5 Flow Control

The Previous Chapter

The previous chapter described the use of HyperTransport *control* and *data* packets to construct HyperTransport link transactions. Control packet types include Information, Request, and Response variants; data packets contain a payload of 0-64 valid bytes. The transmission, structure, and use of each packet type is presented.

This Chapter

This chapter describes HyperTransport *flow control*, used to throttle the movement of packets across each link interface. On a high-performance connection such as HyperTransport, efficient management of transaction flow is nearly as important as the raw bandwidth made possible by clock speed and data bus width. Topics covered here include background information on bus flow control and the initialization and use of the HyperTransport virtual channel flow control buffer mechanism defined for each transmitter-receiver pair.

The Next Chapter

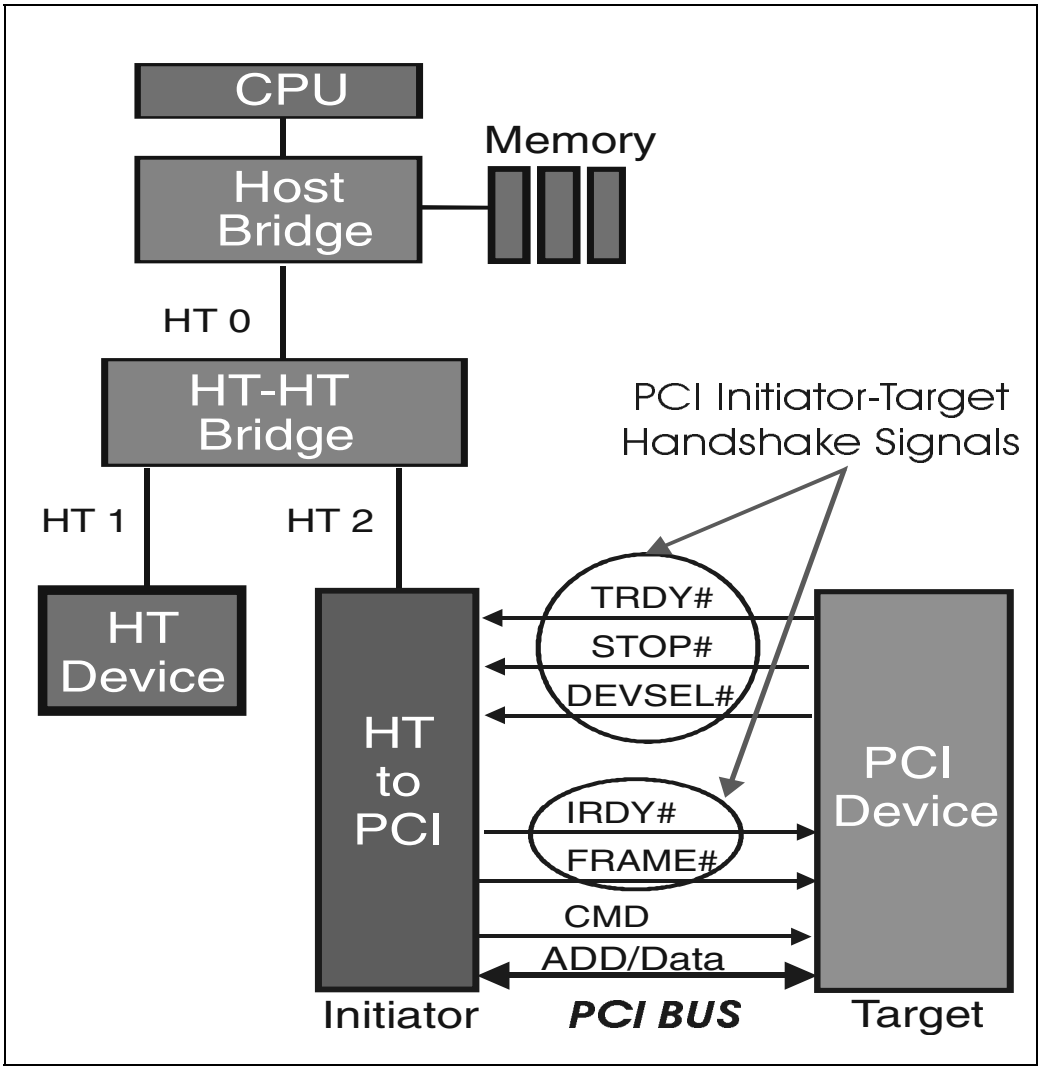
The next chapter describes the rules governing acceptance, forwarding, and rejection of packets seen by HyperTransport devices. Several factors come into play in routing, including the packet type, the direction it is moving, and the device type which sees it. A related topic also covered in this chapter is the fairness algorithm used by a tunnel device as it inserts its own packets into the traffic it forwards upstream on behalf of devices below it. The HyperTransport specification provides a fairness algorithm and a hardware method for tunnel management packet insertion.

The Problem

On any bus where an agent initiates the exchange of information (commands, data, status, etc.) with a *target*, a number of things can cause a delay (or even end) the normal completion of the intended transfer. The throttling of information delivery on a bus is referred to as *flow control*. PCI is a good example of a

bus protocol which has reasonably high burst bandwidth, but is subject to performance hits caused by an unsophisticated flow control mechanism. Before looking at the HyperTransport approach to flow control, some of the general problems in bus flow control are described in the following section in terms of the PCI protocol. Refer to Figure 5-1 on page 100.

Figure 5-1: PCI Interface Handshake Signals



How PCI Handles Flow Control

While the PCI specification permits 64-bit data bus and 66MHz clock options, a generic PCI bus carries only 32 bits (4 bytes) of data and runs at a 33MHz clock speed. This means that the burst bandwidth for this bus is 132MB/s (4 bytes x 33MHz = 132MB/s). In many systems the PCI bus is populated by all sorts of high- and low-performance peripherals such as hard drives, graphics adapters, and serial port adapters. All PCI bus master devices must take turns accessing the shared bus and performing their transfers. The priority of a bus master in accessing the bus and the amount of time it is allowed to retain control of the bus is a function of PCI *arbitration*. In a typical computer system, the PCI arbiter logic resides in the system chipset.

Once a PCI bus master has won arbitration and verifies the bus is idle, it commences its transaction. After decoding the address and command sent by the master, one target claims the cycle by asserting a signal called DEVSEL#. At this point, if both devices are prepared, either *write data* will be sent by the initiator or *read data* will be returned by the target. For cases where either the master or target are not prepared for full-speed transfer of some or all of the data, flow control comes into play. In PCI there are a number of cases that must be dealt with.

PCI Target Flow Control Problems

PCI Target Not Ready To Start. In some cases, a PCI device being targeted for transmission is not prepared to transfer any data at all. This could happen if the target is off-line, does not have buffer space for write data being sent to it, or does not have requested read data available. It may also occur if the transaction must cross a bridge device to a different bus. Many bus protocols, including PCI, place a limit on how long the bus may be stalled before completing a transaction; in cases where a target can't meet the requirement for even the first data, a mechanism is required to indicate the transaction should be abandoned and re-attempted later. PCI calls the target cancellation of a transaction (without transferring any data) a *Retry*; a Retry is indicated when a target asserts the STOP# signal (instead if TRDY#) in the first data phase.

PCI Target Starts Data Transfer, But Can't Continue. Another possibility is that a transaction started properly, some data has transferred, but at some point before completion the target "realizes" it can't continue the transfer within the time allowed by the protocol. The target must indicate to the master that the transaction must be suspended (and resumed later at the point where it

left off). PCI calls this target suspension of a transaction (with a partial transfer of data) a *Disconnect.* A Disconnect is signalled when the target asserts the STOP# signal in a data phase after the first one.

PCI Target Starts, Can Continue, But Needs More Time. Sometimes a transaction is underway and the target requires additional time to complete transmission of a particular data item; in this case, it does not need to suspend the transaction altogether, but simply stretch one or more data phases. The generic name for this is *wait-state insertion*. Wait states are a reasonable alternative to Retry and Disconnect if there are not too many of them; when there are excessive wait states, bus performance would be better served by the devices giving up the bus and allowing it to be used by other devices while they prepare for the resumption of the suspended transaction. PCI targets de-assert the TRDY# signal during any data phase to indicate wait states. A target must be prepared to complete each data phase within 8 PCI clocks (maximum of seven wait states), except for the first data phase which it must complete within 16 clocks. If a target cannot meet the "16 and 8 tick" rules for completing a data phase, it must signal Retry or Disconnect instead.

PCI Initiator Flow Control Problems

While many flow control problems are associated with the target of a transaction, there are a couple which may occur on the initiator side. Again, the cases are described in terms of PCI protocol.

PCI Initiator Starts, But Can't Continue. Some bus protocols also allow an initiator to break off a transaction early in the event it can't accept the next read data or source the next write data within the time allowed by the protocol — even with wait states. PCI initiators suspend transactions simply by de-asserting the FRAME# signal early. As a rule, the master will re-arbitrate later for the PCI bus and perform a new transaction which picks up from where it left off previously.

PCI Initiator Starts, Can Continue, But Needs Wait-States. Some bus protocols allow an initiator to insert wait states in a transfer, just as the target may. Other bus protocols (e.g. PCI-X) only allow targets to insert wait states — based on the assumption that a device which starts a transaction should be ready to complete it before requesting the bus. In any case, PCI initiators de-assert the IRDY# signal to indicate wait states. An initiator must be prepared to complete each data phase within 8 clocks (maximum of seven wait states); if it can't meet this rule for any data phase, it must instead suspend the transaction by de-asserting FRAME#.

All PCI Flow Control Problems Hurt Performance

Each of the initiator and target flow control problems just described impact PCI bus performance for both the devices involved in the transfer, and for devices waiting to access the bus. While not every transaction is afflicted with target retries and disconnects, or early de-assertion of FRAME# by initiators, they happen enough to make effective bandwidth considerably less than 132MB/s on the PCI bus. In addition, arbitration and flow control uncertainties make system performance difficult to estimate.

HyperTransport Flow Control: Overview

All of the flow control problems described previously for PCI severely hurt bus performance and would be even less acceptable on a very high-performance connection. The flow control scheme used in HyperTransport applies independently to each transmitter-receiver pair on each link. The basic features include the following.

Packets Never Start Unless Completion Assured

All transfers across HyperTransport links are packet based. No link transmitter ever starts a packet transfer unless it is known the packet can be accepted by the receiver. This is accomplished with the "coupon based" flow control scheme described in this section, and eliminates the need for the Retry and Disconnect mechanisms used in PCI.

Transfer Length Is Always Known

Hypertransport control packets have a fixed size (four or eight bytes) and data packets have a *known* and *maximum* transfer length, unlike PCI data transfers. This makes buffer sizing and flow control much more straightforward as both transmitter and receiver are aware of their actual transfer commitments. It also makes the interleaving of control packets with data packets much simpler.

Split Transactions Used When Response Is Required

HyperTransport performs all read and non-posted write operations as split transactions, eliminating the need for the inefficient Retry mechanism used in PCI. A split transaction breaks a transfer which requires a response (and maybe data) into two parts — the sending of the request packet, followed later by response/data packets returned by the original target. This keeps the link free during the period between request and response, and means that the burden for completing the transaction is on the device best equipped to know when it is possible to do so — the target.

Flow Control Pins Are Eliminated

Because HyperTransport uses a message-based flow control scheme, it eliminates the flow control handshaking pins and signal traces found on other buses. Instead, each pair of devices on a link convey flow control information related to their receivers by sending update NOP packets over their transmitter connections.

Flow Control Buffers Mean No Bus Wait States

All link receiver interfaces are required to implement a set of buffers which are capable of receiving packets at full speed. Once a transmitter has determined that buffer space is available at the receiver, the transfer of the bytes within the packet always proceeds at full bus speed into the receiver buffer. The buffers are sized such that the full packet can always be accepted. Data packets can be as large as 64 bytes (16 dwords) and control packets can be as large as 8 bytes. The one twist to this is the fact that the transmitter has the option of interleaving new control packets into a large data packet on four byte boundaries. Still, this is done at full speed, without any wait states. The transmitter simply asserts the CTL signal to indicate control packets are moving across the CAD bus, and deasserts it to indicate data packets are moving across; the target uses the CTL signal input to determine which buffer the packet should enter.

Flow Control Buffers For Each Virtual Channel

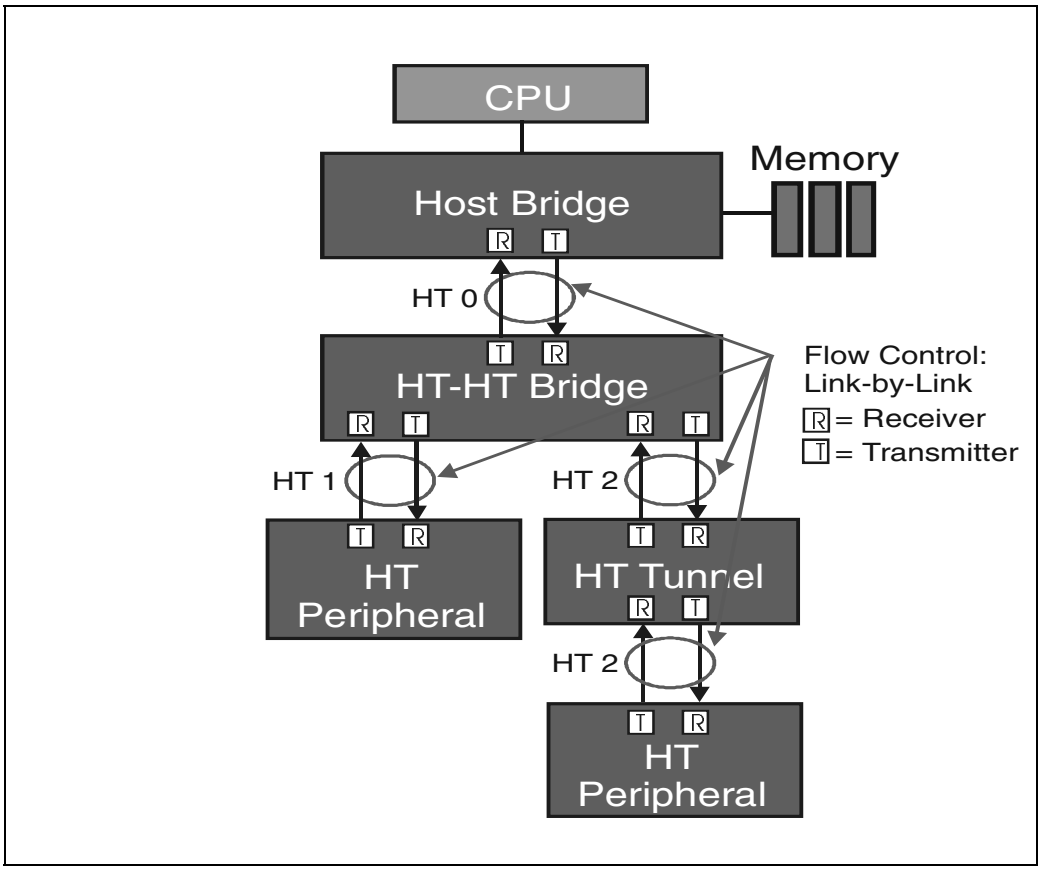
Finally, because there are a minimum of three virtual channels as packets move through HyperTransport, the flow control mechanism maintains separate flow control buffer pairs for the *posted request, non-posted request,* and *response* virtual channels. Each non-posted request has an associated response (and possibly data); that must be tracked internally by the device until the response comes back. Posted requests do not have a response, and may be flushed internally as soon as they are processed. In addition, the separate flow control buffers are important in enforcing the ordering rules that apply to the three virtual channels.

Optionally, devices may also support isochronous transfers; in this case, three additional receiver flow control buffer sets (CMD/Data) would be required to track this traffic.

Flow Control, A System View

While HyperTransport flow control is enforced on a per-link basis, there are system implications if it does not perform properly. As a point-point technology, HyperTransport devices such as tunnels and bridges have responsibilities for generating their own packets upstream as well as forwarding those from devices either above or below them. Refer to Figure 5-2 on page 106. Note that when the peripheral at the bottom of the chain on the right issues a packet upstream towards memory, it is dependent on how well the devices above manage their links.

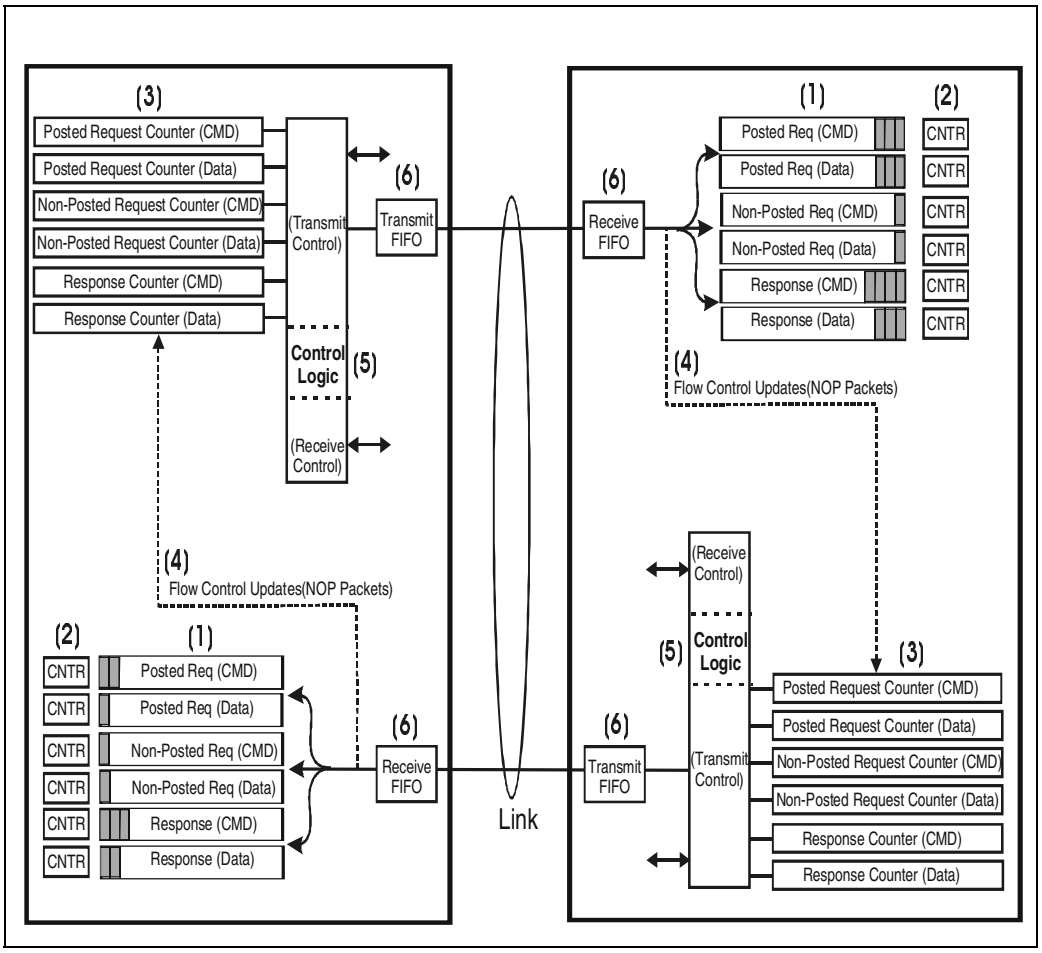
Figure 5-2: Flow Control Is On A Link-By-Link Basis



Flow Control Buffer Arrangement

Figure 5-3 on page 107 illustrates the general arrangement of HyperTransport flow control buffers and counters required of each link receiver interface. Note that while the transmit interface maintains flow control counters, the flow control buffers are only on the receive side of each device.

Figure 5-3: Flow Control Buffers And Counters



Details Associated With Figure 5-3

The following section describes the architectural features shown in Figure 5-3 on page 107. Note that the drawing is conceptual and is intended to show the major features of flow control on a single link. For multiple-link devices such as tunnels, this logic would be replicated for each interface.

Flow Control Buffer Pairs (Item 1)

Each receiver interface is required to implement six buffers to accept the following packet types being sent by the corresponding transmitter. *The specification requires a minimum depth of one for each buffer, meaning that a receiver is permitted to deal with as few as one packet of each type at a time. It may optionally increase the depth of one or more of the buffers to track multiple packets at a time.*

Posted Request Buffer (Command). This buffer stores incoming posted request packets. Because every request packet is either four or eight bytes in length, each entry in this buffer should be eight bytes deep.

Posted Request Buffer (Data). This buffer is used in conjunction with the previous one and stores data associated with a Posted Request. Because posted request data packets may range in size from 1 dword to 16 dwords (64 bytes), each entry in this buffer should be 64 bytes deep.

Non-Posted Request Buffer (Command). This buffer stores incoming non-posted request packets. Because every request packet is either four or eight bytes in length, each entry in this buffer should be eight bytes deep.

Non-Posted Request Buffer (Data). This buffer is used in conjunction with the previous one and stores data associated with a Non-Posted Request. Because non-posted request data packets may range in size from 1 dword to 16 dwords (64 bytes), each entry in this buffer should be 64 bytes deep.

Response Buffer (Command). This buffer stores returning response packets. Because every response packet is four bytes in length, each entry in this buffer should be four bytes deep.

Response Buffer (Data). This buffer is used in conjunction with the previous one and stores data associated with a returning response. Because responses may precede data packets ranging in size from 1 dword to 16 dwords (64 bytes), each entry in this buffer should be 64 bytes deep.

Receiver Flow Control Counters (Item 2)

The receiver interface uses one counter for each of the flow control buffers to track the availability of new buffer entries. The size of the counter is a function of how many entries were designed into the corresponding flow control buffer. After initialization reports the starting buffer size to the transmnitter, the value in each counter only increments when a new entry becomes available due to a packet being consumed or forwarded; it decrements when NOP packets carry-

ing buffer update information are sent to the transmitter on the other side of the link.

Transmitter Flow Control Counters (Item 3)

It is a transmitter responsibility on each link to check the current state of receiver readiness before sending a packet in any of the three required virtual channels. It does this by maintaining its own set of flow control counters, which track the available entries in the corresponding receiver flow control buffer. For example, if the transmitter wishes to send a read request across the link, it would first consult the Non-Posted Request CMD counter to see the current number of credits. If the counter = 0, the receiver is not prepared to accept any additional packets of this type and the transmitter must wait until the count is updated via the NOP mechanism to a value >0. If the counter value is =1, the receiver will accept one packet of this type, etc. Note that for requests that are accompanied by data (e.g. posted or non-posted writes), the transmitter must consult both its CMD counter and the Data counter for that virtual channel. If either is at 0, it must wait until both counters have been updated to non-zero values.

NOP Packet Update Information (Item 4)

During idle times on the link, each device sends NOP packets to the other. If one or more buffer entries in any of the six receiver flow control buffers have become available, designated fields in the NOP packets are encoded to indicate that fact. Otherwise those fields contain 0, indicating no new buffer entries have become available since the previous NOP transmission. In the next section, use of the NOP packet fields for flow control updates is reviewed. Refer to “NOP Packet” on page 71 for additional discussion of the NOP packet format.

Control Logic (Item 5)

This generic representation of internal control logic is intended to indicate that a number of things related to flow control are under the management of each HyperTransport device. In general:

- Logic associated with the transmit side of a link interface always must consult transmitter flow counters before commencing a packet transfer in any virtual channel. This assures that any packet sent will be accepted.
- Logic monitoring the progress of packet processing in the receiver flow control buffers, must translate new entries that become available into NOP update information to be passed back to the transmitter.
- Logic monitoring the receive side of a link interface must parse incoming

NOPs to determine if the receiver is reporting any changes in buffer availability. If so, then the information is used to update the transmitter's flow control counters to match the available buffer entries on the receiver side.

Transmit And Receive FIFO (Item 6)

The transmit and receive FIFOs are not part of flow control at all, and are shown here as a reminder that all packets moving across the high-speed HyperTransport link pass through an additional layer of buffering to help deal with the effects of clock mismatch within the two devices, skew between multiple clocks sourced by the transmitter on a wide interface, etc. See Chapter 15, entitled "Clocking," on page 387 for a discussion concerning the FIFOs.

Example: Initialization And Use Of The Counters

The following three diagrams and associated descriptions explain the initialization of HyperTransport buffer counts, followed by the actions taken by the transmitter and receiver as two packets are sent across the link. The diagrams have been simplified to show a single flow control buffer and the corresponding receiver and transmitter counters used to track available entries. In this example, assume the following:

- The flow control buffer illustrated is the Posted Request Command (CMD) buffer.
- The designer of the receiver interface has decided to construct this flow control buffer with a depth of five entries. Because this is a buffer for receiving requests, each entry in the buffer will hold up to 8 bytes (this covers the case of either four or eight byte request packets)
- Following initialization, the transmitter wishes to send two Posted Request packets to the receiver.

Basic Steps In Counter Initialization And Use

1. At reset, the transmitter counters in each device are reset = 0. This prevents the initiation of any packet transfers until buffer depth has been established.
2. At reset, the receiver interfaces load each of the RCV counters with a value that indicates how many entries its corresponding flow control buffer supports (shown as N in the diagram). This is necessary because the receiver is allowed to implement buffers of any depth.
3. Each device then transmits its initial receiver buffer depth information to the other device using NOP packets. Each NOP packet can indicate a range

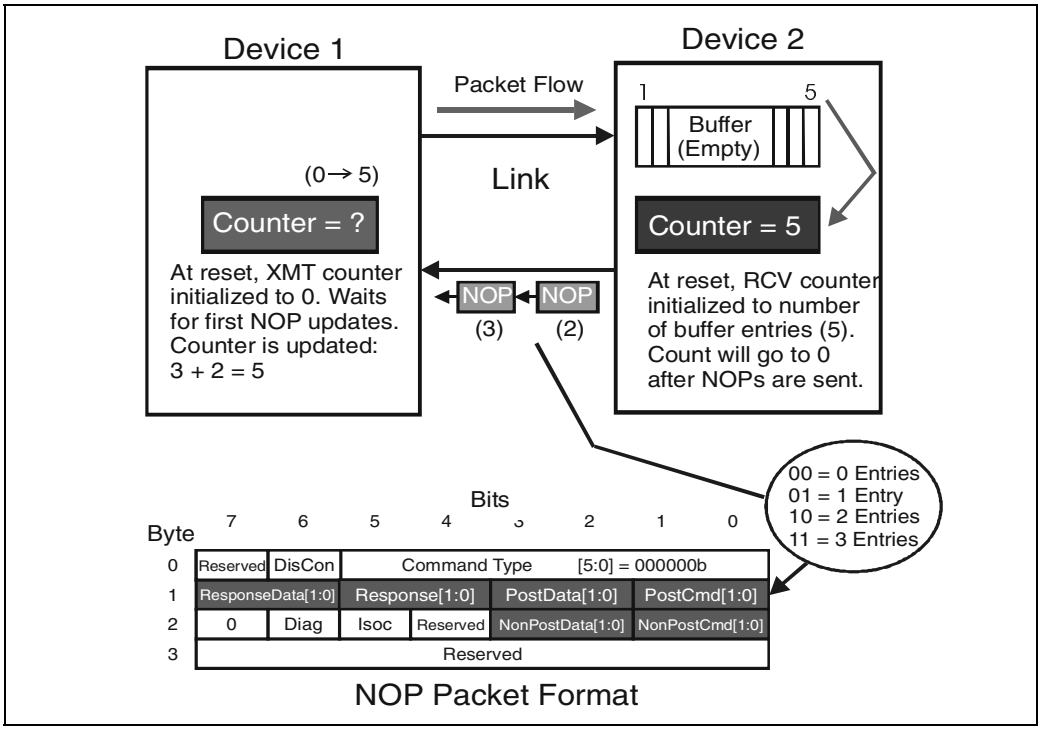
of 0-3 entries. If the receiver buffer being reported is deeper than 3 entries, the device will send additional NOPs which carry the remainder of the count.

4. As each device receives the initial NOP information, it updates its transmitter flow control counters, adding the value indicated in the NOP fields to the appropriate counter total.
5. When a device has a non-zero value in the counter, it can send packets of the appropriate type across the link. Each time it sends packet(s), the device subtracts the number of packets sent from the current transmitter counter value. If the counter decrements to 0, the transmitter must wait for NOP updates before proceeding with any more packet transmission.

Initializing The Flow Control Counter

In Figure 5-4 on page 111, assume that the designer of Device 2 has implemented a Posted Request (CMD) buffer with a depth of 5 entries. After reset, it must convey this initial buffer availability to the transmitter in the other device before any Posted Request packets may be sent.

Figure 5-4: Flow Control Counter Initialization



The basic steps in flow control counter initialization are:

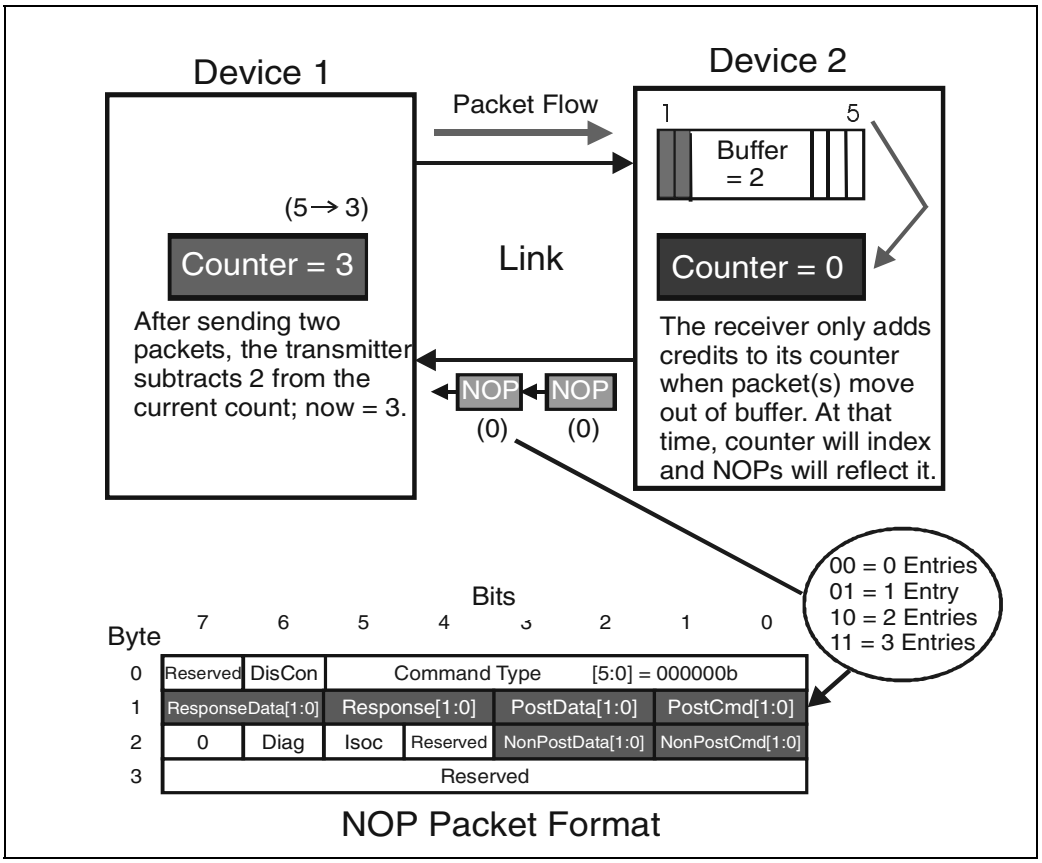
1. The transmitter in Device 1 initializes its Posted Request (CMD) counter to 0 at reset (all transmit counters reset = 0). It then waits for the receiver on the other side to update this counter with the starting buffer depth available (this will be the maximum depth the receiver supports).
2. Device 2 loads its receiver Posted Request counter = 5 (its maximum).
3. Device 2 then sends two NOP packets which carry this buffer availability information: the first NOP has a 11b (3) in the Post CMD field (Byte 1, bits 0,1 above), and the second NOP has a 10b (2) in this field. Total = 5.
4. Upon receipt of these two NOPs, the Device 1 has updated its transmit counter, first by three then again by two. It now has 5 "credits" available for sending Posted Request packets — representing five separate Posted Requests which may be initiated.
5. Having sent the NOPs, the Device 2 RCV counter is now at 0, and will remain that way until additional packets are received, processed, and move out of the buffer, thereby creating new entries.

Note that this process will be repeated for each of the six required flow control buffers; it will also be done for the six isochronous flow control buffers if they are supported. In the NOP packet format (see above), six transmit registers can be updated at once using the six fields provided. The *Isoc* bit (Byte 2, bit 5) would be set if the NOP update was to be applied to the isochronous flow control buffer set.

Device 1 Sends Two Posted Request Packets

Figure 5-5 on page 113 shows the Device 1 transmitter sending two Posted Request packets. Also illustrated is the state of the flow control registers after this has been done, but before the receiver has processed the incoming packets.

Figure 5-5: Device 1 Sends Two Packets



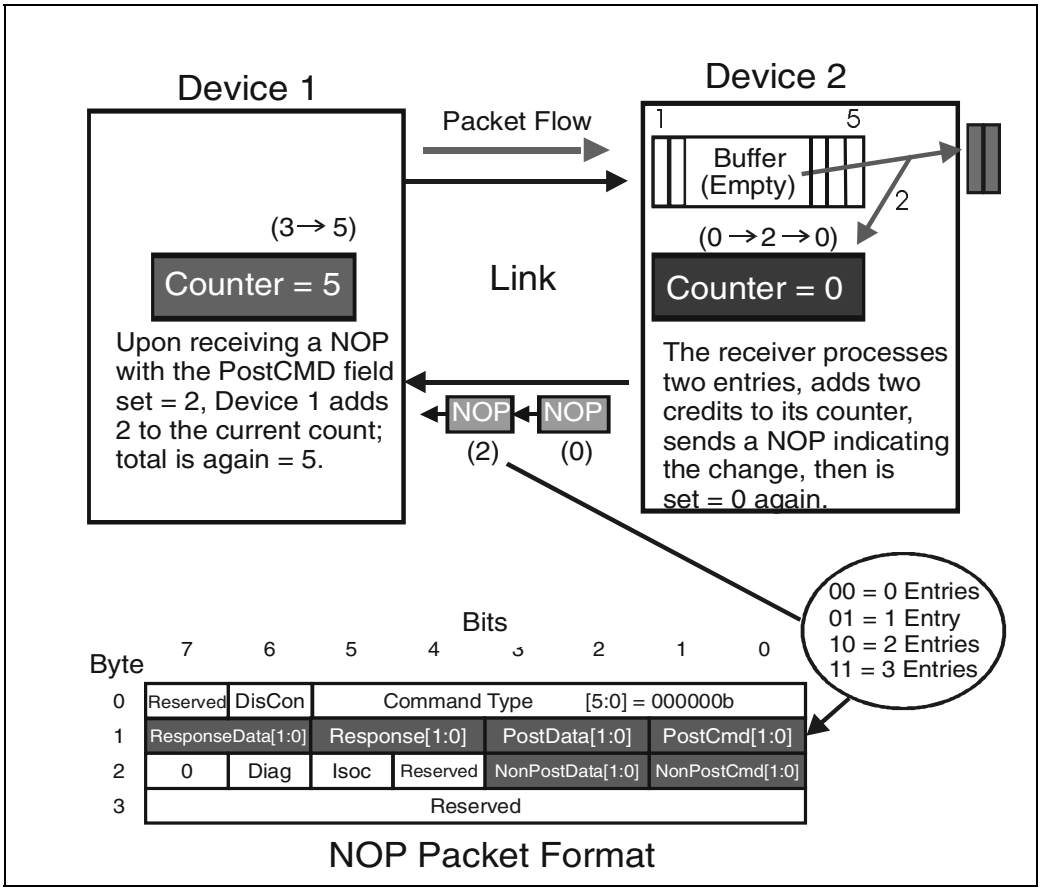
What the diagram shows:

1. After the flow control counters are initialized after reset, the transmitter sends packets for which credits are available in the transmit counter, then subtracts the number of packets sent from the current total. In this case, two credits were subtracted from the starting count of 5. Maintaining its own counter assures the transmitter will never send packets which can't be taken, even if there is a considerable lag in NOP updates from the receiver.
2. The receiver will not update its RCV counter or indicate new entries in its NOP packets until it moves a packet out of the flow control buffer.

New Entries Available: Update Flow Control Information

The last diagram, Figure 5-6 on page 114, shows the updating of flow control information after the receiver has processed the two packets it received previously. These could have been consumed internally, or been forwarded to another link if Device 2 is a bridge or tunnel.

Figure 5-6: Device 2 Updates Flow Control Information



What has happened:

1. When Device 2 moves packets out of its buffer, it indexes its receive counter to reflect the new availability (2), and sends this information to Device 1 with a NOP update packet carrying the value *2* in the PostCMD field. After sending the update NOP, Device 2 clears its receive counter.
2. After sending packets and subtracting credits from its transmit counter, Device 1 will parse incoming NOPs for updates from the receiver enabling it to bump credits in one or more of its counters. In this case, Device 1 bumps its transmit counter by 2 upon receiving the NOP update. It again has 5 credits to use for sending Posted Request command packets.

This dynamic updating of flow control information happens continuously during idle times on the bus. If the receiver has no change in buffer entry availability to report, the NOPs it sends will have all six fields in the NOP packet cleared.

A Few Implementation Notes

Information Packets Not Flow-Controlled

Information packets, including NOP and Sync are not subject to flow control. When sent, they must be accepted, and are used for point-point communication between a transmitter and its corresponding receiver on a given link.

Transmitter Must Be Able To Track 15 Buffer Entries

While a receiver has the option of implementing buffers of any desired depth, including single entry buffers, each transmitter interface must implement its flow control counters such that they can track up to 15 entries in each of the six corresponding receiver flow control buffers (a four bit transmit counter will do this). If the transmitter counter is larger than that of the receiver, only a portion of it will ever be used (because NOP updates are always are based on available receiver flow control buffer entries). In the event that a transmitter implements a counter smaller than that of the receiver, the counter must “saturate” at the maximum value it can handle, and not roll over. The idea is that once the counters are initialized, they will use the maximum count that both devices can accommodate.

Sometimes Two Counters Must Be Checked

A transmitter can't issue a request packet that has data associated with it (e.g. a posted or non-posted write) without assuring the receiver has buffer entries available for both the request and the data. This is necessary because there is no receiver disconnect or retry mechanism once such a transaction starts.

NOP Packets Cannot Be Completely Blocked

The HyperTransport Specification indicates that it is the responsibility of each device to make certain that NOP update packets it sends are not starved (prevented from being sent) because of the sending of other types of traffic. If they are blocked, eventually one or more of the virtual channels may completely stall.

The Isochronous Flow Control Option

In the event that a designer decides to provide Isochronous flow control in addition to the standard three virtual channels, each receiver interface which supports Isochronous will implement six more receiver flow control buffers and counters, and an additional set of transmitter flow control counters as well. The way the receiver determines which flow control buffer (isochronous or standard) a packet should use is determined by a bit in the request packet (*Isoc* bit). If the Isoc bit is asserted in a request, it will also be asserted in the response when it comes back — again identifying the buffer set to use.

How About NOP Updates For Isochronous Buffers?

If a device supports the Isochronous flow control buffers, it will track packet progress through these buffers in the same way as it does for non-isochronous packets it receives. As Isochronous buffer entries become available, the receiver will return NOP update packets to the other device and will set the *Isoc* bit in the NOP packet (byte 2, bit 5) indicating the NOP packet updates should be applied to the isochronous transmit counters.

Isochronous Traffic/Non-Isochronous Flow Control

Receivers which see request packets with the Isoc bit set, but which are not in isochronous flow control mode, do not use the dedicated isochronous flow control buffers to handle them. In this case, the standard six flow control buffers are used and NOP buffer update packets returned to the transmitter all apply to the standard transmitter flow counters. Such devices preserve the Isoc bit in both the request packet and its response as they forward it to the next device; in this way, if there is a device in the path that does support isochronous traffic, it can still be used in that portion of the topology.

Isochronous Traffic Disabled At Initialization

At initialization, all devices are disabled with respect to isochronous traffic. Software can later enable ISOC traffic on a link-by-link basis after support on both sides of the link is determined through configuration space accesses. Once enabled for ISOC traffic, each device which sees isochronous packets and supports them is expected to apply a higher priority to them than for standard virtual channel packets.