**相关命令行**

**基本qinq**

dot1q-tunnel

no dot1q-tunnel

给tag和untag报文都打上一层vlan。

需要配置接口access或trunk模式，access模式下配置access vlan，trunk模式下配置native vlan。加的一层vlan就是配置的access vlan或这native vlan。

**接口cos覆盖**

dot1q-tunnel cos override

no dot1q-tunnel cos override

使能后，基本qinq打上的一层vlan的优先级设置为本命令指定的优先级。本命令生效需要先使能基本qinq。

**匹配cvlan或cvlan+cos，加一层vlan**

switchport vlan-mapping both cvlan 100 add-outer 200 cos 3 translate 300

如果单层vlan 100，那么加上外层vlan 200，且cos值为3，同时将内层vlan修改为300，变成双层vlan (200, 300)转发。

如果双层vlan(100, 200)，外层vlan是100，那么再加一层vlan 200，且cos值为3，同时将原报文外层vlan 100修改为200，变成三层vlan(200,300,200)转发。

switchport vlan-mapping both cvlan 100 cos 2 add-outer 200 cos 3 translate 300

本条命令对vlan的修改同上，只不过在匹配入口vlan时，需要同时匹配vlan的优先级为2。

**匹配cvlan或cvlan+cos，修改vlan**

switchport vlan-mapping both cvlan 100 add-outer 200 cos 3 remove

如果单层vlan 100，那么修改vlan为200，且cos值为3，变成vlan 200转发。

如果双层vlan(100, 200)，那么修改外层vlan 为200，且cos值为3，还是双层vlan(200, 200)转发。

switchport vlan-mapping both cvlan 100 cos 2 add-outer 200 cos 3 remove

本条命令对vlan的修改同上，只不过在匹配入口vlan时，需要同时匹配vlan的优先级为2。

**匹配inner vlan加一层vlan**

switchport vlan-mapping both inner 100 add-outer 200 cos 2

如果单层vlan 100，那么加上一层vlan 200，且优先级指定为2，变成vlan(200,100)转发。

如果双层vlan (100,200)，那么再加上一层vlan 200，且优先级指定为2，变成三层vlan(200，100,200)转发。

配置时，inner vlan可以选择多个vlan。

**接口default-cvlan**

switchport qinq default-cvlan 100

这条命令是配置接口的默认cvlan，需要跟双向灵活qinq配合使用，如下：

switchport qinq default-cvlan 100

switchport vlan-mapping both inner 100 add-outer 200

实现的效果就是：

如果入接口的报文是tag 100的，那正常按照下面的灵活qinq加一层tag，变成（200，100）转发；

如果入接口的报文是untag的，那么就给报文打上设置的default-cvlan，再按灵活qinq匹配vlan 100的规则打上一层200，变成（200，100）转发。

**匹配outer vlan，修改外层vlan**

switchport vlan-mapping both outer 200 translate 300 cos 3

如果单层vlan 200，那么转换为vlan 300，且优先级为3转发。

如果双层vlan (200,100)，那么外层vlan转换为300，且优先级为3，变成vlan (300,100)转发。

**匹配outer vlan和inner vlan修改内外侧vlan**

switchport vlan-mapping both outer 200 inner 100 translate outer 300 inner 200 cos 3

匹配双层vlan (200, 100)，修改内外侧vlan为(300, 200)，同时指定外层vlan的有限级修改为3转发。

**匹配单层vlan，N:1转换**

switchport vlan-mapping both 100,200 translate 300

匹配vlan 100和200，转换为vlan 300转发。反向vlan 300的报文从该接口出去时转换为vlan 100和200两条流转发。

**匹配untag报文加一层vlan**

switchport vlan-mapping both untag add-outer 200 cos 2

匹配untag报文，加上一层vlan 200，优先级指定为2转发。

反向vlan 200的报文从该接口出去时剥掉vlan，untag转发。

**匹配untag报文加两层vlan**

switchport vlan-mapping double-tagging untag add svlan outer 200 cos 2 inner 100 cos 3

匹配untag报文，加上外层vlan 200，且优先级为2，加上内层vlan 100，且优先级为3转发。

反向vlan (200, 100)的报文从该接口出去时，剥掉两层vlan，untag转发。反向匹配双层vlan时不匹配cos。

**匹配cvlan加两层vlan**

switchport vlan-mapping double-tagging cvlan 100 add svlan outer 300 cos 2 inner 200 cos 1

如果单层vlan 100，则再加外层vlan 300，且cos值为2，第二层vlan 200，且cos值为1，最内层vlan不变仍为100，变成三层vlan (300,200,100)转发。

反向vlan (300, 200)的报文从该接口出去时，剥掉两层vlan，untag转发。

反向vlan (300, 200, 100) 的报文从该接口出去时，剥掉最外面两层vlan，变成vlan100转发。

**匹配指定优先级的报文配置添加外层vlan**

switchport vlan-mapping both priority-tagged cos 1 add-outer 100 cos 2

匹配vlan 0，优先级为1的报文添加一层vlan 100，且外层vlan优先级指定为2，变成vlan (100, 0)转发

**匹配指定优先级的报文配置添加svlan，转换cvlan**

switchport vlan-mapping both priority-tagged cos 1 add-outer 100 cos 2 translate 200

//TODO

**配置框图**

//TODO

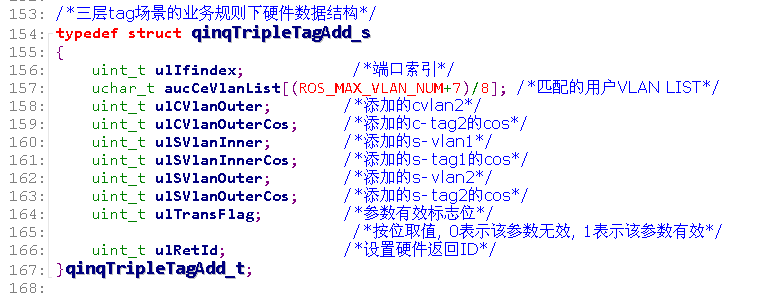
**主要数据结构**

**qinqVlanAddTranslate\_t**



常用的单双层vlan的qinq配置都是通过这个数据结构向ssp下发。

**qinqTripleTagAdd\_t**



**m\_list\_VlanQinq**

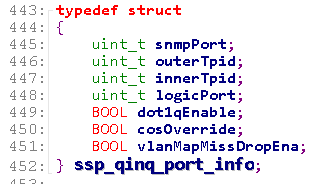
灵活qinq相关规则配置都保存到这个全局链表。

**pQinqPortInfo**

用来保存接口的基本qinq相关配置，以及命令行mls double-tagging tpid配置的接口tpid。当接口的基本qinq使能状态或者cos覆盖值发生变化时，更新这个全局变量。

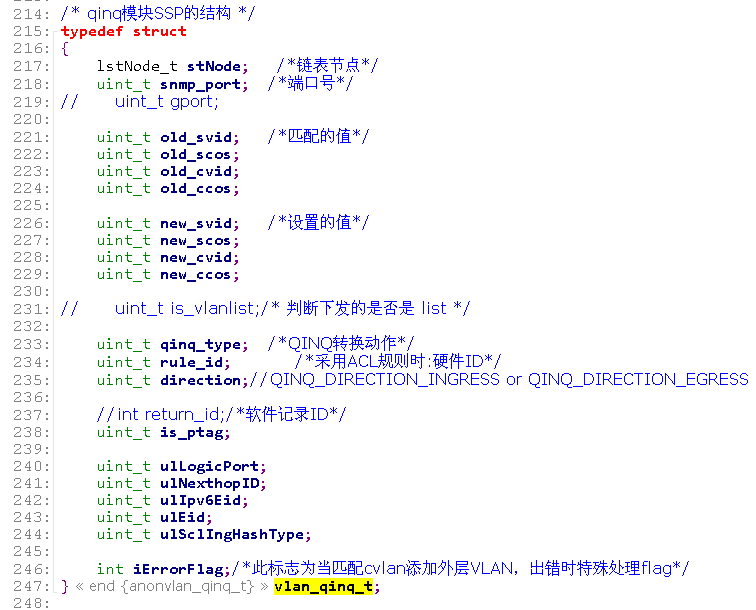
trunk组qinq这种配到所有成员口上的功能，也需要更新这个全局变量。

**ssp\_qinq\_port\_info**



全局变量pQinqPortInfo的类型。

**vlan\_qinq\_t**



ssp层用来保存的qinq规则信息，平台下发的qinqVlanAddTranslate\_t，ssp解析之后下发规则，然后将这些规则内容通过这个vlan\_qinq\_t结构保存到ssp本地，保存到的对象就是全局链表：m\_listVlanQinq。

规则的添加和删除都需要操作这个链表。

**ssp函数接口**

**ctc\_tms\_set\_port\_qinq\_enable：使能接口的基本QINQ功能**

STATUS ctc\_tms\_set\_port\_qinq\_enable(uint\_t snmp\_port, uint\_t ulEnable)

使能接口的基本qinq功能，需要先配置接口模式为access或trunk，如果是access模式，则需要配置access vlan；如果是trunk模式，则需要配置native vlan。、

对tag和untag报文都打上一层vlan。加的一层vlan具体是多少，得看配置的access vlan或者native vlan。

**ctc\_tms\_set\_port\_cos\_override\_enable：使能接口优先级覆盖功能**

STATUS ctc\_tms\_set\_port\_cos\_override\_enable(uint\_t snmp\_port, uint\_t ulEnable)

使能接口的优先级覆盖功能。这条命令需要使能接口的基本qinq功能，加的一层vlan的优先级为本命令指定的cos值。不配本条命令时，加的vlan的优先级默认为0。

**ctc\_tms\_set\_port\_miss\_drop\_enable：使能接口未匹配上规则的报文**

STATUS ctc\_tms\_set\_port\_miss\_drop\_enable(uint\_t snmp\_port, uint\_t ulEnable)

使能接口丢弃不能与灵活 QinQ 或 VLAN 转换规则匹配的 Tagged 报文。比如配置匹配vlan 100加一层vlan 200的规则，接口允许vlan 100，200报文通过。使能该功能后，如果接口收到vlan 200的报文，则丢弃；如果不使能该功能，收到的vlan 200报文正常转发。

**ctc\_tms\_set\_dtag\_sptpid：配置外层 VLAN Tag 的 TPID 值**

STATUS ctc\_tms\_set\_dtag\_sptpid(uint\_t ulTpid)

配置后，接口匹配上规则的qinq或vlan转换，会将外层vlan的tpid修改为配置的tpid。

**ctc\_tms\_set\_dtag\_cetpid：配置内层 VLAN Tag 的 TPID 值**

STATUS ctc\_tms\_set\_dtag\_cetpid(uint\_t ulTpid)

配置后，接口匹配上规则的qinq或vlan转换，会将内层vlan的tpid修改为配置的tpid。

**ctc\_tms\_vlan\_mapping\_ingress\_set：灵活qinq入方向配置的主要使用接口**

STATUS ctc\_tms\_vlan\_mapping\_ingress\_set(qinqVlanAddTranslate\_t \*vlan\_mapping\_info)

大多数灵活qinq命令行都是使用这个函数接口配置入方向的规则。匹配untagged报文加两层也使用这个接口。

**ctc\_tms\_vlan\_mapping\_egress\_set：灵活qinq出方向配置的主要使用接口**

STATUS ctc\_tms\_vlan\_mapping\_egress\_set(qinqVlanAddTranslate\_t \*vlan\_mapping\_info)

大多数灵活qinq命令行都是使用这个函数接口配置出方向的规则。

**ctc\_tms\_vlan\_mapping\_untag\_sadd\_ingress\_set：匹配untag加一层vlan**

STATUS ctc\_tms\_vlan\_mapping\_untag\_sadd\_ingress\_set(qinqVlanAddTranslate\_t \*vlan\_mapping\_info)

接口入方向匹配untagged报文，加一层vlan，可以选择指定加的vlan的优先级。

**\_qinq\_save\_ingress\_data：保存使用vlan\_mapping方式的规则**

int \_qinq\_save\_ingress\_data(ctc\_vlan\_mapping\_t vlan\_mapping, qinqVlanAddTranslate \*vlan\_mapping\_info, int hash\_type, uint\_t sclEid, uint\_t sclIpv6Eid)

灵活qinq有多个配置命令，部分命令下发匹配规则时是直接使用vlan\_mapping的方式匹配，相关的配置保存到本地使用这个接口。

**\_qinq\_save\_tcam\_ingress\_data：保存使用scl方式的规则**

int \_qinq\_save\_ingress\_data(qinqVlanAddTranslate \*vlan\_mapping\_info, uint\_t eid, uint\_t ipv6\_eid)

与使用vlan\_mapping实现匹配不同，部分qinq命令使用创建scl表项的方式来实现匹配，相关配置保存使用这个接口。最后保存到本地的一个全局链表：m\_list\_VlanQinq，与\_qinq\_save\_ingress\_data一样。

还有一些命令行配置的规则使用其他类似接口保存规则，也是保存到全局链表：m\_list\_VlanQinq。这里不多列举。

**ctc\_tms\_scl\_tcam\_create：创建scl tcam表项**

STATUS ctc\_tms\_scl\_tcam\_create(int unit, ctc\_scl\_tcam\_keyfield\_t \*pstKey, ctc\_scl\_vlan\_edit\_t \*vlan\_edit, uint\_t \*ulSclEid)

使用scl tcam表项匹配的规则，需要通过这个函数创建相应的表项。比如匹配单层cvlan加一层vlan时，使用的是vlan-mapping接口，而匹配cvlan+cos加一层vlan时，需要使用这个函数，通过tcam表项实现。

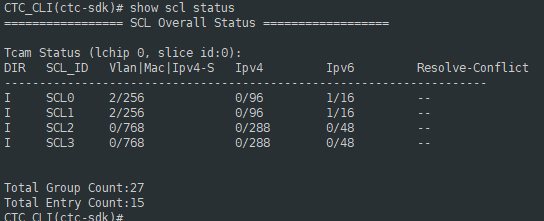
第一个参数是要匹配的vlan信息，以及key的类型，第二个参数是vlan操作的类型，是添加、删除，还是修改，第三个参数是创建的scl表项的索引，作为返回值传出。

**sdk相关（示例和debug命令）**

**sdk相关命令**

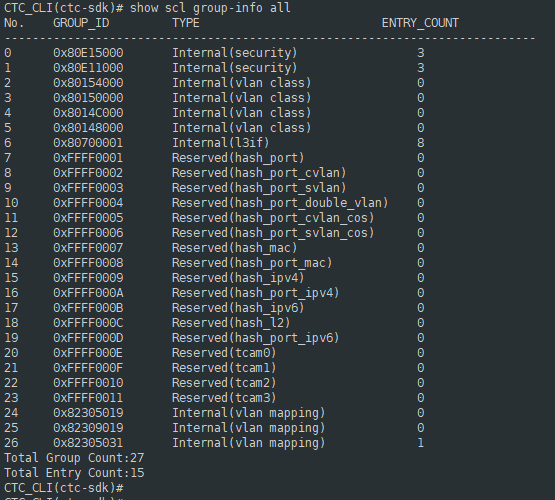
**show scl status**

查看scl的group和entry的使用情况。



**show scl group-info all**

查看所有group的类型，下图是设备初始化创建的所有group。



其中，第三列TYPE，是该group对应的使用模块，比如：

（1）vlan class就是使能基于mac或者IP子网划分vlan需要用到的group。

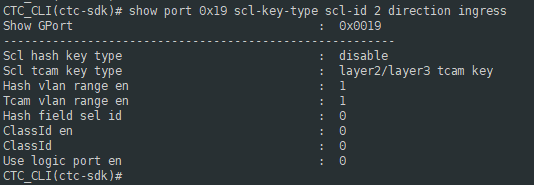
（2）tcam0到tcam3是ssp初始化的四个tcam表项，其中tcam0和tcam1用来做hash查表，hash类型在代码ctcScl.c的初始化函数ctc\_tms\_port\_scl\_init()中可以看到，由全局数组g\_sspIgrSclHashType给出。

（3）tcam2和tcam3是disable了hash type的，使用tcam查表。查表类型由全局变量g\_sspSclTcamType给出。目前初始化是使用IP表项。

（4）其他一些灵活qinq的配置则使用的是倒数第二三行的vlan mapping。

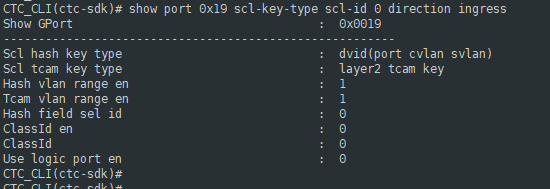
**show port 0x19 scl-key-type scl-id 2 direction ingress**

查看接口scl的配置，可以看到hash type是disable的，tcam的类型则是IP表项。



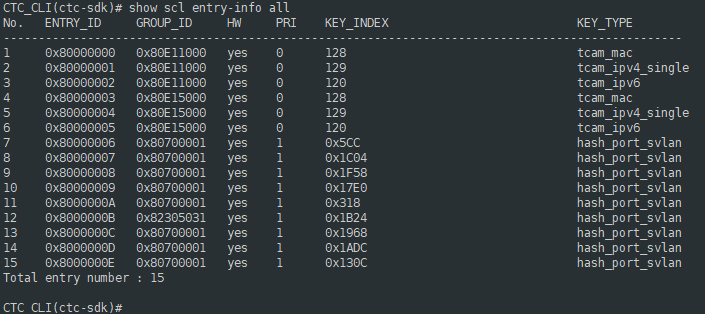
**port 0x19 scl-key-type scl-id 0 direction ingress type dvid tcam-type mac**

设置接口的scl查表类型，hash查表，匹配两层vlan，查表类型为mac。



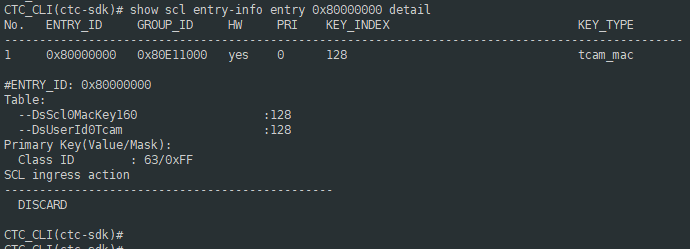
**show scl entry-info all**

查看所有添加的entry表项，不同group下的entry都显示。比如上条命令可以看到id为0x80E11000的group添加了三条entry表项，在下图中就可以看到这三条entry的信息，如果要查看单条entry的详细信息，使用下一条命令。



**show scl entry-info entry 0x80000000 detail**

查看指定entry的详细信息，配置灵活qinq时可以看到ingress和egress的匹配和vlan编辑信息。



**qinq的sdk配置示例**

以设备软件平台配置灵活qinq，匹配cvlan 100+cos 2，添加外层vlan 200，内层vlan修改为300为例：

switchport vlan-mapping both cvlan 100 cos 2 add-outer 200 translate 300

接口使用设备1/5/7口，出接口1/5/8

1. **配置接口查表类型**

因为ssp软件实现匹配cos使用的是tcam查表，而不是hash查找，所以首先设置接口1/5/7的hash查找disable

port 0x19 scl-key-type scl-id 0 direction ingress type disable tcam-type mac

1. **接口下新建一个group**

scl create group 0 type port 0x19

1. **添加匹配规则，以及vlan编辑规则**

scl add group 0 entry 0 mac-entry svlan 100 0xFFF stag-cos 2 0x7 ingress-action vlan-edit stag-op 4 svid-sl 2 new-svid 200 scos-sl 2 new-scos 0 ctag-op 2 cvid-sl 2 new-cvid 300 ccos-sl 2 new-ccos 2

其中各个参数含义：

mac-entry：表示使用mac表项查找

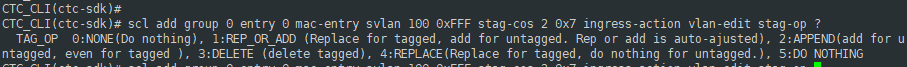
svlan：要匹配的svlan，对应平台命令行的cvlan值100

stag-cos：要匹配的优先级，对应平台命令行的cos 2

ingress-action：表示该规则在接口入方向生效

vlan-edit：表示匹配上报文后要做的操作是修改vlan

stag-op：对外层vlan的操作是4，表示替换原报文的外层vlan



svid-sl：表示要用新的vlan替换



scos-sl：表示用指定的cos替换外层vlan的优先级

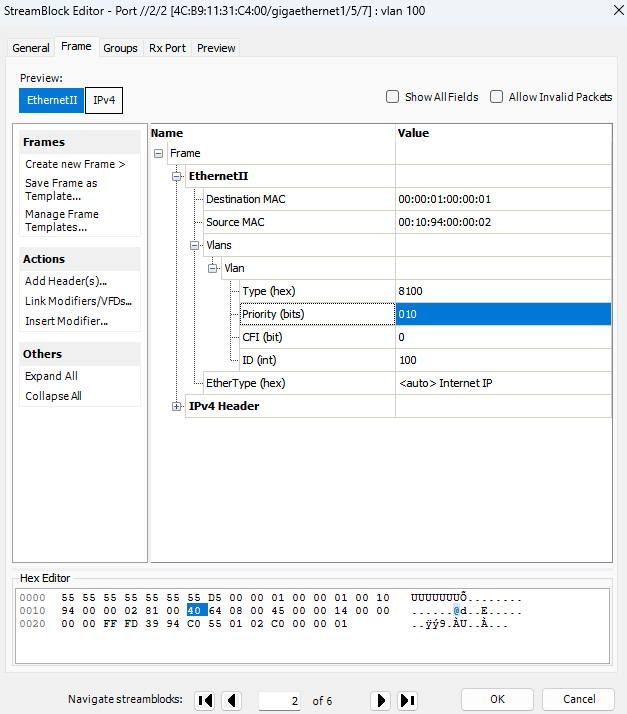


1. **配置完匹配规则以及修改规则后，下发该group：**

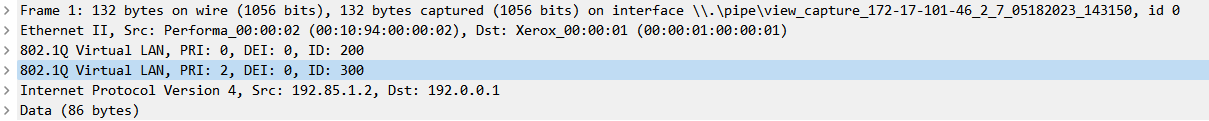
scl install group 0

1. **打流验证**

入口1/5/7接仪表2/2口打流，vlan 100，优先级为2：



在出接口1/5/8接仪表2/7口抓包查看规则是否生效：



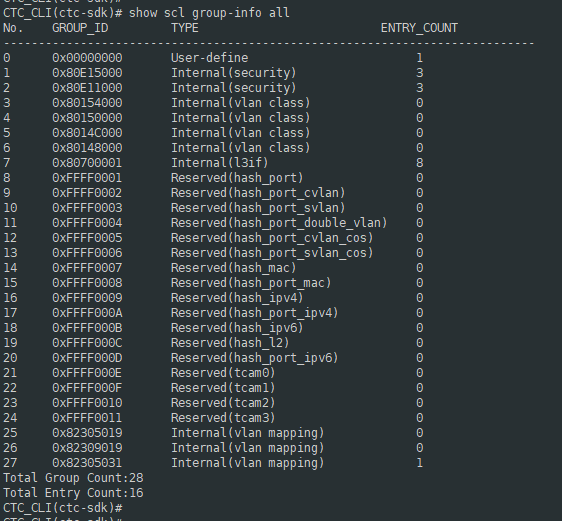
抓包报文源目的mac和IP都与仪表2/2口报文一致，且外层vlan 200，内层vlan 300，优先级保持不变。

1. **查看sdk表项信息**

（1）查看所有group

show scl group-info all

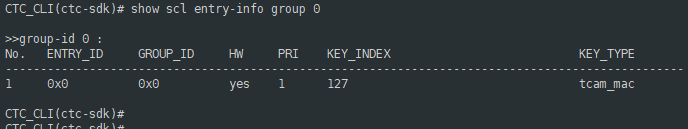
可以看到创建的group 0在第一行：



（2）查看group下的entry表项

show scl entry-info group 0

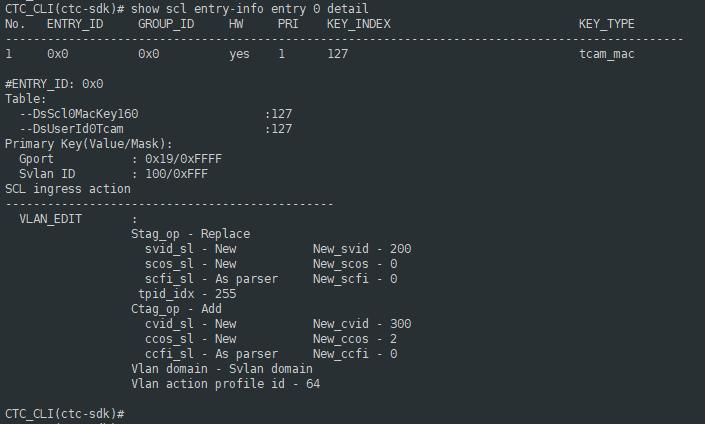
新建的group下只有一个表项，且表项id为0。



（3）查看entry表项的详细信息

show scl entry-info entry 0 detail

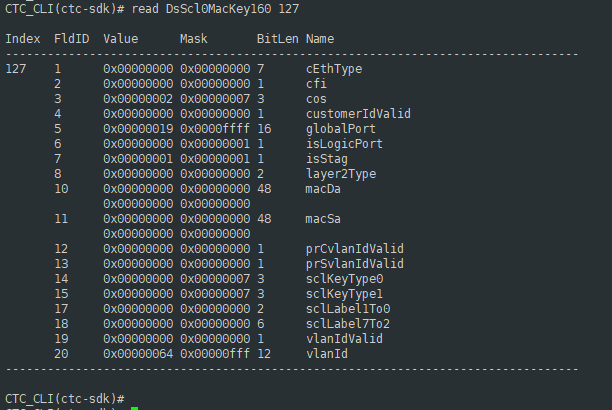
可以看到该表项查表类型是tcam\_mac，ingress方向的vlan编辑规则等信息。



（4）查看寄存器配置

上一步查看到的entry表项详细内容还包括涉及到的寄存器表项，有DsScl0MacKey160和DsUserId0Tcam两个keyindex，同时也给出这两个key的adindex都是127。使用read命令直接读取：

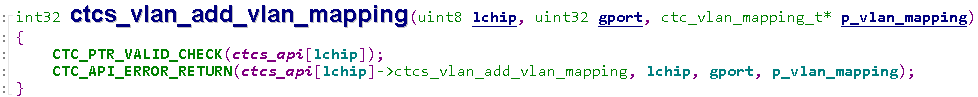
read DsScl0MacKey160 127



对照这些内容可以定位规则不生效时具体的配置问题。

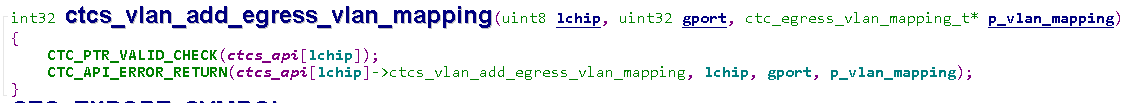
**sdk函数接口**

**ctcs\_vlan\_add\_vlan\_mapping：**



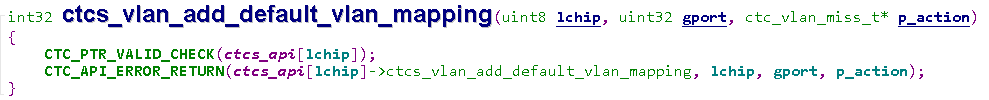
入接口上配置，p\_vlan\_mapping包括匹配的key类型，匹配的规则信息，以及如果操作匹配上的vlan。

**ctcs\_vlan\_add\_egress\_vlan\_mapping：**



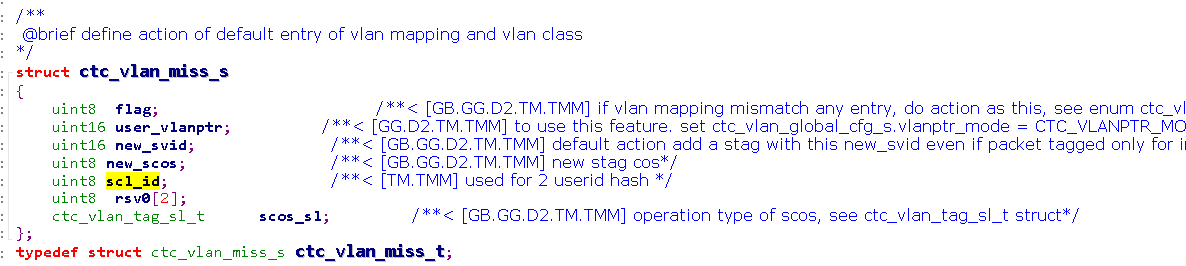
出方向的数据结构比入方向少了很多成员。但是一般用的内容没有差别。

**ctcs\_vlan\_add\_default\_vlan\_mapping：默认的scl hash entry**



每个接口对应一条scl hash default entry，基本qinq功能通过这个函数实现。

传入的参数p\_action：

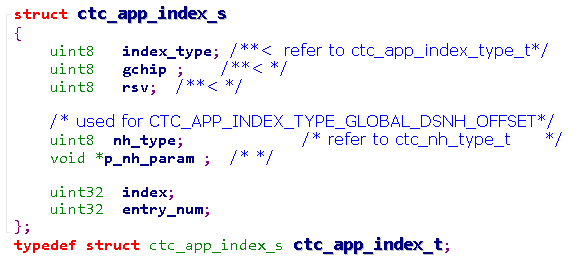


**ctcs\_vlan\_add\_default\_egress\_vlan\_mapping：出方向默认的scl hash entry**

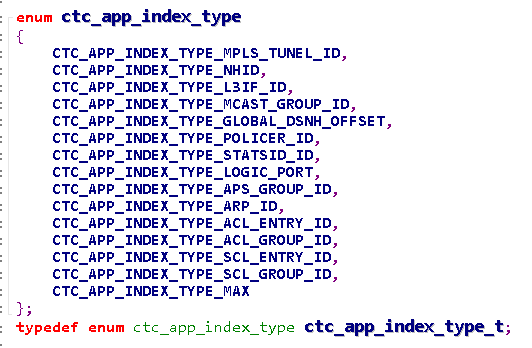
**ctc\_app\_index\_alloc：分配索引，创建sdk表项一般都有用到**

int32 ctc\_app\_index\_alloc(ctc\_app\_index\_t\* app\_index)

创建sdk的表项需要事先分配。这个接口可以分配的表项类型很多，通过参数指定要分配的表项类型即可，参数类型如下：



调用接口时，指定其中index\_type的类型：



比如：

CTC\_APP\_INDEX\_TYPE\_LOGIC\_PORT用来分配逻辑端口；

CTC\_APP\_INDEX\_TYPE\_SCAL\_ENTRY\_ID用来分配scl的entry id;

分配的所以通过ctc\_app\_index\_t的index成员返回。

**ctcs\_l2\_set\_default\_entry\_features**

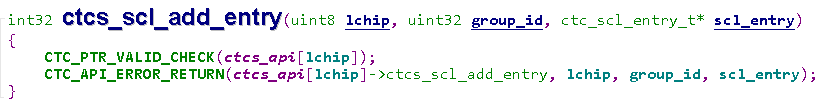
**ctcs\_nh\_add\_xlate**

**ctcs\_l2\_set\_nhid\_by\_logic\_port**

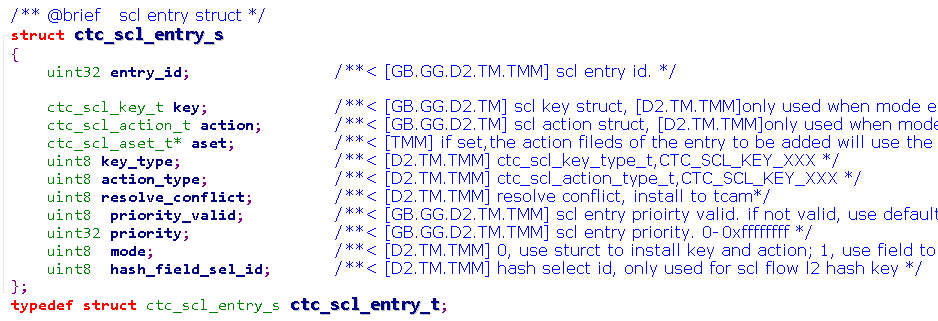
**ctcs\_l2\_add\_port\_to\_defaule\_entry**

N:1 vlan转换用到的sdk接口，主要是反向从接口出去时实现1:N的转换。

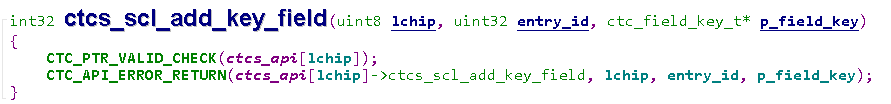
**ctcs\_scl\_add\_entry：创建sdk表项函数**



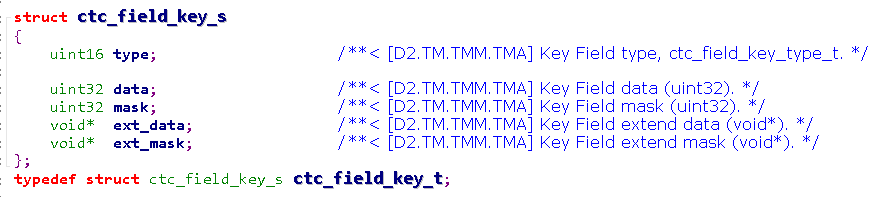
传入的数据结构类型：



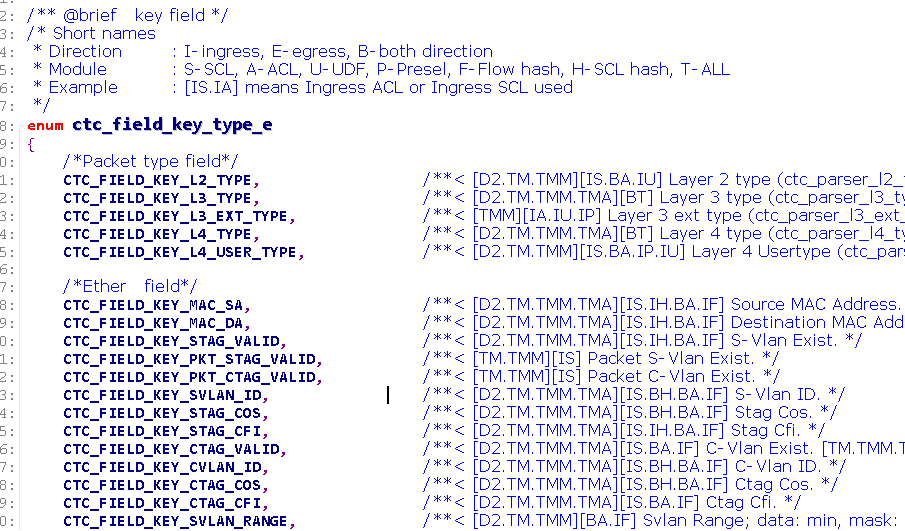
**ctcs\_scl\_add\_key\_field**



配置需要匹配的key类型，匹配的具体内容等。通过数据结构ctc\_field\_key\_t传入。

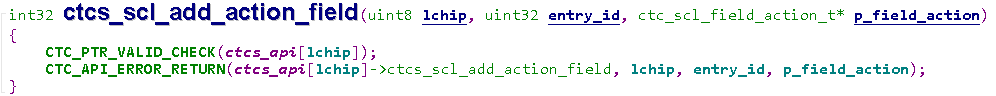


匹配类型通过type指定，一般在ctc\_field\_key\_type\_t内选择：

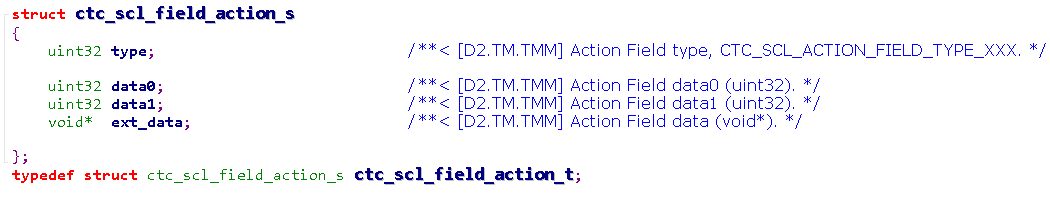


要匹配的具体内容通过data和ext\_data指定，比如：可以直接将mac地址给到ext\_data，再用ext\_mask指定要匹配多少位。

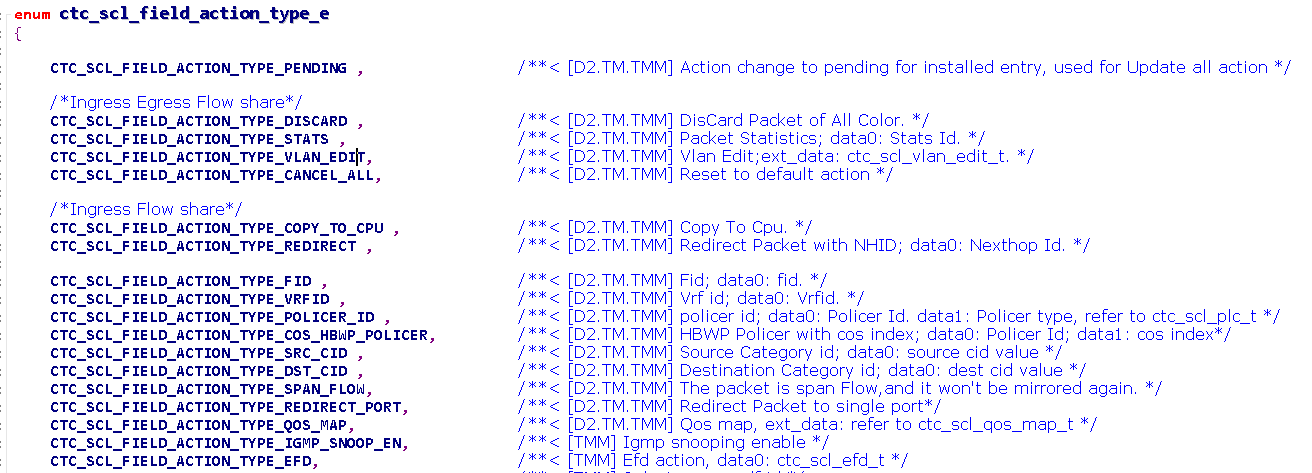
**ctcs\_scl\_add\_action\_field**



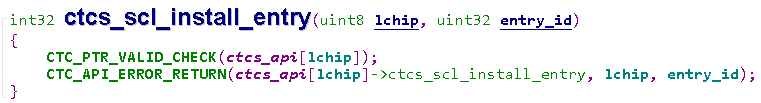
配置scl entry要做的动作，是修改vlan，还是重定向报文等等。数据结构类型：



动作类型通过type在ctc\_scl\_field\_action\_type\_t中指定：



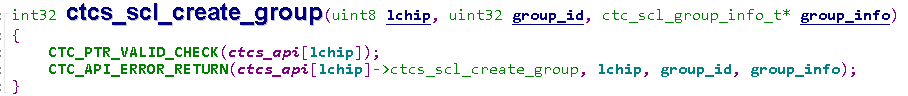
**ctcs\_scl\_install\_entry**



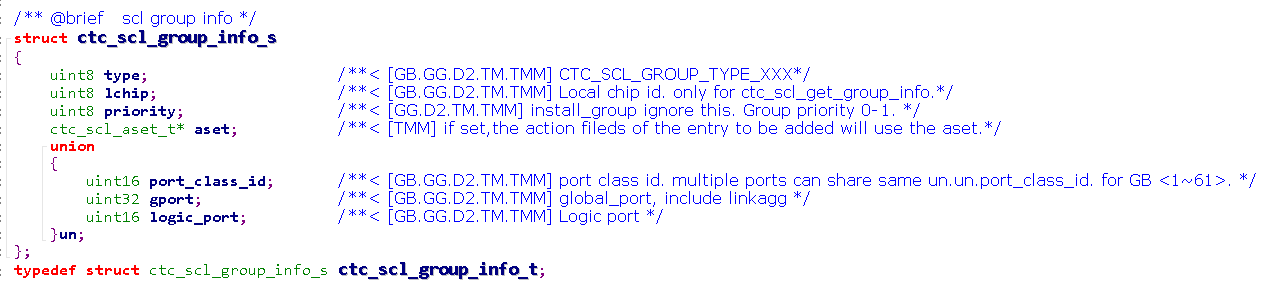
配置好entry的各项属性后，下发scl的entry表项使生效。

scl entry一般是scl group的一个表项，一个group可以有多组entry表项。group在设备初始化时创建。

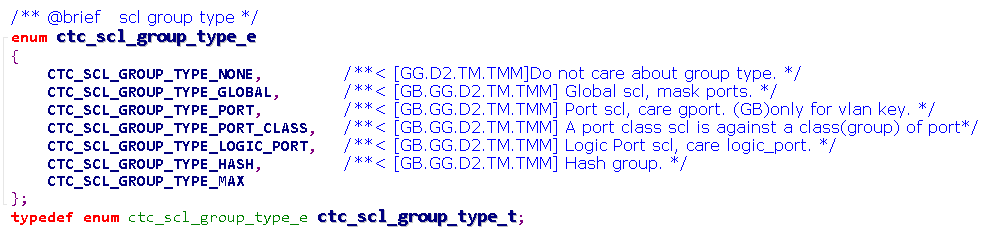
**ctcs\_scl\_create\_group**



参数group\_info的数据结构：

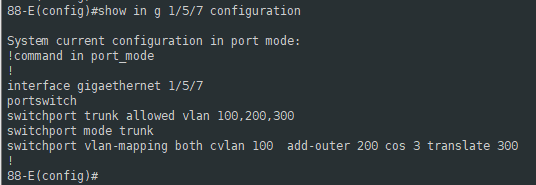


可以选择的group类型：

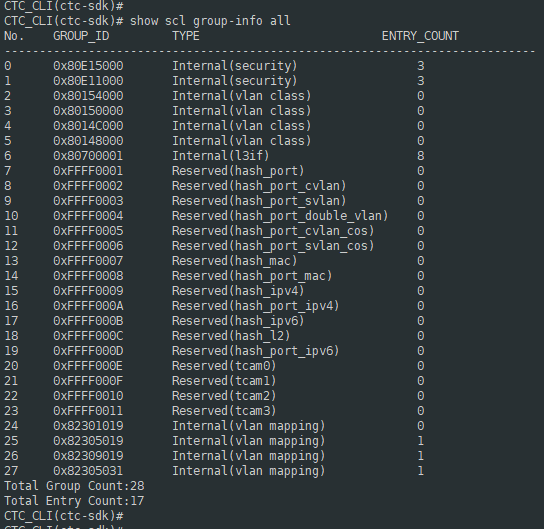


//TODO

**debug常用命令示例：**



配置后查看scl group的entry使用情况，最后一列会显示entry使用统计。

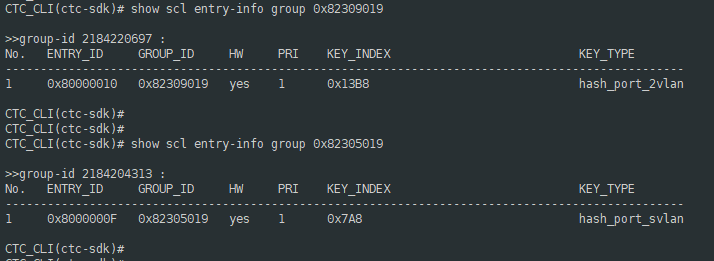


配置灵活qinq：

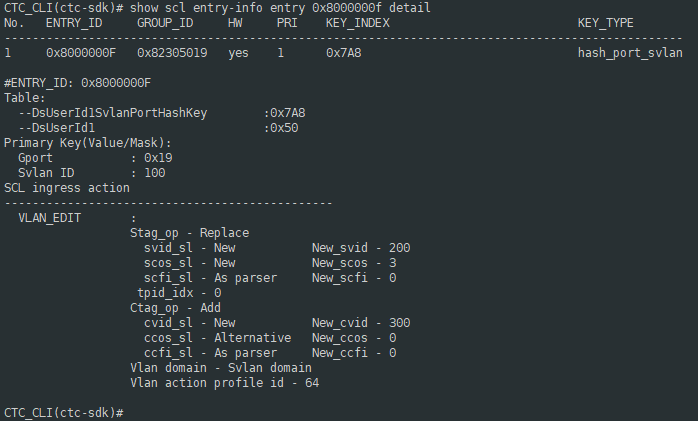
switchport vlan-mapping both cvlan 100 add-outer 200 cos 3 translate 300

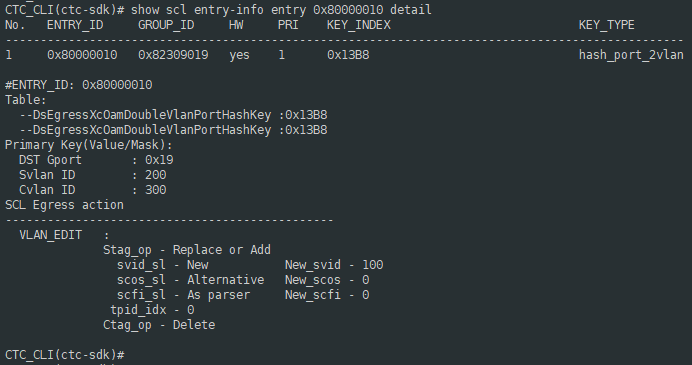
group 0x82305019和0x82309019的ENTRY\_COUNT计数会变成1，也就是说这条命令使用了这两个group中的entry表项。

查看这两个group使用的entry表项id：

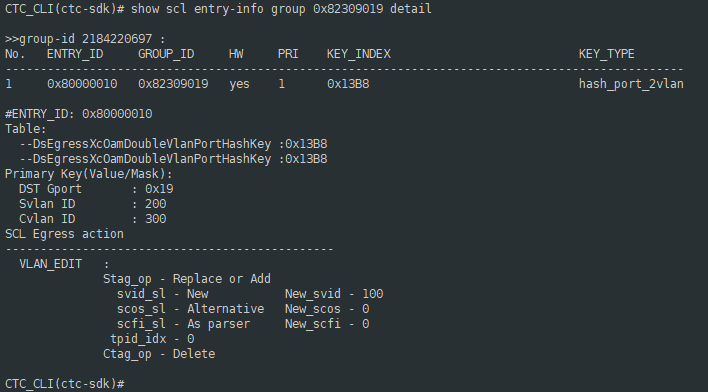


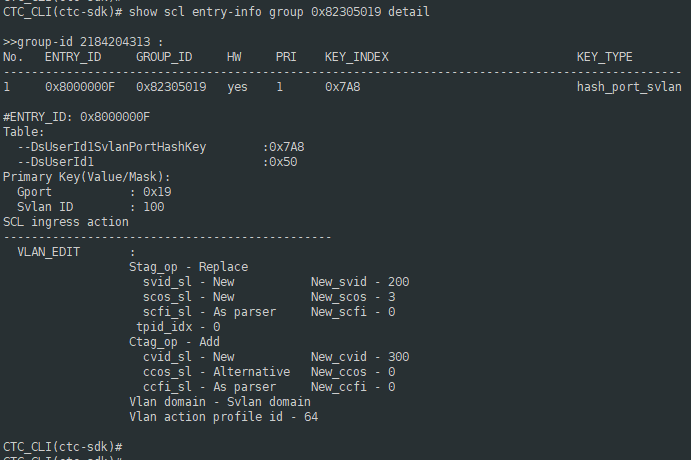
查看entry表项的详细信息：



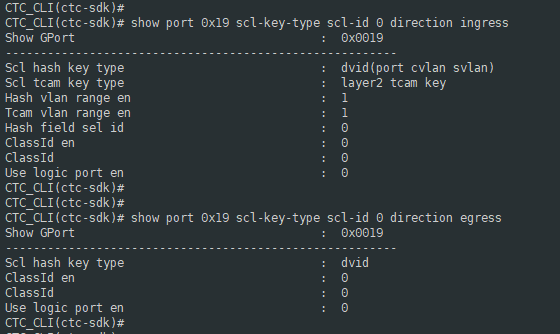


也可以通过group\_id查看entry表项的详细信息：





查看接口的scl属性：



接口0x19是1/5/7口的芯片接口号。

**基本功能测试报告**



**问题**

1. ingress方向vlan\_mapping能实现的，为什么要用scl的tcam查表方案？

现在的qinq命令行中，ingress方向需要匹配vlan+cos和untag做vlan mapping操作时，使用的是scl的tcam查表方案，需要在指定的group中创建新的entry表项，用来匹配vlan+cos或者untag报文。但是，匹配vlan+cos报文，用vlan\_mapping接口（ctcs\_vlan\_add\_vlan\_mapping）是可以实现的，为什么要用scl的tcam方案？

比如设备命令行：

switchport vlan-mapping both cvlan 100 cos 2 add-outer 200 cos 3 translate 300

匹配vlan 100，且cos为2的报文，转换为外层vlan 200，cos值为3，内层vlan 300的报文。

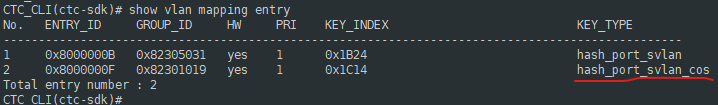
用vlan\_mapping的接口实现，在sdk下配置验证：

vlan mapping add port 0x19 svid 100 stag-cos 2 mapping-to stag-op 4 sl-svid 2 new-svid 200 new-scos 3 ctag-op 2 sl-cvid 2 new-cvid 300 new-ccos 2

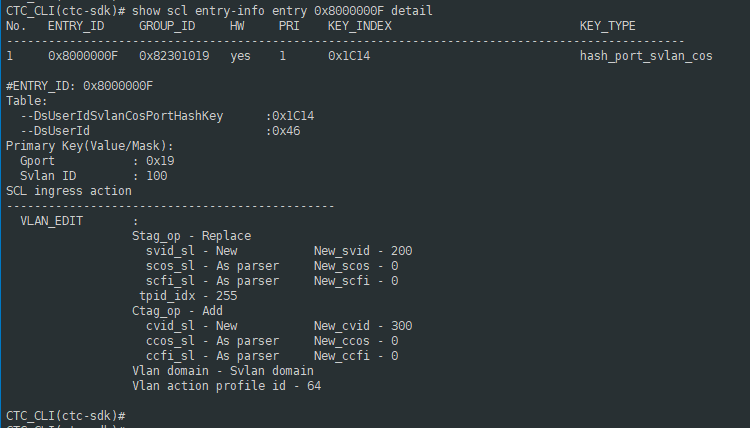
在接口1/5/7下打流验证。

show vlan mapping entry

这个命令可以看到新增一条entry表项，hash查表类型为svlan+cos：



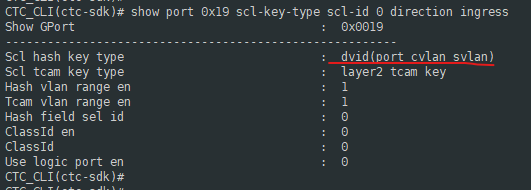
查看该表项的详细内容：



这个时候打流验证，在出口抓包，规则是没有生效的，原因是1/5/7接口下scl id 0的hash类型被初始化为double vlan，用命令：

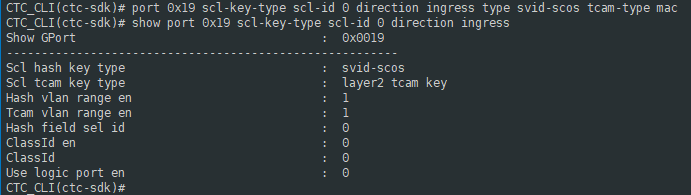
show port 0x19 scl-key-type scl-id 0 direction ingress

可以看到ingress方向的查表类型



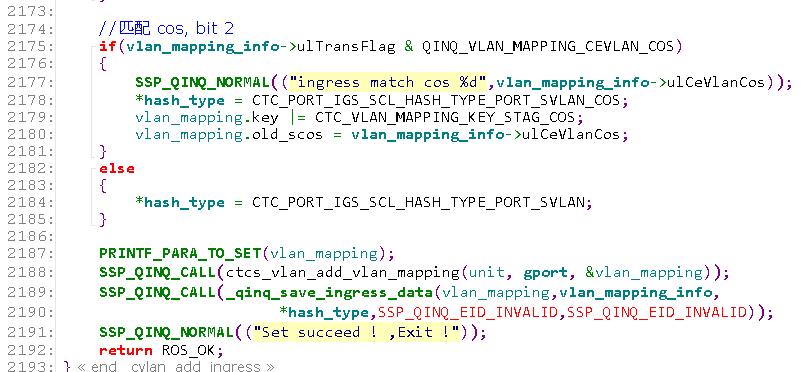
将这个表的查表类型改为svlan+cos：

port 0x19 scl-key-type scl-id 0 direction ingress type svid-scos tcam-type mac

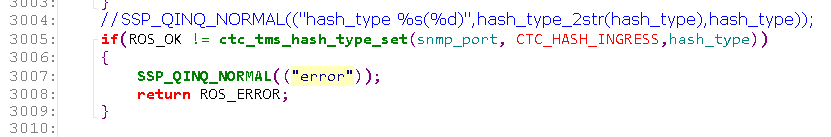


再打流验证是可以看到规则生效了。

代码里面是有考虑到使用ctcs\_vlan\_add\_vlan\_mapping直接匹配vlan+cos的：



同时也考虑到配置时，要相应的修改接口hash类型：



但是，这个修改hash类型的函数没有实现。

原因就是接口的scl资源分配问题了。在ctcScl.c里面对接口的四个tcam分别初始化，0和1初始化为hash查表，类型分别为double vlan和svlan，2和3 disable了hash查表，初始化为tcam查表类型，使用IP资源。



**疑问：**用scl add命令新建一条entry表项，然后也改了scl id 0的hash查表类型，为啥不生效？？？

scl add group 0x82305019 entry 0x3 port-svlan-cos-hash-entry gport 0x19 direction ingress svlan 100 stag-cos 2 ingress-action vlan-edit stag-op 4 svid-sl 2 new-svid 200 scos-sl 2 new-scos 3 ctag-op 2 cvid-sl 2 new-cvid 300 ccos-sl 1

scl install entry 3