IPI Command Queuing

- ALL devices support command queuing
  - Devices have a finite maximum command queue depth
  - Queue depth may be limited to 1

- Devices that support a queue depth of more than one support COMMAND STACKING
  - Queue depth is a readable attribute of the device
  - Command queue must be managed in the SYSTEM
Queue Location

-Queued systems maintain command queues across multiple queue depth devices

Both Systems Have a 10 Command Queue Active

Queue Command Types

- Commands queued in the device must be managed by the device
  - Multiple management forms exist
  - Forms differ on command execution order

- Five forms of queued commands are defined
  - Individual
  - Chained
  - Sequential
  - Ordered
  - Priority

- Different forms may be mixed in a sequence of commands
Individual Commands

- Individual Commands are the "Generic" form of queued commands
  - Each received command is independent of any other command
  - May be executed in any order
    - Order MAY be changed to optimize performance
  - No data dependencies exist within the command

Chained Commands

- Chained Commands are executed in FIFO order by the device
  - A bit in the command specifies chained operation
  - First command received without the chain bit set ends the command chain
  - Device cannot be accessed from other ports during chain
    - Implicit reserve at start of chain
    - Released at end of chain
  - An error in any command terminates the chain and any queued commands of that chain
Chained Commands

- Commands for other devices may be executed in parallel with the chained commands.
- If multiple devices operate from the same controller (IPI-Slave), commands to other devices may complete before commands in the chain complete.

Sequential Commands

- Sequential Commands are similar to chained commands.
  - Commands are executed in FIFO order.
  - A bit in the command specifies sequential operation.
  - First command received without the sequence bit set ends the command sequence.
  - Device CAN be accessed from other ports during sequence:
    - Reserved at start of each command.
    - Released at end of each command.
  - An error in any command terminates the sequence and any queued commands of that sequence.
Sequential Commands

- Sequential Commands are required to be executed in the order specified
  - Commands sequences may span multiple devices
  - Commands not in the sequence may execute in the middle of the sequence

Commands 2 and 3 On Device B Must Complete Before Command 4 On Device A

If Command 5 Is Not Part Of The Sequence It May Complete Before Commands 2-4

Ordered Commands

- Ordered Commands are similar to sequence commands
  - Commands are executed in FIFO order
  - A bit in the command specifies ordered operation
  - First command received without the order bit set ends the command order
  - Commands not in the order MAY NOT execute in the middle of the order
  - Device CAN be accessed from other ports during sequence
    - Reserved at start of each command
    - Released at end of each command
  - An error in any command terminates the order and any queued commands of that order
Ordered Commands

- Ordered Commands may span multiple devices

![Diagram showing host and slave controller with commands 1 to 4 and command 5 execution order]

Command 5 Must Execute After Command 4 Completes

Priority Commands

- Priority Commands are interrupts into the normal command flow
  - Also affect interpretation of commands
  - Terminate any Chain, Sequence or Order currently being sent

- Priority Commands are executed in LIFO order

- May also have Priority Chains, Sequences, and Orders
  - Only first command of Priority Chain, Sequence, or Order is marked as Priority
  - Each sequence marked as Priority is executed completely before returning to any previously received commands
**Priority Commands**

- Execution order is determined by how many commands have already been completed
  - May also be design dependent
  - Example show here is for all individual commands

<table>
<thead>
<tr>
<th>Commands As Sent</th>
<th>Execution Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY2</td>
<td>Command5</td>
</tr>
<tr>
<td>PRIORITY1</td>
<td>Command4</td>
</tr>
<tr>
<td>Command5</td>
<td>Command3</td>
</tr>
<tr>
<td>Command4</td>
<td>Command2</td>
</tr>
<tr>
<td>Command3</td>
<td>PRIORITY1</td>
</tr>
<tr>
<td>Command2</td>
<td>PRIORITY2</td>
</tr>
<tr>
<td>Command1</td>
<td>Command1</td>
</tr>
</tbody>
</table>

**Priority Commands**

- If a sequence is in execution, the sequence is allowed to complete first
  - Last command in sequence may be marked by the Priority Command
  - If command Chain1 had not yet started, both Priority commands would execute first

<table>
<thead>
<tr>
<th>Commands As Sent</th>
<th>Execution Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY2</td>
<td>Chain4</td>
</tr>
<tr>
<td>Chain4</td>
<td>Chain3</td>
</tr>
<tr>
<td>Chain3</td>
<td>PRIORITY1</td>
</tr>
<tr>
<td>PRIORITY1</td>
<td>Chain2</td>
</tr>
<tr>
<td>Chain2</td>
<td>Chain1</td>
</tr>
</tbody>
</table>

Interpreted as two separate Chains
### Priority Abort

- ABORT Command is used to
  - Remove commands from the queue
  - Kill currently running commands

- For ABORT to do ANYTHING it must be sent as a Priority Command
  - Specifies commands to kill by reference number
  - Suspends the executing command to process the ABORT

<table>
<thead>
<tr>
<th>Commands As Sent</th>
<th>Execution Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command4</td>
<td>Command4</td>
</tr>
<tr>
<td>ABORT1,3</td>
<td>Command3</td>
</tr>
<tr>
<td>Command3</td>
<td>Command2</td>
</tr>
<tr>
<td>Command2</td>
<td>Command1</td>
</tr>
<tr>
<td>Command1</td>
<td>ABORT1,3</td>
</tr>
</tbody>
</table>

### Command Handling Software

- Device software for a queued environment
  - Must be multitasking
  - Must not allow deadlocks
  - Must support timeout capabilities
  - Must allocate resources by command

- This requires a Real-Time kernel or executive
  - VRTX
  - PSOS
  - Etc.

- Maximum performance requires excellent software design as well as hardware
  - Minimize number of context switches per command
  - Remove excess kernel tasks
  - Use small/fast kernel with short interrupt latency