



# **RTL8306E/8306M/8304E**

## **PROGRAMMING GUIDE**

**V1.0.2**  
**Feb 2, 2012**



Realtek Semiconductor Corp.

No. 2, Innovation Road II, Hsinchu Science Park, Hsinchu 300, Taiwan

Tel.: +886-3-578-0211. Fax: +886-3-577-6047

[www.realtek.com](http://www.realtek.com)

## **COPYRIGHT**

©2012 Realtek Semiconductor Corp. All rights reserved. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form or by any means without the written permission of Realtek Semiconductor Corp.

## **TRADEMARKS**

Realtek is a trademark of Realtek Semiconductor Corporation. Other names mentioned in this document are trademarks/registered trademarks of their respective owners.

## **DISCLAIMER**

Realtek provides this document “as is”, without warranty of any kind, neither expressed nor implied, including, but not limited to, the particular purpose. Realtek may make improvements and/or changes in this document or in the product described in this document at any time. This document could include technical inaccuracies or typographical errors.

## **USING THIS DOCUMENT**

This document is intended to be used by the software engineer when programming for Realtek RTL8306E controller chips.

Though every effort has been made to assure that this document is current and accurate, more information may have become available subsequent to the production of this guide. In that event, please contact your Realtek representative for additional information that may help in the development process.

## **CONFIDENTIALITY**

This document is confidential and should not be provided to a third-party without the permission of Realtek Semiconductor Corporation.

## **REVISION HISTORY**

| <b>Revision</b> | <b>Release Date</b> | <b>Summary</b>                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.0.0           | 2010/04/16          | First Release                                                                                                                                                                                                                                                                                                                                                                             |
| 1.0.1           | 2010/06/28          | Add IGMP Snooping chapter                                                                                                                                                                                                                                                                                                                                                                 |
| 1.0.2           | 2012/02/02          | (1) Add descriptions of differences about RTL8306E/8306M/8304E<br>(2) Add sector 3.3 to describe code modification in SDK driver code for RTL8306E/8306M/8304E<br>(3) Add sector 7 to describe port number and port mapping for RTL8306E/8306M/8304E<br>(4) update API:<br>rtk_port_macForceLinkExt0_set<br>rtk_port_macForceLinkExt0_get<br>(5) Add API:<br>rtk_port_macForceLinkExt_set |

|  |  |                                                                                                                                                                                       |
|--|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  |  | <code>rtk_port_macForceLinkExt_get</code><br><code>rtk_mib_get;</code><br><code>rtk_mib_cntType_set/get;</code><br><code>rtk_mib_reset;</code><br><code>rtk_mii1Disable_set();</code> |
|--|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

## Table of Contents

---

|                                                             |           |
|-------------------------------------------------------------|-----------|
| <b>1 OVERVIEW .....</b>                                     | <b>6</b>  |
| <b>2 DIRECTORY STRUCTURE.....</b>                           | <b>7</b>  |
| <b>3 ASIC DRIVER.....</b>                                   | <b>8</b>  |
| 3.1 PORTING ISSUE .....                                     | 8         |
| 3.2 INITIALIZATION .....                                    | 9         |
| 3.3 CHIP MODULE.....                                        | 9         |
| <b>4 RTK API FOR RTL8306E .....</b>                         | <b>12</b> |
| 4.1 VLAN .....                                              | 12        |
| 4.2 VLAN TRANSLATION .....                                  | 17        |
| 4.3 Q-IN-Q.....                                             | 19        |
| 4.4 LOOKUP TABLE.....                                       | 22        |
| 4.5 QOS.....                                                | 26        |
| 4.6 CPU PORT .....                                          | 31        |
| 4.7 MIB .....                                               | 33        |
| 4.8 PHY .....                                               | 35        |
| 4.9 FORCE EXTERNAL INTERFACE.....                           | 38        |
| 4.10 DOT1X.....                                             | 43        |
| 4.11 PORT MIRROR .....                                      | 47        |
| 4.12 PORT ISOLATION.....                                    | 48        |
| 4.13 MAC LEARNING LIMIT .....                               | 49        |
| 4.14 ACL.....                                               | 50        |
| 4.15 MISC .....                                             | 52        |
| <b>5 IGMP SNOOPING/MLD SNOOPING.....</b>                    | <b>54</b> |
| 5.1 GENERAL DESCRIPTION .....                               | 54        |
| 5.2 IGMP SNOOPING VERSION 1.....                            | 55        |
| 5.2.1 <i>IGMP Snooping porting guide</i> .....              | 55        |
| 5.2.2 <i>IGMP snooping function usage guide:</i> .....      | 56        |
| 5.2.3 <i>Advanced issues:</i> .....                         | 58        |
| 5.3 IGMP/MLD SNOOPING VERSION 2 .....                       | 59        |
| 5.3.1 <i>Multicast Snooping porting guide</i> .....         | 59        |
| 5.3.2 <i>Multicast snooping function usage guide:</i> ..... | 60        |
| <b>6 APPLICATION.....</b>                                   | <b>63</b> |
| 6.1 XDSL .....                                              | 63        |
| 6.2 EPON ONU.....                                           | 65        |
| <b>7 PORT NUMBER.....</b>                                   | <b>70</b> |
| 7.1 EXTENSION PORTS.....                                    | 70        |
| 7.2 PORT MAPPING .....                                      | 70        |



---

|                                        |           |
|----------------------------------------|-----------|
| <b>REALTEK SEMICONDUCTOR CORP.....</b> | <b>71</b> |
| <b>HEADQUARTERS.....</b>               | <b>71</b> |

---

# 1 Overview

The RTL8306E release package contains ASIC drivers, which provides general APIs that based on user configuration to configure relative ASIC registers. Inside of the ASIC driver, it uses GPIO to emulate MDC/MDIO signal to communicate with RTL8306E. This part needs to be porting to the target platform. The RTL8306E release package can also be applied to other chips, such as RTL8304E/RTL8306M. These ASICs may have different port numbers. For example RTL8304E only have 4 ports, but RTL8306E have 6 ports. The port number descriptions for different ASICs can be found in chapter 7.

This SDK API programming guide file uses RTL8306E as an example to descript the usage of API functions. But for different ASICs, the port parameter which is needed by some API function is different from RTL8306E , due to the AISCs' different port numbers. So in order to use ASIC driver code, it's recommended for you to make clear of the ASIC's port number at first. And then make some configuration code modification based on the descriptions of sector 3.3 and chapter 7.

## 2 Directory Structure

RTL8306E release package is distributed with the following files:

| File name            | Description                                                                                                                                               |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| mdcmdio.c            | Using two GPIO pins to emulate MDC/MDIO signals to access RTL8306E PHY registers. This file should be modified when porting the driver to other platform. |
| mdcmdio.h            | MDC/MDIO API definition                                                                                                                                   |
| rtk_api.c            | Source code of ASIC driver high-level API                                                                                                                 |
| rtk_api.h            | Constant and Structure definition for ASIC driver high-level API.                                                                                         |
| rtk_api_ext.h        | API declaration for ASIC driver high-level API                                                                                                            |
| rtk_error.h          | Error code definition for ASIC driver high-level API                                                                                                      |
| rtl8306e_asicdrv.c   | Source code of ASIC driver low-level API                                                                                                                  |
| rtl8306e_types.h     | Data types for ASIC driver low-level API                                                                                                                  |
| rtl8306e_asicdrv.h   | Constant and API declaration for ASIC driver low-level API.                                                                                               |
| rtl8306e_asictypes.h | Port macro definitions for ASICs                                                                                                                          |

## 3 ASIC Driver

### 3.1 Porting issue

RTL8306E management interface is MDC/MDIO, MDC is clock and MDIO transmits data. The source code using two GPIO pins to emulate MDC/MDIO signals. Porting this driver to customer platform needs to modify files **mdcmdio.c** and **mdcmdio.h**. In these files, there are two kinds of realization, one is realized by calling GPIO API, the other is by configuring CPUGPIO register directly, any one of them is ok. If you want to write your own MDC/MDIO code, you should follow these steps:

Firstly, choose two GPIO pins, specify one as MDC, the other as MDIO;

Secondly, rewrite functions **\_smiZbit**, **\_smiReadBit**, **\_smiWriteBit**, these are local functions. **\_smiZbit** sets two pins low to simulate HiZ state. **\_smiReadBit** generates 1 -> 0 transition and sampled at 1 to 0 transition time, thus read one bit from MDC/MDIO interface. **\_smiWriteBit** generates 0 -> 1 transition and put data ready during 0 to 1 whole period, it write one bit into the interface. Between MDC 1 and 0 transition, proper delay should be added, delay time will affects the interface accessing speed, the less delay, the higher speed. In IEEE 802.3 standard, it says that MDC is an aperiodic signal that has no maximum high or low times, but minimum high and low times for MDC shall be 160 ns each, and the minimum period for MDC shall be 400 ns. Please obey this limitation.

Thirdly, **smiRead** and **smiWrite** need almost no modification, **smiRead** reads one PHY register content, **smiWrite** writes value into PHY register, both of them are global functions.

You should pay attention to another issue that is to prevent CPU from being interrupted by IRQ during smi read and write, so in **smiRead** and **smiWrite**, CPU interrupt should be disabled before smi operation and be enabled after smi operation. We use **rtlglue\_drvMutexLock()** and **rtlglue\_drvMutexUnlock()** which are based on RTL865X platform to realize that, if using other platform, please rewrite it.

## 3.2 Initialization

The system must be properly initialized, following figure shows a complete initialization, please pay attention to sequence of it. If you do not use the module, the module need not be initialized.

After ***rtk\_switch\_init***, ***rtk\_qos\_init*** should be firstly called, then ***rtk\_cpu\_enable\_set*** and ***rtk\_cpu\_tagPort\_set***. ***rtk\_qos\_init*** is to configure port Tx queue numbers, it could support 4 queues at most, default only one queue. Multi-queues have different priority, higher queue could share more bandwidth of the port, thus port could be differentiated maximum four flows. ***rtk\_cpu\_tagPort\_set*** is to specify which port is CPU port and the CPU tag insert mode.

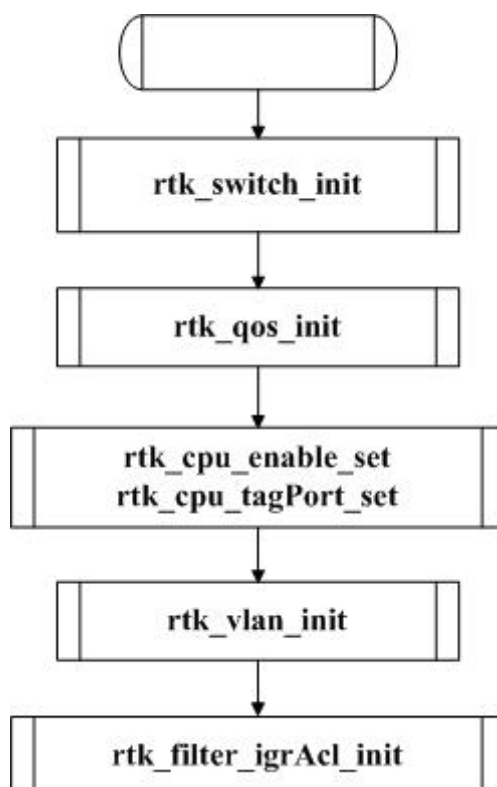


Fig. 3-1 Initialization

## 3.3 Chip module

The ASIC driver supports multiple Realtek switch ASIC family. Users should define one of the following flags in the file “rtl8306e\_asictypes.h” depending on the ASIC which you use. Then the API driver for the ASIC would be compiled and linked into your program. Those code segments for un-supported chip would not be compiled and linked, and will not increase your program size.

- **CHIP\_RTL8306E**
- **CHIP\_RTL8306M**
- **CHIP\_RTL8304E**

Before using the driver code, you have to define one of the above flags depend on the type of the

ASIC you used. When you use the API functions which need port number as input parameter, you are only permitted to use the port macro definitions corresponded to the flag. When using the API functions which need port mask as input parameters, you are recommended to use a macro to translate these ASIC's port mask value to the API's port mask value. After using the API functions which return port mask as output parameters, you are recommended to use a macro to translate these output parameters to ASIC's port mask value. Only RTL8306E and RTL8306M are permitted to use Arabic numerals as port and port mask, and it's not needed to use macro to do the port mask value translation. The port macros and port mask translation macro are defined in "rtl8306e\_asictypes.h".

The port macro and port mask translation macro definitions corresponded to the dedicated flag are listed as follows:

(1) CHIP\_RTL8306E:

RTL8306E\_PORT0  
RTL8306E\_PORT1  
RTL8306E\_PORT2  
RTL8306E\_PORT3  
RTL8306E\_PORT4  
RTL8306E\_PORT5

(2) CHIP\_RTL8306M:

RTL8306M\_PORT0  
RTL8306M\_PORT1  
RTL8306M\_PORT2  
RTL8306M\_PORT3  
RTL8306M\_PORT4  
RTL8306M\_PORT5

(3) CHIP\_RTL8304E:

RTL8304E\_PORT0  
RTL8304E\_PORT1  
RTL8304E\_PORT2  
RTL8304E\_PORT3  
RTL8304E\_PORTMASKIN(x)  
RTL8304E\_PORTMASKOUT(x)

For example, if you use ASIC RTL8304E which have only 4 ports (port0~1 and external port 0~1, for detail please see chapter 7), so before to use the driver code, first "rtl8306e\_asictypes.h" should be included in your code files. You have to define macro CHIP\_RTL8304E in "rtl8306e\_asictypes.h", and then you are only permitted to use port macros for RTL8304E as the port parameter of the API function.

For example, if you want to call API function:

***rtk\_api\_ret\_t rtk\_cpu\_tagPort\_set(rtk\_port\_t port, rtk\_cpu\_insert\_t mode)***

You are only permitted to use RTL8304E\_PORT0, RTL8304E\_PORT1, RTL8304E\_PORT2 or RTL8304E\_PORT3 as port parameter for this API function.

If you want to call API function which use port mask as input parameters as follows:

***rtk\_api\_ret\_t rtk\_vlan\_set(rtk\_vlan\_t vid, rtk\_portmask\_t mbrmsk, rtk\_portmask\_t untagmsk, rtk\_fid\_t fid)***

***mbrmsk*** and ***untagmsk*** are input parameters which are also port mask. If the vlan entry you want to set has vid = 2, fid = 0, member port set = port0 and port2, and untag member port set = port0 and port3. So:

***vid = 2;***

***fid = 0;***

***mbrmsk = 0x5; /\*port 0 and port 2\*/***

***untagmsk = 0x9; /\*port 0 and port 3\*/***

***mbrmsk = RTL8304E\_PORTMASKIN (mbrmsk);***

***untagmsk = RTL8304E\_PORTMASKIN (untagmsk).***

***rtk\_vlan\_set(vid, mbrmsk, untagmsk, fid);***

Then you can call ***rtk\_vlan\_set()*** using ***mbrmsk*** and ***untagmsk*** as input parameters now.

If you want to call API function which use port mask as output parameters as follows:

***rtk\_api\_ret\_t rtk\_vlan\_get(rtk\_vlan\_t vid, rtk\_portmask\_t \*pMbrmsk, rtk\_portmask\_t \*pUntagmsk, rtk\_fid\_t \*pFid)***

***\*pMbrmsk*** and ***\*pUntagms*** are output parameters which are also port mask returned by this API. After call ***rtk\_vlan\_get()*** and you will get ***\*pMbrmsk*** and ***\*pUntagm***. Before using them directly, you have to call macro RTL8304E\_PORTMASKOUT(x) to translate them into ASIC's port mask values corresponded to RTL8304E. You have to call:

***\*pMbrms = RTL8304E\_PORTMASKOUT (\*pMbrms).***

***\*pUntagms = RTL8304E\_PORTMASKOUT (\*pUntagms).***

Then you can use ***\*pMbrmsk*** and ***\*pUntagms*** as normal RTL8304E port mask variables now.

## 4 RTK API for RTL8306E

### 4.1 VLAN

RTL8306E supports 16 VLANs. After the switch resets, it is dumb mode, the VLAN function is disabled as the default configuration. User should use API ***rtk\_vlan\_init*** to initialize the VLAN before using VLAN. The initialization will enable VLAN function and set up a default VLAN with VID 1 that contains all ports. Moreover, each port's PVID will also be set to default VLAN.

After initialization, user can add and set VLAN 1~4094 through API ***rtk\_vlan\_set***, configure its member set and untag set. In order to get the information of member set and untag set on specified VLAN, ***rtk\_vlan\_get*** could be used. For Port-Based VLAN, there are two APIs ***rtk\_vlan\_portPvid\_set*** and ***rtk\_vlan\_portPvid\_get*** provided to set and get the PVID of the ports.

User can also turn on ingress filter function through ***rtk\_vlan\_portIgrFilterEnable\_set*** and accept frame type function through ***rtk\_vlan\_portAcceptFrameType\_set***.

Here are explanations and example codes for VLAN APIs. The example codes ignore the error check.

---

#### ***rtk\_api\_ret\_t rtk\_vlan\_init(void)***

This function enables VLAN and gives the default configuration, and it should be called before using VLAN. It will clear the VLAN table and set up a default VLAN with VID 1 that contains all ports. It also sets each port's PVID to the default VLAN. After VLAN initialization, it will be set tag aware, the switch uses tagged-VID VLAN mapping for tagged frames while using Port-Based VLAN mapping for untagged frames. After invoking ***rtk\_vlan\_init***, user can change the default VLAN configuration through the following introduced API ***rtk\_vlan\_set*** and ***rtk\_vlan\_portPvid\_set***.

Example:

```
/* initialize VLAN */
rtk_vlan_init ();

/* all the ports are in the default VLAN 1 after VLAN initialized, modify it as follows
VLAN1 member: port0, port1, port2;
VLAN2 member: port3, port4, port5 */
rtk_portmask_t mbrmsk, untagmsk;
mbrmsk.bits[0]=0x07;
untagmsk.bits[0]=0x3F;
rtk_vlan_set(1, mbrmsk, untagmsk, 0);
mbrmsk.bits[0]=0x38;
untagmsk.bits[0]=0x3F;
rtk_vlan_set(2, mbrmsk, untagmsk, 0);

/* set PVID for each port */
```



```
rtk_vlan_portPvid_set(0, 1, 0);
rtk_vlan_portPvid_set(1, 1, 0);
rtk_vlan_portPvid_set(2, 1, 0);
rtk_vlan_portPvid_set(3, 2, 0);
rtk_vlan_portPvid_set(4, 2, 0);
rtk_vlan_portPvid_set(5, 2, 0);
```

---

***rtk\_api\_ret\_t rtk\_vlan\_set(rtk\_vlan\_t vid, rtk\_portmask\_t mbrmsk, rtk\_portmask\_t untagmsk, rtk\_fid\_t fid)***

```
typedef uint32  rtk_vlan_t;
typedef struct rtk_portmask_s
{
    uint32 bits[RTK_TOTAL_NUM_OF_WORD_FOR_1BIT_PORT_LIST];
} rtk_portmask_t;
typedef uint32  rtk_fid_t;
```

User could configure the member set and untag set for specified VLAN through this API. The VID should be 1~4094. The *mbrmsk*'s bit N means port N. For example, *mbrmsk.bits[0] = 23=0x17=0b010111* means port 0,1,2,4 in the member set. The input parameter *untagmsk* is used to configure untag set for specified VLAN, which means if the *untagmsk* bit value is 1, the packets egress from the corresponding port will be removed VALN tag. The input parameter *fid* is for SVL/IVL usage, 8306E only supports SVL, so *fid* is no usage and can be any value.

Example:

```
/* set port 0,1,2,3 as member set and port 2,3 as untag set to VLAN 1000 */
rtk_vlan_t vid;
rtk_portmask_t mbrmsk, untagmsk;
rtk_fid_t fid;

vid = 1000;
mbrmsk.bits[0]=0x0F;
untagmsk.bits[0]=0x0C;
fid = 0;
rtk_vlan_set(vid, mbrmsk, untagmsk, fid);
```

---

***rtk\_api\_ret\_t rtk\_vlan\_get(rtk\_vlan\_t vid, rtk\_portmask\_t \*pMbrmsk, rtk\_portmask\_t \*pUntagmsk, rtk\_fid\_t \*pFid)***

```
typedef uint32  rtk_vlan_t;
typedef struct rtk_portmask_s
{
    uint32 bits[RTK_TOTAL_NUM_OF_WORD_FOR_1BIT_PORT_LIST];
} rtk_portmask_t;
typedef uint32  rtk_fid_t;
```

The API can get the information of member set, untagged set, and filtering database on the specified

VLAN. The information is retrieved in *mbrmask*. The *mbrmask*'s bit N means port N. For example, *mbrmask.bits[0] = 23=0x17=0b010111* means port 0,1,2,4 in the member set. 8306E only supports SVL, so the *pFid* value retrieved is meaningless.

Example:

```
/* get the member set and untagged set on VLAN 1000 */
rtk_vlan_t vid;
rtk_portmask_t mbrmsk, untagmsk;
rtk_fid_t fid;

vid = 1000;
rtk_vlan_get(vid, &mbrmsk, &untagmsk, &fid);
```

---

***rtk\_api\_ret\_t rtk\_vlan\_portPvid\_set(rtk\_port\_t port, rtk\_vlan\_t pvid, rtk\_pri\_t priority)***

Set each port's PVID and 802.1Q based default priority for the port. The specified priority will only be assigned to the untagged frame and then system can map the untagged frame to the proper output queue for 802.1Q-based QoS. Just specify the priority to 0 if you don't turn on the QoS or use other QoS mechanisms instead of 802.1Q-based. Before using the API, user should call ***rtk\_vlan\_set*** to set the member set and untag set for the VLAN first.

Example:

```
/* set port0, port1 as member set to VLAN 2000 */
rtk_vlan_t vid;
rtk_portmask_t mbrmsk, untagmsk;
rtk_fid_t fid;

vid = 2000;
mbrmsk.bits[0]=0x03;
untagmsk.bits[0]=0;
fid = 0;
rtk_vlan_set(vid, mbrmsk, untagmsk, fid);

/* set PVID of port 0/1 to 2000 and 802.1Q based default priority to 0*/
rtk_vlan_portPvid_set(0, 2000, 0);
rtk_vlan_portPvid_set(1,2000, 0);
```

---

***rtk\_api\_ret\_t rtk\_vlan\_portPvid\_get(rtk\_port\_t port, rtk\_vlan\_t \*pPvid, rtk\_pri\_t \*pPriority)***

Get each port's PVID and 802.1Q based default priority for the port.

Example:

```
/* get PVID and 802.1Q based default priority of port1 */
rtk_vlan_t vid;
rtk_pri_t priority;
```

---

```
rtk_vlan_portPvid_get(1, &vid, &priority);
```

---

---

***rtk\_api\_ret\_t rtk\_vlan\_portAcceptFrameType\_set(rtk\_port\_t port,  
rtk\_vlan\_acceptFrameType\_t accept\_frame\_type)***

Acceptable frame type could be set for each port, and priority-tagged packet (VID = 0) is treated as untagged packet.

Example:

```
/* port 0: accept tagged and un-tagged packet,*/  
/* port 1: accept only tagged packet,*/  
/* port 2: accept only un-tagged packet.*/  
rtk_vlan_portAcceptFrameType_set (0, ACCEPT_FRAME_TYPE_ALL);  
rtk_vlan_portAcceptFrameType_set (1, ACCEPT_FRAME_TYPE_TAG_ONLY);  
rtk_vlan_portAcceptFrameType_set (2, ACCEPT_FRAME_TYPE_UNTAG_ONLY);
```

---

---

***rtk\_api\_ret\_t rtk\_vlan\_portAcceptFrameType\_get(rtk\_port\_t port,  
rtk\_vlan\_acceptFrameType\_t \*pAccept\_frame\_type)***

Get the acceptable frame type of the port.

Example:

```
/* get acceptable frame type of port 0 */  
rtk_vlan_acceptFrameType_t acceptFrameType;  
rtk_vlan_portAcceptFrameType_get (0, &acceptFrameType);
```

---

---

***rtk\_api\_ret\_t rtk\_vlan\_destroy(rtk\_vlan\_t vid)***

Delete an existed VLAN, which means its VID, port member set and untagged set in VLAN table all will be set to 0. If the VLAN does not exist, it returns RT\_ERR\_VLAN\_ENTRY\_NOT\_FOUND.

Example:

```
/* delete the VLAN whose VID is 2007 */  
rtk_vlan_destroy(2007);
```

---

---

***rtk\_api\_ret\_t rtk\_vlan\_portIgrFilterEnable\_set (rtk\_port\_t port, rtk\_enable\_t igr\_filter)***

Set VLAN ingress for each port. RTL8306E use one ingress filter for whole system, not for each port, so any port you set will affect all ports ingress filter setting. While VLAN function is enabled, ASIC will decide VLAN ID for each received frame and get belonged member ports from VLAN table. If the ingress port is not included in VLAN member ports, ASIC will drop received frame if VLAN ingress function is enabled.

Example:

---

```
/*Enable the VLAN ingress filter function for all ports*/  
rtk_vlan_portIgrFilterEnable_set(0, ENABLED);
```

```
/*Disable the VLAN ingress filter function for all ports*/  
rtk_vlan_portIgrFilterEnable_set(0, DISABLED);
```

---

### ***rtk\_api\_ret\_t rtk\_vlan\_vlanBasedPriority\_set(rtk\_vlan\_t vid, rtk\_pri\_t priority)***

For 8306E, each VLAN could be assigned a priority, which means the VLAN ID from IEEE802.1Q VLAN tag can be mapped to a 2-bit priority. This API will enable VID based priority function for the specified VID and set VID-to-priority mapping.

Example:

```
/*Set priority 2 for VID 100*/  
rtk_vlan_t vid;  
rtk_pri_t priority ;  
vid = 100;  
priority = RTL8306_PRIO2;  
  
rtk_vlan_vlanBasedPriority_set(vid, priority);
```

---

### ***int32 rtl8306e\_vlan\_tagInsert\_set(uint32 egPort, uint32 igPortMsk)***

The RTL8306E can insert VLAN tag with the ingress port's PVID for untagged packets if the egress port's insert tag mask register setting is enable for the ingress port. *egPort* is packet egress port, if the packet is untagged and its ingress port is in the *igPortMsk*, it will be inserted with a VLAN tag. But if ingress packet already has a VALN tag or its ingress port is not in the *igPortMsk*, then VALN tag will not be inserted.

Example:

```
/*set the packets ingress from port1/3/5 to be inserted a VALN tag when they are transmitted out of  
port0 */  
rtl8306e_vlan_tagInsert_set(0, 0x2A);
```

## 4.2 VLAN Translation

The RTL8306E supports VLAN translation feature. With the VID-to-VID mapping in each VLAN entry, the RTL8306E can translate the VLAN IDs from private networks into those used in the service providers' network, and vice versa. The UNI (User-Network Interface) is the interface to the customers' private network and the NNI (Network-Network Interface) is the interface to the service providers' network. Each port of the RTL8306E can be and must be configured as NNI or UNI in VLAN translation application. The VLAN translation can take effect only when the traffic passes across these two kinds of interface. There will be no tag change to the packets which are transmitted from NNI to NNI or from UNI to UNI.

Here are explanations and example codes for VLAN translation API. The example codes ignore the error check.

---

***rtk\_api\_ret\_t rtk\_vlan\_vidTranslationEnable\_set(rtk\_enable\_t enable, rtk\_portmask\_t nniMask)***

This function sets vlan translation function enabled or disabled, and decides which port is NNI port or UNI port. In *nniMask*, bit *n* means port *n*, and bit value 1 means the port is NNI port, 0 means the port is UNI port

Example:

```
/*Enable VLAN translation,*/  
/*UNI port: port0, port1, port2, port3;*/  
/*NNI port: port4, port5 */  
rtk_vlan_vidTranslationEnable_set (ENABLED, 0x30);
```

---

***rtk\_api\_ret\_t rtk\_vlan\_vidTranslationEnable\_get(rtk\_enable\_t \*pEnable, rtk\_portmask\_t \*pNniMask)***

This function gets vlan translation function status (enabled or disabled), and which port is NNI port or UNI port. In *nniMask*, bit *n* means port *n*, and bit value 1 means the port is NNI port, 0 means the port is UNI port.

Example:

```
/*Get VLAN translation status*/  
rtk_enable_t enable;  
rtk_portmask_t nniMask;  
rtk_vlan_vidTranslationEnable_get (&enable, &nniMask);
```

---

***rtk\_api\_ret\_t rtk\_vlan\_vidTranslation\_set(rtk\_vlan\_t vid, rtk\_vlan\_t nvid)***

This API is used to translate a VID to a new VID when VLAN translation happens. The new VID will be set into the VLAN table entry and could be used by Q-in-Q or vlan translation function. Before

using this API, please make sure the old VID already exists in the VLAN table, otherwise it will return RT\_ERR\_VLAN\_ENTRY\_NOT\_FOUND.

Example:

```
/*Suppose VID 1 already exists in VLAN table, translate VID 1 to new VID 200 */  
rtk_vlan_t vid, nvid ;  
  
vid = 1 ;  
nvid = 200 ;  
rtk_vlan_vidTranslation_set (vid, nvid);
```

## 4.3 Q-in-Q

RTL8306E support QinQ function which allows tagging the tagged packet. The format of the double tagged packet is shown below. The Tag Protocol Identifiers (TPID) of the inner tag and the outer tag can be configured via registers. When QinQ function is enabled, the RTL8306E will always insert outer tag to the packet when the egress port is NNI and remove outer tag on UNI port. There will be no tag change to the packets which are transmitted from NNI to NNI or from UNI to UNI. The QinQ Function and VLAN translation can not be employed at the same time.

Here are explanations and example codes for Q-in-Q API. The example codes ignore the error check.

---

### ***rtk\_api\_ret\_t rtk\_svlan\_init(void)***

This API should be called before using SVLAN. It can setup some rules for SVLAN. After calling this API, all ports are set as CVLAN port. User can use ***rtk\_svlan\_servicePort\_add*** to add SVLAN port.

Example:

```
/* initialize SVLAN */  
rtk_svlan_init ();
```

---

### ***rtk\_api\_ret\_t rtk\_svlan\_servicePort\_add(rtk\_port\_t port)***

This API can set the port as NNI port (connected to provider switch). All frames received from this port will recognize Service VLAN Tag.

Example:

```
/* set port 5 as SVLAN service port */  
rtk_svlan_servicePort_add (5);
```

---

### ***rtk\_api\_ret\_t rtk\_svlan\_servicePort\_del(rtk\_port\_t port)***

The API is used to disable one service port in the specified device

Example:

```
/* delete port 5 from service port list */  
rtk_svlan_servicePort_del (5);
```

---

### ***rtk\_api\_ret\_t rtk\_svlan\_tpidEntry\_set(rtk\_svlan\_tpid\_t svlan\_tag\_id)***

The API is used to set S-VLAN tag TPID, and the default value is 0x88a8. For example, when TPID is 0x88a8, the egress tag with VID 1 should be 88-A8-00-01.

Example:

```
/* Set TPID to 0x9100*/
```

---

```
rtk_svlan_tpidEntry_set(0x9100);
```

---

### ***rtk\_api\_ret\_t rtk\_svlan\_portPvid\_set(rtk\_port\_t port, rtk\_vlan\_t pvid, rtk\_pri\_t priority, rtk\_dei\_t dei)***

The API is used to set port default outer tag VID(SVLAN PVID) and default outer tag priority. The input parameter is *dei* meaningless for 8306E.

Example:

```
/* set port0's SVLAN PVID to 1000, default outer tag priority to 2*/  
/* set port1's SVLAN PVID to 2000, default outer tag priority to 3*/  
rtk_svlan_servicePort_add (0);  
rtk_svlan_servicePort_add (1);  
rtk_svlan_portPvid_set(0, 1000, 2, 0);  
rtk_svlan_portPvid_set(1, 2000, 3, 0);
```

---

### ***rtk\_api\_ret\_t rtk\_vlan\_tagSrc\_set(rtk\_port\_t port, rtk\_vidSrc\_t vidSrc, rtk\_priSrc\_t priSrc)***

The API could set outer tag or translated new VLAN tag vid/priority source, when Q-in-Q or VLAN translation modifies VLAN tag.

*vidSrc* could be set following macros:

RTL8306E\_VIDSRC\_POVID - port default outer tag VID  
RTL8306E\_VIDSRC\_NVID - New VID got by VLAN table search

*priSrc* could be set following macros:

RTL8306E\_PRISRC\_PPRI - port default outer tag priority  
RTL8306E\_PRISRC\_1PRMK - original VLAN tag priority or 1p remarking priority(optional)

The port default outer tag VID and priority could be configured by API *rtk\_svlan\_portPvid\_set*.

Example:

```
/*Set the packet transmitted out of port 2 when VLAN translation happens, its new tag VID is*/  
/*decided by NEW VID got by VLAN table search, and priority is decided by original tag priority or*/  
/*1p remarking priority.*/  
rtk_vlan_tagSrc_set (2, RTL8306E_VIDSRC_NVID, RTL8306E_PRISRC_1PRMK);
```

---

### ***rtk\_api\_ret\_t rtk\_vlan\_tagSrc\_get(rtk\_port\_t port, rtk\_vidSrc\_t \*pVidSrc, rtk\_priSrc\_t \*pPriSrc)***

The API could get the tag vid and priority source for Q-in-Q and VLAN translation.

Example:

```
/*Get the packets transmitted out of port 5 when VLAN translation happens, its new tag vid*/  
/*and priority source.*/  
rtk_vidSrc_t vidSrc;
```

---



```
rtk_priSrc_t priSrc;
```

```
rtk_vlan_tagSrc_get (5, &vidSrc, &priSrc);
```

## 4.4 Lookup table

RTL8306E supports 2K-entry lookup table which supports 4-way hashing lookup, CPU could access lookup table through MDC/MDIO interface. Hashing function generates 9-bit lookup index which is the combination of MAC[13:15] and MAC[0:5], for each index, there are 4-entry to save the hash collision MAC address.

MAC address stored in the lookup table does not have any constrain which means I/G-bit can be one or zero. When address with I/G-bit set to be zero, the address is unicast address, while I/G-bit one means multicast address.

Unicast address property includes static or dynamic, authorized or unauthorized. Dynamic unicast MAC address entry is auto-learned by ASIC, and after a period of time(aging time, normal 300 second), it will be aged out and removed. Static unicast MAC address is only written by CPU through ***rtk\_l2\_addr\_add***, it will exist in the lookup table all along until application use API ***rtk\_l2\_addr\_del*** to delete it. Authorized property is also assigned by CPU, it is for IEEE 802.1x application, authorized MAC address also never age out.

Multicast address is for multicast application such as IP multicast, which could not be auto-learned but could be managed by CPU through ***rtk\_l2\_mcastAddr\_add***. If a multicast address has been written into lookup table, packets sent to the address will be properly forwarded according to the port mask configured. Thus, they will not be flooded to all ports anymore. Multicast MAC address entry could be deleted by the API ***rtk\_l2\_mcastAddr\_del***.

Here are explanations and example codes for lookup table API. The example codes ignore the error check.

---

```
rtk_api_ret_t rtk_l2_addr_add(rtk_mac_t *pMac, rtk_l2_ucastAddr_t *pL2_data)
```

```
typedef struct rtk_mac_s
{
    uint8_t octet[ETHER_ADDR_LEN];
} rtk_mac_t;

typedef struct rtk_l2_ucastAddr_s
{
    rtk_mac_t mac;
    uint32_t fid;
    uint32_t port;
    uint32_t sa_block;
    uint32_t auth;
    uint32_t is_static;
} rtk_l2_ucastAddr_t;
```

Add a unicast entry to lookup table. The lowest order bit of the highest order octet (I/G-bit) of MAC address must be 0. If hashed index is full of entries that maintained by CPU, it would return error message "RT\_ERR\_L2\_INDEXTBL\_FULL". CPU should maintain a database to record the MAC

addresses written to lookup table for further deletion.

Example:

```
/* write a unicast static entry with MAC address "00-12-34-56-78-AA", */
/*source port 0, IEEE 802.1x authorized */
rtk_l2_ucastAddr_t l2_entry;
rtk_mac_t mac;
mac.octet[0] = 0x00;
mac.octet[1] = 0x12;
mac.octet[2] = 0x34;
mac.octet[3] = 0x56;
mac.octet[4] = 0x78;
mac.octet[5] = 0xAA;
memset(&l2_entry, 0x00, sizeof(rtk_l2_ucastAddr_t));
l2_entry.port = 0;
l2_entry.is_static = 1;
l2_entry.auth = 1;

rtk_l2_addr_add (&mac, &l2_entry);
```

---

***rtk\_api\_ret\_t mcastAddr\_add(rtk\_mac\_t \*pMac, rtk\_fid\_t fid, rtk\_portmask\_t portmask)***

This API adds a multicast entry to lookup table. The lowest order bit of the highest order octet (I/G-bit) of MAC address must be 1. The input parameter *fid* is meaningless for RTL8306E switch and could be any value.

Example:

```
/* write a multicast entry with MAC address "01-00-5E-11-22-33" and member port 0,1,2 */
rtk_mac_t mac;
rtk_fid_t fid;
rtk_portmask_t portmask;

mac.octet[0] = 0x01;
mac.octet[1] = 0x00;
mac.octet[2] = 0x5E;
mac.octet[3] = 0x11;
mac.octet[4] = 0x22;
mac.octet[5] = 0x33;
fid = 0;
portmask.bist[0] = 0x07;

rtk_l2_mcastAddr_add(&mac, fid, portmask);
```

***rtk\_api\_ret\_t rtk\_l2\_addr\_del(rtk\_mac\_t \*pMac, rtk\_fid\_t fid)***

Delete a unicast entry from lookup table. The input parameter *fid* is meaningless for RTL8306E switch and could be any value.

Example:

```
/* delete a entry with MAC address "00-12-34-56-78-AA" */
rtk_mac_t mac;
rtk_fid_t fid;

mac.octet[0] = 0x00;
mac.octet[1] = 0x12;
mac.octet[2] = 0x34;
mac.octet[3] = 0x56;
mac.octet[4] = 0x78;
mac.octet[5] = 0xAA;
fid = 1;

rtk_l2_addr_del(&mac, fid);
```

---

***rtk\_api\_ret\_t rtk\_l2\_mcastAddr\_del(rtk\_mac\_t \*pMac, rtk\_fid\_t fid)***

Delete a multicast entry from lookup table. The input parameter *fid* is meaningless for RTL8306E switch and could be any value.

Example:

```
/* delete a entry with MAC address "01-00-5E-11-22-33" */
rtk_mac_t mac;
rtk_fid_t fid;

mac.octet[0] = 0x01;
mac.octet[1] = 0x00;
mac.octet[2] = 0x5E;
mac.octet[3] = 0x11;
mac.octet[4] = 0x22;
mac.octet[5] = 0x33;
fid = 0;

rtk_l2_mcastAddr_del(&mac, fid);
```

---

***rtk\_api\_ret\_t rtk\_l2\_addr\_get(rtk\_mac\_t \*pMac, rtk\_fid\_t fid, rtk\_l2\_ucastAddr\_t \*pL2\_data)***

Get a unicast entry from lookup table by MAC and FID. The input parameter *fid* is meaningless for RTL8306E switch and could be any value.

Example:

```
/* Get a unicast entry with MAC address "00-12-34-56-78-AA" */  
rtk_l2_ucastAddr_t l2_entry;  
rtk_mac_t mac;  
rtk_fid_t fid;
```

```
memset(&l2_entry, 0x00, sizeof(rtk_l2_ucastAddr_t));  
mac.octet[0] = 0x00;  
mac.octet[1] = 0x12;  
mac.octet[2] = 0x34;  
mac.octet[3] = 0x56;  
mac.octet[4] = 0x78;  
mac.octet[5] = 0xAA;  
fid = 0;
```

```
rtk_l2_addr_get(&mac, fid, &l2_entry);
```

---

### ***rtk\_api\_ret\_t rtk\_l2\_mcastAddr\_get(rtk\_mac\_t \*pMac, rtk\_fid\_t fid, rtk\_portmask\_t \*pPortmask)***

Get a multicast MAC entry from lookup table by MAC and FID. The input parameter *fid* is meaningless for RTL8306E switch and could be any value.

Example:

```
/* Get a multicast entry with MAC address "01-00-5E-56-78-AA" with fid 0*/  
rtk_mac_t mac;  
rtk_fid_t fid;  
rtk_portmask_t portmask;
```

```
mac.octet[0] = 0x01;  
mac.octet[1] = 0x00;  
mac.octet[2] = 0x5E;  
mac.octet[3] = 0x56;  
mac.octet[4] = 0x78;  
mac.octet[5] = 0xAA;  
fid = 0;
```

```
rtk_l2_mcastAddr_get(&mac, fid, &portmask);
```

## 4.5 QOS

RTL8336E QoS function provides maximum 4 queues per port for packet scheduling with queue weight and priority assignment. Priority assignment specifies the priority of each received packet according to 7 types of QoS priority sources as show in figure.4-2. Among them, CPU tag-based priority will be adopted if it is enabled. The second priority is IP address-based priority. The other five priority sources are decided based on priority selection table which could be defined by user through API **rtk\_qos\_priSel\_set**. Priority assignment source with highest level will be selected and ASIC will use its mapped priority to locate desired queue for outputs.

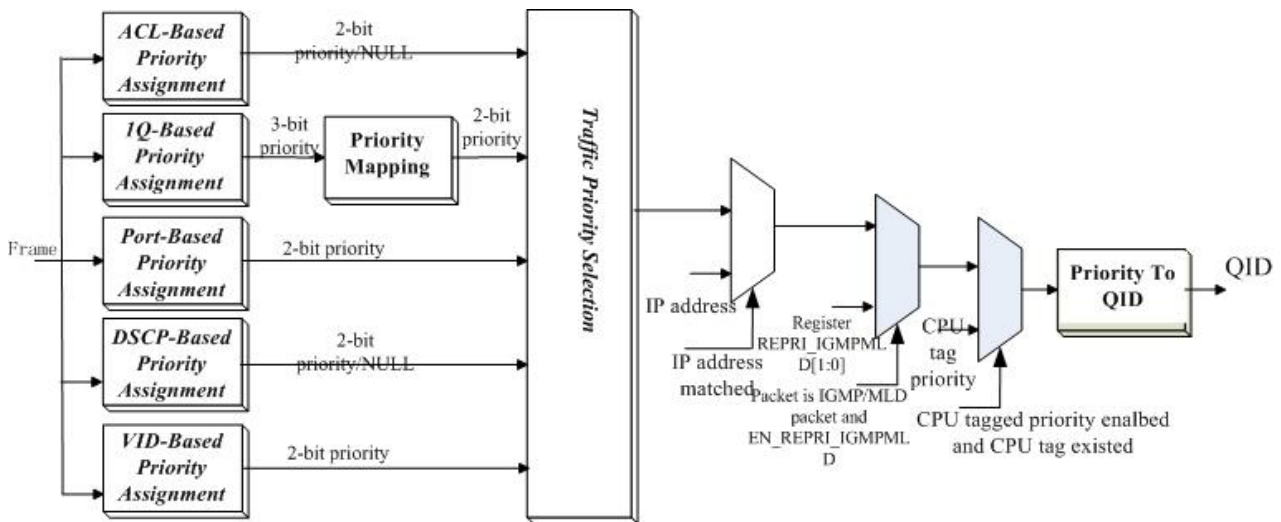


Fig. 4-1 RTL8306E Priority Assignment Diagram

If priority assignment source have the same selected priority, then ASIC will pick up the one with the largest value to be the desired queuing priority.

Each port has 4 Tx queues at most, which are queue 0~3, and packet will be classified into queues according to priority. The mapping relation between priority and QID could be specified by user. Four queues have different priority, queue 3 > queue 2 > queue 1 > queue 0, higher priority queue will occupy more bandwidth of CPU port when there is no queue rate limitation, it means that if all queues are full all along, queue 3 could transmit more packets than other queues, queue 0 transmits least ones. In four queues, only queue 3 and queue 2 equip leaky bucket to control queue rate. Packet scheduling block diagram is shown in following figure.

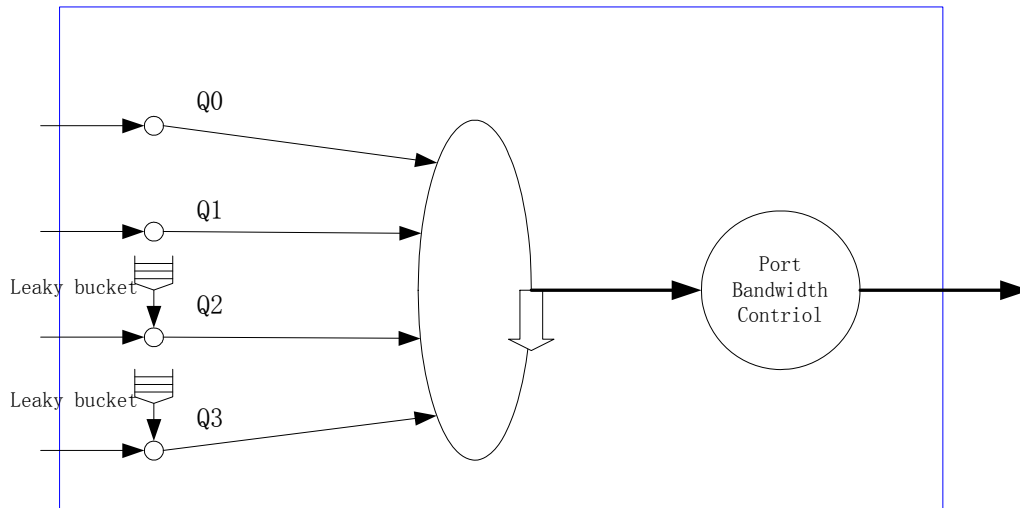


Fig. 4-2 Packet scheduling block diagram

There is a mapping table that defines the mapping between priority and queue id. After packet priority is decided, the packet will be put into table mapped queue. This table could be configured by calling API *rtk\_qos\_priMap\_set*. Here is default mapping table configuration when queue number is selected 4, 3, 2 and 1 after initializing QOS function by API *rtk\_qos\_init*. An example is given to explain the table, in 4 queues situation, packet with priority 0 will be classified into queue 0, packet with priority 1 will be classified into queue 1, packet with priority 2 into queue 2, packet with priority 3 into queue 3.

| Priority   | Queue numbers |   |   |   |
|------------|---------------|---|---|---|
|            | 1             | 2 | 3 | 4 |
| 0(default) | 0             | 0 | 0 | 0 |
| 1          | 0             | 0 | 1 | 1 |
| 2          | 0             | 1 | 1 | 2 |
| 3          | 0             | 1 | 2 | 3 |

Fig. 4-3 mapping table for priority and queue id

Besides, RTL8306E supports bandwidth control function. For each port, both input and output rate could be limited to a specified speed, rate control unit is 64Kbps, the maximum rate is  $1526 \times 64\text{Kbps} = 100\text{Mbps}$ . User could configure the bandwidth control by calling API *rtk\_rate\_igrBandwidthCtrlRate\_set* and *rtk\_rate\_egrBandwidthCtrlRate\_set*. (1Kbps = 1024bps, 1Mbps = 1000,000bps)

Here are explanations and example codes for QOS API. The example codes ignore the error check

---

***rtk\_api\_ret\_t* rtk\_qos\_init(*rtk\_queue\_num\_t* queueNum)**

The *queueNum* is Tx queue number for all ports, 1 ~ 4 is permitted. This API will also assign

different flow control thresholds for different queue number applications, and create a default mapping table for priority and queue id (Fig. 4-4) to locate frame to mapping queue with different priority in different queue number situation.

Example:

```
/* set 4 queues usage for each port */  
rtk_qos_init(4);
```

---

**`rtk_api_ret_t rtk_qos_priSel_set(rtk_priority_select_t *pPriDec)`**

As aforementioned, 8306E could recognize 7 types of priority sources at most, and a packet properly has all of them. Among them, 5 types of priority sources (ACL-based priority, DSCP-based priority, 1Q-based priority, Port-based priority, VID- based priority) could be set weight by user through the API **`rtk_qos_priSel_set`**. Each priority source could be set weight from 1 to 5, and arbitration module will decide their sequence to take. ASIC will select the highest weight assignment source's priority to decide the final priority of the packet.

Example:

```
/* set priority arbitration weight with 802.1q > dscp > port based > ACL > VID */  
rtk_priority_select_t priDec;  
priDec.dot1q_pri = 5;  
priDec.dscp_pri = 4;  
priDec.port_pri = 3;  
priDec.acl_pri = 2;  
priDec.vid_pri = 1;  
  
rtk_qos_priSel_set(&priDec);
```

---

**`rtk_api_ret_t rtk_qos_portPri_set(rtk_port_t port, rtk_pri_t int_pri)`**

This API is used to configure the ingress port-based priority, and this priority should be ranged 0~3.

Example:

```
/* Set port 0~3 to priority 0 and port4~5 to priority 3 */  
rtk_qos_portPri_set(0, 0);  
rtk_qos_portPri_set(1, 0);  
rtk_qos_portPri_set(2, 0);  
rtk_qos_portPri_set(3, 0);  
rtk_qos_portPri_set(4, 3);  
rtk_qos_portPri_set(5, 3);
```

---

**`rtk_api_ret_t rtk_qos_dscpPriRemap_set(rtk_dscp_t dscp, rtk_pri_t int_pri)`**

This API can be used to set the remapping table for translating DSCP value in IP header to internal priority. If the coming frame is not an IP frame, the priority will be NULL and ignored during priority selection process.



The value of *dscp* is selected from macro RTL8306\_DSCP\_EF to RTL8306\_DSCP\_BF, and the range of *int\_pri* is 0~3.

Example:

```
/*set RTL8306_DSCP_AFL(DSCP for the Expedited forwarding PHB, 101110 10) mapping to*/  
/*priority 1 and RTL8306_DSCP_NC(DSCP for network control, 110000 or 111000) mapping */  
/* to priority 2*/  
rtk_qos_dscpPriRemap_set(RTL8306_DSCP_AFL1, 1);  
rtk_qos_dscpPriRemap_set (RTL8306_DSCP_NC, 2);
```

---

***rtk\_api\_ret\_t rtk\_qos\_1pPriRemap\_set(rtk\_pri\_t dot1p\_pri, rtk\_pri\_t int\_pri)***

This API can set the remapping table for translating 1Q priority mapping to internal priority that is used for queue usage and packet scheduling.

The the range of *dot1p\_pri* is 0~7, and *int\_pri* is 0~3.

Example:

```
/*set 1Q priority 5 mapped to internal priority 2*/  
rtk_qos_1pPriRemap_set(5, 2);
```

---

***rtk\_api\_ret\_t rtk\_qos\_priMap\_set(rtk\_queue\_num\_t queue\_num, rtk\_qos\_pri2queue\_t \*pPri2qid)***

RTL8306E supports maximum 4 queues usage for each port and has priority mapping configuration. The API lets user define the mapping relation between internal priority and queue id. There are dedicated configurations for different queue number usage and there are different queue ids for variable queue number usage, too.

Example:

```
/*set 4 Tx queues usage and priority 0, 1 mapped to queue 0, priority 2, 3 mapped to queue 1*/  
rtk_qos_pri2queue_t pri2qid;  
  
pri2qid.pri2queue[0] = 0;  
pri2qid.pri2queue[1] = 0;  
pri2qid.pri2queue[2] = 1;  
pri2qid.pri2queue[3] = 1;  
rtk_qos_priMap_set(4, &pri2qid);
```

---

***rtk\_api\_ret\_t rtk\_qos\_1pRemarkEnable\_set(rtk\_port\_t port, rtk\_enable\_t enable)***

This API is used to set per port 802.1P remarking ability for the specified port.

Example:

```
/*Enable 802.1P remarking for port 2*/  
rtl8306e_qos_1pRemarkEnable_set(2, ENABLED);
```

---

***rtk\_api\_ret\_t rtk\_qos\_1pRemark\_set(rtk\_pri\_t int\_pri, rtk\_pri\_t dot1p\_pri)***

This API is used to configure 802.1P remarking table, which means mapping the 2-bit internal priority to 3-bit priority.

Example:

```
/*set the remarking priority of internal priority 0 to be 3*/
rtk_pri_t int_pri;
rtk_pri_t dot1p_pri ;

int_pri = 0 ;
dot1p_pri = 3 ;
rtk_qos_1pRemark_set (int_pri, dot1p_pri);
```

---

***rtk\_api\_ret\_t rtk\_rate\_igrBandwidthCtrlRate\_set( rtk\_port\_t port, rtk\_rate\_t rate, rtk\_enable\_t ifg\_include)***

This API is used to configure input bandwidth control rate for port. The *rate* unit is 64Kbps and the range is from 64Kbps to 100Mbps. The granularity of rate is 64Kbps. The *ifg\_include* parameter is used to determine rate calculation include/exclude inter-frame-gap and preamble, and its value could be following macros:

ENABLED - include inter-frame-gap and preamble;  
DISABLED - exclude inter-frame-gap and preamble

Example:

```
/*Enable port1 input rate limitation and the rate is 4Mbps(62x64Kbps),*/
/*set the rate's calculation includes IFG and preamble */
rtk_rate_igrBandwidthCtrlRate_set(1, 62, ENABLED);
```

---

***rtk\_api\_ret\_t rtk\_rtk\_rate\_egrBandwidthCtrlRate\_set(rtk\_port\_t port, rtk\_rate\_t rate, rtk\_enable\_t ifg\_include)***

This API is used to configure output bandwidth control rate for port. The *rate* unit is 64Kbps and the range is from 64Kbps to 100Mbps. The granularity of rate is 64Kbps. The *ifg\_include* parameter is used for rate calculation with/without inter-frame-gap and preamble.

Example:

```
/*Enable port4 output rate limitation and the rate is 50Mbps(763x64Kbps),*/
/*set the rate's calculation not include IFG and preamble */
rtk_rate_egrBandwidthCtrlRate_set(4, 763, DISABLED);
```

## 4.6 CPU port

RTL8306E provides CPU port design for switch management. There is no limitation which port is CPU port and user can append any port to CPU communication as desired. ASIC will insert proprietary tag with Ethernet Length/Type 0x8899 to forward frame to CPU port as related configuration. The proprietary tag specification is as below table.

| 15                            | 8                | 7                               |                  | 0             |
|-------------------------------|------------------|---------------------------------|------------------|---------------|
| Realtek EtherType(2) [0x8899] |                  |                                 |                  |               |
| Protocol (4-bit) [0x9]        | Priority (2-bit) | Disable/Enable Learning (1-bit) | Reserved (3-bit) | Tx/Rx (6-bit) |

Fig. 4-4 CPU tag format

The proprietary tag will be parsed from receiving frame from CPU port and be inserted to forward frame to CPU port only. TX/RX field of CPU tag in forwarding frame is indicated which source port that forwarding frame comes from and CPU software can use this one to apply some useful high layer applications. TX/RX field in receiving frame from CPU port is used to force layer 2 forwarding decision and ASIC will use this field as forwarding port mask with desired priority assign. Frames with CPU tag which disable learning field enabled will let the ASIC bypass source MAC learning function.

Here are explanations and example codes for CPU port API. The example codes ignore the error check

### ***rtk\_api\_ret\_t rtk\_cpu\_enable\_set(rtk\_enable\_t enable)***

The API can set CPU port function enable/disable, default port 4 is CPU port. User can modify CPU port setting by calling API ***rtk\_cpu\_tagPort\_set***.

Example:

```
/*Enable CPU port function*/
rtk_cpu_enable_set (ENABLED);
```

### ***rtk\_api\_ret\_t rtk\_cpu\_tagPort\_set(rtk\_port\_t port, rtk\_cpu\_insert\_t mode)***

The API can set CPU port and select the inserting proprietary CPU tag mode for the frame that is transmitted to CPU port.

```
typedef enum rtk_cpu_insert_e
{
    CPU_INSERT_TO_ALL = 0,
    CPU_INSERT_TO_TRAPPING,
    CPU_INSERT_TO_NONE,
    CPU_INSERT_END
}rtk_cpu_insert_t;
```

Example:

```
/*Set port 3 as CPU port and insert CPU tag only to packets which are trapped to CPU*/
```

```
rtk_cpu_insert_t mode;  
mode = CPU_INSERT_TO_TRAPPING;  
rtk_cpu_enable_set(ENABLED);  
rtk_cpu_tagPort_set(3, mode);
```

---

***rtk\_api\_ret\_t rtk\_cpu\_tagPort\_get(rtk\_cpu\_insert\_t \*pMode)***

User could use this API to retrieve current CPU port setting including current CPU port id and inserting tag mode setting.

Example:

```
/* Get current CPU setting */  
rtk_port_t port;  
rtk_cpu_insert_t mode;  
rtk_cpu_tagPort_get(&port, &mode);
```

## 4.7 MIB

The RTL8306E implements five 32-bit MIB Counters on each port for traffic management and diagnostic purposes. The five MIB counters are Tx Counter, Rx Counter, Rx Drop Counter, Rx CRC Counter, and Rx Fragment Counter. The Tx Counter and Rx Counter can be byte-based or packet-based for each port. There is also an on-off register for each port to enable or disable/clear MIB Counters. Pause frame on/off is not counted in any condition.

Here are explanations and example codes for MIB API. The example codes ignore the error check.

---

***rtk\_api\_ret\_t rtk\_mib\_get(rtk\_port\_t port, rtk\_mib\_counter\_t counter, rtk\_mib\_cntValue\_t \*pValue)***

This API is used to get the MIB counter value for the specified port and MIB counter type. The *counter* parameter could be set following macros:

MIB\_CNT1 - TX count  
MIB\_CNT2 - RX count  
MIB\_CNT3 - RX Drop Count  
MIB\_CNT4 - RX CRC error Count  
MIB\_CNT5 - RX Fragment Count

Example:

```
/* Get RX counter of port 0 */  
rtk_mib_cntValue_t cntVal;  
rtk_mib_get(0, MIB_CNT2, &cntVal);
```

---

***rtk\_api\_ret\_t rtk\_mib\_cntType\_set(rtk\_port\_t port, rtk\_mib\_counter\_t counter, rtk\_mib\_cntType\_t type)***

This API is used to set RX/Tx Mib counting type: byte or packet. The *type* parameter could be selected from following macros:

MIB\_TYPE\_PKT - packet  
MIB\_TYPE\_BYTE - byte

And Only TX count and RX count could choose counting type, while the other counters' counting uint is always packet.

Example:

```
/* Set RX counter's counting type as packet for port 1 */  
rtk_mib_cntType_set (1, MIB_CNT2, MIB_TYPE_PKT);  
  
/* Set TX counter's countingtype as byte for port 3 */  
rtk_mib_cntType_set (3, MIB_CNT1, MIB_TYPE_BYTE);
```

***rtk\_api\_ret\_t rtk\_mib\_cntType\_get(rtk\_port\_t port, rtk\_mib\_counter\_t counter, rtk\_mib\_cntType\_t \*pType)***

This API is used to get RX/Tx Mib counting type: byte or packet. The *pType* parameter could be following values:

MIB\_TYPE\_PKT - packet

MIB\_TYPE\_BYTE - byte

Only TX count and RX count could choose counting type, while the other counters' counting uint is always packet.

Example:

```
rtk_mib_cntType_t type;
/* Set RX counter's counting type as packet for port 1*/
rtk_mib_cntType_get (1, MIB_CNT2, & type);

/* Set TX counter's countingtype as byte for port 3*/
rtk_mib_cntType_get (3, MIB_CNT1, & type);
```

---

***rtk\_api\_ret\_t rtk\_mib\_reset(rtk\_port\_t port)***

This API is used to reset MIB counter for the specified port. The corresponding MIB counter will stop counting, clear self to 0, and then start counting again.

Example:

```
/*Reset MIB counters of port 4*/
rtk_mib_reset t(4);
```

---

***rtk\_api\_ret\_t rtk\_stat\_port\_reset(rtk\_port\_t port)***

This API is used to reset MIB counter for the specified port. The corresponding MIB counter will stop counting, clear self to 0, and then start counting again.

Example:

```
/*Reset MIB counters of port 4*/
rtk_stat_port_reset(4);
```

## 4.8 Phy

RTL8306E supports 5-port 10/100M PHYs and allows network management to configure desired capability. Auto-Negotiation function allows a device to advertise modes (100F, 100H, 10F and 10H modes) of operation it possesses to a remote end of a link segment and detect corresponding operational modes. Without Auto-Negotiation function, PHY could be forced as 10/100 modes (100F, 100H, 10F and 10H modes).

Here are explanations and example codes for phy API. The example codes ignore the error check.

---

### ***rtk\_api\_ret\_t rtk\_port\_phyAutoNegoAbility\_set(rtk\_port\_t port, rtk\_port\_phy\_ability\_t \*pAbility)***

The structure of *rtk\_port\_phy\_ability\_t* is defined as following:

```
typedef struct rtk_port_phy_ability_s
{
    uint32    AutoNegotiation;    /*PHY register 0.12 setting for auto-negotiation process*/
    uint32    Half_10;            /*PHY register 4.5 setting for 10BASE-TX half duplex capable*/
    uint32    Full_10;            /*PHY register 4.6 setting for 10BASE-TX full duplex capable*/
    uint32    Half_100;           /*PHY register 4.7 setting for 100BASE-TX half duplex capable*/
    uint32    Full_100;           /*PHY register 4.8 setting for 100BASE-TX full duplex capable*/
    uint32    Full_1000;          /*PHY register 9.9 setting for 1000BASE-T full duplex capable*/
    uint32    FC;                 /*PHY register 4.10 setting for flow control capability*/
    uint32    AsyFC;              /*PHY register 4.11 setting for asymmetric flow control capability*/
} rtk_port_phy_ability_t;
```

This API enables PHY auto-negotiation and configures advertisement ability. RTL8306E switch only has 5 PHYs, so only 0~4 are permitted for input parameter *port*.

Example:

```
/*Set PHY 1 with Auto negotiation, 10F, and enable Symmetric PAUSE flow control capability*/
rtk_port_phy_ability_t  ability;
memset(&ability, 0, sizeof(ability));
ability.Full_10 = 1;
ability.FC = 1;
rtk_port_phyAutoNegoAbility_set(4, &ability);

/*set PHY 2 with Auto negotiation, 100F without flow control*/
rtk_port_phy_ability_t  ability;
memset(&ability, 0, sizeof(ability));
ability.Full_100 = 1;
rtk_port_phyAutoNegoAbility_set(2, &ability);
```

---

***rtk\_api\_ret\_t rtk\_port\_phyAutoNegoAbility\_get(rtk\_port\_t port, rtk\_port\_phy\_ability\_t \*pAbility)***

Get the capability of specified PHY. RTL8306E switch only has 5 PHYs, so only 0~4 are permitted for input parameter *port*

Example:

```
/*Get capability of PHY 1 */
rtk_port_phy_ability_t  ability;
rtk_port_phyAutoNegoAbility_get (1, &ability);
```

---

***rtk\_api\_ret\_t rtk\_port\_phyForceModeAbility\_set (rtk\_port\_t port, rtk\_port\_phy\_ability\_t \*pAbility)***

The API forces PHY as specified ability. RTL8306E switch only has 5 PHYs, so only 0~4 are permitted for input parameter *port*.

Example:

```
/*Force PHY 1 to 10HALF, and enable Symmetric PAUSE flow control capabilities*/
rtk_port_phy_ability_t  ability;
memset(&ability, 0, sizeof(ability));
ability.Half_10 = 1;
ability.FC = 1;
rtk_port_phyForceModeAbility_set (1, &ability);

/*Force PHY4 10FULL without flow control*/
rtk_port_phy_ability_t  ability;
memset(&ability, 0, sizeof(ability));
ability.Full_10 = 1;
rtk_port_phyForceModeAbility_set (4, &ability);
```

---

***rtk\_api\_ret\_t rtk\_port\_phyStatus\_get(rtk\_port\_t port, rtk\_port\_linkStatus\_t \*pLinkStatus, rtk\_port\_speed\_t \*pSpeed, rtk\_port\_duplex\_t \*pDuplex)***

Get current PHY link status.

```
typedef enum rtk_port_linkStatus_e
{
    PORT_LINKDOWN = 0,
    PORT_LINKUP,
    PORT_LINKSTATUS_END
} rtk_port_linkStatus_t;

typedef enum rtk_port_speed_e
{
    PORT_SPEED_10M = 0,
```



```
    PORT_SPEED_100M,  
    PORT_SPEED_1000M,  
    PORT_SPEED_END  
} rtk_port_speed_t;  
typedef enum rtk_port_duplex_e  
{  
    PORT_HALF_DUPLEX = 0,  
    PORT_FULL_DUPLEX,  
    PORT_DUPLEX_END  
} rtk_port_duplex_t;
```

Example:

```
/*Get link status of PHY 1 */  
rtk_port_linkStatus_t linkStatus;  
rtk_port_speed_t speed;  
rtk_port_duplex_t duplex;  
  
rtk_port_phyStatus_get(1, &linkStatus, &speed, &duplex);
```

## 4.9 Force External Interface

The RTL8306E supports dual MII/RMII interface for external devices to connect to the 5<sup>th</sup> and 6<sup>th</sup> MAC (MAC4 and MAC5), which is mapped to port4 and port5 and should work in force mode. When MAC 4 and MAC 5 are needed in system applications, the operating abilities could be configured through API *rtk\_port\_macForceLinkExt0\_set* and *rtk\_port\_macForceLinkExt1\_set* after setting the correct DUALMII/RMII operating mode by hardware strapping pin.

---

***rtk\_api\_ret\_t rtk\_port\_macForceLinkExt0\_set(rtk\_mode\_ext\_t mode,  
rtk\_port\_mac\_ability\_t \*pPortability)***

```
typedef struct rtk_port_mac_ability_s
{
    uint32 forcemode;
    uint32 speed;
    uint32 duplex;
    uint32 link;
    uint32 nway;
    uint32 symflc;
    uint32 txpause;
    uint32 rxpause;
}rtk_port_mac_ability_t;
```

This API is used to configure port5(MAC5) force mode properties, including link status, speed, duplex, and tx pause and tx pause ability. In force mode, the filed *forcemode* is always set to 1, and filed *nway* is always set to 0. *symflc* is set to 1 to enable symmetric flow control ability. When *symflc* is set to 0, the asymmetric flow control ability is enabled and is determined by *txpause* and *rxpause*.

For 8306E, MAC5 operating mode can be following macros:

```
MODE_EXT_MII_MAC - MAC Mode MII
MODE_EXT_MII_PHY - PHY Mode MII
MODE_EXT_TMII_MAC - MAC Mode TMII
MODE_EXT_TMII_PHY - PHY Mode TMII
MODE_EXT_RMII - RMII
```

So the external interface 1(MAC 5) operating mode can be MAC Mode (T)MII, PHY Mode (T)MII and RMII. But the operating mode is determined by strapping pin upon reset, and can not be configured by software, except the selection of MII or TMII. The input parameter *mode* is only used to enable/disable TMII for MAC 5.

Example:

```
/*Suppose port 5 was configured as MAC Mode MII interface by hardware strapping pin,*/
/*now user wants to enable TMII, so parameter mode needs to be set MODE_EXT_TMII_MAC.*/
/*set operating abilities as 100FULL, enable asymmetric tx/rx pause ability, force link up*/
rtk_mode_ext_t mode;
rtk_port_mac_ability_t portAbility;
```

```
mode = MODE_EXT_TMII_MAC;
portAbility.forcemode = 1;
portAbility.nway = 0;
portAbility.symflc = 0;
portAbility.speed = PORT_SPEED_100M;
portAbility.link = 1;
portAbility.duplex = PORT_FULL_DUPLEX;
portAbility.rxpause = 1;
portAbility.txpause = 1;
```

```
rtk_port_macForceLinkExt1_set(mode, &portAbility);
```

```
/*suppose port 5 was configured as RMII interface by hardware strapping pin, */
/*so user can not change its operating mode that means parameter mode is meaningless,*/
/*but can set operating abilities all the same. Set port 5 as 10HALF, enable tx pause, */
/*disable rx pause, force link up.*/
rtk_mode_ext_t mode;
rtk_port_mac_ability_t portAbility;
```

```
mode = MODE_EXT_RMII;
portAbility.forcemode = 1;
portAbility.nway = 0;
portAbility.symflc = 0;
portAbility.speed = PORT_SPEED_10M;
portAbility.link = 1;
portAbility.duplex = PORT_HALF_DUPLEX;
portAbility.rxpause = 0;
portAbility.txpause = 1;
```

```
rtk_port_macForceLinkExt0_set(mode, &portAbility);
```

---

***rtk\_api\_ret\_t rtk\_port\_macForceLinkExt0\_get(rtk\_mode\_ext\_t \*pMode,  
rtk\_port\_mac\_ability\_t \*pPortability)***

This API is used to get port5(MAC5) force mode properties, including operating mode and abilities like link status, speed, duplex, rx pause and tx pause ability. In force mode, auto Negotiation ability is always disabled.

Example:

```
rtk_mode_ext_t mode;
rtk_port_mac_ability_t portAbility;

rtk_port_macForceLinkExt1_get(&mode, &portAbility);
```

---

***rtk\_api\_ret\_t rtk\_port\_macForceLinkExt\_set(rtk\_port\_t port, rtk\_mode\_ext\_t mode, rtk\_port\_mac\_ability\_t \*pPortability)***

```
typedef struct rtk_port_mac_ability_s
{
    uint32 forcemode;
    uint32 speed;
    uint32 duplex;
    uint32 link;
    uint32 nway;
    uint32 symflc;
    uint32 txpause;
    uint32 rxpause;
}rtk_port_mac_ability_t;
```

This API is used to configure port4(MAC4) and port5(MAC5) force mode properties, including link status, speed, duplex, and tx pause and tx pause ability. In force mode, the field *forcemode* is always set to 1, and filed *nway* is always set to 0. *symflc* is set to 1 to enable symmetric flow control ability. When *symflc* is set to 0, the asymmetric flow control ability is enabled and is determined by *txpause* and *rxpause*. For MAC 4, the flow control ability is disabled when *symflc* is set to 0.

For 8306E, MAC 4 operating mode can be following macros:

MODE\_EXT\_MII\_MAC - MAC Mode MII  
MODE\_EXT\_MII\_PHY - PHY Mode MII  
MODE\_EXT\_RMII – RMII

MAC 5 operating mode can be following macros:

MODE\_EXT\_MII\_MAC - MAC Mode MII  
MODE\_EXT\_MII\_PHY - PHY Mode MII  
MODE\_EXT\_TMII\_MAC - MAC Mode TMII  
MODE\_EXT\_TMII\_PHY – PHY Mode TMII  
MODE\_EXT\_RMII - RMII

For MAC4, the operating mode can not be set by software and is determined by strapping pin upon reset. For MAC5, the operating mode is determined by strapping pin upon reset, and can not be configured by software, except the selection of MII or TMII. The input parameter *mode* is only used to enable/disable TMII for MAC 5.

Example:

```
/*Suppose port 4 was configured as MAC Mode MII interface by hardware strapping pin,*/  
/*set operating abilities as 100FULL, enable symmetric pause ability, force link up*/  
rtk_port_t port ;  
rtk_mode_ext_t mode;  
rtk_port_mac_ability_t portAbility;  
  
port = RTL8306E_PORT4
```

```
mode = MODE_EXT_MII_MAC;
portAbility.forcemode = 1;
portAbility.nway = 0;
portAbility.symflc = 1;
portAbility.speed = PORT_SPEED_100M;
portAbility.link = 1;
portAbility.duplex = PORT_FULL_DUPLEX;
portAbility.rxpause = 1;
portAbility.txpause = 1;

rtk_port_macForceLinkExt_set(RTL8306E_PORT4, mode, &portAbility);
```

```
/*suppose port 5 was configured as RMII interface by hardware strapping pin, */
/*so user can not change its operating mode that means parameter mode is meaningless,*/
/*but can set operating abilities all the same. Set port 5 as 10HALF, enable tx pause, */
/*disable rx pause, force link up.*/
Rtk_port_t port;
rtk_mode_ext_t mode;
rtk_port_mac_ability_t portAbility;
```

```
port = RTL8306E_PORT5;
mode = MODE_EXT_RMII;
portAbility.forcemode = 1;
portAbility.nway = 0;
portAbility.symflc = 0;
portAbility.speed = PORT_SPEED_10M;
portAbility.link = 1;
portAbility.duplex = PORT_HALF_DUPLEX;
portAbility.rxpause = 0;
portAbility.txpause = 1;
```

```
rtk_port_macForceLinkExt_set(port, mode, &portAbility);
```

---

***rtk\_api\_ret\_t rtk\_port\_macForceLinkExt\_get(rtk\_port\_t port, rtk\_mode\_ext\_t \*pMode, rtk\_port\_mac\_ability\_t \*pPortability)***

This API is used to get port4 (MAC4) and port5 (MAC5) force mode properties, including operating mode and abilities like link status, speed, duplex, rx pause and tx pause ability. In force mode, auto Negotiation ability is always disabled.

Example:

```
rtk_port_t port;
rtk_mode_ext_t mode;
rtk_port_mac_ability_t portAbility;
```

---

```
port = RTL8306E_PORT4;  
rtk_port_macForceLinkExt_get(port , &mode, &portAbility);
```

---

### ***rtk\_port\_mii1Disable\_set(void)***

This API is used to force MII 1 link down. It can only be called for RTL8304E, and not for the other types of ASIC. When you want to use MII 2 and disable MII 1 for RTL8304E, this API can be called. This API is only compiled when macro CHIP\_RTL8304E is defined in “rtl8306e\_asictypes.h”.

Example:

```
/*suppose MII 2 was configured as MII MAC interface by hardware strapping pin, */  
/*so user can choose MII 2 to be TMII or MII MAC mode. Suppose user set MII 2 as TMII MAC  
/*mode but can set operating abilities all the same. Set MII 2 as 100FULL, enable tx pause, */  
/*disable rx pause, force link up.*/  
/*disable MII 1*/  
Rtk_port_t port;  
rtk_mode_ext_t mode;  
rtk_port_mac_ability_t portAbility;  
  
port = RTL8306E_PORT5;  
mode = MODE_EXT_TMII_MAC;  
portAbility.forcemode = 1;  
portAbility.nway = 0;  
portAbility.symflc = 0;  
portAbility.speed = PORT_SPEED_100M;  
portAbility.link = 1;  
portAbility.duplex = PORT_FULL_DUPLEX;  
portAbility.rxpause = 0;  
portAbility.txpause = 1;  
  
rtk_port_macForceLinkExt_set(port, mode, &portAbility);  
rtk_port_mii1Disable_set();
```

## 4.10 Dot1X

RTL8306E Supports IEEE 802.1X function, it includes port-based and MAC-based authentication. The following APIs are provided to configure IEEE 802.1x function.

---

***rtk\_api\_ret\_t rtk\_dot1x\_unauthPacketOper\_set(rtk\_port\_t port, rtk\_dot1x\_unauth\_action\_t unauth\_action)***

The API could set the unauthorized packet action. For RTL8306E switch, the action is for whole system, so *port* could be any value of 0~5. There are three options: dropping and trapping to CPU, dropping is the default setting.

Example:

```
/* drop unauthorized packets */
rtk_dot1x_unauthPacketOper_set(1, DOT1X_ACTION_DROP);

/* trap unauthorized packets to CPU */
rtk_dot1x_unauthPacketOper_set(0, DOT1X_ACTION_TRAP2CPU);
```

---

***rtk\_api\_ret\_t rtk\_dot1x\_portBasedEnable\_set(rtk\_port\_t port, rtk\_enable\_t enable)***

The API is used to enable or disable IEEE802.1X function by each port. When the port is enabled the function, its authentication status will determine forwarding behavior of the port.

Example:

```
/* Enable port 0 port-based IEEE 802.1x function*/
rtk_dot1x_portBasedEnable_set(0, ENABLED);

/* Disable port 1 port-based IEEE 802.1x function*/
rtk_dot1x_portBasedEnable_set(1, DISABLED);
```

---

***rtk\_api\_ret\_t rtk\_dot1x\_portBasedAuthStatus\_set(rtk\_port\_t port, rtk\_dot1x\_auth\_status\_t port\_auth)***

The API is used to set port authentication status.

Example:

```
/* Enable port 3, 4 port-based IEEE 802.1x function, port 3 unauthorized, port 4 authorized */
rtk_dot1x_portBasedEnable_set(3, ENABLED);
rtk_dot1x_portBasedEnable_set(4, ENABLED);
rtk_dot1x_portBasedAuthStatus_set(3, UNAUTH);
rtk_dot1x_portBasedAuthStatus_set(4, AUTH);
```

---

***rtk\_api\_ret\_t rtk\_dot1x\_portBasedDirection\_set(rtk\_port\_t port, rtk\_dot1x\_direction\_t***

---

***port\_direction)***

The API is used to set port-based operation direction, the direction could be both or in. For a packet, both direction means not only ingress port but also egress port authentication status should be checked, and when one of them is unauthorized port, the packet will be regarded as unauthorized packet. In direction means only ingress port authentication status should be checked, and a packet could be forwarded from authorized port to unauthorized port.

Example:

```
/* port 3 both direction, port 4 in direction*/  
rtk_dot1x_portBasedDirection_set (3, BOTH);  
rtk_dot1x_portBasedDirection_set (4, IN);
```

---

***rtk\_api\_ret\_t rtk\_dot1x\_macBasedEnable\_set(rtk\_port\_t port, rtk\_enable\_t enable)***

RTL8306E supports mac-based 802.1x function. MAC address authentication status of packet will be checked, and the MAC address authentication status is stored in ASIC MAC address table. This API is used to enable/disable mac-based 802.1x function.

Example:

```
/* Enable port 4 mac-based 802.1x function*/  
rtk_dot1x_macBasedEnable_set (4, ENABLED);
```

---

***rtk\_api\_ret\_t rtk\_dot1x\_macBasedDirection\_set(rtk\_dot1x\_direction\_t mac\_direction)***

The API is used to set mac-based 802.1x operation direction, which is similar to port-based operation direction, including both and in direction. Both direction means both SA and DA authentication status should be checked, in direction means only SA check is required.

Example:

```
/* Set mac-based 802.1x operation direction as both*/  
rtk_dot1x_macBasedDirection_set (BOTH);  
  
/* Set mac-based 802.1x operation direction as in*/  
rtk_dot1x_macBasedDirection_set (IN);
```

---

***rtk\_api\_ret\_t rtk\_dot1x\_macBasedAuthMac\_add(rtk\_port\_t port, rtk\_mac\_t \*pAuth\_mac, rtk\_fid\_t fid)***

The API is used to set a MAC address to be authorized status, and parameter *port* is physical port the MAC address belongs to. The *fid* is useless for RTL8306E, and can be any value. Before calling this API, the MAC address must exist in the address table, otherwise the API will return “the L2 entry not found”.

Example:

```
rtk_port_t port;  
rtk_mac_t auth_mac;
```



```
rtk_fid_t fid;
rtk_l2_ucastAddr_t l2_data;

/* Set mac-based 802.1x operation direction as in*/
rtk_dot1x_macBasedDirection_set (IN);

/* Enable port 5 mac-based 802.1x function*/
port = 5;
rtk_dot1x_macBasedEnable_set (port, ENABLED);

/*add authorized mac 00:12:34:56:78:9a */
memset(&l2_data, 0, sizeof(l2_data));
auth_mac.octet[0] = 0x0;
auth_mac.octet[1] = 0x12;
auth_mac.octet[2] = 0x34;
auth_mac.octet[3] = 0x56;
auth_mac.octet[4] = 0x78;
auth_mac.octet[5] = 0x9a;
fid = 0x2;
l2_data.auth = 1;
l2_data.fid = fid;
l2_data.is_static = 0;
l2_data.port = port;

rtk_l2_addr_add(&auth_mac, &l2_data);
rtk_dot1x_macBasedAuthMac_add(port, &auth_mac, fid);
```

---

***rtk\_api\_ret\_t rtk\_dot1x\_macBasedAuthMac\_del(rtk\_port\_t port, rtk\_mac\_t \*pAuth\_mac, rtk\_fid\_t fid)***

The API is used to delete a 802.1x authenticated MAC address from port. It only changes the auth status of the MAC and won't delete it from MAC address table. Parameter *port* is physical port the MAC address belongs to. The *fid* useless for RTL8306E, and can be any value. Before calling this API, the MAC address must exist in the address table, otherwise the API will return “the L2 entry not found”.

Example:

```
/* Suppose the mac 00:12:34:56:78:9a has been added and authorized as in the example code*/
/* of API rtk_dot1x_macBasedAuthMac_add, now user can change its auth status.*/
rtk_port_t port;
rtk_mac_t auth_mac;
rtk_fid_t fid;

port = 5;
fid = 0x2;
auth_mac.octet[0] = 0x0;
```

```
auth_mac.octet[1] = 0x12;  
auth_mac.octet[2] = 0x34;  
auth_mac.octet[3] = 0x56;  
auth_mac.octet[4] = 0x78;  
auth_mac.octet[5] = 0x9a;  
  
rtk_dot1x_macBasedAuthMac_del(port, &auth_mac, fid);
```

## 4.11 Port mirror

RTL8306E supports one set of mirroring functions for network monitoring. The monitor port is called mirroring port, the port whose traffic is to be mirrored is called mirrored port. The mirroring port can mirror both the TX and RX packets of the mirrored port. Only one port can be set as mirroring port.

Here are explanations and example codes for port Mirror API. The example codes ignore the error check.

---

```
rtk_api_ret_t rtk_mirror_portBased_set(rtk_port_t mirroring_port, rtk_portmask_t  
*pMirrored_rx_portmask, rtk_portmask_t *pMirrored_tx_portmask)
```

Example:

```
/*set port 4 to mirror transmitting frames of port 0 and receiving frames of port 1 */  
rtk_portmask_t rx_portmask;  
rtk_portmask_t tx_portmask;  
  
tx_portmask.bits[0] = 0x1;  
rx_portmask.bits[0] = 0x2;  
rtk_mirror_portBased_set(4, &rx_portmask, &tx_portmask);
```

## 4.12 Port Isolation

RTL8306E supports Port Isolation function. Users can use API ***rtk\_port\_isolation\_set*** to set a permitted forwarding port mask for each source port. The permitted forwarding port mask is configured per port. All the packets switched by RTL8306E can't be forwarded to the ports which are not in the rx port's permitted forwarding port mask.

CPU force TX (TX portmask in CPU tag) function doesn't be affected by Port Isolation. Packets which coming from CPU port and have "TX portmask in CPU tag can be forwarding to any other ports. Except this, Port Isolation is the highest priority in forwarding decision.

Here are explanations and example codes for Port Isolation API. The example codes ignore the error check.

---

### ***rtk\_api\_ret\_t rtk\_port\_isolation\_set(rtk\_port\_t port, rtk\_portmask\_t portmask)***

This API can be used to set the permitted port mask that the specified port can transmit packets to. A port can only transmit packets to ports included in permitted forwarding *portmask*.

Example:

```
/* Set port Isolation function on Port 3 and set its permitted forwarding port mask to port 0~2*/  
rtk_portmask_t portmask;  
portmask.bits[0] = 0x07;  
rtk_port_isolation_set(3, portmask);
```

---

### ***rtk\_api\_ret\_t rtk\_port\_isolation\_get(rtk\_port\_t port, rtk\_portmask\_t \*pPortmask)***

This API can be used to get the permitted port mask that the specified port can transmit packets to. A port can only transmit packets to ports included in permitted portmask.

Example:

```
/*Get port isolation configuration on port 5 */  
rtk_portmask_t portmask;  
rtk_port_isolation_get(5, &portmask);
```

## 4.13 MAC Learning Limit

RTL8306E supports source MAC learning limit function. Each port could choose to limit or un-limit the source MAC learning individually via the API ***rtk\_l2\_limitLearningEn\_set***. Port which has have the MAC limitation function enabled and learned too many MACs could be configured to drop or trap the packet with new source MAC to CPU by calling the API ***rtk\_l2\_limitLearningCntAction\_set***. ***rtk\_l2\_learningCnt\_get*** is used to retrieve the number of MACs currently learned on the specified port.

---

### ***rtk\_api\_ret\_t rtk\_l2\_limitLearningCnt\_set(rtk\_port\_t port, rtk\_mac\_cnt\_t mac\_cnt)***

This API can be used to set per-port auto learning MAC limit number from 0 to 31. If *mac\_cnt* is set from 0 to 31, per-port auto learning MAC limit will be enabled, and if *mac\_cnt* is set 0xFF, per-port ASIC auto learning limit will be disabled.

Example:

```
/*Enable MAC learning limit on port 2, and set MAC learning limit to 20 MAC addresses*/  
rtk_l2_limitLearningCnt_set(2, 20);  
  
/*Disable MAC learning limit on Port 2*/  
rtk_l2_limitLearningCnt_set(2, 0xFF);
```

---

### ***rtk\_api\_ret\_t rtk\_l2\_limitLearningCntAction\_set(rtk\_port\_t port, rtk\_l2\_limitLearnCntAction\_t action)***

The API can set the action for new source MAC packet when the MAC limit of a port has been reached. The action is global, so the parameter *port* must be set as *RTK\_WHOLE\_SYSTEM*.

Example:

```
/* set when the MAC limit of a port has been reached, trap the packet with the new source MAC to CPU*/  
rtk_l2_limitLearningCntAction_set(RTK_WHOLE_SYSTEM, LIMIT_LEARN_CNT_ACTION_TO_CPU );  
  
/* set when the MAC limit of a port has been reached, drop the packet with the new source MAC */  
rtk_l2_limitLearningCntAction_set(RTK_WHOLE_SYSTEM, LIMIT_LEARN_CNT_ACTION_DROP);
```

---

### ***rtk\_api\_ret\_t rtk\_l2\_learningCnt\_get(rtk\_port\_t port, rtk\_mac\_cnt\_t \*pMac\_cnt)***

The API can get per-port ASIC auto learned MAC address number.

Example:

```
/*Get current MAC learning counter on Port 2 */  
rtk_mac_cnt_t mac_cnt;  
  
rtk_l2_learningCnt_get (2, &mac_cnt);
```

---

## 4.14 ACL

The 8306E ACL holds 16 entries. When a packet is received, its source port, destination port (if a TCP or UDP packet), or EtherType code (if a non IP packet), are recorded and compared to current ACL entries. If they are matched, and if physical port and protocol are also matched, the entry is valid.

Here are explanations and example codes for ACL API. The example codes ignore the error check.

---

### ***rtk\_api\_ret\_t rtk\_filter\_igrAcl\_init(void)***

This API can initialize ACL, that means empty the ACL table.

Example:

```
/*initialize ACL module before any configuration*/  
rtk_filter_igrAcl_init ( );
```

---

### ***rtk\_api\_ret\_t rtk\_filter\_igrAcl\_rule\_add(rtk\_filter\_rule\_t \*pRule)***

This API is used to add an ACL rule into ACL table. The structure of *rtk\_filter\_rule\_t* is defined as following:

```
typedef struct rtk_filter_rule_e  
{  
    uint32 phyport;      /*the physical port where packet received*/  
    uint32 protocol;     /*Acl protocol which is in the packet*/  
    uint32 data;         /*16bit ether type or TCP/UDP source port, destination port value*/  
    uint32 priority;     /*Acl priority assigned to the packet*/  
    uint32 action;       /*How to deal with the packet*/  
} rtk_filter_rule_t;
```

The phyport could be 0~5: port number, RTL8306\_ACL\_ANYPORT: any port; protocol could be RTL8306\_ACL\_ETHER(ether type), RTL8306\_ACL\_TCP(TCP), RTL8306\_ACL\_UDP(UDP), RTL8306\_ACL\_TCPUDP(TCP or UDP); priority could be RTL8306\_PRIO0~RTL8306\_PRIO3; action could be RTL8306\_ACT\_DROP/RTL8306\_ACT\_PERMIT/RTL8306\_ACT\_TRAPCPU/RTL8306\_ACT\_MIRROR.

Example:

```
/*Add an ACL rule, phyport: port 4, protocol: ether type 0x8800(non IP packet),*/  
/* priority: 0x3, action: trap to CPU */  
rtk_filter_rule_t rule, rule_r;  
  
rule.phyport = RTL8306_PORT4;  
rule.protocol = RTL8306_ACL_ETHER;  
rule.data = 0x8800;
```

```
rule.priority = 0x3;  
rule.action = RTL8306_ACT_TRAPCPU;  
rtk_filter_igrAcl_rule_add (&rule);
```

```
/*Add an ACL rule for IP packet, phyport: port 0, TCP port: 0x5555, priority: 0x2, action: drop */  
rtk_filter_rule_t rule, rule_r;
```

```
rule.phyport = RTL8306_PORT0;  
rule.protocol = RTL8306_ACL_TCP;  
rule.data = 0x5555;  
rule.priority = 0x2;  
rule.action = RTL8306_ACT_DROP;  
rtk_filter_igrAcl_rule_add (&rule);
```

---

***rtk\_api\_ret\_t rtk\_filter\_igrAcl\_rule\_del(rtk\_filter\_rule\_t \*pRule)***

This API is used to delete an ACL rule from ACL table. Only phyport/protocol/data field in parameter *pRule* needs to be specified. If the ACL rule does not exist in the ACL table, it will return error.

Example:

```
/*Add an ACL rule for IP packet, phyport: port 0, TCP port: 0x5555, priority: 0x2, action: drop */  
rtk_filter_rule_t rule, rule_r;  
rule.phyport = RTL8306_PORT0;  
rule.protocol = RTL8306_ACL_TCP;  
rule.data = 0x5555;  
rule.priority = 0x2;  
rule.action = RTL8306_ACT_DROP;  
rtk_filter_igrAcl_rule_add (&rule);
```

```
/*Delete the ACL rule above */  
rtk_filter_rule_t rule, rule_r;  
rule.phyport = RTL8306_PORT0;  
rule.protocol = RTL8306_ACL_TCP;  
rule.data = 0x5555;
```

```
rtk_filter_igrAcl_rule_del(&rule);
```

## 4.15 Misc

### ***rtk\_api\_ret\_t rtk\_switch\_init(void)***

The API can set chip registers to default configuration.

---

### ***rtk\_api\_ret\_t rtk\_switch\_maxPktLen\_set(rtk\_switch\_maxPktLen\_t len)***

The API is used to set the max packet length. For 8306E, the parameter *len* could be set as MAXPKTLEN\_1522B, MAXPKTLEN\_1536B, MAXPKTLEN\_1552B and MAXPKTLEN\_2000B.

Example:

```
/*Set max packet length to 1536Bytes */  
rtk_switch_maxPktLen_set(MAXPKTLEN_1536B);
```

---

### ***rtk\_api\_ret\_t rtk\_trap\_unknownMcastPktAction\_set(rtk\_port\_t port, rtk\_mcast\_type\_t type, rtk\_trap\_mcast\_action\_t mcast\_action)***

The API is used to set the behavior of unknown multicast packet. When an unknown multicast packet is received, switch may trap, drop this packet. The behavior is global for all ports, so parameter *port* doesn't stand for actual physical port and can be any value from 0~5.

Example:

```
rtk_port_t port;  
rtk_mcast_type_t type;  
rtk_trap_mcast_action_t action;  
  
/*Drop unknown IPV4 multicast packet */  
port = 0;  
type = MCAST_IPV4;  
action = MCAST_ACTION_DROP;  
rtk_trap_unknownMcastPktAction_set(port, type, action);  
/*set unknown IPV6 multicast packet to be normal forwarded */  
port = 0;  
type = MCAST_IPV6;  
action = MCAST_ACTION_FORWARD;  
rtk_trap_unknownMcastPktAction_set(port, type, action);
```

---

### ***rtk\_api\_ret\_t rtk\_trap\_igmpCtrlPktAction\_set(rtk\_igmp\_type\_t type, rtk\_trap\_igmp\_action\_t igmp\_action)***

The API is used to set both IPv4 IGMP/IPv6 MLD packet with/without PPPoE header trapping function. All the 4 kinds of IGMP/MLD function can be set separately.

---



The igmp packet type is as following:

- IGMP\_IPV4
- IGMP\_MLD
- IGMP\_PPPOE\_IPV4
- IGMP\_PPPOE\_MLD

Example:

```
rtk_igmp_type_t type;  
rtk_trap_igmp_action_t action;  
  
/* Set the IPv6 IGMP packet without PPPoE header to be normal forwarded */  
type = IGMP_MLD;  
action = IGMP_ACTION_FORWARD;  
rtk_trap_igmpCtrlPktAction_set (type, action);  
  
/* Set the IPv4 IGMP packet with PPPoE header to be trapped to CPU*/  
type = IGMP_PPPOE_IPV4;  
action = IGMP_ACTION_TRAP2CPU;  
rtk_trap_igmpCtrlPktAction_set (type, action);
```

## 5 IGMP Snooping/MLD Snooping

### 5.1 General description

Two version codes are released this time. Folder multicastV1 contains the code of Version 1. These codes process IGMPV1/V2/V3 packets, and we just use the group information of IGMPV3. Folder multicastV2 contains the code of Version 2. Version 2 fully supports IGMPv1/v2/v3 and MLDV1/V2.

Usually, dumb switch broadcast multicast packets. This may create excessive traffic on the network and degrade the performance of hosts attached to the switches. Multicast routers have ability to segment networks and forward traffic only to actual destination interfaces so they have less trouble with rampant broadcast and multicast traffic. With Internet Group Management Protocol (IGMP) snooping, switch also can make intelligent multicast forwarding decisions by examining the contents of each frame's Layer 3 IP header.

IGMP snooping can make switch “listen in” on IGMP conversations between hosts and routers. A station that wishes to become a receiver sends an IGMP “group join” message to that group’s transmitter. When a switch hears a group join message from a host, it records the interface that it heard the message on and add it to the group. Similarly, when switch hears a group leave message or a response timer expires, the switch will remove that host’s switch interface from the group. There are three versions of IGMP for IPv4 and two versions of MLD (Multicast Listener Discovery) for IPv6:

**IGMPv1:** It defines the join message that hosts use to join an IP multicast group. With IGMPv1, routers must use a timer to determine which hosts are still members of the group. (RFC1112)

**IGMPv2:** It adds “group leave” messages to enable multicast member fast leave. (RFC2236)

**IGMPv3:** It adds support for “source filtering”, that is, the ability for a system to report interest in receiving packets only from specific source addresses, or from all but specific source addresses, sent to a particular multicast address. (RFC3376)

**MLDv1:** MLDv1 is derived from IGMPv2 and is used by IPv6 routers to discover the presence of multicast listeners on their directly attached links. (RFC2710)

**MLDv2:** MLDv2 is derived from IGMPv3. (RFC3810)

IP multicast traffic has some special characteristics. Destination IP addresses for multicast packets fall within the range of 224.0.0.1 to 239.255.255.255 (although some addresses within this range are reserved). Destination Ethernet addresses for multicast traffic begin with 01:00:5E and end with the low order 23 bits of the destination IP address.

## 5.2 IGMP Snooping Version 1

RTL8306E IGMP snooping consists of six files:

| File name                 | Description                                                                                                      |
|---------------------------|------------------------------------------------------------------------------------------------------------------|
| rtl_igmp_snooping.c       | The source code of IGMP snooping implementation.                                                                 |
| rtl_igmp_snooping.h       | Header file for IGMP snooping external functions, for user's usage.                                              |
| rtl_igmp_snooping_local.h | Header file for internal data structure or local functions.                                                      |
| rtl_igmp_types.h          | Defines basic data types used by rtl_igmp_snooping.c. User may modify this file to avoid data type redefinition. |
| rtl_igmp_glue.h           | Header file to glue different OS/hardware                                                                        |
| rtl_igmp_glue.c           | Glue functions implementation, user must modify this file when porting IGMP snooping to different platforms.     |

### 5.2.1 IGMP Snooping porting guide

#### 5.2.1.1 Modify the glue file: rtl\_igmp\_glue.c, rtl\_igmp\_glue.h

(1) Modify the glue functions:

```
int32 rtl_igmpGlueMutexLock(void)
int32 rtl_igmpGlueMutexUnlock(void)
void *rtl_igmpGlueMalloc(uint32 NBYTES)
void rtl_igmpGlueFree(void *APTR)
void rtl_igmpGlueNicSend(uint8 *pMacFrame, uint32 macFrameLen)
void rtl_multicastSnoopingV1TimeUpdate(void)
```

(2) Modify the macro definition in rtl\_igmp\_glue.h:

Define macro **CONFIG\_RTL8306E**.

Modify **MII\_PORT\_MASK** according to CPU connection situation.

Remove macro **IGMP\_DEBUG** and **RTL8651B\_SDK**.

If CPU has no multicast hardware forwarding ability, please remove **RTL\_CPU\_HW\_FWD** macro.

#### 5.2.1.2 Include corresponding header files: rtl\_igmp\_types.h, rtl\_igmp\_snooping.h

Example:

```
#include "rtl_igmp_types.h"
#include "rtl_igmp_snooping.h"
```

## 5.2.2 IGMP snooping function usage guide:

### (1) Initialization:

***int32 rtl\_initIgmpSnoopingV1(uint32 mCastDataToCpu, uint32 delPortMaskRevLeave, uint32 maxGroupNum, uint32 hashTableSize, uint32 currentTime)***

***rtl\_initIgmpSnoopingV1*** performs IGMP initialization. *mCastDataToCpu* determines whether or not to trap multicast data to CPU. *delPortMaskRevLeave* indicate the CPU deletes the multicast entry immediately when receiving a leave report. TRUE indicates CPU would delete the port mask when receiving a leave report while FALSE only to lower group member port timer. *maxGroupNum* specify the maximum group number to be supported in IGMP snooping module. *hashTableSize* specify hash table size. Hash table is used to find a group address in a fast way. This API also disable the CPU port learning function and set the CPU MAC into lookup table.

Example:

```
/*Initialize the IGMP module. The multicast data needn't trap to CPU. CPU will delete the
 * multicast entry when receiving a leave report. The maximum group number is 100 and the
 * hash table size is 32. Current system time for IGMP snooping time list is 0.
 */
rtl_initIgmpSnoopingV1(FALSE,TRUE, 100, 32, 0);
```

### (2) IGMP parameter configuration:

***int32 rtl\_setIgmpSnoopingV1Parameter(struct rtl\_igmpSnoopingParameter igmpSnoopingParas, uint8\* gatewayMac, uint32 gatewayIp)***

***rtl\_setIgmpSnoopingV1Parameter*** configures IGMP protocol parameters. *igmpSnoopingParas* is used to set relative timer of IGMP. If the member of *igmpSnoopingParas* has been set to 0, it uses the default value. CPU port learning function is disabled in ***rtl\_initIgmpSnoopingV1***. So user must call ***rtl\_setIgmpSnoopingV1Parameter*** to set gateway Mac into lookup table. Also, the unicast IGMP packet whose destination MAC equals to *gatewayMac* and destination IP equals to *gatewayIp* will be accepted.

Example:

```
/*set last member aging time to 10 seconds, others keep default value*/
uint8 macAddress[6] = {0x00, 0x0c, 0x29, 0x1a, 0xa0, 0xe7};
uint32 ipv4Add = 0xc0a801fe;
struct rtl_igmpSnoopingParameter igmpConfig;
igmpConfig.groupMemberAgingTime = 0;
igmpConfig.lastMemberAgingTime = 10;
igmpConfig.querierPresentInterval = 0;
igmpConfig.dvmrpRouterAgingTime = 0;
igmpConfig.mospfRouterAgingTime = 0;
igmpConfig.pimDmRouterAgingTime = 0;
```

```
rtl_setIgmpSnoopingV1Parameter(igmpConfig, macAddress, ipv4Add);
```

**(3) Maintain IGMP timer list:**

```
int32 rtl_maintainIgmpSnoopingV1TimerList(uint32 currentTime)
```

**rtl\_maintainIgmpSnoopingV1TimerList** maintains IGMP timer list. The input parameter *currentTime* means the current system time (unit: seconds).

**(4) Process input packet:**

```
void rtl_rtl_igmpSnoopingV1Receive(uint8 *pMacFrame , uint8 *pPort)
```

**rtl\_igmpSnoopingV1Receive** processes input MAC frame. *pMacFrame* points to the MAC frame received. If user wants to get source port information from the CPU tag or VLAN tag carried in the frame, he can set *pPort* to NULL. Otherwise, if user wants to assign a fixed source port to this IGMP frame, please assign a non NULL value to *pPort*.

Example:

```
/*If the frame has CPU tag, and user wants to get source port from CPU tag, set pPort to NULL*/  
uint8 *pPort = NULL;  
uint16 realtekEtherType = 0x8899;  
if (macFrame[12] == ((realtekEtherType & 0xff00) >> 8) && \  
    macFrame[13] == (realtekEtherType & 0x00ff) && \  
    macFrame[14] & 0xf0 == 0x90)  
{  
    pPort = NULL;  
    rtl_igmpSnoopingV1Receive(macFrame, pPort);  
}
```

**(5) Process output packet:**

```
int32 rtl_igmpSnoopingV1Send(uint8 *pMacFrame, uint32 priorFreeBytes, uint32  
    posteriorFreeSpace, uint8 **ppNewMacFrame )
```

Call **int32 rtl\_igmpSnoopingV1Send** to forward a multicast MAC frame. *pMacFrame* is the pointer of MAC frame to be transmitted. *priorFreeBytes* and *posteriorFreeSpace* indicate whether there is enough space for CPU tag inserting. *ppNewMacFrame* is the pointer of new MAC frame to be forwarded (maybe insert CPU tag or not).

Example:

```
uint8 *newMac = NULL;  
/*insert CPU tag with appropriate port mask*/  
rtl_igmpSnoopingV1Send(macFrame, 4, 0, &newMac);
```

```
/*call glue function to forward packet with CPU tagged*/  
rtl_igmpGlueNicSend(newMac, macFrameLength + 4);
```

## **(6) Disable IGMP snooping module**

***int32 rtl\_disableIgmpSnoopingV1(void)***

Call ***rtl\_disableIgmpSnoopingV1*** to disable IGMP snooping. It will free the memory for IGMP snooping module.

Note: step (5) is meaningful only when several multicast IP are mapped same MAC address, or RTL8306E lookup table's 4-way indexed entries are full. It can be ignored to achieve a better throughput, however, the sequence will be:

1. If packet's multicast address can't be written into lookup table because of 4-way full, it will be flooded. For example, G1: 224.1.2.3, if its MAC address(0x01:00:5e:01:02:03) hasn't been set into look up table, all the multicast data sent to G1 will be flooded.
2. If packet's multicast address is an aggregator address, it will be forwarded to all the aggregated group ports. For example, G1: 224.1.2.3, G2: 225.1.2.3, G3: 226.1.2.3. They have the same MAC address: 0x 01:00:5e:01:02:03. Any multicast data to G1 or G2 or G3, will be sent to all of them.

## **5.2.3 Advanced issues:**

If CPU has multicast acceleration hardware, porting IGMP snooping is a bit complicated:

- (1) Define a macro ***RTL\_CPU\_HW\_FWD*** in *rtl\_igmp\_glue.h*
- (2) Call ***rtl\_checkGroupStatus*** to check the feasibility of setting CPU hardware, If the group MAC address is not available in RTL8306E lookup table, or it's an aggregator address. It can't be set into CPU hardware.
- (3) Modify ***rtl\_igmpDisCpuHwFwd*** according to user's own system.( ***igmpDisCpuHwFwd*** is called by IGMP snooping module to prevent CPU from hardware forwarding a multicast group).
- (4) When in software forwarding state, call ***rtl\_checkGroupStatus*** to monitor group status and wait to switch back hardware forwarding state.

## 5.3 IGMP/MLD Snooping Version 2

Version 2 fully supports IGMP snooping/MLD snooping (multicast snooping). It consists of six files:

| File name                      | Description                                                                                                           |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| rtl_multicast_snooping.c       | The source code of multicast snooping implementation.                                                                 |
| rtl_multicast_snooping.h       | Header file for multicast snooping external functions, for user's usage.                                              |
| rtl_multicast_snooping_local.h | Header file for internal data structure or local functions.                                                           |
| rtl_multicast_types.h          | Defines basic data types used by rtl_multicast_snooping.c. User may modify this file to avoid data type redefinition. |
| rtl_multicast_glue.h           | Header file to glue different OS/hardware                                                                             |
| rtl_multicast_glue.c           | Glue functions implementation, user must modify this file when porting multicast snooping to different platform.      |
| rtl_multicast_glue.h           | Header file to glue different OS/hardware                                                                             |
| rtl_multicast_glue.c           | Glue functions implementation, user must modify this file when porting multicast snooping to different platform.      |

### 5.3.1 Multicast Snooping porting guide

#### 5.3.1.1 Modify the glue file: rtl\_multicast\_glue.c, rtl\_multicast\_glue.h

(1) Modify the glue functions:

```
int32 rtl_glueMutexLock(void)
int32 rtl_glueMutexUnlock(void)
void* rtl_glueMalloc(uint32 NBYTES)
void rtl_glueFree(void *APTR)
void rtl_glueNicSend(uint8 *macFrame, uint32 macFrameLen)
void rtl_multicastSnoopingV2TimeUpdate(void)
```

(2) Modify the macro definition in rtl\_multicast\_glue.h:

Define macro **CONFIG\_RTL8306E**.  
Modify **MII\_PORT\_MASK** according to CPU connection situation.

### 5.3.1.2 Include corresponding header files: `rtl_multicast_types.h`, `rtl_multicast_snooping.h`

Example:

```
#include "rtl_multicast_types.h"
#include "rtl_multicast_snooping.h"
```

### 5.3.2 Multicast snooping function usage guide:

#### (1) Initialization:

**`int32 rtl_initMulticastSnooping(struct rtl_multicastConfig multicastConfiguration, uint32 currentTime, uint32 delPortMaskRevLeave)`**

**`rtl_initMulticastSnooping`** performs multicast snooping initialization. *multicastConfiguration* Specifies Multicast snooping parameters. *currentTime* Specifies the current system time for multicast snooping timer list initialization. *delPortMaskRevLeave* indicate the CPU whether to delete the multicast entry immediately when receiving a leave report. TRUE indicates CPU would delete the port mask when receiving a leave report while FALSE only to lower group member port timer.

Example:

```
/*The interval of multicast router is 120s. Group membership interval is 260s. Last member query
 * interval is 10s. Query interval is 260s. The maximum group number is 195 and the hash table size
 * is 1000. Current system time for multicast snooping time list is 0. CPU acts as an IGMP proxy.
 */
struct rtl_multicastConfig multicastConfig;
multicastConfig.dvmrpRouterAgingTime = 120;
multicastConfig.groupMemberAgingTime = 260;
multicastConfig.lastMemberAgingTime = 10;
multicastConfig.mospfRouterAgingTime = 120;
multicastConfig.pimRouterAgingTime = 120;
multicastConfig.querierPresentInterval = 260;
multicastConfig.hashTableSize = 64;
multicastConfig.enableSourceList = TRUE;
multicastConfig.maxGroupNum = 195;
multicastConfig.maxSourceNum = 1000;

rtl_initMulticastSnooping(multicastConfig, 0, TRUE);
```

#### (2) Multicast parameter configuration:

**`void rtl_setMulticastParameters(struct rtl_mCastTimerParameters mCastTimerParameters, uint8* gatewayMac, uint32 gatewayIpv4Addr, uint32 * gatewayIpv6Addr)`**



***rtl\_setMulticastParameters*** specifies multicast module parameters. This API can modify the parameters of multicast module after initialization. If the member of *mCastTimerParameters* has been set to 0, it uses the default value. CPU port learning function is disabled in ***rtl\_initMulticastSnooping***. So you must call ***rtl\_setMulticastParameters*** to set gatewayMac into lookuptable. Also, the unicast IGMP packet whose destination MAC equals to *gatewayMac* and destination IP equals to *gatewayIp* will be accepted.

Example:

```
/*set last member aging time to 2 seconds, others keep default value*/
struct rtl_mCastTimerParameters mCastTimerParameters;
uint8 macAddress[6] = {0x00, 0x0c, 0x29, 0x1a, 0xa0, 0xe7};
uint32 ipv4Add = 0xc0a801fe;
mCastTimerParameters.groupMemberAgingTime = 0;
mCastTimerParameters.lastMemberAgingTime = 2;
mCastTimerParameters.querierPresentInterval = 0;
mCastTimerParameters.dvmrpRouterAgingTime = 0;
mCastTimerParameters.mospfRouterAgingTime = 0;
mCastTimerParameters.pimRouterAgingTime = 0;

rtl_setMulticastParameters(mCastTimerParameter, macAddress, ipv4Add, NULL);
```

### (3) Maintain IGMP timer list:

***int32 rtl\_maintainMulticastSnoopingTimerList(uint32 currentSystemTime)***

***rtl\_maintainMulticastSnoopingTimerList*** maintains multicast snooping timer list. The input parameter *currentSystemTime* means the current system time (unit: seconds).

### (4) Process input packet:

***void rtl\_multicastSnoopingReceive(uint8 \*pMacFrame, uint8 \*pPort)***

***rtl\_multicastSnoopingReceive*** processes input MAC frame. *pMacFrame* points to the input MAC frame. If user wants to get source port information from the CPU tag or VLAN tag carried in the frame, he can set *pPort* to NULL. Otherwise, if user wants to assign a fixed source port to this IGMP frame, please assign a non NULL value to *pPort*.

Example:

```
/*If the frame has CPU tag, and user wants to get source port from CPU tag, set pPort to NULL*/
uint8 *pPort = NULL;
uint16 realtekEtherType = 0x8899;
if (macFrame[12] == ((realtekEtherType & 0xff00) >> 8) && \
    macFrame[13] == (realtekEtherType & 0x00ff) && \
    macFrame[14] & 0xf0 == 0x90)
{
```

```
pPort = NULL;  
rtl_multicastSnoopingReceive (macFrame , pPort);  
}
```

**(5) Process output packet:**

***int32 rtl\_multicastSnoopingSend(uint8 \*pMacFrame, uint32 priorFreeBytes, uint32 posteriorFreeSpace, uint8 \*\*ppNewMacFrame)***

Call ***rtl\_multicastSnoopingSend*** to forward a multicast MAC frame. *macFrame* is the pointer of MAC frame to transmitted. *PriorFreeBytes* and *posteriorFreeSpace* indicate if there is enough space for CPU tag inserting. *ppNewMacFrame* is the pointer of new MAC frame to be forwarded (maybe insert CPU tag or not ).

Example:

```
uint8 * newMacFrame = NULL;  
/*insert CPU tag with appropriate port mask*/  
rtl_multicastSnoopingSend(macFrame, 4, 0, &newMacFrame);  
  
/*call glue function to forward packet with CPU tagged*/  
rtl_glueNicSend(newMacFrame, macFrameLength + 4);
```

**(6) Disable multicast snooping:**

***int32 rtl\_disableMulticastSnooping(void)***

Call ***rtl\_disableMulticastSnooping*** to disable multicast snooping. It will free the memory for multicast snooping module.

Note: 8306E couldn't store the source list of IGMPV3/MLDV2. So it must use the software forwarding to ensure the veracity.

## 6 Application

### 6.1 XDSL

In this application, CPU and RTL8306E form a XDSL modem. All the ports of RTL8306E are LAN ports while the port on CPU is WAN port. Let 8306E's port4 work as CPU port, and CPU connects with port4 through MII port. Suppose the CPU has set its MII port as MAC mode, configure port4 of RTL8306E into PHY mode. Both the ports are set into force mode, 100Mbps and full duplex. Packets with internal IP addresses will be transmitted to WAN port with translated IP address and layer4 port.

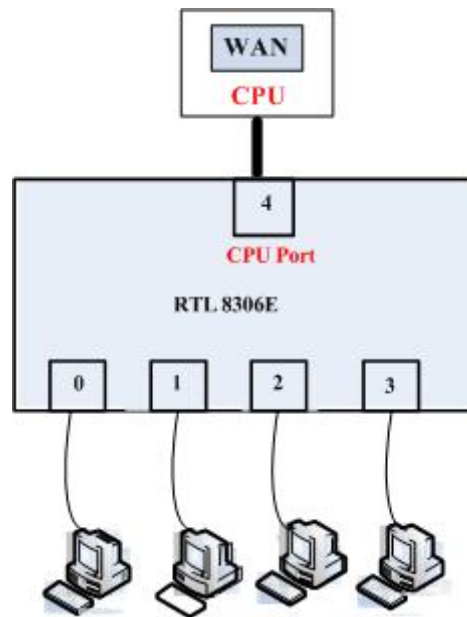


Figure 6-1 XDSL

Example codes:

```

rtk_priority_select_t priDec;
rtk_qos_pri2queue_t pri2qid;
rtl8306e_qosSchPara_t schPara;
rtk_port_phy_ability_t ability;

/* initialize Chip */
rtk_switch_init();

/* initialize QOS */
rtk_qos_init(4);

/*Set port-base priority as highest level, the others are 1*/
priDec.port_pri = 5;
priDec.dot1q_pri = 1;
priDec.dscp_pri = 1;

```

```
priDec.acl_pri = 1;
priDec.vid_pri = 1;
rtk_qos_priSel_set(&priDec);

/*Enable port-based priority, disable DSCP-based, 1Q-based and CPU tag priority*/
for (port = 0; port < 4; port++)
{
    rtl8306e_qos_priSrcEnable_set(port, RTL8306_PBP_PRIO, TRUE);
    rtl8306e_qos_priSrcEnable_set(port, RTL8306_DSCP_PRIO, FALSE);
    rtl8306e_qos_priSrcEnable_set(port, RTL8306_1QBP_PRIO, FALSE);
    rtl8306e_qos_priSrcEnable_set(port, RTL8306_CPUTAG_PRIO, FALSE);

    /* disable priority remarking on port 0~3*/
    rtk_qos_1pRemarkEnable_set(port, DISABLED);
}

/* Set port 0~3 to port-based priority 0~3 */
rtk_qos_portPri_set(0, 0);
rtk_qos_portPri_set(1, 1);
rtk_qos_portPri_set(2, 2);
rtk_qos_portPri_set(3, 3);

/*set schedule parameter for CPU port*/
rtl8306e_qos_portSchedulingMode_set(4, 1, 0x0);
rtl8306e_qos_schedulingPara_get(1, &schPara);
schPara.q0_wt = 0x1;
schPara.q1_wt = 0x2;
schPara.q2_wt = 0x4;
schPara.q3_wt = 0x8;
rtl8306e_qos_schedulingPara_set(1, schPara);

/*set cpu port's flow control mode to set 1*/
rtl8306e_qos_portFlcMode_set(4, 1);
```

## 6.2 EPON ONU

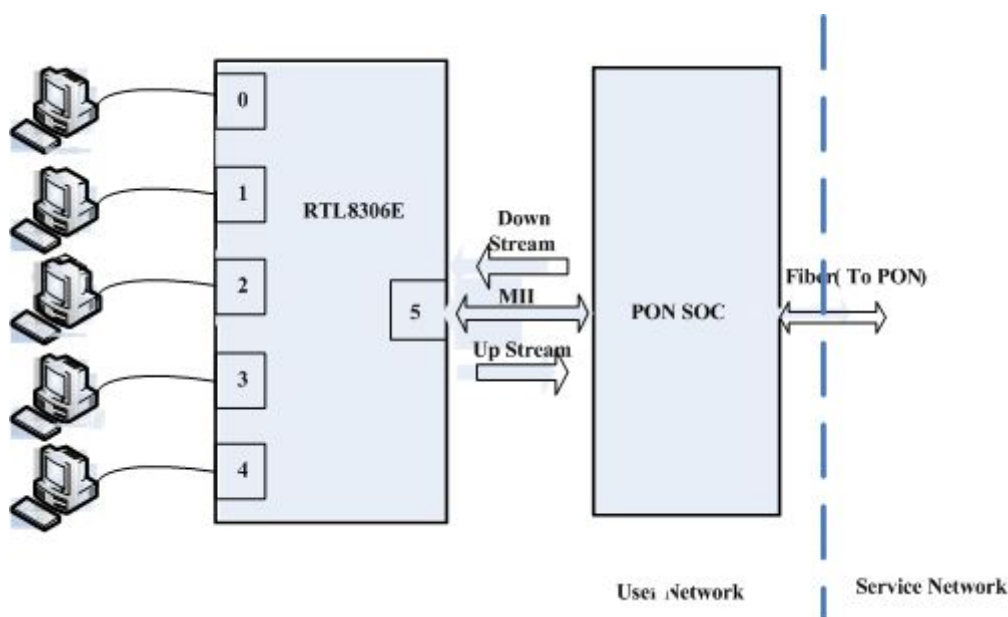


Figure 6-2 EPON ONU

Switch could be configure one of 3 modes, In the transparent mode, the switch is transparent for VLAN; in the untag mode, upstream only accepts untagged packet, downstream only accept tagged packet; in the tag mode, both upstream and downstream only accept tagged packet.

| User Network     | Service Network | Upstream Action |               | Downstream Action |         |
|------------------|-----------------|-----------------|---------------|-------------------|---------|
|                  |                 | Frame type      | Action        | Frame type        | Action  |
| Transparent Mode |                 | untagged        | none          | untagged          | none    |
|                  |                 | tagged          | none          | tagged            | none    |
| Untag Mode       |                 | untagged        | Attach (PVID) | untagged          | discard |
|                  |                 | tagged          | discard       | tagged            | remove  |
| Tag Mode         |                 | untagged        | discard       | untagged          | discard |
|                  |                 | tagged          | none          | tagged            | none    |

Example codes:

```

rtk_mode_ext_t mode;
rtk_port_mac_ability_t portAbility;
rtk_vlan_t vid;
rtk_portmask_t mbrmsk, untagmsk;
rtk_fid_t fid;
uint32 port;

```

**Transparent mode:**

```
/* initialize Chip */
rtk_switch_init();

/* initialize QOS */
rtk_qos_init(4);

/* enable port5 and set it into force mode, 100Mbps and full duplex*/
mode = MODE_EXT_MII_PHY;
portAbility.link = 1;
portAbility.speed = PORT_SPEED_100M;
portAbility.duplex = PORT_FULL_DUPLEX;
portAbility.rxpause = 1;
portAbility.txpause = 1;
rtk_port_macForceLinkExt0_set(mode, &portAbility);

/* Set port5 as CPU port and not insert CPU tag to the packets transmitted to CPU */
rtk_cpu_tagPort_set (5, CPU_INSERT_TO_NONE);

rtl8306e_vlan_transparentEnable_set(TRUE);
```

**Untag mode:**

```
/* initialize Chip */
rtk_switch_init();

/* initialize QOS */
rtk_qos_init(4);

/* enable port5 and set it into force mode, 100Mbps and full duplex*/
mode = MODE_EXT_MII_PHY;
portAbility.link = 1;
portAbility.speed = PORT_SPEED_100M;
portAbility.duplex = PORT_FULL_DUPLEX;
portAbility.rxpause = 1;
portAbility.txpause = 1;
rtk_port_macForceLinkExt0_set(mode, &portAbility);

/* Set port5 as CPU port and do not insert CPU tag to the packets transmitted to CPU */
rtk_cpu_tagPort_set (5, CPU_INSERT_TO_NONE);

/* initialize VLAN */
rtk_vlan_init();

/*delete default vlan*/
rtk_vlan_destroy(1);
```

```
/*Enable the VLAN ingress filter function for all ports*/
rtk_vlan_portIgrFilterEnable_set(0, ENABLED);

/*Set port0~4 accept only un-tagged packet*/
for (port = 0; port <=4; port ++ )
{
    rtk_vlan_portAcceptFrameType_set (port, ACCEPT_FRAME_TYPE_UNTAG_ONLY);
}

/*Set port5 accept only tagged packet*/
rtk_vlan_portAcceptFrameType_set (5, ACCEPT_FRAME_TYPE_TAG_ONLY);

/* create VLAN 10 and set port 0,1,2,5 into its member set; remove port 0/1/2 egress packet's */
/*vlan tag, while do not touch port 5 egress packet's vlan tag */
vid = 10;
mbrmsk.bits[0]=0x27;
untagmsk.bits[0]=0x7;
fid = 0;
rtk_vlan_set(vid, mbrmsk, untagmsk, fid);

/* create VLAN 20 and set port 3,4,5 into its member set; remove port 3/4 egress packet's vlan tag,*/
/* while do not touch port 5 egress packet's vlan tag*/
vid = 20;
mbrmsk.bits[0]=0x38;
untagmsk.bits[0]=0x18;
fid = 0;
rtk_vlan_set(vid, mbrmsk, untagmsk, fid);

/*set port 0/1/2 PVID to 10, and port 3/4/5 PVID to 20*/
rtk_vlan_portPvid_set(0, 10, 0);
rtk_vlan_portPvid_set(1, 10, 0);
rtk_vlan_portPvid_set(2, 10, 0);
rtk_vlan_portPvid_set(3, 20, 0);
rtk_vlan_portPvid_set(4, 20, 0);
rtk_vlan_portPvid_set(5, 20, 0);

/*set the untaged packets ingress from port0~4 to be inserted a VLAN tag when they are*/
/*transmitted out of port5, and the VID in the tag is ingress port's PVID.*/
rtl8306e_vlan_tagInsert_set(5, 0x1F);
```

**Tagged mode:**

```
/* initialize Chip */
rtk_switch_init();
```

```
/* initialize QOS */
rtk_qos_init(4);

/* enable port5 and set it into force mode, 100Mbps and full duplex*/
mode = MODE_EXT_MII_PHY;
portAbility.link = 1;
portAbility.speed = PORT_SPEED_100M;
portAbility.duplex = PORT_FULL_DUPLEX;
portAbility.rxpause = 1;
portAbility.txpause = 1;
rtk_port_macForceLinkExt0_set(mode, &portAbility) ;

/* Set port5 as CPU port and not insert CPU tag to the packets transmitted to CPU */
rtk_cpu_tagPort_set (5, CPU_INSERT_TO_NONE);

/* initialize VLAN */
rtk_vlan_init();

/*delete default vlan*/
rtk_vlan_destroy(1);

/*Enable the VLAN ingress filter function for all ports*/
rtk_vlan_portIgrFilterEnable_set(0, ENABLED);

/*Set port0~5 accept only un-tagged packet*/
for (port = 0; port <=5; port ++ )
{
    rtk_vlan_portAcceptFrameType_set (port, ACCEPT_FRAME_TYPE_TAG_ONLY);
}

/* create VLAN 10 and set port 0,1,2,5 into its member set; and not touch egress packet's vlan tag*/
vid = 10;
mbrmsk.bits[0]=0x27;
untagmsk.bits[0]=0x0;
fid = 0;
rtk_vlan_set(vid, mbrmsk, untagmsk, fid);

/* create VLAN 20 and set port 3,4,5 into its member set; and not touch egress packet's vlan tag.*/
vid = 20;
mbrmsk.bits[0]=0x38;
untagmsk.bits[0]=0x0;
fid = 0;
rtk_vlan_set(vid, mbrmsk, untagmsk, fid);
```



```
/*set port 0/1/2 PVID to 10, and port 3/4/5 PVID to 20*/  
rtk_vlan_portPvid_set(0, 10, 0);  
rtk_vlan_portPvid_set(1, 10, 0);  
rtk_vlan_portPvid_set(2, 10, 0);  
rtk_vlan_portPvid_set(3, 20, 0);  
rtk_vlan_portPvid_set(4, 20, 0);  
rtk_vlan_portPvid_set(5, 20, 0);
```

## 7 Port number

The extension port number definitions between RTL8306E and RTL8306M and RTL8304E are different. For different ASIC users, some configuration code modifications are needed.

Please refer to the following tables. The Mac number in the table shows the physical port ID for RTL8306E/8306M/8304E. These port IDs are also applied to all other APIs which need a port ID as parameters.

### 7.1 Extension ports

#### RTL8306E

| API                                                                                      | MAC number |
|------------------------------------------------------------------------------------------|------------|
| rtk_port_macForceLinkExt0_set(rtk_mode_ext_t mode, rtk_port_mac_ability_t *pPortability) | 4          |
| rtk_port_macForceLinkExt1_set(rtk_mode_ext_t mode, rtk_port_mac_ability_t *pPortability) | 5          |

#### RTL8306M

| API                                                                                      | MAC number |
|------------------------------------------------------------------------------------------|------------|
| rtk_port_macForceLinkExt1_set(rtk_mode_ext_t mode, rtk_port_mac_ability_t *pPortability) | 5          |

#### RTL8304E

| API                                                                                      | MAC number |
|------------------------------------------------------------------------------------------|------------|
| rtk_port_macForceLinkExt0_set(rtk_mode_ext_t mode, rtk_port_mac_ability_t *pPortability) | 2          |
| rtk_port_macForceLinkExt1_set(rtk_mode_ext_t mode, rtk_port_mac_ability_t *pPortability) | 3          |

### 7.2 Port mapping

RTL8306E/8304E support dual MII interfaces, but RTL8306M only support one MII interface.

In RTL8306E, Mac 4 and PHY 4 can be connected internally to be port 4. But Mac 4 and Mac 5 could be also used to be external port 0 and port 1, as (R)MII 1 and (R)MII 2. (R)MII 1 and (R)MII 2 are described in datasheet. In the following table, you will see that RTL8306E support port 0~4 and external port 0~1. But port 4 and external port 0 could not be used at the same time. So RTL8306E only support 6

ports at most at one time.

In RTL8306M, MAC 4 and PHY 4 are connected internally to be port 4, so it only has Mac 5 to be used be external port 1 as (R)MII 1. RTL8306M support port port 0~4 and external port 1. So RTL8306M only support 6 port at most at one time.

In RTL8304E, MAC 2 and MAC 3 could be used to be external port 0 and port 1, as (R)MII 1 and (R)MII 2. RTL8304E support port 0~1 and external port 0~1. So RTL8304E only support 4 port at most at one time.

The port mapping of all supported chip module is listed as follows:

| Chip     | Port 0 | Port 1 | Port 2 | Port 3 | Port 4 | Ext 0((R)MII 1) | Ext 1((R)MII 2) |
|----------|--------|--------|--------|--------|--------|-----------------|-----------------|
| RTL8306E | ✓      | ✓      | ✓      | ✓      | ✓      | ✓               | ✓               |
| RTL8306M | ✓      | ✓      | ✓      | ✓      | ✓      |                 | ✓               |
| RTL8304E | ✓      | ✓      |        |        |        | ✓               | ✓               |

---

**Realtek Semiconductor Corp.****Headquarters**

No. 2, Innovation Road II, Hsinchu Science Park,

Hsinchu 300, Taiwan, R.O.C.

Tel: 886-3-5780211      Fax: 886-3-5776047

[www.realtek.com](http://www.realtek.com)