REALTEK

RTL9607C
SINGLE-CHIP PON

Access Control List (ACL) Application Note
(CONFIDENTIAL: Development Partners Only)

Rev. 1.0.0
01 June 2017

REALTEK
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

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
1.0.0	2017/06/01	First Release

Table of Contents

List of Tables

List of Figures

1. Overview

The ACL (Access Control List) is an ingress filtering function. This function works when a packet received at specified ports. Once the packet is matching one or more ACL rules, corresponding action will be taken. Entire ACL function includes 5 main components, Rules (including data & mask), Actions, Templates, Field Selectors and Range check. The relationship between them and a sample configuration is as following:

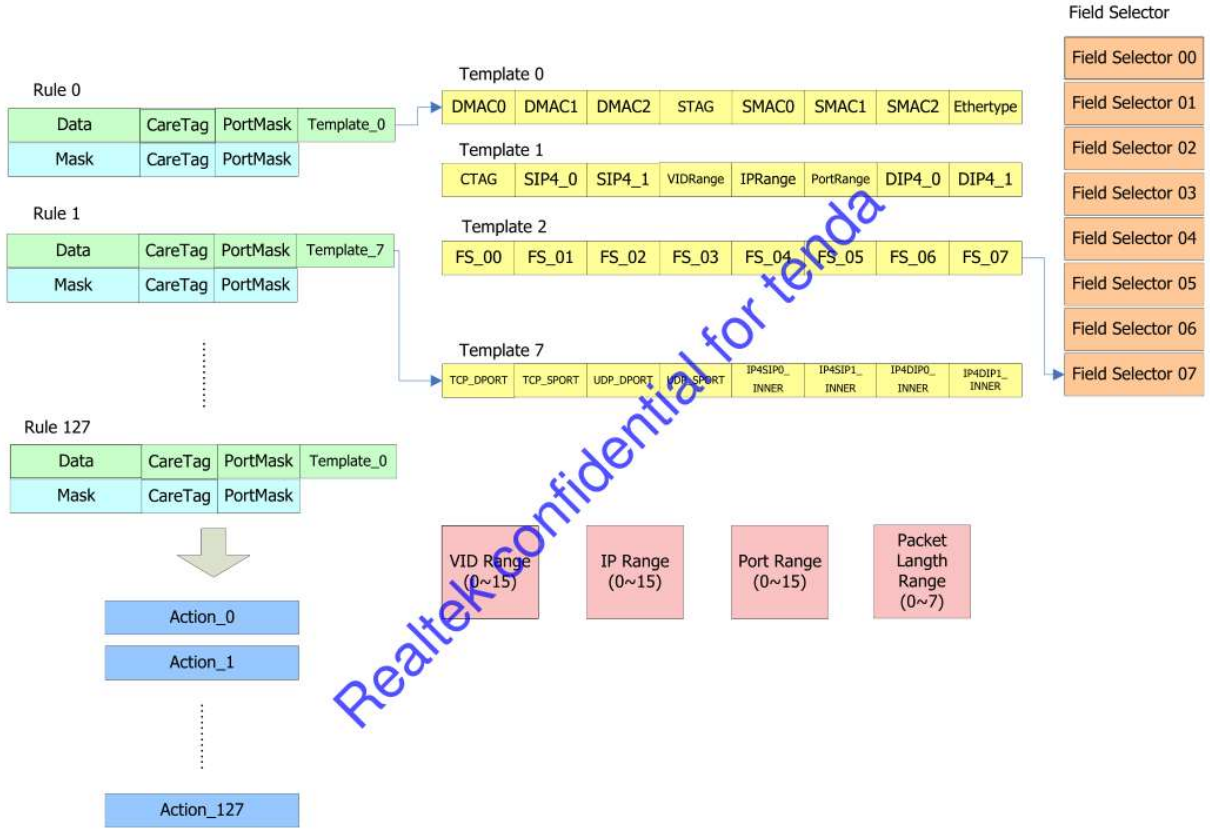

Figure 1. ACL Block Diagram

There are 8 templates and 8 fields in each template. Each filed is a 2 bytes indicator which represents a specified position in a packet. Each ACL rule must pointer to one template. ACL will use the fields in template to decode the data and mask in ACL rule and compare the data/mask with specified position in a packet. Each ACL rule is corresponding to a single entry in Action table. When an incoming packet which is matched with some ACL rules, the corresponding action in Action table will be taken.

Some predefined fields can be set in a template. If these predefined fields can not satisfy user's requirement, user can use Field Selector to define their own field. Field Selector will be discussed in the following Section.

Normally, ACL compare the field of incoming packets with data/mask pair in an ACL rule. For advanced usage, ACL also provides the following 4 range check features: VID, IP, TCP or UDP port and packet length. Users can specify the upper bound and lower bound to an entry of range check. There are 8 entries in Packet-Length range check table and 16 entries VID/IP/TCP or UDP port range check table.

2. ACL Configuration

The ACL function can be enabled/disabled per port basis. Only if the packet received at an ACL-enabled port, the comparison would be made. All packets received at an ACL-disabled port will not get the "matched" result.

For those packets are not matched any ACL rules at ACL-enabled port, another port-basis configuration, ACL_PERMIT, will be applied on them. By setting this configuration to "Drop", only matched packets will be forwarded at this ingress port. If use set this configuration to "Permit", all packets can be forwarded but the actions specified by ACL would only apply on matched packets. This configuration is only referenced if the ACL state of ingress port in enabled.

Table 1. ACL Port Configuration

Field Name	Bits	Description
ACL_EN	1	Per-port Enable/Disable ACL lookup
ACL_PERMIT	1	Per-port ACL Rule not matched permit or drop. This entry is referenced only when ACL_EN is 1

3. Templates

Each ACL rule points to a template. The template defines that how to decode the data/mask in a rule. There are 8 templates supported in ASIC. Each template includes 8 fields. Each filed is an indicator which represents a specify 2 bytes position within a packet. For example, destination MAC address is 48 bits long. Users need to use 3 fields to represent a destination MAC address in a template. The figure below gives the configuration of template 0 with destination MAC and source MAC.

Figure 2. Template Sample Configuration

Some predefined fields can be used to set in a template. All predefined field and their type value are given in the following table. Type 0x40 ~ 0x47 means that this field refer to Field selector 0 ~ 7. Field Selector is a user-defined field. If users need to filter the field in a packet which is not predefined, Field Selector can be used. The discussion of Field Selector will be given in next section.

Table 2. Predefined Field

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}
0x0A	GEMIDX	GPON GEMIDX
0x0B	Frame Type and Tags	Frame Format & Tag indicator flag
0x10	IP4SIP0	IPv4 SIP[15:0]
0x11	IP4SIP1	IPv4 SIP[31:16]
0x12	IP4DIP0	IPv4 DIP[15:0]
0x13	IP4DIP1	IPv4 DIP[31:16]
0x14	IP4(TOS+PROTO)	IPv4 TOS[15:8]+ IP Protocol[7:0].
0x15	IP6(TC+NextHeader)	IPv6 TC[15:8] + Last Next Header[7:0].
0x16	Inner IP4SIP0	IPv4 SIP[15:0] for inner IP header
0x17	Inner IP4SIP1	IPv4 SIP[31:16] for inner IP header
0x18	Inner IP4DIP0	IPv4 DIP[15:0] for inner IP header
0x19	Inner IP4DIP1	IPv4 DIP[31:16] for inner IP header
0x1A	IP4(TOS+PROTO)	IPv4 TOS[15:8]+protocol[7:0] for inner IP header
0x1B	TCP Sport	TCP Source Port(outer L4 for single IP and inner L4 for dual IP)
0x1C	TCP Dport	TCP Destination Port(outer L4 for single IP and inner L4 for dual IP)
0x1D	UDP Sport	UDP Source Port(outer L4 for single IP and inner L4 for dual IP)
0x1E	UDP Dport	UDP Destination Port(outer L4 for single IP and inner L4 for dual IP)
0x20	IP6SIP0	IPv6 SIP[15:0]
0x21	IP6SIP1	IPv6 SIP[31:16]
0x22	IP6SIP2	IPv6 SIP[47:32]

0x23	IP6SIP3	IPv6 SIP[63:48]
0x24	IP6SIP4	IPv6 SIP[79:64]
0x25	IP6SIP5	IPv6 SIP[95:80]
0x26	IP6SIP6	IPv6 SIP[111:96]
0x27	IP6SIP7	IPv6 SIP[127:112]
0x28	IP6DIP0	IPv6 DIP[15:0]
0x29	IP6DIP1	IPv6 DIP[31:16]
0x2A	IP6DIP2	IPv6 DIP[47:32]
0x2B	IP6DIP3	IPv6 DIP[63:48]
0x2C	IP6DIP4	IPv6 DIP[79:64]
0x2D	IP6DIP5	IPv6 DIP[95:80]
0x2E	IP6DIP6	IPv6 DIP[111:96]
0x2F	IP6DIP7	IPv6 DIP[127:112]
0x30	VIDRANGE	S-tag/C-tag VID Range Check Mask
0x31	IPRANGE	IPv4/IPv6 Range Check Mask
0x32	PORTRANGE	TCP/UDP Port Range Check Mask
0x33	PKTLENRANGE	Packet length range check mask
0x34	FIELD_VALID	Field selectors valid tag
0x35	FIELD_EXTMSK	Source Extension Port Mask, {EXT6-1,CPU}[6:0]
0x4n	FIELD_SELECTORn	User defined 16-bits field

4. Field Selectors

There are 16 entries of Field Selectors in ACL function. 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 3. Field Selector

Field Name	Bits	Description
FIELD_SELECTORn_TYPE	3	Format of field selector. It also defines the start address for 16-bits field selecting. 0x0:Raw packet(Start after preamble, begin with DA) 0x1: PPPoE(Both ethertype 0x8864, PPP Session Stage/ Ethertype 8863, Discovery Stage) packet (Start from PPPoE header) 0x2:outer IPv4 Packet (Started from start of outer IPv4 header) 0x3:inner IPv4 Packet (Started from start of inner IPv4 header) 0x4:outer IPv6 Packet (Started from start of outer IPv6 header) 0x5:inner IPv6 Packet (Started from start of inner IPv6 header) 0x6:IP payload(Started from outer IP payload) 0x7:IP payload(Started from inner IP payload)
FIELD_SELECTORn_OFFSET	8	Offset (byte)

For example, if users need to set Field Selector entry 6 to be TCP destination port and entry 7 to be TCP source port, they need to know the position of these 2 fields in TCP header. The position of destination port is at beginning of TCP header and source port field is followed after it. The configuration may be like:

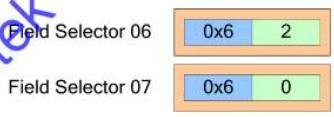

Figure 3. Field Selector Sample Configuration

Type 0x6 means that the starting point of this entry is from IP Payload, which is equal to the first bytes of TCP header. The offset 2 bytes indicate this entry points to the position of 2 bytes after the beginning of TCP header. Thus, (Type 0x6, offset 2 bytes) is the destination port and (Type 0x6, offset 0 bytes) is the source port.

Field Type 0xB is used as "packet valid tag" and "packet type" indication for ACL rule filtering. The bit map is given as following:

Table 4. Field Type 0xB

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
L2TP	PPTP	DSLITE	ARP	IGMP/MLD	TCP	UDP	TCP/UDP
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Inner IPv4	IPv6	Outer IPv4	PPPoE(0x8864)	PPPoE(0x8863)	PPPoE(0x8864/0x8863)	Reserved	Reserved

If an IGMP packet of DSLite is received, bit-13, bit-11 and bit-7 will be turned on and all the other bits in Field Type 0xB will be turned off. By checking these bits, users can filter packets with some tags or protocols without specifying the tag type or protocol number in the data/mask of an ACL rule.

For example, if users want to set ACL rule 14 to filter an IGMP packet of DS-Lite, the configuration may be like following.

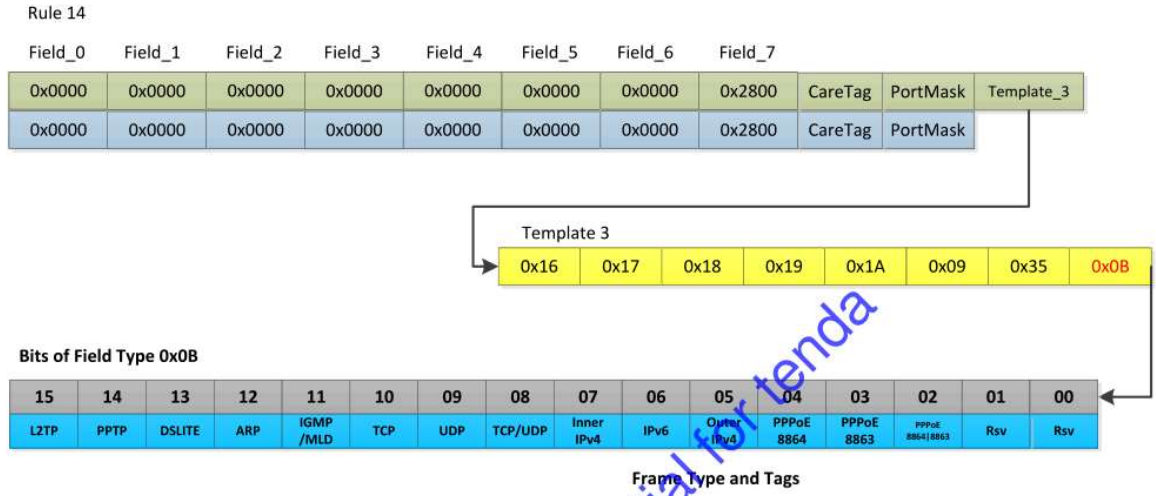

Figure 4. Packet Type and Tag Configuration

5. Rules

128 ACL rules are supported in switch. Each rule includes 128bits data/mask, 7 bits care tag/care mask, 4 bits active port/active mask and 3 bits template pointer. The relationships of each field of ACL rule and template and a sample rule configuration are given as following:

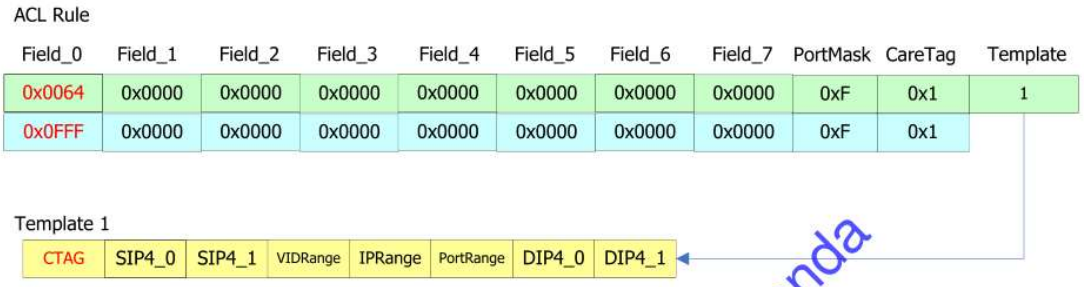

Figure 5. ACL Rule Sample

Each 16 bits fields in an ACL rule would be mapped to the template which pointed by its template field. For Figure 5, the ACL rule points to template 1. Thus the data 0x0064 in field 0 would be decoded as CTAG data. The mask 0x0FFF indicates that users only want to filter VLAN ID 100 no matter its priority and CFI in a CTAG.

Users may notice that it has 8 fields in Template 1. For those fields which users don't want to filter, just set the mask to 0x0000 in ACL rule.

Users may notice that it has 8 fields in Template 1. For those fields which users don't want to filter, just set the mask to 0x0000 in ACL rule.

The field after 8 data/mask file is Care Tag. This field is used to filter some tag or protocol in a packet. 2 predefine Cara Tag is as following table.

Table 5. Care Tag

Bit 1	Bit0
Stag	Ctag

Just like the data/mask fields, for those "don't care" bit, users can set the mask bit to 0. For example, user can set this field to filter those packets which are CVLAN tagged.

Care Tag

Bit 1	Bit 0
0	1
0	1

Figure 6. Sample Configuration of Care Tag

Each ACL rule includes an active port mask field. Users can use this field to specify the activated ports for this rule. For those ports which are not set in port mask field in s specified rule, no packets received in those ports will be matched to the rule.

Each rule also include a NOT bit. By setting this bit, the comparison result of ACL rule will be converted. That is, un-matched packet would be assigned a "matched" result and a matched packet will be considered as "un-matched". This feature can be used to filter those "un-matched" packet.

Table 6. Invert bit

Field Name	Description
NOT	Inverse the ACL comparison result

6. Range Check

The basic operation of ACL is to compare the data on specified position of packets and the data/mask in a rule. If users need to filter a range of data at packets, this kind data & mask pair may not meet users' requirement. ACL function supports 4 different kind of Range Check for a range filtering requirement. They are Packet length Range Check, VID Range Check, IP Range Check and L4 port Range Check.

The Packet Length Range Check (with 8 entries) supports reversed setting. The configuration of Packet Length Range Check is listed below:

Table 7. Packet Length Range Check

Field Name	Bits	Description
PKTLEN_UPPER	14	Packet length range upper bound
PKTLEN_LOWER	14	Packet length range lower bound
PKTLEN_REVERSE	1	Reverse operation 0: Don't reverse the comparison result 1: reverse the comparison result

The VID Range Check (with 16 entries) supports both CVID and SVID. The configuration of VID Range Check is listed below:

Table 8. VID Range Check

Field Name	Bits	Description
VID_UPPER	12	VID range upper bound
VID_LOWER	12	VID range lower bound
TYPE	2	Range checking data type 0x0:non-valid 0x1:CVLAN VID range check 0x2:SVLAN VID range check 0x3:reserved

The IP Range Check (with 16 entries) supports both destination and source IP address. It can be also used to filter the last 32 bits of IPv6 destination and source IP address. The configuration of IP Range Check is listed below:

Table 9. IP Range Check

Field Name	Bits	Description
IP_UPPER	64	IP range upper bound (LSB 32 bits is valid only for IPv4 IP address)
IP_LOWER	64	IP range lower bound (LSB 32 bits is valid only for IPv4 IP address)
TYPE	3	Range checking data type 0:non-valid 1:IPv4 SIP range check for outer IP header 2:IPv4 DIP range check for outer IP header 3:IPv6 SIP range check 4:IPv6 DIP range check 5:IPv4 SIP range check for inner IP header

	6:IPv4 DIP range check for inner IP header 7:reserved

The L4 Port Range Check (with 16 entries) supports both TCP and UDP packet. Users can specify the destination ports or source ports to be filtered. The configuration of L4 Port Range Check is listed below:

Table 10. L4 Port Range Check

Field Name	Bits	Description
PORT_UPPER	16	TCP/UDP port range upper bound
PORT_LOWER	16	TCP/UDP port range lower bound
TYPE	2	Range checking data type 0x0:non-valid 0x1:source port range check 0x2:destination range check 0x3:reserved

If Users want to filter a range of VID, IP or L4 Port, please specify the range type in a field of template and setup a rule to point to this template. The index of Range Check entry is specified in one of the 8 data/mask field in ACL rule. Each bit in that field represents an entry of Range Check. That is, one ACL rule can specify multiple Range Checks. An example of ACL rule which filters 2 L4 Port Range Checks is as following:

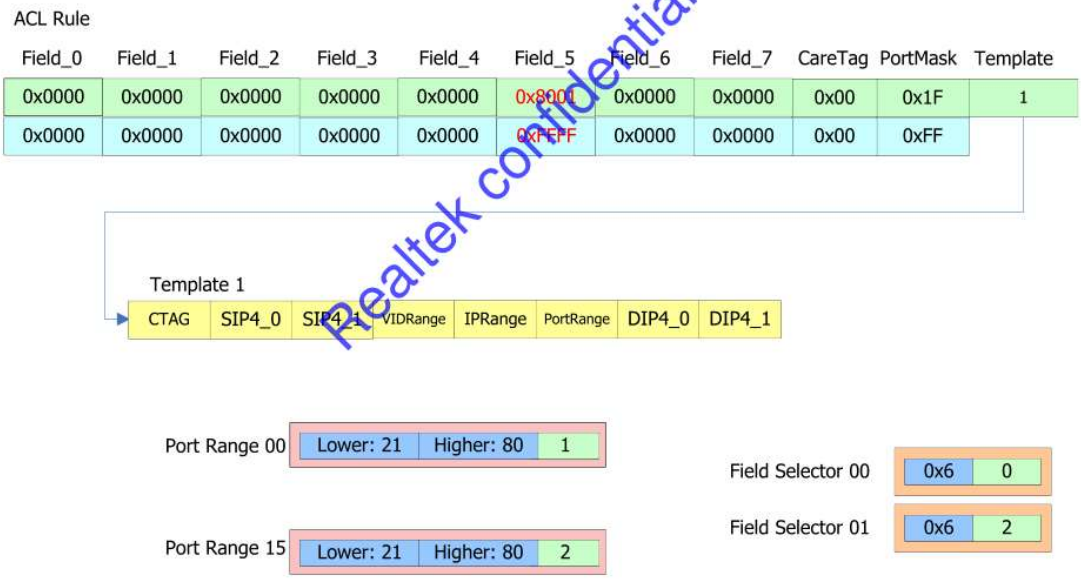

Figure 7. Sample Configuration of Port Range

This configuration uses index 0 & 15 of L4 Port Range Check. Field selector 0 & 1 is also configured to TCP source & destination port(or using default). Field 2 of template is set as Port Range type which means that the field 5 of ACL rule will be decoded as index of L4 Port Range Check. Bit 15 & 0 are set in field 5 of ACL rule. It indicates that entry 15 & 0 of L4 Port Range Check will be checked.

7. Action

There are 128 entries in Action table and these entries are one by one mapped to ACL Rule Table directly. For example, if a packet matched ACL rule 41, the action specified in entry 41 of Action table will be applied to this packet.

6 kinds of action can be specified in an action entry.

Table 11. Action Table

Action 5	Action 4	Action 3	Action 2	Action 1	Action 0
Interrupt and Classification	Forwarding	Policing or Logging	Priority action	SVLAN action	CVLAN action

7.1. Interrupt and Classification latch

Action 5 is Interrupt and Classification. If Action 5 is set in an entry of Action Table, the following configuration can be configured.

Table 12. Interrupt & PON Control

Field Name	Bits	Description
ACLINT	1	ACL interrupt
CFHITLATCH	1	Latch hit ACL index for classification check pattern 0b0:disable 0b1:enable

ACLINT is the register used to control ACL interrupt action. A hardware interrupt will be triggered when all the following situations are true.

1. A matched packet is received.
2. ACLINT is set to 1
3. The configuration of ACL interrupt in interrupt system configuration is enabled.

7.2. Forwarding

Forwarding action is used to change the forwarding behavior of a matched packet. The following configuration can be set.

Table 13. Forwarding Configuration

Field Name	Bits	Description
FWDACT	2	ACL forward decision 0x0: Forward within ACLPMSK egress ports only 0x1: Redirect frame with ACLPMSK 0x2: Ingress mirror to ACLPMSK

		0x3: Trap to CPU
ACLPMSK	11	Port Mask/TC_TOS remarking modify mask bits

If FWDACT is 0x0, the destination port mask of a matched packet will be AND with ACLPMSK. The packet will be forwarded to destination port mask and still follow the VLAN tag/untag setting.

If FWDACT is 0x1, the ACLPMSK is the destination port mask of a matched packet no matter the result of lookup in LUT. Setting FWDACT to 1 and ACLPMSK to 0x00 can drop a matched packet.

If FWDACT is 0x2, the matched packets will be ingress mirrored to ACLPMSK. This mirror is an ingress mirror which means that the packet format would be kept.

If FWDACT is 0x3, the matched packet is trapped to CPU.

7.3. Priority Assignment

Priority assignment is the action which used to assign a new priority for matched packets, change the DSCP field in IP header and change the 1p priority in a VLAN tag. Once the Action 2 of the entry of Action Table is set, the following configuration can be set.

Table 14. Priority Assignement Configuration

Field Name	Bits	Description
PRIACT	3	0x0: ACL Priority 0x1: DSCP Remarking 0x2: 1P Remarking 0x3: Policing 0x4: Logging 0x5: Bandwidth metering 0x6: TC_TOS Remarking
PRIDX	8	ACL priority(0x0-3bits)， DSCP(0x1-6bits)， 1P Priority(0x2-3bits)，Shared meter for policing(0x3-5bits) or Counter MIB index Logging(0x4-5bits), TC_TOS remarking change data bits

If PRIACT is 0x0, the last 3 bit of PRIDX will be assigned to the matched packets as its internal priority.

If PRIACT is 0x1 and the matched packets are IP packet, the DSCP field will be changed according to PRIDX.

If PRIACT is 0x2 and the matched packets are VLAN tagged packet, the 1p priority field will be changed according to the last 3 bits of PRIDX.

If PRIACT is 0x3, the PRIDX will be used as shared meter index and this action is treated as policing.

If PRIACT is 0x4, the PRIDX will be used as MIB index and this action is treated as logging. This feature will be discussed in section 7.6.

If PRIACT is 0x5, the PRIDX will be used as meter index and this action is treated as bandwidth metering. This feature will be discussed in section 7.6.

If PRIACT is 0x6, the PRIDXI will be used as TC/TOS remarking data mask and combine ACLPMSK as remarking care mask for TC/TOS remarking action.

7.4. SVLAN Action

SVLAN Action is used to assign the SVLAN to ingress packet. By setting the Action 1 in the entry of Action Table, the following configuration can be set.

Table 15. SVLAN Action Configuration

Field Name	Bits	Description
SACT	3	0x0:Ingress SVLAN action 0x1:Egress SVLAN action 0x2:Using CVID 0x3:Policing 0x4:Logging 0x5:1P remark 0x6:DSCP remark 0x7:Bandwidth metering
SVID_SACT	12	SVID of SVLAN member configuration, Shared meter for policing, Counter MIB index for Logging, DSCP remarking or 1P Priority remarking

If SACT is 0x0, the downstream matched packet will be assigned a new SVID according to SVID_SACT.

If SACT is 0x1, the egress SVID in S-tag of a matched packet will be changed according to SVID_SACT.

If SACT is 0x2, the egress SVID in S-tag of upstream matched packets will be its original CVID. This action doesn't work on untagged upstream packet.

If SACT is 0x3, the SVID_SACT will be used as shared meter index and this action is treated as policing.

If SACT is 0x4, the SVID_SACT will be used as MIB index and this action is treated as logging. This feature will be discussed in section 7.6.

If SACT is 0x5 and the matched packets are VLAN tagged packet, the 1p priority field will be changed according to the last 3 bits of SVID_SACT.
If SACT is 0x6 and the matched packets are IP packet, the DSCP field will be changed according to SVID_SACT.

If SACT is 0x7, the SVID_CACT will be used as meter index and this action is treated as bandwidth metering. This feature will be discussed in section 7.6.

7.5. CVLAN Action

Like SVLAN Action, CVLAN Action is used to assign CVID for matched packet. By setting the Action 0 in the entry of Action Table, the following configuration can be set.

Table 16. CVLAN Action Configuration

Field Name	Bits	Description
CACT	3	0x0:Ingress CVLAN action 0x1:Egress CVLAN action(replace egress CVID only, both member and untag set is from ingress VID) 0x2:Using SVID(SVLAN downstream only) 0x3:Policing 0x4:Logging 0x5:1P remark 0x6:Bandwidth metering
CVID_CACT	12	CVID of CVLAN member configuration, meter for policing/bandwidth metering, Counter MIB index for Logging, DSCP remarking or 1P Priority remarking

If CACT is 0x0, the CVID of matched packet will be assigned according to the entry pointed by CVID_CACT.

If CACT is 0x1, the CVID of VLAN tag in matched packet will be changed according to the entry pointed by CVID_CACT.

If CACT is 0x2, the CVID of VLAN tag in matched packet will be changed according to the original SVID.

If CACT is 0x3, the CVID_CACT will be used as shared meter index and this action is treated as policing.

If CACT is 0x4, the CVID_CACT will be used as MIB index and this action is treated as logging. This feature will be discussed in section 7.6.

If CACT is 0x5 and the matched packets are IP packet, the DSCP field will be changed according to CVID_CACT.

If CACT is 0x6, the CVID_CACT will be used as meter index and this action is treated as bandwidth metering. This feature will be discussed in section 7.6.

7.6. Policing, Logging and Bandwidth metering

There are 48 shared meters in switch. The shared meter is used to control the ingress rate of matched packets. Each entry of Action table can refer to at most 4 share meters (From Action 3 ~ Action 0). The matched packets can be passed only when the ingress rate is under all shared meters referred by this entry of Action Table.

Table 17. Shared Meter Index Configuration

Field Name	Bits	Description
POLICACT	2	0x0:Policing 0x1:Logging 0x2:Bandwidth Metering

		0x3:1P Remarking
METER_INDEX	6	Shared meter for policing, Counter MIB index for Logging

There are also 64 32-bits logging counter or 32 64-bits logging counter can be used. Each logging counter can be bytes or packets counter based users' configuration. The bytes count or packets counts of matched packets will be increased in the logging counter pointed by METER_INDEX.

The logging counters 0 ~ 7 are organized as following. 2 32-bits logging counter forms a 64-bits logging counter.

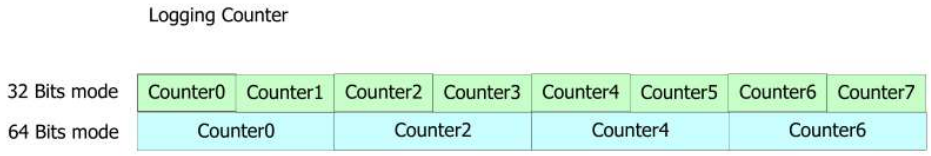

Figure 8. Sample Configuration of Logging Counter

In 64 Bits mode, setting logging counter index to {1, 3, 5,…63} is illegal and the configuration will be ignored.

Different between policing and bandwidth metering is drop action. VID 100 and VID 200 streams are ingress at the same time and with different egress priority and usage. VID 100 stream is higher egress priority and VID 200 stream is shared the same bandwidth with VID 100 stream. As below example, VID 200 stream is policed at egress rate 10Mpbs by meter 1 and shared with VID 100 stream. So, if VID 100 stream is policed at egress 1Mpbs by meter 0, then VID 200 stream can only use remained 9Mbps.

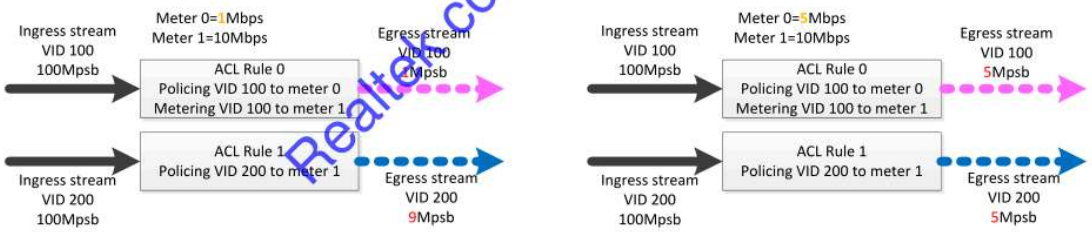

Figure 9. Sample Configuration of Bandwidth Metering Usage

7.7. Multiple Rules Action

ACL function supports multiple rules action if the fields which users want to filter are more the 8 fields. Those ACL rules should be written in continuous position in ACL table. The maximum number rules to forms multiple rules action is 4 rules. The first enabled action of those rules will be taken. For example, users used ACL Rule 0 ~ Rule 3 to form a multiple rules action. The organization of these 4 rules and the actions would be taken is give in the following Figure.

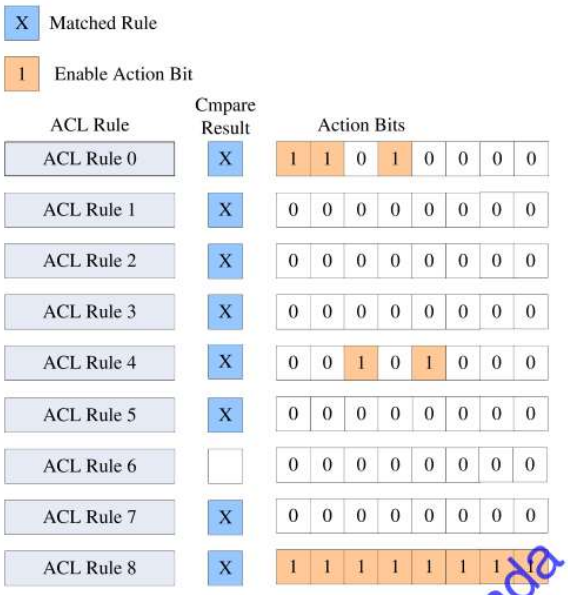

Figure 10. Sample Configuration of Multiple Rules Action

8. API

Realtek API provides a series of interface to let users setup the ACL function without writing register and table directly. This section will discuss these APIs and gives the example.

8.1. Initialization

rtk_acl_init is the first API users should call before setup any configuration. This API will disable the ACL function at all ports and set the un-match action as "Forward". This API also set a default configuration to Templates and Field Selectors. All the default configurations are summarized as following:

Table 18. Default Templates Configuration

	Field_0	Field_1	Field_2	Field_3	Field_4	Field_5	Field_6	Field_7
Template 0	DMAC0	DMAC1	DMAC2	STAG	SMAC0	SMAC1	SMAC2	EtherType
Template 1	CTAG	IPv4SIP0	IPv4SIP1	VIDRange	IPRange	PortRange	IPv4DIP0	IPv4DIP1
Template 2	FS_00	FS_01	FS_02	FS_03	FS_04	FS_05	FS_06	FS_07
Template 3	IP4SIP0 _INNER	IP4SIP1 _INNER	IP4DIP0 _INNER	IP4DIP1 _INNER	TOSPROTO_INNER	CTAG	EXT _MASK	FRAME_ TYPE_TAGS
Template 4	IPv4SIP0	IPv4SIP1	IPv4DIP0	IPv4DIP1	TOS_PRO	TC_NH	PLRange	FRAME_ TYPE_TAGS
Template 5	IPv6SIP0	IPv6SIP1	IPv6SIP2	IPv6SIP3	IPv6SIP4	IPv6SIP5	IPv6SIP6	IPv6SIP7
Template 6	IPv6DIP0	IPv6DIP1	IPv6DIP2	IPv6DIP3	IPv6DIP4	IPv6DIP5	IPv6DIP6	IPv6DIP7
Template	TCP DPORT	TCP SPORT	UDP DPORT	UDP SPORT	IPv4SIP0	IPv4SIP1	IPv4DIP0	IPv4DIP1

Table 19. Default ACL Port Configuration

	State	Un-match
Port 0	Disabled	Forward
Port 1	Disabled	Forward
:	:	:
Port 10	Disabled	Forward

8.2. Templates

Although ***rtk_acl_init*** will setup default template configurations, users can modify these configurations via API ***rtk_template_set***.

Example:

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

rtk_acl_template_t      aclTemplate;

memset(&aclTemplate, 0, sizeof(rtk_acl_template_t));

aclTemplate.index = 2;
aclTemplate.fieldType[0] = ACL_FIELD_DMAC0;
aclTemplate.fieldType[1] = ACL_FIELD_DMAC1;
aclTemplate.fieldType[2] = ACL_FIELD_DMAC2;
aclTemplate.fieldType[3] = ACL_FIELD_SMAC0;
aclTemplate.fieldType[4] = ACL_FIELD_SMAC1;
aclTemplate.fieldType[5] = ACL_FIELD_SMAC2;
aclTemplate.fieldType[6] = ACL_FIELD_CTAG;
aclTemplate.fieldType[7] = ACL_FIELD_STAG;

rtk_acl_template_set(&aclTemplate);
```


8.3. Field Selectors

Like Templates, ***rtk_acl_init*** will setup default Field Selector configurations, users can modify these configurations via API ***rtk_fieldSelect_set***.

Example:

```
/* Configure Field Selector 13 to IP Payload, offset 2 bytes */
int32   ret = RT_ERR_FAILED;
rtk_acl_field_entry_t fieldSel;

fieldSel.index = 13;
fieldSel.format = ACL_FORMAT_IPPAYLOAD;
fieldSel.offset = 2;

if((ret=rtk_acl_fieldSelect_set(&fieldSel)) != RT_ERR_OK)
{
    Return ret;
}
```

8.4. Sample code

This section gives some examples for setup a configuration of ACL rules and actions. The first step of enabling ACL function is to call ***rtk_acl_init*** and ***rtk_acl_igrState_set*** to enable which ports need to enable ACL function. After calling those API, the state of ACL in all ports will be enabled. Thus, the following sample code doesn't include ***rtk_acl_init*** and ***rtk_acl_igrState_set*** in it.

The basic operation of adding an ACL rule can be separated into 2 steps: Add fields into a configuration and write this configuration to H/W register. API ***rtk_acl_igrField_add*** can be used to add fields into a configuration. At this stage, ACL configuration is only a software database to keep all fields information. The following sample code shows that adding STAG filed into a configuration. The content of this field is to filter VID 100 packets.

Example:

```
/* SVID 100 */
int32   ret = RT_ERR_FAILED;
rtk_acl_ingress_entry_t  aclRule;
rtk_acl_field_t          aclField;

memset(&aclRule, 0, sizeof(rtk_acl_ingress_entry_t));
memset(&aclField, 0, sizeof(rtk_acl_field_t));

aclField.fieldType = ACL_FIELD_STAG;
aclField.fieldUnion.l2tag.vid.value = 100;
aclField.fieldUnion.l2tag.vid.mask = 0xFFF;

if((ret = rtk_acl_igrField_add(&aclRule, &aclField))!= RT_ERR_OK)
{
    return ret;
}
```


It is possible to add multiple fields into single configuration. For example, users can add second field to the same configuration above.

Example:

```
/* Ethertype == 0x0806 */
rtk_acl_field_t               aclField2;

memset(&aclField2, 0, sizeof(rtk_acl_field_t));
```

```
aclField2.fieldType = ACL_FIELD_ETHERTYPE;
aclField2.fieldUnion.data.value = 0x0806;
aclField2.fieldUnion.data.mask = 0xFFFF;
aclField2.next = NULL;

if((ret = rtk_acl_igrField_add(&aclRule, &aclField2))!= RT_ERR_OK)
{
    return ret;
}
```

Once all the fields are added to the configuration, users can call API ***rtk_acl_igrRuleEntry_add*** to write the current configuration into H/W register. The action, Care Tag, inversion and the active port mask are also set by this API.

Example:

```
/* Active port mask = 0x04, CTAG packet only, write to rule 1 */
aclRule.valid = ENABLED;
aclRule.index = 1;
aclRule.templateIdx = 0x0

aclRule.careTag.tags[ACL_CARE_TAG_CTAG].value = 1;
aclRule.careTag.tags[ACL_CARE_TAG_CTAG].mask = 1;

aclRule.act.enableAct[ACL_IGR_PRI_ACT] = ENABLED;
aclRule.act.priAct.act = ACL_IGR_PRI_ACL_PRI_ASSIGN_ACT;
aclRule.act.priAct.aclPri = 7;
aclRule.activePorts.bits[0] = 0x4;

if((ret = rtk_acl_igrRuleEntry_add(&aclRule))!= RT_ERR_OK)
{
    return ret;
}
```


The first parameter is rule index. The possible range of this parameter is 0 ~ 63 (mode 0)/0~127 (mode 1). If users specify a position which is occupied by a rule, the original rule will be replaced by new rule without any warring. Knowledge of rules usage doesn't be kept by Realtek API.

8.4.1. Field Selector

If users want to use Field Selector to filter un-predefined field. The Field Selectors and Templates should be configured before adding rules. For example, if users want to filter TCP source port, users can set Field Selectors 13 to IP Payload with offset 0 bytes and setup Field Selectors 13 into Template 3.

Example:

```
/* Configure Field Selector 13 to IP Payload, offset 0 bytes */
int32   ret = RT_ERR_FAILED;
rtk_acl_template_t       aclTemplate;
rtk_acl_field_entry_t    fieldSel;
rtk_acl_ingress_entry_t  aclRule;
rtk_acl_field_t          aclField;

fieldSel.format = ACL_FORMAT_IPPAYLOAD;
fieldSel.index = 13;
fieldSel.offset = 0x0;

if((ret = rtk_acl_fieldSelect_set(&fieldSel))!= RT_ERR_OK)
{
    return ret;
}
/* Configure Template 3 as MAC_DA+MAC_SA+CTAG+Field Selector 13 */
aclTemplate.index = 3;
aclTemplate.fieldType[0] = ACL_FIELD_DMAC0;
aclTemplate.fieldType[1] = ACL_FIELD_DMAC1;
aclTemplate.fieldType[2] = ACL_FIELD_DMAC2;
aclTemplate.fieldType[3] = ACL_FIELD_SMAC0;
aclTemplate.fieldType[4] = ACL_FIELD_SMAC1;
aclTemplate.fieldType[5] = ACL_FIELD_SMAC2;
aclTemplate.fieldType[6] = ACL_FIELD_CTAG;
aclTemplate.fieldType[7] = ACL_FIELD_USER_DEFINED13;

if((ret = rtk_acl_template_set(&aclTemplate))!= RT_ERR_OK)
{
    return ret;
}
/* Setup the rule */
memset(&aclRule, 0, sizeof(rtk_acl_ingress_entry_t));
```

```
memset(&aclField, 0, sizeof(rtk_acl_field_t));
aclField.fieldType = ACL_FIELD_USER_DEFINED13;
aclField.fieldUnion.data.value = 0x1234;/*tcp port*/
aclField.fieldUnion.data.mask = 0xFFFF;

if((ret = rtk_acl_igrField_add(&aclRule, &aclField))!= RT_ERR_OK)
{
    return ret;
}

aclRule.valid = ENABLED;
aclRule.index = 2;
aclRule.templateIdx = 0x3;
/*care tcp only*/
aclRule.careTag.tags[ACL_CARE_TAG_TCP].value = 1;
aclRule.careTag.tags[ACL_CARE_TAG_TCP].mask = 1;

/*drop packets*/
aclRule.act.enableAct[ACL_IGR_FORWARD_ACT] = ENABLED;
aclRule.act.forwardAct.act = ACL_IGR_FORWARD_REDIRECT_ACT;
rtk_switch_portMask_Clear(&aclRule.act.forwardAct.portMask);

rtk_switch_allPortMask_set(&aclRule.activePorts);

if((ret = rtk_acl_igrRuleEntry_add(&aclRule))!= RT_ERR_OK)
{
    return ret;
}
```


Users should pay attention about the relationship between rules and Field Selector. In above example, users directly ask rule to use Field Selector 13 to decode the input value 0x1234 as TCP source port. If Field Selector 13 is changed after the rule setting, this rule still uses Field Selector 13 to decode the input value. Please make sure that all Templates and Field Selectors are configured before setting any rule to prevent un-predicted behavior.

The following fields are the predefined field types. Some of them are not predefined field types. ***rtk_acl_igrRuleEntry_add*** will map them automatically to the assigned Template and Field index.

Table 20. Default ACL Field Type

State
ACL_FIELD_DMAC(ACL_FIELD_DMAC0~2)
ACL_FIELD_SMAC(ACL_FIELD_SMAC0~2)

ACL_FIELD_ETHERTYPE
ACL_FIELD_CTAG
ACL_FIELD_STAG
ACL_FIELD_IPV4_SIP(ACL_FIELD_IPV4_SIP0-1)
ACL_FIELD_IPV4_DIP(ACL_FIELD_IPV4_DIP0-1)
ACL_FIELD_IPV6_SIP(ACL_FIELD_IPV6_SIP0-1)
ACL_FIELD_IPV6_DIP(ACL_FIELD_IPV6_DIP0-1)
ACL_FIELD_VID_RANGE
ACL_FIELD_IP_RANGE
ACL_FIELD_PORT_RANGE
ACL_FIELD_PKT_LEN_RANGE
ACL_FIELD_USER_DEFINED00~15
ACL_FIELD_PATTERN_MATCH

Thus, the following sample gives the same result as above sample code

Example:

```
/* Setup the rule */
memset(&aclRule, 0, sizeof(rtk_acl_ingress_entry_t));
memset(&aclField, 0, sizeof(rtk_acl_field_t));

aclField.fieldType = ACL_FIELD_PATTERN_MATCH;
aclField.fieldUnion.pattern.data.value = 0x1234;
aclField.fieldUnion.pattern.data.mask = 0xFFFF;
aclField.fieldUnion.pattern.fieldIdx = 7;/*assign matched field index*/

if((ret = rtk_acl_igrField_add(&aclRule, &aclField))!= RT_ERR_OK)
{
    return ret;
}

aclRule.valid = ENABLED;
aclRule.index = 2;
aclRule.templateIdx = 0x3;

/*care tcp only*/
aclRule.careTag.tags[ACL_CARE_TAG_TCP].value = 1;
aclRule.careTag.tags[ACL_CARE_TAG_TCP].mask = 1;

/*drop packets*/
aclRule.act.enableAct[ACL_IGR_FORWARD_ACT] = ENABLED;
aclRule.act.forwardAct.act = ACL_IGR_FORWARD_REDIRECT_ACT;
rtk_switch_portMask_Clear(&aclRule.act.forwardAct.portMask);
```

```
rtk_switch_allPortMask_set(&aclRule.activePorts);

if((ret = rtk_acl_igrRuleEntry_add(&aclRule))!= RT_ERR_OK)
{
    return ret;
}
```

8.4.2. Range Check

The Range Check function can also be configured by API. ***rtk_acl_ipRange_set***, ***rtk_acl_vidRange_set,*** ***rtk_acl_portRange_set*** and ***rtk_acl_packetLengthRange_set*** can be used to configure Packet Length, IP, VID and L4 Ports Range Check. For example, the sample code of setting a source IP Range Check at index 2 may be like below:

Example:

```
rtk_acl_rangeCheck_ip_t ipRange;

ipRange.index = 2;
ipRange.type = IPRANGE_IPV4_SIP;
ipRange.lowerIp = 0x01020304;
ipRange.upperIp = 0x05060708;

if((ret = rtk_acl_ipRange_set(&ipRange))!= RT_ERR_OK)
{
    return ret;
}
```


After that, users need to add a rule to use that IP range check to filter packets. It is possible to use multiple range check in a rule. So, the index specified in input data of ACL rule is an index mask. For example, value 0x0004 means that index 2.

Example:

```
memset(&aclRule, 0, sizeof(rtk_acl_ingress_entry_t));
memset(&aclField, 0, sizeof(rtk_acl_field_t));

aclField.fieldType = ACL_FIELD_IP_RANGE;
aclField.fieldUnion.data.value = 0x0004;
aclField.fieldUnion.data.mask = 0x0004;
```

```
if((ret = rtk_acl_igrField_add(&aclRule, &aclField))!= RT_ERR_OK)
{
    return ret;
}

aclRule.valid = ENABLED;
aclRule.index = 3;
aclRule.templateIdx = 0x1;

/*drop packets*/
aclRule.act.enableAct[ACL_IGR_FORWARD_ACT] = ENABLED;
aclRule.act.forwardAct.act = ACL_IGR_FORWARD_REDIRECT_ACT;
rtk_switch_portMask_Clear(&aclRule.act.forwardAct.portMask);

rtk_switch_allPortMask_set(&aclRule.activePorts);

if((ret = rtk_acl_igrRuleEntry_add(&aclRule))!= RT_ERR_OK)
{
    return ret;
}
```


8.4.3. *Raw Data Match*

It is also possible for users to set raw data match ACL rule. In this kind of configuration, users should take care of the field offset in real packet position.

Example:

```
/* Drop IPv6 source IP == 7777:6666:5555:4444:3333:2222:1111:0000 */
int32   ret = RT_ERR_FAILED;
rtk_enable_t state;
rtk_port_t port;
rtk_portmask_t portMask;
rtk_acl_ingress_entry_t  aclRule;
rtk_acl_template_t      aclTemplate;
rtk_acl_field_entry_t   fieldSel;
rtk_acl_field_t   aclField, aclField2;
rtk_acl_field_t   rawField08, rawField09, rawField10;
```

```
rtk_acl_field_t   rawField11, rawField12, rawField13;

if ((ret = rtk_acl_init()) != RT_ERR_OK)
{
    return ret;
}

rtk_switch_allPortMask_set(&portMask);
port = RTK_SWITCH_GET_FIRST_PORT;
while(rtk_switch_nextPortInMask_get(&portMask, &port) == RT_ERR_OK)
{
    if ((ret = rtk_acl_igrState_set(port, ENABLED)) != RT_ERR_OK)
    {
        return ret;
    }
}

fieldSel.format = ACL_FORMAT_IPV6;
fieldSel.index = 8;
fieldSel.offset = 18;

if((ret = rtk_acl_fieldSelect_set(&fieldSel))!= RT_ERR_OK)
{
    return ret;
}

fieldSel.format = ACL_FORMAT_IPV6;
fieldSel.index = 9;
fieldSel.offset = 16;

if((ret = rtk_acl_fieldSelect_set(&fieldSel))!= RT_ERR_OK)
{
    return ret;
}

fieldSel.format = ACL_FORMAT_IPV6;
fieldSel.index = 10;
fieldSel.offset = 14;

if((ret = rtk_acl_fieldSelect_set(&fieldSel))!= RT_ERR_OK)
```

```
{
    return ret;
}

fieldSel.format = ACL_FORMAT_IPV6;
fieldSel.index = 11;
fieldSel.offset = 12;

if((ret = rtk_acl_fieldSelect_set(&fieldSel))!= RT_ERR_OK)
{
    return ret;
}

fieldSel.format = ACL_FORMAT_IPV6;
fieldSel.index = 12;
fieldSel.offset = 10;

if((ret = rtk_acl_fieldSelect_set(&fieldSel))!= RT_ERR_OK)
{
    return ret;
}

fieldSel.format = ACL_FORMAT_IPV6;
fieldSel.index = 13;
fieldSel.offset = 8;

if((ret = rtk_acl_fieldSelect_set(&fieldSel))!= RT_ERR_OK)
{
    return ret;
}
/*set template index 3 as IPv6 SIP address 128 matched bits*/
aclTemplate.index = 3;
aclTemplate.fieldType[0] = ACL_FIELD_IPV6_SIP0;
aclTemplate.fieldType[1] = ACL_FIELD_IPV6_SIP1;
aclTemplate.fieldType[2] = ACL_FIELD_USER_DEFINED08;
aclTemplate.fieldType[3] = ACL_FIELD_USER_DEFINED09;
aclTemplate.fieldType[4] = ACL_FIELD_USER_DEFINED10;
aclTemplate.fieldType[5] = ACL_FIELD_USER_DEFINED11;
aclTemplate.fieldType[6] = ACL_FIELD_USER_DEFINED12;
aclTemplate.fieldType[7] = ACL_FIELD_USER_DEFINED13;
```

```
if((ret = rtk_acl_template_set(&aclTemplate))!= RT_ERR_OK)
{
    return ret;
}
/*set ACL rule as IPv6 SIP addreess*/
osal_memset(&aclRule, 0, sizeof(rtk_acl_ingress_entry_t));

osal_memset(&aclField, 0, sizeof(rtk_acl_field_t));
aclField.fieldType = ACL_FIELD_IPV6_SIP0;
aclField.fieldUnion.ip6.value.ipv6_addr[14]= 0x00;
aclField.fieldUnion.ip6.value.ipv6_addr[15]= 0x00;
aclField.fieldUnion.ip6.mask.ipv6_addr[14]= 0xFF;
aclField.fieldUnion.ip6.mask.ipv6_addr[15]= 0xFF;

if((ret = rtk_acl_igrField_add(&aclRule, &aclField))!= RT_ERR_OK)
{
    return ret;
}

osal_memset(&aclField2, 0, sizeof(rtk_acl_field_t));
aclField2.fieldType = ACL_FIELD_IPV6_SIP1;
aclField2.fieldUnion.ip6.value.ipv6_addr[12]= 0x11;
aclField2.fieldUnion.ip6.value.ipv6_addr[13]= 0x11;
aclField2.fieldUnion.ip6.mask.ipv6_addr[12]= 0xFF;
aclField2.fieldUnion.ip6.mask.ipv6_addr[13]= 0xFF;

if((ret = rtk_acl_igrField_add(&aclRule, &aclField2))!= RT_ERR_OK)
{
    return ret;
}

osal_memset(&rawField08, 0, sizeof(rtk_acl_field_t));
rawField08.fieldType = ACL_FIELD_PATTERN_MATCH;
rawField08.fieldUnion.pattern.data.value = 0x2222;
rawField08.fieldUnion.pattern.data.mask = 0xFFFF;
/*field index to user define 8 as IP6 address bits[47:32]*/
rawField08.fieldUnion.pattern.fieldIdx = 2;
if((ret = rtk_acl_igrField_add(&aclRule, &rawField08))!= RT_ERR_OK)
{
```

```
    return ret;
}

osal_memset(&rawField09, 0, sizeof(rtk_acl_field_t));
rawField09.fieldType = ACL_FIELD_PATTERN_MATCH;
rawField09.fieldUnion.pattern.data.value = 0x3333;
rawField09.fieldUnion.pattern.data.mask = 0xFFFF;
/*field index to user define 9 as IP6 address bits[63:48]*/
rawField09.fieldUnion.pattern.fieldIdx = 3;
if((ret = rtk_acl_igrField_add(&aclRule, &rawField09))!= RT_ERR_OK)
{
    return ret;
}

osal_memset(&rawField10, 0, sizeof(rtk_acl_field_t));
rawField10.fieldType = ACL_FIELD_PATTERN_MATCH;
rawField10.fieldUnion.pattern.data.value = 0x4444;
rawField10.fieldUnion.pattern.data.mask = 0xFFFF;
/*field index to user define 10 as IP6 address bits[79:64]*/
rawField10.fieldUnion.pattern.fieldIdx = 4;
if((ret = rtk_acl_igrField_add(&aclRule, &rawField10))!= RT_ERR_OK)
{
    return ret;
}

osal_memset(&rawField11, 0, sizeof(rtk_acl_field_t));
rawField11.fieldType = ACL_FIELD_PATTERN_MATCH;
rawField11.fieldUnion.pattern.data.value = 0x5555;
rawField11.fieldUnion.pattern.data.mask = 0xFFFF;
/*field index to user define 11 as IP6 address bits[95:80]*/
rawField11.fieldUnion.pattern.fieldIdx = 5;
if((ret = rtk_acl_igrField_add(&aclRule, &rawField11))!= RT_ERR_OK)
{
    return ret;
}

osal_memset(&rawField12, 0, sizeof(rtk_acl_field_t));
rawField12.fieldType = ACL_FIELD_PATTERN_MATCH;
rawField12.fieldUnion.pattern.data.value = 0x6666;
rawField12.fieldUnion.pattern.data.mask = 0xFFFF;
```

```
/*field index to user define 12 as IP6 address bits[111:96]*/
rawField12.fieldUnion.pattern.fieldIdx = 6;
if((ret = rtk_acl_igrField_add(&aclRule, &rawField12))!= RT_ERR_OK)
{
    return ret;
}

osal_memset(&rawField13, 0, sizeof(rtk_acl_field_t));
rawField13.fieldType = ACL_FIELD_PATTERN_MATCH;
rawField13.fieldUnion.pattern.data.value = 0x7777;
rawField13.fieldUnion.pattern.data.mask = 0xFFFF;
/*field index to user define 13 as IP6 address bits[127:112]*/
rawField13.fieldUnion.pattern.fieldIdx = 7;
if((ret = rtk_acl_igrField_add(&aclRule, &rawField13))!= RT_ERR_OK)
{
    return ret;
}

aclRule.valid = ENABLED;
aclRule.index = 4;
aclRule.templateIdx = 0x3;/*using templat index 3*/

/*care IPv6 only*/
aclRule.careTag.tags[ACL_CARE_TAG_IPV6].value = 1;
aclRule.careTag.tags[ACL_CARE_TAG_IPV6].mask = 1;

/*drop packets*/
aclRule.act.enableAct[ACL_IGR_FORWARD_ACT] = ENABLED;
aclRule.act.forwardAct.act = ACL_IGR_FORWARD_REDIRECT_ACT;
rtk_switch_portMask_Clear(&aclRule.act.forwardAct.portMask);

rtk_switch_allPortMask_set(&aclRule.activePorts);

if((ret = rtk_acl_igrRuleEntry_add(&aclRule))!= RT_ERR_OK)
{
    return ret;
}
```

Realtek Semiconductor Corp.
Headquarters
No. 2, Innovation Road II, Hsinchu Science Park,
Hsinchu 300, Taiwan, R.O.C.
Tel: 886-3-5780211 Fax: 886-3-5776047
www.realtek.com

全文阅读已结束，下载本文需要使用

400 积分

下载此文档

阅读了该文档的用户还阅读了这些文档

RTL9607C_LED_Apr

RTL9607C_RTK_Por

RTL9607C_L2_Table

RTL9607C_Storm_Fi

EngNote_RTL9607C note_V01(2019111

Application_Nc

发表评论

验证码： 换一张

匿名评论

提交