# A Memory-Efficient Implementation of the HyperTransport Coupon-Based Flow Control Protocol

Jose Duato
Universidad Politecnica de Valencia, Spain
and Simula Research Laboratory, Norway

Federico Silla
Universidad Politecnica de Valencia, Spain

## I - Introduction

HyperTransport devices communicate by exchanging packets. These packets fall into one of two classes: control packets and data packets. Control packets carry requests and their corresponding responses, if required. When a request or a response packet needs some data to be transferred, an additional data packet following the control packet is sent.

Request packets include read and write requests as well as other commands such as flushes or atomic read-modify-write operations. An example request packet is shown in Figure 1. Depending on the exact contents of the command field (Cmd), this request may be a read, a write, a read-modify-write, or a broadcast. As can be seen in the figure below, this request packet contains an address, and therefore, it is 8 bytes long.

| Bit-Time | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | SeqID[3:2] | | Cmd[5:0] | | | | | |
| 1 | PassPW | SeqID[1:0] | UnitID[4:0] | | | | | |
| 2 | *Command-Specific* | | | | | | | |
| 3 | *Command-Specific* | | | | | | | |
| 4 | Addr[15:8] | | | | | | | |
| 5 | Addr[23:16] | | | | | | | |
| 6 | Addr[31:24] | | | | | | | |
| 7 | Addr[39:32] | | | | | | | |

**Figure 1** – Request packet format with address

In the HyperTransport protocol, not all request packets are 8 bytes long. Some request packets, such as fence or flush packets, are 4 bytes long. On the other hand, packets may be extended by using an address extension, a source identifier extension, or both. Packet extensions are always 4 bytes long. Therefore, the length of request packets exchanged by HyperTransport devices may be 4, 8, 12, or 16 bytes, depending on the type of request and the number of extensions added to the packet.

Response packets are always 4 bytes long. Response packets include read responses and target done packets. Figure 2 shows an example of a read response packet.

| Bit-Time | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Isoc | Rsv | Cmd[5:0]: 110000 | | | | | |
| 1 | PassPW | Bridge | Rsv | UnitID[4:0] | | | | |
| 2 | Count[1:0] | | Error0 | SrcTag[4:0] | | | | |
| 3 | Rsv/RqUID | | Error1 | Rsv/RspVCSet | | | Count[3:2] | |

**Figure 2** – Read response packet

As mentioned above, a data packet may accompany a control packet when some data must be transferred. This is the case for the read response packet shown in Figure 2. As shown in that figure, the packet does not carry any data, despite the fact that it is a read response. The expected read data is contained in a second packet following the one shown in Figure 2. The second packet is a data packet. Data packets follow write requests and read responses. They range in length from 4 to 64 bytes and in multiples of 4 bytes. Figure 3 shows an example of an 8 byte data packet.

| Bit-Time | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Data[7:0] | | | | | | | |
| 1 | Data[15:8] | | | | | | | |
| 2 | Data[23:16] | | | | | | | |
| 3 | Data[31:24] | | | | | | | |
| 4 | Data[39:32] | | | | | | | |
| 5 | Data[47:40] | | | | | | | |
| 6 | Data[55:48] | | | | | | | |
| 7 | Data[63:56] | | | | | | | |

**Figure 3 –**8 byte data packet

Table 1 summarizes the types of packets and their corresponding lengths. See Sections 3.2, 4.4, and 4.5 of the HyperTransport I/O Link Specification Revision 3.0 for a complete description of HyperTransport packets.

| Packet Type | Length |
|---|---|
| Control – Request | 4, 8, 12, or 16 bytes |
| Control – Response | 4 bytes |
| Data | From 4 to 64 bytes in 4 byte steps |

**Table 1 –** Packet types and lengths

## II - The HyperTransport Flow Control Protocol

HyperTransport devices store incoming packets in buffers while deciding whether to forward or accept them. There are six basic types of buffers:

- Nonposted Requests
- Posted Requests
- Responses
- Nonposted Request Data
- Posted Request Data
- Response Data

According to Section 4.8 of the HyperTransport I/O Link Specification Revision 3.0, request and response buffers contain enough storage space to store the largest control packet of that type. Also, all data buffers can hold 64 bytes. Additionally, and in order to improve performance, a HyperTransport device may have several buffers of each type. The exact number of buffers depends on the implementation; it is even possible to design a device with fewer buffers than the minimum required to fully utilize link bandwidth.

The HyperTransport protocol states that a transmitter should not issue a packet that cannot be stored by the receiver. Thus, the transmitter must know how many buffers of each type the receiver has available. To achieve this, a coupon-based[1] scheme is used between transmitters and receivers. With such a scheme, the transmitter has a counter for each type of buffer implemented at the receiver. When the transmitter sends either a control or data packet, it decrements the associated counter. When one of the counters reaches zero, the transmitter stops sending packets of that type. On the other hand, when the receiver frees a buffer, it sends a no-operation (NOP) packet to the transmitter in order to inform it about space availability. Figure 4 shows a NOP packet. In it, the fields used for returning coupons

1 Coupon-based schemes are traditionally known as credit-based schemes.

for each of the buffer types are 2 bits wide, thus allowing up to three buffers to be declared free. If a receiver frees more than three buffers, then it will send several NOP packets.

| Bit-Time | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | Rsv | DisCon | Cmd[5:0]: 000000 | | | | | |
| 1 | ResponseData[1:0] | | Response[1:0] | | PostData[1:0] | | PostCmd[1:0] | |
| 2 | 0 | Diag | Isoc | Rsv | NonPostData[1:0] | | NonPostCmd[1:0] | |
| 3 | RxNextPktToAck [7:0] | | | | | | | |

**Figure 4 –** NOP packet

In order to initialize counters properly, upon link reset, the transmitter clears its counters and the receiver sends NOP packets to indicate how many buffers of each type it has available. As fields in the NOP packet allow declaring only up to three available buffers, receivers having more buffers will send additional NOP packets. Because transmitters and receivers might be independently designed, if a transmitter receives more coupons than it can keep track of, the corresponding counter will saturate, i.e., it will never wrap. In this case, both the transmitter and the receiver will use the maximum number of coupons that they can simultaneously support. Additionally, transmitter counters must allow the tracking of at least 15 buffers, thus making the minimum counter width equal to 4 bits. Refer to Section 4.8 of the HyperTransport I/O Link Specification Revision 3.0 for additional information on the HyperTransport flow control protocol.

## III - A Simple Implementation of HyperTransport Flow Control

The HyperTransport flow control protocol may be implemented just by using a counter at the transmitter for each of the buffer types and a NOP packet generator at the receiver, as presented in the previous section. From the memory usage point of view, such an implementation is very simple and efficient in the case of response packets. However, it is not in the case of request and data packets, in which memory usage might not be efficient. In the former case (when applied to response packets), the implementation is efficient because all response packets have the same length, i.e., the buffer size allocation matches the coupon's size definition; meaning that a coupon reports a free 4 byte buffer and is signalling that there is room for one additional response packet. In the case of request packets, buffer size and coupons do not have the same meaning, for the reason that not all request packets have the same length. Therefore, the buffer size must be enlarged to fit the largest request packet size, i.e., 16 bytes, as described in Section 4.8.1 of the HyperTransport I/O Link Specification Revision 3.0. However, request packets range from 4

to 16 bytes. In this context, a coupon signalling that there is one free 16 byte buffer does not necessarily imply that there is room for only one additional request packet, because incoming packets might be smaller. For example, if incoming request packets are 8 bytes long, there is actually room for two request packets. This mismatch leads to decreased memory usage efficiency, since we could store more incoming packets than the coupon count. Table 2 shows an example of the coupon count and memory occupancy progression when several request packets are received. The last column of Table 2 shows the improved coupon count that results from taking into account the size of incoming packets, i.e., numbers computed as free memory space divided by the maximum packet length. Table 2 assumes that total memory size is the minimum (15 buffers, each of them being 16 bytes long). Incoming request packet lengths included in Table 2 represent one of any possible combinations of request packet lengths.

As seen in Table 2 and as a result of the standard HyperTransport protocol behavior, each time a packet is sent to the link, the transmitter decreases its coupon count, and by taking into account the length of incoming packets, memory usage is lower than the one indicated by the coupon count. Therefore, as shown in the last column of Table 2, when that count reaches zero and forces the transmitter to stop sending more packets, the receiver has enough memory space to store six additional full-sized packets.

| Received Packet Length | Coupon Count | Free Memory | Improved Coupon Count |
|---|---|---|---|
| Initially | 15 | 240 | 15 |
| Packet 1: 8 bytes | 14 | 232 | 14 |
| Packet 2: 12 bytes | 13 | 220 | 13 |
| Packet 3: 8 bytes | 12 | 212 | 13 |
| Packet 4: 12 bytes | 11 | 200 | 12 |
| Packet 5: 4 bytes | 10 | 196 | 12 |
| Packet 6: 16 bytes | 9 | 180 | 11 |
| Packet 7: 4 bytes | 8 | 176 | 11 |
| Packet 8: 8 bytes | 7 | 168 | 10 |
| Packet 9: 12 bytes | 6 | 156 | 9 |
| Packet 10: 8 bytes | 5 | 148 | 9 |
| Packet 11: 4 bytes | 4 | 144 | 9 |
| Packet 12: 16 bytes | 3 | 128 | 8 |
| Packet 13: 12 bytes | 2 | 116 | 7 |
| Packet 14: 8 bytes | 1 | 108 | 6 |
| Packet 15: 8 bytes | 0 | 100 | 6 |

**Table 2 –** Coupon count progression and memory usage while receiving request packets

If we look at the coupon count progression while forwarding packets, the same effect is observed. Table 3 shows this progression when the 15 packets listed in Table 2 are forwarded, thus freeing memory. Table 3 assumes that packets are forwarded in the same order as they were received.

| Forwarded Packet Length | Coupon Count | Free Memory | Improved Coupon Count |
|---|---|---|---|
| Initially | 0 | 100 | 6 |
| Packet 1: 8 bytes | 1 | 108 | 6 |
| Packet 2: 12 bytes | 2 | 120 | 7 |
| Packet 3: 8 bytes | 3 | 128 | 8 |
| Packet 4: 12 bytes | 4 | 140 | 8 |
| Packet 5: 4 bytes | 5 | 144 | 9 |
| Packet 6: 16 bytes | 6 | 160 | 10 |
| Packet 7: 4 bytes | 7 | 164 | 10 |
| Packet 8: 8 bytes | 8 | 172 | 10 |
| Packet 9: 12 bytes | 9 | 184 | 11 |
| Packet 10: 8 bytes | 10 | 192 | 12 |
| Packet 11: 4 bytes | 11 | 196 | 12 |
| Packet 12: 16 bytes | 12 | 212 | 13 |
| Packet 13: 12 bytes | 13 | 224 | 14 |
| Packet 14: 8 bytes | 14 | 232 | 14 |
| Packet 15: 8 bytes | 15 | 240 | 15 |

**Table 3 –** Coupon count and memory usage progression while forwarding request packets

As mentioned above, in the case of data packets, memory occupancy might not be efficient. The reason is the same as for request packets: data packets may have different lengths, ranging from 4 to 64 bytes. Thus, when a coupon signals that there is one free 64 byte buffer available, it does not necessarily imply that there is space for only one additional data packet because incoming data packets might be smaller. This is especially the case for byte reads that are 4 bytes long, byte writes that are up to 32 bytes long, and atomic read-modify-writes that are up to 16 bytes long (see Sections 4.4 and 4.5 of the HyperTransport I/O Link Specification Revision 3.0 for additional information). In all such cases, each data packet occupies a 64 byte buffer at the receiver end, noticeably wasting memory space.

## IV - An Improved Implementation of HyperTransport Flow Control

By leveraging the ideas presented in the previous section, an improved implementation of the HyperTransport flow control protocol can be realized. It is important to note that such implementation does not require changing the HyperTransport Link Specification. Furthermore, the proposed changes apply only to the receiver, i.e., no modifications to the

transmitter are required. In fact, improved receivers can be directly connected to and transparently interoperate with legacy transmitters, therefore ensuring complete silicon and protocol backward compatibility.

The proposed improved implementation of the HyperTransport flow control protocol operates as follows: any time a transmitter sends a packet, it decreases the corresponding counter, as normally done. Every time the transmitter receives a NOP packet signalling the availability of freed buffers, it increases its counters, also as normally done. Thus, the transmitter behavior is not modified. At the receiver end, instead, the improved implementation takes into account free memory bytes instead of free packet buffers. To do so, two separate counters are needed, as shown in Figure 5.

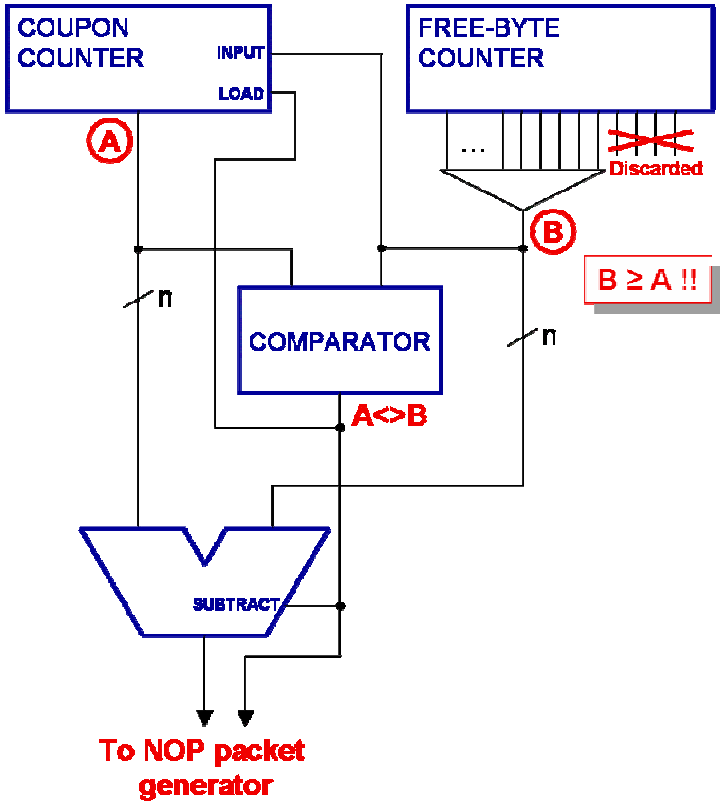


**Figure 5 –** Proposed implementation of the improved HyperTransport flow control protocol at the receiver end. This circuit must be replicated for each buffer type.

The coupon counter in Figure 5 keeps track of the status seen by the transmitter. Every time a new packet is received, count A is decremented by one, so as to precisely reflect the coupon count seen by the transmitter.

The free-byte counter keeps track of the available space at local memory. Every time a packet arrives at the receiver, the free-byte counter is decremented by the packet length. The four least significant bits (LSBs) of this counter are discarded in order to divide the resulting count by 16, as shown in Figure 5. It must be remembered that 16 bytes is the length of the largest request packet, and therefore, being divided by 16, count B represents the amount of full-sized packets that can be stored in memory. Note that count B will always be greater than or equal to count A. If a sequence of full-sized packets is received, then count B will be equal to count A. However, if incoming packets are smaller than 16 bytes, then count B will eventually be greater than count A.

Now that we can accurately track the availability of space at the receiver end, the next step is to update this information at the transmitter end, if required. To do so, count A and count B are first compared. If both are the same, the coupon count at the transmitter end is correct and therefore the process ends. On the other hand, if they differ, updating is required. In this case, updating is accomplished by sending back one or more NOP packets containing additional coupons. The exact number of coupons to be sent is computed by subtracting count A from count B. This operation is triggered by the comparator output, which also triggers the loading of count B into the coupon counter, and additionally, it signals the need of and update to the NOP packet generator. The NOP packet generator should latch the result of the subtraction because after several gate delays, it will become zero, as the loading of the coupon counter with the value in count B will force the comparator to reset its output. At the end of this process, one or more NOP packets will be returned to reflect the availability of additional memory space, and the coupon counter at the receiver will be properly updated to follow the new status at the transmitter.

Let us now analyze what takes place when a packet stored in memory is forwarded. In that case, the legacy behavior would be the return of a coupon because a buffer has been freed. However, with the proposed improved implementation that takes into account free memory bytes instead of free packet buffers, it might happen that the memory space freed by the forwarded packet is not large enough to store a full-sized packet, and thus, no coupon is returned. Therefore, the way to properly update freed memory information while forwarding packets is to simply increase the free-byte counter by the amount of space freed. The rest of the circuit presented in Figure 5 makes the decision of sending or not sending additional NOP packets as explained before. For example, Table 4 shows the operational progression as several packets of different lengths are received and later forwarded. As mentioned earlier, the minimum total memory size for each type of buffer is 240 bytes (15 buffers, each 16 bytes long). We could be tempted to use 256 bytes as the most logical choice. However, 256 bytes would force us to use 5 bit coupon counters in

order to keep track of the sixteenth stored packet. Naturally, using an additional bit just to store one more packet would not be a good decision. Therefore, coupon counters should be 4 bits wide and the free-byte counters shown in Figure 5 should be 8 bits wide, discarding their four least significant bits.

| Packet Length | Coupon Counter (A) | Free-Byte Counter | 4 Most Significant Bits (MSBs) of Free-Byte Counter (B) |
|---|---|---|---|
| Initially | 15 | 240 | 15 |
| Packet 1 received: 8 bytes | 14 | 232 | 14 |
| Packet 2 received: 12 bytes | 13 | 220 | 13 |
| Packet 3 received: 8 bytes | 12 | 212 | 13 |
| **B is greater than A. Thus, the coupon count is adjusted by sending back a coupon and loading the coupon counter with the value in B** | | | |
| | 13 | 212 | 13 |
| Packet 4 received: 12 bytes | 12 | 200 | 12 |
| Packet 5 received: 4 bytes | 11 | 196 | 12 |
| **B is greater than A. Thus, the coupon count is adjusted by sending back a coupon and loading the coupon counter with the value in B** | | | |
| | 12 | 196 | 12 |
| Packet 6 received: 16 bytes | 11 | 180 | 11 |
| Packet 7 received: 4 bytes | 10 | 176 | 11 |
| **B is greater than A. Thus, the coupon count is adjusted by sending back a coupon and loading the coupon counter with the value in B** | | | |
| | 11 | 176 | 11 |
| Packet 8 received: 8 bytes | 10 | 168 | 10 |
| Packet 9 received: 12 bytes | 9 | 156 | 9 |
| Packet 10 received: 8 bytes | 8 | 148 | 9 |
| **B is greater than A. Thus, the coupon count is adjusted by sending back a coupon and loading the coupon counter with the value in B** | | | |
| | 9 | 148 | 9 |
| Packet 1 forwarded: 8 bytes | 9 | 156 | 9 |
| **Once the packet is forwarded, as B is not greater than A, no coupon is sent back** | | | |
| Packet 2 forwarded: 12 bytes | 9 | 168 | 10 |
| **Once the packet is forwarded, as B is greater than A, a coupon is sent back and the coupon counter at the receiver is adjusted** | | | |
| | 10 | 168 | 10 |
| Packet 3 forwarded: 8 bytes | 10 | 176 | 11 |
| **Once the packet is forwarded, as B is greater than A, a coupon is sent back and the coupon counter at the receiver is adjusted** | | | |
| | 11 | 176 | 11 |
| Packet 4 forwarded: 12 bytes | 11 | 188 | 11 |
| **Once the packet is forwarded, as B is not greater than A, no coupon is sent back** | | | |
| Packet 11 received: 4 bytes | 10 | 184 | 11 |
| **B is greater than A. Thus, the coupon count is adjusted by sending back a coupon and loading the coupon counter with the value in B** | | | |
| | 11 | 184 | 11 |
| Packet 5 forwarded: 4 bytes | 11 | 188 | 11 |
| **Once the packet is forwarded, as B is not greater than A, no coupon is sent back** | | | |

**Table 4 –** Coupon counter and free-byte counter progression as several packets are received and forwarded

So far the discussion has focused on the improved flow control protocol implementation for request packets. As mentioned before, data packets may suffer the same inefficiency. In this case, the improved implementation would operate in the same manner as described so far, except for the width of the free-byte counter, which should be at least 10 bits wide, discarding the six LSB bits so as to divide by 64 (representing the data packet length).

## V - Impact of the Improved Implementation on Latency

The implementation of the efficient flow control protocol described in this paper is not expected to have any negative impact on packet latency, since packet transmission at the sender and packet forwarding at the receiver are not at all affected by it.

To analyze the impact of the proposed efficiency enhancement on packet latency, we might separately study how coupons are returned by the receiver when it receives packets and how coupons are returned when the receiver forwards packets. In the first case, every time the transmitter sends a new packet, it decreases its coupon count as usual. Nevertheless, once this packet arrives at the receiver, it will recalculate the improved coupon count as described in the previous section. If the improved count shows that one or more additional packets might be stored at the receiver memory, then it will return additional coupons. Note that the free-byte count B in Figure 5 will always be equal to or greater than the coupon count A. The effect of the proposed enhancement on packet latency while receiving packets will never be negative. In fact, in a worst-case scenario, the behavior of the proposed flow control protocol will be the same as the traditional protocol, therefore having no impact whatsoever on packet latency. In non-worst-case scenarios, the transmitter will send packets earlier because it owns more coupons, thus reducing packet latency.

In the latter case and as we have seen in the previous section, when the receiver is forwarding packets, coupons are returned in a delayed fashion, i.e., when a packet is forwarded, a coupon might not be returned if an additional full-sized packet cannot be stored. Compared with traditional protocol operation, however, this will not translate into any extra delay to the transmission of new packets by the transmitter, as the free-byte count (B) will always be equal to or greater than the coupon count (A). Additionally, coupon count A is continuously adjusted to equal count B when they differ. Thus, coupon count A will always be equal to or greater than the legacy coupon count. Therefore, with the enhanced flow control protocol, the transmitter will always own at least the same number of coupons that it would with the traditional flow control protocol. Consequently, the impact on

packet latency will be zero. If the transmitter owns more coupons, then packet latency will be reduced because packets are transmitted earlier.

In summary and as shown in Table 4, the proposed protocol enhancement returns coupons earlier than the traditional protocol when the coupons can fit into an already allocated buffer. Taking into account that the transmitter owns additional coupons to keep filling the link with new packets (unless buffers are full), the slight extra delay that could be introduced by the proposed circuit will not delay packet transmission since the transmitter does not have to wait for the return of any coupons.

## VI - Conclusions

This white paper has presented an efficient implementation of the HyperTransport flow control protocol that makes better use of memory resources. With this in mind, further considerations are in order:

A) The proposed implementation improves memory usage. This can be leveraged to increase the number of accepted packets per given amount of available memory; or given an average number of packets to be accepted, to reduce memory requirements. Notice that since the benefits of this efficient implementation depend on the specific packets mix, we can only base ourselves on “average number of packets.” Notice also that according to the HyperTransport I/O Link Specification Revision 3.0, the minimum amount of buffers that transmitters must be able to track is 15. This should compel us to take into account the round-trip latency for control or data packets and their associated NOP packets in order to fully utilize link bandwidth.

B) The improved implementation is expected to yield greater benefits in all those cases in which memory space at the receiver end is larger, as the improved protocol technique is based on average packet lengths. Therefore, a larger memory space will allow more packets to be included in the average.

C) The proposed protocol improvement does not lie within the critical path, and therefore, it is not sensitive to small increments in latency due to non-optimized designs. In fact, as the circuitry proposed in this paper consists of several elementary elements, the same can be scattered to improve the overall silicon layout.