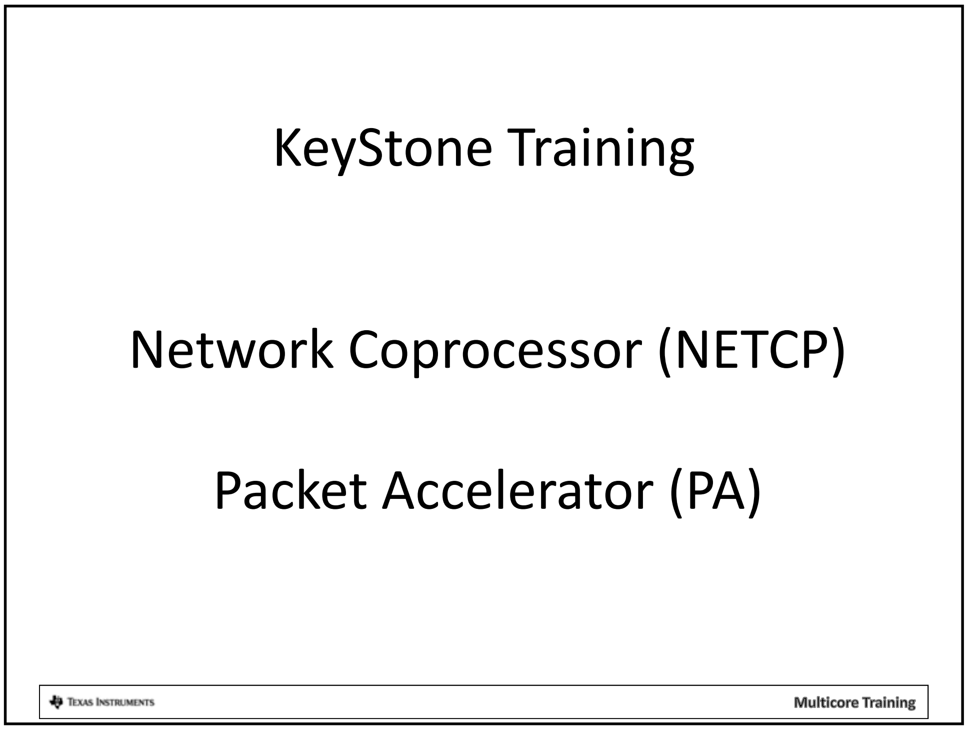
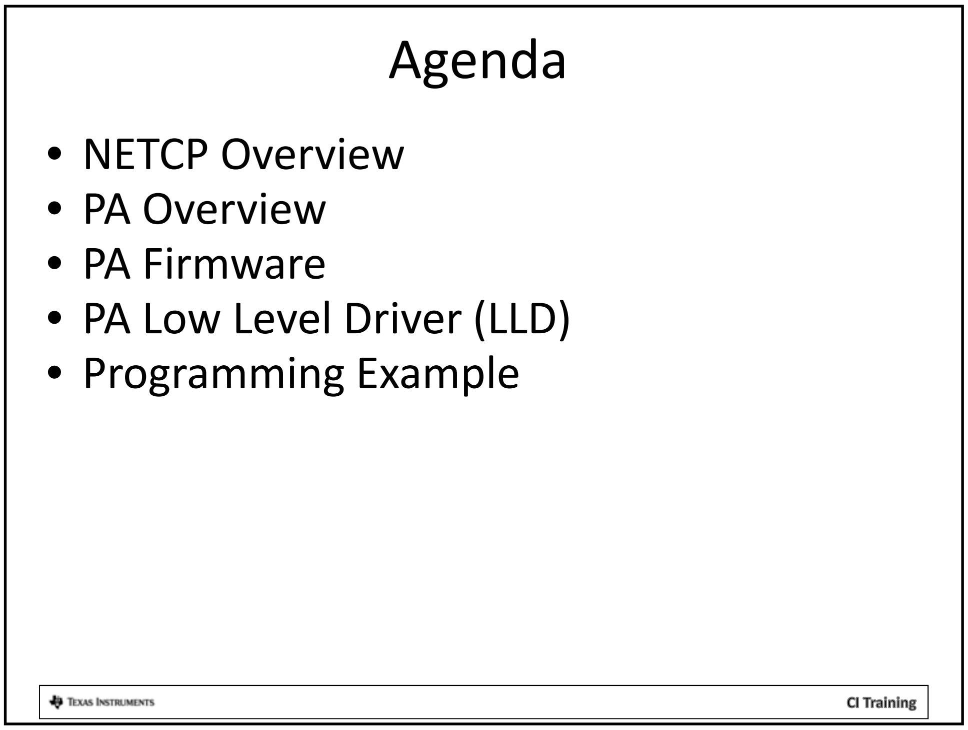


KeyStone Training

Network Coprocessor (NETCP)

Packet Accelerator (PA)



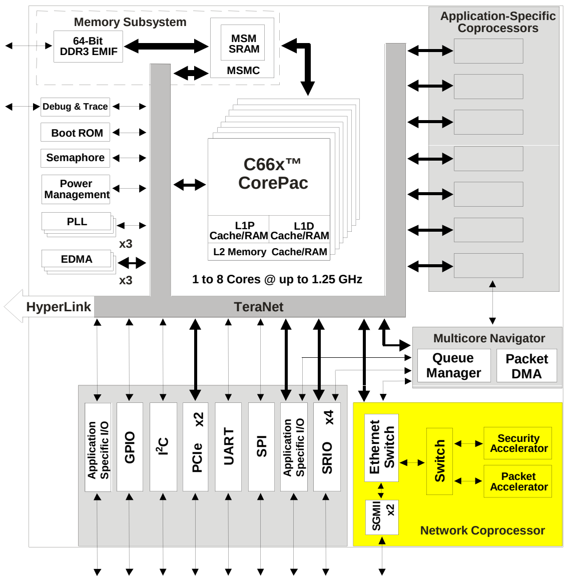
Agenda

- NETCP Overview
- PA Overview
- PA Firmware
- PA Low Level Driver (LLD)
- Programming Example

NETCP Overview

- NETCP Overview
- PA Overview
- PA Firmware
- PA Low Level Driver (LLD)
- Programming Example

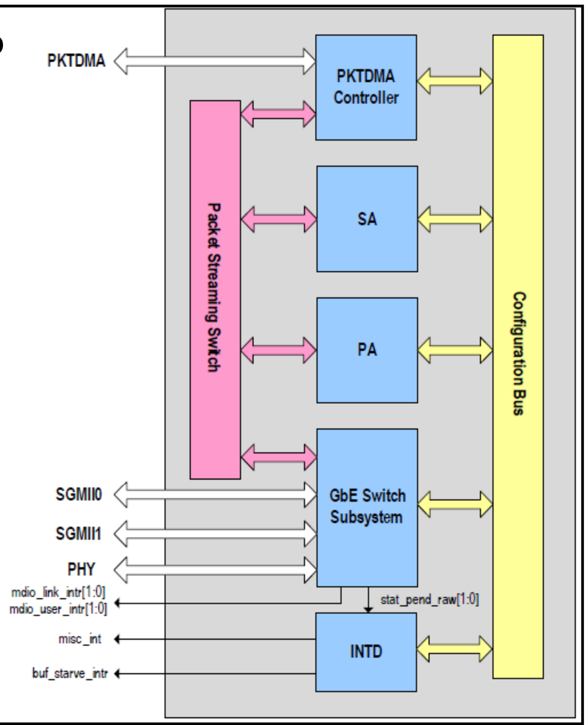
Network Coprocessor (NETCP)



What is NETCP?

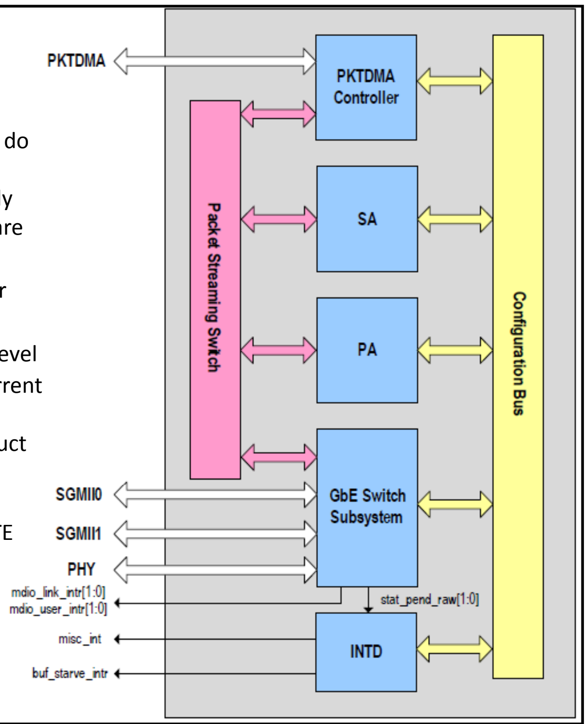
Network Coprocessor consists of the following modules:

- Packet DMA (PKTDMA) Controller
- Security Accelerator (SA)
- Packet Accelerator (PA)
- Gigabit Ethernet Switch Subsystem
- Distributed Interrupt Controller (INTD)



NETCP Purpose

- Motivation behind NETCP:
  - Use hardware accelerators to do L2, L3, and L4 processing and encryption that was previously required to be done in software
- Goals for both PA and SA:
  - Offload DSP processing power
  - Improve system integration
  - Allow cost savings at system level
  - Expand DSP usability w/in current products
  - Allow DSP usage in new product areas
- Security Key applications:
  - IPSec tunnel endpoint (e.g. LTE eNB, ...)
  - Secure RTP (SRTP) between gateways
  - Air interface (3GPP, Wimax) security processing

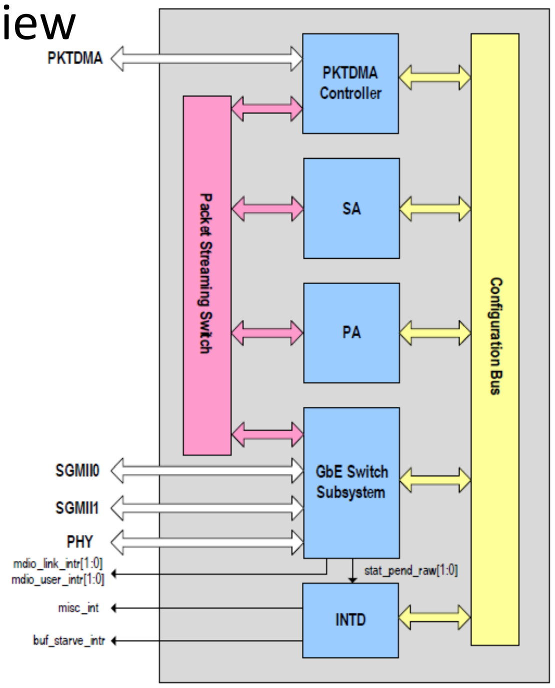


Packet Accelerator: Overview

- NETCP Overview
- PA Overview
- PA Firmware
- PA Low Level Driver (LLD)
- Programming Example

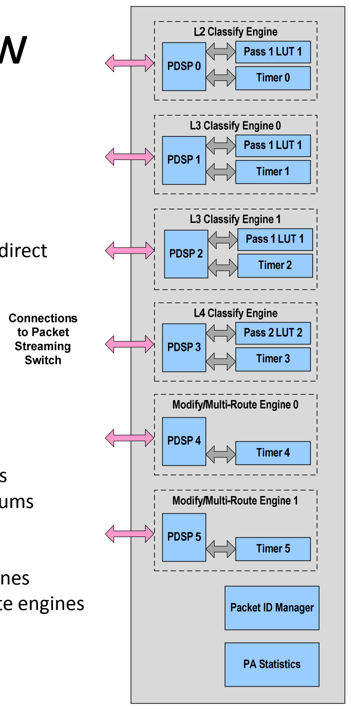
PA Features Overview

- Packet accelerator saves cycles from host DSP cores.
- 1 Gbps wire-speed throughput at 1.5 Mpps
- Option of single IP address for multi-core device; Multicore device internal architecture is abstracted
- UDP (and TCP) Checksum and selected CRCs 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
- QOS support (in conjunction with PKTDMA)
  - Per channel / flow to individual queue towards host DSPs
  - 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
- IEEE 1588 timestamps and configurable generic timestamps

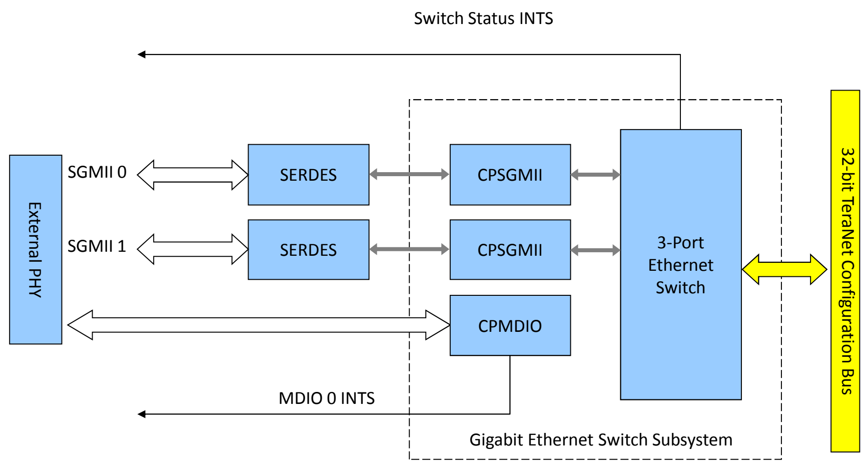


PA: Functional Overview

- 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



SGMII and Ethernet Switch



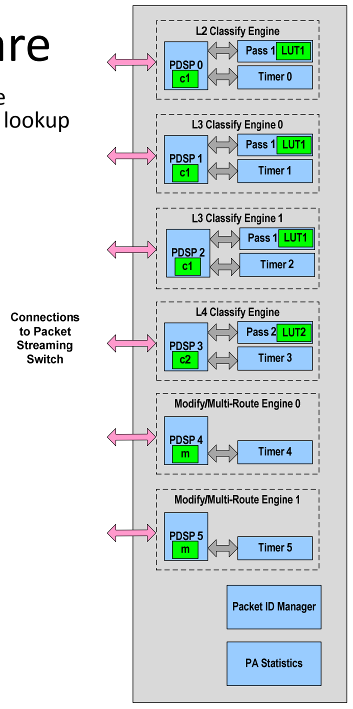
Texas Instruments CI Training

Packet Accelerator: Firmware

- NETCP Overview
- PA Overview
- PA Firmware
- PA Low Level Driver (LLD)
- Programming Example

PA: Hardware and Firmware

- Before using the PA engines, firmware images must be loaded into internal RAM to enable the PDSP to make lookup and routing decisions.
- 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 2 (c2) firmware image
- Two Modify/Multi-Route Engines
  - PDSP
  - Timer
  - Modify (m) firmware image



Packet Accelerator: LLD

- NETCP Overview
- PA Overview
- PA Firmware
- PA Low Level Driver (LLD)
- Programming Example

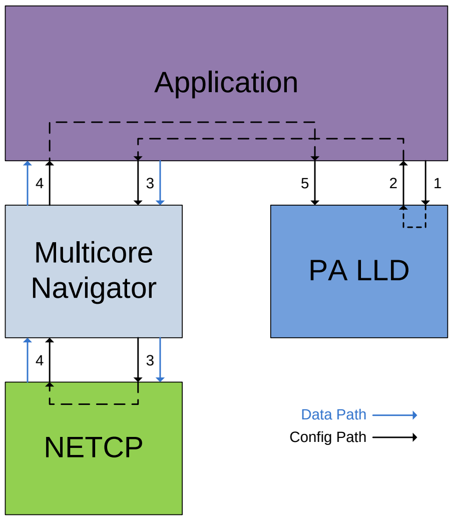
PA LLD Overview

- 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 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 reference within MCSDK: **`pa/docs/paDocs.chm`**

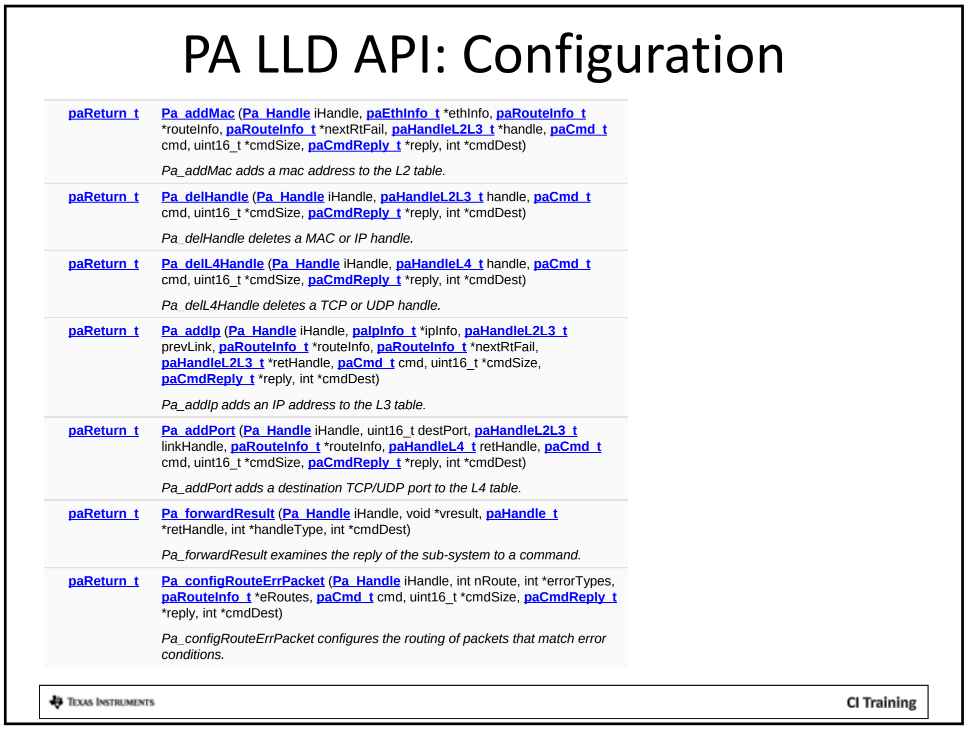
PA LLD Functional Diagram

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



PA LLD API: System

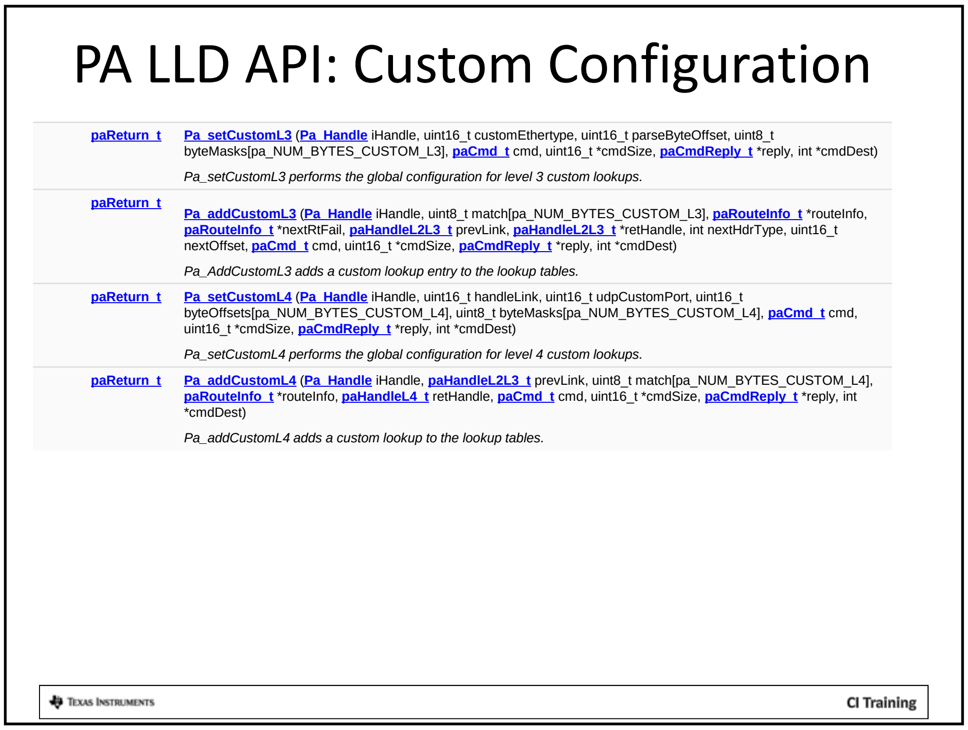
paReturn_t	Pa_getBufferReq (paSizeInfo_t *sizeCfg, int sizes[], int aligns[]) *Pa_getBufferReq returns the memory requirements for the PA driver.*
paReturn_t	Pa_create (paConfig_t *cfg, void *bases[], Pa_Handle *pHandle) *Pa_create creates the PA driver instance.*
paReturn_t	Pa_close (Pa_Handle handle, void *bases[]) *Pa_close decativates 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.*

Texas Instruments CI Training



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.*

Texas Instruments CI Training

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

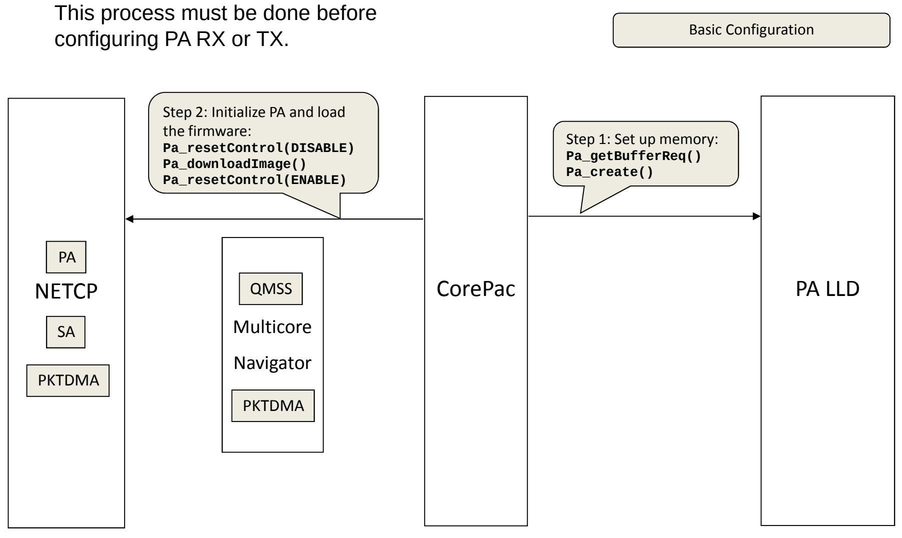
```
Int paDownloadFirmware (void)
{
  Int i;

  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

- NETCP Overview
- PA Overview
- PA Firmware
- PA Low Level Driver (LLD)
- Programming Example

PA LLD: Basic Configuration

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



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:

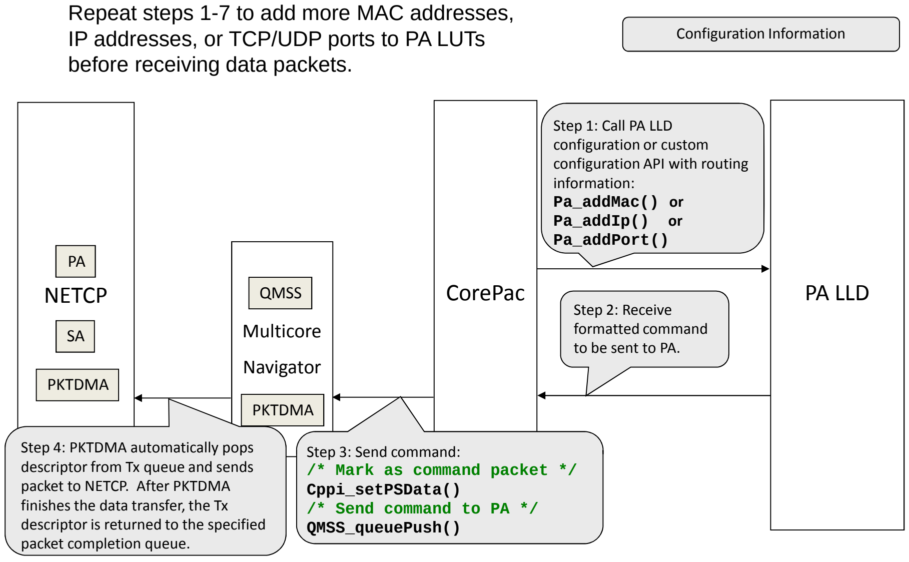
```
paRouteInfo_t macRoute;
/* Continue parsing -- try to match IP
handle*/
macRoute.dest = pa_DEST_CONTINUE_PARSE;
macRoute.flowId = 0;
macRoute.queue = 0;
macRoute.mRouteHandle = -1;
macRoute.swInfo0 = 0;      /* Don't Care */
macRoute.swInfo1 = 0;      /* Don't Care */
```

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.



Step 1: Call PA LLD configuration or custom configuration API with routing information:
`Pa_addMac()` or
`Pa_addIp()` or
`Pa_addPort()`

Step 2: Receive formatted command to be sent to PA.

Step 3: Send command:

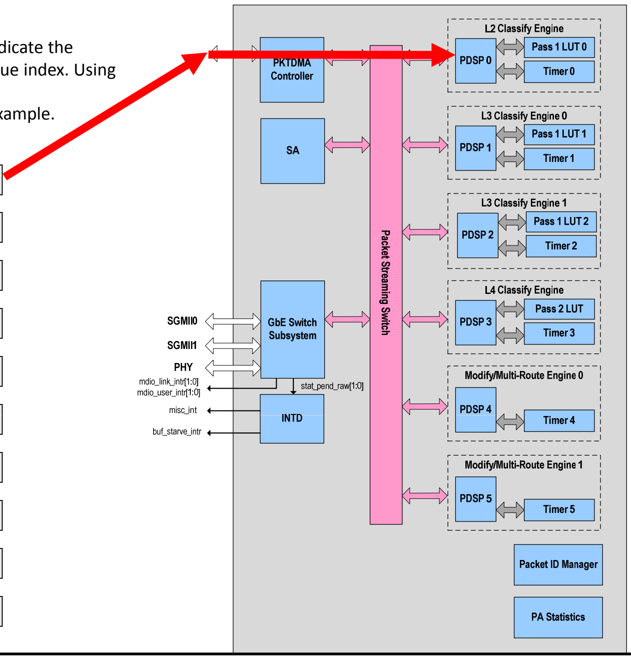
```
/* Mark as command packet */
Cppi_setPSData()
/* Send command to PA */
QMSS_queuePush()
```

Step 4: PKTDMA automatically pops descriptor from Tx queue and sends packet to NETCP. After PKTDMA finishes the data transfer, the Tx descriptor is returned to the specified packet completion queue.

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.

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



PA LLD: Rx Configuration

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

Q640: PDSP0

Q641: PDSP1

Q642: PDSP2

Q643: PDSP3

Q644: PDSP4

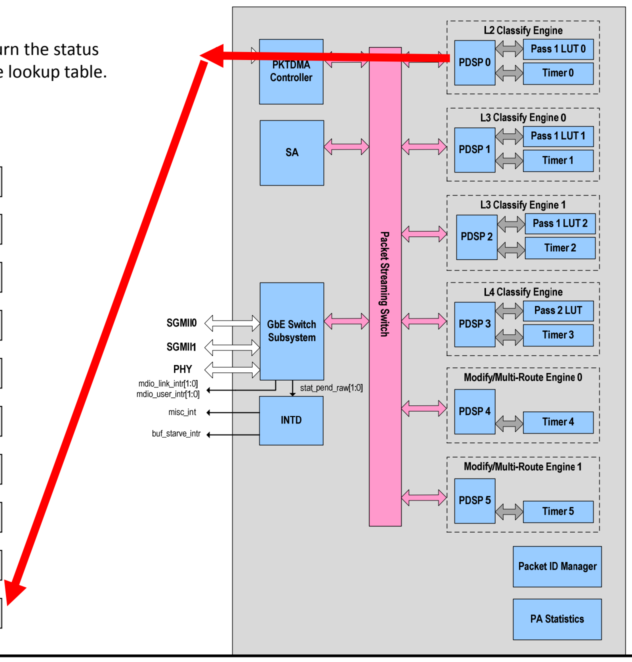
Q645: PDSP5

Q646: SA0

Q647: SA1

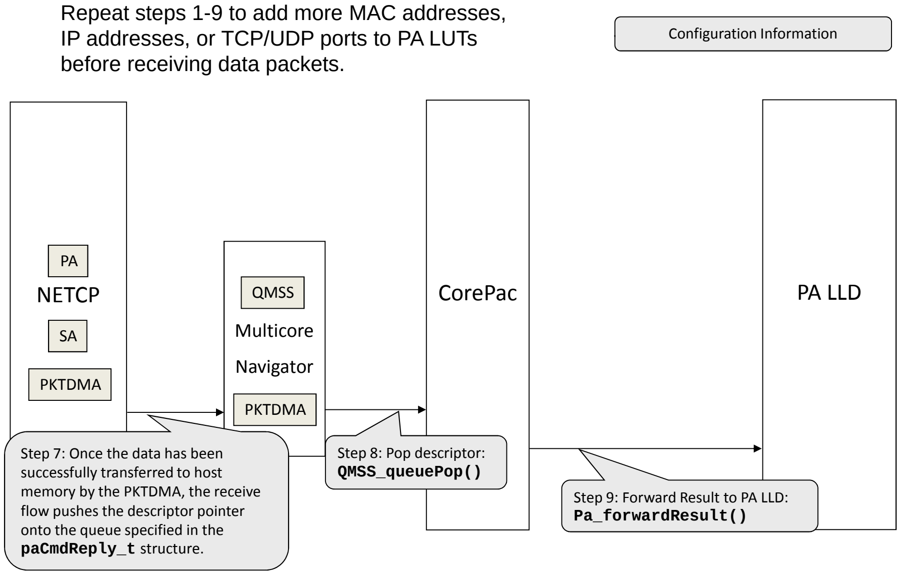
Q648: GbE SW

Q900: RXQUEUE



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.



Step 7: Once the data has been successfully transferred to host memory by the PKTDMA, the receive flow pushes the descriptor pointer onto the queue specified in the **`paCmdReply_t`** structure.

Step 8: Pop descriptor: **`QMSS_queuePop()`**

Step 9: Forward Result to PA LLD: **`Pa_forwardResult()`**

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

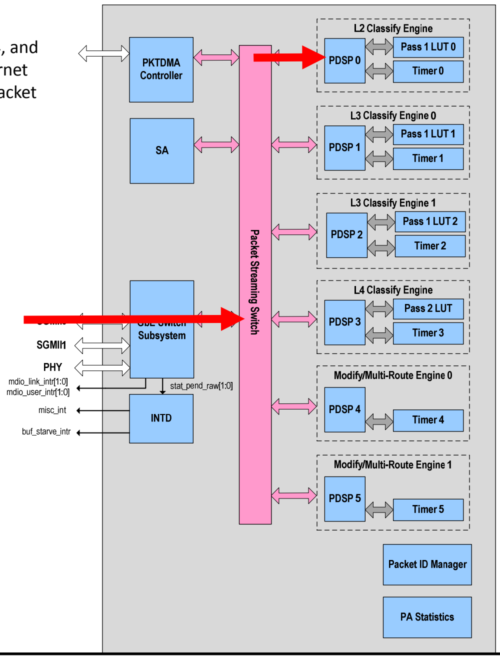
Q645: PDSP5

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

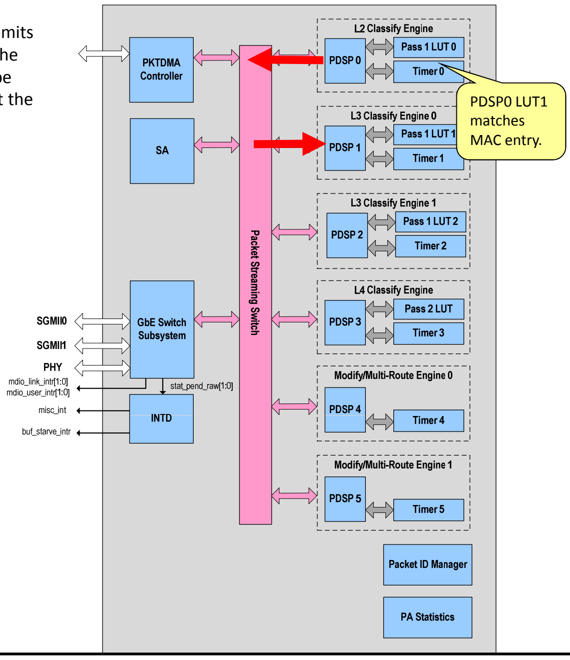
Q645: PDSP5

Q646: SA0

Q647: SA1

Q648: GbE SW

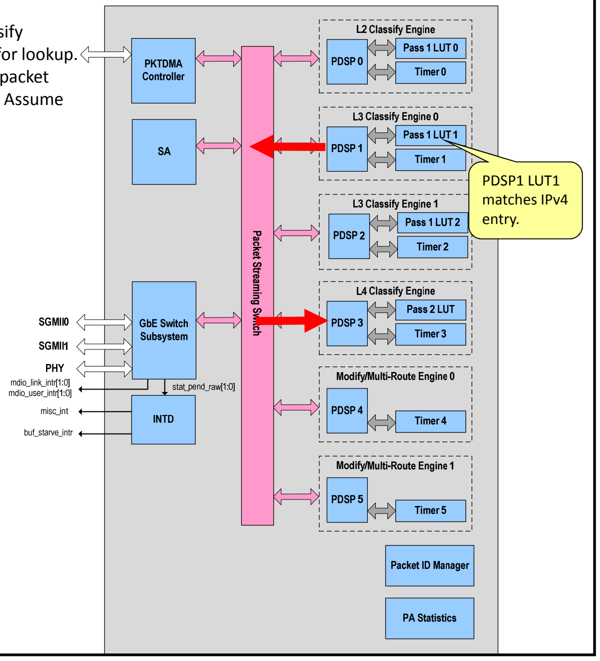
Q900: RXQUEUE



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.

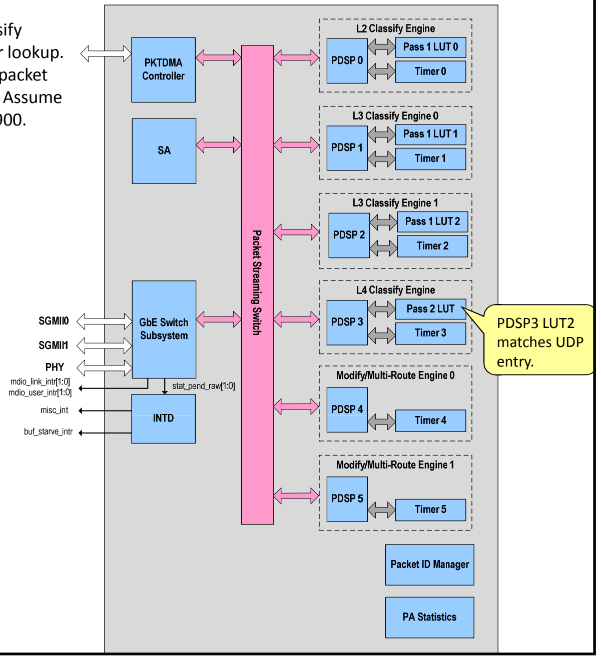
Q640: PDSP0
Q641: PDSP1
Q642: PDSP2
Q643: PDSP3
Q644: PDSP4
Q645: PDSP5
Q646: SA0
Q647: SA1
Q648: GbE SW
Q900: RXQUEUE



PA Rx Hardware Processing

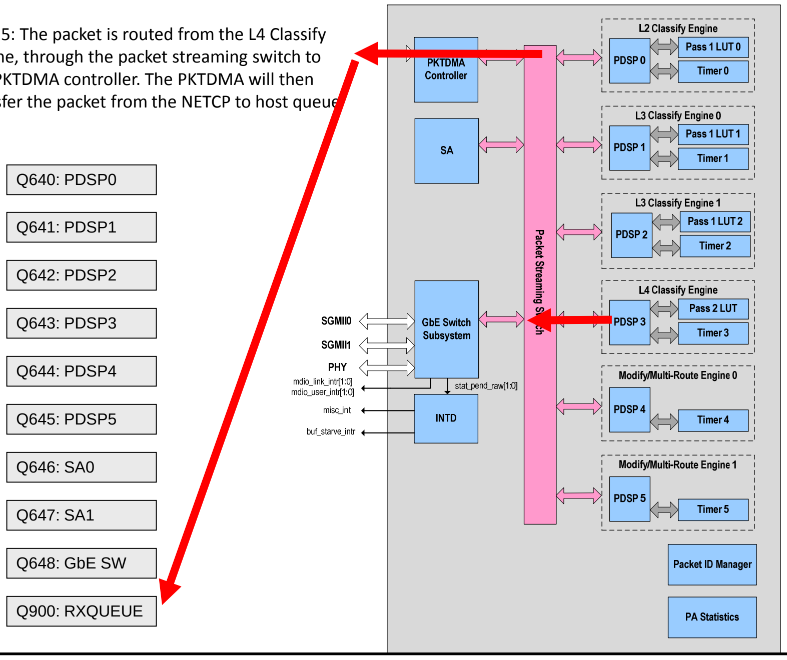
Step 4: The packet is routed to the L4 Classify Engine. PDSP3 submits the UDP header for lookup. Assume that the lookup is successful. The packet will then be routed to its next destination. Assume that the destination is the host on queue 900.

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



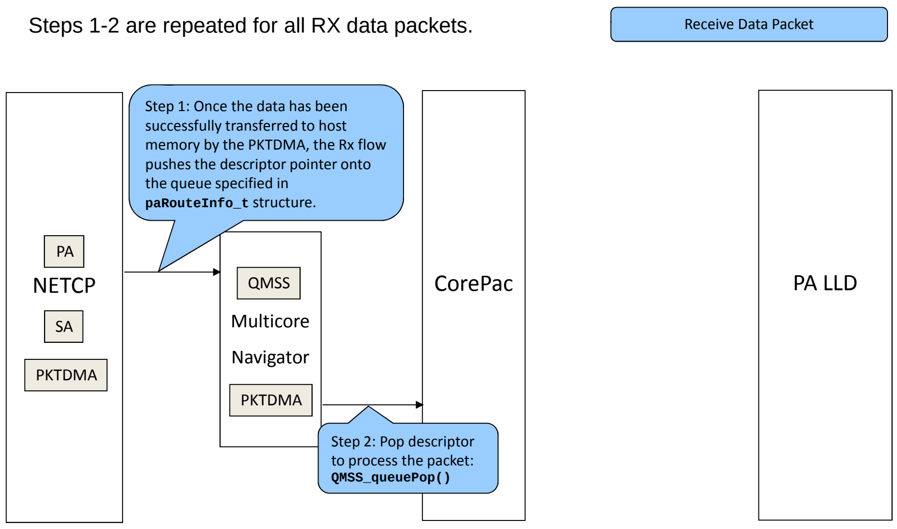
PA Rx Hardware Processing

Step 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.



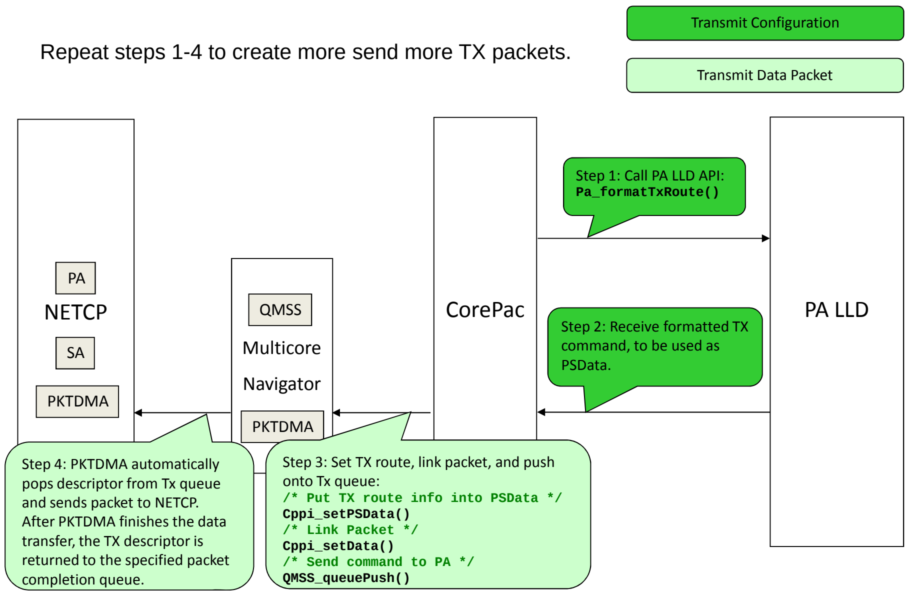
PA LLD: Receive Packets from Ethernet

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


Receive Data Packet
Step 1: Once the data has been successfully transferred to host memory by the PKTDMA, the Rx flow pushes the descriptor pointer onto the queue specified in paRouteInfo_t structure.
PA
NETCP
SA
PKTDMA
QMSS
Multicore
Navigator
PKTDMA
CorePac
PA LLD
Step 2: Pop descriptor to process the packet: QMSS_queuePop()

PA LLD: Send Transmit Packet

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



Step 1: Call PA LLD API: **`Pa_formatTxRoute()`**

Step 2: Receive formatted TX command, to be used as PSData.

Step 3: Set TX route, link packet, and push onto Tx queue:

```
/* Put TX route info into PSData */
Cppi_setPSData()
/* Link Packet */
Cppi_setData()
/* Send command to PA */
QMSS_queuePush()
```

Step 4: PKTDMA automatically pops descriptor from Tx queue and sends packet to NETCP. After PKTDMA finishes the data transfer, the TX descriptor is returned to the specified packet completion queue.

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.

Q640: PDSP0

Q641: PDSP1

Q642: PDSP2

Q643: PDSP3

Q644: PDSP4

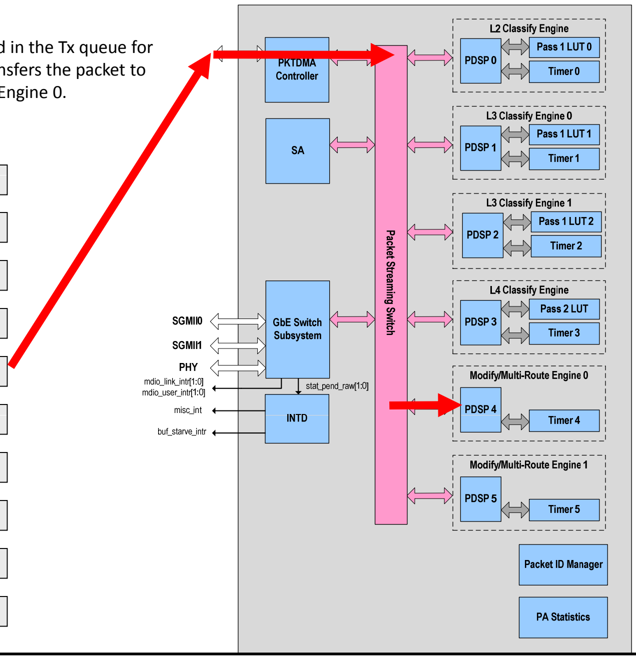
Q645: PDSP5

Q646: SA0

Q647: SA1

Q648: GbE SW

Q900: RXQUEUE



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.

Q640: PDSP0

Q641: PDSP1

Q642: PDSP2

Q643: PDSP3

Q644: PDSP4

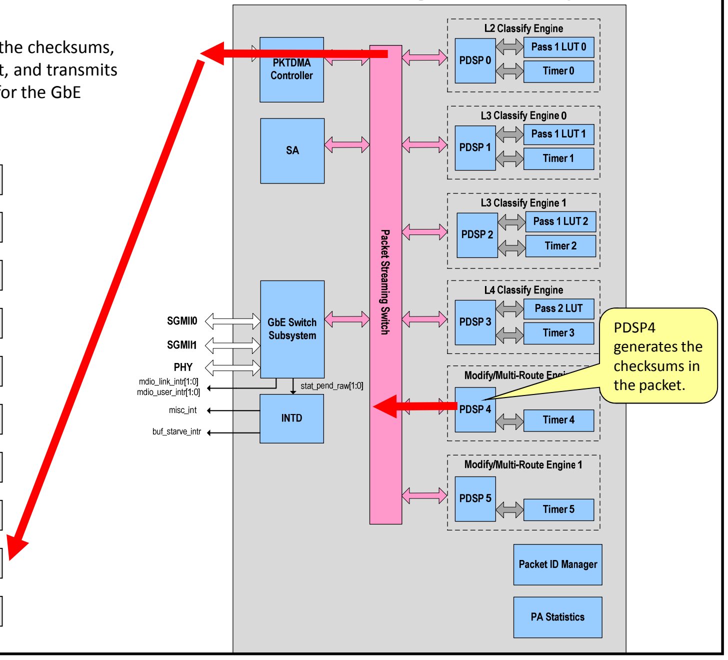
Q645: PDSP5

Q646: SA0

Q647: SA1

Q648: GbE SW

Q900: RXQUEUE



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.

Q640: PDSP0

Q641: PDSP1

Q642: PDSP2

Q643: PDSP3

Q644: PDSP4

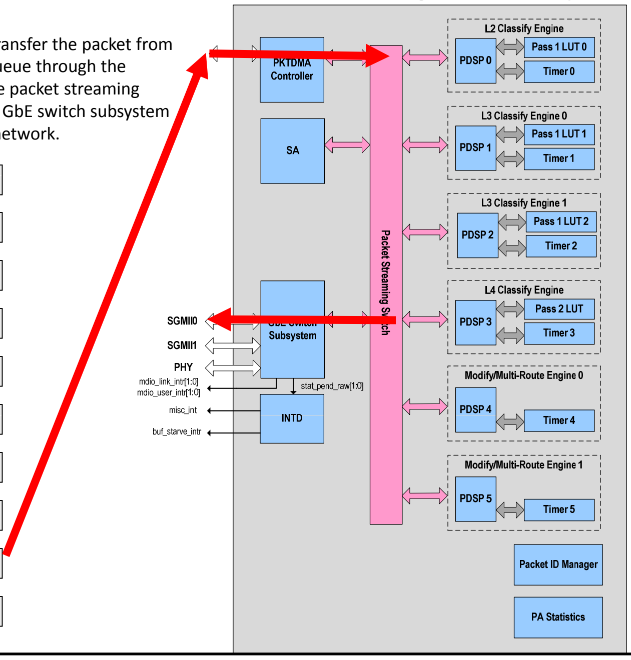
Q645: PDSP5

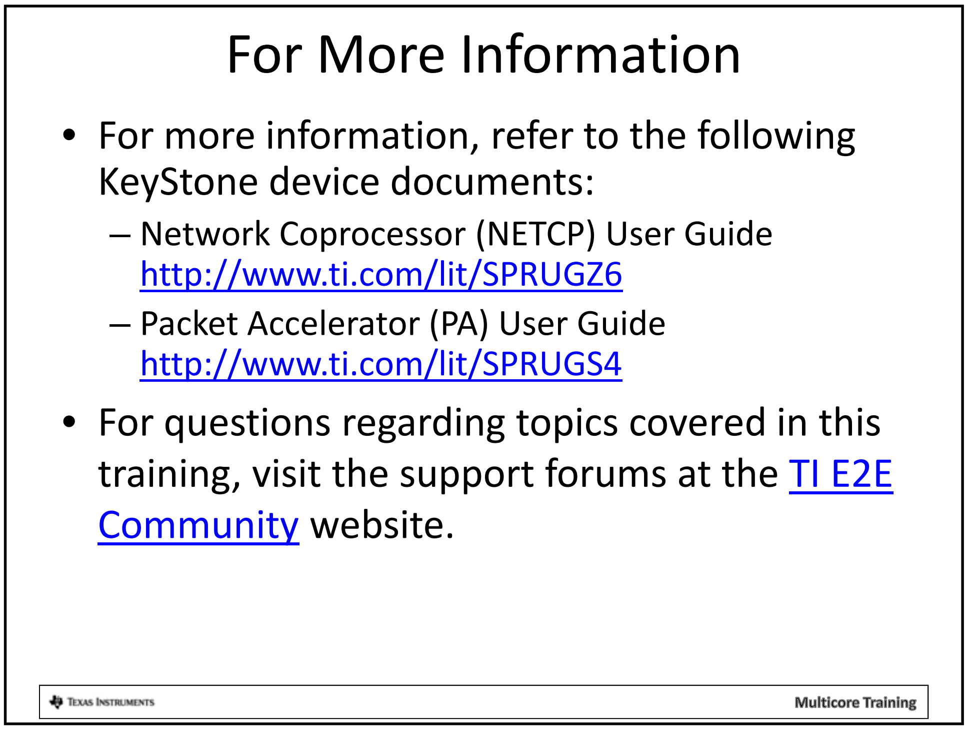
Q646: SA0

Q647: SA1

Q648: GbE SW

Q900: RXQUEUE





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.

Texas Instruments Multicore Training