NOT FOR PUBLIC RELEASE

RTL8363 Series
RTL8364 Series
RTL8365 Series
RTL8366 Series
RTL8367 Series
RTL8370 Series
RTL8310SR

UN-MANAGED 10/100M & 10/100/100M SWITCH CONTROLLER

Programming Guide

(CONFIDENTIAL: Development Partners Only)

Rev. 1.4.0
11th June 2020

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

COPYRIGHT

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

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

Revision	Release Date	Summary
1.4.0	2020/06/11	First Release

Table of Contents

List of Tables

List of Figures

1. Overview

This API package supports multiple Realtek Switches. The term “switch” is used to represent all switch chips supported by this API package.

This API package contains ASIC drivers. ASIC drivers provide general APIs that based on user configuration to configure relative ASIC registers. Inside of the ASIC driver, it uses GPIO to emulate SMI signal, MDC/MDIO or SPI to communicate with switch. This part needs to be porting to the target platform.

2. Directory Structure

The switch release package is distributed with the following files:

Table 1. Switch Software Package File List

File name	Description
dal/	Distribution Abstraction Layer
dal/rtl8367c	Distribution Abstraction Layer of RTL8367C Series
dal/rtl8367d	Distribution Abstraction Layer of RTL8367D Series
acl.c	Source code of API for ACL.
acl.h	Definition of API for ACL.
cpu.c	Source code of API for CPU tag.
cpu.h	Definition of API for CPU tag.
dot1x.c	Source code of API for dot1X
dot1x.h	Definition of API for dot1X
eee.c	Source code of API for EEE
eee.h	Definition of API for EEE
i2c.c	Source code of API for I2C
i2c.h	Definition of API for I2C
igmp.c	Source code of API for IGMP
igmp.h	Definition of API for IGMP
interrupt.c	Source code of API for Interrupt
interrupt.h	Definition of API for Interrupt
l2.c	Source code of API for L2
l2.h	Definition of API for L2
leaky.c	Source code of API for Leaky
leaky.h	Definition of API for Leaky
led.c	Source code of API for LED
led.h	Definition of API for LED
mirror.c	Source code of API for Mirror
mirror.h	Definition of API for Mirror
oam.c	Source code of API for OAM

oam.h	Definition of API for OAM
port.c	Source code of API for port
port.h	Definition of API for port
ptp.c	Source code of API for PTP
ptp.h	Definition of API for PTP
qos.c	Source code of API for QoS
qos.h	Definition of API for QoS
rate.c	Source code of API for Rate
rate.h	Definition of API for Rate
rldp.c	Source code of API for RLDP
rldp.h	Definition of API for RLDP
rtk_switch.c	Source code of API for switch core function
rtk_switch.h	Definition of API for switch core function
stat.c	Source code of API for statistic
stat.h	Definition of API for statistic
storm.c	Source code of API for storm control
storm.h	Definition of API for storm control
svlan.c	Source code of API for SVLAN
svlan.h	Definition of API for SVLAN
trap.c	Source code of API for trap
trap.h	Definition of API for trap
trunk.c	Source code of API for trunk
trunk.h	Definition of API for trunk
vlan.c	Source code of API for VLAN
vlan.h	Definition of API for VLAN
rtk_types.h	Type definition of API
rtk_errno.h	Error codes for ASIC driver

3. ASIC Driver

This API package is a passive software and independent from any OS. That is, by porting it into user's platform, users can initialize and configure switch via this API package.
This API package works like a "switch configuration tool". It is not a driver at kernel of embedded OS. Users should treat this API as an application which response to control switch.

3.1. Porting issue

The management interface of switch is SMI (Serial Management Interface), SCK is clock and SDA transmits data. The demo code using two GPIO pins to emulate SDA/SCK signals. Porting ASIC driver to customer platform needs to modify files **rtlxxxx_smi.c** and **rtlxxxx_smi.h** in particular Distribution Abstraction Layer (dal) directory.

Table 2. DAL Supports chips

DAL	rtl8367c		rtl8367d
	RTL8363NB	RTL8363NB-VB	
	RTL8363SC	RTL8363SC-VB	
	RTL8364NB	RTL8364NB-VB	
	RTL8365MB-VC		RTL8365MB-VD
	RTL8366SC		
			RTL8366SD
	RTL8367RB-VB		RTL8367RB-VC
	RTL8367SB		
	RTL8367S		RTL8367S-VB
			RTL8367SC
	RTL8370MB		
	RTL8310SR		

By default, user should compile and link all .c files in this API package. It can support all switch chips in above table. However, users may want to reduce the size of this API package. In this situation, build only

one DAL (Distribution Abstraction Layer) is possible but users have to make the switch chips which used in your project is supported by a specific DAL.

For example, only RTL8365MB-VD will be used and users can only build 1 DAL which is **rtl8367d**. If RTL8365MB-VC and RTL8365MB-VD will be used, both DALs should be built into user's application/module to support both chips.

If not all DALs are built, users should add additional flag in Makefile.

Only **rtl8367c** DAL Example:

```
/* Makefile in linux system */

CFLAGS += -DCONFIG_DAL_RTL8367C
```

Only **rtl8367d** DAL Example:

```
/* Makefile in linux system */

CFLAGS += -DCONFIG_DAL_RTL8367D
```

If all DALs are built, users don't need to add above flag.

3.2. I2C

In **rtlxxxx_smi.c** and **rtlxxxx_smi.h**, there are two kinds of realization, one is realized by calling GPIO API, and the other is by configuring CPU GPIO register directly. If you want to write your own SDA/SCK code, you should follow these steps:

Firstly, choose two GPIO pins, specify one as SCK, the other as SDA; 2 local variables in smi.c represent these GPIO pins. Customer should redefine/declare them in smi.c

```
rtk_uint32 rtlxxxx_smi_SCK; /* GPIO used for SMI Clock Generation */
rtk_uint32 rtlxxxx_smi_SDA; /* GPIO used for SMI Data signal */
```

Secondly, rewrite four local functions **_smi_start**, **_smi_readBit**, **_smi_writeBit** and **_smi_stop**. **_smi_start** sets SDA pin from high to low at SCK high state. **_smi_stop** sets SDA pin from low to high at SCK high state. **_smi_readBit** generates 1 -> 0 transition and sampled at 1 to 0 transition time, thus read one bit from SDA/SCL interface. **_smi_writeBit** generates 0 -> 1 transition and put data ready during 0 to 1 whole period, it write one bit into the interface. The sample code for generating I2C waveform already in these 4 local functions is pseudo-implemented by 3 GPIO control function. These 3 GPIO control function should be redefined/implemented.

```
#define GPIO_DIRECTION_SET(gpioID, direction)
```

```
#define GPIO_DATA_SET(gpioID, data)
#define GPIO_DATA_GET(gpioID, pData)
```

Between SCK 1 and 0 transitions, proper delay should be added; delay time would affect the interface accessing speed, the less delay, the higher speed. The following define is listed in **rtlxxxx_smi.c**.

```
#define DELAY  10000  /* the less delay, the higher speed */
```

3.3. MDC/MDIO

The MDC/MDIO interface should co-operate with hardware setup. If users want to use MDC/MDIO interface instead of SMI interface, the register access sample code of MDC/MDIO is in **rtlxxxx_smi.c**. For MDC/MDIO interface, users also need to define the flag "MDC_MDIO_OPERATION" in Makefile.

Example:

/* Makefile in linux system */

CFLAGS += -DMDC_MDIO_OPERATION

The PHY ID used at Realtek unmanaged switch could be 0 or 29 depend on the strapping pin configuration. Customer should define the PHY ID in **rtlxxxx_smi.c** which matches the H/W setting.

```
#define MDC_MDIO_PHY_ID     0  /* PHY ID 0 or 29 */
```

The MDC/MDIO register access procedure is 4 steps combination of normal MDC/MDIO access. The PHY ID used in this interface for access switch is 0. The MDC/MDIO interface is only acted as an indirect access interface for accessing internal switch register. Without correct commands order and combination, switch doesn't response. The command order for reading and writing a register is listed as following. This sample codes are also put into rtlxxxx_smi_write() and rtlxxxx_smi_read() in **rtlxxxx_smi.c**.

```
rtk_int32 rtlxxxx_smi_write(rtk_uint32 mAddrs, rtk_uint32 rData)
{
```

```
    /* Write address control code to register 31 */
    MDC_MDIO_WRITE(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_CTRL0_REG, MDC_MDIO_ADDR_OP);

    /* Write address to register 23 */
    MDC_MDIO_WRITE(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_ADDRESS_REG, mAddrs);

    /* Write data to register 24 */
    MDC_MDIO_WRITE(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_DATA_WRITE_REG, rData);

    /* Write data control code to register 21 */
    MDC_MDIO_WRITE(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_CTRL1_REG, MDC_MDIO_WRITE_OP);

    return RT_ERR_OK;
}
```

```
rtk_int32 rtlxxxx_smi_read(rtk_uint32 mAddrs, rtk_uint32 *rData)
{

    /* Write address control code to register 31 */
    MDC_MDIO_WRITE(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_CTRL0_REG, MDC_MDIO_ADDR_OP);

    /* Write address to register 23 */
    MDC_MDIO_WRITE(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_ADDRESS_REG, mAddrs);

    /* Write read control code to register 21 */
    MDC_MDIO_WRITE(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_CTRL1_REG, MDC_MDIO_READ_OP);

    /* Read data from register 25 */
    MDC_MDIO_READ(MDC_MDIO_PREAMBLE_LEN, MDC_MDIO_DUMMY_ID,
MDC_MDIO_DATA_READ_REG, rData);

    return RT_ERR_OK;
```

```
}
```

The ***MDC_MDIO_WRITE*** and ***MDC_MDIO_READ*** should be ported to user's platform. They are standard MDC/MDIO access procedure. rtlxxxx_smi_read() and rtlxxxx_smi_write() use these 2 functions to generate a series of MDC/MDIO accesses (4 steps) to read or write switch register.

Users should redefine/create ***MDC_MDIO_WRITE*** and ***MDC_MDIO_READ***. The prototype of these 2 functions is given as following.

```
Void MDC_MDIO_WRITE (rtk_uint32 preamble_len,
                     rtk_uint32 phy_id,
                     rtk_uint32 register_id,
                     rtk_uint32 data);

void MDC_MDIO_READ (rtk_uint32 preamble_len,
                    rtk_uint32 phy_id,
                    rtk_uint32 register_id,
                    rtk_uint32 *pData);
```

The parameters in these 2 functions are

preamble_len: The preamble length of MDC_MDIO interface.

Phy_id: PHY ID. Switch uses 0 or 29.

Register_id: Register ID.

Data: The data be written.

pData: The pointer of data be read.

3.4. SPI

The SPI interface should co-operate with hardware setup. If users want to use SPI interface instead of SMI interface, the register access sample code of SPI is in **rtlxxxx_smi.c**. For SPI interface, users also need to define the flag "SPI_OPERATION" in Makefile.

Example:

/* Makefile in linux system */

CFLAGS += -DSPI_OPERATION

The following 2 functions should be ported to target platform.

```
#define SPI_WRITE(data, length)    /* SPI interface write function */
#define SPI_READ(pData, length)    /* SPI interface read function */
```

3.5. Boot up time

Switch need 1 second to finish booting (after power up or reset). After that, switch will be ready to be initialized and configured. If users try to configure switch before it finishes booting, the configuration may loss.

3.6. Multi-process Protection

In multi-process environment, API might be interrupted due to a higher priority process calls another API. This may cause unpredictable result. To solve this, RTK_API_LCOK and RTK_API_UNLOCK are defined in rtk_switch.h. Users can redefine these macros to use critical section protection method provided by user's platform. (ie. Mutex lock/unlock in Linux)

4. RTK API for switch

This section introduces some important APIs in this software package. Sample code for using API is also provided. Users should notice that not all APIs are included in this document. Please refer to the "Realtek_Unmanaged_Switch_API_Document.pdf" for all APIs.

4.1. Chip initial

The default registers setting would be written by EEPROM. For those systems without EEPROM, user can set registers to default suggestion values by API ***rtk_switch_init*** after chip power-up.

Int32 rtk_switch_init()

This API should be called before using any other API if there is no EEPROM in the system. Using any other API before using ***rtk_switch_init*** may cause unpredicted error.

4.2. VLAN

Switch supports 4K VLAN. When the switch starts, the VLAN function is disabled as the default configuration. User should use API ***rtk_vlan_init*** to initialize the VLAN before using it. The initialization will enable VLAN functions 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 set or clear member set and untag set for VID 0~4095 through API ***rtk_vlan_set*** and using ***rtk_vlan_get*** to get the information of member set and untag set on specified VLAN. For Port-Based VLAN, there are two APIs ***rtk_vlan_portPvid_set*** and ***rtk_vlan_portPvid_get*** are 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***. Both of them are disabled by default.

Here are explanations and example codes for VLAN APIs.

Int32 rtk_vlan_init(void)

This API should be called before using VLAN. It enables VLAN functions and sets 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, the switch uses 4K VLAN mapping for tagged frames while using Port-Based VLAN mapping for untagged frames. After invocating ***rtk_vlan_init***, user can change the default VLAN arrangement 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 : UTP0, UTP1, UTP2 ;  */
/* VLAN2 member : UTP3, UTP4, EXT0 ;  */
rtk_vlan_cfg_t vlan1, vlan2;

memset(&vlan1, 0x00, sizeof(rtk_vlan_cfg_t));
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT0);
```

```
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT1);
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT2);
RTK_PORTMASK_PORT_SET(vlan1.untag, UTP_PORT0);
RTK_PORTMASK_PORT_SET(vlan1.untag, UTP_PORT1);
RTK_PORTMASK_PORT_SET(vlan1.untag, UTP_PORT2);
rtk_vlan_set(VLAN1, &vlan1);

memset(&vlan2, 0x00, sizeof(rtk_vlan_cfg_t));
RTK_PORTMASK_PORT_SET(vlan2.mbr, UTP_PORT3);
RTK_PORTMASK_PORT_SET(vlan2.mbr, UTP_PORT4);
RTK_PORTMASK_PORT_SET(vlan2.mbr, EXT_PORT0);
RTK_PORTMASK_PORT_SET(vlan2.untag, UTP_PORT3);
RTK_PORTMASK_PORT_SET(vlan2.untag, UTP_PORT4);
RTK_PORTMASK_PORT_SET(vlan2.untag, EXT_PORT0);
rtk_vlan_set(VLAN2, &vlan2);

/* set PVID for each port */
rtk_vlan_portPvid_set(UTP_PORT0, VLAN1, 0);
rtk_vlan_portPvid_set(UTP_PORT1, VLAN1, 0);
rtk_vlan_portPvid_set(UTP_PORT2, VLAN1, 0);
rtk_vlan_portPvid_set(UTP_PORT3, VLAN2, 0);
rtk_vlan_portPvid_set(UTP_PORT4, VLAN2, 0);
rtk_vlan_portPvid_set(EXT_PORT0, VLAN2, 0);
```

*int32 rtk_vlan_set(rtk_vlan_t vid, rtk_vlan_cfg_t *pVlanCfg)*

```
typedef rtk_uint32  rtk_vlan_t;
typedef struct  rtk_vlan_cfg_s
{
    rtk_portmask_t      mbr;
    rtk_portmask_t      untag;
    rtk_uint16      ivl_en;
    rtk_uint16      fid_msti;
    rtk_uint16      envlanpol;
    rtk_uint16      meteridx;
    rtk_uint16      vbpen;
    rtk_uint16      vbpri;
```

```
}rtk_vlan_cfg_t;
```

The API can set member set, untagged set, and filtering database to the specified VLAN. You can also clear the VLAN entry by specifying its member set and untagged set to zero.

Example:

```
/* set port 0,1,2,3 as member set and port 2,3 as untag set to VLAN 1000
with fid 0 */
rtk_vlan_cfg_t vlan1;

memset(&vlan1, 0x00, sizeof(rtk_vlan_cfg_t));
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT0);
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT1);
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT2);
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT3);
RTK_PORTMASK_PORT_SET(vlan1.untag, UTP_PORT2);
RTK_PORTMASK_PORT_SET(vlan1.untag, UTP_PORT3);
rtk_vlan_set(1000, &vlan1);

/* clear member set and untag set for VLAN 1000 */
memset(&vlan1, 0x00, sizeof(rtk_vlan_cfg_t));
RTK_PORTMASK_CLEAR(vlan1.mbr);
RTK_PORTMASK_CLEAR(vlan1.untag);
rtk_vlan_set(1000, &vlan1);
```

*int32 rtk_vlan_get(rtk_vlan_t vid, rtk_vlan_cfg_t *pVlanCfg)*

```
typedef rtk_uint32  rtk_vlan_t;
typedef struct  rtk_vlan_cfg_s
{
    rtk_portmask_t      mbr;
    rtk_portmask_t      untag;
    rtk_uint16      ivl_en;
    rtk_uint16      fid_msti;
    rtk_uint16      envlanpol;
    rtk_uint16      meteridx;
```

```
    rtk_uint16      vbpen;
    rtk_uint16      vbpri;
}rtk_vlan_cfg_t;
```

The API can get the information of member set, untagged set, and filtering database on the specified VLAN. The information is retrieved in ***pVlanCfg***.

Example:

```
/* get the member set and untagged set on VLAN 1000 */
rtk_vlan_cfg_t vlan1;

memset(&vlan1, 0x00, sizeof(rtk_vlan_cfg_t));
rtk_vlan_get(1000, &vlan1);
```

int32 rtk_vlan_portPvid_set(rtk_port_t port, rtk_vlan_t pvid, rtk_pri_t priority)

Set each port's PVID. The priority will not be used to untagged frames, and it should be assigned to 0. The untagged frame will be seen as no 802.1Q-based priority and then system can map the untagged frame to the proper priority by QoS mechanisms. Before using the API, user should call ***rtk_vlan_set*** to set the member set and untag set for the VLAN first

Example1:

```
/* set port0, port1 as member set to VLAN 2000 */
rtk_vlan_cfg_t vlan1;

memset(&vlan1, 0x00, sizeof(rtk_vlan_cfg_t));
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT0);
RTK_PORTMASK_PORT_SET(vlan1.mbr, UTP_PORT1);
rtk_vlan_set(2000, &vlan1);

/* set port0's and port1's PVID to 2000 */
rtk_vlan_portPvid_set(UTP_PORT0, 2000, 0);
rtk_vlan_portPvid_set(UTP_PORT1, 2000, 0);
```

*int32 rtk_vlan_portPvid_get(rtk_port_t port, rtk_vlan_t *pPvid, rtk_pri_t *pPriority)*

Get each port's PVID.

Example:

```
/* get PVID of port1 and 802.1p priority attached to the PVID */
rtk_vlan_t vid ;
rtk_pri_t priority ;

rtk_vlan_portPvid_get(UTP_PORT1, &vid, &priority);
```

---

int32 rtk_vlan_portIgrFilterEnable_set(rtk_port_t port, rtk_enable_t igr_filter)

```
typedef rtk_uint32  rtk_port_t ;
```

The switch will drop the received frames if the ingress port is NOT in the member set of matched VLAN. VLAN ingress filter function is disabled by default and only takes effect while VLAN is enabled.

Example:

```
/* enable the VLAN ingress filter for port 1.*/
rtk_vlan_portIgrFilterEnable_set(UTP_PORT1, ENABLE);
```

---

int32 rtk_vlan_portAcceptFrameType_set(rtk_port_t port, rtk_vlan_acceptFrameType_t accept_frame_type)

```
typedef rtk_uint32  rtk_port_t ;
typedef enum rtk_vlan_acceptFrameType_e
{
    ACCEPT_FRAME_TYPE_ALL = 0, /* untagged, priority-tagged and tagged */
    ACCEPT_FRAME_TYPE_TAG_ONLY,/* tagged */
```

```
    ACCEPT_FRAME_TYPE_UNTAG_ONLY, /* untagged and priority-tagged */
    ACCEPT_FRAME_TYPE_END
} rtk_vlan_acceptFrameType_t;
```

The switch will accept all/tagged/untagged frames. VLAN accept frame type function is disabled (accept all) by default and only take effect while VLAN is enabled.

Example:

```
/* only accept tagged frame on port 1, untagged frame from port1 will be
dropped.*/
rtk_vlan_portAcceptFrameType_set(UTP_PORT1, ACCEPT_FRAME_TYPE_TAG_ONLY);

/* only accept untagged frame on port 2, tagged frame from port2 will be
dropped.*/
rtk_vlan_portAcceptFrameType_set(UTP_PORT2,
ACCEPT_FRAME_TYPE_UNTAG_ONLY);
```

---

int32 rtk_vlan_tagMode_set(rtk_port_t port, rtk_vlan_tagMode_t tag_mode)

```
typedef rtk_uint32  rtk_port_t ;
typedef enum rtk_vlan_tagMode_e
{
    VLAN_TAG_MODE_ORIGINAL = 0,
    VLAN_TAG_MODE_KEEP_FORMAT,
    VLAN_TAG_MODE_REAL_KEEP_FORMAT,
    VLAN_TAG_MODE_PRI,
    VLAN_TAG_MODE_END
} rtk_vlan_tagMode_t;
```

Output frame from ASIC will not be tagged C-tag or untagged C-tag if egress port was set enable keep-Ctag-format function. Receiving frame with untag format will be output with untag format, priority-tag frame will be output as priority-tag frame and c-tagged frame will be output as original c-tagged frame.

Example:

```
/* Keep tag format mode on Port 0 */
```

```
rtk_vlan_tagMode_set(UTP_PORT0, VLAN_TAG_MODE_KEEP_FORMAT) ;

/* Original mode on Port 0*/
rtk_vlan_tagMode_set(UTP_PORT0, VLAN_TAG_MODE_ORIGINAL) ;
```

*int32 rtk_vlan_transparent_set (rtk_port_t egr_port, rtk_portmask_t *pIgr_pmask)*

The API is used to set the VLAN transparent mode setting. A packet which transmitted in a specified egress port and received from one of the port in igr_pmask will not filtered by VLAN member port setting.

Example:

```
/* Packet RX from Port 1,2 and TX to Port 0 should be tranparent */
rtk_portmask_t   pmask ;

RTK_PORTMASK_CLEAR(pmask) ;
RTK_PORTMASK_PORT_SET(pmask, UTP_PORT1) ;
RTK_PORTMASK_PORT_SET(pmask, UTP_PORT2) ;
rtk_vlan_transparent_set(UTP_PORT0, &pmask);
```

4.3. VLAN Stacking

Switch supports VLAN Stacking (SVLAN). Users can assign one port to be SVLAN service port(s). When a packet egress from an uplink port, it will be added an extra tag following IEEE 802.1ad definition. There are only packets with configured tag that can be sent to uplink port(s).

In switch, SVLAN entries are shared with 4K VLAN. The TPID in tag can be configured by users and the default value is 0x88a8. In downstream direction, only packets with configured SVLAN tag can be accepted and un-tag or un-match packets will be dropped or trapped to CPU by setup. In upstream direction, the packet SVLAN VID will be decided by CVLAN to SVLAN, ACL or default configuration.

Here are explanations and example codes for SVLAN APIs.

Int32 rtk_svlan_init(void)

This API should be called before using SVLAN. It can setup some rules for SVLAN, such as drop un-tag and un-match packets and clean all SVLAN configurations.

Example:

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

---

int32 rtk_svlan_servicePort_add(rtk_port_t port)

This API can add a SVLAN service port, and this port will support SVLAN function. If users want to change one service port to another, they should delete the current service port and add another service port. The related APIs are ***rtk_svlan_servicePort_del*** and ***rtk_svlan_servicePort_get***.

Example:

```
/* set extension port 0 as SVLAN service port */
rtk_svlan_servicePort_add(EXT_PORT0);
```

---

int32 rtk_svlan_tpidEntry_set(rtk_svlan_tpid_t svlan_tag_id)

The API can change 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 0x8100*/
rtk_svlan_tpidEntry_set(0x8100);
```

*int32 rtk_svlan_memberPortEntry_set(rtk_vlan_t svid, rtk_svlan_memberCfg_t *pSvlan_cfg)*

The API can set member set, priority, and filtering database to specified SVLAN entry. When users configure member port, the service port should also be included.

```
Typedef struct rtk_svlan_memberCfg_s{
    rtk_uint32 svid;
    rtk_portmask_t memberport;
    rtk_portmask_t untagport;
    rtk_uint32 fiden;
    rtk_uint32 fid;
    rtk_uint32 priority;
    rtk_uint32 efiden;
    rtk_uint32 efid;
    rtk_uint32 chk_ivl_svl;
    rtk_uint32 ivl_svl;
}rtk_svlan_memberCfg_t;
```

Example:

```
rtk_svlan_memberCfg_t svlanCfg;

/* set extension port 0 as SVLAN service port */
rtk_svlan_servicePort_add(EXT_PORT0);

/* Set SVID 100 with member UTP port 0,1,2 & EXT port 0 */
memset(&svlanCfg, 0x00, sizeof(rtk_svlan_memberCfg_t));
svlanCfg.svid = 100;
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT1);
```

```
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT2);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, EXT_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT1);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT2);
rtk_svlan_memberPortEntry_set(100, &svlanCfg);
```

int32 rtk_svlan_defaultSvlan_set(rtk_port_t port, rtk_vlan_t svid)

This API is used to set default SVLAN VID per ingress port. When the default SVID is set, the default egress SVID will be the configured SVID.

Example:

```
/* Set SVID 1 with member port UTP Port 0~4 & EXT port 0 */
rtk_svlan_memberCfg_t svlanCfg;

memset(&svlanCfg, 0x00, sizeof(rtk_svlan_memberCfg_t));
svlanCfg.svid = 1;
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT1);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT2);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT3);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT4);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, EXT_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT1);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT2);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT3);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT4);
rtk_svlan_memberPortEntry_set(1, &svlanCfg);

/* Set UTP Port 0 default SVLAN to SVID 1 */
rtk_svlan_defaultSvlan_set(UTP_PORT0, 1);
```

int32 rtk_svlan_c2s_add(rtk_vlan_t vid, rtk_port_t src_port, rtk_vlan_t svid)

The API is used to assign SVID for CVLAN packets. When the API is used, the packets with defined CVID will be assigned to the configured SVID.

Example:

```
/* Set SVID 200 with member port UTP Port 0~4 & EXT port 0 */
rtk_svlan_memberCfg_t svlanCfg;

memset(&svlanCfg, 0x00, sizeof(rtk_svlan_memberCfg_t));
svlanCfg.svid = 200;
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT1);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT2);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT3);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT4);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, EXT_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT1);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT2);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT3);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT4);
rtk_svlan_memberPortEntry_set(200, &svlanCfg);

/* Set CVID 100 with member port 0,1 to SVID 200 */
rtk_svlan_c2s_add(100, UTP_PORT0, 200);
rtk_svlan_c2s_add(100, UTP_PORT1, 200);

/* Set CVID 101 with member port 2,3 to SVID 200 */
rtk_svlan_c2s_add(101, UTP_PORT2, 200);
rtk_svlan_c2s_add(101, UTP_PORT3, 200);
```

int32 rtk_svlan_sp2c_add(rtk_vlan_t svid, rtk_port_t dst_port, rtk_vlan_t cvid)

This API is used to assign C-tag to packets with only S-tag and configured destination port. So packets without C-tag from uplink port to destination will be added a C-tag with assigned CVID.

Example:

```
/* Set SVID 100 packets without C-tag and destination port is UTP Port 0
to CVID 111. */
rtk_svlan_sp2c_add(100, UTP_PORT0, 111);
```

int32 rtk_svlan_untag_action_set(rtk_svlan_untag_action_t action, rtk_vlan_t svid)

The API can configure action of downstream Un-Stag packet. A SVID assigned to the un-stag is also supported by this API. The parameter of svid is only referenced when the action is set to UNTAG_ASSIGN

Example:

```
/* Set SVID 200 with member port 0,1,2,3,4 & EXT_PORT0 */
rtk_svlan_memberCfg_t svlanCfg;

memset(&svlanCfg, 0x00, sizeof(rtk_svlan_memberCfg_t));
svlanCfg.svid = 200;
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT1);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT2);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT3);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, UTP_PORT4);
RTK_PORTMASK_PORT_SET(svlanCfg.memberport, EXT_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT0);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT1);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT2);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT3);
RTK_PORTMASK_PORT_SET(svlanCfg.untagport, UTP_PORT4);
rtk_svlan_memberPortEntry_set(200, &svlanCfg);

/* Set Un-Stag packet action to UNTAG_ASSIGN, the SVID assigned is 200.
*/
rtk_svlan_untag_action_set(UNTAG_ASSIGN, 200);
```

4.4. Lookup Table

Switch supports 2K-entry lookup table that CPU could access it through control interface. The LUT can store unicast(I/G-bit is 0) MAC address and multicast(I/G-bit is 1) MAC address. The LUT uses 4-way hash to store a L2 entry, that is, LUT hashing function generates an 9-bit lookup index and for each index there are 4 entries to save the hash collision entry.

Unicast MAC address entry could be auto-learned by ASIC or written as static entry by CPU through ***rtk_l2_addr_add***. This API can add an IVL or SVL unicast entry.

After a period of time (aging time), ASIC auto-learned entry will be aging out and removed. But entry written by CPU will never be removed unless CPU deletes it.

All dynamic and static L2 unicast entries can be dumped though ***rtk_l2_addr_next_get***. By calling this API repeatedly, switch would return one entry and the address index of this entry at each time. Whenever user want to get next entry, just increase the address index and pass it to ***rtk_l2_addr_next_get*** for retrieve next entry.

rtk_l2_flushType_set can be used to flush dynamic entries in LUT. Users can specify a particular port or VLAN. With port or VLAN specified flush, all dynamic entries learned at specified port or VLAN would be removed.

Multicast MAC address entry is for multicast application. It could not be auto-learned but written by CPU through ***rtk_l2_mcastAddr_add***. If a multicast MAC 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.

All the LUT entry would be written as “Static” Entry. That is, switch will ignore these entries when processing aging task. Static entries would exist in LUT until application use API *rtk_l2_addr_del* to delete it. For multicast MAC address entry, the API ***rtk_l2_mcastAddr_del*** can be used to delete it.

Here are explanations and example codes for lookup table API.

---

*Int32 rtk_l2_addr_add(rtk_mac_t *pMac, rtk_l2_ucastAddr_t *pL2_data)*

```
typedef struct  rtk_mac_s
{
    uint8 octet[ETHER_ADDR_LEN];
} rtk_mac_t;
```

```
typedef struct rtk_l2_ucastAddr_s
{
    rtk_mac_t       mac;
    rtk_uint32      ivl;
    rtk_uint32      cvid;
    rtk_uint32      fid;
    rtk_uint32      efid;
    rtk_uint32      port;
    rtk_uint32      sa_block;
    rtk_uint32      da_block;
    rtk_uint32      auth;
    rtk_uint32      is_static;
    rtk_uint32      priority;
    rtk_uint32      sa_pri_en;
    rtk_uint32      fwd_pri_en;
    rtk_uint32      address;
}rtk_l2_ucastAddr_t;
```

Add a unicast entry to lookup table. The low order bit of the high order octet (I/G-bit) of MAC address must be 0. If hashed index is full of entries that means all added by CPU, it would return error message "RT_ERR_L2_INDEXTBL_FULL". CPU should maintain a database to record the MAC addresses written to LUT for further deletion.

Example:

```
/* write a unicast static entry with MAC address "00-12-34-56-78-AA",
source port 0, IVL, and VID 100 */
rtk_l2_ucastAddr_t    l2_entry;
rtk_mac_t mac;

mac.octec[0] = 0x00;
mac.octec[1] = 0x12;
mac.octec[2] = 0x34;
mac.octec[3] = 0x56 ;
mac.octec[4] = 0x78 ;
mac.octec[5] = 0xAA;
memset(&l2_entry, 0x00, sizeof(rtk_l2_ucastAddr_t));
l2_entry.port = UTP_PORT0 ;
l2_entry.ivl  = 1 ;
```

```
l2_entry.cvid = 100;
l2_entry.fid  = 0;
l2_entry.is_static = 1;

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

*int32 rtk_l2_addr_del(rtk_mac_t *pMac, rtk_l2_ucastAddr_t *pL2_data)*

Delete a unicast entry from lookup table.

Example:

```
/* delete a entry with MAC address "00-12-34-56-78-AA" with IVL and VID
100 */
rtk_mac_t mac ;
rtk_l2_ucastAddr_t    l2_entry ;

mac.octec[0] = 0x00 ;
mac.octec[1] = 0x12 ;
mac.octec[2] = 0x34 ;
mac.octec[3] = 0x56 ;
mac.octec[4] = 0x78 ;
mac.octec[5] = 0xAA ;
l2_entry.ivl  = 1;
l2_entry.cvid = 100;

rtk_l2_addr_del(&mac, &l2_entry);
```

*int32 rtk_l2_addr_get(rtk_mac_t *pMac, rtk_l2_ucastAddr_t *pL2_data)*

Get a unicast entry from lookup table by MAC and FID (or VID).

Example:

```
/* Get a unicast entry with MAC address "00-12-34-56-78-AA" with IVL, VID
100 */
rtk_l2_ucastAddr_t  l2_entry;
```

```
rtk_mac_t  mac;

memset(&l2_entry, 0x00, sizeof(rtk_l2_ucastAddr_t));
mac.octec[0] = 0x00 ;
mac.octec[1] = 0x12 ;
mac.octec[2] = 0x34 ;
mac.octec[3] = 0x56 ;
mac.octec[4] = 0x78 ;
mac.octec[5] = 0xAA ;
l2_entry.ivl  = 1;
l2_entry.cvid = 100;

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

---

*int32 rtk_l2_addr_next_get(rtk_l2_read_method_t read_method, rtk_port_t port, rtk_uint32 *pAddress, rtk_l2_ucastAddr_t *pL2_data)*

Get next unicast entry from lookup table by address.

Example:

```
/* Get all unicast entry at whole system*/
rtk_uint32  address = 0;
rtk_l2_ucastAddr_t l2_data;

while (1)
{
    if((retVal=rtk_l2_addr_next_get(READMETHOD_NEXT_L2UC,
UTP_PORT0, &address, &l2_data))!=RT_ERR_OK)
    {
          break;
    }
    address++;
}

/* Get all unicast entry at port 2 only*/
rtk_uint32  address = 0;
```

```
rtk_l2_ucastAddr_t l2_data;

while (1)
{
    if((retVal=rtk_l2_addr_next_get(READMETHOD_NEXT_L2UCSPA,
UTP_PORT2, &address, &l2_data))!=RT_ERR_OK)
    {
          break;
    }
   address++;
}
```

*int32 rtk_l2_mcastAddr_add (rtk_l2_mcastAddr_t *pMcastAddr)*

Add a multicast entry to lookup table. The low order bit of the high order octet (I/G-bit) of MAC address must be 1.

```
Typedef struct rtk_l2_mcastAddr_s
{
    rtk_uint32      vid;
    rtk_mac_t       mac;
    rtk_uint32      fid;
    rtk_portmask_t  portmask;
    rtk_uint32      ivl;
    rtk_uint32      priority;
    rtk_uint32      fwd_pri_en;
    rtk_uint32      igmp_asic;
    rtk_uint32      igmp_index;
    rtk_uint32      address;
}rtk_l2_mcastAddr_t;
```

Example:

```
/* write a multicast entry with MAC address "01-00-5E-11-22-33", IVL
entry, VID 100 and member port 0,1,2 */
rtk_l2_mcastAddr_t mcastAddr;
```

```
memset(&mcastAddr, 0x00, sizeof(rtk_l2_mcastAddr_t));
mcastAddr.mac.octet[0] = 0x01;
mcastAddr.mac.octet[1] = 0x00;
mcastAddr.mac.octet[2] = 0x5E;
mcastAddr.mac.octet[3] = 0x11;
mcastAddr.mac.octet[4] = 0x22;
mcastAddr.mac.octet[5] = 0x33;
mcastAddr.ivl = 1
mcastAddr.vid = 100;
RTK_PORTMASK_PORT_SET(mcastAddr.portmask, UTP_PORT0);
RTK_PORTMASK_PORT_SET(mcastAddr.portmask, UTP_PORT1);
RTK_PORTMASK_PORT_SET(mcastAddr.portmask, UTP_PORT2);

rtk_l2_mcastAddr_add(&mcastAddr);
```

---

*int32 rtk_l2_mcastAddr_del(rtk_l2_mcastAddr_t *pMcastAddr)*

Delete a multicast entry from lookup table.

Example:

```
/* delete a entry with MAC address “01-00-5E-11-22-33” , IVL entry with
VID 100 */
rtk_l2_mcastAddr_t mcastAddr;

memset(&mcastAddr, 0x00, sizeof(rtk_l2_mcastAddr_t));
mcastAddr.mac.octet[0] = 0x01;
mcastAddr.mac.octet[1] = 0x00;
mcastAddr.mac.octet[2] = 0x5E;
mcastAddr.mac.octet[3] = 0x11;
mcastAddr.mac.octet[4] = 0x22;
mcastAddr.mac.octet[5] = 0x33;
mcastAddr.ivl = 1
mcastAddr.vid = 100;

rtk_l2_mcastAddr_del(&mcastAddr);
```

```
```

*int32 rtk_l2_mcastAddr_get(rtk_l2_mcastAddr_t *pMcastAddr)*

Get a multicast entry from lookup table by MAC and FID.

Example:

```
/* Get a multicast entry with MAC address "01-00-5E-56-78-AA" IVL entry
with VID 100 */
rtk_l2_mcastAddr_t mcastAddr;

memset(&mcastAddr, 0x00, sizeof(rtk_l2_mcastAddr_t));
mcastAddr.mac.octet[0] = 0x01;
mcastAddr.mac.octet[1] = 0x00;
mcastAddr.mac.octet[2] = 0x5E;
mcastAddr.mac.octet[3] = 0x56
mcastAddr.mac.octet[4] = 0x78
mcastAddr.mac.octet[5] = 0xAA
mcastAddr.ivl = 1
mcastAddr.vid = 100;

rtk_l2_mcastAddr_get(&mcastAddr);
```

*int32 rtk_l2_mcastAddr_next_get (rtk_uint32 *pAddress, rtk_l2_mcastAddr_t *pMcastAddr)*

Get next L2 multicast entry from lookup table by address.

Example:

```
/* Get L2 multicast entry at whole system*/
rtk_uint32  address = 0;
rtk_l2_mcastAddr_t mcastAddr;

while(1)
```

```
{
    if((retVal= rtk_l2_mcastAddr_next_get(&address, &mcastAddr)) !=
RT_ERR_OK)
    {
          break;
    }
    address++;
}
```

*int32 rtk_l2_ipMcastAddr_add(rtk_l2_ipMcastAddr_t *pIpMcastAddr)*

Add a L3 multicast entry into lookup table.

```
Typedef struct rtk_l2_ipMcastAddr_s
{
    ipaddr_t        dip;
    ipaddr_t        sip;
    rtk_portmask_t  portmask;
    rtk_uint32      priority;
    rtk_uint32      fwd_pri_en;
    rtk_uint32      igmp_asic;
    rtk_uint32      igmp_index;
    rtk_uint32      address;
}rtk_l2_ipMcastAddr_t;
```

Example:

```
/* write a multicast entry with destination IP = 224.1.2.3 & source IP =
10.1.1.1, member port 0,1,2 */
rtk_l2_ipMcastAddr_t ipMcastAddr;

memset(&ipMcastAddr, 0x00, sizeof(rtk_l2_ipMcastAddr_t));
ipMcastAddr.dip = 0xE0010203;
ipMcastAddr.sip = 0x0A010101;
RTK_PORTMASK_PORT_SET(ipMcastAddr.portmask, UTP_PORT0);
RTK_PORTMASK_PORT_SET(ipMcastAddr.portmask, UTP_PORT1);
RTK_PORTMASK_PORT_SET(ipMcastAddr.portmask, UTP_PORT2);
```

```
rtk_l2_ipMcastAddr_add(&ipMcastAddr);
```

*int32 rtk_l2_ipMcastAddr_del(rtk_l2_ipMcastAddr_t *pIpMcastAddr)*

Delete a L3 multicast entry from lookup table.

Example:

```
/* Delete a multicast entry with destination IP = 224.1.2.3 & source IP =
10.1.1.1 */
rtk_l2_ipMcastAddr_t ipMcastAddr;

memset(&ipMcastAddr, 0x00, sizeof(rtk_l2_ipMcastAddr_t));
ipMcastAddr.dip = 0xE0010203;
ipMcastAddr.sip = 0x0A010101;

rtk_l2_ipMcastAddr_del(&ipMcastAddr);
```

*int32 rtk_l2_ipMcastAddr_get(rtk_l2_ipMcastAddr_t *pIpMcastAddr)*

Get a L3 multicast entry from lookup table.

Example:

```
/* Get a multicast entry with destination IP = 224.1.2.3 & source IP =
10.1.1.1 */
rtk_l2_ipMcastAddr_t ipMcastAddr;

memset(&ipMcastAddr, 0x00, sizeof(rtk_l2_ipMcastAddr_t));
ipMcastAddr.dip = 0xE0010203;
ipMcastAddr.sip = 0x0A010101;

rtk_l2_ipMcastAddr_get(&ipMcastAddr);
```

*int32 rtk_l2_ipMcastAddr_next_get (rtk_uint32 *pAddress, rtk_l2_ipMcastAddr_t *pIpMcastAddr)*

Get next L3 multicast entry from lookup table by address.

Example:

```
/* Get L3 multicast entry at whole system*/
rtk_uint32  address = 0;
rtk_l2_ipMcastAddr_t ipMcastAddr;

while(1)
{
    if((retVal= rtk_l2_ipMcastAddr_next_get(&address, &ipMcastAddr)) !=
RT_ERR_OK)
    {
          break;
    }
    address++;
}
```

int32 rtk_l2_flushType_set(rtk_l2_flushType_t type, rtk_vlan_t vid, rtk_l2_flushItem_t portOrTid)

This function triggers flushing of per-port or per-VLAN dynamic learning entries.

Example:

```
/* Flush entries at Port 2*/
if((retVal=rtk_l2_flushType_set(FLUSH_TYPE_BY_PORT, 0,
UTP_PORT2))!=RT_ERR_OK)
   return retVal;
```

4.5. QoS

Switch QoS function provides maximum 8 queues per port for packet scheduling with queue weight and priority assignment. With different queue number usage, threshold of flow control mechanism will be an important element in throughput improvement.

ASIC supports 1Q based, DSCP, Port, ACL and SVLAN based priority selection sources. There are dedicated selected priority configurations for these assignment sources. Assignment source with the highest selected priority will be chosen and ASIC will use mapped priority to locate desired queue for outputs.

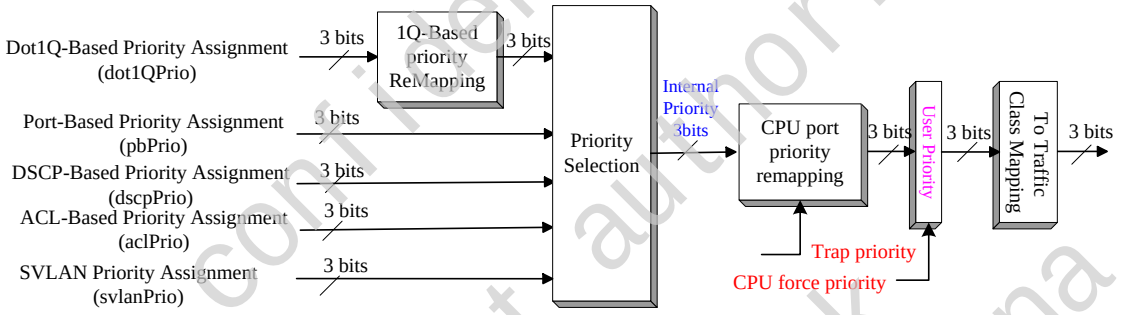

Figure 1. Priority Selection Flow

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

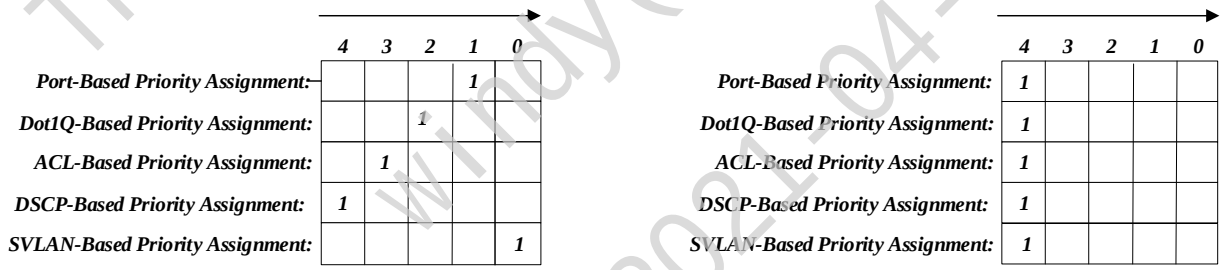

Figure 2. Priority Selection Configuration

Dot1Q priority can be chosen from two methods. One is the receiving frame contained C-tag (Ethernet length/type is 0x8100) and Dot1Q priority is the priority in the priority field of C-tag. Another is received un-tagged frame can be assigned priority from VLAN PVID configuration. Each port has its own PVID setting with VID and priority assignment.

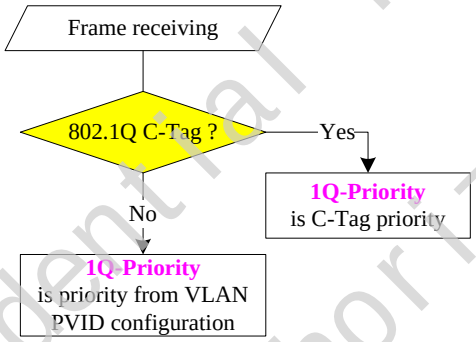

Figure 3. Dot1Q Priority Selection Flow

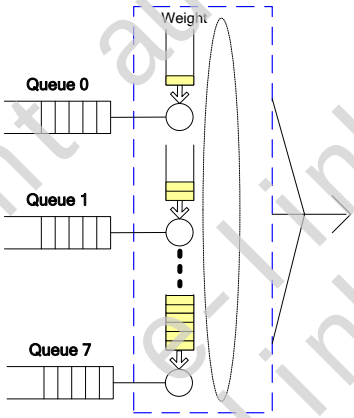

Figure 4. Queue scheduling

While queue number usage is more than one queue, then ASIC will schedule output frames with queue weight. ASIC assigns queue weight as queue id that means queue 0 to queue 7 has weight assigned from 1 to 127. Queue weight is same as output scheduling rate and it gets higher transmit probability in queue id 7 than queue id 0. Each queue can be set as strict, weight fair or weighted round robin. Strict type of queue will be scheduled in higher priority than weight fair queues. It operates as leaky bucket mechanism.

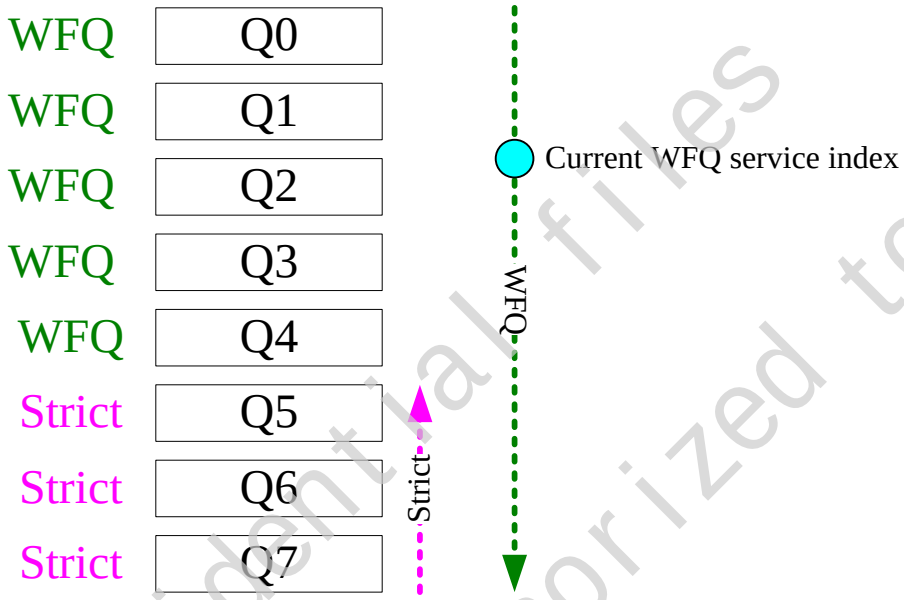

Figure 5. Scheduling between Strict Priority and WFQ

int32 rtk_qos_init(rtk_queue_num_t queueNum)

Queue number setting is 1 that it means the ASIC will have no packet scheduling mechanism and all packets will be transmitted by receiving order. API will assign different flow control thresholds to apply different queue number usage and use below table to locate frame to mapping queue with different priority in different queue number situation.

Table 3. Priority to Queue Assignment

	Number of Available Output Queue							
Priority	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	1	1	1	1	2
3	0	0	0	1	1	2	2	3
4	0	7	1	2	2	3	3	4
5	0	7	1	2	3	4	4	5
6	0	7	7	7	7	7	5	6
7	0	7	7	7	7	7	7	7

The default Dot1Q priority to traffic class (queue priority) mappings is as below table.

Table 4. Dot1Q Priority Remapping

802.1Q priority	Internal priority
0	0

1	1
2	2
3	3
4	4
5	5
6	6
7	7

Example:

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

int32 rtk_qos_portPri_set(rtk_port_t port, rtk_pri_t int_pri))

Switch support Port Priority Setting. User can use this API to set ports' priority at once. This priority would be used as the default priority for frames coming from a port.

Example:

```
/* Set UTP Port 0~4 to priority 0 and EXT Port 0 to Priority 7 */
rtk_qos_portPri_set(UTP_PORT0, 0) ;
rtk_qos_portPri_set(UTP_PORT1, 0) ;
rtk_qos_portPri_set(UTP_PORT2, 0) ;
rtk_qos_portPri_set(UTP_PORT3, 0) ;
rtk_qos_portPri_set(UTP_PORT4, 0) ;
rtk_qos_portPri_set(EXT_PORT0, 7) ;
```

int32 rtk_qos_1pPriRemap_set(rtk_pri_t dot1p_pri , rtk_pri_t int_pri) ;

As mentioned above, 802.1Q priority has a remapping process to map the priority in 802.1Q tag to the absolute priority. (Not internal priority, real internal priority would be selected later) This remapping function can't be turned off. If user doesn't want this function, just set it to the same priority value before remapping.

Example:

```
/* Set Priority 0~3 remap to 0, 4~7 remap to 7 */
```

```
rtk_qos_1pPriRemap_set(0,0);
rtk_qos_1pPriRemap_set(1,0);
rtk_qos_1pPriRemap_set(2,0);
rtk_qos_1pPriRemap_set(3,0);
rtk_qos_1pPriRemap_set(4,7);
rtk_qos_1pPriRemap_set(5,7);
rtk_qos_1pPriRemap_set(6,7);
rtk_qos_1pPriRemap_set(7,7);
```

---

int32 rtk_qos_dscpPriRemap_set(rtk_dscp_t dscp, rtk_pri_t int_pri)

This API can be used to set the translation table for mapping DSCP value in IP header to internal priority. If the coming frame is not an IP frame, the priority will be NULL when switch doing priority selection.

The range of *dscp* is 0 ~ 63 and the range of *priority* is 0~7.

Example:

```
/*set DSCP 10 mapping priority 1 and DSCP 20 mapping to 2*/
rtk_qos_dscpPriRemap_set(10, 1);
rtk_qos_dscpPriRemap_set(20, 2);
```

---

*int32 rtk_qos_priSel_set(rtk_qos_priDecTbl_t index, rtk_priority_select_t *pPriDec)*

ASIC supports user desired priority selection by the API ***rtk_qos_priSel_set.*** ASIC will select the highest priority assignment source priority to locate mapped queue number for frame transmitting.

There are 2 priority selection tables can be set and which are indexed as 0 & 1.

Example:

```
/* set priority decision table 0 with 802.1q  > dscp > port based > ACL >
SVLAN, the value is 0~4*/

rtk_priority_select_t priDec;
```

```
priDec.port_pri  = 2;
priDec.dot1q_pri = 4;
priDec.acl_pri   = 1;
priDec.dscp_pri  = 3;
priDec.svlan_pri = 0;
rtk_qos_priSel_set(0, &priDec);
```

---

int32 rtk_qos_portPriSelIndex_set(rtk_port_t port, rtk_qos_priDecTbl_t index)

This API can be used to configure which priority selection tabled is chosen.

Example:

```
/* Set UTP Port 0~4 using Priority Selection Table 0 */
/* Set EXT Port 0 using Priority Selection Table 1   */

rtk_qos_portPriSelIndex_set(UTP_PORT0, 0) ;
rtk_qos_portPriSelIndex_set(UTP_PORT1, 0) ;
rtk_qos_portPriSelIndex_set(UTP_PORT2, 0) ;
rtk_qos_portPriSelIndex_set(UTP_PORT3, 0) ;
rtk_qos_portPriSelIndex_set(UTP_PORT4, 0) ;
rtk_qos_portPriSelIndex_set(EXT_PORT0, 1) ;
```

---

*int32 rtk_qos_priMap_set(rtk_queue_num_t queue_num, rtk_qos_pri2queue_t *pPri2qid) ;*

Switch supports maximum 8 queues usage for each port and has priority mapping configuration. The API ***rtk_qos_priMap_set*** lets software to define queue id usage for internal priority. There are dedicated configurations for different queue number usage and there are different queue ids for variable queue number usage, too.

Example:

```
/*set priority 0,1 to queue 0, priority 2,3 to queue 1*/
/* priority 4,5 to queue 2 and priority 6,7 to queue 7 */

rtk_qos_pri2queue_t pri2qid ;

pri2qid.pri2queue[0] = 0 ;
pri2qid.pri2queue[1] = 0 ;
pri2qid.pri2queue[2] = 1 ;
pri2qid.pri2queue[3] = 1 ;
pri2qid.pri2queue[4] = 2 ;
pri2qid.pri2queue[5] = 2 ;
pri2qid.pri2queue[6] = 7 ;
pri2qid.pri2queue[7] = 7 ;

rtk_qos_priMap_set(4, &pri2qid) ;
```

---

int32 rtk_qos_schedulingType_set(rtk_qos_scheduling_type_t queueType)

This API can select scheduling type is WFQ or WRR.

Example:

```
/*set scheduling type to WFQ */
rtk_qos_schedulingType_set(RTK_QOS_WFQ);
```

---

*int32 rtk_qos_schedulingQueue_set(rtk_port_t port, rtk_qos_queue_weights_t *pQweights)*

While queue number usage is more than 1, software can configure each queue for strict or weight fair type by the API ***rtk_qos_schedulingQueue_set***. If weight is from 1 to 127, its queue type is weight fair queue, and if weight is 0, its queue type is strict priority.

Example:

```
/*set queue 5 as strict and queue 0,1,2 to WFQ in weight ratio 1,5,10 in
port 0 */
rtk_qos_queue_weights_t qweights;

memset(&qweights, 0x00, sizeof(rtk_qos_queue_weights_t));
qweights.weights[5] = 0;
qweights.weights[0] = 1;
qweights.weights[1] = 5;
qweights.weights[2] = 10;
rtk_qos_schedulingQueue_set(UTP_PORT0, &qweights);
```

4.6. CPU Port

Switch 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.

Table 5. CPU tag format

0			7 8				15
Realtek EtherType(16-bits) [0x8899]							
Protocol (8-bits) [0x4]				Reason (8-bits)			
FID_EN (1-bit)	Reserved (1-bit)	FID (2-bits)	Priority Select (1-bit)	Priority (3-bits)	Keep (1-bit)	Reserved (1-bit)	Disable Learning (1-bit)	Reserved (5-bits)
Allow	Port mask TX/SPA Rx(PortNo bits，LSB 4-bits)						

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 come 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.

Table 6. CPU tag fields

Realtek Ethertype	0x8899, indicate Realtek tag
Protocol	0x04, indicate this is an unmanaged switch CPU tag
Reason	The reason of forwarding this packet to CPU.
FID_EN	FID of this packet is from CPU tag FID field.
FID	FID
Priority Select	1: force priority of this packet, 0: don't force priority of this packet
Priority	Priority
Keep	The VLAN tag format of this packet will not be changed.
Disable Learning	The source MAC of this packet will not be learned

Allow	1: use TX/RX field as an allowance portmask, this packet can't be forward to any port which is not set in TX/RX field. 0: No allowance portmask
TX/RX	TX to CPU: indicate the receive port number of this packet RX from CPU: indicate the forwarding portmask or allowance portmask

int32 rtk_cpu_enable_set(rtk_enable_t enable)

The API can set CPU port function enable/disable.

Int32 rtk_cpu_tagPort_set(rtk_port_t port, rtk_cpu_insert_t mode)

With CPU port assignment, user can select ASIC inserting tag mode.

```
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 EXT Port 0 to CPU port without proprietary tag inserting but trap
packets*/

rtk_cpu_insert_t mode ;

mode = CPU_INSERT_TO_TRAPPING;
rtk_cpu_enable_set(ENABLED);
rtk_cpu_tagPort_set(EXT_PORT0, mode) ;
```

*int32 rtk_cpu_tagPort_get(rtk_port_t *pPort, rtk_cpu_insert_t *pMode)*

User could use this API to retrieve current CPU port setting including current CPU port number 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. Interrupt

ASIC has one dedicated GPIO pin for interrupt trigger to external CPU. The interrupt trigger might be pull pin voltage to high (3.3V) or low (0V) depend on interrupt polarity configuration. Event for interrupt trigging can be set as network link up happening in certain ports.

Int32 rtk_int_polarity_set (rtk_int_polarity_t type)

This API can be used to setup polarity configuration. There are 2 types of polarity: High Polarity and Low polarity. High Polarity means that the GPIO pin for interrupt will stay at low (0V) and be pulled to high (3.3V) when interrupt be triggered. Low Polarity means that the GPIO pin for interrupt will stay at high (3.3V) and be pulled to low (0V) when interrupt be triggered.

After the interrupt is triggered, the GPIO pin for interrupt will stays at high or low (depend on the polarity configuration) until the status of interrupt is retrieved.

Example:

```
/* set GPIO as high polarity, GPIO will stay low by default */

rtk_int_polarity_set(INT_POLAR_HIGH);
```

---

*int32 rtk_int_polarity_get (rtk_int_polarity_t *pType)*

This API can be used to get current polarity configuration.

Int32 rtk_int_control_set(rtk_int_type_t type, rtk_enable_t enable)

This API can be used to configure interrupt setting. All supported interrupt types are listed as following and are disabled by default. Without enabling any interrupt, the GPIO pin for interrupt will not be pulled high or low.

```
Typedef enum rtk_int_type_e
{
    INT_TYPE_LINK_STATUS = 0,
    INT_TYPE_METER_EXCEED,
    INT_TYPE_LEARN_LIMIT,
    INT_TYPE_LINK_SPEED,
    INT_TYPE_CONGEST,
    INT_TYPE_GREEN_FEATURE,
    INT_TYPE_LOOP_DETECT,
    INT_TYPE_8051,
    INT_TYPE_CABLE_DIAG,
    INT_TYPE_ACL,
    INT_TYPE_RESERVED, /* Unused */
    INT_TYPE_SLIENT,
    INT_TYPE_END
}rtk_int_type_t;
```

Example:

```
/* set GPIO pull high trigger with link state changed*/
/* Other interrupts are disabled. */

rtk_int_polarity_set(INT_POLAR_HIGH);
rtk_int_control_set(INT_TYPE_LINK_STATUS, ENABLED);
rtk_int_control_set(INT_TYPE_LINK_SPEED,DISABLED);
rtk_int_control_set(INT_TYPE_LOOP_DETECT, DISABLED);
rtk_int_control_set(INT_TYPE_METER_EXCEED, DISABLED);
rtk_int_control_set(INT_TYPE_LEARN_LIMIT, DISABLED);
```

*int32 rtk_int_control_get(rtk_int_type_t type, rtk_enable_t *pEnable)*

This API can be used to get interrupt setting.

Example:

```
/* Get Link change interrupt state. */
rtk_enable_t state;
```

```
rtk_int_control_get(INT_TYPE_LINK_STATUS, &state);
```

int32 rtk_int_status_get(rtk_int_status_t pStatusMask)*

ASIC will used dedicated GPIO to trigger external CPU as interrupt control setting and CPU can get interrupt status to check which event to trigger current interrupt. This API can be used to get the status. The voltage of interrupt GPIO pin will not return to normal state if users don't clear it.

```
Typedef struct  rtk_int_status_s
{
    rtk_uint16 value[RTK_MAX_NUM_OF_INTERRUPT_TYPE];
} rtk_int_status_t;

The interrupt trigger status is shown in the following:
      - INT_TYPE_LINK_STATUS     (value[0] (Bit0))
      - INT_TYPE_METER_EXCEED    (value[0] (Bit1))
      - INT_TYPE_LEARN_LIMIT     (value[0] (Bit2))
      - INT_TYPE_LINK_SPEED      (value[0] (Bit3))
      - INT_TYPE_CONGEST         (value[0] (Bit4))
      - INT_TYPE_GREEN_FEATURE   (value[0] (Bit5))
      - INT_TYPE_LOOP_DETECT     (value[0] (Bit6))
      - INT_TYPE_8051            (value[0] (Bit7))
      - INT_TYPE_CABLE_DIAG      (value[0] (Bit8))
      - INT_TYPE_ACL             (value[0] (Bit9))
      - INT_TYPE_SLIENT          (value[0] (Bit11))
```

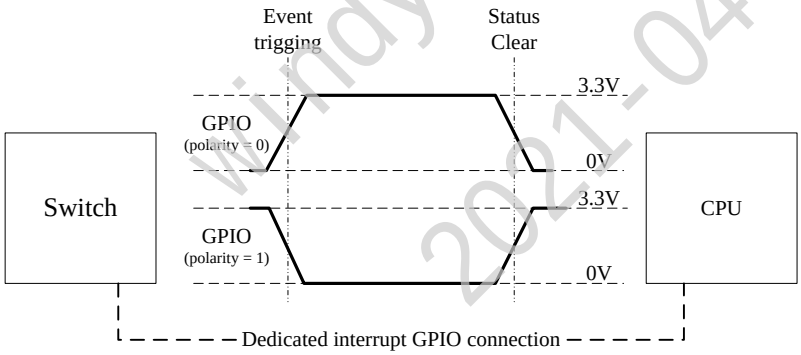

Figure 6. Interrupt Polarity

Example:

```
/*Get interrupt status after GPIO trigger to see if LINK STATUS interrupt
is triggered*/

rtk_int_status_t  statusMask;

rtk_int_status_get(&statusMask);
if(statusMask.value[0] & 0x01)
{
    /* LINK STATUS interrupt is triggered */
}
```

int32 rtk_int_status_set(rtk_int_status_t pStatusMask)*

This API is used to clean interrupt status. If external CPU would not use this API to clear interrupt status, or following network linking events will not be triggered any more.

```
The interrupt trigger status is shown in the following:
      - INT_TYPE_LINK_STATUS     (value[0] (Bit0))
      - INT_TYPE_METER_EXCEED    (value[0] (Bit1))
      - INT_TYPE_LEARN_LIMIT     (value[0] (Bit2))
      - INT_TYPE_LINK_SPEED      (value[0] (Bit3))
      - INT_TYPE_CONGEST         (value[0] (Bit4))
      - INT_TYPE_GREEN_FEATURE   (value[0] (Bit5))
      - INT_TYPE_LOOP_DETECT     (value[0] (Bit6))
      - INT_TYPE_8051            (value[0] (Bit7))
      - INT_TYPE_CABLE_DIAG      (value[0] (Bit8))
      - INT_TYPE_ACL             (value[0] (Bit9))
      - INT_TYPE_SLIENT          (value[0] (Bit11))
```

Example:

```
/* Clear interrupt status of link status & learnling limit */
rtk_int_status_t  statusMask;
statusMask.value[0]= 0x5; /* Bit 0 and Bit 2 */

rtk_int_status_set(&statusMask);
```

*int32 rtk_int_advanceInfo_get(rtk_int_advType_t adv_type, rtk_int_info_t *pInfo)*

This API is used to get and clear advanced information. The supported types are listed as following. Users can specify a type and this API will return the advanced information in parameter “info”.

```
Typedef enum rtk_int_advType_e
{
    ADV_L2_LEARN_PORT_MASK = 0,
    ADV_SPEED_CHANGE_PORT_MASK,
    ADV_SPECIAL_CONGESTION_PORT_MASK,
    ADV_PORT_LINKDOWN_PORT_MASK,
    ADV_PORT_LINKUP_PORT_MASK,
    ADV_METER_EXCEED_MASK,
    ADV_RLDP_LOOPED,
    ADV_RLDP_RELEASED,
    ADV_END,
} rtk_int_advType_t;
```

Example:

```
/* set GPIO pull high trigger with link state changed*/

rtk_int_polarity_set(INT_POLAR_HIGH);
rtk_int_control_set(INT_TYPE_LINK_STATUS, ENABLED);

/* Monitor the event of port 0 is link up. */
rtk_int_status_t    statusmask;
rtk_int_info_t      info ;

rtk_int_status_get(&statusmask);
if(statusmask.value[0] & (0x0001 << INT_TYPE_LINK_STATUS))
{
    /* Clear status */
    statusmask.value[0] = (0x0001 << INT_TYPE_LINK_STATUS);
    rtk_int_status_set(&statusmask);
```

```
    /* Get advanced information */
    rtk_int_advanceInfo_get(ADV_PORT_LINKUP_PORT_MASK, &info);
    if(RTK_PORTMASK_IS_PORT_SET(info.portMask, UTP_PORT0))
    {
        /* Port 0 is link up */
    }
}
```

4.8. MIB

Switch supports MIB counters include MIB-II (RFC 1213), Ethernet-like MIB (RFC 3635), Interface Group MIB (RFC 2863), RMON (RFC 2819), Bridge MIB (RFC1493) and Bridge Extension (RFC 2674). There are 3 group MIB counters as IN counters, OUT counters and whole system counters.

---

*Int32 rtk_stat_port_get(rtk_port_t port, rtk_stat_port_type_t cntr_idx, rtk_stat_counter_t *pCntr)*

This API will return failed in the period of MIB retrieving processing and MIB resetting. Mapping index of MIB counter is defined as following description.

Table 7. Statistic List

Field	Description
ifInOctets	The total number of octets received on the interface， including framing characters.
Dot3StatsFCSErrors	Number of received frames with FCS errors.
Dot3StatsSymbolErrors	Number of data symbol errors.
Dot3InPauseFrames	Number of received PAUSE frames.
dot3ControlInUnknownOpcodes	Number of received MAC control frames with an unknown opcode.
etherStatsFragments	Number of received **invalid (FCS error / alignment error)** packets whose size are less than 64 bytes， including MAC header and FCS， excluding preamble and SFD.
etherStatsJabbers	Number of received **invalid (FCS error / alignment error)** packets whose size are more than 1518 bytes， including MAC header and FCS， excluding preamble and SFD.
ifInUcastPkts	Number of received **valid** unicast packets.
etherStatsDropEvents	Number of received packets dropped due to lack of resource.
etherStatsOctets	number of bytes in **all** received frames， including MAC header and FCS， excluding preamble and SFD.
etherStatsUnderSizePkts	Number of received **valid** packets whose size are less than 64 bytes， including MAC header and FCS， excluding preamble and SFD.
etherOversizeStats	Number of received **valid** packets whose size are more than 1518 bytes， including MAC header and FCS， excluding preamble and SFD.
etherStatsPkts64Octets	Number of **all** received packets whose size are exactly 64 bytes， including MAC header and FCS， excluding

	preamble and SFD.
etherStatsPkts65to127Octets	Number of **all** received packets whose size are between 65 ~ 127 bytes， including MAC header and FCS， excluding preamble and SFD.
etherStatsPkts128to255Octets	Number of **all** received packets whose size are between 128 ~ 255 bytes， including MAC header and FCS， excluding preamble and SFD.
etherStatsPkts256to511Octets	Number of **all** received packets whose size are between 256 ~ 511 bytes， including FCS， excluding MAC header， preamble and SFD.
etherStatsPkts512to1023Octets	Number of **all** received packets whose size are between 512 ~ 1023 bytes， including MAC header and FCS， excluding preamble and SFD.
etherStatsPkts1024to1518Octets	Number of **all** received packets whose size are between 1024 ~ 1518 bytes， including MAC header and FCS， excluding preamble and SFD.
etherStatsMulticastPkts	The number of packets， delivered by this sub-layer to a higher (sub-)layer， which were addressed to a multicast address at this sub-layer. For a MAC layer protocol， this includes both Group and Functional addresses.
etherStatsBroadcastPkts	The number of packets， delivered by this sub-layer to a higher (sub-)layer， which were addressed to a broadcast address at this sub-layer.
ifOutOctets	The total number of octets transmitted out of the interface， including framing characters.
Dot3StatsSingleCollisionFrames	A count of frames that are involved in a single collision， and are subsequently transmitted successfully.
Dot3StatMultipleCollisionFrames	A count of frames that are involved in more than one collision and are subsequently transmitted successfully.
Dot3StatsDeferredTransmissions	A count of frames for which the first transmission attempt on a particular interface is delayed because the medium is busy.
Dot3StatsLateCollisions	The number of times that a collision is detected on a particular interface later than one slotTime into the transmission of a packet.
etherStatsCollisions	Estimate of number of collisions on this thernet segment.
Dot3StatsExcessiveCollisions	A count of frames for which transmission on a particular interface fails due to excessive collisions.
Dot3OutPauseFrames	A count of MAC control frames transmitted on this interface with an opcode indicating the PAUSE operation.
PortDelayExceededDiscards	Not Support
dot1dTpPortInDiscards	Number of received **valid** frames discarded by Forwarding Process.
ifOutUcastPkts	Number of transmitted unicast packets， including discarded or not sent.
ifOutMulticastPkts	Number of transmitted multicast packets， including discarded or not sent.
ifOutBroadcastPkts	Number of transmitted broadcast packets， including discarded or not sent.
OutOampduPkts	A count of OAMPDUs transmitted on this interface.
InOampduPkts	Number of received OAMPDUs.
PktgenPkts	Not Support

InMldChecksumError	The number of received MLD control packet which checksum is fail
InIgmpChecksumError	The number of received IGMP control packets which checksum is fail
InMldSpecificQuery	The number of received MLD Group-Specific query packets
InMldGeneralQuery	The number of received MLD General Query packets
InIgmpSpecificQuery	The number of received IGMP Group-Specific query packets
InIgmpGeneralQuery	The number of received IGMP General Query packets
InMldLeave	The number of received MLD Done packets
InIgmpLeaves	The number of received IGMP Leave packets
InIgmpJoinsSuccess	The number of received IGMP Report packets which join a group in ASIC successfully
InIgmpJoinsFail	The number of received IGMP Report packets which join a group in ASIC unsuccessfully
InMldJoinsSuccess	The number of received MLD Report packets which join a group in ASIC successfully
InMldJoinsFail	The number of received MLD Report packets which join a group in ASIC unsuccessfully
InReportSuppressionDrop	The number of received Report packets dropped due to report suppression
InLeaveSuppressionDrop	The number of received Leave/ Done packets dropped due to report suppression
OutIgmpReport	The number of transmitted IGMP Report packets
OutIgmpLeaves	The number of transmitted IGMP Leave packets
OutIgmpGeneralQuery	The number of transmitted IGMP General Query packets
OutIgmpSpecificQuery	The number of transmitted IGMP Group-Specific query packets
OutMldReport	The number of transmitted MLD Report packets
OutMldLeaves	The number of transmitted MLD Leave packets
OutMldGeneralQuery	The number of transmitted MLD General Query packets
OutMldSpecificQuery	The number of transmitted MLD Group-Specific query packets
InKnownMulticastPkts	The number of received known multicast packets
ifInMulticastPkts	Number of received **valid** multicast packets.
ifInBroadcastPkts	Number of received **valid** broadcast packets.

Example:

```
/*get number of received valid unicast octets of port 0 */
rtk_stat_counter_t cntr;
rtk_stat_port_get(UTP_PORT0, STAT_IfInOctets, &cntr);
```

int32 rtk_stat_port_reset(rtk_port_t port)

There are MIB counters of network traffic information belong to each port and belong to whole system. ASIC will reset all counters belong to the port.

Example:

```
/*reset MIB counters of port 0 and port 4*/
rtk_stat_port_reset(UTP_PORT0) ;
rtk_stat_port_reset(UTP_PORT4) ;
```

4.9. PHY

ASIC supports 1000Base-T PHYs and allows network management to configure desired capability. Auto-Negotiation function allows a device to advertise modes (1000F, 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, there are only 10/100 modes (100F, 100H, 10F and 10H modes). While 1000F is enabled, both abilities 100F and 10F will be supported.

Int32 rtk_port_phyEnableAll_set(rtk_enable_t enable)

This API is used to enable all PHYs if the default PHY status is disabled by pin strap.

Example:

```
/*All PHYs are disabled by hardware pin, use API to enable all PHYs  */
rtk_port_phyEnableAll_set(ENABLED);
```

---

*int32 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
{
    rtk_uint32    AutoNegotiation;
/*PHY register 0.12 setting for auto-negotiation process*/
    rtk_uint32    Half_10;
/*PHY register 4.5 setting for 10BASE-TX half duplex capable*/
    rtk_uint32    Full_10;
/*PHY register 4.6 setting for 10BASE-TX full duplex capable*/
    rtk_uint32    Half_100;
/*PHY register 4.7 setting for 100BASE-TX half duplex capable*/
    rtk_uint32    Full_100;
 /*PHY register 4.8 setting for 100BASE-TX full duplex capable*/
```

```
    rtk_uint32    Full_1000;
/*PHY register 9.9 setting for 1000BASE-T full duplex capable*/
    rtk_uint32    FC;
/*PHY register 4.10 setting for flow control capability*/
    rtk_uint32    AsyFC;
/*PHY register 4.11 setting for asymmetric flow control capability*/
} rtk_port_phy_ability_t;
```

If 1000F capability bit is set to 1, API will also configure Auto-Negotiation register field bit enabled. FC and AsyFC fields are defined as PAUSE bit and ASM_DIR bit as Std 802.3 annex 28B. The encoding of Bit FC and AsyFC is specified as following table.

Table 8. Capability of Flow Control

FC	AsyFC	Capability
0	0	No pause flow control
0	1	Asymmetric pause toward link partner
1	0	Symmetric pause flow control
1	1	Both symmetric pause and asymmetric pause toward local device

Example:

```
/* set PHY 1 with Auto negotiation,1000F,100F*/
/* and Symmetric PAUSE flow control capabilities*/

rtk_port_phy_ability_t  ability;
memset(&ability, 0x00, sizeof(rtk_port_phy_ability_t));
ability.Full_1000 = 1;
ability.FC = 1;
ability.AsyFC = 1;
rtk_port_phyAutoNegoAbility_set(UTP_PORT1, &ability);

/*set PHY 2 with Auto negotiation,100F without flow control*/
rtk_port_phy_ability_t  ability;
memset(&ability, 0x00, sizeof(rtk_port_phy_ability_t));
ability.Full_100 = 1;
rtk_port_phyAutoNegoAbility_set(UTP_PORT2, &ability);
```

*int32 rtk_port_phyAutoNegoAbility_get(rtk_port_t port, rtk_port_phy_ability_t *pAbility)*

Get the capability of specified PHY.

Example:

```
/*Get capability of PHY 1 */
rtk_port_phyAutoNegoAbility_get(UTP_PORT1, &ablility);
```

*int32 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_2500M,
    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 UTP Port 1 */
rtk_port_linkStatus_t linkStatus ;
rtk_port_speed_t speed ;
rtk_port_duplex_t duplex ;

rtk_port_phyStatus_get(UTP_PORT1, &linkStatus, &speed, &duplex);
```

4.10. LED

rtk_api_ret_t rtk_led_OutputEnable_set(rtk_enable_t state)

This API should be called before setup LED.

Example:

```
/* enable LED output function */
rtk_led_OutputEnable_set(ENABLED);
```

int32 rtk_led_enable_set(rtk_led_group_t group, rtk_portmask_t portmask)

This API enables LED by group and port, and user can configure LED by the API according to the hardware layout.

Example:

```
/*The hardware layout only use group 0 & group 1, and the port is Port 0
~ Port 4*/
rtk_portmask_t portmask;

RTK_PORTMASK_PORT_SET(portmask, UTP_PORT0);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT1);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT2);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT3);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT4);

rtk_led_enable_set(LED_GROUP_0, &portmask);
rtk_led_enable_set(LED_GROUP_1, &portmask);
```

int32 rtk_led_operation_set(rtk_led_operation_t mode)

This API can set LED operation mode, and normally the mode is parallel mode.

```
Typedef enum rtk_led_operation_e
{
    LED_OP_SCAN=0,
    LED_OP_PARALLEL,
    LED_OP_SERIAL,
    LED_OP_END,
}rtk_led_operation_t;
```

Example:

```
/*Set LED to parallel mode*/
rtk_led_operation_set(LED_OP_PARALLEL);
```

int32 rtk_led_blinkRate_set(rtk_led_blink_rate_t blinkRate)

```
typedef enum rtk_led_blink_rate_e
{
    LED_BLINKRATE_32MS=0,
    LED_BLINKRATE_64MS,
    LED_BLINKRATE_128MS,
    LED_BLINKRATE_256MS,
    LED_BLINKRATE_512MS,
    LED_BLINKRATE_1024MS,
    LED_BLINKRATE_48MS,
    LED_BLINKRATE_96MS,
    LED_BLINKRATE_END,
}rtk_led_blink_rate_t;
```

Example:

```
/* Set blinking rate to 256ms*/

rtk_led_blinkRate_set(LED_BLINKRATE_256MS);
```

int32 rtk_led_groupConfig_set(rtk_led_group_t group, rtk_led_congig_t config)

This API is used to configure per group LED indicated mode. If the applied LED indicate mode can not satisfy user's application, they can use this API to configure LED indicate mode per group. The definition is as following:

```
typedef enum rtk_led_config_e
{
    LED_CONFIG_LEDOFF=0,
    LED_CONFIG_DUPCOL,
    LED_CONFIG_LINK_ACT,
    LED_CONFIG_SPD1000,
    LED_CONFIG_SPD100,
    LED_CONFIG_SPD10,
    LED_CONFIG_SPD1000ACT,
    LED_CONFIG_SPD100ACT,
    LED_CONFIG_SPD10ACT,
    LED_CONFIG_SPD10010ACT,
    LED_CONFIG_LOOPDETECT,
    LED_CONFIG_EEE,
    LED_CONFIG_LINKRX,
    LED_CONFIG_LINKTX,
    LED_CONFIG_MASTER,
    LED_CONFIG_END,
}rtk_led_congig_t;
```

Example:

```
/*Set LED mode group 0 to LED_CONFIG_LINK_ACT */
rtk_led_groupConfig_set(LED_GROUP_0, LED_CONFIG_LINK_ACT);
```

4.11. RMA

For IEEE 802.1 standard, ASIC supports RMA (Reserved Multicast Address) function for trapping incoming management frames to CPU if high layer applications need to retrieve information in such frames. Following table is list of relevant layer 2 protocol mapping reserved MAC address.

Table 9. Reserved Multicast Address

Protocols description	Destination MAC Address
Bridge Group Address	01-80-C2-00-00-00
IEEE Std 802.3, 1988 Edition, Full Duplex PAUSE operation	01-80-C2-00-00-01
IEEE Std 802.3ad Slow Protocols-Multicast address	01-80-C2-00-00-02
IEEE Std 802.1X PAE address	01-80-C2-00-00-03
Reserved	01-80-C2-00-00-04
Reserved	01-80-C2-00-00-05
Reserved	01-80-C2-00-00-06
Reserved	01-80-C2-00-00-07
Provider Bridge Group Address	01-80-C2-00-00-08
Reserved	01-80-C2-00-00-09
Reserved	01-80-C2-00-00-0A
Reserved	01-80-C2-00-00-0B
Reserved	01-80-C2-00-00-0C
Provider Bridge GVRP Address	01-80-C2-00-00-0D
IEEE Std. 802.1AB Link Layer Discovery Protocol multicast address	01-80-C2-00-00-0E
Reserved	01-80-C2-00-00-0F
All LANs Bridge Management Group Address	01-80-C2-00-00-10
Load Server Generic Address	01-80-C2-00-00-11
Loadable Device Generic Address	01-80-C2-00-00-12
Reserved	01-80-C2-00-00-13
Reserved	01-80-C2-00-00-14
Reserved	01-80-C2-00-00-15
Reserved	01-80-C2-00-00-16
Reserved	01-80-C2-00-00-17
Generic Address for All Manager Stations	01-80-C2-00-00-18
Reserved	01-80-C2-00-00-19
Generic Address for All Agent Stations	01-80-C2-00-00-1A
Reserved	01-80-C2-00-00-1B
Reserved	01-80-C2-00-00-1C
Reserved	01-80-C2-00-00-1D
Reserved	01-80-C2-00-00-1E
Reserved	01-80-C2-00-00-1F
GMRP Address	01-80-C2-00-00-20
GVRP address	01-80-C2-00-00-21
Undefined GARP address	01-80-C2-00-00-22
Undefined GARP address	01-80-C2-00-00-23
Undefined GARP address	01-80-C2-00-00-24

Undefined GARP address	01-80-C2-00-00-25
Undefined GARP address	01-80-C2-00-00-26
Undefined GARP address	01-80-C2-00-00-27
Undefined GARP address	01-80-C2-00-00-28
Undefined GARP address	01-80-C2-00-00-29
Undefined GARP address	01-80-C2-00-00-2A
Undefined GARP address	01-80-C2-00-00-2B
Undefined GARP address	01-80-C2-00-00-2C
Undefined GARP address	01-80-C2-00-00-2D
Undefined GARP address	01-80-C2-00-00-2E
Undefined GARP address	01-80-C2-00-00-2F
CDP	01-00-0C-CC-CC-CC
CSSTP	01-00-0C-CC-CC-CD
LLDP	Ethertype = 0x88CC & 01-80-C2-00-00-00 or 01-80-C2-00-00-03 or 01-80-C2-00-00-0e

*int32 rtk_trap_rmaAction_set(rtk_mac_t *pRma_frame, rtk_trap_rma_action_t rma_action)*

```
typedef enum rtk_trap_rma_action_e
{
    RMA_ACTION_FORWARD = 0,
    RMA_ACTION_TRAP2CPU,
    RMA_ACTION_DROP,
    RMA_ACTION_FORWARD_EXCLUDE_CPU,
    RMA_ACTION_END
} rtk_trap_rma_action_t;
```

Example:

```
/*set trapping both STP protocol frame and LACP frame to CPU port 4*/
rtk_mac_t rma_frame ;
rtk_trap_rma_action_t rma_action ;

rtk_cpu_enable_set(ENABLE);
rtk_cpu_tagPort_set(UTP_PORT4, CPU_INSERT_TO_NONE);
```

```
rma_frame.octec[0] = 0x01;
rma_frame.octec[1] = 0x80;
rma_frame.octec[2] = 0xC2;
rma_frame.octec[3] = 0x00 ;
rma_frame.octec[4] = 0x00 ;
rma_frame.octec[5] = 0x00 ;
rma_action = RMA_ACTION_TRAP2CPU ;
rtk_trap_rmaAction_set(&rma_frame, rma_action) ;

rma_frame.octec[0] = 0x01 ;
rma_frame.octec[1] = 0x80 ;
rma_frame.octec[2] = 0xC2 ;
rma_frame.octec[3] = 0x00 ;
rma_frame.octec[4] = 0x00 ;
rma_frame.octec[5] = 0x02 ;

rtk_trap_rmaAction_set(&rma_frame, rma_action);
```

---

*int32 rtk_trap_rmaAction_get(rtk_mac_t *pRma_frame, rtk_trap_rma_action_t *pRma_action)*

Example:

```
/*Get STP protocol frame action*/
rtk_mac_t rma_frame ;
rtk_trap_rma_action_t rma_action ;

rma_frame.octec[0] = 0x01 ;
rma_frame.octec[1] = 0x80 ;
rma_frame.octec[2] = 0xC2 ;
rma_frame.octec[3] = 0x00 ;
rma_frame.octec[4] = 0x00 ;
rma_frame.octec[5] = 0x00 ;
rtk_trap_rmaAction_get(&rma_frame, &rma_action);
```

4.12. Storm Filtering Control

ASIC supports four kinds of packet storm control mechanism broadcasting storm control, multicasting storm control, unknown multicast storm control and unknown destination address storm control. Each port has dedicated setting and storm flow packet counter. It can avoid undesired network flow taking too mush bandwidth of switch and improve more bandwidth utility for network.

Int32 rtk_rate_stormControlPortEnable_set(rtk_port_t port, rtk_rate_storm_group_t stormType, rtk_enable_t enable)

```
typedef enum rtk_rate_storm_group_e
{
    STORM_GROUP_UNKNOWN_UNICAST = 0,
    STORM_GROUP_UNKNOWN_MULTICAST,
    STORM_GROUP_MULTICAST,
    STORM_GROUP_BROADCAST,
    STORM_GROUP_END
} rtk_rate_storm_group_t;
```

Example:

```
/* Enable Strom control on port 2 for all kinds of Storm Filtering.*/

rtk_rate_stormControlPortEnable_set(UTP_PORT2,STORM_GROUP_UNKNOWN_UNICAST,
ENABLED);
rtk_rate_stormControlPortEnable_set(UTP_PORT2,STORM_GROUP_UNKNOWN_MULTICAST,
ENABLED);
rtk_rate_stormControlPortEnable_set(UTP_PORT2,STORM_GROUP_MULTICAST,
ENABLED);
rtk_rate_stormControlPortEnable_set(UTP_PORT2,STORM_GROUP_BROADCAST,
ENABLED);
```

---

*int32 rtk_rate_stormControlPortEnable_get(rtk_port_t port, rtk_rate_storm_group_t stormType, rtk_enable_t *pEnable)*

User can use this API to get current Storm Control state of a port.

Example:

```
/* Get state of broadcast Strom control on port 2 */
ttk_enable_t state;

rtk_rate_stormControlPortEnable_set(UTP_PORT2,STORM_GROUP_BROADCAST,
&state);
```

---

int32 rtk_rate_stormControlMeterIdx_set(rtk_port_t port, rtk_rate_storm_group_t stormType, rtk_uint32 index)

User can use this API to set Storm Control shared meter index of a port.

Example:

```
/* set shared meter of broadcast Strom control on port 2 to meter 10 */

rtk_rate_stormControlMeterIdx_set(UTP_PORT2, STORM_GROUP_BROADCAST, 10);
```

---

int32 rtk_rate_stormControlMeterIdx_get(rtk_port_t port, rtk_rate_storm_group_t stormType, rtk_uint32 index)

User can use this API to get Storm Control shared meter index of a port.

Example:

```
/* get shared meter of broadcast Strom control on port 2 */
```

```
rtk_uint32 index;

rtk_rate_stormControlMeterIdx_get(UTP_PORT2, STORM_GROUP_BROADCAST,
&index);
```

4.13. Port Mirror

ASIC supports one-port mirror function for network monitoring. User needs to set desired source port for frames monitoring and monitor port for frames are mirrored to.

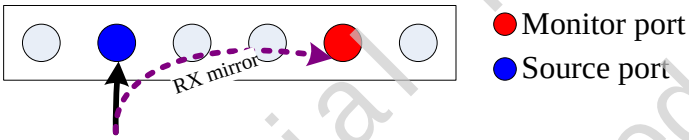

Figure 7. RX Mirror

There are two types of mirror function usage can be set. One is source port receiving frame from mirror function. Another mirror function is source port that frame transmitted to.

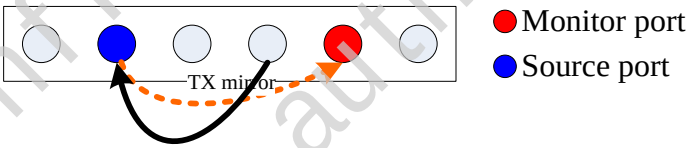

Figure 8. TX Mirror

If user wants to prevent monitoring frame loss issue in monitor port, then the traffic isolation on monitor port function can be set. This function will permit frames transmitted from source port only at monitor port.

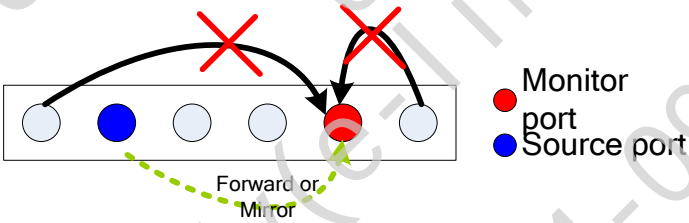

Figure 9. Mirror Isolation

---

int32 rtk_mirror_portBased_set(rtk_port_t mirroring_port, rtk_portmask_t *pMirrored_rx_portmask, rtk_portmask_t *pMirrored_tx_portmask)

Example:

```
/*set port 1 to monitor port 0 transmitting frames and let port 1 for
network management only with port isolation setting*/
rtk_portmask_t rx_portmask;
rtk_portmask_t tx_portmask;

rtk_mirror_portIso_set(ENABLE);
RTK_PORTMASK_CLEAR(rx_portmask);
RTK_PORTMASK_PORT_SET(tx_portmask, UTP_PORT0);
rtk_mirror_portBased_set(UTP_PORT1, &rx_portmask, &tx_portmask);
```

4.14. Force External Interface

Switch provides API to set external interface to a specified mode. This function can be used for limiting the ability of external interface.

User can use *rtk_port_macForceLinkExt_set* to configure switch. This API uses logical port number as first parameter. Only extension port number is acceptable.

---

*Int32 rtk_port_macForceLinkExt_set(rtk_port_t port, rtk_mode_ext_t mode, rtk_port_mac_ability_t *pPortability)*

Example:

```
/* Set extension port 0 to SGMII with Force mode, 1000M, Full-duplex,
enable TX&RX pause*/
rtk_port_mac_ability_t mac_cfg ;
rtk_mode_ext_t mode ;

mode = MODE_EXT_SGMII;
mac_cfg.forcemode = MAC_FORCE;
mac_cfg.speed = PORT_SPEED_1000M;
mac_cfg.duplex = PORT_FULL_DUPLEX;
mac_cfg.link = PORT_LINKUP;
mac_cfg.nway = DISABLED;
mac_cfg.txpause = ENABLED;
mac_cfg.rxpause = ENABLED;

rtk_port_macForceLinkExt_set(EXT_PORT0, mode,&mac_cfg);
```

---

*int32 rtk_port_macForceLinkExt_get(rtk_port_t port, rtk_mode_ext_t *pMode, rtk_port_mac_ability_t *pPortability)*

Example:

```
/* Get extension port 0 Configuration*/
rtk_mode_ext_t mode ;
rtk_port_mac_ability_t mac_cfg mac_cfg;

rtk_port_macForceLinkExt_get(EXT_PORT0, &mode, &mac_cfg);
```

---

int32 rtk_port_rgmiiDelayExt_set(rtk_port_t port, rtk_data_t txDelay, rtk_data_t rxDelay)

The API is used to set RGMII interface TX and RX delay. In TX delay, the value 1 means delay 2ns and 0 means no delay. In RX delay, there are 8 steps to tune the delay status.

Note. This API should be called before rtk_port_macForceLinkExt_set().

Example:

```
/* Set RGMII Interface 0 TX delay to 2ns and RX to step 4 */
    rtk_port_rgmiiDelayExt_set(EXT_PORT0, 1, 4);
```

---

*int32 rtk_port_rgmiiDelayExt_get(rtk_port_t port, rtk_data_t *pTxDelay, rtk_data_t *pRxDelay)*

The API is used to get RGMII interface TX and RX delay.

Example:

```
/* Set RGMII Interface 0 TX delay to 2ns and RX to step 4 */
rtk_data_t txDelay, rxDelay;
rtk_port_rgmiiDelayExt_get(EXT_PORT0, &txDelay, &rxDelay);
```

4.15. Port Isolation

Switch supports Port Isolation function. Users can use API *rtk_port_isolation_set* to set a forwarding port mask for each source port. The forwarding port mask is configured per port. Every packet switched by switch can't be forwarded to the ports which are not in the 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. Besides this, Port Isolation is the highest priority in forwarding decision.

---

*Int32 rtk_port_isolation_set(rtk_port_t port, rtk_portmask_t *pPortmask)*

Example:

```
/* Set Port Isolation function on UTP Port 4 and set its forwarding port
mask to Port 0~3*/
rtk_portmask_t portmask;

RTK_PORTMASK_CLEAR(portmask);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT0);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT1);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT2);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT3);

rtk_port_isolation_set(UTP_PORT4, &portmask) ;
```

---

*int32 rtk_port_isolation_get(rtk_port_t port, rtk_portmask_t *pPortmask)*

This API can be used to get current Port Isolation configuration on a specified port.

Example:

```
/*Get Port Isolation configuration on UTP Port 4 */
rtk_portmask_t portmask;
```

```
rtk_port_isolation_get(UTP_PORT4, &portmask);
```

4.16. MAC Learning Limit

Switch supports source MAC learning limit function. This function can be configured through API "***rtk_l2_limitLearningCnt_set***". API "***rtk_l2_limitLearningCnt_get***" can be used to get current source MAC learning configuration. API "***rtk_l2_learningCnt_get***" can be used to get current source MAC learning number. Default action for overflow MAC addresses is forwarding.

Int32 rtk_l2_limitLearningCnt_set(rtk_port_t port, rtk_mac_cnt_t mac_cnt)

Example:

```
/*Set MAC learning limit on Port 2 to 100 MAC addresses*/
rtk_l2_limitLearningCnt_set(UTP_PORT2, 100);
```

---

*int32 rtk_l2_limitLearningCnt_get(rtk_port_t port, rtk_mac_cnt_t *pMac_cnt)*

Example:

```
/*Get current MAC learning limit configuration on Port 2 */
rtk_mac_cnt_t mac_cnt ;

rtk_l2_limitLearningCnt_get(UTP_PORT2, &mac_cnt);
```

---

*int32 rtk_l2_learningCnt_get(rtk_port_t port, rtk_mac_cnt_t *pMac_cnt)*

Example:

```
/*Get current MAC learning  counter on Port 2 */
rtk_mac_cnt_t mac_cnt ;

rtk_l2_learningCnt_get(UTP_PORT2, &mac_cnt);
```

4.17. ACL

Switch has 64 shared ACL rules for system and each port claim its own ACL rules. Each ACL rule contains 144 bits and has four main components: 128 bits data fields, frame type & tag existed indicators, ACL rule template ID and ingress port mask.

Table 10. ACL Rule Format

143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112
Field 7																Field 6															
111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
Field 5																Field 4															
79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
Field 3																Field 2															
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field 1																Field 0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																
Active Portmask[7:0]								Frame Type & Tag								Template 0-4															

Template ID field in ACL rule indicate that which template should be used to decode these 8 fields in rule. There are 22 types of format including L2/L3/L4 header. These formats are referenced to the meaning of data fields. The below table is the rule types.

Table 11. ACL Template Type

Type	Name	Bits of Value
0x01	DMAC0	DMAC[15:0]
0x02	DMAC1	DMAC[31:16]
0x03	DMAC2	DMAC[47:32]
0x04	SMAC0	SMAC[15:0]
0x05	SMAC1	SMAC[31:16]
0x06	SMAC2	SMAC[47:32]
0x07	ETHERTYPE	Type/Length
0x08	STAG	SPRI{15:13}+DEI{12}+SVID{11:0}
0x09	CTAG	CPRI{15:13}+CFI{12}+CVID{11:0}
0x10	IP4SIP0	Ipv4 SIP[15:0]
0x11	IP4SIP1	Ipv4 SIP[31:16]
0x12	IP4DIP0	Ipv4 DIP[15:0]
0x13	IP4DIP1	Ipv4 DIP[31:16]
0x20	IP6SIP0	Ipv6 SIP[15:0]
0x21	IP6SIP1	Ipv6 SIP[31:16]
0x28	IP6DIP0	Ipv6 DIP[15:0]
0x29	IP6DIP1	Ipv6 DIP[31:16]
0x2a	TCP/UDP destination port	L4 destination port
0x2b	TCP/UDP source port	L4 source port

0x31	IPRANGE	Ipv4/Ipv6 Range Check Mask
0x33	FIELD_VALID	Field selectors valid tag
0x4n	FIELD_SELECTORn	User defined 16-bits field

For example, if template 0 is configured like below:

Table 12. ACL Template Configuration Example

Field_0	Field_1	Field_2	Field_3	Field_4	Field_5	Field_6	Field_7
DMAC0	DMAC1	DMAC2	SMAC0	SMAC1	SMAC2	CTAG	STAG

The data field in ACL rule will be decoded as following

- Field 0 ~ 2: MAC_DA
- Field 3 ~ 5: MAC_SA
- Field 6: CTAG
- Field 7: STAG

Switch support 5 independent templates. User should configure these templates before adding rules. By default, after initialization, the 5 templates would be reset to the following combination:

Table 13. Default ACL Template Configuration

	Field_0	Field_1	Field_2	Field_3	Field_4	Field_5	Field_6	Field_7
Template 0	DMAC0	DMAC1	DMAC2	SMAC0	SMAC1	SMAC2	EtherType	FS_07
Template 1	Ipv4SIP0	Ipv4SIP1	Ipv4DIP0	Ipv4DIP1	L4SPORT	L4DPORT	FS_02	FS_07
Template 2	Ipv6SIP0	Ipv6SIP1	L4SPORT	L4DPORT	FS_05	FS_06	FS_00	FS_01
Template 3	Ipv6DIP0	Ipv6DIP1	L4SPORT	L4DPORT	FS_00	FS_03	FS_04	FS_07
Template 4	FS_01	IPRange	FS_02	CTAG	STAG	FS_04	FS_03	FS_07

If user doesn't need any special requirement or combination, default template setting is enough for adding rule. If user wants to change the 5 templates setting for their own usage, API "*rtk_filter_igrAcl_template_set*" can be used.

Switch supports 8 Field Selectors. These entries can be used as user-defined field if the predefined field can not satisfy user's requirement. Each entry includes two setting: type and offset.

Table 14. Field Selector

Field	Description	Bits
FIELD_SELECTORn_TYPE	Format of field selector. It also defines the start address for 16-bits field selecting. 0x0:ASIC default setting 0x1:Raw packet(Start after preamble, begin with DA) 0x2:LLC packet 0x3:Ipv4 Packet (Started from start of Ipv4 header) 0x4:ARP Packet (Started after Ethernet II EtherType 0x0806) 0x5:Ipv6 Packet (Started from start of Ipv6 header)	3

	0x6:IP Payload(Started from IP payload, also means start of layer 4 packet) 0x7:L4 Payload (Started after TCP/UDP header, For ICMP, started at 4 bytes offset from ICMP header)	
FIELD_SELECTORn_OFFSET	Offset (byte)	8

The below table gives the default configuration of Field Selectors after ACL is initialized. Users can change it via API *"rtk_filter_igrAcl_field_sel_set"*.

Table 15. Default Field Selector Configuration

	Type	Offset
Field Selector 0	IPv6 (0x5)	0
Field Selector 1	IPv6 (0x5)	6
Field Selector 2	IP_Payload (0x6)	12
Field Selector 3	IPv4 (0x3)	6
Field Selector 4	IP_Payload (0x6)	0
Field Selector 5	IPv4 (0x3)	0
Field Selector 6	IPv4 (0x3)	8
Field Selector 7	Default (0x0)	0

Some other switch features also occupy some Field Selectors. The relationship between these features and Field Selectors are listed in below table.

Table 16. Switch Features and Field Selector

Field	DOS	RLDP	Valid Tag
Field Selector 0		O	
Field Selector 1		O	
Field Selector 2	O	O	
Field Selector 3		O	
Field Selector 4		O	
Field Selector 5	O	O	
Field Selector 6		O	
Field Selector 7			O

For example, if users use Field Selector 2 or 5 for their own usage, the switch can't recognize DOS packet. That means switch can't prevent DOS attack based on users' configuration.

If users need to use ACL and the switch feature in above table at the same time, the default Templates and Field Selectors configurations in API should be redefined to avoid the conflict.

Switch also supports "Care Tag" in ACL function. This function can be used for filtering some protocol or packet format with writing a protocol ID or packet content. For example, if users want to filter Ipv4 packet, he didn't writing a 0x0800 into a specified field. Just use care tag to specify the "Ipv4" into a rule. Check API "rtk_filter_igrAcl_cfg_add" for sample code. All the supported care tag types are listed as

following. Please notice that the care tag type after “CARE_TAG_IPV6” will use Field Selector 7 for store packet type. If users want to use care tag type after “CARE_TAG_IPV6”, please keep Field Selector 7 as default value.

Table 17. Care Tag List

Care Tag
CARE_TAG_CTAG
CARE_TAG_STAG
CARE_TAG_PPPOE
CARE_TAG_IPV4
CARE_TAG_IPV6
CARE_TAG_TCP
CARE_TAG_UDP
CARE_TAG_ARP
CARE_TAG_RSV1
CARE_TAG_RSV2
CARE_TAG_ICMP
CARE_TAG_IGMP
CARE_TAG_LLC
CARE_TAG_RSV3
CARE_TAG_HTTP
CARE_TAG_RSV4
CARE_TAG_RSV5
CARE_TAG_DHCP
CARE_TAG_DHCPV6
CARE_TAG_SNMP
CARE_TAG_OAM

int32 rtk_filter_igrAcl_init(void)

This API can initialize ACL and enable ACL function to all ports.

*Int32 rtk_filter_igrAcl_template_set(rtk_filter_template_t *aclTemplate)*

This API can configure template. The parameter aclTemplate carries 8 field types for a specified template.

Example:

```
/* Configure Template 2 as MAC_DA+MAC_SA+CTAG+STAG */
```

```
rtk_filter_template_t acltemp ;
memset(&acltemp, 0x00, sizeof(rtk_filter_template_t));

acltemp.index = 2;
acltemp.fieldType[0] = FILTER_FIELD_RAW_DMAC_15_0;
acltemp.fieldType[1] = FILTER_FIELD_RAW_DMAC_31_16;
acltemp.fieldType[2] = FILTER_FIELD_RAW_DMAC_47_32;
acltemp.fieldType[3] = FILTER_FIELD_RAW_SMAC_15_0;
acltemp.fieldType[4] = FILTER_FIELD_RAW_SMAC_31_16;
acltemp.fieldType[5] = FILTER_FIELD_RAW_SMAC_47_32;
acltemp.fieldType[6] = FILTER_FIELD_RAW_CTAG;
acltemp.fieldType[7] = FILTER_FIELD_RAW_STAG;

rtk_filter_igrAcl_template_set(&acltemp);
```

int32 rtk_filter_igrAcl_field_add(rtk_filter_cfg_t pFilter_cfg, rtk_filter_field_t* pFilter_field))*

This API adds a comparison rule to an ACL configuration. The pointer *pFilter_cfg* points to an ACL configuration structure, and this structure keeps multiple ACL comparison rules by means of linked list. The pointer *pFilter_field* will be added to linked list kept by structure that *pFilter_cfg* points to.

Example:

```
/*Add a destination MAC address 00-01-02-03-04-05 to ACL configuration*/
rtk_filter_field_t *field;
rtk_filter_cfg_t cfg ;

field = malloc(sizeof(rtk_filter_field_t));
field->fieldType = FILTER_FIELD_DMAC;
field->filter_pattern_union.dmac.dataType = FILTER_FIELD_DATA_MASK;
field->filter_pattern_union.dmac.value.octet[0] = 0;
field->filter_pattern_union.dmac.value.octet[1] = 0x01;
field->filter_pattern_union.dmac.value.octet[2] = 0x02;
field->filter_pattern_union.dmac.value.octet[3] = 0x03;
field->filter_pattern_union.dmac.value.octet[4] = 0x04;
field->filter_pattern_union.dmac.value.octet[5] = 0x05;
```

```
field->filter_pattern_union.dmac.mask.octet[0] = 0xFF;
field->filter_pattern_union.dmac.mask.octet[1] = 0xFF;
field->filter_pattern_union.dmac.mask.octet[2] = 0xFF;
field->filter_pattern_union.dmac.mask.octet[3] = 0xFF;
field->filter_pattern_union.dmac.mask.octet[4] = 0xFF;
field->filter_pattern_union.dmac.mask.octet[5] = 0xFF;

if ((retVal = rtk_filter_igrAcl_field_add(&cfg, field)) != RT_ERR_OK)
return retVal;
```

---

int32 rtk_filter_igrAcl_cfg_add(rtk_filter_id_t filter_id, rtk_filter_cfg_t pFilter_cfg, rtk_filter_action_t* pFilter_action, rtk_filter_number_t *ruleNum)*

This API can be used to add an ACL configuration to ASIC. The *filter_id* means the start rule id in the 64 ACL rules used for this configuration. The pointer of *pFilter_cfg* includes the data fields that needs to be filtered, the cared tag & frame types, and the active port mask. The return value of *ruleNum* means the rule number of the used ACL rules in this configuration.

Example 1:

```
/*ACL configuration that trap all packets with source IP 192.168.0.100,
destination IP 10.0.0.100, TCP with destination port number 0x8080  to
CPU.*/
Rtk_uint32 retVal, ruleNum;
rtk_filter_field_t *field1,*field2, *field3;
rtk_filter_cfg_t cfg ;
rtk_filter_action_t act;

/*Set CPU trap port to external interface 0*/
rtk_cpu_enable_set(ENABLED);
rtk_cpu_tagPort_set(EXT_PORT0, CPU_INSERT_TO_TRAPPING);

/* Reset Config & Action */
memset(&cfg, 0x00, sizeof(rtk_filter_cfg_t));
```

```
memset(&act, 0x00, sizeof(rtk_filter_action_t));

/*Add Source IP 192.168.0.100*/
field1= malloc(sizeof(rtk_filter_field_t));
memset(field1, 0, sizeof(rtk_filter_field_t));
field1->fieldType = FILTER_FIELD_IPV4_SIP;
field1->filter_pattern_union.sip.dataType = FILTER_FIELD_DATA_MASK;
field1->filter_pattern_union.sip.value    = 0xC0A80064;
field1->filter_pattern_union.sip.mask     = 0xFFFFFFFF;
field1->next = NULL;
if ((retVal = rtk_filter_igrAcl_field_add(&cfg, field1)) != RT_ERR_OK)
return retVal;

/*Add Destination IP 10.0.0.100*/
field2 = malloc(sizeof(rtk_filter_field_t));
memset(field2, 0, sizeof(rtk_filter_field_t));
field2->fieldType = FILTER_FIELD_IPV4_DIP;
field2->filter_pattern_union.sip.dataType = FILTER_FIELD_DATA_MASK;
field2->filter_pattern_union.dip.value    = 0x0A000064;
field2->filter_pattern_union.dip.mask     = 0xFFFFFFFF;
if ((retVal = rtk_filter_igrAcl_field_add(&cfg, field2)) != RT_ERR_OK)
return retVal;

/*Add TCP Port Number 0x8080*/
field3= malloc(sizeof(rtk_filter_field_t));
memset(field3, 0, sizeof(rtk_filter_field_t));
field3->fieldType = FILTER_FIELD_TCP_SPORT;
field3->filter_pattern_union.tcpSrcPort.dataType =
FILTER_FIELD_DATA_MASK;
field3->filter_pattern_union.tcpSrcPort.value    = 0x8080;
field3->filter_pattern_union.tcpSrcPort.mask     = 0xFFFF;
if ((retVal = rtk_filter_igrAcl_field_add(&cfg, field3)) != RT_ERR_OK)
return retVal;

/*Set TCP to be cared tag and add all ports to active ports*/
RTK_PORTMASK_ALLPORT_SET(cfg.activeport.value);
RTK_PORTMASK_ALLPORT_SET(cfg.activeport.mask);
cfg.careTag.tagType[CARE_TAG_TCP].value = TRUE;
cfg.careTag.tagType[CARE_TAG_TCP].mask = TRUE;
cfg.invert = FALSE;
```

```
/*Set Action to trap-to-CPU*/
act.actEnable[FILTER_ENACT_TRAP_CPU] = TRUE;

/*Add ACL configuration to ID 0*/
 if ((retVal = rtk_filter_igrAcl_cfg_add(0, &cfg, &act, & ruleNum)) !=
RT_ERR_OK)
    return retVal;
```

*int32 rtk_filter_igrAcl_cfg_get(rtk_filter_id_t filter_id, rtk_filter_cfg_raw_t *pFilter_cfg, rtk_filter_action_t *pAction)*

This API can get ACL rule configuration. The "*rtk_filter_cfg_raw_t*" data structure is as following:

```
typedef struct
{
    rtk_filter_field_raw_t
dataFieldRaw[RTK_FILTER_RAW_FIELD_NUMBER];
    rtk_filter_field_raw_t
careFieldRaw[RTK_FILTER_RAW_FIELD_NUMBER];
    rtk_filter_field_type_raw_t
fieldRawType[RTK_FILTER_RAW_FIELD_NUMBER];
    rtk_filter_care_tag_t       careTag;
    rtk_filter_activeport_t     activeport;

    rtk_filter_invert_t         invert;
    rtk_enable_t                valid;
} rtk_filter_cfg_raw_t;
```

The parameter "*dataFieldRaw*" means the data fields of this ACL rule, and " *careFieldRaw*" means the mask fields of this ACL rule. The definition of the data fields are in the parameter "*fieldRawType*", and the following is the define macro.

```
typedef enum rtk_filter_field_type_raw_e
{
```

```
    FILTER_FIELD_RAW_UNUSED = 0,
    FILTER_FIELD_RAW_DMAC_15_0,
    FILTER_FIELD_RAW_DMAC_31_16,
    FILTER_FIELD_RAW_DMAC_47_32,
    FILTER_FIELD_RAW_SMAC_15_0,
    FILTER_FIELD_RAW_SMAC_31_16,
    FILTER_FIELD_RAW_SMAC_47_32,
    FILTER_FIELD_RAW_ETHERTYPE,
    FILTER_FIELD_RAW_STAG,
    FILTER_FIELD_RAW_CTAG,

    FILTER_FIELD_RAW_IPV4_SIP_15_0 = 0x10,
    FILTER_FIELD_RAW_IPV4_SIP_31_16,
    FILTER_FIELD_RAW_IPV4_DIP_15_0,
    FILTER_FIELD_RAW_IPV4_DIP_31_16,



    FILTER_FIELD_RAW_IPV6_SIP_15_0 = 0x20,
    FILTER_FIELD_RAW_IPV6_SIP_31_16,
    FILTER_FIELD_RAW_IPV6_DIP_15_0 = 0x28,
    FILTER_FIELD_RAW_IPV6_DIP_31_16,

    FILTER_FIELD_RAW_L4_DPORT = 0x2A,
    FILTER_FIELD_RAW_L4_SPORT,

    FILTER_FIELD_RAW_VIDRANGE = 0x30,
    FILTER_FIELD_RAW_IPRANGE,
    FILTER_FIELD_RAW_PORTRANGE,
    FILTER_FIELD_RAW_FIELD_VALID,

    FILTER_FIELD_RAW_FIELD_SELECT00 = 0x40,
    FILTER_FIELD_RAW_FIELD_SELECT01,
    FILTER_FIELD_RAW_FIELD_SELECT02,
    FILTER_FIELD_RAW_FIELD_SELECT03,
    FILTER_FIELD_RAW_FIELD_SELECT04,
    FILTER_FIELD_RAW_FIELD_SELECT05,
    FILTER_FIELD_RAW_FIELD_SELECT06,
    FILTER_FIELD_RAW_FIELD_SELECT07,
    FILTER_FIELD_RAW_FIELD_SELECT08,
    FILTER_FIELD_RAW_FIELD_SELECT09,
```

```
    FILTER_FIELD_RAW_FIELD_SELECT10,
    FILTER_FIELD_RAW_FIELD_SELECT11,
    FILTER_FIELD_RAW_FIELD_SELECT12,
    FILTER_FIELD_RAW_FIELD_SELECT13,
    FILTER_FIELD_RAW_FIELD_SELECT14,
    FILTER_FIELD_RAW_FIELD_SELECT15,

    FILTER_FIELD_RAW_END,
} rtk_filter_field_type_raw_t;
```

Example:

```
/* Get ACL configuration from ID 0 */
rtk_filter_cfg_raw_t cfg;
rtk_filter_action_t act;

 if ((retVal = rtk_filter_igrAcl_cfg_get(0, &cfg, &act)) != RT_ERR_OK)
    return retVal;
```

---

int32 rtk_filter_igrAcl_cfg_del(rtk_filter_id_t filter_id)

This API is used to delete an ACL configuration with dedicated ACL rule ID.

Example:

```
/*Delete ACL configuration with rule ID 0*/
 if ((retVal = rtk_filter_igrAcl_cfg_del(0)) != RT_ERR_OK)
    return retVal;
```

---

int32 rtk_filter_igrAcl_state_set(rtk_filter_port_t port, rtk_filter_state_t state)

This API can enable/disable ACL function to a dedicated port.

Example:

```
/*Enable ACL function to Port 0, and disable ACL function to Port 1*/
if ((retVal = rtk_filter_igrAcl_state_set(UTP_PORT0,ENABLED)) !=
```

```
RT_ERR_OK)
return retVal;
if ((retVal = rtk_filter_igrAcl_state_set(UTP_PORT1,DISABLED)) !=
RT_ERR_OK)
   return retVal;
```

---

int32 rtk_filter_igrAcl_field_sel_set(rtk_uint32 index, rtk_field_sel_t format, rtk_uint32 offset)

This API can configure field selector. System support 16 user defined field selectors. Each selector can be enabled or disable. Using 16-bits in many predefined standard l2/l3/l4 payload.

Example:

```
/* Set Field Selector index 4 as Ipv4, offset = 6 bytes */
if((retVal=rtk_filter_igrAcl_field_sel_set(4, FORMAT_IPV4, 6))!=
RT_ERR_OK)
   return revVal
```

---

int32 rtk_filter_iprange_set(rtk_uint32 index, rtk_filter_iprange_t type, ipaddr_t upperIp, ipaddr_t lowerIp)

This API is used to Set IP Range check. By setting one or multiple IP Range check entries and reference them via an ACL rule, the rule can check all IP address between lower bound and upper bound.

Example:

```
    rtk_api_ret_t           ret_val ;
    rtk_filter_field_t      field1;
    rtk_filter_cfg_t        cfg ;
    rtk_filter_action_t     act;
    rtk_uint32              ruleNum;
    ipaddr_t                lowerip, upperip;
```

```
    if ((ret_val = rtk_filter_igrAcl_init()) != RT_ERR_OK)
        return ret_val;

    lowerip = 0x01020304; /* 1.2.3.4 */
    upperip = 0x05060708; /* 5.6.7.8 */
    if ((ret_val = rtk_filter_iprange_set(2, IPRANGE_IPV4_SIP, upperip,
lowerip)) != RT_ERR_OK)
        return ret_val;

    memset(&cfg, 0x00, sizeof(rtk_filter_cfg_t));
    memset(&act, 0x00, sizeof(rtk_filter_action_t));
    memset(&field1, 0x00, sizeof(rtk_filter_field_t));

    field1.fieldType = FILTER_FIELD_IP_RANGE;
    field1.filter_pattern_union.inData.dataType = FILTER_FIELD_DATA_MASK;
    field1.filter_pattern_union.inData.value = 0x0004;/* IP Range index 2
*/
    field1.filter_pattern_union.inData.mask  = 0xFFFF;

    if ((ret_val = rtk_filter_igrAcl_field_add(&cfg, &field1)) !=
RT_ERR_OK)
        return ret_val;

    RTK_PORTMASK_ALLPORT_SET(cfg.activeport.value);
    RTK_PORTMASK_ALLPORT_SET(cfg.activeport.mask);
    cfg.invert = FALSE;

    act.actEnable[FILTER_ENACT_DROP] = TRUE;
    if ((ret_val = rtk_filter_igrAcl_cfg_add(0, &cfg, &act, &ruleNum)) !=
RT_ERR_OK)
        return ret_val;
```

4.18. EEE

In switch, IEEE 802.3az Energy Efficient Ethernet (EEE) is supported. Users can use the following APIs to initialize EEE and enable/disable per port EEE function.

Int32 rtk_eee_init(void)

The API can initialize EEE.

Int32 rtk_eee_portEnable_set(rtk_port_t port, rtk_enable_t enable)

This API can be used to enable EEE function per port.

Example :

```
/*Enable EEE on switch */
/*Initialize EEE status*/
rtk_eee_init();

/*Enable EEE on Port 0 to Port 4*/
rtk_eee_portEnable_set(UTP_PORT0,ENABLED);
rtk_eee_portEnable_set(UTP_PORT1,ENABLED);
rtk_eee_portEnable_set(UTP_PORT2,ENABLED);
rtk_eee_portEnable_set(UTP_PORT3,ENABLED);
rtk_eee_portEnable_set(UTP_PORT4,ENABLED);
```

4.19. 802.1X

802.1X is an IEEE standard for media-level access control, offering the capability to permit or deny network connectivity, control VLAN access and apply traffic policy, based on user or machine identity.

In switch, hardware can trap 802.1X EAPOL packets to CPU to process authentication from LAN side. After authentication, CPU can set a port to authenticated/ un-authenticated port or a MAC to authenticated/ un-authenticated MAC. Besides, CPU can assign the un-authenticated action of port or MAC.

Int32 rtk_dot1x_eapolFrame2CpuEnable_set(rtk_enable_t enable)

The API is used to enable the functionality of trap EAPOL frames to CPU.

Int32 rtk_dot1x_portBasedEnable_set(rtk_port_t port, rtk_enable_t enable)

The API can set the port-based 802.1X ability. If a port is 802.1x port based network access control "enabled", it should be authenticated and packets from that port won't be dropped or trapped to CPU.

Int32 rtk_dot1x_portBasedAuthStatus_set(rtk_port_t port, rtk_dot1x_auth_status_t port_auth)

When the port-based 802.1X ability is enabled, the API is used to set the authentication status of the port.

Int32 rtk_dot1x_portBasedDirection_set(rtk_port_t port, rtk_dot1x_direction_t port_direction)

The API is used to set operational direction of 802.1x port-based network access control.

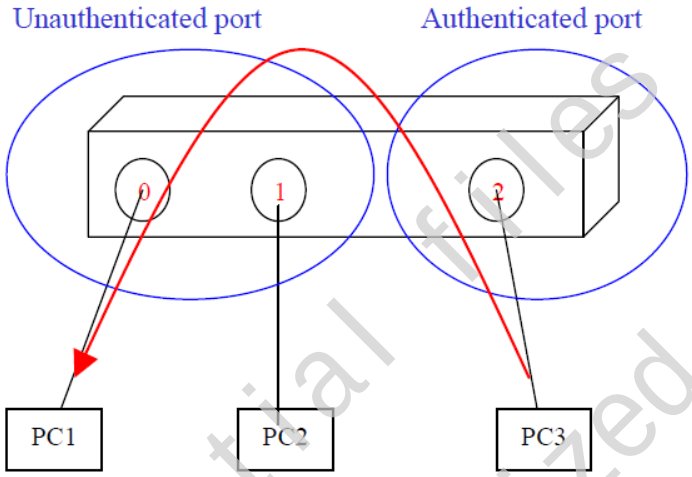

Figure 10. 302.1x Direction Configuration

DIR_IN:

- Only drop ingress packet if the port is not authenticated. The port still can transmit packet received from other ports. In above figure, PC3 still can transmit packet to PC1.

DIR_BOTH:

- Both ingress & egress packet are forbidden on the unauthenticated port. In above figure, PC3 can't transmit packet to PC1.

Int32 rtk_dot1x_unauthPacketOper_set(rtk_port_t port, rtk_dot1x_unauth_action_t unauth_action)

The API is used to set un-authentication action of 802.1x port-based network access control.

Example:

```
/* Set EXT port 0 to CPU without proprietary tag inserting but trap
packets*/
rtk_cpu_insert_t mode ;
mode = CPU_INSERT_TO_TRAPPING;
rtk_cpu_enable_set(ENABLE);
rtk_cpu_tagPort_set(EXT_PORT0, mode) ;

/*Enable trap-EAPOL-to-CPU */
```

```
rtk_dot1x_eapolFrame2CpuEnable_set(ENABLED);

/*Set Port 0 to Port 3 un-auth action to drop */
rtk_dot1x_unauthPacketOper_set(UTP_PORT0, DOT1X_ACTION_DROP);
rtk_dot1x_unauthPacketOper_set(UTP_PORT1, DOT1X_ACTION_DROP);
rtk_dot1x_unauthPacketOper_set(UTP_PORT2, DOT1X_ACTION_DROP);
rtk_dot1x_unauthPacketOper_set(UTP_PORT3, DOT1X_ACTION_DROP);

/*Set Port 0 to Port 3 to enable 802.1X ability and set port status to
un-authentication */
rtk_dot1x_portBasedEnable_set(UTP_PORT0, ENABLED);
rtk_dot1x_portBasedAuthStatus_set(UTP_PORT0, UNAUTH);
rtk_dot1x_portBasedEnable_set(UTP_PORT1, ENABLED);
rtk_dot1x_portBasedAuthStatus_set(UTP_PORT1, UNAUTH);
rtk_dot1x_portBasedEnable_set(UTP_PORT2, ENABLED);
rtk_dot1x_portBasedAuthStatus_set(UTP_PORT2, UNAUTH);
rtk_dot1x_portBasedEnable_set(UTP_PORT3, ENABLED);
rtk_dot1x_portBasedAuthStatus_set(UTP_PORT3, UNAUTH);

/*After Port0 is passed the authentication process, set Port 0 to auth
state and assign the direction to IN*/
rtk_dot1x_portBasedAuthStatus_set(UTP_PORT0, AUTH);
rtk_dot1x_portBasedDirection_set((UTP_PORT0, IN);
```

4.20. Share Meter

Switch supports 40 share meters for rate control setup. The share meters can be assigned to ACL policing, storm control and queue APR control functions.

There are 2 types of meter: rate-based and packet-based. Each meter can be configured to be one of these 2 types. The granularity of rate-based share meter is 8 kbps, and the setup range is 8kbps to 1Gbps. The setup rage of packet-based share meter is from 1 packet/s to 524,287 packet/s.

The bucket size of each share meter can be configured.

Int32 rtk_rate_shareMeter_set(rtk_meter_id_t index, rtk_meter_type_t type, rtk_rate_t rate, rtk_enable_t ifg_include)

Example:

```
/* Setup share meter index 10 as rate-based with 256K and exclude ifg*/
if(RT_ERR_OK != rtk_rate_shareMeter_set(10, METER_TYPE_KBPS, 256,
DISABLE))
    return RT_ERR_FAILED;

/* Setup share meter index 20 as packet-based with 100 packet/s */
if(RT_ERR_OK != rtk_rate_shareMeter_set(20, METER_TYPE_PPS, 100,
DISABLE))
    return RT_ERR_FAILED;
```

---

*int32 rtk_rate_shareMeter_get(rtk_meter_id_t index, rtk_meter_type_t *pType, rtk_rate_t *pRate, rtk_enable_t *pIfg_include)*

Example:

```
/*Get meter index 10 type, rate and inter-frame-gap ability*/
rtk_meter_type_t type;
```

```
rtk_rate_t rate;
rtk_enable_t ifg;

if(RT_ERR_OK != rtk_rate_shareMeter_get(10, &type, &rate, &ifg))
    return RT_ERR_FAILED;
```

---

int32 rtk_rate_shareMeterBucket_set(rtk_meter_id_t index, rtk_uint32 bucket_size)

Example:

```
/*Set Bucket size of meter index 10 to 65535 bytes*/
if(RT_ERR_OK != rtk_rate_shareMeterBucket_set(10, 65535))
    return RT_ERR_FAILED;
```

---

*int32 rtk_rate_shareMeterBucket_get(rtk_meter_id_t index, rtk_uint32 *pBucket_size)*

Example:

```
/*Get Bucket size of meter index 10 */
if(RT_ERR_OK != rtk_rate_shareMeterBucket_get(10, &size))
    return RT_ERR_FAILED;
```

4.21. IGMP

Switch supports H/W IGMP/MLD snooping with maximum 256 groups without extra software effort. These multicast groups are learned and aged out automatically. For those packets of known multicast group, switch would forward them according to the group membership it learned.

IGMPv1/v2/v3 and MLDv1/v2 are supported. The Ipv4 multicast data packets are forwarded per group IP. Ipv6 multicast data packets are forwarded per destination MAC. That is, those Ipv6 multicast groups shares the same destination MAC will be treated as the same group. This is called address ambiguity.

Some reserved range IP addresses will always be flooded to all ports by default. If IGMP or MLD report message request to join these groups, this request will be ignored silently. These default reserved IP addresses are:

Ipv4: 224.0.0.0 ~ 224.0.0.255

Ipv6: 33:33:00:00:00:00 ~ 33:33:00:00:00:FF

User can specify "Static Router Ports" via API. With this configuration, users can force ports to act as a real router port. All Report and Leave messages would be forward to "Static Router ports" just like "Dynamic Router Ports"

Even IGMPv3 and MLDv2 are supported by H/W IGMP Snooping. However, source filtering feature of IGMPv3 and MLDv2 is not supported. That is, when switch receives an IGMPv3 or MLDv2 report massage, it simply joins the receiving port into specified multicast group.

Int32 rtk_igmp_init(void)

The API initialize IGMP/MLD snooping module and configure IGMP/MLD protocol action as following

IGMPv1: H/W process

IGMPv2: H/W process

IGMPv3: Flooding without process

MLDv1: H/W process

MLDv2: Flooding without process.

Fast Leave: Enabled

Dynamic router port learning: Disabled

Example:

```
/*Initialize H/W IGMP function*/
if(RT_ERR_OK != rtk_igmp_init())
    return RT_ERR_FAILED;
```

int32 rtk_igmp_state_set(rtk_enable_t enabled)

Example:

```
/*  Enable H/W IGMP function*/
if(RT_ERR_OK != rtk_igmp_state_set(ENABLED))
    return RT_ERR_FAILED;

/*  Disable H/W IGMP function*/
if(RT_ERR_OK != rtk_igmp_state_set(DISABLED))
    return RT_ERR_FAILED;
```

*int32 rtk_igmp_state_get(rtk_enable_t *pEnabled)*

Example:

```
/* Get current state of H/W IGMP function*/
rtk_enable_t enabled;

if(RT_ERR_OK != rtk_igmp_state_get(&enabled))
    return RT_ERR_FAILED;
```

*int32 rtk_igmp_static_router_port_set(rtk_portmask_t *pPortmask)*

Example:

```
/* Set Port 0 and Port 2 as static router port */

rtk_portmask_t    portmask;
```

```
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT0);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT2);
if(RT_ERR_OK != rtk_igmp_static_router_port_set(&portmask))
    return RT_ERR_FAILED;
```

*int32 rtk_igmp_static_router_port_get(rtk_portmask_t *pPortmask)*

Example:

```
/* Get current static router port configuration */
rtk_portmask_t    port_mask;

if(RT_ERR_OK != rtk_igmp_static_router_port_get(&portmask))
    return RT_ERR_FAILED;
```

int32 rtk_igmp_protocol_set(rtk_port_t port, rtk_igmp_protocol_t protocol, rtk_trap_igmp_action_t action)

The API is used to set per port configuration of IGMP/MLD packet action.

Example:

```
/* Set UTP Port 0 IGMP v1/v2 supported by H/W. IGMPv3 is flooding. MLDv1
MLDv2 is dropped.*/

rtk_igmp_protocol_set(UTP_PORT0, PROTOCOL_IGMPv1, IGMP_ACTION_ASIC);
rtk_igmp_protocol_set(UTP_PORT0, PROTOCOL_IGMPv2, IGMP_ACTION_ASIC);
rtk_igmp_protocol_set(UTP_PORT0, PROTOCOL_IGMPv3, IGMP_ACTION_FORWARD);
rtk_igmp_protocol_set(UTP_PORT0, PROTOCOL_MLDv1, IGMP_ACTION_DROP);
rtk_igmp_protocol_set(UTP_PORT0, PROTOCOL_MLDv2, IGMP_ACTION_DROP);
```

*rtk_igmp_protocol_get(rtk_port_t port, rtk_igmp_protocol_t protocol, rtk_trap_igmp_action_t *pAction)*

The API is used to get per port configuration of IGMP/MLD packet action.

Example:

```
/* Get IGMP v1 action at Port 0 */
rtk_trap_igmp_action_t  action ;

if(RT_ERR_OK != rtk_igmp_protocol_get(UTP_PORT0, PROTOCOL_IGMPv1,
&action))
    return RT_ERR_FAILED;
```

---

rtk_api_ret_t rtk_igmp_fastLeave_set(rtk_enable_t state)

The API is used to set IGMP/MLD Fast Leave state. If users specify this function as “ENABLED”, the port membership will be removed immediately when a Leave packet is received.

Example:

```
/* Enabled Fast Leave */

if(RT_ERR_OK != rtk_igmp_fastLeave_set(ENABLED))
    return RT_ERR_FAILED;
```

---

*rtk_api_ret_t rtk_igmp_fastLeave_get(rtk_enable_t *pState)*

The API is used to get IGMP/MLD Fast Leave state.

Example:

```
/* Get Fast Leave state */
rtk_enable_t state;

if(RT_ERR_OK != rtk_igmp_fastLeave_get(&state))
    return RT_ERR_FAILED;
```

```
```

rtk_api_ret_t rtk_igmp_maxGroup_set(rtk_port_t port, rtk_uint32 group)

The API is used to set per port multicast group learning limit..

Example:

```
/* Set port 0 maximum group to 50 */

if(RT_ERR_OK != rtk_igmp_maxGroup_set(UTP_PORT0, 50))
    return RT_ERR_FAILED;
```

*rtk_api_ret_t rtk_igmp_maxGroup_get(rtk_port_t port, rtk_uint32 *pGroup)*

The API is used to get per port multicast group learning limit.

Example:

```
/* get port 0 maximum group limitation */
rtk_uint32 group;

if(RT_ERR_OK != rtk_igmp_maxGroup_get(UTP_PORT0, &group))
    return RT_ERR_FAILED;
```

*rtk_api_ret_t rtk_igmp_currentGroup_get(rtk_port_t port, rtk_uint32 *pGroup)*

The API is used to get per port current multicast group count

Example:

```
/* get port 0 current group number */
rtk_uint32 group;

if(RT_ERR_OK != rtk_igmp_currentGroup_get(UTP_PORT0, &group))
```

```
    return RT_ERR_FAILED;
```

5. Application

5.1. Wireless Router

In this application, one WAN port and four LAN ports are defined by VLAN partition while CPU port overlapped with all the other ports. Let UTP_PORT0 to be WAN port, UTP_PORT0 ~ UTP_PORT4 to be LAN ports and EXT_PORT0 to be CPU port. UTP_PORT0 and EXT_PORT0 are in VLAN 1 while UTP_PORT0 ~ UTP_PORT4 and EXT_PORT0 in VLAN 2. CPU can use VLAN tag in packets to parser the packet is from LAN or WAN. LEDs are assigned to P0~P4, and use group 0&1.

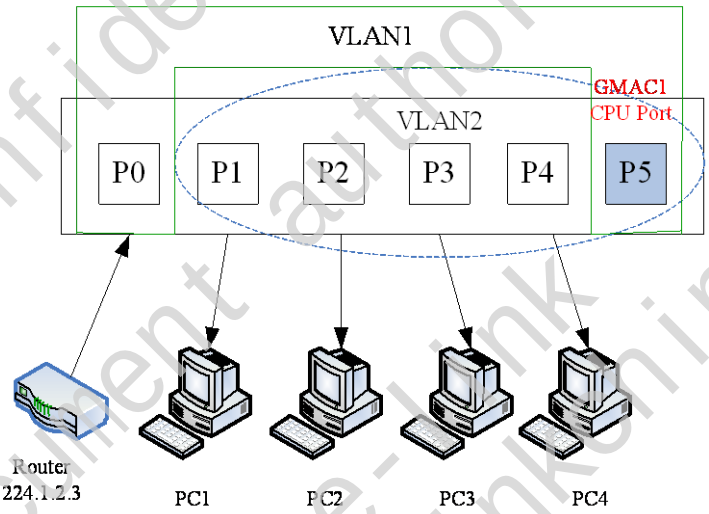

Figure 11. Wireless Router Application Diagram

Example codes:

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

/* Enable LED Group 0&1 from P0 to P4 */
rtk_portmask_t portmask;
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT0);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT1);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT2);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT3);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT4);
```

```
rtk_led_enable_set(LED_GROUP_0, &portmask);
rtk_led_enable_set(LED_GROUP_1, &portmask);

/* initialize VLAN */
rtk_vlan_init();

/*
all the ports are in the default VLAN 1 after VLAN initialized, modify it
as follows:
VLAN1 member : UTP_PORT0, EXT_PORT0 with fid 1 ;
VLAN2 member : UTP_PORT1~UTP_PORT4, EXT_PORT0 with fid 2
*/
rtk_portmask_t mbrmsk;
rtk_portmask_t untagmsk;

RTK_PORTMASK_CLEAR(mbrmsk);
RTK_PORTMASK_PORT_SET(mbrmsk, UTP_PORT0);
RTK_PORTMASK_PORT_SET(mbrmsk, EXT_PORT0);
RTK_PORTMASK_CLEAR(untagmask);
RTK_PORTMASK_PORT_SET(untagmsk, UTP_PORT0);
rtk_vlan_set(1, mbrmsk, untagmsk, 1);

RTK_PORTMASK_CLEAR(mbrmsk);
RTK_PORTMASK_PORT_SET(mbrmsk, UTP_PORT1);
RTK_PORTMASK_PORT_SET(mbrmsk, UTP_PORT2);
RTK_PORTMASK_PORT_SET(mbrmsk, UTP_PORT3);
RTK_PORTMASK_PORT_SET(mbrmsk, UTP_PORT4);
RTK_PORTMASK_PORT_SET(mbrmsk, EXT_PORT0);
RTK_PORTMASK_CLEAR(untagmask);
RTK_PORTMASK_PORT_SET(untagmsk, UTP_PORT0);
RTK_PORTMASK_PORT_SET(untagmsk, UTP_PORT1);
RTK_PORTMASK_PORT_SET(untagmsk, UTP_PORT2);
RTK_PORTMASK_PORT_SET(untagmsk, UTP_PORT3);
rtk_vlan_set(2,mbrmsk, untagmsk, 2);

/* set PVID for each port */
rtk_vlan_portPvid_set(UTP_PORT0, 1, 0);
rtk_vlan_portPvid_set(UTP_PORT1, 2, 0);
rtk_vlan_portPvid_set(UTP_PORT2, 2, 0);
rtk_vlan_portPvid_set(UTP_PORT3, 2, 0);
```

```
rtk_vlan_portPvid_set(UTP_PORT4, 2, 0);
rtk_vlan_portPvid_set(EXT_PORT0, 2, 0);
```

5.2. XDSL Modem

In this application, CPU and switch form a XDSL modem. All the ports of switch are LAN ports while WAN port is on CPU. Let EXT_PORT0 to be CPU port and CPU connects with EXT_PORT0 through MII port. Suppose the CPU worked with switch has set its MII port as MAC mode. Packets with internal IP address will be transmitted to WAN port with translated IP address and layer4 port. LEDs are assigned to UTP_PORT0~UTP_PORT4, and use group 0&1.

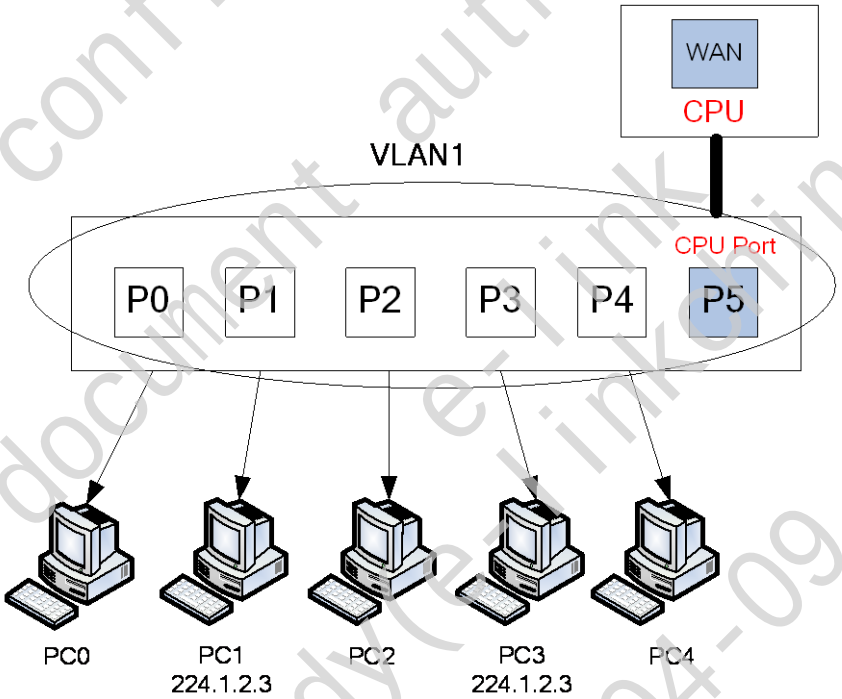

Figure 12. XDSL Modem Application Diagram

Example codes :

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

/* Enable LED Group 0&1 from UTP_PORT0 to UTP_PORT4 */
rtk_portmask_t portmask
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT0);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT1);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT2);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT3);
RTK_PORTMASK_PORT_SET(portmask, UTP_PORT4);
rtk_led_enable_set(LED_GROUP_0, &portmask);
rtk_led_enable_set(LED_GROUP_1, &portmask);

/* set port 5 as CPU port */
rtk_cpu_enable_set(ENABLE);
rtk_cpu_tagPort_set(EXT_PORT0, CPU_INSERT_TO_ALL);
```

6. Port Number in switch

6.1. Ports Mapping

The port mapping (include UTP port and Extension port) of all supported chip module is listed as following:

Table 18. Port Mapping For UTP and Ext Port.

Port ID in API	UTP_PORT0	UTP_PORT1	UTP_PORT2	UTP_PORT3	UTP_PORT4	UTP_PORT5	UTP_PORT6	UTP_PORT7
RTL8363NB		V		V				
RTL8364NB		V		V				
RTL8365MB-VC	V	V	V	V				
RTL8367RB-VB	V	V	V	V	V			
RTL8367SB	V	V	V	V	V			
RTL8367S	V	V	V	V	V			
RTL8366SC	V	V	V	V				
RTL8363SC			V	V				
RTL8370MB	V	V	V	V	V	V	V	V
RTL8310SR	V	V	V	V	V	V	V	V
RTL8363NB-VB		V		V				
RTL8364NB-VB		V		V				
RTL8363SC-VB			V	V				
RTL8366SD	V	V	V	V	V			
RTL8367S-VB	V	V	V	V	V			
RTL8367RB-VC	V	V	V	V	V			
RTL8367SC	V	V	V	V	V			
RTL8365MB-VD	V	V	V	V				

Port ID in API	UTP_PORT4	EXT_PORT0(GMAC1)	EXT_PORT1(GMAC2)
RTL8363NB		MII / TMII / RMII / RGMII SGMII / High-SGMII	
RTL8364NB		MII / TMII / RMII / RGMII SGMII / High-SGMII	MII / TMII / RMII / RGMII
RTL8365MB-VC		MII / TMII / RMII / RGMII	
RTL8367RB-VB		MII / TMII / RMII / RGMII	MII / TMII / RMII / RGMII
RTL8367SB		MII / TMII / RMII / RGMII SGMII / High-SGMII	MII / TMII / RMII / RGMII
RTL8367S		SGMII / High-SGMII	MII / TMII / RMII / RGMII
RTL8366SC		SGMII / High-SGMII	MII / TMII / RMII / RGMII
RTL8363SC		SGMII / High-SGMII	
RTL8370MB		MII / TMII / RMII / RGMII SGMII	MII / TMII / RMII / RGMII SGMII / High-SGMII
RTL8310SR		MII / TMII / RMII / RGMII SGMII	MII / TMII / RMII / RGMII
RTL8363NB-VB		MII / TMII / RMII / RGMII SGMII / High-SGMII	
RTL8364NB-VB		MII / TMII / RMII / RGMII SGMII / High-SGMII	MII / TMII / RMII / RGMII SGMII
RTL8363SC-VB		SGMII / High-SGMII	
RTL8366SD		SGMII / High-SGMII	
RTL8367S-VB		SGMII / High-SGMII	MII / TMII / RMII / RGMII
RTL8367RB-VC		SGMII / High-SGMII	MII / TMII / RMII / RGMII
RTL8367SC	RGMII	SGMII / High-SGMII	SGMII / High-SGMII
RTL8365MB-VD			MII / TMII / RMII / RGMII