KeyStone Training

Multicore Navigator: Queue Manager Subsystem (QMSS)

Agenda

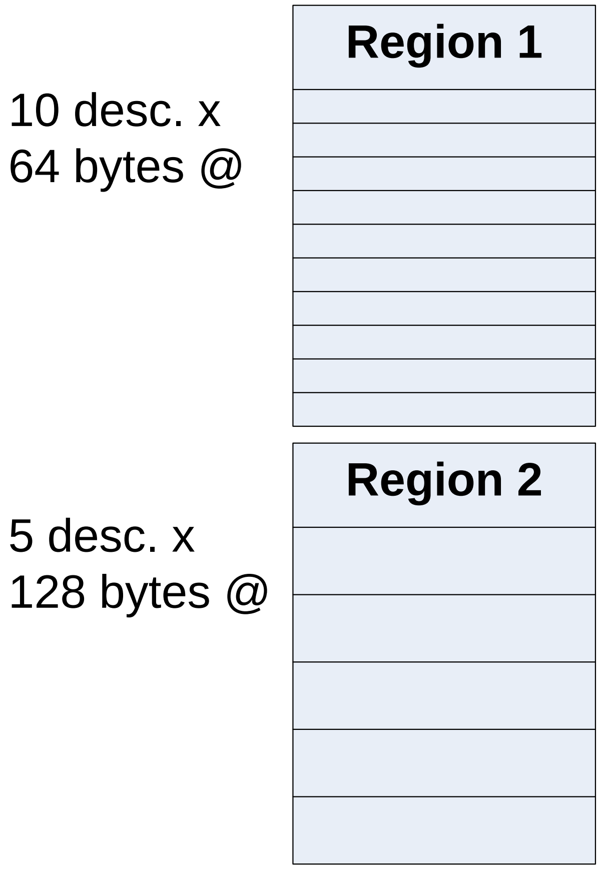
- How Does the Queue Manager Work?
  - Memory Regions
  - Link RAMs
  - Address/Index Conversion
  - Queue Pend Signals
  - Accumulators
- How to Program the QMSS Modules
  - Registers
  - Low Level Driver

How Does the Queue Manager Work?

- How Does the Queue Manager Work?
  - Memory Regions
  - Link RAMs
  - Address/Index Conversion
  - Queue Pend Signals
  - Accumulators
- How to Program the QMSS Modules
  - Registers
  - Low Level Driver

Memory Region Sizing

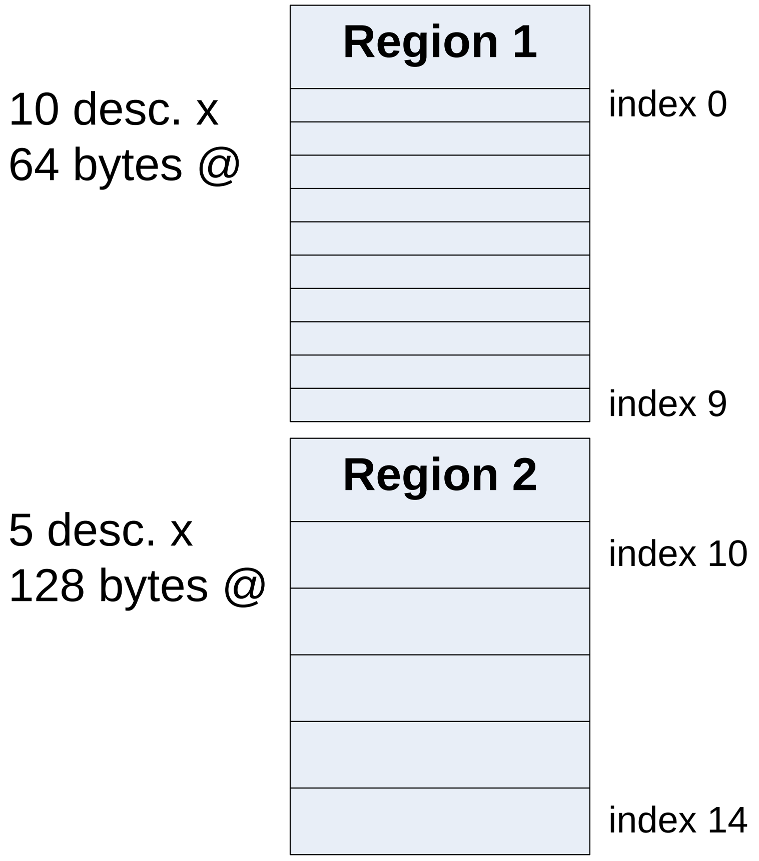
- All Navigator descriptor memory regions are divided into *equal sized* descriptors. For example:



Memory regions are *always* aligned to 16-byte boundaries and descriptors are *always* multiples of 16 bytes.

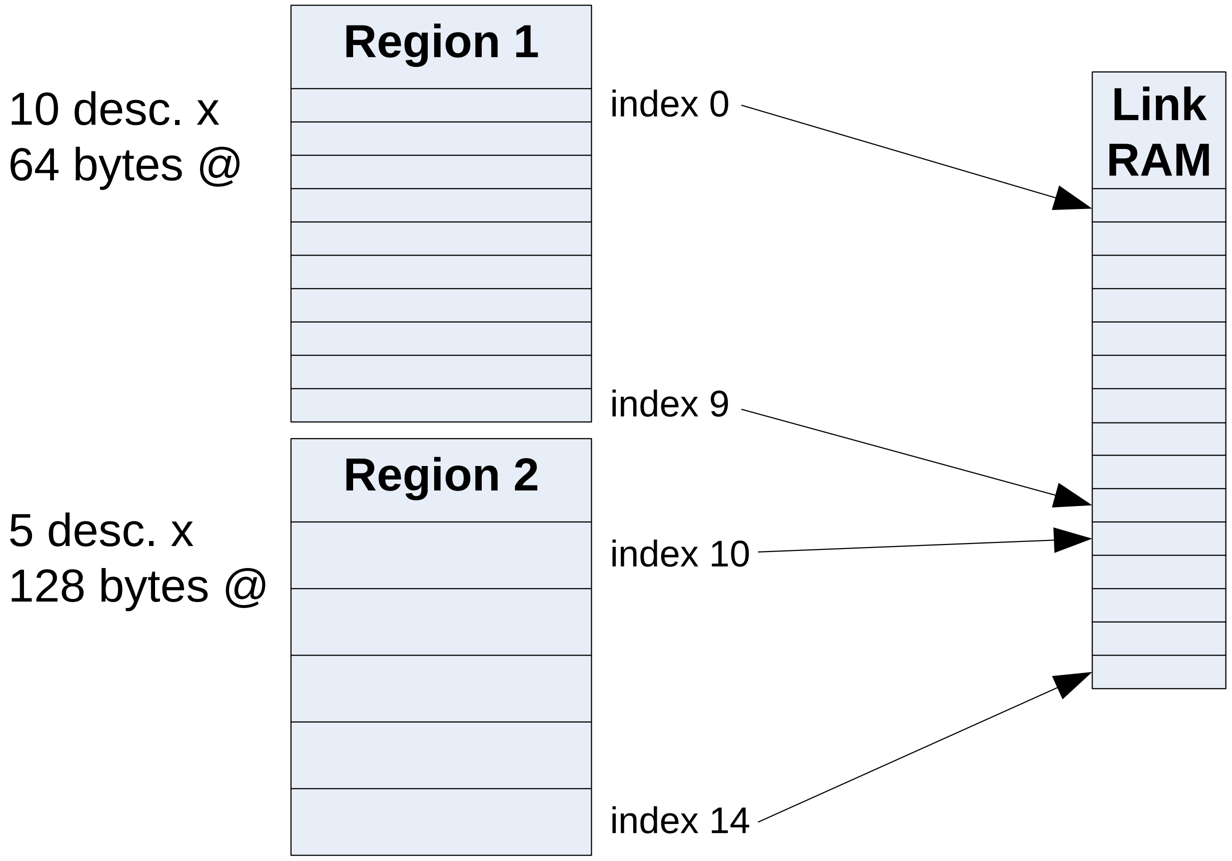
Memory Region Indexing

- Each memory region is given a unique range of indexes for its descriptors.



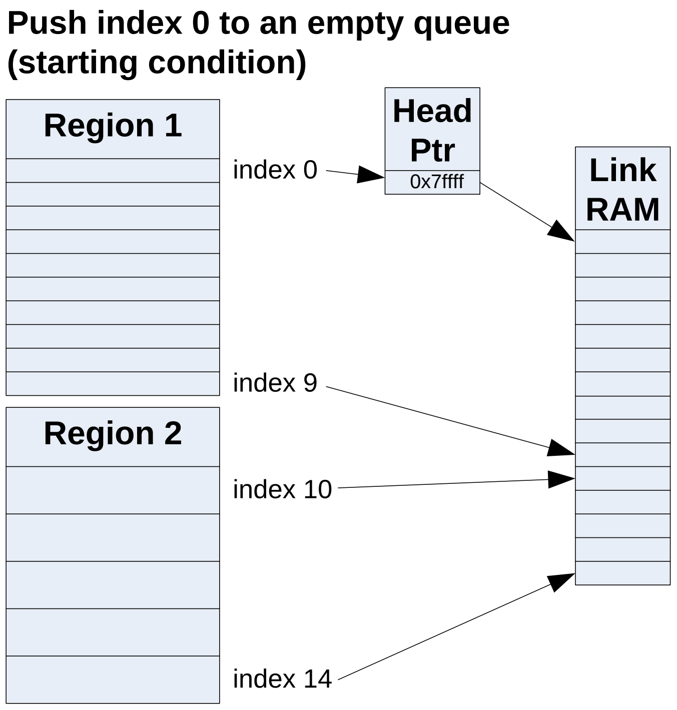
Memory Region Maps to Link RAM

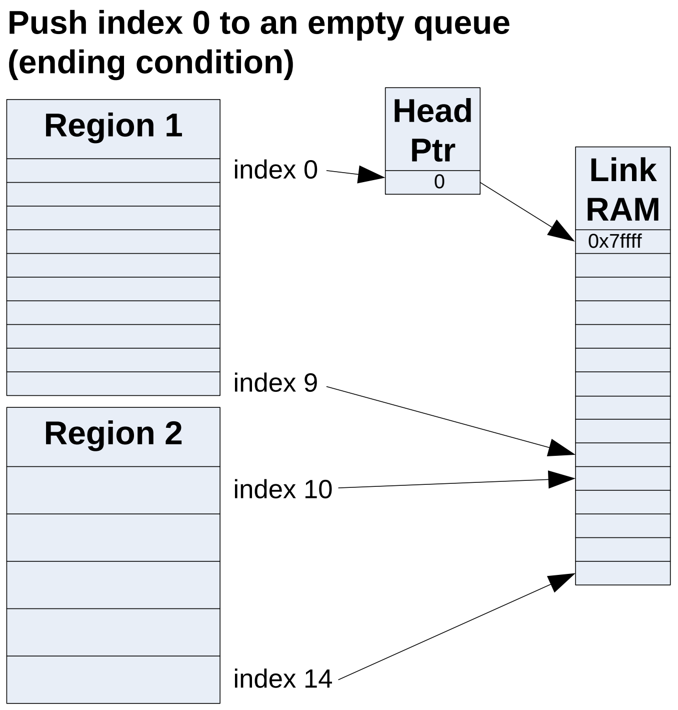
- Each descriptor index maps to the corresponding entry in the Link RAM (each Link RAM also has index range).



Queuing Example 1

- The Queue Manager maintains a head pointer for each queue, which are initialized to be empty.



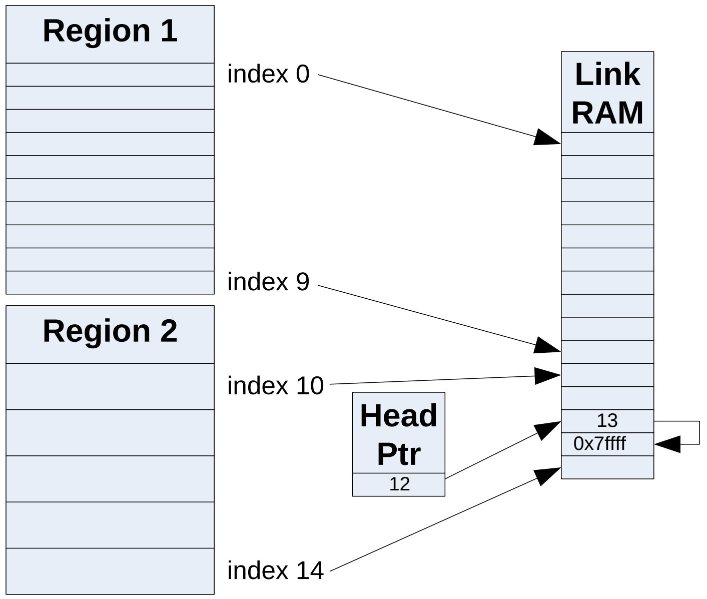


- We actually do not push indexes; We push descriptor addresses. The QM converts addresses to indexes.

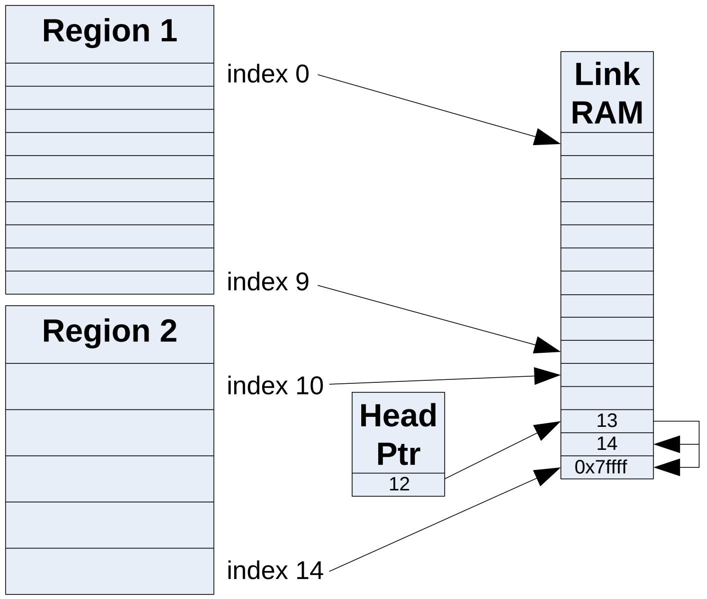
Queuing Example 2

- In this example, a descriptor is pushed into a queue that is not empty.

Push index 14 to a non-empty queue (starting condition)



Push index 14 to a non-empty queue (ending condition)



- Descriptors may be pushed to the head or tail, but are always popped from the head.

Address to Index Conversion

- When a descriptor is *pushed* onto a queue, the Queue Manager converts the address to an index:

  index = (address – region_base) / descriptor_size + region_first_index;

  – The descriptor is added to the queue by threading the indexed entry of the Link RAM into the queue's linked list.

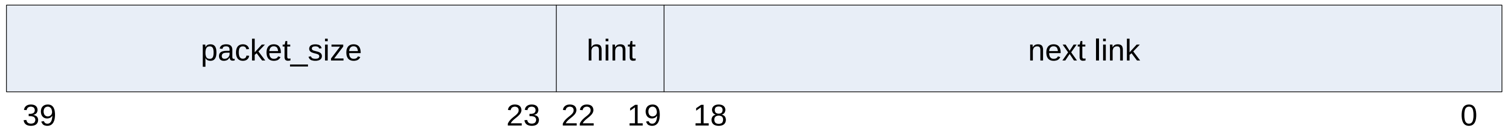
- When a queue is *popped*, the Queue Manager converts the index back into an address:

  address = (index – region_first_index) * descriptor_size + region_base;

  – The Link RAM is then rethreaded to remove this index.

Linking RAM Contents

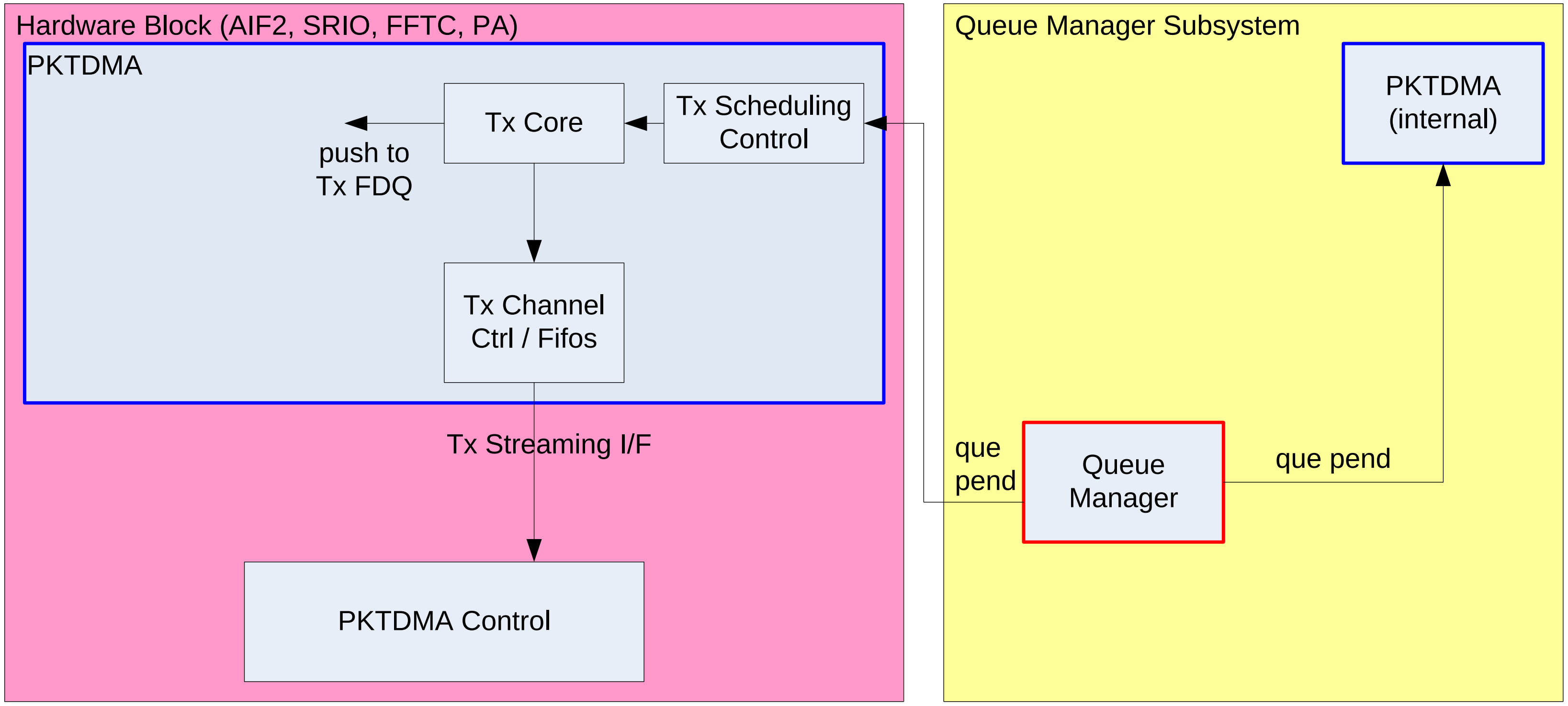
- So what is in a Linking RAM entry?
  - A link to the next index in the queue.
  - A hint field (used by the Tx DMA) -- The hint is taken from the four LSBs of the push value; It is a coded descriptor size used by the Tx DMA for prefetching the descriptor control header.
  - The packet size -- This is what was written to Que N Register C just prior to the push – it is not a calculated packet size (*in fact, the queue manager never touches descriptor memory*!).



  - The internal Link RAM is 16K 40-bit words.
  - An external Link RAM requires one 64-bit word per descriptor.

Queue Manager Control of Tx DMA

- One-to-one mapping of queue number to Tx channel for each Tx DMA in the system:
  - This queue drives a qpend signal to a specific Tx channel.
  - The Tx channel is triggered by the first push to the queue.
  - e.g.: Queue 672 maps to SRIO channel 0, 673 to channel 1...

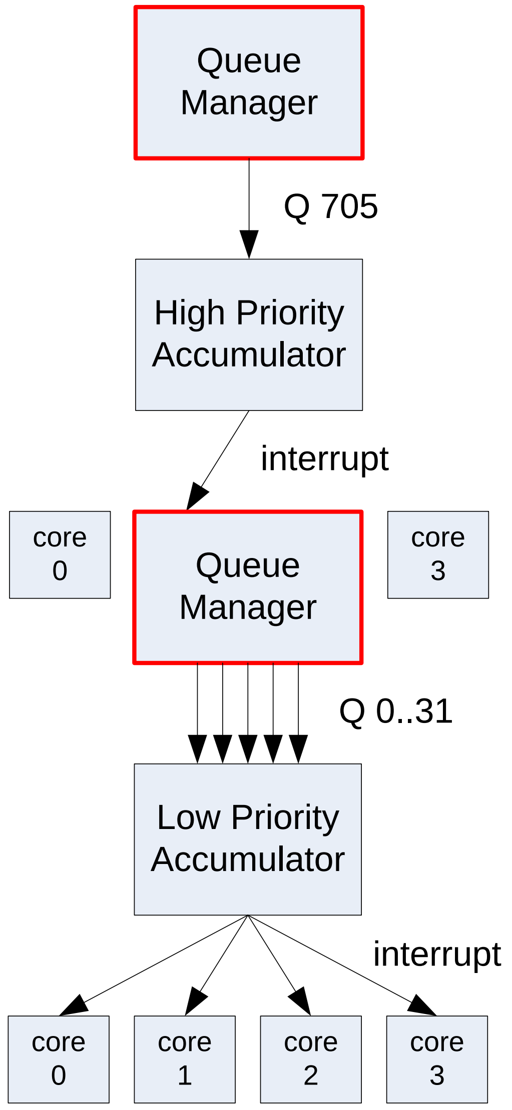


Descriptor Accumulators

- Firmware designed to keep DSPs from polling.
- Run in background, interrupts DSP with a list of descriptor addresses popped by the firmware.
- Host software must recycle these descriptors.

- High Priority Accumulator:
  - 32 channels, one queue per channel
  - All channels continuously scanned, 0 to 31
  - Average latency per scanning loop, ~7us
  - Each channel/event maps to one core
  - Programmable list size and options

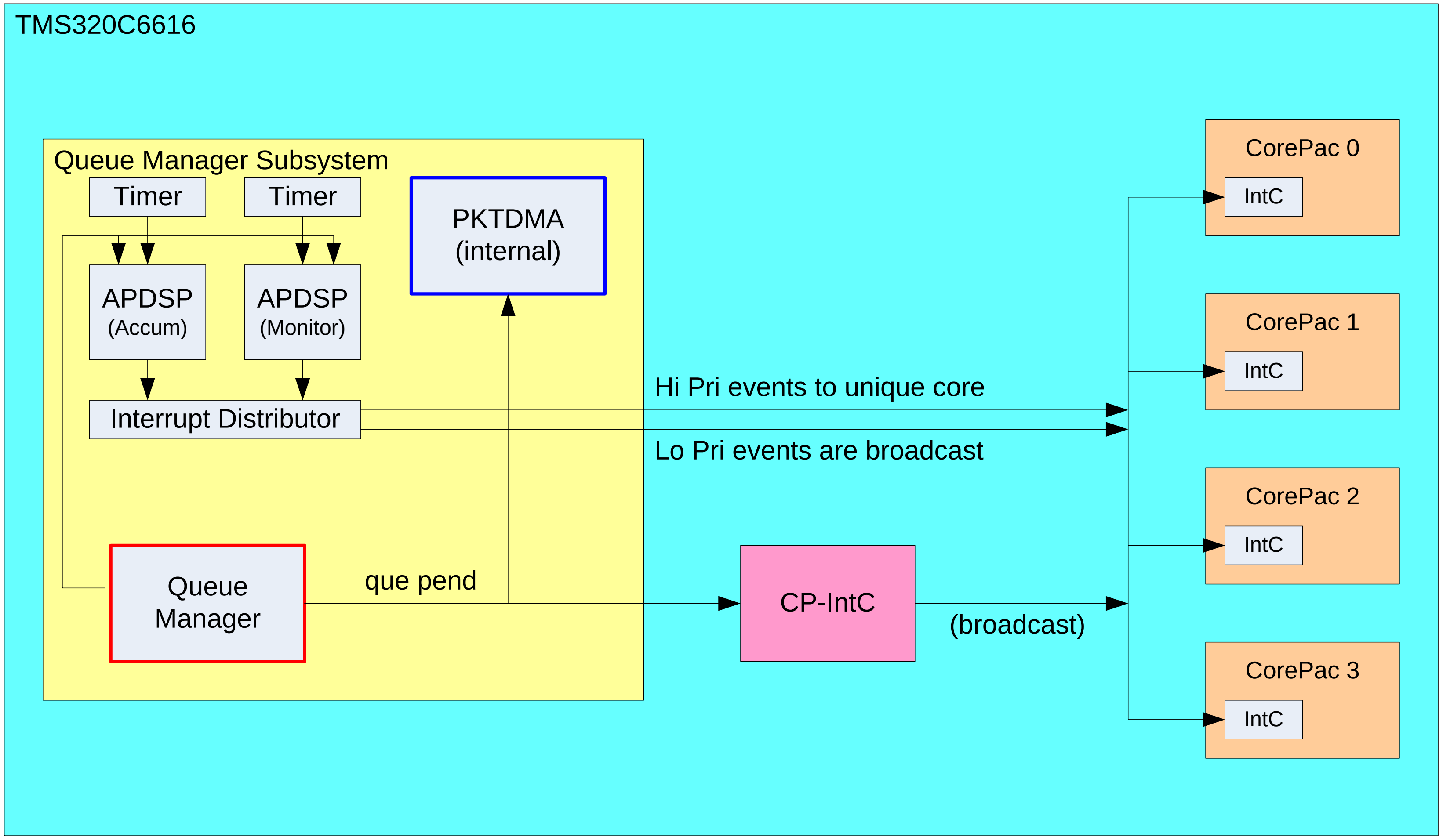
- Low Priority Accumulator:
  - 16 channels, up to 32 queues per channel
  - One channel scanned each pass through High Priority
  - Each channel/event maps to all cores
  - Programmable list size and options



QMSS Interrupt Generation

Two sources of QMSS interrupts:

- QMSS' INTD (Hi Priority / Lo Priority Accumulator interrupts)
- Queues with dedicated CP-IntC queue pend signals



How to Program the QMSS Modules

- How Does the Queue Manager Work?
  - Memory Regions
  - Link RAMs
  - Address/Index Conversion
  - Queue Pend Signals
  - Accumulators
- How to Program the QMSS Modules
  - Registers
  - Low Level Driver

Step 1: Memory Region Setup

- The most important part of Navigator initialization.
  - The configuration of other components depends on this.
  - It is critical to understand the memory movement requirements of the application!
- 20 memory regions – descriptor/payload storage.
- Descriptor size is multiple of 16 bytes, min 32.
- Descriptor count is a power of 2, minimum 2^5.
- Descriptors must be placed at programmed intervals.
- Each descriptor has a unique index within the region and within the Linking RAM.

Memory Region Registers

- Three registers must be programmed for each memory region configuration:
  - Memory Region R Base Address:
    - Must be a global address
    - Must be aligned to a 16-byte address boundary
  - Memory Region R Start Index:
    - Indicates the index of the first descriptor in this region.
  - Memory Region R Descriptor Setup:
    - Programs the size and count of descriptors.

Memory Region Use: Tips & Pitfalls

- Issue: Resolution of descriptor count becomes lower as count increases (since it is a power of two). (e.g. 2048, 4096 are available, but not 3000)
  - Resolution: Allocate memory for what is needed.
  - Caution! Never place one memory region within the programmed address range of another memory region! Link RAM corruption will result!
- Issue: Monolithic descriptor size is fixed.
  - Resolution: Use adjacent descriptors to create a larger descriptor if needed. (this is a software control issue)
  - Caution! Do not push the adjacent addresses! Data corruption will result! Rx DMA does not check size!

Step 2: Linking RAM Setup

- The Linking Ram is configured in one or two pieces, though it is used (by QM) as a single unit.
  - Requires one n-bit word per descriptor index, where n =
    - 40 bits for the internal linking RAM
      - Benefit: speed (much faster than any other memory)
    - 64 bits for an external linking RAM
      - Benefit: size (up to 512K entries)
  - Any global address can be used for the second Linking RAM.

Linking RAM Registers

- One or two registers must be programmed for each Linking RAM used:
  - Linking RAM Region N Base Address:
    - Must be a global address
    - To use the QMSS internal link RAM memory, Region 0 Base Address must be set to the internal address, which currently is 0x00080000.
  - Linking RAM Region 0 Size:
    - Indicates the number of descriptor indexes in the first link RAM.
      - If this register is set to 999, then Region 0 indexes will be 0 to 999; Region 1 indexes will start at 1000.
      - Set to 0x3FFF to use the entire internal Linking RAM
    - This register is not needed for the second linking RAM.

Step 3: Push and Pop

- For the QM, the only thing remaining is to push and pop descriptors (the remaining initialization tasks are to populate free descriptor queues and program PKTDMAs).

- Queue N Register D:
  - Writing to these registers <u>pushes</u> a descriptor into a queue
  - Reading these registers <u>pops</u> a descriptor from a queue

- Note: While a descriptor is linked in a queue it should be considered “owned” by hardware and not modified. When software pops it (or is given the address by the accumulator), it “owns” the descriptor, and may modify the contents until it is pushed once again into a queue.

QMSS Low Level Driver (LLD)

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.

QMSS LLD Initialization

- The following are one-time initialization routines to configure the LLD globally:
  - Qmss_init(parms, queue_mapping);
    - Configures Link RAM, # descriptors, queue mapping
    - May be called on one or all cores
  - Qmss_exit();
    - Deinitializes the QMSS LLD

QMSS Configuration

- More QMSS configuration:
  - Qmss_start( );
    - Called once on every core to initialize config parms on those cores.
    - Must be called immediately following Qmss_init()
  - Qmss_insertMemoryRegion(mem_parms);
    - Configures a single memory region.
    - Should be called with protection so that no other tasks or cores could simultaneously create an overlapping region.

QMSS LLD Queue Usage

- To allocate and release queues:
  - queue_handle = Qmss_queueOpen(type, que, *flag);
    - Once "open", the DSP may push and pop to the queue.
      - type refers to an enum (tx queue, general purpose, etc.).
      - que refers to the requested queue number.
      - flag is returned true if the queue is already allocated.
  - Qmss_queueClose(queue_handle);
    - Releases the handle preventing further use of the queue

Queue Push and Pop

- Queue management APIs:
  - Qmss_queuePushDesc(queue_handle, desc_ptr);
    - Pushes a descriptor address to the handle’s queue.
    - Other APIs are available for pushing sideband info as well.
  - desc_ptr = Qmss_queuePop(queue_handle);
    - Pops a descriptor address from the handle’s queue.
  - count = Qmss_getQueueEntryCount(queue_handle);
    - Returns the number of descriptors in the queue.

QMSS Accumulator

- The following functions are available to program, enable, and disable an accumulator:
  - Qmss_programAccumulator(type, *program);
    - Programs/enables one accumulator channel (high or low)
    - Setup of the ISR is done outside the LLD using INTC
  - Qmss_disableAccumulator(type, channel);
    - Disables one accumulator channel (high or low)

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.