Chapter 2
Link-Level Flow Control and Buffering

The simplest form of a network is composed of a single link with one sender and one receiver. In parallel to the data wires, the sender and the receiver need to exchange some extra information that will allow them to develop a common understanding on the intentions of each side. Figure 2.1a shows a sender and a receiver that besides the data wires drive two extra wires, a ready and a valid bit, that are responsible for co-ordinating the flow of data from one side to the other.

When the sender wants to put new data on the link it asserts the valid signal. The receiver samples new data from the link only when it sees valid = 1. If the sender wants to stall transmission it just drives the valid signal to 0.

Equivalently, at the other side of the link, the receiver may stall too. If the sender is not aware of the receiver's stall, it will provide new words on the link that will not be sampled by the receiver and destroyed by the subsequent words transmitted by the sender. Therefore, a mechanism is required that will inform the sender for the receiver's availability to receive new words. This is achieved by the ready signal. Any communication takes place only when the receiver is ready to get new data (ready = 1) and the sender has actually sent new data (valid = 1). When the receiver is not ready (ready = 0), the data on the link are not sampled and they should not change until the receiver resumes from the stall. The sender locally knows that when valid = 1 and ready = 1 the transmitted work is correctly consumed by the receiver and can send a new one.

The different values of the ready/valid signals put the link, between the sender and the receiver, in three possible states:

- Transfer: when valid = 1 and ready = 1, indicating that the sender is providing valid data and the receiver is accepting them.
- Idle: when valid = 0, indicating that the sender is not providing any valid data, irrespective the value of the ready signal.
- Wait: when valid = 1 and ready = 0, indicating that the sender is providing data but the receiver is not able to accept it. The sender has a persistent behavior and maintains the valid data until the receiver is able to read them.

© Springer Science+Business Media New York 2015

G. Dimitrakopoulos et al., *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*, DOI 10.1007/978-1-4614-4301-8_2

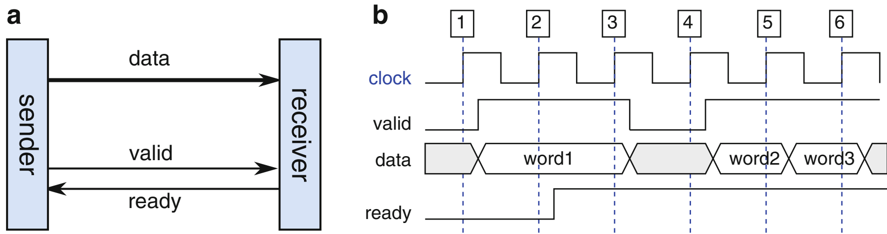


Fig. 2.1 (**a**) A flow-controlled channel with ready/valid handshake and (**b**) an example of transferring of three words between the sender and the receiver

Figure 2.1b shows an example of data transfers between a sender and a receiver on a flow-controlled link. In cycle 2, the sender has a valid word on its output (word1), but the receiver cannot accept it. The channel is in wait state. In cycle 3, data transfer actually happens since the sender and the receiver independently observe the channel's valid and ready signals being true. In cycle 4, the channel is in idle state since the receiver is ready but sender does not offer valid data. Channel state changes in cycle 5, where the receiver is ready and word2 is immediately transferred. The same happens in the next cycle. The sender is not obliged to wait for the ready signal to be asserted before asserting the valid signal. However, once valid data are on the link they should not change until the handshake occurs.

In this example, and in the rest of the book we assume that data transfer occurs at the edges of the clock and all communicating modules belong to the same clock domain, which is a reasonable assumption and holds for the majority of the cases. When the sender and the receiver belong to different clock domains, some form of synchronization needs to take place before the receiver actually receives the transmitted word. A concrete description synchronization-related issues can be found in Ginosar (2011).

2.1 Elastic Buffers

The ready/valid handshake allows the sender and the receiver to stop their operation for an arbitrary amount of time. Therefore, some form of buffering should be implemented in both sides to keep the available data that cannot be consumed during a stall in either side of the link.

The elastic buffer is the most primitive form of a register (or buffer) that implements the ready/valid handshake protocol. Elastic buffers can be attached to the sender and the receiver as shown in Fig. 2.2. The EB at the sender implements a dual interface; it accepts (enqueues) new data from its internal logic and transfers (dequeues) the available data to the link, when the valid and ready signal are both equal to 1. The same holds for the EB at the receiver that enqueues new valid data when it is ready and drains the stored words to its internal logic (Butts et al. 2007).

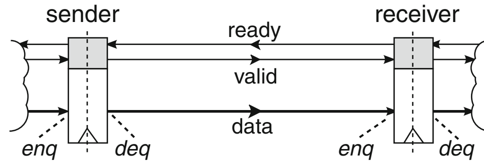


Fig. 2.2 An elastic buffer attached at the sender and the receiver's interfaces

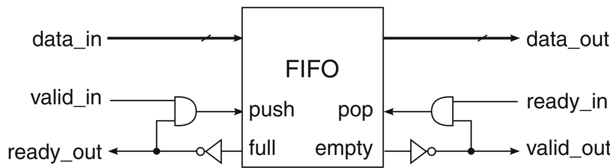
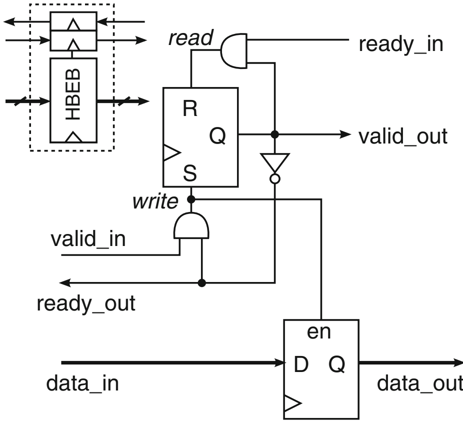


Fig. 2.3 An elastic buffer built around an abstract FIFO queue model

In an abstract form an EB can be built around a FIFO queue. An abstract FIFO provides a push and a pop interface and informs its connecting modules when it is full or empty. Figure 2.3 depicts how an abstract FIFO can be adapted to the ready/valid protocol both in the upstream and the downstream connections. The abstract FIFO model does not provide any guarantees on how a push to a full queue or a pop from an empty queue is handled. The AND gates outside the FIFO provide such protection. A push (write) is done when valid data are present at the input of the FIFO and the FIFO is not full. At the read side, a pop (read) occurs when the upstream channel is ready to receive new data and the FIFO is not empty, i.e., it has valid data to send. In both sides of the EB we can observe that a transfer to/from the FIFO occurs, when the corresponding ready/valid signals are both asserted (as implemented by the AND gates in front of the push and pop interfaces).

2.1.1 Half-Bandwidth Elastic Buffer

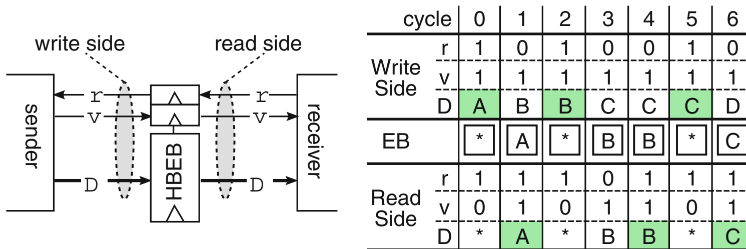
Using this abstract representation we can design EBs of arbitrary size. The simplest form of an EB can be designed using one data register and letting the EB practically act as a 1-slot FIFO queue. Besides the data register for each design we assume the existence of one state flip-flop F that denotes if the 1-slot FIFO is Full ($F = 1$) or Empty ($F = 0$). The state flip-flop actually acts as an R-S flip flop. It is set (S) when the EB writes a new valid data (push) and it is reset (R) when data are popped out of the 1-slot FIFO. Figure 2.4 depicts the organization of the primitive EB including also its abstract VHDL description. The AND gates connecting the full/empty signals of the EB and the incoming valid and ready signals are the same as in the abstract implementation shown in Fig. 2.3. Any R-S register can be implemented




```
valid_out <= full;
ready_out <= not(full);

process(clk)
begin
  if rising_edge(clk) then
    if valid_in='1' and full='0' then
      -- write
      full <= '1';
      data_out <= data_in;
    elsif ready_in='1' and full='1' then
      -- read
       full <= '0';
    end if;
  end if;
end process;
```

Fig. 2.4 The primitive 1-slot elastic buffer



cycle		0	1	2	3	4	5	6
Write Side	r	1	0	1	0	0	1	0
	v	1	1	1	1	1	1	1
	D	A	B	B	C	C	C	D
EB		*	A	*	B	B	*	C
Read Side	r	1	1	1	0	1	1	1
	v	0	1	0	1	1	0	1
	D	*	A	*	B	B	*	C

Fig. 2.5 Data transfer between two flow-control channels connected via a half-bandwidth elastic buffer

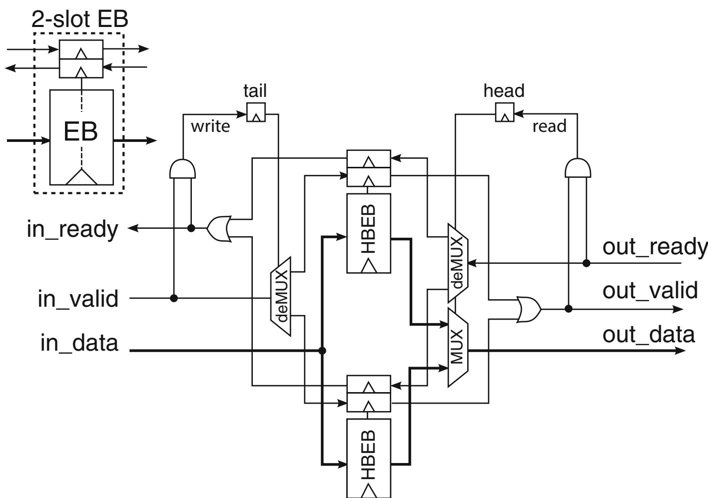
using simple registers after deciding on how the circuit will function when R and S are both asserted (giving priority to R, giving priority to S or keeping the register's old value). For the implementation shown in Fig. 2.4, we assume that set (write) has the highest priority although read (R) and write (S) cannot be asserted simultaneously.

The presented EB allows either a push or a pop to take place in each cycle, and never both. This characteristic imposes 50 % throughput on the incoming and the outgoing links, since each EB should be first emptied in one cycle and then filled with new data in the next cycle. Thus, we call this EB a Half-Bandwidth EB (HBEB). A running example of data transfers that pass through a HBEB is shown in Fig. 2.5. The HBEB, although slower in terms of throughput, is very scalable in terms of timing. Every signal out of the HBEB is driven by a local register and no direct combinational path connects any of its inputs to any of its outputs, thus allowing the safe connection of many HBEB in series.

2.1.2 Full-Bandwidth 2-Slot Elastic Buffer

The throughput limitation of HBEB can be resolved by adding two of them in each EB stage and using them in a time-multiplexed manner. In each cycle, data are written in one HBEB and read out from the second HBEB thus giving the impression in the upstream and the downstream channel of 100 % of write/read throughput.

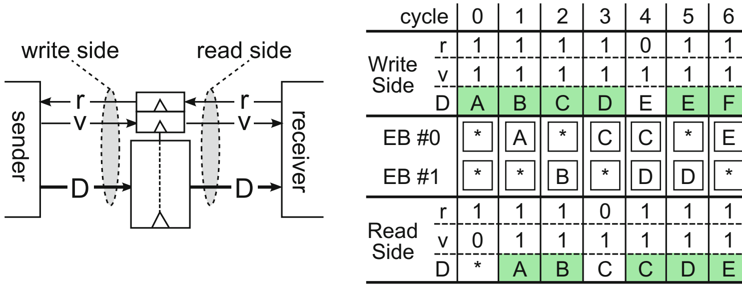
The design of the 2-slot EB that consists of two HBEBs needs some additional control logic that indexes the read and writes position; the organization of the 2-slot EB is shown in Fig. 2.6 along with its abstract VHDL description. When new data




```vhdl
valid_out <= full(0) or full(1);
ready_out <= not(full(0)) or not(full(1));
data_out <= data(head);

process(clk)
begin
  if rising_edge(clk) then
    -- write
    if valid_in='1' and full(tail)='0' then
      full(tail) <= '1';
      data(tail) <= data_in;
      tail <= not(tail);
    end if;
    -- read
    if ready_in='1' and full(head)='1' then
       full(head) <= '0';
       head <= not(head);
    end if;
  end if;
end process;
```

Fig. 2.6 The organization and the abstract VHDL description of a full-throughput 2-slot EB using two HBEBs in parallel that are accessed in a time interleaved manner as guided by the head (for read) and tail pointers (for write)



		cycle	0	1	2	3	4	5	6
Write Side	r		1	1	1	1	0	1	1
	v		1	1	1	1	1	1	1
	D		A	B	C	D	E	E	F
EB #0			*	A	*	C	C	*	E
EB #1			*	*	B	*	D	D	*
Read Side	r		1	1	1	0	1	1	1
	v		0	1	1	1	1	1	1
	D		*	A	B	C	C	D	E

Fig. 2.7 Data transfer between two flow-control channels connected via a 2-slot elastic buffer that consists of a parallel set of HBEBs

are pushed in the buffer they are written in the position indexed by the 1-bit tail pointer; on the same cycle the tail pointer is inverted pointing to the next available buffer. Equivalently, when new data are popped from the buffer, the selected HBEB is indexed by the 1-bit head pointer. During the dequeue the head pointer is inverted. The 2-slot EB has valid data when at least one of the HBEB holds valid data and it is ready when at least one of the two HBEBs is ready. The incoming valid and ready signals are transferred via de-multiplexers to the appropriate HBEB depending on the position shown by the head and tail pointers.

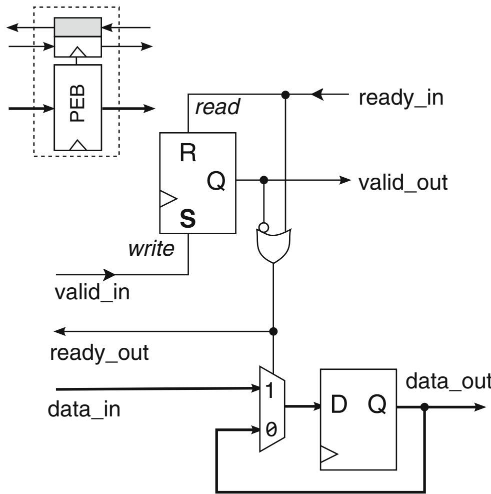
In overall the 2-slot EB offers 100 % throughput of operation, fully isolates the timing paths between the input and output handshake signals and constitutes a primitive form of buffering for NoCs. A running example of the 2-slot EB connecting two channels is shown in Fig. 2.7.

2.1.3 *Alternative Full-Throughput Elastic Buffers*

Full throughput operation does not need necessarily 2-slot EBs and can be achieved even with 1-slot buffers that introduce a throughput-timing scalability tradeoff. The 1-slot EBs presented in this section can be designed by extending the functionality of the HBEB in order to enable higher read/write concurrency.

The first approach increases the concurrency on the write port (push) of the HBEB. New data can be written when the buffer is empty (as in the HBEB) or if it becomes empty in the same cycle (enqueue on full if dequeue in the same cycle). Adding this additional condition in the write side of the HBEB results in a new implementation shown in Fig. 2.8 and called pipelined EB (PEB) according to the terminology used in Arvind (2013). The PEB is ready to load new data even when at Full state, given that a pop (ready_in) is requested in the same cycle, thus offering 100 % of data-transfer throughput.

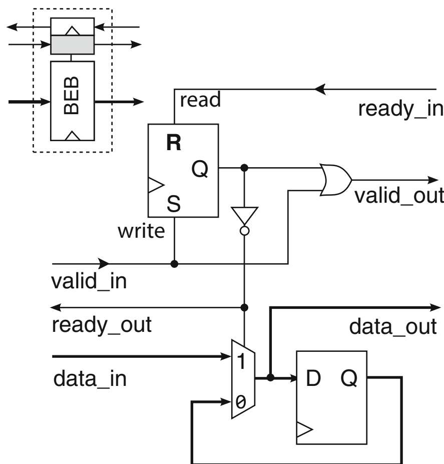
The second approach, called a bypass EB (BEB), offers more concurrency on the read port. In this case, a pop from the EB can occur even if the EB does not have




```
valid_out <= full;
ready_out <= not(full) or ready_in;

process(clk)
begin
  if rising_edge(clk) then
    if valid_in='1' then
      -- write
      full <= '1';
    elsif ready_in='1' then
       -- read
       full <= '0';
    end if;
    -- data reg
    if ready_out='1' then
       data_out <= data_in;
    end if;
  end if;
end process;
```

Fig. 2.8 The pipelined EB that offers full throughput of data transfer and introduces direct combinational paths between ready_in and ready_out backward notification signals




```
data_out <= data_in when full='0' else
            data_r;
valid_out <= full or valid_in;
ready_out <= not(full);

process(clk)
begin
  if rising_edge(clk) then
    if ready_in='1' then
      -- read
      full <= '0';
    else
       -- write
       full <= valid_in;
    end if;
    -- data reg
    data_r <= data_out;
  end if;
end process;
```

Fig. 2.9 The bypass EB that offers full throughput of data transfer and introduces direct combinational paths between data_in (valid_in) and data_out (valid_out) forward signals

valid data, assuming that an enqueue is performed in the same cycle (dequeue on empty if enqueue). In order for the incoming data to be available to the output of the BEB on the same cycle, a data bypass path is required as shown in Fig. 2.9. The bypass condition is only met when the EB is empty. In the case of the BEB, the priority of the R-S state flip flop is given to Reset.

Both buffers solve the low bandwidth problem of the HBEB and can propagate data forward at full throughput. However, certain handshake signals propagate via

a fully combinational logic path. This characteristic is a limiting factor in terms of delay since in large pipelines of EBs, possibly spanning across many NoC routers, the delay due to the combinational propagation of the handshake signals may exceed the available delay budget.

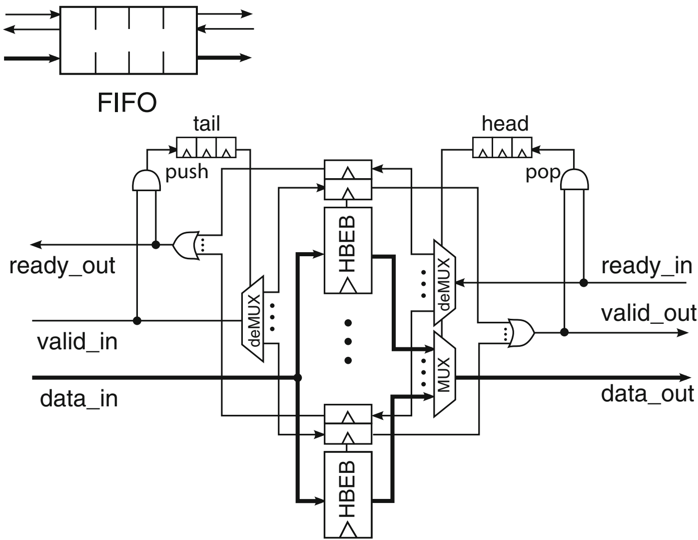
The design of a 2-slot EB can be alternatively achieved by connecting in series a pair of bypass EB and a pipelined EB. This organization leads to the designs presented in Cortadella et al. (2006) where the 2-slot EB have been derived using FSM logic synthesis. Also, in the same paper, it was shown how to implement a 2-slot EB using 2 latches in series, a main and an auxiliary one, by controlling accordingly the clock phases, and the transparency of each latch.

2.2 Generic FIFO Queues

Even if the sender and the receiver can be "synchronized" by exchanging the necessary flow control information via the ready/valid signals, the designer still needs to answer several critical questions. For example, how can we keep the sender busy before the receiver is stalled? The direct answer to this question is to replace simple 2-slot EBs with larger FIFO buffers that will store many more incoming words and implement the same handshake. In this way, when the receiver is stalled the sender can be kept busy for some extra cycles. If the receiver remains stalled for a long period of time then inevitably all the slots of the buffer will be occupied and the sender should be informed and stop transmission. Actually, FIFOs are needed to absorb any bursty incoming traffic at the receiver and effectively increase the overall throughput, since the network can host a larger number of words per channel before being stalled.

Larger FIFOs can be designed by adding more HBEBs in parallel and by enhancing the tail and head pointers to address a larger set of buffer positions for a push or a pop, respectively (Fig. 2.10). When new data are pushed in the FIFO they are written in the position indexed by the tail pointer; in the same cycle the tail pointer is increased (modulo the size of the FIFO buffer) pointing to the next available buffer. Equivalently, when new data are popped from the FIFO, the selected EB is indexed by the head pointer. During the dequeue the head pointer is increased (modulo the size of the FIFO buffer). The ready and valid signals sent outside the FIFO are generated in exactly the same way as in the case of the 2-slot EB. If the head and tail pointers follow the onehot encoding, their increment operation does not include any logic and can be implemented using a simple cyclic shift register (ring counter).

Designing a FIFO queue using multiple HBEBs in parallel can scale efficiently to multiple queue positions. However, the read (pop) path of the queue involves a large multiplexer that induces a non-negligible delay overhead. This read path can be completely isolated by adding a 2-slot EB at the output of the parallel FIFO, supported also by the appropriate bypass logic shown in Fig. 2.11. When the FIFO is empty, data are written to the frontmost 2-slot EB. The parallel FIFO starts to fill




```
valid_out <= or(full);
ready_out <= or(not(full);
data_out <= data(head);

process(clk)
begin
  if rising_edge(clk) then
    -- write
    if valid_in='1' and full(tail)='0' then
      full(tail) <= '1';
      data(tail) <= data_in;
      tail <= tail+1;
    end if;
    -- read
    if ready_in='1' and full(head)='1' then
       full(head) <= '0';
       head <= head+1;
    end if;
  end if;
end process;
```

Fig. 2.10 The organization of a FIFO queue using many HBEBs in parallel and indexing the push and pop operations via the tail and head pointers

with new data once the output EB becomes full. During a read the output interface checks only the words stored in the EB. When the output EB becomes empty, automatically data is transferred from the parallel FIFO to the output EB without waiting any event from the output interfaces. The output EB should be seen as an extension of the capacity of the main FIFO by two more positions.

For large FIFOs the buffer slots can be implemented by a two-port SRAM array that supports two independent ports one for writing (enqueue) and one for reading (dequeue). The read and write addresses are driven again by the head and tail pointers and the control FSM produces the signals needed to interface the FIFO

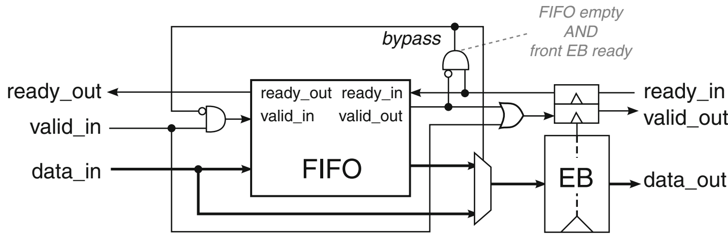


Fig. 2.11 A bypassable FIFO extended by a 2-slot EB to isolate the read delay path of the FIFO from the rest modules of the system

with other system modules. SRAM-based buffers offer higher storage density than the register-based implementation. In small buffers of 4–8 slots little benefits are expected and both design options have the same characteristics.

2.3 Abstract Flow Control Model

Having described in detail the ready/valid flow control mechanism and the associated buffer structures that will be used in the developed NoC routers, in this section, we will try to build a useful abstraction that will help us understand better the operation of flow control and to clarify the similarities and differences between the various flow control policies that are used widely today in real systems.

Every FIFO or simpler EB with 1 or 2 slots that implements a ready/valid handshake can be modeled by an abstract flow-control model that includes a buffer of arbitrary size that holds the arriving data and a counter. The abstract model for a ready/valid flow-controlled channel is depicted in Fig. 2.1a. The counter counts the free slots of the associated buffer, and thus, its value can move between 0 and the maximum buffer size. The free-slots counter is updated by the buffer to increase its value, when a new data item has left the buffer (update signal), and also notified by the link to reduce its value, when a new data item has arrived at the buffer (incoming valid signal). When the valid and the update signal are asserted in the same cycle the number of free slots remains unchanged. The counter is responsible for producing the ready signal that reveals to the sender the availability of at least one empty position at the buffer.

The counter is nothing more than a mechanism to let the receiver manage the available buffer slots it has available. In the buffers we have presented so far, this counting procedure is implicitly implemented by the full flags of each EB or the head and tail pointers.

Following the developed abstraction, we observe that there is no need for the free slots counter to be associated directly with the receive part of the link and can be placed anywhere, as shown in Fig. 2.12b, assuming that the connections with the receiver (status update when a new data item leaves the buffer) and with the sender

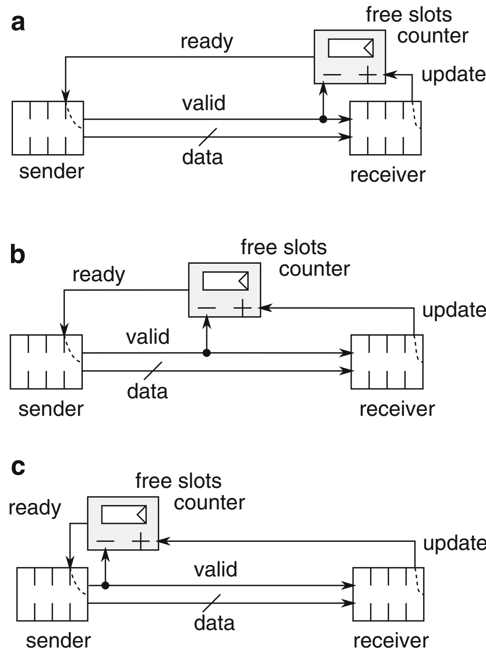


Fig. 2.12 An abstract model of a ready-valid flow-controlled link including a free-slot counter (**a**) at the receiver's side, (**b**) in the middle and (**c**) at the sender

(ready signal that shows buffer availability and valid signal that declares the arrival of a new data item at the receiver's buffer) do not change.

Equivalently, we could move this counter at the sender's side (see Fig. 2.12c). Then, the valid signal from the sender and the ready signal from the counter are both local to the sender. The read signal tells to the sender which it can send a new data item, while the receiver sends to the counter only the necessary status update signals that denote the removal of a data item from the receiver's buffer. Assuming that the counter reflects the number of empty slots at the receive side, it should be incremented when it receives a new status update from the receiver and decremented when the sender wants to transmit a new data item. Also, the receiver is ready to accept new data when the value of the free slots counter is larger than zero.

2.4 Credit-Based Flow Control

When the counter is attached to the sender, the derived flow control policy is called credit-based flow control and gives to the sender all the necessary knowledge to start, stop, and resume the transmission (Kung and Morris 1995; Dally and Towles 2004). In credit-based flow control, the sender explicitly keeps track of the available

buffer slots of the receiver. The number of available slots is called credits and they are stored at the sender side in a credit counter. When the number of credits is larger than zero then the sender is allowed to send a new word consuming one available credit. At each new transmission the credit counter is decremented by one reflecting that one less buffer slot at the receive side is now available. When one word is consumed at the receive side, leaving the input buffer of the receiver, the sender is notified via a credit update signal to increase the available credit count.

An example of the operation of the credit-based flow control is shown in Fig. 2.13. At the beginning the available credits of the sender are reset to 3 meaning that the sender can utilize at most 3 slots of the receiver's buffer. When the number of available credits is larger than 0 the sender sends out a new word. Whenever the sink of the receiver consumes one new word, the receiver asserts a credit update signal that reaches the sender one cycle later and it increases the credit counter. The credit updates, although arrive with cycle delay, they are immediately consumed in the same cycle. This immediate credit reuse is clearly shown in the clock cycles where the available credits are denoted as 0*. In those clock cycles, the credit counter that was originally equal to 0 stays at 0, since, it is simultaneously incremented due to credit update and decremented due to the transmission of a new word. When the words are not drained at the sink they are buffered at the receiver. No word can be dropped or lost since each word reaches the receiver after having first consumed the credit associated with a free buffer position.

2.5 Pipelined Data Transfer and the Round-Trip Time

When the delay of the link exceeds the desired clock period we need to cut the link to smaller parts by inserting the appropriate number of pipeline registers. In this case, it takes more cycles for the signals to propagate in both the forward and the backward direction. This may also happen when the internal operation of the sender and the receiver, requires multiple cycles to complete as done in the case of pipelined routers that will be elaborated in the following chapters.

An example of a flow-controlled data transmission over a pipelined link is shown in Fig. 2.14, where the valid and ready pass through the pipeline registers at the middle of link before reaching the receiver and the sender, accordingly. The sender, before sending new data on the link by asserting its valid signal, should check its local ready signal, i.e., the delayed version of the ready generated by the receiver.[1] If the sender asserted the valid signal irrespective the value of the incoming ready signal, then either the transmitted words would have been dropped, if the receiver's buffer was full, or, multiple copies of the same data would have been written in the receiver, if buffer space was available. The second scenario occurs because the sender is not aware of the ready status of the receiver and it does not dequeue the corresponding data.

[1]This is not needed in the case that two flow-controlled buffers communicate directly without any intermediate pipeline registers.

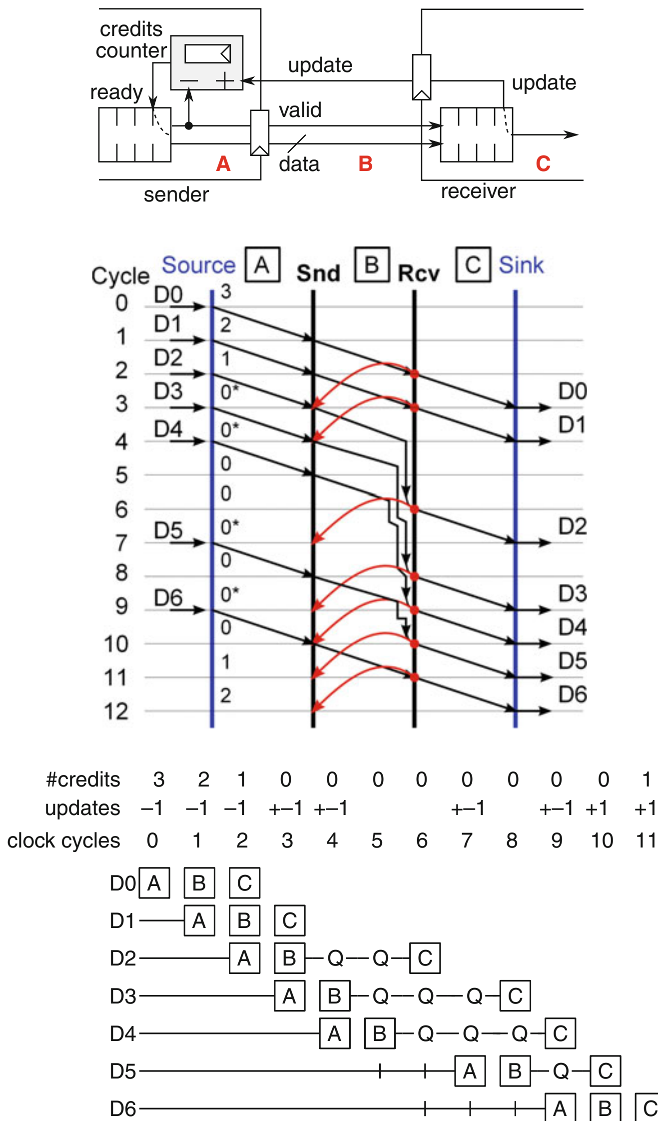


Fig. 2.13 An example of data transfers on a link between a sender and a receiver governed by credit-based flow control. The figure includes the organization of the sender and receiver pair and the flow of information in time and space

When the receiver stops draining incoming data, 5 words are assembled at its input queue. If the receiver supported fewer positions some of them would have been lost and replaced by newly arriving words. As shown by the example of Fig. 2.14, in

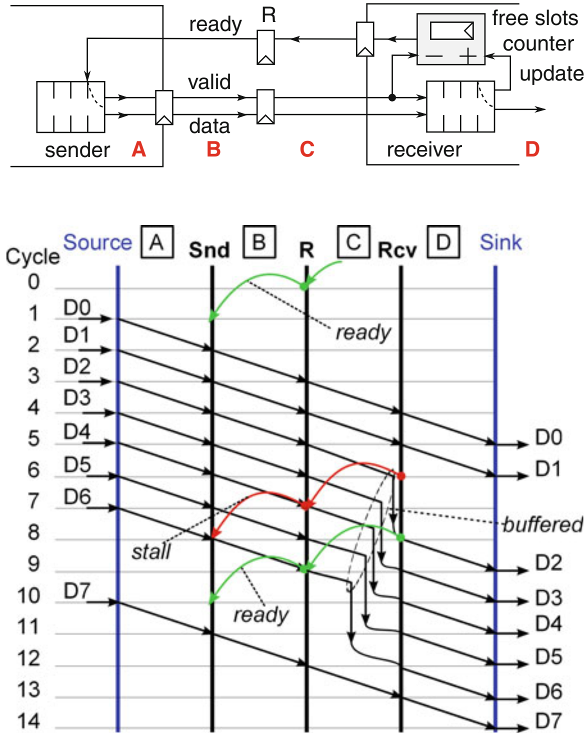


Fig. 2.14 An example of data transfers on a pipelined link between a sender and a receiver governed by ready/valid flow control

the case of pipelined links, the FIFO buffer at the receiver's side needs to be sized appropriately in order to guarantee safe lossless operation, i.e., every in flight word finds a free buffer slot to use.

2.5.1 Pipelined Links with Ready/Valid Flow Control

The latency experienced by the forward and the backward flow control signals affect not only the correct operation of the link but also the achieved throughput, i.e., the number words delivered at the receiver per cycle. The behavior of the flow control mechanism and how the latency and the slots per buffer interact will be highlighted in the following paragraphs.

Let L_f and L_b denote the number of pipeline registers in the forward and in the backward direction, respectively, in the case of a pipelined flow-controlled

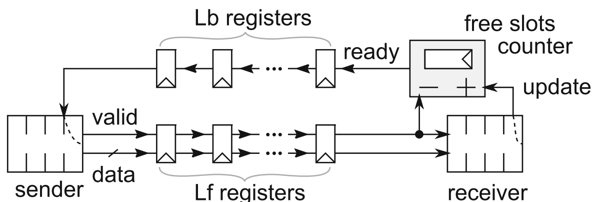


Fig. 2.15 An abstract model of a pipelined link with L_f registers in the forward direction and L_b registers in the backward direction governed by the ready/valid flow control

link.[2] The equivalent flow-control model for the ready/valid handshake is shown in Fig. 2.15. The slot counter that belongs to the receiver measures the number of available buffer slots. The slot counter gets updated (incremented) with zero latency, while it gets decremented (a new valid word arrives at the receiver) with a delay of L_f cycles relative to the time that a new word has left the sender. Also, the ready signal that is generated by the slot counter reaches the sender after L_b cycles. Please note that once the ready signal reaches the sender it can be directly consumed in the same cycle thus not incurring any additional latency. Equivalently, at the receiver, the arrival of a new word can stop the readiness of the receiver in the same cycle. This behavior at the sender and at the receiver is depicted by the dotted lines in Fig. 2.15.

At first, we need to examine how many words the buffer of the receiver can host to allow for safe and lossless operation. Let's assume that the receiver declares its readiness via the ready signal; ready is set to 1 when there is at least one empty slot, e.g., freeSlots$>$ 0. When the buffer at the receiver is empty, the counter asserts the ready signal. The sender will observe the readiness of the receiver after L_b cycles and immediately starts to send new data by asserting its valid signal. The first data item will arrive at the receiver after $L_f + L_b$ cycles. This is the first time that the receiver can react by possibly de-asserting its ready signal. If this is done, i.e., ready $= 0$, then under the worst case assumption, the receiver should be able to accept the $L_f - 1$ words that are already on the link plus the L_b words that may arrive in the next cycles; the sender will be notified that the receiver is stalled L_b cycles later. Thus, once the receiver stalls, it should have at least $L_f + L_b$ buffers empty to ensure lossless operation.

Even if we have decided that $L_f + L_b$ positions are required for safe operation the condition on which the ready signal is asserted or de-asserted needs further elaboration. Assume for example that the receiver has the minimum number of buffer slots required, i.e., $L_f + L_b$. We have shown already that once the ready signal makes a transition from 1 to 0 it means that in the worst case $L_f + L_b$ words

[2]The internal latency imposed by the sender and receiver can be included in L_f and L_b respectively.

may arrive. Therefore, if all the available words are equal to $L_f + L_b$ the ready is asserted only when the buffer at the receiver is empty, i.e., freeSlots=$L_f + L_b$.

Although this configuration allows for lossless operation, it experiences limited transmission throughput. For example, assume that the receiver is full, storing $L_f + L_b$ words, and stalled. Once the stall condition is removed, the receiver starts dequeuing one word per cycle. The ready signal is equal to 0 until all $L_f + L_b$ words are drained. In the meantime, although free slots exist at the receiver, they are left unused until the sender is notified that the stall is over and new words can be accepted. After $L_f + L_b$ cycles, all $L_f + L_b$ slots are emptied and the ready signal is set to 1. However, any new words will only arrive after $L_f + L_b$ cycles. During this time frame the receiver remains idle having its buffer empty. Therefore, in a time frame of $2(L_f + L_b)$ cycles the receiver was able to drain $L_f + L_b$ words. This behavior translates to a throughput of 50 %.

More throughput can be gained by increasing the buffer size of the receiver to $L_f + L_b + k$ positions. In this scenario we can relax the condition for the assertion of the ready signal to: ready = 1 when freeslots$\geq L_f + L_b$ (from just equality in the baseline case). Therefore, if the buffer at the receiver is full with $L_f + L_b + k$ words at time t_0, $L_f + L_b$ words should leave to allow the ready signal to return to one. $L_f + L_b$ cycles later the first new words will arrive due to the assertion of the ready signal. In the meantime the receiver will be able to drain k more words. Therefore, the throughput seen at the output of the receiver is $\frac{L_f + L_b + k}{2(L_f + L_b)}$. The throughput can reach 100 % when $k = L_f + L_b$; the receiver has $2(L_f + L_b)$ buffer slots and ready is asserted when the number of empty slots is at least $L_f + L_b$.

The derived bounds hold for the general case. However, if we take into account some small details that are present in most real implementations the derived bounds can be relaxed showing that the ready/valid handshake protocol achieves full throughput with slightly less buffer requirements.

First the minimum number of buffers required to achieve lossless operation can drop from $L_f + L_b$ to $L_f + L_b - 1$. This reduction is achieved since a ready = 0 that reaches the sender can stop directly the transmission of a new word (as shown by the dotted lines of Fig. 2.15) at the output of the sender. Therefore, the actual in-flight words in the forward path are $L_f - 1$ and not L_f since the last one is actually stopped at the output register of the sender itself. Therefore, the ready signal out of the receiver is computed as follows: ready = 1 when freeSlots = $L_f + L_b - 1$ else 0.

Second, when a new word is dequeued from the receiver the slot counter is updated in the same cycle. In this case, when a receiver with $k + (L_f + L_b - 1)$ buffers is full and starts dequeuing one word per cycle, it will declare its readiness the same time that it dequeues the $(L_f + L_b - 1)$th word. The first new word due will arrive $L_f + L_b - 1$ cycles later. Thus, during $2(L_f + L_b - 1)$ clock cycles the receiver can drain $k + (L_f + L_b - 1)$ words. When $k = L_f + L_b - 1$ the receiver can achieve 100 % throughput; when the $L_f + L_b - 1$th word is dequeued the first new word is enqueued thus leaving no gaps at the receiver's buffer.

Primitive Cases

The derived results can be applied even to the simple EBs presented in the beginning of this chapter. An equivalent flow-control model for a 2-slot EB that operates under ready/valid handshake experiences a forward latency of $L_f = 1$ due to the register present at the output of the HBEBs and a backward latency $L_b = 1$, since the ready signal is produced by the full flags of the HBEBs. According to the analysis presented, this configuration needs $L_f + L_b - 1 = 1$ buffer for lossless operation and 2 times that for 100 % throughput as already supported by the 2-slot EB. The configuration that uses only 1 slot, while keeping L_f and L_b equal to 1, corresponds to the HBEB that offers lossless operation while allowing only for 50 % of link-level throughput.

The derived model does not cover the degenerate case of 1-slot pipelined and bypass EBs. For example even if the pipelined EB has $L_b = 0$ (fully combinational backpressure propagation) and $L_f = 1$, the ready backepressure signal spans multiple stages of buffering and extend the borders of a single sender-receiver pair.

2.5.2 Pipelined Links with Elastic Buffers

As shown so far the use of simple pipeline registers between two flow-controlled endpoints increases the round-trip time of the flow control mechanism and necessitates the use of additional buffering at the receiver to accommodate all in-flight words. In a NoC environment, it is possible and also desirable to replace the forward and backward pipeline registers with flow-controlled EB stages, thus limiting the flow-control notification cycle per stage (Concer et al. 2008; Michelogiannakis and Dally 2013).

Figure 2.16a shows a pipelined link that uses only pipeline registers and needs 10 buffers at the receiver for achieving 100 % throughput and safe operation. Recall that in this pipelined configuration the sender sets valid = 1 when it observes locally a ready signal equal to 1 to avoid the receiver writing by mistake multiple copies of the same word. If one stage of the pipeline is transformed to an EB, as shown in Fig. 2.16b, then the round-trip time at the second part of the link reduces by 2 and thus a buffer with 6 slots suffices for the receiver. By extending this approach to all pipeline stages, the same operation can be achieved by the architecture shown in Fig. 2.16c where the pipelined link consists of only EBs. In this case, the buffer at the receiver can have only a 2-slot EB, since it experiences a local $L_f = L_b = 1$ at the last stage of the link. In overall, this strategy achieves both to isolate the timing paths by registering both the data and the backpressure signals and to reduce the total number of buffers required for lossless and full throughput communication. In fact, in this example, with using only 3 stages of 2-slot EBs and one 2-slot EB at the receiver achieves the same behavior as in the baseline case of Fig. 2.16a using 8 buffers in total that are distributed at the receiver and on the link.

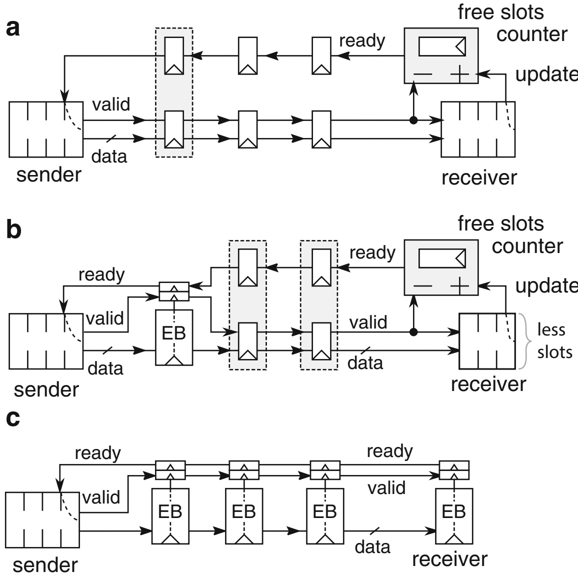


Fig. 2.16 The replacement of pipeline registers with 2-slot EBs on the link

In general, if k pairs of forward and backward registers are replaced by k 2-slot EBs, L_f and L_b are reduced by k. Thus, the worst case buffering at the receiver reduces from $2(L_f + L_b - 1)$ to $2((L_f - k) + (L_b - k) - 1)$. If we sum to this number the amount of buffering present on the link, i.e., the k 2-slot EBs, we end up having $2(L_f + L_b - 1) - 2k$ buffers in total (both at the receiver and on the link). Therefore, under ready/valid flow-control the use of EBs in the place of pipelining stages distributed across the link is always beneficial in terms of buffering and should be always preferred.

2.5.3 *Pipelined Links and Credit-Based Flow Control*

The equivalent flow control model for credit-based flow control on pipelined links is depicted in Fig. 2.17. In this case, the ready signal produced by the credit counter and the valid signal that consumes the credits are generated locally at the sender with zero latency. On the contrary, the credit update signal reaches the credit counter through L_b registers. The same holds also for the data in the forward direction that pass through L_f registers to get to the sender. At this point a critical detail needs to be pointed out. With ready/valid handshake both the valid signal that decreased the number of free slots and the associated data had to go through the same number of registers after leaving the sender. However, in this case, the valid signal that consumes the credit sees zero latency, while the data arrive at the receiver after L_f

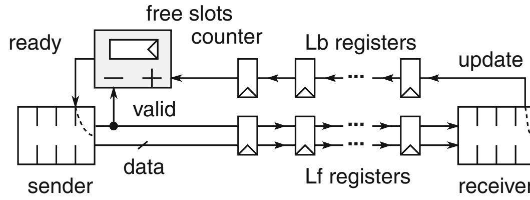


Fig. 2.17 The abstract flow-control model of a pipelined link using credit-based flow control

cycles. Therefore, credit consumption and data transmission experience different latencies. This detail will be extremely useful in understanding how to compute the required number of buffers for achieving full throughput in pipelined router implementations described in later chapters.

The throughput of transmission is closely related to the number of credits used by the sender and the number of clock cycles that pass from the time a credit update leaves the sender until the first new word that consumes this returned credit reaches the input buffer of the receiver.

This relation is illustrated by the examples shown in Fig. 2.18. In the first case, the sender has only one credit available (possibly meaning that the receiver has only 1 buffer slot). It directly consumes this credit and sends out a new word in cycle 0. Once the word reaches the receiver it is immediately drained from the receiver's buffer and a credit update is sent in the backward direction. The update needs one cycle to reach the sender. Immediately the source consumes this credit update by sending out a new word. Due to the forward and the backward delay of the data and the credit-update signal, the receiver is utilized only once every 3 cycles. By increasing the available credits to 2 that directly reflect more slots at the input buffer of the receiver, this notification gap is partially filled and the throughput is increased to 2/3. Finally, if the sender has 3 credits available the flow-control notification loop is fully covered and the transmission achieves 100 % throughput keeping the receiver busy in each cycle.

In the general case of having a L_f registers in the forward path and L_b registers in the backward (credit update) path the minimum number of buffers needed at the receiver under credit-based flow control to guarantee lossless operation and 100 % throughput is $L_f + L_b$. Lossless operation is offered by default when using credit-based flow control without a minimum buffering requirement, since the sender does not send anything when there is not at least one available position at the receiver side. As far as maximum throughput is concerned, when the receiver sends backwards a credit update, a new word will arrive at the receiver that consumed this credit after $L_f + L_b$ cycles (L_b cycles are needed for the credit update to reach the sender plus L_f cycles for the new word to reach the receiver). Therefore, the number of words that will arrive at the receiver in a time window of $L_f + L_b$ cycles is equal to the number of credit updates sent backwards leading a throughput of $\frac{\#creditupdates}{L_f+L_b}$. Full throughput requires the number of credit updates being equal to $L_f + L_b$ reflecting an equal number buffer slots at the receiver.

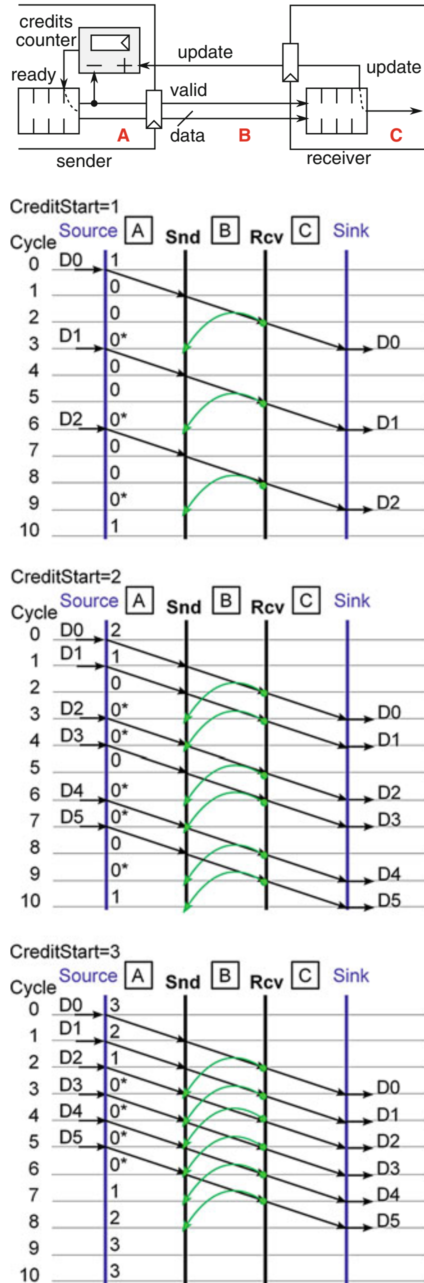


Fig. 2.18 An example of data transfer using credit-based flow control when the number of available credits is varied relative to the round-trip time (sum of forward and backward update latency)

By changing the available credits relative to the round-trip delay of $L_f + L_b$, the designer can adapt dynamically the rate of communication of each link. This feature can be easily applied for allowing dynamic power adaptivity both on the NoC links

as well as the buffers. Switching activity on the links besides data correlations is directly related to the rate of new valid words appearing on the wires of the link, while the buffers not used due to less credits can be gated to save dynamic (clock gating) or idle power (power gating).

2.6 Request–Acknowledge Handshake and Bufferless Flow Control

Similar to ready/valid handshake, link level flow control can be implemented using another 2-wire handshake protocol, called the req/ack protocol (Dally and Towles 2004; Bertozzi and Benini 2004). A request is made by the sender when it wants to send a new valid word, while an acknowledgment (ack = 1) is returned by the receiver when the word is actually written at the receiver. Equivalently, when there is no buffer space available to store the new word a not acknowledgement (ack = 0 or nack) is sent back to the sender. With req/ack protocol the sender is not aware of receiver's buffer status as done in ready/valid or credit-based flow control protocols. Therefore, every issued request is always optimistic meaning "data are sent". The sender after issuing a request has two choices: Either to wait for an ack, possibly arriving in the next cycles, before placing next available data on the channel or to actually send new data and manage possible nacks as they arrive.

In the first case that the sender waits for an ack throughput is limited to 50 % since a new transaction can begin every other cycle (one cycle to request, one cycle to wait for an ack before trying to send a new piece of data). In the second version, the sender puts a word in the channel in cycle i as long as an ack was received in cycle $i - 1$ referring to a previous transmission. The next cycle, since no ack has returned the sender prepares a new word to put on the channel. The previous one is not erased but it is put on hold in an auxiliary buffer. If the receiver acknowledges the receipt of data, the word in the auxiliary buffer is erased and replaced by the current data on the channel. If not, the sender understands that the receiver was stalled and stops transmission. In the next cycles, it continues trying to send its data but now sends first the data in the auxiliary buffer that have not been acknowledged yet by the receiver and delays the propagation of new data. This primitive form of speculative req/ack protocol works just like a 2-slot EB which requires at least 2 extra places to hold the in-flight data (not acknowledged in this case). For larger round-trip times it can be proven that req/ack has the same buffering requirements as the ready/valid protocol (Dally and Towles 2004), unless other hybrid flow control techniques are employed (Minkenberg and Gusat 2009).

Another flow control strategy that was developed around the idea of minimum buffering (equal to one register per stage) is bufferless flow control (Moscibroda and Mutlu 2009). Bufferless flow control is a degenerate case of req/ack flow control, where data that cannot be written at the receiver (that would have not been acknowledged) are not kept at the sender and are dropped. In the next cycle, new

data take their place and some upper-level protocol should care for retrieving the lost data. Dropping in a NoC environment is of limited importance since the complexity involved in retrieving the lost data is not substantiated by the hardware savings of bufferless flow control; if the buffering cost at the sender or the receiver cannot exceed the cost of one register, 1-slot EBs (or a 2-slot EB implemented with latches) can be used that allow for lossless and full throughput operation.

2.7 Wide Message Transmission

On-chip processing elements may need to exchange wide piece of information. Transferring wide messages in a single cycle may require close to thousands of wires between a sender and a receiver. Such amount of wiring is hard to handle especially in the case of an automated placement and routing design flow that performs routing in an unstructured row-based substrate of placed gates and registers. Besides the physical integration challenges that such wide links may cause, their utilization will always be under question. In real systems a variety of messages is transferred from time to time. Small memory request messages can be of the order of tens of bytes or less, while long reply messages can carry hundreds of bytes. Therefore, making the links equally wide to the largest message that the system can support is not a cost effective solution and would leave the majority of the wires undriven most of the time. Common practice keeps the link width close to the width of the most commonly used message that is transferred in the system and impose the larger messages to be serialized and pass the link in multiple clock cycles.

Wide messages are organized as packets of words. The first word, called the header of the packet, denotes the beginning of the packet and contains the identification and addressing information needed by the packet, including the address of its source and its destination. The last word of the packet is called the tail word and all intermediate words are called the body words. Each packet should travel on each link of the network as a unified entity since only the header of the packet carries all necessary information about the packet's source and destination. To differentiate from the words of a processor, the words that travel on the network are described with the term flit (derived from flow-control digit). An example packet format is depicted in Fig. 2.19.

Figure 2.19 depicts the wires needed in a network-on-chip channel that supports many flit packets. Besides data wires and necessary flow control signals (ready/valid is used in this example) two additional wires, e.g., isHead and isTail are needed that encode the type of the flit that traverses the channel per cycle. isHead and isTail signals are mutually exclusive and cannot be asserted simultaneously. When they are both inactive and valid = 1, it means that the channel holds a body flit.

Figure 2.20 depicts the transmission of two 4-flit packets over 3 links separated by 2 intermediate nodes. During packet transmission, when an output port is free the received word is transferred to the next output immediately without waiting the rest words of the packet, i.e., flit transmission is pipelined. The transfer of flits on

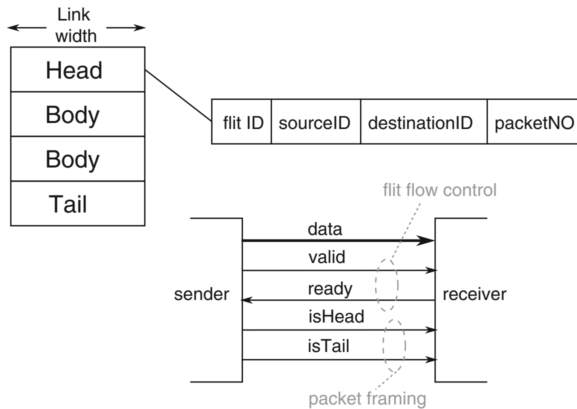


Fig. 2.19 The organization of packets and the additional signals added in the channel to distinguish the type of each arriving flit

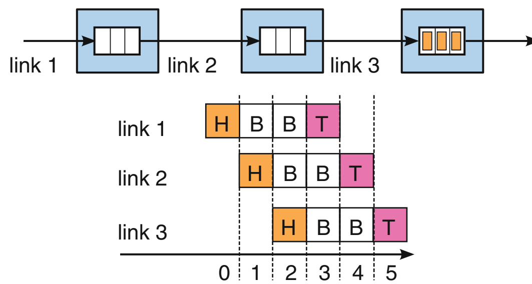


Fig. 2.20 The pipelined flow of the flits of three consecutives packets crossing the links between three nodes

each link follows the rules of the selected flow control policy independently per link. Therefore, the buffers at the end of each link should provide the necessary space for accommodating all incoming flits and offer full transmission throughput. Store-and-forward required the entire packet to reach each node before initiating next transmission for the next node.

The requirement of storing and not dropping the incoming flits to intermediate nodes raises the following question. How much free buffering should be guaranteed before sending the first word of a packet to the next node? The answer to this question has two directions. Virtual Cut Through (VCT) requires that the available downstream buffer slots to be equal to the number of flits of the packet (Kermani and Kleinrock 1979). With this technique, each blocked packet stays together and consumes the buffers of only one node since there is always enough room to fit the whole packet. On the contrary, wormhole (WH) removes this limitation and each node can host only a few flits of the packet (Dally and Seitz 1986). Then, inevitably, in the case of a downstream blocking, the flits of the packet will be spread out in the

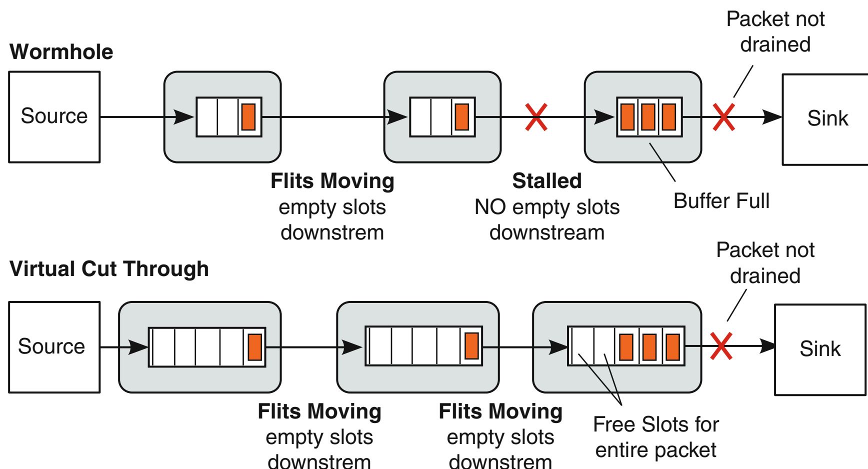


Fig. 2.21 The granularity of buffer allocation in virtual-cut throughput and wormhole packet flow policies

buffers of more than one intermediate node. This spreading does not add any other complication provided that all flits stay in their place and not dropped due to link-level flow control. The selection of a link-level flow control policy is orthogonal to either VCT or wormhole-based message flow. Any policy can be selected without any other complication to the operation of the system. Even if WH does not impose any limitation on the number of buffers slots required per link, still the round-trip time of each link sets the lower limit for high throughput data transfer.

The difference in the granularity of buffer allocation, per packet or per flit, imposed by the two policies is better clarified by the example shown in Fig. 2.21. Each flit moves to the downstream node as long as it has guaranteed an empty buffer either via ready/valid or credit-based flow control. Both VCT and wormhole employ pipelined transfers where each flit is immediately transferred to the next node irrespective the arrival of the next flits of the packet. When an output of a node is blocked, the flits continue moving closer to the blockage point until all buffers are full in front of them and oblige them to stop.

In the case of VCT, each intermediate node is obliged to have at least 5 buffer slots (equal to the number of flits per packet) that allows a whole packet to be stored in the blockage point. In the case of WH, arbitrary buffer slots can exist per node (the minimum number depends on the lossless property of the link-level flow control protocol). In our WH example, we selected to have 3 buffer slots per node. When all downstream buffers are full the flits cannot move and remain buffered in the node they are. In this way, the flits of the packet may occupy the buffers of a path of intermediate nodes. The way the granularity of buffer allocation (flit or packet level) affects the operation of NoC routers and how it can be actually implemented will be clarified in the following chapter.

2.8 Take-Away Points

Flow control is needed for guaranteeing lossless data transfer between any two peers and its operation is directly related to the selected buffering architecture. Buffers at the sender and the receiver can be from simple 1-slot and 2-slot elastic elements to more sophisticated FIFO queues that can host multiple in-flight words. Ready/valid and credit-based flow control are equivalent flow control mechanisms but with different characteristics in terms of their minimum buffering requirements. Pipelined links that increase the notification cycle of any flow-control mechanism increase also the minimum buffering requirements for supporting full throughput transmissions. The transmission of wide messages requires the packetization and the serialization of each message to packets of smaller flits that travel in the network one after the other passing all intermediate nodes in multiple cycles.