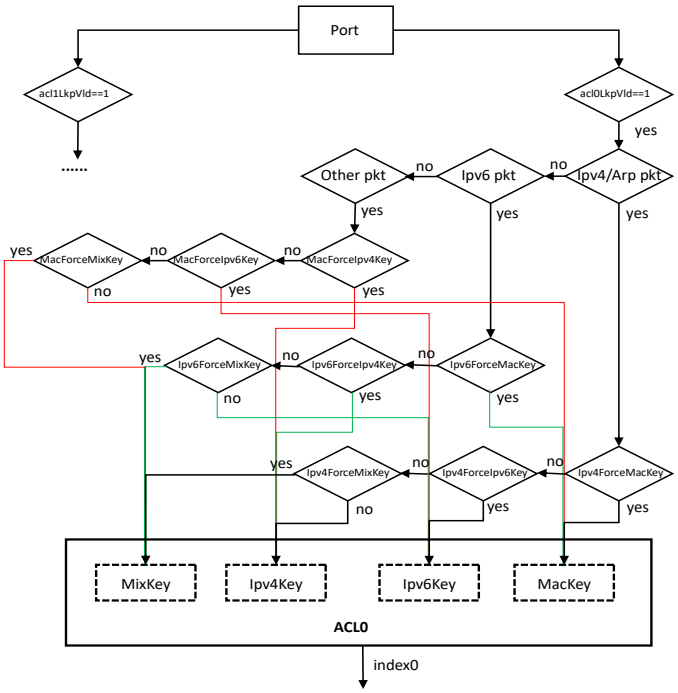


图 4-3 ACL TCAM 业务处理流程

基于端口开启 ACL0 和 ACL1 的查找开关，以 ACL0 为例进行说明。如果报文类型为 IPv4 或 Arp 时，默认采用 Ipv4Key 查找；如果报文类型为 IPv6 时，默认采用 Ipv6Key 查找；如果报文类型为非 Ipv4、Arp 和 Ipv6 时，默认采用 MacKey 查找。也可以基于端口来控制是否强制采用 MacKey、Ipv6Key 或者 MixKey 进行 ACL 查找。

EACL 的 Key 组成及查找方式同 ACL 基本一致。

5 Qos map and policy

5.1 Map

芯片内部优先级产生模块示意图如图 5-1 所示，按优先级从高到低依次为 Policing 模块的优先级映射，Acl 指定优先级，Bridge 中 mac 表命中后修改优先级，vlan 转发表中指定内部优先级，vlan 翻译中修改优先级，内部优先级主要用于后续的报文 vlan 和 dscp 的重标记，TM 模块中的队列调度。

其中应用最多的使通过 policing 模块的 map 功能将入向报文的 L2、L3 优先级映射成报文在芯片流水中的内部优先级。具体流程如下：

根据 iVtPortTblAct 表中设置的 priVld 为 0，执行 policing 模块中的 pri map 流程，根据 iPolQosProSrm 表中设置的优先级映射模板，判定映射使用的是 L2 头还是 L3 头（如果要用 L3 头，报文的 L3 type 必须是 IPV4 或者 IPV6），如果使用 L2 头的话，是使用 stag 还是 ctag 中的 cos 作为映射初始值。

使用 L3 头部信息映射，根据 iPolQosProSrm 表中的表索引指针 phbPtr 和 L3 头部信息中的 dscp 值计算出 DSCP 映射表的索引{phbPtr，dscp}，通过 tos 的低 2bit 得到映射后的内部优先级和颜色；

使用 L2 头部信息映射，根据模板中的 stag 和 ctag 使用信息来判定是使用 scos 还是 ccos，根据模板中的索引指针和 L2 头部信息计算出 cos 映射表的索引{phbPtr，cos，cfi}，通过该索引查 iPolCosPriMapSrm 表得到映射后的内部优先级和颜色。

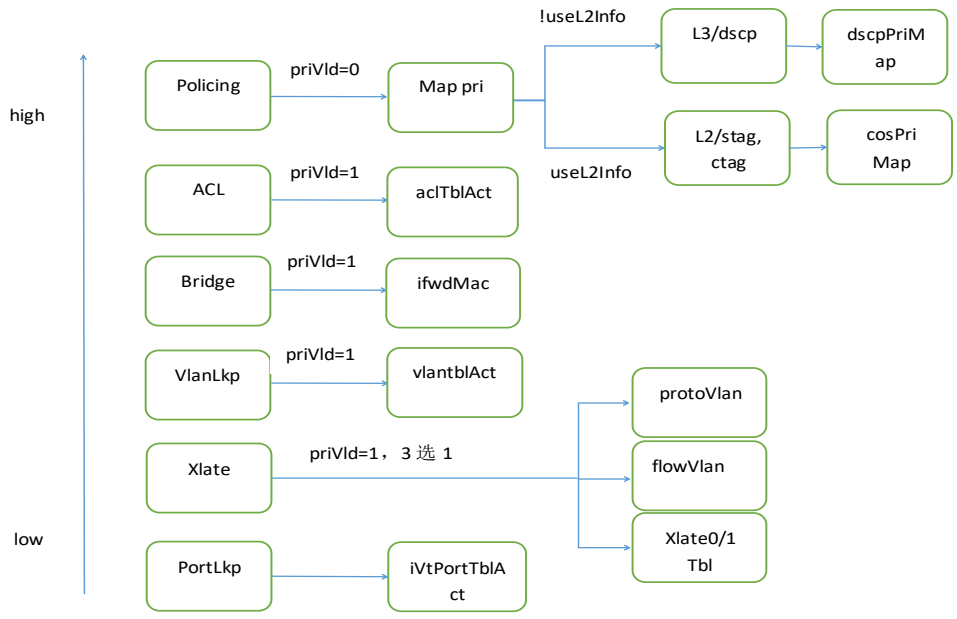


图 5-1 芯片内部优先级产品模块示意图

5.2 ReMark

Remark 功能主要是将报文在芯片内部的优先级重标记到出去的报文中，修改出去报文的优先级。此芯片 Remark 功能支持 L2 和 L3 即 vlan 和 tos 的优先级重标记，在 eDstRmkInfoSrm 表中能配置 scos、ccos 和 tos 的重标记使能标识，也能够在重标记 tos 的情况下只修改报文的 dscp。其中很重要的一点是，重标记 tos 必须是在报文的 L3 type 是 IPV4 和 IPV6 的条件下才能生效。

通过配置 eDstRmkInfoSrm 表中的重标记索引 index，结合报文在芯片内部的优先级 pri，得到 eDstPriRmkSrm 表的索引{index，pri}，该表项中存在三组 tos、pri 和 cfi 值，分别对应不用颜色（绿、黄、红）报文的优先级重标记值。

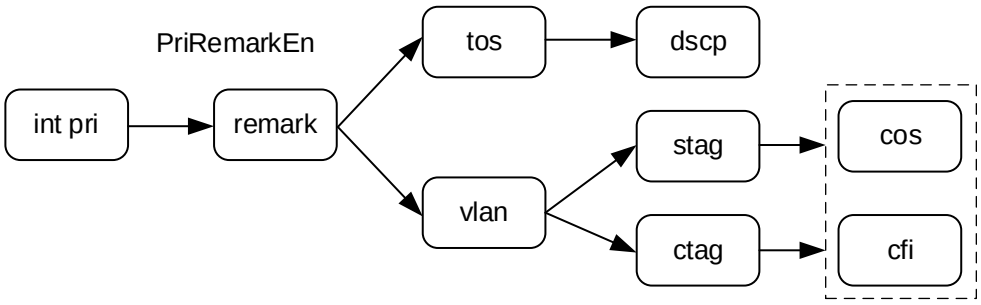


图 5-2 Remark 功能示意图

5.3 Policing

此芯片支持两种 policing 类型，基于端口和基于 flow 的 policing，并支持两者的层次化叠加。其中单层 policing 支持 srTCM 和 trTCM 两种模式，支持色盲和色敏感两种模式。层次化 policing 支持三种模式：MIN_ONLY，MAX_ONLY 和 MIN_MAX。

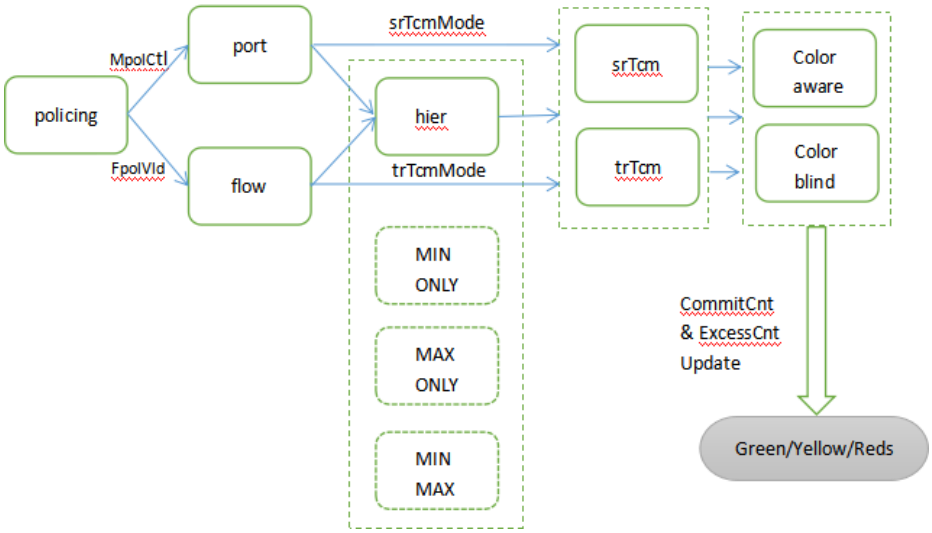


图 5-3 policing 类型介绍

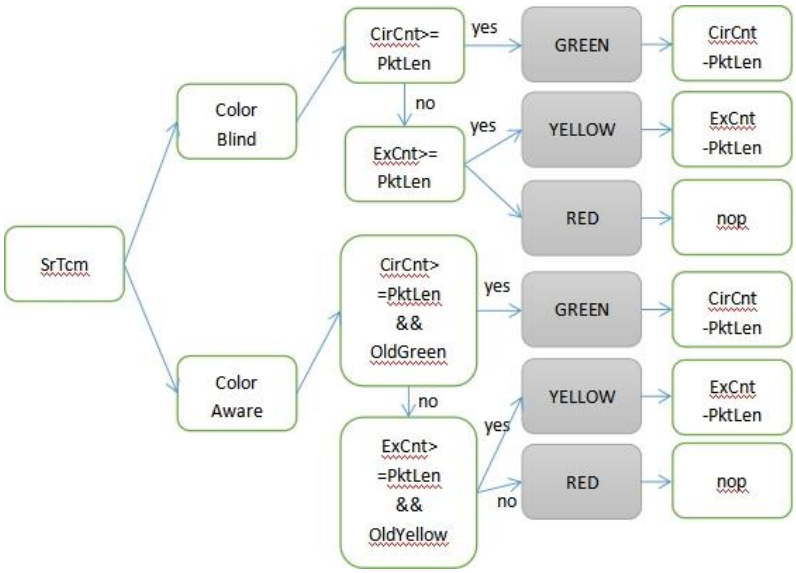


图 5-4 SrTcm 模式介绍

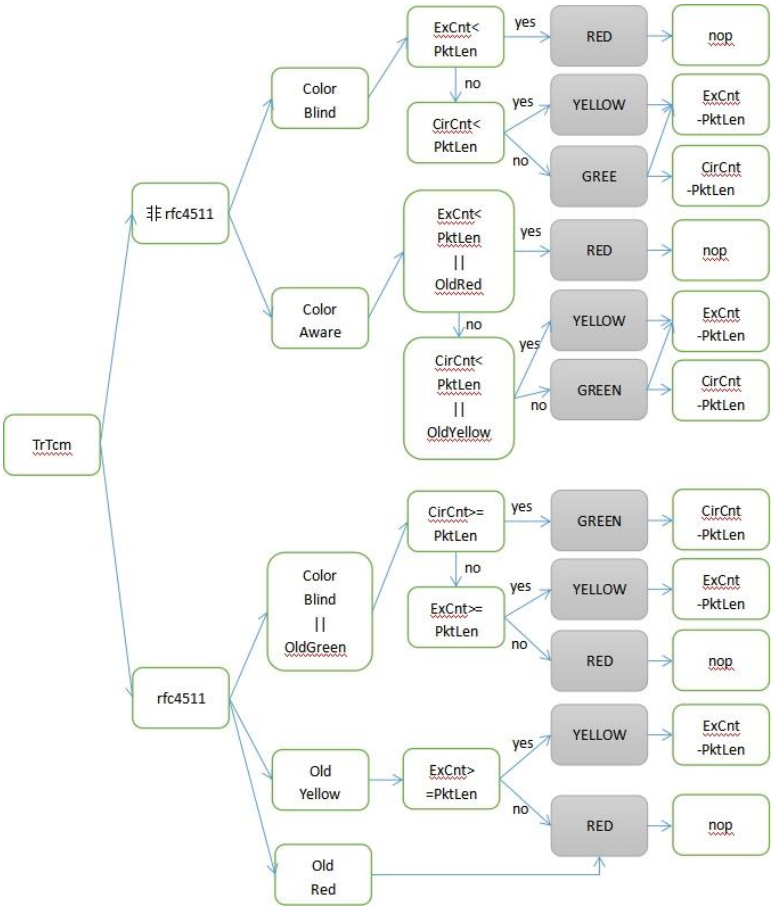


图 5-5 TrTcm 模式介绍

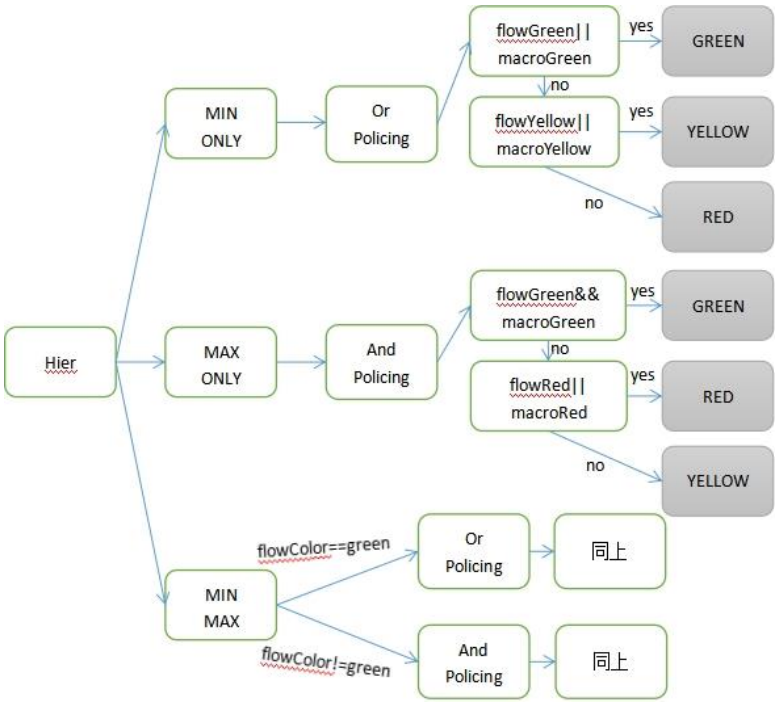


图 5-6 Hier 模式介绍

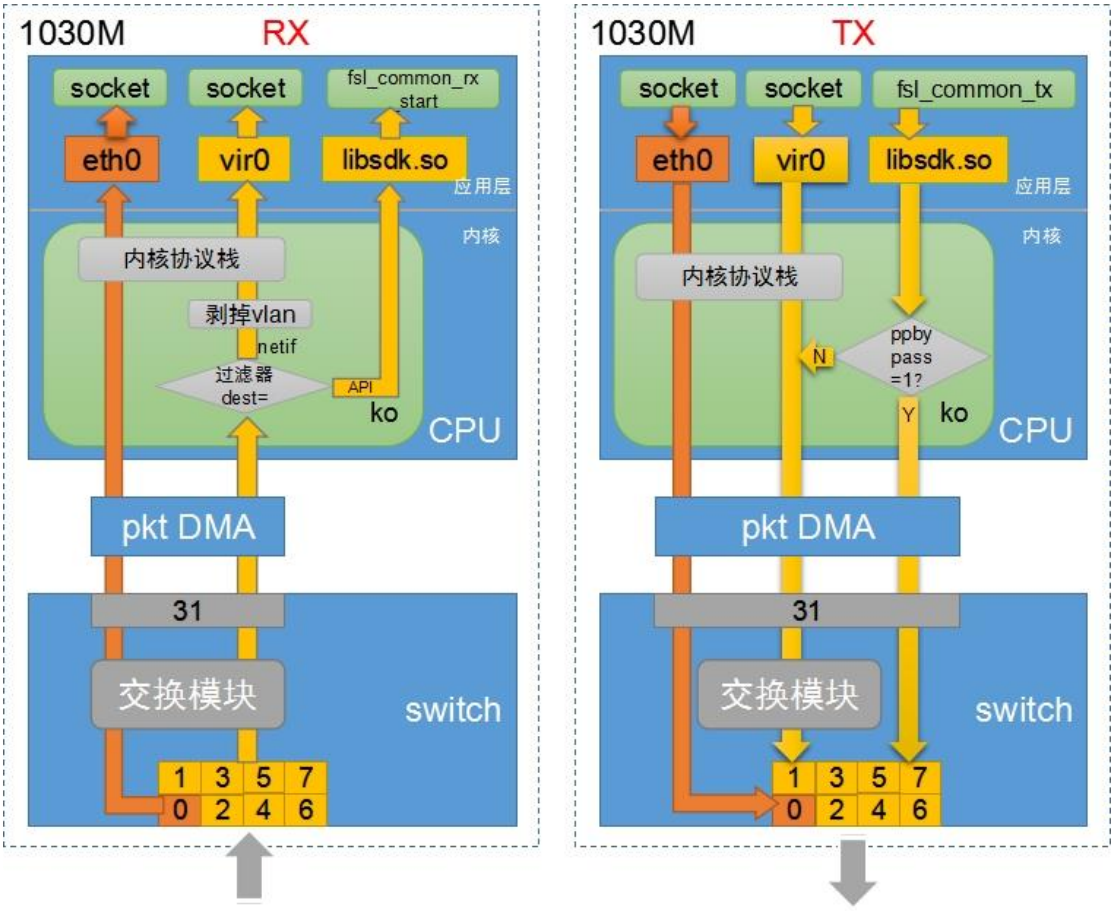
Policing 的限速值和芯片的时钟频率息息相关。在 ipolFlowUpdCtl 和 ipolMacroUpdCtl 中。能够设置令牌桶的填充使能，令牌桶的更新周期等参数（timer0，timer1，timer0Num，timer1Num），这些参数和时钟频率一起决定了 iPolFlowMeterSrm 和 iPolMacroMeterSrm 中 commitRate、excessRate、cbs、ebs 等的控制粒度。具体的计算公式如下：

base=clock*(timer0Num+timer1Num)/(timer0*timer0Num+timer1*timer1Num)

真实的 c 桶和 e 桶的更新速率为：commitRate*8*base，excessRate*8*base，单位为 bit/s。

芯片的使用场景固定以后，clock 是固定值，可以通过修改 timer0、timer1、timer0Num 和 timer1Num 来调整 cir 和 eir 的更新颗粒，达到调整限速精度和限速范围的目的。timer0 和 timer1 不能设置得太小，如果需要填充令牌的条目过多，time0 和 timer1 设置过小会导致无法在 time0 和 time1 的时间内完整的填充所有条目，出现限速不准，建议设置值在 2000 以上。

6 Cpu 收发包



1、eth0和vir0为网络驱动（xy1000_net.ko）创建的2个网络设备，有独立的ip地址和mac地址。
2、API (fsl_common_rx_start 和 fsl_common_tx) 为 libsdk.so 提供的收发包接口函数，该接口通过 netlink 将包绕过协议栈进行收发。
3、eth0处理经由调试网口，且经过协议栈的包。
4、vir0处理经由非调试网口，且经过内核协议栈的包。
5、API处理经由非调试网口，且不经过内核协议栈的包。
6、该调试网口可任意设置。

7 修订信息

修订时间	版本	描述
2021.5.8	V1.0	初始版本。
2021.5.21	V1.1	新增总体框图。
2022.12.23	V1.2	内容优化。