Texas Instruments
20450 Century Boulevard
Germantown, MD 20874

PA LLD 1.2.1

Software Design Specification (SDS)

Revision A

SDOCM # 00085803

March 6, 2012

Revision Record	
Document Title:	**Software Design Specification**
Revision	Description of Change
A	Initial Release for PA 1.2 • Based on the latest version of PA 1.1 • Update APIs per PA 1.2 requirements • Add APIs per PA 1.1 requirements • Add Firmware action flowcharts • Add Sample code requirements
A	Update after final code review • Clean up data structures • Remove PBIST related sections since it is no longer required.
A	Update per PA 1.2.1 enhancements

Note: Be sure the Revision of this document matches the QRSA record Revision letter. The revision letter increments only upon approval via the Quality Record System.

TABLE OF CONTENTS

1 Scope

This document describes the functionality, architecture, and operation of the Packet Accelerator Low Level Driver.

2 References

The following references are related to the feature described in this document and shall be consulted as necessary.

No	Referenced Document	Control Number	Description
1	PA LLD 1.2 PRD	QRSA 00015477	Product Requirements
2	PA LLD 1.1 SDS	QRSA 00015031	PA 1.1 Software Design Specification

Table 1. Referenced Materials

3 Definitions

Acronym	Description
AppSvc	Application Services
API	Application Programming Interface
DSP	Digital Signal Processor
GPRS	General Packet Radio Service
GRE	Generic Routing Encapsulation
GTP-U	GPRS Tunneling Protocol- User Data
LLD	Low Level Driver
LUT	Look –Up Table
PA	Packet Accelerator
PASS	Packet Accelerator Sub-System
PDSP	Packet Data Structure Processor
PTP	Precision Timing Protocol
RTP	Real-time Transport Protocol
SASS	Security Accelerator Sub-System
SCTP	Stream Control Transmission Protocol
SRIO	Serial RapidIO
TI	Texas Instruments

Table 2. Definitions

4 Overview

4.1 Driver Interaction with Application and PASS

The pass low level driver is used to configure the Packet Accelerator Sub System. The driver does not contain a transport layer and is always non-blocking. This means that the driver translates the upper level packet routing requirements into configuration information that is used by the PASS firmware. The connection between the driver, the application, and the PASS subsystem is illustrated in Figure 1.

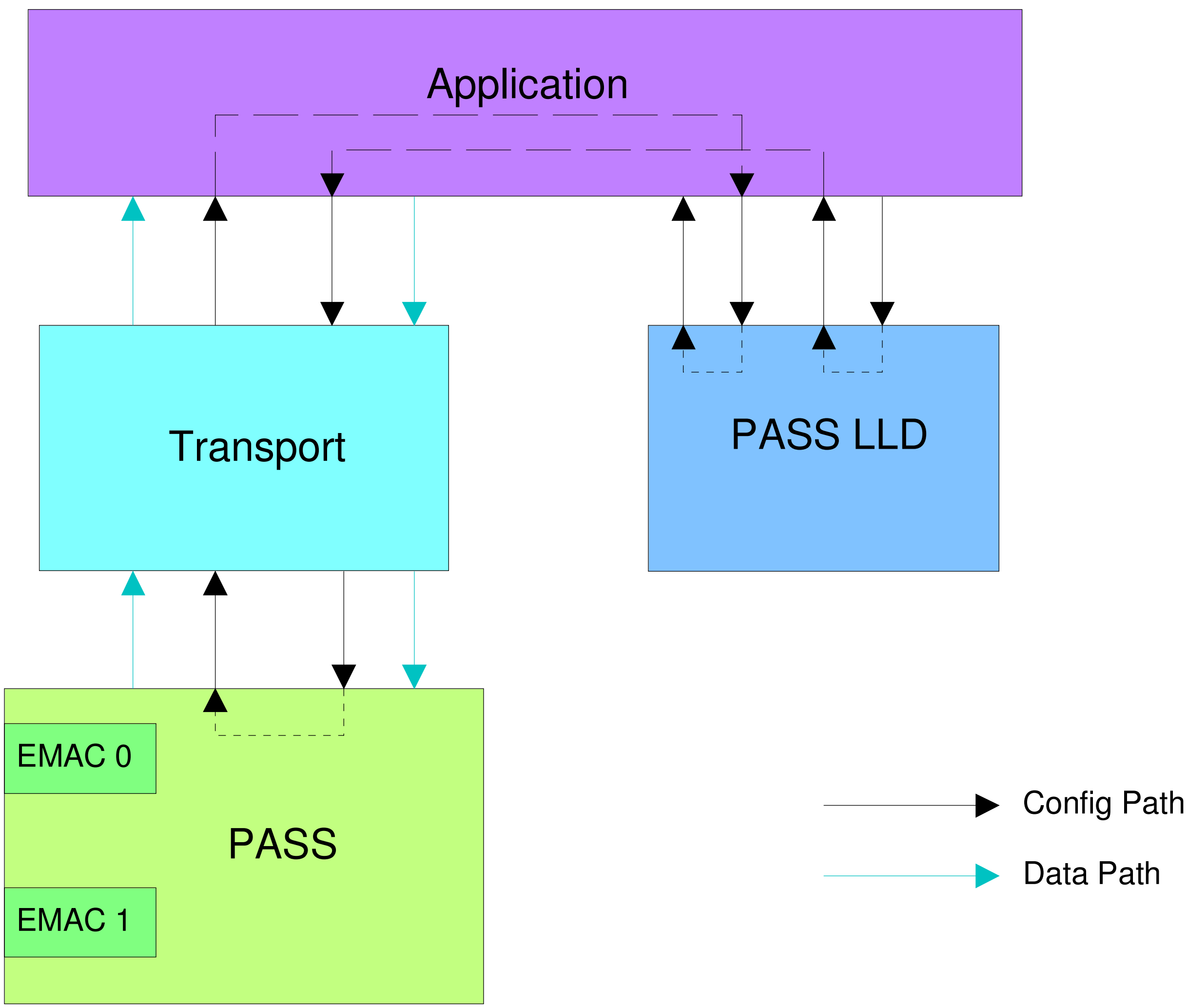


Figure 1

4.2 PASS Resources Overview

The PASS consists of the following resources to perform the input packet classification, checksum/CRC verification, data manipulation and etc.

- Six PDSPs for packet and command processing
- Three 64-entry LUT1 for Layer 2/3 or custom lookup
- One 8192-entry LUT2 for Layer 4/5 or custom lookup
- Six programmable CRC engines for CRC computation and verification
- Six 16-bit general purpose timer

Table 3 shows the inter-connections among the PASS resources.

PDSP	LUT1	LUT2	CRC Engine	Timer
PDSP0	LUT1_0		CRC Engine 0	Timer 0-5
PDSP1	LUT1_1		CRC Engine 1	Timer 0-5
PDSP2	LUT1_2		CRC Engine 2	Timer 0-5
PDSP3		LUT2	CRC Engine 3	Timer 0-5
PDSP4			CRC Engine 4	Timer 0-5
PDSP5			CRC Engine 5	Timer 0-5

Table 3 PASS sub-module inter-connection

4.3 Implicit PASS Configuration by the LLD

The LLD attempts to abstract the operation of the PASS from the application. The LLD uses the following rules when configuring the PASS:

- All received packets from Ethernet and SRIO are routed to PDSP0
- PDSP0 does L0-L2 (MAC/SRIO) lookup using LUT1-0. If the packet is IP it is forwarded to PDSP1
- PDSP1 does the outer IP or Custom LUT1 lookup using LUT1-1
- PDSP2 does any subsequent IP or Custom LUT1 lookup using LUT1-2
- PDSP3 does all TCP/UDP and Custom LUT2 lookup using LUT2
- PDSP4 is used for post-lookup processes such as checksum/CRC result verification.
- PDSP4/5 can be used for pre-transmission operation such as transmit checksum generation.

The LUT1 consists of 64 entries which are indexed as 0 to 63 from top to bottom. The LUT1 will perform top to bottom search and return the index of the first entry which matches the search criteria per search request. It is not required to maintain the used entries in a continuous region. The empty entry will be skipped during the search operation.

The LLD APIs are provided to allow the LUT1 entry to be allocated by the PASS firmware or specified by the application. If the LLD entry is allocated by the firmware, it is done from the bottom to the top. Because of this the most general configuration must be done first before any overlapping more specific configuration. For example, to route packets with MAC address X and no vlan tags, as well as packets with MAC address X and vlan tag Y, it is necessary to first do the MAC only configuration first, as this is the most general. If done in the other order then all packets with MAC address X would go to the destination specified for the MAC only address, even if they contained vlan tag Y.

It is up to the application to maintain all entries in each LUT1 if it would like to call the LUT1 configuration APIs with desired entry index. The PA LLD and firmware will simply replace the specified entry with the new configuration without any error checking. Please note that these two methods should not be mixed.

4.4 LLD Functions

The LLD provides the system interface, configuration and control interface and utility functions with a set of APIs as described at section 5.

4.4.1 System Interface

The system interface maintains the system level resources and needs to be coordinated among multiple CorePacs. All the data access provided by the system interface should invoke the PASS CSL layer. The system interface performs the following tasks:

- Reset, download and update the PASS PDSP images.
- Configure the PASS timer

4.4.2 Configuration and Control interface

The LLD provides an abstraction layer between the application and PASS to configure and control the PASS. It is implemented as a command and response mechanism as described below:

1. The PASS configuration and control API produces a command packet based on the configuration or control parameters.
2. The command packet is delivered to the PASS through the transport layer by the application.
3. The PASS generates a command response packet which will be delivered to the application through the transport layer.
4. The application invoke the “forward PASS result” API as described at section 5.2.21
5. The application may repeat step 2 if retransmission is required.

The configuration and control interface performs the following tasks:

- Configure LUT1 entry
- Configure LUT2 entry
- Configure multi-route entry
- Configure exception routing entry

- Configure command set entry
- Configure CRC engines
- Configure User-defined statistics
- Configure system (global) parameters
- Inquire system statistics
- Inquire User-defined statistics
- Inquire system timestamp

4.4.3 Utility Functions

The LLD also provide a set of utility functions and macros to format the tx commands which need to be delivered to the PASS as the protocol specific information with the tx packets through the transport layer. The desired commands will be executed within PASS prior to the final route.

4.4.4 Sample Codes

There are some features which are not fully-supported by this generation of PASS. The PA LLD and firmware are enhanced to provide hardware-assistance to facilitate those operations such as IP reassembly. Sample codes will be provided as examples to show the module user how to use those features. The interfaces of the sample code and its functionalities will be defined at section 7.

5 LLD External Interface Definitions (APIs)

5.1 Constant and Type Definition

5.1.1 Pass Size Info

paSizeInfo_t

This structure contains the information required to determine run time memory requirements. It is used by Pa_getBufferReq as described at section 5.2.1.

Name	Description
nMaxL2	The maximum number of level 2 (MAC, SRIO) handles that the driver can support at one time
nMaxL3	The maximum number of level 3 (IP) protocols supported. Note that other L3 protocols (ARP, RARP, etc) are not handled in PA and are routed out of PA in the L2 lookup.
nUsrStats	The maximum number of user-defined statistics. Note: It is required because PA LLD will maintain the link table for the user-defined statistics as defined at section 5.1.21

5.1.2 Run time initialization

paConfig_t

This structure contains information required to initialize an instance of the pass module. It is used by Pa_create as described at section 5.2.2.

Name	Description
initTable	If TRUE the L2 and L3 tables are initialized. For multi-core modes which share a single set of tables only one DSP should initialize the tables.
initDefaultRoute	If TRUE then the switch default route is set to PASS PDSP0
baseAddr	Specify the PASS base address
sizeCfg	Pointer to the size configuration information

5.1.3 Core (Thread) configuration

paStartCfg_t

This structure contains information required to initialize the core/thread specific local object of the pass module. It is used by Pa_startCfg as described at section 5.2.3.

Name	Description
rmHandle	a handle to the Resource Manager instance

5.1.4 Pass System Control Info

This section defines the system control information including the system-level configuration parameters used by Pa_control as described at section 5.2.5.

The system level configuration parameters are divided into subgroups which can be set independently.

paProtocolLimit_t

This structure defines the protocol-specific restrictions. For example, it is necessary to limit the number of protocol layers such as GRE of the input packets to prevent the irregular packets take too much processing time. The PASS will detect the packets which violate the protocol-specific restrictions and either discard or forward the packets to host queues which can be specified through API Pa_configExceptionRoute.

Name	Description	Default	Max

vlanMax	Maximum number of VLANs supported	2	3
ipMax	Maximum number of IP layers supported	2	7
greMax	Maximum number of GRE layers supported	2	7

paIpReassmConfigl_t

This structure contains information to configure the IP reassembly-assistance operation. Two separate structures are used for the outer IP and inner IP respectively. The IP reassembly assistance feature is disabled by default until this information is provided.
The maximum number of traffic flows is limited due to processing time and internal memory restriction.

Name	Description	Default	Max
numTrafficFlow	Maximum number of IP reassembly traffic flows supported	0	32
destQueue	Destination host queue where PASS will deliver the packets which require reassembly assistance	NA	NA
destFlowId	Specify the CPPI flow which instructs how the link-buffer queues are used for forwarding packets.	NA	NA

paCmdSetConfig_t

This structure defines command set configuration parameters. The module user can specify the number of command sets supported. The PASS supports either 64 of 64-byte or 32 of 128-byte command sets. The number of command sets should be configured at system startup.

Name	Description	Default	Max
numCmdSets	Number of command sets supported (32, 64) Note: If the number of command sets is set to 64, then each command entry will be limited to 64 bytes.	64	64

paUsrDefinedStatsConfig_t

This structure defines the configuration parameters for multi-level hierarchical user-defined statistics operation. The user-defined statistics feature is disabled until this configuration is invoked through API Pa_control.

Name	Description	Default	Max
numCounters	Number of user-defined counters	0	512
num64bCounters	Number of 64-bit user-defined counters	0	256

paQueueDivertConfigl_t

The PASS supports optional queue diversion operation per LUT2 entry replacement. This structure contains information for the atomic queue diversion operation. The queue diversion feature is disabled until this configuration is invoked through API Pa_control.

Name	Description	Default	Max
destQueue	The destination queue where PASS will deliver the LUT2 response packet which contains the queue diversion information	NA	NA
destFlowId	Specify the CPPI flow which instructs how the link-buffer queues are used for forwarding the LUT2 response packet	NA	NA

paPacketVerifyConfig_t

The PASS always performs basic protocol header verification to ensure that it can continue parsing the current and next protocol header. The PASS will perform enhanced error check of protocol headers specified by this configuration. The advanced packet verification feature is disabled until this configuration is invoked through API Pa_control.

Name	Description	Default	Max
protoBitMap	Packet verification protocol bit B0: PPPoE header check B1: IPv4 header checkl	NA	NA

paSysConfig_t

This structure contains pointers to the system-level configuration structures defined above. The null pointer indicates the configuration of the corresponding sub-group is not required.

Name	Description
protoLimit	Pointer to the protocol limit configuration structure
outIpReassmConfig	Pointer to the outer IP Reassembly configuration structure
inIpReassmConfig	Pointer to the inner IP Reassembly configuration structure
cmdSetConfig	Pointer to the command set configuration structure
usrDefinedStatsConfig	Pointer to the user-defined statistics configuration structure
queueDivertConfig	Pointer to the queue-diversion configuration structure
pktVerifyConfig	Pointer to the packet verification configuration structure

paCtrlCode_t

This enumeration lists the control codes supported by PASS

Pa_CONTROL_SYS_CONFIG
Pa_CONTROL_802_1ag_CONFIG
Pa_CONTROL_IPSEC_NAT_T_CONFIG

pa802p1agDetConfig_t

The 802.1ag packet can be recognized with ether type equal to 0x8902 normally. However, the PASS can be configured to further qualify the IEEE 802.1ag packet per one of the following criteria:

- 802.1ag standard: Destion MAC address = 01-80-c2-00-00-3x, Ether type = 0x8902
- 802.1ag draft: Destion MAC address = 01-80-c2-xx-xx-xx, Ether type = 0x8902

The 802.1ag packet detector is disabled until this configuration is invoked through API Pa_control.

Name	Description	Default	Max
ctrlBitMap	802.1ag detector control bitmap B0: Disable/Enable B1: Draft/Standard	Disable	NA

paIpsecNatTConfig_t

This data structure is used to configure the IPSEC NAT-T packet detector which is disabled until this configuration is invoked through API @ref Pa_control.

Name	Description	Default	Max
ctrlBitMap	IPSEC Nat-T detector control bitmap B0: Disable/Enable	Disable	NA
udpPort	Specify the UDP port number which uniquely identifies the IPSEC NAT-T packets		

paCtrlInfo_t

This structure is used to define the global control information which is invoked by the API Pa_control as described at section 5.2.5.

Name	Description
code	Specify the PA control code as defined above
params	Specify the control-specific configuration parameters such as the system-level configuration parameters as specified by paSysConfig_t pa802p1agDetConfig_t paIpsecNatTConfig_t

5.1.5 Pass Handle

paHandleL2L3_t

This structure defines a pass handle to LUT1 entry. For the application point of view it is typecast to be a void *. A predefined handle pointer called pa_LLD_HANDLE_IP_INNER is used to for adding entries to the IP LUT1 when adding nested inner IP addresses to the LUT without wanting to associate the address with an outer IP address. Without this value IP addresses added without a linking MAC or IP handle will be placed in the outer IP lookup table.

5.1.6 Exception Routing Types

This enumeration lists the exception routing types including error conditions and lookup failures which can occur in the PASS. It is used by Pa_configExceptionRoute described at section 5.2.16

pa_EROUTE_L2L3_FAIL
pa_EROUTE_VLAN_MAX_DEPTH
pa_EROUTE_IP_MAX_DEPTH
pa_EROUTE_MPLS_MAX_DEPTH
pa_EROUTE_GRE_MAX_DEPTH
pa_EROUTE_PARSE_FAIL
pa_EROUTE_L4_FAIL
pa_EROUTE_IP_FRAG,
pa_EROUTE_IPV6_OPT_FAIL
pa_EROUTE_UDP_LITE_FAIL
pa_EROUTE_ROUTE_OPTION
pa_EROUTE_SYSTEM_FAIL
pa_EROUTE_MAC_BROADCAST
pa_EROUTE_MAC_MULTICAST
pa_EROUTE_IP_BROADCAST
pa_EROUTE_IP_MULTICAST
pa_EROUTE_GTPU_MESSAGE_TYPE_1
pa_EROUTE_GTPU_MESSAGE_TYPE_2
pa_EROUTE_GTPU_MESSAGE_TYPE_26
pa_EROUTE_GTPU_MESSAGE_TYPE_31
pa_EROUTE_GTPU_MESSAGE_TYPE_254
pa_EROUTE_GTPU_FAIL
pa_EROUTE_PPPoE_FAIL
pa_EROUTE_802_1ag
pa_EROUTE_IP_FAIL
pa_EROUTE_NAT_T_KEEPALIVE
pa_EROUTE_NAT_T_CTRL
pa_EROUTE_NAT_T_DATA
pa_EROUTE_NAT_T_FAIL

5.1.7 Packet Destinations

This specifies destinations for packets when a LUT matching occurs. It is used at the data structure paRouteInfo_t described at section 5.1.9.

pa_DEST_HOST
pa_DEST_EMAC
pa_DEST_SASS
pa_DEST_SRIO
pa_DEST_DISCARD
pa_DEST_CONTINUE_PARSE_LUT1
pa_DEST_CONTINUE_PARSE_LUT2

5.1.8 Custom Type Destinations

This specifies types of custom classification supported by PASS. It is used at the data structure paRouteInfo_t described at section 5.1.9.
pa_CUSTOM_TYPE_NONE
pa_CUSTOM_TYPE_LUT1
pa_CUSTOM_TYPE_LUT2

5.1.9 Routing information

paRouteInfo_t
This structure defines how receive packets are routed in the case of either match or failed match. It contains the following elements

Name	Description
dest	Defines the forward location for the packet (pa_DEST_xxx)
flowId	Flow ID is used to specify which free queues are used for receiving packets. This is only valid on packets from PA to host , SA or SRIO
queue	If the destination is the host, SA or SRIO then this specifies the destination queue
mRouteIndexe	Specifies the index of the multi-route configuration to use, pa_NO_MULTI_ROUTE if not used
swInfo0	A 32 bit value returned in the descriptor as swInfo0 if the destination is host or SA. A 32 bit value returned in the descriptor as psInfo0 if the destination is SRIO.
swInfo1	A 32 bit value returned in the descriptor as swInfo1 if the destination is SA. A 32 bit value returned in the descriptor as psInfo1 if the destination is SRIO.

customType	Specifies the custom type of the next lookup. pa_CUSTOM_TYPE_NONE if standard protocols. Vaild only if the dest is pa_DEST_CONTINUE_PARSE
customIndex	Specifies the custom classification entry index. Valid if customType is not pa_CUSTOM_TYPE_NONE
pktType_emacCtrl	For destination SRIO, specify the 5-bit packet type toward SRIO For destination HOST, EMAC, specify the EMAC control bits
pCmd	Pointer to the command info as defined at section 5.1.16. A simple command can be executed upon a LUT2 matching[1], NULL if not used. Note: Only a limited set of commands such as "Pactch 2 bytes" can be used here. Specify a command set if command is too big or a sequence of commands is required.

5.1.10 Multi-route information

paMultiRouteEntry_t

This structure defines the physical routing of packets for each multi-route entry. It is only a subset of the Routing information defined at section 5.1.9 because those common parameters such as swInfo0, swInfo1 must be already present in the packet descriptor. This structure is used at the API Pa_configureMultiRoute defined at section 5.2.17.

Name	Description
ctrlBitfield	The multi-route control information as defined below: • pa_NEXT_ROUTE_DESCRIPTOR_ONLY: Forward the packet descriptor only • pa_MULTI_ROUTE_REPLACE_SWINFO: Replace the swInfo0 with the value provided here
flowId	Specifies which free queues are used for receiving packets.
queue	Specifies the destination queue
swInfo0	Placed in SwInfo0 for packets to host

5.1.11 Eth Info

paEthInfo_t

[1] Post –classification command is only supported for LUT2 match. The command field will be ignored after a LUT1 match.

This structure is used to pass information about the Ethernet header used for packet routing. Any value this is input as 0 is considered a "don't care". It is used at the API Pa_addMac defined at section 5.2.6.

Name	Description
srcMac	The source MAC address
dstMac	The destination MAC address
Vlan	The VLAN ID
Ethertype	The ethertype
MPLS tag	The MPLS tag value. Only the outer tag is examined.
inport	The input EMAC port number

5.1.12 SRIO Info

paSrioInfo_t

This structure is used to pass information about the SRIO type 9 and type 11 L0-L2 message headers used for packet routing. It is used at the API Pa_addSrio defined at section 5.2.7.

Name	Description
validBitMap	The bitmap indicating which parameters are valid
srcId	The source ID
destId	The destination ID
tt	The transport type
cc	The completion code
pri	The message priority
msgType	Type 9 or Type 11
cos	The type 9 class of service
streamId	The type 9 stream ID
letter	The type 11 letter
mbox	The type 11 mailbox

5.1.13 IP Info

paIpInfo_t

This structure is used to pass information about the IP and related L3 headers used for L3 packet routing. The non-IP parameter is one of the followings:

- SPI for IPSEC ESP or AH packet
- GRE protocol for GRE packet
- Destination port for SCTP packet

With the exception of the Tos field, any value that is input as 0 is considered a "don't care". This structure is used at the API Pa_addIp described at section 5.2.8.

Name	Description
Src	The source IP address
Dst	The destination IP address
Spi	ESP or AH header security parameters index
Flow	Ipv6 flow label in the 20 lsbs
Ip type	Ipv4 or Ipv6
Gre protocol	The GRE protocol field
Proto	The IPv4 protocol field or IPv6 next header field
Tos	Type of service (IPv4) or traffic class (IPv6)
tosCare	TRUE if the TOS value should be used for packet matching
sctpPort	The SCTP destination port

5.1.14 PASS Command Reply Routing

paCmdReply_t

This structure is used to specify command reply (from PASS) routing information. It is passed to the LLD APIs when the result is a command packet that must be forwarded to the PASS through the transport layer.

Name	Description
Dest	Command reply destination. Must be host or discard
Reply ID	A 32 bit value placed in swinfo0 to identify the packet as a command reply
Queue	The destination queue where the PASS will route the command reply (for destination host only)
Flow ID	Flow ID is used to specify which free queues are used for receiving command reply. This is only valid if command reply destination is host.

5.1.15 PASS System State

This enumeration defines the operating states of the PA sub-system. They are used both to set the state of PASS (pa_STATE_RESET and pa_STATE_ENABLE) as well as show the current state of the system (all values). It is used for API Pa_resetControl described at section 5.2.30.

PA_STATE_RESET
PA_STATE_ENABLE
pa_STATE_QUERY
PA_STATE_INCONSISTENT
pa_STATE_INVALID_REQUEST
pa_STATE_ENABLE_FAILED

5.1.16 PASS Command (Action)

A single command or a set of commands can be executed to support fully-offloaded data path in both the transmit (to-network) and receive (from-network) directions.
In the to-network direction, the stack of commands formatted by the LLD should be stored as the protocol-specific information in the packet descriptor with the packet. The commands will be executed in order at PASS and the associated SASS. The executed commands will be removed by PASS and SASS so that the output packet will not contain any command.
In the from-network direction, the stack of commands formatted by the LLD should be stored at the PASS as a command set which can be referred to by the command set index. Therefore a single command including a command set can be executed per the enhanced routing information after a LUT1/LUT2 match.
The following sub-sections describe each command and its associated data structure in details.

5.1.16.1 Command (Action) Definitions

This specifies the transmission and post-classification commands (actions) supported by PASS. These definitions are used at data structure paCmdInfo_t desribed at section 5.1.16.3.

Command	Description
pa_CMD_NONE	The pa_CMD_NONE is used to indicate that there is no command. It can be used to terminate a sequence of command.
pa_CMD_NEXT_ROUTE	This command tells the PASS where to forward the packet once all preceding commands have been executed.
pa_CMD_CRC_OP	This command informs the PASS to calculate or validate CRC.
pa_CMD_COPY_DATA_TO_PSINFO	The copy command copies up to 8 bytes from a packet to the PS Info area.

pa_CMD_PATCH_DATA	The patch command allows overwrite of up to 32 bytes into a packet.
pa_CMD_MULTI_ROUTE	The multi-route command tells the PASS to route the packets to multiple destinations.
pa_CMD_REPORT_TX_TIMESTAMP	This command instructs the PASS to report the PA timestamp when the packet is forwarding out of PASS.
pa_CMD_REMOVE_HEADER	This command instructs the PASS to remove the parsed header before forwarding the packet.
pa_CMD_REMOVE_TAIL	This command instructs the PASS to remove the parsed tail before forwarding the packet.
pa_CMD_CMDSET	This command specifies the command set to be executed
pa_CMD_SA_PAYLOAD	This command provides the payload information required by SASS.
pa_CMD_IP_FRAGMENT	This command instructs the PASS to fragment IPv4 packet
pa_CMD_USER_STATS	This command instructs the PASS to update the specified user-defined counter and the counters which are linked to this counter
pa_CMD_CMDSET_AND_USR_STATS	Combination of the CMDSET and USR_STATS commands
pa_CMD_PATCH_MSG_LEN	This command instructs the PASS to update the message length field within some L2 protocol header such as 802.3 and PPPoE after the potential IP fragmentation operation

5.1.16.2 PASS Command (Action) Specific Configuration

The following structures define the command specific configuration parameters.

The next route command can be used in both to-network and from-network directions. In the to-network direction, it may be used multiple times to route traffic between PASS and SASS before the packet is finally forwarded to the network. For example, the following steps show the SRTP over IPSEC AH to-network traffic:

1. Packet is delivered to SASS for SRTP operation
2. Packet is delivered to PASS for UDP checksum operation
3. Packet is delivered to SASS for IPSEC AH operation
4. Packet is delivered to PASS for AH authentication tag insertion
5. Packet is delivered to the network.

The next route commands are required for step 3 and 5. The complete routing information should be provided in the to-network direction.

In the from-network direction, the optional next route command provides the multi-route information only if multi-route is required. In this case, only the parameter “ctrlBitfield” and “multiRouteIndex” is valid. The packet will be forwarded to its final destination provided at the routing information after all the actions specified by the command set are executed.

paCmdNextRoute_t:

Name	Description
ctrlBitfield	The routing control information as defined below • pa_NEXT_ROUTE_PARAM_PRESENT: Routing information such as flowId, queue are present in command • pa_NEXT_ROUTE_PROC_NEXT_CMD: Process the next command prior to forward the packet to its final destination • pa_NEXT_ROUTE_PROC_MULTI_ROUTE: Forward the packet and then perform multi-route as specified by multiRouteIndex
dest	Defines the forward location for the packet (pa_DEST_xxx)
flowId	Flow ID is used to specify which free queues are used for receive packets. This is only valid on packets from PA to host or SA
queue	If the destination is the host or SA then this specifies the destination queue
pktType_emacCtrl	If the destination is SRIO, this parameter specifies the packet type If the destination is ETH or HOST, this parameter specifies the EMAC control bits
swInfo0	A 32 bit value returned in the descriptor as swInfo0 if the destination is host or SA. A 32 bit value returned in the descriptor as psInfo0 if the destination is SRIO.
swInfo1	A 32 bit value returned in the descriptor as swInfo1 if the destination is SA. A 32 bit value returned in the descriptor as psInfo1 if the destination is SRIO.
multiRouteIndex	Multi-route index

The CRC operation command is used to instruct the PASS to perform CRC operation in both to-network and from-network directions. In the to-network direction, the payload offset, CRC length and CRC offset should be available in the command.

In the from-network direction, the payload length is either a constant or available in the custom header. Both the payload length and the byte location where CRC calculation begins may vary in some protocol frame such as WCDMA FP HS-DSCH Data Frame type 2 and type 3. The following table specifies the supported frame types:

Name	Description
FP_HS_DSCH_TYPE2	WCDMA FP HS-DSCH Data Frame Type
FP_HS_DSCH_TYPE3	WCDMA FP HS-DSCH Data Frame Type 2

The PASS should verify and restrict the offset and length value to the packet boundary if those parameters cause the CRC calculation cross over the packet boundary. In this case, the CRC calculation will be incorrect.

paCmdCrcOp_t:

Name	Description

ctrlBitfield	The CRC operation control information as defined below
	• pa_CRC_OP_CRC_VALIDATE:
	  o Set: CRC validation
	  o Clear: CRC computation
	• pa_CRC_OP_PAYLOAD_LENGTH_IN_HEADER:
	  o Set: CRC length field in the header
	  o Clear: CRC length specified in command
	• pa_CRC_OP_PAYLOAD_LENGTH_OFFSET_NEGATIVE:
	  o Set: Payload length field resides prior to the custom header
	  o Clear: Payload length field resides within the custom header
	• pa_CRC_OP_CRC_FRAME_TYPE:
	  o Set: Frame Type is specified
	  o Clear: Frame Type is not specified, use offset parameter
	• pa_CRC_OP_CRC_FOLLOW_PAYLOAD
	  o Set: CRC field following payload
	  o Clear: CRC offset specified in command
	Add a control flags to (For receive) Parsing offset

startOffset	Byte location, from SOP/Protocol Header, where the CRC computation begins. In to-network direction: offset from SOP In from-network direction: offset from the current parsed header
len	Number of bytes covered by the CRC computation, valid if pa_CRC_OP_PAYLOAD_LENGTH_IN_HEADER is clear
lenOffset	Payload length field offset in the header, valid if pa_CRC_OP_PAYLOAD_LENGTH_IN_HEADER is set
lenAdjust	Payload length adjustment
crcOffset	Offset from SOP/Protocol Header to the CRC field, valid if pa_CRC_OP_CRC_FOLLOW_PAYLOAD is clear In to-network direction: offset from SOP In from-network direction: offset from the current parsed header
frameType	Frame type defined above, vaild if pa_CRC_OP_CRC_FRAME_TYPE is set

The copy command is used to instruct the PASS to copy up to 8 byte from packet to the PS info section in the packet descriptor in the from-network direction. If the desired copy area crosses over the packet boundary, then garbage data will be copied.
paCmdCopy_t:

Name	Description
ctrlBitfield	The Copy operation control information as defined below • pa_COPY_OP_FROM_END: o Set: Copy data from the end of the payload o Clear: Copy data from the beginning of the payload
srcOffset	Offset from the start of the current protocol header for the data copy to begin
destOffset	Offset from the top of the PSInfo for the data to be copied to
numBytes	Number of bytes to be copied (1 to 8) without regard to the packet length.

The patch command is used to patch existing data or insert data in the packet in both to-network and from-network directions.
In the to-network direction, it is used to patch the authentication tag provided by SASS into the AH header within the packet. In this case, the patch data is not present at the command when it is formatted and it is appended by the SASS.
In the from-network direction, it can be used to insert up to 32 bytes to the offset location.

This command can be used to patch the entire MAC header for MAC router functionality. This command may be further enhanced and combined with other commands to support IP forwarding operation in the future.
If the number of bytes is less than two, then this command can be used as part of the routing information.
If the desired patching area crosses over the packet boundary, only the data within the packet boundary will be overwritten.

paPatchInfo_t:

Name	Description

ctrlBitfield	The Patch operation control information as defined below
	• pa_PATCH_OP_INSERT:
	  o Set: Insert data into the packet
	  o Clear: the patch data replaces existing packet data
	• pa_PATCH_OP_MAC_HDR
	  o Set: Replace MAC header
	  o Clear: Normal Patch/Insert operation
	• pa_PATCH_OP_DELETE
	  o Set: Delete data in the packet
	  o Clear: Normal Patch/Insert operation

nPatchBytes	The number of bytes to be patched (1-32)
totalPatchSize	The number of patch bytes in the patch command (0-32)
offset	Offset from the SOP or protocol header for the patch to begin. In to-network direction: offset from SOP In from-network direction: offset from the current parsed header
patchDataBuf	Pointer to the patch data, NULL indicates that patch data is not available in application. It will be provided by the PASS or SASS internally.

The Tx checksum command is used to instruct the PASS to perform checksum operation in to-network direction. It is not used in the from-network direction.
The PASS should verify and restrict the offset and length value to the packet boundary if those parameters cause the checksum calculation cross over the packet boundary. In this case, the checksum calculation will be incorrect.

paTxChksum_t:

Name	Description

Start Offset	Where to begin the checksum, from the start of the packet data (in bytes)
Length	The length of the checksum, in 16 bit words
Result Offset	Where to put the checksum in the packet, when done (in bytes)
Initial value	The initial value of the checksum
Negative 0	If TRUE then a checksum value of 0 is written as 0xffff

The report tx timestamp command is used to instruct the PASS to report the PA timestamp when the packet is transmitting out of PASS in a return (null) packet to the specified host queue. The transmit timestamp may be used for the Precision Timing Protocol (PTP).

paCmdTxTimestamp_t:

Name	Description
flowId	Specify the CPPI flow which instructs how free queues are used for sending return packets.
destQueue	Host queue for the return packet
swInfo0	A 32 bit value returned in the descriptor as swInfo0 which can be used as event identifier

The IP fragment command is used to instruct the PASS to perform IP fragmentation operation. This operation can be applied to both inner IP prior to IPSEC encapsulation and outer IP after IPSEC encapsulation. Packets are sent to PASS PDSP5 with both IP fragment command and next route command which specifies the final destination, the entire packet or its fragments will be delivered to the final destination based on the packet size and the MTU size specified at the IP fragment command.
This command is always the last command and it must follow the next route command so that the destination information will be set prior to the fragmentation operation.

For the inner IP fragmentation, follow the following procedure:

1. Host sends packets with the IP fragment command and the destination queue set to a host queue to PASS PDSP5 for IP fragmentation operation.
2. All fragments will be delivered to the specified host queue.
3. Host adds the outer MAC/IP header, invokes the SA LLD sendData function and then sends the fragments to the SA queue.
4. Each fragment will be encrypted, authenticated and forwarded to the final destination.

For the outer IP fragmentation, the overall operation is stated below:

1. Packet is delivered to SASS for IPSEC operation
2. Packet is delivered to PASS for IP Fragmentation operation
3. The entire packet or its fragments are delivered to the network.

The next route command is required for step 2.

Packets are sent to PASS PDSP5 with both IP fragment command and next route command which specifies the final destination, the entire packet or its fragments will be delivered to the final destination based on the packet size and the MTU size specified at the IP fragment command.

paCmdIpFrag_t:

Name	Description
IP Offset	Offset to the IP header
MTU size	Size of the maximum transmission unit (>=68)[2]

The multi-route command instructs the PASS to route the packets to multiple destinations. It is typical the last command within a command set. It is used in the from-network direction only.

paCmdMultiRoute_t:

Name	Description
index	Multi-route set index

This "message length patching" command instructs the PASS to update the message length field within some L2 protocol header such as 802.3 and PPPoE after the potential IP fragmentation operation. It is used in conjunction with the Ipv4 fragmentation command and is used in the to-network direction only.

paPatchMsgLenInfo_t:

Name	Description
msgLenSize	Size of message length field in bytes (only 2-byte message length is supported)
offset	Offset from the start of the packet to the message length field
msgLen	Message length excluding the IP header and payload length

The command set command instructs the PASS to execute a stack of commands after a LUT1 or LUT2 match occurs. It is used in the from-network direction only.

paCmdSet_t:

Name	Description
index	Command set index

[2] It is the minimum MTU size specified by RFC 791. However, the PASS is designed to handle the MTU size as low as 28 bytes.

The user stats command instructs the PASS to update the specified user-defined counter and all the counters in the linking chain.

paCmdUsrStats_t:

Name	Description
index	User-defined statistics index

This cmdSetUsrstats command provides the module user a mechanism to specify different user-defined counter with the same command set for different LUT entries and vice versa. This command instructs the PASS to update the specified user-defined counter and all the counters which are linked to this counter and then execute the specified command set.

paCmdSetUsrStats_t:

Name	Description
setIndex	Commad Set Index
statsIndex	User-defined statistics index

The payload info command provides the payload information required by the SASS in the to-network direction. It is not used in the from-network direction.

paPayloadInfo_t:

Name	Description
offset	Offset to the desired protocol header. IPSEC/ESP: ESP header IPSEC/AH: IP header SRTP: RTP header Air Ciphering: PDU
payloadLen	The payload length

5.1.16.3 PASS Command (Action) Configuration Information

paCmdInfo_t

This structure is used to define a PASS command which is invoked by the paRouteInfo_t as described at section 5.1.9 and the API Pa_setCmdSet as described at section 5.2.18.

Name	Description

cmd	Specify the PA command code as defined at section 5.1.16.1
params	Specify the command-specific configuration parameters as defined at section 5.1.16.2

5.1.17 PASS CRC Engine Configuration

paCrcConfig_t

This structure is used to configure the CRC engines within the PA sub-system. The LLD uses this structure to create a CRC configuration command packet to be forwarded to the PASS as described at section 5.2.19.

Name	Description

ctrlBitfield	The CRC configuration control information as defined below:
	• pa_CRC_CONFIG_RIGHT_SHIFT
	  ○ Set: Right shift CRC (b0 to b7)
	  ○ Clear: Left shift CRC (b7 to b0)
	• pa_CRC_CONFIG_INVERSE_RESULT: a 'NOT' operation is applied to the final CRC result

Size	Specify the CRC size (8, 16, 24 or 32 bits)
Polynomial	Specify the CRC polynomial in the format of 0xabcdefgh. For example, $x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^{9} + x^{8} + x^{6} + 1$ ➔ 0x1EDC6F41 $x^{16} + x^{15} + x^{2} + 1$ ➔ 0x80050000 $x^{8} + x^{7} + x^{6} + x^{4} + x^{2} + 1$ ➔ 0xD5000000
Initial value	Specify the initial value of the CRC computation

5.1.18 PASS Timestamp Configuration

paTimestampConfig_t

This structure is used to configure the timer which is used to generate timestamp in the PA sub-system. The LLD uses this structure to enable/disable the timer and set its scale factor as described at section 5.2.20.

Name	Description
enable	Enable/Disable(1/0) the timestamp generation
Scale Factor	Timestamp unit scale factor as defined below: pa_TIMESTAMP_SCALER_FACTOR_1 = -1,

	pa_TIMESTAMP_SCALER_FACTOR_2 = 0, pa_TIMESTAMP_SCALER_FACTOR_4, pa_TIMESTAMP_SCALER_FACTOR_8, pa_TIMESTAMP_SCALER_FACTOR_16, pa_TIMESTAMP_SCALER_FACTOR_32, pa_TIMESTAMP_SCALER_FACTOR_64, pa_TIMESTAMP_SCALER_FACTOR_128, pa_TIMESTAMP_SCALER_FACTOR_256, pa_TIMESTAMP_SCALER_FACTOR_512, pa_TIMESTAMP_SCALER_FACTOR_1024, pa_TIMESTAMP_SCALER_FACTOR_2048, pa_TIMESTAMP_SCALER_FACTOR_4096, pa_TIMESTAMP_SCALER_FACTOR_8192

5.1.19 System Timestamp

paTimestamp_t

This structure defines the 48-bit timestamp provided upon request with API Pa_getTimestamp () which is defined at section 5.2.21.

Name	Description
hi	Upper 32 bits of the 48-bit PASS timestamp
lo	Lower 16 bits of the 48-bit PASS timestamp.

5.1.20 PASS Statistics

The PASS maintains a set of system statistics which can be inquired and cleared by the API Pa_requestStats as described at section 5.2.24.

5.1.20.1 PASS Operation Specific Statistics

The following structures define the operation specific statistics.

paClassify1Stats_t:

Name	Description
nPackets	Number of packets entering PDSP0, PDSP1 and PDSP2
nIpv4Packets	Number of IPv4 packets
nIpv6Packets	Number of IPv6 packets

nCustomPackets	Number of custom packets
nNonIpPacket	Number of non-IP packets
nLlcSnapFail	Number of packets with corrupt LLC Snap
nTableMatch	Number of packets with table match found
nNoTableMatch	Number of packets without table match found
nIpFrag	Number of fragmented IP packets
nIpDepthOverflow	Number of packets with too many IP layers
nVlanDepthOverflow	Number of packets with too many VLANs
nGreDepthOverflow	Number of packets with too many GREs
nMplsPackets	Number of MPLS packets
nParseFail	Number of packets which can not be parsed
nInvalidIPv6Opt	Number of IPv6 packets which contains invalid IPv6 options
nInvalidComReplyDest	Number of commands with invalid reply destination
nSilentDiscard	Number of packets dropped
nInvalidControl	Number of packet received with invalid control information
nInvalidState	Number of times the PA detected an illegal state and recovered
nSystemFail	Number of times the PA detected an unrecoverable state and restarted

paClassify2Stats_t:

Name	Description
nParseFail	Number of packets which can not be parsed
nInvldHdr	Number of packets with invalid header
nUdp	Number of UDP packets
nTcp	Number of TCP packets
nCustom	Number of custom packets
nCommandFail	Number of invalid commands
nInvalidComReplyDest	Number of commands with invalid reply destination
nSilentDiscard	Number of packets dropped
nInvalidControl	Number of packet received with invalid control information

paModifyStats_t:

Name	Description
nCommandFail	Number of invalid commands

paCommonStats_t:

Name	Description
nIdAllocationFail	Number of times that Id is not available

5.1.20.2 PASS System Statistics

paSysStats_t

This structure defines the PA system statistics which consists of all the operation-specific statistics.

Name	Description
Classify1	Classify1-specific statistics
Classify2	Classify2-specific statistics
Modify	Modifier-specific statistics
Common	Common statistics

5.1.21 User Defined Statistics

The PA LLD and PASS maintain up to 256 user-defined hierarchical statistics. Each statistic is classified to a level which can be linked to one of the next level statistics. When one counter is incremented, all counters in its linking chain will be incremented, too. The User-defined statistics can be configured through the API Pa_configUsrStats as described at section 5.2.21 and inquired and cleared by the API Pa_requestUsrStats as described at section 5.2.26.

This section describes user-defined statistics related data structures.

5.1.21.1 User Defined Statistics Configuration

The PA LLD supports up to 256 user-defined hierarchical statistics as defined below:

```
#define PA_USR_STATS_MAX_COUNTERS          512
#define PA_USR_STATS_MAX_64B_COUNTERS      256
```

#define PA_USR_STATS_MAX_32B_COUNTERS 512

paUsrStatsCounterEntryConfig_t

This structure defines the operation parameters of each user-defined statistics.

Name	Description
cntIndex	Index of the counter [0, 255]
cntType	Counter type (packet counter or byte counter)
cnkLnk	Index of the next level counter. 0xFFFF: Indicates there is no linking counter

paUsrStatsCounterConfig_t

This structure consists of an array of the counter configuration information

Name	Description
numCnt	Number of counters to be configured
cntInfo[]	Array of counter configuration as specified at paUsrStatsCounterEntryConfig_t

paUsrStatsConfigInfo_t

This structure is used to perform user-defined statistics related configuration. It is used by API Pa_configUsrStats as described at section.5.2.21

Name	Description
counterCfg	Pointer to the user-defined statistics counter configuration. Set to NULL if not provided

5.1.21.2 User Defined Statistics Definition

paUsrDefinedStats_t

Name	Description
Count64[PA_USR_STATS_MAX_64B_COUNTERS]	Array of 64-bit general purpose counters
Count32[PA_USR_STATS_MAX_32B_COUNTERS]	Array of 32-bit general purpose counters

5.2 Call-in APIs

The APIs listed in this section are presented as pseudo code. In cases where functions have large argument lists it is likely that one or more structures will be used instead of the argument list.

5.2.1 GetBufferReq

5.2.1.1 Description

This function is used get the memory requirements of the driver (size, alignment and type).

5.2.1.2 Prototype

```
Result = Pa_getBufferReq(   paSizeInfo_t  *sizeCfg,    /* Initialized  paSizeInfo_t structure */
                            int  size[],               /* array of size requirements */
                            int  align[]);             /* array of alignment requirements */
```

5.2.1.3 Implementation

The memory size and alignment requirements are returned. The application must allocate the size and alignment requirement array with the number of elements defined by pa_N_BUFS. The application must allocate the requested memory and provide it back to the driver in the call to Pa_create.

5.2.1.4 Return Value

pa OK
pa_ERR_CONFIG

5.2.2 Create

5.2.2.1 Description

This function is used to create and initialize an instance of the PA driver

5.2.2.2 Prototype

```
Result = Pa_create (   paConfig_t *config,    /* Pointer to a pasConfig_t structure */
                       void* base[] ,         /* Array contains the buffer addresses */
                       Pa_Handle *pHdl);      /* Return the PA handle */
```

5.2.2.3 Implementation

This function initializes an instance of the PA driver. The application must complete the memory requirements by filling in the base addresses of the memory buffers. The paConfig_t structure is used to control the run time configuration.

5.2.2.4 Return value

pa_OK
pa_ERR_CONFIG Invalid memory table buffer provided

5.2.3 Start

5.2.3.1 Description

This function is used to start a local object of the PA driver instance

5.2.3.2 Prototype

```
Result = Pa_startCfg (Pa_Handle handle,            /* Pointer to the PA handle*/
                      paStartCfg_t *startCfg);    /* Per-core(thread) configuration */
```

5.2.3.3 Implementation

This function initializes a core (thread)-specific local object of the PA driver instance. This function needs to be called from all cores to initialize PA with per core configurations.

5.2.3.4 Return value

pa_OK
pa_ERR_CONFIG Invalid memory table buffer provided

5.2.4 Close

5.2.4.1 Description

This function is used to close the PA instance and return the memory buffer information so that the application can free the allocated buffers.

5.2.4.2 Prototype

```
Result = Pa_close (Pa_Handle handle,    /* Pointer to the PA handle*/
                   void* bases[]);       /* Memory base addresses filled in by the driver */
```

5.2.4.3 Implementation

This function returns the memory buffers used by the driver. This is typically used to tear down the driver and free the memory allocated for the driver.

5.2.4.4 Return value

pa_OK

5.2.5 Control

5.2.5.1 Description

This function is used to perform PASS global control operations including system-level configurations. The system-level configurations are divided into several sub-groups which can be configured independently. The default configuration will is used until this API is invoked.

5.2.5.2 Prototype

```
Result = Pa_control (  Pa_Handle        handle,       /* Pointer to the PA handle*/
                       paCtrlInfo_t*    ;             /* Pointer to the Control information */
                        paCmd_t ,                     /* The generated PASS command */
                        uint16_t *,                   /* Size of command buffer used */
                       paCmdReply_t *,                /* Where to send the PASS reply */
                        int *);                       /* Where to send the command */
```

5.2.5.3 Implementation

This function records the system-level configuration parameters in the LLD instance and converts those parameters into a configuration command packet to be forwarded to the PASS .The PA reply routing is optional because this command is always processed by the PA sub-system.

5.2.5.4 Return value

pa_OK
pa_ERR_CONFIG Invalid configuration parameters provided

5.2.6 Add MAC entry

5.2.6.1 Description

This function adds a MAC entry to the level 2 lookup. Any fields which are 0 are considered “don’t cares” in the match criteria. The MAC entry will be added to the first available location in the LUT1 if the input index is not specified. Otherwise, it will use the LUT1 location specified by the input index. The command response is mandatory since it contains the LUT1 index for this MAC entry. Therefore, the command reply destination (see section 5.1.7) must be host .

5.2.6.2 Prototype

```
result = Pa_addMac ( Pa_Handle,          /* Driver handle */
                     int,                /* Index of the LUT1 entry */
                     paEthInfo_t,        /* Ethernet match info */
                     paRouteInfo_t,      /* Where to send a match */
                     paRouteInfo_t,      /* Where to send a subsequent fail */
                     paHandleL2L3_t *,   /* Returns associated handle */
                     paCmd_t,            /* Where the command is placed */
                     uint16_t *,          /* The size of the command buffer used */
                     paCmdReply_t *,     /* Where to send the PASS reply */
                     int *);             /* Where to send the command */
```

5.2.6.3 Implementation

An internal L2 table entry is made if not already present. If the entry already exists in the table then the existing handle is returned, and the size of the passCmd is set to 0. This function returns error if the cmdReplyDest is not host, the command buffer is too small, or other error conditions occur. Then this function formats a LUT1 configuration command based on the MAC match criteria and the routing information. The configuration command should be forwarded to PDSP0 for entry into LUT1-0.

5.2.6.4 Return values

pa OK
pa_INSUFFICIENT_CMD_BUFFER_SIZE
pa_INVALID_DUP_ENTRY
pa_INVALID_TABLE_MORE_SPECIFIC_ENTRY_PRESENT
pa_INVALID_MPLS_LABEL
pa_ERR_CONFIG
pa_INVALID_CMD_REPLY_DEST
pa_HANDLE_TABLE_FULL

passHandle will contain the handle information for this MAC entry.

5.2.7 Add SRIO entry

5.2.7.1 Description

This function adds SRIO entry to a level 2 lookup. The SRIO entry will be added to the first available location in the LUT1 if the input index is not specified. Otherwise, it will use the LUT1 location specified by the input index. The command response is mandatory since it contains the LUT1 index for this SRIO entry. Therefore, the command reply destination (see section 5.1.7) must be host.

5.2.7.2 Prototype

```
result = Pa_addSrio ( Pa_Handle,          /* Driver handle */
                      int,                /* Index of the LUT1 entry */
                      paSrioInfo_t,       /* SRIO match info */
                      paRouteInfo_t,      /* Where to send a match */
                      paRouteInfo_t,      /* Where to send a subsequent fail */
                      paHandleL2L3_t *,   /* Returns associated handle */
                      paCmd_t,            /* Where the command is placed */
                      uint16_t *,         /* The size of the command buffer used */
                      paCmdReply_t *,     /* Where to send the PASS reply */
                      int *);             /* Where to send the command */
```

5.2.7.3 Implementation

An internal L2 table entry is made if not already present. If the entry already exists in the table then the existing handle is returned, and the size of the passCmd is set to 0. This function returns error if the cmdReplyDest is not host, the command buffer is too small, or other error conditions occur. Then this function formats a LUT1 configuration command based on the SRIO match criteria and the routing information. The configuration command should be forwarded to PDSP0 for entry into LUT1-0.

5.2.7.4 Return values

pa OK
pa_INSUFFICIENT_CMD_BUFFER_SIZE
pa_INVALID_DUP_ENTRY
pa_INVALID_TABLE_MORE_SPECIFIC_ENTRY_PRESENT
pa_ERR_CONFIG
pa_INVALID_CMD_REPLY_DEST
pa_HANDLE_TABLE_FULL

passHandle will contain the handle information for this SRIO entry.

5.2.8 Add IP entry

5.2.8.1 Description

This function adds an IP entry to a level 3 lookup. A Null value for L2L3Handle means that this entry is independent of any previous header. With the exception of TOS, all values that are 0 are considered “don’t care” in the match criteria. The IP entry will be added to the first available location in the LUT1 if the input index is not specified. Otherwise, it will use the LUT1 location specified by the input index. The command response is mandatory since it contains the LUT1 index for this IP entry. Therefore, the command reply destination (see section 5.1.7) must be host.

5.2.8.2 Prototype

```
result = Pa_addIp (  Pa_Handle,           /* Driver handle */
                     int,                 /* Specify which LUT1 (0-2) should be used */
                     int,                 /* Index of the LUT1 entry */
                     paIpInfo_t *,        /* IP match information */
                     paHandleL2L3_t *,    /* Links the IP to a previous handle */
                     paRouteInfo_t *,     /* Where to send a match */
                     paRouteInfo_t *,     /* Where to send a next stage match failure */
                     paHandleL2L3_t *,    /* Returns associated handle */
                     paCmd_t,             /* The generated PASS command */
                     uint16_t *,          /* Size of command buffer used */
                     paCmdReply_t *,      /* Where to send the PASS reply */
                     int *);              /* Where to send the command */
```

5.2.8.3 Implementation

An internal L3 table entry is made if not already present. If an identical entry is in the table then that handle is returned and the size of the passCmd is set to 0. This function returns error if the cmdReplyDest is not host, the command buffer is too small, or other error conditions occur. Then this function formats a LUT1 configuration command based on the IP match criteria and the routing information. The configuration command should be forwarded to PDSP1 for entry into LUT1-1 if no linking handle is specified or if the linking handle is a MAC handle. Otherwise, it should be forwarded to PDSP2 for entry into LUT1-2. The module user can overwrite the default rules by specifying the desired LUT1 instance. The required command destination is provided by this function.

5.2.8.4 Return Values

pa_OK
pa_INSUFFICIENT_CMD_BUFFER_SIZE
pa_ERR_CONFIG
pa_INVALID_DUP_ENTRY
pa_INVALID_TABLE_MORE_SPECIFIC_ENTRY_PRESENT
pa_INVALID_IP_FLOW
pa_INVALID_INPUT_HANDLE
pa_INVALID_CMD_REPLY_DEST
pa_HANDLE_TABLE_FULL

passHandle will contain the handle information for this IP address.

5.2.9 Set Custom LUT1 Configuration

5.2.9.1 Description

This function performs the global configuration for a level 3 (LUT1) custom lookup. It specifies the offset and byte masks which the PA subsystem uses for parsing a packet that has entered custom level 3 (LUT1) classification directed from the previous match route. This function is called for each LUT1 custom lookup type referred by the custom index. The command reply routing is optional because this command is always processed by the PA sub-system.

5.2.9.2 Prototype

```
result = Pa_setCustomLUT1 (Pa_Handle,        /* Driver handle */
                           int,              /* Cutsom LUT1 index */
                           int,              /* Specify which LUT1 (0-2) should be used */
                           uint16_t ,        /* Where the PA begins custom match
                                                (from L3 start) */
                           uint8_t*,         /* byte array of the bit-mask */
                           paCmd_t ,         /* The generated PASS command */
                           uint16_t *,       /* Size of command buffer used */
                           paCmdReply_t *,   /* Where to send the PASS reply */
                           int *);           /* Where to send the command */
```

5.2.9.3 Implementation

Format a global custom LUT1 configuration command to be sent to the PA sub-system.

5.2.9.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFIENT_CMD_BUFFER_SIZE

5.2.10 Add Custom LUT1 entry

5.2.10.1 Description

This function adds a custom LUT1 entry to a level 3 lookup. A Null value for L2L3Handle means that this entry is independent of any previous header. The custom L3 entry will be added to the first available location in the LUT1 if the input index is not specified. Otherwise, it will use the LUT1 location specified by the input index. The command response is mandatory since it contains the LUT1 index for this custom L3 entry. Therefore, the command reply destination (see section 5.1.7) must be host.

5.2.10.2 Prototype

```
result = Pa_addCustomLUT1 (Pa_Handle,        /* Driver handle */
                           int,              /* Cutsom LUT1 index  */
```

```
int,                 /* Index of the LUT1 entry */
uint8_t *,           /* Byte array of the matching value */
paHandleL2L3_t *,    /* Links the custom L3 entry to
                        a previous handle */
paRouteInfo_t *,     /* Where to send a match */
paRouteInfo_t *,     /* Where to send a next stage match failure
                      */
paHandleL2L3_t *,    /* Returns associated handle */
paCmd_t,             /* The generated PASS command */
uint16_t *,          /* Size of command buffer used */
paCmdReply_t *,      /* Where to send the PASS reply */
int *);              /* Where to send the command */
```

5.2.10.3 Implementation

An internal L3 table entry is made if not already present. If an identical entry is in the table then that handle is returned and the size of the passCmd is set to 0. This function returns error if the cmdReplyDest is not host, the command buffer is too small, or other error conditions occur. Then this function formats a LUT1 configuration command based on the custom L3 match criteria and the routing information. The configuration command should be forwarded to PDSP1 for entry into LUT1-1 if no linking handle is specified or if the linking handle is a MAC handle. Otherwise, it should be forwarded to PDSP2 for entry into LUT1-2. The module user can overwrite the default rules by specifying the desired LUT1 instance. The required command destination is provided by this function.

5.2.10.4 Return Values

pa_OK
pa_INSUFFICIENT_CMD_BUFFER_SIZE
pa_ERR_CONFIG
pa_INVALID_DUP_ENTRY
pa_INVALID_TABLE_MORE_SPECIFIC_ENTRY_PRESENT
pa_INVALID_INPUT_HANDLE
pa_INVALID_CMD_REPLY_DEST
pa_HANDLE_TABLE_FULL

passHandle will contain the handle information for this custom LUT1 entry.

5.2.11 Add TCP/UDP/GTPU Destination entry

5.2.11.1 Description

This function adds a TCP/UDP destination port or GTU-U Tunnel ID to the level 4 (LUT2) table. The L3Handle is used to associate a MAC/IP address with the destination port. The formatted command must be retained after sending to the PA until the reply is accepted by the LLD through a call to paForwardResult. The command response is mandatory since this command can be rejected if the sub-system queue to add destination ports is full. Therefore, the command reply destination (see section 5.1.7) must be host. This API also initiates the atomic queue diversion operation, which means that the QMSS moves the entries in the diverted queue to the destination queue, if the divertedQueue is specified and replace flag is set. In this case, the PASS will complete the LUT2 update, wait for the queue diversion to be complete and then process incoming packets..

5.2.11.2 Prototype

```
result = Pa_addPort ( Pa_Handle,          /* Driver handle */
                      uint16_t,           /* Port size (16 or 32) */
                      uint16_t,           /* Destination port */
                      paHandleL2L3_t,     /* L3 linking handles */
                      uint16_t replace    /* Flag to indicate whether the entry exists */
                      uint16_t divertQ,   /* diverted queue for atomic queue diversion  with
                                             LUT2 update*/
                      paRouteInfo_t,      /* Where to send a match */
                      paHandleL4_t,       /* Returns associated handle */
                      paCmd_t,            /* The generated PASS command */
                      uint16_t *,         /* The generated command size */
                      paCmdReply_t *,     /* Where to send the PASS reply */
                      int *);             /* Where to send the command */
```

5.2.11.3 Implementation

A LUT2 configuration command to add an entry to LUT2 is generated to be sent to PDSP 3. No internal table is maintained for TCP/UDP/GTP-U ports. Instead the returned handle pointer incorporates the port number and L3 handle LUT1 index value. If the diverted queue is specified, both the source queue and destination queue number will be provided in the command packet with a control flag set.

5.2.11.4 Return Values

pa_OK
pa_INSUFFICIENT_CMD_BUFFER_SIZE
pa_INVALID_INPUT_HANDLE
pa_INVALID_CMD_REPLY_DEST

passHandle will contain the handle information for destination port.

5.2.12 Set Custom LUT2 Configuration

5.2.12.1 Description

This function performs the global configuration for a level 4 (LUT2) custom lookup. It specifies the offset and byte mask arrays which the PA subsystem uses for parsing a packet that has entered custom level 4 (LUT2) classification directed from the previous match route. If handleLink is true then only 3 byte masks and 3 offsets are available for matching. The fourth one is used to store the previous match information. In this case only the first 3 values in the Offsets and Byte Masks arrays are valid. If setMask is non-zero, it will be OR with the first byte mask. It can be used to distinguish this particular custom L4 lookup entry with other custom L3 or standard lookup entries. This function is called for each LUT2 custom lookup referred by the custom index. The command reply routing is optional because this command is always processed by the PA sub-system.

5.2.12.2 Prototype

```
result = Pa_setCustomLUT2 (Pa_Handle,         /* Driver handle */
                           int,               /* Cutsom LUT2 index */
                           uint16_t ,         /* handleLink: TRUE to use previous link */
                           uint16_t*,         /* offset array to the bytes to use in custom
                                                 matching */
                           uint8_t*,          /* byte array of the bit-mask */
                           uint8_t,           /* bits to be set at the first match byte */
                           paCmd_t ,          /* The generated PASS command */
                           uint16_t *,        /* Size of command buffer used */
                           paCmdReply_t *,    /* Where to send the PASS reply */
                           int *);            /* Where to send the command */
```

5.2.12.3 Implementation

Format a global custom LUT2 configuration command to be sent to the PA sub-system.

5.2.12.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFIENT_CMD_BUFFER_SIZE

5.2.13 Add Custom LUT2 entry

5.2.13.1 Description

This function adds a custom L4 (LUT2) entry to the LUT2 table. The L3Handle is used to associate a MAC/IP address with the custom L4 entry. The formatted command must be retained after sending to the PA until the reply is accepted by the LLD through a call to paForwardResult.

The command response is mandatory since this command can be rejected if the sub-system queue to add L4 entry is full. Therefore, the command reply destination (see section 5.1.7) must be host. This API also initiates the atomic queue diversion operation, which means that the QMSS moves the entries in the diverted queue to the destination queue, if the divertedQueue is specified and replace flag is set. In this case, the PASS will complete the LUT2 update, wait for the queue diversion to be complete and then process incoming packets..

5.2.13.2 Prototype

```
result = Pa_addCustomLUT2 (Pa_Handle,          /* Driver handle */
                           int,                /* L4 (LUT2) custom index */
                           uint8_t*,           /* array of match bytes */
                           paHandleL2L3_t,     /* L3 linking handles */
                           uint16_t replace,   /* Flag to indicate whether the entry exists
          */
                           uint16_t divertQ,   /* diverted queue for atomic queue diversion
                                                  with LUT2 update*/
                           paRouteInfo_t,      /* Where to send a match */
                           paHandleL4_t,       /* Returns associated handle */
                           paCmd_t,            /* The generated PASS command */
                           uint16_t *,         /* The generated command size */
                           paCmdReply_t *,     /* Where to send the PASS reply */
                           int *);             /* Where to send the command */
```

5.2.13.3 Implementation

A LUT2 configuration command to add an entry to LUT2 is generated to be sent to PDSP 3. No internal table is maintained for custom LUT2 entries. Instead the returned handle pointer incorporates the custom LUT2 entry and L3 handle LUT1 index value. If the diverted queue is specified, both the source queue and destination queue number will be provided in the command packet with a control flag set.

5.2.13.4 Return Values

pa_OK
pa_INSUFFICIENT_CMD_BUFFER_SIZE
pa_INVALID_INPUT_HANDLE
pa_INVALID_CMD_REPLY_DEST

passHandle will contain the handle information for custom L4 entry.

5.2.14 Delete L2/L3 (LUT1) Handle/Configuration

5.2.14.1 Description

This function deletes a L2/L3 (LUT1) handle.

5.2.14.2 Prototype

```
result = Pa_DelHandle (   Pa_Handle,           /* Driver handle */
                          paHandleL2L3_t *,    /* Point to the handle to delete */
                          paCmd_t ,            /* The generated PASS command */
                          uint16_t *,          /* The size of the command */
                          int *);              /* Where to send the command */
```

5.2.14.3 Implementation

The configuration command to the appropriate PDSP is generated to delete a table entry. No PA reply routing is required because this command is always processed by the PA sub-system.

5.2.14.4 Return Values

pa_OK
pa_INVALID_INPUT_HANDLE
pa_INSUFFICIENT_CMD_BUFFER_SIZE
pa_HANDLE_INACTIVE

5.2.15 Delete an L4 (LUT2) handle

5.2.15.1 Description

This function deletes an L4 (LUT2) handle

5.2.15.2 Prototype

```
Result = Pa_DelL4Handle ( Pa_Handle,           /* Driver handle */
                          paHandleL4_t,        /* The handle to delete */
                          paCmd_t,             /* The generated PASS command */
                          uint16_t *,          /* The size of the command */
                          paCmdReply_t,        /* Where to send the reply */
                          int *);              /* Where to send the command */
```

5.2.15.3 Implementation

The configuration command to the appropriate PDSP is generated to delete an L4 (LUT2) table entry. Unlike the L2/L3 deletion, this command can fail because the PDSP was processing more configuration commands than it can buffer. In that case the command reply will indicate this and the command must be re-sent to the PA sub-system.

5.2.15.4 Return Values

pa_OK
pa_INVALID_INPUT_HANDLE
pa_INSUFFICIENT_CMD_BUFFER_SIZE

5.2.16 Configure Exception Routes

5.2.16.1 Description

By default all exception routes including error conditions and lookup failures result in discarding the packet and incrementing a stat. This function allows for overriding this operation. The parameter nRoute is the number of routes being setup, condition is the routing condition (section 5.1.6) and "routes" is an array of routing information (section 5.1.9). The command reply routing is optional because this command is always processed by the PA sub-system. This function may be invoked to configure and update a set of global routes multiple times.

5.2.16.2 Prototype

```
result = Pa_configExceptionRoute (Pa_Handle,        /* Driver handle */
                                  int nRoute        /* Number of exception routes to configure */
                                  int *,            /* Array of exception route condition */
                                  paRouteInfo_t *,  /* Array of routing info */
                                  paCmd_t ,         /* The generated PASS command */
                                  uint16_t *,       /* The generated command size */
                                  paCmdReply_t *,   /* Where to send the PASS reply */
                                  int *);           /* Where to send the command */
```

5.2.16.3 Implementation

This function converts the exception routing information to a configuration command to be forwarded to the PA .The PA reply routing is optional because this command is always processed by the PA sub-system.

5.2.16.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFIENT_CMD_BUFFER_SIZE

5.2.17 Configure Multi-route

5.2.17.1 Description

This function performs the global configuration for a multi-route set which consists of up to 8 multi-route entries. The multi-route group is created and referred to based on the multi-route index. Once the multi-route group is created through a call to this function it remains effective until the function is called again to explicitly overwrite its content. It is not recommended to update a multi-route group when it is still used by one or more packet routes. The command reply routing is optional because this command is always processed by the PA sub-system.

5.2.17.2 Prototype

```
result = Pa_configMultiRoute (Pa_Handle,          /* Driver handle */
                              int,                /* Multi-route index */
                              int,                /* Number of routing entries
                                                     specified */
                              paMultiRouteEntry_t*, /* Array of routing information*/
                              paCmd_t ,           /* The generated PASS command */
                              uint16_t *,         /* Size of command buffer used */
                              paCmdReply_t *,     /* Where to send the PASS reply */
                               int *);            /* Where to send the command */
```

5.2.17.3 Implementation

Format a global multi-route configuration command to be sent to the PA sub-system.

5.2.17.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFIENT_CMD_BUFFER_SIZE

5.2.18 Configure Command Set

5.2.18.1 Description

This function performs the global configuration for a command set which consists of a stack of commands. The commands within a command set will be executed in order after a LUT1 or LUT2 match if it is specified by the routing information. The command set is created and referred to according to the command set index. Once the command set is created through a call to this function it remains effective until the function is called again to explicitly overwrite its content. It is not recommended to update a command set when it is still used by one or more packet routes. The command reply routing is optional because this command is always processed by the PA sub-system.

The commands within the command set will be executed in order. The module user is responsible for placing the commands in such ways that the packet offsets required by commands should be in ascending order. The following commands are supported:

- pa_CMD_REMOVE_HEADER
- pa_CMD_COPY_DATA_TO_PSINFO
- pa_CMD_CRC_OP
- pa_CMD_PATCH_DATA
- pa_CMD_REMOVE_TAIL
- pa_CMD_NEXT_ROUTE
- pa_CMD_MULTI_ROUTE
- pa_CMD_USR_STATS

5.2.18.2 Prototype

```
result = Pa_configCmdSet (  Pa_Handle,        /* Driver handle */
                            int,              /* command set index */
                            int,              /* Number of commands */
                            paCmdInfo_t*,     /* Array of command configurations */
                            paCmd_t ,         /* The generated PASS command */
                            uint16_t *,       /* Size of command buffer used */
                            paCmdReply_t *,   /* Where to send the PASS reply */
                             int *);          /* Where to send the command */
```

5.2.18.3 Implementation

Format a global command-set configuration command to be sent to the PA sub-system.

5.2.18.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFIENT_CMD_BUFFER_SIZE

5.2.19 Configure CRC Engine

5.2.19.1 Description

This function is called to generate command to configure the CRC engine within the PA sub-system. There are 6 CRC engines in the PA sun-system. Each CRC engine is connected to its corresponding PDSP. It performs CRC operation required by the some network protocol such as SCTP and/or the user-specified CRC command. The CRC engine can be only accessed by its corresponding PDSP. Therefore, it is referred by the PDSP number. The command reply routing is optional because this command is always processed by the PA sub-system.

5.2.19.2 Prototype

```
result = Pa_configCRCEngine (Pa_Handle,   /* Driver handle */
```

```
                          int,               /* CRC engine index (PDSP number) */
                          paCrcConfig_t      /* CRC configuration information */
                          paCmd_t ,          /* The generated PASS command */
                          uint16_t *,        /* Size of command buffer used */
                          paCmdReply_t *,    /* Where to send the PASS reply */
                          int *);            /* Where to send the command */
```

5.2.19.3 Implementation

Format a CRC engine configuration command to be sent to the PA sub-system.

5.2.19.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFIENT_CMD_BUFFER_SIZE

5.2.20 Configure Timestamp

5.2.20.1 Description

This function is called to configure the 16-bit timer reserved to generate the 32-bit system timestamp. The system timestamp will be inserted into the timestamp field in the packet descriptor for all input packets. It can be inserted into the outgoing packets per tx command. The 16-bit timer connected to PDSP0 is reserved for timestamp generation.

5.2.20.2 Prototype

```
result = Pa_configTimestamp (  Pa_Handle,              /* Driver handle */
                               paTimestampConfig_t);   /* Timestamp configuration
                                                          information */
```

5.2.20.3 Implementation

The function uses the CSL layer to make direct writes to the PASS Timer registers. Because these registers exist in config space and a read is always required, this command can block while the config bus is unavailable.

5.2.20.4 Return Values

pa_OK
pa_ERR_CONFIG

5.2.21 Query System Timestamp

5.2.21.1 Description

This function returns the 48-bit system timestamp.

5.2.21.2 Prototype

```
result = Pa_getTimestamp (
                Pa_Handle,            /* Driver handle */
                paTimestamp_t *);     /* Pointer to the system timestamp */
```

5.2.21.3 Implementation

Extract and report the 48-bit system timestamp.

5.2.21.4 Return Values

pa_OK

5.2.22 Configure User-defined Statistics

5.2.22.1 Description

This function performs the counter configuration for the user-defined statistics which consists of up to 256 multi-level hierarchical counters. Each counter can be linked to one of the next level counter. All counters in its linking chain will be incremented when the lowest level counter is updated. The module user can specify the type of each counter and how the counter is linked to the next level counter. It is not recommended to re-configure the user-defined statistics when one or more counters are still used by PASS. The command reply routing is optional because this command is always processed by the PA sub-system.

5.2.22.2 Prototype

```
result = Pa_configUsrDefinedStats (
                Pa_Handle,                  /* Driver handle */
                paUsrStatsConfigInfo_t*,    /* Configuration information*/
                 paCmd_t ,                  /* The generated PASS command */
                uint16_t *,                 /* Size of command buffer used */
                paCmdReply_t *,             /* Where to send the PASS reply */
                 int *);                    /* Where to send the command */
```

5.2.22.3 Implementation

Record, verify the array of counter configurations and format the counter configuration command to be sent to the PA sub-system.

5.2.22.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFIENT_CMD_BUFFER_SIZE

5.2.23 Forward PASS Result

5.2.23.1 Description

The PA APIs to add and delete SRIO, MAC, IP, and UDP/TCP port entries will generate configuration commands to be sent to the PASS via the queue manager. The PASS will generate replies for these commands, and these replies must be then forwarded back to the LLD by calling this function which processes the command response and converts the command error code to the corresponding error return. Failure to forward the result will result in a dangling entry in the LUT1 table that cannot be deleted, and creation of future handles which attempt to refer to this handle will fail. When adding entries to LUT2 (UDP/TCP) it is possible for the command to be rejected because the local command queue is full. In that case the command must be re-submitted.

5.2.23.2 Prototype

```
Result = Pa_forwardResult ( Pa_Handle,         /* Driver handle */
                            void *,            /* Response packet from the sub-system */
                            paEntryHandle_t *, /* Returns associated handle */
                            int *,             /* Identifies handle type */
                            int *);            /* Resubmit command destination */
```

5.2.23.3 Implementation

Contained in the reply from the PASS is the resulting LUT 1 index which holds the specified MAC or IP address, as well as any error codes. The error codes are translated and returned by this function.

5.2.23.4 Return Values

pa_OK
pa_LUT_ENTRY_FAILED
pa_WARN_ACTIVE_HANDLE_ACKED - an ack received for already active entry
pa_RESUBMIT_COMMAND

5.2.24 Request Statistics

5.2.24.1 Description

Calling this function returns the command packet to request statistics from the PASS. Statistics can be optionally cleared after reading.

5.2.24.2 Prototype

```
result = Pa_requestStats (  Pa_Handle,        /* Driver handle */
                            uint16_t,         /* TRUE to clear stats after read */
                            paCmd_t ,         /* The generated PASS command */
                            uint16_t *,       /* Size of command buffer used */
                            paCmdReply_t *,   /* Where the statistics are sent */
```

```
int *);                 /* Where to send the command */
```

5.2.24.3 Implementation

The request stats command is generated to be forwarded to the PA sub-system.

5.2.24.4 Return Values

pa_OK
pa_INVALID_CMD_BUFFER_SIZE

5.2.25 Format Statistics Output

5.2.25.1 Description

The packet returned from the PASS in response to a request stats command will be in big Endian format regardless of the Endianness of the CorePacs. This function will format the stats response to match the CorePac.

5.2.25.2 Prototype

```
paSysStats_t *Pa_formatStatsReply ( Pa_Handle,  /* Driver handle */
                                    paCmd_t );  /* Stats reply from PA */
```

5.2.25.3 Implementation

The stats reply structure is formatted to match the Endianness of the CorePac.

5.2.25.4 Return value

The returned value is a pointer to the reformatted stats reply.

5.2.26 Request User-Defined Statistics

5.2.26.1 Description

This function is used to query the user-defined statistics from the sub-system. The statistics will be formatted and copied to the buffer provided.The sub-system statistics can be then optionally cleared if doClear is set.In this case, the command buffer (cmd) contains a formatted command for the sub-system. The destination for the command is provided in cmdDest. The module user must send the formatted command to the sub-system.

5.2.26.2 Prototype

```
result = Pa_requestUserStats (
                    Pa_Handle,          /* Driver handle */
                    uint16_t,           /* TRUE to clear stats after read */
                    paCmd_t ,           /* The generated PASS command */
```

```
                                uint16_t *,              /* Size of command buffer used */
                                paCmdReply_t *,          /* Where the statistics are sent */
                                int *,                   /* Where to send the command */
                                paUsrStats_t  *pUsrStats); /* Pointer to the usrStats buffer */
```

5.2.26.3 Implementation

The request user-defined stats command is generated to be forwarded to the PA sub-system.

5.2.26.4 Return Values

pa_OK
pa_INVALID_CMD_BUFFER_SIZE

5.2.27 Format Transmit Routing with Checksum Requests

5.2.27.1 Description

When transmitting packets, the PASS can compute two checksums on each entry. The information required for the checksum generation and the routing are placed in the protocol specific words in the CPPI packet descriptor. Unlike other APIs of the LLD this one does not format the command in a buffer which is intended to be sent to the PASS, but instead formats a memory block which is copied to the protocol specific section of the CPPI packet descriptor for a transmitted packet. This API may be called only once, and the same protocol specific section can be used for every packet in the channel. If the length of the checksum area changes with each packet the macro PASS_SET_TX_CHKSUM_LENGTH (see section 5.3.5).

5.2.27.2 Prototype

```
result = Pa_formatTxRoute ( paTxChksum_t *,      /* Chksum  info 0 */
                            paTxChksum_t *,      /* Chksum info 1 */
                            paRouteInfo_t *,     /* Where to send packet after checksum */
                            void*,               /* Result buffer */
                            uint16_t *);         /* Result buffer size */
```

5.2.27.3 Implementation

The protocol specific information block is generated. This block contains the checksum and routing info for transmitted packets.

5.2.27.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFICIENT_CMD_BUFFER_SIZE

5.2.28 Format Transmit Routing with Blind Patch

5.2.28.1 Description

When transmitting packets, the PASS can perform data patching on each entry. The information required for the data patching and the routing are placed in the protocol specific words in the CPPI packet descriptor. This API may be called only once, and the same protocol specific section can be used for every packet in the channel.

5.2.28.2 Prototype

```
result = Pa_formatRoutePatch (paRouteInfo_t *,    /* Where to send packet after patch */
                              paPatchInfo_t *,    /* Patch information */
                              void*,              /* Result buffer */
                              uint16_t *);        /* Result buffer size */
```

5.2.28.3 Implementation

The protocol specific information block is generated. This block contains the patching and routing info for transmitted packets.

5.2.28.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFICIENT_CMD_BUFFER_SIZE

5.2.29 Format Transmit Commands

5.2.29.1 Description

When transmitting packets, the PASS can execute a stack of commands on each packet. This function is used to create, append and update the stack of commands which will be executed by the packet accelerator and security accelerator sub-systems to perform a sequence of actions on the packet. The command block should be attached to data packets in the protocol specific section of the packet descriptor. This API may be called multiple times to add or update the command block. The same protocol specific section can be used for every packet in the channel. Multiple MACROs may be used to update some parameters such as packet length in the command buffer for each packet. This API supports the following commands:

- pa_CMD_NEXT_ROUTE
- pa_CMD_CRC_OP
- pa_CMD_PATCH_DATA
- pa_CMD_TX_CHECKSUM
- pa_CMD_INSERT_TIMESTAMP
- pa_CMD_SA_PAYLOAD
- pa_CMD_IP_FRAGMENT
- pa_CMD_PATCH_MSG_LEN

5.2.29.2 Prototype

```
result = Pa_formatTxCmds (  int,              /* Number of commands */
                            paCmdInfo_t*,     /* Array of command configurations */
                            uint16_t,          /* The command buffer location where the
                                                  new commands are inserted */
                            paCmd_t ,         /* The generated PASS command */
                            uint16_t *);      /* Size of command buffer used */
```

5.2.29.3 Implementation

The protocol specific information block is generated. This block contains the set of commands to be executed on transmitted packets.

5.2.29.4 Return Values

pa_OK
pa_ERR_CONFIG
pa_INSUFFICIENT_CMD_BUFFER_SIZE

5.2.30 Reset/Enable/Get PASS state

5.2.30.1 Description

This function will set or release reset for the PASS subsystem. Because this command results in direct access to the PASS it could be blocking. Because of this a macro is provided which the system can use to provide its own transport mechanism.

5.2.30.2 Prototype

```
Result = Pa_resetControl (Pa_Handle,    /* Driver handle */
                          paSSstate_t);  /* The desired state */
```

The following values are used both for the input argument and the result.

```
Enum newState  {
  pa_STATE_RESET,                  /* Input arg and output result */
  pa_STATE_ENABLED,                /* Input arg and output result */
  pa_STATE_QUERY,                  /* Input arg only                */
  pa_STATE_INCONSISTENT            /* Output only                   */
  pa_STATE_INVALID_REQUEST
};
```

They are also defined at section 5.1.15.

5.2.30.3 Implementation

The function uses the CSL layer to make direct writes to the PASS control registers. Because these registers exist in config space and a read is always required, this command can block while the config bus is unavailable.

5.2.30.4 Return value

The return values are the same as the input value, except that pa_STATE_QUERY will never be returned. The value pa_STATE_INCONSISENT means that some of the elements of the PASS are out of reset, but others are in reset.

5.2.31 Download PA Image

5.2.31.1 Description

This function is used to download a PDSP image to one of the 6 PDSPs. Because this command results in direct access to the PASS it could be blocking. Because of this a macro is provided which the system can use to provide its own transport mechanism.

5.2.31.2 Prototype

```
Result = Pa_DownloadImage (    Pa_Handle,    /*Driver handle */
                               int,          /* Identifies the PDSP */
                               void*,        /* The image source */
                               int);         /* The image size */
```

5.2.31.3 Implementation

The LLD will check the state of the PDSP prior to writing the image. An error is returned if the specified PDSP is no in reset. Otherwise the image is copied to the PDSP program memory using direct writes.

5.2.31.4 Return values

pa_OK
pa_SYSTEM_INVALID_STATE

5.2.32 Get Handle Reference Count

5.2.32.1 Description

The LLD maintains the reference counter for LUT1 handles: MAC/IP. Given a handle, the LLD would return how many references are being used in next header entry by invoking the function

5.2.32.2 Prototype

```
Result = Pa_getHandleRefCount (   Pa_Handle,          /* Driver handle */
                                  paHandleL2L3_t *,   /* The handle to delete */
                                  uint16_t*);         /* The reference count */
```

5.2.32.3 Implementation

The LLD will return the reference count associated with the link.

5.2.32.4 Return values

pa_OK

5.2.33 Get LLD Version Number

5.2.33.1 Description

This function is used to get the version information of the PA LLD in 0xAABBCCDD format where Arch (AA); API Changes (BB); Major (CC); Minor (DD)

5.2.33.2 Prototype

```
uint32_t  = Pa_getVersion (void);
```

5.2.33.3 Implementation

The LLD will return its version number.

5.2.33.4 Return values

PA LLD version number

5.2.34 Get LLD Version String

5.2.34.1 Description

This function is used to get the version string of the PA LLD.

5.2.34.2 Prototype

```
const char* Pa_getVersionStr (void);
```

5.2.34.3 Implementation

The LLD will return its version string.

5.2.34.4 Return values

PA LLD version string

5.3 Macros

To separate the commands from the transport layer, each macro makes use of the following macros that must be defined outside the system. These macros are only used by the pass utility macros in this section.

SYSTEM_WRITE32(address32, uvalue32)
SYSTEM_COPY(destAddr32, srcAddr32, sizeBytes)
X = SYSTEM_READ32(address32)

5.3.1 Reset Subsystem

5.3.1.1 Description

The subsystem consists of multiple PDSPs and hardware sub-modules. A subsystem reset will assert reset all of these modules.

5.3.1.2 Invocation

pa_RESET_SUBSYSTEM()

5.3.1.3 Implementation

The macro performs multiple invocations of SYSTEM_WRITE32 to put every PDSP into reset.

5.3.2 Enable Subsystem

5.3.2.1 Description

Enable subsystem removes all systems from reset

5.3.2.2 Invocation

pa_ENABLE_SUBSYSTEM()

5.3.2.3 Implementation

The macro performs multiple invocations of SYSTEM_WRITE32 to release every PDSP from reset.

5.3.3 Download Image

5.3.3.1 Description

An image is downloaded into a single PDSP. The PDSP must be in reset or unpredictable behavior from the PASS could occur.

5.3.3.2 Invocation

pa_DOWNLOAD_IMAGE(moduleId, address32, sizeBytes)

5.3.3.3 Implementation

The macro invokes SYSTEM_COPY to copy the image to the desired module. Module IDs have values 0-5 but are not symbolically enumerated.

5.3.4 Get Reset State

5.3.4.1 Description

This macro returns the reset state of the PASS.

5.3.4.2 Invocation

pa_GET_SYSTEM_STATE()

5.3.4.3 Implementation

The macro invokes the SYSTEM_READ macro to determine the operational state of the subsystem. The result is a value as defined in section 5.1.15.

5.3.5 Set tx checksum length

5.3.5.1 Description

This macro is used to adjust the length field in the data formatted through the passFormatTxRoute API. This prevents the need to call the passFormatTxRoute function for a change in only the length field of one of the checksums (specified in blockNum), typically the TCP/UDP checksum.

5.3.5.2 Invocation

pa_SET_TX_CHKSUM_LENGTH(data *, blockNum, length)

5.3.5.3 Implementation

The data field is a fixed format field created by the passFormatTxRouteFunction. The macro changes one of the two checksum length fields.

5.4 Call-Out APIs

The LLD uses the OSAL procedure for call outs. The PA performs a call out whenever it will read or write to the handle tables, and again when the access is complete. The application using the LLD must implement these call outs to allow only one access to the table at a time. If the application is using the LLD with each core having its own table, then this call out must only prevent access from the same core. If the application is not multi-threaded and if these calls cannot happen from an interrupt then the calls can be removed. For the case where a single set of tables is used across cores the application must implement a cross core semaphore to prevent multiple simultaneous accesses to the tables.

5.4.1 Pa_osalBeginMemAccess

5.4.1.1 Description

This function is called by the LLD before it begins reading or writing to PA managed internal resources include its instance, L2 and L3 tables and the potential user stats link table.

5.4.1.2 Prototype

void Osal_paBeginMemAccess void* addr, uint32_t sizeWords);

5.4.2 Pa_osalEndMemAccess

5.4.2.1 Description

This function is called by the LLD when it is done reading or writing to PA managed internal resources include it's instance, L2 and L3 tables and the potential user stats link table.

5.4.2.2 Prototype

void Osal_paEndMemAccess (Ptr addr, uint32_t sizeWords);

5.4.3 Pa_osalMtCsEnter

5.4.3.1 Description

This function is called by the LLD to provide critical section to protect its global and shared resources access from multiple cores. .

5.4.3.2 Prototype

void Osal_paMtCsEnter (uint32_t *key);

5.4.4 Pa_osalMtCsExit

5.4.4.1 Description

This function is called by the LLD to exit a critical section protected using the Osal_paMtCsEnter() API..

5.4.4.2 Prototype

void Osal_paMtCsExit (uint32_t key);

6 PASS PDSP Firmware Enhancements

The PASS PDSP firmware should be enhanced to support several PA 1.2 features. This section describes the detailed design in term of interface, control structure, algorithm of the following features.

- IP Fragmentation
- PA -assisted IP Reassembly

6.1 IP Fragmentation

6.1.1 Description

The Internet Protocol allows IP fragment so that datagrams can be fragmented into pieces small enough to pass over a link with a smaller MTU than the original datagram size. The IP fragmentation is performed at the Modifier PDSP (PDSP4/5) when the pa_CMD_IP_FRAGMENT command is issued. The transmit IP packets will be divided into smaller IP fragments based on the specified MTU size and forwarded to the destination queue.

6.1.2 Data Structures

The IP fragmentation control block is defined at the following table.

Name	Description
ipLength	The total IP length including IP header
ipOffset	The offset to the IP header from the top of the packet
mtuSize	The desired MTU size
ipHdrLength	The IP header size
baseOffset	The original payload offset. It will be non-zero only if the original packet is fragmented.
payloadSize	The payload size of the current fragment
loopOffset	The offset to the payload of the next fragment

6.1.3 Implementation

The PDSP will initialize the control block based on the IP fragment command and the original IP header and perform the following procedure until the last fragment is forwarded:

- Calculate the fragment payload size and offset.
- Update the IP header and issue IP checksum command.
- Issue CDE command to flush out the payload up to the fragment offset.
- Issue CDE command to move packet window by the payload size of this fragment.
- Issue CDE command to flush out the rest of payload.
- If it is the last fragment, issue CDE command to forward this fragment and exit.
- If it is not the last fragment, issue CDE command to copy the original packet and forward the fragment.
- Repeat this procedure.

6.2 PA -assisted IP Reassembly

6.2.1 Description

The current version of PASS does not support IP reassembly, all the IP fragments are detected and forwarded to and reassembled at host. The reassembled IP packet may be forwarded back to PASS for continuous classification. The drawback of this approach is that the order of the incoming packets will not be maintained.

To provide better support for IP reassembly, the PA-assisted IP Reassembly operation is introduced and summarized below:

- Array of traffic flows which consist of source IP, destination IP, protocol and counter are maintained at PDSP.
- Traffic flow is activated by the PDSP when the first IP fragment is detected and forwarded.
- Traffic flow is freed when its packet count reaches 0
- All packets belong to any active traffic flow will be forwarded to the host so the packet order will be maintained.
- Number of active traffic flow is configurable [0, 32]
- IP fragments should be forwarded to host with “none” traffic flow id if no traffic flow is available. In this case, the packet order is not guaranteed to be maintained.

The PA-assisted IP reassembly host requirements are defined at section 7.1. The host IP reassembly module is not part of PA LLD. But, a sample implementation will be provided as part of PA LLD release package.

We need to document that the 64-byte wire rate throughput will not be guaranteed when this feature is enabled and there are active traffic flows.

6.2.2 Data Structures

This section describes both the PDSP internal data structures and the PDSP-Host interface.

PDSP Traffic Flow:

Name	Description
srcIp	Source IP address
destIP	Destination IP address
Proto	Protocol field at the IPv4 header
Count	Number of pending IP fragments and packets

PDSP Traffic Flow Control Block:

Name	Description
numTF	Maximum number of Traffic Flow entries
numActiveTF	Number of active Traffic Flows
tfMap	32-bit TF bitmap. 0: inactive 1: active
destQueue	The destination queue where PASS will deliver the packets which require reassembly assistance
destFlowId	Specify the CPPI flow which instructs how free queues are used for receiving packets

Host-PDSP interface

The following parameters should be provided in the CPPI packet descriptor such as protocol-specific information area

Name	Description
tfIndex	Traffic Flow index where 0xFF indicates traffic flow is not available
Count	Number of fragments in the reassembled packet
ctrlFlag	The IP Reassembly operation control information as defined below • pafrm_IP_REASSM_2NDPASS (to-PDSP only) o Set: It is the second pass (from host) o Clear: It is the first pass (from network)

The following Macros are provided to extract and set the desired parameters inside the 24-byte packet data which resides in the CPPI protocol-specific information area:

Name	Description
PASAHO_LINFO_READ_TFINDEX(x)	Extract the IP Reassembly Traffic Flow Index
PASAHO_LINFO_READ_FRANCNT(x)	Extract the IP Reassembly Fragment count
PASAHO_LINFO_SET_TFINDEX(x, v)	Set the IP Reassembly Traffic Flow Index
PASAHO_LINFO_SET_FRANCNT(x, v)	Set the IP Reassembly Fragment count
PASAHO_LINFO_IS_IPSEC(x)	Indicate whether it is an IPSEC packet
PASAHO_LINFO_IS_IPSEC_ESP(x)	Indicate whether it is an IPSEC ESP packet
PASAHO_LINFO_IS_IPSEC_AH(x)	Indicate whether it is an IPSEC AH packet
PASAHO_LINFO_CLR_IPSEC(x)	Clear IPSEC indication bits
PASAHO_LINFO_CLR_IPSEC_ESP(x)	Clear IPSEC ESP indication bit
PASAHO_LINFO_CLR_IPSEC_AH(x)	Clear IPSEC AH indication bit
PASAHO_LINFO_CLR_FLAG_FRAG(x)	Clear the fragmentation found flag
PASAHO_LINFO_SET_START_OFFSET(x, v)	Update the next parse start offset
PASAHO_LINFO_SET_END_OFFSET(x, v)	Update the end of packet parse offset
PASAHO_LINFO_SET_2ND_PASS(x, v)	Set the end of packet parse offset
PASAHO_LINFO_SET_NULL_PKT_IND(x, v)	Set the null packet flag which indicates that the packet should be dropped

6.2.3 Implementation

The PDSP will perform the flowing actions:

First Pass: (traffics from network)

- Search to find the matching traffic flow (cycle :12 * number of active traffic flows)
- If traffic flow is identified, increment its packet counter and forward the packet with its traffic flow id to the host queue
- If traffic flow is not identified
    - Non-fragmented: normal lookup operation

- Fragments: allocate a new traffic flow, set its packet counter to 1 and forward the fragment with its traffic flow id to the host queue. If traffic flow is not available, just forward the fragment with "NA" traffic flow id to the host queue

Second Pass: (traffics from the host)

- Decrement the packet counter by the specific count if the traffic flow id is specified. Free the traffic flow if its counter reaches 0
- Normal lookup operation

7 Sample Code

This section describes the interfaces and functionalities of the sample code for the following features.

- IP Reassembly

7.1 IP Reassembly

7.1.1 Description

The host IP reassembly module should interact with the PASS and perform the full IP reassembly operation. The module user may choose to implement a simplified version of IP reassembly algorithm to save CPU cycle in controlled IP environment.

The sample code shall implement a simplified version of IP reassembly algorithm which supports non-overlapping segments only. The sample should perform the following tasks:

- Maintain the IP reassembly contexts consist of source IP, destination IP, IP identification, protocol, fragments count and the corresponding traffic flow id.
- Forward the non-fragmented IP packet with its flow id and count = 1 to PA PDSP queue. This avoids reordering the non-fragmented packets.
- For IPSEC inner IP fragments, call SA LLD to perform the post-decryption operation including padding check and IPSEC header and authentication tag removal
- Forward the reassembled IP packet with its flow id and fragments count to PA PDSP queue
- Send a null packet with its flow id and fragments count to PA PDSP queue if the fragments are discarded due to timeout or other error.

7.1.2 Data Structures

paIpReassmCfg_t

This structure contains the configurable parameters of the IP Reassembly module

Name	Description
timeout	IP reassembly timeout value
queueIn1	Input queue for outer IP reassembly
queueIn2	Input queue for inner IP reassembly
queueOut1	PA PDSP queue for outer IP reassembly
queueOut2	PA PDSP queue for inner IP reassembly

8 Multi Core Considerations

The PA LLD will work so that multiple instances of the driver can operate simultaneously on different cores. When the LLD on one core creates a handle, that handle can be passed to another core and used to add new handles which are linked to the ones created on another core.

When deleting handles there is no upward percolation of deletes. For example, core 0 creates handle X for a MAC address, and handle Y for an IP address that uses X. core 1 creates handle Z for an IP address that uses handle X. If core 0 then deletes handle X and Y, handle Z still remains. If core 1 tries to create handle A for a UDP port, the creation will fail because of a stale handle. But if handle Z had been activated it remains active.

9 Design

9.1 Internal Structure Definitions

The main function of the driver is to associate handles, and pass configuration to the PASS. Internally the data memory required by the LLD is used to store handle information.

9.1.1 Handle pointers

There are three types of handles maintained in the driver. L2, L3 and L4 handles. L2 and L3 handles are structures which define the fields relevant to the handles. L4 handles are different. To reduce memory usage the L4 handle information is stored in the handle itself. The 64-bit L4 handle should be maintained by the module user.

9.1.2 MAC/SRIO/IP common handle fields

The following fields are common to MAC, SRIO and IP handles

Name	Description
Type	Identifies the handle type
State	The state is either inactive, active, or pending PASS result
PDSP num	Identifies the PDSP that holds this entry in its LUT
LUT1 entry	The LUT1 index that holds the lookup info

9.1.3 MAC handle

The MAC handle contains the following information. Values which are 0 are considered “don’t cares” for the search criteria.

Name	Description
Src	MAC source address
Dest	MAC destination address
Vtag	The VLAN tag. For nested tags this is the inner tag
Ethertype	The ethertype
MPLS tag	The outer most MPLS tag.
inport	The input EMAC port

9.1.4 SRIO handle

The SRIO handle contains the following information. Values which are 0xFFFF are considered "don't cares" for the search criteria.

Name	Description
srcId	The 16-bit source ID
destId	The 16-bit destination ID
tt	The transport type
cc	The completion code
msgType	Type 9 or Type 11
cos	The type 9 class of service
streamId	The type 9 stream ID
letter	The type 11 letter
mbox	The type 11 mailbox

9.1.5 IP handle

The IP handle contains the following information. Values which are 0, except for TOS are considered don't cares for the search criteria.

Name	Description
Src	The source IP address (sized for IPv6)
Dest	The destination IP address
Protocol	The IPv4 protocol / IPv6 next header
TOS	The IPv4 TOS / IPv6 Traffic class
GRE protocol	The GRE protocol
SPI	The ESP/AH SPI value
sctpPort	The SCTP destination port

9.2 CorePac Software Resource Requirements

9.2.1 LLD MCPS Requirements

The will only be MCPS requirements if each transmit packet in a channel can vary in size. All other operations of the LLD are done only when a configuration change is required.

9.2.2 LLD Memory Requirements

9.2.2.1 Per System/Per IP block Instance Memory

- L2 Table
- L3 Table
- User-defined statistics Link Table if used
- LLD Instance

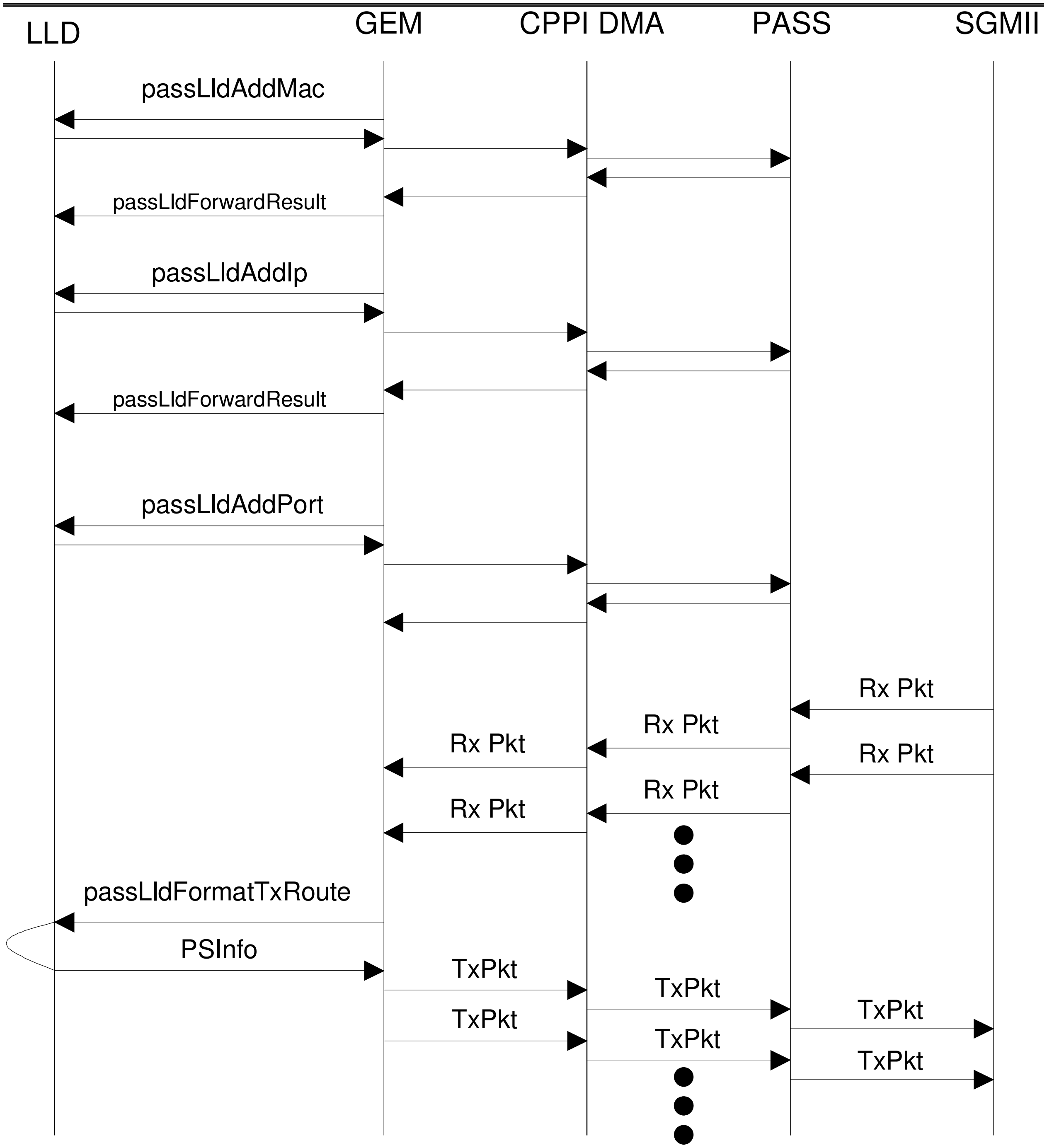
Note that there is no level 4 table. The TCP/UDP/GTPU port/tunnel id value must be embedded in the handle.

9.2.2.2 Per Channel/Connection/Application Link Memory Requirements

The application must maintain a handle for each entry created.

9.3 Subsystem Integration Call Flow

The figure illustrates the call flow for setting up a receive operation, and a transmit operation where the packet sizes do not change.



9.3.1 Subsystem Control/Initialization

Initialization is done by using the APIs or macros to put the subsystem into reset, download all required images, and release reset. Reset release can be verified with the get reset macro.

9.3.2 Data Flow

Voice path data does not flow through the LLD. Packets sent to the network are provided in CPPI format to the PKTDMA via the QM, and received on specified receive queues.

10 Testing Considerations

It must be possible for one core to create a connection, and have another core use one or more of the created handles to either build on the connection or tear it down.

Handling of default sub-module assignments must be tested. Connections that are configured as nested should have the inner connections assigned to the later classify PDSPs.