# VLAN+QINQ

VLAN（Virtual Local Area Network）虚拟局域网，它的作用是将物理上互联的网络在逻辑上划分为多个互不相关的网络，这些网络之间是无法通信，因此广播就被隔离开了。可以看成，一个VLAN就是一个广播域。

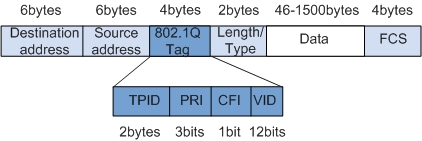
# VLAN的帧格式

传统的以太网数据帧在目的MAC地址和源MAC地址之后封装的是上层协议的类型字段，如[图1](file:///C:\\Users\\Administrator\\Desktop\\VLAN.docx" \l "dc_fd_vlan_0004__fig_dc_cfg_vlan_000201)-1所示。

C:\Users\Administrator\AppData\Local\Temp\ksohtml8712\wps1.jpg

**图1-1** 传统的以太网数据帧格式

IEEE 802.1q是虚拟桥接局域网的正式标准，对Ethernet帧格式进行了修改，在源MAC地址字段和协议类型字段之间加入4字节的802.1q Tag，如[图1-2](file:///C:\\Users\\Administrator\\Desktop\\VLAN.docx" \l "dc_fd_vlan_0004__fig_dc_cfg_vlan_000202)所示。



**图1-2** 802.1q帧格式

802.1Q Tag包含4个字段，各字段解释如[表1](file:///C:\\Users\\Administrator\\Desktop\\VLAN.docx" \l "dc_fd_vlan_0004__table_01)所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **长度** | **名称** | **解释** |
| TPID | 2bytes | Tag Protocol Identifier（标签协议标识符），表示帧类型。 | 取值为0x8100时表示802.1q Tag帧。如果不支持802.1q的设备收到这样的帧，会将其丢弃。 |
| PRI | 3bits | Priority，表示帧的优先级。 | 取值范围为0～7，值越大优先级越高。用于当交换机阻塞时，优先发送优先级高的数据帧。 |
| CFI | 1bit | Canonical Format Indicator（标准格式指示位），表示MAC地址是否是经典格式。 | CFI为0说明是经典格式，CFI为1表示为非经典格式。用于兼容以太网和令牌环网。在以太网中，CFI的值为0。 |
| VID | 12bits | VLAN ID，表示该帧所属的VLAN。 | VLAN ID取值范围是0～4095。由于0和4095为协议保留取值，所以VLAN ID的有效取值范围是1～4094。 |

在一个VLAN交换网络中，以太网帧有以下两种形式：

* 有标记帧（tagged frame）：加入了4字节802.1q Tag的帧
* 无标记帧（untagged frame）：原始的、未加入4字节802.1q Tag的帧

# 基本VLAN

## 端口VLAN模式

端口的VLAN模式比较常见的有access、trunk、hybrid模式，RC产品端口模式形式上看只有access和trunk模式，hybrid模式是通过trunk配置做到的。

### access模式

仅仅允许唯一的VLAN ID通过本接口，这个VLAN ID与接口的缺省VLAN ID相同，Access接口发往对端设备的以太网帧永远是不带标签的帧。主机、一些终端并不需要知道自己属于哪个VLAN，主机硬件通常也不能识别带有VLAN标记的帧，这种设备接入网络一般要通过access口接入网络，给它赋予一个默认VLAN。

**RC产品**端口切换到交换口模式时，默认是access模式，且默认没有默认VLAN，也没有加入VLAN 1。代码里面DEFAULT\_VLAN宏如果是0，就表示交换口默认没有默认VLAN，这个宏会在各个产品的devdef.h文件里面定义。

**相关命令行**

**switchport mode access**

**switchport access vlan *xxx***

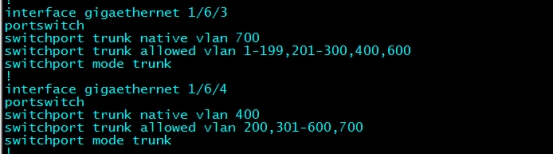
### Trunk模式

Trunk接口是交换机上用来和其他交换机连接的接口，它只能连接干道链路，允许多个VLAN的帧（带Tag标记和不带tag）通过。

RC产品的Trunk模式和一些友商的trunk有些许区别，RC产品的trunk可以说把友商的hybrid口模式也囊括到trunk里面了。等于trunk模式可以设置多个tag vlan和多个untag vlan，每个trunk口都可以设置一个native vlan，也就是端口的默认VLAN，用来标识收到的untag报文应该走哪个VLAN。如果trunk口没有设置native vlan，那么untag报文会被丢弃。设置trunk口的native vlan会将端口加入VLAN。

**注意事项：**

* 端口默认不是交换模式，由路由模式切换到交换模式后，端口默认是access模式；
* Switchport mode没有切换到trunk时，可以配置trunk模式的VLAN，但是实际上并没有将端口加入VLAN，只有当端口加入VLAN模式改成trunk时，才会将相应VLAN加入端口。（这点可能会对用户造成一定的迷惑）
* hybrid模式是通过trunk模式变换而来，没有直接的hybrid模式，可以通过trunk模式设置VLAN加入端口是tag还是untag来设置。
* 设置trunk口的native vlan就会将端口已untag形式加入VLAN，并且只要配置了native vlan，不能通过remove命令将native vlan从端口退出（这点可能是RC有意而为之）。以下这个配置是通过remove命令将VLAN 700移出1/6/3，但是实际上并没有下发硬件，1/6/3仍然能转发vlan 700的流。



**相关命令行**

**switchport trunk allowed vlan *vlanlist* confirm 端口加入VLAN**

**switchport trunk allowed vlan add *vlanlist* 追加VLAN加入端口**

**switchport trunk untagged vlan *vlanlist* confirm 端口以untag形式加入VLAN**

**Switchport trunk untagged vlan remove *vlan-list* 端口退出untag变成tag模式**

**Switchport trunk allowed vlan remove *vlan-list* 端口退出VLAN**

**switchport trunk native vlan tagged | untagged 设置trunk口的native vlan是tag还是untag形式**

### VLAN模式帧处理

Access、trunk、hybrid模式都是软件概念，端口加入VLAN的模式硬件层面来说，就是端口加入/退出VLAN，加入端口是tag还是untag形式加入。Untag和tag形式主要是说对应VLAN的报文出端口的行为，untag形式加入端口，那么对应VLAN的报文从端口转发出去的时候，就需要剥掉tag；如果tag形式加入端口，那么对应VLAN的报文从端口转发出去的时候，如果没有tag就需要加一层tag。

**表1** 各类型接口对数据帧的处理方式

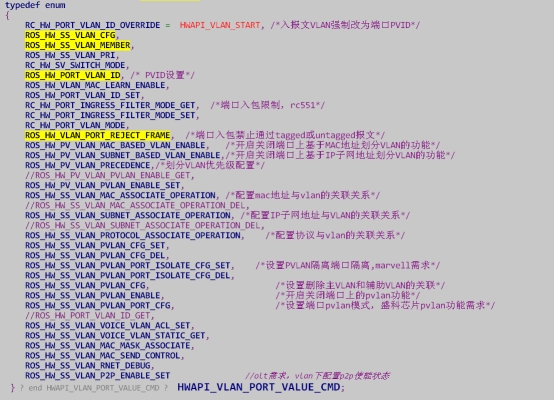
|  |  |  |  |
| --- | --- | --- | --- |
| **接口类型** | **对接收不带Tag的报文处理** | **对接收带Tag的报文处理** | **发送帧处理过程** |
| Access模式 | 接收该报文，并标记缺省VLAN的Tag。 | * 当VLAN ID与缺省VLAN ID相同时，接收该报文。 * 当VLAN ID与缺省VLAN ID不同时，丢弃该报文。 | 先剥离帧的PVID Tag，然后再发送。也就是access口发送出去的肯定是untag帧。 |
| Trunk模式 | * 标记缺省的VLAN ID，当缺省VLAN ID在允许通过的VLAN ID列表里时，接收该报文。 * 打上缺省的VLAN ID，当缺省VLAN ID不在允许通过的VLAN ID列表里时，丢弃该报文。（如果配置了native vlan,trunk口的native vlan无论如何不会被退出端口，所以也就不会出现丢弃untag报文的情况） | * 当VLAN ID在接口允许通过的VLAN ID列表里时，接收该报文。 * 当VLAN ID不在接口允许通过的VLAN ID列表里时，丢弃该报文。 | * 当VLAN ID与缺省VLAN ID相同，且缺省VLAN在是untag模式，去掉Tag，发送该报文。 如果缺省VLAN配置为tag模式，则保留tag直接发送出去。 * 当VLAN ID与缺省VLAN ID不同，且是该接口允许通过的tag vlan列表时，保持原有Tag，发送该报文。 * 当VLAN ID是该接口允许通过的untag vlan列表里面，去掉tag，发送该报文。 |

### 动态VLAN

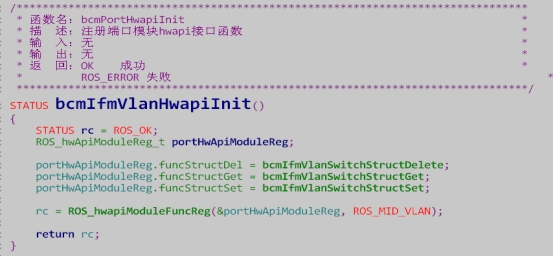
静态VLAN和动态VLAN仅仅是一种软件概念，对于硬件层面只有静态VLAN的概念。

## SSP VLAN软件实现

VLAN相关的产品驱动命令字如下，其中**基本VLAN相关为高亮部分**：

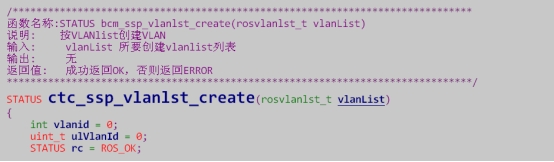


Ssp层注册的处理API接口：

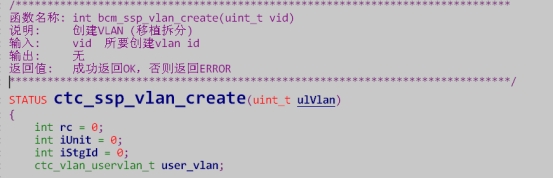
SSP层面处理基本VLAN和端口的VLAN属性总接口就是上面注册的get、set、del回调函数，协议软件通过对应的命令字调用到ssp层的接口进行处理。

### 创建/删除VLAN

**创建VLAN涉及ssp层接口主要有：**

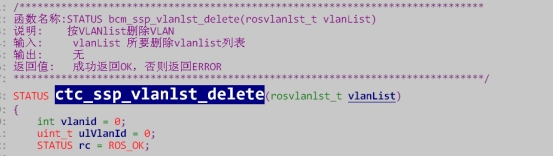


**逻辑说明**：上层传下来的VLAN list表以bitlist的形式进行解析，一个一个创建，如果其中一个创建失败，则需要回退本次所有创建的VLAN。

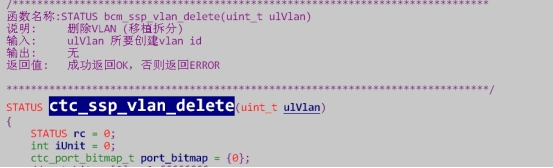


**逻辑说明：**调用SDK的API创建VLAN，创建成功后，从软件层面获取VLAN所在的stgId，并设置到硬件将VLAN加入stgId；打开VLAN下的ACL功能（这个应该是CTC芯片特有的）

**删除VLAN所涉及的ssp接口:**



**逻辑说明**：上层传下来的VLAN list表以bitlist的形式进行解析，一个一个删除，如果其中一个删除失败，则需要回退本次所有删除的VLAN。



**逻辑说明：**删除VLAN时，并不是真正从硬件层面删除VLAN，而是将VLAN退出所有端口（因为子接口使能MPLS是在VLAN表中使能，所以初始化创建所有VLAN），并发送删除VLAN的清MAC的消息给ssp mac基于VLAN清MAC。

**相关命令：**

**create vlan** *vlanlist* **active**

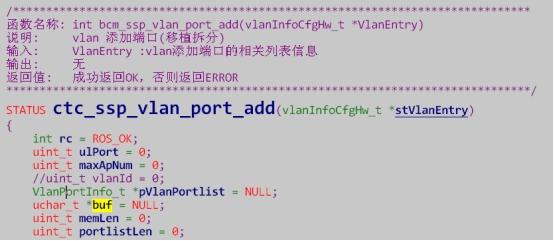
**no vlan** *vlan-list*

#### CTC SDK API

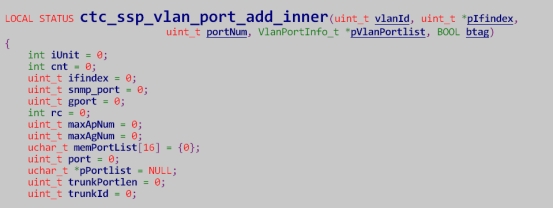


### 端口加入/退出VLAN

**涉及ssp层接口主要有：**



**逻辑说明**：从上层传下来的结构体中解析出加入VLAN的tag port list和untag port list，并将port list分别按照tag和untag模式加入VLAN，硬件设备完成后，软件记录端口和VLAN的关系。



**逻辑说明**：从上层传下来的结构体中解析出加入VLAN的tag port list和untag port list，并将port list分别按照tag和untag模式加入VLAN，硬件设备完成后，软件记录端口和VLAN的关系。聚合口加入VLAN是解析到每个成员口加入的，软件记录是成员口和ag聚合口都记录。

#### CTC SDK API

**Int32 ctcs\_vlan\_set\_tagged\_port(uint8 lchip, uint16 vlan\_id, uint32\_gport, bool tagged)**

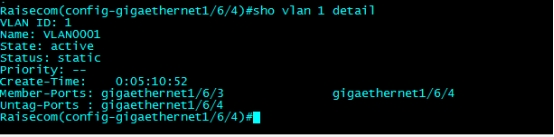
**Int32 ctcs\_vlan\_remove\_port(uint8 lchip, uint16 vlan\_id, uint32\_gport, bool tagged)**

**Int32 ctcs\_vlan\_add\_port(uint8 lchip, uint16 vlan\_id, uint32\_gport, bool tagged)**

## 查看VLAN和端口命令

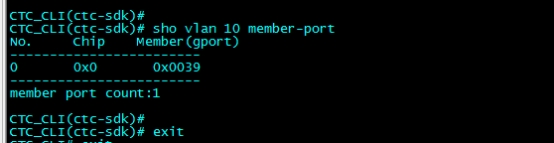
### 软件层面

**show vlan xxx detail**

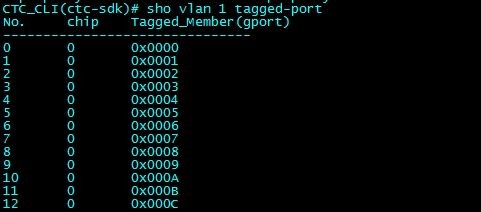


### CTC SDK层面

**show vlan xxx member-port**



**Show vlan xxx tagged-port**

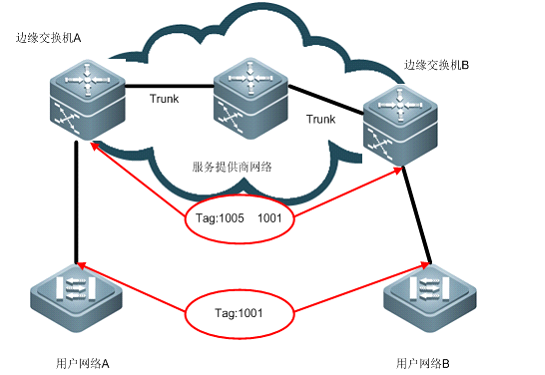


# 基本QINQ

## QINQ的产生

QinQ（802.1Q-in-802.1Q）技术是一项扩展VLAN空间的技术，通过在802.1Q标签报文的基础上再增加一层802.1Q的Tag来达到扩展VLAN空间的功能，可以使私网VLAN透传公网。由于在骨干网中传递的报文有两层802.1Q Tag头（一层公网Tag，一层私网Tag），即802.1Q-in-802.1Q，所以称之为QinQ协议。QinQ最初主要是为拓展VLAN的数量空间而产生的。它是通过在原有的802.1Q报文的基础上增加一层802.1Q标签来实现的，使得VLAN数量增加到4094×4094。另外，可以通过不同的标签，代表不同的业务，对不同的用户和不同的业务提供不同的Qos服务部署。

## 应用场景

qinq应用场景图

## 基本QINQ原理

基本QinQ是基于接口方式实现的，所以也叫基于端口的QINQ。开启接口的基本QinQ功能后，当该接口接收到报文，设备会为该报文打上本接口缺省VLAN的VLAN Tag。如果接收到的是已经带有VLAN Tag的报文，该报文就成为双Tag的报文；如果接收到的是不带VLAN Tag的报文，该报文就成为带有接口缺省VLAN Tag的报文。

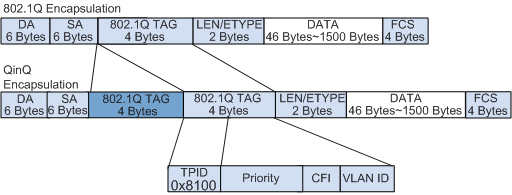


图3-1 Qinq封装

## CTC芯片实现原理

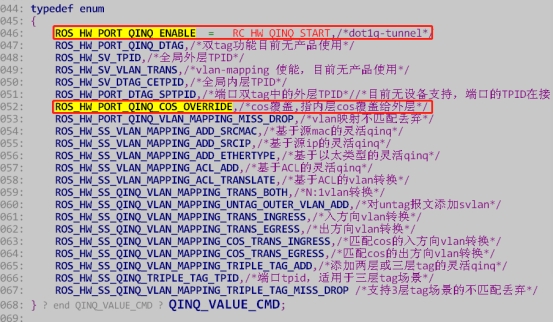
CTC 7132芯片RC软件实现并不是用的CTC推荐的方案，CTC SDK用户开发指南推荐的方案，会丢弃双层TAG报文，没办法继承报文的C-cos，S-cos只能是0 ，把端口的vlan-ctl改成allow-all，也无法对双层TAG的报文额外再加一层tag。

CTC 7132芯片**SSP有关基本QINQ的实现**，利用的是SCL资源SCL hash default entry，设置端口的vlan-mapp 没有匹配到时候的default行为，当scl hash和tcam都没有找到的时候，会使用scl hash default entry，每个port对应一条scl hash default entry，可以配置每个端口的default entry action行为，基本QINQ利用这个action来实现。采用这种方案的缺点：必须依赖端口的默认VLAN，且需要明确下发到default entry的action里面，所以，如果端口的默认VLAN变化了，需要及时处理更新action里面的VLAN。

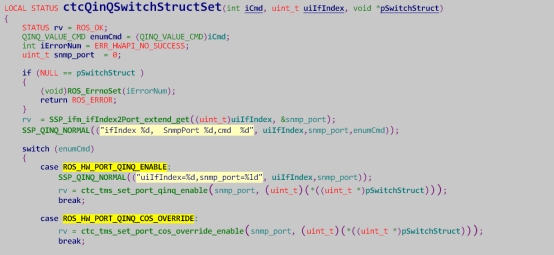


## SSP软件实现

**驱动接口及命令字，和基本QINQ有关的命令字只有黄色高亮部分，其他基本都是灵活QINQ相关。**



Ssp层基本QINQ实现相关接口：



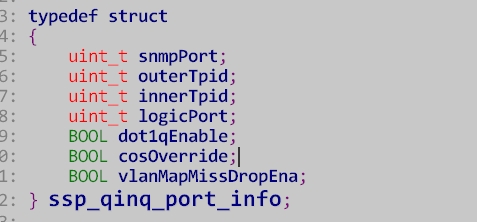
**ssp软件实现流程：**



软件实现上，分了两种分支，一种是带cos override和不带两种，不设置cos override的基本QINQ添加的S-VLAN tag不继承C-VLAN的优先级，优先级S-COS为0；如果设置了cos override的基本QINQ添加S-VLAN会继承C-VLAN的优先级，即S-cos的和C-cos优先级一样。

Agg聚合口的基本QINQ是下发到每个成员口上的，所以，如果有成员口的加入退出，QINQ都必须关注相关事件，并做相应的处理。

**相关软件结构体**：

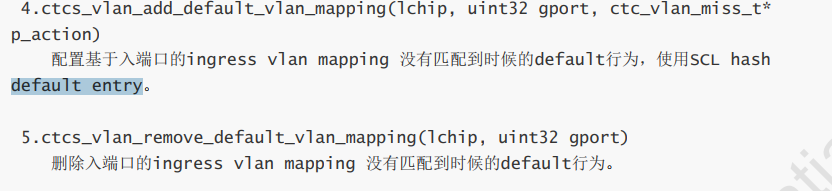




全局端口数组**pQinqPortInfo**，用来记录所有端口基本QINQ相关信息。

## CTC SDK相关命令及API

**CTC SDK相关 API:**



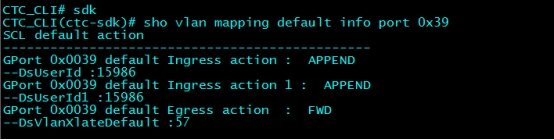
配置端口的基本QINQ

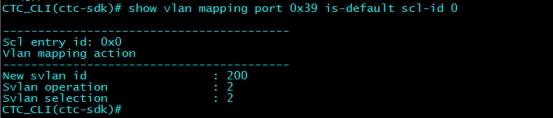
**vlan mapping add port 0x39 mismatch append-stag new-svid 600 sl-scos 0 scl-id 0**

查看端口的基本QINQ信息

show vlan mapping port 0x39 is-default scl-id 0

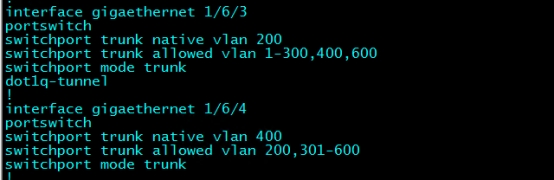
show vlan mapping default info port 0x39



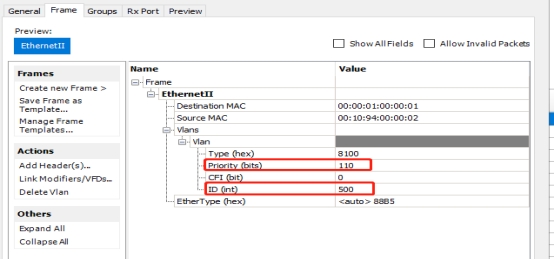


## 测试结果

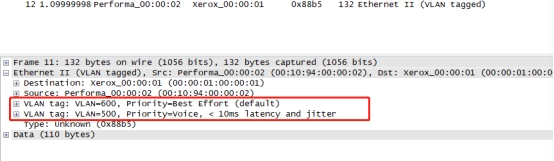
1、1/6/3和1/6/4口配置如下（不带cos配置），从1/6/3口打流，从1/6/4口抓包观察：



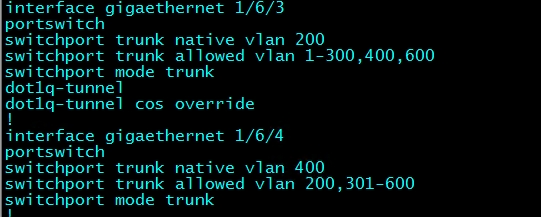
从1/6/3口打流信息，**VLAN500，优先级pri 6的报文**：



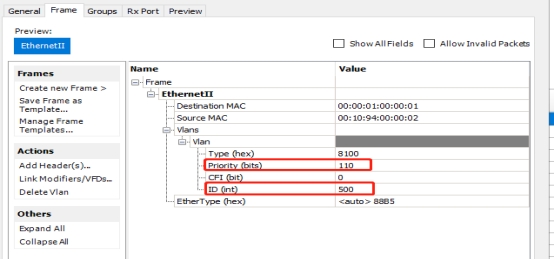
从1/6/4口抓包信息信息(**加了外层tag，外层tag的cos默认为0，没有继承报文原始tag的优先级**)：



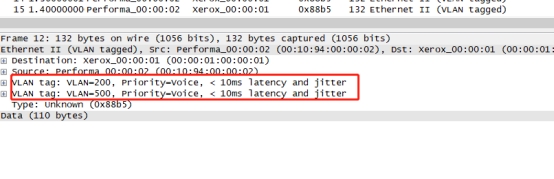
2、1/6/3和1/6/4口配置如下（带COS配置），从1/6/3口打流，从1/6/4口抓包观察：



从1/6/3口打流信息，**VLAN500，优先级pri 6的报文**：



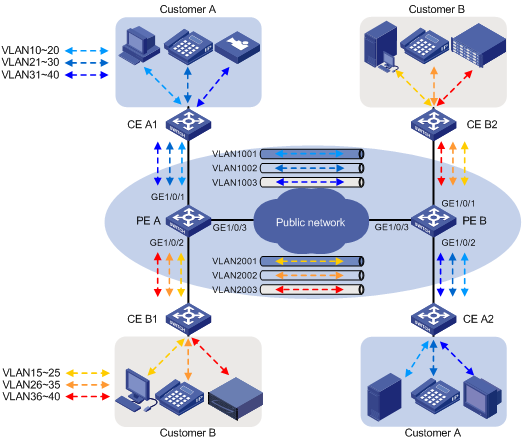
从1/6/4口抓包信息信息(**加了外层tag，外层tag的cos为6，继承报文原始tag的优先级**)：



# 灵活QINQ

上一章节讲到了基本QinQ是基于端口的，它的缺点在于不够灵活，满足不了不同业务、不同用户的灵活透传和优先级服务。 灵活QinQ是对QinQ的一种更灵活的实现，主要有vlan-mapping、vlan-stack、基于流的QINQ修改或添加VLAN tag等。

## 应用场景



## 灵活QinQ原理



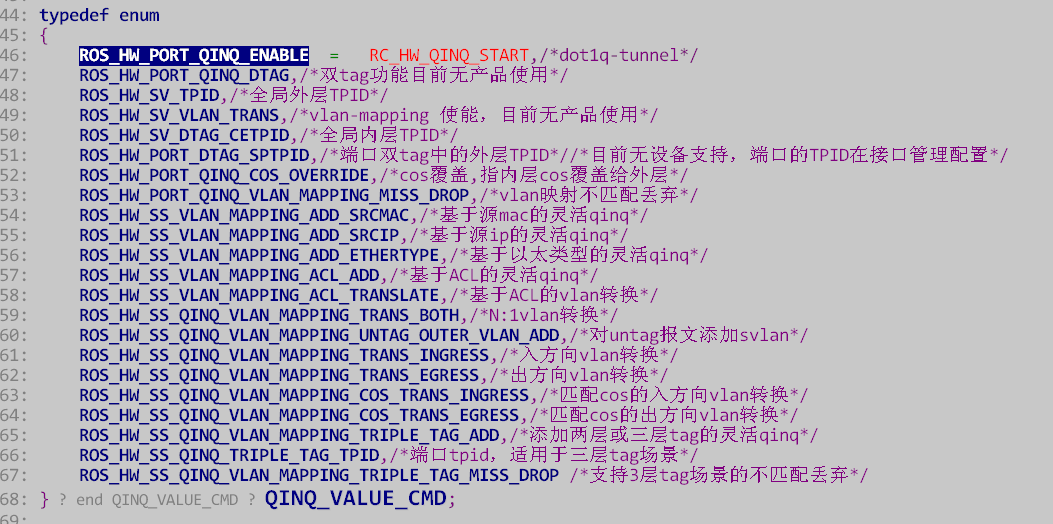
## CTC芯片实现原理



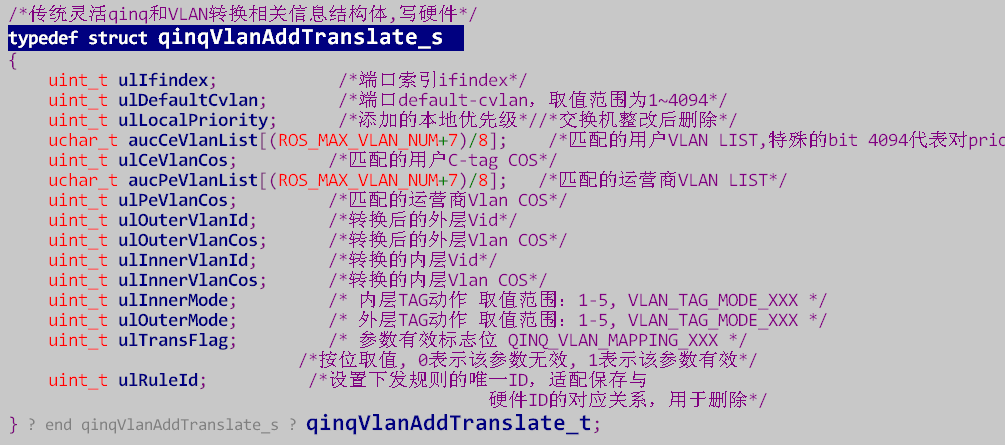
从代码实现来看，scl hash可以基于SVLAN，但不能基于VLAN+cos，TCAM可以基于VLAN+cos来查找，一般条件里面携带cos条件的，都通过SCL TCAM来实现。这个有待和FAE确认，从芯片资料里面对SCL资源介绍没有很好。

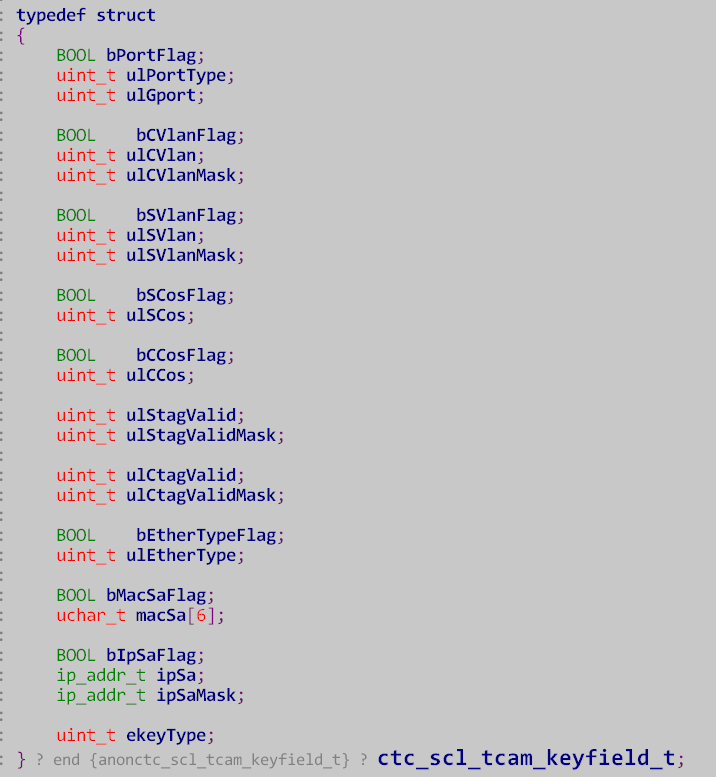
## SSP软件实现

驱动接口及命令字，除基本QINQ外的其他基本都是灵活QINQ相关。

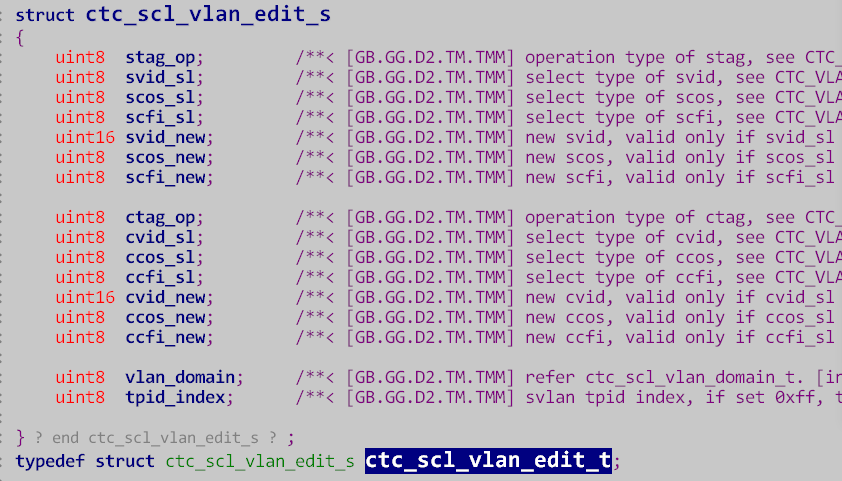


**协议设置硬件到ssp层的结构体：**

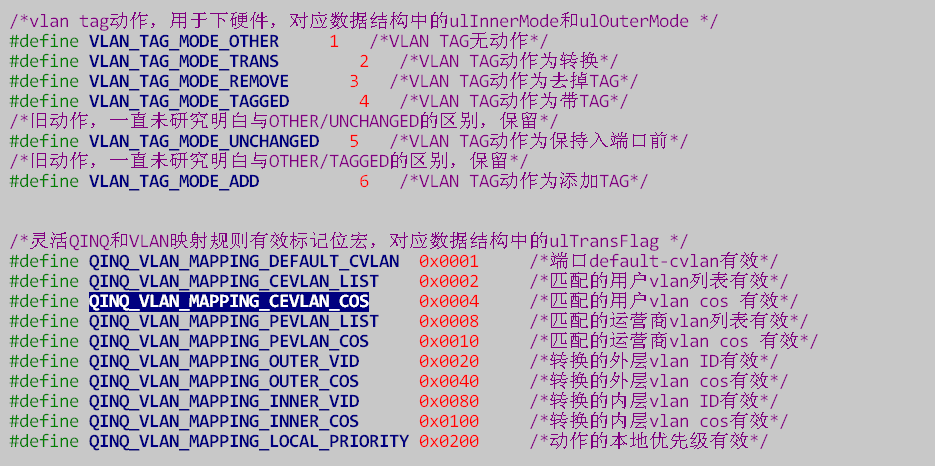




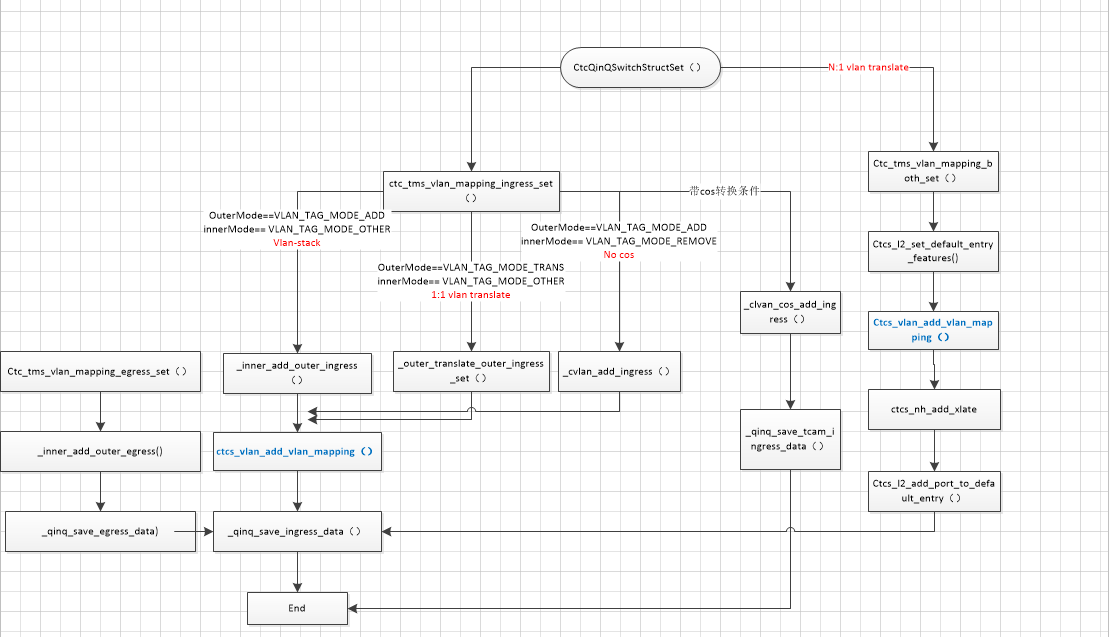
**SDK结构体：**



**灵活QINQ动作相关宏：**



**Ssp层软件实现接口调用关系：**



**资源使用情况：**

基本QINQ和灵活QINQ基本都用的芯片的scl hash资源，基本QINQ和2:2的vlan mapping使用的是scl hash 0资源，其他灵活QINQ基本使用的是scl hash 1资源。其它功能模块（MPLS、IFM、QOS等）在用到scl hash资源时，整体上也基本遵循这个原则使用。在vlan mapping中基于VLAN+COS的mapping使用的scl tcam资源。

和VLAN划分以及QINQ会用的scl tcam资源规划为4个已知的scl tcam group和一个自动创建的scl tcam group资源（不关心group类型，信息由entry来定，这也是CTC推荐的用法），其中带cos的灵活QINQ、MAC-VLAN、IPsub-VLAN都是使用的已经创建好的scl tcam group资源，ip-single-vlan使用自定义的scl tcam group资源。

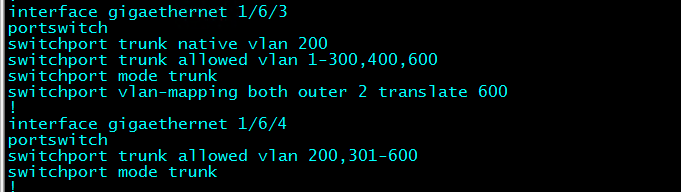
初始化使能使能端口入方向和出方向基于单tag和双tag的scl hash查找，入方向基于单tag和双tag的scl tcam 查找。初始化创建自定义了



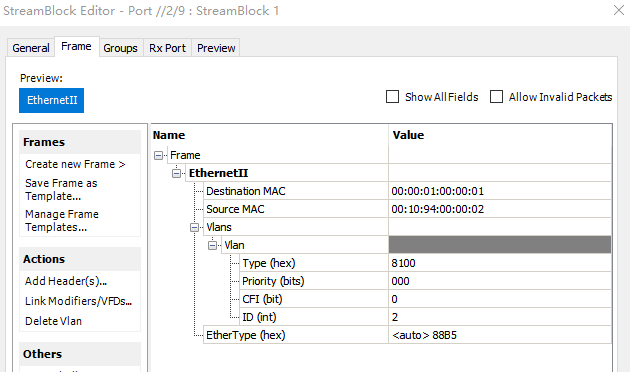
**SCL TCAM entry下发流程**

## 测试结果

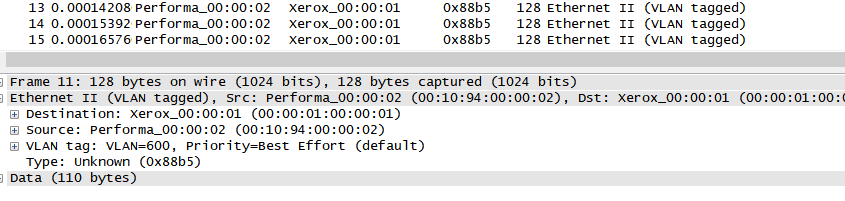
### 1:1 vlan-mapping



**翻译**：在1/6/3口上发送vlan 2的报文，在1/6/4口上收到vlan 600的报文（没有继承优先级，修改代码测试可以继承优先级）

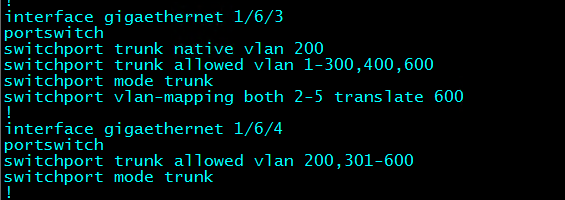


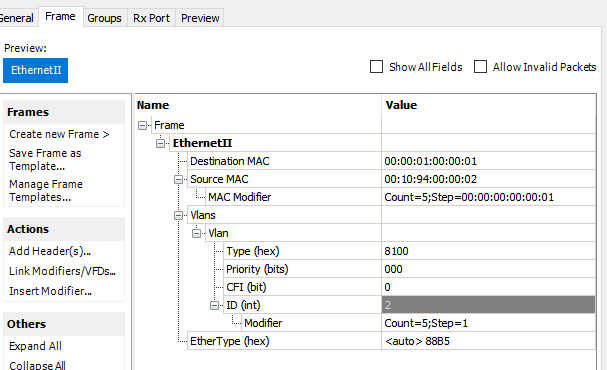
**回翻**：在1/6/4口上发送vlan 600的报文，在1/6/3口上收到vlan 2的报文



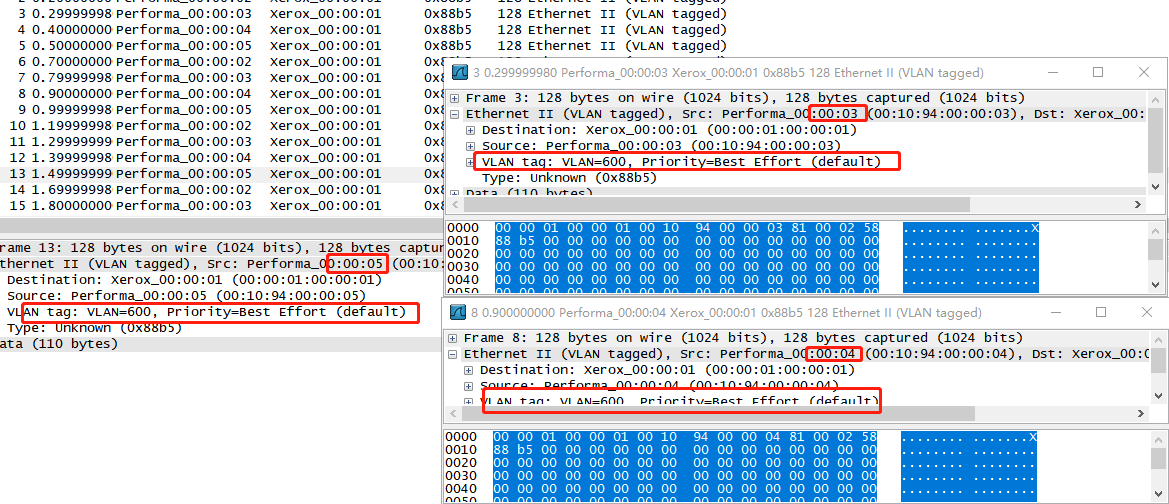
### N:1 vlan-mapping

**命令行：switchport vlan-mapping both *vlanlist* translate *vlan***



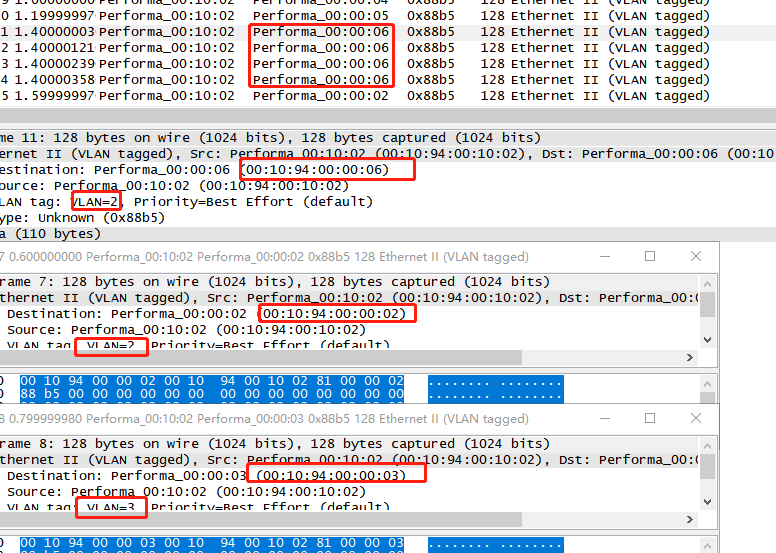


**翻译**：1/6/3上打vlan 2-5的流，在1/6/4上收到vlan 600的流（**没有优先级继承**）

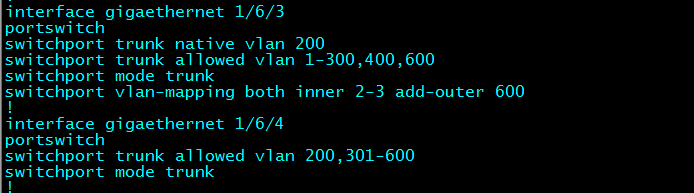


**回翻验证：**

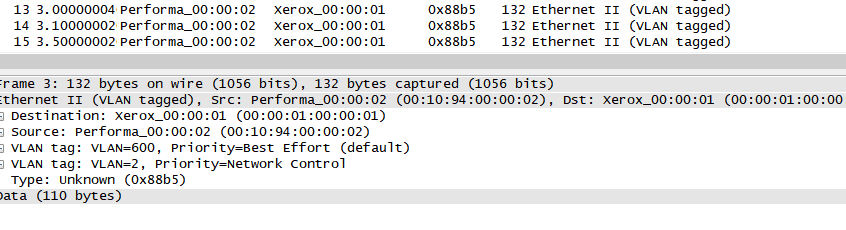
验证**结果不符合预期**，不应该回翻的，做了**错误回翻**，从结果来看，表现为未知流会在每个VLAN内做一次回翻。比如配置里面从1/6/4口回翻的VLAN 600流，**如果找不到MAC，就会每条流都回翻一个N**。已知流的回翻没有问题，未知流一般在原VLAN内洪范应该会更好。



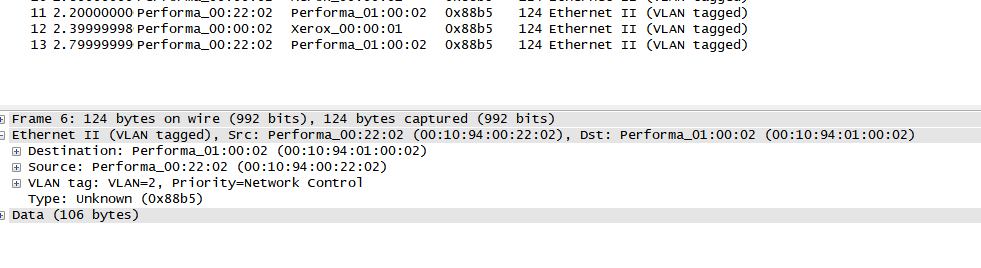
### CVLAN-🡪SVLAN+CVLAN



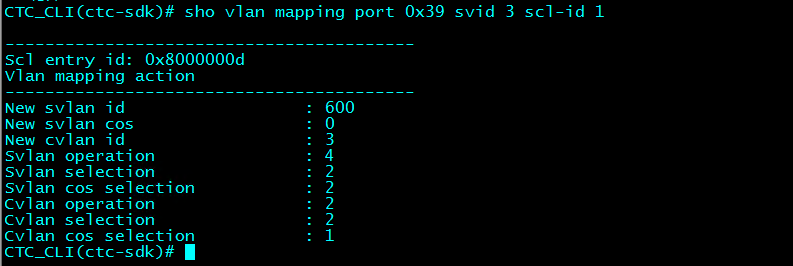
**翻译**：1/6/3口vlan 是2-5的流，会在1/6/4口上收到外tag 600内tag 2-3的流（没有优先级继承）



**回翻**：在1/6/4口上打外tag 600，内tag 2-5的流，在1/6/3上收到vlan 2-3的流和外tag 600内tag 4-5流。



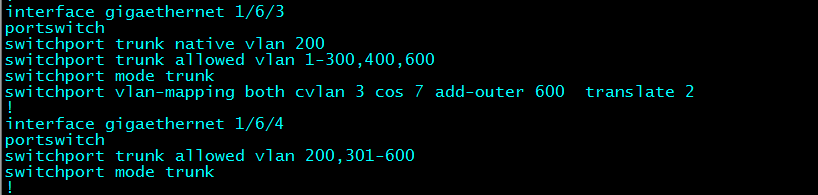
**底层命令查看规则：**



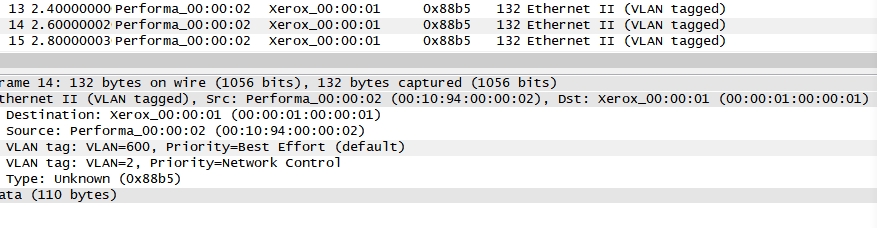
### 带cos的vlan maping

**配置：**

1/6/3口配置cvlan 3 cos 7的报文添加vlan 600 并将内tag 翻译成2（**不继承优先级**）



**翻译：**1/6/3口打vlan 3 cos 7的报文，会在/1/6/4口上收到外tag 600，内tag 2 cos 7的报文



**回翻**：从1/6/4口发送目的MAC已知和未知，SVLAN 600, cvlan 2的报文，在1/6/3口都会收到VLAN 3 cos 7的报文。从测试来看，回翻不关心MAC，不关心cos。

