

# KeyStone Training

Network Coprocessor (NETCP)

Packet Accelerator (PA)

# Agenda

- Applications
- Hardware Modules
- Firmware
- PA Low Level Driver (LLD)
- Programming Example

# Packet Accelerator: Applications

- Applications
- Hardware Modules
- Firmware
- PA Low Level Driver (LLD)
- Programming Example

# Packet Accelerator Applications

- Packet accelerator saves cycles from host DSP cores.
- Option of single IP address for multi-core device
  - Multicore device internal architecture is abstracted
- 8192 multiple-in, multiple-out HW queues
- UDP (and TCP) Checksum and selected CRCs also for proprietary header formats
  - Verification on ingress and generation on egress
- L2 Support
  - Ethernet: Ethertype and VLAN
  - MPLS
- L3/L4 Support
  - IPv4/6 and UDP port based routing
  - Raw Ethernet or IPv4/6 and Proprietary UDP like protocol support
  - GRE
- QOS
  - Per channel / flow to individual queue towards host DSPs
  - TX traffic shaping
- Access to the Security Accelerator
  - IPSec ESP and AH tunnel, SRTP
- Multicast to multiple queues
  - For example, Ethernet broadcast copied and pushed to 1-8 queues
- Timestamps
  - IEEE 1588 timestamps and configurable generic timestamps

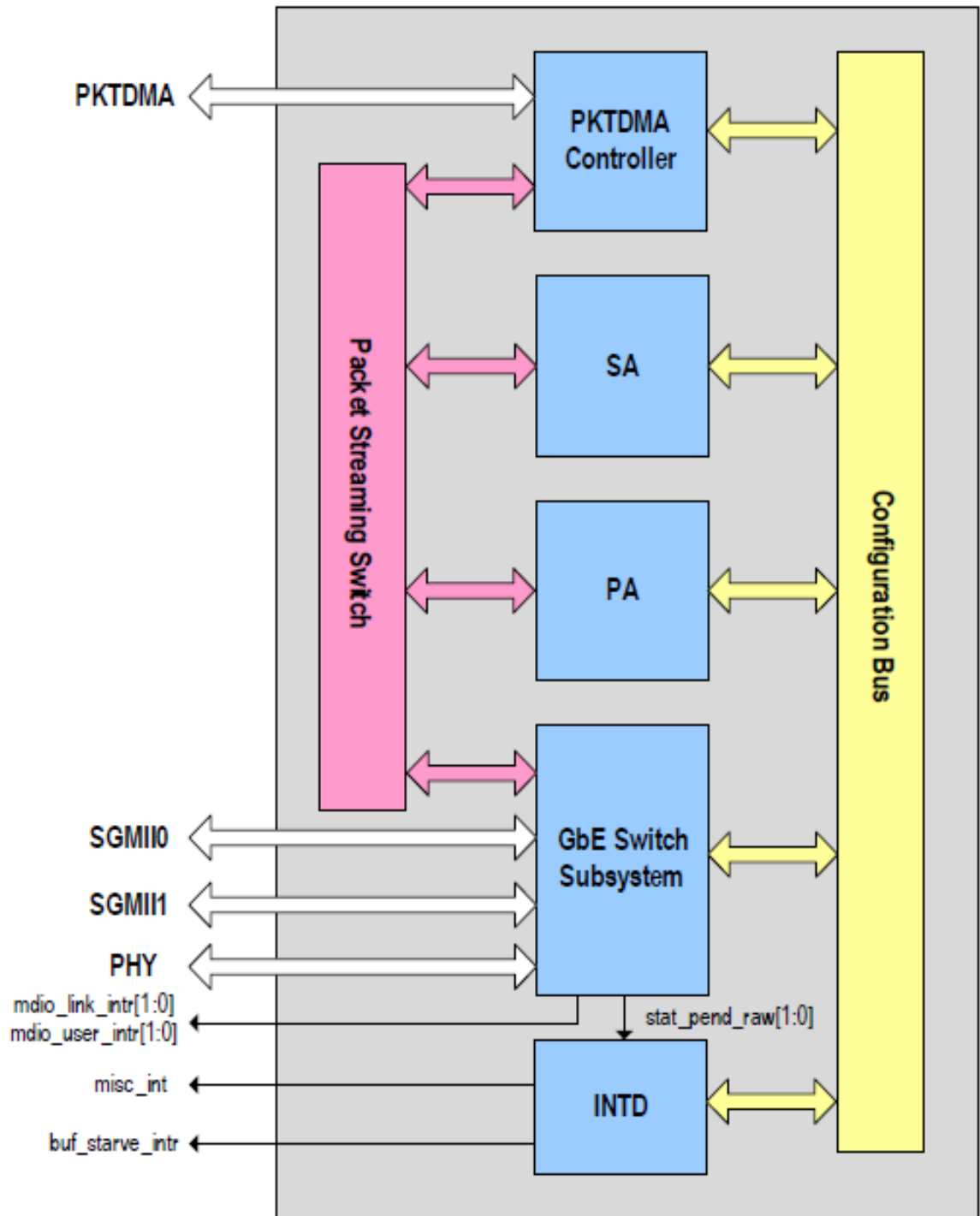
# Packet Accelerator: Applications

- Applications
- Hardware Modules
- Firmware
- PA Low Level Driver (LLD)
- Programming Example

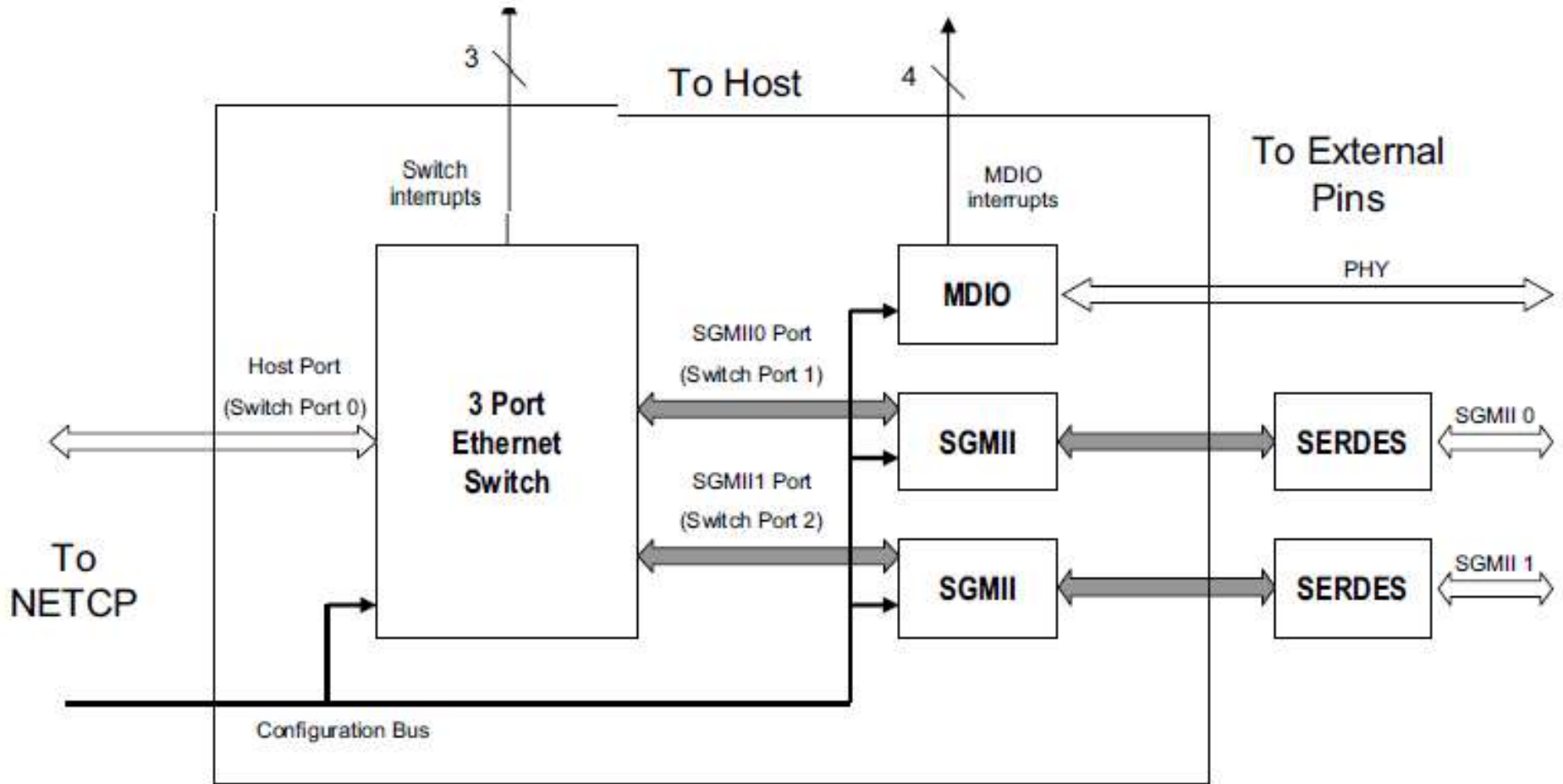
# Network Coprocessor (NETCP)

## Packet Accelerator (PA)

- Support for single or multiple IP addresses
- 1 Gbps wire-speed throughput at 1.5 Mpps
- UDP Checksum processing
- IPSec ESP and AH tunnels with fast path fully offloaded
- L2 support: Ethernet, Ethertype, and VLAN
- L3/L4 Support: IPv4/IPv6 and UDP port-based raw Ethernet or IPv4/6 and SCTP port-based routing
- Multicast to multiple queues
- QoS capability: Per channel/flow to individual queue towards DSP cores and support for TX traffic shaping per device



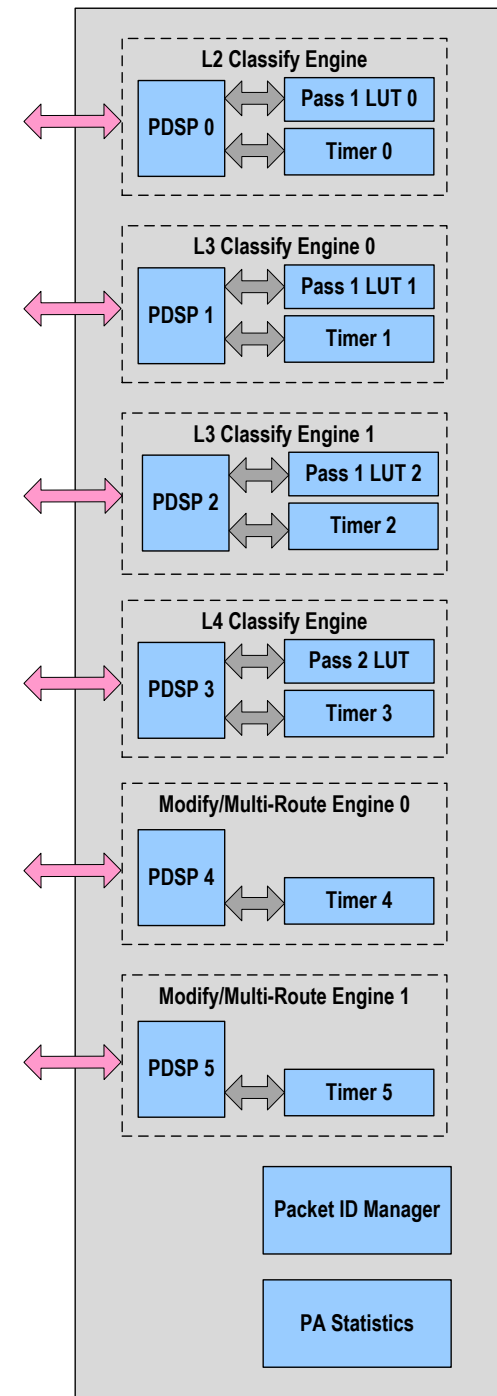
# SGMII and Ethernet Switch



# PA: High-Level Overview

- L2 Classify Engine
  - Used for matching L2 headers
  - Example headers: MAC, VLAN, LLC snap
- L3 Classify Engine 0
  - Used for matching L3 headers
  - Example headers: IPv4, IPv6, Custom L3
  - Also uses Multicore Navigator to match ESP headers and direct packets to SA
- L3 Classify Engine 1
  - Typically used for matching L3 headers in IPsec tunnels
  - Example headers: IPv4, IPv6, Custom L3
- L4 Classify Engine
  - Used for matching L4 Headers
  - Example headers: UDP, TCP, Custom L4
- Modify/Multi-Route Engines
  - Used for Modification, Multi-route, and Statistics requests
  - Modification Example: generate IP or UDP header checksums
  - Multi-route Example: route a packet to multiple queues
- PA Statistics Block
  - Stores statistics for packets processed by the classify engines
  - Statistics requests typically handled by Modify/Multi-route engines
- Packet ID Manager
  - Assigns packet ID to packets

Connections  
to Packet  
Streaming  
Switch



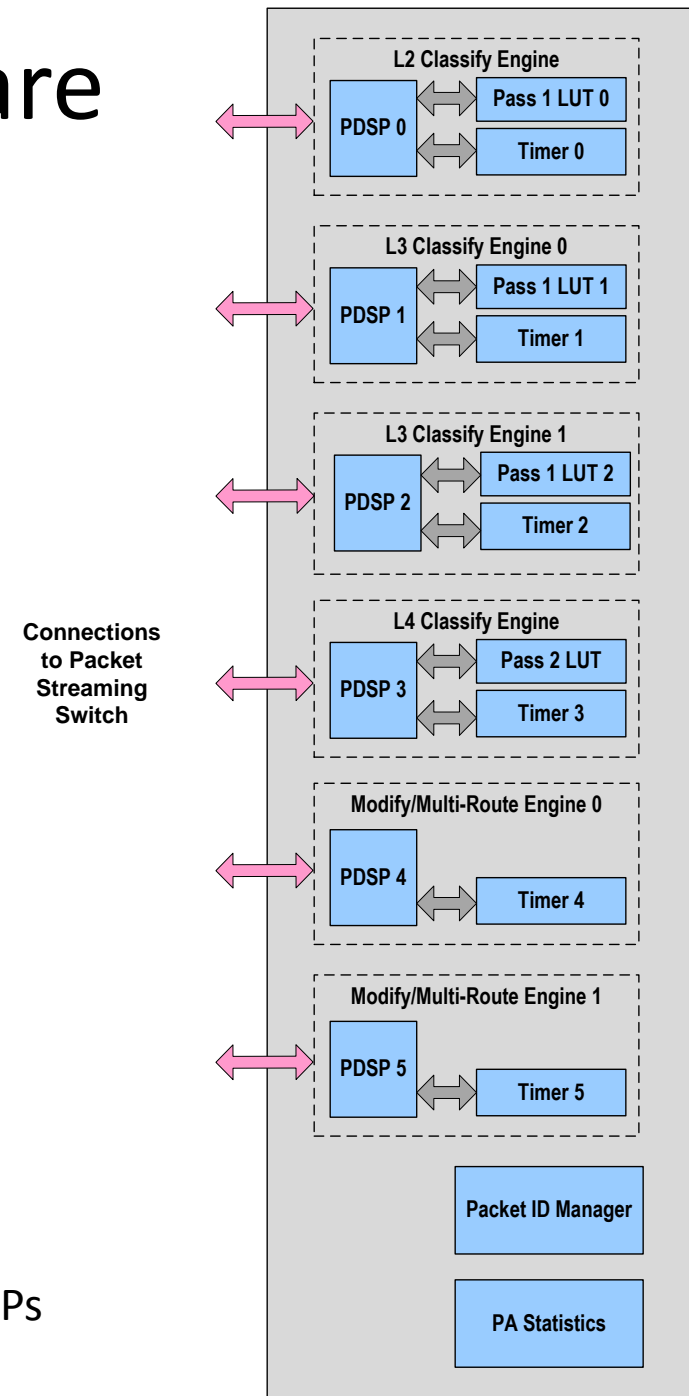


# Packet Accelerator: Firmware

- Applications
- Hardware Modules
- Firmware
- PA Low Level Driver (LLD)
- Programming Example

# PA: Hardware and Firmware

- One L2 Classify Engine
  - PDSP
  - Pass 1 Lookup Table (LUT1)
  - Timer
  - Classify 1 (c1) firmware image
- Two L3 Classify Engines
  - PDSP
  - Pass 1 Lookup Table (LUT1)
  - Timer
  - Classify 1 (c1) firmware image
- One L4 Classify Engine
  - PDSP
  - Pass 2 Lookup Table (LUT2)
  - Timer
  - Classify 1 (c1) firmware image
- Two Modify/Multi-Route Engines
  - PDSP
  - Timer
  - Modify (m) firmware image
- One Packet ID Manager
- One PA Statistics Block
- NOTE: Firmware images must be loaded onto the PDSPs before using the PA engines.



# PA Low Level Driver Overview

- Applications
- Hardware Modules
- Firmware
- PA Low Level Driver (LLD)
- Programming Example

# PA LLD Overview

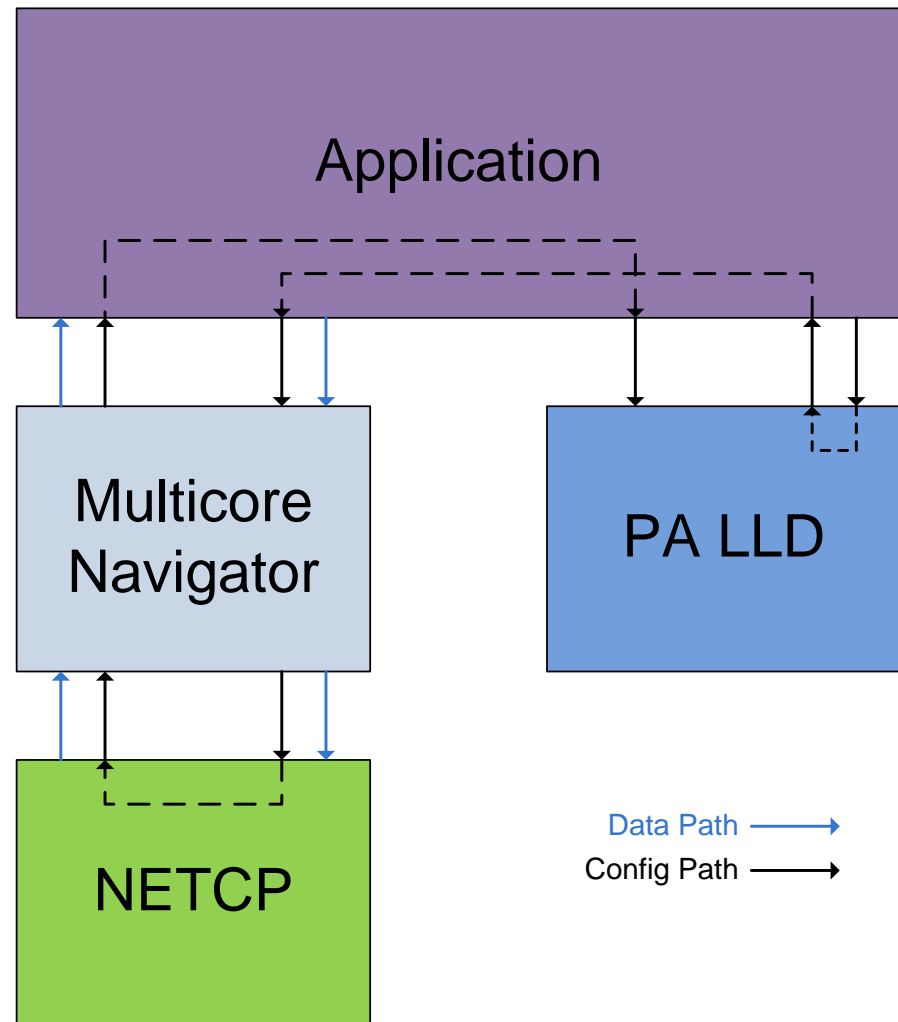
- PA LLD provides an abstraction layer between the application and the PA. It translates packet headers and routing requirements into configuration information that is used by the PA firmware.
- PA LLD provides the command/response interface for PA configurations:
  - LUT1
  - LUT2
  - CRC generation
  - Multi-route

NOTE: The most general configuration must be entered into the PDSPs before any overlapping, more specific configuration

- The PA LLD also handles linking together entries in separate lookup tables. For example, linking an entry in an L2 classify lookup table to an entry in an L3 classify lookup table.
- PA LLD does not provide transport layer; This is handled by the Multicore Navigator.
- API calls are non-blocking.
- PA LLD references:
  - **`pa/docs/paDocs.chm`**
  - **`pa/docs/pa_sds.pdf`**

# PA LLD Functional Diagram

- Benefits
  - Abstracts the operation of the PA from the application
  - OS-independent
  - Multi-instance for multicore
- NOTE:
  - PA LLD runs on the host DSP and is external in the PA.



# PA Low Level Drivers (PA LLD)

- Applications
- Hardware Modules
- Firmware
- PA Low Level Driver (LLD)
- Programming Example

# PA LLD API: System

[paReturn\\_t](#)    [Pa\\_getBufferReq](#) ([paSizeInfo\\_t](#) \*sizeCfg, int sizes[], int aligns[])

*Pa\_getBufferReq returns the memory requirements for the PA driver.*

[paReturn\\_t](#)    [Pa\\_create](#) ([paConfig\\_t](#) \*cfg, void \*bases[], [Pa\\_Handle](#) \*pHandle)

*Pa\_create creates the PA driver instance.*

[paReturn\\_t](#)    [Pa\\_close](#) ([Pa\\_Handle](#) handle, void \*bases[])

*Pa\_close deactivates the PA driver instance.*

[paReturn\\_t](#)    [Pa\\_requestStats](#) ([Pa\\_Handle](#) iHandle, uint16\_t doClear, [paCmd\\_t](#) cmd, uint16\_t \*cmdSize, [paCmdReply\\_t](#) \*reply, int \*cmdDest)

*Pa\_requestStats requests sub-system statistics.*

[paSysStats\\_t](#) \*    [Pa\\_formatStatsReply](#) ([Pa\\_Handle](#) handle, [paCmd\\_t](#) cmd)

*Pa\_formatStatsReply formats a stats request from the PA.*

[paSSstate\\_t](#)    [Pa\\_resetControl](#) ([paSSstate\\_t](#) newState)

*Pa\_resetControl controls the reset state of the Sub-system.*

[paReturn\\_t](#)    [Pa\\_downloadImage](#) (int modId, void \*image, int sizeBytes)

*Pa\_downloadImage downloads a PDSP image to a sub-system with the packet processing modules in reset.*

# PA LLD API: Configuration

**[paReturn\\_t](#)** **[Pa\\_addMac](#)** (**[Pa\\_Handle](#)** iHandle, **[paEthInfo\\_t](#)** \*ethInfo, **[paRouteInfo\\_t](#)** \*routeInfo, **[paRouteInfo\\_t](#)** \*nextRtFail, **[paHandleL2L3\\_t](#)** \*handle, **[paCmd\\_t](#)** cmd, uint16\_t \*cmdSize, **[paCmdReply\\_t](#)** \*reply, int \*cmdDest)

*Pa\_addMac adds a mac address to the L2 table.*

**[paReturn\\_t](#)** **[Pa\\_delHandle](#)** (**[Pa\\_Handle](#)** iHandle, **[paHandleL2L3\\_t](#)** handle, **[paCmd\\_t](#)** cmd, uint16\_t \*cmdSize, **[paCmdReply\\_t](#)** \*reply, int \*cmdDest)

*Pa\_delHandle deletes a MAC or IP handle.*

**[paReturn\\_t](#)** **[Pa\\_delL4Handle](#)** (**[Pa\\_Handle](#)** iHandle, **[paHandleL4\\_t](#)** handle, **[paCmd\\_t](#)** cmd, uint16\_t \*cmdSize, **[paCmdReply\\_t](#)** \*reply, int \*cmdDest)

*Pa\_delL4Handle deletes a TCP or UDP handle.*

**[paReturn\\_t](#)** **[Pa\\_addIp](#)** (**[Pa\\_Handle](#)** iHandle, **[paIpInfo\\_t](#)** \*ipInfo, **[paHandleL2L3\\_t](#)** prevLink, **[paRouteInfo\\_t](#)** \*routeInfo, **[paRouteInfo\\_t](#)** \*nextRtFail, **[paHandleL2L3\\_t](#)** \*retHandle, **[paCmd\\_t](#)** cmd, uint16\_t \*cmdSize, **[paCmdReply\\_t](#)** \*reply, int \*cmdDest)

*Pa\_addIp adds an IP address to the L3 table.*

**[paReturn\\_t](#)** **[Pa\\_addPort](#)** (**[Pa\\_Handle](#)** iHandle, uint16\_t destPort, **[paHandleL2L3\\_t](#)** linkHandle, **[paRouteInfo\\_t](#)** \*routeInfo, **[paHandleL4\\_t](#)** retHandle, **[paCmd\\_t](#)** cmd, uint16\_t \*cmdSize, **[paCmdReply\\_t](#)** \*reply, int \*cmdDest)

*Pa\_addPort adds a destination TCP/UDP port to the L4 table.*

**[paReturn\\_t](#)** **[Pa\\_forwardResult](#)** (**[Pa\\_Handle](#)** iHandle, void \*vresult, **[paHandle\\_t](#)** \*retHandle, int \*handleType, int \*cmdDest)

*Pa\_forwardResult examines the reply of the sub-system to a command.*

**[paReturn\\_t](#)** **[Pa\\_configRouteErrPacket](#)** (**[Pa\\_Handle](#)** iHandle, int nRoute, int \*errorTypes, **[paRouteInfo\\_t](#)** \*eRoutes, **[paCmd\\_t](#)** cmd, uint16\_t \*cmdSize, **[paCmdReply\\_t](#)** \*reply, int \*cmdDest)

*Pa\_configRouteErrPacket configures the routing of packets that match error conditions.*



# PA LLD API: Custom Configuration

[paReturn\\_t](#) [Pa\\_setCustomL3](#) ([Pa\\_Handle](#) iHandle, uint16\_t customEthertype, uint16\_t parseByteOffset, uint8\_t byteMasks[pa\_NUM\_BYTES\_CUSTOM\_L3], [paCmd\\_t](#) cmd, uint16\_t \*cmdSize, [paCmdReply\\_t](#) \*reply, int \*cmdDest)

*Pa\_setCustomL3 performs the global configuration for level 3 custom lookups.*

[paReturn\\_t](#) [Pa\\_addCustomL3](#) ([Pa\\_Handle](#) iHandle, uint8\_t match[pa\_NUM\_BYTES\_CUSTOM\_L3], [paRouteInfo\\_t](#) \*routeInfo, [paRouteInfo\\_t](#) \*nextRtFail, [paHandleL2L3\\_t](#) prevLink, [paHandleL2L3\\_t](#) \*retHandle, int nextHdrType, uint16\_t nextOffset, [paCmd\\_t](#) cmd, uint16\_t \*cmdSize, [paCmdReply\\_t](#) \*reply, int \*cmdDest)

*Pa\_AddCustomL3 adds a custom lookup entry to the lookup tables.*

[paReturn\\_t](#) [Pa\\_setCustomL4](#) ([Pa\\_Handle](#) iHandle, uint16\_t handleLink, uint16\_t udpCustomPort, uint16\_t byteOffsets[pa\_NUM\_BYTES\_CUSTOM\_L4], uint8\_t byteMasks[pa\_NUM\_BYTES\_CUSTOM\_L4], [paCmd\\_t](#) cmd, uint16\_t \*cmdSize, [paCmdReply\\_t](#) \*reply, int \*cmdDest)

*Pa\_setCustomL4 performs the global configuration for level 4 custom lookups.*

[paReturn\\_t](#) [Pa\\_addCustomL4](#) ([Pa\\_Handle](#) iHandle, [paHandleL2L3\\_t](#) prevLink, uint8\_t match[pa\_NUM\_BYTES\_CUSTOM\_L4], [paRouteInfo\\_t](#) \*routeInfo, [paHandleL4\\_t](#) retHandle, [paCmd\\_t](#) cmd, uint16\_t \*cmdSize, [paCmdReply\\_t](#) \*reply, int \*cmdDest)

*Pa\_addCustomL4 adds a custom lookup to the lookup tables.*

# PA LLD API: Utility Functions

[paReturn\\_t](#)    [Pa\\_formatTxRoute](#) ([paTxChksum\\_t](#) \*chk0, [paTxChksum\\_t](#) \*chk1, [paRouteInfo\\_t](#) \*route, void \*cmdBuffer, int \*cmdSize)

*Pa\_formatTxRoute formats the commands to add checksums and route a Tx packet.*

[paReturn\\_t](#)    [Pa\\_formatRoutePatch](#) ([paRouteInfo\\_t](#) \*route, [paPatchInfo\\_t](#) \*patch, void \*cmdBuffer, int \*cmdSize)

*Pa\_formatRoutePatch formats the commands to route a packet and blind patch.*

# LLD HTML Documentation

Show example from the HTML file:

- Pa\_addMac
- Pa\_configExceptionRoute

# Download the Firmware

```
Int paDownloadFirmware (void)
{
    Int i;

    Pa_resetControl (paInst, pa_STATE_RESET);

    /* PDPSS 0-2 use image c1 */
    for (i = 0; i < 3; i++)
        Pa_downloadImage (paInst, i, (Ptr)c1, c1Size);

    /* PDSP 3 uses image c2 */
    Pa_downloadImage (paInst, 3, (Ptr)c2, c2Size);

    /* PDSPs 4-5 use image m */
    for (i = 4; i < 6; i++)
        Pa_downloadImage (paInst, i, (Ptr)m, mSize);

    Pa_resetControl (paInst, pa_STATE_ENABLE);

    return (0);
}
```

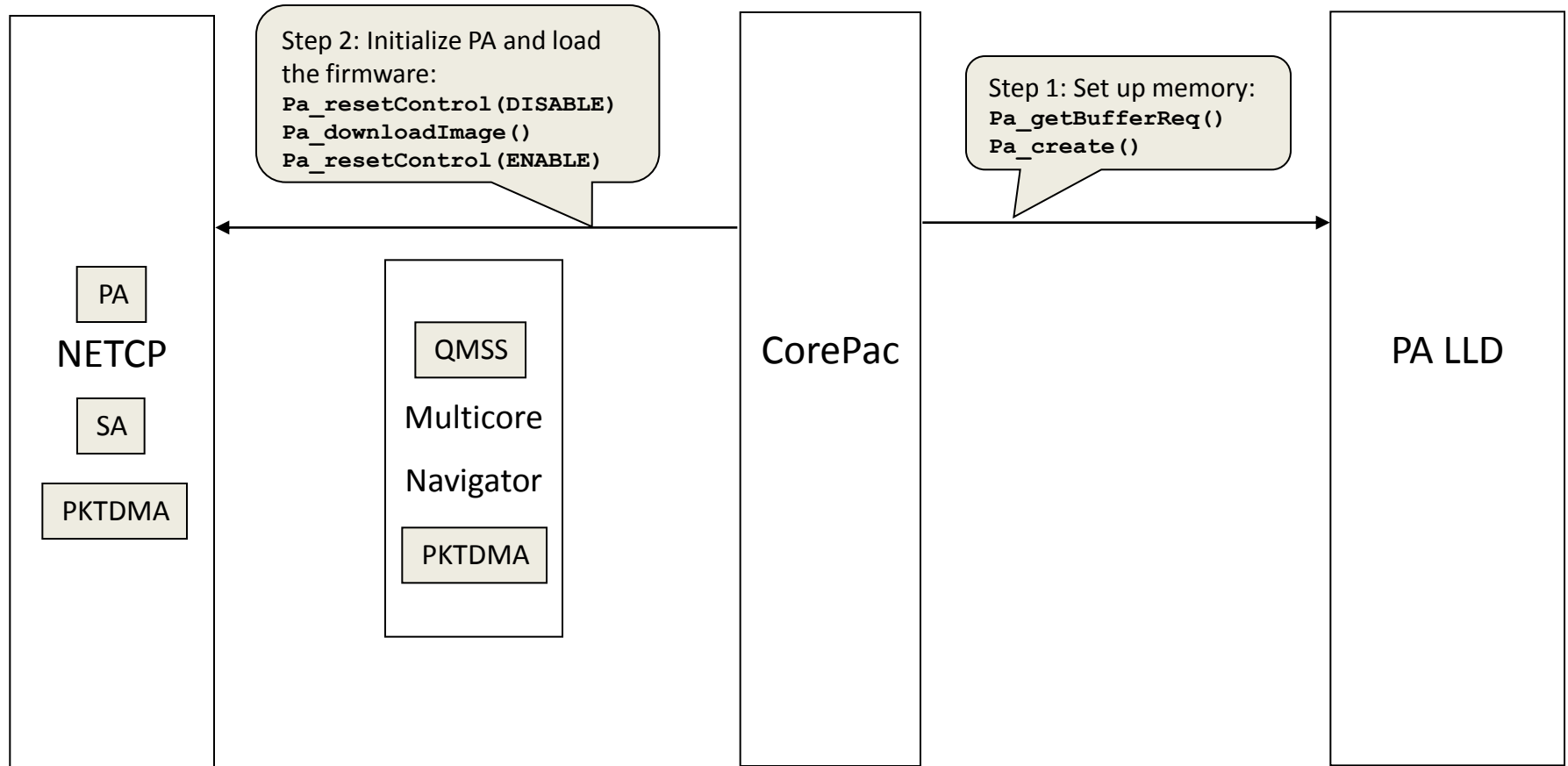
# PA LLD: Programming Example

- Applications
- Hardware Modules
- Firmware
- PA Low Level Driver (LLD)
- Programming Example

# PA LLD: Basic Configuration

This process must be done before configuring PA RX or TX.

Basic Configuration



# PA LLD: PA Routing

- PA LLD provides a routing structure which allows the following parameters to be configured:
  - Destination
  - Flow ID
  - Queue
  - Multi-Route Handle (Index)
  - Software Info 0
  - Software Info 1
- Several possible destinations
  - pa\_DEST\_HOST
  - pa\_DEST\_EMAC
  - pa\_DEST\_SASS0
  - pa\_DEST\_SASS1
  - pa\_DEST\_DISCARD
  - pa\_DEST\_CONTINUE\_PARSE

## MAC Routing Example:

```
paRouteInfo_t macRoute;  
  
/* Continue parsing -- try to match IP  
handle*/  
  
macRoute.dest = pa_DEST_CONTINUE_PARSE;  
macRoute.flowId = 0;  
macRoute.queue = 0;  
macRoute.mRouteHandle = -1;  
macRoute.swInfo0 = 0;      /* Don't Care */  
macRoute.swInfo1 = 0;      /* Don't Care */
```

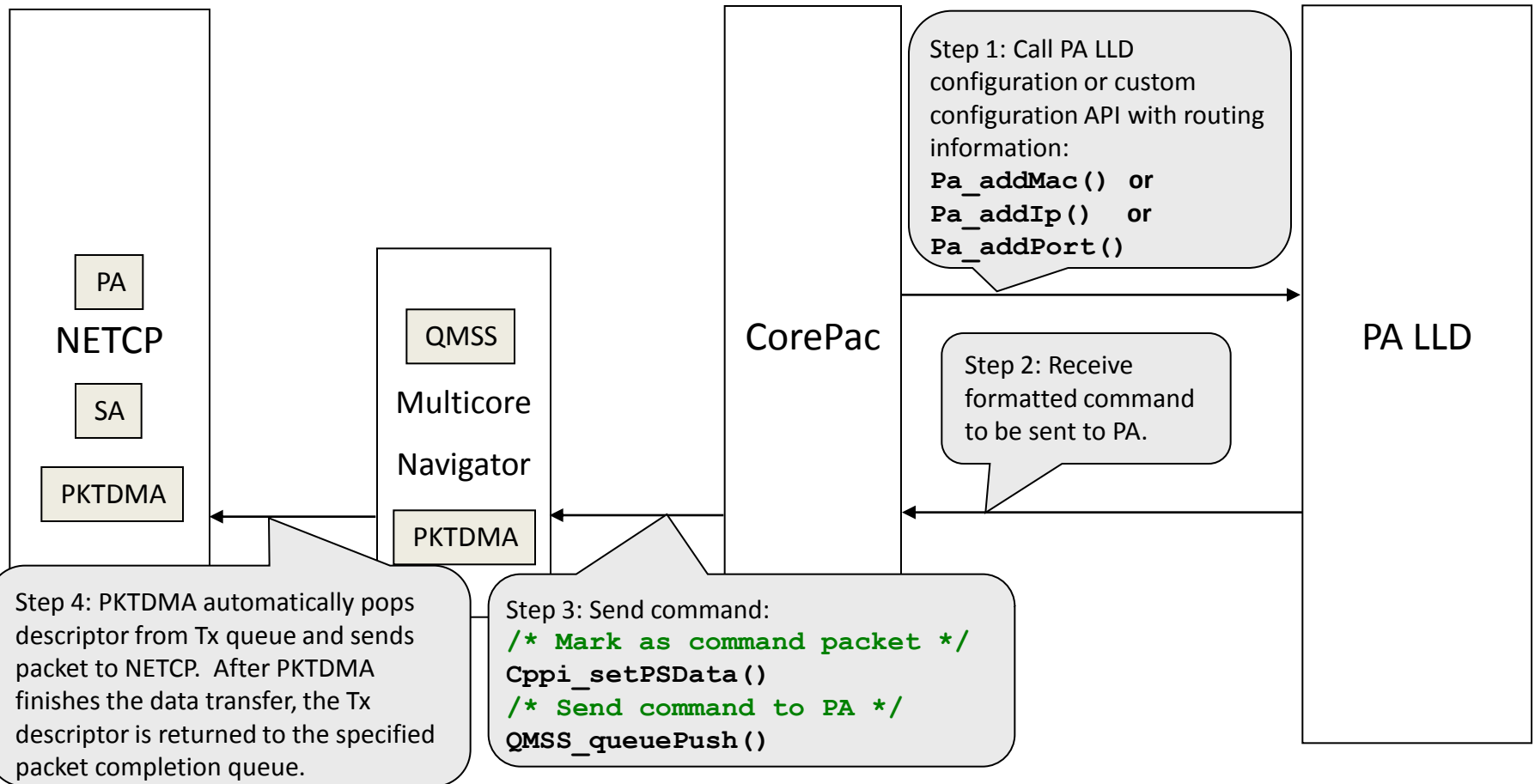
## Port Routing Example:

```
paRouteInfo_t portRoute;  
  
/* Send all matches to the queue specified  
*/  
  
portRoute.dest = pa_DEST_HOST;  
portRoute.flowId = 5;  
portRoute.queue = 900;  
portRoute.mRouteHandle = -1;  
portRoute.swInfo0 = 0;      /* Don't Care  
*/  
portRoute.swInfo1 = 0;      /* Don't Care  
*/
```

# PA LLD: Rx Configuration

Repeat steps 1-7 to add more MAC addresses, IP addresses, or TCP/UDP ports to PA LUTs before receiving data packets.

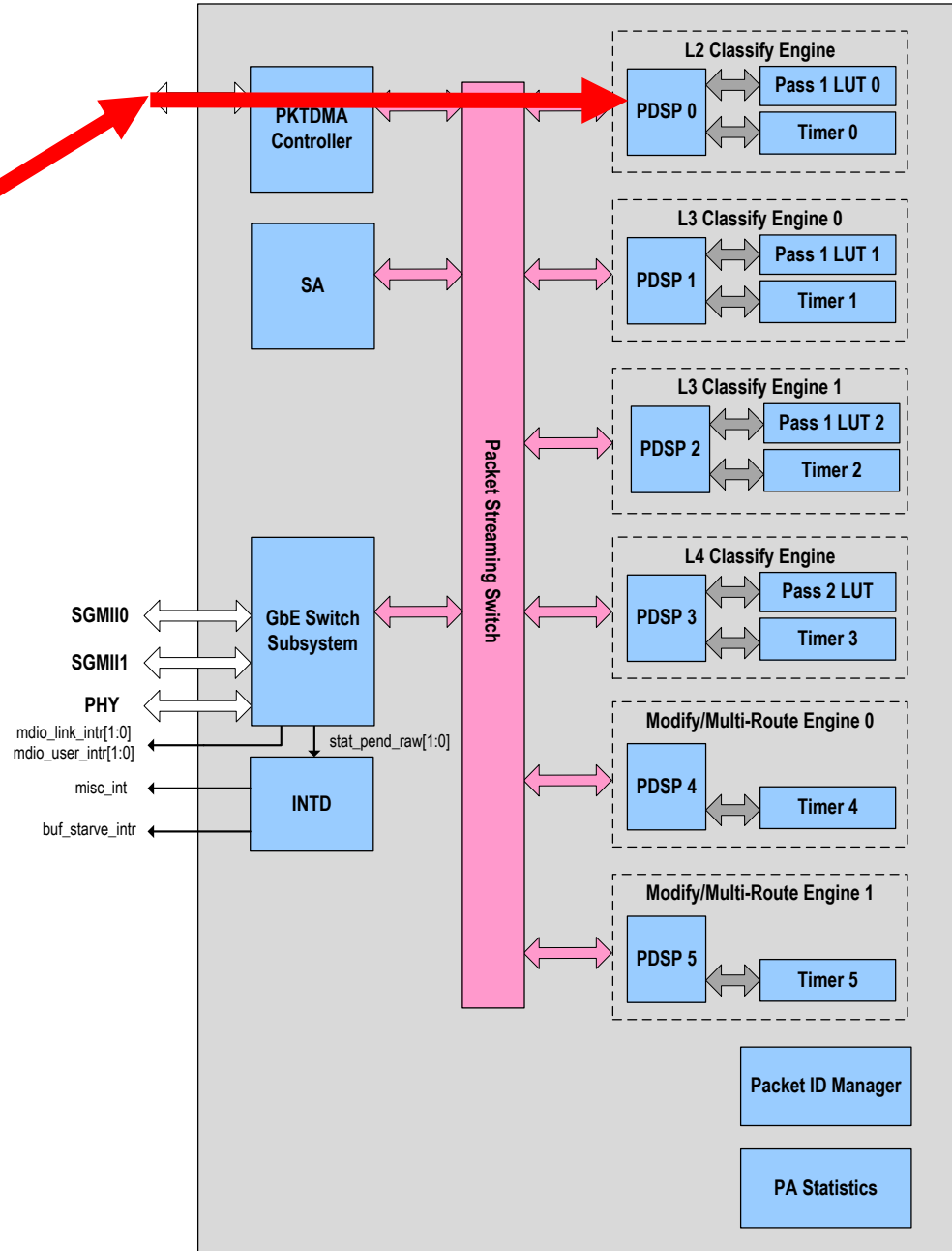
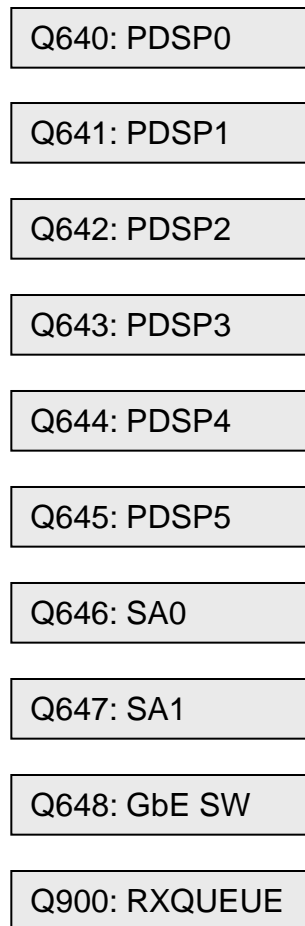
Configuration Information





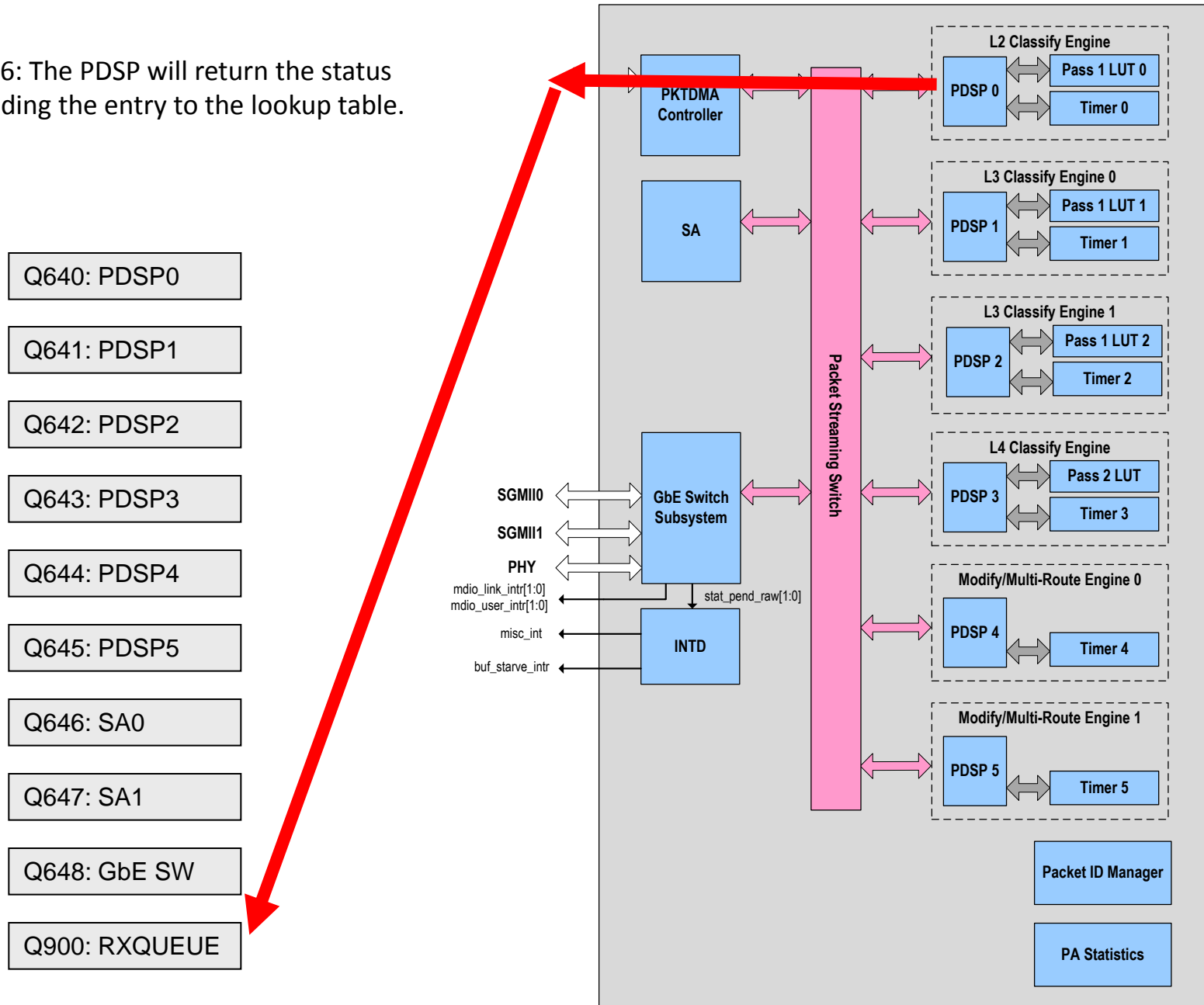
# PA LLD: Rx Configuration

Step 5: The PA LLD will indicate the appropriate transmit queue index. Using the queue returned from **Pa\_addMac()** in this example.



# PA LLD: Rx Configuration

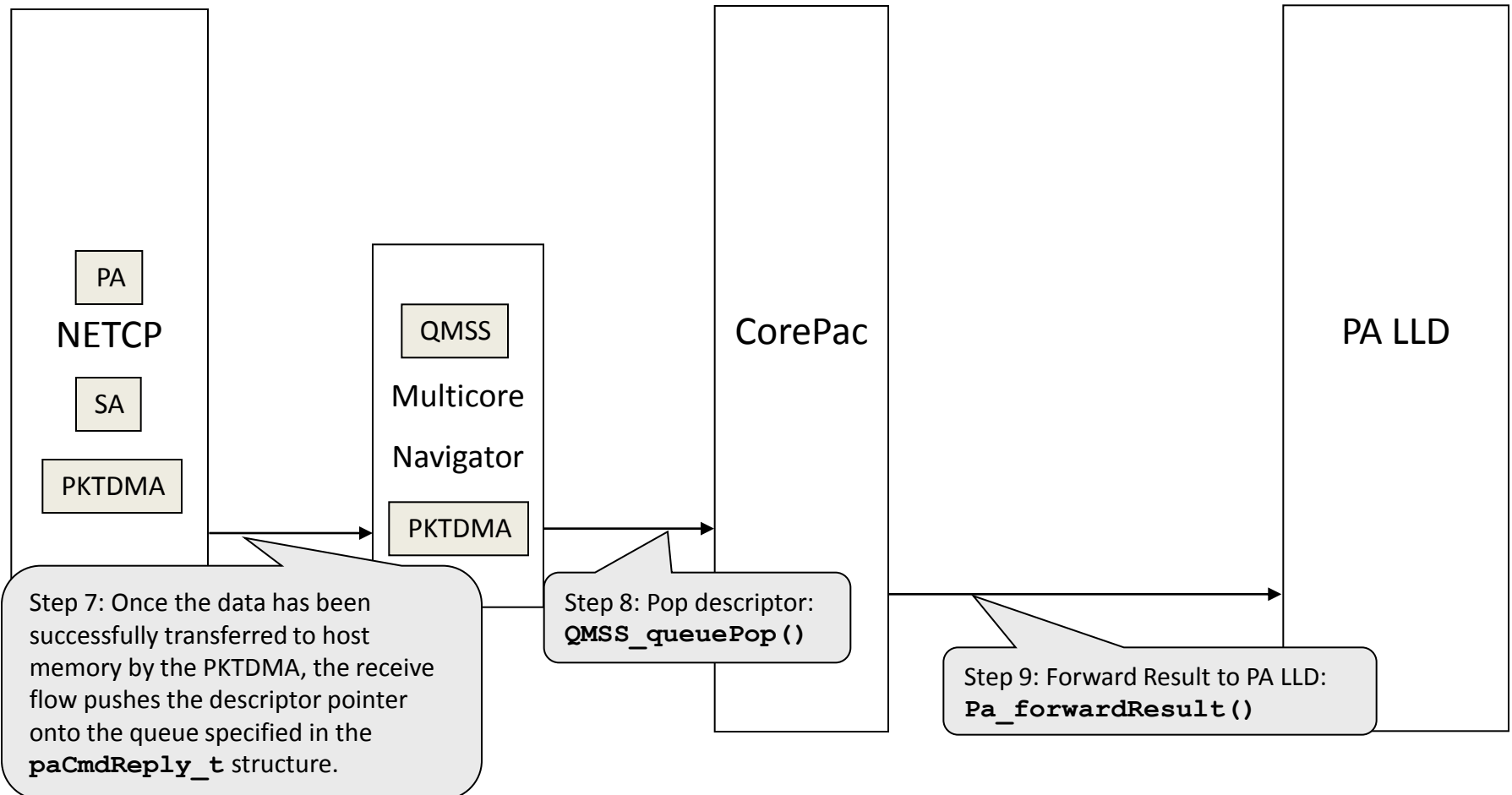
Step 6: The PDSP will return the status of adding the entry to the lookup table.



# PA LLD: Rx Configuration

Repeat steps 1-9 to add more MAC addresses, IP addresses, or TCP/UDP ports to PA LUTs before receiving data packets.

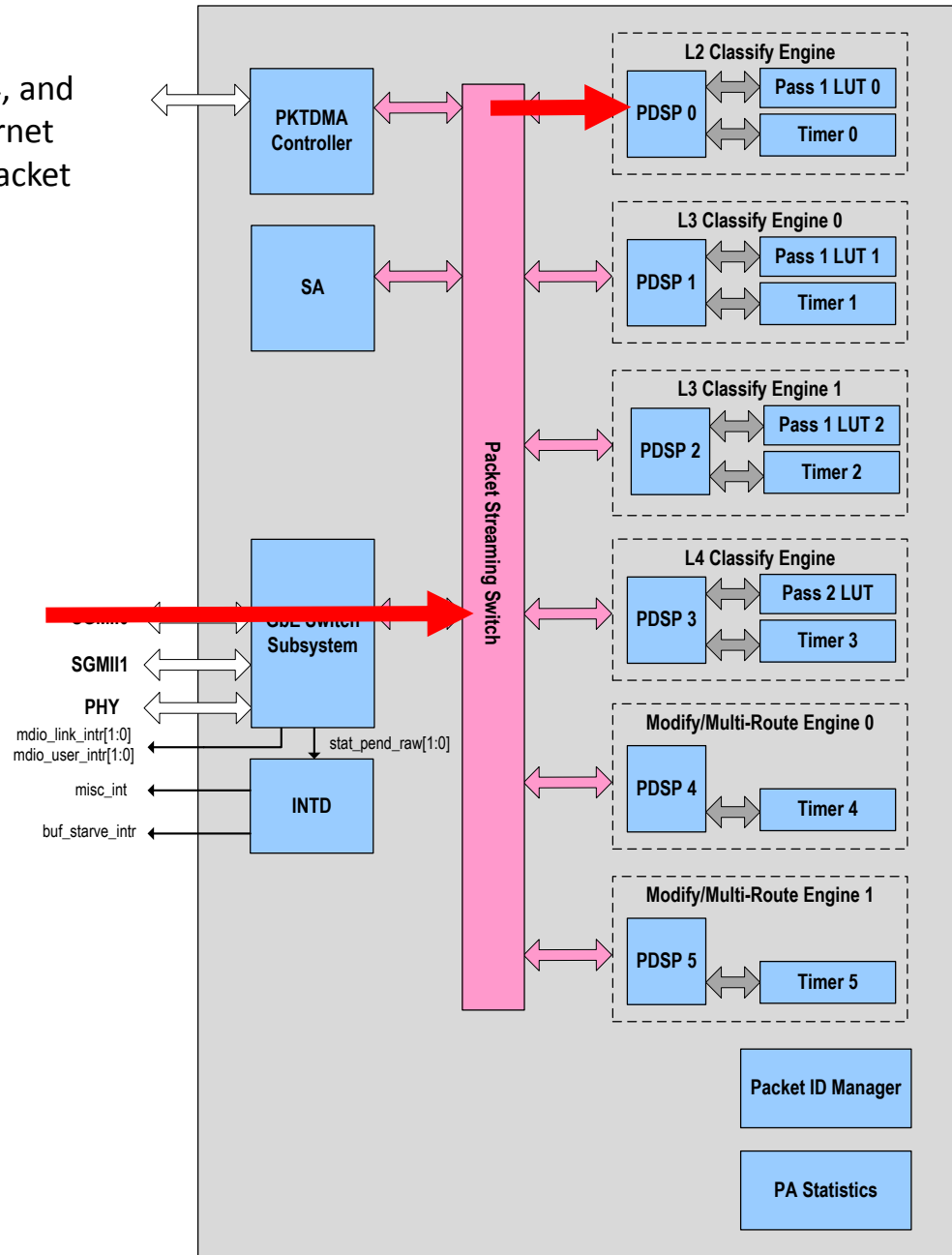
Configuration Information



# PA Rx Hardware Processing

Step 1: A packet formatted with MAC, IPv4, and UDP headers arrives from the gigabit Ethernet switch subsystem and is routed over the packet streaming switch to the L2 Classify Engine.

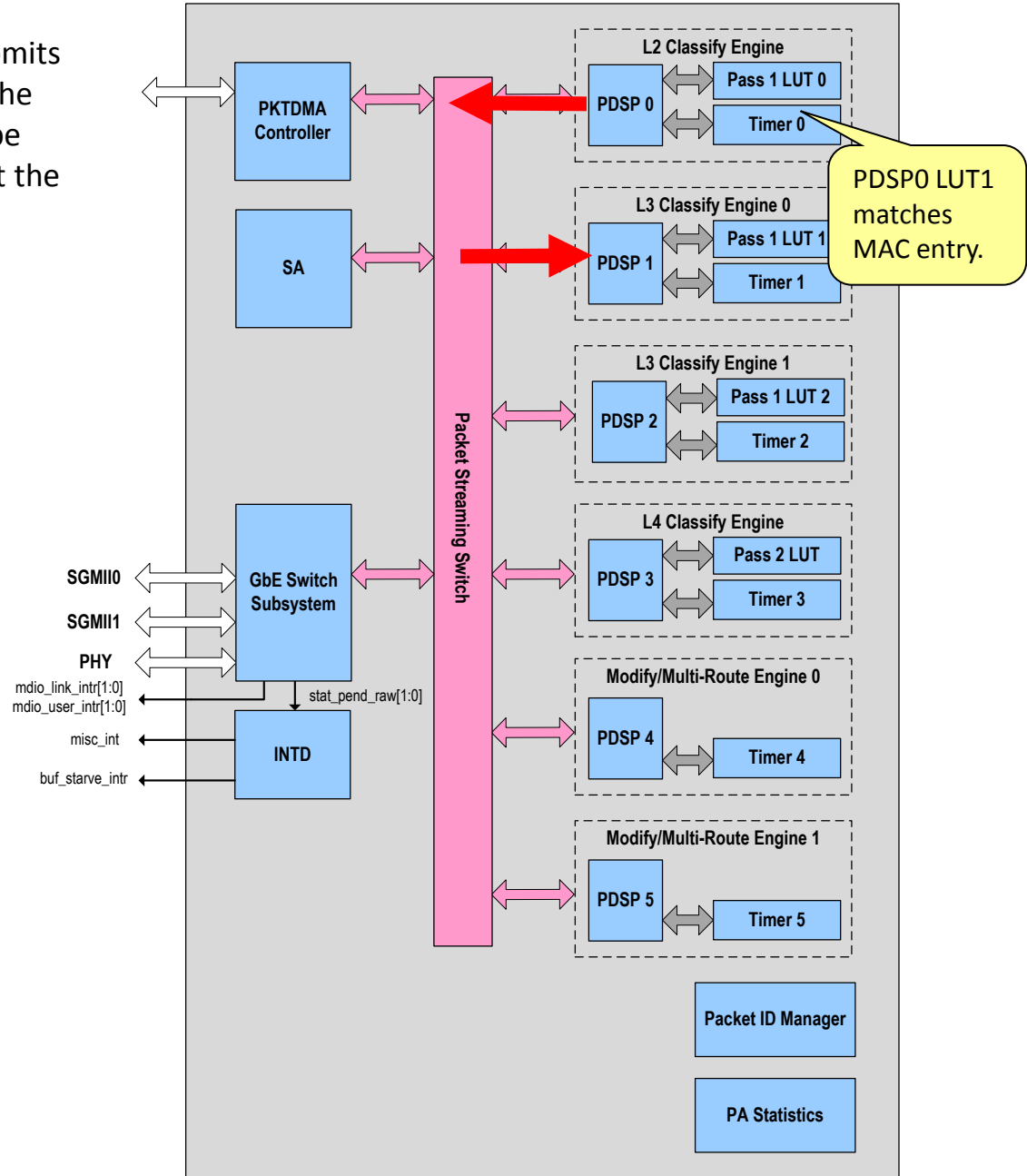
Q640: PDSP0
Q641: PDSP1
Q642: PDSP2
Q643: PDSP3
Q644: PDSP4
Q645: PDSP5
Q646: SA0
Q647: SA1
Q648: GbE SW
Q900: RXQUEUE



# PA Rx Hardware Processing

Step 2: PDSP0 in the L2 Classify Engine submits the MAC header for lookup. Assume that the lookup is successful. The packet will then be routed to its next destination. Assume that the destination is L3 Classify Engine 0.

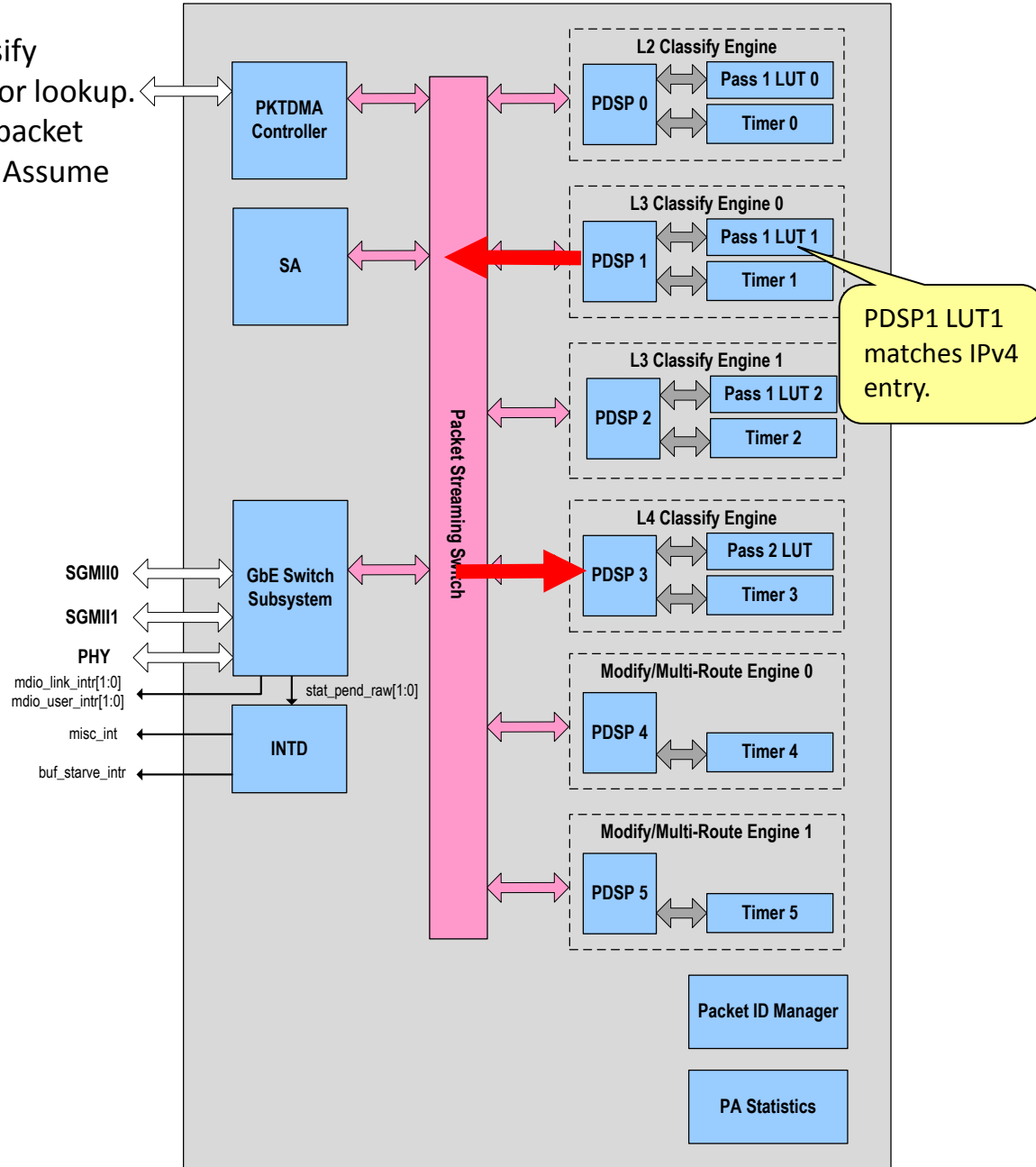
Q640: PDSP0
Q641: PDSP1
Q642: PDSP2
Q643: PDSP3
Q644: PDSP4
Q645: PDSP5
Q646: SA0
Q647: SA1
Q648: GbE SW
Q900: RXQUEUE



# PA Rx Hardware Processing

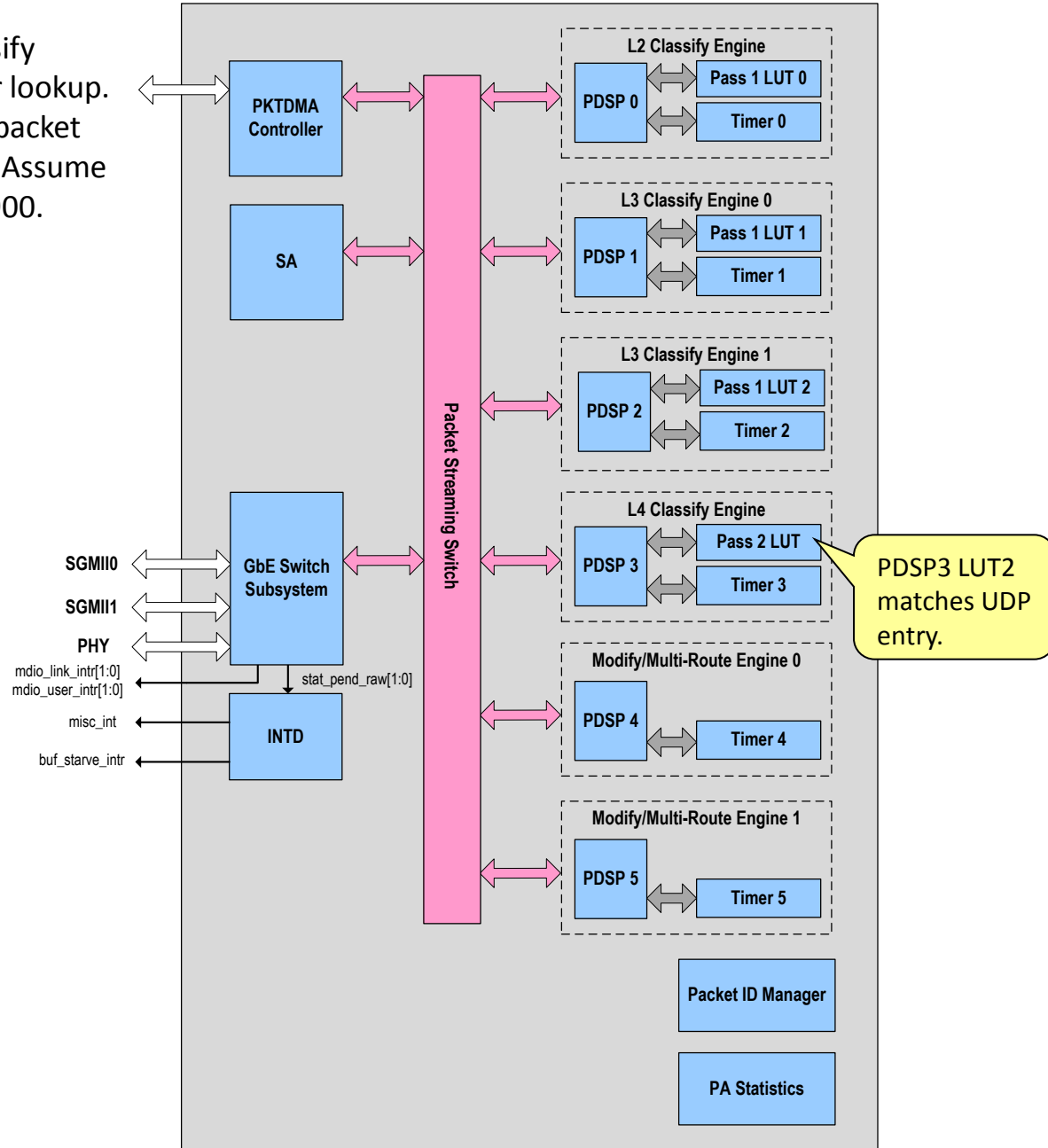
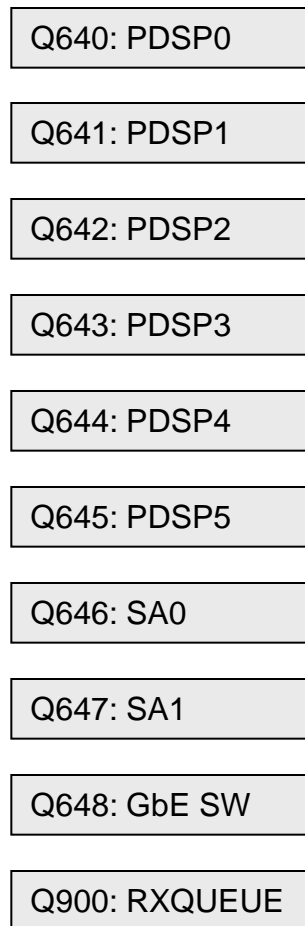
Step 3: The packet is routed to the L3 Classify Engine 0. PDSP1 submits the IPv4 header for lookup. Assume that the lookup is successful. The packet will then be routed to its next destination. Assume that it is the L4 Classify Engine.

Q640: PDSP0
Q641: PDSP1
Q642: PDSP2
Q643: PDSP3
Q644: PDSP4
Q645: PDSP5
Q646: SA0
Q647: SA1
Q648: GbE SW
Q900: RXQUEUE



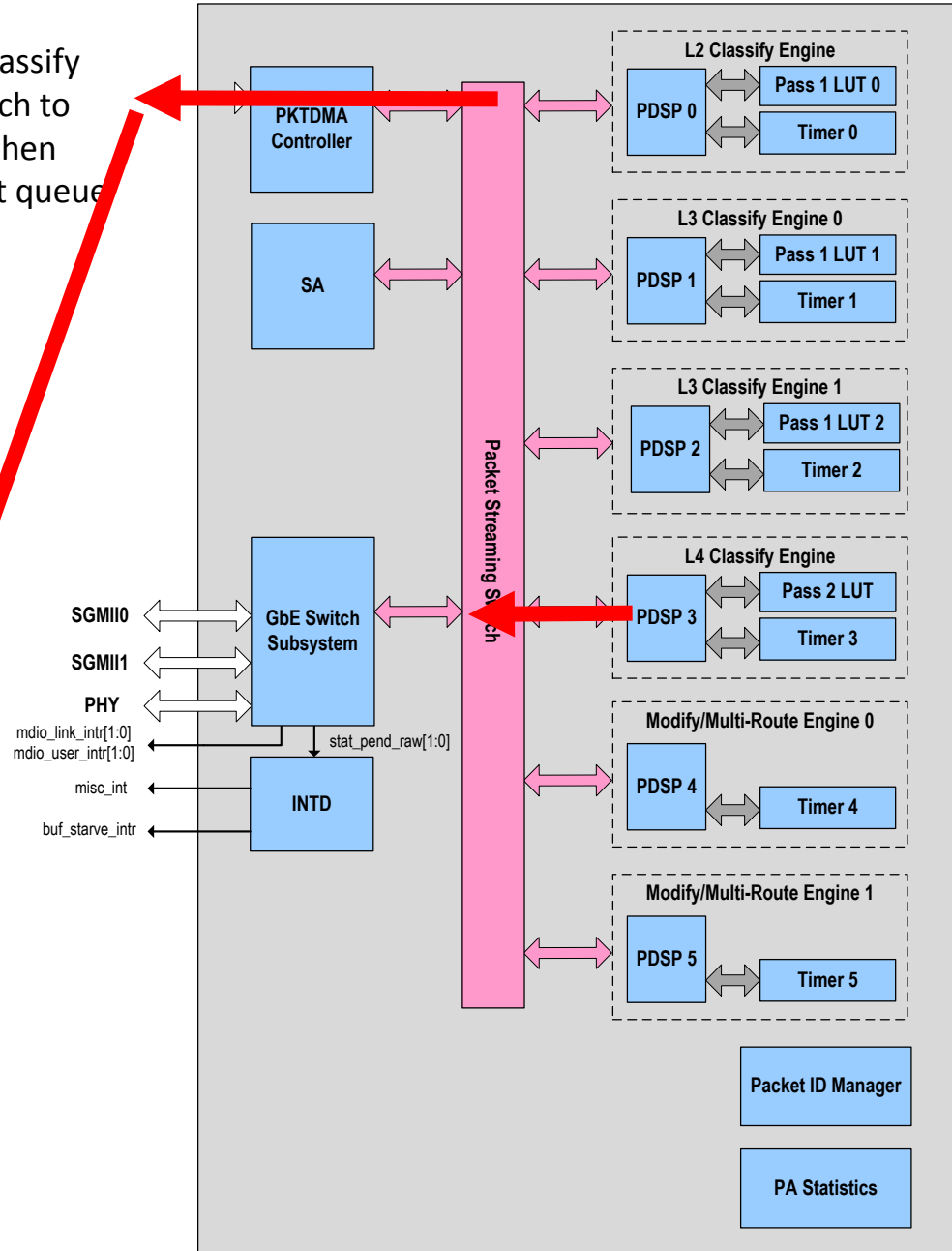
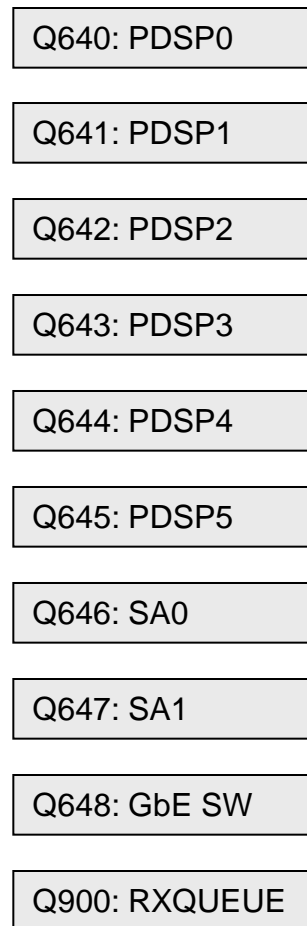
# PA Rx Hardware Processing

Step 4: The packet is routed to the L4 Classify Engine. PDSP3 submits the UDP header for lookup. Assume that the lookup is successful. The packet will then be routed to its next destination. Assume that the destination is the host on queue 900.



# PA Rx Hardware Processing

Step 5: The packet is routed from the L4 Classify Engine, through the packet streaming switch to the PKTDMA controller. The PKTDMA will then transfer the packet from the NETCP to host queue 900.

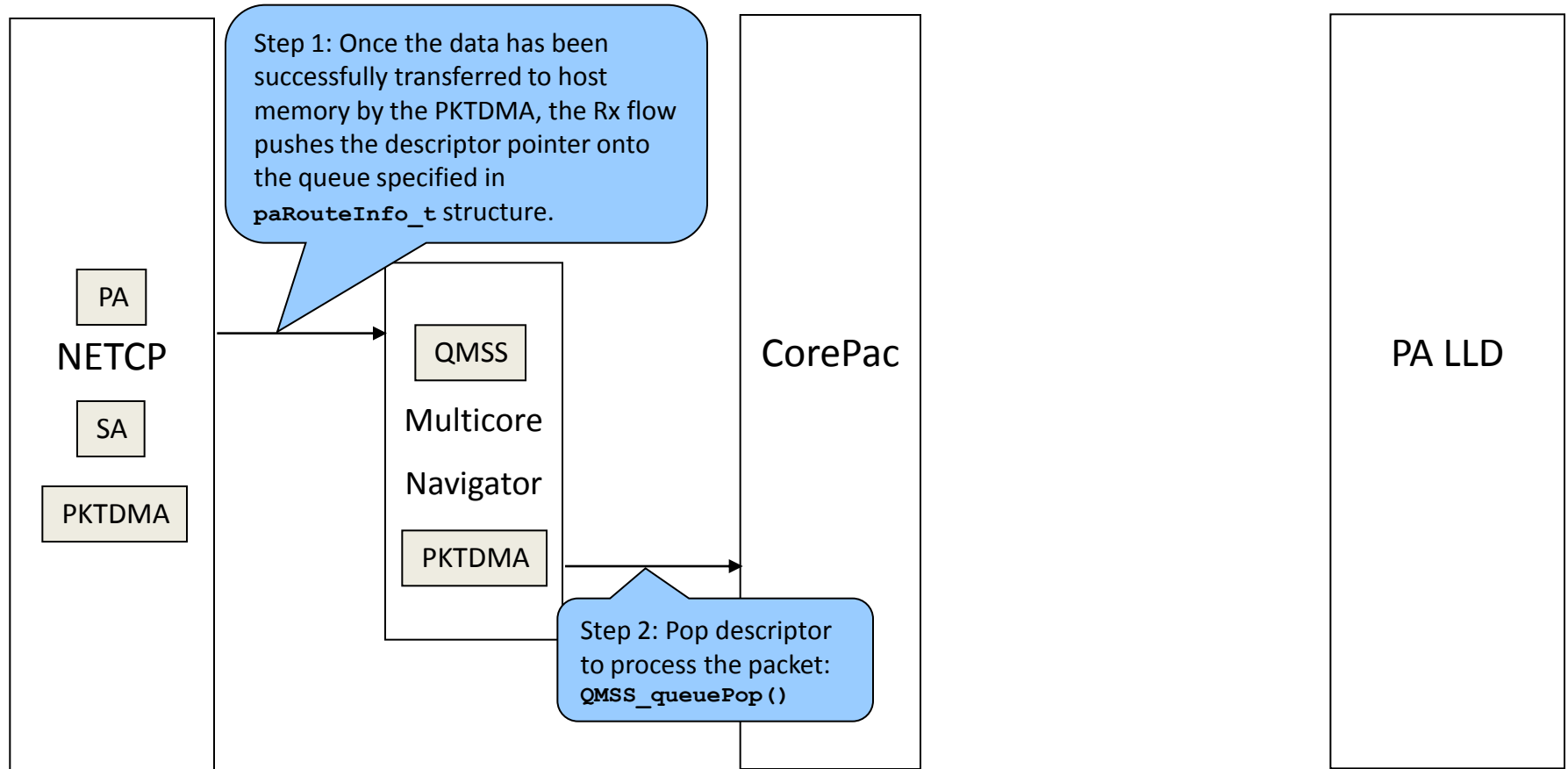




# PA LLD: Receive Packets from Ethernet

Steps 1-2 are repeated for all RX data packets.

Receive Data Packet

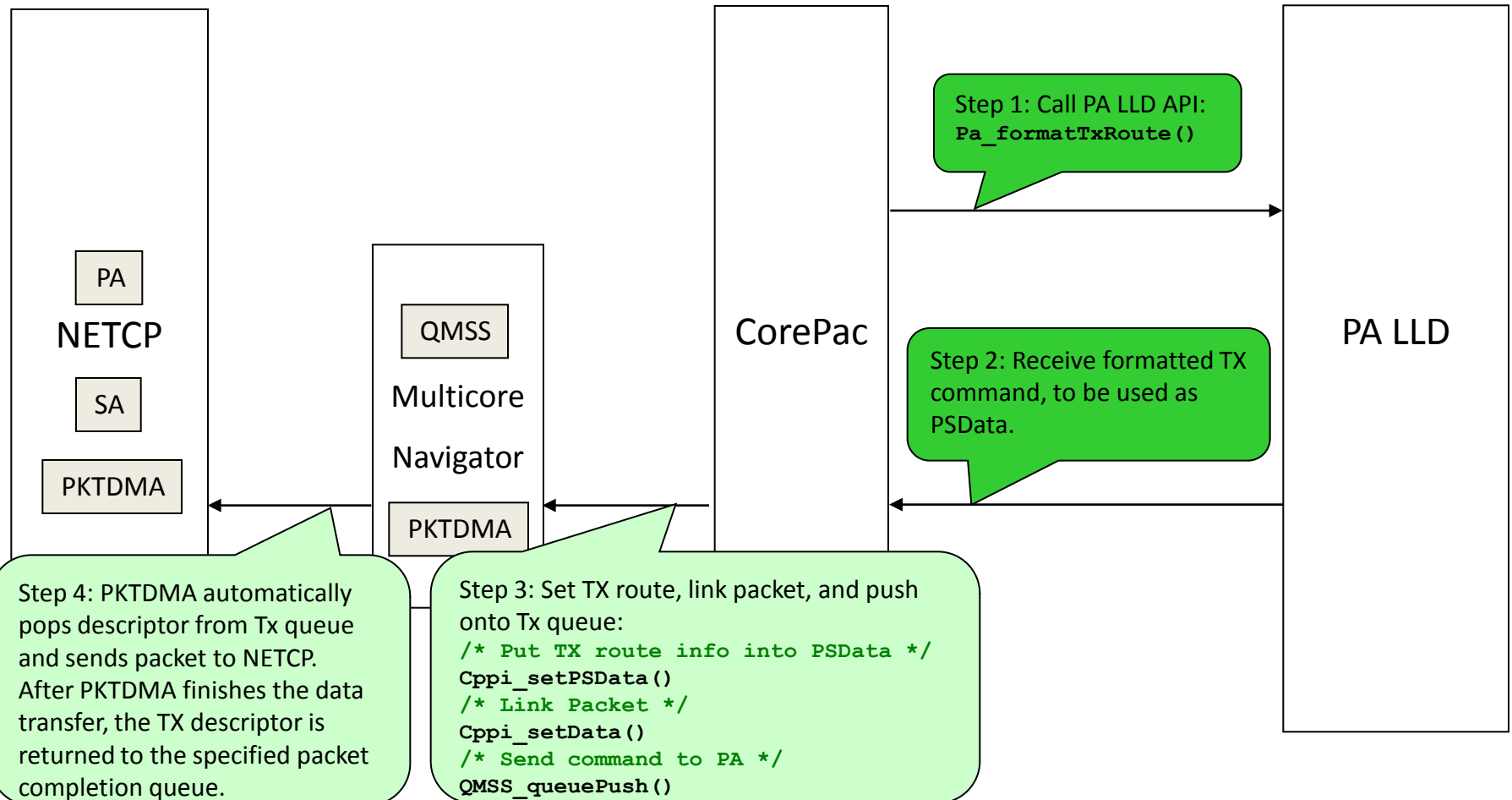


# PA LLD: Send Transmit Packet

Repeat steps 1-4 to create more send more TX packets.

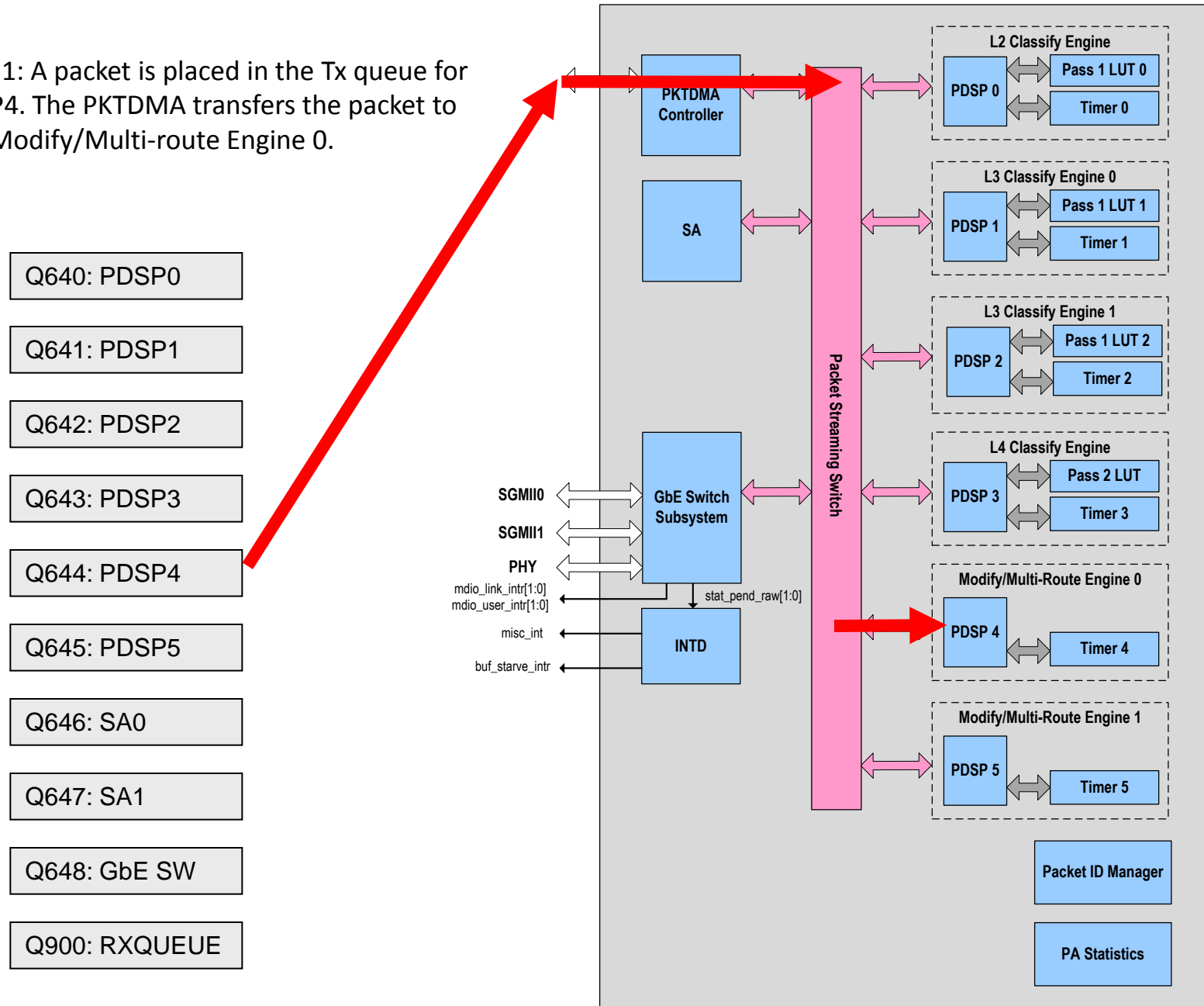
Transmit Configuration

Transmit Data Packet



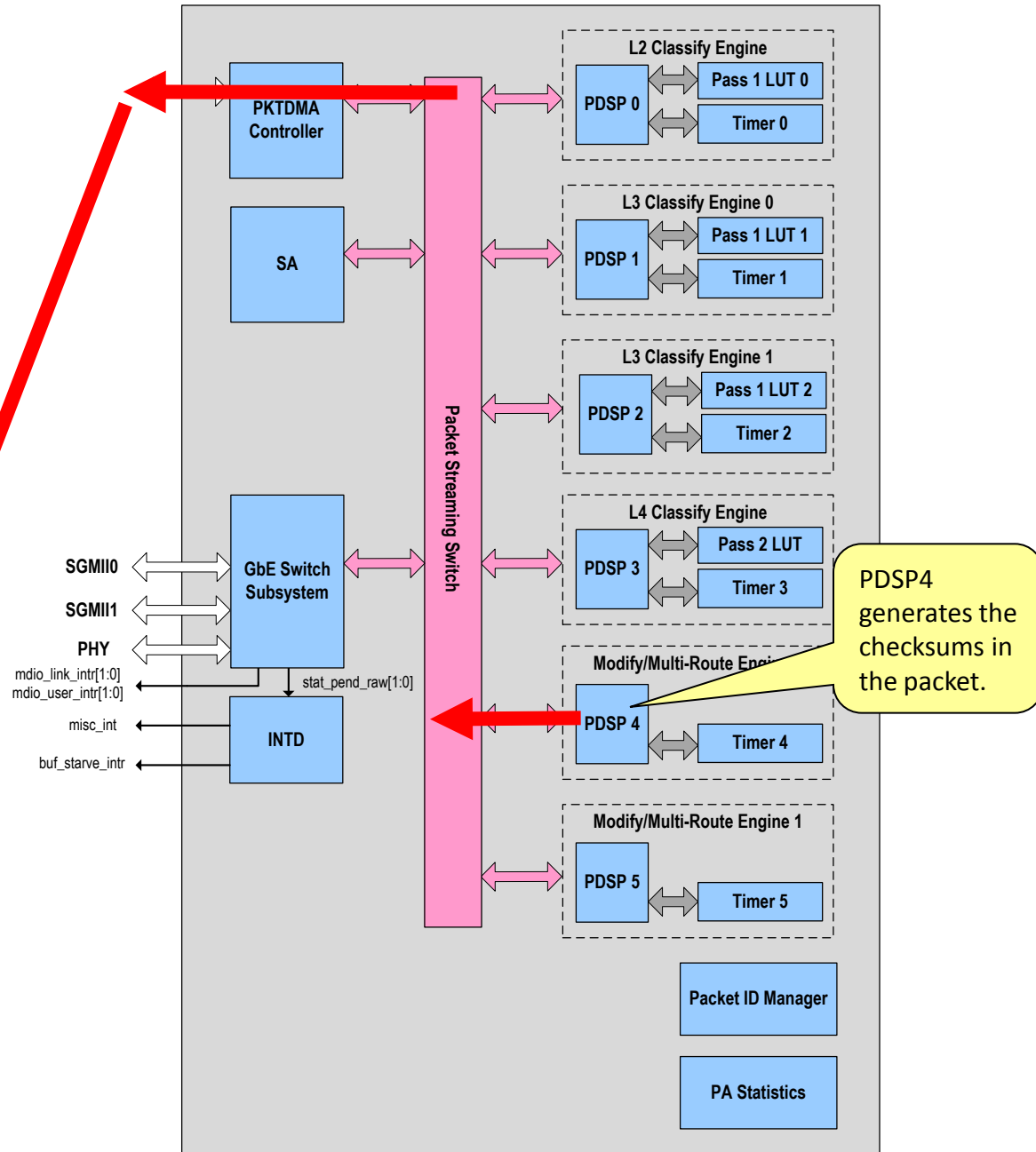
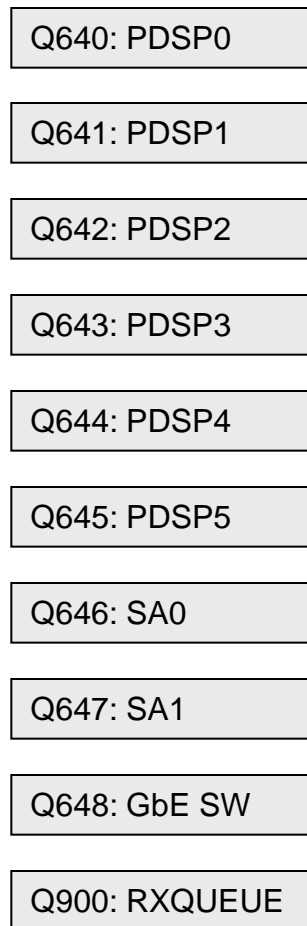
# PA Tx Hardware Processing

Step 1: A packet is placed in the Tx queue for PDSP4. The PKTDMA transfers the packet to the Modify/Multi-route Engine 0.



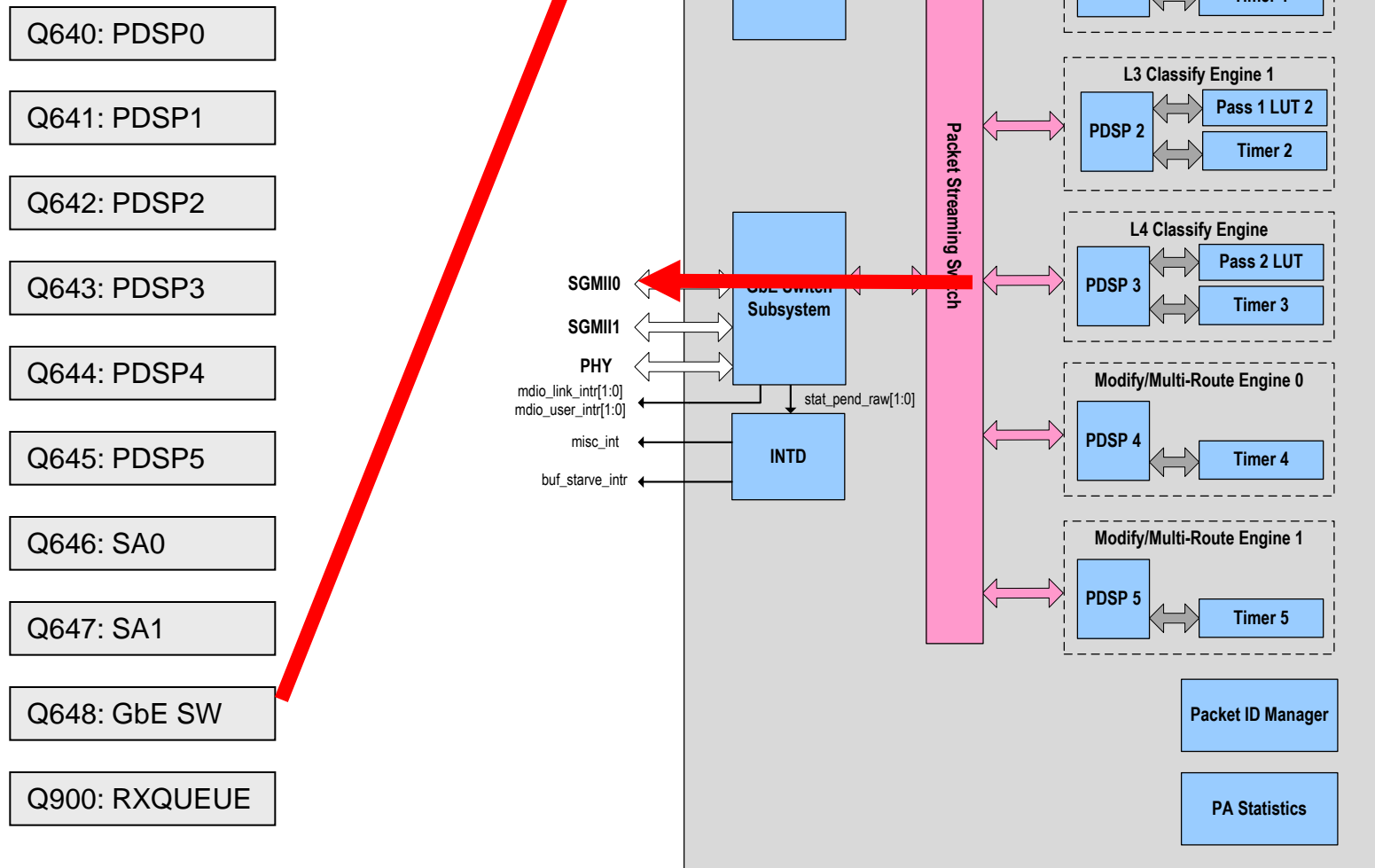
# PA TX Hardware Processing Example

Step 2: PDSP4 generates the checksums, writes them to the packet, and transmits the packet to the queue for the GbE switch via the PKTDMA.



# PA TX Hardware Processing Example

Step 3: The PKTDMA will transfer the packet from the GbE switch transmit queue through the PKTDMA controller and the packet streaming switch in the NETCP to the GbE switch subsystem for transmission over the network.



# For More Information

- For more information, refer to the following KeyStone device documents:
  - Network Coprocessor (NETCP) User Guide  
<http://www.ti.com/lit/SPRUGZ6>
  - Packet Accelerator (PA) User Guide  
<http://www.ti.com/lit/SPRUGS4>
- For questions regarding topics covered in this training, visit the support forums at the [TI E2E Community](#) website.