Chapter 8
High-Speed Allocators for VC-Based Routers

The packets arriving to a VC-based router should allocate two kinds of resources before being able to move to their destined output port. Each packet should allocate an output VC and each flit should guarantee exclusive access to a router's output port, on a cycle-by-cycle basis. The output VCs are allocated to the packets of the input VCs during VC Allocation (VA), while Switch Allocation (SA) decides which input VC will move to which output in each clock cycle. Both allocation operations evolve in two steps.

For example, VA is split in two parts called VA1 and VA2. VA1 decides locally the output VC that each input VC will ask for. In case that there are several candidate output VCs, VA1 performs a local arbitration step and selects only one of them. Thus after VA1 each input VC holds only one output VC request (the output port that the output VC belongs to is known beforehand by routing computation). In a baseline router organization, VA1 selects among output VC requests that are checked to be available; an input VC will never ask for an output VC that is not currently available. In a loosely coupled VA1 implementation, this availability check of output VCs is not always necessary. In any case, after VA1, the second stage of VC allocation, called VA2, is executed per output VC; each output VC receives at most one request from each input VC and grants only them. After VA2, a match between input VCs and output VCs has been derived that contains no conflicts. The matched output VCs become un-available and the matching input VCs are set to a lock state using the *outVCLock* state variables.

Equivalently, SA also evolves in two steps. In the first step, called SA1, one input VC from each input is selected to try to reach the selected output port of the router. Then, each output independent from the rest, in SA2, decides which input to select. After SA1 and SA2 the winning flits move to their destined output port. The movement is done via the per-input and per-output multiplexers that get configured by the grants produced by SA1 and SA2, respectively. Before SA begins, it is assumed that the input VCs have already allocated an output VC. This conservative assumption can be removed with care and let packets that have not yet allocated an

G. Dimitrakopoulos et al., *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*, DOI 10.1007/978-1-4614-4301-8_8

output VC to try to win in SA and in parallel to receive an output VC. Depending on the implementation, receiving an output VC in parallel to SA, can be done in several ways.

In the rest paragraphs, we will analyze in detail all the possible alternatives of implementing VA and SA, with the goal to minimize the delay of the allocation process by letting the intermediate steps to execute in parallel when possible. The material of this chapter should be considered as an enhancement of the basic allocation strategies presented in Chap. 7 that lead to high-speed router implementations.

8.1 Virtual Networks: Reducing the Complexity of VC Allocation

The baseline organization of a VC allocator requires a $V : 1$ arbiter per input VC for implementing VA1, followed by a $N \times V : 1$ arbiter per output VC for implementing VA2 and also some non trivial signal gathering, masking and distribution logic. The complexity of the VC allocator can be reduced significantly if the "freedom" enjoyed by each input VC is reduced. VCs can be used for the definition of virtual networks (VNs). Packets that belong to a VN complete their whole trip in the network it the same VN and jumping from a VN to another VN is either prohibited or done with very restrictive rules that are used to guarantee deadlock freedom (Azimi et al. 2009).

When a set VCs, say k, belong to a specific VN, the requests for VC allocation are restricted to the VCs of the same VN. In this case, VC allocation is split into parallel and smaller VC allocators where each one is serving NV/k input VCs. As shown in Fig. 8.1, each input VC sends the requests and receives grants from the smaller VC allocator (with NV/k inputs and NV/k outputs) that corresponds to the same VN.

In the minimum case that each VC is always a VN, the design of the VC allocator is further simplified since VA1 is completely removed. If an input VC does not want or is not allowed to change the VC that it belongs to, then VA1 is not needed, since

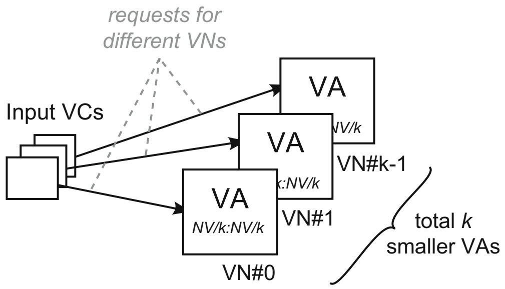


Fig. 8.1 The separation of the VCs of the network to VNs and restricting a packet from changing VN simplifies a lot the design of the VC allocator that now involves multiple parallel VC allocators that each one serves only the VCs of a specific VN

the *i*th input VC will always ask for the *i*th output VC. Even if the *i*th output VC is not available, the *i*th input VC does not have any other choice rather than waiting for the *i*th output VC to become available before trying to match to it.

8.2 Lookahead VA1

The second alternative for removing VA1 from the critical path of the VC allocator, but still allow packets to change VC in flight irrespective the VN they belong, is called lookahead VA1 (LVA1) and works similar to the lookahead routing computation. Instead of waiting each input VC to perform VA1, when it reaches the frontmost position of its input VC buffer, we allow VA1 to complete beforehand. Each input VC in parallel to the VA step performed in the previous router (or even in parallel to the SA of the previous router) selects which output VC will ask for when it reaches the next router; the selection is done via arbitration among the candidate output VCs and is stored at a special field of the packet's head flit. Thus, once a head flit of a packet reaches the frontmost position of the input VC buffer it can immediately participate in VA2 since it already stores in one of its fields the output VC requests, as depicted in Fig. 8.2. In parallel to VA2, each input VC performs VA1 for selecting the output VC that will ask for when it reaches the next router.

In lookahead VA1 each input VC is oblivious of the state of the output VCs. Therefore, there is the possibility that the selected output VC of the head flit to be un-available when the flit asks for it in the VA2 of the next router. In this case, the

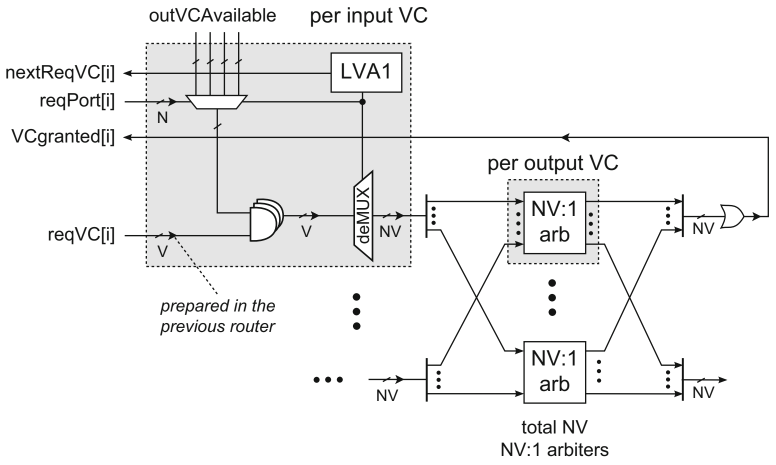


Fig. 8.2 The organization of the VC allocator that accepts the pre-computed output VC requests of each input VC that are directly propagated to VA2. In parallel, lookahead VA1 takes place in parallel and saves the output VC request that will be used in the next router in the head flit of the packet

head flit does not have any choice rather than to wait for the selected output VC to become available. Even if other eligible output VCs are available the head flit cannot change its output VC request decided during LVA1. Instead of performing the lookahead VA1 step in the previous router, VA1 can be also performed at the end of link traversal. In this way, the head flit decides for an output VC request at the time that it is written to the input VC buffer. If done this way, it is more easy for the head flit to know the status of the output VCs since it already reached the next router.

Inevitably, the lack of information about which output VCs are available during lookahead VA1, will cause several input VCs to pre-select and fight for the same output VC, even if there are other output VCs that are available to use. Depending on the application and the number of VCs in the network this feature may limit the throughput of the network. From our measurements only slight reductions are expected that can be possibly alleviated by the increased operating speed offered by lookahead VA1.

8.3 VC Allocation Without VA2: Combined Allocation

By either not letting a packet to change VC while it is traversing the network or performing VA1 in a lookahead manner, we achieved to remove VA1 from the critical path of VC allocation. In this case, the needed allocation steps that should be executed in series include VA2 to match an output VC to a certain input VC and then the two steps of SA that match an input VC to an output port on a cycle-by-cycle basis. Even in this reduced-complexity allocation organization, we assume that all requesting input VCs can be allocated simultaneously to available output VCs assuming that no other input VC is asking for the same output VC. However, we know that due to SA1 only one input VC will be allowed to leave the router from each input. This is a structural requirement imposed by the datapath of the router (the input VCs of the same input share an input of the crossbar). Therefore, there is no reason for letting more than one VCs per input to get matched to an output VC; at the end, at most one VC per input will be allowed to leave the router.

The restriction that at most one new VC per input is allowed to match to a new VC per output, can be applied by allocating an output VC only to the input VC that won in SA. In this way, the allocation of an output port in SA is accompanied by the allocation of an output VC. This combined allocation eliminates completely the VA2 stage of VC allocation (Lu et al. 2012). VA1 is still needed in order for every input VC to know beforehand which output VC to request, when it wins in SA.

From the previous discussion we know that the VA1 step can be performed either in series with SA, or using lookahead VA1. Since the selected output VC will be used directly for driving SA, it should be checked both for availability and for available credits. Credit masking can be performed prior to VC availability checking excluding those output VCs that even if they are free do not have available credits to host a new incoming flit.

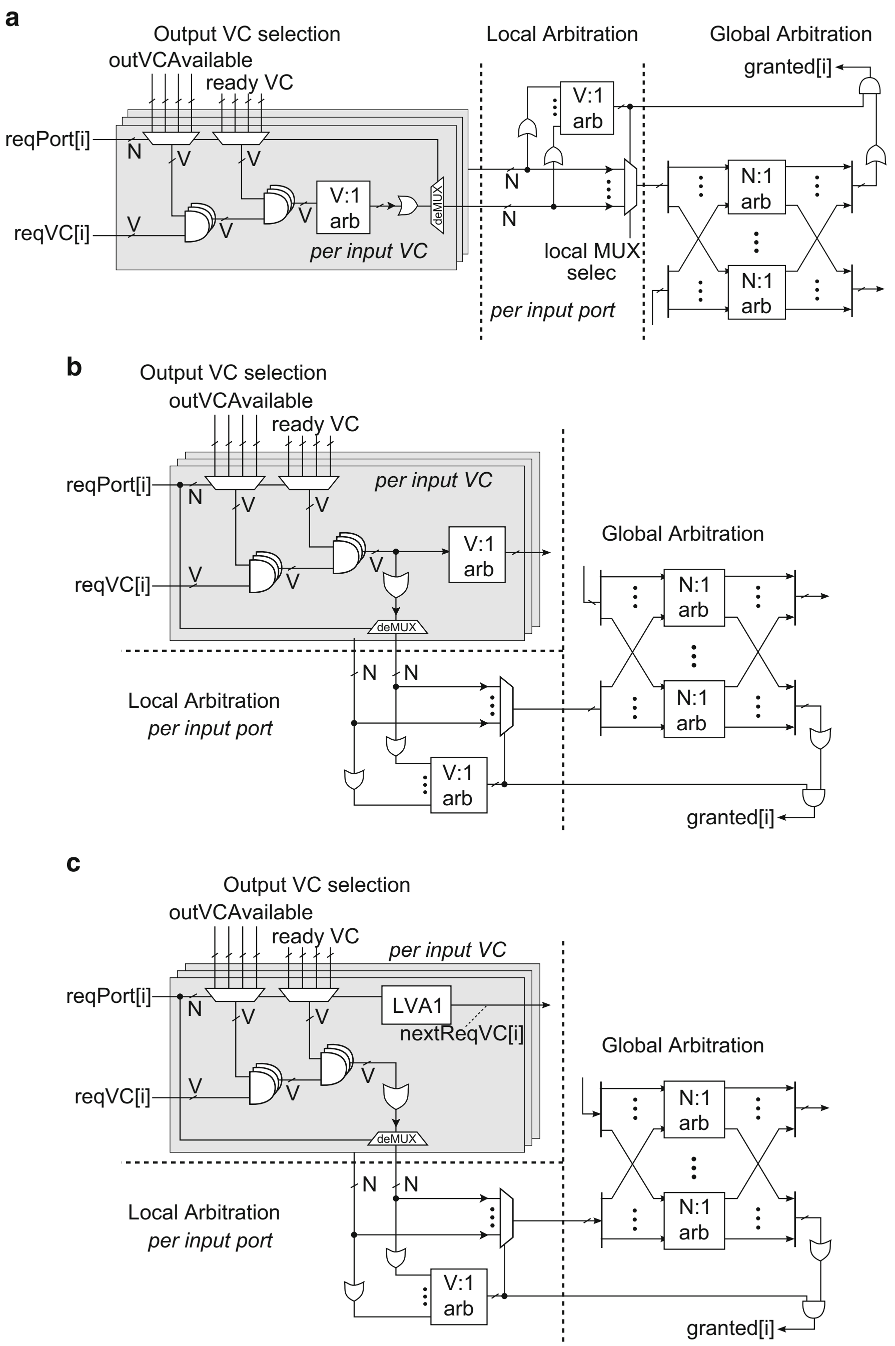
The relative placement of VA1 with respect to switch allocation (SA) for the implementation of combined allocation is discussed in the following paragraphs.

8.3.1 Combined Allocation with VA1 in Series to SA

The organization of the combined allocator that performs VA1 for each input VC in series to the SA stage is shown in Fig. 8.3a. Each input VC is searching in VA1 for an output VC that is available and has at least one credit available. The input VCs that managed to find an output VC that fulfills both criteria are eligible to participate in SA1. SA1 accepts also the requests of those input VCs that have allocated an output VC in the past and secured the existence of at least one available credit. The winning input VC as in traditional SA passes its input request via a multiplexer to the output stage of switch allocation (SA2). The grants of the per-output arbiters are gathered per input and the existence of at least one grant for each input is computed via an OR gate. Then, each input taking into account the grants of SA1 knows which input VC (if any) has won in switch allocation. The grants per input VC have a dual meaning: First it allows the flit to move to the output, and, at the same time, if it is a head flit that has not allocated an output VC, to receive the output VC that has been selected during VA1.

8.3.2 Combined Allocation with VA1 in Parallel to SA

In the previous paragraph before letting SA to begin, each input VC should select an output VC to request, selecting one from the pool of available ones that had at least one credit as well. The output VC that each input VC has selected after arbitration (VA1) is not used before the end of SA2. Therefore, there is no reason for SA1 and SA2 to wait for VA1 to finish but can execute in parallel to it, as illustrated in Fig. 8.3b. The only information that each input VC needs in order to participate in SA1 is that there is at least one output VC at the selected output port that is available and with credits. There is no need the input VC to know which output VC exactly fulfills the two criteria of availability and readiness. This will be found during VA1 that executes in parallel to SA1 and SA2. Then, once SA2 decided which VC won per input, if this winning input VC has not allocated yet an output VC, it gets allocated in the same cycle to the output VC that was selected by VA1 in parallel. This is needed only for the head flits that acquire an output VC at the same time they win in SA. The rest flits keep the output VC that has been allocated to them in previous cycle and actually participate only in the SA part of combined allocation. In Kumar et al. (2007), a similar approach has been followed that does not include



(c)

Fig. 8.3 The possible organizations for a combined allocator using (**a**) a VA1 module in series to SA, (**b**) a VA1 module in parallel to SA, and (**c**) running SA using the output VC requests prepared beforehand using LVA1

VA1 but gives to the winning head flit the first available output VC. If routing computation involves the selection of certain output VCs this last approach cannot be applied.

8.3.3 Combined Allocation with Lookahead VA1

LVA1 in combined allocation is a degenerate case of the organization presented in the previous paragraph that enables VA1 to evolve in parallel to SA. With LVA1 each input VC knows already the requested output VC. Therefore, it can participate in SA1 after checking the availability of the corresponding output VC and its readiness in terms of credits. If the pre-selected output VC satisfies the two needed criteria the input VC moves in SA1. In the case that it wins SA2, it gets allocated to the pre-selected output VC. Of course, in parallel to SA, the input VC should execute LVA1 for the next router. The input VCs that own already an output VC participate only in the SA stage of the combined allocator.

Combined allocation removes the need for the VA2 stage of VC allocation. When applied using the presented techniques that let VA1 execute in parallel to the SA, it offers high-speed router implementations where the incoming packets are allowed to change VC in flight while the critical path of allocation includes only the SA1 and SA2 steps and none of the VA1 or VA2.

8.4 Speculative Switch Allocation

Following another school of thought the serial dependency that exists between VA and SA can be removed by performing SA speculatively (Peh and Dally 2001; Mullins et al. 2004). Under speculative switch allocation a head flit is allowed to try to get access to an output port via switch allocation without having allocated first an output VC. VA and SA run in parallel for the head flit of a packet, and depending on the outcome of each module four cases can occur:

- A packet fails in both VC and switch allocation: The packet tries again in the next cycle for both allocations.
- A packet is granted by the VC allocator and fails in switch allocation: Although the head flit lost in switch allocation, it keeps the assignment made by the VC allocator and retries for switch allocation in the next cycle (non speculatively this time).
- A packet fails in VC allocation but is granted in switch allocation: This is the case of miss-speculation and is the worst scenario that can occur. Although, a head flit has allocated an output is obliged to not to use it in this cycle, since it does not own yet an output VC.

- A packet gets granted by both allocators: The head flit received all the necessary resources and can leave the input VC buffer. This is the best case of speculative switch allocation and is expected to happen often under low traffic conditions, where the output resources (VCs and ports) are free most of the time and contention probability is very low.

In order to reduce the probability of miss-speculation, the speculative VC-based router involves two separate switch allocators. The first switch allocator receives only the speculative requests, i.e., those coming from head flits that have not allocated an output VC. The second switch allocator receives the remaining requests including the requests from the head flits that have been assigned to an output VC and the requests from the body and tail flits that always participate in SA non-speculatively (always a preceding head flit has allocated for them an output VC). If we give higher priority to the grants of the non-speculative switch allocator, we guarantee that the winning flit can always move to the selected output. To satisfy this feature the grants of the speculative switch allocator should be rejected, when there is a grant for the same input–output pair from the non-speculative switch allocator. The organization of the allocation logic in the case of a speculative VC-based router, including the two switch allocators and the VC allocation logic that runs in parallel, is illustrated in Fig. 8.4.

When an input VC has already allocated an output VC it participates only in the non-speculative SA. On the contrary, the input VCs that do not have allocated yet an output VC participate in VA and in the speculative SA. During VA, they try to match to an output VC, but they should guarantee that the requested output VC selected at VA1 not only is available (as needed in baseline VA), but it is also ready, i.e., it

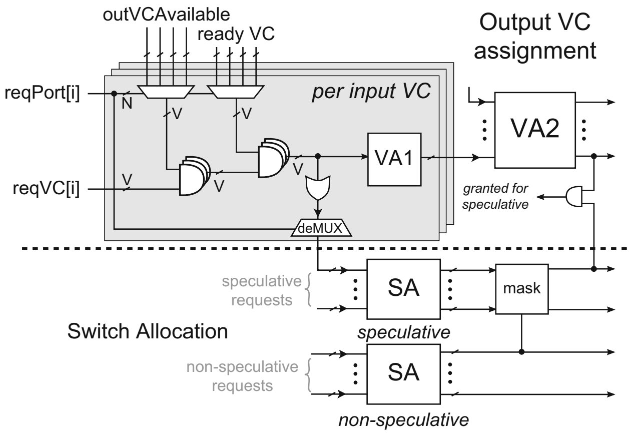


Fig. 8.4 The organization of the speculative VC-based router. Each input VC generates in parallel requests to the VC allocator and to the speculative SA for the head flits and to the non-speculative SA for the rest flits and the head flits that managed to allocate an output VC in a previous cycle. The grants that return from the three allocation units are handled according to the four scenarios described in the beginning of this section

has enough credits (similar to combined allocation); else a grant from the SA will be useless. At the same time, during speculative SA, they compete with other input VCs that do not have allocated an output VC, after checking that there is at least one available output VC and with credits. Although they don't know which specific output VC to check for availability and readiness (they will know only when VA1 that runs in parallel, finishes), checking the existence of at least one output VC that satisfies both criteria is enough, since the same two-criteria qualify also the candidates of VA1.

8.4.1 Handling the Speculative and the Non-speculative Grants

Once the availability and readiness of the selected output VCs is checked for all flits both switch allocators run in parallel. A grant from the speculative SA is considered valid only when there is no other grant from the non-speculative SA referring to the same input and output port. All invalid speculative grants are masked away and the rest are kept and included in the final input-output SA match. Figure 8.5a depicts the switch allocator and its two subcomponents used in the case of speculative VC-based router including also the masking logic of the speculative grants.

Masking of the invalid speculative grants is done in two steps. In the first step, we need to identify the input-output pairs that have been matched by the non-speculative switch allocator that produces always valid matches. Two bit vectors, named *grantInput* and *grantOutput* are computed in parallel; $\mathit{grantInput}(i) = 1$ when the ith input has received a grant from the non speculative SA, and $\mathit{grantOutput}(j) = 1$ when the jth output has been granted from the non-speculative SA. Then, using the *grantInput* and the *grantOutput* bit vectors a 2D matrix of bits is computed called the *free* matrix; $\mathit{free}(i, j) = 1$ when input i and output j have not received a grant from the non-speculative SA. Therefore, the input–output pair i, j is a candidate for accepting a grant from the speculative switch allocator. Using the *free* matrix we can derive the final valid grants of the speculative SA as follows: $\mathit{validSpecGrants}(i, j) = \mathit{free}(i, j) \wedge \mathit{initialSpecGrants}(i, j)$.

Becker and Dally in (2009) observed that instead of waiting the non-speculative SA to produce grants and mask afterwards the invalid grants of the non-speculative SA, we can achieve the same result, if pessimistically, mask the speculative grants with the corresponding non-speculative requests. The organization of the SA with a pessimistic masking of the speculative grants is shown in Fig. 8.5b. Using this approach, we are allowed to compute the row- and column-wise reduction trees for computing the free bits, in parallel with allocation, removing them from the critical path. This pessimism makes sense at low network traffic, where a non-speculative request is likely to be granted due to the low contention in the network. As the network traffic increases more and more speculative grants are unnecessarily masked by non-speculative requests that fail to produce a grant.

The grants returning from the VC allocator, the speculative and the non-speculative switch allocator should be treated accordingly so that no output with

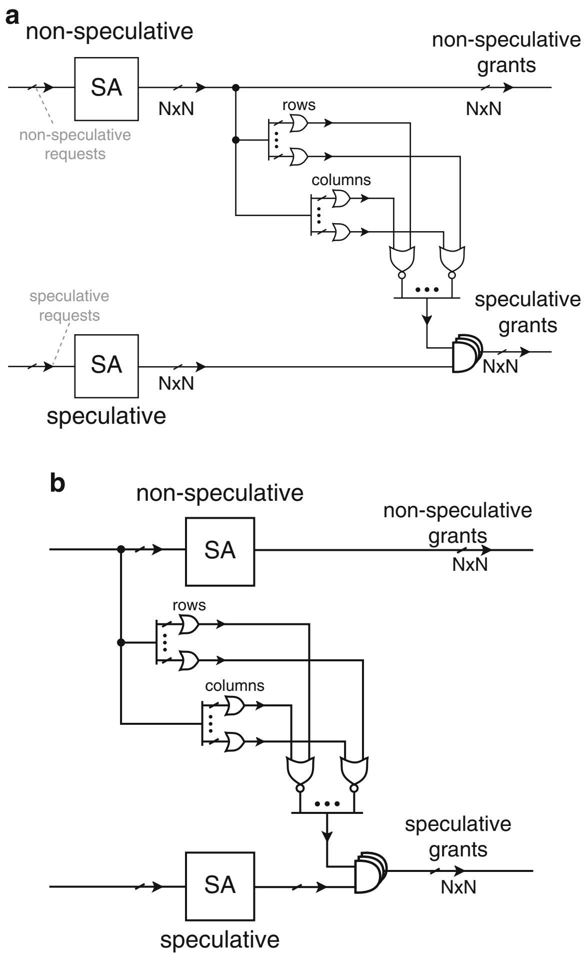


Fig. 8.5 The masking of the invalid grants of a speculative SA using (**a**) either the grants produced by the non-speculative SA or (**b**) using the requests of the non-speculative SA that masks pessimistically a speculative grant even if the request that caused the masking was not granted after all

a valid assignment remains idle. The four possible scenarios have been discussed at the beginning of this section. A non-speculative grant always means that the corresponding flit of the packet (the flit can be of any kind) has already allocated an output VC and can leave the input VC buffer and move to the selected output in this cycle. On the contrary, a speculative grant from the SA is not a guarantee for success. The corresponding head flit that generated the speculative request should match to an output VC in the same cycle. If not, the speculative grant is lost and the head flit should retry in the next cycle. If the VA match is successful, the head flit allocates at once two resources, e.g., an output VC and a time slot for an output port, and leaves the input VC buffer. In the case that a head flit receives a grant only from the VA and not the speculative SA, it stores the matched output VC and in the next cycle it participates in the non-speculative SA. This last option actually differentiates speculative switch allocation from combined allocation.

8.5 VC-Based Routers with Input Speedup

Allocation efficiency can be improved by letting more than one VCs per input to reach independently the crossbar thus allowing the switch to offer speedup at the input side.[1] In the baseline organization of a VC-based router discussed so far, there is only one input to the crossbar per input port, and thus only one virtual channel in an input port can transmit a flit in a cycle. Allowing more than one VCs per input to reach the crossbar directly, means that the inputs of the crossbar are multiplied with the number of transmit ports per input. The input VCs can be separated in m groups where each group receives a dedicated input port of the crossbar. This organization is depicted in Fig. 8.6. In this case the one per-input multiplexer that switched V input VCs is replaced by m smaller multiplexers that select between V/m input VCs. Equivalently, the input port that used only one input of crossbar, now sees m inputs available. The output multiplexers of the router grow from N inputs to $m \times N$ inputs, since now every input is allowed to send flits from m different VCs assuming that they move to different outputs.

Virtual channel allocation does not need to change, unless it is simplified to treat the group input VCs as virtual networks and thus limiting the set of output VCs that each input VC can allocate. On the contrary, switch allocation should change in order to support the input speedup of the VC-based routers. In this case, SA1 that operated using a $V : 1$ arbiter per input, now should support m arbiters that run in parallel each one receiving V/m requests. At the output side the number of SA2 arbiters do not change and remains equal to one per output. However, since now each input may receive the flits from the $m \times N$ different inputs of the crossbar, each output arbiter should handle $m \times N$ requests. In overall, the delay of the SA is not expected to change since the arbiter's delay depends logarithmically on the

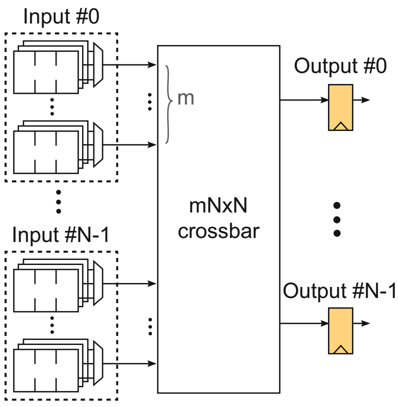


Fig. 8.6 Input speedup increases the input ports of the crossbar thus allowing multiple input VCs of the same input to allocate an output port and move to their selected output. This organization partially alleviates the problems inherent in separable switch allocation thus increasing the overall throughput of the router

[1]Input speedup means that although the rate of incoming flits is one flit per cycle for each input the rate of the flits that can leave each input towards the crossbar is larger than one.

number of inputs and thus the increase of the logic depth of the SA2 arbiters is balanced by the decrease of the logic depth of the SA1 arbiters.

Input speedup is a useful technique for handling the possible inefficiencies of separable switch allocation (Dally and Towles 2004; Rao et al. 2014). The only drawback remains the increased wiring between the input VC buffers and the output multiplexers of the crossbar that may limit the effectiveness of input speedup when the network operates using very wide flits. At the same time, input speedup requires changing also the flow-control mechanism, since in this case, multiple credits update need to return in each cycle; one from each input VC that was selected by the switch allocator.

Applying input speedup to its maximum extent and by assuming that each input VC represents an isolated virtual network, allows us to design VC-based routers using simple wormhole switches in parallel. Once each VC is a separate virtual network, VA is not required, since no packet will ever change the VC that it already uses. Also, since maximum speedup is enabled, the packets that belong to the same VC but come from different inputs can be switched together using a private wormhole router as shown in Fig. 8.7. For a router that supports V VCs, V wormhole routers are used in parallel that each one handles the flits of one VC from all inputs (Gilabert et al. 2010). Each wormhole router independently from the rest solves the output contention and prepares the flits that should depart from each output. At the output of the VC-based router the flit of one sub-router should be selected, effectively selecting which VC will use the output link in this cycle. This requires an additional arbiter and multiplexer that selects which VC should be served by each output. By keeping an independent flow control mechanism between the input VC buffers and the inputs of the wormhole router, as well as, the output of the VC-based router and the outputs of the wormhole routers, single, or multi-cycle/pipelined configurations can be derived. For example, if we assume that each

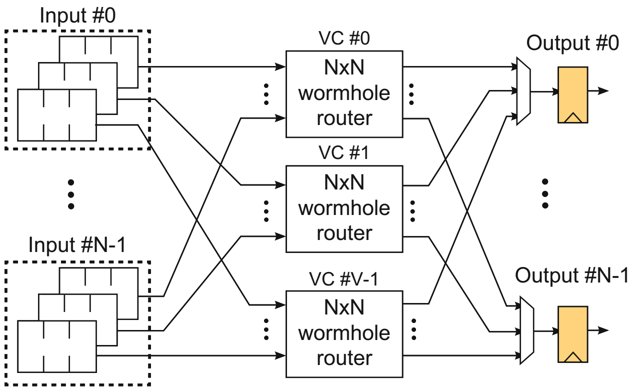


Fig. 8.7 Each VC of the router can be serviced by a private wormhole router. The results of all sub-routers are merged at the output of the VC-based router using another arbitration and multiplexing step that merges also the VC-based flow control of the output links

smaller wormhole router has elastic buffers at its inputs and output ports, then each flit of the VC-based routers will spend three cycles in the VC-based router: one for moving from the input VC to the input of the wormhole router; the next to switch to the selected output port of the sub-router, and the last one to move from the output of the sub-router to the output of the VC-based router.

Using the freedom offered by input speed up we can design hierarchical switching modules similar to the ones presented in Sect. 3.7 for wormhole routers that instead of elastic buffers would include Elastistores at each merging point and in the place of an arbiter and a multiplexer would contain a combined allocator with a parallel VA1 stage. The design of such hierarchical VC-based networks was proposed in ElastiNoC (Seitanidis et al. 2014b), and represent the first truly distributed VC-based NoC architectures.

8.6 Take-Away Points

The allocation steps in a VC-based router are responsible for the largest part of the router's delay. Speeding up the allocation process requires either the application of lookahead VA1 techniques that remove VA1 completely from the critical path, or the adoption of combined allocation that removes the need for VA2. On the contrary, instead of removing any of the required tasks, speculative allocation manages to parallelize the execution of VA and SA by employing more hardware modules for switch allocation. The employment of virtual networks or input speedup can further simplify the allocation process with the cost of additional multiplexing area.