3.3.17 Connection Paced Resume Register [RegisterManager]

When software writes to this register, the connection is resumed in the ChainManager, and then resumed in the Pacer (if the connection’s bandwidth doesn’t violate the max APIC bw).

```
Kernel       User
0x0500 00E8  0x0700 00E8
0x0500 00EC  0x0700 00EC
```

R: Connection has been resumed as a Paced connection

3.3.18 Connection BestEffort Resume Register [RegisterManager]

When software writes to this register, the connection is resumed in the ChainMgr, then resumed as BestEffort in the Pacer.

```
Kernel       User
0x0500 00F0  0x0700 00F0
0x0500 00F4  0x0700 00F4
```

3.3.19 Access Mask Register [RegisterManager]

This register has bits set to allow user-level access to each of the registers for an RxConnection. Bit 0 controls register access to location 0x0700 0000, while bit 1 controls register access to location 0x0700 0008. With 32 bits, we can control access to locations from 0x0700 0000 to 0x0700 00F8. At boot-time, bit 31 (which controls access to this access mask register, location 0x0700 00F8) is cleared to disallow user access to the access mask register. XXX Zubin: there is a trade-off here between area used by the memory that holds these access mask registers and the ability to allow growth in the registers.

```
Kernel       User
0x0500 00F8  0x0700 00F8
0x0500 00FC  0x0700 00FC
```

AccessMask 32
3.3.13 Connection Pacing Parameter Register [Pacer]

This write-only register holds the pacing parameter for a connection. Zubin: we may have to split this to simplify the memory accesses in the Pacer (the integer and fractional parts of PacingParameter are stored in separate locations).

3.3.14 Connection Interrupt Control Register [IntrMgr]

This read-write register holds enable bits for each of the types of interrupts that can be generated by transmit connections.

3.3.15 Connection Notification Control Register [IntrMgr]

This read-write register holds enable bits for each of the types of notifications that can be generated by transmit connections.

3.3.16 Connection LowDelay Resume Register [RegisterManager]

When software writes to this register, the connection is resumed in the ChainManager, then resumed as low delay in the Pacer.
### 3.3.9 Connection Paced Resume Register [Pacer]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0500 0048 0x0700 0048</td>
<td></td>
</tr>
<tr>
<td>0x0500 004C 0x0700 004C</td>
<td></td>
</tr>
</tbody>
</table>

When software reads from this register, the connection is resumed (if the bandwidth doesn’t exceed APIC Pacing BW) and the APIC returns an S bit of 1.

When software writes to this register, the connection is resumed (if the bandwidth doesn’t exceed APIC Pacing BW).

### 3.3.10 Connection BestEffort Resume Register [Pacer]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0500 0050 0x0700 0050</td>
<td></td>
</tr>
<tr>
<td>0x0500 0054 0x0700 0054</td>
<td></td>
</tr>
</tbody>
</table>

When software writes to this register, the connection is resumed.

### 3.3.11 Connection Suspend Register [Pacer]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0500 0058 0x0700 0058</td>
<td></td>
</tr>
<tr>
<td>0x0500 005C 0x0700 005C</td>
<td></td>
</tr>
</tbody>
</table>

When software writes to this register, the connection is suspended. If the

### 3.3.12 Connection Pacing BW Register [Pacer]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0500 0060 0x0700 0060</td>
<td></td>
</tr>
<tr>
<td>0x0500 0064 0x0700 0064</td>
<td></td>
</tr>
</tbody>
</table>

This read/write register holds the pacing bandwidth and a copy of the connection’s state. The channel state portion of the register is read-only, and reflects the current connection state in the APIC.
3.3.5 **Current Descriptor Offset [ChainManager]**

This read/write register holds the offset of the current descriptor in the descriptor pool.

3.3.6 **Current Descriptor Part 1 Register [ChainManager]**

This read/write register holds the descriptor currently used by the connection. Read operations will return the actual descriptor, even if it is in use. Write operations will have no effect if the target connection is currently in use.

3.3.7 **Current Descriptor Part 2 Register [ChainManager]**

This read/write register holds the second half of the descriptor currently used by the connection. Read operations will return the actual descriptor, even if it is in use. Write operations will have no effect if the target connection is currently in use.

3.3.8 **Connection LowDelay Resume Register [Pacer]**

When software writes to this register, the connection is resumed as Low Delay.
3.3 Transmit Connection Registers

3.3.1 Connection Processing Register [ChainMgr]

This read-write register contains parameters that describe how the connection behaves. XXX Zubin: note that the Type field (lowdelay, paced, besteffort) has been removed. The Pacer now has 3 resume registers (ResumeLowDelay, ResumePaced, and ResumeBestEffort with appropriate mask bits in the mask register). This makes it more difficult for the software to determine the type of a connection by looking at the hardware state. Shouldn’t be a problem, as long as software keeps track of connection types anyway. True?

This read-write register holds the type bits for each connection.

3.3.2 Connection Command/Status Register [ChainMgr]

When software reads from this register, it finds an error code. An error code of 0 indicates that the connection is currently resumed and running. When software writes a value of 0 to this register, the connection is resumed. When software writes any other value, the connection is suspended.

3.3.3 Connection ATM Header [ChainMgr]

This read-write register contains the ATM Header for all cells sent on the connection.

3.3.4 Descriptor Pool Base [ChainManager]

This read/write register holds this connection’s offset into the descriptor pool.
3.2.8 Connection Interrupt Control Register [IntrMgr]

This read-write register holds enable bits for each of the types of interrupts that can be issued by channels.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0080</td>
<td>0x0600 0080</td>
<td>4 3 2 1 0</td>
</tr>
<tr>
<td>0x0400 0084</td>
<td>0x0600 0084</td>
<td></td>
</tr>
</tbody>
</table>

- 4: Done Interrupt Enable
- 3: Always Interrupt On Done
- 2: Descriptor Read Interrupt Enable
- 1: Error Interrupt Enable
- 0: Abort Interrupt Enable

3.2.9 Connection Notification Control Register [IntrMgr]

This read-write register holds enable bits for each of the types of notifications that can be issued by channels.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0088</td>
<td>0x0600 0088</td>
<td>4 3 2 1 0</td>
</tr>
<tr>
<td>0x0400 008C</td>
<td>0x0600 008C</td>
<td></td>
</tr>
</tbody>
</table>

- 4: Done Notification Enable
- 3: Always Notify On Done
- 2: Descriptor Read Notification Enable
- 1: Error Notification Enable
- 0: Abort Notification Enable

3.2.10 Connection Input Register [VCXT]

This read-write register contains the VCXT-specific information for each connection.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 00D0</td>
<td>0x0600 00D0</td>
<td>P L R O</td>
</tr>
<tr>
<td>0x0400 00D4</td>
<td>0x0600 00D4</td>
<td>MV1 MV0</td>
</tr>
</tbody>
</table>

- P: Connection Open
- L: Low Delay [NORMAL, LOWDELAY]
- R: RMAccept [DISCARD, ACCEPT]
- O: OAMAccept [DISCARD, ACCEPT]

3.2.11 Access Mask Register [RegisterManager]

This register has bits set to allow user-level access to each of the registers for an RxConnection. Bit 0 controls register access to location 0x0600 0000, while bit 1 controls register access to location 0x0600 0008. With 32 bits, we can control access to locations from 0x0600 0000 to 0x0600 00F8. At boot-time, bit 31 (which controls access to this access mask register, location 0x0600 00F8 is cleared to disallow user access to the access mask register. XXX Zubin: there is a trade-off here between area used by the memory that holds these access mask registers and the ability to allow growth in the registers.
### 3.2.5 Current Descriptor Offset [ChainManager]

This read/write register holds the offset of the current descriptor in the descriptor pool.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0020</td>
<td>0x0600 0020</td>
<td>CurrentDescOffset</td>
</tr>
<tr>
<td>0x0400 0024</td>
<td>0x0600 0024</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2.6 Current Descriptor Part 1 Register [ChainManager]

This read/write register holds the descriptor currently used by the connection. Read operations will return the actual descriptor, even if it is in use. Write operations will have no effect if the target connection is currently in use.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>Preserved</th>
<th>BufferLen</th>
<th>NextDescriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0028</td>
<td>0x0600 0028</td>
<td>V</td>
<td>I</td>
<td>S</td>
</tr>
<tr>
<td>0x0400 002c</td>
<td>0x0600 002c</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **V**: Buffer is Volatile (don’t use MWI transaction on bus)
- **I**: Interrupt/Notify on Descriptor Read
- **S**: SAM Enable
- **O**: Read Only Descriptor (don’t write back)
- **E**: End of Frame in Buffer
- **C**: CRC OK
- **L**: Loss Priority (= CLP of last cell)
- **X**: Congestion Indication (from last cell’s PTI)
- **T**: Buffer Type (DATA, RM, SEGOAM, E2EOAM)
- **Y**: Descriptor Sync (DoneValidLink, DoneInvalidLink, NotReady, Ready)

### 3.2.7 Current Descriptor Part 2 Register [ChainManager]

This read/write register holds the second half of the descriptor currently used by the connection. Read operations will return the actual descriptor, even if it is in use. Write operations will have no effect if the target connection is currently in use.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>BufferAddrLo32</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0030</td>
<td>0x0600 0030</td>
<td>hi4</td>
</tr>
<tr>
<td>0x0400 0034</td>
<td>0x0600 0034</td>
<td></td>
</tr>
</tbody>
</table>

RegisterManagerDesign.mif
3.2 Receive Connection Registers

Two address spaces are used to access Rx Connection registers. The kernel address space is compacted to minimize the number of pages allocated to APIC at boot time, while the user address space is spread so that each connection occupies a single memory page to allow for inter-process memory protection.

### 3.2.1 Connection Processing Register [ChainMgr]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0000 0x0600 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HeaderLen</td>
<td></td>
</tr>
<tr>
<td>0x0400 0004 0x0600 0004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This read-write register holds the type bits for each connection.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0008 0x0600 0008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0400 000C 0x0600 000C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When software reads from this register, it finds an error code. An error code of 0 indicates that the connection is currently resumed and running. When software writes a value of 0 to this register, the connection is resumed. When software writes any other value, the connection is suspended.

### 3.2.3 Frame Status Register [ChainMgr]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0010 0x0600 0010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CurHdrLen</td>
<td></td>
</tr>
<tr>
<td>0x0400 0014 0x0600 0014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This read-only register holds the status of the current frame.

### 3.2.4 Descriptor Pool Base [ChainManager]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0400 0018 0x0600 0018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0400 001C 0x0600 001C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This read/write register holds this connection’s offset into the descriptor pool.
### 3.1.39 Global Notification Enable Register [IntrManager]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>Bits 3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0528 0x---- ----</td>
<td></td>
<td>V E T nfyStatus</td>
</tr>
<tr>
<td>0x0000 052C 0x---- ----</td>
<td></td>
<td>CID</td>
</tr>
</tbody>
</table>

- **Bit 3**: Done Notification Enable
- **Bit 2**: Descriptor Read Notification Enable
- **Bit 1**: Error Notification Enable
- **Bit 0**: Abort Notification Enable

This read-write register holds bits to indicate which events are capable of generating a notification to the processor. All notifications should be enabled in both the global notification enable register as well as in the per-channel notification control register in order for an event on a channel to result in a new notification being added to the notification list.

### 3.1.40 Notification Register [IntrManager]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>Bits 3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0530 0x---- ----</td>
<td></td>
<td>V E T nfyStatus</td>
</tr>
<tr>
<td>0x0000 0534 0x---- ----</td>
<td></td>
<td>CID</td>
</tr>
</tbody>
</table>

- **V**: Valid
- **E**: End of Notification List
- **T**: Tx/Rx
- **Bits 16-19**: nfyStatus
  - **Bit 19**: Done Notification
  - **Bit 18**: Descriptor Read Notification
  - **Bit 17**: Error Notification
  - **Bit 16**: Abort Notification

This read-only register holds information about the head of the notification list. The information in this register is a valid notification only if the valid bit is set. If the valid bit is cleared, the notification list is empty. For valid notifications that have been returned, the “end of notification list” bit, if set, tells the processor that there are no more entries in the notification list.
3.1.36 Interrupt Status Register [IntrManager]

This read-only register holds bits to indicate the type of interrupt that occurred.

3.1.37 Interrupt Acknowledge Register [IntrManager]

When this read-only register is read, the Interrupt Status Register (above) is returned and all bits are cleared, which effectively acknowledges the interrupt.

3.1.38 Global Interrupt Enable Register [IntrManager]

This read-write register holds bits to indicate which interrupts are capable of generating an interrupt to the processor. With the exception of external interrupts, all other interrupts should be enabled in both the global interrupt enable register as well as in the per-channel interrupt control register in order for an event on a channel to result in a real interrupt.
3.1.31 Pool Chain 3 Head Pointer [ChainManager]

```
Kernel   User
0x0000 0418 0x---- ----
0x0000 041c 0x---- ----
```

This read/write register contains the base address of the descriptor area. The address is 36 bits long.

3.1.32 Descriptor Area Pointer Register [ChainManager]

```
Kernel   User
0x0000 0420 0x---- ----
0x0000 0424 0x---- ----
```

This read/write register contains the base address of the descriptor area. The address is 36 bits long.

3.1.33 Control Cell Sequence Number [ChainManager]

```
Kernel   User
0x0000 0428 0x---- ----
0x0000 042c 0x---- ----
```

This read-only register holds a single bit to indicate the sequence number (0 or 1) of the last control cell received.

3.1.34 Interrupt Wait Time [IntrManager]

```
Kernel   User
0x0000 0500 0x---- ----
0x0000 0504 0x---- ----
```

This read/write register contains the time the APIC should wait between presenting interrupts to the processor.

3.1.35 Remote Interrupt Register [IntrManager]

```
Kernel   User
0x0000 0508 0x---- ----
0x0000 050c 0x---- ----
```

This read/write register holds a single bit that indicates whether interrupt cells should be generated in response to an interrupt, and the CID to resume for remote interrupts.
### 3.1.27 Global RM Routing Register [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>3</th>
<th>3</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0370</td>
<td>0x---- -----</td>
<td></td>
<td></td>
<td></td>
<td>MV1 MV0</td>
</tr>
<tr>
<td>0x0000 0374</td>
<td>0x---- -----</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

This register indicates whether RM cells on receive connections should be redirected to the ports specified in R1 or R0.

- **0**: InPort 0 Alternate RM Routing Enable
- **1**: InPort 1 Alternate RM Routing Enable
- **MV0**: RM Multicast Vector for InPort 0
- **MV1**: RM Multicast Vector for InPort 1

### 3.1.28 Pool Chain 0 Head Pointer [ChainManager]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0400</td>
<td>0x---- -----</td>
<td></td>
</tr>
<tr>
<td>0x0000 0404</td>
<td>0x---- -----</td>
<td></td>
</tr>
</tbody>
</table>

This read/write register contains the offset for the head of the descriptor chain for pool 0.

### 3.1.29 Pool Chain 1 Head Pointer [ChainManager]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0408</td>
<td>0x---- -----</td>
<td></td>
</tr>
<tr>
<td>0x0000 040c</td>
<td>0x---- -----</td>
<td></td>
</tr>
</tbody>
</table>

This read/write register contains the offset for the head of the descriptor chain for pool 1.

### 3.1.30 Pool Chain 2 Head Pointer [ChainManager]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0410</td>
<td>0x---- -----</td>
<td></td>
</tr>
<tr>
<td>0x0000 0414</td>
<td>0x---- -----</td>
<td></td>
</tr>
</tbody>
</table>

This read/write register contains the offset for the head of the descriptor chain for pool 2.
3.1.23 Dropped Cell Count, BusPort [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0350</td>
<td>0x----- ----</td>
</tr>
<tr>
<td>0x0000 0354</td>
<td>0x----- ----</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.

3.1.24 Clock Estimator Register [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0358</td>
<td>0x----- ----</td>
</tr>
<tr>
<td>0x0000 035C</td>
<td>0x----- ----</td>
</tr>
</tbody>
</table>

This read-only register gives an indication of the internal clock speed relative to the bus clock speed. This estimator indicates how many internal clock ticks are equal to 8192 bus clock ticks. Armed with knowledge of the APIC’s internal clock rate (printed on the card), the software can determine the bus clock rate to within a fraction of a percent.

3.1.25 HEC Dropped Cell Count, Port 0 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0360</td>
<td>0x----- ----</td>
</tr>
<tr>
<td>0x0000 0364</td>
<td>0x----- ----</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.

3.1.26 HEC Dropped Cell Count, Port 1 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0368</td>
<td>0x----- ----</td>
</tr>
<tr>
<td>0x0000 036C</td>
<td>0x----- ----</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.
3.1.18 Dropped Cell Count, InputPort 0 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 032c 0x----- ----</td>
<td>32</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.

3.1.19 Valid Cell Count, InputPort 1 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0330 0x----- ----</td>
<td>32</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.

3.1.20 Transit Cell Count, InputPort 1 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0338 0x----- ----</td>
<td>32</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.

3.1.21 Dropped Cell Count, InputPort 1 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0340 0x----- ----</td>
<td>32</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.

3.1.22 Valid Cell Count, BusPort [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0348 0x----- ----</td>
<td>32</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.
### 3.1.13 Port Enable Register [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0300</td>
<td>0x---- ----</td>
</tr>
<tr>
<td>0x0000 0304</td>
<td>0x---- ----</td>
</tr>
</tbody>
</table>

This read/write register is held in the VCXT and is used to enable/disable the UTOPIA inputs and outputs.

#### Port Enable Register

- **i0**: port 0 input enable
- **i1**: port 1 input enable

---

### 3.1.14 Flow Control Threshold Register [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0308</td>
<td>0x---- ----</td>
</tr>
<tr>
<td>0x0000 030C</td>
<td>0x---- ----</td>
</tr>
</tbody>
</table>

This read/write VCXT register holds the threshold at which the CellStore should assert flow control.

---

### 3.1.15 Grant Threshold Register [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0310</td>
<td>0x---- ----</td>
</tr>
<tr>
<td>0x0000 0314</td>
<td>0x---- ----</td>
</tr>
</tbody>
</table>

This read/write VCXT register holds the thresholds at which the CellStore should stop giving grants for each of the four traffic types.

---

### 3.1.16 Valid Cell Count, InputPort 0 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0318</td>
<td>0x---- ----</td>
</tr>
<tr>
<td>0x0000 031C</td>
<td>0x---- ----</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.

---

### 3.1.17 Transit Cell Count, InputPort 0 [VCXT]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0320</td>
<td>0x---- ----</td>
</tr>
<tr>
<td>0x0000 0324</td>
<td>0x---- ----</td>
</tr>
</tbody>
</table>

On read, this register returns its value and resets the count to zero. On write, the register is set to the desired value. The counter does not wrap, but sticks at the all-1s value.
3.1.8 APIC Current BW Register [Pacer]

This read-only register holds the bandwidth of all currently active paced connections. Each paced connection that attempts to resume must have BW less than or equal to APICPacingBW-APICCurrentBW.

3.1.9 APIC PacedTime Register [Pacer]

This read/write register allows the tester to advance the current pacedTime in the heap. This register should be used only during testing and verification, as its behavior is undefined during real APIC operation.

3.1.10 APIC UnPacedTime Register [Pacer]

This read/write register allows the tester to advance the current UnPacedTime in the Pacer Throttle. This register should be used only during testing and verification, as its behavior is undefined during real APIC operation.

3.1.11 APIC ExpireTime Register [Pacer]

This read/write register allows the tester to advance the current APICExpireTime in the Pacer Throttle. This register should be used only during testing and verification, as its behavior is undefined during real APIC operation.

3.1.12 APIC LastExpireTime Register [Pacer]

This read/write register allows the tester to advance the current APIC lastExpireTime in the Pacer Throttle. This register should be used only during testing and verification, as its behavior is undefined during real APIC operation.
3.1.4 Big Endian Data [RegisterManager]

This read/write register is a single bit that indicates whether DMA accesses by APIC will be big-endian or little-endian. This allows the APIC to utilize buses with different endianness. On MBus, accesses are typically big-endian (lowest address on the left hand side of the bus), while on PCI, accesses be either big- or little-endian, depending on the processor configuration.

This register is drawn here as 32 bits wide, but is really just a single bit in the APIC. When writing this register, remember to clear or set all of the bits, as they’ll be ORed together before storing in the APIC. This allows the endianness of DMA accesses by APIC to be set, regardless of the endianness of register accesses (see above).

3.1.5 Big Endian Descriptors [RegisterManager]

This read/write register is a single bit that indicates whether Descriptor accesses by APIC will be big-endian or little-endian. This allows the APIC to utilize buses with different endianness. On MBus, accesses are typically big-endian (lowest address on the left hand side of the bus), while on PCI, accesses be either big- or little-endian, depending on the processor configuration.

This register is drawn here as 32 bits wide, but is really just a single bit in the APIC. When writing this register, remember to clear or set all of the bits, as they’ll be ORed together before storing in the APIC. This allows the endianness of Descriptor accesses by APIC to be set, regardless of the endianness of register accesses (see above).

3.1.6 APIC Pacing BW Register [Pacer]

This read/write register holds the maximum bandwidth allowed for paced transmit traffic in the APIC.

3.1.7 APIC Pacing Parameter Register [Pacer]

This read/write register holds the APIC Pacing Parameter.
3.1.1 APIC Hardware Status/Reset Register [Register Manager]

<table>
<thead>
<tr>
<th>Kernel Address</th>
<th>User Address</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0000</td>
<td>0x0000</td>
<td>BT</td>
<td>U1</td>
<td>U0</td>
<td>version</td>
<td>match</td>
</tr>
<tr>
<td>0x0000 0004</td>
<td>0x0004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: Writing to this read-only register resets the APIC.

3.1.2 APIC ID Register [RegisterManager]

<table>
<thead>
<tr>
<th>Kernel Address</th>
<th>User Address</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0008</td>
<td>0x0008</td>
<td>APIC ID</td>
</tr>
<tr>
<td>0x0000 000C</td>
<td>0x000C</td>
<td></td>
</tr>
</tbody>
</table>

This read-only register holds the APIC ID, as set by pins.

3.1.3 Big Endian Register Accesses [RegisterManager]

<table>
<thead>
<tr>
<th>Kernel Address</th>
<th>User Address</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0010</td>
<td>0x0010</td>
<td>BigEndianRegs</td>
</tr>
<tr>
<td>0x0000 0014</td>
<td>0x0014</td>
<td></td>
</tr>
</tbody>
</table>

This read/write register is a single bit that indicates whether register access data will be big-endian or little-endian. All registers in this document are shown in little-endian format (lowest address is on the right, where the least-significant bit of multibit registers lies). To maximize performance of the software, the APIC allows these registers to be byte-swapped when they are moved to or from the bus. When this register is set, the lowest addressed byte will appear on the highest bit numbers of the bus (typically drawn as the left hand side of the bus).

Note that for registers that span 32 bit boundaries (like the descriptor area base address), no endianness swapping is done, as all register accesses MUST be exactly 32 bits. 64 bit register accesses have no meaning in the APIC.

When setting or clearing this register, software must set or clear all bits. All bits are ORed into a single bit to minimize storage in the APIC. This allows the software to set or clear the bit, regardless of the current endianness of register accesses.

Prediction: For MBus, we’ll set this bit. For PCI buses on Intel machines, we’ll clear this bit. For PCI buses on other machines, we’ll set this bit.
The register address map for the APIC is as follows:

The global configuration registers are mapped into the first 1536 bytes of the APIC address space. The memory page allocated in the kernel to this space is protected from access by any user process.

The RxConnection registers are mapped twice. Once for kernel mode accesses at 0x0400 0000 in a page that is mapped so that only the kernel can access it, and once for user mode accesses at 0x0600 0000 with one page per connection to allow per-user protection.

The TxConnection registers are similarly mapped at addresses 0x0500 0000 and 0x0700 0000.

Note that the APIC currently requires a 128MByte address space.

### 3.1 Global Configuration/Status Registers

The lowest 1536 bytes in the APIC address space contains global configuration and status registers:

```plaintext
<table>
<thead>
<tr>
<th>APICAddr</th>
<th>Global Registers</th>
<th>1536 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0000</td>
<td>0x0000 05FF</td>
<td>256 bytes</td>
</tr>
<tr>
<td>0x0400 0000</td>
<td>0x0400 FFFF</td>
<td>256 bytes/conn</td>
</tr>
<tr>
<td>0x0500 0000</td>
<td>0x0500 FFFF</td>
<td>256 bytes/conn</td>
</tr>
<tr>
<td>0x0600 0000</td>
<td>0x06FF FFFF</td>
<td>64K bytes/conn</td>
</tr>
<tr>
<td>0x0700 0000</td>
<td>0x07FF FFFF</td>
<td>64K bytes/conn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>APICAddr</th>
<th>RegisterManager</th>
<th>256 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0000</td>
<td>0x0000 00FF</td>
<td>256 bytes</td>
</tr>
<tr>
<td>0x0000 0100</td>
<td>0x0000 01FF</td>
<td>256 bytes</td>
</tr>
<tr>
<td>0x0000 0200</td>
<td>0x0000 02FF</td>
<td>256 bytes</td>
</tr>
<tr>
<td>0x0000 0300</td>
<td>0x0000 03FF</td>
<td>256 bytes</td>
</tr>
<tr>
<td>0x0000 0400</td>
<td>0x0000 04FF</td>
<td>256 bytes</td>
</tr>
<tr>
<td>0x0000 0500</td>
<td>0x0000 05FF</td>
<td>256 bytes</td>
</tr>
</tbody>
</table>
```
2 RegisterManager Requirements

The RegisterManager takes register accesses from the BusInterface slave device and passes them to the appropriate block in the APIC. In addition, the RegisterManager holds per-connection access masks that void any user space writes to specific per-connection registers.

3 APIC Address Space

The address space of APIC is divided into three parts: Global Registers, Transmit Connection Registers, and Receive Connection Registers. In addition, the Tx and Rx connection registers are duplicated for user and kernel space accesses.

This section describes the registers found in each of these classes. Each register has an address at which it can be accessed by the kernel. The Tx and Rx connection registers also have an address at which a user space process can access the register (if it isn’t masked).

The APIC uses the upper 9 bits of a 36 bit MBus address (5 bits of a 32 bit PCI bus address) to generate an internal chip select. The remaining 27 bits are used to determine the register being accessed inside the APIC:

Global Regs:

```
<table>
<thead>
<tr>
<th>CSHi</th>
<th>CSLo</th>
<th>00</th>
<th>xxx</th>
<th>RegID</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>14</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Conn. Regs, Kernel:

```
<table>
<thead>
<tr>
<th>CSHi</th>
<th>CSLo</th>
<th>txt</th>
<th>xxx</th>
<th>CID</th>
<th>RegID</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Conn. Regs, User:

```
<table>
<thead>
<tr>
<th>CSHi</th>
<th>CSLo</th>
<th>txt</th>
<th>CID</th>
<th>xxx</th>
<th>RegID</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
```

When the CSHi and CSLo fields match the upper 9 bits of an address on the bus (5 bits for PCI), the APIC claims the transaction and begins a register access.
1 Change Log

- Removed Pool Empty Bits
- Moved RxConnMgr registers into the ChainMgr
- Removed Rx Partial CRC Register
- Changed Ready, Done bits to NextDesc Invalid, APICOwnsDesc
- Moved Rx Intr/Notify Enable bits from ChainMgr to IntrMgr
- Moved TxConnMgr registers into the ChainMgr
- Removed Tx Partial CRC Register
- Removed Connection Expire Time register from Pacer
- Made all other per-connection Pacer registers write-only
- Moved Rx Intr/Notify Enable bits from ChainMgr to IntrMgr
- Changed ChainManager global registers to Zubin’s new location.
- Decided on new register map for per-connection registers.
- Deleted output port enable bits in VCXT regs
- Added VCXT statistics registers
- Added BigEndianDescriptors register.
- Rearranged ChainManager registers to fit the latest revision of Zubin’s code.
- Added TxConnection resume registers to the RegMgr so it can perform Pacer+ChainMgr resume itself.
- Zubin updated interrupt and notification registers, and current descriptor registers.
- Added Volatile bit to descriptors.
- Added HEC dropped cell count registers to the VCXT.
- Added UnPacedTime, ExpireTime, and LastExpireTime for the Pacer Throttle.