KeyStone Training

Multicore Navigator: Packet DMA (PKTDMA)

Agenda

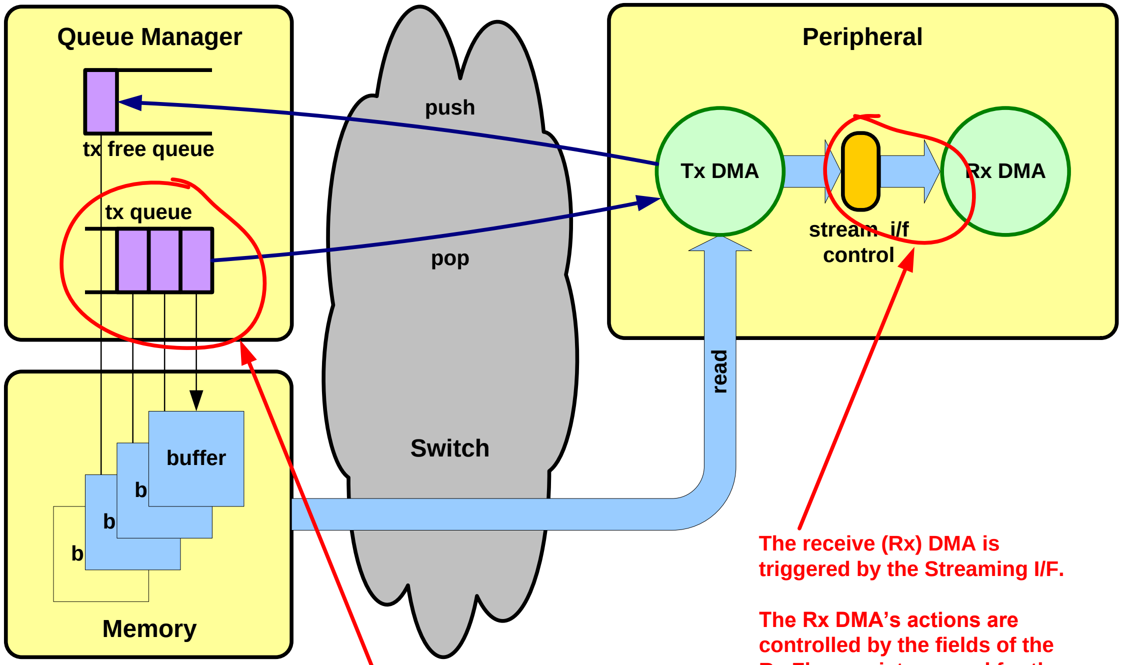
- How Does the Packet DMA Work?
  - Triggering
  - Tx Processing
  - Rx Processing
  - Infrastructure Processing
- How to Program the Packet DMA
  - Registers
  - Low Level Driver
- Final Advice / Tips

How Does the Packet DMA Work?

- How Does the Packet DMA Work?
  - Triggering
  - Tx Processing
  - Rx Processing
  - Infrastructure Processing
- How to Program the Packet DMA
  - Registers
  - Low Level Driver
- Final Advice / Tips

Packet DMA Control

Understanding how the PKTDMAs are triggered and controlled is critical.



The transmit (Tx) DMA is triggered by a DSP task or other hardware pushing to a Tx queue.

The Tx DMA's actions are controlled by the fields of the descriptor.

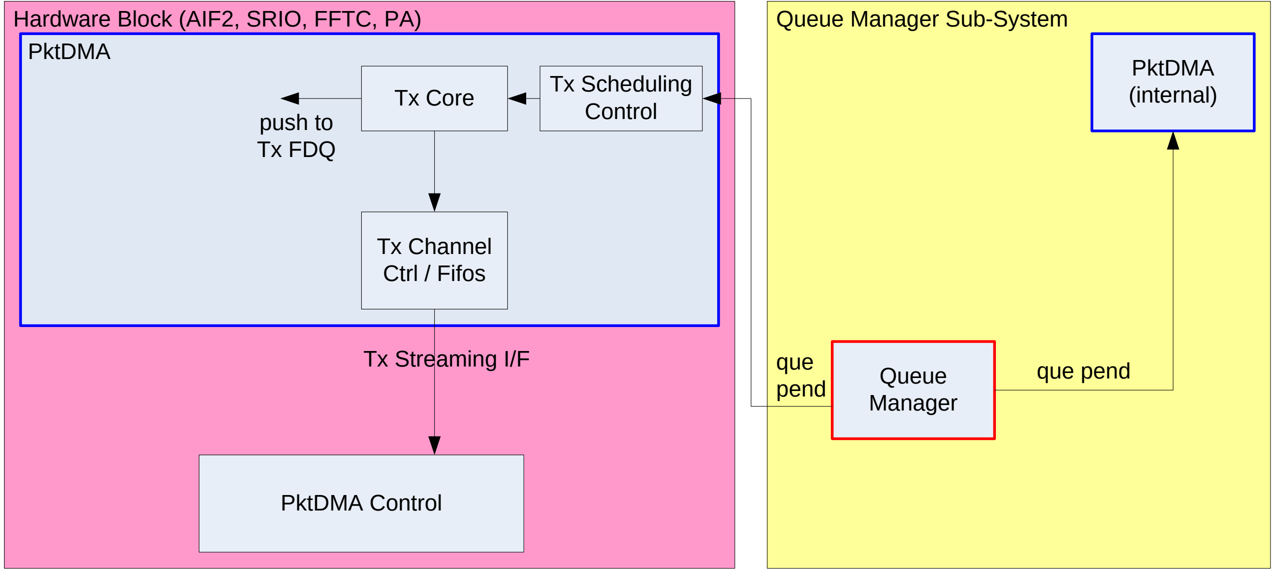
The receive (Rx) DMA is triggered by the Streaming I/F.

The Rx DMA's actions are controlled by the fields of the Rx Flow registers used for the packet being sent.

In Infrastructure (loopback) mode, certain fields in the Tx descriptor drive the Rx DMA.

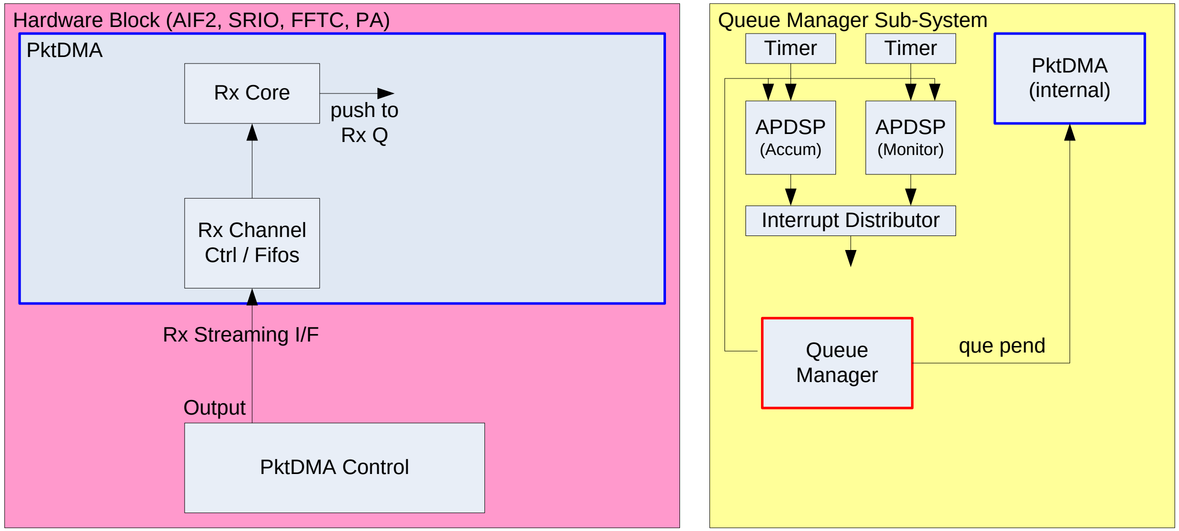
Tx DMA Processing

- Once triggered, the Tx channel is included in a 4-level priority round robin by the Tx Scheduling Control.
- Once selected for processing, Tx Core is started, pops the first descriptor off the associated queue and begins processing.
- Data are buffered temporarily in a per-channel Tx FIFO.
- Data are sent out the Streaming I/F at 128 bits/clock.
- When done, the descriptor is automatically recycled to a Tx FDQ.



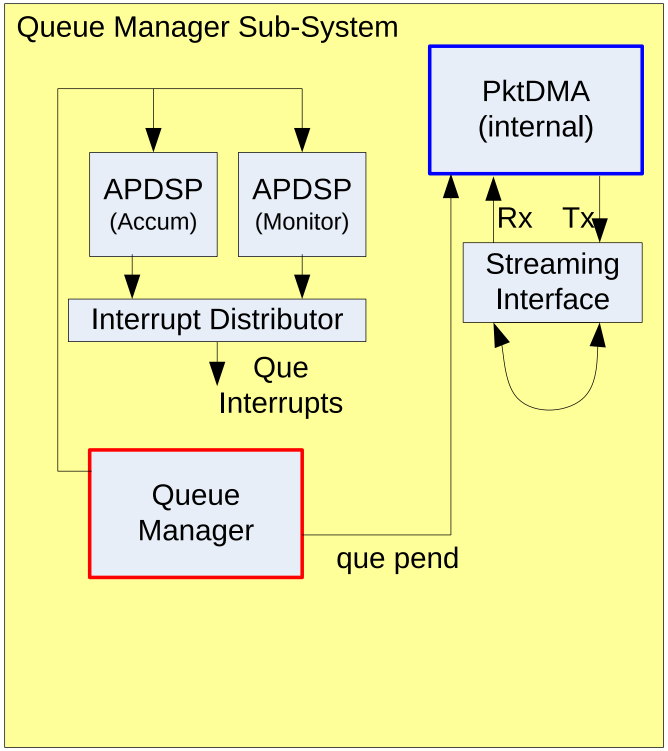
Rx DMA Processing

- For receive, the Rx DMA is triggered by receiving data from the Rx Streaming I/F (also at 128 bits/clock).
- Based on the channel (or packet info), the Rx DMA opens an Rx Flow (which contains processing instructions such as the destination Rx queue and Rx FDQs that can be used).
- Data are buffered temporarily in a per-channel Rx FIFO.
- Data are written to destination memory and the descriptor is pushed to the destination Rx queue, which should be recycled by the Host.



Infrastructure Packet DMA

- The QMSS PKTDMA's Rx and Tx Streaming I/F are wired together to create loopback.
- Data sent out the Tx side are immediately received by the Rx side.
- This PKTDMA is used for core-to-core transfers and peripheral-to-DSP transfers.
- Because the DSP is often the recipient, a descriptor accumulator can be used to gather (pop) descriptors and interrupt the host with a list of descriptor addresses. The host must recycle them.



How Does the Packet DMA Work?

- How Does the Packet DMA Work?
  - Triggering
  - Tx Processing
  - Rx Processing
  - Infrastructure Processing
- How to Program the Packet DMA
  - Registers
  - Low Level Driver
- Final Advice / Tips

Step 1: Logical QM Mapping (Optional)

- The physical 8192 queue Queue Manager is viewed by each PKTDMA as four logical queue managers, each having up to 4095 usable queues each.
  - Two fields are used: a 2-bit "qmgr" (queue manager 0..3) and a 12-bit "qnum" (queue number 0..4094).
  - A logical QM is created by programming this register to the queue region base address + 16 * Q, where Q is the first physical queue. Example:

    QM2 = 0x34020000 + 0xc000 maps queue 3072 to qmgr=2, qnum=0.
  - A logical 8K QM can be created: QM0 maps to 0, QM1 maps to 4096. In this way only can a queue number be used as a single14-bit value.
  - If overlapping QMs are programmed, a physical queue may be referenced by more than one qmgr:qnum pair.
  - Software must map logical to physical queue numbers.

– Queue Manager N Base Address:

- Queue Manager 0 reset value points to physical queue zero.
- All others must point to an offset in units of 16 bytes from queue zero.
- Unused QMs can be left uninitialized.

Step 2: Rx Flow Setup

- Rx Flow
  - Flows are not tied to a channel; They can change at packet rate since the flow ID is sent from the Streaming I/F with each SOP.
  - The flow instructs the Rx DMA how to create an output descriptor based on the following:
    - Descriptor type
    - Data offset
    - Sideband info include or ignore (PS, EPIB, etc.)
    - Rx qmgr:qnum (destination queue)
    - Rx FDQ selection:
      - Single
      - Based on packet size
      - Based on buffer number (Host packet type only)

  - Rx Flow N Config Registers A-H:
    - See Register Definitions in the Multicore Navigator Users Guide.

Step 3: Rx DMA Setup

- For each Rx channel of each Navigator peripheral to be used, initialize the following:
  - Rx Channel Control
    - Rx Channel N Global Config Register A:
      - Channel enable, pause, teardown enable
      - Channel should be disabled while configuring (this also includes configuring the Rx Flows).
      - This is typically the last initialization step for each Rx DMA. (Rx flows should also be programmed prior to this register)

Step 4: Tx DMA Setup

- For each Tx channel of each Navigator peripheral to be used, initialize the following:
  - Tx Channel Priority
    - Tx Channel N Scheduler Configuration:
      - Configures a 4-bit priority for the 4 level round robin.
      - Defaults to zero (high)
  - Tx Channel Control
    - Tx Channel N Global Config Register A:
      - Channel enable, pause, teardown enable
      - Channel should be disabled while configuring
      - This is typically the last initialization step for each Tx DMA.

Step 5: Other Initialization

- Initialize Rx and Tx FDQs:
  - Initialize the descriptor fields
    - Rx Host descriptors must have buffers attached.
    - Rx Host descriptors must not be chained together.
    - Tx Host descriptors for FDQs can be chained or not chained, and buffers attached or not (software decides how to handle this).
  - Push the initialized descriptors into an FDQ with Que N Register D (QM registers)

  - Assuming the Queue Manager has been initialized prior to the PKTDMA instances, Navigator is now ready to be used

Packet DMA Low Level Driver (LLD)

- Packet DMA is commonly represented in the LLD as CPPI
- Provides an abstraction of register-level details
- Provides two usage modes:
  - User manages/selects resources to be used.
    - Generally faster
  - LLD manages/selects resources.
    - Generally easier
- Allocates a minimal amount of memory for bookkeeping purposes.

- Built as two drivers:
  - QMSS LLD is a standalone driver for QM and Accumulators.
  - CPPI LLD is a driver for PKTDMA that requires the QMSS LLD.

- The following slides do not present the full API.

CPPI LLD Initialization

- The following are one-time initialization routines to configure the LLD globally:
  - Cppi_init(pktdma_global_parms);
    - Configures the LLD for one PKTDMA instance
    - May be called on one or all cores
    - Must be called once <u>for each</u> PKTDMA to be used
  - Cppi_exit();
    - Deinitializes the CPPI LLD

CPPI LLD: PKTDMA Channel Setup

- More handles to manage in using the PKTDMA LLD
- To allocate a handle for a PKTDMA:
  - pktdma_handle = CPPI_open(pktdma_parms);
    - Returns a handle for <u>one</u> PKTDMA instance
    - Should be called once for each PKTDMA required.
- To allocate and release Rx channels:
  - rx_handle = Cppi_rxChannelOpen(pktdma_handle, cfg, *flag);
    - Once "open", the DSP may use the Rx channel.
      - cfg refers to the Rx channel's setup parameters
      - flag is returned true if the channel is already allocated
  - Cppi_channelClose(rx_handle);
    - Releases the handle preventing further use of the queue

More Packet DMA Channel Setup

- To allocate and release Tx channels:
  - tx_handle = Cppi_txChannelOpen(pktdma_handle, cfg, *flag);
    - Same as the Rx counterpart
  - Cppi_channelClose(tx_handle);
    - Same as the Rx counterpart
- To configure/open an Rx Flow:
  - flow_handle = Cppi_configureRxFlow(pktdma_handle, cfg, *flag);
    - Similar to the Rx channel counterpart

PKTDMA Channel Control

- APIs to control Rx and Tx channel use:
  - Cppi_channelEnable(tx/rx_handle);
    - Allows the channel to begin operation
  - Cppi_channelDisable(tx/rx_handle);
    - Allows for an immediate, hard stop.
    - Usually not recommended unless following a pause.
  - Cppi_channelPause(tx/rx_handle);
    - Allows for a graceful stop at next end-of-packet
  - Cppi_channelTeardown(tx/rx_handle);
    - Allows for a coordinated stop

QMSS/CPPI LLD – Runtime Use

- Once initialization is finally complete, control is very simple:
  - desc_ptr = Qmss_queuePop(queue_handle);
    - Pop a descriptor address from a queue.
  - Cppi_setData(type, *inbuf, *desc_ptr, len);
    - Converts an “LLD format” descriptor to hardware format.
  - Qmss_queuePushDesc(queue_handle, desc_ptr);
    - Push the filled descriptor to a queue corresponding to a Tx DMA channel for processing.

Programming Tips

- How Does the Packet DMA Work?
  - Triggering
  - Tx Processing
  - Rx Processing
  - Infrastructure Processing
- How to Program the Packet DMA
  - Registers
  - Low Level Driver
- Final Advice / Tips

Final Advice

- Multicore Navigator is very flexible and can be used in many ways to solve the same problem. Your application must create *and follow* it's own rules. For example:
  - FDQ usage – Tx Host type pre-linked or not? Buffers attached?
  - FDQ mapping – to core, size, memory, function, flow?
  - Keep it simple – use one resource for one purpose!
- Multicore Navigator hardware assumes you have programmed it correctly – which can make it difficult to find errors.
  - Follow good software development practices to encourage proper use of the Navigator hardware blocks.
  - The Simulator catches several common configuration problems and will throw warnings to the debug log files.

For More Information

- For more information, refer to the to Multicore Navigator User Guide http://www.ti.com/lit/SPRUGR9
- For questions regarding topics covered in this training, visit the support forums at the TI E2E Community website.