ADLINK PacketManager
API Programming Guide

Manual Revision:	0.10 Preliminary
Revision Date:	March 26, 2015
Part Number:	50-1G043-1000

Advance Technologies; Automate the World.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Revision History

Rev	Date	Description
0.10	26/03/2015	Preliminary release

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Preface

Copyright 2015 ADLINK Technology, Inc.

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of the manufacturer.

Disclaimer

The information in this document is subject to change without prior notice in order to improve reliability, design, and function and does not represent a commitment on the part of the manufacturer. In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

Environmental Responsibility

ADLINK is committed to fulfill its social responsibility to global environmental preservation through compliance with the European Union's Restriction of Hazardous Substances (RoHS) directive and Waste Electrical and Electronic Equipment (WEEE) directive. Environmental protection is a top priority for ADLINK. We have enforced measures to ensure that our products, manufacturing processes, components, and raw materials have as little impact on the environment as possible. When products are at their end of life, our customers are encouraged to dispose of them in accordance with the product disposal and/or recovery programs prescribed by their nation or company.

Trademarks

Product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Table of Contents

Revision History **2**
Preface **3**
1 Introduction **7**
1.1 Architecture Overview 7
1.2 Deliverables 8
2 Application API **9**
2.1 Base 9
2.1.1 Macro Definitions 9
2.1.2 Data Structure 11
2.1.3 Enum Type 14
2.1.4 Functions 17
2.2 Static Routing 18
2.2.1 Macro Definitions 18
2.2.2 Enum Type 18
2.2.3 Data Structure 18
2.2.4 Functions 19
2.3 Port 21
2.3.1 Macro Definitions 21
2.3.2 Enum Type 21
2.3.3 Data Structure 23
2.3.4 Function 25
2.4 VLAN 29
2.4.1 Macro Definitions 29
2.4.2 Enum Type 29
2.4.3 Data Structure 29
2.4.4 Functions 30
2.5 Trunk 33
2.5.1 Macro Definitions 33
2.5.2 Enum Type 33
2.5.3 Data Structure 34
2.5.4 Functions 34
2.6 Layer-2 37
2.6.1 Macro Definitions 37
2.6.2 Enum Type 37
2.6.3 Data Structure 37
2.6.4 Functions 38
2.7 Access Control List 38
2.7.1 Macro Definitions 39
2.7.2 Enum Type 39
2.7.3 Data Structure 40

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.7.4 Functions 42
2.8 Spanning Tree Protocol 43
2.8.1 Macro Definitions 43
2.8.2 Enum Type 43
2.8.3 Data Structure 43
2.8.4 Functions 46
3 Hardware Abstraction Layer API 51
3.1 Base 51
3.1.1 Macro Definitions 51
3.1.2 Enum Type 51
3.1.3 Data Structure 51
3.1.4 FunctionS 51
3.2 Init 51
3.2.1 Macro Definitions 51
3.2.2 Enum Type 51
3.2.3 Data Structure 51
3.2.4 FunctionS 52
3.3 Port Information 53
3.3.1 Macro Definitions 53
3.3.2 Enum Type 53
3.3.3 Data Structure 53
3.3.4 Function 54
3.4 Atomic Operation 57
3.4.1 Macro Definitions 57
3.4.2 Enum Type 57
3.4.3 Data Structure 57
3.4.4 Function 57
3.5 Lock 59
3.5.1 Macro Definitions 59
3.5.2 Enum Type 59
3.5.3 Data Structure 59
3.5.4 Functions 60
3.6 LPM 65
3.6.1 Macro Definitions 65
3.6.2 Enum Type 65
3.6.3 Data Structure 65
3.6.4 Functions 67
3.7 ARP 69
3.7.1 Macro Definitions 69
3.7.2 Data Structure 69
3.7.3 Functions 70
3.8 Packet Information 72
3.8.1 Macro Definitions 72

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.8.2 Enum Type 72
3.8.3 Data Structure 72
hal_5_tuple 72
3.8.4 Functions 72
3.9 Packet Transmit and Receive 79
3.9.1 Macro Definitions 79
3.9.2 Enum Type 79
3.9.3 Data Structure 79
3.9.4 Functions 80
Getting Service 83

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1 Introduction

This guide describes how use the ADLINK PacketManager Software application program interface (API).

1.1 Architecture Overview

All the APIs listed in this document are programmed using the hardware architecture, shown in Figure 1. In this figure, the client is running on a standalone server. However, it also can run on the Local Management Processor (LMP) on the blade.

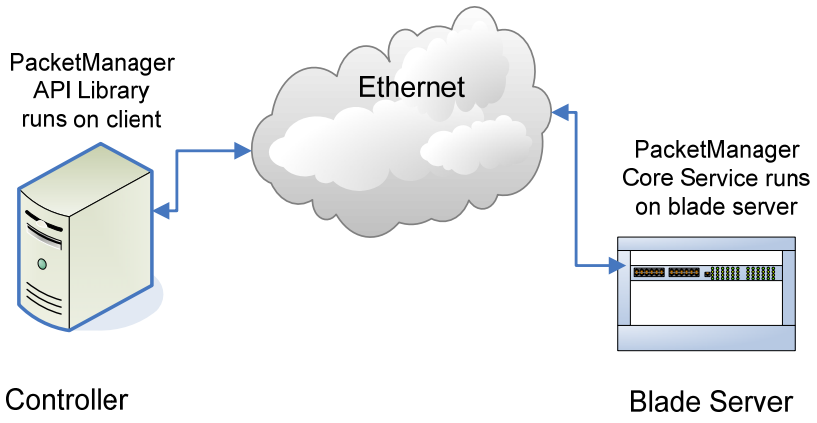


Figure 1 Hardware Architecture for PacketManager API Programming

ADLINK PacketManager Software implements the network connections between server and client so that customers do not have to worry about the connections between them. Client API libraries, header files and relevant documentation are provided for easy programming.

PacketManager is designed to support different hardware architectures, including Intel x86, Cavium, Broadcom and Intel® Red Rock Canyon switch. For switch blades, PacketManager provides application APIs. For processor blades, PacketManager provides both application APIs and hardware abstraction layer (HAL) APIs. The application APIs include the ability to handle operations for protocol stack, ports and other modules. The HAL APIs include the ability to handle operations for ICs that handle packet information, memory and other specific functions. Some APIs are dedicated for specific ICs and others are general purpose.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1.2 Deliverables

The ADLINK PacketManager API Library includes the following software components:

Include:	Header files of the ADLINK PackageManager API library
Lib:	Static library including the API functions listed in this document
Bin:	Binary files for controlling the system
Example:	Application demo to show usage of the APIs provided
ReadMe:	Release notes for different library revisions

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2 Application API

2.1 Base

This section describes the macro, enum, struct and functions for fundamentals. They are defined in the header file “adlink_api_base.h” or “adlink_base.h”. Users do not need to worry about where they are defined. Just include the header file “adlink_api_base.h”.

2.1.1 Macro Definitions

MAC_SIZE

Supported on: Switch blade, processor blade.

MAC address size

PROTOCOL_UDP

Supported on: Switch blade, processor blade.

UDP protocol id.

PROTOCOL_TCP

Supported on: Switch blade, processor blade.

TCP protocol id

PROTOCOL_ICMP

Supported on: Switch blade, processor blade.

ICMP protocol id

ADLINK_DISABLE/ADLINK_ENABLE

Supported on: Processor blade.

Disable and enable flag

ADLINK_TRUE/ADLINK_FALSE

Supported on: Processor blade.

True and false flag

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

ETHER_ADDR_LENGTH

Supported on: Processor blade.

The length of Ethernet address

IPV4_ADDR_LENGTH

Supported on: Processor blade.

The length of IPV4 address.

IPV6_ADDR_LENGTH

Supported on: Processor blade.

The length of IPV6 address.

ADLINK_LITTLE_ENDIAN

Supported on: Processor blade.

Little endian byte order.

ADLINK_BIG_ENDIAN

Supported on: Processor blade.

Big endian byte order.

ADLINK_BYTE_ORDER

Supported on: Processor blade.

Defined the byte order.

ADLINK_GLOBAL

Supported on: Processor blade.

This is used for a global variable for multi-cores, especially used in SE-S mode of Cavium NPUs.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.1.2 Data Structure

adlink_ether_address_t

Supported on: Switch blade, processor blade.

```
typedef struct adlink_ether_address_s
{
    uint8_t addr_bytes[ETHER_ADDR_LENGTH]; /**< Address bytes in transmission
order */
} __attribute__ ((__packed__)) adlink_ether_address_t;
```

Define the Ethernet address format.

adlink_ether_header_t

Supported on: Processor blade.

```
typedef struct adlink_ether_header_s
{
    adlink_ether_address_t dmac; /**< Destination address. */
    adlink_ether_address_t smac; /**< Source address. */
    uint16_t eth_type;          /**< Frame type. */
} __attribute__ ((__packed__)) adlink_ether_header_t;
```

Define the Ethernet head format

adlink_ip_hdr_t

Supported on: Processor blade.

```
typedef struct adlink_ip_hdr
{
#if (ADLINK_BYTE_ORDER == ADLINK_BIG_ENDIAN)
    uint8_t version : 4;
    uint8_t hdr_len : 4;
    uint8_t tos;
    uint16_t length;
    uint16_t id;
    uint16_t frag_flags : 3;
    uint16_t frag_offset : 13;
    uint8_t ttl;
    uint8_t protocol;
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
        uint16_t checksum;
        uint32_t src_ip;
        uint32_t dst_ip;
    #else
        uint8_t hdr_len : 4;
        uint8_t version : 4;
        uint8_t tos;
        uint16_t length;
        uint16_t id;
        uint16_t frag_flags : 3;
        uint16_t frag_offset : 13;
        uint8_t ttl;
        uint8_t protocol;
        uint16_t checksum;
        uint32_t src_ip;
        uint32_t dst_ip;
    #endif

    } adlink_ip_hdr_t;
```

Define the ip head format.

adlink_tcp_hdr_t

Supported on: Processor blade.

```
    typedef struct adlink_tcp_hdr
    {
    #if (ADLINK_BYTE_ORDER == ADLINK_BIG_ENDIAN)
        uint16_t src_port;
            uint16_t dst_port;
            uint32_t seq;
            uint32_t ack;
            uint8_t hdr_len : 4;
            uint8_t reserved1 : 4;
            uint8_t reserved2 : 2;
            uint8_t f_urg : 1;
            uint8_t f_ack : 1;
            uint8_t f_psh : 1;
            uint8_t f_rst : 1;
            uint8_t f_syn : 1;
            uint8_t f_fin : 1;
            uint16_t window_size;
            uint16_t checksum;
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
            uint16_t urg_ptr;
#else
            uint16_t src_port;
            uint16_t dst_port;
            uint32_t seq;
            uint32_t ack;
            uint8_t reserved1 : 4;
            uint8_t hdr_len : 4;
            uint8_t f_fin : 1;
            uint8_t f_syn : 1;
            uint8_t f_rst : 1;
            uint8_t f_psh : 1;
            uint8_t f_ack : 1;
            uint8_t f_urg : 1;
            uint8_t reserved2 : 2;
            uint16_t window_size;
            uint16_t checksum;
          uint16_t urg_ptr;
#endif
} adlink_tcp_hdr_t;
```

adlink_udp_hdr_t

Supported on: Processor blade.

```
typedef struct adlink_udp_hdr
{
    uint16_t src_port;
    uint16_t dst_port;
    uint16_t length;
    uint16_t checksum;
} adlink_udp_hdr_t;
```

Define the UDP head format.

adlink_icmp_hdr_t

Supported on: Processor blade.

```
typedef struct adlink_icmp_hdr
{
    uint8_t     type;
    uint8_t     code;
    uint16_t    checksum;
    union
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
    {
        struct
        {
            uint16_t    id;
            uint16_t    sequence;
        } echo;
        uint32_t    gateway;
        struct
        {
            uint16_t    __unused;
            uint16_t    mtu;
        } frag;
    } un;
}adlink_icmp_hdr_t;
```

Define the ICMP head format.

2.1.3 Enum Type

status_t

Supported on: Switch blade, processor blade.

```
typedef enum
{
    ADLINK_E_None = 0,
    ADLINK_E_INTERNAL = -1,
    ADLINK_E_MALLOC = -2,
    ADLINK_E_PARA = -3,
    ADLINK_E_TABLE_EMPTY = -4,
    ADLINK_E_TABLE_FULL = -5,
    ADLINK_E_NOT_FOUND = -6,
    ADLINK_E_FIND = ADLINK_E_NOT_FOUND,
    ADLINK_E_EXIST = -7,
    ADLINK_E_TIMEOUT = -8 ,
    ADLINK_E_BUSY = -9,
    ADLINK_E_FAIL = -10,
    ADLINK_E_DISABLED = -11,
    ADLINK_E_BADID = -12,
    ADLINK_E_RESOURCE = -13,
    ADLINK_E_CONFIG = -14,
    ADLINK_E_UNAVAIL = -15,
    ADLINK_E_INIT = -16,
    ADLINK_E_SOCKET = -17,
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
        ADLINK_E_RING = -18,
        ADLINK_E_IPC = -19,
        ADLINK_E_RPC = -20,
        ADLINK_E_FILE = -21,
        ADLINK_E_RANGE = -22,
        ADLINK_E_NO_LOCK = -23,
        ADLINK_E_AUTH = -24,
        ADLINK_E_NOMSG = -25,
        ADLINK_E_OVERFLOW = -26,
        ADLINK_E_NOT_SUPPORT = -27,
        ADLINK_E_BAD_MSG = -28,
        ADLINK_E_FUNC_NOT_IMPLMENT = -29,
        ADLINK_E_NOENT = -30,
        ADLINK_E_NOSPC = -31,
        ADLINK_E_NO_TAILQ = -32,
        ADLINK_E_OPEN = -33,
        ADLINK_E_ROUTE = -34,
        ADLINK_E_PTHREAD = -35,
        ADLINK_E_PORT_EXIST   = -36,
        ADLINK_E_PORT_FIND = -37,
        ADLINK_E_OTHER = -127,
        ADLINK_E_SERVER_BASE = -128,
    } status_t;
```

This enum describes the return codes of the APIs. They are defined in “adlink_api_base.h”. Table 1 shows the meaning of each error code.

Return Code	Description
ADLINK_E_None	The routine completed successfully
ADLINK_E_INTERNAL	Some internal operations failed in an unexpected way. This usually indicates that an access to some hardware resource failed
ADLINK_E_MALLOC	Dynamically allocated memory was not available to complete the operation
ADLINK_E_PARA	A parameter to a ADLINK API routines was invalid
ADLINK_E_TABLE_EMPTY	The routine expected a table not to be empty
ADLINK_E_TABLE_FULL	The routine could not add an entry to the table
ADLINK_E_NOT_FOUND ADLINK_E_FIND	The specific entry in a table could not be found
ADLINK_E_EXIST	The specific entry was already in a table
ADLINK_E_TIMEOUT	The routine started an operation that did not complete
ADLINK_E_BUSY	The routine tried to start an operation, but a previous operation was still running

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return Code	Description
ADLINK_E_FAIL	The routine attempted an operation that could not be completed. This may be caused by a hardware failure or a configuration failure
ADLINK_E_DISABLED	The routine could not complete because a required feature was temporarily disabled
ADLINK_E_BADID	An identifier was not valid
ADLINK_E_RESOURCE	The routine needed more resources
ADLINK_E_CONFIG	Invalid configuration
ADLINK_E_UNAVAIL	Feature unavailable
ADLINK_E_INIT	Feature not initialized
ADLINK_E_SOCKET	Socket error
ADLINK_E_RING	RING error
ADLINK_E_IPC	IPC error
ADLINK_E_RPC	RPC error
ADLINK_E_FILE	File operate error
ADLINK_E_RANGE	Result too large or too small
ADLINK_E_NO_LOCK	No locks available
ADLINK_E_AUTH	Authentication error
ADLINK_E_NOMSG	No message of desired type
ADLINK_E_OVERFLOW	Value too large to be stored in data type
ADLINK_E_NOT_SUPPORTED	Not supported
ADLINK_E_BAD_MSG	Bad or Corrupt message
ADLINK_E_FUNC_NOT_IMPLMENT	Function not implemented
ADLINK_E_NOENT	NO such file or such directory
ADLINK_E_NOSPC	No space left on device
ADLINK_E_NO_TAILQ	Uninitialized TAIL Queue
ADLINK_E_OPEN	Open/fopen operation failed
ADLINK_E_ROUTE	Route error
ADLINK_E_PTHREAD	Pthread error
ADLINK_E_PORT_EXIST	Port already exist
ADLINK_E_PORT_FIND	Port not find
ADLINK_E_OTHER	Error except above
ADLINK_E_SERVER_BASE	For a meaningful error code form server, we return it with this value added. So, if the returned code is less than this value, we can subtract this value and get the real return value from server. This is useful for debugging.

Table 1. Return Code Description

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.1.4 Functions

adlink_chip_init ()

Supported on: Processor blade.

Prototype

int adlink_chip_init(char* server_ip)

Description

This function is used to init the switch chip.

Parameters

server_ip [IN] IP address of the server

Return

status_t.

adlink_switch_chip_init ()

Supported on: Switch blade.

Prototype

int adlink_switch_chip_init(UINT32 server_ip)

Description

This function is used to init the switch chip.

Parameters

server_ip [IN] IP address of the server

Return

status_t.

adlink_config_write ()

Supported on: Switch blade, processor blade.

Prototype

int adlink_config_write()

Description

This function is used to write the current configuration into file. After the function is called successfully, the configuration will still be in effect after the system reboots.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters
None

Return
status_t

2.2 Static Routing

This section describes the macro, enum, struct and functions for routing. They are defined in the header file “adlink_api_route.h”.

2.2.1 Macro Definitions

MAX_ROUTE_ENTRY_ONE_TIME_GET

Supported on: Switch blade.

This macro is used for getting route entries from the server. Due to packet size limitations allowed in the network (MTU), it isn't possible to get all the entries at once. This macro defines the maximum number of entries retrievable at one time.

2.2.2 Enum Type

None

2.2.3 Data Structure

adlink_route_info_t

Supported on: Switch blade.

```
typedef struct adlink_route_info
{
    UINT32 ip_addr;      //ip address
    UINT32 ip_mask;      //ip mask
    UINT32 gateway_ip;   // next hop ip address
    UINT16 out_port;     //exit port ID
    UINT16 resv;         // alignment
} adlink_route_info_t;
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.2.4 Functions

adlink_route_add ()

Prototype

status_t adlink_route_add(int unit,UINT32 ip_addr,UINT32 ip_mask,UINT32 gateway_ip,UINT16 metric);

Supported on: Switch blade.

status_t adlink_route_add(uint32_t ip_addr, uint32_t ip_mask, uint32_t gateway_ip, uint16_t metric);

Supported on: Processor blade

Description

This function is used to add a rule entry.

Parameters

ip_addr [IN] ip address.
ip_mask [IN] ip address mask
gateway_ip [IN] gateway ip
metric [IN] route metric

Return

status_t

adlink_route_delete ()

Prototype

status_t adlink_route_delete(int unit,UINT32 ip_addr,UINT32 ip_mask,UINT32 gateway_ip,UINT16 metric);

Supported on: Switch blade.

status_t adlink_route_delete(uint32_t ip_addr, uint32_t ip_mask, uint32_t gateway_ip, uint16_t metric);
Supported on: Processor blade.

Description

This function is used to delete a route entry.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters

ip_addr	[IN] ip address.
ip_mask	[IN] ip address mask.
gateway_ip	[IN] gateway ip
metric	[IN] route metric

Return

status_t.

adlink_route_table_clear ()

Prototype

status_t adlink_route_table_clear ();

Supported on: Switch blade

Description

This function is used to clear all routing table entries that have been added by users.

Parameters

None

Return

status_t.

adlink_route_table_dump ()

Prototype

status_t adlink_route_table_dump (UINT32 ip_last, UINT32 mask_last, int *count, adlink_route_info_t *route_info);

Supported on: Switch blade

Description

This function is used to get all route entry info that has been added.

Parameters

ip_last /mask_last	[IN] These two parameters pass the last route entry that was retrieved. The function call will start retrieving route entry info after these values. Set both of these parameters to 0 the first time this function is called.
Count	[OUT] This parameter passes the number of route entry values retreived. If "Count" is less than MAX_ROUTE_ENTRY_ONE_TIME_GET, then all the route entries have been retrieved. Otherwise, we need to call this function again to continue retrieving route entry info.
route_info	[OUT] This parameter is the pointer for saving the route entry info. It must have enough space to save as many as

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

MAX_ROUTE_ENTRY_ONE_TIME_GET route entries.

Return
status_t.

2.3 Port

This section describes the macro, enum, struct and functions for port. They are defined in the header file “adlink_api_port.h”.

2.3.1 Macro Definitions

None

2.3.2 Enum Type

None

port_speed_t

Supported on: Switch blade

```
typedef enum port_speed_e
{
    PORT_SPEED_10M    = 10,
    PORT_SPEED_100M = 100,
    PORT_SPEED_1G     = 1000,
    PORT_SPEED_10G    = 10000,
} port_speed_t;
```

Define the codes of port speed mode.

port_link_status_t

Supported on: Switch blade

```
typedef enum port_link_status_e
{ /*Heritage*/
    PORT_LINK_STATUS_DOWN              = 0,
    PORT_LINK_STATUS_UP                = 1,
    PORT_LINK_STATUS_FAILED            = 2,
    PORT_LINK_STATUS_REMOTE_FAULT    = 3,
} port_link_status_t;
```

Define the codes of port physical link status.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

port_stp_state_t

Supported on: Switch blade

```
typedef enum port_stp_state_e { /*Heritage */
    PORT_STP_DISABLE = 0, /* Disabled. */
    PORT_STP_BLOCK = 1, /* BPDUs/no learns. */
    PORT_STP_LISTEN = 2, /* BPDUs/no learns. */
    PORT_STP_LEARN = 3, /* BPDUs/learns. */
    PORT_STP_FORWARD = 4, /* Normal operation. */
    PORT_STP_COUNT = 5
} port_stp_state_t;
```

Define the codes of port status for Spanning Tree Protocol.

port_vlan_filter_mode_mask_t

Supported on: Switch blade.

```
typedef enum port_vlan_filter_mode_mask_e { /*Heritage*/
    PORT_VLAN_FILTER_INGRESS     = 0x00000001,
    PORT_VLAN_FILTER_EGRESS      = 0x00000002,
} port_vlan_filter_mode_mask_t;
```

Define the codes of VLAN filter mode in port attribute.

port_rate_limit_mask_t

Supported on: Switch blade

```
typedef enum port_rate_limit_mask_e { /*Heritage*/
    PORT_RATE_DLF                = 0x01,
    PORT_RATE_MCAST              = 0x02,
    PORT_RATE_BCAST              = 0x04,
    PORT_RATE_UCAST              = 0x08,
} port_rate_limit_mask_t;
```

Define the codes of rate limit type mask in port attribute.

nic_config_mask_t

Supported on: Switch blade

```
typedef enum nic_config_mask
{
    NIC_CONFIG_IP_MASK   = 1,      /*Set this bit to flag IP&MASK operation*/
    NIC_CONFIG_STATUS = 2,         /*Set this bit to flag NIC running status operation*/
}nic_config_mask_t;
```

Define the codes of NIC operation.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.3.3 Data Structure

adlink_port_info_t

Supported on: Switch blade

```
typedef struct adlink_port_info_s
{
    int enable;
    int linkstatus;
    int autoneg;
    int speed;
    int duplex;
    UINT32 vlanfilter;
    int untagged_priority;
    UINT16 untagged_vlan;
    int stp_state;
    int pfm;       /*port filter mode for multicast*/
    int loopback;
    int pause_tx;
    int pause_rx;
    int encap_mode;
    int mcast_limit;
    int mcast_limit_enable;
    int bcast_limit;
    int bcast_limit_enable;
    int dlfbc_limit;
    int dlfbc_limit_enable;
    int frame_max;
    UINT32 fault;     /*link fault reason*/
}adlink_port_info_t;
```

Define the entry format of the port attribute table.

adlink_port_info_t

Supported on: Processor blade

```
typedef struct adlink_port_info_s
{
    char name[IFNAMSIZ];
    uint32_t fd;
    uint32_t ip;
    uint32_t mask;
    uint32_t bcast;
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
    adlink_mac_address_t mac;
    adlink_eth_link_t link;
}adlink_port_info_t;
```

adlink_portstat_t

Supported on: Switch blade

```
typedef struct adlink_portstat_s
{
    UINT32 rx_byte_num;
    UINT32 rx_byte_rate;
    UINT32 rx_pkt_num;
    UINT32 rx_pkt_rate;
    UINT32 tx_byte_num;
    UINT32 tx_byte_rate;
    UINT32 tx_pkt_num;
    UINT32 tx_pkt_rate;
}adlink_portstat_t;
```

Define the entry format of the port statistic table.

adlink_set_port_status_cmd_t

Supported on: Switch blade

```
typedef struct adlink_set_port_status_cmd {
    UINT32 port;            /*port ID*/
    UINT32 flag;            /*PORT_ATTR_xxx in port_attribute_t*/
    int status;            /*ENABLE/DISABLE*/
    int autoneg_mode;     /*ENABLE/DISABLE*/
    int duplex_mode;      /*PORT_DUPLEX_xxx in port_duplex_t*/
    int speed_mode;       /*PORT_SPEED_xxx in port_speed_t*/
    int pvid;             /*default 802.1Q vid for untagged packets*/
    int cos;              /*default 802.1Q cos for untagged packets*/
    int mtu;              /*max data transmission unit*/
} adlink_set_port_status_cmd_t;
```

Define the format of parameters used to set port attributes.

adlink_nic_config_set_cmd_t

Supported on: Switch blade

```
typedef struct adl_nic_config_set_cmd {
    char if_name[TAPNAMSIZ];
    UINT32 config_mask;          //nic_config_mask_t
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
    UINT32 ip;
    UINT32 netmask;
    UINT32 status;    //0 down, 1 up
}adlink_nic_config_set_cmd_t;
```

Define the format of parameters used to set NIC configuration.

adlink_nic_config_info_t

Supported on: Switch blade

```
typedef struct adlink_nic_config_info
{
    UINT32 ip;
    UINT32 mask;
    UINT32 status;
}adlink_nic_config_info_t;
```

Define the format of retrieved NIC info.

adlink_nic_config_get_cmd_t

Supported on: Switch blade

```
typedef struct adlink_nic_config_get_cmd {
    char if_name[TAPNAMSIZ];
}adlink_nic_config_get_cmd_t;
```

Define the format of the get NIC info command parameter.

2.3.4 Function

adlink_port_ip_set

Prototype

status_t adlink_port_ip_set(uint8_t port, uint32_t ip, uint32_t mask);

Supported on: Processor blade

Description

Set port ip.

Parameters

port	[IN] logic port id;
ip	[IN] ip to be set;
mask	[IN] ip mask to be set;

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
status_t.

adlink_port_info_get

Prototype
status_t adlink_port_info_get(uint8_t port, adlink_port_info_t* port_info);

Supported on: Switch blade，processor blade.

Description
It’s to get port info.

Parameters
port [IN] logic port id;
port_info [OUT] port status info.

Return
status_t.

adlink_port_statistic_get

Prototype
status_t adlink_port_statistic_get(uint8_t port, int clear, hal_port_statistics_info_t* port_statistic);

Supported on: Processor blade

status_t adlink_port_statistic_get (UINT32 port, adlink_portstat_t *stat);
Supported on: Switch blade

Description
It’s to get port statistic info.

Parameters
port [IN] logic port id;
clear [IN] 1:reset port
0: not reset port
port_statistic/stat [OUT] port statistic info ;

Return
status_t.

adlink_port_statistic_clear

Prototype

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

status_t adlink_port_statistic_clear(uint8_t port, int clear, hal_port_statistics_info_t* port_statistic);

Supported on: Processor blade.

status_t adlink_port_statistic_clear (UINT32 port);
Supported on: Switch blade.

Description
Clear port statistic info.

Parameters
port [IN] logic port id;

Return
status_t.

adlink_dp2cp_statis_get

Prototype
status_t adlink_dp2cp_statis_get(hal_exception_path_stats_t* dp2cp_statis);

Supported on: Processor blade

Description
Get dp2cp statistic info.

Parameters
dp2cp_statis [OUT] dp2cp statistic information;

Return
status_t.

adlink_nic_config_set ()

adlink_nic_config_get()

Prototype
status_t adlink_nic_config_set(adl_nic_config_set_cmd_t *cmd);
status_t adlink_nic_config_get(adl_nic_config_get_cmd_t *cmd, adl_nic_config_info_t *info);

Supported on: Switch blade

Description
The function ‘get’ is used to get configuration of a NIC, including running status and IP address. The function ‘set’ is used to configure a NIC.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters

cmd [IN] NIC name and operation code;

info [OUT] NIC information;

Return

status_t.

adlink_port_pipe_get ()

adlink_port_pipe_set()

adlink_port_pipe_clear()

Prototype

status_t adlink_port_pipe_set(UINT32 src_port,UINT32 dst_port);

status_t adlink_port_pipe_clear(UINT32 src_port);

status_t adlink_port_pipe_get(UINT32 src_port,UINT32 *dst_port);

Supported on: Switch blade

Description

These functions are used to pipe a specific port to another one. “pipe” will directly deliver ingress traffic on a specific port to another one.

Parameters

src_port [IN] Ingress port;

dst_port [IN|OUT] Destination port traffic will be delivered to;

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.4 VLAN

This section describes the macro, enum, struct and functions for VLAN. They are defined in the header file “adlink_api_vlan.h”. The vlan module is supported on switch blade.

2.4.1 Macro Definitions

MAX_VLAN_MAC_CNT

Maximum entry count of MAC based VLAN.

2.4.2 Enum Type

adlink_vlan_tag_mode_t

```
typedef enum adlink_vlan_tag_mode_e{
    VLAN_MODE_UNTAGGED = 0,
    VLAN_MODE_TAGGED = 1,
} adlink_vlan_tag_mode_t;
```

Define the codes of port mode in VLAN.

2.4.3 Data Structure

adlink_vlan_info_t

```
typedef struct adlink_vlan_info_s {
    UINT8    enable;                          /*admin status*/
    UINT8    created;                         /*whether it is created on hardware*/
    UINT8    rsv1[2];                         /*for align purpose*/
    UINT32 pbmp;                                  /*port bitmap in this vlan*/
    UINT32 untag_pbmp;                            /*port bitmap of untagged ports*/
    UINT32 pvid_pbmp;                             /*port bitmap take VID as default vlan*/
    char     name[MAX_VLAN_NAME_SIZE];     /*vlan name. 31 characters at maximum*/
}adlink_vlan_info_t;
```

Define the entry format of port based VLAN table.

adlink_vlan_mac_t

```
typedef struct adlink_vlan_mac_s {
    UINT8    valid;              /*in using or not. VALID/INVALID*/
    UINT8    cos;                 /*priority value. 0-7*/
    UINT8    mac[MAC_SIZE];     /*MAC address*/
    UINT16 vid;                  /*VLAN id. 1-4094*/
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
    UINT8    rsv[2];              /*reserved*/
}adlink_vlan_mac_t;
```

Define the entry format of MAC based VLAN table.

2.4.4 Functions

adlink_vlan_info_get ()

Prototype

status_t adlink_vlan_info_get (int vlan, adlink_vlan_info_t *info);

Description

This function is used to get port based VLAN information.

Parameters

vlan [IN] The VLAN ID to be configed;
info [OUT] VLAN information;

Return

status_t

adlink_vlan_admin_status_set ()

Prototype

status_t adlink_vlan_admin_status_set (int vlan, int status);

Description

This function is used to switch admin status of a VLAN. If the status is disabled, VLAN can be configured but not used. This function is used to created VLANs, set them to ENABLE.

Parameters

vlan [IN] The VLAN ID to be configured;
status [IN] Admin status. Use ‘ENABLE’ or ‘DISABLE’;

Return

status_t

adlink_vlan_create ()

Prototype

status_t adlink_vlan_create (int vlan, char*name);

Description

This function is used to create a new VLAN.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters

vlan [IN] The VLAN ID to be created;

name [IN] Description for new created VLAN;

Return

status_t

adlink_vlan_destroy ()

Prototype

status_t adlink_vlan_destroy (int vlan);

Description

This function is used to destroy an existing VLAN.

Parameters

vlan [IN] The VLAN ID to be destroyed;

Return

status_t.

adlink_vlan_member_add ()

adlink_vlan_member_del ()

Prototype

status_t adlink_vlan_member_add (int vlan, UINT32 port, UINT32 mode);

status_t adlink_vlan_member_del (int vlan, UINT32 port);

Description

These two functions are used to add port into or remove port from a VLAN.

Parameters

vlan [IN] The VLAN ID in which port member is configured.

port [IN] Port ID to be added/removed.

mode [IN] Port is tagged or untagged in the VLAN. Use ‘VLAN_MODE_TAGGED’ or ‘VLAN_MODE_UNTAGGED’

Return

status_t.

adlink_port_default_vlan_set ()

PPROTOTYE

status_t adlink_port_default_vlan_set (int vlan, UINT32 port);

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Description

This function is used to assign a port's default VLAN tag for untagged packets.

Parameters

vlan [IN] The default VLAN ID of a port;

port [IN] The port ID which will be assigned a default VLAN tag;

Return

status_t

adlink_mac_based_vlan_add ()

adlink_mac_based_vlan_del ()

adlink_mac_based_vlan_get ()

Prototype

status_t adlink_mac_based_vlan_add (int vlan, UINT8 *mac, UINT8 cos);
status_t adlink_mac_based_vlan_del (UINT8 *mac);
status_t adlink_mac_based_vlan_get (UINT8 *mac, UINT16 *vlan, UINT8 *cos);

Description

These functions are used to configure MAC base VLAN. MAC address can be associated with a VLAN id.

Parameters

vlan [IN] The VLAN ID which will be assigned to a specified MAC address;

mac [IN] A pointer points to specified MAC address;

cos [IN] Default COS value which will be filled in VLAN tag;

Return

status_t.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.5 Trunk

This section describes the macro, enum, struct and functions for trunk. They are defined in the header file “adlink_api_trunk.h”. The trunk module is supported on switch blade.

2.5.1 Macro Definitions

MIN_TRUNK_ID

MAX_TRUNK_ID

MAX_TRUNK_MEMBER

Trunk group index range and member count limit of a group.

2.5.2 Enum Type

adlink_trunk_psc_t

```
typedef enum adlink_trunk_psc_e
{
    PSC_ZERO      = 0,
    PSC_SA        = 1,
    PSC_DA        = 2,
    PSC_SA_DA     = 3,
    PSC_SIP       = 4,
    PSC_DIP       = 5,
    PSC_SIP_DIP = 6,
}adlink_trunk_psc_t;
```

Define the codes of trunk port selection criteria. Packets will be dispatched to different ports based on the criteria.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.5.3 Data Structure

adlink_trunk_monitor_t

```
typedef struct tm_port_node_s
{
    UINT32 id;              /*port number. start at 1 */
    UINT32 status;          /*link status. 0-link down/1-link up */
}tm_port_node_t;
```

tm_group_node_t

```
typedef struct tm_group_node_s
{
    UINT32 valid;               /*indicate whether group is in monitor. 0-invalid/1-valid */
    UINT32 cnt;                 /*port count in a trunk group*/
    tm_port_node_t item[MAX_TRUNK_MEMBER];      /*port info*/
}tm_group_node_t;
```

adlink_trunk_monitor_t

```
typedef struct adlink_trunk_monitor_s
{
    tm_group_node_t group[MAX_TRUNK_ID];
}adlink_trunk_monitor_t;
```

Define the entry format of port attribute table.

2.5.4 Functions

adlink_trunk_member_add ()

adlink_trunk_member_del ()

Prototype

status_t adlink_trunk_member_add (int trunk_id, UINT32 member_bmp);
status_t adlink_trunk_member_del (int trunk_id, UINT32 member_bmp);

Description

These functions are used to change trunk group member ports.

Parameters

trunk_id [IN] Trunk group id; Start from ‘1’;
member_bmp [IN] Member ports in bitmap;

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
status_t

adlink_trunk_clear ()

Prototype
status_t adlink_trunk_clear ();

Description
This function is used to destroy all created trunk groups.

Parameters
None

Return
status_t.

adlink_trunk_mode_set ()

Prototype
status_t adlink_trunk_mode_set (int trunk_id, int mode);

Description
This function is used to change trunk group port selection criteria.

Parameters
trunk_id [IN] Trunk group id; Start from ‘1’;
mode [IN] Trunk group PSC mode. PSC_XXX;

Return
status_t.

adlink_trunk_info_get ()

Prototype
status_t adlink_trunk_info_get (int tid, adlink_trunk_db_t *info);

Description
This function is used to get trunk group information.

Parameters
tid [IN] Trunk group id; Start from ‘1’;
info [OUT] Trunk group information;

Return
status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

adlink_trunk_monitor_add ()

adlink_trunk_monitor_del ()

adlink_trunk_monitor_get ()

Prototype

status_t adlink_trunk_monitor_add (int group_index);
status_t adlink_trunk_monitor_del (int group_index);
status_t adlink_trunk_monitor_get (adlink_trunk_monitor_t *tm);

Description

These functions are used to control trunk groups to be monitored or not. For groups in monitor, member ports with a link failure will be removed from group.

Parameters

group_index [IN] Trunk group id; Start from ‘1’;
tm [OUT] Trunk group status in monitor;

Return

status_t.

adlink_trunk_rtag7_set ()

adlink_trunk_rtag7_get ()

Prototype

status_t adlink_trunk_rtag7_set (int unit, int mode);
status_t adlink_trunk_rtag7_get (int unit, int *mode);

Description

These functions are used to control the Advanced Hash Mode, an ADLINK-defined hash method utilizing "mode7" in the Broadcom SDK. Advanced Hash Mode can be set to use 4-tuple (SrcIp, DstIp. SrcPort, DstPort) or 5-tuple (SrcIp, DstIp, SrcPort, DstPort, IpProtocol) to generate the hash value. Note that this is a system wide configuration and modification of this mode will affect all trunk groups that are using Advanced Hash Mode.

Parameters

unit [IN] Fabric or Base chip unit;
mode [IN|OUT] 4-tuple/5-tuple;

Return

status_t.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.6 Layer-2

This section describes the macro, enum, struct and functions for Layer-2. They are defined in the header file “adlink_api_l2.h”. The trunk module is supported switch blade.

2.6.1 Macro Definitions

MAC_SIZE

MAC addresses data size in bytes.

ADLINK_L2_xxx

L2 entry flag masks. Flags are special attributes of the MAC entry. For details, please refer to the definitions in “adlink_api_l2. h”

2.6.2 Enum Type

None

2.6.3 Data Structure

adlink_mac_entry_t

```
typedef struct adlink_mac_entry_s
{
    UINT32   port;              /*port ID associated with the entry, start from 1*/
    UINT16   vid;               /*vlan id*/
    UINT8    rsv0[2];           /*reserved*/
    UINT32   flag;              /*entry flag, ADLINK_L2_XXX*/
    UINT8    mac[MAC_SIZE];     /*mac address*/
    UINT8    rsv1[2];           /*reserved*/
} adlink_mac_entry_t;
```

Define the entry format of the MAC table.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.6.4 Functions

adlink_mac_entry_add ()

adlink_mac_entry_del ()

adlink_mac_table_clear ()

Prototype

status_t adlink_mac_entry_add (adlink_mac_entry_t *entry);
status_t adlink_mac_entry_del (adlink_mac_entry_t *entry);
status_t adlink_mac_table_clear ();

Description

These functions are used to change the MAC address table. Add/delete an item or clear all items.

Parameters

entry [IN] Entry that will be added or deleted;

Return

status_t.

adlink_mac_table_dump ()

Prototype

status_t adlink_mac_table_dump (proxy_index_t *start_index, proxy_mac_dump_t *list);

Description

This function is used to dump the entire MAC address table.

Parameters

start_index [IN] Start index of MAC address table;
list [OUT] Retrieved MAC entries;

Return

status_t

2.7 Access Control List

This section describes the macro, enum, struct and functions for Access Control List (ACL). They are defined in the header file “adlink_api_acl.h”. The ACL module is supported switch blade.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.7.1 Macro Definitions

MAX_QUALIFIER_NUM

The ACL supports multiple qualifiers. This macro defines the maximum qualifier number allowed.

2.7.2 Enum Type

adlink_qualifier_type_t

```
typedef enum adlink_qualifier_type
{
    INPORTS = 1,      //port bit mask,data32
    SRCIP,            //data32
    DSTIP,            //data32
    IPPROTOCOL,       //data8
    L4SRCPORT,        //data16
    L4DSTPORT,        //data16
    SRCMAC,           //data48
    DSTMAC,           //data48
    SRCTRUNK,         //data16
    ETHERTYPE,        //data16
    INNERVLAN,        //data16
    OUTERVLAN,        //data16
} adlink_qualifier_type_t;
```

Qualifier Type	Description
INPORTS	Filter for the port ID a packet comes from
SRCIP	Filter for source IP address of a packet
DSTIP	Filter for destination IP address of a packet
IPPROTOCOL	Filter for IP protocol (such as 0x06 for TCP) of a packet
L4SRCPORT	Filter for L4 source port of a packet
L4DSTPORT	Filter for L4 destination port of a packet
SRCMAC	Filter for source MAC address of a packet
DSTMAC	Filter for destination MAC address of a packet
SRCTRUNK	Filter for the trunk ID a packet comes from
ETHERTYPE	Filter for Ethernet type (such as 0x0806 for ARP) of a packet
INNERVLAN	Filter for the inner VLAN ID of a packet
OUTERVLAN	Filter for the outer VLAN ID of a packet

Table 2. Return Code Description

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

adlink_action_type_t

```
typedef enum adlink_action_type
{
    REDIRECTPORT = 1,     //port bit mask, data32, only one port is permitted
    REDIRECTTRUNK,    //trunk id, data16
    DROP,                 //no param
} adlink_action_type_t;
```

Action Type	Description
REDIRECTPORT	Redirect the packet to a port (cpu0, xe0, xe1, xe2…).
REDIRECTTRUNK	Redirect the packet to a trunk
DROP	Drop the packet

Table 3. Action Type Description

2.7.3 Data Structure

adlink_acl_data_t

```
typedef struct adlink_acl_data
{
    UINT8      data128[16];//data128 use, such as ipv6 addr
    UINT64     data64;       //data64 use
    UINT8      data48[6];    //data48 use, such as mac addr
    UINT32     data32;       //data32 use, such as ip addr
    UINT16     data16;       //data16 use, such as l4port
    UINT8      data8;        //data8 use, such as ip protocol
} adlink_acl_data_t;
```

For different qualifiers and actions, there are different data lengths. For example, an IP address is 32 bits long, a MAC address is 48 bits long, and an L4 port ID is 16 bits long. This data structure was defined in order to accommodate these different situations. Careful attention must be paid to refer to the appropriate member according to the actual situation. The appropriate members for existing qualifiers and actions are listed in the tables above.

adlink_acl_qualifier_info_t

```
typedef struct adlink_acl_qualifier_info
{
    adlink_qualifier_type_t      type;//qualifier type
    adlink_acl_data_t              data;//union, use the correct member for different types
    adlink_acl_data_t              mask;//union, use the correct member for different types
} adlink_acl_qualifier_info_t;
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Qualifier Type	Data	mask
INPORTS	data32 (port bit maps,cpu0 is bit 0,xe0 is bit 1,…)	None
SRCIP	data32	data32
DSTIP	data32	data32
IPPROTOCOL	Data8	Data8
L4SRCPORT	Data16	Data16
L4DSTPORT	Data16	Data16
SRCMAC	Data48	Data48
DSTMAC	Data48	Data48
SRCTRUNK	Data16	Data16
ETHERTYPE	Data16	Data16
INNERVLAN	Data16	Data16
OUTERVLAN	Data16	Data16

Table 4. Data Descriptions for Different Qualifier Types

adlink_acl_action_info_t

```
typedef struct adlink_acl_action_info
{
    adlink_action_type_t    type;
    adlink_acl_data_t       param0;//union,use the correct member for different types
    adlink_acl_data_t       param1;//union,use the correct member for different types
} adlink_acl_action_info_t;
```

Action Type	Param0	Param1
REDIRECTPORT	data32 (port bit maps, only one port is allowed, cpu0 is bit 0, xe0 is bit 1,…)	None
REDIRECTTRUNK	Data16	None
DROP	None	None

Table 5. Data Descriptions for Different Action Types

adlink_acl_rule_info_t

```
typedef struct adlink_acl_rule_info
{
    UINT16 priority;        // the priority of the ACL rule
    UINT16 qualifier_num;   //the number of qualifier of the ACL rule
    adlink_acl_qualifier_info_t qualifier[MAX_QUALIFIER_NUM];
                            //record the qualifier info
    adlink_acl_action_info_t action;// record the action info
} adlink_acl_rule_info_t;
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.7.4 Functions

adlink_acl_rule_add ()

Prototype

status_t adlink_acl_rule_add (adlink_acl_rule_info_t *rule_info,UINT16 *rule_id);

Description

This function is used to add an ACL rule.

Parameters

rule_info [IN] the ACL rule info needed to be added.

rule_id [OUT] the pointer for saving the rule ID.

Return

status_t.

adlink_acl_rule_delete ()

Prototype

status_t adlink_acl_rule_delete (UINT16 rule_id);

Description

This function is used to delete an ACL rule.

Parameters

rule _id [IN] the ID of the ACL rule that want to delete.

Return

status_t.

adlink_acl_rule_clear ()

Prototype

status_t adlink_acl_clear ();

Description

This function is used to clear the ACL table in the system. After executing this function, all ACL rules in the system will be lost.

Parameters

None

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

adlink_acl_rule_get ()

Prototype

status_t adlink_acl_rule_get (UINT16 rule_id, adlink_acl_rule_info_t *rule_info);

Description

This function is used to get the detail information of an ACL rule.

Parameters

rule_id [IN] The ID of the rule that want to get information.

rule_info [OUT] The pointer for saving the rule information.

Return

status_t

2.8 Spanning Tree Protocol

This section describes the macro, enum, struct and functions for Spanning Tree Protocol (STP). They are defined in the header file “adlink_api_stp.h”. The STP module is supported on switch blade.

2.8.1 Macro Definitions

MAX_BR_PORTS

This is used to limit the number of ports that can be added to the STP bridge.

2.8.2 Enum Type

None

2.8.3 Data Structure

stp_para_u

```
union stp_para_u{
        int stp_status; /*set stp status on/off */
        int br_pri;      /*bridge priority*/
        int path_cost;   /*port path cost */
        int port_pri;     /*port priority*/
        uint time;        /*forward delay timer or hello timer*/
};
```

Define the stp parameter.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

adlink_stp_para_set_t

```
typedef struct adlink_stp_para_set_s{
        int unit;       /*Fabric or base unit*/
        char bridge_name[32];    /*bridge name that created*/
        int       port_id;      /*port index*/
        union stp_para_u stp_para;    /*stp relate parameter*/
}adlink_stp_para_set_t;
```

Define the stp parameter to set.

port_name

```
struct port_name
{
      char phy_port_name[16]; /*port name*/
};
```

Define the port name.

adlink_br_info_t

```
typedef struct adlink_br_info_s{
    char br_name[32]; /*bridge name*/
    bridge_id_t br_id;    /*bridge id*/
    unsigned char stp_enabled; /*stp state on/off*/
    int port_num;    /*Number of ports*/
    struct port_name port_name_list[MAX_BR_PORTS]; /*Port name list*/
}adlink_br_info_t;
```

Define the bridge info.

adlink_br_list_info_t

```
typedef struct adlink_br_list_info_s{
      int br_num;      /*bridge number */
      adlink_br_info_t br_info[MAX_BR_NUM];    /*bridge info*/
}adlink_br_list_info_t;
```

Define the bridge list info.

time_val

```
struct time_val{
      unsigned int tv_sec;
      unsigned int tv_usec;
};
```

Define the time.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

bridge_stp_info

```
struct bridge_stp_info
{
        bridge_id_t designated_root;    /*designed root bridge id*/
        bridge_id_t bridge_id; /*bridge id */
        unsigned root_path_cost; /*root path cost*/
        struct time_val max_age; /*max age timer*/
        struct time_val hello_time; /*hello timer*/
        struct time_val forward_delay; /*forward delay timer*/
        struct time_val bridge_max_age; /*bridge max age timer*/
        struct time_val bridge_hello_time;/*bridge hello timer*/
        struct time_val bridge_forward_delay; /*bridge forward delay timer*/
        unsigned short    root_port;    /*root port*/
        unsigned char stp_enabled; /*stp state*/
        unsigned char topology_change; /*topology change*/
        unsigned char topology_change_detected; /*topology change detected*/
        struct time_val ageing_time; /*age timer value*/
        struct time_val hello_timer_value; /*hello timer value*/
        struct time_val tcn_timer_value; /*tcn timer value*/
        struct time_val topology_change_timer_value;/*topology change timer value*/
        struct time_val gc_timer_value; /*gc timer value, you can ignore it*/
};
```

Define the stp bridge info.

br_stp_port_info

```
struct br_stp_port_info
 {
        unsigned port_no; /*port number*/
        bridge_id_t designated_root; /*designated root id*/
        bridge_id_t designated_bridge; /*designated bridge id*/
        unsigned short port_id; /*port id*/
        unsigned short designated_port; /*designated port */
        unsigned char priority;/*port priority*/
        unsigned char top_change_ack;/*topology change ack flag*/
        unsigned char config_pending; /*config pending flag*/
        unsigned char state;/*port stp state*/
        unsigned path_cost;/*port path cost*/
        unsigned designated_cost;/*designated cost*/
        struct time_val message_age_timer_value;/*message age timer value*/
        struct time_val forward_delay_timer_value; /*forward delay timer value */
        struct time_val hold_timer_value; /*hold timer value*/
 };
```

Define the STP port information.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

adlink_br_stp_info_t

```
typedef struct adlink_br_stp_info_s{
    int bridge_find;    /*bridge exist flag*/
    struct bridge_stp_info bridge_stp; /*bridge info*/
    int port_num; /*port num*/
    struct port_name br_port_name[MAX_BR_PORTS]; /*bridge port name*/
    struct br_stp_port_info port_info[MAX_BR_PORTS]; /*bridge port info*/
}adlink_br_stp_info_t;
```

Define the bridge and port information.

2.8.4 Functions

adlink_stp_bridge_add()

adlink_stp_bridge_del()

Prototype

status_t adlink_stp_bridge_add(adlink_stp_modify_t *status);
status_t adlink_stp_bridge_del(adlink_stp_modify_t *status);

Description

This function is used to add or delete a bridge.

Parameters

status [IN] A pointer pointing to the STP modify message including bridge name;

Return

status_t.

adlink_stp_itf_add()

adlink_stp_itf_del()

Prototype

status_t adlink_stp_itf_add(adlink_stp_modify_t *status);
status_t adlink_stp_itf_del(adlink_stp_modify_t *status);

Description

This function is used to add or delete the interface from bridge.

Parameters

status [IN] A pointer pointing to the STP modify message include bridge name and port bitmap ;

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
status_t.

adlink_stp_status_set()

Prototype
status_t adlink_stp_status_set(adlink_stp_para_set_t *status);

Description
This function is used to set STP bridge on/off;

Parameters
status [IN] A pointer pointing to the STP parameter set message include bridge and bridge state;

Return
status_t.

adlink_stp_br_pri_set()

Prototype
status_t adlink_stp_br_pri_set(adlink_stp_para_set_t *status);

Description
This function is used to set bridge priority.

Parameters
status [IN] A pointer pointing to the STP parameter set message include bridge name and bridge priority;

Return
status_t.

adlink_stp_fd_time_set()

Prototype
status_t adlink_stp_fd_time_set(adlink_stp_para_set_t *status);

Description
This function is used to set forward delay timer.

Parameters
status [IN] A pointer pointing to the STP parameter set message include bridge name and forward delay timer;

Return
status_t.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

adlink_stp_hello_time_set()

Prototype

status_t adlink_stp_hello_time_set(adlink_stp_para_set_t *status);

Description

This function is used to set hello timer.

Parameters

status [IN] A pointer pointing to the STP parameter set message include bridge name and hello timer;

Return

status_t.

adlink_stp_path_cost_set()

Prototype

status_t adlink_stp_path_cost_set(adlink_stp_para_set_t *status);

Description

This function is used to set port path cost.

Parameters

status [IN] A pointer pointing to the STP parameter set message include bridge name ,port id and path cost value;

Return

status_t.

adlink_stp_port_pri_set()

Prototype

status_t adlink_stp_port_pri_set(adlink_stp_para_set_t *status);

Description

This function is used to set port priority.

Parameters

status [IN] A pointer pointing to the STP parameter set message include bridge name ,port id and port priority;

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

adlink_stp_br_list_info_get()

adlink_stp_br_list_info_print()

Prototype

status_t adlink_stp_br_list_info_get(adlink_br_list_info_t *stat);
status_t adlink_stp_br_list_info_print(adlink_br_list_info_t *stat);

Description

The function is used to get/print the bridge info.

Parameters

status [IN/OUT] A pointer pointing to the STP bridge parameters.

Return

status_t.

adlink_br_stp_info_get()

Prototype

status_t adlink_br_stp_info_get(char *br_name,adlink_br_stp_info_t *stat);

Description

This function is used to get the bridge and port info with bridge name.

Parameters

br_name [IN] bridge name
stat [OUT] bridge info.

Return

status_t.

adlink_br_info_print()

adlink_br_port_info_print()

adlink_br_stp_info_print()

Prototype

status_t adlink_br_info_print(char *br_name, adlink_br_stp_info_t *stat);
status_t adlink_br_port_info_print(adlink_br_stp_info_t *stat);
status_t adlink_br_stp_info_print(char *br_name,adlink_br_stp_info_t *stat);

Description

The function is used to print the bridge info or bridge’s port info or both.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters

br_name	[IN]bridge name.
stat	[IN]bridge and port info.

Return

status_t.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3 Hardware Abstraction Layer API

This section describes the macro, enum, struct and functions for HAL. They are defined in the header file “adlink_hal.h”. HAL APIs are supported on processor blade

3.1 Base

3.1.1 Macro Definitions

The same with application API’s base part

3.1.2 Enum Type

status_t

3.1.3 Data Structure

The same as Application API data structure. Refer to **2.1.2 Data Structure**.

3.1.4 FunctionS

None

3.2 Init

3.2.1 Macro Definitions

None

3.2.2 Enum Type

None

3.2.3 Data Structure

None

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.2.4 FunctionS

hal_sys_init()

Prototype

status_t hal_sys_init(int argc, char **argv);

Description

This function is used to initialize the system. It will perform all hardware related initialization, including CPU, I/O interfaces, and memory.

Parameters

The typical agrc and argv for the application.

Return

status_t

Hal_enter_main_loop()

Prototype

status_t hal_main_loop_launch (int (*f)(void *), void *arg);

Description

This function is used to call the main loop function. The arg parameter is only used for DPDK.

Parameters

f [IN] the function point of the main loop.

Arg [IN] the point of the parameters. It is only used by DPDK.

Return

status_t.

Hal_sys_teardown()

Prototype

status_t hal_sys_teardown() (int argc, char **argv);

Description

This function is used to free system resources. It is the inverse process of hal_sys_init().

Parameters

None.

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.3 Port Information

3.3.1 Macro Definitions

None

3.3.2 Enum Type

None

3.3.3 Data Structure

hal_port_statistics_info_t

```
typedef struct hal_port_statistics_info
{
    uint64_t rx_pkts;
    uint64_t rx_bytes;
    uint64_t rx_err;
    uint64_t rx_bps;
    uint64_t rx_pps;
    uint64_t tx_pkts;
    uint64_t tx_bytes;
    uint64_t tx_err;
    uint64_t tx_bps;
    uint64_t tx_pps;
    uint64_t pre_cycle;
} hal_port_statistics_info_t;
```

The struct of port statistics.

hal_exception_path_stats_t

```
struct hal_exception_path_stats_s {
    /* data plane */
    int64_t dp_rx_ok;
    int64_t dp_tx_ok;
    int64_t dp_rx_err;
    int64_t dp_tx_err;

    /* control plane */
    int64_t cp_rx_ok;
    int64_t cp_tx_ok;
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
        int64_t cp_rx_err;
        int64_t cp_tx_err;
    }__attribute__((__packed__)) hal_exception_path_stats_t;
    Typedef struct hal_exception_path_stats_s hal_exception_path_stats_t;
```

The struct of the statistics for CP←→DP packets sending.

3.3.4 Function

hal_port_num_get()

hal_port_mask_get()

Prototype

uint8_t hal_port_num_get();
status_t hal_port_mask_get(uint32_t*port_mask)

Description

This function is used to get the port count/mask of the running system.

Parameters

port_mask [OUT] logical port mask of the running system.

Return

The count of the port.

hal_port_logic_id_to_phy_id()

hal_port_phy_id_to_logic_id()

Prototype

int hal_port_logic_id_to_phy_id(int log_id)
int hal_port_phy_id_to_logic_id(int phy_id)

Description

These two functions are used to logical port id and physical port id transformation.

Parameters

log_id [IN] the logical port id, such as 0, 1, 2, 3 and so on.
phy_id [IN] the physical port id. For Cavium platform, it is the IPD port id. For DPDK, it is the same as logical id.

Return

The logical port id and physical port id.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_eth_macaddr_get()

Prototype

status_t hal_eth_macaddr_get(int log_port, adlink_mac_address_t* mac);

Description

This function is used to get the mac addr of one port.

Parameters

port_id	[IN] the logical port id.
mac	[OUT] the pointer to save the mac addr.

Return

status_t.

hal_eth_link_get()

Prototype

status_t hal_eth_link_get(int log_port,adlink_eth_link_t* link);

Description

This function is used to get the link status of one port.

Parameters

port_id	[IN] the logical port id.
link	[OUT] the pointer to save the link status.

Return

status_t.

hal_port_statistics_poll()

Prototype

void* hal_port_statistics_poll(void*arg);

Description

This function is used to generate the port statistics of all port. It must be running without stopping. Usually it is used as the function handler of a new thread.

Parameters

None.

Return

None.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Hal_port_statistic_get()

Prototype

status_t Adlink_port_statistics_get(int port_id, int clear, adlink_port_statistics_info_t *stat);

Description

This function is used to get the tx/rx statistics of one port. The application developer can use it to get the pps/bps and total packets/bytes count of one port.

Parameters

port_id [IN] the logical port id.

Clear [OUT] the flag for statistical register clearing. When the assignment is 1, the register will be cleared after reading. Otherwise, the register will not be cleared. As the register is only 32 bits in Octeon platform, it is suggested to set the value to 1.

Return

status_t.

hal_exception_path_stats_get()

Prototype

status_t hal_exception_path_stats_get(hal_exception_path_stats_t* except_path_info);

Description

This function is used to get the statistics of **CP←→DP packets**.

Parameters

except_path_info [IN] the pointer to save the statistics data.

Return

status_t

hal_exception_path_stats_clear()

Prototype

void hal_exception_path_stats_clear();

Description

This function is used to clear the statistics of **CP←→DP packets**.

Parameters

None.

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.4 Atomic Operation

3.4.1 Macro Definitions

None

3.4.2 Enum Type

None

3.4.3 Data Structure

3.4.4 Function

hal_atomic32_get()

Prototype

int32_t hal_atomic32_get(int32_t *ptr);

Description

This function is used to atomic get current value of a 32-bit (aligned) memory location.

Parameters

ptr [IN] Address of memory to get;

Return

The current value of a 32-bit (aligned) memory location

hal_atomic32_set()

Prototype

void hal_atomic32_set(int32_t *ptr, int32_t value);

Description

Atomically sets a 32 bit (aligned) memory location to a value.

Parameters

ptr [IN] Address of memory;

value [IN]value to set

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
None

hal_atomic32_add( )

Prototype
void hal_atomic32_add(int32_t *ptr, int32_t incr);

Description
Atomically adds a signed value to a 32 bit (aligned) memory location.

Parameters
ptr [IN] address in memory to add incr to
incr [IN] amount to increment memory location by (signed)

Return
None

hal_atomic64_get()

Prototype
int64_t hal_atomic64_get(int64_t *ptr);

Description
Get the current value of a 64 bit (aligned) memory location.

Parameters
ptr [IN] Address of memory to get

Return
Returns the current value of a 64 bit (aligned) memory location.

hal_atomic64_set()

Prototype
void hal_atomic64_set(int64_t *ptr, int64_t value);

Description
Set the current value of a 64 bit (aligned) memory location.

Parameters
ptr [IN]Address of memory to get

Return
None

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_atomic64_add()

Prototype

void hal_atomic64_add(int64_t *ptr, int64_t incr);

Description

Atomically adds a signed value to a 64 bit (aligned) memory location.

Parameters

ptr [IN] address in memory to add incr to

incr [IN] amount to increment memory location by (signed)

Return

None

3.5 Lock

3.5.1 Macro Definitions

None

3.5.2 Enum Type

None

3.5.3 Data Structure

hal_spinlock_t

```
typedef struct hal_spinlock
{
    volatile int32_t locked;
} hal_spinlock_t;
```

Define the spin lock format

hal_spinlock_rec_t

```
typedef struct hal_spinlock_rec
{
    hal_spinlock_t lock;
    volatile int32_t user;
    volatile int32_t count;
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
} hal_spinlock_rec_t;
```

Define the recursive spin-lock format

hal_rwlock_lock_t

```
typedef struct hal_rwlock_lock
{
    volatile uint32_t lock;
    volatile uint32_t write_req;
    volatile uint32_t write_comp;
} hal_rwlock_lock_t;
```

Define the read-write lock format

3.5.4 Functions

hal_spinlock_init()

Prototype

void hal_spinlock_init(hal_spinlock_t *lock);

Description

Initialize the spinlock to an unlocked state

Parameters

lock [IN] A pointer to the spinlock

Return

None

hal_spinlock_unlock()

Prototype

void hal_spinlock_unlock(hal_spinlock_t *lock);

Description

Release the spinlock.

Parameters

lock [IN] A pointer to the spinlock.

Return

None

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_spinlock_lock()

Prototype

void hal_spinlock_lock(hal_spinlock_t *lock);

Description

Take the spinlock.

Parameters

lock [IN] A pointer to the spinlock.

Return

None

hal_spinlock_trylock()

Prototype

int hal_spinlock_trylock(hal_spinlock_t *lock);

Description

Try to take the lock but does not spin if lock is not available.

Parameters

lock [IN] A pointer to the spinlock.

Return

1 if the lock is successfully taken.
0 otherwise.

hal_spinlock_is_locked()

Prototype

int hal_spinlock_is_locked(hal_spinlock_t *lock);

Description

Test if the lock is taken.

Parameters

lock [IN] A pointer to the spinlock.

Return

1 if the lock is currently taken.
0 otherwise.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_spinlock_rec_init()

Prototype

void hal_spinlock_rec_init(hal_spinlock_rec_t *lock);

Description

Initialize the recursive spinlock to an unlocked state.

Parameters

lock [IN] A pointer to the spinlock.

Return

None

hal_spinlock_rec_unlock()

Prototype

void hal_spinlock_rec_unlock(hal_spinlock_rec_t *lock);

Description

Release the recursive spinlock.

Parameters

lock [IN] A pointer to the spinlock.

Return

None

hal_spinlock_rec_lock()

Prototype

void hal_spinlock_rec_lock(hal_spinlock_rec_t *lock);

Description

Take the recursive spinlock.

Parameters

lock [IN] A pointer to the spinlock.

Return

None

hal_spinlock_rec_is_locked()

Prototype

int hal_spinlock_rec_is_locked(hal_spinlock_rec_t *lock);

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Description
Return non-zero if the recursive spinlock is currently locked

Parameters
lock [IN] A pointer to the spinlock.

Return
Non-zero if locked

hal_rwlock_init()

Prototype
void hal_rwlock_init(hal_rwlock_lock_t *rwlock);

Description
Initialize the rwlock to an unlocked state.

Parameters
rwlock [IN] A pointer to the rwlock structure.

Return
None

hal_rwlock_read_lock()

Prototype
void hal_rwlock_read_lock(hal_rwlock_lock_t *rwlock);

Description
Perform a reader lock. If a writer is pending, this will wait for that writer to complete before locking.
NOTE: Each thread/process must only lock any rwlock once, or else a deadlock may result.

Parameters
rwlock [IN] A pointer to the rwlock structure.

Return
None

hal_rwlock_read_unlock()

void hal_rwlock_read_unlock(hal_rwlock_lock_t *rwlock);
Description
Perform a reader unlock.

Parameters
rwlock [IN]pointer to rwlock structure

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
None

hal_rwlock_write_lock()

Prototype
void hal_rwlock_write_lock(hal_rwlock_lock_t *rwlock);

Description
Perform a writer lock. Any readers that attempt to get a lock while there are any pending write locks will wait until all writers have completed. Starvation of readers by writers is possible and must be avoided by the application.

Parameters
rwlock [IN]pointer to rwlock structure

Return
None

hal_rwlock_write_unlock()

Prototype
void hal_rwlock_write_unlock(hal_rwlock_lock_t *rwlock);

Description
Perform a writer unlock.

Parameters
rwlock [IN]pointer to rwlock structure

Return
None

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.6 Longest Prefix Match (LPM)

3.6.1 Macro Definitions

HAL_LPM_NAMESIZE

Max number of characters in LPM name.

HAL_LPM_MAX_DEPTH

Maximum depth value possible for IPv4 LPM.

HAL_LPM_RULE_KEY_LEN

Length of key for LPM rule

HAL_LPM_TBL24_NUM_ENTRIES

Internal total number of tbl24 entries.

HAL_LPM_TBL8_GROUP_NUM_ENTRIES

Internal number of entries in a tbl8 group.

HAL_LPM_TBL8_NUM_GROUPS

Internal total number of tbl8 groups in the tbl8.

HAL_LPM_TBL8_NUM_ENTRIES

Internal total number of tbl8 entries.

3.6.2 Enum Type

None

3.6.3 Data Structure

hal_lpm_tbl24_entry_t

```
typedef struct hal_lpm_tbl24_entry
{
    /* Stores key or group index (i.e. gindex)into tbl8. */
    union{
        uint8_t key[HAL_LPM_RULE_KEY_LEN];
        uint8_t tbl8_gindex;
    };
    /* Using single uint8_t to store 3 values. */
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
    uint8_t valid      : 1;          /**< Validation flag. */
    uint8_t ext_entry : 1;           /**< External entry. */
    uint8_t depth      : 6;          /**< Rule depth. */
} hal_lpm_tbl24_entry_t;
```

Internal Tbl24 entry structure.

hal_lpm_tbl8_entry_t

```
typedef struct hal_lpm_tbl8_entry
{
    uint8_t key[HAL_LPM_RULE_KEY_LEN];
    /* Using single uint8_t to store 3 values. */
    uint8_t valid       : 1;          /**< Validation flag. */
    uint8_t valid_group : 1;          /**< Group validation flag. */
    uint8_t depth       : 6;          /**< Rule depth. */
} hal_lpm_tbl8_entry_t;
```

Internal Tbl8 entry structure.

hal_lpm_rule_t

```
typedef struct hal_lpm_rule
{
    uint32_t ip;                                /**< Rule IP address. */
    uint8_t key[HAL_LPM_RULE_KEY_LEN];  /**< Rule key. */
} hal_lpm_rule_t;
```

Internal Rule structure.

hal_lpm_rule_info_t

```
typedef struct hal_lpm_rule_info
{
uint32_t used_rules;          /**< Used rules so far. */
uint32_t first_rule;          /**< Indexes the first rule of a given depth. */
} hal_lpm_rule_info_t;
```

Internal contains metadata about the rules table.

hal_lpm_t

```
typedef struct hal_lpm
{
    char name[HAL_LPM_NAMESIZE];        /**< Name of the lpm. */
    int mem_location;
    uint32_t max_rules;                 /**< Max. balanced rules per lpm. */
    hal_rwlock_lock_t lpm_lock;     /**LPM lock. */
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
        hal_lpm_rule_info_t rule_info[HAL_LPM_MAX_DEPTH]; /**< Rule info table. */
        hal_lpm_tbl24_entry_t tbl24[HAL_LPM_TBL24_NUM_ENTRIES] \
        __hal_cache_aligned;       /**< LPM tbl24 table. */
        hal_lpm_tbl8_entry_t tbl8[HAL_LPM_TBL8_NUM_ENTRIES] \
        __hal_cache_aligned;       /**< LPM tbl8 table. */
        hal_lpm_rule_t default_rule_tbl \
        __hal_cache_aligned;       /**< LPM default rule. */
        hal_lpm_rule_t rules_tbl[0] \
        __hal_cache_aligned;       /**< LPM rules. */
    } hal_lpm_t;
```

Internal LPM structure.

3.6.4 Functions

hal_lpm_create

Prototype

hal_lpm_t *hal_lpm_create(const char *name, int socket_id, int max_rules);

Description

This function is used to allocate memory for LPM object.

Parameters

name [IN]The lpm table name;
socket_id [IN] NUMA socket ID for LPM table memory allocation;
max_rules [IN] Maximum number of LPM rules;

Return

Handle to LPM object on success, NULL on an err values.

hal_lpm_add

Prototype

status_t hal_lpm_add(hal_lpm_t* lpm, uint32_t ip, uint8_t depth, uint8_t* key, uint32_t key_len);

Description

Add a rule to the LPM table.

Parameters

lpm [IN] LPM object handle;
ip [IN] IP of the rule to be added to the LPM table
depth [IN] Depth of the rule to be added to the LPM table

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

key [IN] key of the rule to be added to the LPM table
key_len [IN] key length of the rule

Return
See the description of **status_t**.

hal_lpm_delete

Prototype
status_t hal_lpm_delete(hal_lpm_t *lpm, uint32_t ip, uint8_t depth);

Description
Delete a rule from the LPM table.

Parameters
lpm [IN] LPM object handle;
ip [IN] IP of the rule to be deleted from the LPM table;
depth [IN] Depth of the rule to be deleted from the LPM table;

Return
status_t.

Prototype
status_t hal_lpm_lookup(hal_lpm_t *lpm, uint32_t ip, uint8_t* key, uint32_t key_len);

Description
Lookup an ip into the LPM table.

Parameters
lpm [IN] LPM object handle;
ip [IN] IP to be looked up in the LPM table;
key [IN] key of the most specific rule found for IP (valid on lookup hit only);
key_len [IN] key length of the most specific rule for key;

Return
See the description of **status_t** in Section 1.

Prototype
status_t hal_lpm_table_dump(hal_lpm_t *lpm);

Description
Dump lpm content.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters

lpm [IN] LPM object handle;

Return

See the description of **status_t**.

3.7 Address Resolution Protocol (ARP)

3.7.1 Macro Definitions

HAL_ARP_KEY_ENTRIES

Internal total number of ARP table entries.

HAL_ARP_TABLE_NAME_LEN

Max number of characters in ARP name.

3.7.2 Data Structure

hal_arp_key_t

```
typedef struct hal_arp_key_s
{
    uint32_t ip;                    /**< Key IP address. */
    adlink_ether_address_t mac;     /**< Key mac address. */
} hal_arp_key_t;
```

Internal Key structure

hal_arp_table_t

```
typedef struct hal_arp_table_s
{
    char name[HAL_ARP_TABLE_NAME_LEN];      /**< Name of the arp. */
    uint16_t key_max_rules;                 /**< Max. balanced rules per arp. */
    hal_rwlock_lock_t key_lock[HAL_ARP_KEY_ENTRIES];    /** key lock*/
    hal_arp_key_t arp_key[0];                           /**arp default key. */
} hal_arp_table_t;
```

Internal ARP structure.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.7.3 Functions

hal_arp_table_create

Prototype

hal_arp_table_t* hal_arp_table_create(const char* arp_table_name, uint16_t arp_key_num);

Description

This function is used to allocate memory for arp object.

Parameters

arp_table_name [IN] the arp table name;
arp_key_num [IN] Maximum number of ARP rules;

Return

Handle to ARP object on success, NULL on an err values.

hal_arp_key_find_by_ip

Prototype

int hal_arp_key_find_by_ip(hal_arp_table_t* arp_table, uint32_t ip);

Description

Lookup a key index in arp table.

Parameters

arp_table [IN] an arp object handle;
ip [IN] ip to be found in arp table;

Return

Return ip index.

hal_arp_key_add

Prototype

status_t hal_arp_key_add(hal_arp_table_t* arp_table, uint32_t ip, adlink_ether_address_t *mac);

Description

Add a key to arp table.

Parameters

arp_table [IN] an arp object handle;
ip [IN] ip to be found in arp table;
mac [OUT] mac to be added to arp table;

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
status_t

hal_arp_key_delete

Prototype
status_t hal_arp_key_delete(hal_arp_table_t* arp_table, uint32_t ip);

Description
delete an arp rule on arp table

Parameters
arp_table [IN] an arp object handle;
ip [IN] ip to be delete on arp table;

Return
status_t

hal_arp_key_delete_all

Prototype
status_t hal_arp_key_delete_all(hal_arp_table_t* arp_table);

Description
delete all keys from an arp table.

Parameters
arp_table [IN] an arp object handle;

Return
status_t

hal_arp_table_dump

Prototype
status_t hal_arp_table_dump(hal_arp_table_t *arp_table);

Description
Dump the arp table.

Parameters
arp_table [IN] arp table pointer;

Return
status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.8 Packet Information

3.8.1 Macro Definitions

None

3.8.2 Enum Type

None

3.8.3 Data Structure

hal_5_tuple

```
struct hal_5_tuple
{
    uint32_t ip_dst;
    uint32_t ip_src;
    uint16_t port_dst;
    uint16_t port_src;
    uint8_t    proto;
    uint8_t    reserved[3];
} __attribute__((__packed__));
```

Define the 5 tuple structure.

3.8.4 Functions

hal_packet_inport_get ()

Prototype

status_t hal_packet_inport_get (void* ptr, uint16_t* port_id);

Description

This function is used to get the input port of the input packet. For Cavium platform, it is the IPD port number in the WQE.

Parameters

ptr [IN] The pointer to the inport packet;

port_id [OUT] The input port id queried out;

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
status_t.

hal_packet_l2_head_get ()

Prototype
uint8_t* hal_packet_l2_head_get (void* ptr);

Description
This function is used to get the layer 2 header of the input packet.

Parameters
ptr [IN] The pointer to the inport packet;

Return
The packet layer 2 header ptr.

hal_packet_ip_head_get ()

Prototype
uint8_t* hal_packet_ip_head_get (void* ptr);

Description
This function is used to get the ip layer header of the input packet.

Parameters
ptr [IN] The pointer to the inport packet;

Return
The packet ip header ptr.

hal_packet_l4_head_get ()

Prototype
uint8_t* hal_packet_l4_head_get (void* ptr);

Description
This function is used to get the layer 4 header of the input packet.

Parameters
ptr [IN] The pointer to the inport packet;

Return
The layer 4 header of the input packet.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_packet_l7_head_get ()

Prototype

uint8_t* hal_packet_l7_head_get (void* ptr);

Description

This function is used to get the layer 7(application layer) header ptr of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;

Return

The layer 7 header of the input packet.

hal_src_mac_get ()

Prototype

status_t hal_src_mac_get (void* ptr, uint8_t *smac);

Description

This function is used to get the source mac address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
smac [OUT] The output source mac address;

Return

status_t

hal_dst_mac_get ()

Prototype

status_t hal_dst_mac_get(void*ptr, uint8_t *dmac);

Description

This function is used to get the dest mac address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
dmac [OUT] The output dest mac address;

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_ethprotocol_get ()

Prototype

status_t hal_ethprotocol_get (void* ptr, uint16_t* ethproto);

Description

This function is used to get the Ethernet protocol of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
ethproto [OUT] The output Ethernet protocol;

Return

status_t.

hal_src_ip_get ()

Prototype

status_t hal_src_ip_get(void* ptr, uint32_t* sip);

Description

This function is used to get the source ip address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
smac [OUT] The output source ip address;

Return

status_t.

hal_dst_ip_get ()

Prototype

status_t hal_dst_ip_get(void* ptr, uint32_t* dip);

Description

This function is used to get the dest ip address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
smac [OUT] The output dest ip address;

Return

status_t.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_ipprotocol_get ()

Prototype

status_t hal_ipprotocol_get(void* ptr, uint8_t* ippr);

Description

This function is used to get the source ip protocol of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
ippr [OUT] The output ip protocol;

Return

status_t .

hal_l4_src_port_get ()

Prototype

status_t hal_l4_src_port_get(void* ptr, uint16_t* sport);

Description

This function is used to get the layer 4 source port of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
sport [OUT] The output layer 4 source port;

Return

status_t.

hal_l4_dst_port_get ()

Prototype

status_t hal_l4_dst_port_get(void* ptr, uint16_t* dport);

Description

This function is used to get the layer 4 dest port of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
dport [OUT] The output layer 4 dest port;

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_5_tuple_get ()

Prototype

status_t hal_5_tuple_get (void* ptr, struct hal_5_tuple* buf);

Description

This function is used to get the 5 tuple of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
buf [OUT] The output 5 tuple info;

Return

status_t

hal_src_mac_set ()

Prototype

status_t hal_src_mac_set (void* ptr, uint8_t* smac);

Description

This function is used to set the source mac address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
smac [IN] The input source mac address;

Return

status_t

hal_dst_mac_set ()

Prototype

status_t hal_dst_mac_set (void* ptr, uint8_t* dmac);

Description

This function is used to set the dest mac address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
dmac [IN] The input dest mac address;

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_src_ip_set ()

Prototype

status_t hal_src_ip_set (void* ptr, uint32_t ip);

Description

This function is used to set the source ip address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
ip [IN] The input source ip address;

Return

status_t

hal_dst_ip_set ()

Prototype

status_t hal_dst_ip_set (void* ptr, uint32_t ip);

Description

This function is used to set the dest ip address of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
ip [IN] The input dest ip address;

Return

status_t

hal_ipprotocol_set ()

Prototype

status_t hal_ipprotocol_set (void* ptr, uint8_t protocol);

Description

This function is used to set the ip protocol of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;
protocol [IN] The input ip protocol;

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_l4_src_port_set ()

Prototype

status_t hal_l4_src_port_set (void* ptr, uint16_t port);

Description

This function is used to set the layer 4 source port of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;

port [IN] The input layer 4 source port;

Return

status_t

hal_l4_dst_port_set ()

Prototype

status_t hal_l4_dst_port_set (void* ptr, uint16_t port);

Description

This function is used to set the layer 4 dest port of the input packet.

Parameters

ptr [IN] The pointer to the inport packet;

port [IN] The input layer 4 dest port;

Return

status_t

3.9 Packet Transmit and Receive

3.9.1 Macro Definitions

None

3.9.2 Enum Type

None

3.9.3 Data Structure

None

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.9.4 Functions

hal_packet_receive_and_callback ()

Prototype

status_t hal_packet_receive_and_callback(int(*app_process_packet_handle)(void*));

Description

This function is used to receive packets and then call the given handle function to process each packet. For a data plane application, this function should be called in the main loop.

Parameters

app_process_packet_handle [IN] The callback function of processing packet received.

Return

status_t

hal_packet_send_out()

Prototype

status_t hal_packet_send_out(void*ptr, int port_id, int dont_free);

Description

This function is used to send a packet out of the system.

Parameters

ptr [IN] The pointer to the packet to be sent.

Port_id [IN] The port id which port to send the packet out. For Cavium platform, it is the IPD port id.

dont_free [IN] The flag is used to control whether the packet will be freed after sent.

Return

status_t

hal_packet_send_to_cp()

Prototype

status_t hal_packet_send_to_cp(void*ptr);

Description

This function is used to send the packet to cp (control plane).

Parameters

ptr [IN] The pointer to the packet to be sent.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return
status_t

hal_packet_recv_from_cp()

Prototype
status_t hal_packet_recv_from_cp(void **ptr);

Description
This function is used to receive a packet from cp (control plane).

Parameters
ptr [IN] The pointer to the pointer of the packet that received.

Return
status_t

hal_packet_send_to_dp()

Prototype
status_t hal_packet_send_to_dp(void *ptr);

Description
This function is used to send the packet from to dp(data plane).

Parameters
ptr [IN] The pointer of the packet that to be sent.

Return
status_t

hal_packet_recv_from_dp()

Prototype
void * hal_packet_recv_from_dp();

Description
This function is used to receive the packet from DP(data plane).

Parameters
None.

Return
It will return the pointer to the packet received if success, or it will return NULL if fail.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

hal_vnic_send_out()

Prototype

status_t hal_vnic_send_out(int vnic_id, uint8_t*data, int len);

Description

This function is used to send out the data.

Parameters

vnic_id	[IN] The logical port id.
data	[IN] The data to be sent.
len	[IN] The length of the data to be sent.

Return

status_t

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Getting Service

ADLINK Technology, Inc.
Address: 9F, No.166 Jian Yi Road, Zhonghe District
New Taipei City 235, Taiwan
Tel: +886-2-8226-5877
Fax: +886-2-8226-5717
Email: service@adlinktech.com

Ampro ADLINK Technology, Inc.
Address: 5215 Hellyer Avenue, #110, San Jose, CA 95138, USA
Tel: +1-408-360-0200
Toll Free: +1-800-966-5200 (USA only)
Fax: +1-408-360-0222
Email: info@adlinktech.com

ADLINK Technology (China) Co., Ltd.
Address: 300 Fang Chun Rd., Zhangjiang Hi-Tech Park, Pudong New Area
Shanghai, 201203 China
Tel: +86-21-5132-8988
Fax: +86-21-5132-3588
Email: market@adlinktech.com

ADLINK Technology Beijing
Address: Rm. 801, Power Creative E, No. 1, B/D, Shang Di East Rd.
Beijing, 100085 China
Tel: +86-10-5885-8666
Fax: +86-10-5885-8625
Email: market@adlinktech.com

ADLINK Technology Shenzhen
Address: 2F, C Block, Bldg. A1, Cyber-Tech Zone, Gao Xin Ave. Sec. 7
High-Tech Industrial Park S., Shenzhen, 518054 China
Tel: +86-755-2643-4858
Fax: +86-755-2664-6353
Email: market@adlinktech.com

LiPPERT ADLINK Technology GmbH
Address: Hans-Thoma-Strasse 11, D-68163, Mannheim, Germany
Tel: +49-621-43214-0
Fax: +49-621 43214-30
Email: emea@adlinktech.com

ADLINK Technology, Inc. (French Liaison Office)
Address: 6 allée de Londres, Immeuble Ceylan
91940 Les Ulis, France
Tel: +33 (0) 1 60 12 35 66
Fax: +33 (0) 1 60 12 35 66
Email: france@adlinktech.com

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

ADLINK Technology Japan Corporation
Address: KANDA374 Bldg. 4F, 3-7-4 Kanda Kajicho,
Chiyoda-ku, Tokyo 101-0045, Japan
Tel: +81-3-4455-3722
Fax: +81-3-5209-6013
Email: japan@adlinktech.com

ADLINK Technology, Inc. (Korean Liaison Office)
Address: 802, Mointer B/D, 326 Seocho-daero, Seocho-Gu,
Seoul 137-881, Korea
Tel: +82-2-2057-0565
Fax: +82-2-2057-0563
Email: korea@adlinktech.com

ADLINK Technology Singapore Pte. Ltd.
Address: 84 Genting Lane #07-02A, Cityneon Design Centre,
Singapore 349584
Tel: +65-6844-2261
Fax: +65-6844-2263
Email: singapore@adlinktech.com

ADLINK Technology Singapore Pte. Ltd. (Indian Liaison Office)
Address: #50-56, First Floor, Spearhead Towers
Margosa Main Road (between 16th/17th Cross)
Malleswaram, Bangalore - 560 055, India
Tel: +91-80-65605817, +91-80-42246107
Fax: +91-80- 23464606
Email: india@adlinktech.com

ADLINK Technology, Inc. (Israeli Liaison Office)
Address: 27 Maskit St., Corex Building
PO Box 12777
Herzliya 4673300, Israel
Tel: +972-77-208-0230
Fax: +972-77-208-0230
Email: israel@adlinktech.com

ADLINK Technology, Inc. (UK Liaison Office)
Tel: +44 774 010 59 65
Email: UK@adlinktech.com

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000